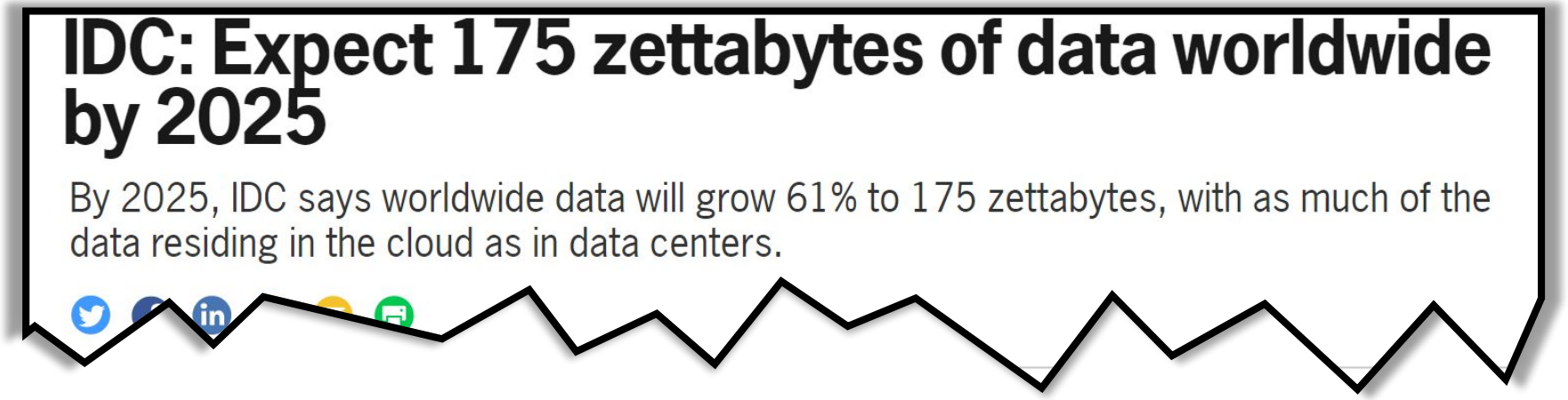
Scalable Storage Stack for Persistent Memory

Sanidhya Kashyap

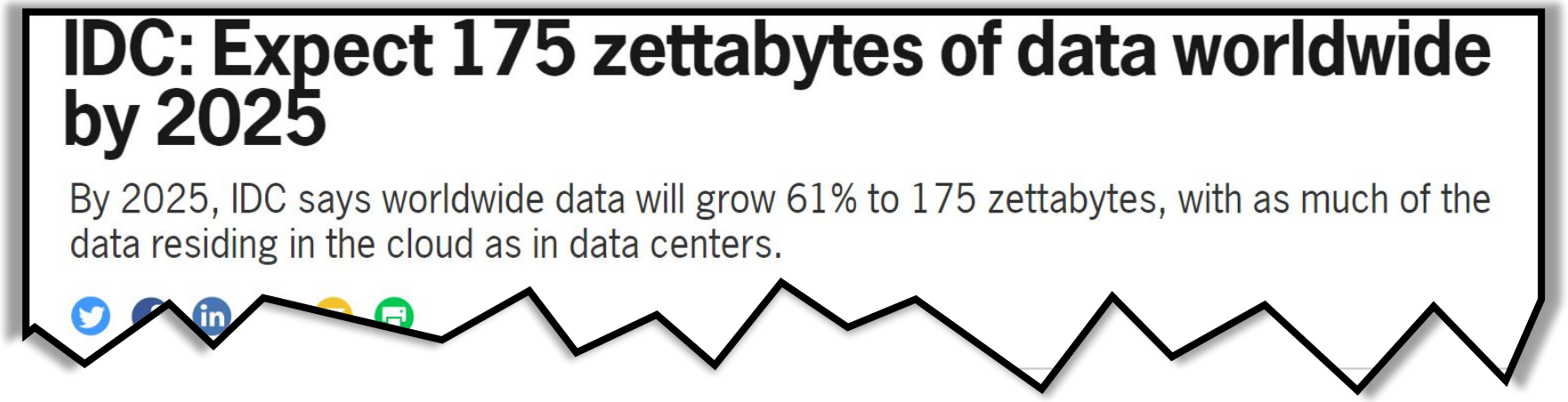
EPFL

Data is growing by 61% every year



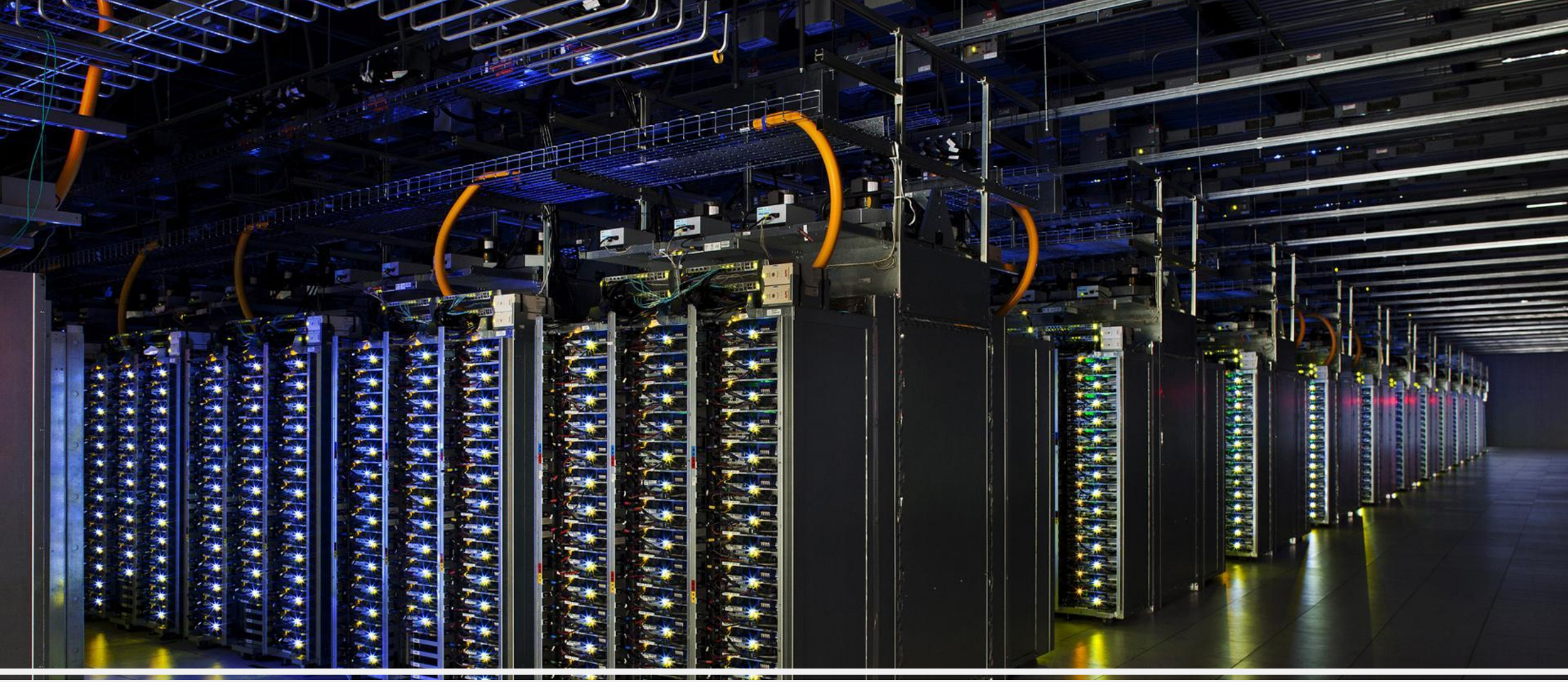
Source: NetworkWorld (2018) / Data Age 2025 by IDC (2017)

Data is growing by 61% every year



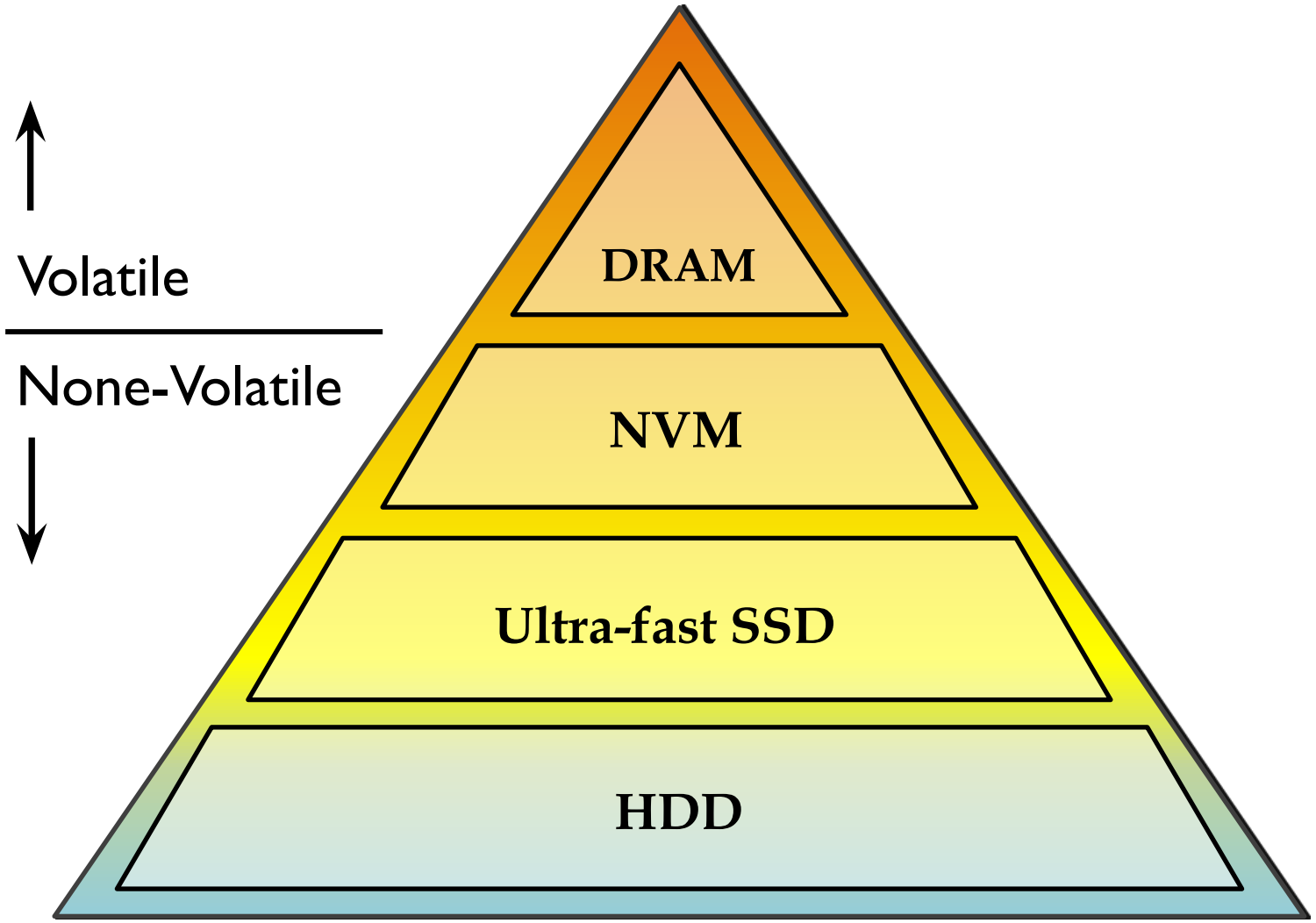
Data requires more *computation* and more **storage**

Source: NetworkWorld (2018) / Data Age 2025 by IDC (2017)



Data centers are responsible for handling data

Persistent memory (PM) in the storage jungle



	Latency	B/W	$/GB
DRAM	**300 ns**	**10 GB/s**	**4.0**
NVM	40 us	2.0 MB/s	0.25
Ultra-fast SSD			
HDD	5 ms	2.6 MB/s	0.02

Persistent memory brings …

1. **Byte addressability**
   - Loads & stores at the granularity of bytes
   - Faster random data access
2. **Non-volatility**
   - Persists data across power cycle
3. **Large capacity**
   - 3TB-per socket

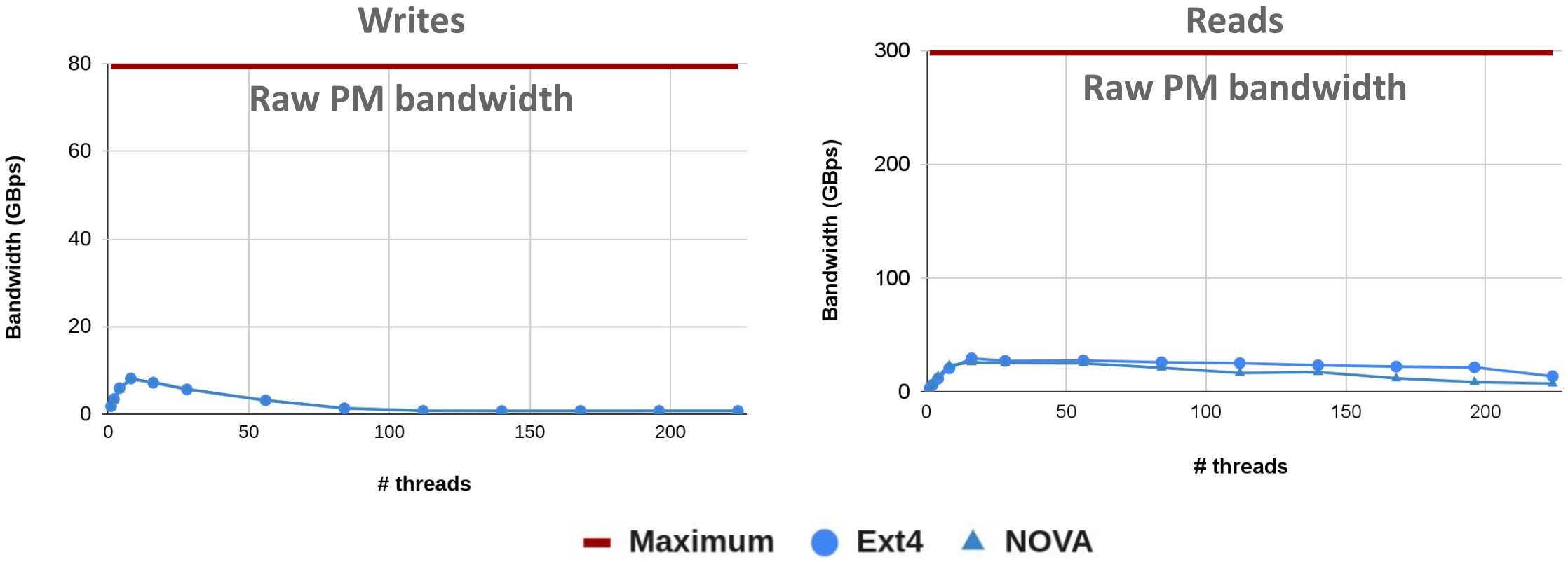
Persistent memory brings …

1. **Byte addressability**
   - Loads & stores at the granularity of bytes
   - Faster random data access
2. **Non-volatility**
   - Persists data across power cycle
3. **Large capacity**
   - 3TB-per socket

Exciting opportunity for **faster storage stack!**

Current IO performance with PM

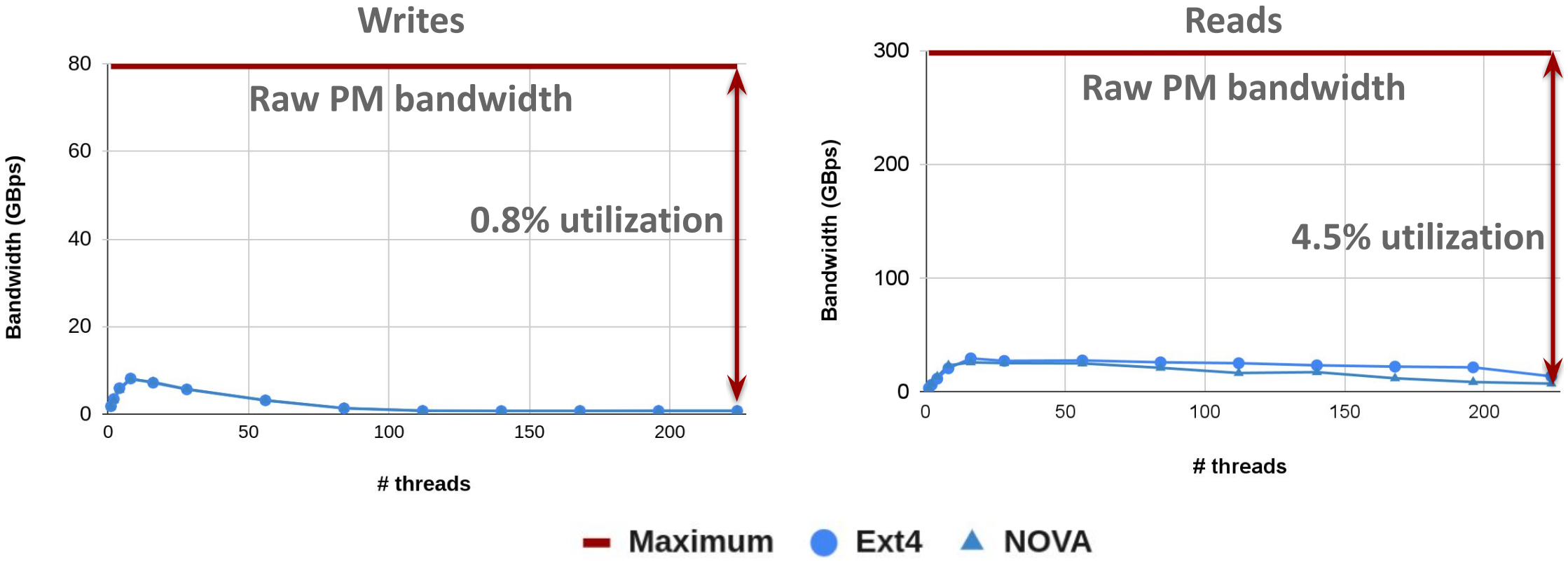
Benchmark: Each thread reads/writes 2MB data privately in a 1GB file



Setup: 224-core/8-socket machine

Current IO performance with PM

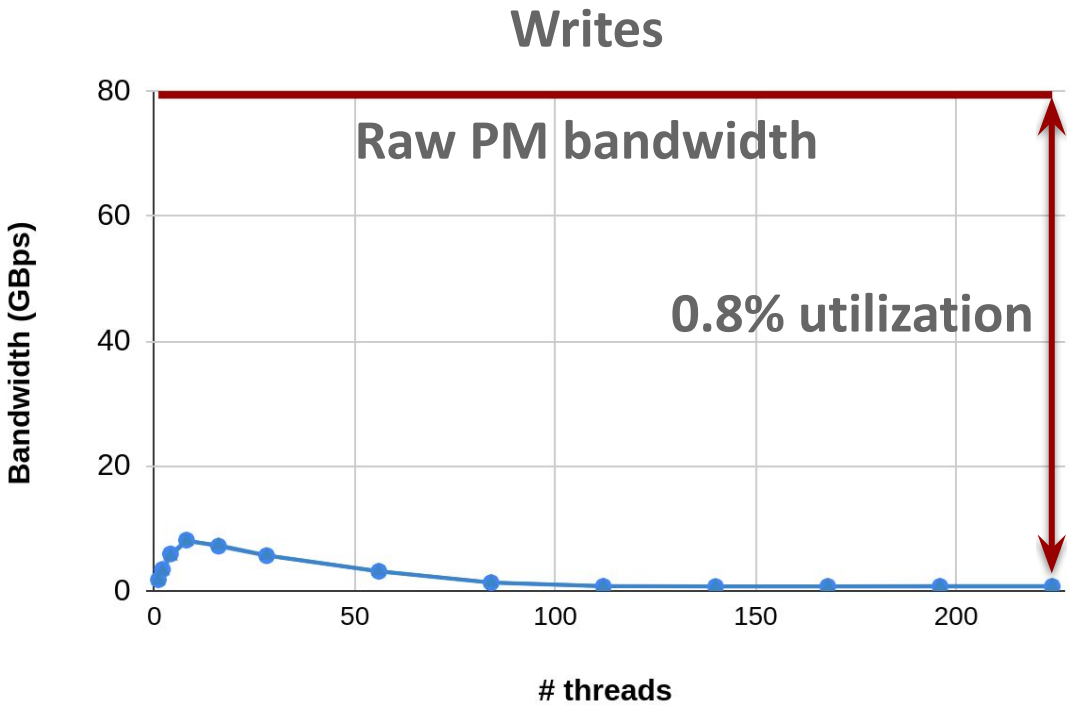
Benchmark: Each thread reads/writes 2MB data privately in a 1GB file

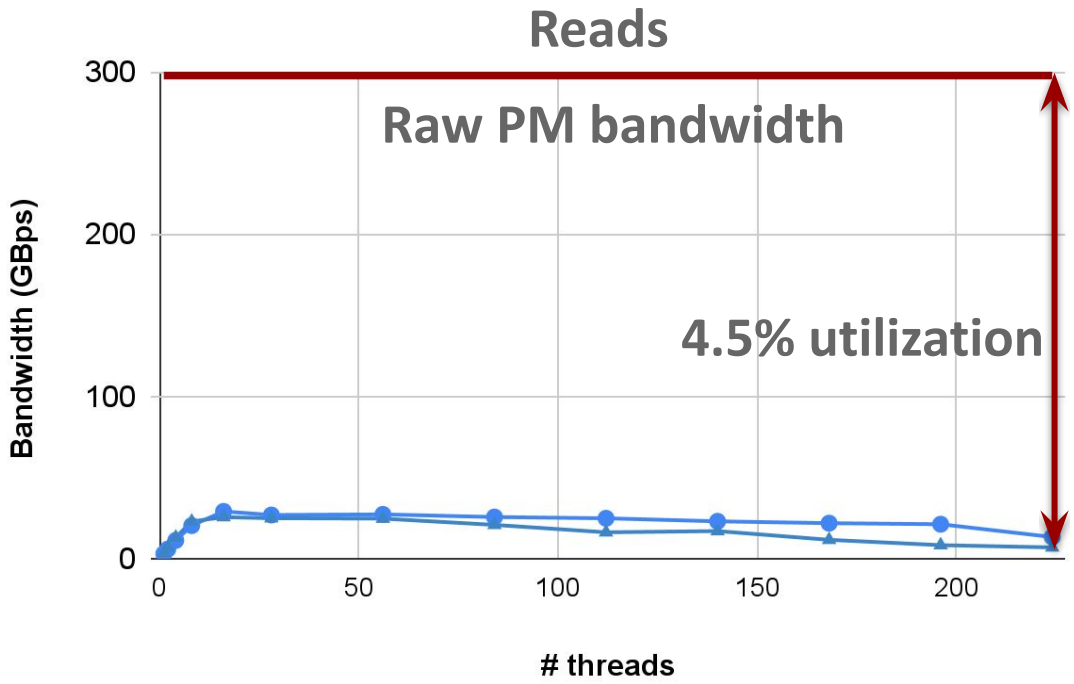


Setup: 224-core/8-socket machine

Current IO performance with PM

Benchmark: Each thread reads/writes 2MB data privately in a 1GB file





Unable to utilize existing PM's capability

Goal of our new file system design

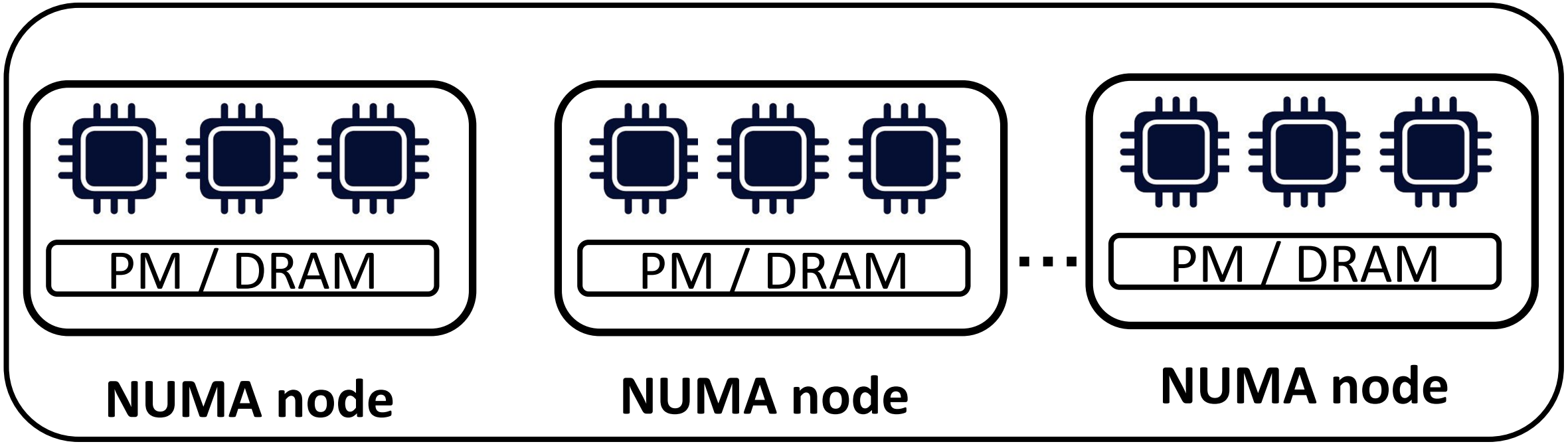
1) **Maximize PM's bandwidth utilization**

2) **Scale with increasing thread/core count**

Current hardware setup

PM DIMMs are attached to multiple different NUMA nodes

NUMA: Accessing the local socket memory is faster than the remote socket



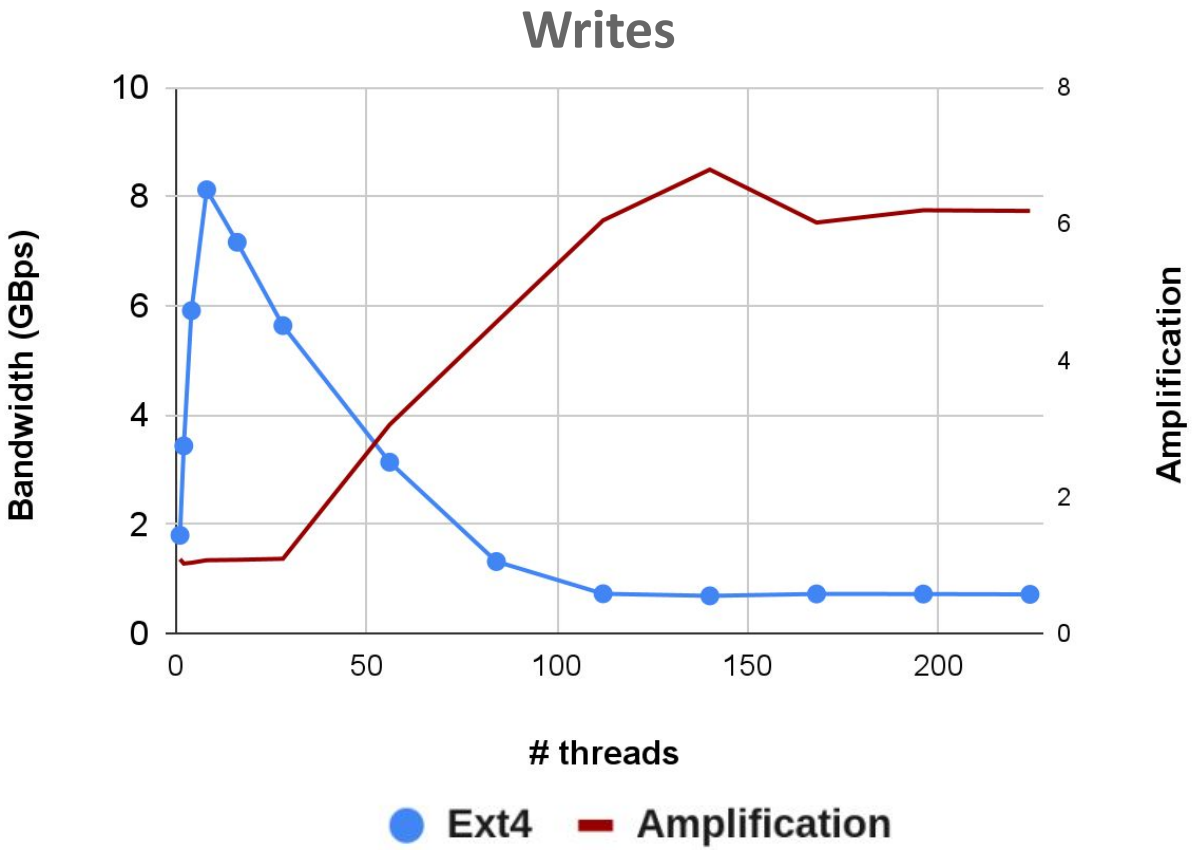
Hardware setup: Consider one NUMA node

PM DIMMs are attached to multiple different NUMA nodes

NUMA: Accessing the local socket memory is faster than the remote socket

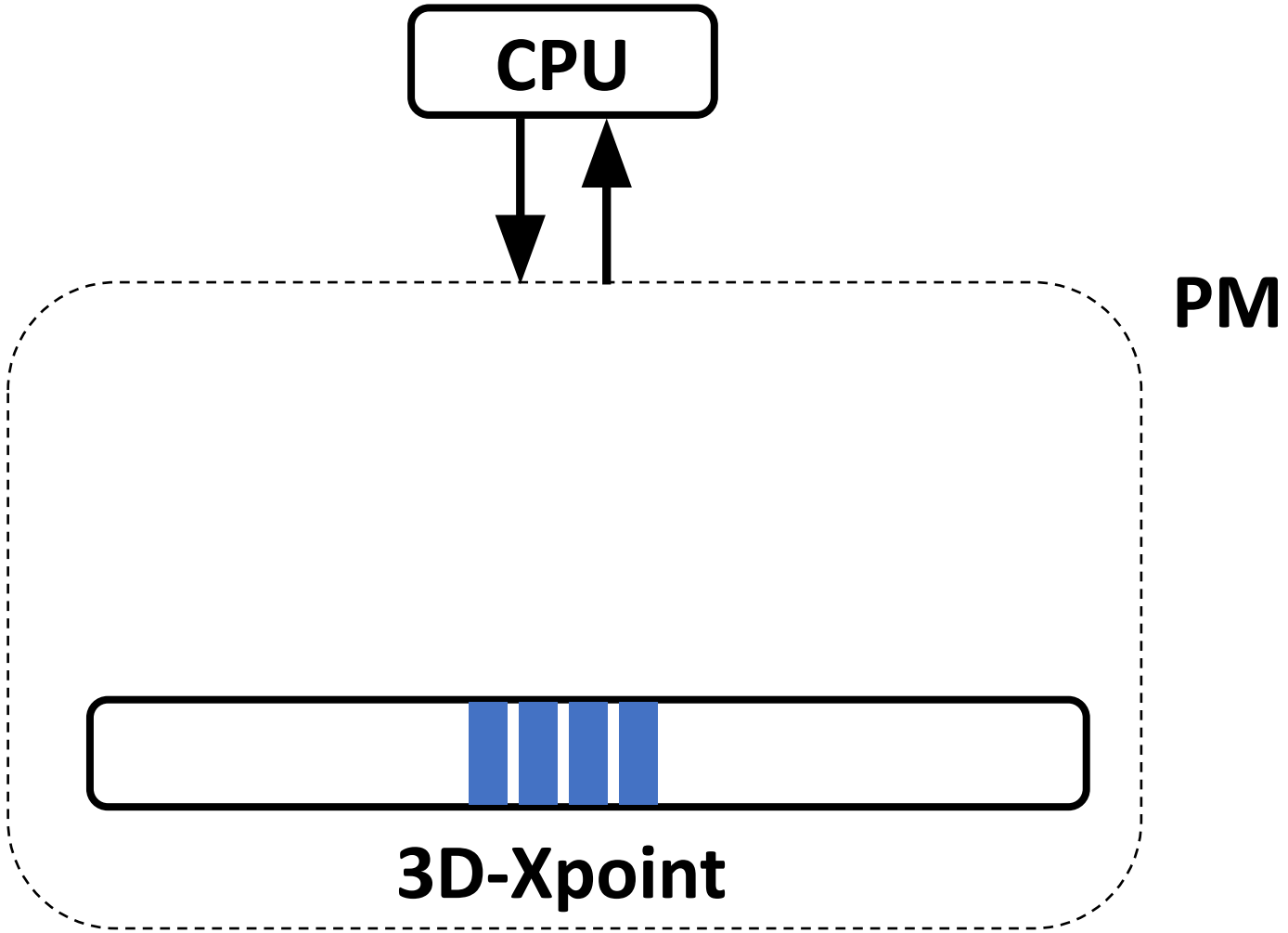
Concurrent access degrades IO performance

