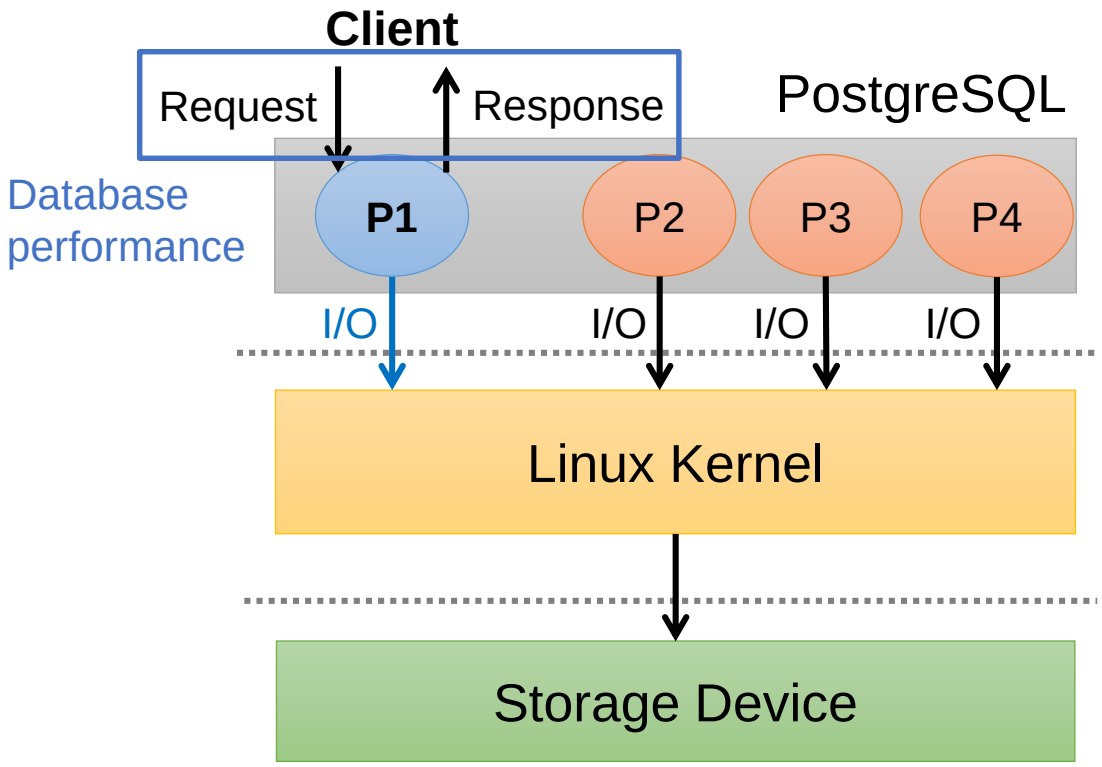
Taming Performance Variability in PostgreSQL

Shawn S. Kim

appOsha

PostgreSQL Execution Model

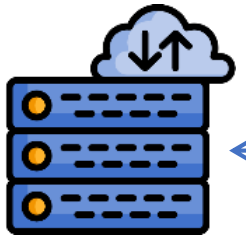


*** PostgreSQL's processes**

- **Backend (foreground)**
- **Checkpointer**
- **Autovacuum workers**
- **WAL writer**
- **Writer**
- **…**

Impact of Background Tasks

SysBench 1.0.15
--oltp-table-size=10000000
--oltp-tables-count=24
(50GB dataset)



r5.xlarge
4 vCPUs
32GiB memory

r5d.4xlarge
16 vCPUs
128GiB memory

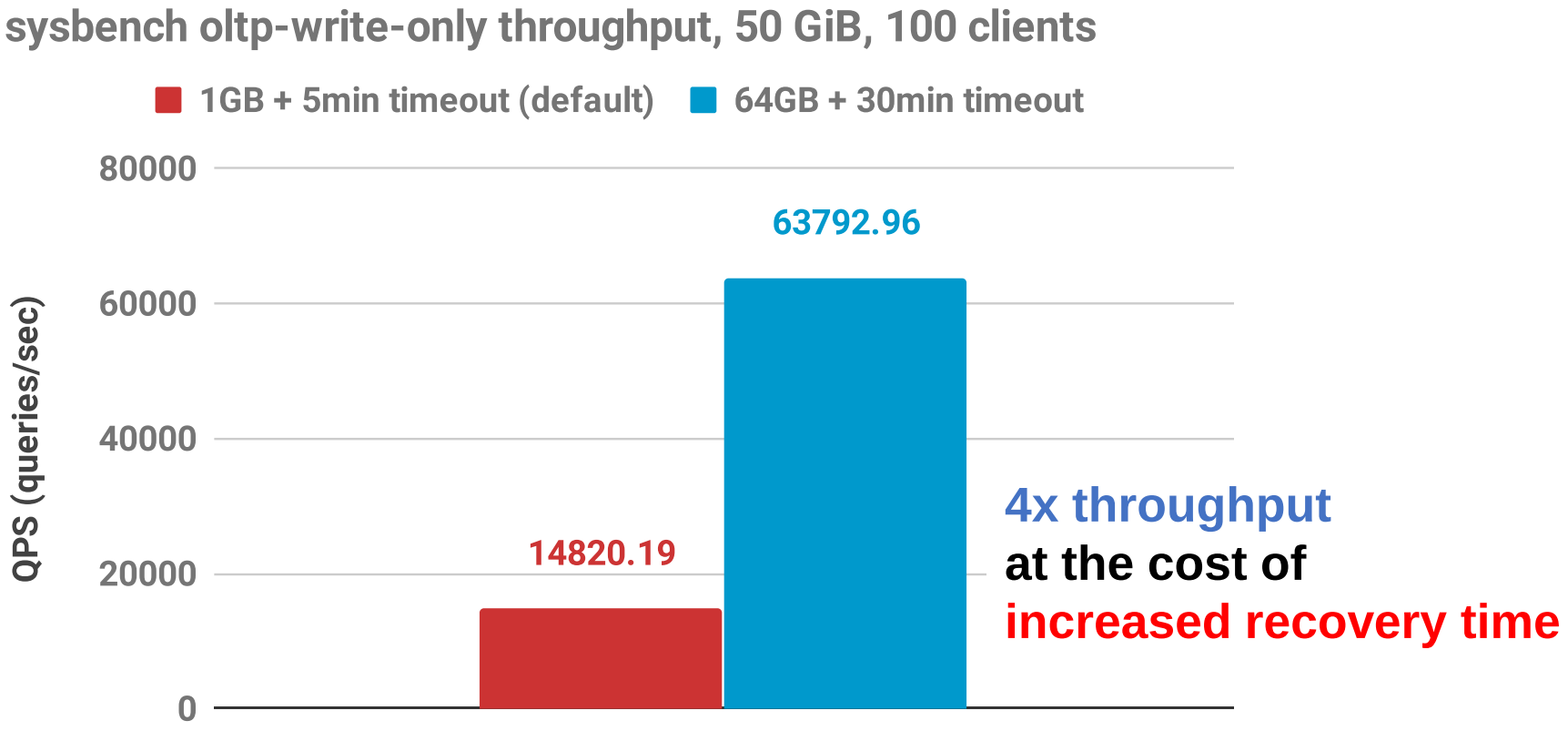
AWS Seoul region

PostgreSQL 11.3
32GB shared_buffer
64GB effective_cache_size
…

Local NVMe SSD

Impact of Background Tasks

Checkpoint tuning* improves average throughput

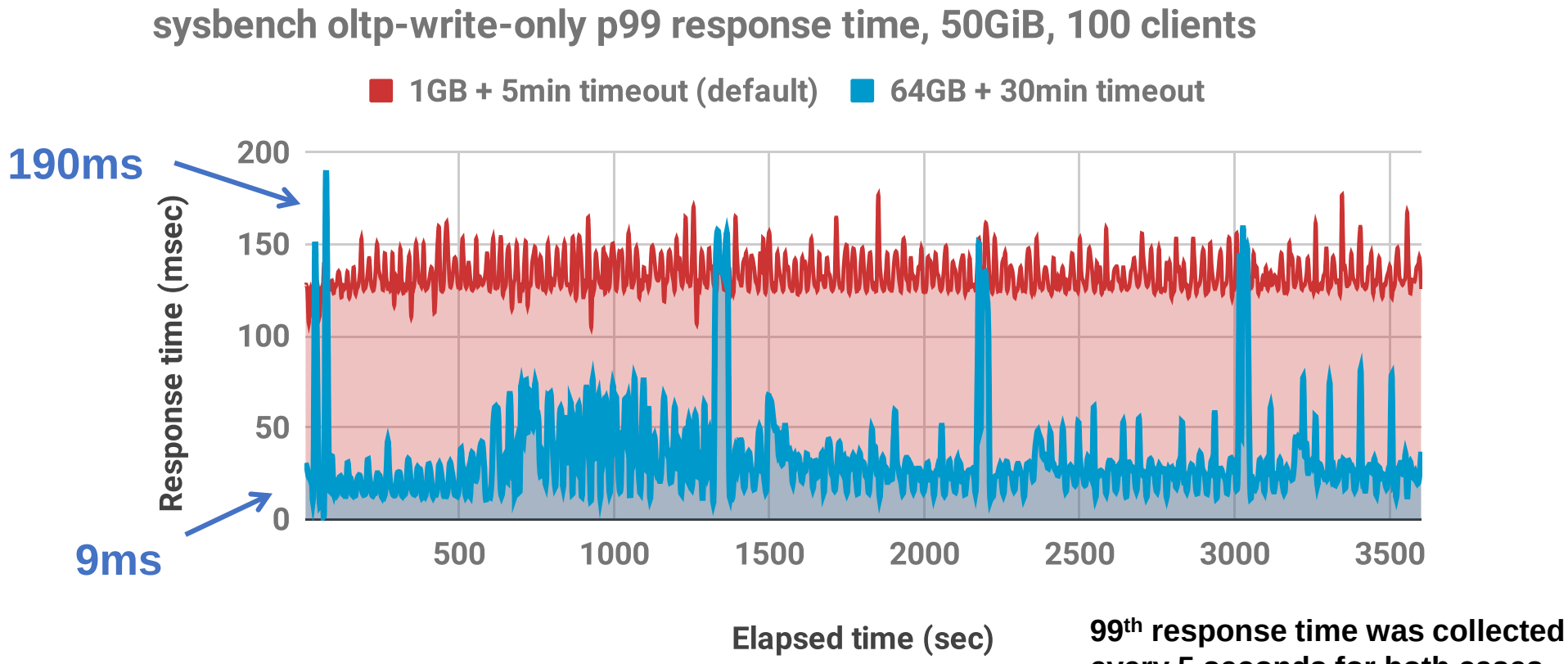


* Basics of Tuning Checkpoints (https://www.2ndquadrant.com/en/blog/basics-of-tuning-checkpoints)

Done..?

Performance Variability

Checkpoint tuning makes PostgreSQL unpredictable

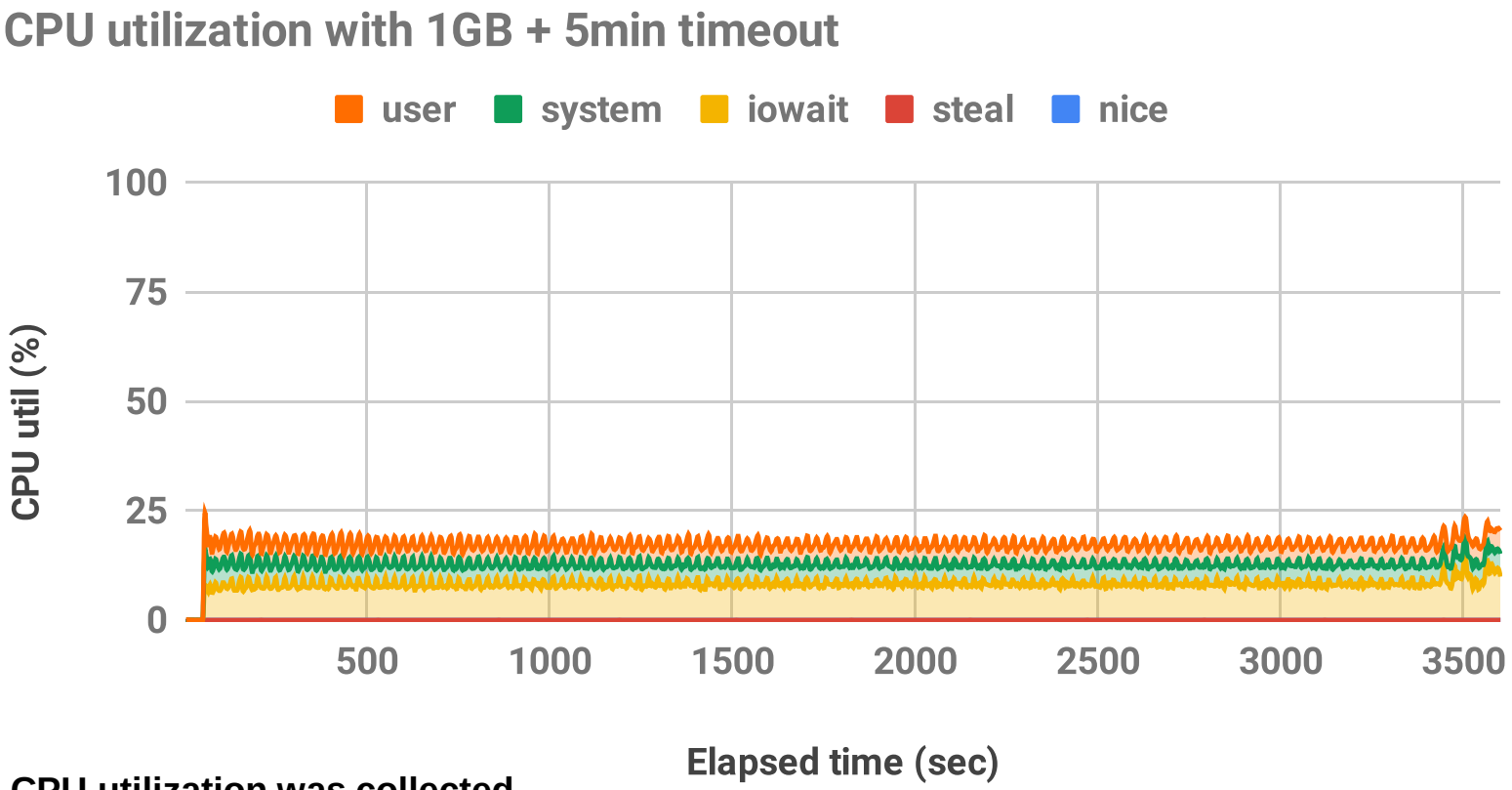


99th response time was collected every 5 seconds for both cases

Default is better in terms of variability

What's the Problem?

