Non-volatile Memory

An emerging technology

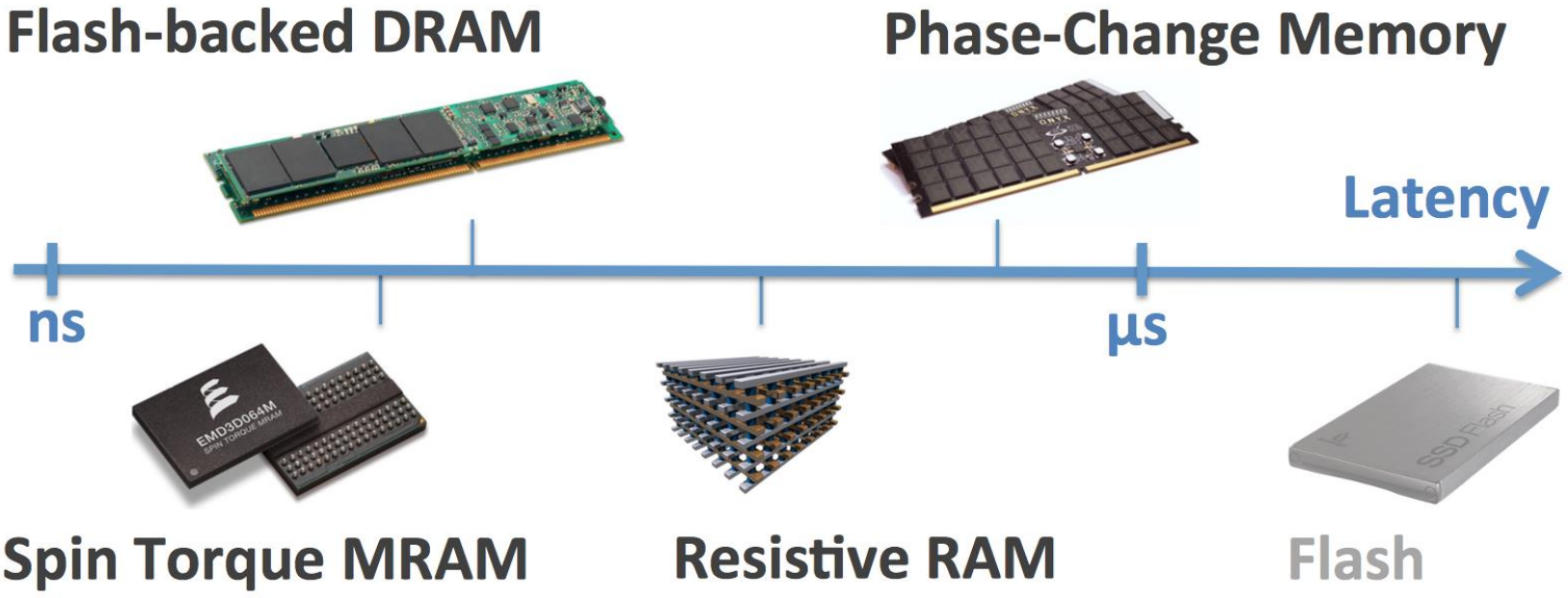
Yubin Xia

Some slides from http://research.cs.wisc.edu/sonar/tutorial/

What is non-volatile memory?

NVM: WHAT & WHY

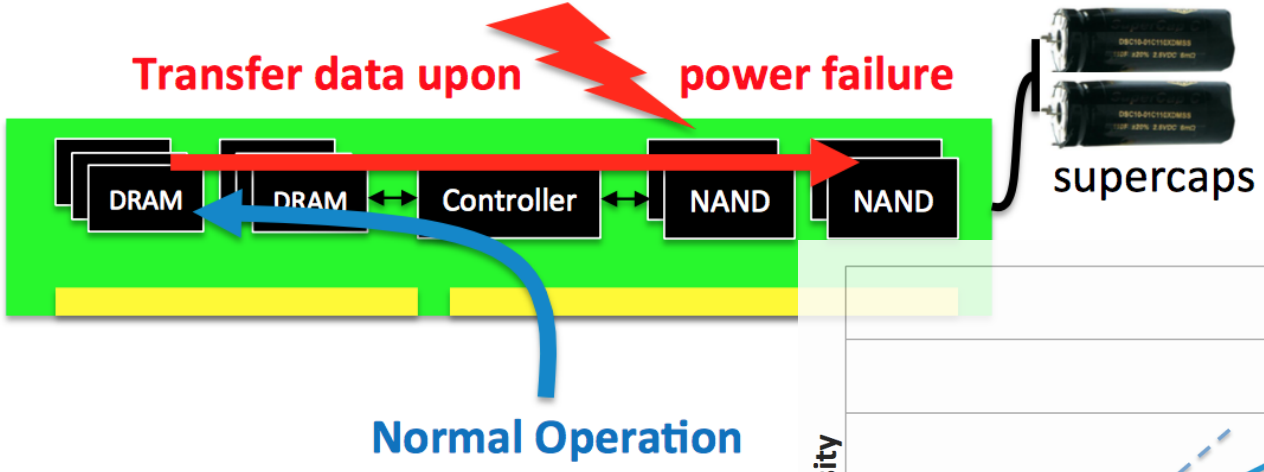
NVM Technologies



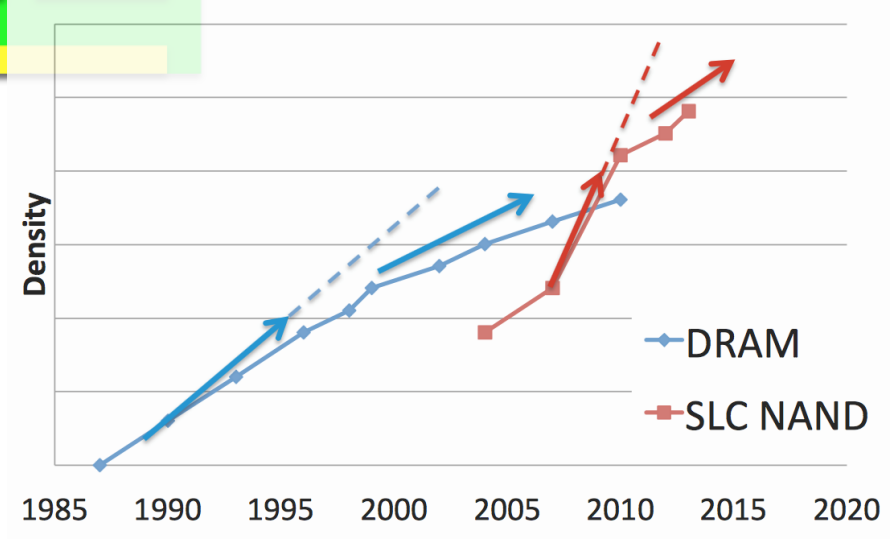
- Persistent
- Short access time
- Byte addressable

NVDIMM

- Store data in DRAM backed by NAND Flash

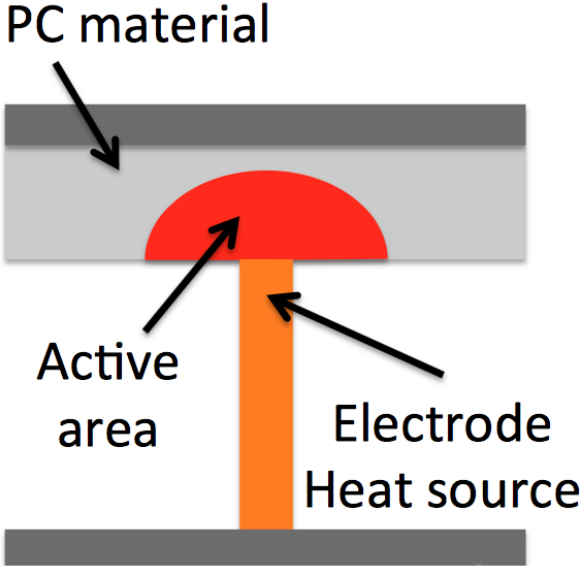


- Capacity is hard to scale



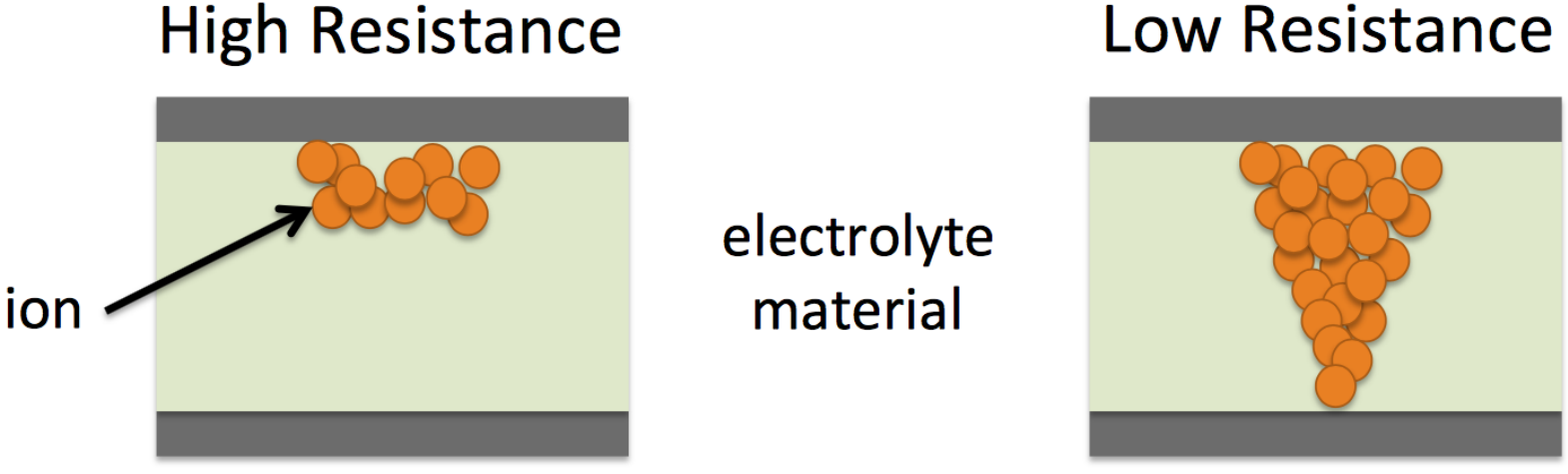
Phase-Change Memory (PCM)

- Store data within phase-change material
  - Amorphous phase: high resistivity (0)
  - Crystalline phase: low resistivity (1)
- Set phase via current pulse
  - Fast cooling → Amorphous
  - Slow cooling → Crystalline



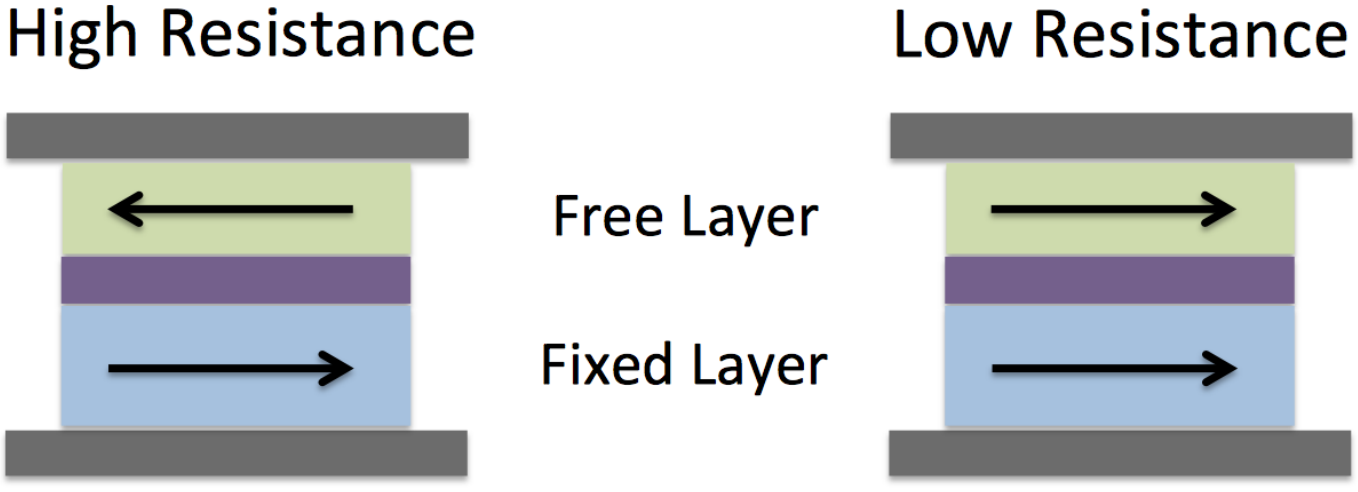
Resistive RAM (RRAM)

- Store data by dissolving ions within electrolyte memristive material (e.g., TiOx)



Spin-Transfer Torque RAM (STT-RAM)

- Store data within magnetic-tunnel junction
  - Anti-parallel orientation: high resistance (0)
  - Parallel orientation: low resistance (1)



