ctFS:

Replacing File Indexing with Hardware Memory Translation through Contiguous File Allocation for Persistent Memory

Ruibin Li

Xiang Ren

Xu Zhao

Siwei He

Michael Stumm

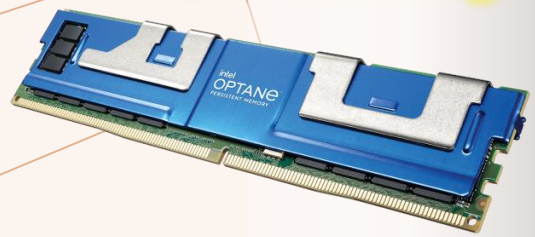
Ding Yuan

UNIVERSITY OF TORONTO

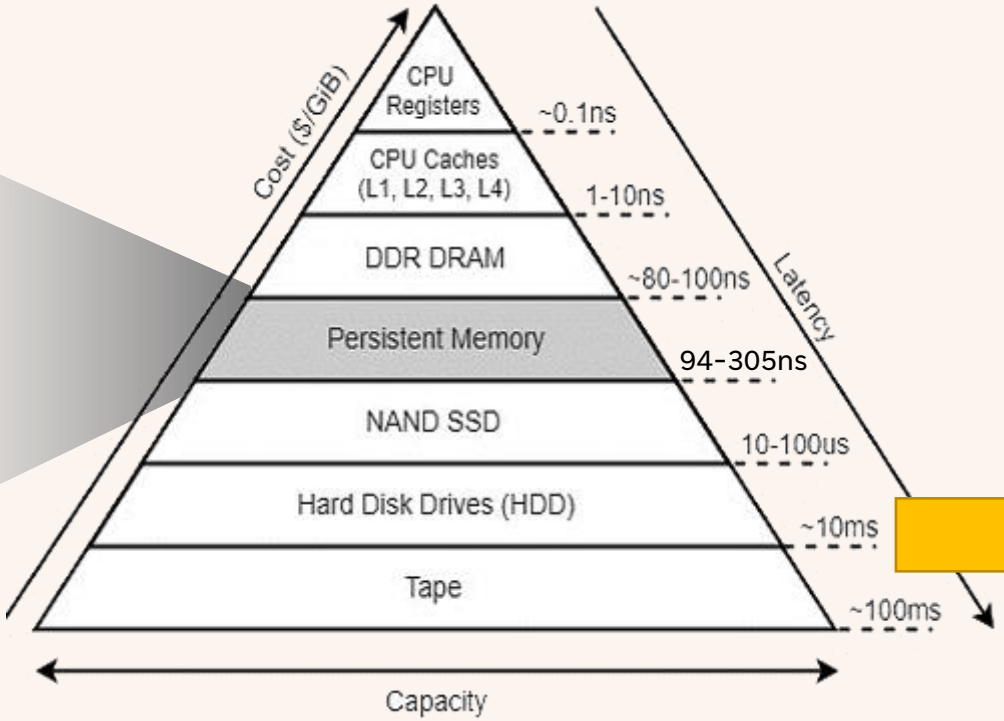
Persistent Memory

-Future of Computer Storage

LOW Latency



Persistency



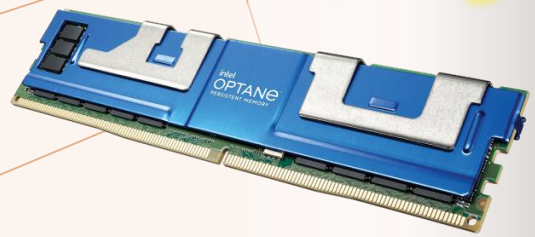
Access Time Breakdown

Source: https://docs.pmem.io/persistent-memory/getting-started-guide/introduction

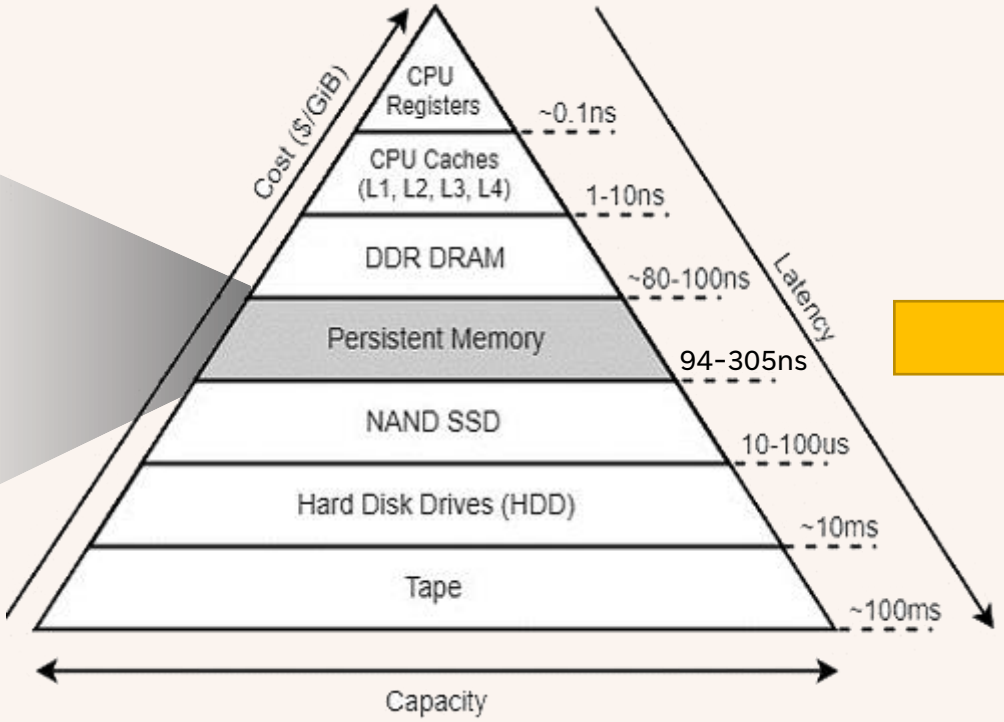
Persistent Memory

-Future of Computer Storage

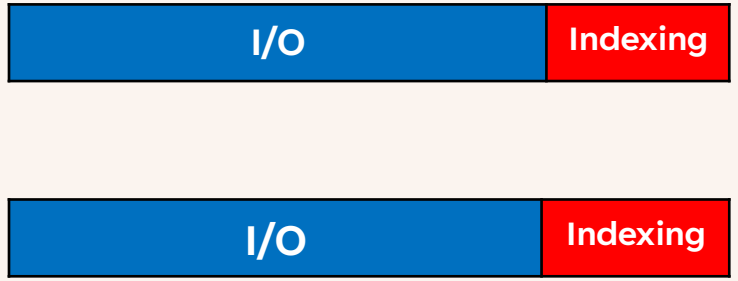
LOW Latency



Persistency



Access Time Breakdown

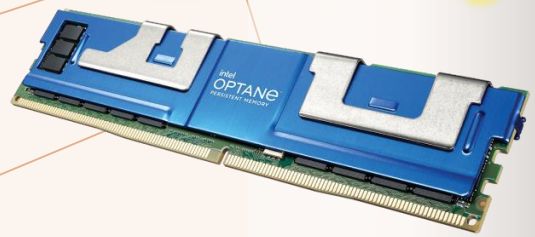


Source: https://docs.pmem.io/persistent-memory/getting-started-guide/introduction

