IO in PostgreSQL: Past, Present, Future

Andres Freund
PostgreSQL Developer & Committer
Microsoft

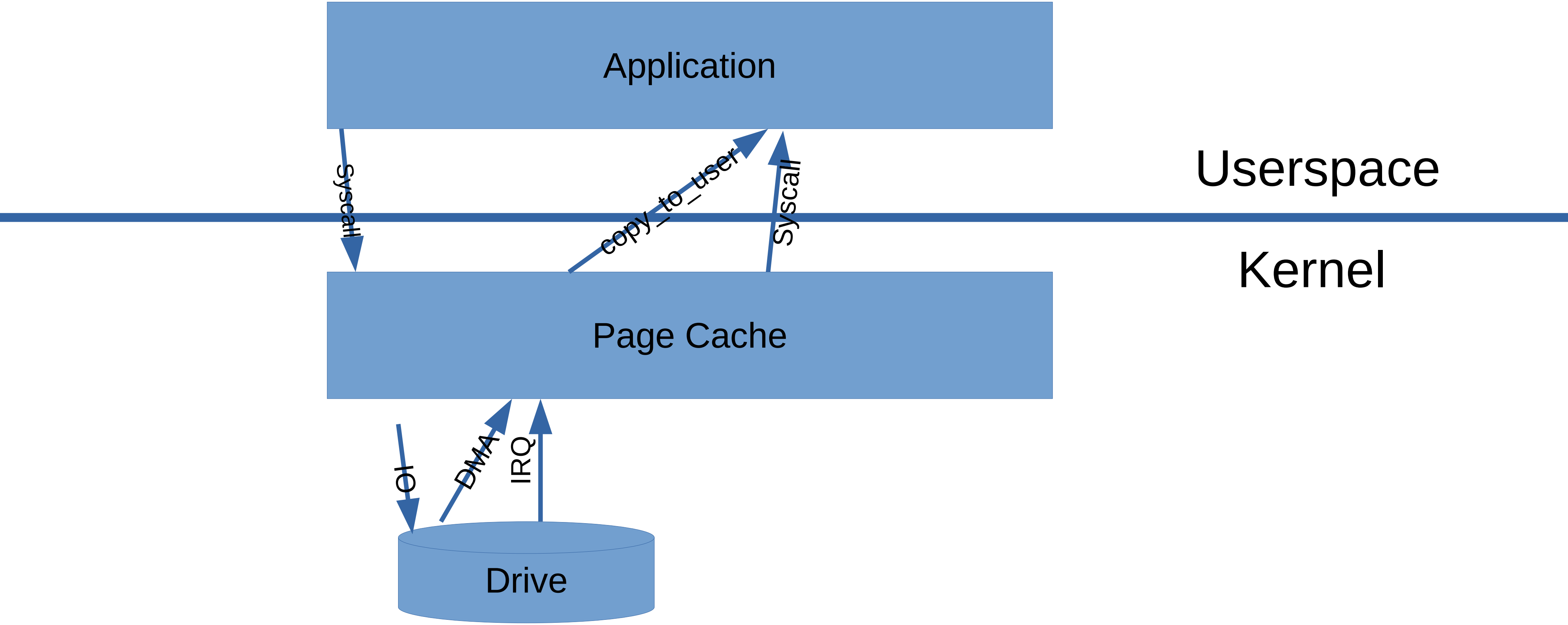
andres@anarazel.de
andres.freund@microsoft.com
@AndresFreundTec

https://anarazel.de/talks/2022-05-02-vaccination-db-aio/2022-05-02-vaccination-db-aio.pdf