Benchmark: Each thread reads/writes 2MB data privately in a 1GB file



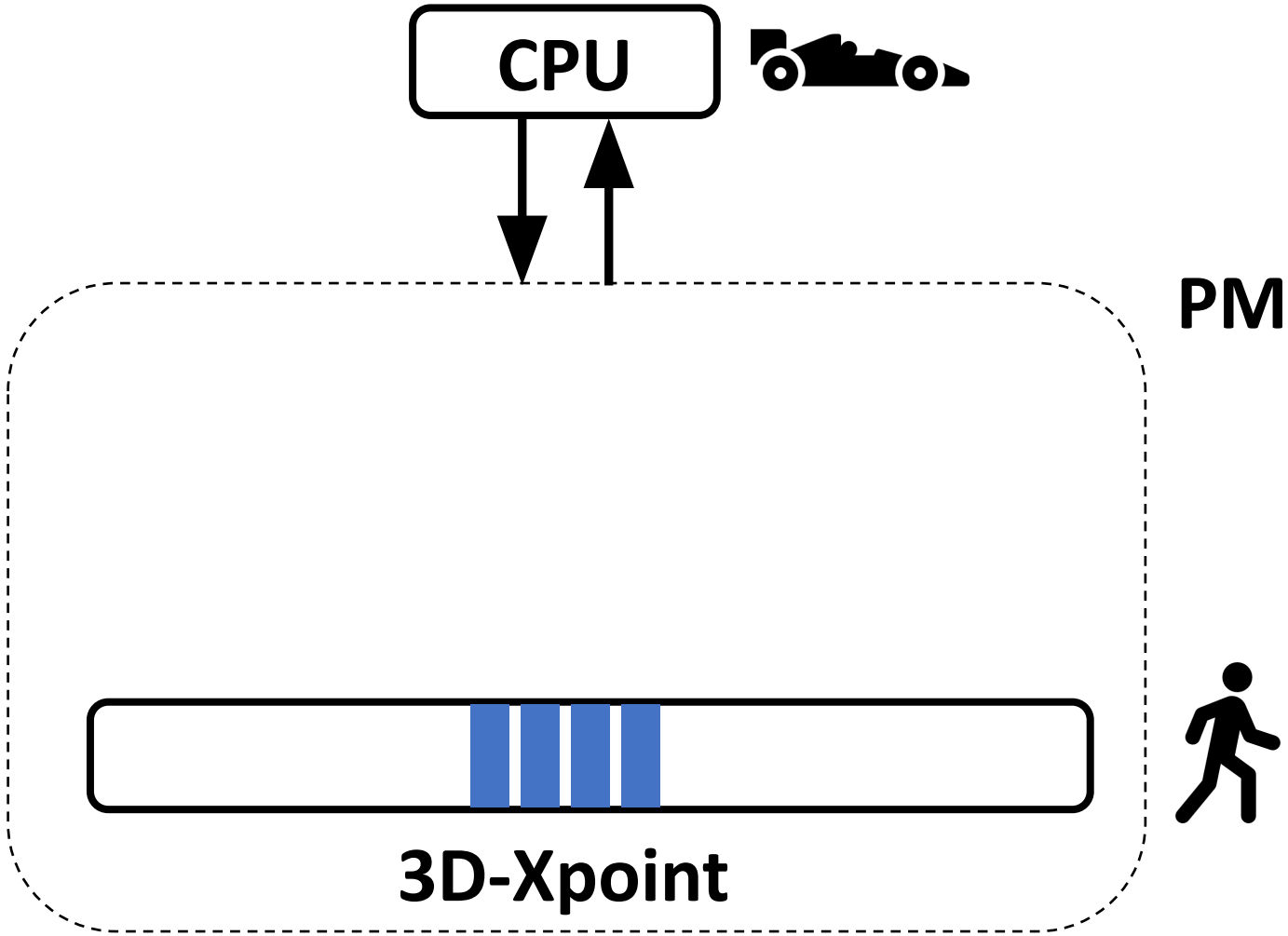
- High IO amplification measured
  - PM is slower than DRAM
    - **PM cache thrashing**

Primer on caching/prefetching in PM



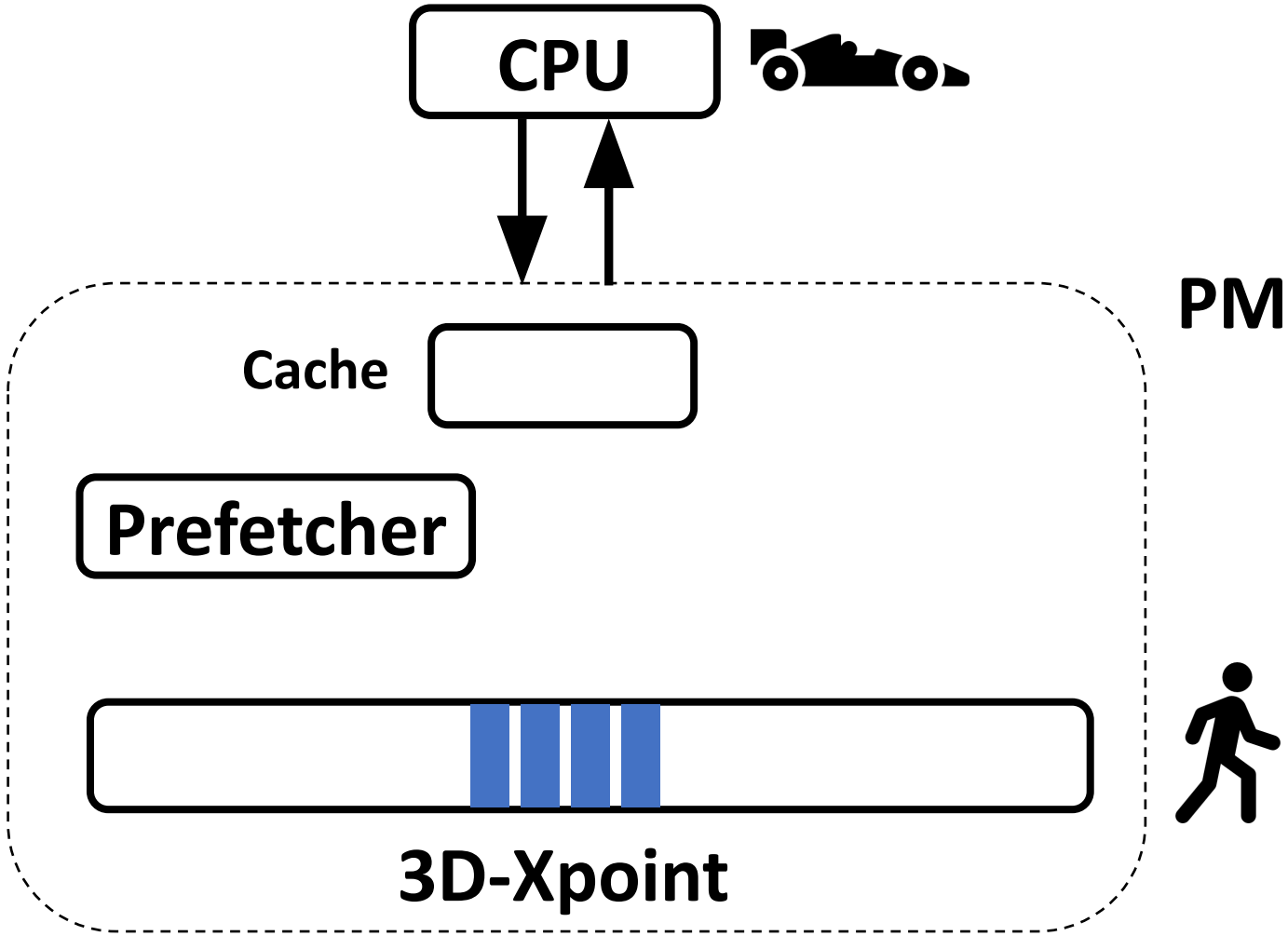
- PM is slower than DRAM
- Minimizes latency via caching and prefetching
- Cache thrashing more likely due to concurrent accesses

Primer on caching/prefetching in PM



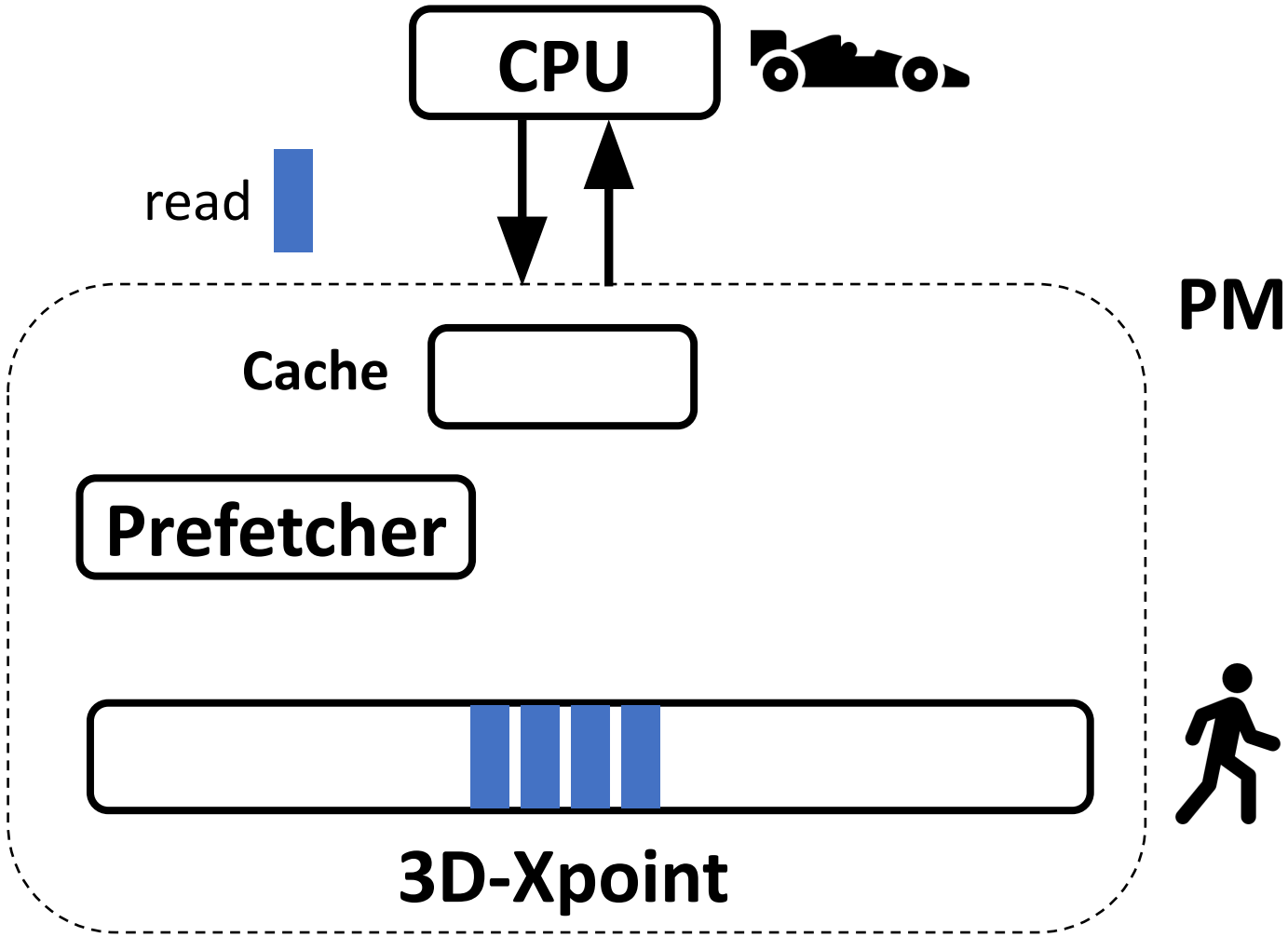
- PM is slower than DRAM
- Minimizes latency via caching and prefetching
- Cache thrashing more likely due to concurrent accesses

Primer on caching/prefetching in PM



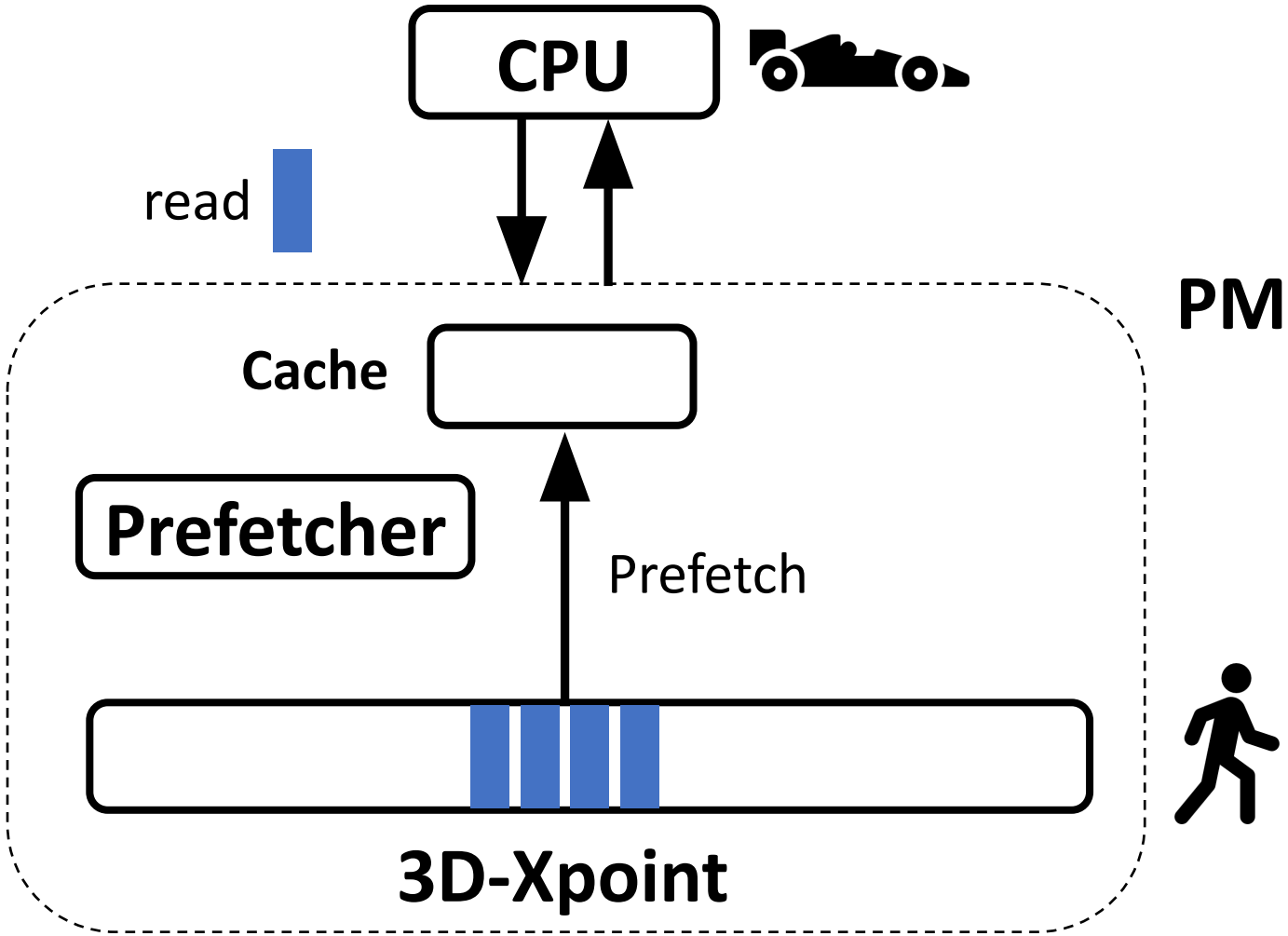
- PM is slower than DRAM
- Minimizes latency via caching and prefetching
- Cache thrashing more likely due to concurrent accesses

Primer on caching/prefetching in PM



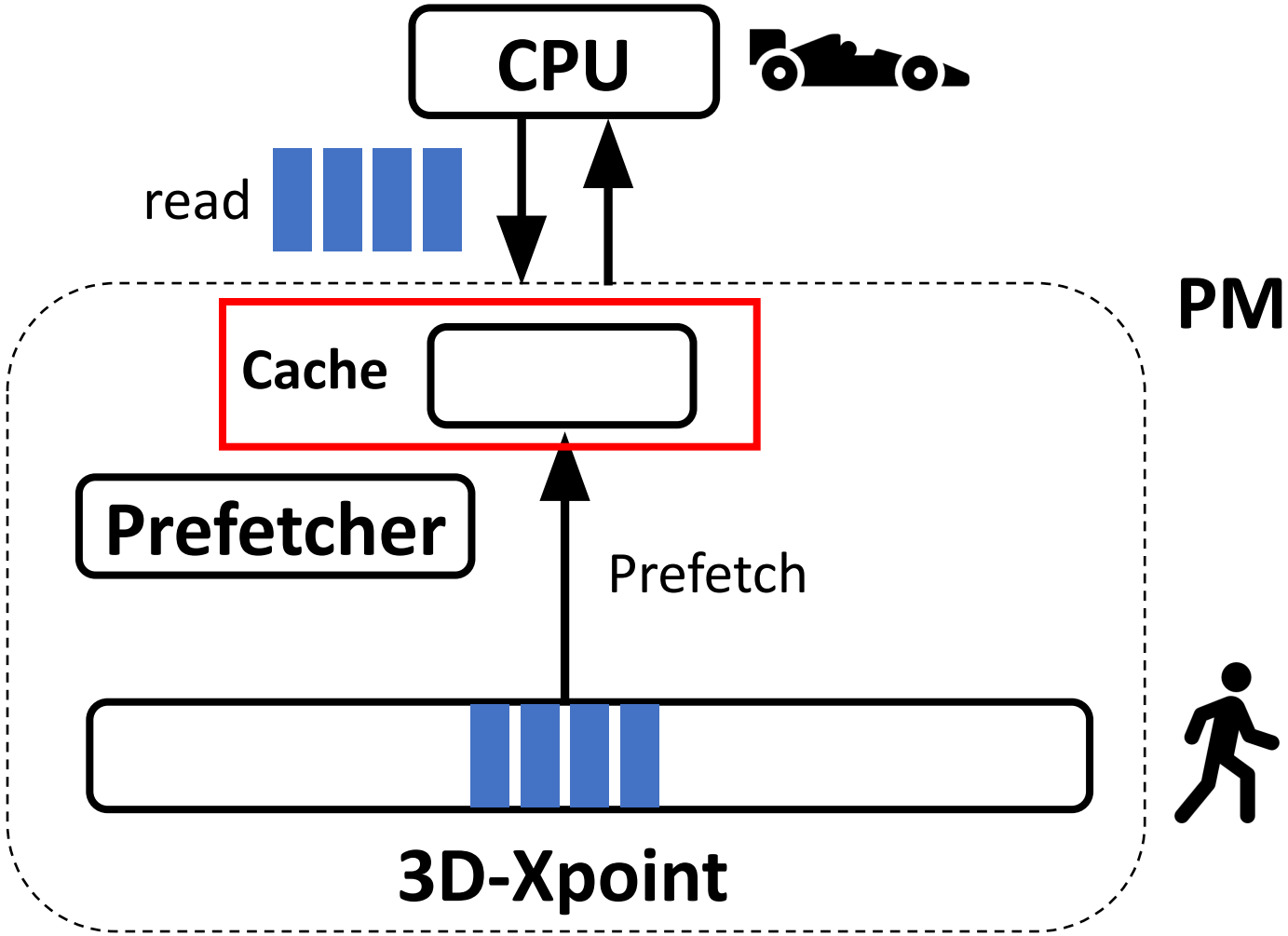
- PM is slower than DRAM
- Minimizes latency via caching and prefetching
- Cache thrashing more likely due to concurrent accesses

Primer on caching/prefetching in PM



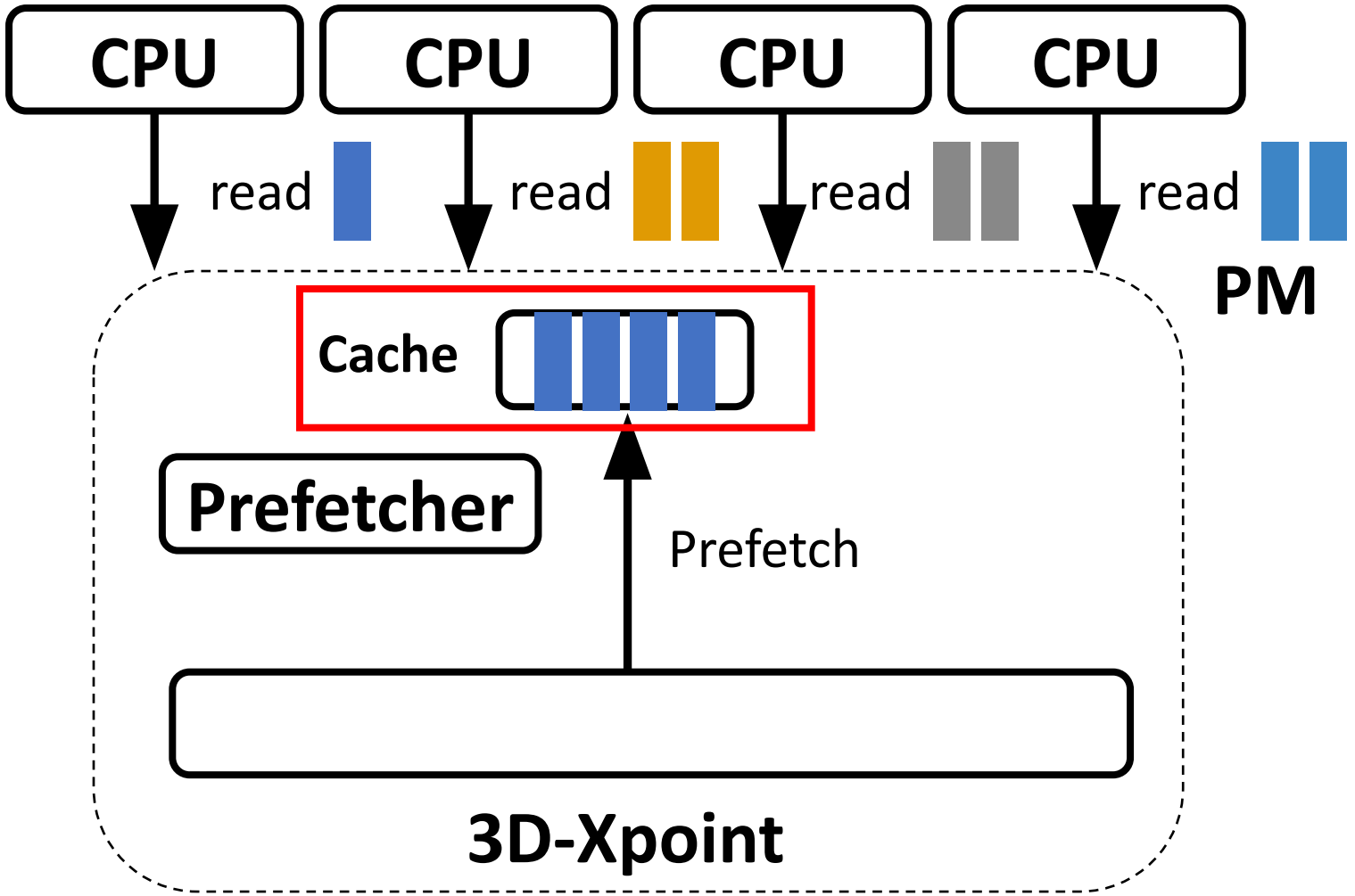
- PM is slower than DRAM
- Minimizes latency via caching and prefetching
- Cache thrashing more likely due to concurrent accesses

Primer on caching/prefetching in PM



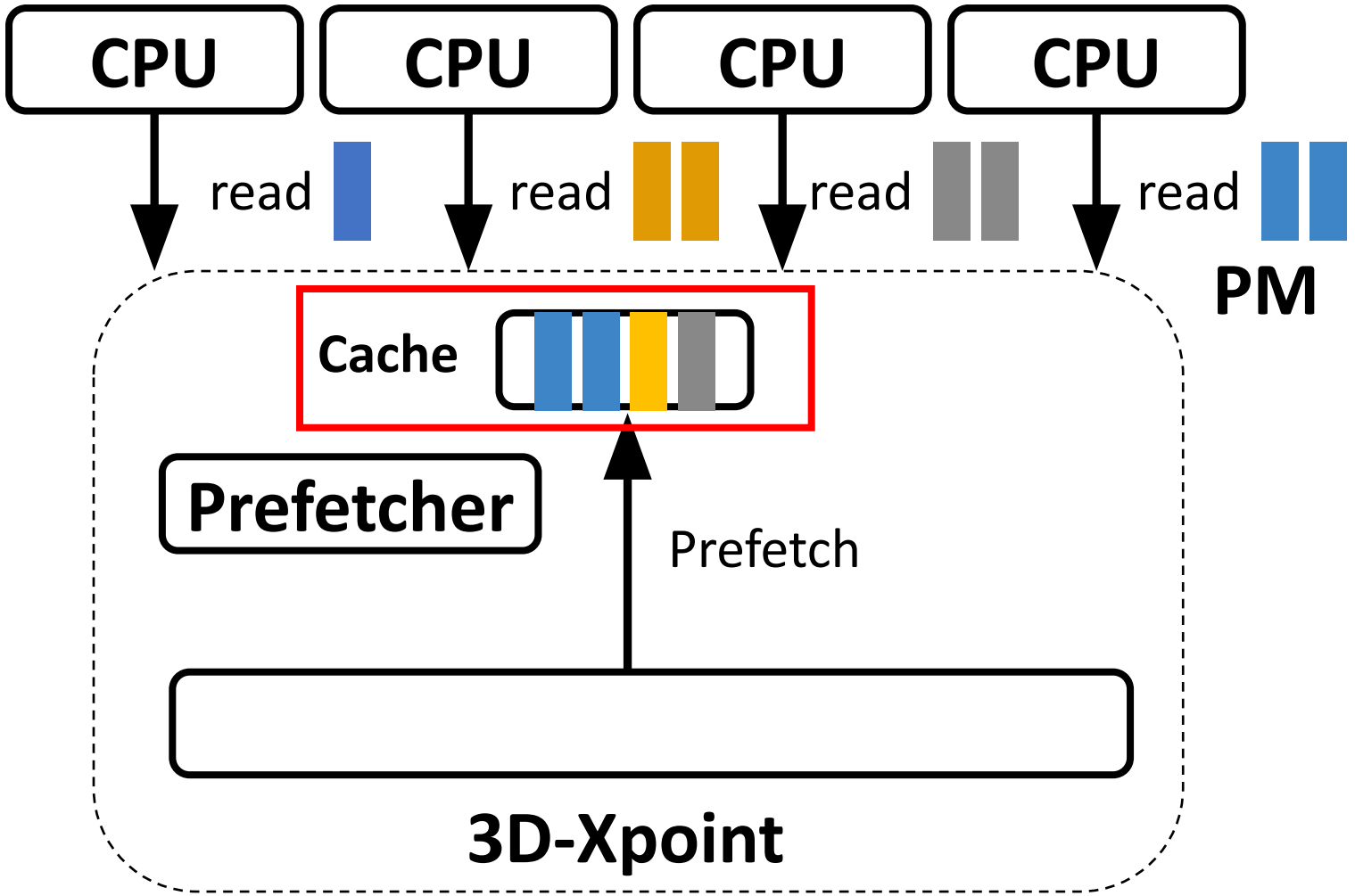
- PM is slower than DRAM
- Minimizes latency via caching and prefetching
- Cache thrashing more likely due to concurrent accesses