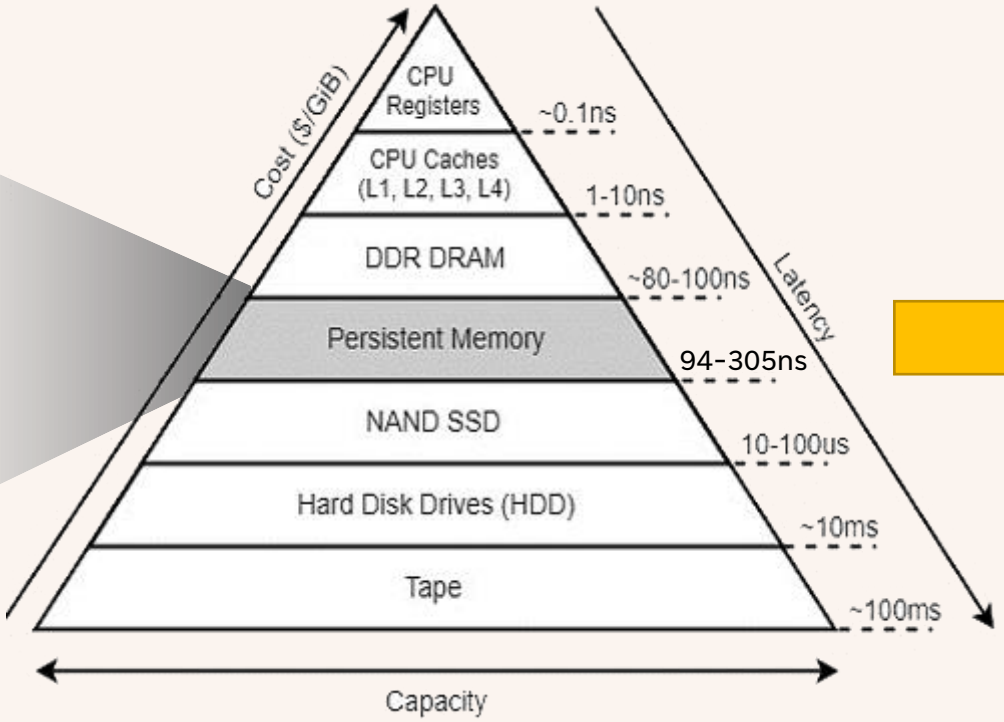
Persistent Memory

-Future of Computer Storage

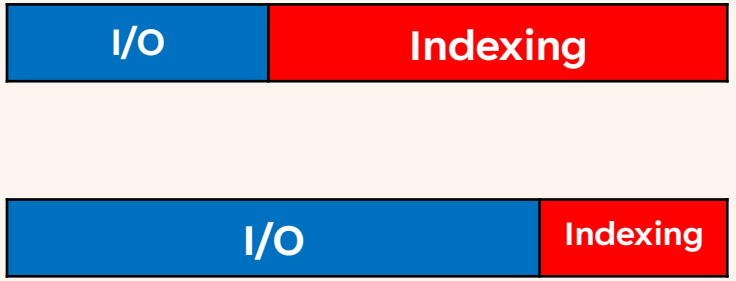
LOW Latency



Persistency

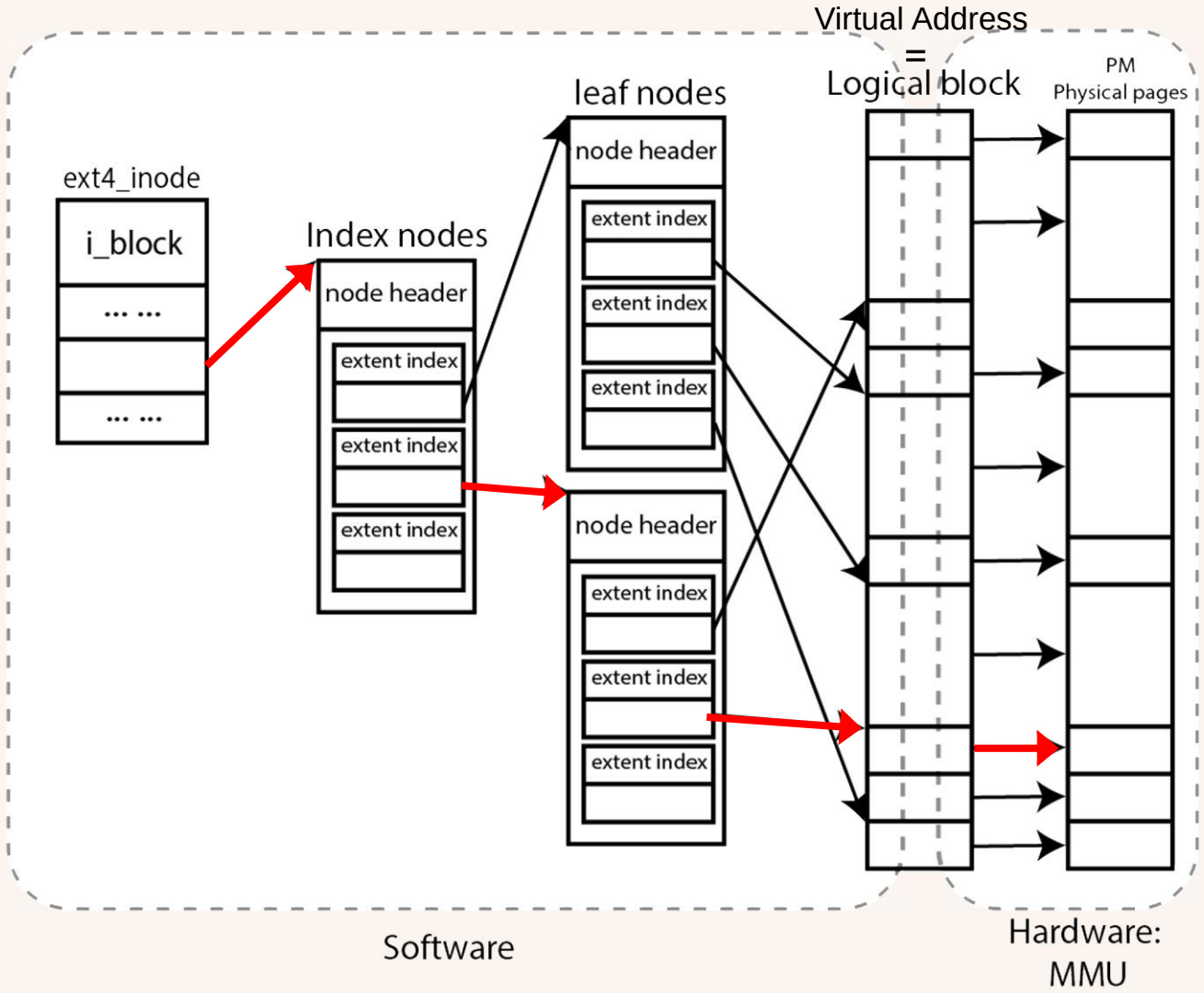


Access Time Breakdown



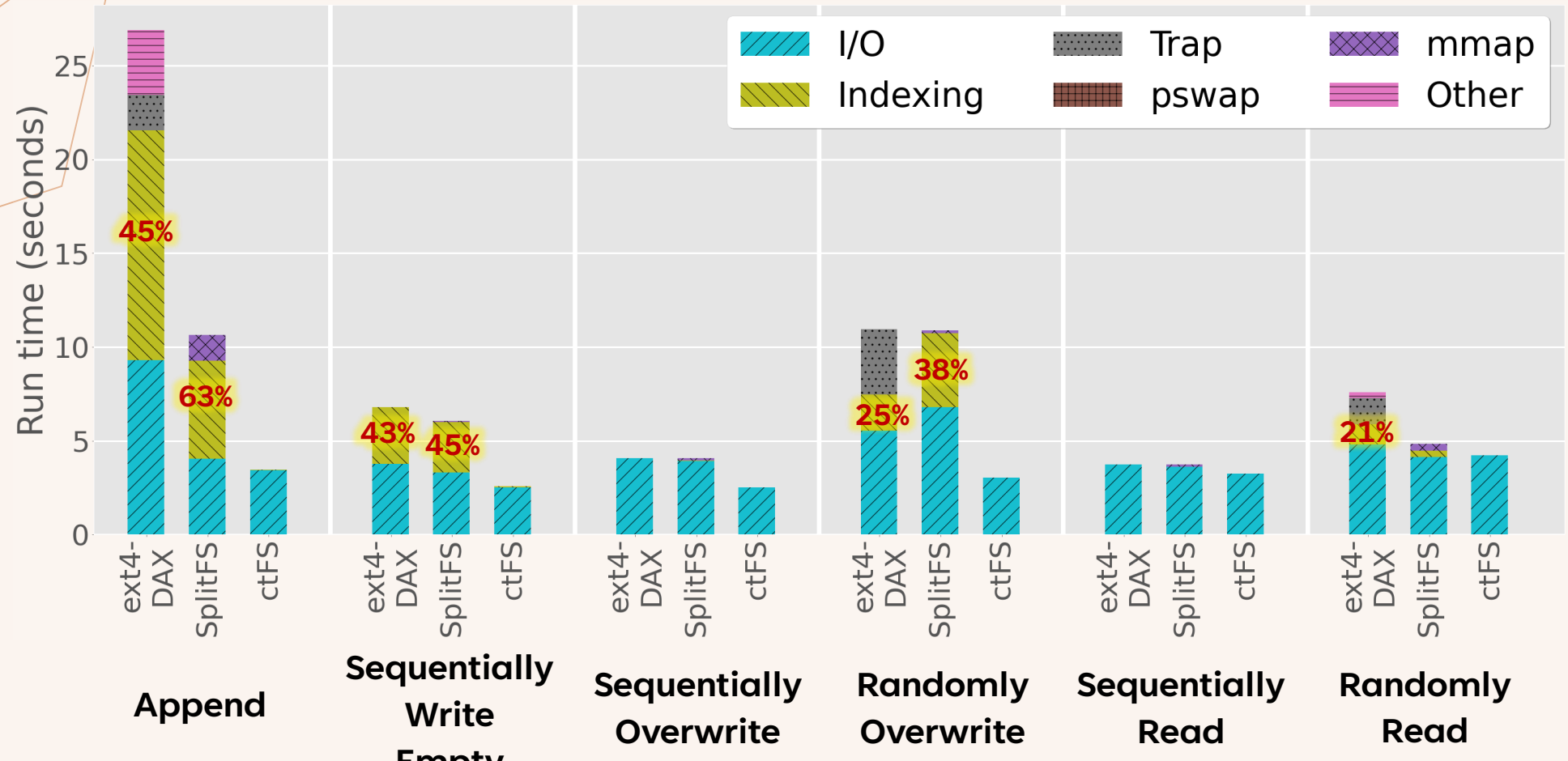
Source: https://docs.pmem.io/persistent-memory/getting-started-guide/introduction

Ext4-DAX Indexing Path is COMPLEX!



*Top is lower address

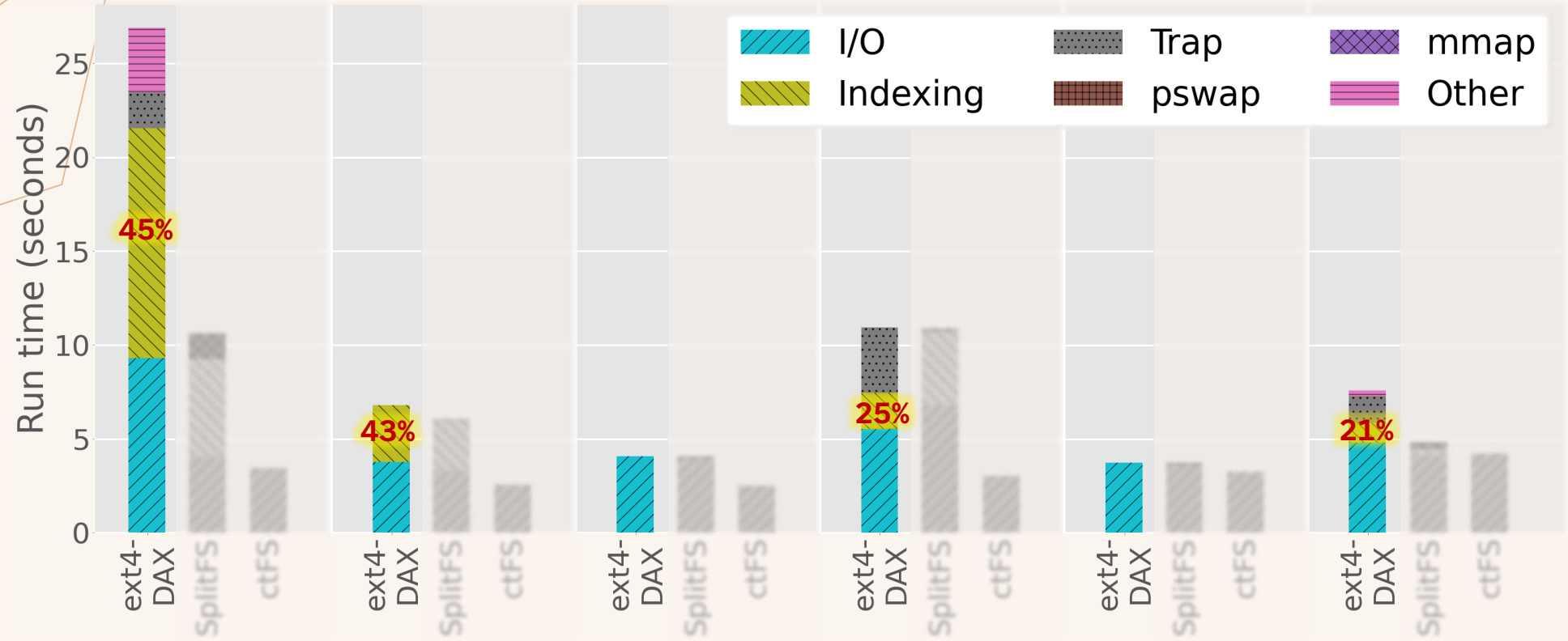
Software Indexing is Slow!



*10GB Total Workload for each test
*Lower is better

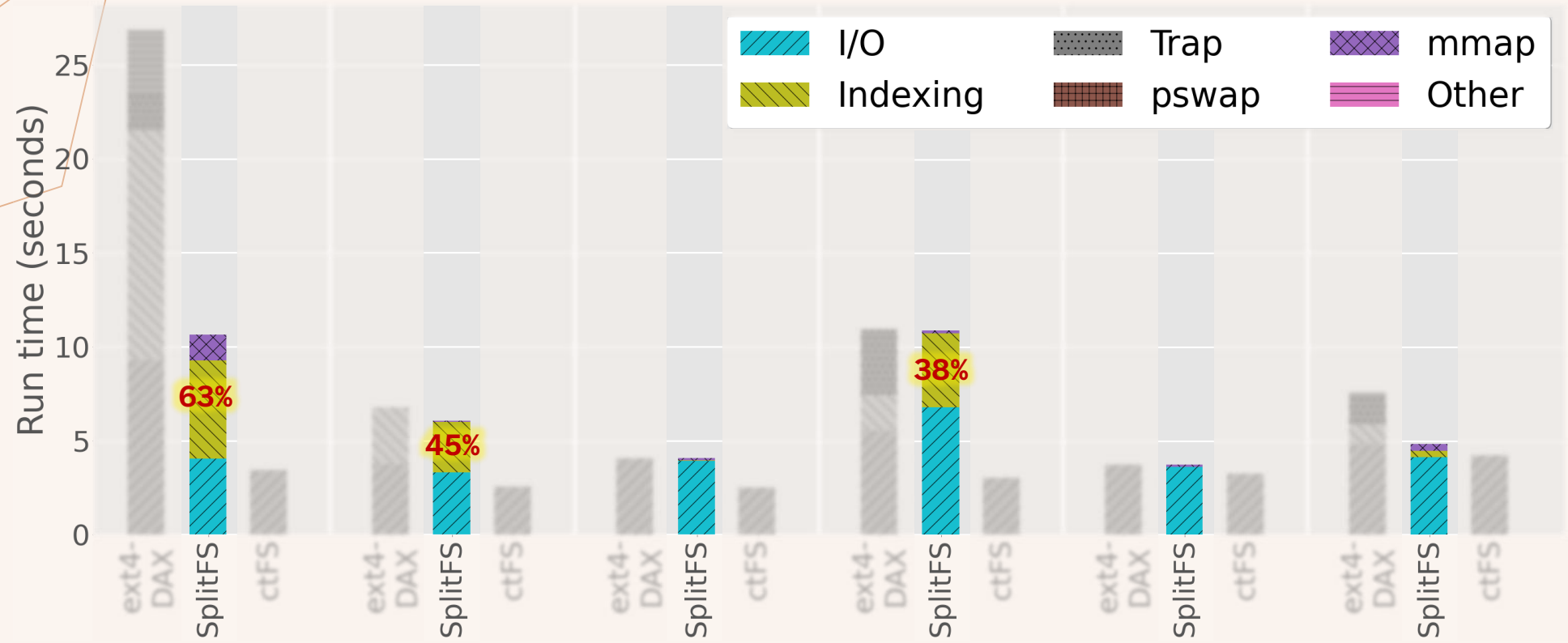
Source: SplitFS[SOSP19]. Rohan Kadekodi, et al.

Software Indexing is Slow!



*10GB Total Workload for each test
*Lower is better

Software Indexing is Slow!



*10GB Total Workload for each test
*Lower is better

Source: SplitFS[SOSP19]. Rohan Kadekodi, et al.

OTHER RELATED WORKS

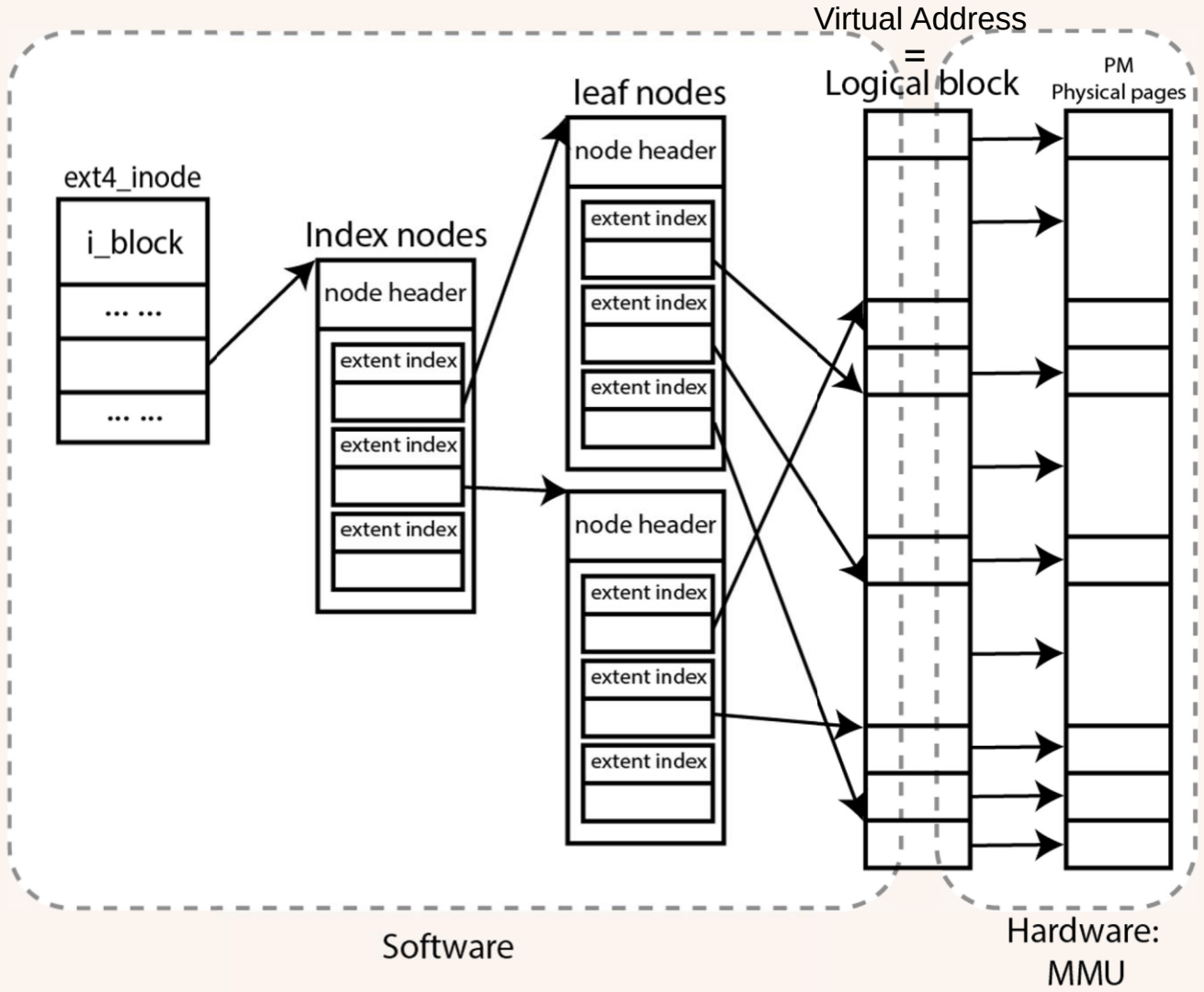
HashFS [FAST ’21] Neal, et al.

