The path to using AIO in Postgres

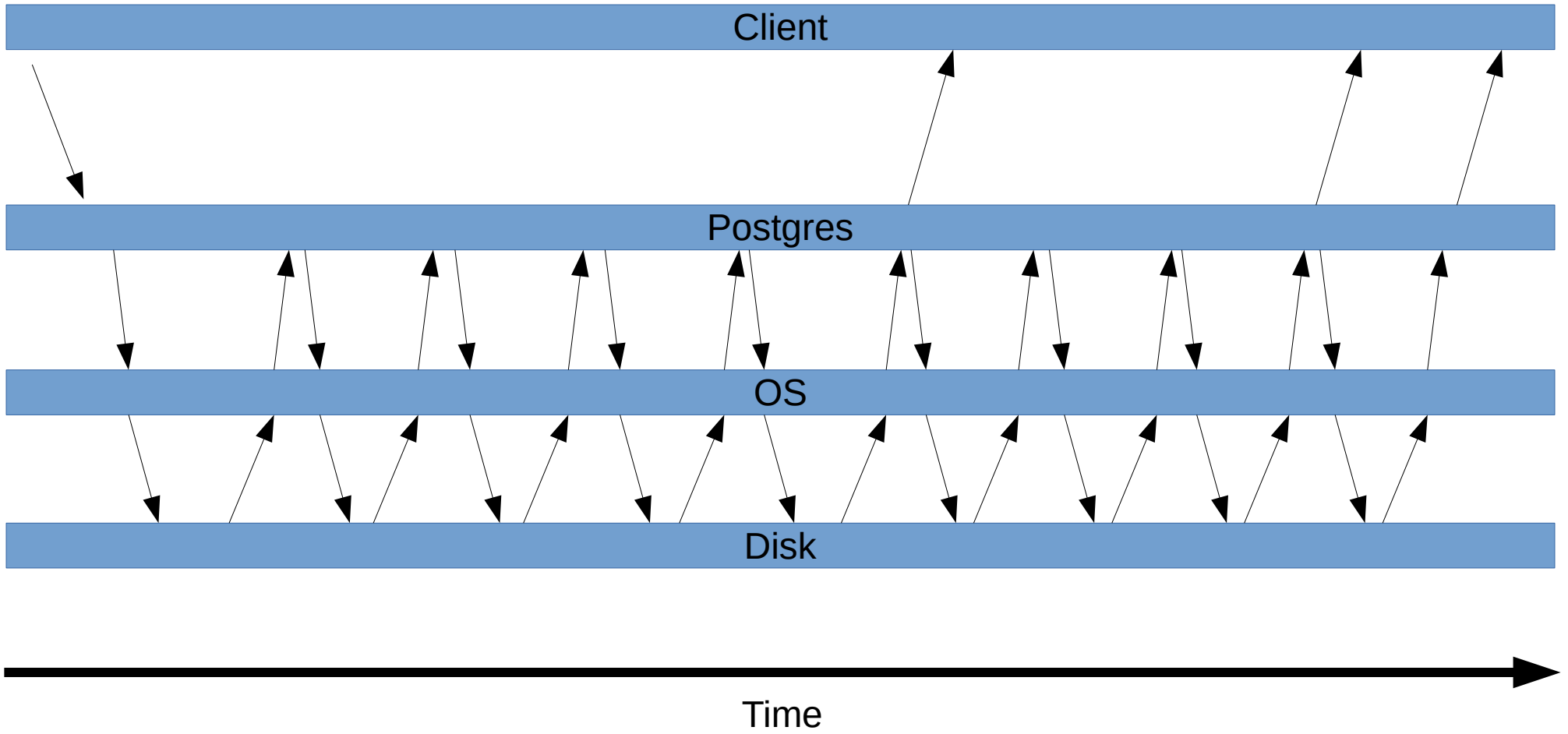
Andres Freund
PostgreSQL Developer & Committer

Email: andres@anarazel.de
Email: andres.freund@microsoft.com
anarazel.de/talks/2023-10-04-pgconf-nyc-path-to-aio/path-to-aio.pdf

