# How to identify and tune PostgreSQL performance issues using  wait events

Sameer Kumar
Database Specialist TAM, AWS

Roneel Kumar
Database Specialist SA, AWS

# About us

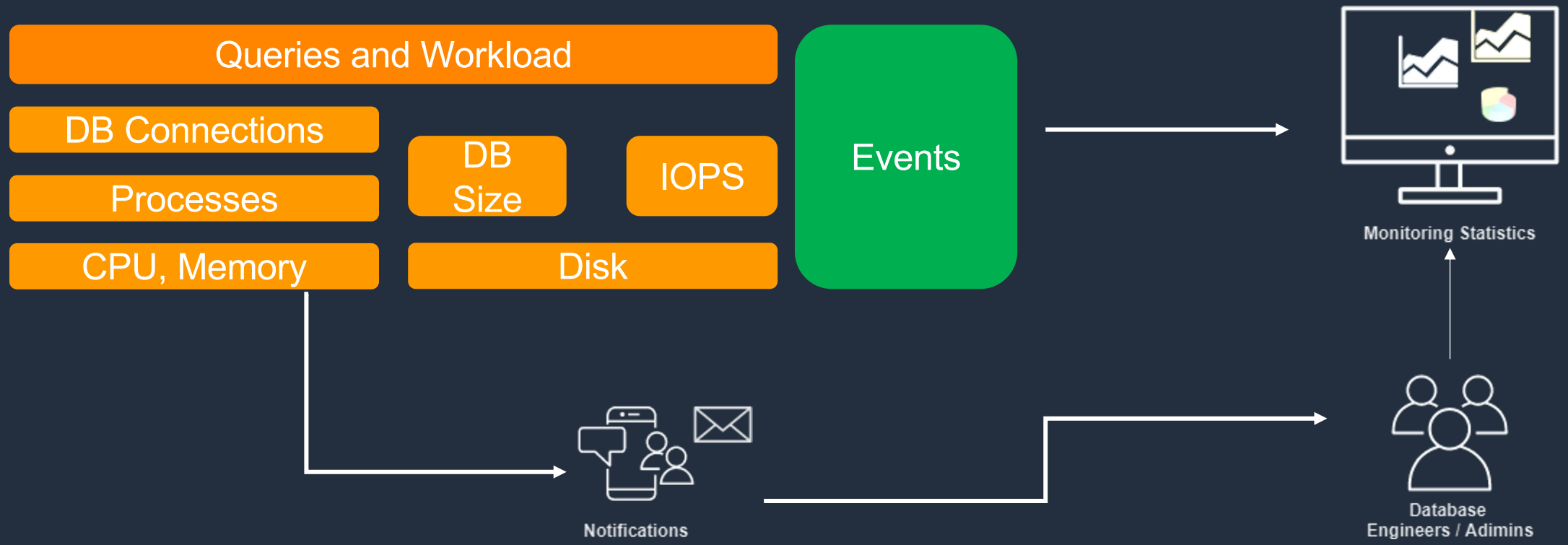
**Sameer Kumar**

**Roneel Kumar**

# Agenda

- Monitoring slow SQL statements
- Locks in PostgreSQL
- Session monitoring
- Sampling wait events
- Identifying bottleneck using wait events

# Monitoring database performance



# Monitoring slow SQL statements

- Log slow queries
  - log_min_duration_statement
  - Note: Only SQLs that have completed will be logged
- Statement statistics
  - pg_stat_statements
    - Number of calls
    - Average time to execute
    - Slowest time to execute
    - IO wait time
    - Buffer utilization
    - Temp file usage

# Typical bottlenecks in workload

- CPU
- IO
- Buffer access
- Memory structures
- Locks – table, row, page

# Monitoring session activity with pg_stat_activity

```
datid            | 16428
datname          | pgclass
pid              | 32708
leader_pid       |
usesysid         | 16427
usename          | pgadmin
application_name | pgbench
client_addr      | 10.1.0.147
client_hostname  |
client_port      | 46124
backend_start    | 2023-02-13 14:36:37.057172+00
xact_start       | 2023-02-13 14:37:47.944037+00
query_start      | 2023-02-13 14:37:47.944037+00
state_change     | 2023-02-13 14:37:47.944039+00
wait_event_type  | Lock
wait_event       | tuple
state            | active
backend_xid      | 388160634
backend_xmin     | 388159555
query_id         | 3542641618637214762
query            | update events set event_category='anomaly'
```

| | |
|---|---|
| Connection attributes | datid<br>datname<br>pid<br>leader_pid<br>usesysid<br>usename |
| Client details | application_name<br>client_addr<br>client_hostname<br>client_port |
| Time/ age | backend_start<br>xact_start<br>query_start<br>state_change |
| State | wait_event_type<br>wait_event<br>state |
| Transaction and query details | backend_xid<br>backend_xmin<br>query_id<br>query |

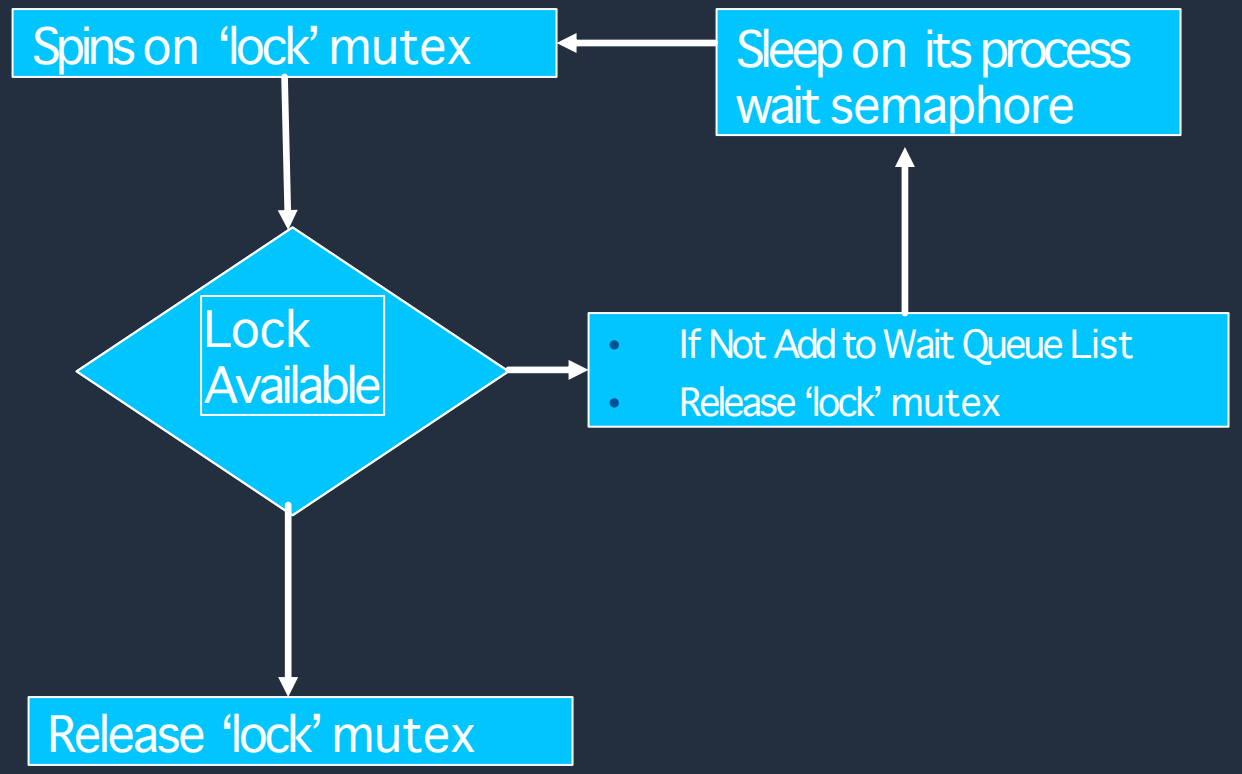
# Session state

| State | Details |
|---|---|
| active | Actively executing a query |
| idle | Idle connection – waiting for client to send a request (connection pool) |
| idle in transaction (IIT) | The session is waiting client to send a request while in middle of a transaction |
| idle in transaction (aborted) | Same as IIT but when previous command resulted in an error |
| fastpath function call | The backend is executing a fast-path function |
| disabled | Activity tracking is disabled in session (track_activity) |