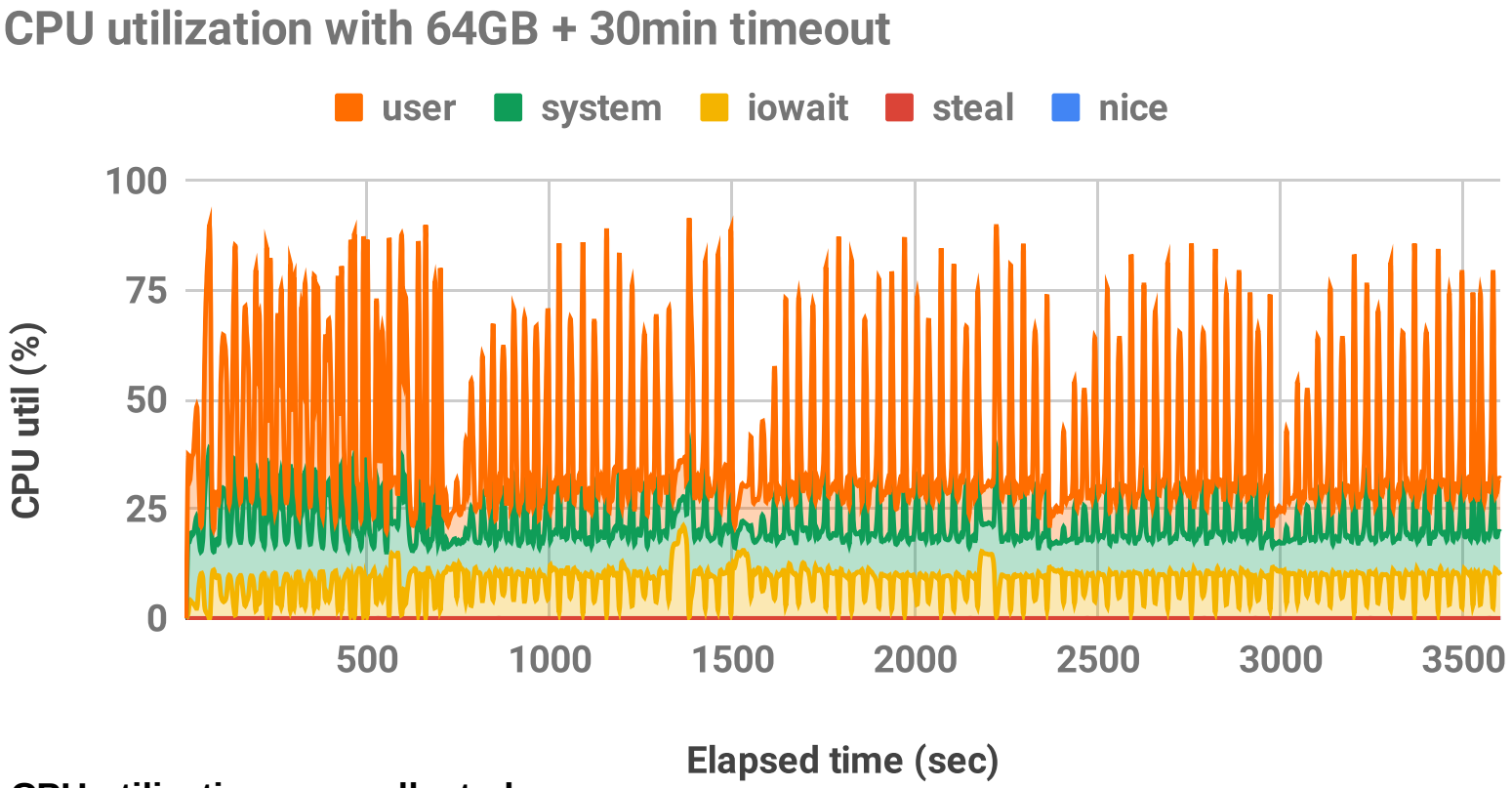
iowait is the main bottleneck for the frequent checkpoint case



CPU utilization was collected every 5 seconds using sar

What's the Problem?

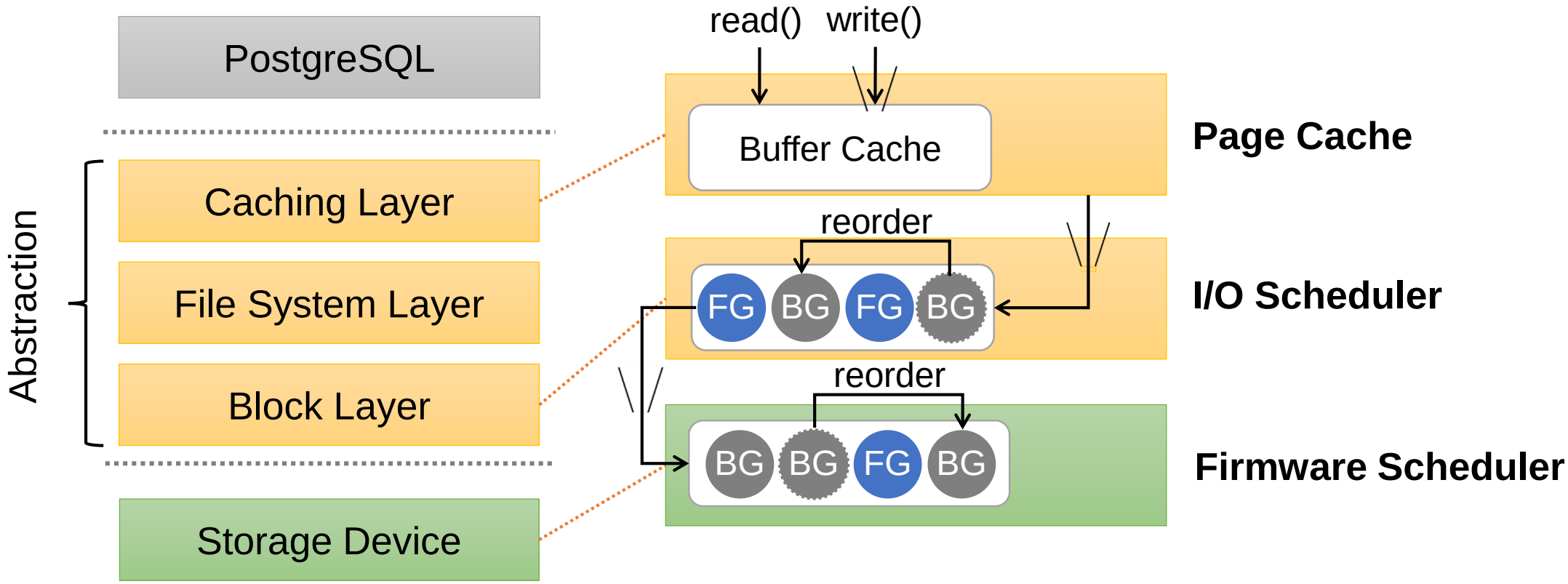
CPU utilization is highly fluctuated even without checkpoint



CPU utilization was collected every 5 seconds using sar

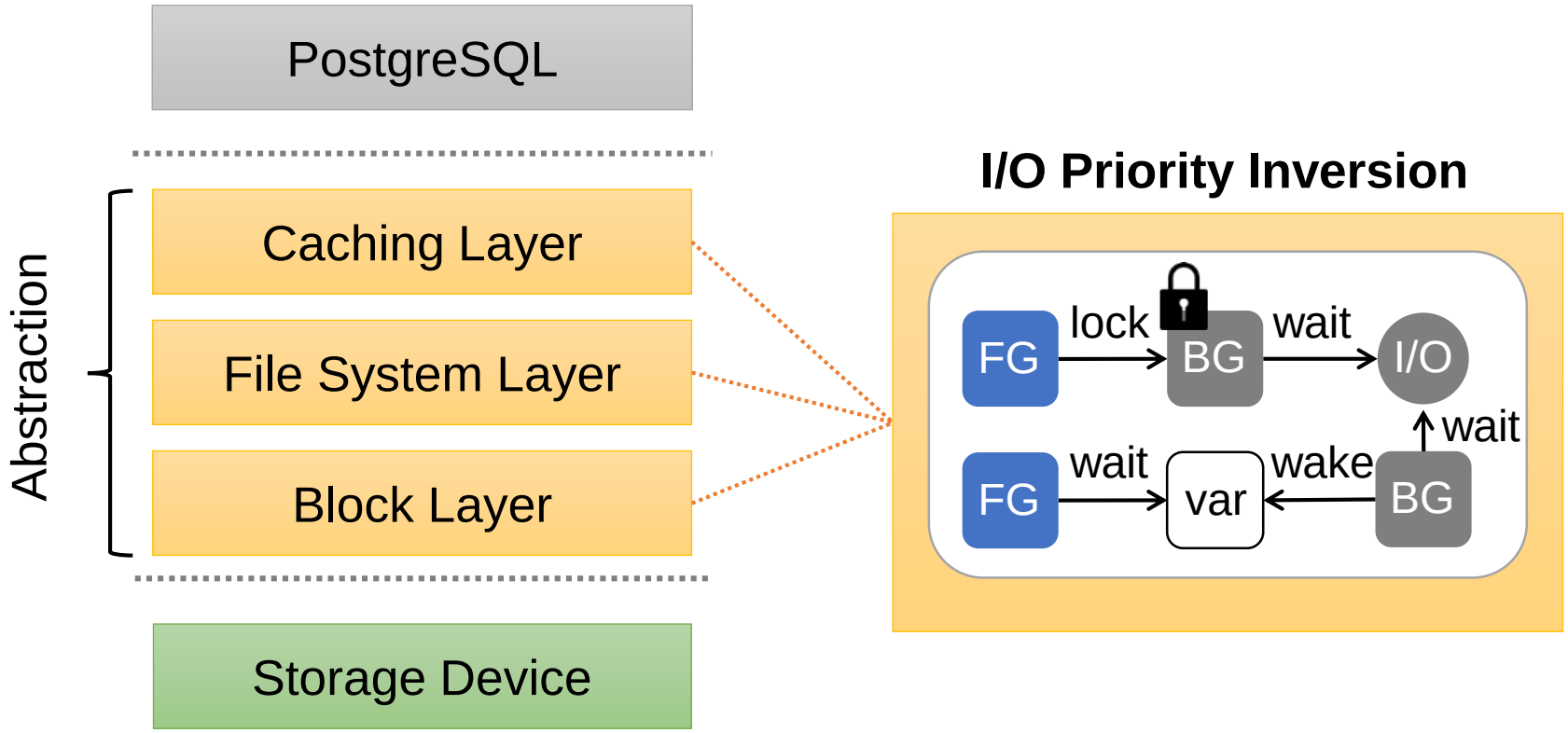
What's the Problem?

Background I/Os interfere foreground I/Os inside Linux



What's the Problem?