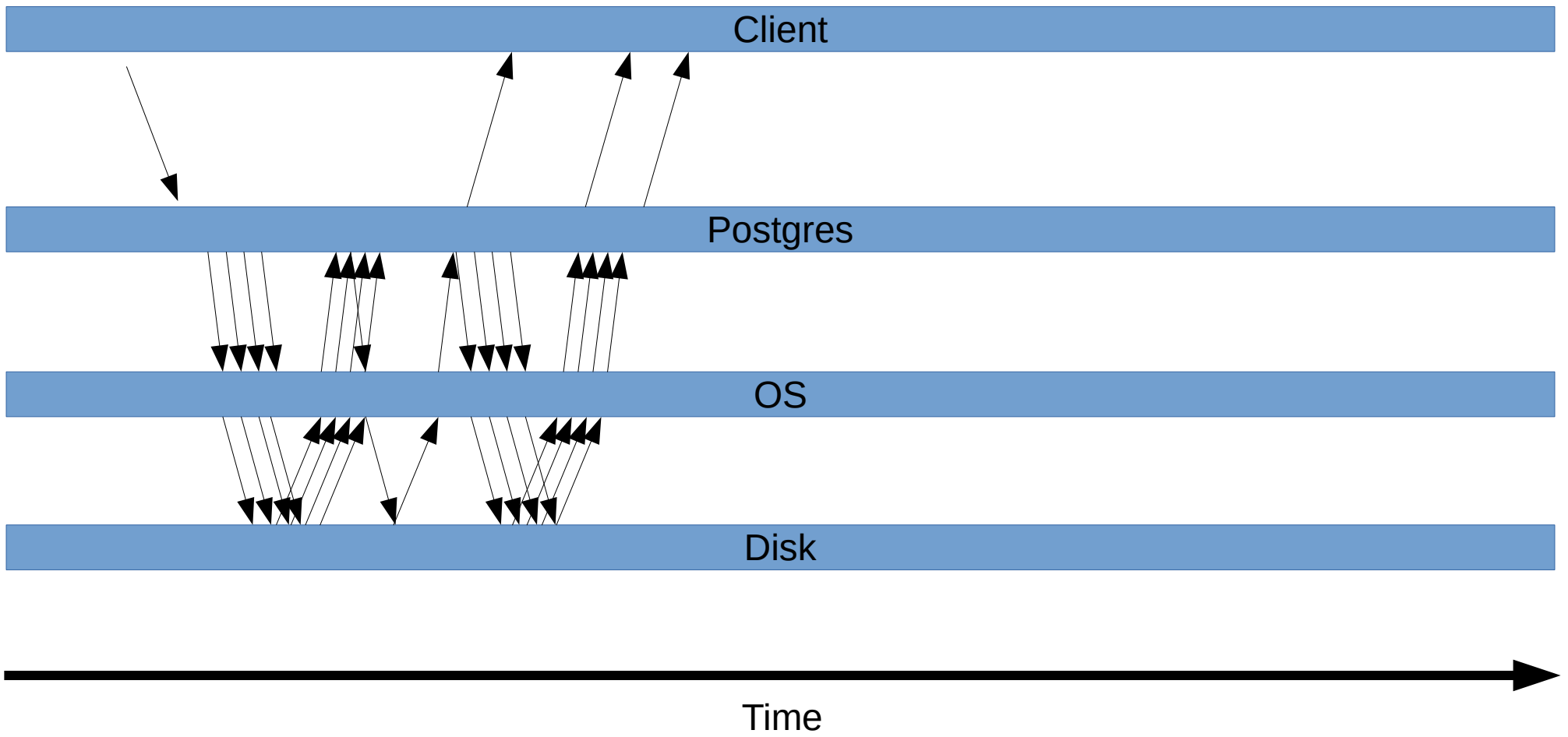
It’s Long

It’s Hard

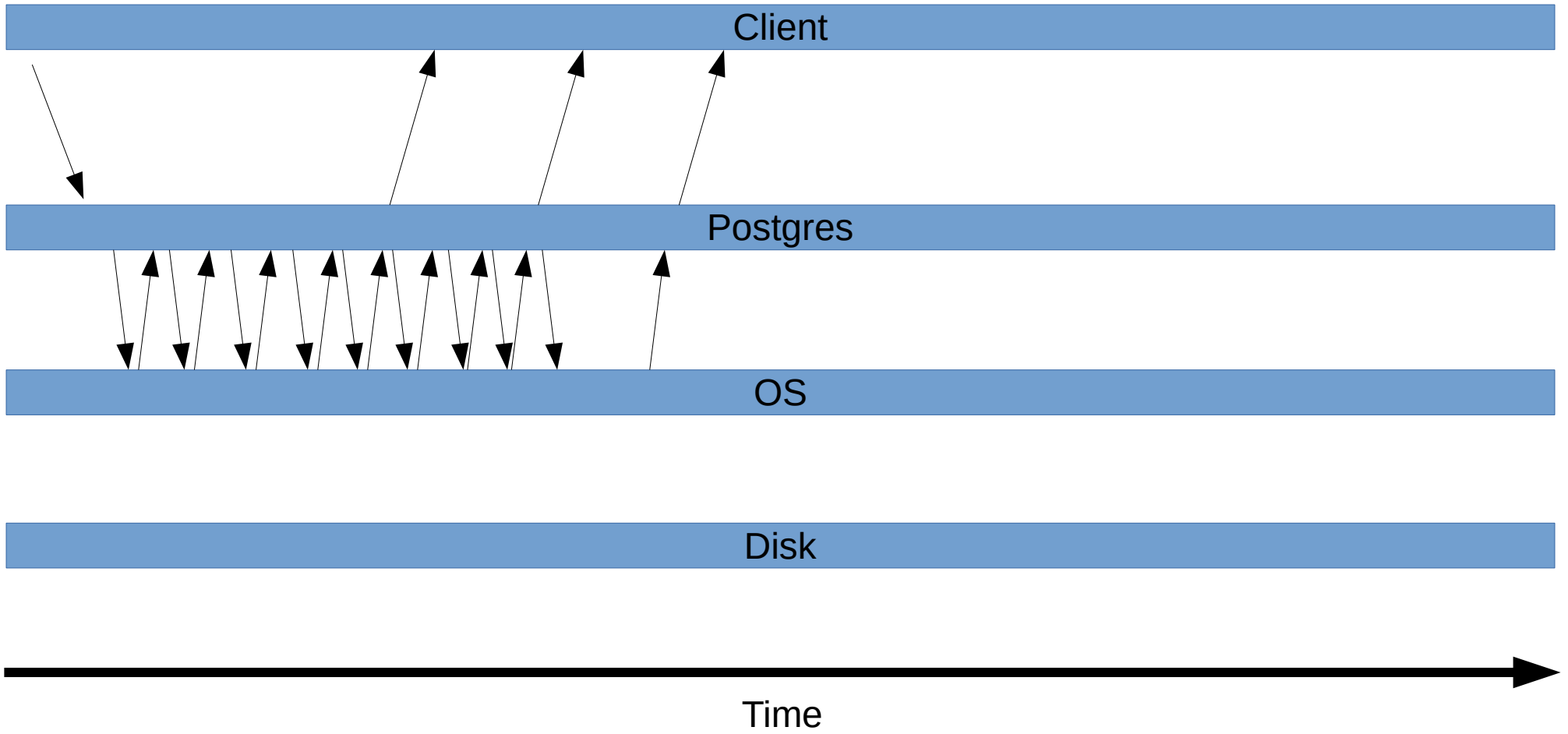
Reads: synchronous, not cached



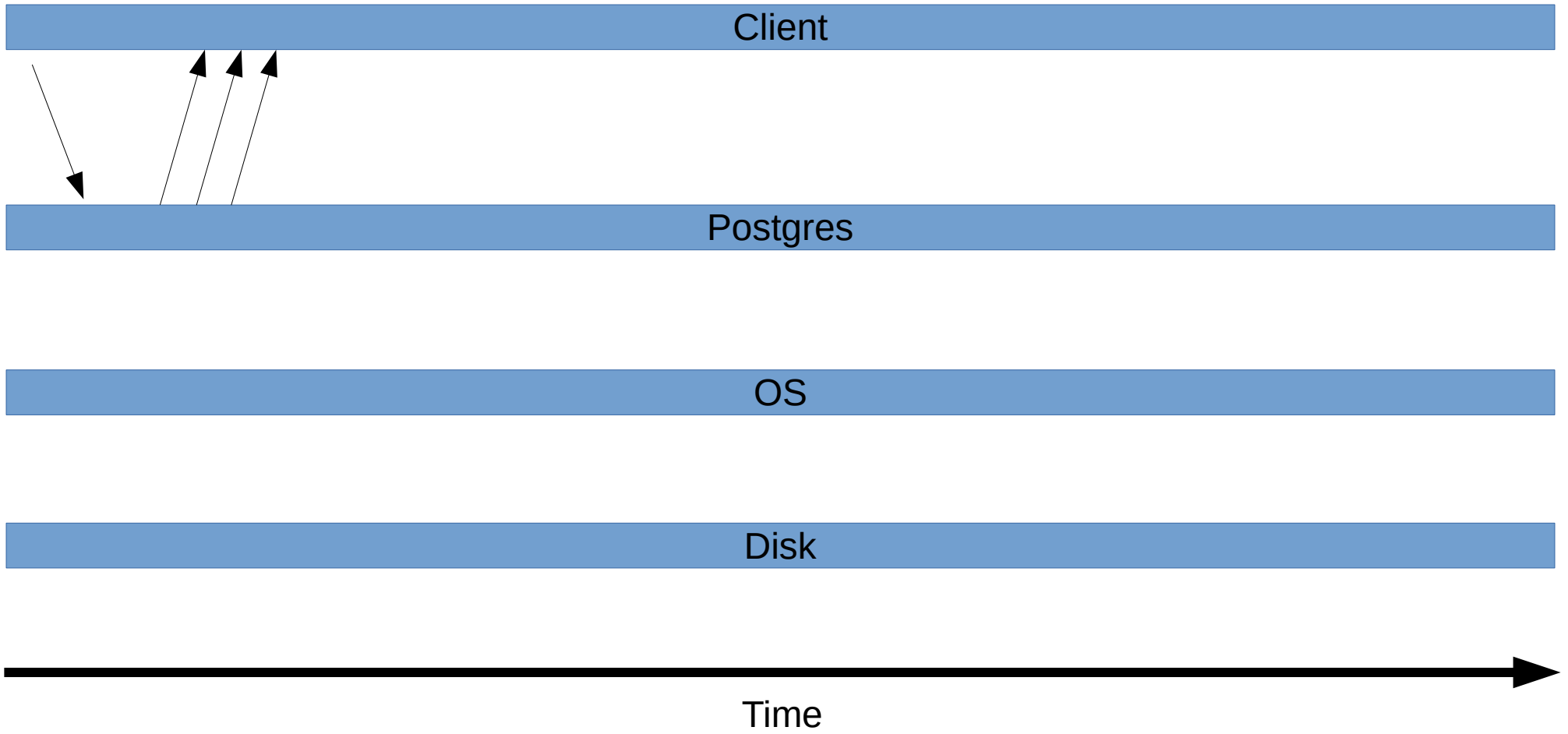
