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
Aug 2025

micron Intelligence Accelerated™

Background: CXL / Disaggregated memory usage models

Pooling (composable system-ram)

System-RAM

(Owned/allocated by Linux)

DAX or Famfs mem

(not allocated by Linux)

- Memory is added as System RAM (managed by Linux)
- Tiering and migration are viable (`migrate_pages()`, TPP, DAMON, etc.)
- Incompatible with multi-host sharing (memory gets zeroed when Linux “onlines” it)
- It’s possible to provision very large amounts of memory for jobs that can’t run in 3-4T

Sharing / ‘not system ram’

- The hardware supports this (CXL3, DCD, etc.)
- These cases include
  - Both concurrent and sequential sharing
  - Other use cases that use Linux memory-mgmt
- Software usage is too complicated
- Famfs is the missing link
  - “All” apps can use data in files
  - Files already map to memory
  - Many apps use big data in files
  - RAS “blast radius” is limited to apps that access the memory

famfs

Famfs: The Core Insight

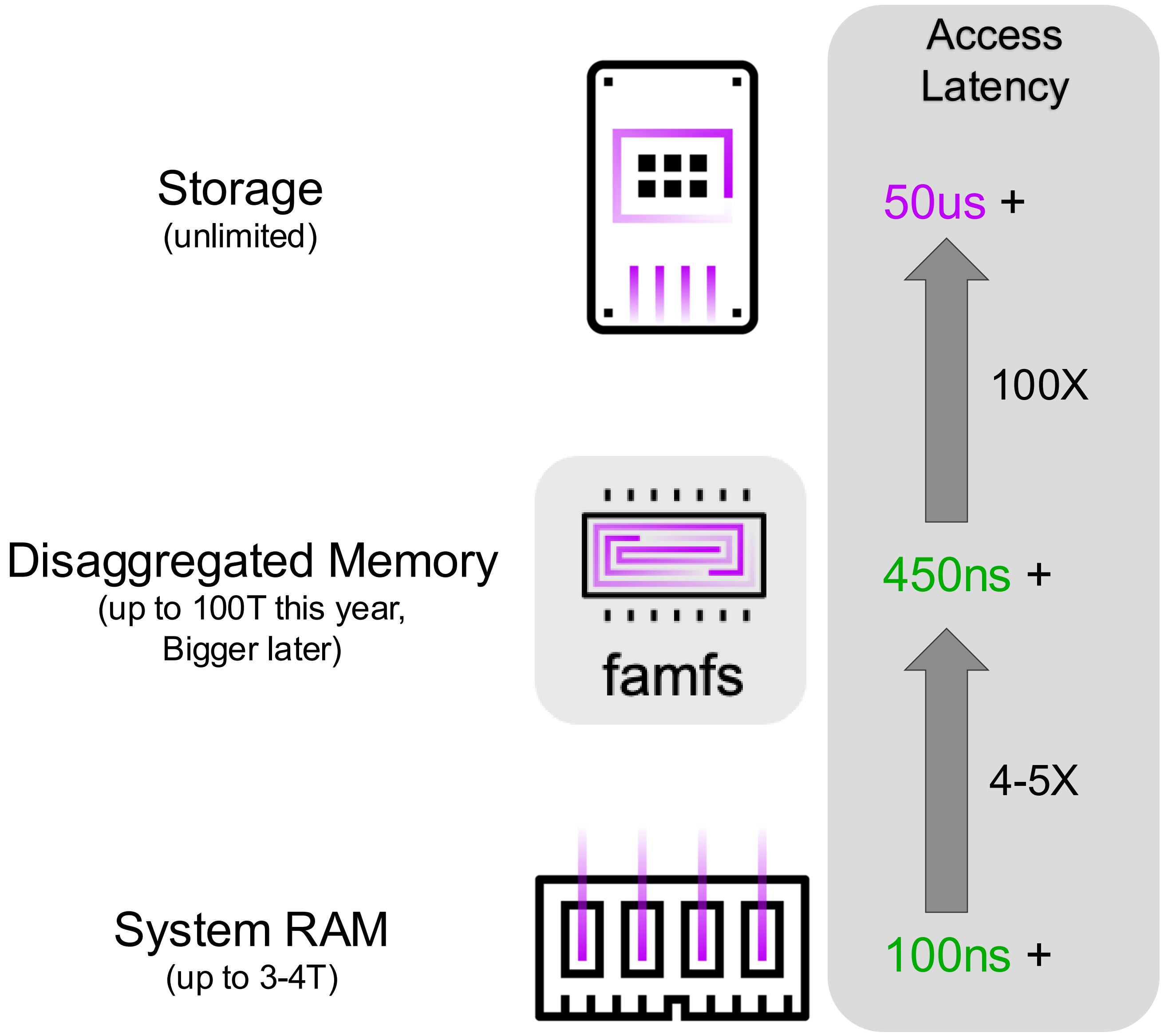
- Sharable memory needs a standard access method
  - Linux has no concept of memory that isn't wholly owned

- 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!"
  - See HP's "The Machine"

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



Storage
(unlimited)

Disaggregated Memory
(up to 100T this year,
Bigger later)

famfs

System RAM
(up to 3-4T)

