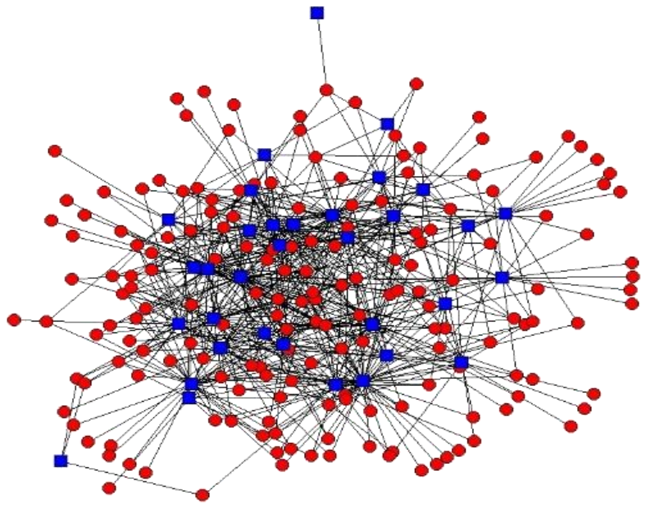
Evaluating the Cost of Atomic Operations on Modern Architectures

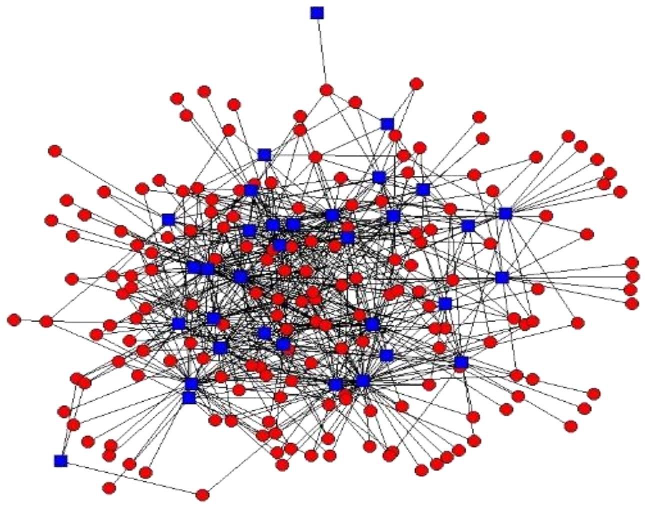
MACIEJ BESTA, HERMANN SCHWEIZER, TORSTEN HOEFLER

LARGE-SCALE IRREGULAR GRAPH PROCESSING

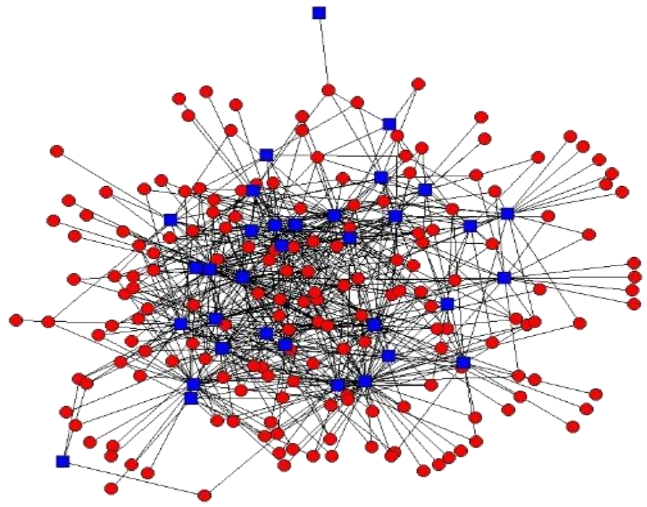
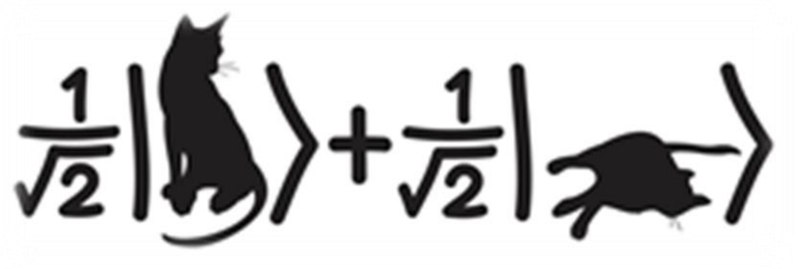
LARGE-SCALE IRREGULAR GRAPH PROCESSING



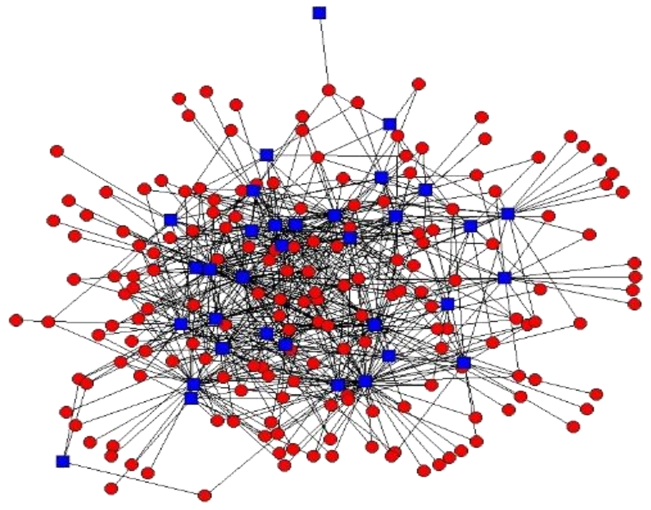
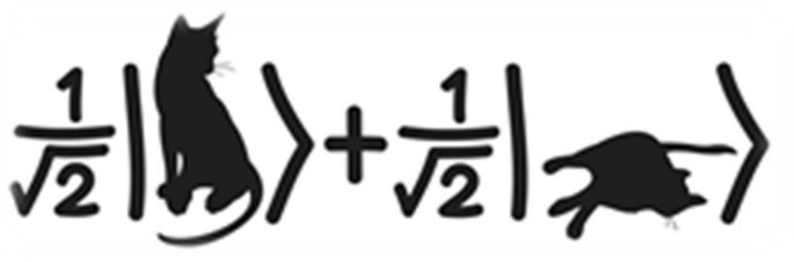
Large-Scale Irregular Graph Processing



Large-Scale Irregular Graph Processing

LARGE-SCALE IRREGULAR GRAPH PROCESSING

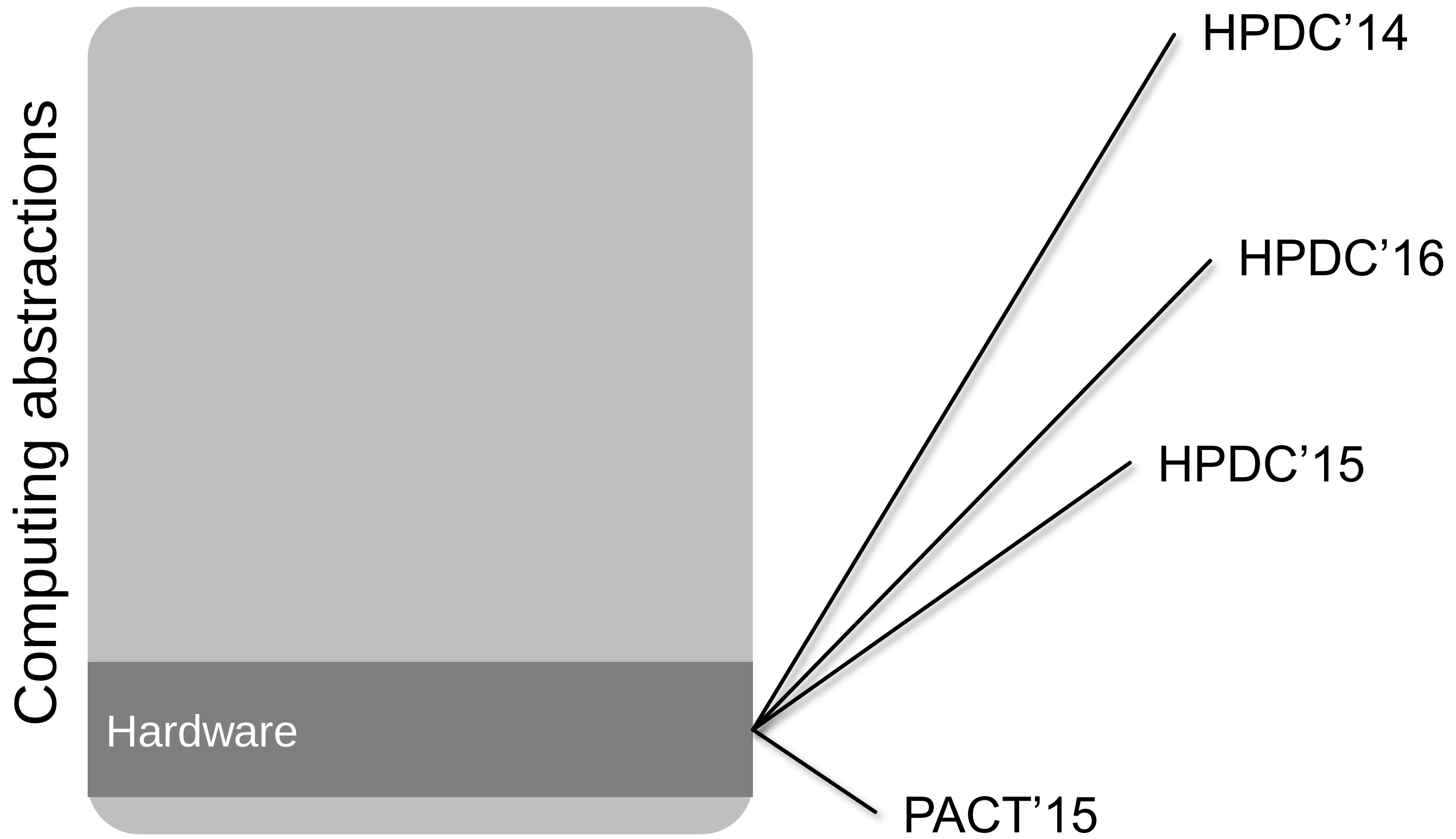
[1] A. Lumsdaine et al. Challenges in Parallel Graph Processing. Parallel Processing Letters. 2007.