Conc. access → Inefficient caching/prefetching



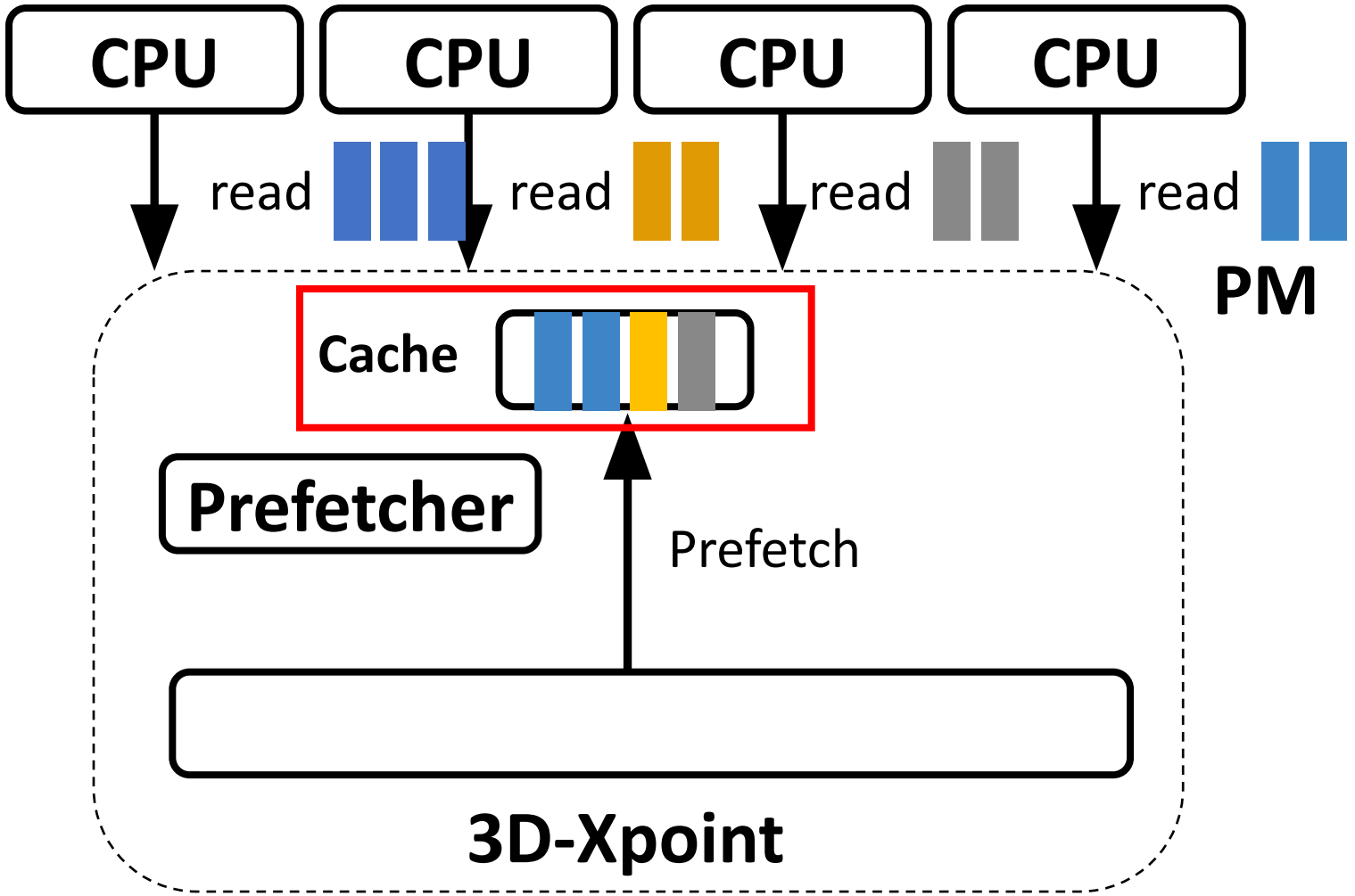
- PM is slower than DRAM
- Minimizes latency via caching and prefetching
- Cache thrashing more likely due to concurrent accesses

Conc. access → Inefficient caching/prefetching



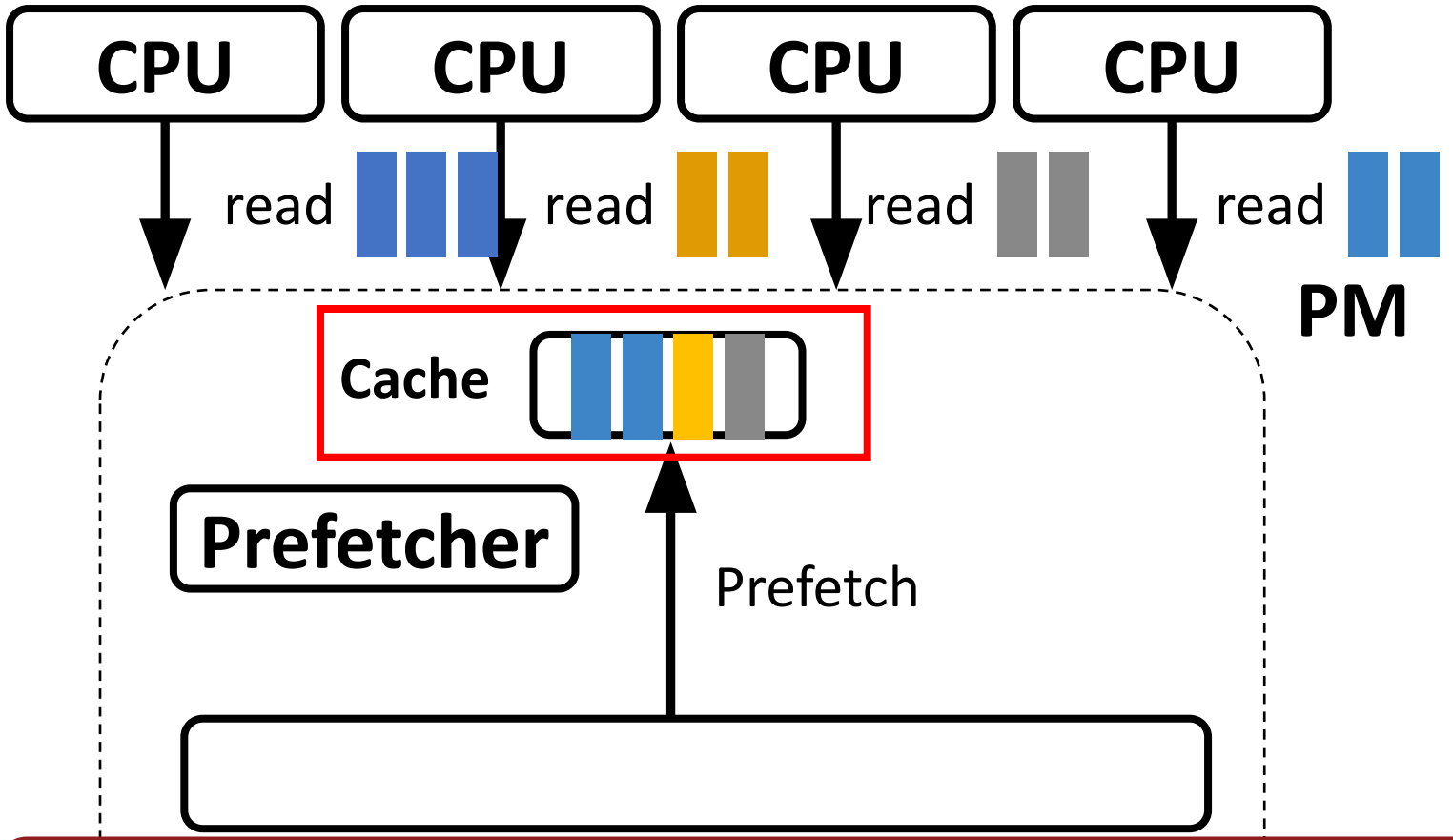
- PM is slower than DRAM
- Minimizes latency via caching and prefetching
- Cache thrashing more likely due to concurrent accesses

Conc. access → Inefficient caching/prefetching



- PM is slower than DRAM
- Minimizes latency via caching and prefetching
- Cache thrashing more likely due to concurrent accesses

Conc. access → Inefficient caching/prefetching



- PM is slower than DRAM
- Minimizes latency via caching and prefetching
- Cache thrashing more likely due to concurrent accesses

PM cannot handle arbitrary concurrent accesses

Approach …

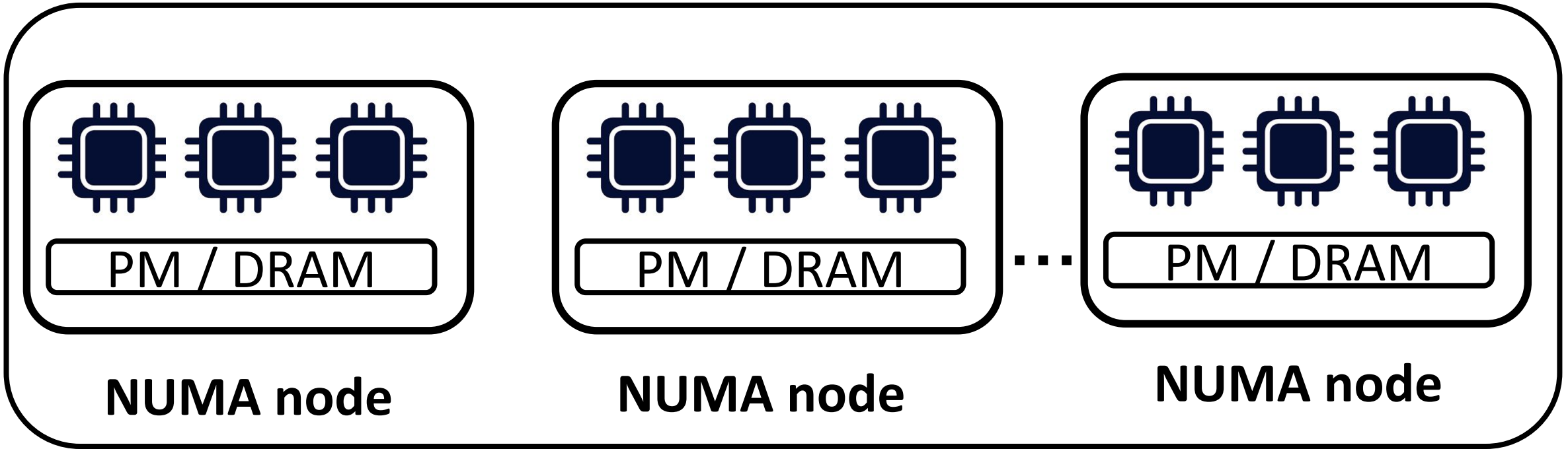
Finding: PM cannot handle arbitrary concurrent accesses

Arbitrate the number of threads accessing PM

Hardware setup: consider multiple PM nodes

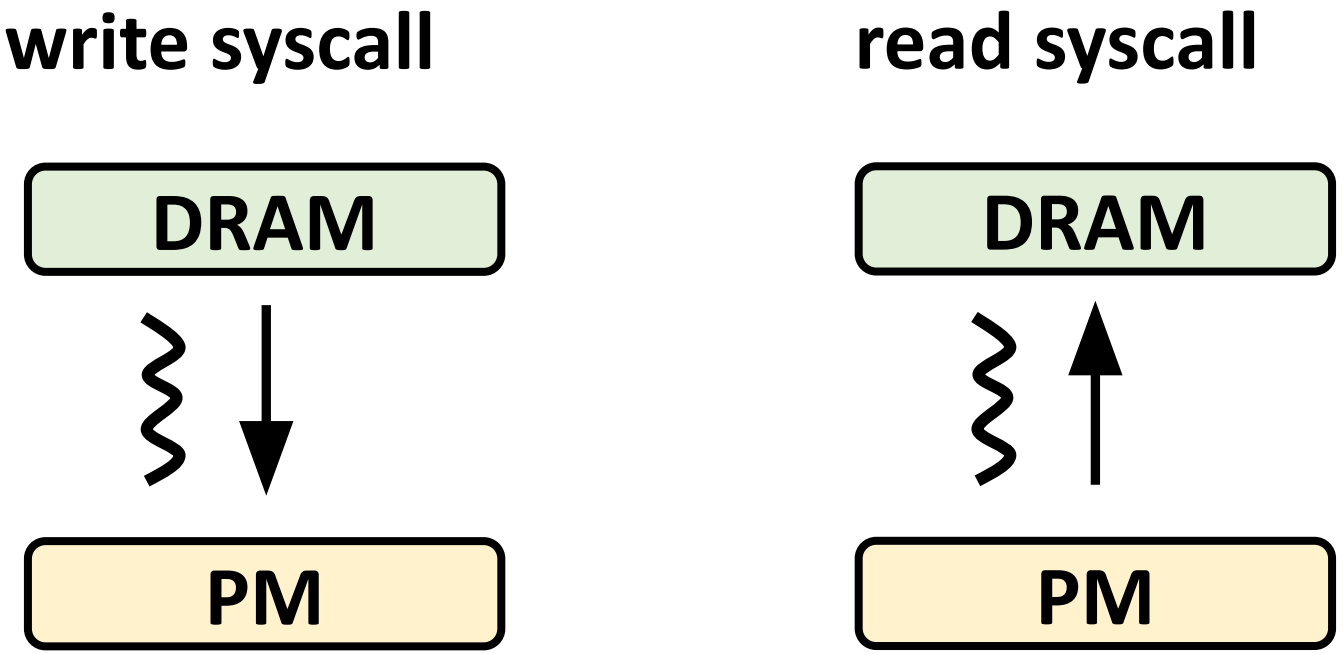
PM DIMMs are attached to multiple different NUMA nodes

NUMA: Accessing the local socket memory is faster than the remote socket



IO operations in a PM file system

Data transfer from/to the buffer (in DRAM) to/from the storage media (PM)

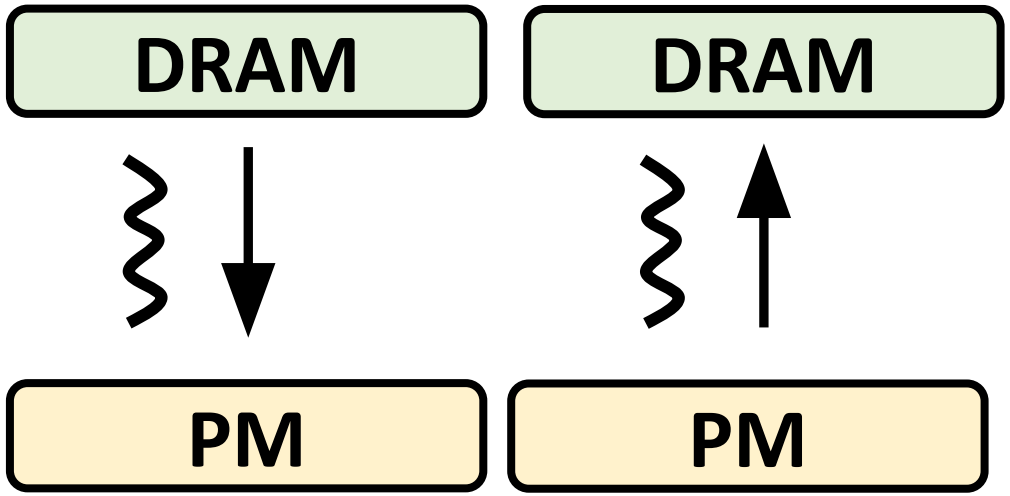


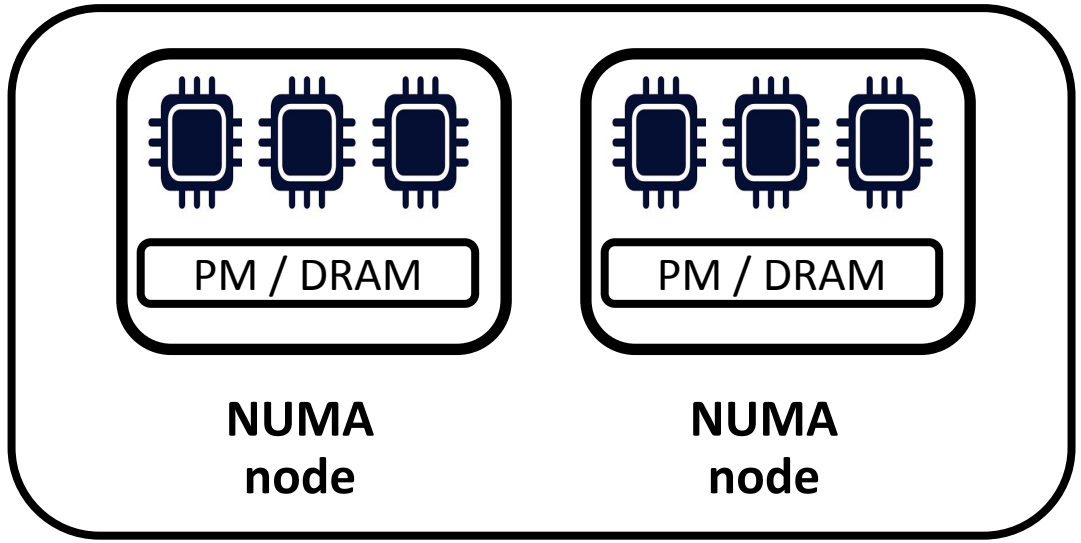
thread

IO ops in a PM file system in NUMA setup

Data transfer from/to the buffer (in DRAM) to/from the storage media (PM)

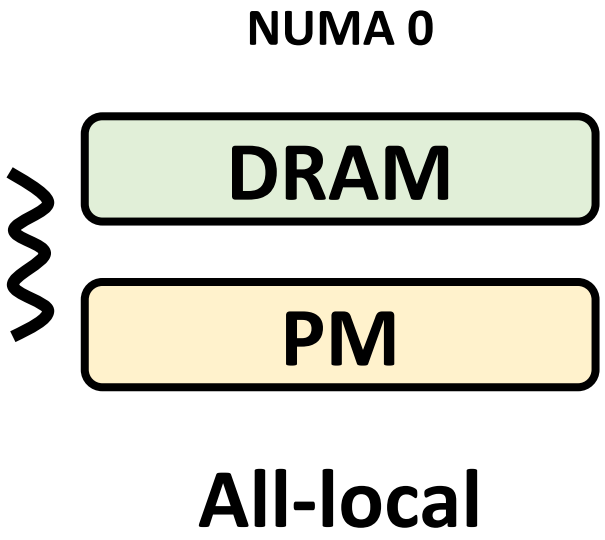
write syscall **read syscall**





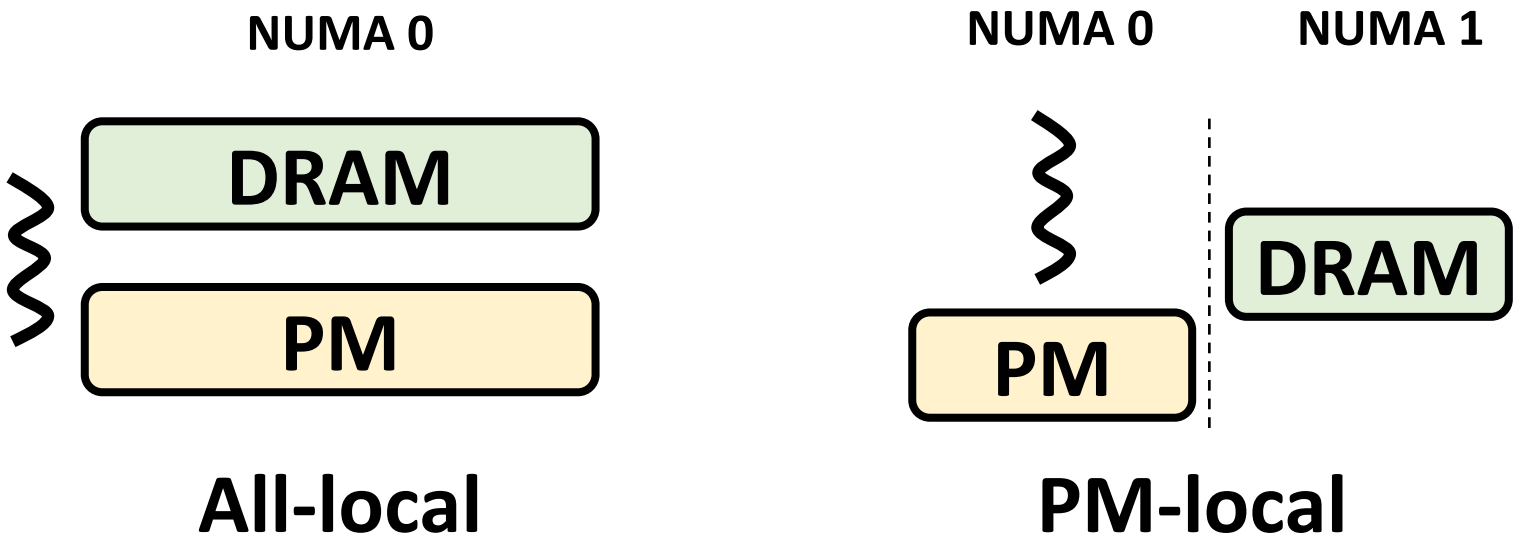
Q. Impact of NUMA placement of DRAM, PM, and access thread on application performance?

PM IO operation in a NUMA setup



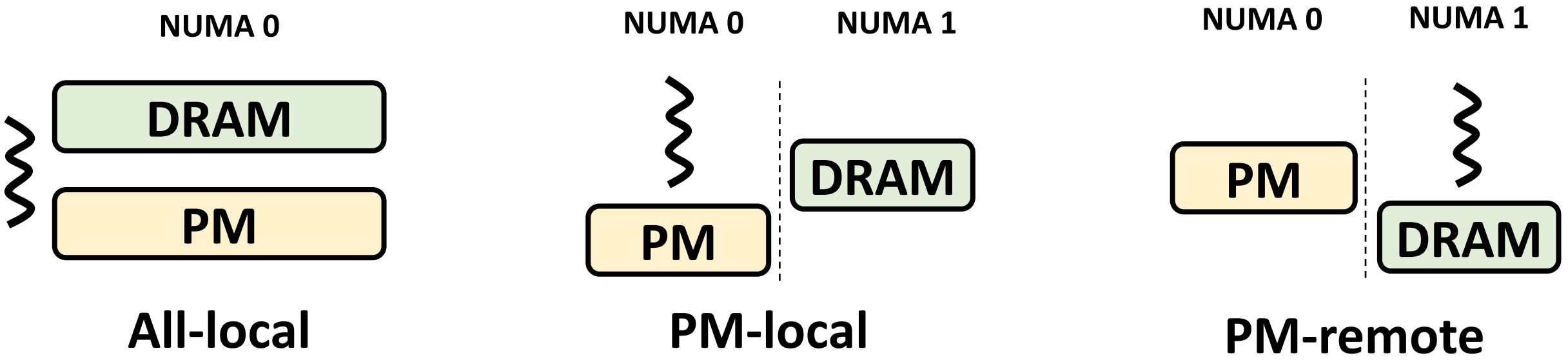
thread

PM IO operation in a NUMA setup



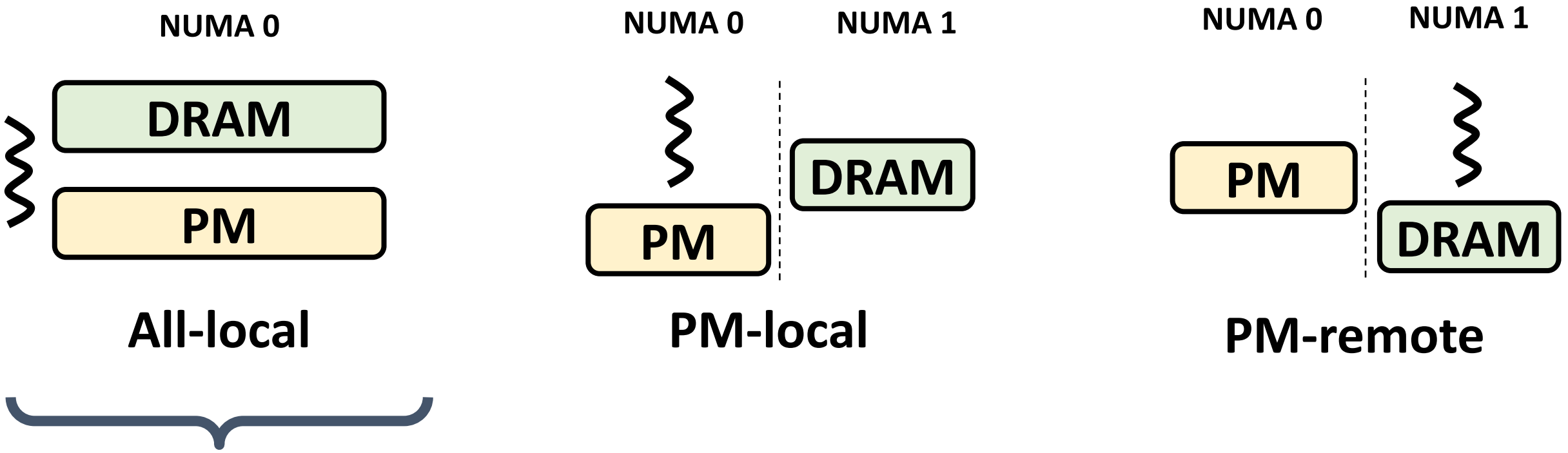
thread

PM IO operation in a NUMA setup



thread

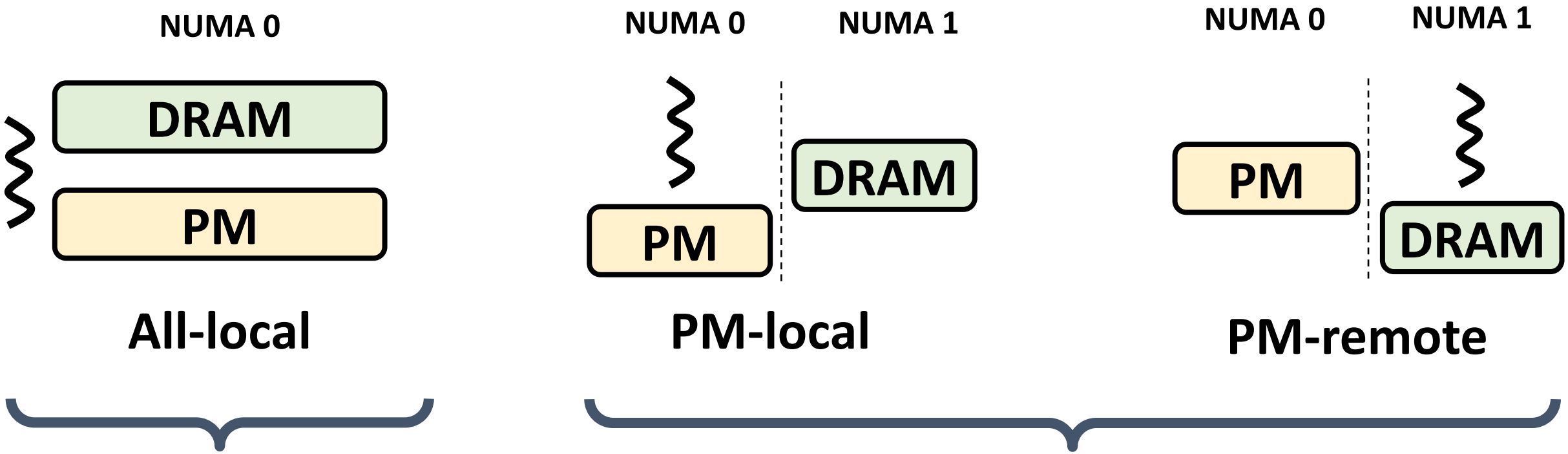
PM IO operation in a NUMA setup



Best case scenario

thread

PM IO operation in a NUMA setup



Best case scenario

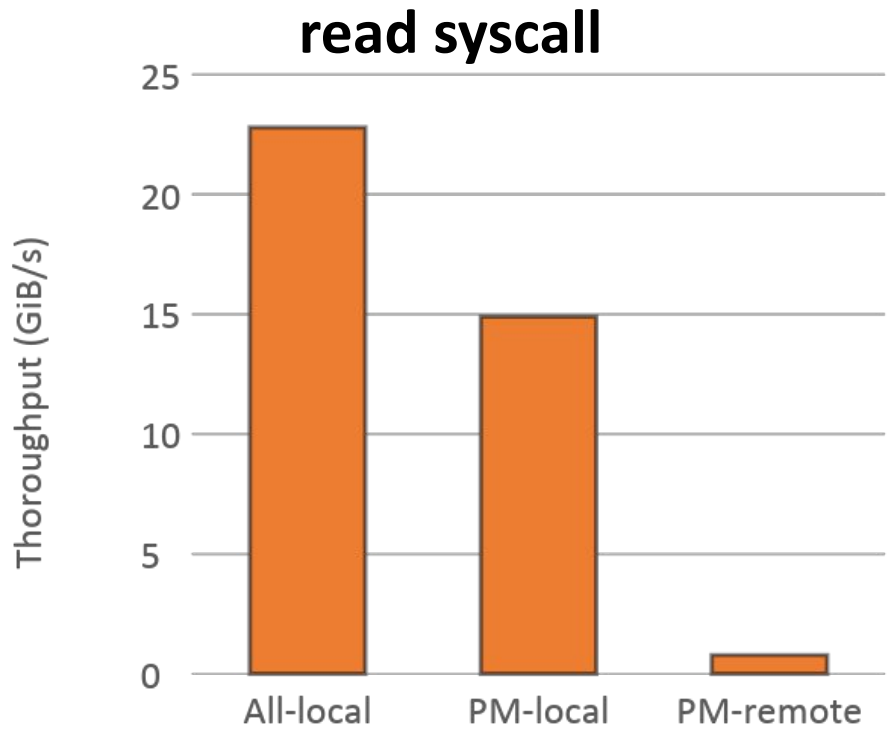
Copy data b/w PM (NUMA 0) & DRAM (NUMA 1)