Linux makes backends to indirectly wait for background I/Os

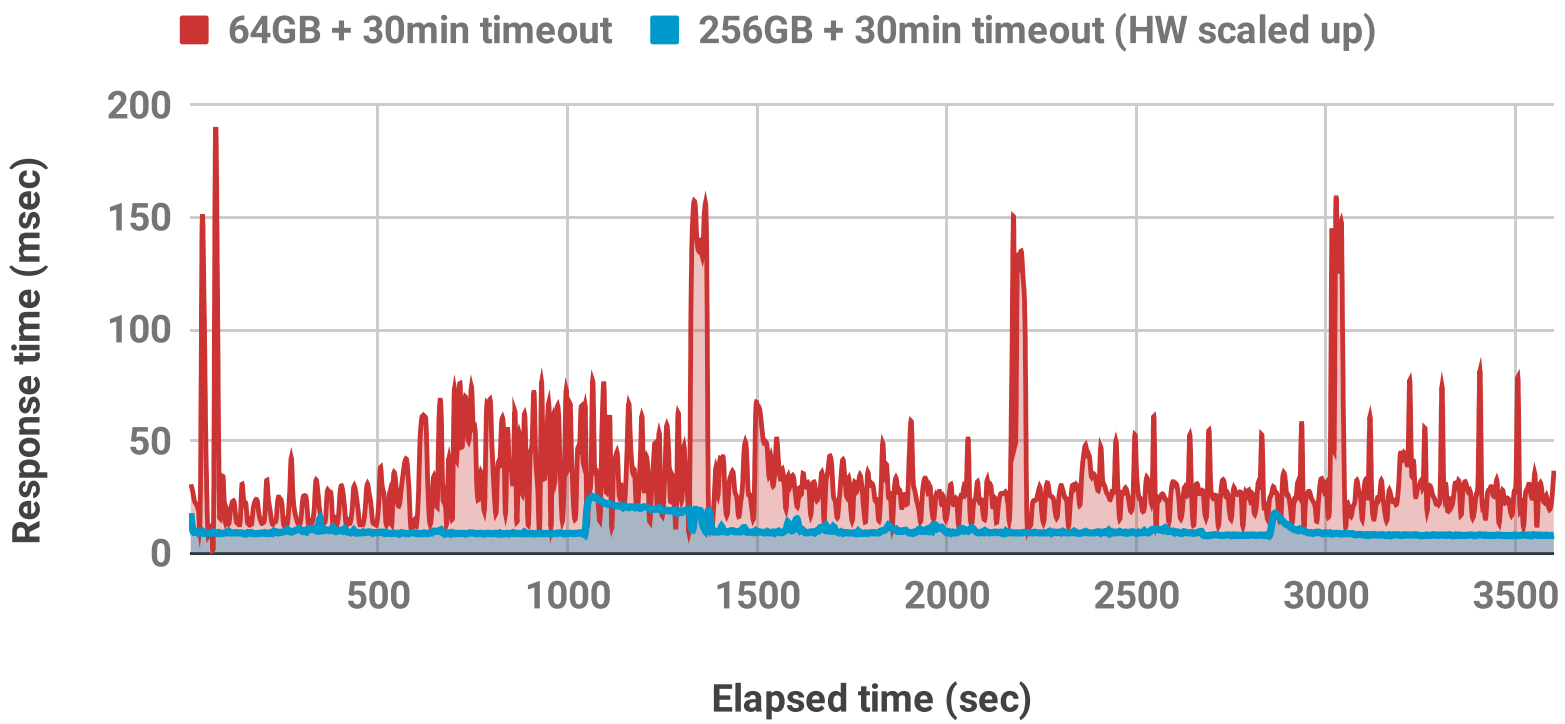


Hardware Upgrade to the Rescue

6x HW scale up mostly resolves the variability

31.37 ms vs 9.56 ms

sysbench oltp-write-only p99 response time, 50GiB, 100 clients



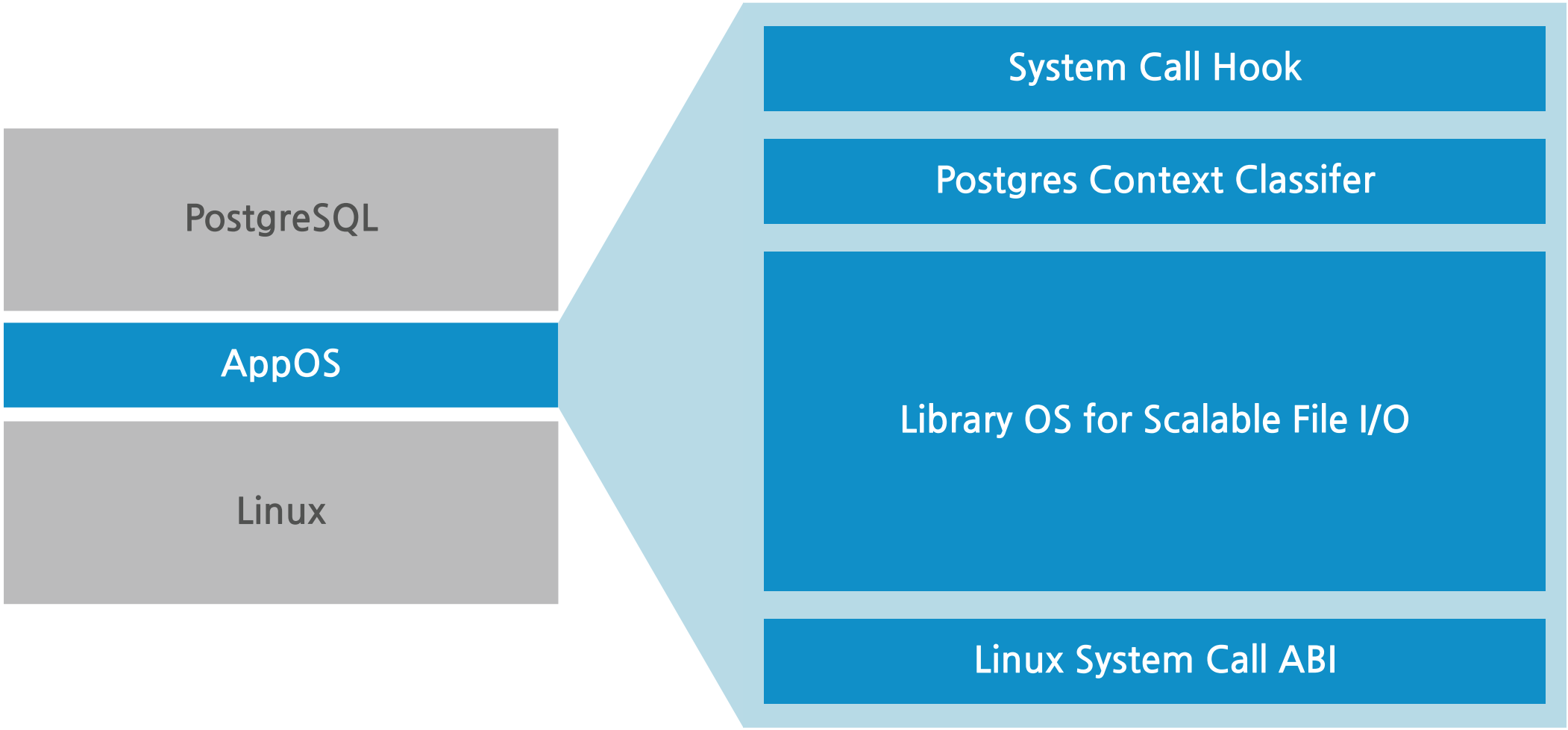
- 6x CPU cores +
  6x DRAM +
  10x storage throughput
  = 6x infra cost

- Whole dataset fit in PostgreSQL buffer

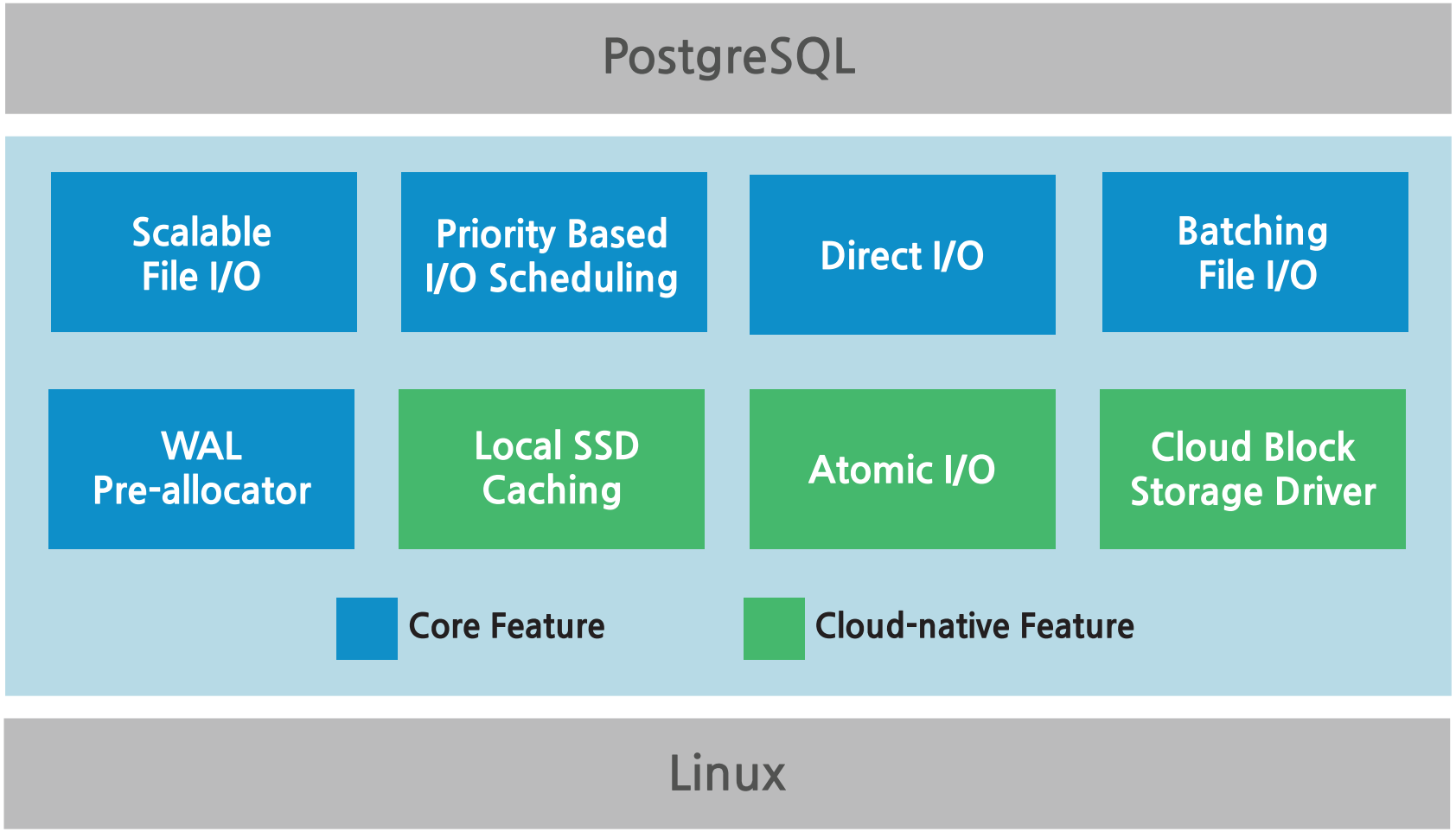
Software solution..?

