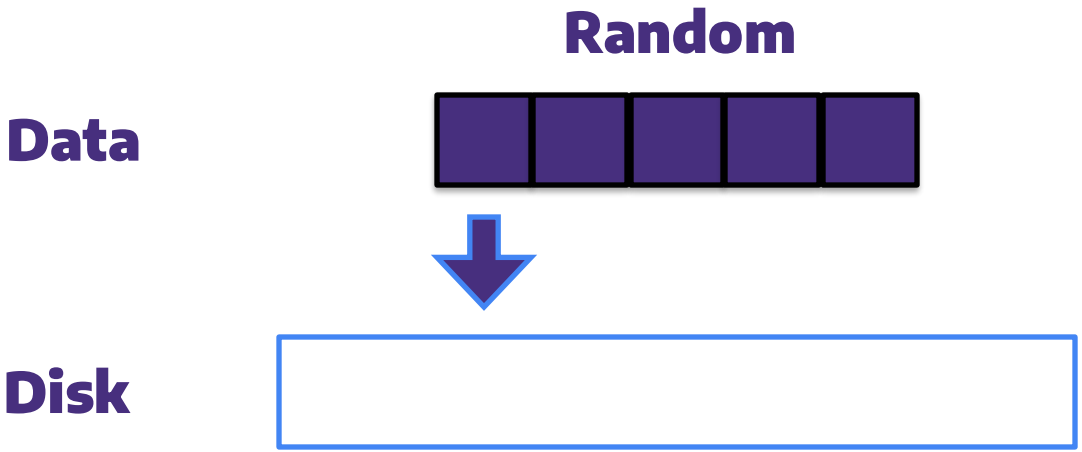
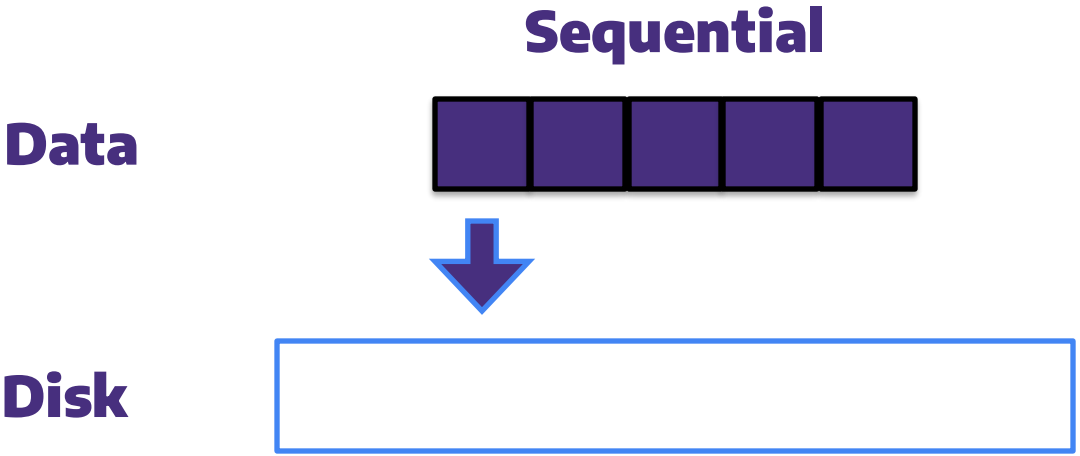
Log structured file system