3D-XPoint

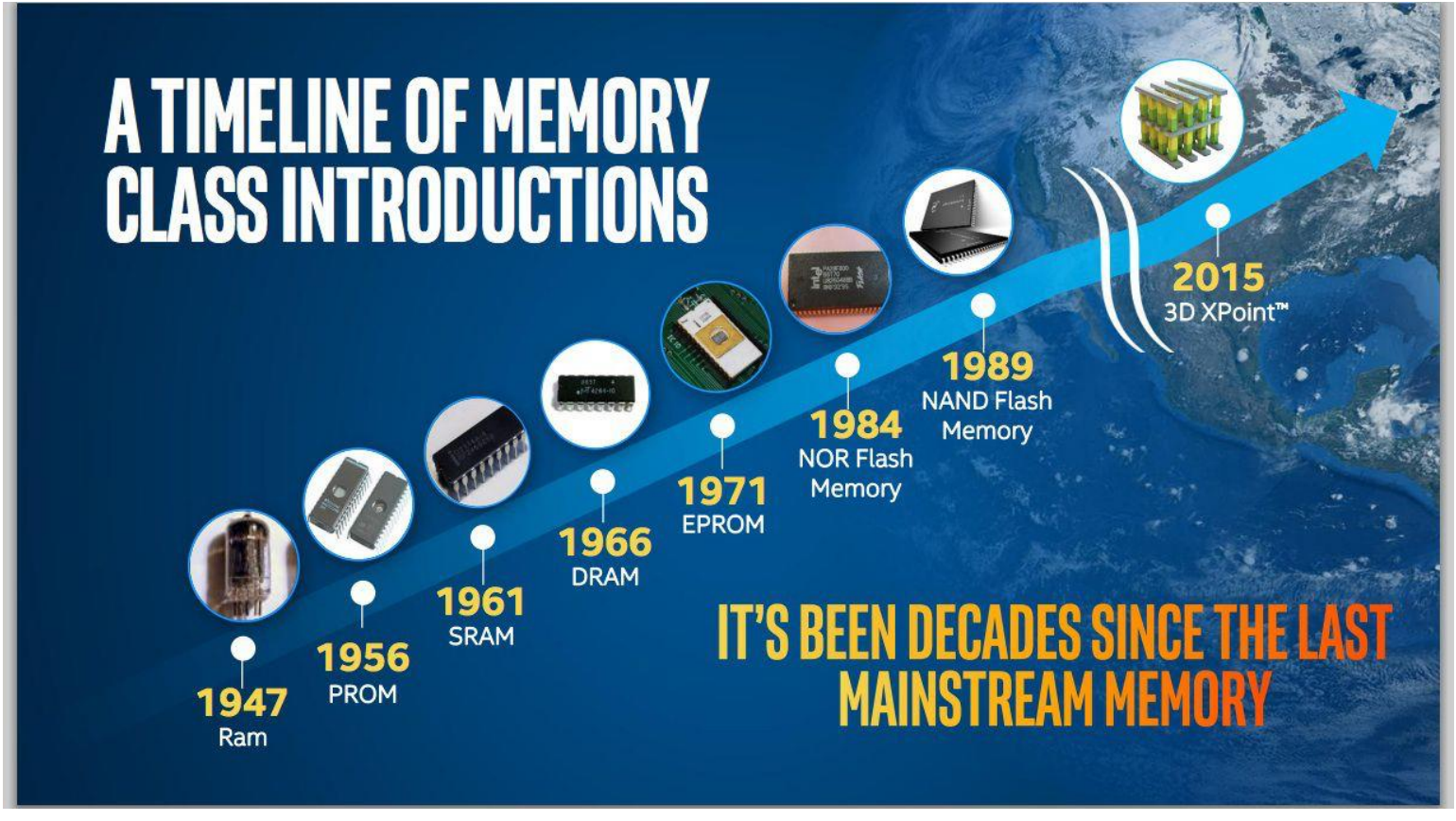


Figure from https://fortune.com/2015/07/28/intel-micron-memory/

Intel Optane DC Persistent Memory

- Available last year

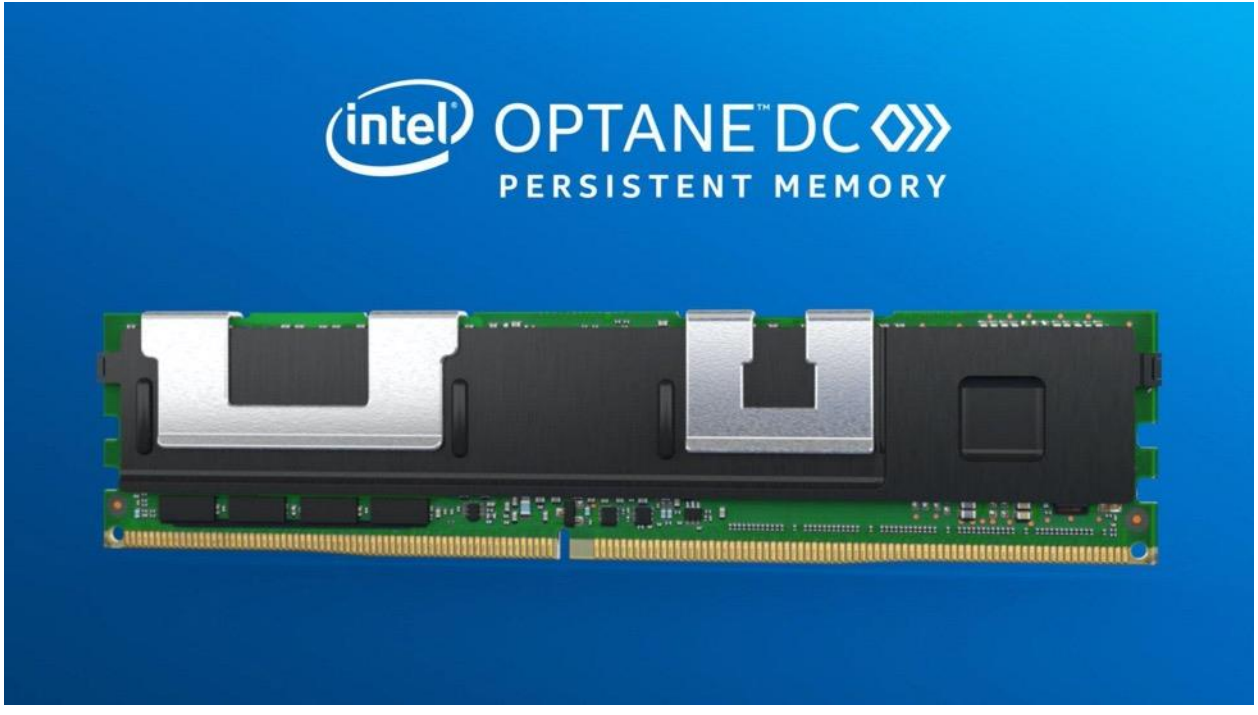
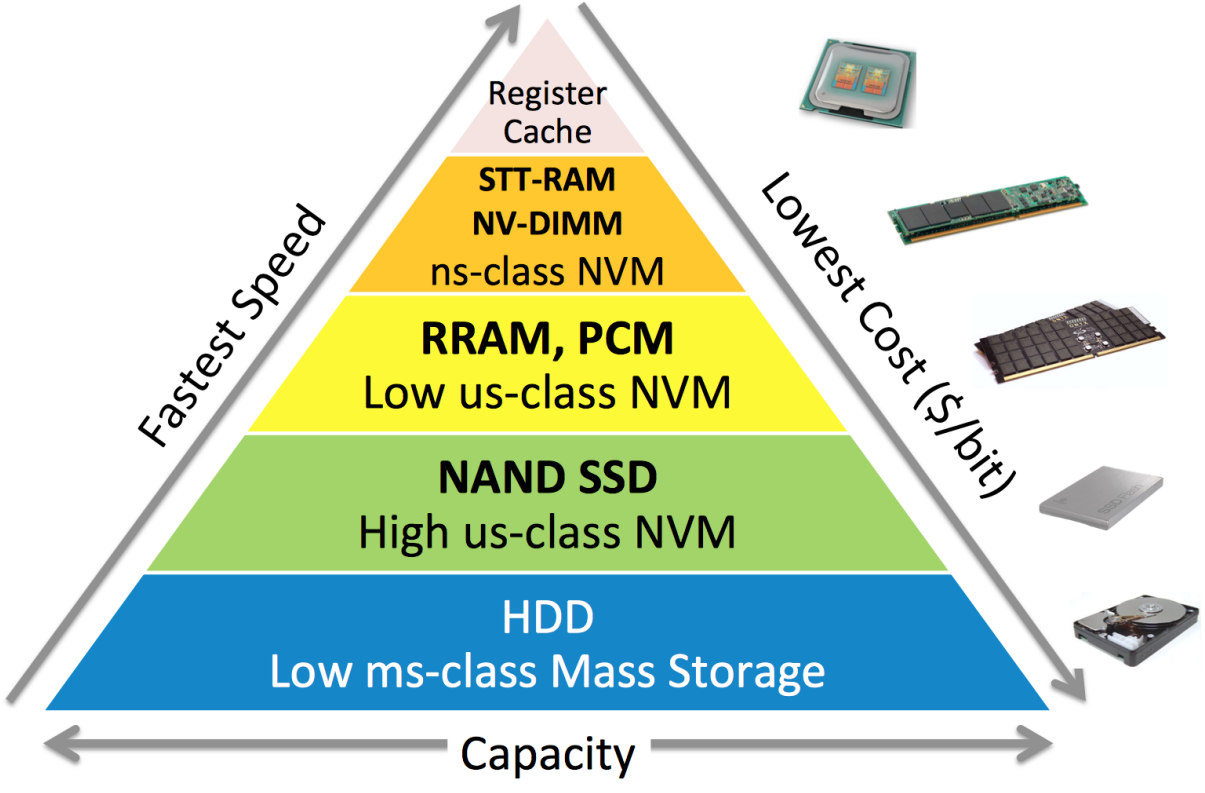


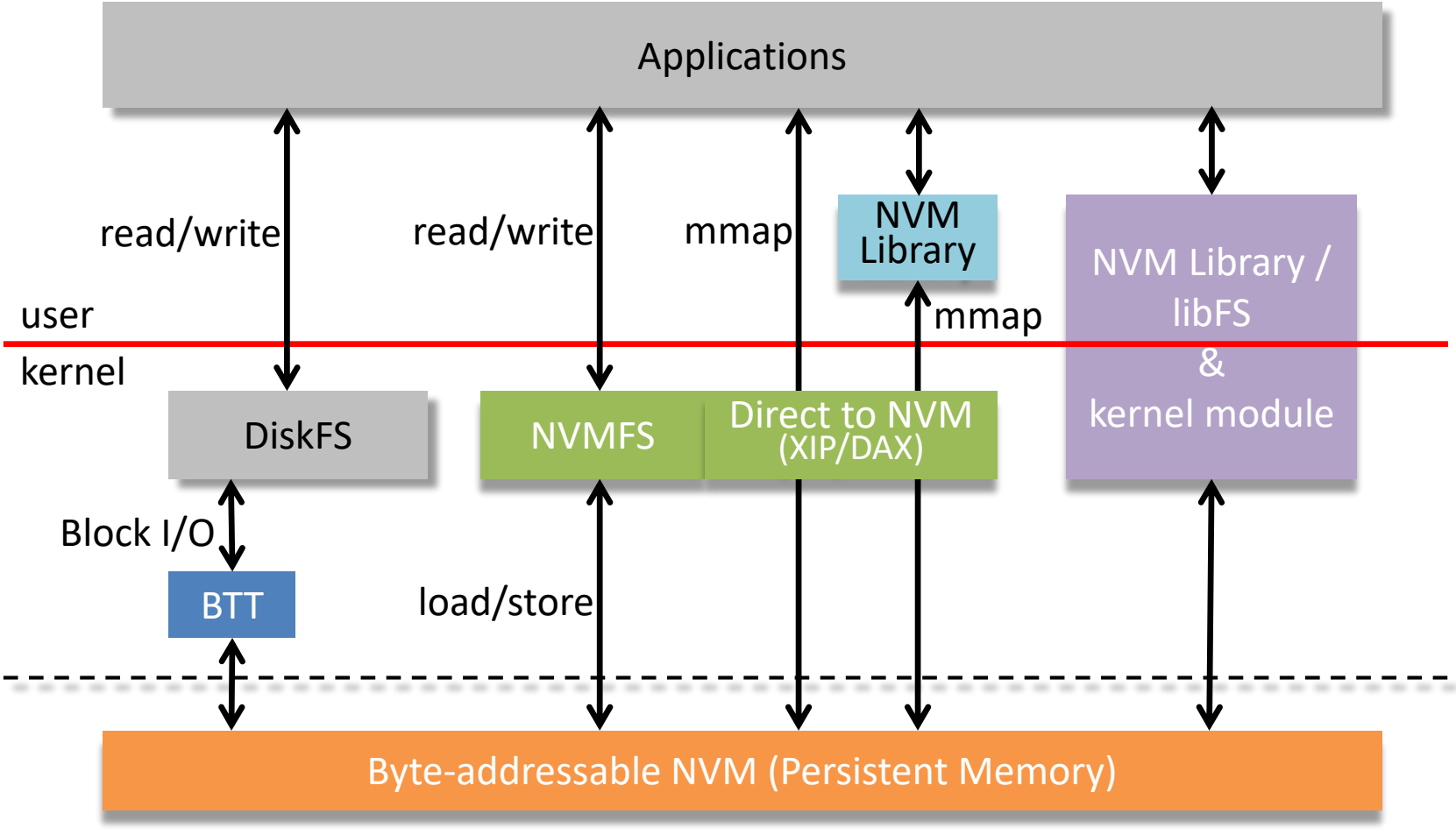
Figure from Intel, https://www.intel.com/content/www/us/en/architecture-and-technology/optane-dc-persistent-memory.html