Reads: asynchronous, not cached



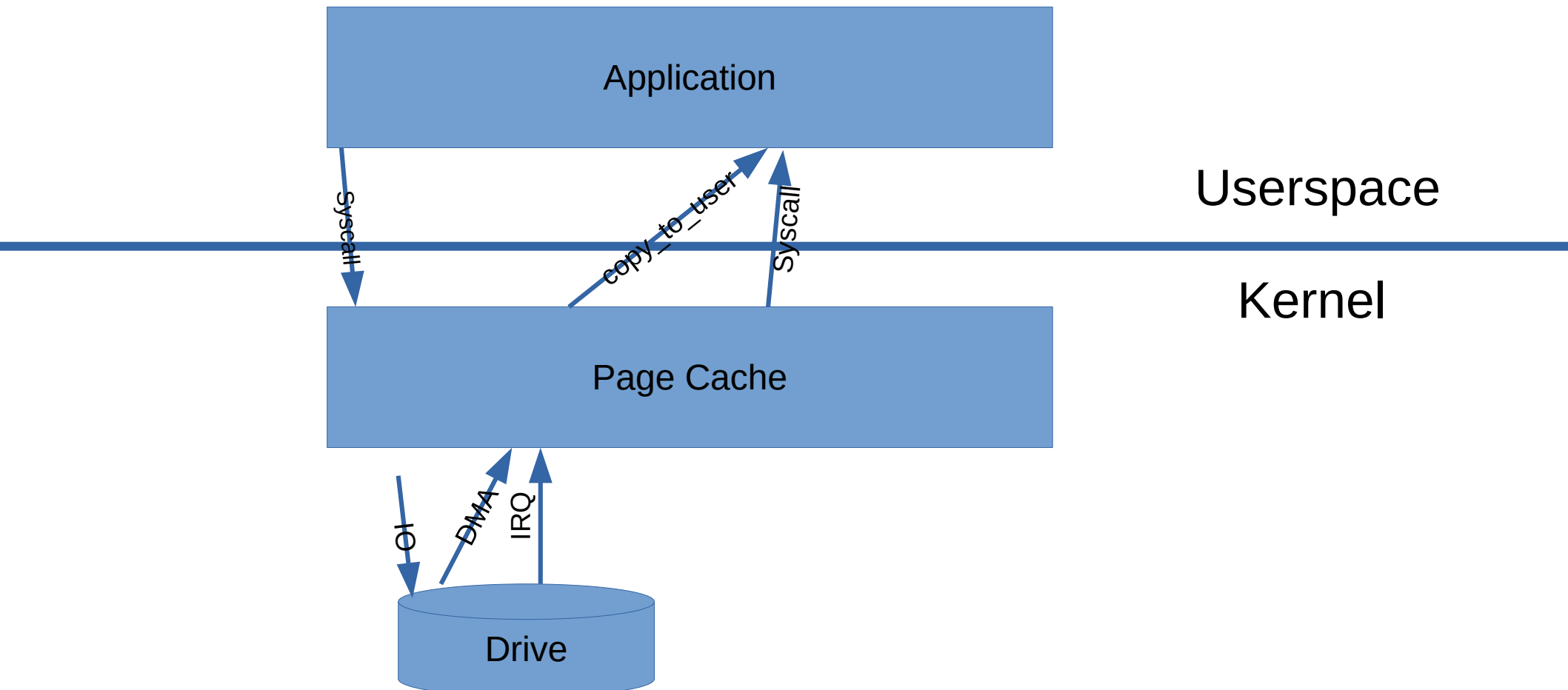
Reads: synchronous, OS cached



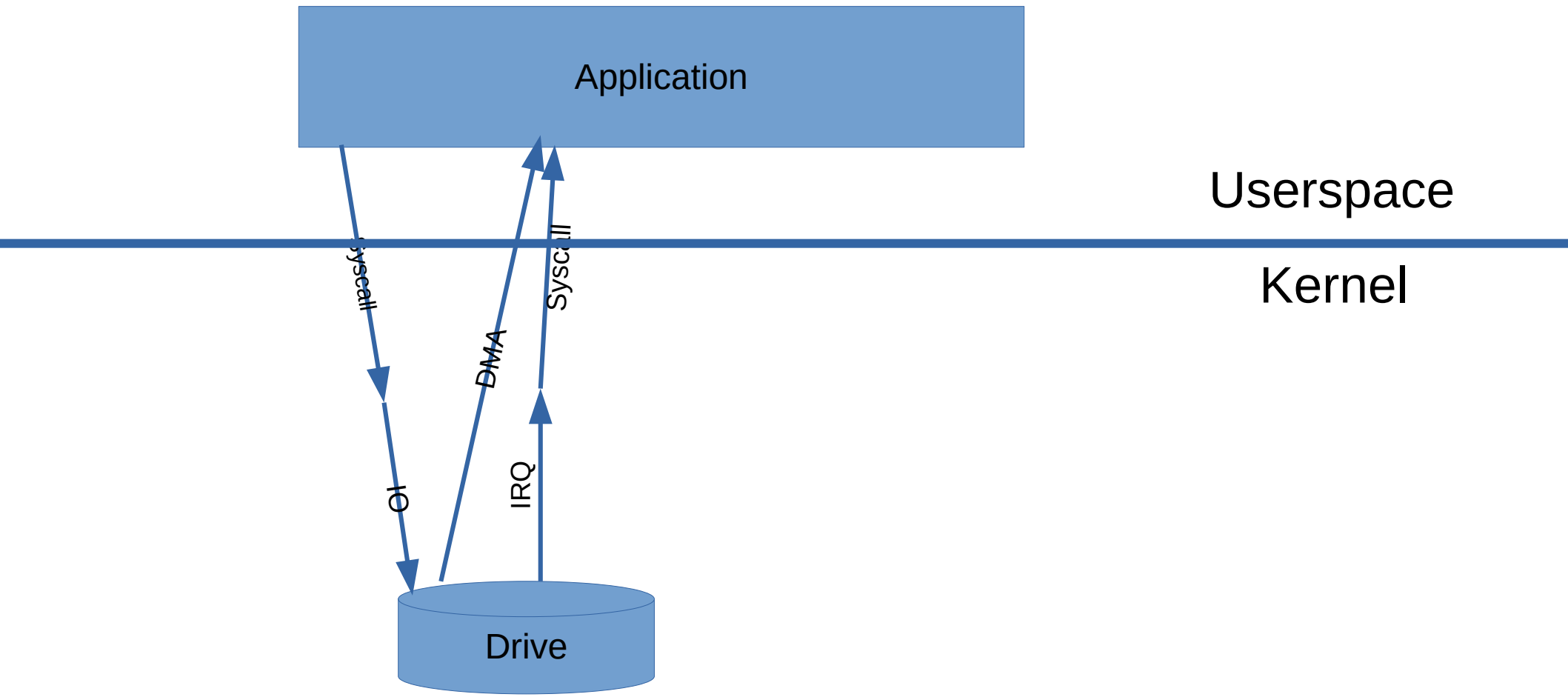
Reads: synchronous, postgres cached



Buffered read()