A Brief Summary of Research I Do in HPC

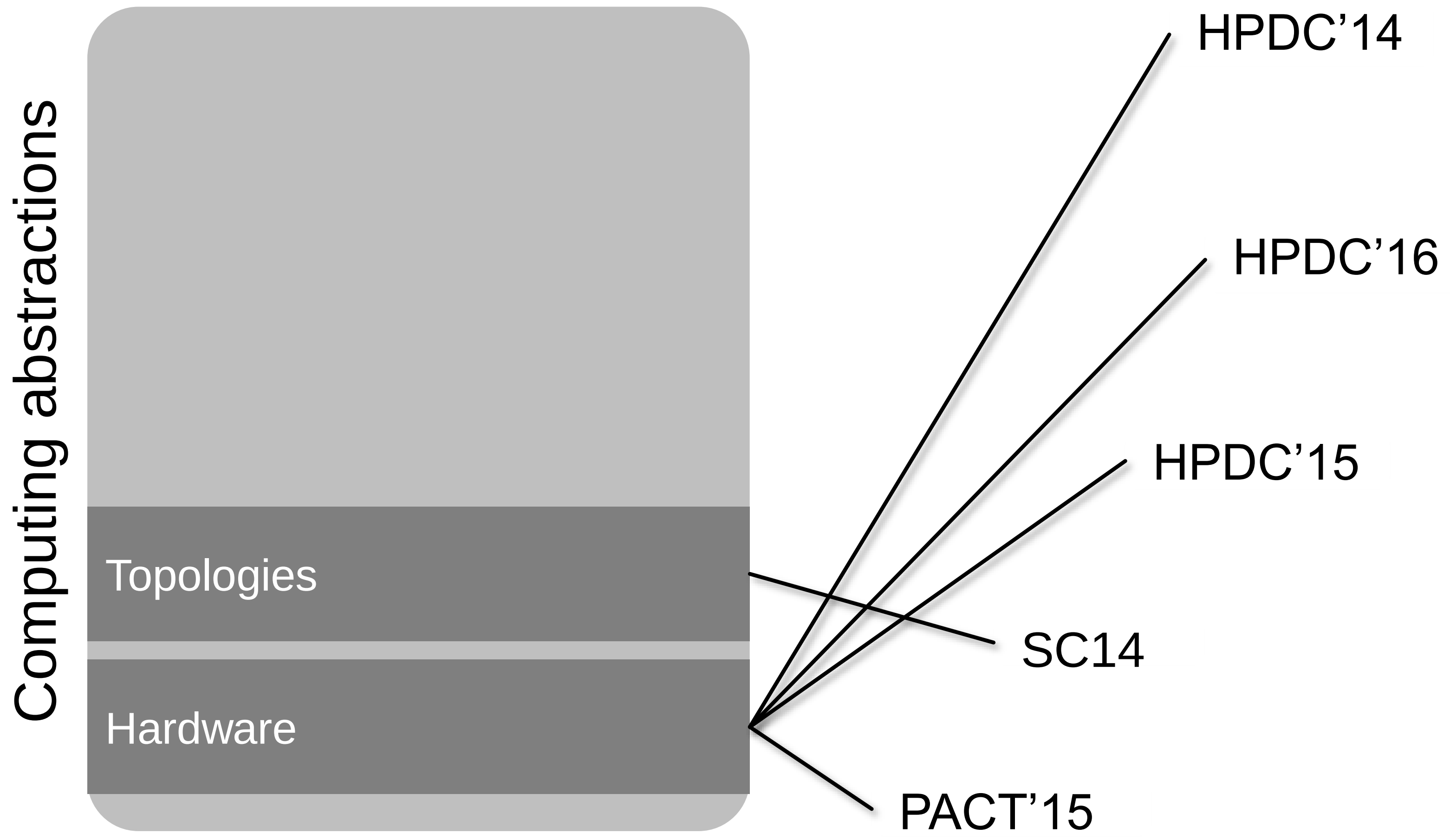
A Brief Summary of Research I Do in HPC

Computing abstractions

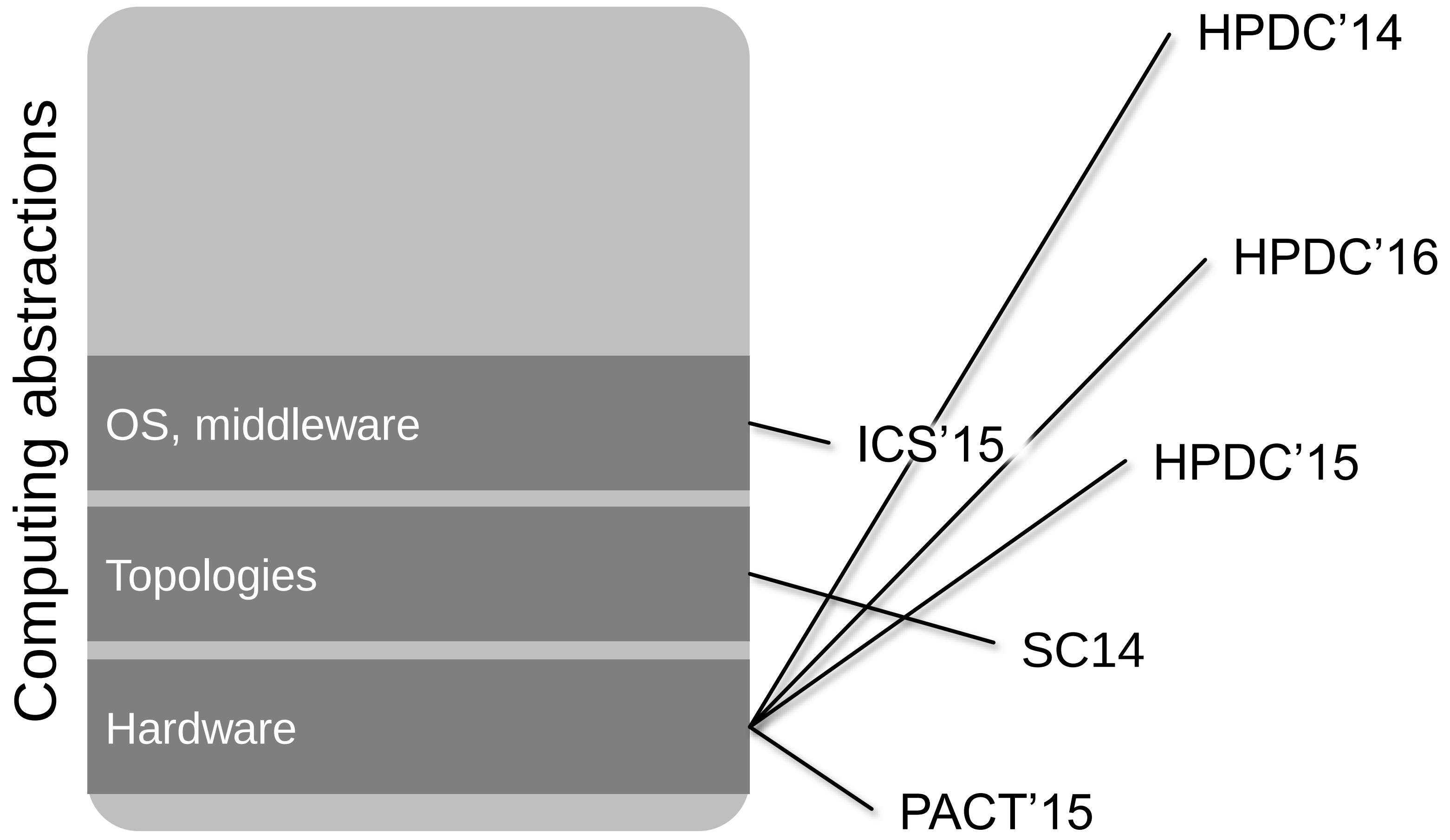
A Brief Summary of Research I Do in HPC



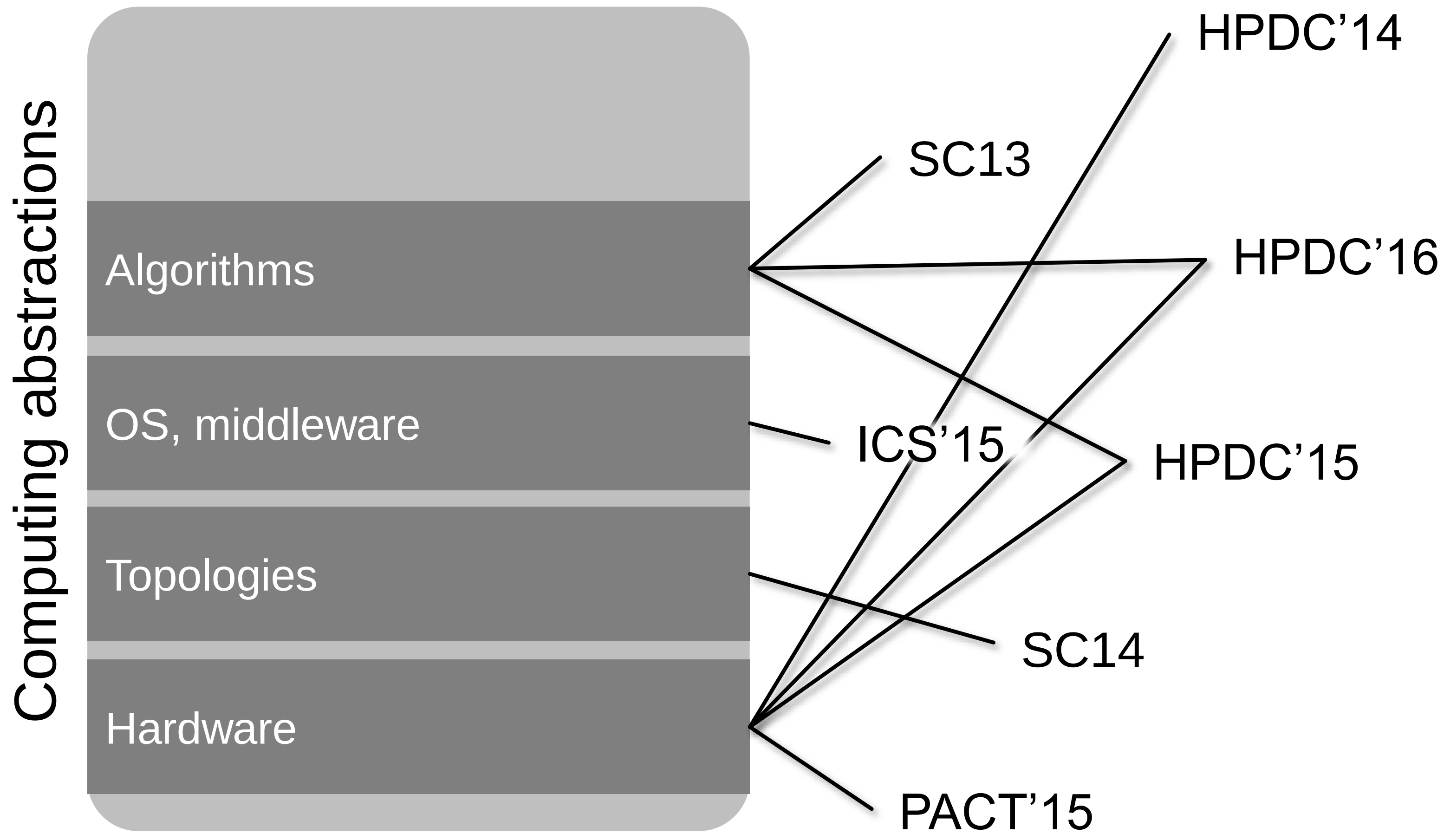
A Brief Summary of Research I Do in HPC



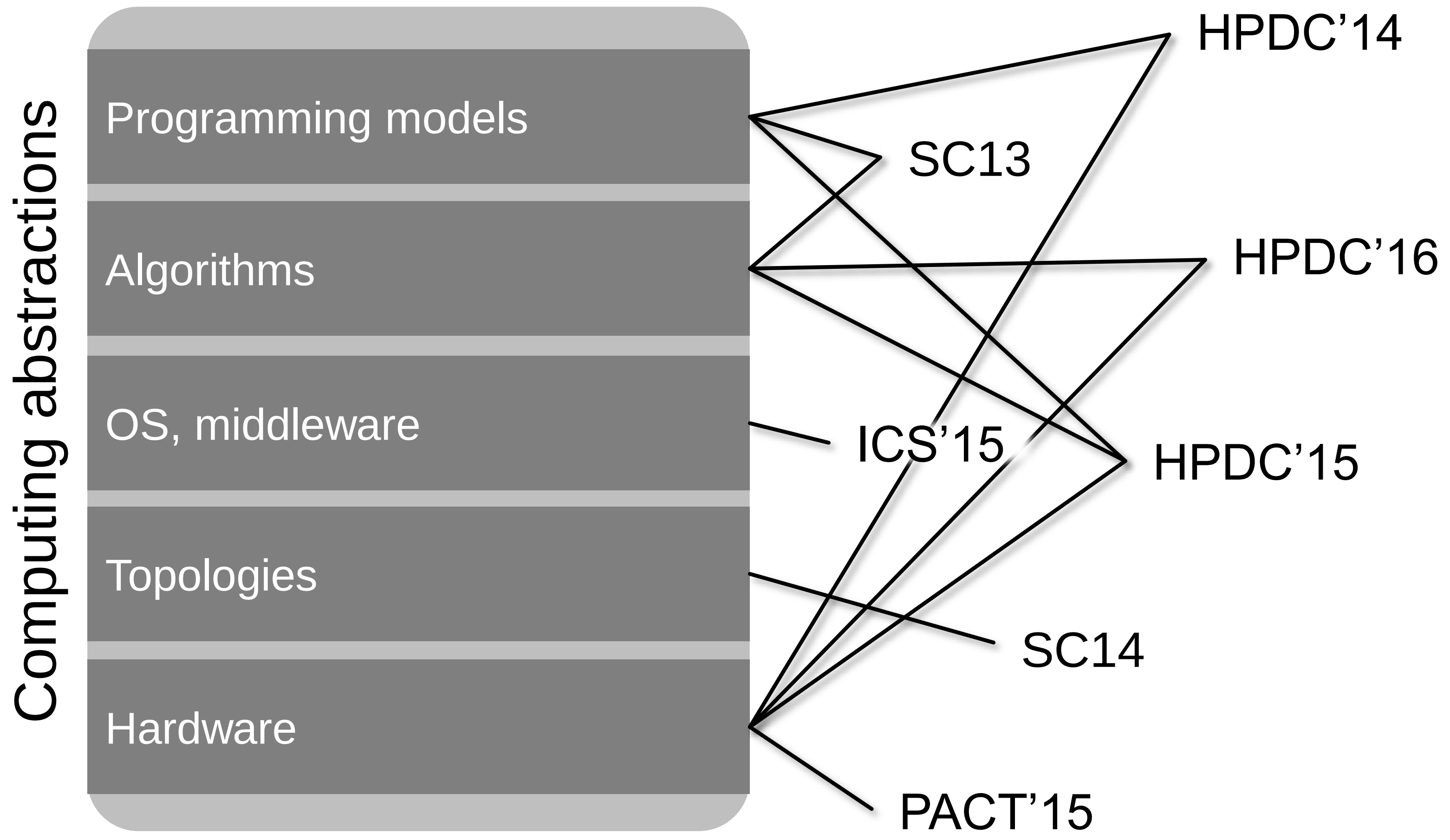
A Brief Summary of Research I Do in HPC



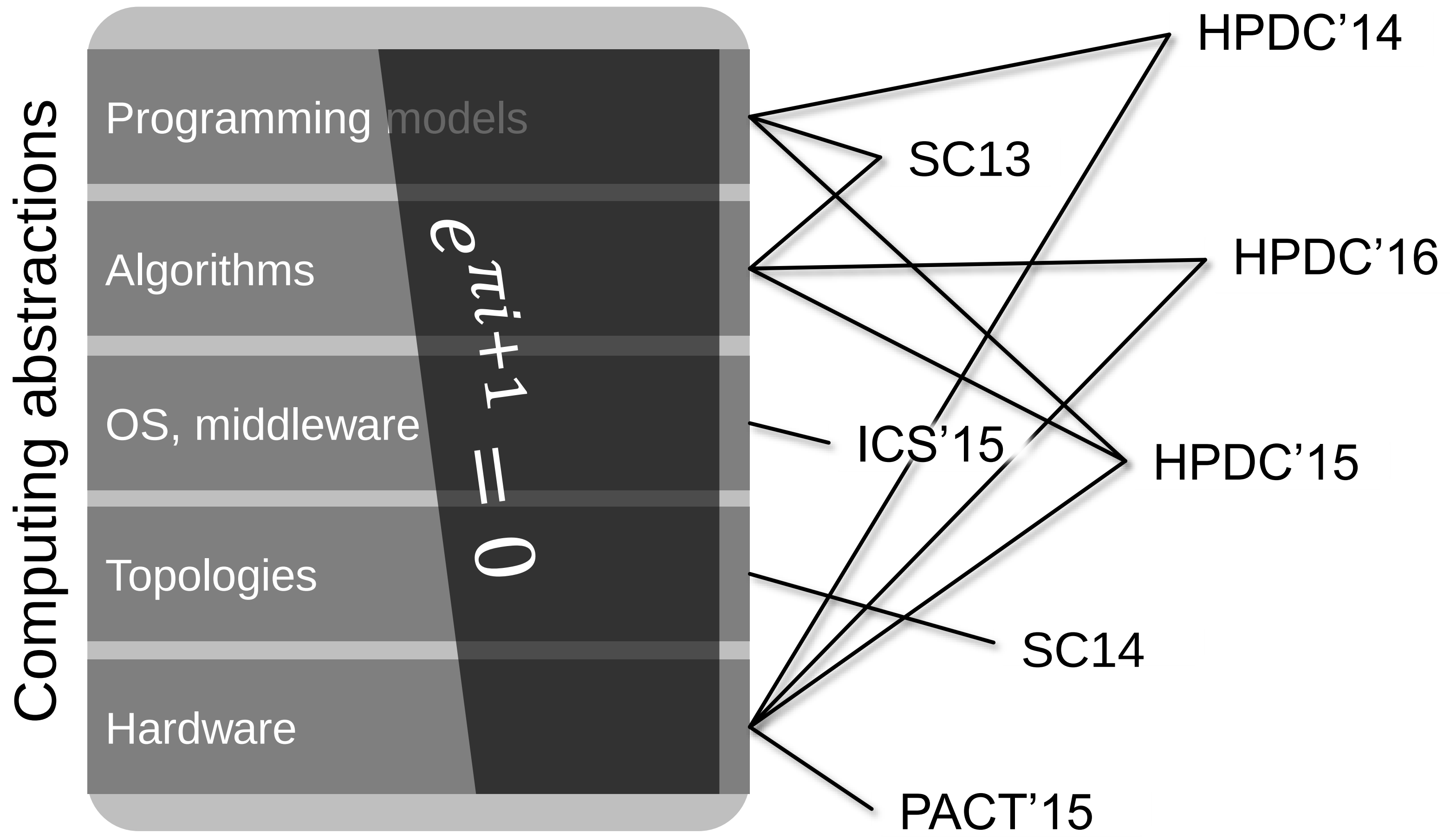
A Brief Summary of Research I Do in HPC



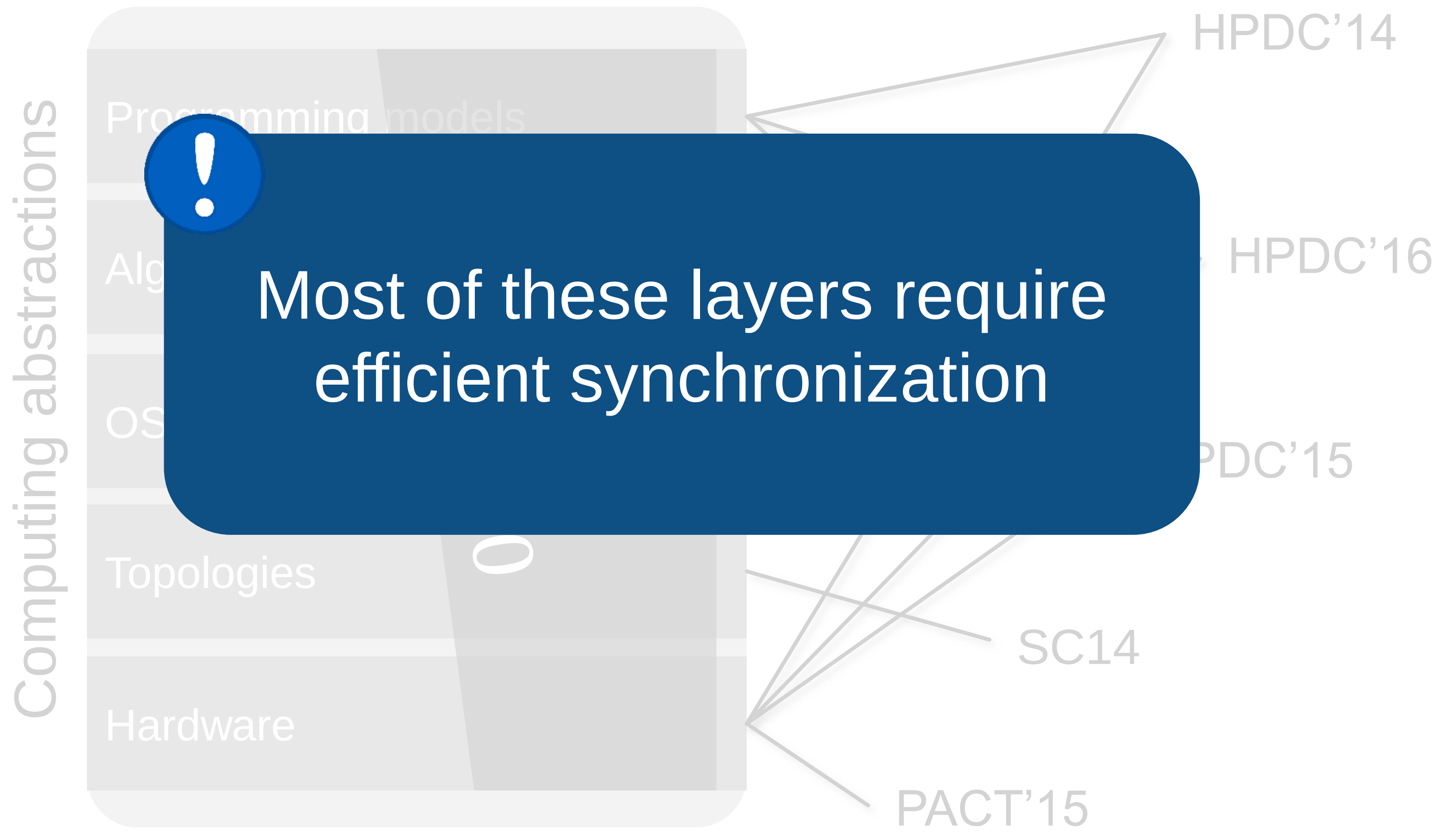
A Brief Summary of Research I Do in HPC



A Brief Summary of Research I Do in HPC



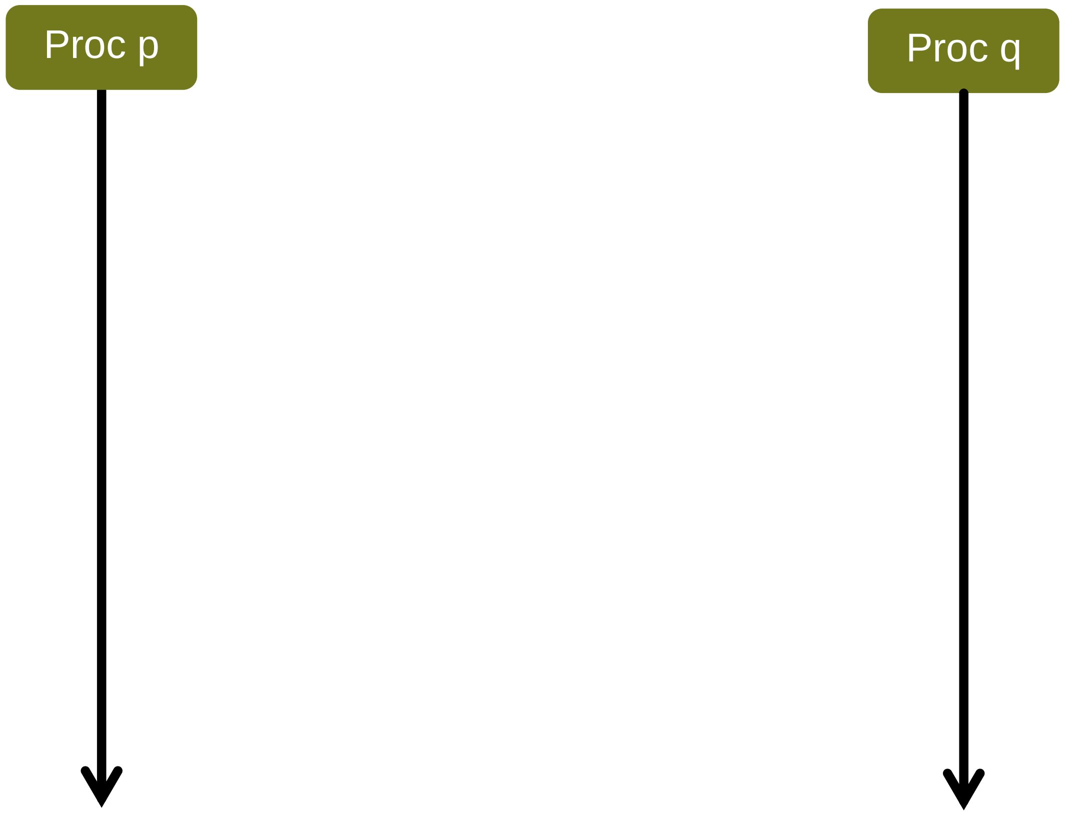
A Brief Summary of Research I Do in HPC



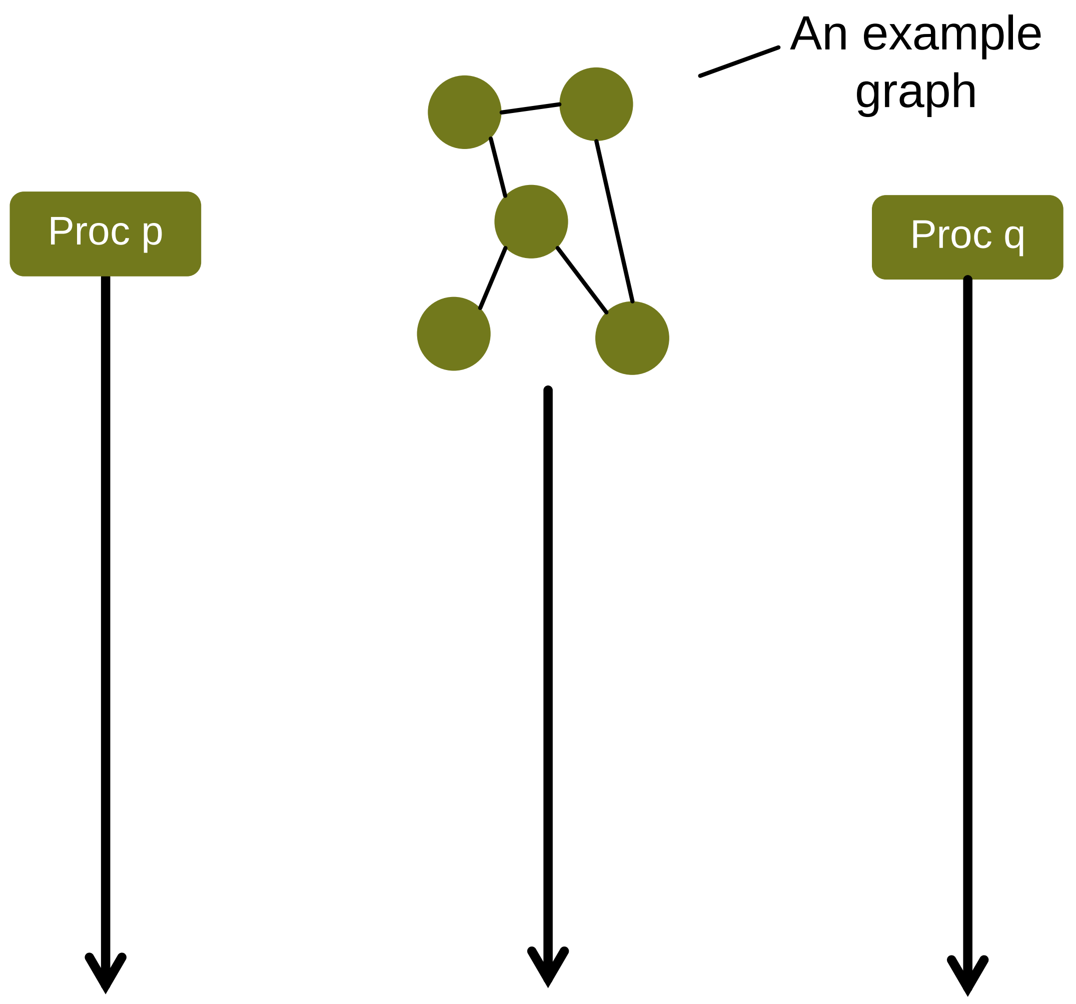
Most of these layers require efficient synchronization