Summary


Register
Cache
STT-RAM
NV-DIMM
ns-class NVM
RRAM, PCM
Low us-class NVM
NAND SSD
High us-class NVM
HDD
Low ms-class Mass Storage
Fastest Speed
Lowest Cost ($/bit)
Capacity

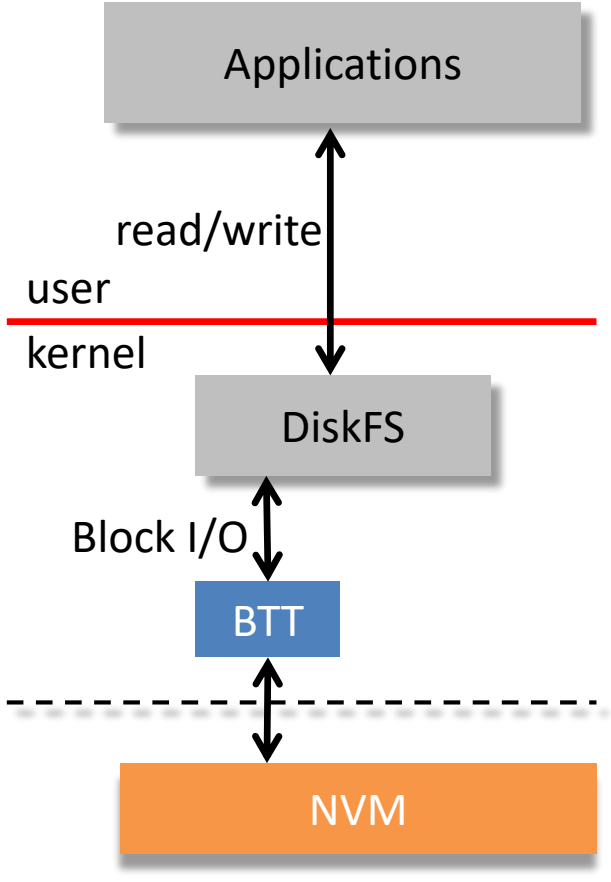
WHERE TO USE

Ecosystem Overview



DiskFS on NVM

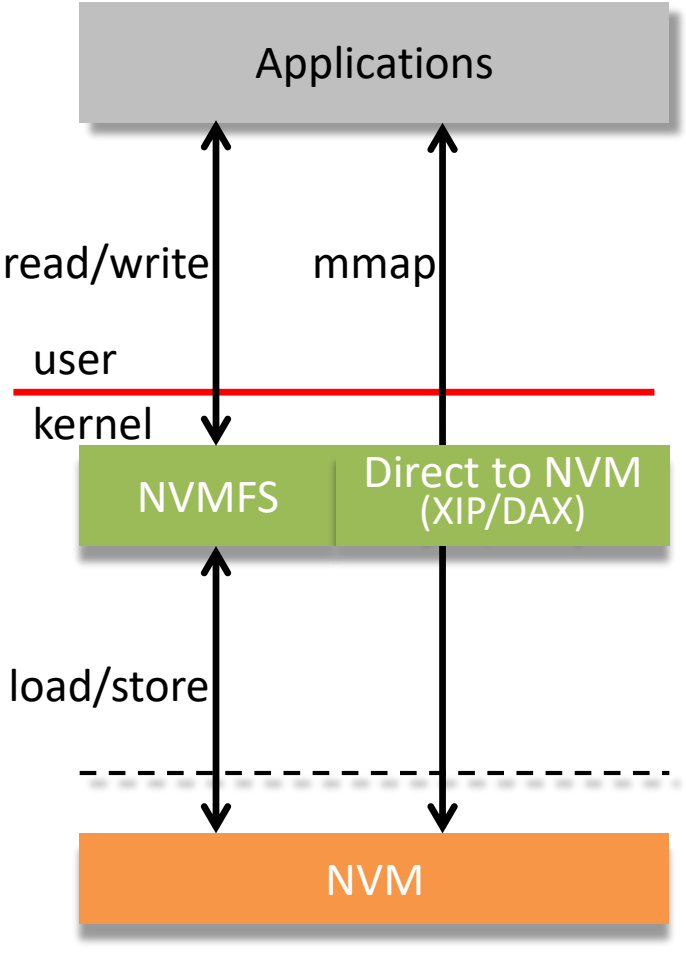
- Traditional DiskFS
  - Ext2/3/4, XFS, BtrFS, F2FS,…
- Block Translation Table (BTT)
  - Block interface over NVM
  - Atomicity



Q: Why DiskFS on NVM?

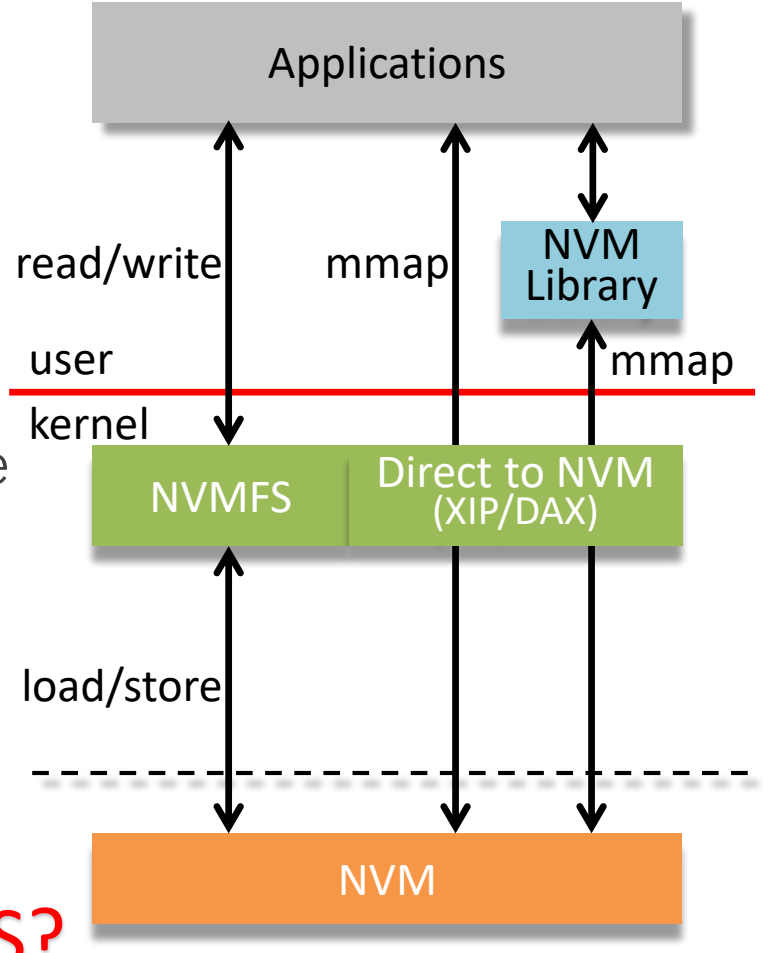
NVMFS

- Designed for NVM
  - BPFS, PMFS, NOVA, SoupFS,…
- Direct Access (DAX)
  - Direct load/store on NVM
  - Exposed to app using mmap



NVM Library over FS

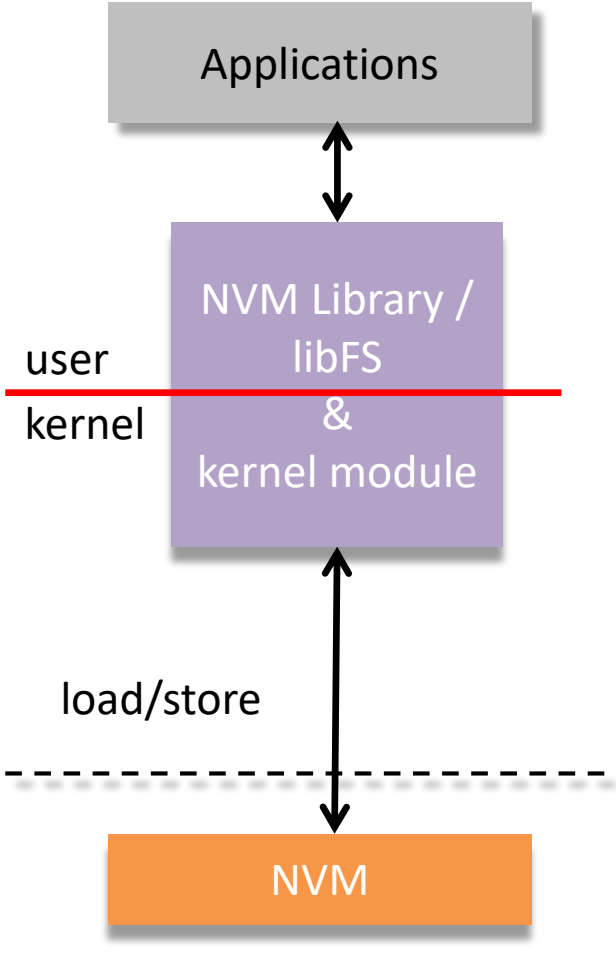
- Relies on FS
  - PMDK,…
  - FS organizes the space
  - Library/App controls how to use
- Interfaces
  - Objects, transactions, memory pool,…



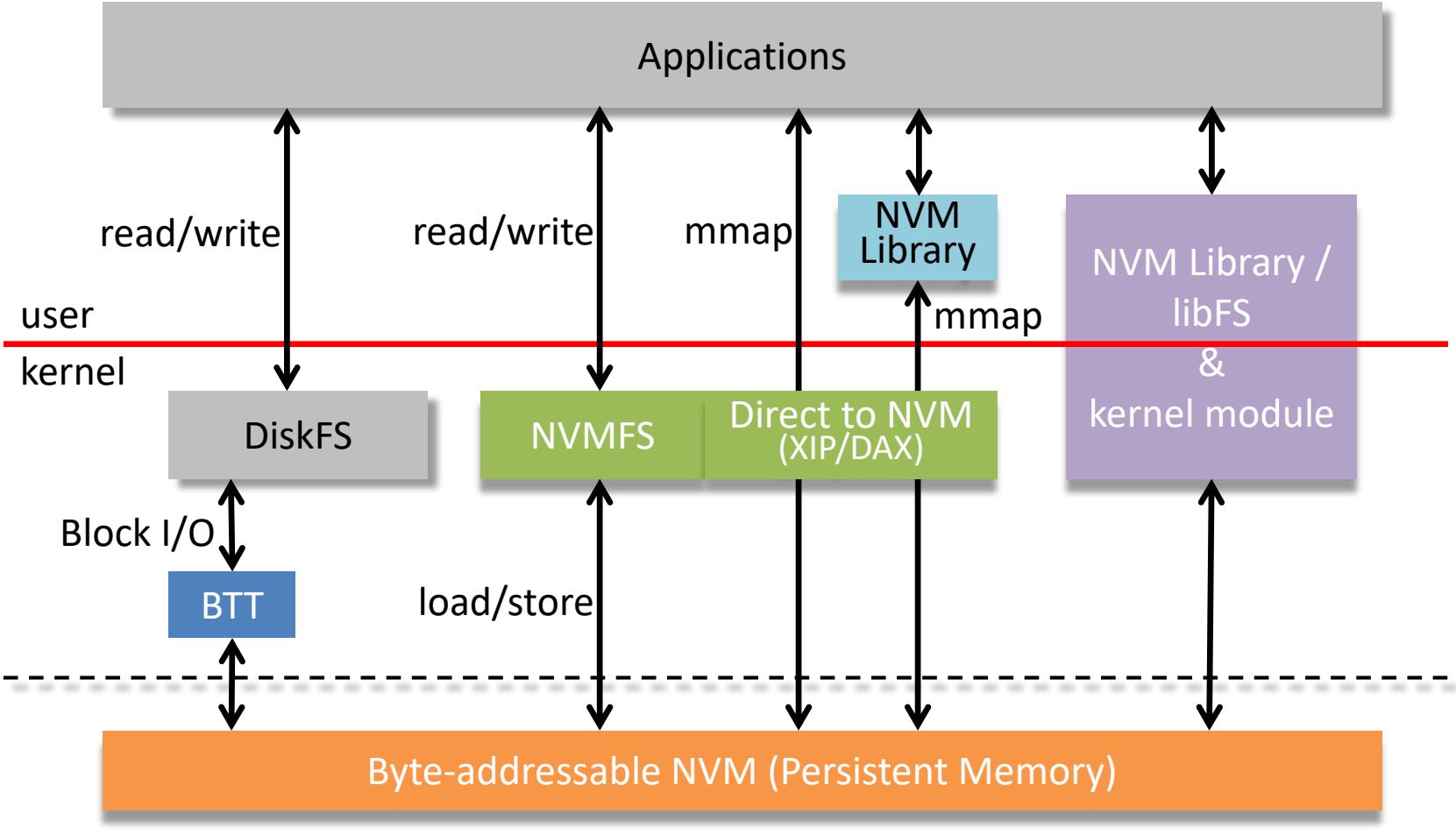
Q: Why are they built upon FS?

