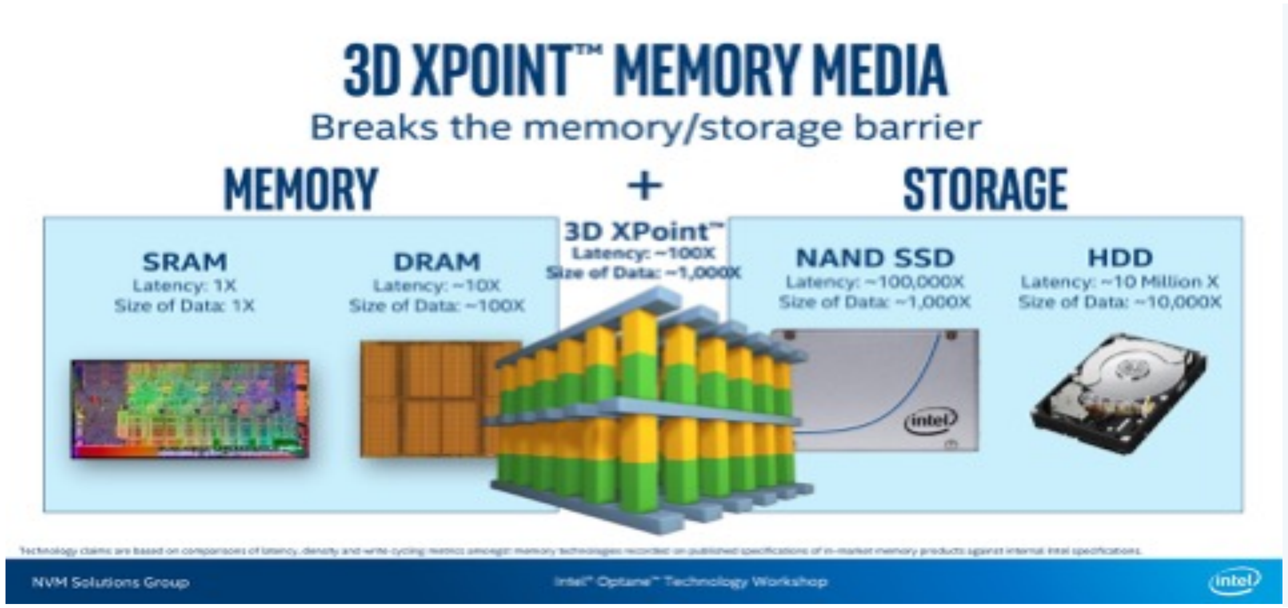
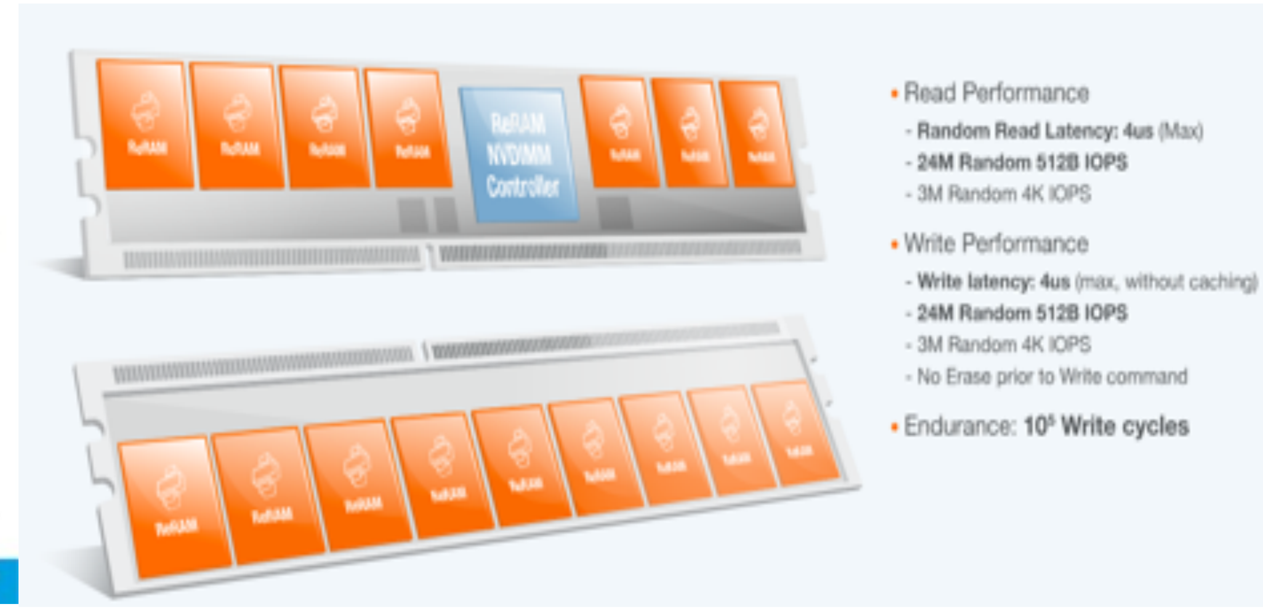
Distributed Shared Persistent Memory

(SoCC '17)

Yizhou Shan, Yiying Zhang

Persistent Memory (PM/NVM)





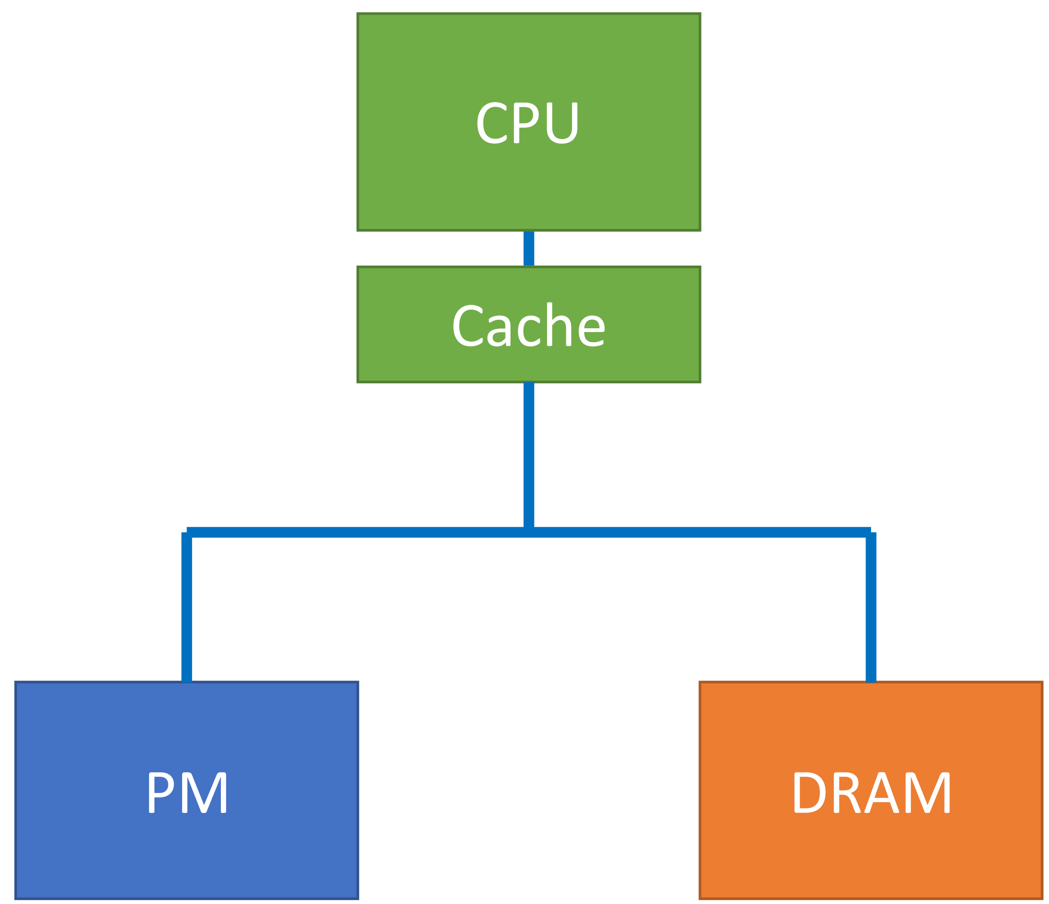
Byte Addressable

Persistent

Low Latency

Capacity

Cost effective



Many PM Work, but All in Single Machine

- Local memory models
  - NV-Heaps [ASPLOS '11], Mnemosyne [ASPLOS '11]
  - Memory Persistency [ISCA '14], Synchronous Ordering [Micro'16]

