T. HOEFLER

Extreme-Scale Graphs

Invited talk at Supercomputing Frontiers and Innovation 2019, Warsaw, Poland

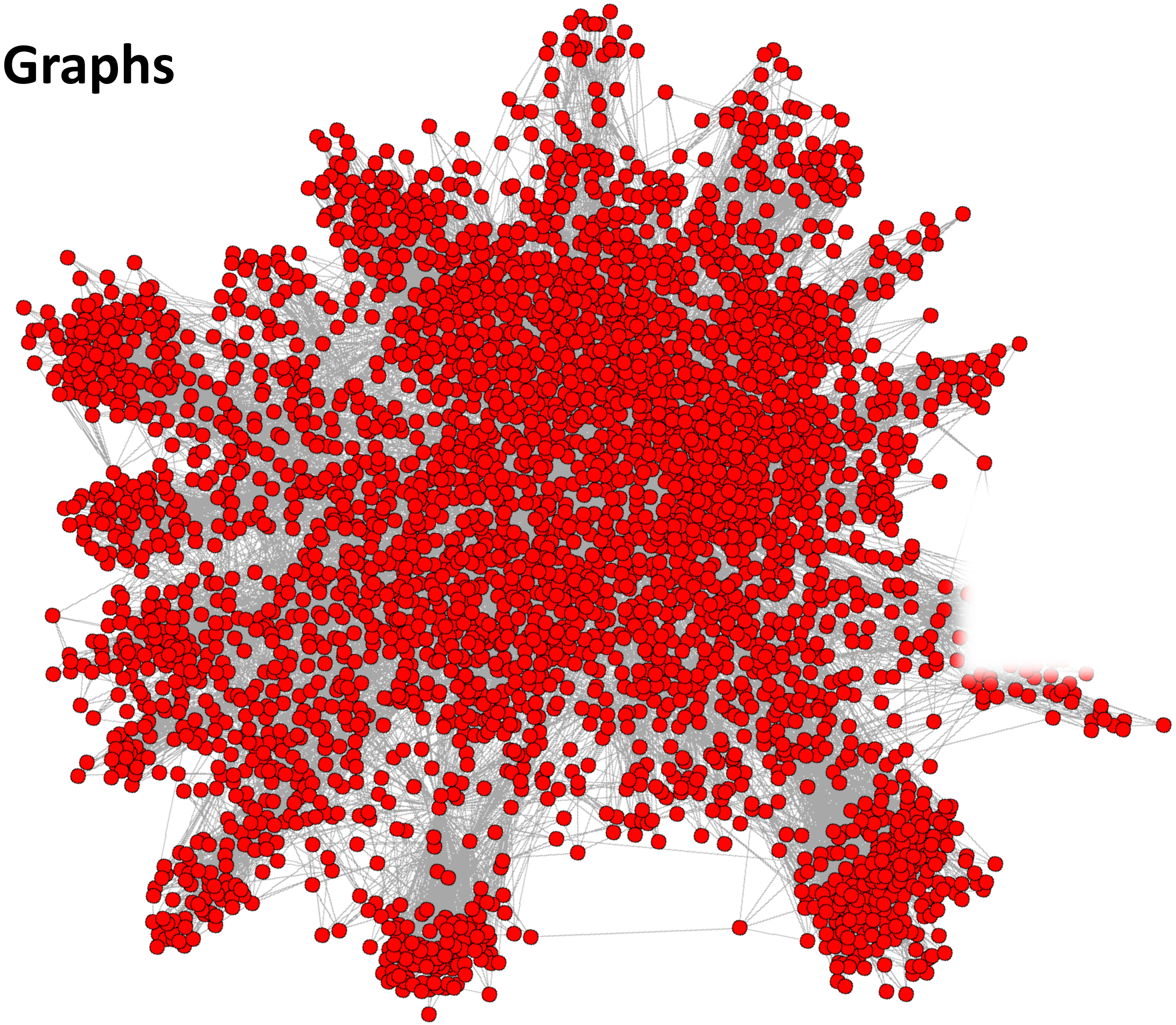
Special thanks to my student Maciej Besta

With contributions from Heng Lin, Xiaowei Zhu, Bowen Yu, Xiongchao Tang, Wei Xue, Wenguang Chen, Lufei Zhang, Xiaosong Ma, Xin Liu, Weimin Zheng, and Jingfang Xu and others at SPCL and Tsinghua University

SPCL

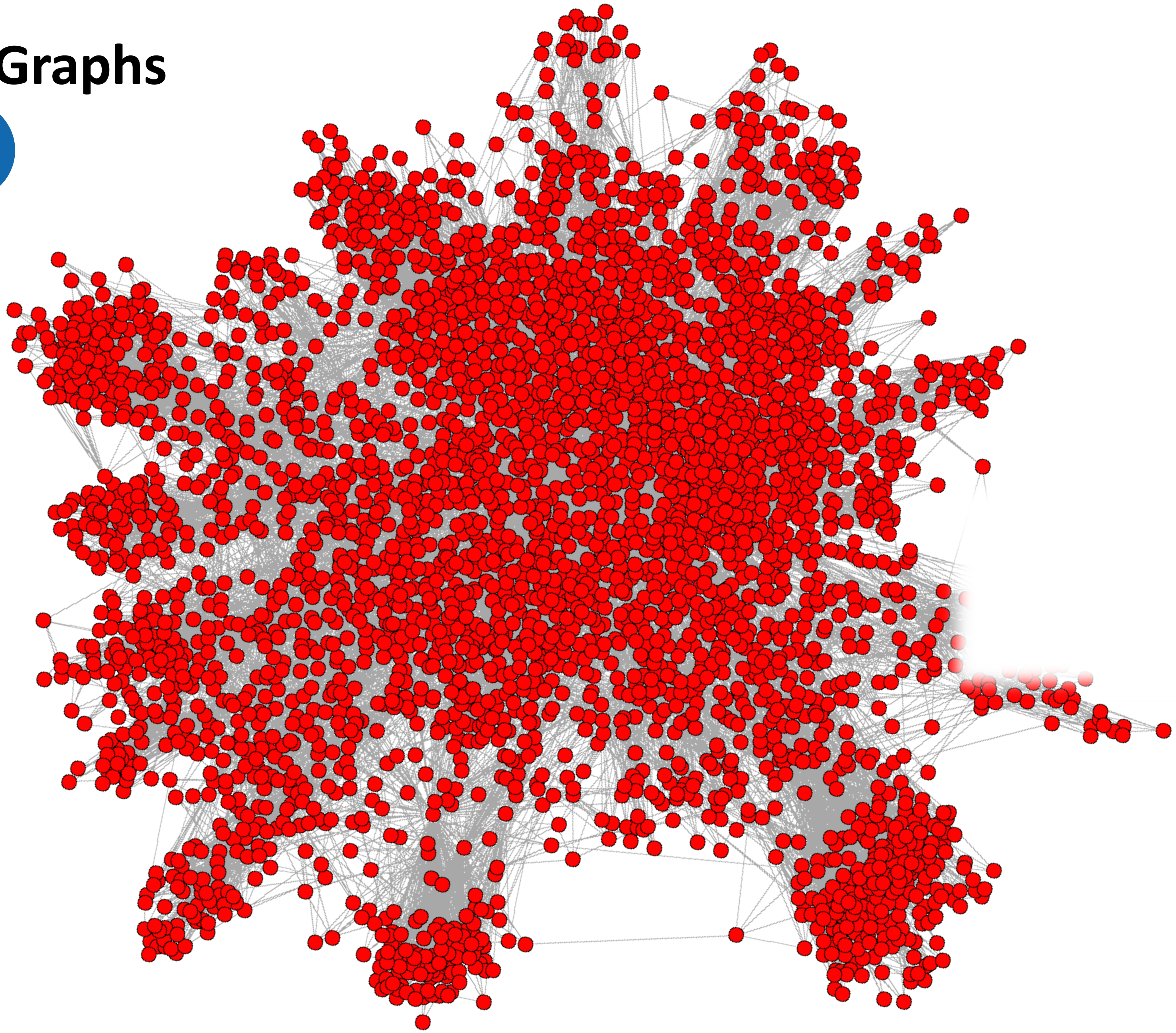
Extreme-Scale Graphs

Extreme-Scale Graphs



Extreme-Scale Graphs

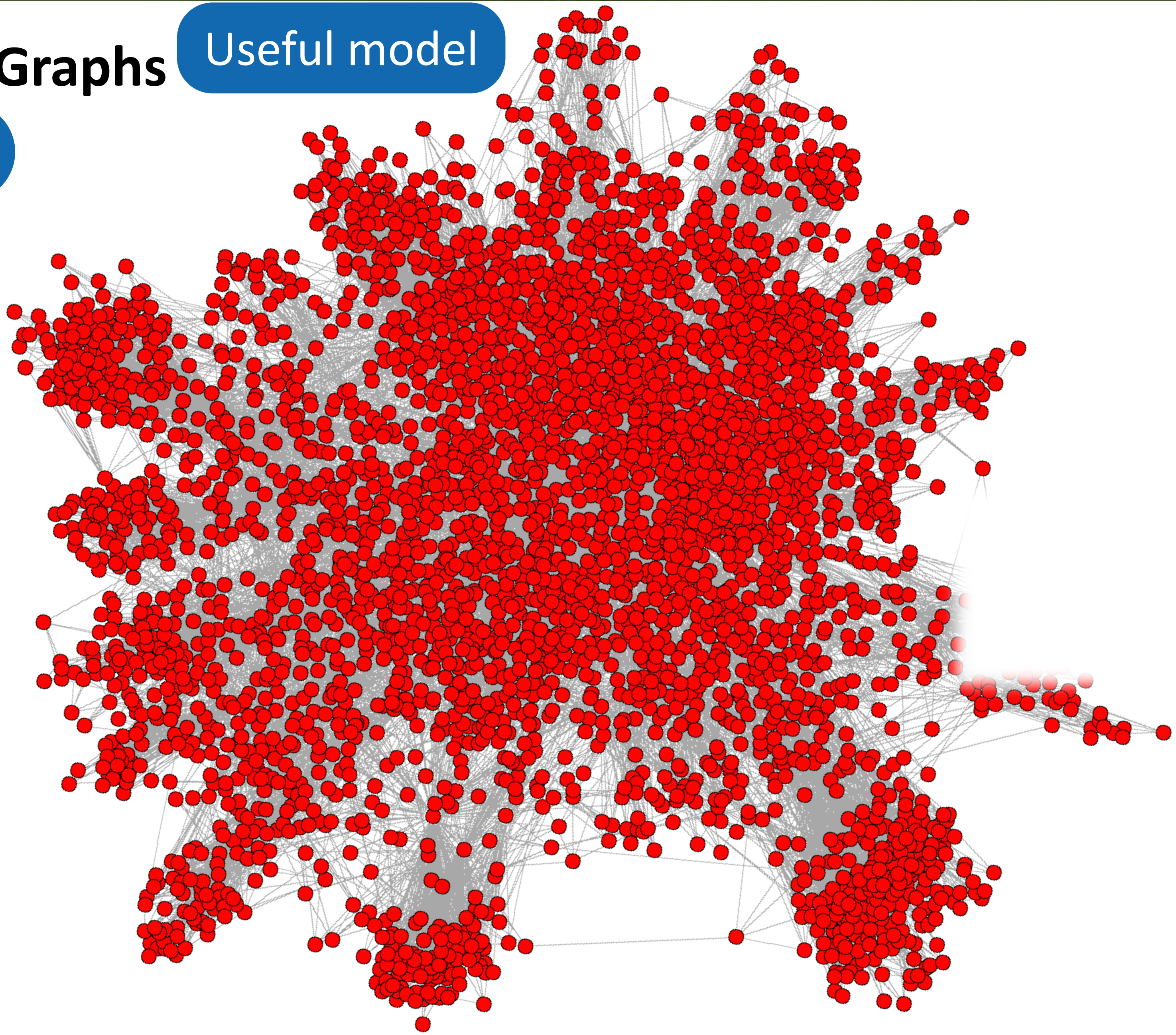
Why do we care?



Extreme-Scale Graphs

Useful model

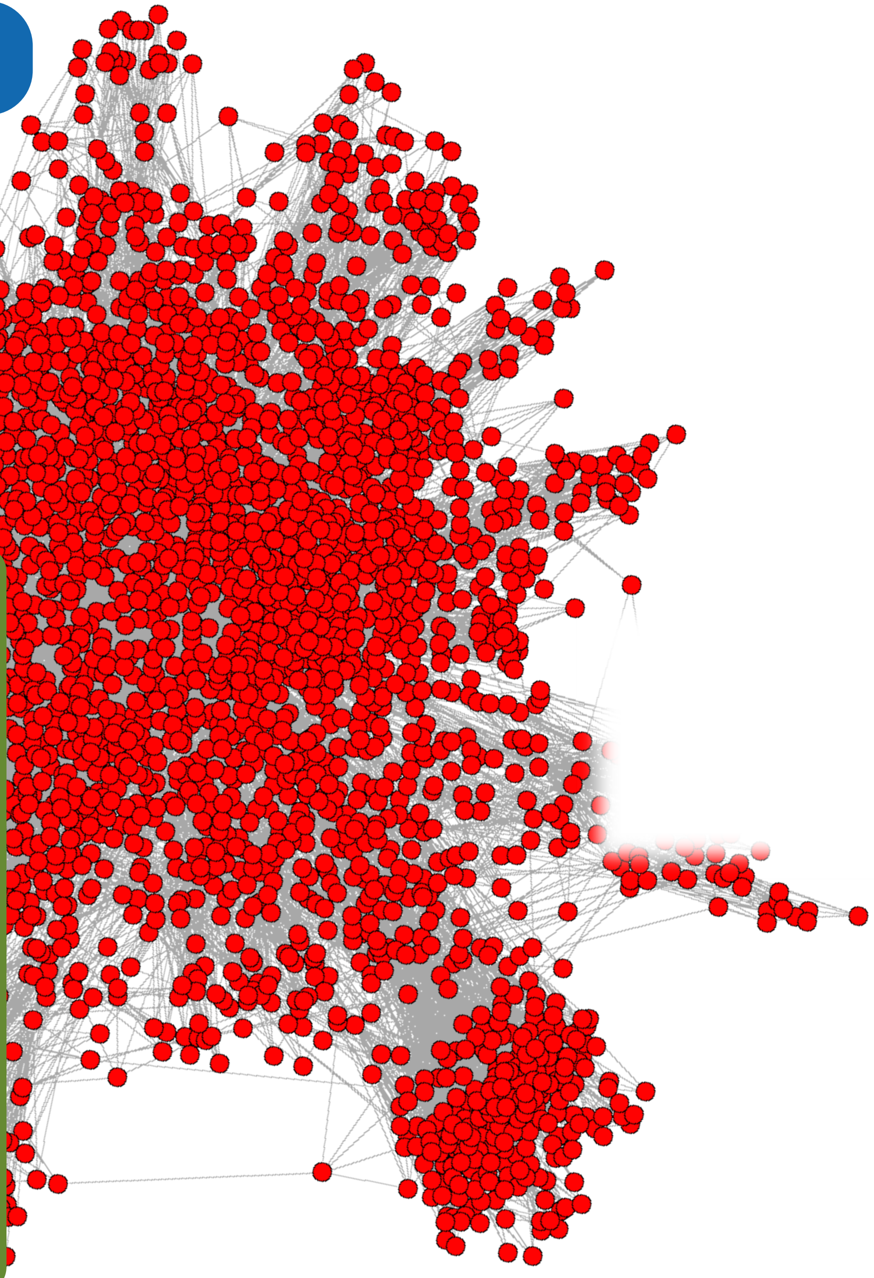
Why do we care?



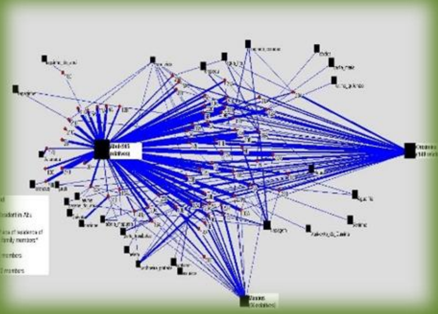
Extreme-Scale Graphs

Useful model

Why do we care?



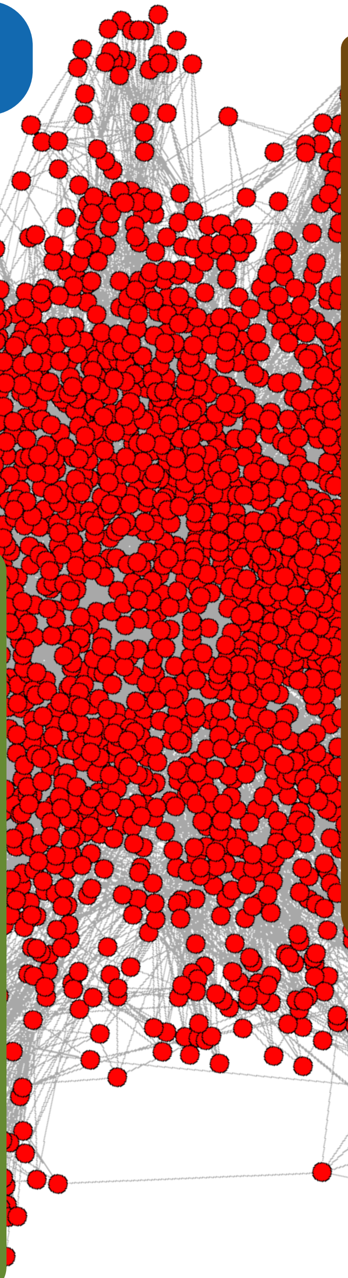
Social networks



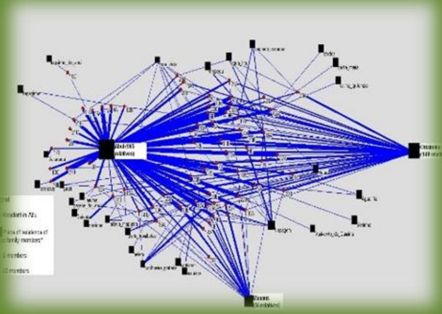
Extreme-Scale Graphs

Useful model

Why do we care?



Social networks



Engineering networks

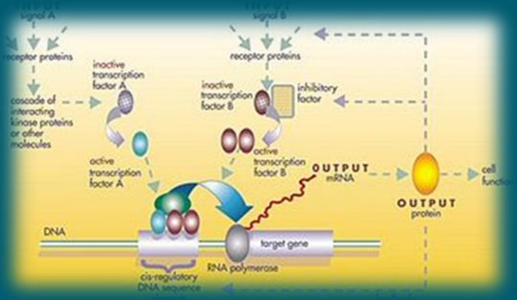
Extreme-Scale Graphs

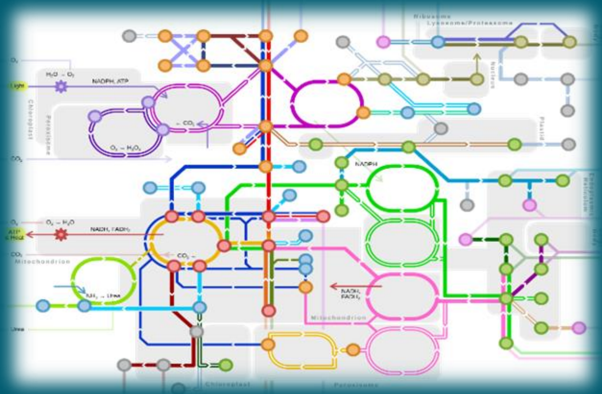
Useful model

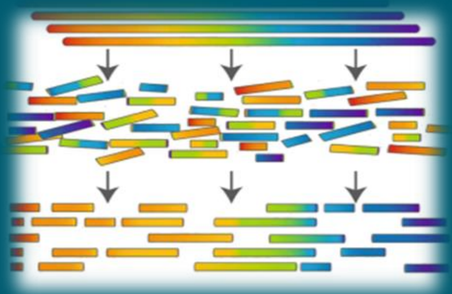
Why do we care?

Social networks

Biological networks







Engineering networks

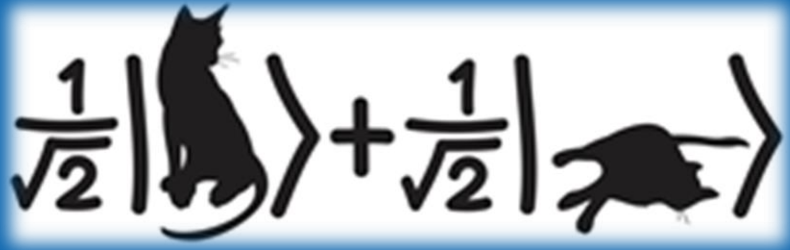
Extreme-Scale Graphs

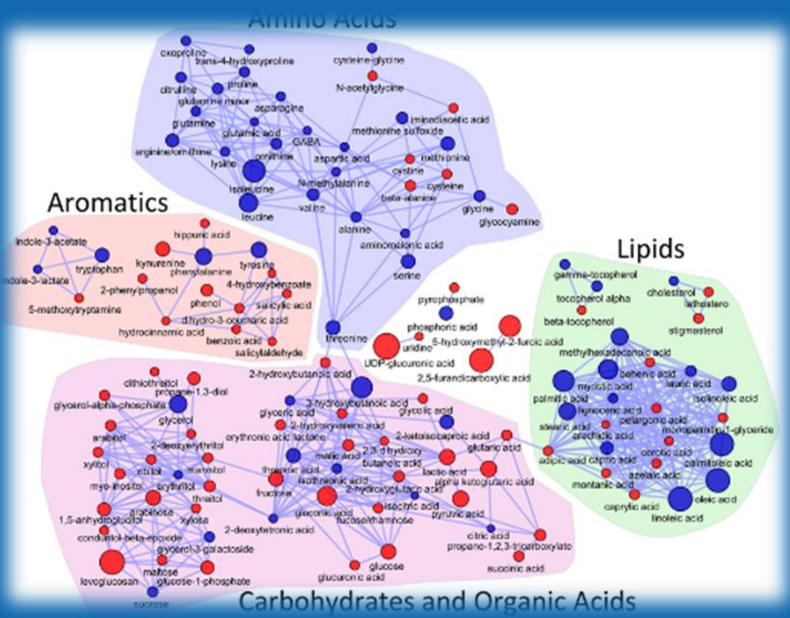
Useful model

Why do we care?

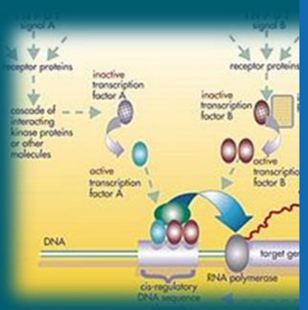
Engineering networks

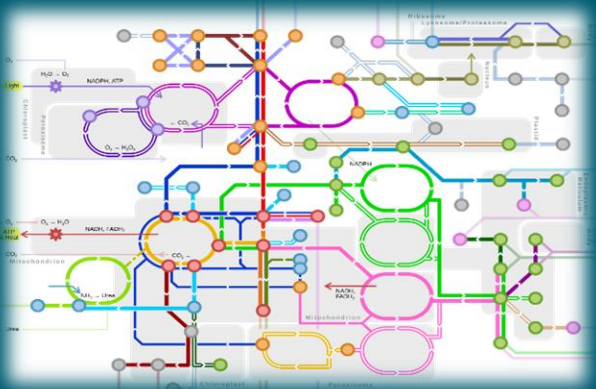
Physics, chemistry

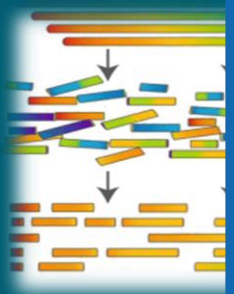




Biological networks





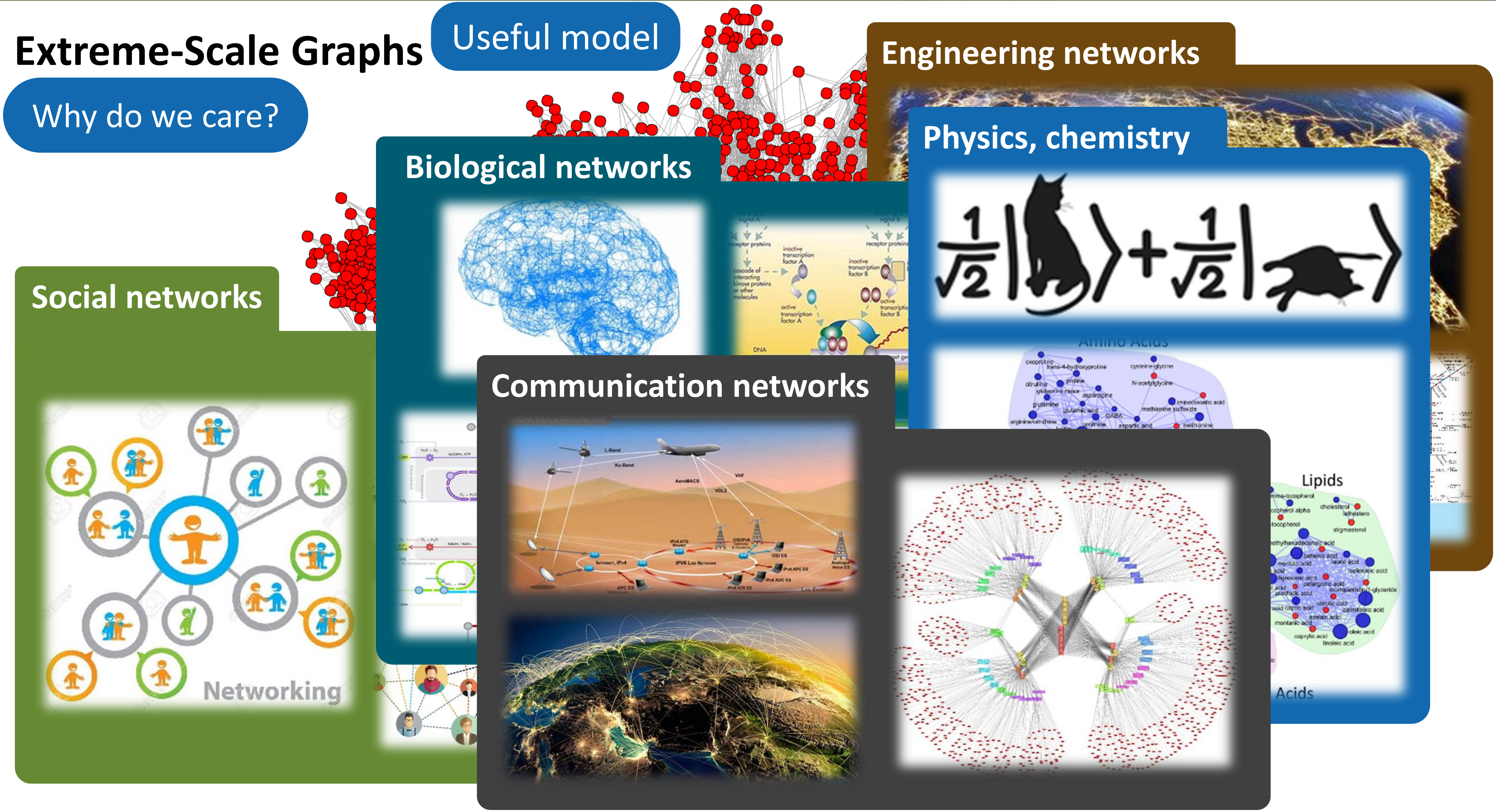


Social networks

Extreme-Scale Graphs

Useful model

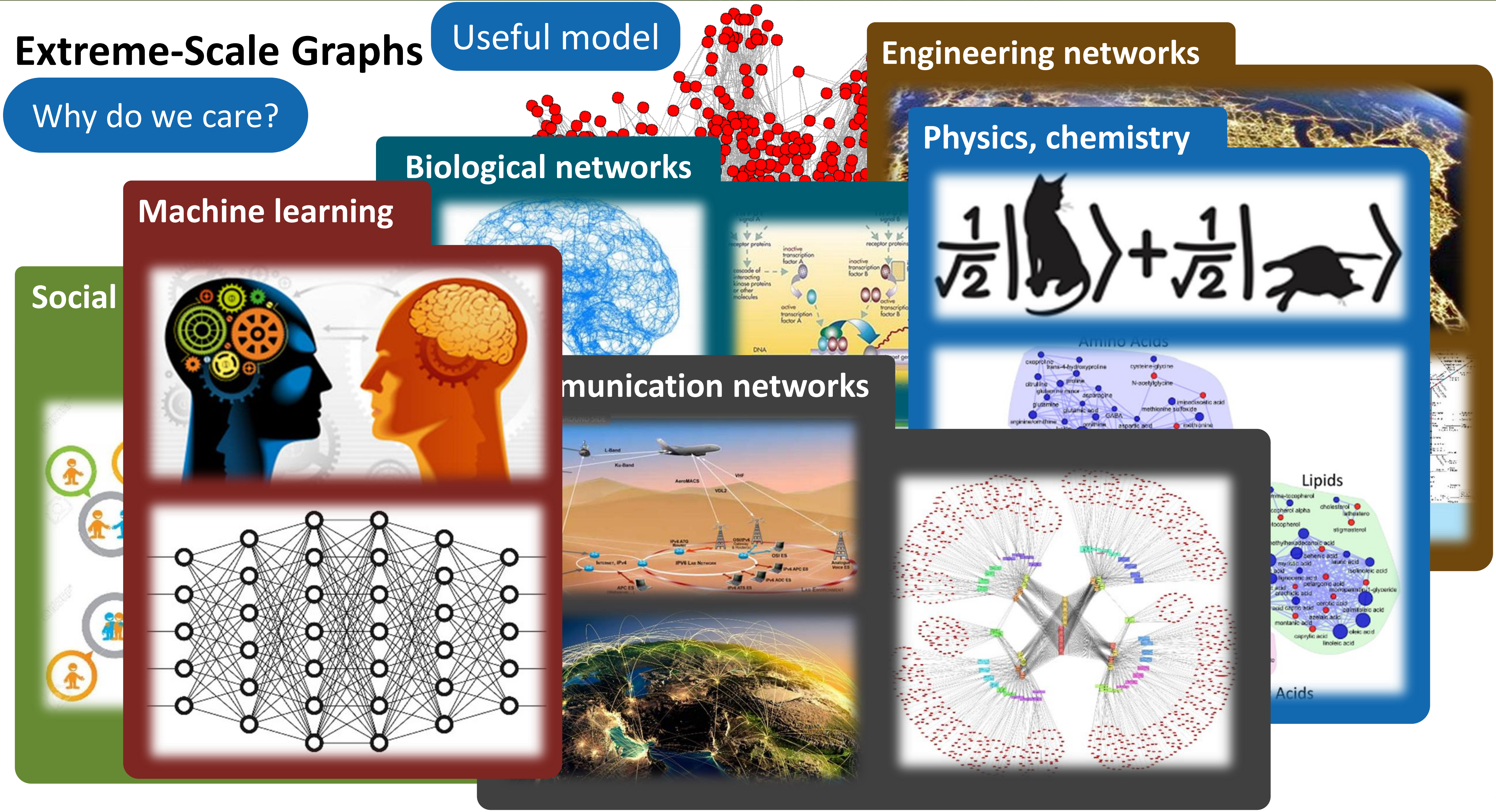
Why do we care?



Extreme-Scale Graphs

Why do we care?

Useful model



Extreme-Scale Graphs

Useful model

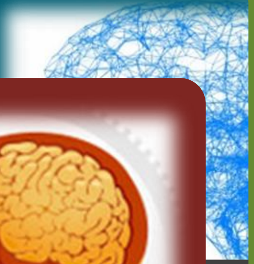
Why do we care?

Engineering networks

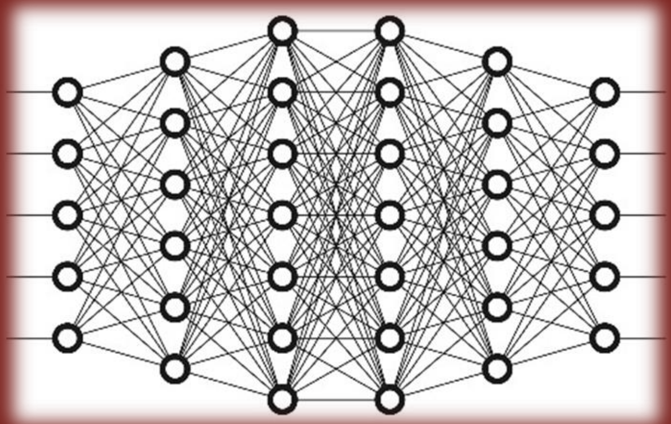
…even philosophy ☺

hysics, chemistry

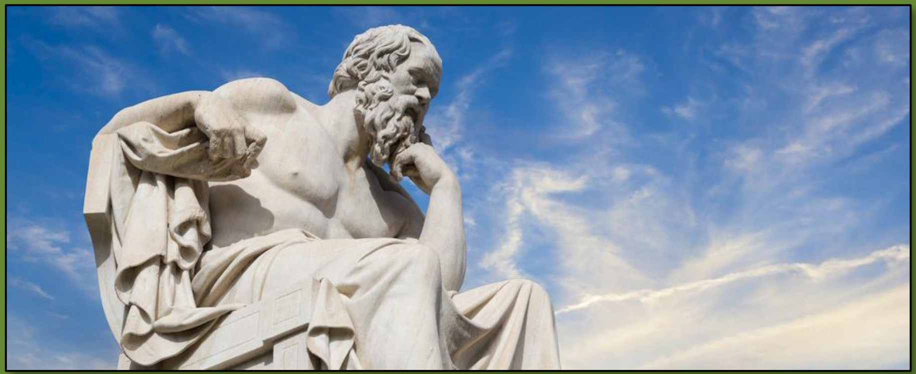
Biological ne

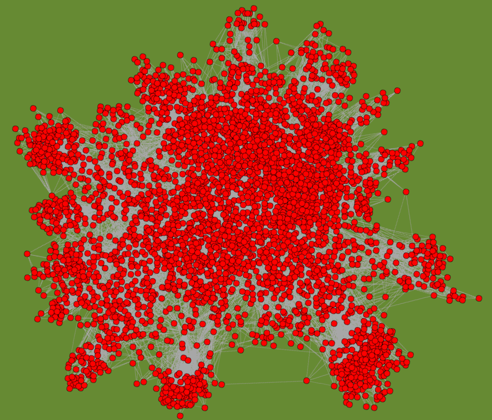


Machine learning



Social





FOSDEM 2016 / Schedule / Events / Developer rooms / Graph Processing / Modeling a Philosophical Inquiry: from MySQL to a graph database

Modeling a Philosophical Inquiry: from MySQL to a graph database

The short story of a long refactoring process

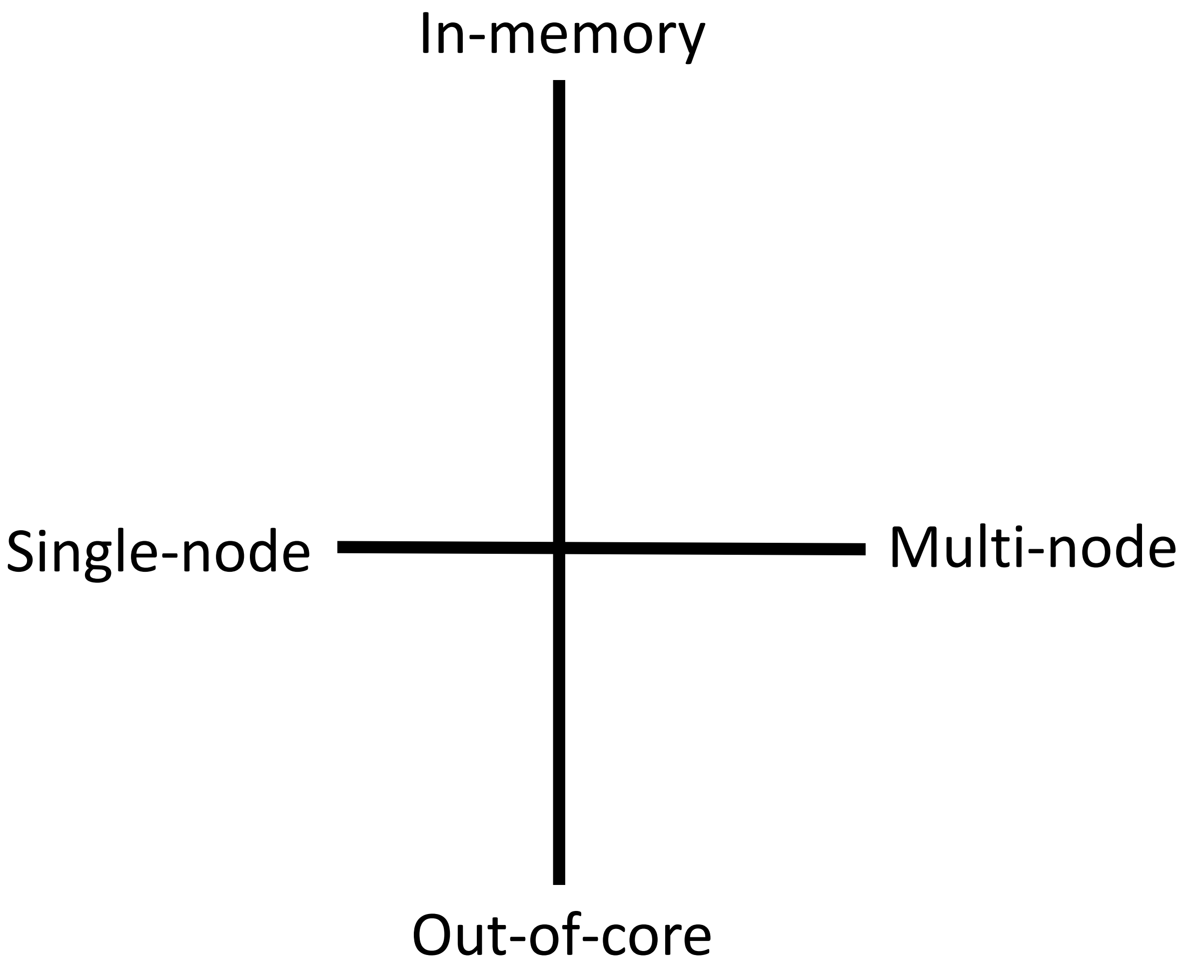
- Track: Graph Processing devroom
- Room: AW1.126
- Day: Saturday
- Start: 12:45
- End: 13:35

Bruno Latour wrote a book about philosophy (an inquiry into modes of existence). He decided that the paper book was no place for the numerous footnotes, documentation or glossary, instead giving access to all this information surrounding the book through a web application which would present itself as a reading companion. He also offered to the community of readers to submit their contributions to his inquiry by writing new documents to be added to the platform. The first version

Practice of Extreme-Scale Graph Processing

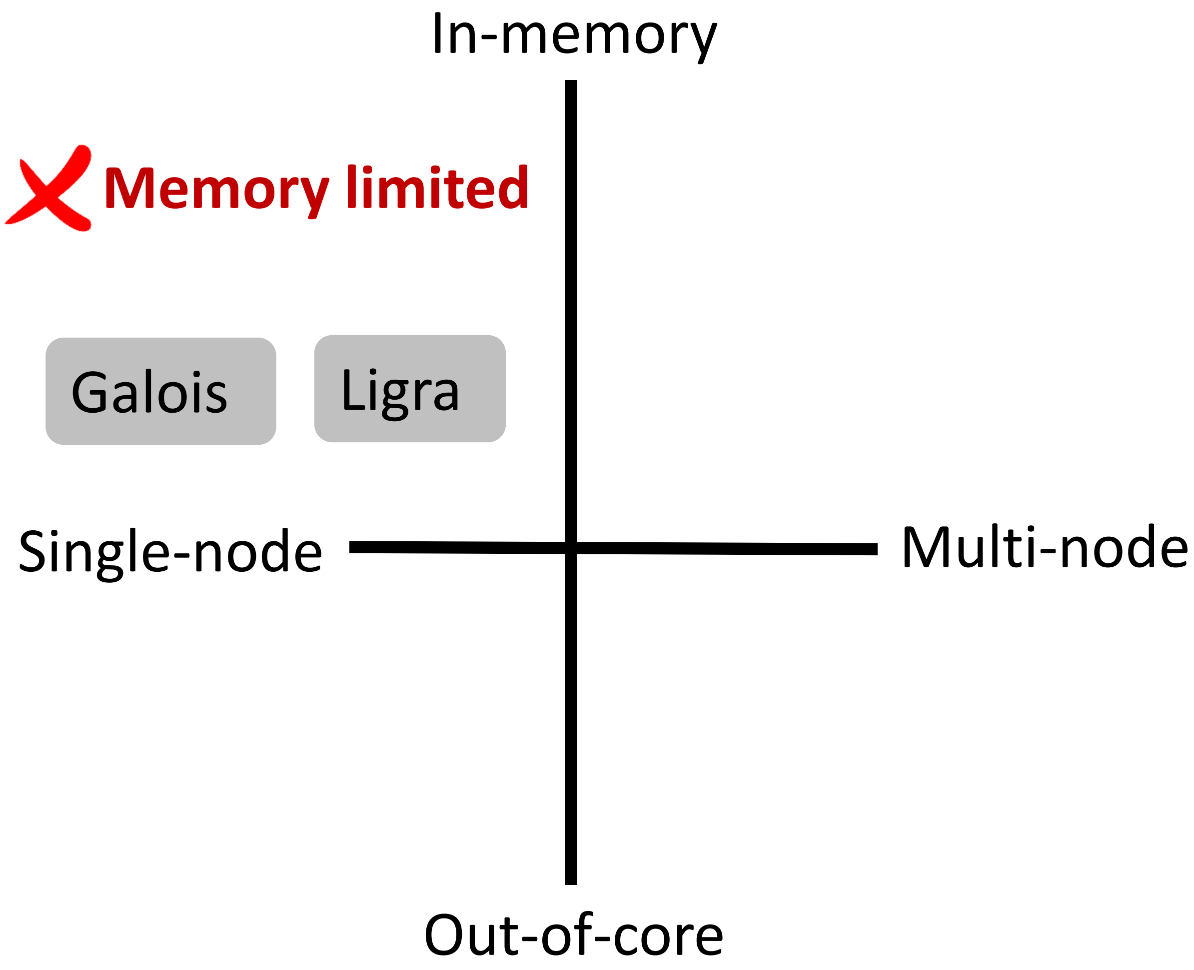
Heng Lin et al.: ShenTu: Processing Multi-Trillion Edge Graphs on Millions of Cores in Seconds, SC18, Gordon Bell Finalist

Practice of Extreme-Scale Graph Processing



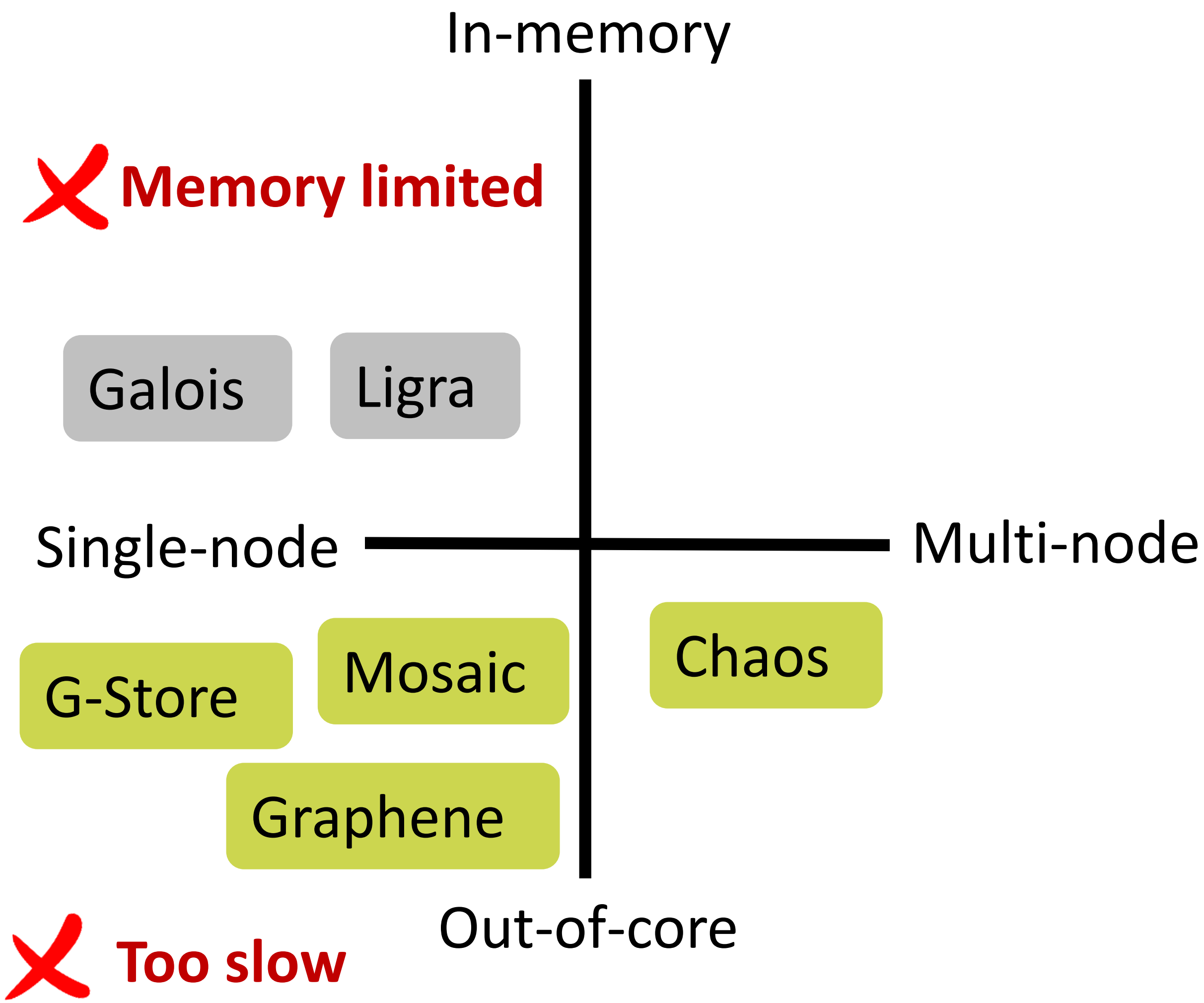
Heng Lin et al.: ShenTu: Processing Multi-Trillion Edge Graphs on Millions of Cores in Seconds, SC18, Gordon Bell Finalist

Practice of Extreme-Scale Graph Processing



Heng Lin et al.: ShenTu: Processing Multi-Trillion Edge Graphs on Millions of Cores in Seconds, SC18, Gordon Bell Finalist

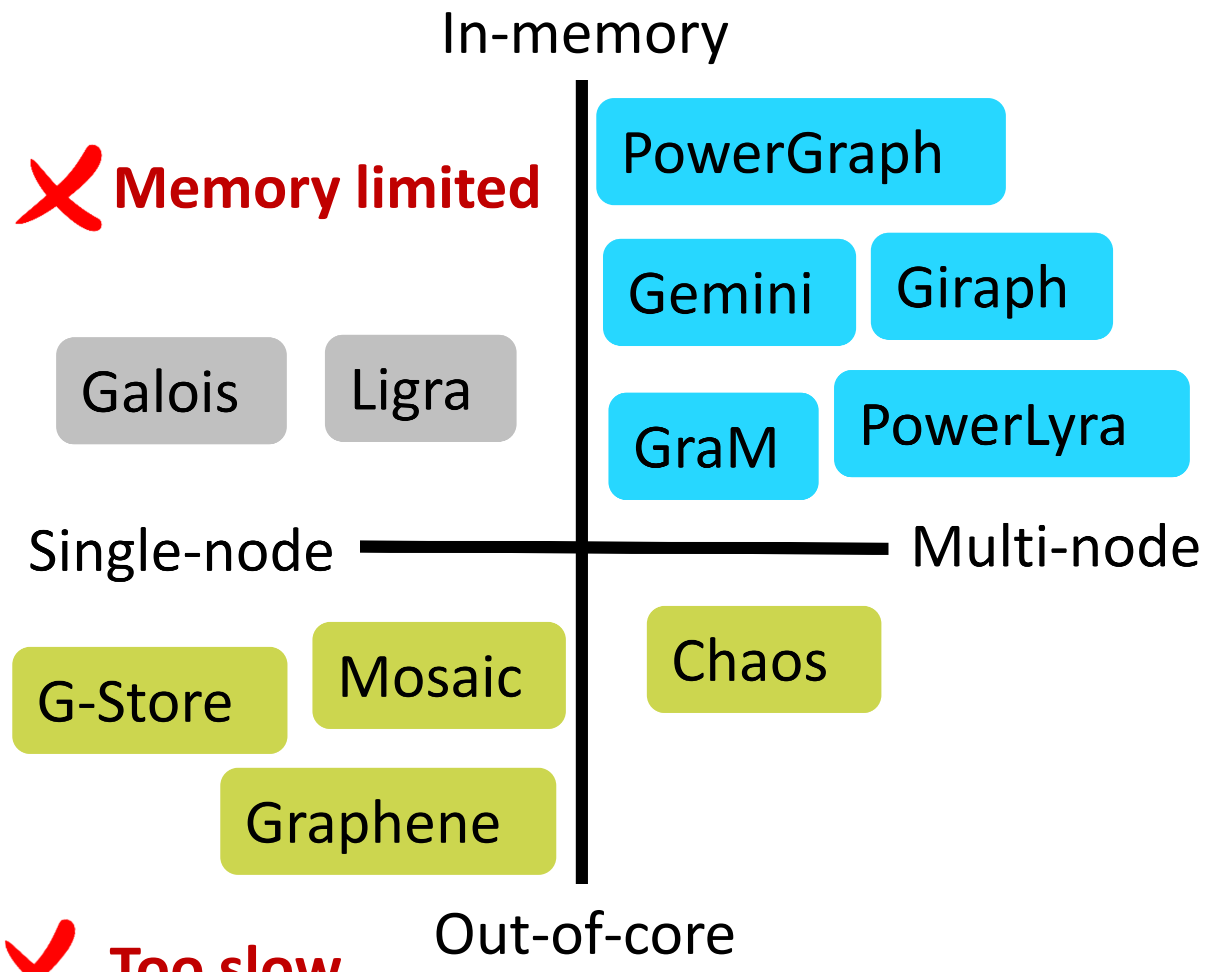
Practice of Extreme-Scale Graph Processing



(> 20 mins per PageRank iteration on 1-trillion-edge graph)

Heng Lin et al.: ShenTu: Processing Multi-Trillion Edge Graphs on Millions of Cores in Seconds, SC18, Gordon Bell Finalist

Practice of Extreme-Scale Graph Processing

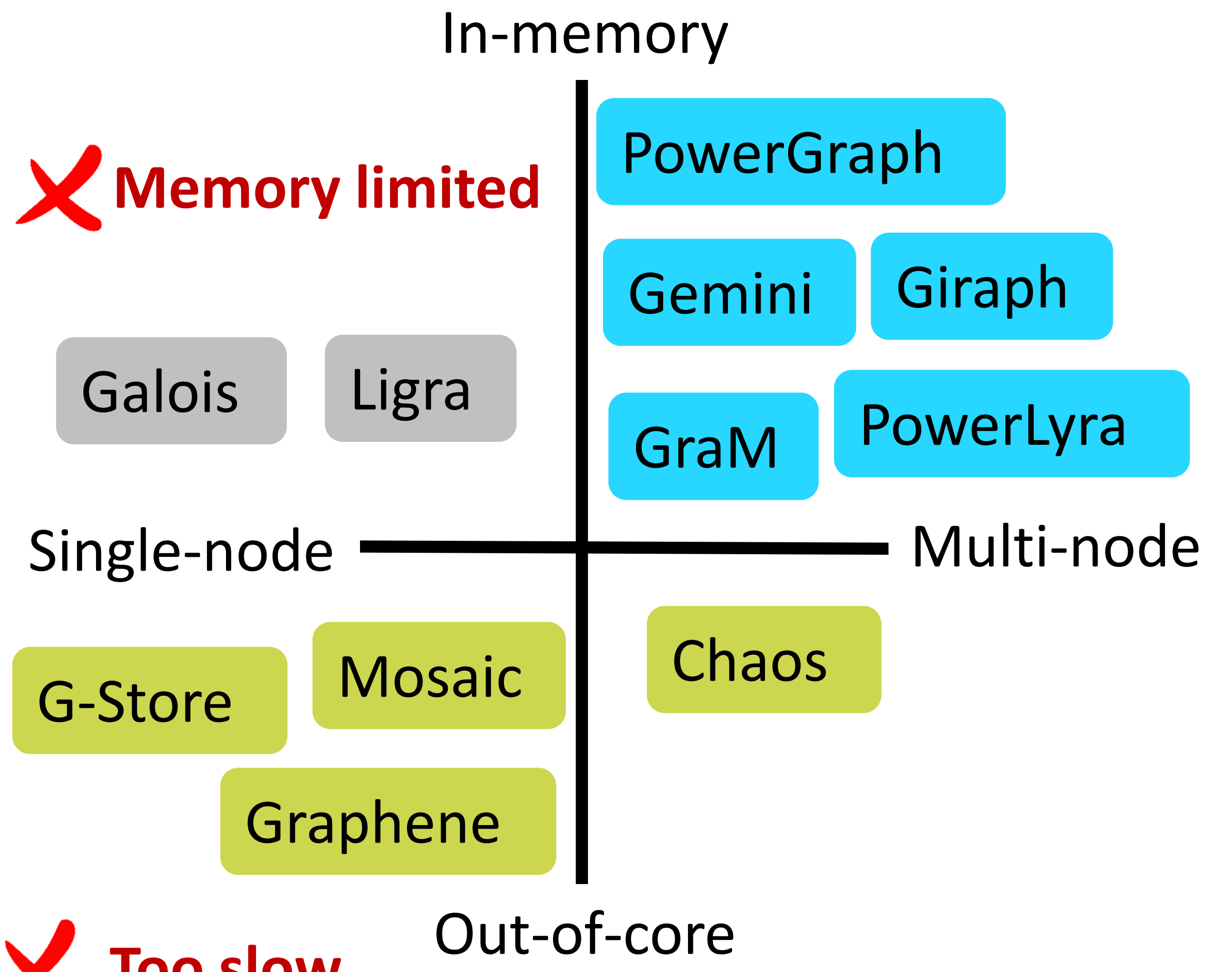


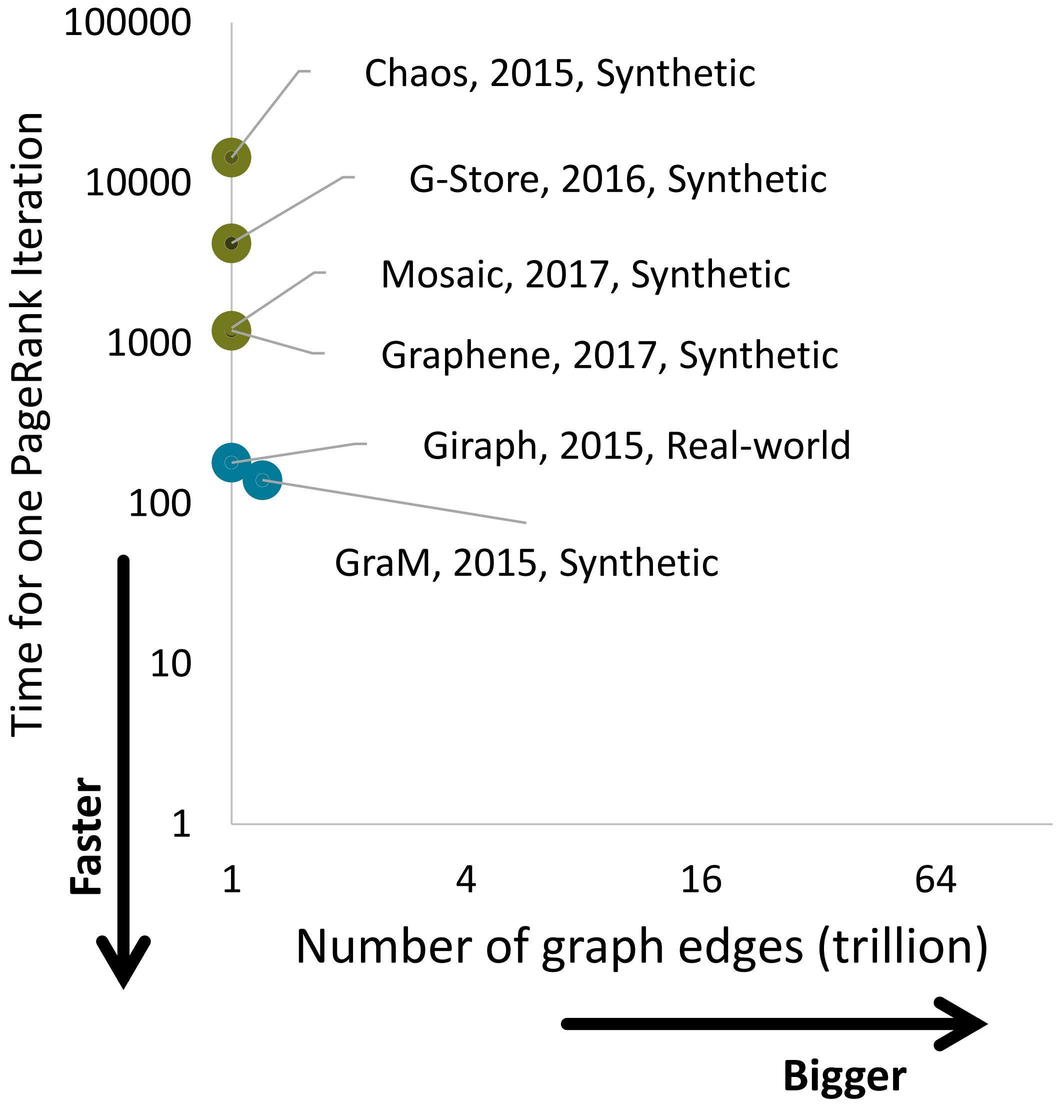
Too slow

(> 20 mins per PageRank iteration on 1-trillion-edge graph)

Heng Lin et al.: ShenTu: Processing Multi-Trillion Edge Graphs on Millions of Cores in Seconds, SC18, Gordon Bell Finalist

Practice of Extreme-Scale Graph Processing



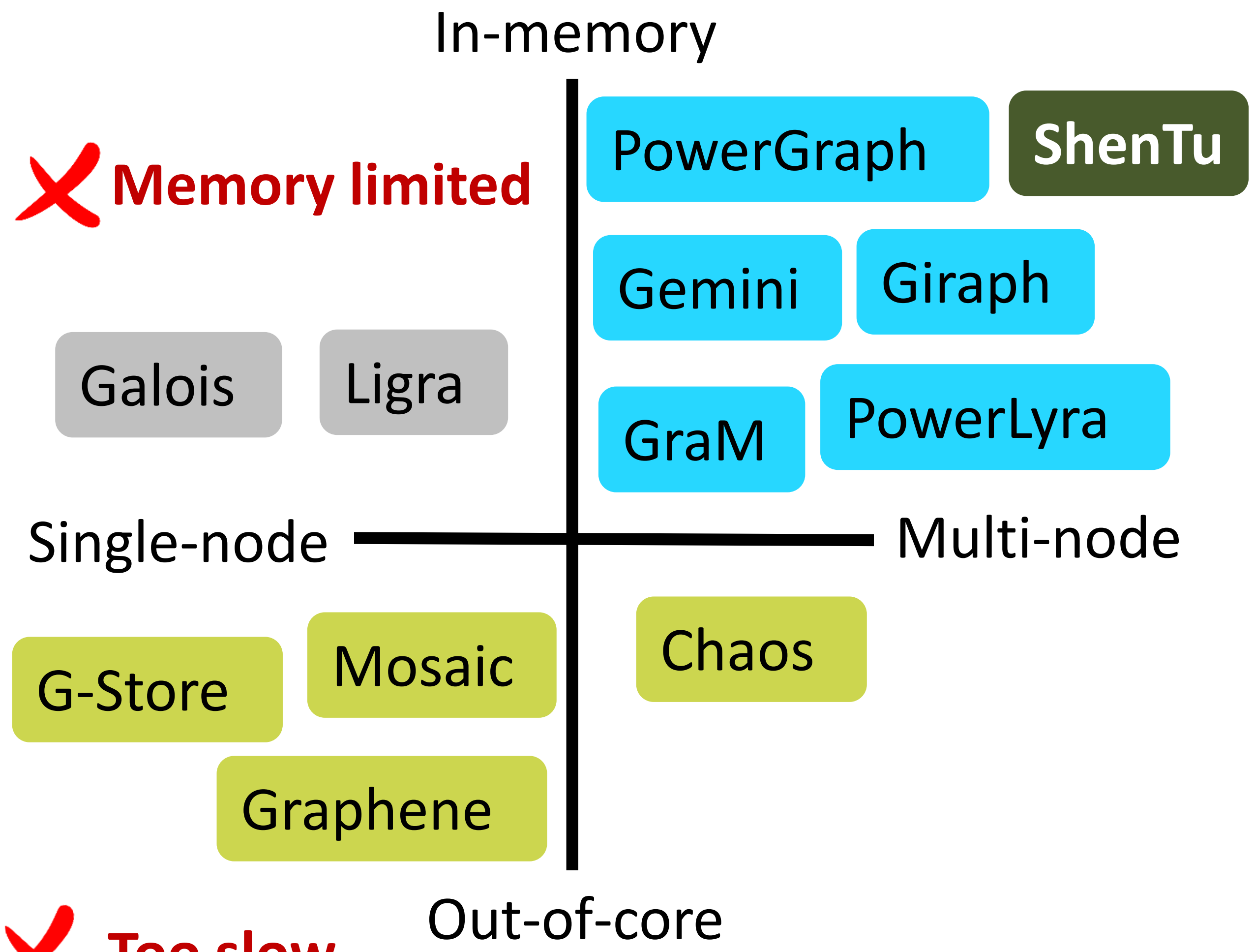


Too slow

(> 20 mins per PageRank iteration on 1-trillion-edge graph)

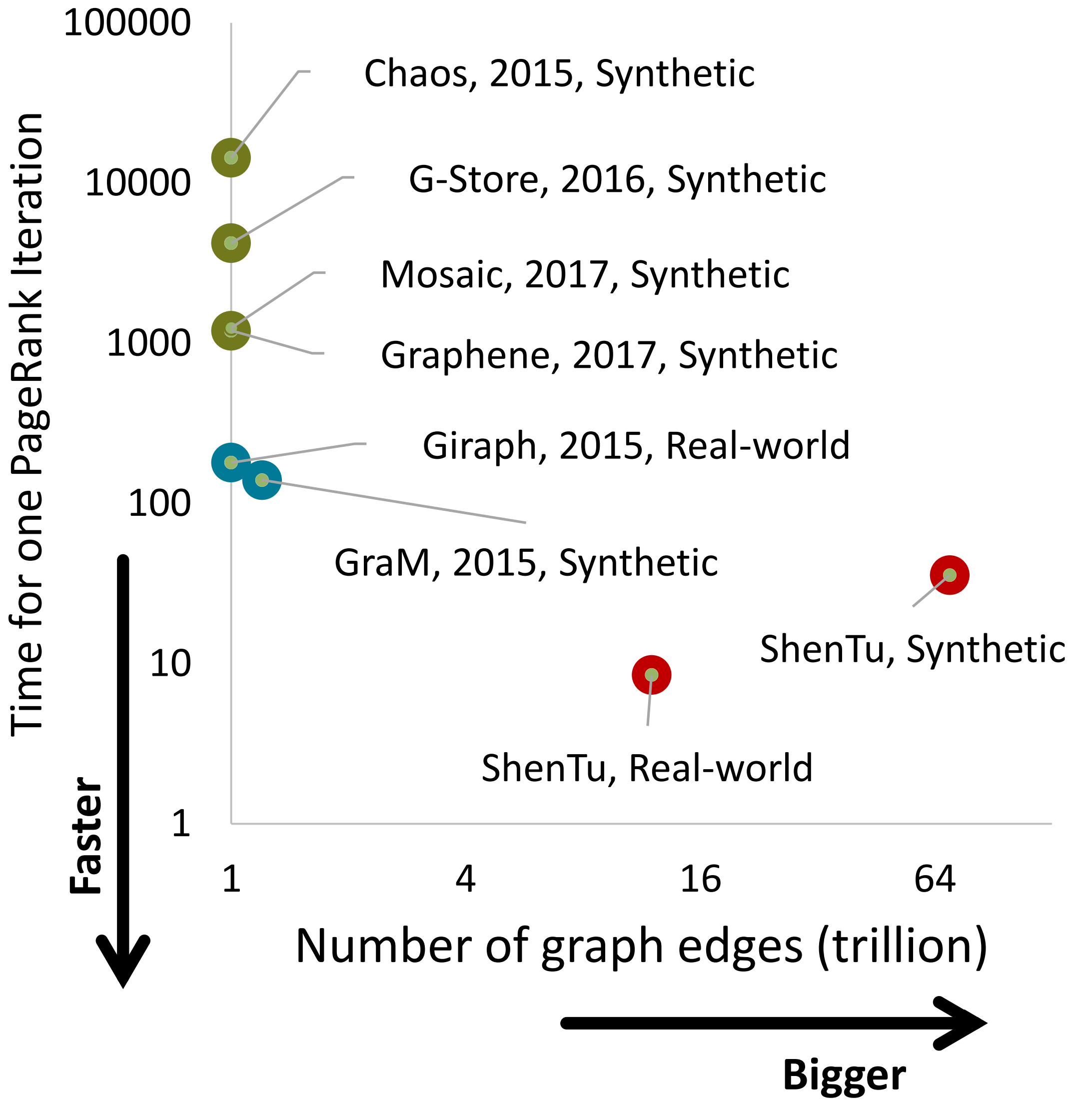
Heng Lin et al.: ShenTu: Processing Multi-Trillion Edge Graphs on Millions of Cores in Seconds, SC18, Gordon Bell Finalist

Practice of Extreme-Scale Graph Processing



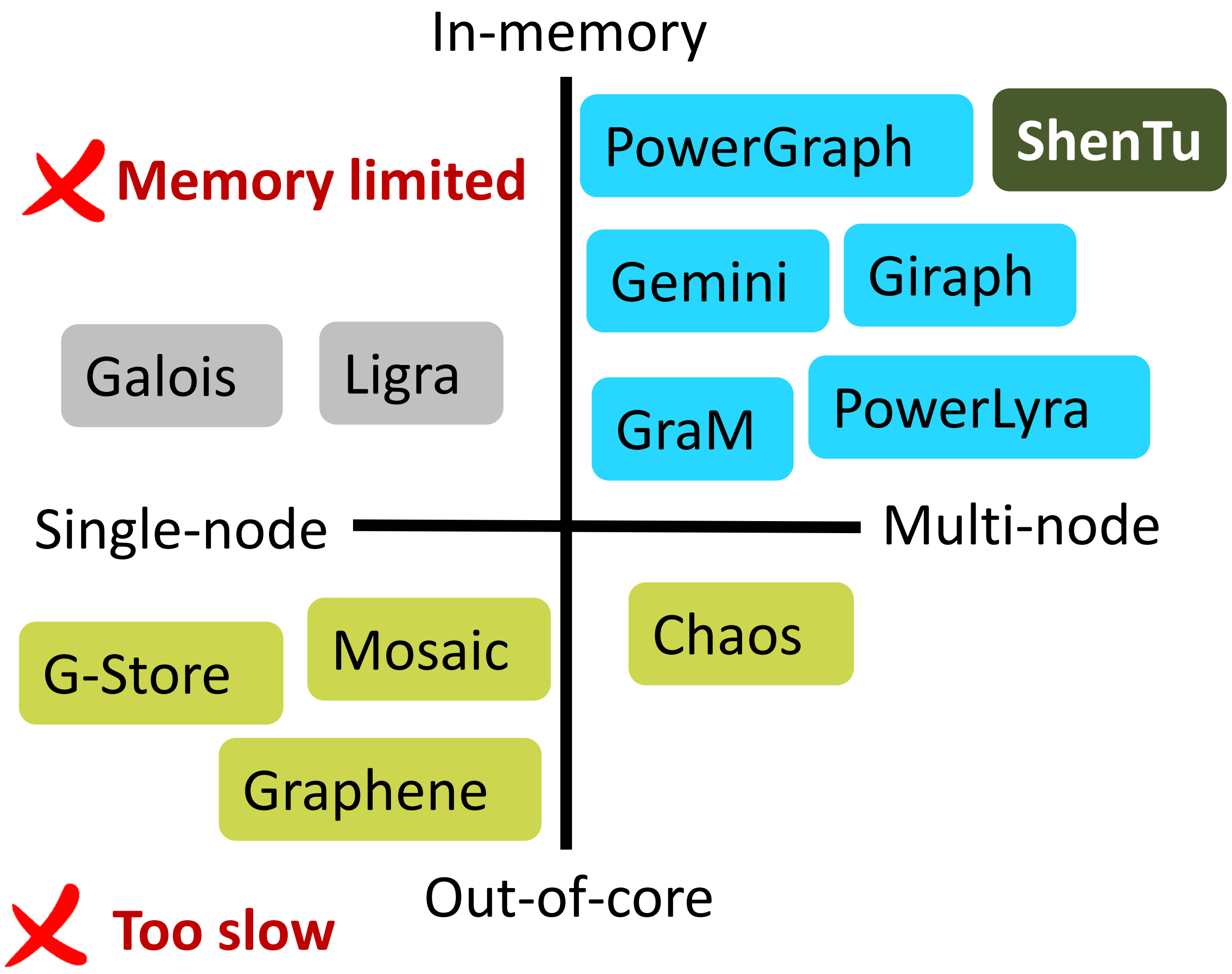
Too slow

(> 20 mins per PageRank iteration on 1-trillion-edge graph)

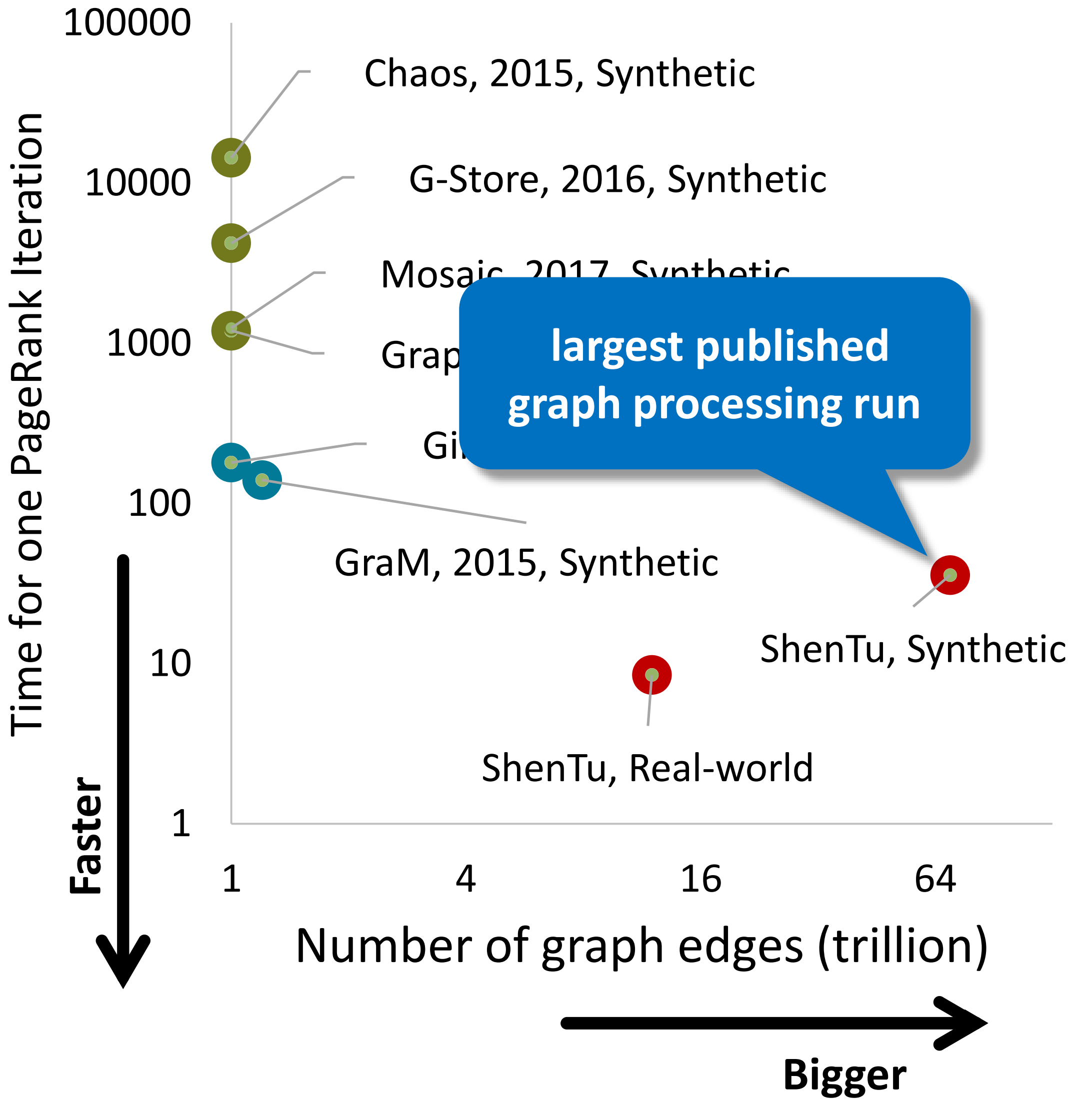


Heng Lin et al.: ShenTu: Processing Multi-Trillion Edge Graphs on Millions of Cores in Seconds, SC18, Gordon Bell Finalist

Practice of Extreme-Scale Graph Processing



(> 20 mins per PageRank iteration on 1-trillion-edge graph)



How large are extreme-scale graphs today?

Heng Lin et al.: ShenTu: Processing Multi-Trillion Edge Graphs on Millions of Cores in Seconds, SC18, Gordon Bell Finalist

How large are extreme-scale graphs today?

$1 = 10^0$

Heng Lin et al.: ShenTu: Processing Multi-Trillion Edge Graphs on Millions of Cores in Seconds, SC18, Gordon Bell Finalist

How large are extreme-scale graphs today?

$1 = 10^0$
$10 = 10^1$

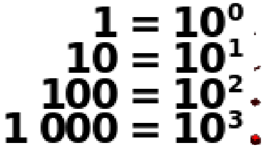
Heng Lin et al.: ShenTu: Processing Multi-Trillion Edge Graphs on Millions of Cores in Seconds, SC18, Gordon Bell Finalist

How large are extreme-scale graphs today?

$1 = 10^0$
$10 = 10^1$
$100 = 10^2$

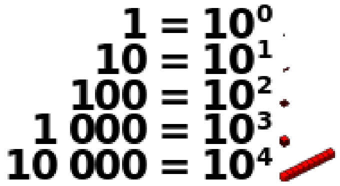
Heng Lin et al.: ShenTu: Processing Multi-Trillion Edge Graphs on Millions of Cores in Seconds, SC18, Gordon Bell Finalist

How large are extreme-scale graphs today?



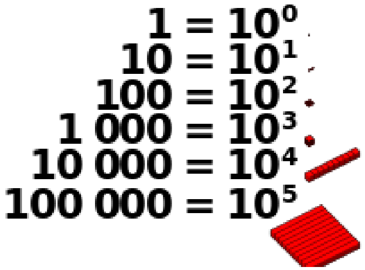
Heng Lin et al.: ShenTu: Processing Multi-Trillion Edge Graphs on Millions of Cores in Seconds, SC18, Gordon Bell Finalist

How large are extreme-scale graphs today?



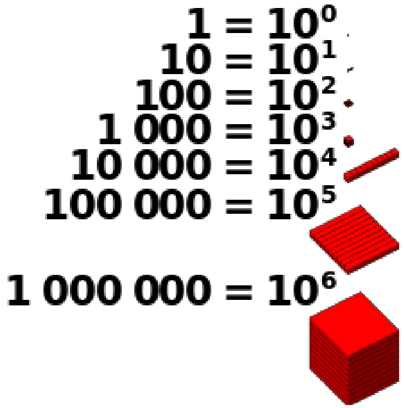
Heng Lin et al.: ShenTu: Processing Multi-Trillion Edge Graphs on Millions of Cores in Seconds, SC18, Gordon Bell Finalist

How large are extreme-scale graphs today?



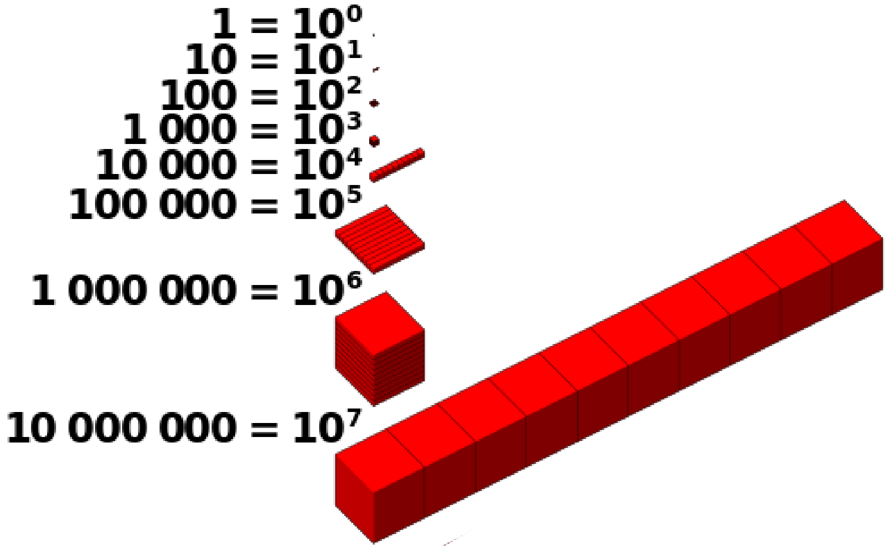
Heng Lin et al.: ShenTu: Processing Multi-Trillion Edge Graphs on Millions of Cores in Seconds, SC18, Gordon Bell Finalist

How large are extreme-scale graphs today?



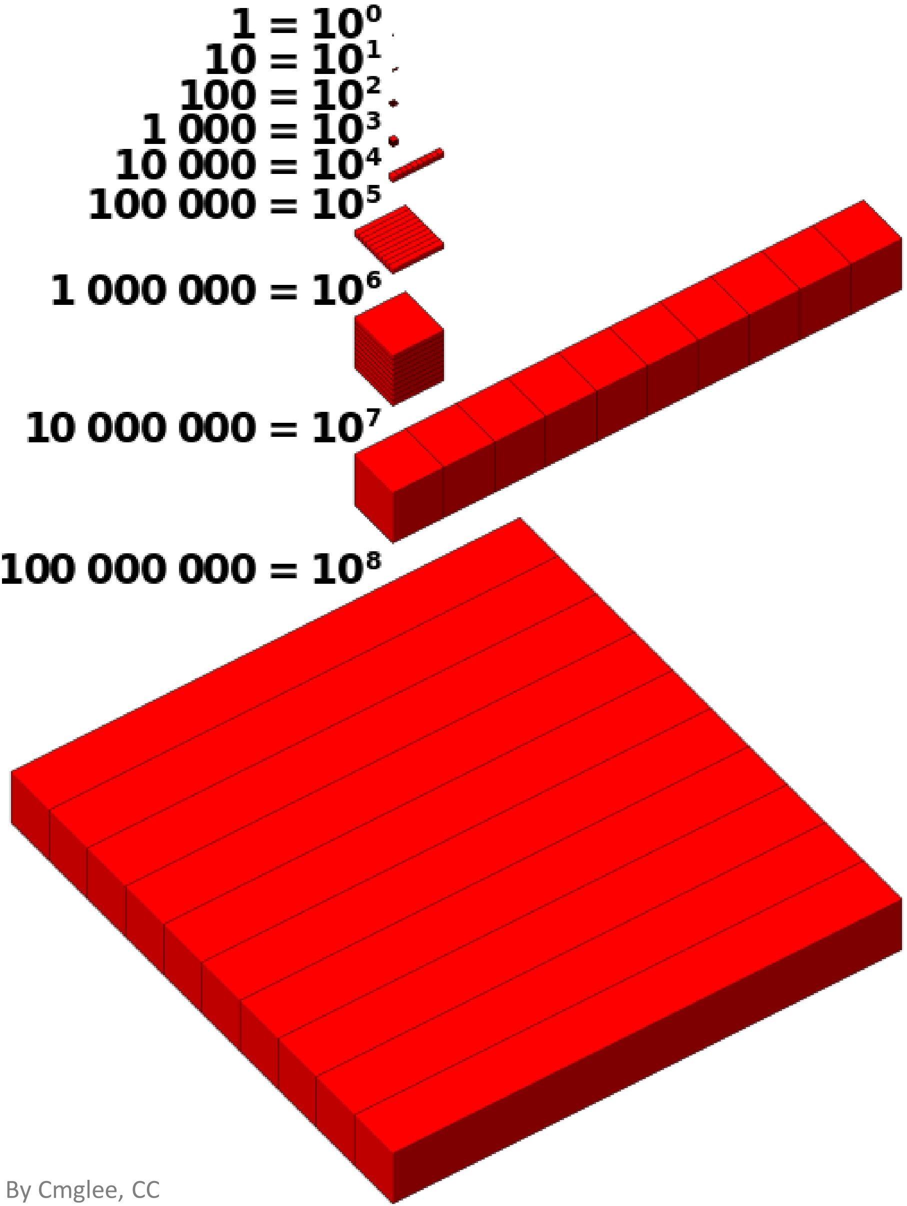
Heng Lin et al.: ShenTu: Processing Multi-Trillion Edge Graphs on Millions of Cores in Seconds, SC18, Gordon Bell Finalist

How large are extreme-scale graphs today?



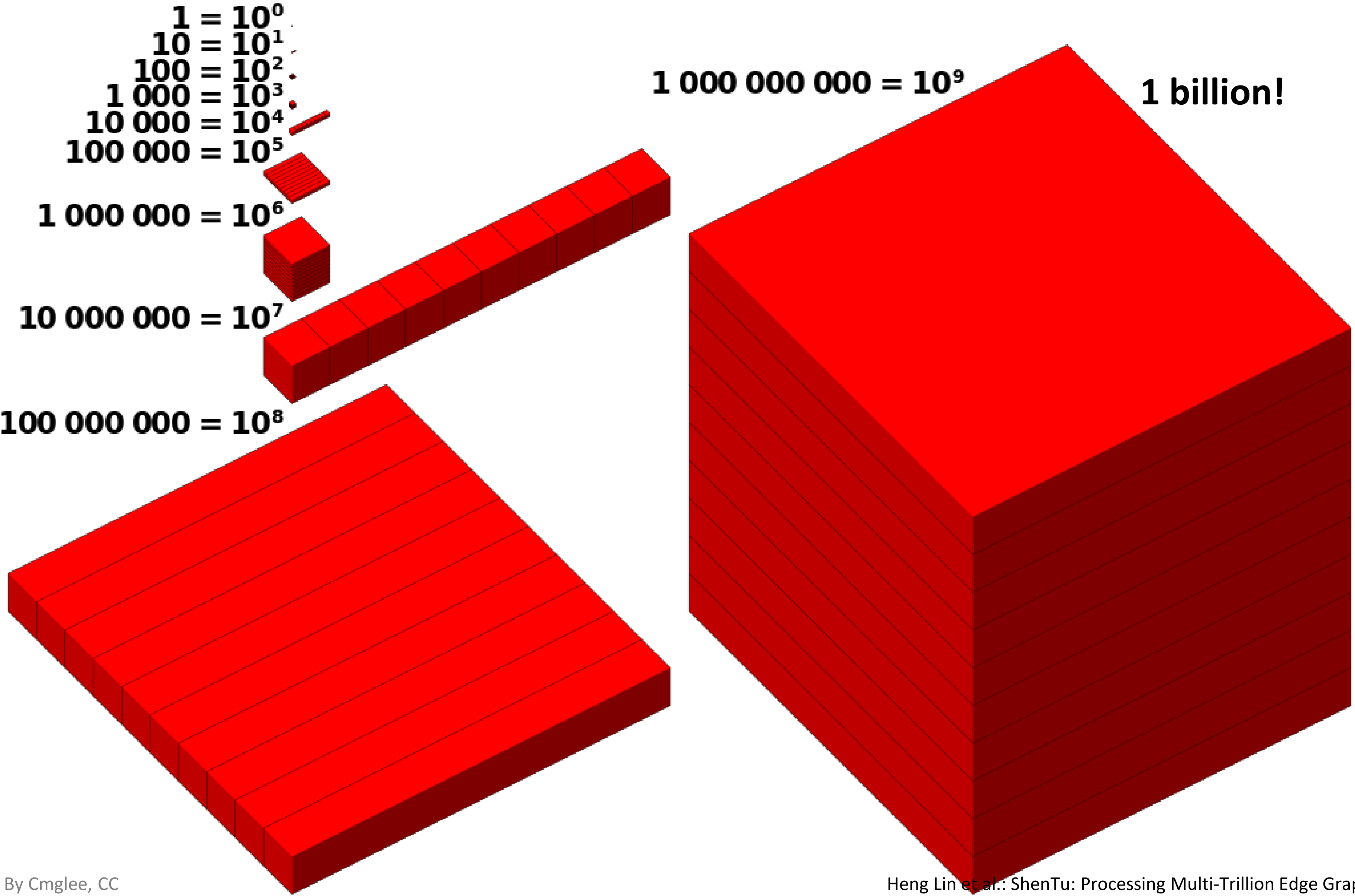
Heng Lin et al.: ShenTu: Processing Multi-Trillion Edge Graphs on Millions of Cores in Seconds, SC18, Gordon Bell Finalist

How large are extreme-scale graphs today?



Heng Lin et al.: ShenTu: Processing Multi-Trillion Edge Graphs on Millions of Cores in Seconds, SC18, Gordon Bell Finalist

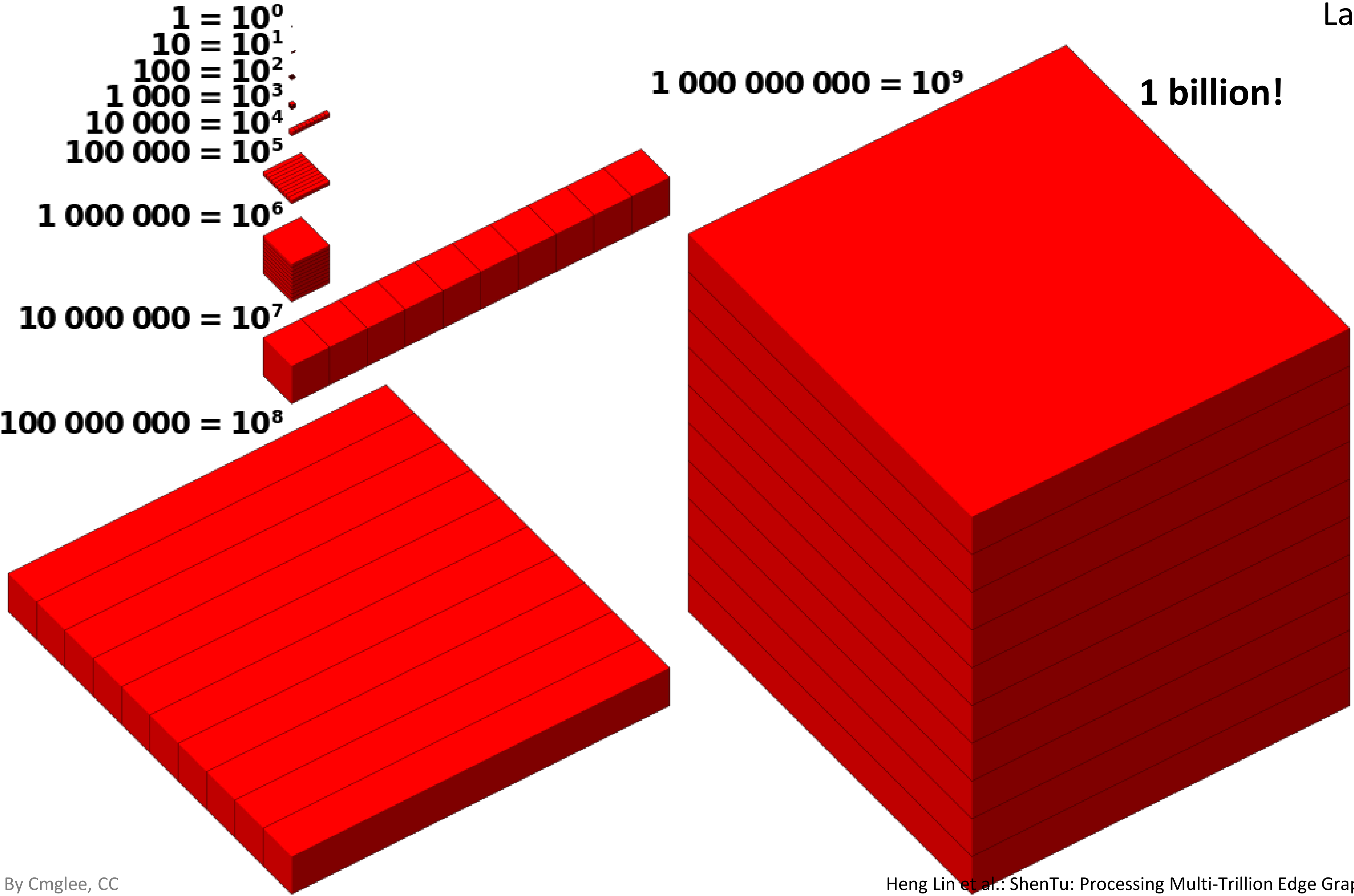
How large are extreme-scale graphs today?