Access Latency

50us +

100X

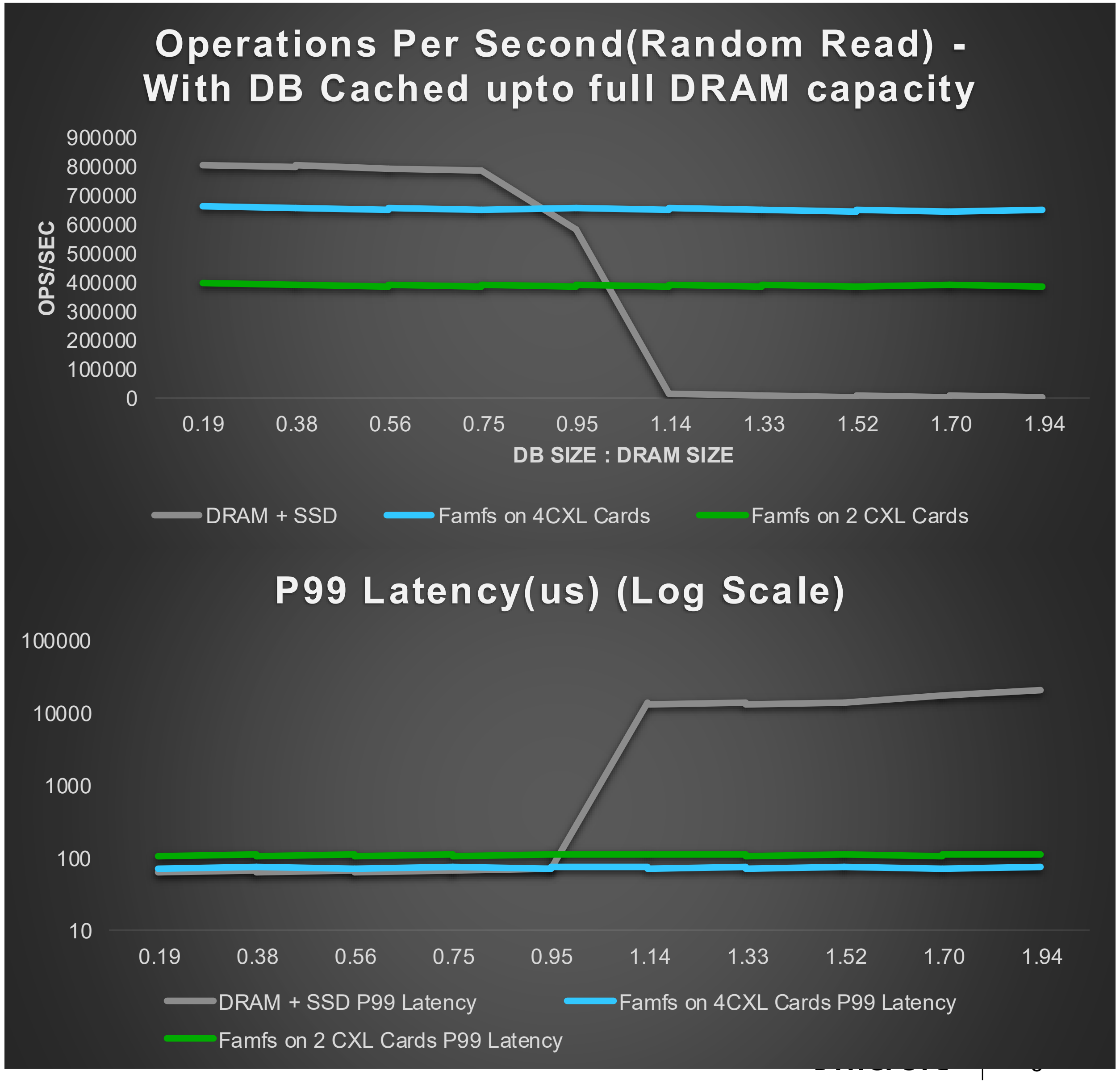
450ns +

4-5X

100ns +

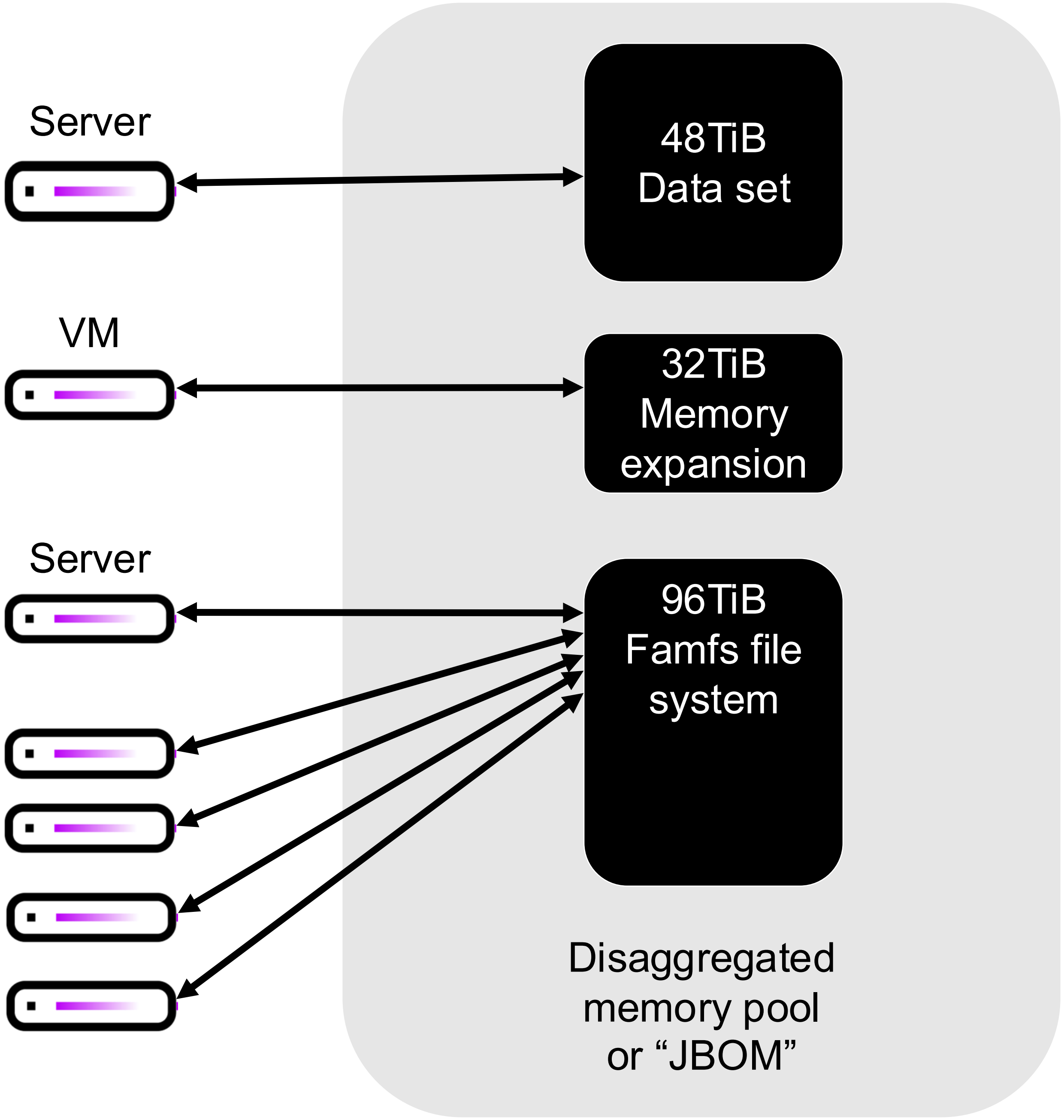
Famfs: bigger data in shared memory

- 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)



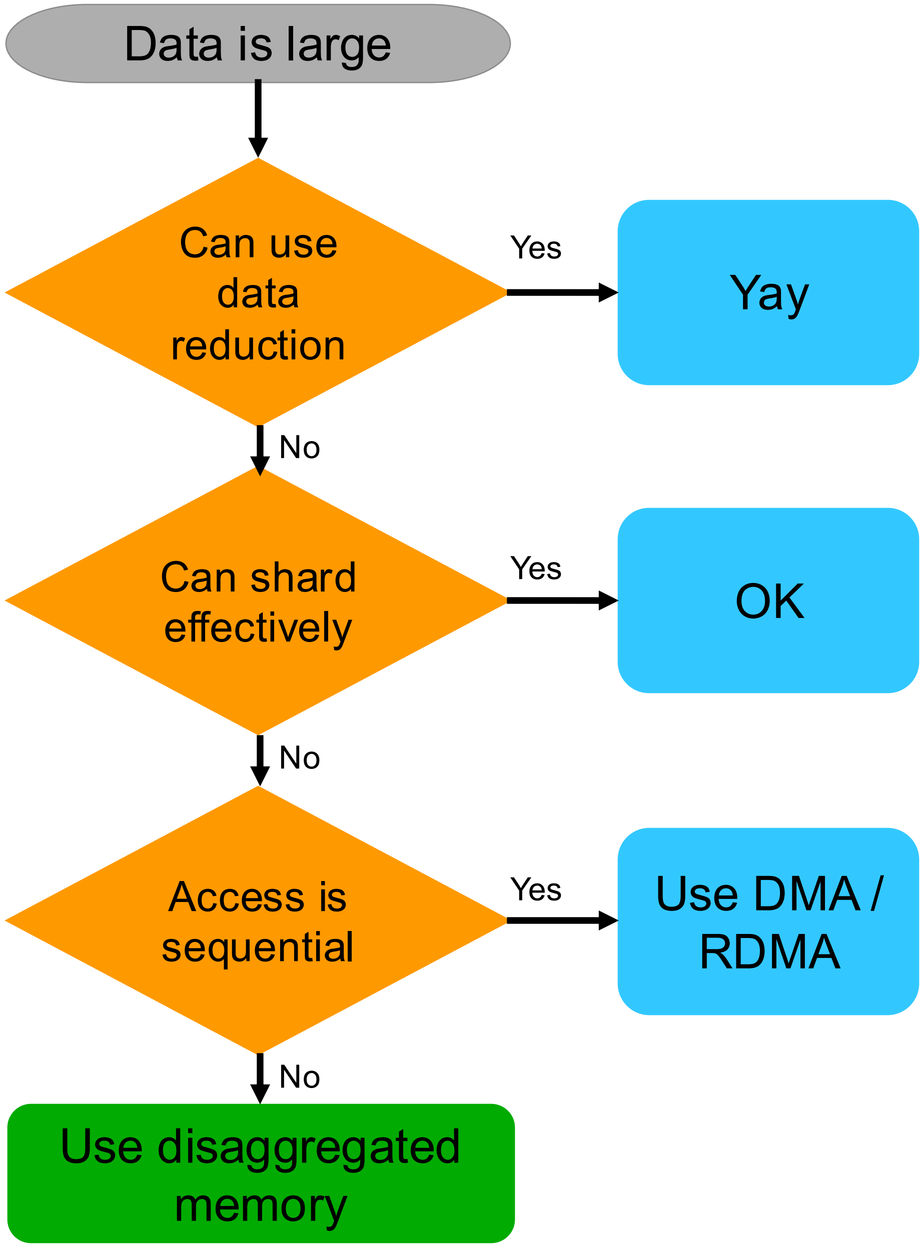
Large datasets don’t just appear, they get “wrangled”

- 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?