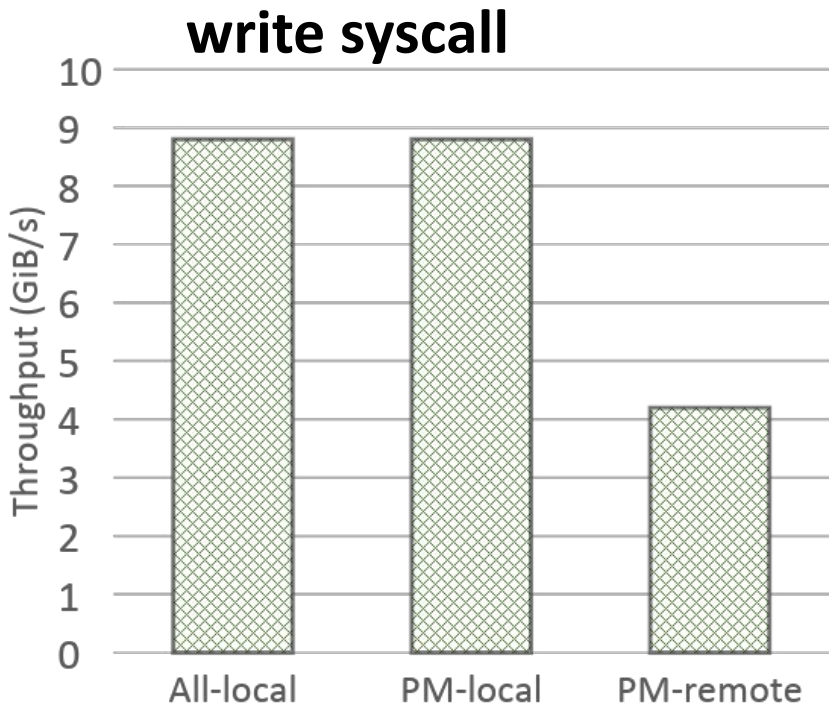
Mostly common but not the best case scenario

thread

NUMA impact: PM >> DRAM remote access

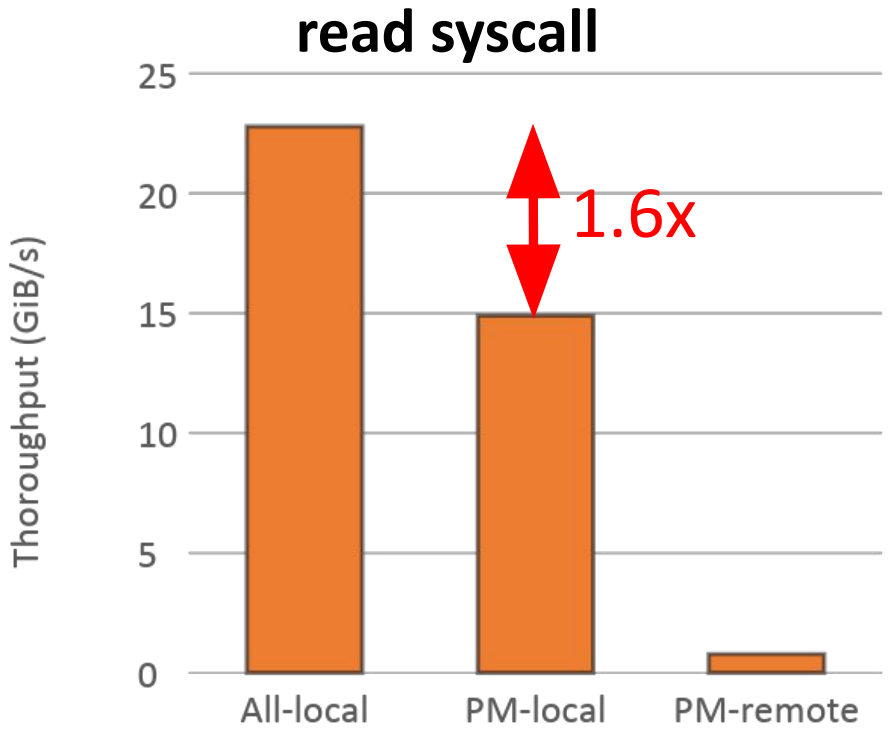
Benchmark: Single thread reads/writes 2MB data privately in a 1GB file

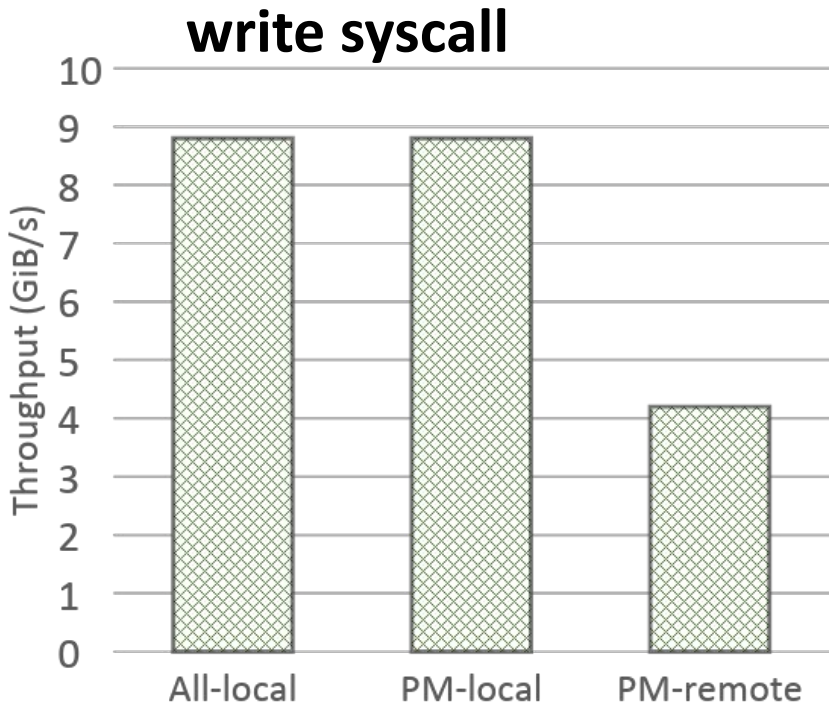




NUMA impact: PM >> DRAM remote access

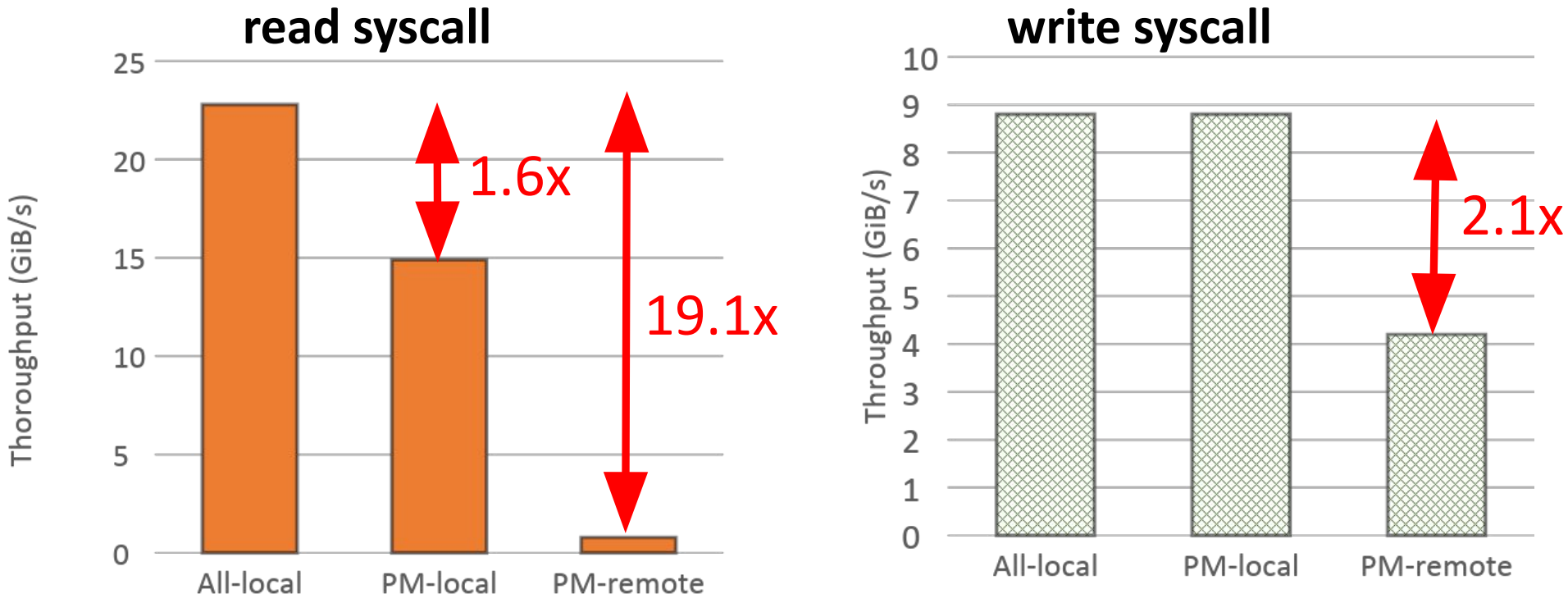
Benchmark: Single thread reads/writes 2MB data privately in a 1GB file





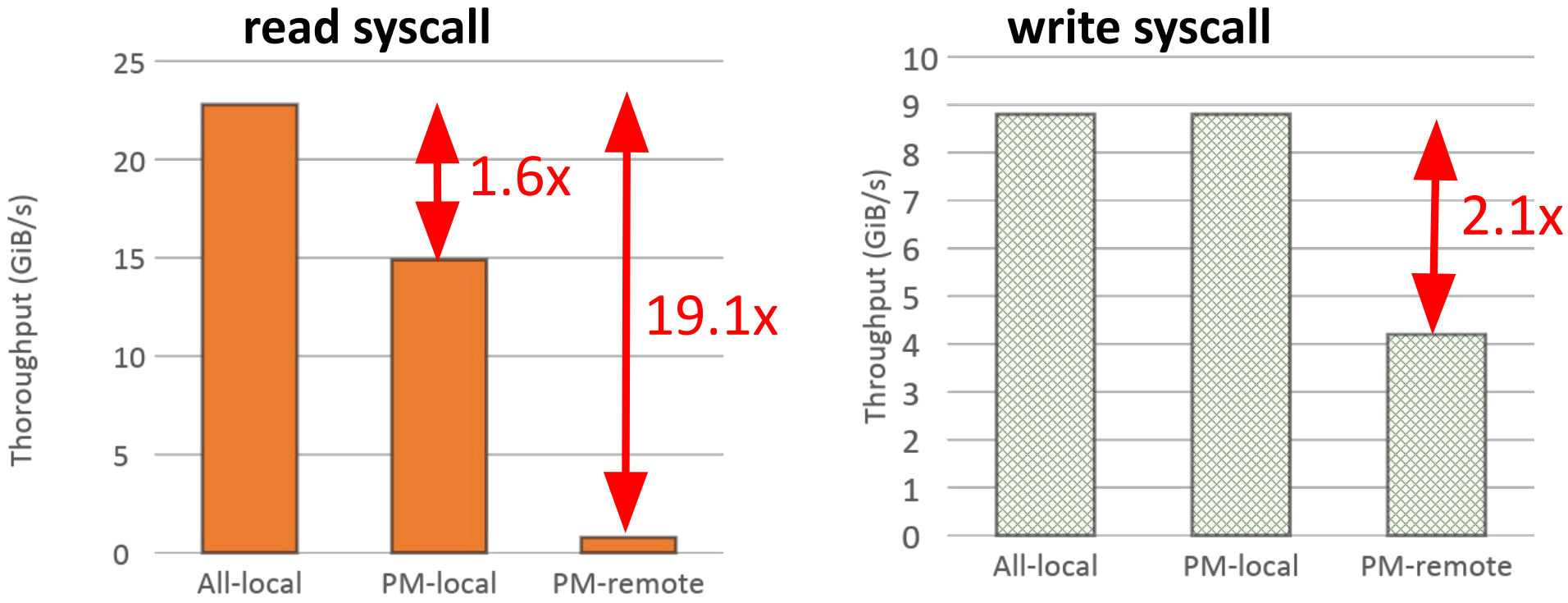
NUMA impact: PM >> DRAM remote access

Benchmark: Single thread reads/writes 2MB data privately in a 1GB file



NUMA impact: PM >> DRAM remote access

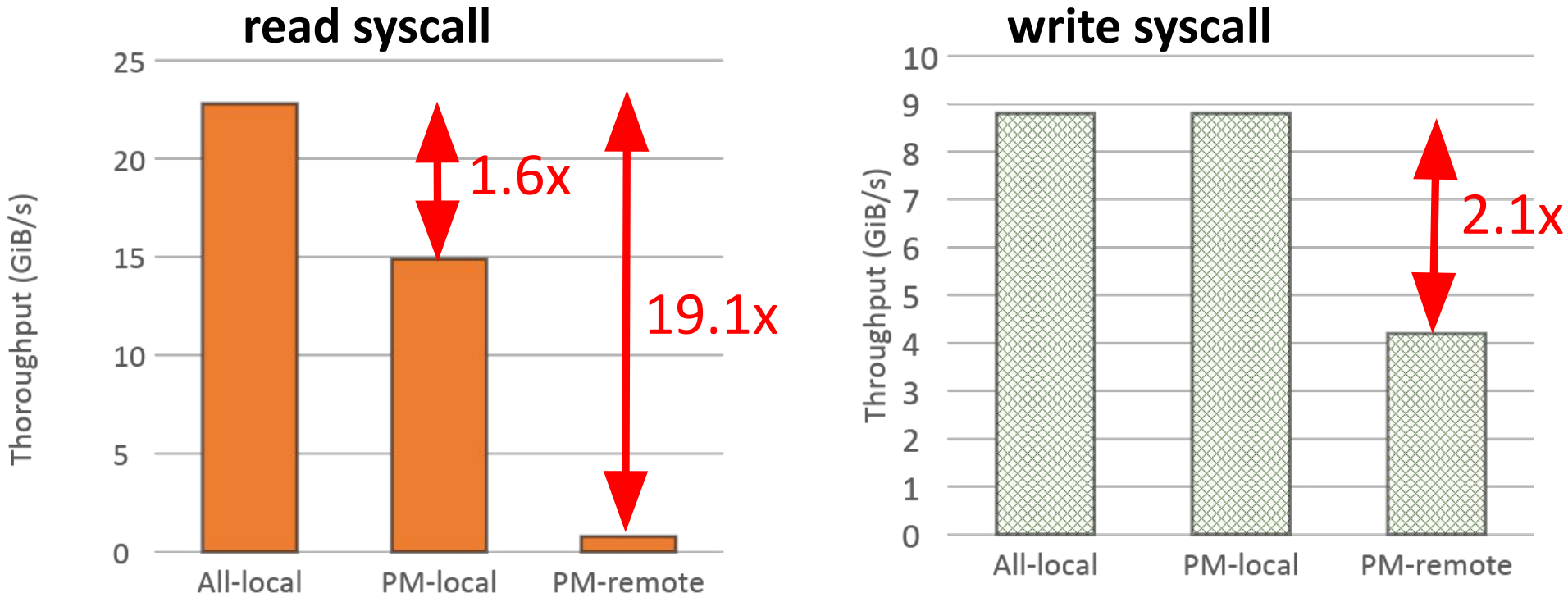
Benchmark: Single thread reads/writes 2MB data privately in a 1GB file