1 billion!

Heng Lin et al.: ShenTu: Processing Multi-Trillion Edge Graphs on Millions of Cores in Seconds, SC18, Gordon Bell Finalist

How large are extreme-scale graphs today?



Largest Published Graph Computation

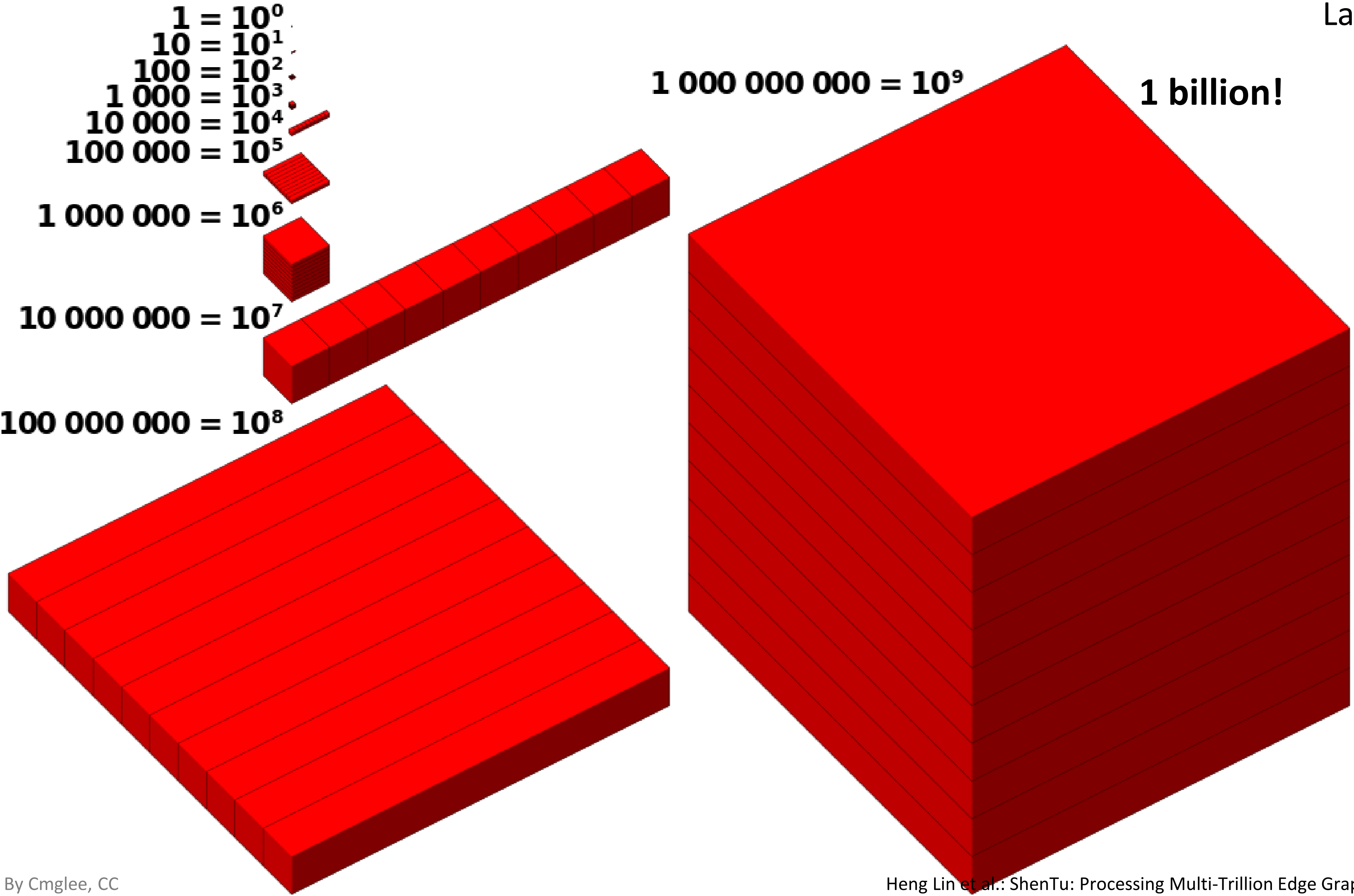
Gordon Bell Finalist 2018

ShenTu on Sunway TaihuLight

By Cmglee, CC

Heng Lin et al.: ShenTu: Processing Multi-Trillion Edge Graphs on Millions of Cores in Seconds, SC18, Gordon Bell Finalist

How large are extreme-scale graphs today?



1 billion!

Largest Published Graph Computation

Gordon Bell Finalist 2018

ShenTu on Sunway TaihuLight

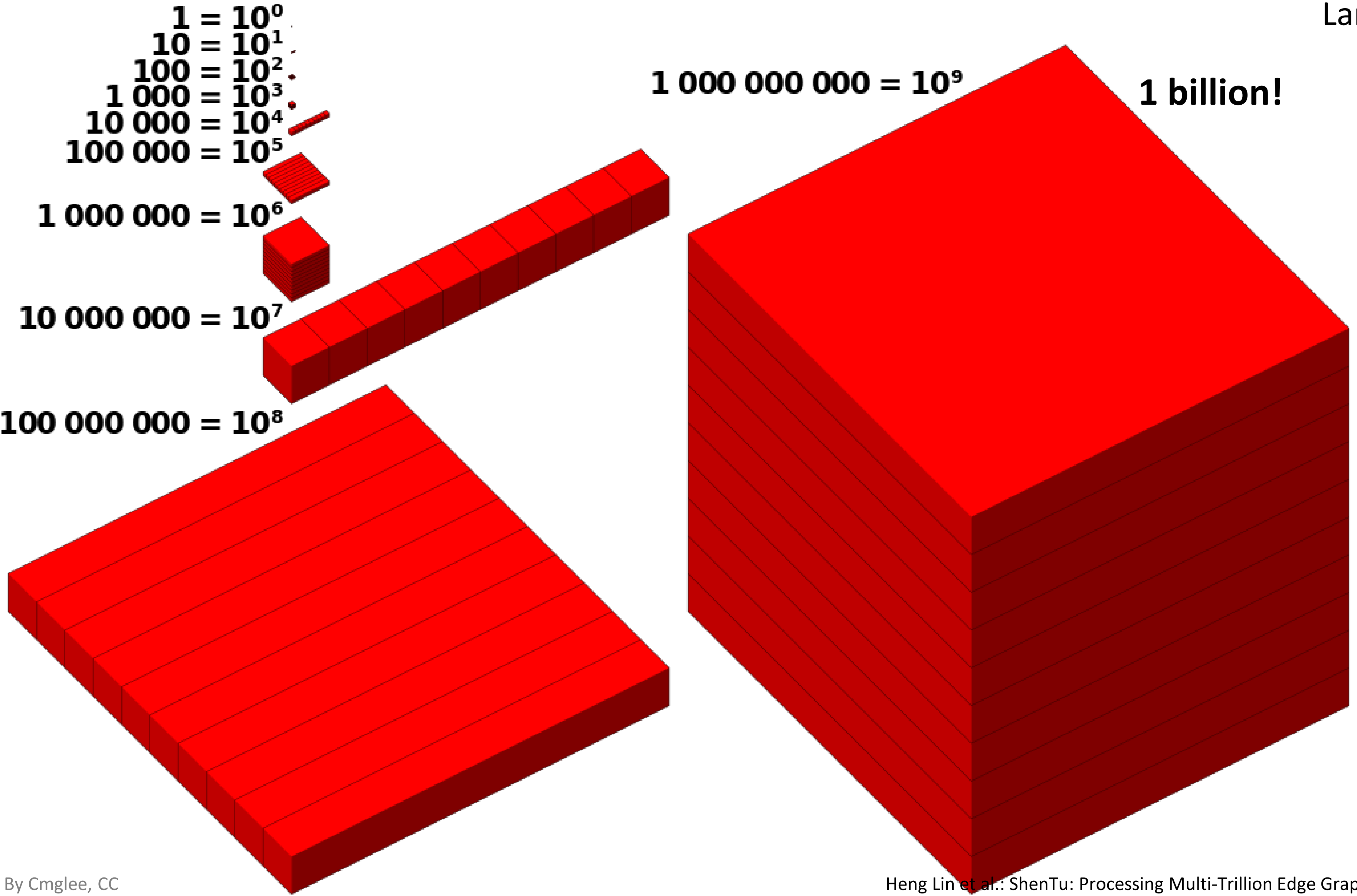
Sogou 搜狗

271 billion vertices

12 trillion edges

Heng Lin et al.: ShenTu: Processing Multi-Trillion Edge Graphs on Millions of Cores in Seconds, SC18, Gordon Bell Finalist

How large are extreme-scale graphs today?



1 billion!

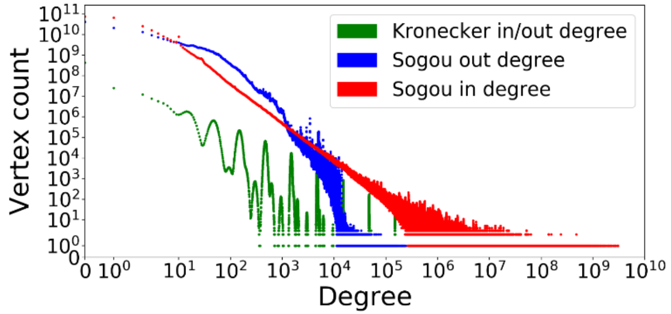
Largest Published Graph Computation

Gordon Bell Finalist 2018

ShenTu on Sunway TaihuLight

271 billion vertices

12 trillion edges



4.4 trillion vertices

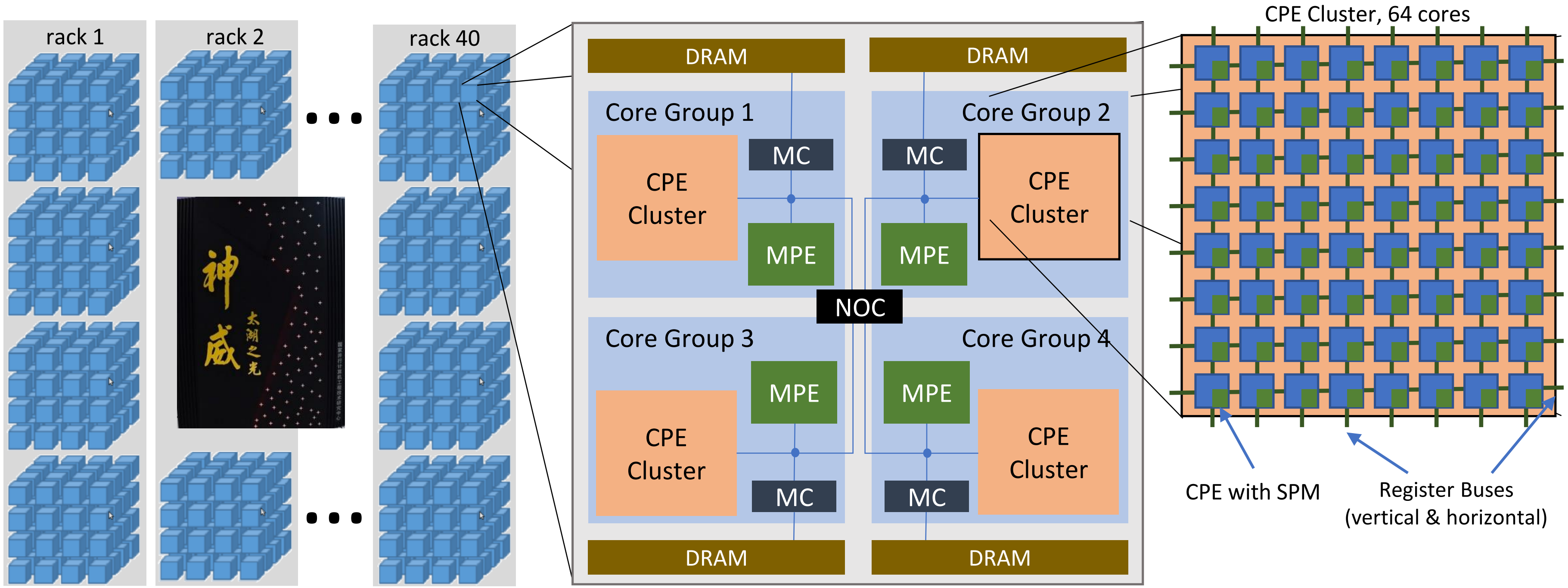
70 trillion edges

Heng Lin et al.: ShenTu: Processing Multi-Trillion Edge Graphs on Millions of Cores in Seconds, SC18, Gordon Bell Finalist

Sunway TaihuLight

Heng Lin et al.: ShenTu: Processing Multi-Trillion Edge Graphs on Millions of Cores in Seconds, SC18, Gordon Bell Finalist

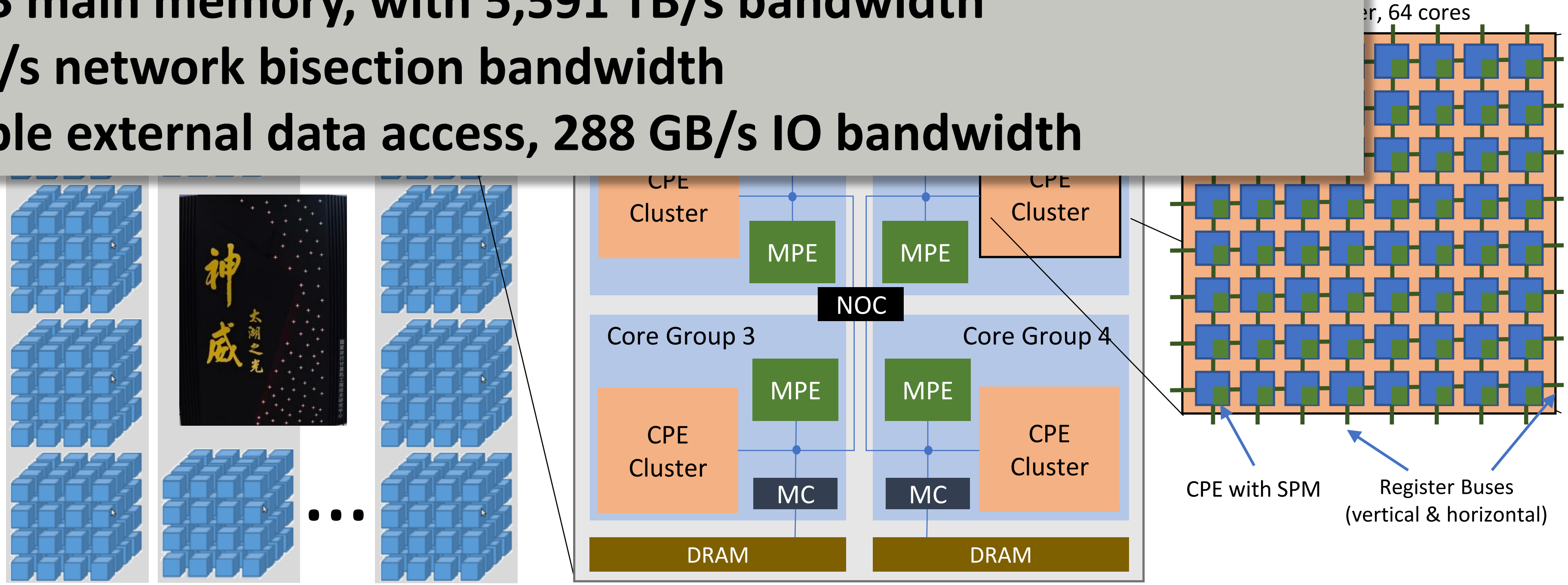
Sunway TaihuLight



TaihuLight Top500 ranking: #3 (2018 Nov), #1 (2016, 2017)

Sunway TaihuLight

- 1/8 EFLOPS peak performance
- 1.3 PB main memory, with 5,591 TB/s bandwidth
- 70 TB/s network bisection bandwidth
- Reliable external data access, 288 GB/s IO bandwidth

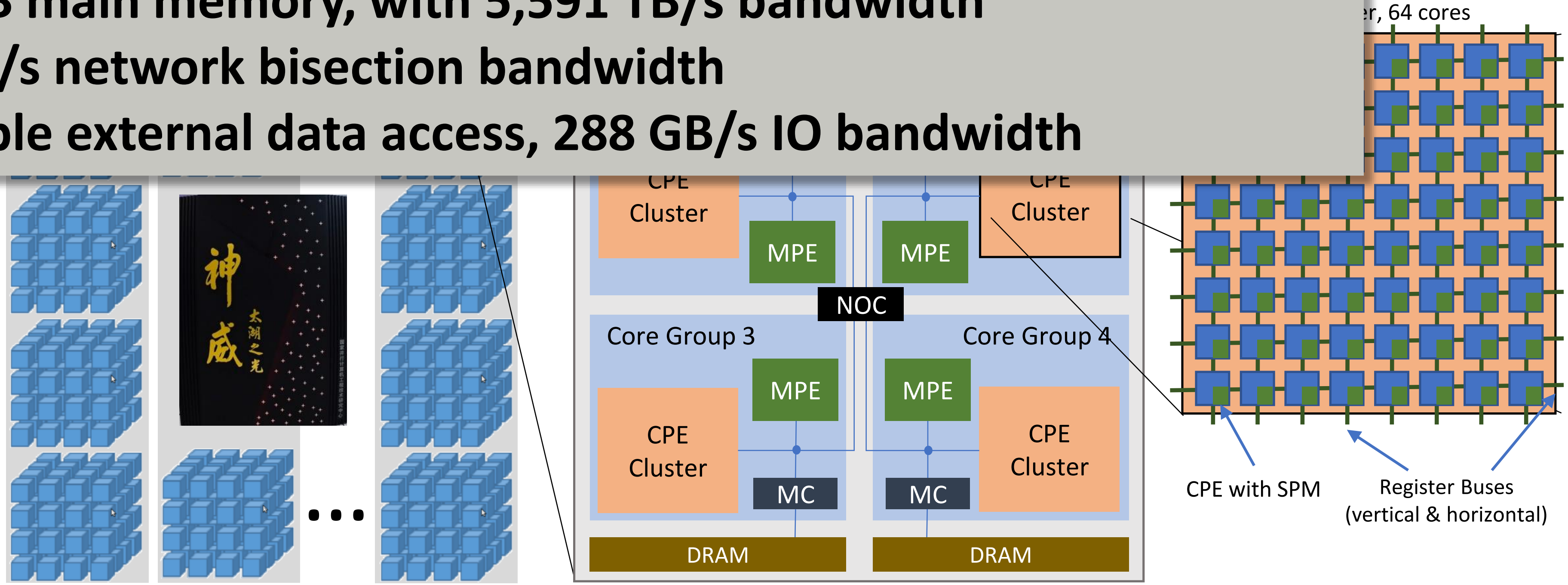


TaihuLight Top500 ranking: #3 (2018 Nov), #1 (2016, 2017)

Heng Lin et al.: ShenTu: Processing Multi-Trillion Edge Graphs on Millions of Cores in Seconds, SC18, Gordon Bell Finalist

Sunway TaihuLight

- 1/8 EFLOPS peak performance
- 1.3 PB main memory, with 5,591 TB/s bandwidth
- 70 TB/s network bisection bandwidth
- Reliable external data access, 288 GB/s IO bandwidth



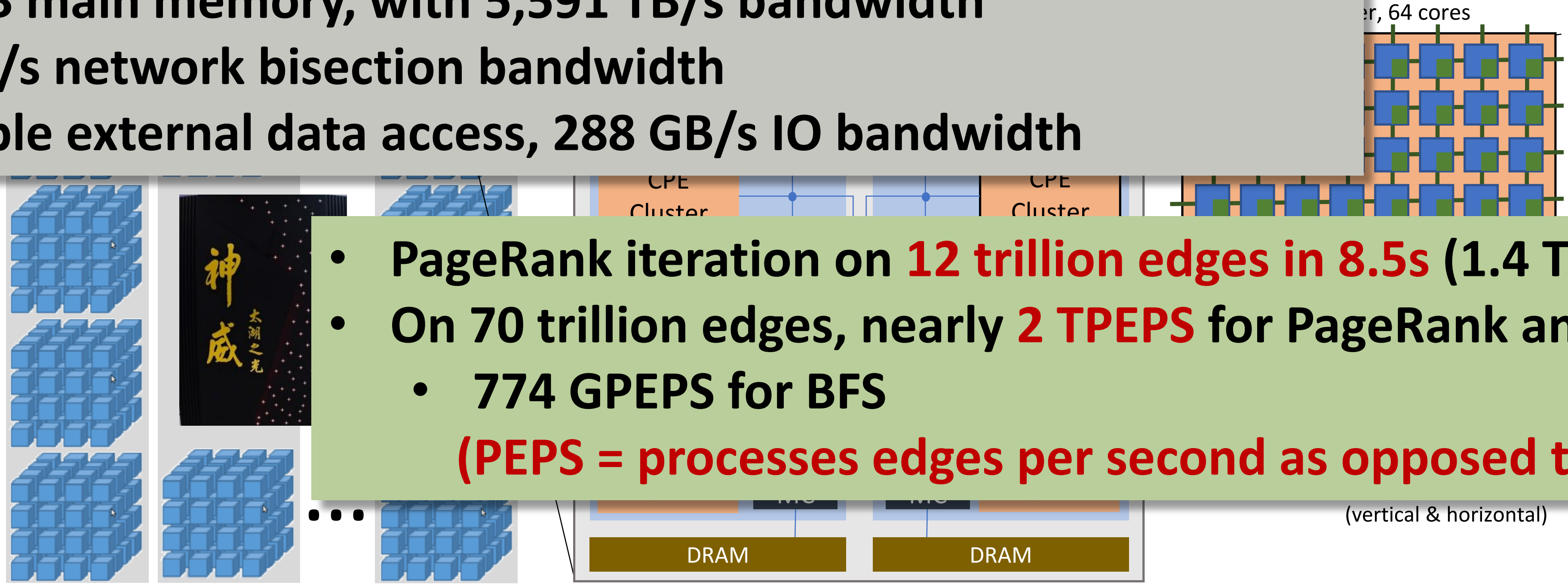
TaihuLight Top500 ranking: #3 (2018 Nov), #1 (2016, 2017)

- Handling of huge number of messages among 40,960 nodes
- Complex workload to map to its heterogeneous processing units
- Irregular data flow to be scheduled in regular accelerator cores

Heng Lin et al.: ShenTu: Processing Multi-Trillion Edge Graphs on Millions of Cores in Seconds, SC18, Gordon Bell Finalist

Sunway TaihuLight

- 1/8 EFLOPS peak performance
- 1.3 PB main memory, with 5,591 TB/s bandwidth
- 70 TB/s network bisection bandwidth
- Reliable external data access, 288 GB/s IO bandwidth



- PageRank iteration on **12 trillion edges in 8.5s** (1.4 TPEPS)
- On 70 trillion edges, nearly **2 TPEPS** for PageRank and WCC
  - 774 GPEPS for BFS

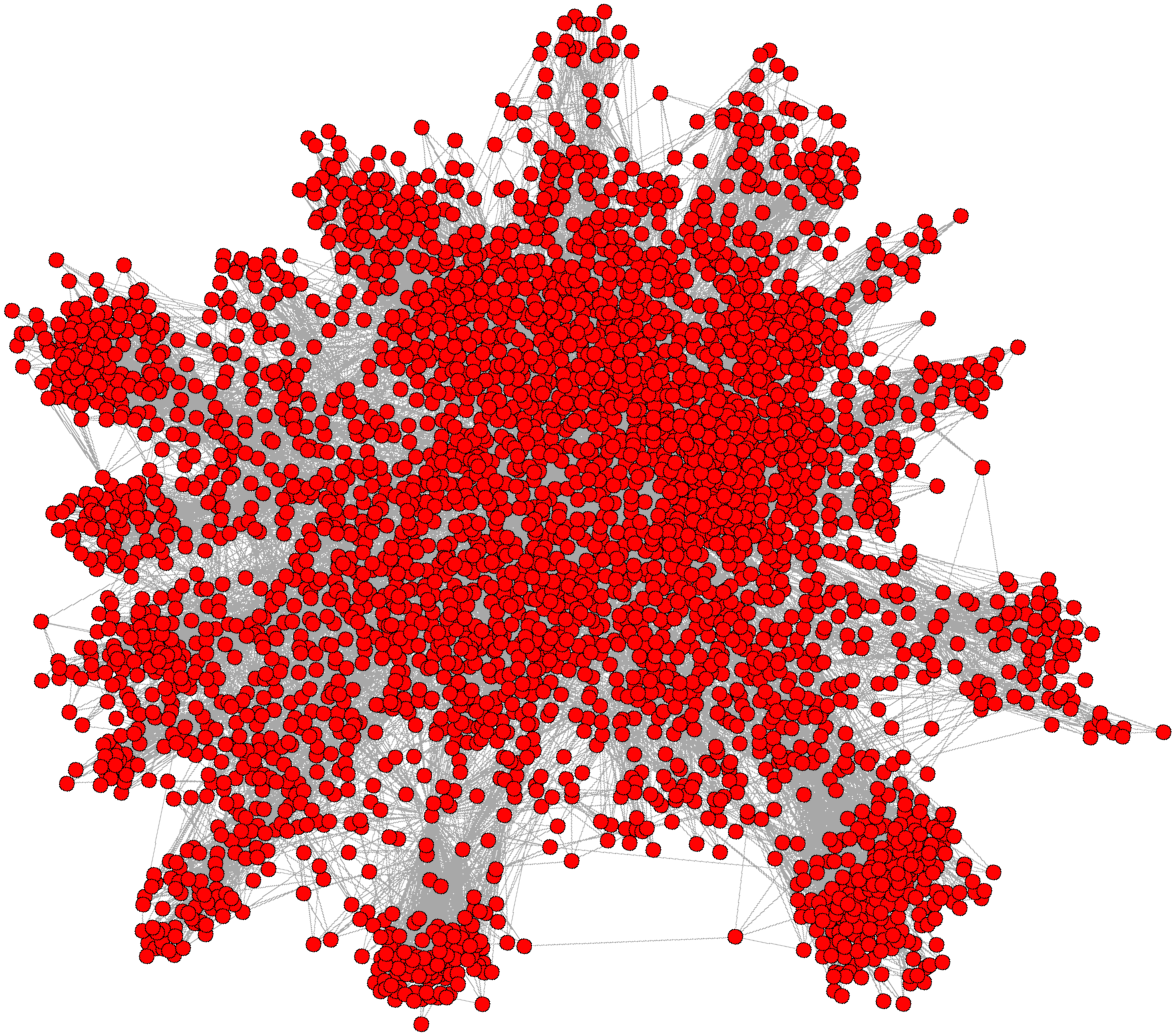
  (PEPS = processes edges per second as opposed to TEPS)

TaihuLight Top500 ranking: #3 (2018 Nov), #1 (2016, 2017)

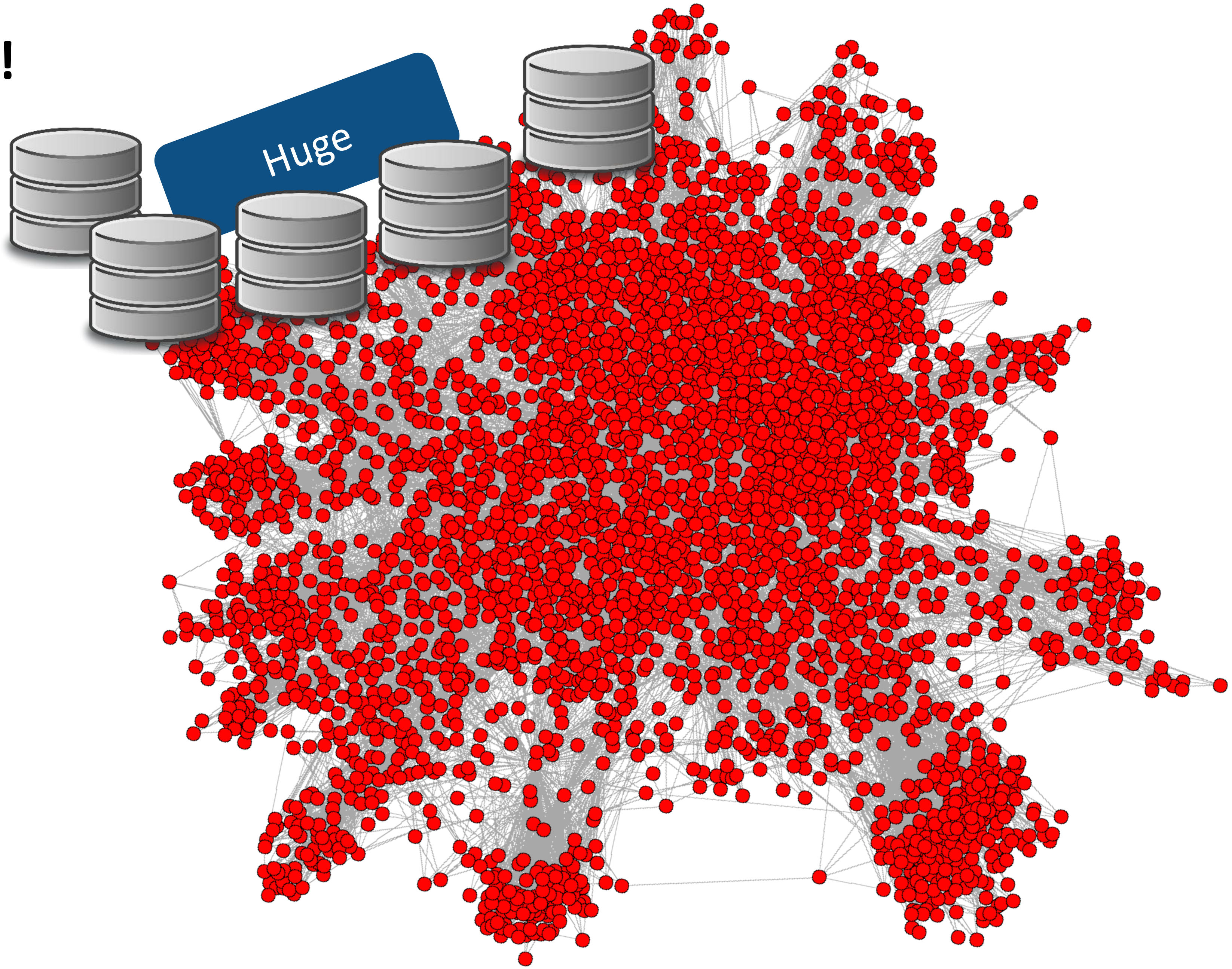
- Handling of huge number of messages among 40,960 nodes
- Complex workload to map to its heterogeneous processing units
- Irregular data flow to be scheduled in regular accelerator cores

Heng Lin et al.: ShenTu: Processing Multi-Trillion Edge Graphs on Millions of Cores in Seconds, SC18, Gordon Bell Finalist

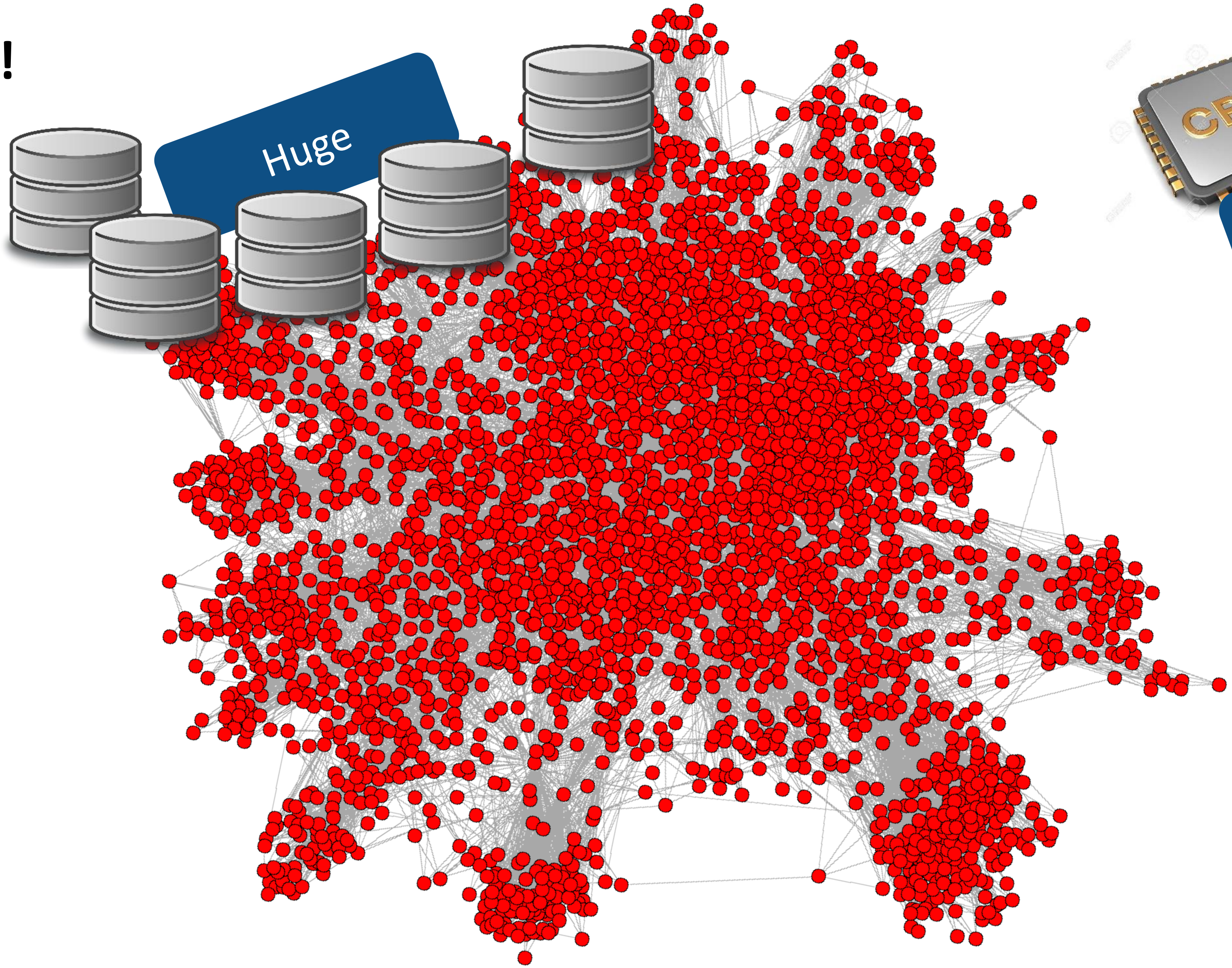
Problems!

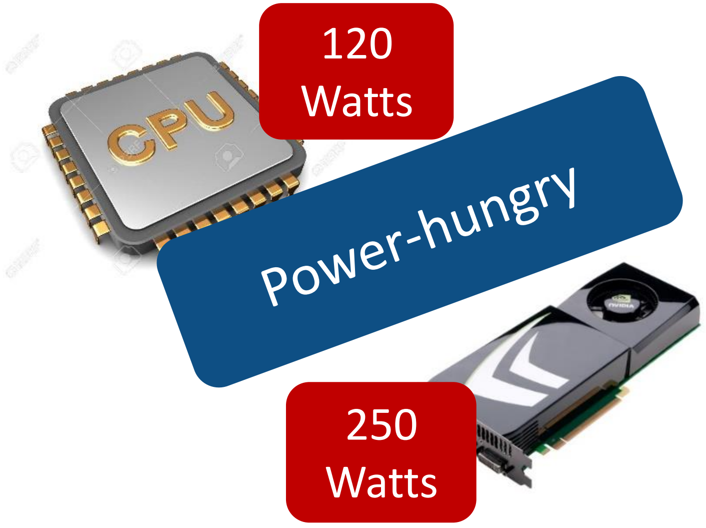


Problems!



Problems!





Problems!

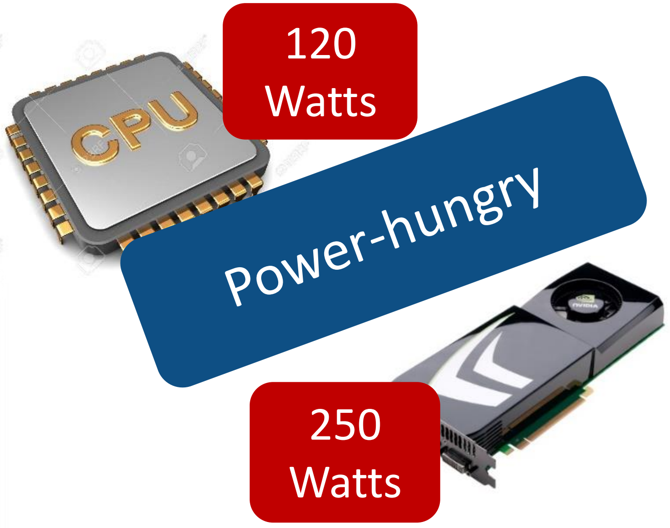

Huge
120
Watts
CPU
Power-hungry
250
Watts
Communication-heavy

Problems!


Huge
120
Watts
Power-hungry
250
Watts
Communication-heavy
Synchronization-heavy

Problems!


Huge
Communication-heavy
Irregular

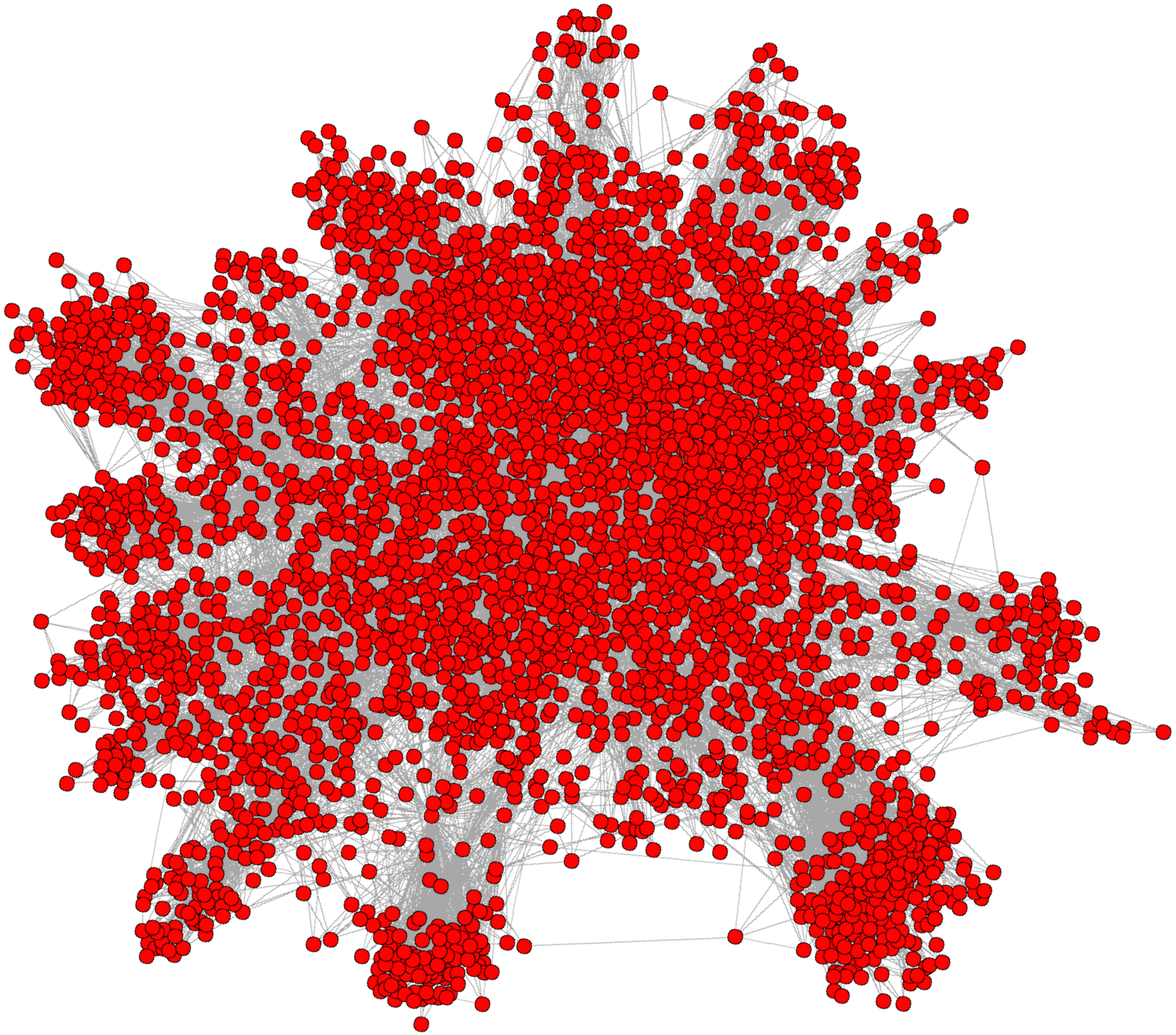

CPU
120
Watts
Power-hungry
250
Watts


Synchronization-heavy

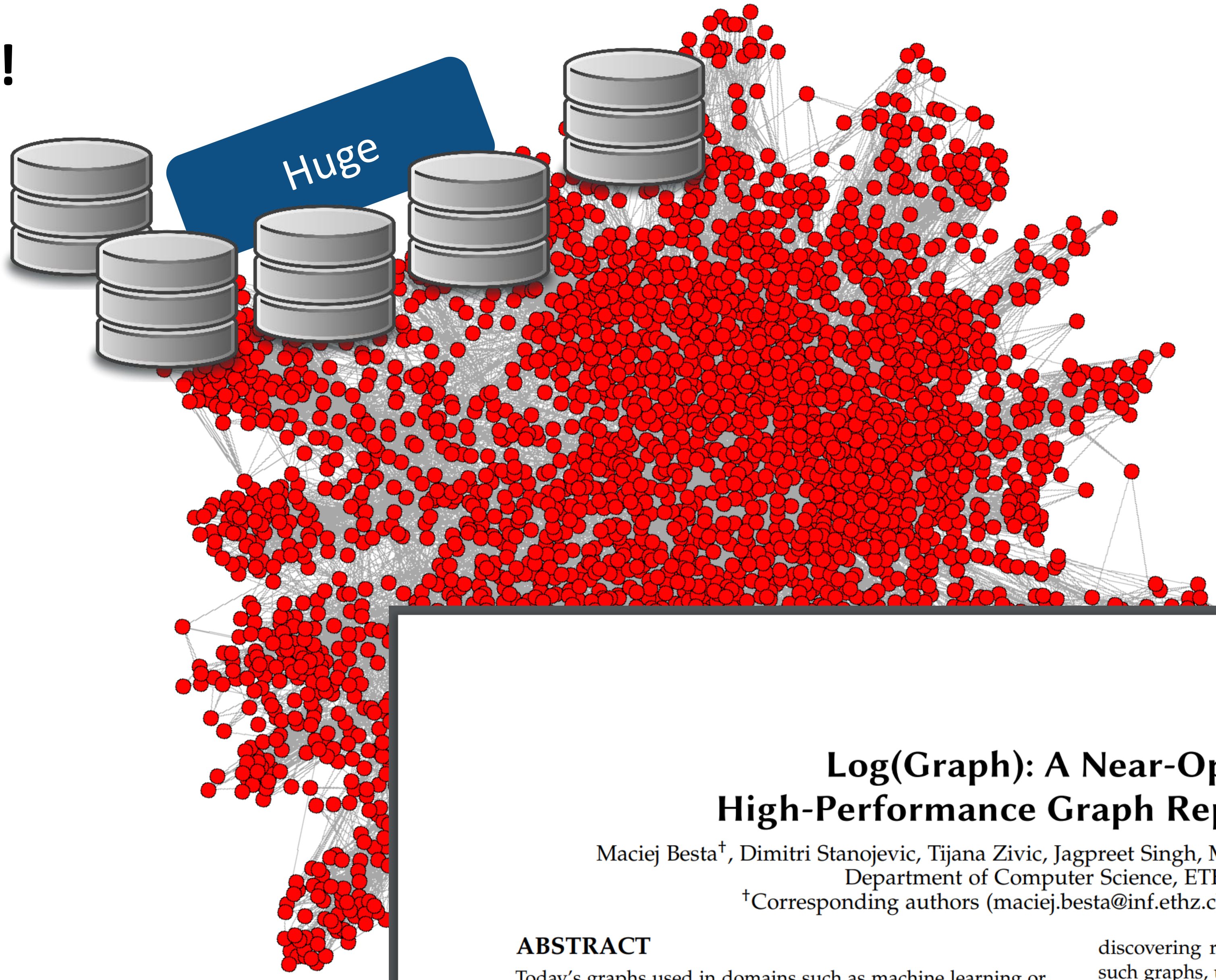
Problems!


Huge
120
Watts
Power-hungry
250
Watts
It's Complicated...
Communication-heavy
Synchronization-heavy
Irregular

Problems!



Problems!



Log(Graph): A Near-Optimal High-Performance Graph Representation

Maciej Besta†, Dimitri Stanojevic, Tijana Zivic, Jagpreet Singh, Maurice Hoerold, Torsten Hoefler†
Department of Computer Science, ETH Zurich
†Corresponding authors (maciej.besta@inf.ethz.ch, htor@inf.ethz.ch)

ABSTRACT

Today's graphs used in domains such as machine learning or social network analysis may contain hundreds of billions of edges. Yet, they are not necessarily stored efficiently, and standard graph representations such as adjacency lists waste a significant number of bits while graph compression schemes

discovering relationships in graph data. The sheer size of such graphs, up to hundreds of billions of edges, exacerbates the number of needed memory banks, increases the amount of data transferred between CPUs and memory, and may lead to I/O accesses while processing graphs. Thus, reducing the size of such graphs is becoming increasingly important.

What is **the lowest storage** we can (hope to) use to store a graph?

What is **the lowest storage** we can (hope to) use to store a graph?

The storage **lower bound** Ω

What is **the lowest storage** we can (hope to) use to store a graph?

The storage **lower bound** Ω

Which one? ☺

What is **the lowest storage** we can (hope to) use to store a graph?

The storage **lower bound** Ω

Which one? ☺

Shannon's approach
logarithmic
(one needs at least log|S| bits to store an object from an arbitrary set S)

M. Besta et al.: "Log(Graph): A Near-Optimal High-Performance Graph Representation", PACT'18

What is **the lowest storage** we can (hope to) use to store a graph?

The storage **lower bound** Ω

Which one? ☺

Shannon's approach **logarithmic** (one needs at least log|S| bits to store an object from an arbitrary set S)

$$S = \{x_1, x_2, x_3, \dots\}$$

$$\begin{array}{l} x_1 \rightarrow 0 \dots 01 \\ x_2 \rightarrow 0 \dots 10 \\ x_3 \rightarrow 0 \dots 11 \\ \quad \dots \end{array}$$

M. Besta et al.: "Log(Graph): A Near-Optimal High-Performance Graph Representation", PACT'18

What is **the lowest storage** we can (hope to) use to store a graph?

The storage **lower bound** Ω

Which one? ☺

Shannon's approach **logarithmic** (one needs at least log|S| bits to store an object from an arbitrary set S)

$S = \{x_1, x_2, x_3, \ldots\}$

$x_1 \rightarrow 0 \ldots 01$
$x_2 \rightarrow 0 \ldots 10$
$x_3 \rightarrow 0 \ldots 11$
$\ldots$

Key idea

M. Besta et al.: "Log(Graph): A Near-Optimal High-Performance Graph Representation", PACT'18

What is **the lowest storage** we can (hope to) use to store a graph?

The storage **lower bound** Ω

Which one? ☺

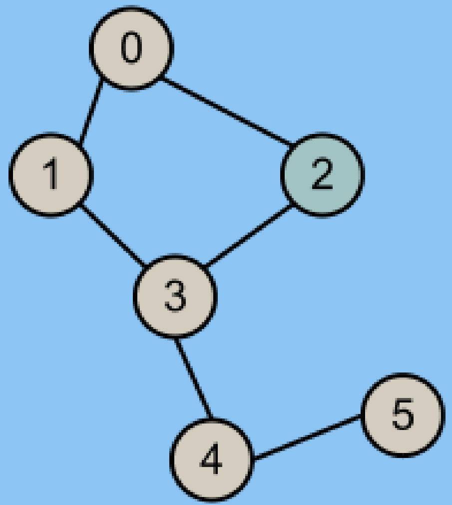
Shannon's approach **logarithmic** (one needs at least log|S| bits to store an object from an arbitrary set S)

$S = \{x_1, x_2, x_3, \dots\}$

$x_1 \rightarrow 0 \dots 01$
$x_2 \rightarrow 0 \dots 10$
$x_3 \rightarrow 0 \dots 11$
$\dots$

Key idea

Encode different parts of a graph representation using (logarithmic) **storage lower bounds**



M. Besta et al.: "Log(Graph): A Near-Optimal High-Performance Graph Representation", PACT'18

What is **the lowest storage** we can (hope to) use to store a graph?

The storage **lower bound** Ω

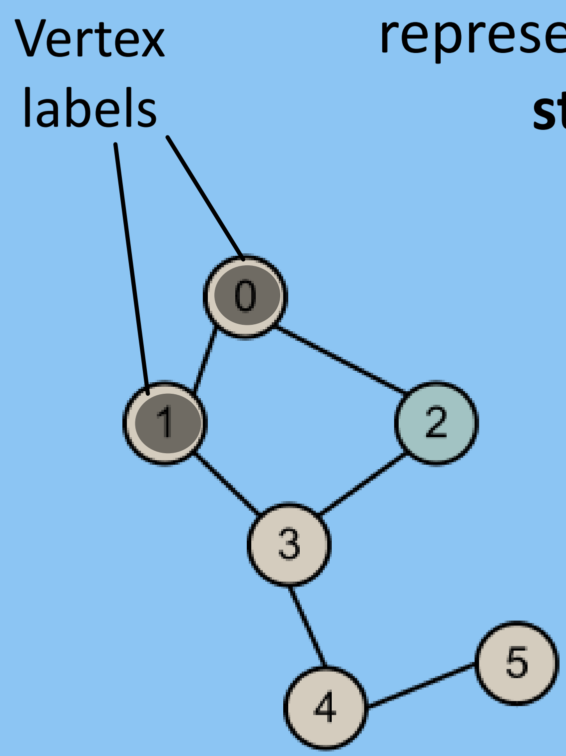
Which one? ☺

Shannon's approach **logarithmic** (one needs at least log|S| bits to store an object from an arbitrary set S)

$$S = \{x_1, x_2, x_3, \dots\} \quad \begin{array}{l} x_1 \rightarrow 0 \dots 01 \\ x_2 \rightarrow 0 \dots 10 \\ x_3 \rightarrow 0 \dots 11 \\ \quad \dots \end{array}$$

Key idea

Encode different parts of a graph representation using (logarithmic) **storage lower bounds**



What is **the lowest storage** we can (hope to) use to store a graph?

The storage **lower bound** Ω

Which one? ☺

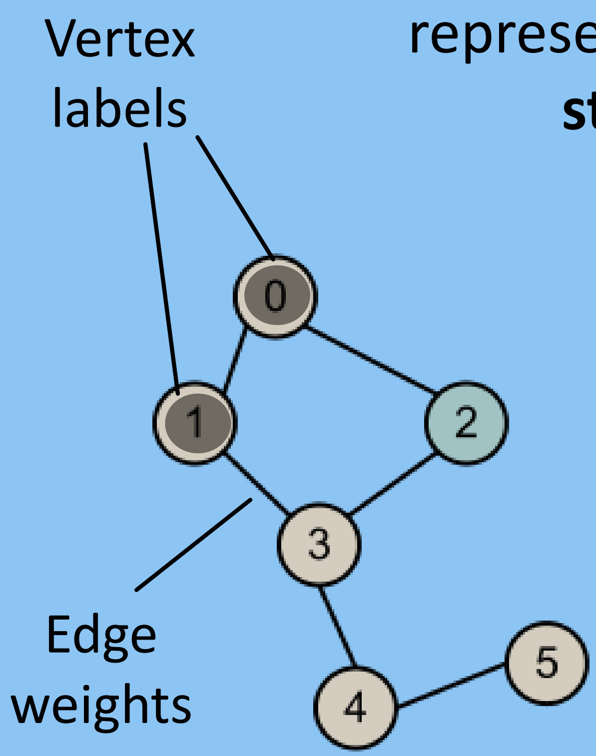
Shannon's approach **logarithmic** (one needs at least log|S| bits to store an object from an arbitrary set S)

$S = \{x_1, x_2, x_3, \ldots\}$

$x_1 \to 0 \ldots 01$
$x_2 \to 0 \ldots 10$
$x_3 \to 0 \ldots 11$
…

Key idea

Encode different parts of a graph representation using (logarithmic) **storage lower bounds**



M. Besta et al.: "Log(Graph): A Near-Optimal High-Performance Graph Representation", PACT'18

What is **the lowest storage** we can (hope to) use to store a graph?

$$S = \{x_1, x_2, x_3, \dots\} \quad \begin{array}{l} x_1 \to 0\dots01 \\ x_2 \to 0\dots10 \\ x_3 \to 0\dots11 \\ \dots \end{array}$$

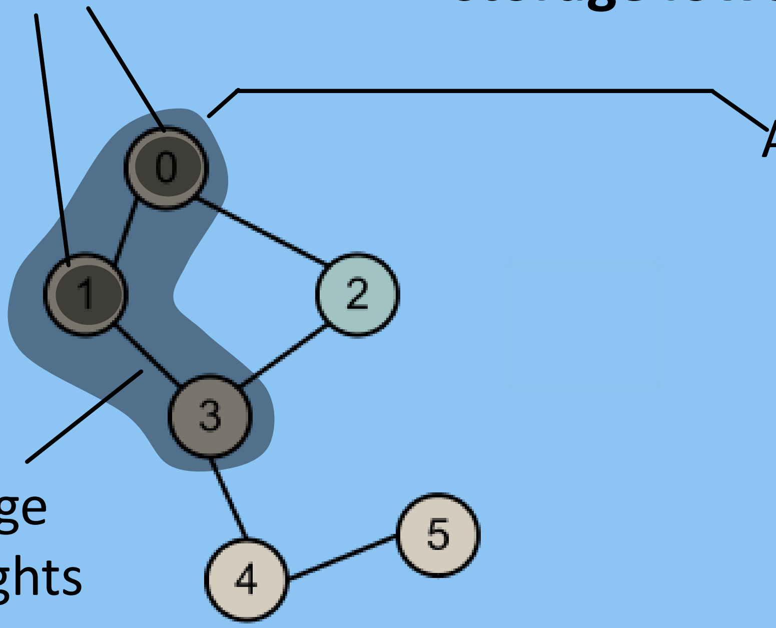
The storage **lower bound** Ω

Which one? ☺

Shannon's approach **logarithmic** (one needs at least log|S| bits to store an object from an arbitrary set S)

Key idea

Encode different parts of a graph representation using (logarithmic) **storage lower bounds**



What is **the lowest storage** we can (hope to) use to store a graph?

The storage **lower bound** Ω

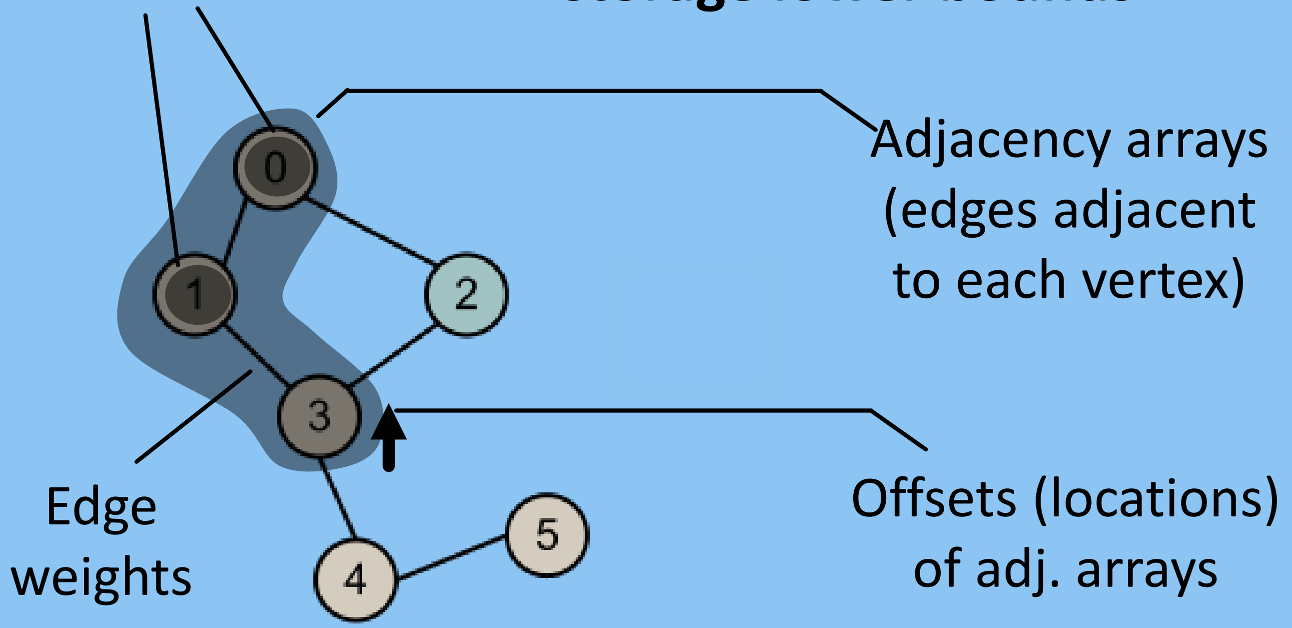
Which one? ☺

Shannon's approach **logarithmic** (one needs at least log|S| bits to store an object from an arbitrary set S)

$$S = \{x_1, x_2, x_3, \dots\} \quad \begin{array}{l} x_1 \rightarrow 0 \dots 01 \\ x_2 \rightarrow 0 \dots 10 \\ x_3 \rightarrow 0 \dots 11 \\ \quad\dots \end{array}$$

Key idea

Encode different parts of a graph representation using (logarithmic) **storage lower bounds**



M. Besta et al.: "Log(Graph): A Near-Optimal High-Performance Graph Representation", PACT'18

What is **the lowest storage** we can (hope to) use to store a graph?

$$S = \{x_1, x_2, x_3, \dots\} \quad \begin{array}{l} x_1 \rightarrow 0\dots01 \\ x_2 \rightarrow 0\dots10 \\ x_3 \rightarrow 0\dots11 \\ \dots \end{array}$$

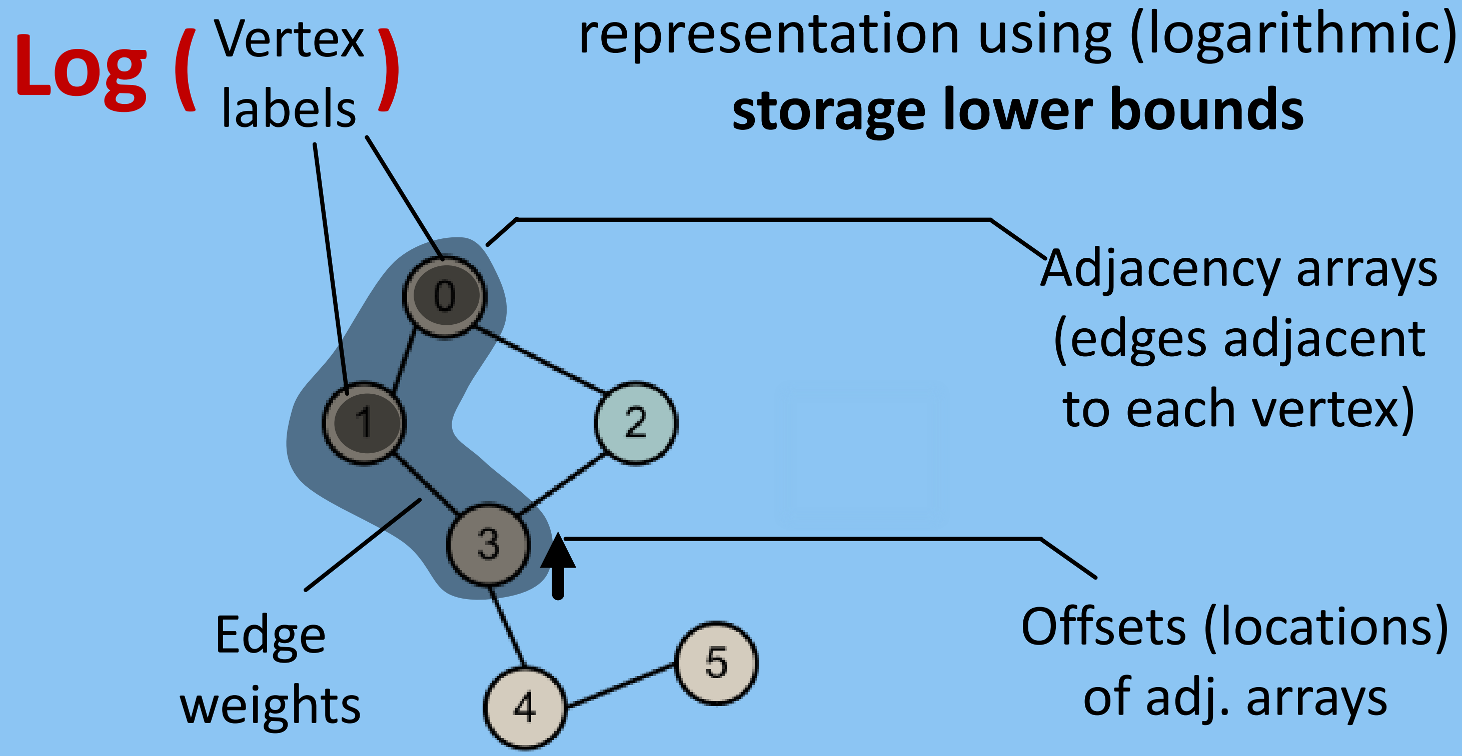
The storage **lower bound** Ω

Which one? ☺

Shannon's approach **logarithmic** (one needs at least log|S| bits to store an object from an arbitrary set S)

Key idea

Encode different parts of a graph representation using (logarithmic) **storage lower bounds**



M. Besta et al.: "Log(Graph): A Near-Optimal High-Performance Graph Representation", PACT'18

What is **the lowest storage** we can (hope to) use to store a graph?

The storage **lower bound** Ω

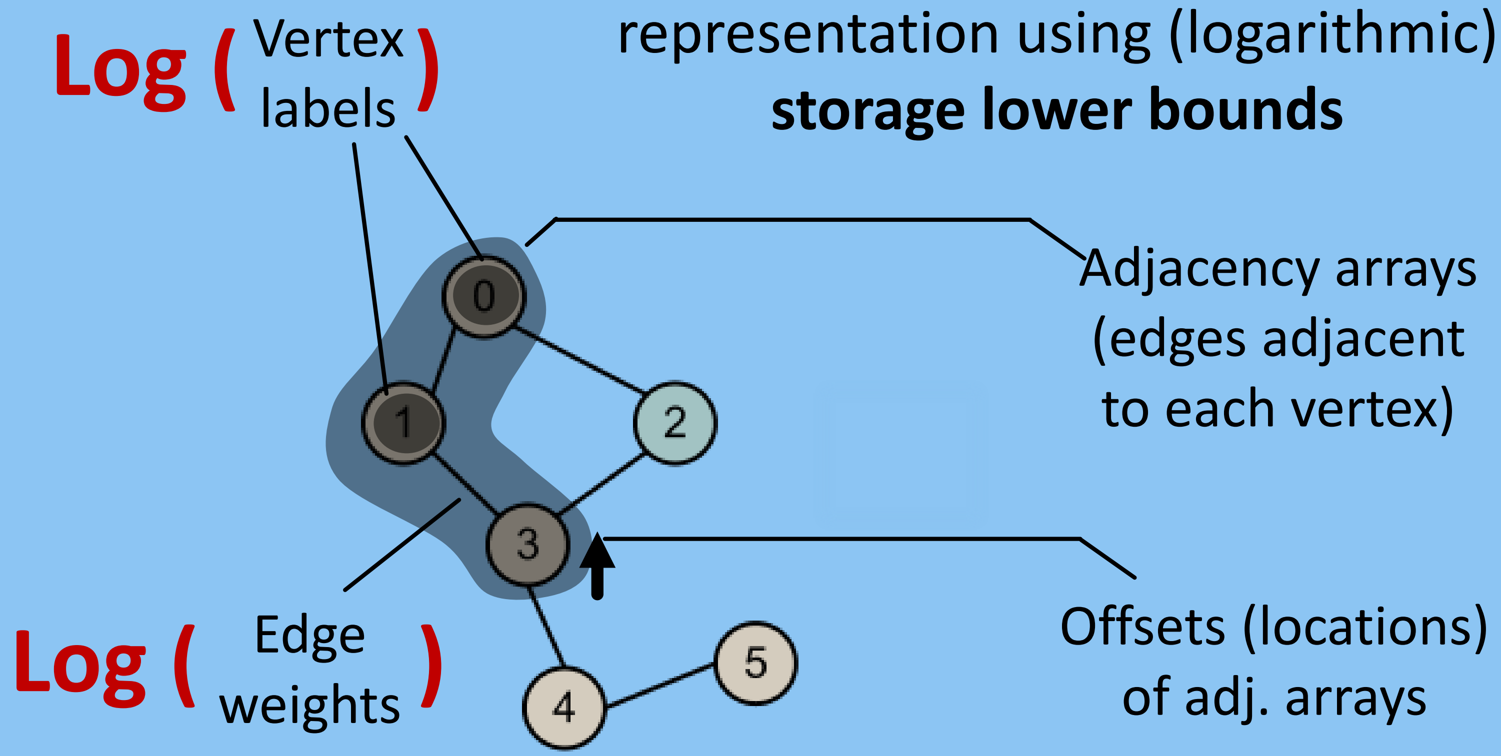
Which one? ☺

Shannon's approach
logarithmic
(one needs at least log|S| bits to store an object from an arbitrary set S)

$$S = \{x_1, x_2, x_3, \dots\} \quad \begin{array}{l} x_1 \rightarrow 0 \dots 01 \\ x_2 \rightarrow 0 \dots 10 \\ x_3 \rightarrow 0 \dots 11 \\ \quad \dots \end{array}$$

Key idea

Encode different parts of a graph representation using (logarithmic) **storage lower bounds**



What is **the lowest storage** we can (hope to) use to store a graph?

$$S = \{x_1, x_2, x_3, \ldots\} \quad \begin{array}{l} x_1 \to 0 \ldots 01 \\ x_2 \to 0 \ldots 10 \\ x_3 \to 0 \ldots 11 \\ \ldots \end{array}$$

The storage **lower bound** Ω

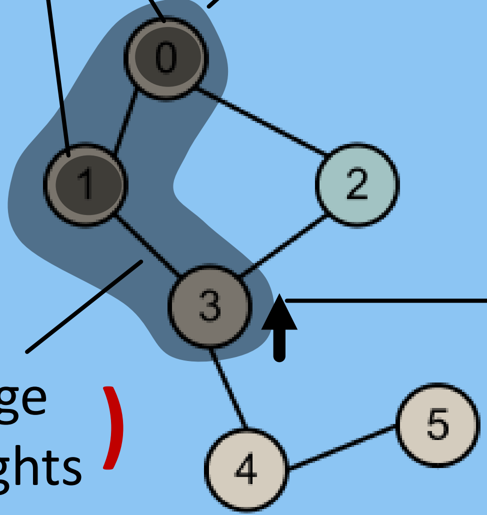
Which one? ☺

Shannon's approach **logarithmic** (one needs at least log|S| bits to store an object from an arbitrary set S)

Key idea

Encode different parts of a graph representation using (logarithmic) **storage lower bounds**

Log (Vertex labels **)**



Log (Adjacency arrays (edges adjacent to each vertex) **)**

Offsets (locations) of adj. arrays

Log (Edge weights **)**

What is **the lowest storage** we can (hope to) use to store a graph?

The storage **lower bound** Ω

Which one? ☺

Shannon's approach **logarithmic** (one needs at least log|S| bits to store an object from an arbitrary set S)

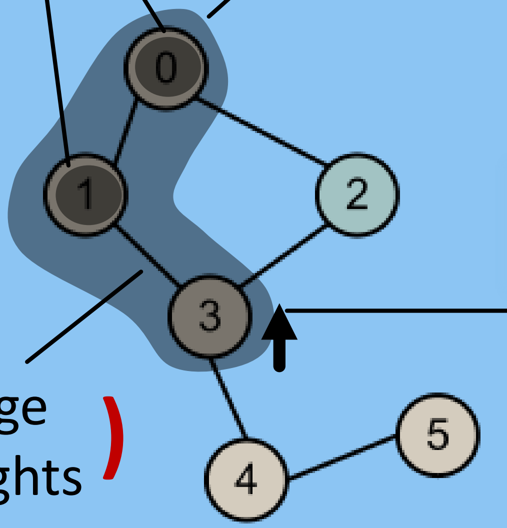
$$S = \{x_1, x_2, x_3, \dots\} \quad \begin{array}{l} x_1 \rightarrow 0 \dots 01 \\ x_2 \rightarrow 0 \dots 10 \\ x_3 \rightarrow 0 \dots 11 \\ \quad \dots \end{array}$$

Key idea

Encode different parts of a graph representation using (logarithmic) **storage lower bounds**

Log (Vertex labels **)**

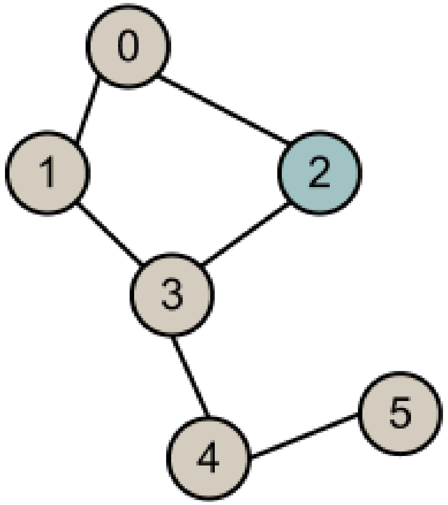
Log (Edge weights **)**



Log (Adjacency arrays (edges adjacent to each vertex) **)**

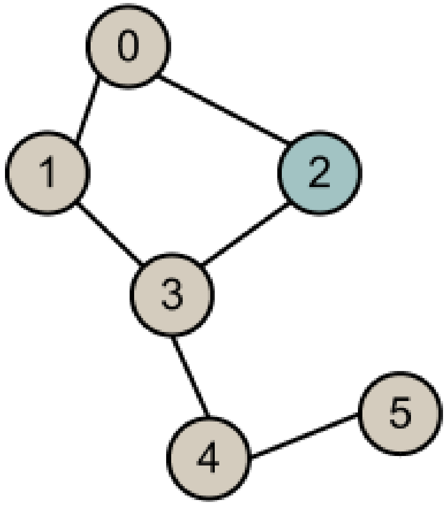
Log (Offsets (locations) of adj. arrays **)**

ADJACENCY ARRAY GRAPH REPRESENTATION



M. Besta et al.: "Log(Graph): A Near-Optimal High-Performance Graph Representation", PACT'18

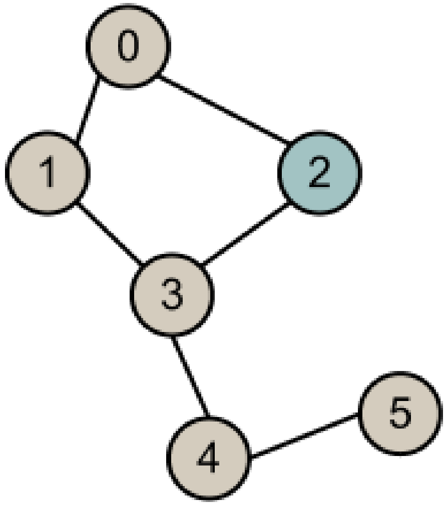
Adjacency Array Graph Representation



Representation

M. Besta et al.: "Log(Graph): A Near-Optimal High-Performance Graph Representation", PACT'18

Adjacency Array Graph Representation



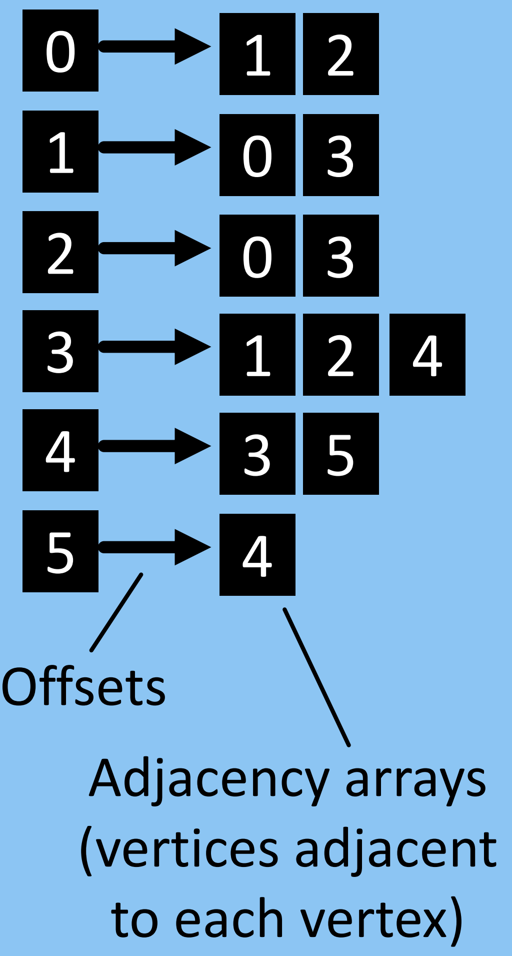
Representation

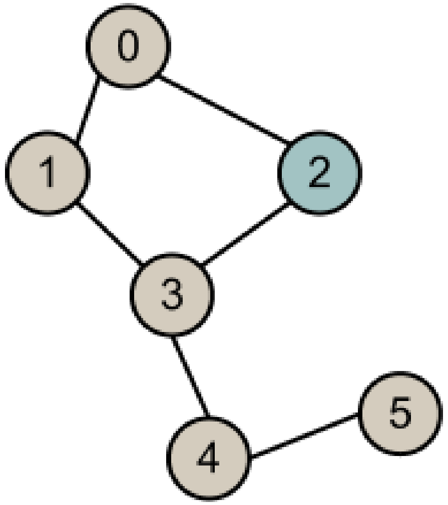
0
1
2
3
4
5

M. Besta et al.: “Log(Graph): A Near-Optimal High-Performance Graph Representation”, PACT’18

ADJACENCY ARRAY GRAPH REPRESENTATION

Representation

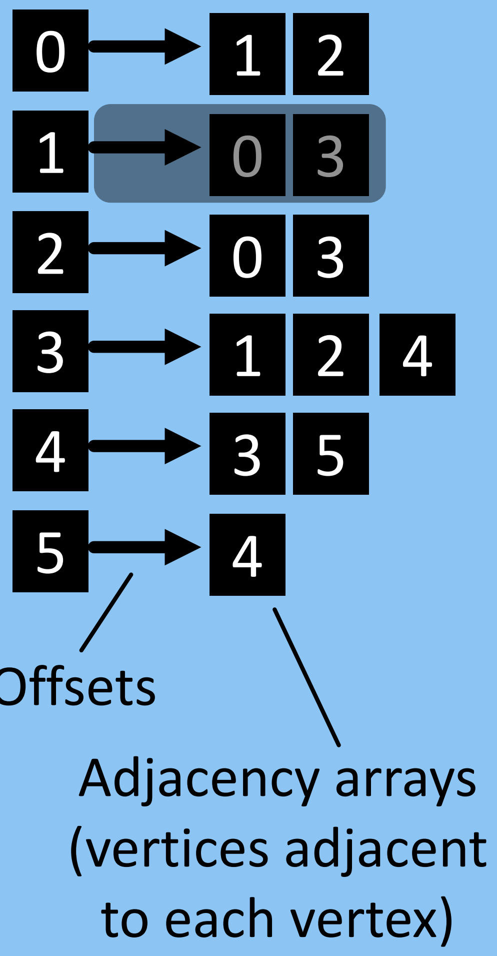


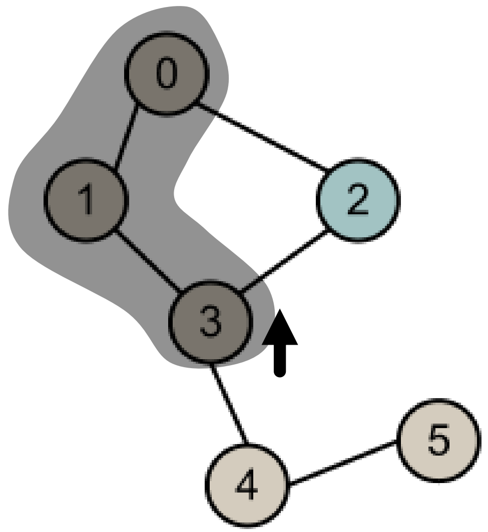


M. Besta et al.: “Log(Graph): A Near-Optimal High-Performance Graph Representation”, PACT’18

Adjacency Array Graph Representation

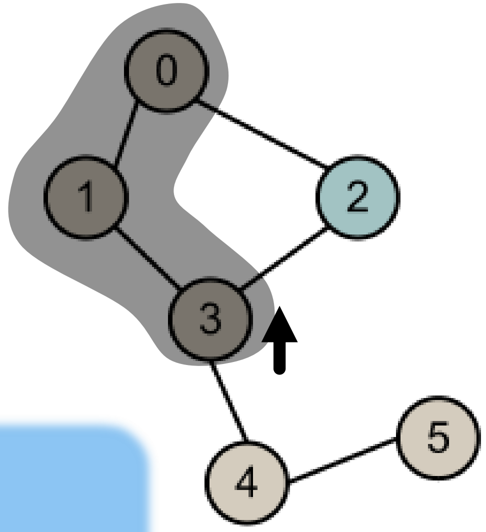
Representation



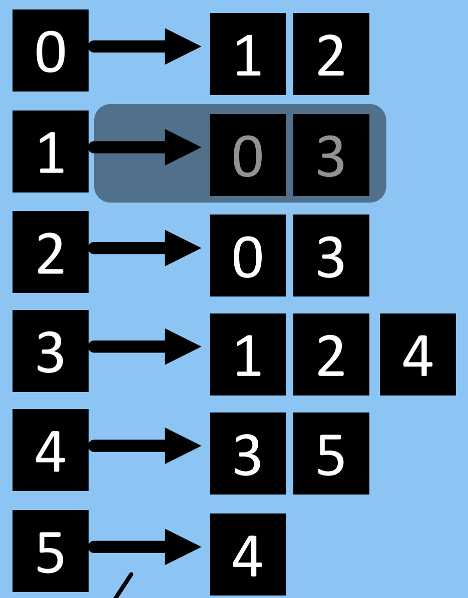


M. Besta et al.: “Log(Graph): A Near-Optimal High-Performance Graph Representation”, PACT’18

Adjacency Array Graph Representation



Representation



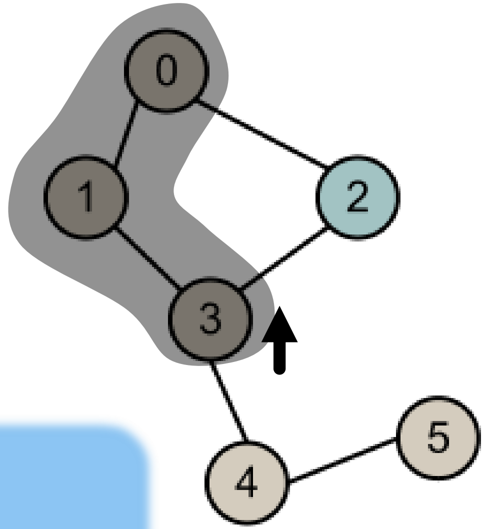
Offsets

Adjacency arrays (vertices adjacent to each vertex)

Physical realization

M. Besta et al.: "Log(Graph): A Near-Optimal High-Performance Graph Representation", PACT'18

Adjacency Array Graph Representation



Representation

0 → 1 2
1 → 0 3
2 → 0 3
3 → 1 2 4
4 → 3 5
5 → 4

Offsets

Adjacency arrays (vertices adjacent to each vertex)

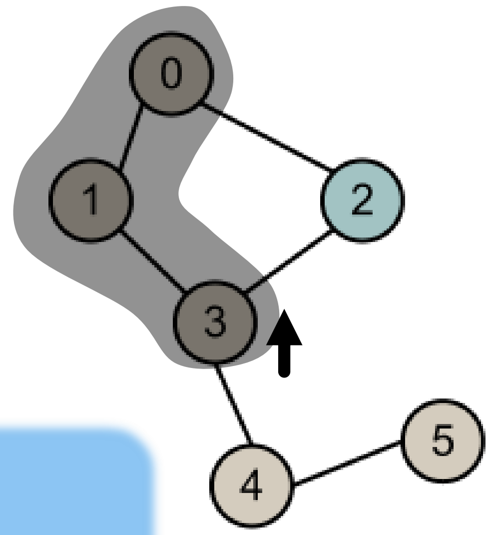
Physical realization

Adjacency arrays (one contiguous array)

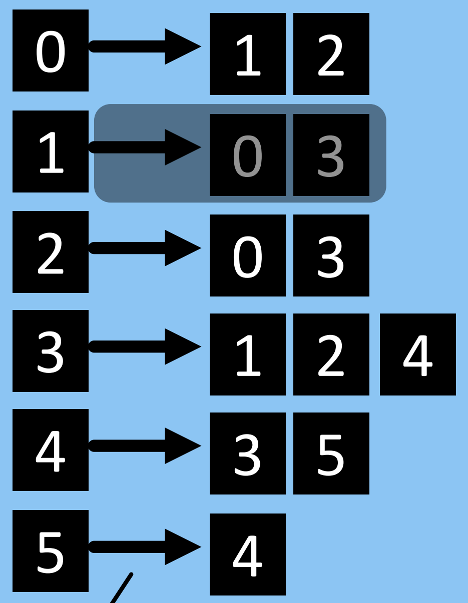
1 2 | 0 3 | 0 3 | 1 2 4 | 3 5 | 4

M. Besta et al.: “Log(Graph): A Near-Optimal High-Performance Graph Representation”, PACT’18

Adjacency Array Graph Representation



Representation

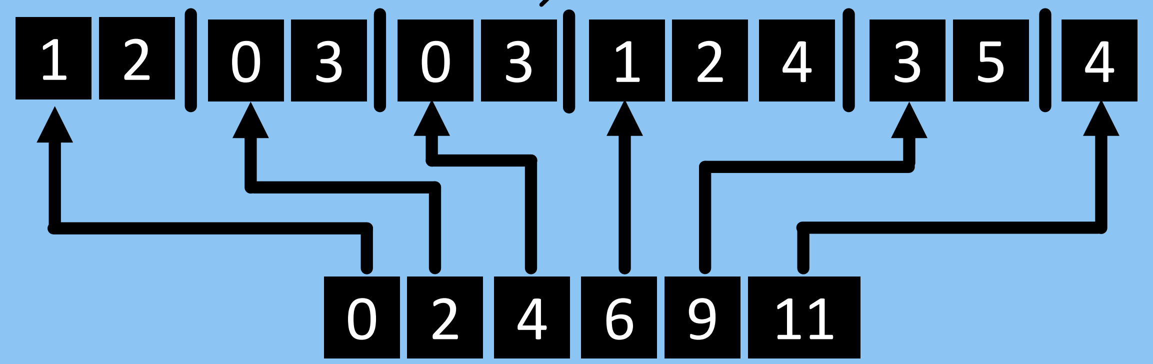


Offsets

Adjacency arrays (vertices adjacent to each vertex)

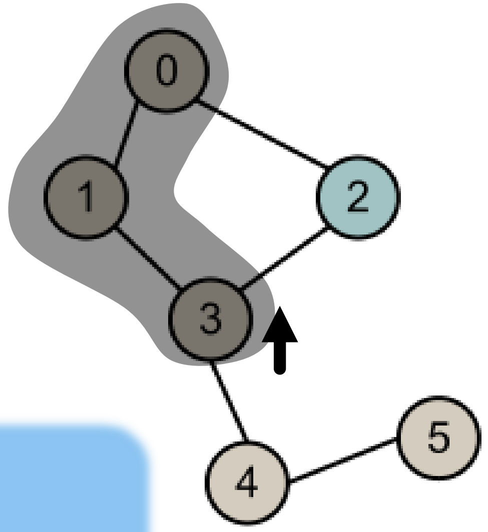
Physical realization

Adjacency arrays (one contiguous array)

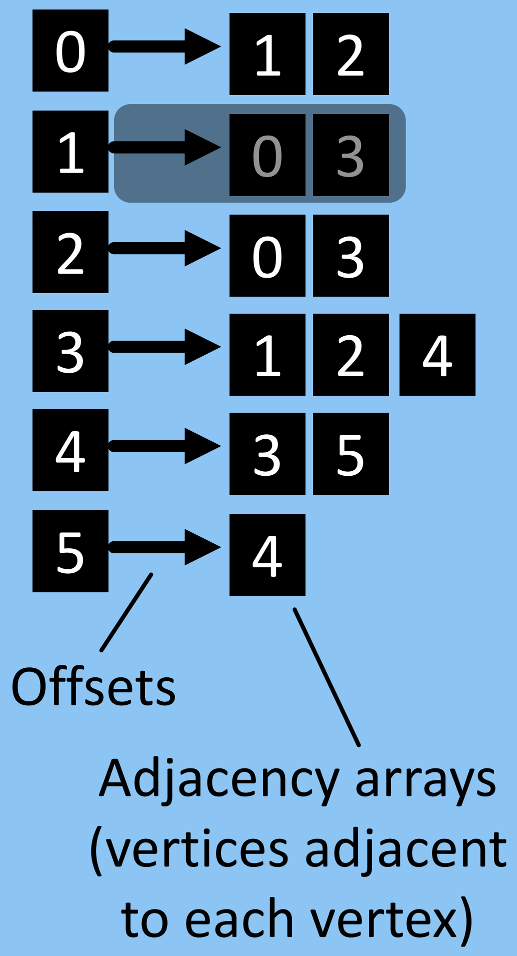


Offsets (another contiguous array)