Synchronization Mechanisms
Locks

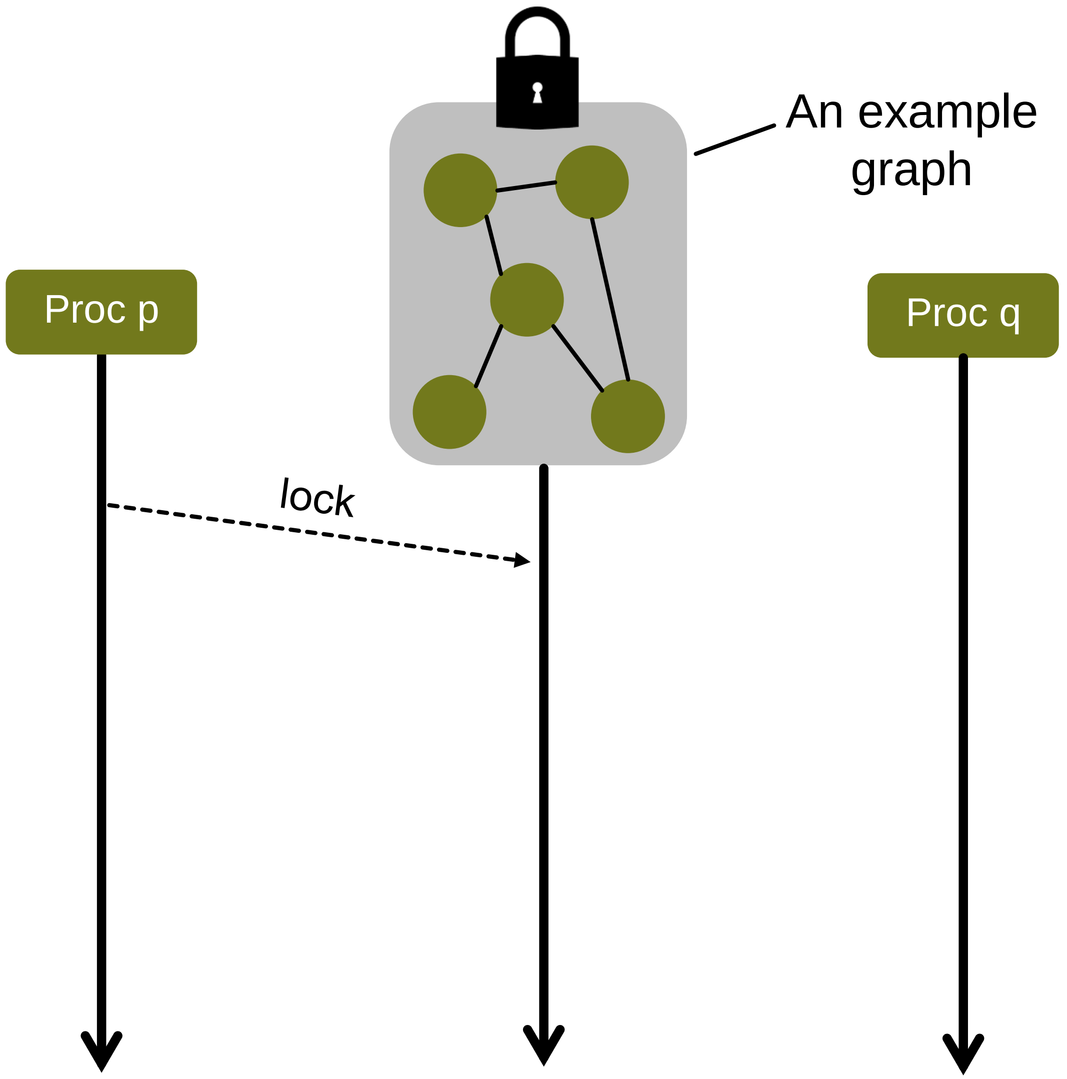
SYNCHRONIZATION MECHANISMS
LOCKS



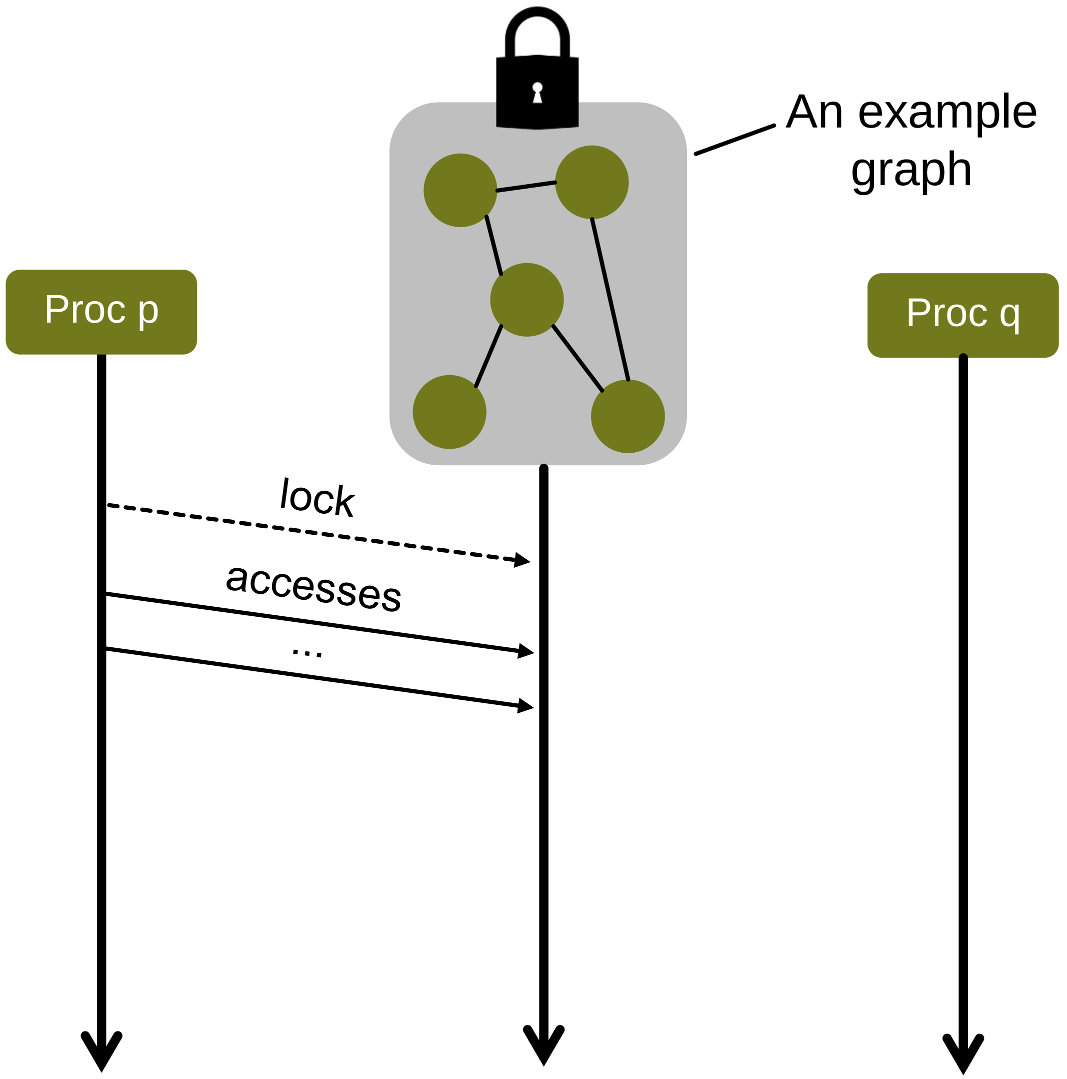
SYNCHRONIZATION MECHANISMS
LOCKS



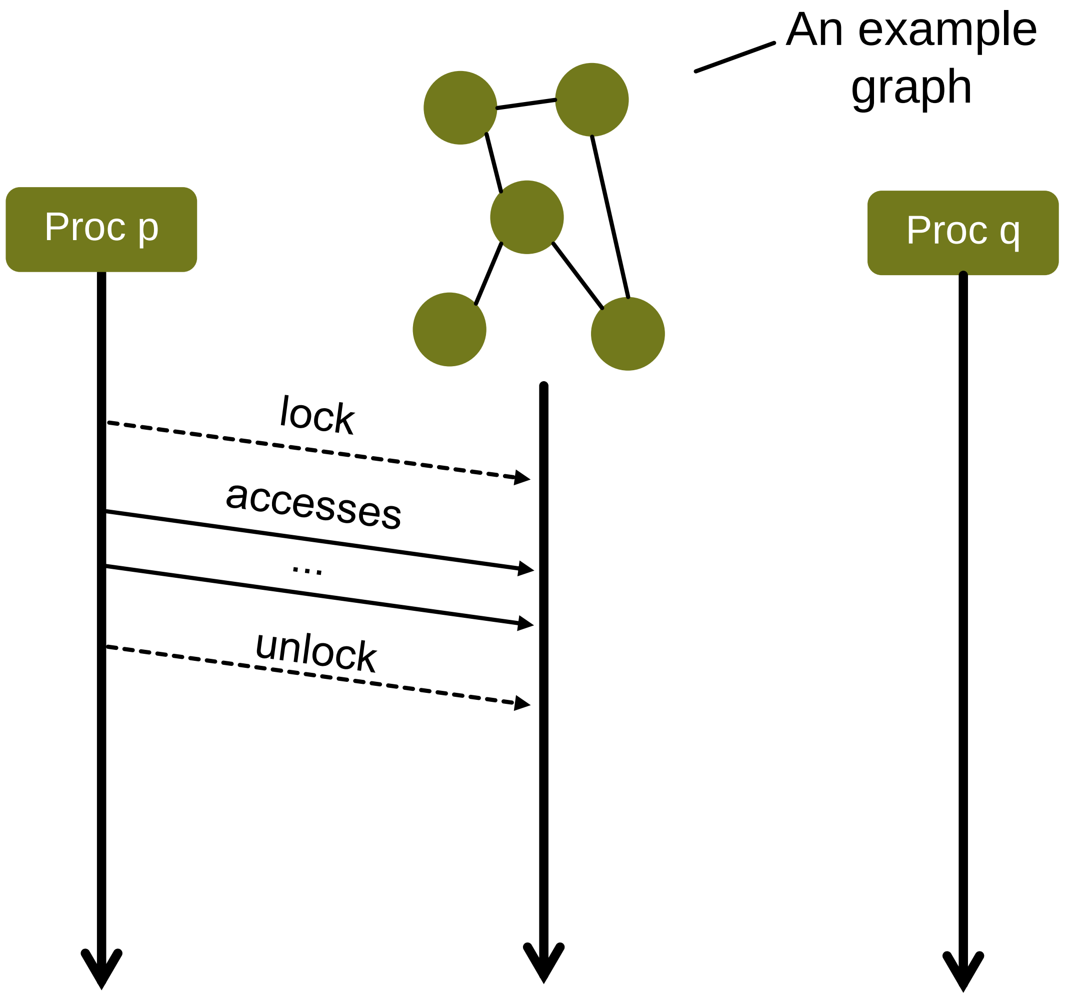
Synchronization Mechanisms
Locks



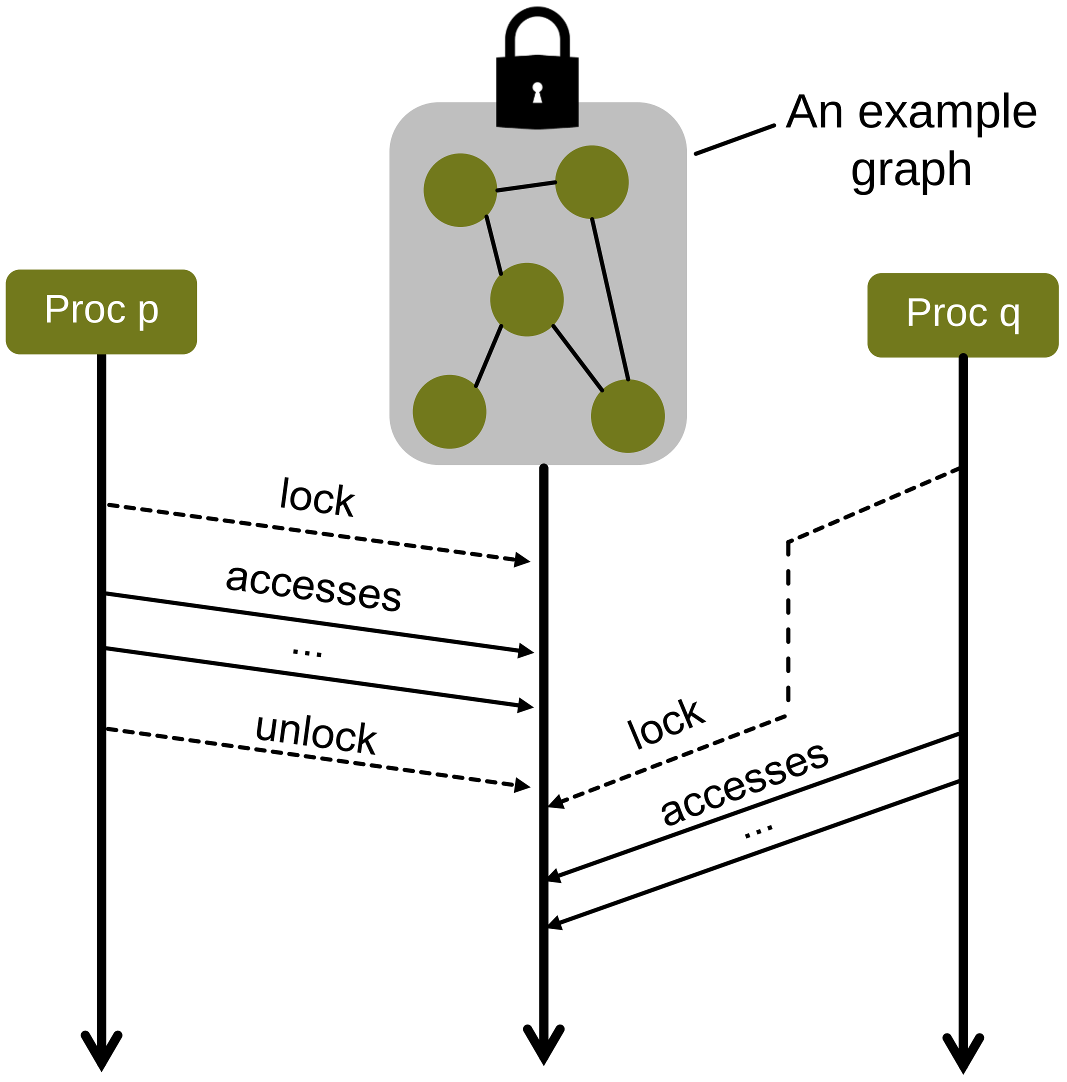
Synchronization Mechanisms
Locks



Synchronization Mechanisms
Locks

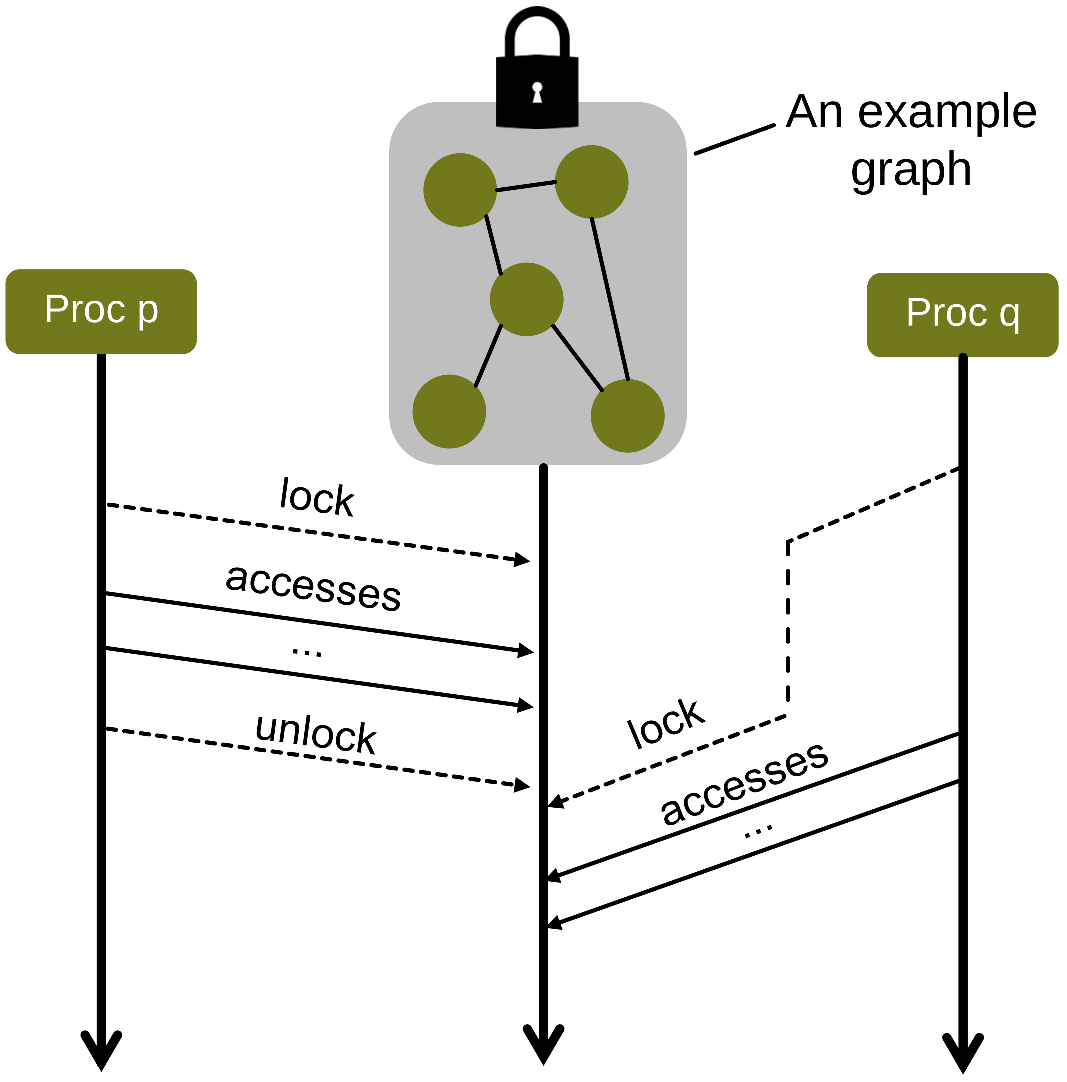


Synchronization Mechanisms
Locks



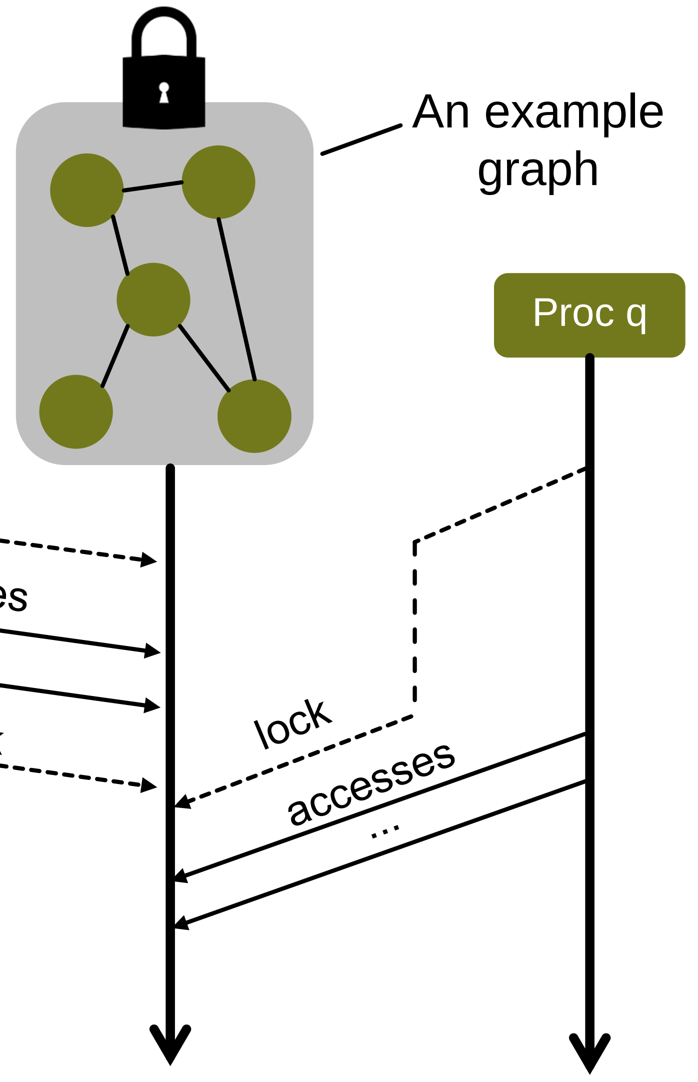
Synchronization Mechanisms
Locks

✓ Intuitive semantics



Synchronization Mechanisms
Locks

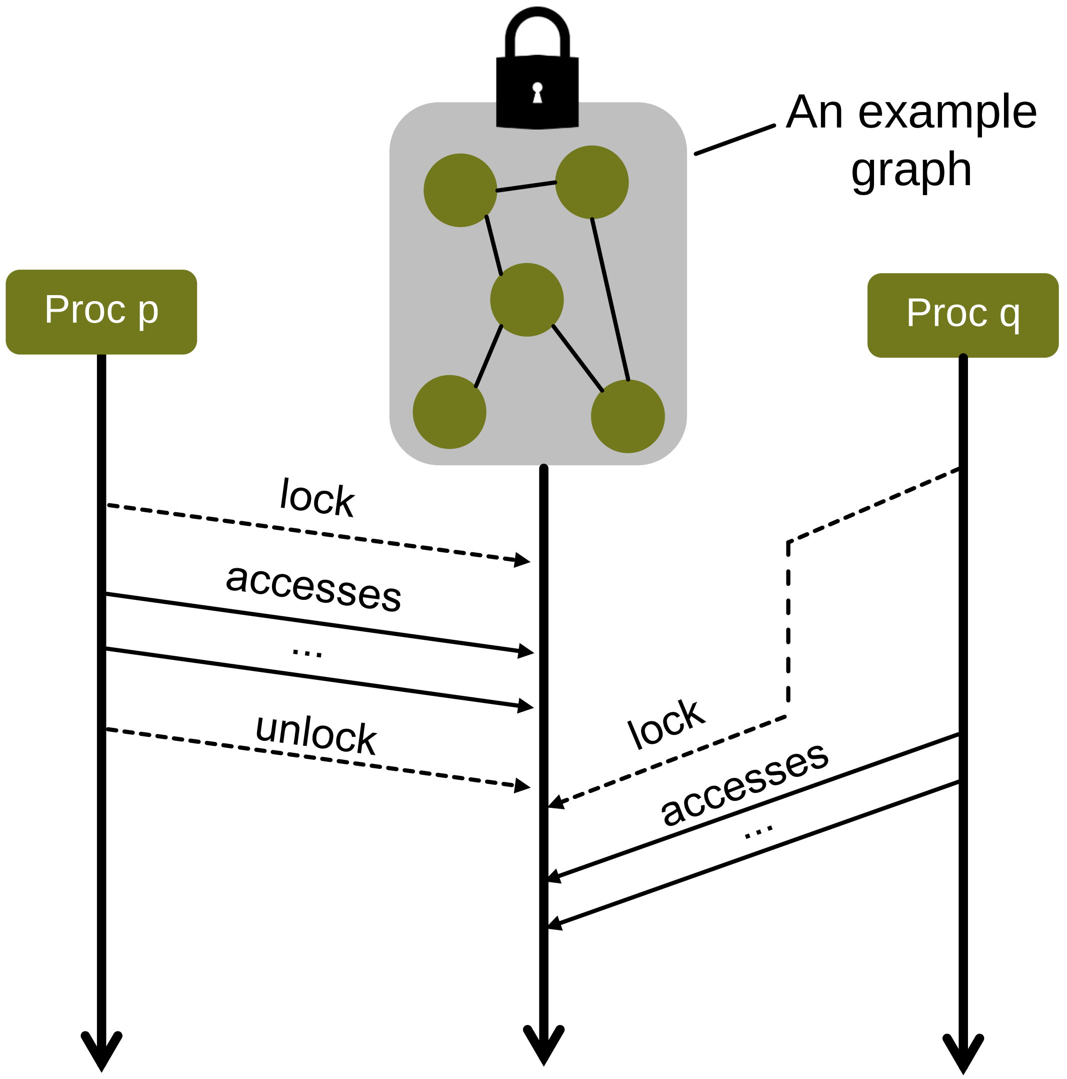
✓ Intuitive semantics



Synchronization Mechanisms
Locks

✓ Intuitive semantics

✗ Serialization

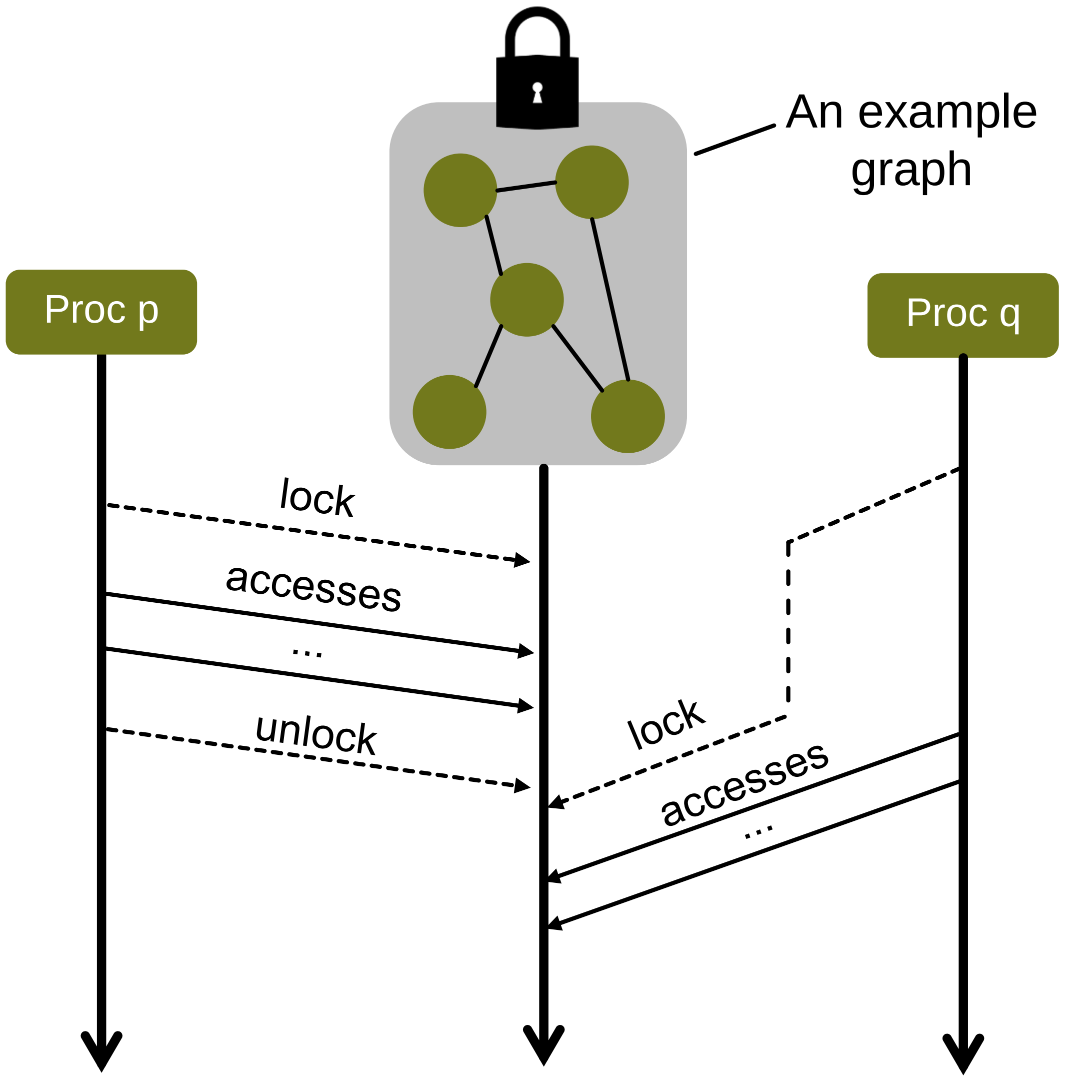


Synchronization Mechanisms
Locks

✓ Intuitive semantics

✗ Serialization

✗ Possibly complex protocols



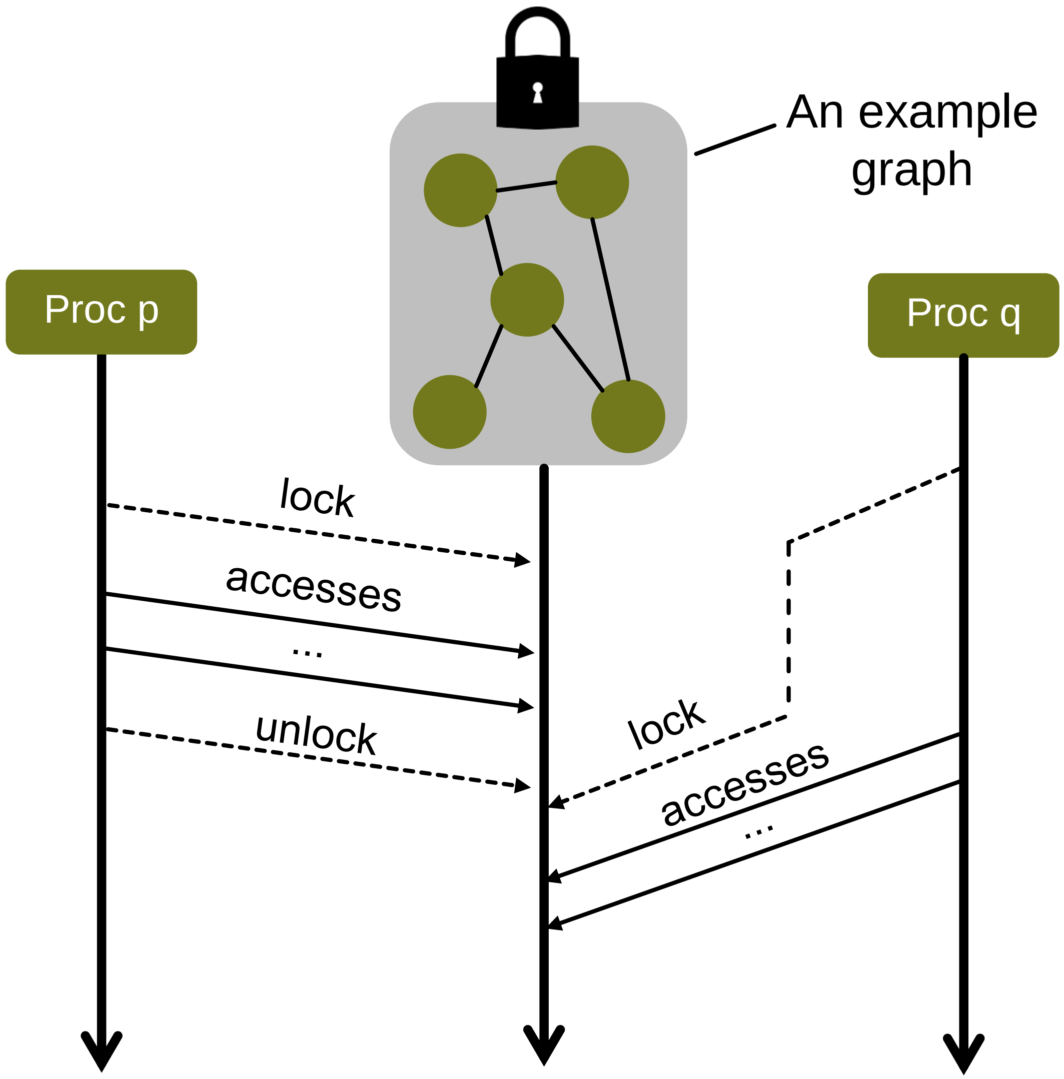
Synchronization Mechanisms
Locks

✓ Intuitive semantics

✗ Serialization

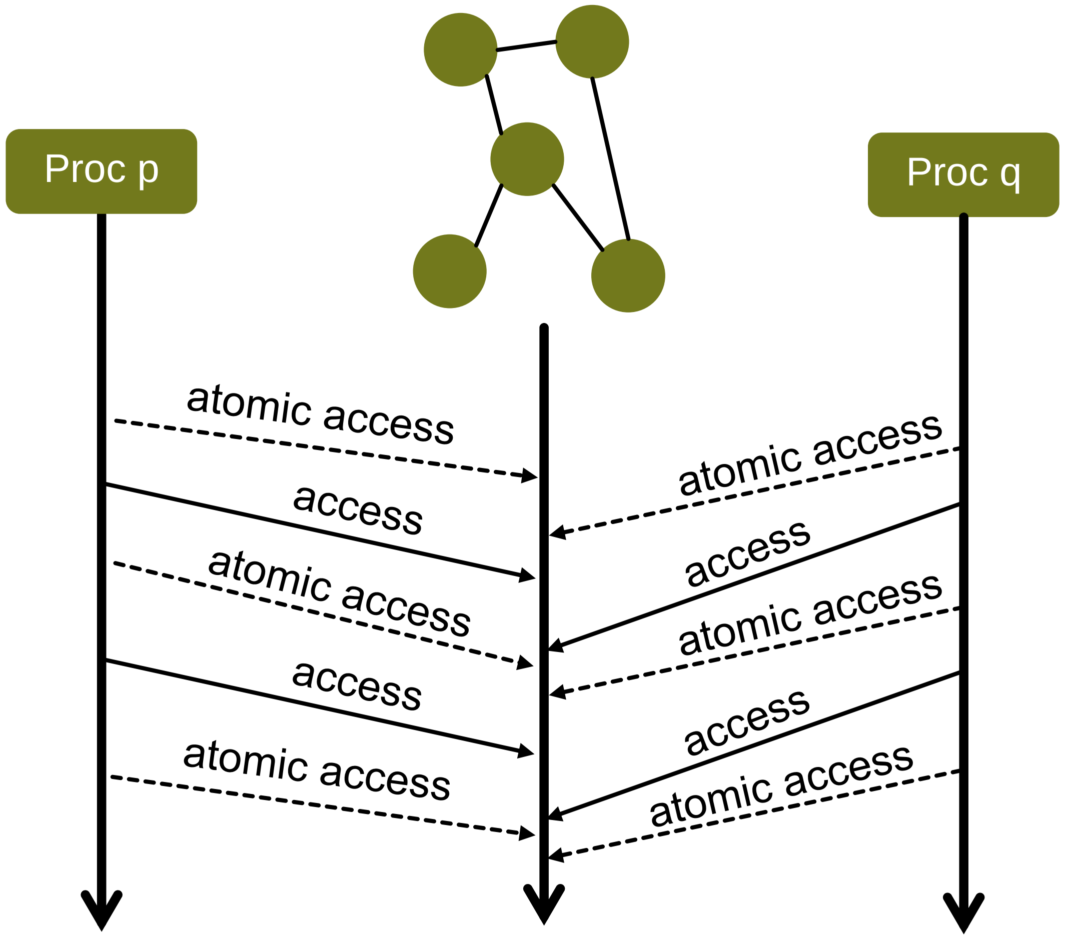
✗ Possibly complex protocols

? High performance distributed locks?



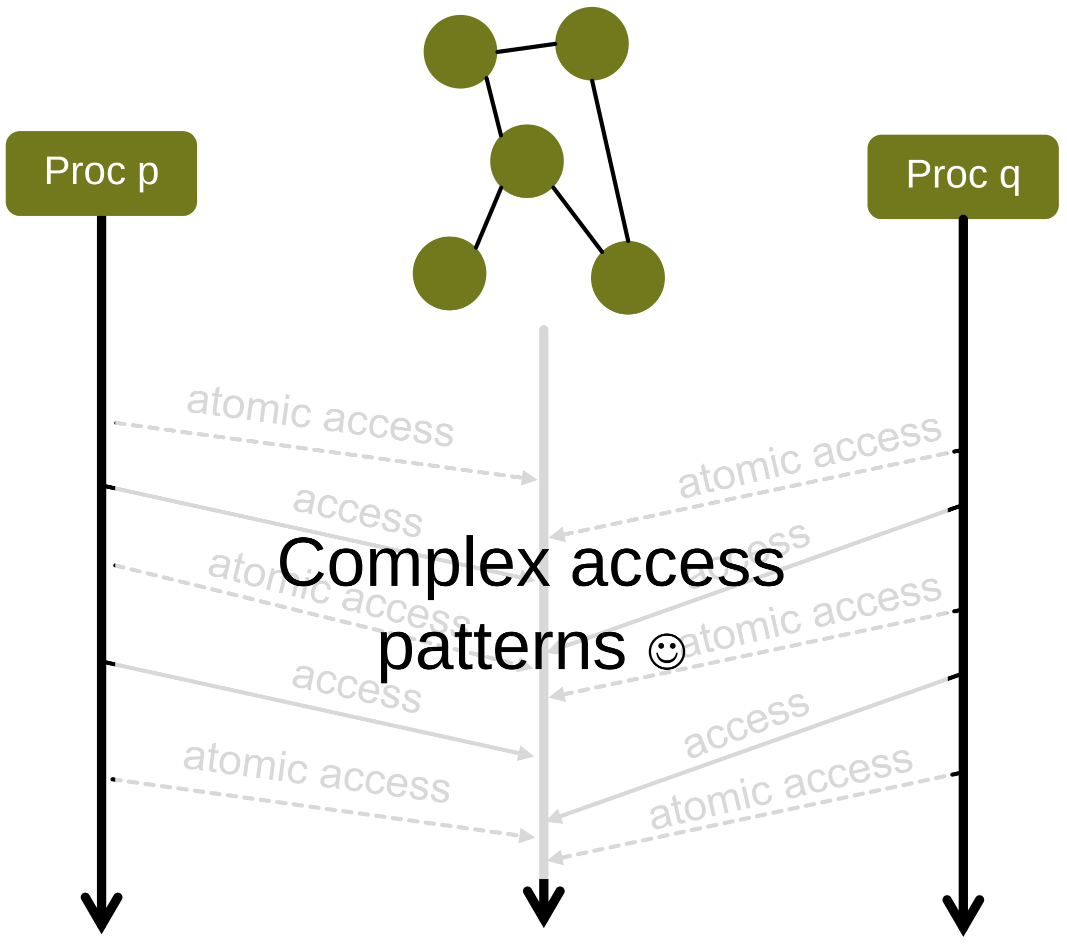
Synchronization Mechanisms

Atomic operations



Synchronization Mechanisms

Atomic operations

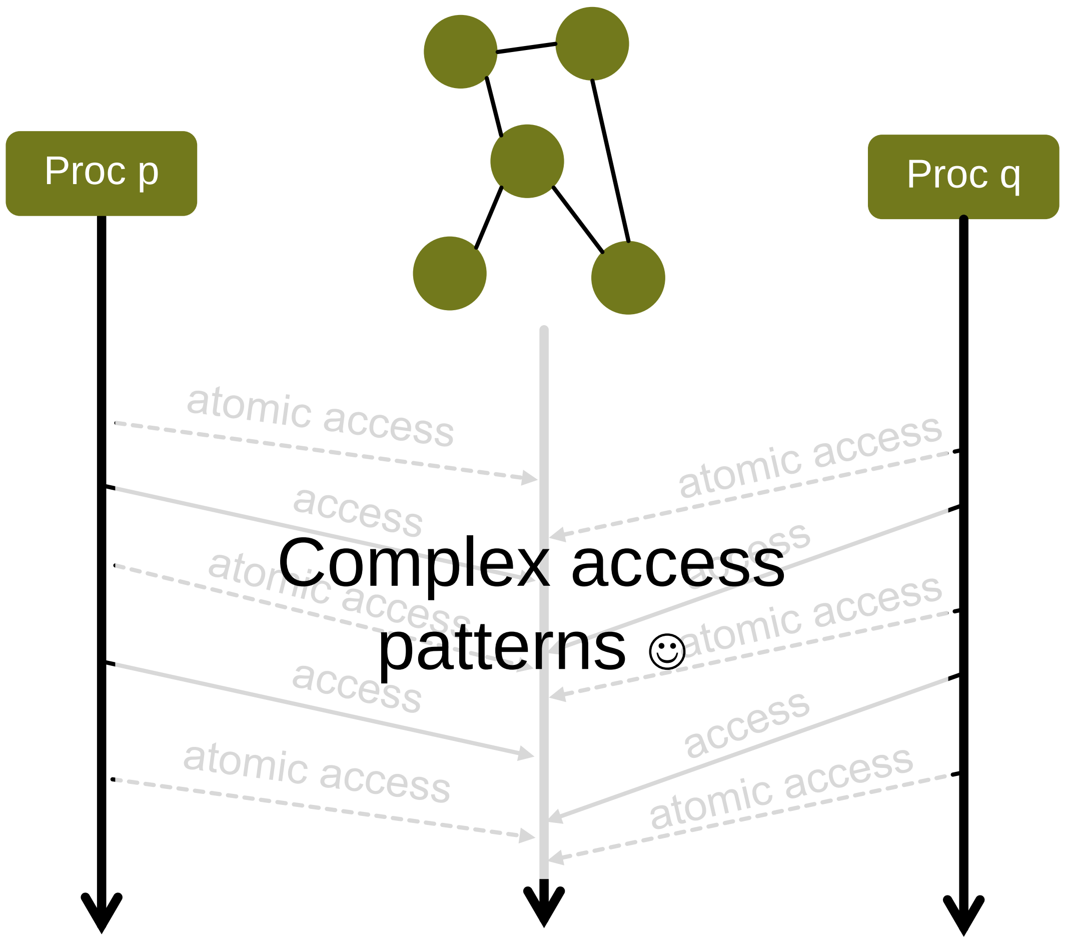


Complex access patterns ☺

Synchronization Mechanisms

Atomic operations

✓ High performance

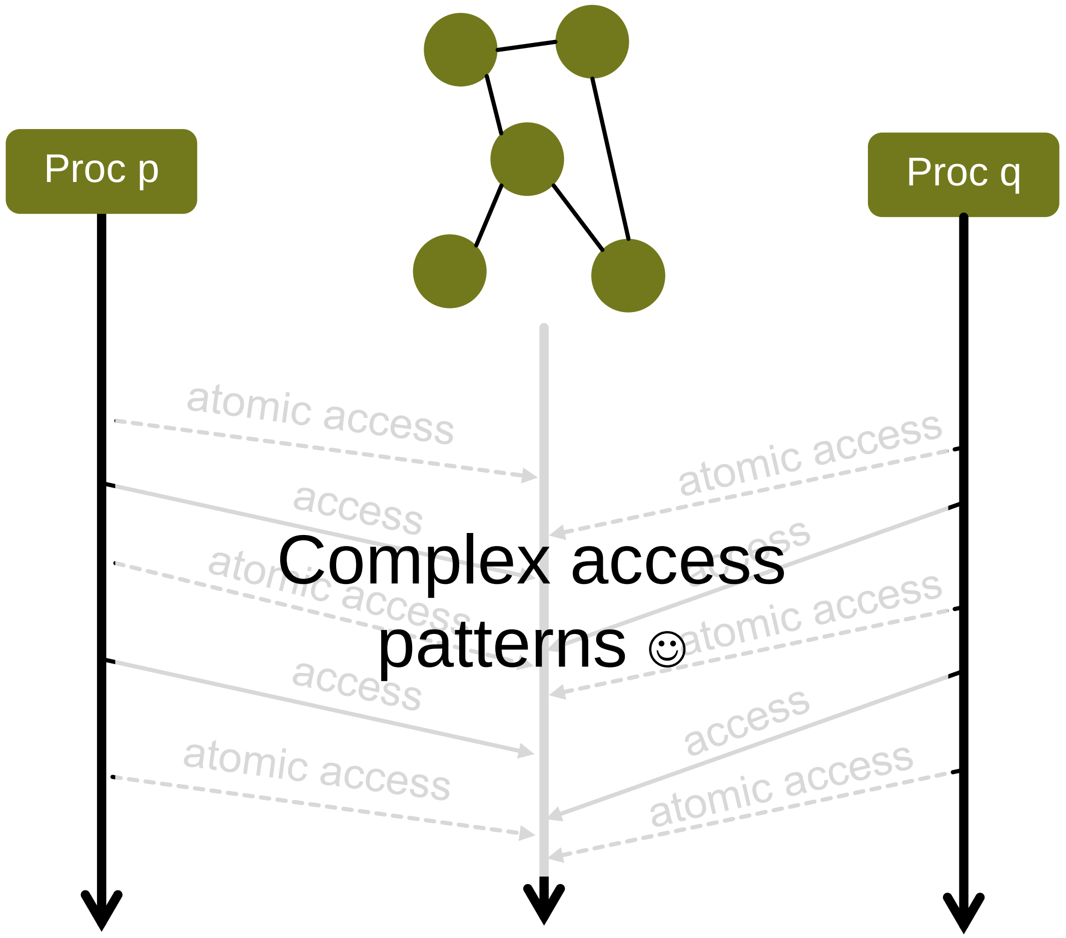


Complex access patterns ☺

SYNCHRONIZATION MECHANISMS
ATOMIC OPERATIONS

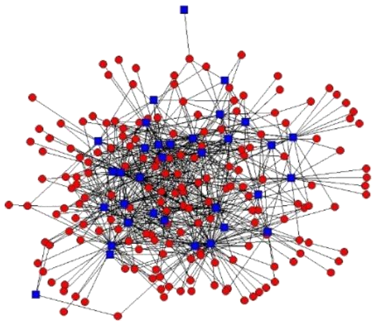
✓ High performance

✓ Very common, truly hardware mechanizm



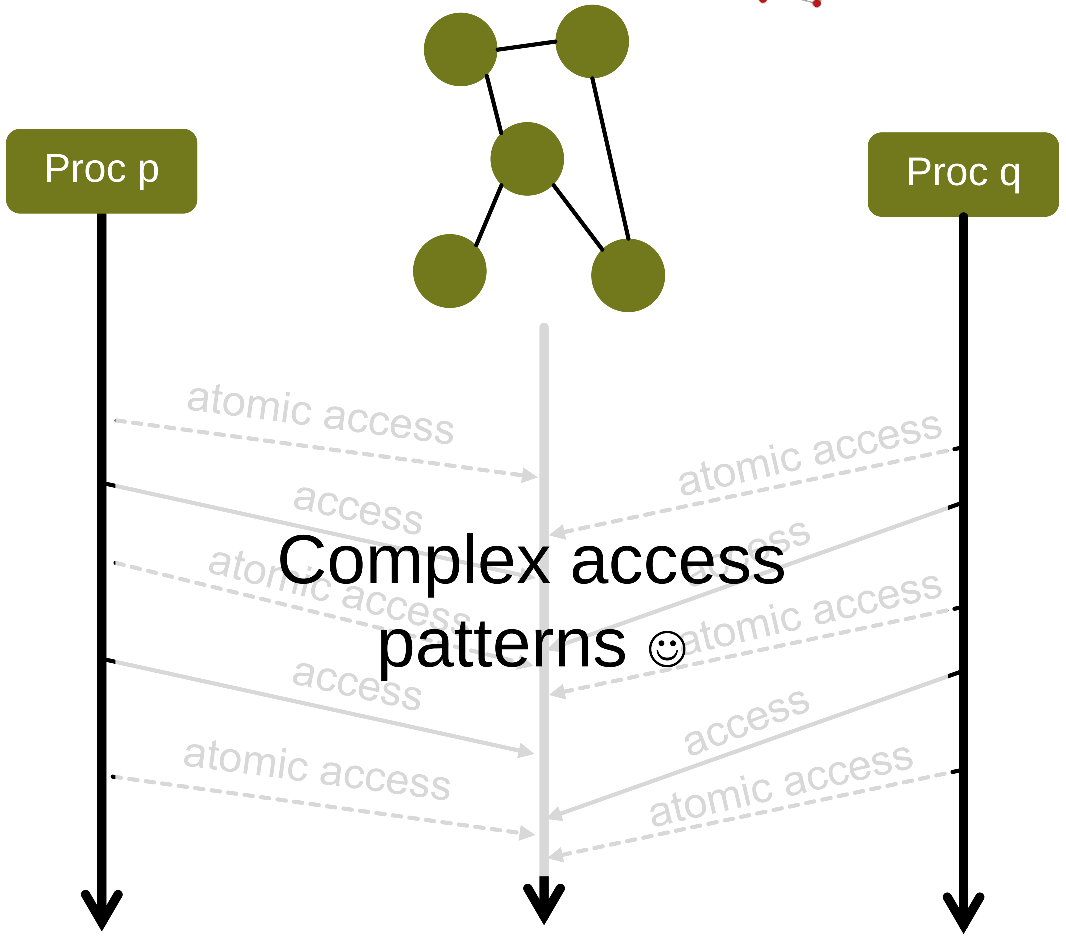
Complex access patterns ☺

Synchronization Mechanisms
Atomic operations



High performance

Very common, truly hardware mechanizm



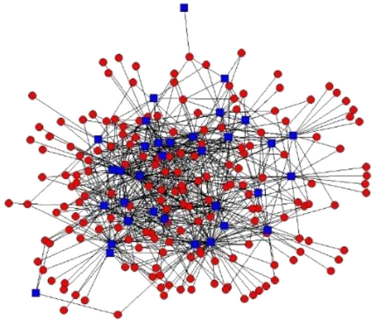
Complex access patterns ☺

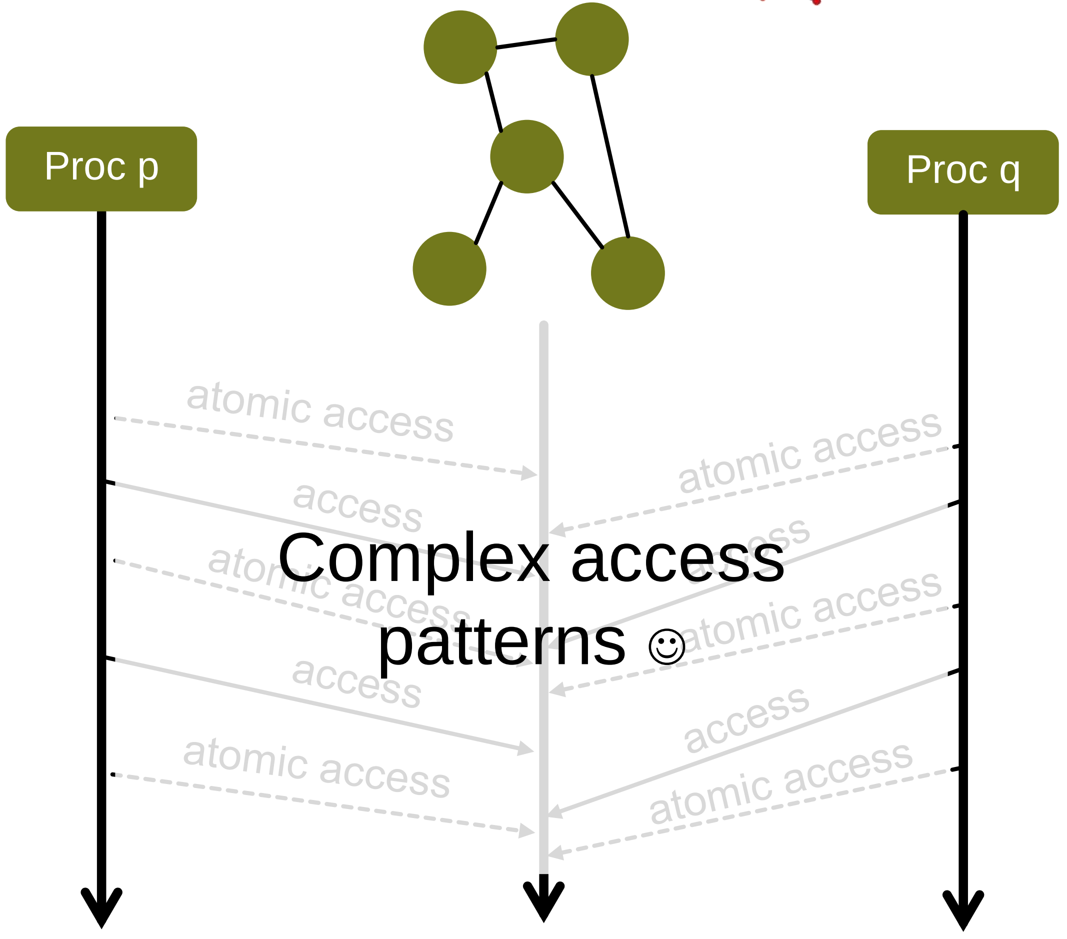
SYNCHRONIZATION MECHANISMS
ATOMIC OPERATIONS

✓ High performance

✓ Very common, truly hardware mechanizm

✗ Complex protocols

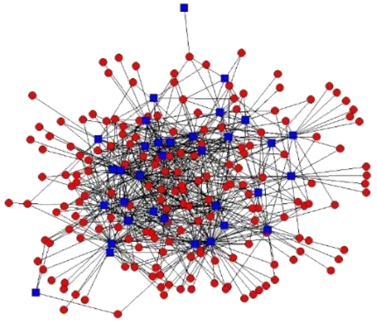




Complex access patterns ☺

Synchronization Mechanisms

Atomic operations

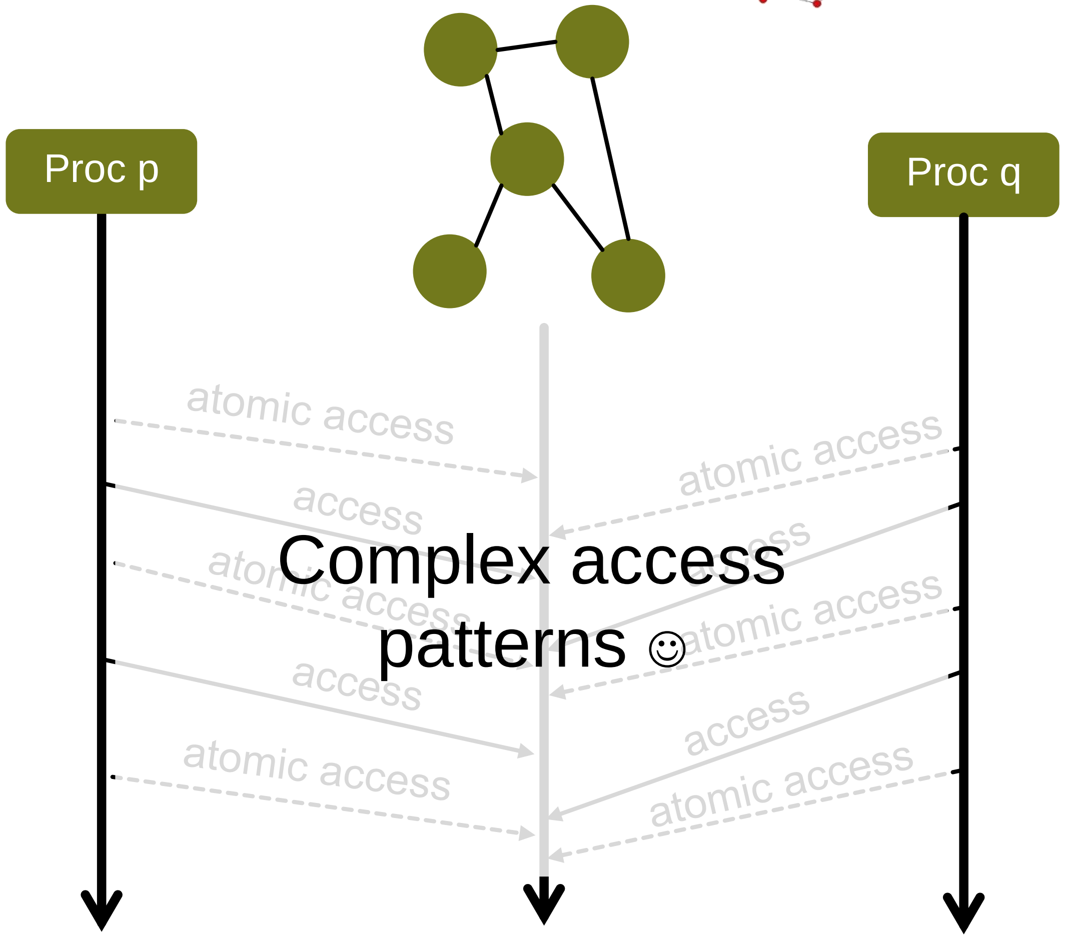


✓ High performance

✓ Very common, truly hardware mechanizm

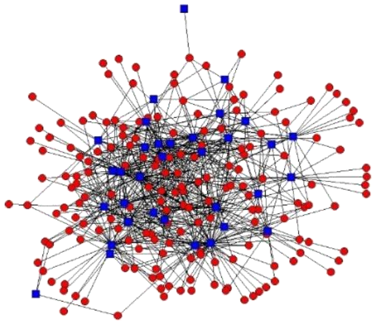
✗ Complex protocols

✗ Subtle issues (ABA problem, ...)



