LUG 2023: Buffered I/O, DIO & Unaligned DIO

Patrick Farrell

Lustre Data I/O Path

- Data I/O Path: How data moves between program memory and storage
- "What does the file system do when you call `read()` or `write()`?"
- Data flows from userspace, into Lustre client, through the network, and to storage (and back)
- POSIX gives two ways to do data I/O:
  - Buffered I/O
  - Direct I/O
- Each has benefits and drawbacks

Buffered I/O: Page cached I/O

- Buffered means ‘Uses the page cache’
  - All user data is copied through the page cache
- What’s a page cache?
  - An ordered set of pages in kernel memory which contain data from a file
  - Shared between all processes using a file
  - Tracked with a cousin of the classic binary tree
    - Allows parallel lookups but serial insertions (adding new pages)
  - Pages are created; inserted into cache; then data is copied to the page
    - Copied from userspace for writes
    - Copied from storage for reads
  - Copying into the page cache **aligns** data; allows a 1-to-1 mapping for copies to/from storage
  - Storage and RDMA requires aligned data for good performance

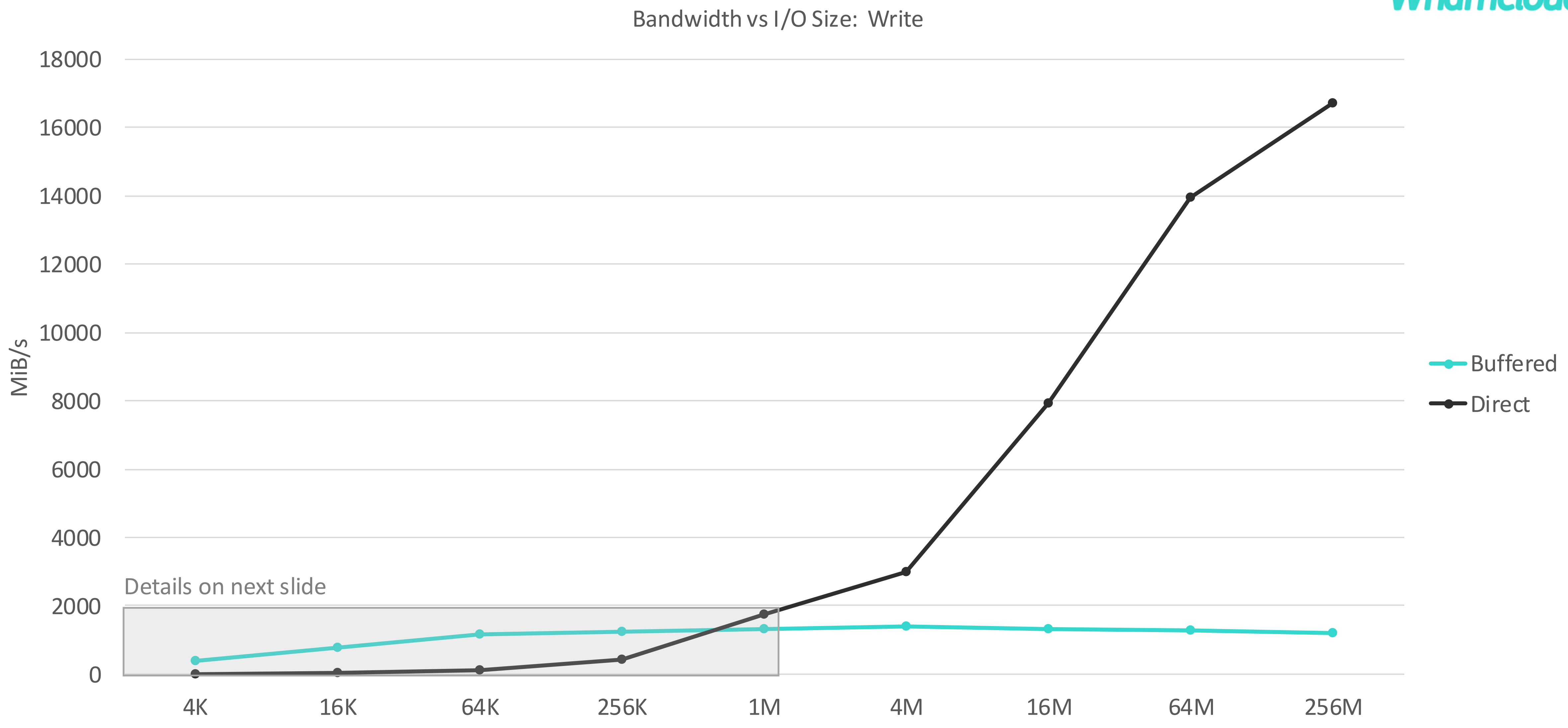
Buffered I/O

- Pros – Flexible:
  - Allows any I/O – no memory alignment requirements for userspace
  - Allows read ahead and write aggregation, converting small application I/O to large I/O on disk
  - Async writes and readahead are perfect for hiding latency of slow devices (HDD)
  - Repeat reads can be served from local cache
- Cons – Not scalable:
  - Significant overhead for cache management
    - Low single stream performance (max 1-3 GiB/s)
    - Minimal multi-process scalability due to locking

