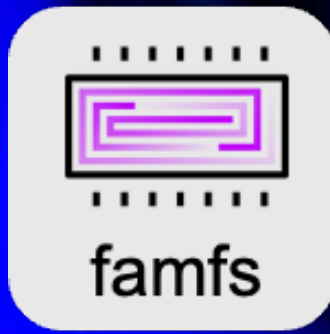
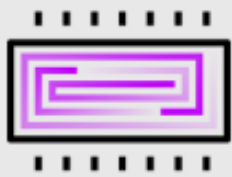


# Famfs: Open Source Scale-Out Shared Memory File System



Aug 2025  
John Groves  
Technical Director

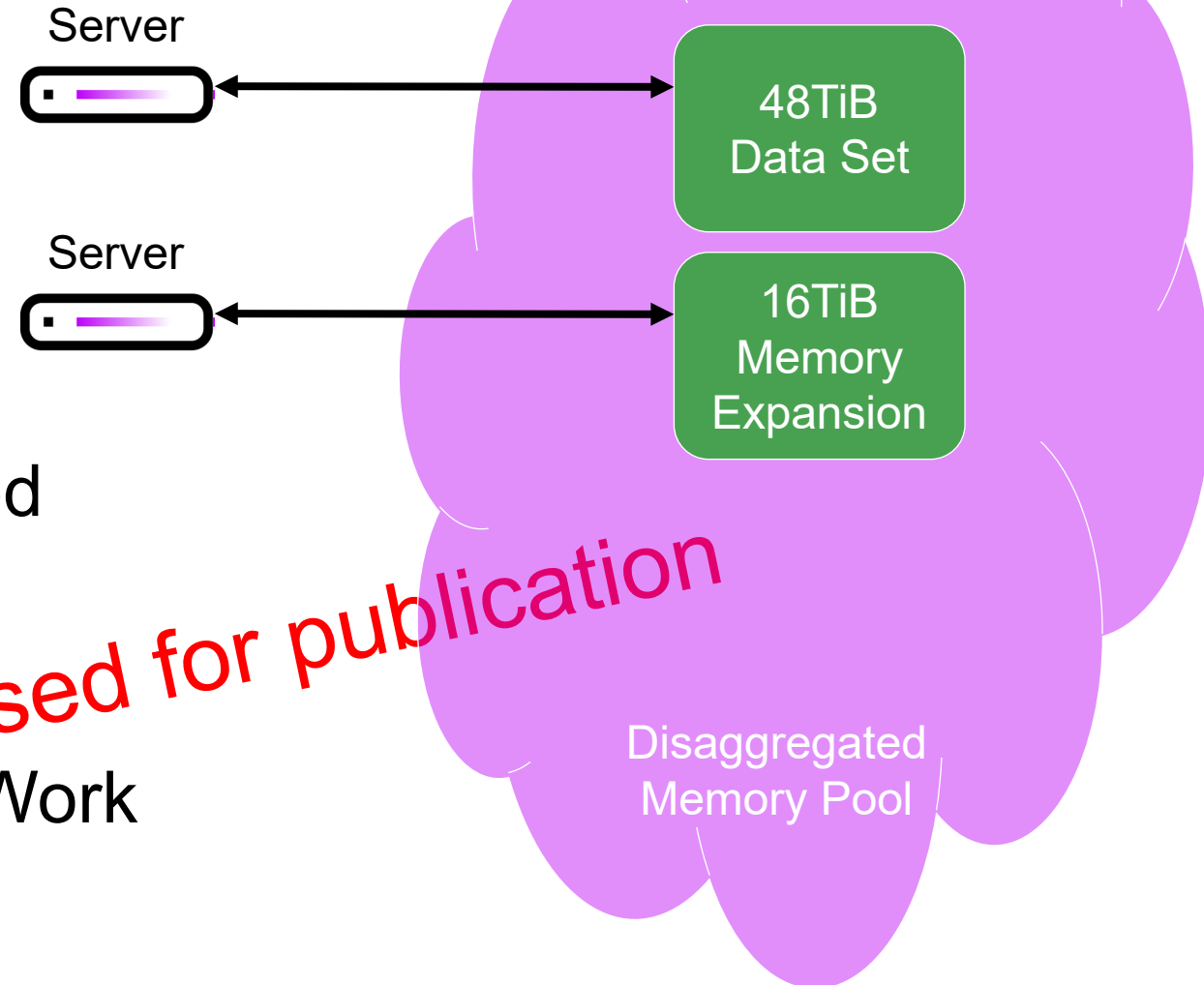
micron

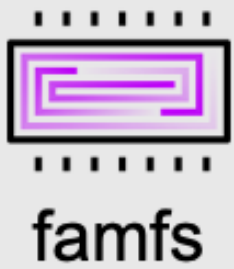


famfs

# Get Ready for Big Pools of Disaggregated Memory

- Larger data sets fit in memory
  - Sharding can be avoided
  - Shared Memory is effectively “Deduplicated”
  - A Good Access Method is Needed
- The File System Interface is Quite Natural for Shared Memory
  - But Existing File Systems Don’t Work Here

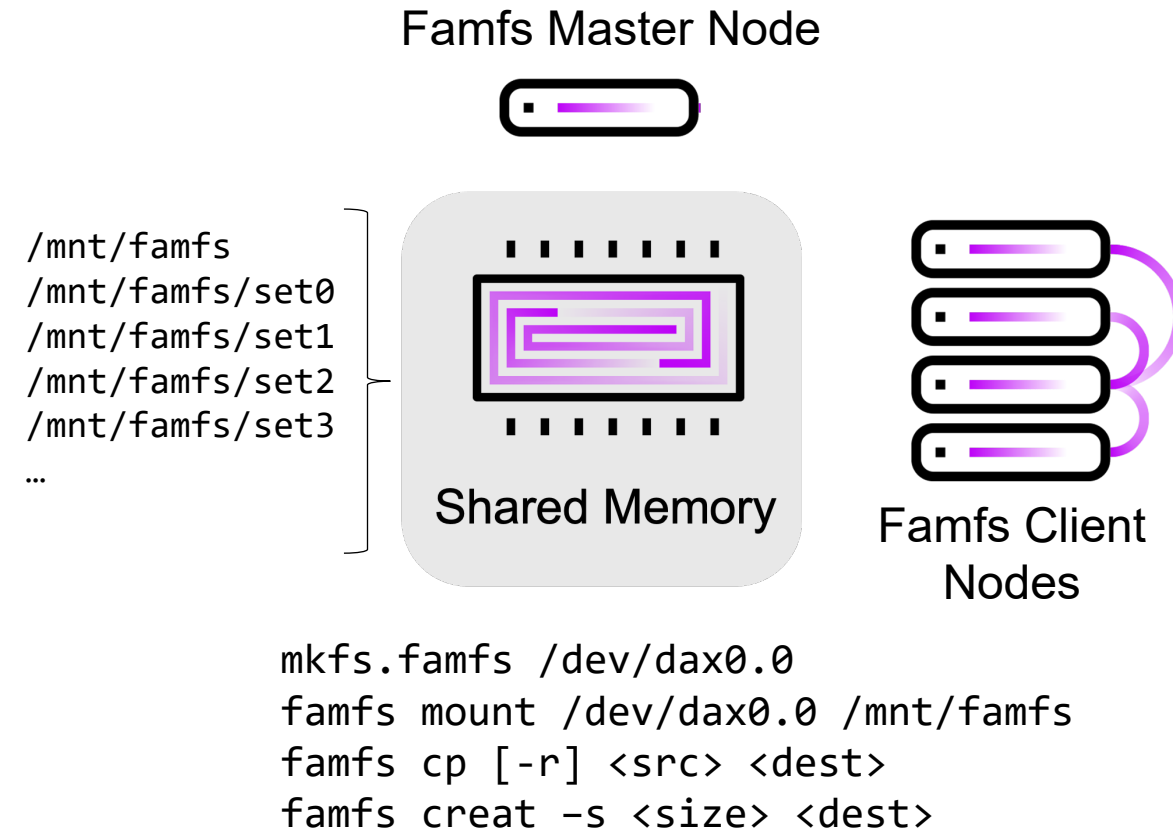


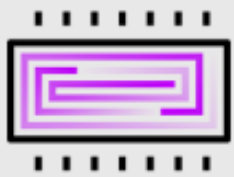


# Famfs Organizes Shared Memory as a Scale-Out File System

Enabling shared memory for all apps that can use files !!