Complex access patterns ☺

Synchronization Mechanisms
Atomic operations



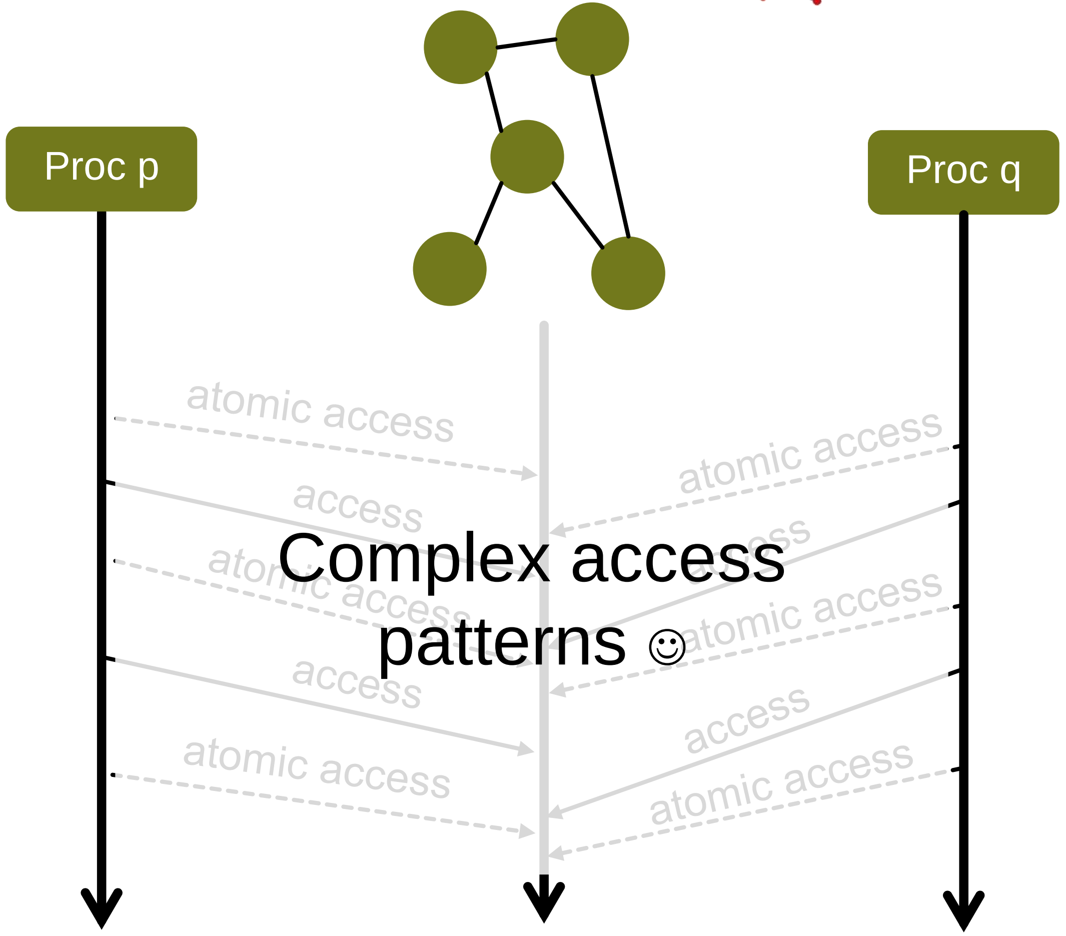
✓ High performance

✓ Very common, truly hardware mechanizm

✗ Complex protocols

✗ Subtle issues (ABA problem, ...)

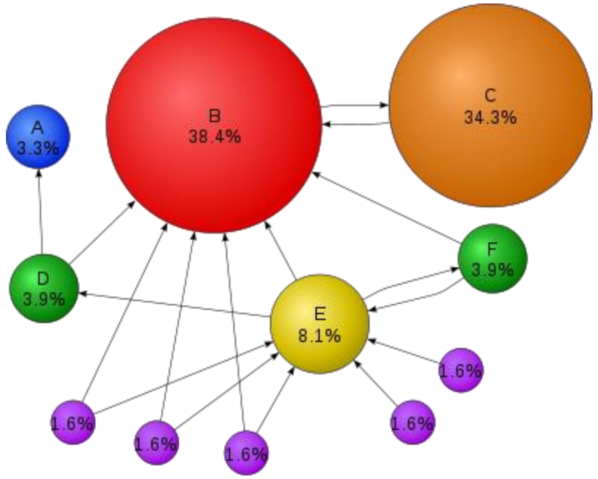
? Do we really understand their performance?



Complex access patterns ☺

ATOMICS: POPULARITY

ATOMICS: POPULARITY



ATOMICS: POPULARITY

[PPoPP'14] A General Technique for Non-blocking Trees

[PPoPP'15]

[SPAA'16]

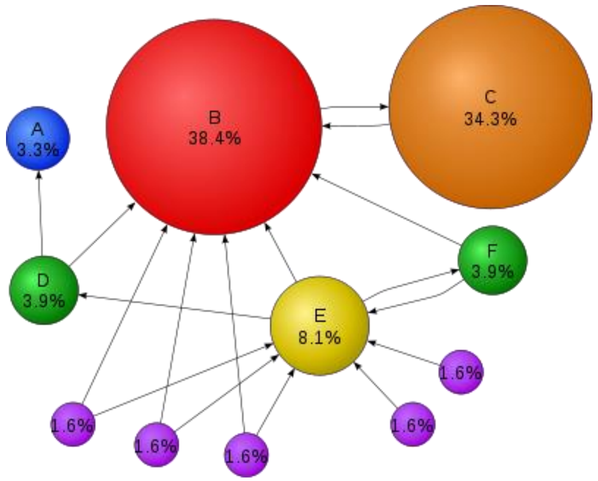
[PPoPP'14] Fast Concurrent Lock-Free Binary Search Trees

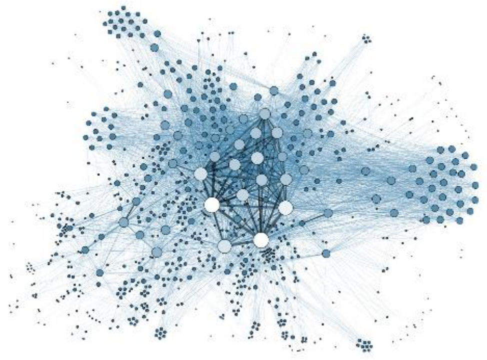
[PPoPP'14] A Practical Wait-Free Simulation for Lock-Free Data Structures

[PPoPP'15]

[PPoPP'14] Practical Concurrent Binary Search Trees via Logical Ordering

[SPAA'15]



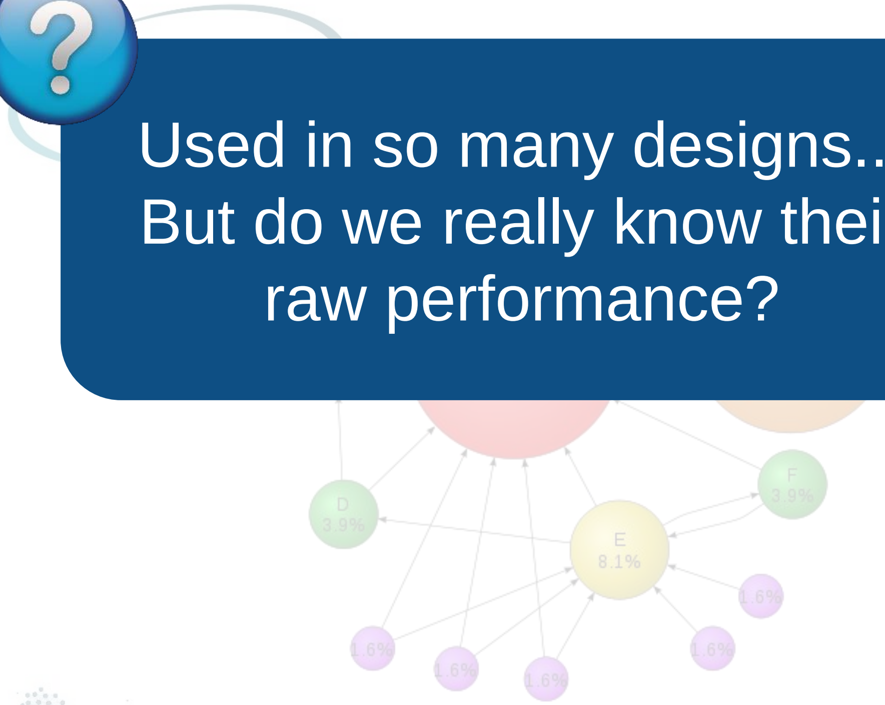


ATOMICS: POPULARITY

[PPoPP'15]

[PPoPP'14] A General Technique for Non-blocking Trees

'16]



Used in so many designs...
But do we really know their raw performance?

[PPoPP'14] Fast Concurrent Lock-Free Binary Search Trees

[PPoPP'14] A Practical Wait-Free Simulation for Lock-Free Data Structures

[PPoPP'15]

[SPAA'15]

[PPoPP'14] Practical Concurrent Binary Search Trees via Logical Ordering

Atomics: Popularity

[PPoPP'15]

[PPoPP'14] A General Technique for Non-blocking Trees

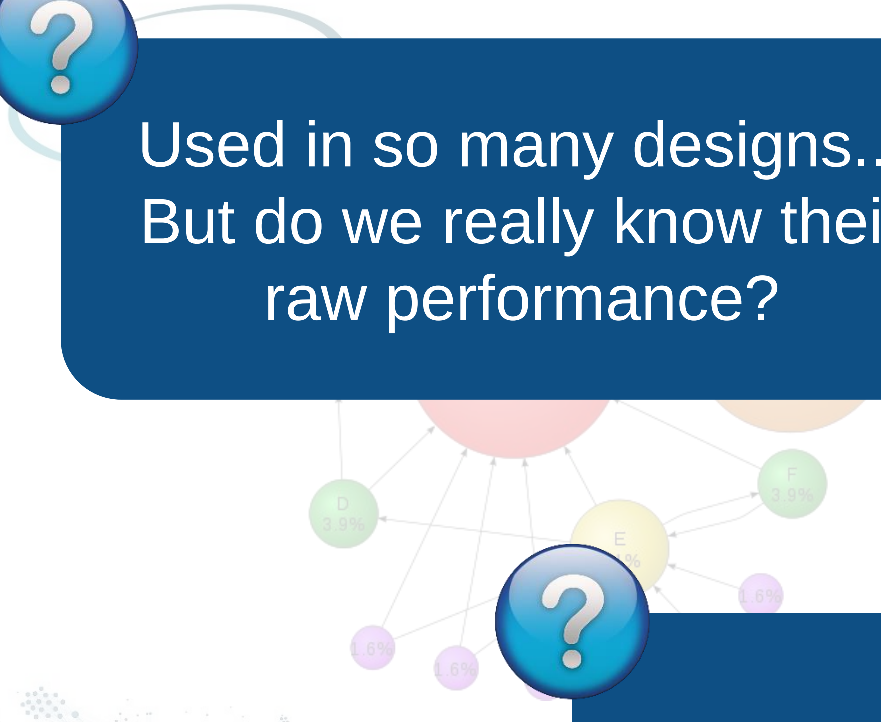
'16]

[PPoPP'14] Fast Concurrent Lock-Free Binary Search Trees

[PPoPP'14] A Practical
for
ures

al
Concurrent Binary Search Trees via Logical Ordering

[SPAA'15]

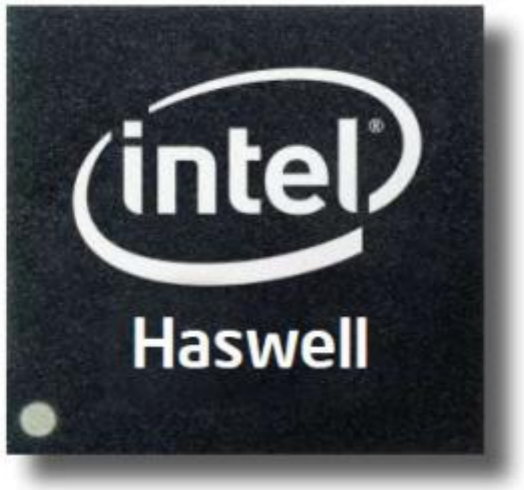


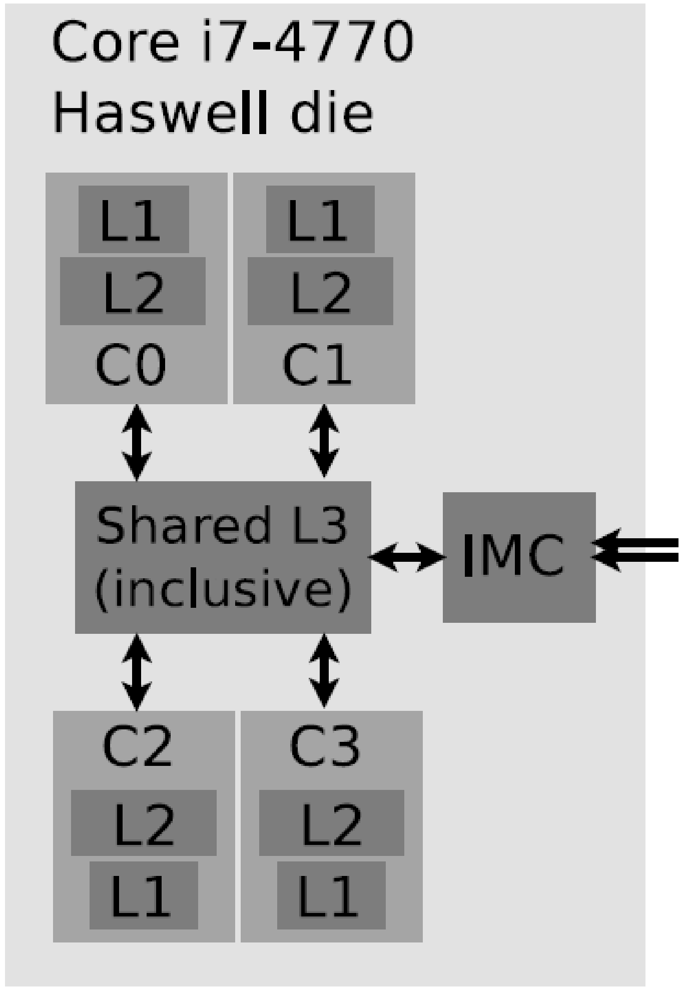
Used in so many designs... But do we really know their raw performance?

Raw performance... Of what?

Atomics: Performance Dimensions

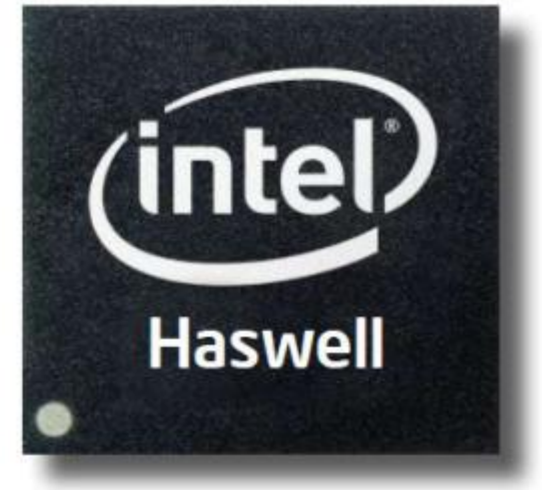
ATOMICS: PERFORMANCE DIMENSIONS

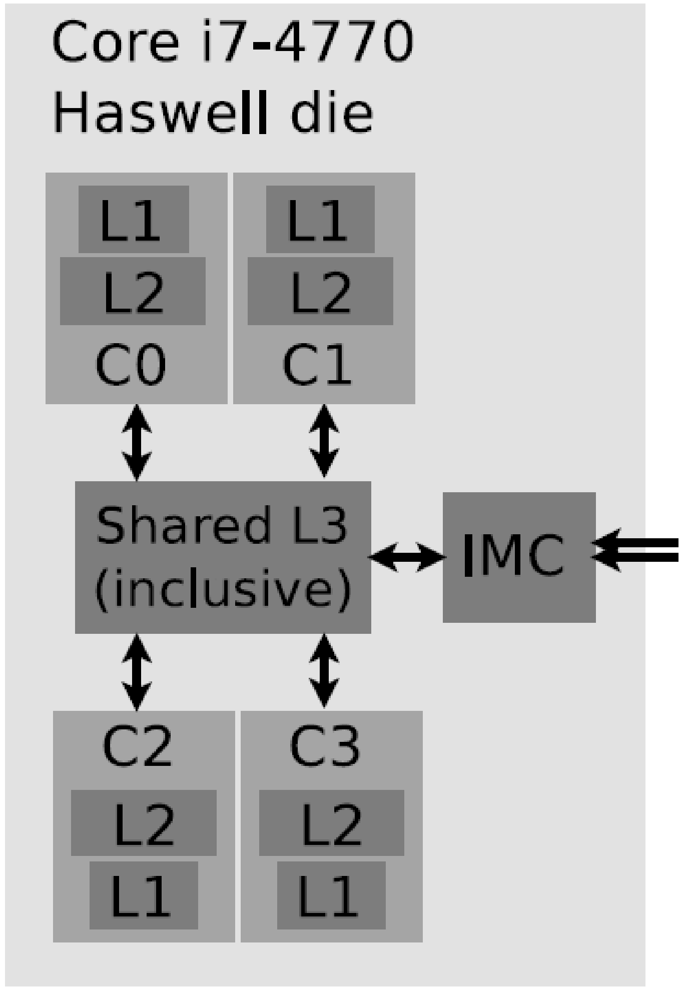




Atomics: Performance Dimensions

Cache level?

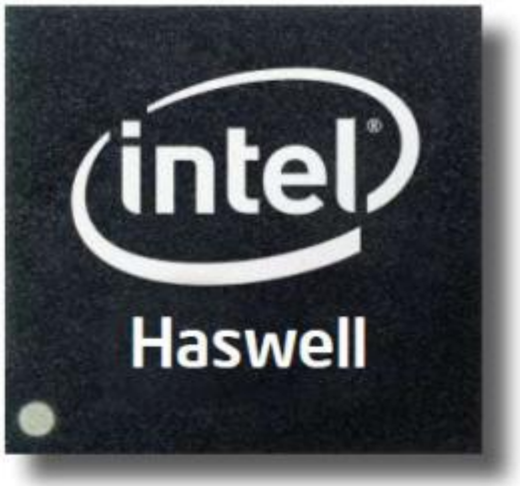


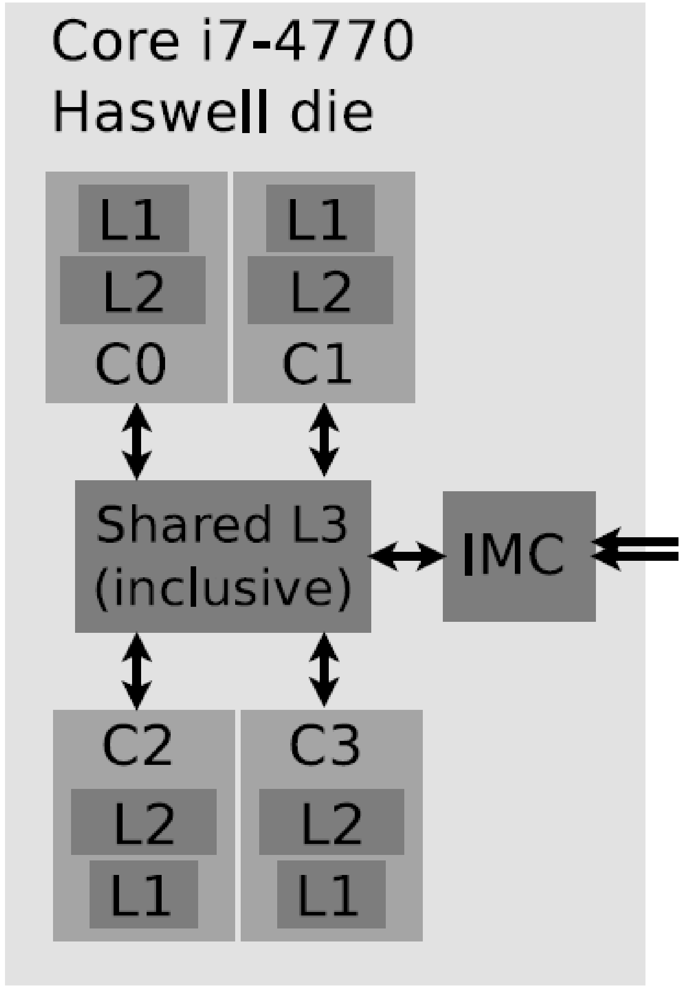


Atomics: Performance Dimensions

Cache level?

Locality?





ATOMICS: PERFORMANCE DIMENSIONS

Atomics: Performance Dimensions

Atomic?

Atomics: Performance Dimensions

Atomic?

Performance metrics?

Atomics: Performance Dimensions

Atomic?

Performance metrics?

Architecture

Atomics: Performance Dimensions

Atomic?

Performance metrics?

Contention?

Architecture

Atomics: Performance Dimensions

Cache coherence state?

Atomic?

Performance metrics?

Contention?

Architecture

Atomics: Performance Dimensions

Cache coherence state?

Atomic?

Operand size?

Performance metrics?

Contention?

Architecture

Atomics: Performance Dimensions

Cache coherence state?

Atomic?

Operand size?

Performance metrics?

Alignment?

Contention?

Architecture

Atomics: Performance Dimensions

Cache coherence state?

Atomic?

Mechanisms?

Operand size?

Performance metrics?

Alignment?

Contention?

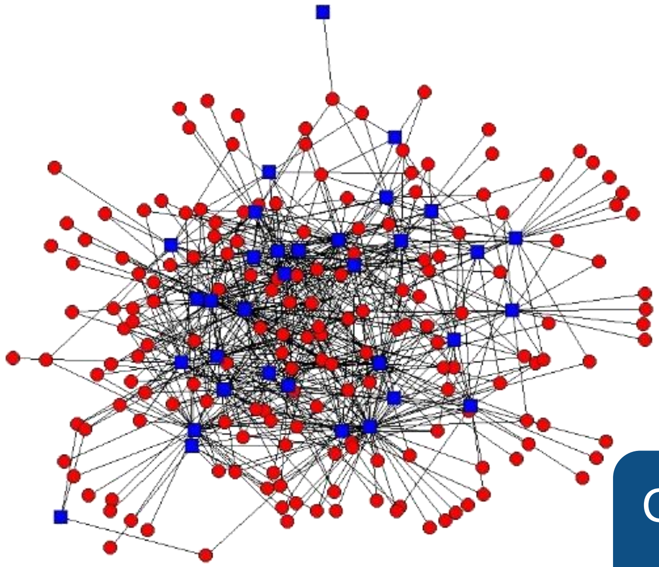
Architecture

Atomics: Performance Dimensions

Atomic?

Atomics: Performance Dimensions

Atomic?

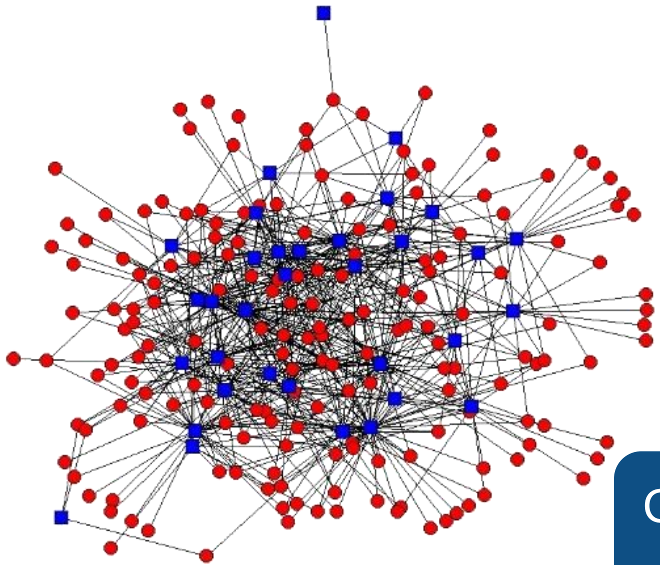


Compare-and-Swap (CAS)

Atomics: Performance Dimensions

Atomic?

Fetch-and-Add (FAA)

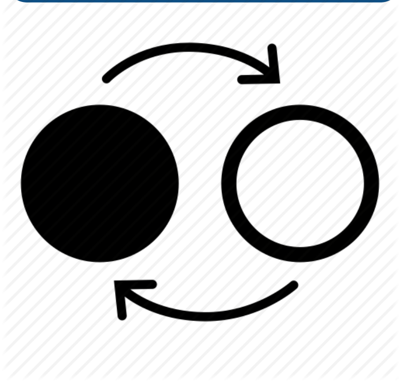


Compare-and-Swap (CAS)

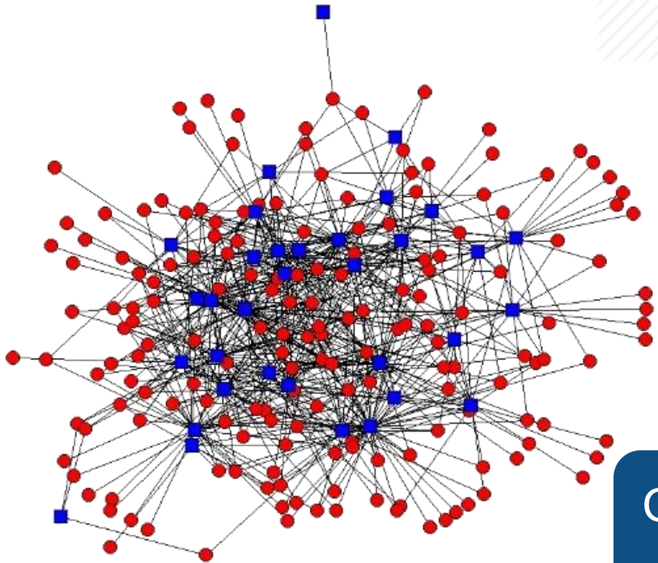
ATOMICS: PERFORMANCE DIMENSIONS

Atomic?

Swap (SWP)



Fetch-and-Add (FAA)



Compare-and-Swap (CAS)

Atomics: Performance Dimensions

! Performance metrics?

Atomics: Performance Dimensions

Latency

Performance metrics?

ATOMICS: PERFORMANCE DIMENSIONS

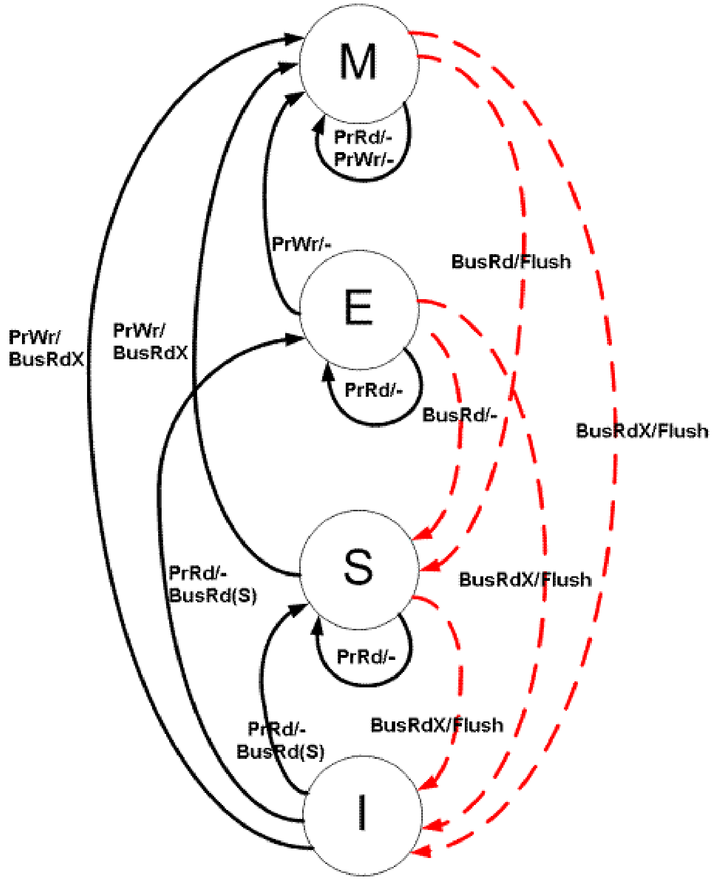
Latency

Performance metrics?

Bandwidth

ATOMICS: PERFORMANCE DIMENSIONS

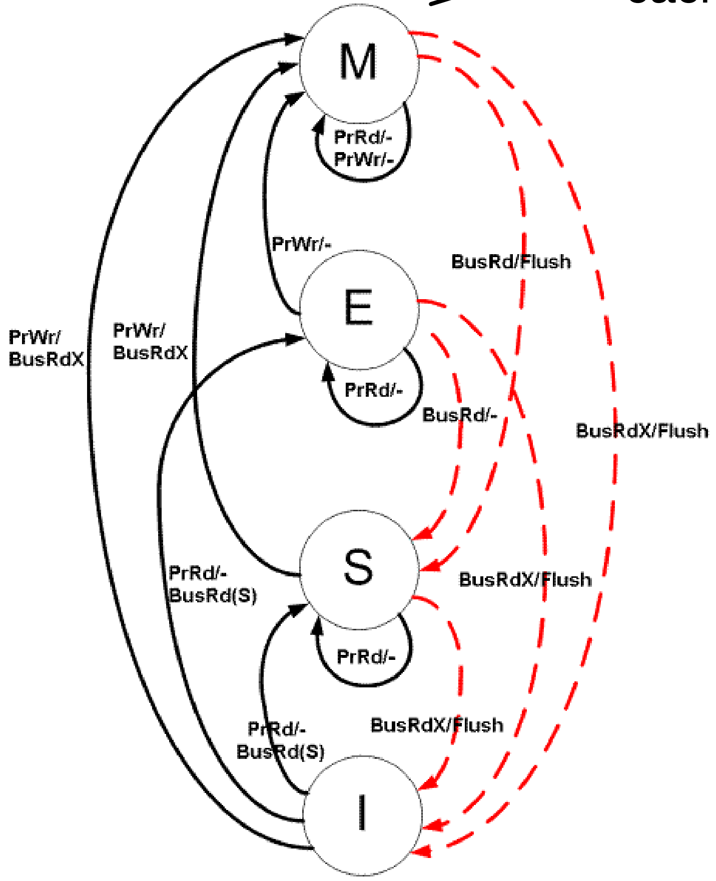
! Cache coherence state?