- Local file systems
  - BPFS [SOSP'09], PMFS [EuroSys'14], SCMFS [SC'11], HiNFS [EuroSys'16]

- Local transaction/logging systems
  - NVWAL [ASPLOS'16], SCT/DCT [ASPLOS'16], Kamino-Tx [Eurosys'17]

Moving PM into Datacenters

- PM fits datacenter
  - Applications require a lot memory
  - and accessing persistent data fast
  - with low monetary cost

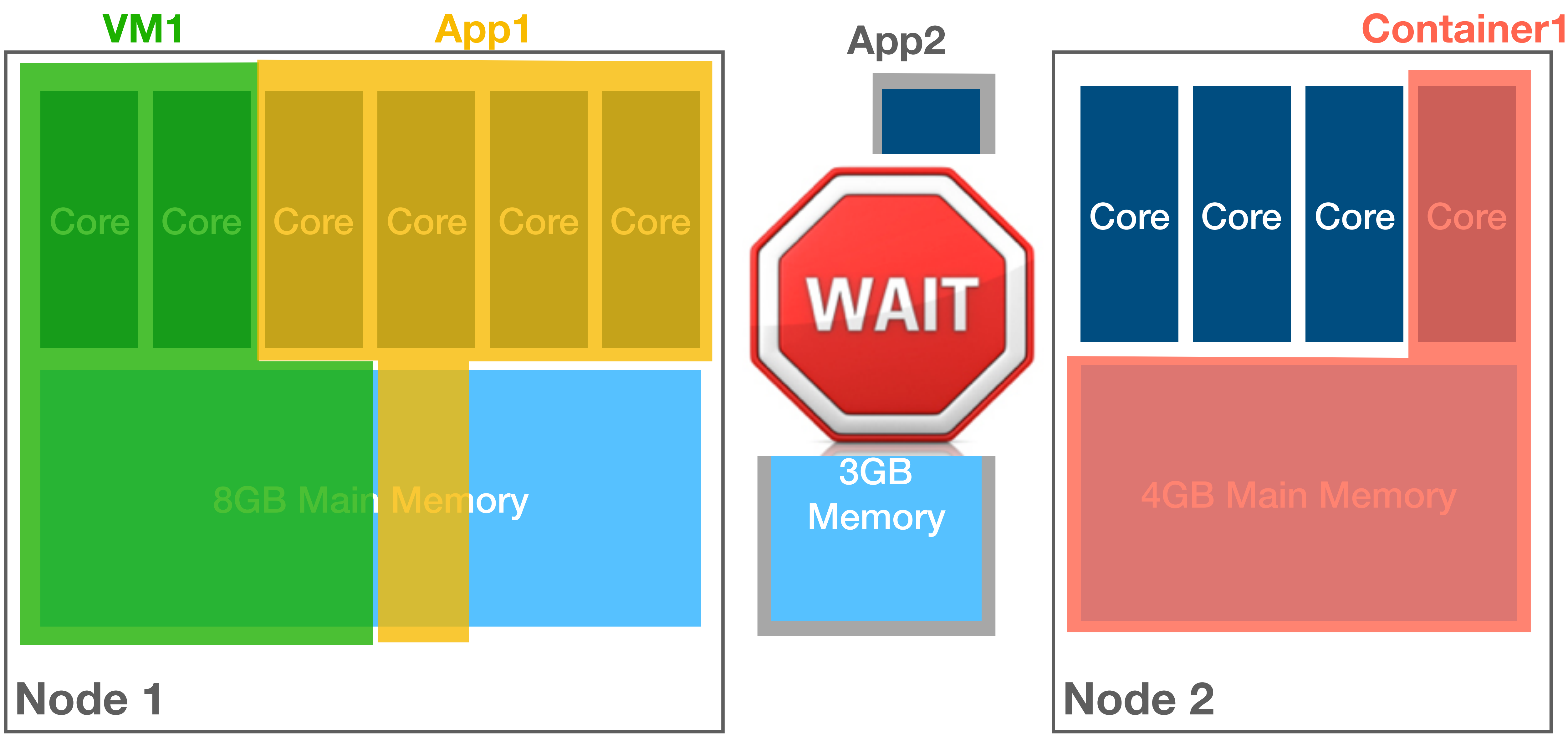
- Challenges
  - Handle node failure
  - Ensure good performance and scalability
  - Easy-to-use abstraction

How to Use PM in Distributed Environments?

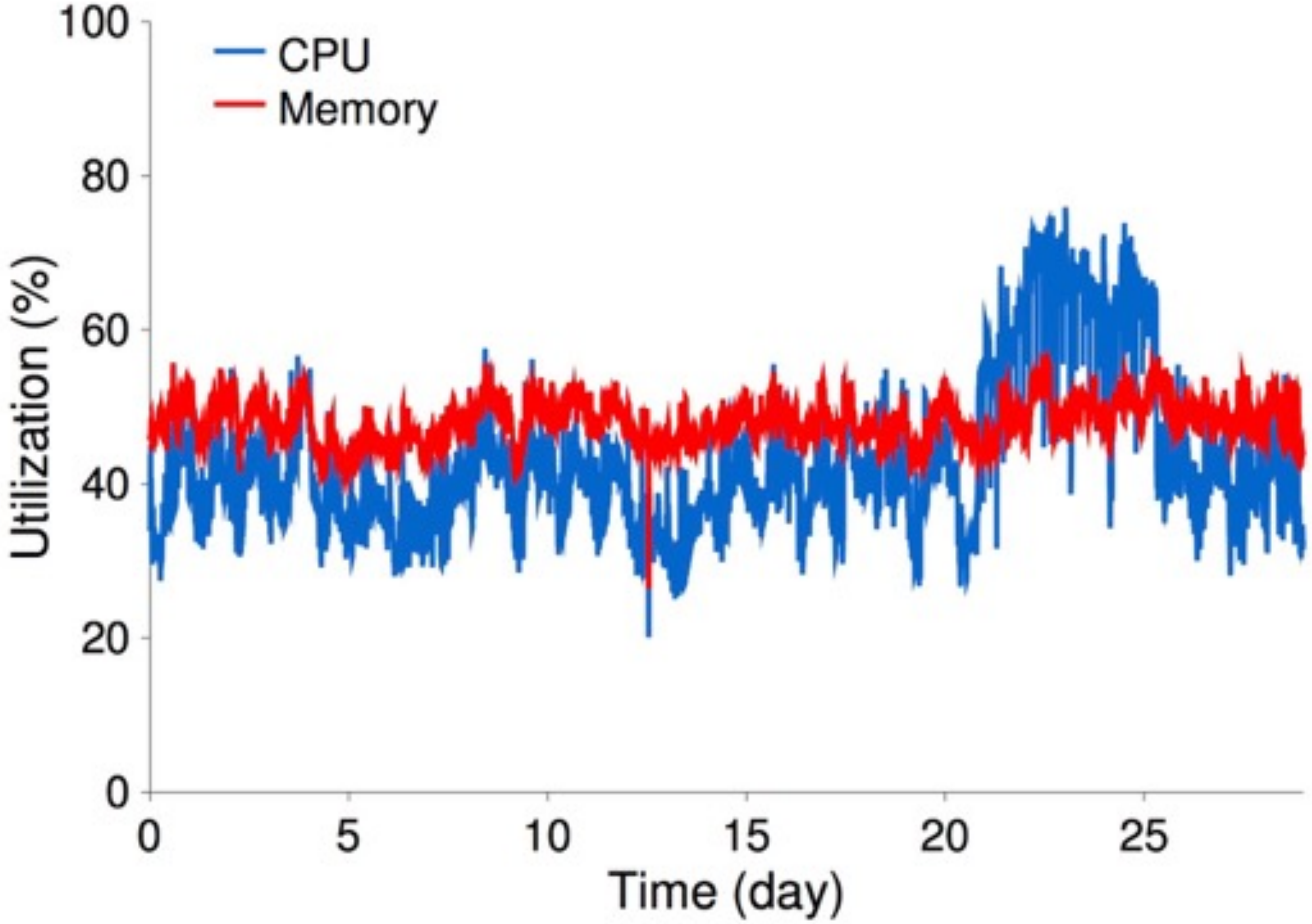
- As distributed memory?
- As distributed storage?

