OSDI 25

Fast and Synchronous Crash Consistency with Metadata Write Once File System

Yanqi Pan, Wen Xia, Yifeng Zhang, Xiangyu Zou, Hao Huang,
Zhenhua Li, Chentao Wu

哈爾濱工業大學(深圳)
HARBIN INSTITUTE OF TECHNOLOGY, SHENZHEN

清華大學

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

Crash Consistency

Crash consistency is the **fundamental** demand for file systems to ensure **correct crash recovery** for applications.

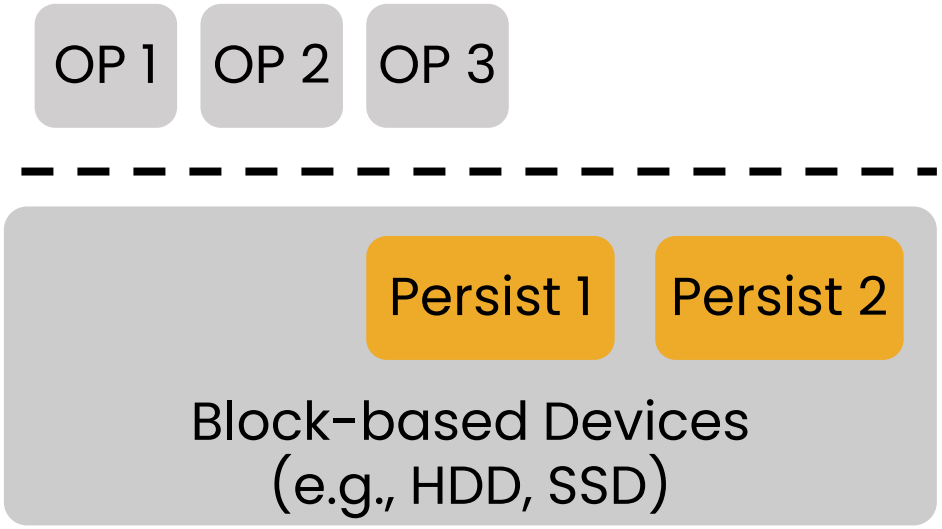
Database applications

Network storage

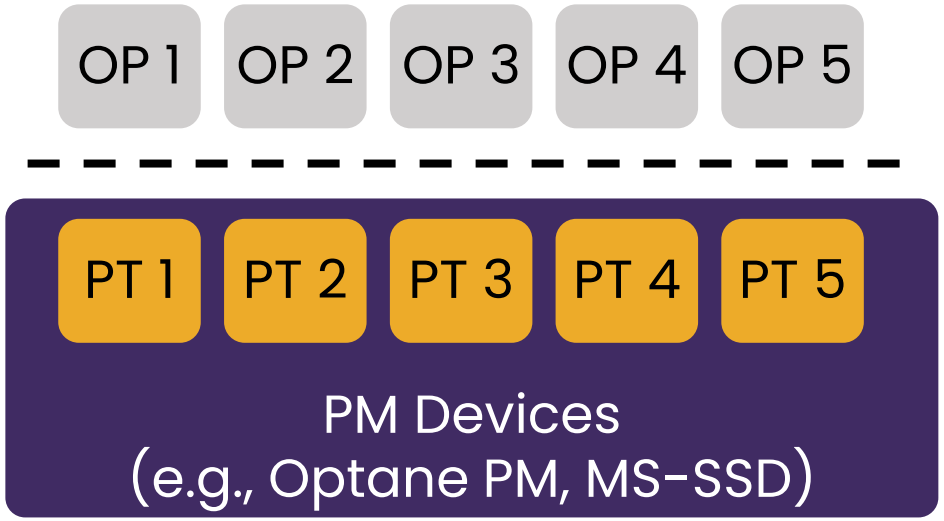
Storage for LLM

Synchronous Crash Consistency on PM

Low-latency persistent memory (PM) encourages file systems to pursue **synchronous crash consistency**.



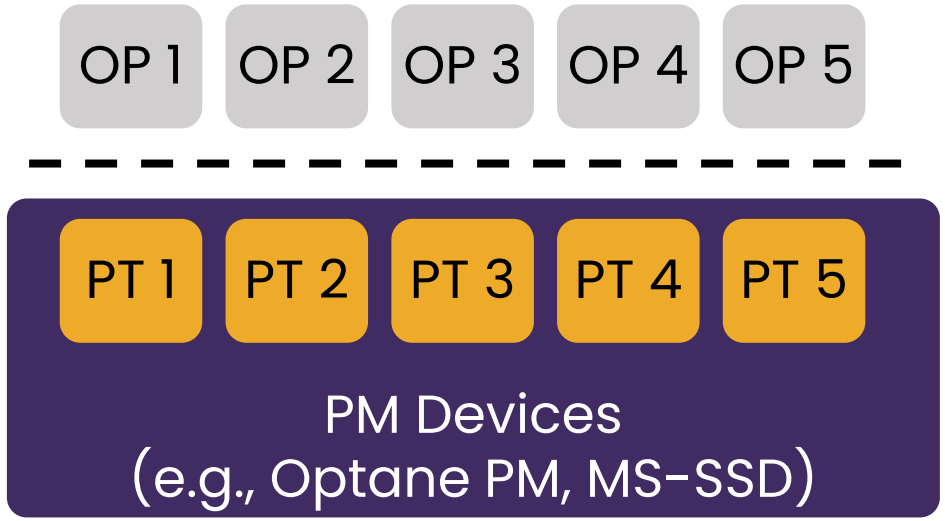
Traditional Async Crash Consistency (for Block-based File Systems)



Synchronous Crash Consistency (for PM File Systems)

Synchronous Crash Consistency on PM

Low-latency persistent memory (PM) encourages file systems to pursue **synchronous crash consistency**.



Synchronous Crash Consistency (for PM File Systems)

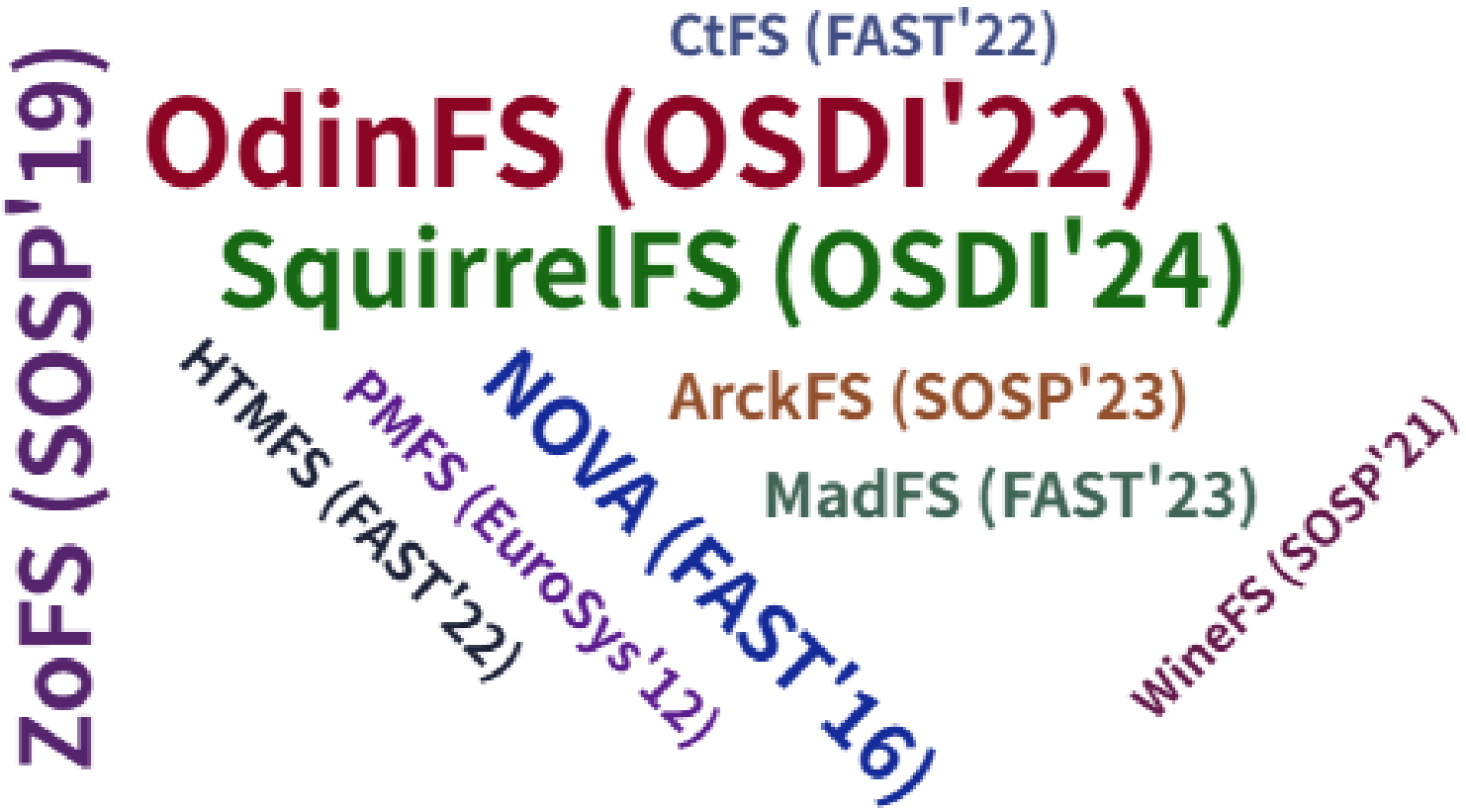
Advantages

✓**Avoid synchronization BUG**
(e.g., Avoid misused fsync)

✓**Reduce sync overhead**
(e.g., NFS strict sync protocol)