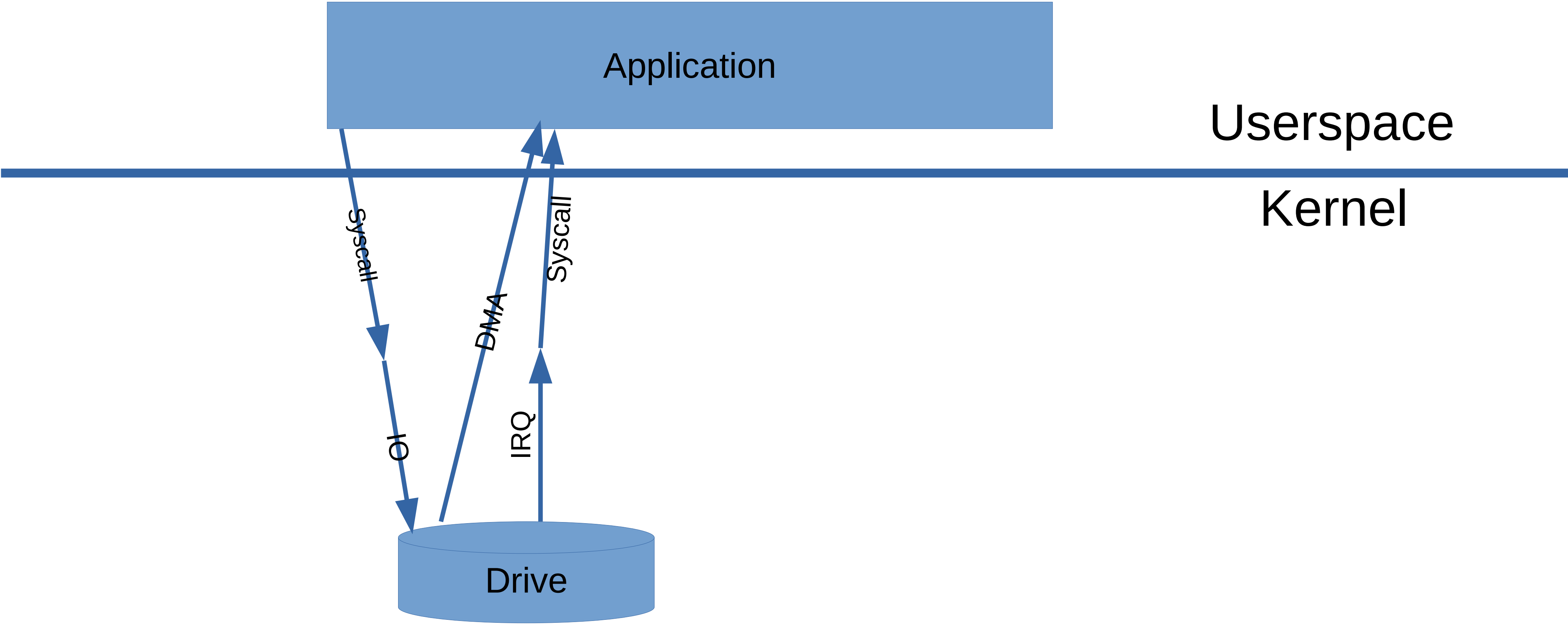
Past → Current postgres IO

- data buffer pool in shared memory
- buffered IO (`[p]read(), [p]write(), preadv(), pwritev()`)
- WAL (sync log before data writes) buffered in shared memory
  - group flush / commit, concurrent in-memory generation
- checkpointer, wal writer, background writer
- readahead via `posix_fadvise()`, OS
- postgres added control over dirty data in kernel page cache
  - `sync_file_range(SYNC_FILE_RANGE_WRITE), msync()`

Buffered read()



Direct IO (DIO) read()



Past → Current postgres IO

- WAL writer
  - flushes WAL, also be done by backends
- Buffer Replacement uses variant of CLOCK
  - state advanced by backends
  - good concurrency, but not much else is good
- Background writer
  - writes data buffers out before backends need to
- Checkpointer
  - performs spread checkpoints, often near continuously
  - syncs files, other tasks

Why could postgres still succeed?

Why could postgres still succeed?

- linux has ok-ish readahead, page-cache
  - hides weaknesses of buffer replacement algorithm
- spinning disks are **SLOW** and not concurrent
- prefetching, parallelism hide costs / latency

Why O Why?