- Memory is accessible as files
  - Write/read become memcpy
  - Mmap maps the memory for byte-level 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



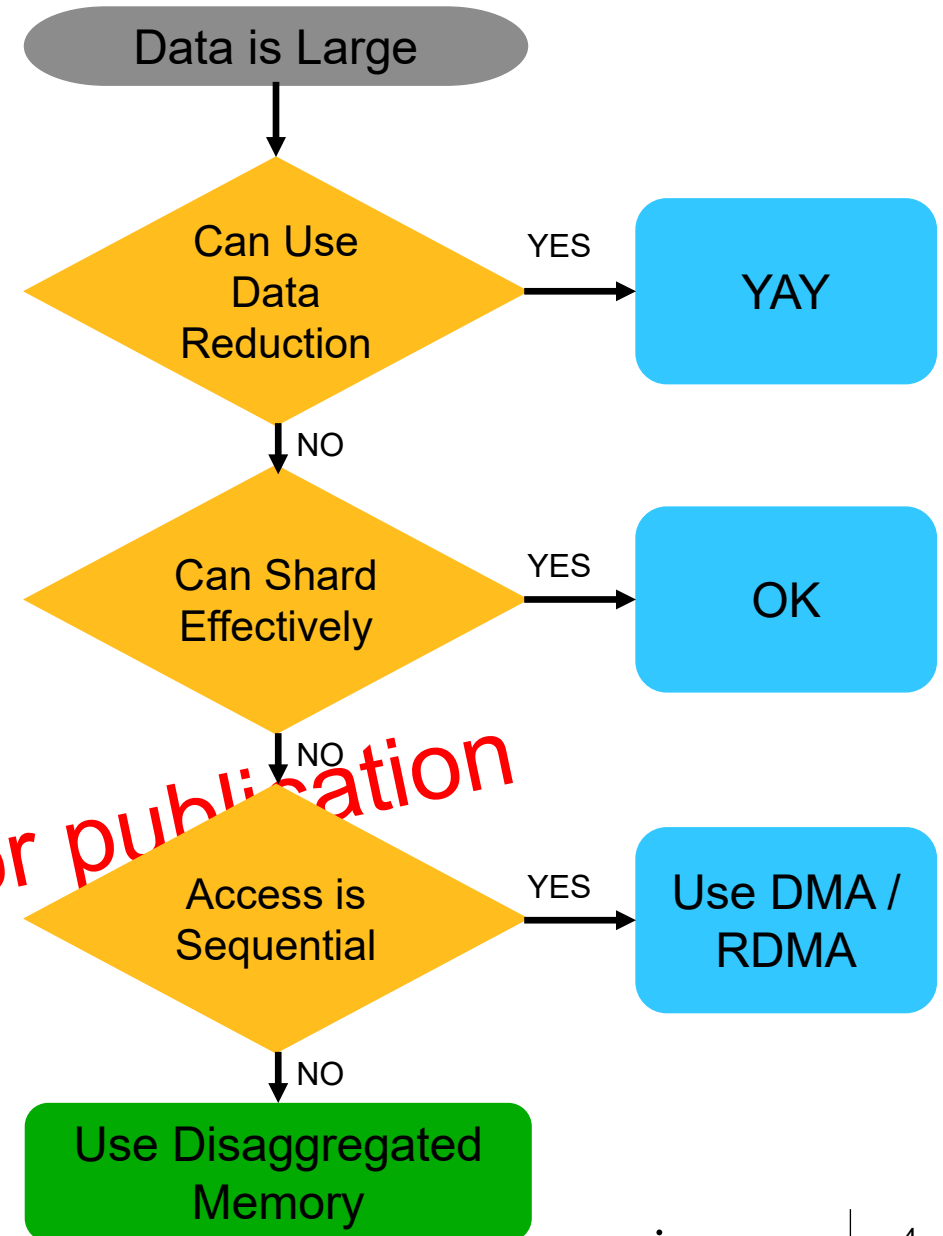


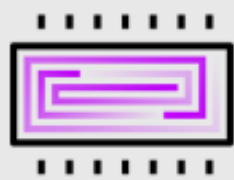
famfs

# What if Data is [Much] Bigger than Memory

- Some data can be reduced effectively
- Some data can be sharded 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)

*Draft – will be revised for publication*

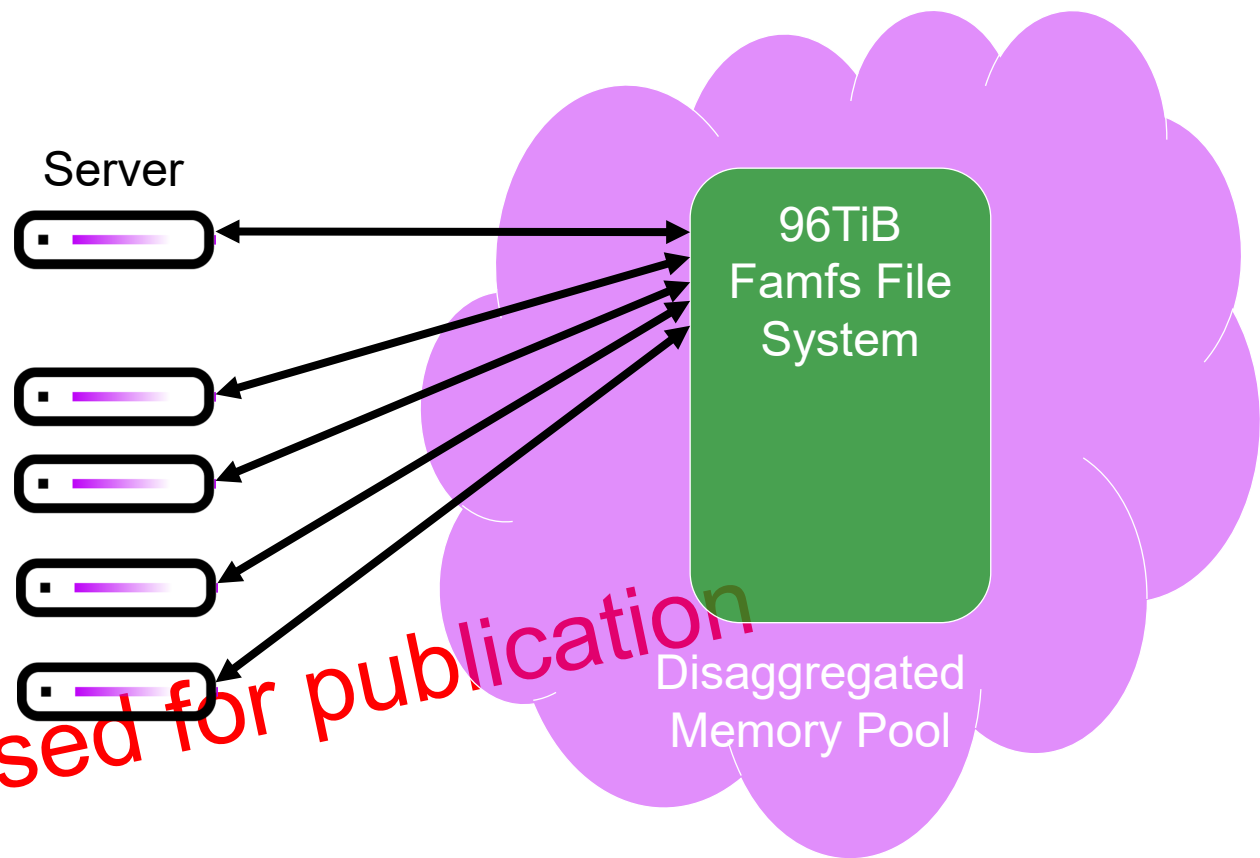




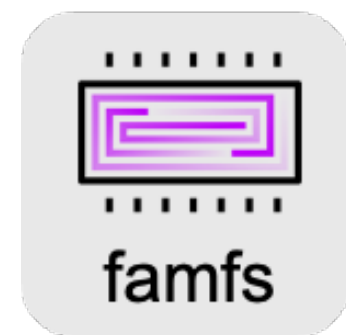
famfs

# Disaggregated Memory can be Much Larger than System RAM

- Re-think What Problems Fit in Memory
- Multiple nodes can share data in memory
  - Memory is “deduplicated”
  - Compute can still scale out without sharding or duplication