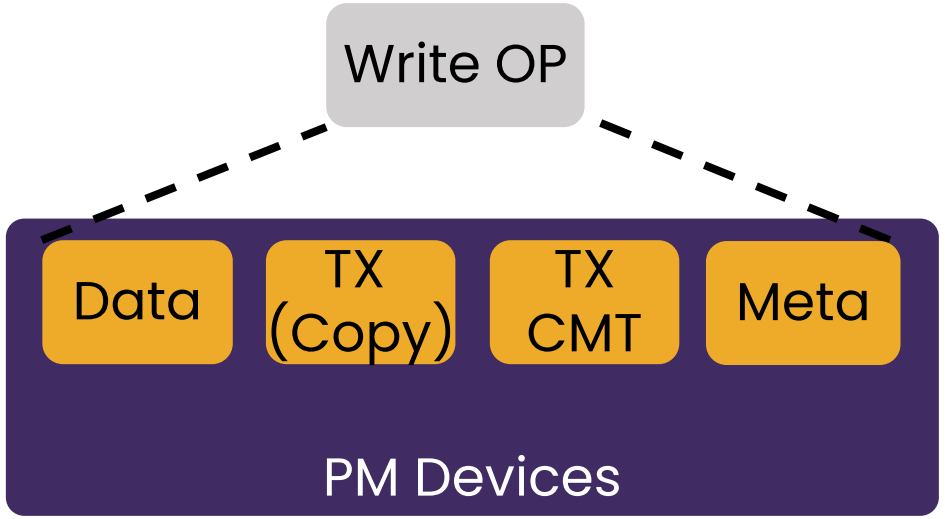
PM File Systems

Numerous PM file systems adopt synchronous crash consistency

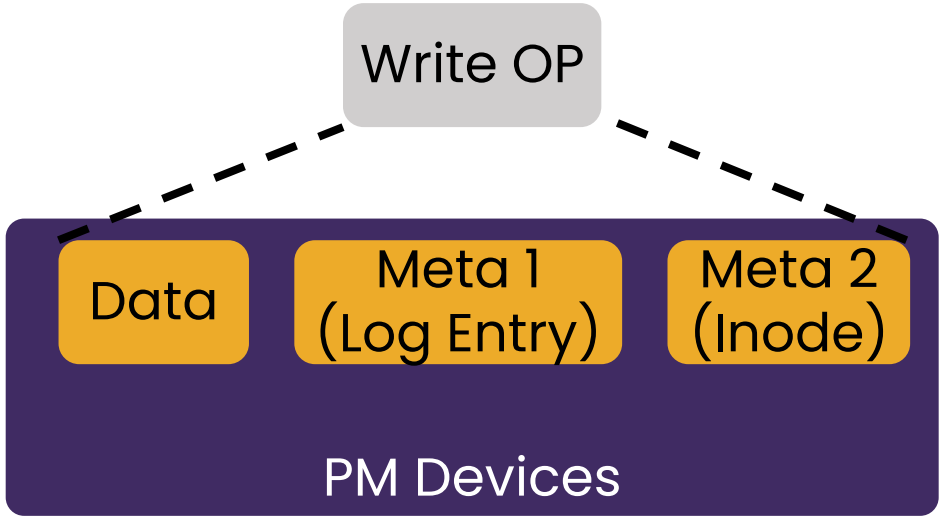


Synchronous Crash Consistency: How?

Two traditional methodologies are used for PM file systems



With Additional Writes
(e.g., Journaling File Systems: PMFS)



Without Additional Writes
(e.g., Log-structured File Systems: NOVA)

Synchronous Crash Consistency: How?

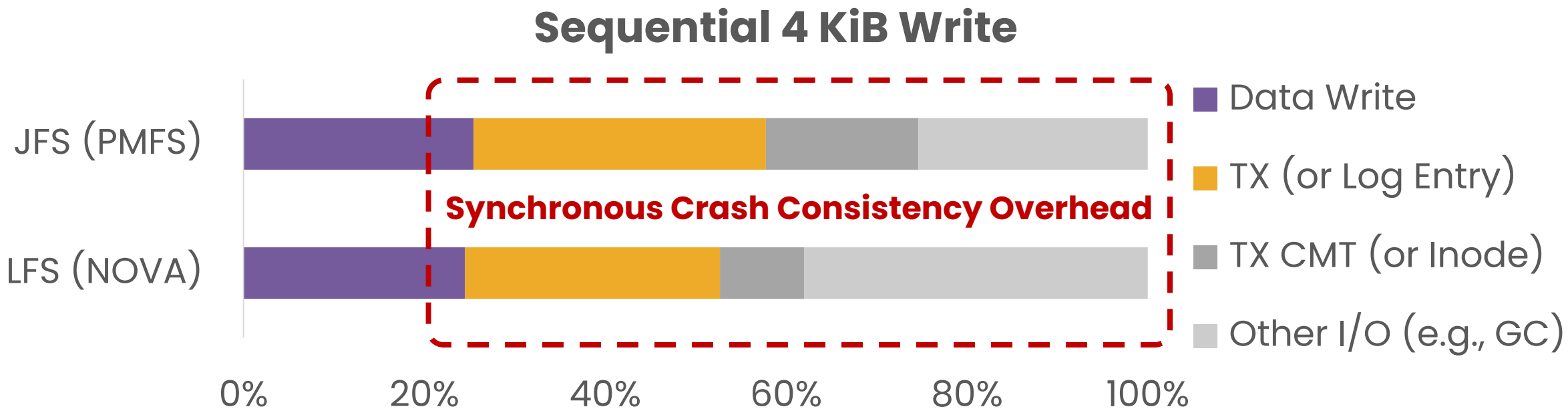
Two traditional methodologies are used for PM file systems

Do they perform well atop PM?

With Additional Writes
(e.g., Journaling File Systems: PMFS)

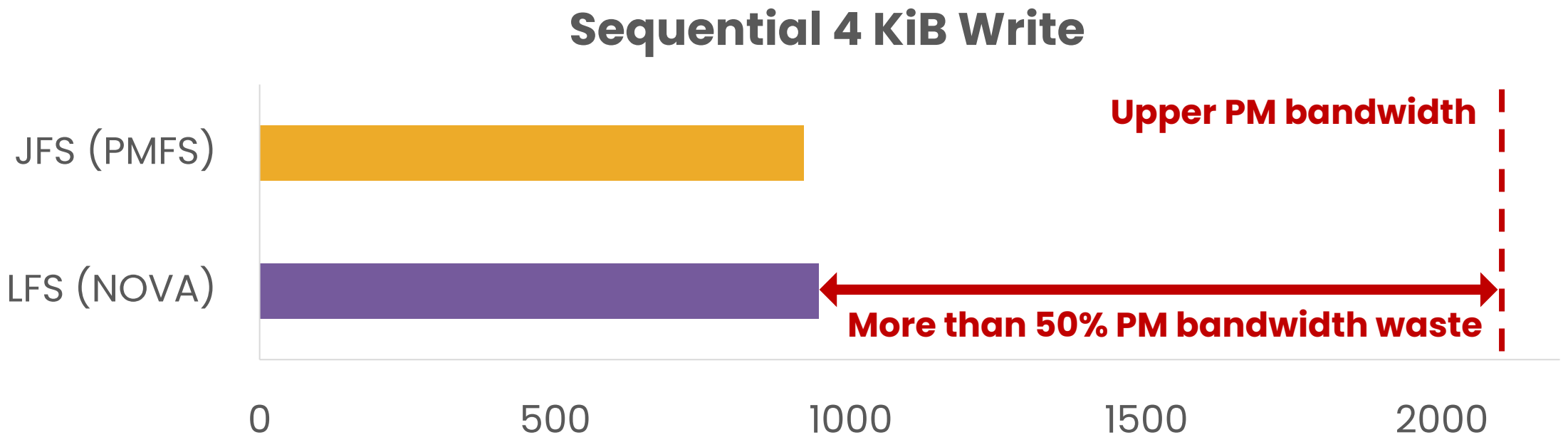
Without Additional Writes
(e.g., Log-structured File Systems: NOVA)

Crash Consistency Overhead on PM (1/2)



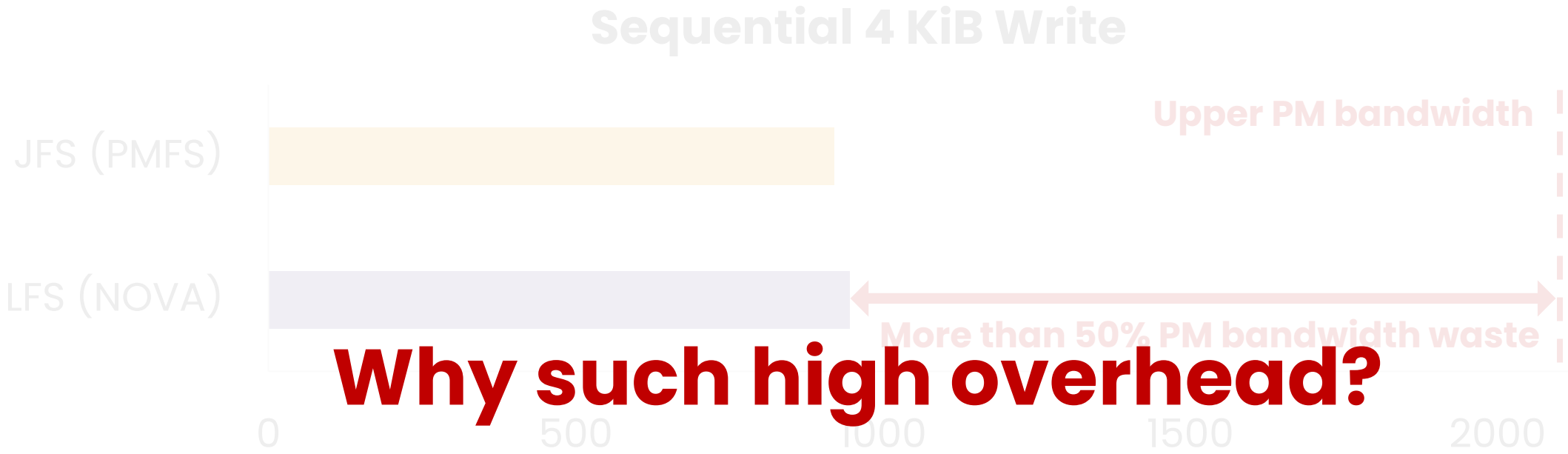
- **Additional Writes:** PMFS Journaling file system (JFS)
- **No Additional Writes:** NOVA Log-structured file system (LFS)
- **Takeaways:** Synchronous crash consistency overhead atop PM dominates **more than 75% overhead** of I/O path