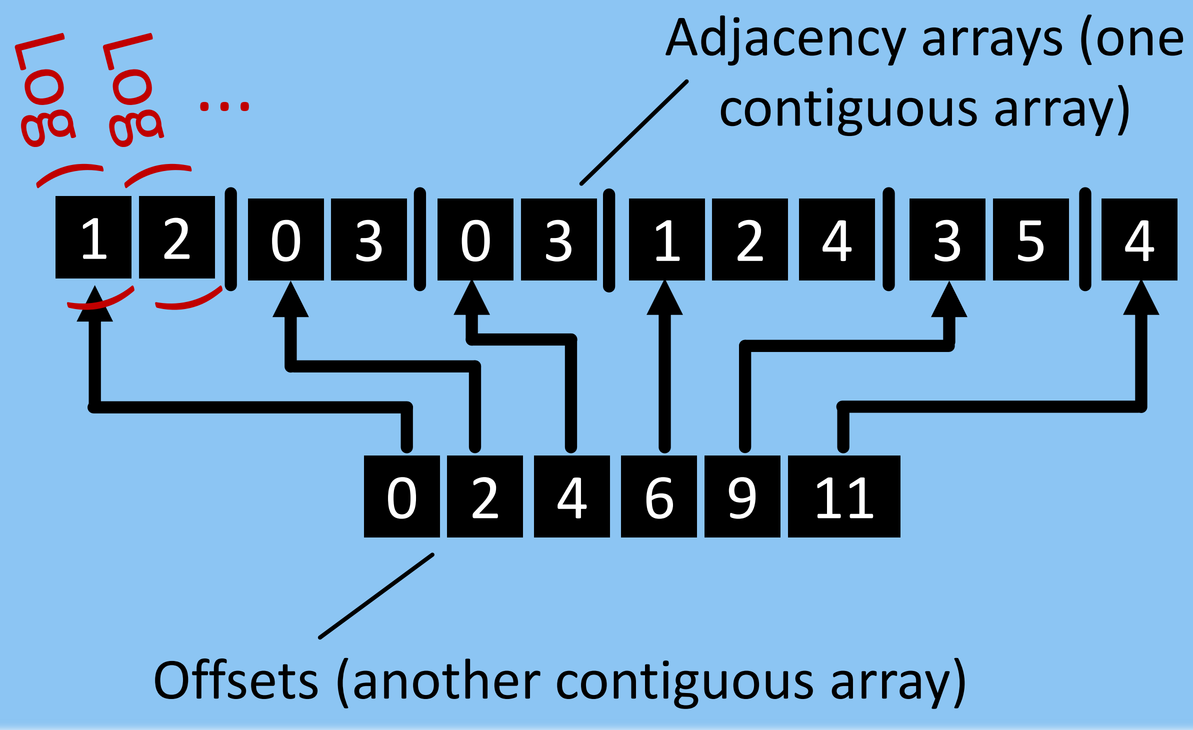
ADJACENCY ARRAY GRAPH REPRESENTATION



Representation

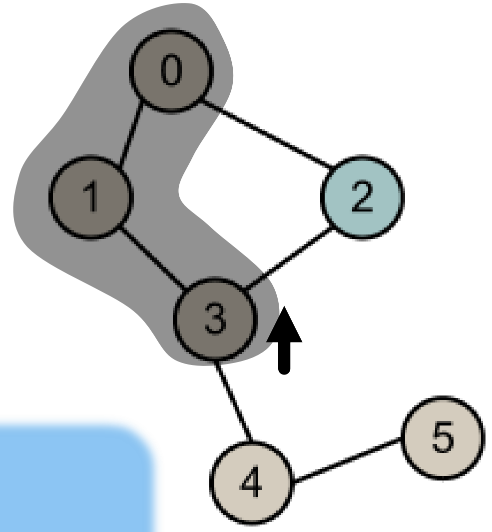


Physical realization

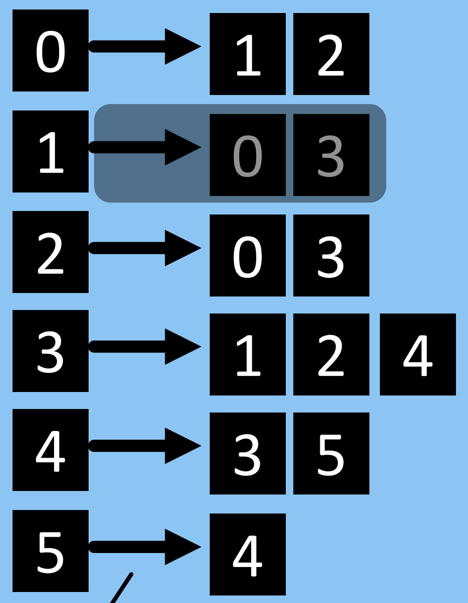


M. Besta et al.: "Log(Graph): A Near-Optimal High-Performance Graph Representation", PACT'18

Adjacency Array Graph Representation



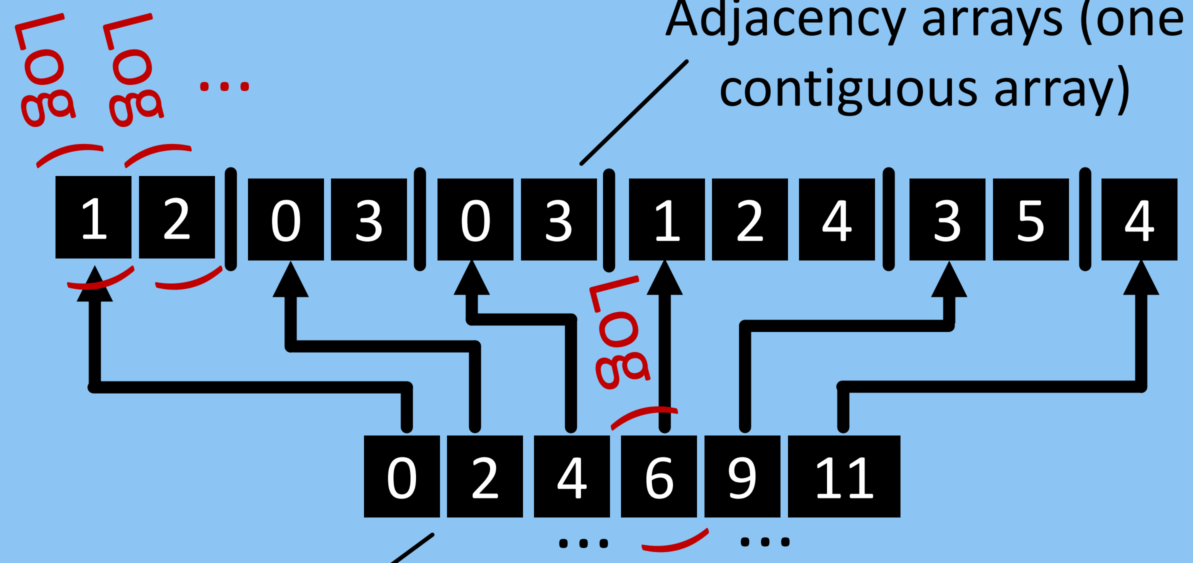
Representation



Offsets

Adjacency arrays (vertices adjacent to each vertex)

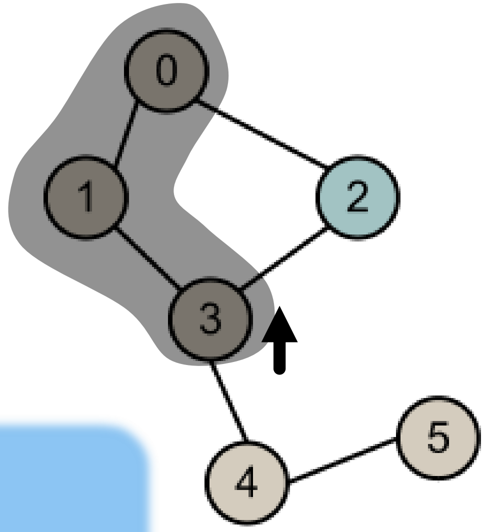
Physical realization



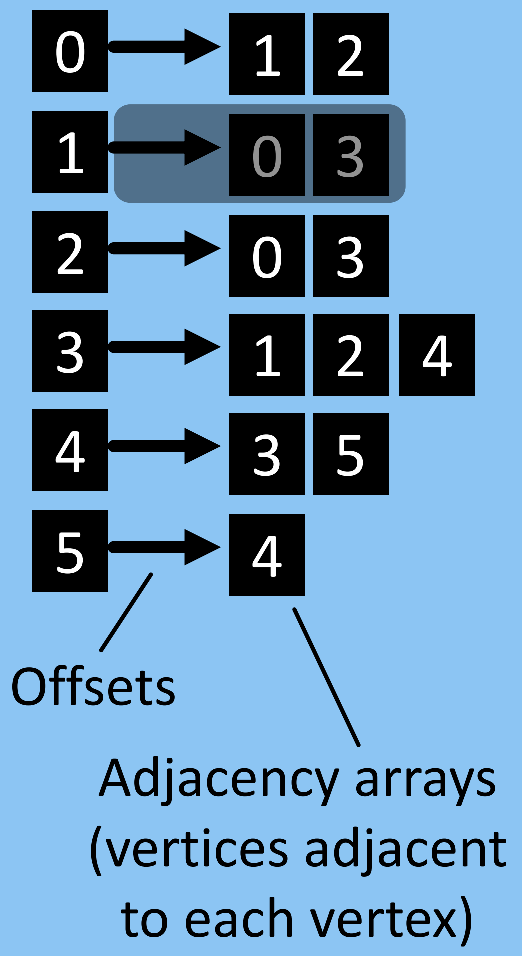
Adjacency arrays (one contiguous array)

Offsets (another contiguous array)

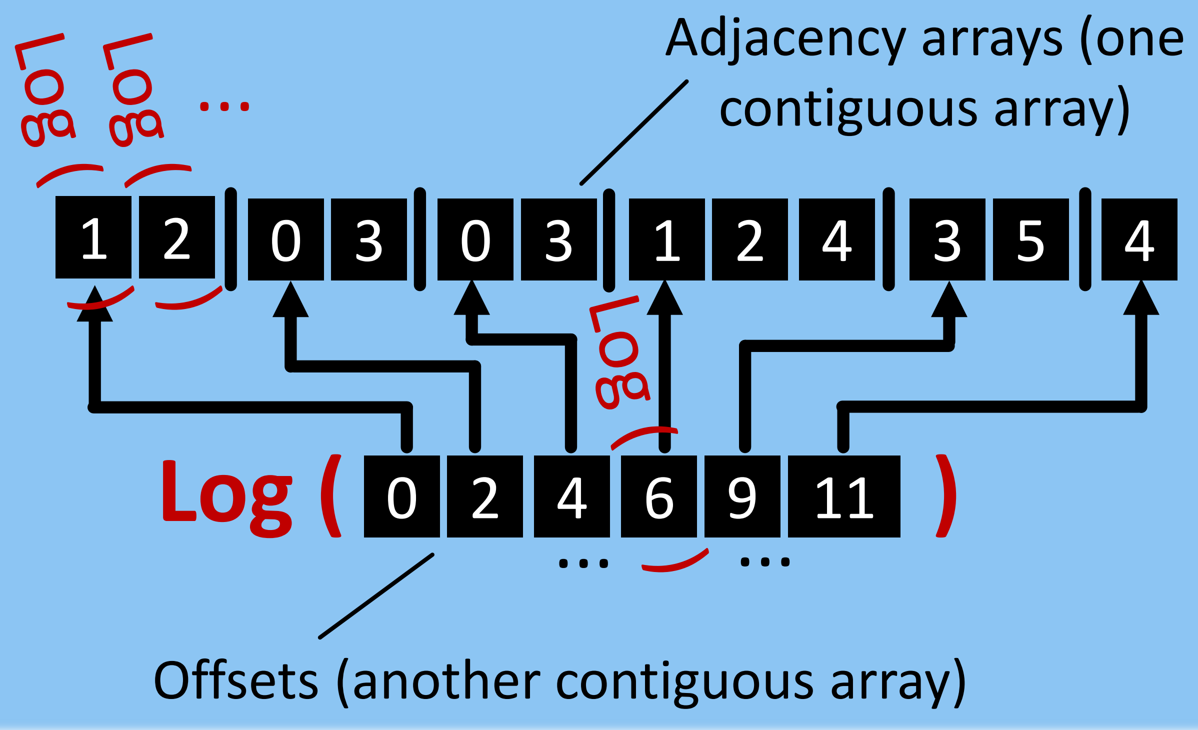
Adjacency Array Graph Representation



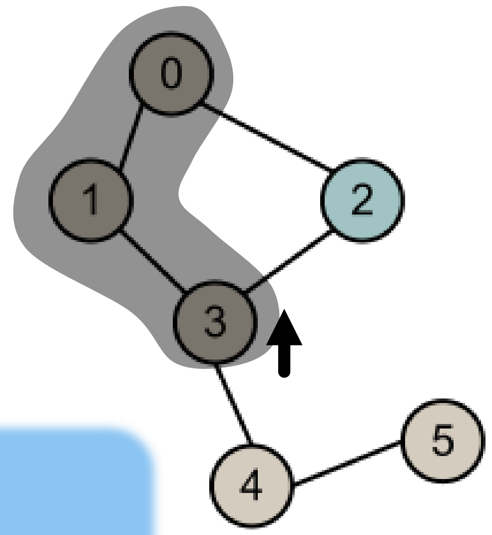
Representation



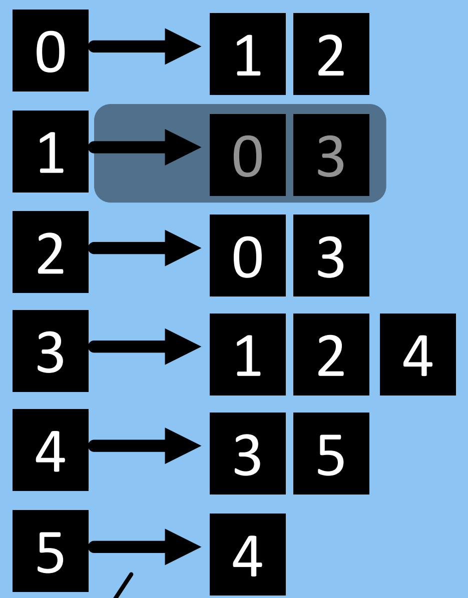
Physical realization



Adjacency Array Graph Representation



Representation

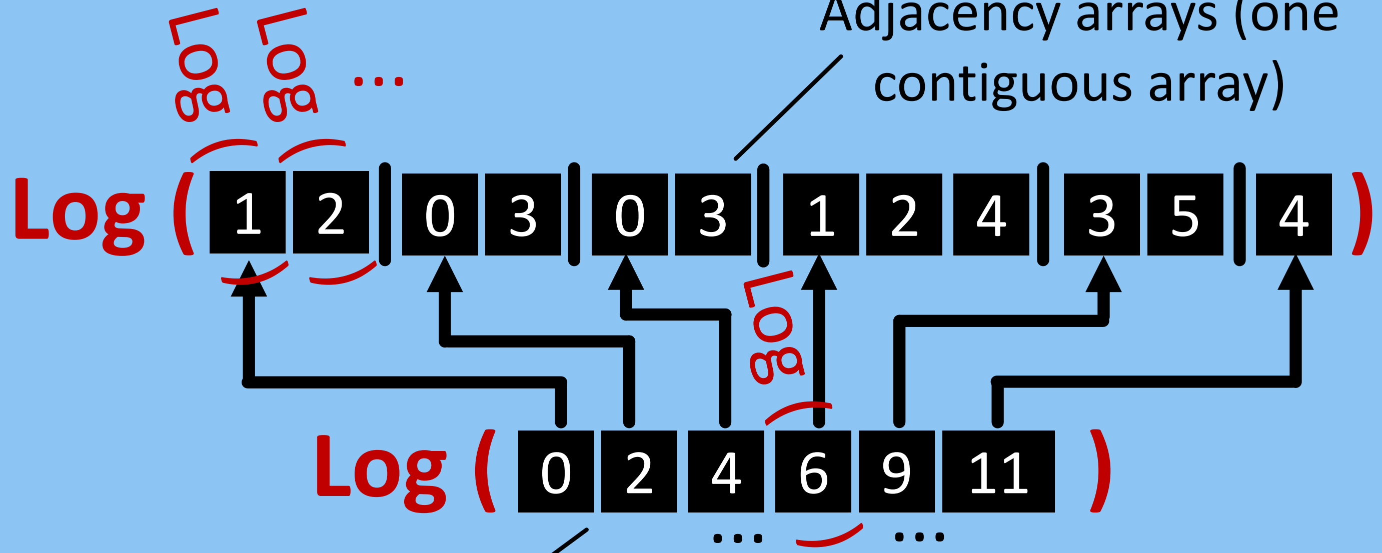


Offsets

Adjacency arrays (vertices adjacent to each vertex)

Physical realization

Adjacency arrays (one contiguous array)

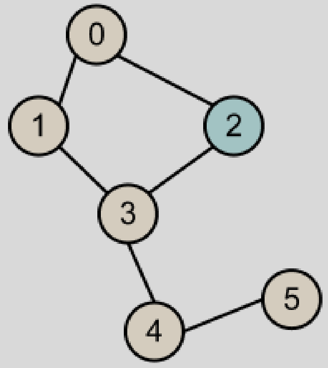


Offsets (another contiguous array)

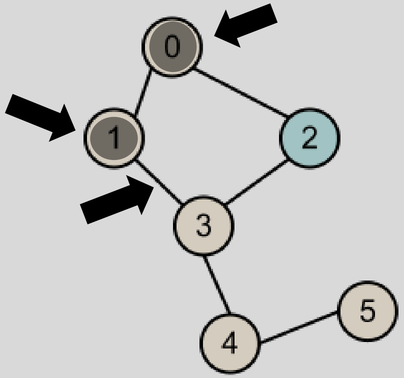
1 **Log (Vertex labels), Log (Edge weights)**

M. Besta et al.: “Log(Graph): A Near-Optimal High-Performance Graph Representation”, PACT’18

1 **Log (** Vertex labels **), Log (** Edge weights **)**



1 **Log** (Vertex labels), **Log** (Edge weights)

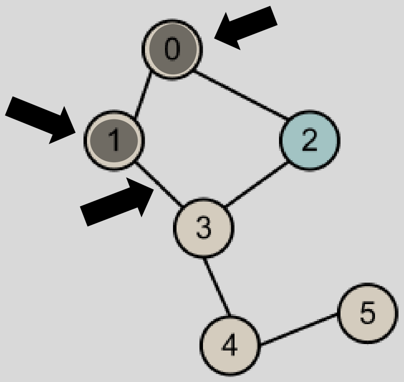


M. Besta et al.: "Log(Graph): A Near-Optimal High-Performance Graph Representation", PACT'18

1 **Log (Vertex labels), Log (Edge weights)**

Symbols

n : #vertices,
m : #edges,
d_v : degree of vertex v,
N_v : neighbors (adj. array) of vertex v,
$\widehat{N_v}$: maximum among N_v



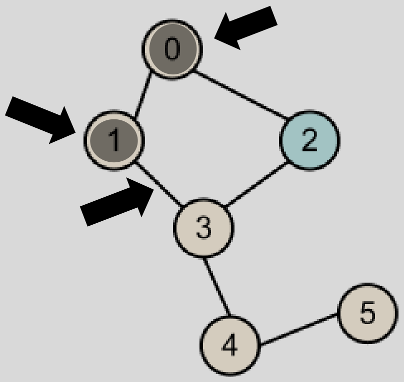
M. Besta et al.: "Log(Graph): A Near-Optimal High-Performance Graph Representation", PACT'18

1 **Log (Vertex labels), Log (Edge weights)**

Lower bounds (global)

Symbols

n : #vertices,
m : #edges,
d_v : degree of vertex v,
N_v : neighbors (adj. array) of vertex v,
$\widehat{N_v}$: maximum among N_v



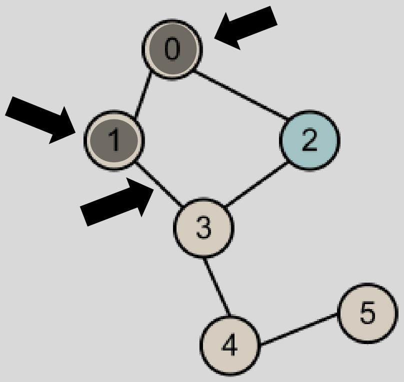
M. Besta et al.: "Log(Graph): A Near-Optimal High-Performance Graph Representation", PACT'18

1 **Log (** Vertex labels **), Log (** Edge weights **)**

Lower bounds (global)

$$\lceil \log n \rceil$$

Symbols



n : #vertices,
m : #edges,
d_v : degree of vertex v,
N_v : neighbors (adj. array) of vertex v,
$\widehat{N_v}$: maximum among N_v

M. Besta et al.: "Log(Graph): A Near-Optimal High-Performance Graph Representation", PACT'18

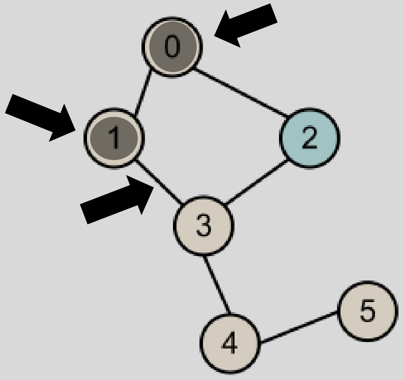
1 **Log (** Vertex labels **), Log (** Edge weights **)**

Lower bounds (global)

$$\lceil \log n \rceil$$

This is it?
Not really ☺

Symbols

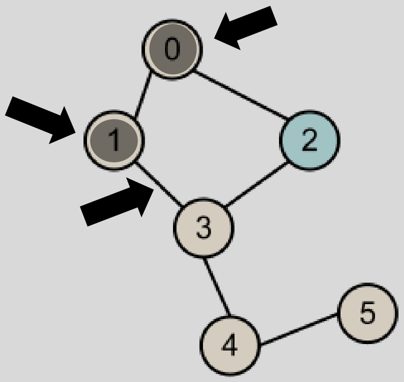


n : #vertices,
m : #edges,
d_v : degree of vertex v,
N_v : neighbors (adj. array) of vertex v,
$\widehat{N_v}$: maximum among N_v

1 **Log (** Vertex labels **), Log (** Edge weights **)**

Symbols

n : #vertices,
m : #edges,
d_v : degree of vertex v,
N_v : neighbors (adj. array) of vertex v,
$\widehat{N_v}$: maximum among N_v



Lower bounds (global)

$$\lceil \log n \rceil$$

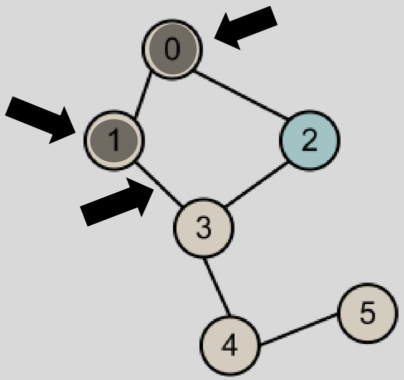
This is it?
Not really ☺

Lower bounds (local)

M. Besta et al.: "Log(Graph): A Near-Optimal High-Performance Graph Representation", PACT'18

1 Log (Vertex labels), Log (Edge weights)

Symbols



n : #vertices,
m : #edges,
d_v : degree of vertex v,
N_v : neighbors (adj. array) of vertex v,
$\widehat{N_v}$: maximum among N_v

Lower bounds (global)

$$\lceil \log n \rceil$$

This is it?
Not really ☺

Lower bounds (local)

Assume:

M. Besta et al.: “Log(Graph): A Near-Optimal High-Performance Graph Representation”, PACT’18

1 Log (Vertex labels), Log (Edge weights)

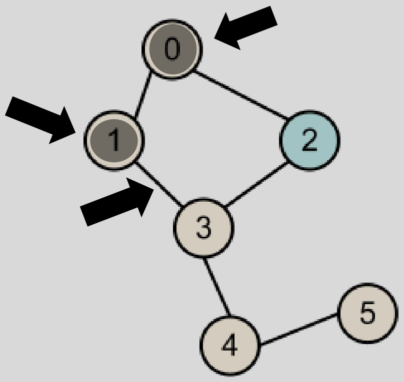
Lower bounds (global)

$\lceil \log n \rceil$

This is it?
Not really ☺

Symbols

n : #vertices,
m : #edges,
d_v : degree of vertex v,
N_v : neighbors (adj. array) of vertex v,
$\widehat{N_v}$: maximum among N_v



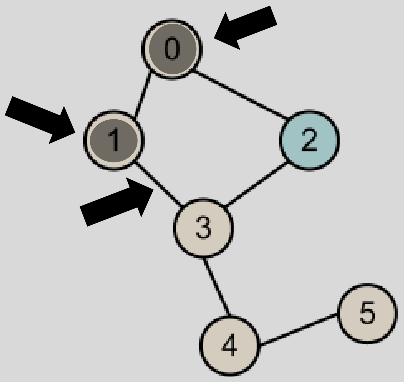
Lower bounds (local)

Assume:

- a graph, e.g., $V = \{1, \dots, 2^{22}\}$

1 Log (Vertex labels), Log (Edge weights)

Symbols



n : #vertices,
m : #edges,
d_v : degree of vertex v,
N_v : neighbors (adj. array) of vertex v,
$\widehat{N_v}$: maximum among N_v

Lower bounds (global)

$$\lceil \log n \rceil$$

This is it?
Not really ☺

Lower bounds (local)

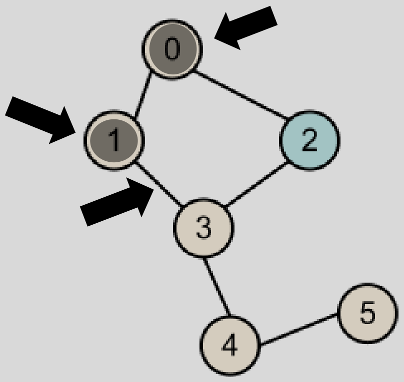
Assume:

- a graph, e.g., $V = \{1, \dots, 2^{22}\}$
- A vertex v with few neighbors: $d_v \ll n$

1 Log (Vertex labels), Log (Edge weights)

Symbols

n : #vertices,
m : #edges,
d_v : degree of vertex v,
N_v : neighbors (adj. array) of vertex v,
$\widehat{N_v}$: maximum among N_v



Lower bounds (global)

$$\lceil \log n \rceil$$

This is it?
Not really ☺

Lower bounds (local)

Assume:

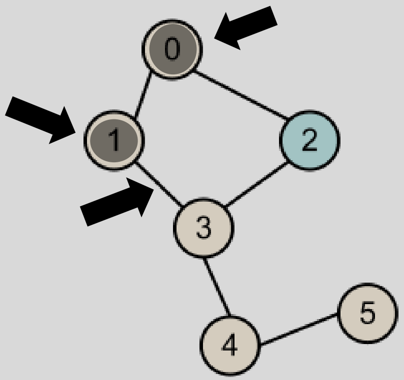
- a graph, e.g., $V = \{1, \dots, 2^{22}\}$
- A vertex v with few neighbors: $d_v \ll n$
- ...all these neighbors have small labels: $\widehat{N_v} \ll n$

M. Besta et al.: "Log(Graph): A Near-Optimal High-Performance Graph Representation", PACT'18

1 Log (Vertex labels), Log (Edge weights)

Symbols

n : #vertices,
m : #edges,
d_v : degree of vertex v,
N_v : neighbors (adj. array) of vertex v,
$\widehat{N_v}$: maximum among N_v



Lower bounds (global)

$\lceil \log n \rceil$

This is it?
Not really ☺

Lower bounds (local)

Assume:

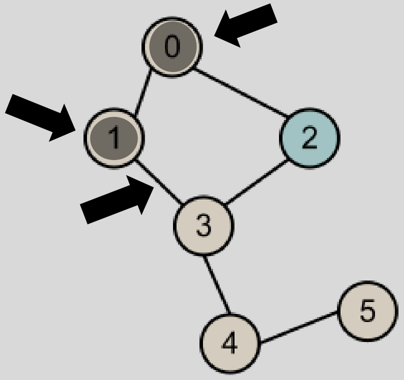
- a graph, e.g., $V = \{1, \dots, 2^{22}\}$
- A vertex v with few neighbors: $d_v \ll n$
- ...all these neighbors have small labels: $\widehat{N_v} \ll n$

v → | 2 | 3 | 4 | 5 |

M. Besta et al.: "Log(Graph): A Near-Optimal High-Performance Graph Representation", PACT'18

1 Log (Vertex labels), Log (Edge weights)

Symbols



n : #vertices,
m : #edges,
d_v : degree of vertex v,
N_v : neighbors (adj. array) of vertex v,
$\widehat{N_v}$: maximum among N_v

Lower bounds (global)

$$\lceil \log n \rceil$$

This is it?
Not really ☺

Lower bounds (local)

Assume:

- a graph, e.g., $V = \{1, \ldots, 2^{22}\}$
- A vertex v with few neighbors: $d_v \ll n$
- ...all these neighbors have small labels: $\widehat{N_v} \ll n$

$$\lceil \log 2^{22} \rceil = 22$$

$v \rightarrow$ | 2 | 3 | 4 | 5 |

M. Besta et al.: “Log(Graph): A Near-Optimal High-Performance Graph Representation”, PACT’18

1 **Log (Vertex labels), Log (Edge weights)**

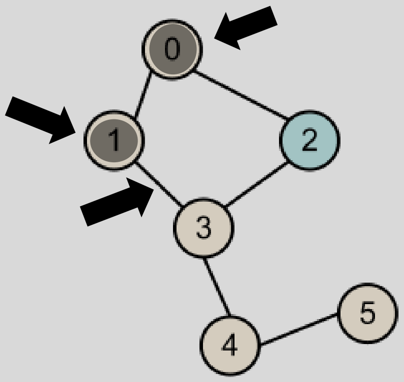
Lower bounds (global)

$\lceil \log n \rceil$

This is it?
Not really ☺

Symbols

n : #vertices,
m : #edges,
d_v : degree of vertex v,
N_v : neighbors (adj. array) of vertex v,
$\widehat{N_v}$: maximum among N_v



Lower bounds (local)

Assume:
- a graph, e.g., $V = \{1, \ldots, 2^{22}\}$
- A vertex v with few neighbors: $d_v \ll n$
- ...all these neighbors have small labels: $\widehat{N_v} \ll n$

$\lceil \log 2^{22} \rceil = 22$

v → | 0...10 | 0...11 | 0...100 | 0...101 |

v → | 2 | 3 | 4 | 5 |

M. Besta et al.: "Log(Graph): A Near-Optimal High-Performance Graph Representation", PACT'18

1 Log (Vertex labels), Log (Edge weights)

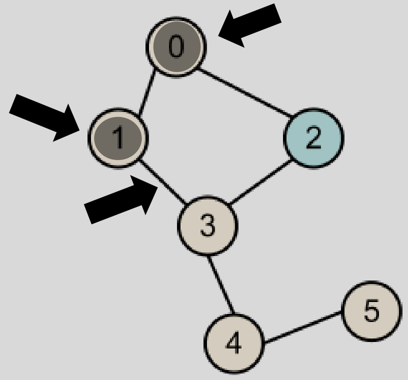
Lower bounds (global)

$\lceil \log n \rceil$

This is it?
Not really ☺

Symbols

n : #vertices,
m : #edges,
d_v : degree of vertex v,
N_v : neighbors (adj. array) of vertex v,
$\widehat{N_v}$: maximum among N_v



Lower bounds (local)

Assume:
- a graph, e.g., $V = \{1, \dots, 2^{22}\}$
- A vertex v with few neighbors: $d_v \ll n$
- ...all these neighbors have small labels: $\widehat{N_v} \ll n$

v → | 2 | 3 | 4 | 5 |

$\lceil \log 2^{22} \rceil = 22$

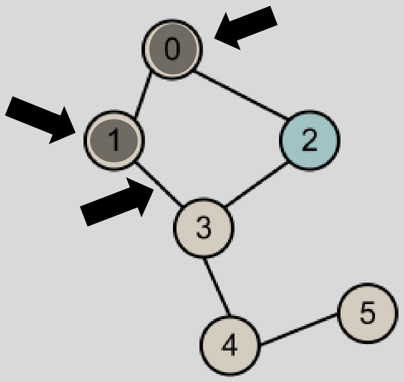
v → | 0...10 | 0...11 |
| 0...100 | 0...101 |

19 zeros!

M. Besta et al.: "Log(Graph): A Near-Optimal High-Performance Graph Representation", PACT'18

1 Log (Vertex labels), Log (Edge weights)

Symbols



n : #vertices,
m : #edges,
d_v : degree of vertex v,
N_v : neighbors (adj. array) of vertex v,
$\widehat{N_v}$: maximum among N_v

Lower bounds (global)

$\lceil \log n \rceil$

This is it?
Not really ☺

Lower bounds (local)

Assume:

- a graph, e.g., $V = \{1, \ldots, 2^{22}\}$
- A vertex v with few neighbors: $d_v \ll n$
- ...all these neighbors have small labels: $\widehat{N_v} \ll n$

v → | 2 | 3 | 4 | 5 |

$\lceil \log 2^{22} \rceil = 22$

v → | 0...10 | 0...11 |
| 0...100 | 0...101 |

19 zeros!

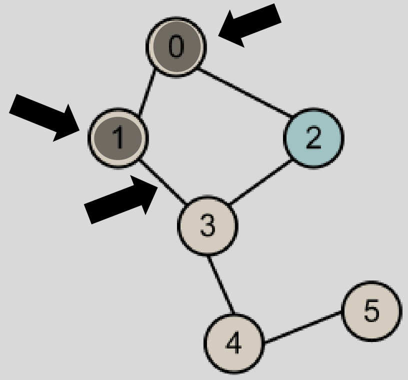
Thus, use the local bound $\lceil \log \widehat{N_v} \rceil$

M. Besta et al.: “Log(Graph): A Near-Optimal High-Performance Graph Representation”, PACT’18

1 **Log (** Vertex labels **), Log (** Edge weights **)**

This is it?
Not really ☺

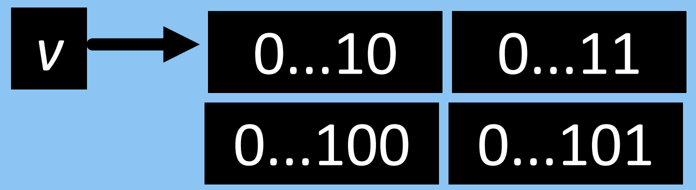
Symbols



n : #vertices,
m : #edges,
d_v : degree of vertex v,
N_v : neighbors (adj. array) of vertex v,
$\widehat{N_v}$: maximum among N_v

Lower bounds (local): problem

- a graph, e.g., $V = \{1, \ldots, 2^{22}\}$
- A vertex v with few neighbors: $d_v \ll n$
- ...all these neighbors have small labels: $\widehat{N_v} \ll n$

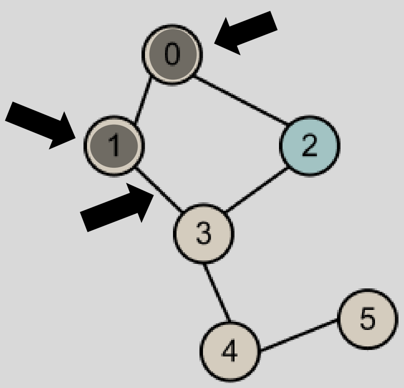


1 Log (Vertex labels), Log (Edge weights)

This is it?
Not really ☺

Symbols

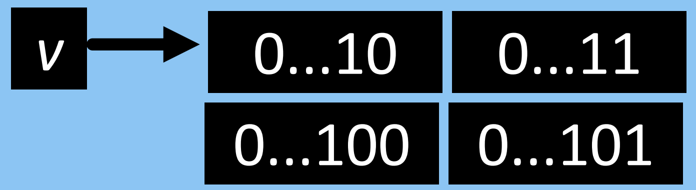
n : #vertices,
m : #edges,
d_v : degree of vertex v,
N_v : neighbors (adj. array) of vertex v,
$\widehat{N_v}$: maximum among N_v



Lower bounds (local): problem

What if:

- a graph, e.g., $V = \{1, \ldots, 2^{22}\}$
- A vertex v with few neighbors: $d_v \ll n$
- ...all these neighbors have small labels: $\widehat{N_v} \ll n$

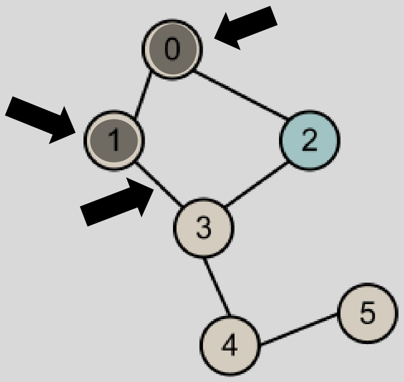


1 Log (Vertex labels), Log (Edge weights)

This is it?
Not really ☺

Symbols

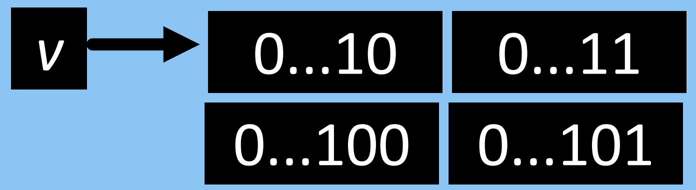
n : #vertices,
m : #edges,
d_v : degree of vertex v,
N_v : neighbors (adj. array) of vertex v,
$\widehat{N_v}$: maximum among N_v



Lower bounds (local): problem

What if:

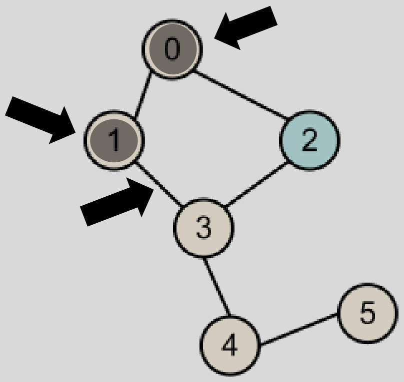
- a graph, e.g., $V = \{1, \dots, 2^{22}\}$
- A vertex v with few neighbors: $d_v \ll n$
- ...all these neighbors have small labels: $\widehat{N_v} \ll n$
- ...one neighbor has a large ID:



1 Log (Vertex labels), Log (Edge weights)

Symbols

n : #vertices,
m : #edges,
d_v : degree of vertex v,
N_v : neighbors (adj. array) of vertex v,
$\widehat{N_v}$: maximum among N_v



This is it?
Not really ☺

Lower bounds (local): problem

What if:

- a graph, e.g., $V = \{1, \dots, 2^{22}\}$
- A vertex v with few neighbors: $d_v \ll n$
- ...all these neighbors have small labels: $\widehat{N_v} \ll n$
- ...one neighbor has a large ID:

v → | 2 | 3 | 4 | 5 | 1M |

v → | 0...10 | 0...11 |
| 0...100 | 0...101 |

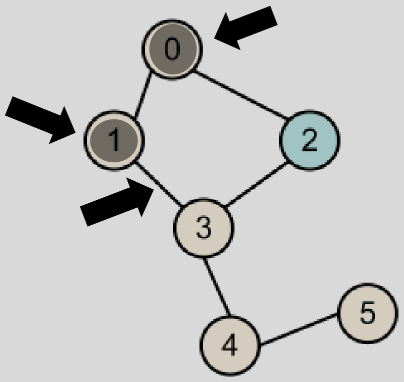
M. Besta et al.: "Log(Graph): A Near-Optimal High-Performance Graph Representation", PACT'18

1 Log (Vertex labels), Log (Edge weights)

This is it?
Not really ☺

Symbols

n : #vertices,
m : #edges,
d_v : degree of vertex v,
N_v : neighbors (adj. array) of vertex v,
$\widehat{N_v}$: maximum among N_v



Lower bounds (local): problem

What if:

- a graph, e.g., $V = \{1, \dots, 2^{22}\}$
- A vertex v with few neighbors: $d_v \ll n$
- ...all these neighbors have small labels: $\widehat{N_v} \ll n$
- ...one neighbor has a large ID:

$v \rightarrow$ | 2 | 3 | 4 | 5 | 1M |

$$\lceil \log 2^{20} \rceil = 20$$

$v \rightarrow$ | 0...10 | 0...11 | 0...100 | 0...101 |

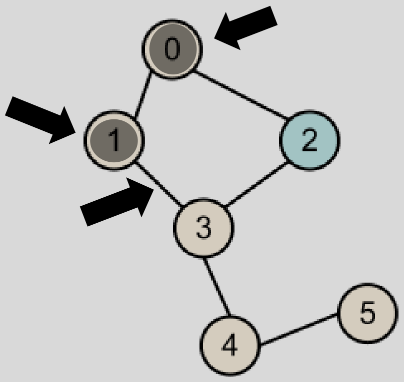
M. Besta et al.: “Log(Graph): A Near-Optimal High-Performance Graph Representation”, PACT’18

1 Log (Vertex labels), Log (Edge weights)

This is it?
Not really ☺

Symbols

n : #vertices,
m : #edges,
d_v : degree of vertex v,
N_v : neighbors (adj. array) of vertex v,
$\widehat{N_v}$: maximum among N_v



Lower bounds (local): problem

What if:

- a graph, e.g., $V = \{1, \dots, 2^{22}\}$
- A vertex v with few neighbors: $d_v \ll n$
- ...all these neighbors have small labels: $\widehat{N_v} \ll n$
- ...one neighbor has a large ID:

$v \rightarrow$ | 2 | 3 | 4 | 5 | 1M |

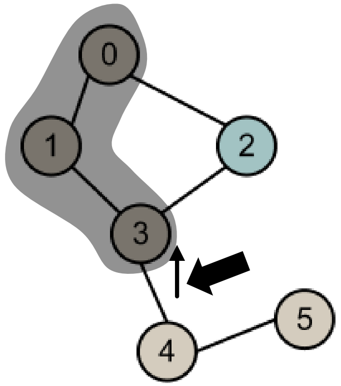
$\lceil \log 2^{20} \rceil = 20$

$v \rightarrow$ | 0...10 | 0...11 |
| 0...100 | 0...101 |

17 zeros!

2 Log (Offset structure)

2 Log (Offset structure)

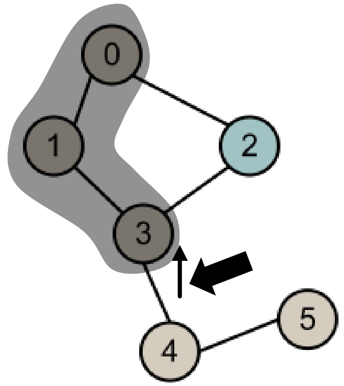


M. Besta et al.: "Log(Graph): A Near-Optimal High-Performance Graph Representation", PACT'18

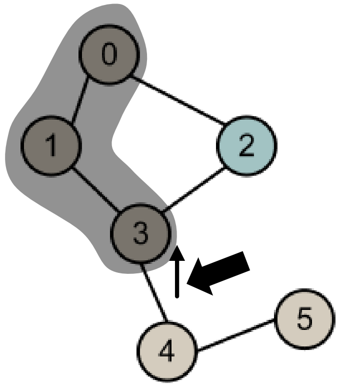
2 Log (Offset structure)

Use a **bit vector** instead of an array of offsets...

!



2 Log (Offset structure)

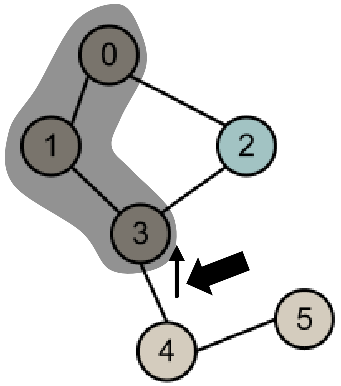


Use a **bit vector** instead of an array of offsets... !

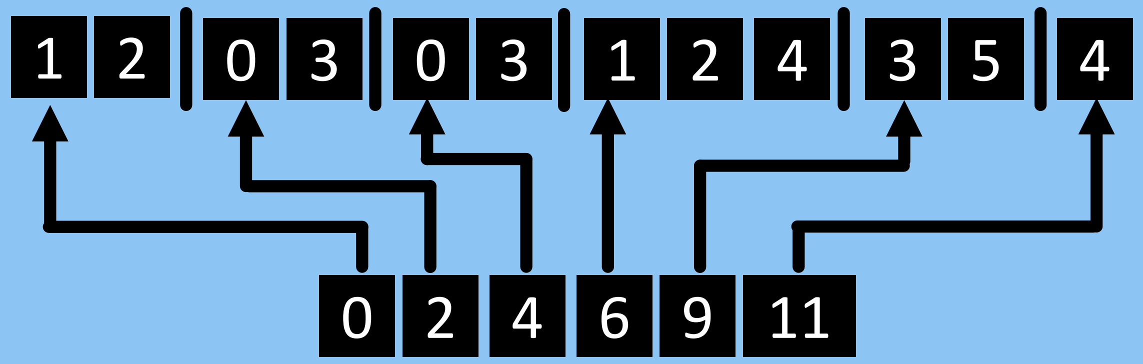
Bit vectors instead of offset arrays

2 Log (Offset structure)

Use a **bit vector** instead of an array of offsets... !



Bit vectors instead of offset arrays

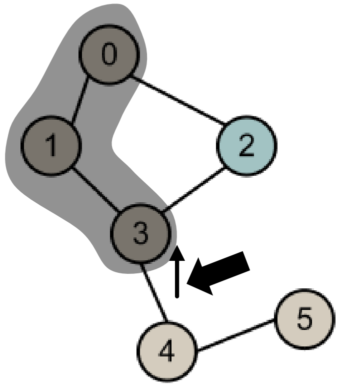


M. Besta et al.: "Log(Graph): A Near-Optimal High-Performance Graph Representation", PACT'18

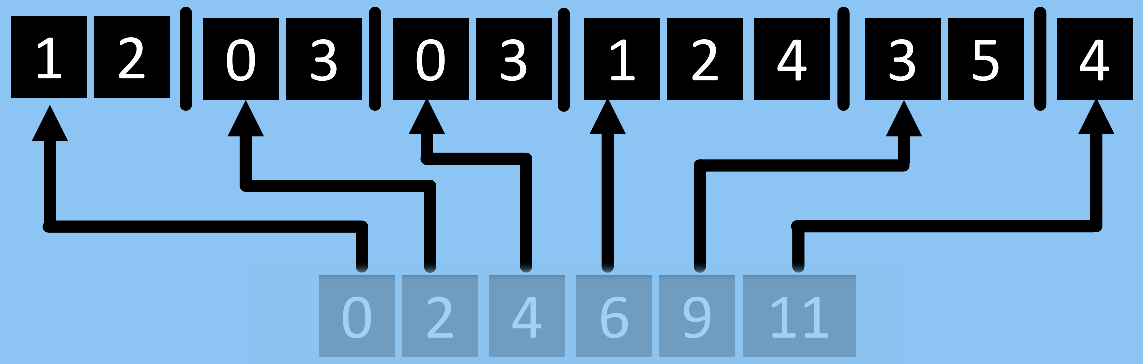
2 Log (Offset structure)

Use a **bit vector** instead of an array of offsets...

!



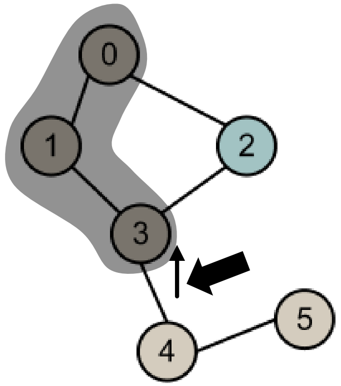
Bit vectors instead of offset arrays



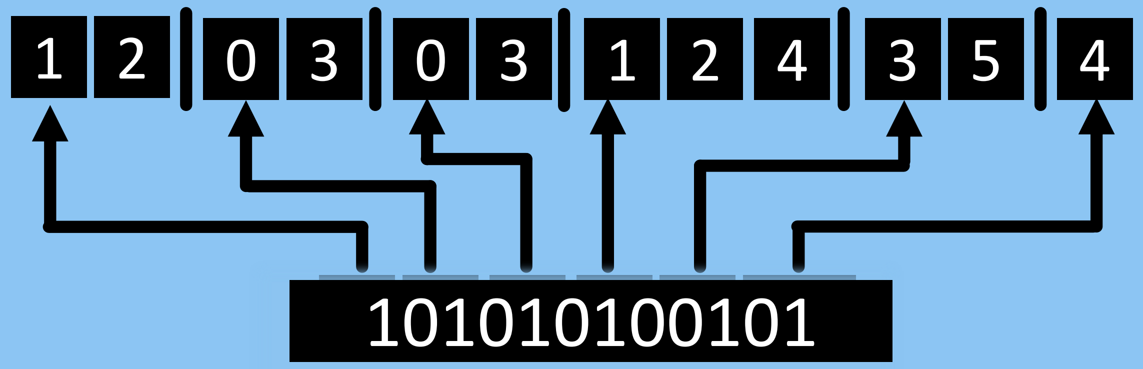
M. Besta et al.: "Log(Graph): A Near-Optimal High-Performance Graph Representation", PACT'18

2 Log (Offset structure)

Use a **bit vector** instead of an array of offsets... !



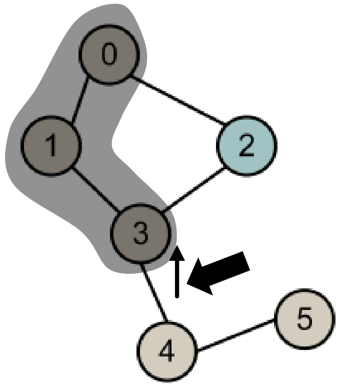
Bit vectors instead of offset arrays



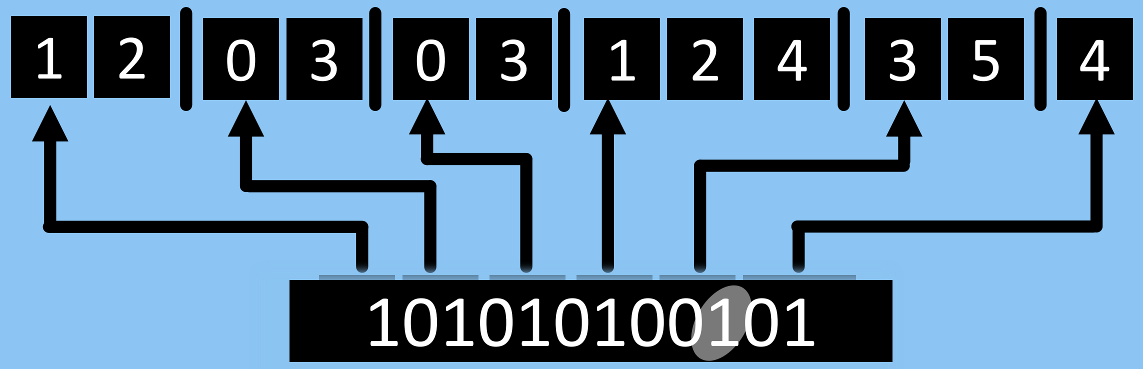
M. Besta et al.: “Log(Graph): A Near-Optimal High-Performance Graph Representation”, PACT’18

2 Log (Offset structure)

Use a **bit vector** instead of an array of offsets... !

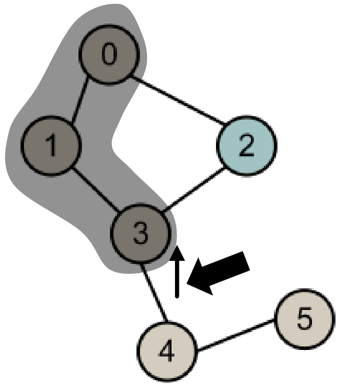


Bit vectors instead of offset arrays



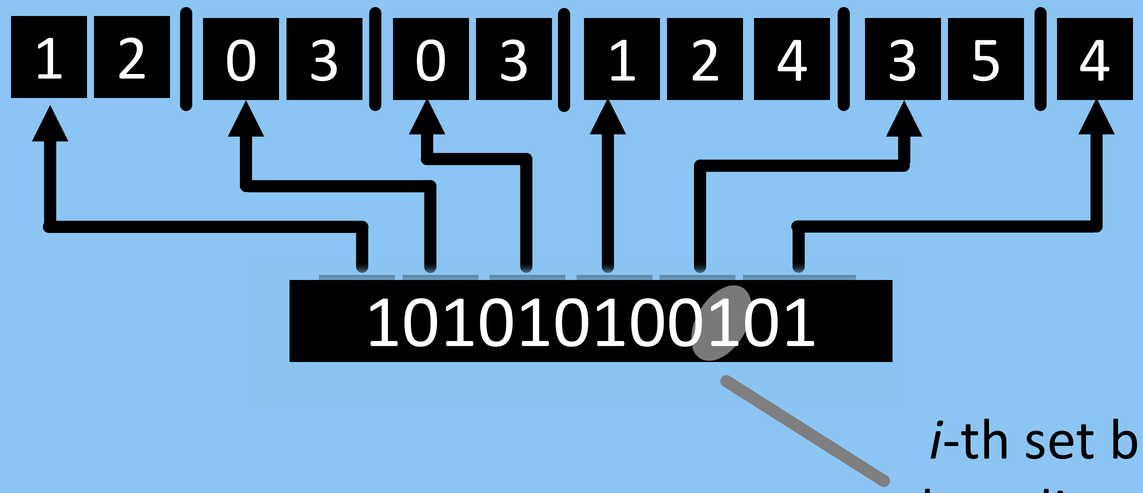
i-th set bit has a position *x* ➔ the adjacency array of a vertex *i* starts at a word *x*

2 Log (Offset structure)



Use a **bit vector** instead of an array of offsets... !

Bit vectors instead of offset arrays



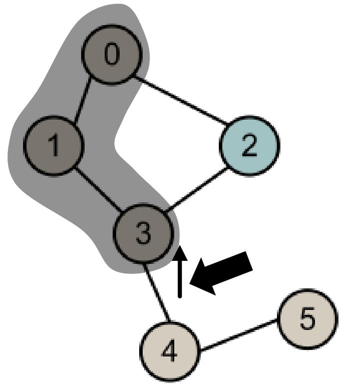
How many 1s are set before a given i-th bit?

i-th set bit has a position *x* ➔ the adjacency array of a vertex *i* starts at a word *x*

2 **Log (** Offset structure **)**

...Encode the resulting bit vectors as **succinct bit vectors [1]**

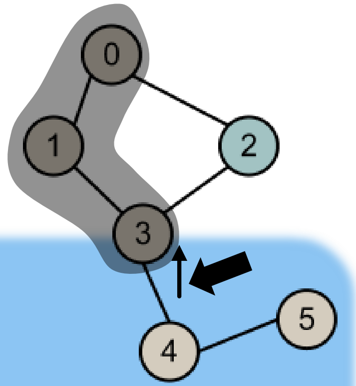
!



[1] G. J. Jacobson. Succinct Static Data Structures. 1988

2 **Log (** Offset structure **)**

...Encode the resulting bit vectors as **succinct bit vectors [1]** !



Succinct bit vectors

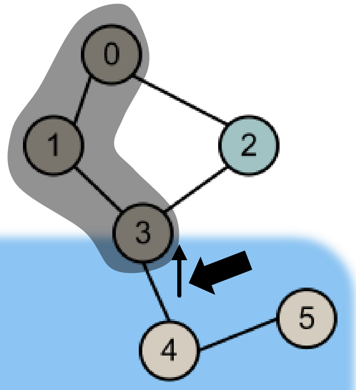
[1] G. J. Jacobson. Succinct Static Data Structures. 1988

M. Besta et al.: "Log(Graph): A Near-Optimal High-Performance Graph Representation", PACT'18

2 **Log (** Offset structure **)**

...Encode the resulting bit vectors as **succinct bit vectors [1]**

!



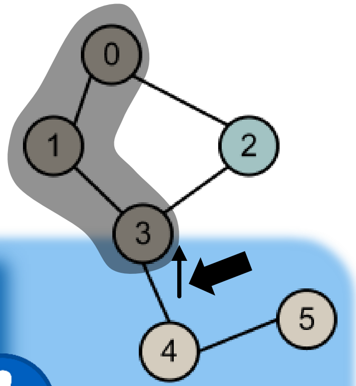
Succinct bit vectors

They use **[Q] + *o*(Q) bits** ([Q] - lower bound), they answer various queries in ***o*(Q) time**.

[1] G. J. Jacobson. Succinct Static Data Structures. 1988

2 Log (Offset structure)

...Encode the resulting bit vectors as **succinct bit vectors [1]** !



Succinct bit vectors

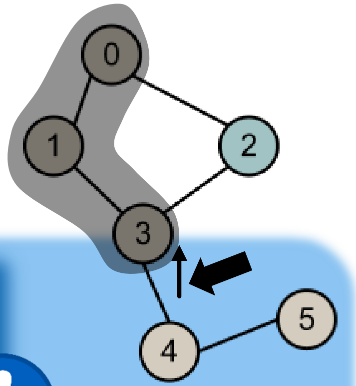
They use **[Q] + *o*(Q) bits** ([Q] - lower bound), they answer various queries in ***o*(Q) time**.

= small + fast (hopefully) !

[1] G. J. Jacobson. Succinct Static Data Structures. 1988

2 Log (Offset structure)

...Encode the resulting bit vectors as **succinct bit vectors [1]** !



Succinct bit vectors

They use **[Q] + *o*(Q) bits** ([Q] - lower bound), they answer various queries in ***o*(Q) time**.

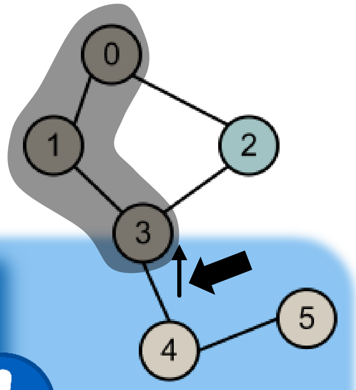
= small + fast (hopefully) !

1010101001010001010101111100000001100001...

[1] G. J. Jacobson. Succinct Static Data Structures. 1988

2 Log (Offset structure)

...Encode the resulting bit vectors as **succinct bit vectors [1]** !



Succinct bit vectors

They use **[Q] + *o*(Q) bits** ([Q] - lower bound), they answer various queries in ***o*(Q) time**.

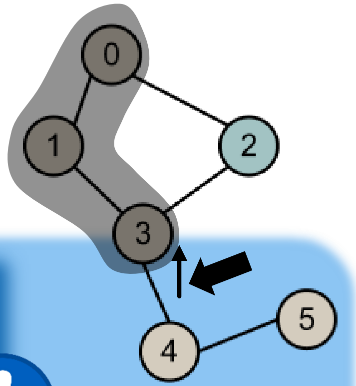
= small + fast (hopefully) !

n bits 1010101001010001010101111100000011000 01...

[1] G. J. Jacobson. Succinct Static Data Structures. 1988

2 Log (Offset structure)

...Encode the resulting bit vectors as **succinct bit vectors [1]** !



Succinct bit vectors

They use **[Q] + *o*(Q) bits** ([Q] - lower bound), they answer various queries in ***o*(Q) time**.

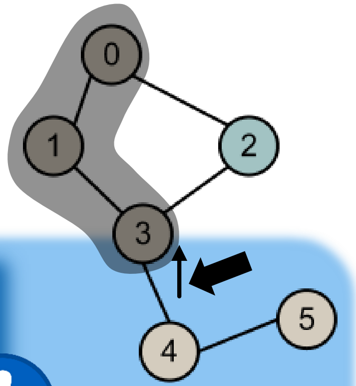
= small + fast (hopefully) !

n bits 1 0 1 0 1 0 1 0 0 1 0 | 1 0 0 0 1 0 1 0 1 0 1 | 1 1 1 1 0 0 0 0 0 0 1 | 1 0 0 0 0 1 . . .

[1] G. J. Jacobson. Succinct Static Data Structures. 1988

2 Log (Offset structure)

...Encode the resulting bit vectors as **succinct bit vectors [1]** !



Succinct bit vectors

They use **[Q] + *o*(Q) bits** ([Q] - lower bound), they answer various queries in ***o*(Q) time**.

= small + fast (hopefully) !

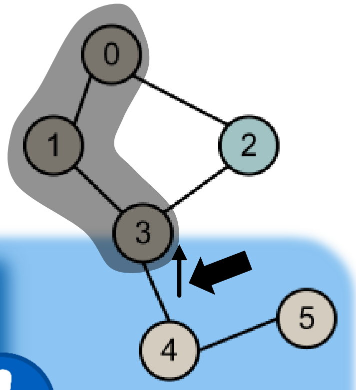
$$\overbrace{1010101001 0}^{\log^2 n}\ \overbrace{10001010101}^{\log^2 n}\ \overbrace{11110000001}^{\log^2 n}\ 100001\ldots$$

n bits: 1 0 1 0 1 0 1 0 0 1 0 | 1 0 0 0 1 0 1 0 1 0 1 | 1 1 1 1 0 0 0 0 0 0 1 | 1 0 0 0 0 1 . . .

[1] G. J. Jacobson. Succinct Static Data Structures. 1988

2 Log (Offset structure)

...Encode the resulting bit vectors as **succinct bit vectors [1]** !



Succinct bit vectors

They use **[Q] + *o*(Q) bits** ([Q] - lower bound), they answer various queries in ***o*(Q) time**.

= small + fast (hopefully) !

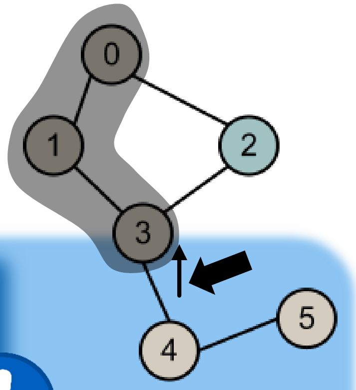
$\log^2 n = t_1$ $\quad \log^2 n \quad \log^2 n \quad \ldots$

n bits: 1 0 1 0 1 0 1 0 0 1 0 | 1 0 0 0 1 0 1 0 1 0 1 | 1 1 1 1 0 0 0 0 0 0 1 | 1 0 0 0 0 1 . . .

[1] G. J. Jacobson. Succinct Static Data Structures. 1988

2 Log (Offset structure)

...Encode the resulting bit vectors as **succinct bit vectors [1]** !



Succinct bit vectors

They use **[Q] + *o*(Q) bits** ([Q] - lower bound), they answer various queries in ***o*(Q) time**.

= small + fast (hopefully) !

$\log^2 n = t_1$ $\quad \log^2 n \quad \log^2 n$ ...

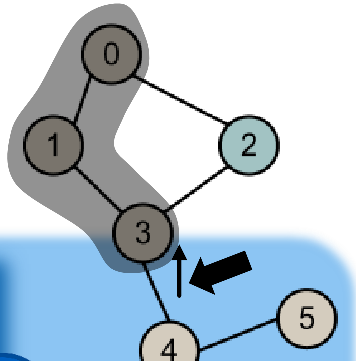
n bits 1 0 1 | 0 1 0 | 1 0 0 1 0 | 1 0 0 | 0 1 0 | 1 0 1 0 1 | 1 1 1 | 1 0 0 | 0 0 0 1 | 1 0 0 0 0 1 . . .

... ...

[1] G. J. Jacobson. Succinct Static Data Structures. 1988

2 Log (Offset structure)

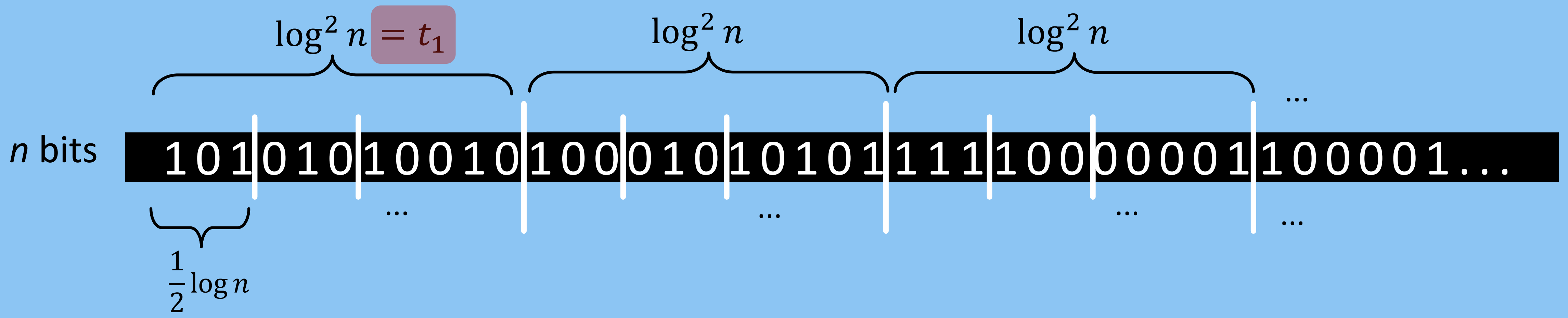
...Encode the resulting bit vectors as **succinct bit vectors [1]** !



Succinct bit vectors

They use **[Q] + *o*(Q) bits** ([Q] - lower bound), they answer various queries in ***o*(Q) time**.

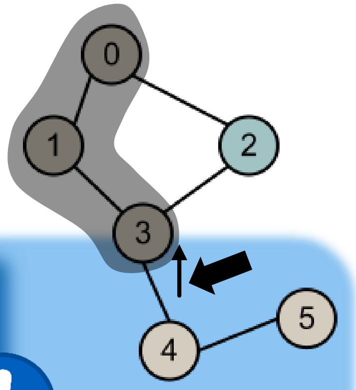
= small + fast (hopefully) !



[1] G. J. Jacobson. Succinct Static Data Structures. 1988

2 Log (Offset structure)

...Encode the resulting bit vectors as **succinct bit vectors [1]** !



Succinct bit vectors

They use **[Q] + *o*(Q) bits** ([Q] - lower bound), they answer various queries in ***o*(Q) time**.

= small + fast (hopefully) !

$\log^2 n = t_1$ | $\log^2 n$ | $\log^2 n$ | ...

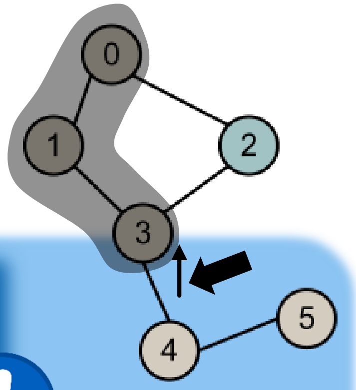
n bits: 1 0 1 | 0 1 0 | 1 0 0 1 0 | 1 0 0 | 0 1 0 | 1 0 1 0 1 | 1 1 1 | 1 0 0 | 0 0 0 1 | 1 0 0 0 0 1 . . .

$\frac{1}{2}\log n$ $\frac{1}{2}\log n$ ... $\frac{1}{2}\log n$ $\frac{1}{2}\log n$ ... $\frac{1}{2}\log n$ $\frac{1}{2}\log n$ ... ...

[1] G. J. Jacobson. Succinct Static Data Structures. 1988

2 Log (Offset structure)

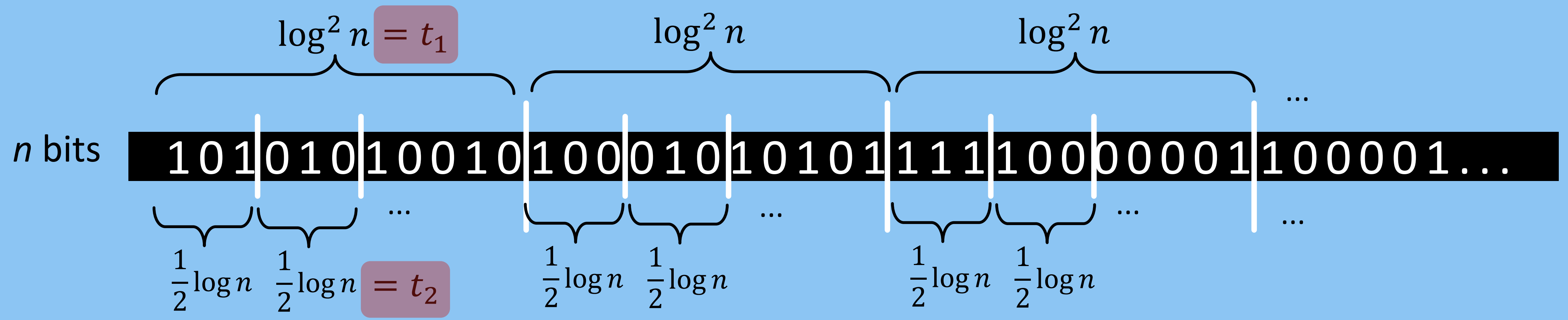
...Encode the resulting bit vectors as **succinct bit vectors [1]** !



Succinct bit vectors

They use **[Q] + *o*(Q) bits** ([Q] - lower bound), they answer various queries in ***o*(Q) time**.

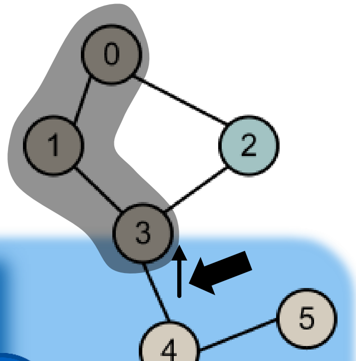
= small + fast (hopefully) !



[1] G. J. Jacobson. Succinct Static Data Structures. 1988

2 Log (Offset structure)

...Encode the resulting bit vectors as **succinct bit vectors [1]** !



Succinct bit vectors

They use **[Q] + *o*(Q) bits** ([Q] - lower bound), they answer various queries in ***o*(Q) time**.

= small + fast (hopefully) !

Compute & store the number of 1s

$\log^2 n = t_1$ $\log^2 n$ $\log^2 n$ ...

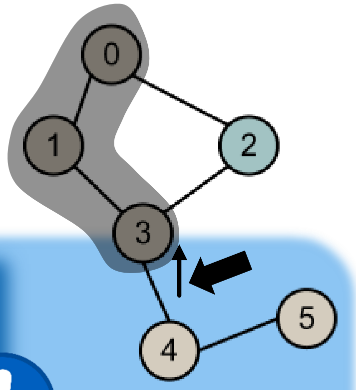
n bits 1 0 1 | 0 1 0 | 1 0 0 1 0 | 1 0 0 | 0 1 0 | 1 0 1 0 1 | 1 1 1 | 1 0 0 | 0 0 0 1 | 1 0 0 0 0 1 . . .

$\frac{1}{2}\log n$ $\frac{1}{2}\log n = t_2$ ... $\frac{1}{2}\log n$ $\frac{1}{2}\log n$ ... $\frac{1}{2}\log n$ $\frac{1}{2}\log n$ ... ...

[1] G. J. Jacobson. Succinct Static Data Structures. 1988

2 Log (Offset structure)

...Encode the resulting bit vectors as **succinct bit vectors [1]** !



Succinct bit vectors

They use **⌈Q⌉ + *o*(Q) bits** (⌈Q⌉ - lower bound), they answer various queries in ***o*(Q) time**.

= small + fast (hopefully) !

Compute & store the number of 1s

$\log^2 n = t_1$ $\quad \log^2 n \quad \log^2 n$ ...

n bits 1 0 1 | 0 1 0 | 1 0 0 1 0 | 1 0 0 | 0 1 0 | 1 0 1 0 1 | 1 1 1 | 1 0 0 | 0 0 0 1 | 1 0 0 0 0 1 . . .

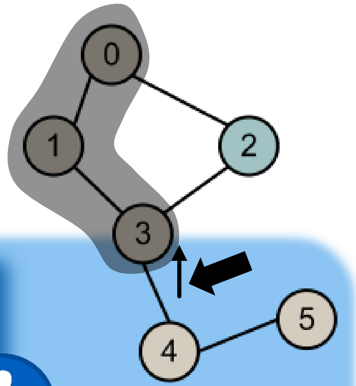
$\frac{1}{2}\log n \quad \frac{1}{2}\log n = t_2$ ... $\frac{1}{2}\log n \quad \frac{1}{2}\log n$ ... $\frac{1}{2}\log n \quad \frac{1}{2}\log n$ ... ...

Compute & store the number of 1s

[1] G. J. Jacobson. Succinct Static Data Structures. 1988

2 Log (Offset structure)

...Encode the resulting bit vectors as **succinct bit vectors [1]** !



Succinct bit vectors

They use **[Q] + *o*(Q) bits** ([Q] - lower bound), they answer various queries in ***o*(Q) time**.

= small + fast (hopefully) !

Compute & store the number of 1s $= O\left(\frac{n}{t_1}\log n\right) = O\left(\frac{n}{\log n}\right) = o(n)$

$\log^2 n = t_1$ $\log^2 n$ $\log^2 n$ ...

n bits 1 0 1|0 1 0|1 0 0 1 0|1 0 0|0 1 0|1 0 1 0 1|1 1 1|1 0 0|0 0 0 0 1|1 0 0 0 0 1 . . .

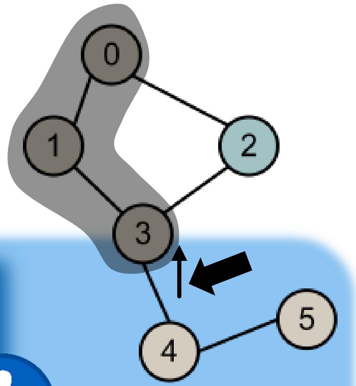
$\frac{1}{2}\log n$ $\frac{1}{2}\log n = t_2$ ... $\frac{1}{2}\log n$ $\frac{1}{2}\log n$ ... $\frac{1}{2}\log n$ $\frac{1}{2}\log n$ ... ...

Compute & store the number of 1s

[1] G. J. Jacobson. Succinct Static Data Structures. 1988

2 Log (Offset structure)

...Encode the resulting bit vectors as succinct bit vectors [1]



Succinct bit vectors

They use **[Q] + *o*(Q) bits** ([Q] - lower bound), they answer various queries in ***o*(Q) time**.

= small + fast (hopefully)

Compute & store the number of 1s $= O\left(\frac{n}{t_1}\log n\right) = O\left(\frac{n}{\log n}\right) = o(n)$

$\log^2 n = t_1$ $\quad \log^2 n \quad \log^2 n \quad \ldots$

n bits: 1 0 1 | 0 1 0 | 1 0 0 1 0 | 1 0 0 | 0 1 0 | 1 0 1 0 1 | 1 1 1 | 1 0 0 | 0 0 0 1 | 1 0 0 0 0 1 . . .

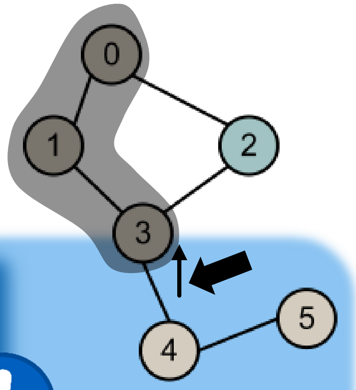
$\frac{1}{2}\log n \quad \frac{1}{2}\log n = t_2 \quad \ldots \quad \frac{1}{2}\log n \quad \frac{1}{2}\log n \quad \ldots \quad \frac{1}{2}\log n \quad \frac{1}{2}\log n \quad \ldots \quad \ldots$

Compute & store the number of 1s $= O\left(\frac{n}{t_2}\log t_1\right) = O\left(\frac{n\log\log n}{\log n}\right) = o(n)$

[1] G. J. Jacobson. Succinct Static Data Structures. 1988

2 Log (Offset structure)

...Encode the resulting bit vectors as **succinct bit vectors [1]** !



Succinct bit vectors

They use **⌈Q⌉ + *o*(Q) bits** (⌈Q⌉ - lower bound), they answer various queries in ***o*(Q) time**.

= small + fast (hopefully) !

Compute & store the number of 1s $= O\left(\frac{n}{t_1}\log n\right) = O\left(\frac{n}{\log n}\right) = o(n)$

$\log^2 n = t_1$ $\log^2 n$ $\log^2 n$ ...

n bits 1 0 1|0 1 0|1 0 0 1 0|1 0 0|0 1 0|1 0 1 0 1|1 1 1|1 0 0|0 0 0 1|1 0 0 0 0 1 . . .

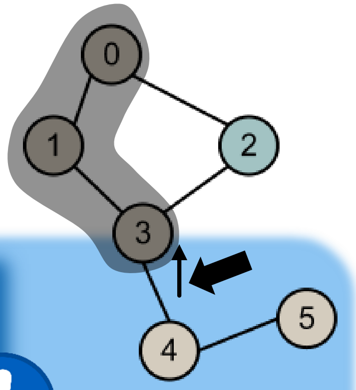
$\frac{1}{2}\log n$ $\frac{1}{2}\log n = t_2$ ... $\frac{1}{2}\log n$ $\frac{1}{2}\log n$ ... $\frac{1}{2}\log n$ $\frac{1}{2}\log n$ ... ...

Compute & store the number of 1s $= O\left(\frac{n}{t_2}\log t_1\right) = O\left(\frac{n\log\log n}{\log n}\right) = o(n)$

[1] G. J. Jacobson. Succinct Static Data Structures. 1988

2 Log (Offset structure)

...Encode the resulting bit vectors as **succinct bit vectors [1]** !



Succinct bit vectors

They use **[Q] + *o*(Q) bits** ([Q] - lower bound), they answer various queries in ***o*(Q) time**.

= small + fast (hopefully) !

Total storage:

$n + o(n) + o(n) + \cdots$
$= n + o(n)$!

Compute & store the number of 1s $= O\left(\frac{n}{t_1}\log n\right) = O\left(\frac{n}{\log n}\right) = o(n)$

$\log^2 n = t_1$ $\quad \log^2 n \quad \log^2 n \quad \ldots$

n bits 1 0 1 0 1 0 1 0 0 1 0 1 0 0 0 1 0 1 0 1 0 1 1 1 1 1 0 0 0 0 0 1 1 0 0 0 0 1 . . .

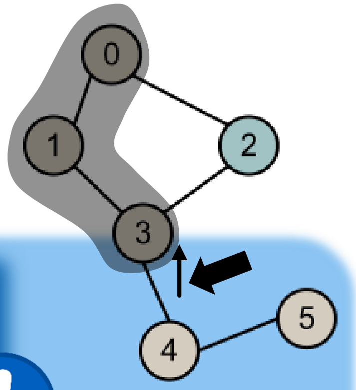
$\frac{1}{2}\log n \quad \frac{1}{2}\log n = t_2 \quad \ldots \quad \frac{1}{2}\log n \quad \frac{1}{2}\log n \quad \ldots \quad \frac{1}{2}\log n \quad \frac{1}{2}\log n \quad \ldots \quad \ldots$

Compute & store the number of 1s $= O\left(\frac{n}{t_2}\log t_1\right) = O\left(\frac{n\log\log n}{\log n}\right) = o(n)$

[1] G. J. Jacobson. Succinct Static Data Structures. 1988

2 Log (Offset structure)

...Encode the resulting bit vectors as **succinct bit vectors [1]** !



Succinct bit vectors

They use **[Q] + *o*(Q) bits** ([Q] - lower bound), they answer various queries in ***o*(Q) time**.

= small + fast (hopefully) !

Total storage:
$n + o(n) + o(n) + \cdots$
$= n + o(n)$!

Compute & store the number of 1s $= O\left(\frac{n}{t_1}\log n\right) = O\left(\frac{n}{\log n}\right) = o(n)$

$\log^2 n =$

n bits 1 0 1 | 0 1 0 | 1 ... 1 | 1 0 0 0 0 1 . . .

...

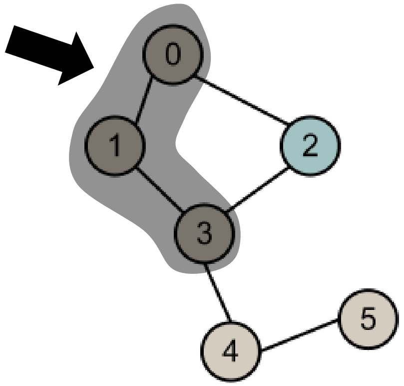
$\frac{1}{2}\log n$ $\frac{1}{2}\log n$ $= t_2$ ... $\frac{1}{2}\log n$ $\frac{1}{2}\log n$ $\frac{1}{2}\log n$ $\frac{1}{2}\log n$

Compute & store the number of 1s $= O\left(\frac{n}{t_2}\log t_1\right) = O\left(\frac{n\log\log n}{\log n}\right) = o(n)$

We will show that they are in practice both small and fast!

[1] G. J. Jacobson. Succinct Static Data Structures. 1988

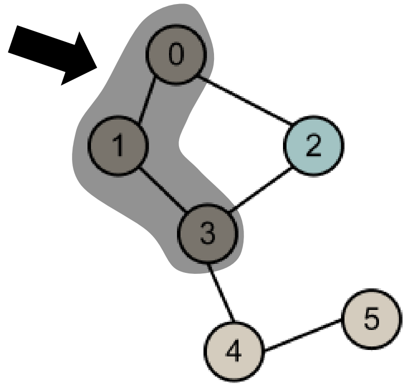
3 **Log (** Adjacency structure **)**



3 **Log (** Adjacency structure **)**

Use different relabelings

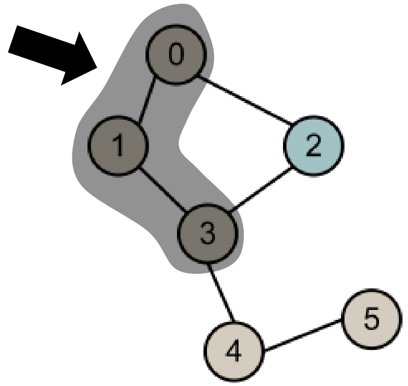
!



M. Besta et al.: “Log(Graph): A Near-Optimal High-Performance Graph Representation”, PACT’18

3 **Log (** Adjacency structure **)**

Use different relabelings !

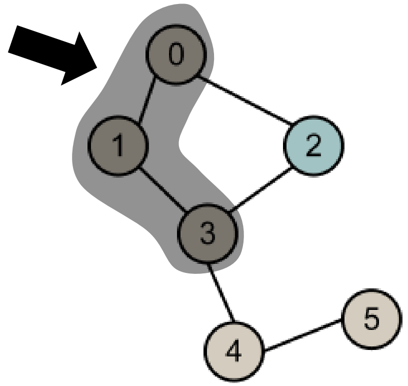


Degree-Minimizing: Targeting general graphs (no assumptions on graph structure)

3 **Log (** Adjacency structure **)**

Use different relabelings

!



Degree-Minimizing: Targeting general graphs (no assumptions on graph structure)

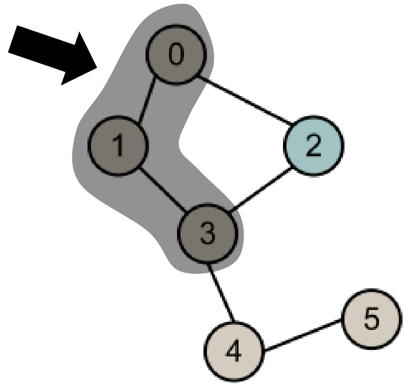
More schemes that assume specific classes of graphs

...

M. Besta et al.: "Log(Graph): A Near-Optimal High-Performance Graph Representation", PACT'18

3 **Log (** Adjacency structure **)**

Use different relabelings !



Degree-Minimizing: Targeting general graphs (no assumptions on graph structure)

Permute(2 | 3 | 4 | 5 | 1M) = v | w | x | y | z

(simultaneously for all other neighborhoods)

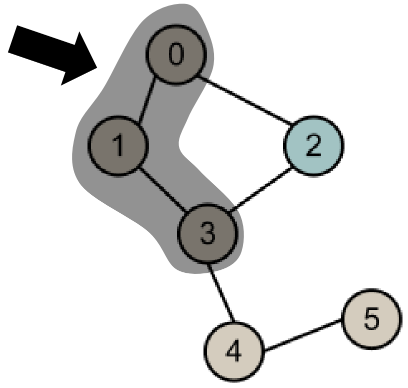
More schemes that assume specific classes of graphs

…

M. Besta et al.: "Log(Graph): A Near-Optimal High-Performance Graph Representation", PACT'18

3 **Log (** Adjacency structure **)**

Use different relabelings !



Degree-Minimizing: Targeting general graphs (no assumptions on graph structure)

Permute(2 | 3 | 4 | 5 | 1M) = v | w | x | y | z

(simultaneously for all other neighborhoods)

(1) The more often a label occurs (i.e., the higher vertex degree), the smaller permuted value it receives

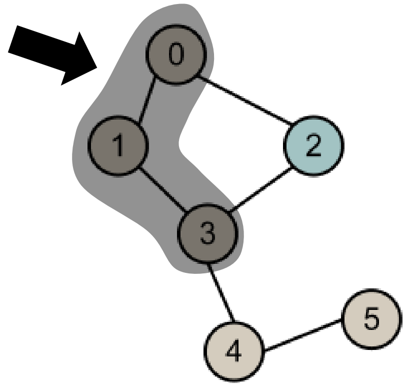
More schemes that assume specific classes of graphs

…

M. Besta et al.: "Log(Graph): A Near-Optimal High-Performance Graph Representation", PACT'18

3 **Log (** Adjacency structure **)**

Use different relabelings !



Degree-Minimizing: Targeting general graphs (no assumptions on graph structure)

Permute(2 | 3 | 4 | 5 | 1M) = v | w | x | y | z

(simultaneously for all other neighborhoods)

(1) The more often a label occurs (i.e., the higher vertex degree), the smaller permuted value it receives

Gap-encode(v | w | x | y | z) = v | w-v | x-w | y-x | z-y

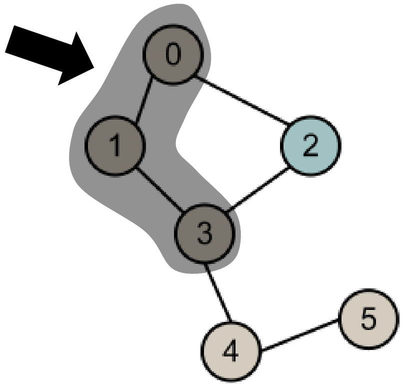
More schemes that assume specific classes of graphs

...

M. Besta et al.: "Log(Graph): A Near-Optimal High-Performance Graph Representation", PACT'18

3 Log (Adjacency structure)

Use different relabelings !



Degree-Minimizing: Targeting general graphs (no assumptions on graph structure)

Permute(2 | 3 | 4 | 5 | 1M) = v | w | x | y | z

(simultaneously for all other neighborhoods)

(1) The more often a label occurs (i.e., the higher vertex degree), the smaller permuted value it receives

Gap-encode(v | w | x | y | z) = v | w-v | x-w | y-x | z-y

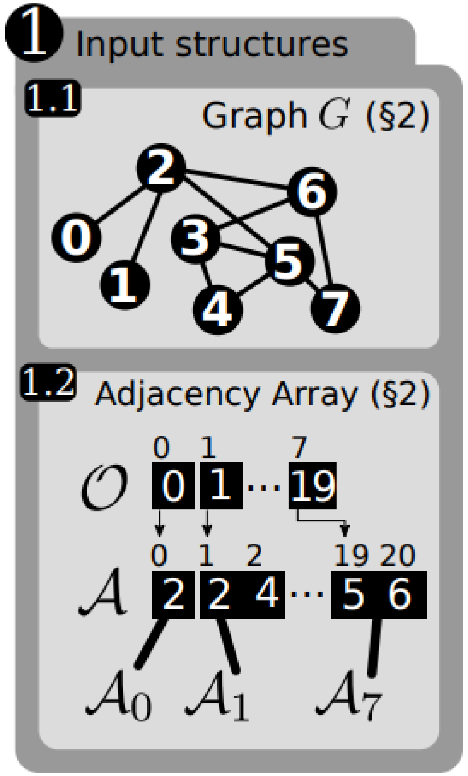
(2) Encode new labels with gap encoding (differences between consecutive labels instead of full labels)

More schemes that assume specific classes of graphs

...