# Wait Event Types

| Wait Event Type | Description |
|---|---|
| `Activity` | An idle backend server process waiting for activity in its main processing loop. |
| `BufferPin` | Waiting for exclusive access to a data buffer |
| `Client` | Wating for activity on a socket connected to a user application (external to server) |
| `Extension` | Waiting on condition defined by an Extension module |
| `IO` | Waiting on disk IO |
| `IPC` | Waiting on another PostgreSQL server process |
| `Lock` | Waiting on heavy weight locks – such as row lock or table lock |
| `LWLock` | Waiting on light weight locks (typically used to protect in-memory structure) |
| `Timeout` | Waiting for a timeout to expire – for example Vacuum Sleep |

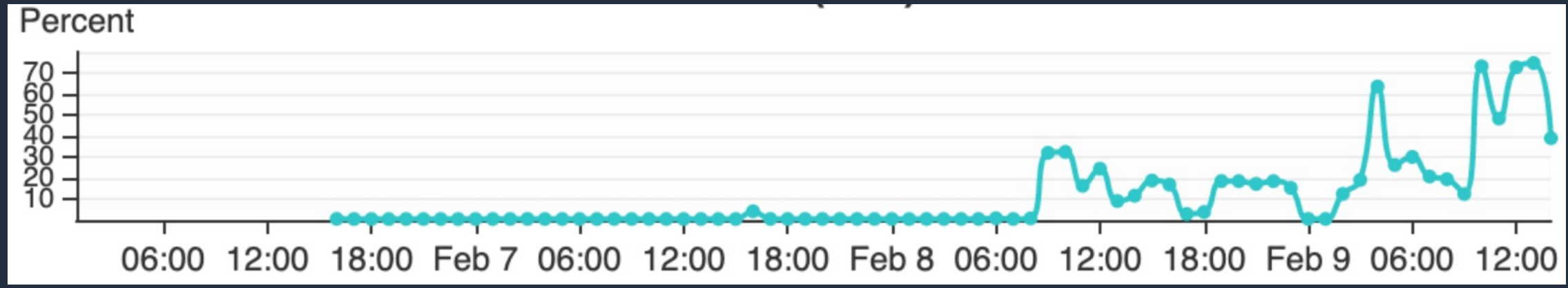
# Lock acquisition



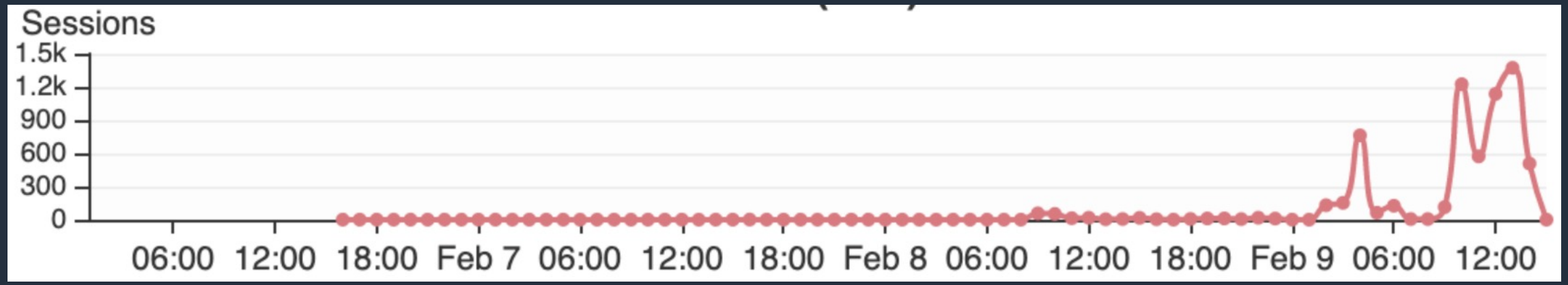

| Wait Event Type | Description |
|---|---|
| `Spinlock` | Lightweight lock |
| `LwLockNamed` | Waiting for exclusive access to a data buffer |
| `LwLockTranche` | Wating for activity on a socket connected to a user application (external to server) |
| `BufferPin` | Exclusive access to shared buffer |
| `Lock` | a heavy-weight lock — used mostly for SQL level objects |