Use hash table for indexing, built on Strata

KUCO [FAST ’21] Chen, et al.

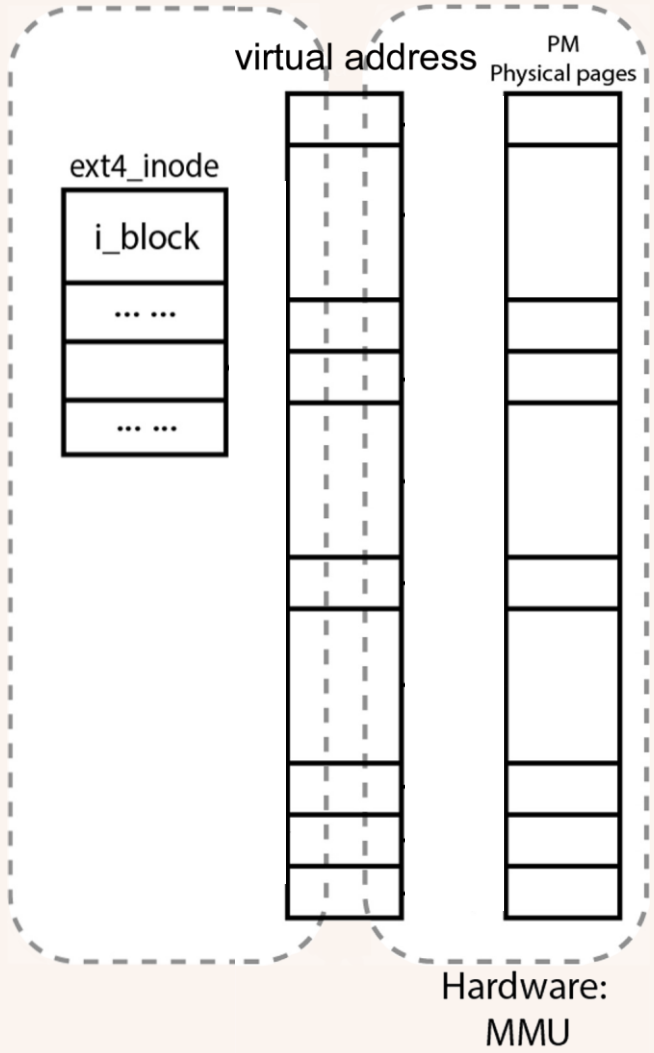
Offload some of the indexing job to user level, based on ext2-style block mapping

RETHINK Indexing!



*Top is lower address

RETHINK Indexing!



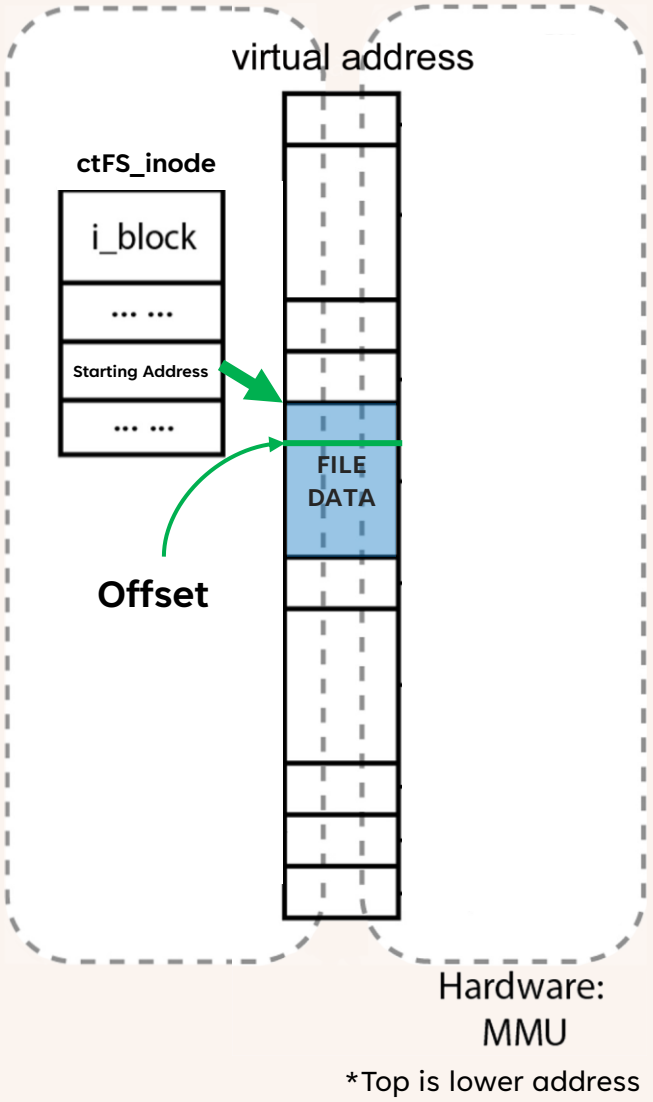
*Top is lower address

Replace Software Indexing With MMU Translation

Treat PM as Memory

Instead of Disks

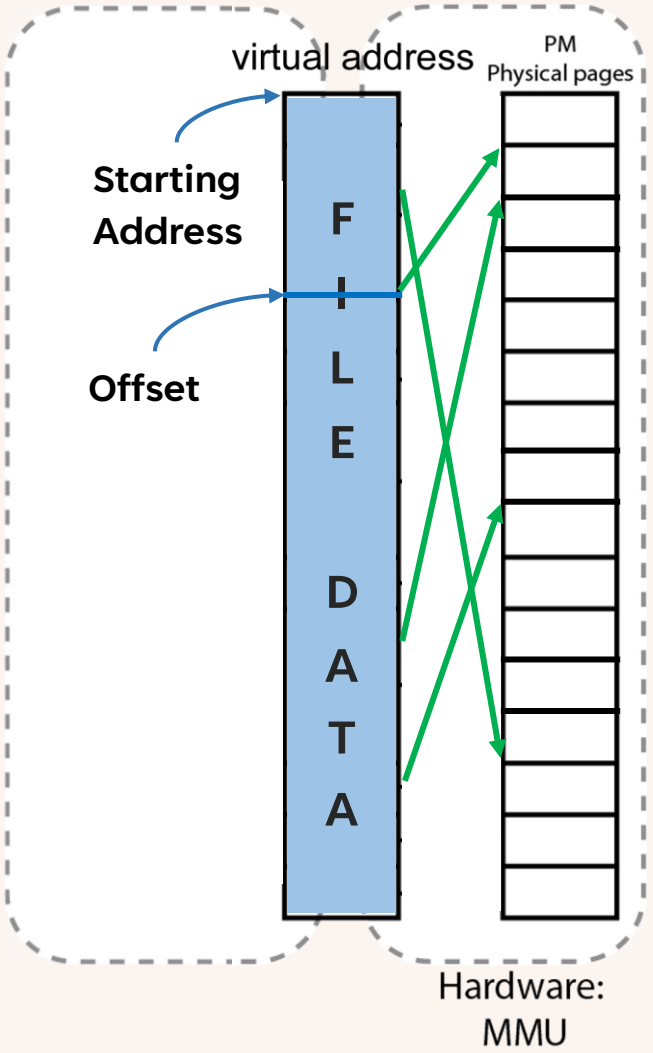
ctFS: Replace Indexing with Hardware Memory Translation



*Top is lower address

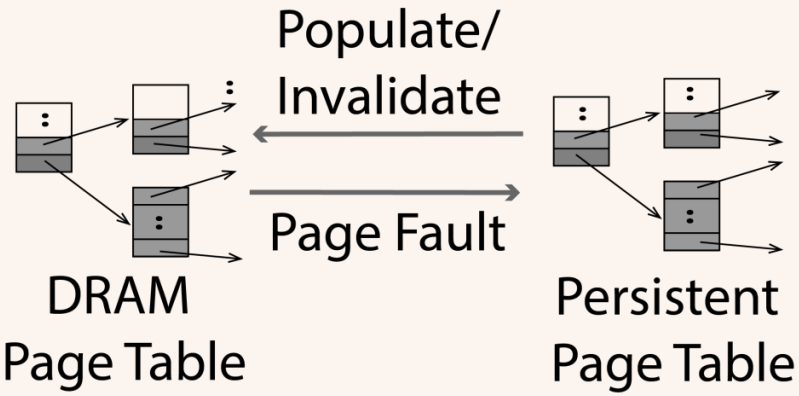
- Files are **Contiguously** Allocated in **Virtual** Address Space
  - Only store Starting Address in inode
  - File access -> start + offset

ctFS: Replace Indexing with Hardware Memory Translation



*Top is lower address

- Virtual to physical mapping is managed by **Persistent Page Tables**
  - Copy to DRAM on page fault
  - Demand Paging



ctFS: Replace Indexing with Hardware Memory Translation

ctU: User Part

FS Functionalities

File System APIs

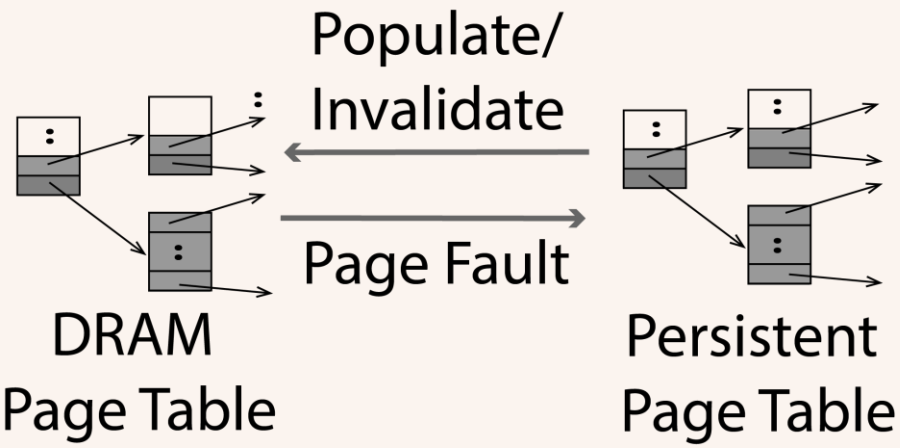
File Layout

ACID Guarantees

⋮

ctK: Kernel Part

Memory Abstractions

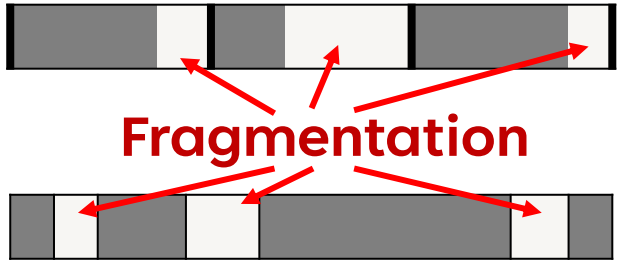


Challenges of being Contiguous

Occupied

Free

Allocate?



- Fixed-size Partition: Internal Fragmentation
- Variable-size Partition: External Fragmentation and Garbage Collection

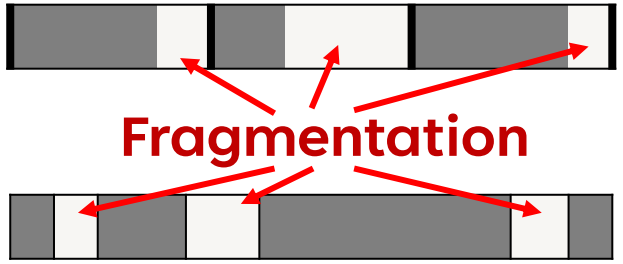
Hierarchical Layout

Challenges of being Contiguous

Occupied

Free

Allocate?



- Fixed-size Partition: Internal Fragmentation
- Variable-size Partition: External Fragmentation and Garbage Collection

Resize?

- A file grows and conflicts with another
- Need Migration

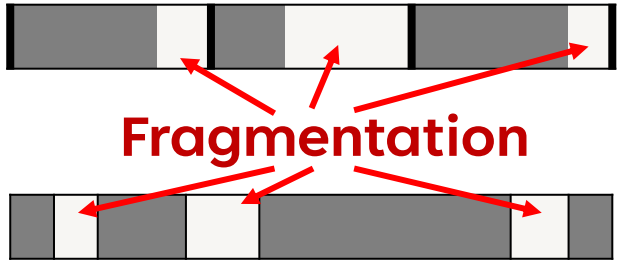
Hierarchical Layout

Challenges of being Contiguous

Occupied

Free

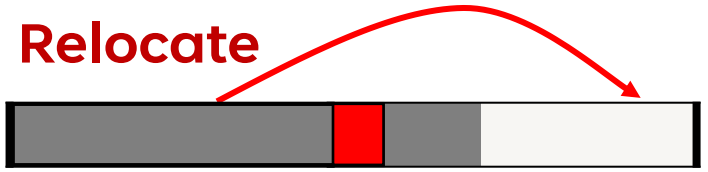
Allocate?