# Background: CXL Memory Usage Models

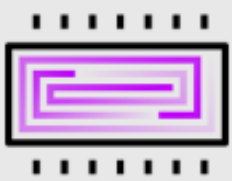


## Pooling

- 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

- The hardware supports this (CXL3, DCD, etc.)
- 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
- These cases include
  - Both concurrent and sequential sharing
  - Other use cases that use Linux memory-mgmt



famfs

# 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
- Famfs Files Can be Interleaved Across Many CXL Memory Devices
  - Famfs has no constraints about DPA ranges

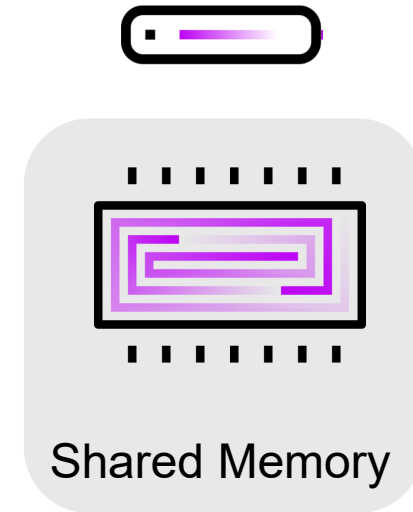
**Draft – will be revised for publication**

# Famfs Status: On Track for Linux Upstream in 2025

- Nov 2023 – [Introduced famfs at the Linux Plumbers Conference](#)
- Feb 2024 – [Famfs V1 Linux patch series released](#)
- April 2024 – [Famfs V2 Linux patch series released](#)
- May 2024 – [Famfs session at LSFMM](#)  
(Linux Storage, File System and Memory Management summit)
  - Consensus: Famfs should be merged into fuse
  - Work is in progress, in collaboration with the fuse maintainers
- 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 & fuse port is imminent
  - Famfs documentation:

<https://github.com/cxl-micron-reskit/famfs/blob/master/README.md>

Famfs Master Node



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



# Famfs Use Cases

Great leverage for “data frames” space (analytics + AI) and in-memory database applications

- All apps & tools can access data frames in files
- Data analytics and AI applications share a lot of infrastructure in the “data frames” and data lake space
- Data frames ecosystem already uses shared data sets (Already accesses data frames as memory-mapped files)
- Many of these use cases are read-only during the data-sharing portion of the work-flow
- Putting shared data frames in famfs enables CXL memory without requiring app modifications
  - (workflows may need to be modified, but this class of apps can easily do that)
- Data lake / data file / in-memory database formats



