SDC

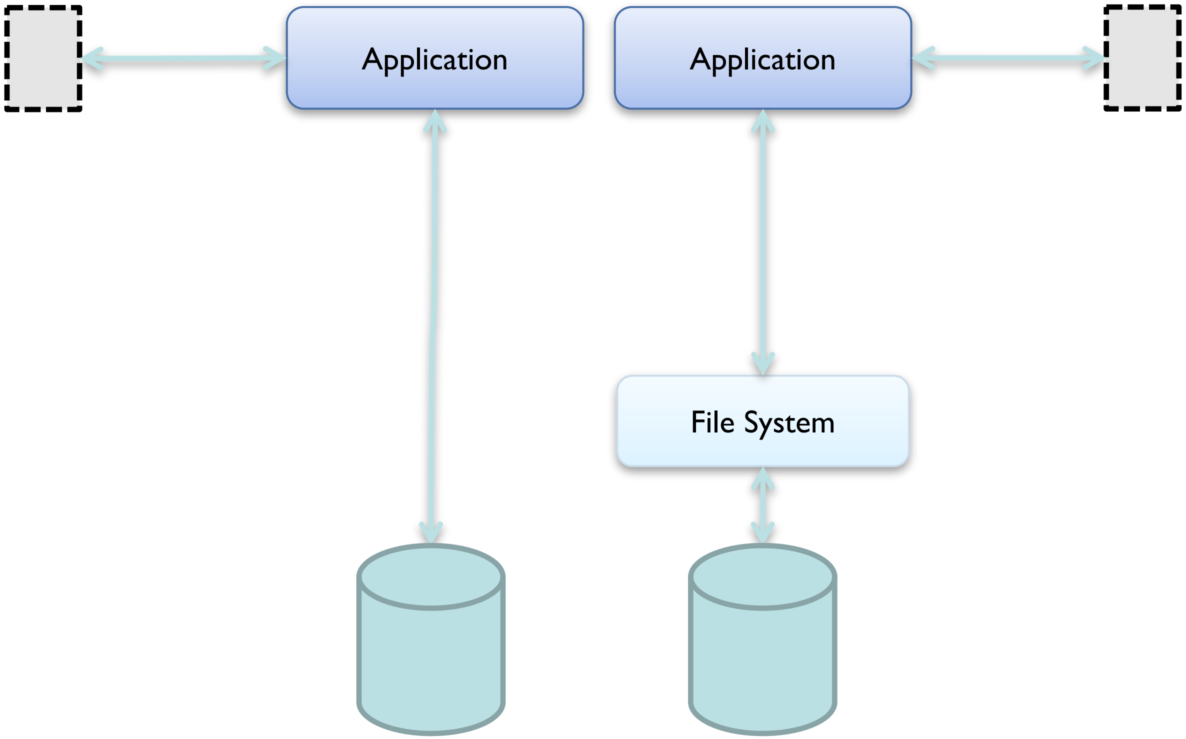
STORAGE DEVELOPER CONFERENCE

SNIA ■ SANTA CLARA, 2016

Breaking Barriers: Making Adoption of Persistent Memory Easier

Andy Rudoff
Intel Corporation

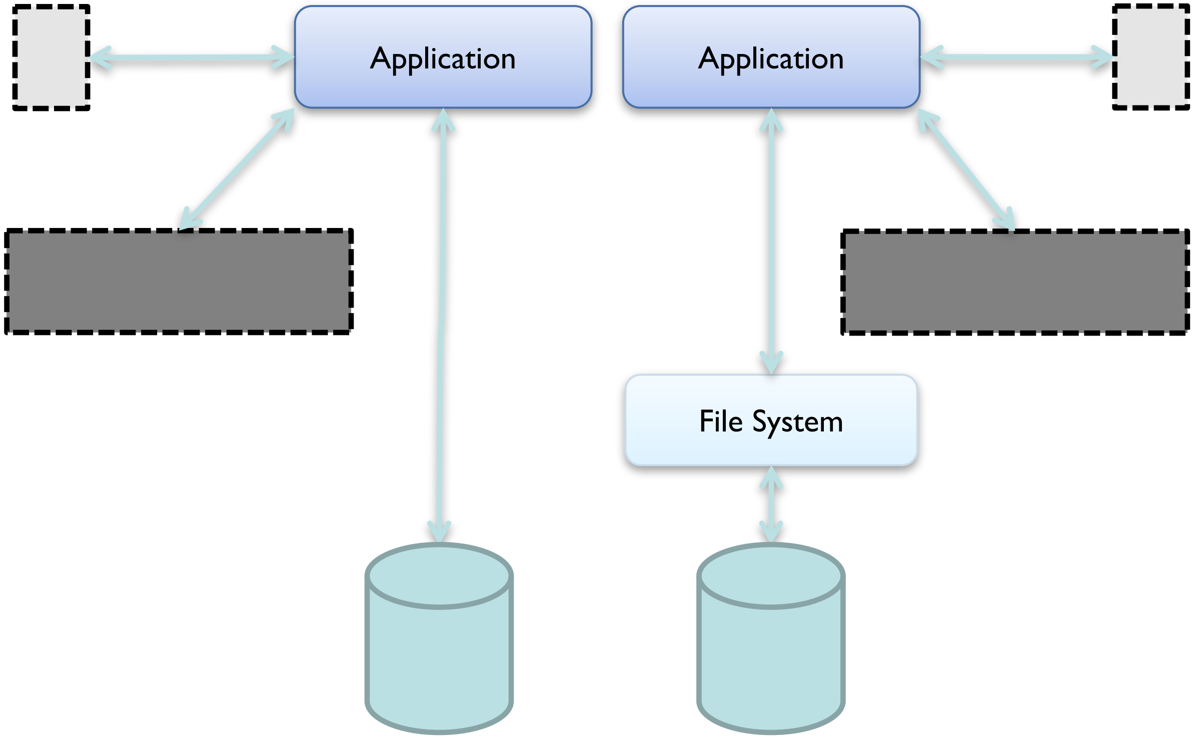
The Past:Two Primary Tiers for Run-Time Data