AppOS is a PostgreSQL extension that provides specialized file I/O stack

AppOS Extension



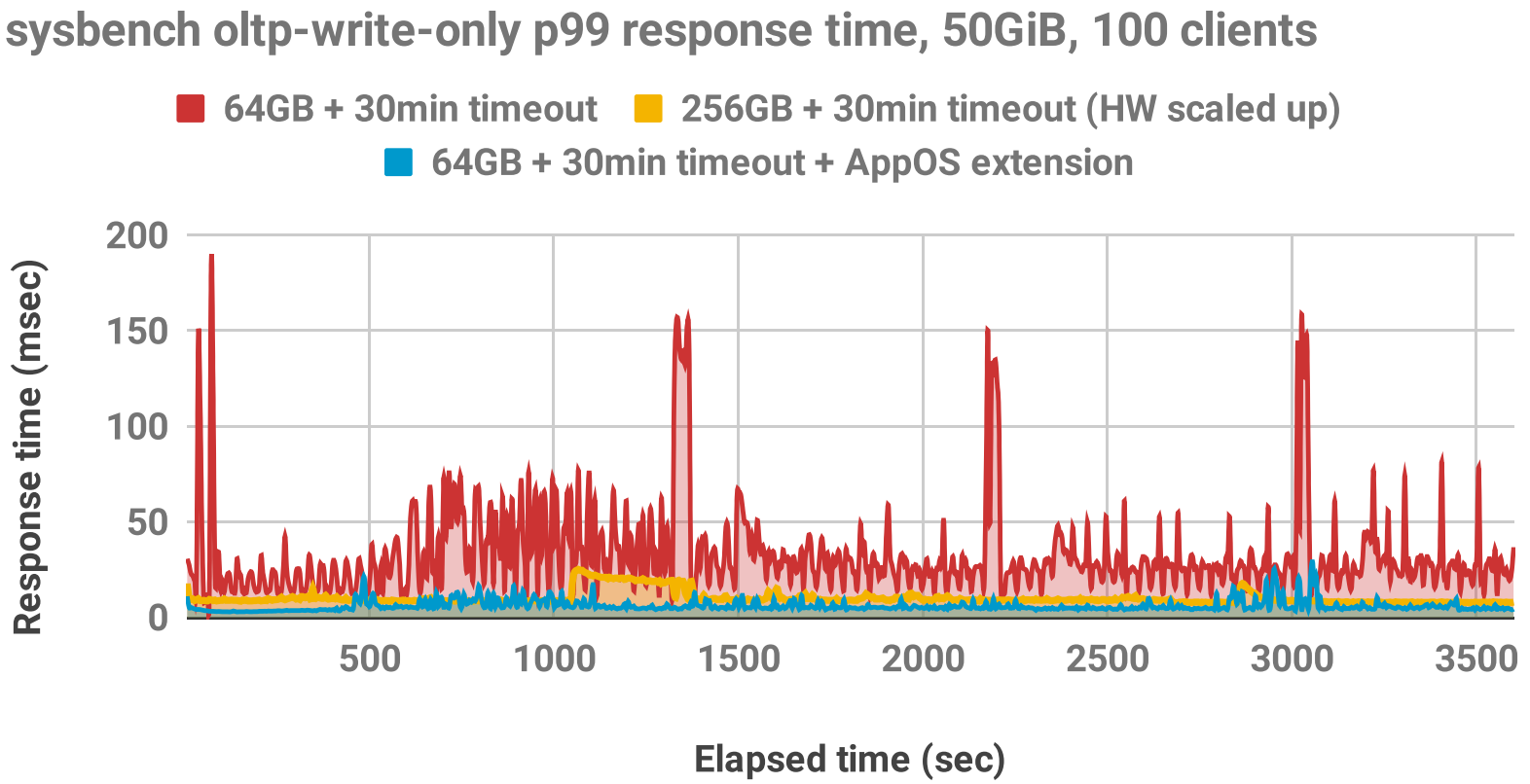
AppOS Extension



Performance Variability with AppOS

AppOS extension efficiently resolves the variability

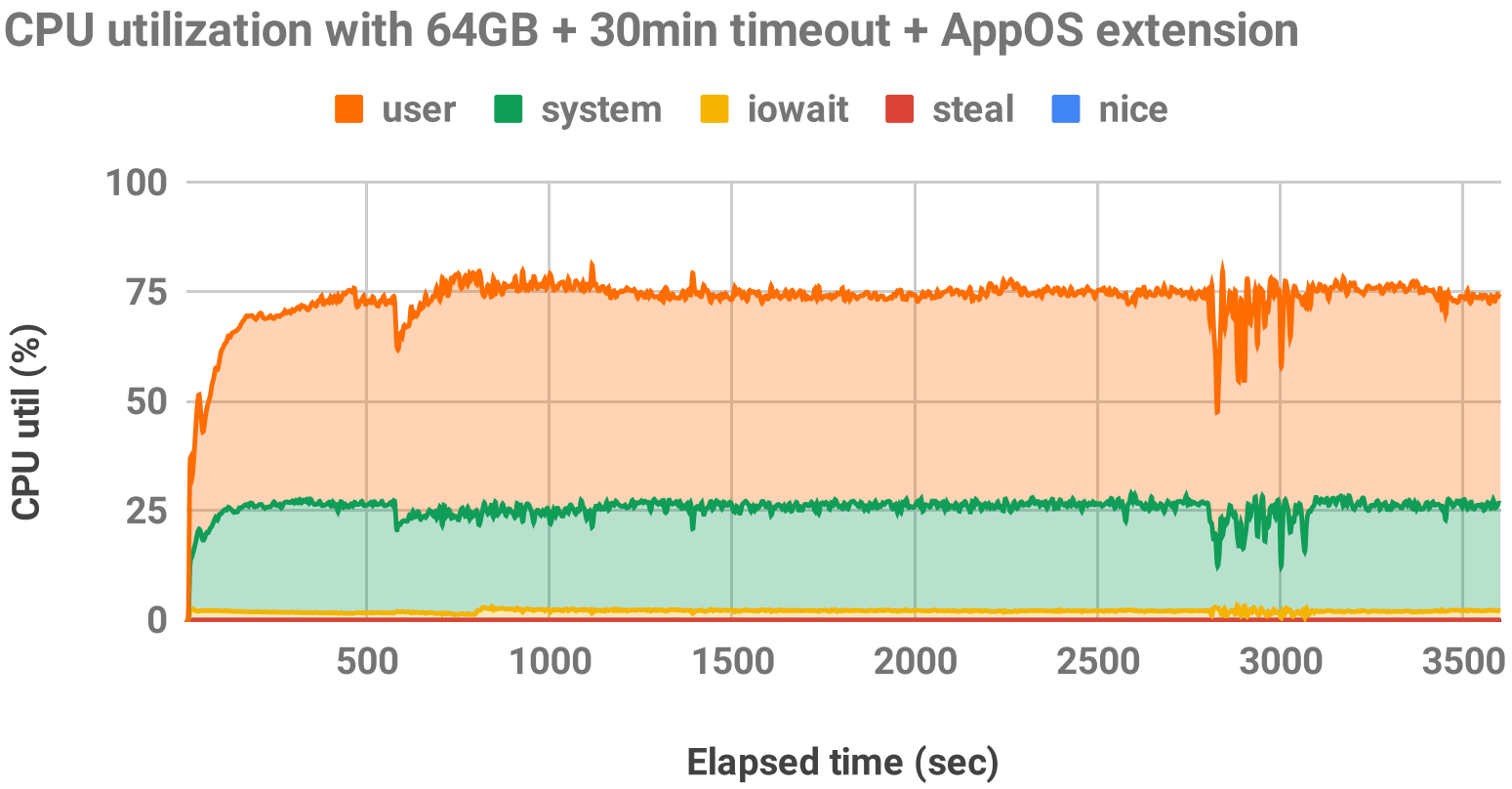
31.37 ms vs 9.56 ms vs 5.77 ms



Stable response without extra HW

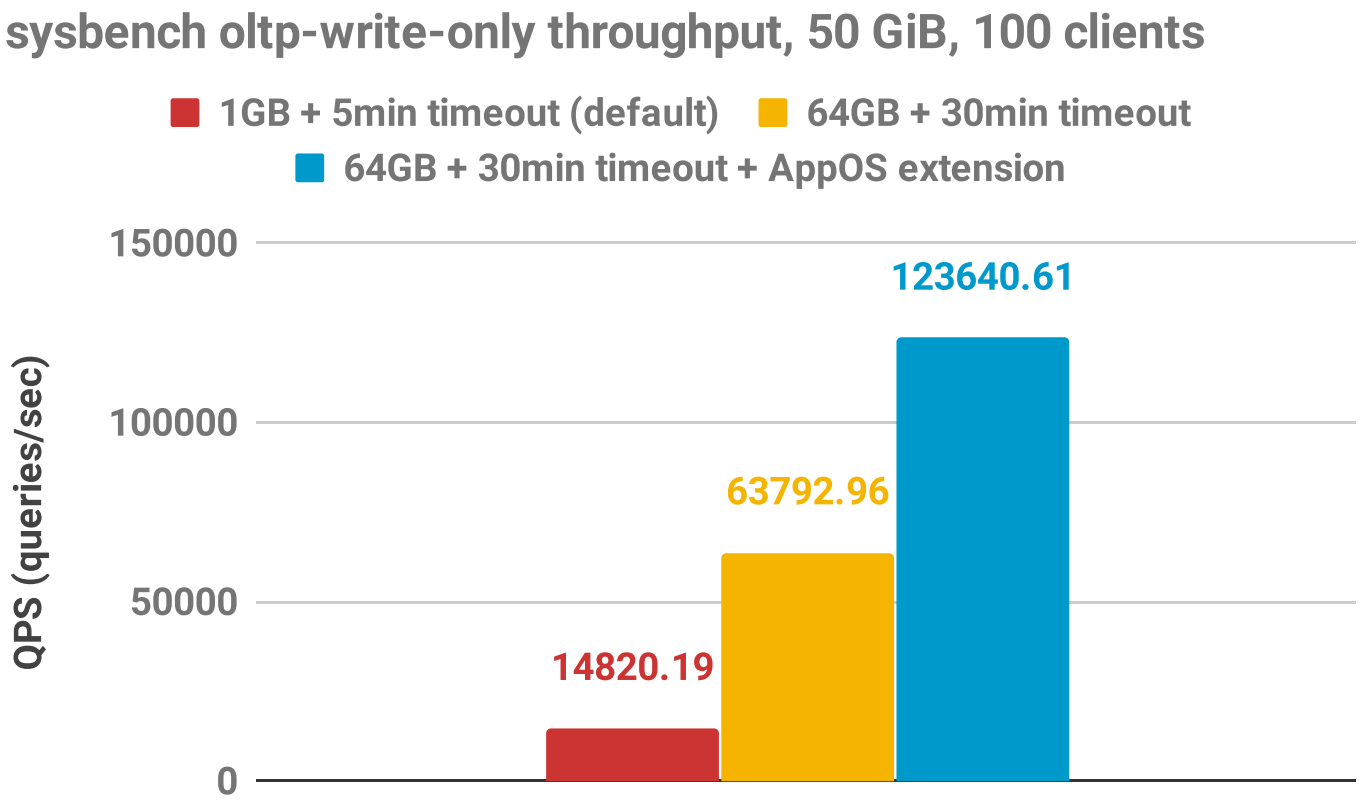
Performance Variability with AppOS

AppOS extension efficiently resolves the variability



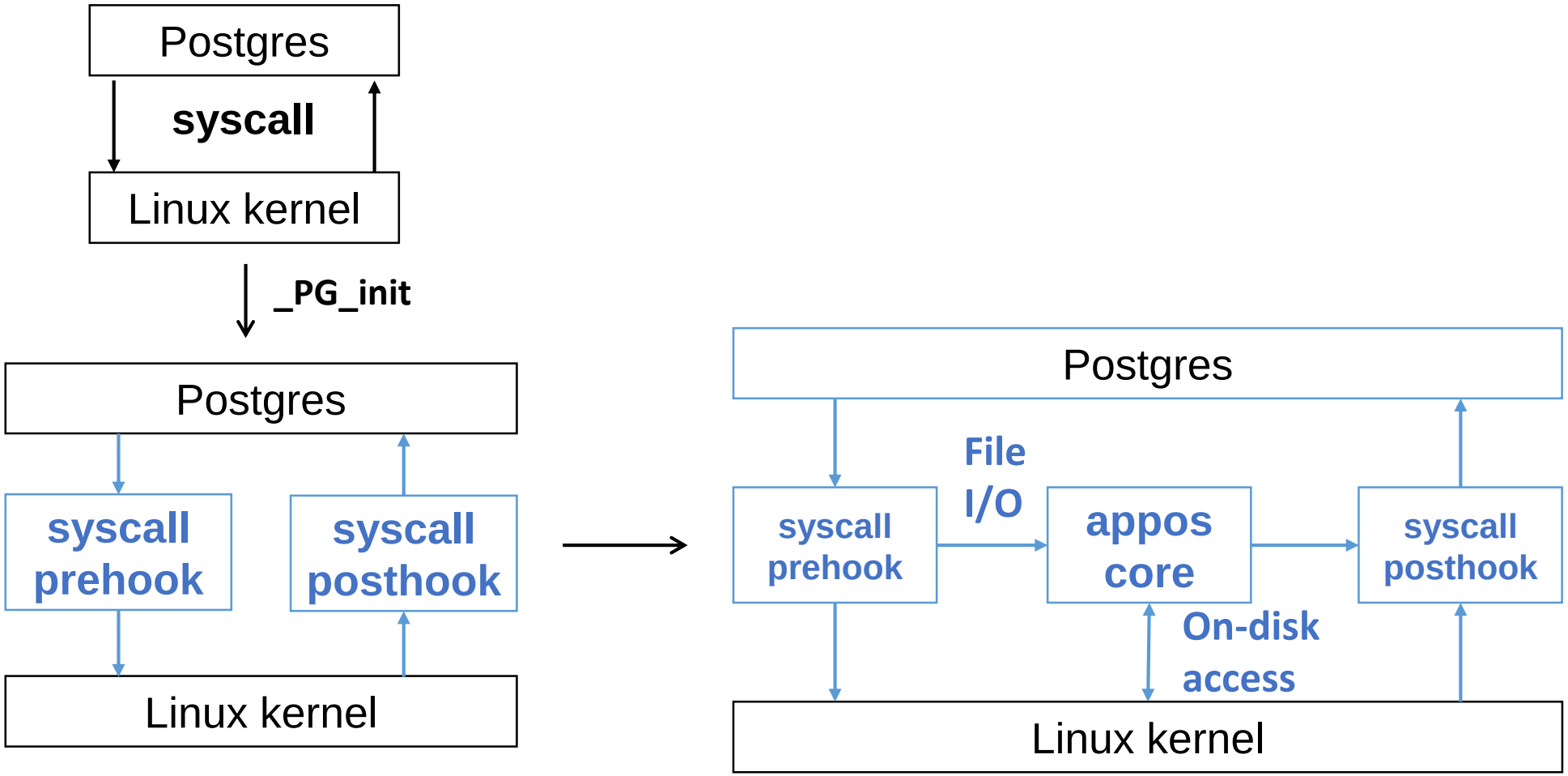
Performance Variability with AppOS

AppOS extension improves average throughput as well

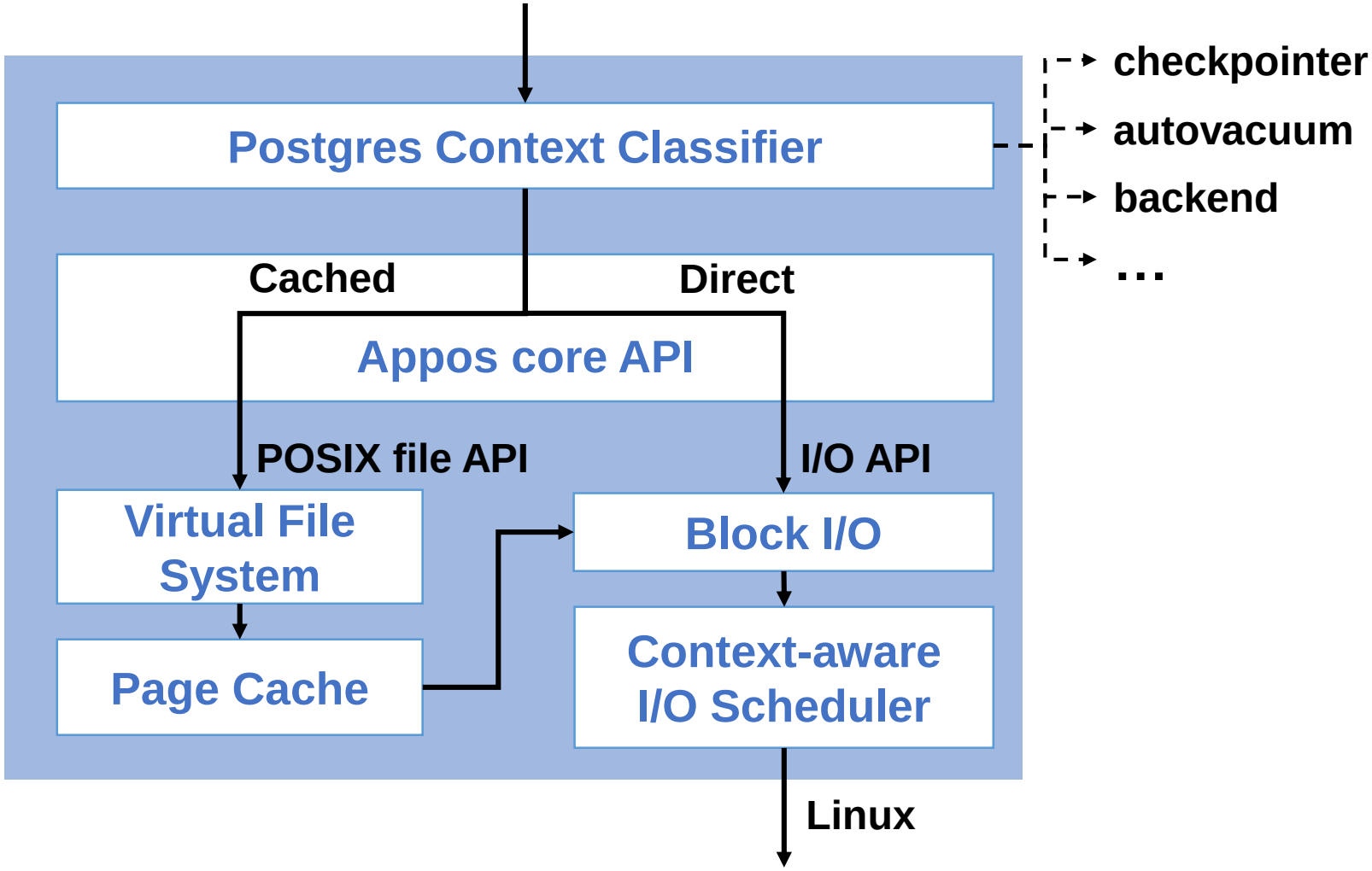


2x throughput with AppOS extension

AppOS Internals

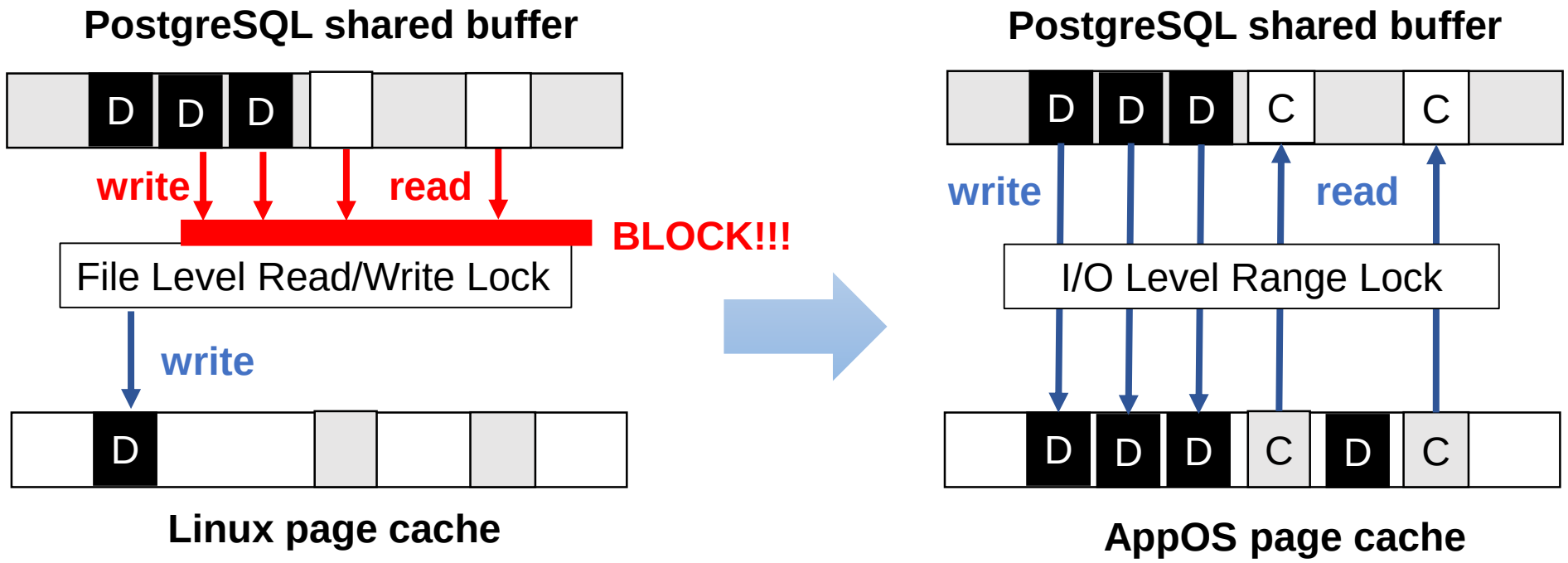


AppOS Internals