RocksDB



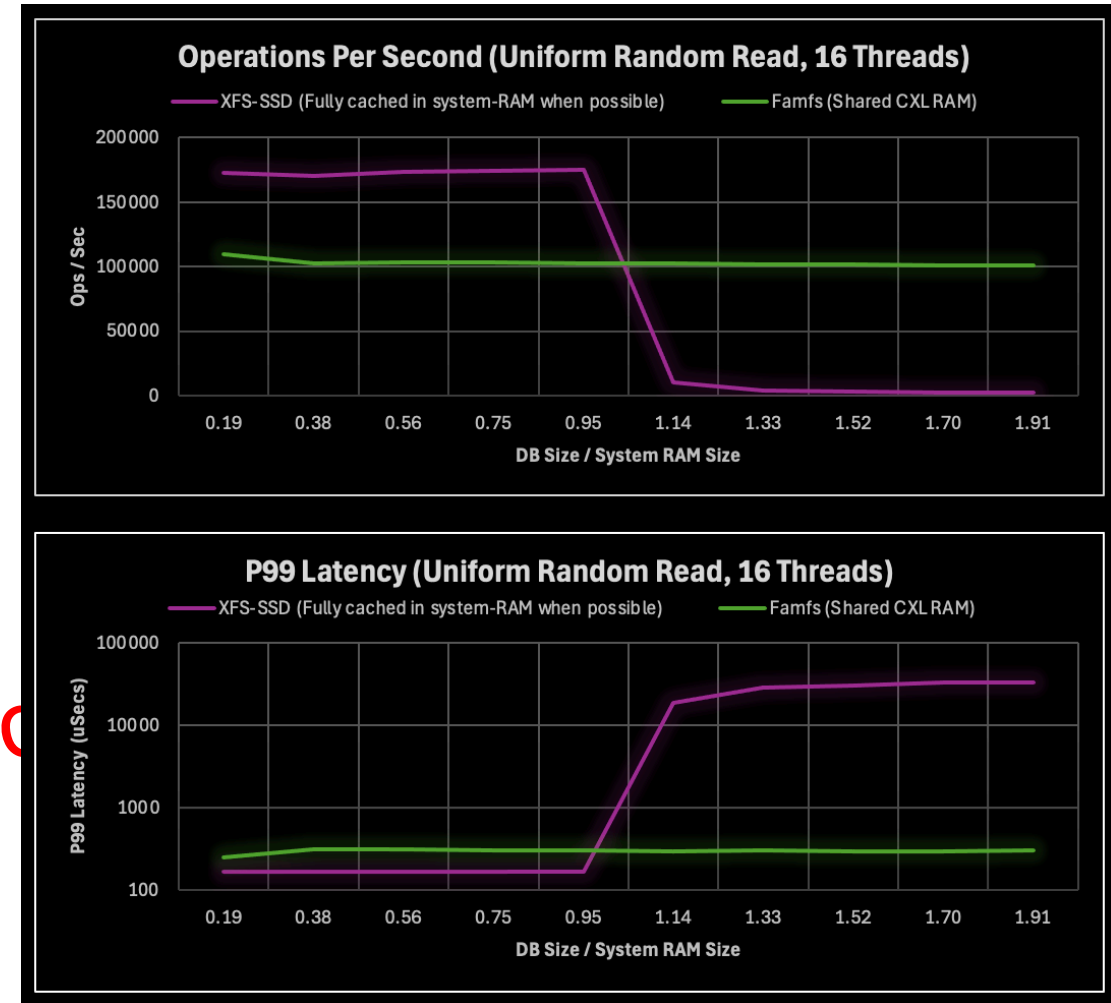
DuckDB



# Famfs: Bigger Data In Shared Memory

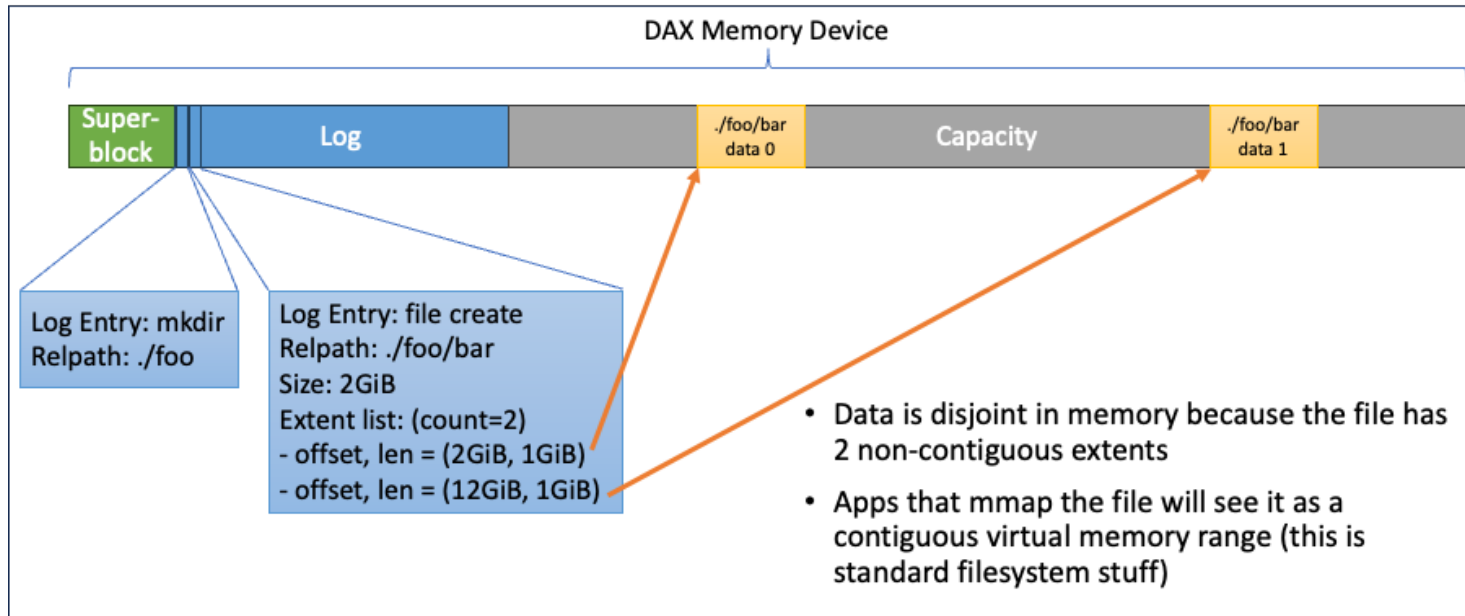
- RocksDB read-only benchmark
- Famfs benchmarks (Green)
  - RocksDB database stored in famfs
  - RocksDB instances on multiple hosts can share the same files/memory
  - No modifications to RocksDB (famfs is just files)
- Control Group (Purple)
  - RocksDB database stored in xfs backed by nvme
  - Cached in DDR; Performance great then it fits in mem
- This data was shown at FMS '24
- Benefits:4
  - Data is de-duplicated
  - Or sharding / shuffling is avoided
  - Cache line access (less read amplification)

Note: Charts will be replaced with more recent data

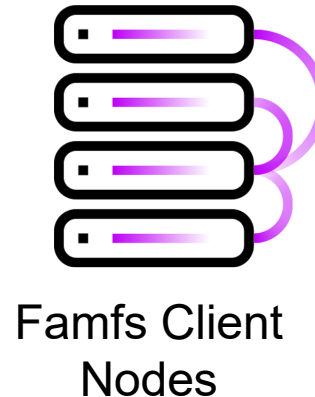
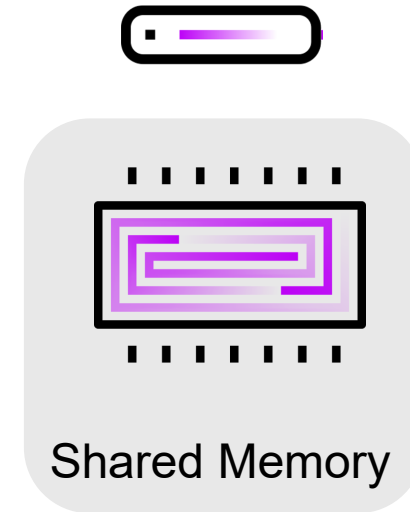