NVM Library / libFS with Kernel Module

- User-kernel space co-design
  - Mnemosyne, Aerie,…
  - Better performance
- Interfaces
  - Objects, transactions, memory pool,…
  - FS



Ecosystem Overview

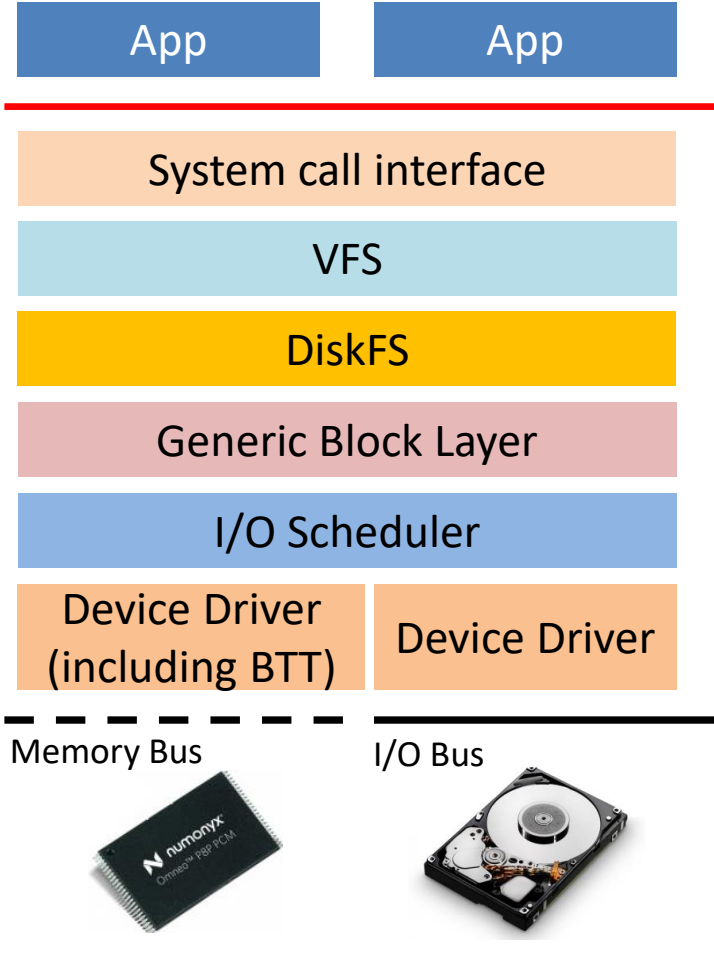


Why?

- Why FS?
  - FS interfaces matter
  - Backward compatibility
- Why NVMFS?

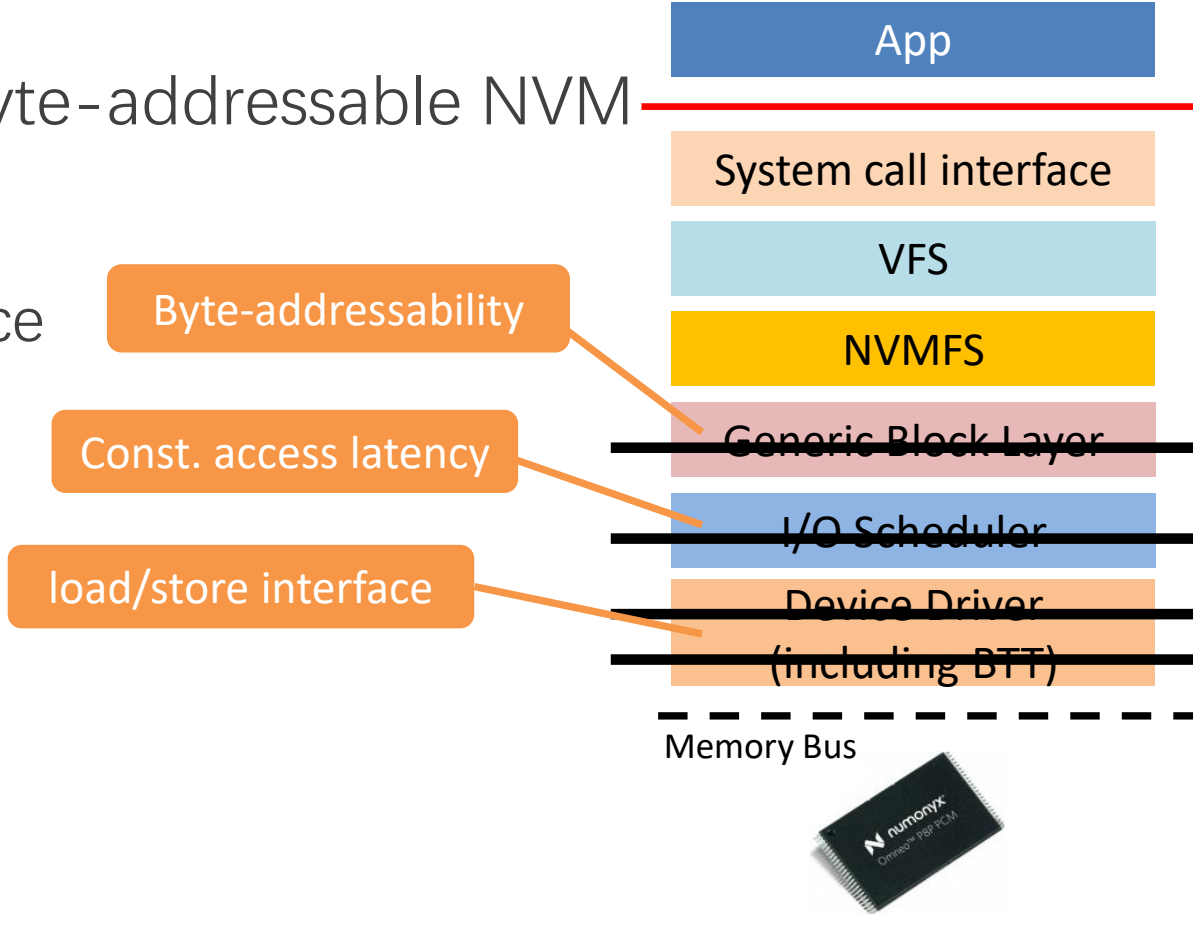
DiskFS

- FS designed for block-based disks
  - Needs BTT for correctness
  - Too many layers leading to inefficiency



NVMFS

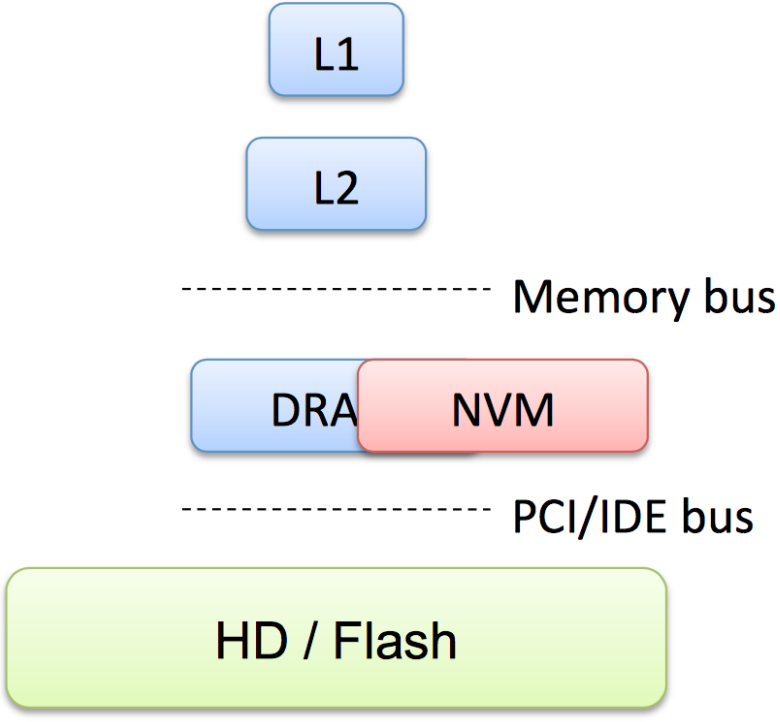
- FS designed for byte-addressable NVM
  - Simplified stack
  - High performance



Nothing comes free in life...

CHALLENGES

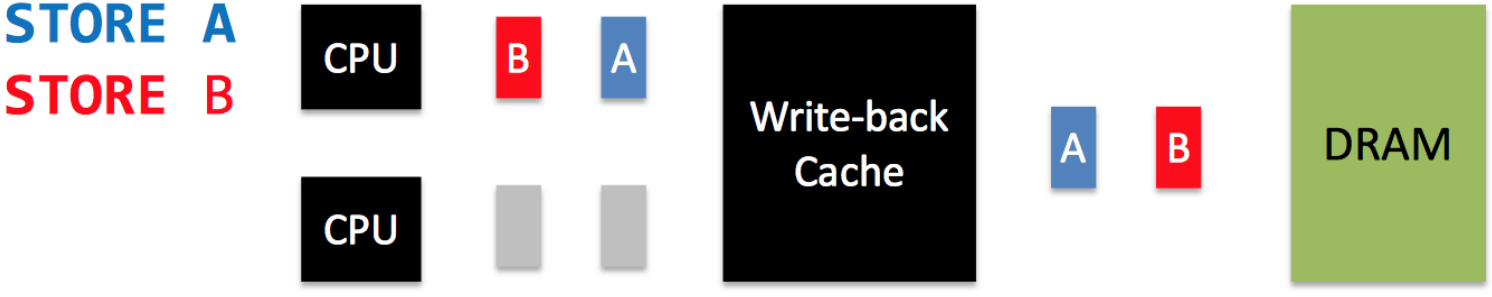
Cache and Write-back



	DISK	NVM
Data cached in	DRAM	CPU cache
Write-back controlled by	Software	Hardware

Volatile Memory Ordering

- Write-back caching
  - Improves performance
  - Reorders writes to DRAM



- Reordering to DRAM does not break correctness

Non-volatile Memory Ordering

- Recovery depends on write ordering

```
STORE data[0] = 0xFOOD
STORE data[1] = 0xBEEF
STORE valid = 1
```

Non-volatile Memory Ordering

- Recovery depends on write ordering

```
STORE data[0] = 0xFOOD
STORE data[1] = 0xBEEF
STORE valid = 1
```

Reordering breaks recovery

Recovery incorrectly considers garbage as valid data

Simple Solutions

- Disable caching
- Write-through caching
- Flush entire cache at commit

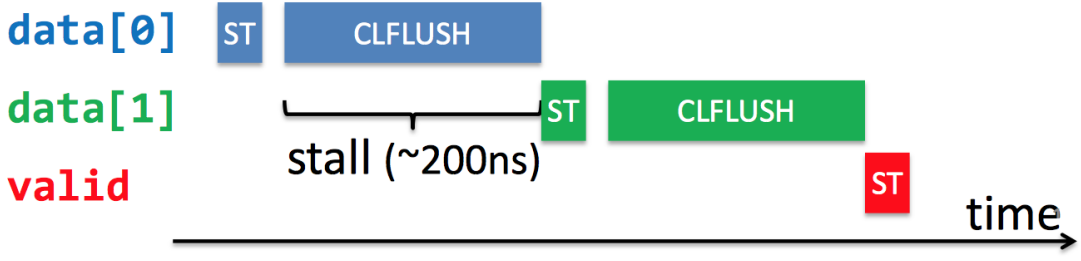
Significantly affect performance!

Ordering with Existing Hardware

- Order writes by flushing cachelines via CLFLUSH

```
STORE data[0] = 0xFOOD
STORE data[1] = 0xBEEF
CLFLUSH data[0]
CLFLUSH data[1]
STORE valid = 1
```

- But CLFLUSH
  - Stalls the CPU pipeline and serializes execution
  - Invalidates the cacheline

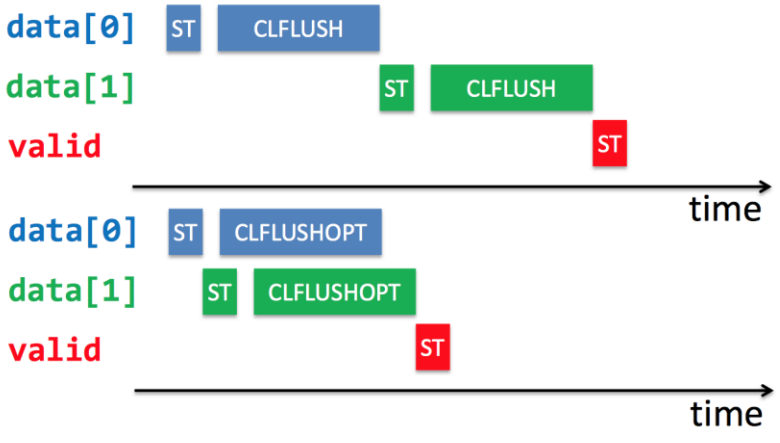


Fixing CLFLUSH: Intel x86 Extensions

- CLFLUSHOPT
  - Unordered version of CLFLUSH

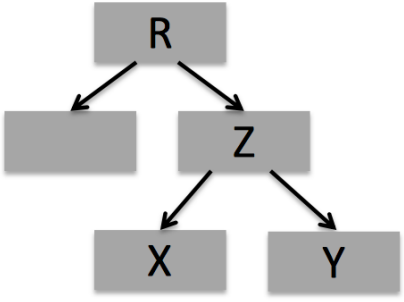
```
STORE data[0] = 0xFOOD
STORE data[1] = 0xBEEF
CLFLUSHOPT data[0]
CLFLUSHOPT data[1]
SFENCE // explicit ordering point
STORE valid = 1
```

