



The Machine: An Architecture for Memory-centric Computing

Kimberly Keeton

Workshop on Runtime and Operating Systems for Supercomputers (ROSS)

June 2015

Next wave: Cyber physical age





Internet of people



Internet of Things



By 2020

	Connected devices	30+ billion⁽²⁾	44 ZB
	IoT "Things"	200 billion⁽¹⁾	
	Smart meters	1 billion⁽³⁾	
		...for 8 billion⁽⁴⁾	

Pervasive connectivity

Smart device expansion

Explosion of information

(1) IDC "Worldwide Internet of Things (IoT) 2013-2020 forecast" October 2013. (2) IDC "The Digital Universe of Opportunities: Rich Data and the Increasing Value of the Internet of Things" April 2014
 (3) Global Smart Meter Forecasts, 2012-2020. Smart Grid Insights (Zypryme), November 2013 (4) <http://en.wikipedia.org>



The past 60 years



1950s



1970s

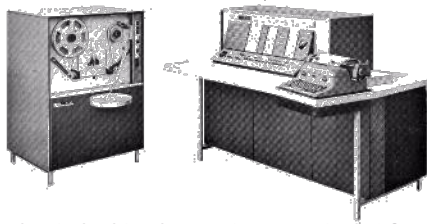


1990s



Today

1960s



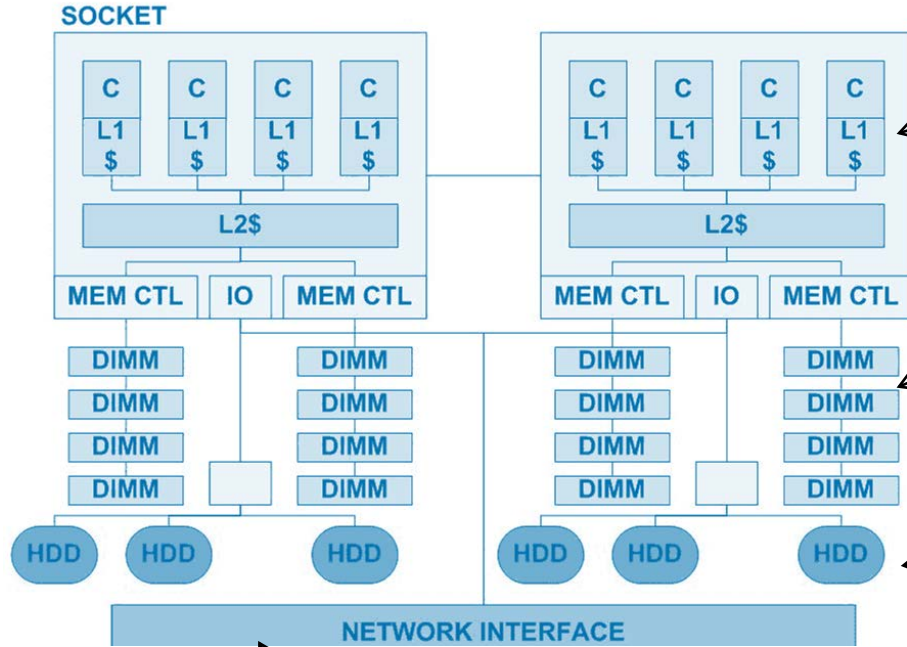
1980s



2000s



Architecture walls



Compute

Single-thread performance wall.
Diminishing return of multi-core.
Chip-edge bandwidth wall

Memory

DRAM reaching technology scaling wall

Storage

HDD/SSD layer is a significant performance bottleneck. Prevents big data getting closer to compute

Data Movement

Too slow (and cumbersome) for real-time access to shared memory



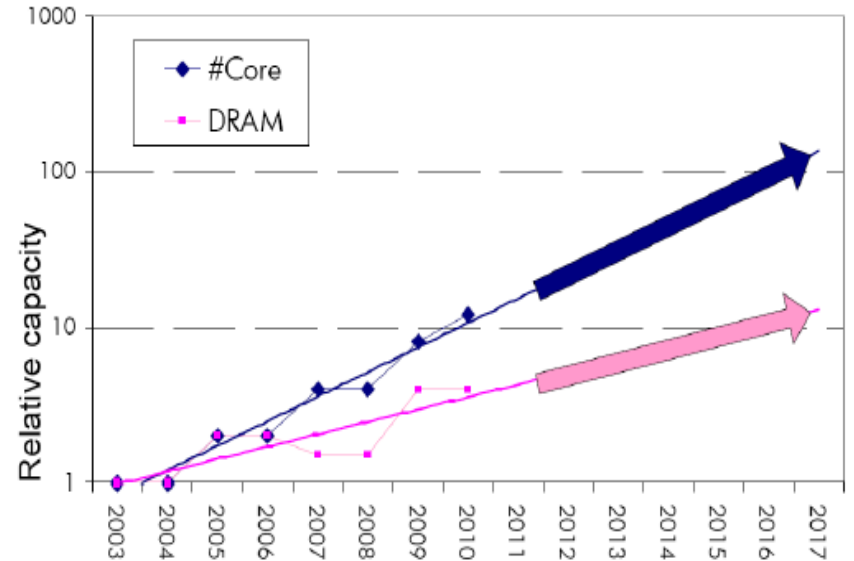
Things are becoming very unbalanced

National Labs plans for leading-edge supercomputer

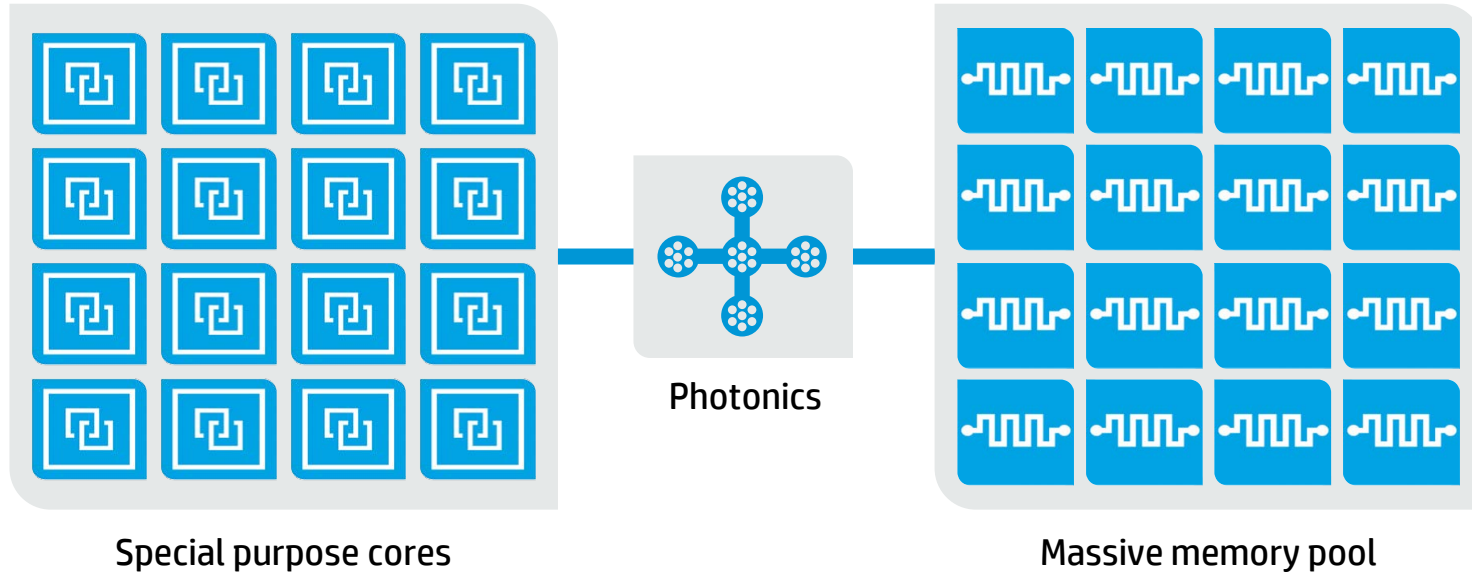
	2012	2016	2020	2024
Peak PFLOP/s	10-20	100-200	500-2000	2000-4000
Memory (PB)	0.5-1	5-10	32-64	50-100
Flops/Bytes	5-10	10 - 20	16 - 32	40 - 80

