Software Challenges for The Machine

Sam Fineberg
Distinguished Technologist, HP Storage
MSST 15, June 2015
(many thanks to my colleagues at HP Labs who are doing the real work)

What this talk is about

1. The perfect storm

Data explosion + architecture walls + device crisis

2. What is HP doing about it

“The Machine” project at HP Labs

3. The memory is the computer

The software revolution when everything becomes persistent

The Approaching Cyber physical age

Internet of People

Internet of Things

By 2020

30 Billion[2] Connected Devices

44 ZB

200 Billion[1] IoT "Things"

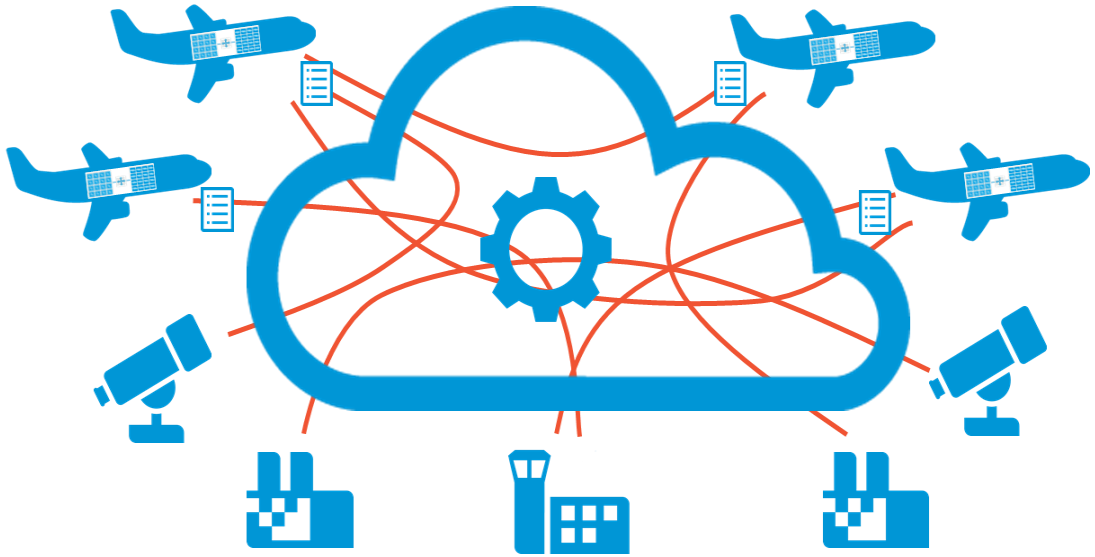
1 Billion[3] Smart Meters

… 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