Implicit orderings

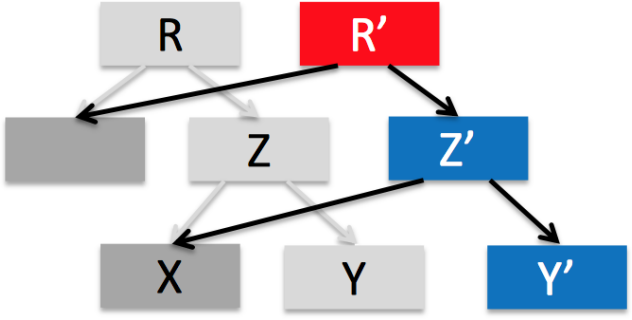


- CLWB
  - Same as CLFLUSHOPT
  - Does not invalidate cache

Example: Copy-on-Write

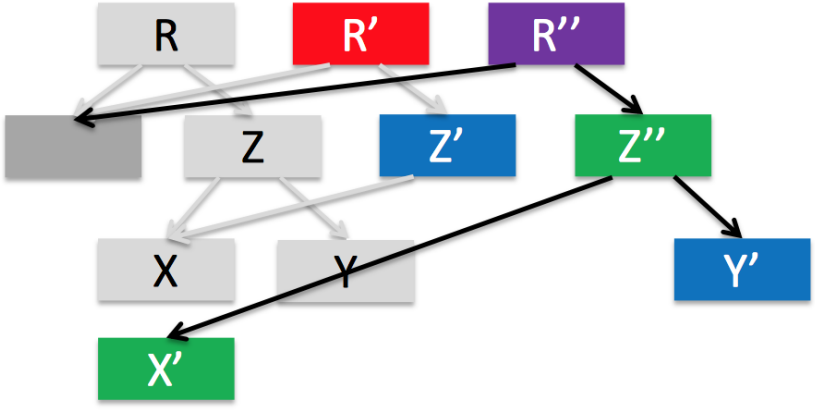


Example: Copy-on-Write




```
STORE Y'
STORE Z'
CLWB Y'
CLWB Z'
SFENCE
STORE R'
CLWB R'
```

Example: Copy-on-Write




```
STORE Y'
STORE Z'
CLWB Y'
CLWB Z'
SFENCE
STORE R'
CLWB R'
SFENCE
STORE X'
STORE Z''
CLWB X'
CLWB Z''
SFENCE
STORE R'
CLWB R'
```

Back to NVMFS

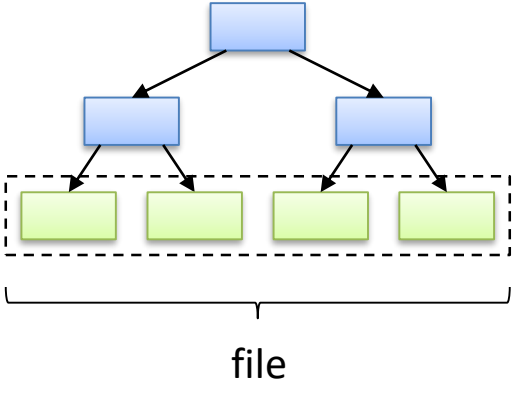
CASE STUDY: BPFS

[Condit, SOSP'09]

BPFS: A BPRAM File System

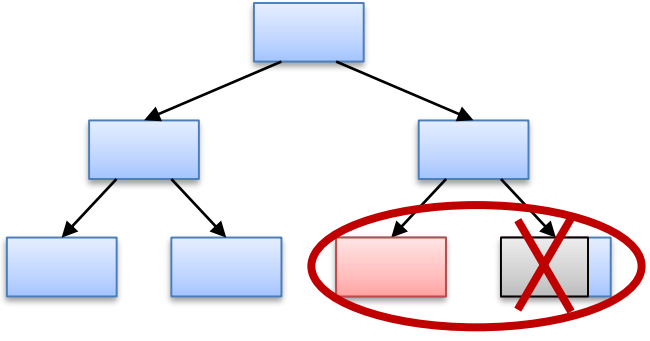
- Guarantees that all file operations execute atomically and in program order
- Despite guarantees, significant performance improvements over NTFS on the same media
- Short-circuit shadow paging often allows atomic, in-place updates

BPFS: A BPRAM File System



Enforcing FS Consistency Guarantees

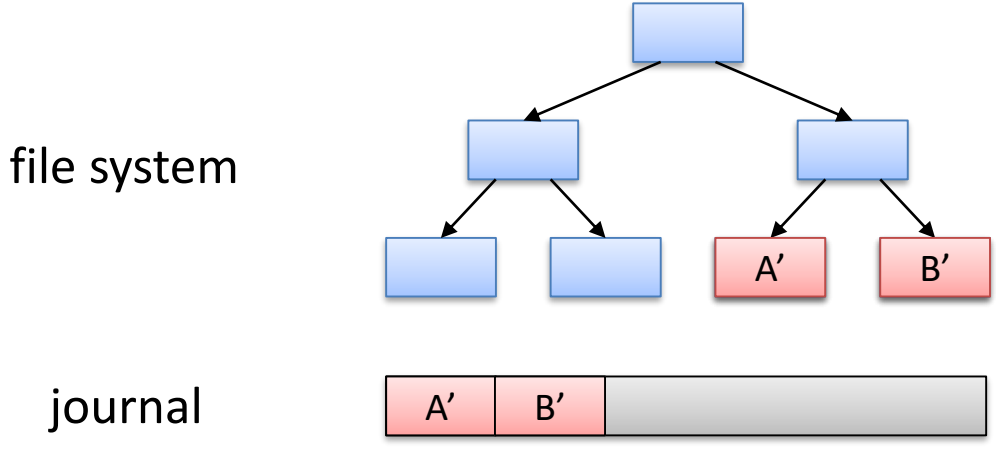
- What happens if we crash during an update?



  - Disk: Use journaling or shadow paging
  - BPRAM: Use short-circuit shadow paging

Review 1: Journaling

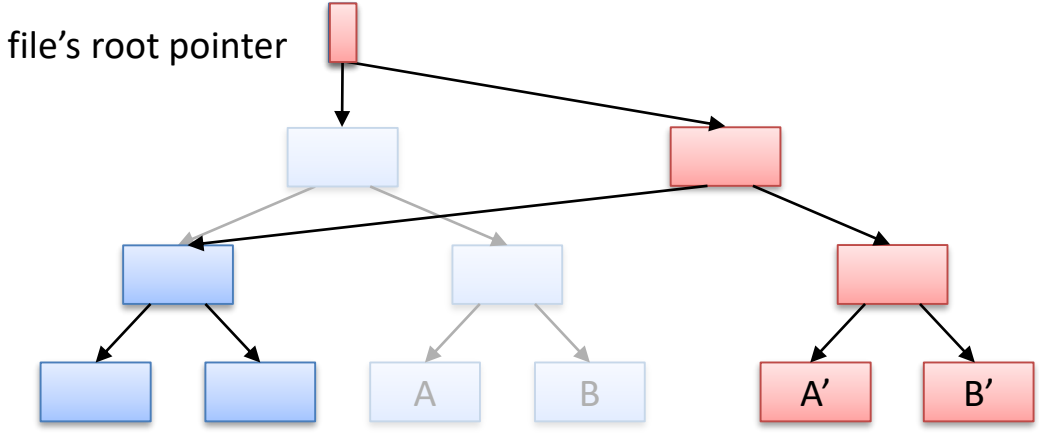
- Write to journal, then write to file system



- Reliable, but all data is written twice

Review 2: Shadow Paging

- Use copy-on-write up to root of file system



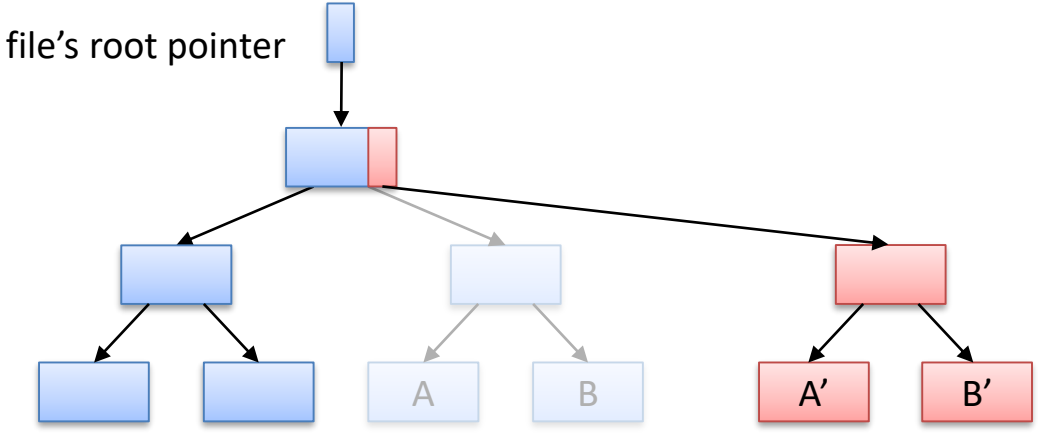
- Any change requires bubbling to the FS root
- Small writes require large copying overhead

Short-Circuit Shadow Paging

- Inspired by shadow paging
  - Optimization: In-place update when possible

Short-Circuit Shadow Paging

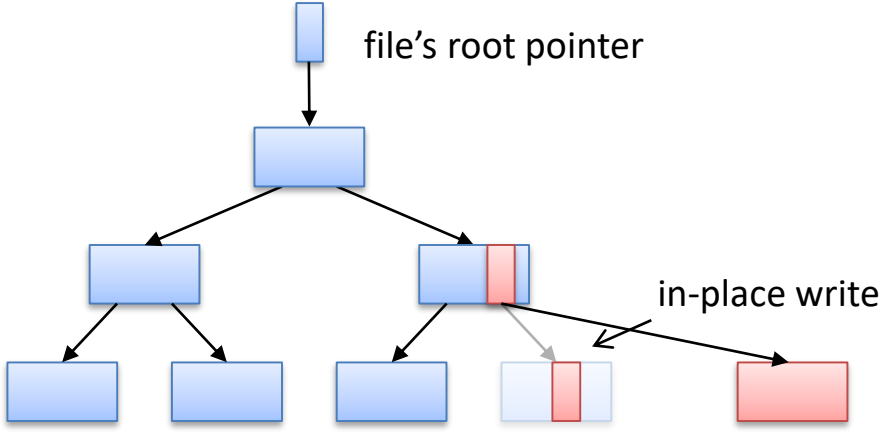
- Inspired by shadow paging
  - Optimization: In-place update when possible