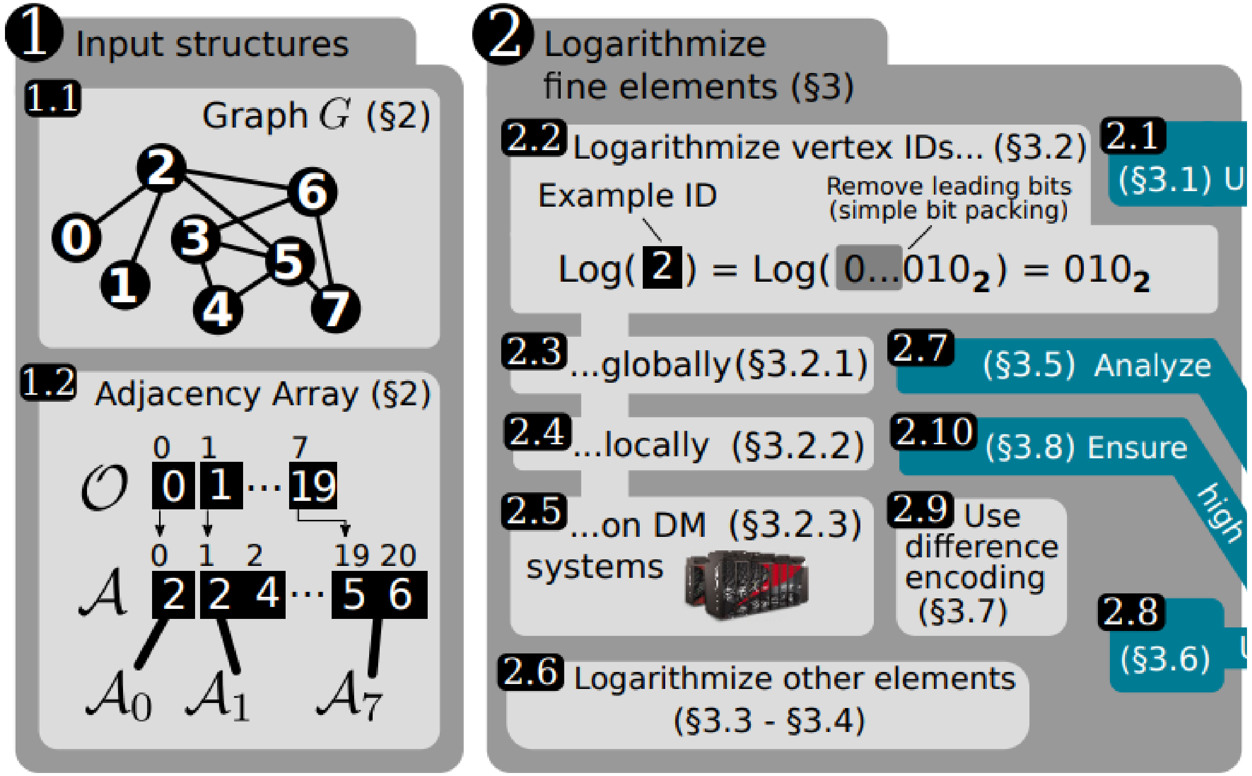
OVERVIEW OF FULL LOG(GRAPH) DESIGN

Overview of Full Log(Graph) Design



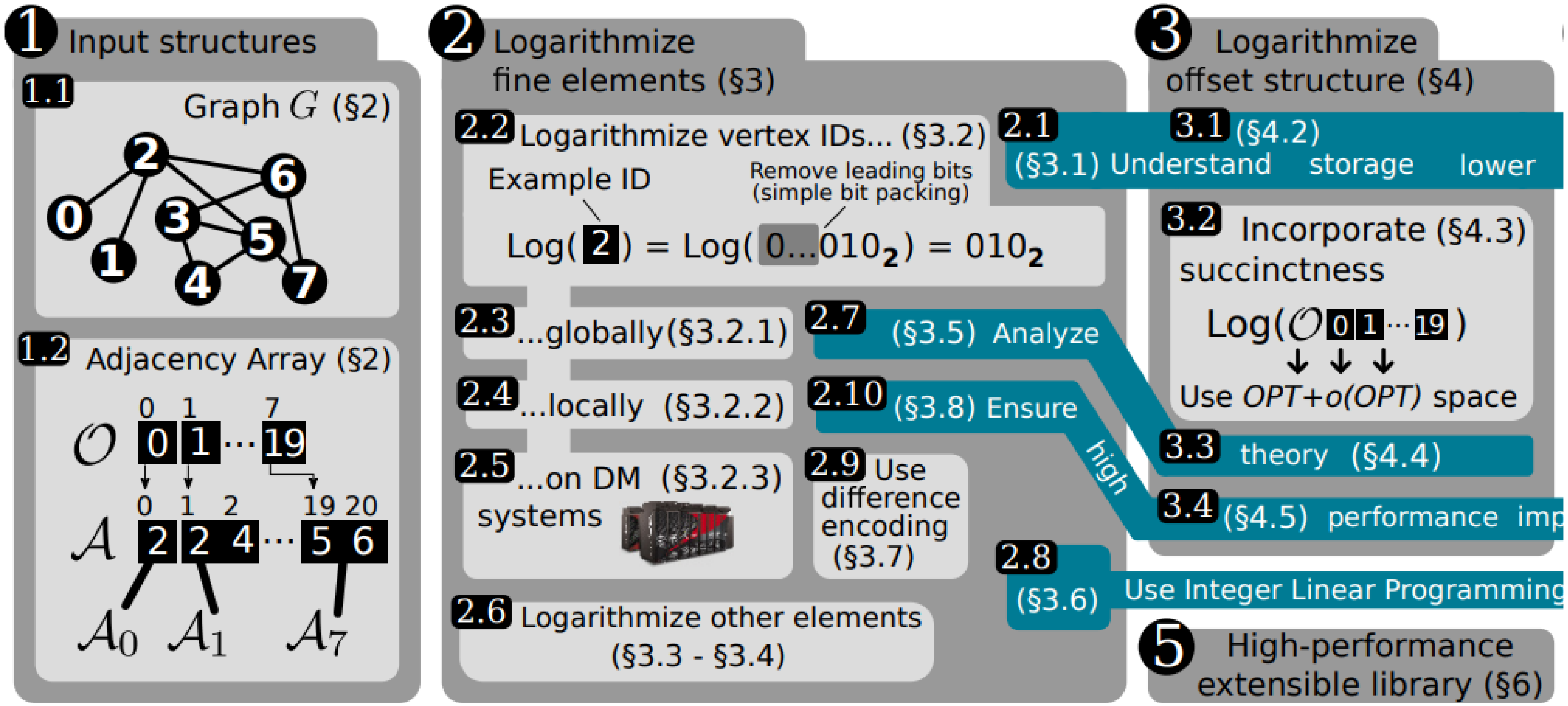
M. Besta et al.: “Log(Graph): A Near-Optimal High-Performance Graph Representation”, PACT’18

Overview of Full Log(Graph) Design



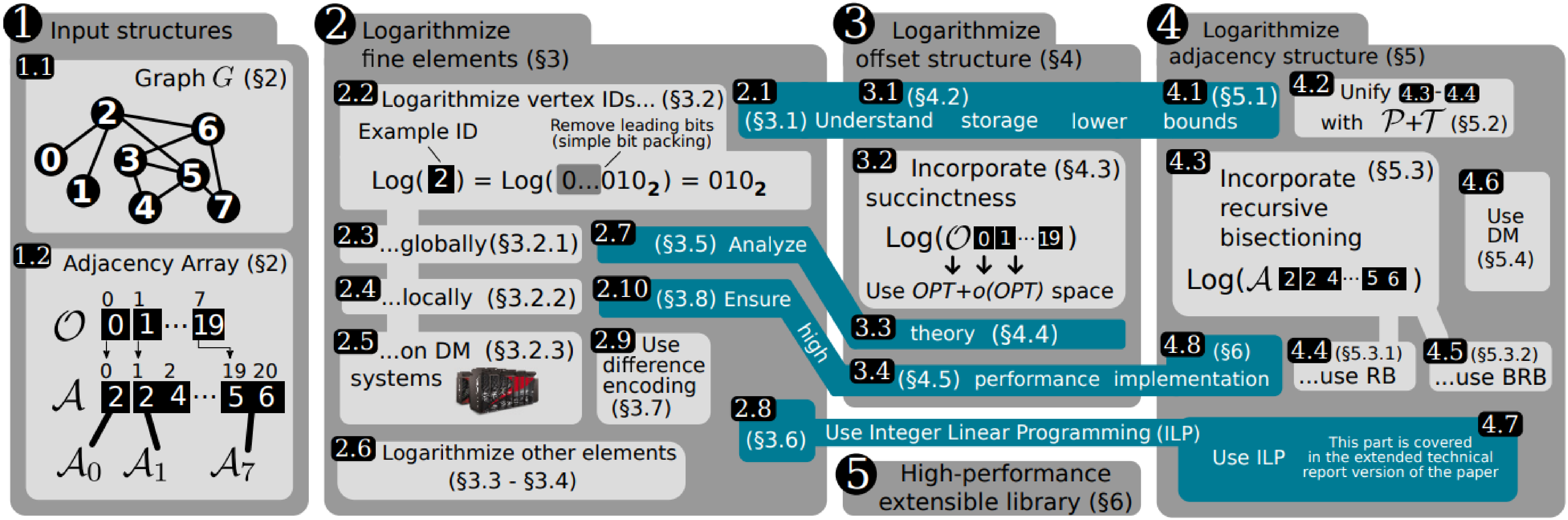
M. Besta et al.: "Log(Graph): A Near-Optimal High-Performance Graph Representation", PACT'18

Overview of Full Log(Graph) Design

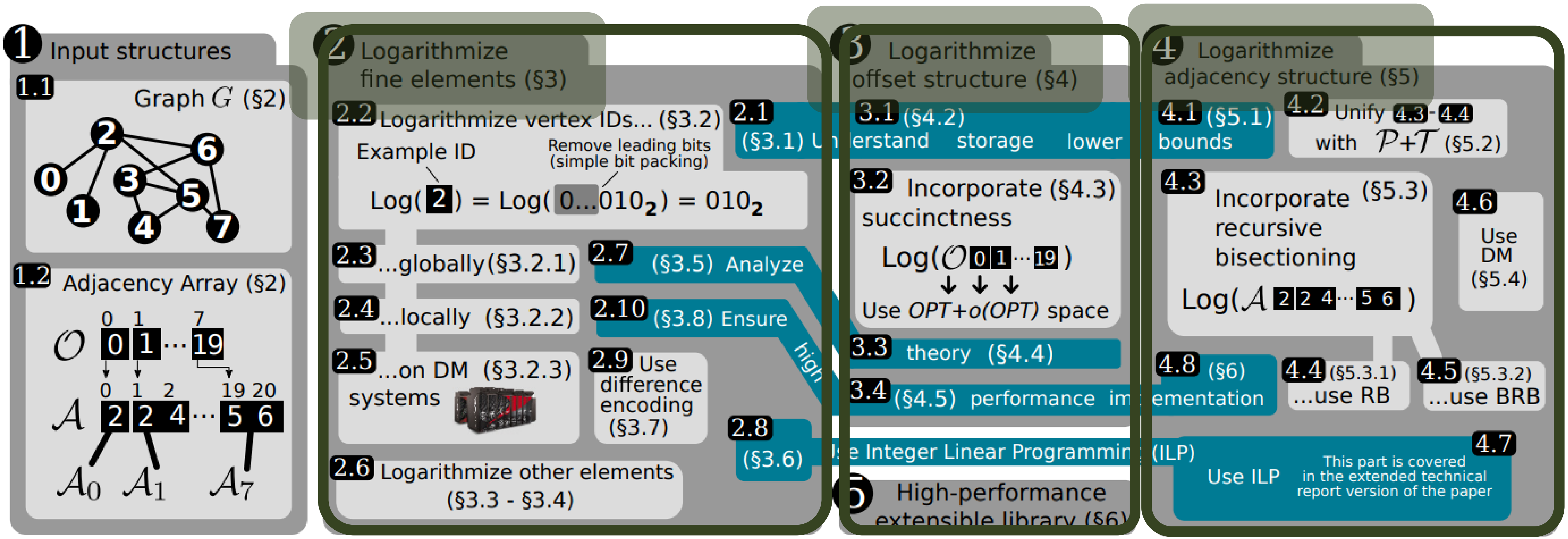


M. Besta et al.: “Log(Graph): A Near-Optimal High-Performance Graph Representation”, PACT’18

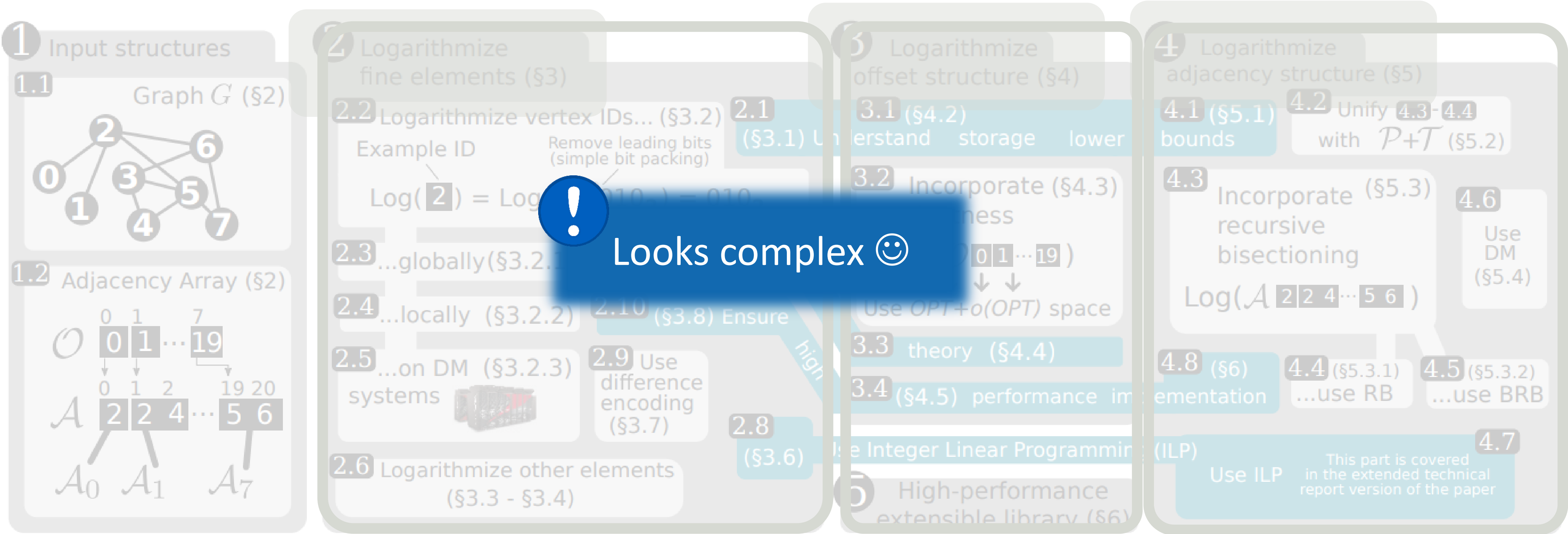
OVERVIEW OF FULL LOG(GRAPH) DESIGN



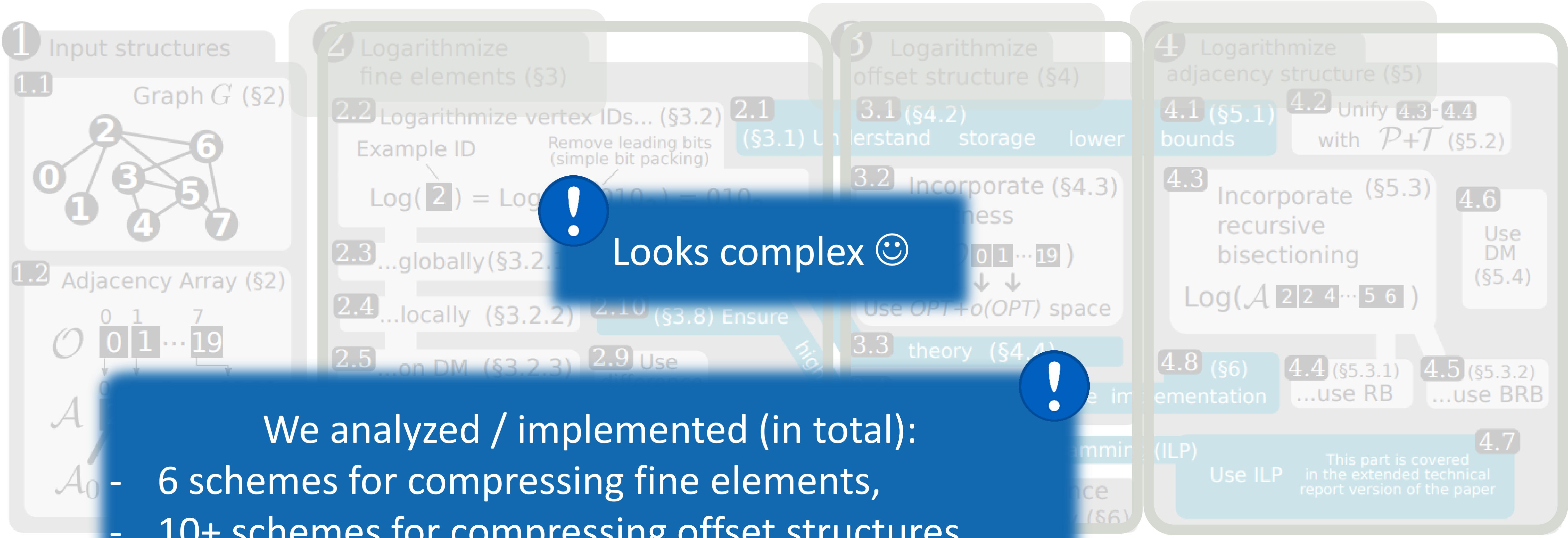
Overview of Full Log(Graph) Design



Overview of Full Log(Graph) Design



Overview of Full Log(Graph) Design

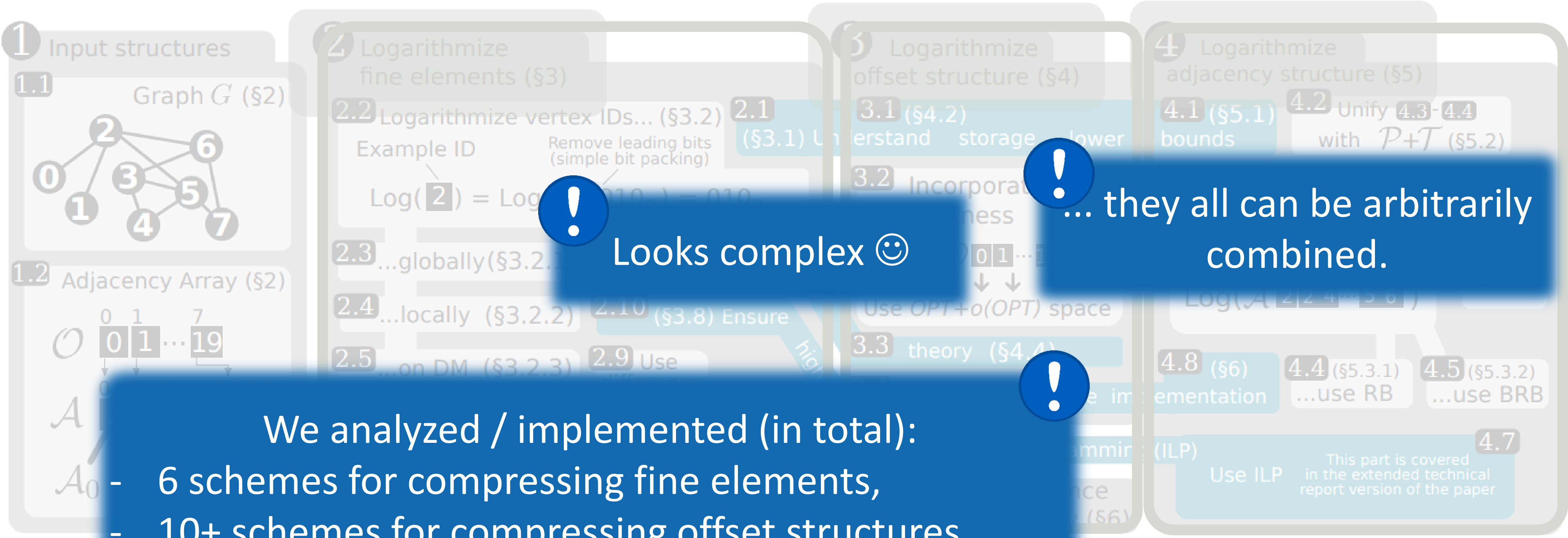


Looks complex ☺

We analyzed / implemented (in total):

- 6 schemes for compressing fine elements,
- 10+ schemes for compressing offset structures,
- 4+ schemes for compressing adjacency structures

Overview of Full Log(Graph) Design



Looks complex ☺

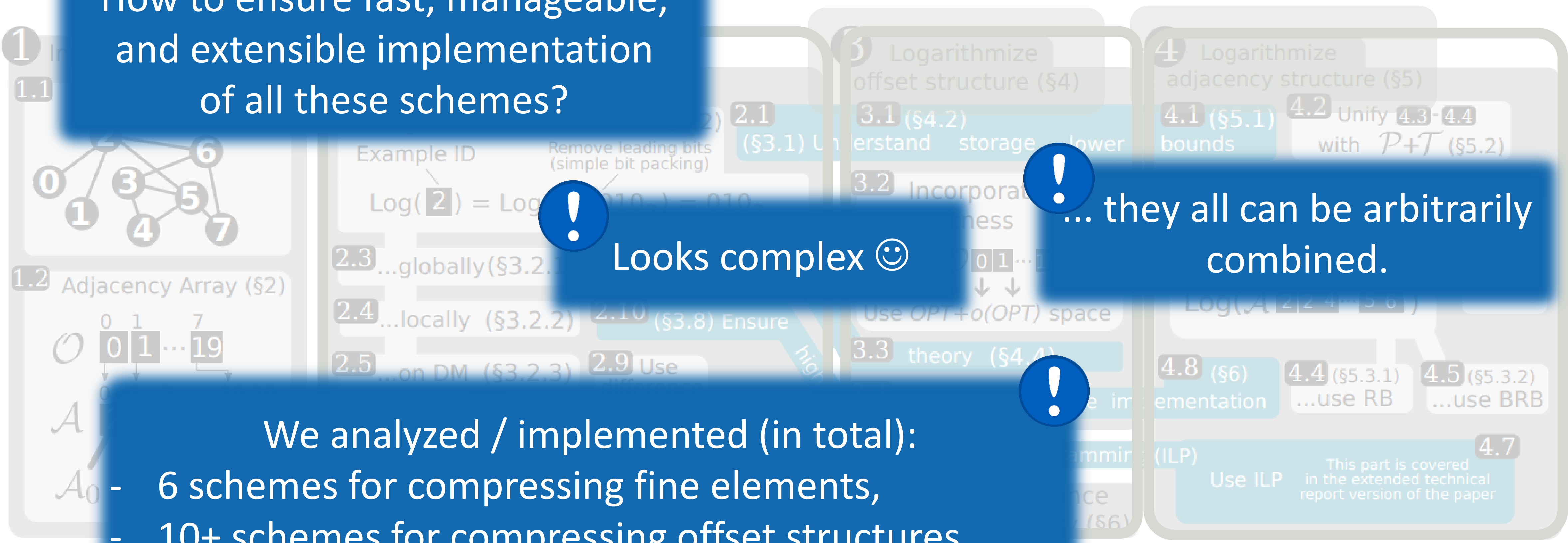
… they all can be arbitrarily combined.

We analyzed / implemented (in total):

- 6 schemes for compressing fine elements,
- 10+ schemes for compressing offset structures,
- 4+ schemes for compressing adjacency structures

Overview of Full Log(Graph) Design

How to ensure fast, manageable, and extensible implementation of all these schemes?



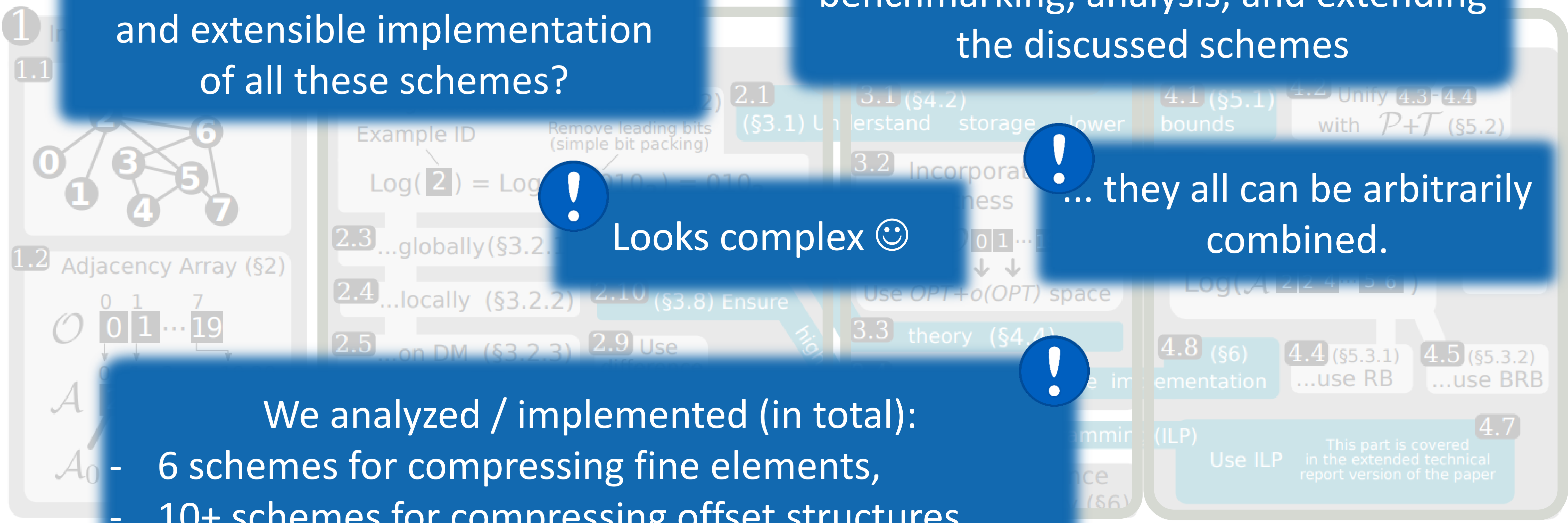
Looks complex ☺

... they all can be arbitrarily combined.

We analyzed / implemented (in total):

- 6 schemes for compressing fine elements,
- 10+ schemes for compressing offset structures,
- 4+ schemes for compressing adjacency structures

Overview of Full Log(Graph) Design



How to ensure fast, manageable, and extensible implementation of all these schemes?

We use C++ templates to develop a library that facilitates implementation, benchmarking, analysis, and extending the discussed schemes

Looks complex ☺

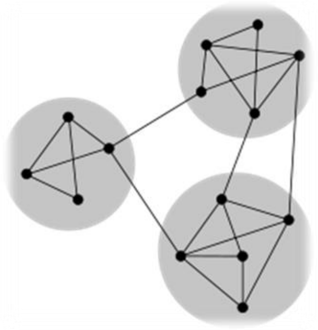
... they all can be arbitrarily combined.

We analyzed / implemented (in total):

- 6 schemes for compressing fine elements,
- 10+ schemes for compressing offset structures,
- 4+ schemes for compressing adjacency structures

1 **Log (Vertex labels), Log (Edge weights)** **Storage, Performance**

SSSP

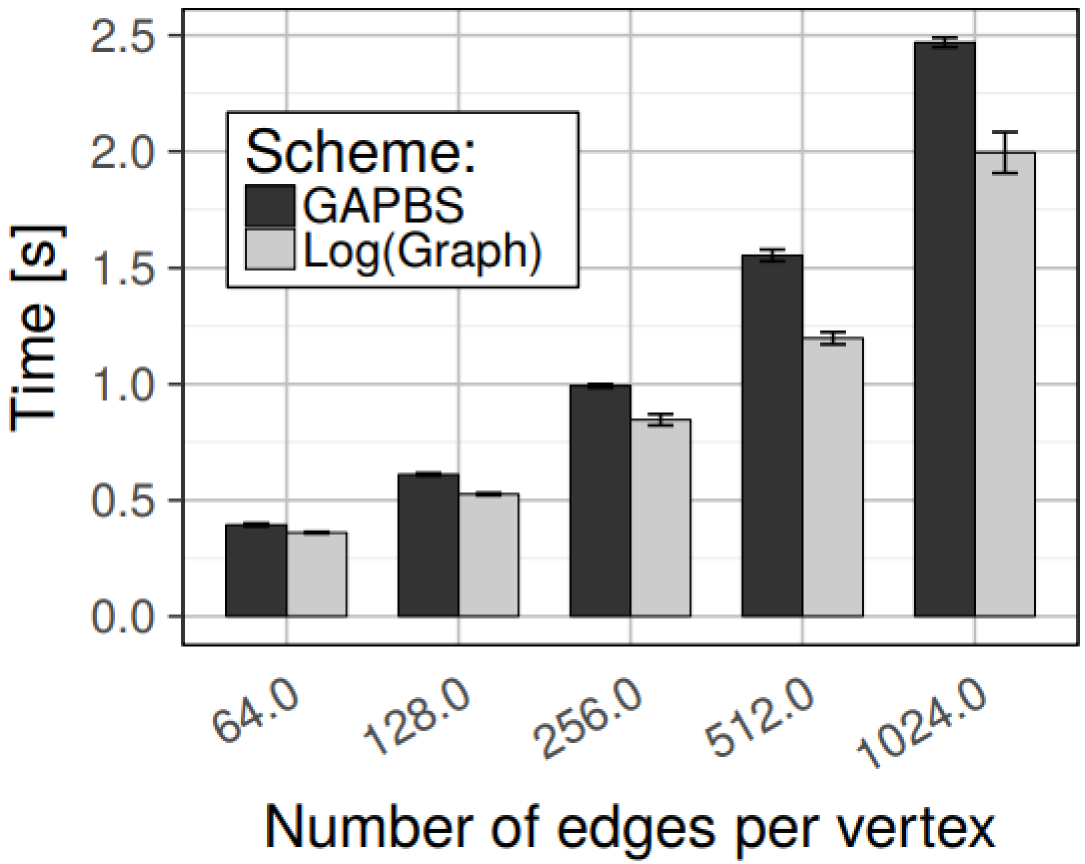


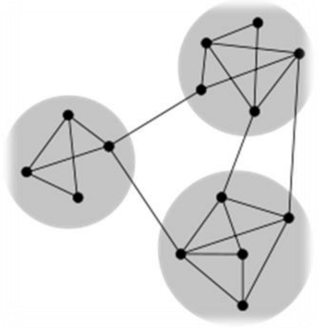
Kronecker graphs
Number of vertices: 4M

M. Besta et al.: “Log(Graph): A Near-Optimal High-Performance Graph Representation”, PACT’18

SSSP

1 Log (Vertex labels), Log (Edge weights) **Storage, Performance**



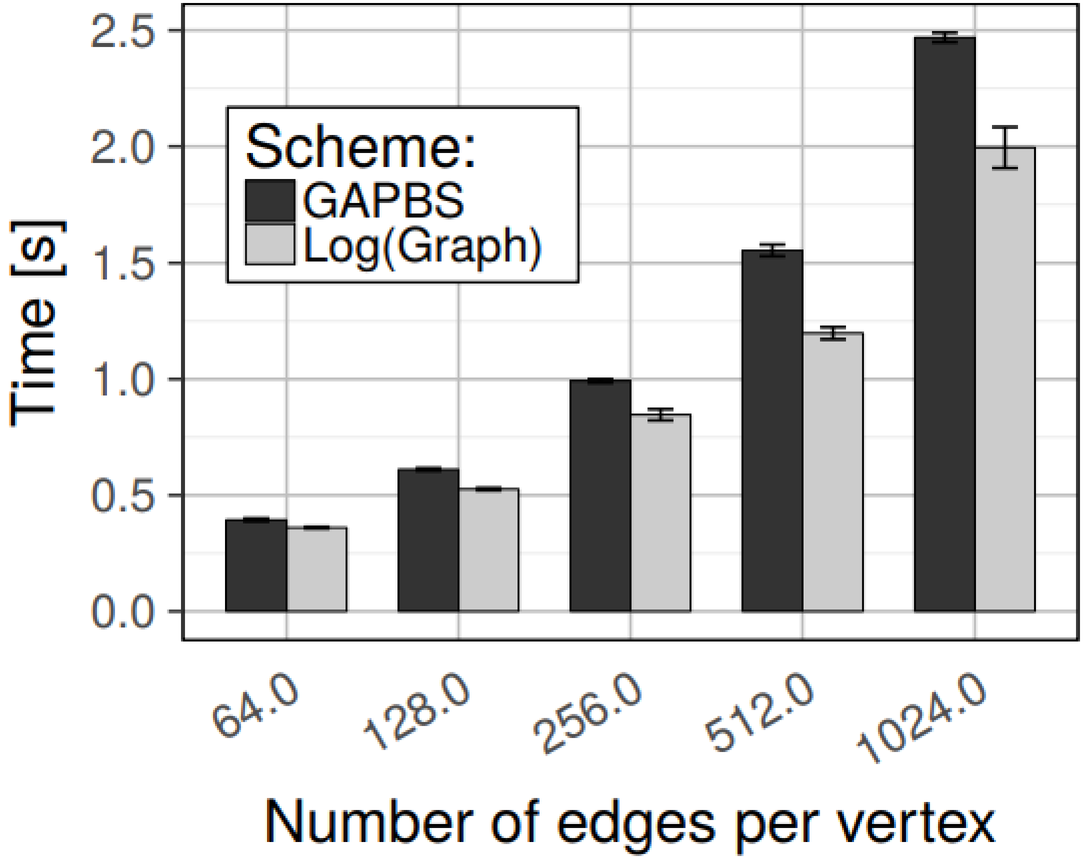


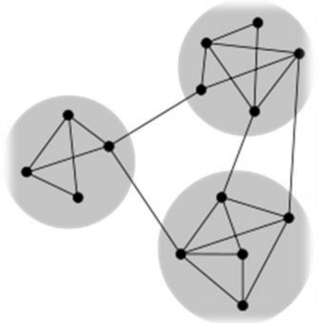
Kronecker graphs
Number of vertices: 4M

M. Besta et al.: "Log(Graph): A Near-Optimal High-Performance Graph Representation", PACT'18

SSSP

1 **Log (Vertex labels), Log (Edge weights)** **Storage, Performance**





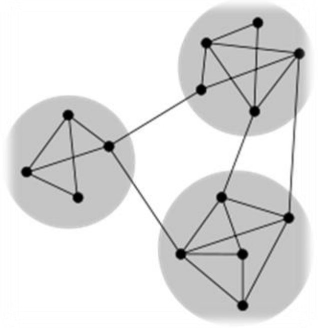
Kronecker graphs
Number of vertices: 4M

Log(Graph) consistently reduces storage overhead (by 20-35%)

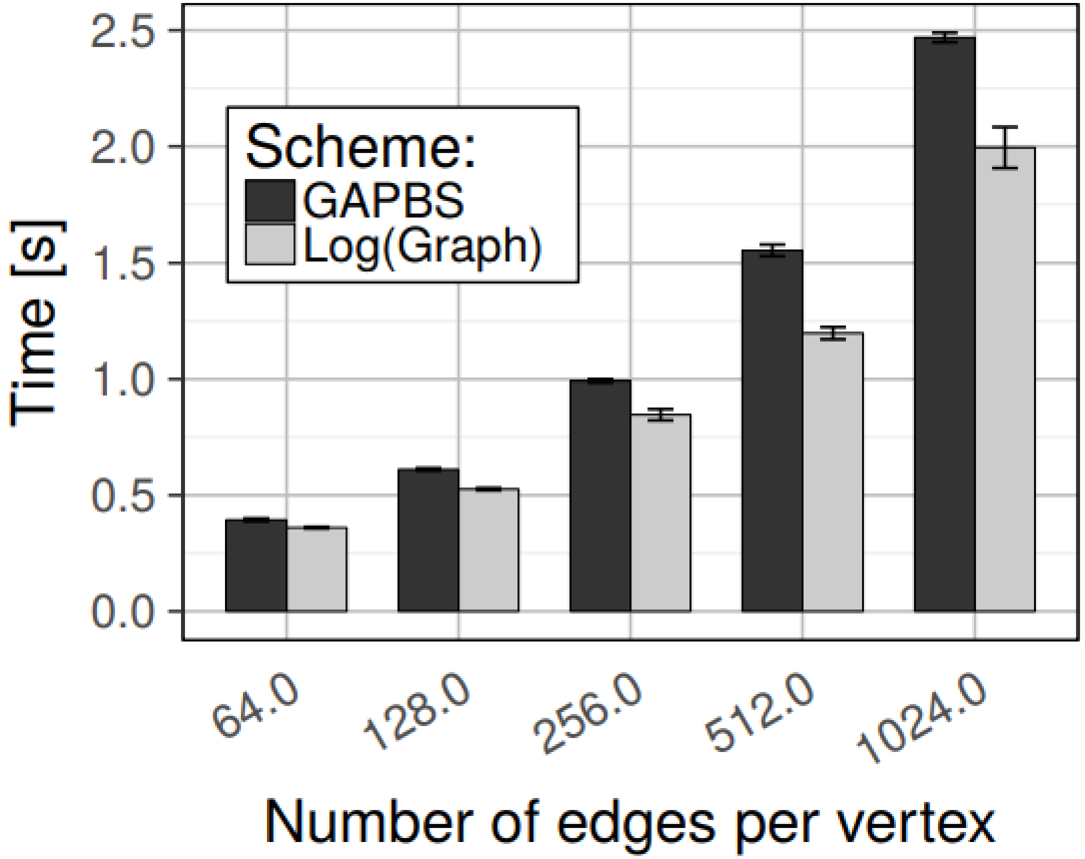
M. Besta et al.: "Log(Graph): A Near-Optimal High-Performance Graph Representation", PACT'18

SSSP

❶ **Log (Vertex labels), Log (Edge weights)** **Storage, Performance**



Log(Graph) **accelerates** GAPBS

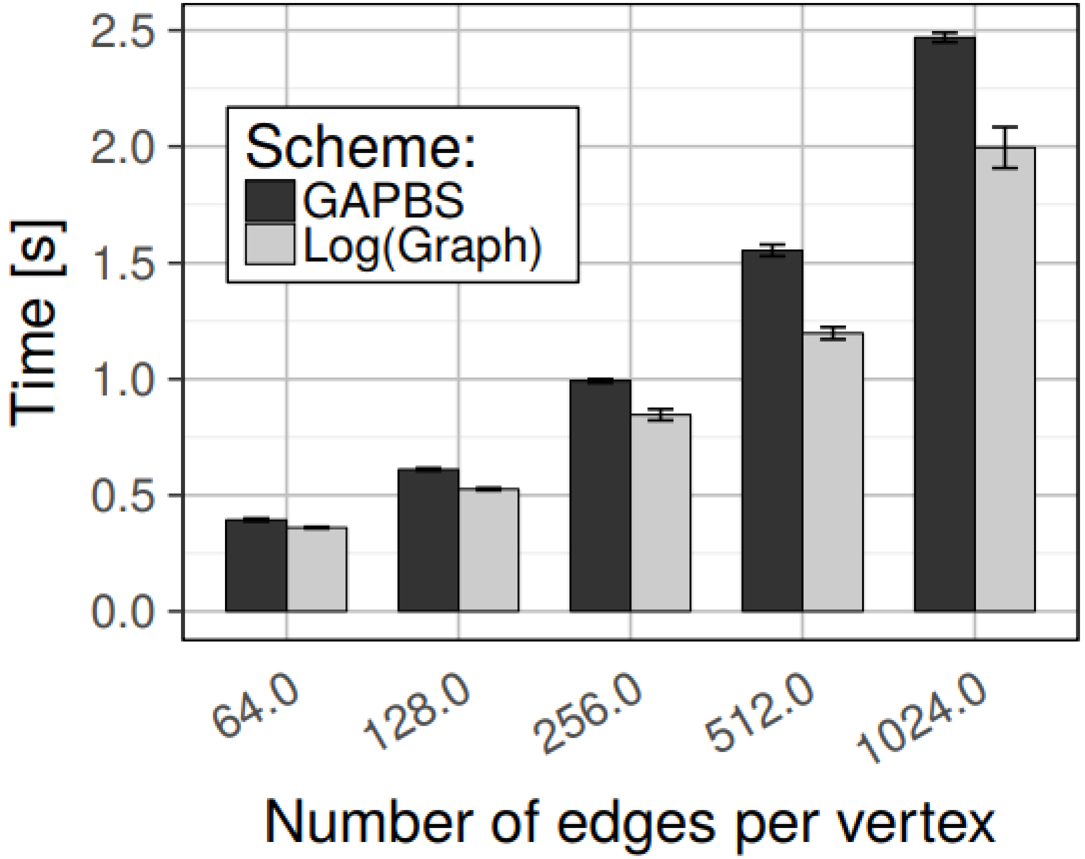


Log(Graph) consistently reduces storage overhead (by 20-35%)

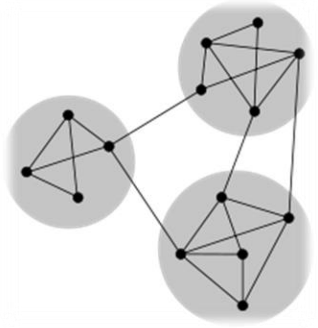
M. Besta et al.: "Log(Graph): A Near-Optimal High-Performance Graph Representation", PACT'18

1 Log (Vertex labels), Log (Edge weights) Storage, Performance

SSSP



Log(Graph) **accelerates** GAPBS



Kronecker graphs
Number of vertices: 4M

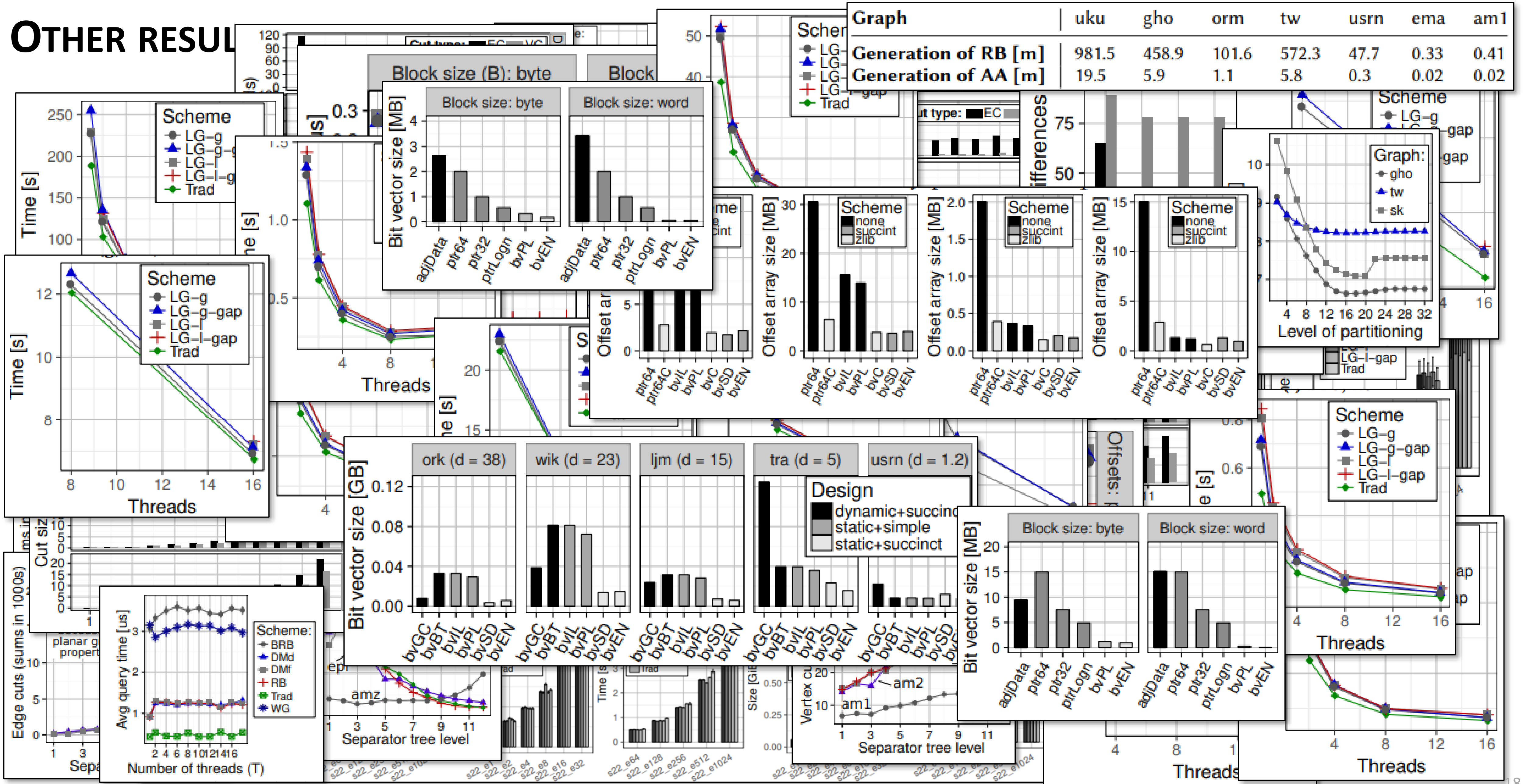
Both storage and performance are improved **simultaneously**

Log(Graph) consistently reduces storage overhead (by 20-35%)

M. Besta et al.: "Log(Graph): A Near-Optimal High-Performance Graph Representation", PACT'18

OTHER RESULTS

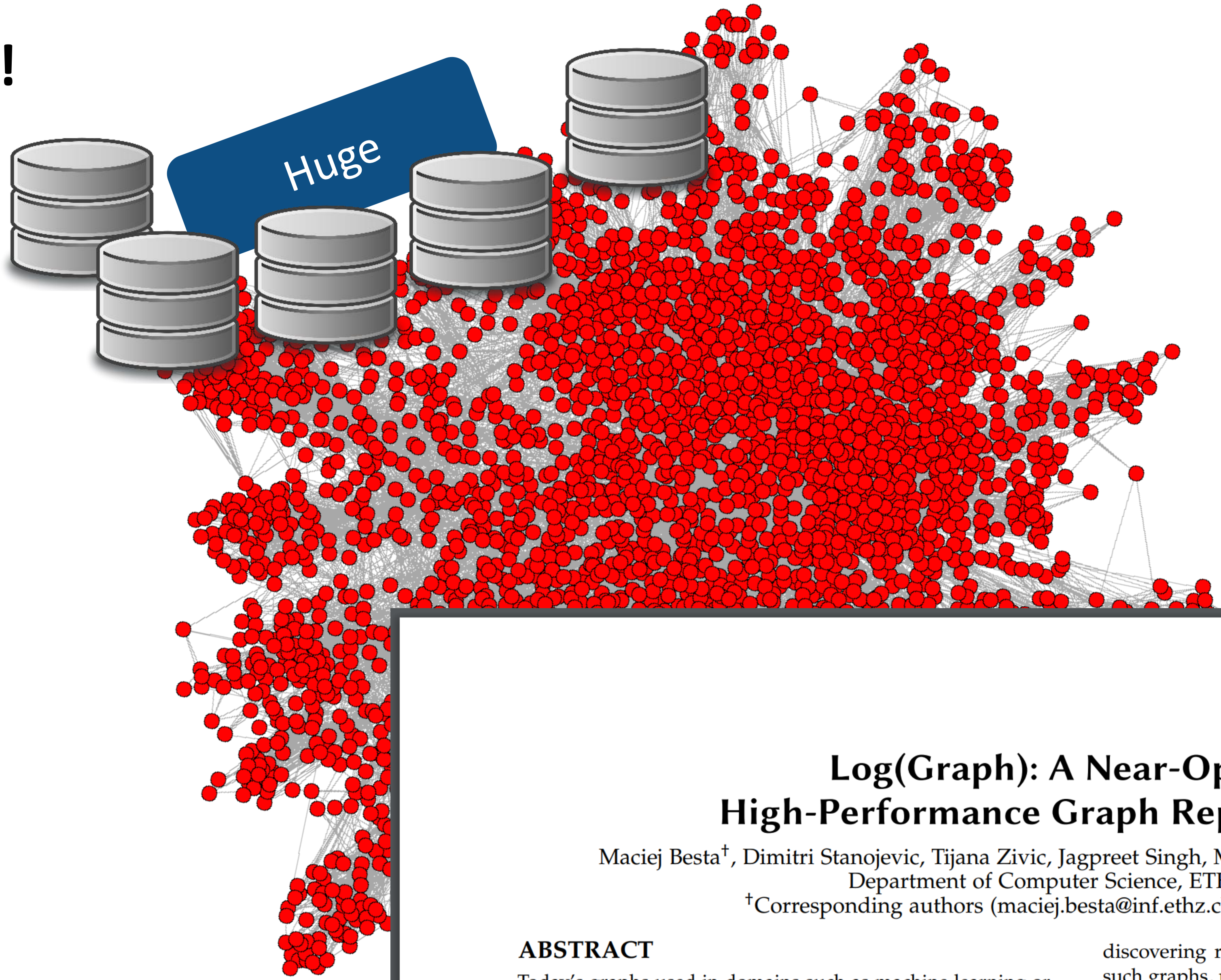
OTHER RESUL

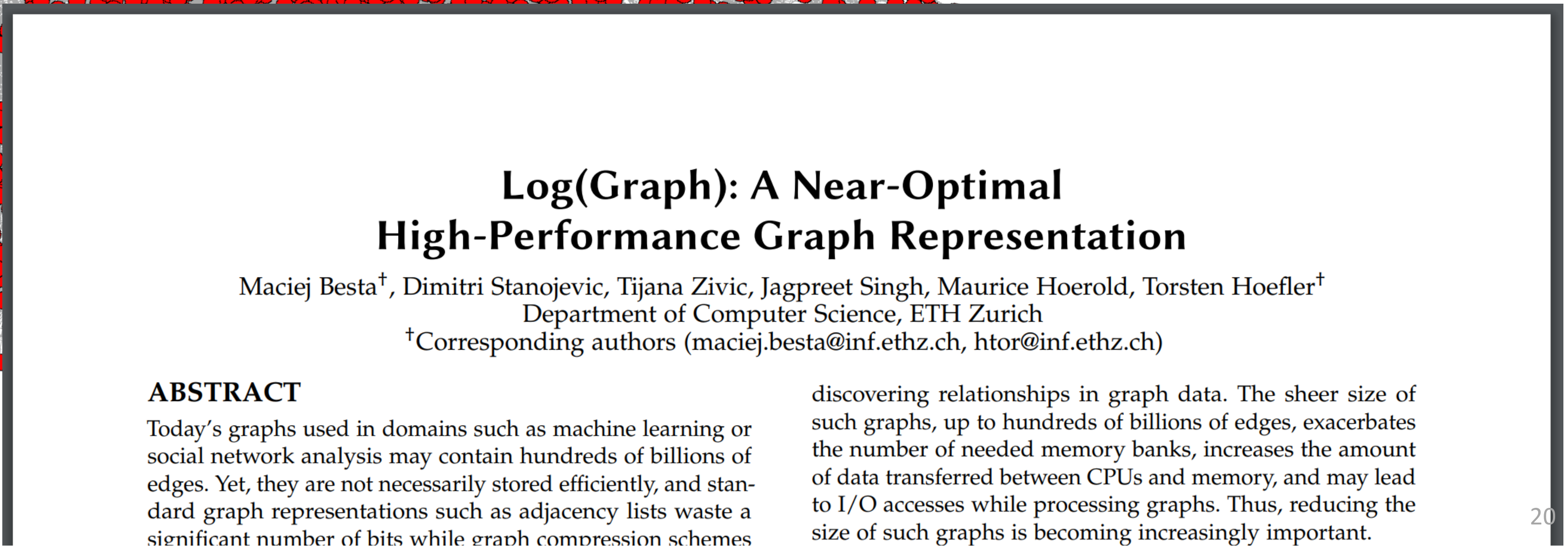


Graph	uku	gho	orm	tw	usrn	ema	am1
Generation of RB [m]	981.5	458.9	101.6	572.3	47.7	0.33	0.41
Generation of AA [m]	19.5	5.9	1.1	5.8	0.3	0.02	0.02

M. Besta et al.: “Log(Graph): A Near-Optimal High-Performance Graph Representation”, PACT’18

Problems!





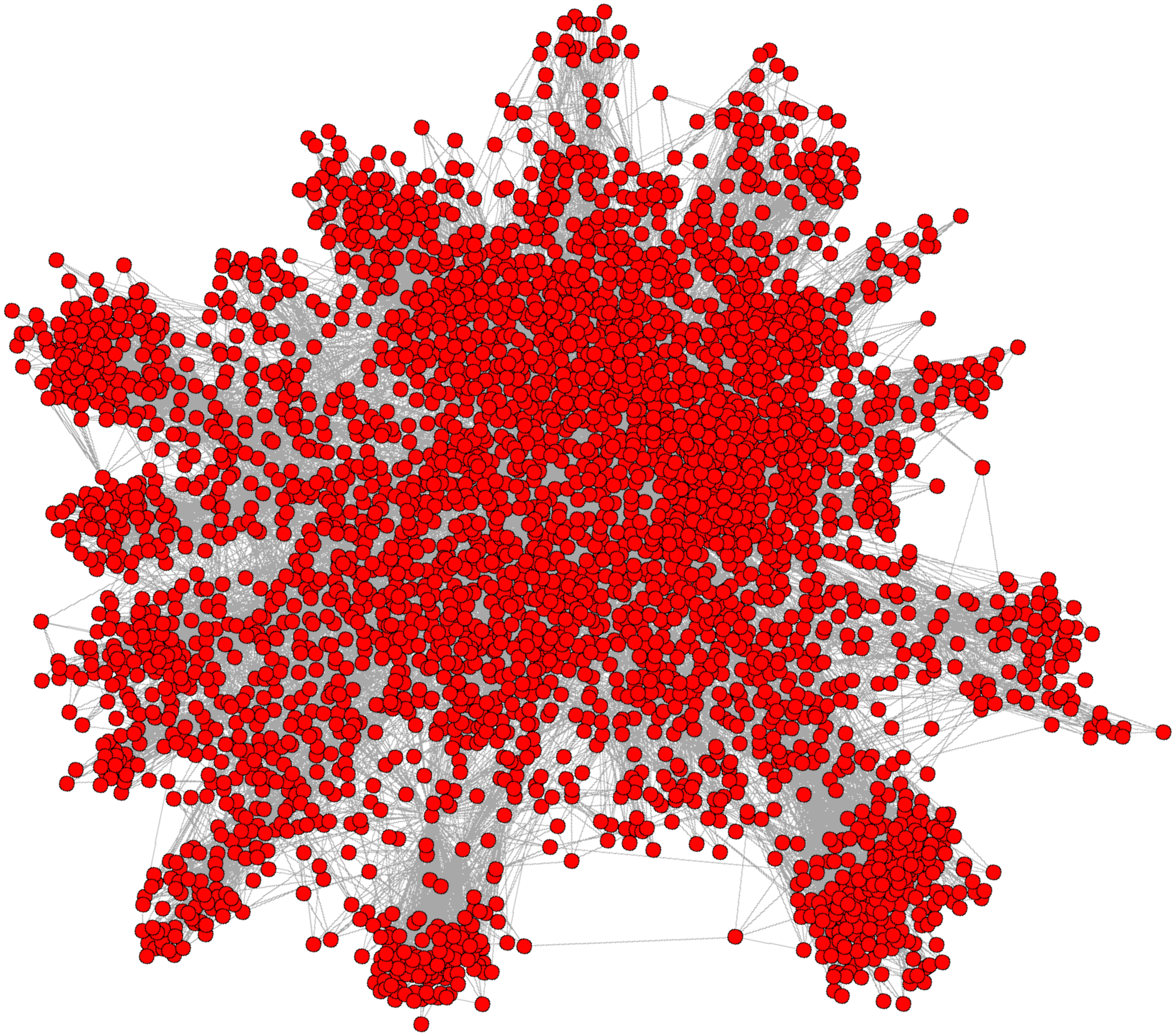
Log(Graph): A Near-Optimal High-Performance Graph Representation

Maciej Besta†, Dimitri Stanojevic, Tijana Zivic, Jagpreet Singh, Maurice Hoerold, Torsten Hoefler†
Department of Computer Science, ETH Zurich
†Corresponding authors (maciej.besta@inf.ethz.ch, htor@inf.ethz.ch)

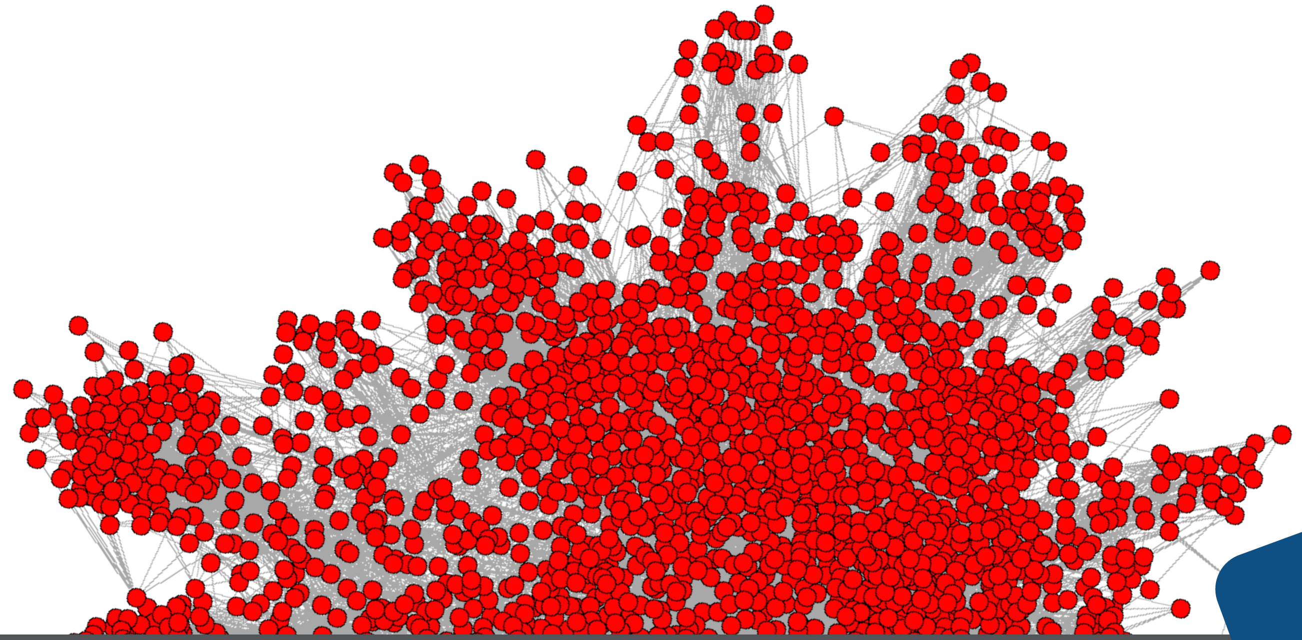
ABSTRACT

Today's graphs used in domains such as machine learning or social network analysis may contain hundreds of billions of edges. Yet, they are not necessarily stored efficiently, and standard graph representations such as adjacency lists waste a significant number of bits while graph compression schemes discovering relationships in graph data. The sheer size of such graphs, up to hundreds of billions of edges, exacerbates the number of needed memory banks, increases the amount of data transferred between CPUs and memory, and may lead to I/O accesses while processing graphs. Thus, reducing the size of such graphs is becoming increasingly important.

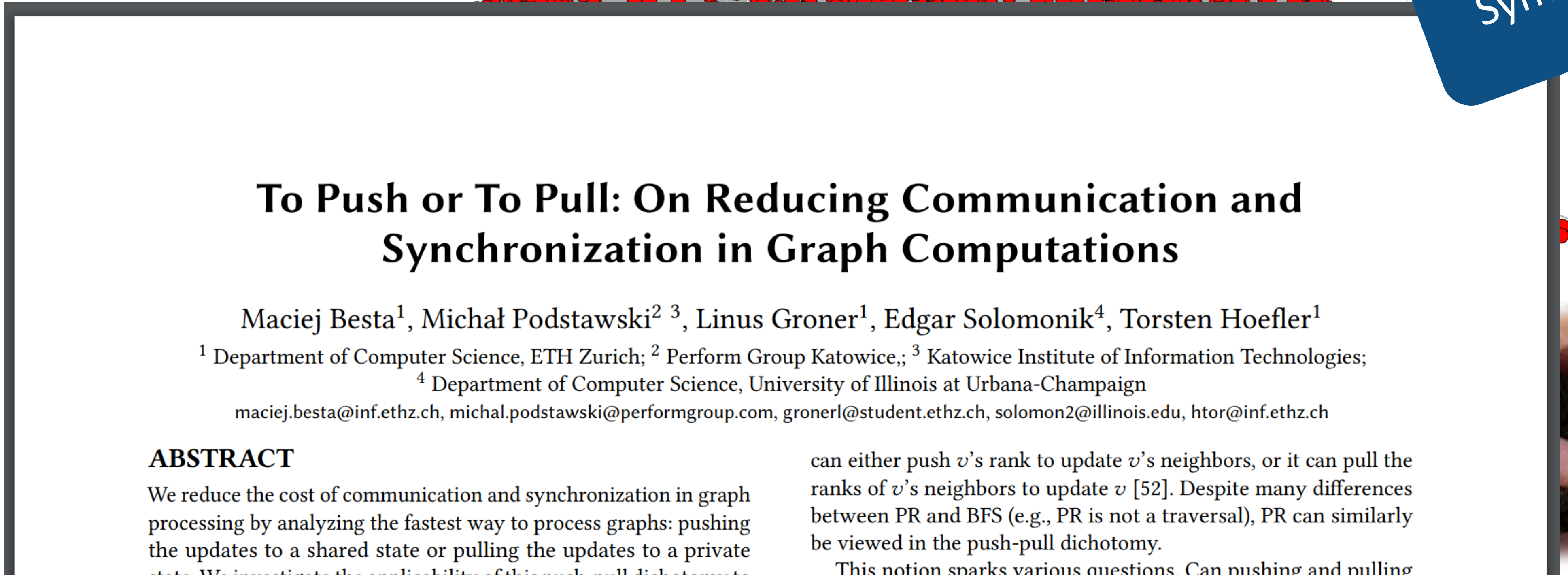
Problems!



Problems!



Synchronization-heavy



To Push or To Pull: On Reducing Communication and Synchronization in Graph Computations

Maciej Besta[1], Michał Podstawski[2 3], Linus Groner[1], Edgar Solomonik[4], Torsten Hoefler[1]
[1] Department of Computer Science, ETH Zurich; [2] Perform Group Katowice,; [3] Katowice Institute of Information Technologies;
[4] Department of Computer Science, University of Illinois at Urbana-Champaign
maciej.besta@inf.ethz.ch, michal.podstawski@performgroup.com, gronerl@student.ethz.ch, solomon2@illinois.edu, htor@inf.ethz.ch

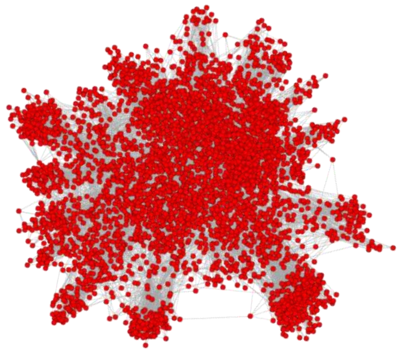
ABSTRACT
We reduce the cost of communication and synchronization in graph processing by analyzing the fastest way to process graphs: pushing the updates to a shared state or pulling the updates to a private

can either push v's rank to update v's neighbors, or it can pull the ranks of v's neighbors to update v [52]. Despite many differences between PR and BFS (e.g., PR is not a traversal), PR can similarly be viewed in the push-pull dichotomy.

This notion sparks various questions. Can pushing and pulling

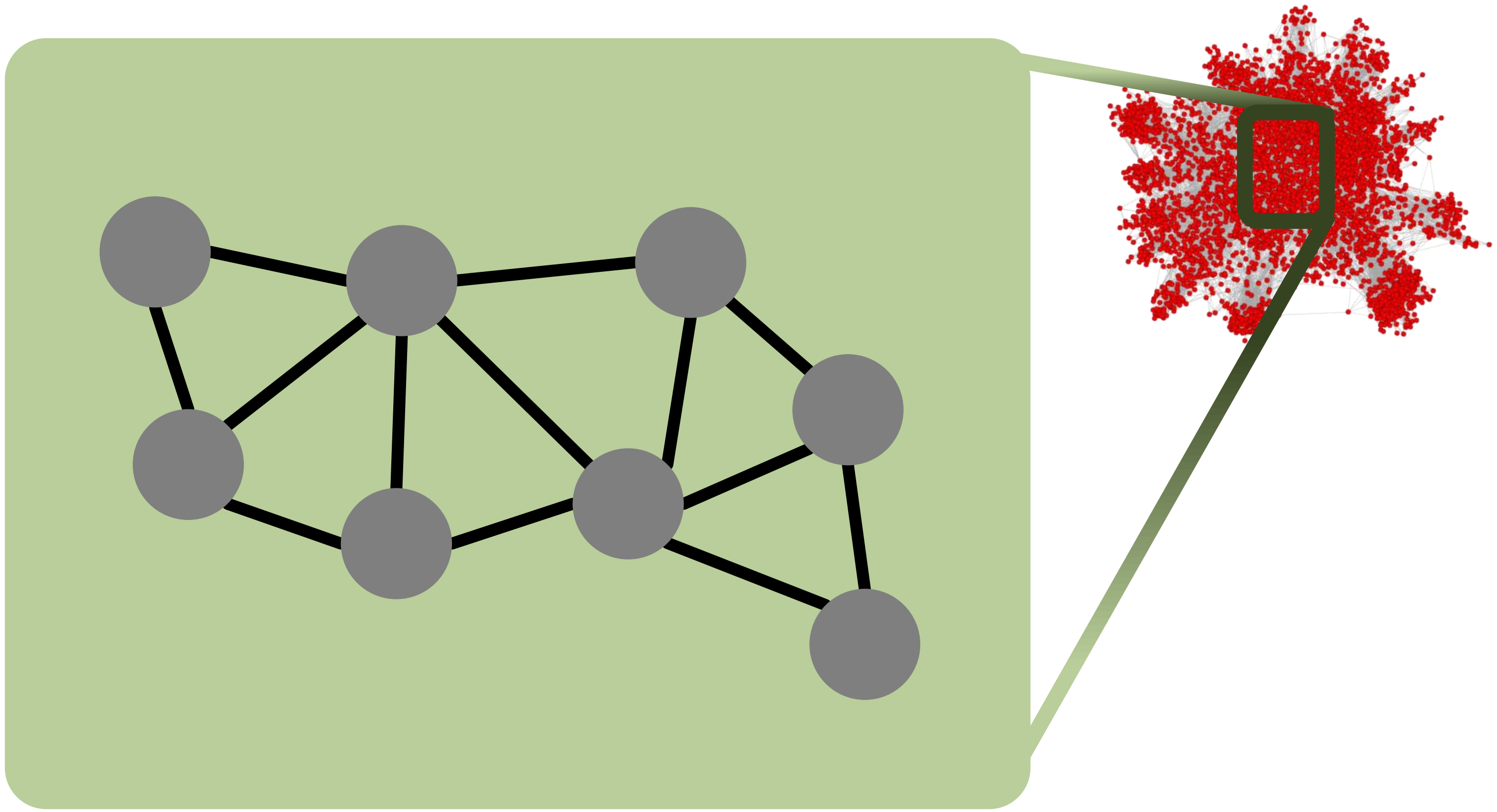
PageRank

PAGERANK



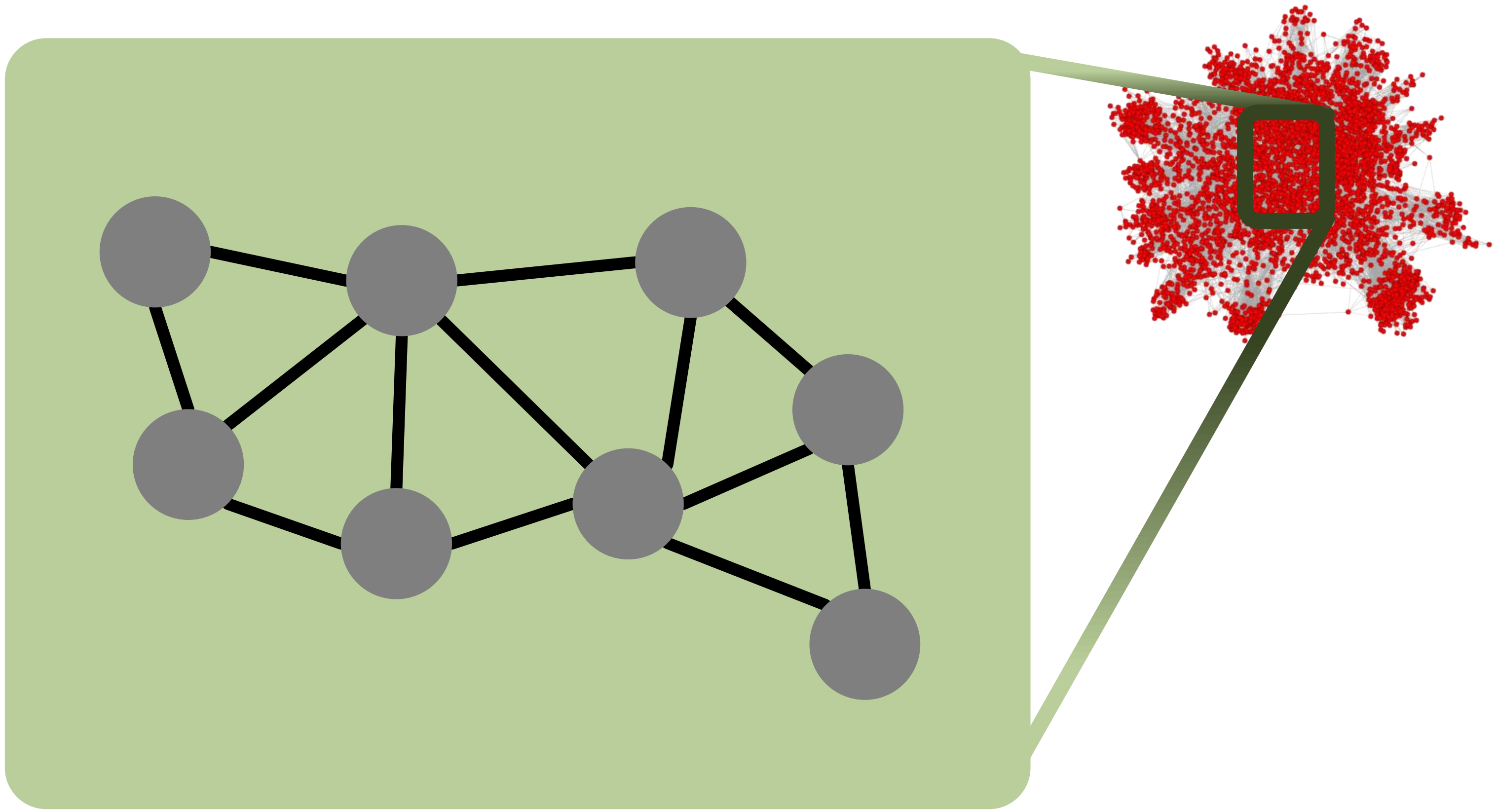
PageRank

PAGERANK



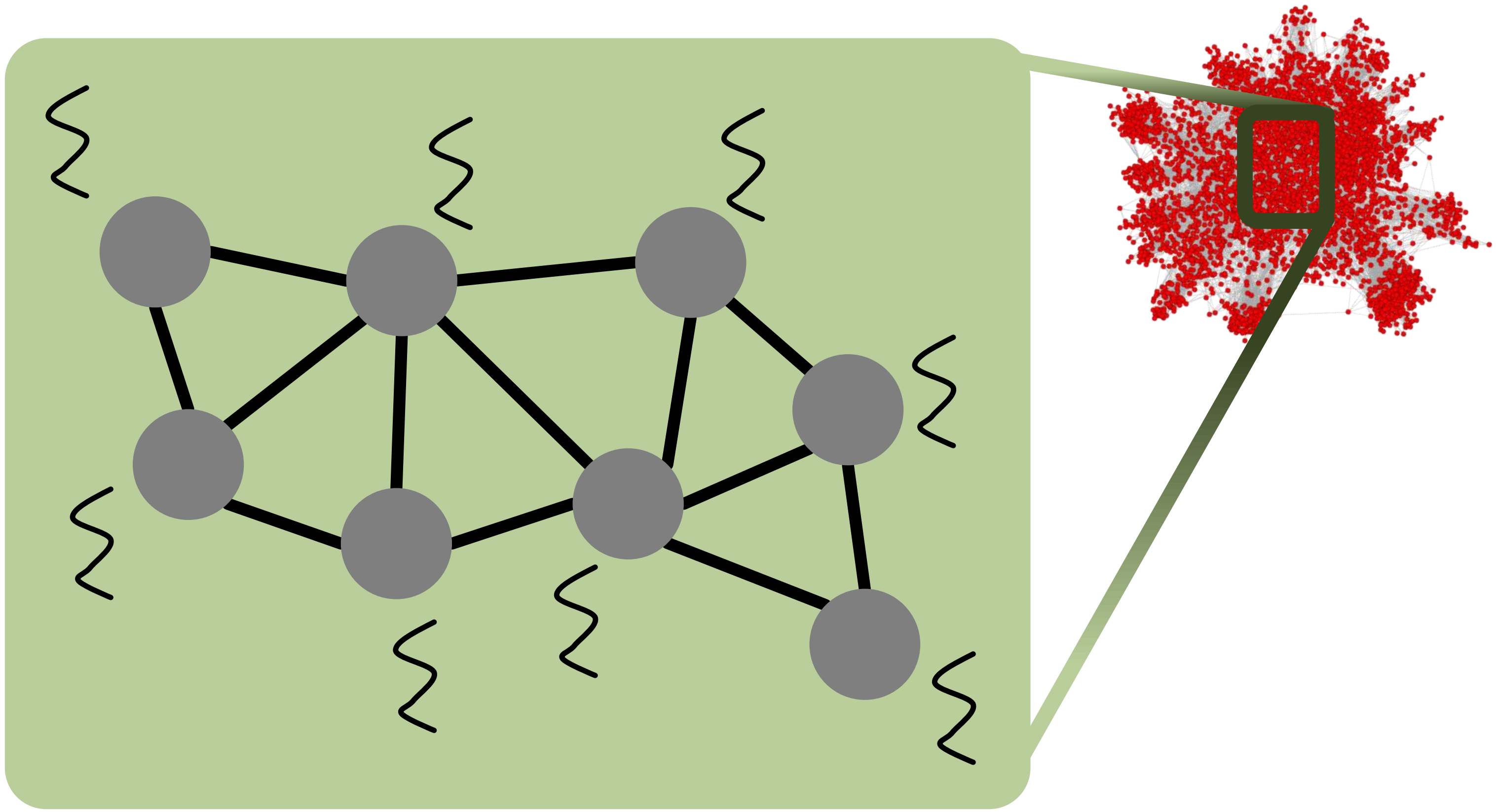
PAGERANK

P threads are used



PAGERANK

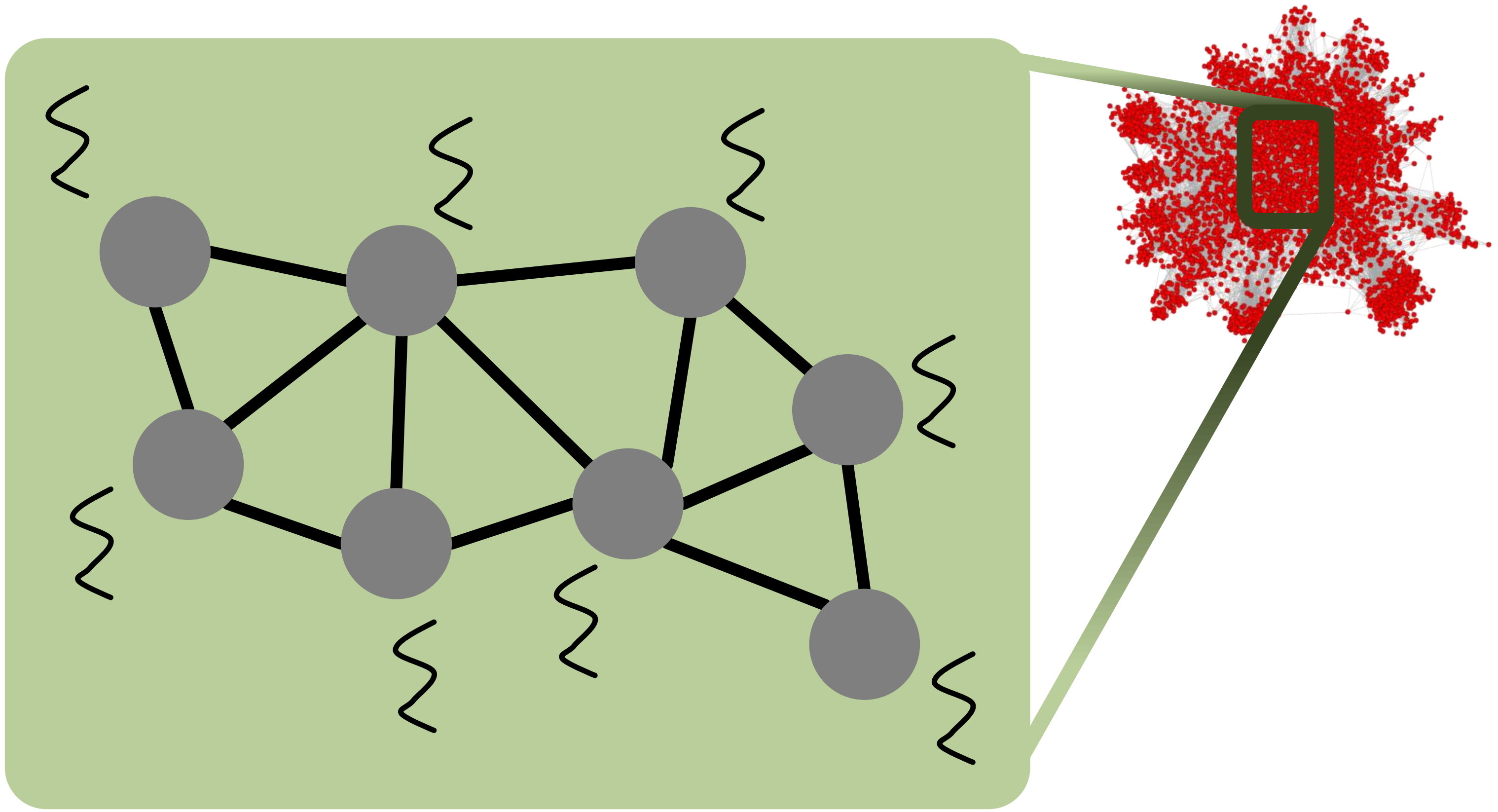
P threads are used



[1] J. J. Whang et al. Scalable Data-Driven PageRank: Algorithms, System Issues, and Lessons Learned. Euro-Par 2015.

PAGERANK

P threads are used

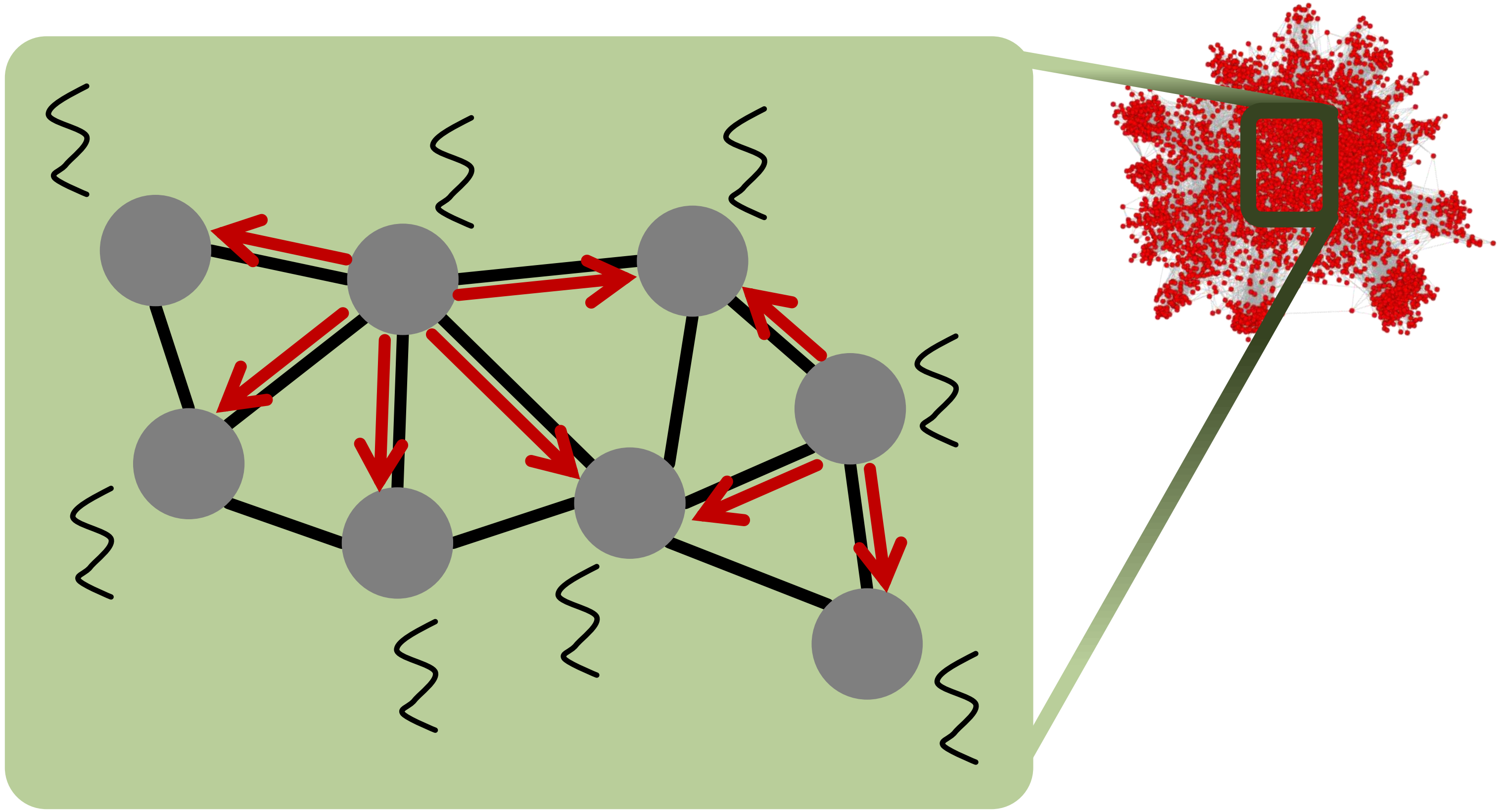


Pushing

[1] J. J. Whang et al. Scalable Data-Driven PageRank: Algorithms, System Issues, and Lessons Learned. Euro-Par 2015.

PAGERANK

P threads are used

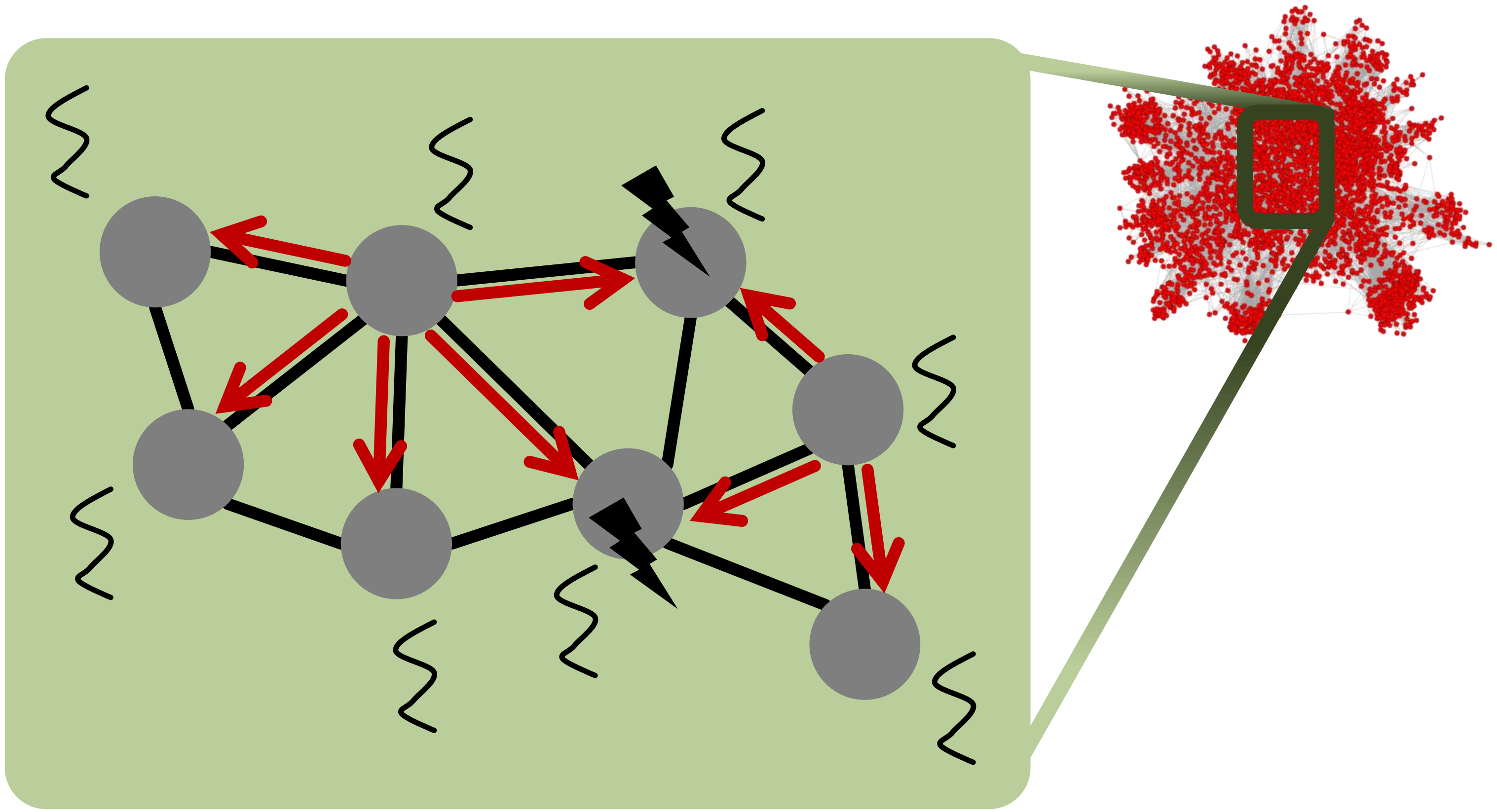


Pushing

[1] J. J. Whang et al. Scalable Data-Driven PageRank: Algorithms, System Issues, and Lessons Learned. Euro-Par 2015.