Memory

Storage

Moving to Three Tiers

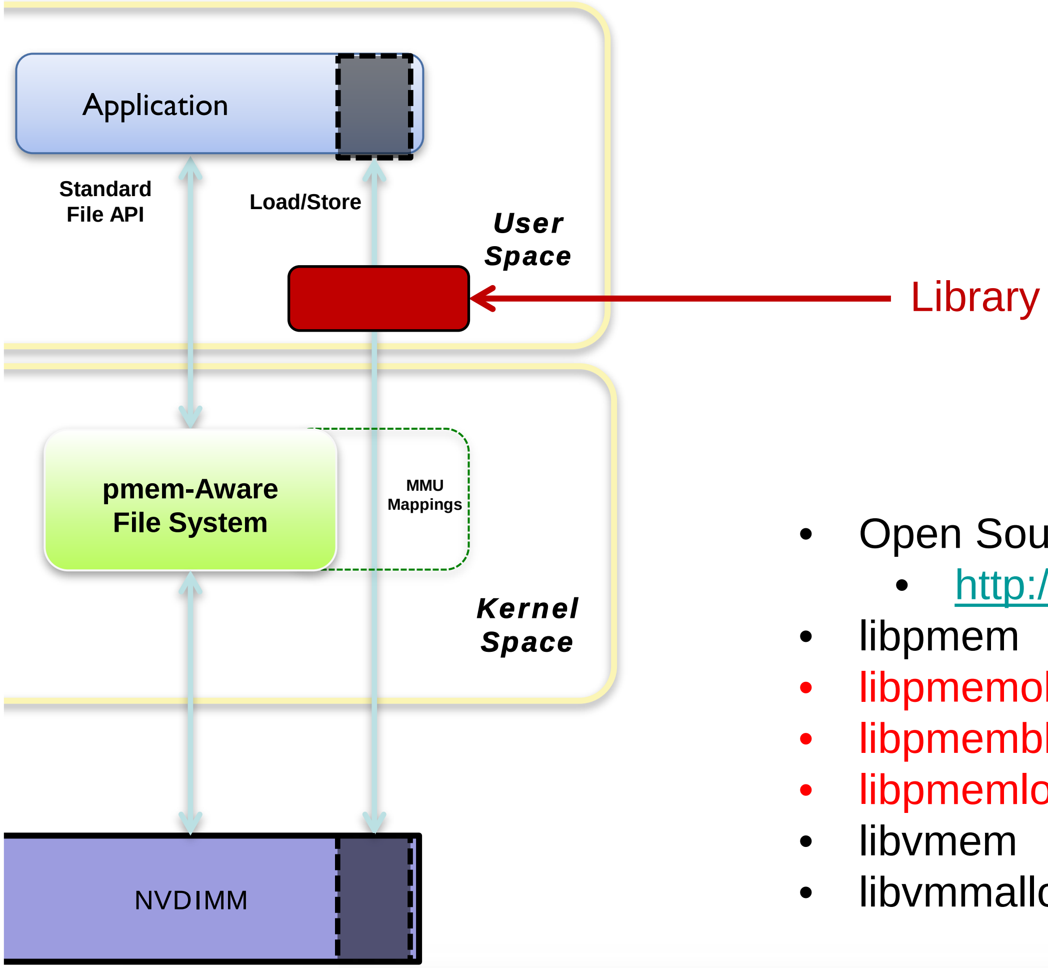


Memory

Persistent Memory

Storage

Modifying Applications for pmem…



- Open Source
  - http://pmem.io
- libpmem
- libpmemobj
- libpmemblk
- libpmemlog
- libvmem
- libvmmalloc