- Uses byte-addressability and atomic 64b writes

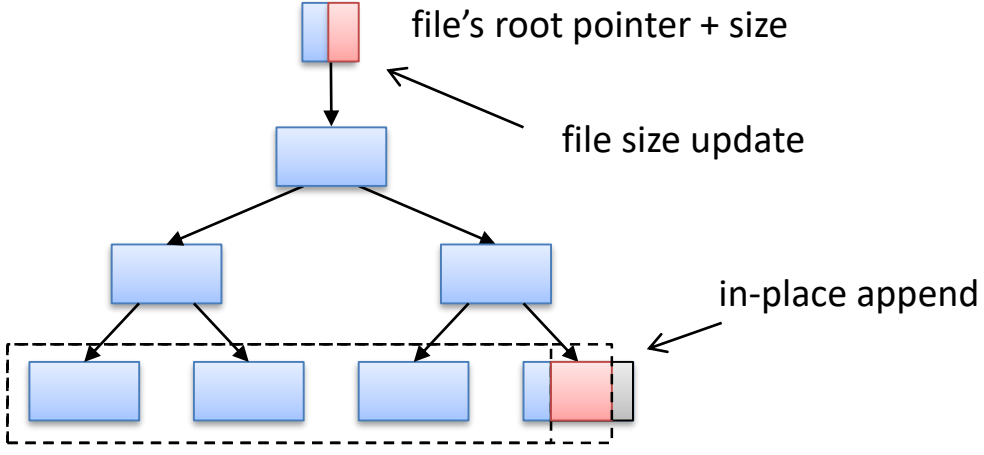
Opt. 1: In-Place Writes

- Aligned 64-bit writes are performed in place
  - Data and metadata

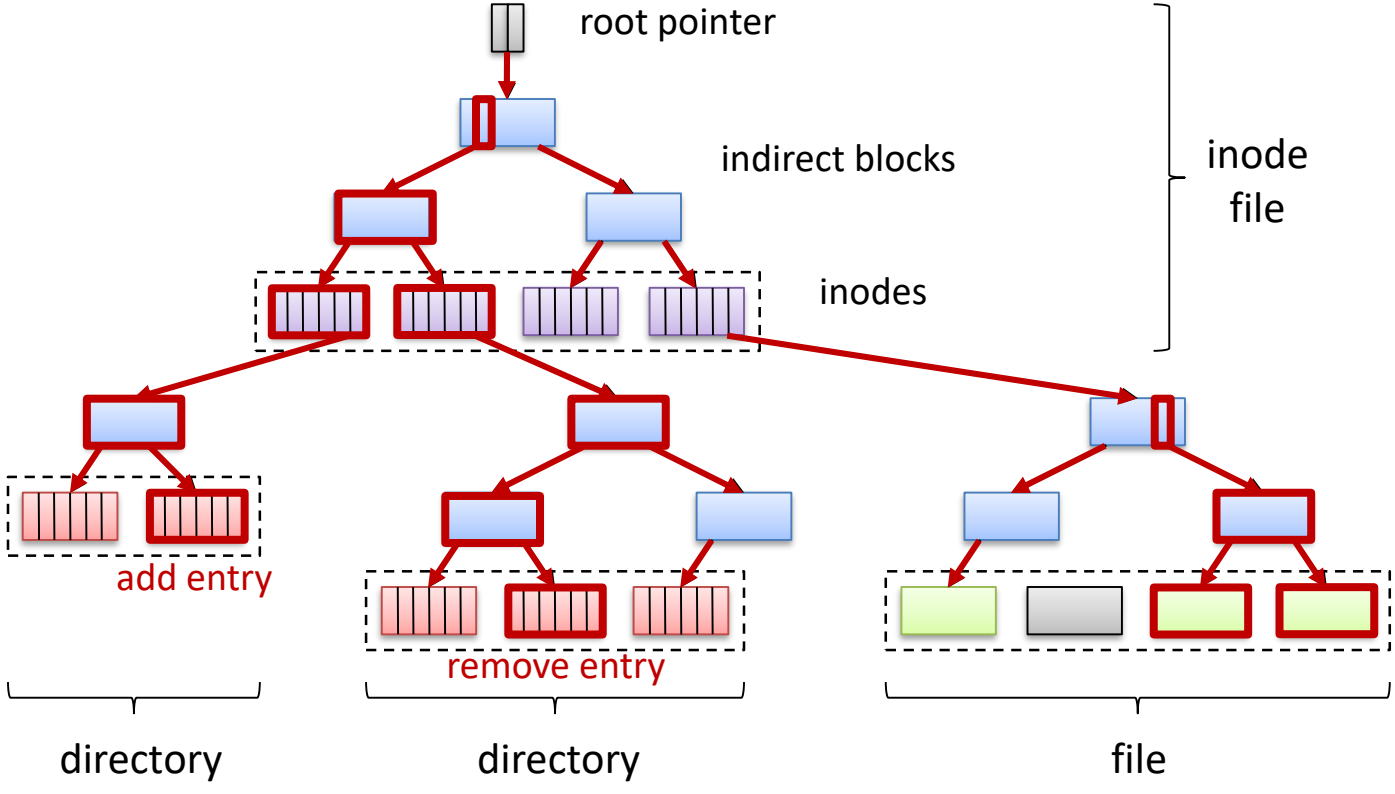


Opt. 2: Exploit Data-Metadata Invariants

- Appends committed by updating file size



BPFS Example



Cross-directory rename bubbles to common ancestor

Another

CASE STUDY: PMFS

[Kumar, EuroSys'14]

PMFS Layout

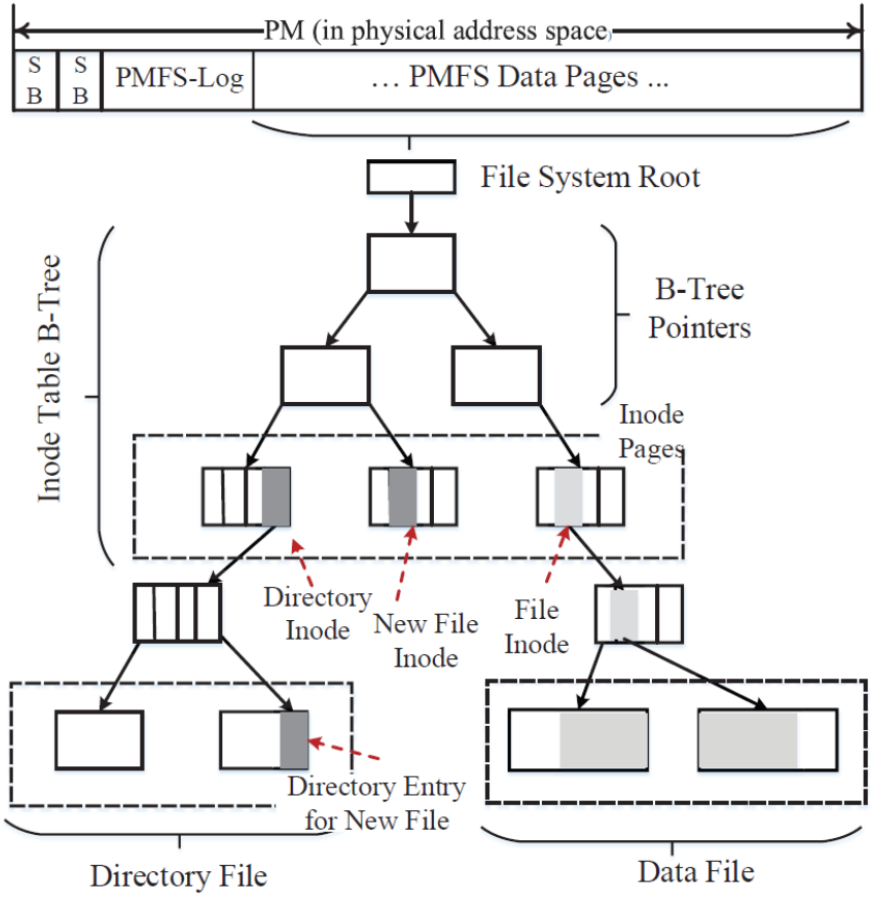


Figure 3: PMFS data layout

Consistency

- Three existing techniques:
  - Copy-on-Write (Shadow Paging)
  - Journaling
  - Log-structured updates
- One more NVM specific technique
  - Atomic in-place writes

Used for updates on Data Area

Used for updates on Metadata (e.g. inodes)

Used for updates of small portion of data

Extended Atomic In-place Writes

- **8 bytes**
  - CPU natively supports 8-byte atomic writes
  - Update inode's access time
- **16 bytes**
  - Using *cmpxchg16b* instruction
  - Update inode's size and modification time
- **64 bytes**
  - Using RTM (introduced in Haswell)
  - Update a number of inode fields like delete

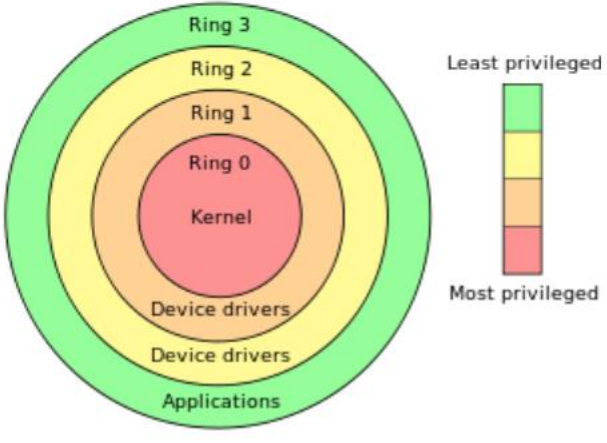
Write Protection Stray Writes

- Supervisor Mode Access Protection (SMAP)
  - Prohibit writes into user area
- *Write windows* (introduced in PMFS)
  - Mount as read-only

    When writing, CR0.WP is set to zero

	User	Kernel
User	Process Isolation	SMAP
Kernel	Privilege Levels	Write windows

Table 2: Overview of PM Write Protection



Right) http://en.wikipedia.org/wiki/Protection_ring

One more

CASE STUDY: NOVA

[Xu, FAST'16]

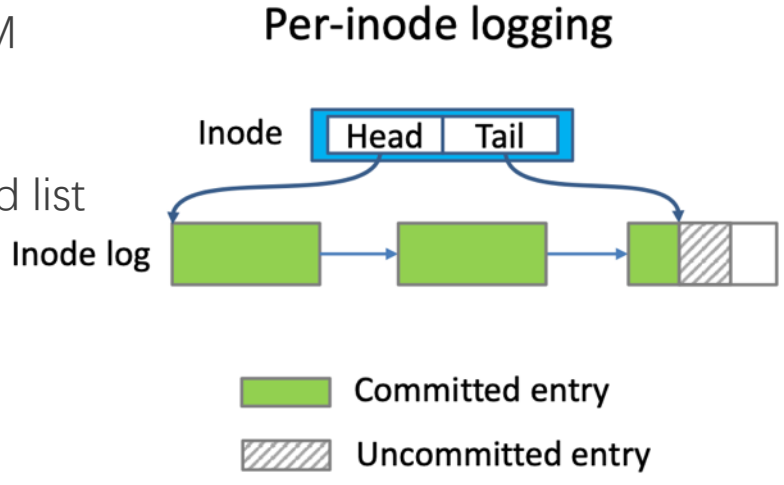
A Log-structured File System

- Log-structuring provides cheaper atomicity than journaling and shadow paging
- NVMM supports fast, highly concurrent random accesses
  - Using multiple logs does not negatively impact performance
  - Log does not need to be contiguous

- Rethink and redesign log-structuring entirely

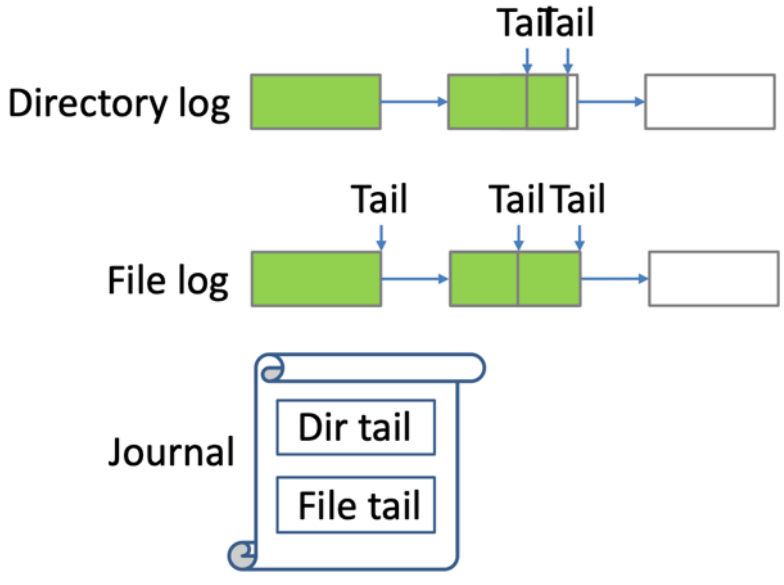
NOVA design goals

- Atomicity
  - Combine log-structuring, journaling and copy-on-write
- High performance
  - Split data structure between DRAM and NVM
- Efficient garbage collection
  - Fine-grained log cleaning with log as a linked list
  - Log only contains metadata
- Fast recovery
  - Lazy rebuild
  - Parallel scan



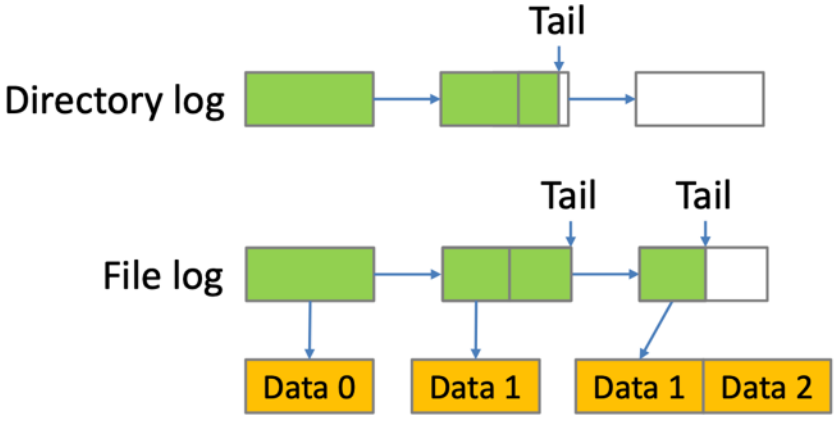
Atomicity

- Log-structuring for single log update
  - Write, msync, chmod, etc
  - Strictly commit log entry to NVMM before updating log tail
- Lightweight journaling for update across logs
  - Unlink, rename, etc
  - Journal log tails instead of metadata or data
- Copy-on-write for file data
  - Log only contains metadata
  - Log is short



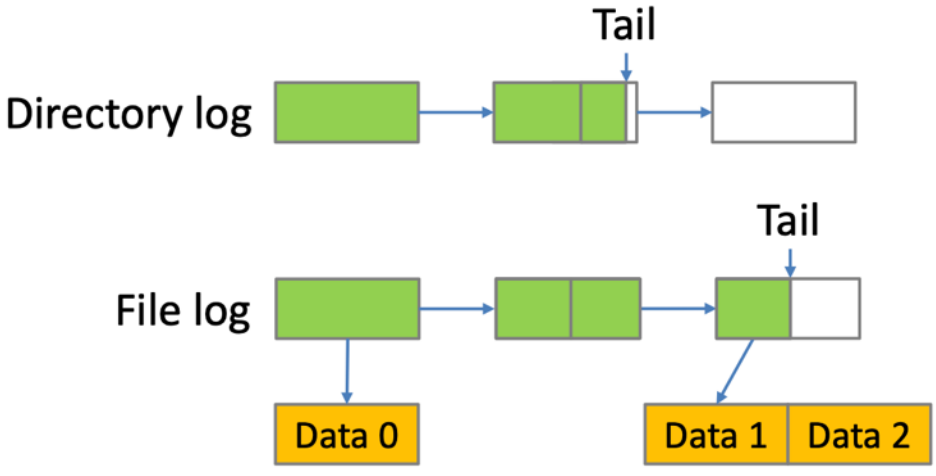
Atomicity

- Log-structuring for single log update
  - Write, msync, chmod, etc
  - Strictly commit log entry to NVMM before updating log tail
- Lightweight journaling for update across logs
  - Unlink, rename, etc
  - Journal log tails instead of metadata or data
- Copy-on-write for file data
  - Log only contains metadata
  - Log is short