Impact of remote PM is more severe than remote DRAM access

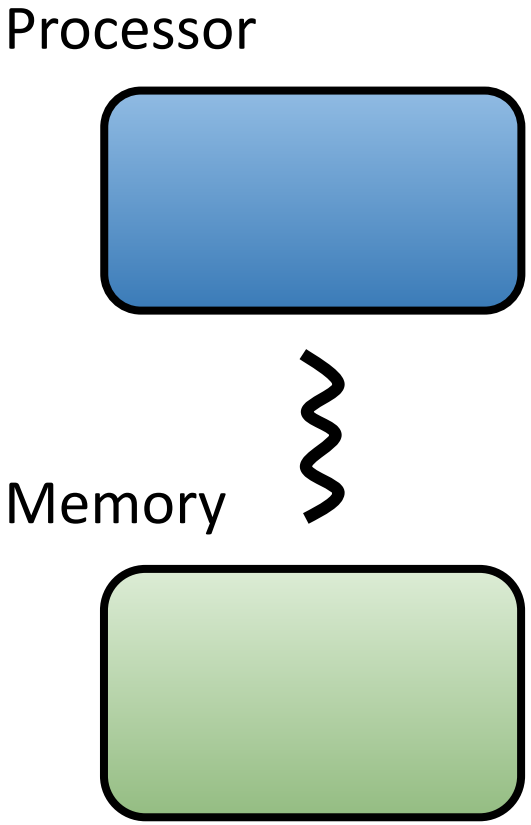
NUMA impact: PM >> DRAM remote access

Benchmark: Single thread reads/writes 2MB data privately in a 1GB file

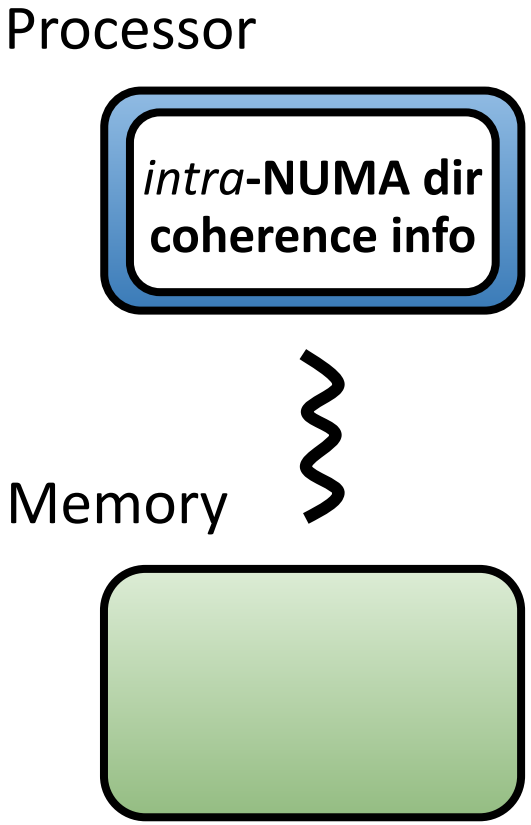


Cache coherence impedes NUMA scalability

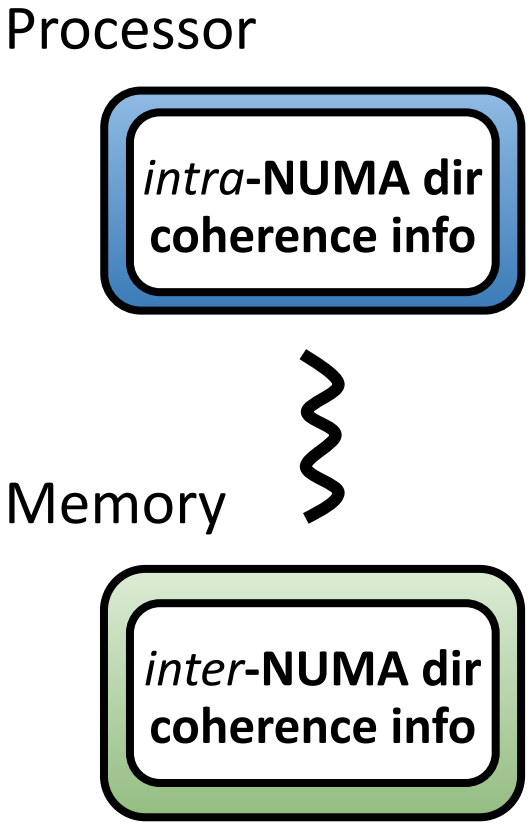
NUMA PM impedes performance



NUMA PM impedes performance

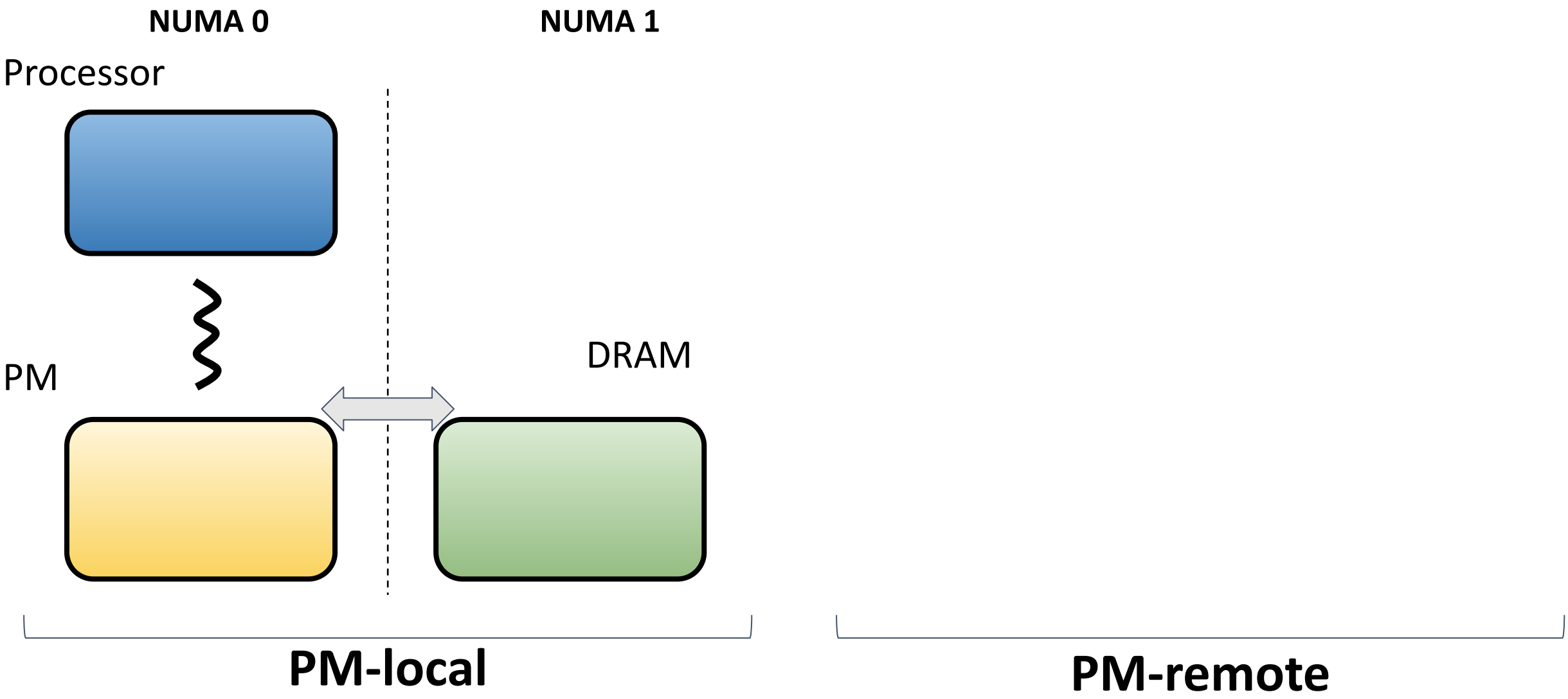


NUMA PM impedes performance

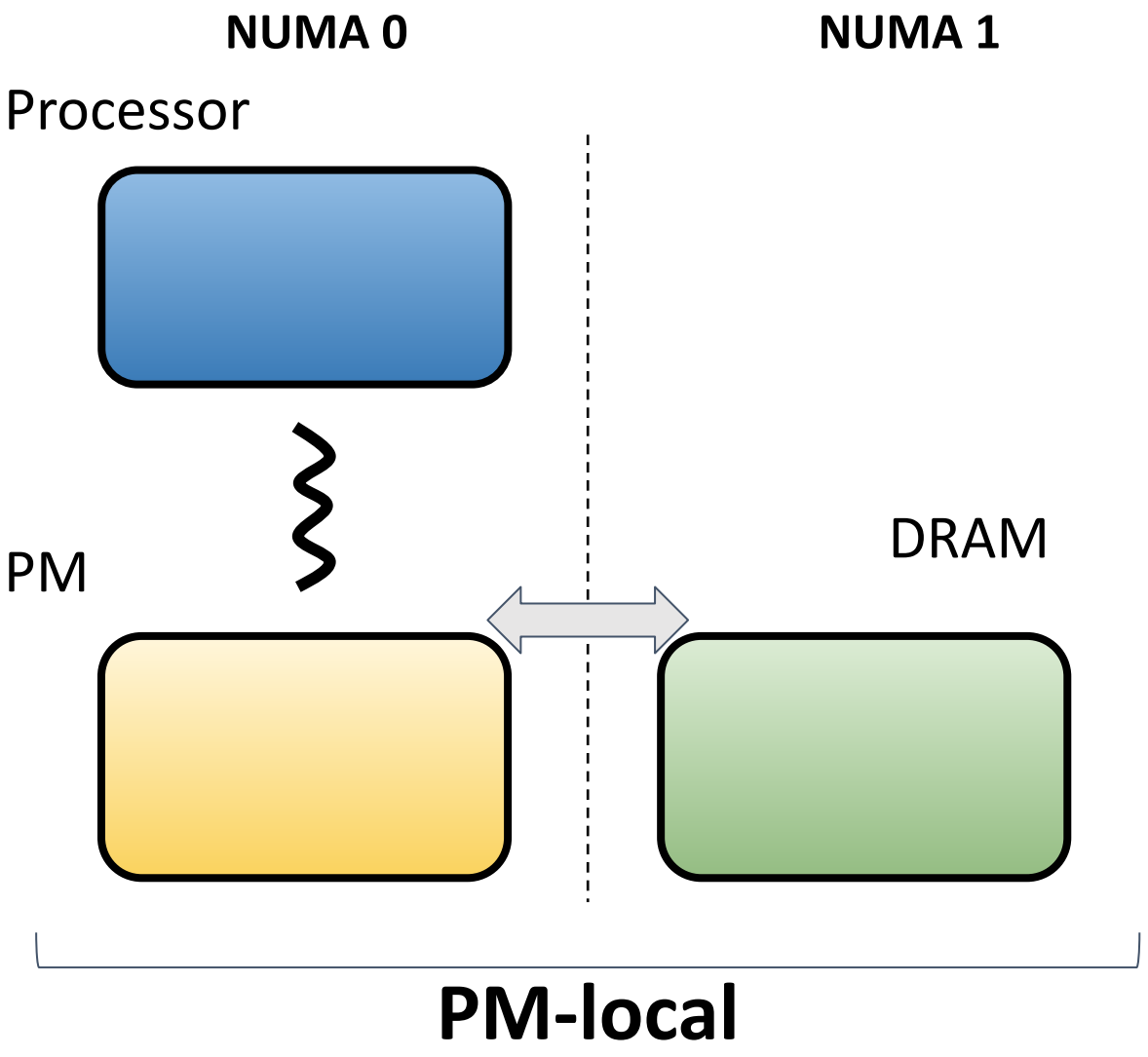


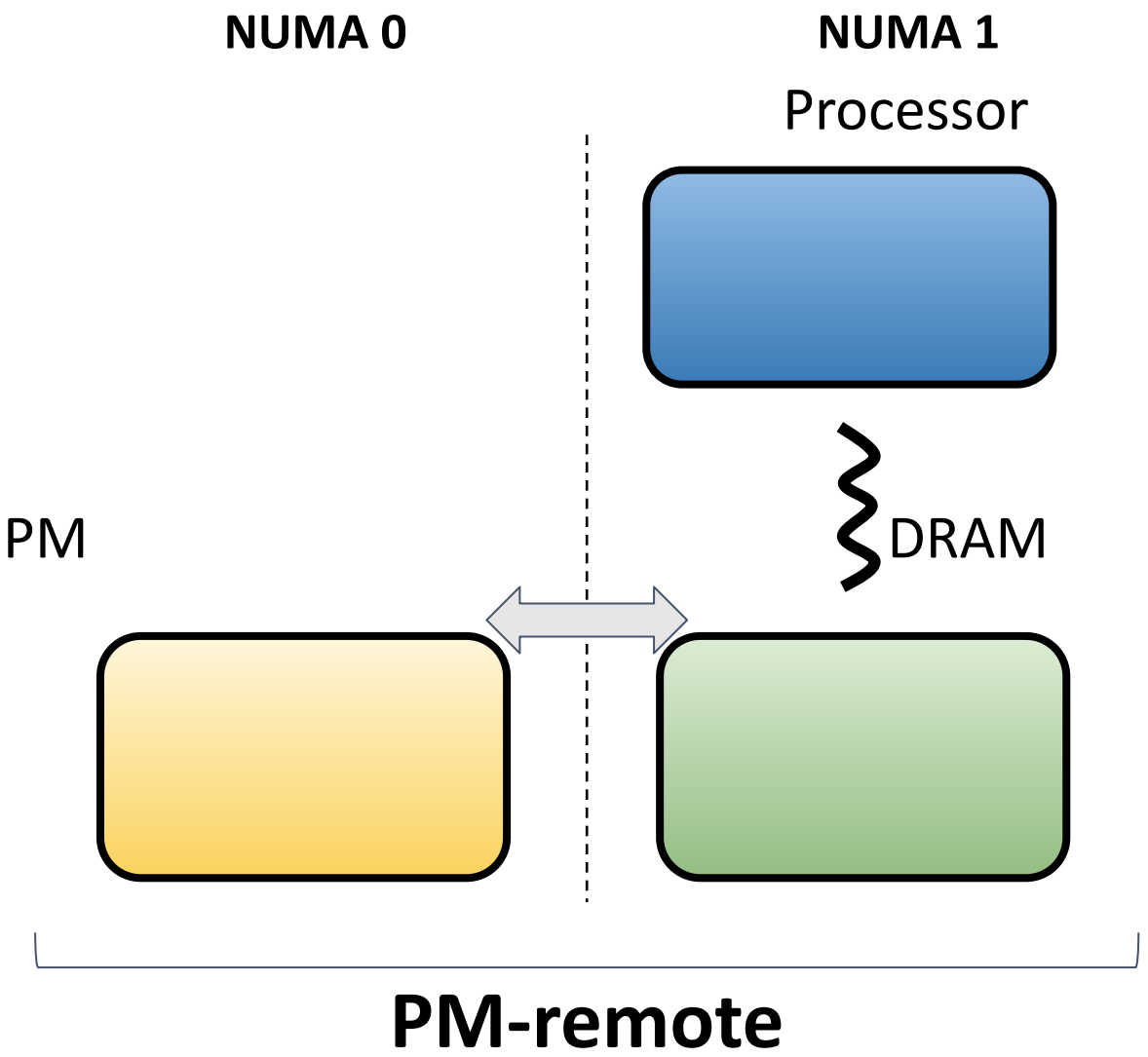
NUMA PM impedes performance

NUMA PM impedes performance

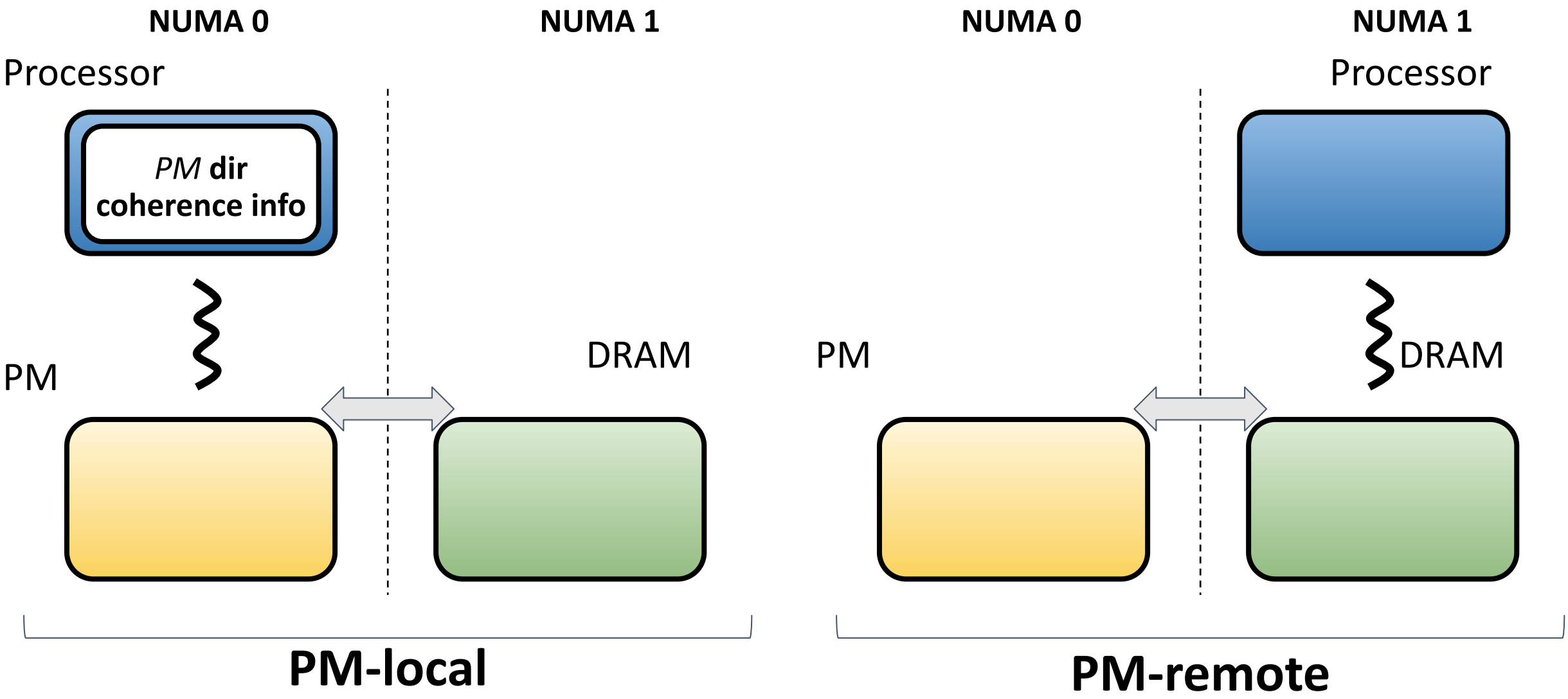


NUMA PM impedes performance

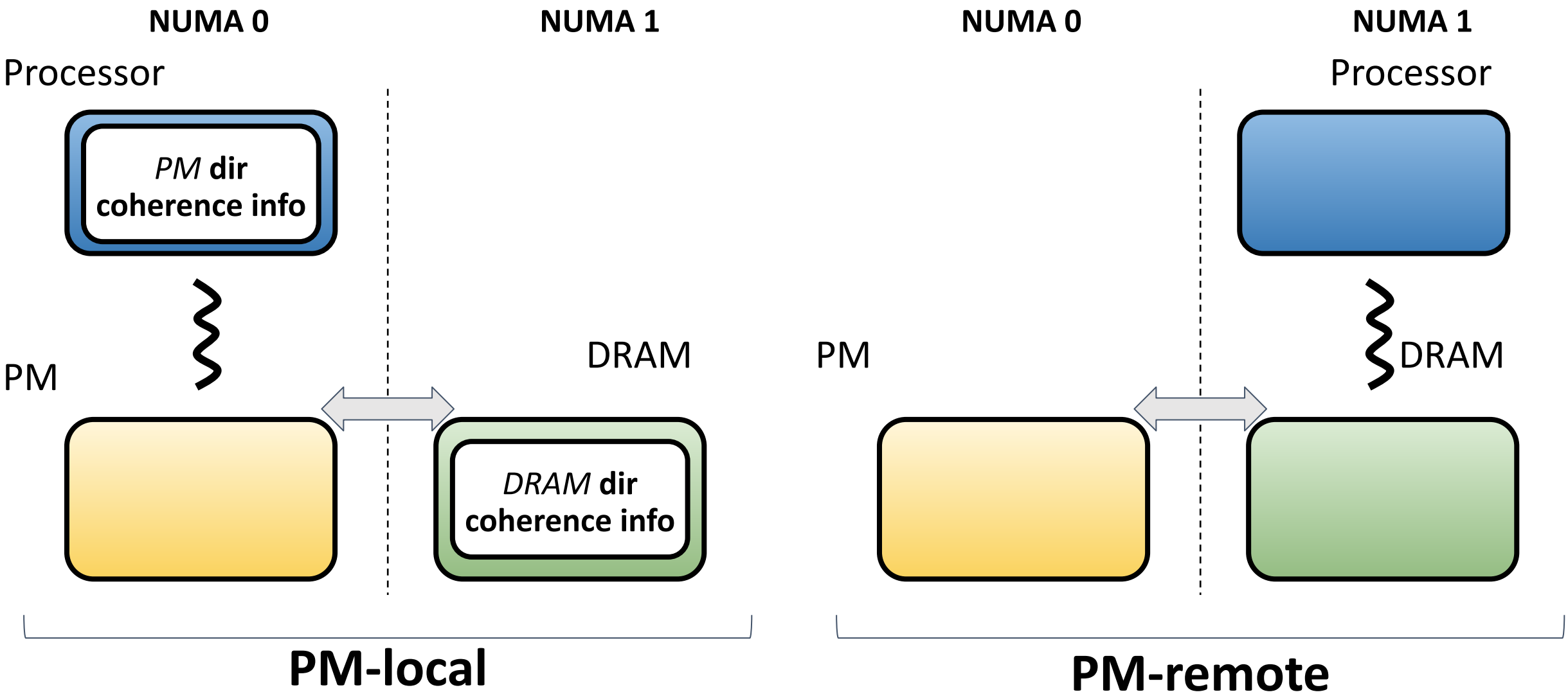




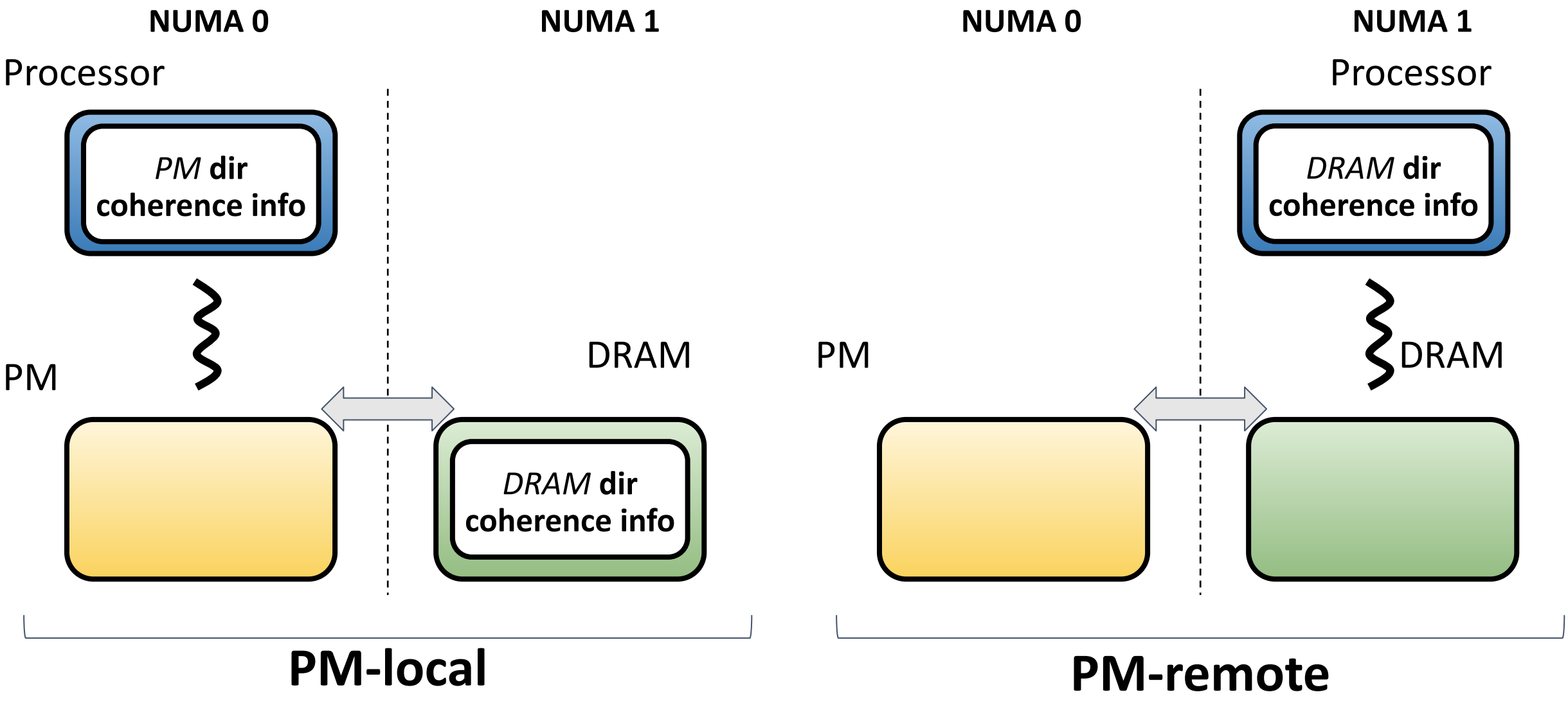
NUMA PM impedes performance



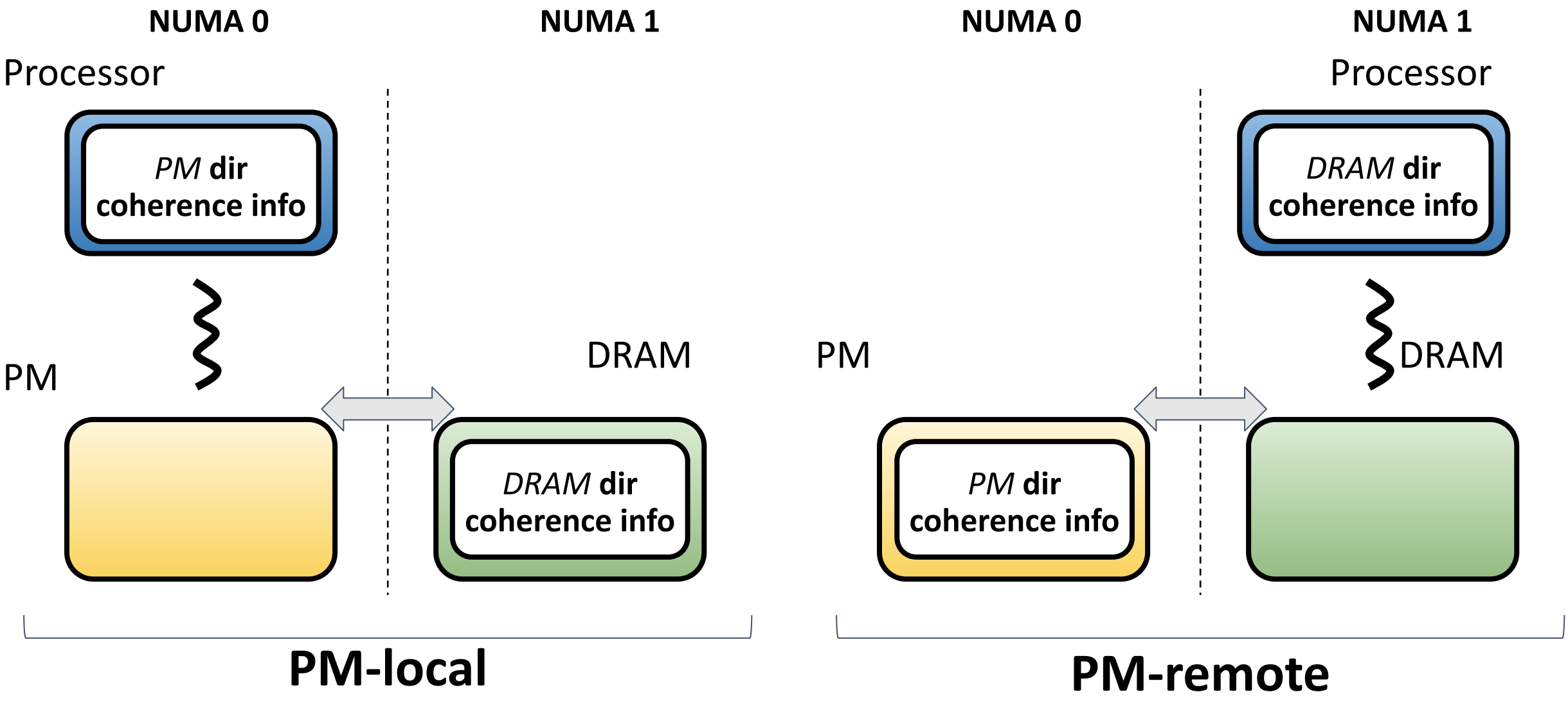
NUMA PM impedes performance



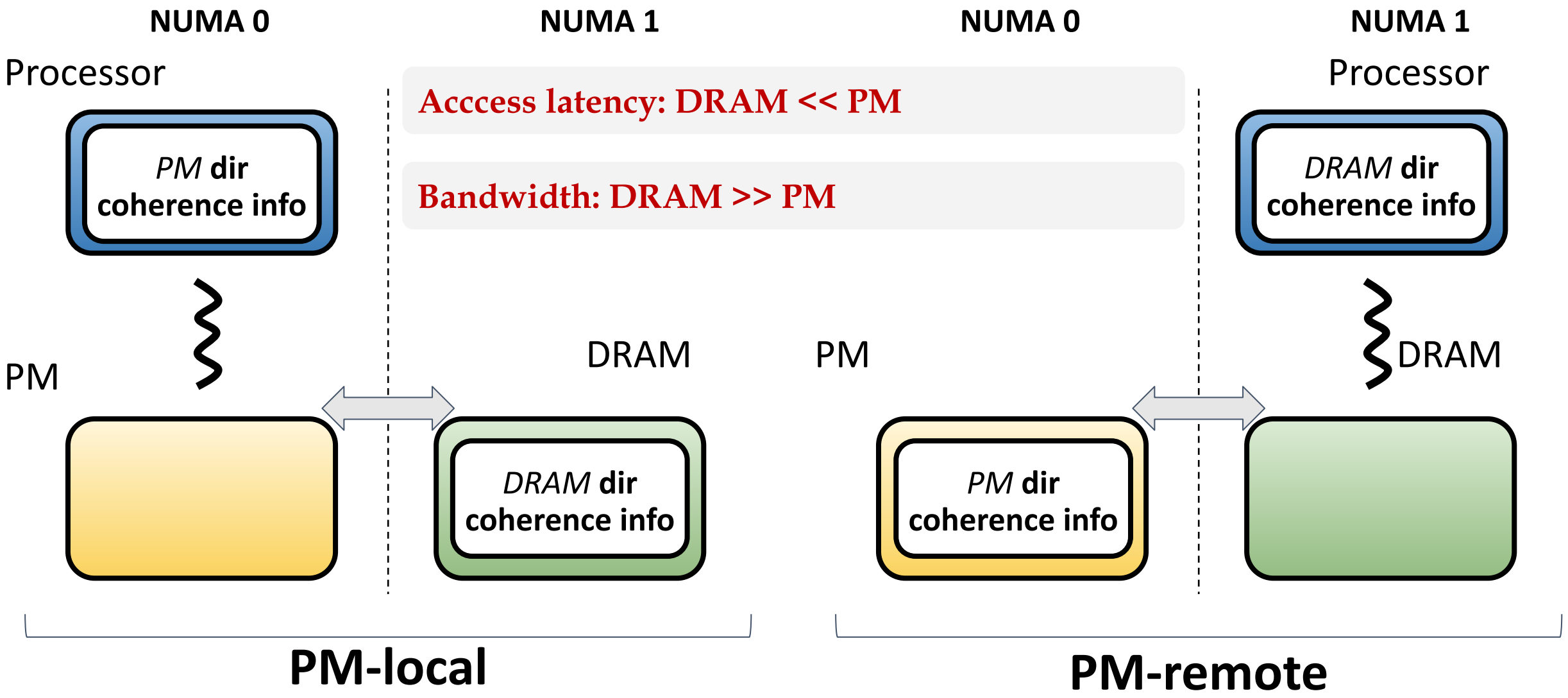
NUMA PM impedes performance



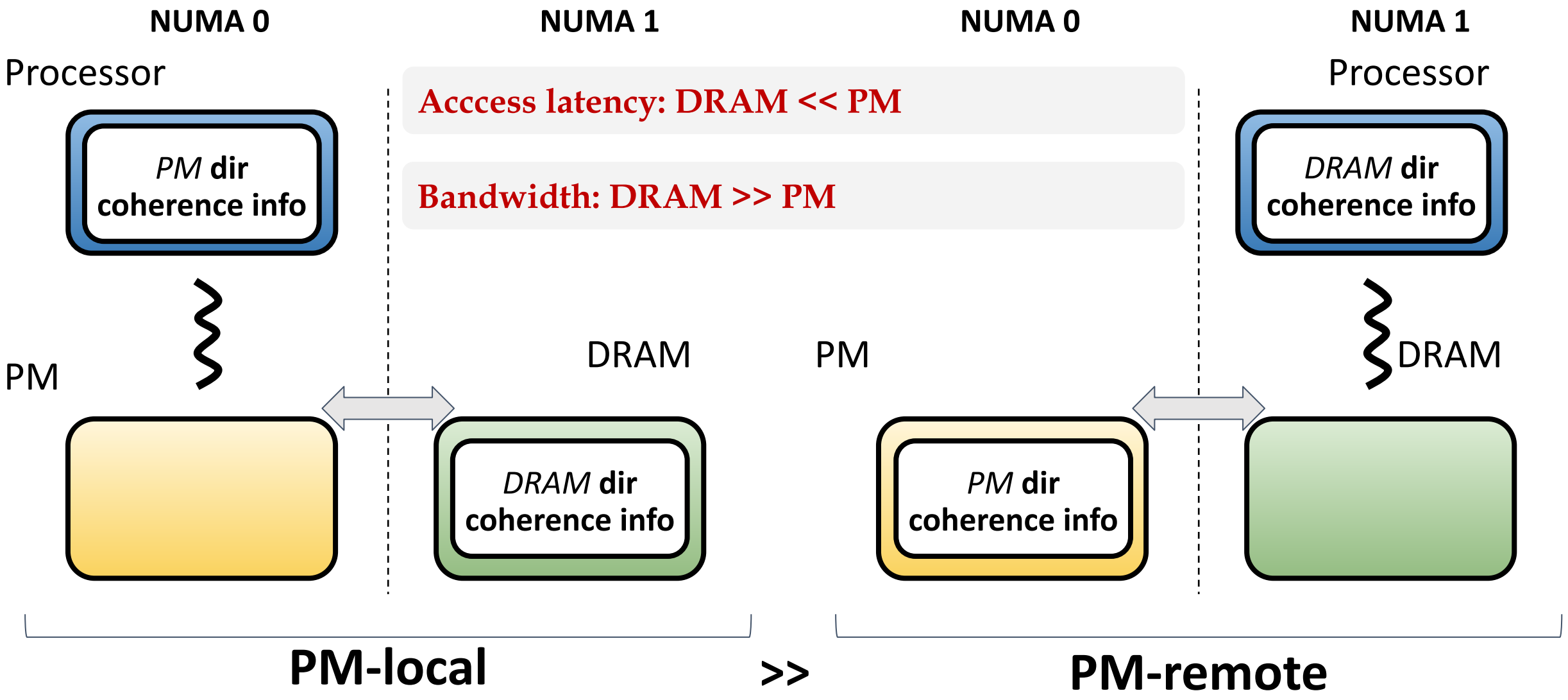
NUMA PM impedes performance



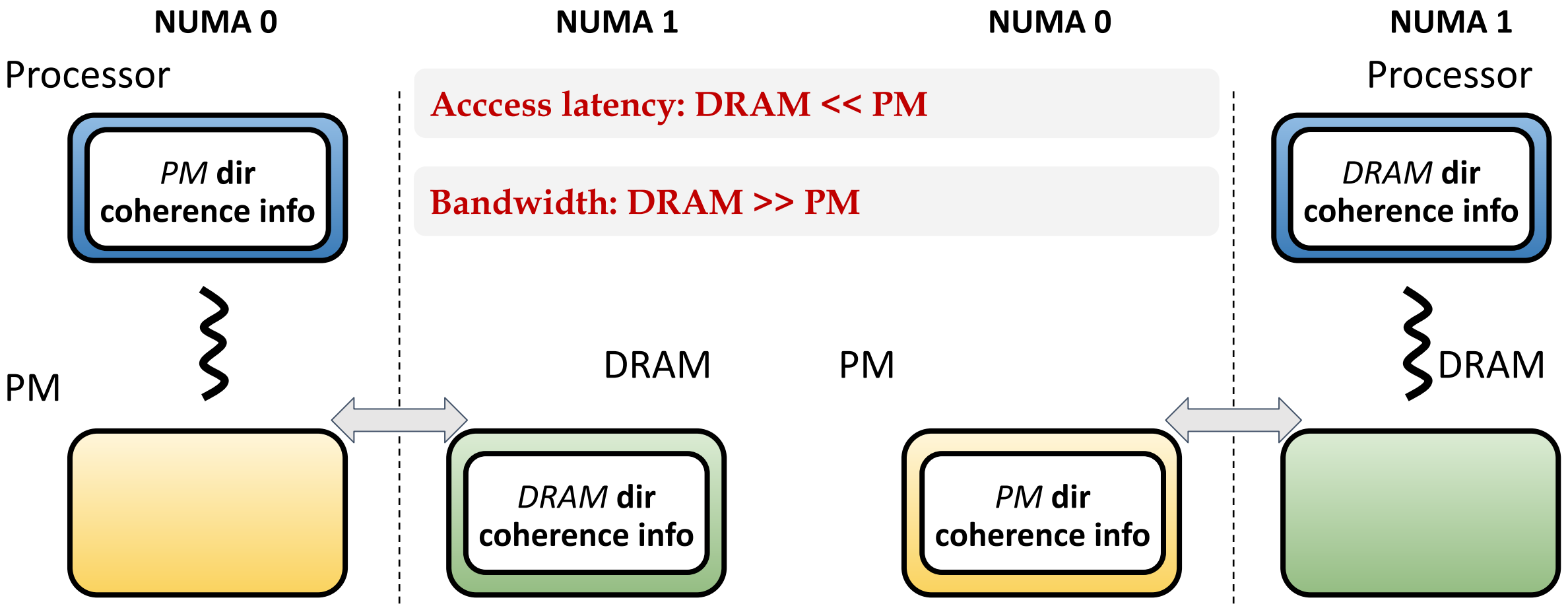
NUMA PM impedes performance



NUMA PM impedes performance



NUMA PM impedes performance



Remote PM access degrades severely

Approach …

Finding: PM cannot handle arbitrary concurrent accesses

Arbitrate the number of threads accessing PM

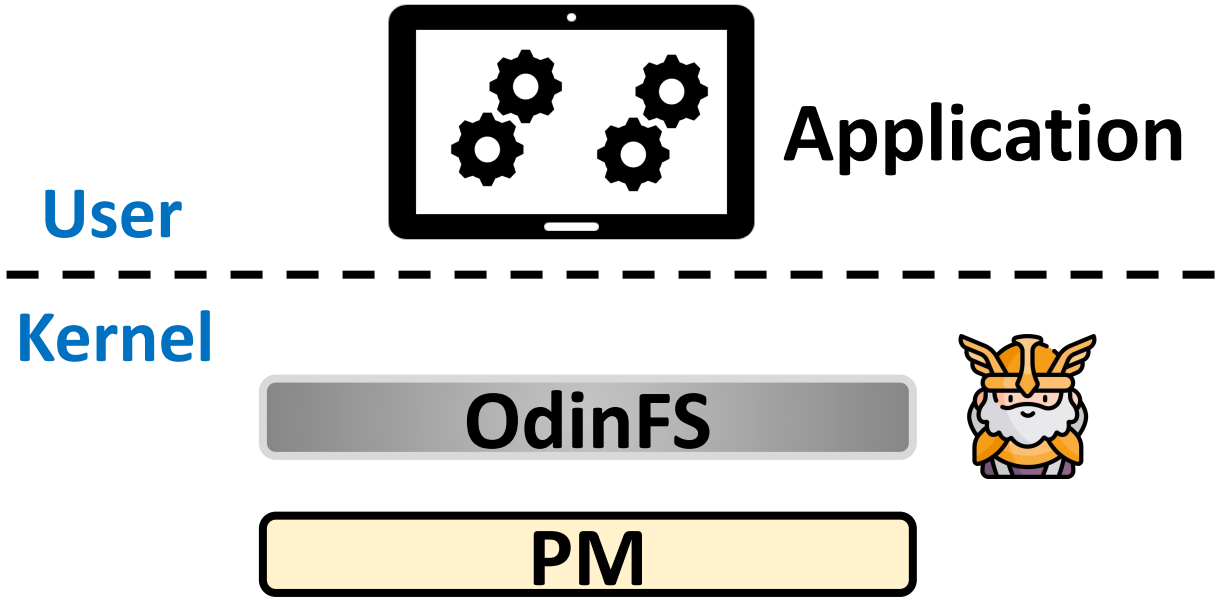
Finding: Remote PM access degrades severely