Random vs sequential writes

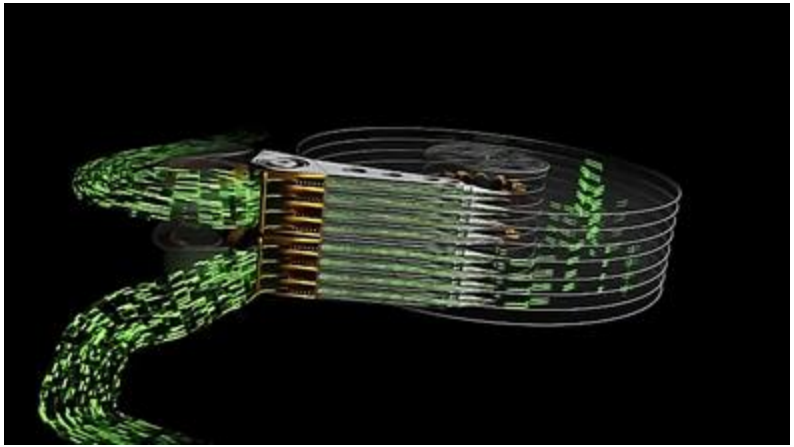


Random vs sequential writes



Disk latencies

- 3 sources:
  - Rotational latency
  - Seek time
  - Data transfer time



(Emory College of Arts and Sciences, CS377)

Rotational latency

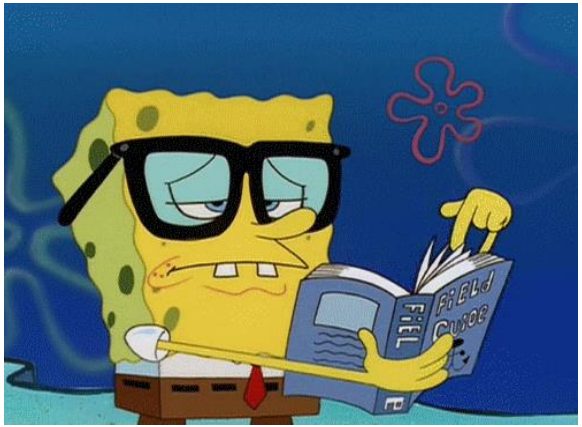
- Time taken for disk to spin to required location
- 4 – 6 ms

Seek time

- Time taken for read/write head to reach required location
- ~9 ms

Data transfer time

- Time taken to read/write data
- Transfer rate: ~100 – 300 Mbps
- ~0.3 ms for 4 KB of data



Challenges

LFS mainly deals with two challenges:

1. Finding data
2. Maintaining free space

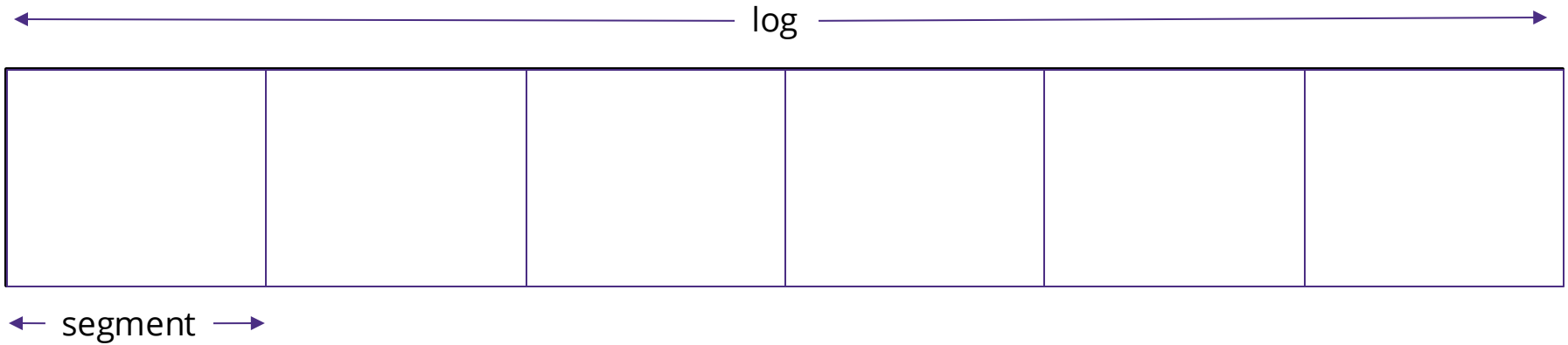
Reading/finding data

Data structure	Purpose	Location	Section
Inode	Locates blocks of file, holds protection bits, modify time, etc.	Log	3.1
Inode map	Locates position of inode in log, holds time of last access plus version number.	Log	3.1
Indirect block	Locates blocks of large files.	Log	3.1
Segment summary	Identifies contents of segment (file number and offset for each block).	Log	3.2
Segment usage table	Counts live bytes still left in segments, stores last write time for data in segments.	Log	3.6
Superblock	Holds static configuration information such as number of segments and segment size.	Fixed	None
Checkpoint region	Locates blocks of inode map and segment usage table, identifies last checkpoint in log.	Fixed	4.1
Directory change log	Records directory operations to maintain consistency of reference counts in inodes.	Log	4.2

Table 1 — Summary of the major data structures stored on disk by Sprite LFS.
For each data structure the table indicates the purpose served by the data structure in Sprite LFS. The table also indicates whether the data structure is stored in the log or at a fixed position on disk and where in the paper the data structure is discussed in detail. Inodes, indirect blocks, and superblocks are similar to the Unix FFS data structures with the same names. Note that Sprite LFS contains neither a bitmap nor a free list.