PAGERANK

P threads are used



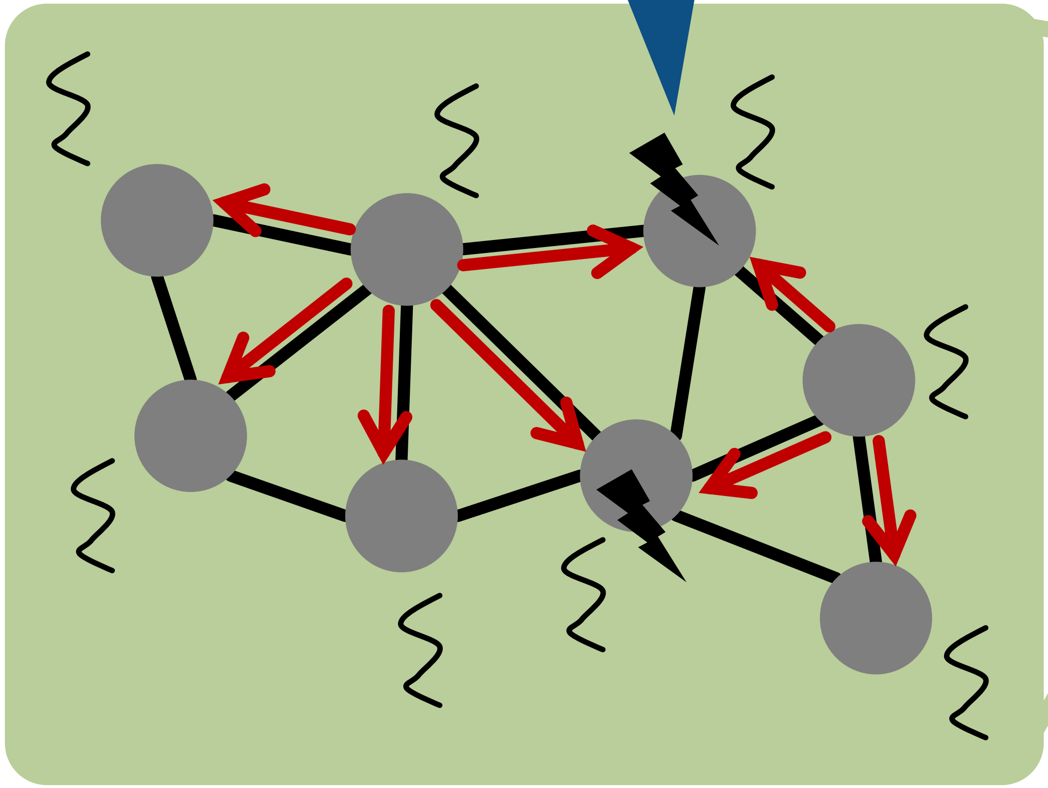
Pushing

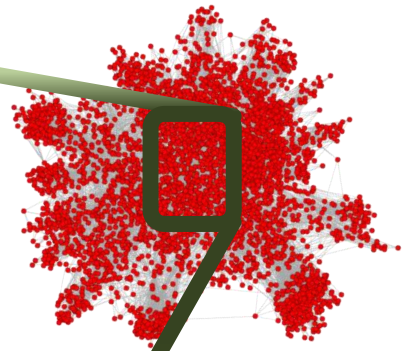
[1] J. J. Whang et al. Scalable Data-Driven PageRank: Algorithms, System Issues, and Lessons Learned. Euro-Par 2015.

PAGERANK

Write conflicts

P threads are used



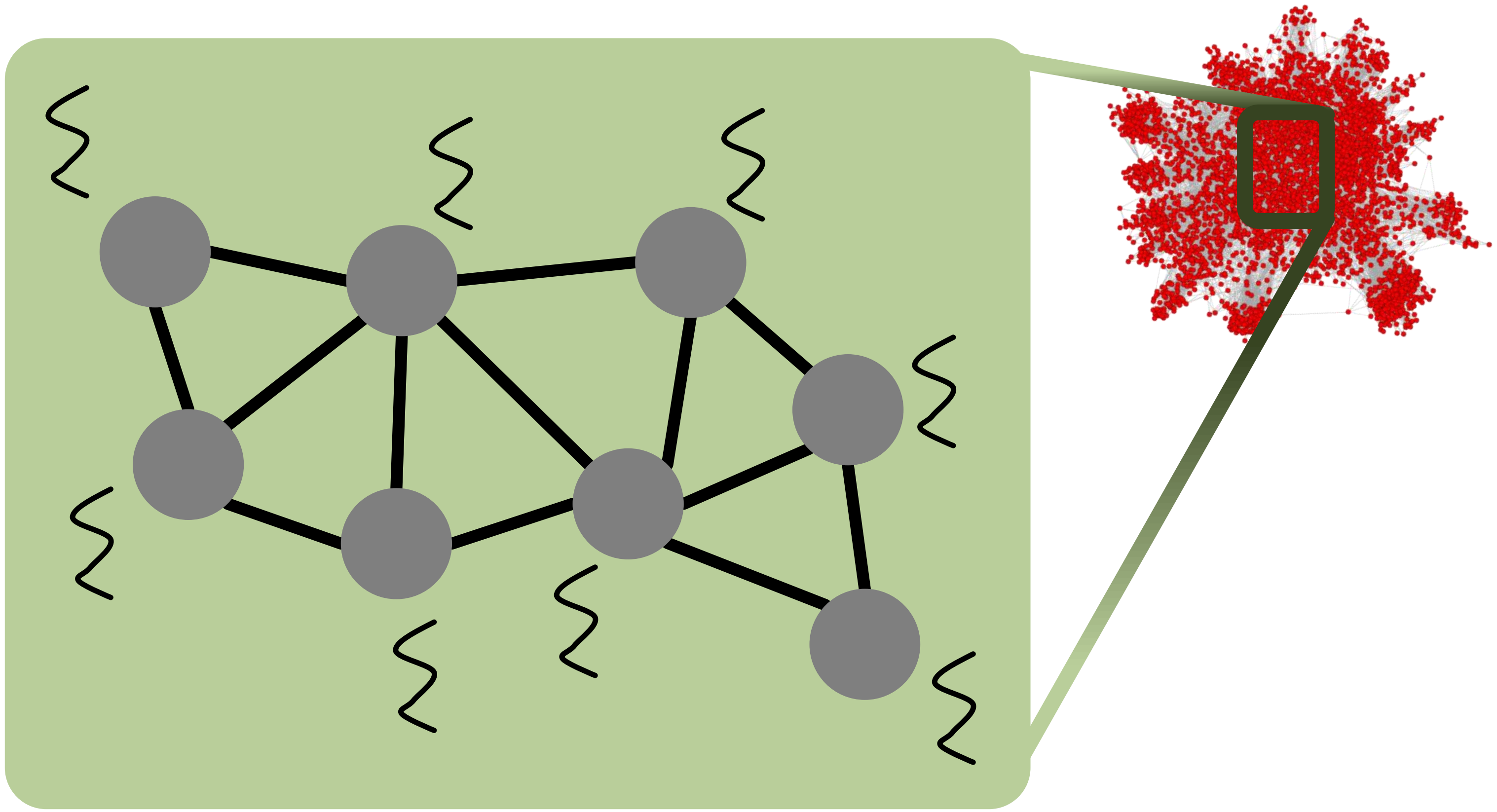


Pushing

[1] J. J. Whang et al. Scalable Data-Driven PageRank: Algorithms, System Issues, and Lessons Learned. Euro-Par 2015.

PageRank

P threads are used

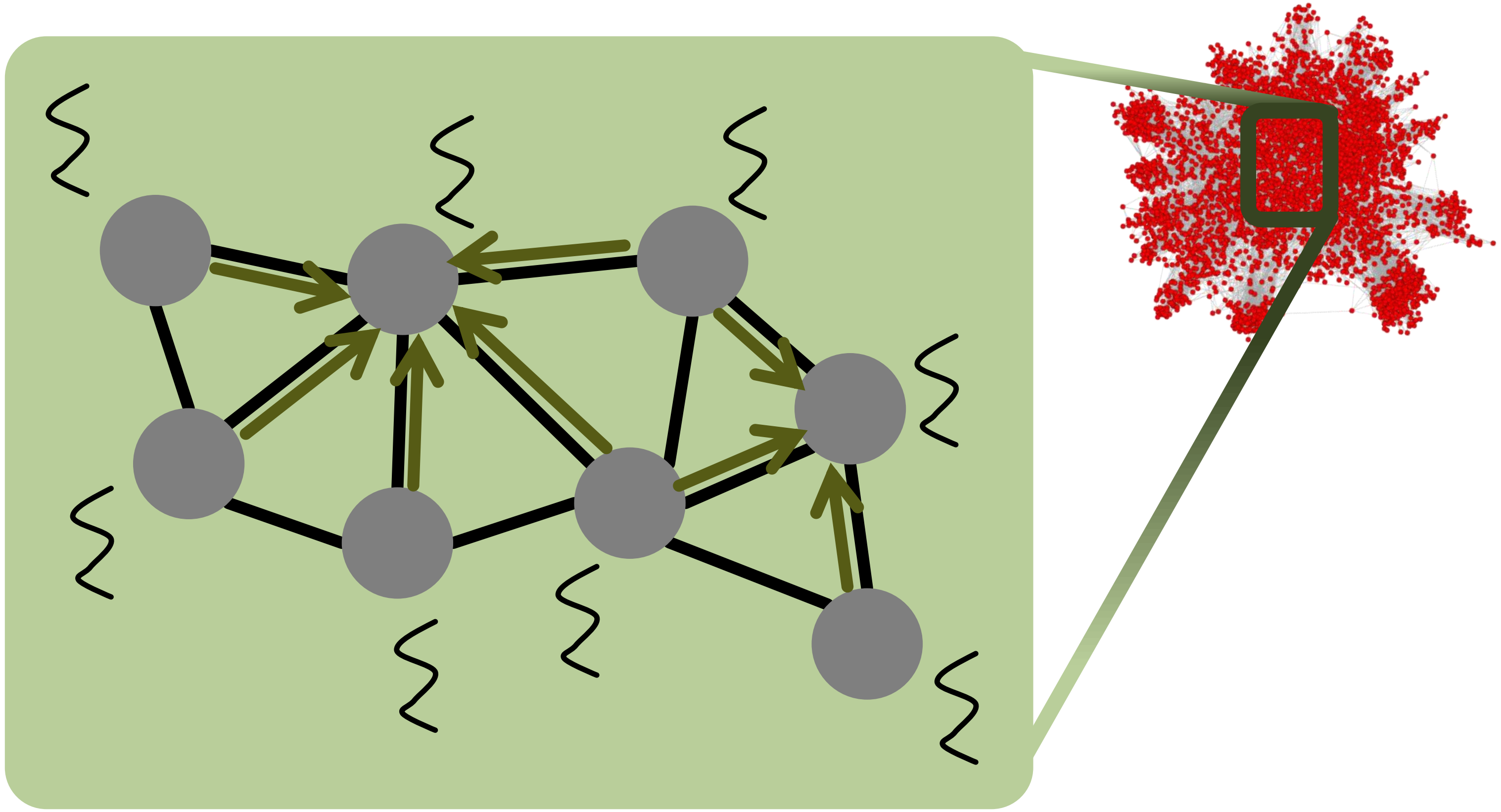


Pulling

[1] J. J. Whang et al. Scalable Data-Driven PageRank: Algorithms, System Issues, and Lessons Learned. Euro-Par 2015.

PageRank

P threads are used

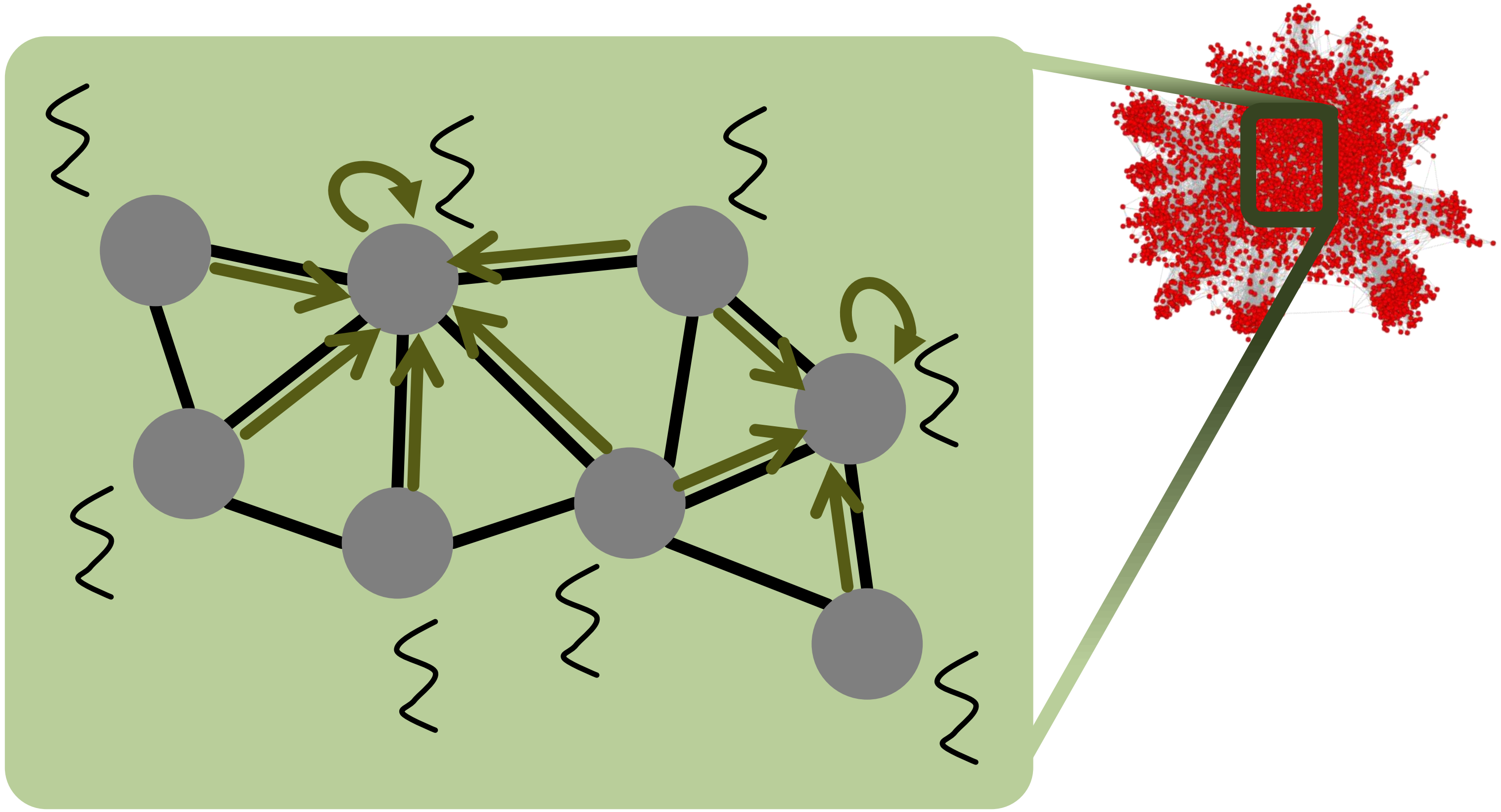


Pulling

[1] J. J. Whang et al. Scalable Data-Driven PageRank: Algorithms, System Issues, and Lessons Learned. Euro-Par 2015.

PAGERANK

P threads are used

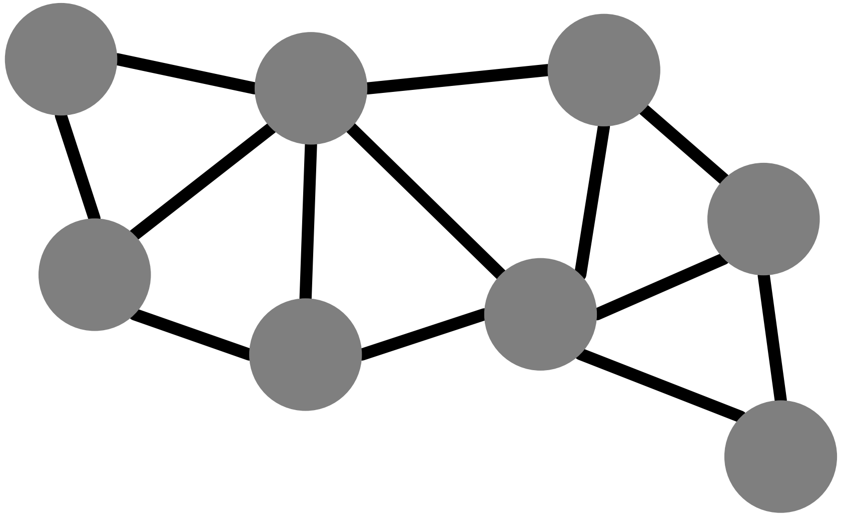


Pulling

[1] J. J. Whang et al. Scalable Data-Driven PageRank: Algorithms, System Issues, and Lessons Learned. Euro-Par 2015.

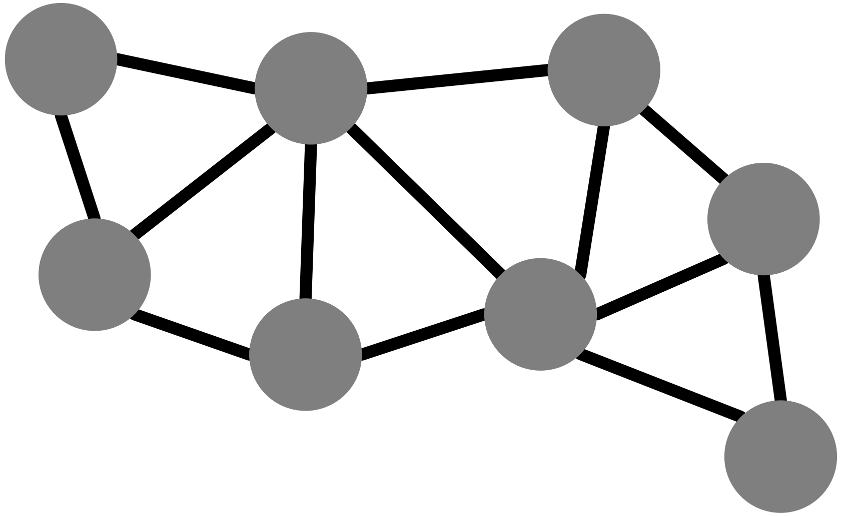
BFS

TOP-DOWN VS. BOTTOM-UP [1]



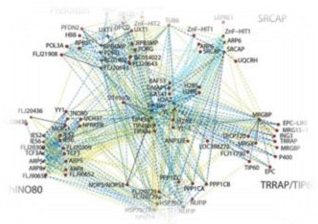
BFS

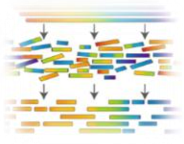
Top-Down vs. Bottom-Up [1]

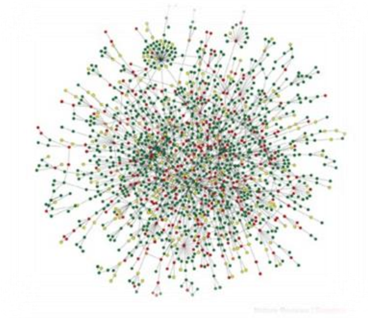


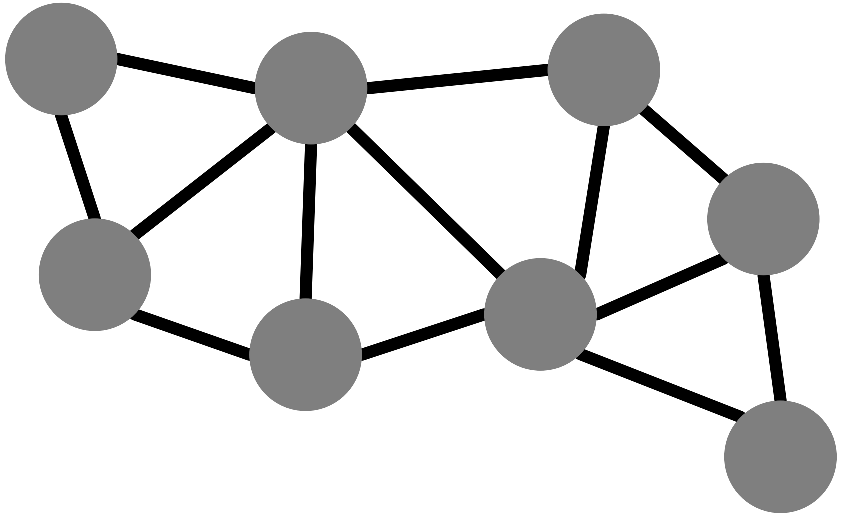
BFS

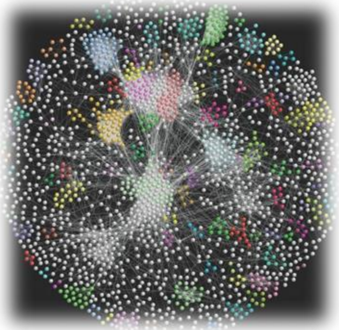
Top-Down vs. Bottom-Up [1]



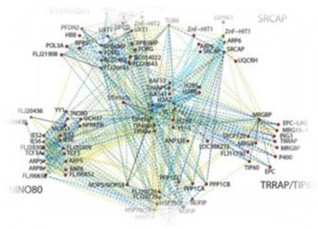


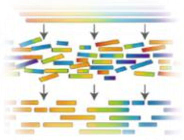


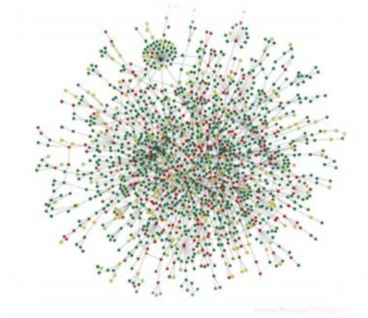


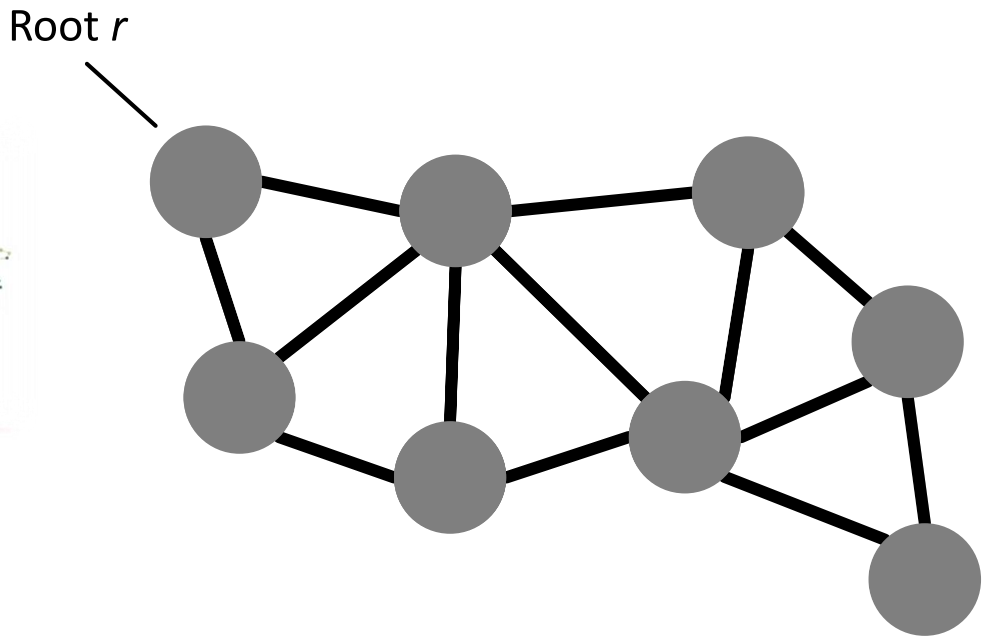


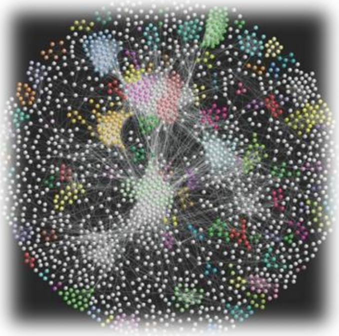
BFS
Top-Down vs. Bottom-Up [1]



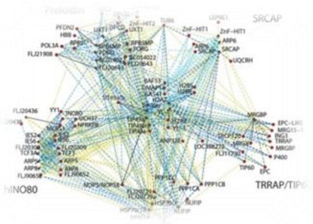


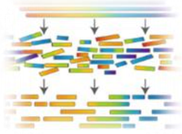


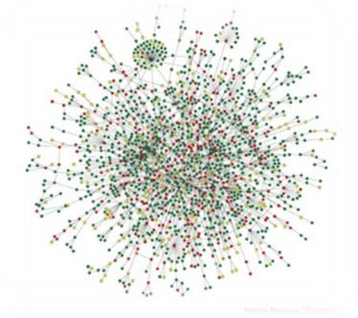


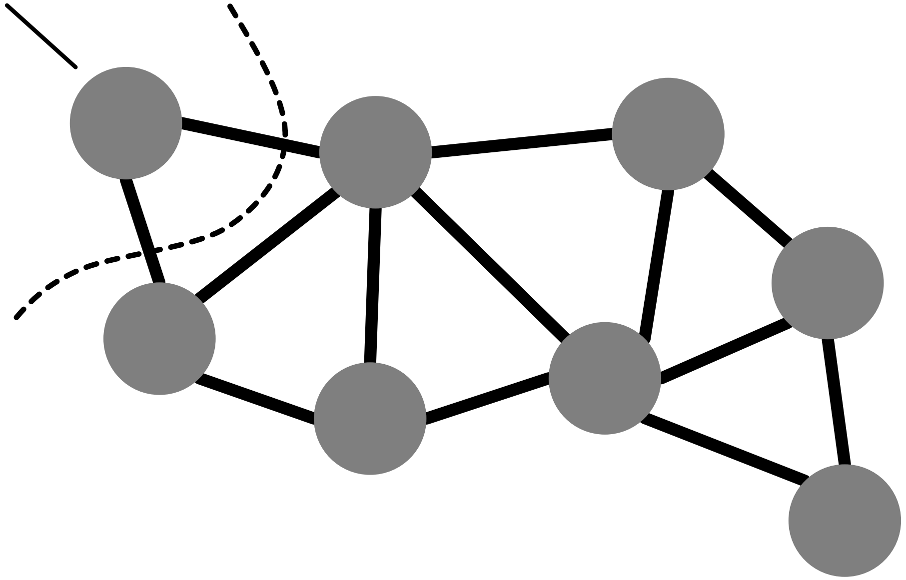


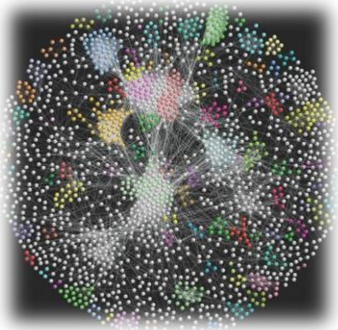
BFS
Top-Down vs. Bottom-Up [1]



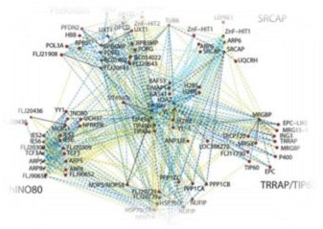


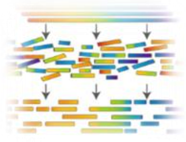


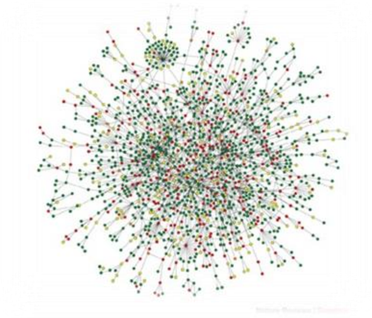


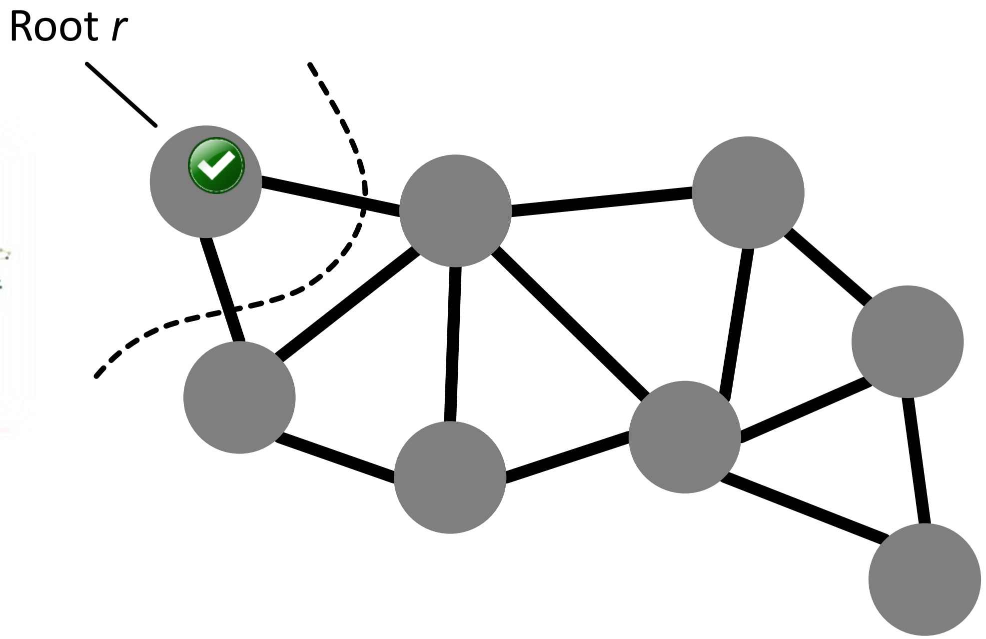


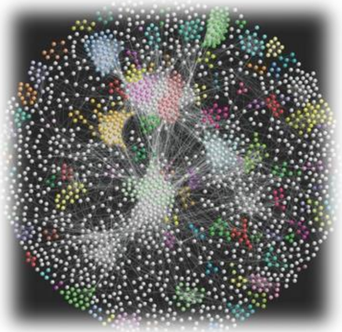
BFS
Top-Down vs. Bottom-Up [1]



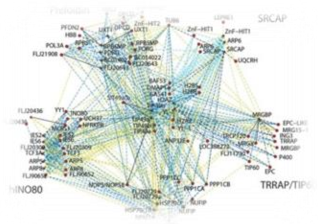


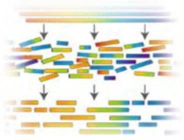


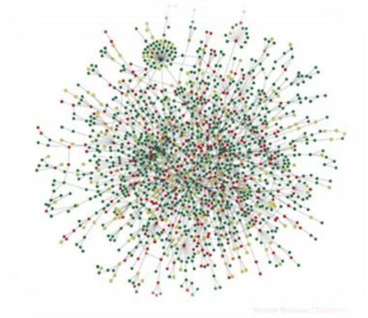


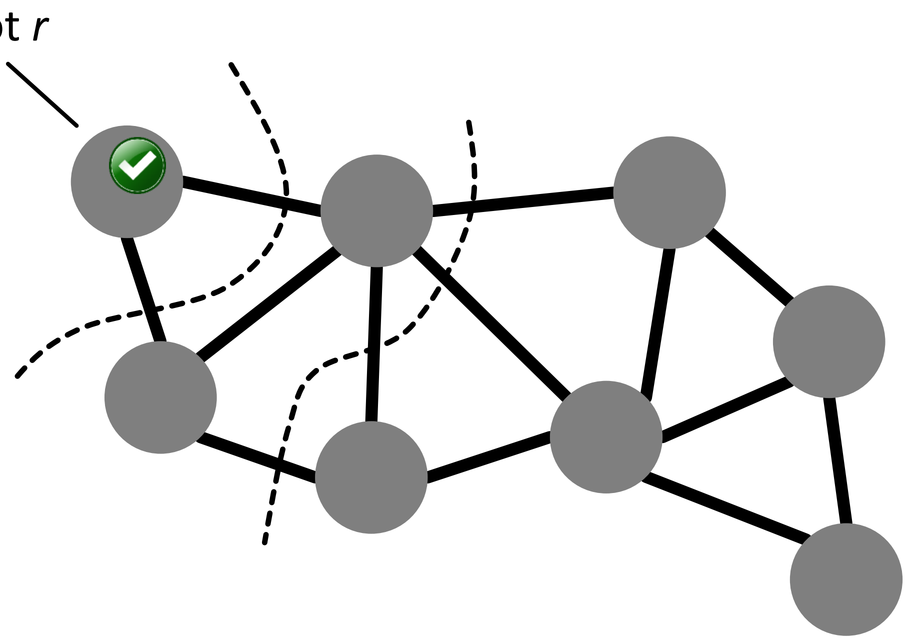


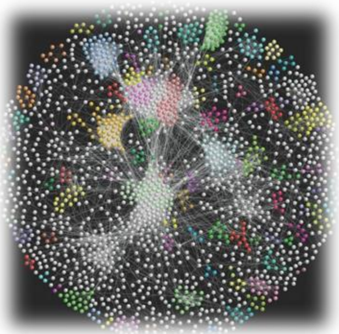
BFS
Top-Down vs. Bottom-Up [1]





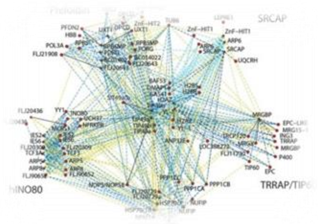


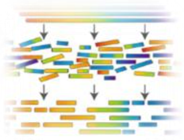


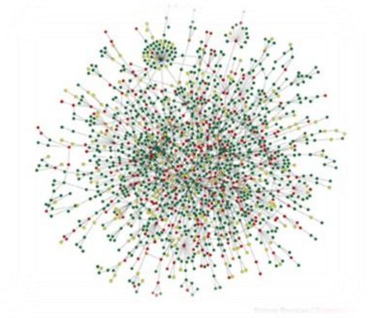


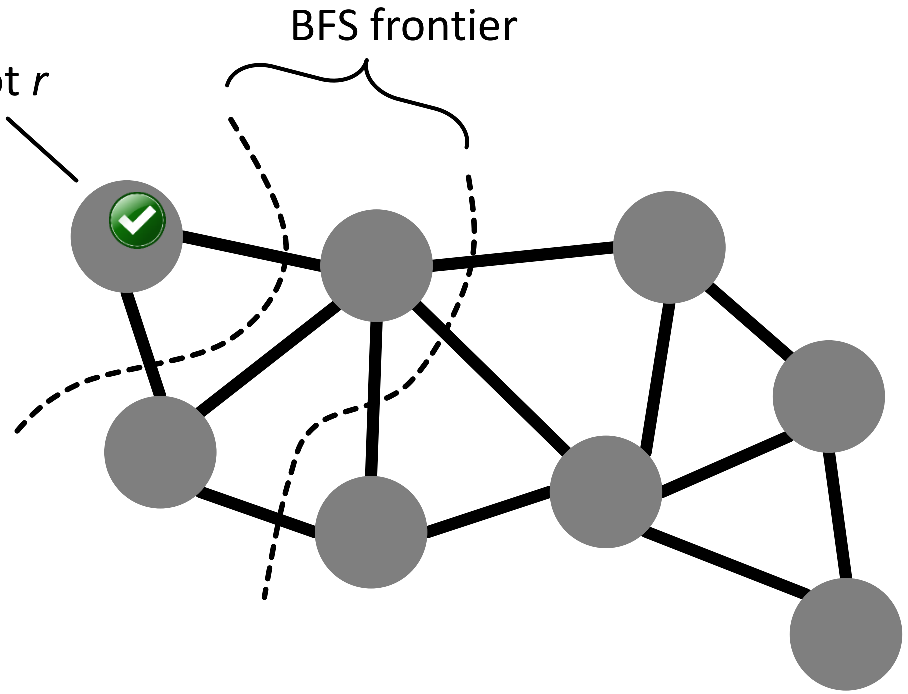
BFS

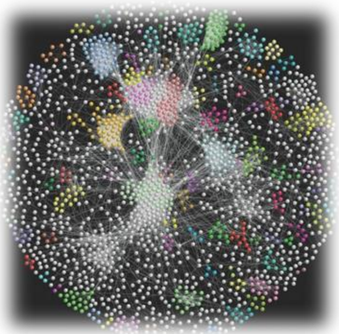
Top-Down vs. Bottom-Up [1]



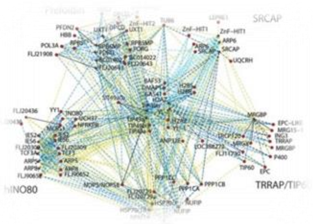


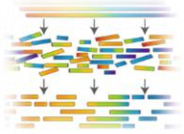


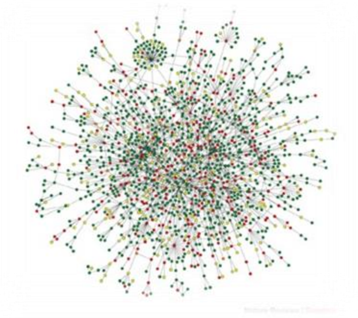


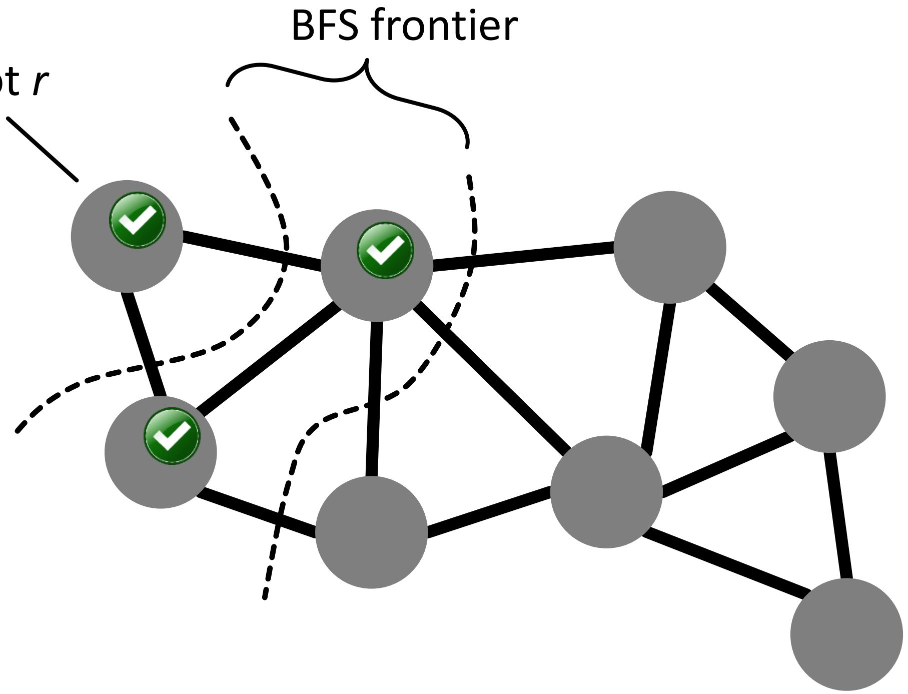


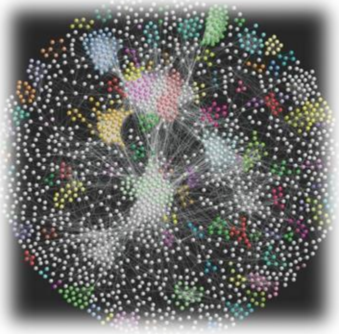
BFS
Top-Down vs. Bottom-Up [1]



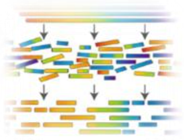


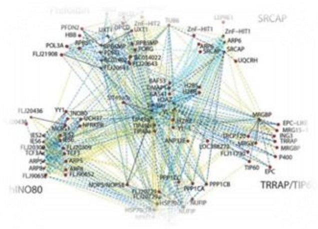


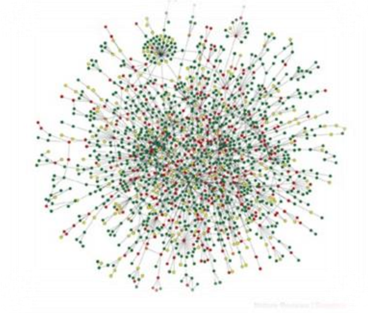


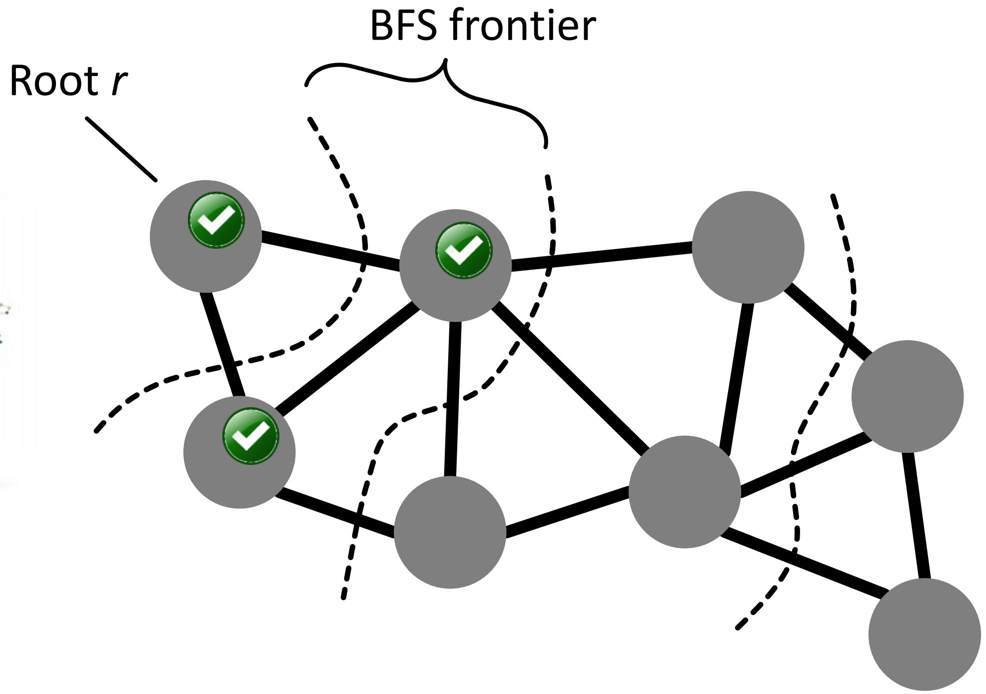


BFS
Top-Down vs. Bottom-Up [1]

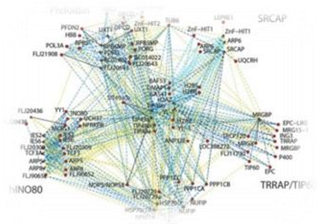


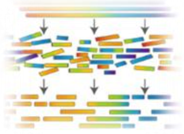


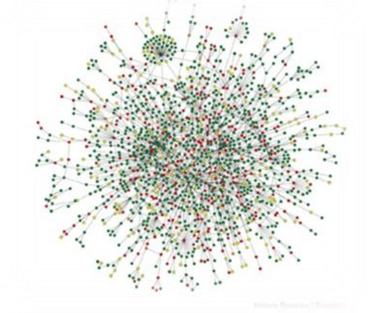


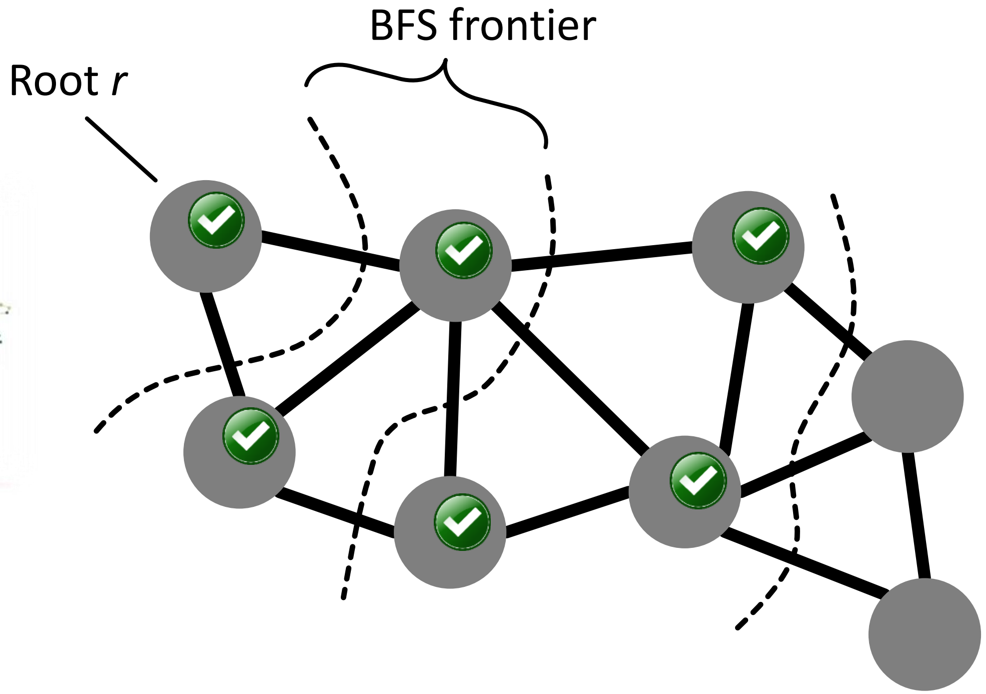


BFS
Top-Down vs. Bottom-Up [1]



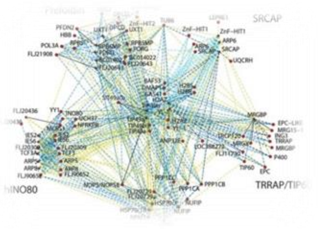


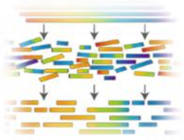


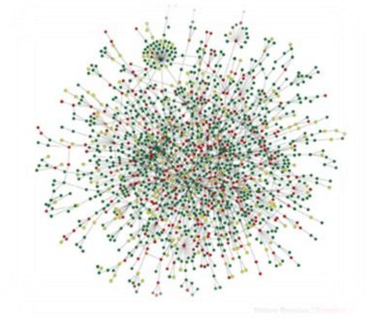


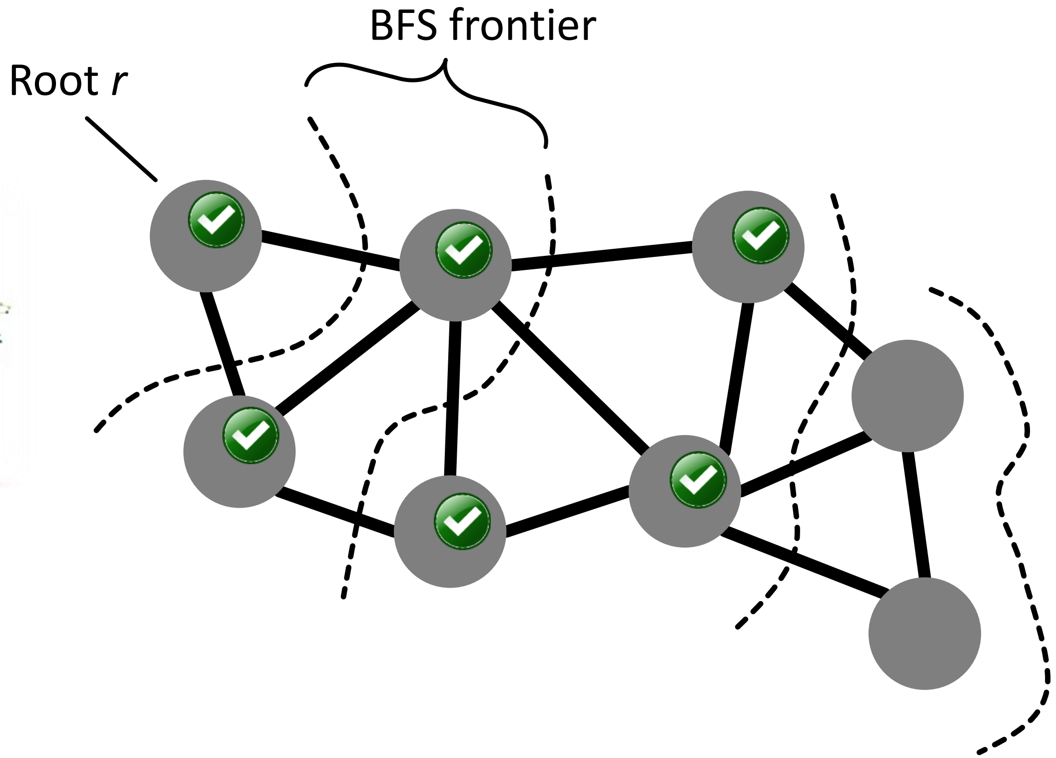
BFS

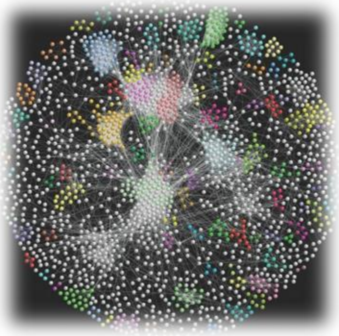
Top-Down vs. Bottom-Up [1]



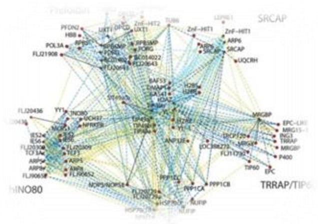


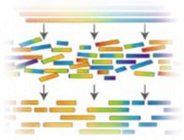


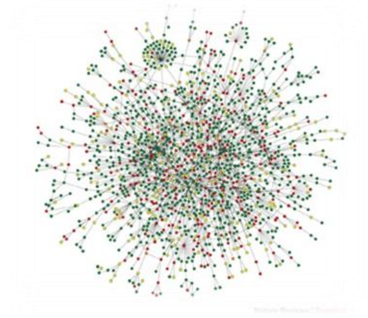


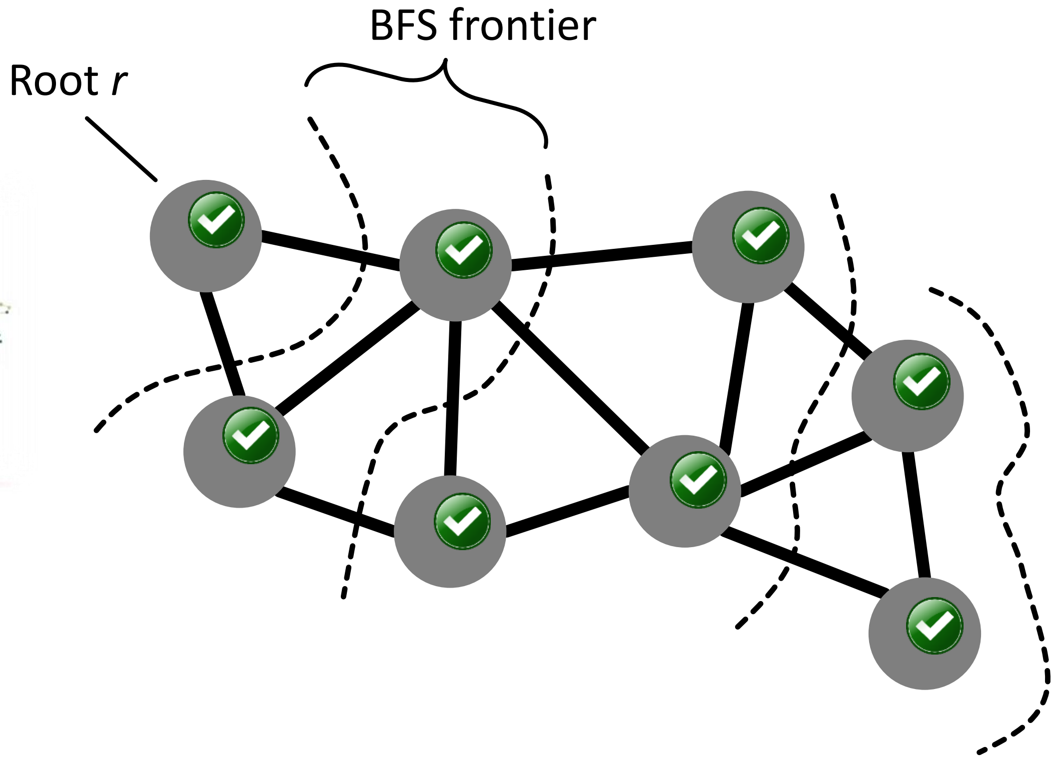


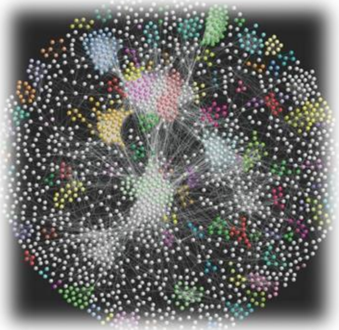
BFS
Top-Down vs. Bottom-Up [1]



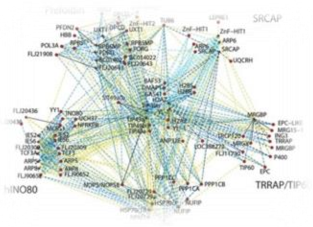


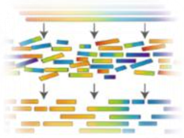


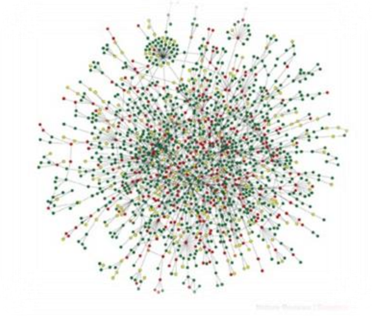


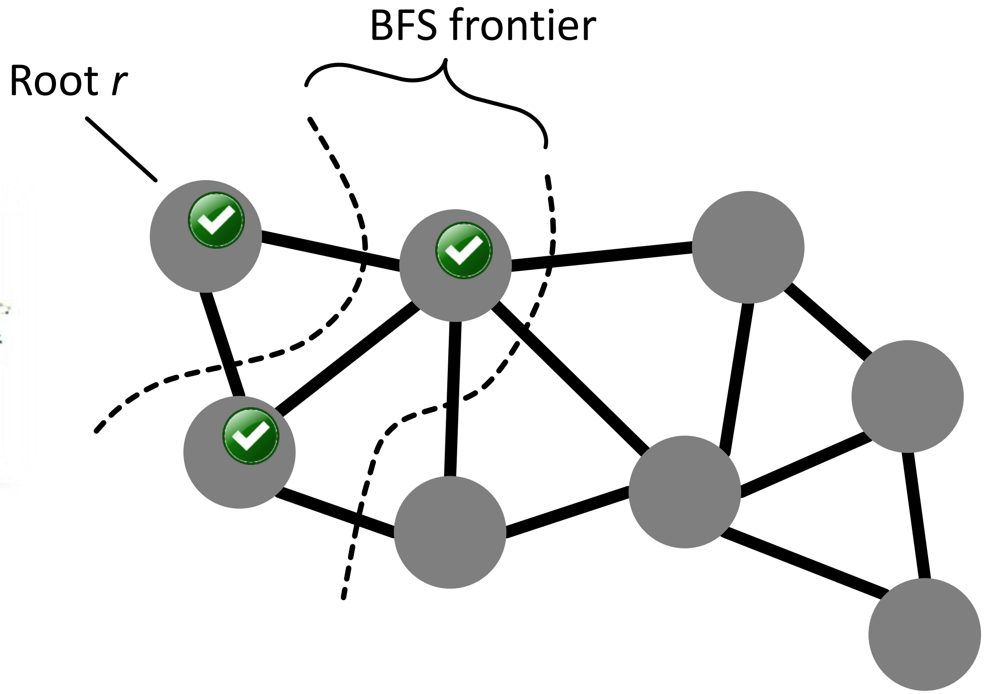


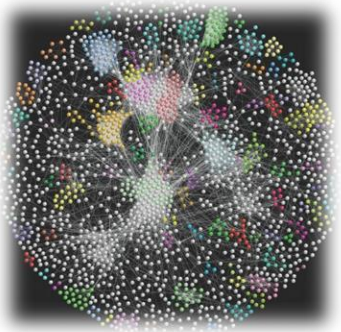
BFS
Top-Down vs. Bottom-Up [1]





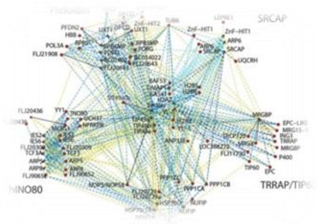


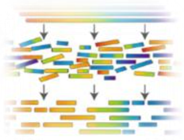




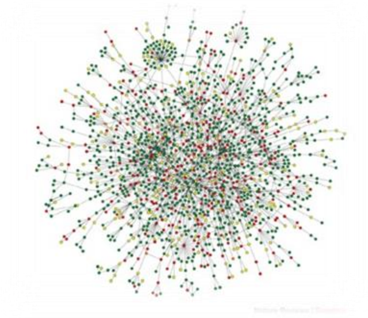
BFS

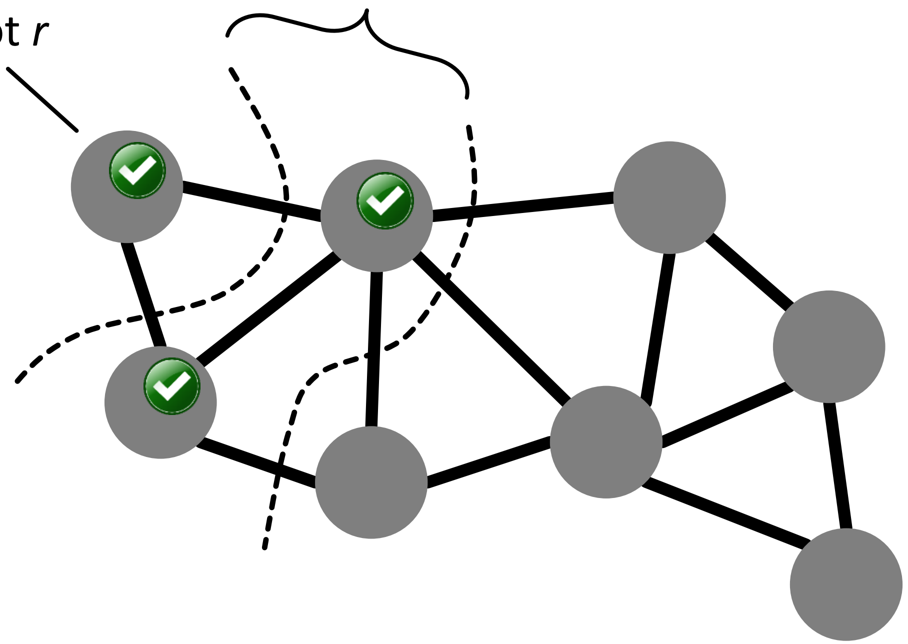
Top-Down vs. Bottom-Up [1]

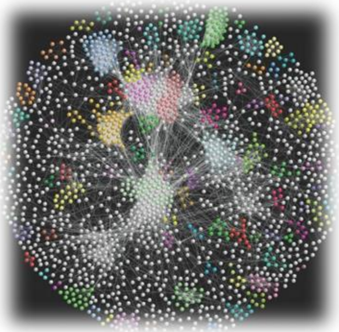




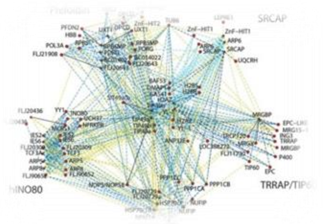
Pushing or pulling when expanding a frontier

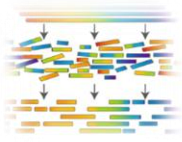




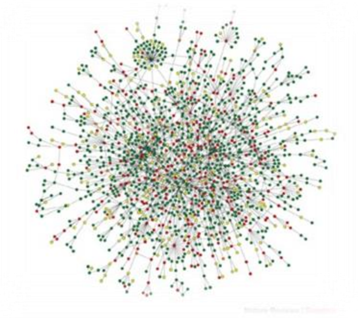


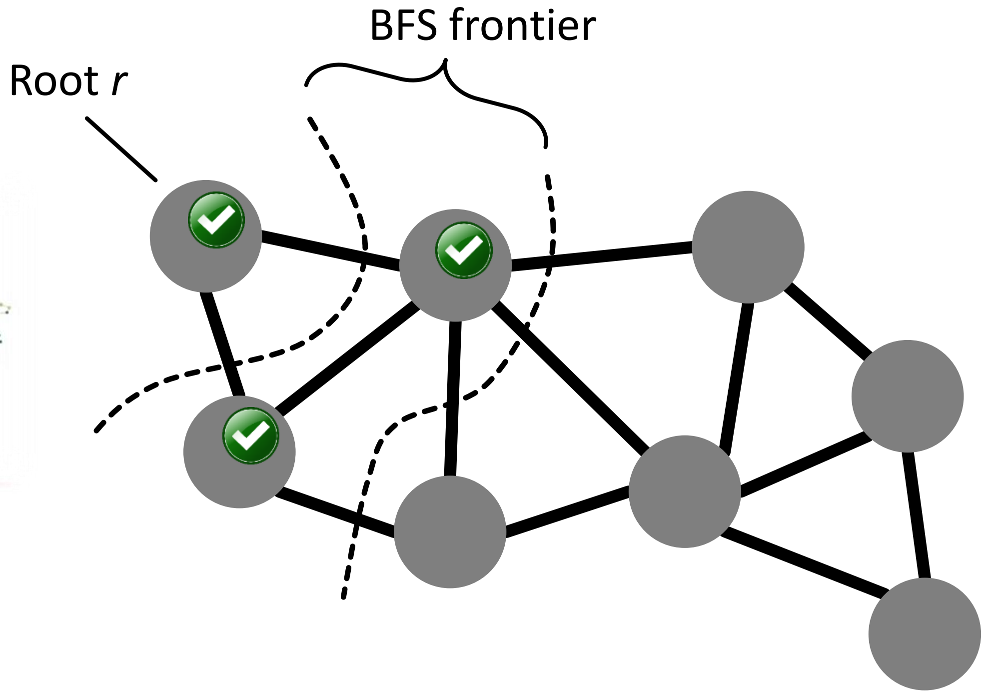
BFS
Top-Down vs. Bottom-Up [1]

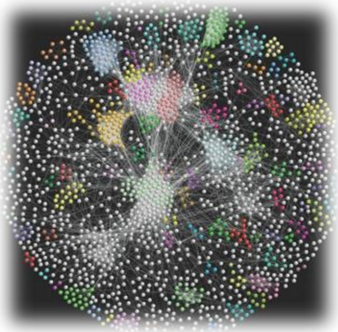




Pushing or pulling when expanding a frontier



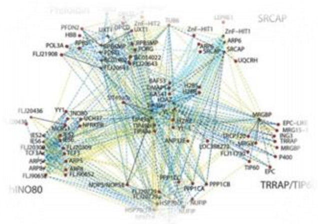


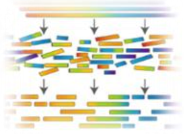


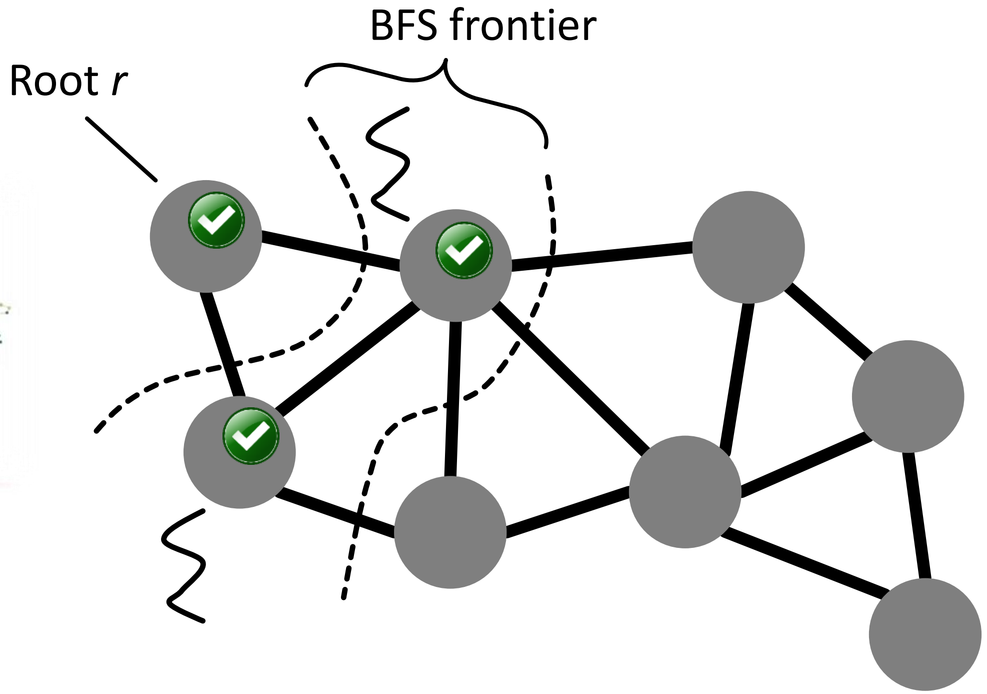
Pushing

[1] S. Beamer, K. Asanović, and D. Patterson. Direction-optimizing breadth-first search. SC12.

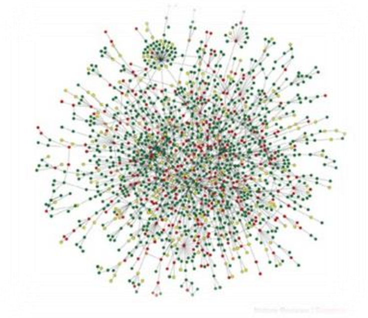
BFS
Top-Down vs. Bottom-Up [1]

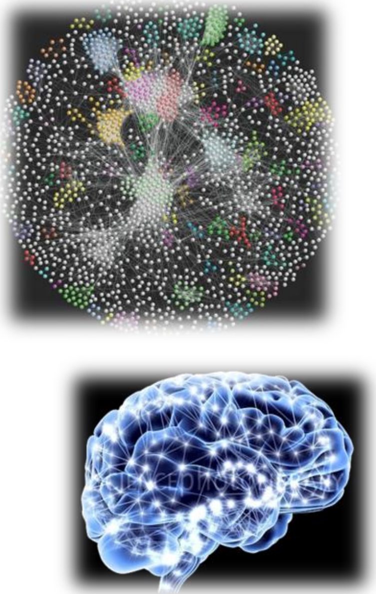






Pushing or pulling when expanding a frontier



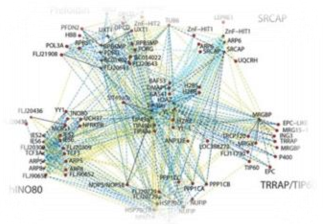


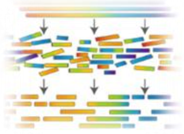
Pushing

[1] S. Beamer, K. Asanović, and D. Patterson. Direction-optimizing breadth-first search. SC12.

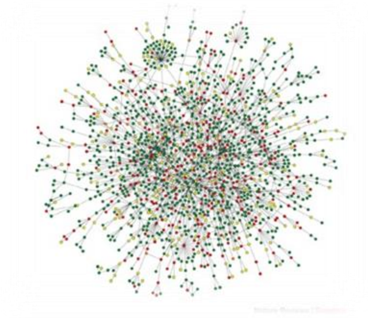
BFS

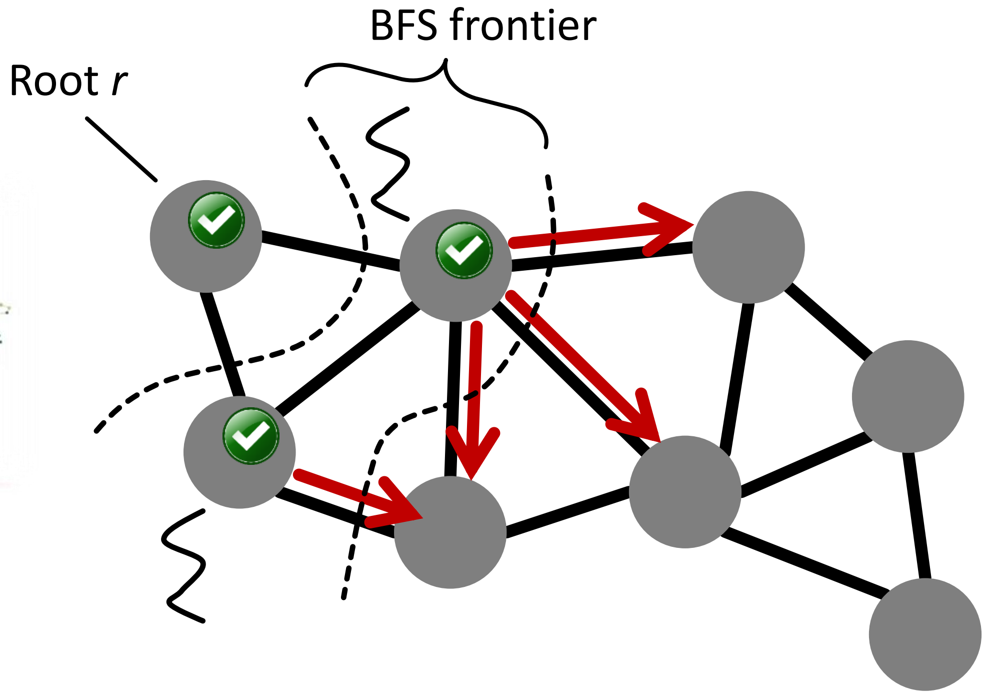
TOP-DOWN VS. BOTTOM-UP [1]

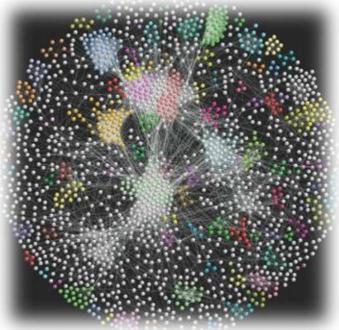




Pushing or pulling when expanding a frontier





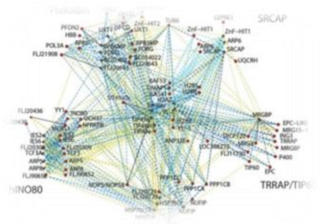


Pushing

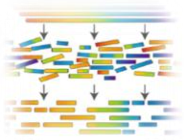
[1] S. Beamer, K. Asanović, and D. Patterson. Direction-optimizing breadth-first search. SC12.

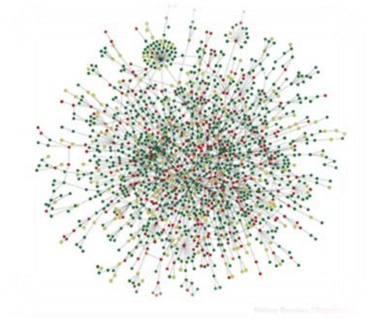
BFS

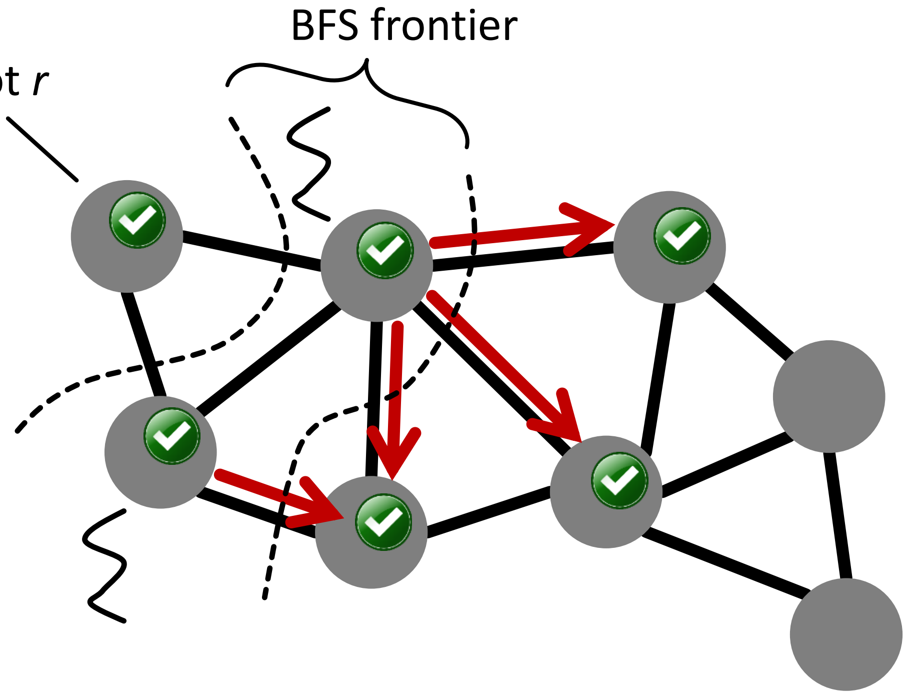
TOP-DOWN VS. BOTTOM-UP [1]

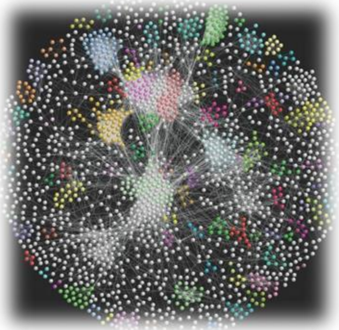


Pushing or pulling when expanding a frontier







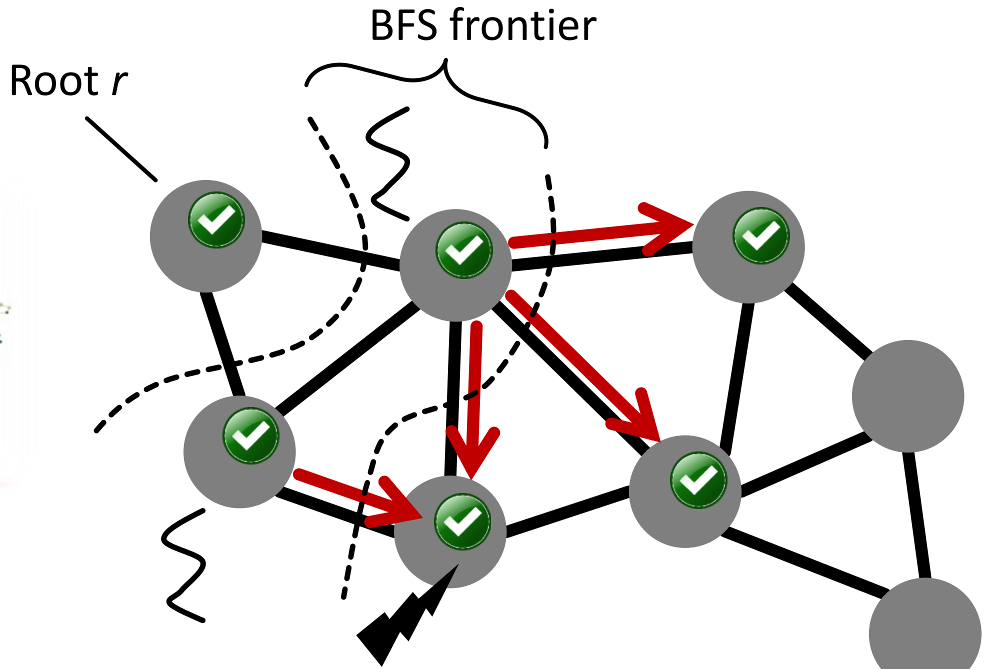


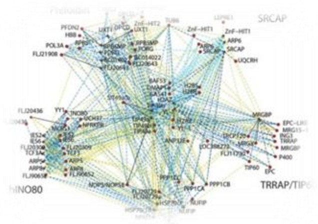
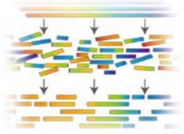
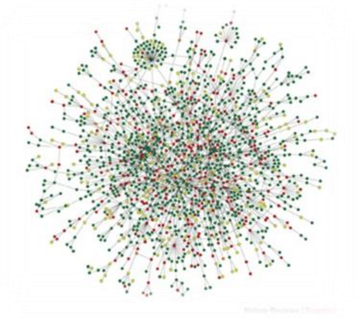
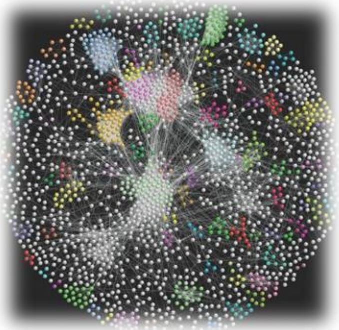
Pushing

[1] S. Beamer, K. Asanović, and D. Patterson. Direction-optimizing breadth-first search. SC12.

BFS
TOP-DOWN VS. BOTTOM-UP [1]

Pushing or pulling when expanding a frontier



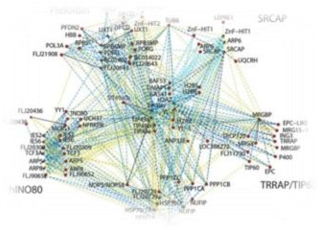





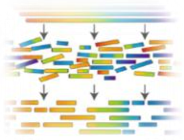

Pushing

[1] S. Beamer, K. Asanović, and D. Patterson. Direction-optimizing breadth-first search. SC12.

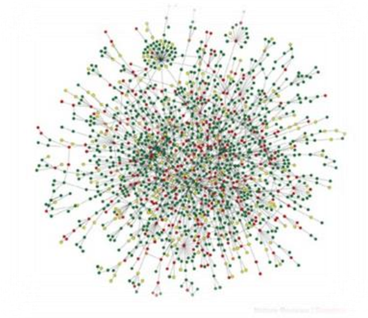
BFS

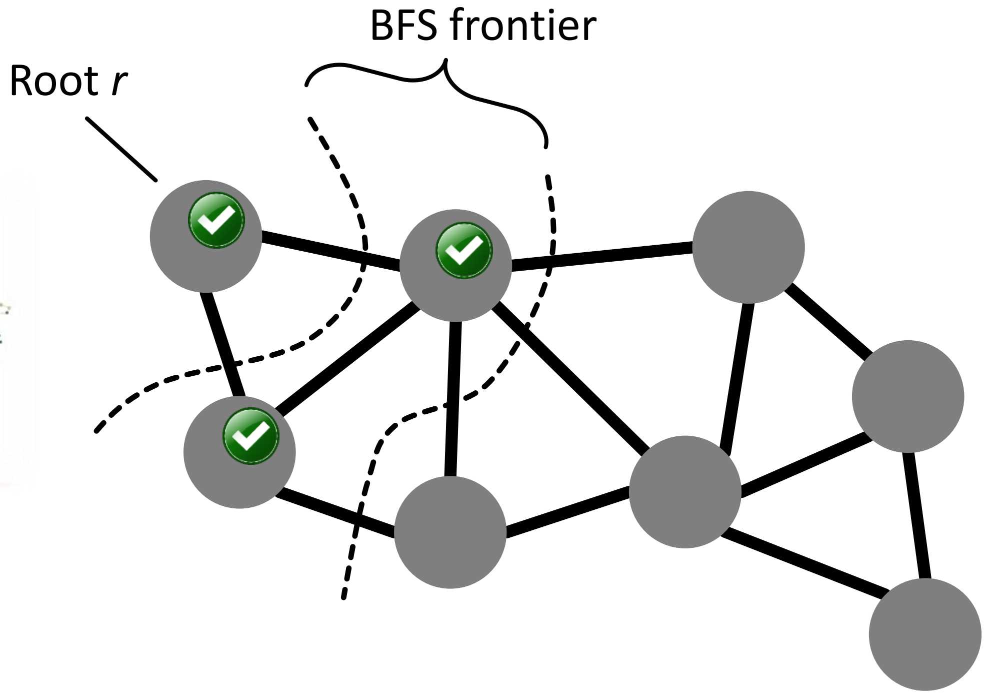
TOP-DOWN VS. BOTTOM-UP [1]

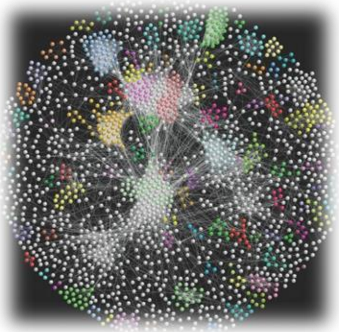




Pushing or pulling when expanding a frontier

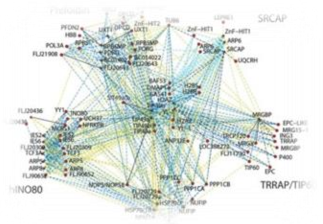


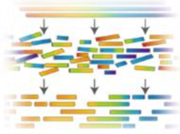




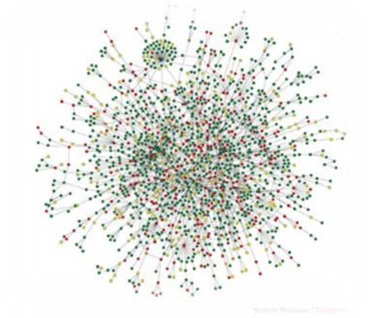
[1] S. Beamer, K. Asanović, and D. Patterson. Direction-optimizing breadth-first search. SC12.

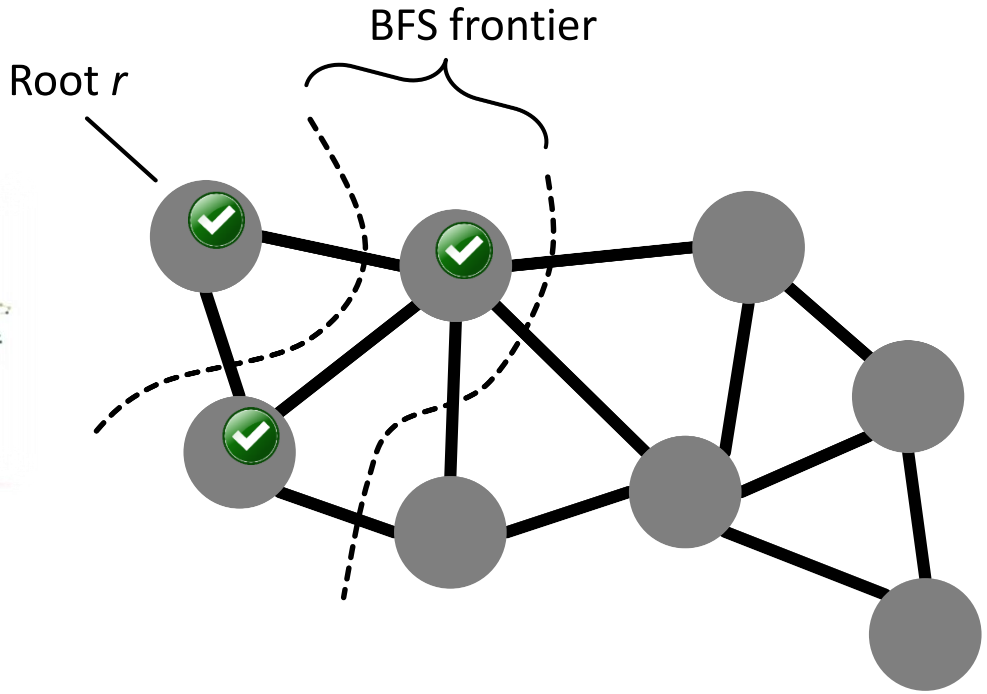
BFS
Top-Down vs. Bottom-Up [1]

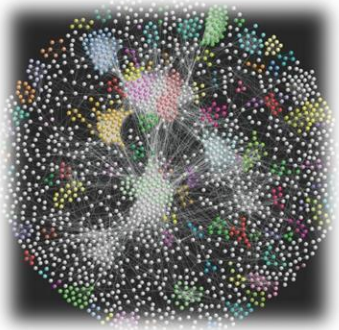




Pushing or pulling when expanding a frontier





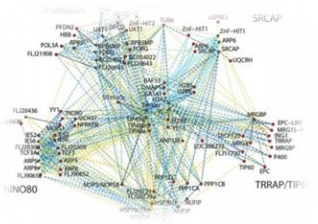


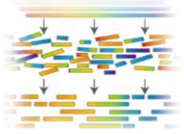
Pulling

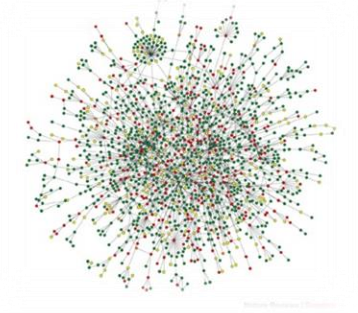
[1] S. Beamer, K. Asanović, and D. Patterson. Direction-optimizing breadth-first search. SC12.

BFS

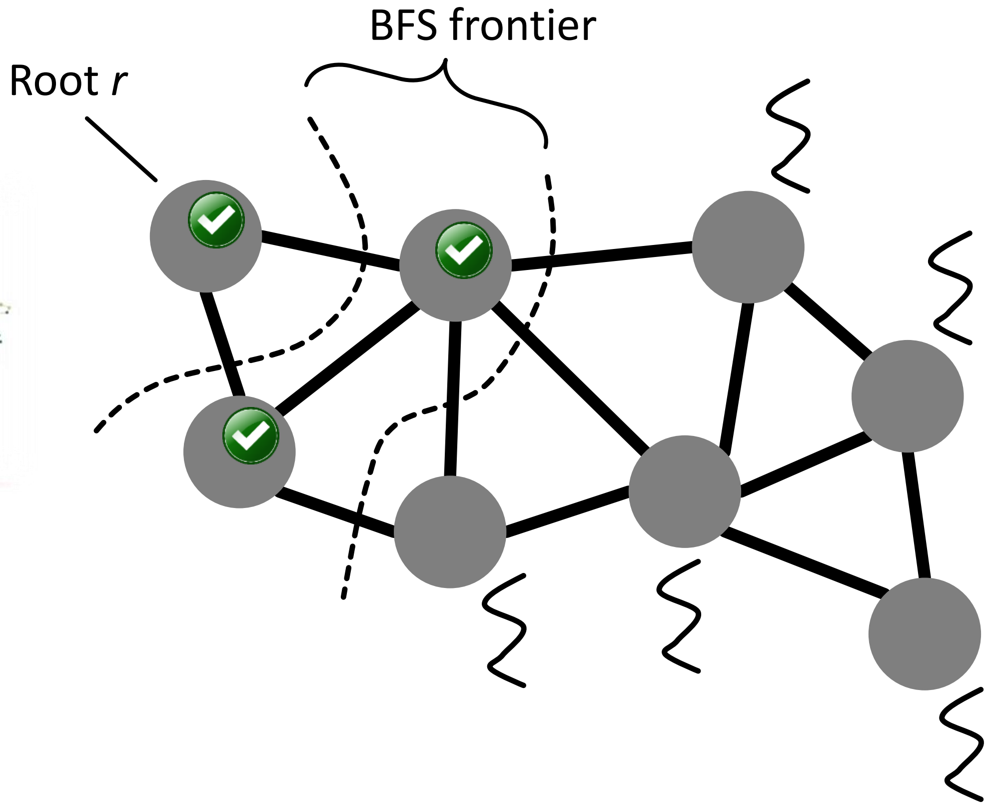
Top-Down vs. Bottom-Up [1]

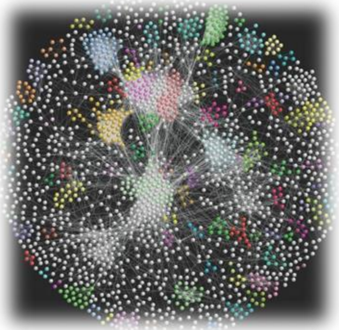






Pushing or pulling when expanding a frontier



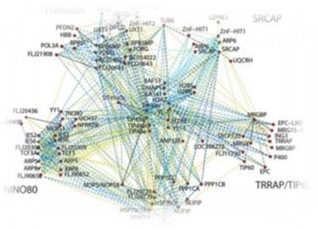


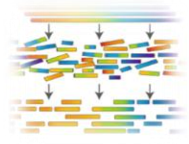
Pulling

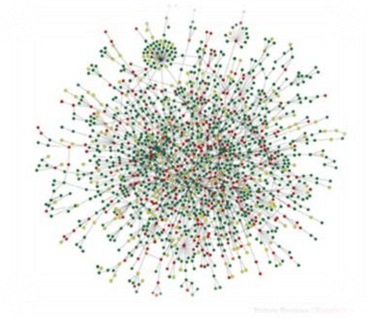
[1] S. Beamer, K. Asanović, and D. Patterson. Direction-optimizing breadth-first search. SC12.