# Impact of active sessions on utilization

## CPU Utilization

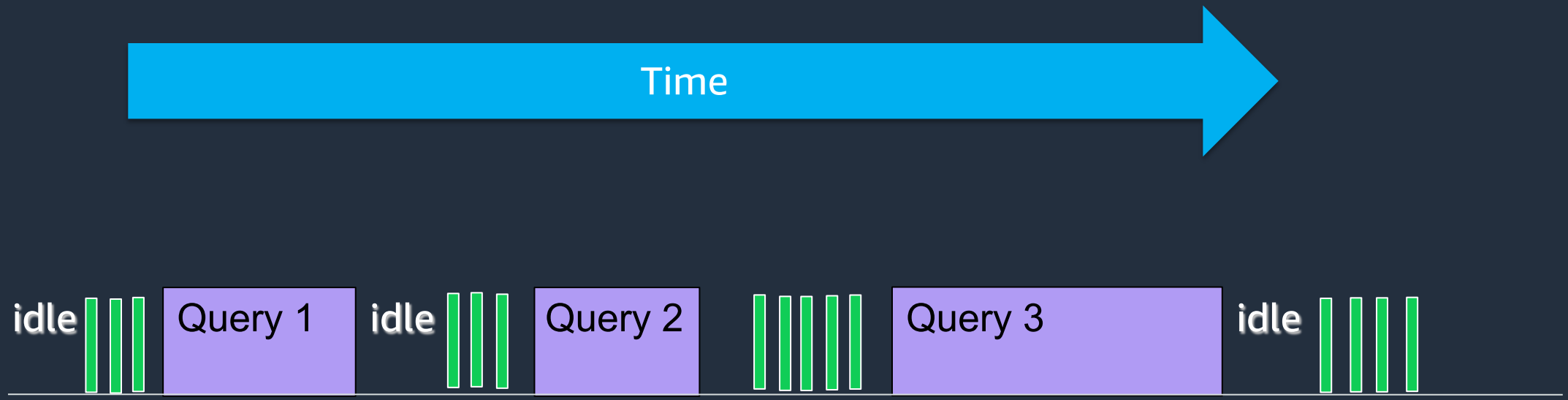



## Active sessions



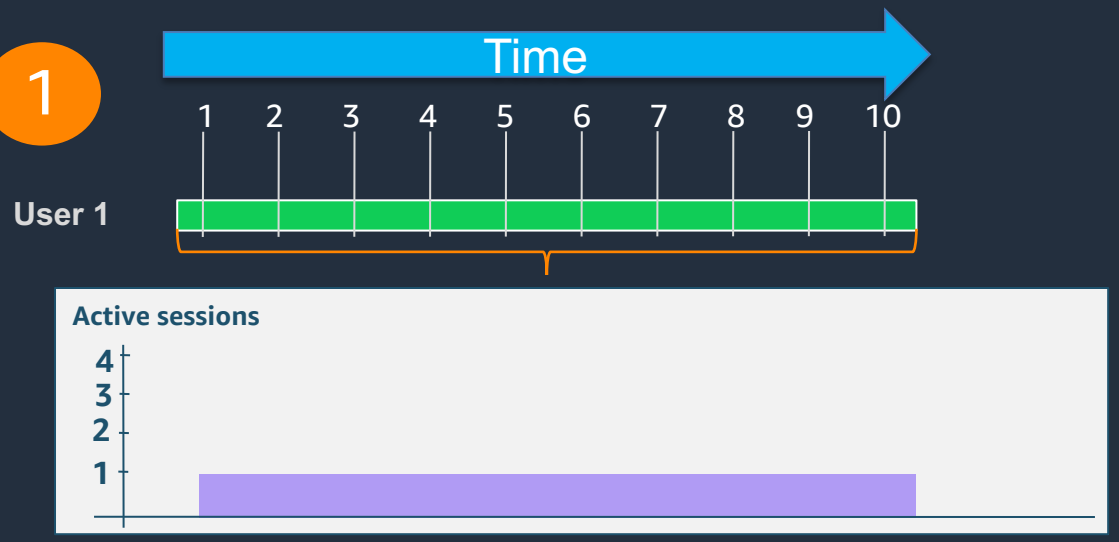
# Sampling backend activity

# Logging slow queries

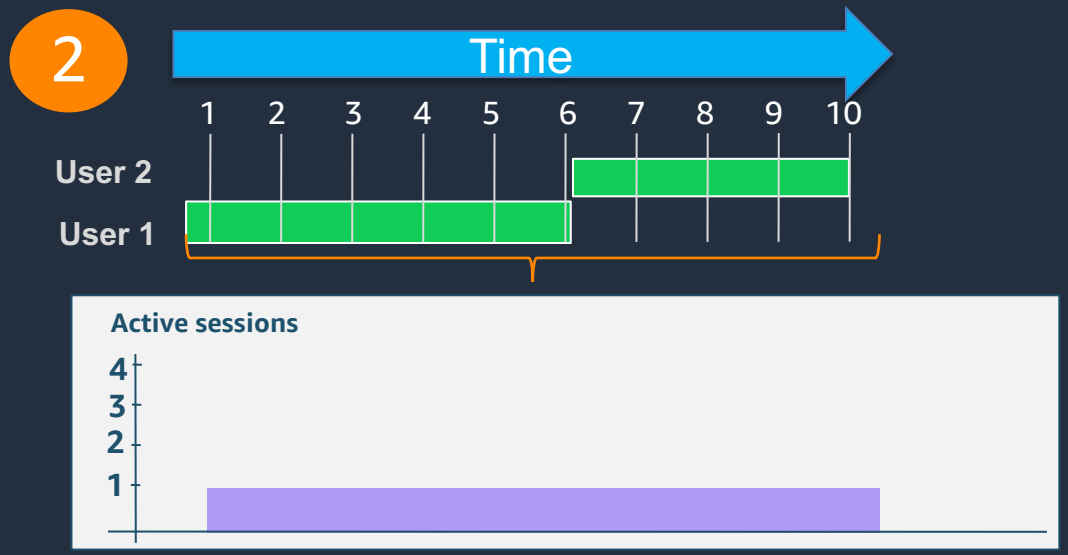


# Sampling sessions

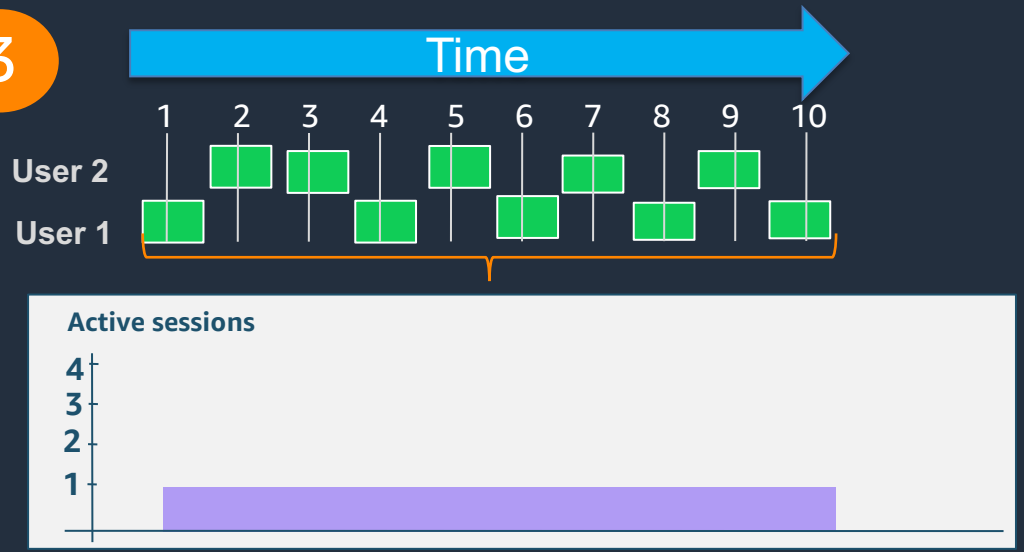
1



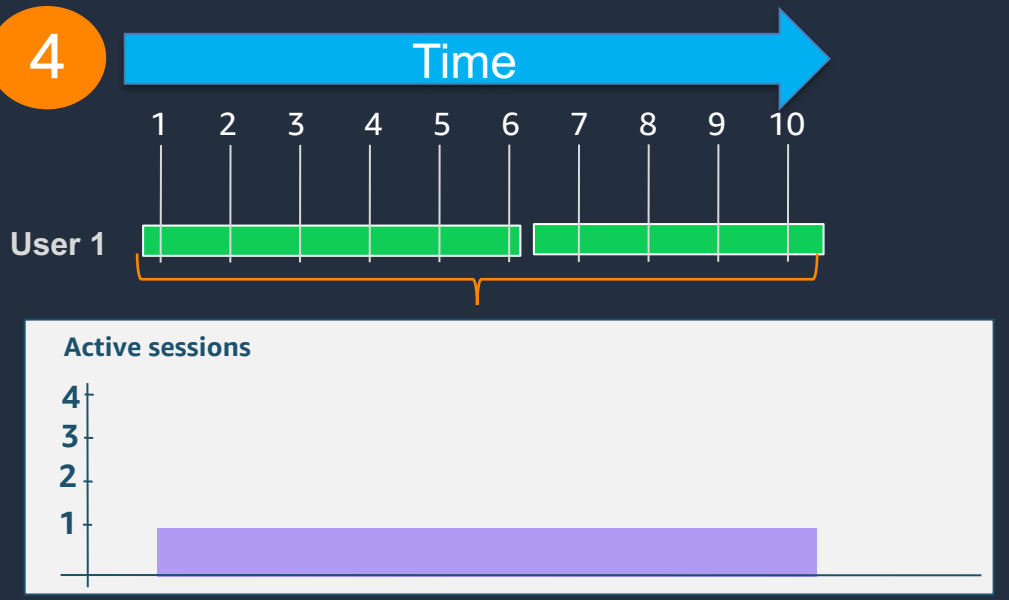

2




3

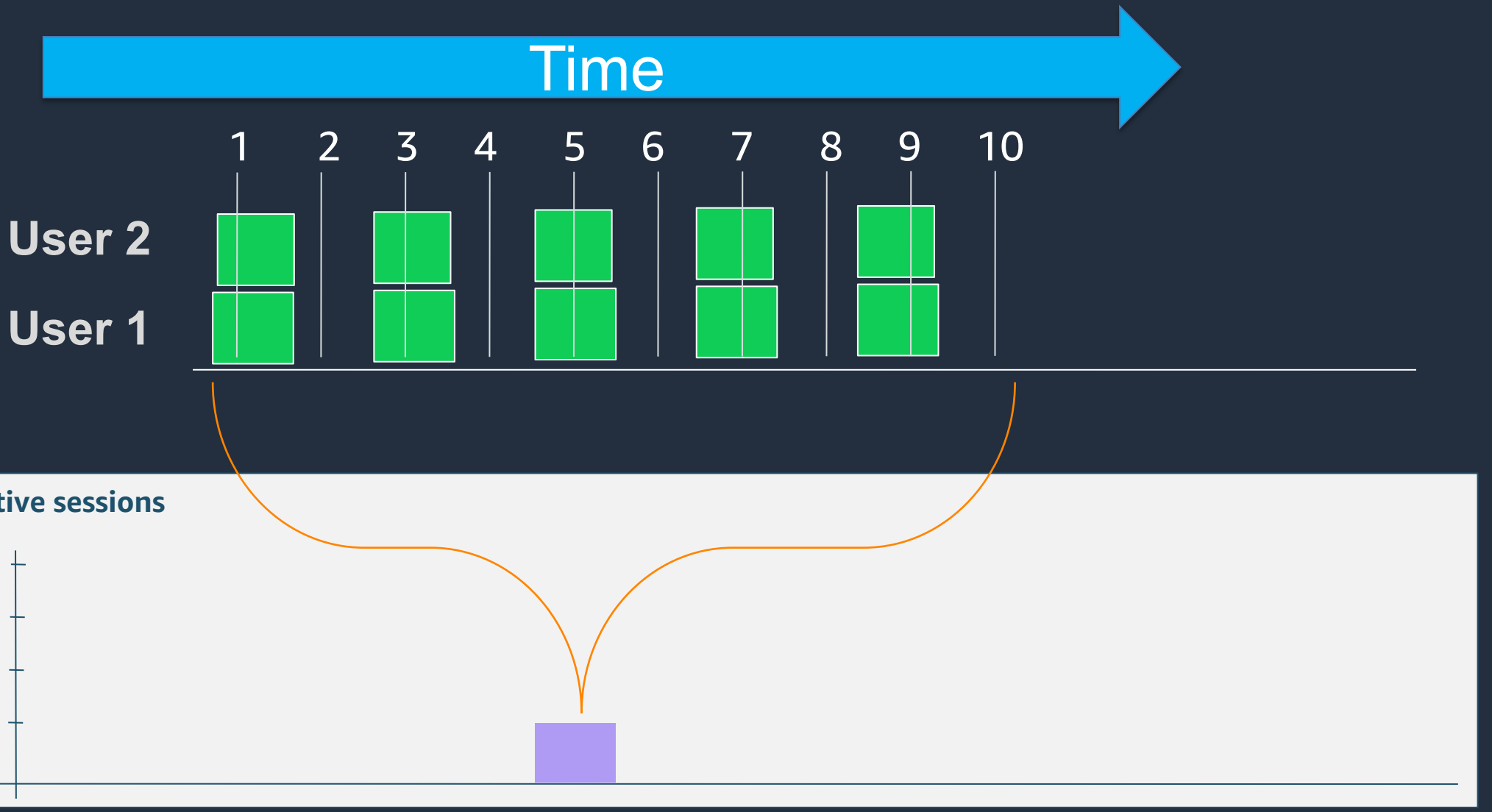



4

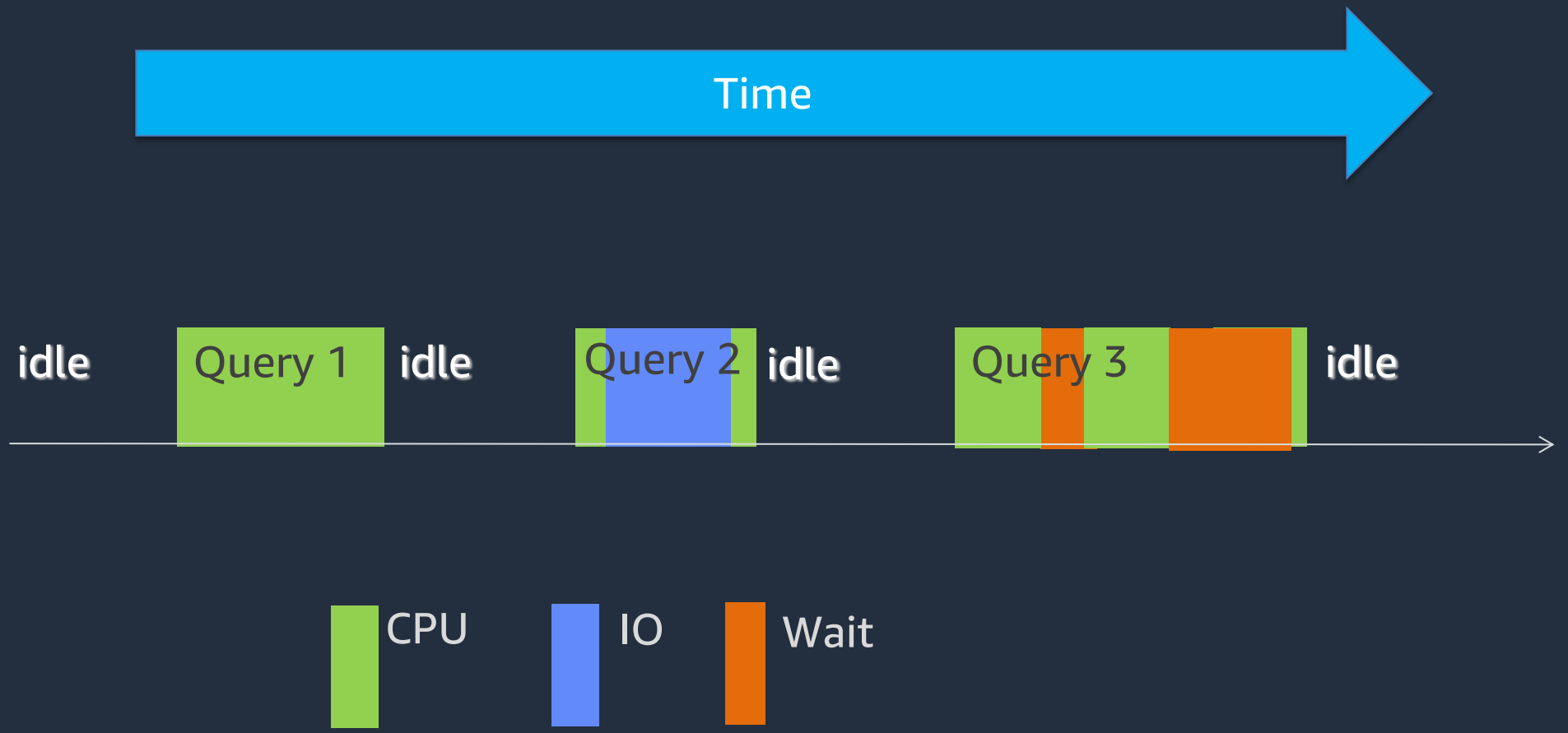