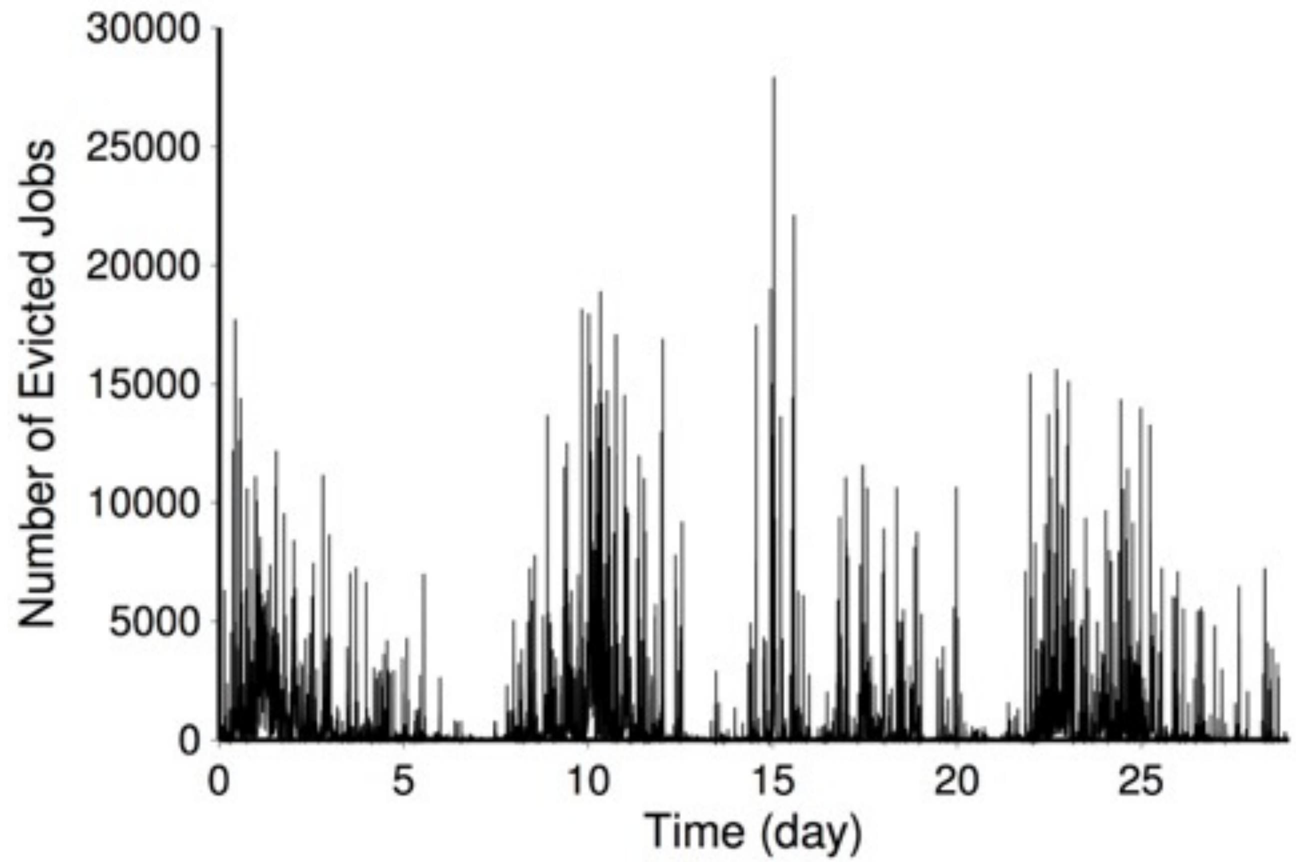
- Mojim *[Zhang etal., ASPLOS'15]*
    - ***First*** PM work in distributed environments
    - Efficient **PM replication**
    - *But far from a full-fledged distributed NVM system*

Resource Allocation in Datacenters



Resource Utilization in Production Clusters





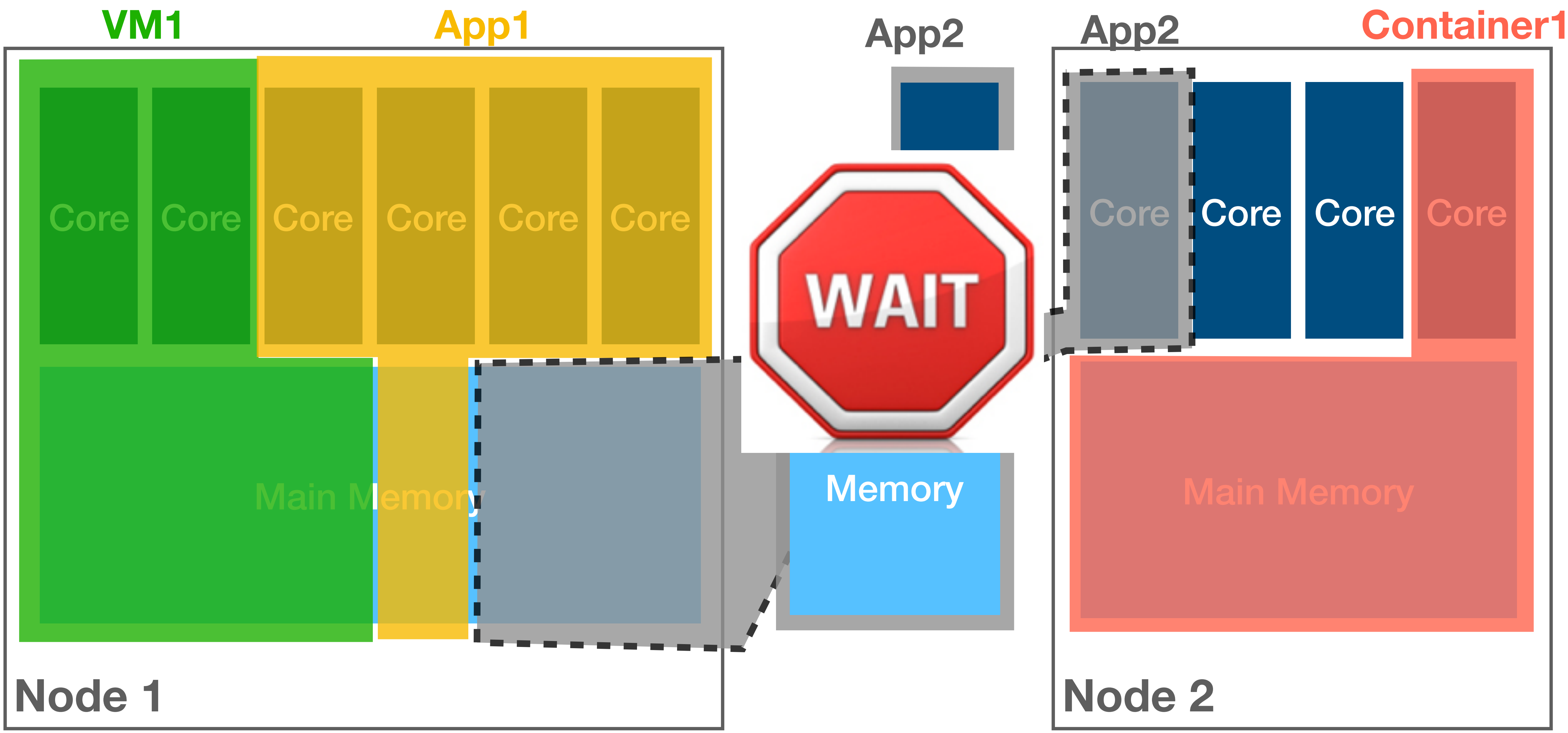
* Google Production Cluster Trace Data. "https://github.com/google/cluster-data"