# Famfs Architecture

- 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



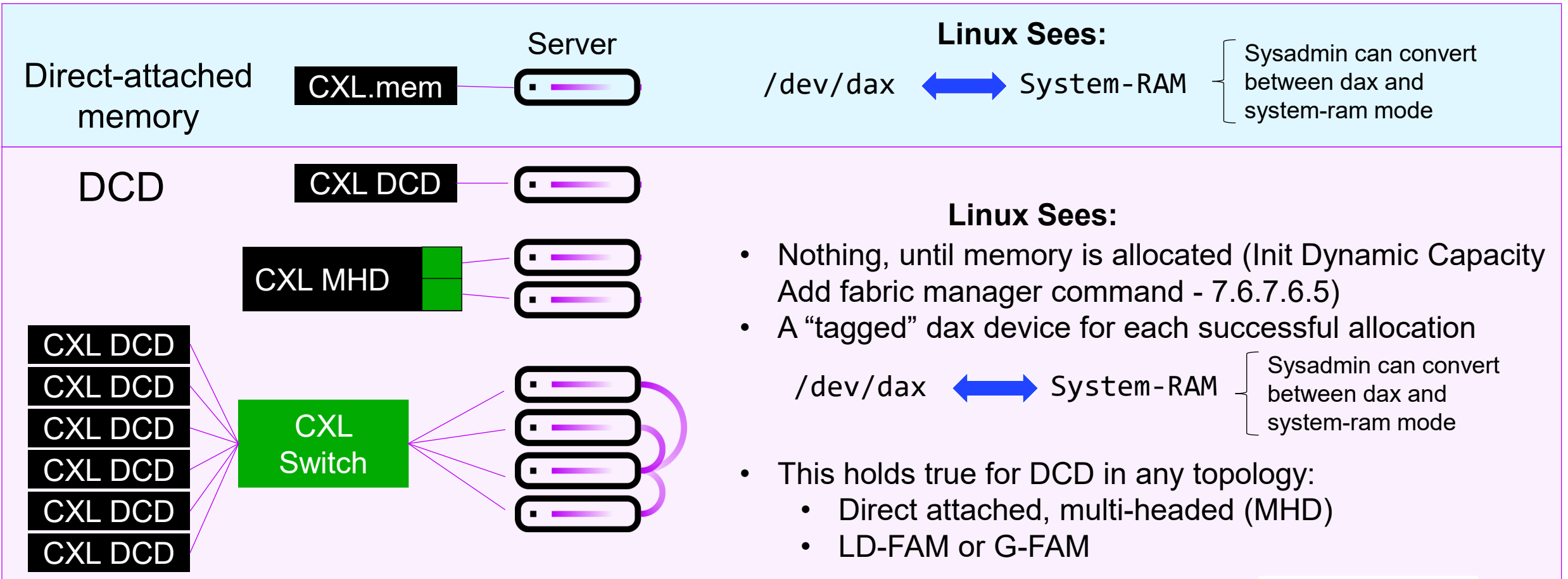
Famfs Master Node



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

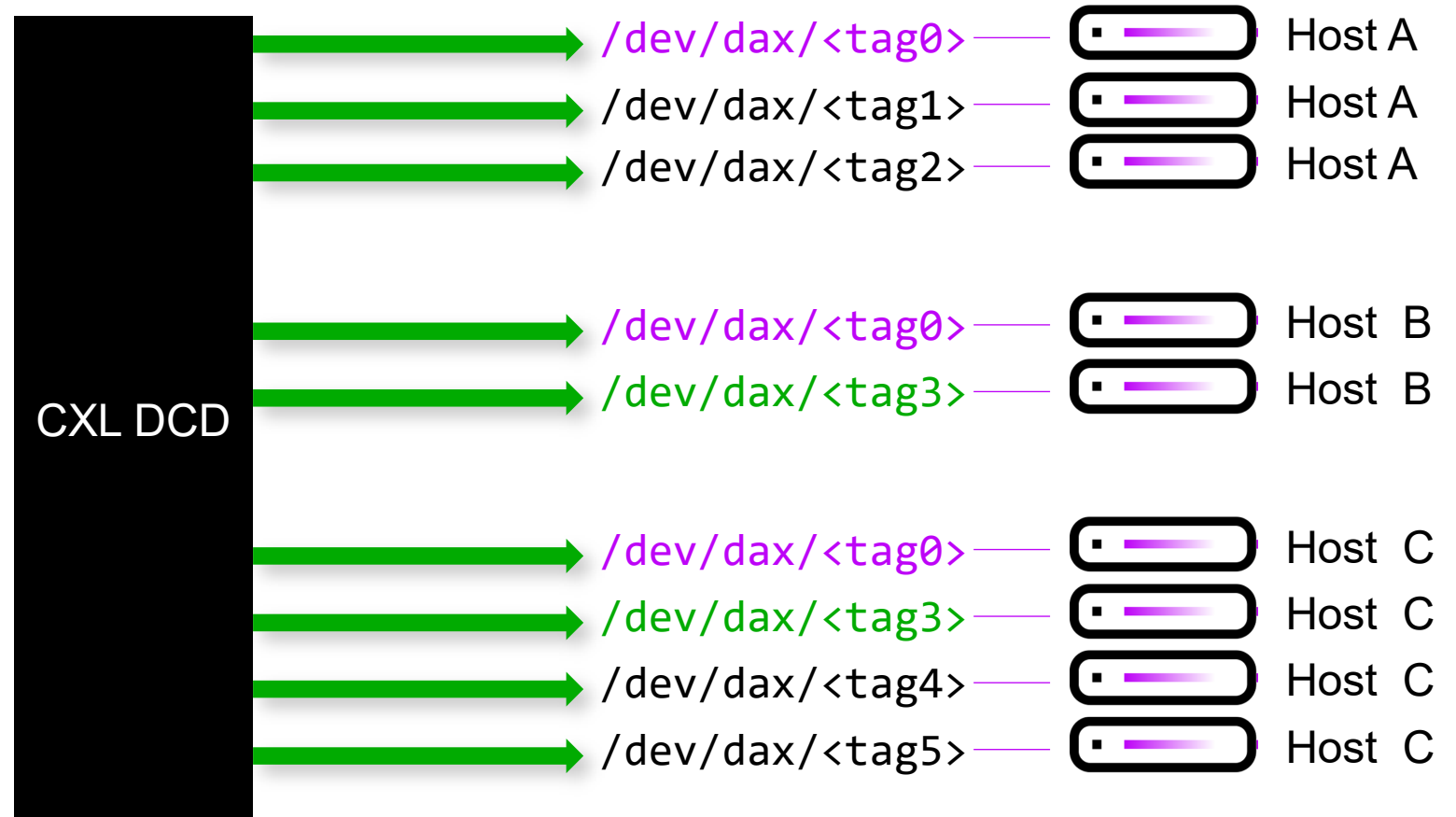
# Background: CXL Memory Sharing Topology

- 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\*

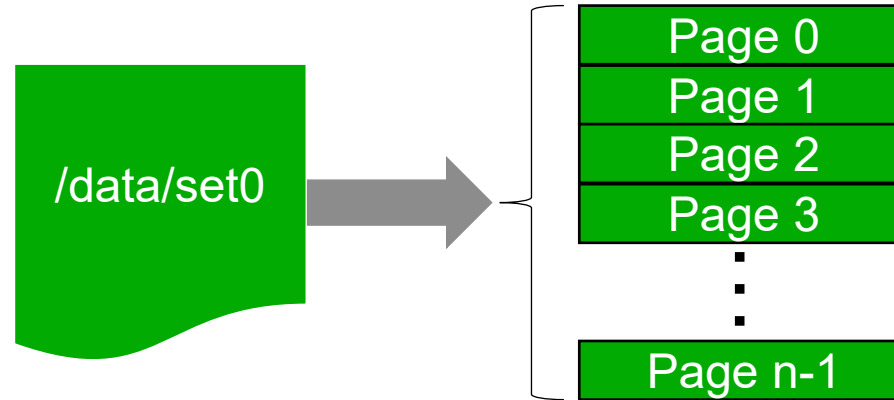


# File System Layer

Technical Details

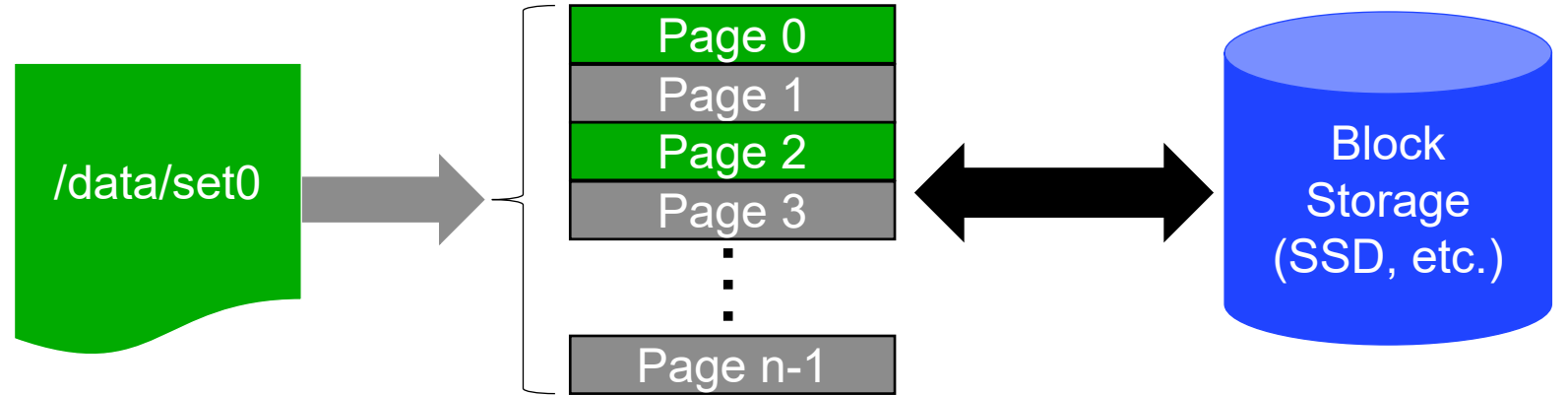
# Conventional Files as Memory

- 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

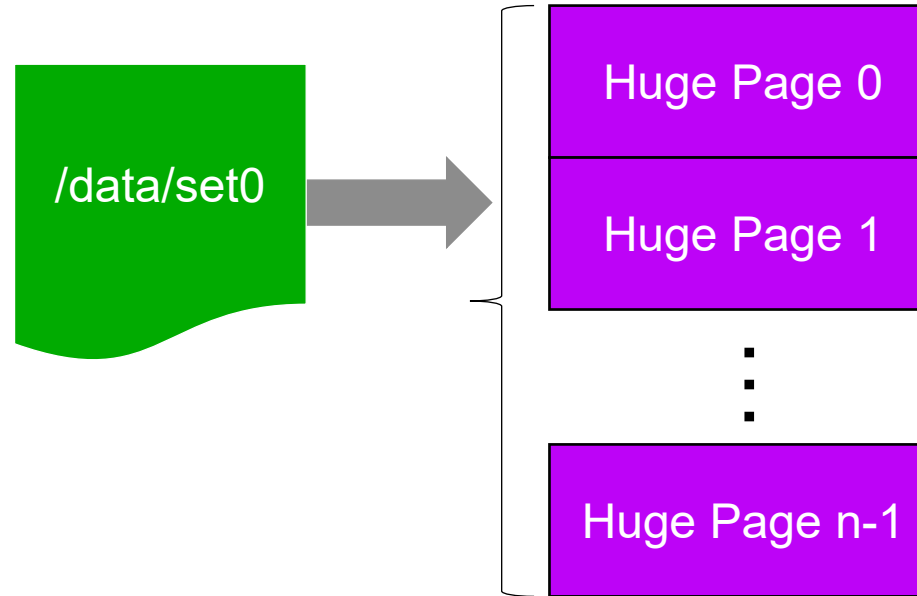


- Conventional file systems sparsely cache pages from a files backing store
  - Meaning a `fault()` call might have to allocate memory and retrieve data from backing storage
- Pages that are cached (green) are accessed as memory
- Pages that are not in cache (gray) must be faulted in from backing store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