Crash Consistency Overhead on PM (2/2)



- **Upper bound:** PM write BW is around 2.2 GiB/s in our machine
- **Results:** Existing synchronous crash consistency results in a more than 50% PM bandwidth waste.
- **Takeaways:** **Existing approaches do NOT suit well for PM.**

Crash Consistency Overhead on PM (2/2)

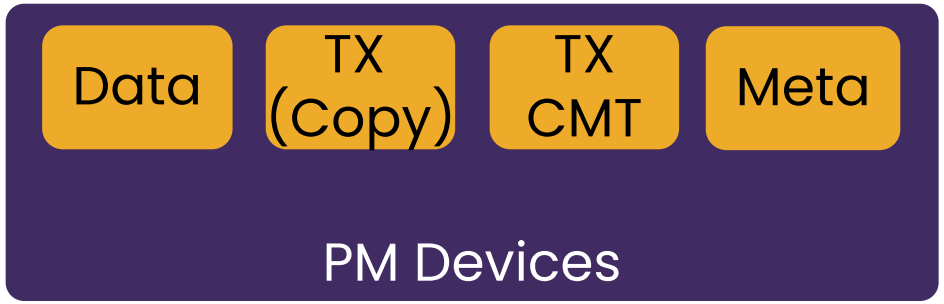


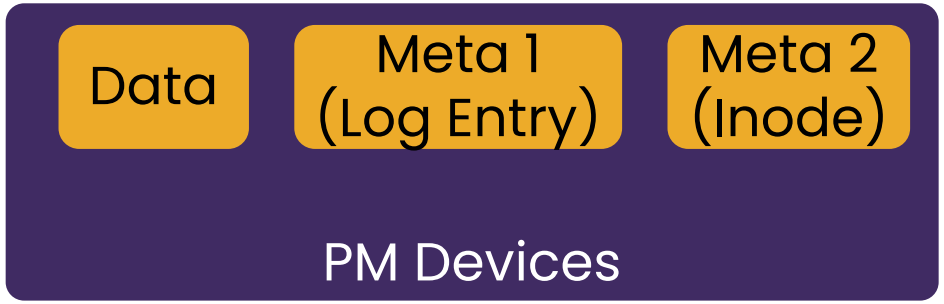
Why such high overhead?

- **Upper bound:** PM write BW is around 2.2 GiB/s in our machine
- **Results:** Existing synchronous crash consistency results in a more than 50% PM bandwidth waste.
- **Takeaways:** **Existing approaches do NOT suit well for PM.**

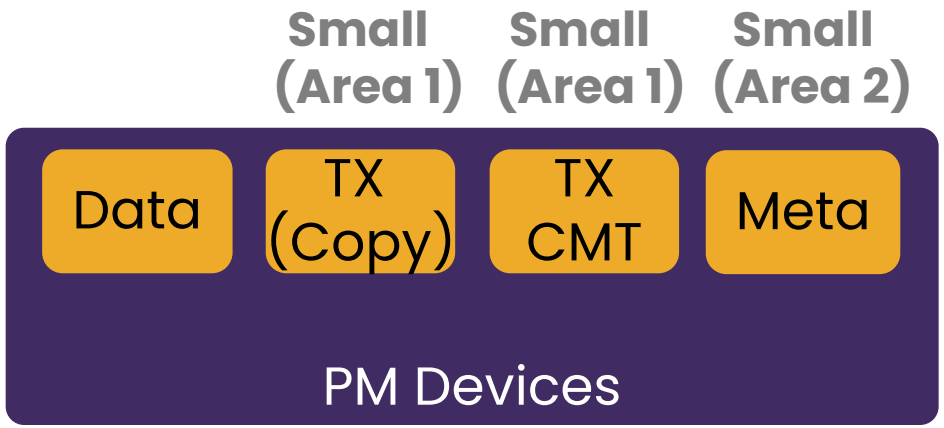
Deficiency Analysis

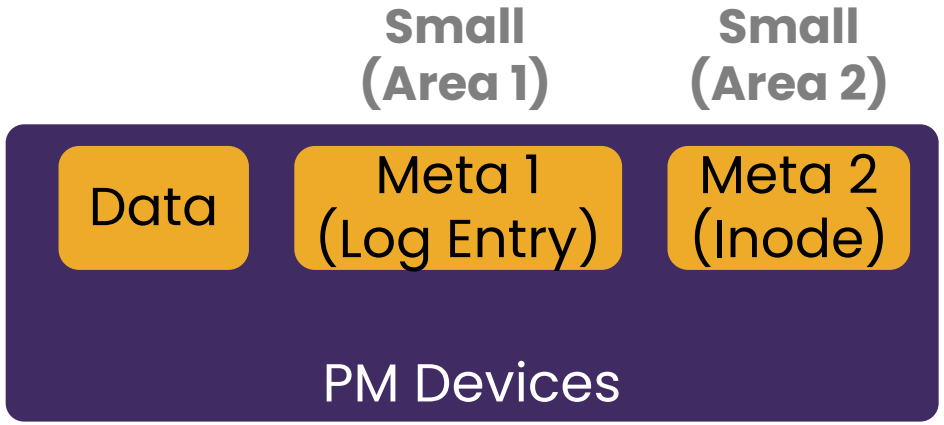
- **Root cause: Many small, random,** and **ordered** metadata I/O





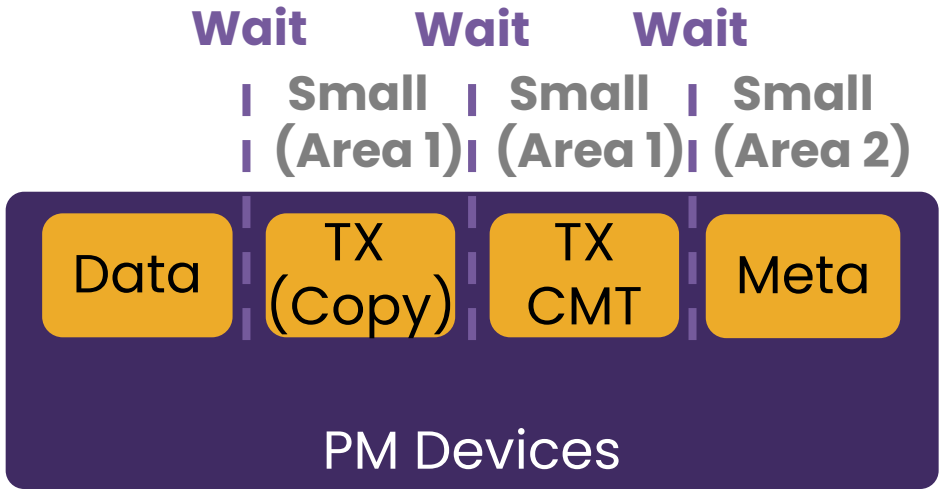
Deficiency Analysis

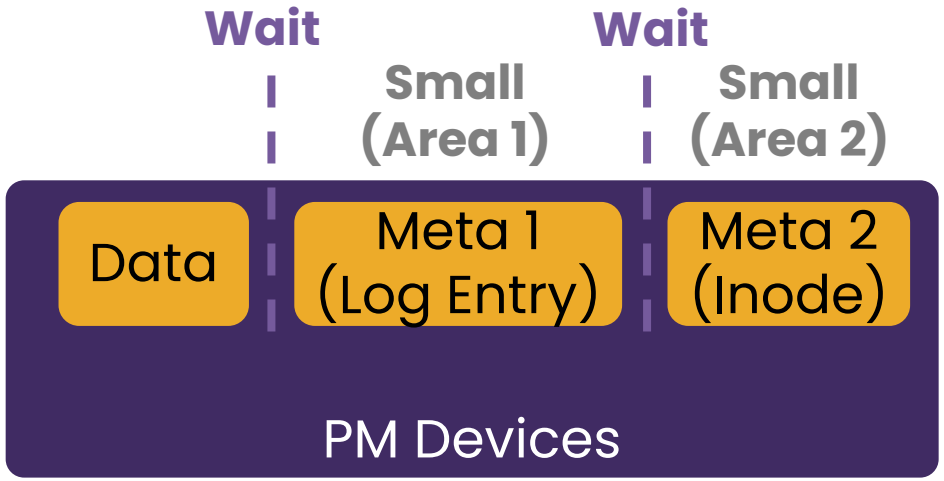




- **Root cause: Many small, random, and ordered** metadata I/O
- **Small & Random:** I/O amplification due to PM's coarse I/O granularity