- Fixed-size Partition: Internal Fragmentation
- Variable-size Partition: External Fragmentation and Garbage Collection

Hierarchical Layout

Resize?

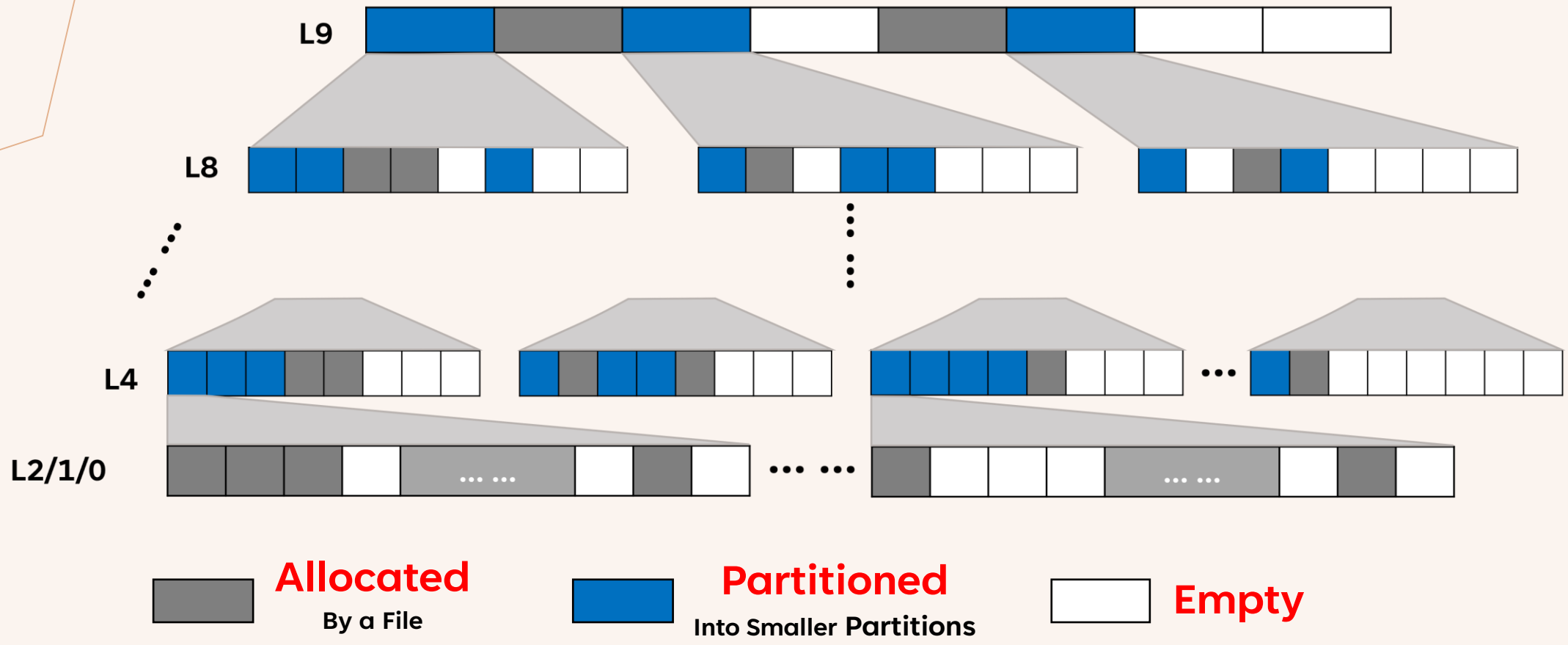


- A file grows and conflicts with another
- Need Migration

New Syscall: pswap()

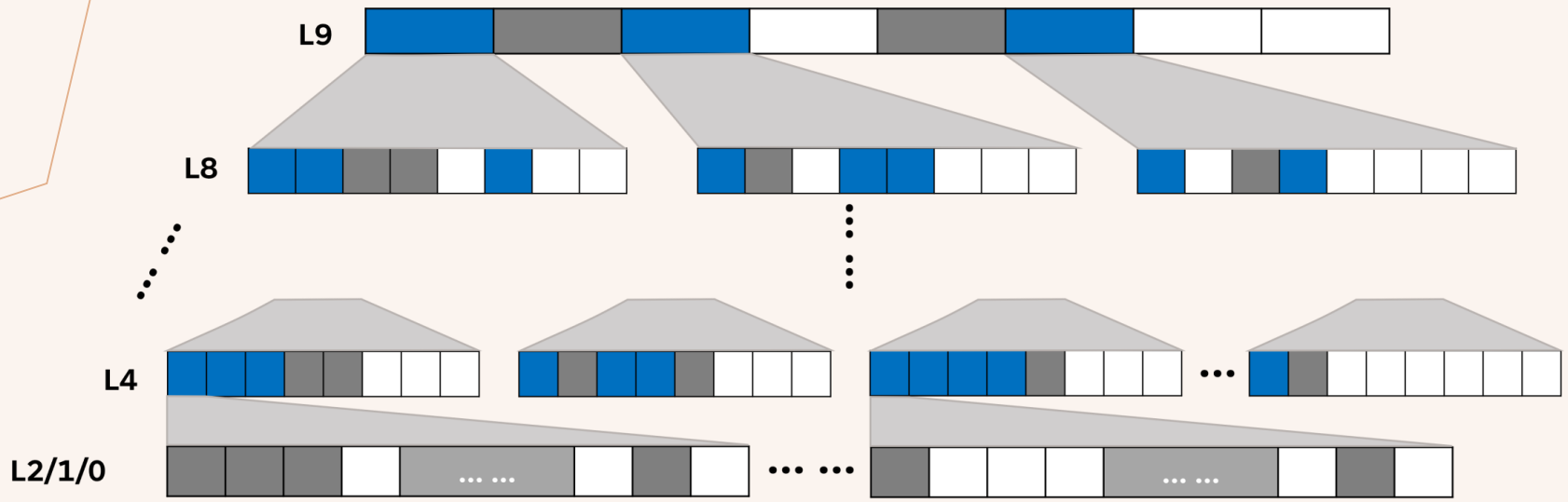
ctFS: Hierarchical File System Layout

L9 512GB
L8 64GB
L7 8GB
L6 1GB
L5 128MB
L4 16MB
L3 2MB
L2 256KB
L1 32KB
L0 4KB



ctFS: **Hierarchical** File System Layout

L9 512GB
L8 64GB
L7 8GB
L6 1GB
L5 128MB
L4 16MB
L3 2MB
L2 256KB
L1 32KB
L0 4KB



Allocated
By a File

Partitioned
Into Smaller Partitions

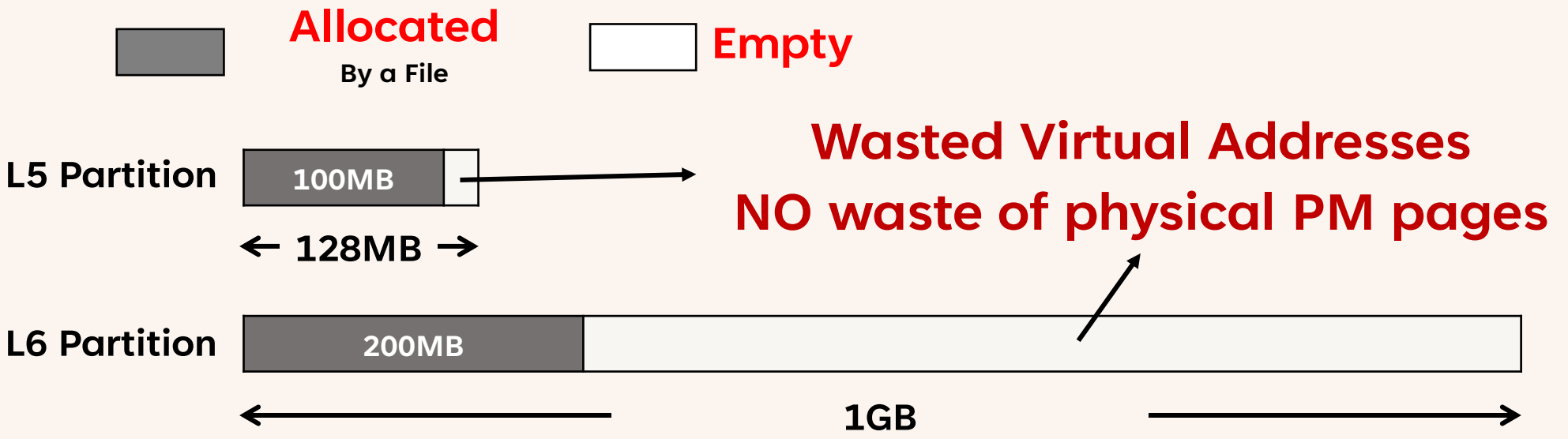
Empty

- Partition of different levels
- Size is **8x** between levels
- Starting Address is **Aligned** to its size

L9 512GB
L8 64GB
L7 8GB
L6 1GB
L5 128MB
L4 16MB
L3 2MB
L2 256KB
L1 32KB
L0 4KB

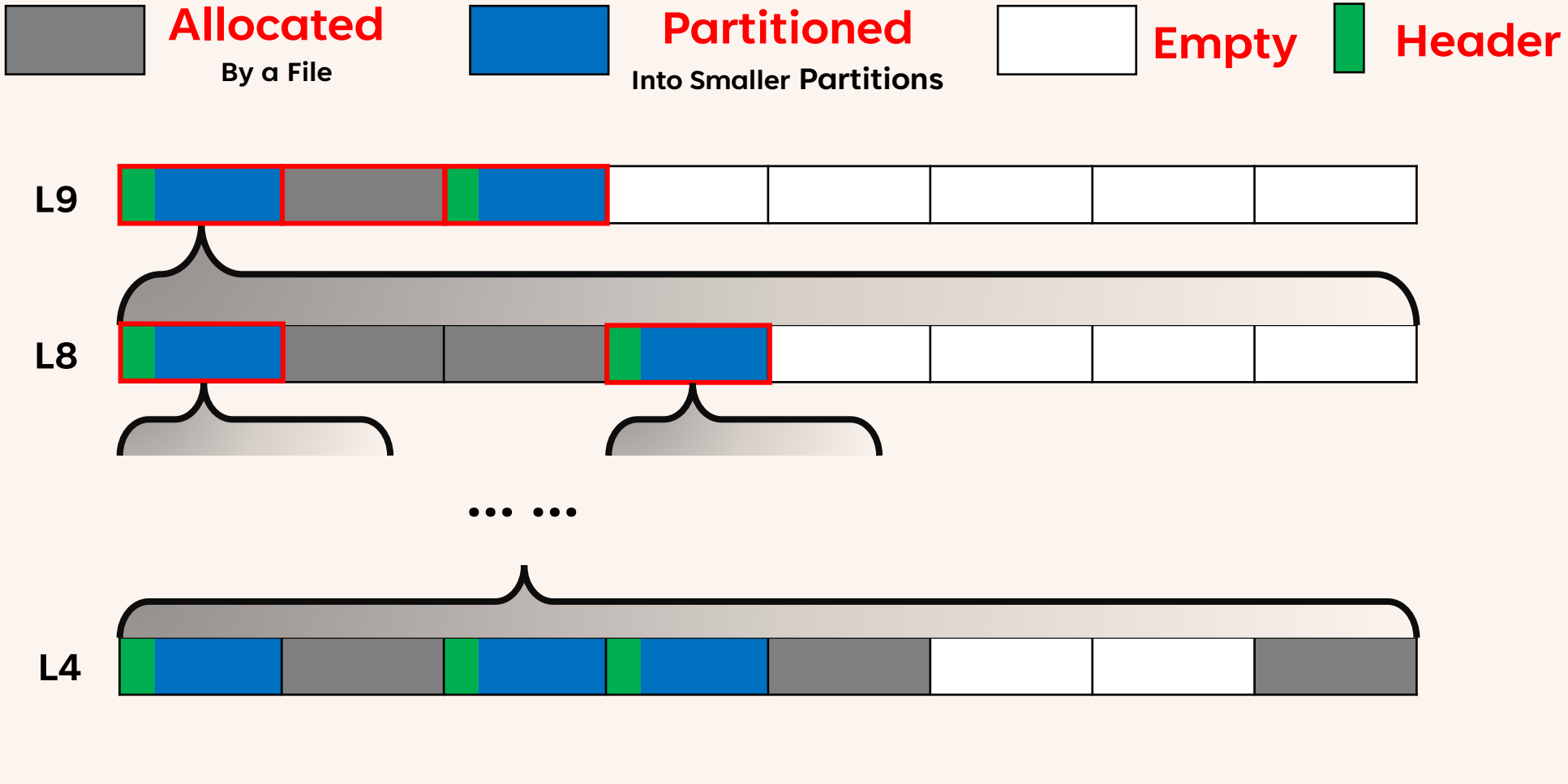
ctFS: **Hierarchical** File System Layout

