CS 744: SPLIT-FS

Shivaram Venkataraman

Fall 2021

ADMINISTRIVIA

- Course Project: Check in: Today! → Instructions on Piazza
- Midterm 2 next week!

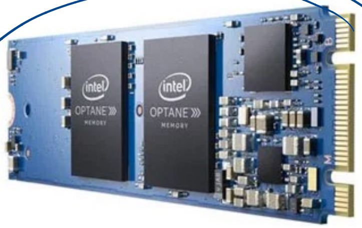
→ From SQL to TPU paper

1 page → Canvas

Serverless Computing

Compute Accelerators

Infiniband Networks

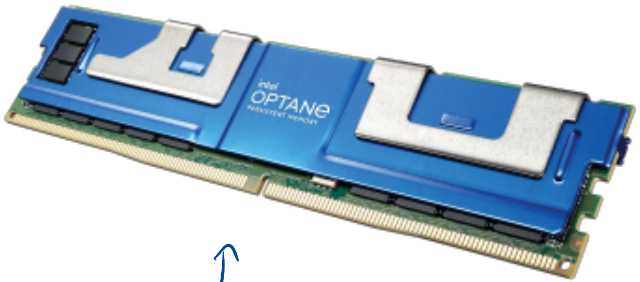


Non-Volatile Memory

PERSISTENT MEMORY

L1 / L2 Cache — DRAM — PMEM — SSDs — Tape Drives

Prior benchmarking study



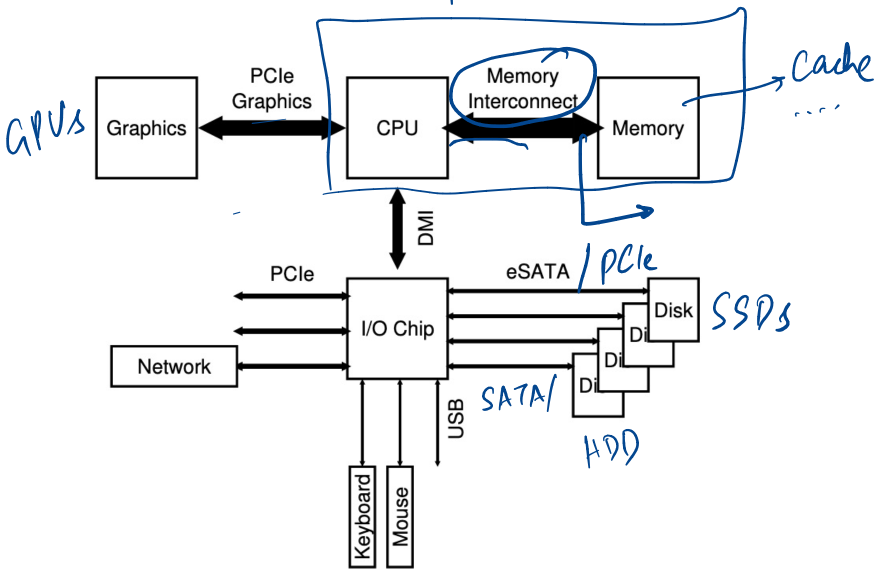
Intel Optane PM

Property	SSD	DRAM	Intel PM
Sequential read latency (ns)	~1000	81	169 (2.08×)
Random read latency (ns)		81	305 (3.76×)
Store + flush + fence (ns)		86	91 (1.05×)
Read bandwidth (GB/s)		120	39.4 (0.33×)
Write bandwidth (GB/s)		80	13.9 (0.17×)

much higher than PCIe bw (SSDs)

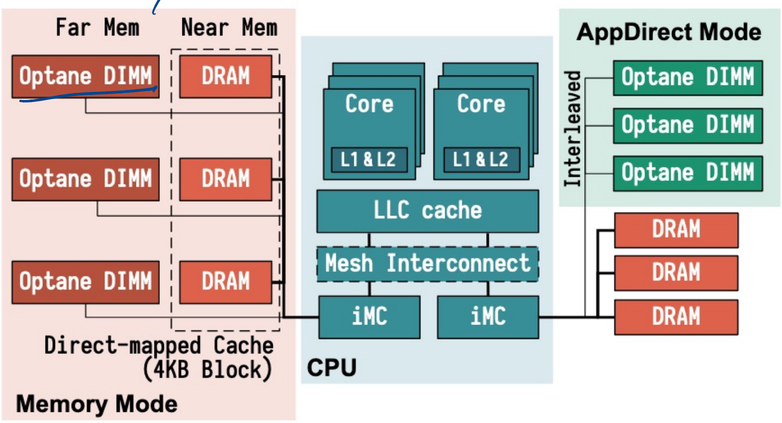
WHAT IS DIFFERENT?

