

Hewlett Packard
Enterprise

Memory-Driven Computing

Kimberly Keeton

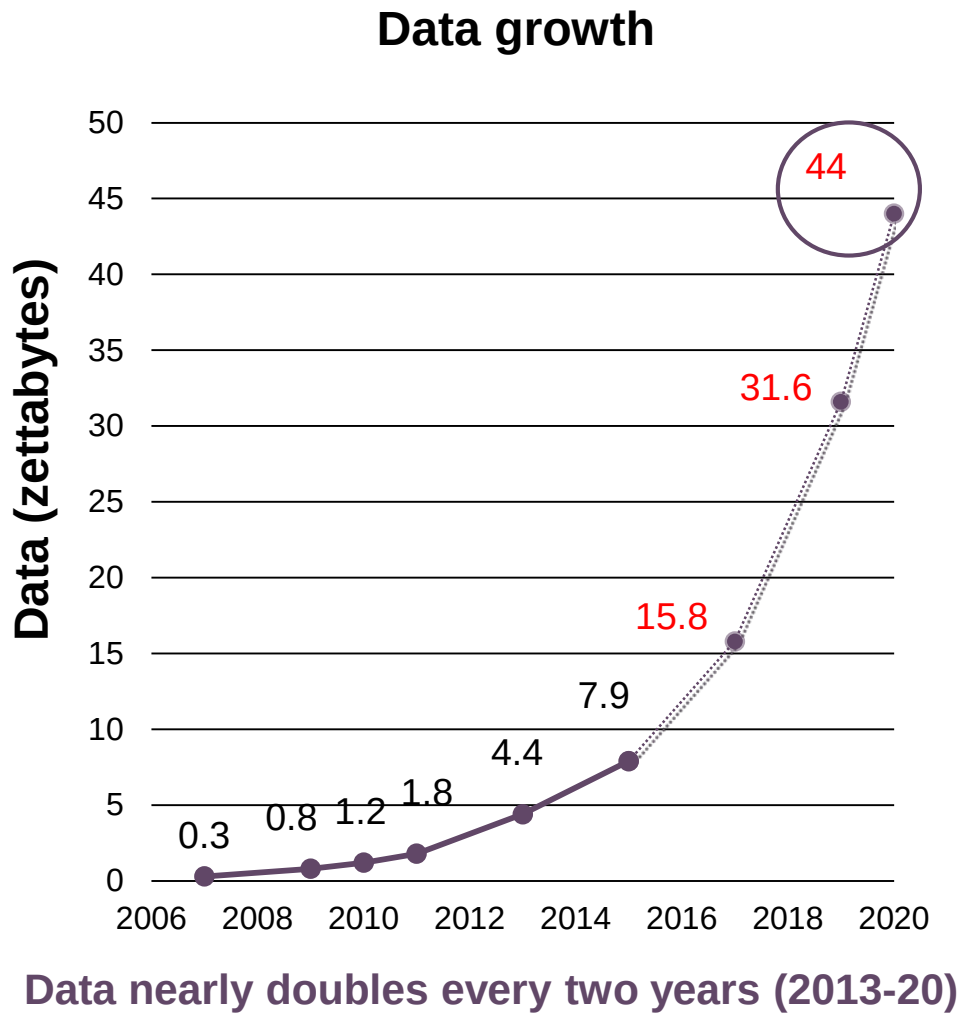
Distinguished Technologist
kimberly.keeton@hpe.com

USENIX FAST 2017



Hewlett Packard
Labs

Need answers quickly and on bigger data



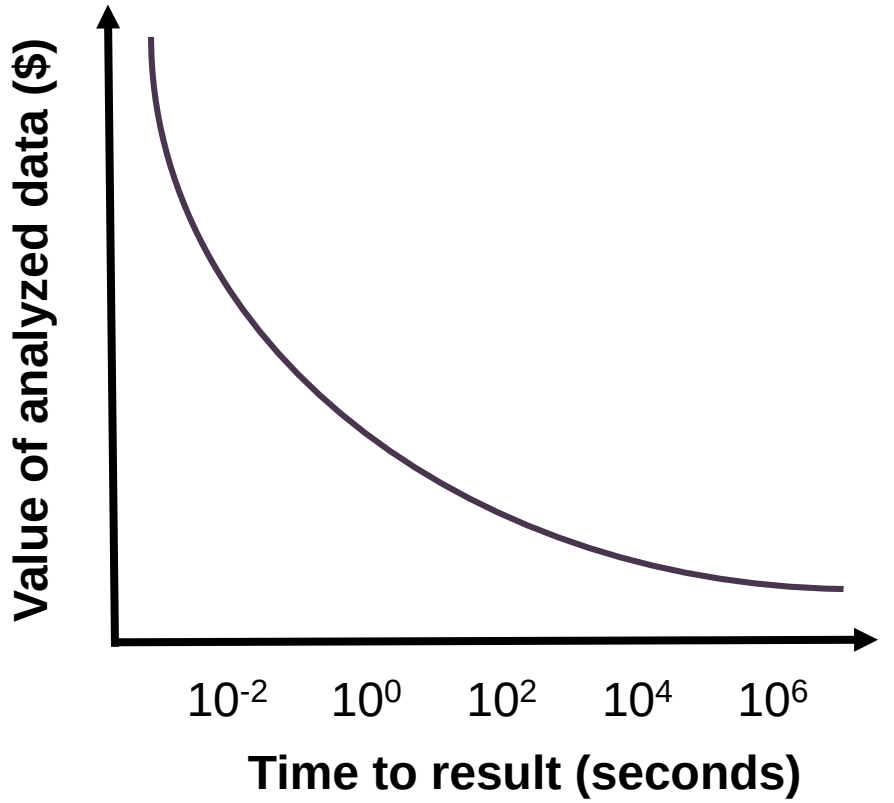
Data nearly doubles every two years (2013-20)

By 2020:

8B people

20B connected devices

100B infrastructure devices

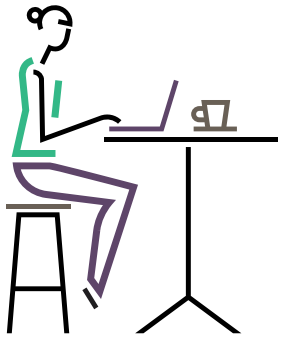


Hewlett Packard Enterprise

©Copyright 2017 Hewlett Packard Enterprise Company

A need for something new…

System of record



Electronic record of event
Ex: banking
Mediated by people
Structured data
Accurate, traceable

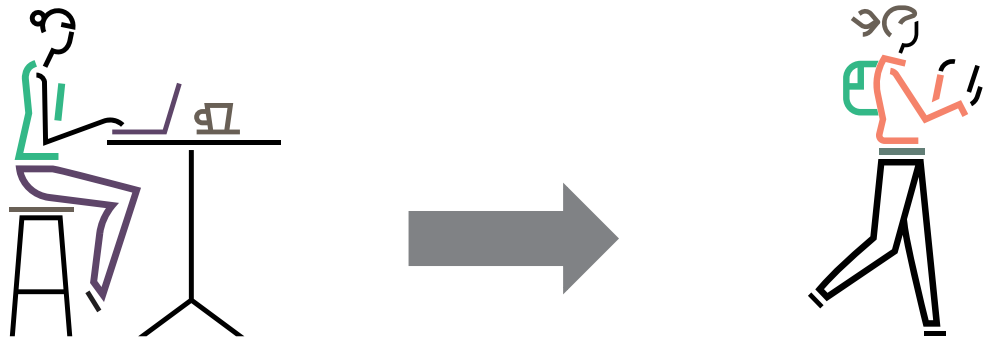
Hewlett Packard Enterprise

©Copyright 2017 Hewlett Packard Enterprise Company

A need for something new…

System of record

System of engagement



Electronic record of event	Interactive apps for humans
Ex: banking	Ex: social media
Mediated by people	Interactive
Structured data	Unstructured data
Accurate, traceable	Complete accuracy not required

Hewlett Packard Enterprise

©Copyright 2017 Hewlett Packard Enterprise Company

A need for something new…

System of record | **System of engagement** | **System of action**

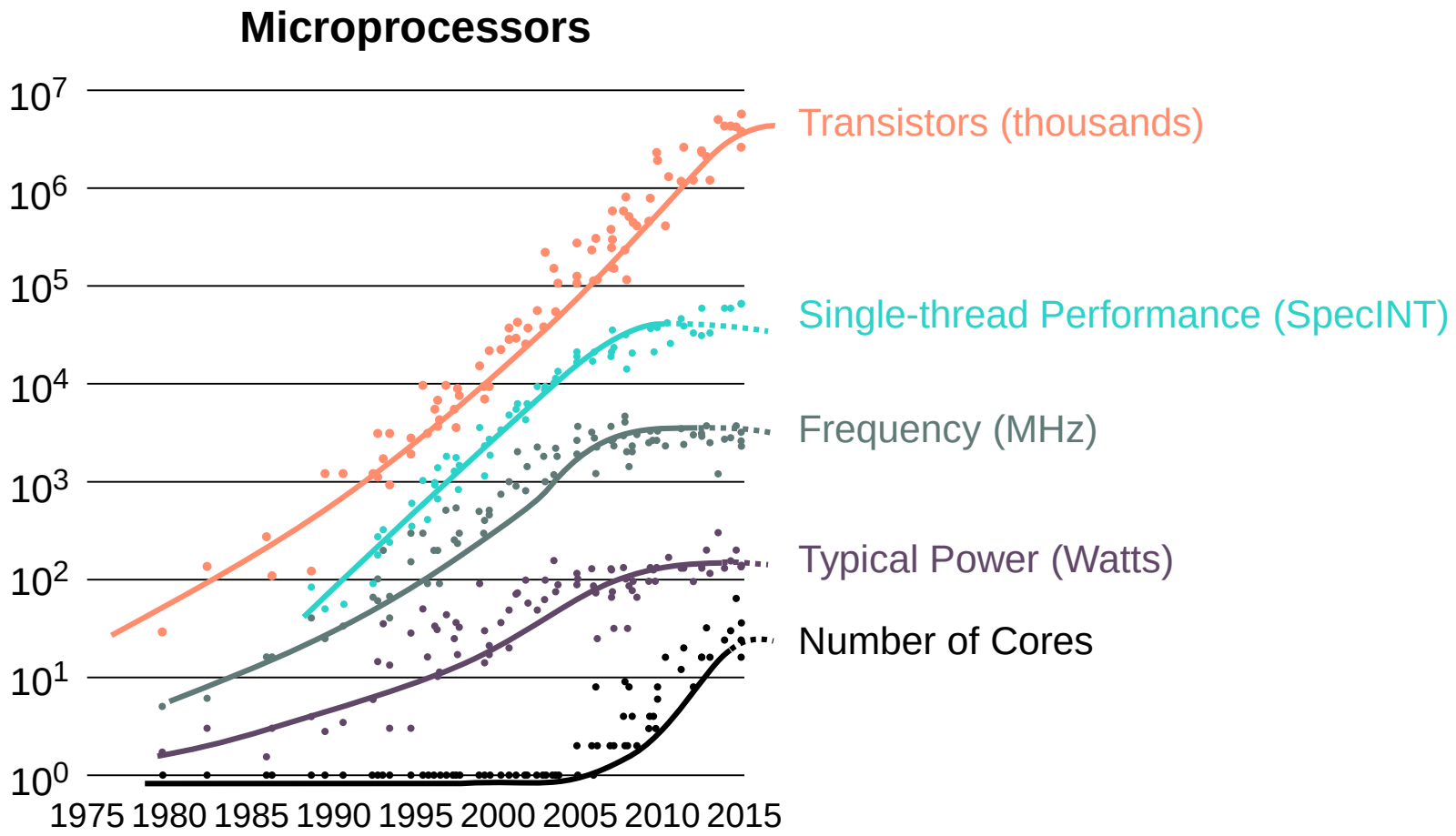


Electronic record of event	Interactive apps for humans	Machines making decisions
Ex: banking	Ex: social media	Ex: smart and self-driving cars
Mediated by people	Interactive	Real time, low latency
Structured data	Unstructured data	Structured and unstructured data
Accurate, traceable	Complete accuracy not required	Accurate and traceable

Hewlett Packard Enterprise

©Copyright 2017 Hewlett Packard Enterprise Company

The New Normal: traditional compute not keeping up

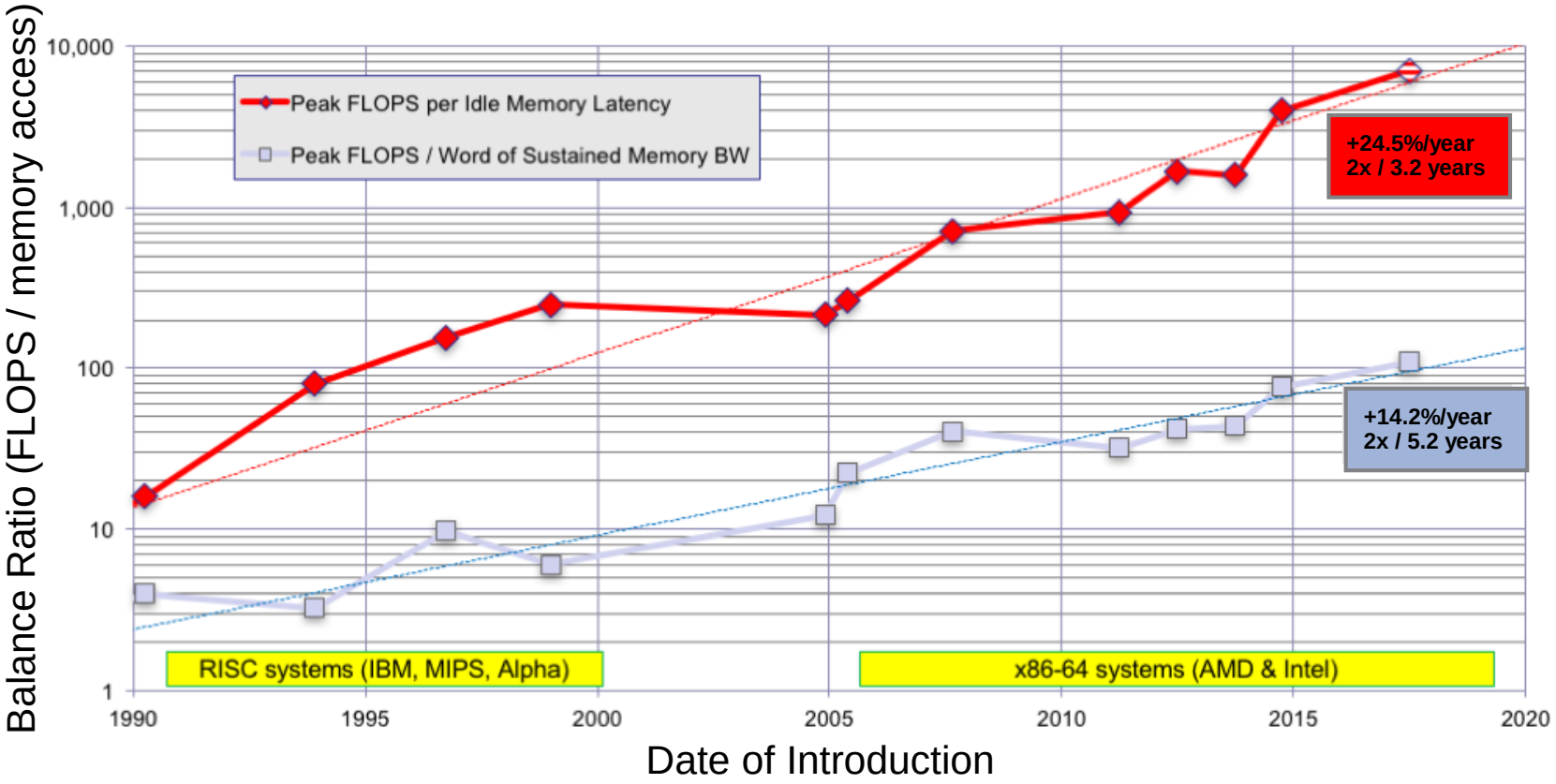


Future microprocessor improvements limited by sunset of Moore's Law

Hewlett Packard Enterprise

©Copyright 2017 Hewlett Packard Enterprise Company

The New Normal: memory isn't keeping up

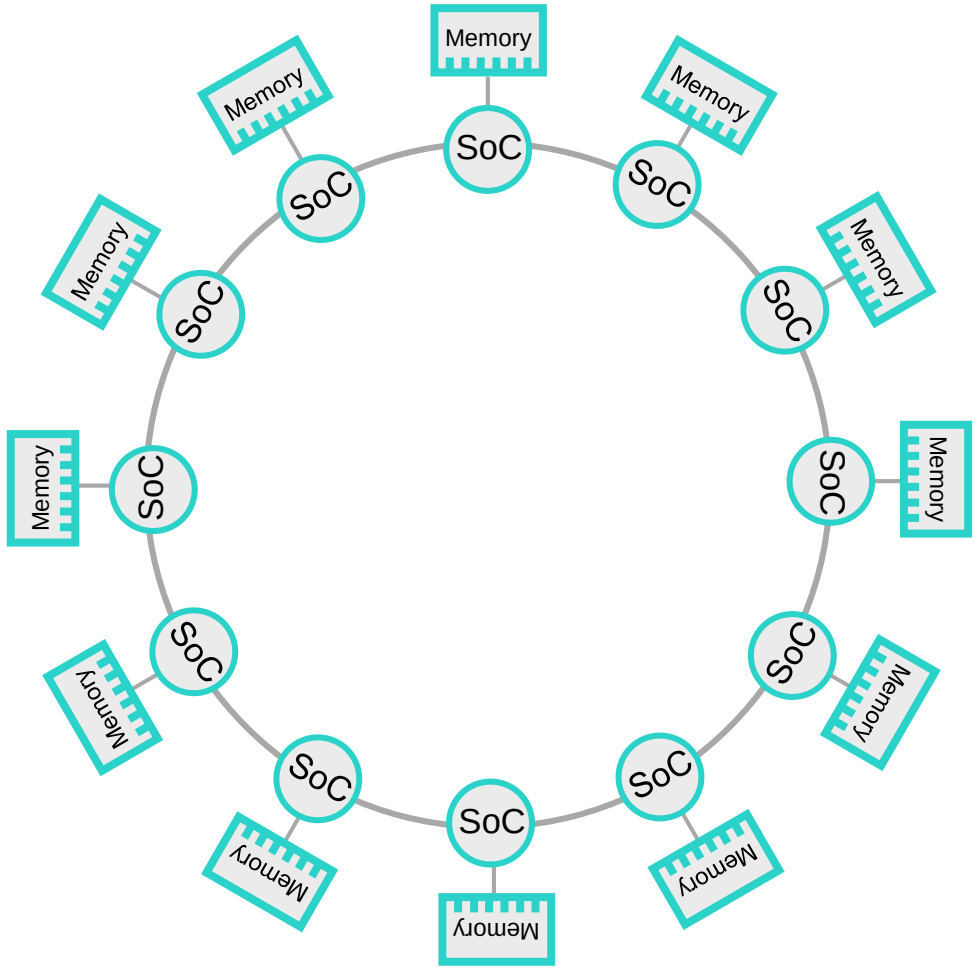


Processors are becoming ***increasingly imbalanced*** with respect to data motion