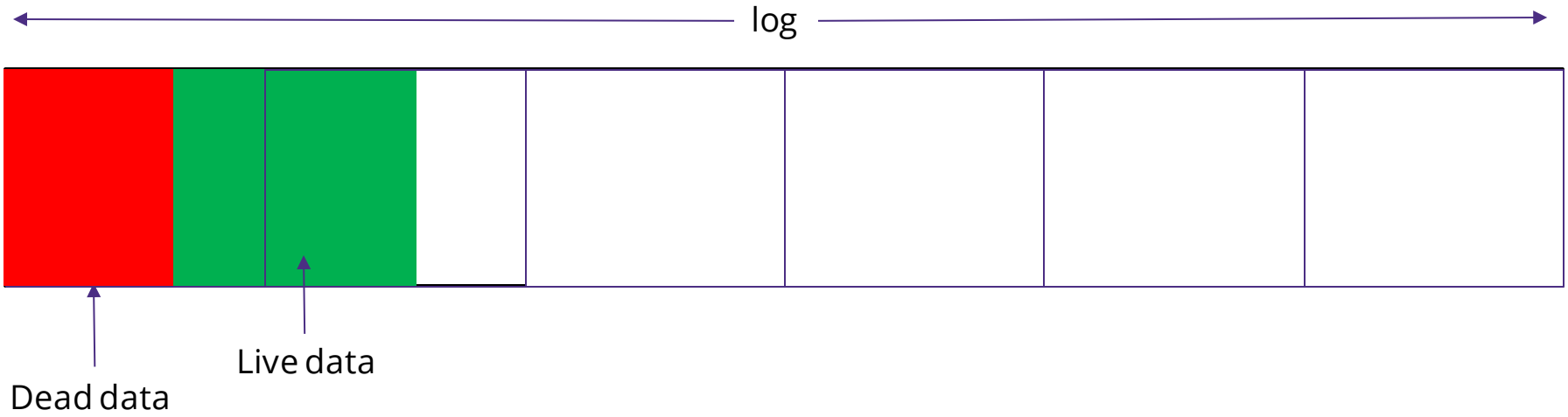
Segment cleaning

Use a combination of threading and copying



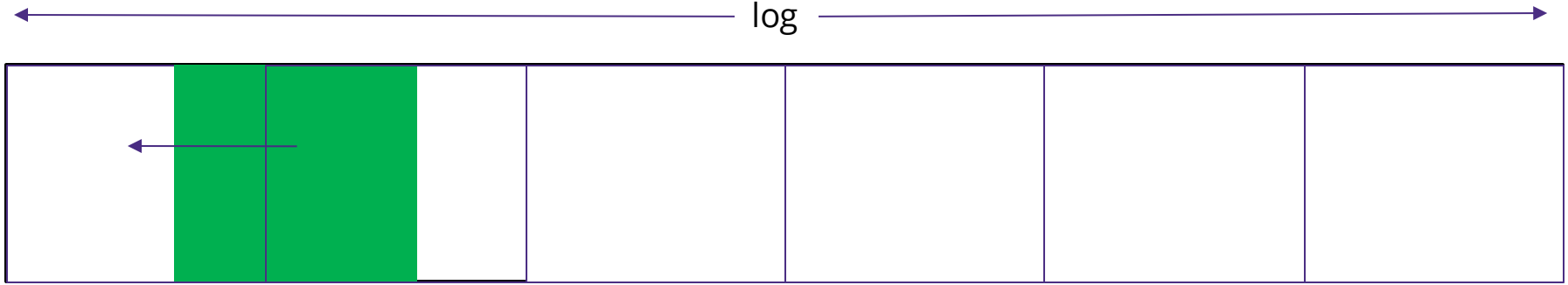
Segment cleaning

Use a combination of threading and copying



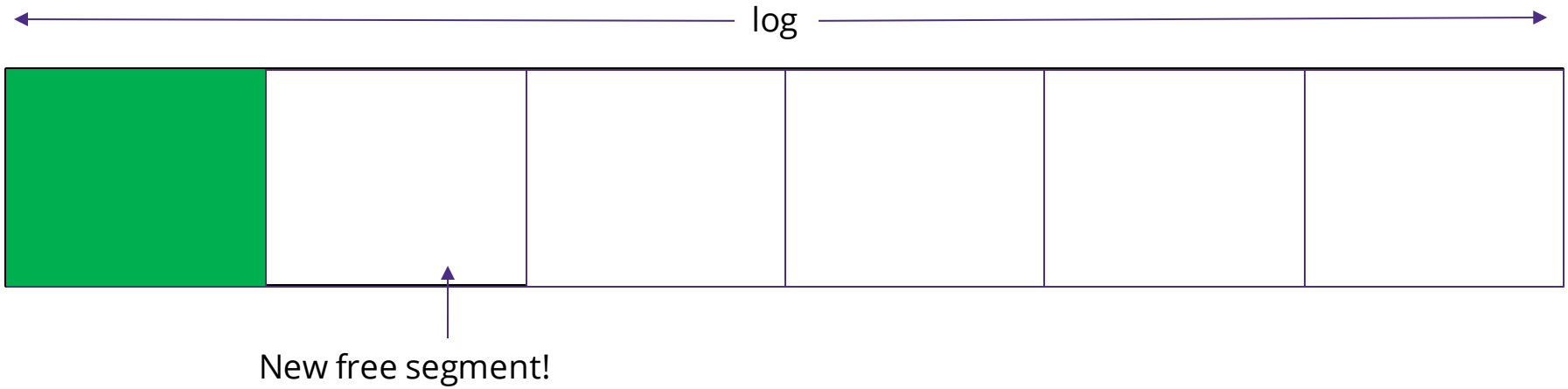
Segment cleaning

Use a combination of threading and copying



Segment cleaning

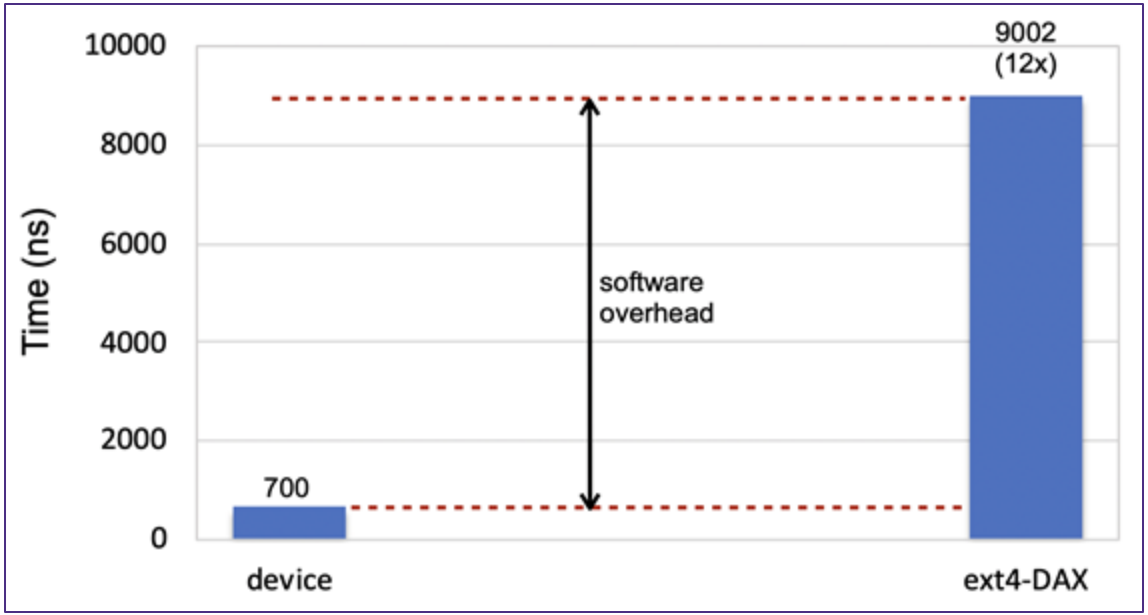
