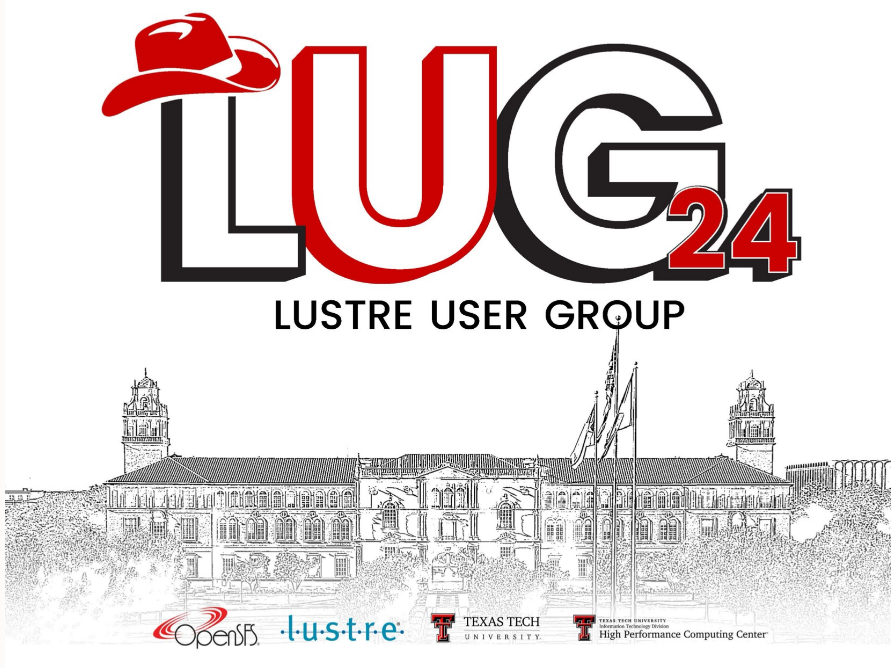
ORACLE



LUG 2024: Hybrid IO Update

Hybrid IO, or the new client data path

Patrick Farrell

Principal Engineer

OCI Storage

May, 2024

Lustre: Hybrid IO – A New Lustre Data Path

- Hybrid IO is a new data path for the Lustre client.
  - Hybrid IO combines buffered and direct IO, getting the best of both
- Today we'll talk about…
  - What Hybrid IO is…
  - And exactly what's in Lustre 2.16

Data Path

- Data 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 (on write, and opposite on read)
- POSIX gives two ways to do data I/O:
    - Buffered I/O
    - Direct I/O

Buffered 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
  - 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 to storage
  - Storage and RDMA require aligned data for good performance