Source: Rick Stevens

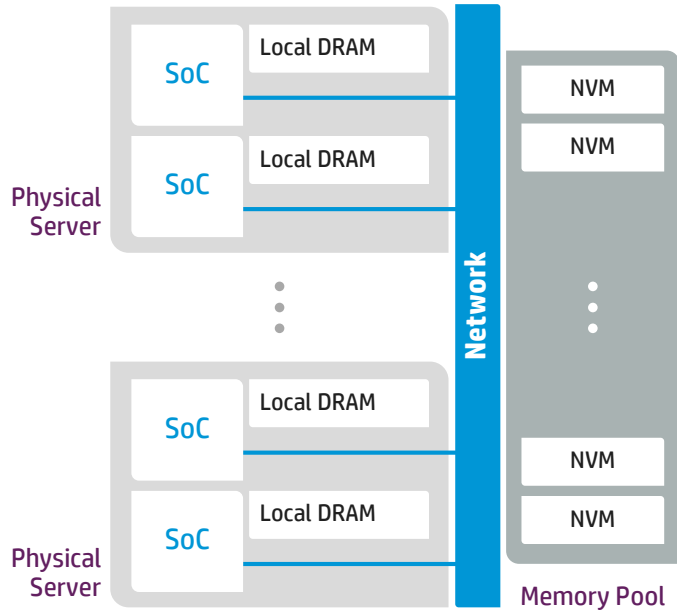
Core count doubling ~ every 2 years
DRAM DIMM capacity doubling ~ every 3 years



Architecture of the future: The Machine



Essential characteristics of The Machine



Converging memory and storage

- Byte-addressable non-volatile memory (NVM) replaces hard drives and SSDs

Shared memory pool

- NVM pool is accessible by all compute resources
- Optical networking advances provide near-uniform low latency
- Local memory provides lower latency, high performance tier

Heterogeneous compute resources distributed closer to data

Outline

Motivation

The Machine's memory-centric architecture

Implications for memory-centric operating systems

Implications for systems software

Implications for HPC

Conclusions



The Machine's memory-centric architecture



Today's trend to rack-scale architectures

Rack is the new unit of deployment in data centers

Pools of disaggregated resources

- Storage, networking and compute

Example: HP Moonshot m800 server cartridge

- Built around TI's KeyStone II System on Chip (SoC)
 - 4 general purpose ARM cores
 - 8 accelerated VLIW DSP cores



Moonshot
47" rack

20,000
cores

14TB
DRAM

100TB
Flash

Rack-scale computing in 2020?

	Today's rack	Hypothetical 2020 rack
# Cores	O(10,000)	O(100,000)
Memory	O(10 TB)	O(100 TB)
Storage	O(100 TB) (flash + SSD + HDD)	O(10-100 PB) (NVM)
Bandwidth / server	10 Gbps	1 Tbps

Paolo Costa "Towards Rack-scale Computing: Challenges and Opportunities," *Proc. First Intl. Workshop on Rack-scale Computing, 2014.*



Memory-centric rack-scale architectures

 Intel
**Rack Scale
Architecture**

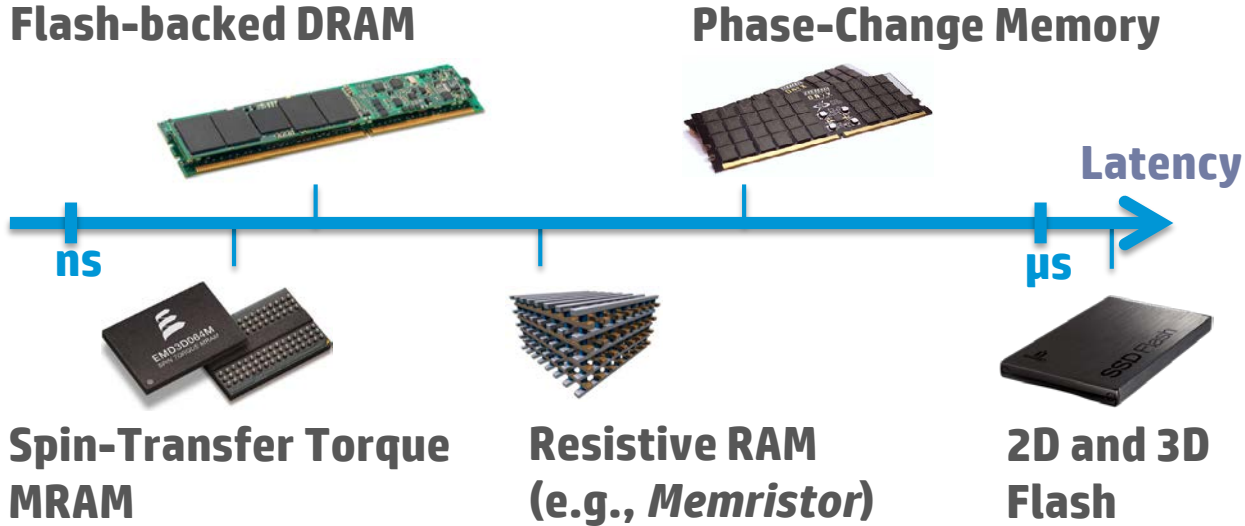


**UC Berkeley
Firebox**



**HP
The Machine**

Persistent storage will be accessible by load/store



Persistently stores data

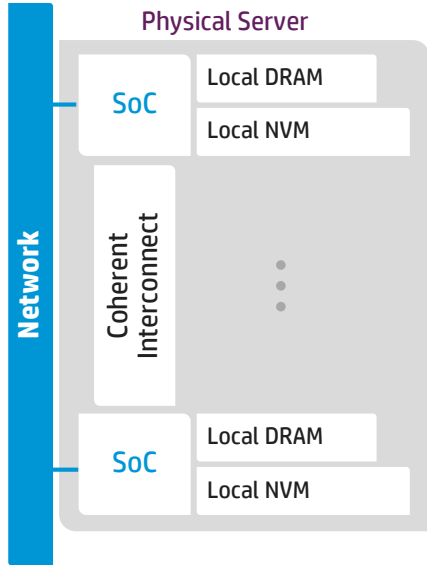
Access latencies comparable to DRAM

Byte addressable (load/store) rather than block addressable (read/write)

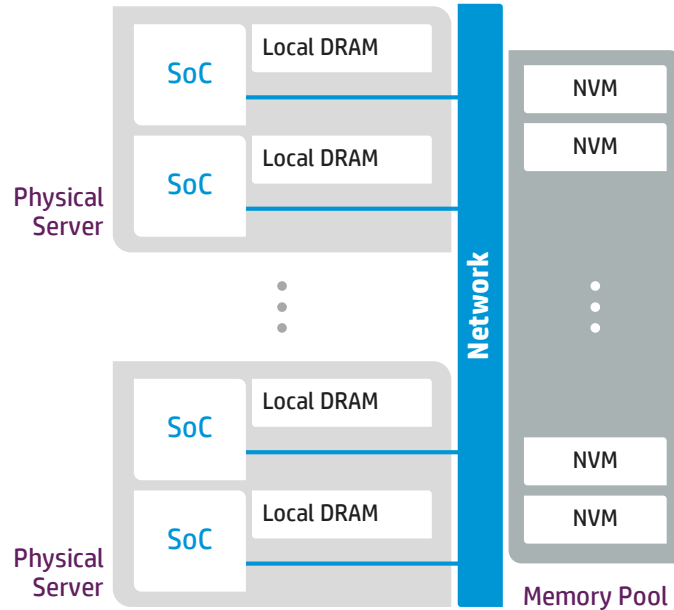
Haris Volos, et al. "Aerie: Flexible File-System Interfaces to Storage-Class Memory," *Proc. EuroSys 2014*.

