Linux
Plumbers
Conference | Richmond, VA | Nov. 13-15, 2023

CXL 3 Shared Memory

What Will Be Required?

John Groves, Micron
jgroves@micron.com

Linux Plumbers Conference | Richmond, VA | Nov. 13-15, 2023

FAMFS: Goals / Plans

- Raise awareness of the coming CXL 3 fabric-attached memory capabilities
- Point out the need for a scale-out FSDAX file system
- Start a dialog about the architectural trade-offs and mvp feature set

CXL Memory: 2 Top-Level Use Cases

Pooling (add memory to 1 host)

- Memory is usually “onlined” as System-RAM
  - (But can be accessed as DAX device, e.g. for VM-backing memory)
  - Online memory is zeroed
- Lots of interesting features in development (interleaving, tiering, migration, etc.)

➢ Not compatible with multiple-host sharing

Linux Plumbers Conference | Richmond, VA | Nov. 13-15, 2023

CXL Memory: 2 Top-Level Use Cases

Pooling (add memory to 1 host)

- Memory is usually "onlined" as System-RAM
  - (But can be accessed as DAX device, e.g. for VM-backing memory)
  - Online memory is zeroed
- Lots of interesting features in development (interleaving, tiering, migration, etc.)

➢ Not compatible with multiple-host sharing

Sharing (multiple hosts)

- Memory is DCD "Tagged Capacity"
- Mapped via /dev/dax device
- Found by something like: /sys/devices/dax/<tag>

Is this sufficient to make shared FAM broadly usable?

NO - Tagged capacity creates the foundation, but it will require a File system abstraction layer

What Does CXL 3 Shared FAM* Look Like?

- Devices are “Dynamic Capacity Devices” (DCDs)
  - DCDs handle allocation and access control
  - A DCD provides no memory until it is allocated
- Allocated capacity is “Tagged Capacity”
  - Allocated via the Initiate Dynamic Capacity Add command to a DCD
  - Tag (think UUID) is necessary for sharable allocations
  - Init DC Add is re-issued by Tag to add shared access for each host (LDFAM) or host group (GFAM)

* FAM = Fabric Attached memory

Some Observations about Sharable Memory

- It does not make sense to online sharable memory as System-RAM
  - System-RAM gets *zeroed*
- The most accessible use cases will be adaptations of apps that can access or share data in [memory-mapped] files
- Sharable FAM is pmem-like
  - Hosts may map FAM to get access to pre-existing data
- Cache Coherency
  - CXL supports both hardware and software-based cache coherency (but the laws of physics still apply)
  - Managing cache coherency is easy if data is not being written

About FSDAX

- The VFS layer already supports files that map directly to non-sparse special-purpose memory (SPM)
- SPM is not System-RAM; it is mapped via DAX
  - But it can be owned and/or managed by an FSDAX file system
- Inodes with the S_DAX flag do not map via the page cache, they map directly to DAX memory
  - Resolving vma faults is handled in conjunction with the DAX driver

Can We Enable FAM for a Lot of Use Cases?

- The AI and data science tool chains make heavy use of data sets in files, including shared data sets
- "Zero-copy formats" are a thing:
  - Data in files is mmapped for consumption
  - Data is vectorized in memory for efficient computation
- If these tools mmap an S_DAX file, memory is accessed directly (not via the system-ram page cache)
- Read-only data is common, making cache coherency simple in those cases
- Are datasets in raw tagged capacity (raw DAX device) good enough? NO
  (Tools know how to mmap *files*, not DAX)
- We need a sharable / scale-out FSDAX file system to expose sharable data sets

Data Science / AI apps & tools

- Jupyter
- Pandas
- Numpy
- Apache Arrow
- Apache Spark
- Ray
- Pytorch
- Dask
- Velox
- Etc…

Dev DAX vs. File Constraints

	Dev dax	S_DAX file
Size	• Size must be a multiple of applicable page size • Size must also be a multiple of the DCD extent size (which can be 1GiB or greater) • Size cannot necessarily match the intrinsic size of a data set • Size detected via sysfs (stat does not work) ➢ Apps must be dax-aware	• Can be any size • Can exactly match data set size • Size detected via stat() • Apps only need to be file-aware (and they already are)
Subdividing	• Devdax instance analogous to one file • Subdividing a devdax requires daxctl or app awareness	• Tagged capacity instance (devdax) can be subdivided into an arbitrary number of files
Usability	• Apps must adapt to devdax quirks	• Any app that can mmap data from files can access data in FAMFS
Orchestrators (e.g. K8S)	• Don’t handle DAX devices	• Do handle file and file system access

Why Not Existing FSDAX File Systems?