Example: a mesh of connected aircraft



20 TB ✖ 2 ✖ 3 ✖

20 terabytes of information per engine per hour

twin-engine aircraft

three-hour flight duration

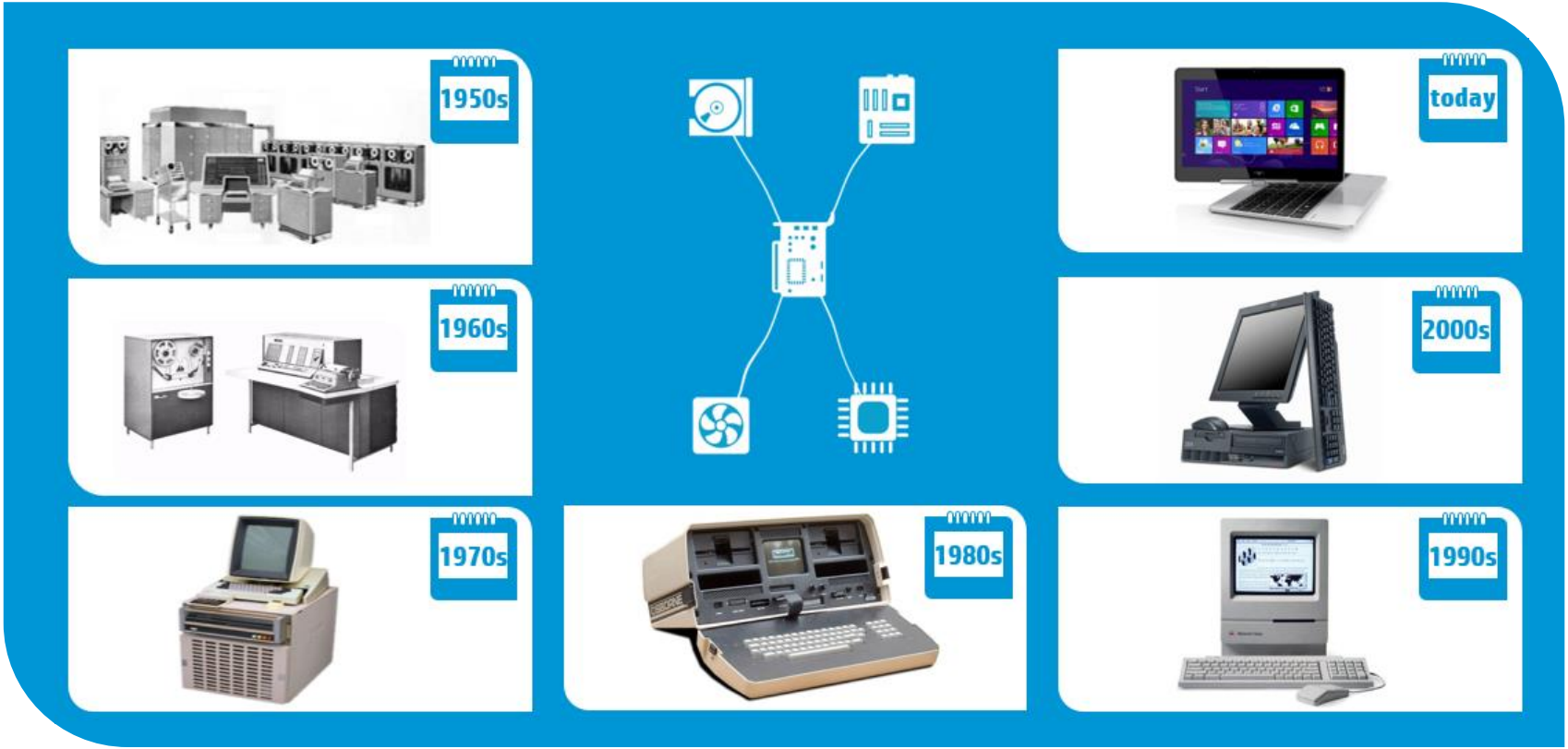
25,000 ✖ 365

commercial flights per day (USA)

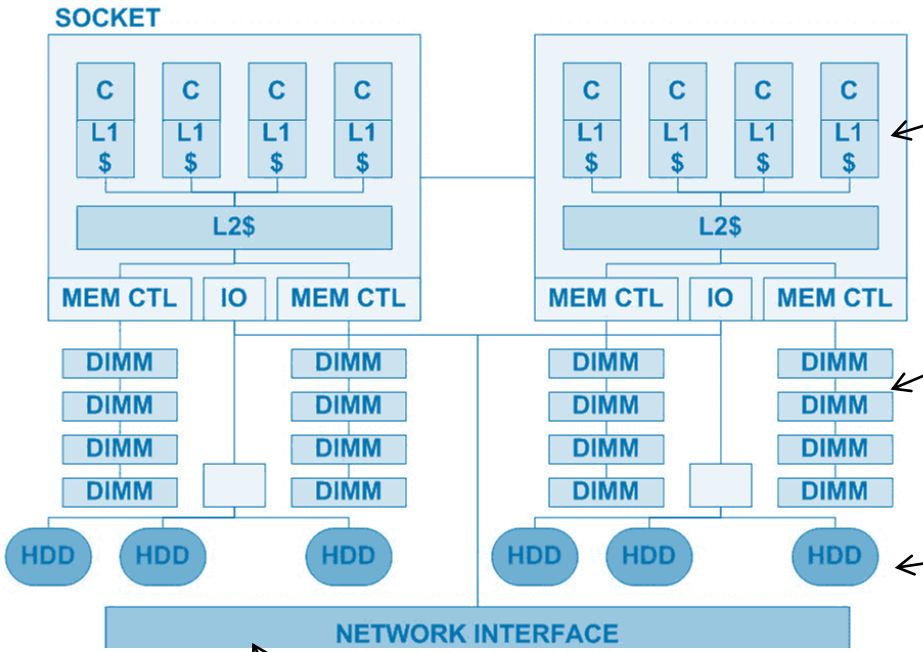
days in a year

= **1,095,000,000 TB (1 ZB)**

Architecture has not changed for 60 years



Architecture Walls



Compute

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

Memory

DRAM reaching technology scaling wall

Storage

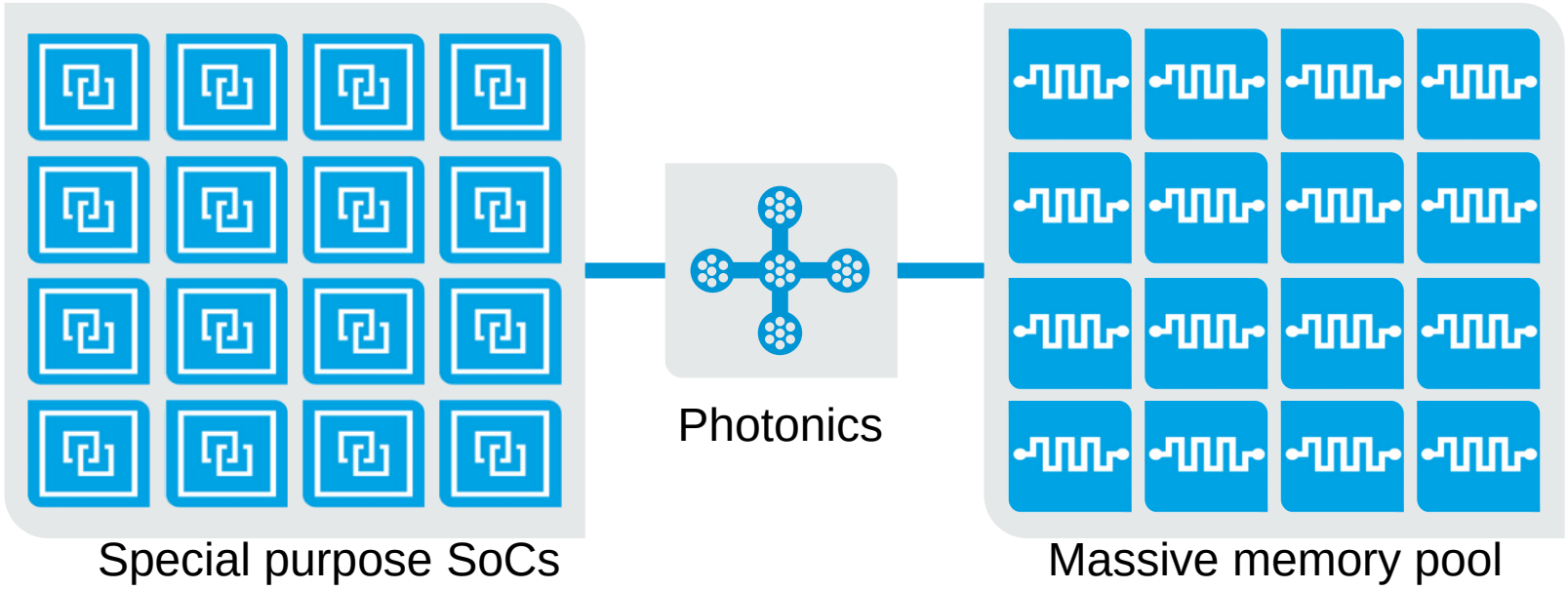
HDD/SSD layer is a significant performance bottleneck. Prevents big data getting closer to compute