NVM will become rack-scale pooled resource

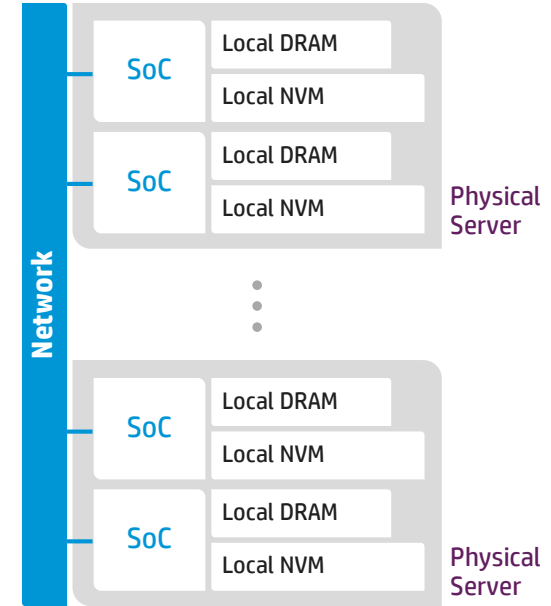
Shared everything



Shared something



Shared nothing



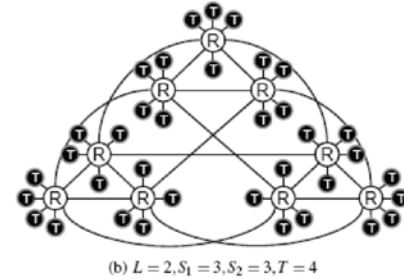
Optical networking will make most NVM equidistant

High-radix optical switches enable low-diameter network topologies

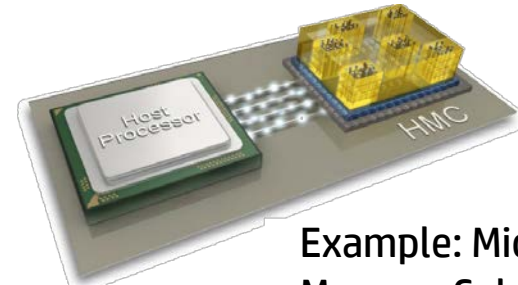
- Pooled NVM will appear at near-uniform low latency

Locality still matters

- Stacking and co-packaging permit node-local memory
- Local memory provides lower-latency, higher-bandwidth performance tier



Source: J. H. Ahn, et al., "HyperX: topology, routing, and packaging of efficient large-scale networks," *Proc. SC*, 2009.



Source: Micron

Example: Micron's Hybrid Memory Cube (HMC)



Heterogeneous and decentralized compute

Dark silicon effects

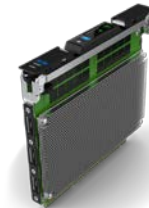
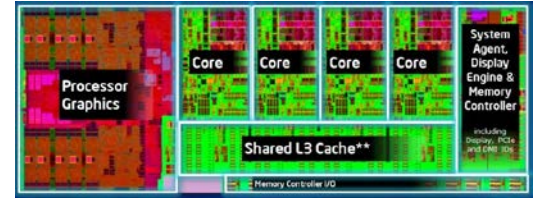
- Microprocessor designs are limited by power, not area

Application functionality will be offloaded

- Functionality migrates from application CPU to accelerators
- Computation moves closer to data

Memory will become more intelligent

- Memory takes active role in protection, allocation, synchronization, resilience, etc.



**HP Moonshot
ProLiant m710**

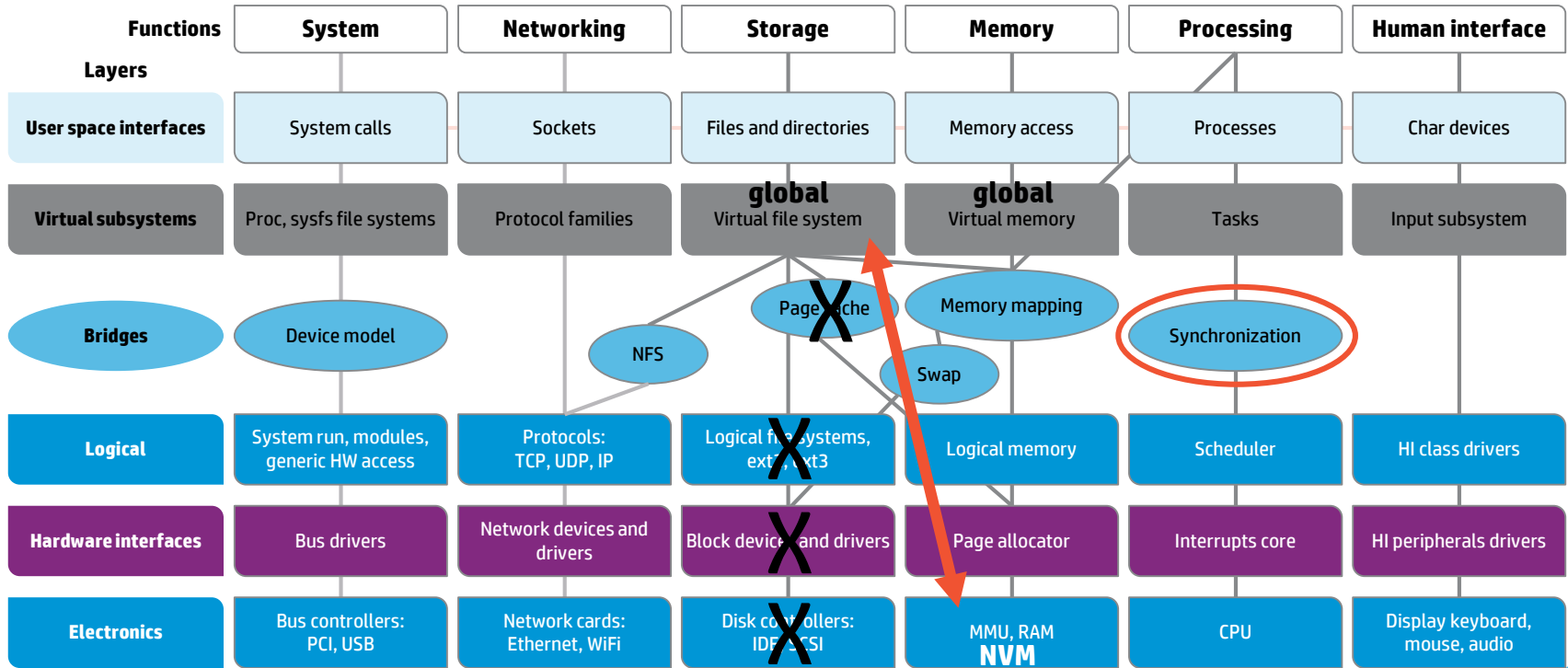


Implications for memory-centric operating systems

P. Faraboschi, K. Keeton, T. Marsland, D. Milojcic, “Beyond processor-centric operating systems,” *Proc. Workshop on Hot Topics in Operating Systems (HotOS)*, 2015.



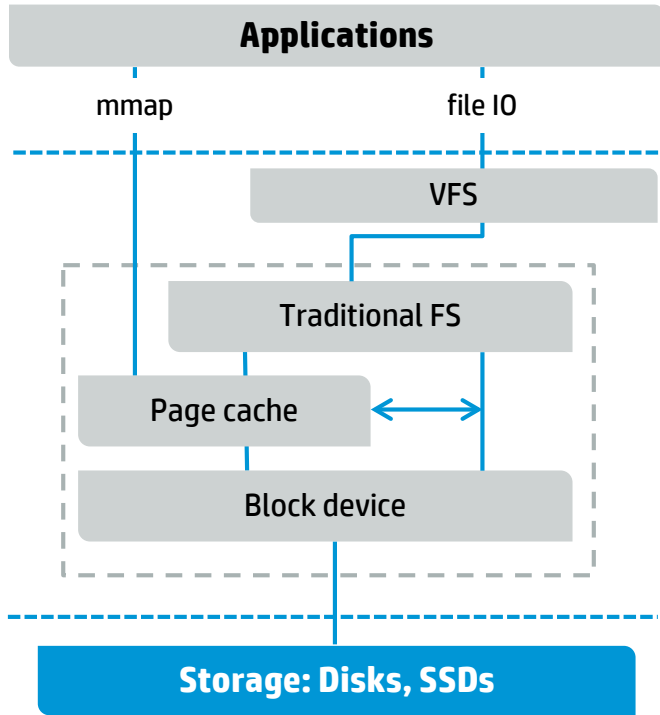
Linux kernel diagram



©2007-2009 Constantine Shulyupin <http://www.MakeLinux.net/kernel/diagram>



Traditional file systems



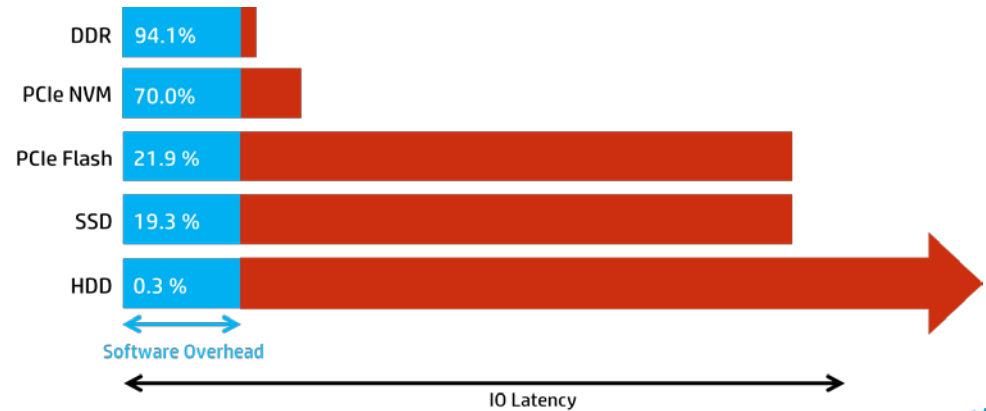
Separate storage address space

- Data is copied between storage and DRAM
- Block-level abstraction leads to inefficiencies

