Linux Plumbers Conference

Vienna, Austria | September 18-20, 2024

Famfs[1] and CXL Shared Memory: Progress, Challenges and Usability

John Groves - Micron

[1] Famfs stands for Fabric-Attached Memory File System

Contents

- System-RAM vs. DAX mode
- John’s Dynamic Capacity Device (DCD) Overview
- Famfs: Core insight, overview and status update
- Cache coherency and memory sharing

Using CXL Memory: System-RAM vs. DAX

System-RAM

- Memory is onlined and appears as a NUMA node with no local CPUs
- Cgroups / numactl policies applicable
- Autonuma and `migrate_pages()` work
- Hetergeneous interleaving is possible, including ratio / weight based
- System-RAM can’t be shared by separate systems
- BUT: memory / connectivity failures affect system RAS

DAX Mode

- Memory is not directly accessed by the kernel – only by apps that use the memory
- Apps can access DAX memory; (few already know how, but qemu is one)
- Shared memory via DAX works if apps know how (but very few apps know how)
- Shared memory via famfs over DAX provides scale-out sharing for apps that can share files (which many apps can do)
- AND: memory/connectivity failures only affect the RAS of apps that are using the memory or files – not the Linux kernel

John’s DCD Overview

- A Dynamic Capacity Device (DCD) is a memory device with allocation and access-control built in

- No actual memory is provided until it is allocated

- Tagged allocations are “file like” (but not file-like enough)

- When memory is allocated, it should surface as a DAX “virtual device” (also known as “tagged capacity”)
  - Sharable if the allocation request specified a sharable DCD region (Regions also control writable vs. read-only, and HW vs SW cache coherency)
  - Tags (which are UUIDs) are the namespace to find DCD memory allocations – to agree on “which memory is which”
  - Tagged Capacity DAX devices must be findable by Tag…

John’s DCD Overview (cont)

- Tags are essential to find and identify memory that was allocated for a specific purpose, or which contains specific content

- If memory is sharable, it must remain as DAX rather than System-RAM
  - System-RAM gets zeroed…

- It’s possible to program hardware interleaving for tagged DAX devices
  - …but they all must have identical extent lists in DPA space (which is a complicated ask)

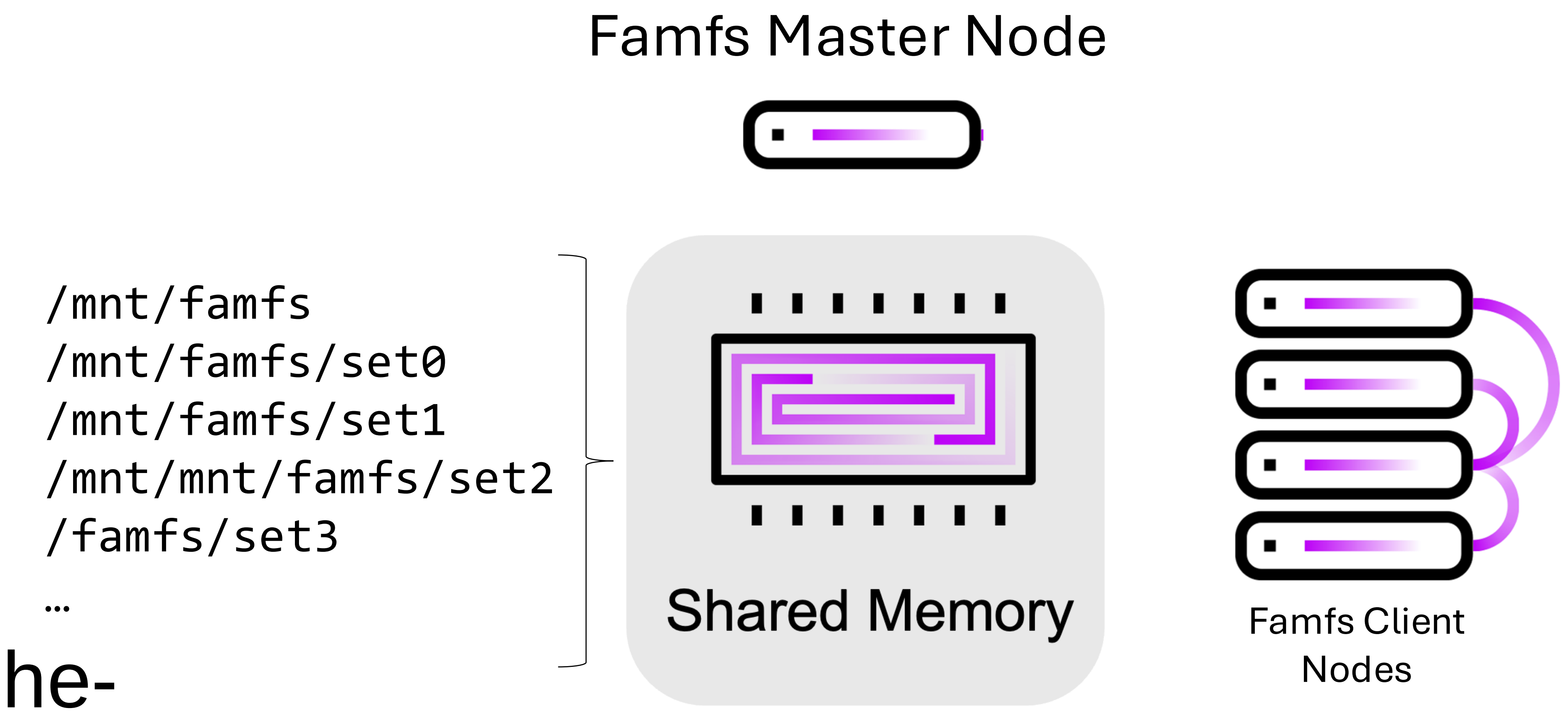
- Famfs can interleave files across [tagged] DAX devices, with no constraints on DPA (or HPA or any) address range particulars