Why O Why?

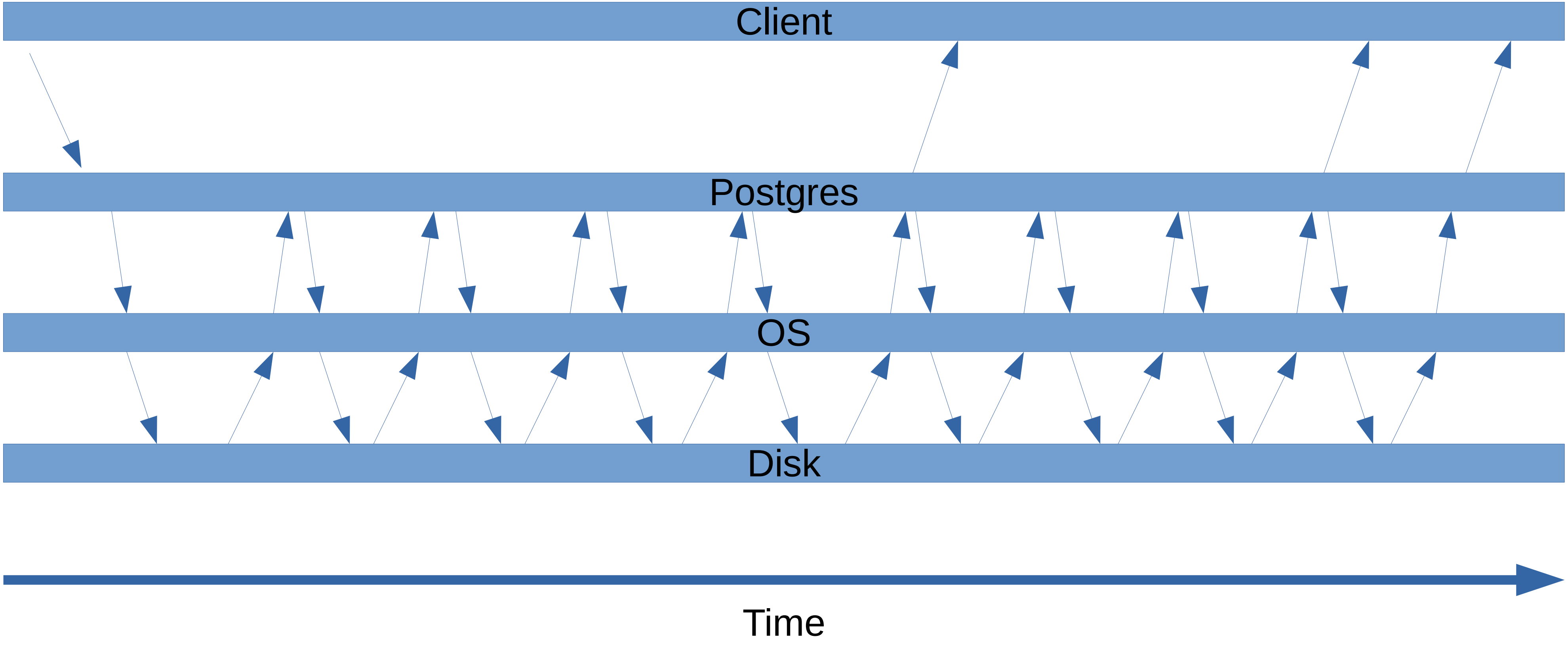
- Pragmatism: A small team
- Hard to change due to some architectural reasons
  - process based
  - cache replacement algorithm not great, tree-walk executor
- Direct IO alone is not usable
- AIO platform dependent, significant investments needed to be better than current state

In POSTGRES they are run as subprocesses managed by the POSTMASTER. A last aspect of our design concerns the operating system process structure. **Currently, POSTGRES runs as one process for each active user. This was done as an expedient to get a system operational as quickly as possible. We plan on converting POSTGRES to use lightweight processes available in the operating systems we are using**. These include PRESTO for the Sequent Symmetry and threads in Version 4 of Sun/OS.