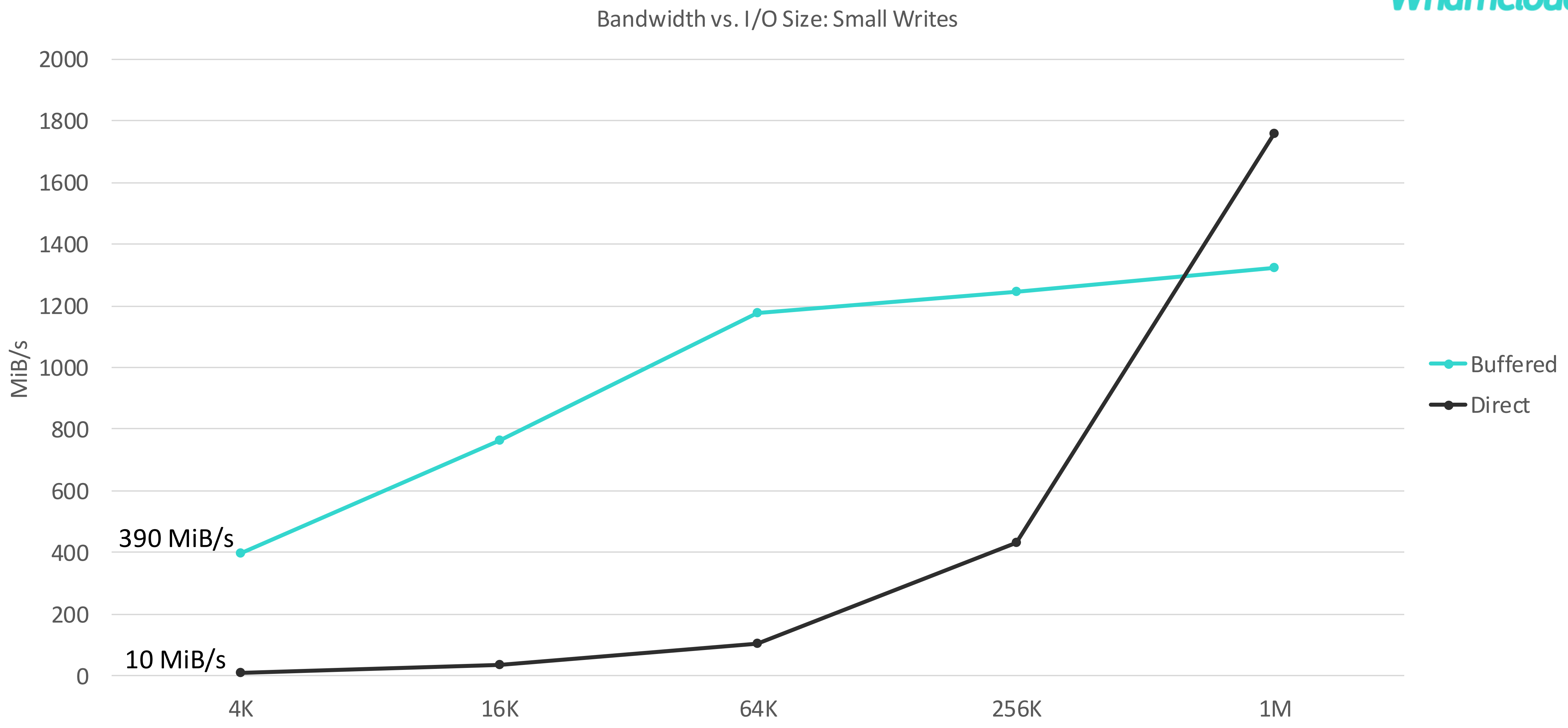
Direct I/O

- Direct I/O means 'Direct from user memory, does not use the page cache'
  - Very simple and clean – no locking required
- Pros – Scalable:
  - Very high single stream performance with large I/O – 18+ GiB/s
  - Scalable as processes are added (for I/O to 1 file or to many files)
- Cons – Inflexible:
  - Synchronous. I/O must go directly to disk, no async write or readahead
    - Exposes latency of slow devices
    - Can't do readahead or write aggregation
    - Bad for small I/O
  - Alignment requirement
    - Size of I/O and location in memory must be a multiple of page size
    - Can't be used without special effort from user program/libraries