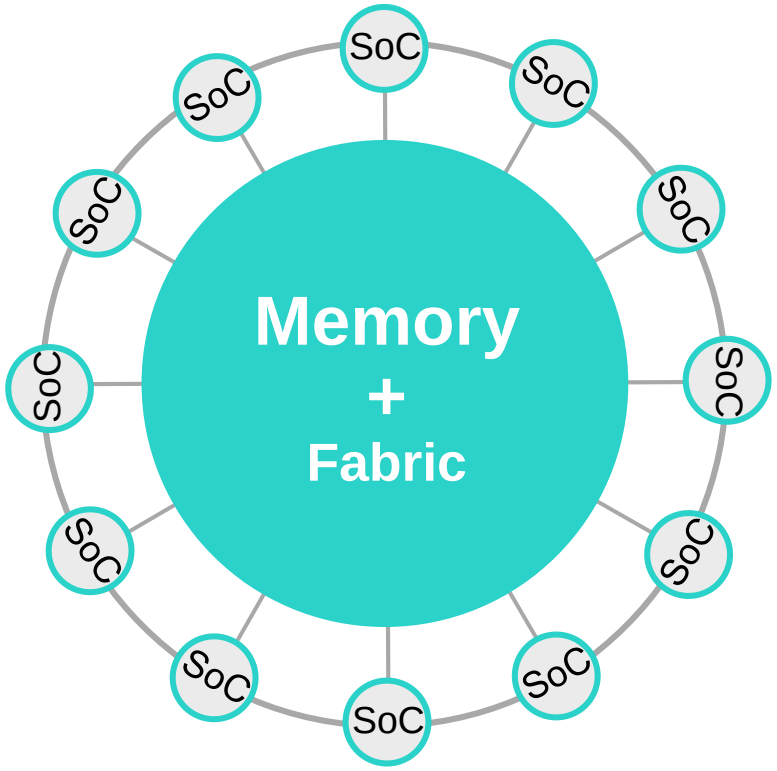
J. McCalpin, "Memory Bandwidth and System Balance in HPC Systems," Invited talk at *SC16*, 2016.
http://sites.utexas.edu/jdm4372/2016/11/22/sc16-invited-talk-memory-bandwidth-and-system-balance-in-hpc-systems/

Hewlett Packard Enterprise

©Copyright 2017 Hewlett Packard Enterprise Company



From Processor-Centric Computing…



…to Memory-Driven Computing

Hewlett Packard Enterprise

©Copyright 2017 Hewlett Packard Enterprise Company

Core Memory-Driven Computing components

Fast, persistent memory

Fast memory fabric

Task-specific processing

New software



Hewlett Packard Enterprise

©Copyright 2017 Hewlett Packard Enterprise Company

Outline

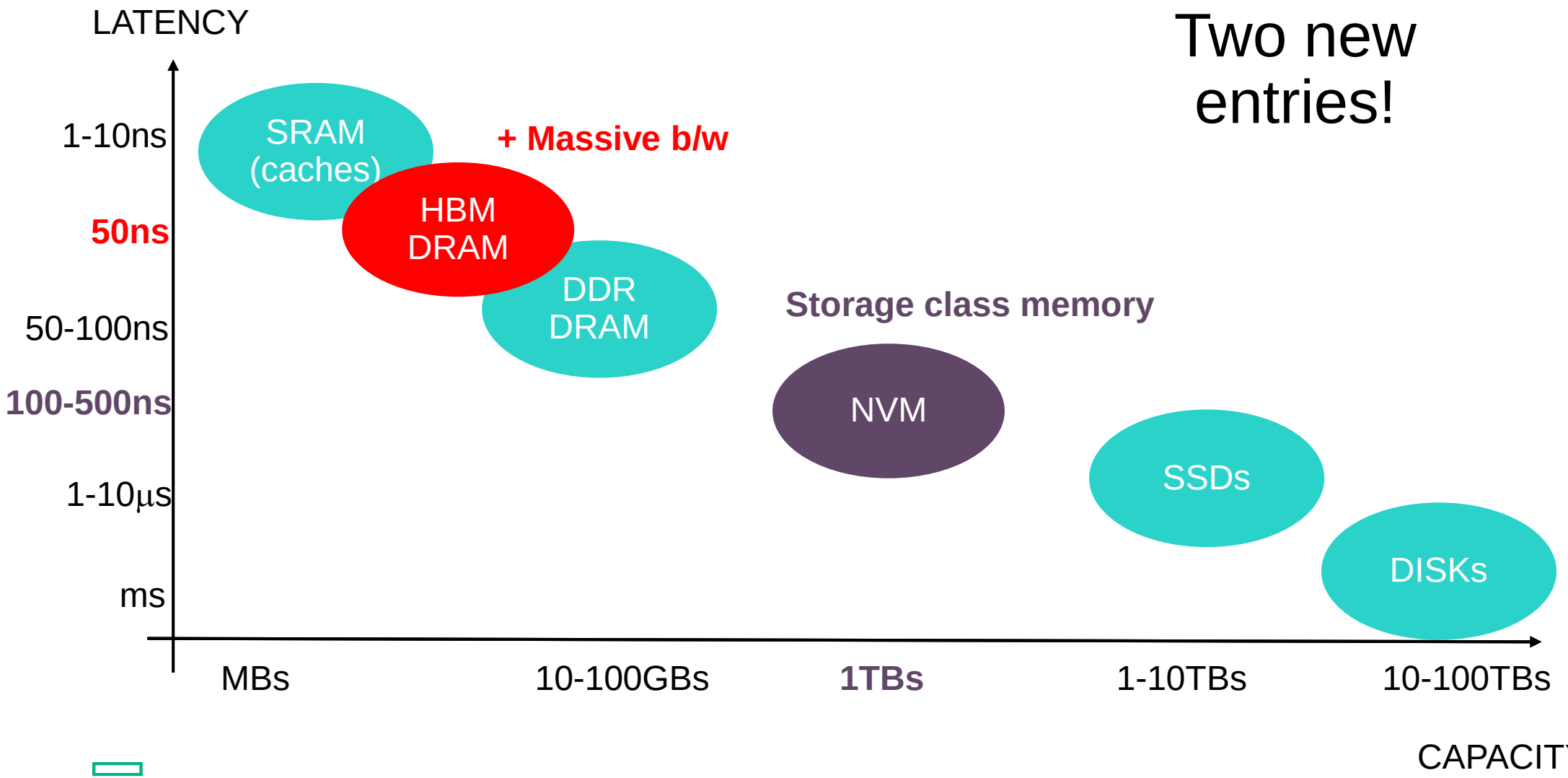
- Overview: Memory-Driven Computing
- Technology trends enabling Memory-Driven Computing
- How Memory-Driven Computing benefits applications
- How do we get to Memory-Driven Computing
  - Data management and programming models
- Memory-Driven Computing implications for the FAST community
- Summary

©Copyright 2017 Hewlett Packard Enterprise Company

Technology trends enabling Memory-Driven Computing

Hewlett Packard Enterprise

Memory + storage hierarchy technologies

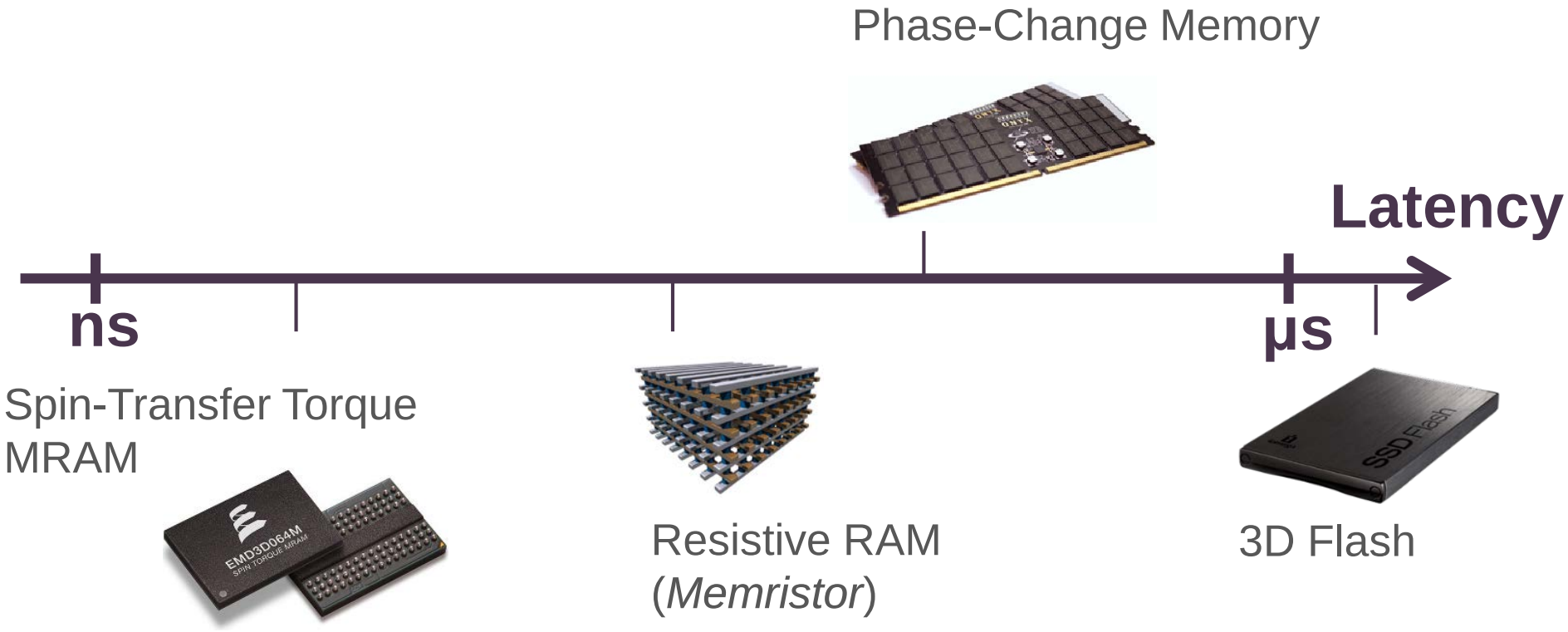


Two new entries!

Hewlett Packard Enterprise

©Copyright 2017 Hewlett Packard Enterprise Company

Non-Volatile Memory (NVM)



– Persistently stores data
– Access latencies comparable to DRAM
– Byte addressable (load/store) rather than block addressable (read/write)
– More energy efficient and denser than DRAM

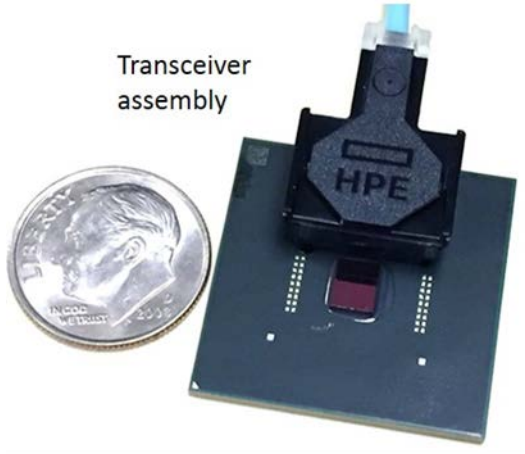
Haris Volos, et al. "Aerie: Flexible File-System Interfaces to Storage-Class Memory," *Proc. EuroSys 2014*.

Hewlett Packard Enterprise

©Copyright 2017 Hewlett Packard Enterprise Company

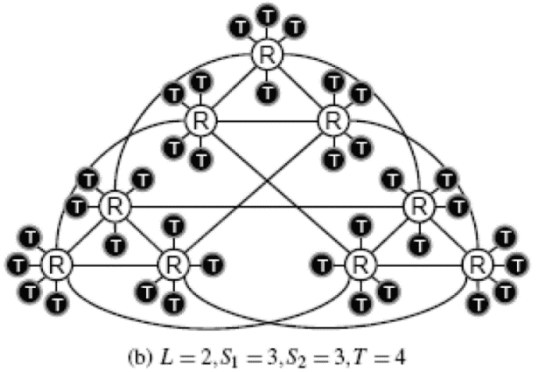
Interconnect advances

- Photonic interconnects
  - Ex: Vertical Cavity Surface Emitting Lasers (VCSELs)
  - 4 λ Coarse Wavelength Division Multiplexing (CWDM)
  - 100Gbps/fiber; 1.2Tbps with 12 fibers
  - Low power ~ < 5pJ/bit (target)
  - Low cost << $1/Gbps



VCSEL optics

- High-radix switches enable low-diameter network topologies
  - Pooled NVM will appear at near-uniform low latency



(b) $L = 2, S_1 = 3, S_2 = 3, T = 4$

HyperX topology

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

Hewlett Packard Enterprise

©Copyright 2017 Hewlett Packard Enterprise Company

Gen-Z: open systems interconnect standard

http://www.genzconsortium.org

– Open standard for memory-semantic interconnect

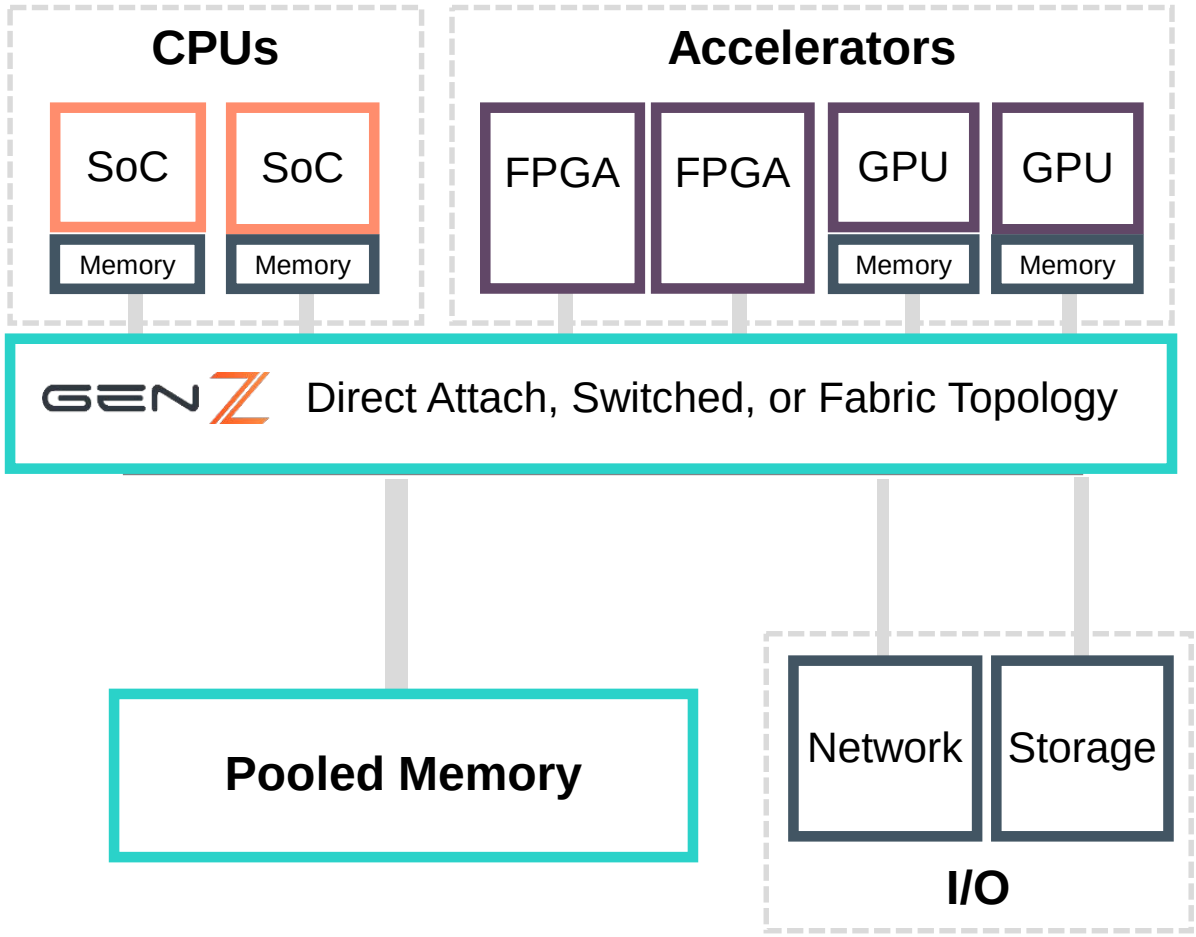
– Members: 25+ companies covering SoC, memory, I/O, networking, mechanical, system software, etc.