Traditional / Existing



OSTEP → Andrea / Remzi

memory mode



(a) Optane Platform Modes (Memory and AppDirect)

Optane DIMMs are on memory / DRAM slots

BACKGROUND: FILE SYSTEM API

POSIX API for files

```
int fd = open(char *path, int flag, mode_t mode)
read(int fd, void *buf, size_t nbyte)
write(int fd, void *buf, size_t nbyte)
close(int fd)

rename(char *old, char *new)

fsync(int fd)
```

open a file, read a FD

rename → move a file to a diff location

fsync → flush any state that has been buffered

MOTIVATION: OVERHEADS

DAX: mmap pages into virtual memory.

No page caches!

Total time taken ≈ 9 μs

Overhead ≈ 8 μs ⇒ Actual append ≈ 1 μs

File system	Append Time (*ns*)	Overhead (*ns*)	Overhead (%)
ext4 DAX	9002	8331	1241%
PMFS	4150	3479	518%
NOVA-Strict	3021	2350	350%
SplitFS-Strict	1251	580	86%
SplitFS-POSIX	1160	488	73%

These overheads are ok for slower storage devices!

SPLIT-FS: GOALS

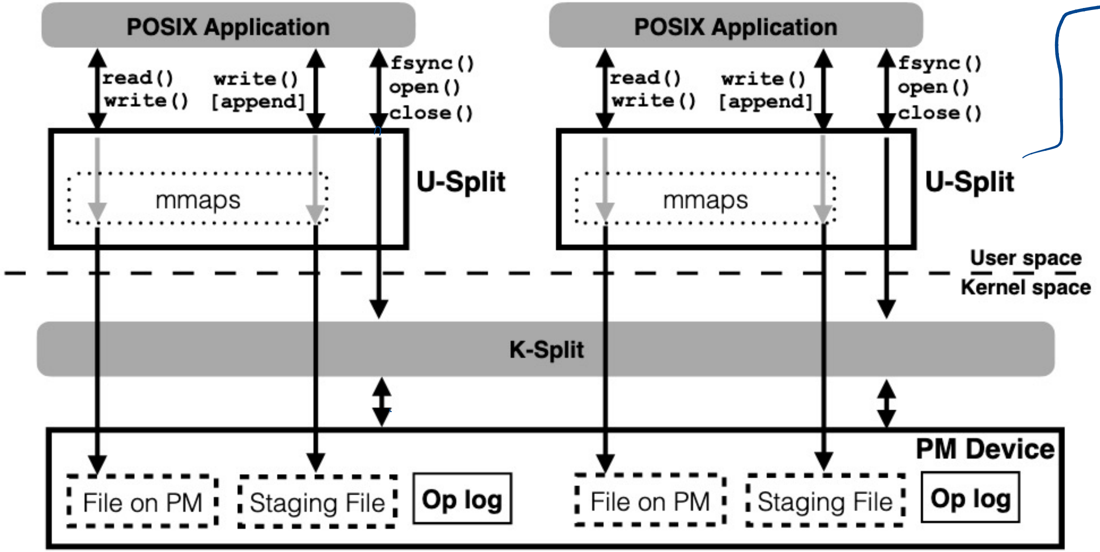
Low software overhead → compared to time spent on device