# How often should you sample?

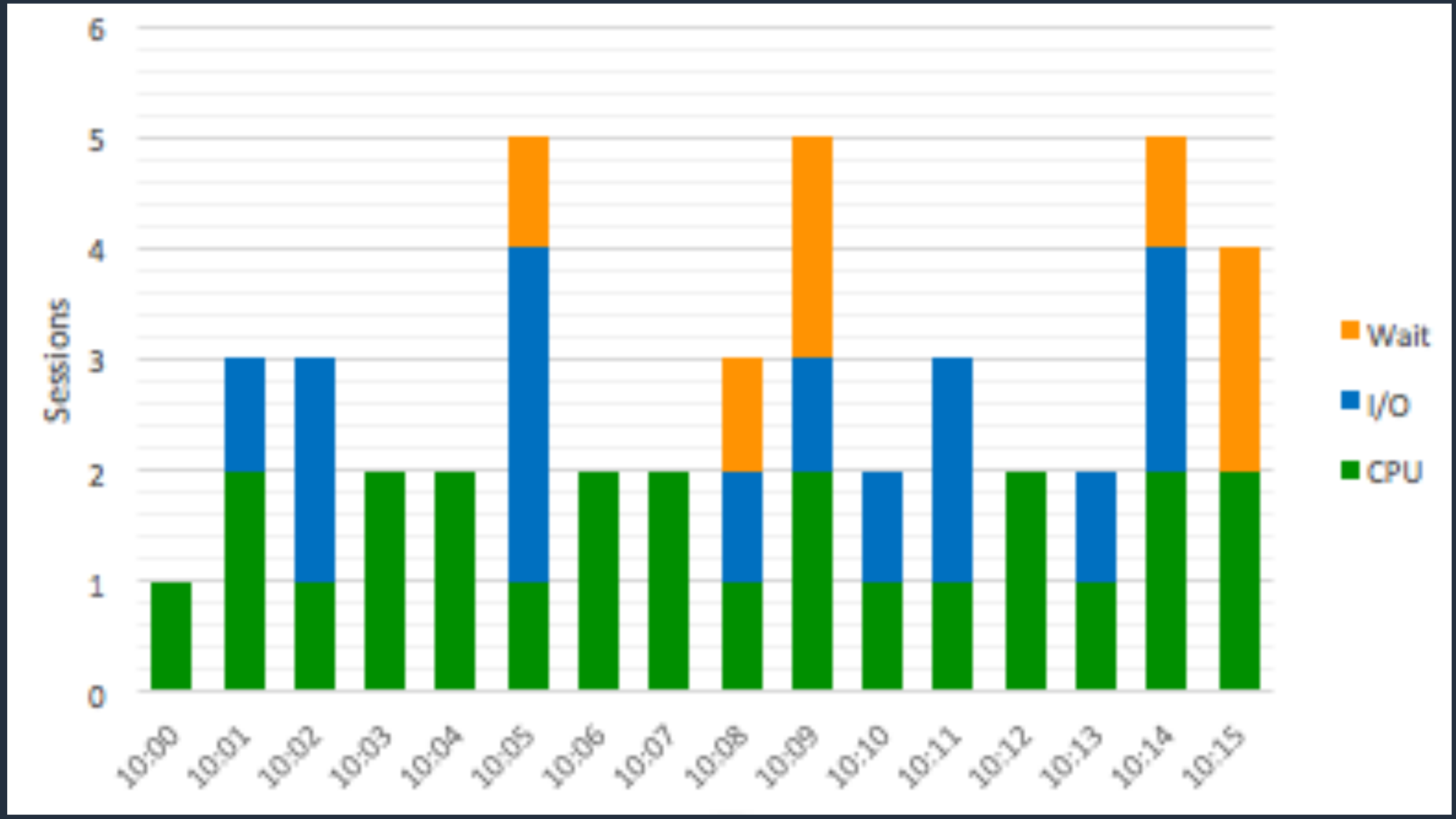
# Average Active Sessions



# Active session state

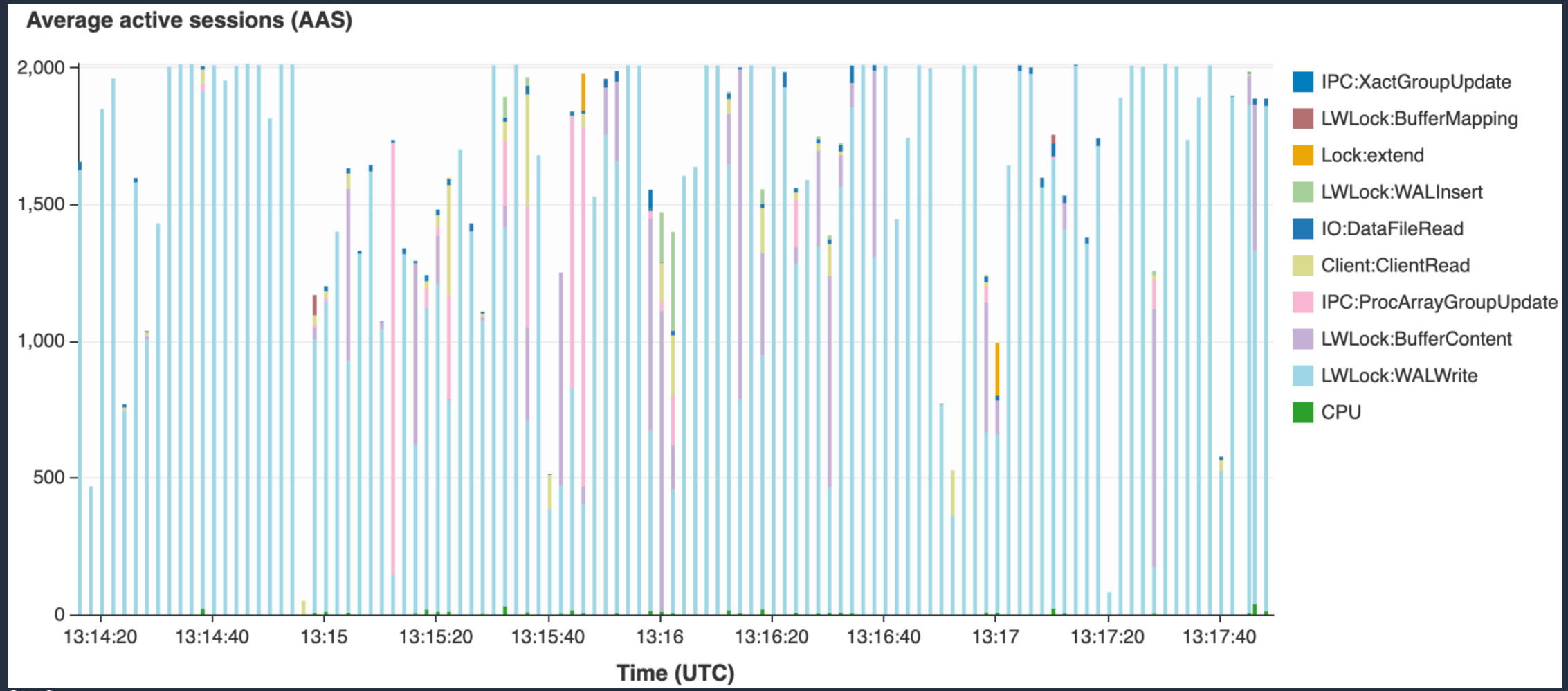


# Average active session by session state



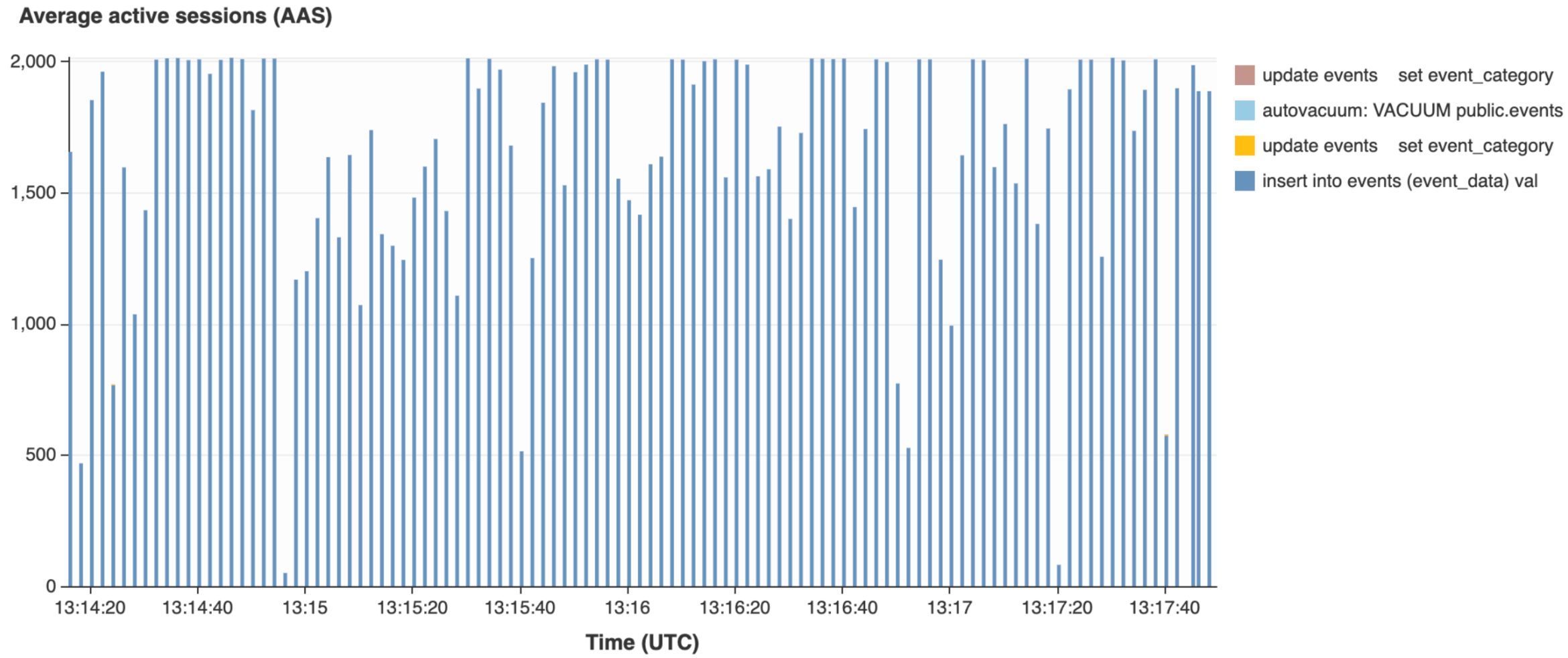
# What’s causing bottleneck in the workload?

## Average active session sliced by wait events



# Where should I focus tuning efforts?

## Average active session sliced by queries



# Where is the query stuck?

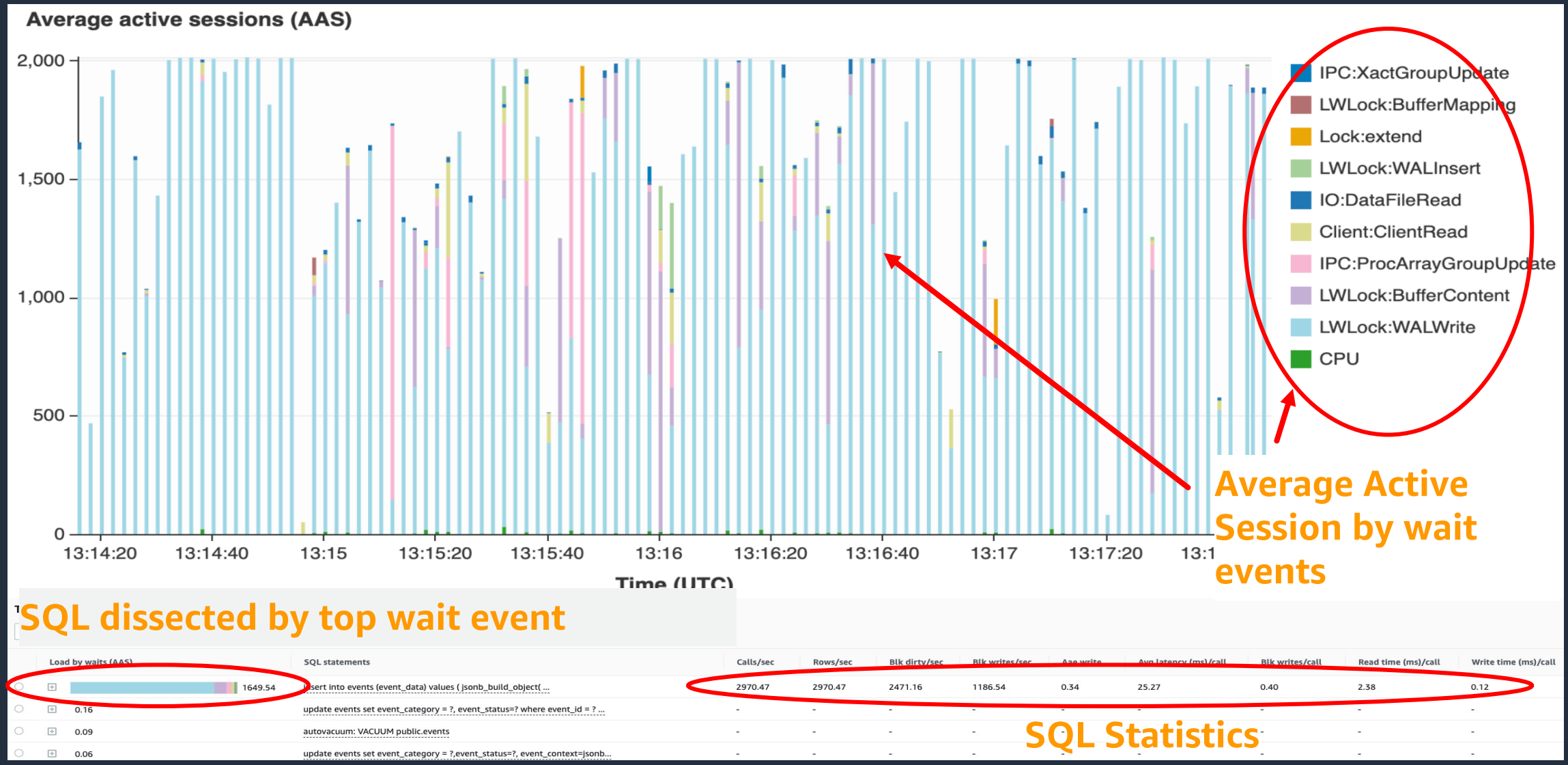
## Top queries sliced by wait events

| Load by waits (AAS) | SQL statements |
| --- | --- |
|  1649.54 | insert into events (event_data) values ( jsonb_build_object( ... |
| 0.16 | update events set event_category = ?, event_status=? where event_id = ? ... |
| 0.09 | autovacuum: VACUUM public.events |
| 0.06 | update events set event_category = ?,event_status=?, event_context=jsonb... |

# Include statement statistics

| SQL statements | Calls/sec | Rows/sec | Blk dirty/sec | Blk writes/sec | Aae write | Avg latency (ms)/call | Blk writes/call | Read time (ms)/call |
|---|---|---|---|---|---|---|---|---|
| insert into events (event_data) values ( jsonb_build_object( ... | 2970.47 | 2970.47 | 2471.16 | 1186.54 | 0.34 | 25.27 | 0.40 | 2.38 |
| update events set event_category = ?, event_status=? where event_id = ? ... | - | - | - | - | - | - | - | - |
| autovacuum: VACUUM public.events | - | - | - | - | - | - | - | - |
| update events set event_category = ?,event_status=?, event_context=jsonb... | - | - | - | - | - | - | - | - |