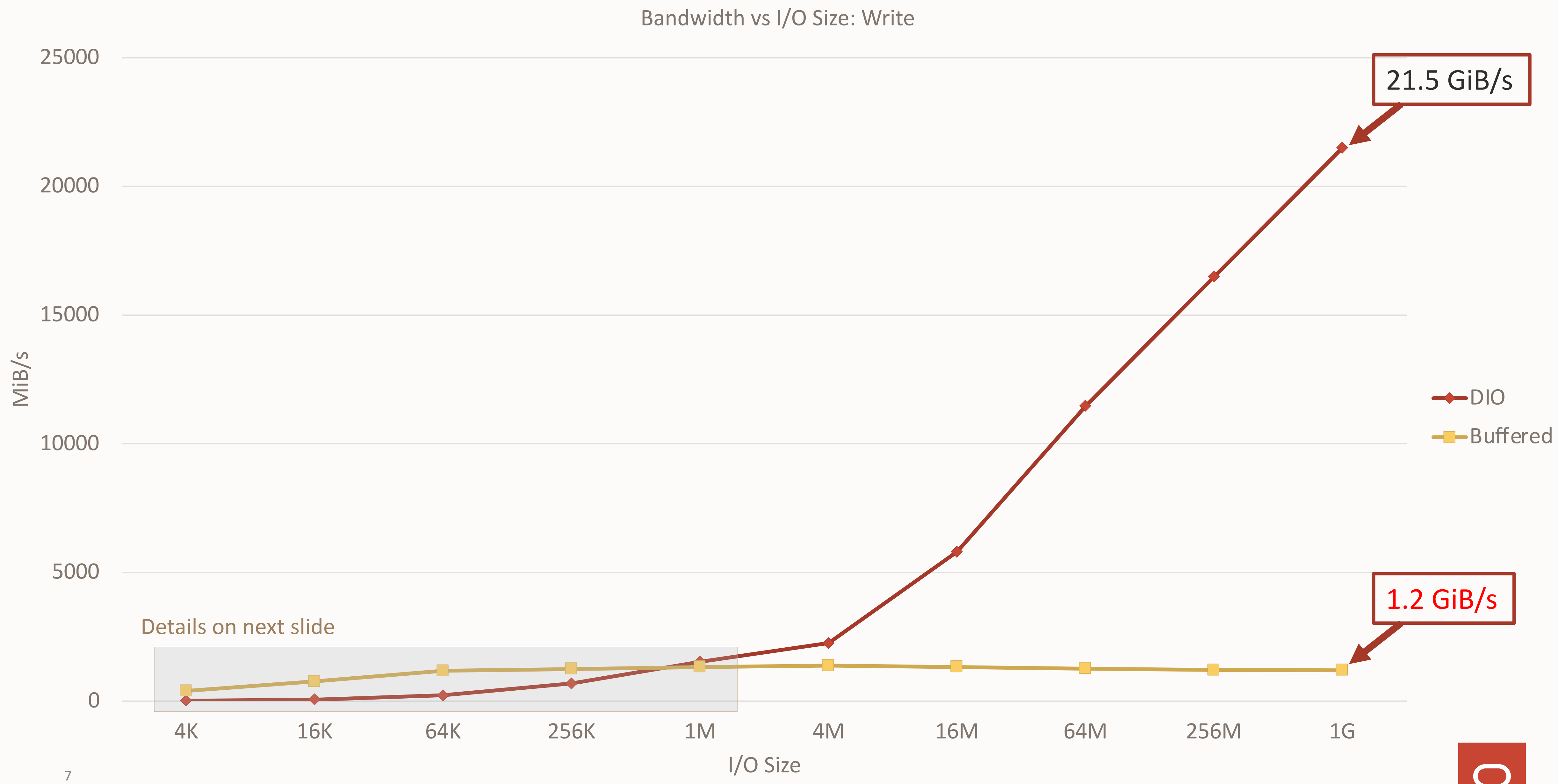
Buffered I/O

- 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)
- 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 – 20+ 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