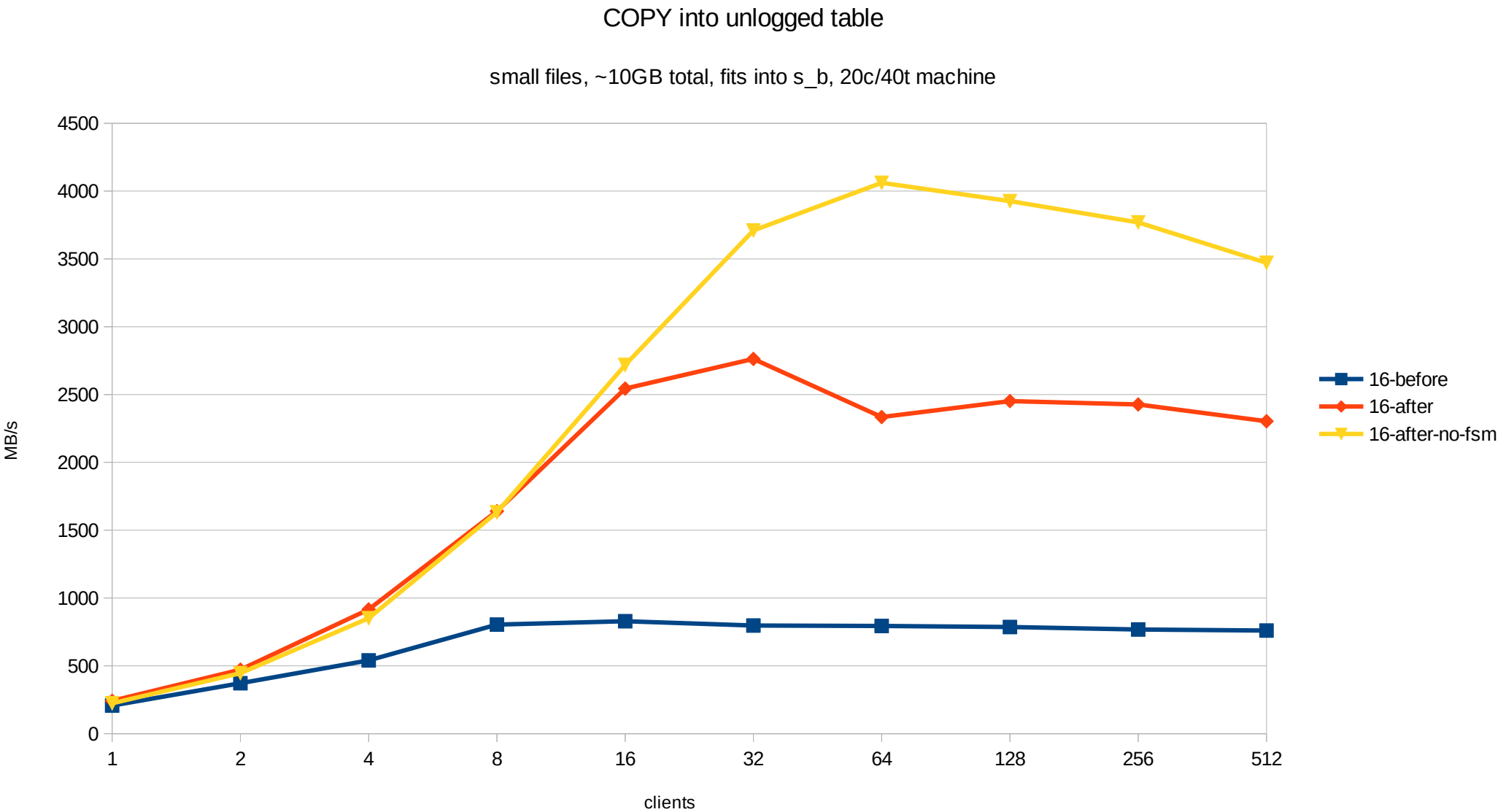


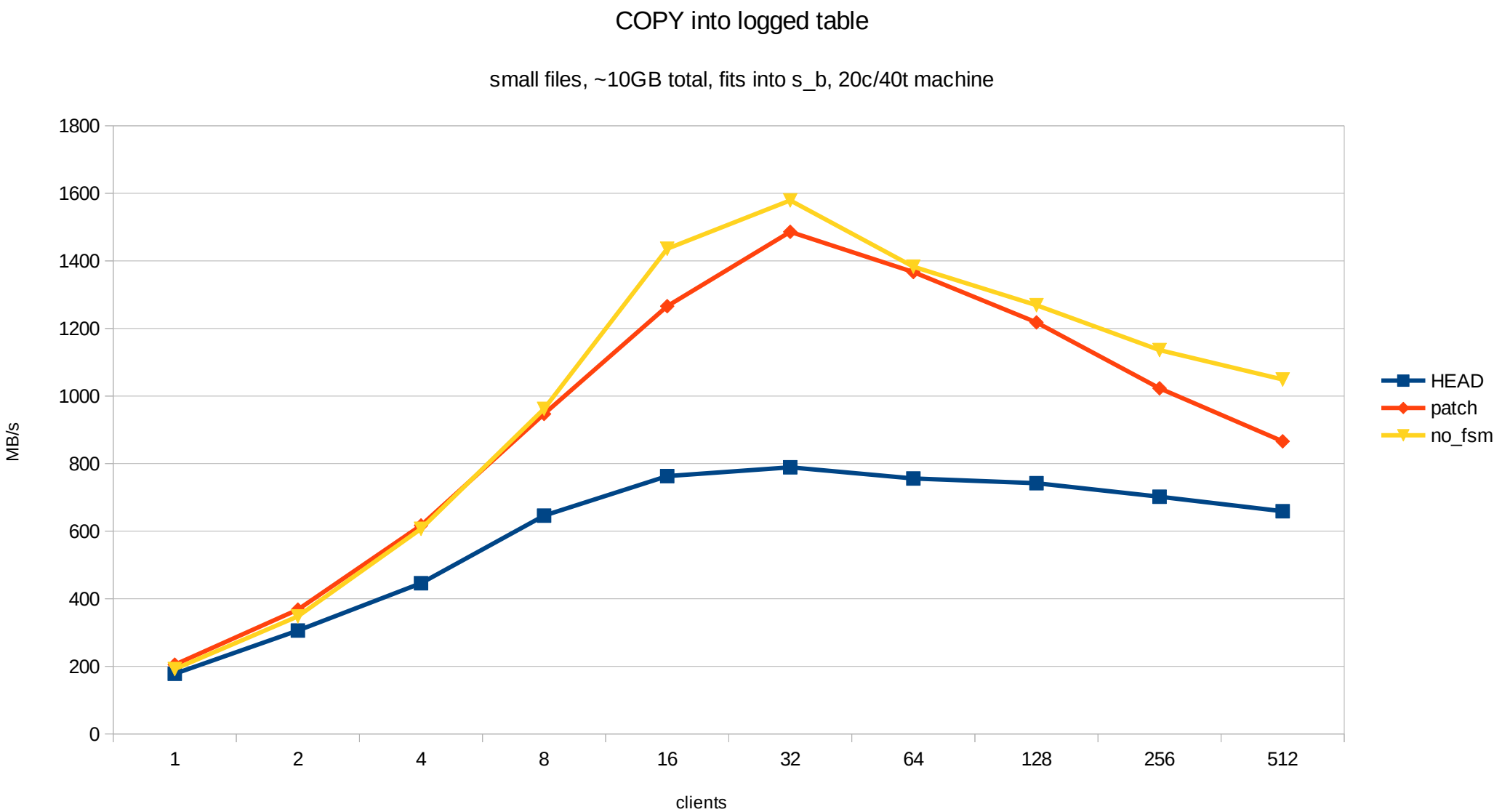
Direct IO (DIO) read()



16: Bulk Relation Extension, Buffer Replacement

- Infrastructure for multiple “in progress” Buffers
- Buffer Replacement – “Get a free buffer when there are no free ones”
- Relation Extension - “Making a table bigger”