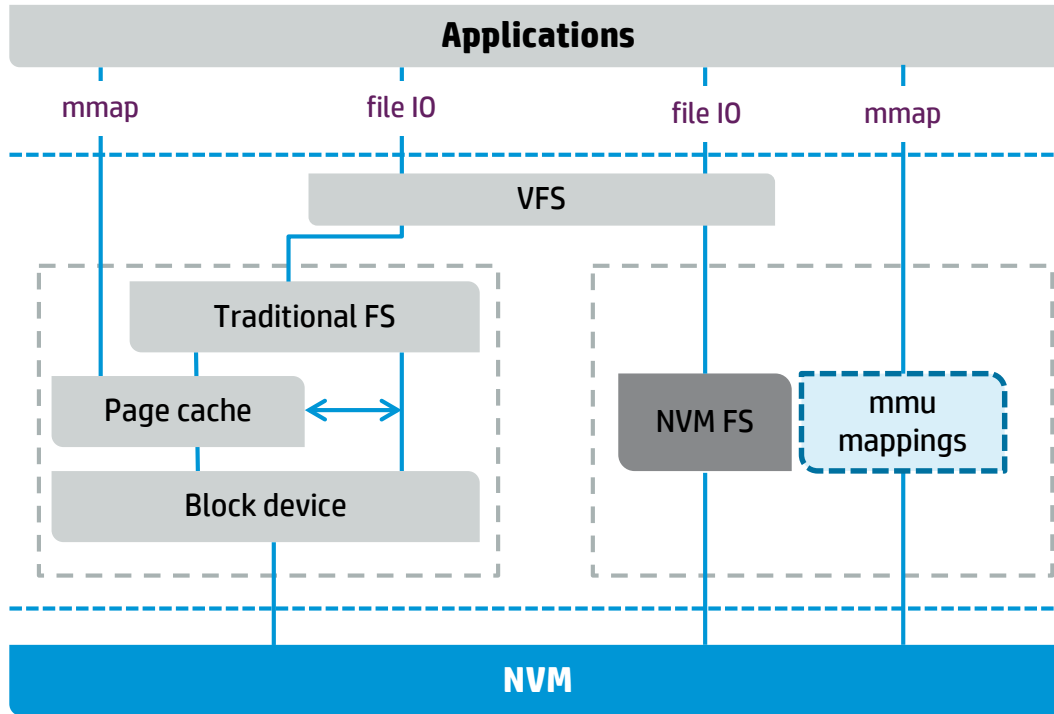
Use of page cache leads to extra copies

- True even for memory-mapped I/O

Software layers add overhead



Non-volatile memory aware file systems



Examples

- Linux DAX (pmem.io)

Low overhead access to persistent memory

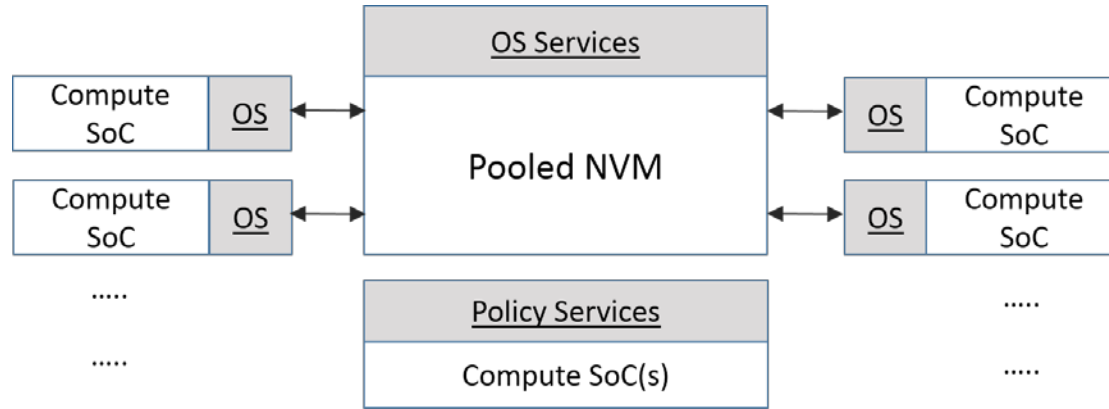
- No page cache
- Direct access with mmap

Leverage hardware support for consistency

Source: S. R Dullloor, et al. "System Software for Persistent Memory," *Proc. EuroSys*, 2014.



Distribution of memory management functionality

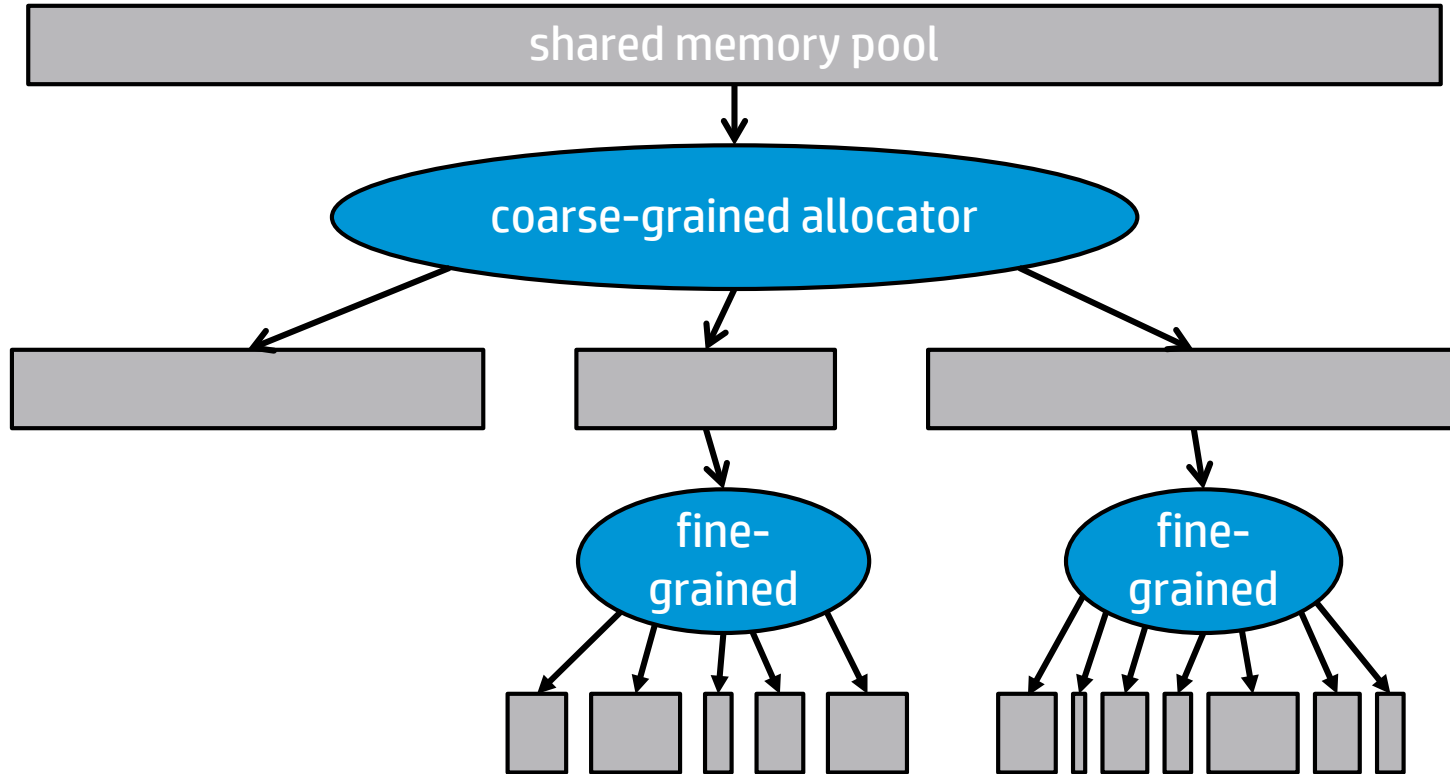


Memory management moves from processor-centric OS to distributed services

- Allocation, protection, synchronization, de/encryption, (de)compression, error handling
- Policy services: quotas, QoS