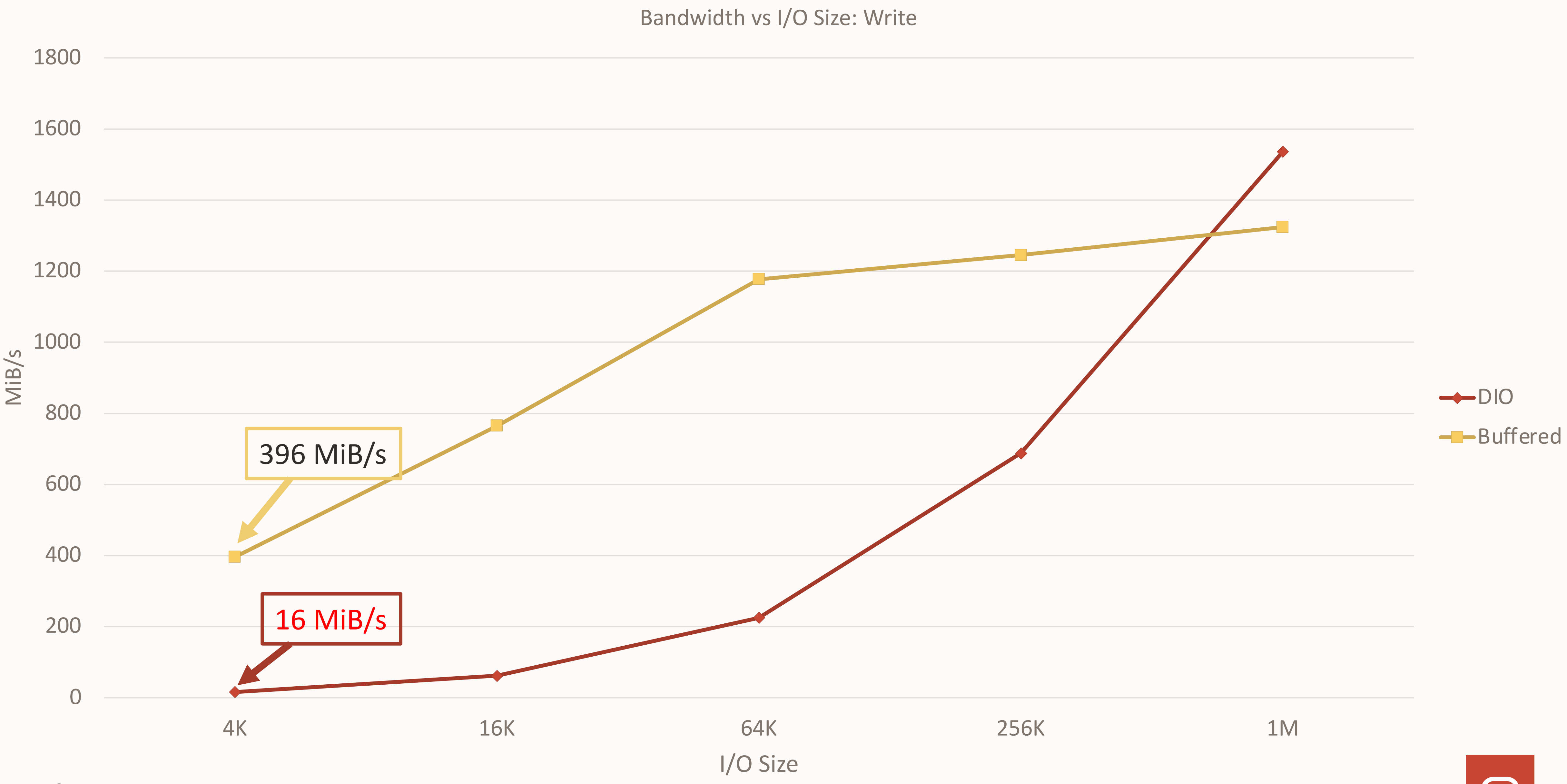


21.5 GiB/s

1.2 GiB/s

Details on next slide

Buffered vs Direct: Performance at Small Sizes



396 MiB/s

16 MiB/s

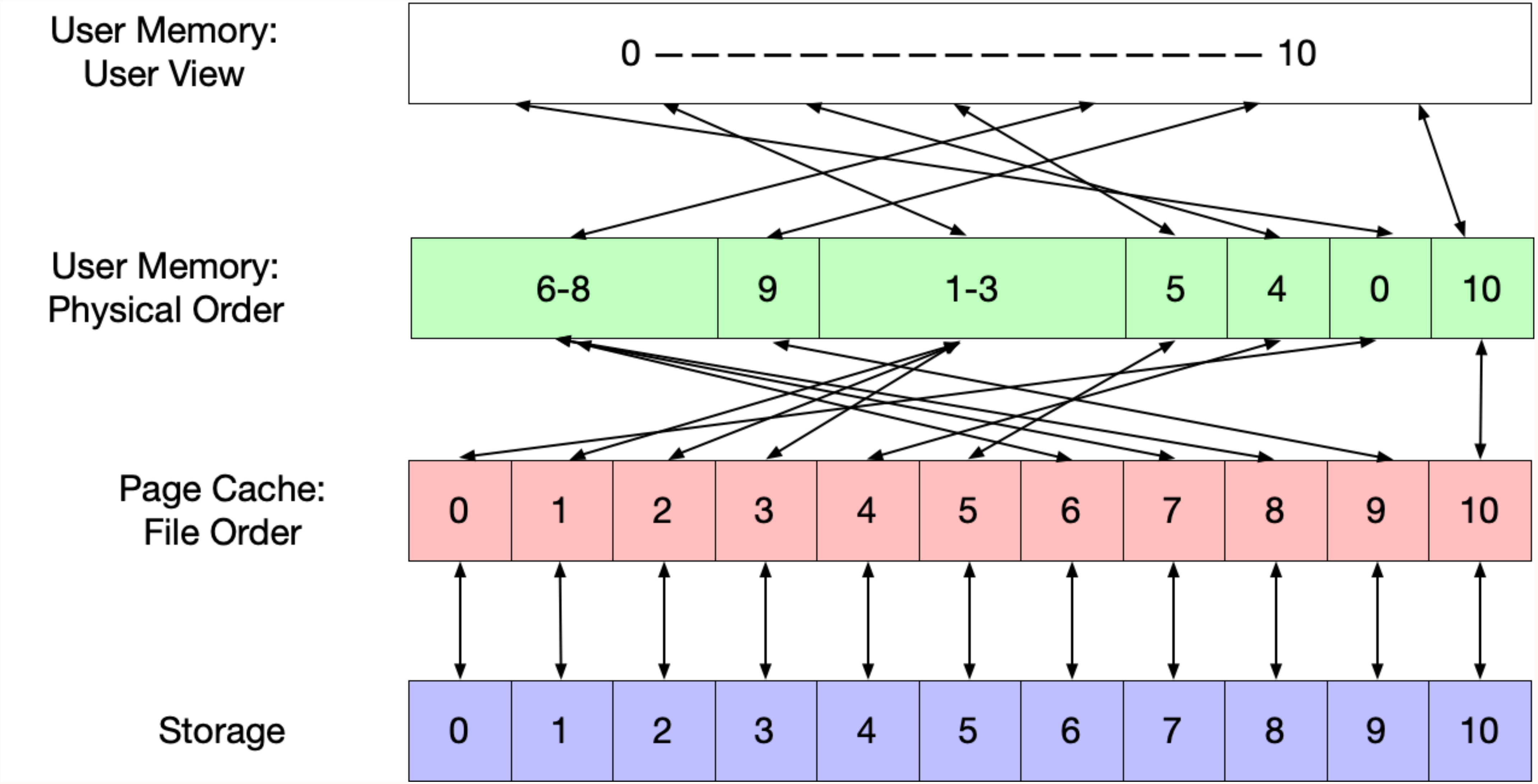
Buffered vs Direct: Summary

	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

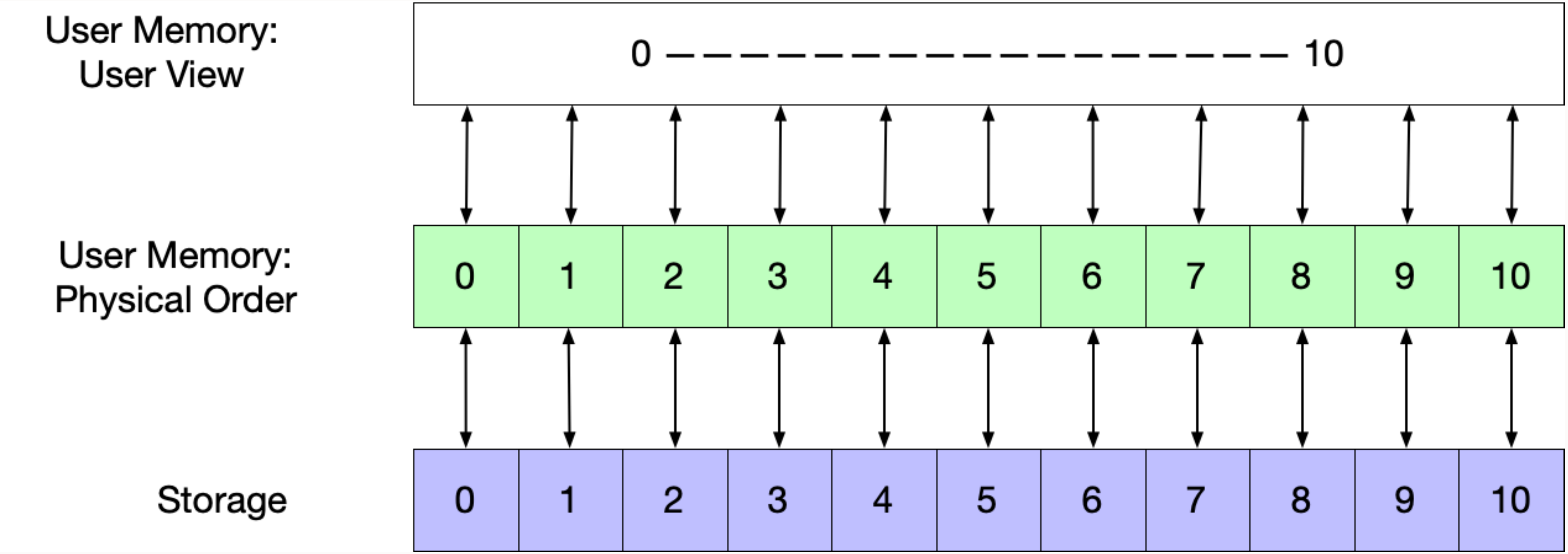
Buffered + Direct: Let’s have it all

- 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 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