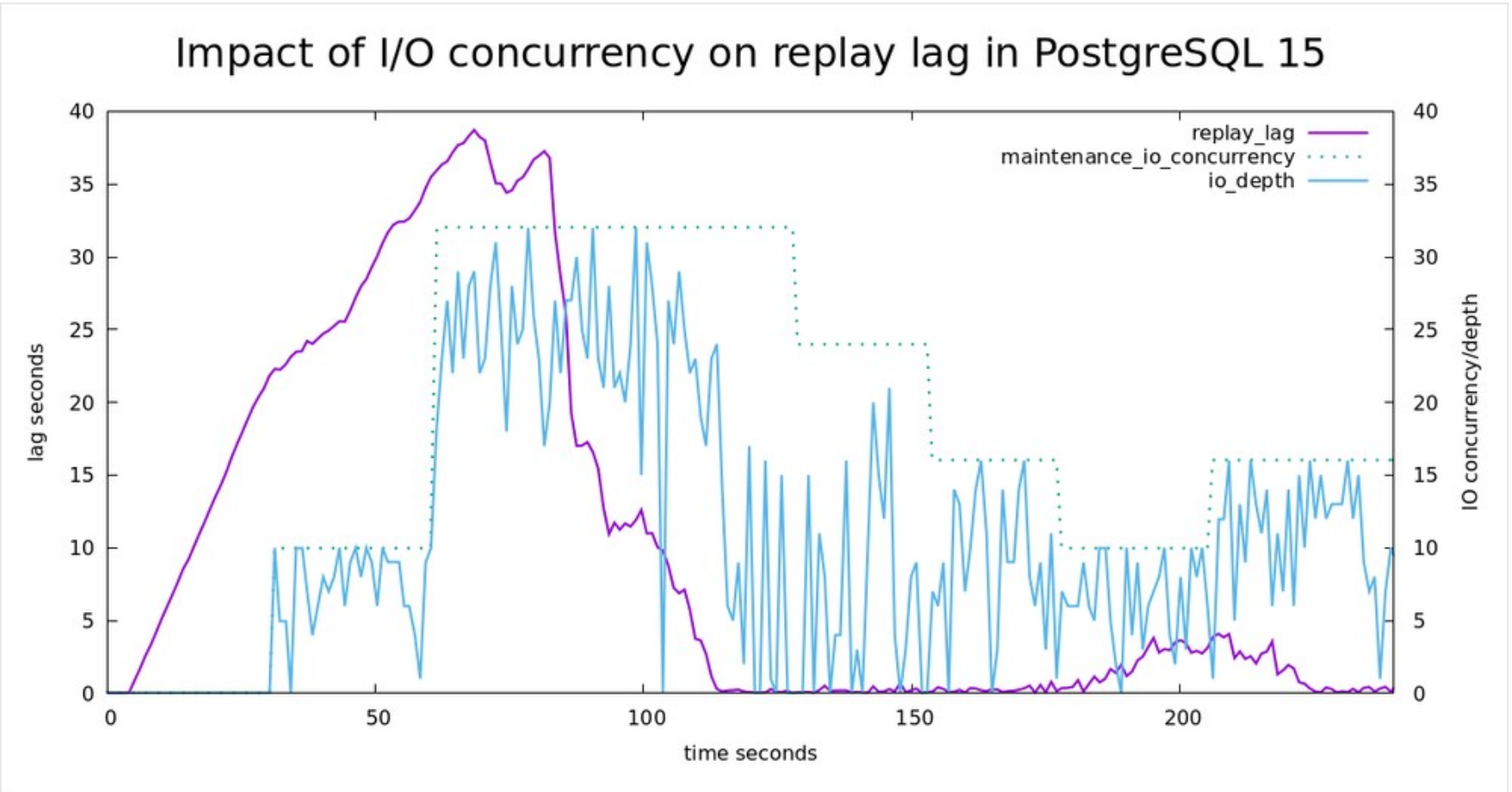

COPY into unlogged table
small files, ~10GB total, fits into s_b, 20c/40t machine
MB/s
4500
4000
3500
3000
2500
2000
1500
1000
500
0
1
2
4
8
16
32
64
128
256
512
clients
16-before
16-after
16-after-no-fsm


COPY into logged table
small files, ~10GB total, fits into s_b, 20c/40t machine
MB/s
1800
1600
1400
1200
1000
800
600
400
200
0
1
2
4
8
16
32
64
128
256
512
clients
HEAD
patch
no_fsm

16: “Add debug_io_direct setting for developer usage”

- Can be set to data, wal, wal_init
- NOT RECOMMENDED FOR PRODUCTION
- “data”
  - relation IO
  - is disastrous for performance
- “wal”
  - can already show benefits, particularly with wal_sync_method=open_datasync
- “wal_init”
  - creation of new WAL files

15: Recovery Prefetching (Thomas Munro)



https://techcommunity.microsoft.com/t5/azure-database-for-postgresql/reducing-replication-lag-with-io-concurrency-in-postgres-15/ba-p/3673169

17?: Streaming Read Abstraction

- Simple interface for most read IO
- No AIO, just “fadvise” style prefetching
- Main Goal: Parallelize development
- Minor Goal: Small performance gains
- Convert some users to new interface
- See

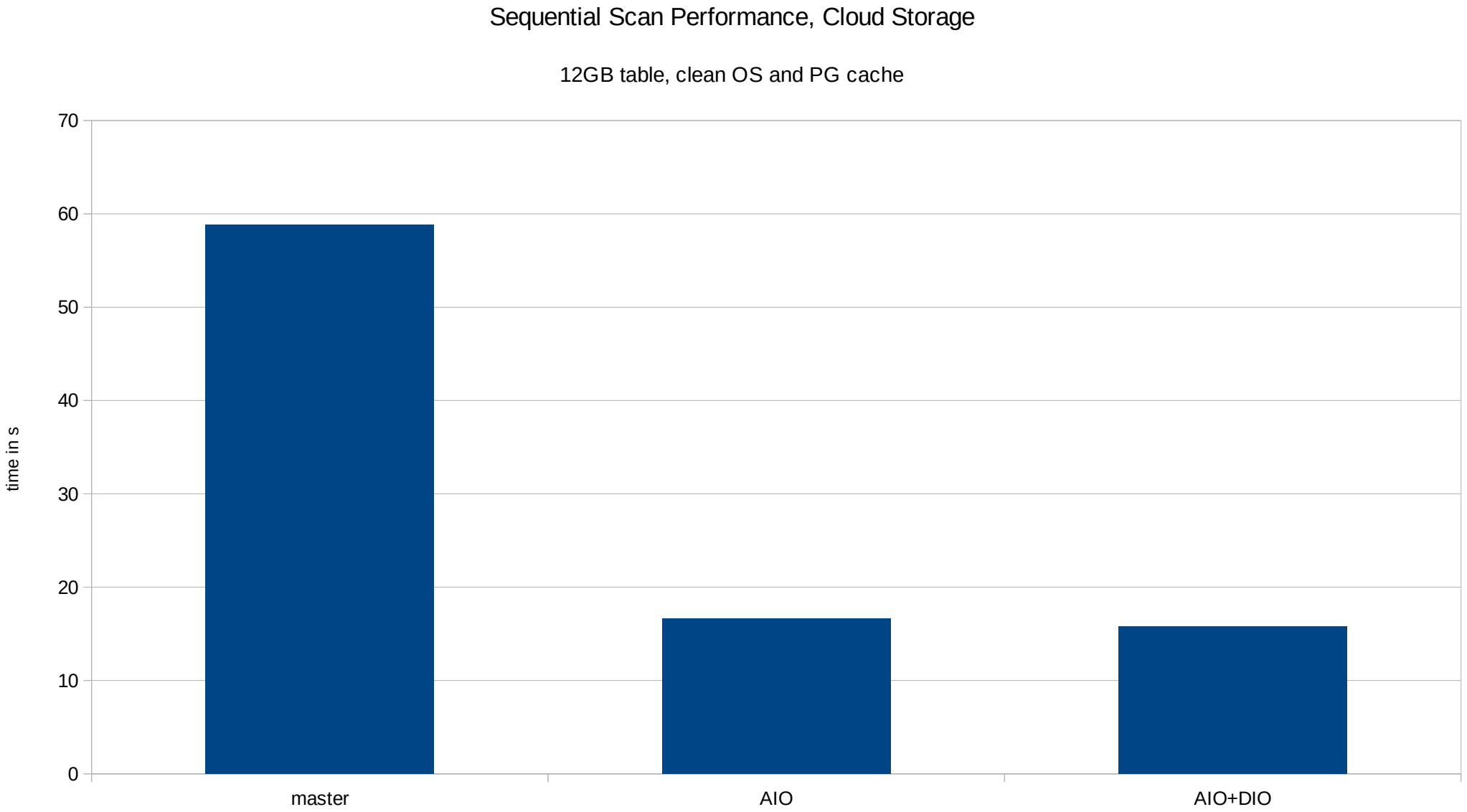
https://postgr.es/m/CA+hUKGJkOiOCa+mag4BF+zHo7qo=o9CFheB8=g6uT5TUm2gkvA@mail.gmail.com