- Current FSDAX file systems don't handle metadata or space allocation in a sharable way
  - Metadata is write-back; only a single host can mount an FSDAX file system*, even if more than one host can see the memory
  - Allocate-on-write does not scale out (and don't get me started about delayed allocation)

➢Cannot scale out on a cluster

* Multiple read-only mounts work, but are hacky and of limited use

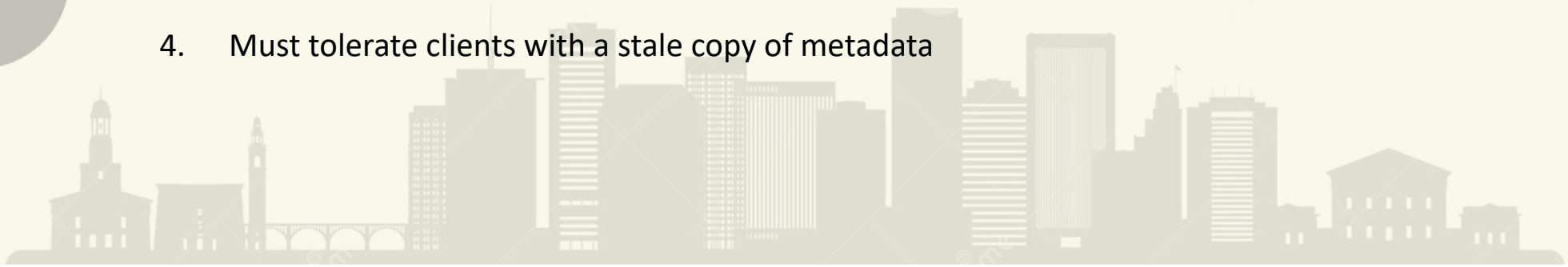
FAMFS Requirements

1. Must create an FSDAX file system abstraction atop Tagged Capacity (shared dax devices)

2. Files must efficiently handle VMA faults
   - We're exposing memory – must run at memory speeds
   - Fast resolution of TLB/page-table faults to dax device offsets is essential

3. Must distribute metadata in a sharable way

4. Must tolerate clients with a stale copy of metadata



FAMFS: What is it Today?

We have a prototype…

- RAMFS clone
- Ioctl call to convert a ramfs file to a DAX file and provide DAX-based extent list
- A log format, and a log distribution mechanism
  - FAMFS metadata is distributed as an append-only log
  - Log entries: mkfile, mkdir, …
  - Log Play instantiates local files that map to the appropriate shared dax devices
  - Log Play is handled from user space on each client
  - In-memory metadata is not written back to the log (client-side metadata is ephemeral)
- Files are allocated by the master before they are committed to the log
  - Never allocate-on-write (or –after-write)
- Data *may* be writable by clients, but logged metadata is read-only

FAMFS: Master vs. Clients

Operation	Master	Client
Mkfs	Init superblock and log	n/a
Mount	Play log (if any)	Play log (instantiates files)
File Create	• Allocate FAM capacity • Create local file backed by space • Initialize data • Commit log entry	Ask master to create / allocate file
File Usage	Same as clients	• Apps can mmap / read / *write files according to permissions • Apps cannot truncate or append

FAMFS: One More Problem

- FSDAX file systems work with a /dev/pmem block device (not a /dev/dax character device)
  - Current FSDAX file systems sort-of pretend the memory is a block device for the purpose of metadata I/O
  - But volatile CXL memory appears as a /dev/dax character device
- Currently /dev/dax character devices do not support the iomap…() machinery that is used to resolve vma faults from file offsets to dax device offsets for address resolution
- We have not solved this problem yet (we're currently forcing our dax devices to think they're pmem), but a FAMFS patch series should also solve this problem…

Linux Plumbers Conference | Richmond, VA | Nov. 13-15, 2023

FAMFS: Goals / Plans

- Raise awareness of the need
- Start a dialog about the architectural trade-offs and mvp feature set
- We hope to start posting RFC patch series' in the near future
  (But no firm commitment yet)

Backup

FAMFS: Can we Use FUSE?