User Memory: User View	0 – – – – – – – – – – – – – – – – – 10										
User Memory: Physical Order	0	1	2	3	4	5	6	7	8	9	10
Storage	0	1	2	3	4	5	6	7	8	9	10

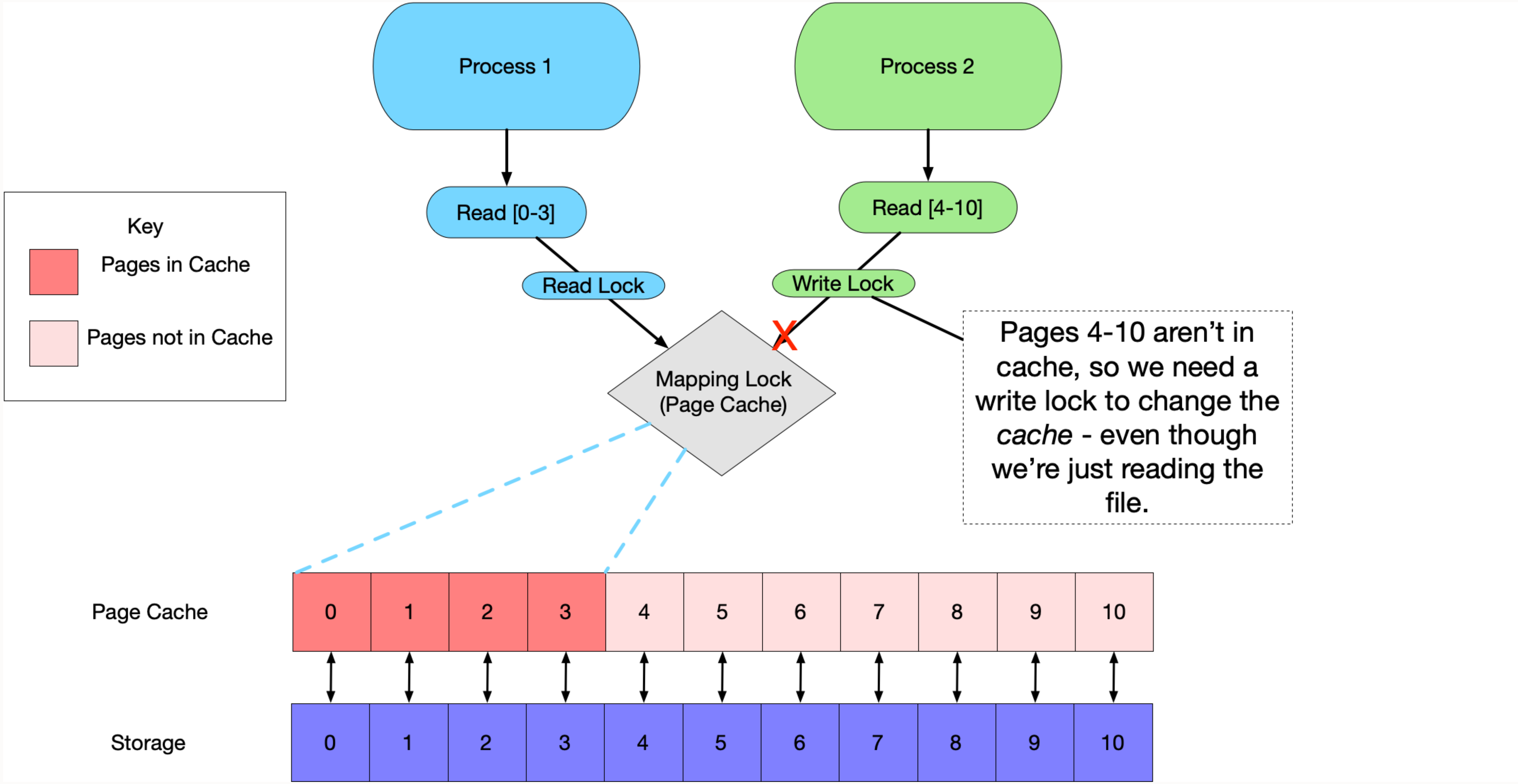
Getting Alignment: Caches vs Buffers

- 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 need 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