– Motivation
  – Emergence of low-latency storage class memory
  – Demand for large capacity, rack-scale resource pools and multi-node architectures

– Memory semantics
  – All communication as memory operations (load/store, put/get, atomics)

– High performance
  – Tens to hundreds GB/s bandwidth
  – Sub-microsecond load-to-use memory latency

– *Draft spec available for public download*

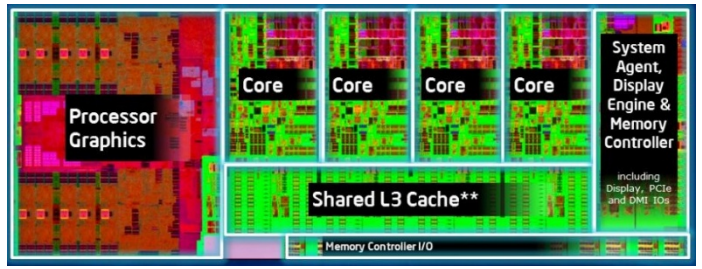


Hewlett Packard Enterprise

©Copyright 2017 Hewlett Packard Enterprise Company

Heterogeneous compute

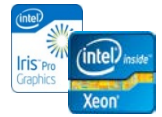
- Dark silicon effects
  - Microprocessor designs are limited by power, not area
  - Solution: combination of function blocks that are selectively activated



- Task-specific accelerators augment CPU compute
  - Examples: GPUs, FPGAs, ASICs
  - Enables higher energy efficiency



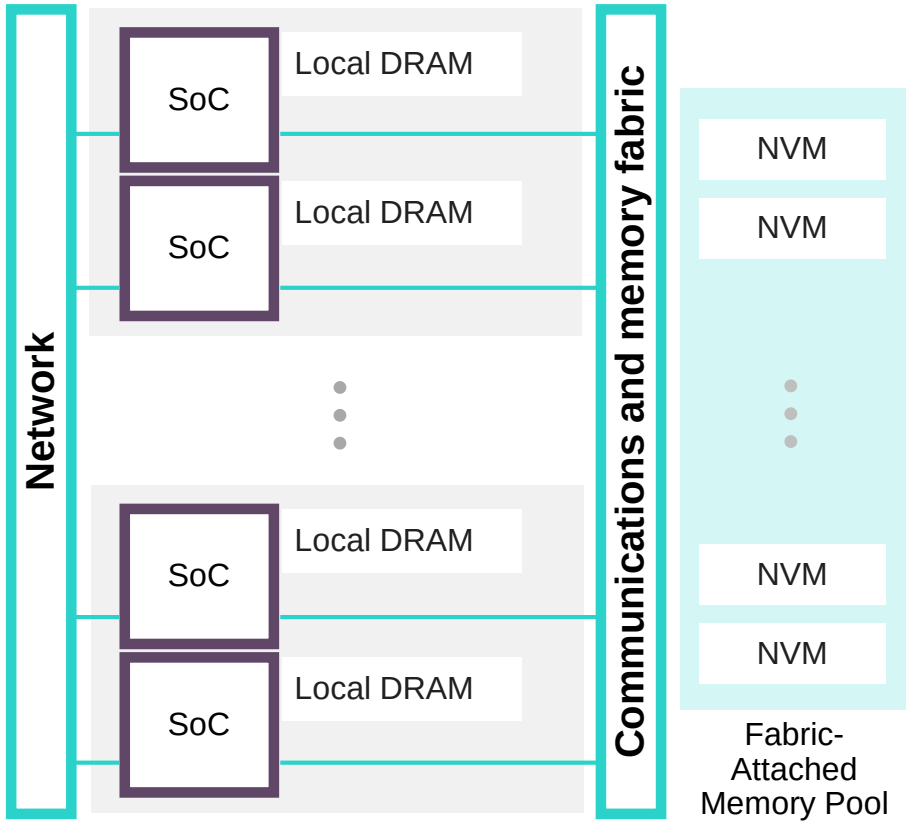
HPE Edgeline ProLiant m710x



Hewlett Packard Enterprise

©Copyright 2017 Hewlett Packard Enterprise Company

Putting it all together: Memory-Driven Computing

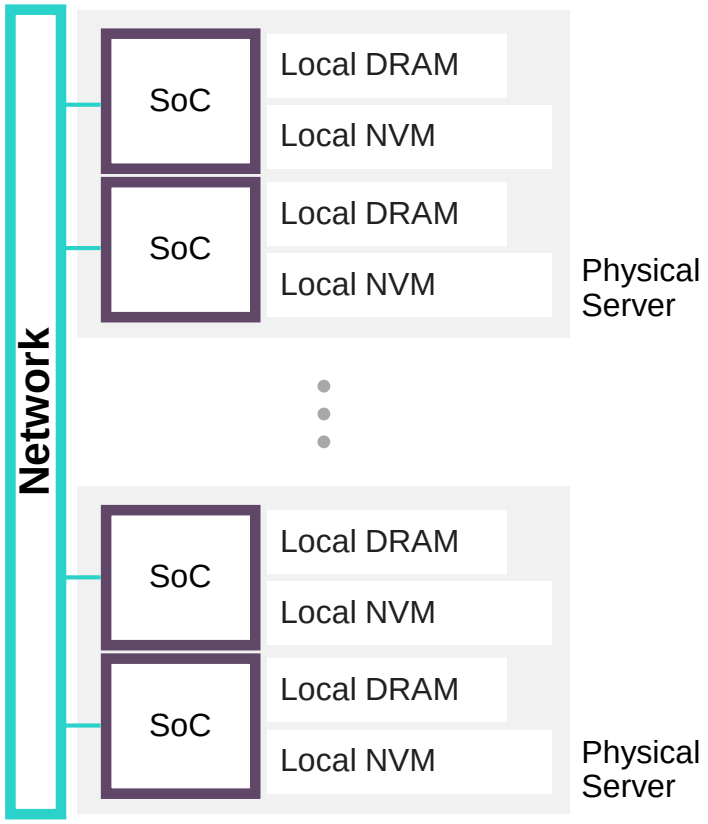


- **Converging memory and storage**
  - Byte-addressable NVM replaces hard drives and SSDs
- **Resource disaggregation leads to shared memory pool**
  - Fabric-attached memory pool is accessible by all compute resources
  - Optical networking provides near-uniform low latency
- **Distributed heterogeneous compute resources**
- **Local volatile memory provides lower latency, high performance tier**
- **Software**
  - Memory-speed persistence
  - Direct, unmediated access to all fabric-attached NVM across the memory fabric
  - Non-coherent accesses between compute nodes

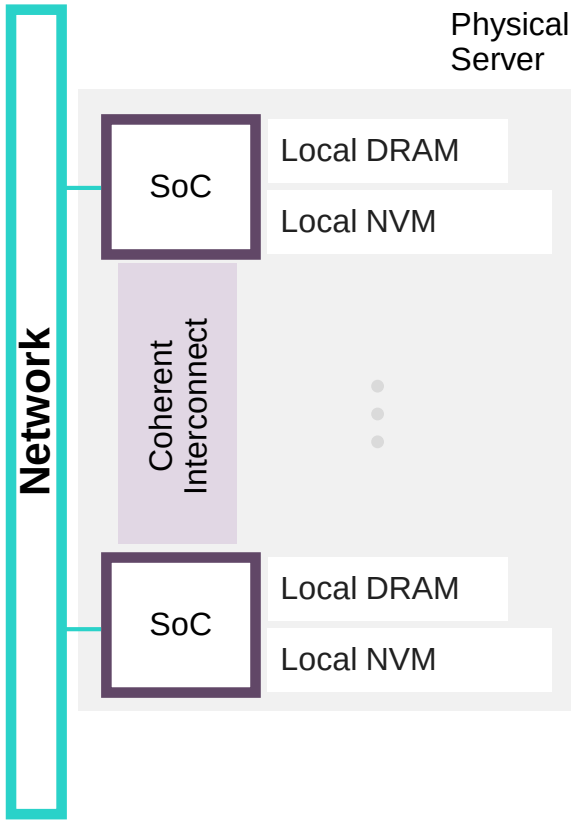
Hewlett Packard Enterprise

©Copyright 2017 Hewlett Packard Enterprise Company

Memory-Driven Computing in context



Shared nothing

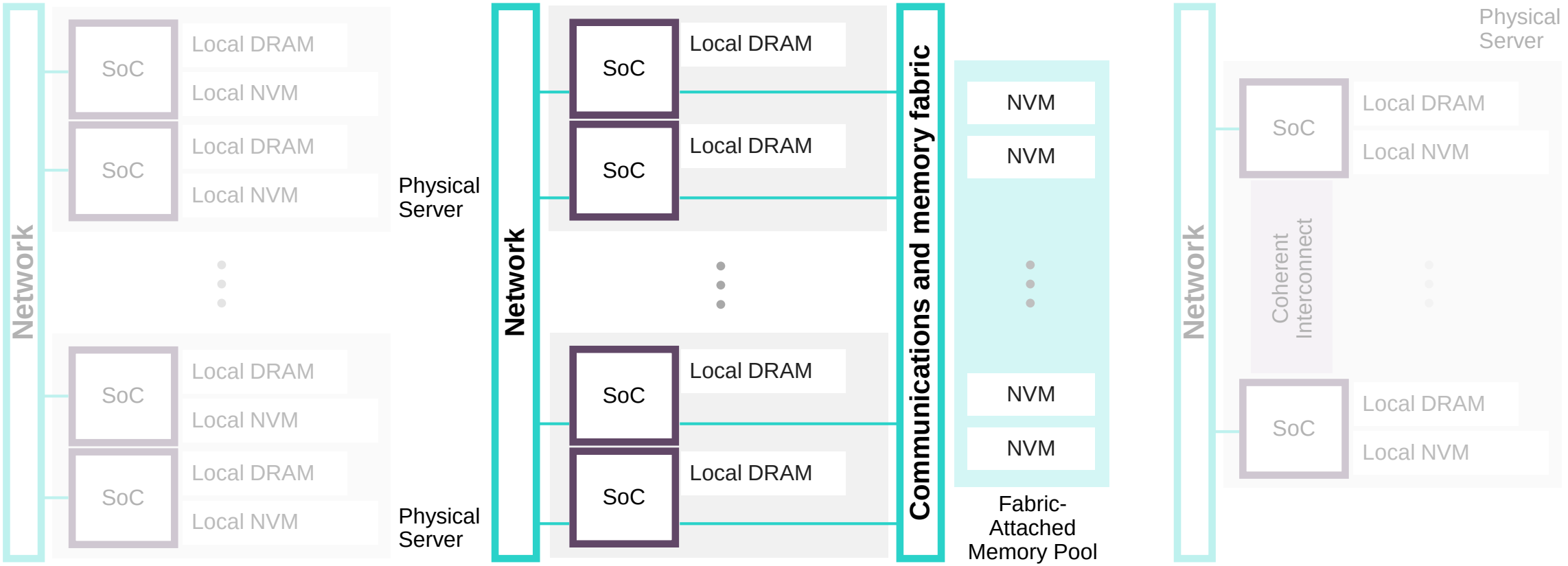


Shared everything

Hewlett Packard Enterprise

©Copyright 2017 Hewlett Packard Enterprise Company

Memory-Driven Computing in context



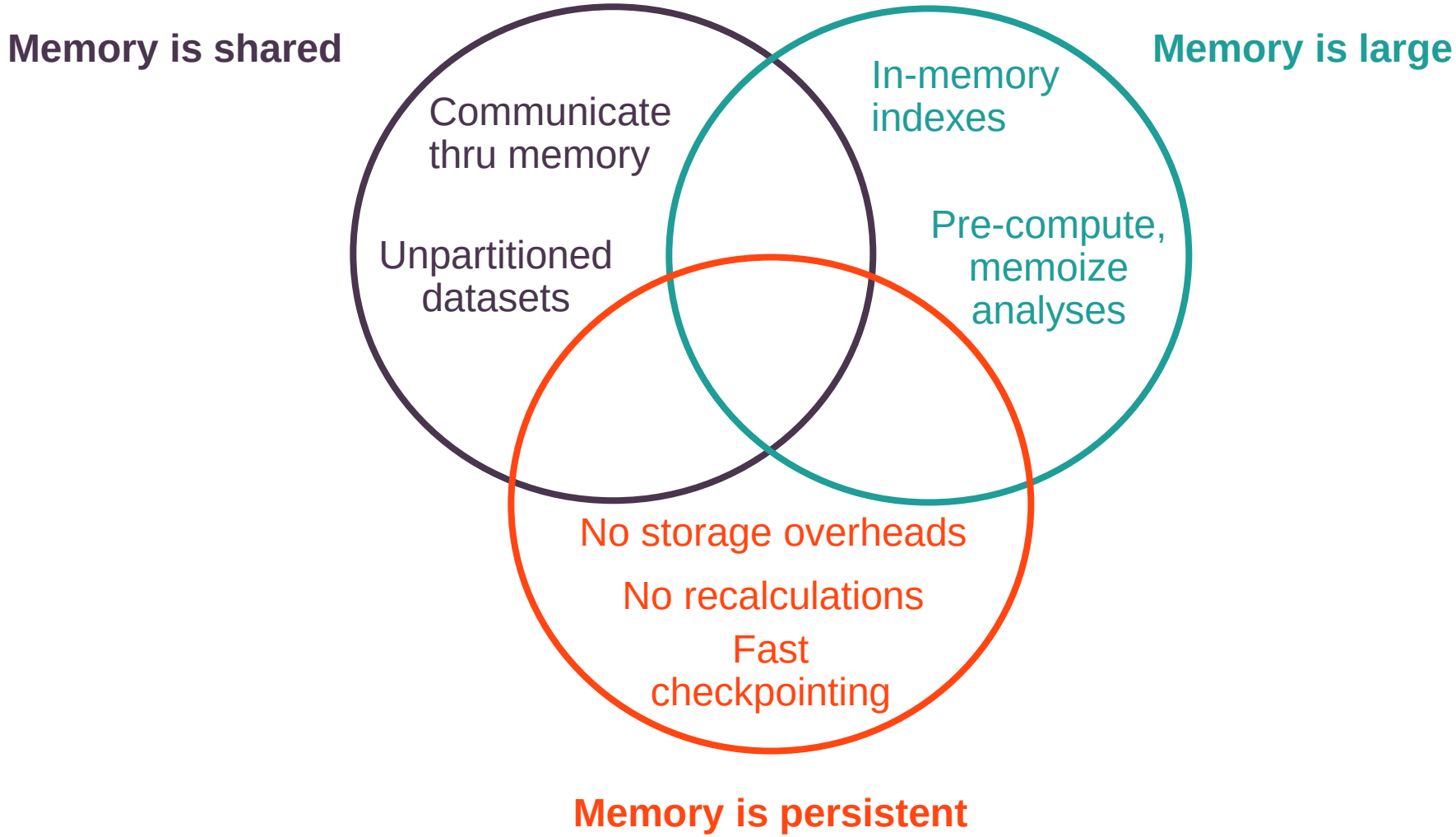
Hewlett Packard Enterprise

©Copyright 2017 Hewlett Packard Enterprise Company

How Memory-Driven Computing benefits applications

Hewlett Packard
Enterprise

Memory-Driven Computing benefits applications



Hewlett Packard Enterprise

©Copyright 2017 Hewlett Packard Enterprise Company

Large in-memory processing for Spark

Spark with Superdome X



Our approach:

– In-memory data shuffle

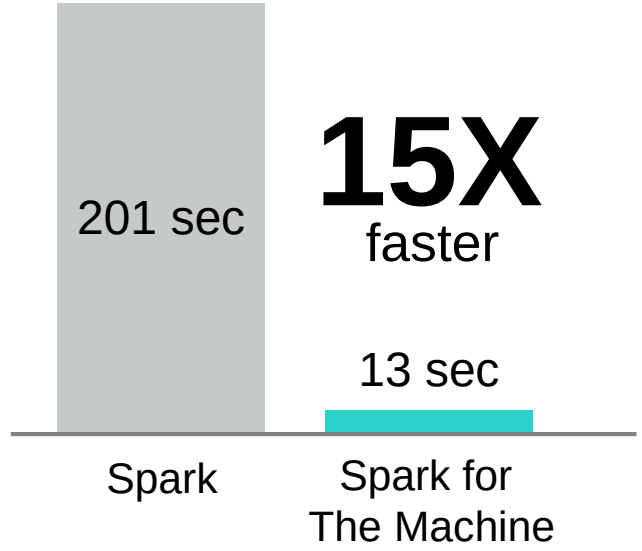
– Off-heap memory management
  – Reduce garbage collection overhead
  – Exploit large NVM pool for data caching of per-iteration data sets

