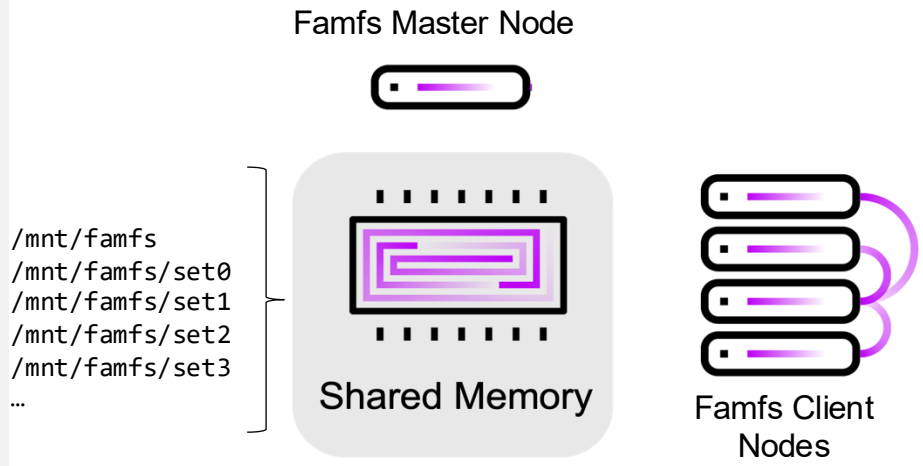
Famfs: open source scale-out shared memory file system

John Groves
Technical Director
And Co-Chair of the CXL Software and Systems Working Group
https://famfs.org
June 2026

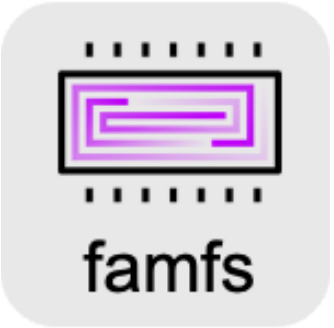
Famfs Is a Scale-Out File System for Disaggregated Shared Memory

Enabling shared JBOM for all apps that can use files

- Memory is accessible as files
  - Write/read become memcpy
  - Mmap provides byte / cache-line access
- “All” apps can access data in files
- Famfs files are memory and not storage
  - Move data into famfs for in-memory access
  - Move data out of famfs to store persistently
- Posix permissions apply, along with strict partitioning of data from separate files
- Orchestration layers such as PNFS could use famfs as a tier – providing memory performance + scale-out sharing




```
mkfs.famfs /dev/dax0.0
famfs mount /dev/dax0.0 /mnt/famfs
famfs cp [-r] <src> <dest>
famfs creat –s <size> <dest>
```



Famfs: The Core Insight

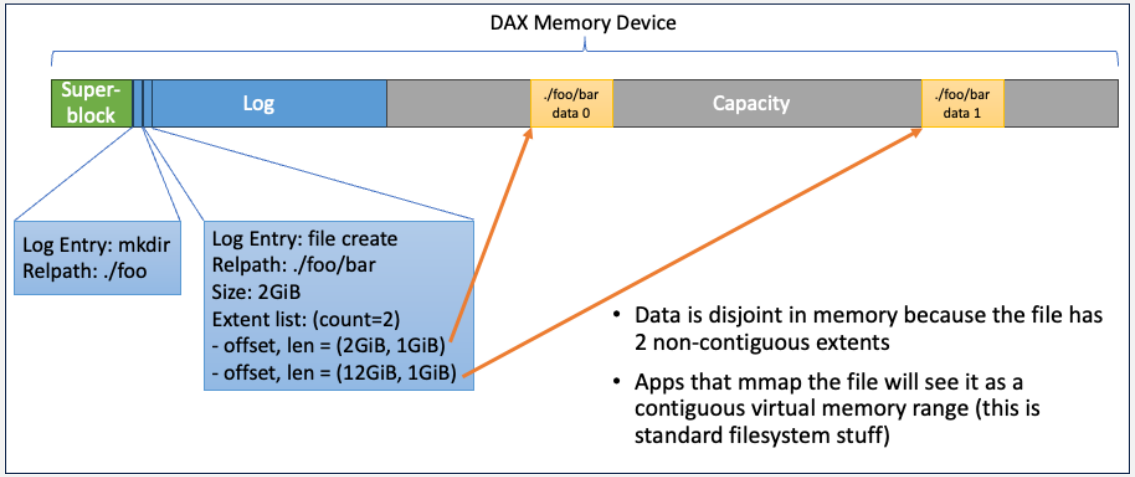
- Sharable memory needs a standard access method
  - Linux has no concept of memory that isn't wholly owned

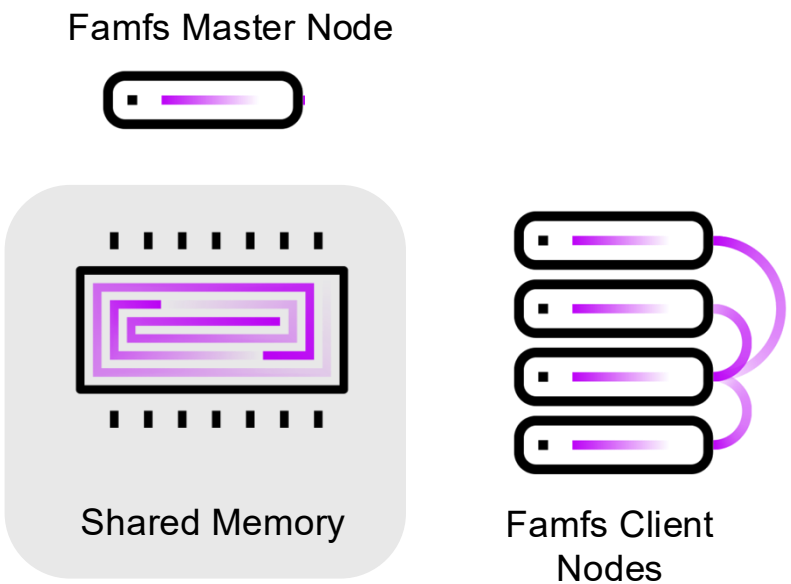
- The file system is the natural abstraction for shared memory
  - No fundamental new abstractions required
  - Software already understands files!!
  - Posix permissions apply, etc.

- Prior proposals to enable of shared memory might be paraphrased as "It's a new paradigm, requiring new abstractions!" – not necessarily true…
  - See HP's "The Machine"

Famfs architecture (MVP)

- Metadata is stored in an append-only log
- Log is written by Master and ”played” by Clients
- V1 handles clients with stale metadata by not supporting truncate or delete
- Metadata handled in user space (library, cli)
- Read / write / mmap / vma faults handled in kernel
- Memory mapping from famfs == cache-line level access to shared mem
- Many of the limitations can be addressed in future versions






```
mkfs.famfs /dev/dax0.0
famfs mount /dev/dax0.0 /mnt/famfs
famfs cp [-r] <src> <dest>
famfs creat -s <size> <dest>
```

Why Famfs?
100TB – 1PB Graph DBs are possible and practical

Via adding disaggregated memory decoupled from compute

100TB Graph DB cluster
18 servers, 8TB/server
400G Infiniband network

- Perf 1X
- Baseline cost[2]

3 35TB JBOM cluster
1 server, 4TB/server

- 6-10X[1] higher perf
- 37% lower cost than baseline

3 35TB JBOM cluster
4 server, 4TB/server

- 24-40X higher perf or throughput
- 23% lower cost than baseline

24X to 40X

performance and throughput leap

Speed-up due to no sharding

32X to 54X Perf/$

performance / cost

Reduction of TCO by 97% for the same performance

100TB to 1PB all in-mem

Unprecedented graph size

Enabled by the CXL memory chassis; adding scale

1. Source: “A Unified Graph Framework for Storage-Compute Coupled Cluster and High-Density Computing Cluster”, Lynsey Lin, Jamie Chen, ..
2. Baseline cost includes only hardware cost, not software licensing, which could be more significant

We Have a Paper in IEEE Transactions on Computers

April 2026

IEEE TRANSACTIONS ON COMPUTERS, VOL. 75, NO. 4, APRIL 2026 1291

No Atomics, No Problem. Developing a RAG Pipeline for Shared CXL Memory

Alfred Bratterud, *Member, IEEE*, Gisle Dankel, Amin Farajianzadeh, John Groves, Chengyi Juan, Joshua Suetterlein, Andrés Márquez, Petter Gustad, *Member, IEEE*, and Arnt Emil Ingulstad

***Abstract*—We share early experiments with software development for shared CXL memory on the H3 Falcon C5022 CXL switch. We describe how the different stages in the development of a commercial RAG pipeline were impacted by the presence of CXL memory. All of Wikipedia is split into 46 million text passages which are embedded into a high-dimensional embedding space and subjected to heavy load in single- and multi-host experiments. We observe significant performance advantages available to DuckDB and Faiss without changing the software; we measure up to 12x latency reduction with real workloads and 40x reduction with synthetic workloads on CXL, compared to the same queries run with demand paging on NVMe. We explain the current challenges of working with two disjoint cache coherency domains and explain how the Fabric-Attached Memory File System (famfs) and famfs producer-consumer queues provide synchronization patterns without atomic operations on current x86 CPUs. We then discuss open challenges remaining for high performance atomics and locking mechanisms in shared memory. Lastly we show how famfs page-level interleaving enables near-linear throughput scaling when a second host serves queries from the same Faiss index on shared fabric-attached memory and how shared memory allocations can be orchestrated by Kubernetes in a full vertical RAG deployment.**