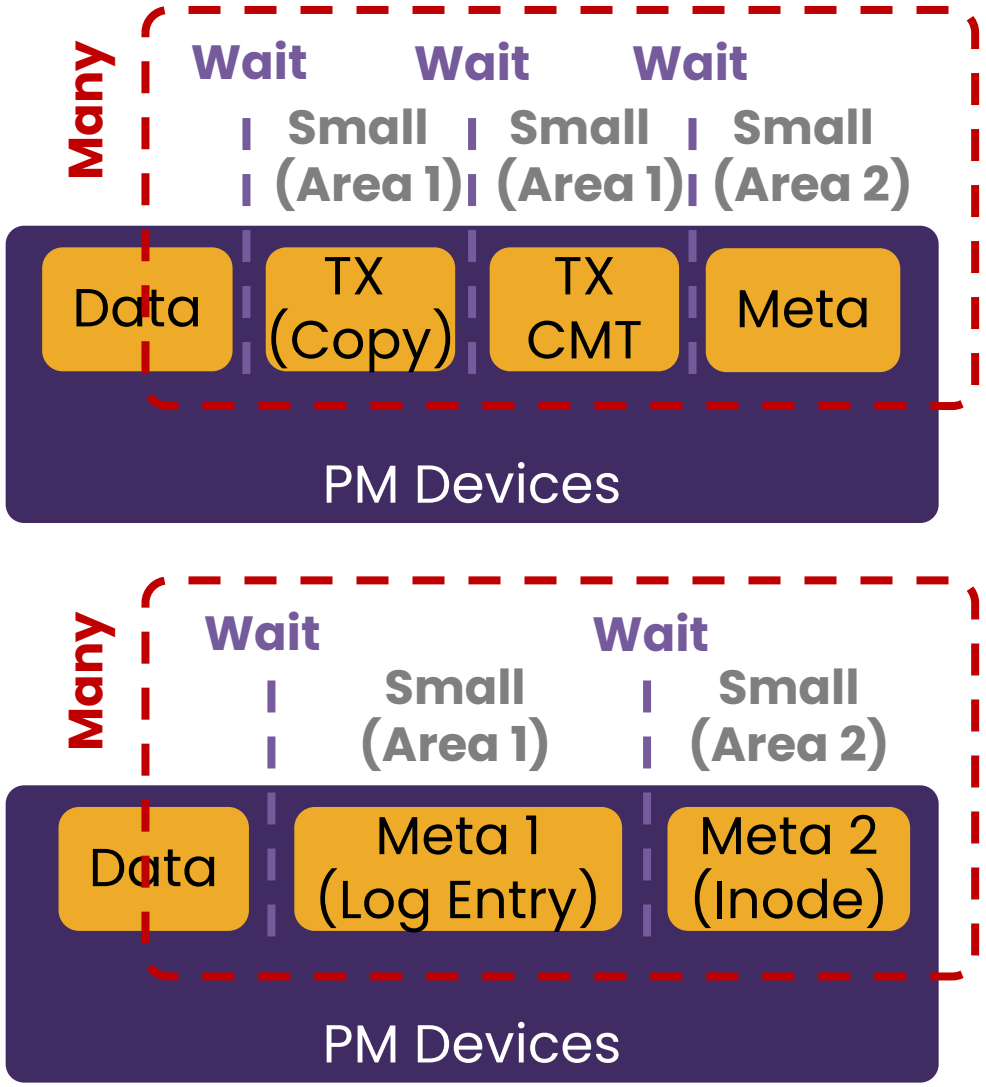
Deficiency Analysis





- **Root cause: Many small, random,** and **ordered** metadata I/O
- **Small & Random:** I/O amplification due to PM's coarse I/O granularity
- **Ordered:** Waiting for previous data transfer, limiting I/O concurrency

Deficiency Analysis

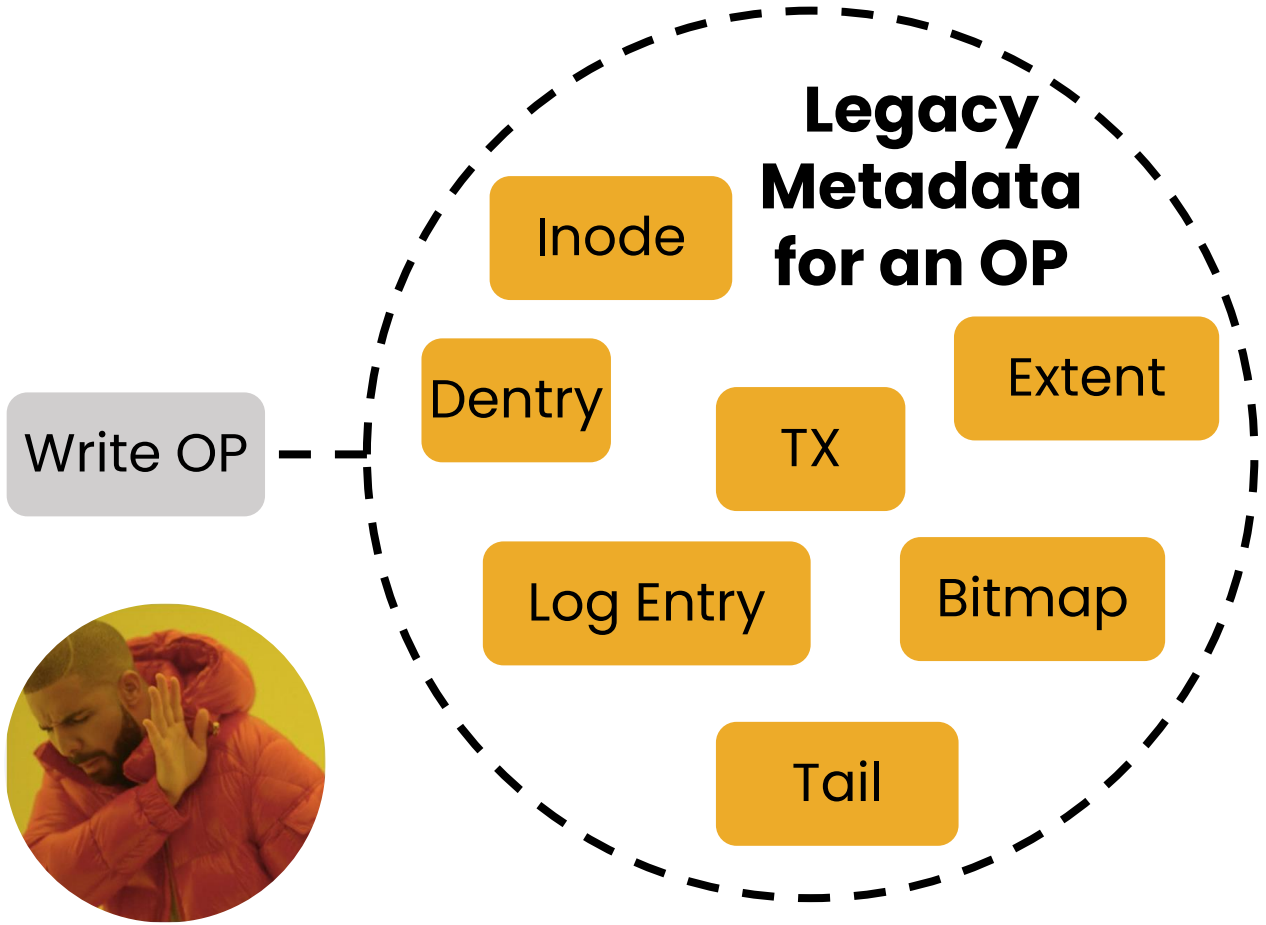


- **Root cause: Many small, random, and ordered** metadata I/O
- **Small & Random:** I/O amplification due to PM's coarse I/O granularity
- **Ordered:** Waiting for previous data transfer, limiting I/O concurrency
- **Many:** Further exacerbate the previous I/O overheads.

Our Goal

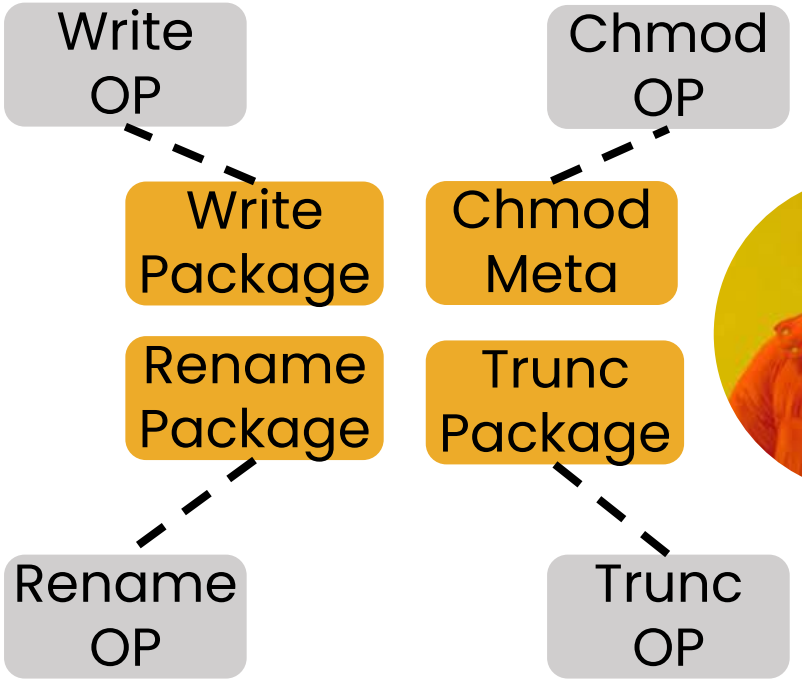
A new crash consistency mechanism to minimize metadata I/O and ordering points