Transparency → Compatible with existing applications

Minimal data-copy/IO

Flexible semantics

SPLIT FS DESIGN: READ/WRITES

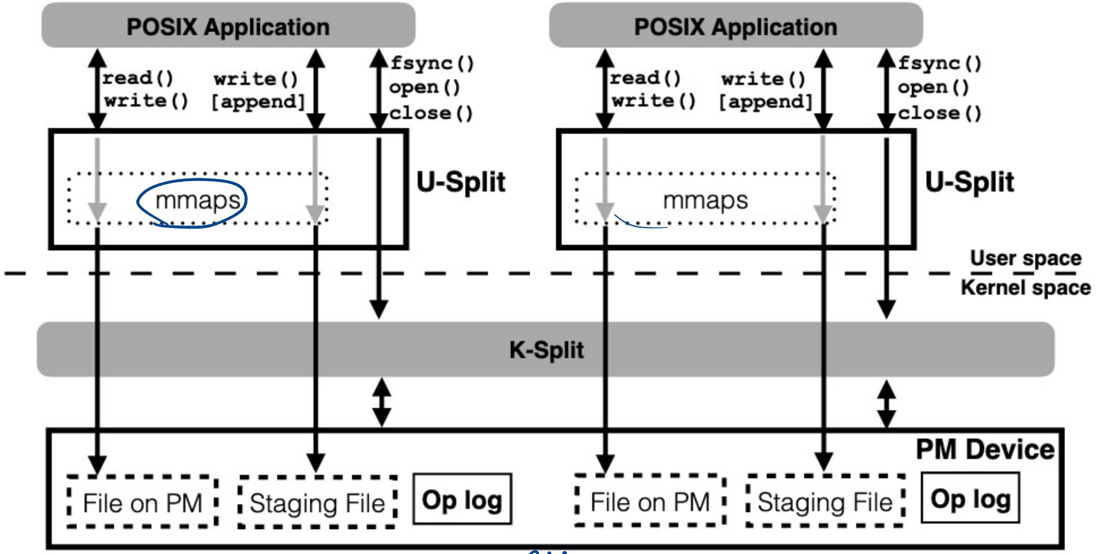


FS is split into part in userspace and part in kernelspace

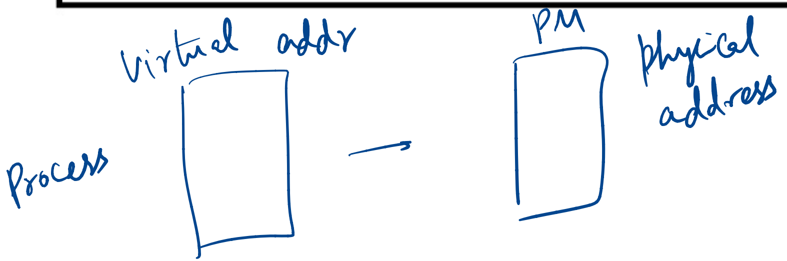
don't pay system call overheads

Data operations will be in USplit
Metadata operations will be in K-split

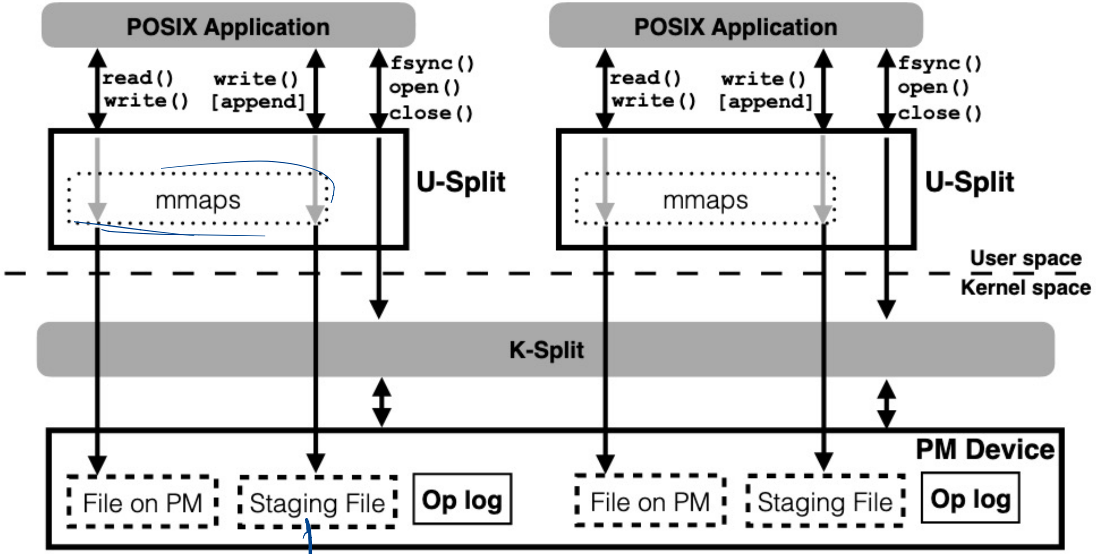
SPLIT FS DESIGN: READ/WRITES



- File is already open
- mmaps relevant region of the file
- reads happen using LOAD instructions which go to PM
- write do a STORE or MOV instructions
- "non-temporal" store



SPLIT FS DESIGN: APPEND



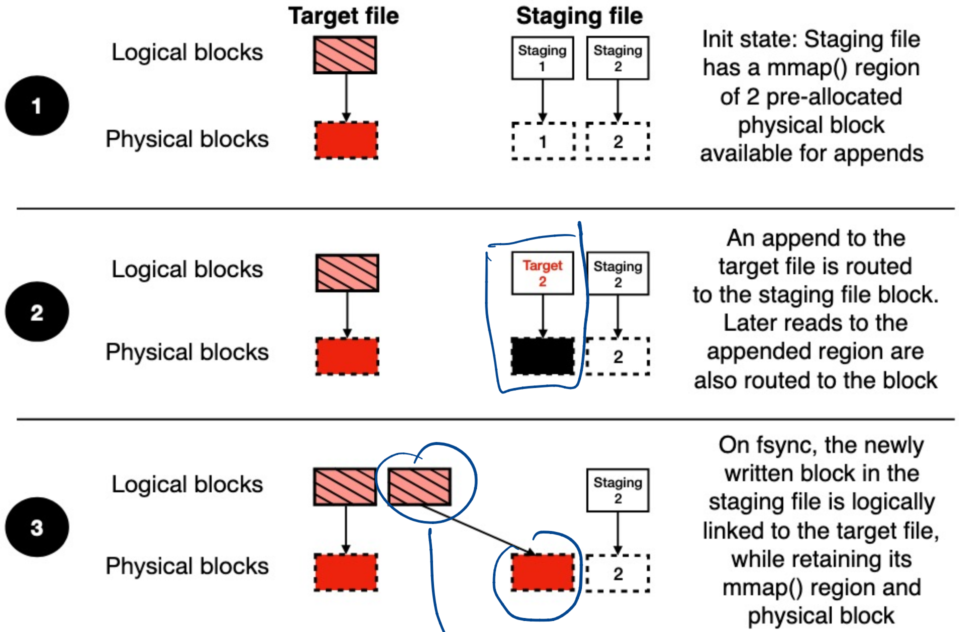
pre-allocated at start up and replenished when needed