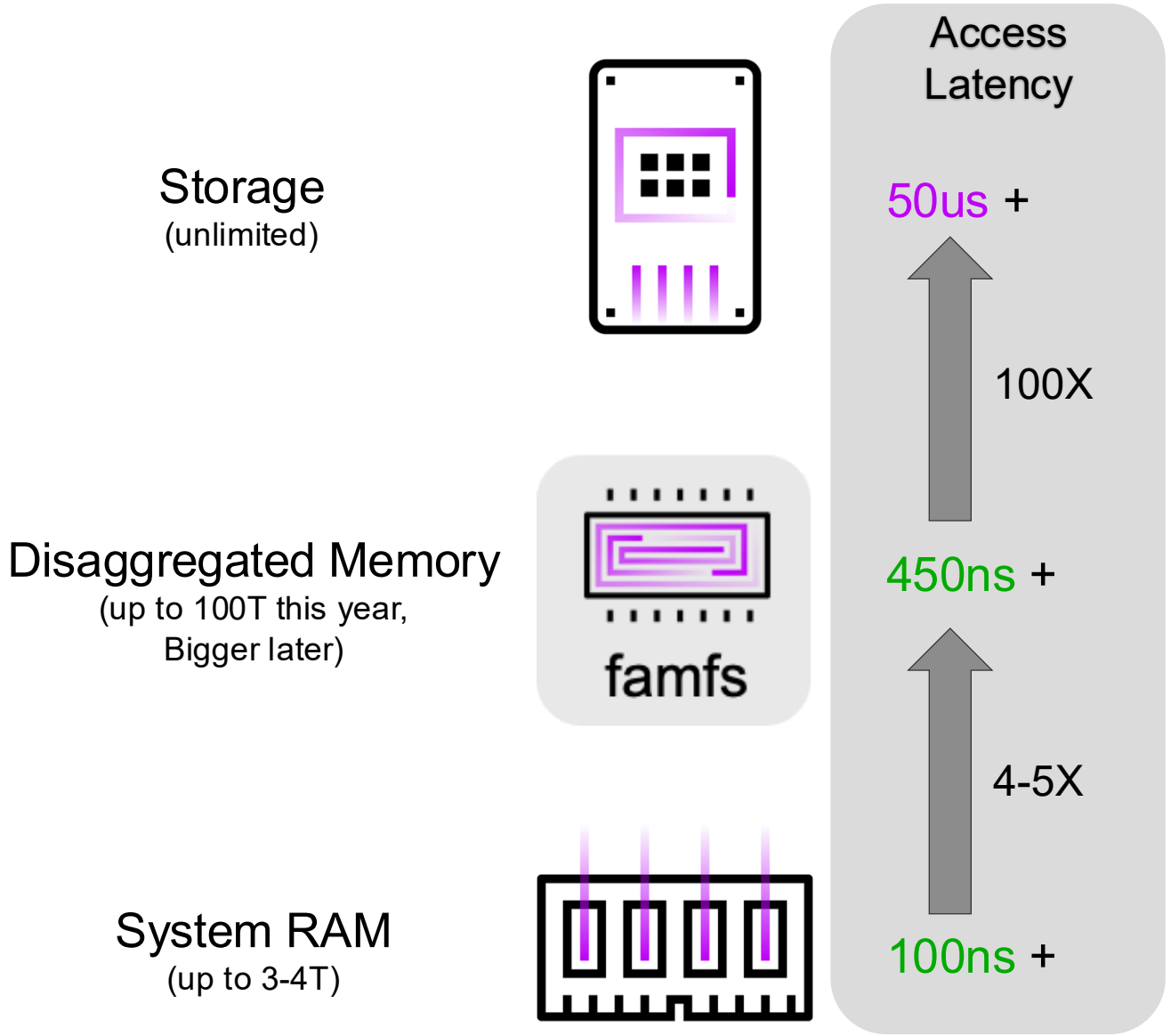
***Index Terms*—CXL, RAG, systems software.**

memory-intensive applications while showing the practical benefits of technology available today. Compute Express Link (CXL) is an evolving standard for load/store semantics over PCIe, overcoming the physical limitations of a single memory bus and enabling unprecedented amounts of memory to be accessed and shared between physical hosts via CXL switches. Early research on this technology shows that extending the memory capacity of individual hosts can greatly improve overall memory utilization for data centers at the cost of ~100ns additional latency [1]. Previous works primarily consider *memory pooling*, where individual CXL memory allocations are used by one physical host at the time. In such a scenario the additional memory is exposed to the kernel as a CPU-less NUMA node. In this work we consider the effects of adding a disaggregated memory pool large enough to hold the entire dataset used by several stages of a commercial software development process, with special emphasis on how CXL enables sharing of memory between hosts, where onlining shared memory as system RAM managed by the kernel is not feasible. The two main challenges this paradigm currently presents to software development are:

- Allocation of shared memory must be coordinated across

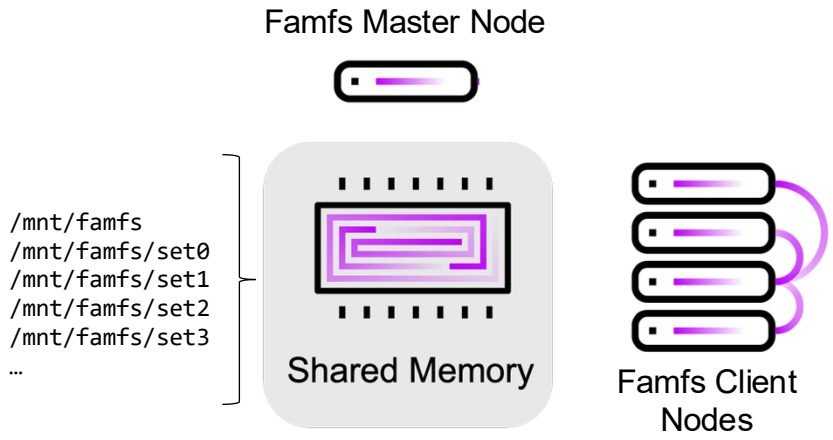
The superpower of memory is low-latency random access

- Memory access latency is much lower than storage latency
- Compare disaggregated memory to storage, not system-ram
- Data that doesn't fit in System-RAM can be random-accessed in disaggregated memory 100x faster than storage



Famfs: The Journey Into Upstream Linux Has been, uh, interesting

- Nov 2023 – Introduced famfs at the Linux Plumbers Conference
- Spring 2025 – Famfs Linux patch sets released (v1, v2)
- May 2025 – Famfs session at LSFMM
  (Linux Storage, File System and Memory Management summit)
  - Conclusion: I was asked to try merging famfs into fuse
- Aug 2024 – Famfs adds interleaved file support
- 2024 onward – Famfs in pilot use at CERN, Alibaba, Intel, Universities, etc.
- Linux Plumbers Sessions 2023, 2024, 2025
- LSFMM Sessions 2024, 2025, 2026
- April 2026: about half of famfs was accepted into upstream Linux
- June 2026: uncertainty about whether famfs will go upstream as a FUSE file system or standalone




```
mkfs.famfs /dev/dax0.0
famfs mount /dev/dax0.0 /mnt/famfs
famfs cp [-r] <src> <dest>
famfs creat -s <size> <dest>
```

Two Versions of Famfs Currently

- Standalone and FUSE-based
- At LSFMM 2024, the Linux file system brain trust asked me to try a port into FUSE
- Fuse port was hard, took a year – released around LSFMM 2025
- As of March 2026, no substantive review from the FUSE maintainers
  - Since March, it's a bit of a <spitshow>

Standalone famfs

- Kernel component resembles RAMFS (but for DAX memory)
- Great performance
- Metadata handled in user space

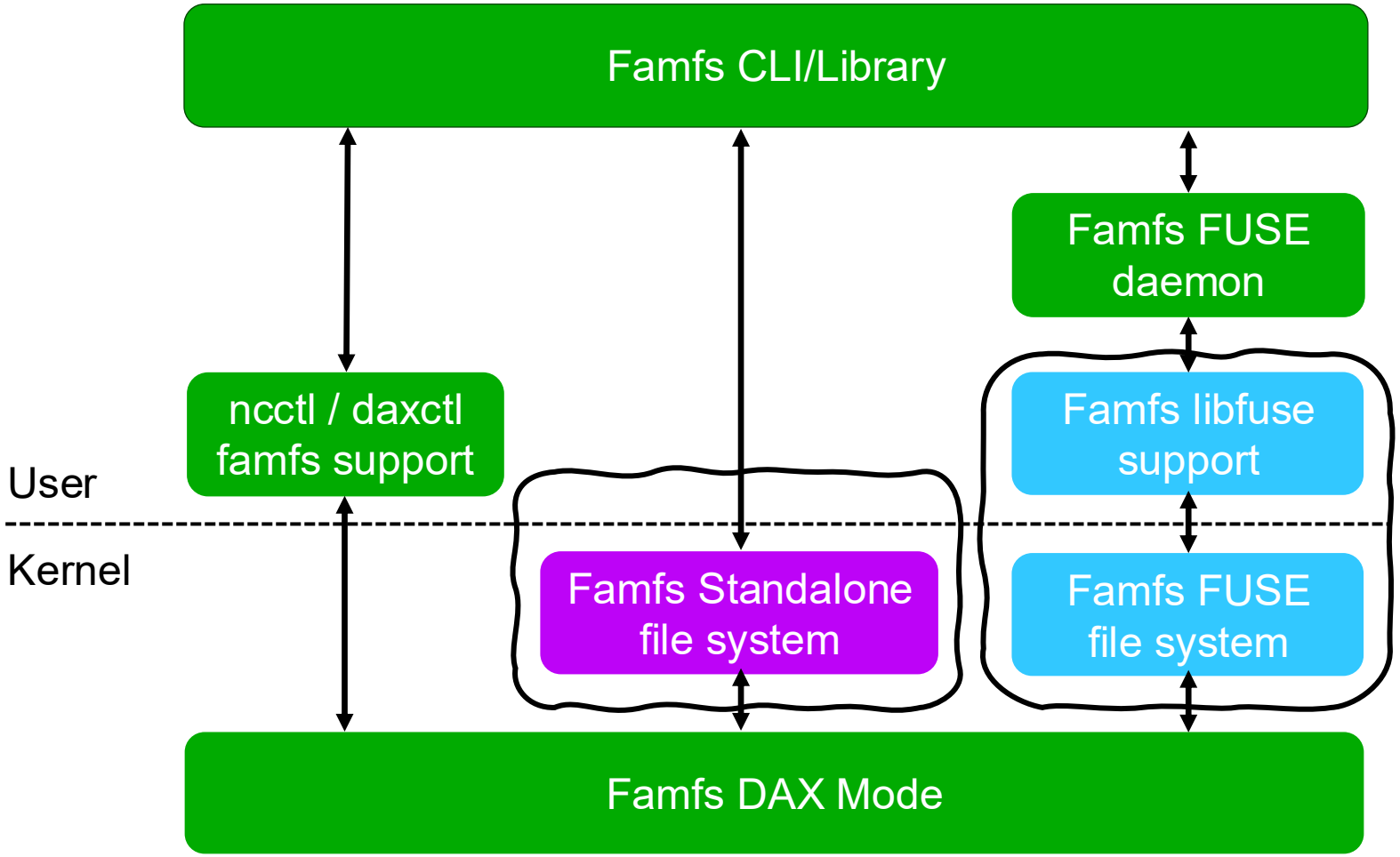
FUSE famfs

- More complex
- Lower metadata performance (but same data-path perf.)
- Sometimes lower system-ram usage
- Metadata handled in user space

What comprises Famfs?