Transactional

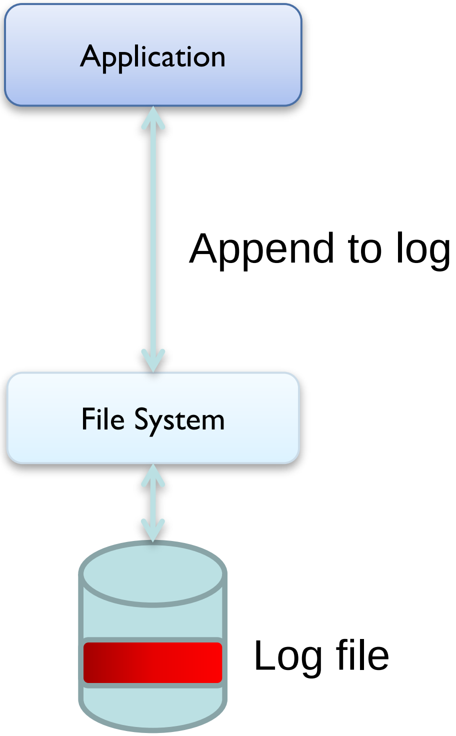
Reasons to Re-architect an Application

- Large data set
  - Terabytes
- Persistent
- Byte addressable
  - Especially random, small accesses
  - Storage must convert all accesses to blocks
- DMA target
- Performance critical

Reasons NOT to Re-architect

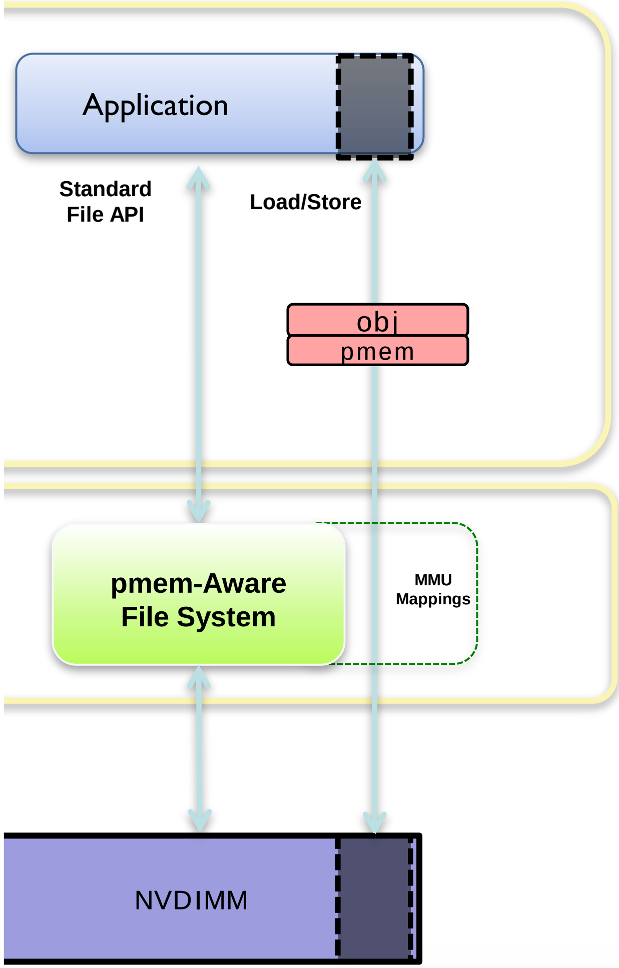
- One of the transparent ways to use pmem works well enough
  - Supplementing memory (paging)
  - Block mode driver
  - Some middleware using it transparently
- When cost outweighs benefit
  - Architecture, design, implementation
  - Validation

Example: A Good Candidate for pmem



- Database-like application
- Transactional updates to tables
  - (Tables might be in-memory)
- Write-Ahead-Logging
  - Written, never read
    - (Except after crash)
  - Appending to log file
  - Path includes FS

Example: Non-transparent Solution



- ❒ Application uses libpmemobj API
- ❒ Log appends become transactions to pmem
- ❒ Much faster, but…
- ❒ App had to change

Learning a new API

```
fd =
open(LOGFILE, …);
…
write(fd, buf, len);
…
fsync(fd);
```

```
pop =
pmemobj_open(FILE, …);
…
TX_BEGIN(pop) {
        …
        TX_MEMCPY(…);
        …
} TX_END
```