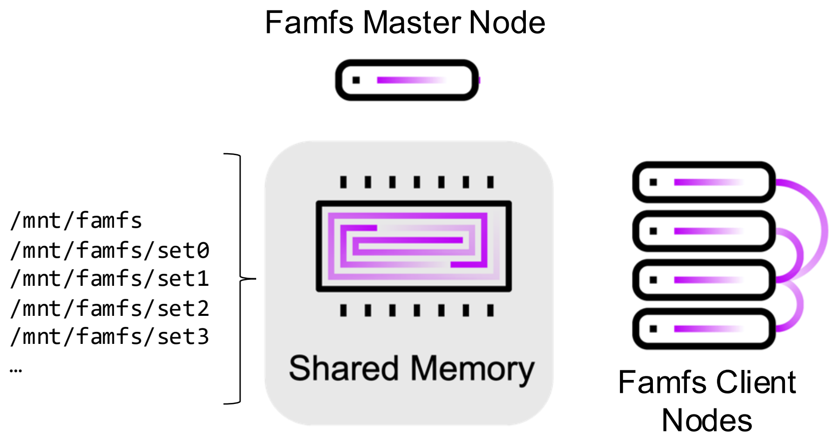
- Some data can be reduced in size effectively
- Some data can be sharded (split across hosts) effectively
- Some data is accessed sequentially, and can be staged via DMA / RDMA
- Random access in disaggregated memory is 2 orders of magnitude lower latency than NVME (100x Improvement)



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>
```

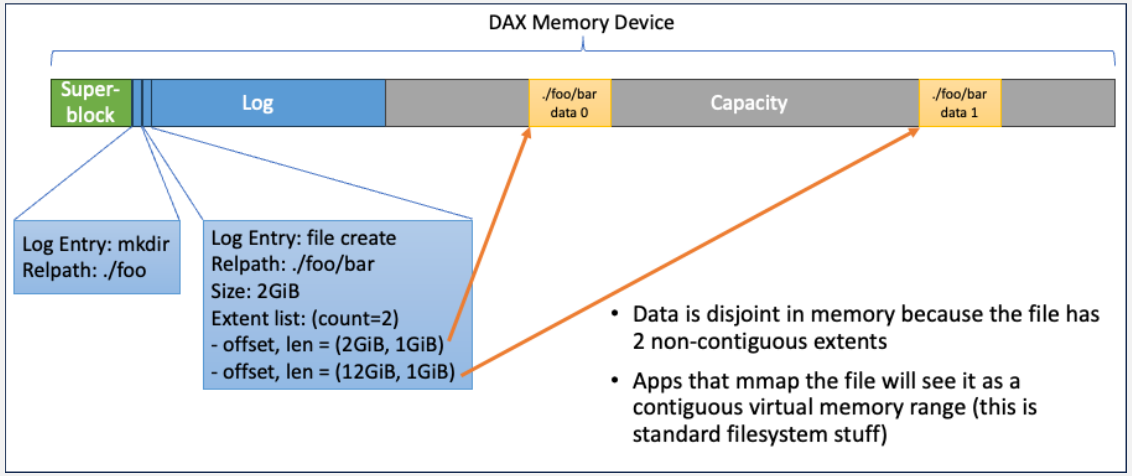
Interleaving is critical for memory performance

- 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)

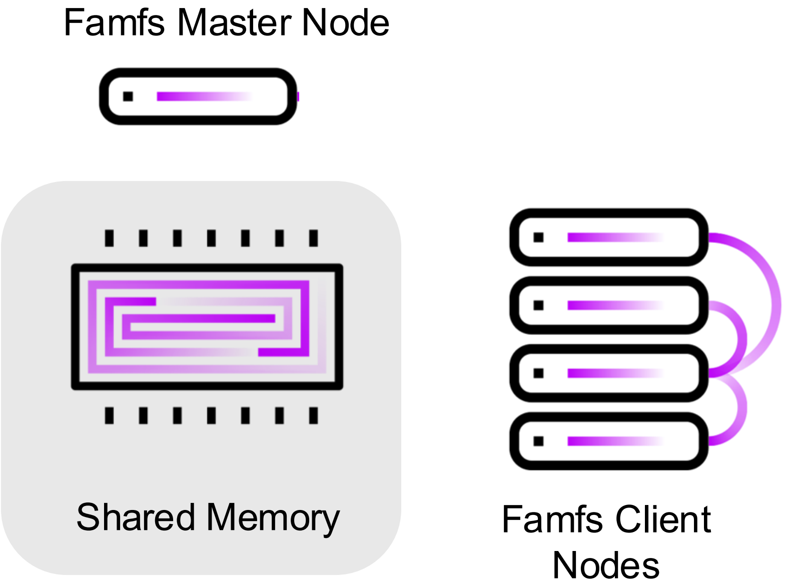
- All metadata is stored in an append-only log
- Log is written by Master and ”played” by Clients
- V1 handles clients with stale metadata by not supporting truncate or delete
- Metadata handled in user space (library, cli, currently no daemons)
- Read / write / mmap / vma faults handled in kernel
-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



Log Entry: mkdir
Relpath: ./foo

Log Entry: file create
Relpath: ./foo/bar
Size: 2GiB
Extent list: (count=2)
- offset, len = (2GiB, 1GiB)
- offset, len = (12GiB, 1GiB)

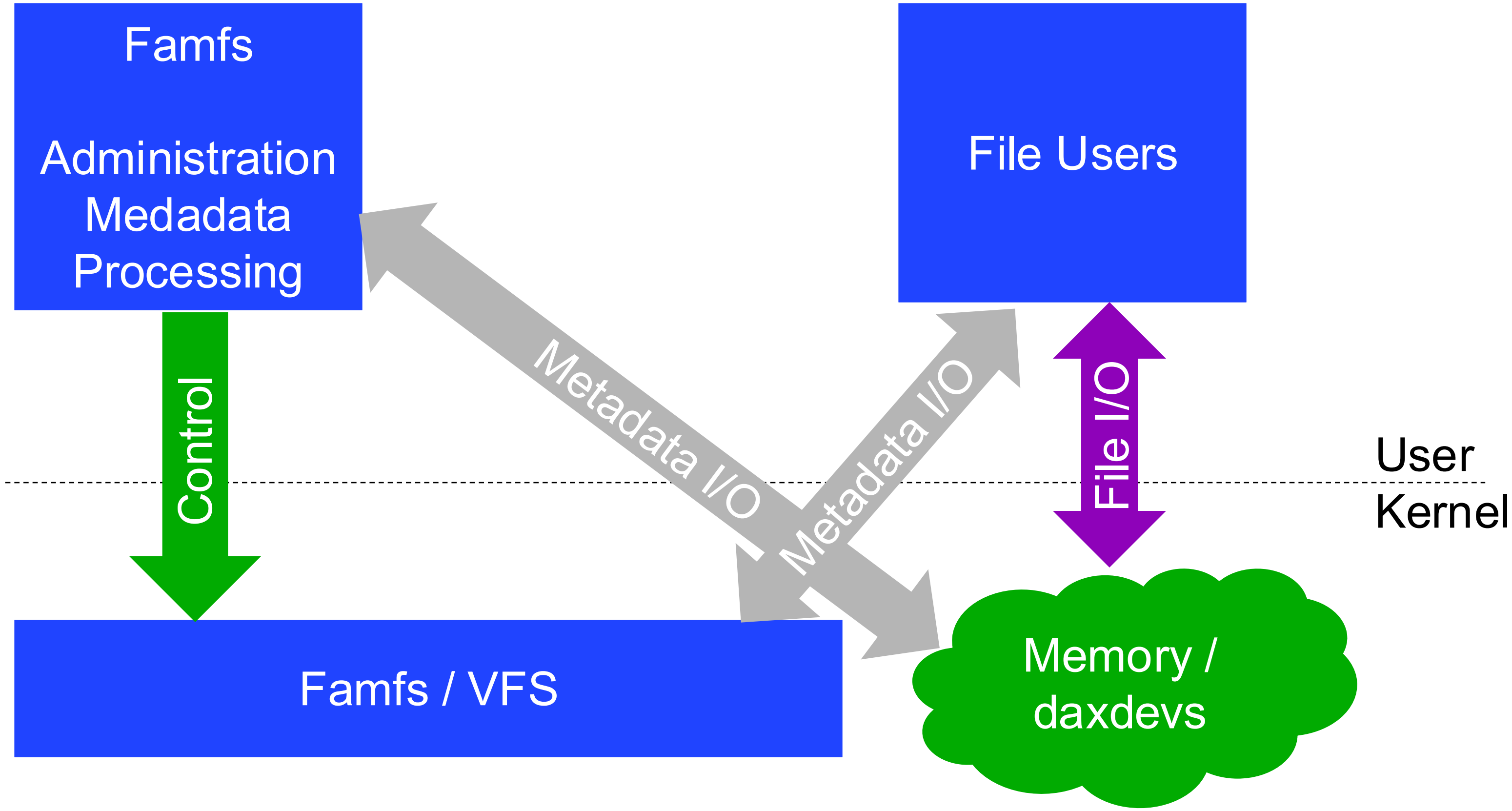
- Data is disjoint in memory because the file has 2 non-contiguous extents
- Apps that mmap the file will see it as a contiguous virtual memory range (this is standard filesystem stuff)