Buffered vs Direct: Performance with I/O Size



Buffered vs Direct: Small I/O Performance



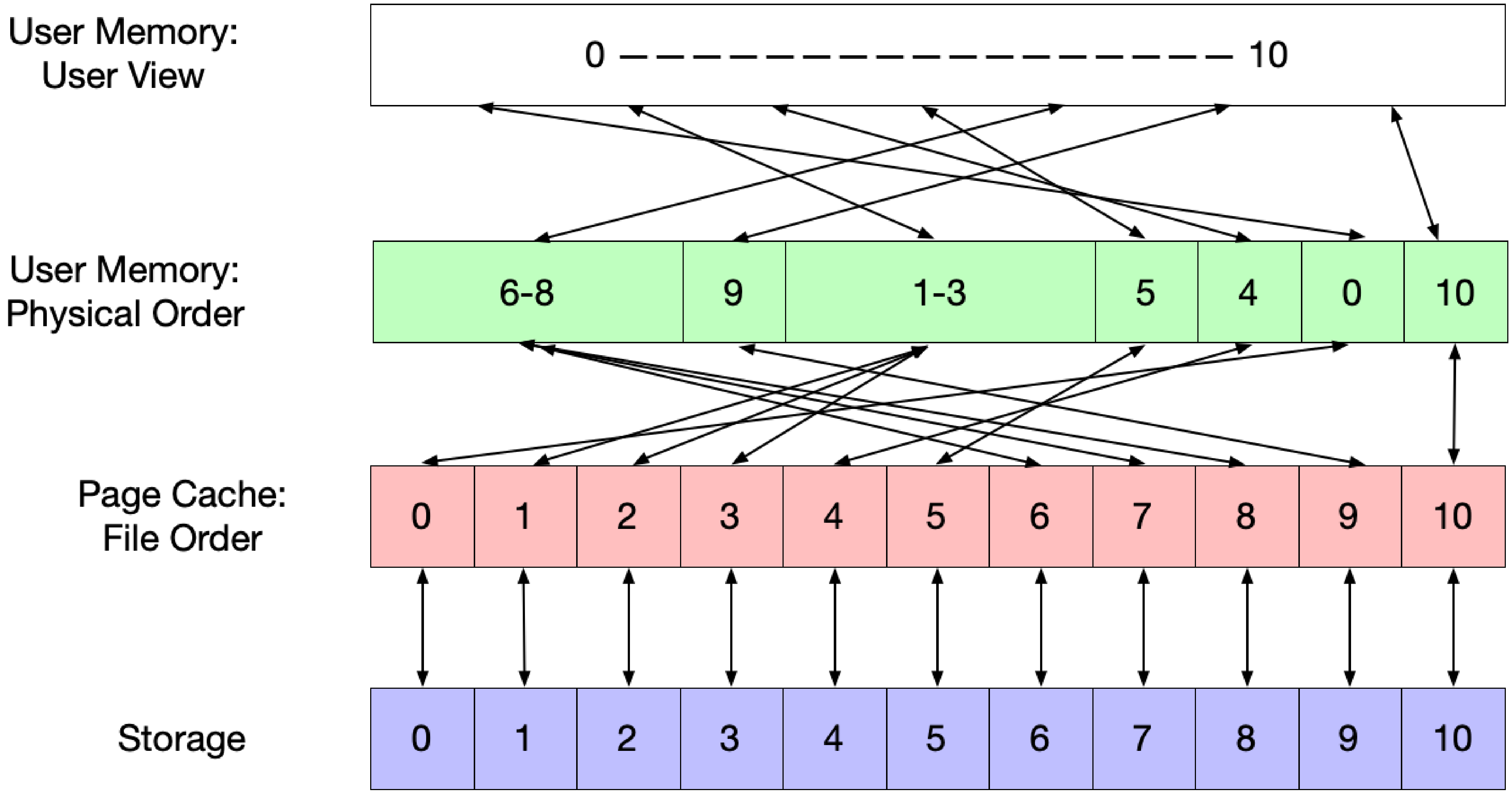
Buffered vs Direct: Summary

	Buffered I/O	Direct I/O
Small I/O Performance	✓	X
Large I/O Performance	X	✓
Many Processes	X	✓
High latency Storage (HDD)	✓	X
Unaligned I/O	✓	X