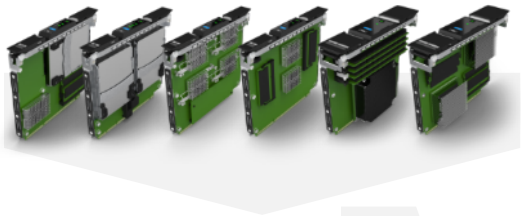
Data Movement

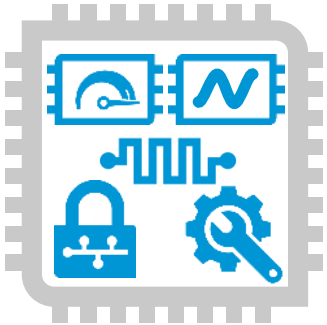
Too slow (and cumbersome) for real-time access to shared memory

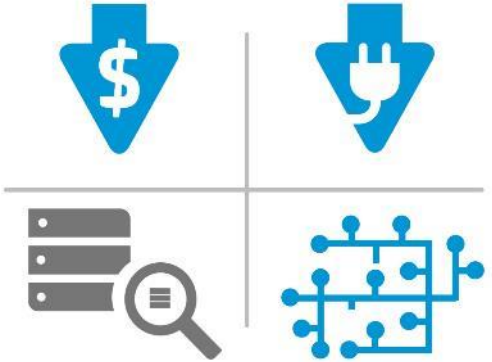
The Machine Project

Architecture of the future: The Machine







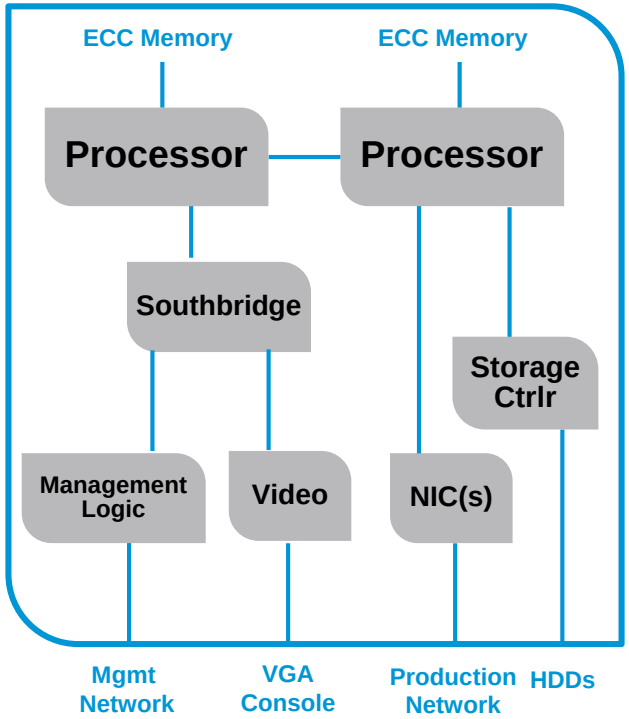


SoCs customized to the workload

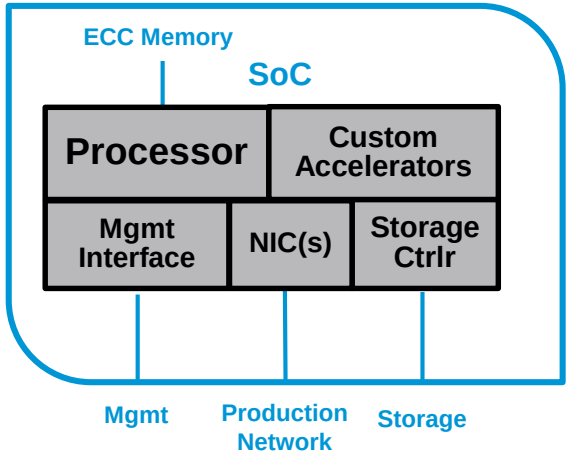
Application-focused silicon

Special purpose SoCs

Traditional Server Motherboard



System on a Chip (SoC)-based Server



- Less general-purpose, more workload focused
- Dramatic reduction in power, cost, and space
- SoC vendors bring their own differentiated features and opportunities to disrupt markets

Why photonics?