- Currently: No
- Handling local consumption of distributed metadata would work
- Resolving TLB/page-table faults via upcall to user space is a non-starter
  - Fuse would need to cache DAX extent lists for files (and support revoking them, etc.)
  - Without this, we can't meet requirement #2

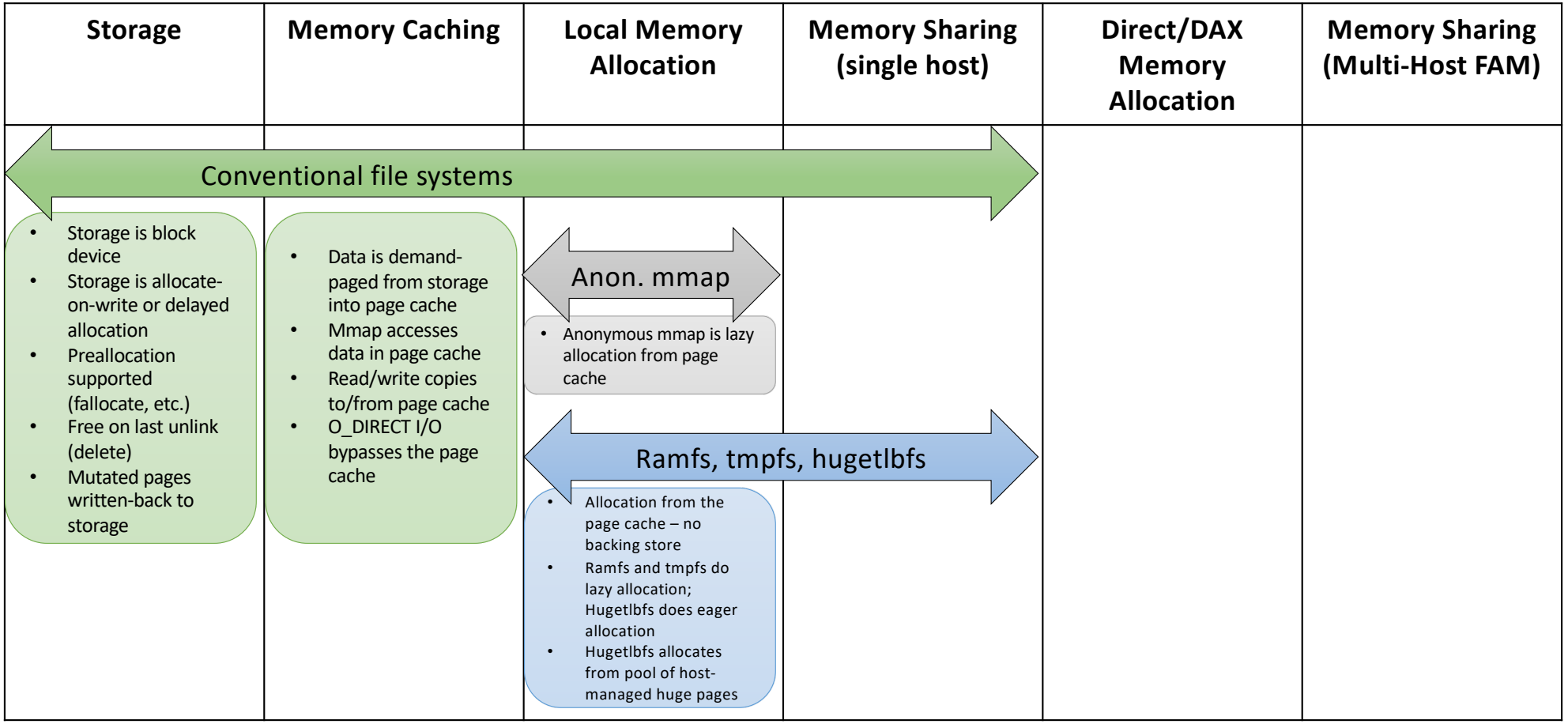
File System / VFS Functionality

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				

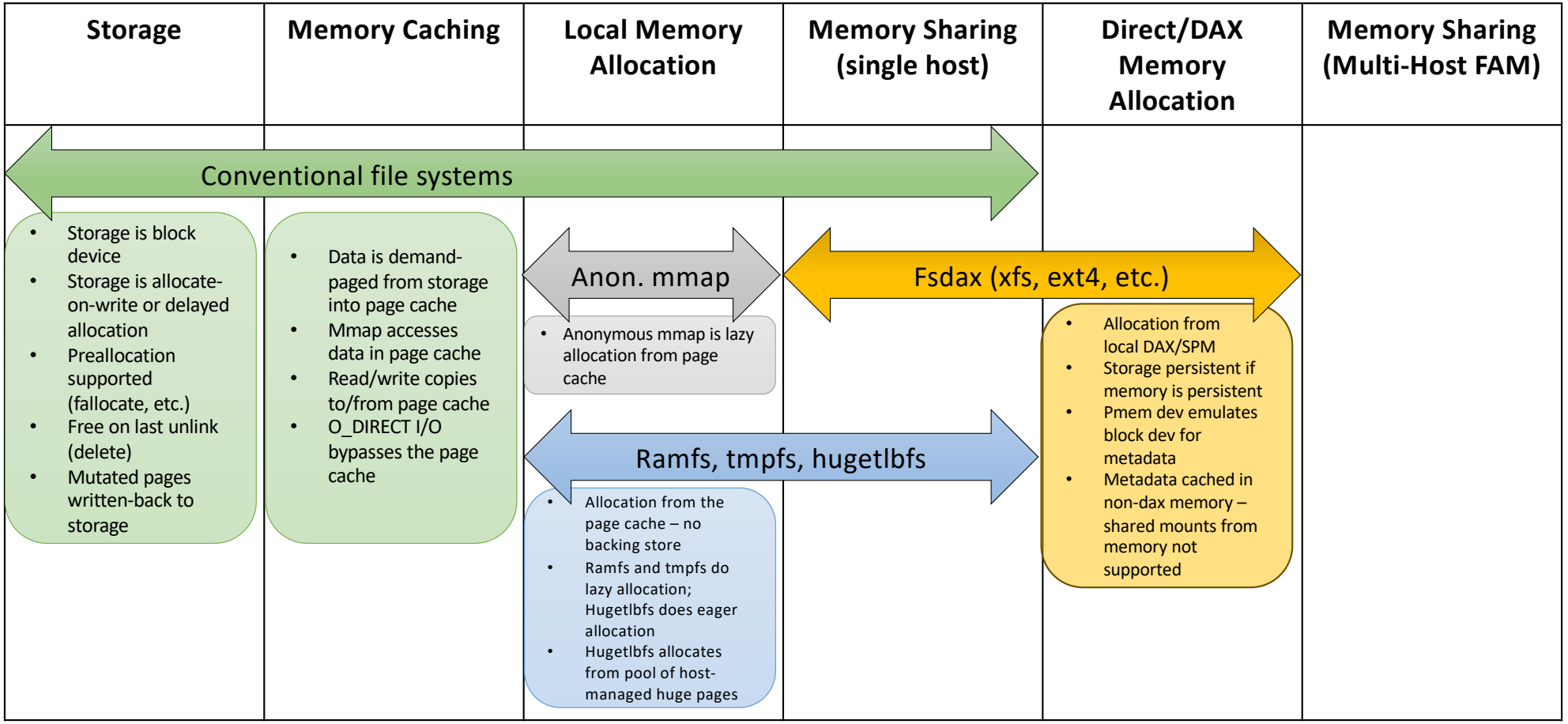
File System / VFS Functionality

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