Use a combination of threading and copying



Discussion

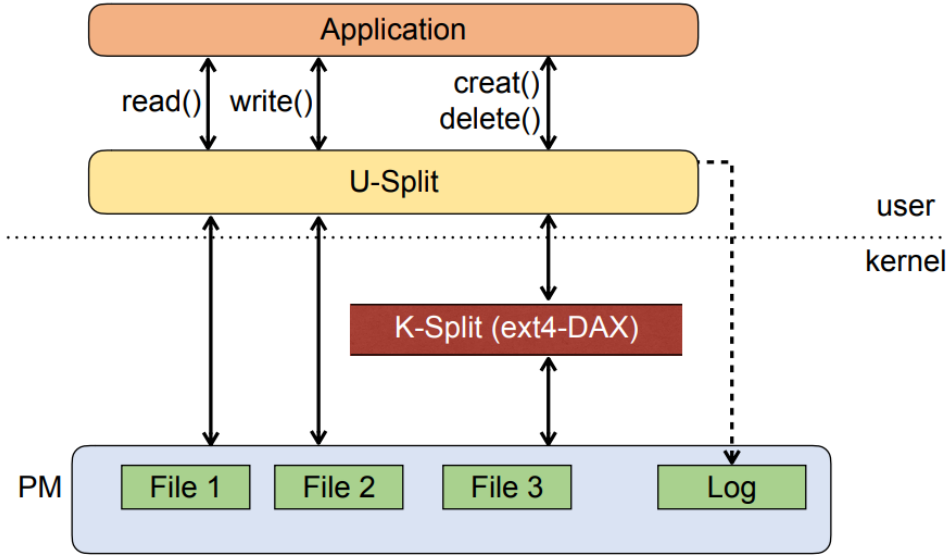
1. How would you design a FS for different workloads (eg: HPC, databases, ML)?
2. How would LFS perform for these workloads?
3. Is LFS still useful today? If so, where?
4. What would you do with SSDs/Flash or NVM? Are the bottlenecks the same as disks? (Hint: No)