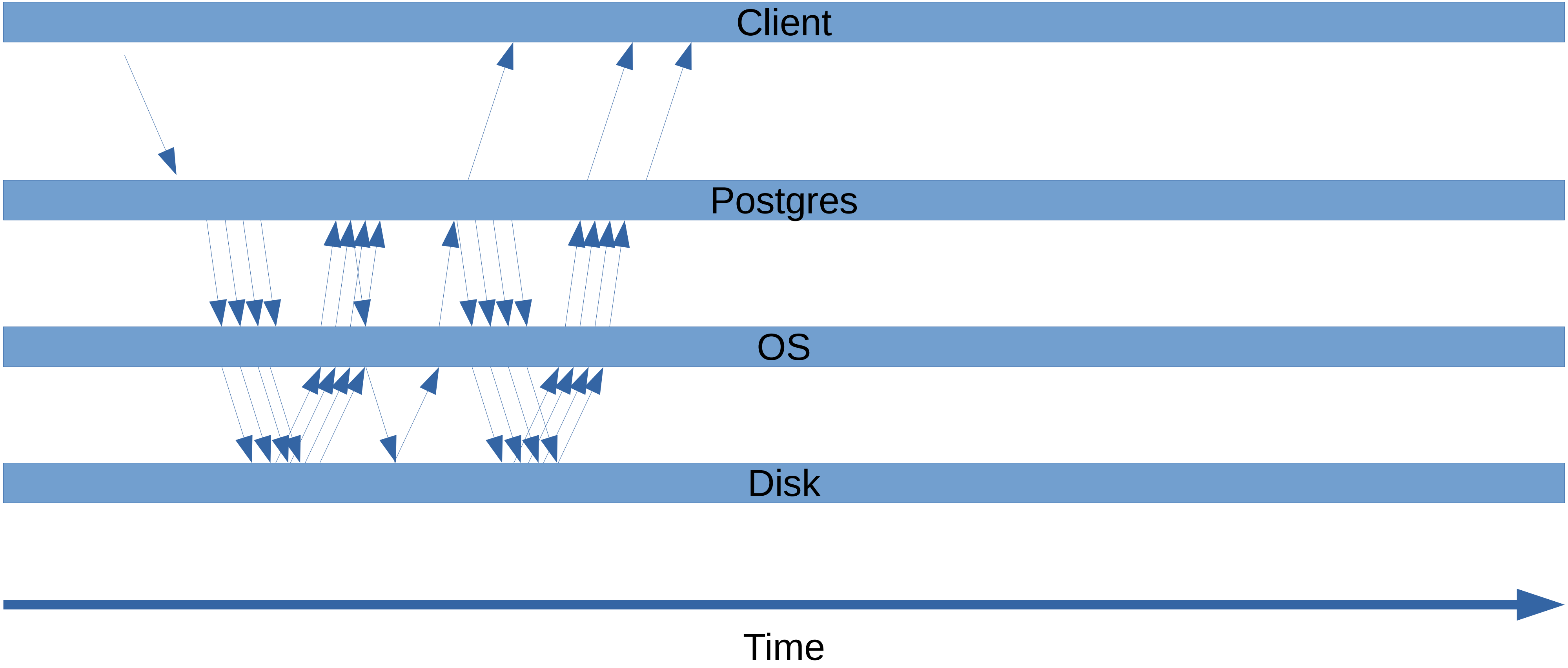
The Implementation of POSTGRES
Stonebraker, Michael and Rowe, Lawrence A. and Hirohama, M.
April 1990

(hat-tip to Thomas Munro for finding this quote)

Reads: synchronous, not cached



Reads: asynchronous, not cached



Why O Why?

- Lots of postgres use without elaborate tuning, not on dedicated machines, on overcommitted hardware
  - kernel management of page cache beneficial, AIO traditionally requires DIO
- Bad reasons: Believing our own excuses too much

Why Change?