– Use case: predictive analytics using GraphX

– Superdome X: 240 cores, 12 TB DRAM

https://github.com/HewlettPackard/sparkle

Dataset 1: web graph
101 million nodes
1.7 billion edges



Dataset 2: synthetic
1.7 billion nodes
11.4 billion edges

Spark for The Machine: 300 sec
Spark: *does not complete*

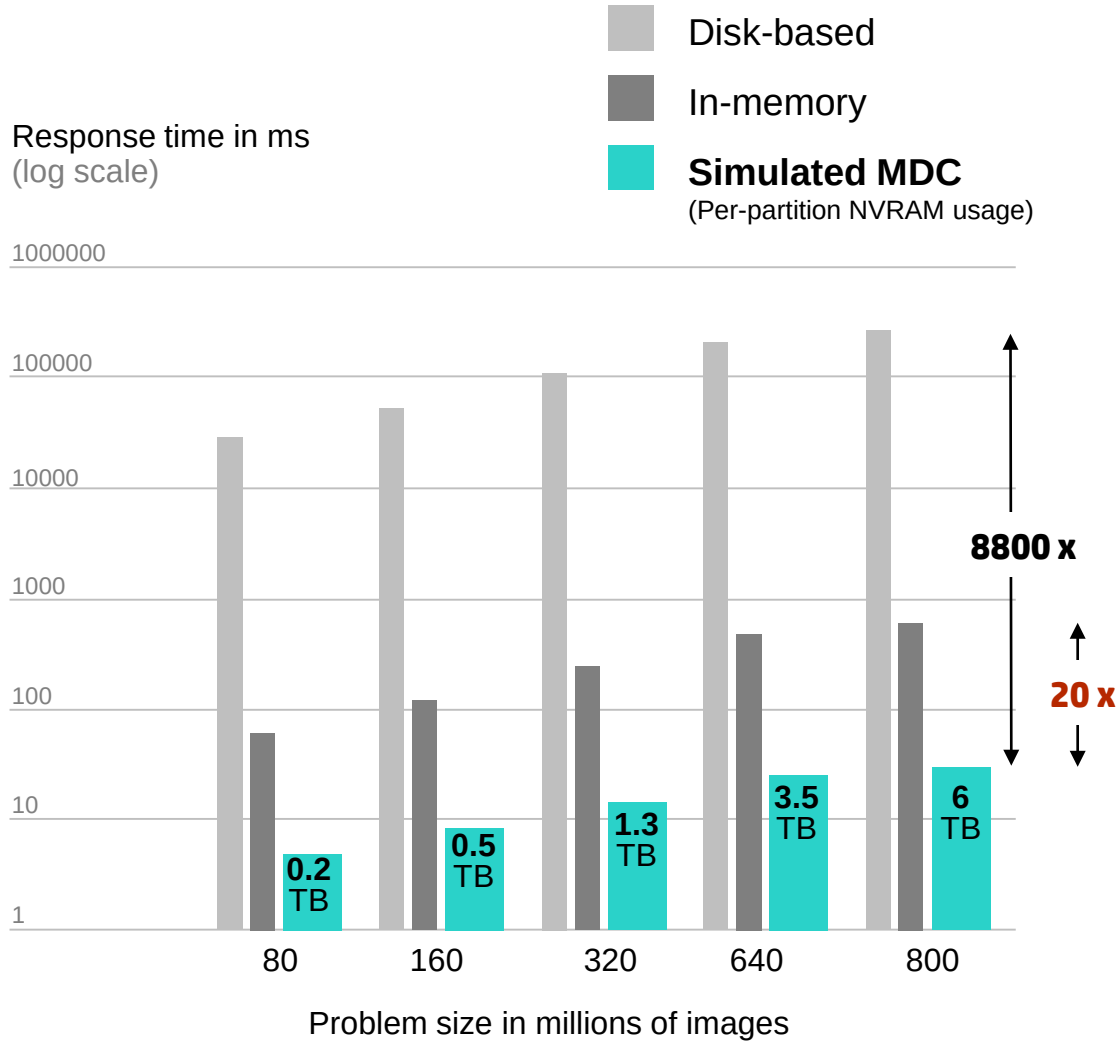
Hewlett Packard
Enterprise

©Copyright 2017 Hewlett Packard Enterprise Company

Extreme similarity search

Buying speed with large persistent memory

– Search for similar items in high-dimensional space
  – Ex: image search, e-commerce fraud mitigation
– Linear scan over feature vectors too slow for interactive queries
– Memory-Driven Computing: locality-sensitive hashing
  – Partition data, build per-partition index, search indexes in parallel, aggregate results
  – Index size depends on desired accuracy: typically large
– Comparison points
  – Disk-based platform using Hadoop
  – In-memory: linear search on feature vectors in DRAM
  – Simulated MDC: locality-sensitive hashing indexes in emulated fabric-attached NVM
– MDC outperforms alternatives by orders of magnitude



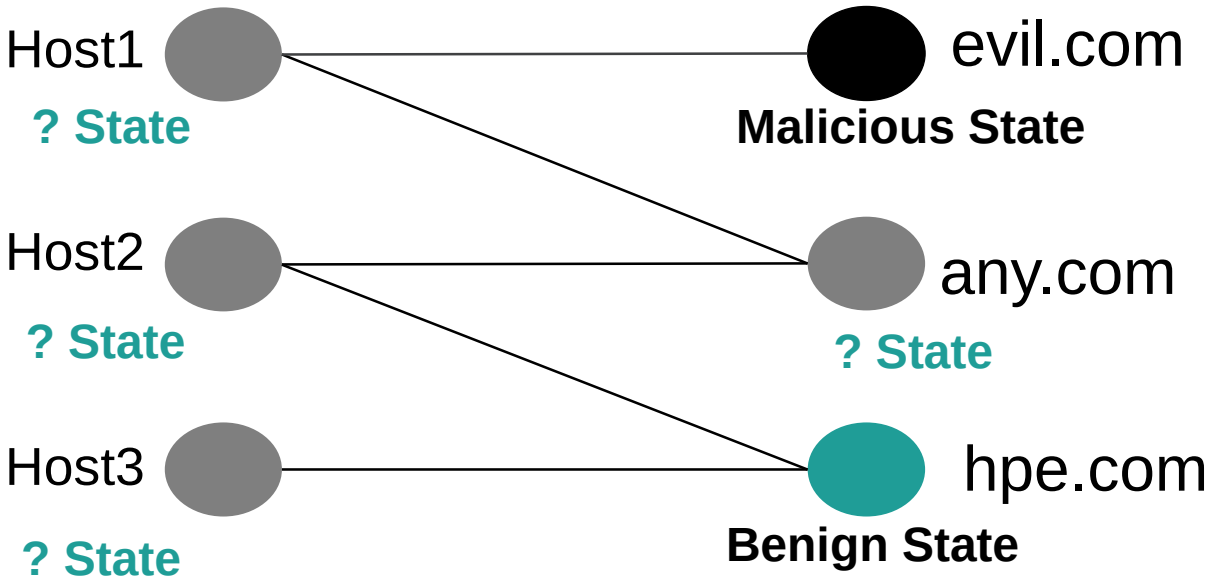
K. Viswanathan, M. Kim, J. Li, M. Gonzalez, “A memory-driven computing approach to high-dimensional similarity search,” Hewlett Packard Labs Technical Report HPE-2016-45, May 2016. Open source coming soon at https://www.labs.hpe.com/the-machine/the-machine-distribution

Hewlett Packard Enterprise

©Copyright 2017 Hewlett Packard Enterprise Company

Large-scale graph inference

- Large-scale graph inference
  - Compute probabilities across whole graph based on a small known set of vertices
  - Popular algorithms like belief propagation, Gibbs sampling, label propagation
  - Ex: malware detection, online advertising
- Challenges
  - Expensive: random data accesses, locking, CPU operations
  - Network-based synchronization overheads for distributed graph processing



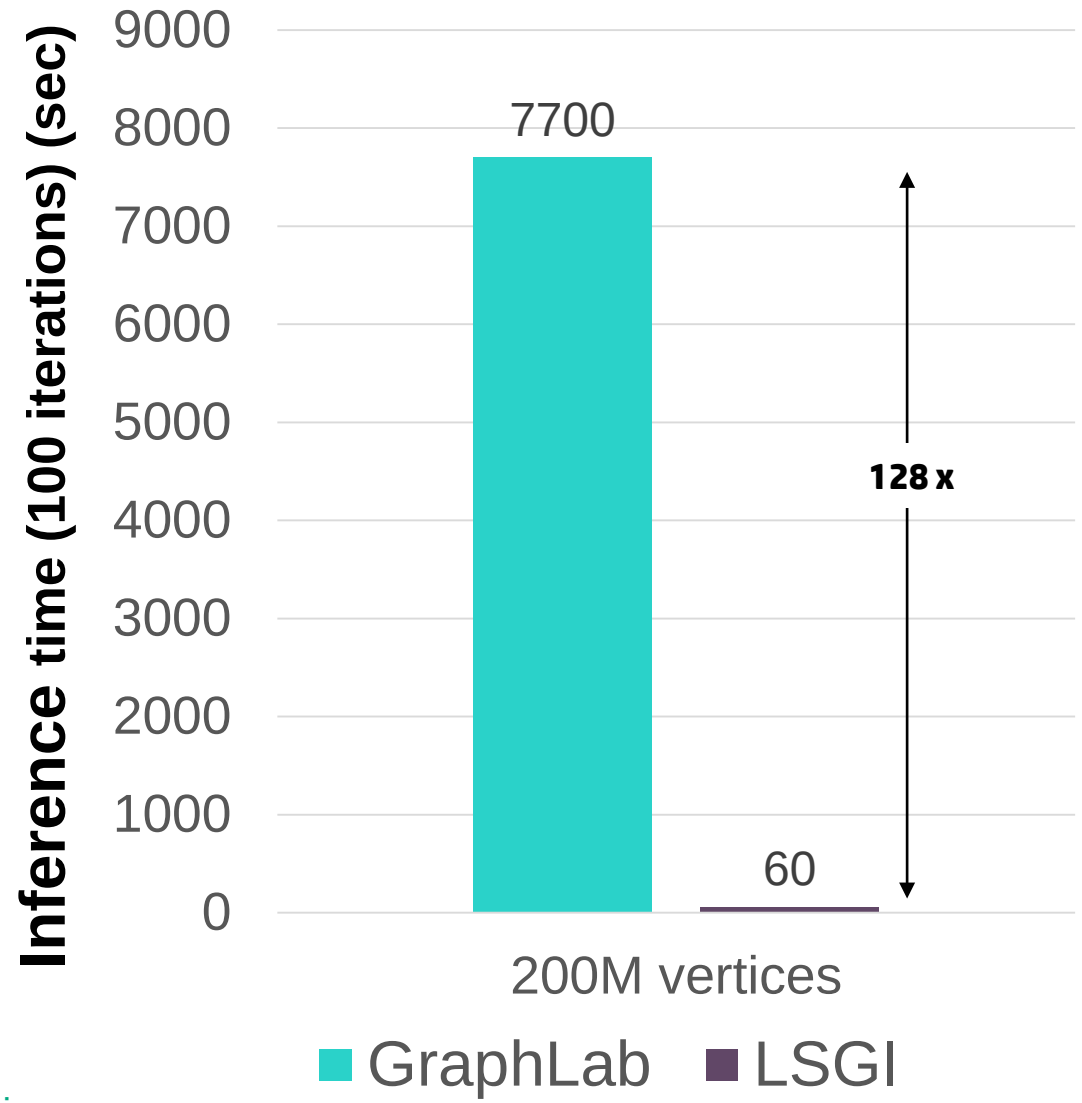
Repeat
Infer the unknown vertex state:
Malicious or not ?
Until Converge

Hewlett Packard Enterprise

©Copyright 2017 Hewlett Packard Enterprise Company

Large-scale graph inference

- Memory-Driven Computing approach
  - Maximize sequential memory operations
  - Lock-free vertex updates avoid lock overheads
  - Asynchronous coordination through fabric-attached memory minimizes synchronization overheads of vertex states
- Comparison on Superdome X
  - 240 cores, 12TB DRAM
  - Fabric-attached memory surrogate
- Memory-Driven Computing achieves orders of magnitude improvements



F. Chen, M. Gonzalez, et al., "Billion node graph inference: iterative processing on The Machine," Hewlett Packard Labs Technical Report HPE-2016-101, December 2016.

Open source coming soon at
https://www.labs.hpe.com/the-machine/the-machine-distribution

Hewlett Packard Enterprise

©Copyright 2017 Hewlett Packard Enterprise Company

Memory-Driven Monte Carlo (MC) simulations



Traditional

Step 1: Create a parametric model $y = f(x_1,\ldots,x_k)$
Step 2: Generate a set of random inputs
Step 3: Evaluate the model and store the results
Step 4: Repeat steps 2 and 3 many times
Step 5: Analyze the results

Memory-Driven

Replace steps 2 and 3 with look-ups, transformations

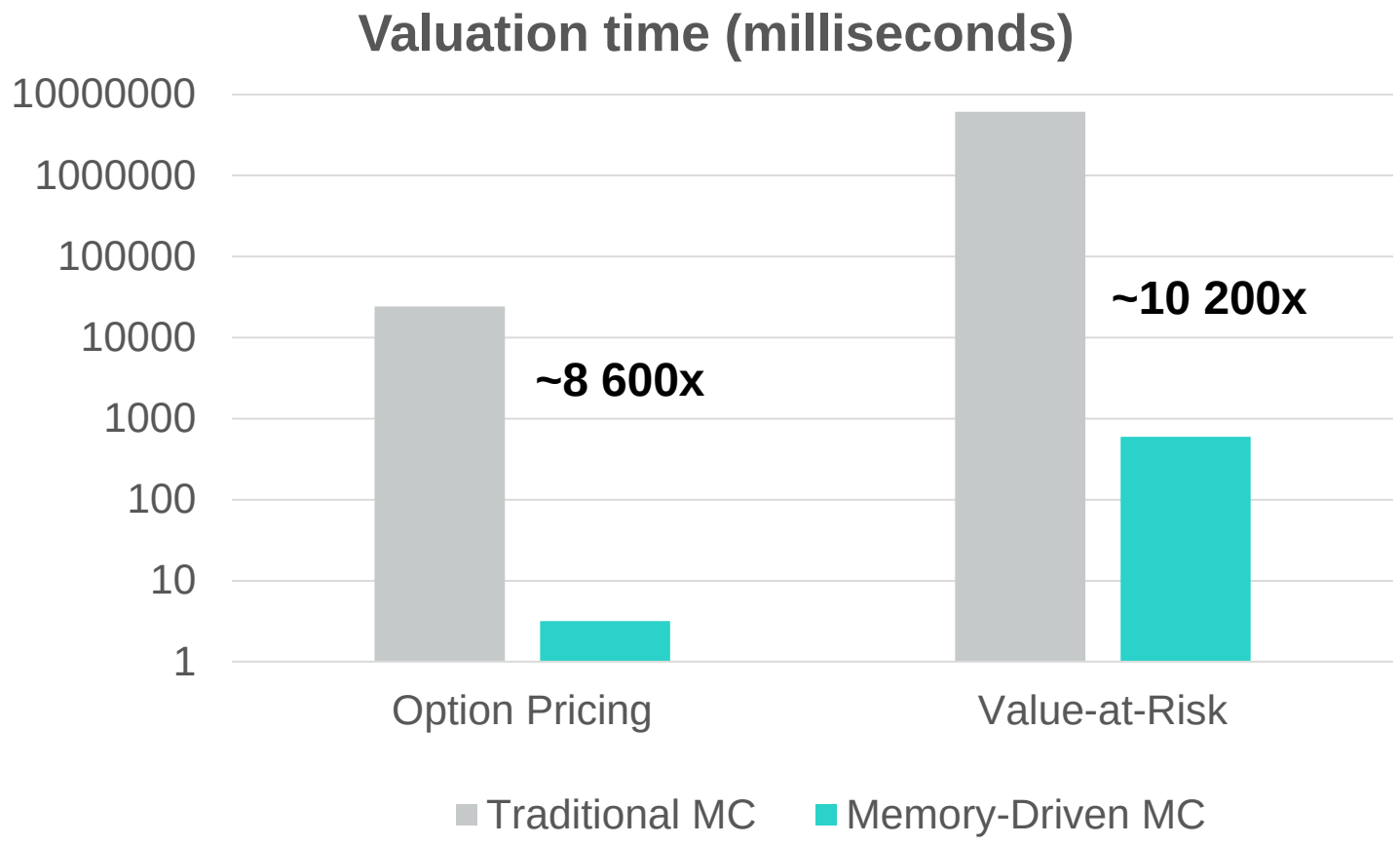
- Pre-compute representative simulations and store in memory
- Use transformations of stored simulations instead of computing new simulations from scratch

Hewlett Packard Enterprise

©Copyright 2017 Hewlett Packard Enterprise Company

Experimental comparison: Memory-Driven MC vs. traditional MC

Speed of option pricing and risk management



Option pricing

S&P 500 call option

Value-at-Risk

Portfolio of 10000 products with 500 correlated underlying assets

Time horizon: 14 days

Hewlett Packard Enterprise

©Copyright 2017 Hewlett Packard Enterprise Company

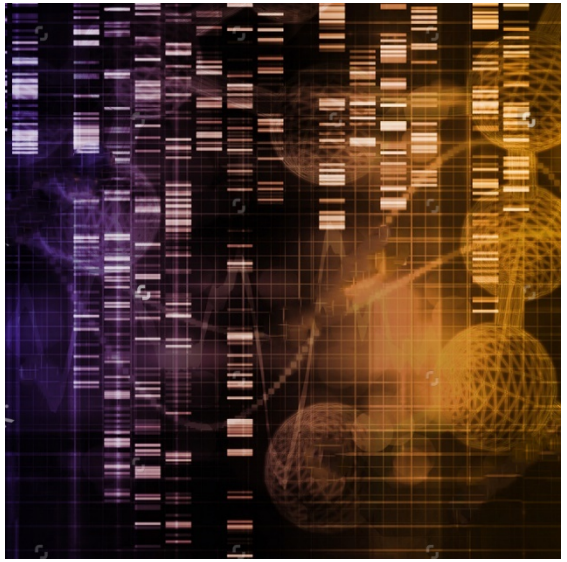
Performance possible with Memory-Driven programming