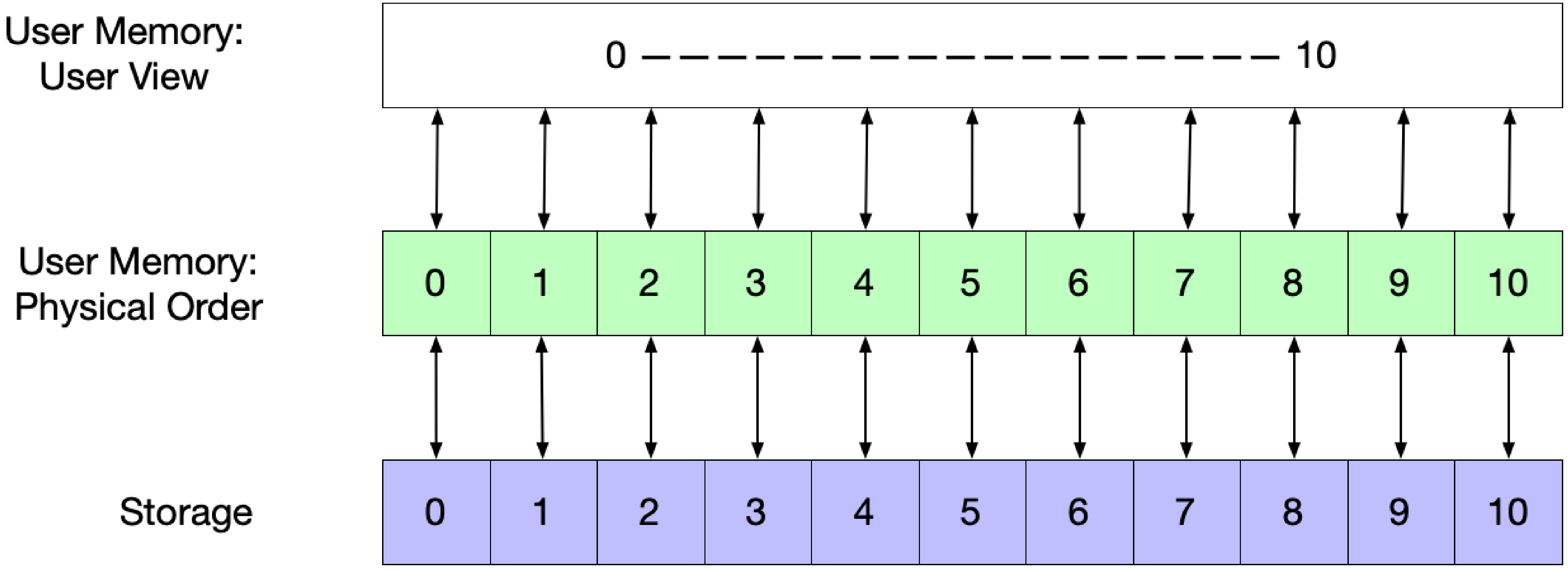
Buffered + Direct: Let’s have it all

- Strengths and weakness of buffered I/O and direct I/O pair up perfectly
- Use buffered I/O for small I/O and direct I/O for large I/O
  - Userspace can do this, but requires application/library modification
- Can we dynamically select the IO type to use inside the file system?
- **Ah, but alignment requirements…**
  - Can’t do arbitrary I/O as direct I/O, because I/O isn’t necessarily memory or size aligned.
- Must be aligned for good performance with RDMA and read/write from/to storage
  - Unaligned RDMA and disk I/O can be done, but at significant cost
- Buffered I/O is aligned by copying into the page cache
- Direct I/O must be aligned in userspace by application