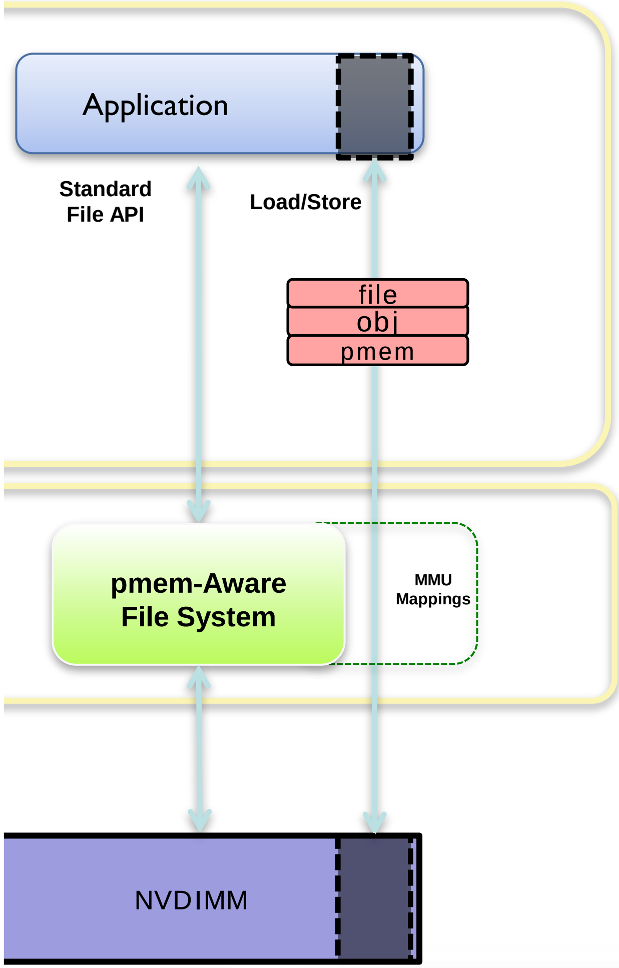
Learning an Easier API

```
fd =
open(LOGFILE, …);
…
write(fd, buf, len);
…
fsync(fd);
```

→

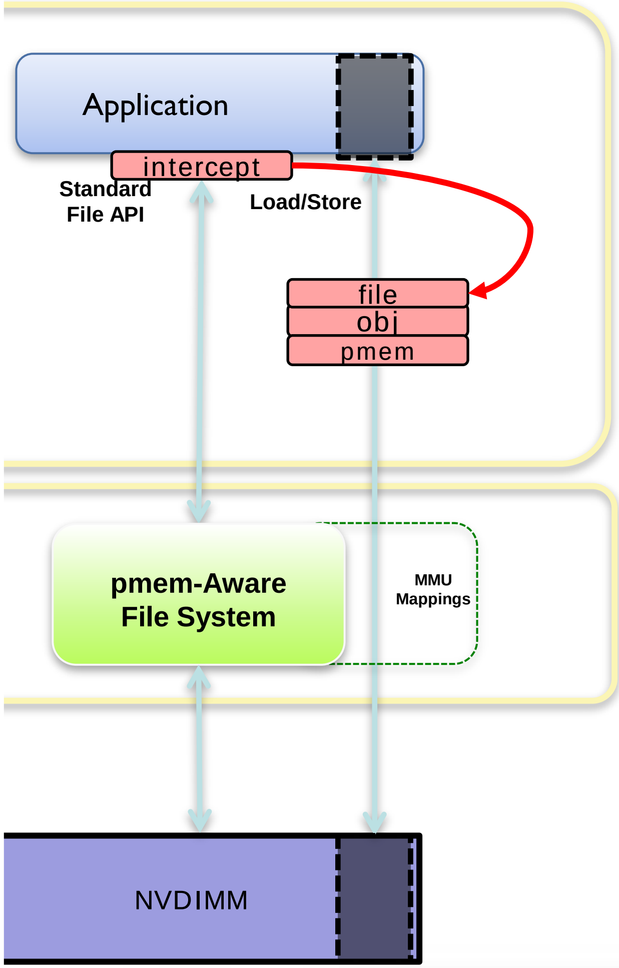
```
fd =
pmemfile_open(LOGFILE, …);
…
pmemfile_write(fd, buf, len);
…
/* fsync(fd); */
```