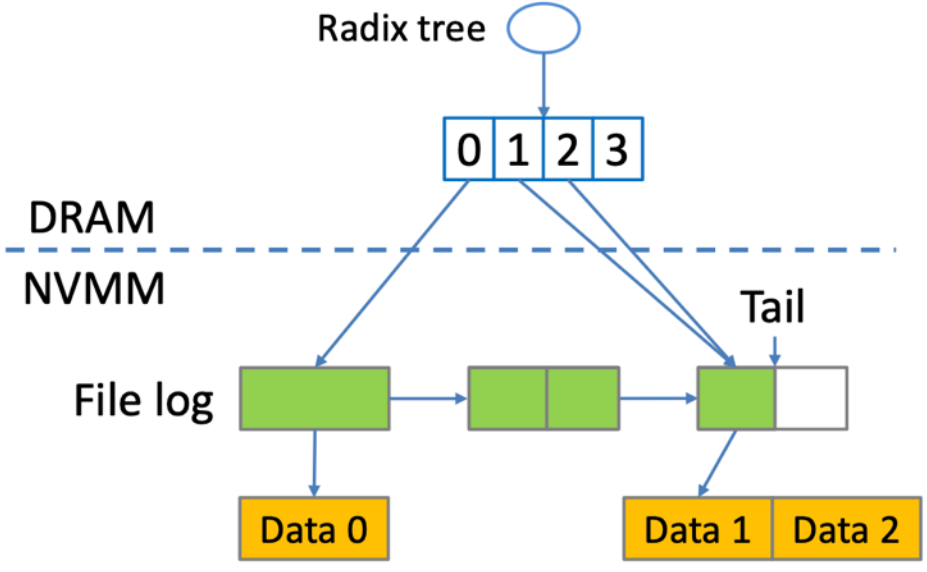
Performance

- Per-inode logging allows for high concurrency
- Split data structure between DRAM and NVM
  - Performance log is simple and efficient
  - Volatile tree structure has no consistency overhead



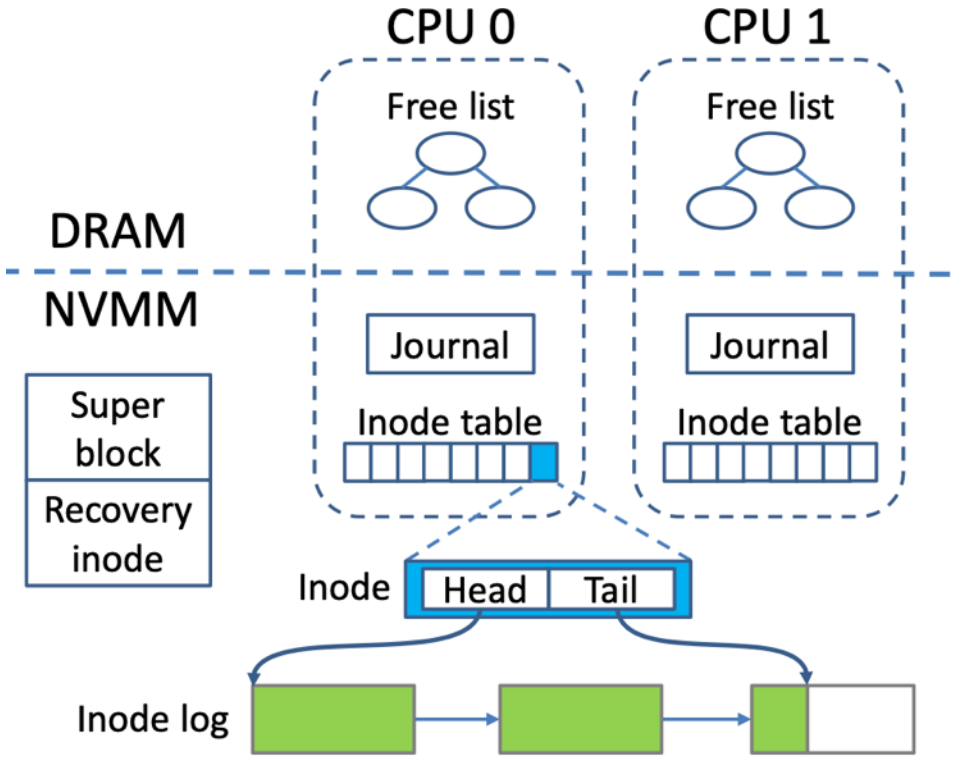
Performance

- Per-inode logging allows for high concurrency
- Split data structure between DRAM and NVM
  - Performance log is simple and efficient
  - Volatile tree structure has no consistency overhead



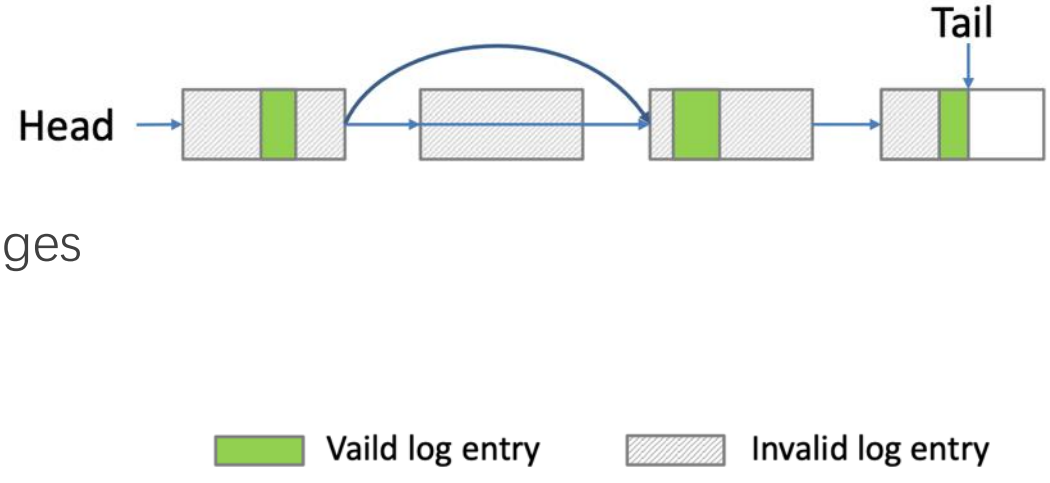
NOVA layout

- Put allocator in DRAM
- High scalability
  - Per-CPU NVM free list, journal and inode table
  - Concurrent transactions and allocation/deallocation



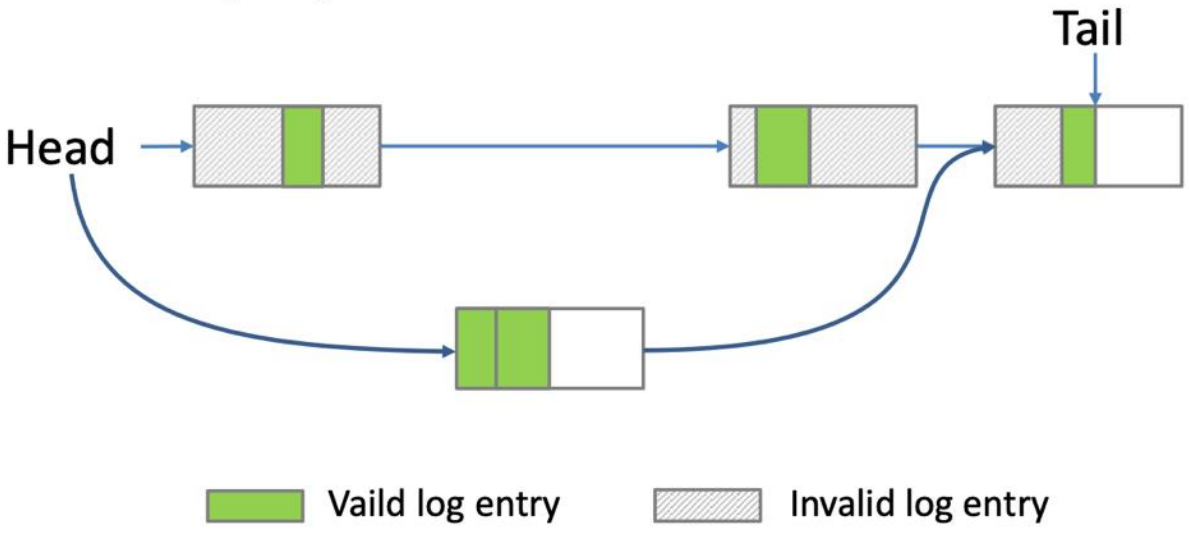
Fast garbage collection

- Log is a linked list
- Log only contains metadata
- Fast GC deletes dead log pages from the linked list
- No copying



Thorough garbage collection

- Starts if valid log entries < 50% log length
- Format a new log and atomically replace the old one
- Only copy metadata



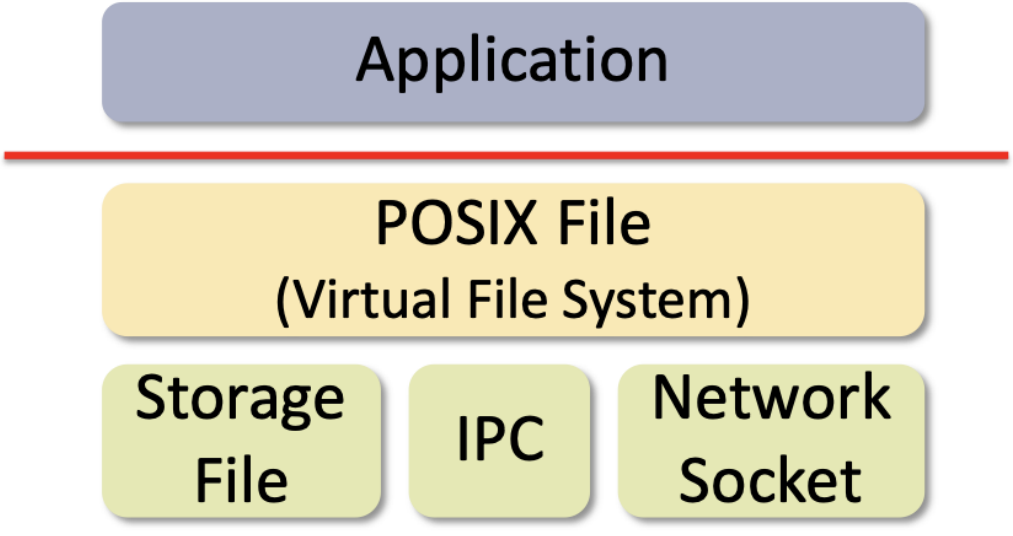
A final

CASE STUDY: AERIE

[Volos, EuroSys'14]

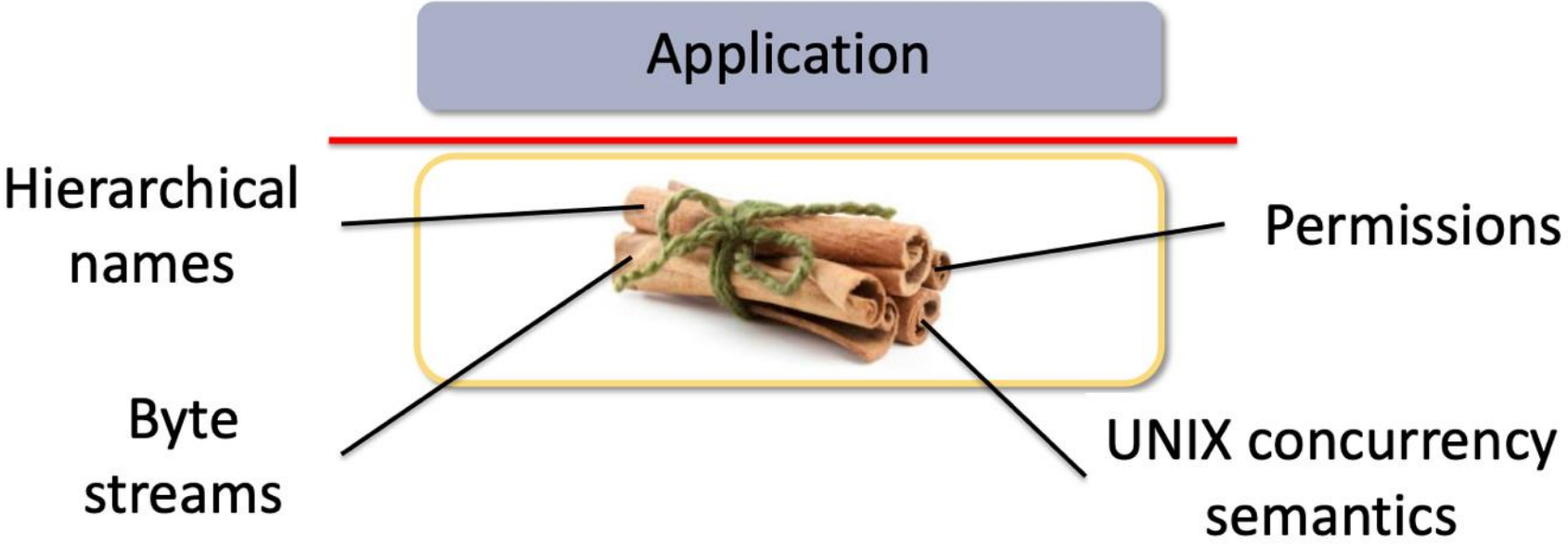
POSIX File: Expensive abstraction

- Universal abstraction: *Everything is a file*
  - Has generic-overhead cost



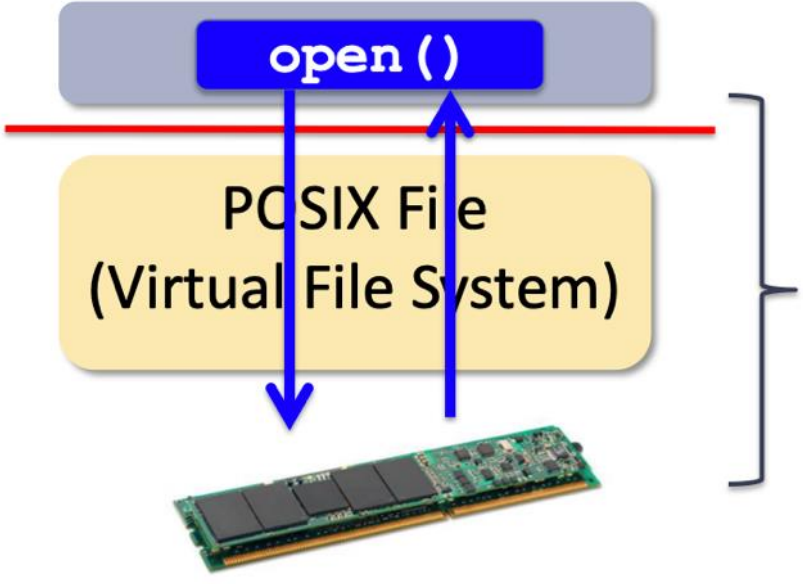
POSIX File: Expensive abstraction

- Rigid interface and policies
  - Has fixed components and costs
  - Hinders application-specific customization