Overhead of NVM file systems



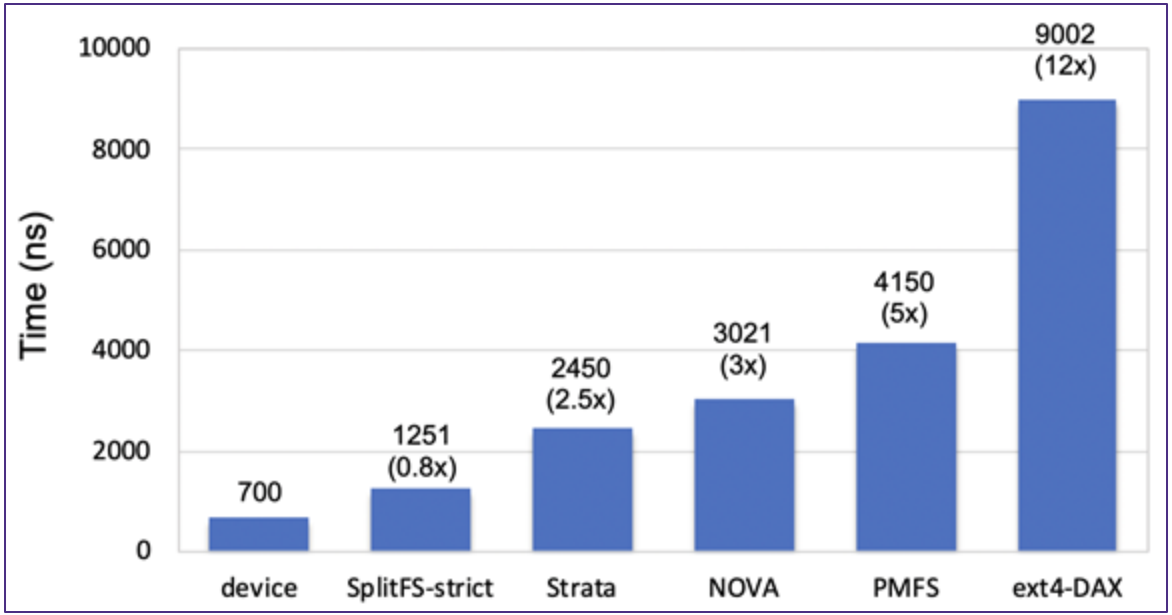
(Kadekodi et al. [SOSP '19])

How SplitFS reduces overheads



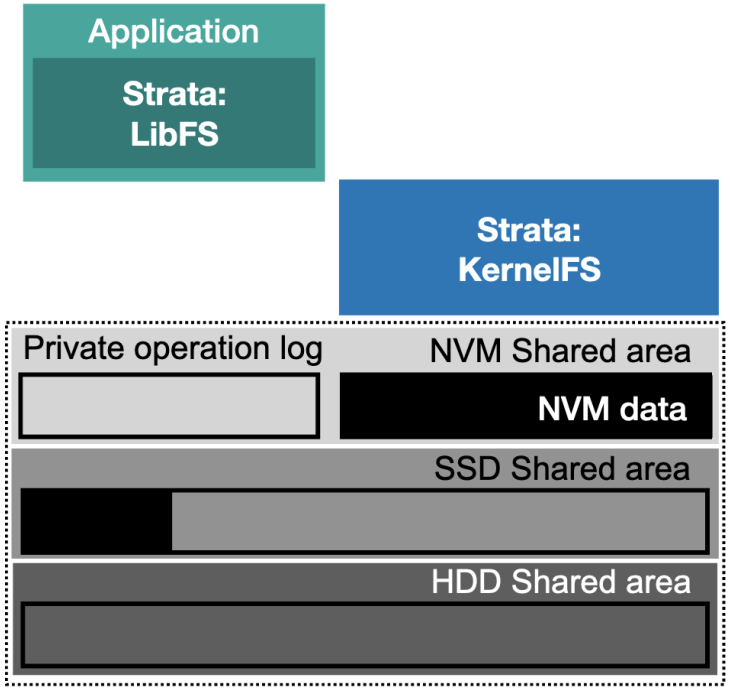
(Kadekodi et al. [SOSP '19])

Comparison of NVM file systems



(Kadekodi et al. [SOSP '19])

Strata


Application
Strata:
LibFS
Strata:
KernelFS
Private operation log
NVM Shared area
NVM data
SSD Shared area
HDD Shared area

Discussion

1. How would you design a FS that spans multiple technologies?
2. How would you split responsibilities between a distributed FS and a local FS?
3. What guarantees would you give up in return for performance when it comes to accessing storage?