- **One** file **One** partition
- Occupied by a file: **Allocated**

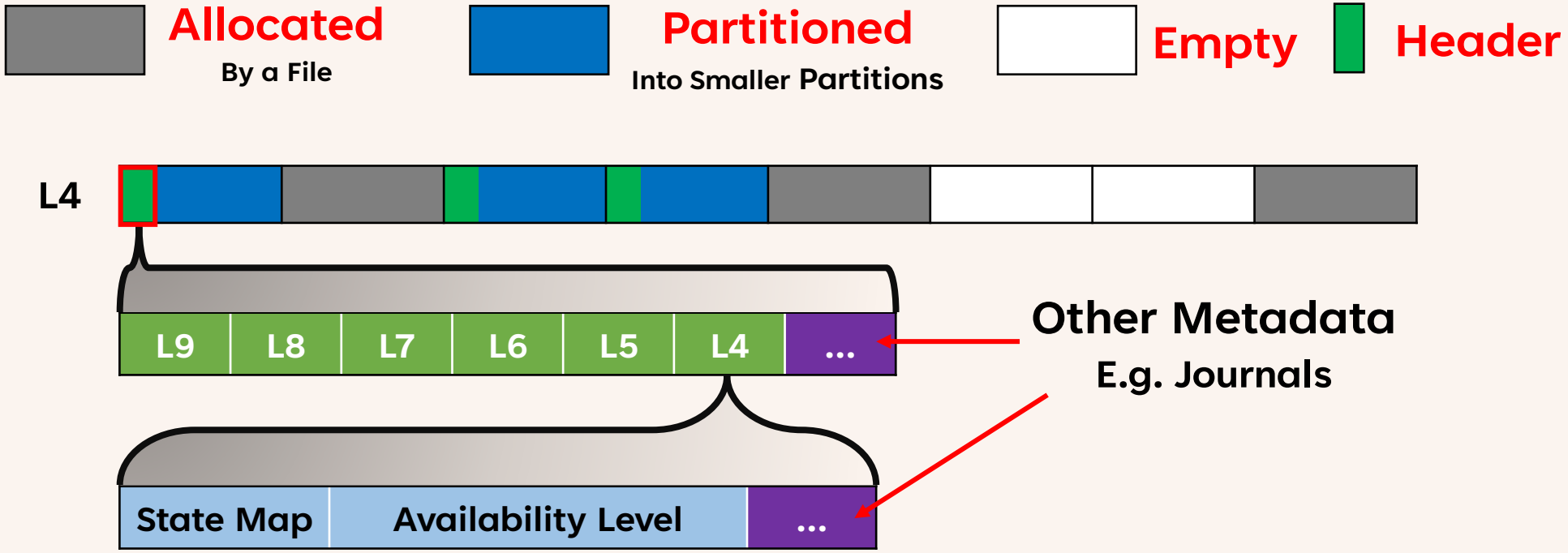


ctFS: Hierarchical File System Layout

L9 512GB
L8 64GB
L7 8GB
L6 1GB
L5 128MB
L4 16MB
L3 2MB
L2 256KB
L1 32KB
L0 4KB

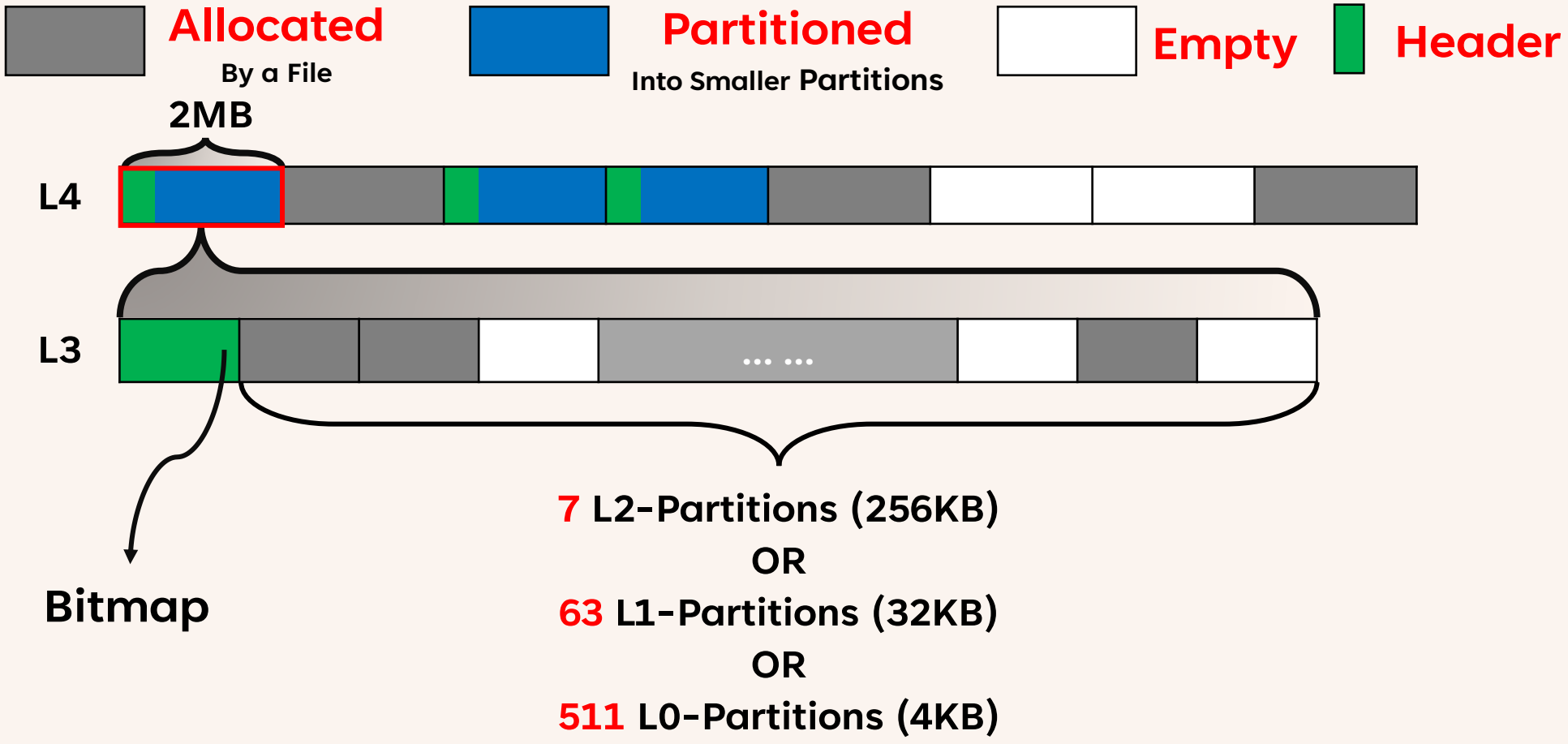


ctFS: Hierarchical File System Layout



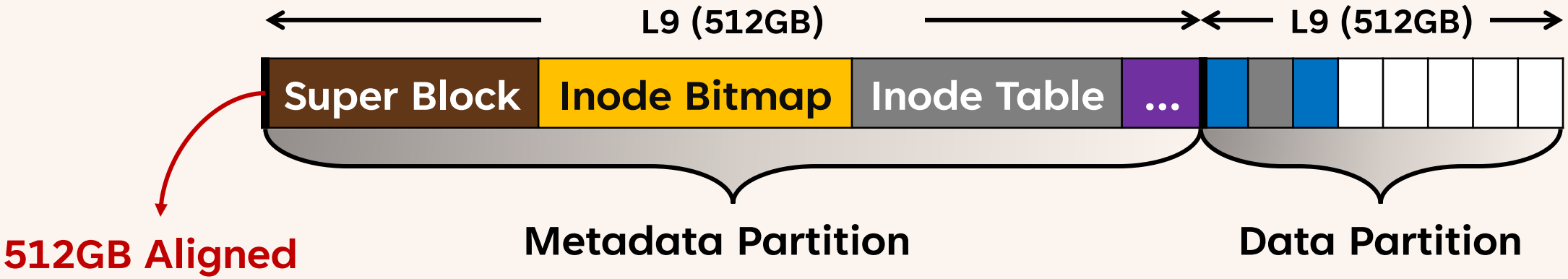
Header Layout

ctFS: Hierarchical File System Layout



ctFS: Hierarchical File System Layout

- Highest Level ≥ 2 x PM Capacity
  - E.g. 256GB PM needs L9
- Two highest level partitions are reserved
  - Metadata Partition and Data Partition
  - E.g. Two L9-Partitions reserved below

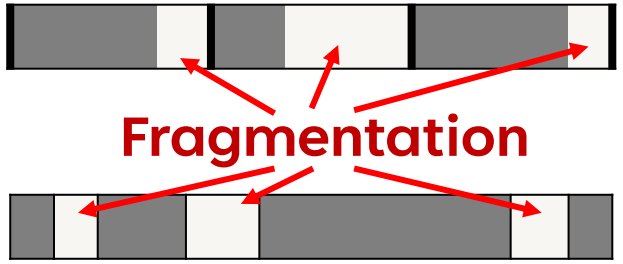


Challenges of being Contiguous

Occupied

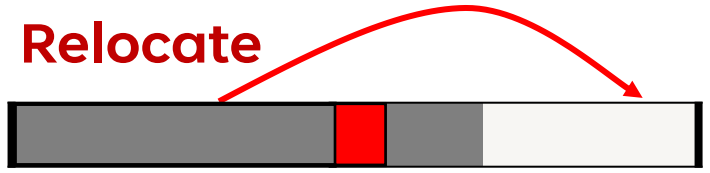
Free

Allocate?



- Fixed-size Partition: Internal Fragmentation
- Variable-size Partition: External Fragmentation and Garbage Collection

Resize?



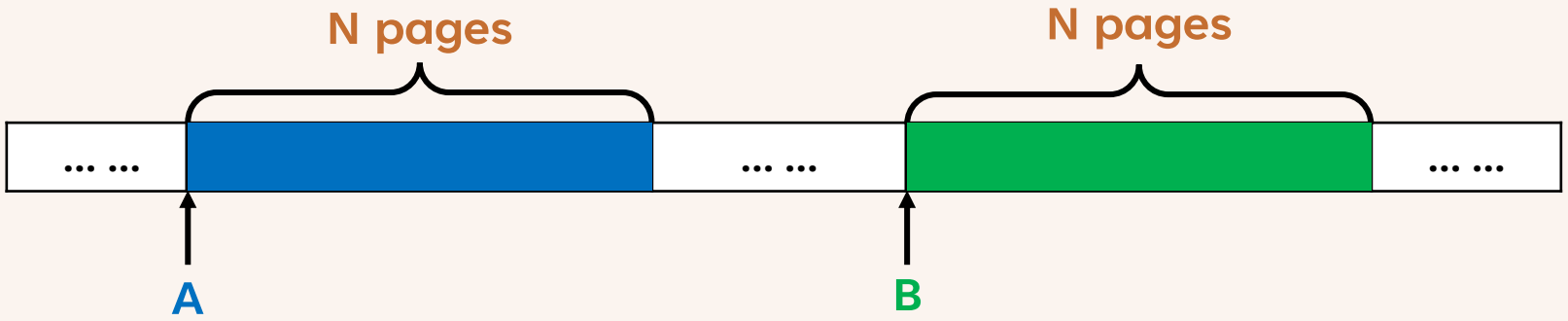
- A file grows and conflicts with another
- Need Migration

Hierarchical Layout

New Syscall: pswap()

ctFS: New Syscall: pswap()

```
int pswap(void* A, void* B, unsigned int N, ...);
```

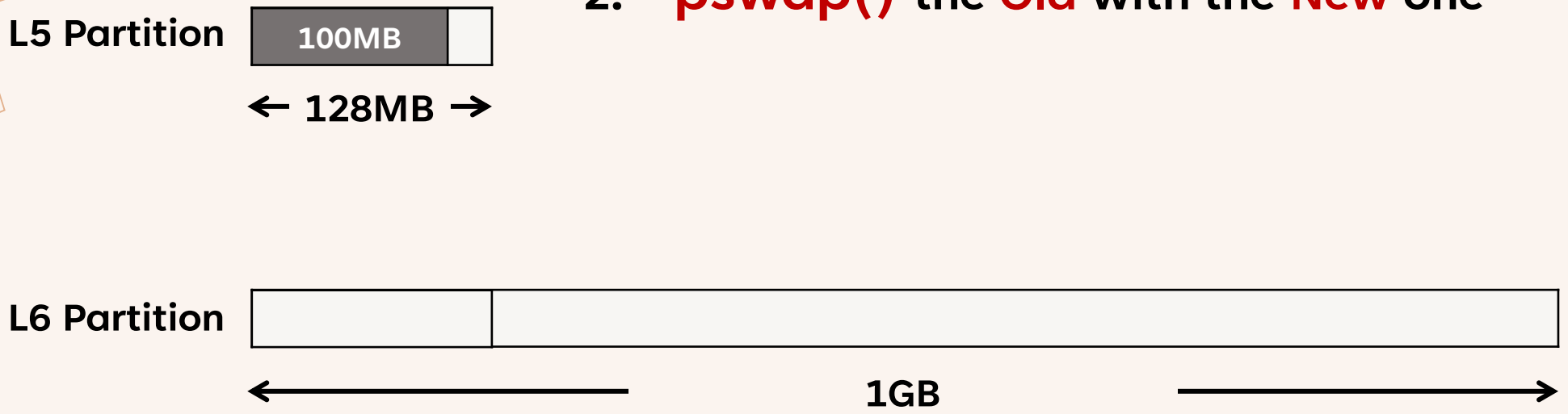


- Atomically swap the mapping of two **Same-Sized Contiguous** sequences of virtual pages
- Crash consistency guaranteed using REDO log

Use pswap() To Resize

```
int pswap(void* A, void* B, unsigned int N, int* flag);
```