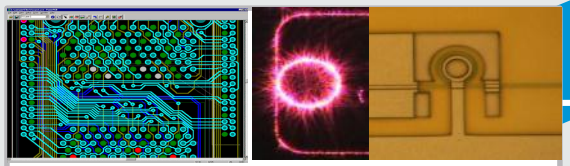
Capability / Value

Time

Next-generation

Photonic-based architectures:
The Fabric for intern SoC and cross SOC for highly integrated, and hyper-efficient architectures

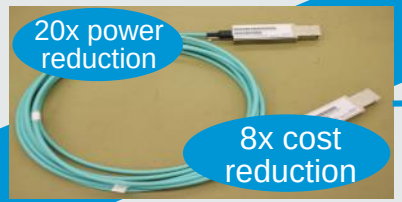
Best of breed, better together:
- Dense PM
- Low-power processors
- Flat, SDN networks
- Photonic interconnection



From this To this

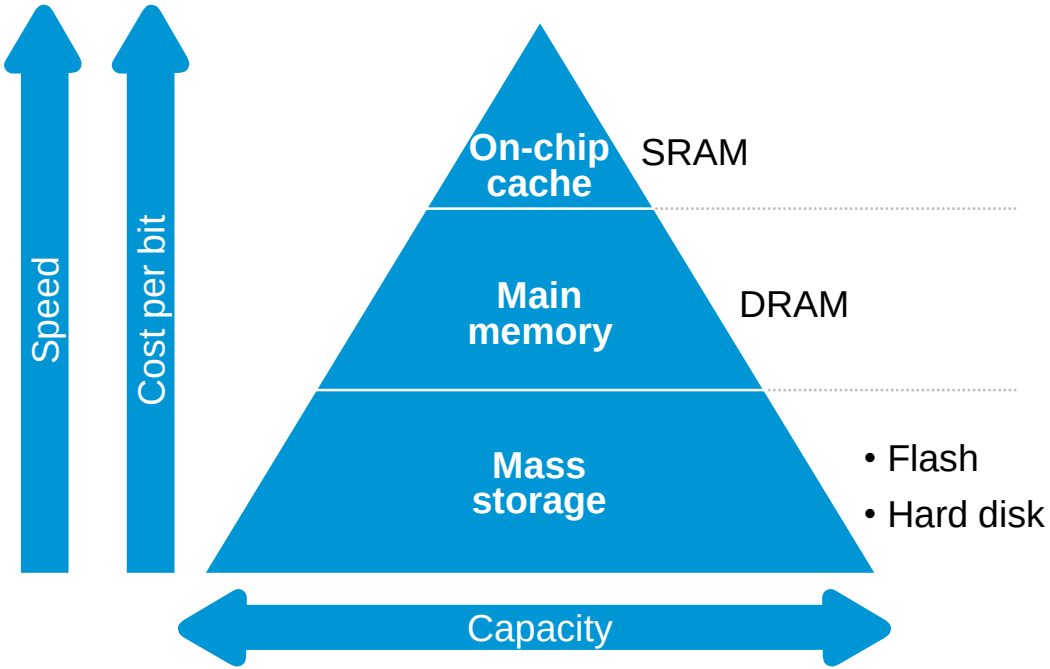
Tomorrow

Photonic-integrated fabrics:
Enable next-generation blades, storage, and network fabrics