```
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

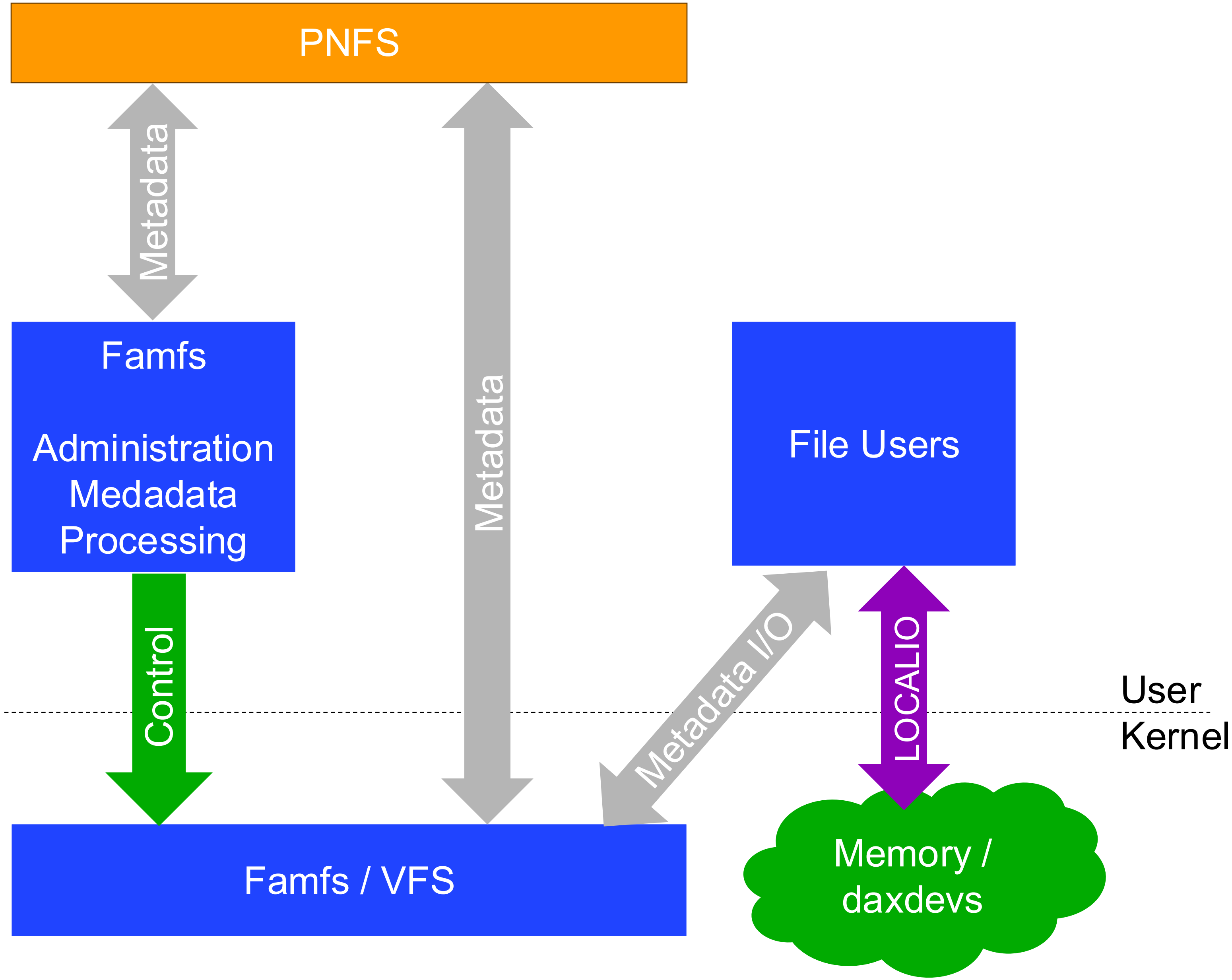
- Metadata log is written and read by user space components
- File ”fmaps” are pushed into the kernel from user space
- Users see regular files
- Metadata distribution model could change (pnfs integration, anyone?)



Famfs: Functional Blocks

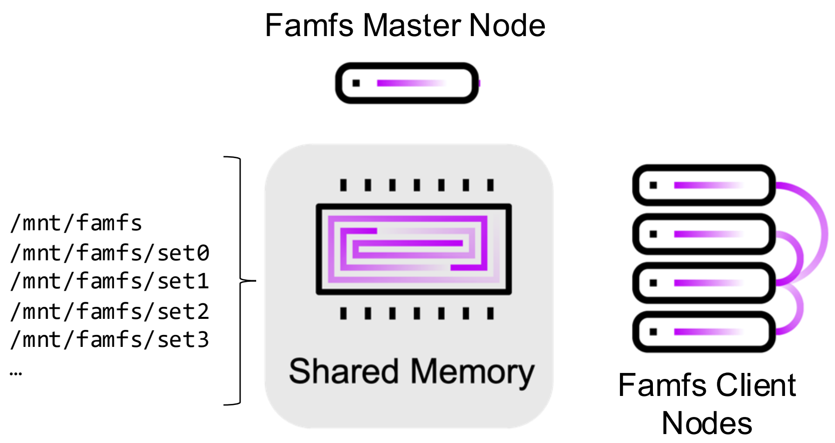
- Metadata log is written and read by user space components
- File ”fmaps” are pushed into the kernel from user space
- Users see regular files

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



Famfs status: on track for Linux upstream in late 2025 / early 2026

- Nov 2023 – Introduced famfs at the Linux Plumbers Conference
- Spring 2025 – Famfs Linux patch sets released (v1, v2)
- May 2024 – Famfs session at LSFMM
  (Linux Storage, File System and Memory Management summit)
  - Conclusion: Famfs merging into fuse
- Aug 2024 – Famfs adds interleaved file support
- Nov 2024 – Famfs covered in Storage Newsletter piece on SC24
- 2024 – Famfs in pilot use at CERN, Alibaba, Intel, Universities, etc.
- Sep 2024 – Famfs session at Linux Plumbers Conference
- Feb 2025 – Famfs poster at Usenix FAST Conference
- Mar 2025 – Famfs session at LSFMM (LWN Article)
- Spring 2025 – Famfs fuse-based patch sets released (v1, v2)
- Famfs documentation:
  https://github.com/cxl-micron-reskit/famfs/blob/master/README.md



Famfs Master Node

```
/mnt/famfs
/mnt/famfs/set0
/mnt/famfs/set1
/mnt/famfs/set2
/mnt/famfs/set3
…
```