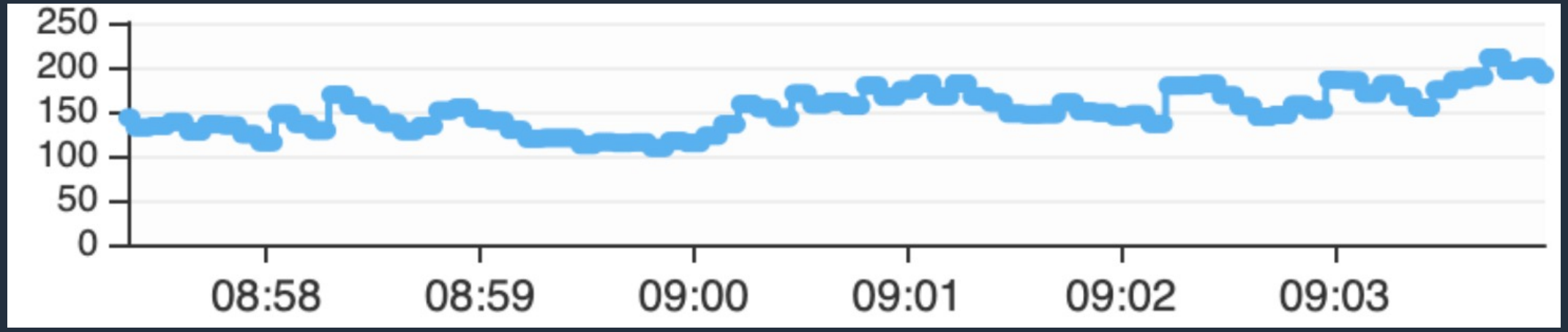
# Putting it all together



# Scenario: Increase in average load

# High load on host

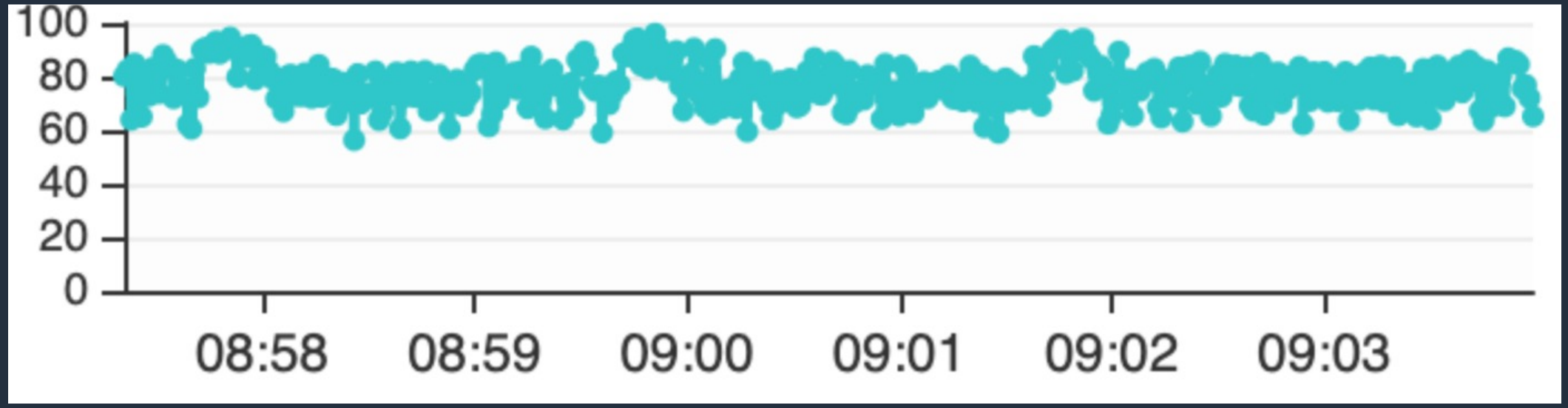
## Load average for 1 minutes



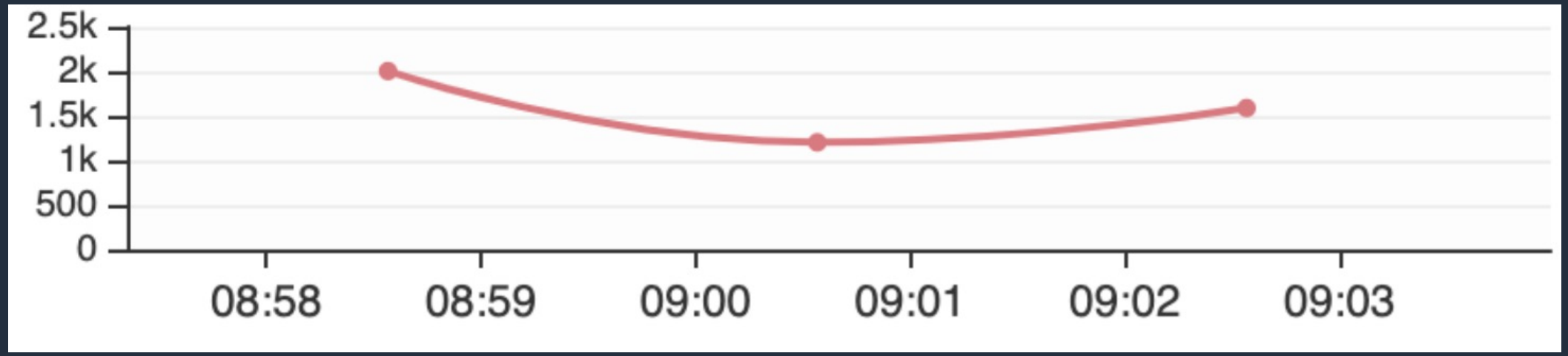
# Workloads notes

- Nothing has changed in the workload
- The workload is meant to process events which are collected through sensors
- Sensors send data collected – INSERT INTO events
- Events are analyzed and updated – UPDATE events
- Events table has two jsonb columns
- 1 Primary key and 3 secondary indexes (expression based)