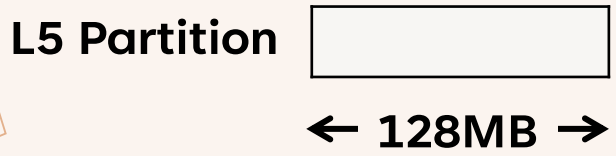
1. Allocate an Empty Higher-Level Partition
2. pswap() the Old with the New one

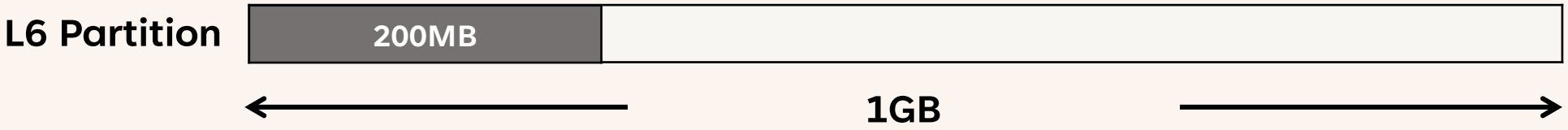


Use pswap() To Resize

```
int pswap(void* A, void* B, unsigned int N, int* flag);
```

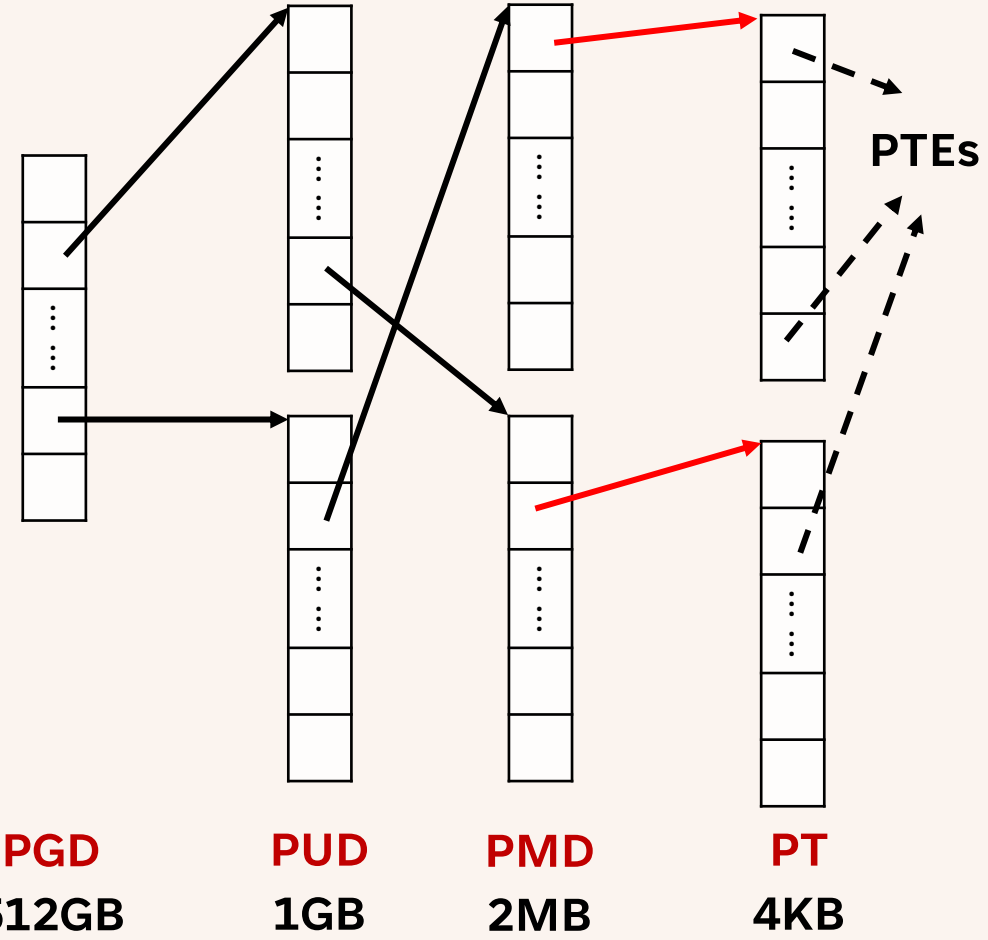
1. Allocate an **Empty** Higher-Level Partition
2. **pswap()** the **Old** with the **New** one
3. Write the **Extra Data** and Alter the Metadata





pswap()'s Key to Efficiency

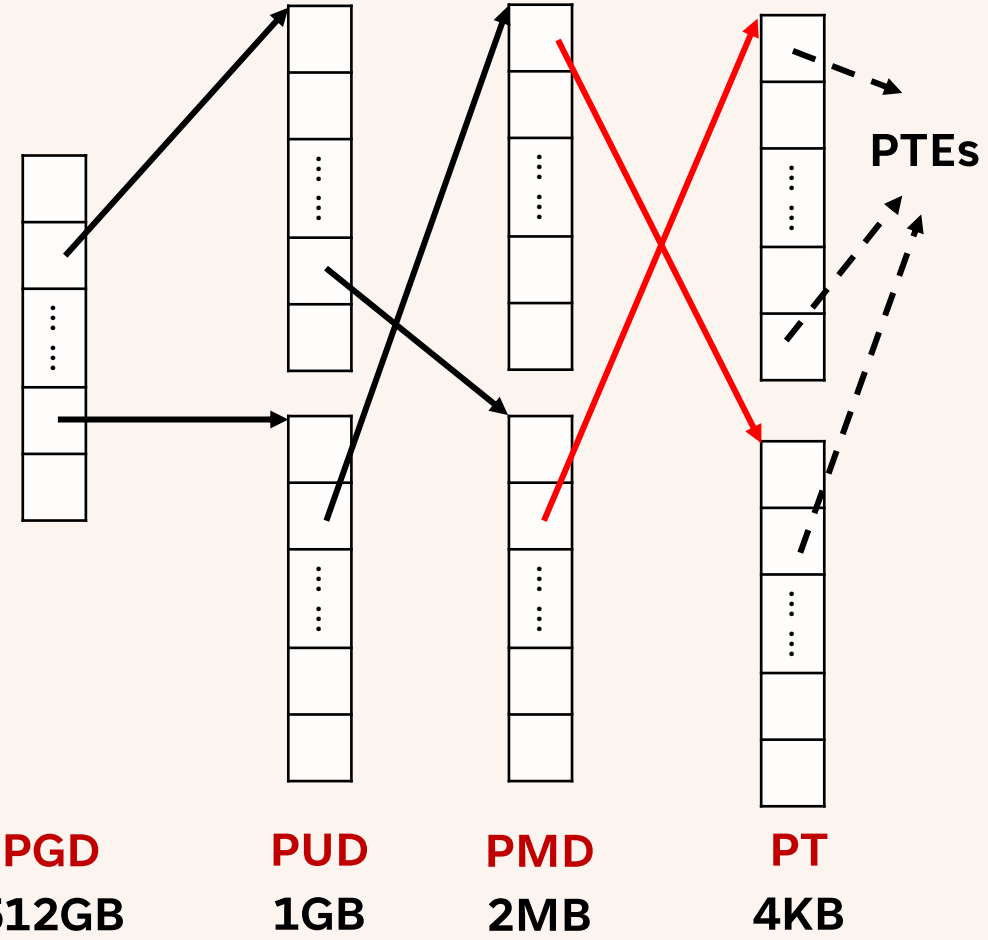
int pswap(void* A, void* B, unsigned int N, int* flag);



- pswap() 128MB:
  - *64* pairs of PMD entries instead of *32k* PT entries
- Perfect only when Aligned
  - ctFS partitions are strictly size Aligned

pswap()'s Key to Efficiency

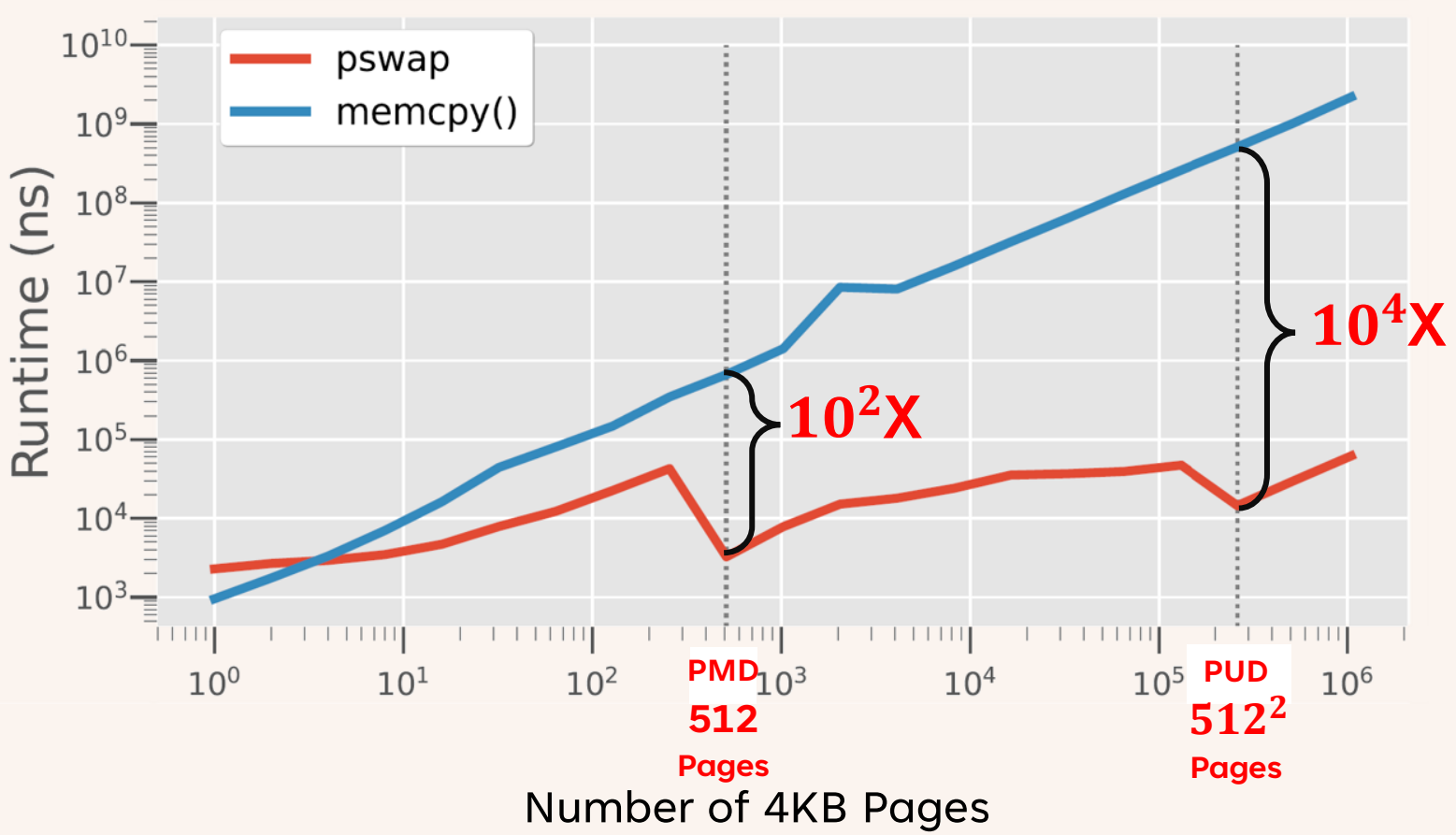
int pswap(void* A, void* B, unsigned int N, int* flag);



- pswap() 128MB:
  - *64* pairs of PMD entries instead of *32k* PT entries
- Perfect only when Aligned
  - ctFS partitions are strictly size Aligned

pswap() Performance

int pswap(**void*** A, **void*** B, **unsigned int** N, **int*** flag);

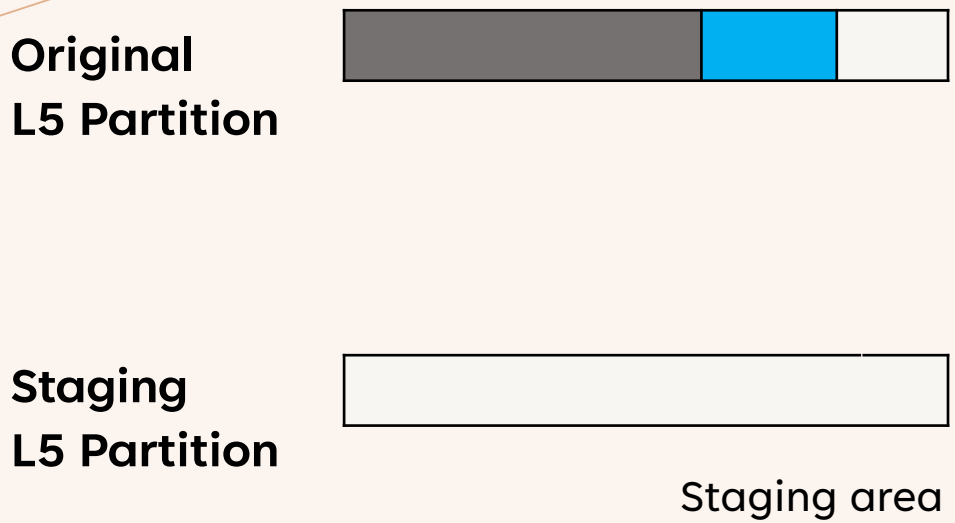


Atomic Write Powered By pswap()

1. Allocate a staging partition
2. Write new data to staging partition
3. Copy the data fragment to the new partition if the write is not page-aligned
4. Call pswap() to atomically swap affected pages