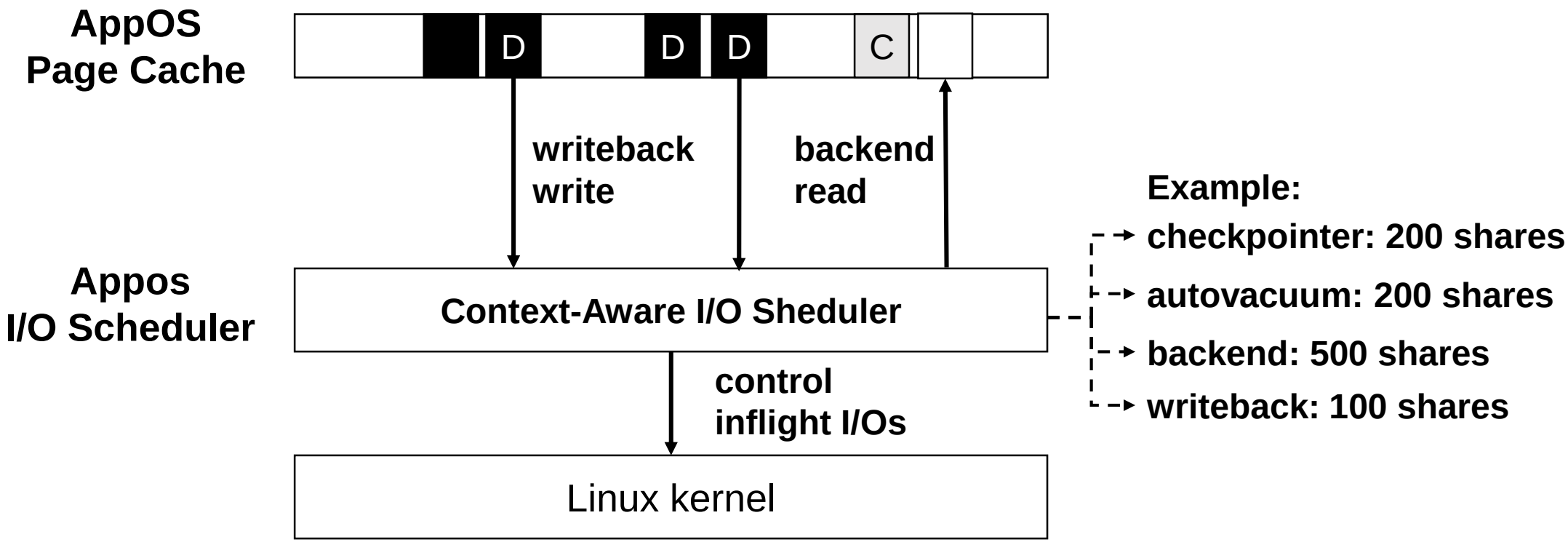
AppOS Internals

AppOS uses range lock instead of big file lock



AppOS Internals

AppOS schedules I/Os based on context and congestion



Use Cases (1)

AppOS makes PostgreSQL more predictable

Real-time SLA **Autovacuum** **Replication lag**

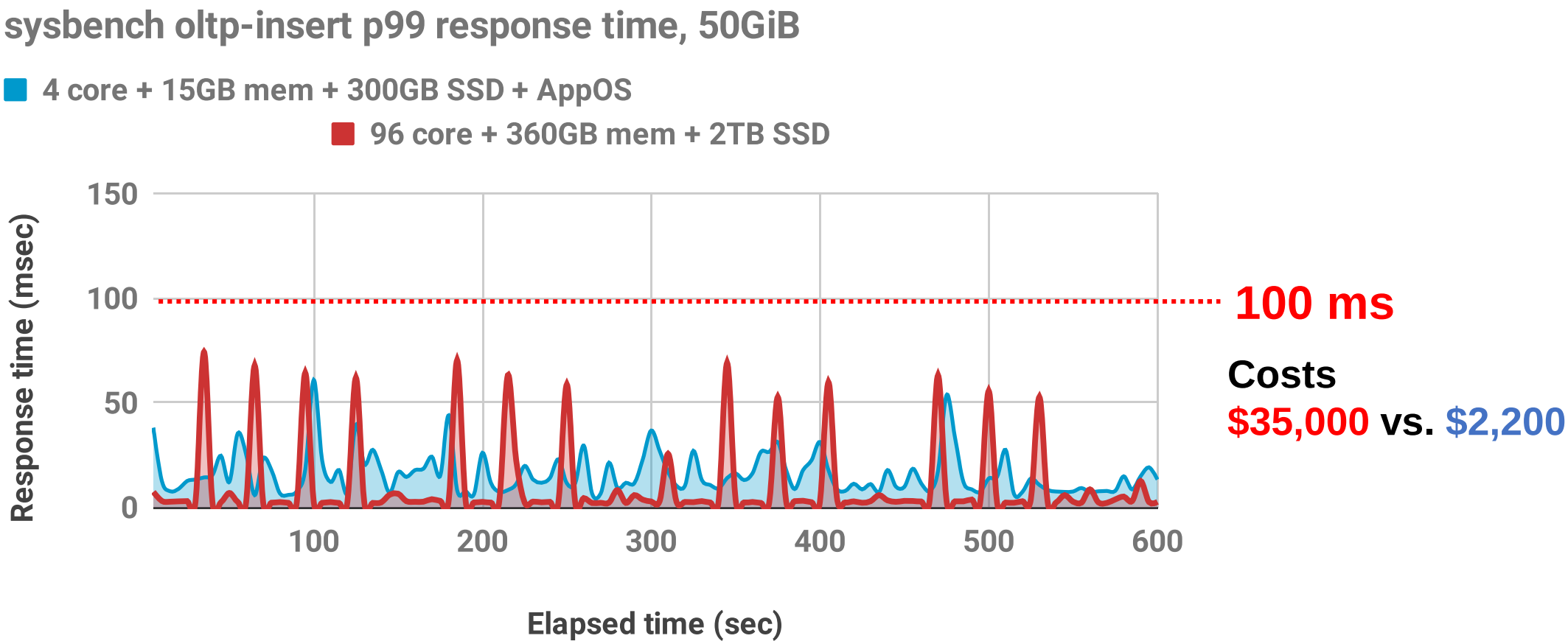
* GCP Tokyo: n1-standard-4 ~ n1-standard-96 (300GB or 2TB SSD) * SysBench 1.0.15 (client): n1-highcpu-8 (8 cores), 50GB initial dataset * PostgreSQL 11.4: shared_buffers 25% of mem, max_wal_size 8GB

Use Cases (1)

Real-time SLA

Use Cases (1)

Real-time SLA



* GCP Tokyo: n1-standard-4 ~ n1-standard-96 (300GB or 2TB SSD)
* SysBench 1.0.15 (client): n1-highcpu-8 (8 cores), 50GB initial dataset
* PostgreSQL 11.4: shared_buffers 25% of mem, max_wal_size 8GB

Use Cases (1)