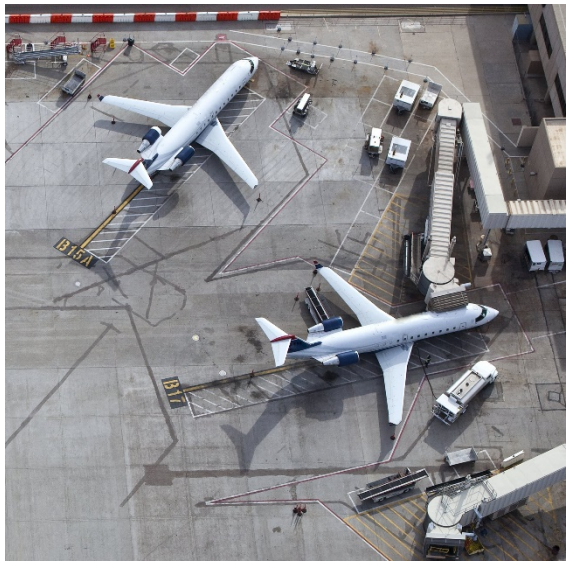
Modify existing frameworks

New algorithms

Completely rethink









In-memory analytics

15x
faster

Similarity search

20x
faster

Large-scale graph inference

100x
faster

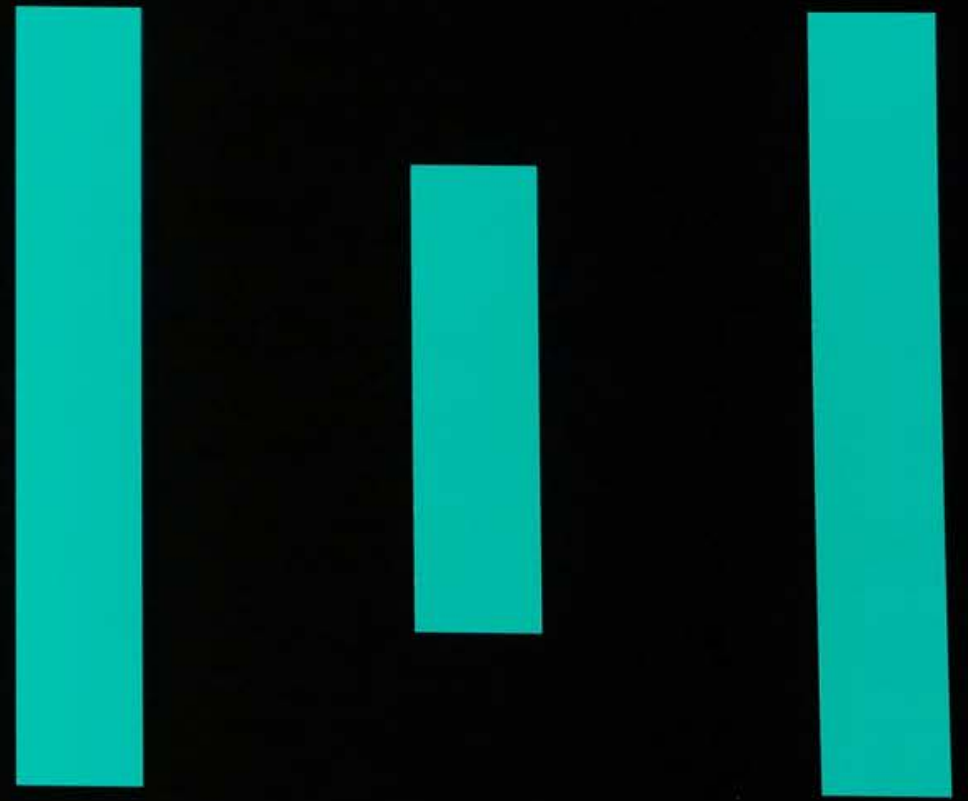
Financial models

8,000x
faster

Hewlett Packard Enterprise

©Copyright 2017 Hewlett Packard Enterprise Company

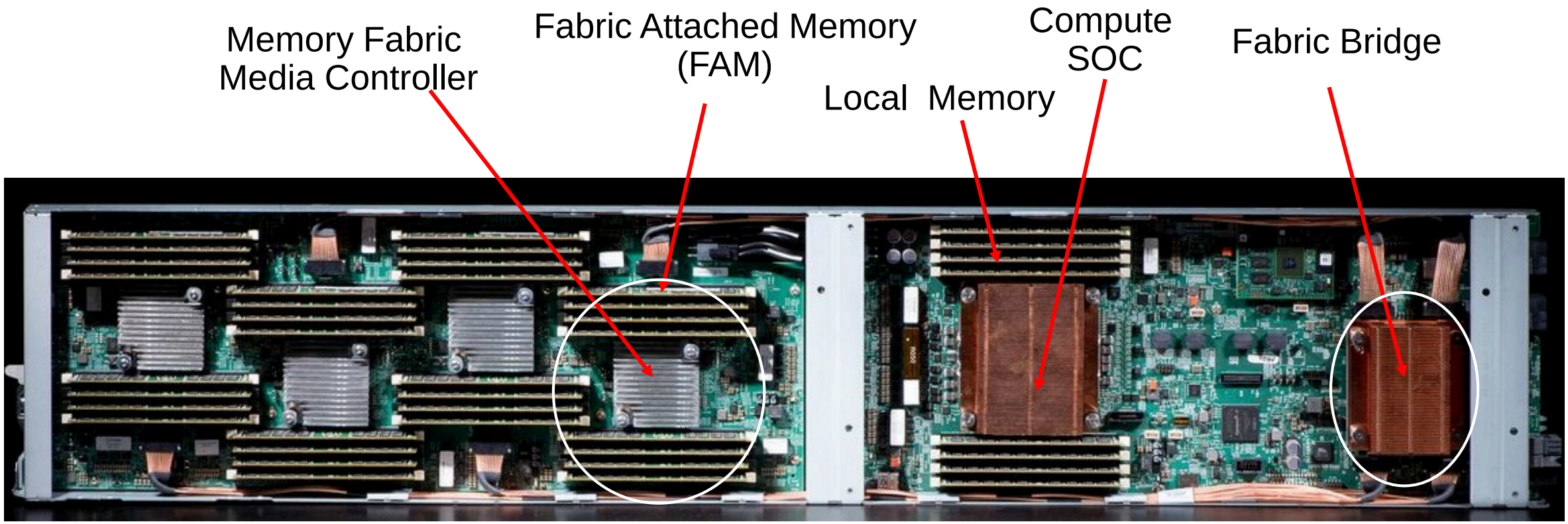
How do we get to Memory-Driven Computing?