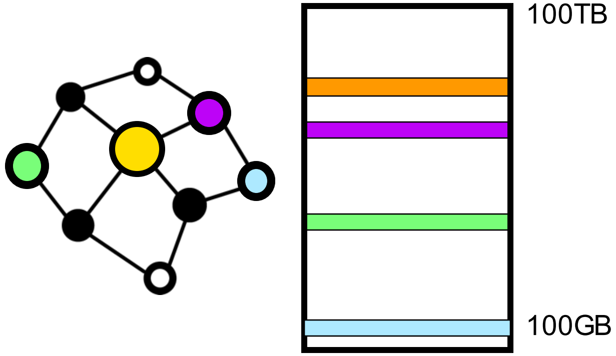
```
mkfs.famfs /dev/dax0.0
famfs mount /dev/dax0.0 /mnt/famfs
famfs cp [-r] <src> <dest>
famfs creat -s <size> <dest>
```

The superpower of memory is low-latency random access

- Famfs with big memory breaks scaling barriers for
  - Graph analytics
  - Rag pipelines
  - In-memory databases and indexes

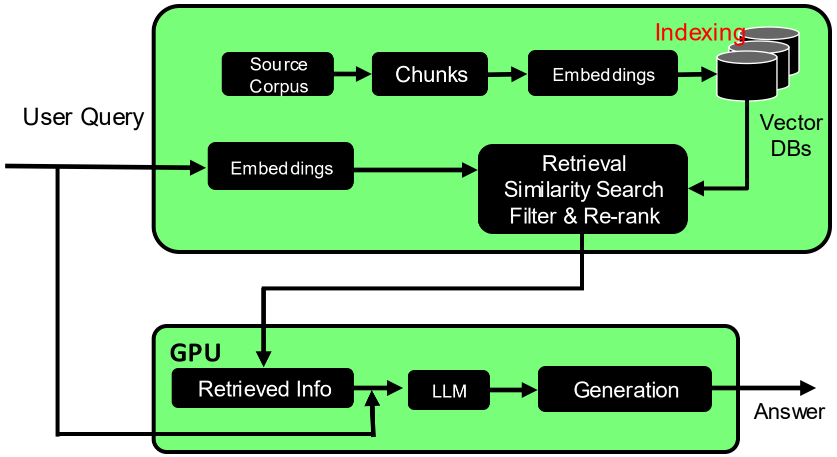
- Graph databases, RAG/LLM pipelines and indexes can scale to 100T and beyond without sharding or demand-paging

Graph analytics and neural networks

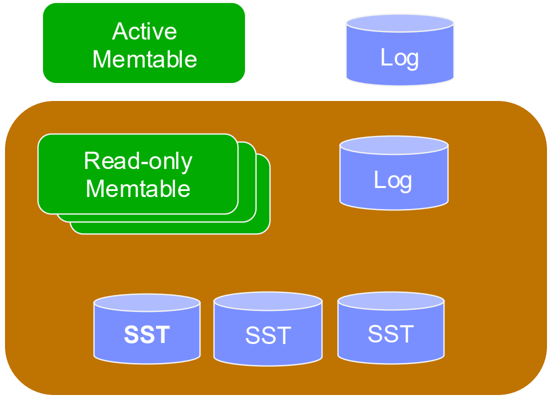


Node/edges/embedding data in storage (PBs)

RAG/LLM Pipeline



Big Data Index



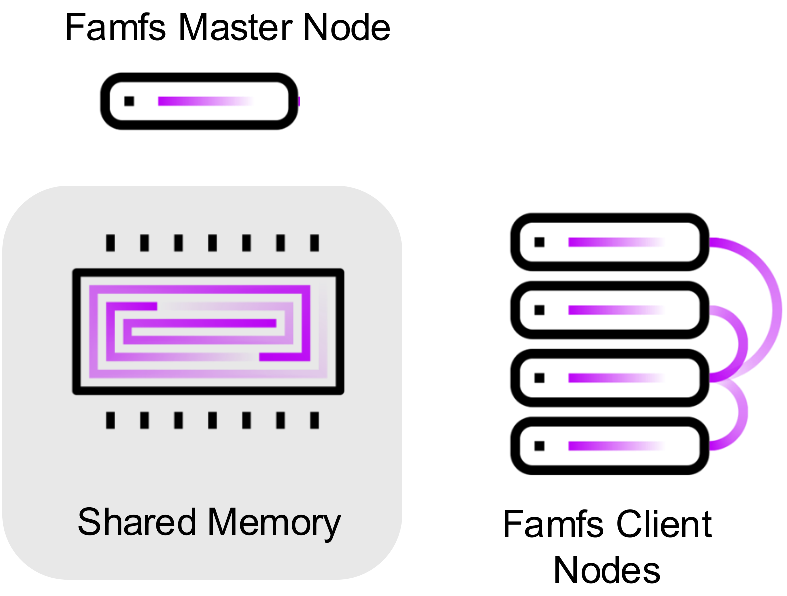
RocksDB Architecture with 100GB-1PB Datasets using key-value store

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!



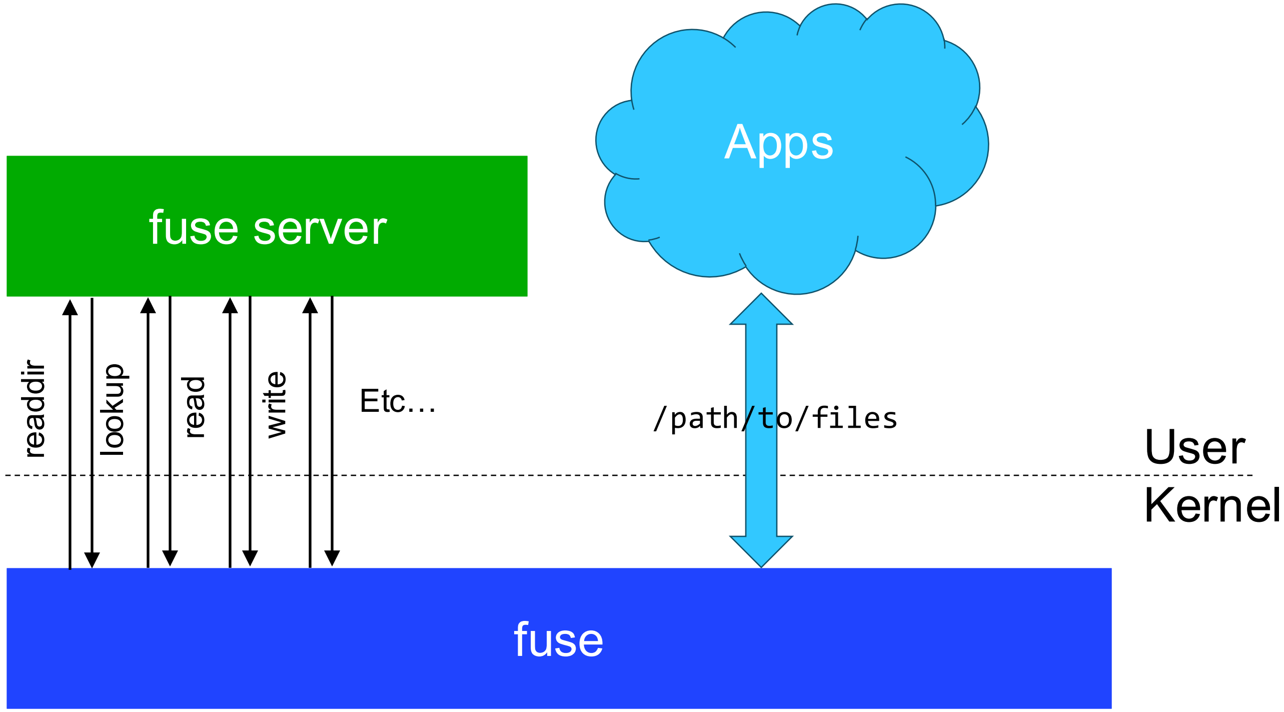
```
mkfs.famfs /dev/dax0.0
famfs mount /dev/dax0.0 /mnt/famfs
famfs cp [-r] <src> <dest>
famfs creat -s <size> <dest>
```