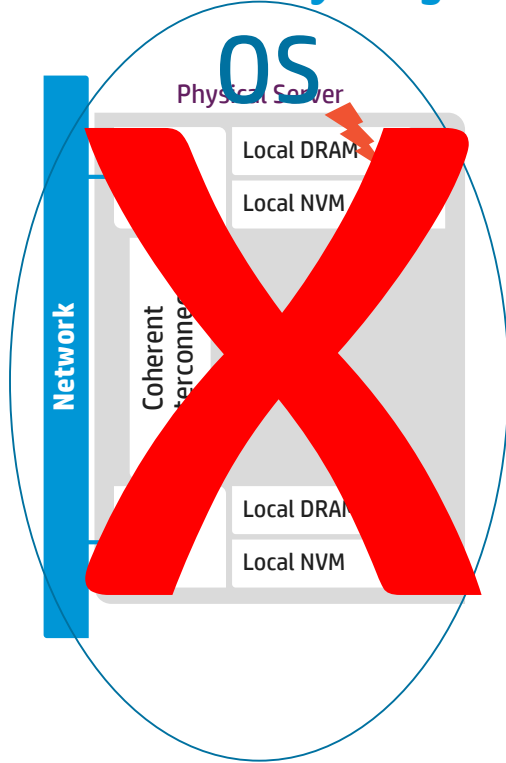
Cluster: compute SoCs, memory-side controllers, and accelerators

Example: hierarchical memory allocation

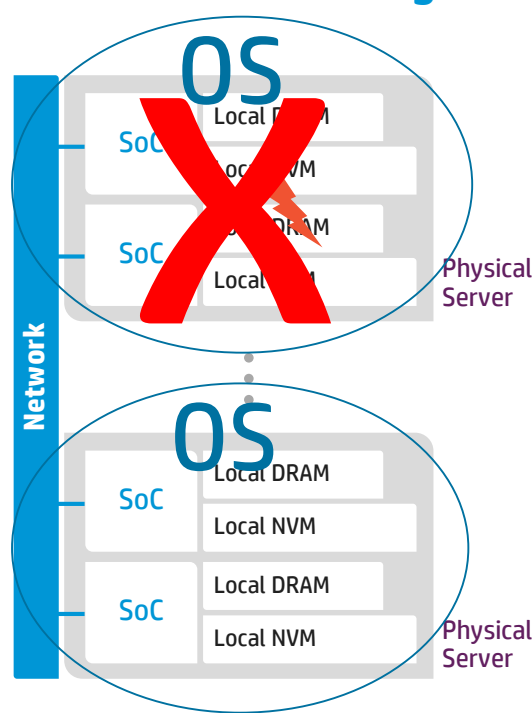


Shared NVM blurs fault domain boundaries

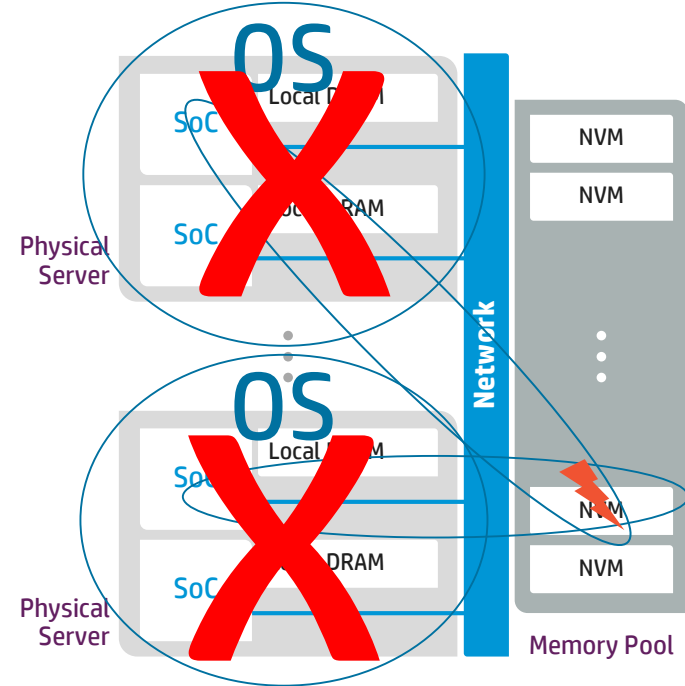
Shared everything



Shared nothing

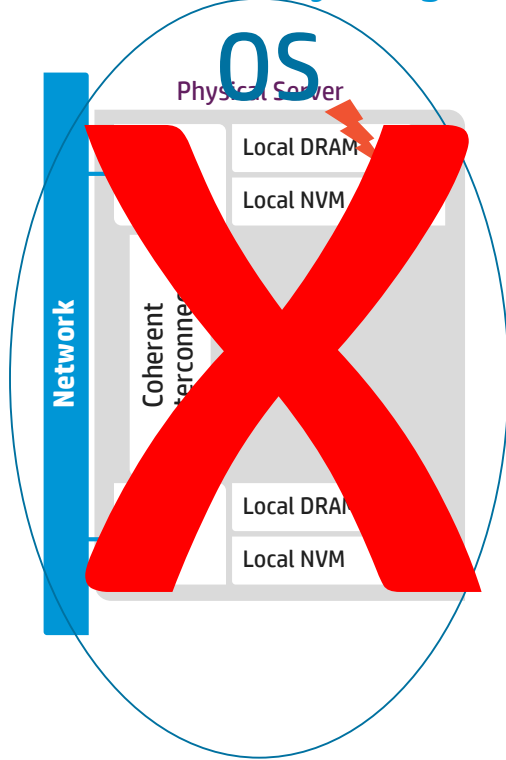


Shared something

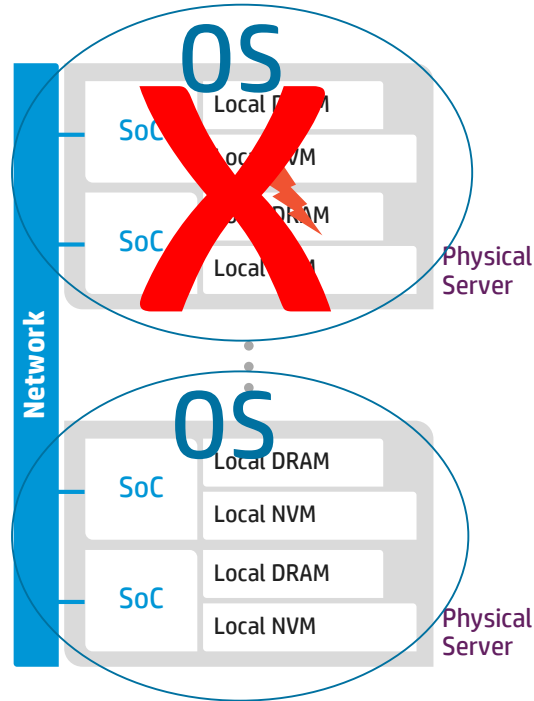


Shared NVM blurs fault domain boundaries

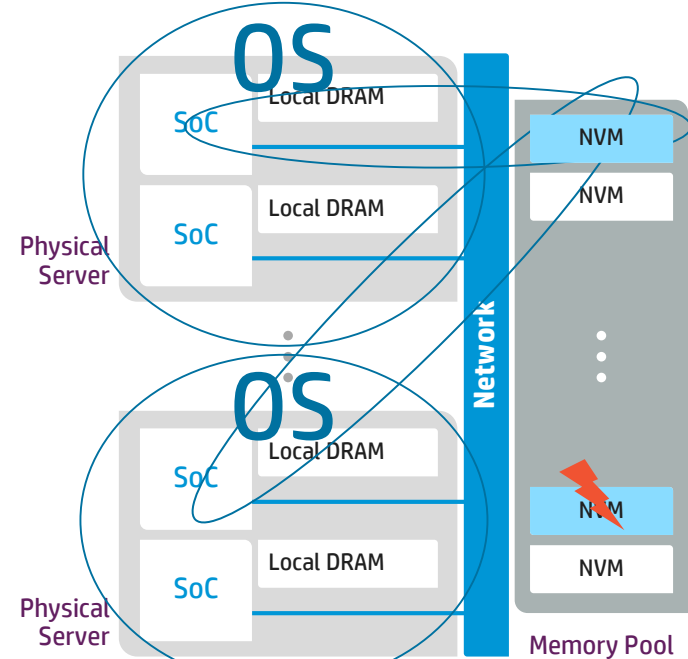
Shared everything



Shared nothing



Shared something



Memory error handling

Issue

- As size of memory increases, memory errors will be commonplace, not rare
- Memory interconnect errors (possibly transient) will manifest as memory errors
- Applications and OS must be cope with load/store access to NVM failing