- Kernel Components
  - Famfs mode of DAX
  - Famfs standalone file system
  - Famfs fuse file system
- User space components
  - Famfs CLI and library (see https://famfs.org)
  - Famfs updates to ndctl/daxctl
  - Famfs updates to libfuse

Famfs runs as either a fuse file system or a standalone FS

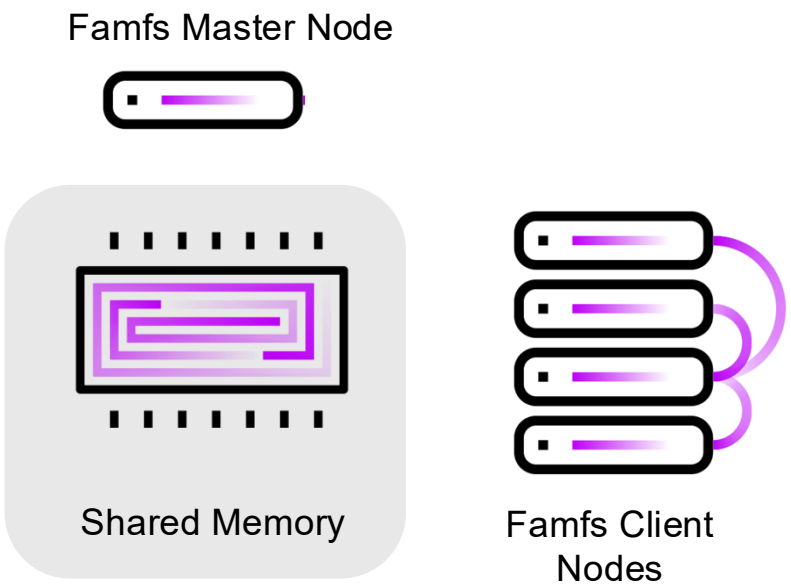


What Is Taking So Long?

- Linux is rightfully cautious about upstreaming new file systems

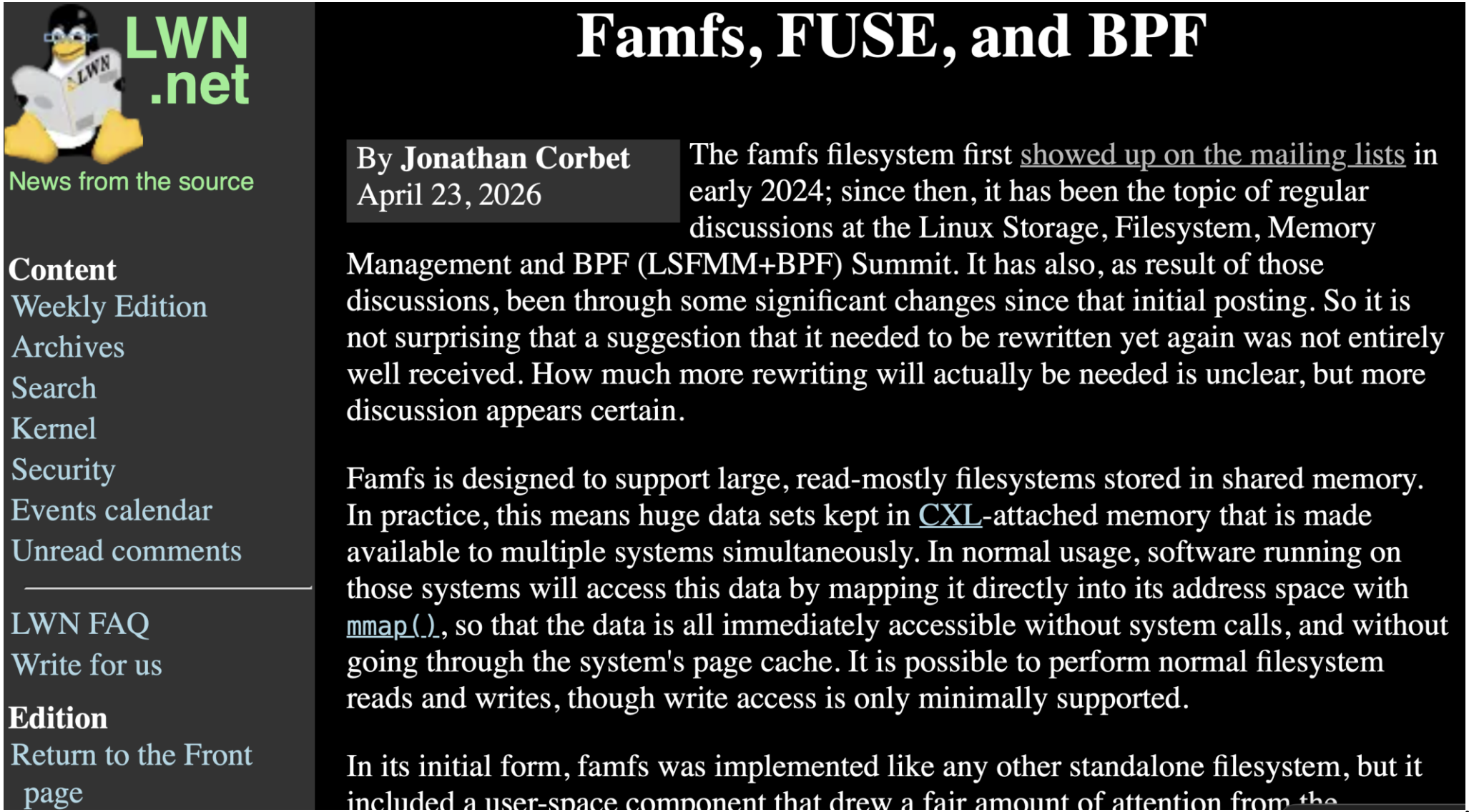
- It is “Truthy” that:
  - Famfs might ”fit in” with FUSE because it manages metadata from user space
  - A file system added to FUSE is lower risk

- But…
  - FUSE is huge and the maintainers are overworked
  - FUSE adds a lot of complexity to famfs
  - Famfs legitimately requires new ABIs that no other FUSE file systems need




```
mkfs.famfs /dev/dax0.0
famfs mount /dev/dax0.0 /mnt/famfs
famfs cp [-r] <src> <dest>
famfs creat -s <size> <dest>
```

Famfs Languishing is a News Story



LWN.net

News from the source

Content
Weekly Edition
Archives
Search
Kernel
Security
Events calendar
Unread comments

LWN FAQ
Write for us

Edition
Return to the Front page

Famfs, FUSE, and BPF

By **Jonathan Corbet**
April 23, 2026

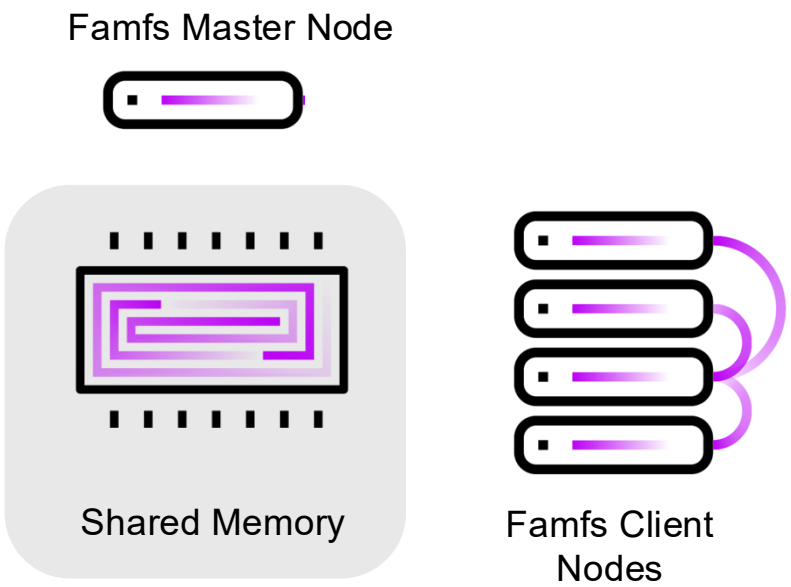
The famfs filesystem first showed up on the mailing lists in early 2024; since then, it has been the topic of regular discussions at the Linux Storage, Filesystem, Memory Management and BPF (LSFMM+BPF) Summit. It has also, as result of those discussions, been through some significant changes since that initial posting. So it is not surprising that a suggestion that it needed to be rewritten yet again was not entirely well received. How much more rewriting will actually be needed is unclear, but more discussion appears certain.

Famfs is designed to support large, read-mostly filesystems stored in shared memory. In practice, this means huge data sets kept in CXL-attached memory that is made available to multiple systems simultaneously. In normal usage, software running on those systems will access this data by mapping it directly into its address space with `mmap()`, so that the data is all immediately accessible without system calls, and without going through the system's page cache. It is possible to perform normal filesystem reads and writes, though write access is only minimally supported.

In its initial form, famfs was implemented like any other standalone filesystem, but it included a user-space component that drew a fair amount of attention from the

Predictions

- Linux 7.1 will include the famfs dax mode
- The famfs file system code will miss the 7.2 window (late June)
- The famfs file system will make the 7.3 merge window (August)