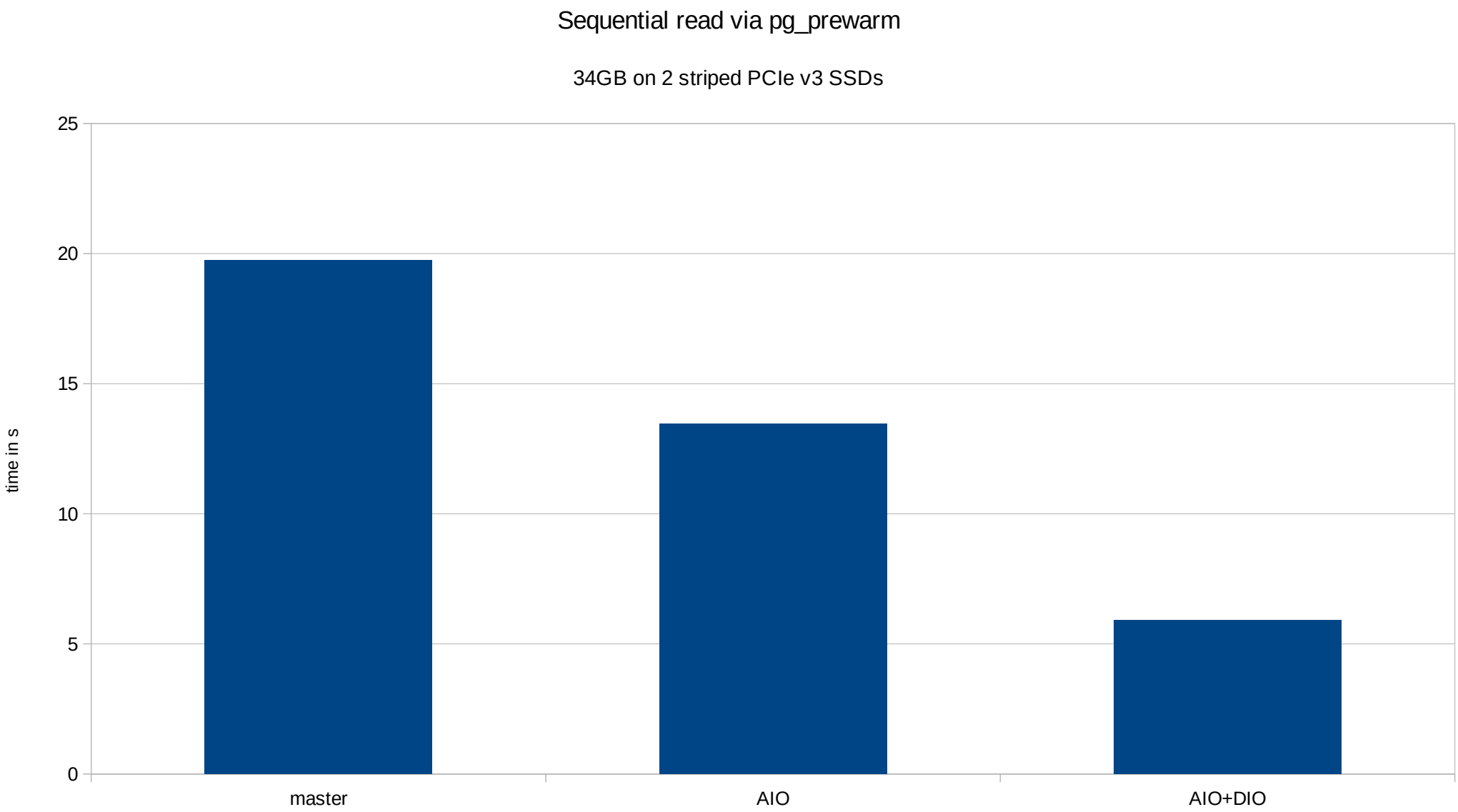
17 ??: AIO infrastructure

- io_method=(worker|io_uring|posix_aio)
- goal: can use AIO infrastructure without loss of performance, even when no AIO support present, to avoid duplicating code
- Not yet used

17 ?: sequential scans

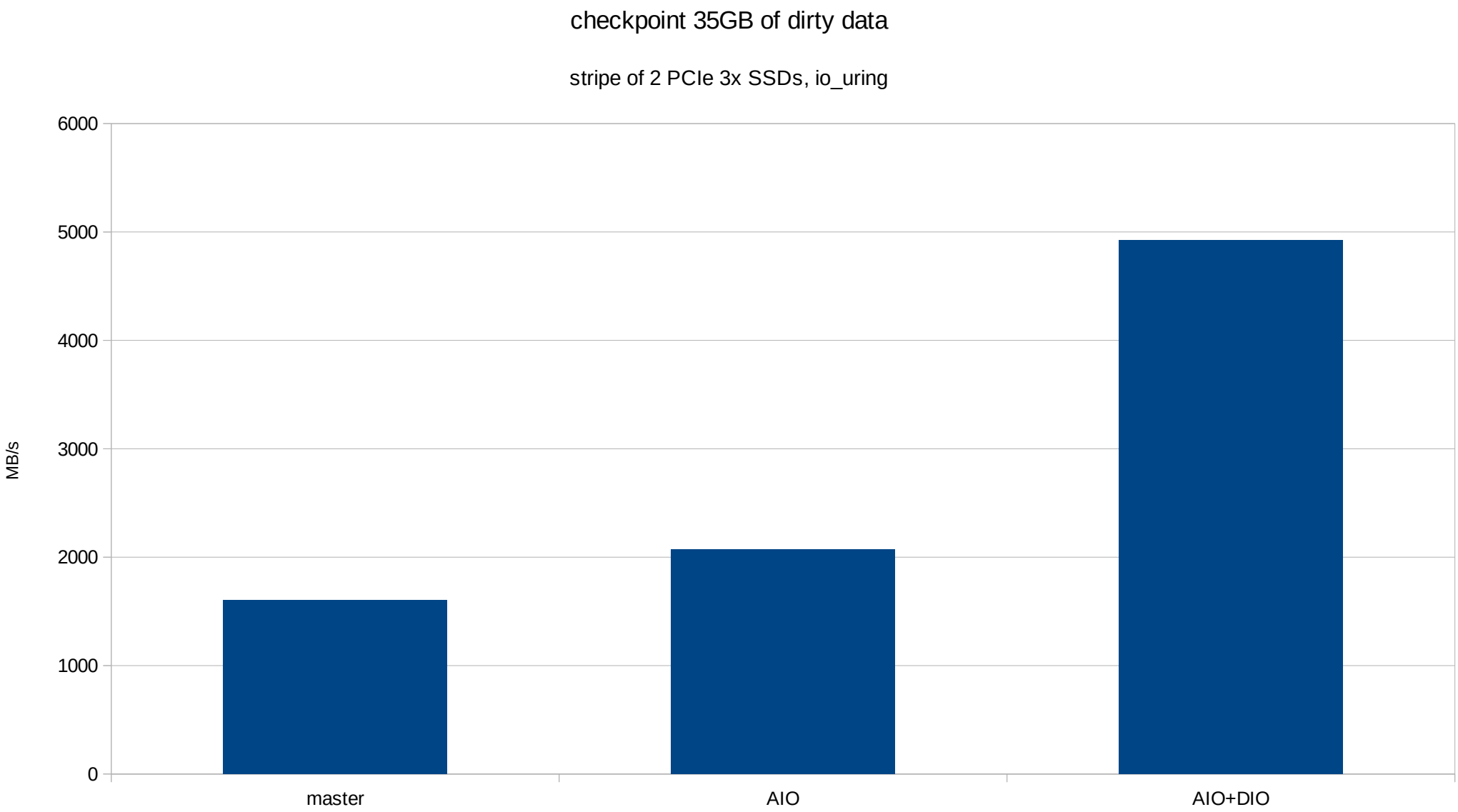
- Problem:
  - Only use OS readahead
  - Double Buffering
  - Can get confused (skipped blocks, segments)
  - Not guaranteed to be present
  - OS readahead doesn't know workload / not aggressive enough
  - Simple patch due to “streaming read” interface