Famfs: the Core Insight

- Prior proposals to enable of shared memory might be paraphrased as "It's a new paradigm, requiring new abstractions!"
  - See HP's "The Machine"
- But creating new abstractions tends to require software to be adapted or re-written
  - (a huge barrier to adoption)
- But the core plumbing already existed in Linux to provide a file system interface to shared memory
  - No fundamental new abstractions required
  - Many apps and work flows can adapt to famfs without the "new paradigm" re-write – because they already work with data in files!

Famfs Organizes Shared Memory as a File System

- The hard problems are:
  - Tolerating clients with a stale view of metadata
  - Providing efficient vma fault resolution
  - Reclaiming space
- Metadata is managed from user space
- Files are strictly pre-allocated (by the Master)
- Space is not reclaimed
- A memory-mapped file provides byte-addressable cache-line-level access to its backing memory
  - (conventional file systems must load an entire page into memory and then load the cache)
- Files default to read-only on Clients, but writable access is supported
- Famfs manages cache coherency for its own metadata, but not for apps

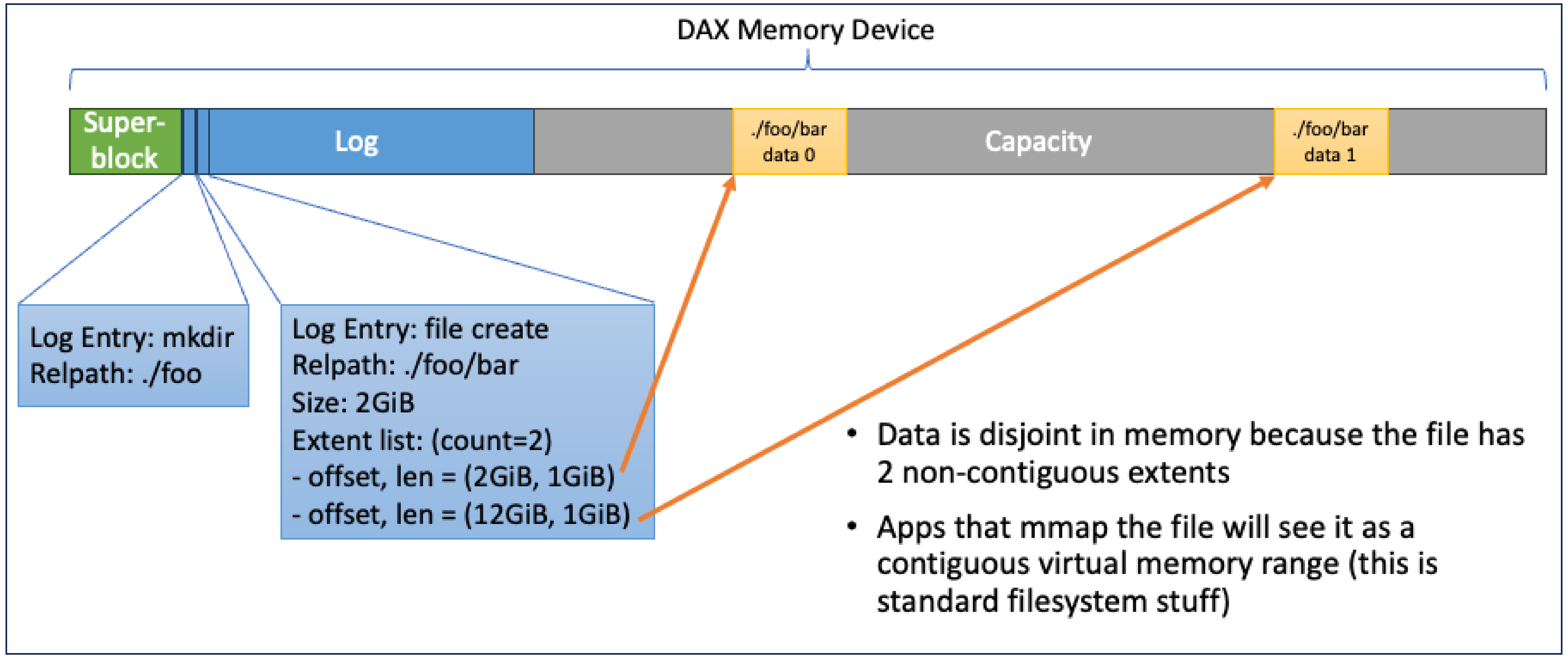


```
/mnt/famfs
/mnt/famfs/set0
/mnt/famfs/set1
/mnt/mnt/famfs/set2
/famfs/set3
…
```

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

Famfs Architecture

- Append-only metadata log written by Master and ”played” by Clients
- 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)

Famfs Status Update

- Introduced at LPC 2023
- Famfs on github
- V1 RFC in Feb 2024
- V2 RFC in April 2024
- LSFMM (May 2024) consensus was that a FUSE port should be attempted (lwn)
  - This looks feasible but it's a lot of work
  - Much of the famfs kernel functionality will land in fuse
  - Patches later this year…barring setbacks
- Interleaved files: August 2024

Famfs: Interesting Use Cases

- Larger working sets than can currently fit in server memory

- Avoid sharding and shuffling when data fits in FAM / CXL
  - Not all use cases can be readily sharded
  - Shuffling (redistributing data to where compute cycles are available) can have order n^2 (for n nodes)