POSIX File: Expensive abstraction

- **Generic-overhead** costs
- **Rigid interfaces** and policies



~2.5us ≈ 25x NVM latency

Customizing the file system today

- Modify the kernel
- Add a layer over existing kernel file system
- Use a user-mode framework such as FUSE

Need flexible interfaces

Kernel safely multiplexes NVM

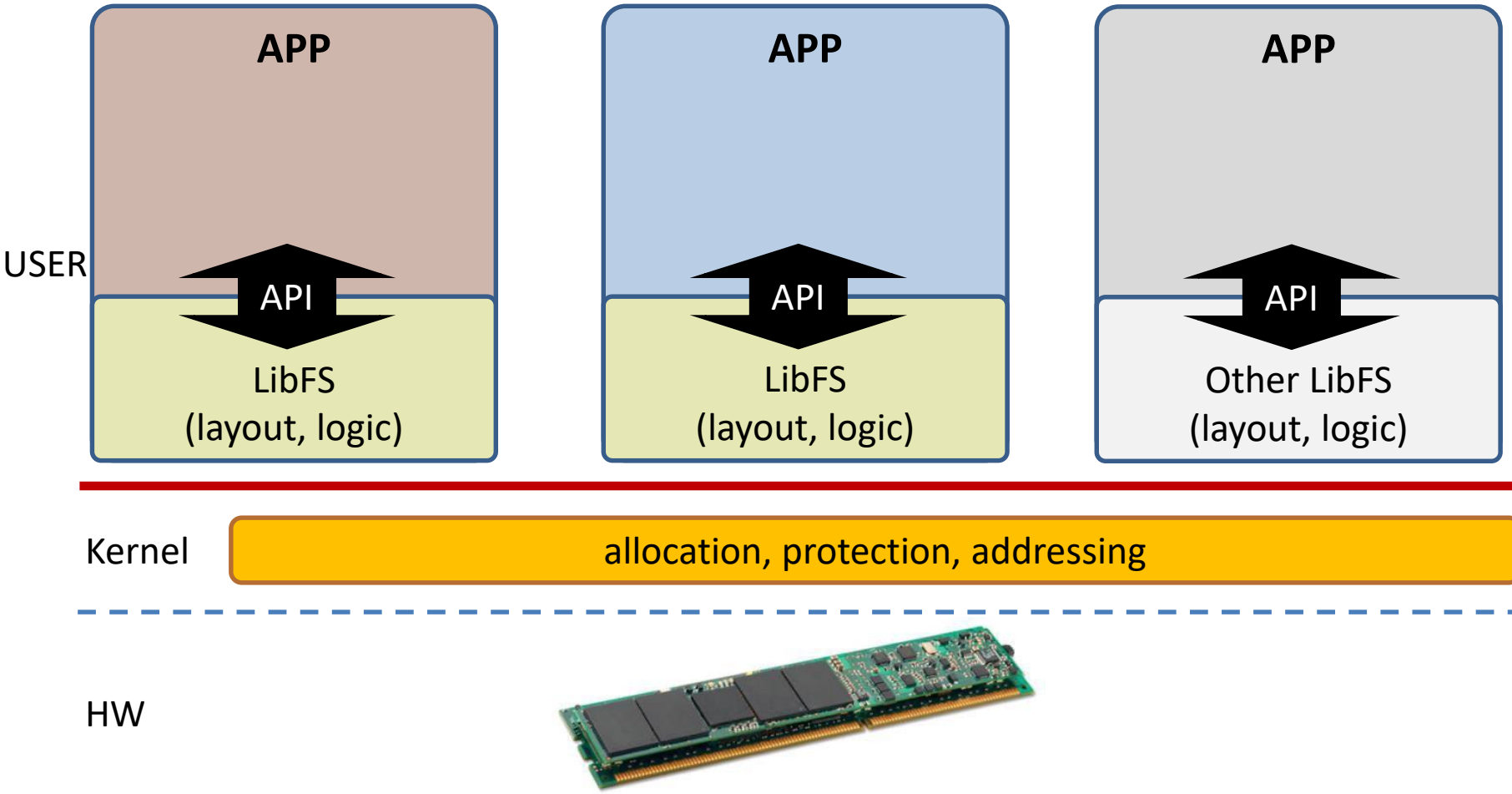
- Allocation: allocates NVM regions
- Addressing: Memory-maps NVM regions
- Protection: Keep track of region access rights

Kernel

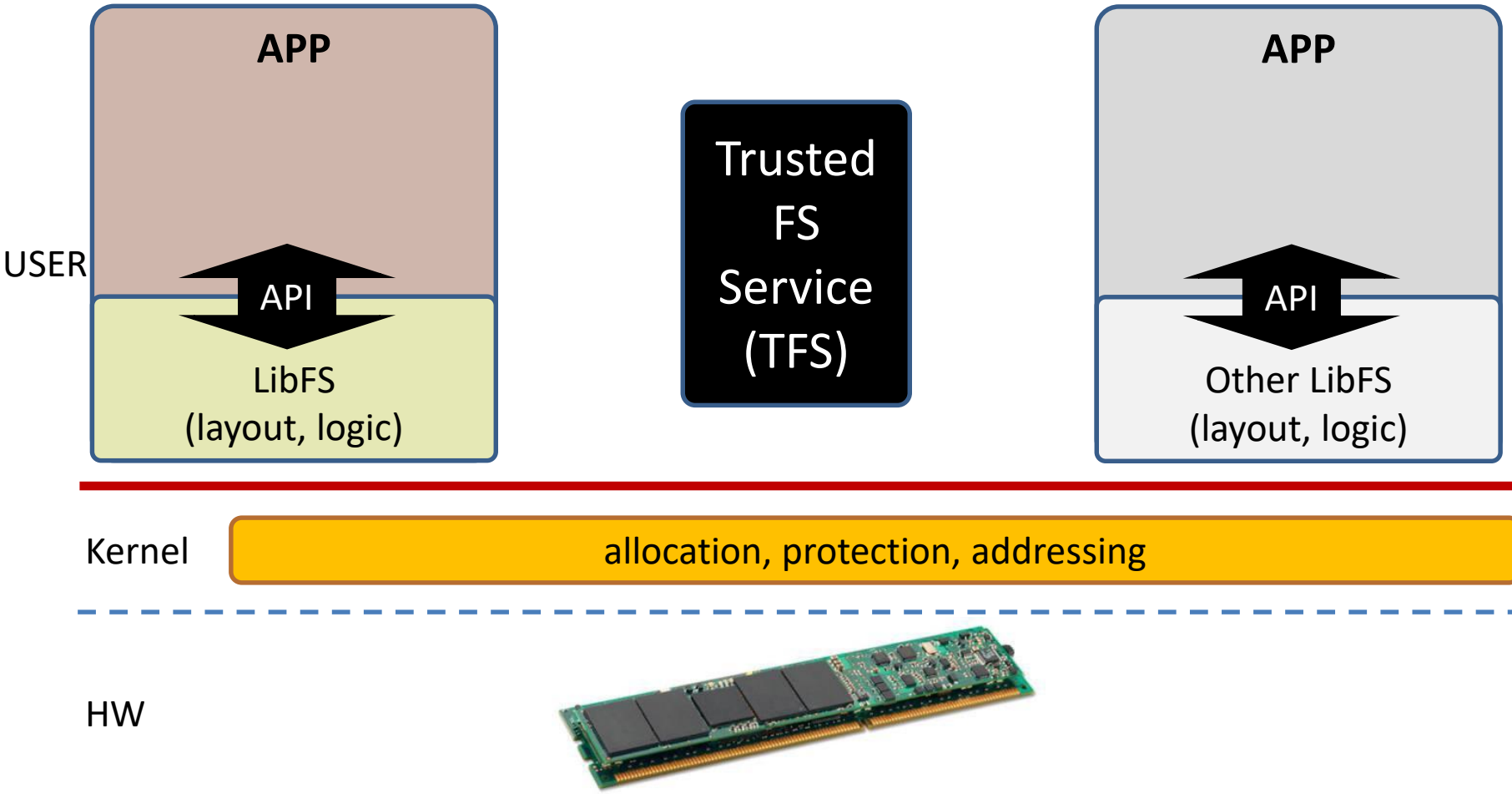
allocation, protection, addressing

HW

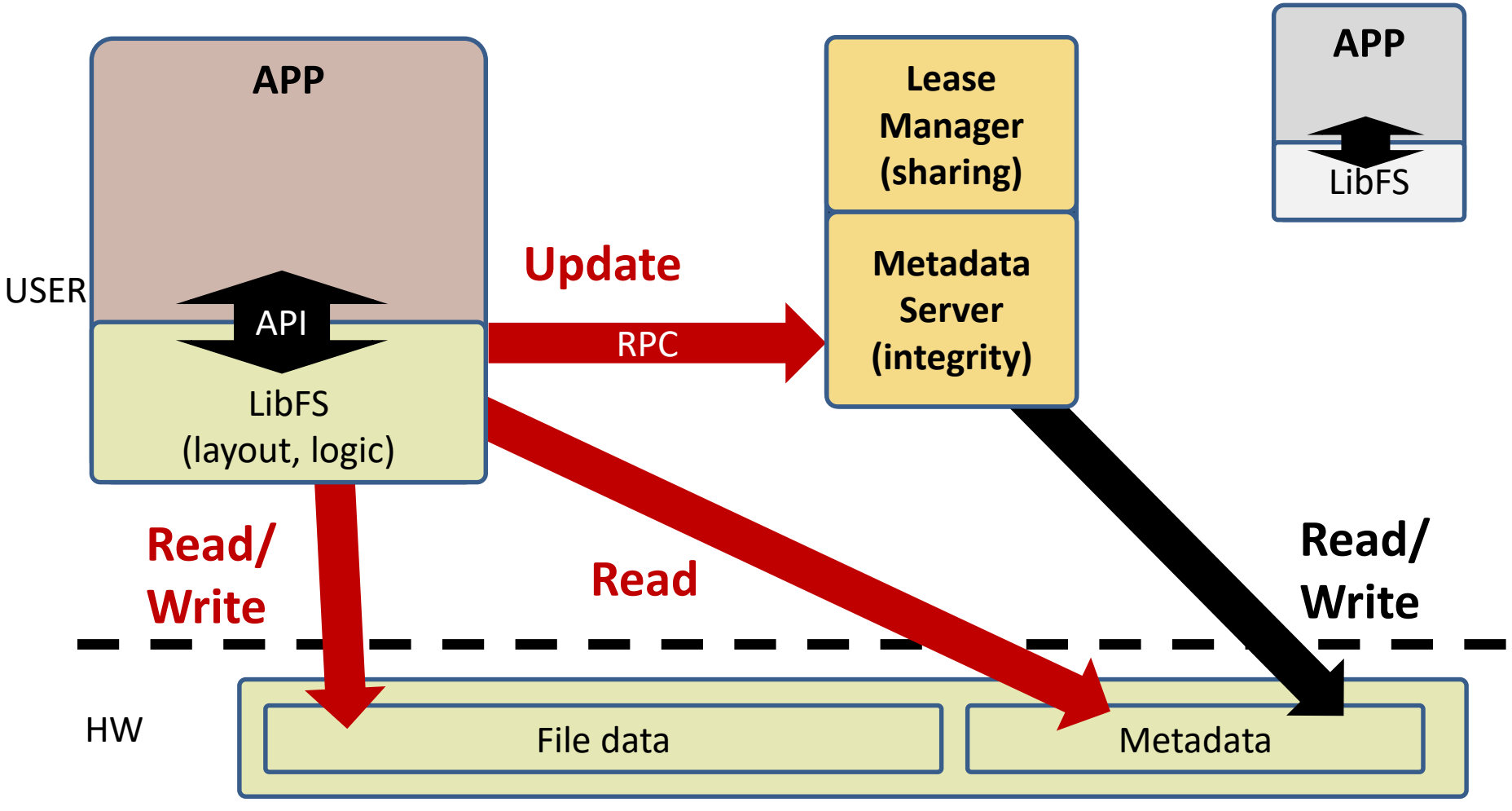
Library implements functionality



Integrity via Trusted File Service



Integrity via Trusted File Service



QUESTIONS