Unused Resource + Waiting/Killed Jobs Because of Physical-Node Constraints

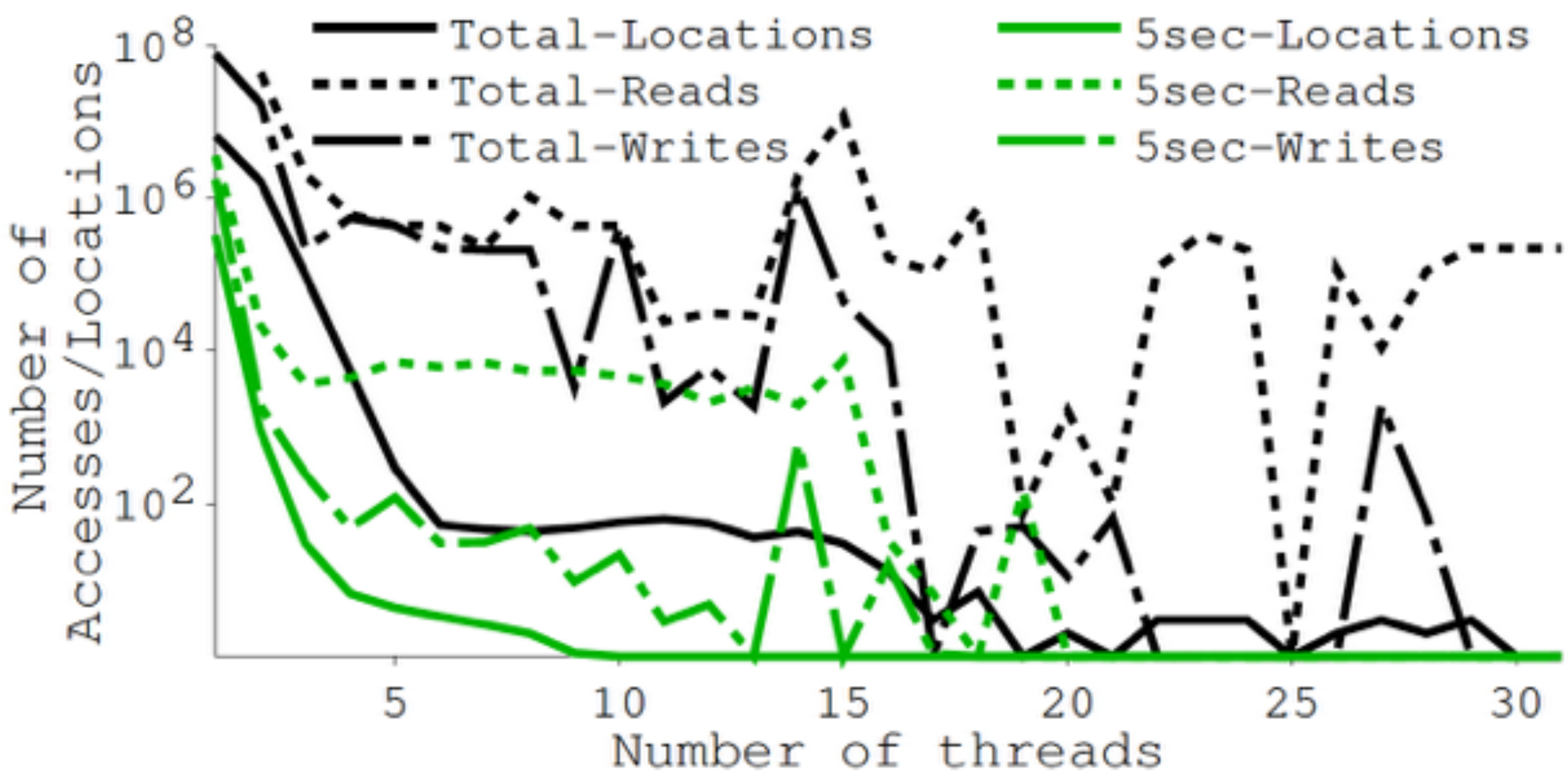
Q1: How to achieve better resource utilization?

Use remote memory

Distributed (Remote) Memory

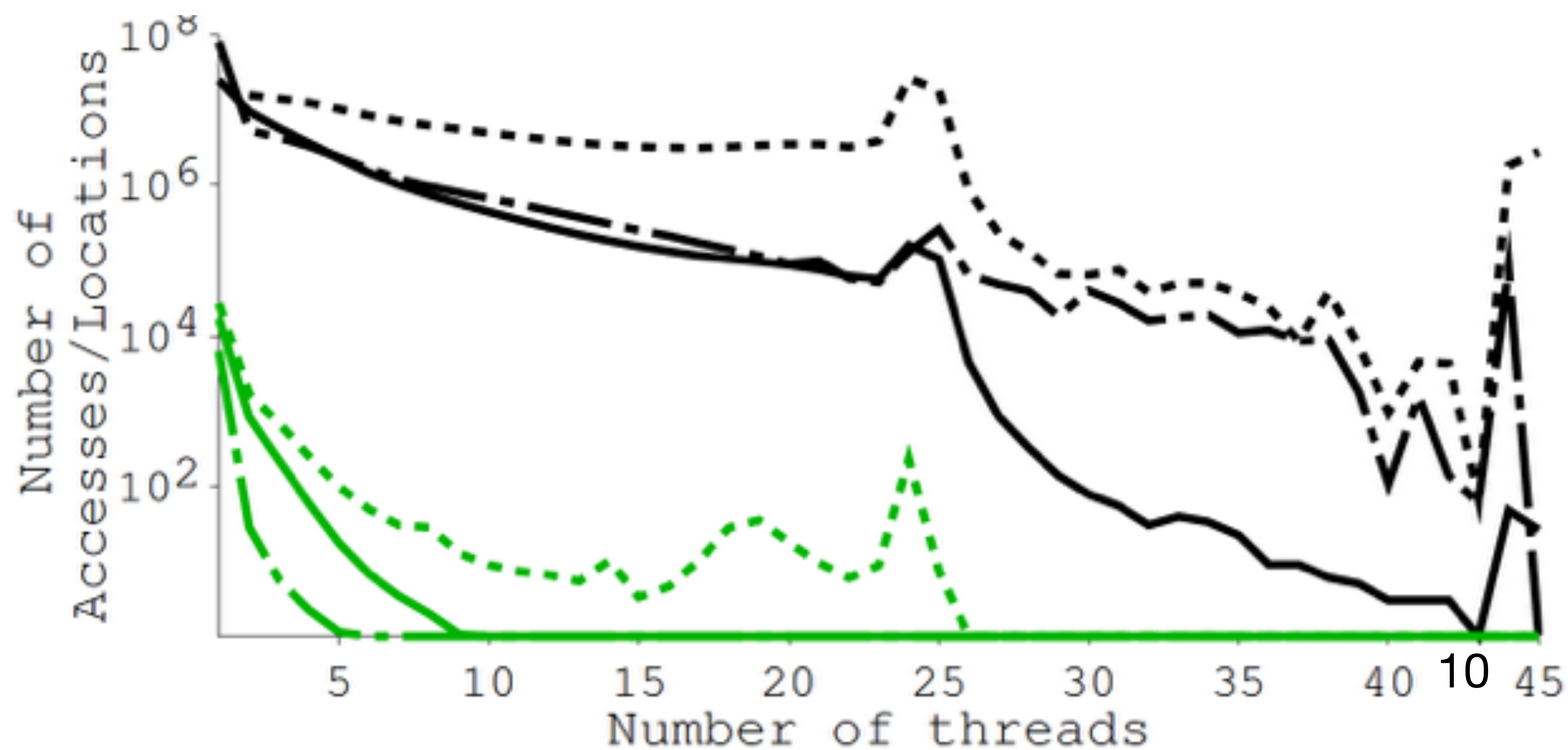


Modern Datacenter Applications Have Significant Memory Sharing



PowerGraph

TensorFlow



Q2: How to scale out parallel applications?

Distributed **shared** memory

What about persistence?

- Data persistence is useful
  - Many existing data storage systems

  ➡ Performance
  - Memory-based, long-running applications

  ➡ Checkpointing

Q3: How to provide data persistence?