Appends are forwarded to a staging file

fsync() → relinked to the original file

mmaps also tracks staging files used for appends

RELINK



Avoid copying data from staging file to the original file

create new logical block

Metadata
↳ Directory hierarchy
↳ sizes, ACL etc.

SPLIT-FS MODES

Sync → call returns changes are visible

Atomic → all changes happen simultaneously

Mode	*Sync. Data Ops*	*Atomic Data Ops*	*Sync. Metadata Ops*	*Atomic Metadata Ops*	*Equivalent to*
POSIX	✗	✗	✗	✓	ext4-DAX
sync	✓	✗	✓	✓	Nova-Relaxed, PMFS
strict	✓	✓	✓	✓	NOVA-Strict, Strata

append (10), append (4) fsync (fd) → Atomicity 10 and 4

SPLIT-FS: LOGGING

Atomicity →
↳ overwrites

strict

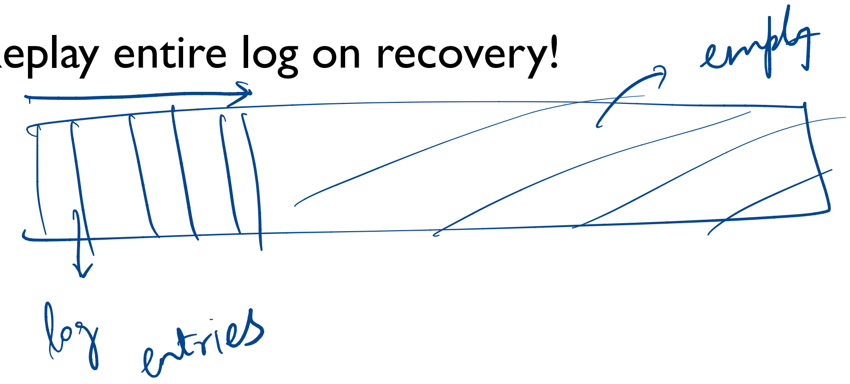
Logical redo logging

Log entry: 64B in size! 4B checksum! → valid log entry

sfence to ensure ordering

Fixed length log: 128 MB per-application

Replay entire log on recovery!



Logical | Physical logging
↳ write down what operation needs to be done in the log

Redo | Undo logging
↳ the operations are idempotent

SUMMARY

Persistent Memory: New opportunities, new challenges

~~Split-FS: split Pipelining to use CPU, GPU~~

~~Partition buffer, BETA ordering~~

Split-FS: Partition FS between User | Kernel

$\downarrow$ Data

$\downarrow$ Metadata

- Support for efficient appends, atomic ops using staging files

DISCUSSION

https://forms.gle/8TwGgqXhVyuiRCpx8

appends are faster because of pre-allocation

unlink is expensive in SplitFS → mmap entries need to be removed.

System call	Strict	Sync	POSIX	ext4 DAX
open	2.09	2.08	1.82	1.54
close	0.78	0.69	0.69	0.34
append	3.14	3.09	2.84	11.05
fsync	6.85	6.80	6.80	28.98
read	4.57	4.53	4.53	5.04
unlink	14.60	13.56	14.33	8.60

Table 6: SplitFS system call overheads. The table compares the latency (in us) of different system calls for various modes of SplitFS and ext4 DAX.

fsync is more expensive in ext4-DAX

↳ SplitFS

↳ no copy relink

Data operations are faster

Metadata operations are slower

OK! because data operations are frequent

In what ways can SplitFS improve performance of Big Data frameworks like MR/Spark?

→ Faster read / writes to input / output files

⇒ Better performance

→ Appends being optimized is good for MR / Spark

→ Persistence

↳ helps fault tolerance and reduce recomputation on failure

→ Capacity can also helps increase the RDDs in "cache"

NEXT STEPS

Next class: TPU

Project check-ins tonight!

DISCUSSION

Staging files in DRAM?

Page faults are expensive on open()