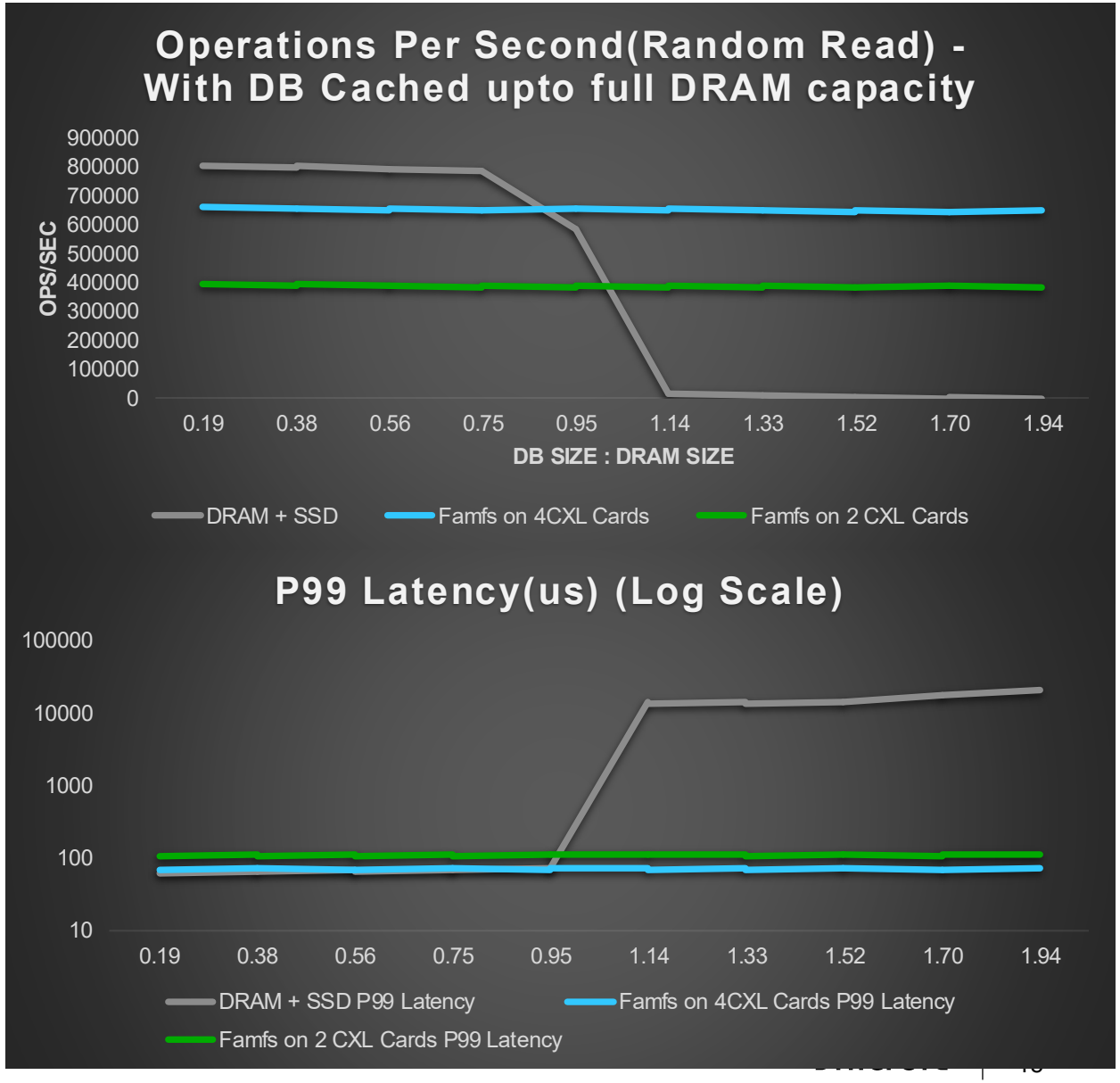



```
mkfs.famfs /dev/dax0.0
famfs mount /dev/dax0.0 /mnt/famfs
famfs cp [-r] <src> <dest>
famfs creat -s <size> <dest>
```

Backup

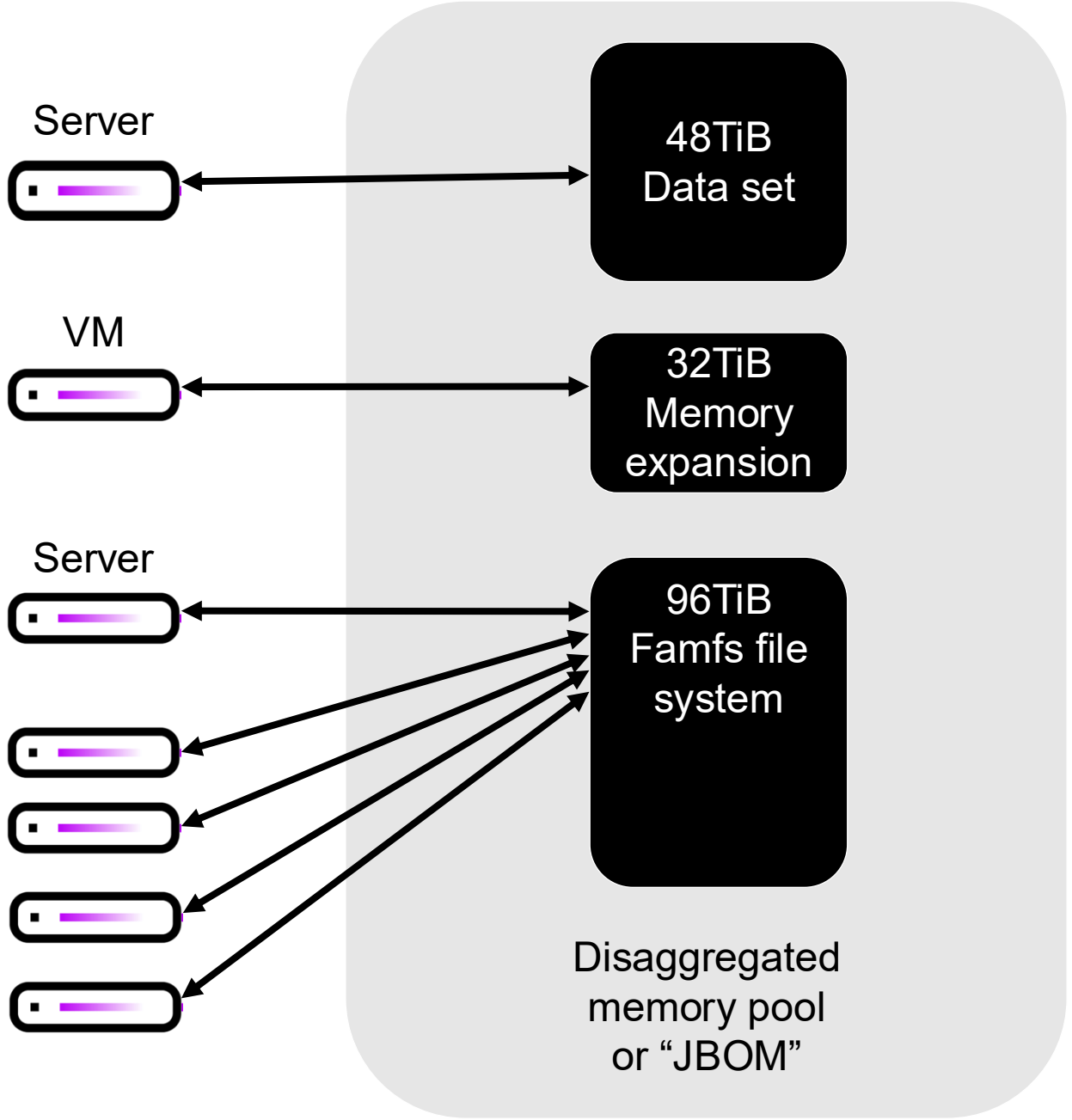
Famfs: bigger data in shared memory

- RocksDB read-only benchmark
- Famfs benchmarks (Green)
  - RocksDB database stored in famfs
  - RocksDB instances on multiple hosts can share the same files/memory
  - No modifications to RocksDB (famfs is just files)
- Control Group (Gray)
  - RocksDB database stored in xfs backed by nvme
  - Cached in DDR; Performance great when it fits in mem
- Benefits:
  - Data is de-duplicated
  - Or sharding / shuffling is avoided
  - Cache line access (less read amplification)



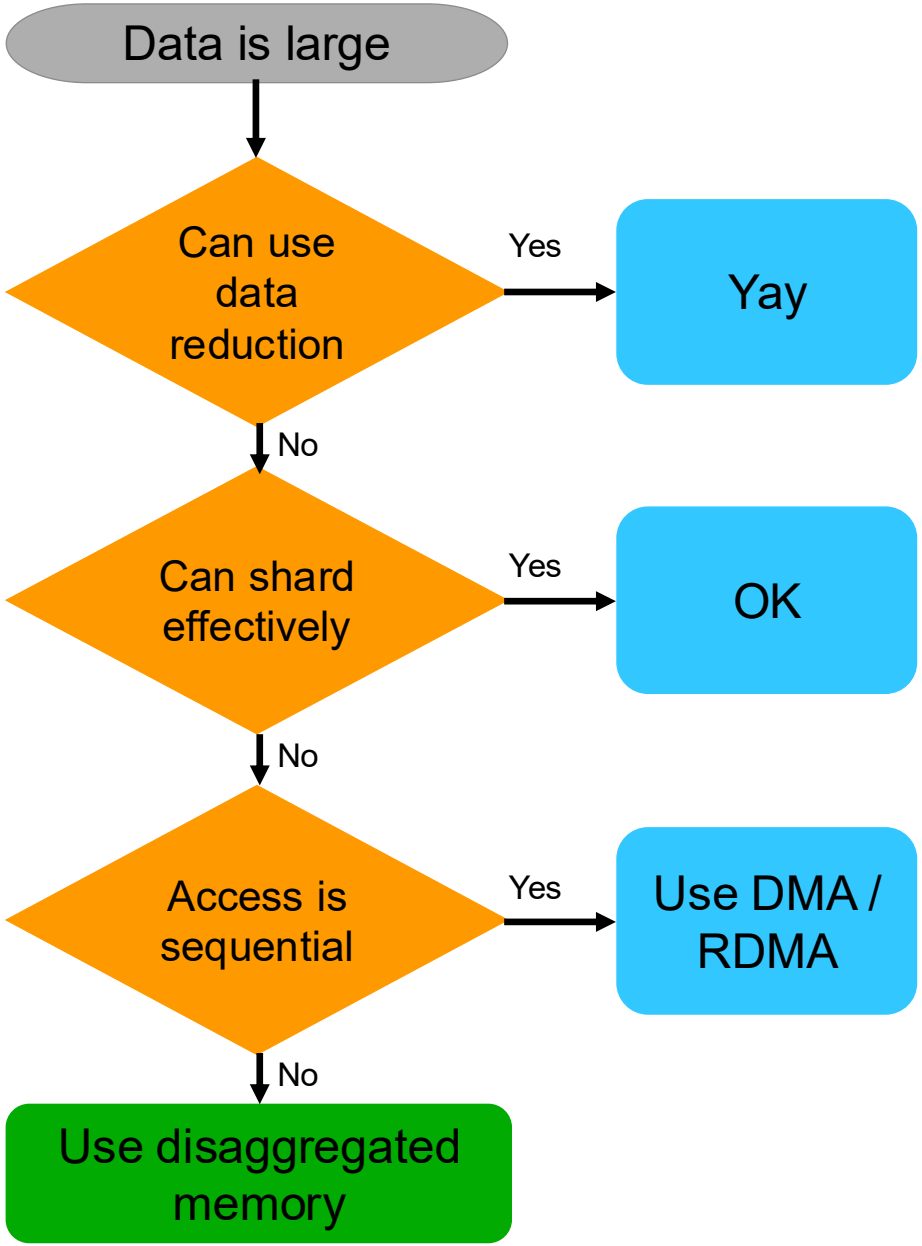
Large datasets don’t just appear, they get “wrangled”

- Not all problems fit in memory
- The problems (data sets) get bigger, but the available techniques remain the same
  - Scale up (bigger servers / mem / GPUs)
  - Scale out (more servers / mem / GPUs)
- Wrangling tools aren’t necessarily memory-efficient: very large system-ram can be needed



What if data is [much] bigger than memory?