Atomics: Performance Dimensions

Cache coherence state?

Modified: in one cache and dirty

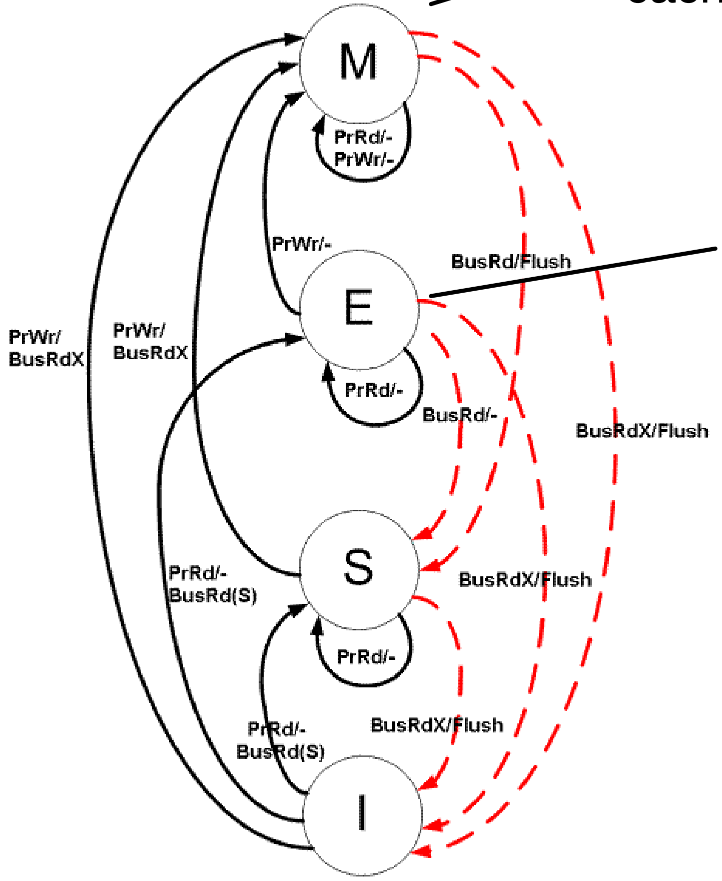


Atomics: Performance Dimensions

Cache coherence state?

Modified: in one cache and dirty

Exclusive: in one cache and clean



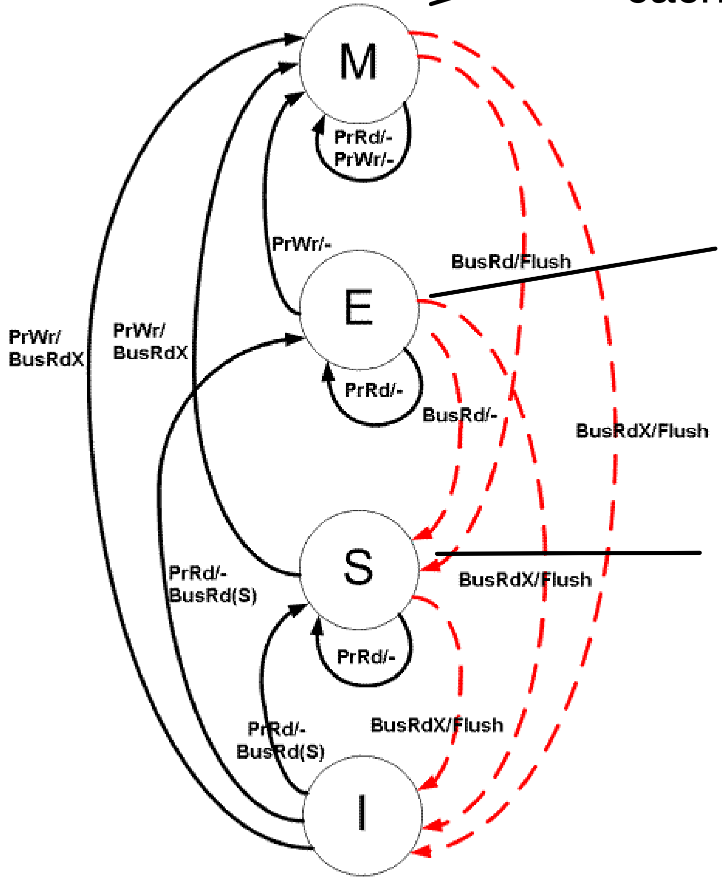
Atomics: Performance Dimensions

Cache coherence state?

Modified: in one cache and dirty

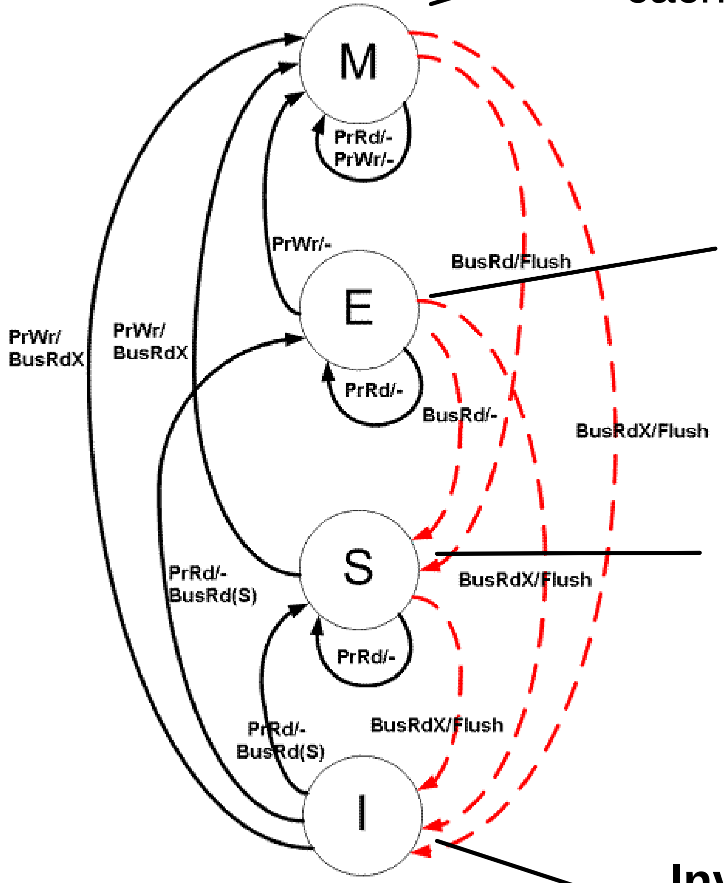
Exclusive: in one cache and clean

Shared: in >1 cache and clean



ATOMICS: PERFORMANCE DIMENSIONS

Cache coherence state?



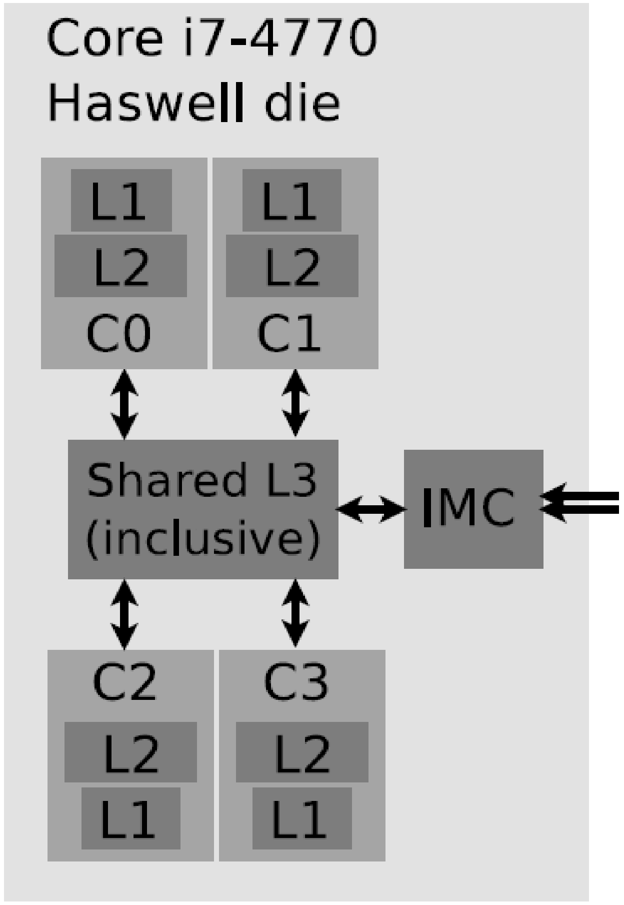
Modified: in one cache and dirty

Exclusive: in one cache and clean

Shared: in >1 cache and clean

Invalid: garbage data

Atomics: Performance Dimensions

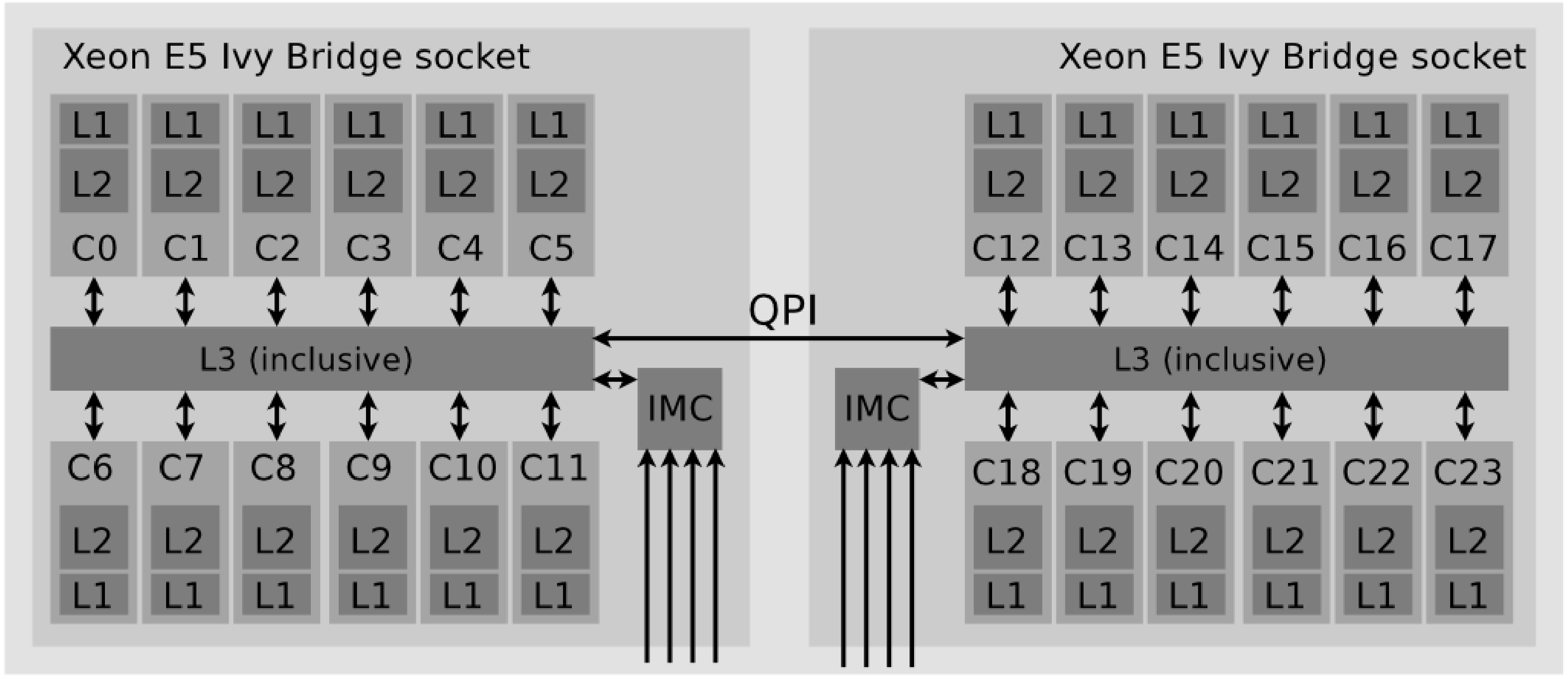


Architecture

ATOMICS: PERFORMANCE DIMENSIONS

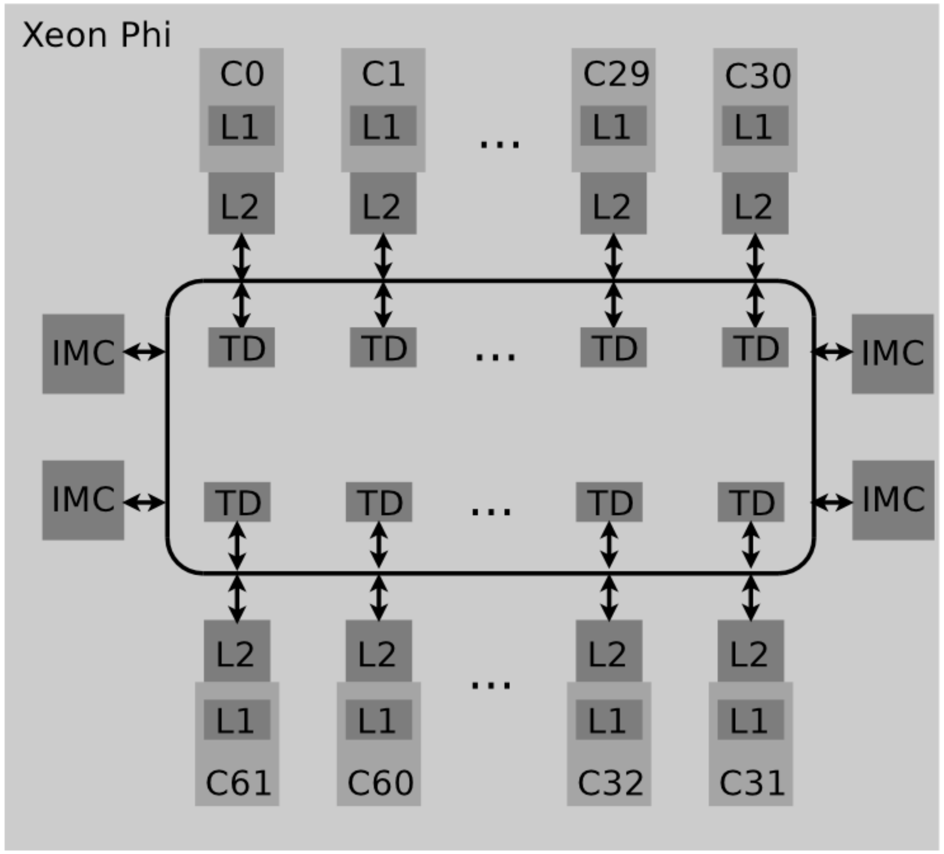
Architecture

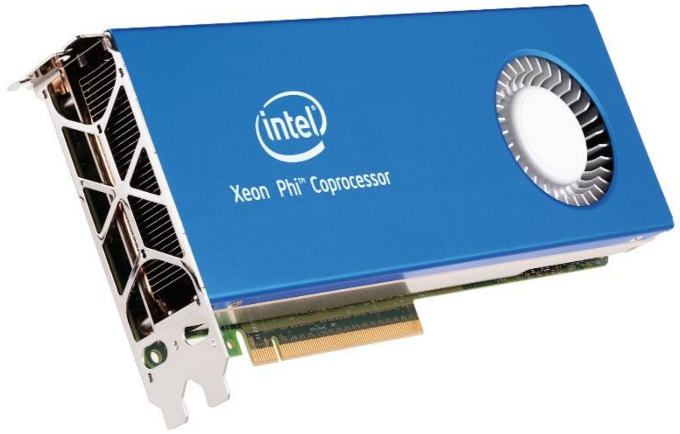
Atomics: Performance Dimensions



Architecture

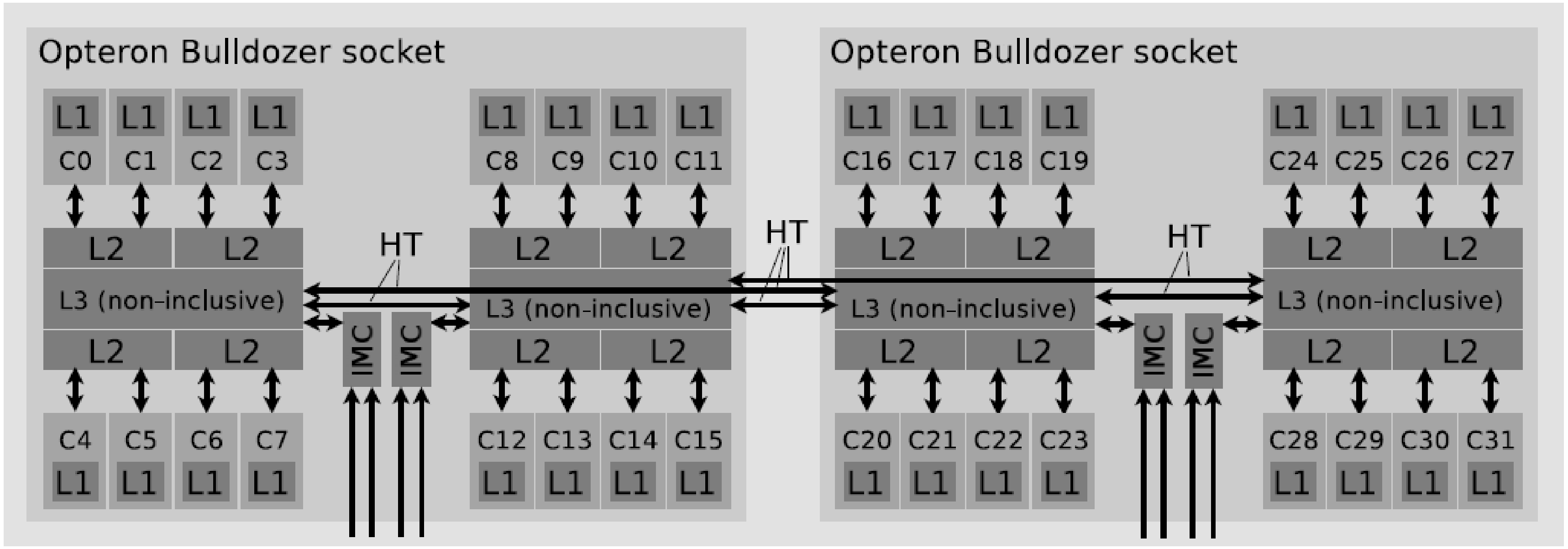
Atomics: Performance Dimensions





Architecture

Atomics: Performance Dimensions



Architecture

RESEARCH QUESTIONS

Research Questions

How do we model the performance of atomics?

Research Questions

How do we model the performance of atomics?

What is the performance difference between various atomics?

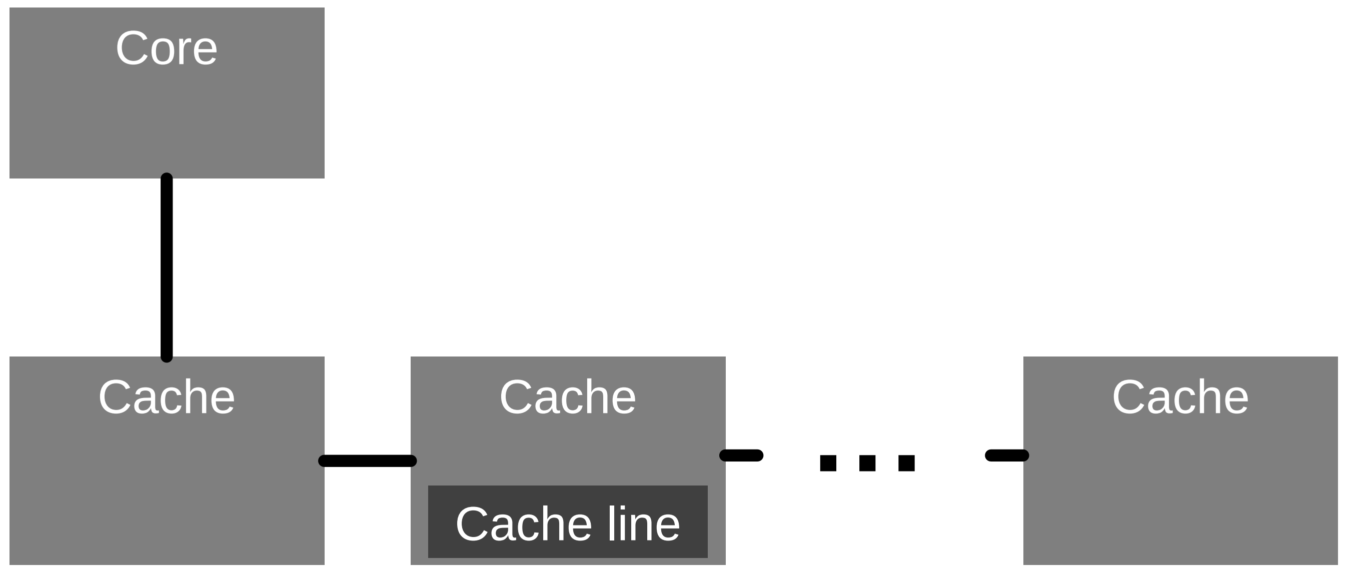
Research Questions

How do we model the performance of atomics?

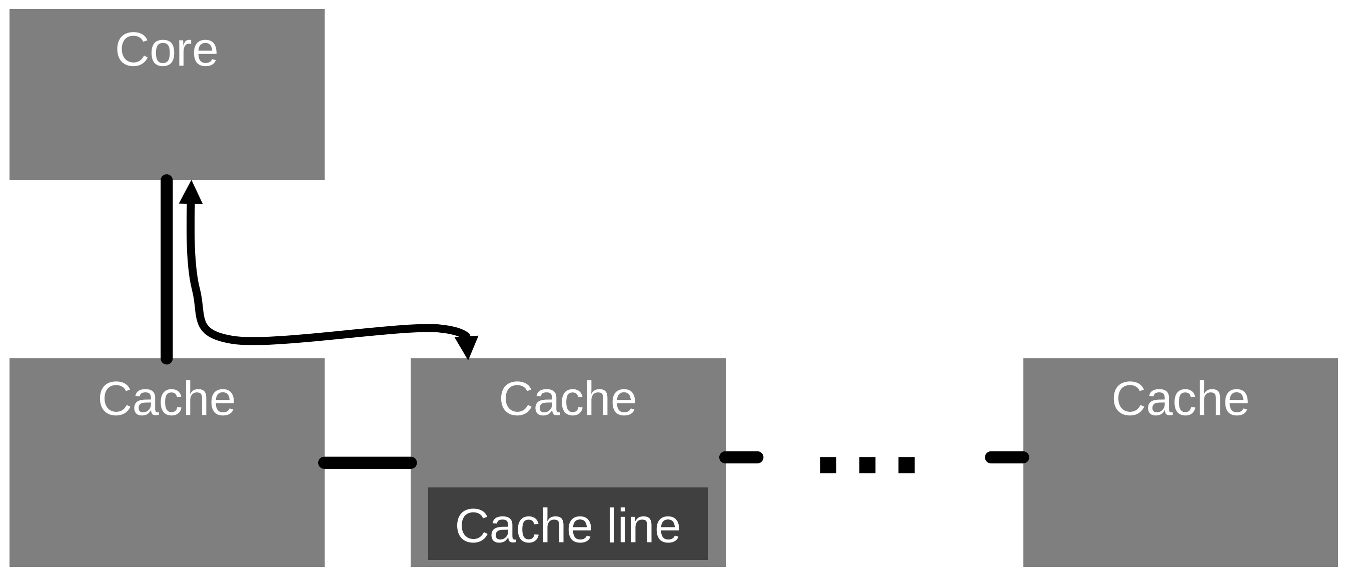
What is the performance difference between various atomics?

What is the influence of various parameters and mechanisms?