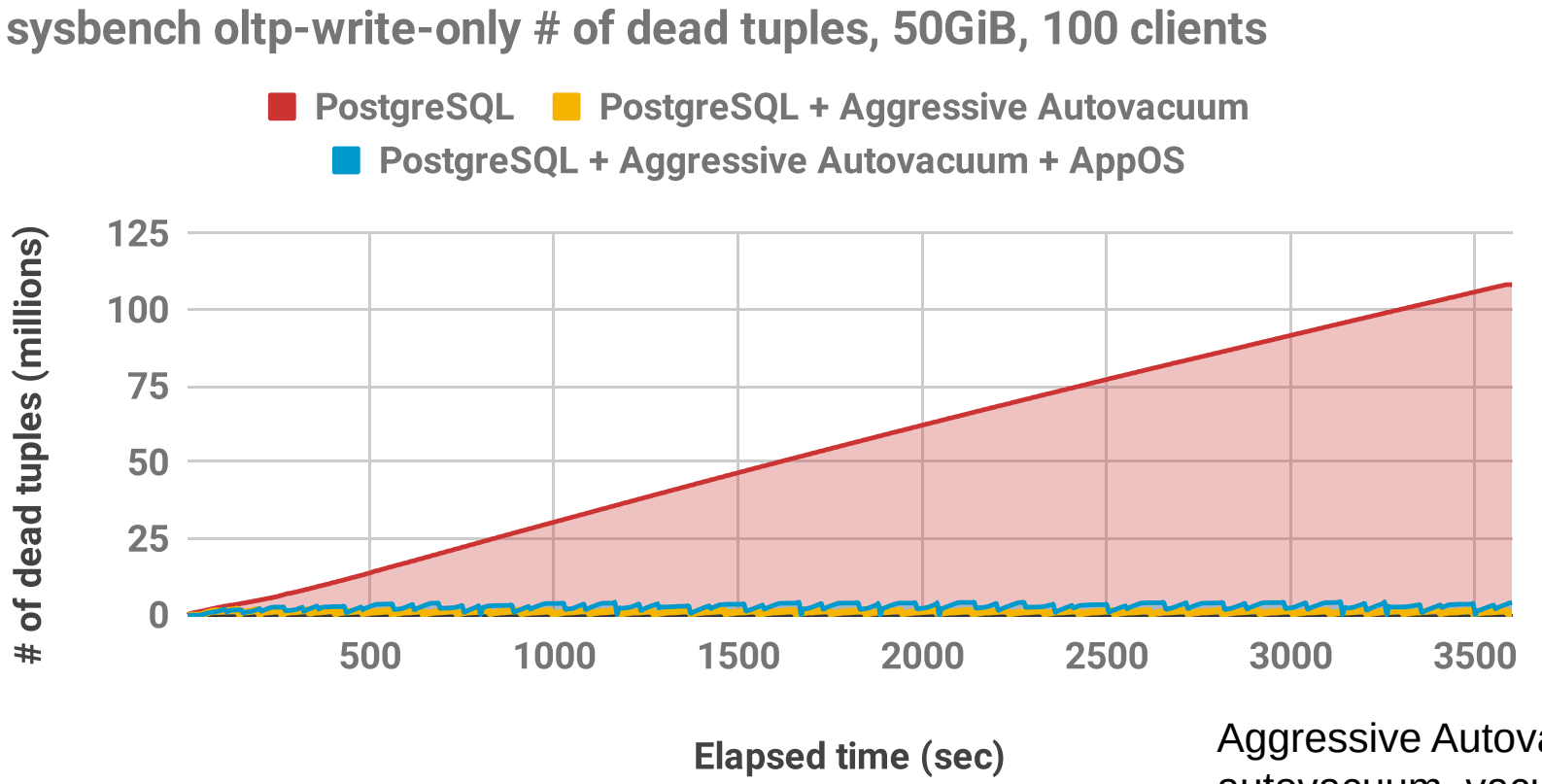
Autovacuum

Vacuum

Analyze

Use Cases (1)

Autovacuum

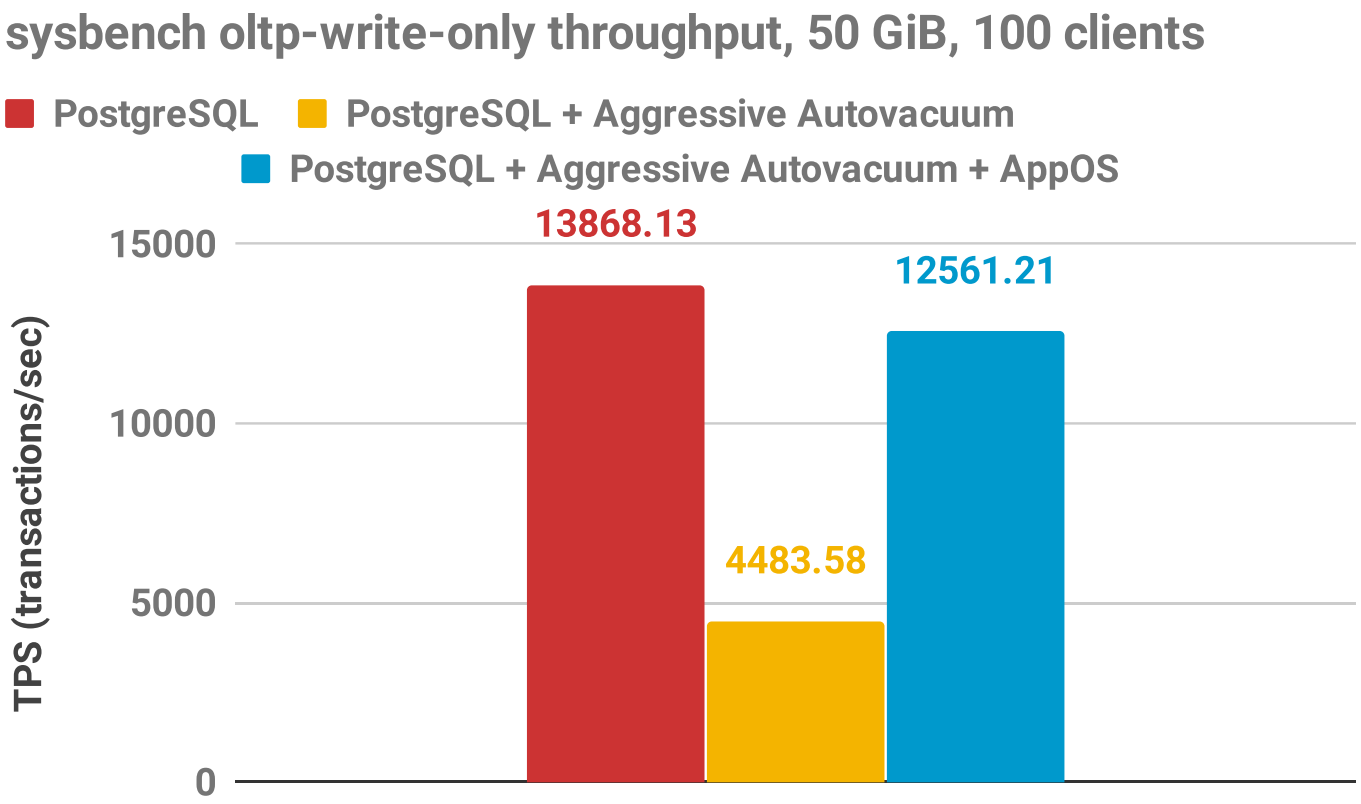


Aggressive Autovacuum:
autovacuum_vacuum_cost_delay=1s
autovacuum_vacuum_cost_limit=10000

Use Cases (1)

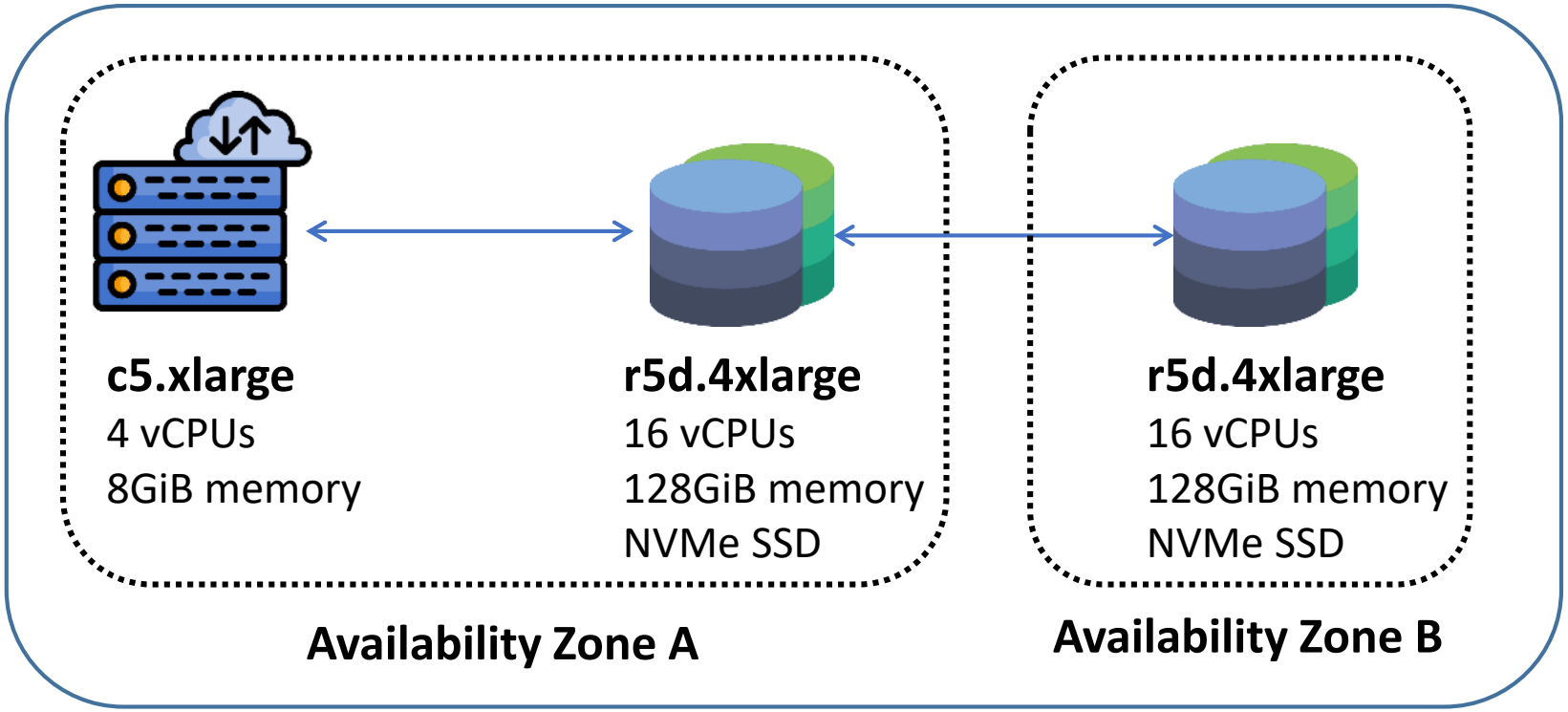
Autovacuum

13,868 TPS vs 4483 TPS vs 12,561 TPS



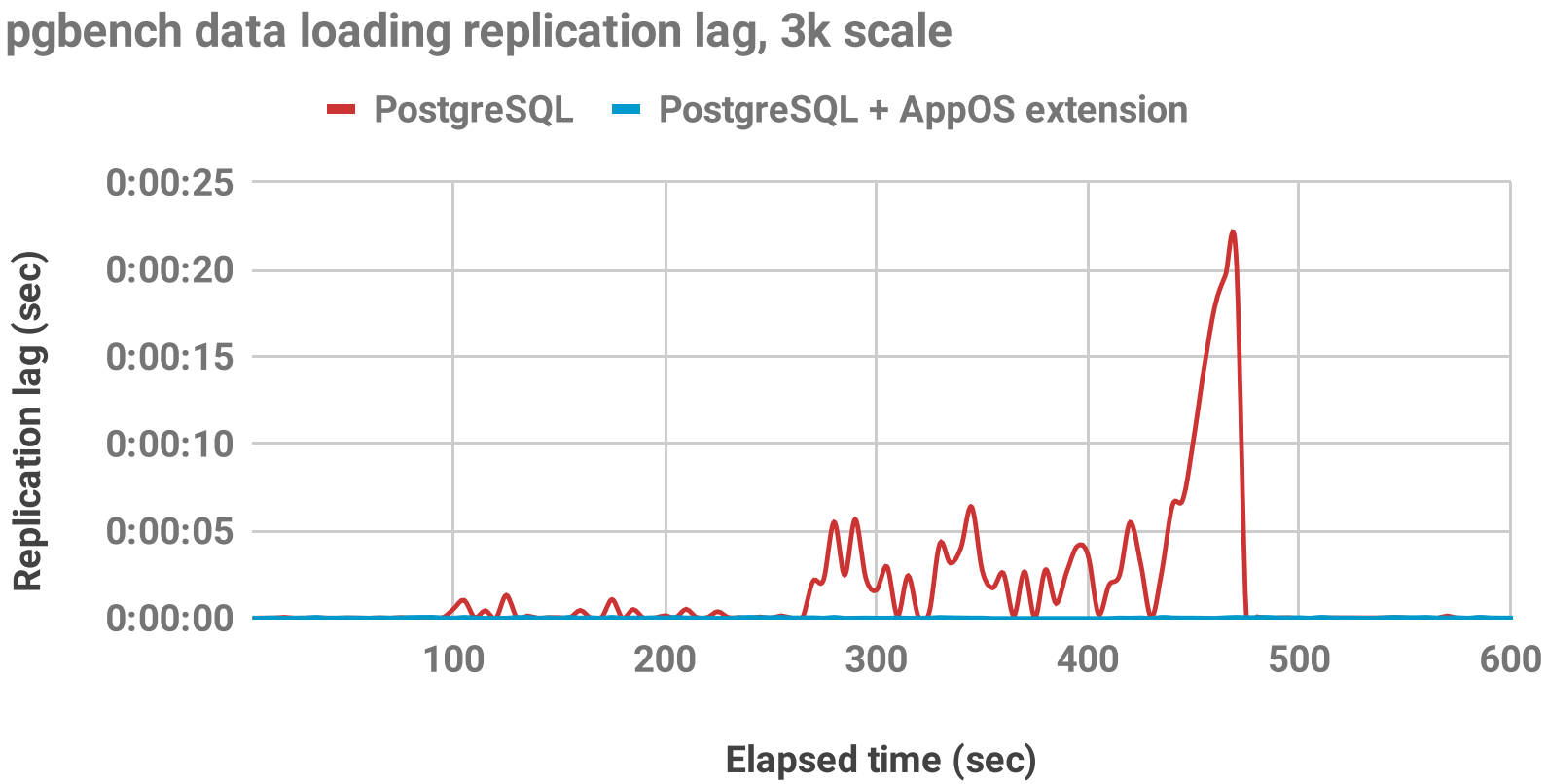
Use Cases (1)