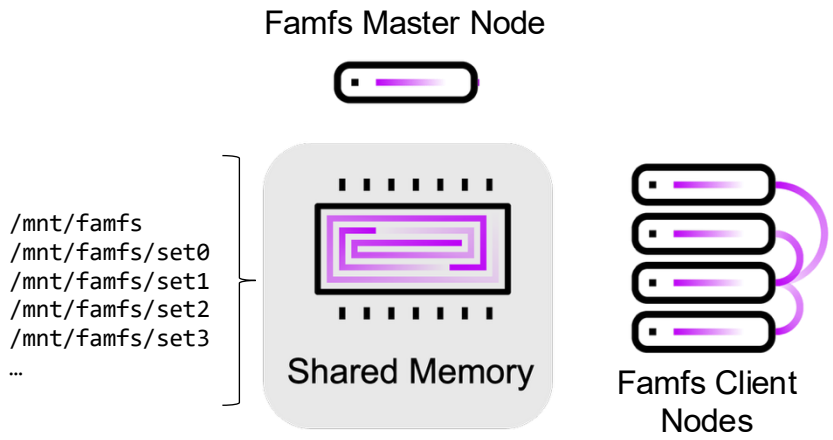
- Some data can be reduced in size effectively
- Some data can be sharded (split across hosts) effectively
- Some data is accessed sequentially, and can be staged via DMA / RDMA
- Random access in disaggregated memory is 2 orders of magnitude lower latency than NVME (100x Improvement)



Famfs organizes disaggregated memory as a scale-out file system

Enabling shared JBOM for all apps that can use files

- Memory is accessible as files
  - Write/read become memcpy
  - Mmap provides byte / cache-line access
- “All” apps can access data in files
- Famfs files are memory and not storage
  - Move data into famfs for in-memory access
  - Move data out of famfs to store persistently
- Posix permissions apply, along with strict partitioning of data from separate files
- Orchestration layers such as PNFS can use famfs as a tier – providing memory performance + scale-out sharing




```
mkfs.famfs /dev/dax0.0
famfs mount /dev/dax0.0 /mnt/famfs
famfs cp [-r] <src> <dest>
famfs creat -s <size> <dest>
```

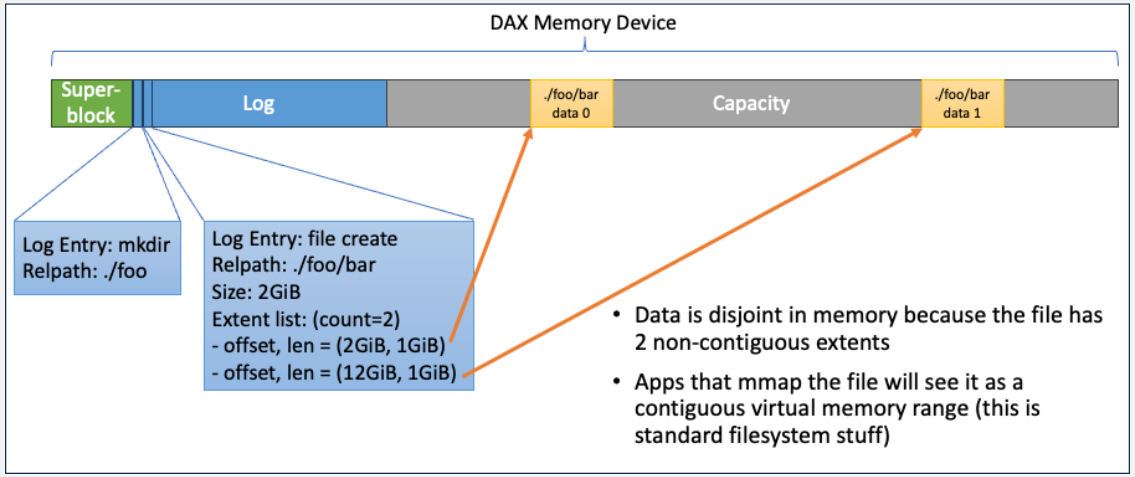
Interleaving is critical for memory performance

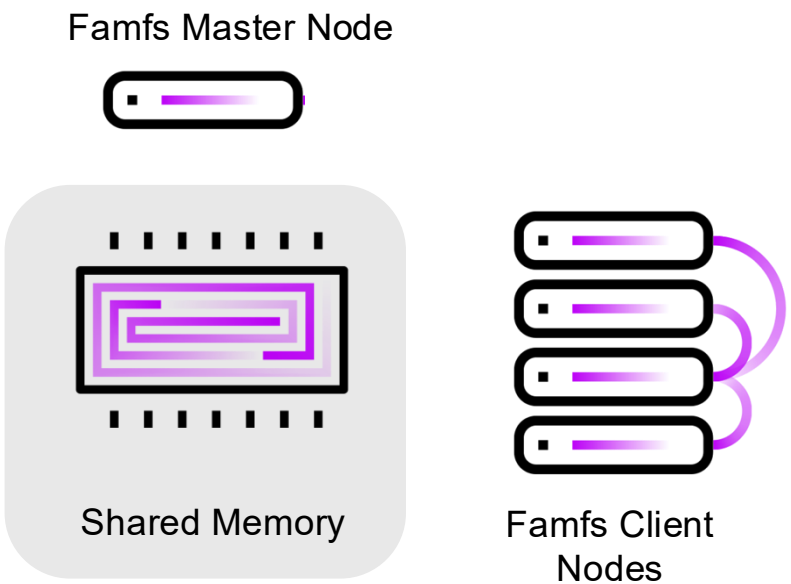
- CXL supports hardware interleaving but…
  - The device physical address (DPA) range must be identical on all memory devices in an interleaved set
  - But “memory devices” are virtual – based on DCD (dynamic capacity device) allocations
  - The normal fragmentation of alloc / release will make it difficult or impossible to allocate the same DPA range on, say, 16 allocations from different CXL memories

- Each famfs file can be interleaved across many CXL memory devices
  - Famfs has no constraints about DPA ranges

Famfs architecture (MVP)

- All metadata is stored in an append-only log
- Log is written by Master and ”played” by Clients
- V1 handles clients with stale metadata by not supporting truncate or delete
- Metadata handled in user space (library, cli, currently no daemons)
- Read / write / mmap / vma faults handled in kernel
- Memory mapping from famfs == cache-line level access to shared mem
- Many of the limitations can be addressed in future versions



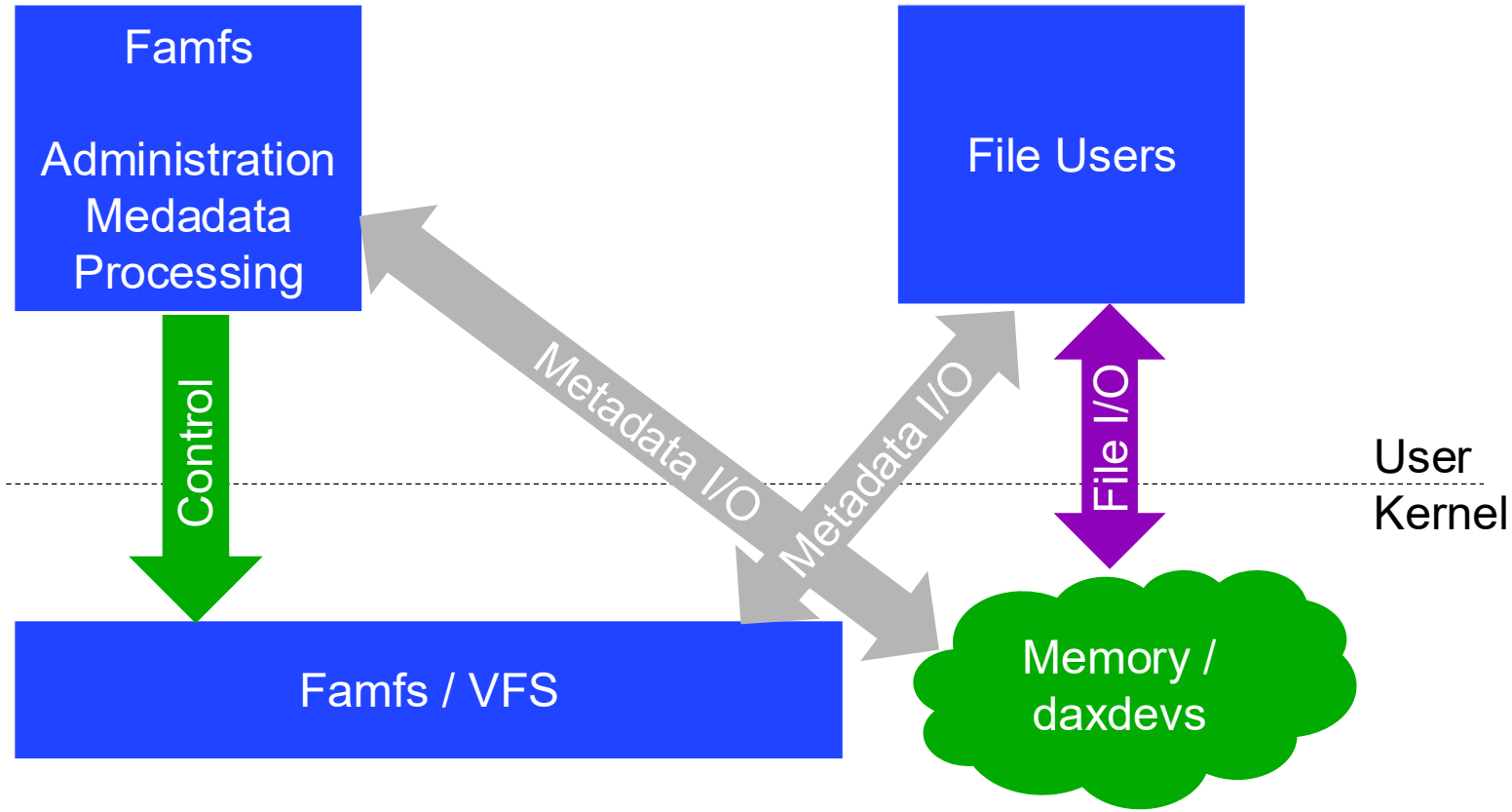



```
mkfs.famfs /dev/dax0.0
famfs mount /dev/dax0.0 /mnt/famfs
famfs cp [-r] <src> <dest>
famfs creat -s <size> <dest>
```