libpmemfile



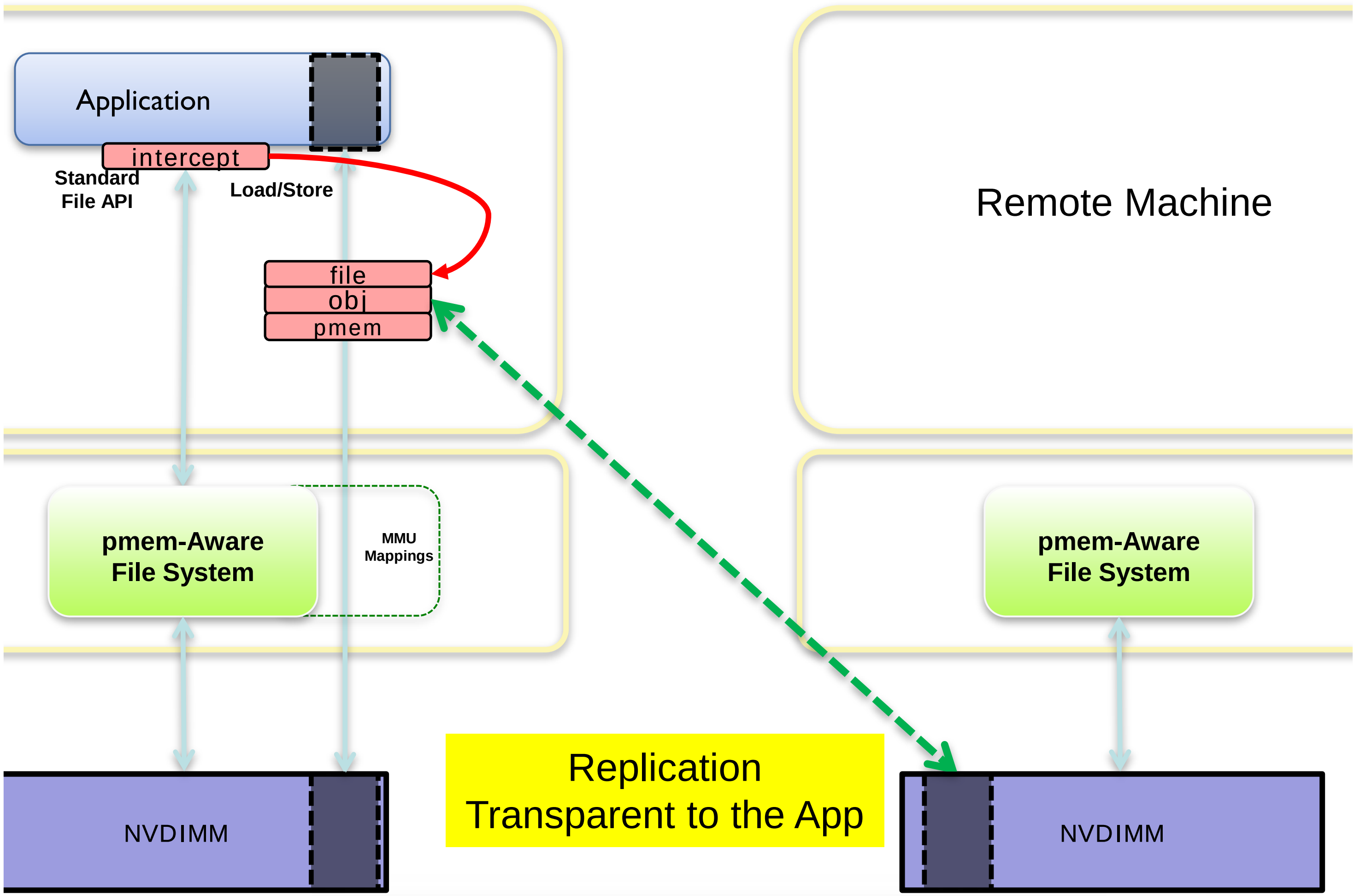
- libpmemfile
  - Modeled after POSIX
- Familiar API
- App had to change

Using libpmemfile transparently



- Linker magic
  - Loads libpmemfile
  - Helps with intercept
- Admin configures which files live on pmem
- App binary unchanged

Built on libpmemobj, So We Inherit…



Replication
Transparent to the App

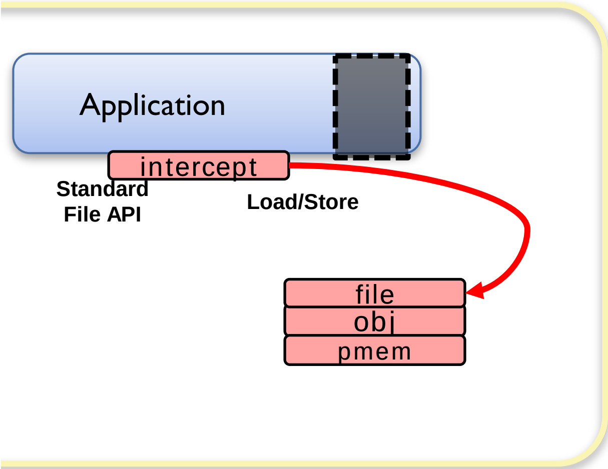
What Operations "Just Work?"

- Basic file I/O syscalls
  - `open` / `close` / `read` / `write` …
- libc functions that build on basic file I/O
  - `fopen` / `fprintf` / `opendir` / `readdir` …
- App sees normal files, directories, etc.
  - But sometimes they live in a pmem pool

What Operations Are Problematic?

- `fork` (with no `exec`)
  - *might* not work as expected
- `select` on files
  - Who does this?
- `mmap`
  - Just use pmem-aware FS for this
- `aio`
- Some rare syscalls
- Multi-process access (multi-thread ok)
  - Also a limitation of libpmemobj
  - Still looking for requirements on this
- Key is how to report when something doesn't work