Hardware Trend: NVMe

- “Specification for accessing storage medium
  - typically via PCIe, but also over network, fibre channel, etc
  - multiple queues
  - used for SSDs etc

- low dispatch overhead (single non-cacheable write)
  - latency increase due to kernel page cache significant
  - intra process / thread dispatch latency problematic
  - ~3 orders of magnitude, while memory latency improved ~1.5x

Hardware Trend: NVMe

- bandwidth often limited by PCIe 4x (~3GB/s for PCIe 3, ~7GB/s for PCIe4)
  - memory copy overhead due to copy of page cache a major limiting factor
  - kernel page cache allocation performance currently limiting factor
    - getting improved in linux
- very high IO concurrency
  - OS can't feed it from page cache / readahead

“Hardware” Trend: "Cloud" networked storage

- medium-high latency
  - 0.3ms for more expensive storage
  - 1-4ms for more commonly used storage, higher for spinning disk backed storage)
- but lots of IO concurrency and random access characteristics
  - lots of IO need to be in flight for decent performance - hard to do via buffered IO
  - WAL write latency a major issue
- decent bandwidth

Moving towards AIO / DIO

AIO in PG

- I started working on it 2019, with lots of other project interrupting it
  - long interested in topic, re-invigorated due to io_uring supporting buffered AIO
- since then have had help, most prominently by Thomas Munro and Melanie Plageman

AIO in PG: Design Constraints

- Process Model
- Avoid deadlocks due to processes that submitted IO blocking on lock, holder of lock blocked on completing IO
- Avoid intra-process context switches for IOs when possible
- OS / IO method abstraction from most code
- Individual AIO users shouldn’t need to know much about AIO

AIO in PG: Architecture

- AIO completions can be processed in any process
- base AIO layer doesn't know about buffers etc
- IO combining etc done in AIO layer, using scatter / gather IO
- lots of boring infrastructure improvements, some committed

- “Accurate” readahead, via "streaming read" interface
  - requires just a callback providing details about IO needed in future

AIO in PG: Backends

- io_uring
  - IO submission / completion handling doesn't require intra-process context switches, even syscalls can be optimized away
  - supports AIO with buffered IO
- "worker" fallback
  - crucial so we don’t need AIO and non-AIO paths
- posix AIO (ugh)
- windows iocp (some issues remain)

AIO in PG: Converted Subystems

- readahead during WAL replay
  - potentially *disastrous* recovery performance when not using DIO otherwise
- checkpointer (data file writes, concurrent file flushes), bgwriter
- WAL writes: multiple in-flight writes, potentially using using O_DSYNC
- vacuum (heap, indexes)
- sequential scans, bitmap index scans
- asynchronous writeback by backends
- file extensions

AIO in PG: Result, Good

- checkpointing at disk limits (~12GB/s in my workstation, unfortunately PCIe3)
- sequential scans up to 2.5x faster per core, scaling much better to larger amounts of memory
- concurrent COPY much faster
- concurrent write-heavy OLTP much faster due to concurrent flushes **IF** concurrent enough, or padding of WAL records enabled
  - padding can cause vast increase in WAL volume
- …

AIO in PG: Result, Ugly

- file extension with DIO still leads to fragmented files
- too many paths still not optimized for DIO
- implementation of asynchronous writeback by backends is, uh, not great
- slowdowns with io_uring in edge-cases
  - single process vs multiple kernel threads doing work; use heuristics?
- Buffer replacement weaknesses show

- Non-AIO bottlenecks preventing bigger gains
  - "SLRUs" for transaction status
  - memory access patterns in VACUUM