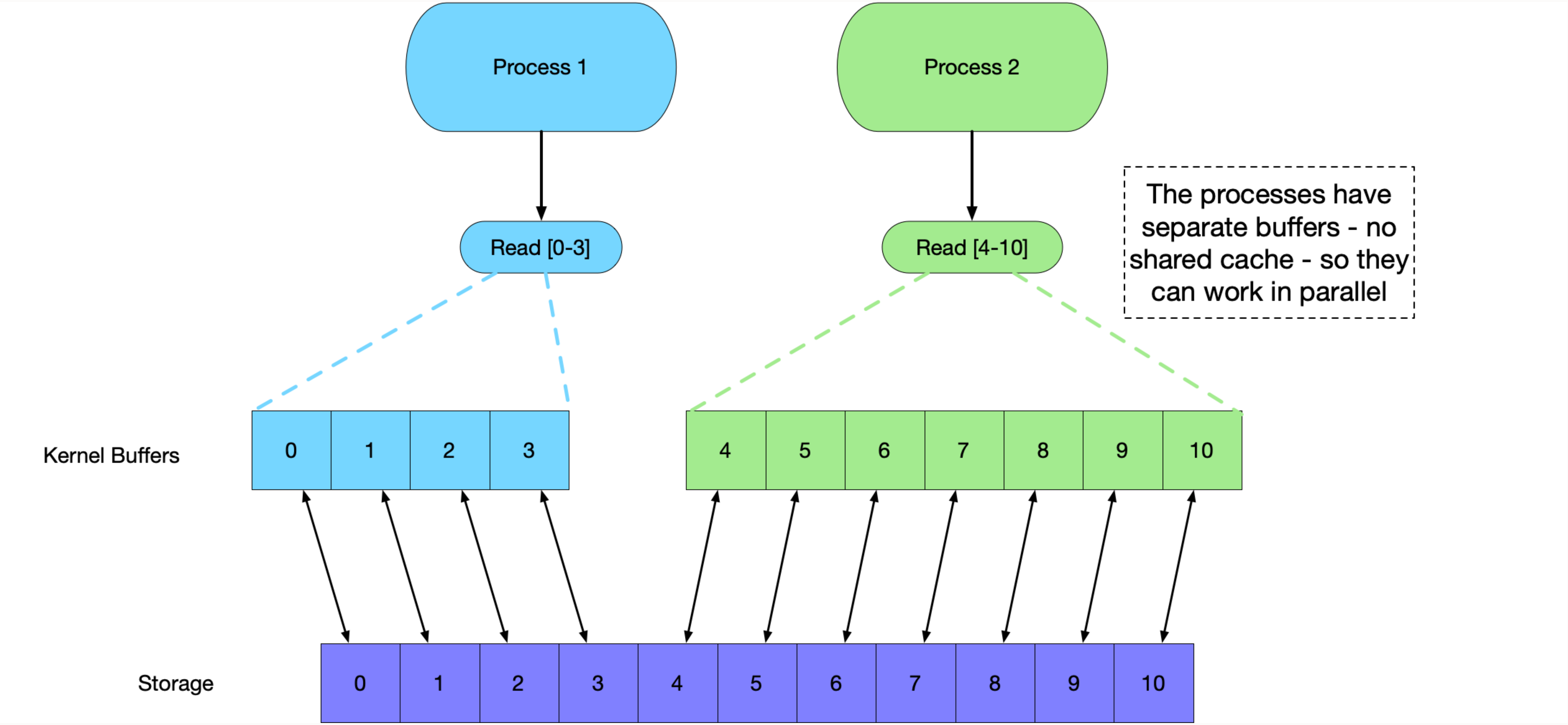
Reference: Page Cache Locking



Pages 4-10 aren’t in cache, so we need a write lock to change the *cache* - even though we’re just reading the file.

Page Cache	0	1	2	3	4	5	6	7	8	9	10
Storage	0	1	2	3	4	5	6	7	8	9	10

Unaligned DIO: Buffers, but no cache



The processes have separate buffers - no shared cache - so they can work in parallel