Key Insight



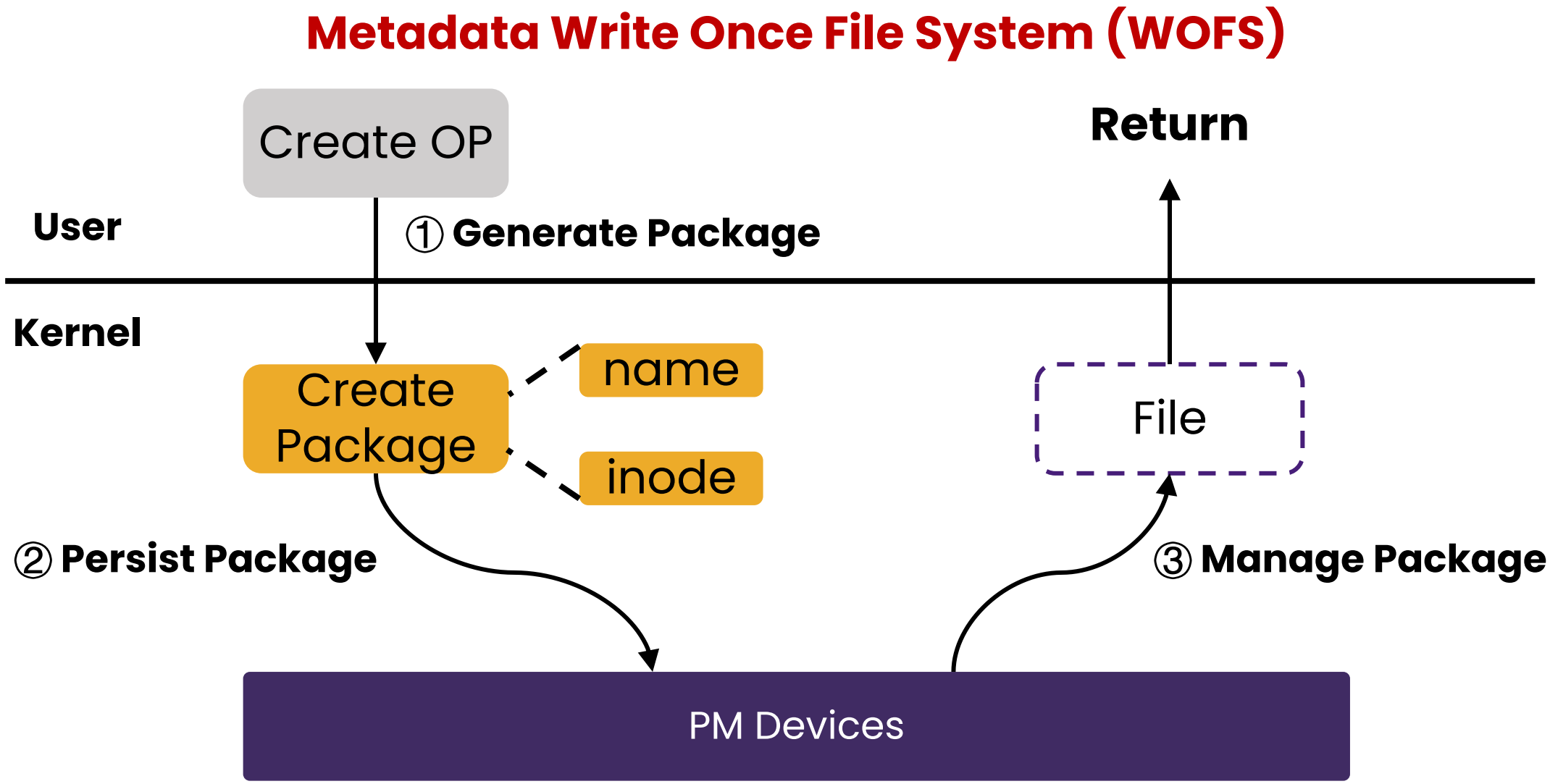
Struggle to orchestrate I/O for crash consistency

One package for one OP

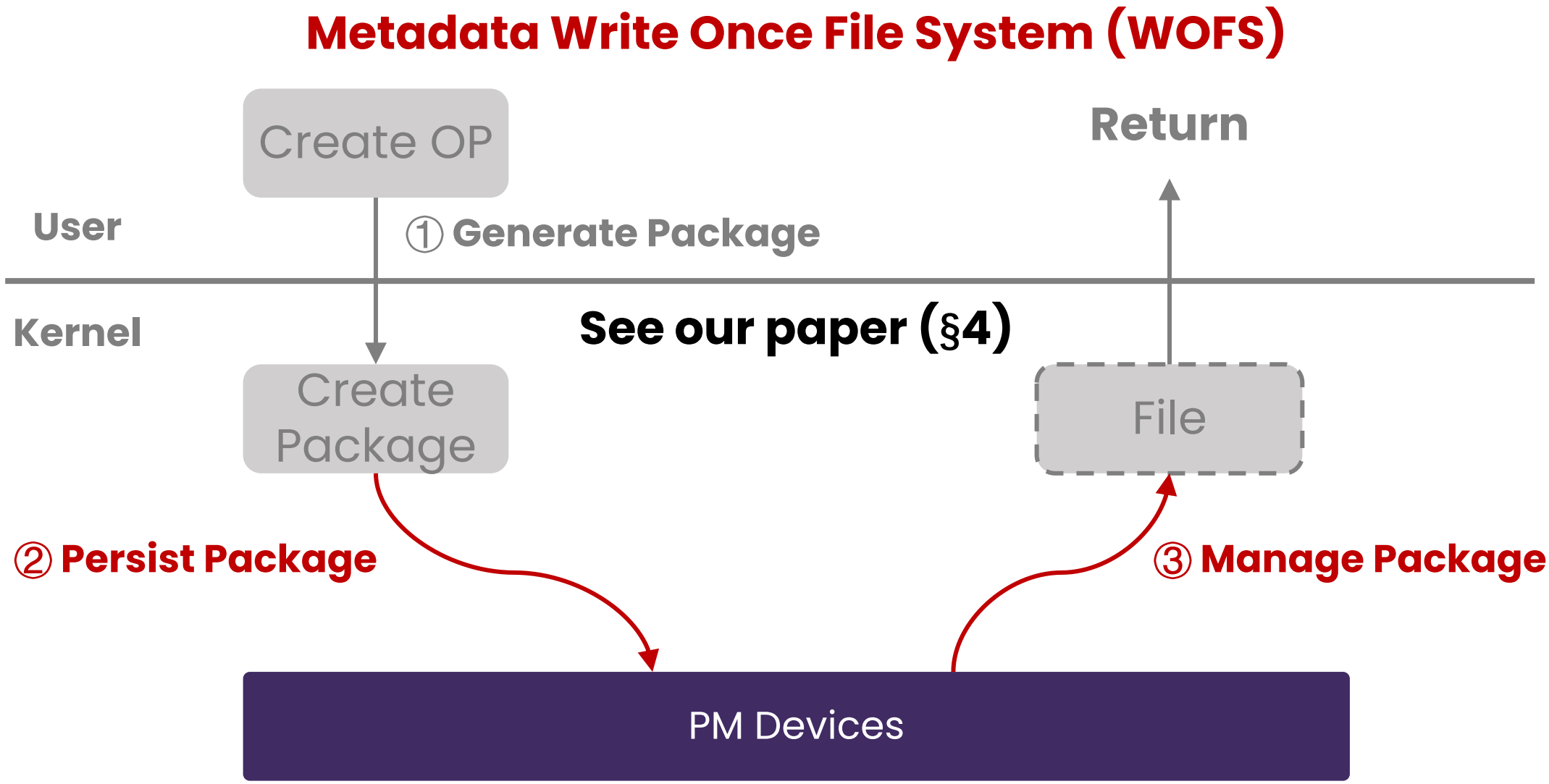


Rethink metadata for the optimal crash consistency

Intuitive Example: Create



Intuitive Example: Create



Package Persistence (1/2)

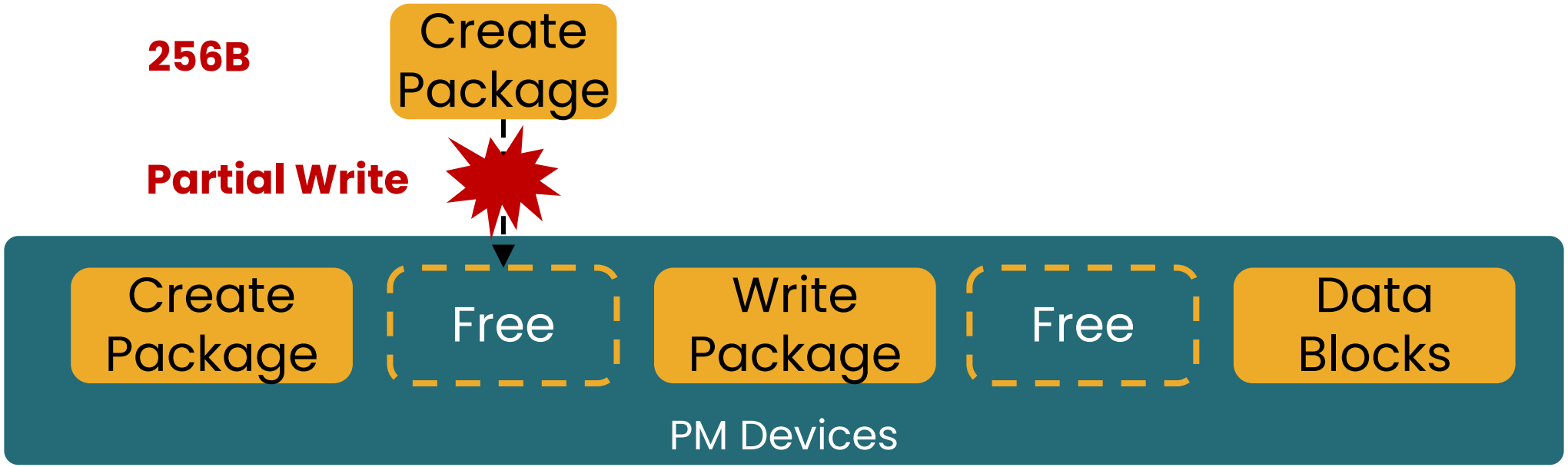
- **Where to persist a package?**
  - **Rationale:** Log is not a must; GC can be avoided

Package Persistence (1/2)

- **Where to persist a package?**
  - **Rationale:** Log is not a must; GC can be avoided
  - **Our approach:** **Non-log layout, using free lists for allocation**

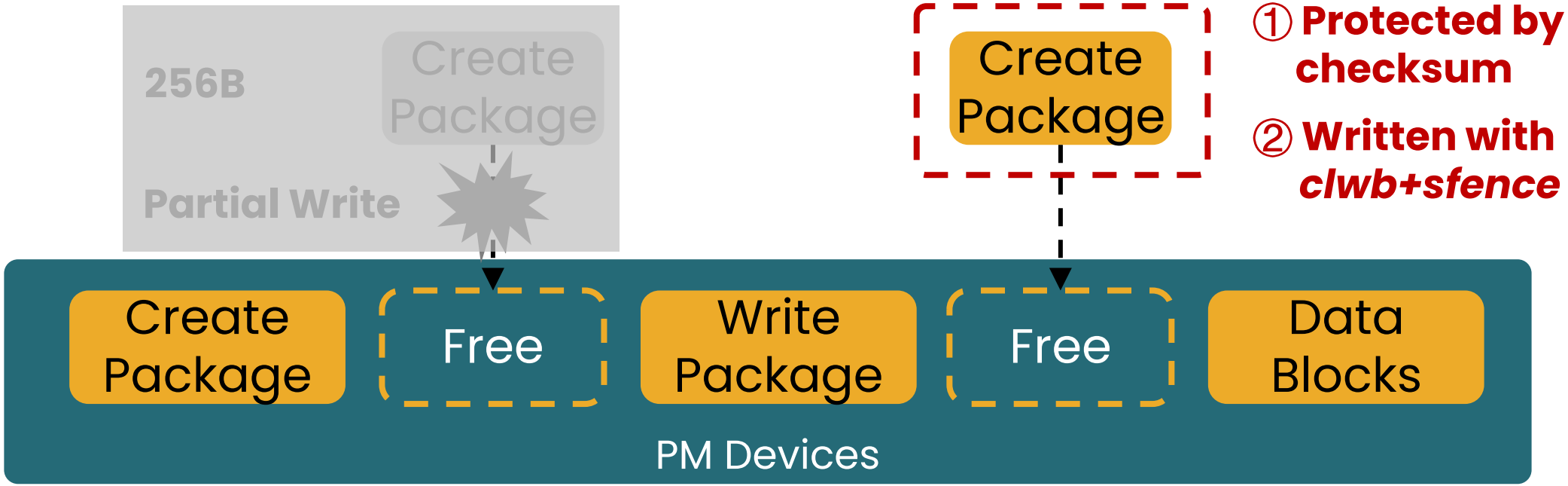
Package Persistence (2/2)