Sequential Scan Performance, Cloud Storage
12GB table, clean OS and PG cache
time in s
70
60
50
40
30
20
10
0
master
AIO
AIO+DIO


Sequential read via pg_prewarm
34GB on 2 striped PCIe v3 SSDs
time in s
25
20
15
10
5
0
master
AIO
AIO+DIO

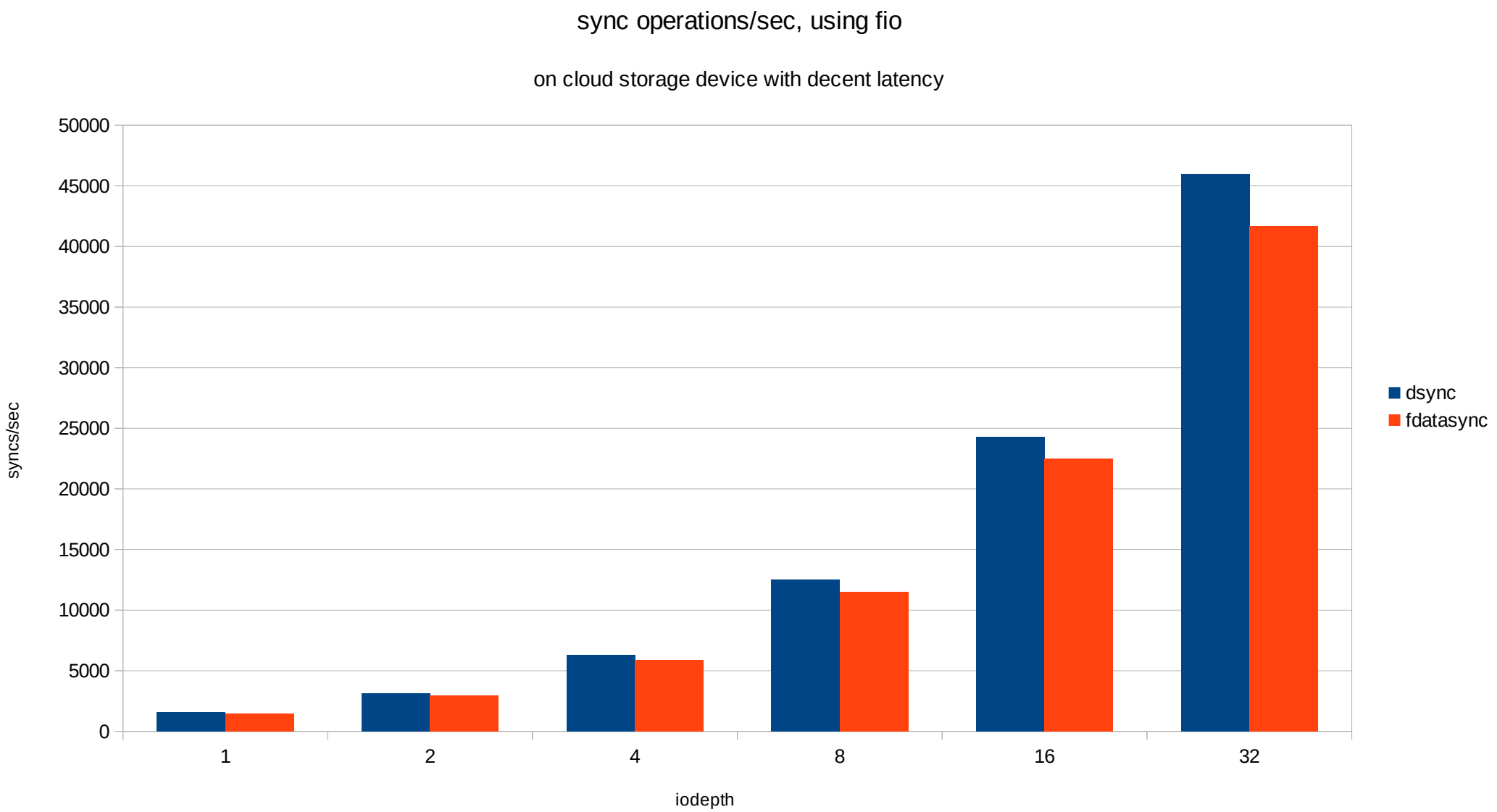
17??: checkpointer, bgwriter

- Throughput limited due to CPU overhead
- Limited control over latency impact with buffered IO


checkpoint 35GB of dirty data
stripe of 2 PCIe 3x SSDs, io_uring
MB/s
6000
5000
4000
3000
2000
1000
0
master
AIO
AIO+DIO