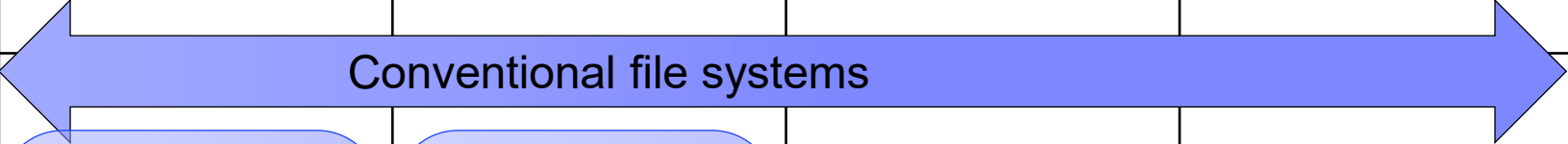
# Famfs Files as Memory

- 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

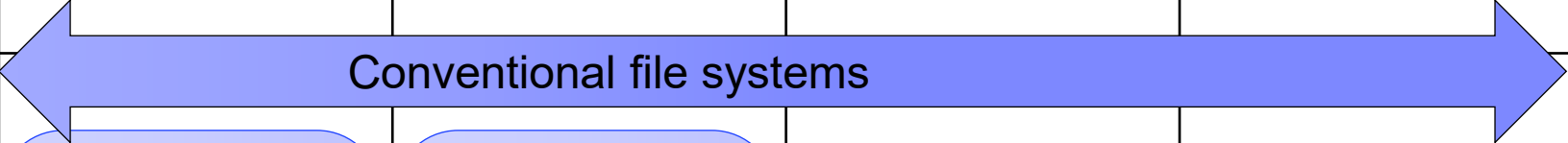
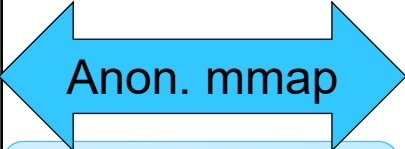


- Famfs is not sparse; files are always fully mapped to memory
- Famfs data lives in [sharable] dax memory devices
- Huge page mapping reduces virtual memory mapping overhead by 512x
- Since the backing memory is not sparse, there is no downside to huge page mapping

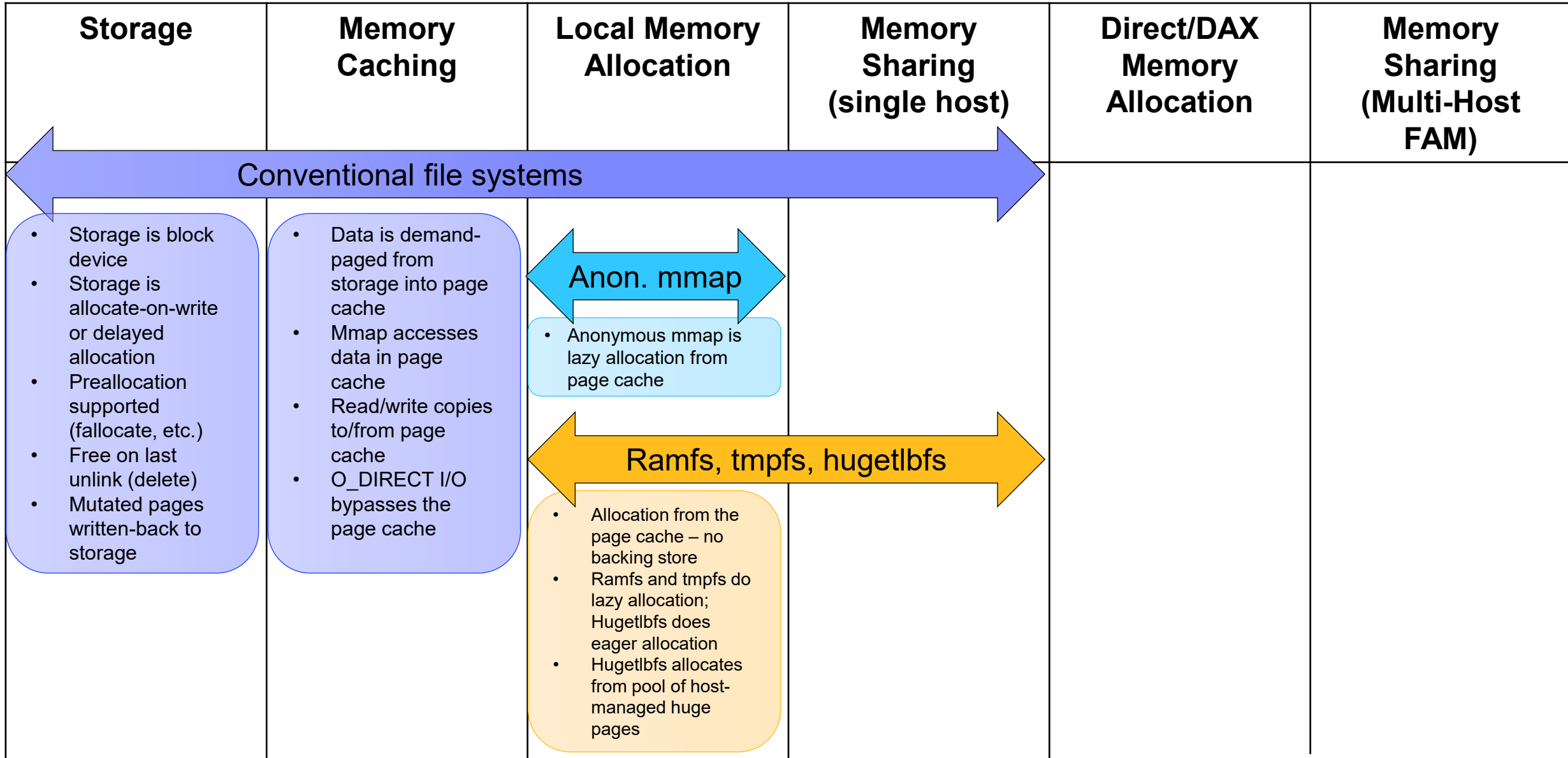
# File System / VFS Functionality

Storage	Memory Caching	Local Memory Allocation	Memory Sharing (single host)	Direct/DAX Memory Allocation	Memory Sharing (Multi-Host FAM)
					
<ul style="list-style-type: none"> <li>• Storage is block device</li> <li>• Storage is allocate-on-write or delayed allocation</li> <li>• Preallocation supported (fallocate, etc.)</li> <li>• Free on last unlink (delete)</li> <li>• Mutated pages written-back to storage</li> </ul>	<ul style="list-style-type: none"> <li>• Data is demand-paged from storage into page cache</li> <li>• Mmap accesses data in page cache</li> <li>• Read/write copies to/from page cache</li> <li>• O_DIRECT I/O bypasses the page cache</li> </ul>				