Traditional

- OS memory failures considered unrecoverable, resulting in machine check
- User process with memory errors would be killed

Potential solutions

- Use replication and remapping to survive memory hardware failures
- Adapt traditional mitigation techniques to detect/correct problems transparently
- Use exceptions for memory error reporting, to permit the OS and apps to recover if possible



Protection and translation

Issue: tension between needs of translation and protection

- Translation should be as efficient as possible via very large pages or direct mapping
- Protection should be very fine-grained (e.g., object metadata)

Traditional

- OS supports multiple page sizes and uses them for both protection and translation
- No “goldilocks” page size due to conflict between translation and protection

Potential solutions

- Decouple translation and protection
- Provide protection by combination of processor-managed MMUs and memory elements
- Explore alternative protection mechanisms (e.g., capabilities, Mondrian memory protection)



Coping with volatility in NVM world

Issue

- Components of the memory system continue to be volatile (e.g., caches, store buffers)
- OS and applications need to control data movement between volatile caches and NVM

Traditional

- NVM transactions with undo logging
- ISA support beginning to become available (e.g., Intel clwb, pcommit)

Potential solutions

- Provide “flush-on-failure”: kernel flushes all CPU caches to NVM when failure is imminent
- Employ non-volatile last level cache of processor



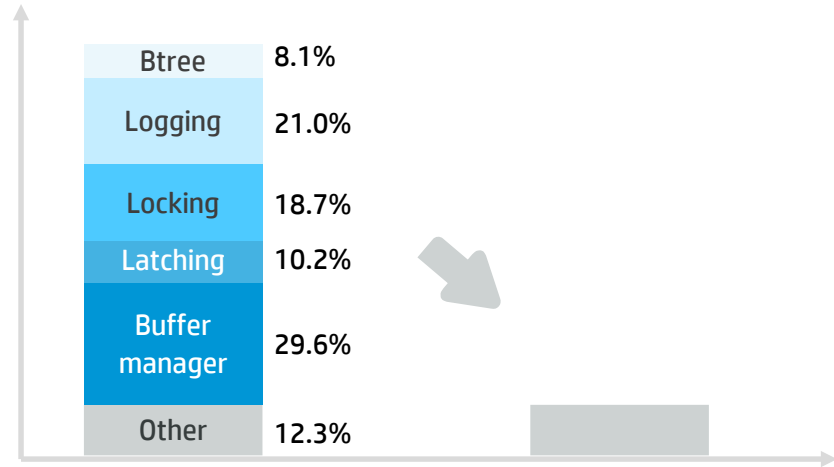
Implications for systems software



Traditional databases

Example: A database (write) transaction

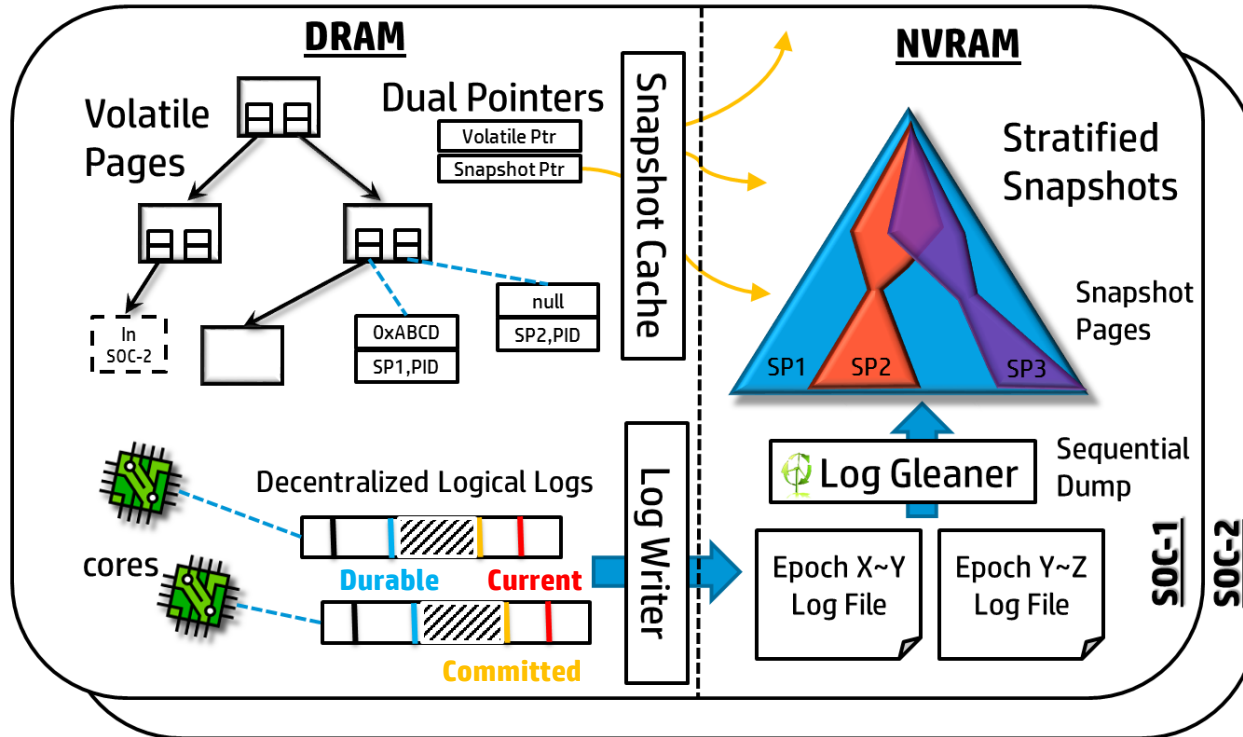
- Traditional databases struggle with **big & fast data**
- **90%** of a database transaction is overhead
- Memory-semantics nonvolatile memory: **up to 10x improvement**



S. Harizopoulos, D. Abadi, S. Madden, and M. Stonebraker, "OLTP Through the Looking Glass, and What We Found There," *Proc. SIGMOD*, 2008.



NVM-optimized embedded database: FOEDUS

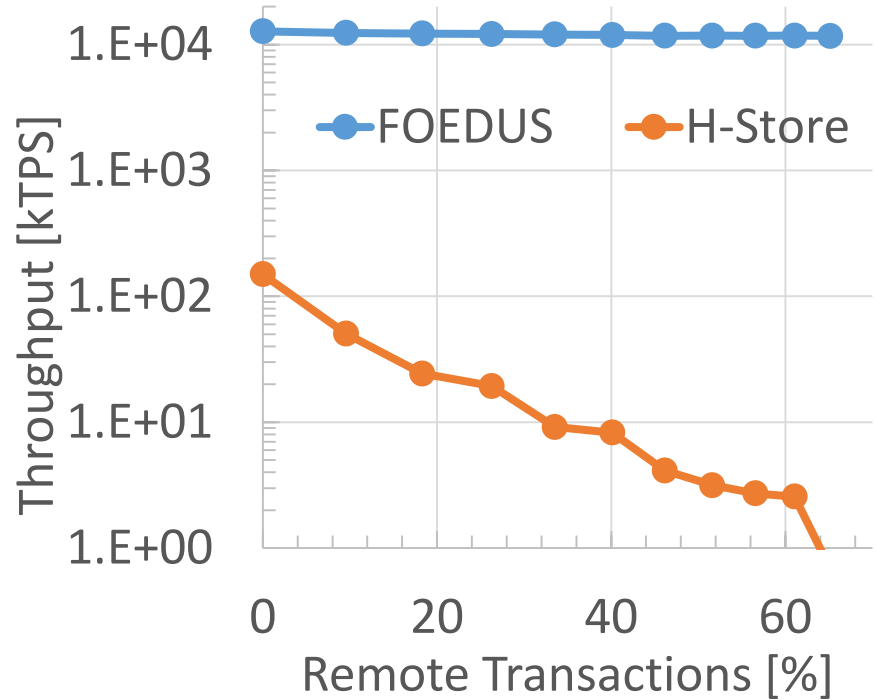


H. Kimura, "FOEDUS: OLTP engine for a thousand cores and NVRAM," *Proc. SIGMOD*, 2015.



Comparison with in-memory DBMS

- Workload: TPC-C Benchmark
- HP Superdome X
 - 16 sockets, 240 cores, 12TB DRAM
- H-Store: main-memory parallel DBMS
- Optimistic concurrency control more resilient to contention
- ~**100x** faster than H-Store