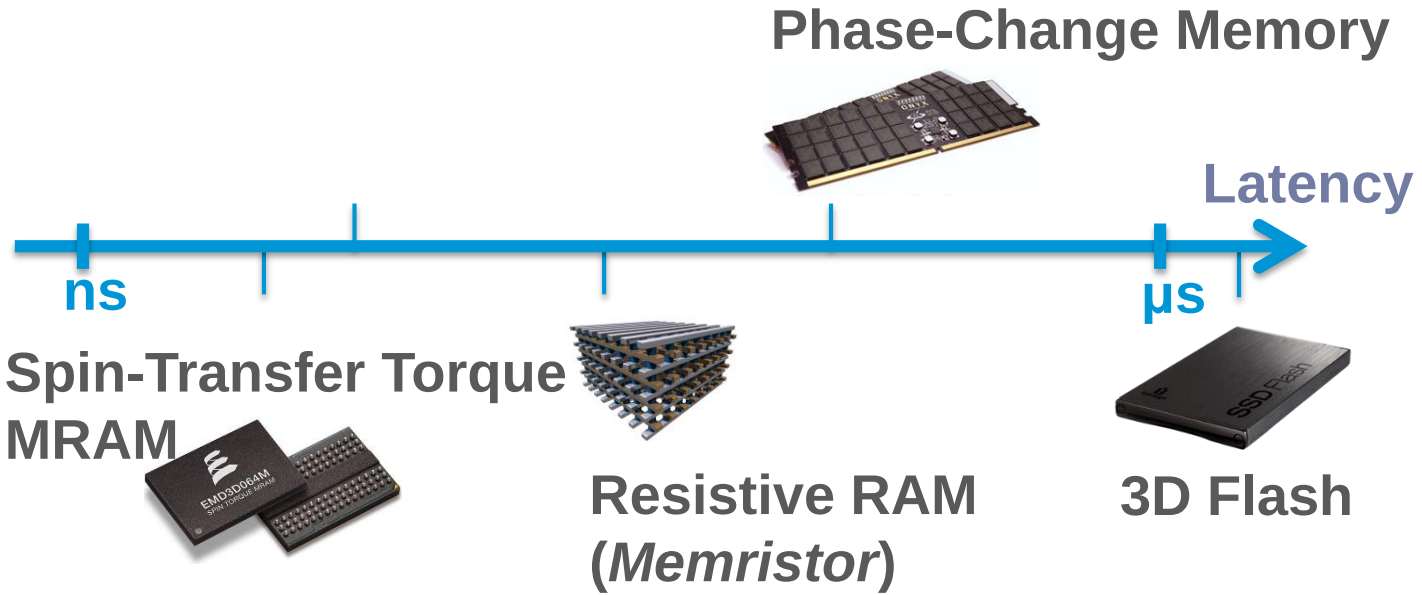
Today

Photonics destroys distance:
Enables flat, hyper-efficient networks by avoiding latency, boosting BW, $/Gb/s/m, and pj/bit are key metrics



Universal memory obsoletes this hierarchy

Beyond DRAM: Non-Volatile Memory



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*.

Architectural characteristics of The Machine

Opportunities and challenges

Many hardware threads per SoC

Very large NVRAM for both memory and storage (<1µs latency)

Significant amount of fast local DRAM

Photonic memory fabric that permits fast load/store access to NVRAM

No global cache coherence

Volatile caches: minimal instruction set architecture support for persistence

Virtual Memory: translation vs. protection?

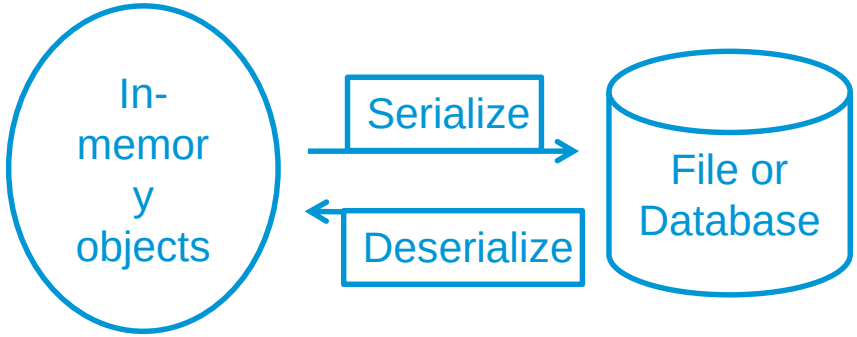
How can app developers utilize distributed persistent memory – what are the right abstractions?

The Software Revolution

Data Representations

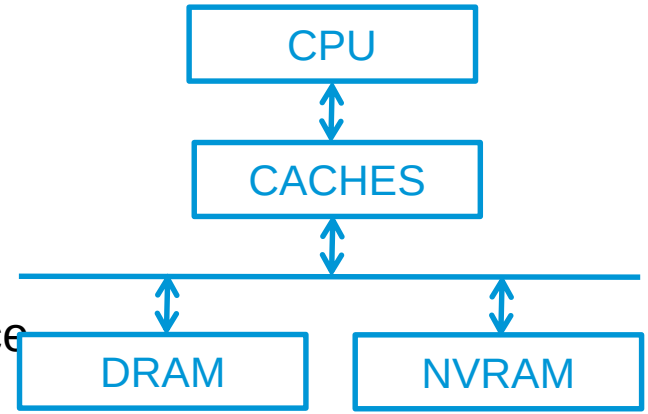
In-storage durability

- Separate object and persistent formats
- Programmability and performance issues
- Translation code error-prone and insecure
+ Clean separation of persistent state