BFS

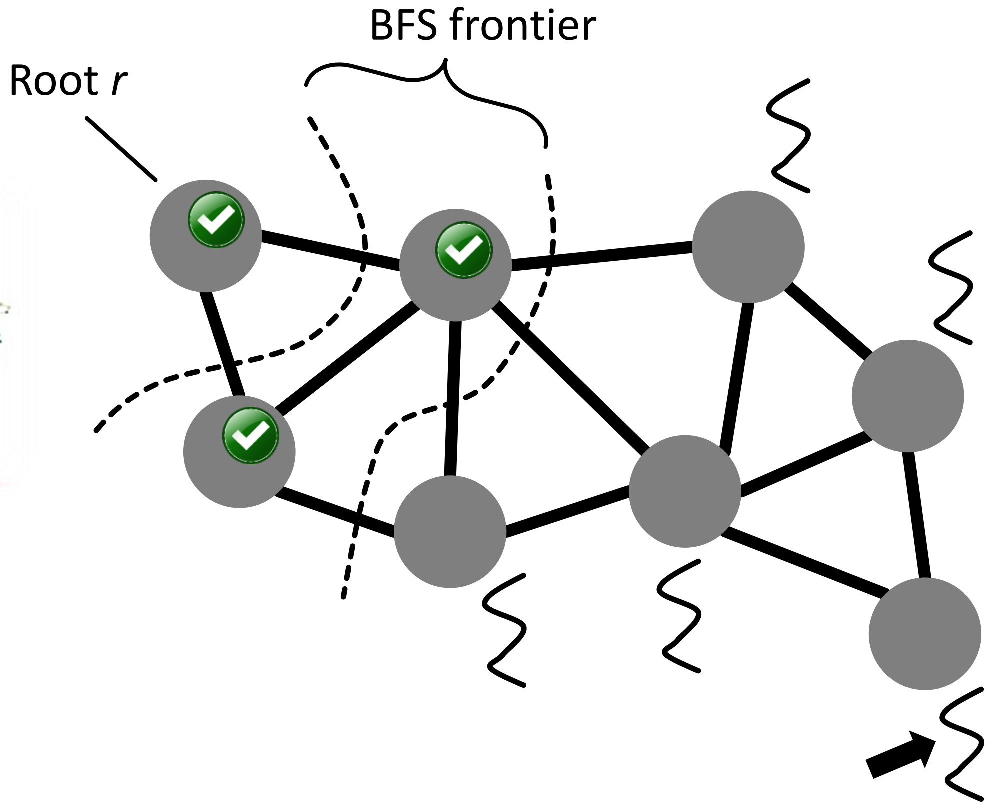
TOP-DOWN VS. BOTTOM-UP [1]

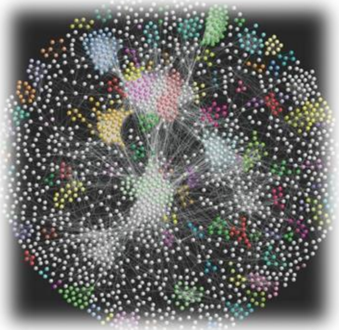






Pushing or pulling when expanding a frontier

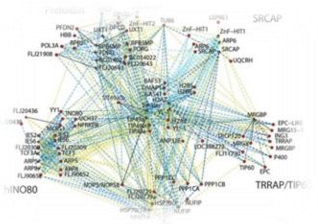


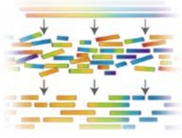


Pulling

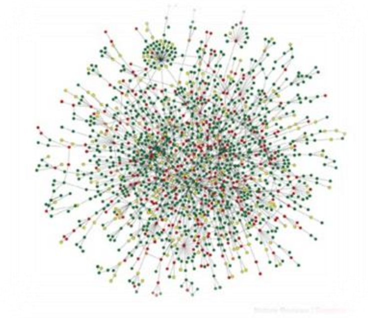
[1] S. Beamer, K. Asanović, and D. Patterson. Direction-optimizing breadth-first search. SC12.

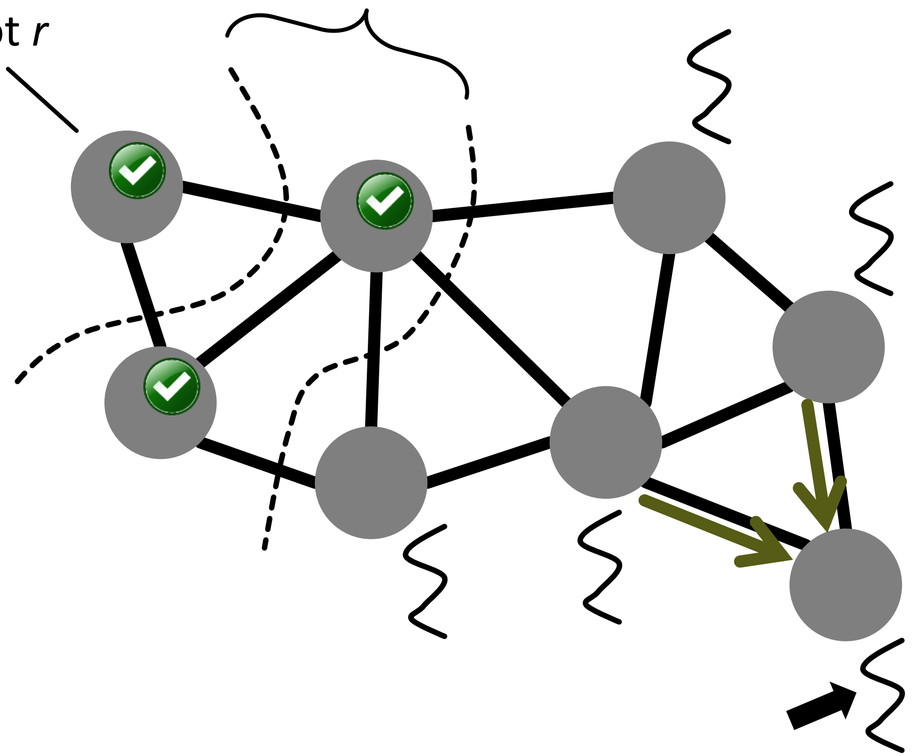
BFS
TOP-DOWN VS. BOTTOM-UP [1]

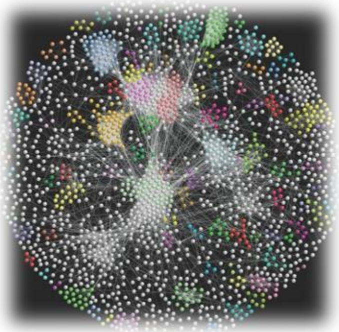




Pushing or pulling when expanding a frontier



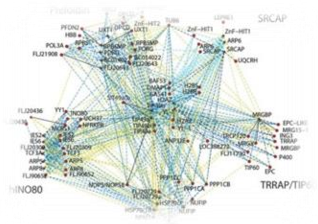


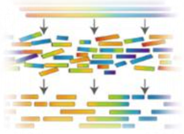


Pulling

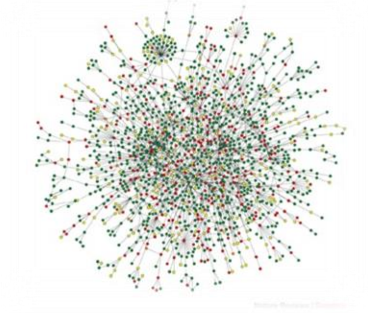
[1] S. Beamer, K. Asanović, and D. Patterson. Direction-optimizing breadth-first search. SC12.

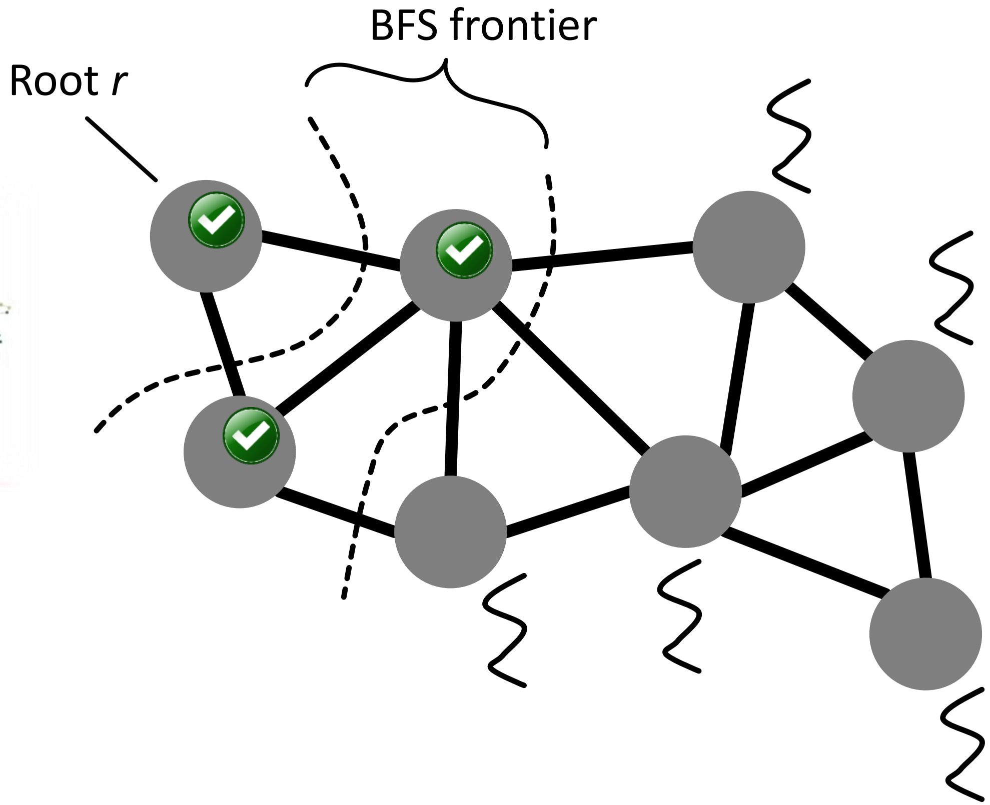
BFS
Top-Down vs. Bottom-Up [1]

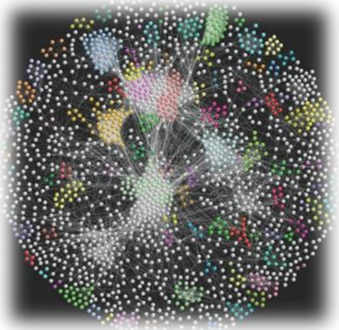




Pushing or pulling when expanding a frontier



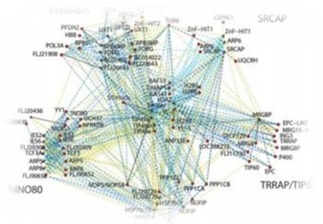


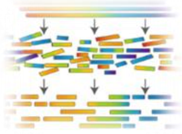


Pulling

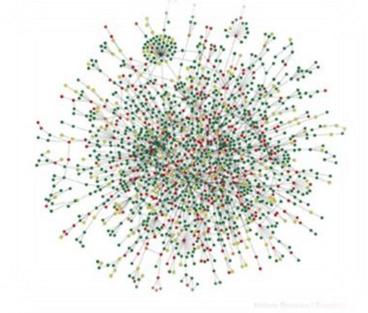
[1] S. Beamer, K. Asanović, and D. Patterson. Direction-optimizing breadth-first search. SC12.

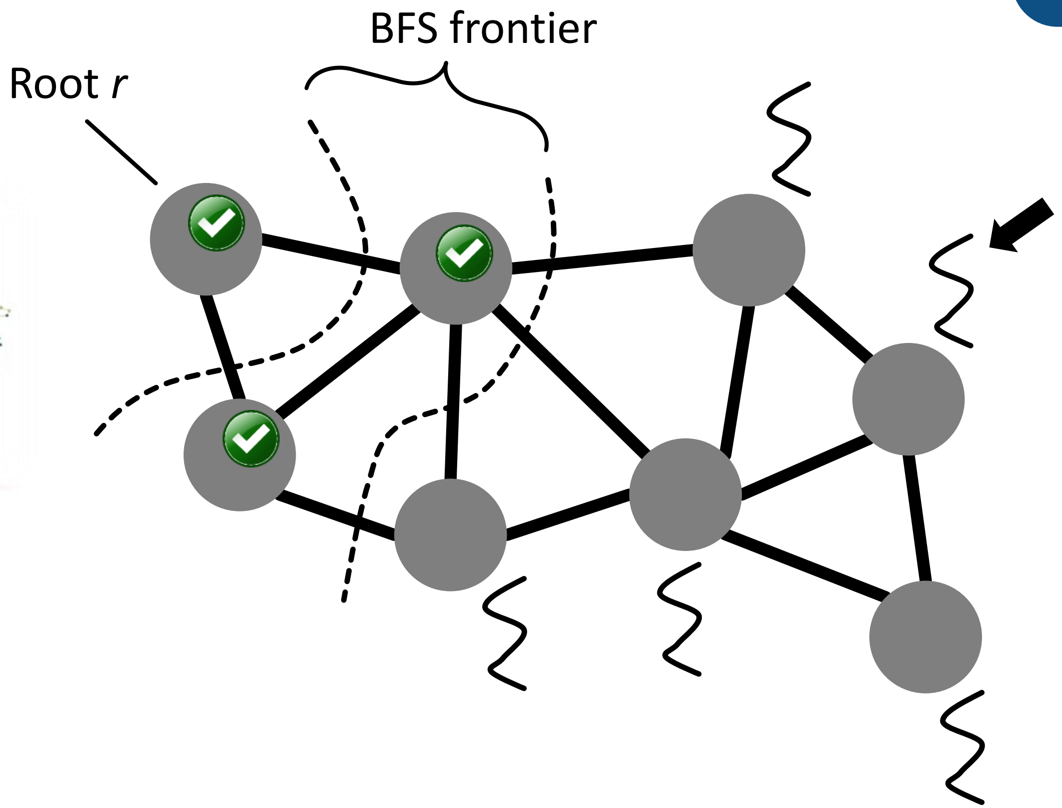
BFS
Top-Down vs. Bottom-Up [1]

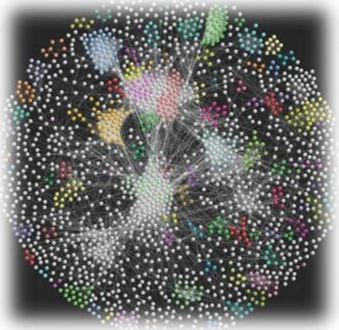




Pushing or pulling when expanding a frontier





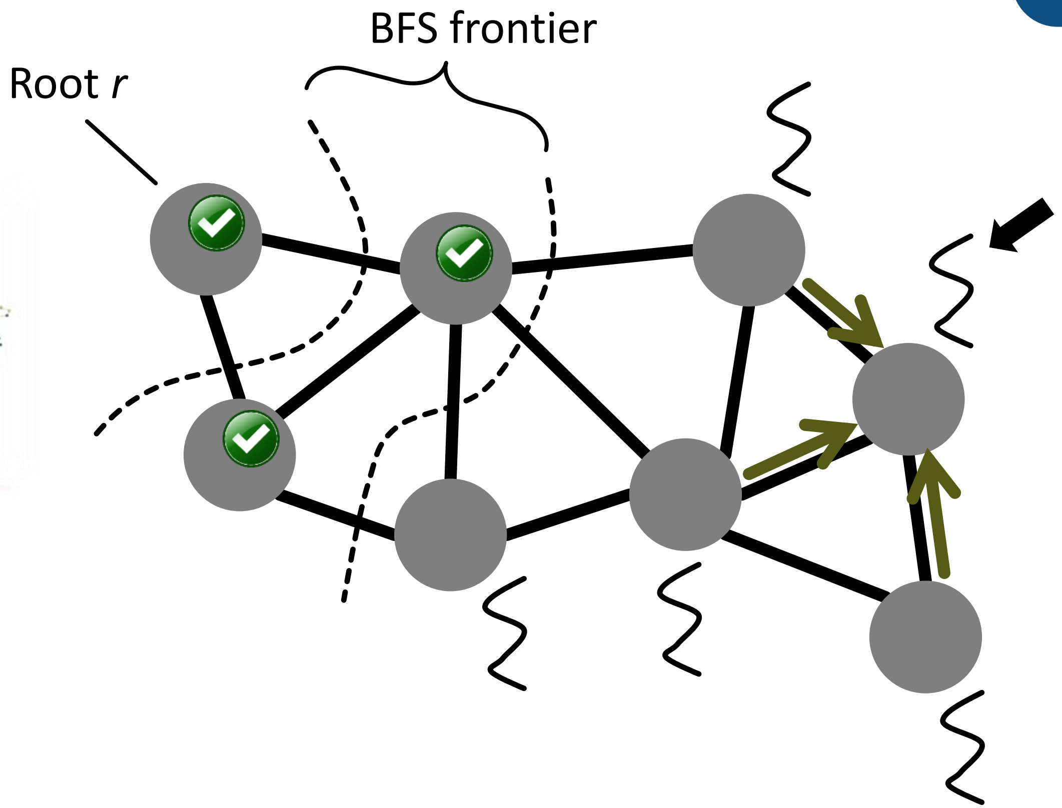


Pulling

[1] S. Beamer, K. Asanović, and D. Patterson. Direction-optimizing breadth-first search. SC12.

BFS
TOP-DOWN VS. BOTTOM-UP [1]

Pushing or pulling when expanding a frontier

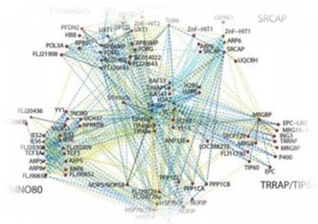


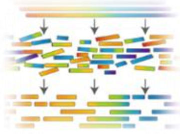
Pulling

[1] S. Beamer, K. Asanović, and D. Patterson. Direction-optimizing breadth-first search. SC12.

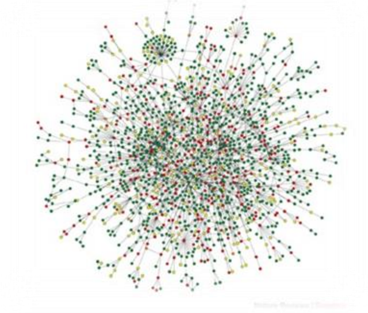
BFS

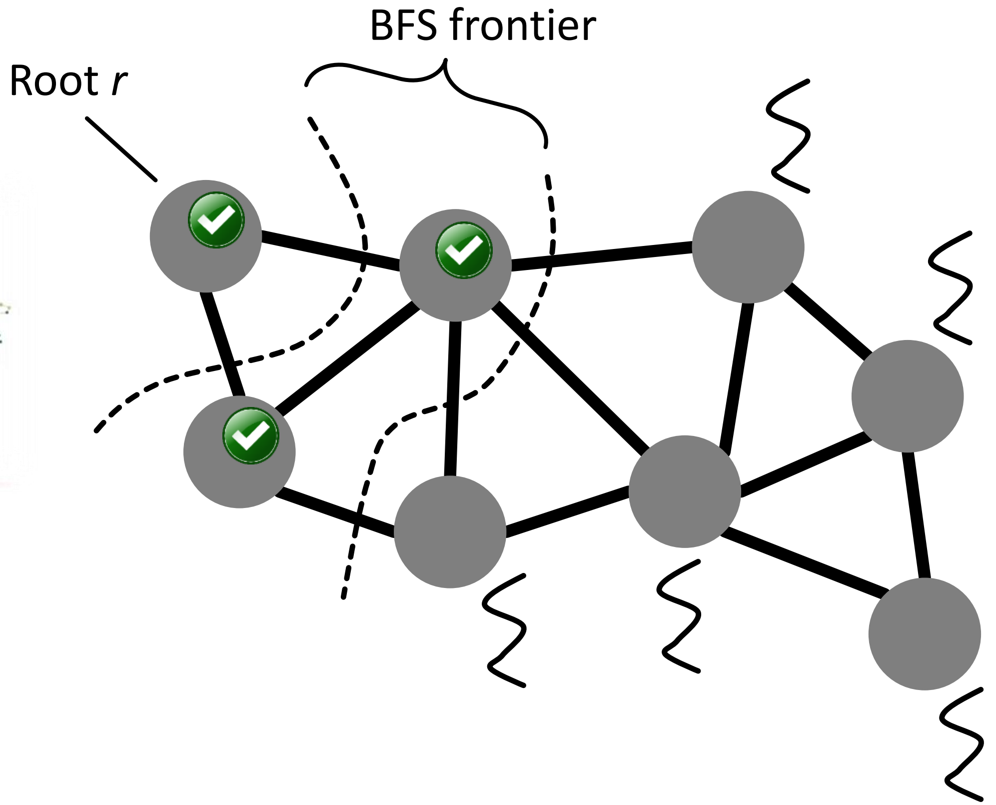
Top-Down vs. Bottom-Up [1]

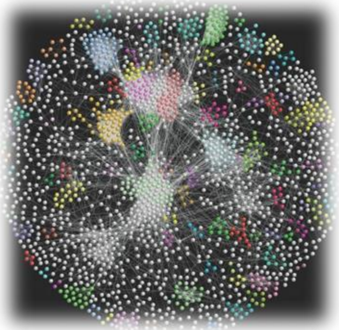




Pushing or pulling when expanding a frontier





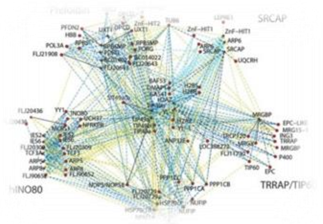


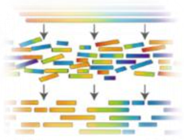
Pulling

[1] S. Beamer, K. Asanović, and D. Patterson. Direction-optimizing breadth-first search. SC12.

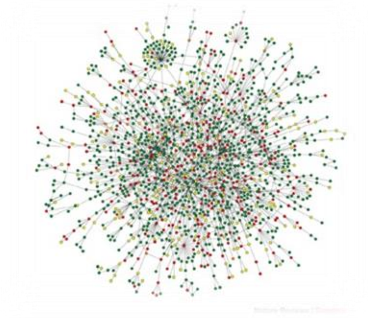
BFS

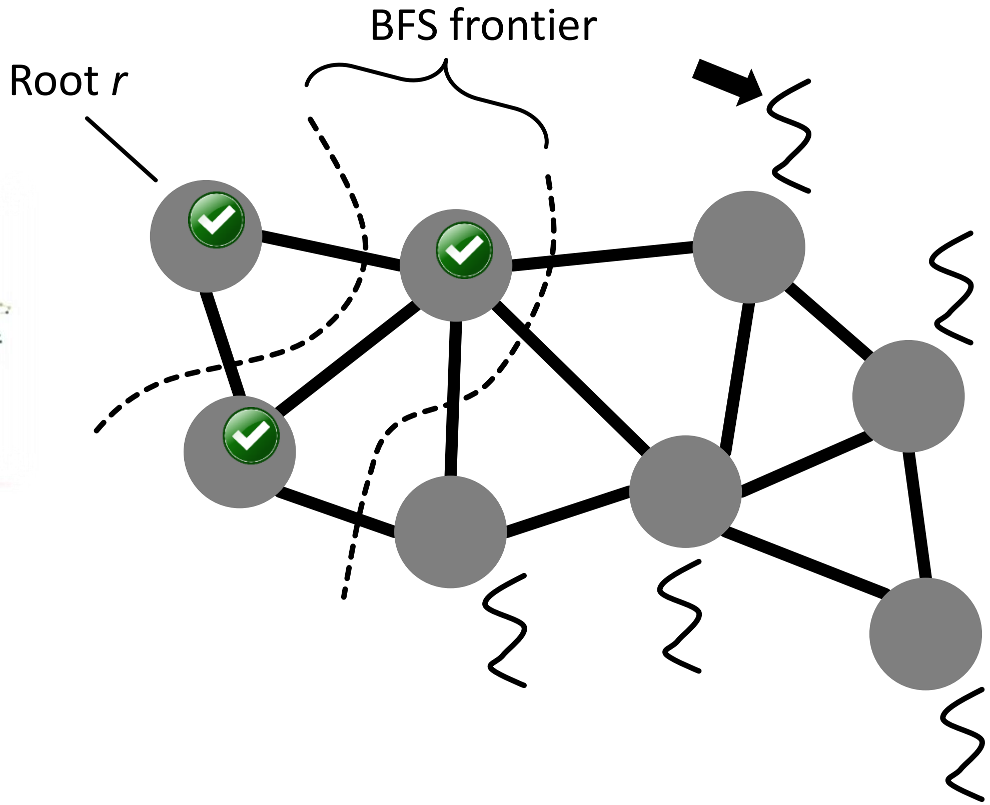
TOP-DOWN VS. BOTTOM-UP [1]

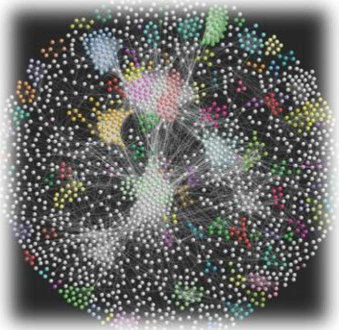




Pushing or pulling when expanding a frontier



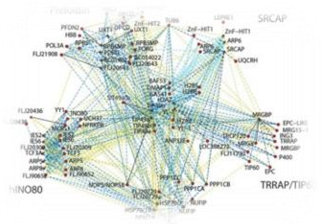


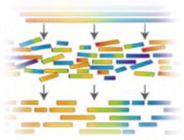


Pulling

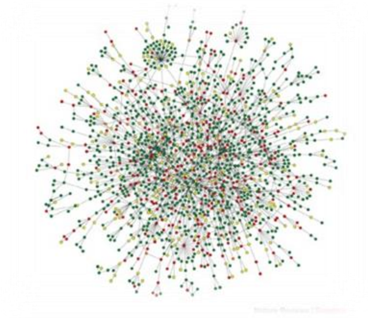
[1] S. Beamer, K. Asanović, and D. Patterson. Direction-optimizing breadth-first search. SC12.

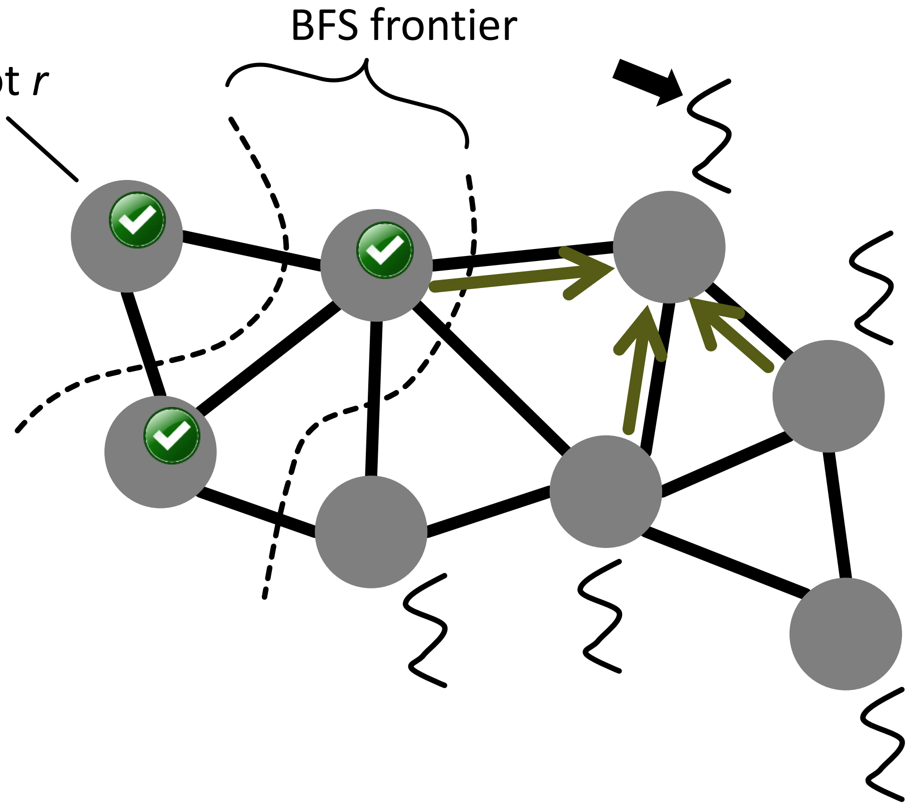
BFS
TOP-DOWN VS. BOTTOM-UP [1]

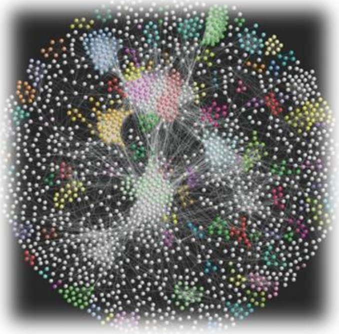




Pushing or pulling when expanding a frontier



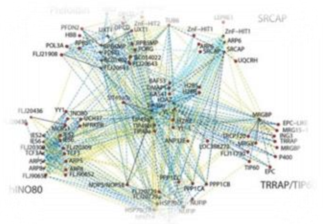


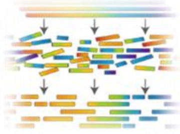


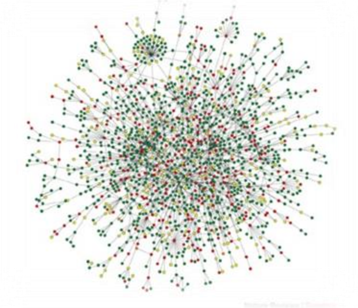
Pulling

[1] S. Beamer, K. Asanović, and D. Patterson. Direction-optimizing breadth-first search. SC12.

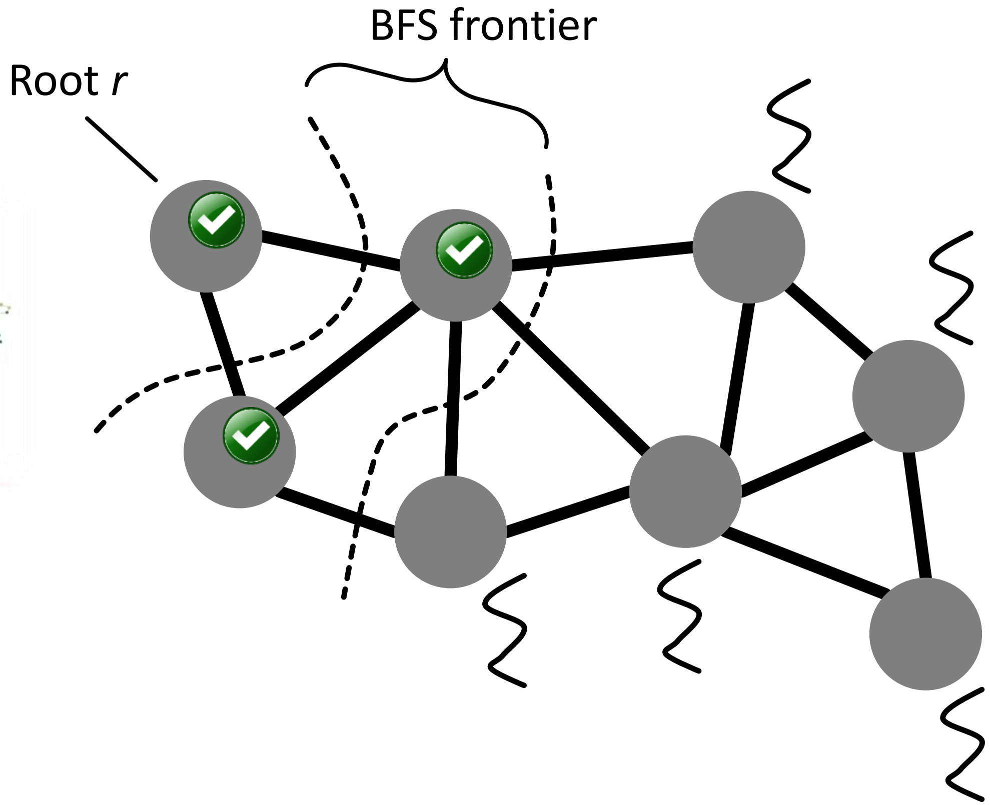
BFS
Top-Down vs. Bottom-Up [1]

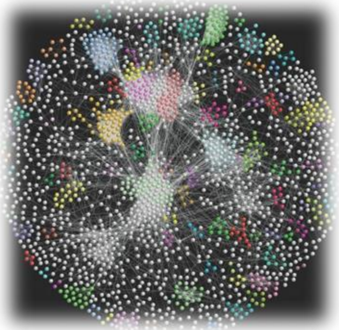






Pushing or pulling when expanding a frontier

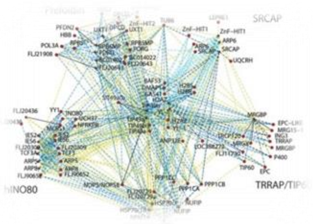


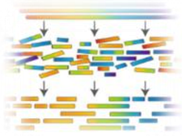


Pulling

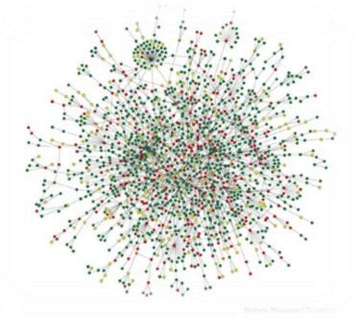
[1] S. Beamer, K. Asanović, and D. Patterson. Direction-optimizing breadth-first search. SC12.

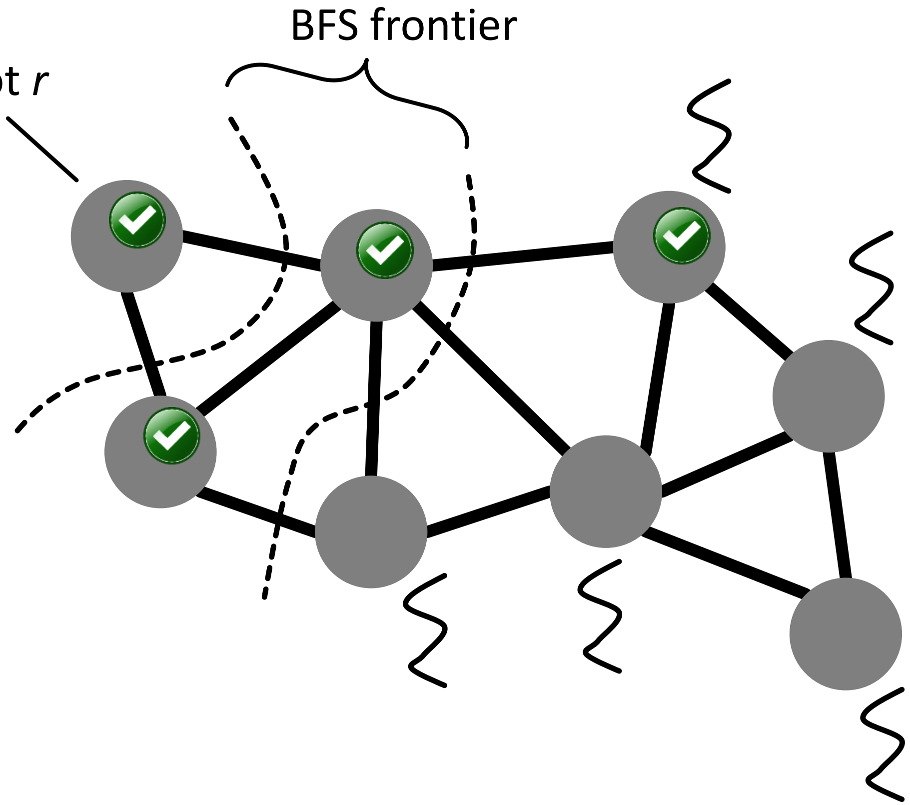
BFS
TOP-DOWN VS. BOTTOM-UP [1]

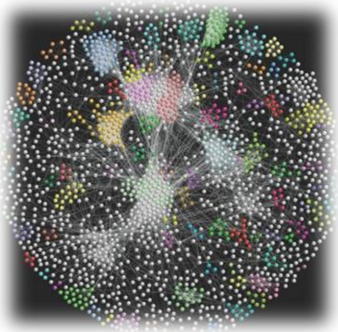




Pushing or pulling when expanding a frontier



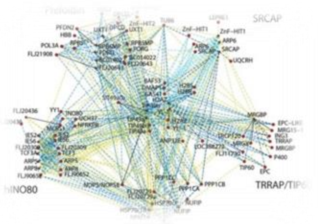


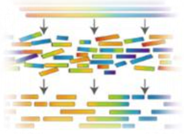


Pulling

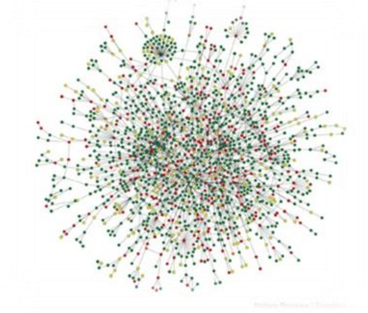
[1] S. Beamer, K. Asanović, and D. Patterson. Direction-optimizing breadth-first search. SC12.

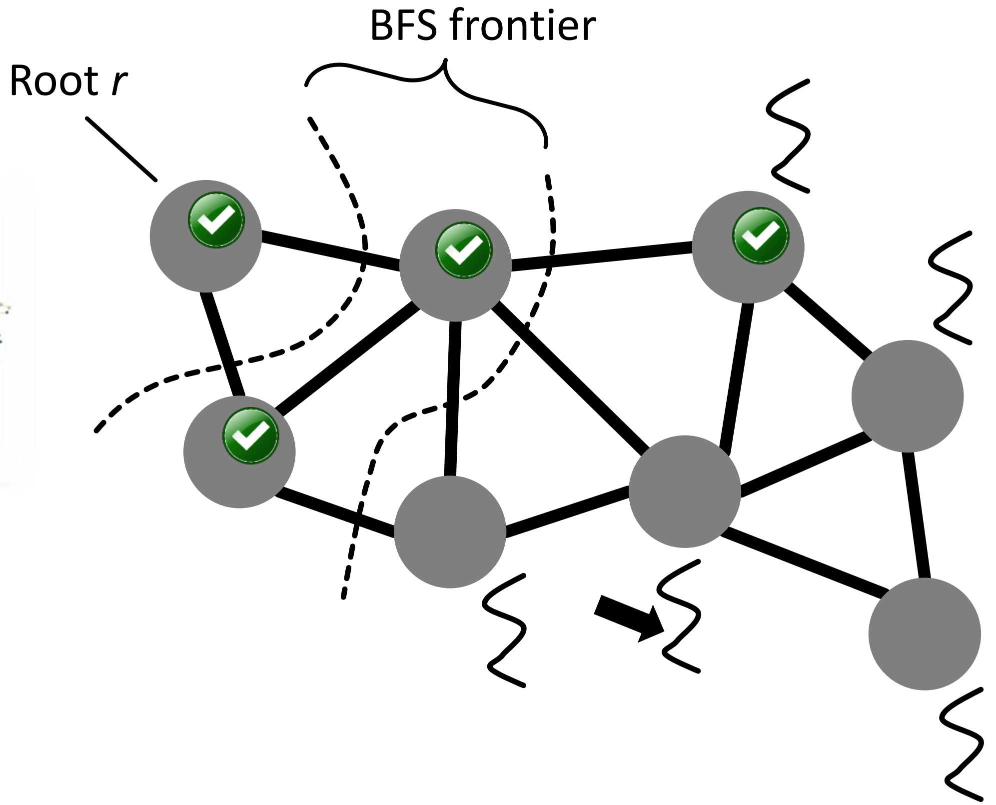
BFS
Top-Down vs. Bottom-Up [1]

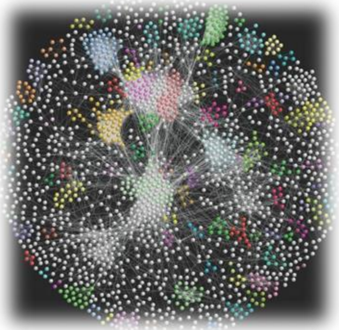




Pushing or pulling when expanding a frontier



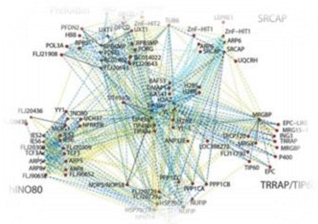


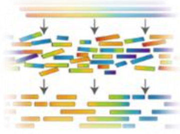


Pulling

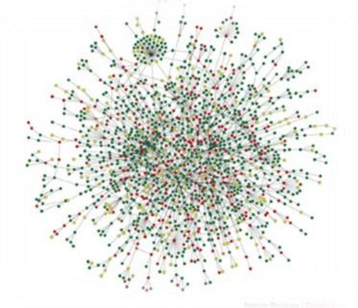
[1] S. Beamer, K. Asanović, and D. Patterson. Direction-optimizing breadth-first search. SC12.

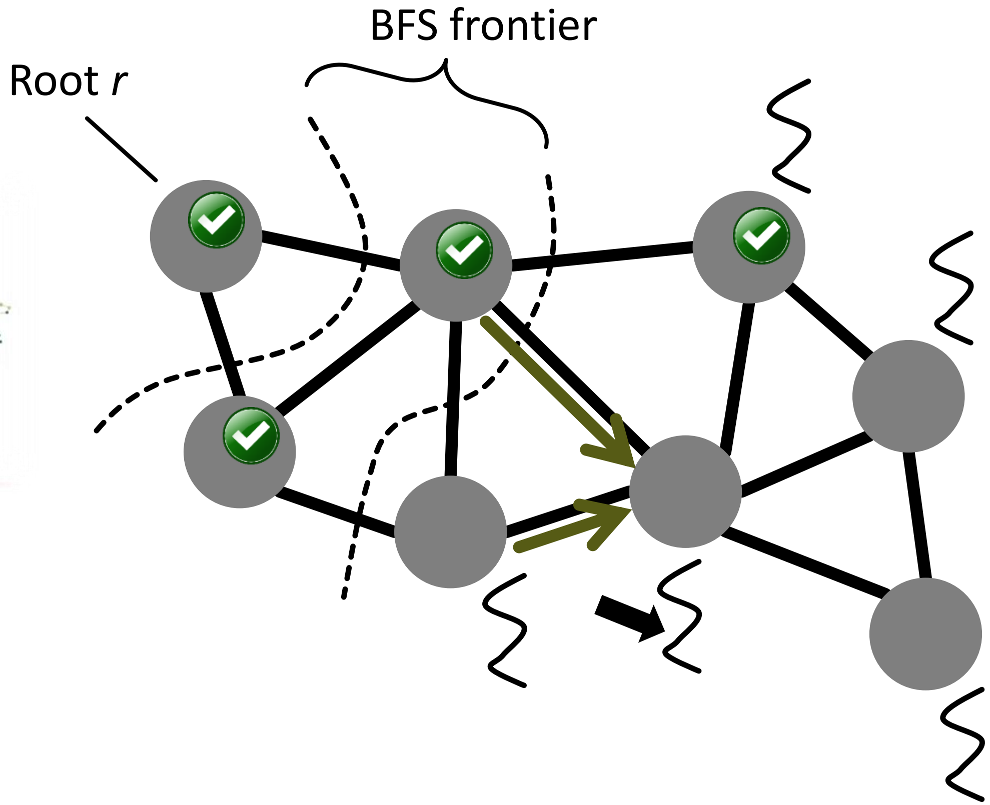
BFS
TOP-DOWN VS. BOTTOM-UP [1]

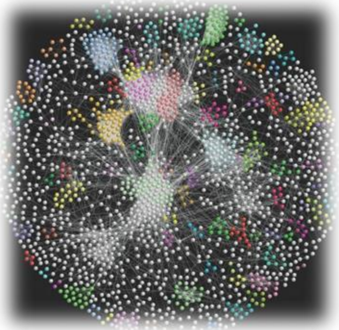




Pushing or pulling when expanding a frontier



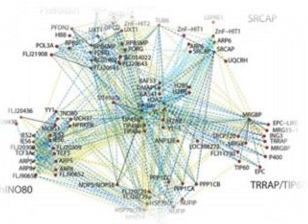


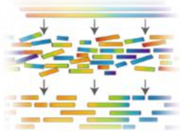


Pulling

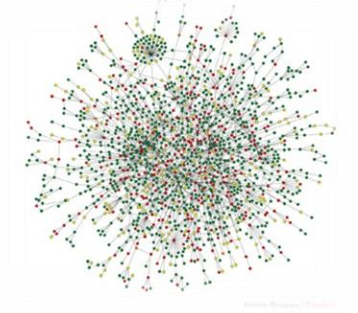
[1] S. Beamer, K. Asanović, and D. Patterson. Direction-optimizing breadth-first search. SC12.

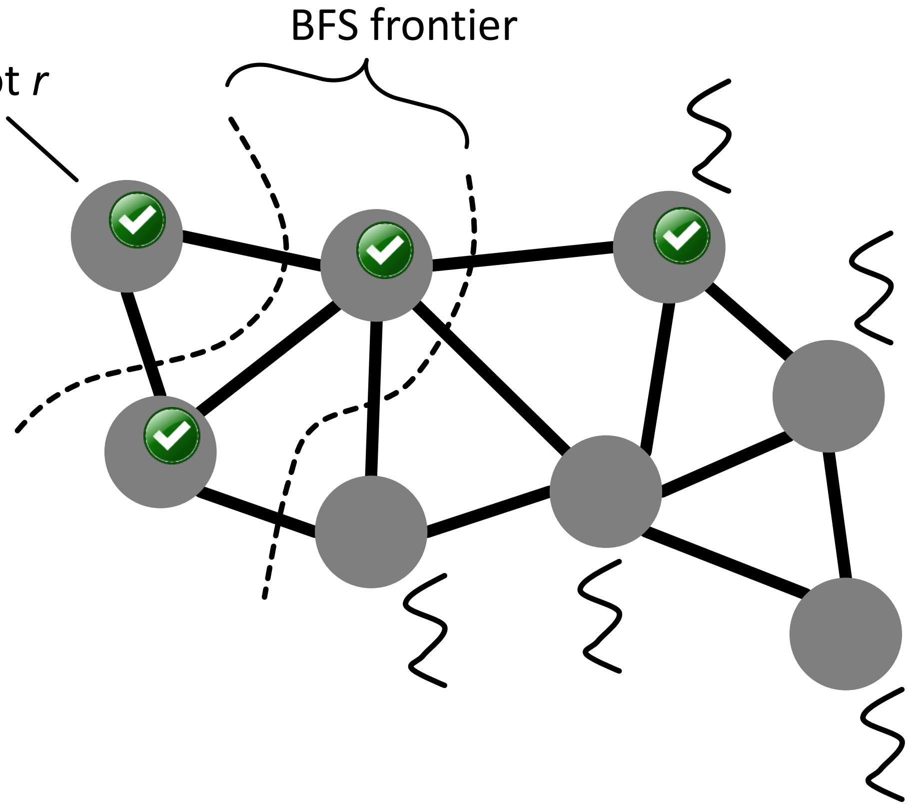
BFS
Top-Down vs. Bottom-Up [1]

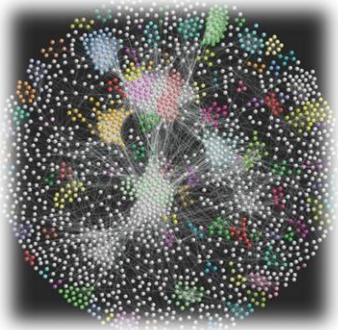




Pushing or pulling when expanding a frontier





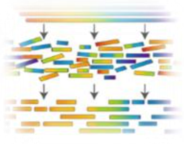


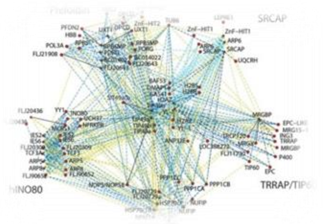
Pulling

[1] S. Beamer, K. Asanović, and D. Patterson. Direction-optimizing breadth-first search. SC12.

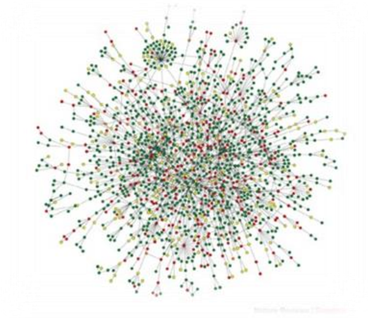
BFS

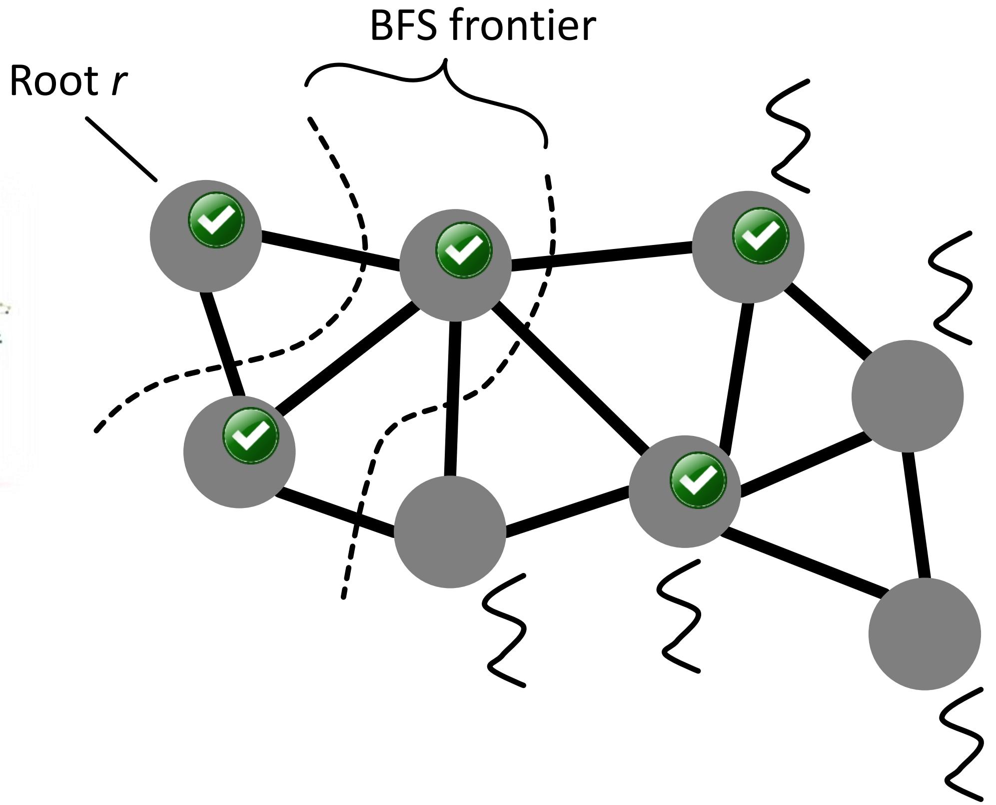
Top-Down vs. Bottom-Up [1]

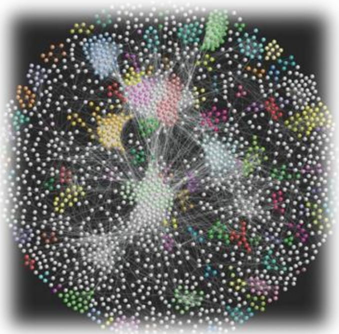




Pushing or pulling when expanding a frontier



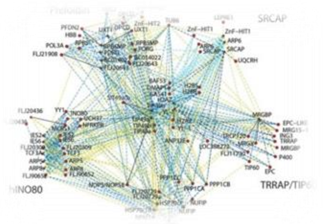


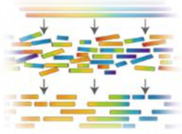


Pulling

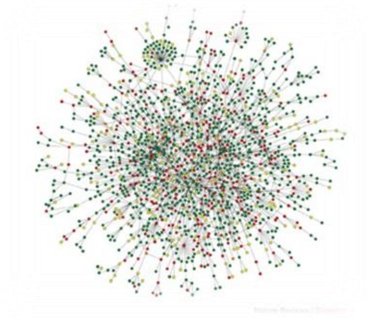
[1] S. Beamer, K. Asanović, and D. Patterson. Direction-optimizing breadth-first search. SC12.

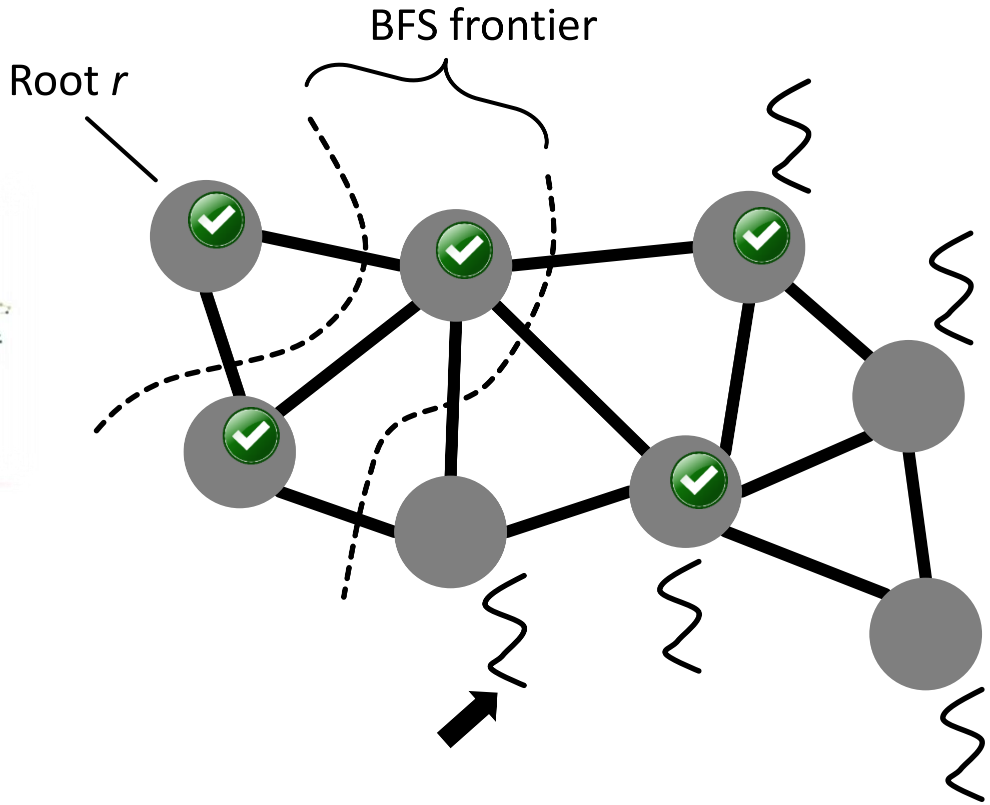
BFS
Top-Down vs. Bottom-Up [1]

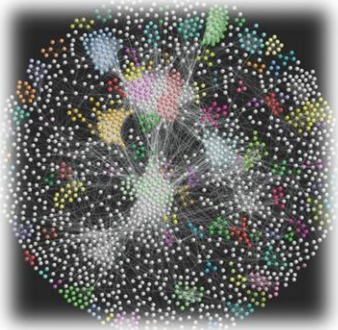




Pushing or pulling when expanding a frontier





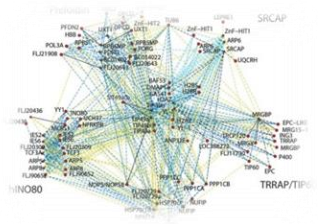


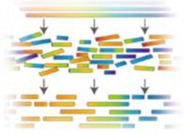
Pulling

[1] S. Beamer, K. Asanović, and D. Patterson. Direction-optimizing breadth-first search. SC12.

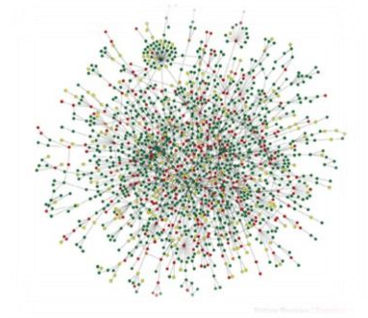
BFS

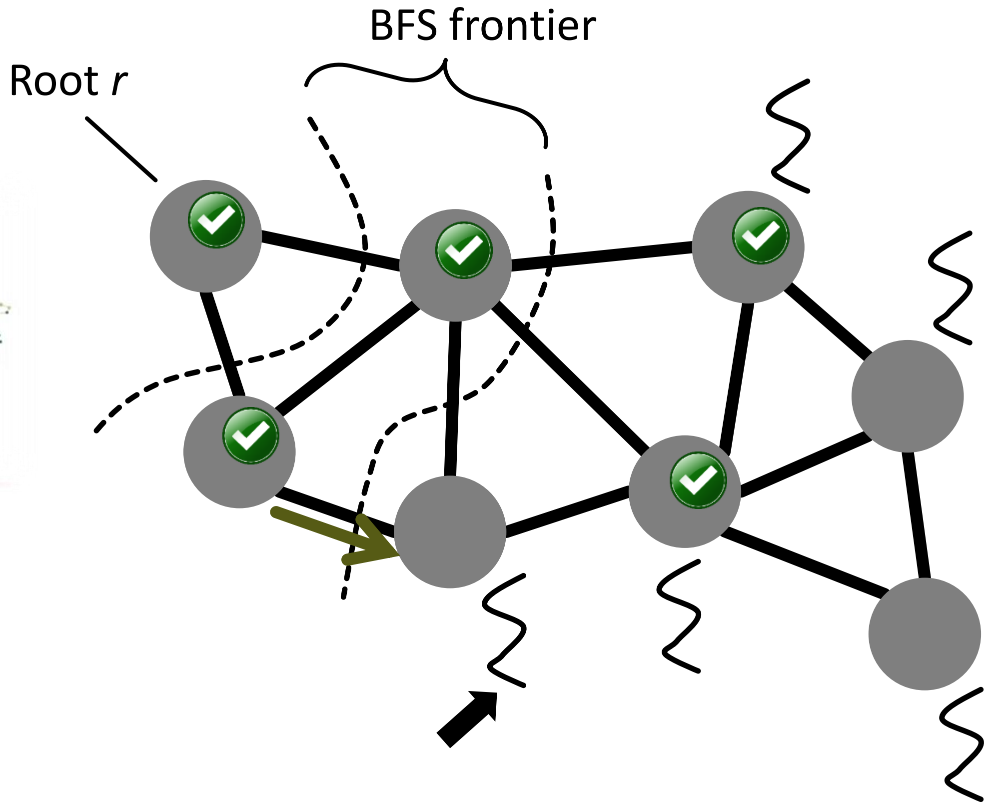
Top-Down vs. Bottom-Up [1]

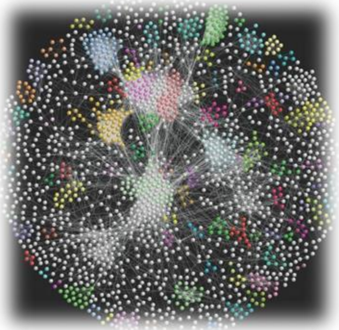




Pushing or pulling when expanding a frontier







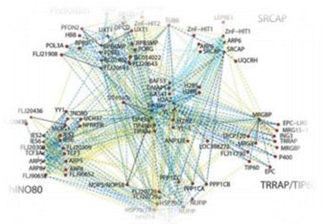
Pulling

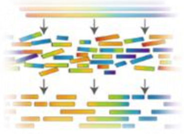
GRAPH 500

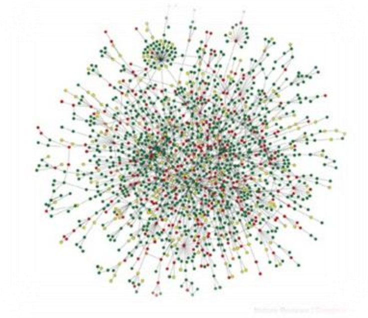
[1] S. Beamer, K. Asanović, and D. Patterson. Direction-optimizing breadth-first search. SC12.

BFS

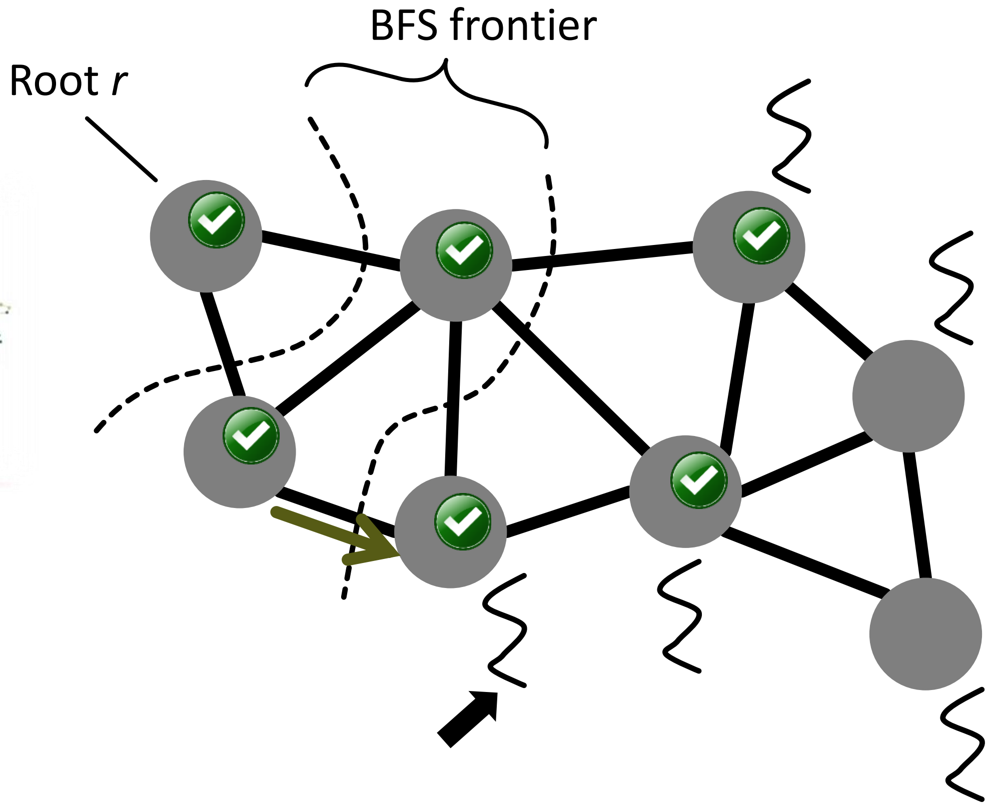
Top-Down vs. Bottom-Up [1]

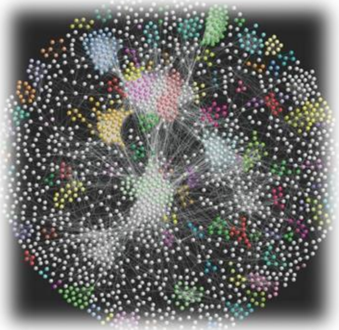






Pushing or pulling when expanding a frontier

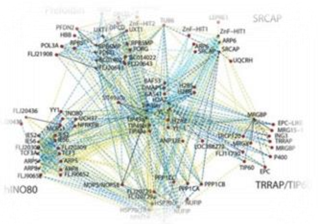


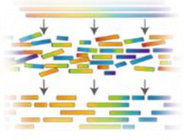


Pulling

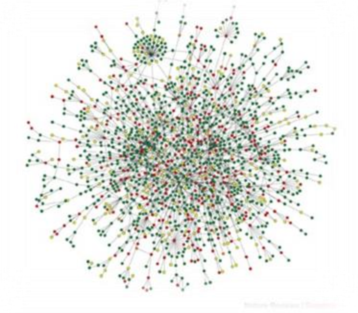
[1] S. Beamer, K. Asanović, and D. Patterson. Direction-optimizing breadth-first search. SC12.

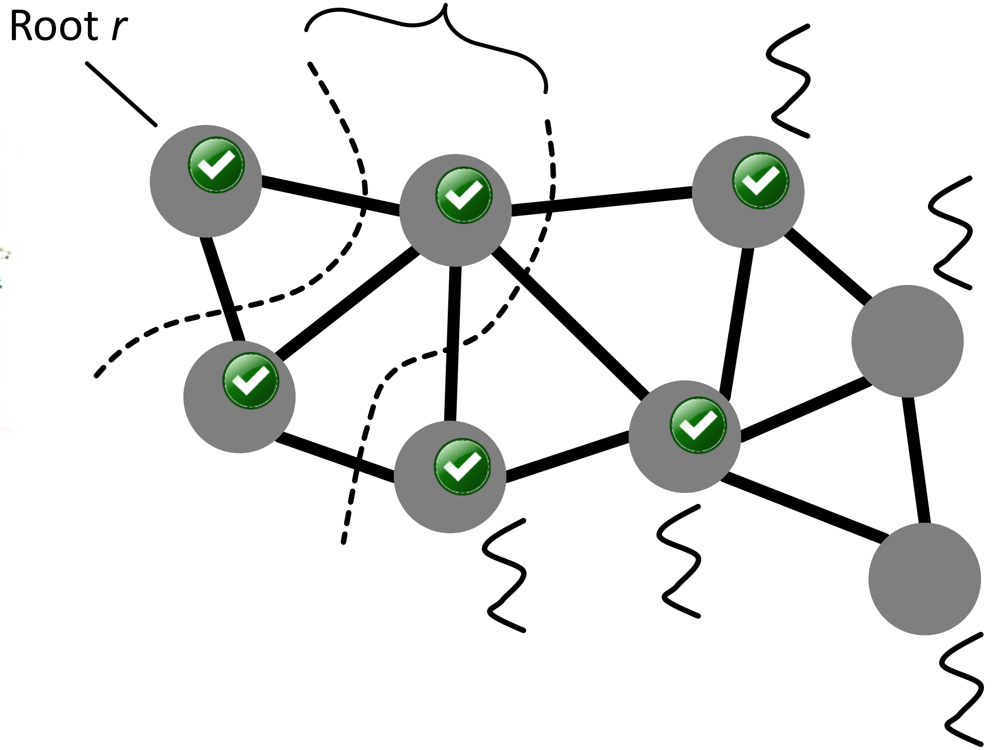
BFS
Top-Down vs. Bottom-Up [1]

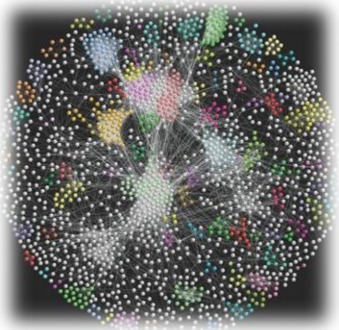




Pushing or pulling when expanding a frontier







Pulling

[1] S. Beamer, K. Asanović, and D. Patterson. Direction-optimizing breadth-first search. SC12.

OTHER ALGORITHMS & FORMULATIONS

M. Besta et al.: “To Push or To Pull: On Reducing Communication and Synchronization in Graph Computations”, HPDC’17

Other Algorithms & Formulations

Triangle Counting

```
/* Input: a graph G. Output: An array
 * tc[1..n] that each vertex belongs to

function TC(G) {tc[1..n] = [0..0]
  for v ∈ V do in par
    for w1 ∈ N(v) do [in par]
      for w2 ∈ N(v) do [in par]
        if adj(w1,w2) R update_tc();
  tc[1..n] = [tc[1]/2 .. tc[n]/2]; }
function update_tc() {
  {++tc[w1]; /* or ++tc[w2]. */} W i

  {++tc[v];}
}
```

Δ-Stepping

```
/* Input: a graph G, a vertex r, the Δ parameter.
   Output: An array of distances d */

function Δ-Stepping(G, r, Δ){
 bckt=[∞..∞]; d=[∞..∞]; active=[false..false];
 bckt_set={0}; bckt[r]=0; d[r]=0; active[r]=true; itr=0;

 for b ∈ bckt_set do { //For every bucket do...
  do {bckt_empty = false; //Process b until it is empty.
  process_buckets();} while(!bckt_empty); } }

function process_buckets() {
 for v ∈ bckt_set[b] do in par                         PUSHING
  if(bckt[v]==b && (itr == 0 or active[v])) {
   active[v] = false; //Now, expand v's neighbors.
   for w ∈ N(v) {weight = d[v] + W(v,w);
    if(weight < d[w]) { R //Proceed to relax w.
     new_b = weight/Δ; bckt[v] = new_b;
     bckt_set[new_b] = bckt_set[new_b] ∪ {w};}
    d[w] = weight; W i;
    if(bckt[w]==b)R {active[w]=true; bckt_empty=true;}}} R
 for v ∈ V do in par
  if(d[v] > b) {for w ∈ N(v) do {
   if(bckt[w] == b && (active[w] o
    weight = d[w] + W(w,v) R;
    if(weight < d[v]) {d[v]=weight
     if(bckt[v] > new_b) {
```

BFS

```
/* Input: a graph
   R0 for each
```

BC (algebraic notation)

```
/* Input: a graph G. Output: centrality scores bc[1..n].

function BC(G) { bc[1..n] = [0..0]
 Define Π so that any Π ∋ u = (index_u, pred_u, mult_u, part_u);
 Define u ⇐pred v with u, v ∈ Π so that u becomes
     u = (index_u, pred_u ∪ index_v, mult_u + mult_v, part_u);
 Define u ⇐part v with u, v ∈ Π so that u becomes
     u = (index_u, pred_u, mult_u, part_u + (mult_u/mult_v)(1 + part_v));

 for s ∈ V do [in par] {
  ready = [1,...,1]; ready[s] = 0;
  R = BFS(G, ready, [(1,∅,0,0)..(s,∅,1,0)..(n,∅,0,0)], ⇐pred);
  Define graph G' = (V, E') where (u,v) ∈ E'
  Let ready[u] be the in-degree of u ∈ V
  R = BFS(G', ready, R, ⇐part);
  for (index_u, pred_u, mult_u, part_u) ∈ R do [in
   bc[u] += part_u; }
  par] {

  my_F[p_ID] = my_F[p_ID] ∪ {v}; }
```

Betweenness Centrality (BC)

```
/* Input: a graph G. Output: centrality scores bc[1..n]. */

function BC(G) { bc[1..n] = [0..0]
 for s ∈ V do [in par] {
  for t ∈ V do in par {                             PART 1: INITIALIZATION
   pred[t]=succ[t]=∅; σ[t]=0; dist[t]=∞;}
  σ[s]=enqueued=1; dist[s]=itr=0; δ[1..n]=[0..0]
  Q[0]={s}; Q_1[1..p]=pred_1[1..p]=succ_1[1..p]=[∅..∅];
  while enqueued > 0 do                             PART 2: COUNTING SHORTEST PATHS
   count_shortest_paths();
  --itr
  while itr > 0 do
   accumulate_dependencies();                       PART 3: DEPENDENCY ACCUMULATION
} }

function count_shortest_paths() { enqueued = 0;
#if defined PUSHING_IN_PART_2                        PUSHING (IN PART 2)
 for v ∈ Q[itr] do in par {
  for w ∈ N(v) do [in par] {
   if dist[w] == ∞ R {
    Q_1[itr + 1] = Q_1[itr + 1] ∪ {w} i
    dist[w] = dist[v] + 1 W i; ++enqueued;}
   if dist[w] == dist[v] + 1 R {
    σ[w] += σ[v] W i; pred_1[w] = pred_1[w] ∪ {v};
 IN_PART_2                                           PULLING (IN PART 2)
 r {
 [in par] {
```

Graph Coloring

```
// Input: a graph G. Output: An array of vertex colors c[1..n].
// In the code, the details of functions seq_color_partition and
// init are omitted due to space constrains.

function Boman-GC(G) {
  done = false; c[1..n] = [∅..∅]; //No vertex is colored yet
  //avail[i][j]=1 means that color j can be used for vertex i.
  avail[1..n][1..C] = [1..1][1..1]; init(𝓑, 𝒫);
  while (!done) {
    for 𝒫 ∈ 𝒫 do in par {seq_color_partition(𝒫);}
    fix_conflicts(); } }

function fix_conflicts() {
  for v ∈ 𝓑 in par do {for u ∈ N(v) do
    if (c[u] == c[v]) {
      {avail[u][c[v]] = ∅ W i;}                          PUSHING

      {avail[v][c[v]] = ∅ R i;}                          PULLING
  }}
```

PageRank

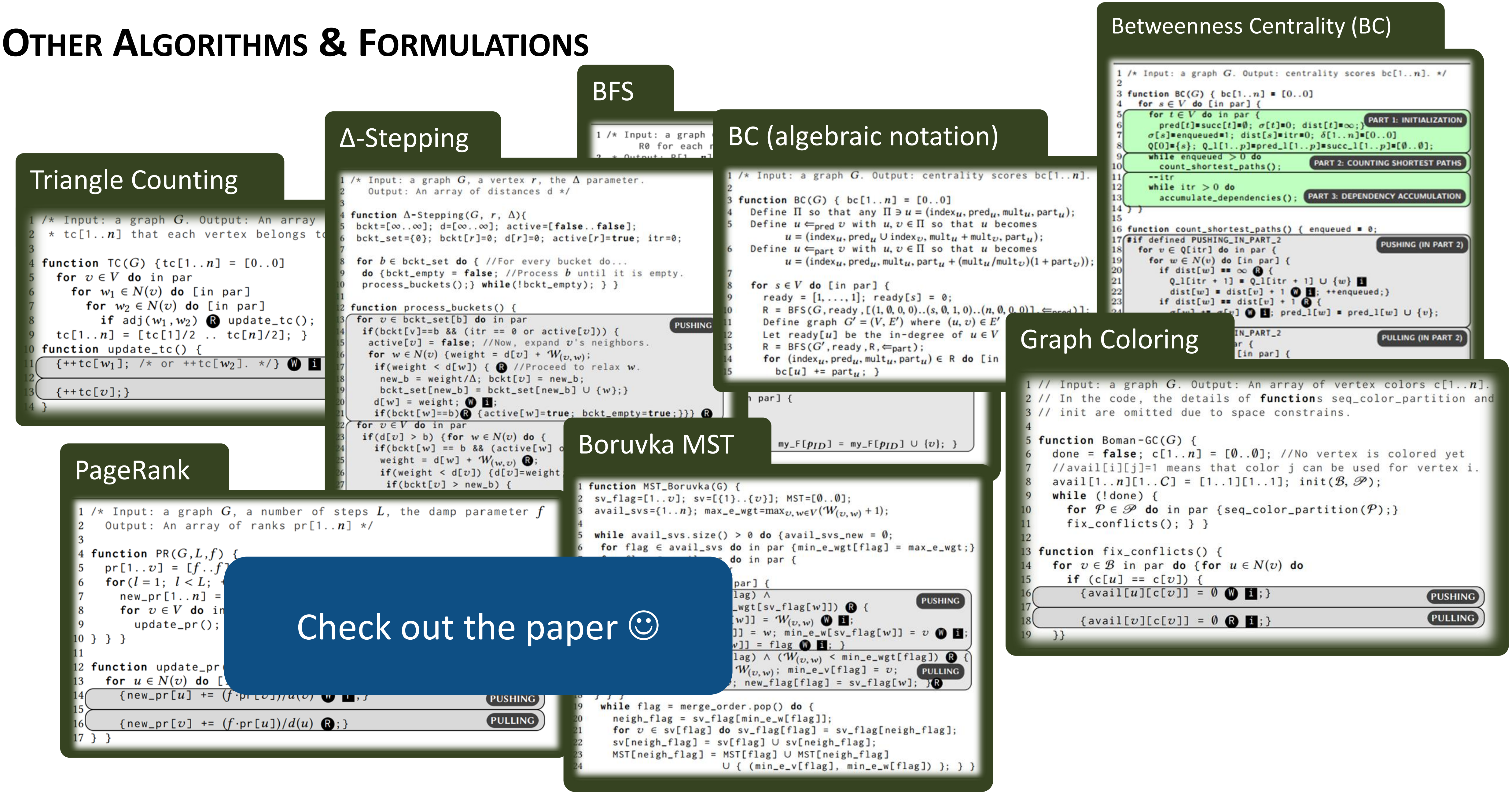
```
/* Input: a graph G, a number of steps L, the damp parameter f
   Output: An array of ranks pr[1..n] */

function PR(G,L,f) {
  pr[1..v] = [f..f]; //Initialize PR values.
  for(l = 1; l < L; ++l) {
    new_pr[1..n] = [0..0];
    for v ∈ V do in par {
      update_pr(); new_pr[v] += (1 − f)/n; pr[v] = new_pr[v];
} } }

function update_pr() {
  for u ∈ N(v) do [in par] {
    {new_pr[u] += (f·pr[v])/d(v) W f;}                    PUSHING

    {new_pr[v] += (f·pr[u])/d(u) R;}                      PULLING
} }
```

Boruvka MST

```
function MST_Boruvka(G) {
 sv_flag=[1..v]; sv=[{1}..{v}]; MST=[∅..∅];
 avail_svs={1..n}; max_e_wgt=max_{v,w∈V}(W(v,w) + 1);

 while avail_svs.size() > 0 do {avail_svs_new = ∅;
  for flag ∈ avail_svs do in par {min_e_wgt[flag] = max_e_wgt;}
  for flag ∈ avail_svs do in par {
   for v ∈ sv[flag] do {
    for w ∈ N(v) do [in par] {
     if (sv_flag[w] ≠ flag) ∧                                PUSHING
        (W(v,w) < min_e_wgt[sv_flag[w]]) R {
      min_e_wgt[sv_flag[w]] = W(v,w) W i;
      min_e_v[sv_flag[w]] = w; min_e_w[sv_flag[w]] = v W i;
      new_flag[sv_flag[w]] = flag W i; }
     if (sv_flag[w] ≠ flag) ∧ (W(v,w) < min_e_wgt[flag]) R {
      min_e_wgt[flag] = W(v,w); min_e_v[flag] = v;          PULLING
      min_e_w[flag] = w; new_flag[flag] = sv_flag[w]; }R
 } } }
 while flag = merge_order.pop() do {
  neigh_flag = sv_flag[min_e_w[flag]];
  for v ∈ sv[flag] do sv_flag[flag] = sv_flag[neigh_flag];
  sv[neigh_flag] = sv[flag] ∪ sv[neigh_flag];
  MST[neigh_flag] = MST[flag] ∪ MST[neigh_flag]
               ∪ { (min_e_v[flag], min_e_w[flag]) }; } }
```

M. Besta et al.: "To Push or To Pull: On Reducing Communication and Synchronization in Graph Computations", HPDC'17

Other Algorithms & Formulations

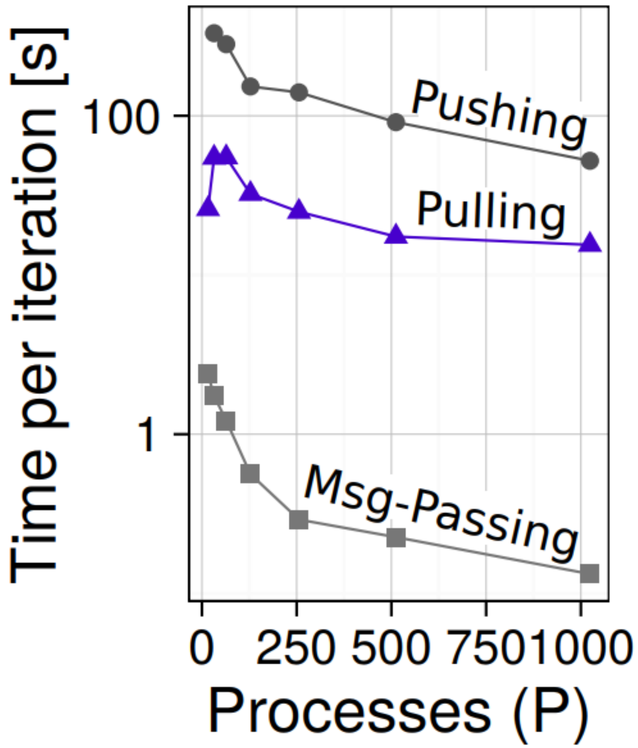


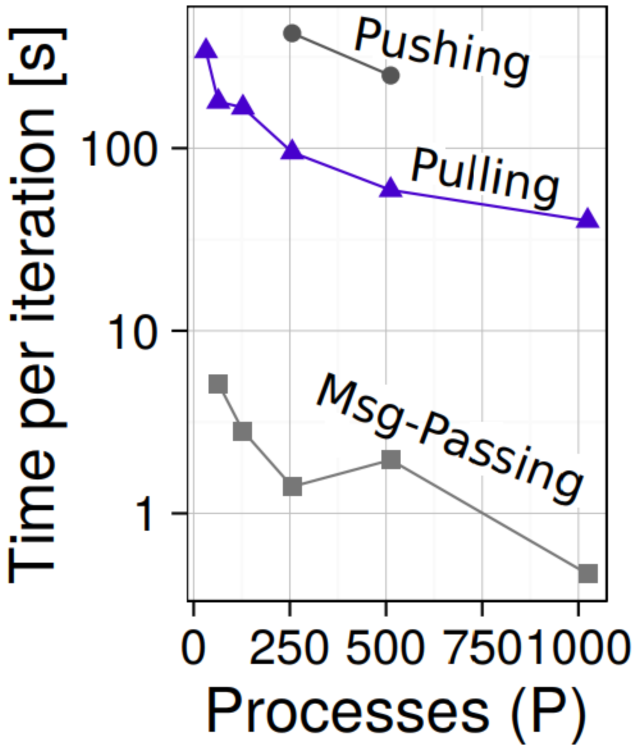
M. Besta et al.: “To Push or To Pull: On Reducing Communication and Synchronization in Graph Computations”, HPDC’17

Performance Analysis
PageRank

Kronecker graphs

Distributed-Memory





M. Besta et al.: "To Push or To Pull: On Reducing Communication and Synchronization in Graph Computations", HPDC'17

PERFORMANCE ANALYSIS

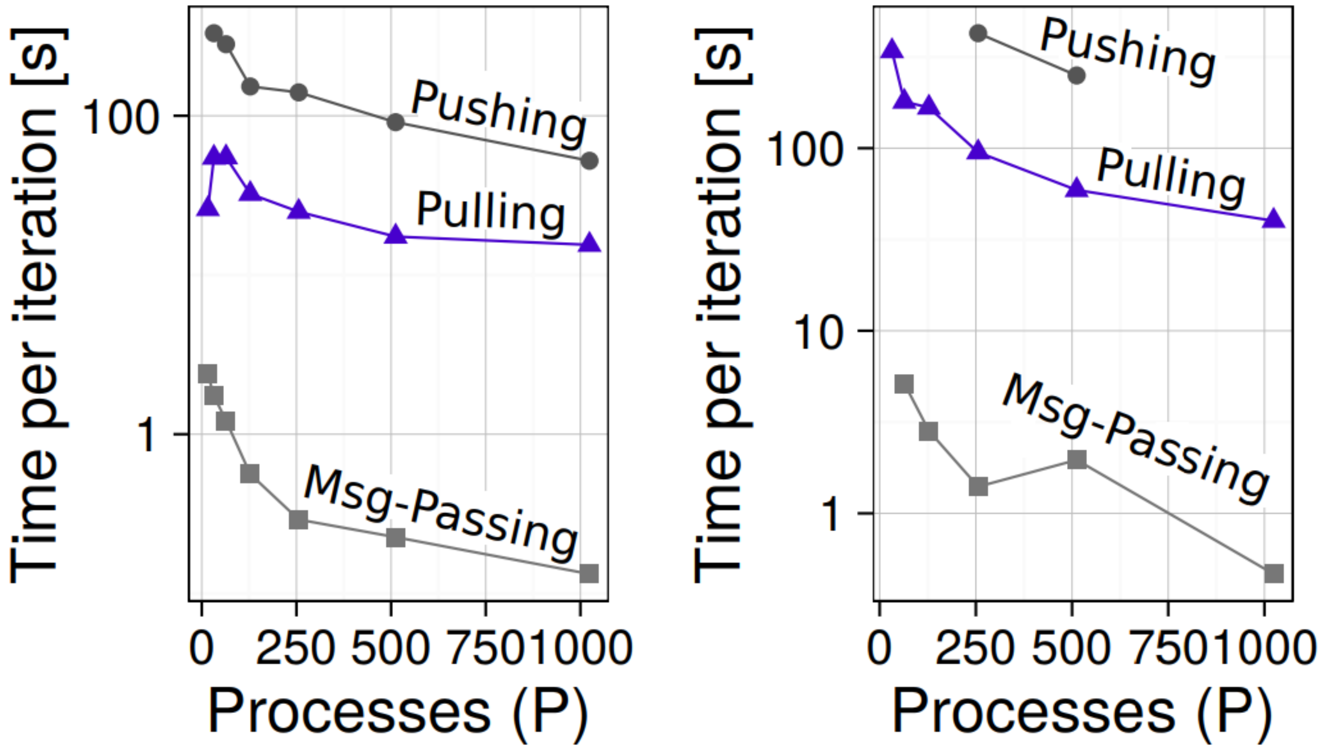
PAGERANK

Kronecker graphs

Distributed-Memory

$n = 2^{25}, m = 2^{27}$

$n = 2^{27}, m = 2^{29}$



M. Besta et al.: "To Push or To Pull: On Reducing Communication and Synchronization in Graph Computations", HPDC'17

PERFORMANCE ANALYSIS
PAGERANK

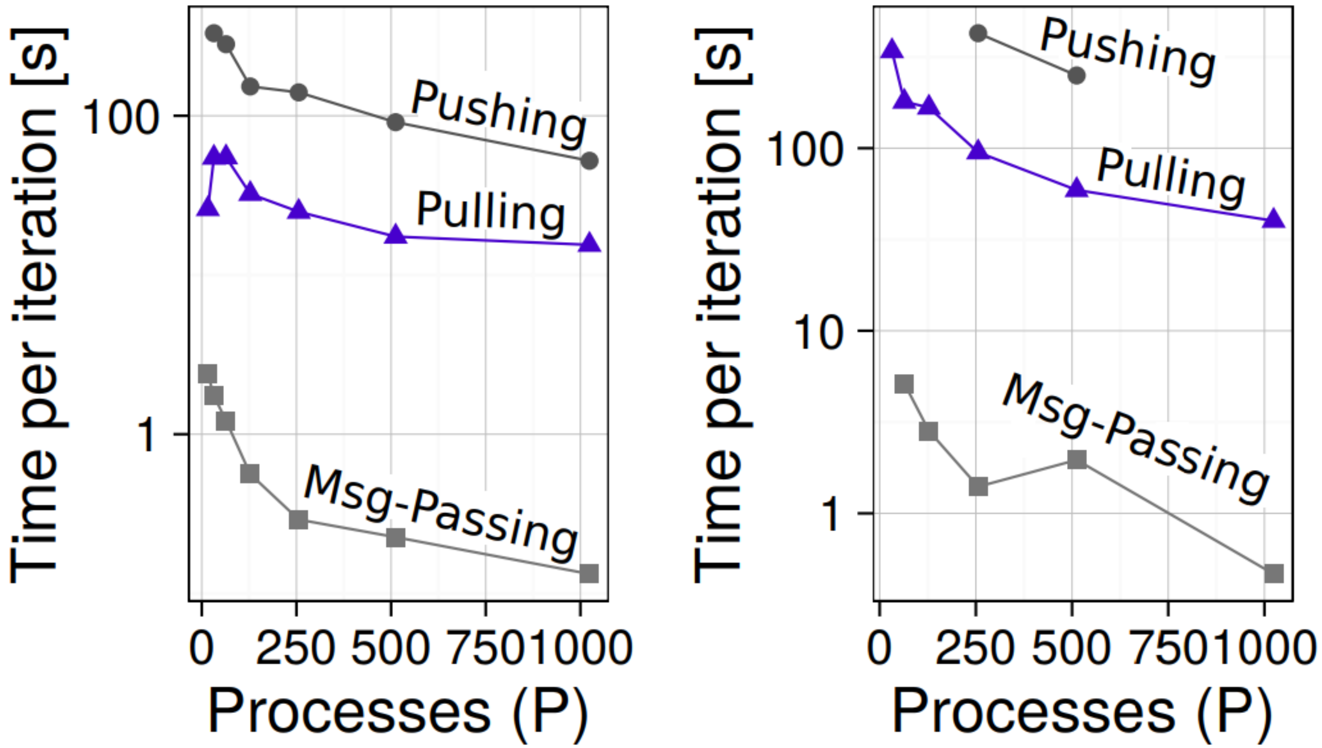
Kronecker graphs

Distributed-Memory

Msg-Passing fastest

$n = 2^{25}, m = 2^{27}$

$n = 2^{27}, m = 2^{29}$



M. Besta et al.: “To Push or To Pull: On Reducing Communication and Synchronization in Graph Computations”, HPDC’17

Performance Analysis
PageRank

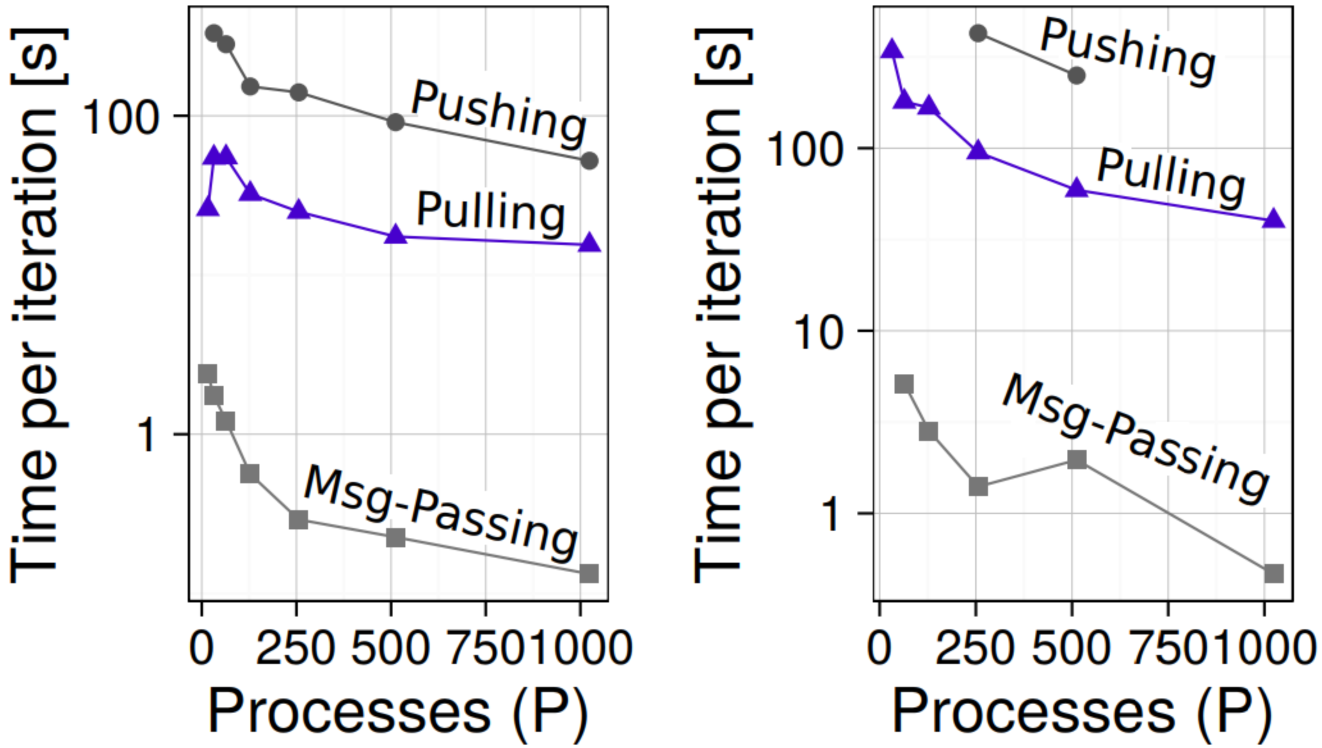
Kronecker graphs

Distributed-Memory

Msg-Passing fastest

Pulling incurs more communication while pushing expensive underlying locking

$n = 2^{25}, m = 2^{27}$ $n = 2^{27}, m = 2^{29}$



M. Besta et al.: "To Push or To Pull: On Reducing Communication and Synchronization in Graph Computations", HPDC'17

PERFORMANCE ANALYSIS
PAGERANK

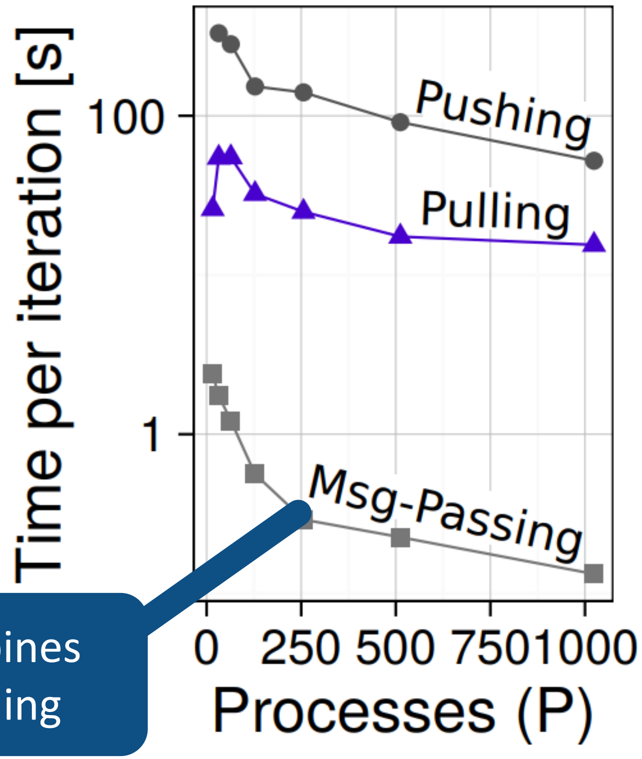
Kronecker graphs

Distributed-Memory

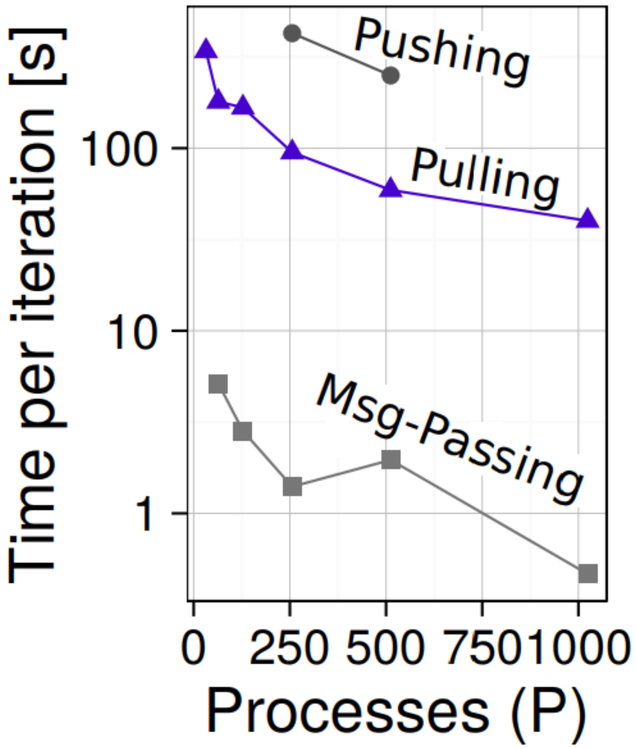
Msg-Passing fastest

Pulling incurs more communication while pushing expensive underlying locking

$n = 2^{25}, m = 2^{27}$



$n = 2^{27}, m = 2^{29}$



Collectives: combines pushing and pulling

M. Besta et al.: "To Push or To Pull: On Reducing Communication and Synchronization in Graph Computations", HPDC'17

To Push or To Pull?