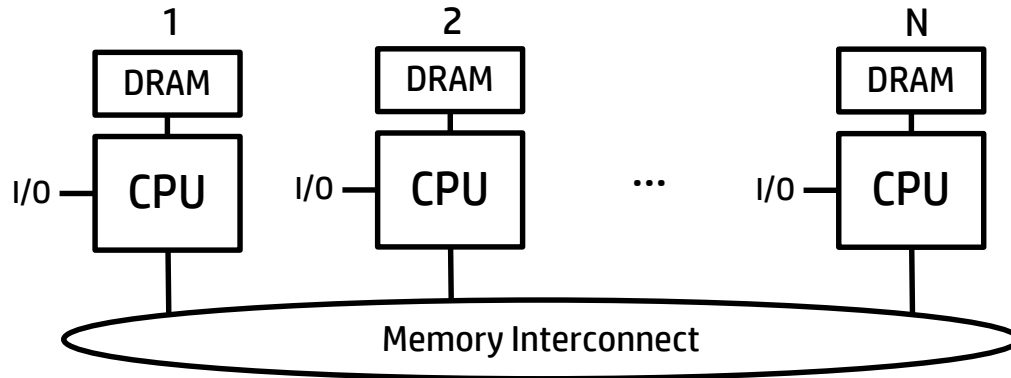
H. Kimura, "FOEDUS: OLTP engine for a thousand cores and NVRAM," *Proc. SIGMOD*, 2015.



NVM-aware data serving: distributed hash table

CREW: concurrent reads, exclusive write

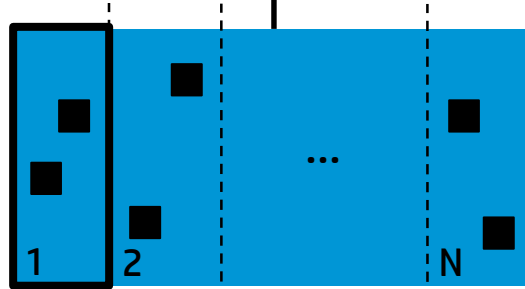
Each server owns write permission to one region and read permission to whole NVM



Permissions:

Read-Write = {1}

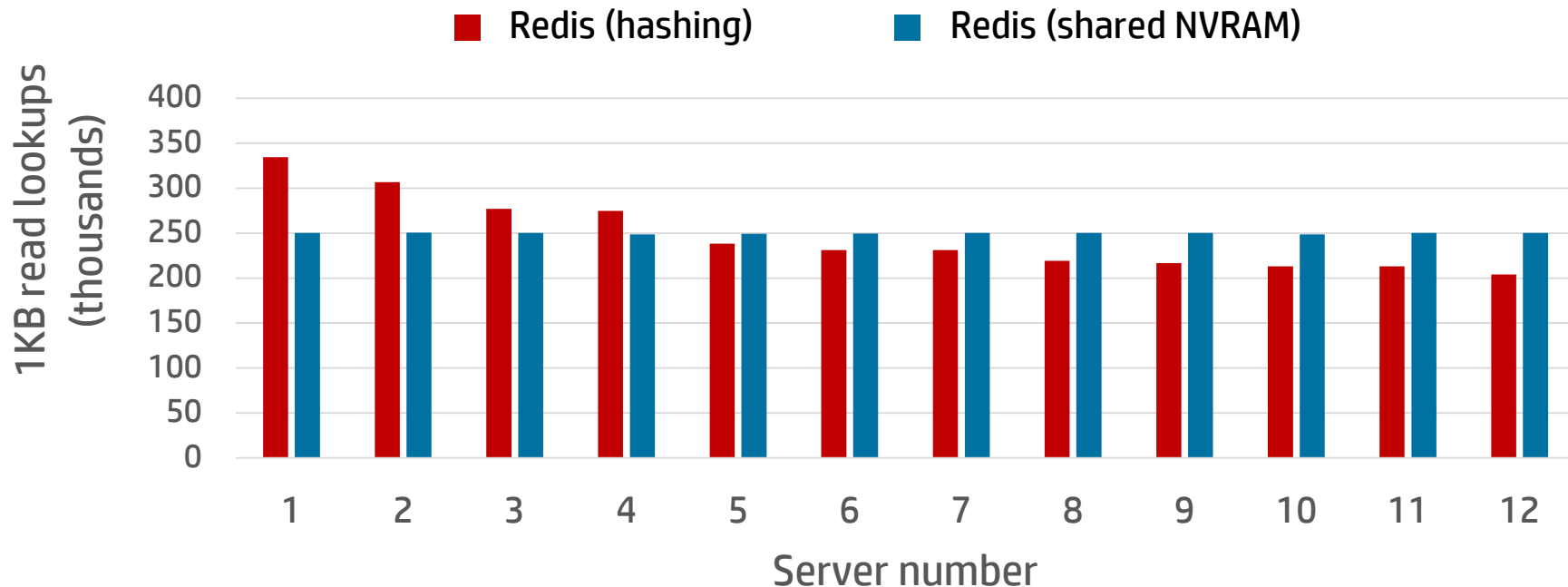
Read-Only = {1,2,...N}



← Data stored in shared NVM

Per-server network utilization

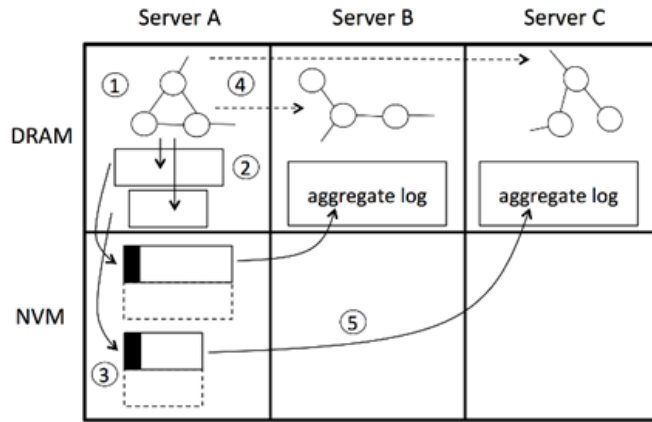
Yahoo Cloud Serving Benchmark: 0.99 Zipf read workload



Shared NVRAM eliminates load imbalance

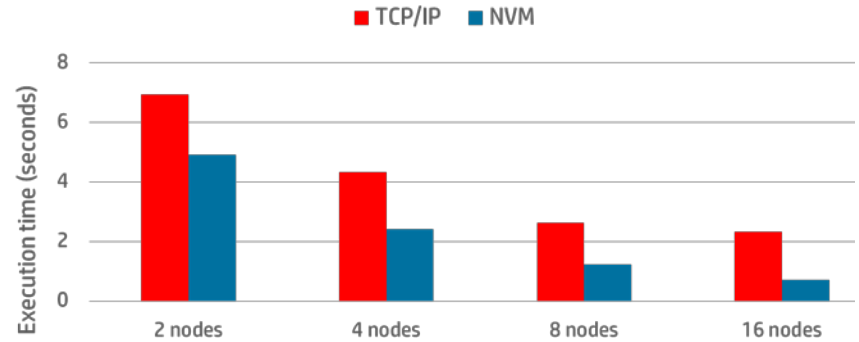


NVM-aware data analytics



- Pregel-like graph processing with NVM-based log communication
- Bulk synchronous parallel (BSP) compute model
- Senders log updates in NVM, notify receivers

PageRank on Twitter graph data set



Accelerating shuffle phase improves overall execution time

S. Novakovic, K. Keeton, P. Faraboschi, R. Schreiber, E. Bugnion. "Using shared non-volatile memory in scale-out software," *Proc. 2nd Intl. Workshop on Rack-scale Computing (WRSC)*, 2015.