the MACHINE

Hewlett Packard Enterprise

The Machine program: Memory Fabric Testbed

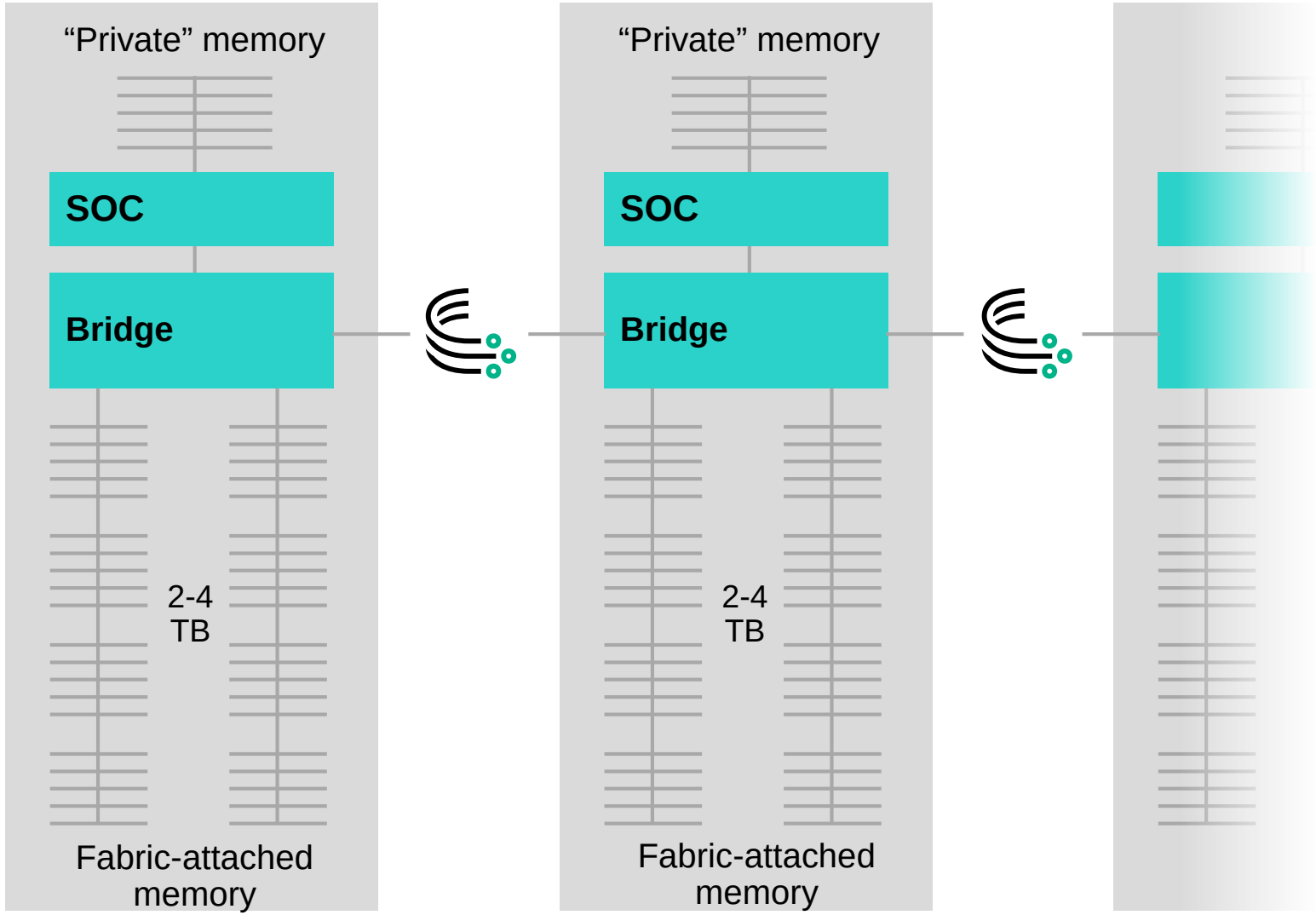


Hewlett Packard
Enterprise

©Copyright 2017 Hewlett Packard Enterprise Company

How fabric-attached memory works

Allows a compute node to access any part of the fabric-attached memory pool

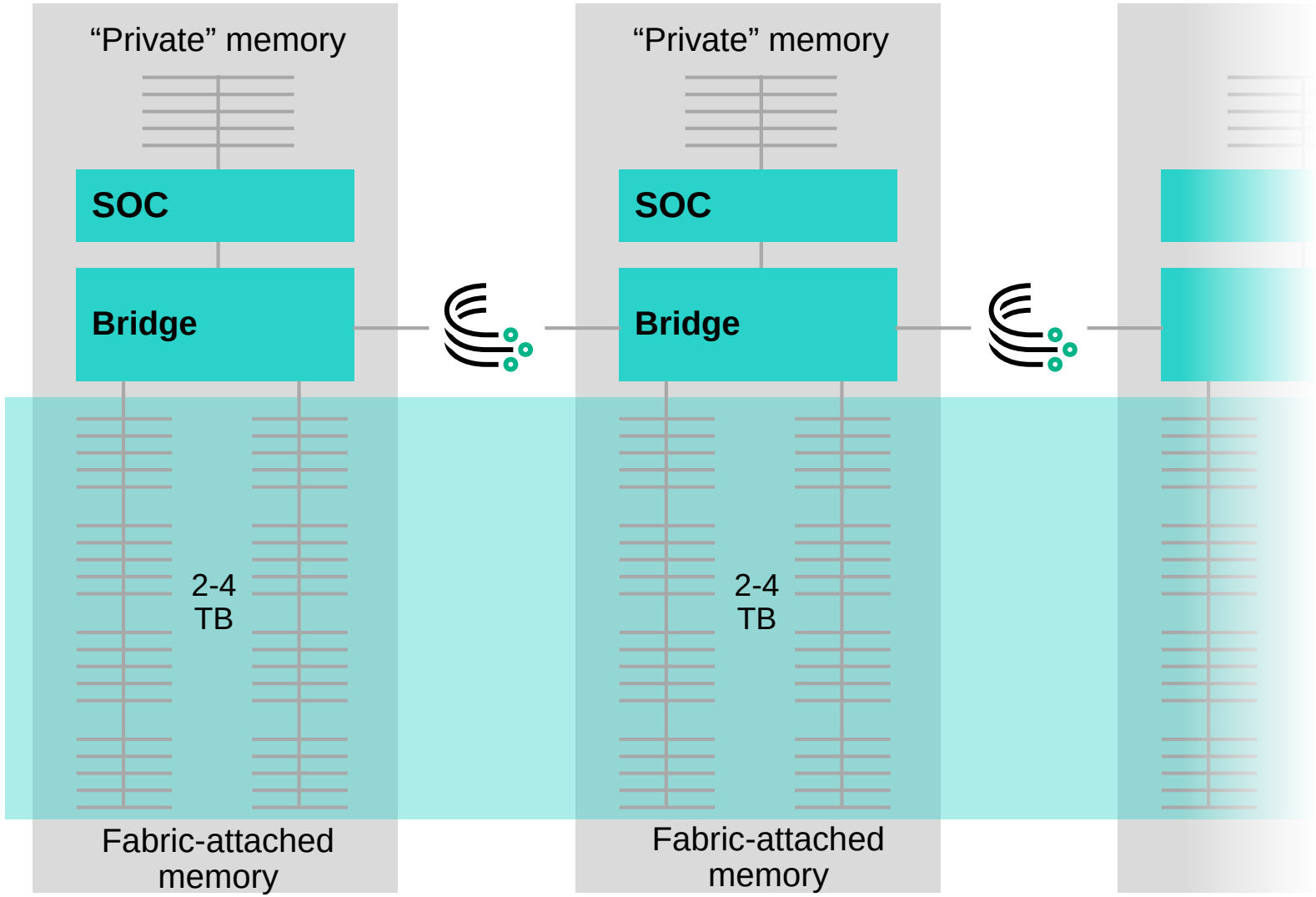


Hewlett Packard Enterprise

©Copyright 2017 Hewlett Packard Enterprise Company

How fabric-attached memory works

Allows a compute node to access any part of the fabric-attached memory pool

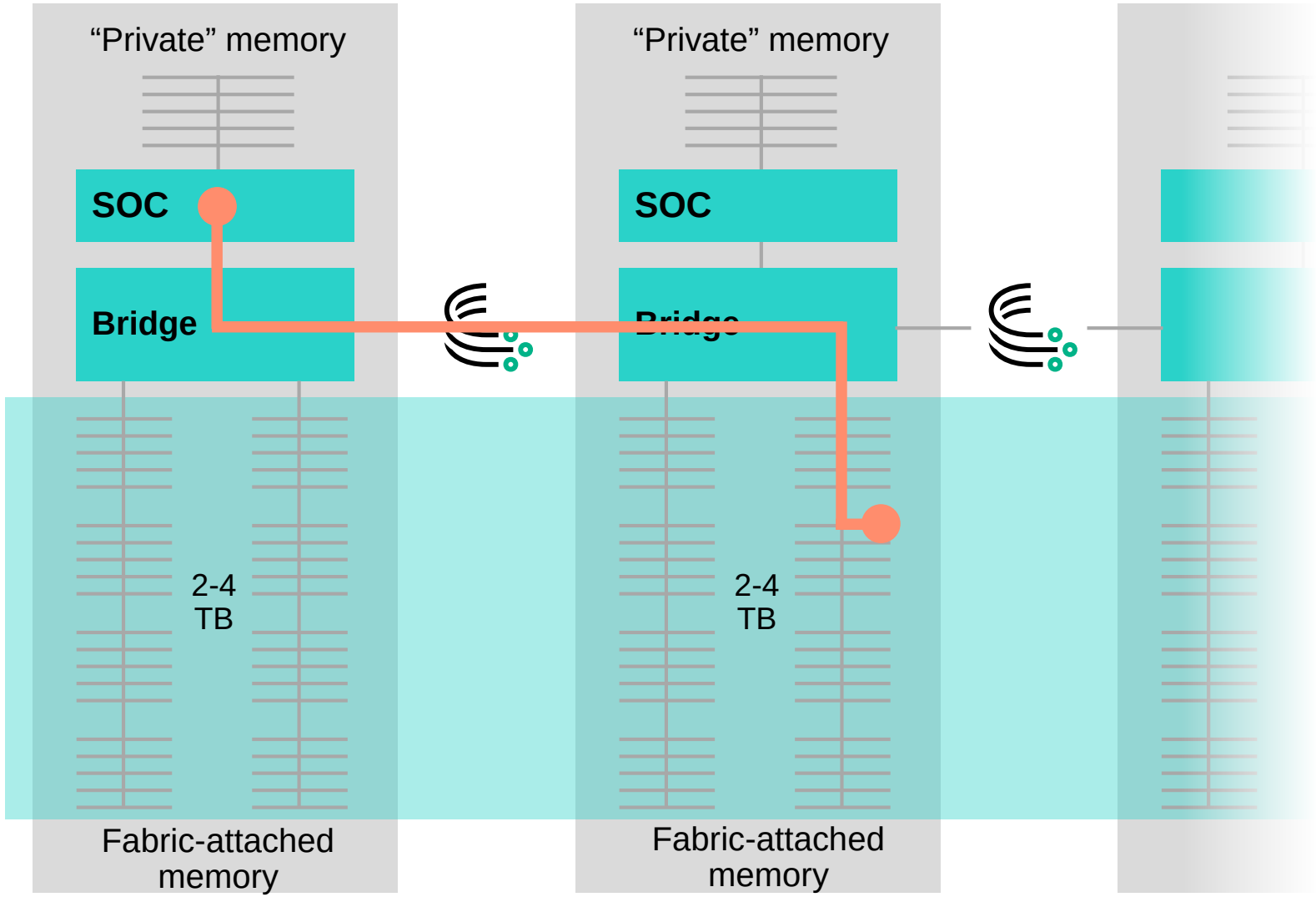


Hewlett Packard Enterprise

©Copyright 2017 Hewlett Packard Enterprise Company

How fabric-attached memory works

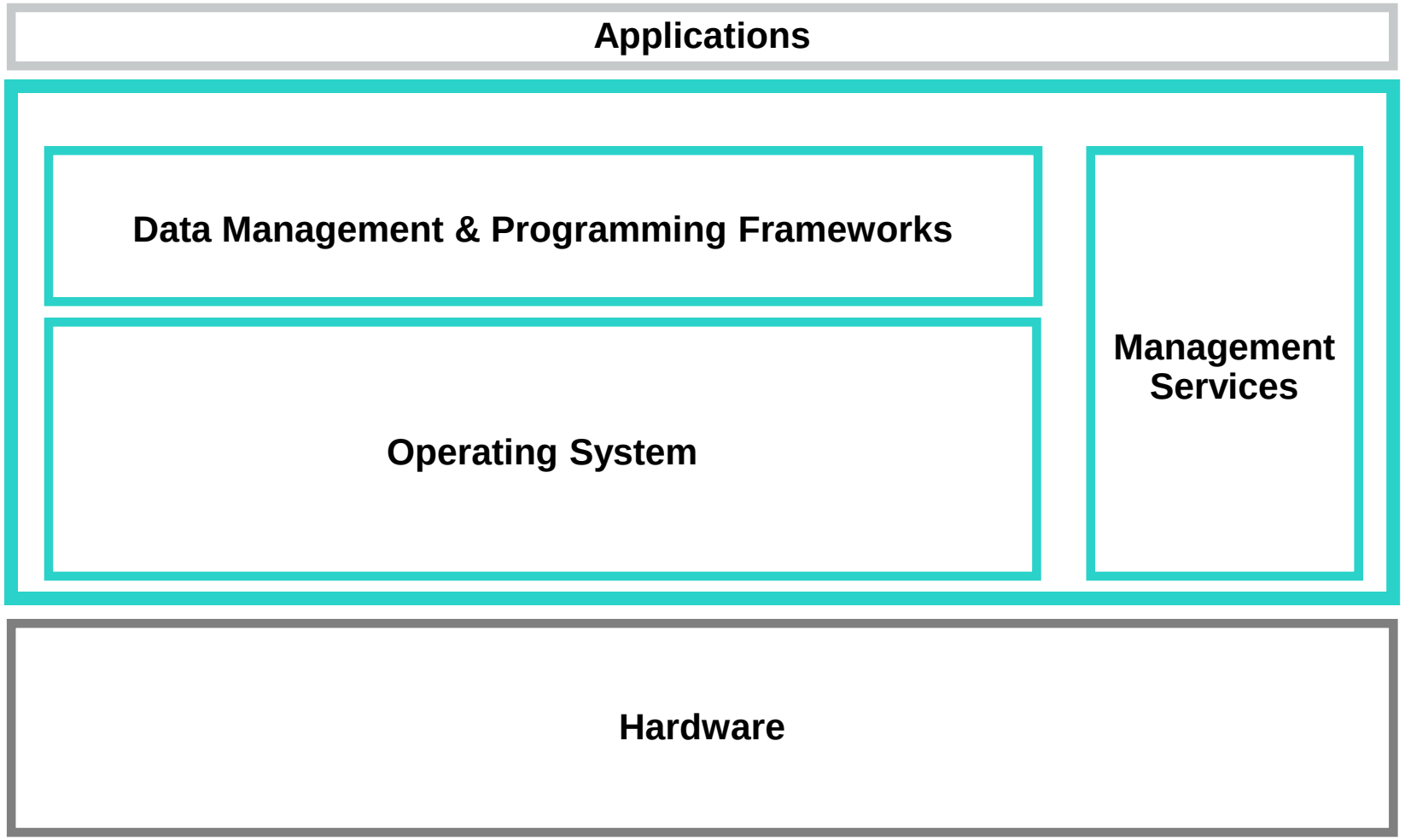
Allows a compute node to access any part of the fabric-attached memory pool



Hewlett Packard Enterprise

©Copyright 2017 Hewlett Packard Enterprise Company

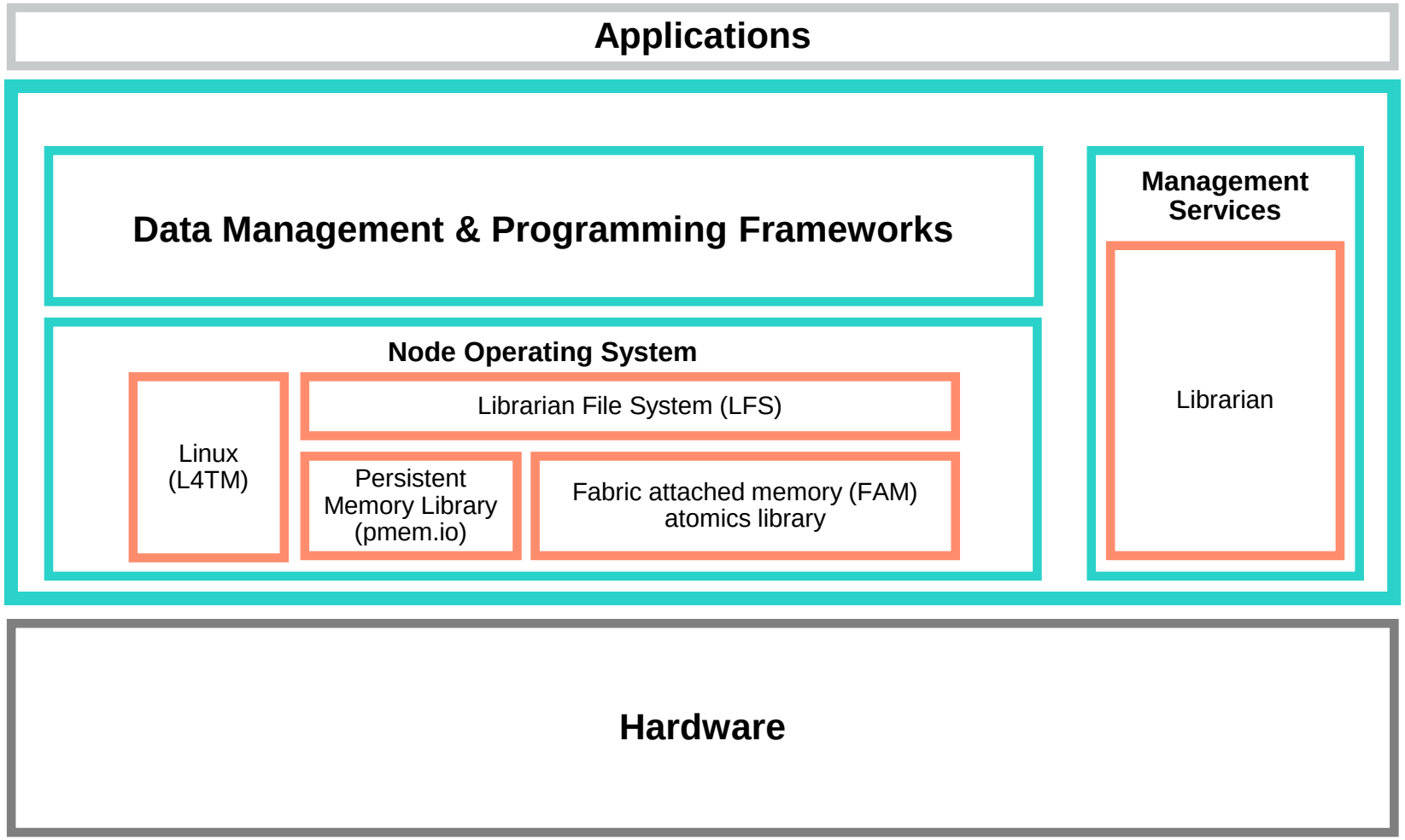
Opportunities to rethink the whole software stack



Hewlett Packard Enterprise

©Copyright 2017 Hewlett Packard Enterprise Company

Linux for The Machine



- L4TM: Linux modifications to support fabric-attached persistent memory
- FAM atomics primitives to handle sharing across nodes
- Pmem.io modifications to support non-coherent access
- LFS exposes fabric-attached memory as mmap'd shared FS
- Librarian for cross-node fabric memory allocation

Open sourced components

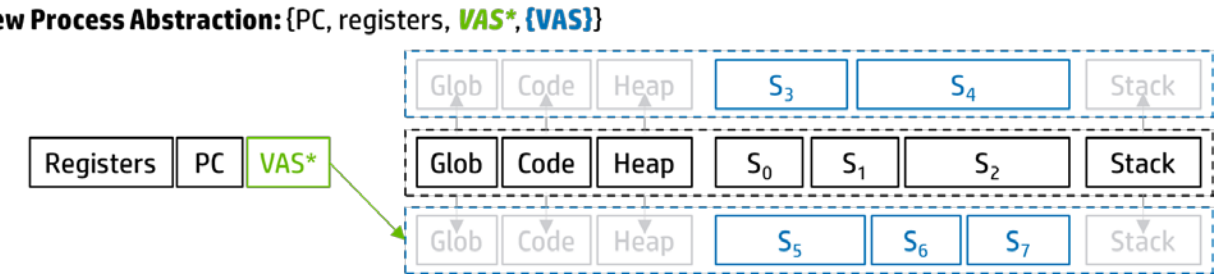
https://github.com/FabricAttachedMemory

Hewlett Packard Enterprise

©Copyright 2017 Hewlett Packard Enterprise Company

SpaceJMP: Programming with Multiple Virtual Address Spaces

- Virtual address space as first-class citizen
- Process can have multiple virtual address spaces



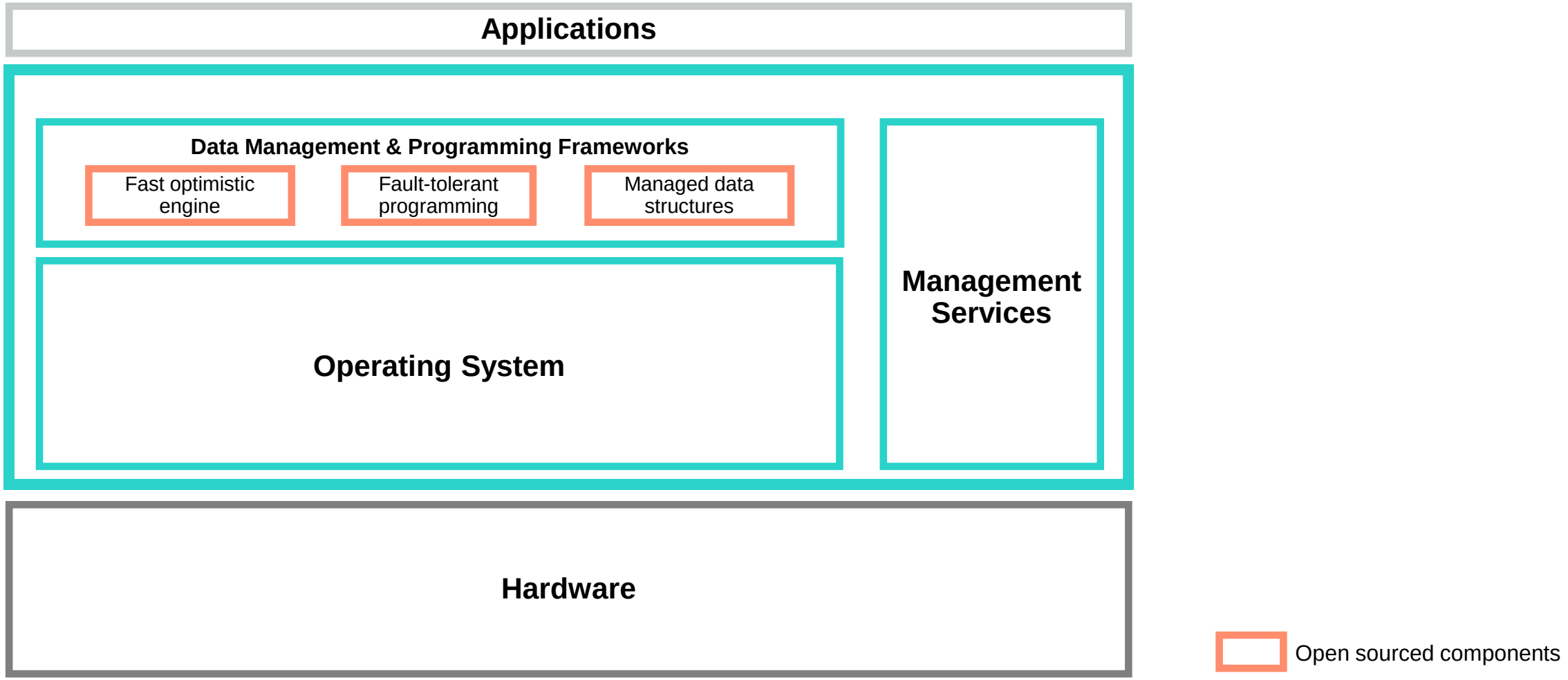
- Efficient safe programming and sharing for huge memories
- Data sharing and communication between processes
- Versioning and checkpointing
- Co-design between OS, programming languages, compilers, and runtimes
- Prototype implementations in BSD, Linux, and Barrelfish

I. El Hajj, et al. “SpaceJMP: Programming with Multiple Virtual Address Spaces,” *Proc. Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2016.

Hewlett Packard Enterprise

©Copyright 2017 Hewlett Packard Enterprise Company

Data management and programming frameworks



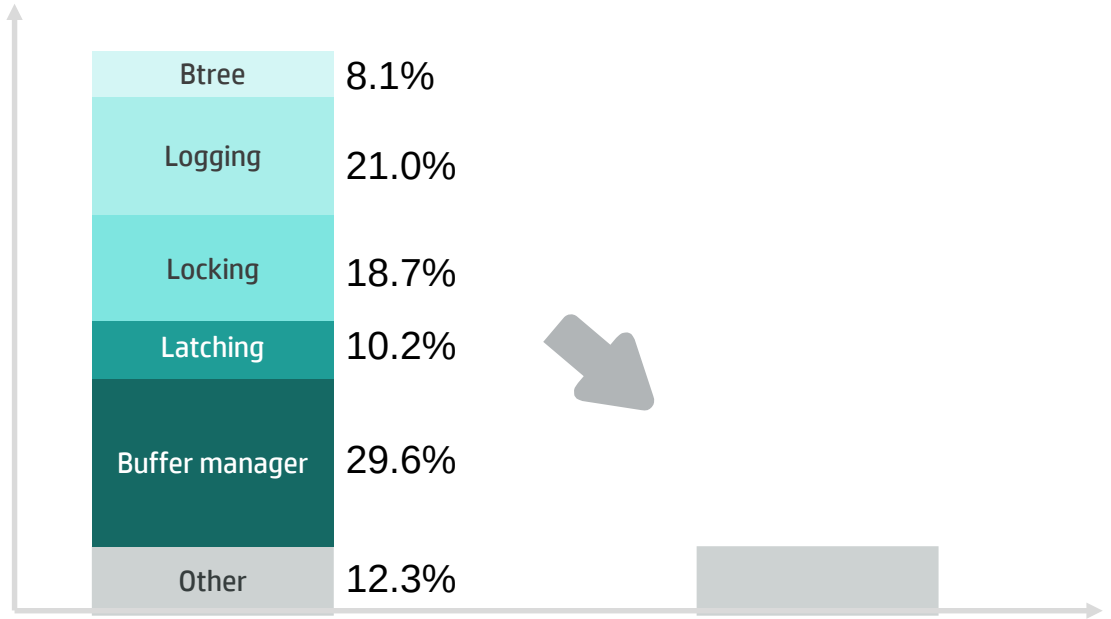
Hewlett Packard Enterprise

©Copyright 2017 Hewlett Packard Enterprise Company

Traditional databases

–Example: A database (write) transaction

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

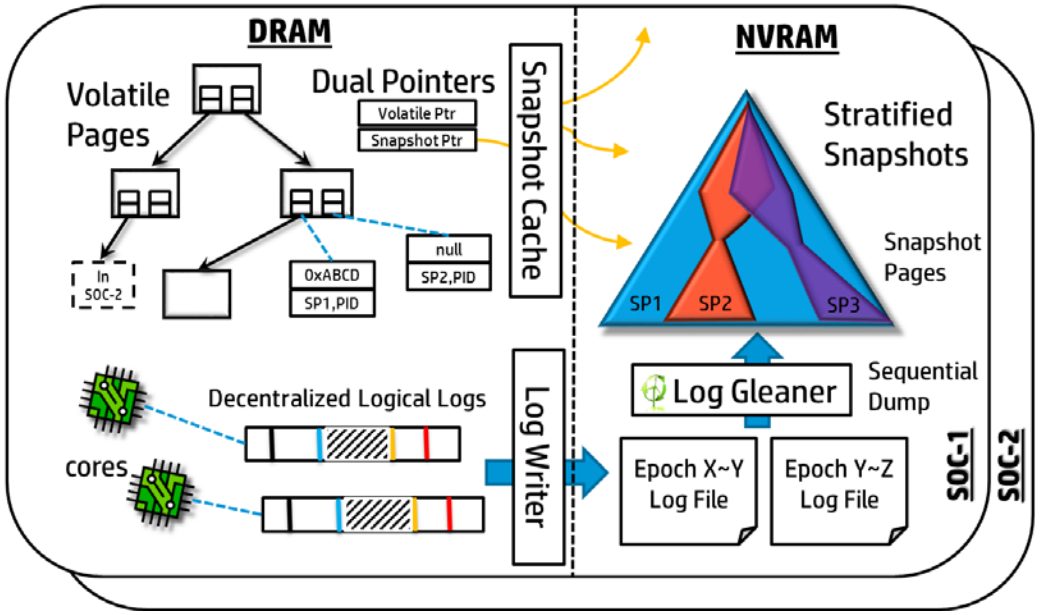


S. Harizopoulos, D. Abadi, S. Madden, and M. Stonebraker, “OLTP Through the Looking Glass, and What We Found There,” *Proc. SIGMOD*, 2008.