Access PM as local as possible

OdinFS

File system maximizes PM performance via opportunistic delegation

OdinFS



- In-kernel PM file systems
- POSIX-compliant

OdinFS design: Delegate IO requests

Idea: **Decouple PM access from application threads**

Limits concurrent access

→ Preserves maximal PM performance within a NUMA node

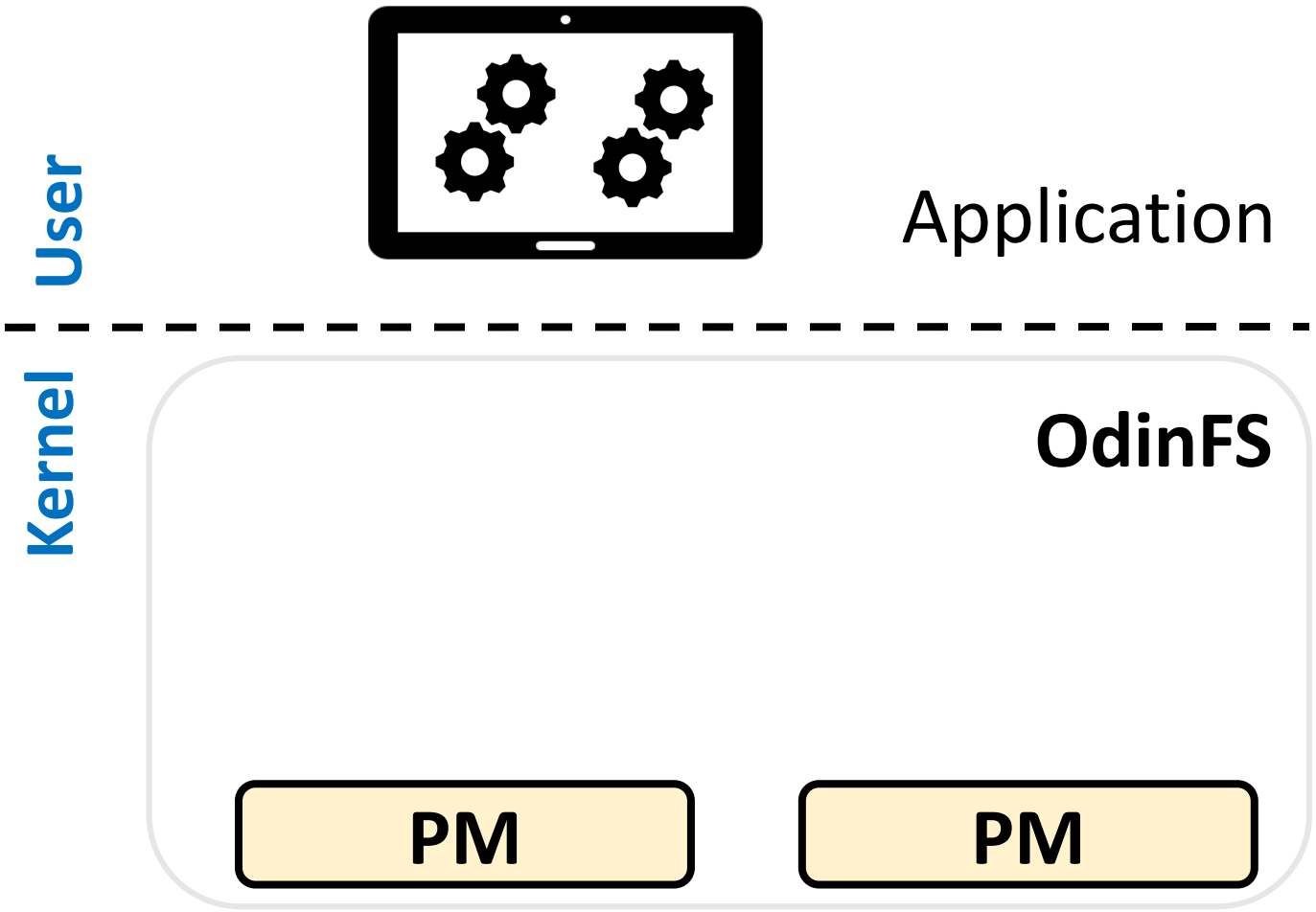
Localized PM access

→ Minimizes PM NUMA impact and enables efficient remote access

Efficiently utilizes the aggregated PM bandwidth

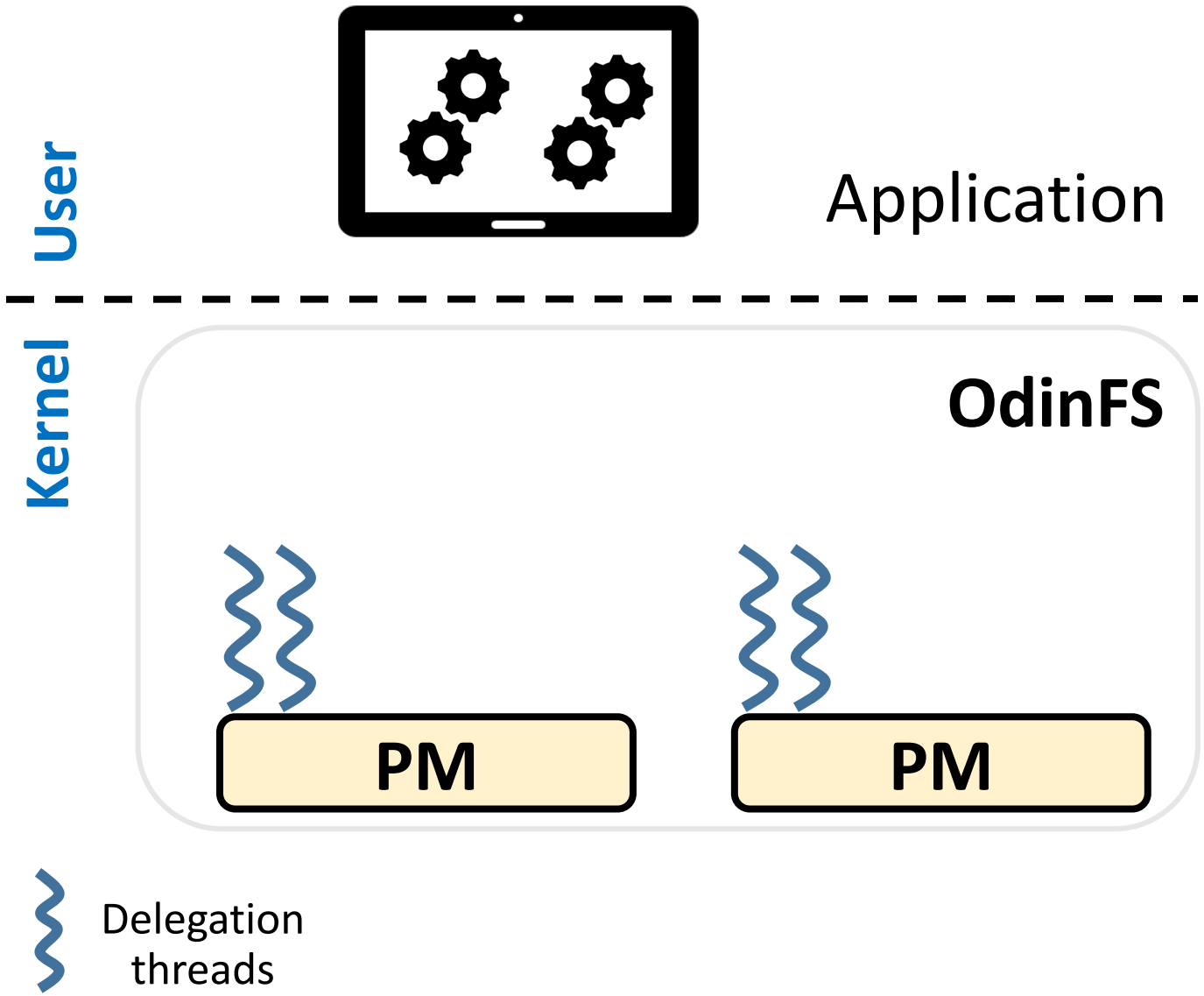
→ Arbitrary applications can utilize the entire machine bandwidth

Decouple PM access from application threads



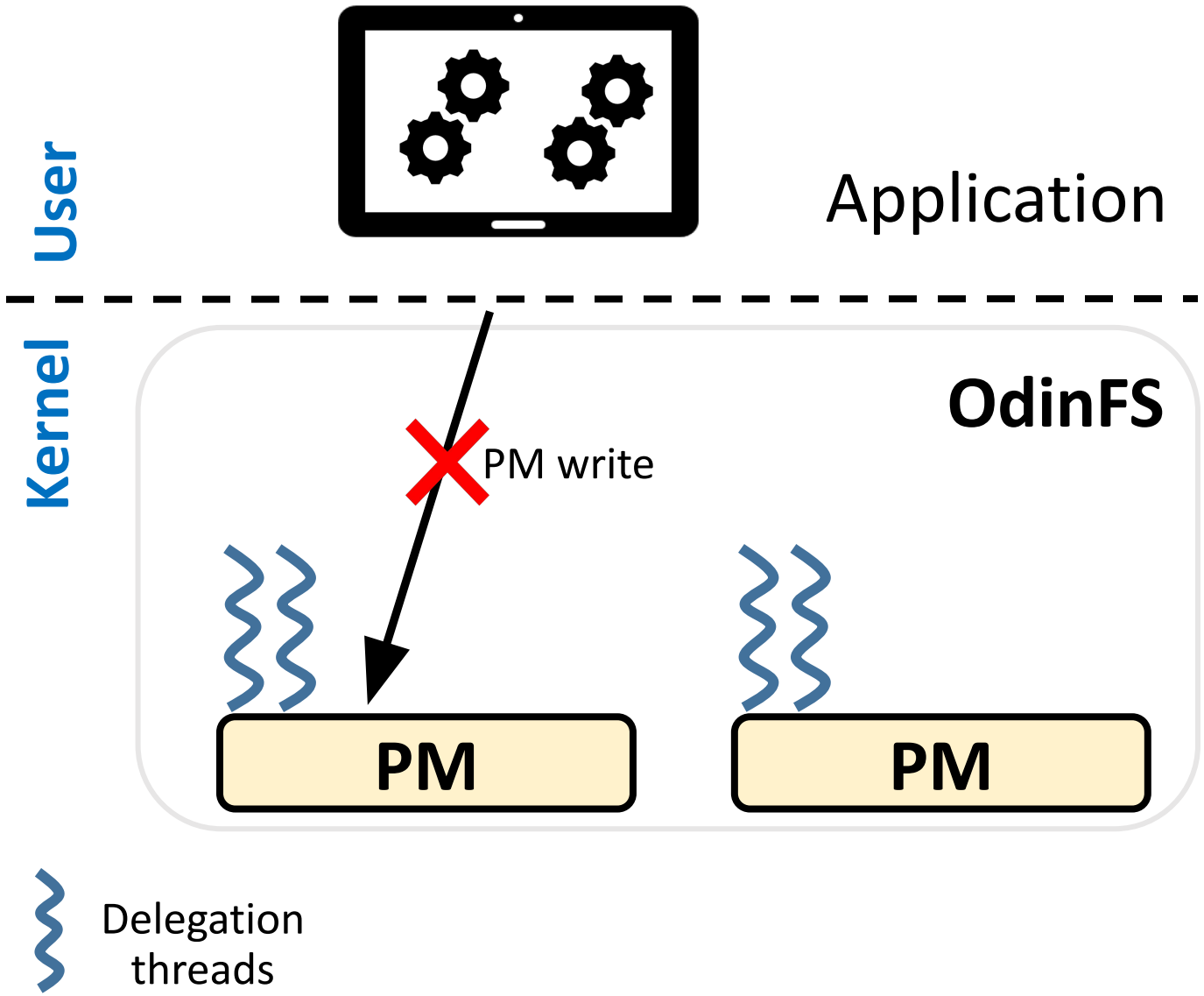
- **Delegation threads access PM on application's behalf**

Decouple PM access from application threads



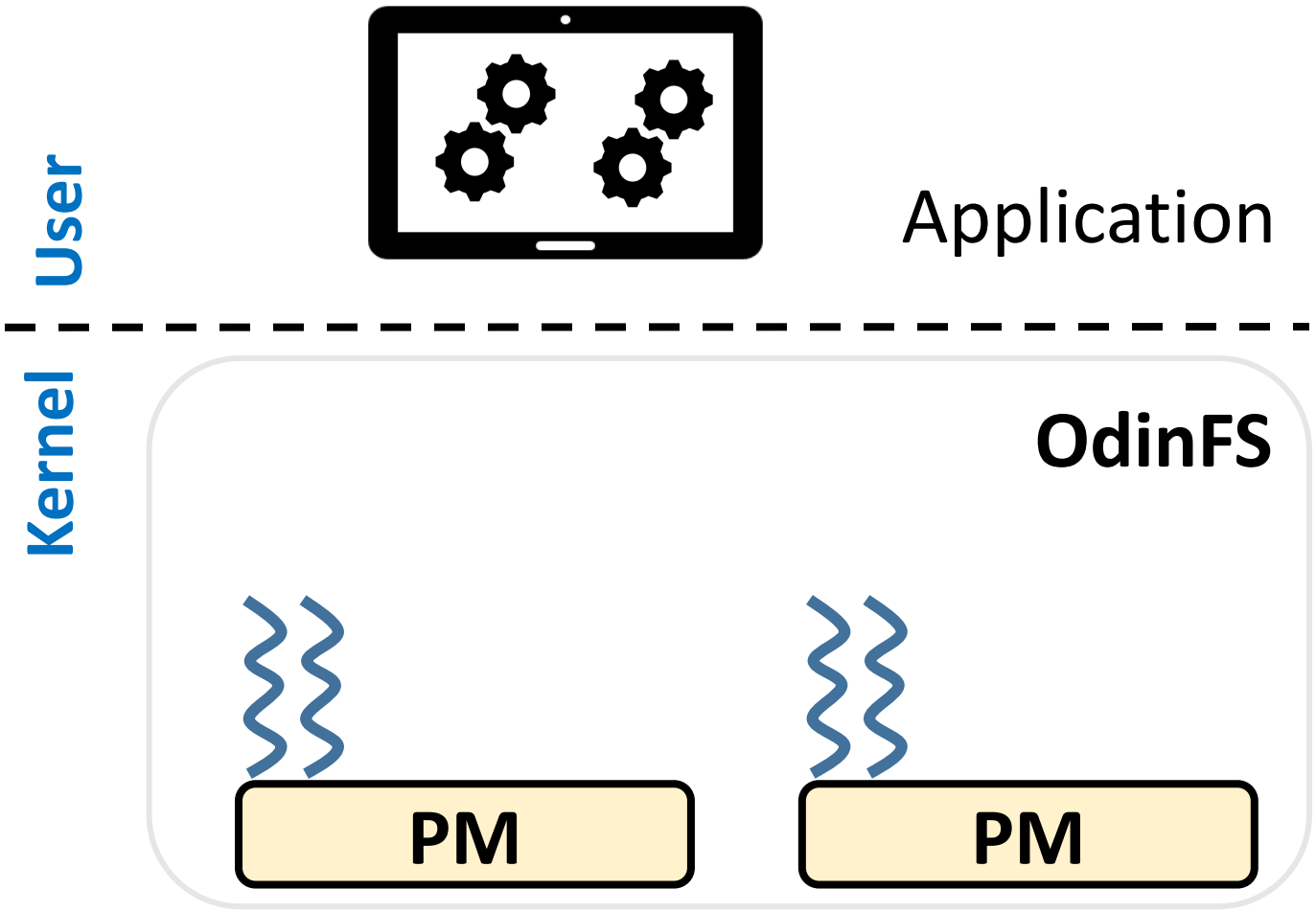
- Delegation threads access PM on application's behalf

Decouple PM access from application threads



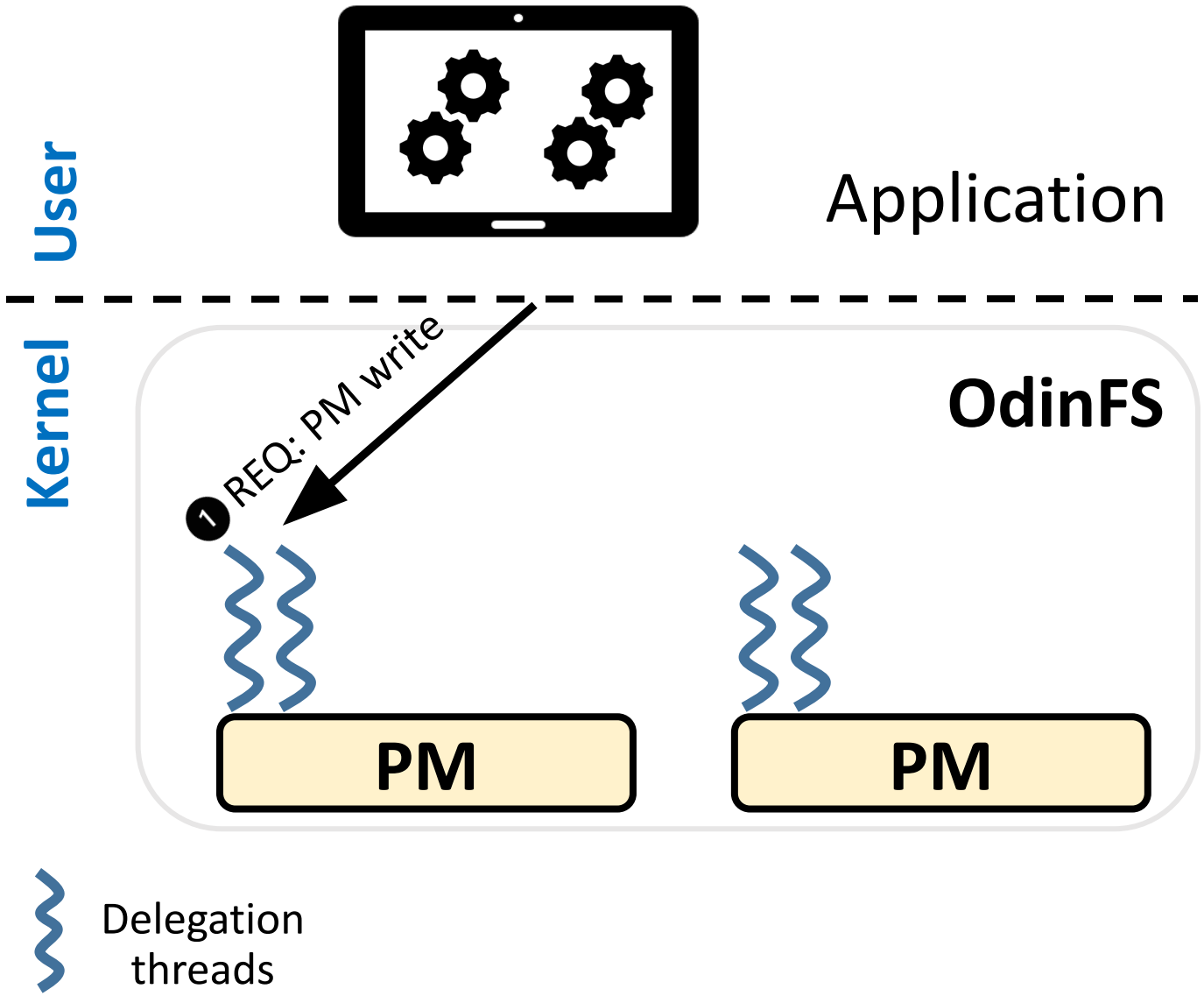
- Delegation threads access PM on application's behalf

Decouple PM access from application threads



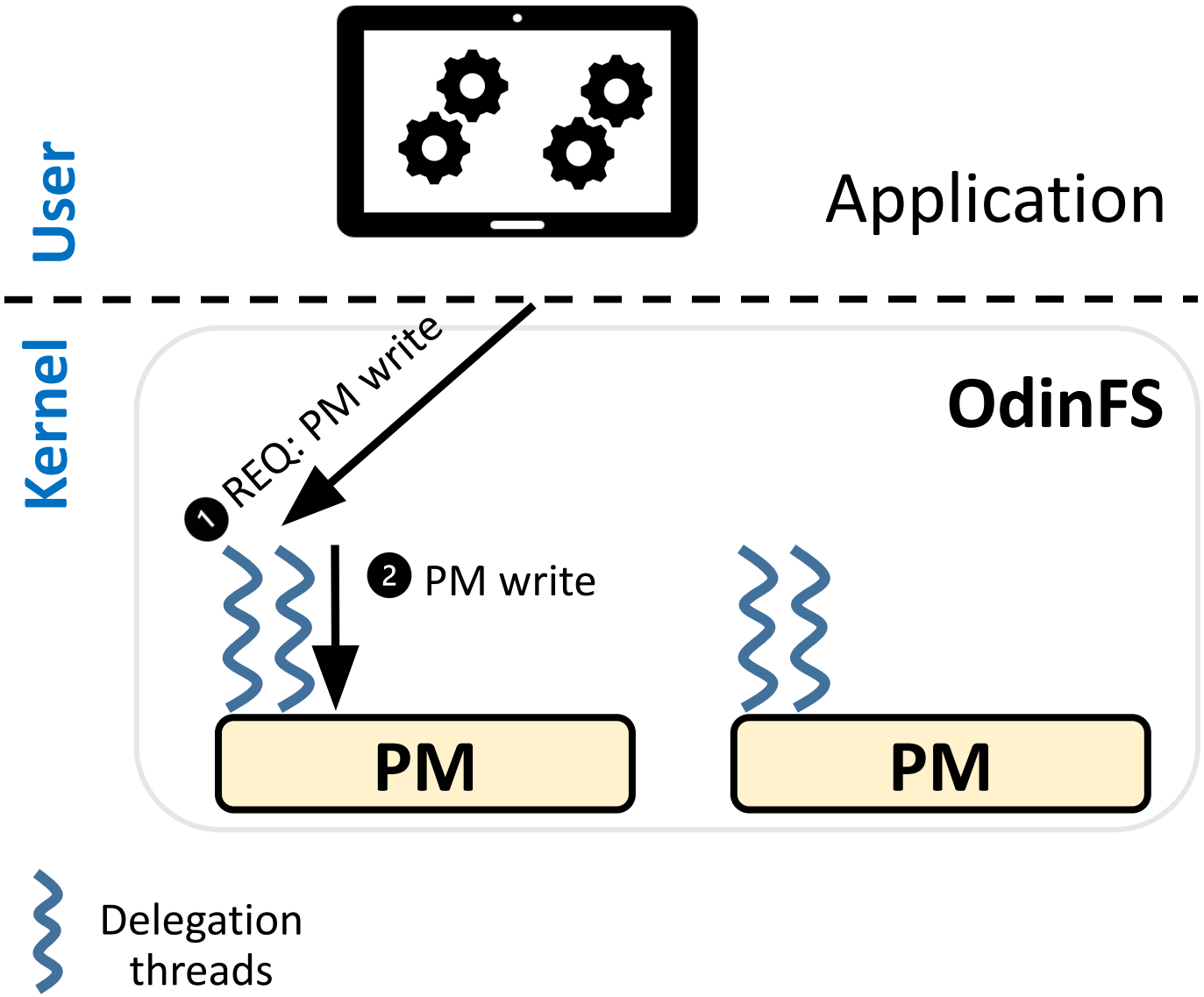
- Delegation threads access PM on application's behalf

Decouple PM access from application threads



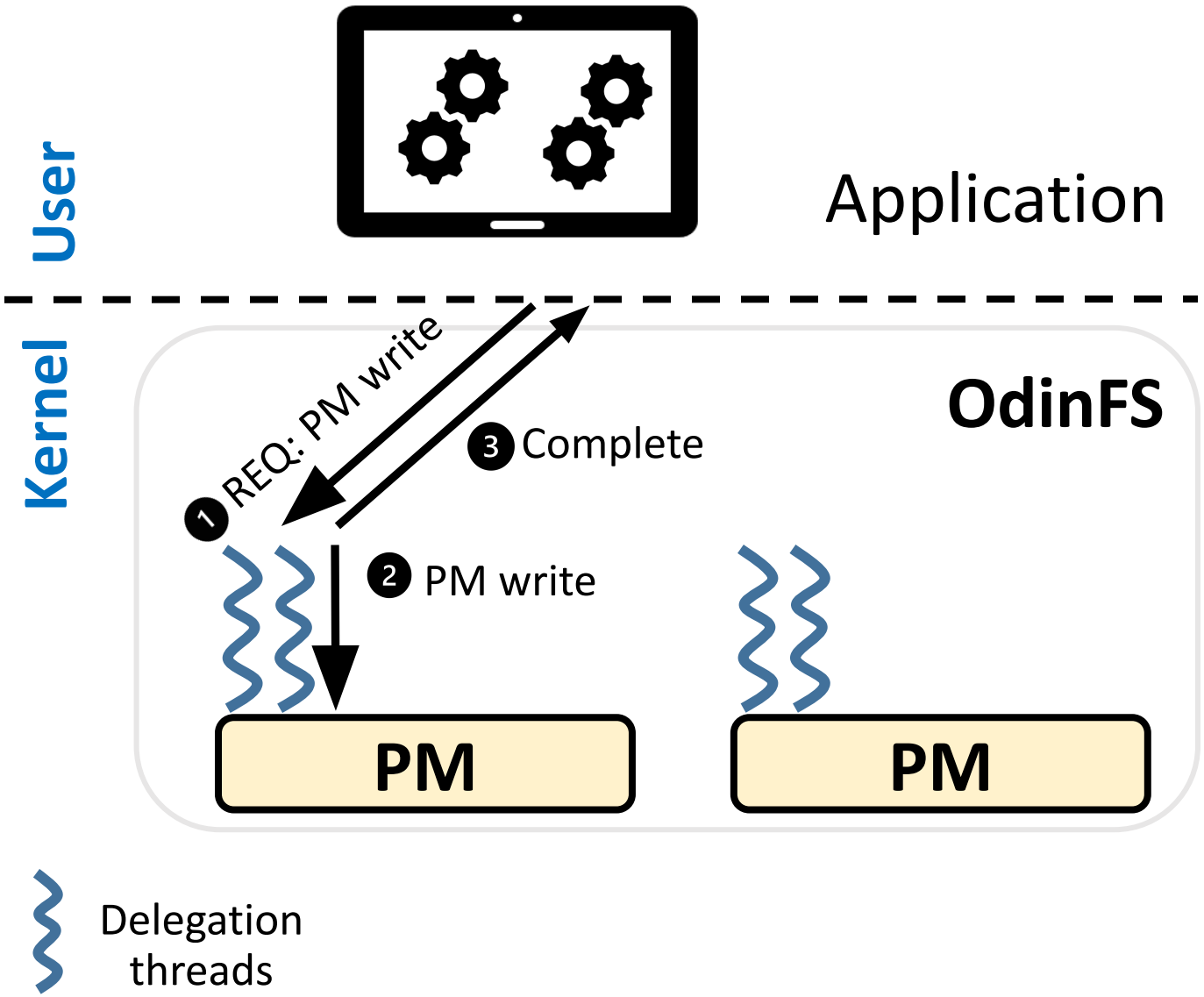
- Delegation threads access PM on application's behalf

Decouple PM access from application threads



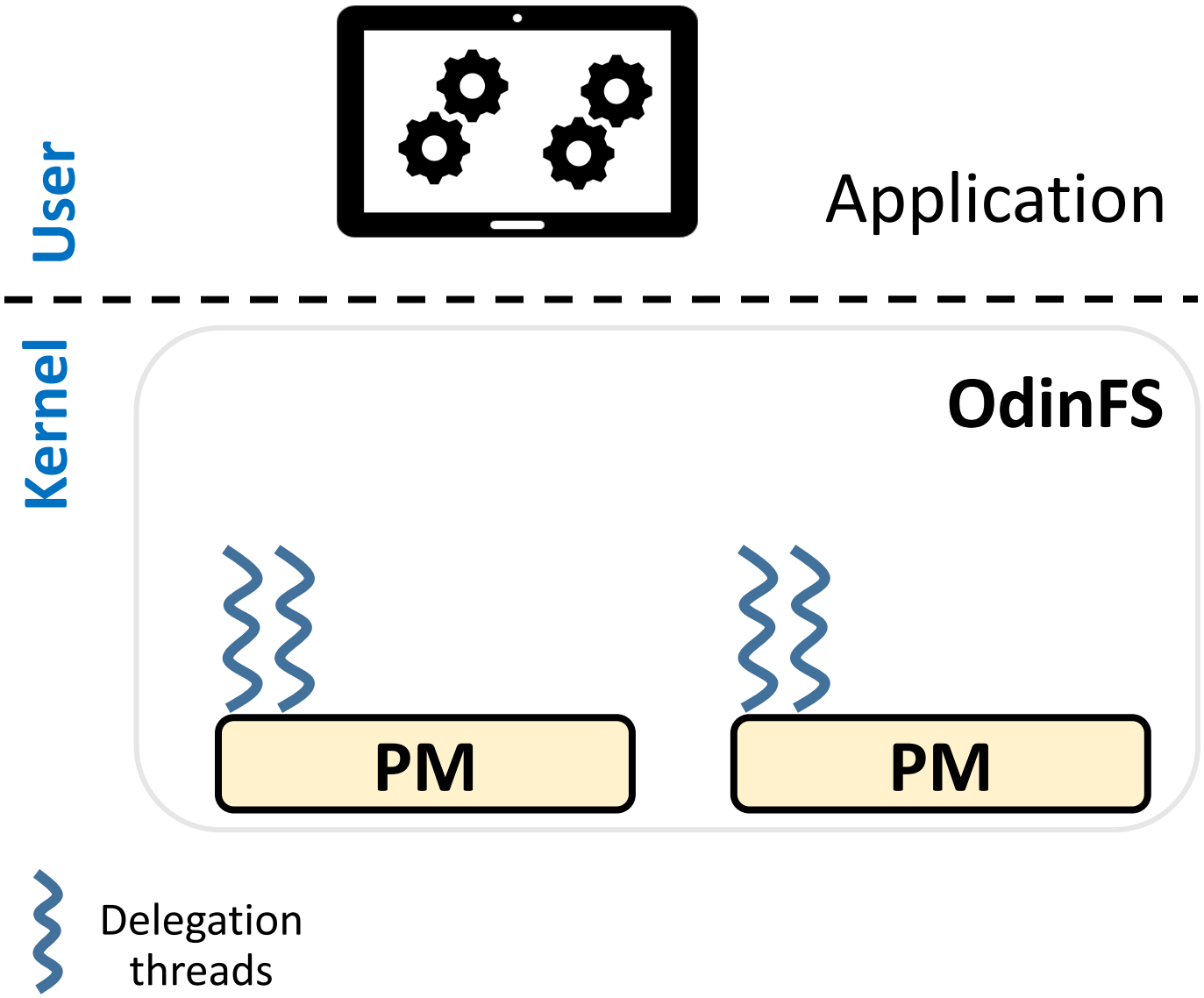
- Delegation threads access PM on application's behalf