- **How to persist a package?**
  - **Problem:** The package can be large



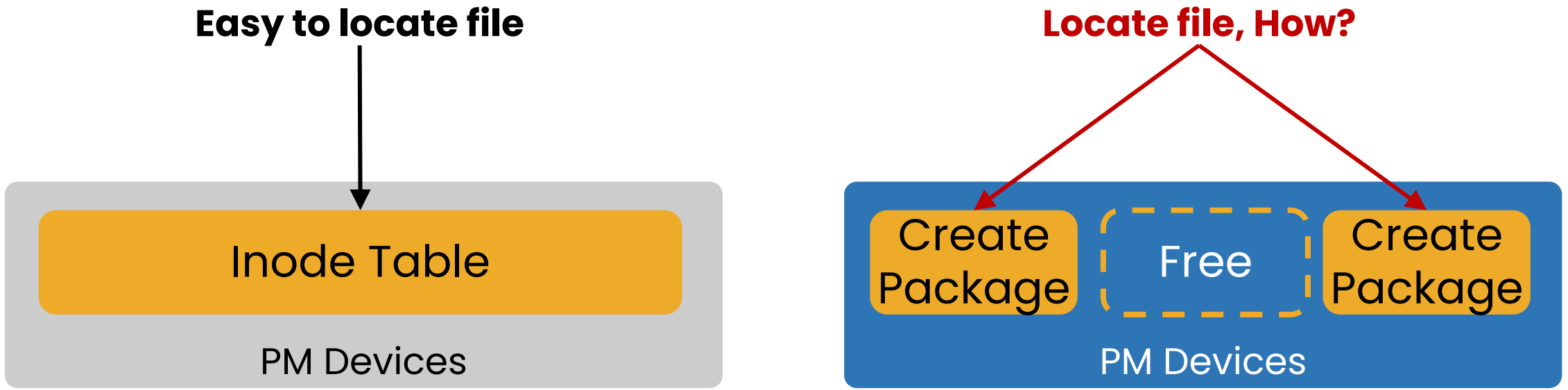
Package Persistence (2/2)

- **How to persist a package?**
  - **Problem:** The package can be large
  - **Our approach: Protect package with checksum**



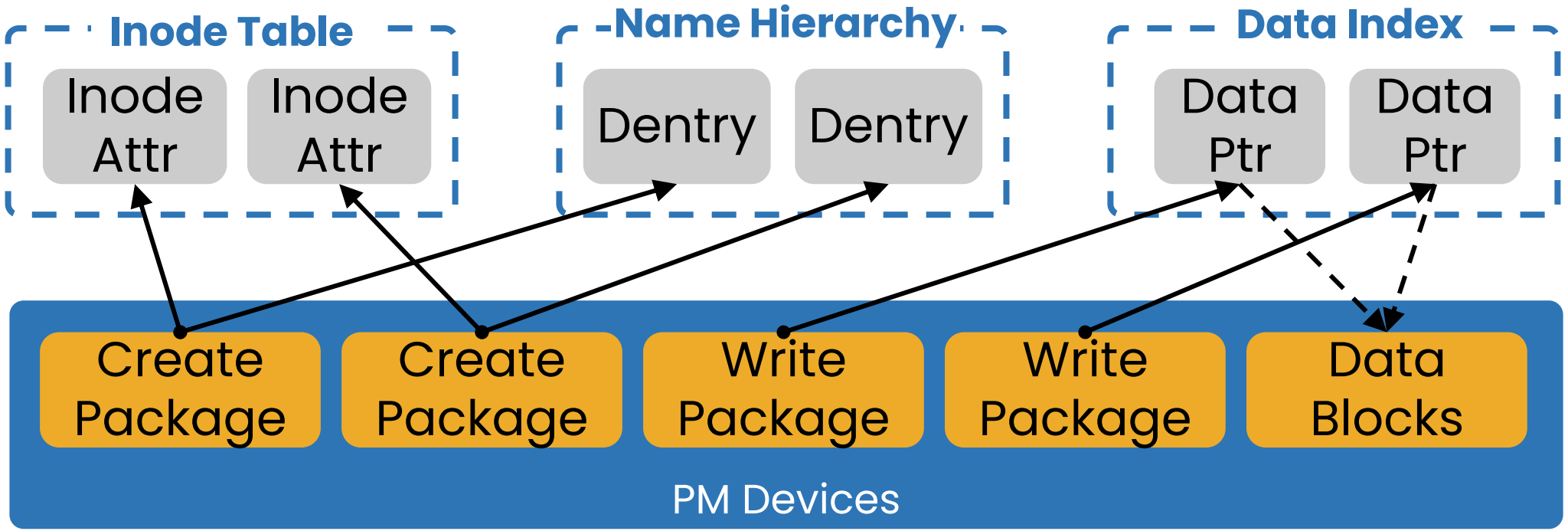
Package Management (1/2)

- **How to provide compatible services?**
  - **Problem:** Packages violate metadata objects



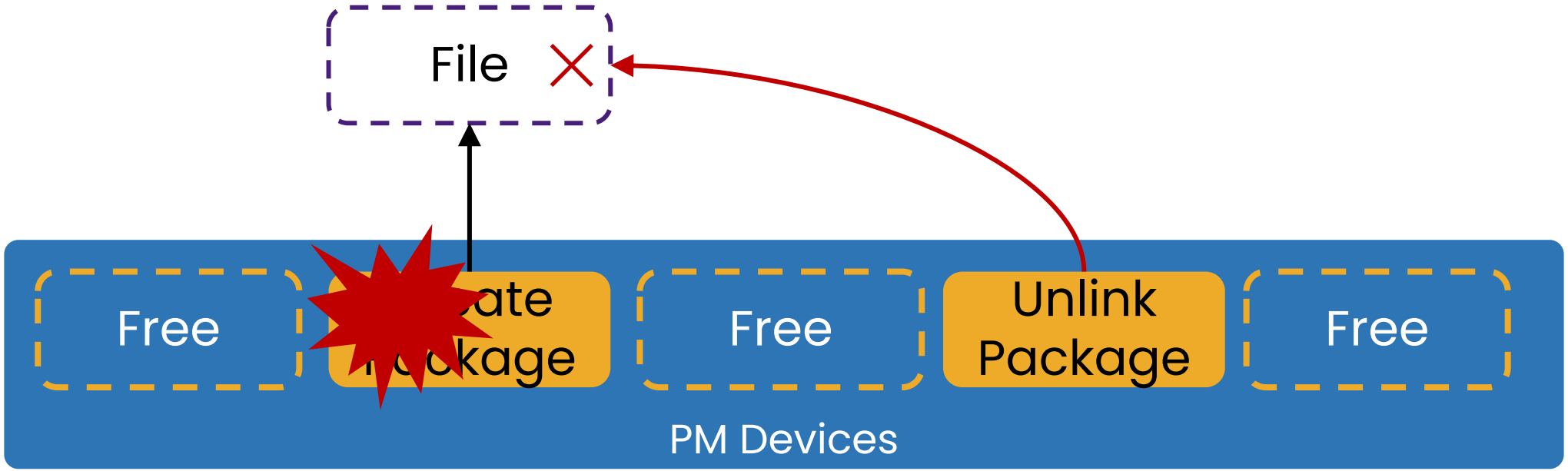
Package Management (1/2)

- **How to provide compatible services?**
  - **Problem:** Packages violate metadata objects
  - **Our approach:** **Package translation layer (PTL)**



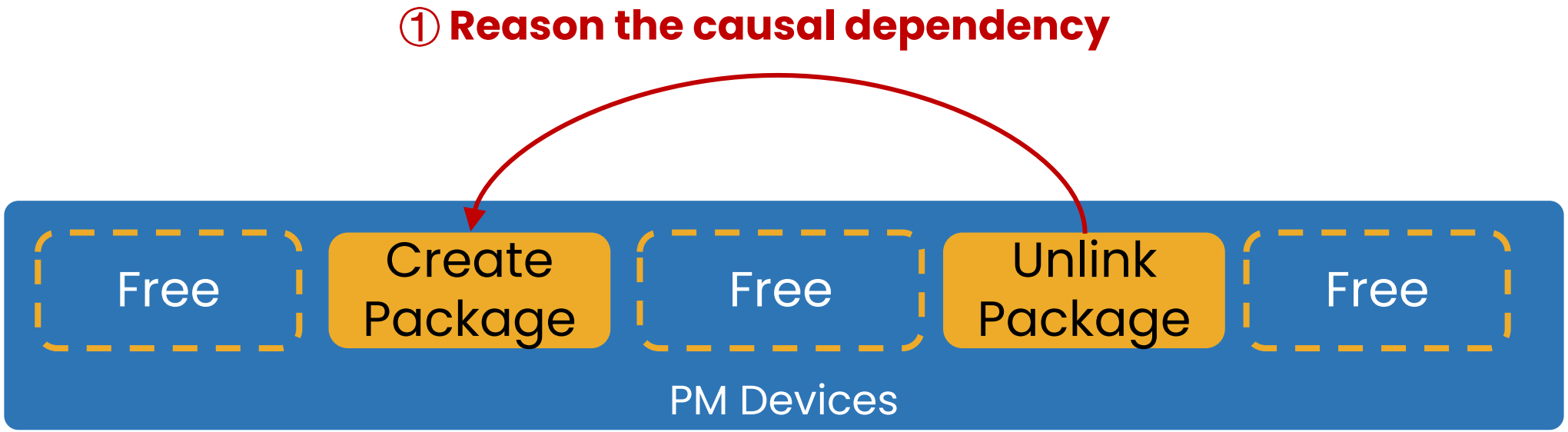
Package Management (2/2)