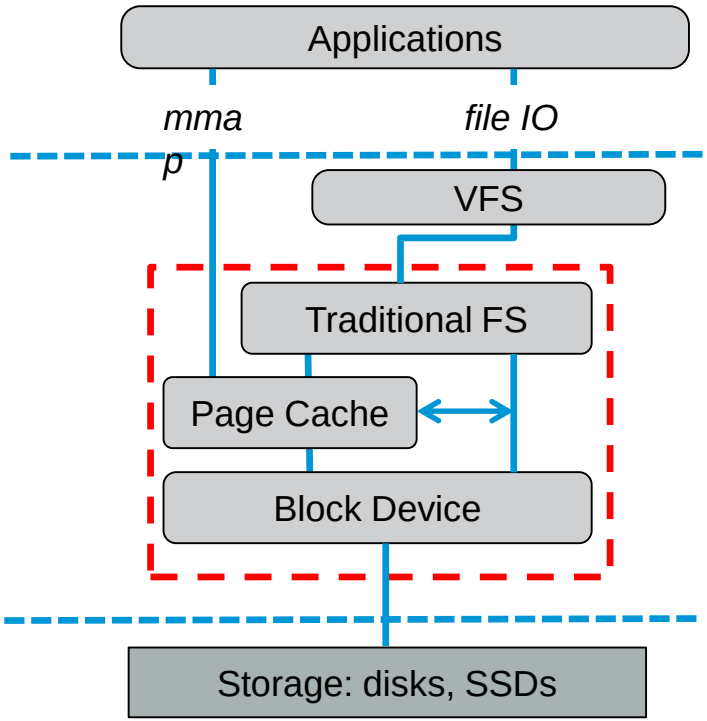


In-memory durability

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



Traditional File System



Separate storage address space

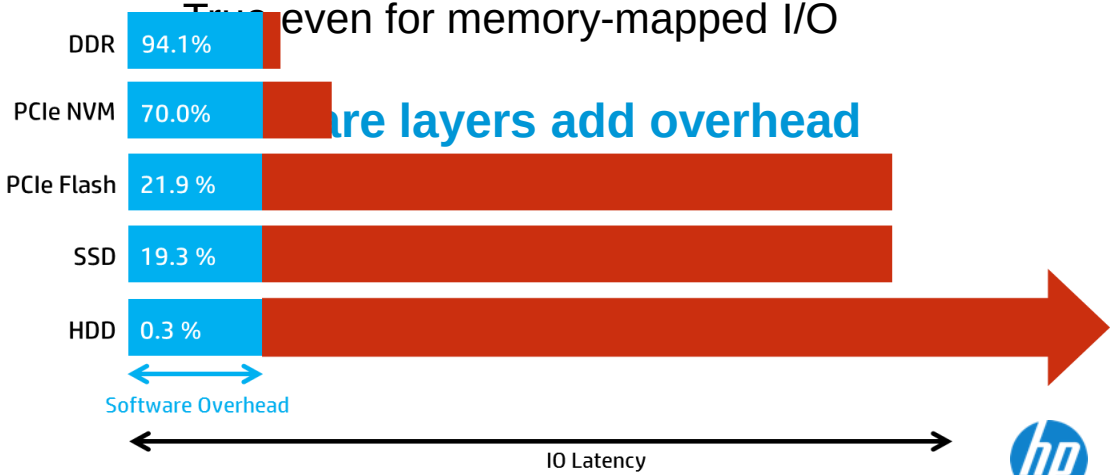
Data is copied between storage and DRAM

Block-level abstraction leads to inefficiencies

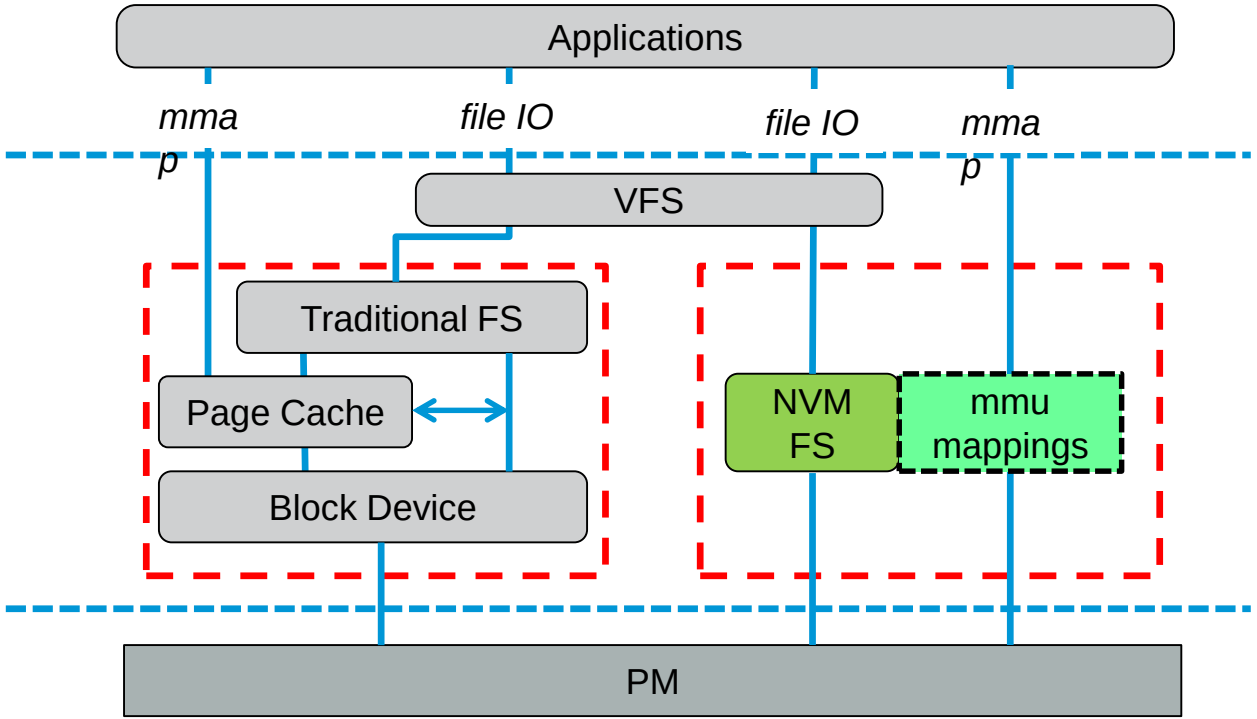
Use of page cache leads to extra copies

True even for memory-mapped I/O

are layers add overhead



NVM-optimized File System



Examples

Microsoft BPFS

Intel PMFS → DAX (pmem.io)