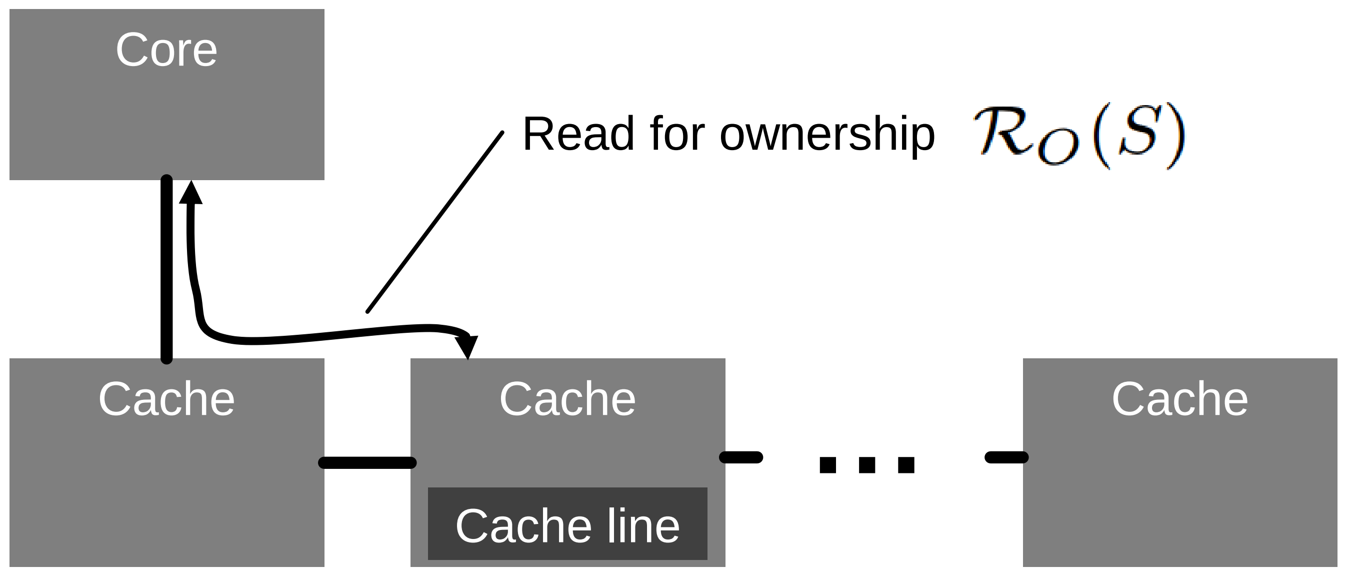
LATENCY MODEL



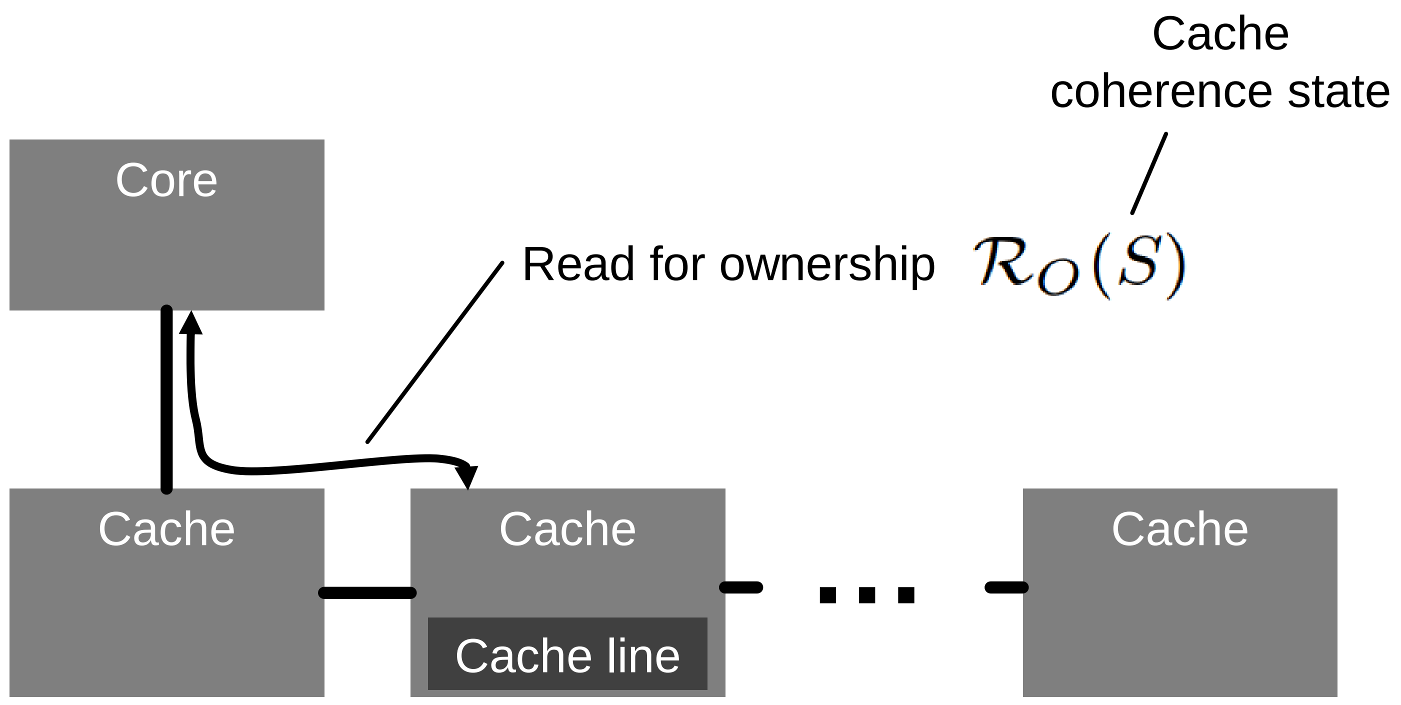
Latency Model



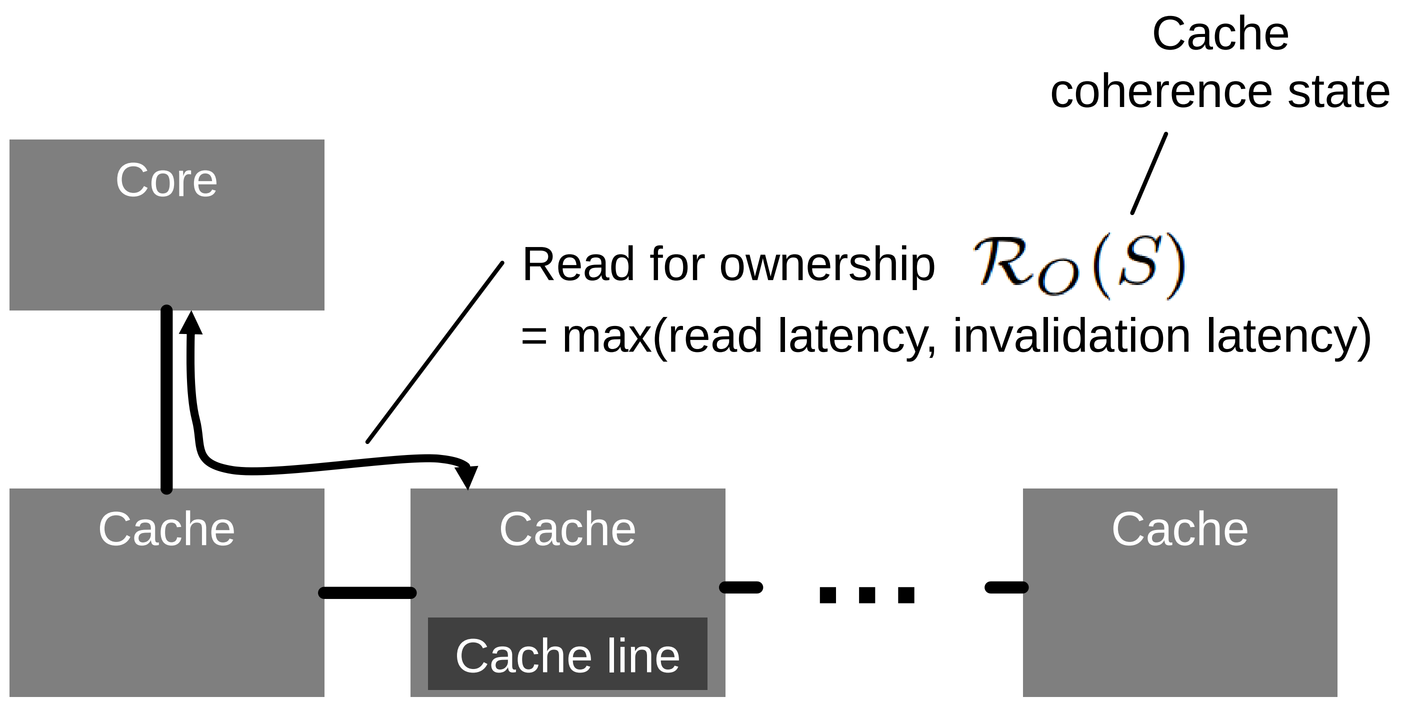
Latency Model



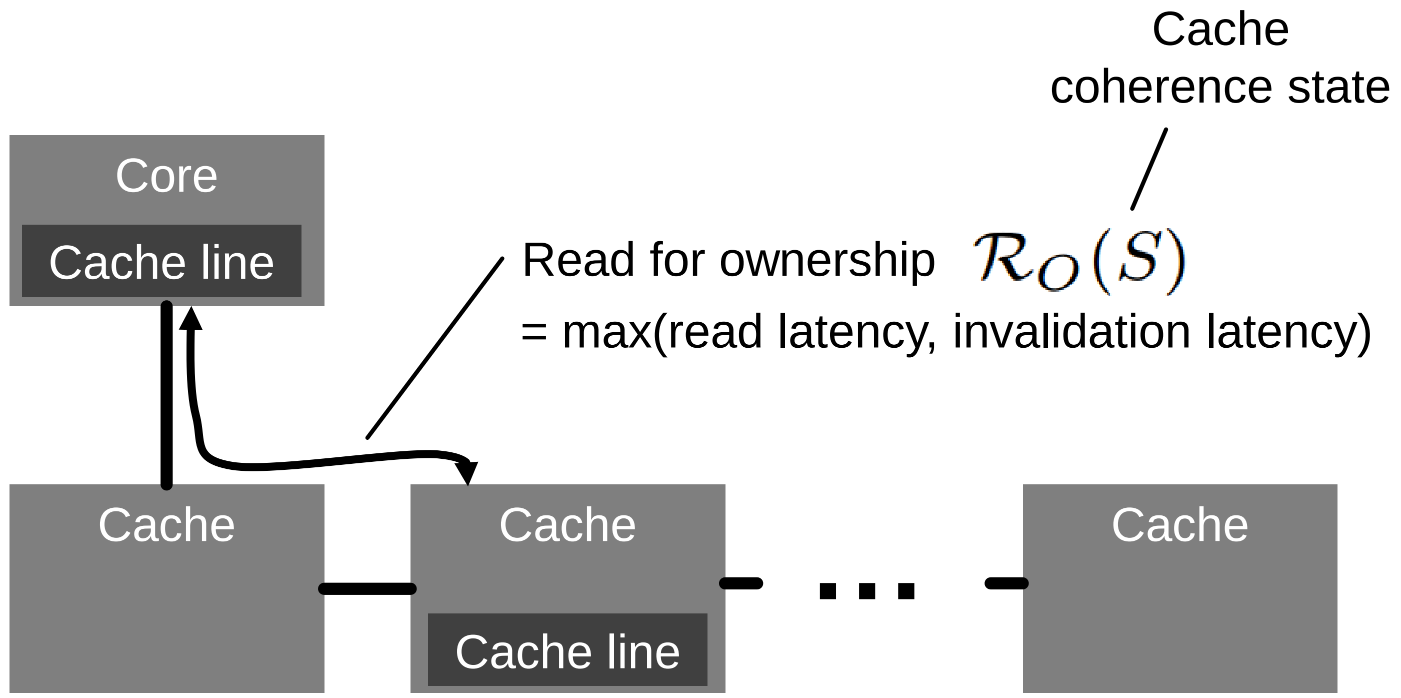
Latency Model



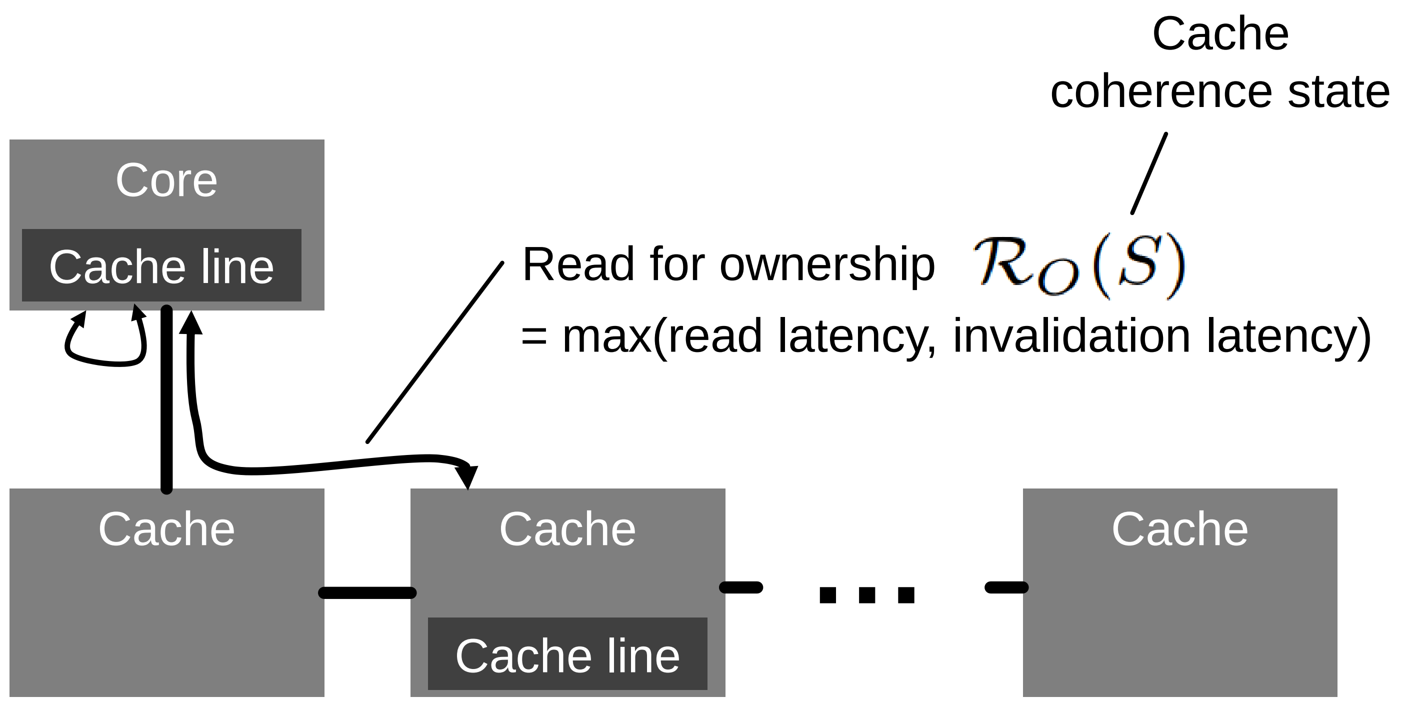
Latency Model


Cache
coherence state
Core
Read for ownership $\mathcal{R}_O(S)$
= max(read latency, invalidation latency)
Cache
Cache
Cache line
...
Cache

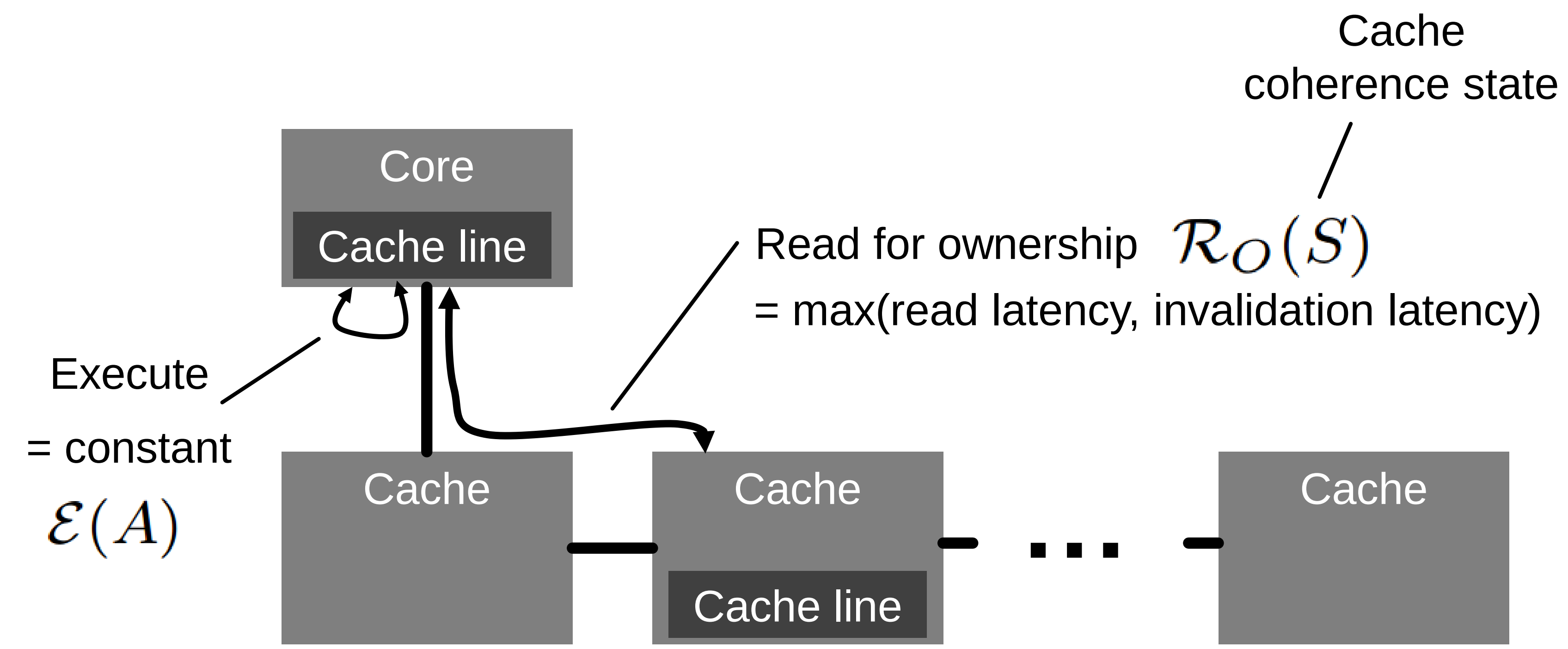
Latency Model


Cache
coherence state
Core
Cache line
Read for ownership $\mathcal{R}_O(S)$
= max(read latency, invalidation latency)
Cache
Cache
Cache line
…
Cache

LATENCY MODEL


Cache
coherence state
Core
Cache line
Read for ownership $\mathcal{R}_O(S)$
= max(read latency, invalidation latency)
Cache
Cache
Cache line
...
Cache

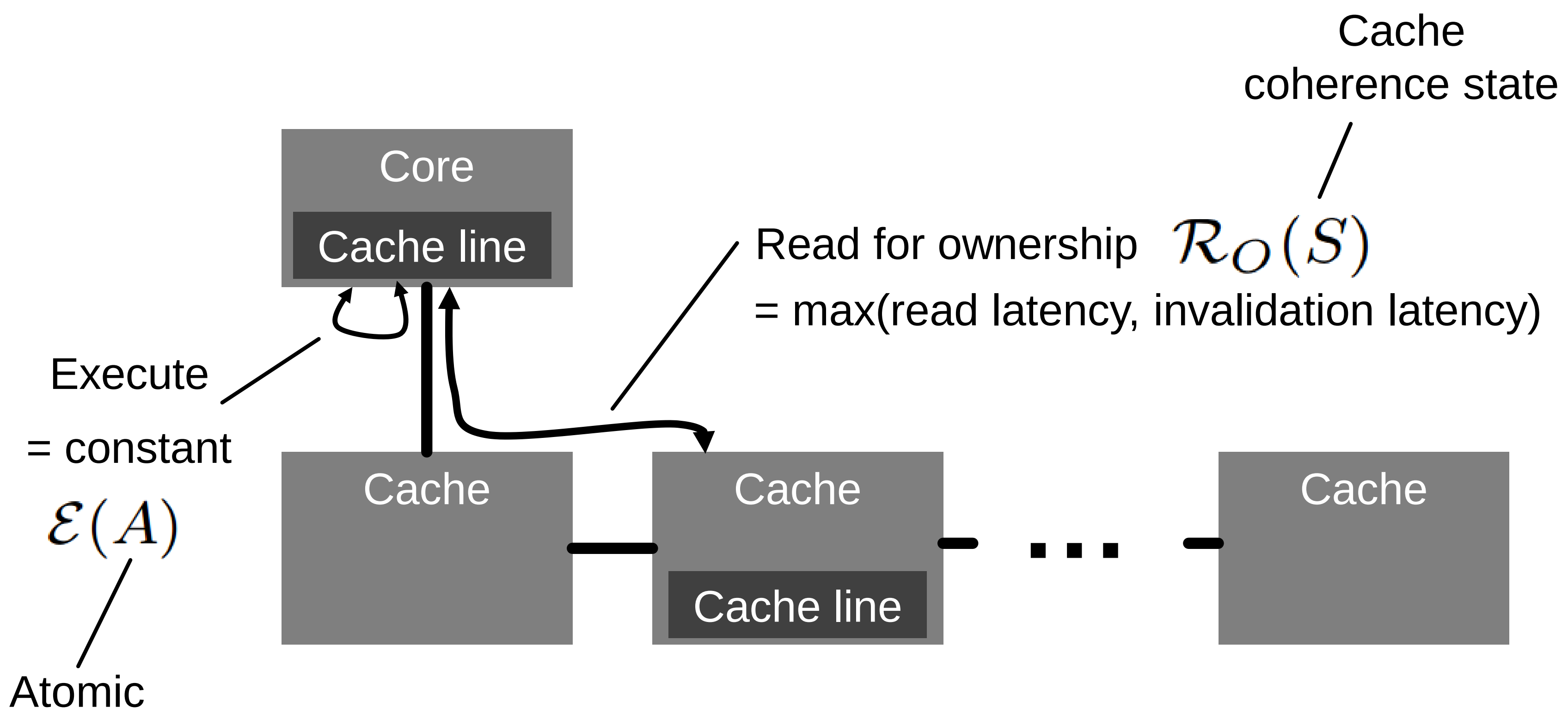
Latency Model



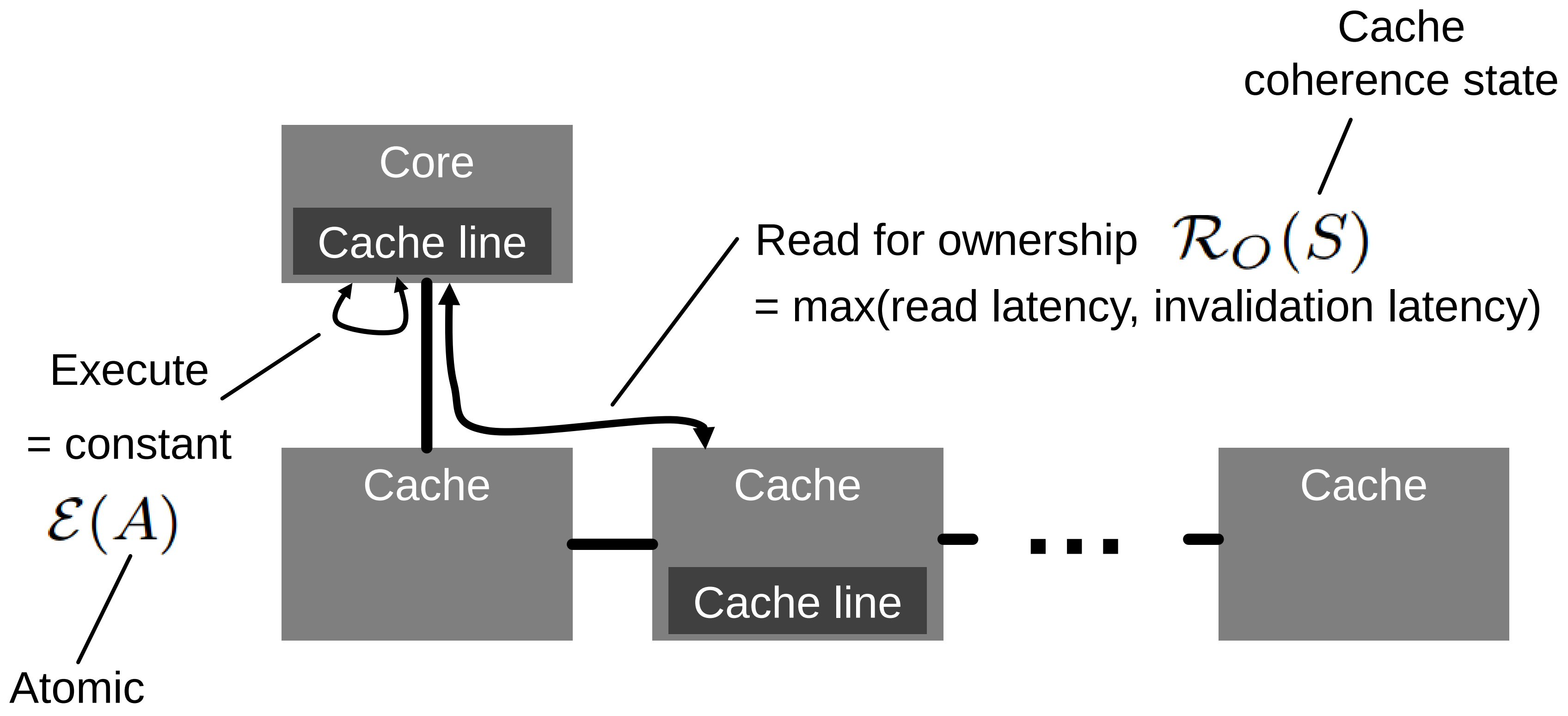
Read for ownership $\mathcal{R}_O(S)$
= max(read latency, invalidation latency)

Execute
= constant
$\mathcal{E}(A)$

Latency Model

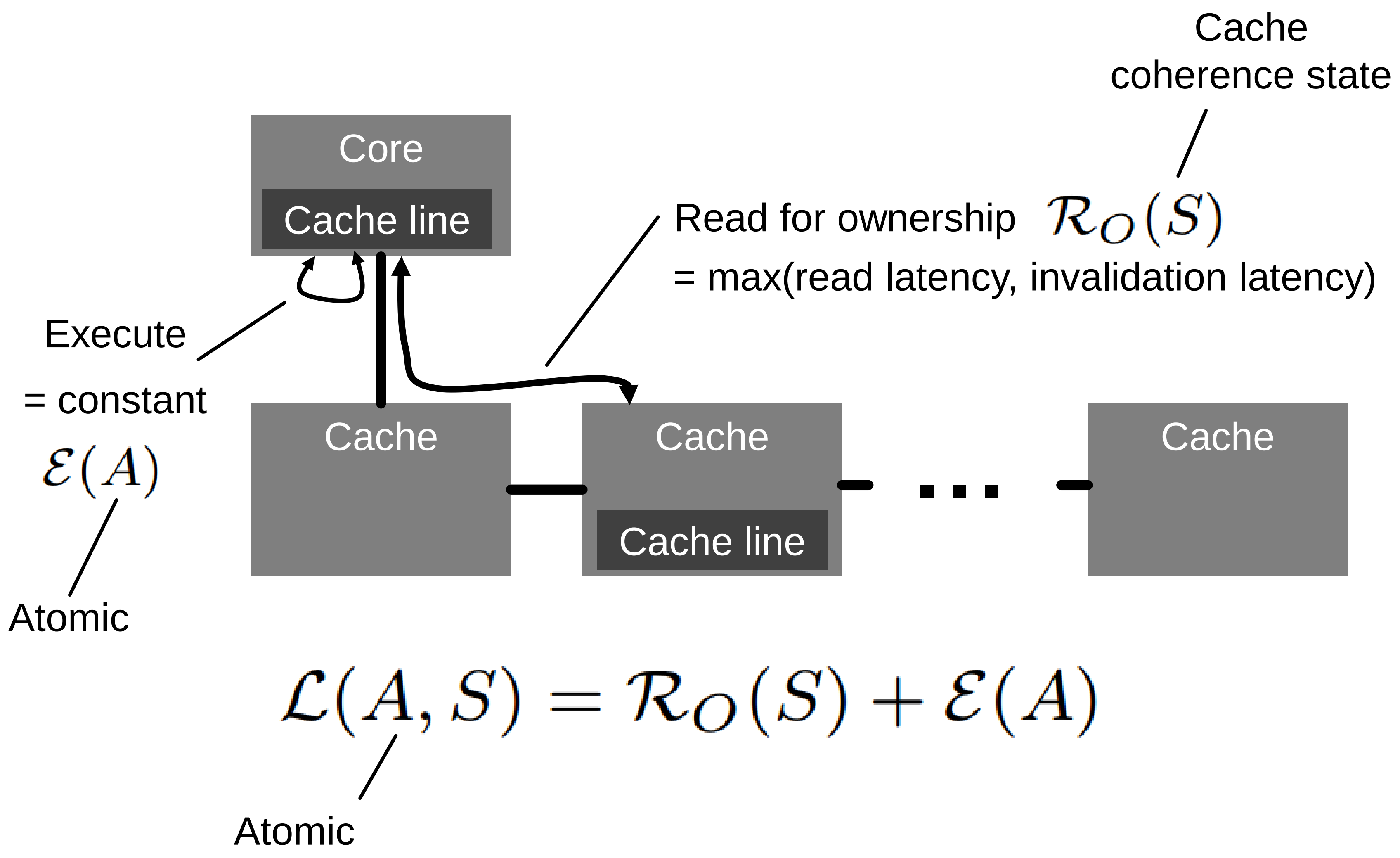


Latency Model



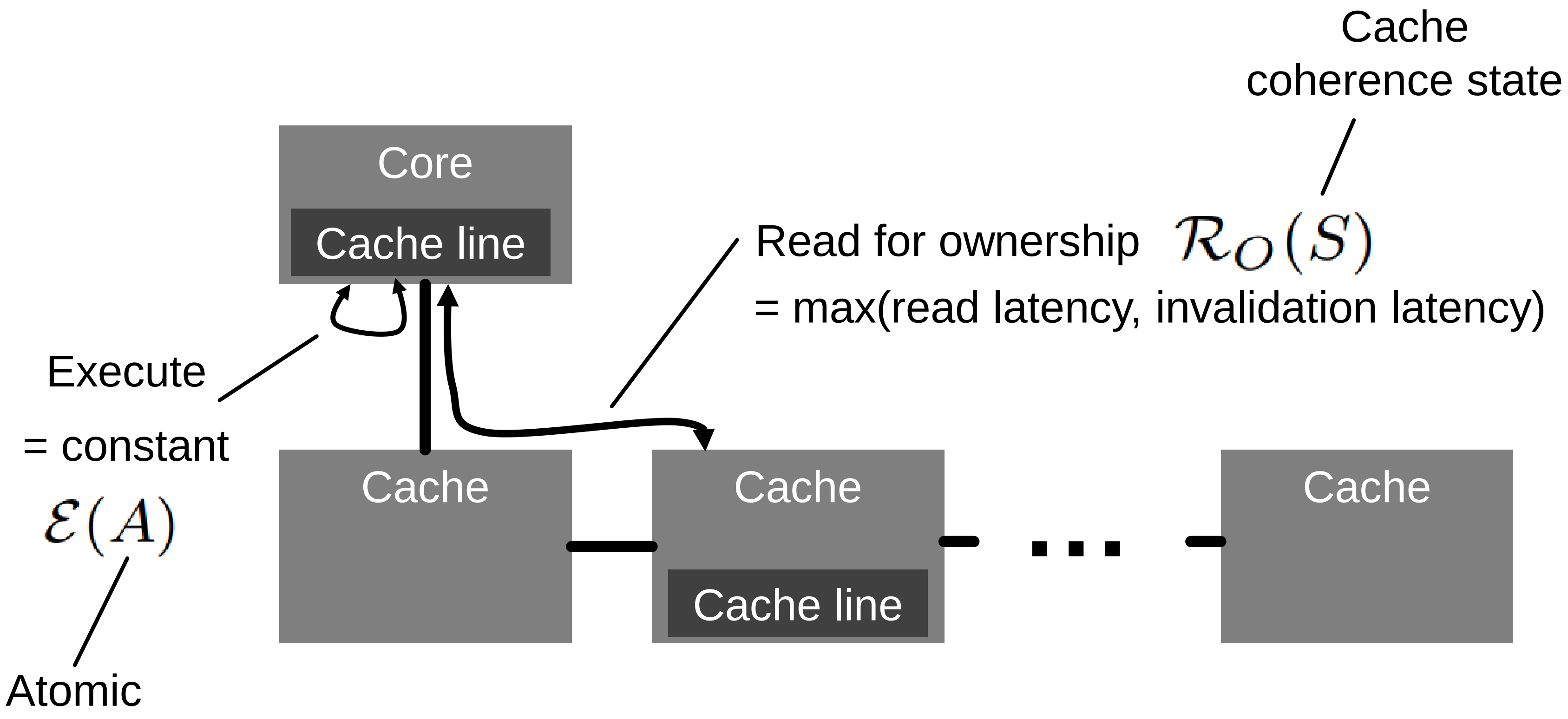
$$\mathcal{L}(A,S) = \mathcal{R}_O(S) + \mathcal{E}(A)$$

Latency Model



$$\mathcal{L}(A, S) = \mathcal{R}_O(S) + \mathcal{E}(A)$$

Latency Model

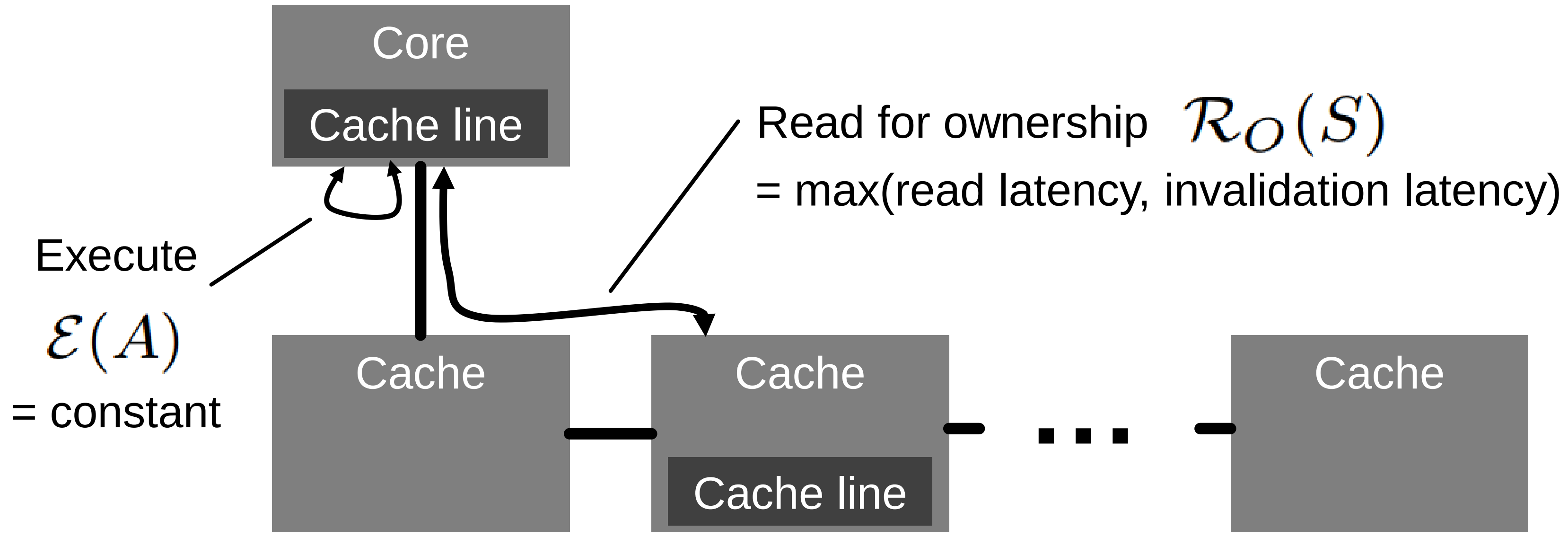


$$\mathcal{L}(A, S) = \mathcal{R}_O(S) + \mathcal{E}(A)$$

Atomic

Cache coherence state

Latency Model
Exclusive or Modified State


Core
Cache line
Execute
$\mathcal{E}(A)$
= constant
Read for ownership $\mathcal{R}_O(S)$
= max(read latency, invalidation latency)
Cache
Cache
Cache line
...
Cache

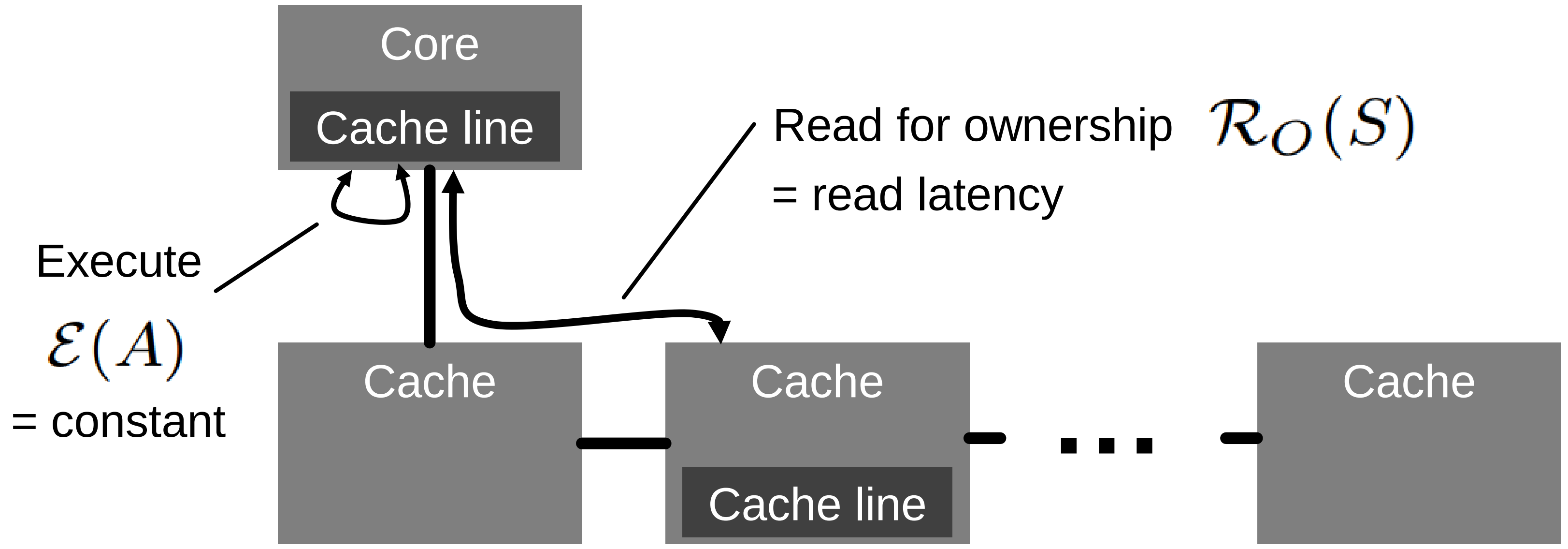
$$\mathcal{L}(A, S) = \mathcal{R}_O(S) + \mathcal{E}(A)$$