File system layer

Technical details

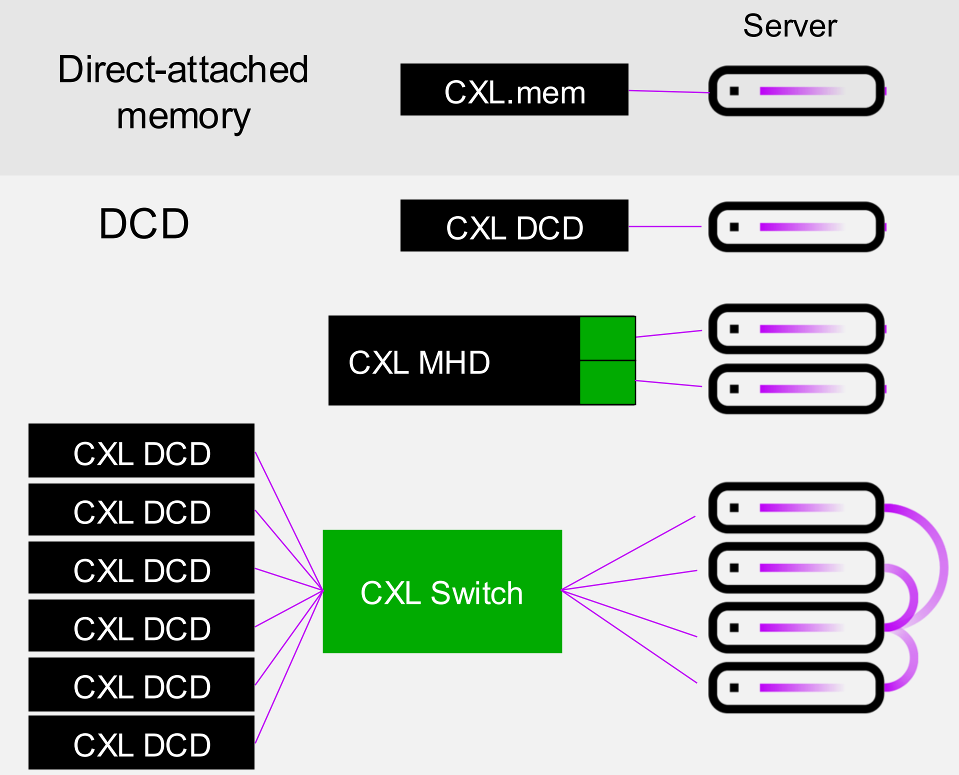
Fuse (File System in User Space)

- 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.



Linux Sees:

`/dev/dax` ⟷ `System-RAM`

Sysadmin can convert between dax and system-ram mode

Linux Sees:

- Nothing, until memory is allocated (Init Dynamic Capacity Add fabric manager command - 7.6.7.6.5)
- A “tagged” dax device for each successful allocation

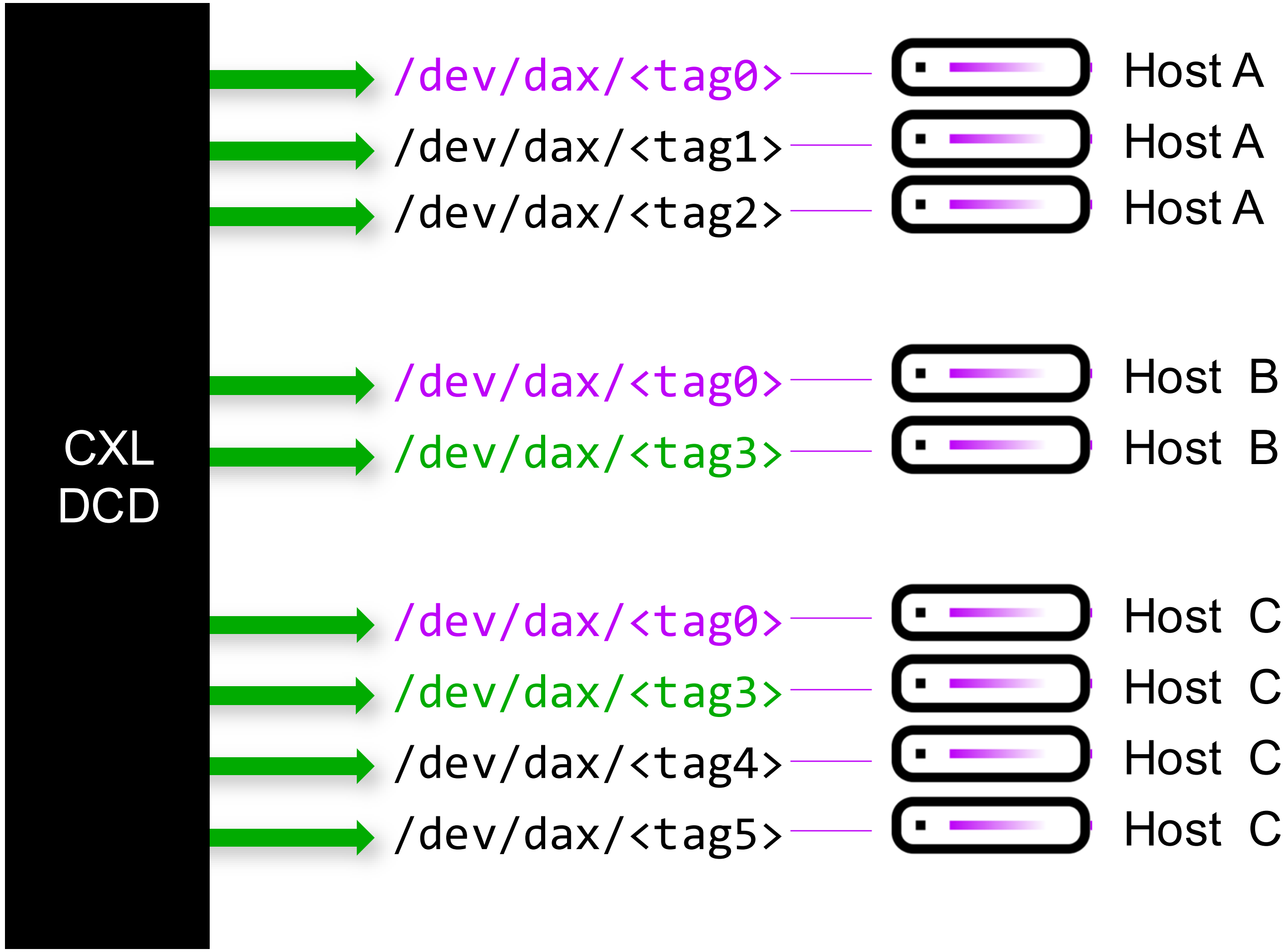
`/dev/dax` ⟷ `System-RAM`

Sysadmin can convert between dax and system-ram mode

- This holds true for DCD in any topology:
  - Direct attached, multi-headed (MHD)
  - LD-FAM or G-FAM