Low overhead access to persistent memory

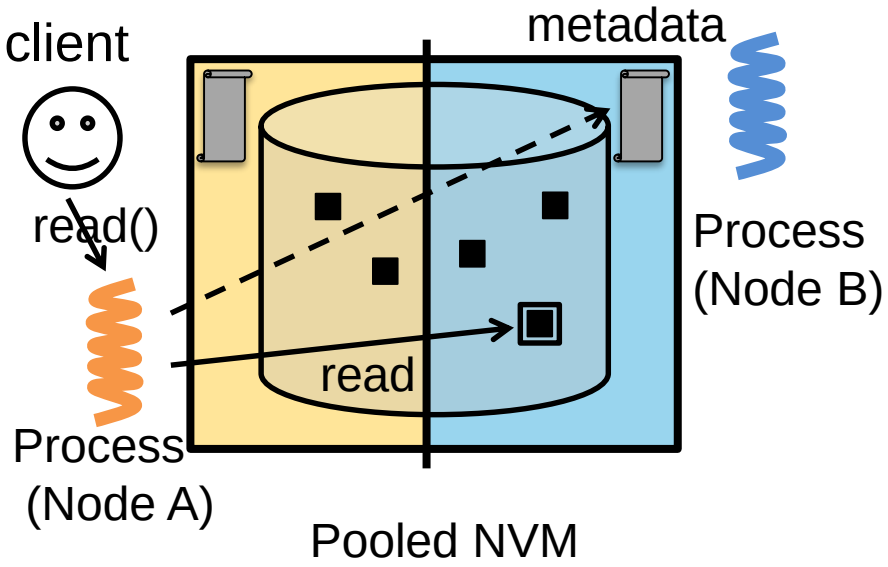
No page cache

Direct access with mmap

Leverage hardware support for consistency

Subramanya R Dulloor, et al. "System Software for Persistent Memory," *Proc. EuroSys 2014*.

NVM-optimized Distributed File System



Pooled NVM enables direct access to non-local data

Reduces need for replication

Provides more natural load balancing

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
    accounts[from] -= amount;
    accounts[to] += amount;
}
```

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

Need code that plays back undo log on restart. Getting this to work with threads and locks is very hard

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

The Atlas programming model

Programmer distinguishes persistent | transient

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

- Automatic durability support at a fine granularity, complete with recovery code
- Supports consistency of durable data derived from concurrency constructs

Protection against failures

- Process crash: works with existing architecture
- Tolerating kernel panics and power failures requires NVRAM and CPU cache flushes

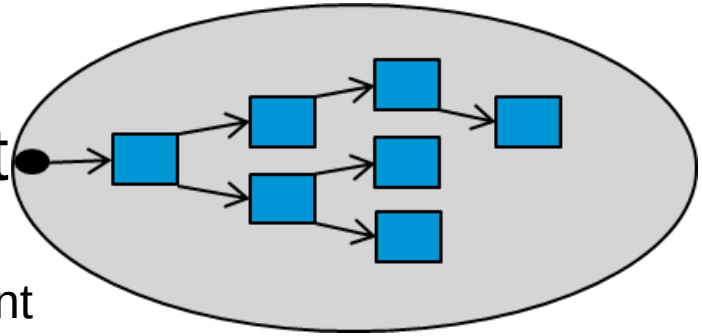
```
persistent double accounts[2];
transfer(
  int from, int to, double amount) {
  __atomic {
          accounts[from] -= amount;
          accounts[to]   += amount;
  }
}
```

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.

Persistent Regions



Named container for all persistent data

- Analogous to file-backed memory mapping
- Data outside persistent region considered transient
- Easy to slide beneath complex (legacy) software
- Transparent support preserves data integrity from crashes

Key mechanism: failure-atomic updates

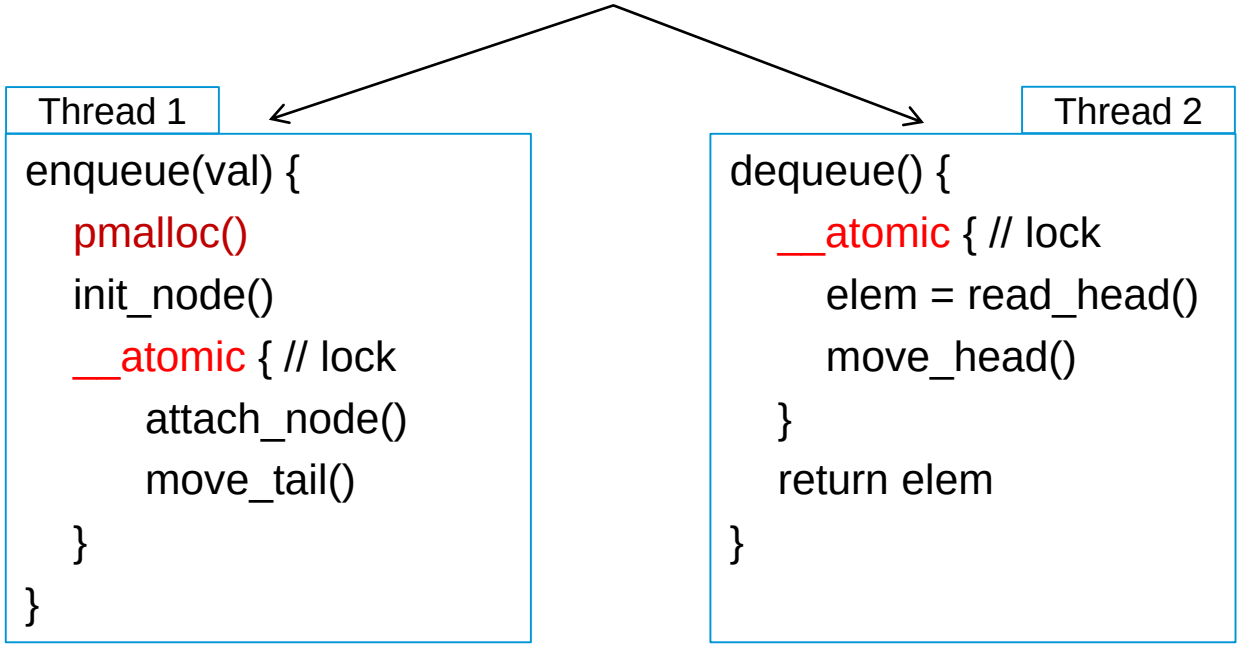
- All-or-nothing guarantee for a failure-atomic batch of updates

Admits several implementations

- Failure-atomic update of files via msync/fsync *[Eurosys2013]*
- Lexically-scoped atomic{} sections with durability semantics *[Mnemosyne, Nvheaps, ASPLOS11]*
- Durability support for lock-based critical sections *[Atlas, OOPSLA'14]*

Example: a Persistent Queue