User Memory & the Page Cache



Aligned User Memory & Direct I/O



Getting Alignment: Caches vs Buffers

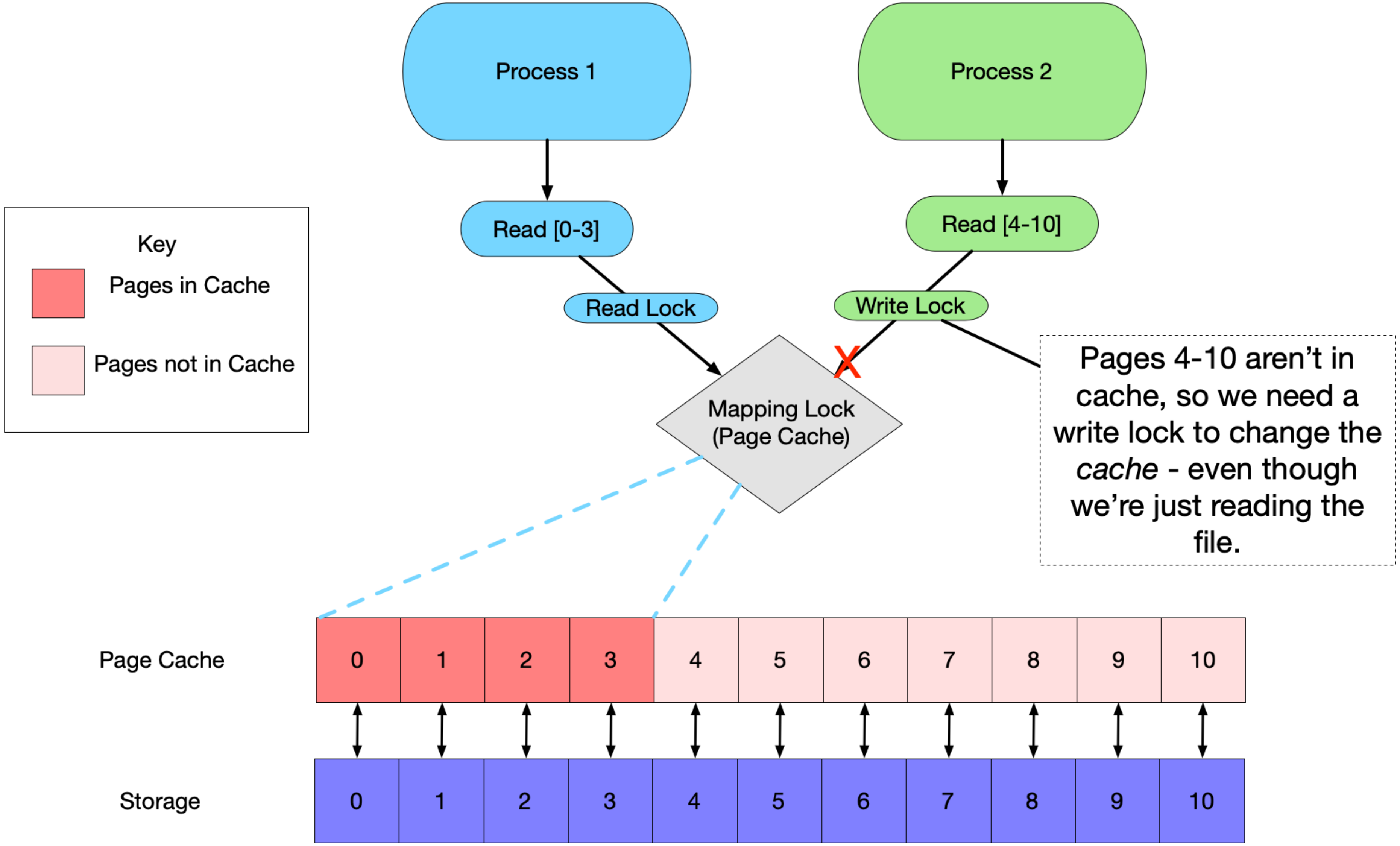
Whamcloud

- Page cache gives you alignment, but is very expensive
- Copies unaligned data in to aligned pages
- A cache can be used repeatedly & accessed from multiple threads
  - Requires lots of concurrency management and locking
  - Most cost of cache is not in data copying – cost is in cache setup
- But copying to aligned pages is what gets you alignment – no needed for a cache