M. Besta et al.: "To Push or To Pull: On Reducing Communication and Synchronization in Graph Computations", HPDC'17

To Push or To Pull?

If the complexities match: pull

M. Besta et al.: “To Push or To Pull: On Reducing Communication and Synchronization in Graph Computations”, HPDC’17

To Push or To Pull?

If the complexities match: pull

Otherwise: push

M. Besta et al.: “To Push or To Pull: On Reducing Communication and Synchronization in Graph Computations”, HPDC’17

To Push or To Pull?

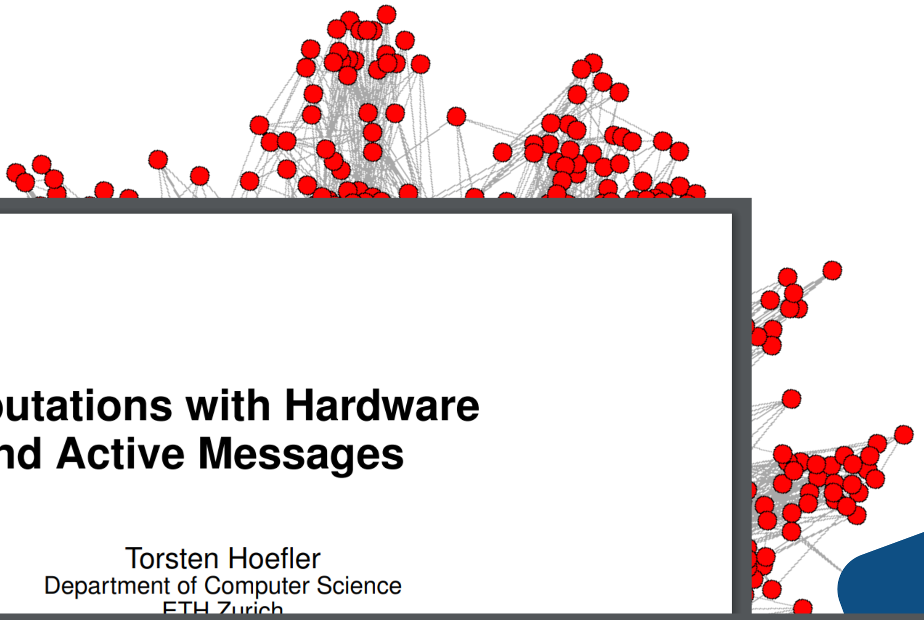
If the complexities match: pull

Otherwise: push

+ check your hardware ☺

M. Besta et al.: “To Push or To Pull: On Reducing Communication and Synchronization in Graph Computations”, HPDC’17

Moving on …



chronization-heavy

Accelerating Irregular Computations with Hardware Transactional Memory and Active Messages

Maciej Besta
Department of Computer Science
ETH Zurich

Torsten Hoefler
Department of Computer Science
ETH Zurich

To Push or To … Synchr…

Maciej Besta[1], Michał …
[1] Department of Computer Science…
[4] Departm…
maciej.besta@inf.ethz.ch, michal.p…

ABSTRACT

We reduce the cost of communication an…
processing by analyzing the fastest way t…
the updates to a shared state or pulling…

High-Performance Distributed RMA Locks

Patrick Schmid*
Department of Computer Science
ETH Zurich
patrick.schmid@ieffects.com

Maciej Besta*
Department of Computer Science
ETH Zurich
bestam@inf.ethz.ch

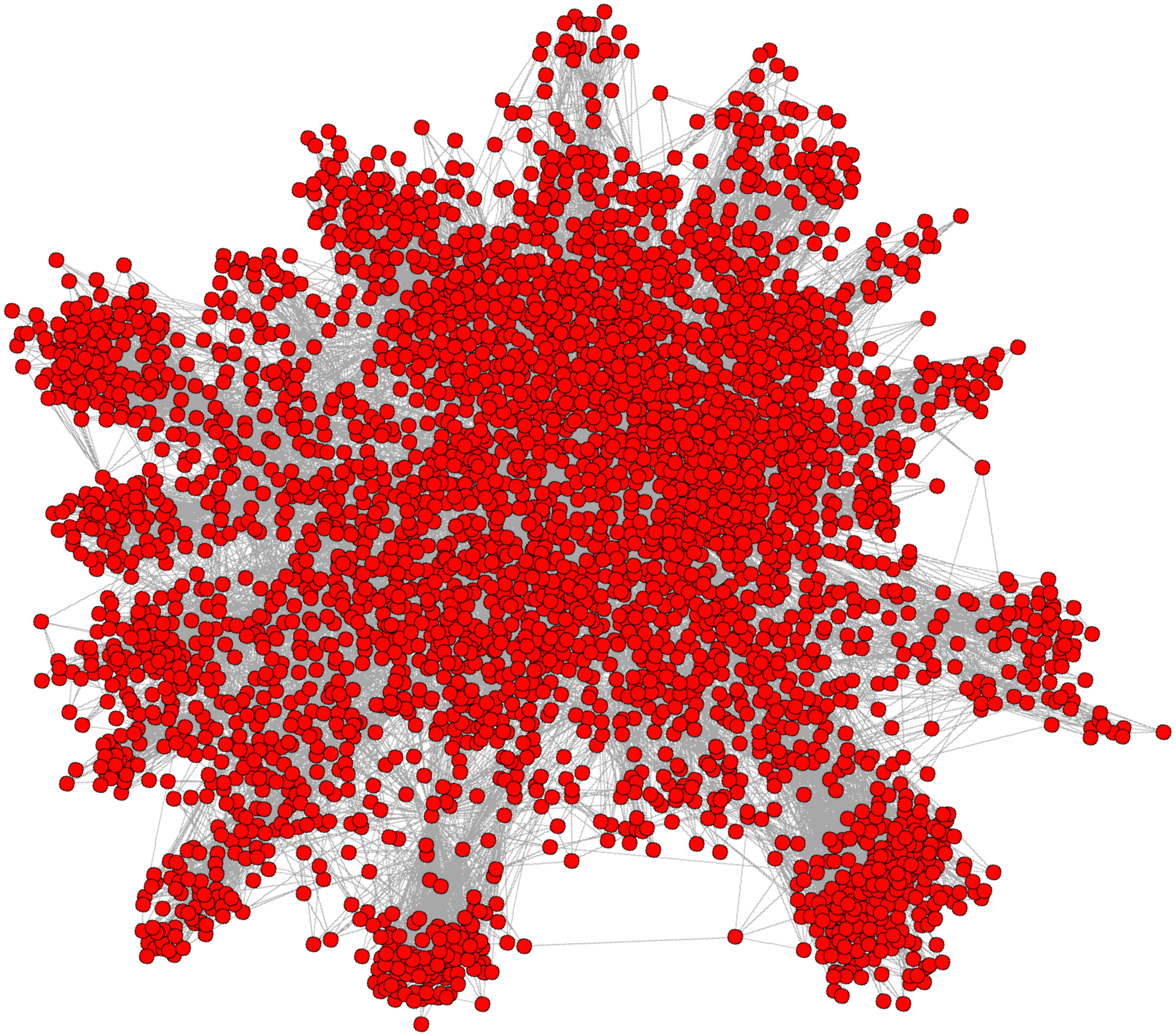
Torsten Hoefler
Department of Computer Science
ETH Zurich
htor@inf.ethz.ch

ABSTRACT

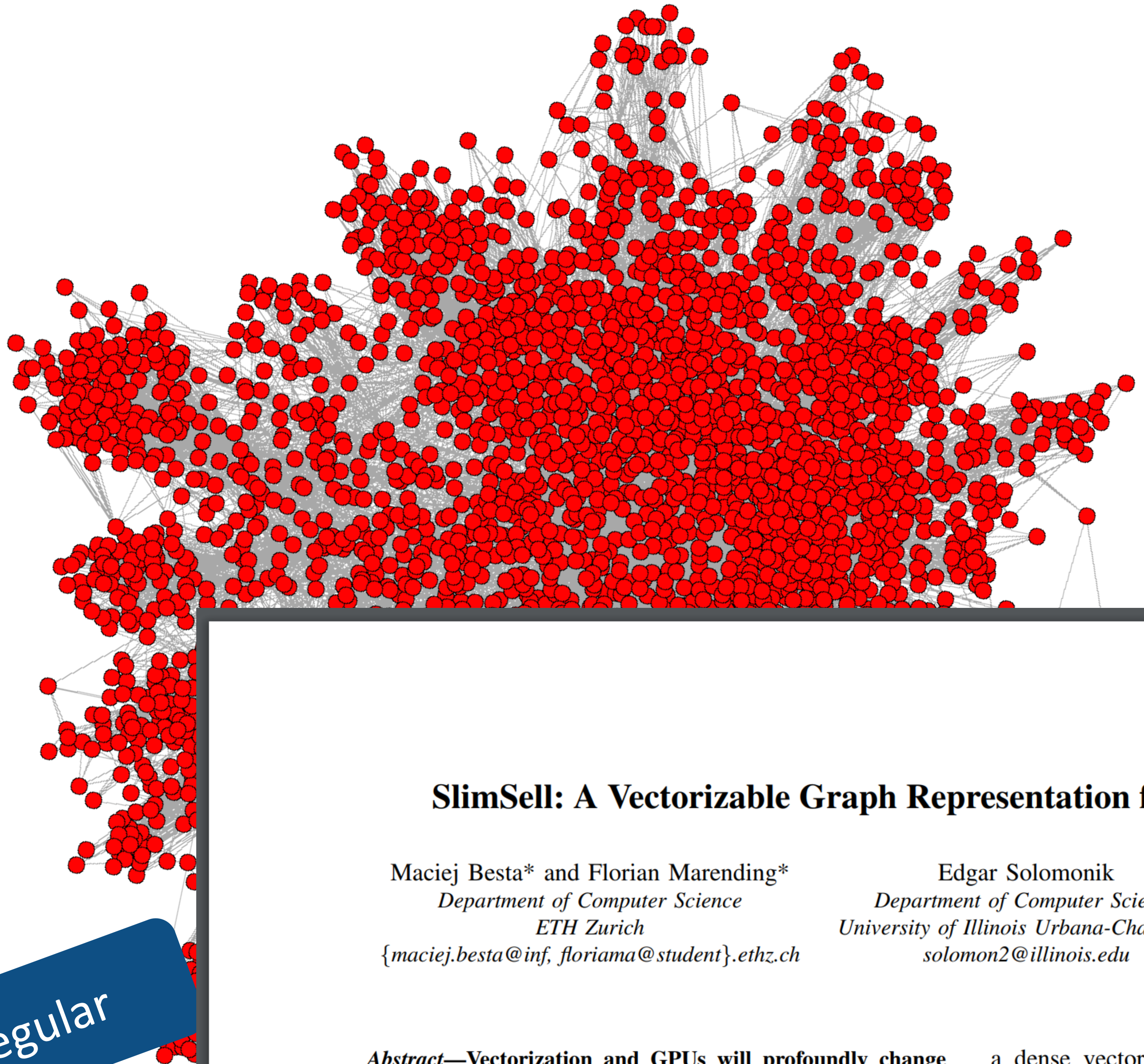
We propose a topology-aware distributed Reader-Writer lock …

cesses competing for the same lock. Assume that two of them (A and B) run on one socket and the remaining two (C …

Moving on …



Moving on …



Irregular

SlimSell: A Vectorizable Graph Representation for Breadth-First Search

Maciej Besta* and Florian Marending*
Department of Computer Science
ETH Zurich
{maciej.besta@inf, floriama@student}.ethz.ch

Edgar Solomonik
Department of Computer Science
University of Illinois Urbana-Champaign
solomon2@illinois.edu

Torsten Hoefler
Department of Computer Science
ETH Zurich
htor@inf.ethz.ch

***Abstract*—Vectorization and GPUs will profoundly change graph processing. Traditional graph algorithms tuned for 32- or 64-bit based memory accesses will be inefficient on architectures with 512-bit wide (or larger) instruction units that are already present in the Intel Knights Landing (KNL) manycore CPU. Anticipating this shift, we propose SlimSell: a**

a dense vector (SpMV) or a sparse matrix and a sparse vector (SpMSpV). BFS based on SpMV (BFS-SpMV) uses no explicit locking or atomics and has a succinct description as well as good locality [13]. Yet, it needs more work than traditional BFS and BFS based on SpMSpV [29]

VECTORIZATION

VECTORIZATION

- Deployed in various hardware

Vectorization

- Deployed in various hardware
- Becoming more popular

VECTORIZATION

- Deployed in various hardware
- Becoming more popular

AVX

$C = 8$ (SIMD width)

VECTORIZATION

- Deployed in various hardware
- Becoming more popular

AVX

$C = 16$ (SIMD width)

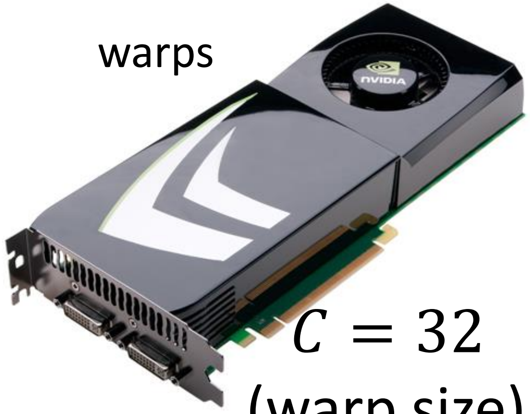
AVX

$C = 8$ (SIMD width)

VECTORIZATION

- Deployed in various hardware
- Becoming more popular

warps



$C = 32$
(warp size)

AVX

$C = 16$ (SIMD width)

AVX

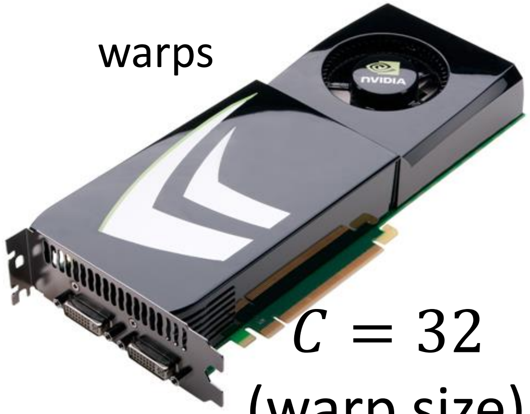
$C = 8$ (SIMD width)

C: „Chunk" size: SIMD width (CPUs, KNLs), warp size (GPUs)

Vectorization

- Deployed in various hardware
- Becoming more popular
- Offers a lot of „regular" compute power

warps



$C = 32$
(warp size)

AVX

$C = 16$ (SIMD width)

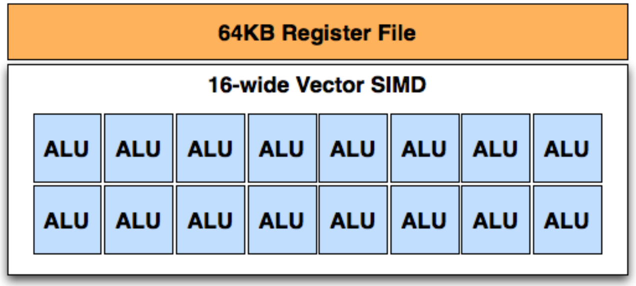
AVX

$C = 8$ (SIMD width)

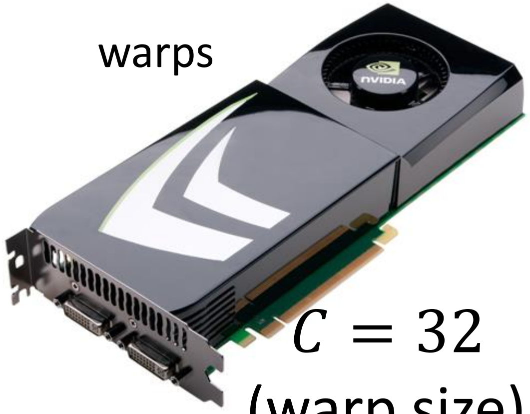
C: **„Chunk" size: SIMD width (CPUs, KNLs), warp size (GPUs)**

VECTORIZATION

- Deployed in various hardware
- Becoming more popular
- Offers a lot of „regular" compute power



warps



$C = 32$ (warp size)

AVX

$C = 16$ (SIMD width)

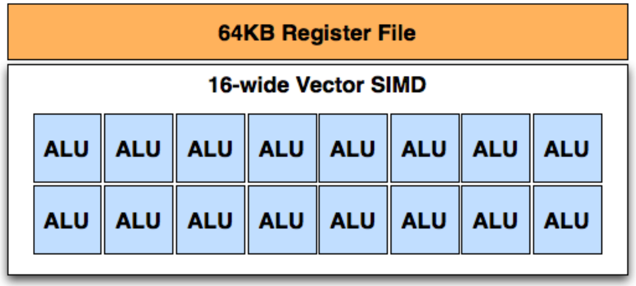
AVX

$C = 8$ (SIMD width)

C: **„Chunk" size: SIMD width (CPUs, KNLs), warp size (GPUs)**

VECTORIZATION

- Deployed in various hardware
- Becoming more popular
- Offers a lot of „regular” compute power



Regular

warps

$C = 32$
(warp size)

AVX

$C = 16$ (SIMD width)

AVX

$C = 8$ (SIMD width)

C: **„Chunk” size: SIMD width (CPUs, KNLs), warp size (GPUs)**

VECTORIZATION

- Deployed in various hardware
- Becoming more popular
- Offers a lot of „regular” compute power

Regular

warps

$C = 32$ (warp size)

AVX

$C = 16$ (SIMD width)

AVX

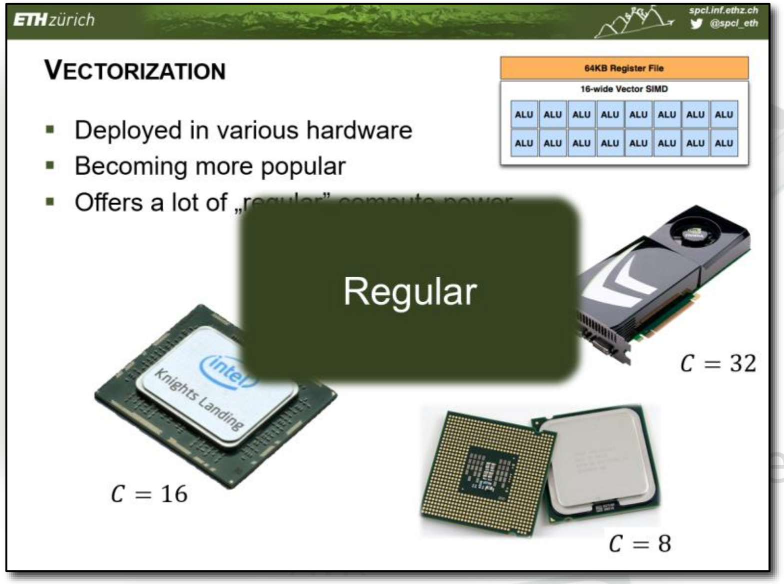
$C = 8$ (SIMD width)

C: „Chunk” size: SIMD width (CPUs, KNLs), warp size (GPUs)

VECTORIZATION

- Deployed in various hardware
- Becoming more popular
- Offers a lot of „regular” compute power

+



AVX

$C = 16$ (SIMD width)

$C = 8$ (SIMD width)

C: „Chunk” size: SIMD width (CPUs, KNLs), warp size (GPUs)

Breadth-First Search

Traditional Formulation

M. Besta et al.: "SlimSell: A Vectorized Graph Representation for Breadth-First Search", IPDPS'17

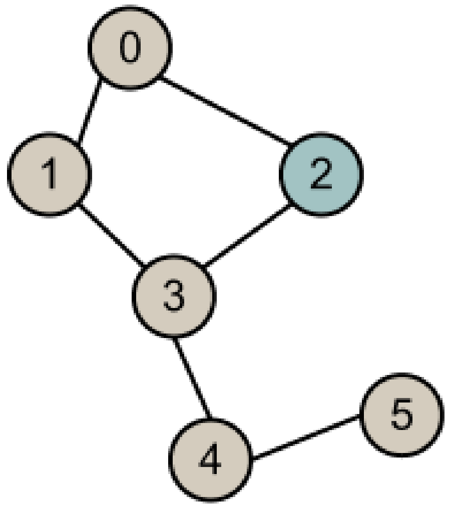
Breadth-First Search
Traditional Formulation

- BFS is based on primitives such as queues

M. Besta et al.: "SlimSell: A Vectorized Graph Representation for Breadth-First Search", IPDPS'17

Breadth-First Search
Traditional Formulation

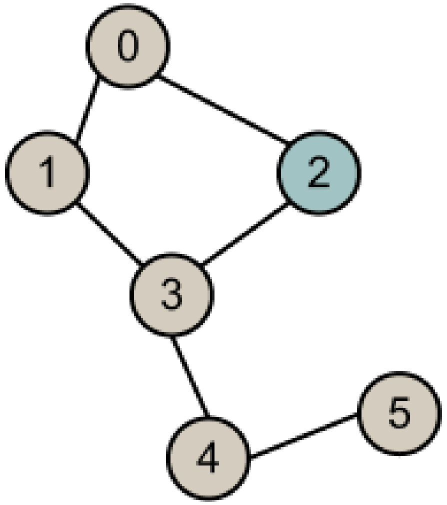
- BFS is based on primitives such as queues



M. Besta et al.: "SlimSell: A Vectorized Graph Representation for Breadth-First Search", IPDPS'17

Breadth-First Search
Traditional Formulation

- BFS is based on primitives such as queues

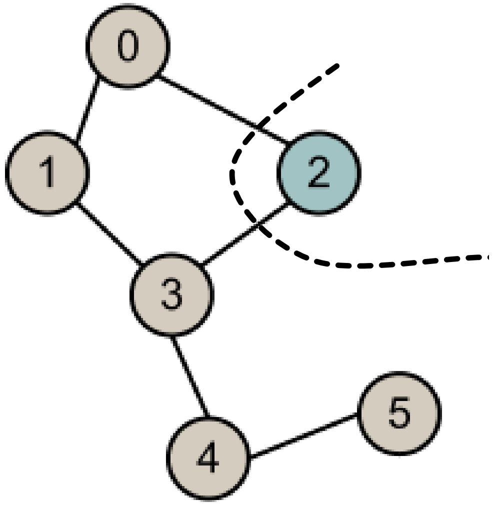


1) F = {}

M. Besta et al.: “SlimSell: A Vectorized Graph Representation for Breadth-First Search”, IPDPS’17

Breadth-First Search
Traditional Formulation

- BFS is based on primitives such as queues

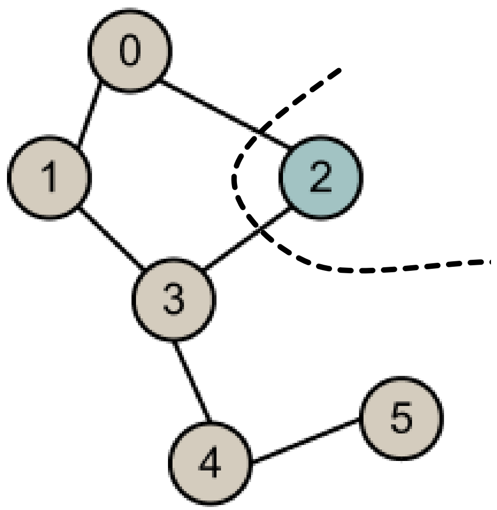


1) F = {}

M. Besta et al.: "SlimSell: A Vectorized Graph Representation for Breadth-First Search", IPDPS'17

Breadth-First Search
Traditional Formulation

- BFS is based on primitives such as queues



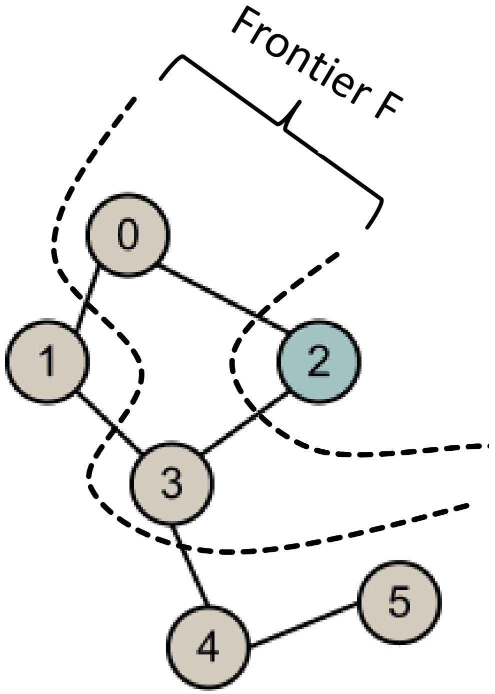
1) F = {}
2) F = {2}

M. Besta et al.: "SlimSell: A Vectorized Graph Representation for Breadth-First Search", IPDPS'17

Breadth-First Search

Traditional Formulation

- BFS is based on primitives such as queues

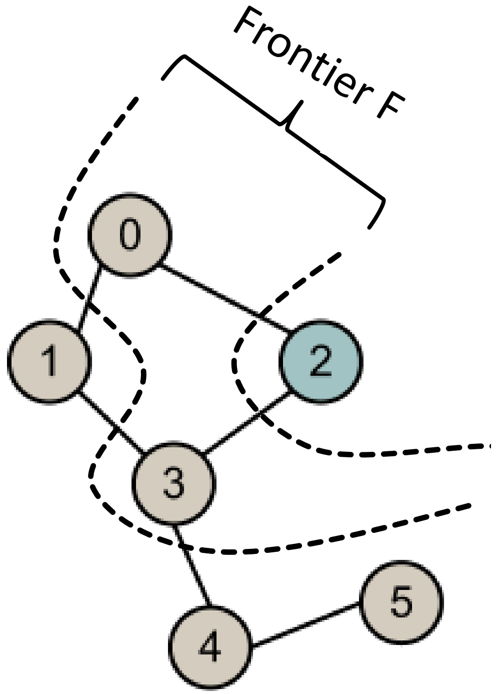


1) F = {}
2) F = {2}

M. Besta et al.: “SlimSell: A Vectorized Graph Representation for Breadth-First Search”, IPDPS’17

Breadth-First Search
Traditional Formulation

- BFS is based on primitives such as queues



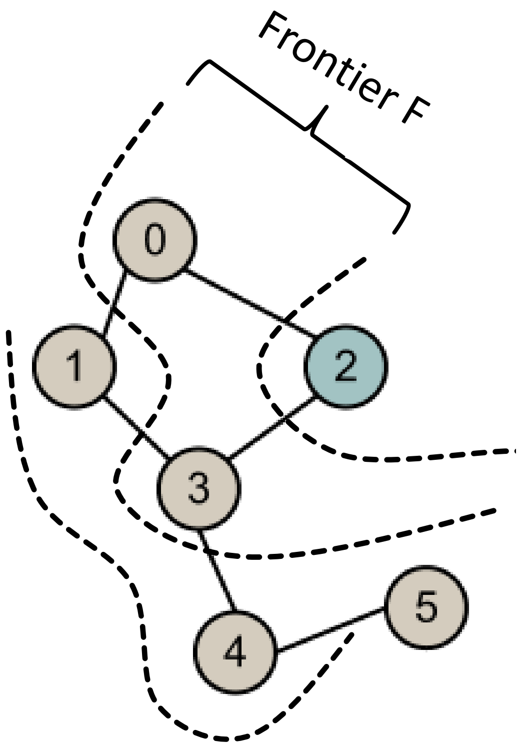
1) F = {}
2) F = {2}
3) F = {0,3}

M. Besta et al.: "SlimSell: A Vectorized Graph Representation for Breadth-First Search", IPDPS'17

Breadth-First Search

Traditional Formulation

- BFS is based on primitives such as queues

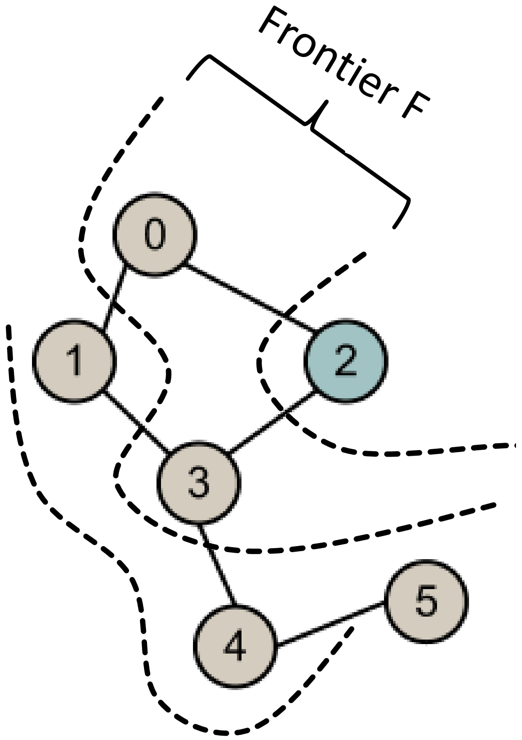


1) F = {}
2) F = {2}
3) F = {0,3}

M. Besta et al.: “SlimSell: A Vectorized Graph Representation for Breadth-First Search”, IPDPS’17

Breadth-First Search
Traditional Formulation

- BFS is based on primitives such as queues

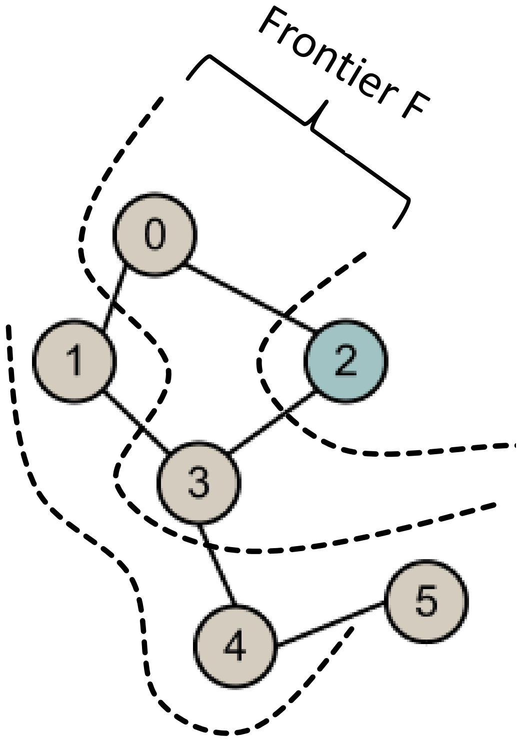


1) F = {}
2) F = {2}
3) F = {0,3}
4) F = {1,4}

M. Besta et al.: "SlimSell: A Vectorized Graph Representation for Breadth-First Search", IPDPS'17

Breadth-First Search
Traditional Formulation

- BFS is based on primitives such as queues



1) F = {}
2) F = {2}
3) F = {0,3}
4) F = {1,4}

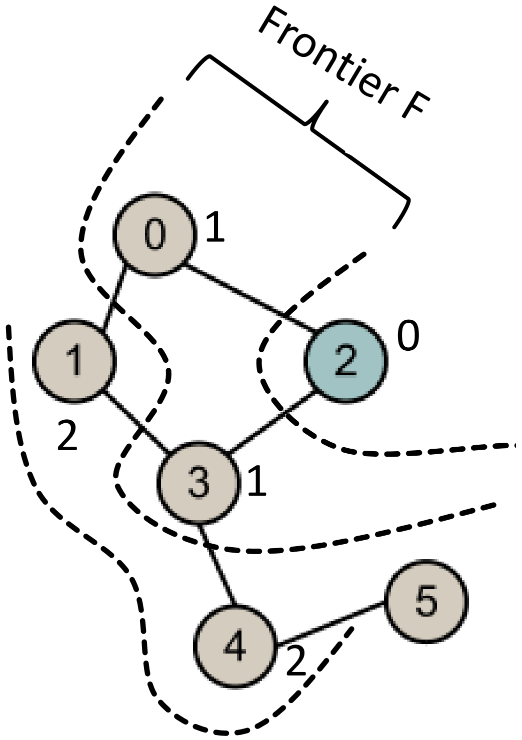
Distances from the root

M. Besta et al.: "SlimSell: A Vectorized Graph Representation for Breadth-First Search", IPDPS'17

Breadth-First Search

Traditional Formulation

- BFS is based on primitives such as queues



1) F = {}
2) F = {2}
3) F = {0,3}
4) F = {1,4}

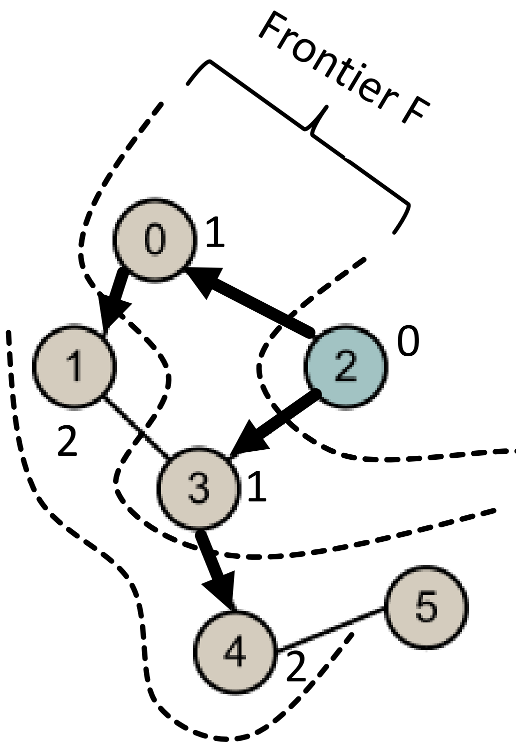
Distances from the root

M. Besta et al.: "SlimSell: A Vectorized Graph Representation for Breadth-First Search", IPDPS'17

Breadth-First Search

Traditional Formulation

- BFS is based on primitives such as queues



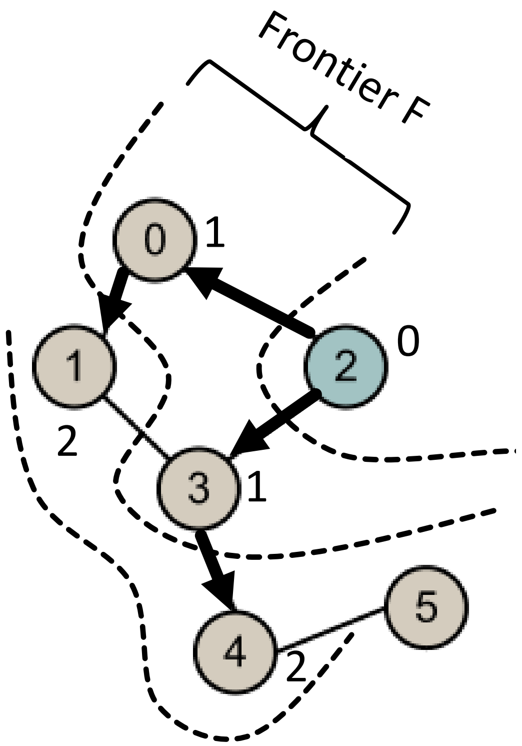
1) F = {}
2) F = {2}
3) F = {0,3}
4) F = {1,4}

Distances from the root

M. Besta et al.: "SlimSell: A Vectorized Graph Representation for Breadth-First Search", IPDPS'17

Breadth-First Search
Traditional Formulation

- BFS is based on primitives such as queues



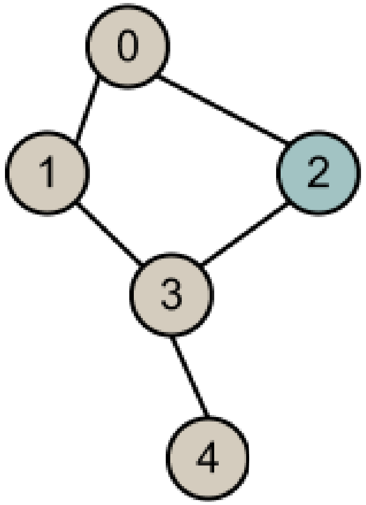
1) F = {}
2) F = {2}
3) F = {0,3}
4) F = {1,4}

Distances from the root

Parents (predecessors) in the traversal tree

M. Besta et al.: “SlimSell: A Vectorized Graph Representation for Breadth-First Search”, IPDPS’17

BREADTH-FIRST SEARCH
ALGEBRAIC FORMULATION

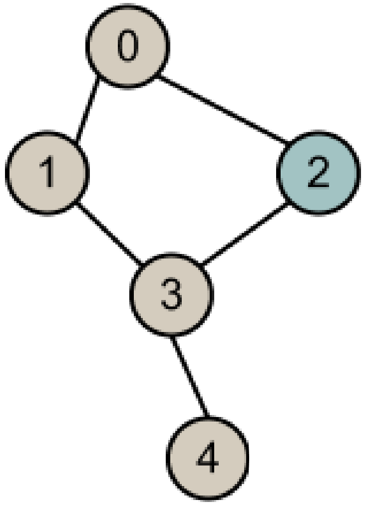


M. Besta et al.: "SlimSell: A Vectorized Graph Representation for Breadth-First Search", IPDPS'17

Breadth-First Search

Algebraic Formulation

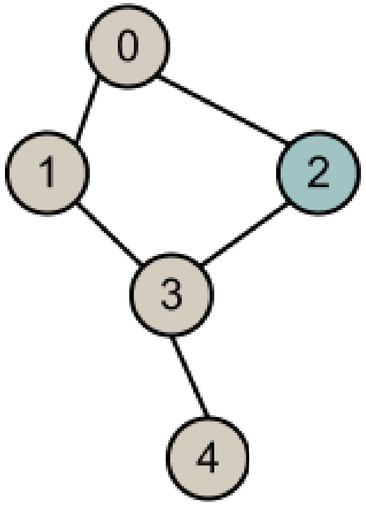
- BFS is a series of matrix-vector products



M. Besta et al.: “SlimSell: A Vectorized Graph Representation for Breadth-First Search”, IPDPS’17

Breadth-First Search
Algebraic Formulation

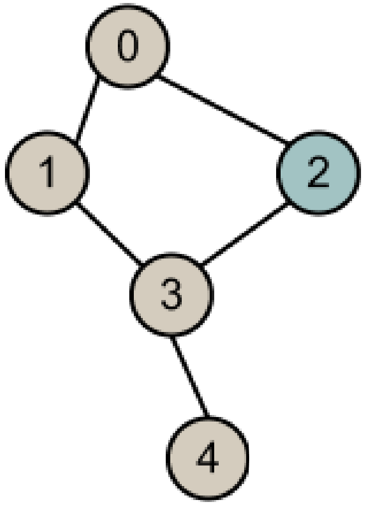
- BFS is a series of matrix-vector products
- Graph is modeled by an adjacency matrix



M. Besta et al.: “SlimSell: A Vectorized Graph Representation for Breadth-First Search”, IPDPS’17

Breadth-First Search
Algebraic Formulation

- BFS is a series of matrix-vector products
- Graph is modeled by an adjacency matrix



Adjacency Matrix:

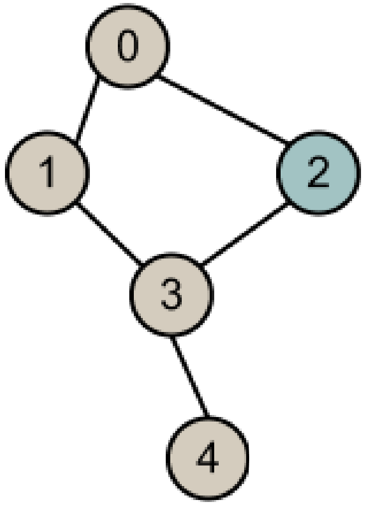
$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

M. Besta et al.: “SlimSell: A Vectorized Graph Representation for Breadth-First Search”, IPDPS’17

Breadth-First Search

Algebraic Formulation

- BFS is a series of matrix-vector products
- Graph is modeled by an adjacency matrix
- Multiplication is done over a semiring



Adjacency Matrix:

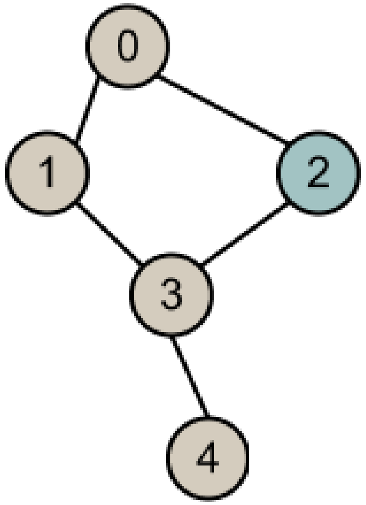
$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

M. Besta et al.: "SlimSell: A Vectorized Graph Representation for Breadth-First Search", IPDPS'17

Breadth-First Search

Algebraic Formulation

- BFS is a series of matrix-vector products
- Graph is modeled by an adjacency matrix
- Multiplication is done over a semiring



Adjacency Matrix:

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

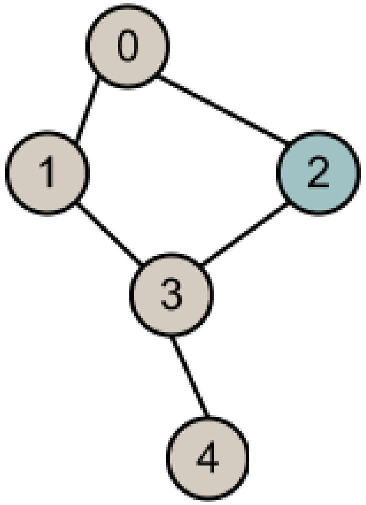
Semiring:

$$(\mathbb{R}\,, op_1, op_2, el_1, el_2)$$

M. Besta et al.: "SlimSell: A Vectorized Graph Representation for Breadth-First Search", IPDPS'17

Breadth-First Search
Algebraic Formulation

- BFS is a series of matrix-vector products
- Graph is modeled by an adjacency matrix
- Multiplication is done over a semiring



Adjacency Matrix:

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Semiring:

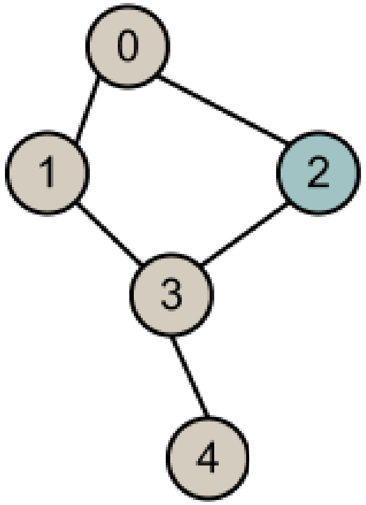
$$(\mathbb{R}, op_1, op_2, el_1, el_2)$$

$$\begin{bmatrix} 1 & 2 \\ 3 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 2 \end{bmatrix} = \begin{bmatrix} 4 \\ 2 \end{bmatrix}$$

M. Besta et al.: “SlimSell: A Vectorized Graph Representation for Breadth-First Search”, IPDPS’17

Breadth-First Search
Algebraic Formulation

- BFS is a series of matrix-vector products
- Graph is modeled by an adjacency matrix
- Multiplication is done over a semiring



Adjacency Matrix:

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Semiring:

$$(\mathbb{R}\,, op_1, op_2, el_1, el_2)$$

$$(\mathbb{R}, +, \cdot, 0, 1)$$

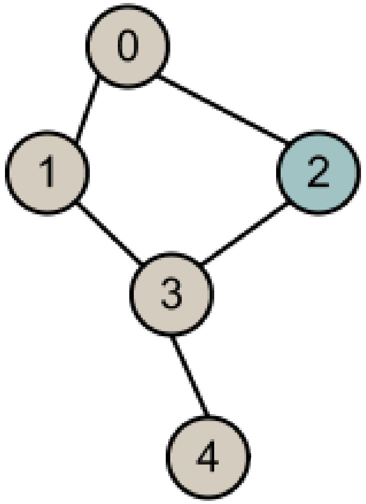
$$\begin{bmatrix} 1 & 2 \\ 3 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 2 \end{bmatrix} = \begin{bmatrix} 4 \\ 2 \end{bmatrix}$$

M. Besta et al.: “SlimSell: A Vectorized Graph Representation for Breadth-First Search”, IPDPS’17

Breadth-First Search
Algebraic Formulation

Tropical Semiring

$(\mathbb{R} \cup \{\infty\}, min, +, \infty, 0)$

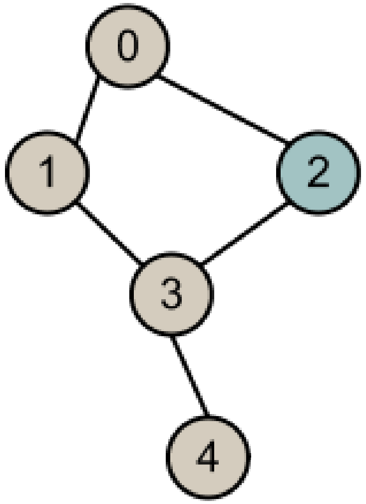


M. Besta et al.: "SlimSell: A Vectorized Graph Representation for Breadth-First Search", IPDPS'17

Breadth-First Search
Algebraic Formulation

Tropical Semiring

$(\mathbb{R} \cup \{\infty\}, min, +, \infty, 0)$



$$A' = \begin{bmatrix} 0 & 1 & 1 & \infty & \infty \\ 1 & 0 & \infty & 1 & \infty \\ 1 & \infty & 0 & 1 & \infty \\ \infty & 1 & 1 & 0 & 1 \\ \infty & \infty & \infty & 1 & 0 \end{bmatrix}$$

M. Besta et al.: “SlimSell: A Vectorized Graph Representation for Breadth-First Search”, IPDPS’17

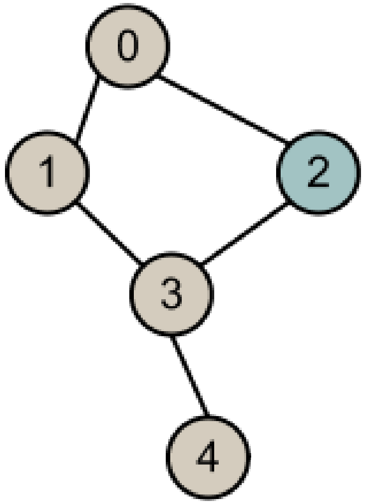
Breadth-First Search

Algebraic Formulation

Tropical Semiring

$$(\mathbb{R} \cup \{\infty\}, min, +, \infty, 0)$$

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$



$$A' = \begin{bmatrix} 0 & 1 & 1 & \infty & \infty \\ 1 & 0 & \infty & 1 & \infty \\ 1 & \infty & 0 & 1 & \infty \\ \infty & 1 & 1 & 0 & 1 \\ \infty & \infty & \infty & 1 & 0 \end{bmatrix}$$

M. Besta et al.: "SlimSell: A Vectorized Graph Representation for Breadth-First Search", IPDPS'17

Breadth-First Search

Algebraic Formulation

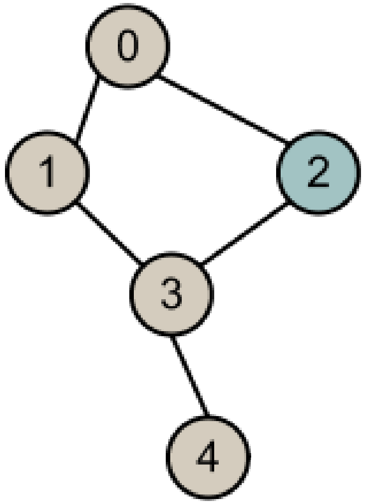
Tropical Semiring

$(\mathbb{R} \cup \{\infty\}, min, +, \infty, 0)$

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Usually stored using a sparse format

$$A' = \begin{bmatrix} 0 & 1 & 1 & \infty & \infty \\ 1 & 0 & \infty & 1 & \infty \\ 1 & \infty & 0 & 1 & \infty \\ \infty & 1 & 1 & 0 & 1 \\ \infty & \infty & \infty & 1 & 0 \end{bmatrix}$$



M. Besta et al.: "SlimSell: A Vectorized Graph Representation for Breadth-First Search", IPDPS'17

Breadth-First Search
Algebraic Formulation

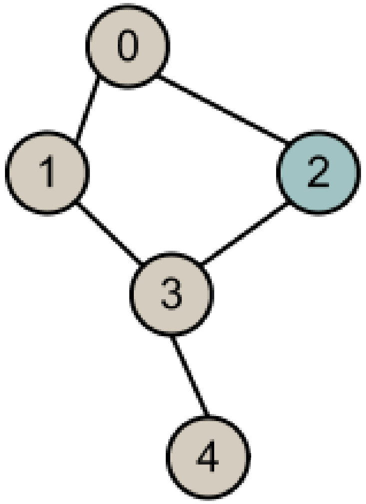
Tropical Semiring

$(\mathbb{R} \cup \{\infty\}, min, +, \infty, 0)$

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Usually stored using a sparse format

$$A' = \begin{bmatrix} 0 & 1 & 1 & \infty & \infty \\ 1 & 0 & \infty & 1 & \infty \\ 1 & \infty & 0 & 1 & \infty \\ \infty & 1 & 1 & 0 & 1 \\ \infty & \infty & \infty & 1 & 0 \end{bmatrix} \qquad f_0 = \begin{pmatrix} \infty \\ \infty \\ 0 \\ \infty \\ \infty \end{pmatrix}$$



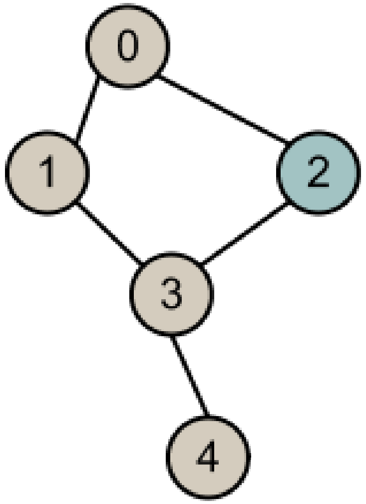
M. Besta et al.: "SlimSell: A Vectorized Graph Representation for Breadth-First Search", IPDPS'17

Breadth-First Search
Algebraic Formulation

Tropical Semiring

$(\mathbb{R} \cup \{\infty\}, min, +, \infty, 0)$

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$



Usually stored using a sparse format

Stored with a dense or a sparse format

$$A' = \begin{bmatrix} 0 & 1 & 1 & \infty & \infty \\ 1 & 0 & \infty & 1 & \infty \\ 1 & \infty & 0 & 1 & \infty \\ \infty & 1 & 1 & 0 & 1 \\ \infty & \infty & \infty & 1 & 0 \end{bmatrix} \qquad f_0 = \begin{pmatrix} \infty \\ \infty \\ 0 \\ \infty \\ \infty \end{pmatrix}$$

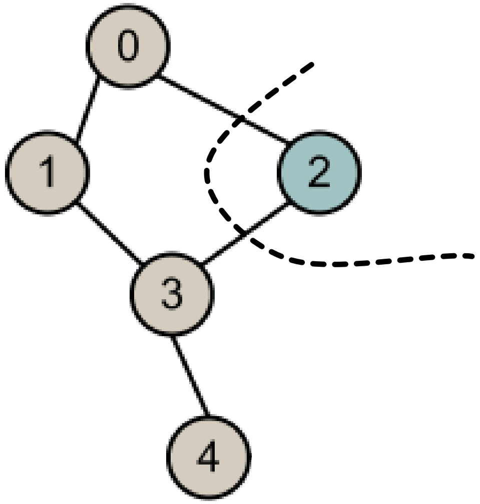
M. Besta et al.: “SlimSell: A Vectorized Graph Representation for Breadth-First Search”, IPDPS’17

Breadth-First Search
Algebraic Formulation

Tropical Semiring

$(\mathbb{R} \cup \{\infty\}, min, +, \infty, 0)$

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$



Usually stored using a sparse format

$$A' = \begin{bmatrix} 0 & 1 & 1 & \infty & \infty \\ 1 & 0 & \infty & 1 & \infty \\ 1 & \infty & 0 & 1 & \infty \\ \infty & 1 & 1 & 0 & 1 \\ \infty & \infty & \infty & 1 & 0 \end{bmatrix}$$

Stored with a dense or a sparse format

$$f_0 = \begin{pmatrix} \infty \\ \infty \\ 0 \\ \infty \\ \infty \end{pmatrix}$$

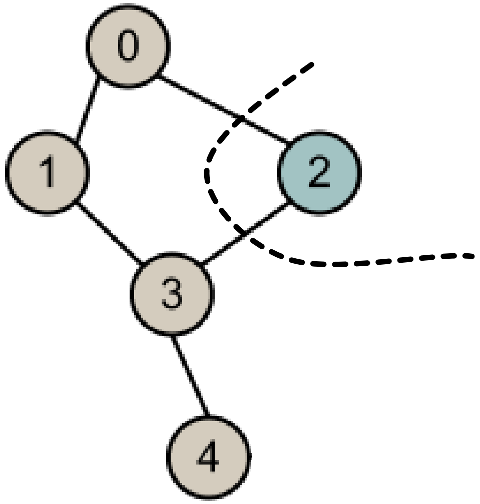
M. Besta et al.: "SlimSell: A Vectorized Graph Representation for Breadth-First Search", IPDPS'17

Breadth-First Search
Algebraic Formulation

Tropical Semiring

$(\mathbb{R} \cup \{\infty\}, min, +, \infty, 0)$

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$



Usually stored using a sparse format

$$A' = \begin{bmatrix} 0 & 1 & 1 & \infty & \infty \\ 1 & 0 & \infty & 1 & \infty \\ 1 & \infty & 0 & 1 & \infty \\ \infty & 1 & 1 & 0 & 1 \\ \infty & \infty & \infty & 1 & 0 \end{bmatrix}$$

Stored with a dense or a sparse format

$$f_0 = \begin{pmatrix} \infty \\ \infty \\ 0 \\ \infty \\ \infty \end{pmatrix}$$

$$f_1 = A'^T \otimes_T f_0 = \begin{pmatrix} \\ \\ \\ \\ \\ \end{pmatrix}$$

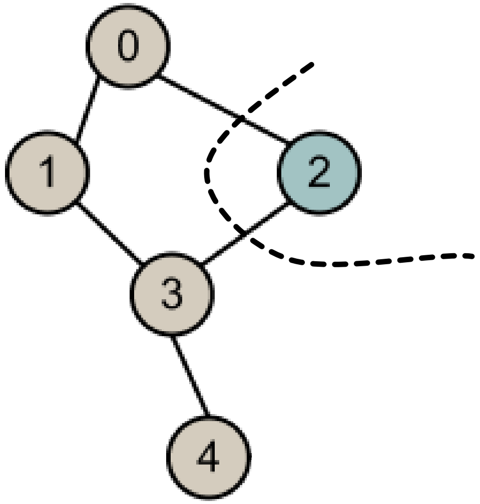
Breadth-First Search

Algebraic Formulation

Tropical Semiring

$(\mathbb{R} \cup \{\infty\}, min, +, \infty, 0)$

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$



Usually stored using a sparse format

$$A' = \begin{bmatrix} 0 & 1 & 1 & \infty & \infty \\ 1 & 0 & \infty & 1 & \infty \\ 1 & \infty & 0 & 1 & \infty \\ \infty & 1 & 1 & 0 & 1 \\ \infty & \infty & \infty & 1 & 0 \end{bmatrix}$$

Stored with a dense or a sparse format

$$f_0 = \begin{pmatrix} \infty \\ \infty \\ 0 \\ \infty \\ \infty \end{pmatrix}$$

$$f_1 = A'^T \otimes_T f_0 = \begin{pmatrix} \\ \\ \\ \end{pmatrix}$$

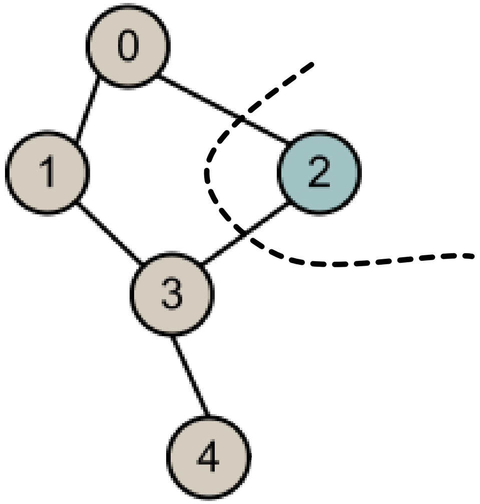
M. Besta et al.: "SlimSell: A Vectorized Graph Representation for Breadth-First Search", IPDPS'17

Breadth-First Search
Algebraic Formulation

Tropical Semiring

$(\mathbb{R} \cup \{\infty\}, min, +, \infty, 0)$

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$



Usually stored using a sparse format

Stored with a dense or a sparse format

$$A' = \begin{bmatrix} 0 & 1 & 1 & \infty & \infty \\ 1 & 0 & \infty & 1 & \infty \\ 1 & \infty & 0 & 1 & \infty \\ \infty & 1 & 1 & 0 & 1 \\ \infty & \infty & \infty & 1 & 0 \end{bmatrix} \quad f_0 = \begin{pmatrix} \infty \\ \infty \\ 0 \\ \infty \\ \infty \end{pmatrix}$$

$$f_1 = A'^T \otimes_T f_0 = \begin{pmatrix} 1 \\ \\ \\ \\ \end{pmatrix}$$

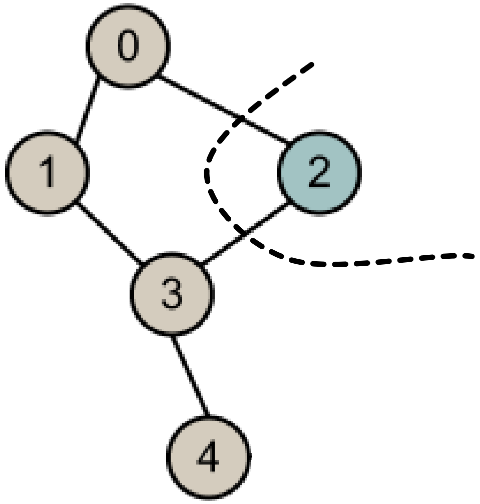
M. Besta et al.: "SlimSell: A Vectorized Graph Representation for Breadth-First Search", IPDPS'17

Breadth-First Search
Algebraic Formulation

Tropical Semiring

$(\mathbb{R} \cup \{\infty\}, min, +, \infty, 0)$

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$



Usually stored using a sparse format

Stored with a dense or a sparse format

$$A' = \begin{bmatrix} 0 & 1 & 1 & \infty & \infty \\ 1 & 0 & \infty & 1 & \infty \\ 1 & \infty & 0 & 1 & \infty \\ \infty & 1 & 1 & 0 & 1 \\ \infty & \infty & \infty & 1 & 0 \end{bmatrix} \qquad f_0 = \begin{pmatrix} \infty \\ \infty \\ 0 \\ \infty \\ \infty \end{pmatrix}$$

$$f_1 = A'^T \otimes_T f_0 = \begin{pmatrix} 1 \\ \infty \\ 0 \\ 1 \\ \infty \end{pmatrix}$$

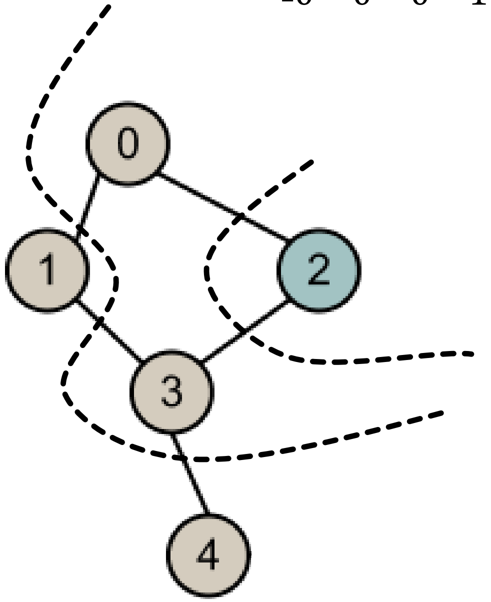
M. Besta et al.: “SlimSell: A Vectorized Graph Representation for Breadth-First Search”, IPDPS’17

Breadth-First Search
Algebraic Formulation

Tropical Semiring

$(\mathbb{R} \cup \{\infty\}, min, +, \infty, 0)$

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$



Usually stored using a sparse format

Stored with a dense or a sparse format

$$A' = \begin{bmatrix} 0 & 1 & 1 & \infty & \infty \\ 1 & 0 & \infty & 1 & \infty \\ 1 & \infty & 0 & 1 & \infty \\ \infty & 1 & 1 & 0 & 1 \\ \infty & \infty & \infty & 1 & 0 \end{bmatrix} \qquad f_0 = \begin{pmatrix} \infty \\ \infty \\ 0 \\ \infty \\ \infty \end{pmatrix}$$

$$f_1 = A'^T \otimes_T f_0 = \begin{pmatrix} 1 \\ \infty \\ 0 \\ 1 \\ \infty \end{pmatrix}$$

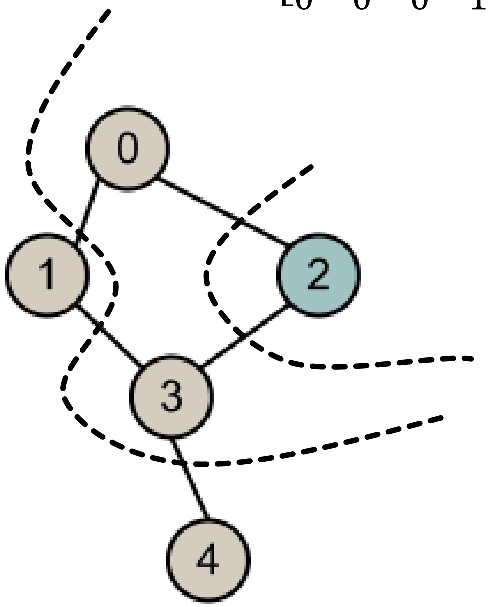
M. Besta et al.: “SlimSell: A Vectorized Graph Representation for Breadth-First Search”, IPDPS’17

Breadth-First Search
Algebraic Formulation

Tropical Semiring

$$(\mathbb{R} \cup \{\infty\}, min, +, \infty, 0)$$

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$



Usually stored using a sparse format

Stored with a dense or a sparse format

$$A' = \begin{bmatrix} 0 & 1 & 1 & \infty & \infty \\ 1 & 0 & \infty & 1 & \infty \\ 1 & \infty & 0 & 1 & \infty \\ \infty & 1 & 1 & 0 & 1 \\ \infty & \infty & \infty & 1 & 0 \end{bmatrix} \qquad f_0 = \begin{pmatrix} \infty \\ \infty \\ 0 \\ \infty \\ \infty \end{pmatrix}$$

$$f_1 = A'^T \otimes_T f_0 = \begin{pmatrix} 1 \\ \infty \\ 0 \\ 1 \\ \infty \end{pmatrix} \qquad f_2 = \begin{pmatrix} 1 \\ 2 \\ 0 \\ 1 \\ 2 \end{pmatrix}$$

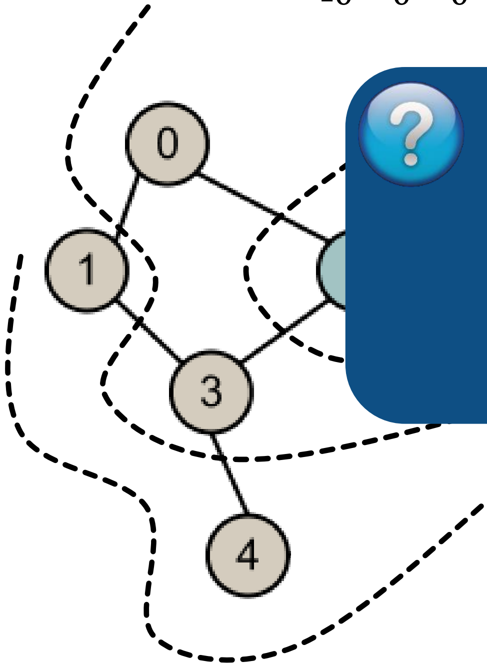
M. Besta et al.: “SlimSell: A Vectorized Graph Representation for Breadth-First Search”, IPDPS’17

Breadth-First Search
Algebraic Formulation

Tropical Semiring

$(\mathbb{R} \cup \{\infty\}, min, +, \infty, 0)$

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$



Usually stored using a sparse format

$$\begin{bmatrix} 0 & 1 & 1 & \infty & \infty \end{bmatrix}$$

Stored with a dense or a sparse format

$$f_0 = \begin{pmatrix} \infty \\ \infty \\ 0 \\ \infty \\ \infty \end{pmatrix}$$

How to do this in practice?

$$f_1 = A'^T \otimes_T f_0 = \begin{pmatrix} \infty \\ 0 \\ 1 \\ \infty \end{pmatrix}$$

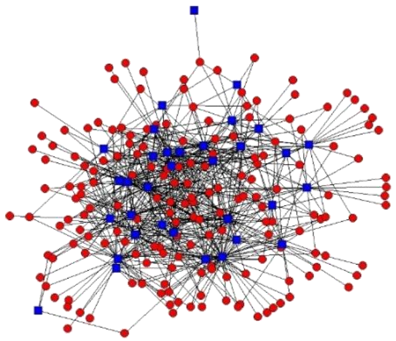
$$f_2 = \begin{pmatrix} 1 \\ 2 \\ 0 \\ 1 \\ 2 \end{pmatrix}$$

M. Besta et al.: "SlimSell: A Vectorized Graph Representation for Breadth-First Search", IPDPS'17

GRAPH REPRESENTATIONS
COMPRESSED SPARSE ROW (CSR)

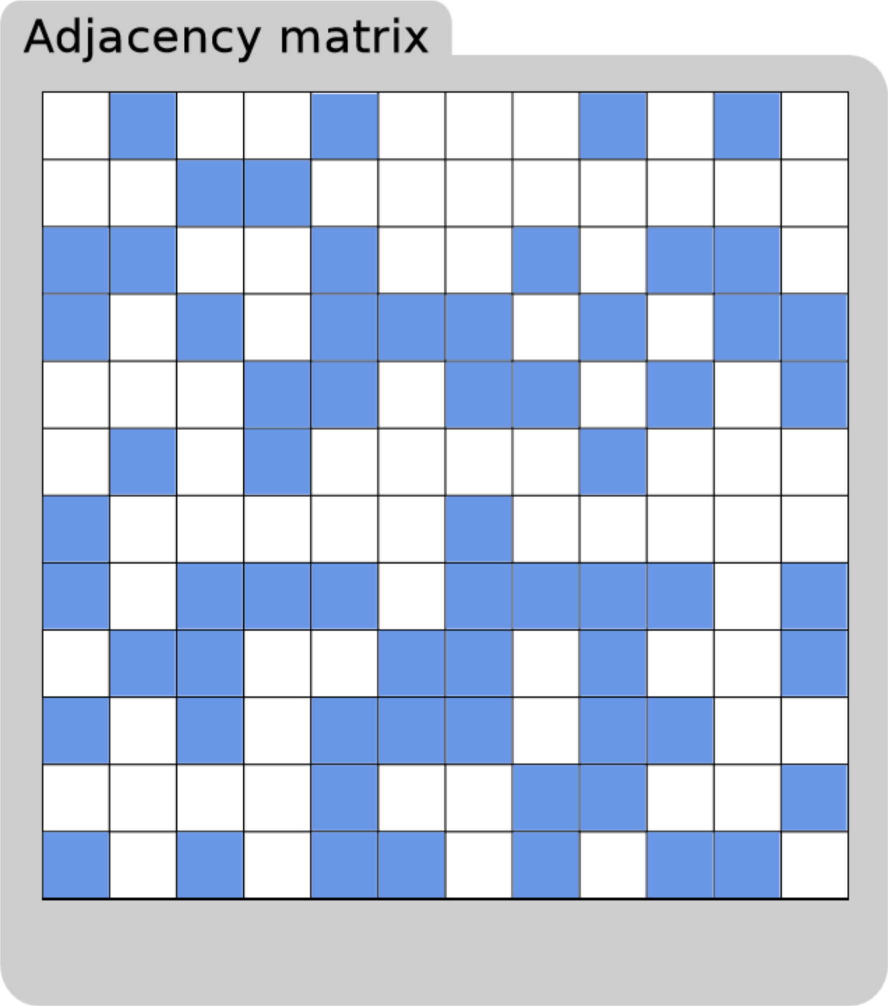
M. Besta et al.: “SlimSell: A Vectorized Graph Representation for Breadth-First Search”, IPDPS’17

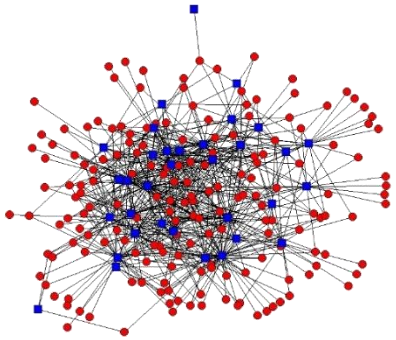
Graph Representations
Compressed Sparse Row (CSR)



M. Besta et al.: “SlimSell: A Vectorized Graph Representation for Breadth-First Search”, IPDPS’17

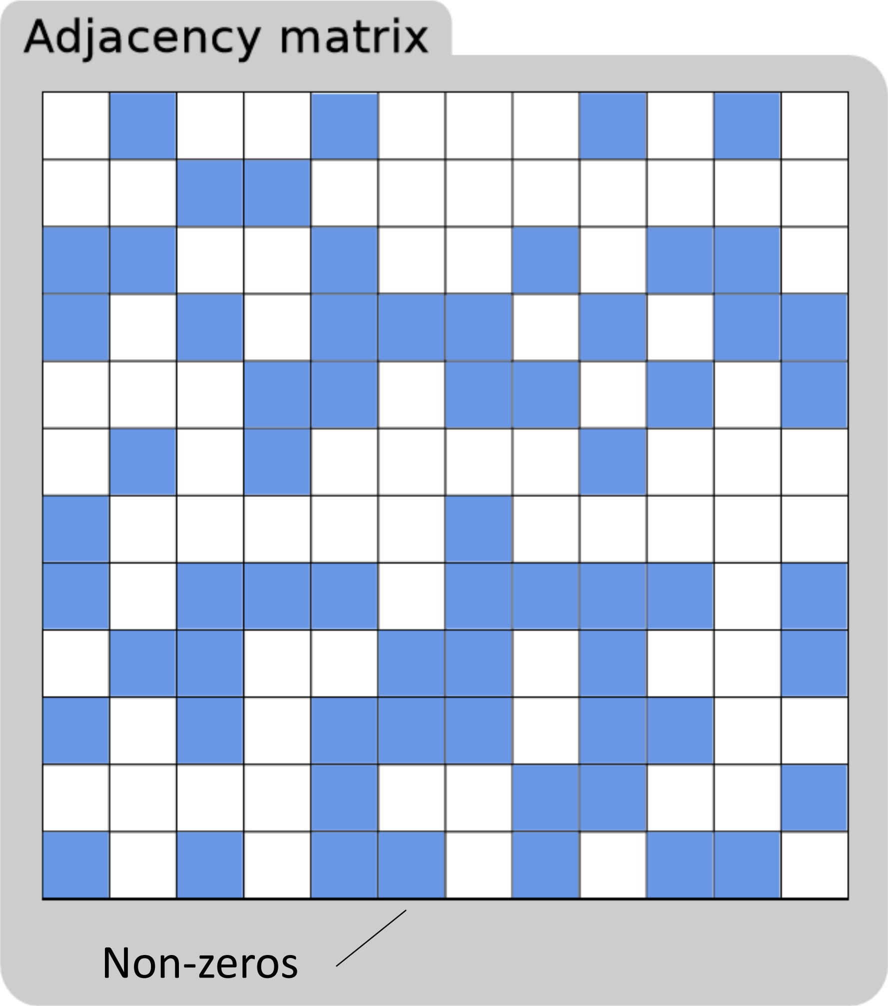
Graph Representations
Compressed Sparse Row (CSR)

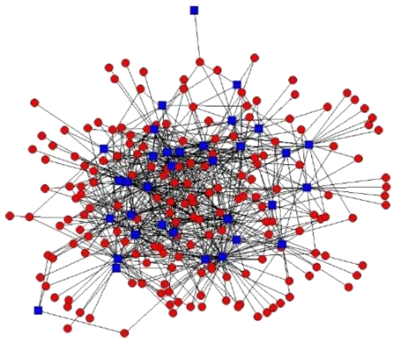




M. Besta et al.: “SlimSell: A Vectorized Graph Representation for Breadth-First Search”, IPDPS’17

Graph Representations
Compressed Sparse Row (CSR)

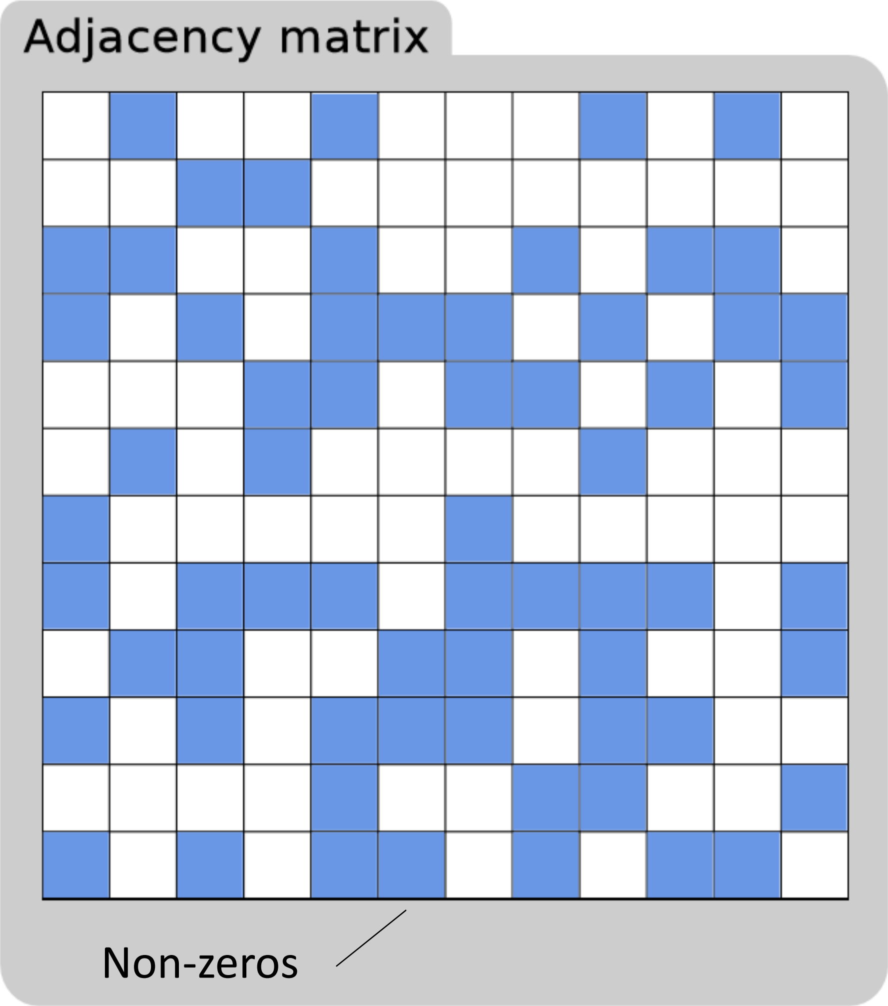


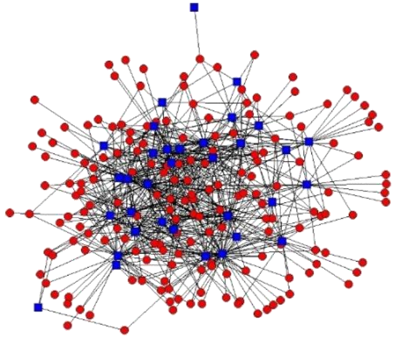


M. Besta et al.: “SlimSell: A Vectorized Graph Representation for Breadth-First Search”, IPDPS’17

Graph Representations

Compressed Sparse Row (CSR)





Non-zeros are stored in the *val* array

size: *2m* cells

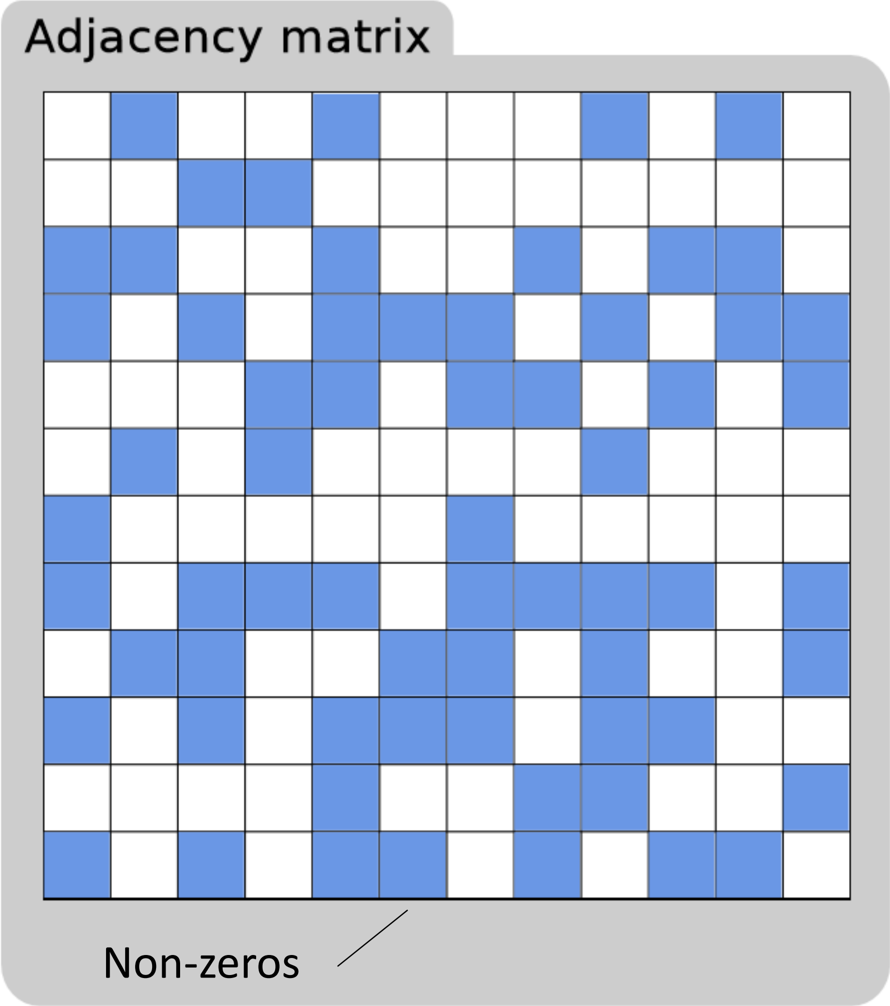
n: number of vertices

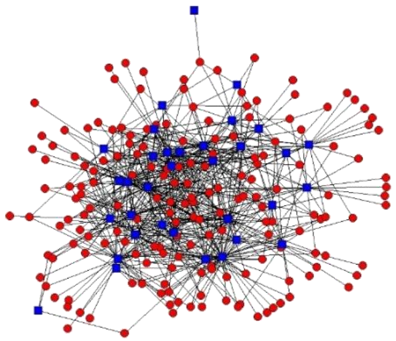
m: number of edges

M. Besta et al.: "SlimSell: A Vectorized Graph Representation for Breadth-First Search", IPDPS'17

Graph Representations

Compressed Sparse Row (CSR)





Non-zeros are stored in the *val* array

size: *2m* cells

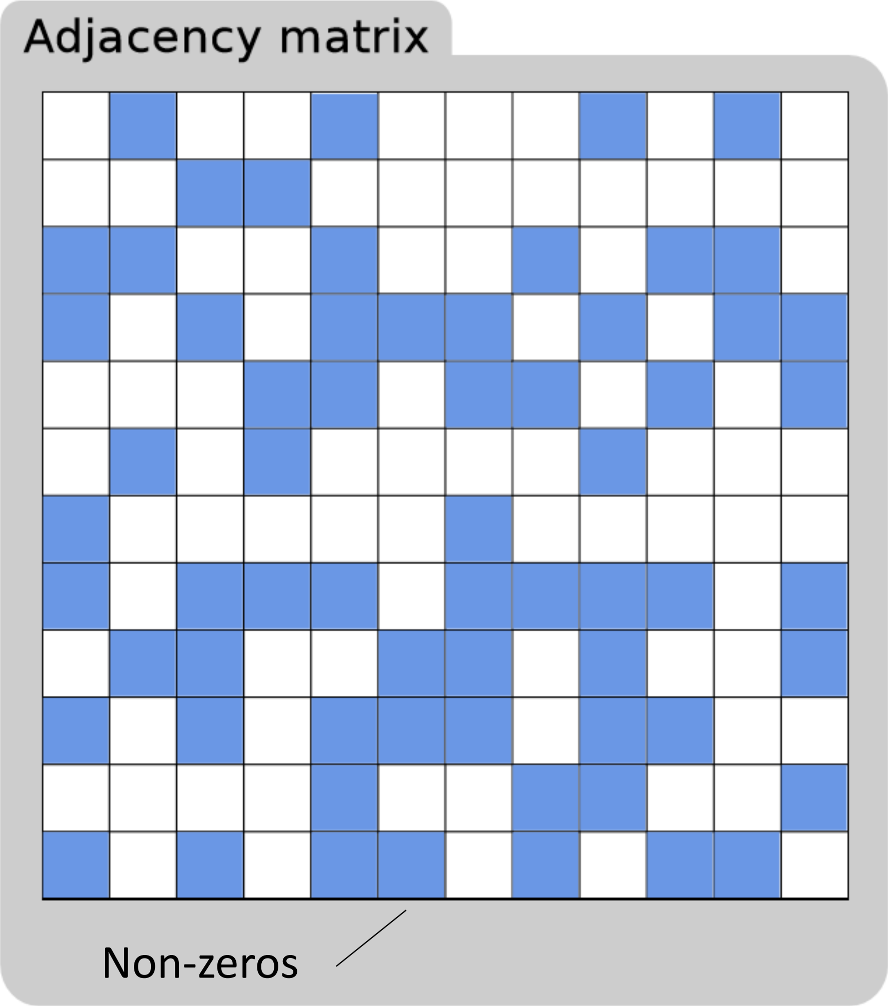
Column indices stored in the *col* array

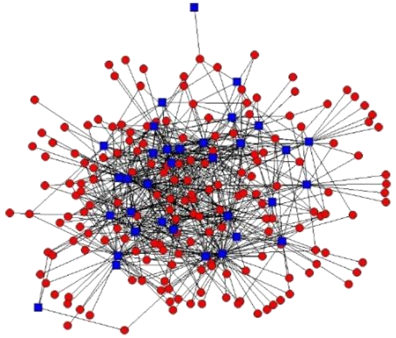
size: *2m* cells

n: number of vertices
m: number of edges

M. Besta et al.: “SlimSell: A Vectorized Graph Representation for Breadth-First Search”, IPDPS’17

Graph Representations
Compressed Sparse Row (CSR)

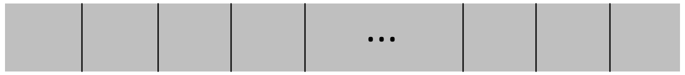




Non-zeros are stored in the *val* array — size: *2m* cells

Column indices stored in the *col* array — size: *2m* cells

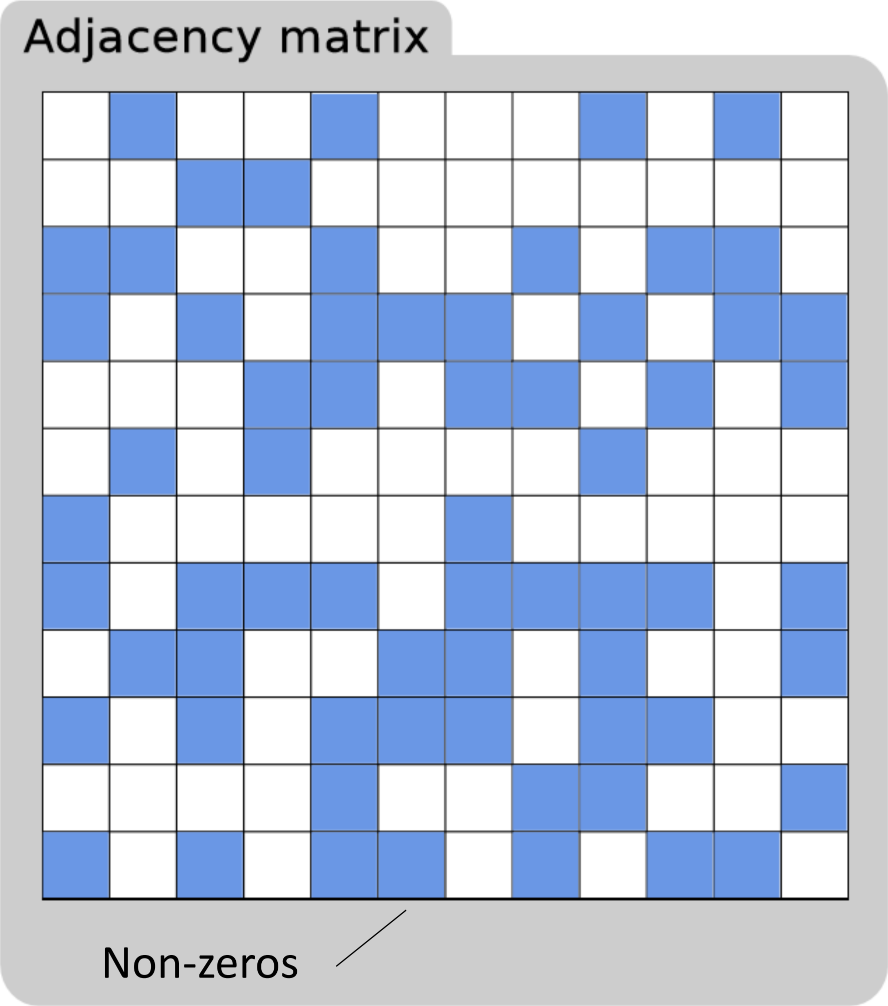
Row indices are stored in the *row* array — size: *n* cells

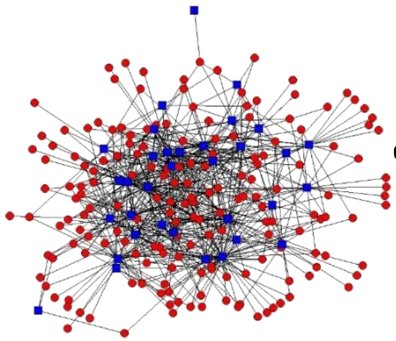


n: number of vertices
m: number of edges

M. Besta et al.: “SlimSell: A Vectorized Graph Representation for Breadth-First Search”, IPDPS’17

Graph Representations
Compressed Sparse Row (CSR)



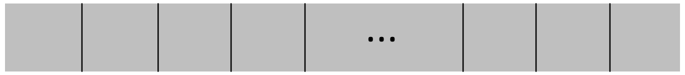


?