Replication lag



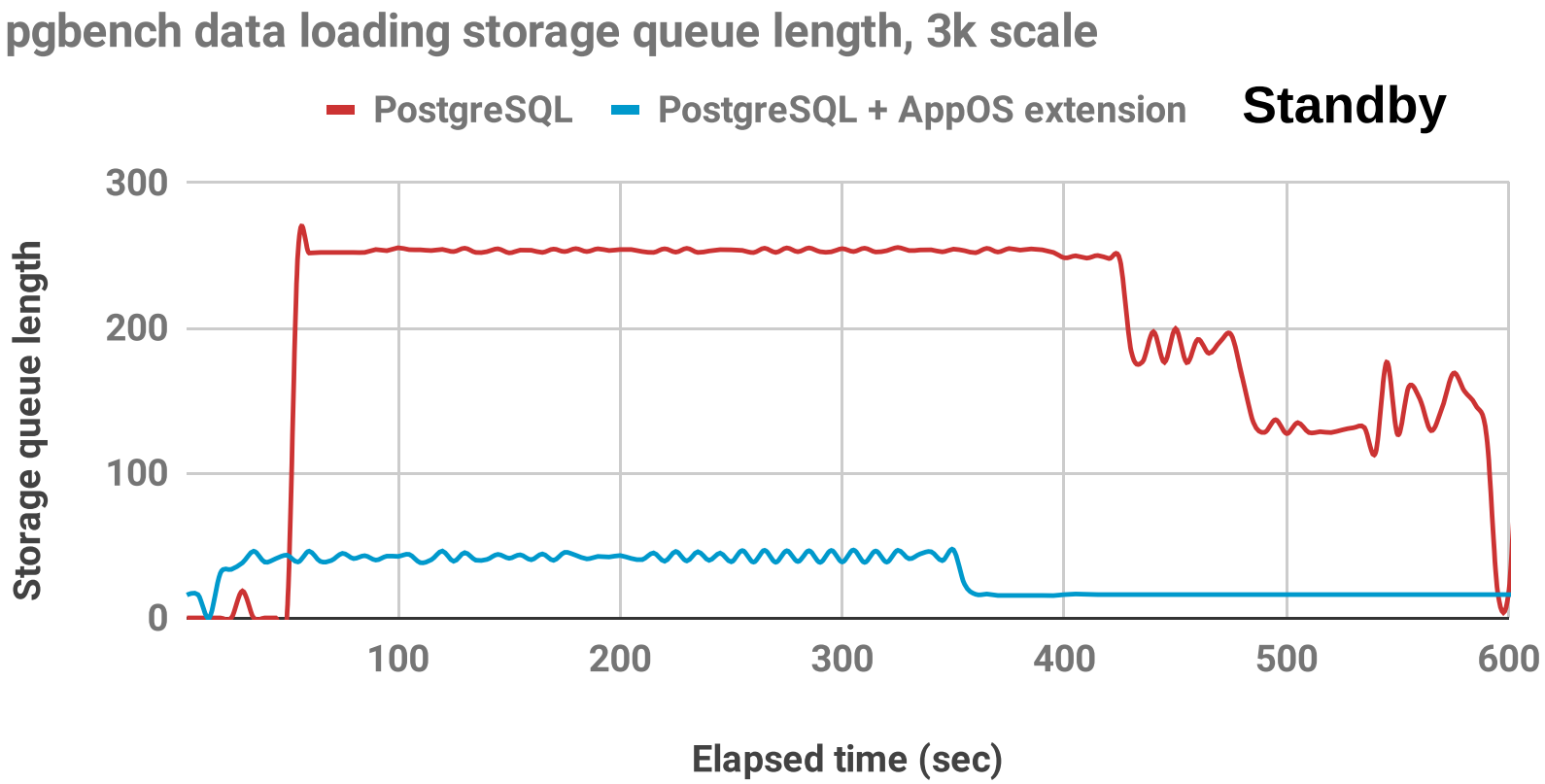
Use Cases (1)

Replication lag



Use Cases (1)

Replication lag



Use Cases (2)

AppOS makes PostgreSQL cloud storage-native

Cloud block storage

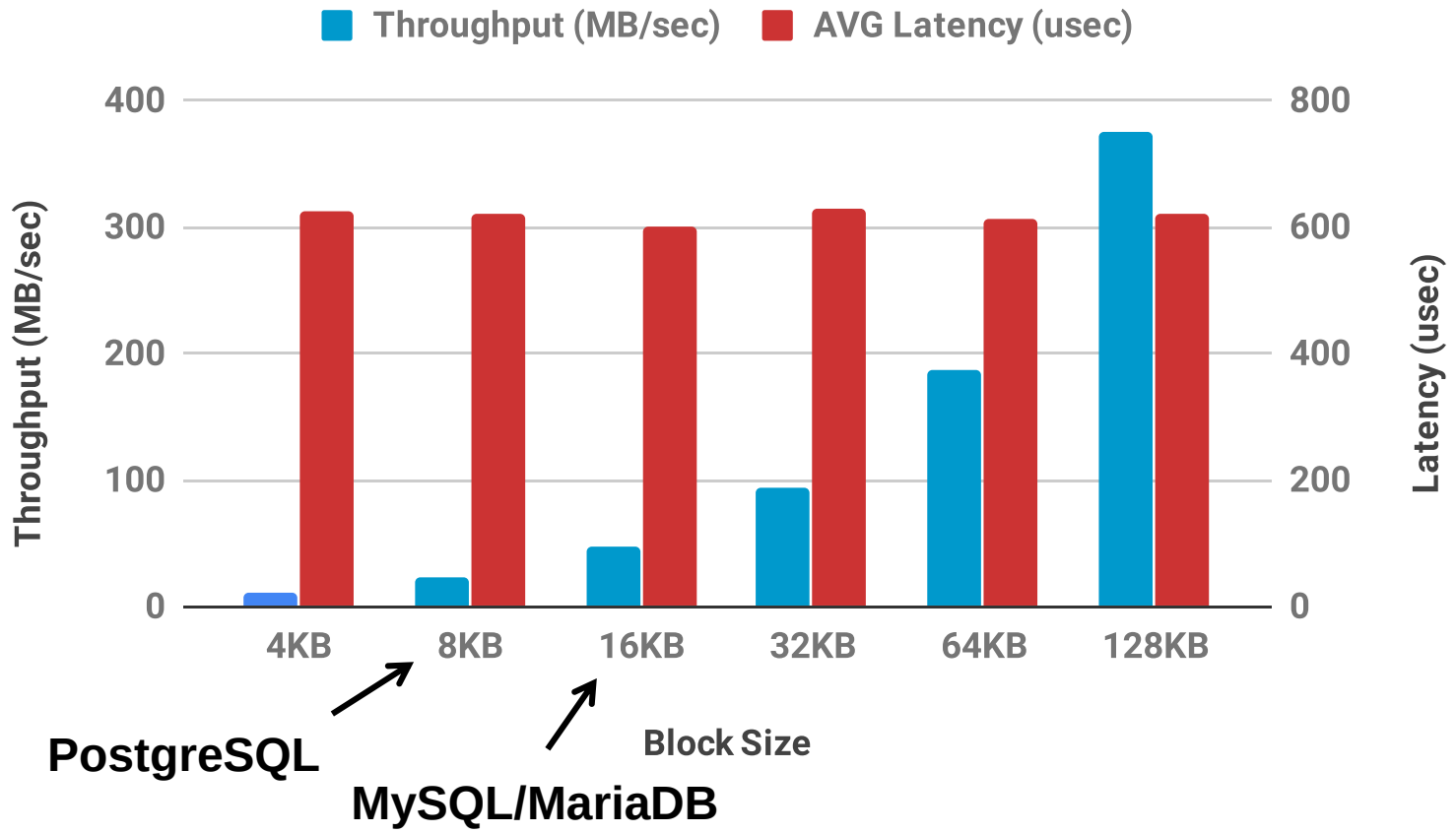
Atomic write support

Local SSD

Use Cases (2)

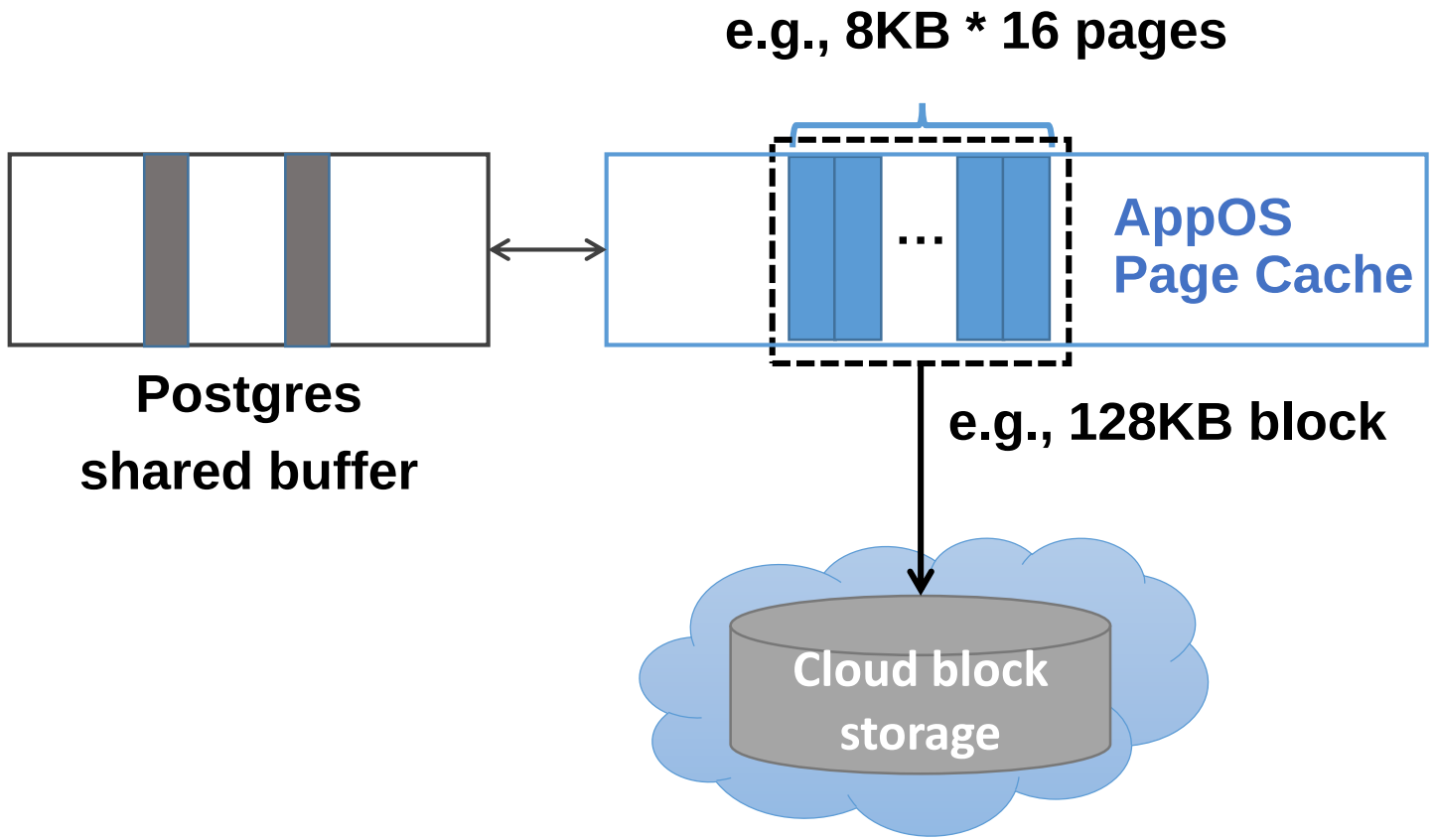
Cloud block storage

FIO random I/O test with 3000 IOPS AWS EBS



Use Cases (2)

Cloud block storage



Use Cases (2)

Atomic write support

Building a 16 KB atomic write path from database to block device

You can build an end-to-end 16 KB atomic write path from the database to the block device leveraging a 16 KB persistent disk, so you can safely disable the doublewrite feature in MySQL/ `InnoDB` and achieve a more stable and better performance for a high-write load.