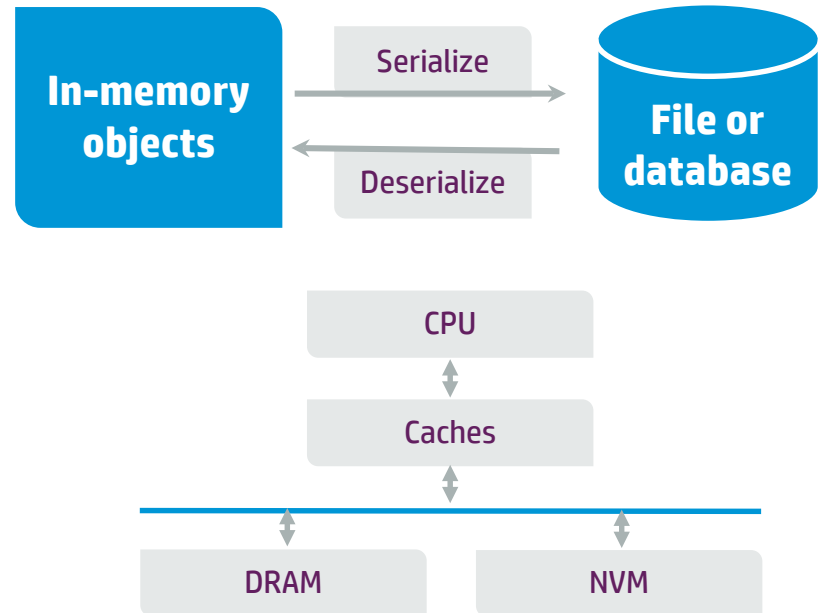
Do we need separate data representations?

In-storage durability

- + Separate object and persistent formats
- Programmability and performance issues
- Translation code error-prone and insecure

In-memory durability

- + In-memory objects are durable throughout
- + Byte-addressability simplifies programmability
- + Low load/store latencies offer high performance
- Persistent does not mean consistent!



NVM-aware application programming

Why can't I just write my program, and have all my data be persistent?

Consider a simple banking program (just two accounts):

```
double accounts[2];
```

Between which I want to transfer money. Naïve implementation:

```
transfer(int from, int to, double amount) {  
    accounts[from] -= amount;  
    accounts[to] += amount;  
}
```



What if I crash here?



What if I crash here?

Crashes cause corruption, which prevents us from merely restarting the computation

Manual solution

```
persistent double accounts[2];
transfer(int from, int to, double amount) {
  <save old value of accounts[from] in undo log>;
  <flush log entry to NVM>
      accounts[from] -= amount;
  <save old value of accounts[to] in undo log>;
  <flush log entry to NVM>
      accounts[to] += amount;
  <flush all other persistent stores to NVRAM>
  <clear and flush log>
}
```

- Need code that plays back undo log on restart
- Getting this to work with threads and locks is very hard
- Really want to optimize it
- **Very unlikely application programmers will get it right**



Our solution: Consistent sections

Provide a construct that atomically updates NVM

```
persistent double accounts[2];
transfer(int from, int to, double amount) {
  __atomic {
    accounts[from] -= amount;
    accounts[to]   += amount;
  }
}
```

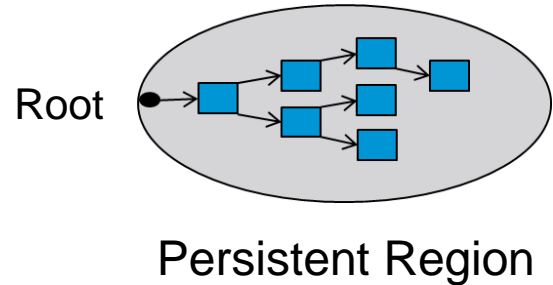
- Ensures that updates in **__atomic** block are either completely visible after crash or not at all
- If updates in **__atomic** block are visible, then so are prior updates to persistent memory

D. Chakrabarti, H. Boehm and K. Bhandari. “Atlas: Leveraging Locks for Non-volatile Memory Consistency,” *Proc. OOPSLA*, 2014.



The Atlas programming model

- **Programmer distinguishes persistent and transient data**
- **Persistent data lives in a “persistent region”**
 - Mappable into process address space (no DRAM buffers)
 - Accessed via CPU loads and stores
- **Programmer writes ordinary multithreaded code**
 - Atlas provides automatic durability support at a fine granularity, complete with recovery code
 - Atlas derives durable data consistency from existing concurrency constructs
- **Protection against failures**
 - Process crashes: Works with traditional architectures
 - Kernel panics and power failures: Requires NVM and CPU cache flushes



D. Chakrabarti, H. Boehm and K. Bhandari. “Atlas: Leveraging Locks for Non-volatile Memory Consistency,” *Proc. OOPSLA*, 2014.

Use cases

Replace existing durability support (file/database) with direct load/store of *NVM*

- Example: OpenLDAP with memory-mapped database
- NVM-based implementation **300x** faster than hard disks for series of gets and puts

Enable a new class of applications where in-memory objects are always *durable*

- Example: durable memcached
- Existing transient cache is persisted in NVM, enabling hot restart
- Overhead of durability is about **60%** of total time for series of gets and puts
- Overhead reduction possible for systems that provide flush on failure

D. Chakrabarti, H. Boehm and K. Bhandari. "Atlas: Leveraging Locks for Non-volatile Memory Consistency," *Proc. OOPSLA*, 2014.



Implications for HPC



Potential benefits of The Machine for HPC workloads

- **Memory is fast**

- Application checkpoints will be dramatically faster
- No need to explicit load data from disk
- Faster post-experiment visualization and verification

- **Memory is large**

- Overcomes challenges of weak scaling at very large scale
- Permits simultaneous execution of many related problem instances
 - Uncertainty quantification, comparison of alternatives for optimization, etc.

- **Memory is shared**

- Shared datasets permit low overhead work stealing – potential solution to static load balancing challenges
- Shared data structures limit wasted capacity for replicating read-only data structures across nodes

How would you leverage The Machine for your applications?



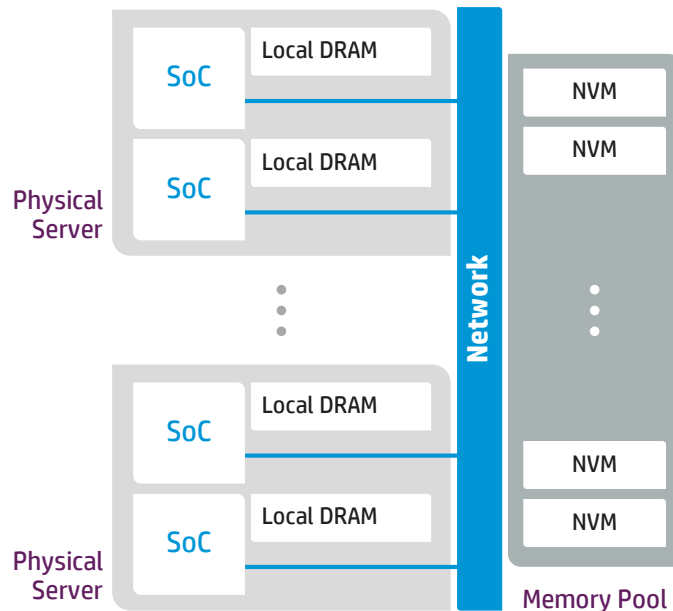
Wrapping up

The Machine: memory-centric computing

- Fast load/store access to large shared pool of non-volatile memory

Many opportunities for software innovation

- Operating systems
- Data stores
- Analytics platforms
- Programming models and tools
- Algorithms



For more information...part 1

Website and research papers

- <http://www.hpl.hp.com/research/systems-research/themachine/>
- D. Chakrabarti, H. Boehm and K. Bhandari. “Atlas: Leveraging Locks for Non-volatile Memory Consistency,” *Proc. Object-Oriented Programming, Systems, Languages & Applications (OOPSLA)*, 2014.
- P. Faraboschi, K. Keeton, T. Marsland, D. Milojevic, “Beyond processor-centric operating systems,” *Proc. Workshop on Hot Topics in Operating Systems (HotOS)*, 2015.
- H. Kimura, “FOEDUS: OLTP engine for a thousand cores and NVRAM,” *Proc. SIGMOD*, 2015.
- S. Novakovic, K. Keeton, P. Faraboschi, R. Schreiber, E. Bugnion. “Using shared non-volatile memory in scale-out software,” *Proc. Intl. Workshop on Rack-scale Computing (WRSC)*, 2015.
- H. Volos, S. Nalli, S. Panneerselvam, V. Varadarajan, P. Saxena, M. Swift. "Aerie: Flexible File-System Interfaces to Storage-Class Memory," *Proc. EuroSys*, 2014.



For more information...part 2

Videos on The Machine



- HP Discover 2014 talks
 - HP Labs Director Martin Fink's announcement: <https://www.youtube.com/watch?v=Gxn5ru7klUQ>
 - Kim Keeton's talk on hardware technologies: https://www.youtube.com/watch?v=J6_xq3mHnng
- HP Discover 2015 talks
 - Kim Keeton's talk on systems software: <https://www.youtube.com/watch?v=WZbPyV5AnKM>
- Linaro Connect 2015 talks
 - Dejan Milojicic's keynote: <http://connect.linaro.org/hkg15/>