- Used in ctFS's Strict Mode

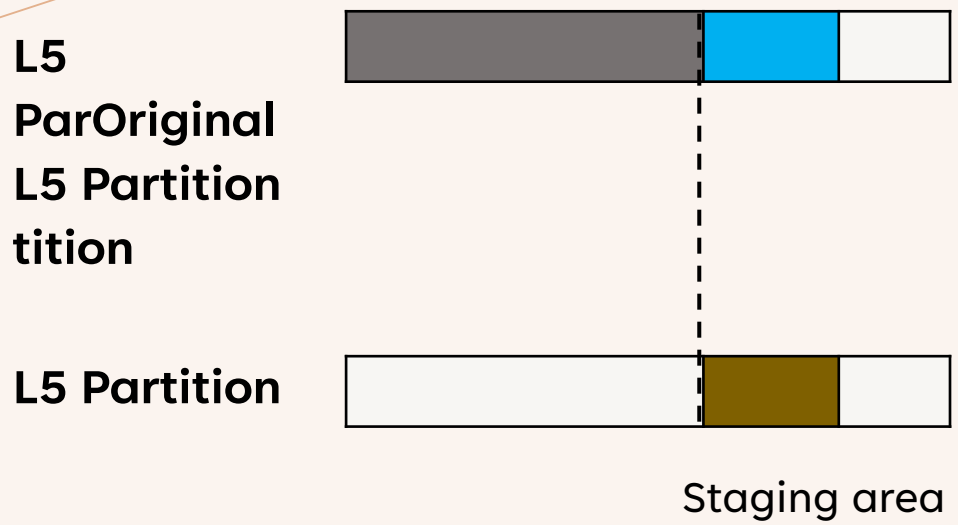
Atomic Write Powered By pswap()



1. **Allocate a staging partition**
2. Write new data to staging partition
3. Copy the data fragment to the new partition if the write is not page-aligned
4. Call pswap() to atomically swap affected pages

- Used in ctFS's Strict Mode

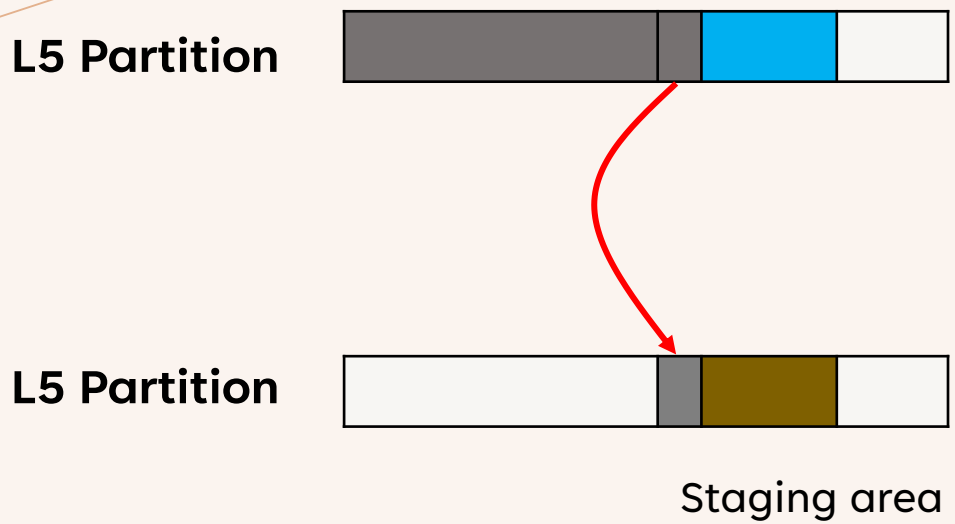
Atomic Write Powered By pswap()



1. Allocate a staging partition
2. **Write new data to staging partition**
3. Copy the data fragment to the new partition if the write is not page-aligned
4. Call pswap() to atomically swap affected pages

- Used in ctFS's Strict Mode

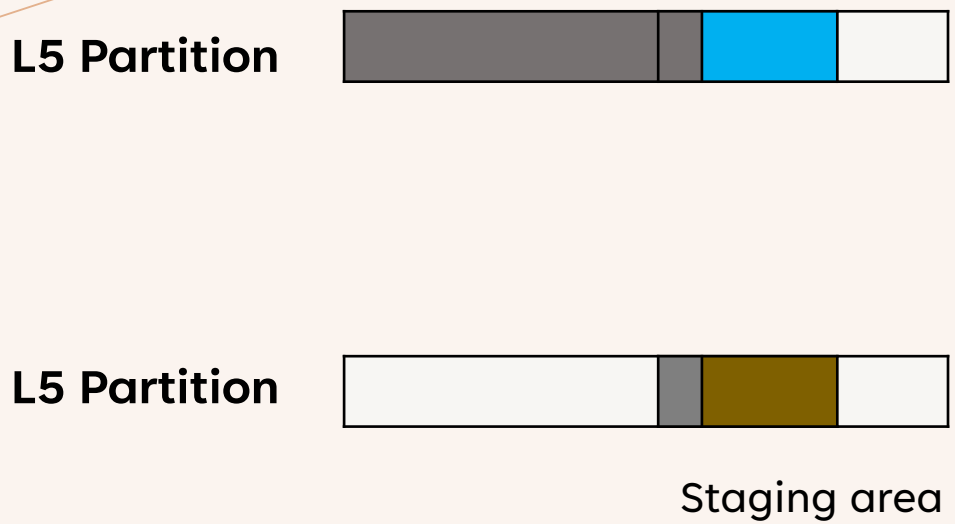
Atomic Write Powered By pswap()



1. Allocate a staging partition
2. Write new data to staging partition
3. **Copy the data fragment to the new partition if the write is not page-aligned**
4. Call pswap() to atomically swap affected pages

- Used in ctFS's Strict Mode

Atomic Write Powered By pswap()



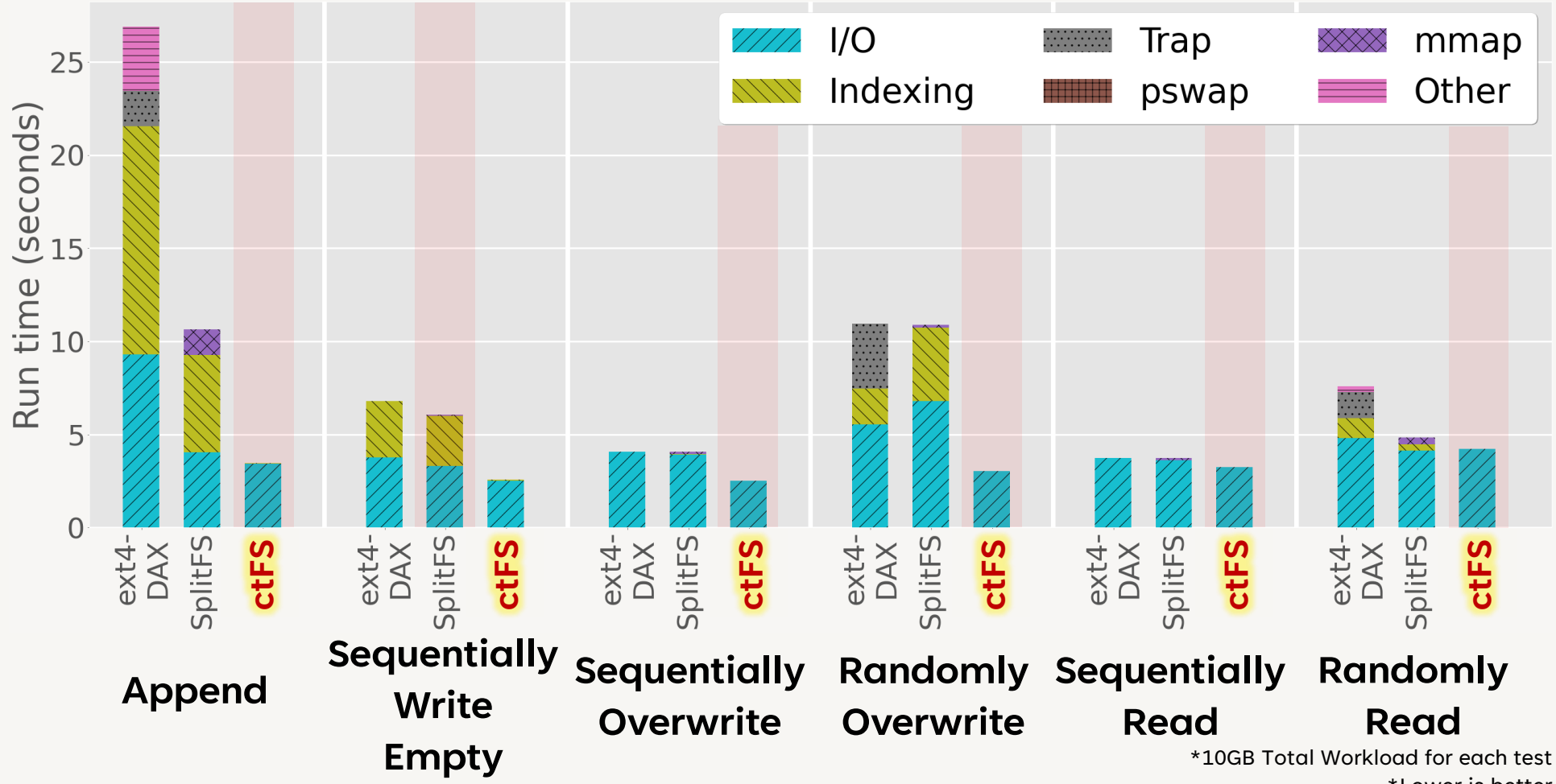
1. Allocate a staging partition
2. Write new data to staging partition
3. Copy the data fragment to the new partition if the write is not page-aligned
4. **Call pswap() to atomically swap affected pages**

- Used in ctFS's Strict Mode

Evaluation

- Micro Benchmarks
- Application Evaluation
- All tests are conducted in the following setup:
  - CPU: 8-core Intel Xeon 4215 CPU @2.5 GHz
  - DRAM: 96GB DDR4 RDIMM
  - PM: 2 X 128GB Intel Optane NVDIMM
  - OS: Ubuntu 20.04 LTS
  - Kernel: Linux Version v5.7.0-rc7+

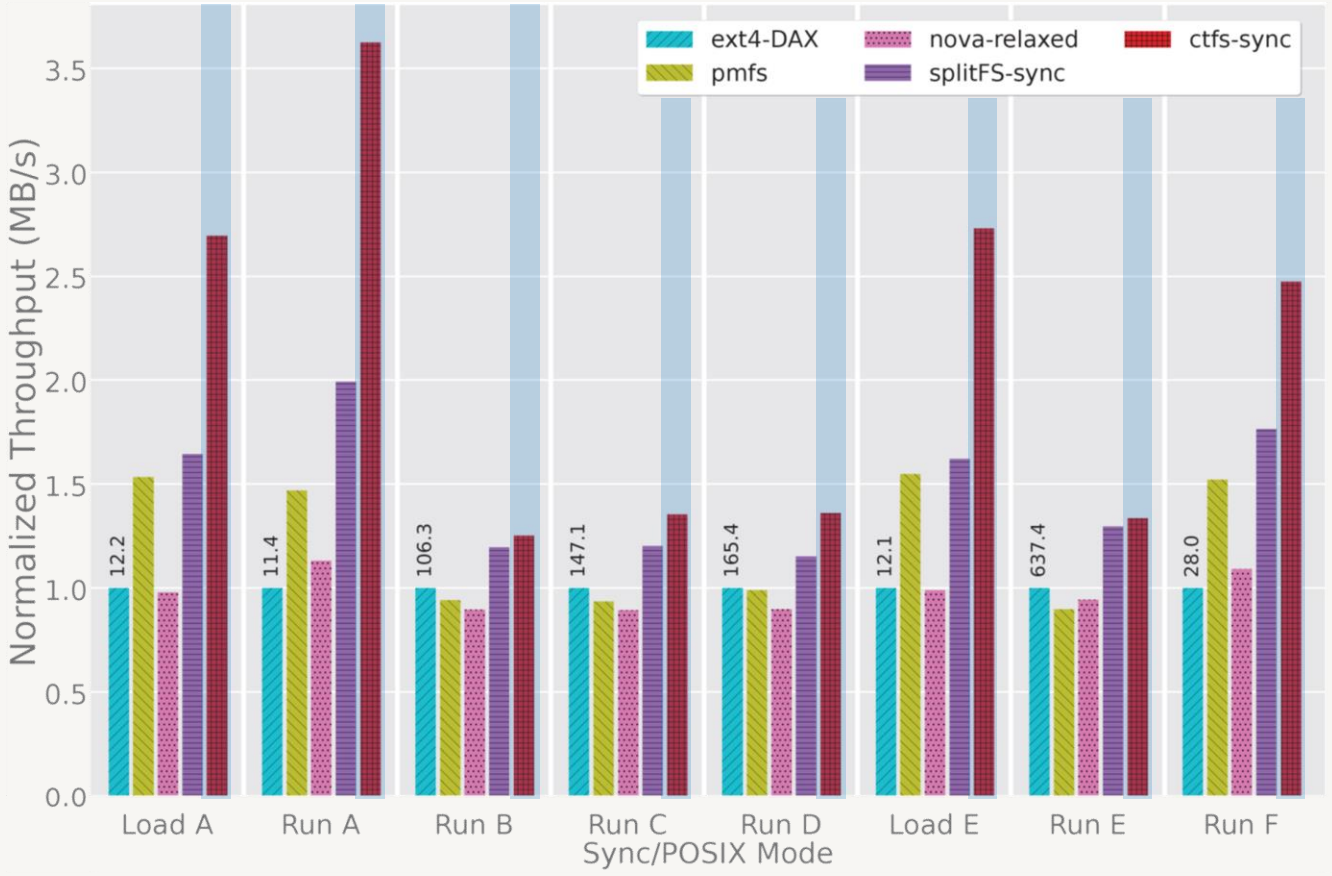
Microbenchmarks: Indexing Cost **Eliminated**