- **How to reclaim packages?**
  - **Problem:** Package can be invalidated



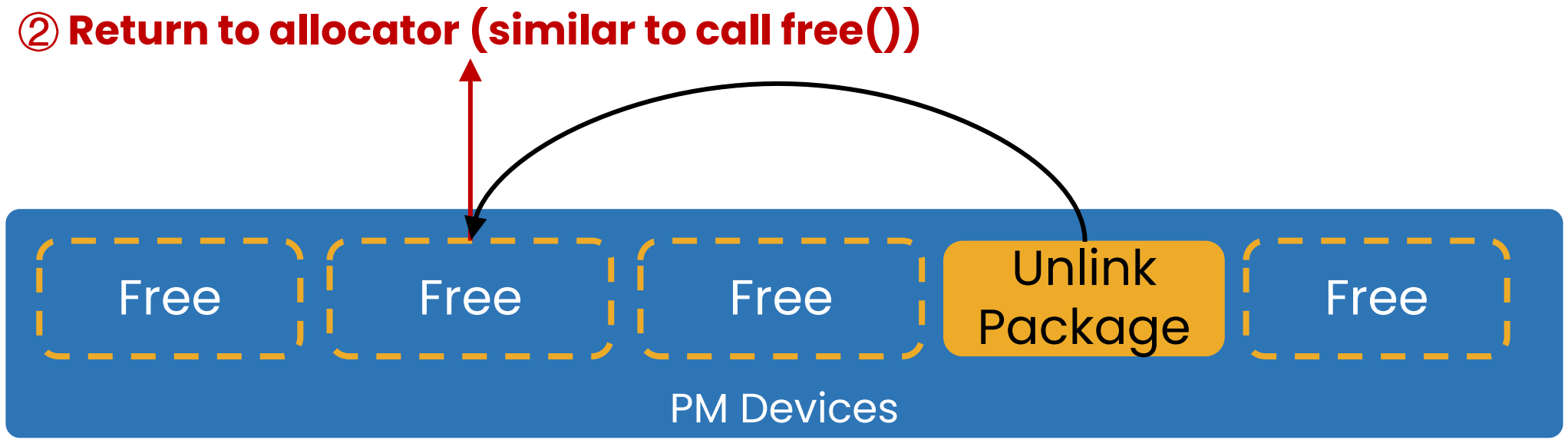
Package Management (2/2)

- **How to reclaim packages?**
  - **Problem:** Package can be invalidated
  - **Our approach:** **Immediate package reclamation**



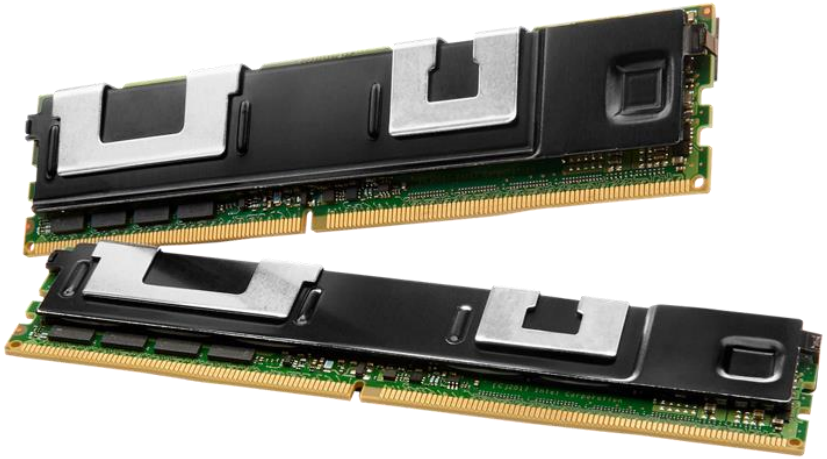
Package Management (2/2)

- **How to reclaim packages?**
  - **Problem:** Package can be invalidated
  - **Our approach:** **Immediate package reclamation**



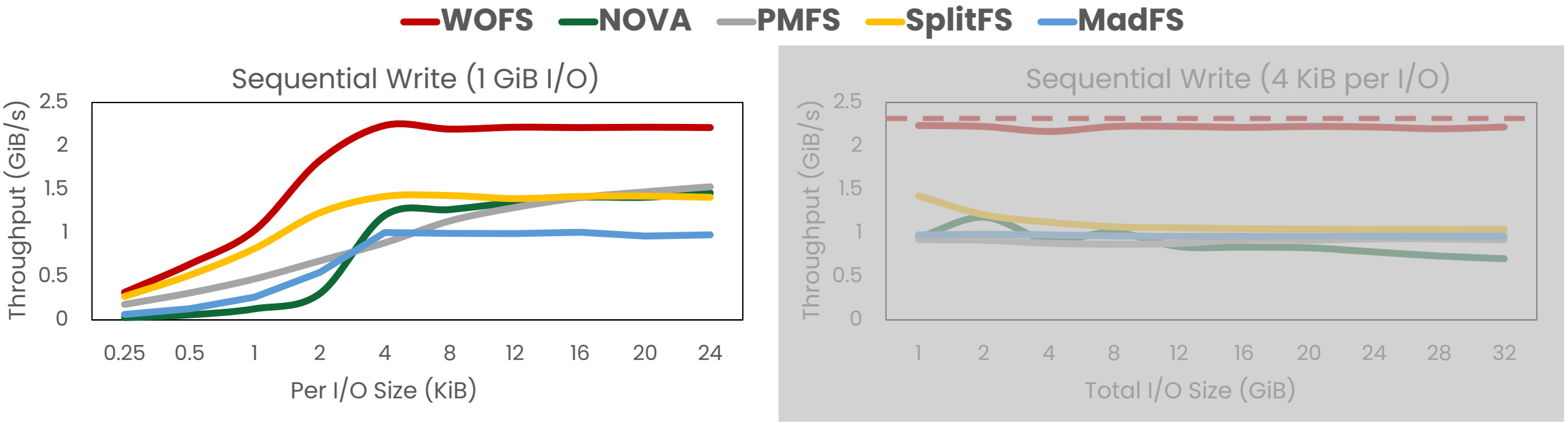
WOFS Evaluation

- **Experimental setup**
  - Implement WOFS in Linux kernel 5.1.0
  - Intel Xeon Gold 5218 CPU @ 2.3GHz
  - 256 GiB Optane DCPMM
  - 128 GiB DRAM



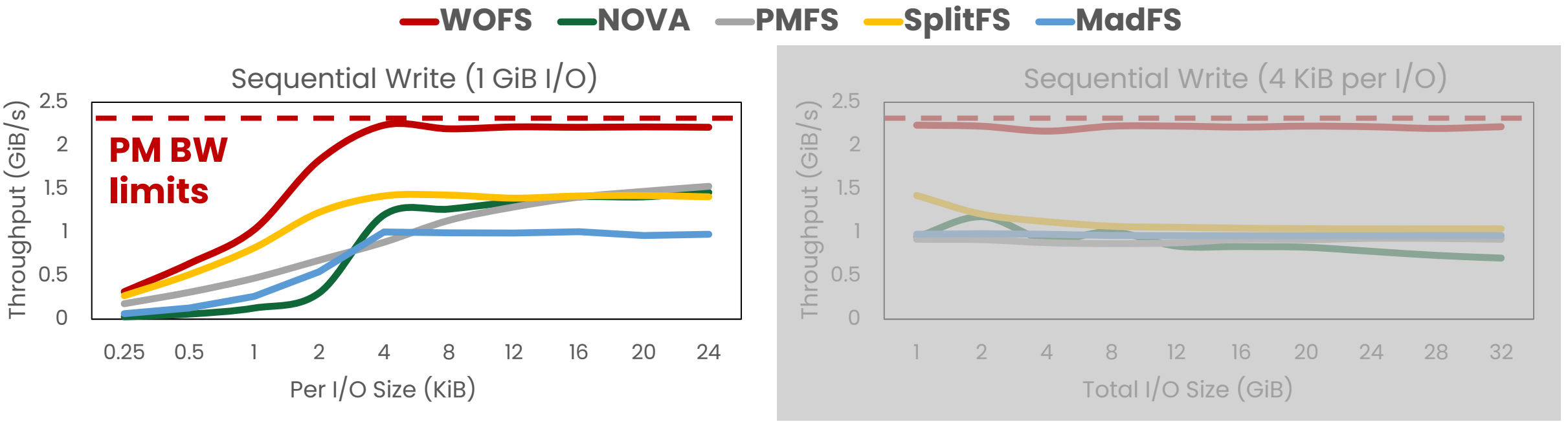
- **Competitors**
  - A number of existing PM file systems (e.g., NOVA, PMFS, SplitFS, etc.)