DSM

Distributed Shared Persistent Memory (DSPM)

a significant step towards using PM in datacenters

DSPM

- Native memory load/store interface
  - Local or remote (transparent)
  - Pointers and in-memory data structures

- Supports memory read/write sharing

DSM

Distributed Shared Persistent Memory (DSPM)

a significant step towards using PM in datacenters

DSPM

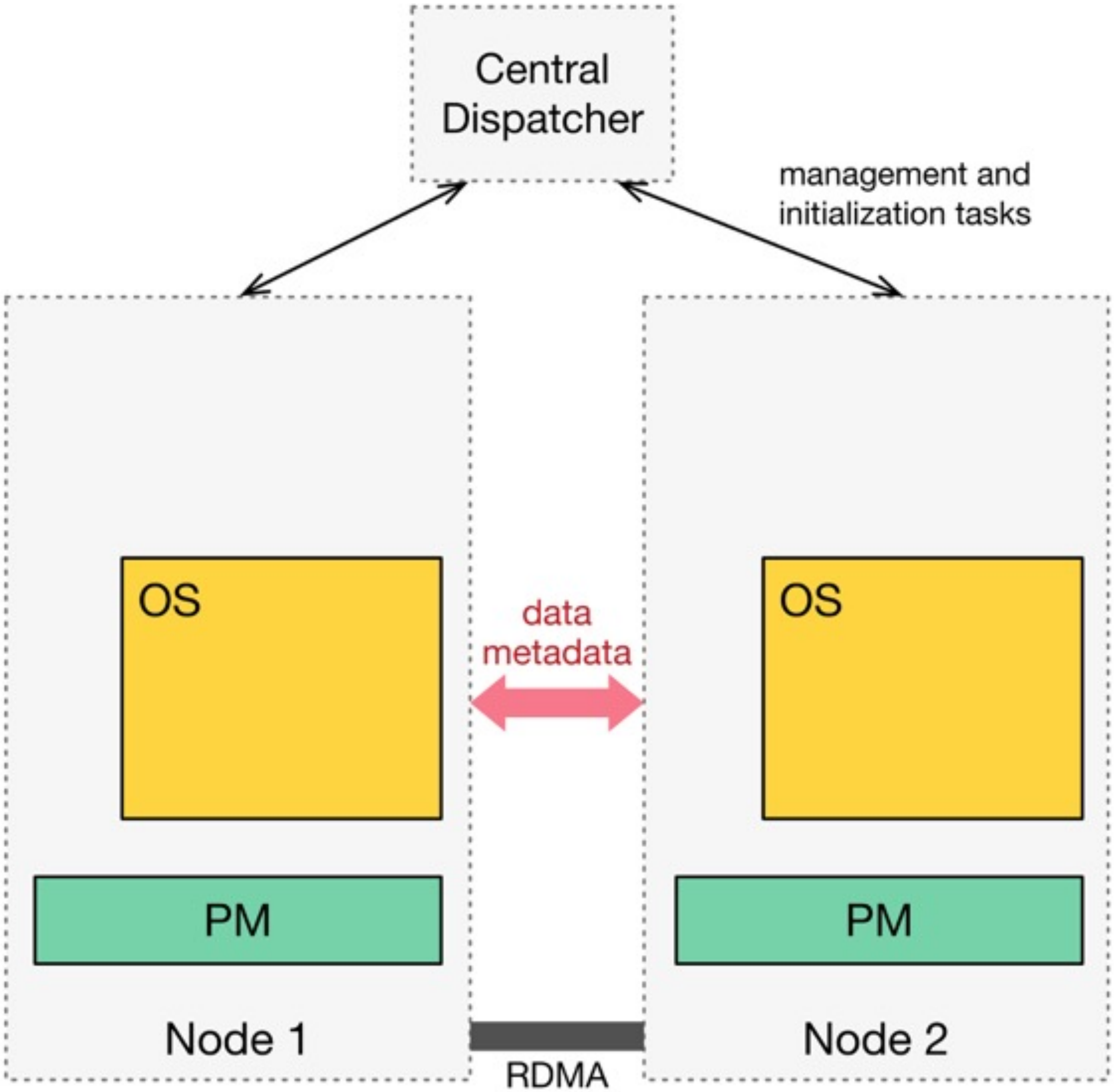
DSPM: One Layer Approach

Benefits of both memory and storage
No redundant layers
No data marshaling/unmarshalling

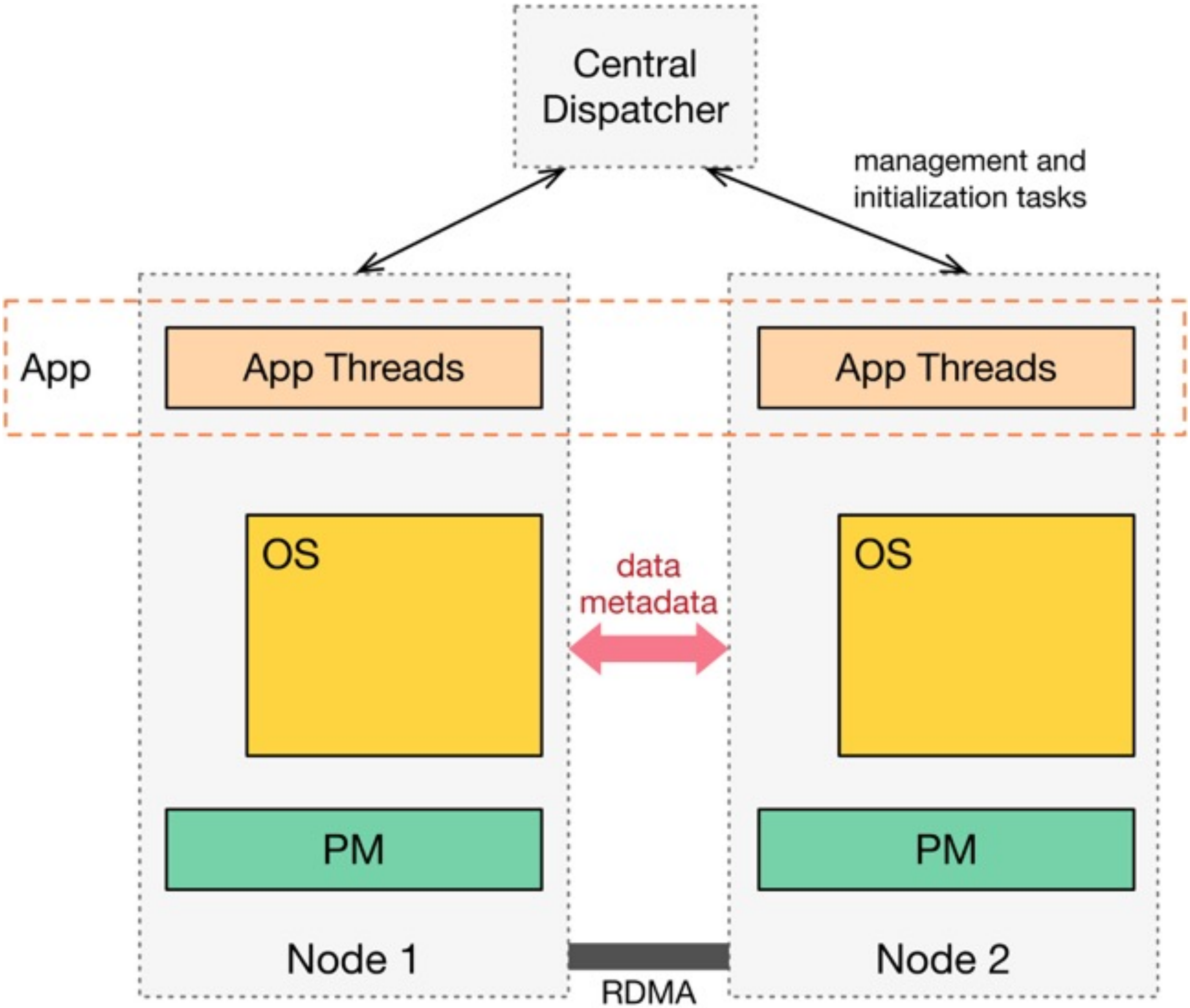
Hotpot: A Kernel-Level RDMA-Based DSPM System

- Easy to use
- Native memory interface
- Fast, scalable
- Flexible consistency levels
- Data durability & reliability