Implementing the Interception Logic



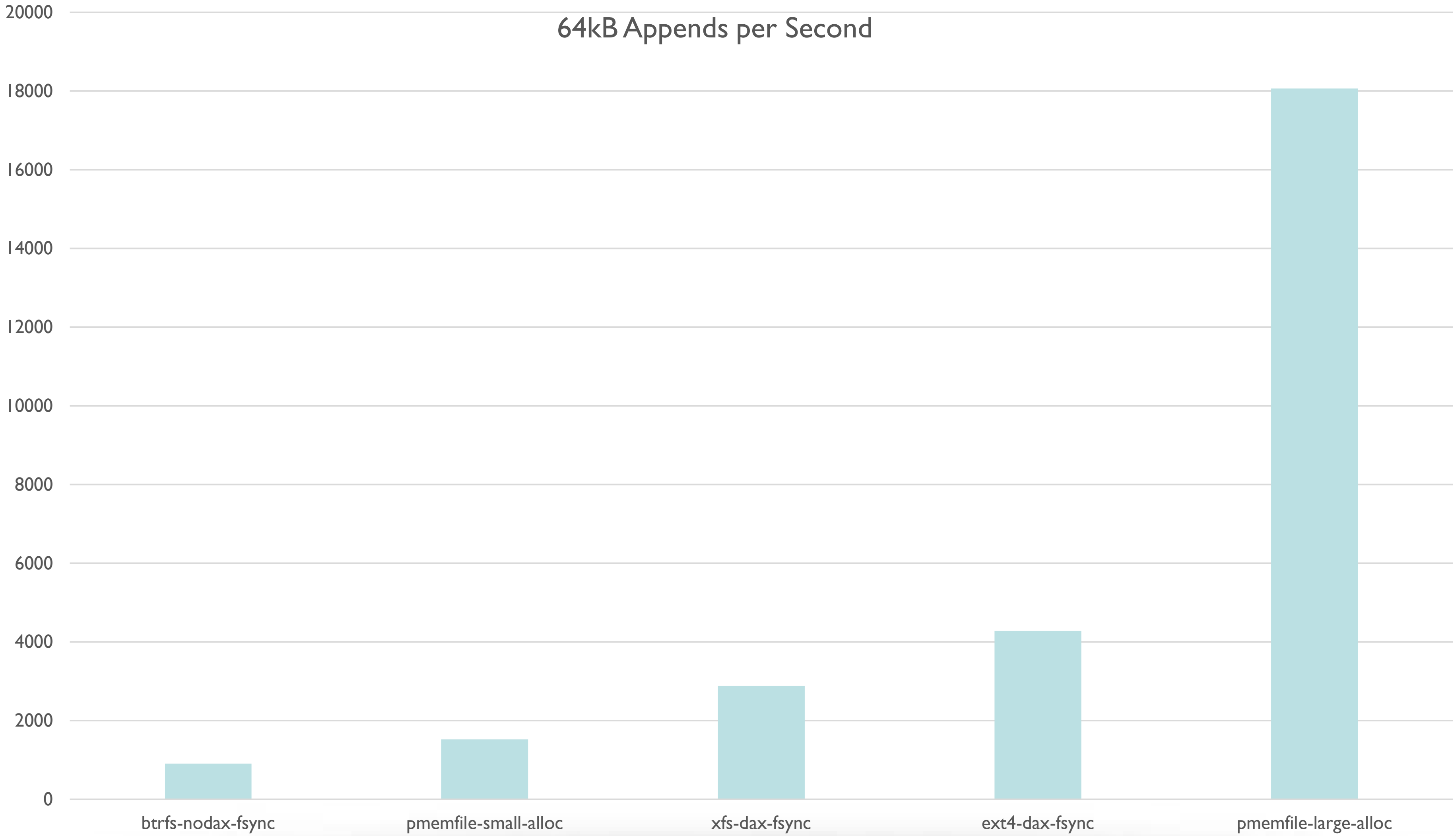
- ❐ ld.so and libc try to protect the app from unexpected behavior
- ❐ No well-specified, high-performance interception method available

- ❐ Like supported syscalls, simple interposition may be "good enough"