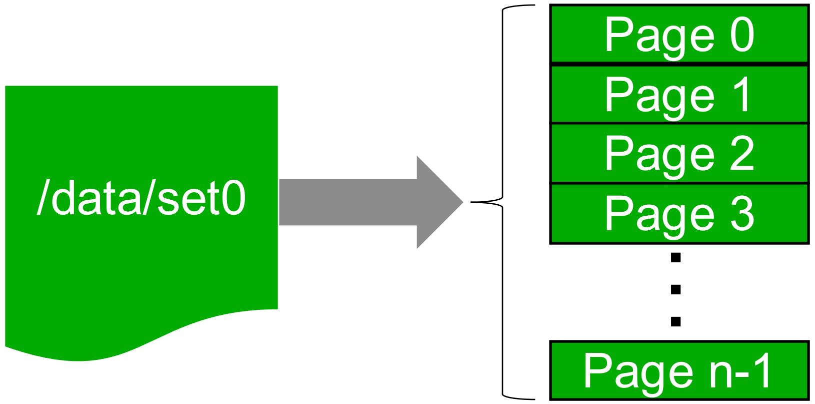
CXL tagged capacity name space

- 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*



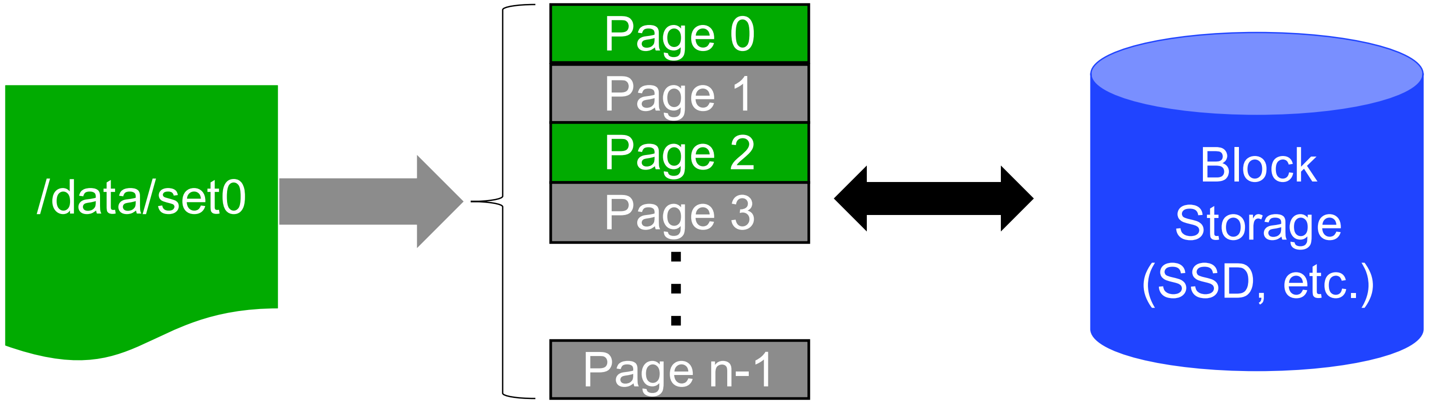
Conventional files as memory

- 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

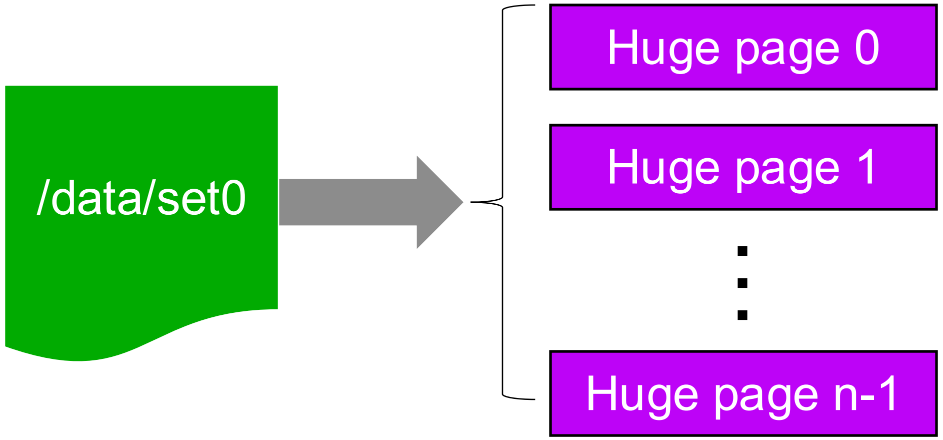
- Files [already] map to memory …if the data is in memory
- When the data is in memory:
  - Read/Write are just memcpy() variants
  - Memory mapping assembles the pages into a virtual address range that is directly accessed as memory
- Many are aware of TLBs and page tables, which resolve a virtual address 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 <u>miss</u> 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

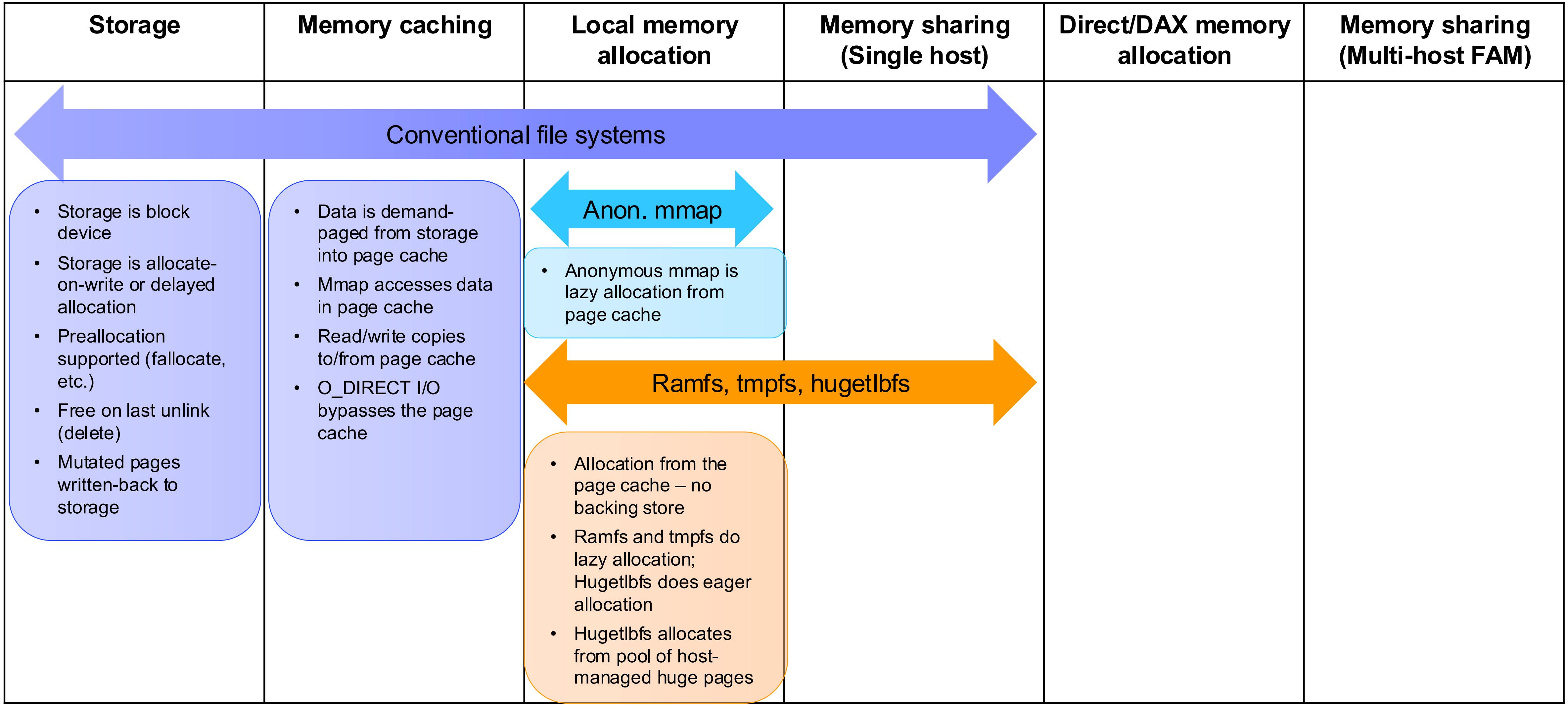
File system / VFS functionality

<table>
<tr><th>Storage</th><th>Memory caching</th><th>Local memory allocation</th><th>Memory sharing (Single host)</th><th>Direct/DAX memory allocation</th><th>Memory sharing (Multi-host FAM)</th></tr>
<tr><td colspan="4">Conventional file systems</td><td></td><td></td></tr>
<tr><td>• 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</td><td>• 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</td><td></td><td></td><td></td><td></td></tr>
</table>

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					
		Anon. mmap			
• 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	• Anonymous mmap is lazy allocation from 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
Anon. mmap
Ramfs, tmpfs, hugetlbfs