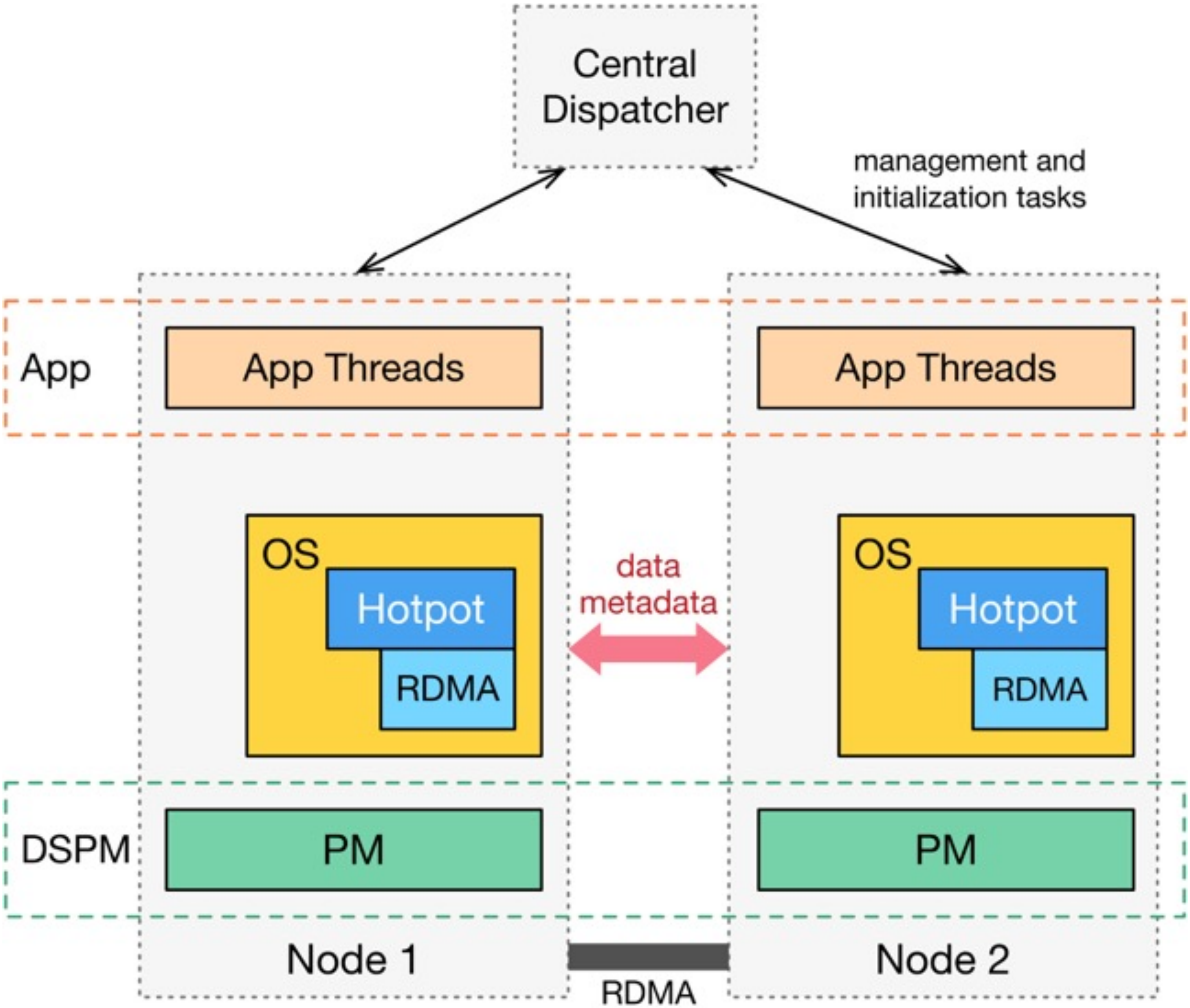
Hotpot Architecture



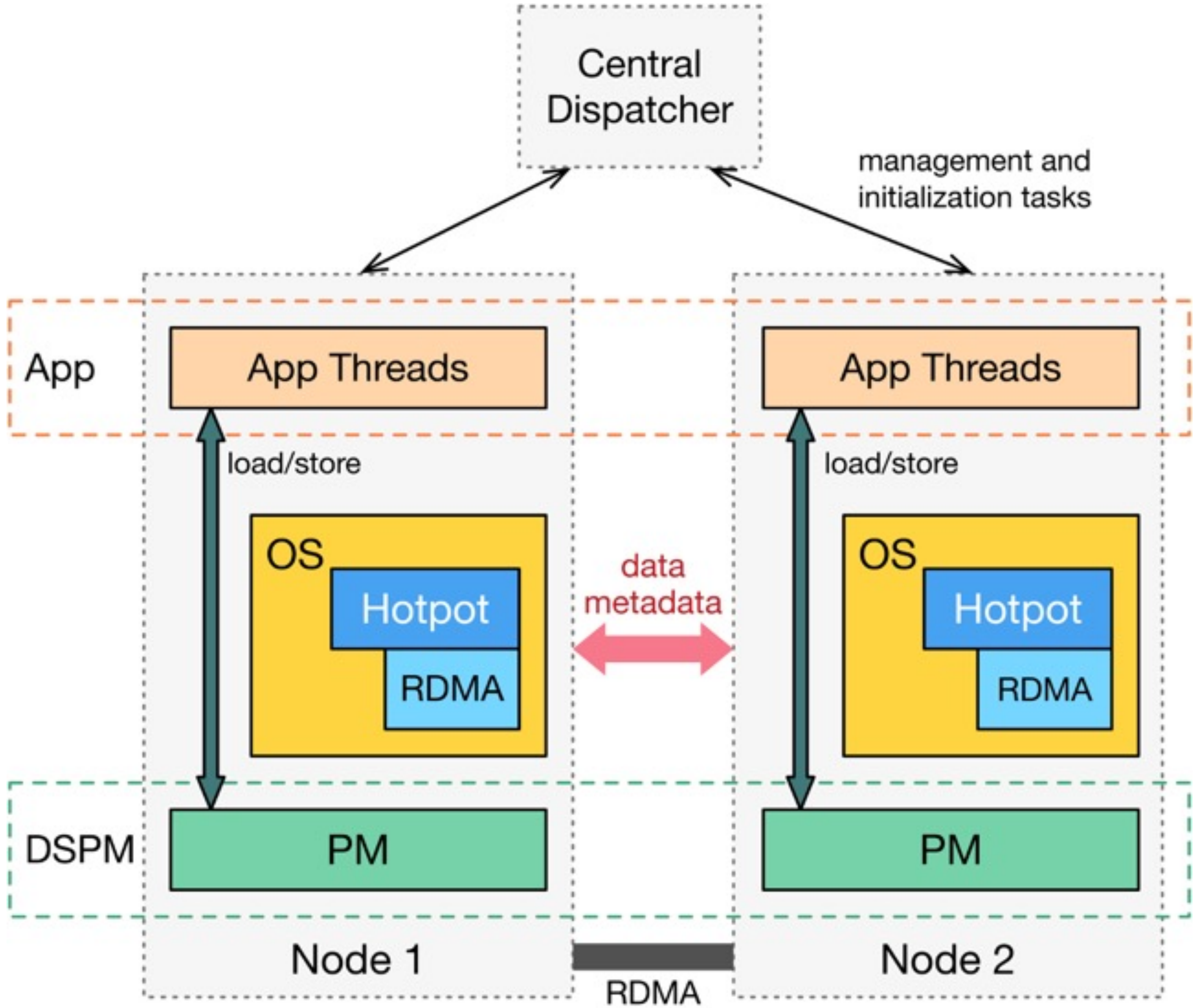
Hotpot Architecture



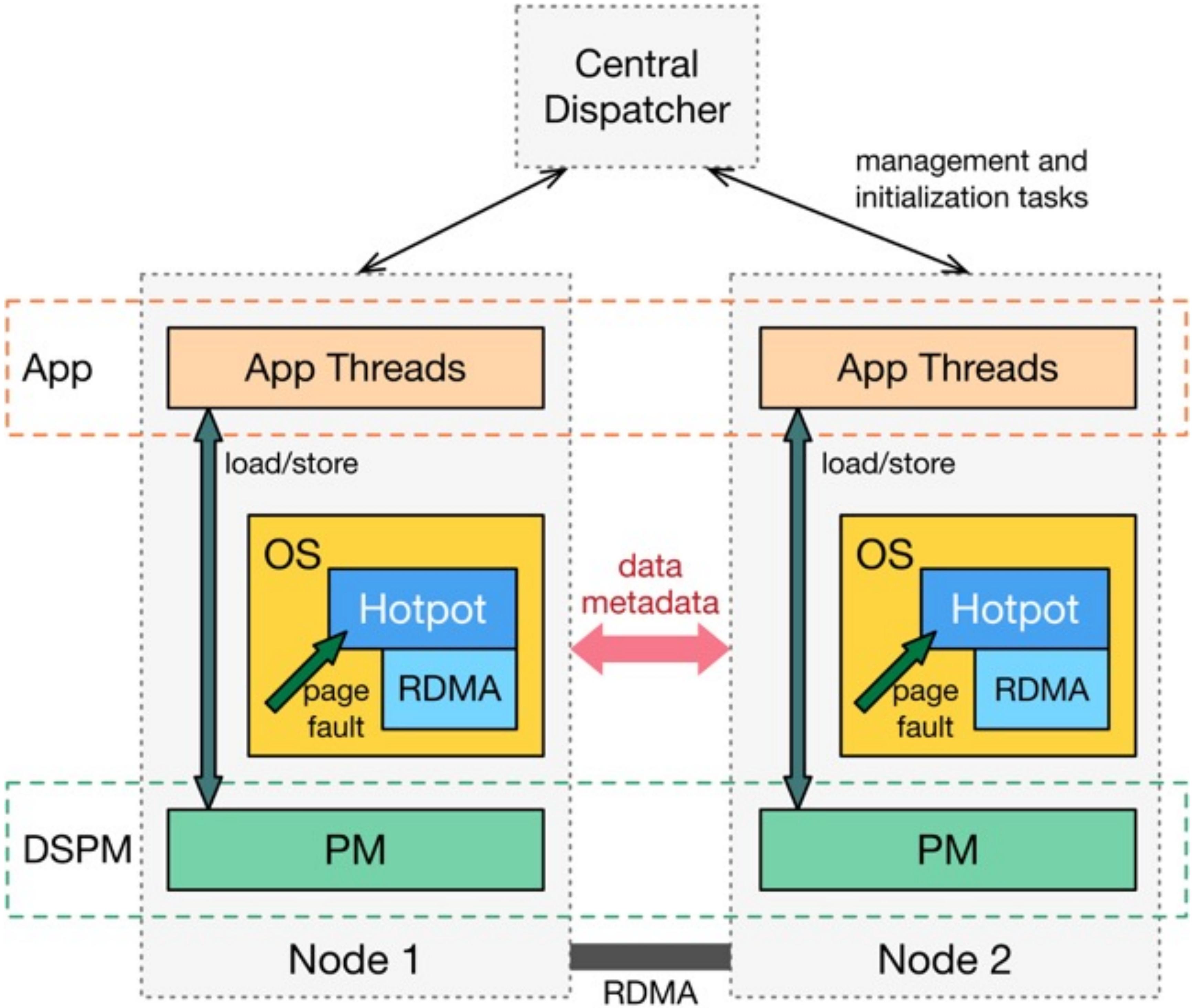
Hotpot Architecture



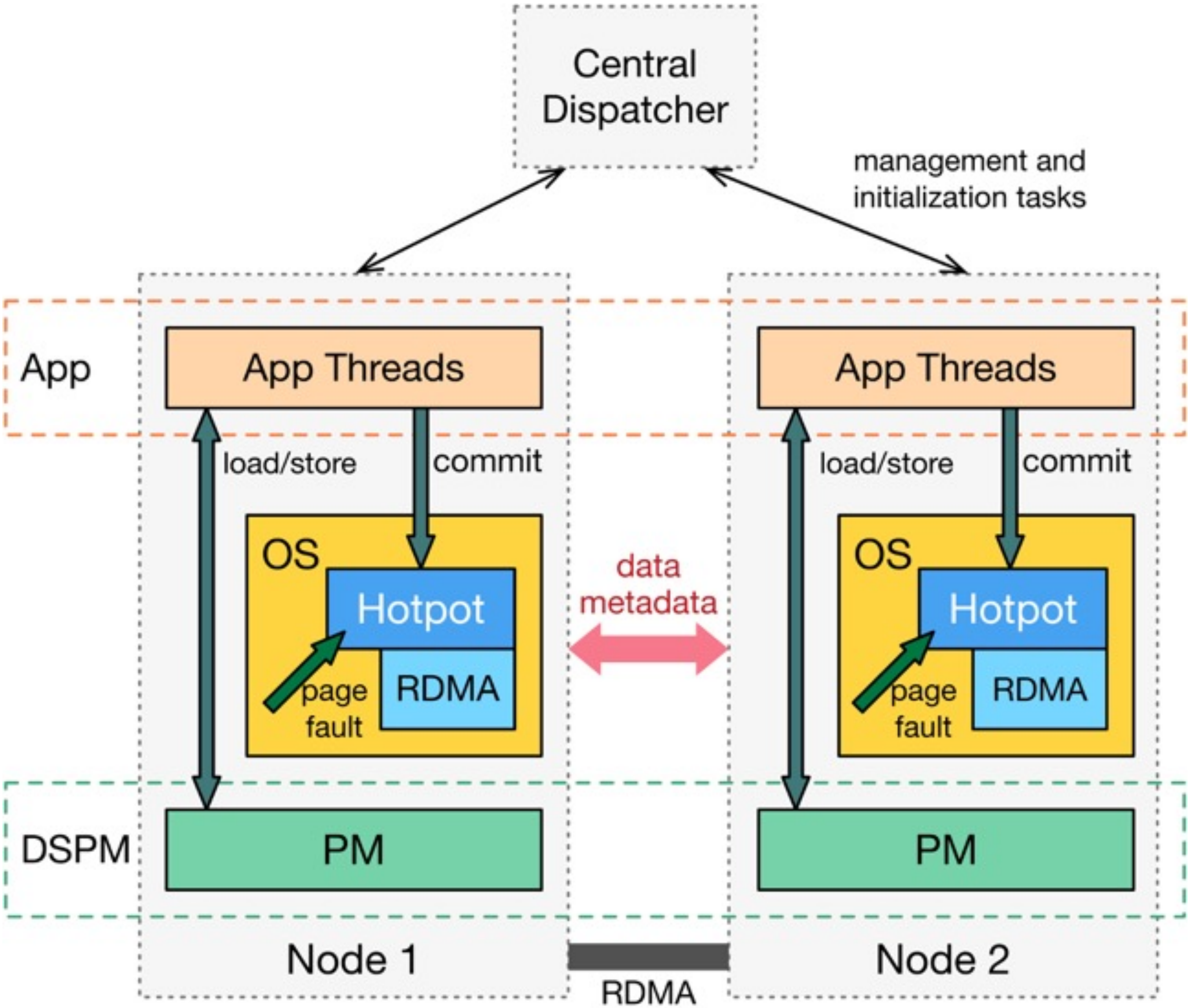
Hotpot Architecture



Hotpot Architecture



Hotpot Architecture



Hotpot Code Example

```
/* Open a dataset named 'boilermaker' */
int fd = open("/mnt/hotpot/boilermaker", O_CREAT|O_RDWR);
/* map it to application's virtual address space */
void *base = mmap(0, 40960, PROT_WRITE, MAP_PRIVATE, fd, 0);

/* First access: Hotpot will fetch page from remote */