©Copyright 2017 Hewlett Packard Enterprise Company

Fast optimistic engine for data unification services

- Open-source, from-scratch database engine designed to
  - Take advantage of large multi-core machines
  - Manipulate data both in DRAM and NVM
- Fully ACID, serializable database kernel
  - Can be embedded in applications as a library
  - Simplified in-memory applications
- Designed to eliminate scalability bottlenecks
  - Lightweight optimistic concurrency control
  - Decentralized logs are SoC-friendly
  - Design maximizes NVM bandwidth and endurance



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

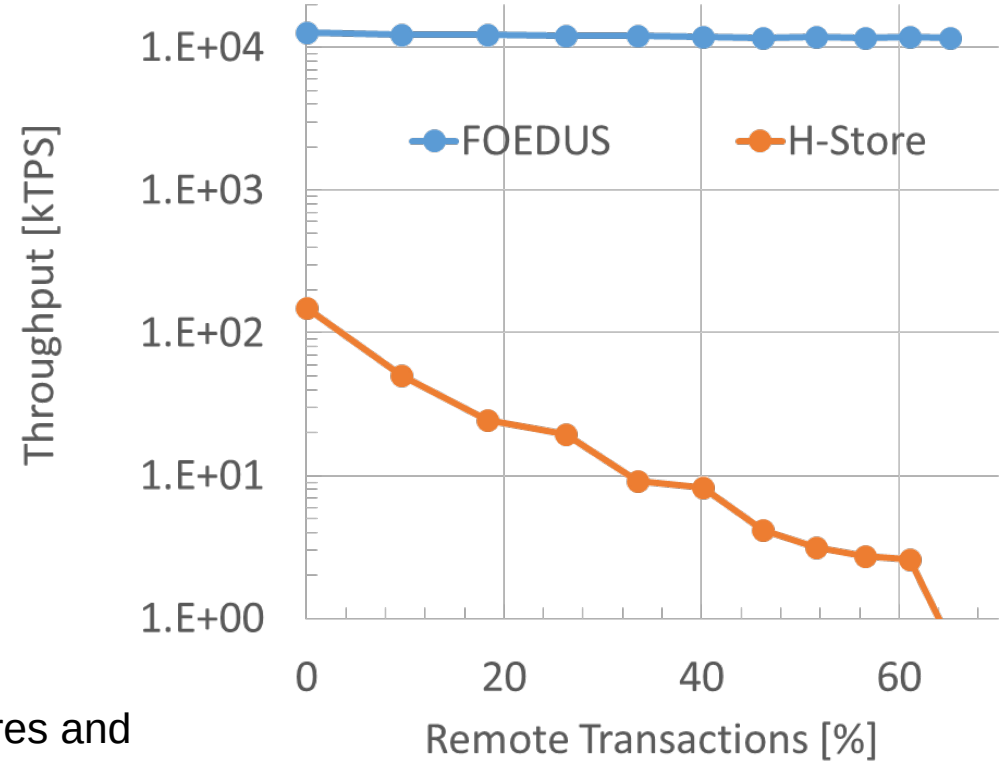
Hewlett Packard Enterprise

©Copyright 2017 Hewlett Packard Enterprise Company

Fast optimistic engine performance

– Scalable up to tens of SoCs
  – Tested scale: Superdome X: 12 TB DRAM, 240 cores
– Efficiently handles datasets larger than DRAM
– Orders of magnitude faster when compared to state-of-the-art in-memory engines

– Open source code, documentation and papers at https://github.com/HewlettPackard/foedus



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

H. Kimura, A. Simitsis, K. Wilkinson, “Janus: Transactional processing of navigational and analytical graph queries on many-core servers,” *Proc. CIDR*, 2017.

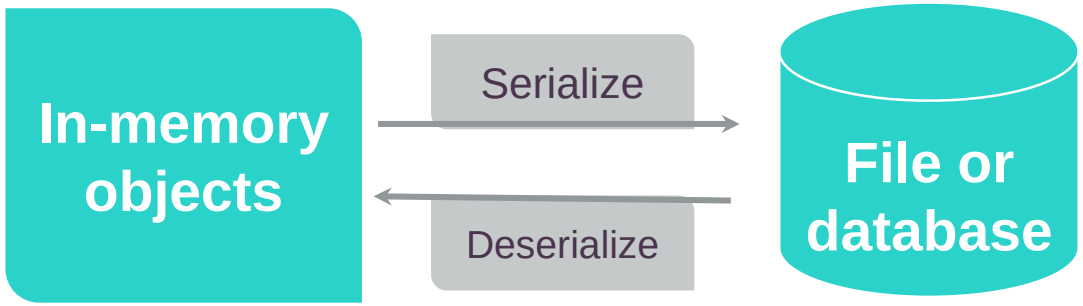
Hewlett Packard Enterprise

©Copyright 2017 Hewlett Packard Enterprise Company

Do we need separate data representations?

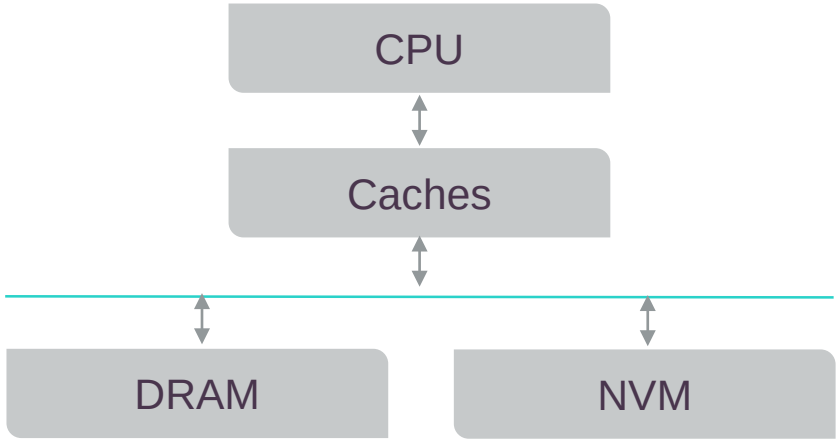
In-storage durability

+ 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 ld/st latencies offer high performance
  – Persistent does not mean consistent!



©Copyright 2017 Hewlett Packard Enterprise Company

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];
```

I want to transfer money between accounts. 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?

Processor caches are still volatile

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

Hewlett Packard Enterprise

©Copyright 2017 Hewlett Packard Enterprise Company

Manual solution

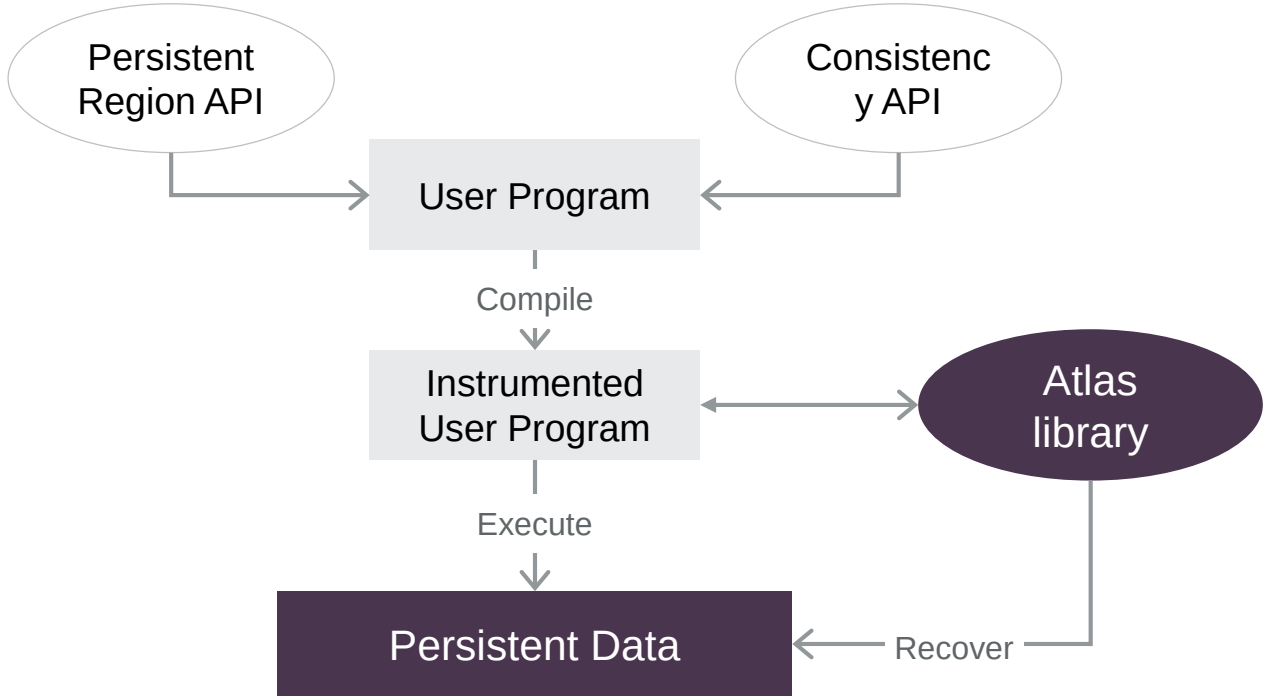
```
persistent double accounts[2];
transfer(int from, int to, double amount) {
<save old value of accounts[from] in undo log>;
<flush log entry to NVM>
                              accounts[from] -= amount;
<save old value of accounts[to] in undo log>;
<flush log entry to NVM>
                              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**

©Copyright 2017 Hewlett Packard Enterprise Company

Fault-tolerant programming model for non-volatile memory
Atlas

- Programmer distinguishes persistent and transient data
- Persistent data lives in a “persistent region”
  - E.g., in pseudo-file-system in NVM
  - Directly mapped 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
- Open source code available at https://github.com/HewlettPackard/Atlas



D. Chakrabarti, H. Boehm and K. Bhandari. “Atlas: Leveraging Locks for Non-volatile Memory Consistency,” *Proc. Object-Oriented Programming, Systems, Languages & Applications* (*OOPSLA)*, 2014.

©Copyright 2017 Hewlett Packard Enterprise Company

Fault-tolerant programming model for non-volatile memory

NVthreads

– Approach: multi-process execution + redo logs + intercept lock operations

– Drop-in replacement for the *pthreads* library

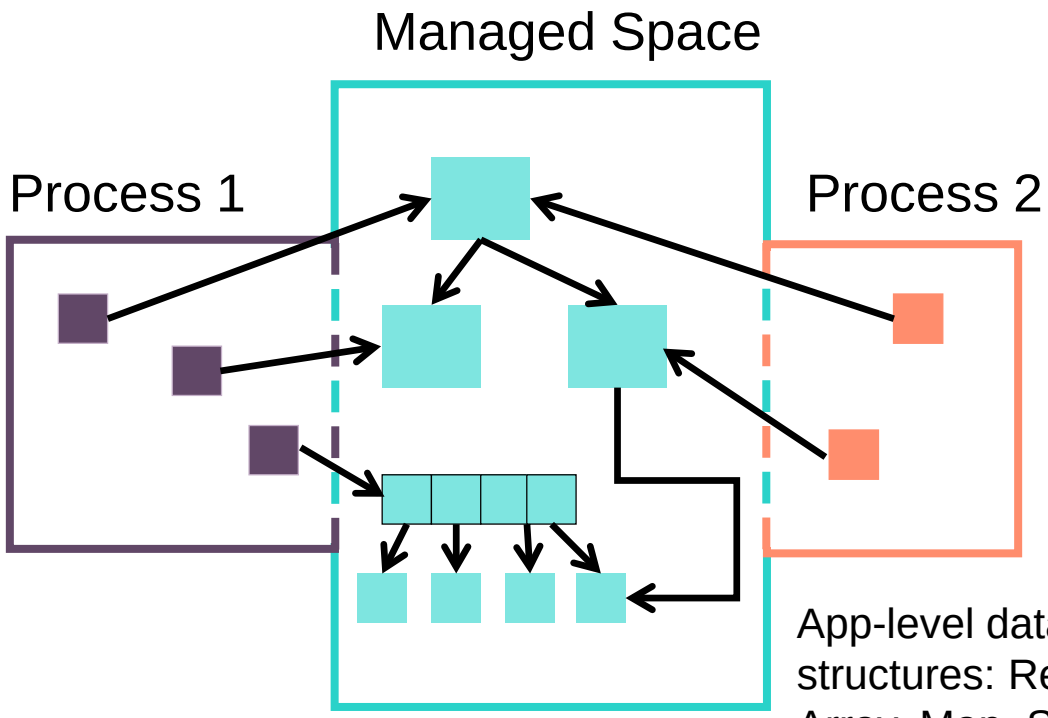
Issue	Common solutions	NVthreads approach	NVthreads advantage
Determine consistency points	Use durable transactions	Infer from locks	Fewer app. modifications
Ensure failure atomicity	Undo logs, redo logs, or word level STM	Process memory, and redo logs	Lower overheads, simpler recovery
Track updates	Intercept each store	Intercept page writes	Good performance
Volatile caches	Flush each store	Flush memory pages	Amortized costs

T. Hsu, H. Brugner, I. Roy, K. Keeton, P. Eugster, “NVthreads: Practical Persistence for Multi-threaded Applications,” To appear in *Proc. EuroSys*, 2017.
Open source code at https://github.com/HewlettPackard/nvthreads

Managed Data Structures (MDS)

Simplify programming on persistent in-memory data

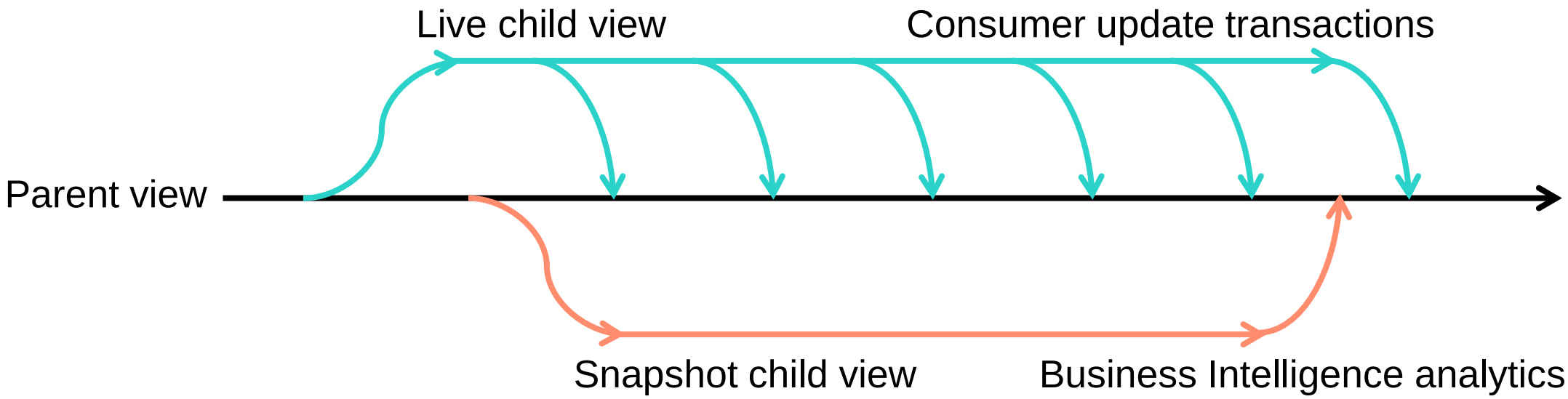
- Ease of Programming
  - Programmer manages only application-level data structures
    - MDS data structures are automatically persisted in NVM
  - APIs in multiple programming languages: Java, C++
    - Programmer access through references to data
    - Direct reads and writes
- Ease of Data Sharing
  - Just pass a reference
    - Each program treats the data as if it was local to the program
  - High-level concurrency controls
    - Ensure consistent data in the face of data sharing by multiple threads/processes