Famfs: Functional Blocks

- Metadata log is written and read by user space components
- File ”fmaps” are pushed into the kernel from user space
- Users see regular files

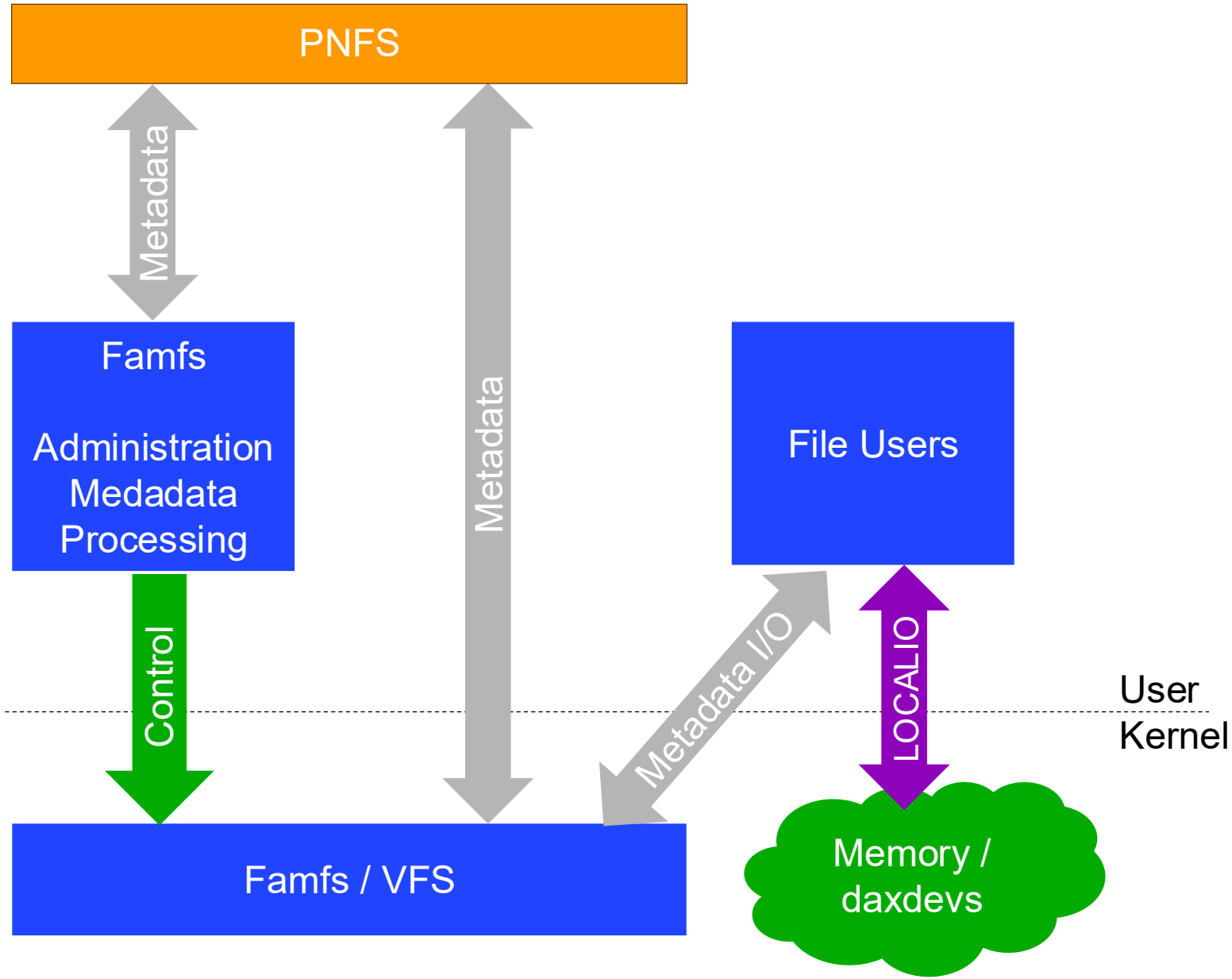
- Metadata distribution model could change (pnfs integration, anyone?)



Famfs: Functional Blocks

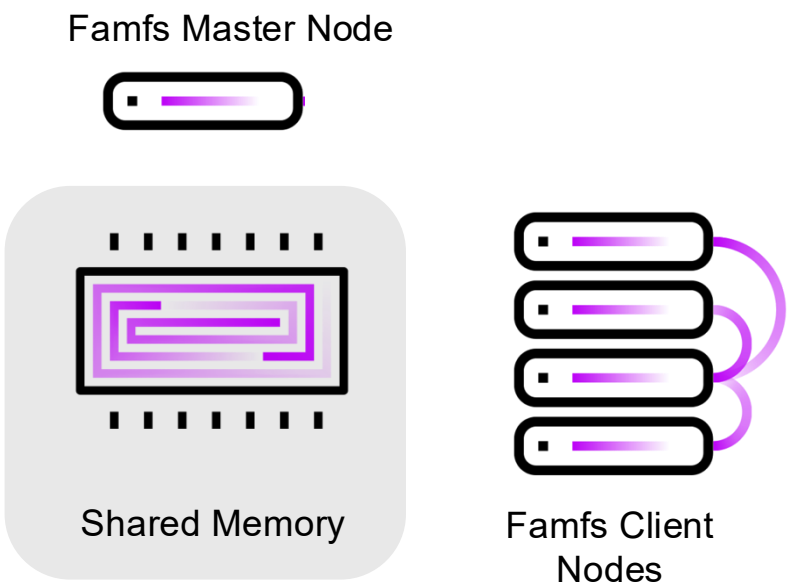
- Metadata log is written and read by user space components
- File ”fmaps” are pushed into the kernel from user space
- Users see regular files

- PNFS could solve metadata consistency
- Probably need something in the file I/O path



Summary

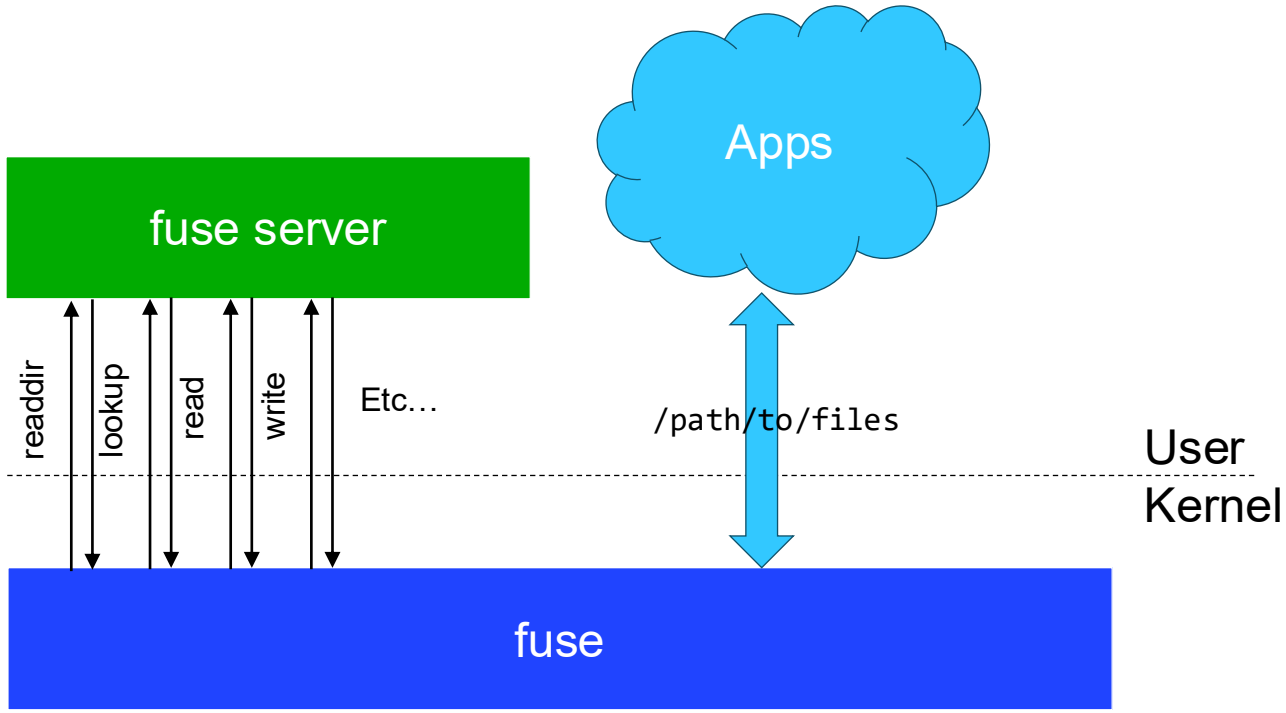
- Disaggregated memory breaks scaling barriers for latency-sensitive workloads
- Famfs provides a natural, scale-out access method for data in disaggregated, shared memory
- Software does not need to change to use shared memory!




```
mkfs.famfs /dev/dax0.0
famfs mount /dev/dax0.0 /mnt/famfs
famfs cp [-r] <src> <dest>
famfs creat –s <size> <dest>
```