*base = 9;

/* Later accesses: Direct memory load/store */
memset(base, 0x27, PAGE_SIZE);

/* Commit data: making data coherent, durable, and replicated */
msync(sg_addr, sg_len, MSYNC_HOTPOT);
```

How to efficiently add P to “DSM”?

- Distributed Shared Memory
  - Cache remote memory ***on-demand*** for fast local access
  - Multiple ***redundant*** copies

- Distributed Storage Systems
  - ***Actively*** add more ***redundancy*** to provide data reliability

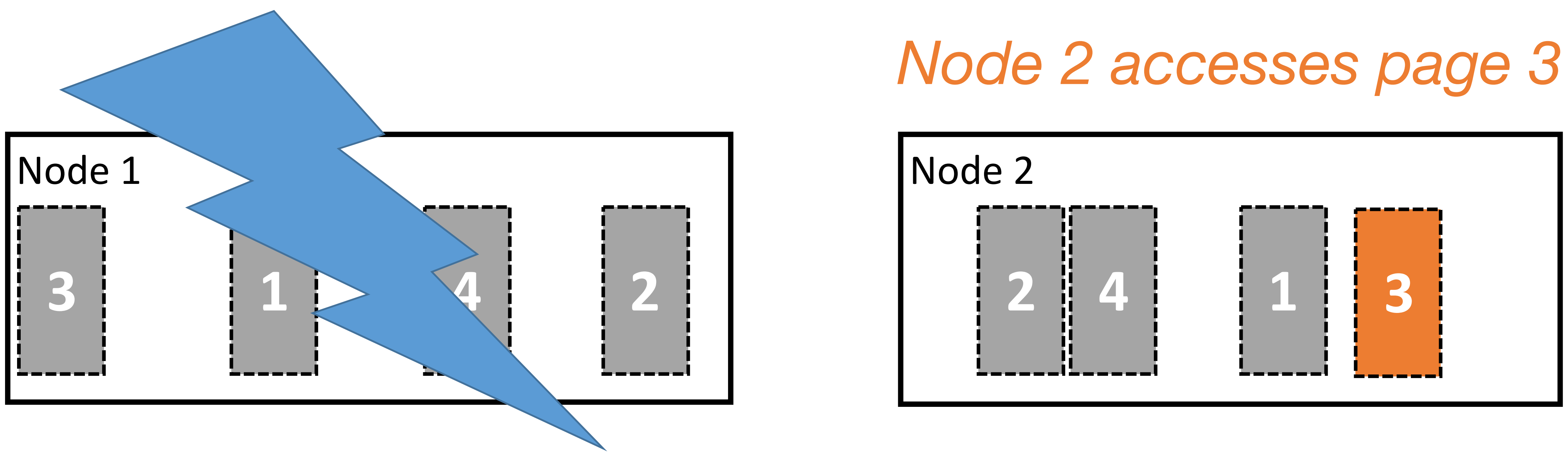
Integrate two forms of redundancy with *morphable page states*