Decouple PM access from application threads



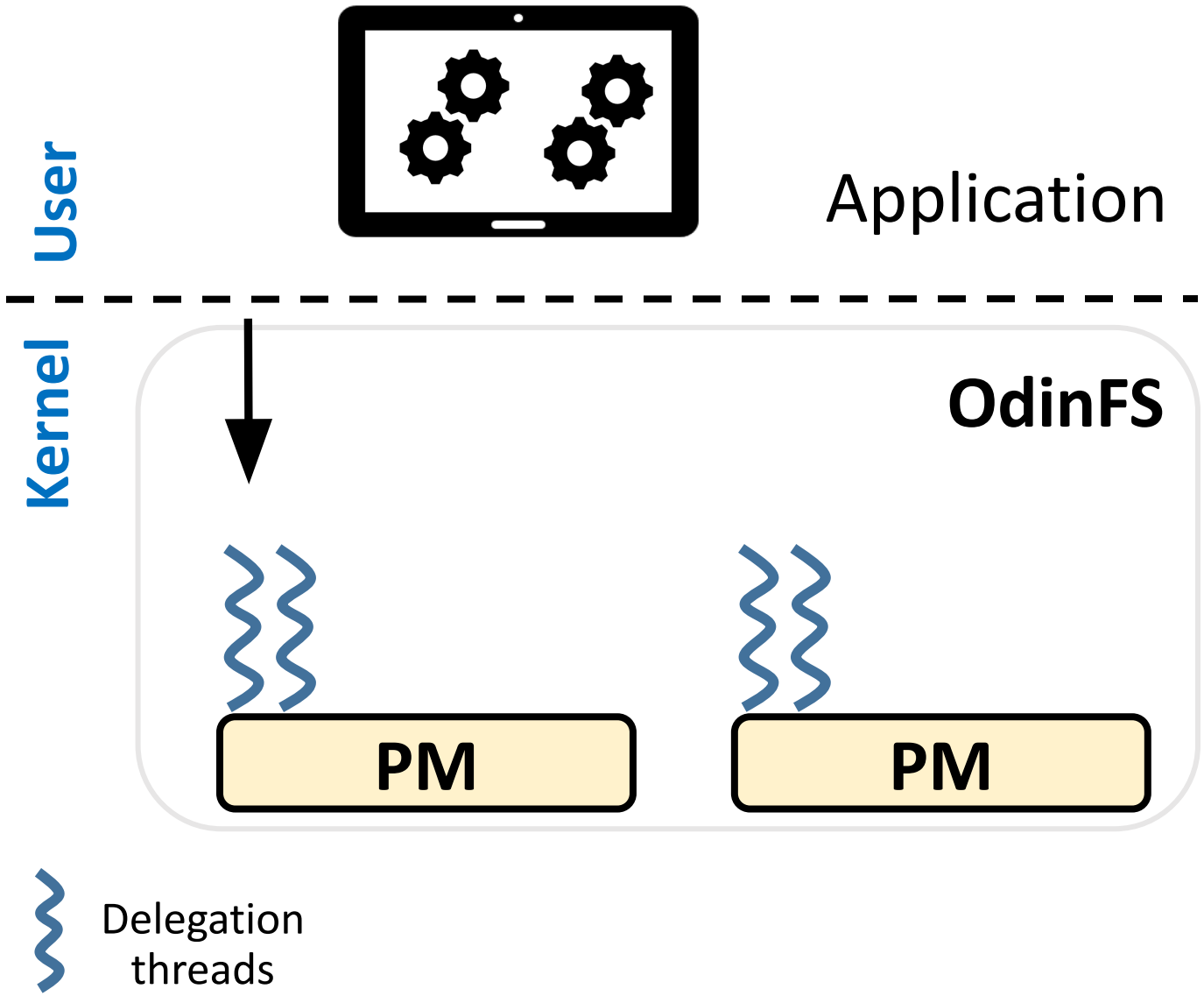
- Delegation threads access PM on application's behalf

Delegation limits and localizes PM access



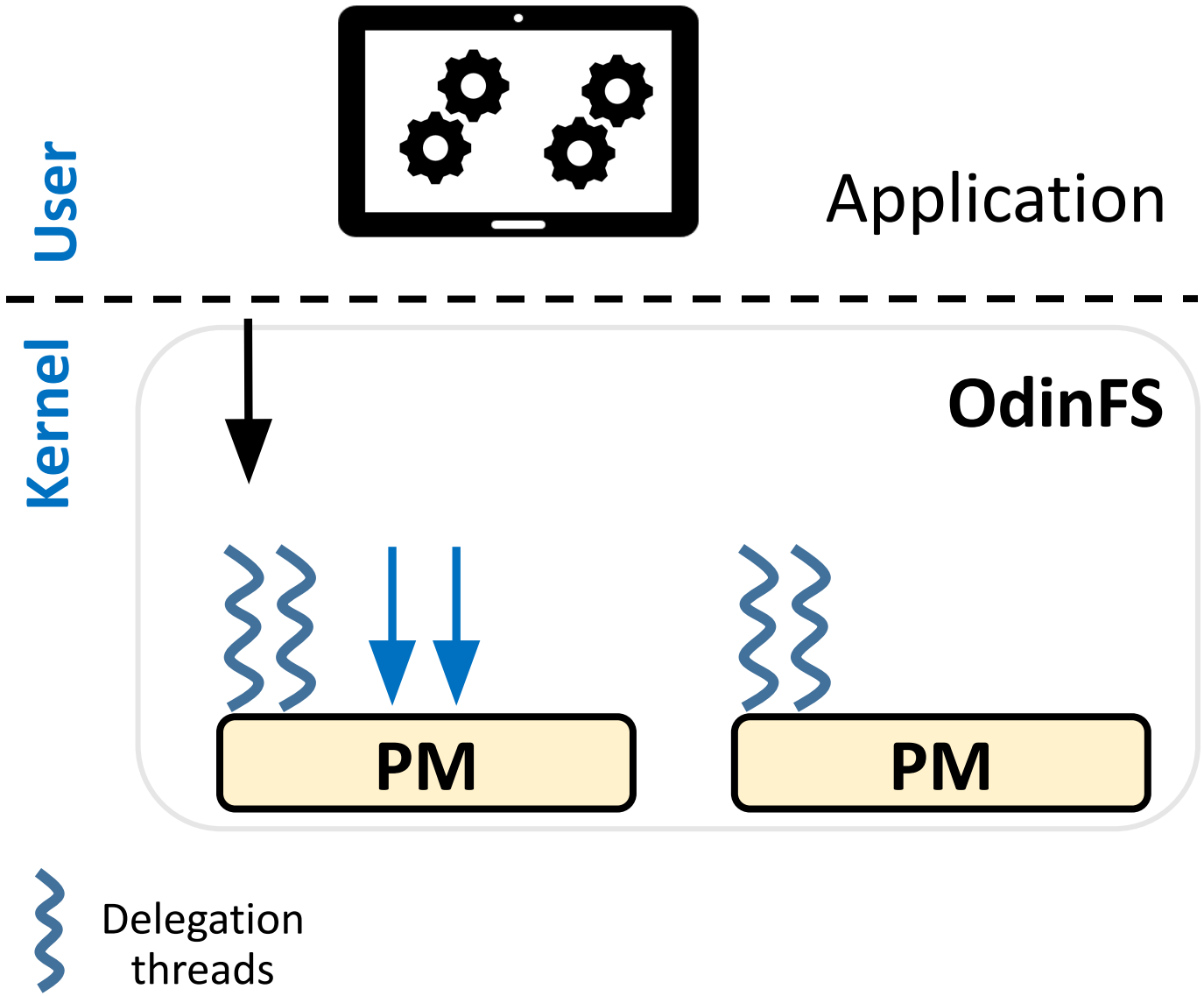
- Delegation threads access PM on application's behalf
- **Delegation threads control the access to PM**

Delegation limits and localizes PM access



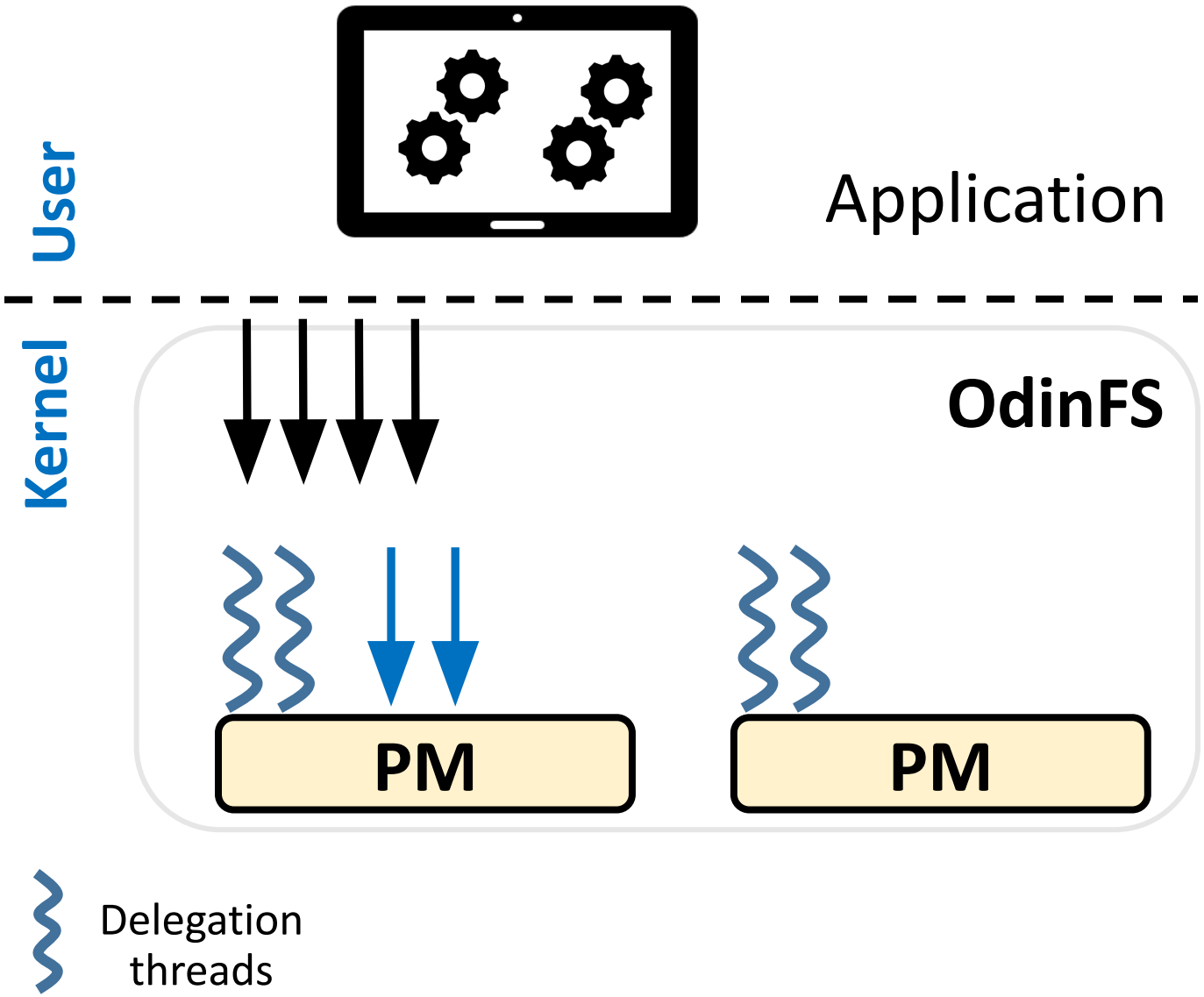
- Delegation threads access PM on application's behalf
- **Delegation threads control the access to PM**

Delegation limits and localizes PM access



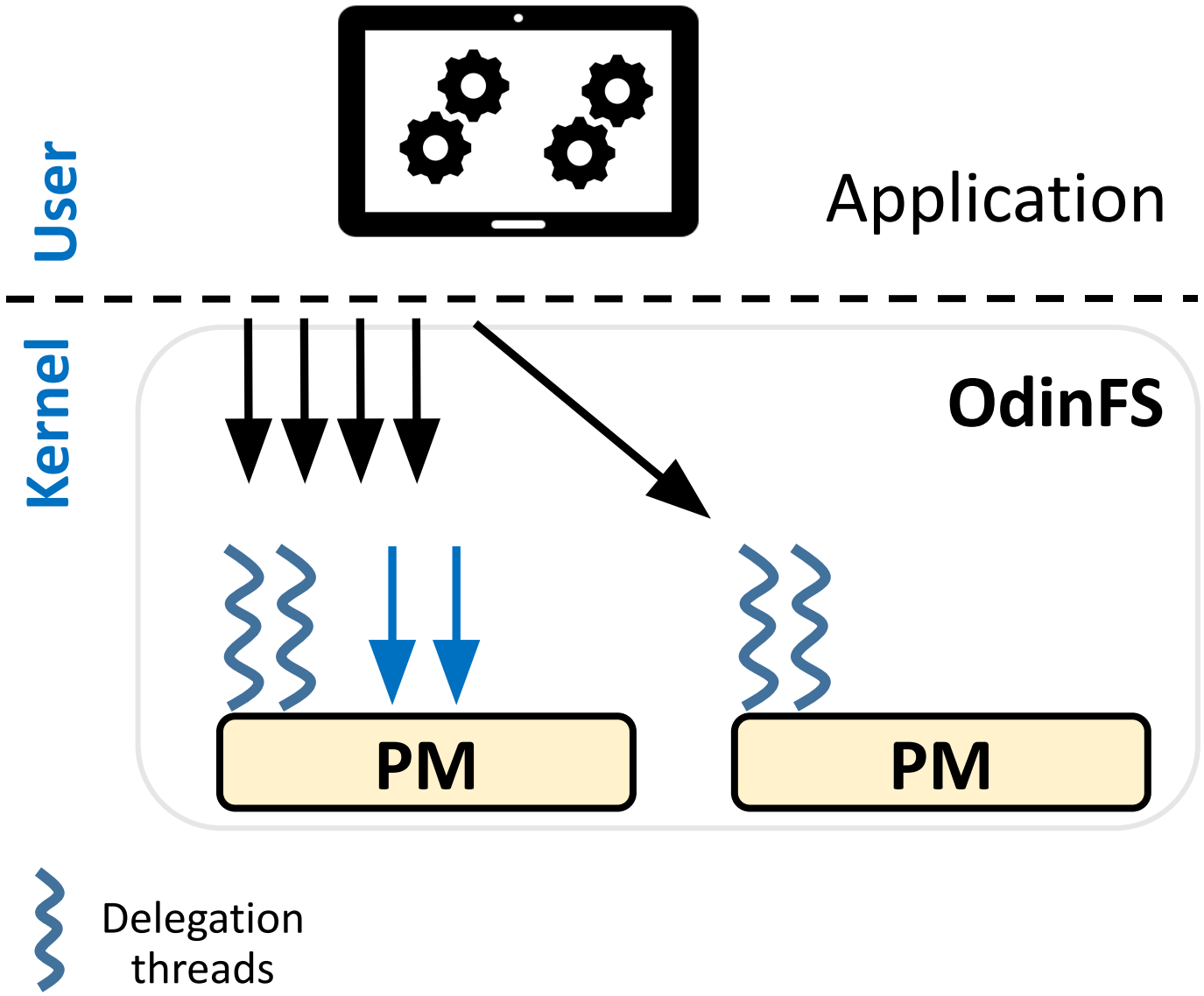
- Delegation threads access PM on application's behalf
- **Delegation threads control the access to PM**

Delegation limits and localizes PM access



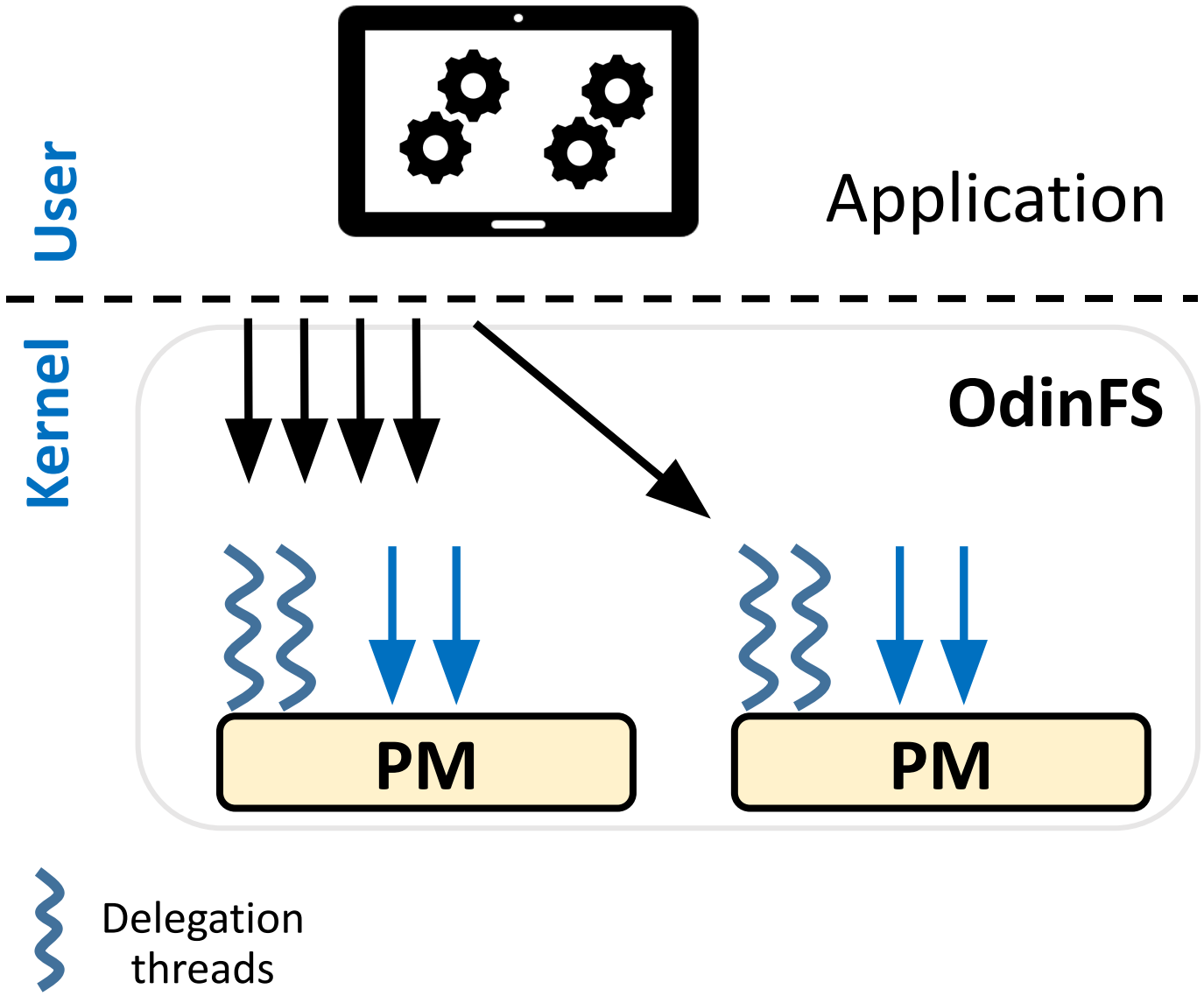
- Delegation threads access PM on application's behalf
- **Delegation threads control the access to PM**

Delegation limits and localizes PM access



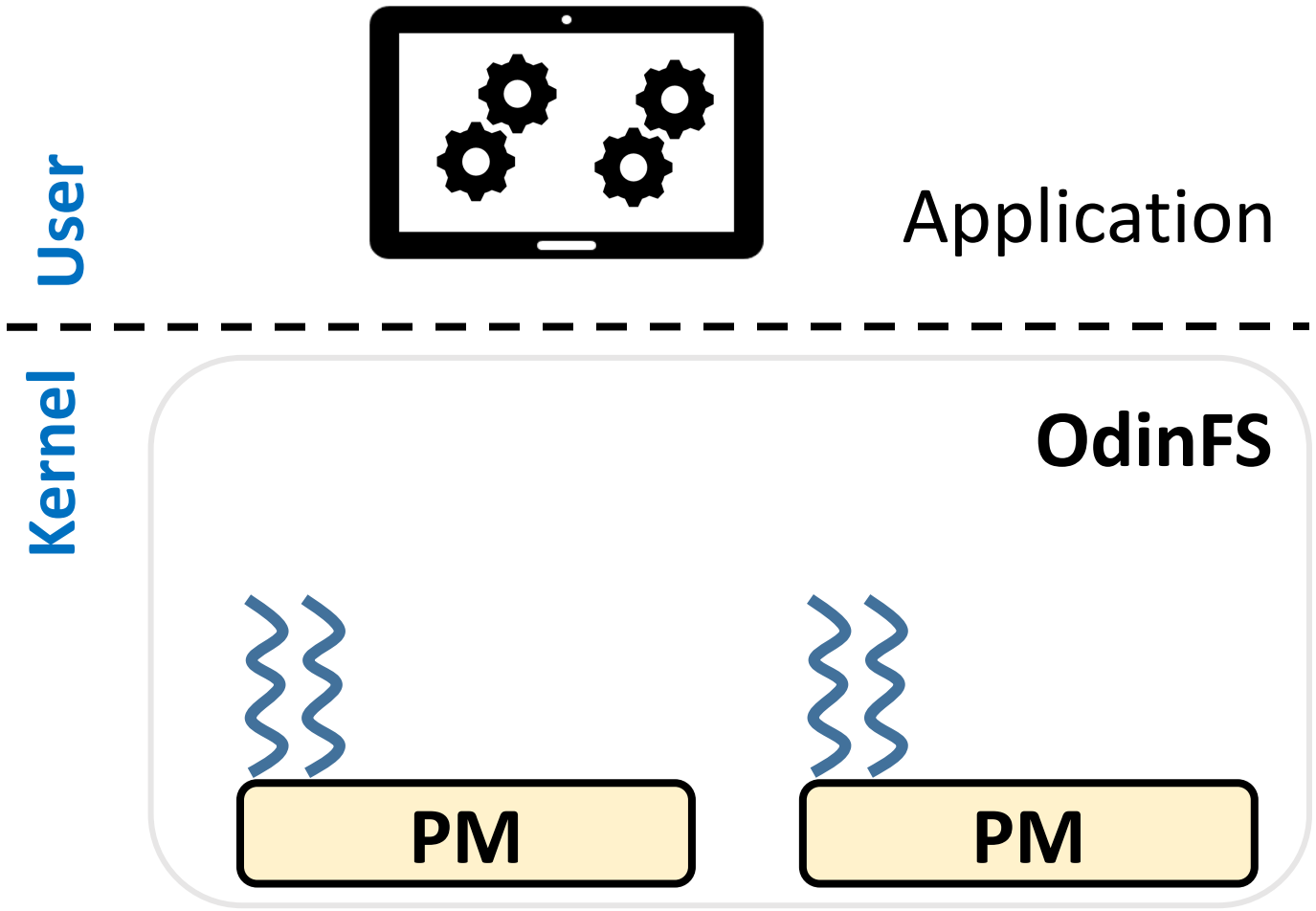
- Delegation threads access PM on application's behalf
- Delegation threads control the access to PM
- **Delegation threads always access PM locally**

Delegation limits and localizes PM access



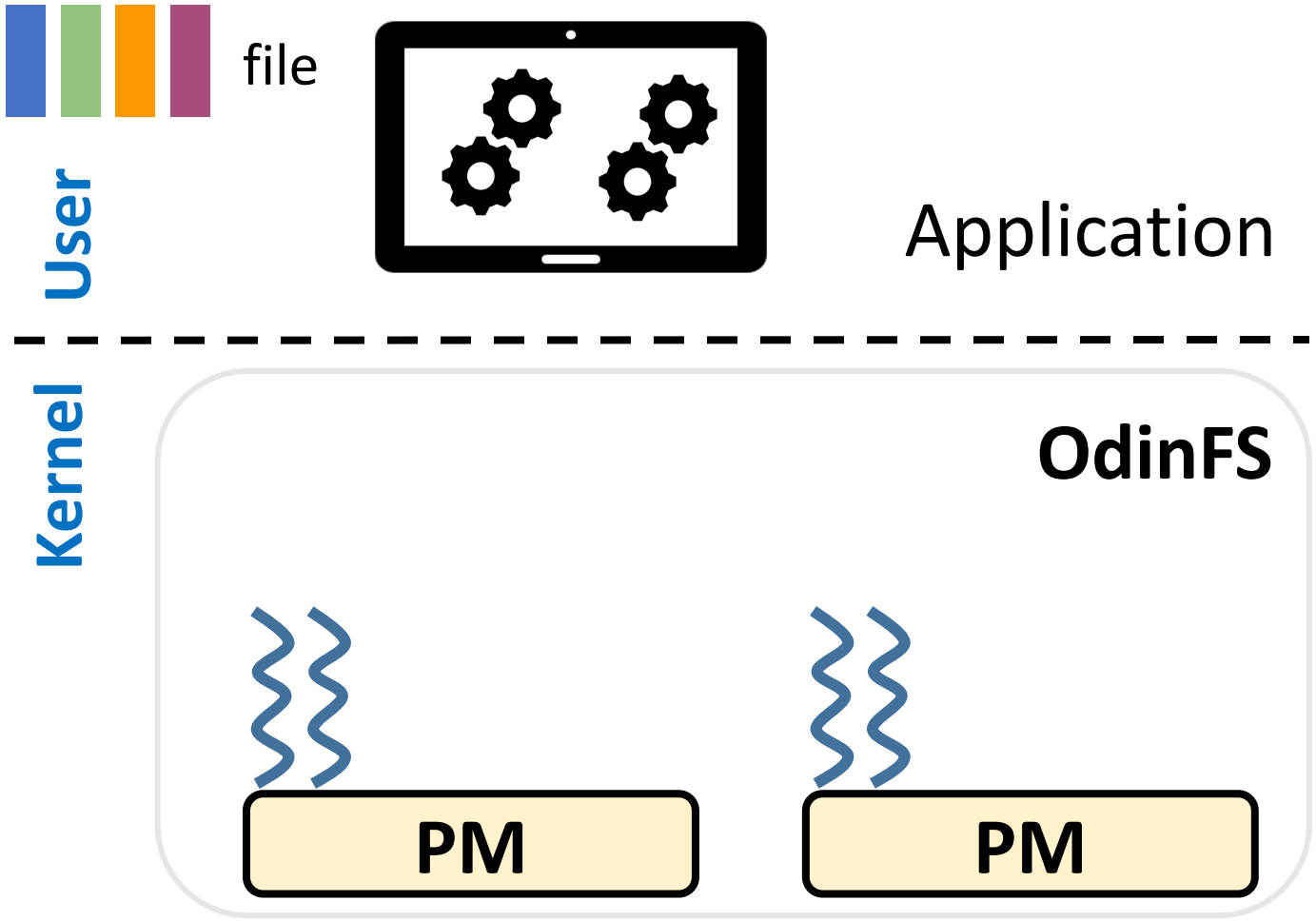
- Delegation threads access PM on application's behalf
- Delegation threads control the access to PM
- **Delegation threads always access PM locally**

Delegation + data striping parallelize IO ops



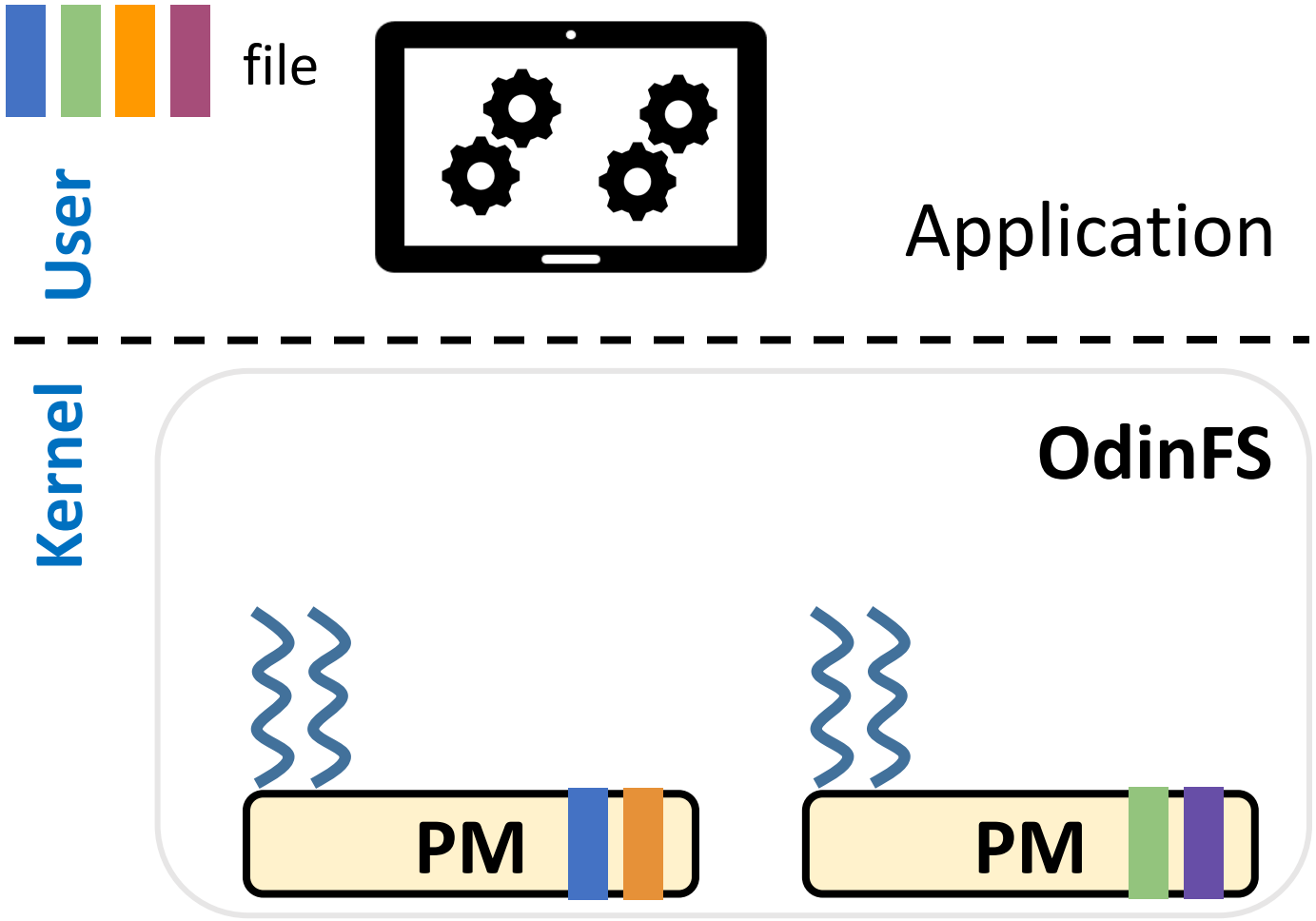
- Delegation threads access PM on application's behalf
- Delegation threads control the access to PM
- Delegation threads always access PM locally
- **Aggregate PM bandwidth utilization by partitioning data**