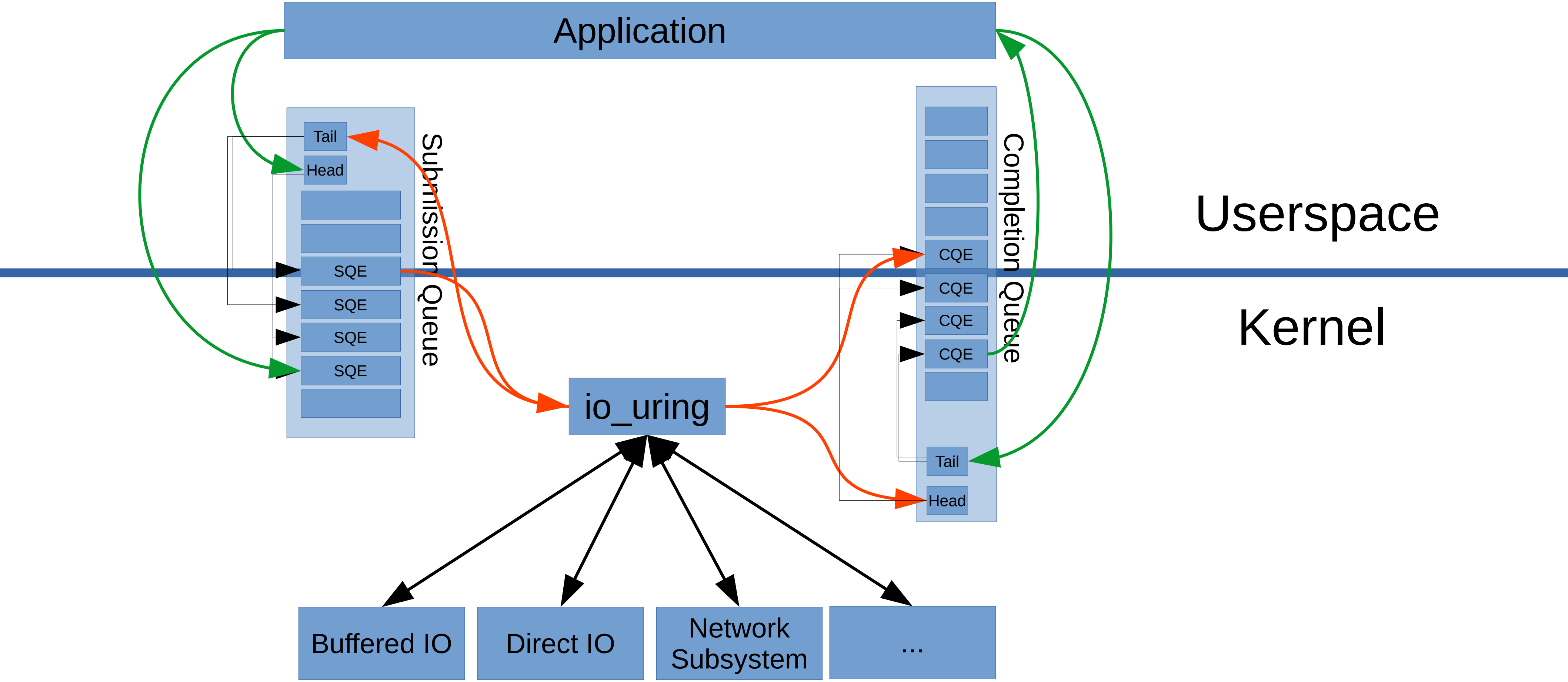
AIO in PG: Next Steps

- Polish, polish, polish
- Starting merging prerequisites into PG 16
- lots of OS specific improvements, collaboration with kernel folks
  - FreeBSD (avoid kernel thread, IO completion processing, DMA improvements)
  - Linux (improve buffered IO perf, general perf work)
  - Windows (evaluate IO uring equivalent API)

AIO in PG: Bigger TODOs

- Algorithm for adaptive prefetch distance / concurrency
  - working on a simulator for playing with algorithms
- Executor (& Planner) improvements
  - most importantly: non-bitmap index scans
- Improve / Switch Buffer Replacement Algorithm
- Heuristic prefetching needed?
  - nice experimental results with reading in neighboring pages
- Write out neighboring pages in background writer (backends?

io_uring basics



Clock-Sweep