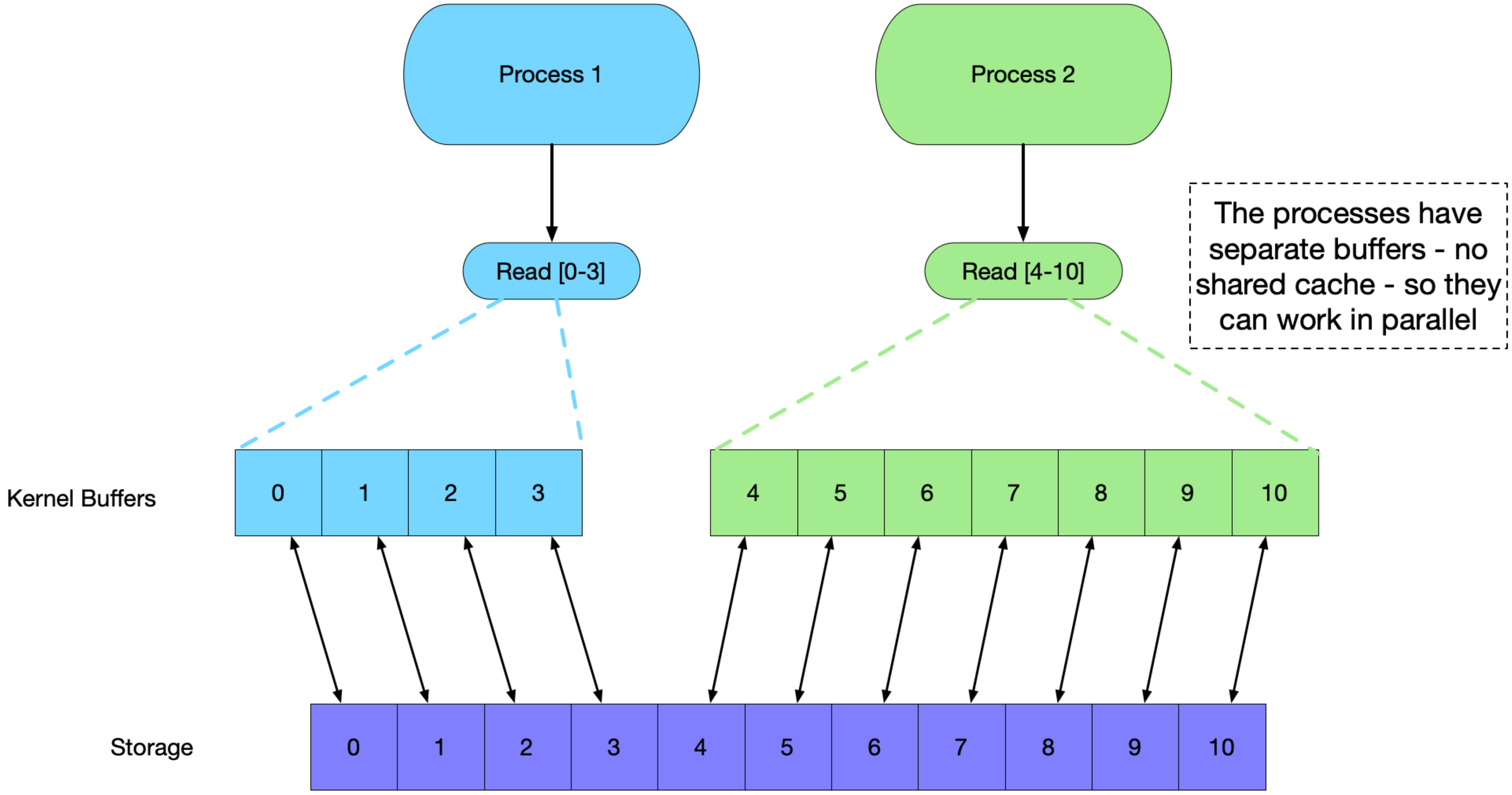
Unaligned DIO: Buffer, no cache

- To get alignment:
  - Allocate an aligned buffer
  - Copy data to/from the buffer
  - Do direct I/O from the buffer
- I/O is still synchronous – when write() returns, I/O is complete
- Buffer isn't accessible from other threads
- No need for cache setup or locking

Reference: Page Cache Locking



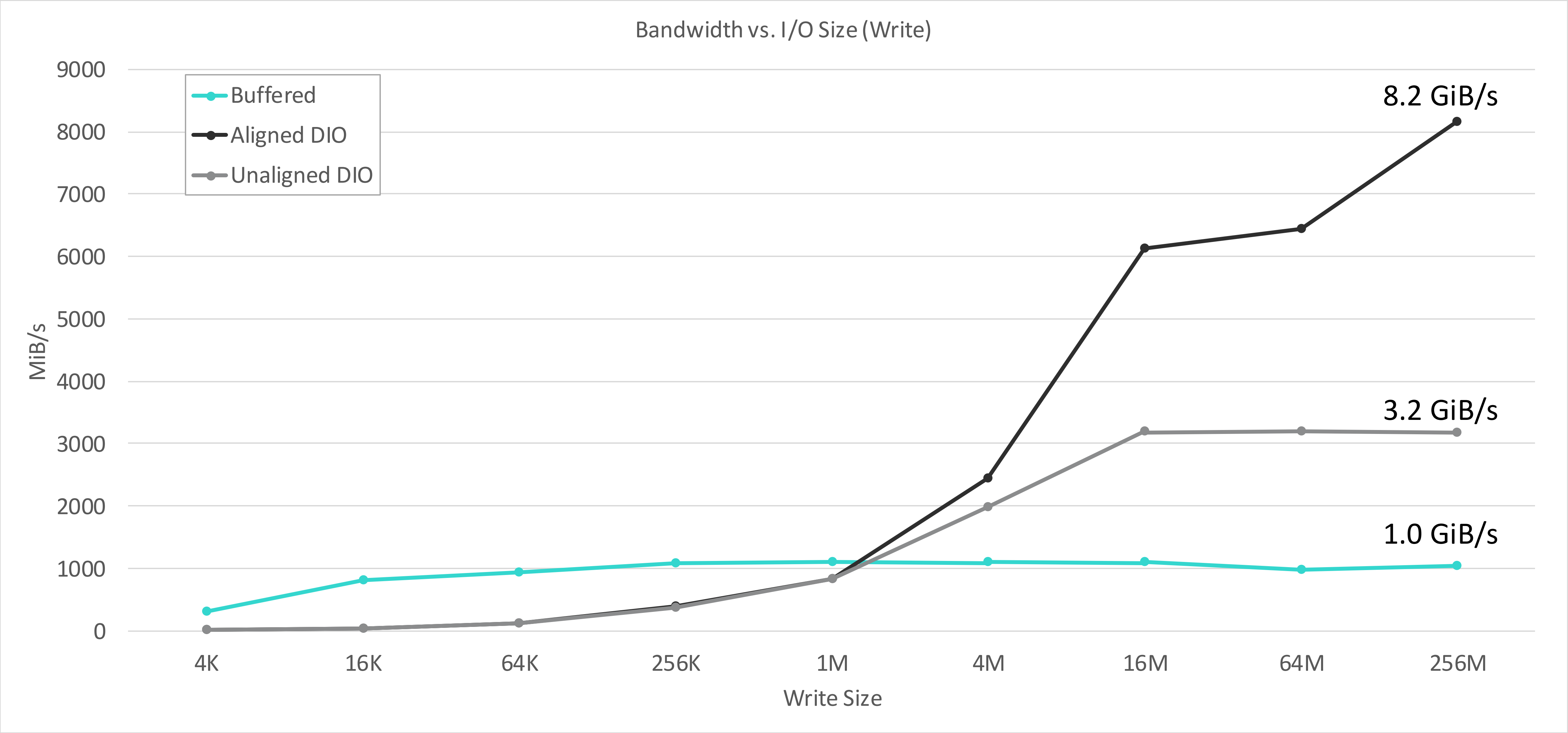
Unaligned DIO: Buffers, but no cache



Caveat on Numbers

- Hardware is different than previous graphs
  - This hardware's limit is ~10 GiB/s for single threaded DIO
  - Not 18 GiB/s limit on previous hardware
- This is v1, various optimizations can be made in the future