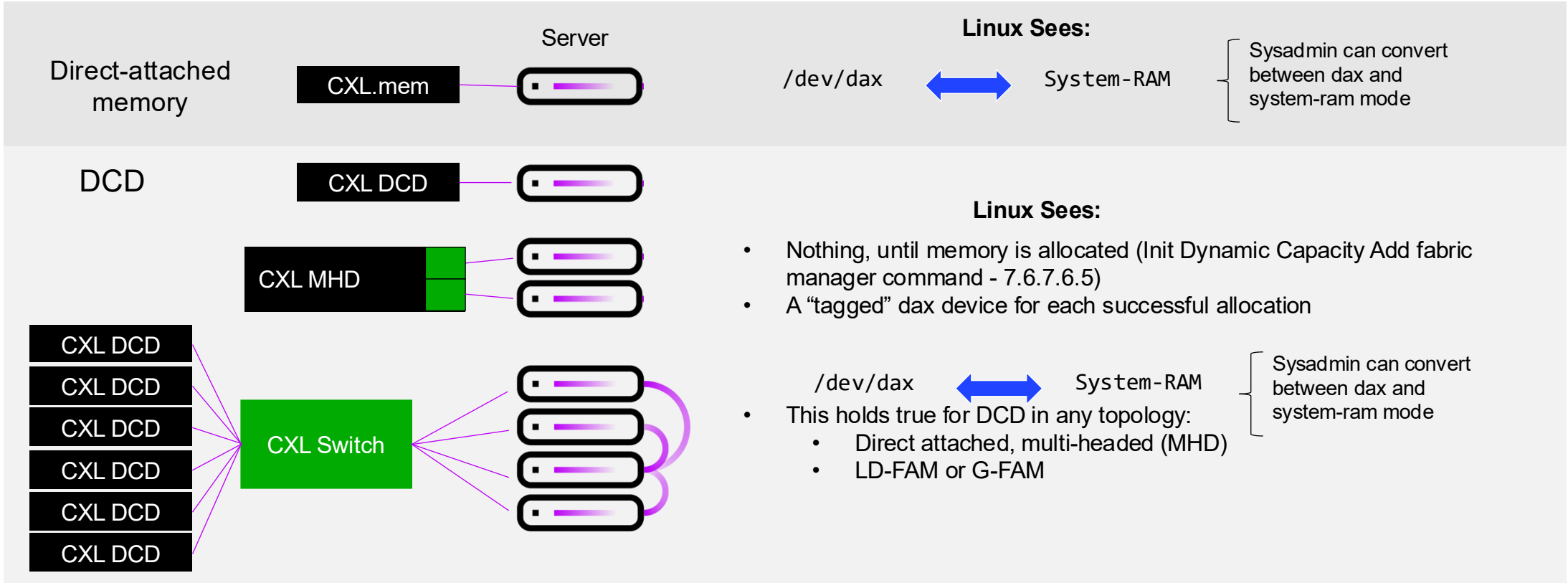
Fuse (File System in User Space)

- Fuse kernel component provides a file system mount
- Fuse server (AKA Fuse Daemon) is authoritative as to what files exist
- Fuse server facilitates I/O
- Fuse server enforces any limitations (with help from fuse in the kernel)
- Famfs fits logically, in that it already handles metadata from user space
- But fuse has notoriously poor performance



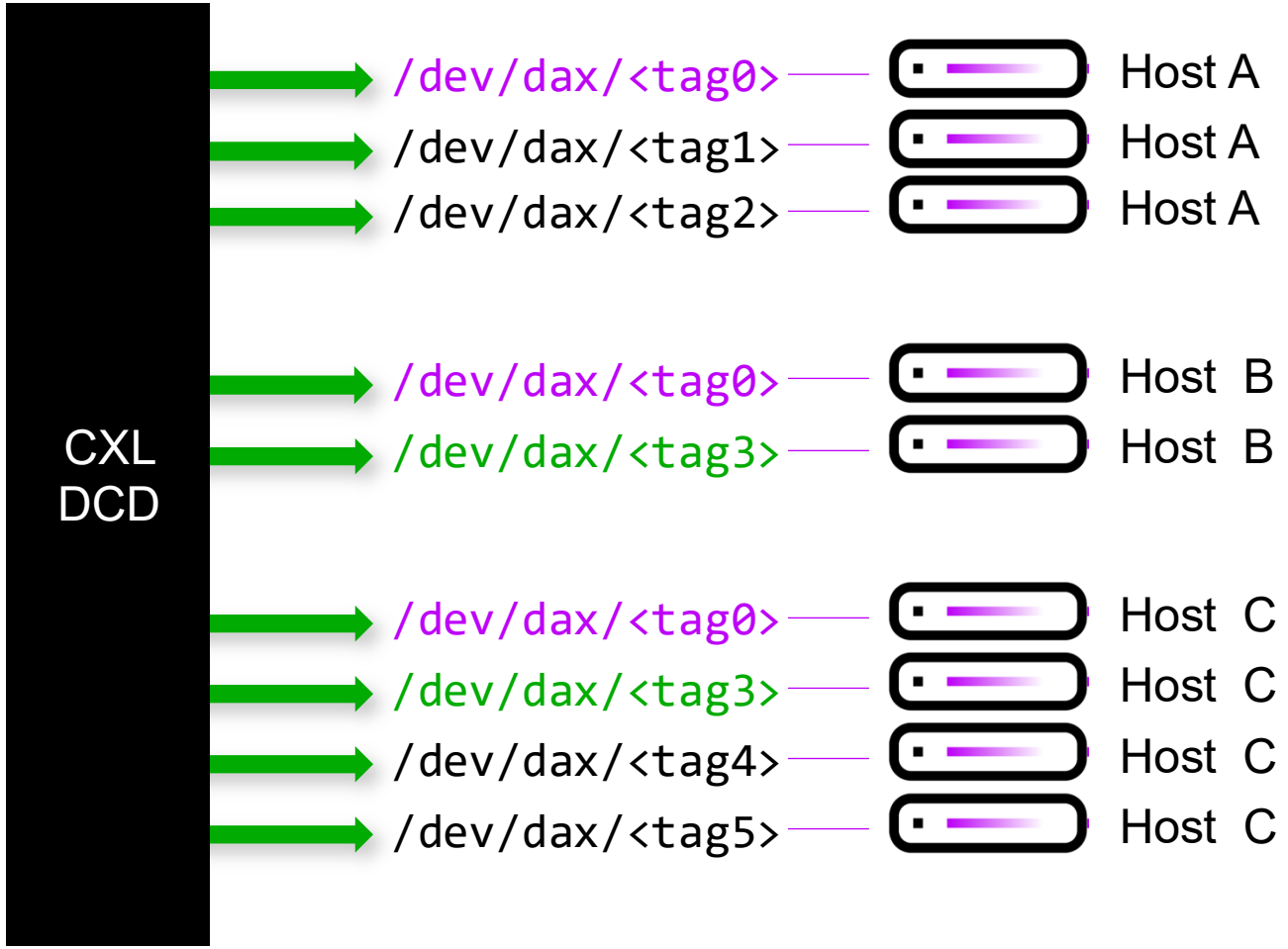
Background: CXL Memory sharing topology

- Think of a **Dynamic Capacity Device** (**DCD**) as a memory device with built-in allocator and access control
- The allocator is necessary for multi-host environments
- DCD (via fabric manager) can give additional hosts access to a sharable allocation, writable or read-only, etc.



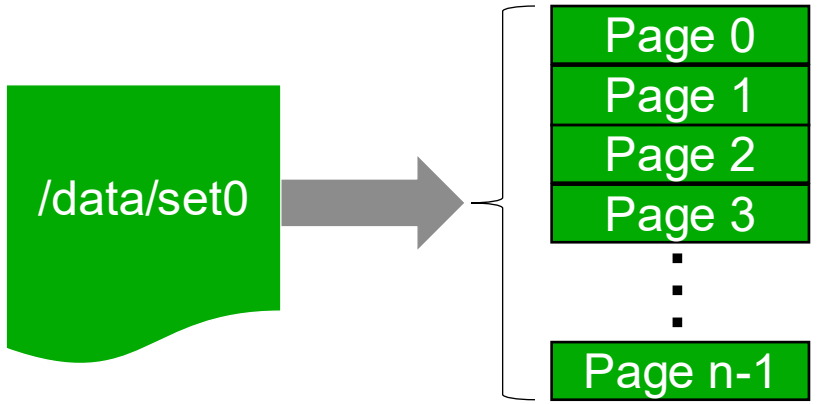
CXL tagged capacity name space

- DCD is not usable until memory is allocated
- Allocation (Init DC Add)(sharable allocations are ”tagged”, and appear as “virtual” dax devices)
- Tagged dax memory can be “onlined” as system-ram (non-shared memory)
- Sharable memory can surface simultaneously or not
- A famfs instance lives on one or more tagged dax memory instances
- Famfs can also interleave files across an arbitrary number of Tags
- CXL interleave can be programmed across multiple tagged allocations*



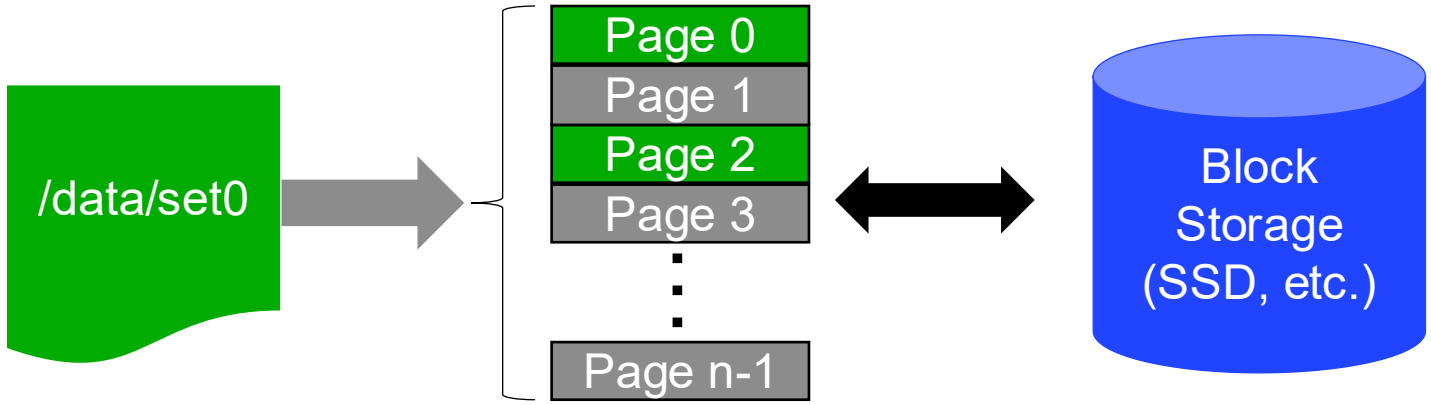
Conventional files as memory

- Files [already] map to memory …if the data is in memory
- When the data is in memory:
  - Read/Write are just memcpy() variants
  - Memory mapping assembles the pages into a virtual address range that is directly accessed as memory
- Many are aware of TLBs and page tables, which resolve a virtual address to memory
  - A TLB + page-table miss results in a `fault()` call to the file system to resolve the file offset to a page