Non-zeros are stored in the *val* array — size: *2m* cells

Column indices stored in the *col* array — size: *2m* cells

Row indices are stored in the *row* array — size: *n* cells



n: number of vertices
m: number of edges

M. Besta et al.: “SlimSell: A Vectorized Graph Representation for Breadth-First Search”, IPDPS’17

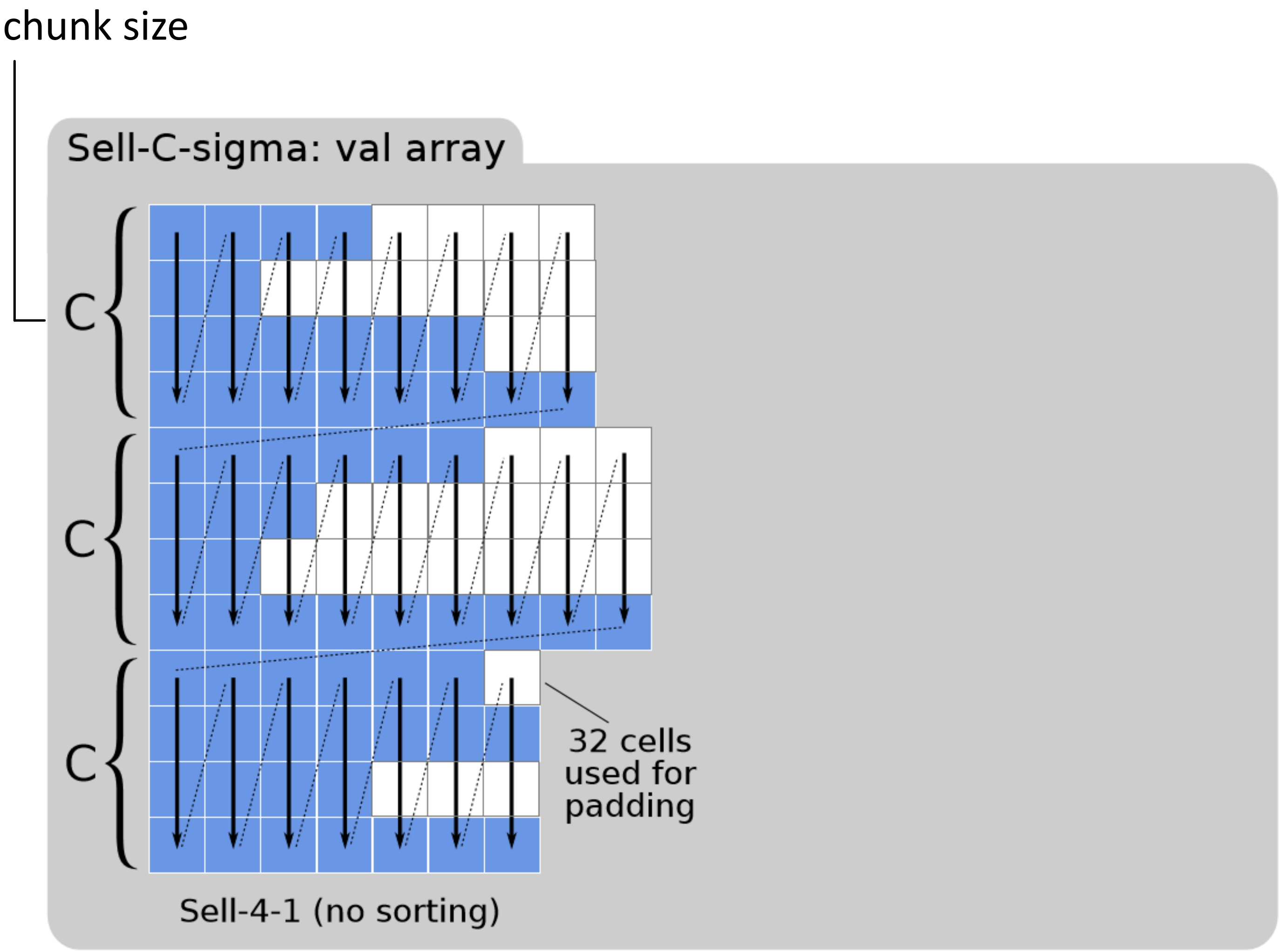
Graph Representations

Sell-C-Sigma

M. Besta et al.: “SlimSell: A Vectorized Graph Representation for Breadth-First Search”, IPDPS’17

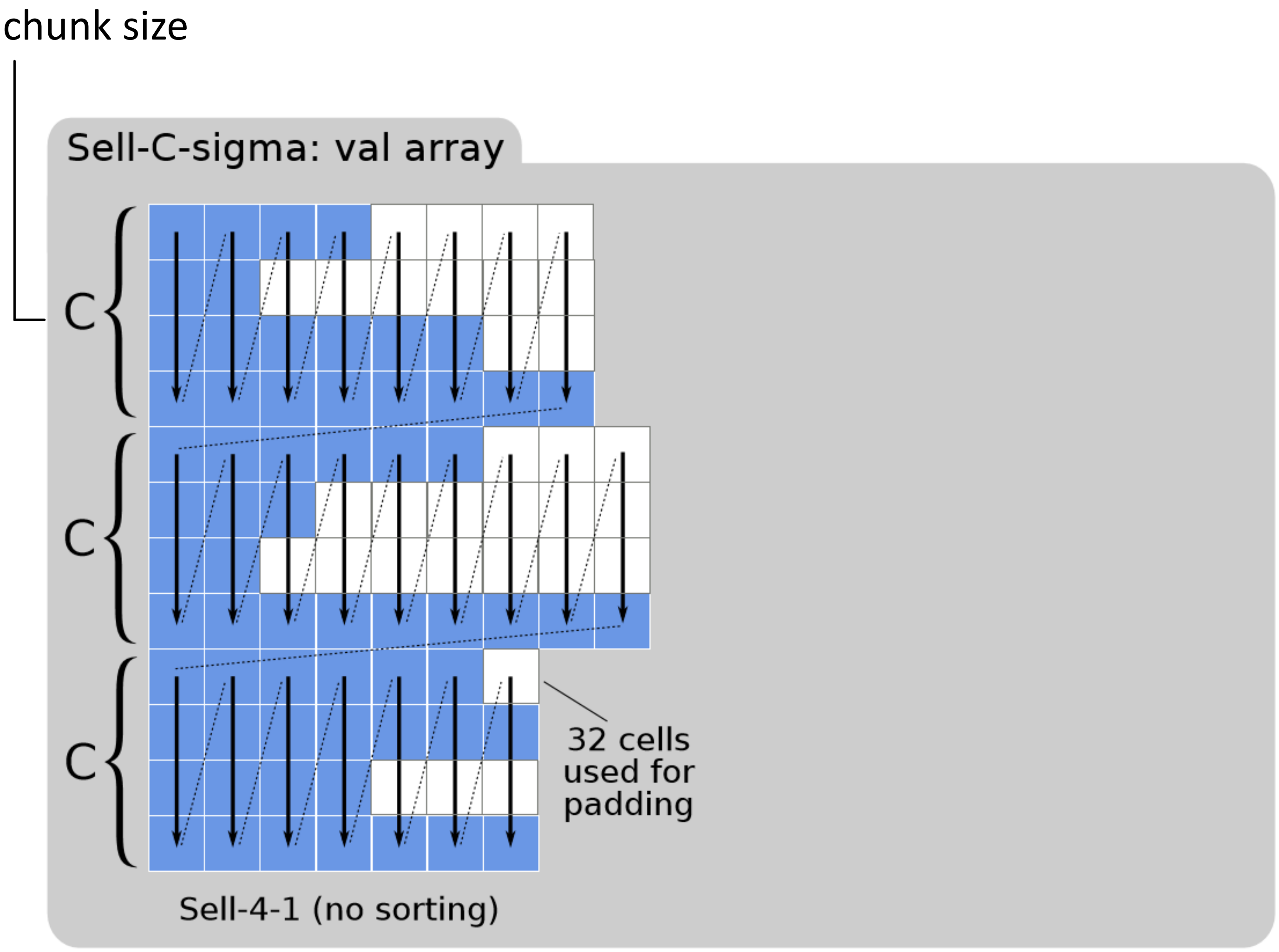
Graph Representations

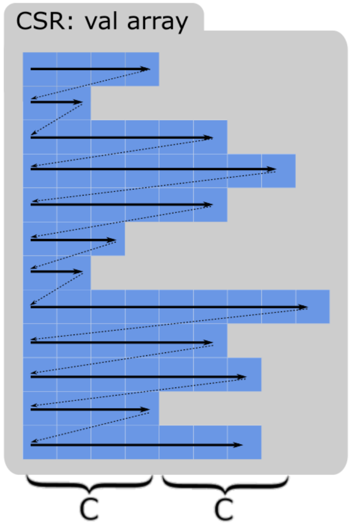
Sell-C-Sigma



M. Besta et al.: "SlimSell: A Vectorized Graph Representation for Breadth-First Search", IPDPS'17

GRAPH REPRESENTATIONS
SELL-C-SIGMA

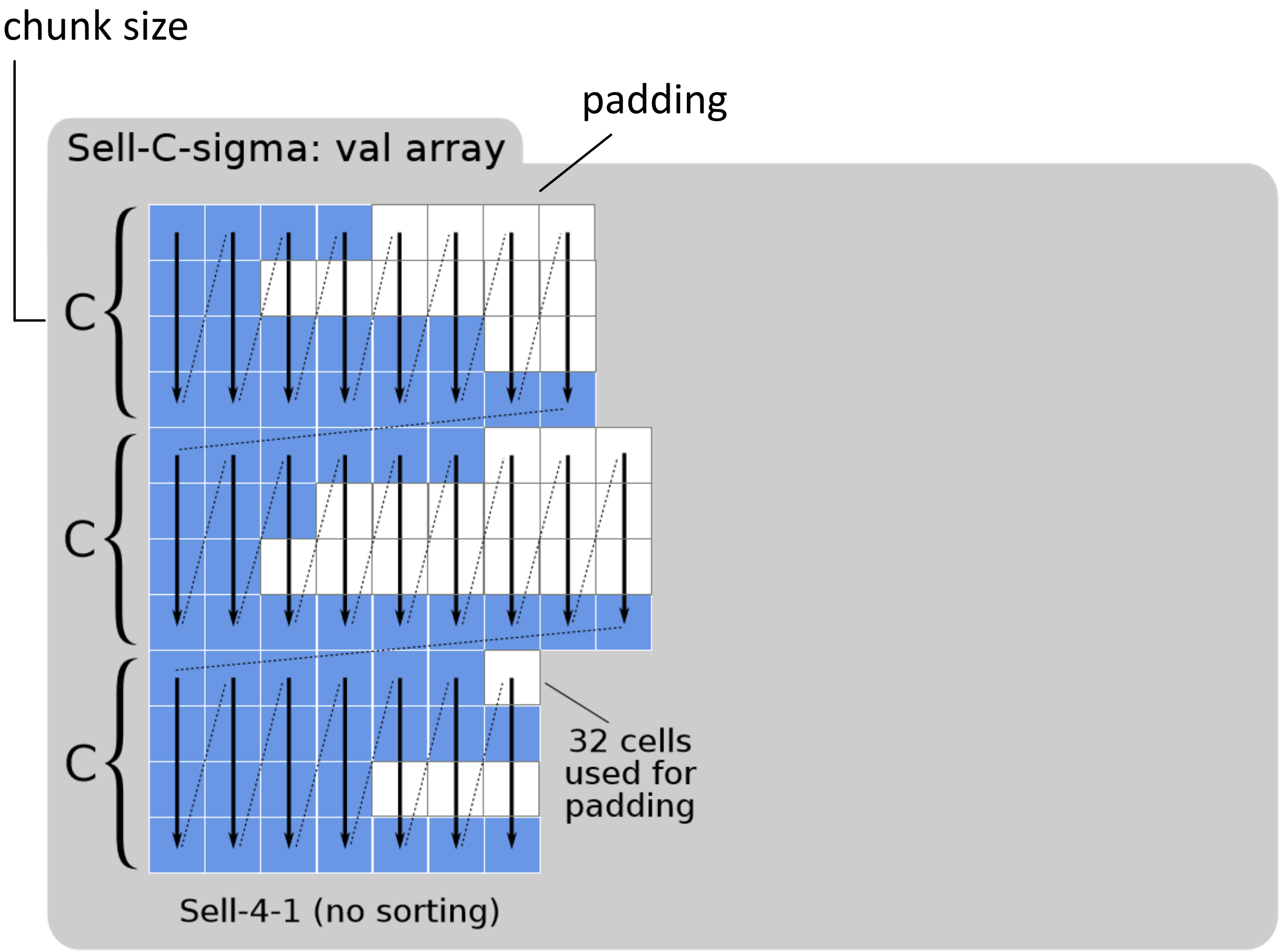


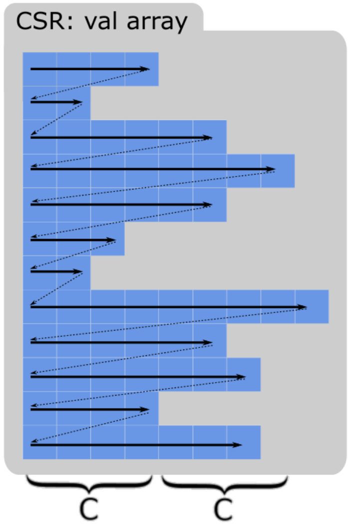


M. Besta et al.: "SlimSell: A Vectorized Graph Representation for Breadth-First Search", IPDPS'17

Graph Representations

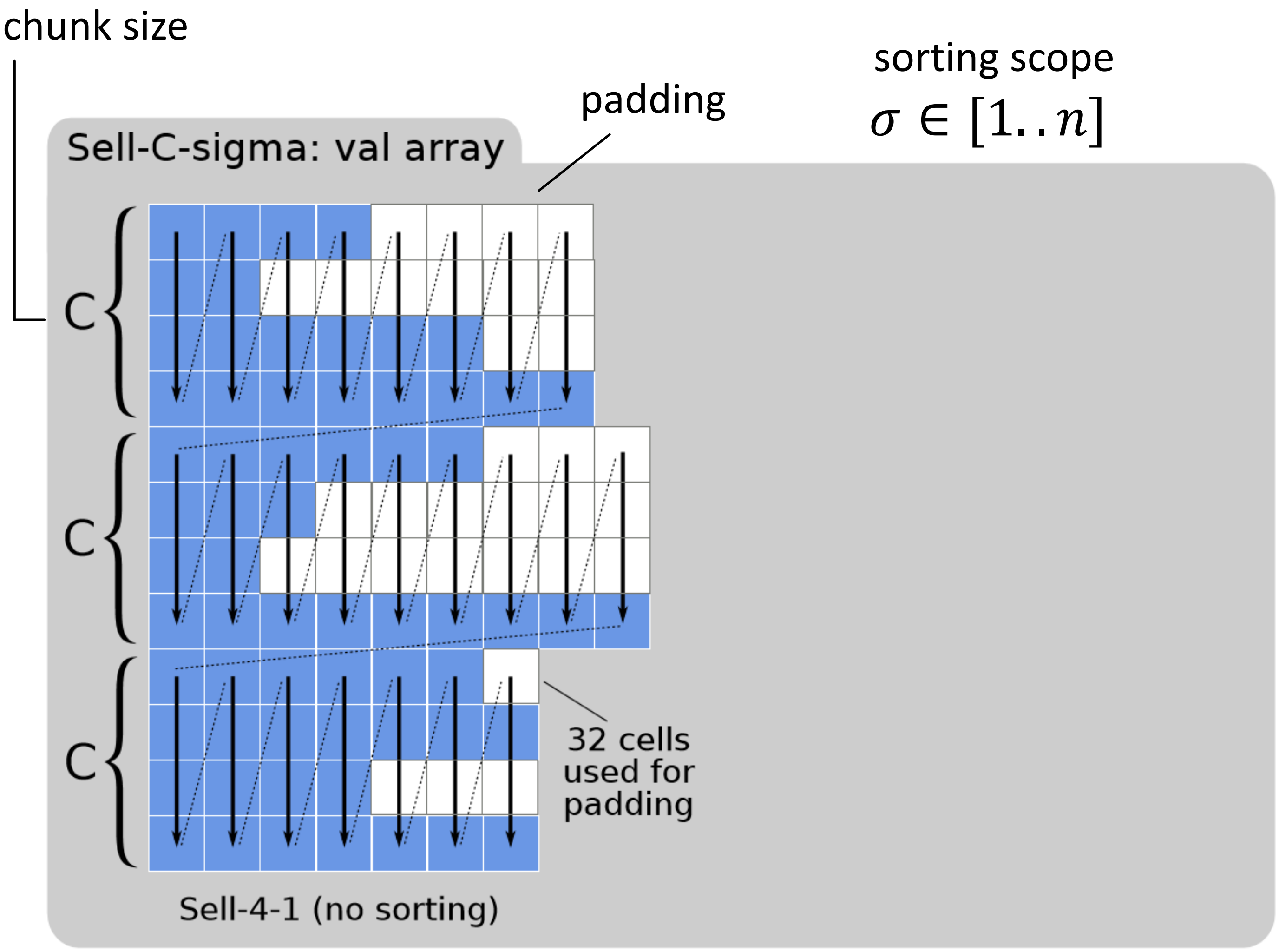
Sell-C-Sigma

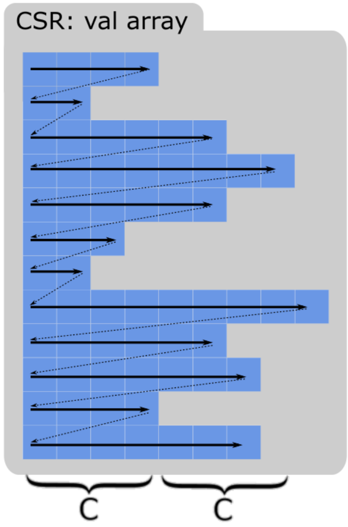




M. Besta et al.: "SlimSell: A Vectorized Graph Representation for Breadth-First Search", IPDPS'17

Graph Representations
Sell-C-Sigma

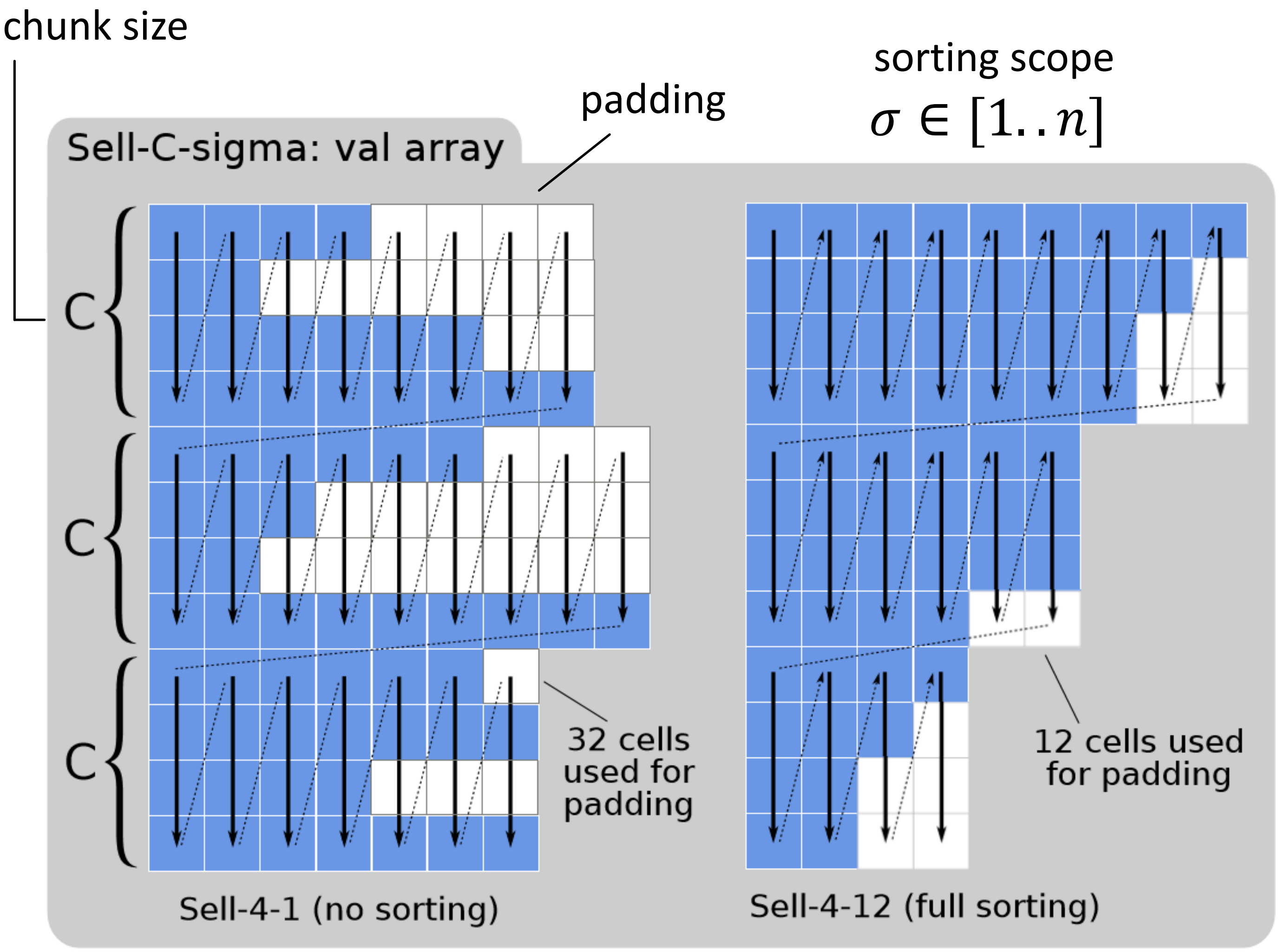


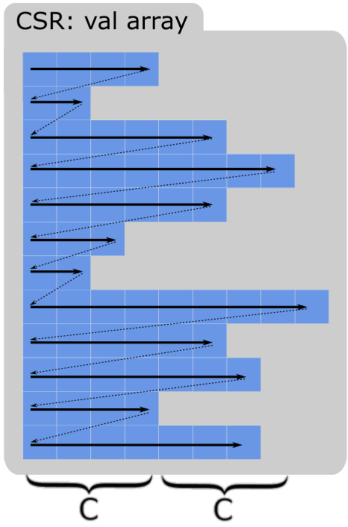


M. Besta et al.: "SlimSell: A Vectorized Graph Representation for Breadth-First Search", IPDPS'17

Graph Representations

Sell-C-Sigma

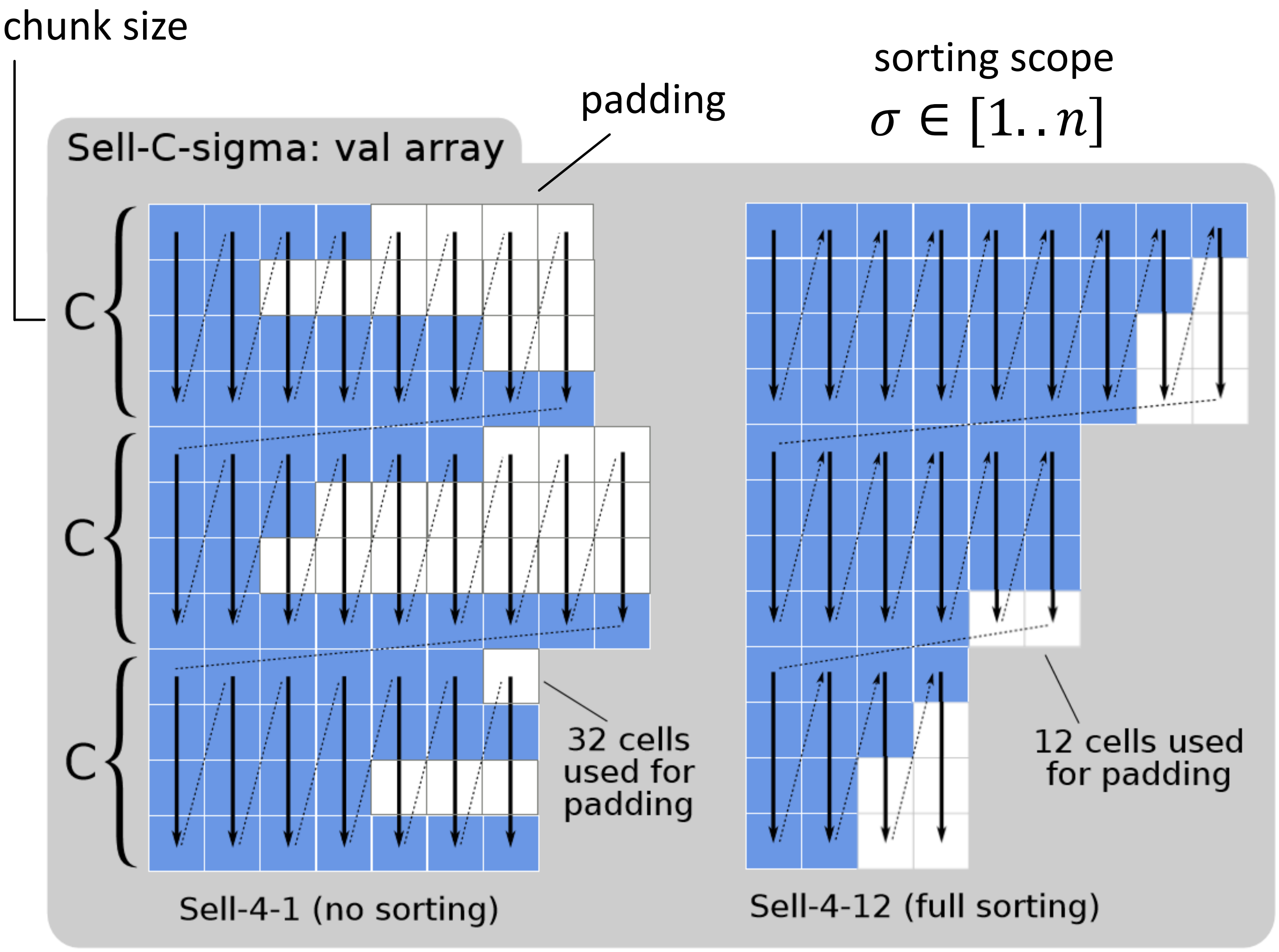


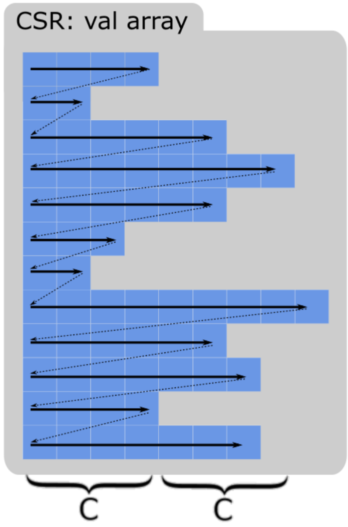


M. Besta et al.: "SlimSell: A Vectorized Graph Representation for Breadth-First Search", IPDPS'17

Graph Representations

Sell-C-Sigma



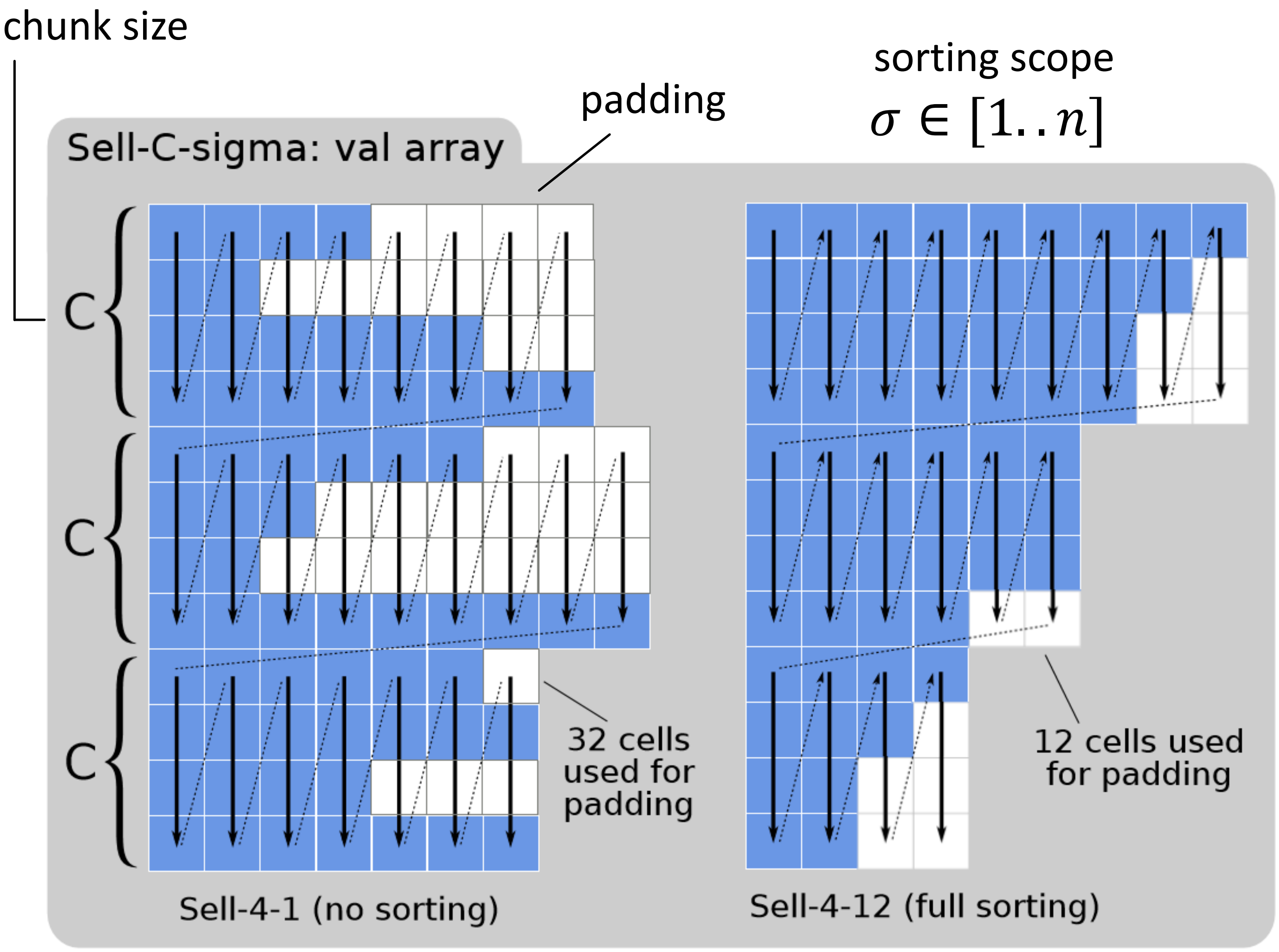


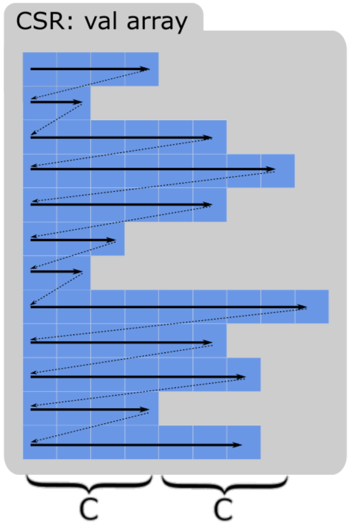
Reductions fast with SIMD operations

M. Besta et al.: “SlimSell: A Vectorized Graph Representation for Breadth-First Search”, IPDPS’17

Graph Representations

Sell-C-Sigma





Reductions fast with SIMD operations

Portable

M. Besta et al.: "SlimSell: A Vectorized Graph Representation for Breadth-First Search", IPDPS'17

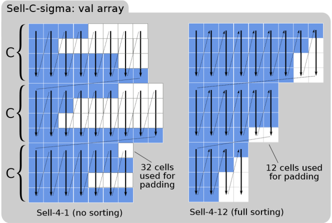
SELL-C-SIGMA + SEMIRINGS + (…) = SLIMSELL

FORMULATIONS

M. Besta et al.: "SlimSell: A Vectorized Graph Representation for Breadth-First Search", IPDPS'17

SELL-C-SIGMA + SEMIRINGS + (…) = SLIMSELL

FORMULATIONS



+

$(X, op_1, op_2, el_1, el_2)$

$(\mathbb{R} \cup \{\infty\}, min, +, \infty, 0)$

$(\mathbb{R}, +, \cdot, 0, 1)$

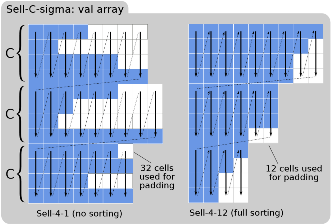
$(\{0,1\}, |, \&, 0, 1)$

$(\mathbb{R}, max, \cdot, -\infty, 1)$

M. Besta et al.: "SlimSell: A Vectorized Graph Representation for Breadth-First Search", IPDPS'17

Sell-C-Sigma + Semirings + (…) = SlimSell

Formulations



+

$(X, op_1, op_2, el_1, el_2)$

$(\mathbb{R} \cup \{\infty\}, min, +, \infty, 0)$

$(\mathbb{R}, +, \cdot, 0, 1)$

$(\{0,1\}, |, \&, 0, 1)$

$(\mathbb{R}, max, \cdot, -\infty, 1)$

TROPICAL SEMIRING

BOOLEAN SEMIRING

SEL-MAX SEMIRING

```
    // Compute x_k (versions differ based on the used semiring):
#ifdef USE_TROPICAL_SEMIRING
    x = MIN(ADD(rhs, vals), x);
#elif defined USE_BOOLEAN_SEMIRING
    x = OR(AND(rhs, vals), x);
#elif defined USE_SELMAX_SEMIRING
    x = MAX(MUL(rhs, vals), x);
#endif
    index += C;
  }
  // Now, derive f_k (versions differ based on the used semiring):
#ifdef USE_TROPICAL_SEMIRING
  STORE(&f_k[i*C], x); // Just a store.
#elif defined USE_BOOLEAN_SEMIRING
  // First, derive f_k using filtering.
  V g = LOAD(&g_{k-1}[i*C]); // Load the filter g_{k-1}.
  x = CMP(AND(x, g), [0,0,...0], NEQ); STORE(&x_k[i*C], x);

  // Second, update distances d; depth is the iteration number.
  V x_mask = x; x = MUL(x, [depth,...,depth]);
  x = BLEND(LOAD(&d[i*C]), x, x_mask); STORE(&d[i*C], x);

  // Third, update the filtering term.
  g = AND(NOT(x_mask), g); STORE(&g_k[i*C], g);
#elif defined USE_SELMAX_SEMIRING:
  // Update parents.
  V pars = LOAD(&p_{k-1}[i*C]); // Load the required part of p_{k-1}
  V pnz = CMP(pars, [0,0,...,0], NEQ);
  pars = BLEND([0,0,...,0], pars, pnz); STORE(&p_k[i*C], pars);

  // Set new x_k vector.
  V tmpnz = CMP(x, [0,0,...,0], NEQ);
  x = BLEND(x, &v[i*C], tmpnz); STORE(&x_k[i*C], x);
#endif
```

TROPICAL SEMIRING

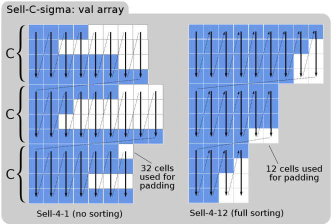
BOOLEAN SEMIRING

SEL-MAX SEMIRING

M. Besta et al.: "SlimSell: A Vectorized Graph Representation for Breadth-First Search", IPDPS'17

Sell-C-Sigma + Semirings + (…) = SlimSell

Formulations



+

$(X, op_1, op_2, el_1, el_2)$

$(\mathbb{R} \cup \{\infty\}, min, +, \infty, 0)$

$(\mathbb{R}, +, \cdot, 0, 1)$

$(\{0,1\}, |, \&, 0, 1)$

$(\mathbb{R}, max, \cdot, -\infty, 1)$

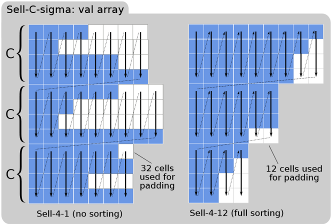
```
    // Compute x_k (versions differ based on the used semiring):
#ifdef USE_TROPICAL_SEMIRING                                   TROPICAL SEMIRING
    x = MIN(ADD(rhs, vals), x);
#elif defined USE_BOOLEAN_SEMIRING                             BOOLEAN SEMIRING
    x = OR(AND(rhs, vals), x);
#elif defined USE_SELMAX_SEMIRING                              SEL-MAX SEMIRING
    x = MAX(MUL(rhs, vals), x);
#endif
    index += C;
  }
  // Now, derive f_k (versions differ based on the used semiring):
#ifdef USE_TROPICAL_SEMIRING                                   TROPICAL SEMIRING
  STORE(&f_k[i*C], x); // Just a store.
#elif defined USE_BOOLEAN_SEMIRING                             BOOLEAN SEMIRING
  // First, derive f_k using filtering.
  V g = LOAD(&g_{k-1}[i*C]); // Load the filter g_{k-1}.
  x = CMP(AND(x, g), [0,0,...0], NEQ); STORE(&x_k[i*C], x);

  // Second, update distances d; depth is the iteration number.
  V x_mask = x; x = MUL(x, [depth,...,depth]);
  x = BLEND(LOAD(&d[i*C]), x, x_mask); STORE(&d[i*C], x);

  // Third, update the filtering term.
  g = AND(NOT(x_mask), g); STORE(&g_k[i*C], g);
#elif defined USE_SELMAX_SEMIRING:                             SEL-MAX SEMIRING
  // Update parents.
  V pars = LOAD(&p_{k-1}[i*C]); // Load the required part of p_{k-1}
  V pnz = CMP(pars, [0,0,...,0], NEQ);
  pars = BLEND([0,0,...,0], pars, pnz); STORE(&p_k[i*C], pars);

  // Set new x_k vector.
  V tmpnz = CMP(x, [0,0,...,0], NEQ);
  x = BLEND(x, &v[i*C], tmpnz); STORE(&x_k[i*C], x);
#endif
```

! Detailed formulations are in the paper ☺

M. Besta et al.: "SlimSell: A Vectorized Graph Representation for Breadth-First Search", IPDPS'17

SELL-C-SIGMA + SEMIRINGS + (…) = SLIMSELL

FORMULATIONS



+

$(X, op_1, op_2, el_1, el_2)$

$(\mathbb{R} \cup \{\infty\}, min, +, \infty, 0)$

$(\mathbb{R}, +, \cdot, 0, 1)$

$(\{0,1\}, |, \&, 0, 1)$

$(\mathbb{R}, max, \cdot, -\infty, 1)$

```
    // Compute x_k (versions differ based on the used semiring):
#ifdef USE_TROPICAL_SEMIRING                                   TROPICAL SEMIRING
    x = MIN(ADD(rhs, vals), x);
#elif defined USE_BOOLEAN_SEMIRING                             BOOLEAN SEMIRING
    x = OR(AND(rhs, vals), x);
#elif defined USE_SELMAX_SEMIRING                              SEL-MAX SEMIRING
    x = MAX(MUL(rhs, vals), x);
#endif
    index += C;
  }
  // Now, derive f_k (versions differ based on the used semiring):
#ifdef USE_TROPICAL_SEMIRING                                   TROPICAL SEMIRING
  STORE(&f_k[i*C], x); // Just a store.
#elif defined USE_BOOLEAN_SEMIRING                             BOOLEAN SEMIRING
  // First, derive f_k using filtering.
  V g = LOAD(&g_{k-1}[i*C]); // Load the filter g_{k-1}.
  x = CMP(AND(x, g), [0,0,...0], NEQ); STORE(&x_k[i*C], x);

  // Second, update distances d; depth is the iteration number.
  V x_mask = x; x = MUL(x, [depth,...,depth]);
  x = BLEND(LOAD(&d[i*C]), x, x_mask); STORE(&d[i*C], x);

  // Third, update the filtering term.
  g = AND(NOT(x_mask), g); STORE(&g_k[i*C], g);
#elif defined USE_SELMAX_SEMIRING:                             SEL-MAX SEMIRING
  // Update parents.
  V pars = LOAD(&p_{k-1}[i*C]); // Load the required part of p_{k-1}
  V pnz = CMP(pars, [0,0,...,0], NEQ);
  pars = BLEND([0,0,...,0], pars, pnz); STORE(&p_k[i*C], pars);

  // Set new x_k vector.
  V tmpnz = CMP(x, [0,0,...,0], NEQ);
  x = BLEND(x, &v[i*C], tmpnz); STORE(&x_k[i*C], x);
#endif
```

! What vector operations are required for each semiring when using Sell-C-sigma

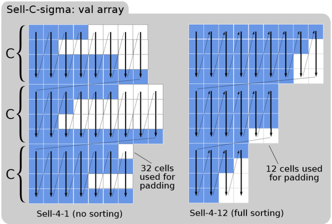
! Detailed formulations are in the paper ☺

M. Besta et al.: "SlimSell: A Vectorized Graph Representation for Breadth-First Search", IPDPS'17

SELL-C-SIGMA + SEMIRINGS + (…) = SLIMSELL

FORMULATIONS

```
    // Compute x_k (versions differ based on the used semiring):
#ifdef USE_TROPICAL_SEMIRING                                   TROPICAL SEMIRING
    x = MIN(ADD(rhs, vals), x);
#elif defined USE_BOOLEAN_SEMIRING                             BOOLEAN SEMIRING
    x = OR(AND(rhs, vals), x);
#elif defined USE_SELMAX_SEMIRING                              SEL-MAX SEMIRING
    x = MAX(MUL(rhs, vals), x);
#endif
    index += C;
  }
  // Now, derive f_k (versions differ based on the used semiring):
#ifdef USE_TROPICAL_SEMIRING                                   TROPICAL SEMIRING
  STORE(&f_k[i*C], x); // Just a store.
#elif defined USE_BOOLEAN_SEMIRING                             BOOLEAN SEMIRING
  // First, derive f_k using filtering.
  V g = LOAD(&g_{k-1}[i*C]); // Load the filter g_{k-1}.
  x = CMP(AND(x, g), [0,0,...0], NEQ); STORE(&x_k[i*C], x);

  // Second, update distances d; depth is the iteration number.
  V x_mask = x; x = MUL(x, [depth,...,depth]);
  x = BLEND(LOAD(&d[i*C]), x, x_mask); STORE(&d[i*C], x);

  // Third, update the filtering term.
  g = AND(NOT(x_mask), g); STORE(&g_k[i*C], g);
#elif defined USE_SELMAX_SEMIRING:                             SEL-MAX SEMIRING
  // Update parents.
  V pars = LOAD(&p_{k-1}[i*C]); // Load the required part of p_{k-1}
  V pnz = CMP(pars, [0,0,...,0], NEQ);
  pars = BLEND([0,0,...,0], pars, pnz); STORE(&p_k[i*C], pars);

  // Set new x_k vector.
  V tmpnz = CMP(x, [0,0,...,0], NEQ);
  x = BLEND(x, &v[i*C], tmpnz); STORE(&x_k[i*C], x);
#endif
```



+

$(X, op_1, op_2, el_1, el_2)$

$(\mathbb{R} \cup \{\infty\}, min, +, \infty, 0)$

$(\mathbb{R}, +, \cdot, 0, 1)$

$(\{0,1\}, |, \&, 0, 1)$

$(\mathbb{R}, max, \cdot, -\infty, 1)$

What vector operations are required for each semiring when using Sell-C-sigma

Detailed formulations are in the paper ☺

M. Besta et al.: “SlimSell: A Vectorized Graph Representation for Breadth-First Search”, IPDPS’17

Performance Analysis

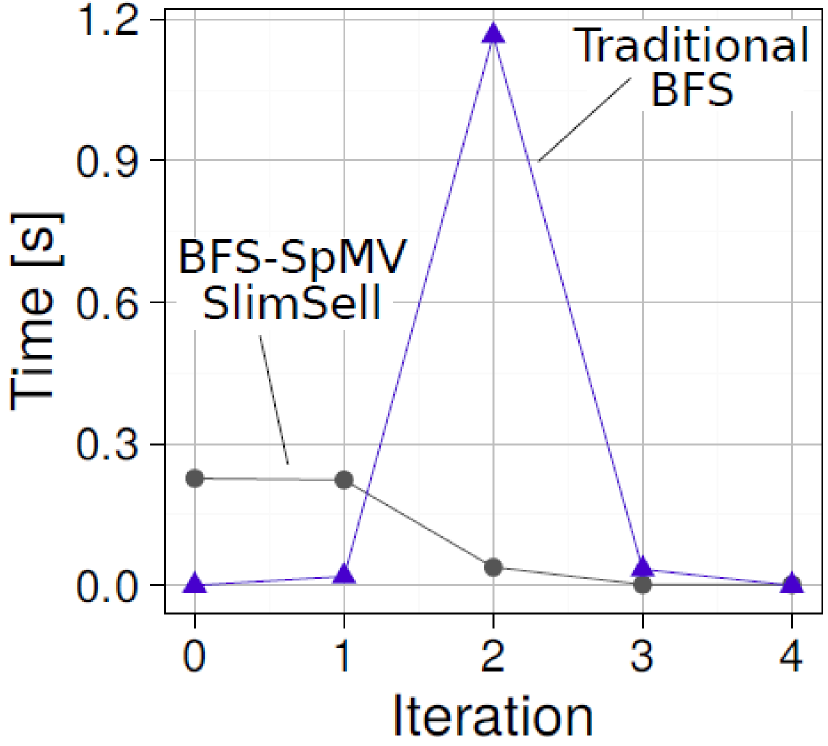
Comparison to Graph500

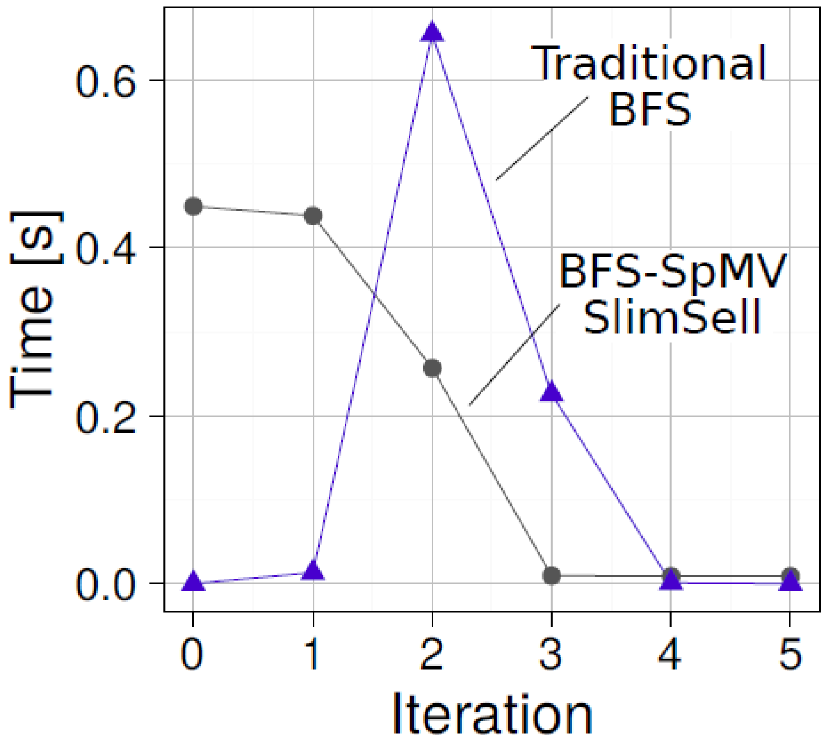
Kronecker power-law graphs

Intel KNL, $C = 16$

$\log \sigma \in \{20,21,22\}$

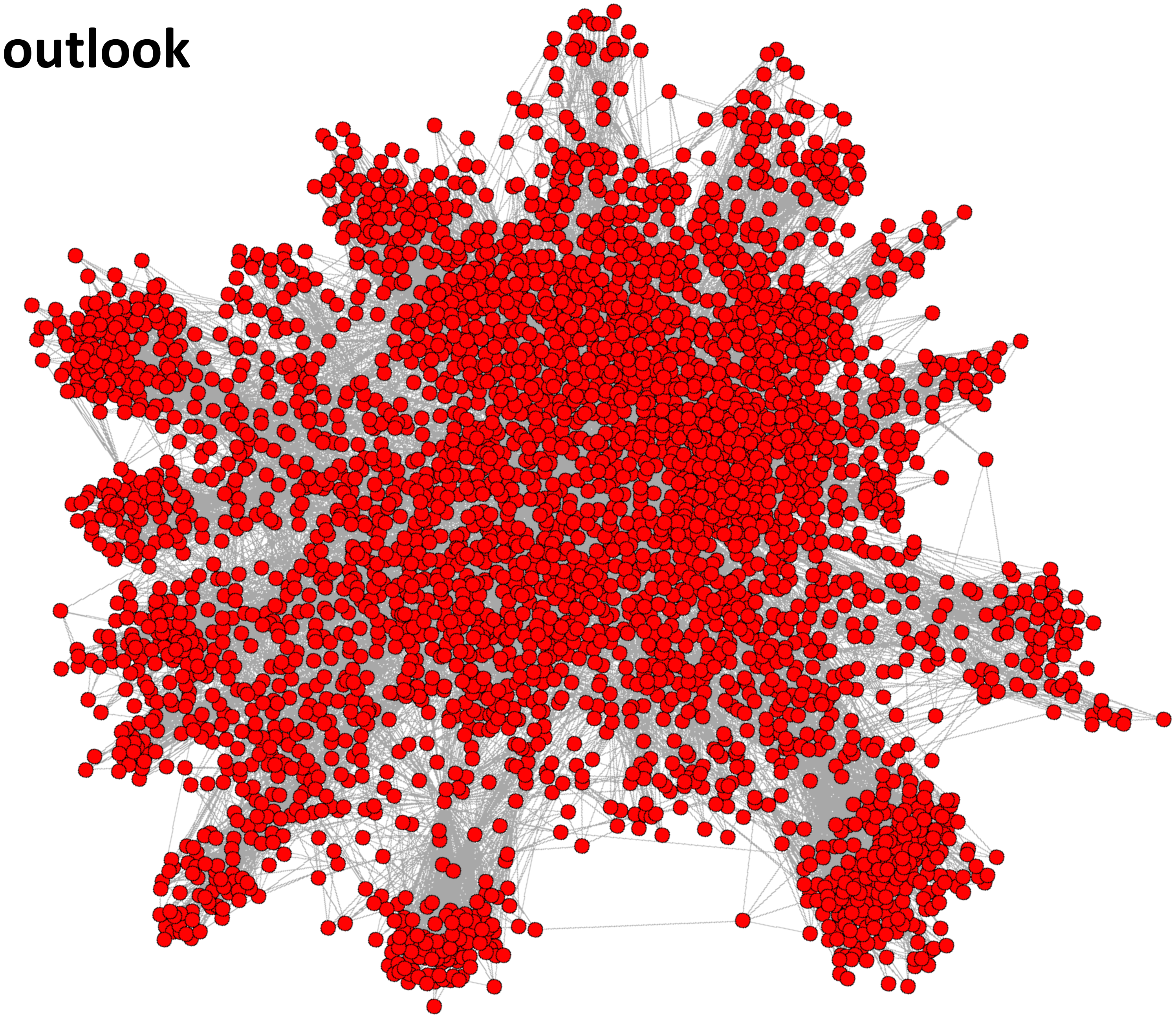
Dynamic scheduling



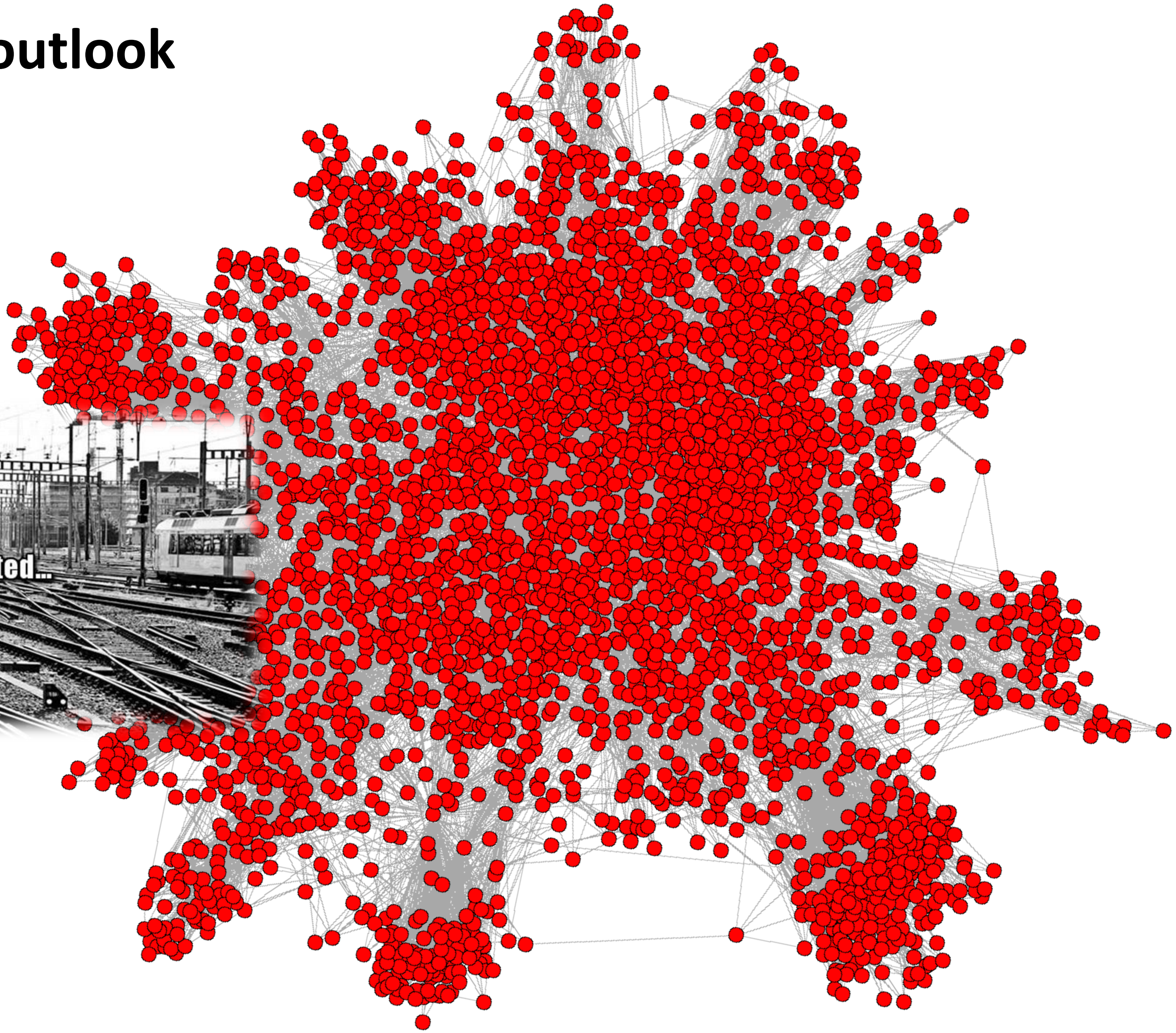


M. Besta et al.: "SlimSell: A Vectorized Graph Representation for Breadth-First Search", IPDPS'17

Summary and outlook



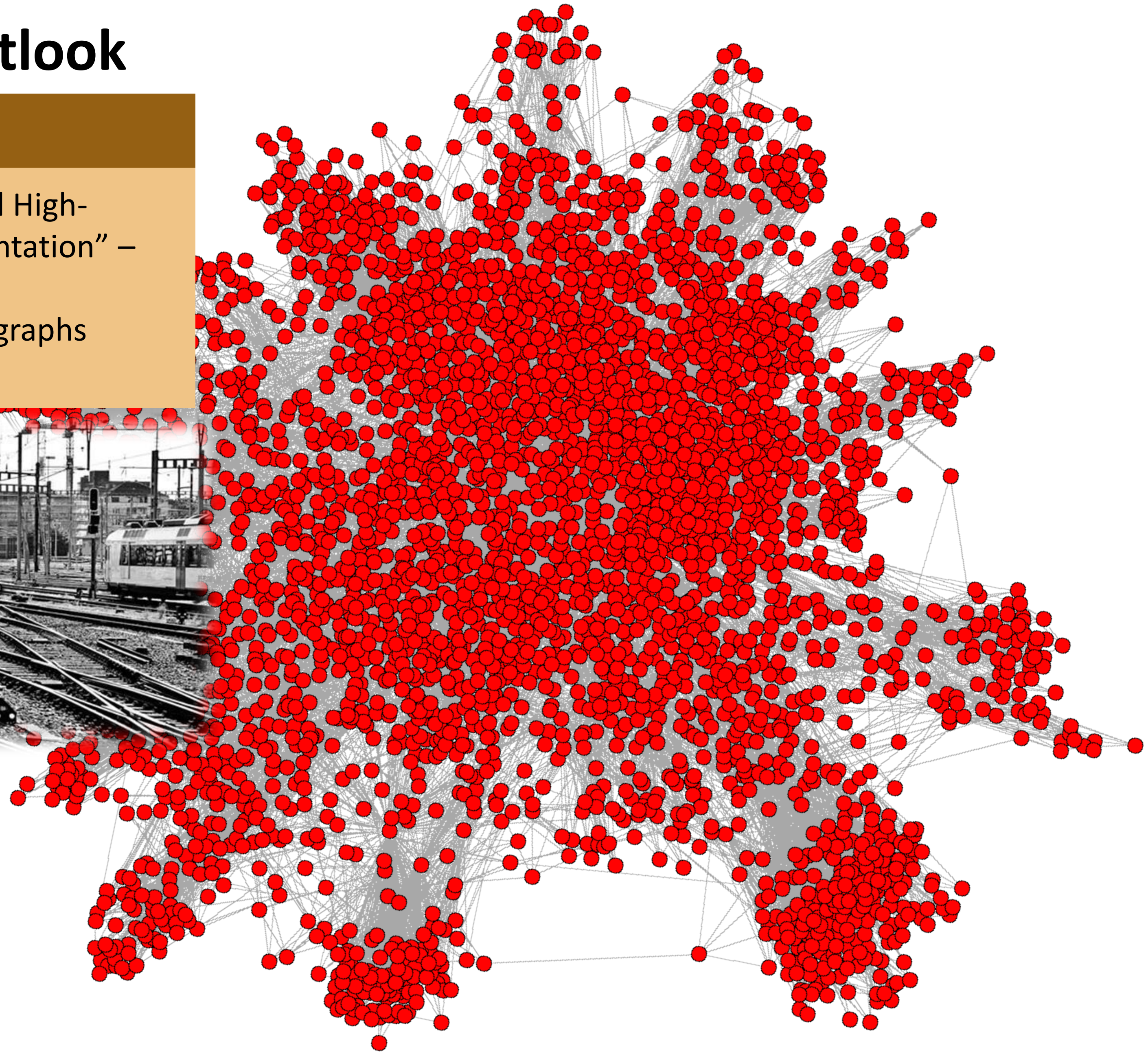
Summary and outlook



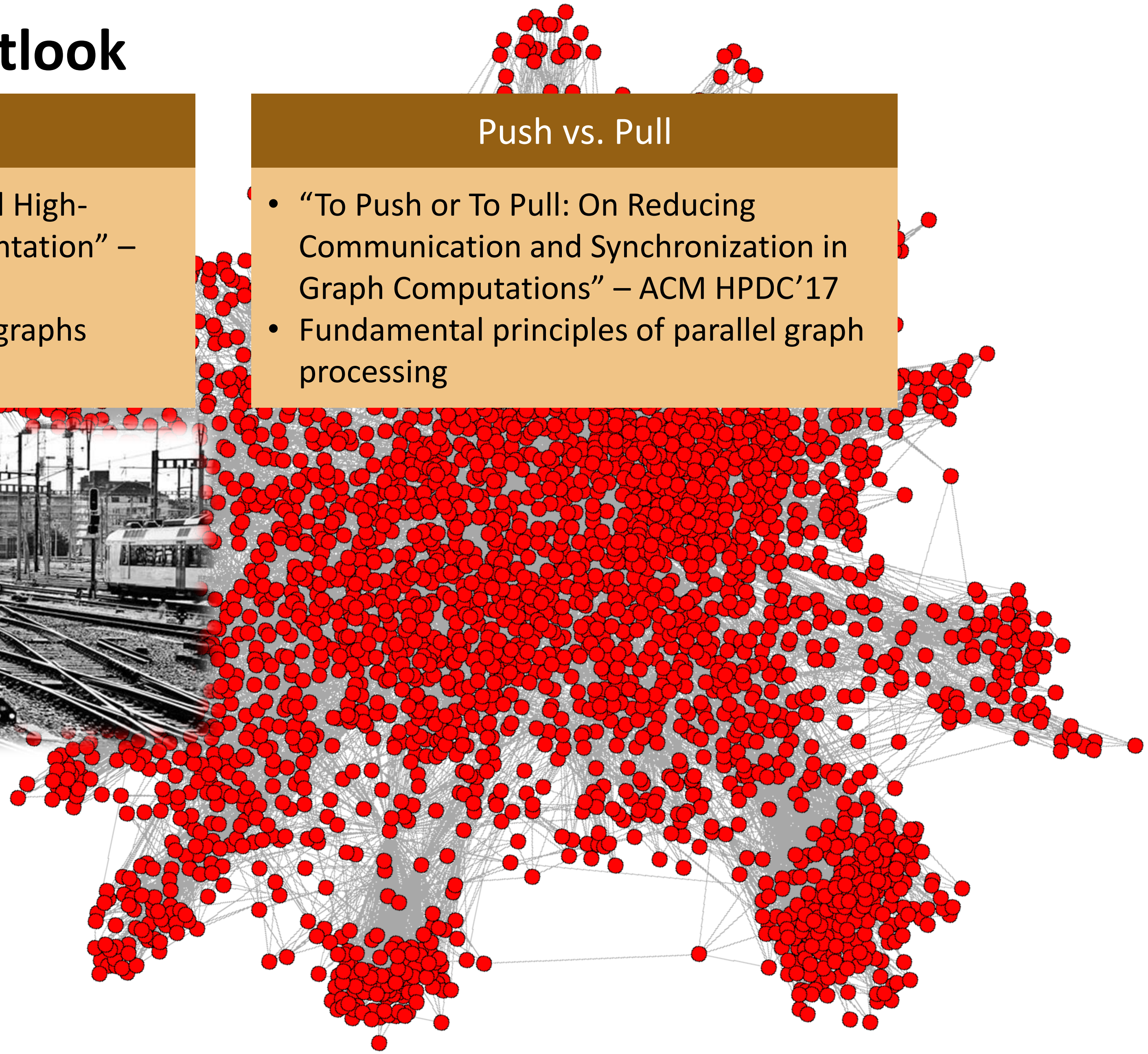
Summary and outlook

Log(Graph)

- “Log(Graph): A Near-Optimal High-Performance Graph Representation” – ACM PACT’18
- Minimal storage bounds for graphs during processing



Summary and outlook



Log(Graph)

- “Log(Graph): A Near-Optimal High-Performance Graph Representation” – ACM PACT’18
- Minimal storage bounds for graphs during processing

Push vs. Pull

- “To Push or To Pull: On Reducing Communication and Synchronization in Graph Computations” – ACM HPDC’17
- Fundamental principles of parallel graph processing

Summary and outlook

Log(Graph)

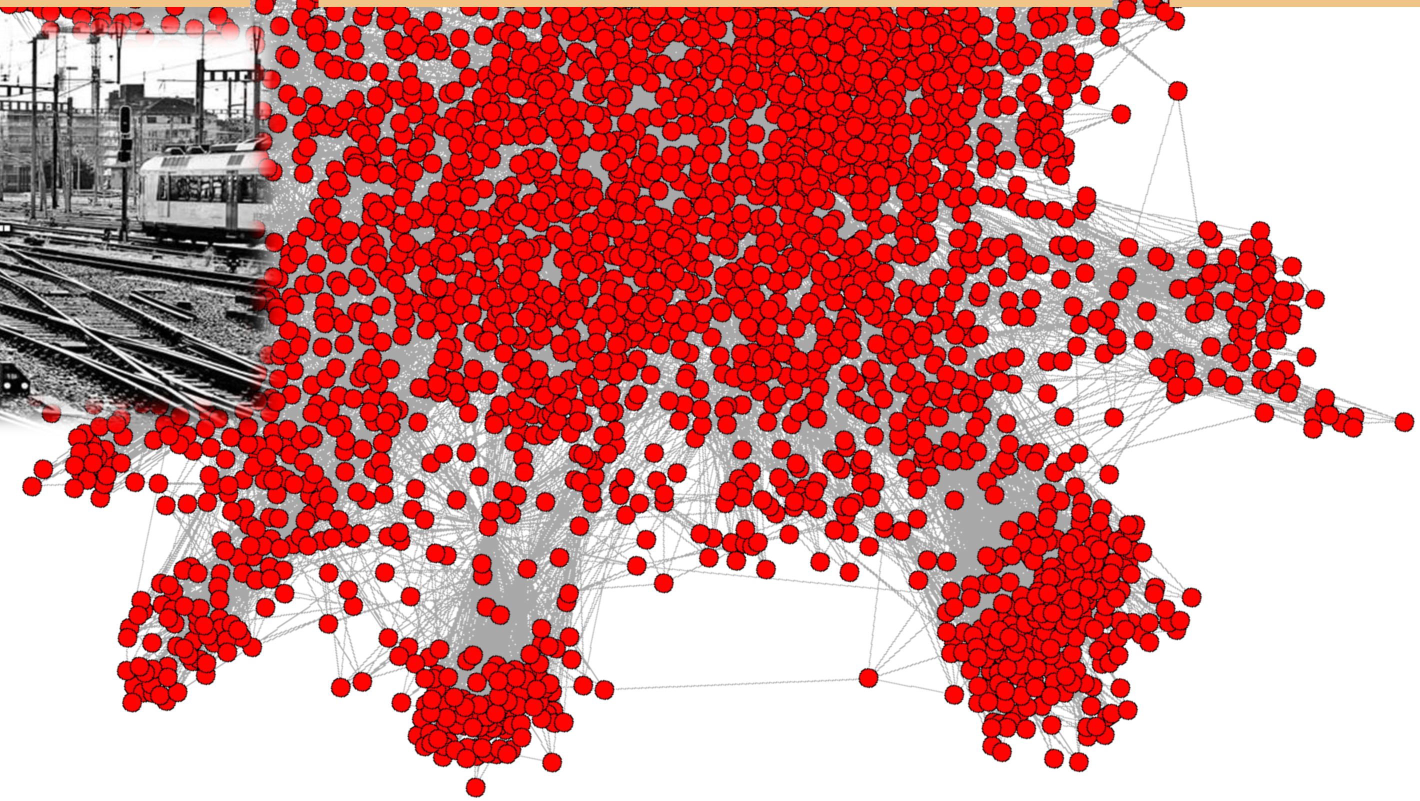
- “Log(Graph): A Near-Optimal High-Performance Graph Representation” – ACM PACT’18
- Minimal storage bounds for graphs during processing

Push vs. Pull

- “To Push or To Pull: On Reducing Communication and Synchronization in Graph Computations” – ACM HPDC’17
- Fundamental principles of parallel graph processing

Vectorization of Graph Computations

- “SlimSell: A Vectorized Graph Representation for Breadth-First Search” – IEEE IPDPS’17
- Vectorization schemes for parallel graph processing



Summary and outlook

Log(Graph)

- “Log(Graph): A Near-Optimal High-Performance Graph Representation” – ACM PACT’18
- Minimal storage bounds for graphs during processing

Push vs. Pull

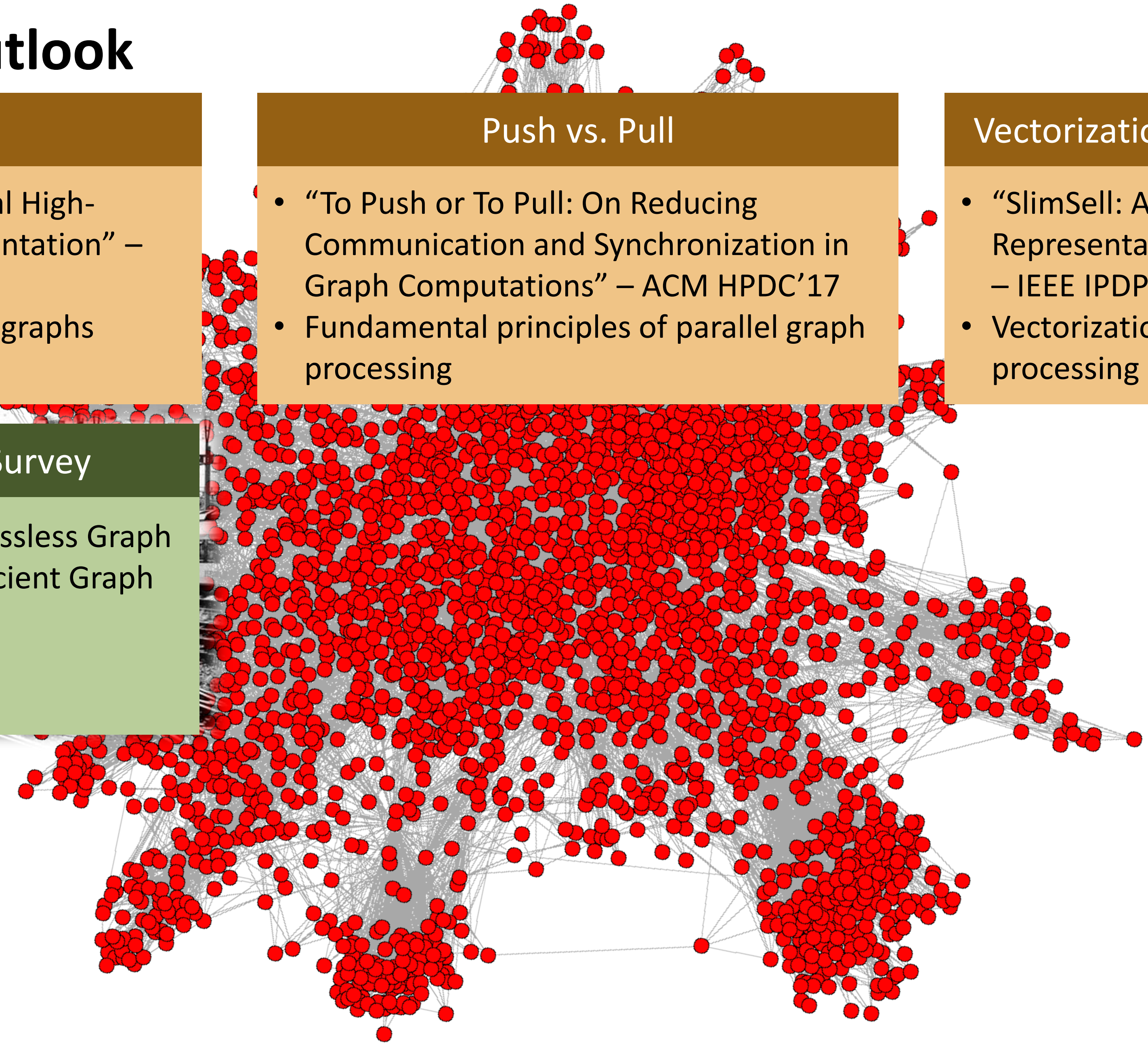
- “To Push or To Pull: On Reducing Communication and Synchronization in Graph Computations” – ACM HPDC’17
- Fundamental principles of parallel graph processing

Vectorization of Graph Computations

- “SlimSell: A Vectorized Graph Representation for Breadth-First Search” – IEEE IPDPS’17
- Vectorization schemes for parallel graph processing

Graph Compression Survey

- “Survey and Taxonomy of Lossless Graph Compression and Space-Efficient Graph Representations” – arXiv
- Comprehensive overview
  - 54 pages, 465 references



Summary and outlook

Log(Graph)

- "Log(Graph): A Near-Optimal High-Performance Graph Representation" – ACM PACT'18
- Minimal storage bounds for graphs during processing

Push vs. Pull

- "To Push or To Pull: On Reducing Communication and Synchronization in Graph Computations" – ACM HPDC'17
- Fundamental principles of parallel graph processing

Vectorization of Graph Computations

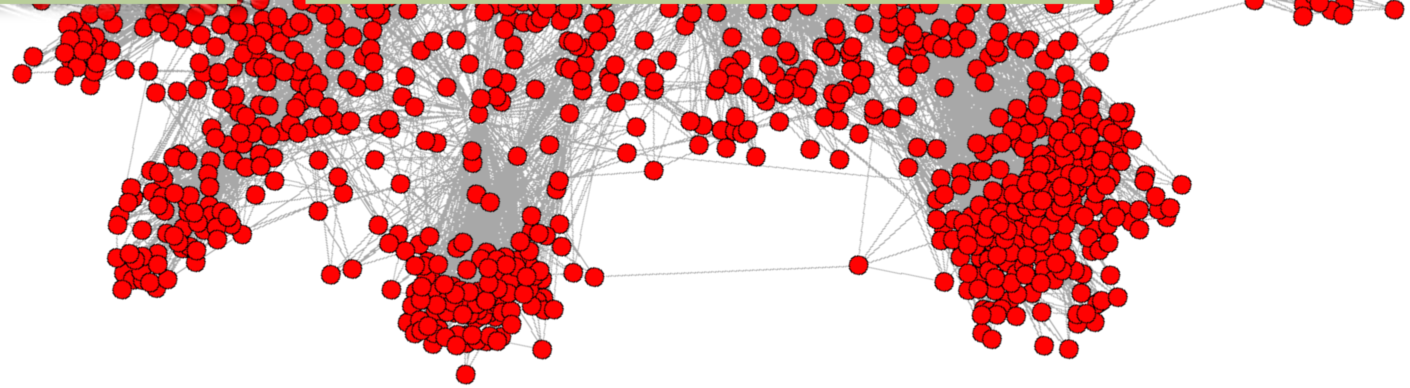
- "SlimSell: A Vectorized Graph Representation for Breadth-First Search" – IEEE IPDPS'17
- Vectorization schemes for parallel graph processing

Graph Compression Survey

- "Survey and Taxonomy of Lossless Graph Compression and Space-Efficient Graph Representations" – arXiv
- Comprehensive overview
  - 54 pages, 465 references

Comm-avoiding Graph Processing

- "Communication-Avoiding Parallel Minimum Cuts and Connected Components" – ACM PPoPP'18
- Uses randomization to achieve O(1) global alltoall steps



Summary and outlook

Log(Graph)

- "Log(Graph): A Near-Optimal High-Performance Graph Representation" – ACM PACT'18
- Minimal storage bounds for graphs during processing

Push vs. Pull

- "To Push or To Pull: On Reducing Communication and Synchronization in Graph Computations" – ACM HPDC'17
- Fundamental principles of parallel graph processing

Vectorization of Graph Computations

- "SlimSell: A Vectorized Graph Representation for Breadth-First Search" – IEEE IPDPS'17
- Vectorization schemes for parallel graph processing

Graph Compression Survey

- "Survey and Taxonomy of Lossless Graph Compression and Space-Efficient Graph Representations" – arXiv
- Comprehensive overview
  - 54 pages, 465 references

Comm-avoiding Graph Processing

- "Communication-Avoiding Parallel Minimum Cuts and Connected Components" – ACM PPoPP'18
- Uses randomization to achieve O(1) global alltoall steps

Algebraic Betweenness Centrality

- "Scaling Betweenness Centrality using Communication-Efficient Sparse Matrix Multiplication" – ACM SC'17
- More on the algebraic view – complex example, large-scale sparse matrices

Summary and outlook

Log(Graph)

- "Log(Graph): A Near-Optimal High-Performance Graph Representation" – ACM PACT'18
- Minimal storage bounds for graphs during processing

Push vs. Pull

- "To Push or To Pull: On Reducing Communication and Synchronization in Graph Computations" – ACM HPDC'17
- Fundamental principles of parallel graph processing

Vectorization of Graph Computations

- "SlimSell: A Vectorized Graph Representation for Breadth-First Search" – IEEE IPDPS'17
- Vectorization schemes for parallel graph processing

Graph Compression Survey

- "Survey and Taxonomy of Lossless Graph Compression and Space-Efficient Graph Representations" – arXiv
- Comprehensive overview
  - 54 pages, 465 references

Comm-avoiding Graph Processing

- "Communication-Avoiding Parallel Minimum Cuts and Connected Components" – ACM PPoPP'18
- Uses randomization to achieve O(1) global alltoall steps

Algebraic Betweenness Centrality

- "Scaling Betweenness Centrality using Communication-Efficient Sparse Matrix Multiplication" – ACM SC'17
- More on the algebraic view – complex example, large-scale sparse matrices

Streaming Graphs on FPGA

- "Substream-Centric Maximum Matchings on FPGA" – FPGA'19
- New paradigm for parallelizing across substreams
  - Integrates with pipelining in HW/FPGA
  - Blueprint for efficient processing in HW

Summary and outlook

Log(Graph)
- "Log(Graph): A Near-Optimal High-Performance Graph Representation" – ACM PACT'18
- Minimal storage bounds for graphs during processing

Push vs. Pull
- "To Push or To Pull: On Reducing Communication and Synchronization in Graph Computations" – ACM HPDC'17
- Fundamental principles of parallel graph processing

Vectorization of Graph Computations
- "SlimSell: A Vectorized Graph Representation for Breadth-First Search" – IEEE IPDPS'17
- Vectorization schemes for parallel graph processing

Graph Compression Survey
- "Survey and Taxonomy of Lossless Graph Compression and Space-Efficient Graph Representations" – arXiv
- Comprehensive overview
  - 54 pages, 465 references

Comm-avoiding Graph Processing
- "Communication-Avoiding Parallel Minimum Cuts and Connected Components" – ACM PPoPP'18
- Uses randomization to achieve O(1) global alltoall steps

Algebraic Betweenness Centrality
- "Scaling Betweenness Centrality using Communication-Efficient Sparse Matrix Multiplication" – ACM SC'17
- More on the algebraic view – complex example, large-scale sparse matrices

Streaming Graphs on FPGA
- "Substream-Centric Maximum Matchings on FPGA" – FPGA'19
- New paradigm for parallelizing across substreams
  - Integrates with pipelining in HW/FPGA
  - Blueprint for efficient processing in HW

Streaming Graphs Survey
- "Survey and Taxonomy of Models and Algorithms for Streaming Graph Processing" – arXiv
- Overview of streaming algorithms, approximations, research gaps
  - Way forward for FPGA?

Summary and outlook

Log(Graph)

- "Log(Graph): A Near-Optimal High-Performance Graph Representation" – ACM PACT'18
- Minimal storage bounds for graphs during processing

Push vs. Pull

- "To Push or To Pull: On Reducing Communication and Synchronization in Graph Computations" – ACM HPDC'17
- Fundamental principles of parallel graph processing

Vectorization of Graph Computations

- "SlimSell: A Vectorized Graph Representation for Breadth-First Search" – IEEE IPDPS'17
- Vectorization schemes for parallel graph processing

Graph Compression Survey

- "Survey and Taxonomy of Lossless Graph Compression and Space-Efficient Graph Representations" – arXiv
- Comprehensive overview
  - 54 pages, 465 references

Comm-avoiding Graph Processing

- "Communication-Avoiding Parallel Minimum Cuts and Connected Components" – ACM PPoPP'18
- Uses randomization to achieve O(1) global alltoall steps

Algebraic Betweenness Centrality

- "Scaling Betweenness Centrality using Communication-Efficient Sparse Matrix Multiplication" – ACM SC'17
- More on the algebraic view – complex example, large-scale sparse matrices

Streaming Graphs on FPGA

- "Substream-Centric Maximum Matchings on FPGA" – FPGA'19
- New paradigm for parallelizing across substreams
  - Integrates with pipelining in HW/FPGA
  - Blueprint for efficient processing in HW

Streaming Graphs Survey

- "Survey and Taxonomy of Models and Algorithms for Streaming Graph Processing" – arXiv
- Overview of streaming algorithms, approximations, research gaps
  - Way forward for FPGA?

Graphs on FPGA Survey

- "Graph Processing on FPGAs: Taxonomy, Survey, Challenges" – arXiv
- Relatively young field of graph processing on FPGAs / in hardware
  - Identify research opportunities

Summary and outlook

Log(Graph)

- "Log(Graph): A Near-Optimal High-Performance Graph Representation" – ACM PACT'18
- Minimal storage bounds for graphs during processing

Push vs. Pull

- "To Push or To Pull: On Reducing Communication and Synchronization in Graph Computations" – ACM HPDC'17
- Fundamental principles of parallel graph processing

Vectorization of Graph Computations

- "SlimSell: A Vectorized Graph Representation for Breadth-First Search" – IEEE IPDPS'17
- Vectorization schemes for parallel graph processing

Graph Compression Surve

- "Survey and Taxonomy of Lossles Compression and Space-Efficient Representations" – arXiv
- Comprehensive overview
  - 54 pages, 465 references

ebraic Betweenness Centrality

- ling Betweenness Centrality using munication-Efficient Sparse Matrix tiplication" – ACM SC'17
- e on the algebraic view – complex nple, large-scale sparse matrices

Streaming Graphs on FPG

- "Substream-Centric Maximum Matchings on FPGA" – FPGA'19
- New paradigm for parallelizing across substreams
  - Integrates with pipelining in HW/FPGA
  - Blueprint for efficient processing in HW

Algorithms for Streaming Graph Processing" – arXiv
- Overview of streaming algorithms, approximations, research gaps
  - Way forward for FPGA?

Graphs on FPGA Survey

- "Graph Processing on FPGAs: Taxonomy, Survey, Challenges" – arXiv
- Relatively young field of graph processing on FPGAs / in hardware
  - Identify research opportunities

A big challenges ahead: develop a framework to integrate all techniques!

SPCL's approach: *stateful dataflow graphs*

We're always hiring excellent PhD students and postdocs at SPCL/ETH at spcl.inf.ethz.ch/Jobs