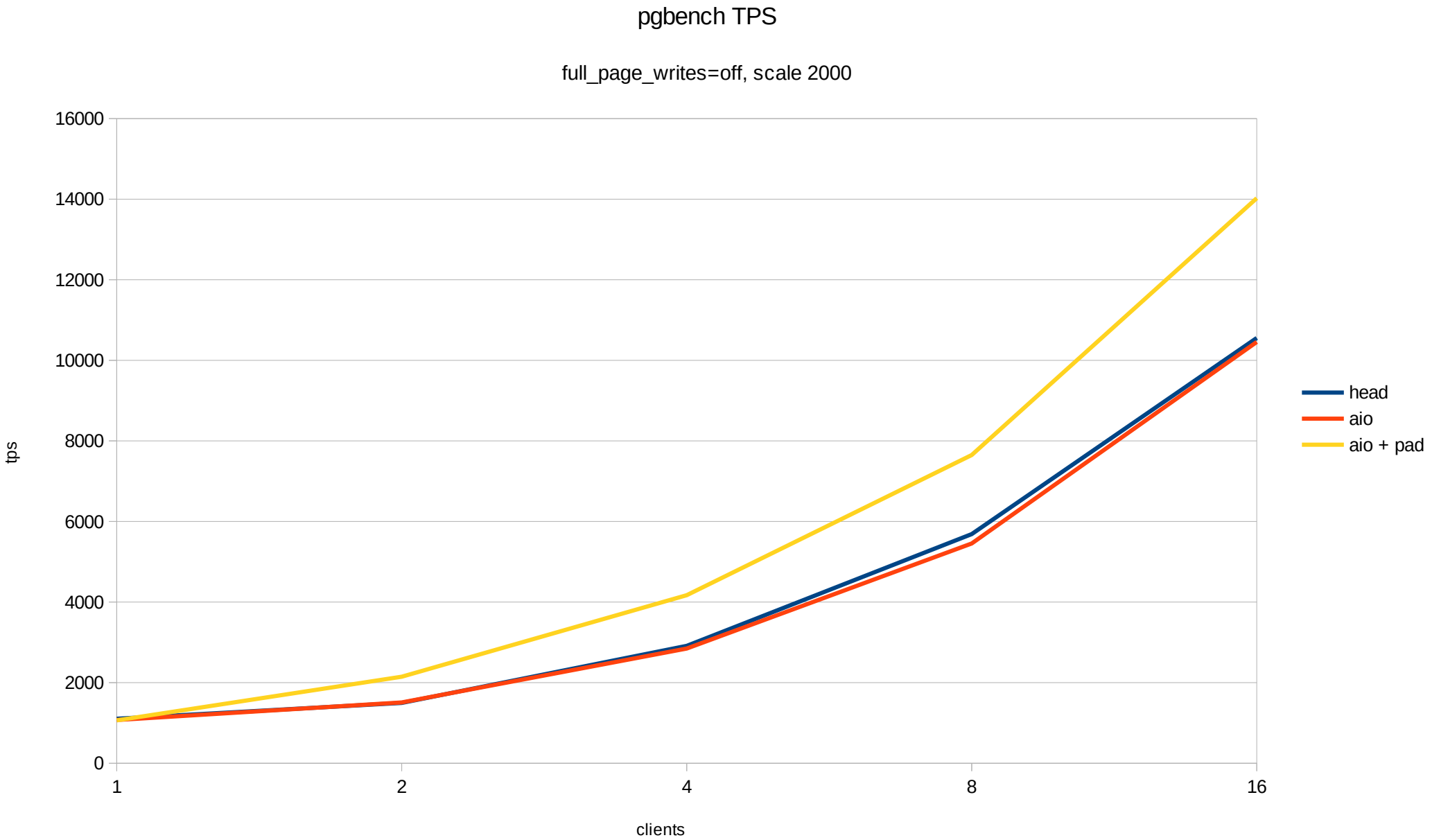
18?: WAL writes

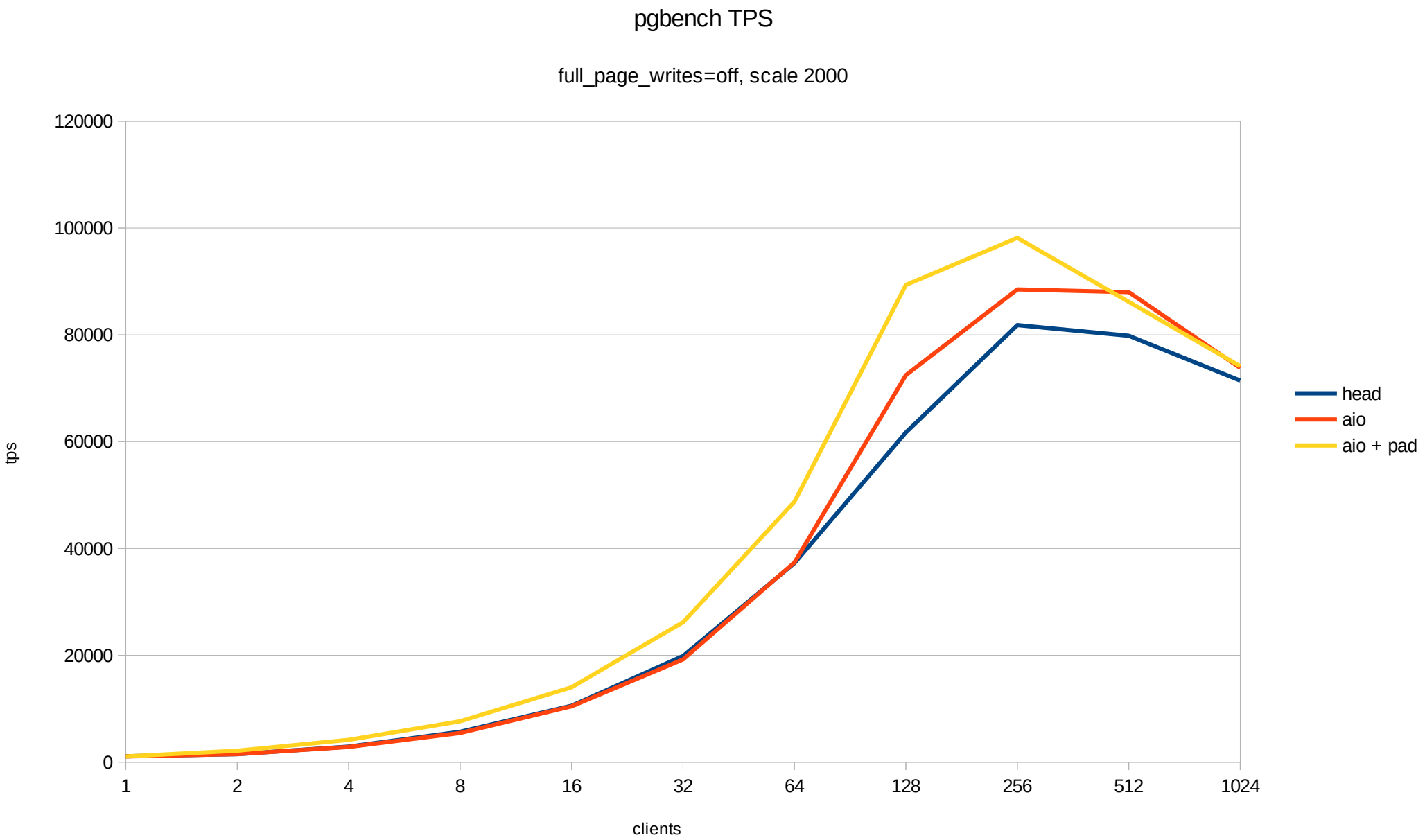
- benefit #1: Do something else during WAL write / flush
  - could get rid of bgwriter (often overloaded, not adaptive)
- benefit #2: Multiple WAL flushes concurrently
  - we have group commit
  - but only one flush in progress
- Hard, gains only very partially realized right now


sync operations/sec, using fio
on cloud storage device with decent latency
50000
45000
40000
35000
30000
25000
20000
15000
10000
5000
0
syncs/sec
1
2
4
8
16
32
iodepth
dsync
fdatasync

18?: WAL writes

- pgbench transaction: ~450 bytes
- default WAL page size: 8kB
- in-write page cannot be written again before completion
- Problem: to-be-flushed-page is rarely full
- Solution (?): optionally pad partial pages


pgbench TPS
full_page_writes=off, scale 2000
16000
14000
12000
10000
8000
6000
4000
2000
0
tps
1
2
4
8
16
clients
head
aio
aio + pad


pgbench TPS
full_page_writes=off, scale 2000
120000
100000
80000
60000
40000
20000
0
tps
1
2
4
8
16
32
64
128
256
512
1024
clients
head
aio
aio + pad

17?/18?: VACUUM

- improved read performance
- improved *write* performance
  - due to asynchronous WAL flushing
  - due to DIO
- better control over latency effects

	high lat cloud disk	lower lat cloud disk
master	94.673 s	33.37 s
aio	12.349 s	7.737 s

17, 18: Other working AIO conversions

- SyncDataDirectory()
- Bitmap heap scans

Potential future AIO users

- various index scans
  - Tomas Vondra is working on some bits
- More vacuuming
- Temp table support
- More everything
- lower-level operations
  - create database
  - vacuum full
  - on startup cleanups
  - filesystem directory iteration

Thanks!

- Colleagues working with me on this
  - David, Melanie, Thomas and others
- Others working on related important pieces
  - Tomas Vondra is working index prefetching
  - Bharath Rupireddy is working feelding walsender from buffers
  - ...
- github.com/anarazel/postgres/tree/aio
- http://wiki.postgresql.org/wiki/AIO