Atomic

Cache coherence state

Latency Model

Exclusive or Modified State



$$\mathcal{L}(A,S) = \mathcal{R}_O(S) + \mathcal{E}(A)$$

Atomic

Cache coherence state

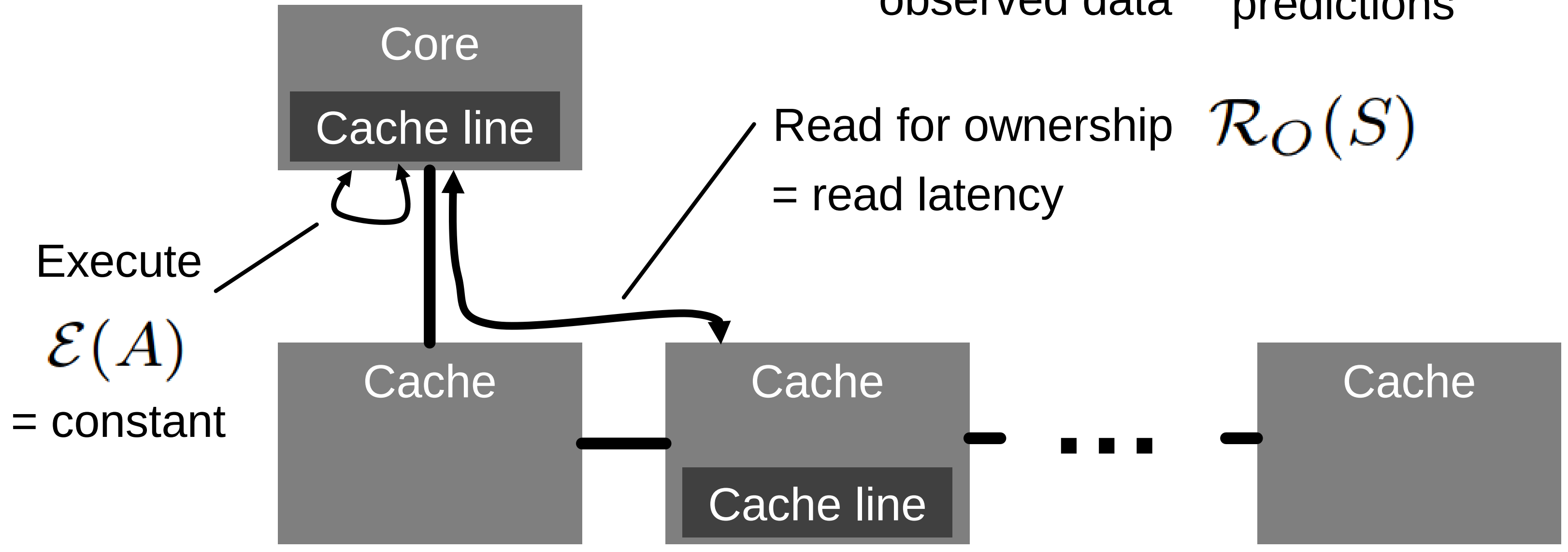
Latency Model
Exclusive or Modified State

$$\text{NRMSE} = \frac{1}{\bar{x}}\sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{x}_i - x_i)^2}$$

mean of observed data

predictions

observed data



Read for ownership $\mathcal{R}_O(S)$

= read latency

Execute

$\mathcal{E}(A)$

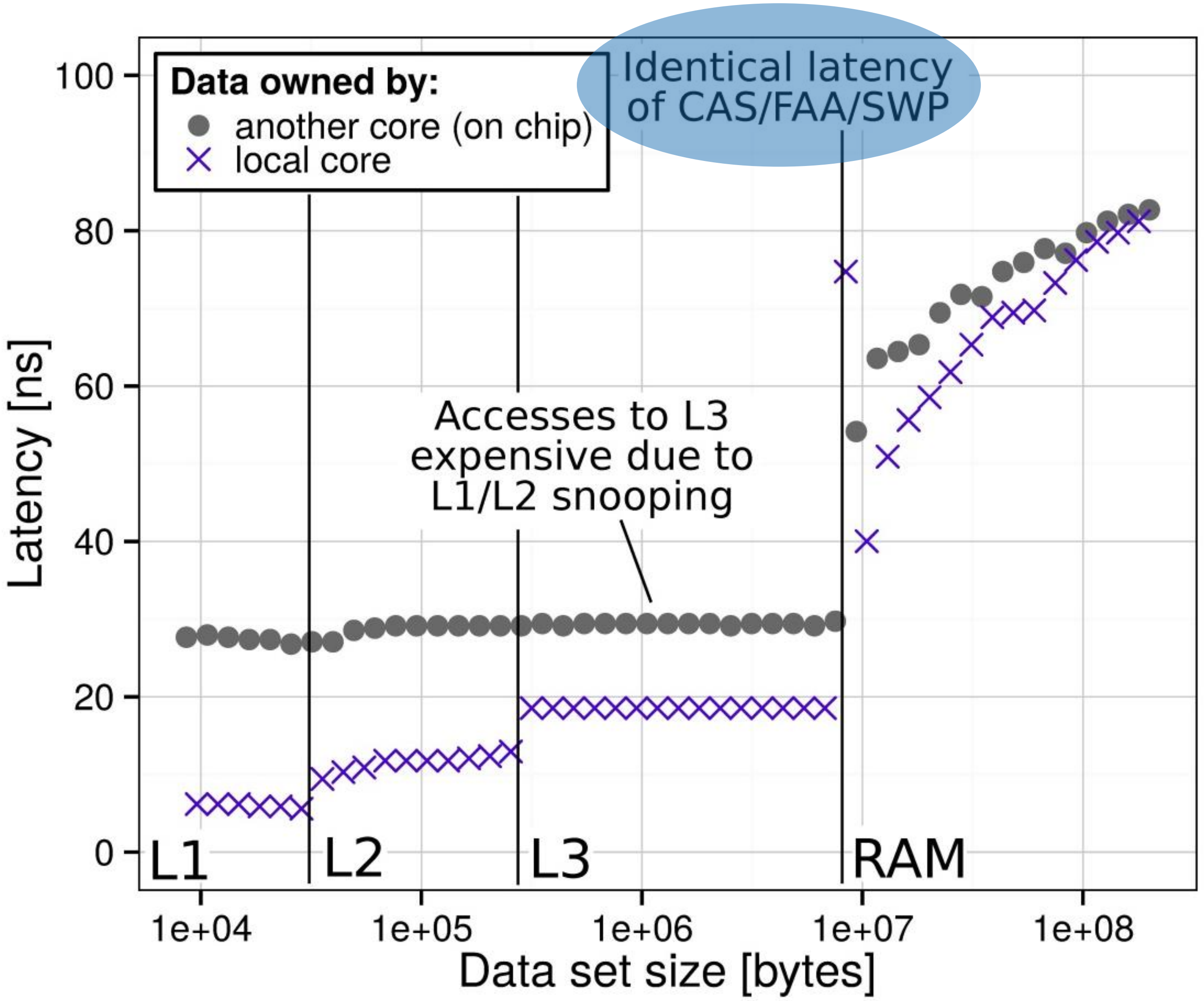
= constant

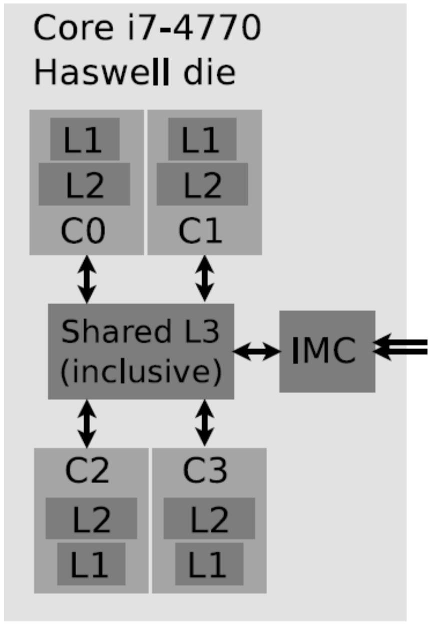
$$\mathcal{L}(A,S) = \mathcal{R}_O(S) + \mathcal{E}(A)$$

Atomic

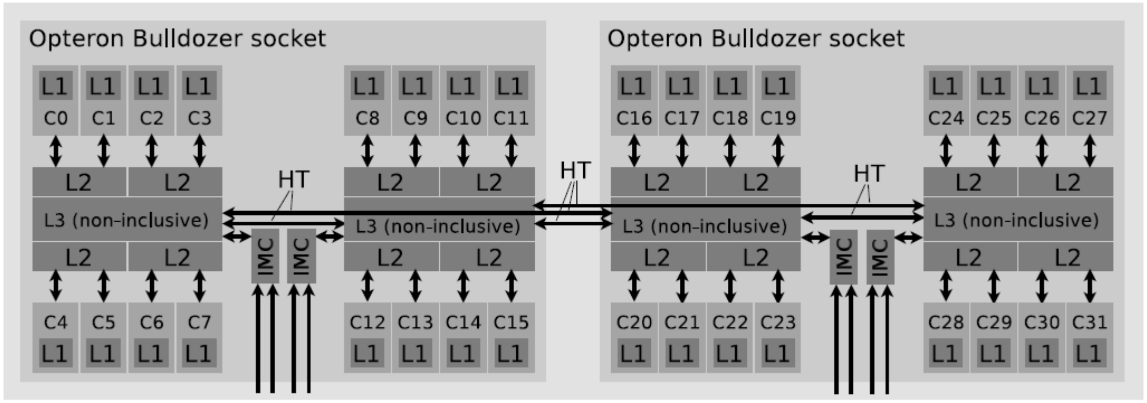
Cache coherence state

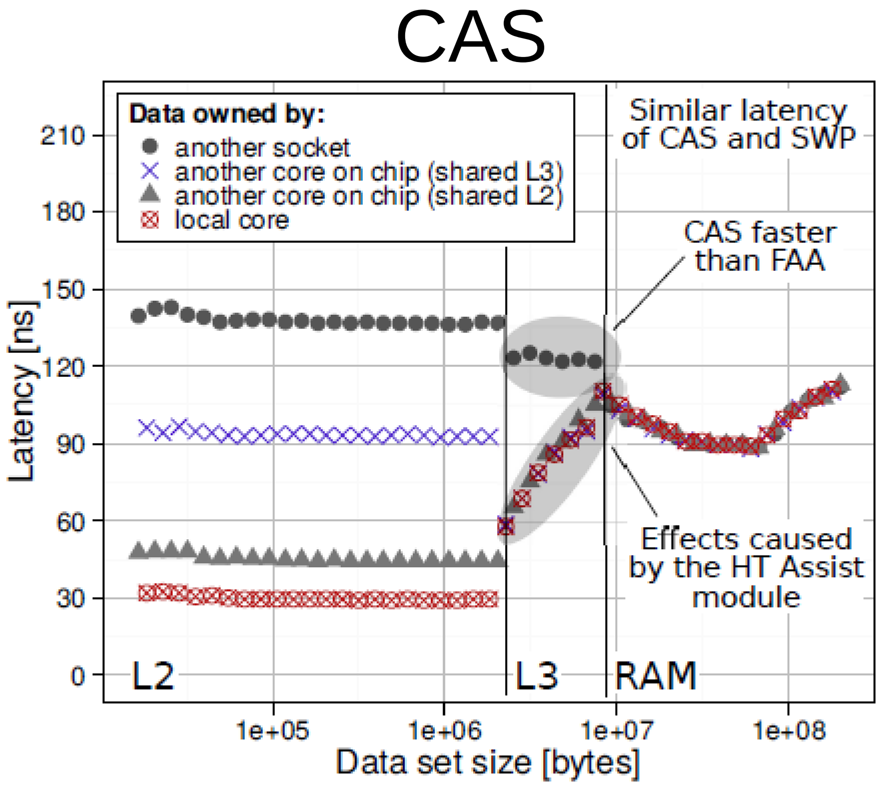
LATENCY
HASWELL, EXCLUSIVE

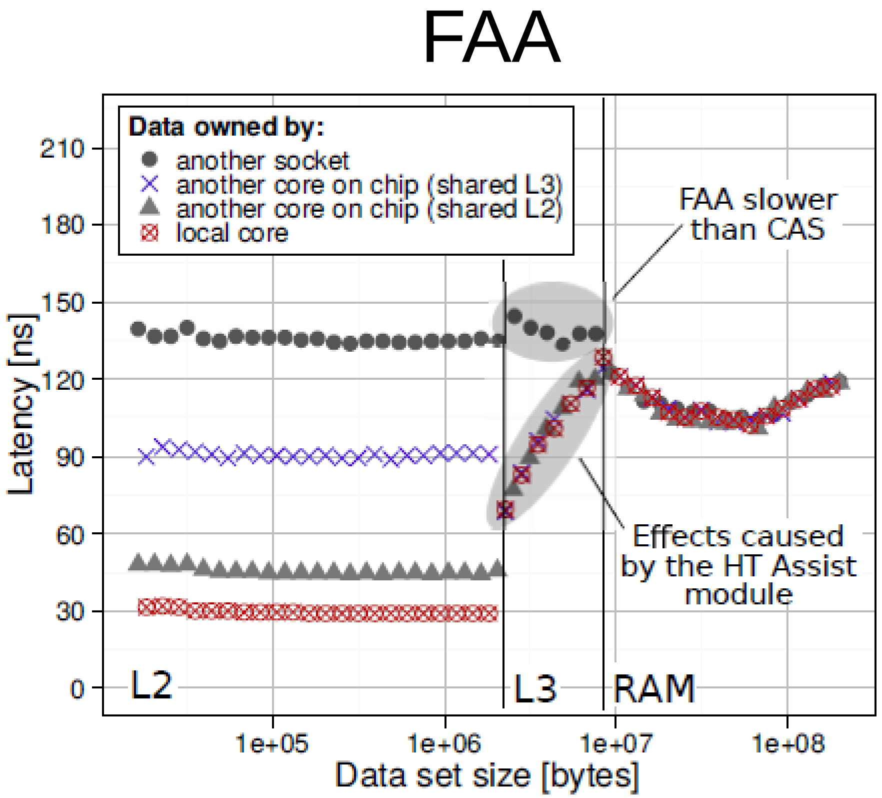




Latency
Bulldozer, Exclusive

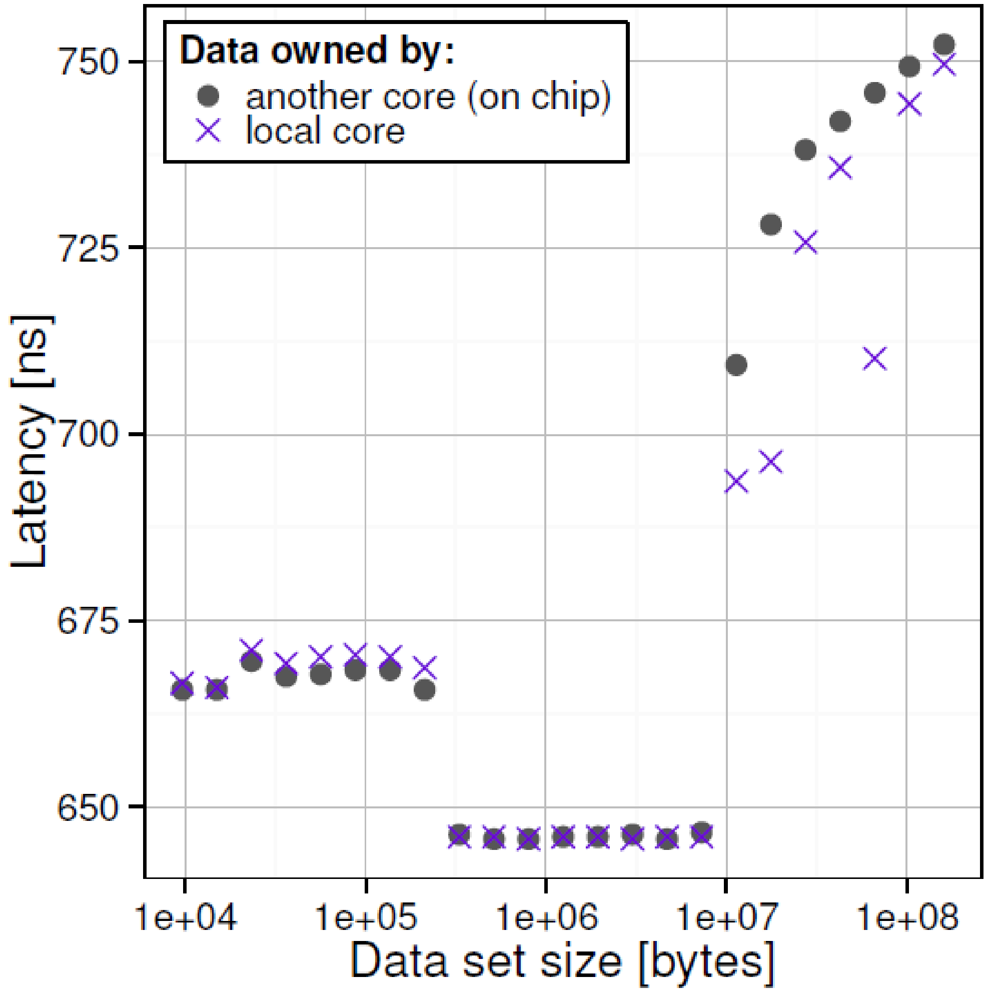

Opteron Bulldozer socket
Opteron Bulldozer socket
L1
C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 C10 C11 C12 C13 C14 C15 C16 C17 C18 C19 C20 C21 C22 C23 C24 C25 C26 C27 C28 C29 C30 C31
L2
L3 (non-inclusive)
HT
IMC





Latency
Haswell, Exclusive

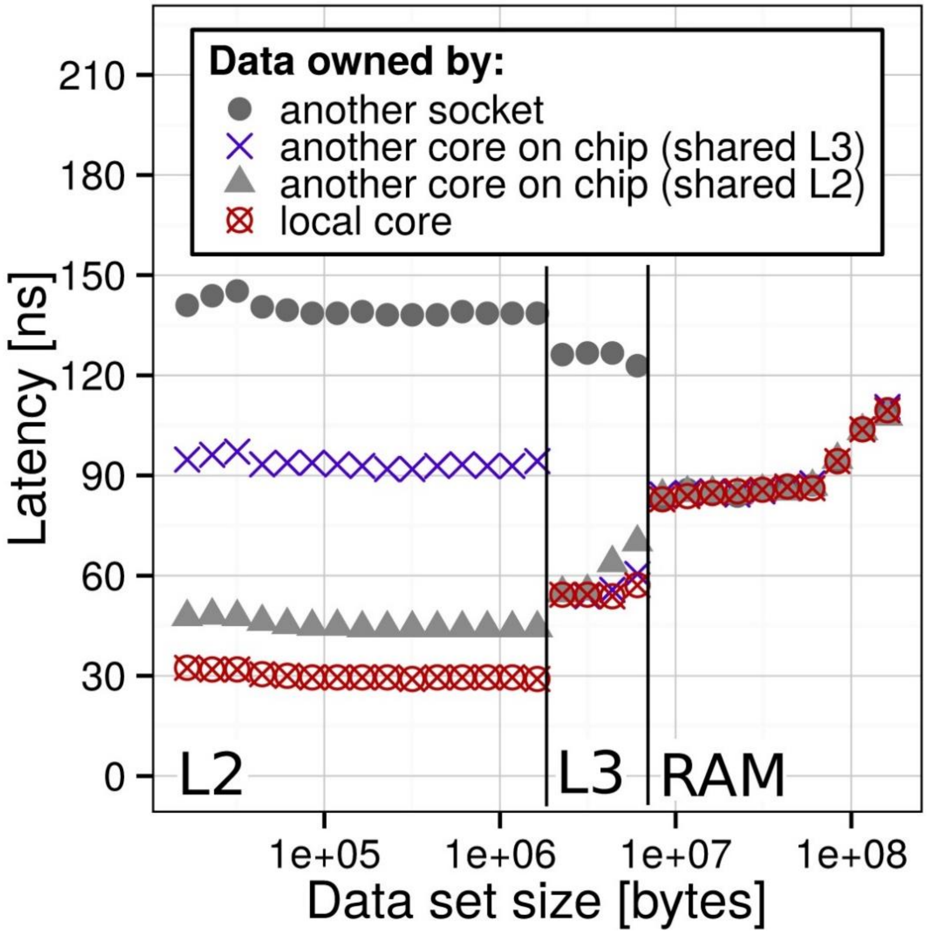
Alignment?



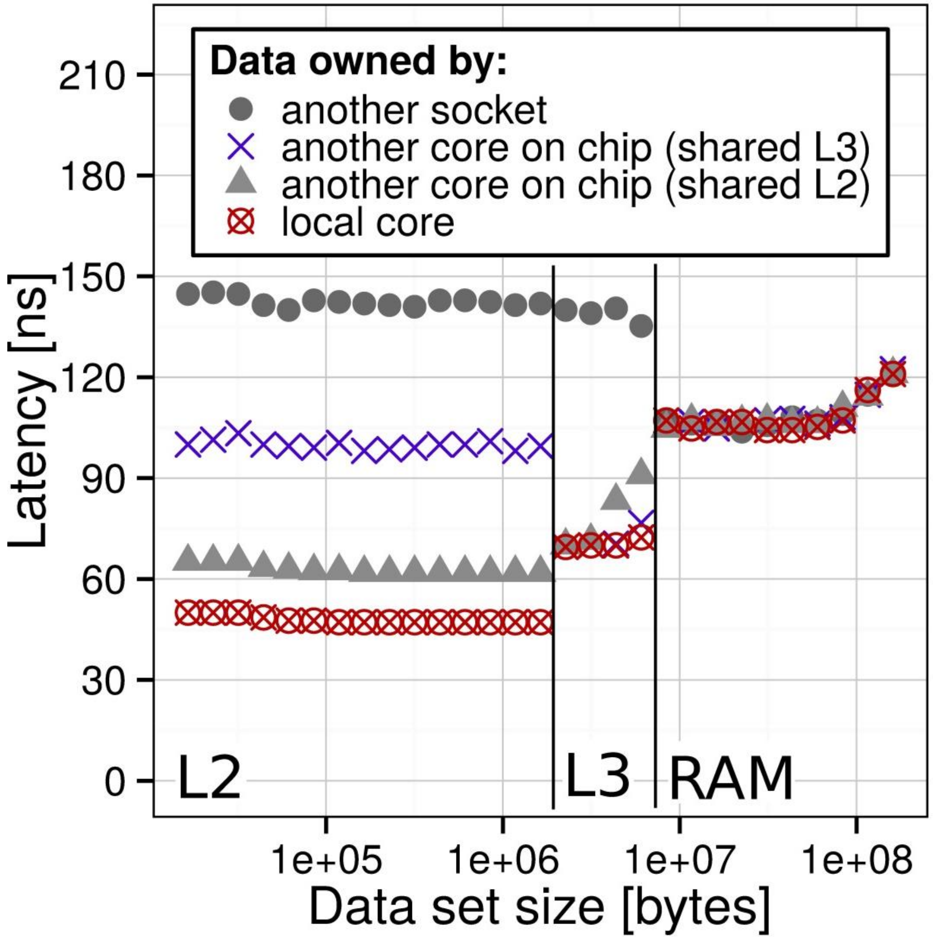
Latency
Bulldozer, Exclusive

! Operand size?

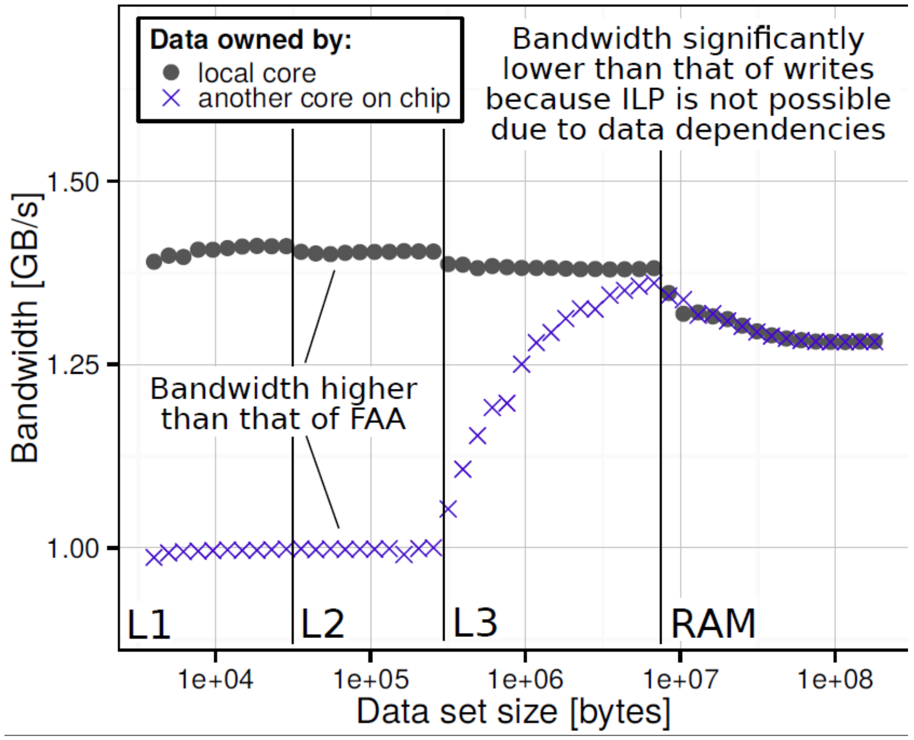
64 bit

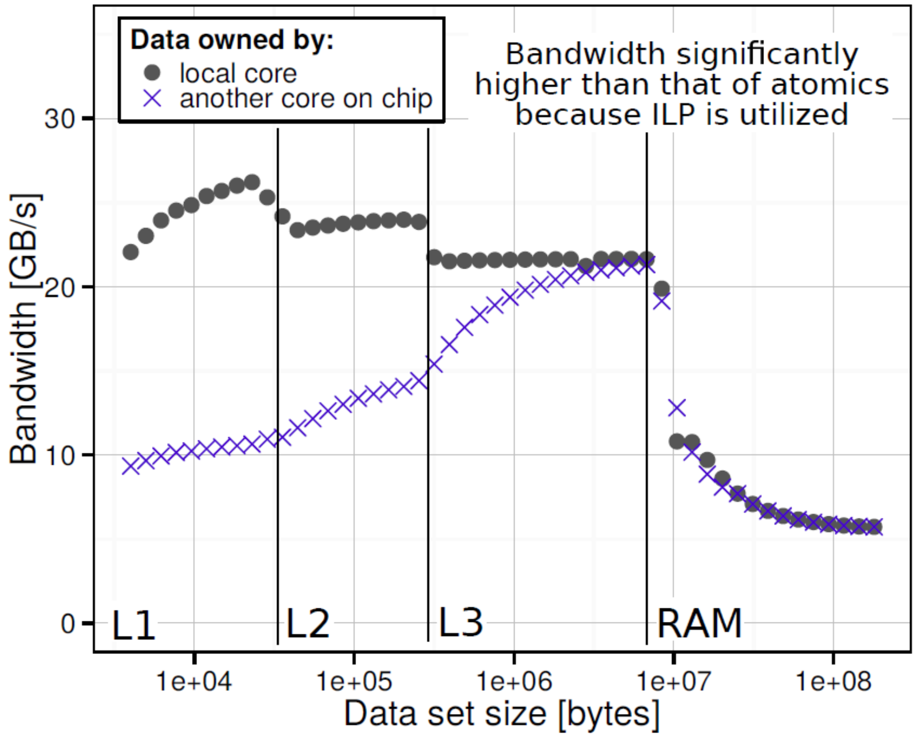


128 bit



Bandwidth
Haswell, Atomics





CONCLUSIONS
PERFORMANCE INSIGHTS

Conclusions

Performance Insights

The same latency of different atomics in most scenarios

Conclusions

Performance Insights

The same latency of different atomics in most scenarios

CAS is the fastest for some cases

Conclusions
Performance Insights

Unaligned atomics should be avoided at all costs

The same latency of different atomics in most scenarios

CAS is the fastest for some cases

Conclusions
Performance Insights

Unaligned atomics should be avoided at all costs

The same latency of different atomics in most scenarios

No parallel execution (low bandwidth) even if there are no data deps

CAS is the fastest for some cases

CONCLUSIONS
PERFORMANCE INSIGHTS

Unaligned atomics should be avoided at all costs

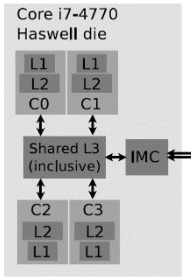
The same latency of different atomics in most scenarios