Hewlett Packard Enterprise

©Copyright 2017 Hewlett Packard Enterprise Company

Isolation contexts support safe data sharing



Non-blocking transactions

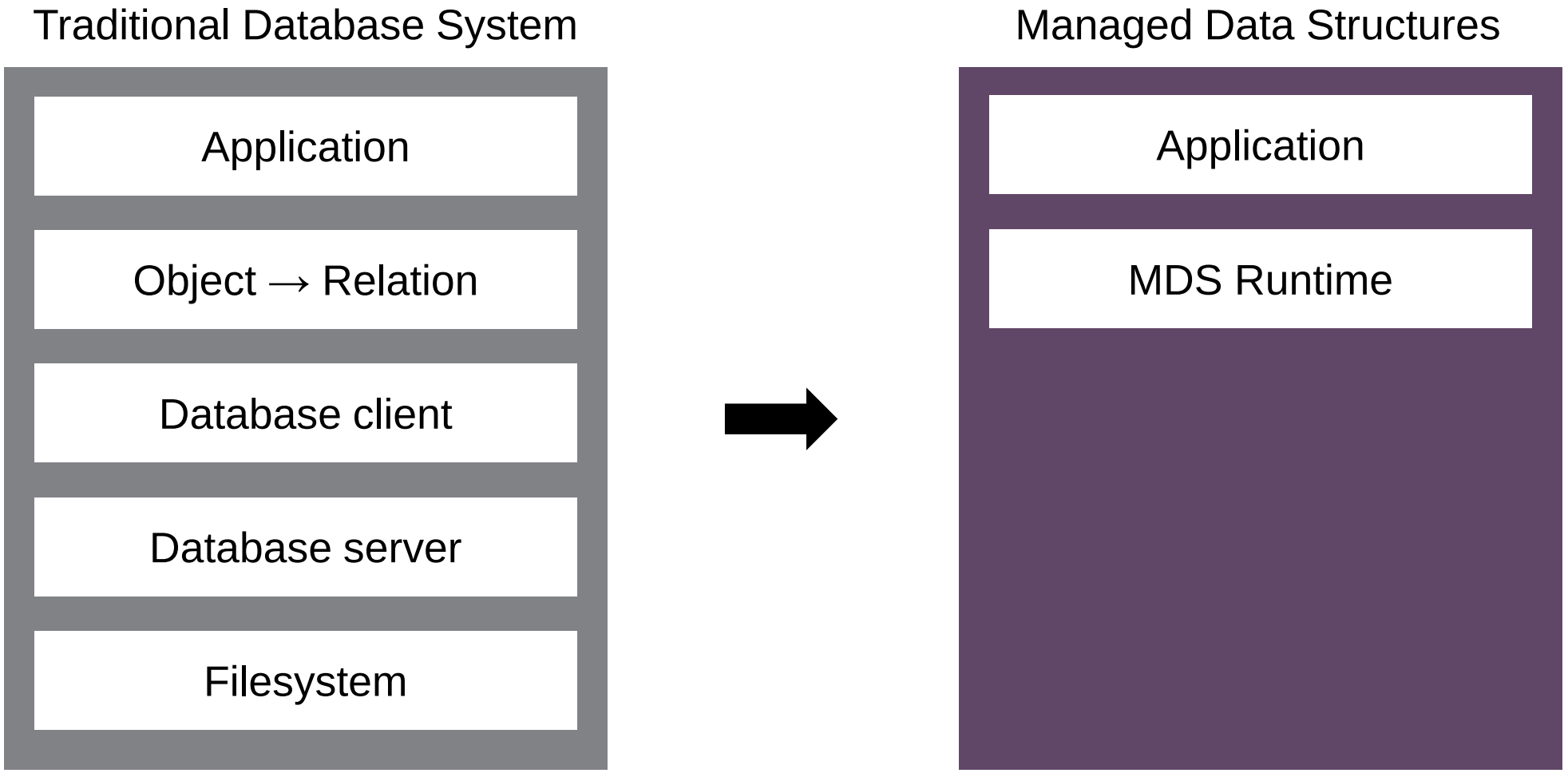
Zero-copy snapshots

Conflict resolution

Hewlett Packard Enterprise

©Copyright 2017 Hewlett Packard Enterprise Company

Fewer software layers



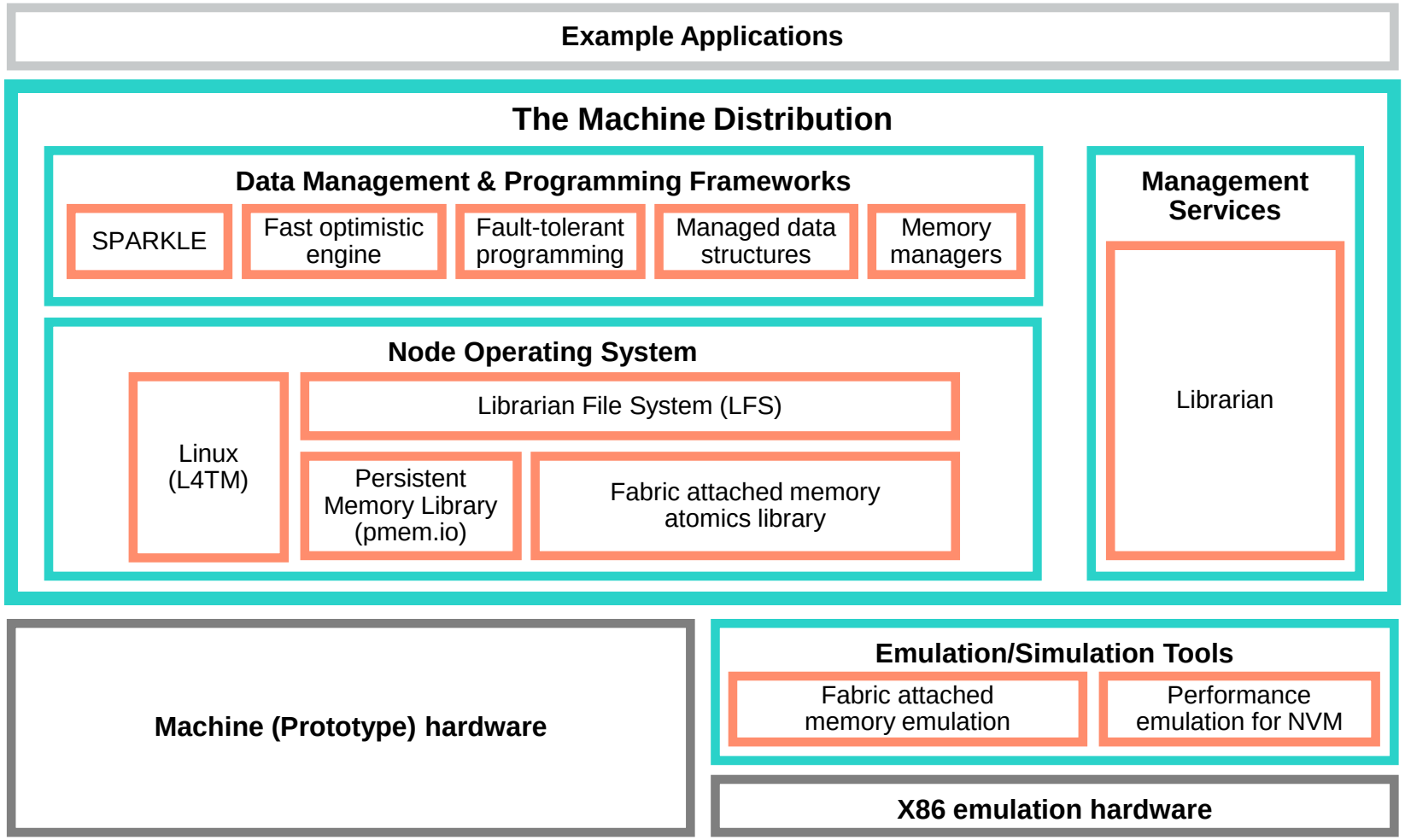
Open source code at https://github.com/HewlettPackard/mds

Hewlett Packard Enterprise

©Copyright 2017 Hewlett Packard Enterprise Company

The Machine Distribution

Software stack for Memory-Driven Computing



Programming and analytics tools

Operating system support

Emulation/simulation tools

Open sourced components

https://github.com/HewlettPackard/mdc-toolkit

https://www.labs.hpe.com/the-machine/the-machine-distribution

Hewlett Packard Enterprise

©Copyright 2017 Hewlett Packard Enterprise Company

Memory-Driven Computing challenges for the FAST community

Hewlett Packard Enterprise

Storage-related Memory-Driven Computing challenges

– Traditional storage services

– Traditional data access

– Heterogeneity and tiering

– Coping with failures

– Programming models

– Characterization opportunities

©Copyright 2017 Hewlett Packard Enterprise Company

Traditional storage services

- Issue: keeping persistently stored data safe in fabric-attached NVM
  - NVM failures may result in loss of persistent data
  - Persistently stored data is subject to theft
- Revisit traditional storage services: redundancy, encryption, compression, deduplication
- New challenges:
  - Need to operate at memory speeds, not storage speeds
  - Space-efficient redundancy for NVM
  - Traditional solutions (e.g., encryption, compression) complicate direct access
  - Providing performant access to encrypted data
- Memory-side hardware acceleration
  - Memory speeds may demand acceleration (e.g., DMA-style data movement, memset, encryption, compression)
  - What memory-side acceleration functions strike good balance between application performance and generality?

©Copyright 2017 Hewlett Packard Enterprise Company

Traditional data access

- Performant file systems for fabric-attached NVM
  - Applications favor familiar file API, but traditional implementations are inefficient and don't handle disaggregation
  - How to extend NVM file system work to support Memory-Driven Computing environment?

- Memory access control
  - As access granularity gets smaller, access control enforcement needs to adjust
  - What approaches (e.g., capabilities) provide fine-grained access control and protection for Memory-Driven Computing environment?

©Copyright 2017 Hewlett Packard Enterprise Company

Heterogeneity and tiering

- Managing multi-tiered memory/storage systems
  - Load/store memory isn't most cost-effective medium for cold data
  - How to manage multi-tiered hierarchy, to ensure data is in "right" tier for performant access, long-term storage, etc.?
  - What opportunities does Memory-Driven Computing provide for multi-site storage?

- Heterogeneous memory
  - DRAM stores ephemeral data, NVM stores persistent data: different access patterns
  - How to manage allocation, reclamation, naming, etc. for different memory technologies?

- Leveraging local memory
  - Memory-Driven Computing includes both local and fabric-attached memory
  - How should local memory be managed – system-managed inclusive cache? scratchpad? application-managed?

©Copyright 2017 Hewlett Packard Enterprise Company

Coping with failures

– Ensuring crash consistency in face of failures
  – Crash consistency methods can degrade application performance
  – What are strategies for performant persistence and crash consistency?

– New memory fabric error models
  – Memory fabric errors may lead to load/store failures, which may be visible only after the originating instruction
  – How to provide reasonable reporting and selective handling of memory errors?

©Copyright 2017 Hewlett Packard Enterprise Company

Programming models

– System organization: remote memory vs. disaggregated memory
  – Memory disaggregation provides many benefits, but departs from popular scale-out programming model
  – Should non-local memory be treated as disaggregated (unmediated) resource or remote (mediated) resource?

– Memory-driven coordination
  – Shared persistent memory can provide global view of shared state that enables coordination
  – What are “memory-driven” approaches for concurrency control, consensus, failure detection, etc.?

©Copyright 2017 Hewlett Packard Enterprise Company

Characterization opportunities

- Characterization of new NVM and interconnect technologies
  - New technologies present new performance, endurance, and management opportunities and challenges
  - How do these new technologies behave under expected typical usage patterns?

- Implications of “wider” API for persistent data
  - “Narrow” storage APIs require explicit action for persistence, but “wider” memory APIs allow stores to persist data
  - Does widening of API for persisting data lead to more errors or more data corruption?

©Copyright 2017 Hewlett Packard Enterprise Company

For more information...https://www.labs.hpe.com/the-machine/

A sampling of our papers on Memory-Driven Computing:

- T. Hsu, H. Brugner, I. Roy, K. Keeton, P. Eugster. “NVthreads: Practical Persistence for Multi-threaded Applications,” To appear in *Proc. ACM EuroSys*, 2017.
- S. Nalli, S. Haria, M. Swift, M. Hill, H. Volos, K. Keeton. ”An Analysis of Persistent Memory Use with WHISPER,” *Proc. ACM Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2017.
- H. Kimura, A. Simitsis, K. Wilkinson, “Janus: Transactional processing of navigational and analytical graph queries on many-core servers,” *Proc. CIDR*, 2017.
- F. Chen, M. Gonzalez, K. Viswanathan, H. Laffitte, J. Rivera, A. Mitchell, S. Singhal. “Billion node graph inference: iterative processing on The Machine,” Hewlett Packard Labs Technical Report HPE-2016-101, December 2016.
- P. Laplante and D. Milojicic. "Rethinking operating systems for rebooted computing," *Proc. IEEE International Conference on Rebooting Computing (ICRC)*, 2016.
- D. Chakrabarti, H. Volos, I. Roy, and M. Swift. “How Should We Program Non-volatile Memory?”, tutorial at *ACM Conf. on Programming Language Design and Implementation (PLDI)*, 2016.
- K. Viswanathan, M. Kim, J. Li, M. Gonzalez. “A memory-driven computing approach to high-dimensional similarity search,” Hewlett Packard Labs Technical Report HPE-2016-45, May 2016.
- N. Farooqui, I. Roy, Y. Chen, V. Talwar, and K. Schwan. “Accelerating Graph Applications on Integrated GPU Platforms via Instrumentation-Driven Optimization,” *Proc. ACM Conf. on Computing Frontiers (CF’16)*, May 2016.
- I. El Hajj, A. Merritt, G. Zellweger, D. Milojicic, W. Hwu, K. Schwan, T. Roscoe, R. Achermann, P. Faraboschi. “SpaceJMP: Programming with multiple virtual address spaces,” *ASPLOS*, 2016.
- J. Izraelevitz, T. Kelly, A. Kolli. “Failure-atomic persistent memory updates via JUSTDO logging,” *Proc. ACM ASPLOS*, 2016.
- D. Milojicic, T. Roscoe. “Outlook on Operating Systems,” *IEEE Computer*, January 2016.
- K. Bresniker, S. Singhal, and S. Williams. “Adapting to thrive in a new economy of memory abundance,” *IEEE Computer*, December 2015.
- H. Volos, G, Magalhaes, L, Cherkasova, J, Li. “Quartz: A lightweight performance emulator for persistent memory software,” *Proc. of ACM/USENIX/IFIP Conference on Middleware*, 2015.
- J. Li, C. Pu, Y. Chen, V. Talwar, and D. Milojicic. “Improving Preemptive Scheduling with Application-Transparent Checkpointing in Shared Clusters,” *Proc. Middleware*, 2015.
- H. Kimura. “FOEDUS: OLTP engine for a thousand cores and NVRAM,” *Proc. ACM SIGMOD*, 2015.
- P. Faraboschi, K. Keeton, T. Marsland, D. Milojicic. “Beyond processor-centric operating systems,” *Proc. USENIX Workshop on Hot Topics in Operating Systems (HotOS)*, 2015.
- S. Gerber, G. Zellweger, R. Achermann, K. Kourtis, and T. Roscoe, D. Milojicic. “Not your parents’ physical address space,” *Proc. HotOS,* 2015.
- F. Nawab, D. Chakrabarti, T. Kelly, C. Morrey III. “Procrastination beats prevention: Timely sufficient persistence for efficient crash resilience,” *Proc. Conf. on Extending Database Technology (EDBT)*, 2015.
- S. Novakovic, K. Keeton, P. Faraboschi, R. Schreiber, E. Bugnion. “Using shared non-volatile memory in scale-out software,” *Proc. ACM Workshop on Rack-scale Computing (WRSC)*, 2015.
- M. Swift and H. Volos. “Programming and usage models for non-volatile memory,” Tutorial at *ACM ASPLOS*, 2015.
- D. Chakrabarti, H. Boehm and K. Bhandari. “Atlas: Leveraging locks for non-volatile memory consistency,” *Proc. ACM Conf. on Object-Oriented Programming, Systems, Languages & Applications (OOPSLA)*, 2014.
- H. Volos, S. Nalli, S. Panneerselvam, V. Varadarajan, P. Saxena, M. Swift. "Aerie: Flexible file-system interfaces to storage-class memory," *Proc. EuroSys*, 2014.



Hewlett Packard
Enterprise

©Copyright 2017 Hewlett Packard Enterprise Company

For open source code…
https://www.labs.hpe.com/the-machine/the-machine-distribution

- **Spark for The Machine:** https://github.com/HewlettPackard/sparkle
- **Fast optimistic engine for data unification services:** https://github.com/HewlettPackard/foedus
- **Fault-tolerant programming model for non-volatile memory:**
  - Atlas: https://github.com/HewlettPackard/Atlas
  - NVthreads: https://github.com/HewlettPackard/nvthreads
- **Managed Data Structures:** https://github.com/HewlettPackard/mds
- **Memory-Driven Computing toolkit:** https://github.com/HewlettPackard/mdc-toolkit
- **Linux for The Machine:** https://github.com/FabricAttachedMemory
- **Fabric Attached Memory Emulation:** https://github.com/FabricAttachedMemory/Emulation
- **Performance emulation for NVM latency and bandwidth:** https://github.com/HewlettPackard/Quartz

©Copyright 2017 Hewlett Packard Enterprise Company

Acknowledgments

Ablimit Aji
Alvin AuYoung
Cullen Bash
Susan Benzel
Suparna Bhattacharya
Hans Boehm
Kumud Bhandari
Kirk Bresniker
Helge Brugner
John Byrne
Dhruva Chakrabarti
Fei Chen
Yuan Chen
Al Davis
Izzat El Hajj
Katy Evertson
Paolo Faraboschi
Dan Feldman
Martin Fink
Rich Friedrich
Lokesh Gidra
Daniel Gmach
Maria Teresa Gonzalez
Goetz Graefe
Wey Guy
Terry Hsu
Terence Kelly
Mijung Kim
Hideaki Kimura
Evan Kirshenbaum
Mike Krause
Harumi Kuno
Hernan Laffitte
Christina Lee
Richard Lewington
Jun Li
Mark Lillibridge
Alexander Merritt
Dejan Milojicic
April Mitchell
Brad Morrey
Siamak Nazari
Lisa Pallotti
Kivanc Ozonat
Janneth Rivera
Indrajit Roy
Mehmet Sayal
Rob Schreiber
Sergey Serebryakov
Amit Sharma
Alkis Simitsis
Sharad Singhal
John Sontag
Susan Spence
Ram Swaminathan
Mike Tan
Joe Tucek
Natalia Vassilieva
Krishna Viswanathan
Doug Voigt
Haris Volos
Andrew Wheeler
Kevin Wilkinson
Gerd Zellweger
Yupu Zhang
Linux for The Machine team
Silicon Design Lab
The Machine Architectural Simulator team

Hewlett Packard Enterprise

©Copyright 2017 Hewlett Packard Enterprise Company

Wrapping up

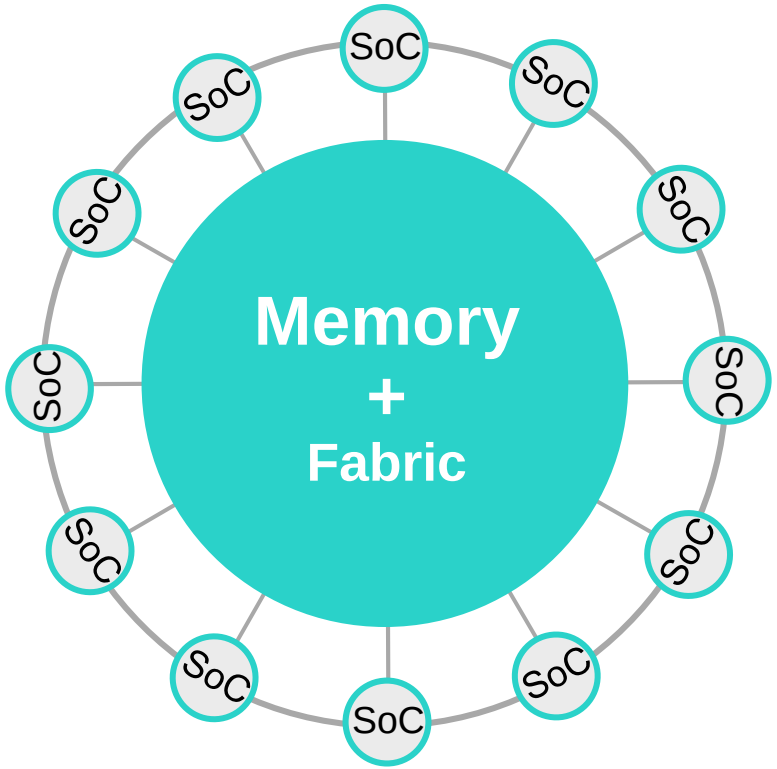
Memory-Driven Computing

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

Many opportunities for software innovation

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

How would *you* exploit Memory-Driven Computing?



Hewlett Packard Enterprise

©Copyright 2017 Hewlett Packard Enterprise Company