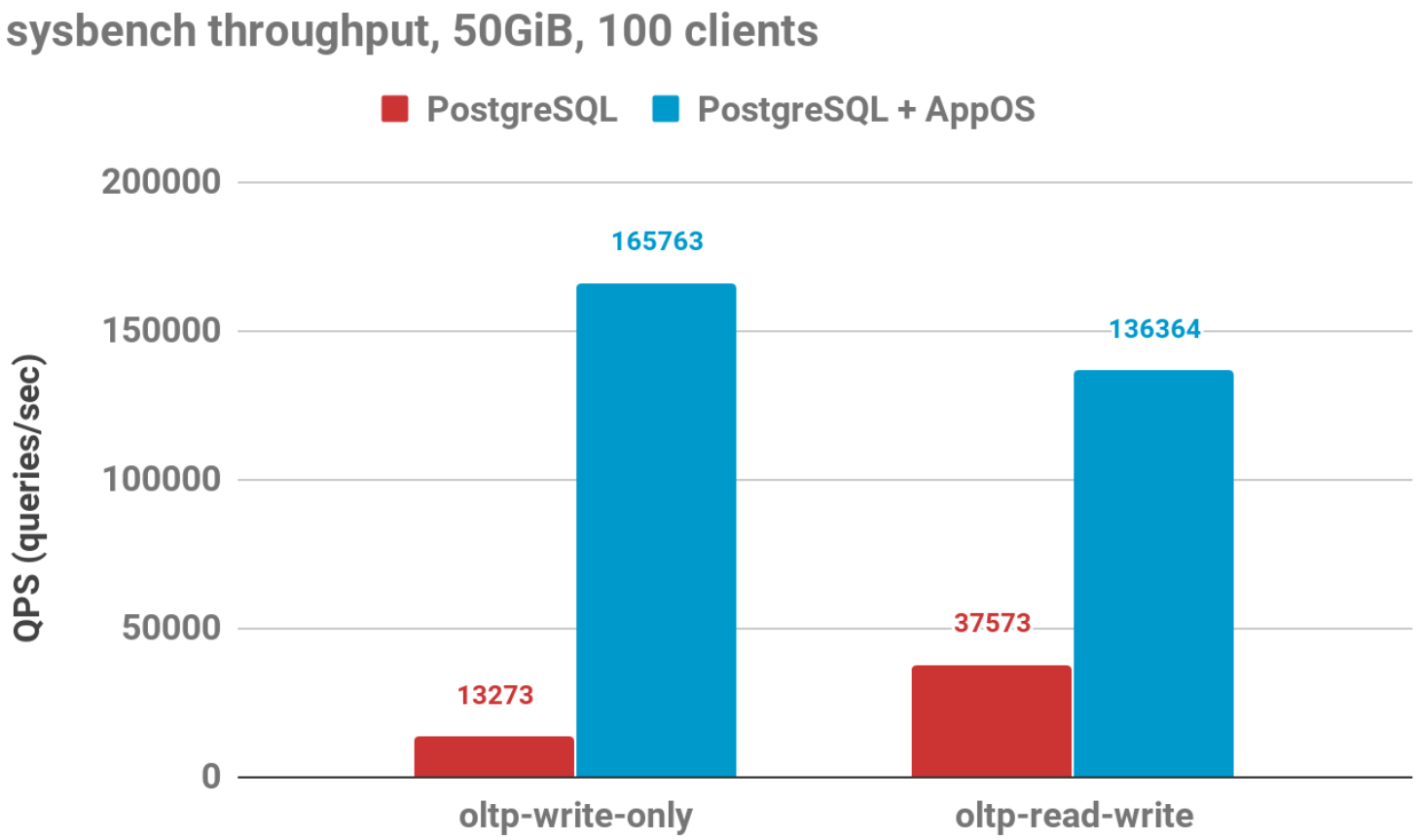
Create and attach a persistent disk through the Google Cloud Platform Console, the `gcloud` tool, or the API.

1. Create a 16 KB block size persistent disk and attach it to your VM. The 16 KB persistent disk provides 16 KB write atomicity at the physical block level.

   Although optional, it is recommended that you configure your MySQL instance to store data files only to the 16 KB persistent disk. Store log files, especially redo log and binlogs, to a 4 KB persistent disk that is attached to the same VM. This ensures log file writes continue to be high performance because small log writes on 16 KB persistent disk might trigger lots of read-modify-writes, which are slower.

Use Cases (2)

Atomic write support



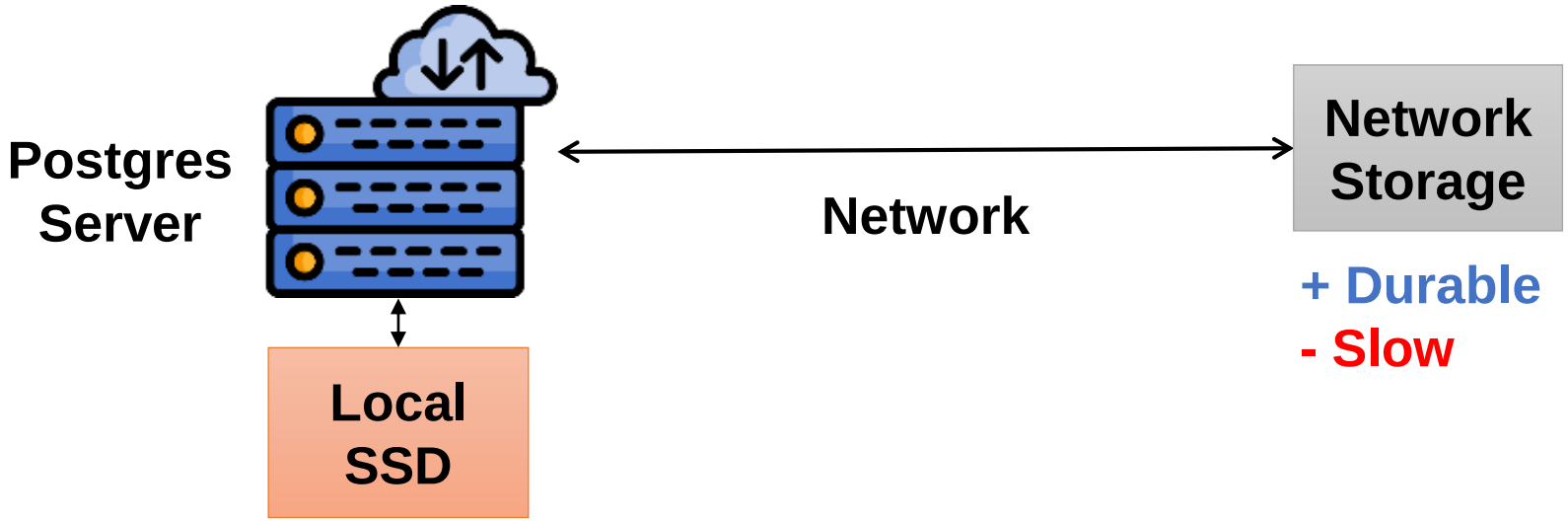
4x ~ 12x

with atomic write support on Google Cloud

* GCP Tokyo: n1-standard-32 (32 vCPUs, 120GB memory, 500GB 16KB SSD PD) * SysBench 1.0.15 (client): n1-highcpu-8 (8 vCPUs), 50GB initial dataset * PostgreSQL 9.6.12: shared_buffers 25% memory, max_wal_size 2GB

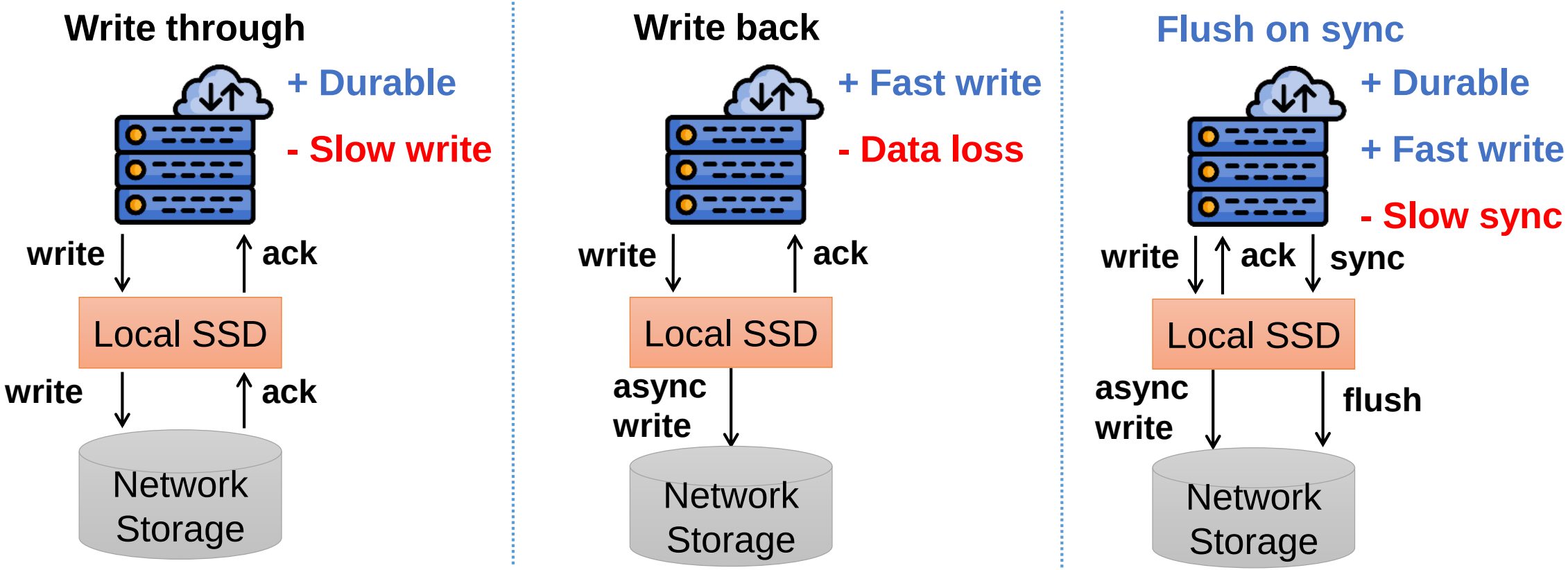
Use Cases (2)

Local SSD as a memory extension



Use Cases (2)

Local SSD as a memory extension

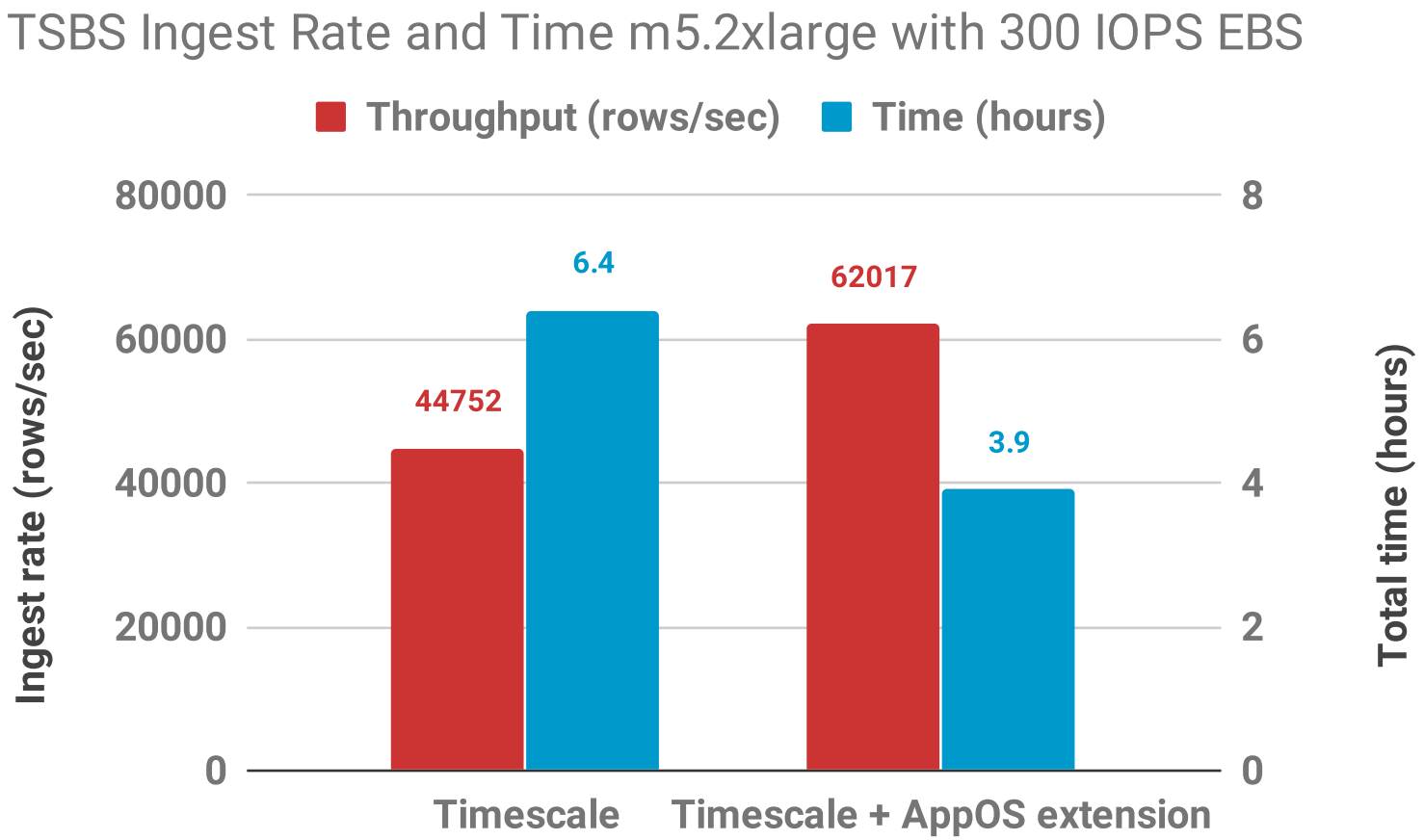


Use Cases (3)

AppOS can seamlessly work with PostgreSQL-derived DBs

Use Cases (3)

TimescaleDB



https://blog.timescale.com/blog/timescaledb-vs-6a696248104e

* AWS Seoul: m5.2xlarge (8 cores, 32gb mem, 1000GB 300 iops ebs) * SysBench 1.0.15 (client): m5.2xlarge (8 cores), 50GB initial dataset * PostgreSQL 9.6.13(Timescale 1.4.0): shared_buffers 8gb, max_wal_size 8GB

https://apposha.io
shawn@apposha.io