Conventional files as memory

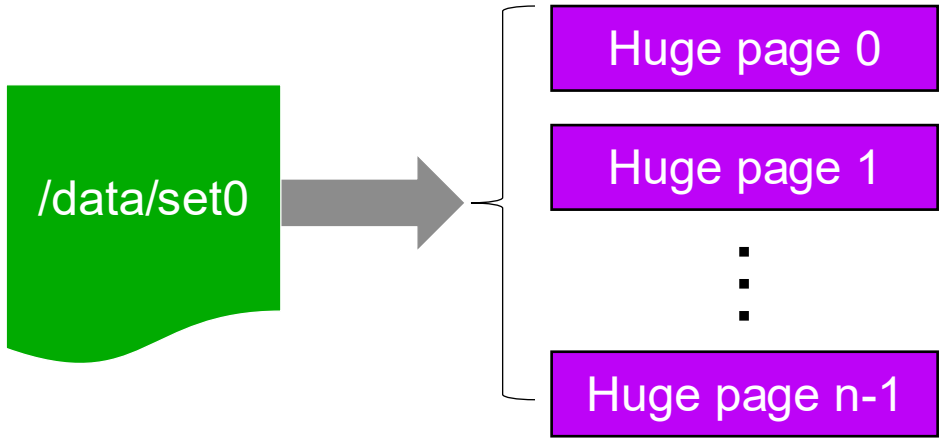
- Files [already] map to memory
  …if the data is in memory
- When the data is in memory:
  - Read/Write are just memcpy() variants
  - Memory mapping assembles the pages into a virtual address range that is directly accessed as memory
- Many are aware of TLBs and page tables, which resolve a virtual address to memory
  - A TLB + page-table miss results in a `fault()` call to the file system to resolve the file offset to a page



- Conventional file systems sparsely cache pages from a files backing store
  - Meaning a `fault()` call might have to allocate memory and retrieve data from backing storage
- Pages that are cached (green) are accessed as memory
- Pages that are not in cache (gray) must be faulted in from backing store if accessed

Famfs files as memory

- Files [already] map to memory
  …if the data is in memory
- When the data is in memory:
  - Read/Write are just memcpy() variants
  - Memory mapping assembles the pages into a virtual address range that is directly accessed as memory
- Many are aware of TLBs and page tables, which resolve a virtual address to memory
  - A TLB + page-table miss results in a `fault()` call to the file system to resolve the file offset to a page



- Famfs is not sparse; files are always fully mapped to memory
- Famfs data lives in [sharable] dax memory devices
- Huge page mapping reduces virtual memory mapping overhead by 512x
- Since the backing memory is not sparse, there is no downside to huge page mapping

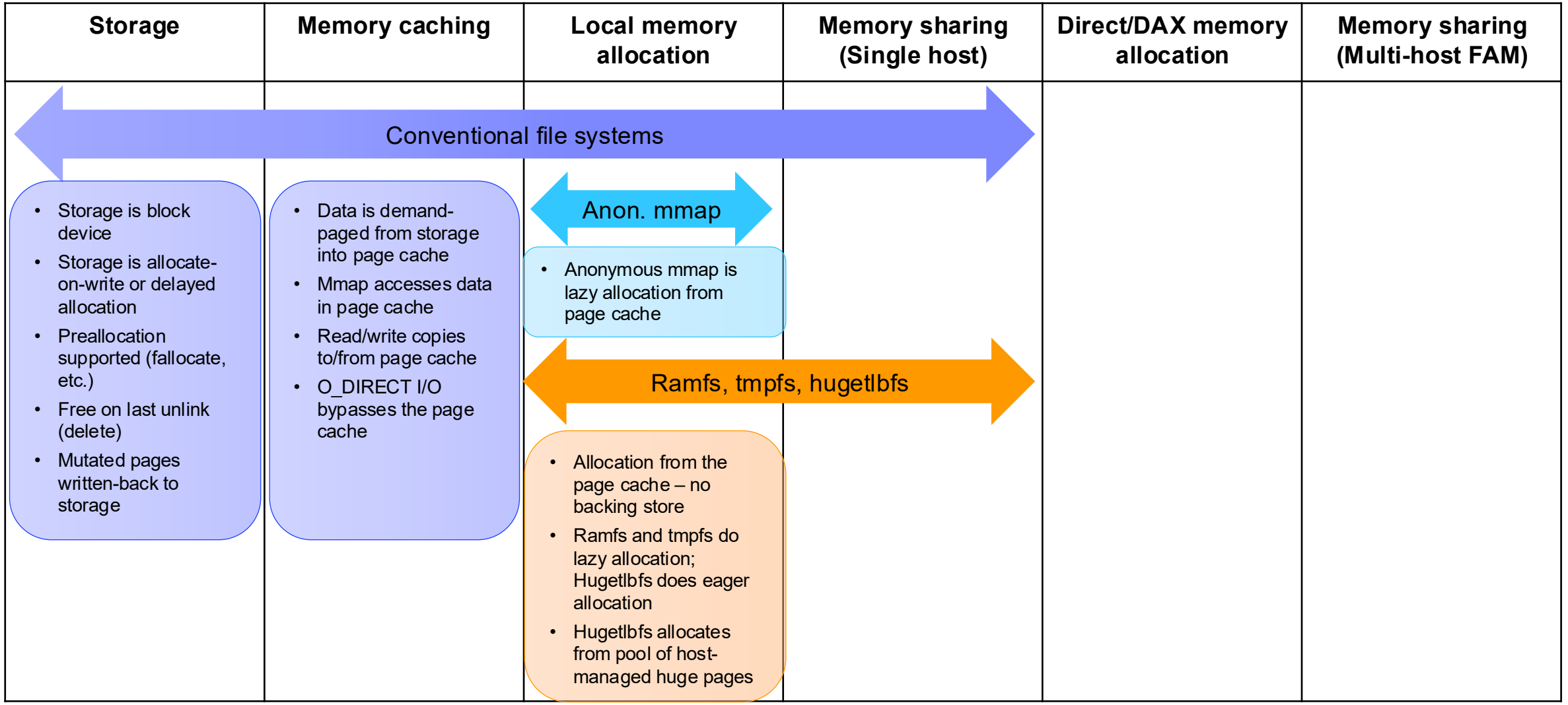
File system / VFS functionality

Storage	Memory caching	Local memory allocation	Memory sharing (Single host)	Direct/DAX memory allocation	Memory sharing (Multi-host FAM)
Conventional file systems					
• Storage is block device • Storage is allocate-on-write or delayed allocation • Preallocation supported (fallocate, etc.) • Free on last unlink (delete) • Mutated pages written-back to storage	• Data is demand-paged from storage into page cache • Mmap accesses data in page cache • Read/write copies to/from page cache • O_DIRECT I/O bypasses the page cache				

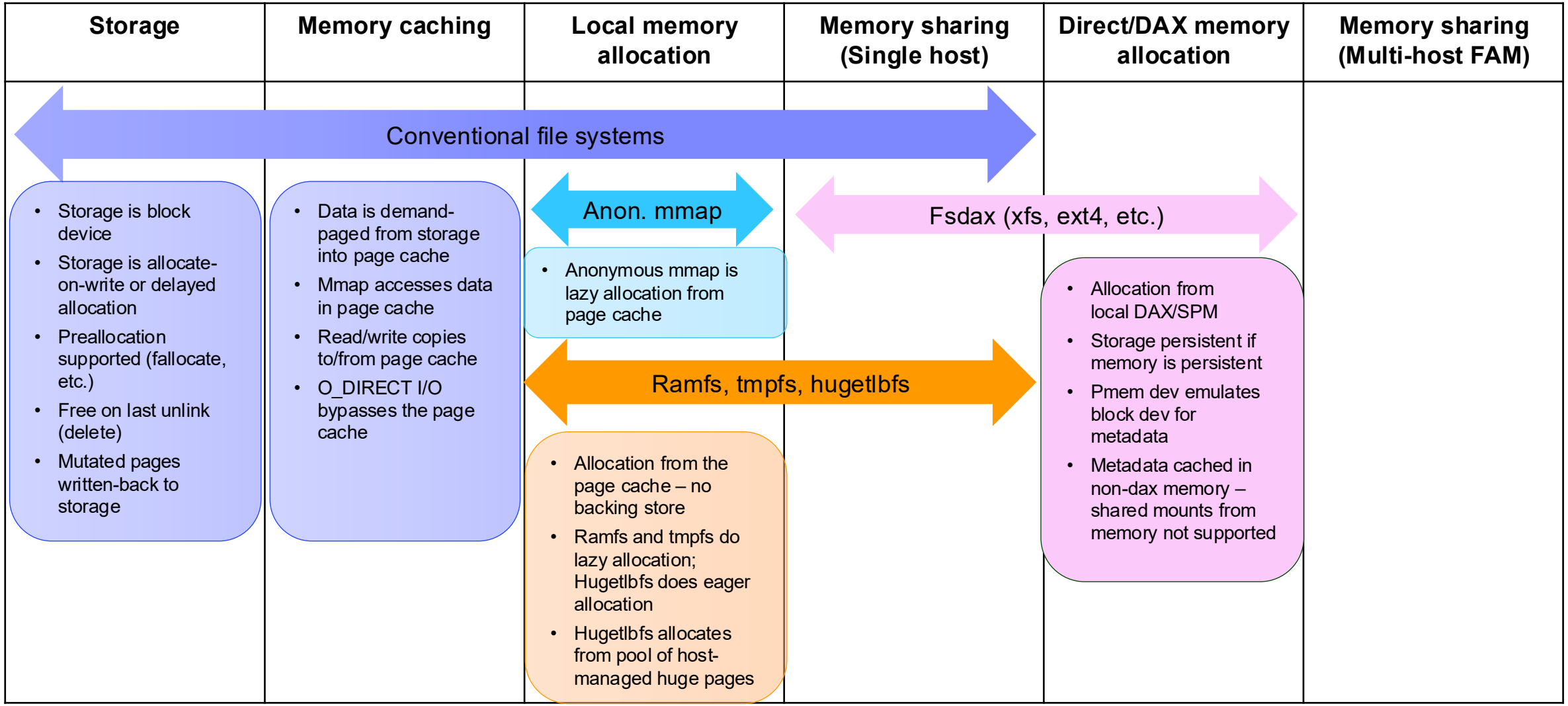
File system / VFS functionality

Storage	Memory caching	Local memory allocation	Memory sharing (Single host)	Direct/DAX memory allocation	Memory sharing (Multi-host FAM)
Conventional file systems					
• Storage is block device • Storage is allocate-on-write or delayed allocation • Preallocation supported (fallocate, etc.) • Free on last unlink (delete) • Mutated pages written-back to storage	• Data is demand-paged from storage into page cache • Mmap accesses data in page cache • Read/write copies to/from page cache • O_DIRECT I/O bypasses the page cache	Anon. mmap • Anonymous mmap is lazy allocation from page cache			

File system / VFS functionality



File system / VFS functionality



File system / VFS functionality