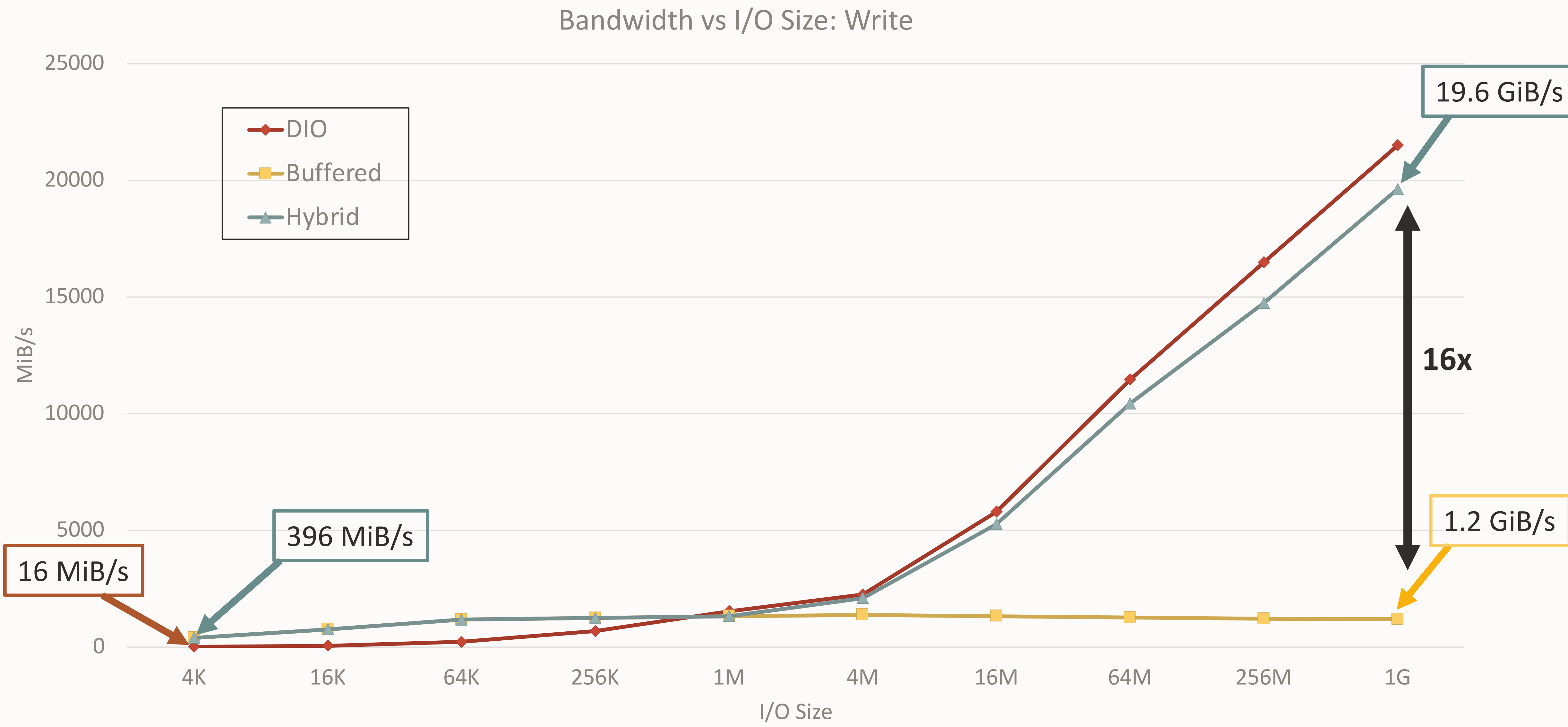
Unaligned DIO → Hybrid I/O

- Unaligned DIO allows any IO as DIO
  - Removes memory alignment requirement
  - A little slower than regular DIO – But still very fast
  - Enables hybrid IO
- Hybrid IO:
  - Use buffered IO for small IO
    - Benefit from readahead and write aggregation
  - Use unaligned direct IO for large IO
  - Performance goes up with I/O size throughout
- Let’s see what that looks like…

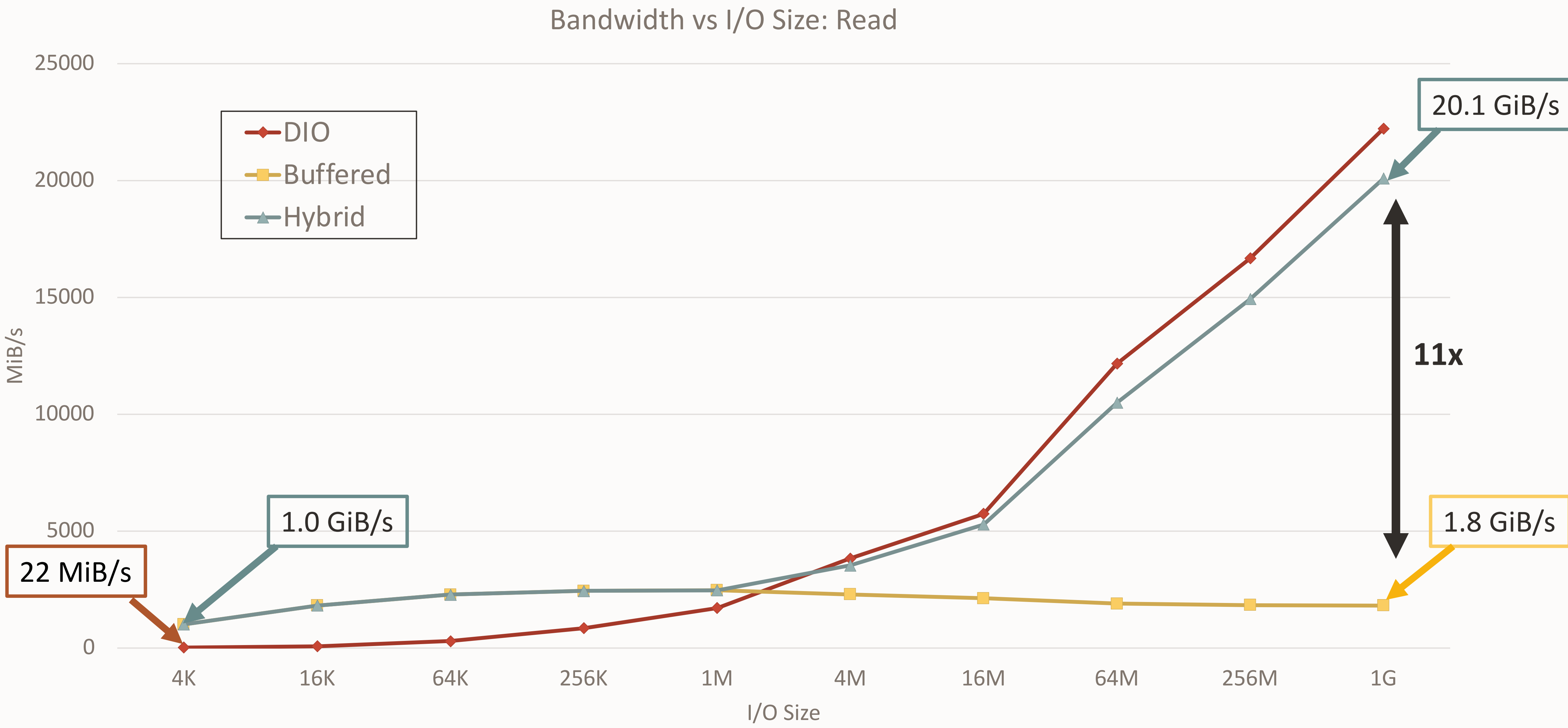
Notes on Numbers

- Most data gathered on an NVIDIA DGX, many CPUs and multiple NICs
  - Shared file info gathered on dual socket OCI systems
- All numbers should be understood as general guidance – look at trends and relative values, not specific numbers