```
pr = find_or_create_persistent_region("queue");
persistent q = get_root(pr);
if (q is absent) initialize q and call set_root
```



Role of Compiler and Runtime

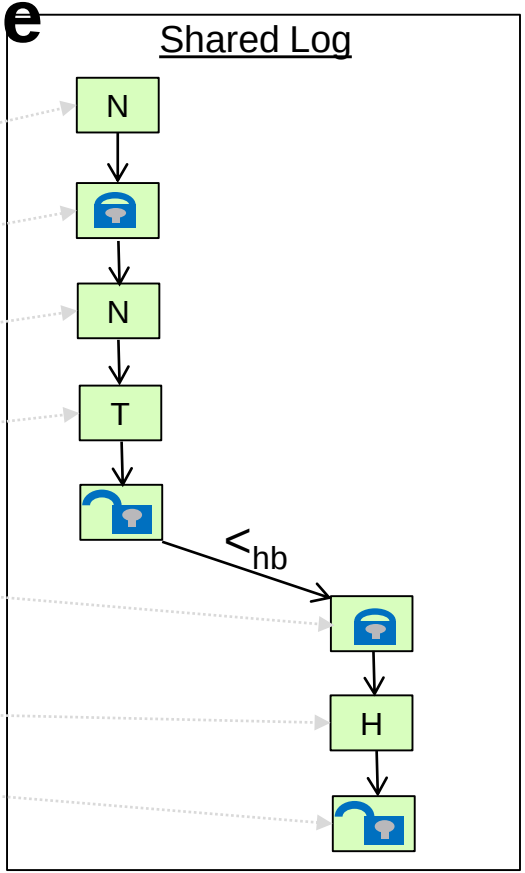
Time

Thread 1

```
enqueue(val) {
    pmalloc()
    init_node()
    L.lock()
        attach_node()
        move_tail()
    L.unlock()
}
```

Thread 2

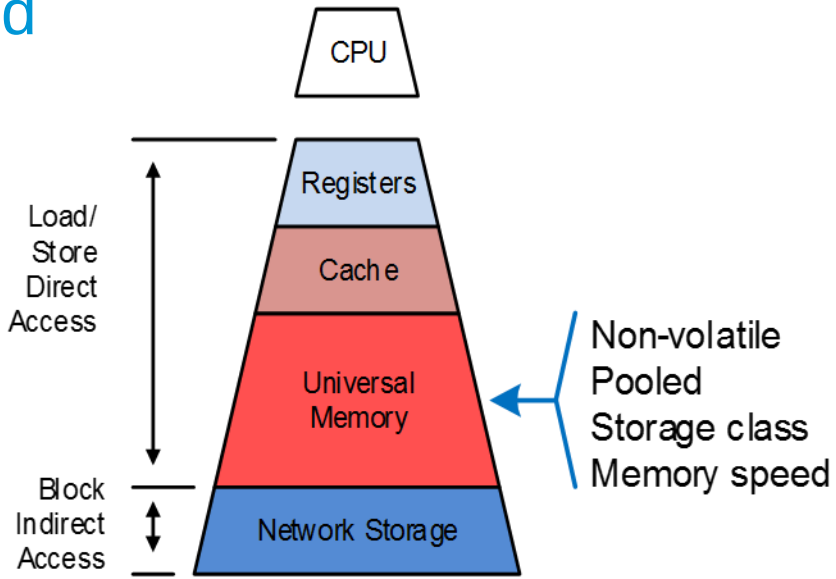
```
dequeue() {
    L.lock()
        elem = read_head()
        move_head()
    L.unlock()
    return elem
}
```



Wrapping up

Universal memory is coming

Computing shifts to a persistent world



Everything changes…

Hardware

- Memory controller

Architecture

- Coherence/sharing model
- Consistency model
- Error handling, RAS

Software

- OS, memory management
- Compilers and runtime
- Algorithms and data structures
- Storage hierarchy
- Applications
- Security and Protection

Learn more about The Machine

Developed with HP Labs

The Machine provides new computing architecture

Specialized SoCs + massive shared NVM pool + photonic interconnects

Many opportunities for OS and application software innovation

Where to look for more information

http://www.hpl.hp.com/research/systems-research/themachine/

HP Discover 2014 talks on The Machine

- HP Labs Director Martin Fink's announcement: https://www.youtube.com/watch?v=Gxn5ru7klUQ
- Kim Keeton’s talk on technologies: https://www.youtube.com/watch?v=J6_xg3mHnng

Dejan Milojicic’s keynote at Linaro Connect: http://connect.linaro.org/hkg15/

Paolo Faraboschi’s keynote at HPCA/PPoPP/CGO

Thank You