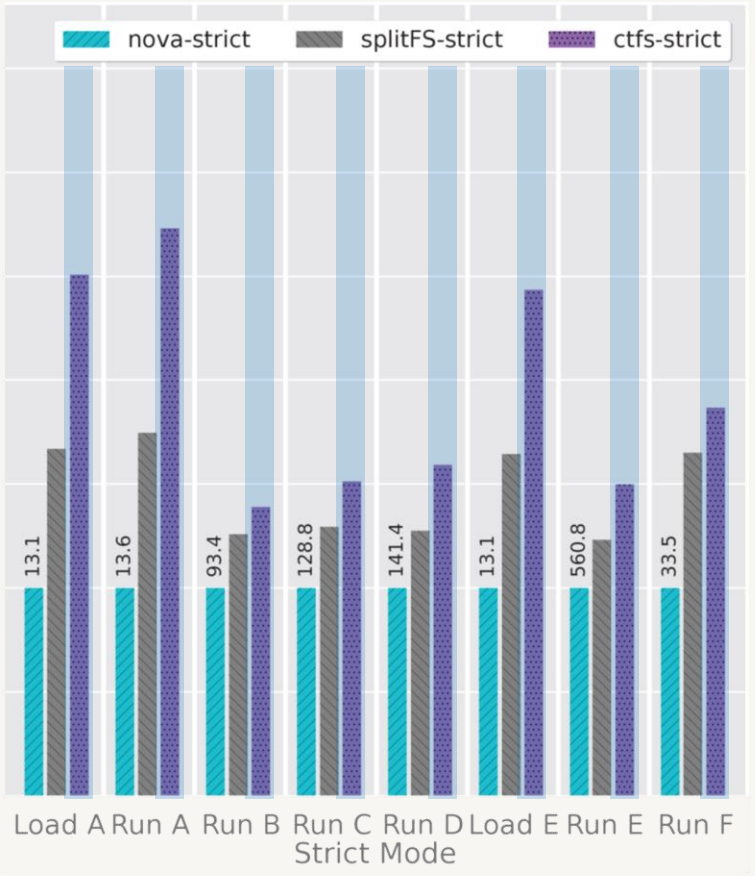


*10GB Total Workload for each test

*Lower is better

LevelDB on YCSB:





*Higher is better

Protection Issue

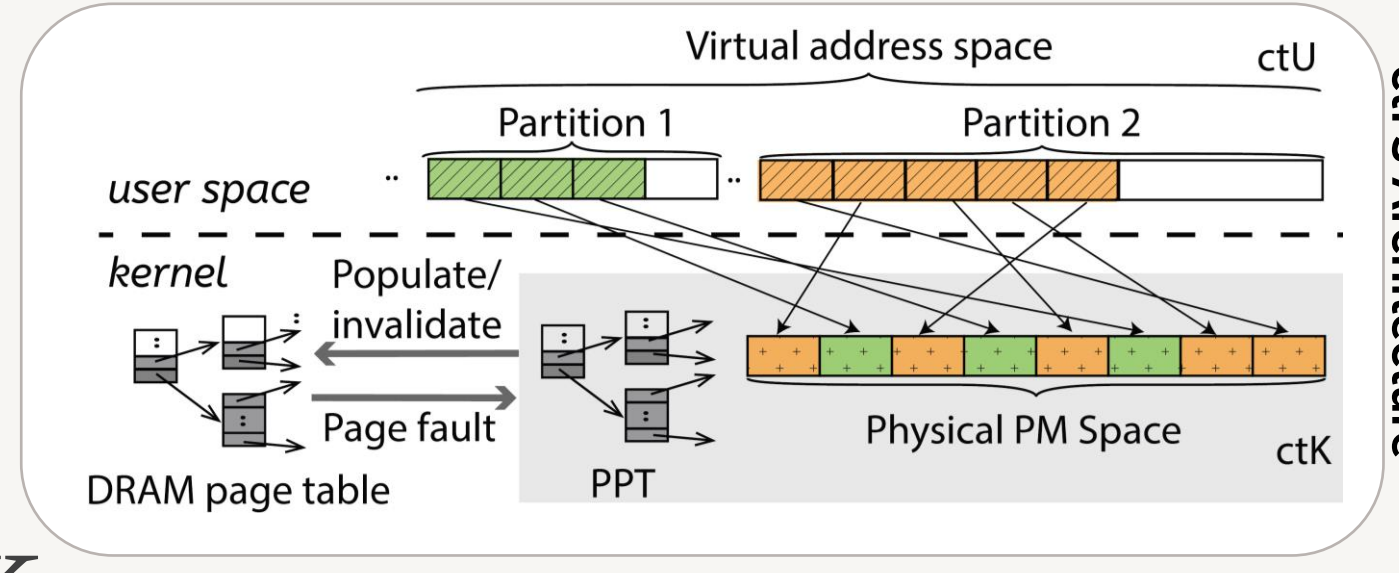
- Weak Protection
  - Due to user-level meta-data handling
  - MPK used to prevent corruption from memory bugs
  - May be vulnerable to intentional attacks
- If it’s a concern
  - One ctFS instance for each application
  - Migration to the kernel in progress

Paper

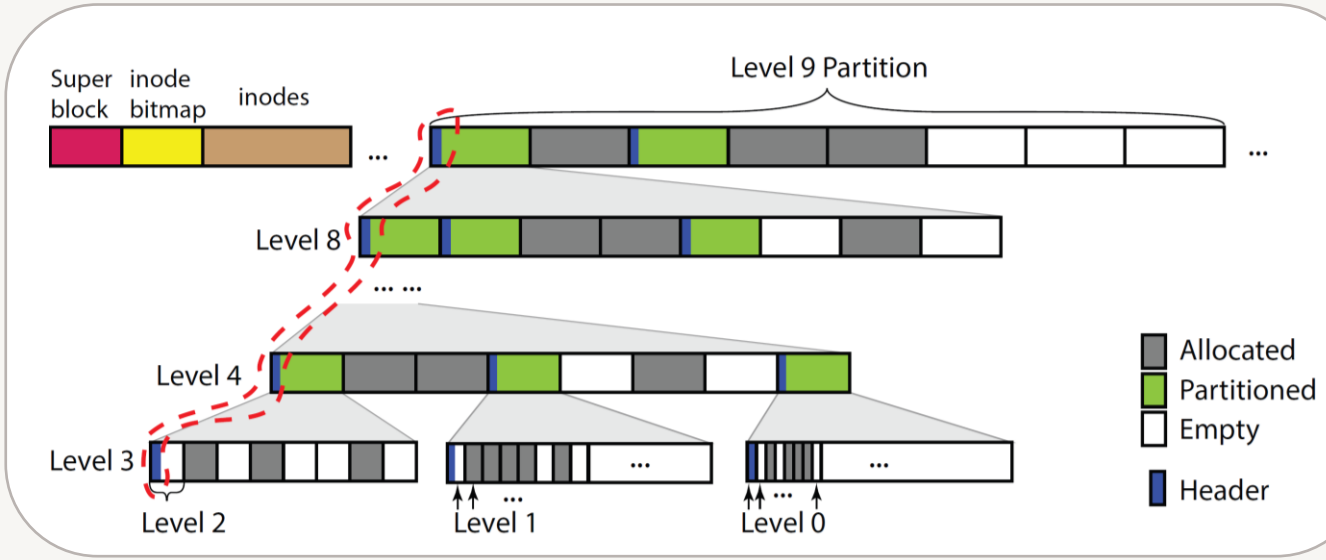
THANK YOU :)

GitHub

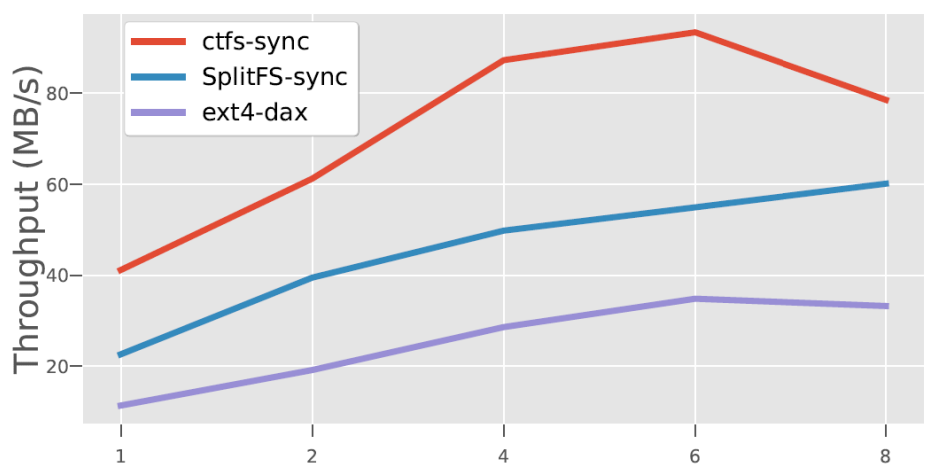
ctFS Architecture



Hierarchical Allocator

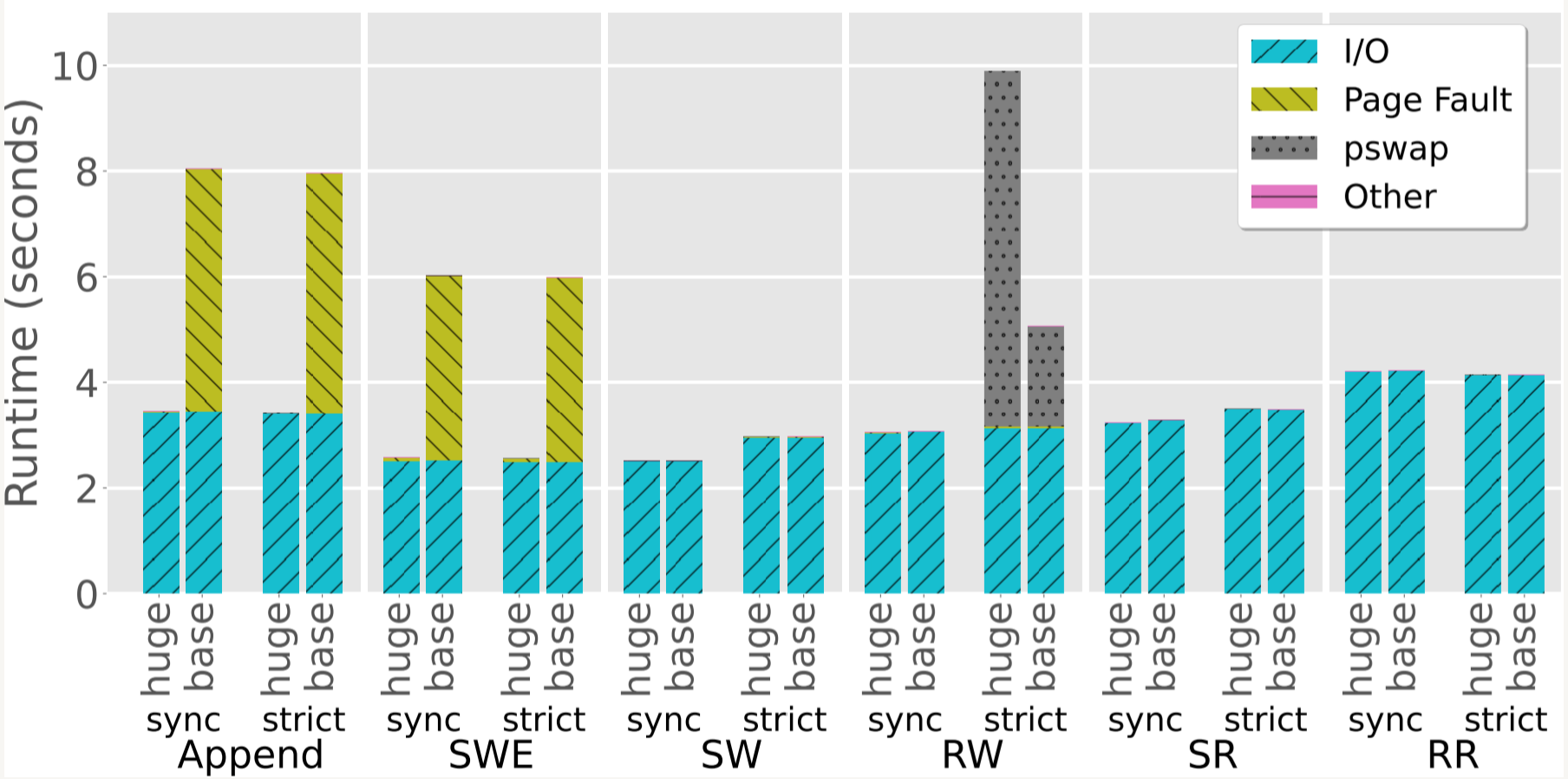


Scalability & Resource Usage



	SplitFS		Ext4-Dax	ctFS
	sync	strict		
Bootstrap (µs)	1.4×10^6	1.1×10^6	0	5
open (create) (µs)	40	40	15	2
open (existing) (µs)	4	10	4	2
unlink (µs)	32	43	31	1.6
DRAM usage (MB)	198	572	N/A	0.516
Space available after format (GB)	230.69		230.69	248.05
Space consumed after YCSB LoadA (MB)	5486.3		5336.7	5378.1

ctFS runtime breakdown & huge page impact



Internal Fragmentation

- Manageable
  - Max waste: < 7x of file size
  - Physical space is not wasted
- Virtual Memory space is HUGE
  - Currently in use: 2^{48}B (256 TB)
  - Now supporting: 2^{57}B (128 EB)