Hybrid IO: Write Performance



Hybrid IO: Read Performance



20.1 GiB/s

1.0 GiB/s

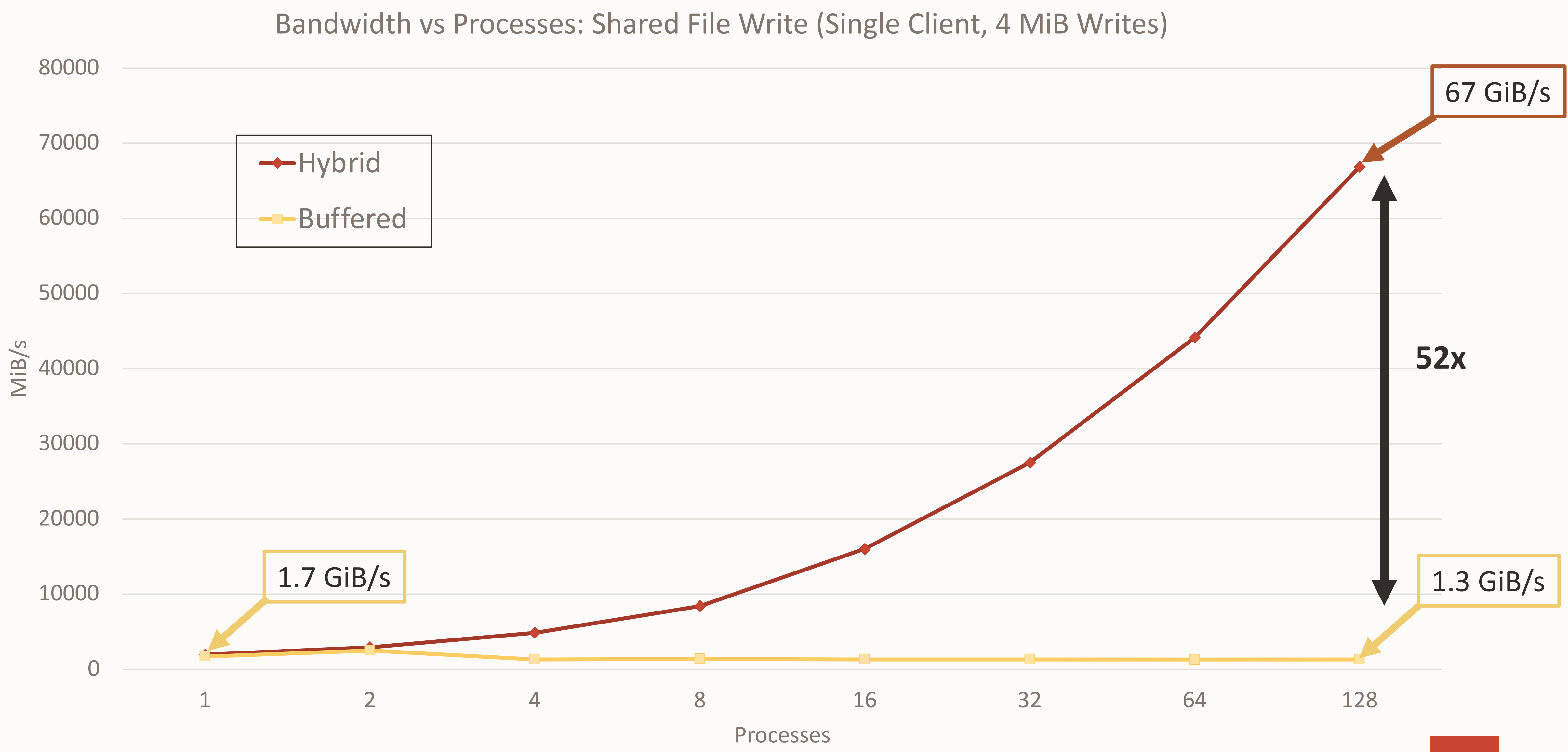
22 MiB/s

1.8 GiB/s

Hybrid I/O Performance

- Same as buffered at small sizes
- Uses unaligned DIO for larger sizes
  - Max at 19 GiB/s – About 90% of DIO
  - Compared to DIO, Unaligned DIO must:
    - Allocate buffer memory
    - Copy data to/from buffers
  - Buffered IO has to do these steps, but also cache setup
    - But when Hybrid uses **Direct IO**, there’s no cache
    - And we can do that allocation and copying in parallel. So it’s very fast.
  - Hybrid IO is also lockless, which has implications for shared file writes…