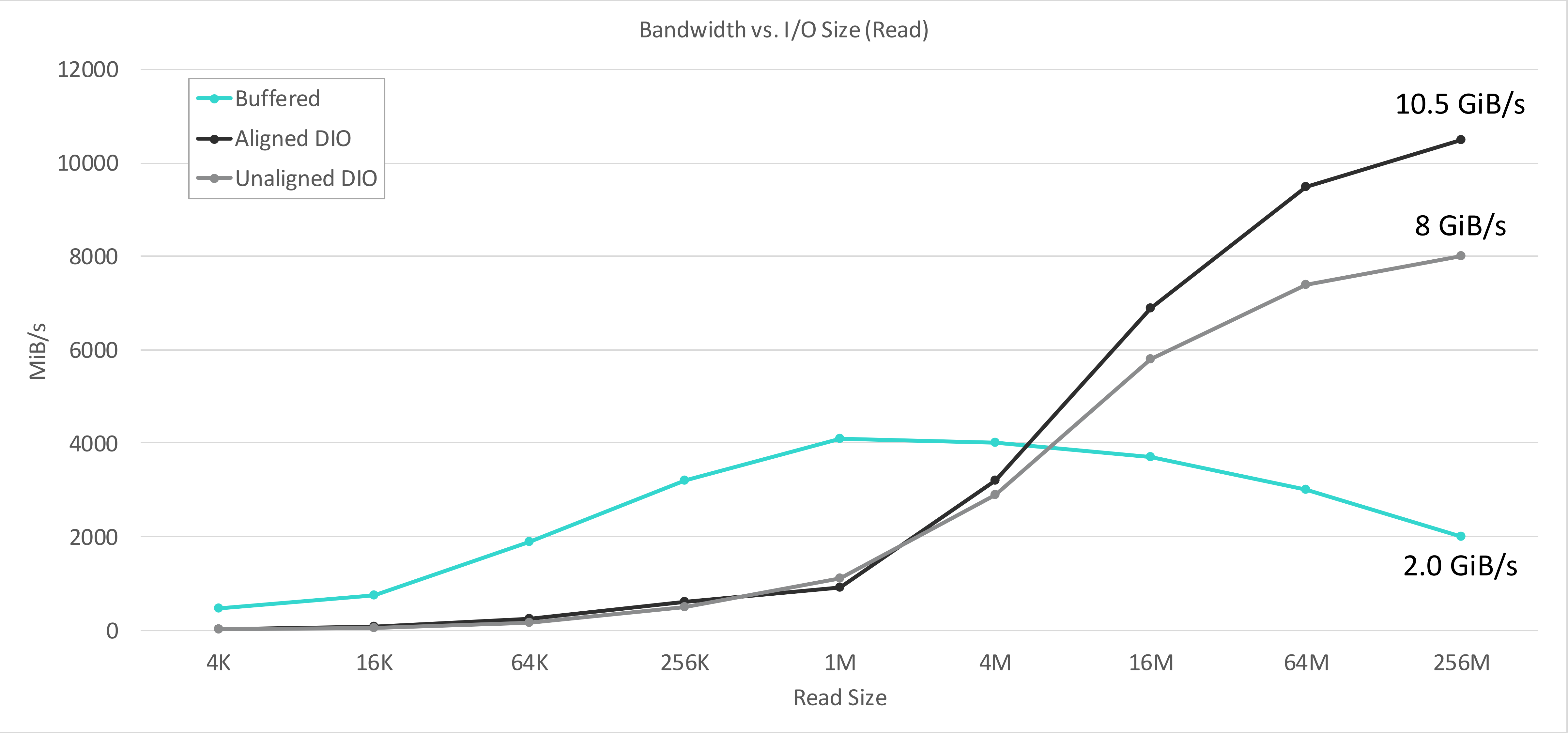
Unaligned DIO: Write Performance



Unaligned Direct I/O: Performance

- 3.2 GiB/s single threaded write is nice, but just 40% of aligned DIO (8 GiB/s here)
- Well, data copy and memory allocation are pretty time consuming
- But, yes, we can do better
- `memcpy()` for buffered I/O is single threaded
  - It's not any faster to parallelize
  - Locking and coordination of cache bottlenecks
- But DIO is different. No locking, so we can parallelize