One Layer Principle

Morphable Page States

- A PM page can serve different purposes, possibly at different times
  - as a local cached copy to improve performance
  - as a redundant data page to improve data reliability

Node 2 accesses page 3



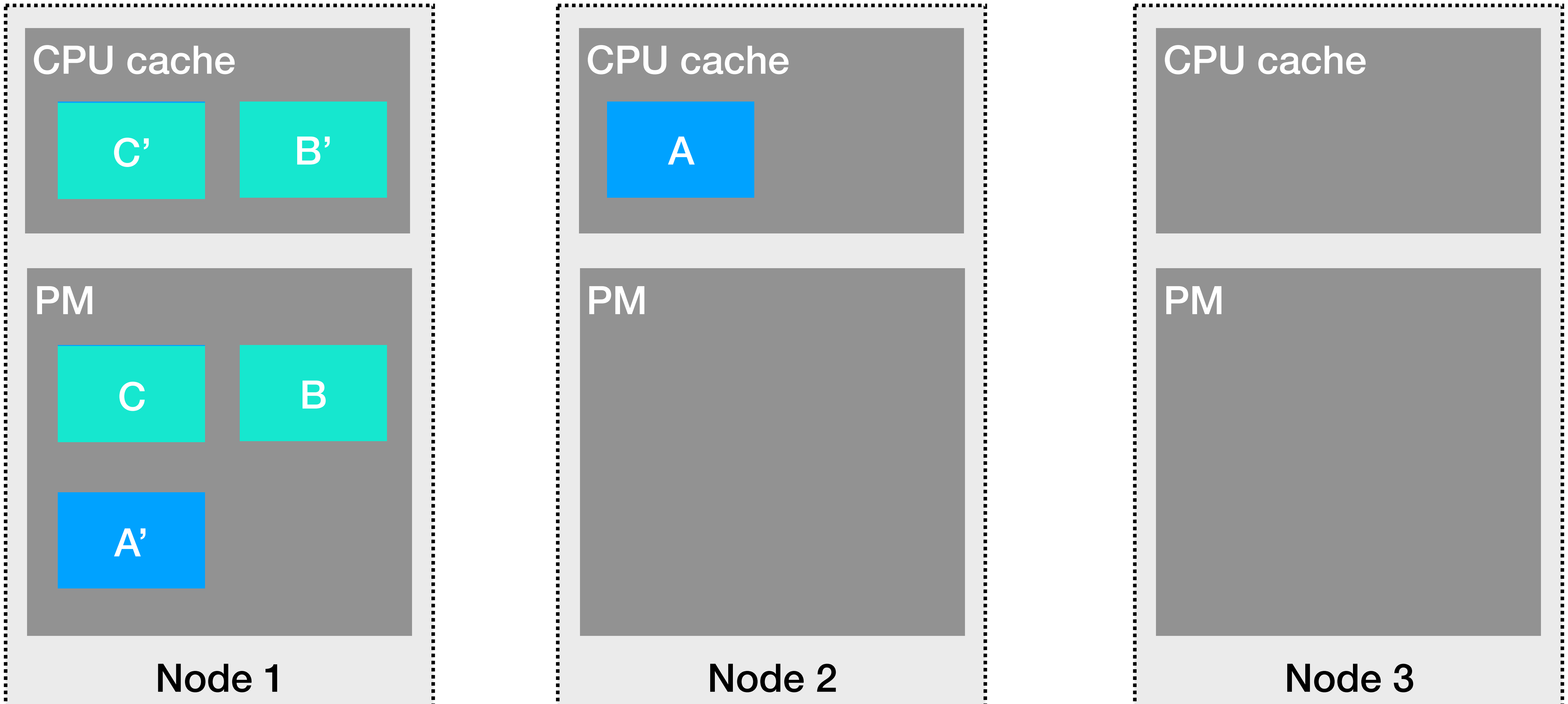
How to efficiently add P to “DSM”?

- When to make cached copies coherent?
- When to make data durable and reliability?

- Observations
  - Data-store applications have well-defined ***commit points***
  - Commit points: time to make data persistent
  - Visible to storage devices => visible to other nodes

Exploit application behavior:
Make data coherent only at commit points

Commit Point



- durable
- coherent
- reliable
- single-node and distributed consistency
- two consistency modes: single/multiple writer

Flexible Coherence Levels

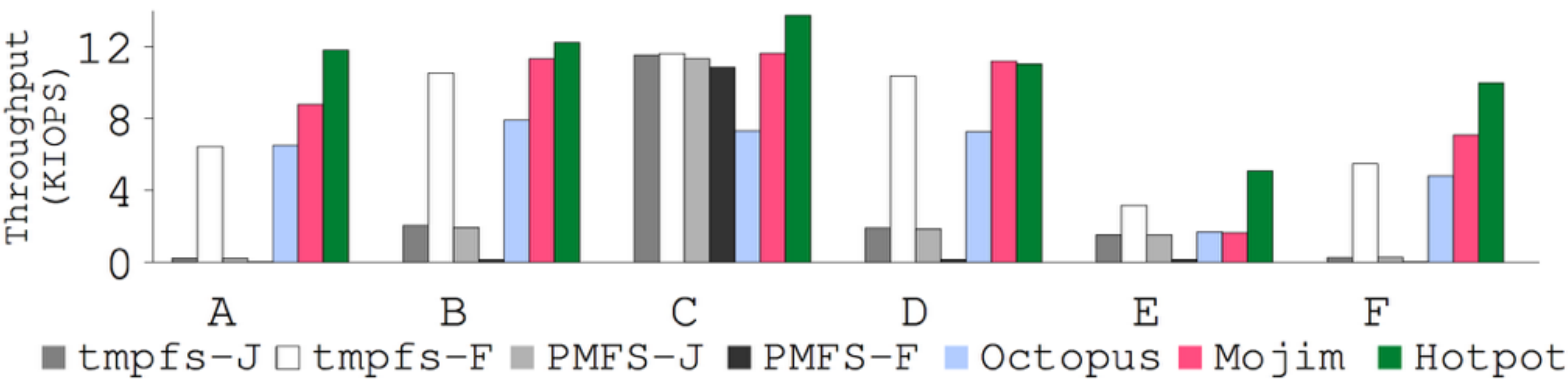
- Multiple Reader Multiple Writer (***MRMW***)
  - Allows multiple concurrent dirty copies
  - Great parallelism, but weaker consistency
  - ***Three-phase*** commit protocol

- Multiple Reader Single Writer (***MRSW***)
  - Allows only one dirty copy
  - Trades parallelism for stronger consistency
  - ***Single phase*** commit protocol

MongoDB Results

- Modify MongoDB with ~120 LOC, use MRMW mode
- Compare with ***tmpfs***, ***PMFS***, ***Mojim***, ***Octopus*** using ***YCSB***

Workload	Read	Update	Scan	Insert	R&U
A	50%	50%	-	-	-
B	95%	5%	-	-	-
C	100%	-	-	-	-
D	95%	-	-	5%	-
E	-	-	95%	5%	-
F	50%	-	-	-	50%



Conclusion

- One layer approach: challenges and benefits
- Hotpot: a kernel-level RDMA-based DSPM system
- Hide complexity behind simple abstraction
- Calls for attention to use PM in datacenter
- **Many open problems in distributed PM!**

Thank You
Questions?

Get Hotpot at: *https://github.com/WukLab/Hotpot*

WukLab

wuklab.io