I/O Performance



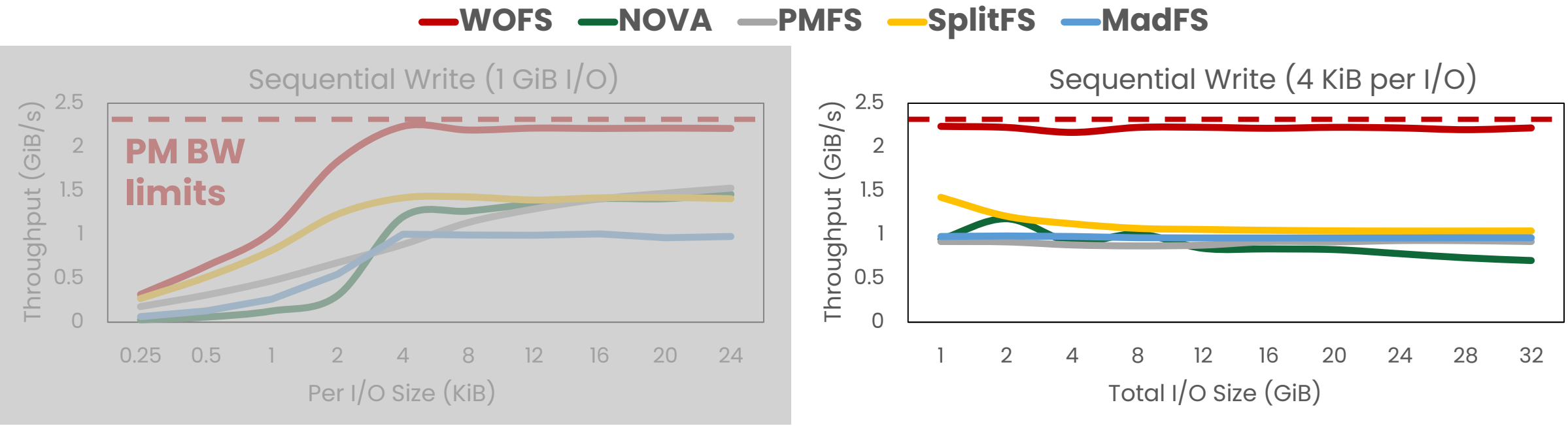
- **Setup:** FIO with 1 GiB total I/O size, varying per I/O size
- **Results:** WOFS consistently outperforms competitors thanks to the metadata write once scheme

I/O Performance



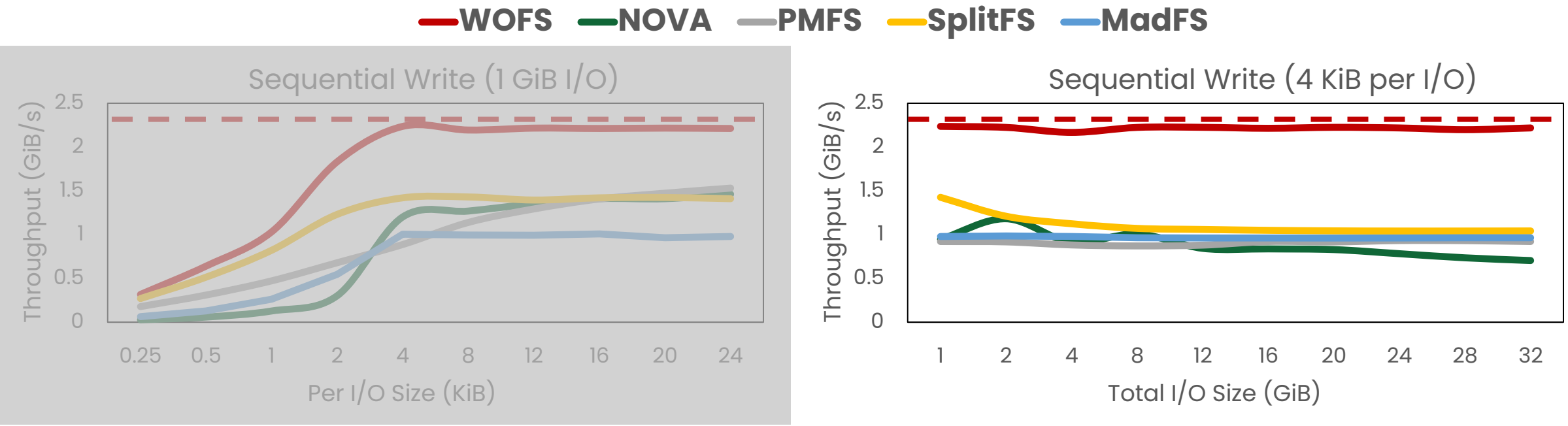
- **Setup:** FIO with 1 GiB total I/O size, varying per I/O size
- **Results:** WOFS consistently outperforms competitors thanks to the metadata write once scheme
- **Important NOTE:** WOFS can reach PM bandwidth limits

I/O Performance



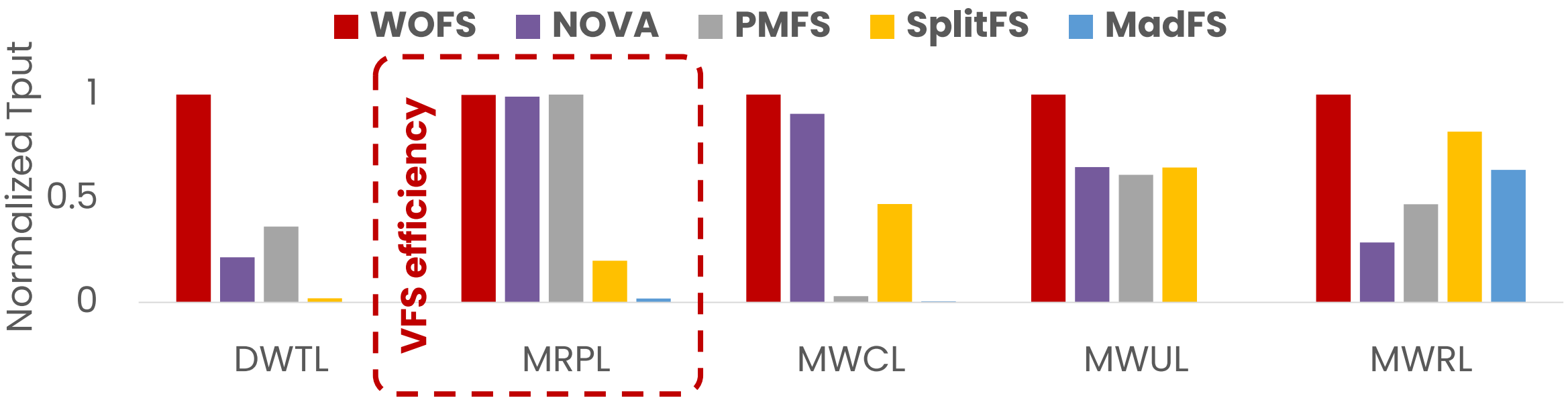
- **Setup:** FIO with 4 KiB per I/O, varying total I/O size
- **Results:** Similarly, WOFS outperforms competitors by more than 50%, very close to upper bandwidth limits

I/O Performance



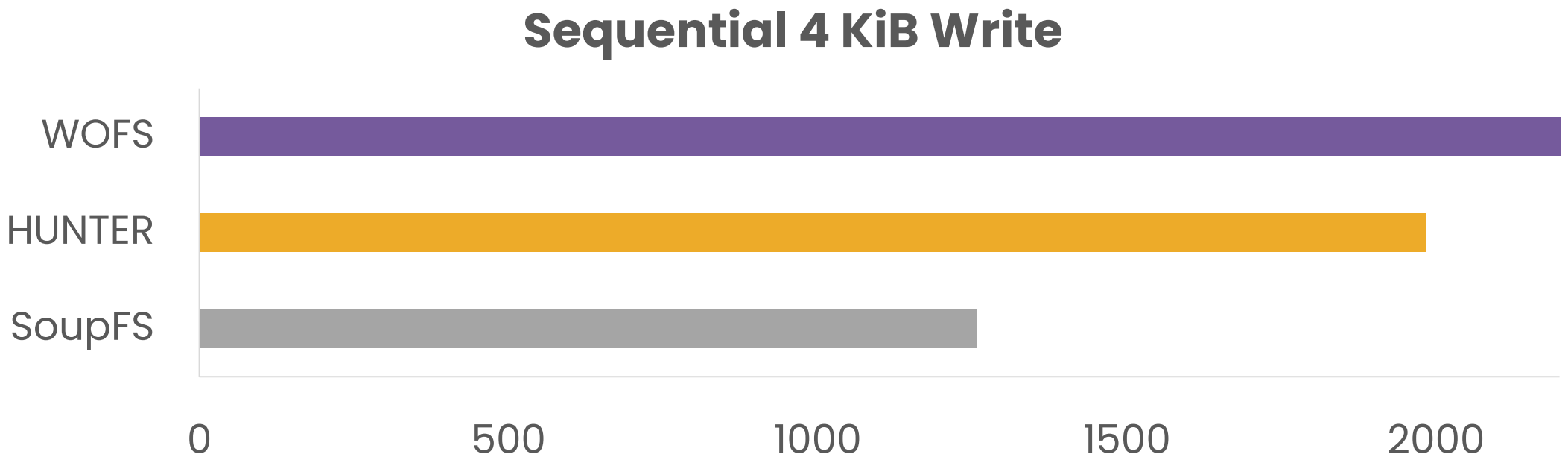
- **Setup:** FIO with 4 KiB per I/O, varying total I/O size
- **Results:** Similarly, WOFS outperforms competitors by more than 50%, very close to upper bandwidth limits
- **Important NOTE:** WOFS shows a stable performance trend

Metadata Performance



- **Setup:** FxMark with five single-threaded workloads, including data operation, dir traverse, file create/deletion, and rename
- **Results:** WOFS always shows the highest throughput as it minimizes metadata I/O and ordering points.

Vs. Asynchronous Crash Consistency



- **Setup:** FIO with 4 KiB per I/O, no fsync issued
- **Competitors with asynchronous crash consistency:** HUNTER and SoupFS both delay metadata writes to the background
- **Results:** WOFS still outperforms others, as latter's background flush interference with their foreground I/O performance

Other Results

- **Other Results**
  - Concurrency and tail latency
  - Real-world workload evaluation
  - Recovery overhead
  - Aging and fragmentation
  - Performance beyond Optane DCPMM
  - etc.

Please refer to our paper (§6)

Conclusion

- **Analysis of synchronous crash consistency overhead atop PM**
  - Many small, random and ordered metadata I/O can be the bottleneck
- **WOFS model to minimize crash consistency overhead**
  - Rethink file system metadata for optimal crash consistency
  - Design a specific package as one metadata for a single operation
  - Metadata write-once can minimize I/O and ordering points for crash consistency
- **Make WOFS architecture practical and efficient**
  - Propose a range of techniques to manage packages efficiently
  - Results. Outperform SOTA PM file systems, reaching upper PM bandwidth limits

github.com/WOFS-for-PM/

Thanks & QA