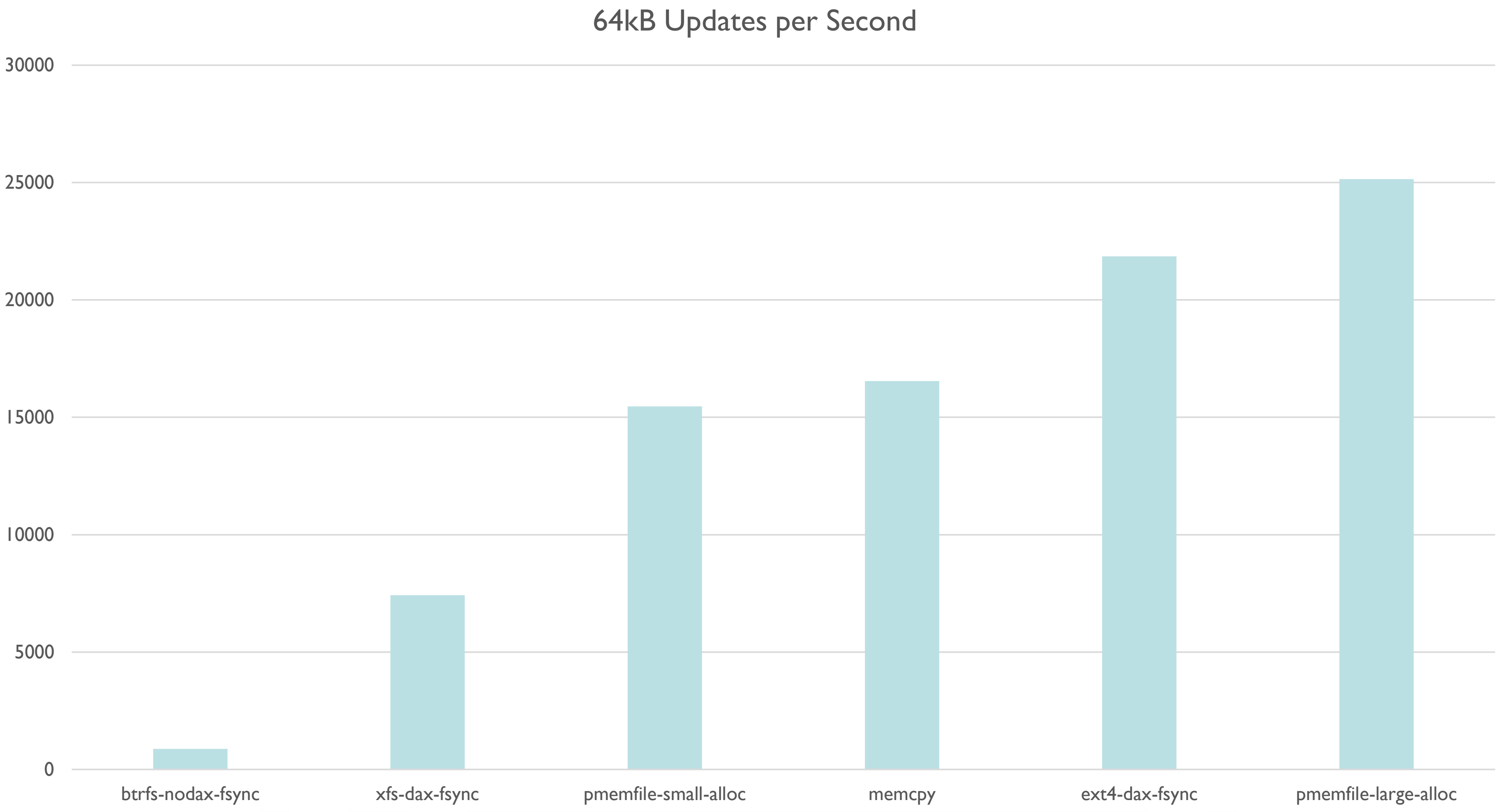
libpmemfile Performance

- The thing to beat…
  - pmem-aware file system
    - ext4, xfs, ntfs
  - Or traditional file system on block driver
- Code path for things like append…
  - Traditional
    - Deep through FS code, includes metadata updates
  - libpmemfile
    - load/store/cache flush instructions in user space