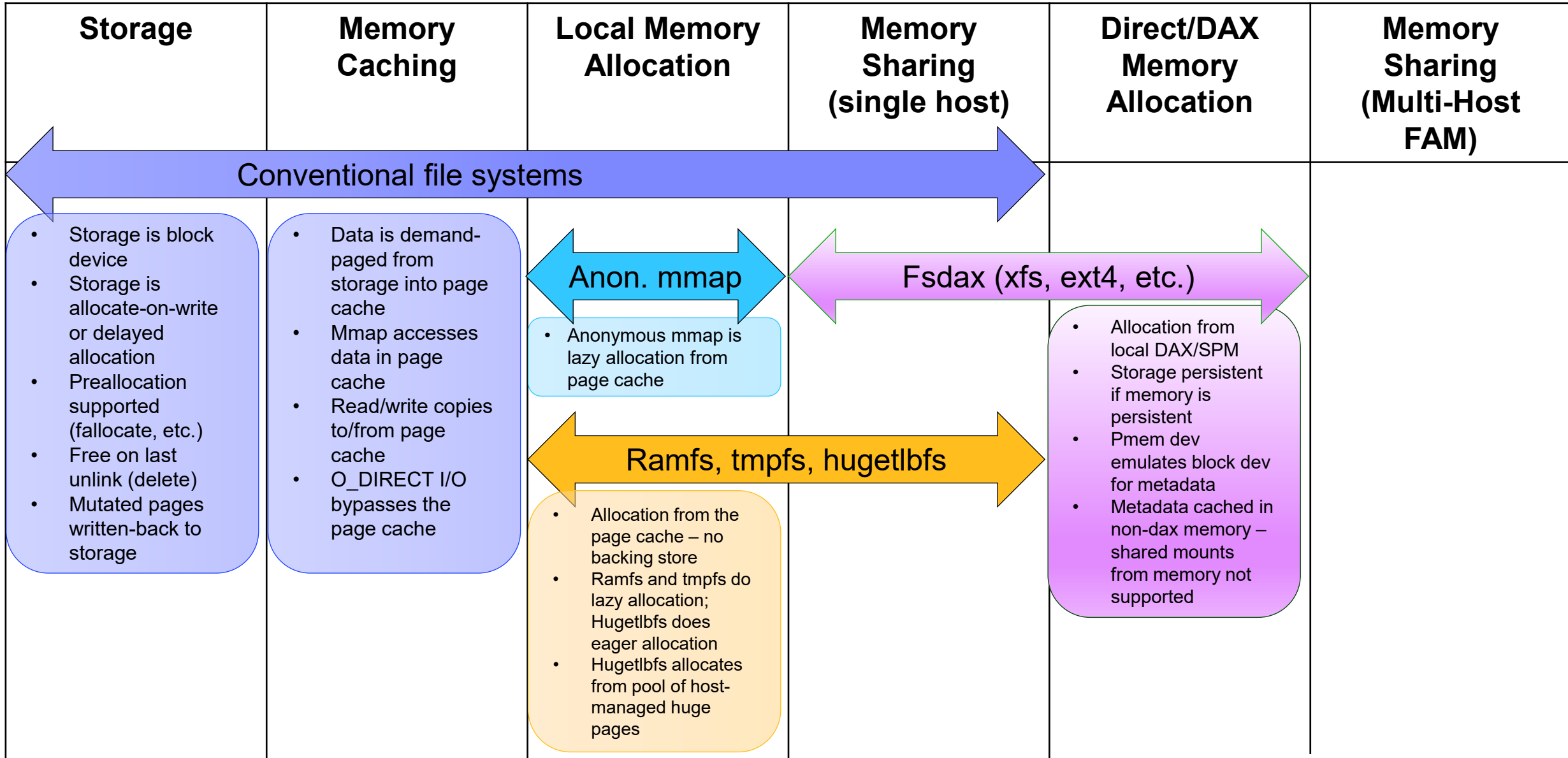
# File System / VFS Functionality

Storage	Memory Caching	Local Memory Allocation	Memory Sharing (single host)	Direct/DAX Memory Allocation	Memory Sharing (Multi-Host FAM)
					
<ul style="list-style-type: none"> <li>Storage is block device</li> <li>Storage is allocate-on-write or delayed allocation</li> <li>Preallocation supported (fallocate, etc.)</li> <li>Free on last unlink (delete)</li> <li>Mutated pages written-back to storage</li> </ul>	<ul style="list-style-type: none"> <li>Data is demand-paged from storage into page cache</li> <li>Mmap accesses data in page cache</li> <li>Read/write copies to/from page cache</li> <li>O_DIRECT I/O bypasses the page cache</li> </ul>	 <ul style="list-style-type: none"> <li>Anonymous mmap is lazy allocation from page cache</li> </ul>			