Single Client Shared File: Writes



67 GiB/s

52x

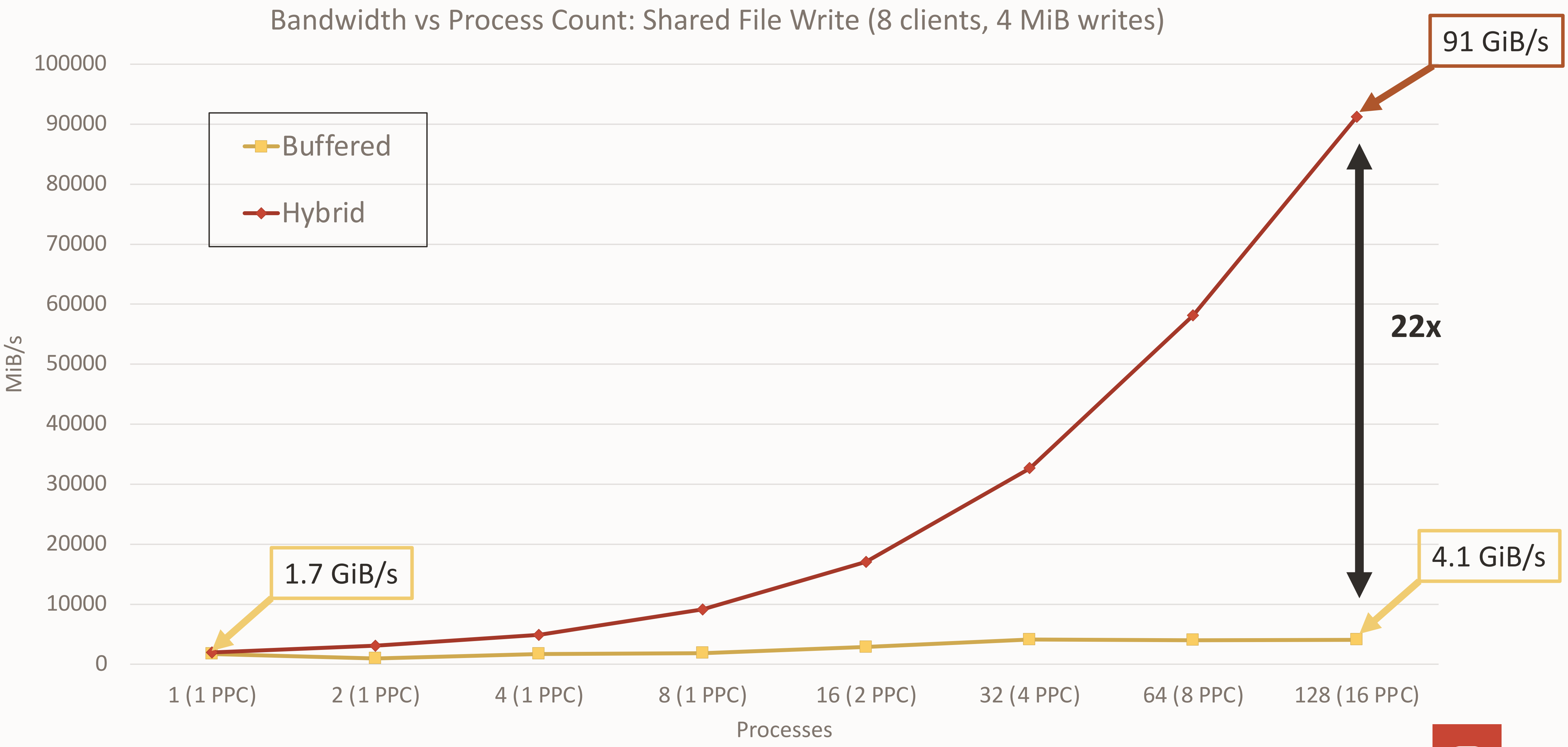
1.7 GiB/s

1.3 GiB/s

Hybrid IO: Shared File – Multi-Client Write

- Single client shared file write blocks on page cache locking
  - Hybrid IO avoids this entirely
  - But what about using multiple clients?
- Multiple clients writing to a shared file is a well-known pain point
  - Clients get Lustre distributed locks (LDLM) from the servers
  - Multiple clients writing to the same file bounce the locks around
    - Even if writes don't overlap
  - Write to one file doesn't scale with more clients unless you do something special
    - MPIIO, lockahead, overstriping, group locks…
- But Direct IO doesn't use the cache… So clients don't use LDLM locks.
  - So for larger writes, neither does hybrid IO
  - The effect of this is dramatic.

Multi-Client Shared File Writes



Hybrid IO in 2.16

- Hybrid IO is in Lustre 2.16
  - Automatically switches between buffered and unaligned direct IO based on size
  - Opt-in for now – Must be enabled:
    - lctl set_param llite.*.hybrid = 1
    - Some obscure bugs in testing, but passes all IO consistency tests, etc
  - Aiming for gradual phase in
    - On by default in 2.17
    - Currently only switch based on IO size
      - Switch in more situations as we are sure it improves performance
      - eg, lock contention
    - Also planning performance improvements…

Direct IO and Hybrid IO in 2.17+

- DIO path rewrite: LU-13814
  - Convert DIO from complex CLIO pages to simple arrays
    - No complex lists of dedicated structures for every page – just simple arrays
    - Huge efficiency improvement
  - Max single threaded DIO speed is 22 GiB/s today
  - LU-13814 takes single threaded DIO to 100 GiB/s
- DIO improvements boost hybrid IO performance
  - Hybrid + LU-13814 → 45 GiB/s (from 20 GiB/s today)

Hybrid IO Recap

- Buffered IO is:
  - Good for small IO (readahead, write aggregation)
  - Poor for large IO & shared file (no scalability)
- Direct IO is:
  - Terrible for small IO (synchronous)
  - Scalable for large IO & shared file
- Hybrid IO: Automatically get the best of both
  - No application modification – Just do normal read() or write() calls
    - Direct IO is unchanged – If you use O_DIRECT, you'll still get direct IO
  - Buffered performance for small reads & writes (20x improvement vs small DIO)
  - DIO-like scaling for large reads & writes (~10x improvement vs large buffered)
  - Solves shared file write problem on local node & across cluster (20-50x+ improvement)
- Hybrid IO is in Lustre 2.16, out later this year.

Thank you

Thank you for listening.
See LU-13805 for further details.

Questions to patrick.farrell@oracle.com

Thanks to Shaun Tancheff of HPE for critical assistance with parallel writes.

Thanks to Whamcloud and Oracle for supporting this work.

Hybrid I/O 2023 vs Hybrid I/O 2024

- Last year, Hybrid IO performance was limited…
  - Hybrid IO writes were at only 3 GiB/s (now 20 GiB/s)
  - Hybrid IO reads at 12 GiB/s max (now 20 GiB/s)
- Switched to page pools for allocations
  - Borrowed from the compression & encryption code
- Added parallel copy for writes
  - Thanks to Shaun Tancheff of HPE, who was instrumental in getting this working