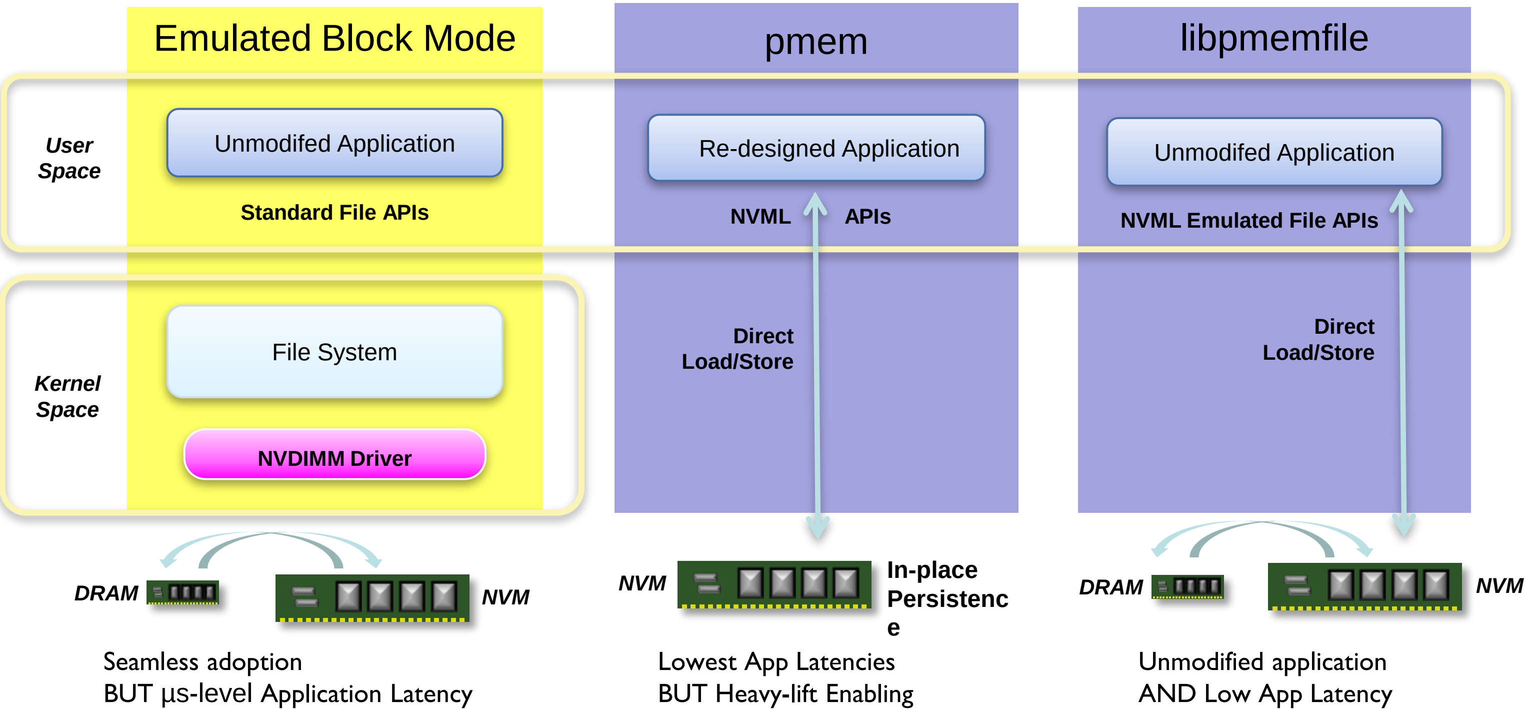
Proof-of-concept Results



Proof-of-concept Results



Summary



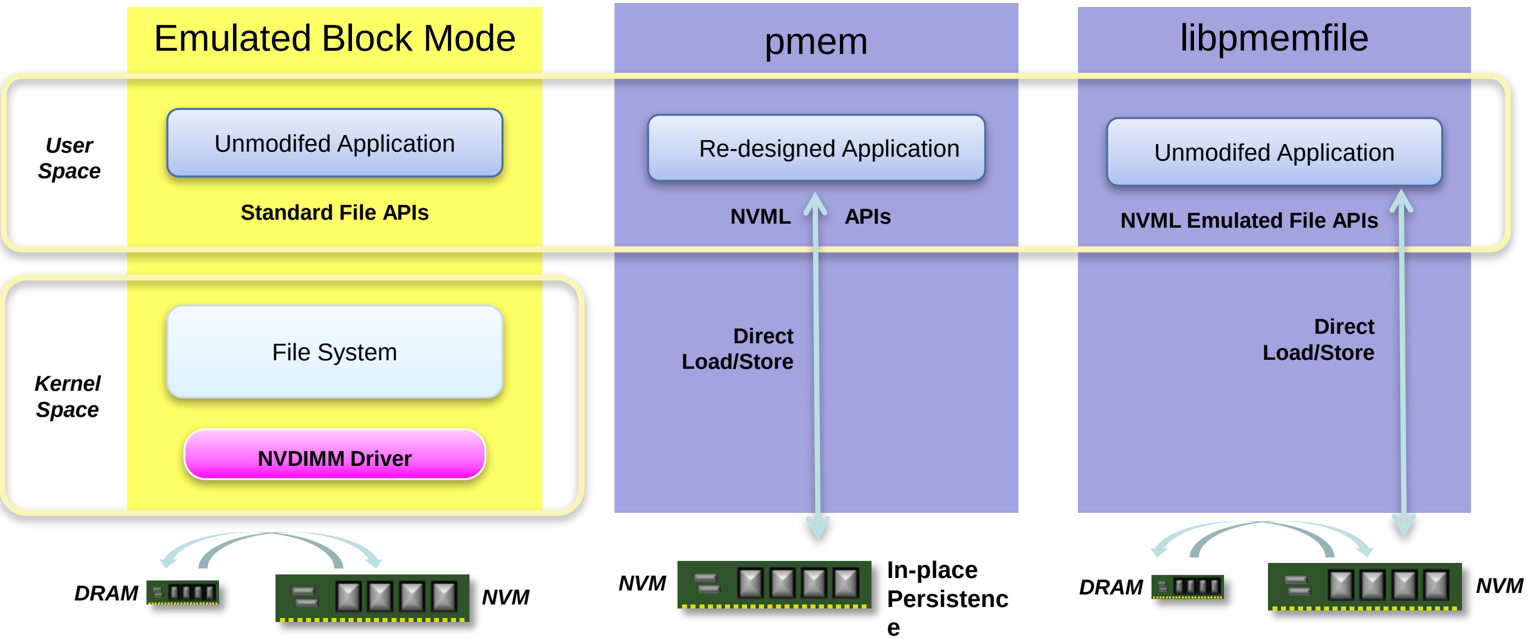
Seamless adoption
BUT µs-level Application Latency

Lowest App Latencies
BUT Heavy-lift Enabling

Unmodified application
AND Low App Latency

libpmemfile can provide much of the latency benefit without App changes

Summary



Seamless adoption
BUT µs-level Application Latency

Lowest App Latencies
BUT Heavy-lift Enabling

Unmodified application
AND Low App Latency

libpmemfile can provide much of the latency benefit without App changes

Inherits libpmemobj features like **replication**!

Summary

- Many ideas for transparent use of pmem
  - We describe one idea here, there are more!
  - Lowers the barrier to adoption
- Nobody is claiming they have the One True Answer yet (that I'm aware of)
  - Want to encourage multiple, competing ideas
  - Want to get some experience with solutions
  - Want to try pmem before re-architecting app
- Watch for libpmemfile sometime next year