Unaligned DIO: Read Performance



Unaligned Direct I/O: Performance

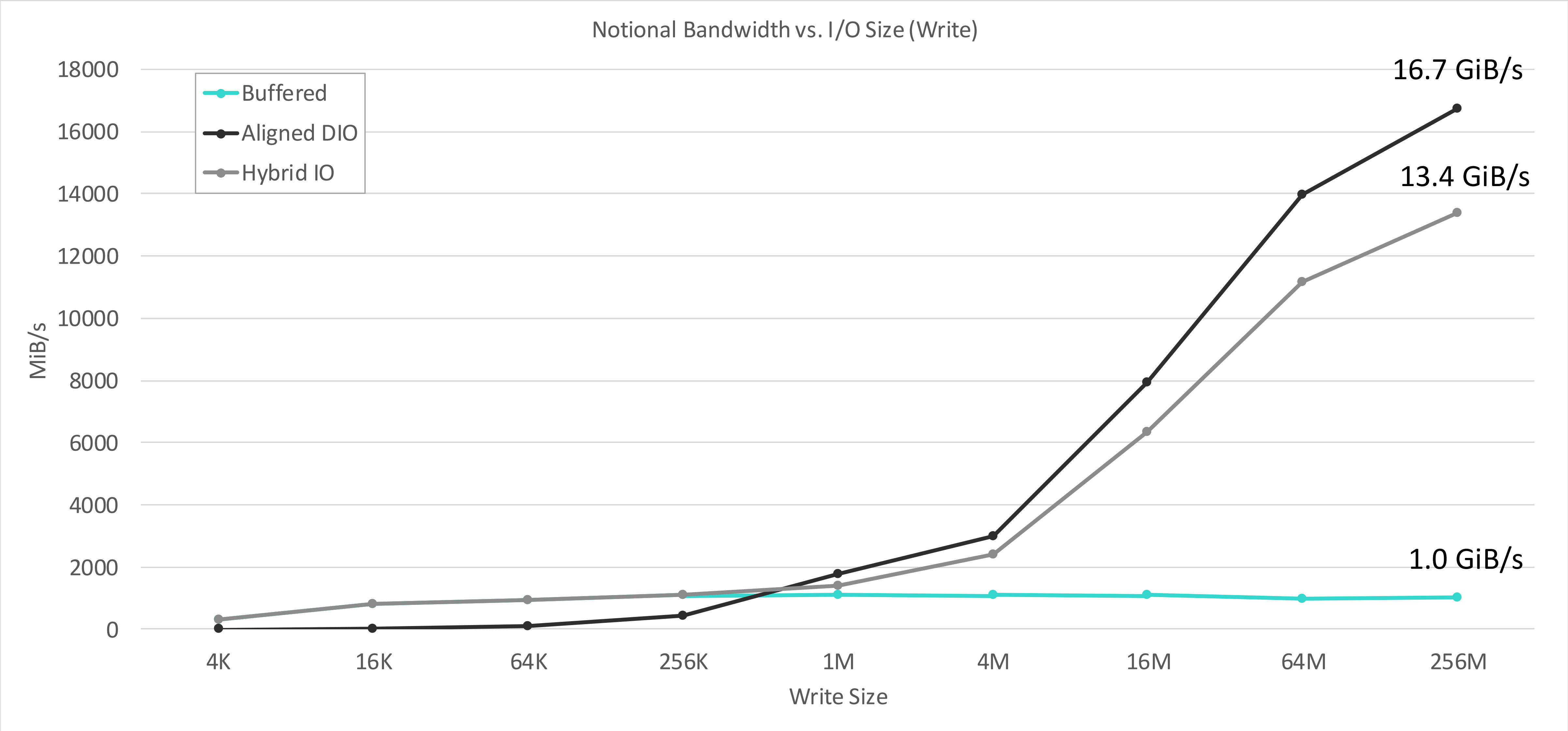
Whamcloud

- Unaligned DIO read is at 8 GiB/s of 10.5 GiB/s for DIO (76%)
- Copy for unaligned DIO read is parallelized
  - Farms out data copy for each DIO to many daemon threads
- Data copy for write **will** be parallelized but is trickier. Will not be in initial version.
- Read & write will have both allocation and copy parallelized
  - Expect >76% of DIO performance
- Will scale with DIO performance
  - 18 GiB/s DIO implies ~13 GiB/s unaligned DIO

Unaligned Direct I/O & Hybrid I/O: The Plan

- Finish unaligned direct I/O
- Test and optimize
- Once that's done:
- Implement hybrid I/O path
  - Userspace does simple `read()` or `write()` calls
  - Lustre decides internally to do buffered I/O, or unaligned DIO (or aligned DIO if possible)
- Gets the best of both worlds
  - Readahead and write aggregation at small sizes
  - High efficiency at large sizes

Hybrid I/O: Where We're Headed



Unaligned Direct I/O and direct I/O: Future work

- Unaligned direct I/O: Lustre 2.16
  - Will allow direct I/O which is not a multiple of page size
  - Still strictly **opt-in**, does nothing if you're not using O_DIRECT
- Hybrid I/O: 2.16+
  - Simplest version should follow quickly after unaligned DIO
  - Aiming for gradual phase in
  - Use in increasingly more situations as we are sure it improves performance there
- Further DIO efficiency improvements
  - Referenced in LUG 2022 presentation Unaligned DIO & I/O Path Futures
  - DIO path is 18 GiB/s today, hope to reach 30+ GiB/s in future (LU-16640, LU-13814)
  - Will correspondingly boost hybrid I/O path performance

Thank you

- Thank you for listening
- See LU-13805 for further details
- See my LUG 2022 presentation for more on DIO improvements
- Questions to pfarrell@whamcloud.com
- Thanks to Nathan Rutman for a useful question in 2020

Thank You!