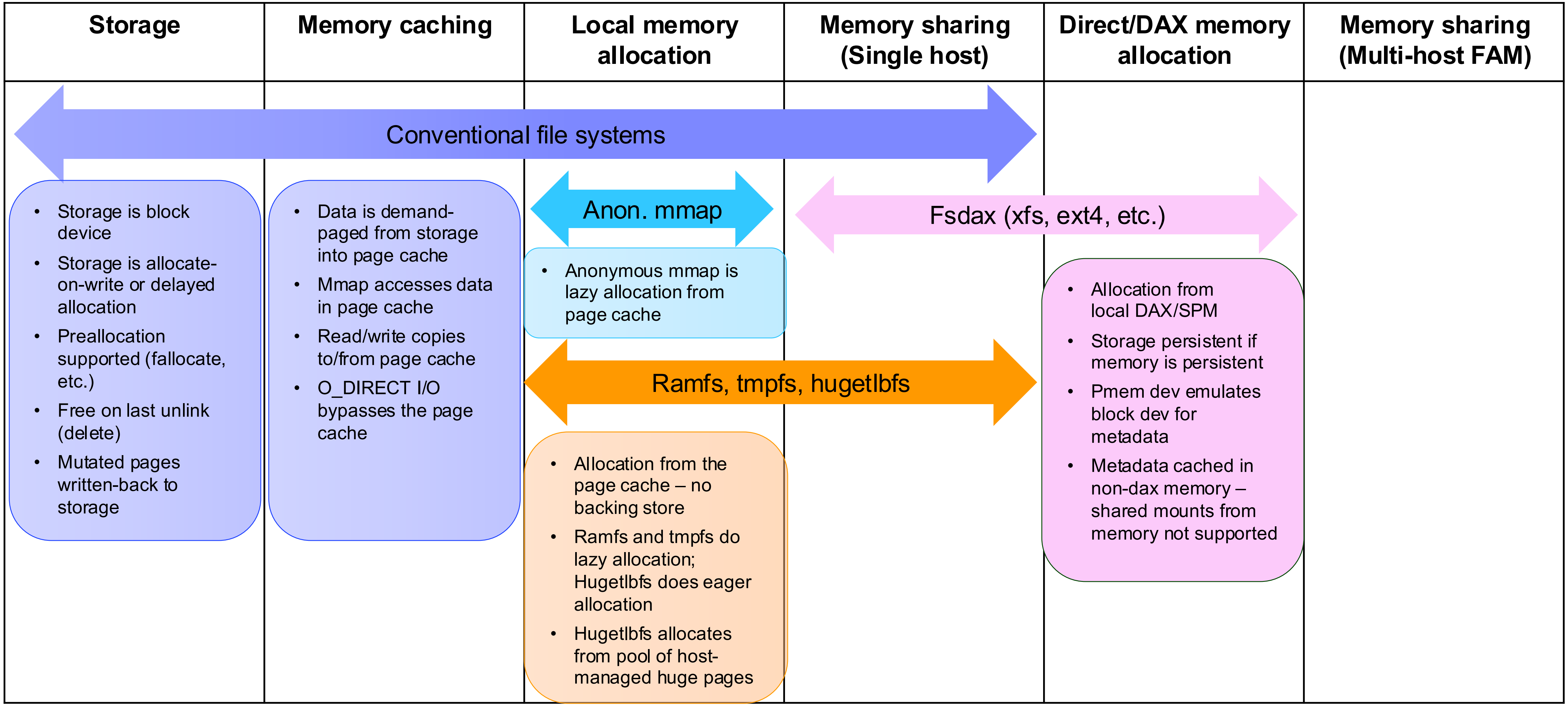
- Storage is block device
- Storage is allocate-on-write or delayed allocation
- Preallocation supported (fallocate, etc.)
- Free on last unlink (delete)
- Mutated pages written-back to storage

- 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

- Anonymous mmap is lazy allocation from page cache

- Allocation from the page cache – no backing store
- Ramfs and tmpfs do lazy allocation; Hugetlbfs does eager allocation
- Hugetlbfs allocates from pool of host-managed huge pages

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
Anon. mmap
Fsdax (xfs, ext4, etc.)
Ramfs, tmpfs, hugetlbfs

Storage

- 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

Memory caching

- 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

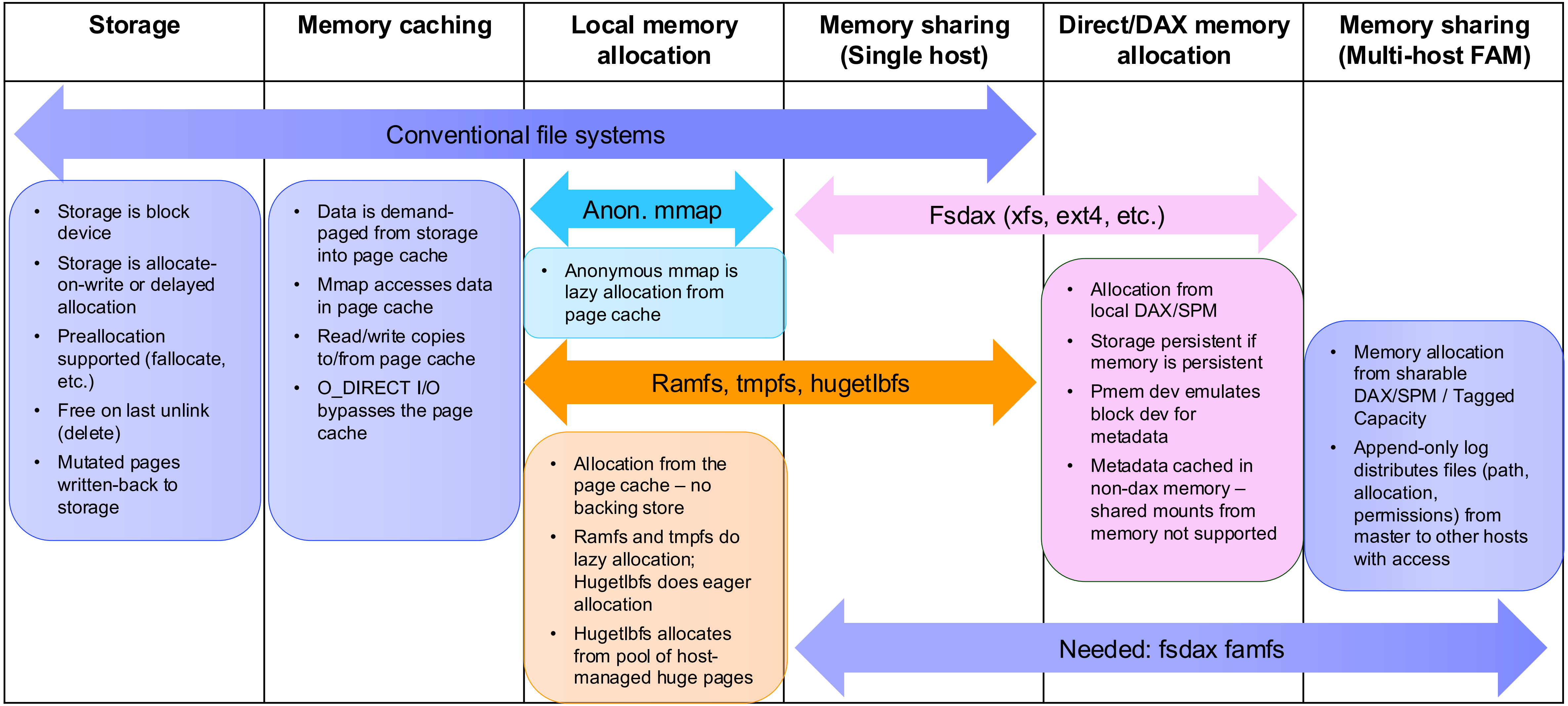
Ramfs, tmpfs, hugetlbfs

- Allocation from the page cache – no backing store
- Ramfs and tmpfs do lazy allocation; Hugetlbfs does eager allocation
- Hugetlbfs allocates from pool of host-managed huge pages

Fsdax (xfs, ext4, etc.)

- Allocation from local DAX/SPM
- Storage persistent if memory is persistent
- Pmem dev emulates block dev for metadata
- Metadata cached in non-dax memory – shared mounts from memory not supported

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

Ramfs, tmpfs, hugetlbfs

- Allocation from the page cache – no backing store
- Ramfs and tmpfs do lazy allocation; Hugetlbfs does eager allocation
- Hugetlbfs allocates from pool of host-managed huge pages

Fsdax (xfs, ext4, etc.)

- Allocation from local DAX/SPM
- Storage persistent if memory is persistent
- Pmem dev emulates block dev for metadata
- Metadata cached in non-dax memory – shared mounts from memory not supported

- Memory allocation from sharable DAX/SPM / Tagged Capacity
- Append-only log distributes files (path, allocation, permissions) from master to other hosts with access

Needed: fsdax famfs

How does Famfs work

Backup

micron Intelligence Accelerated™