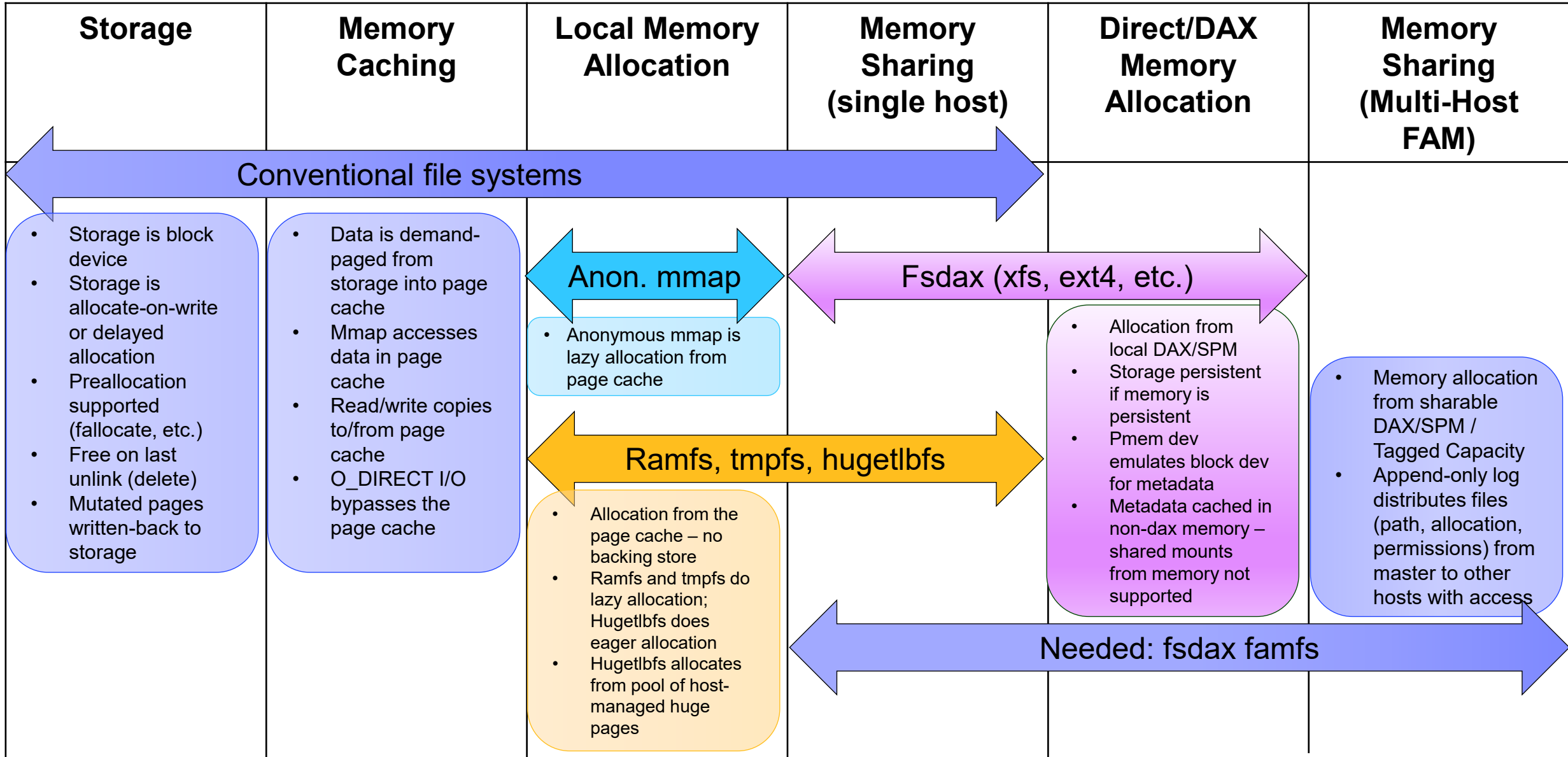
# File System / VFS Functionality



# File System / VFS Functionality



# File System / VFS Functionality

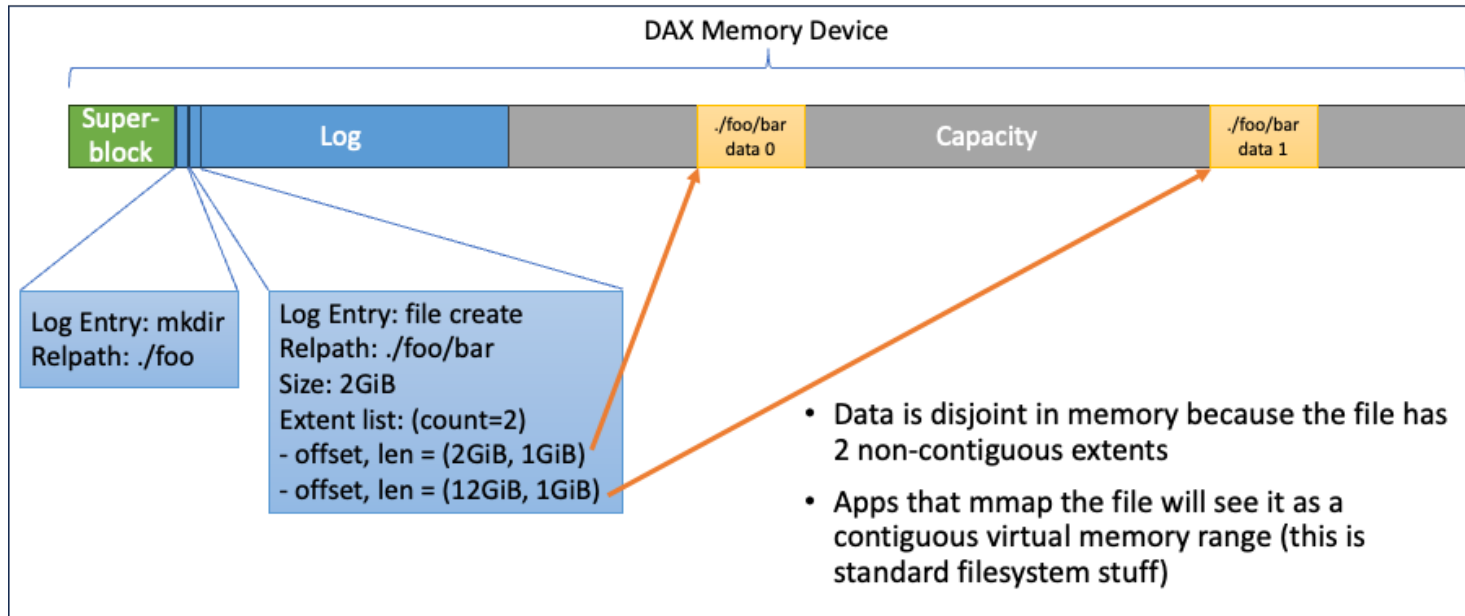


# How Does Famfs Work

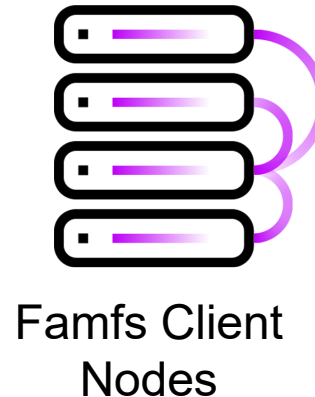
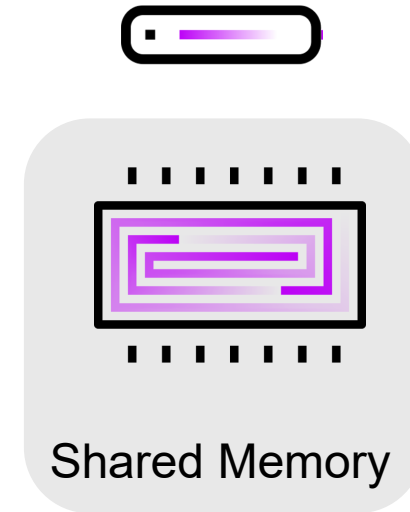
Backup

# Famfs Architecture

- 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



Famfs Master Node



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



# Famfs Striped Files

Famfs has supported striped files since August 2024

- An extent describes each *strip*  
(daxdev\_index, offset, length)

Famfs striped file map

Chunk = 0 Strip = 0 Stripe = 0	Chunk = 1 Strip = 1 Stripe = 0	Chunk = 2 Strip = 2 Stripe = 0	Chunk = 3 Strip = 3 Stripe = 0
Chunk = 4 Strip = 0 Stripe = 1	Chunk = 5 Strip = 1 Stripe = 1	Chunk = 6 Strip = 2 Stripe = 1	Chunk = 7 Strip = 3 Stripe = 1
Chunk = 5 Strip = 0 Stripe = 2	Chunk = 6 Strip = 1 Stripe = 2	Chunk = 7 Strip = 2 Stripe = 2	Chunk = 8 Strip = 3 Stripe = 2

# Famfs Striped Files

Famfs has supported striped files since August 2024

- An extent describes each *strip*  
(daxdev\_index, offset, length)

Famfs striped file map

Strip 0

Chunk = 0 Strip = 0 Stripe = 0	Chunk = 1 Strip = 1 Stripe = 0	Chunk = 2 Strip = 2 Stripe = 0	Chunk = 3 Strip = 3 Stripe = 0
Chunk = 4 Strip = 0 Stripe = 1	Chunk = 5 Strip = 1 Stripe = 1	Chunk = 6 Strip = 2 Stripe = 1	Chunk = 7 Strip = 3 Stripe = 1
Chunk = 5 Strip = 0 Stripe = 2	Chunk = 6 Strip = 1 Stripe = 2	Chunk = 7 Strip = 2 Stripe = 2	Chunk = 8 Strip = 3 Stripe = 2

# Famfs Striped Files

Famfs has supported striped files since August 2024

- An extent describes each *strip* (daxdev\_index, offset, length)

Famfs striped file map

Strip 1

Chunk = 0 Strip = 0 Stripe = 0	Chunk = 1 Strip = 1 Stripe = 0	Chunk = 2 Strip = 2 Stripe = 0	Chunk = 3 Strip = 3 Stripe = 0
Chunk = 4 Strip = 0 Stripe = 1	Chunk = 5 Strip = 1 Stripe = 1	Chunk = 6 Strip = 2 Stripe = 1	Chunk = 7 Strip = 3 Stripe = 1
Chunk = 5 Strip = 0 Stripe = 2	Chunk = 6 Strip = 1 Stripe = 2	Chunk = 7 Strip = 2 Stripe = 2	Chunk = 8 Strip = 3 Stripe = 2

# Famfs Striped Files

Famfs has supported striped files since August 2024

- An extent describes each *strip* (daxdev\_index, offset, length)
- Because chunks and stripes are fixed size, resolving a file offset to a (strip, offset) pair is order 1
- Chunks must be aligned page-size multiples
- Strips go on separate memory devices
- Famfs can hide discontinuities for free

Famfs striped file map

Chunk = 0 Strip = 0 Stripe = 0	Chunk = 1 Strip = 1 Stripe = 0	Chunk = 2 Strip = 2 Stripe = 0	Chunk = 3 Strip = 3 Stripe = 0
Chunk = 4 Strip = 0 Stripe = 1	Chunk = 5 Strip = 1 Stripe = 1	Chunk = 6 Strip = 2 Stripe = 1	Chunk = 7 Strip = 3 Stripe = 1
Chunk = 5 Strip = 0 Stripe = 2	Chunk = 6 Strip = 1 Stripe = 2	Chunk = 7 Strip = 2 Stripe = 2	Chunk = 8 Strip = 3 Stripe = 2

The Micron logo is displayed in a white, lowercase, sans-serif font against a black background. The letter 'm' is stylized with a thick, rounded vertical stroke and a horizontal base that extends to the left.

© 2024 Micron Technology, Inc. All rights reserved. Information, products, and/or specifications are subject to change without notice. All information is provided on an "AS IS" basis without warranties of any kind. Statements regarding products, including statements regarding product features, availability, functionality, or compatibility, are provided for informational purposes only and do not modify the warranty, if any, applicable to any product. Drawings may not be to scale. Micron, the Micron logo, and other Micron trademarks are the property of Micron Technology, Inc. All other trademarks are the property of their respective owners.