Delegation + data striping parallelize IO ops



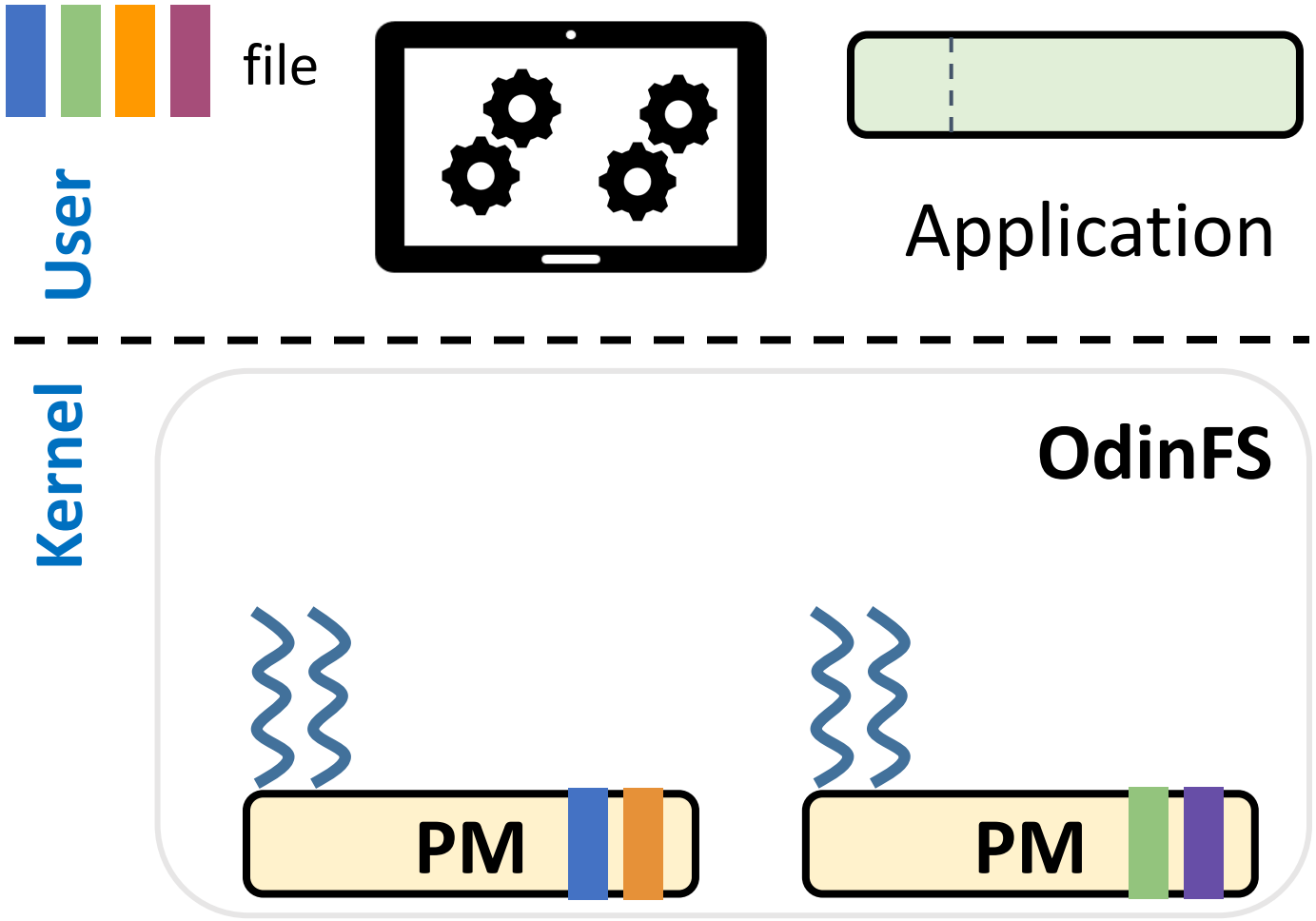
- Delegation threads access PM on application's behalf
- Delegation threads control the access to PM
- Delegation threads always access PM locally
- **Aggregate PM bandwidth utilization by partitioning data**

Delegation + data striping parallelize IO ops



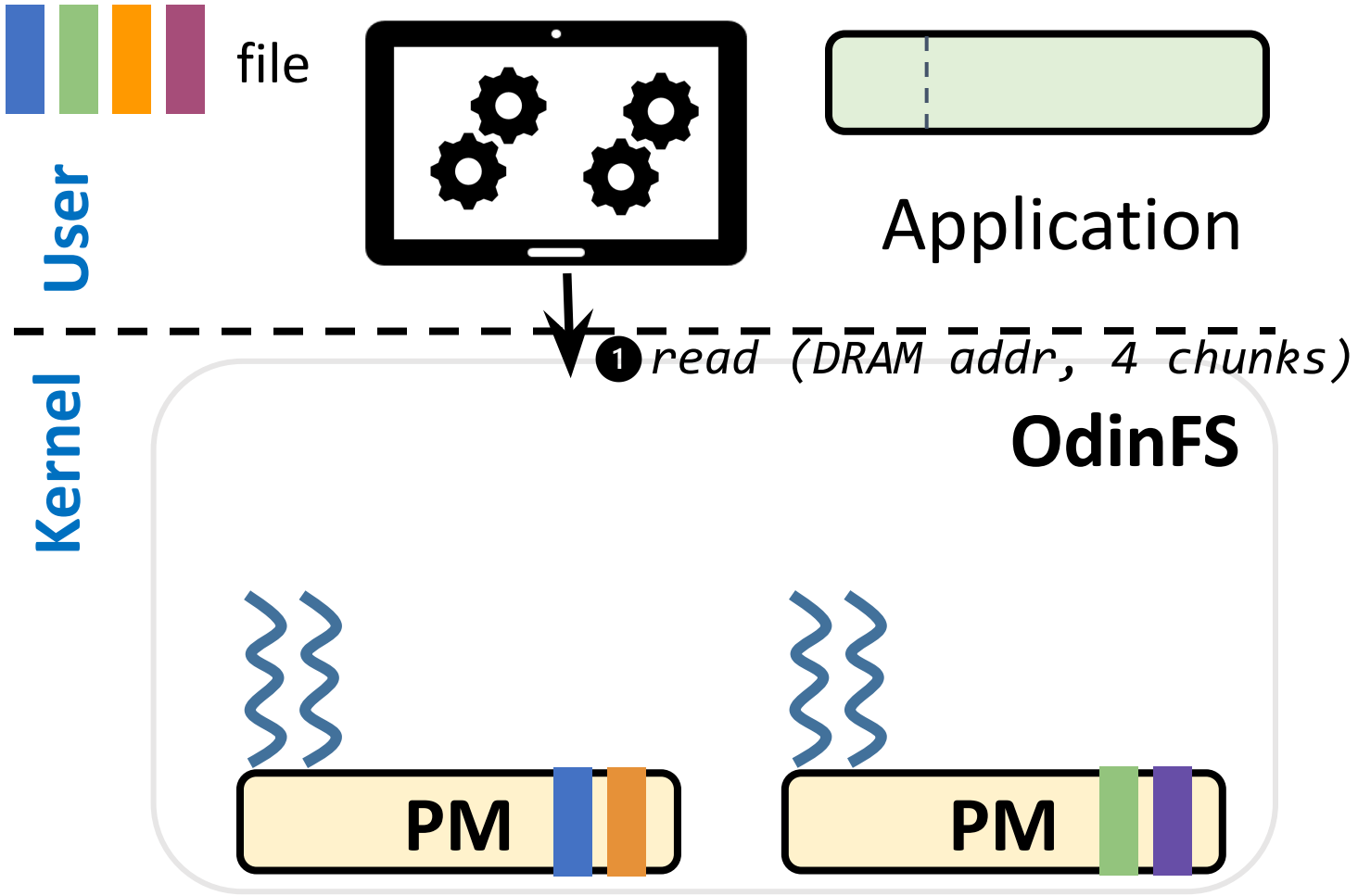
- Delegation threads access PM on application's behalf
- Delegation threads control the access to PM
- Delegation threads always access PM locally
- **Aggregate PM bandwidth utilization by partitioning data**

Delegation + data striping parallelize IO ops



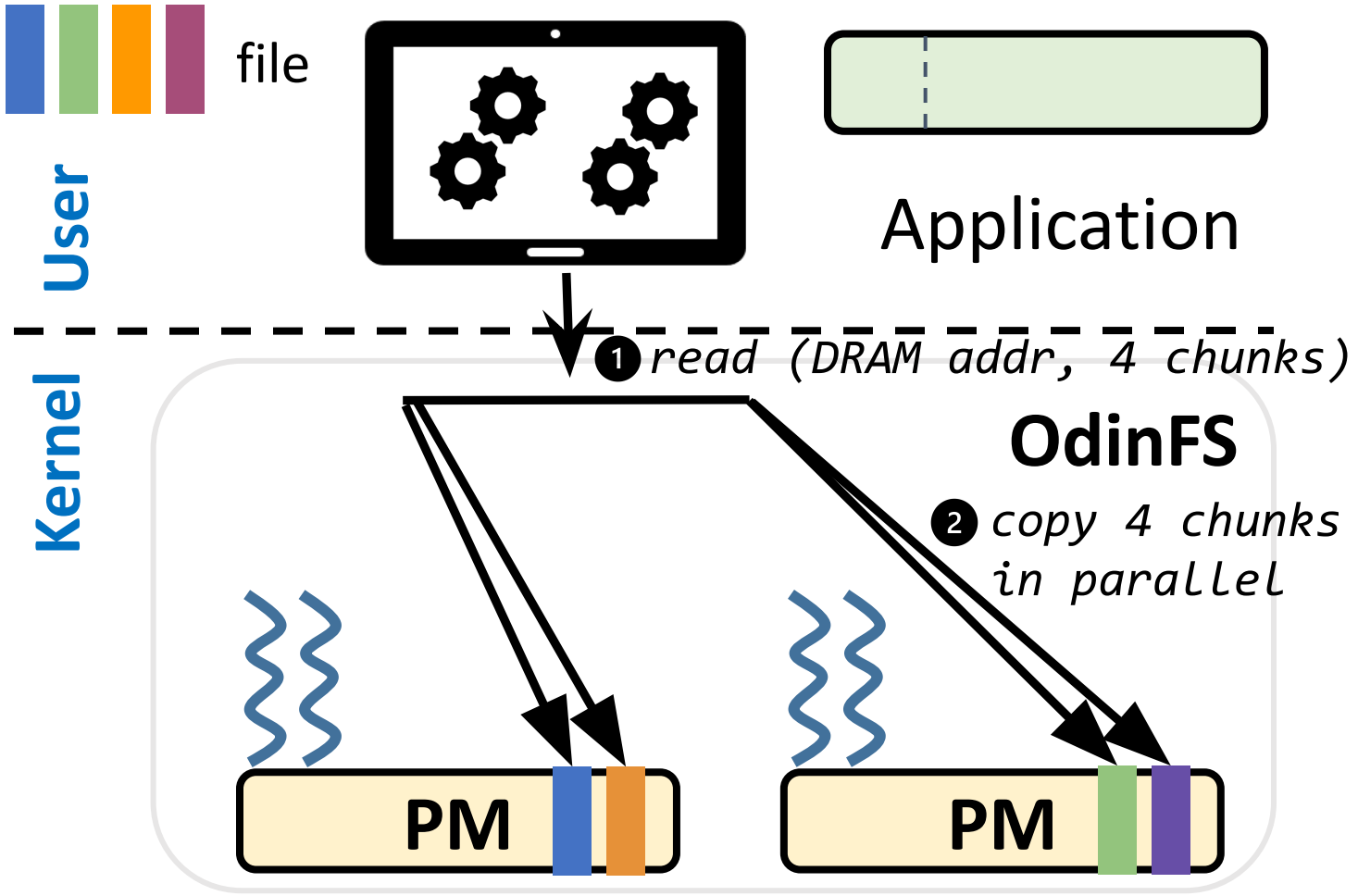
- Delegation threads access PM on application's behalf
- Delegation threads control the access to PM
- Delegation threads always access PM locally
- Aggregate PM bandwidth utilization by partitioning data
- **Parallelize IO requests with per-NUMA delegation threads**

Delegation + data striping parallelize IO ops



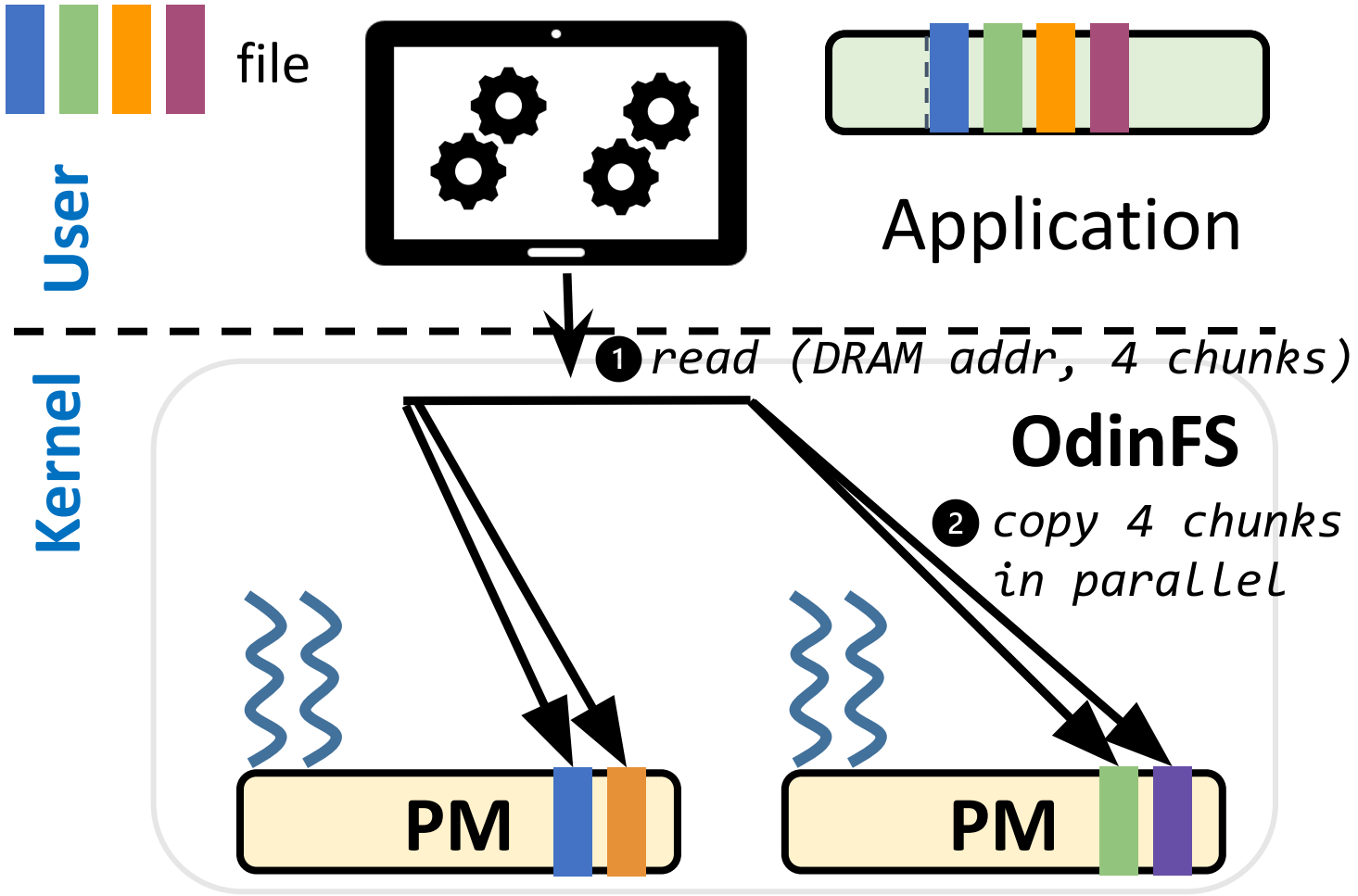
- Delegation threads access PM on application's behalf
- Delegation threads control the access to PM
- Delegation threads always access PM locally
- Aggregate PM bandwidth utilization by partitioning data
- **Parallelize IO requests with per-NUMA delegation threads**

Delegation + data striping parallelize IO ops



- Delegation threads access PM on application's behalf
- Delegation threads control the access to PM
- Delegation threads always access PM locally
- Aggregate PM bandwidth utilization by partitioning data
- **Parallelize IO requests with per-NUMA delegation threads**

Delegation + data striping parallelize IO ops



- Delegation threads access PM on application's behalf
- Delegation threads control the access to PM
- Delegation threads always access PM locally
- Aggregate PM bandwidth utilization by partitioning data
- **Parallelize IO requests with per-NUMA delegation threads**

OdinFS: Other design considerations

- Maximizes concurrent access by reducing synchronization overhead
  - Within a file: range locks
  - Across file system: per-CPU data structures

- Employs a new mechanism to ensure crash consistency

- Ensures minimal overhead of delegation threads

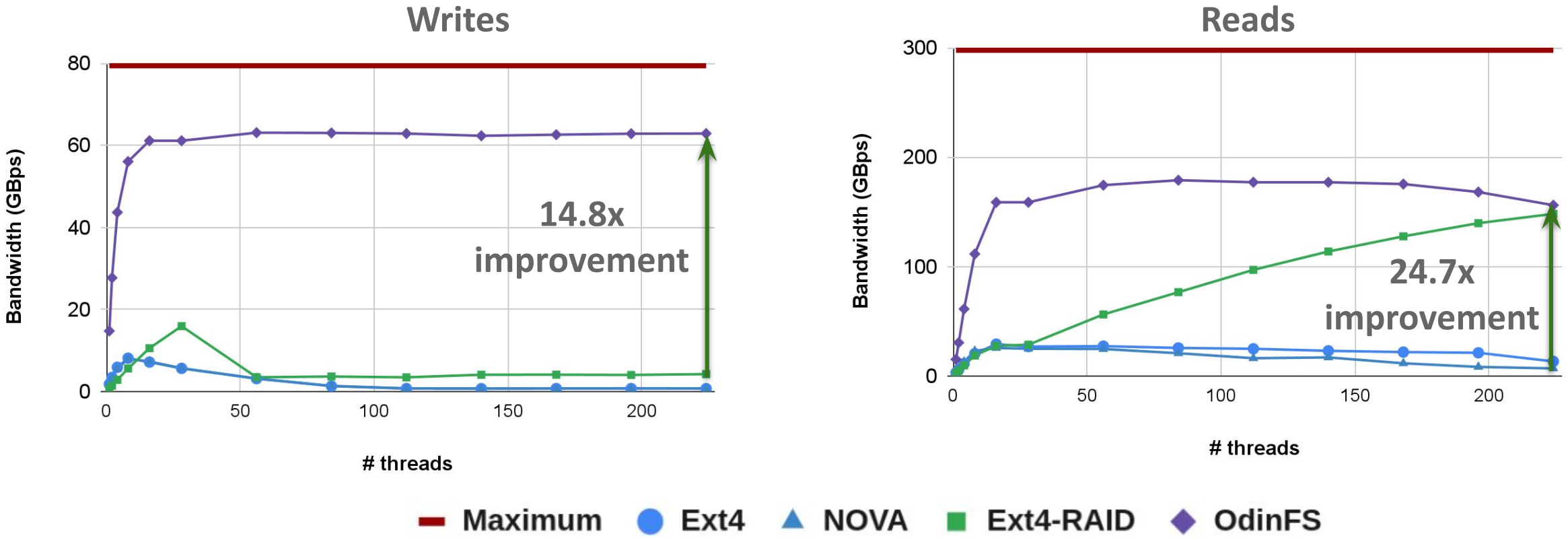
Analysis of OdinFS

- OdinFS performance:
  - Does OdinFS improves the IO performance?
    - Throughput, small access, high concurrency

Setup: 224-core/8-socket machine

OdinFS performance

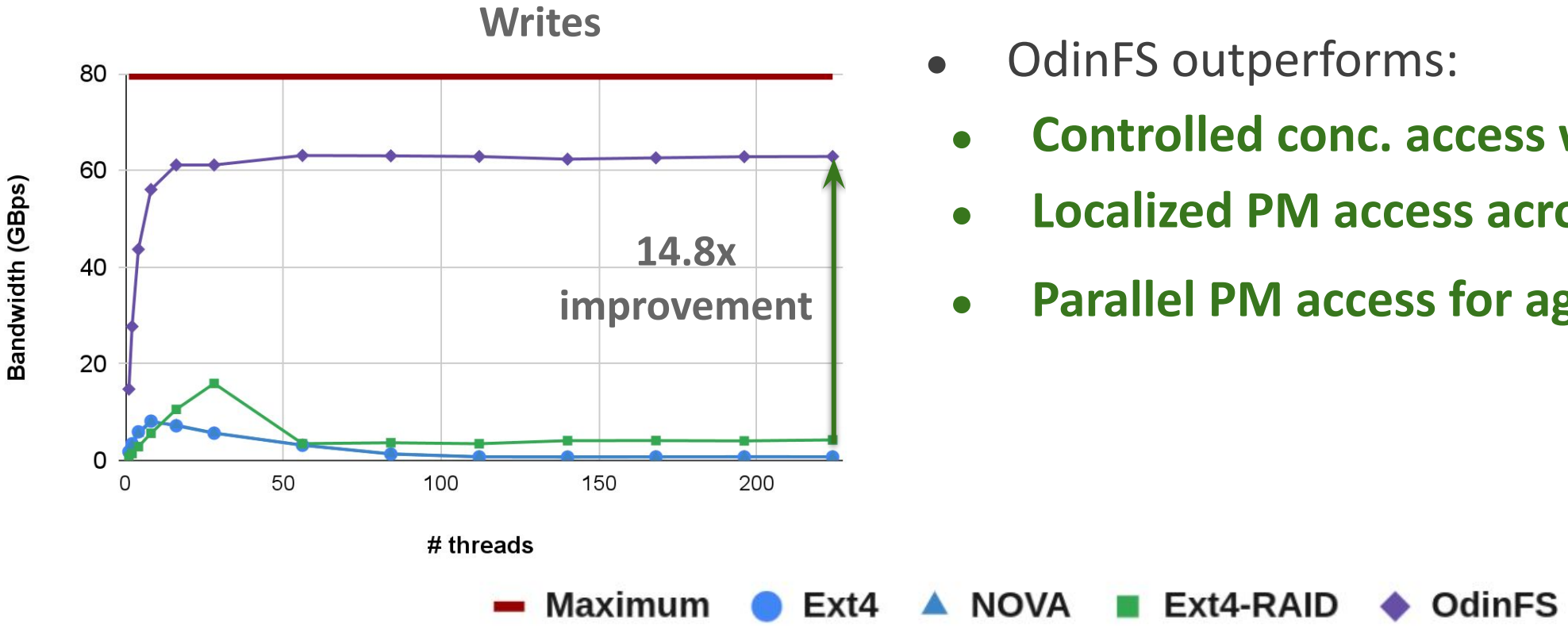
Benchmark: Each thread reads/writes 2MB data privately in a 1GB file



Setup: 224-core/8-socket machine

OdinFS performance

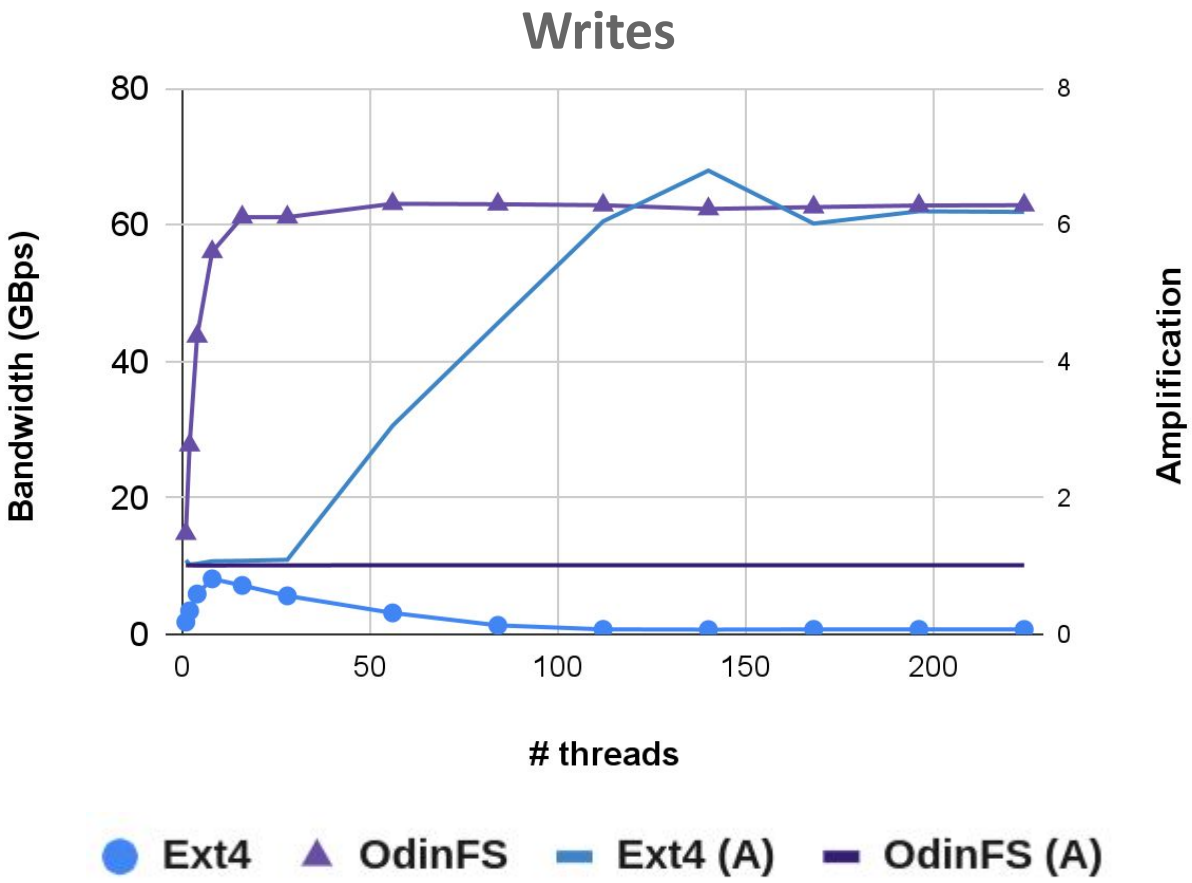
Benchmark: Each thread reads/writes 2MB data privately in a 1GB file



- OdinFS outperforms:
  - **Controlled conc. access within one PM**
  - **Localized PM access across PMs**
  - **Parallel PM access for aggregate PM bw**

Setup: 224-core/8-socket machine

OdinFS performance: IO amplification



- OdinFS's amplification is ~1:
  - **Localized PM access across PMs**

OdinFS performance

- OdinFS outperforms other file systems:
  - By 9.4x and 8.1x for 4KB reads and writes respectively
  - Upto 24.7x on IO intensive real-world workloads
  - Upto 269x faster on highly contended data path scenarios

Conclusion

- Existing PM file systems do not utilize PM efficiently:
  - Uncontrolled concurrent access
  - Arbitrary remote access
  - Do not utilize the aggregate bandwidth

- **OdinFS**: Decouples PM access from application threads via delegation
  - Enables **controlled**, **localized**, and **parallel** PM access

- **Future**: Exploring the delegation model for other faster storage media

Conclusion

- Existing PM file systems do not utilize PM efficiently:
  - Uncontrolled concurrent access
  - Arbitrary remote access
  - Do not utilize the aggregate bandwidth

- **OdinFS**: Decouples PM access from application threads via delegation
  - Enables **controlled**, **localized**, and **parallel** PM access

- **Future**: Exploring the delegation model for other faster storage media

Thank you!