No parallel execution (low bandwidth) even if there are no data deps

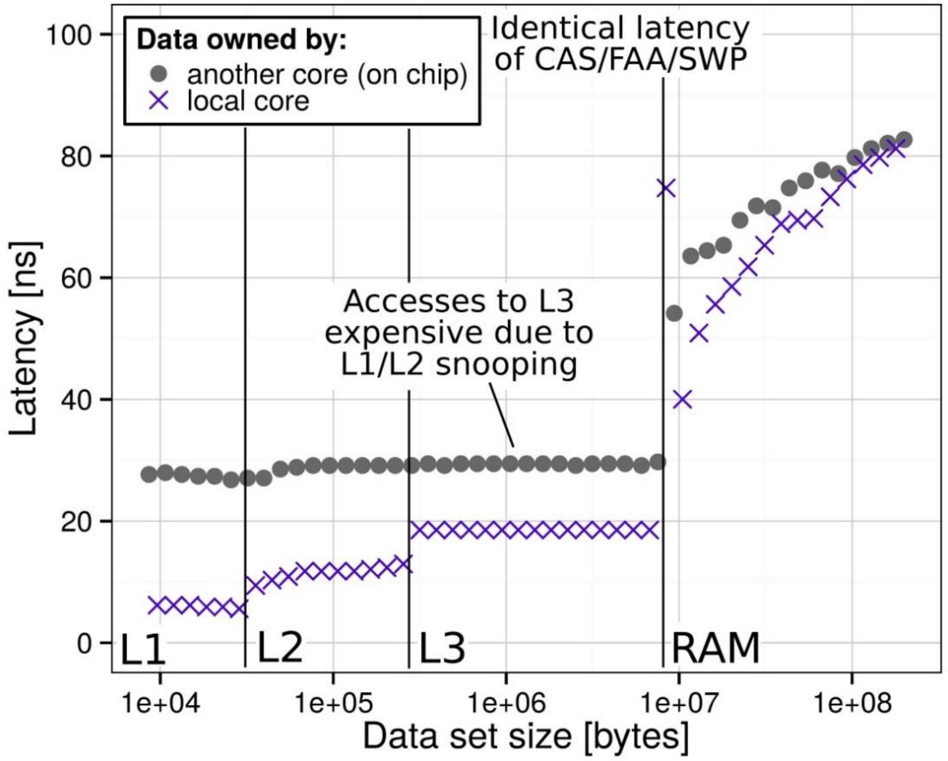
CAS is the fastest for some cases

Small operand sizes give best performance

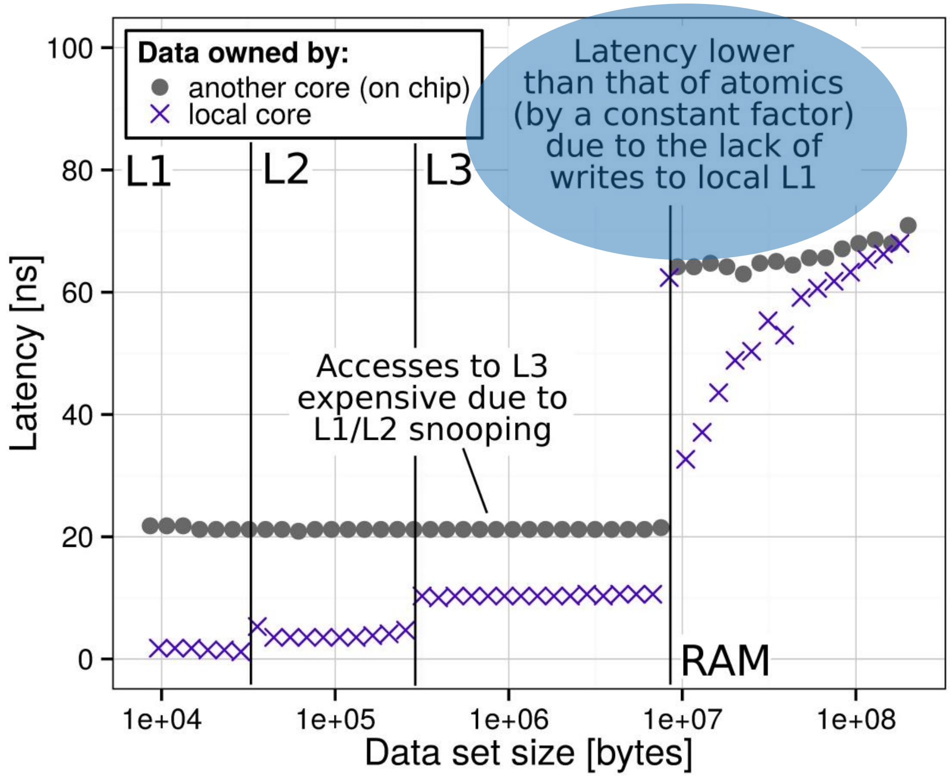
Latency
Haswell, Exclusive



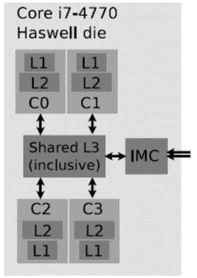
Atomics



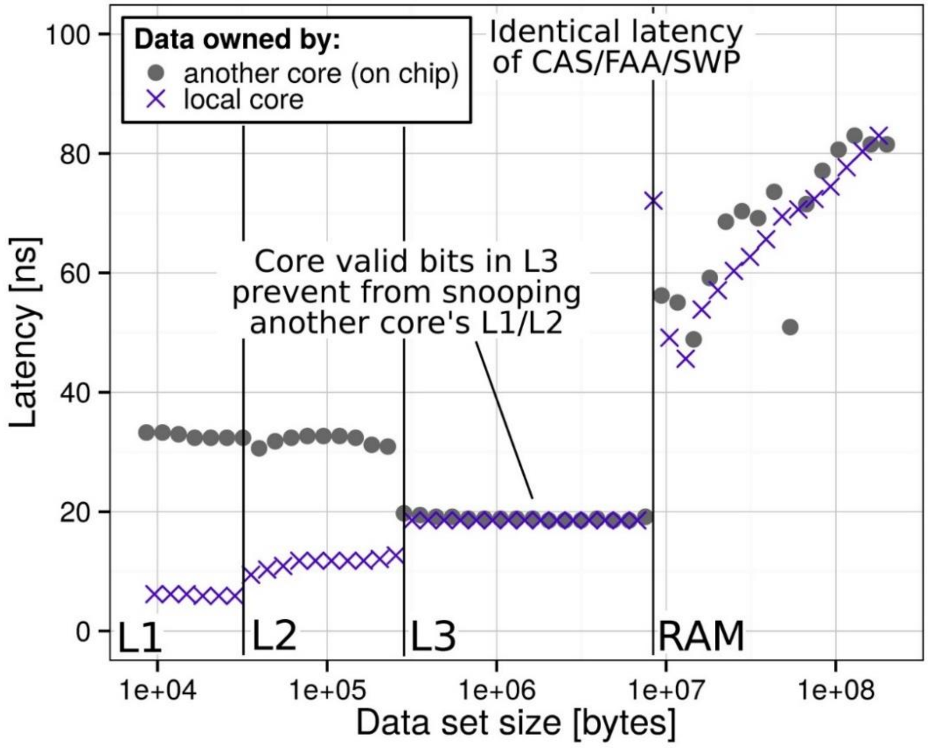
Read



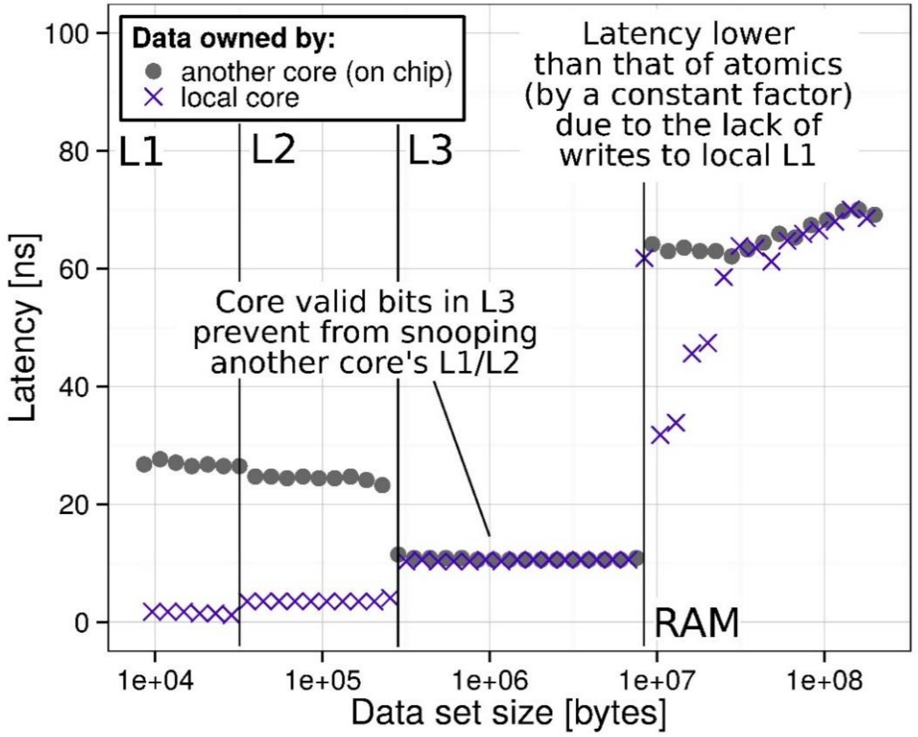
Latency
Haswell, Modified



Atomics

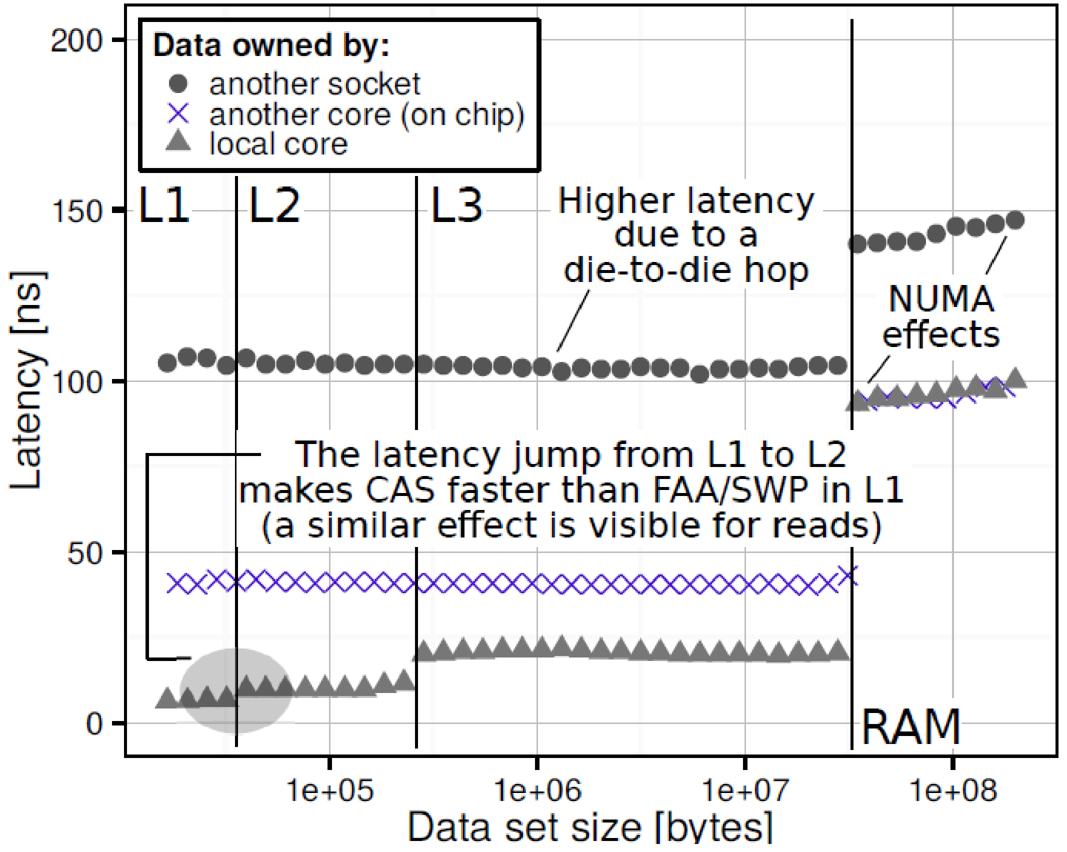


Read

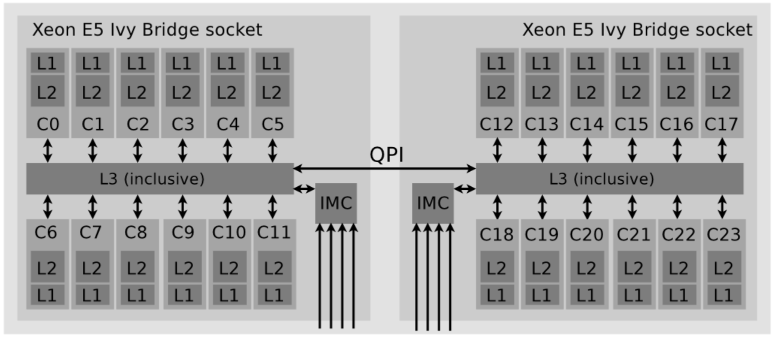


Latency
Ivy Bridge, Exclusive

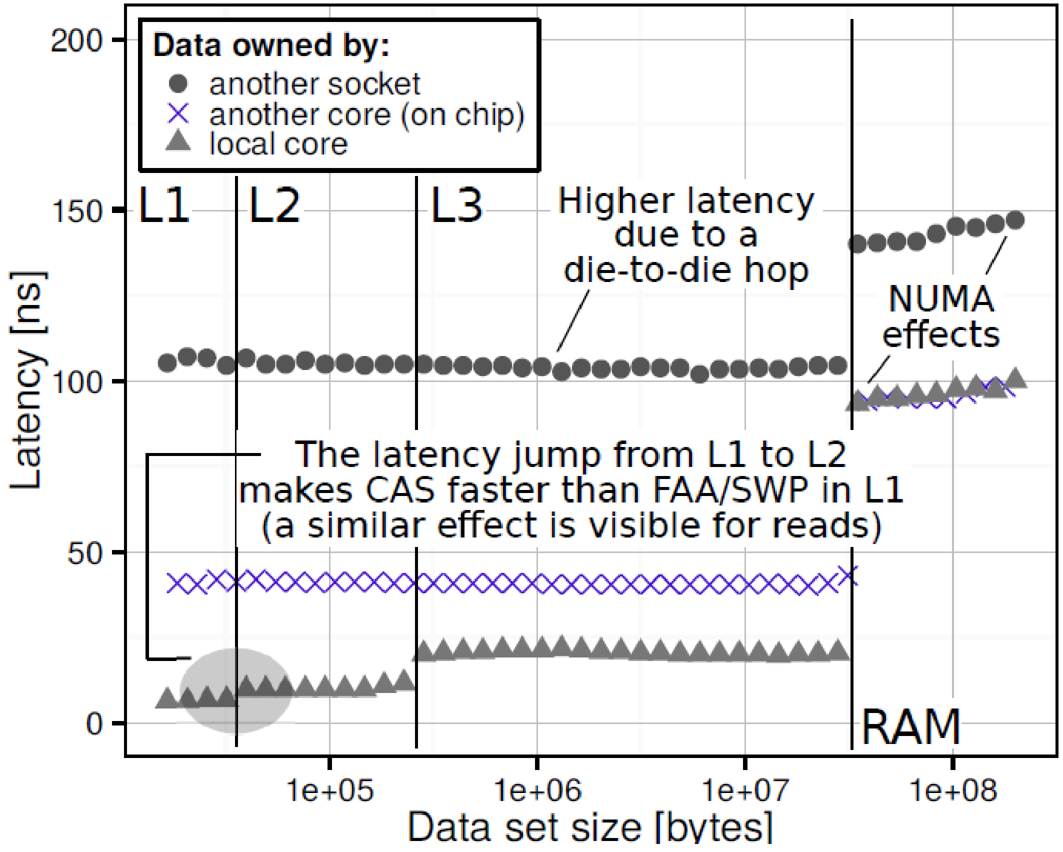
CAS



Latency
Ivy Bridge, Exclusive

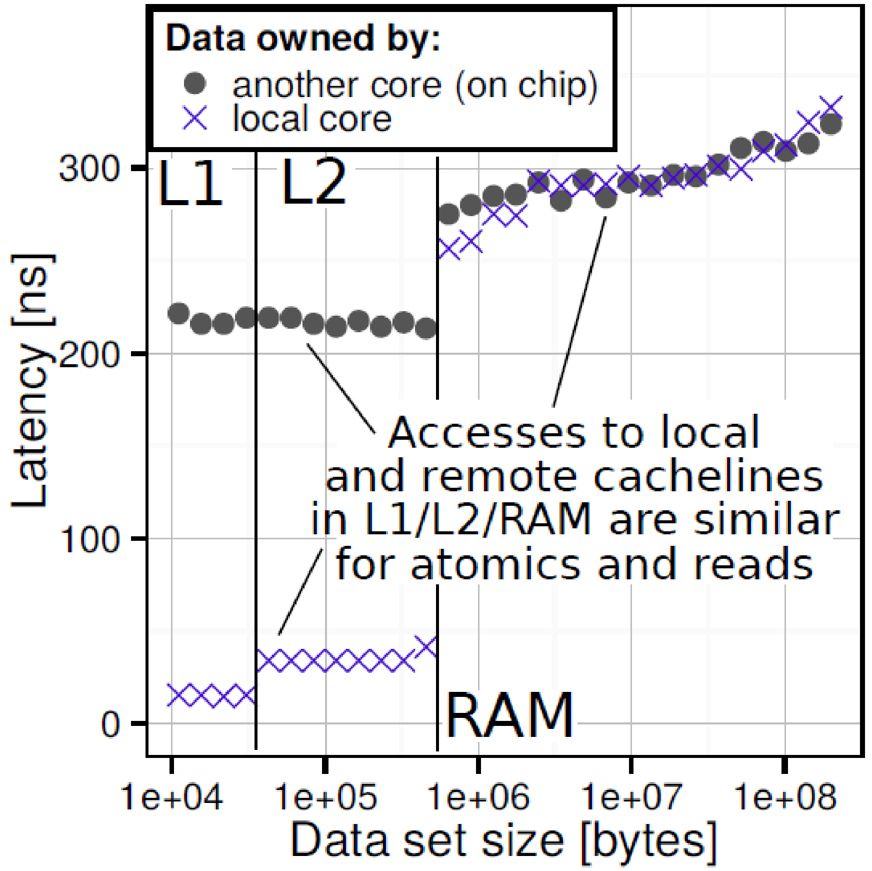


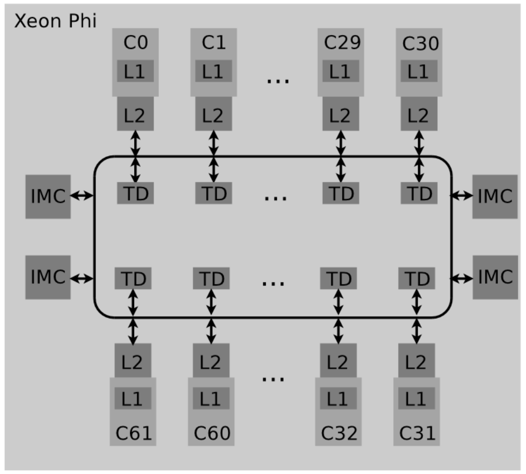
CAS



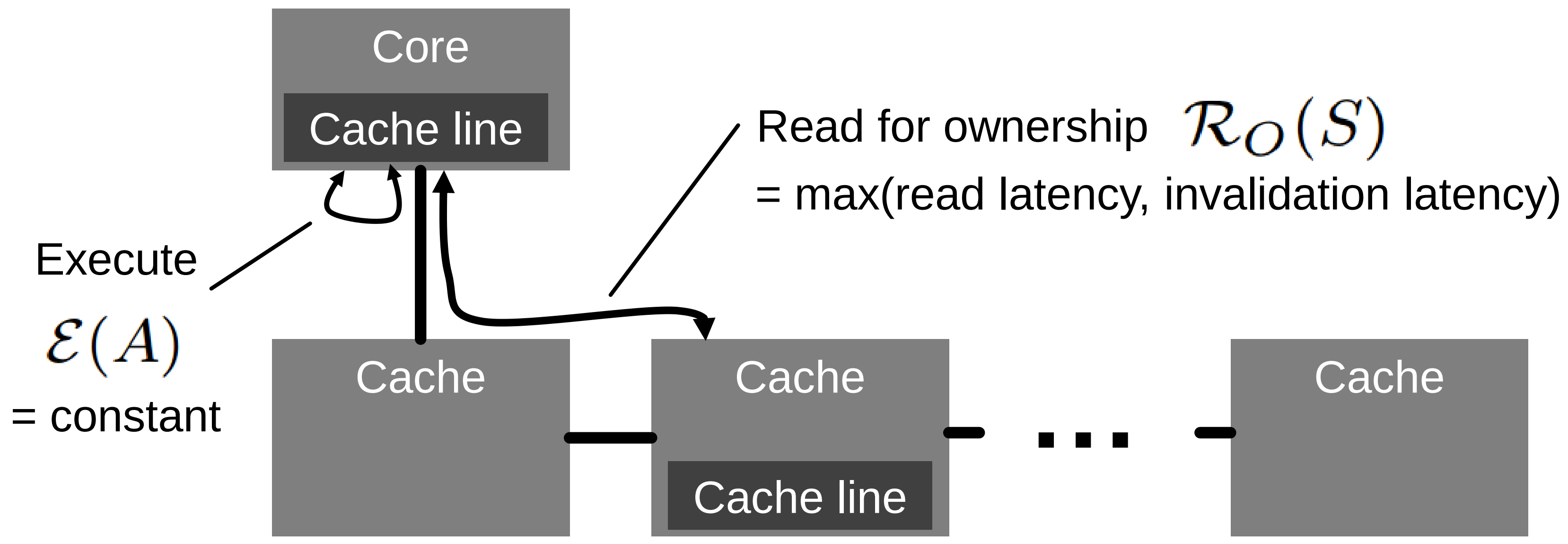
Latency
Xeon Phi, Modified / Exclusive

CAS





Latency Model
Exclusive or Modified State

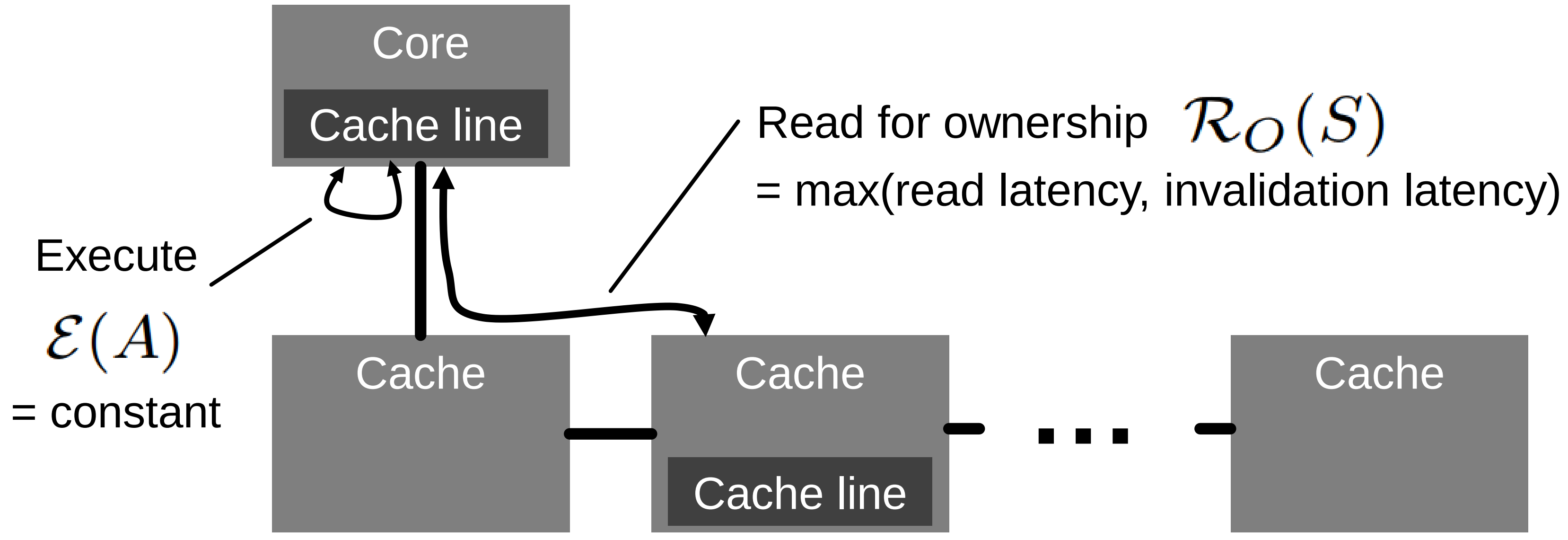

Core
Cache line
Read for ownership $\mathcal{R}_O(S)$
= max(read latency, invalidation latency)
Execute
$\mathcal{E}(A)$
= constant
Cache
Cache
Cache line
…
Cache

$$\mathcal{L}(A, S) = \mathcal{R}_O(S) + \mathcal{E}(A)$$

Atomic

Cache coherence state

LATENCY MODEL
SHARED STATE

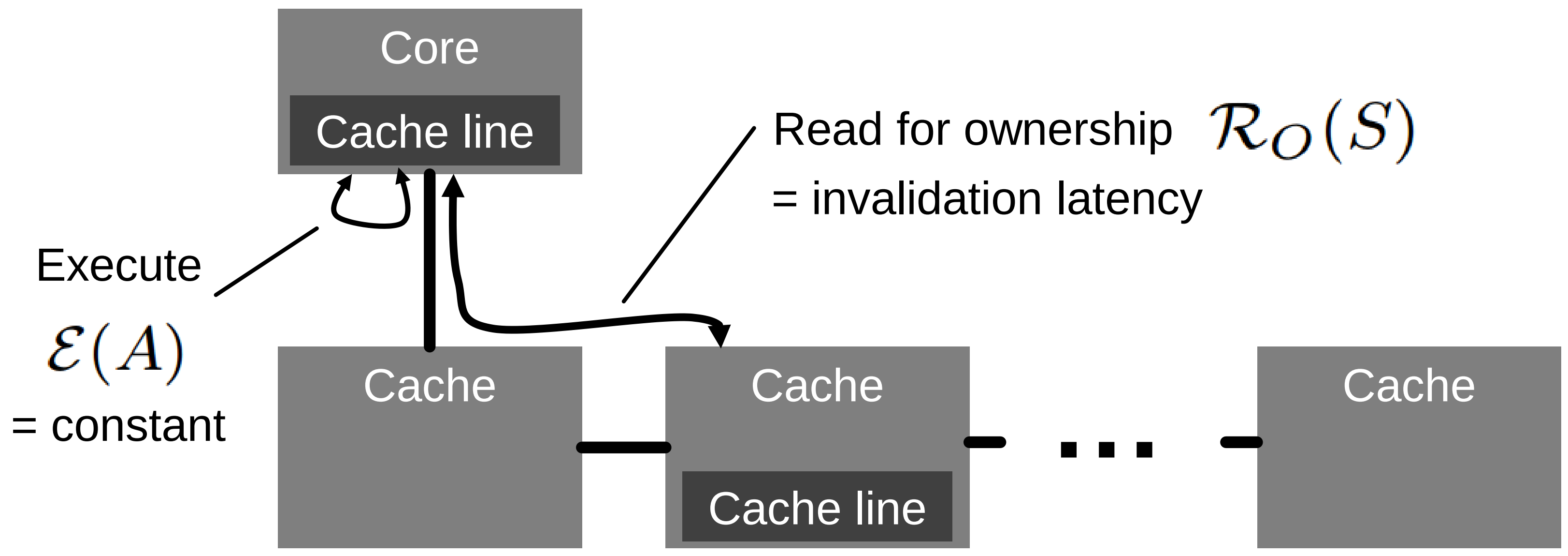

Core
Cache line
Read for ownership $\mathcal{R}_O(S)$
= max(read latency, invalidation latency)
Execute
$\mathcal{E}(A)$
= constant
Cache
Cache
Cache line
Cache
...

$$\mathcal{L}(A, S) = \mathcal{R}_O(S) + \mathcal{E}(A)$$

Atomic

Cache coherence state

Latency Model
Shared State



$$\underbrace{\mathcal{L}(A, S)}_{} = \mathcal{R}_O(S) + \mathcal{E}(A)$$

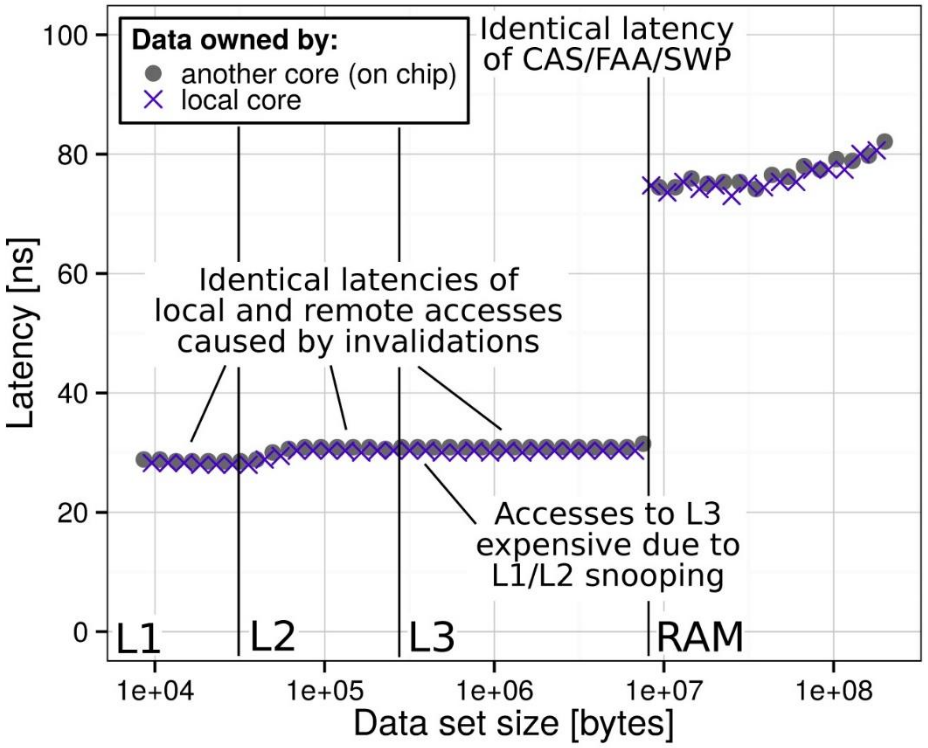
Atomic

Cache coherence state

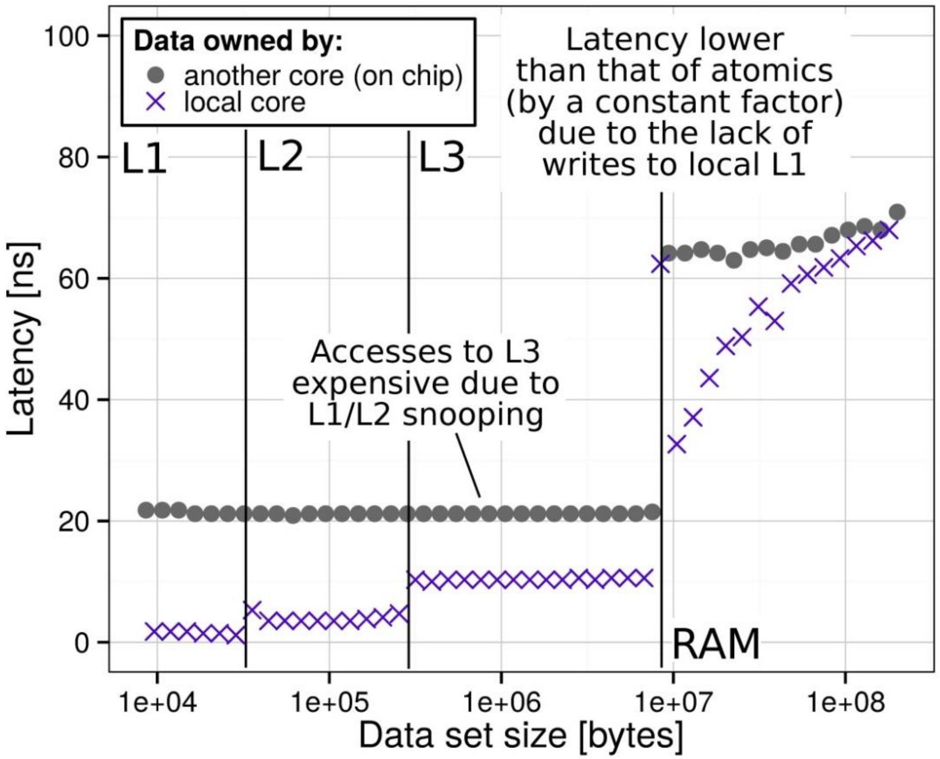
Latency

Haswell, Shared

Atomics



Read



How to force cache coherence state

- **F(M): Write cache line (invalidates all copies)**
- **F(E): F(M) → flush → read**
- **F(S): F(E) → read by some other core**

D. Hackenberg, D. Molka, W. Nagel. Comparing Cache Architectures and Coherency Protocols on x86-64 Multicore SMP Systems. MICRO'09