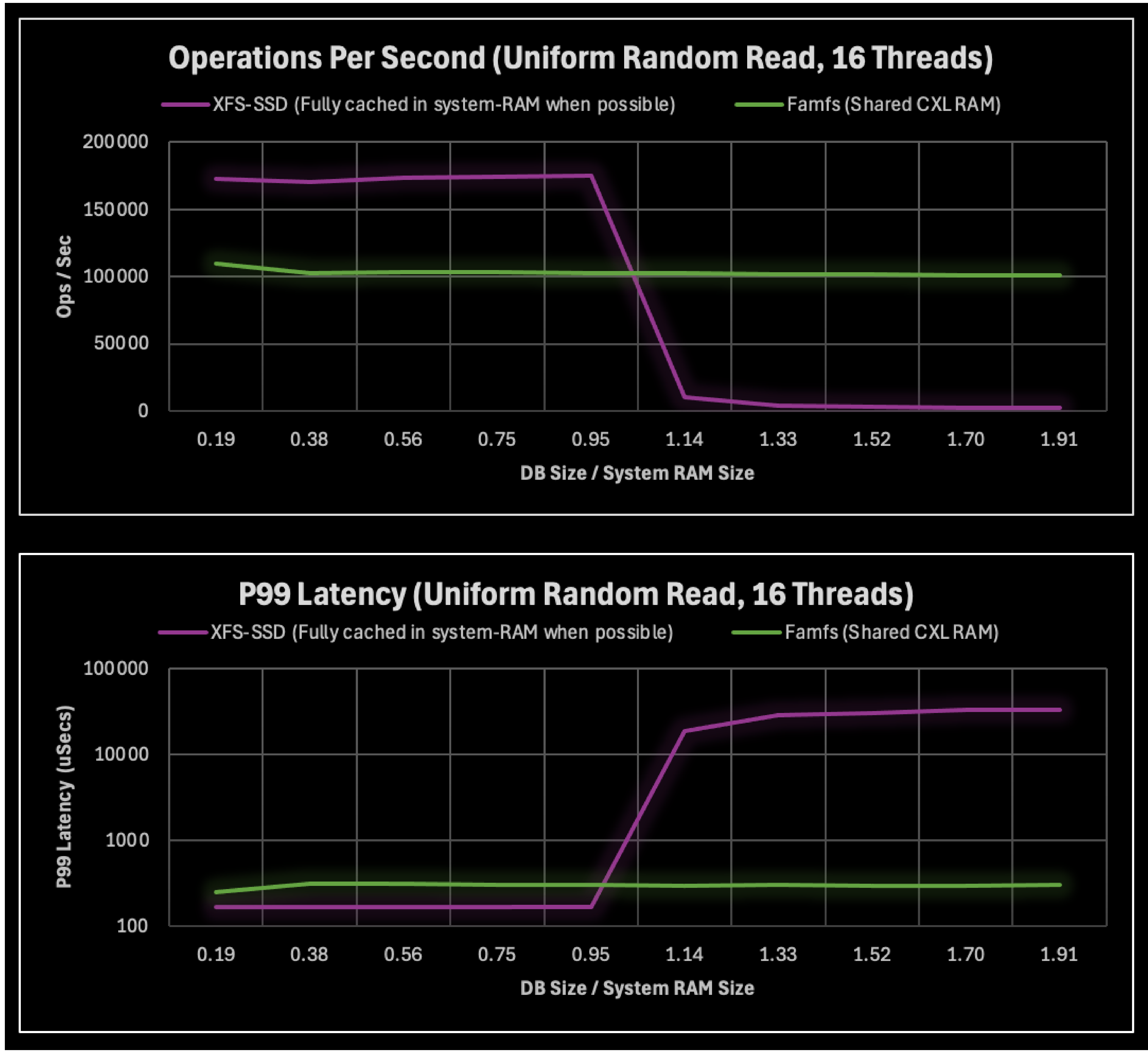
- Sharing data is effectively de-duplicates in memory

- FAM does not create any new cache coherency problems – it just exacerbates some old ones

- Agree on location of data for computational offload
  - (Both the Tag namespace and famfs files help with this)

Famfs: Bigger Data In Memory

- RocksDB read-only benchmark
  - Note FAM is slower but bigger than system-ram
  - Performance will improve
- Conventional file system (database fully cached when it fits) vs. Famfs
- FAM can be scaled independently of server memory capacity
  - Typical limit is 12 DDR slots x 256GiB = 3TiB
- X Axis normalized to system-ram size
- This data was shown at FMS '24



Cache Coherency

- There are **not** a lot of apps that are candidates for concurrent writer shared FAM applications
  - But if there are, managing cache coherency for disaggregated memory will be (significantly) more expensive than it already is for local memory
  - Shared FAM doesn't create new problems – it just exacerbates some old problems

- There are a lot of apps and use cases that share data sets read-only
  - cache coherency is almost free

- There are a lot of apps that share data in a "pipeline" fashion:
  one writer at a time, handing off to the next stage when finished
  - Coherency is pretty easy

- Shared memory and famfs are well suited to these apps and use cases (and is compatible with read/write shared if the app has its act together)

- If shared data is read-only, hardware cache coherency is actively detrimental

Famfs: Some Viable Apps

APACHE ARROW
jupyter
NumPy
pandas
DASK
PyTorch
DuckDB
APACHE Spark™
RAY
Velox

Linux Plumbers Conference

Vienna, Austria | September 18-20, 2024