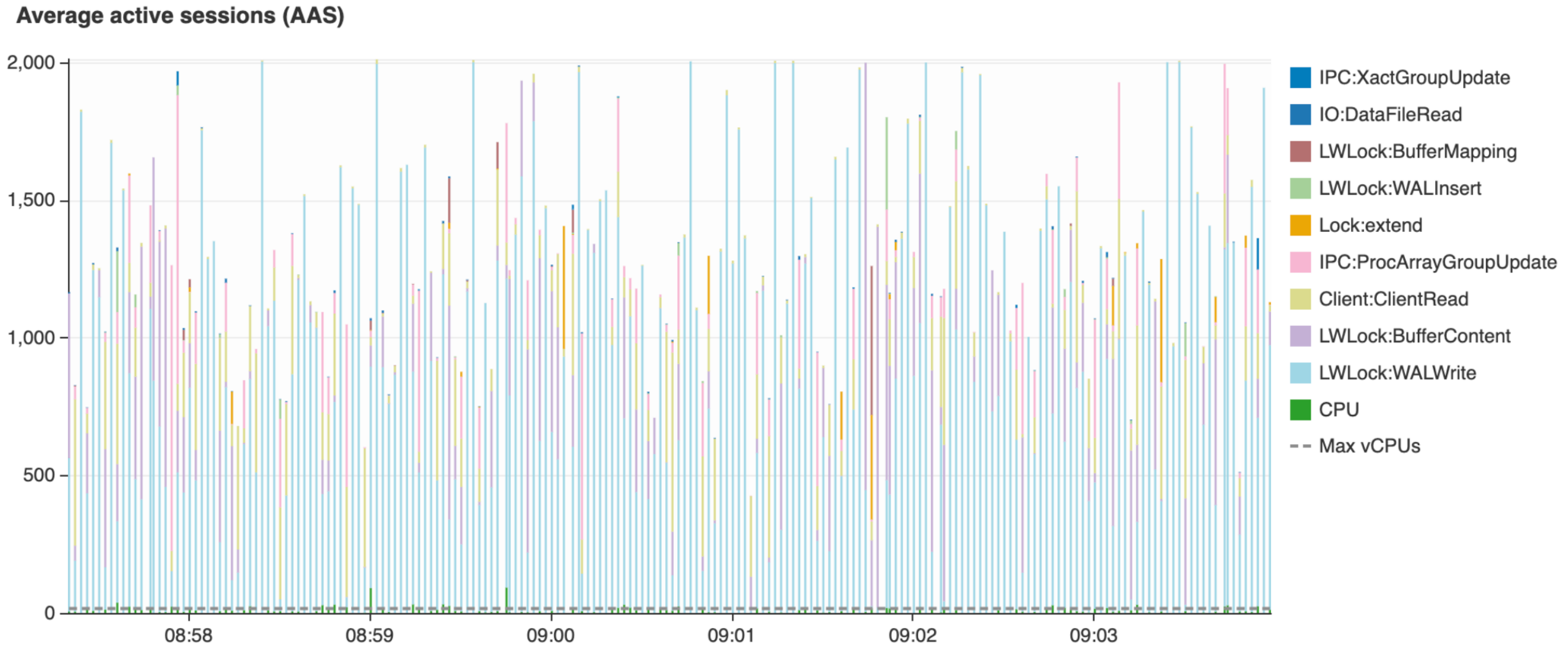
# CPU Utilization



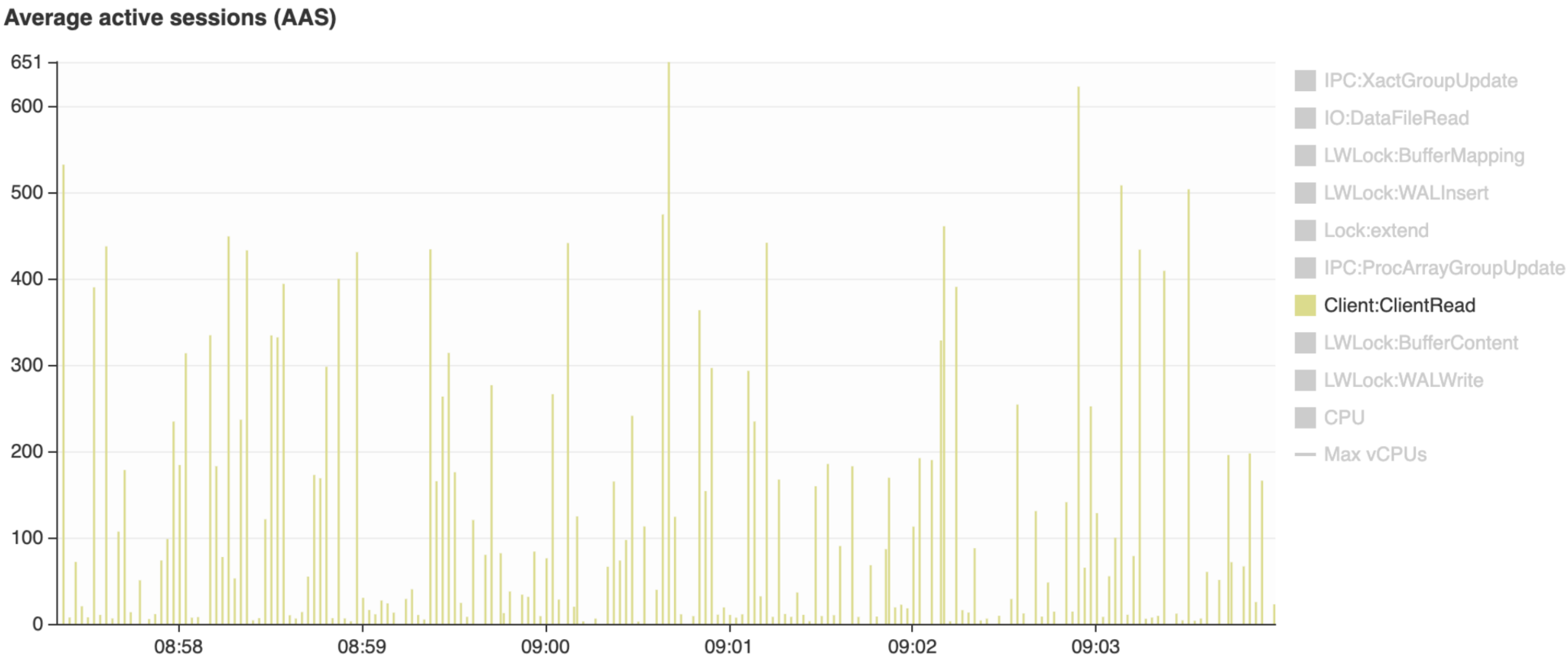

# Active sessions



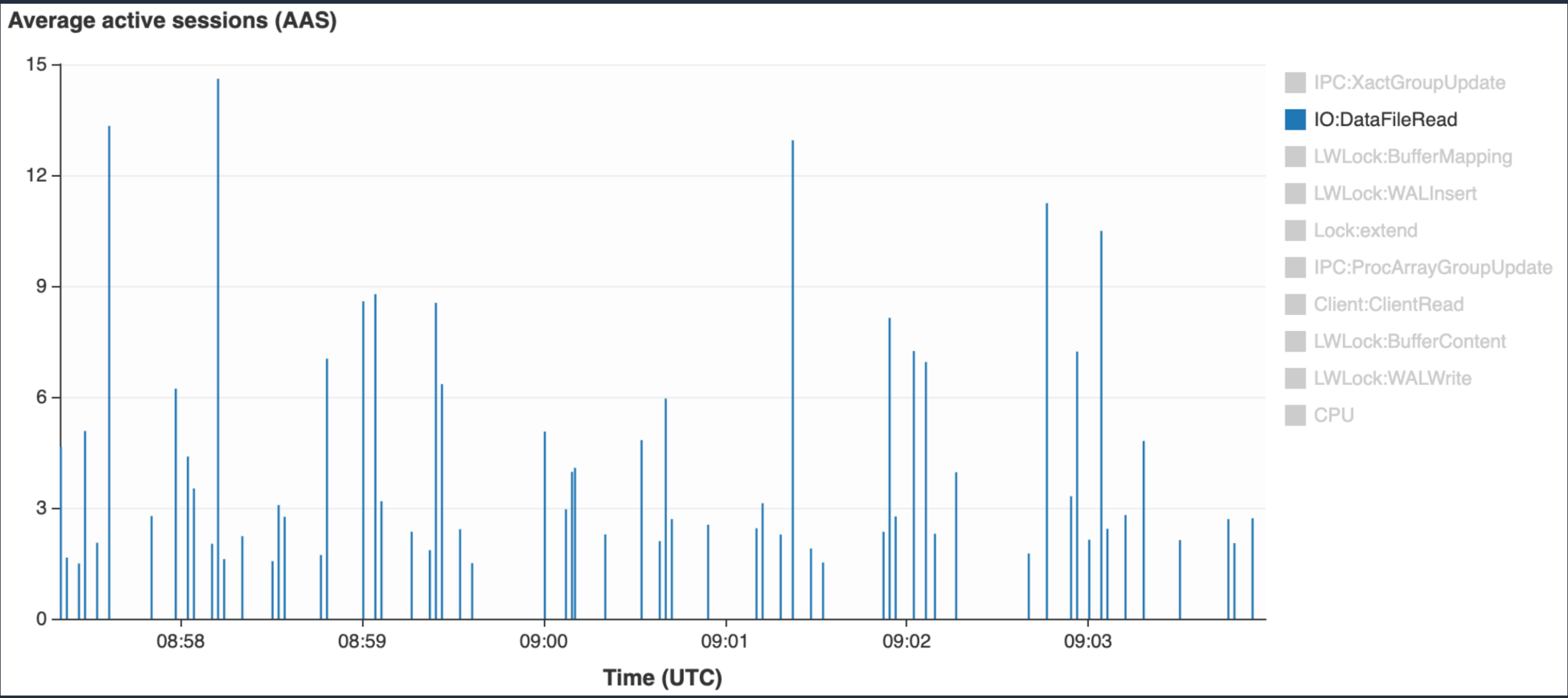
# Let’s slice the average active sessions by wait events



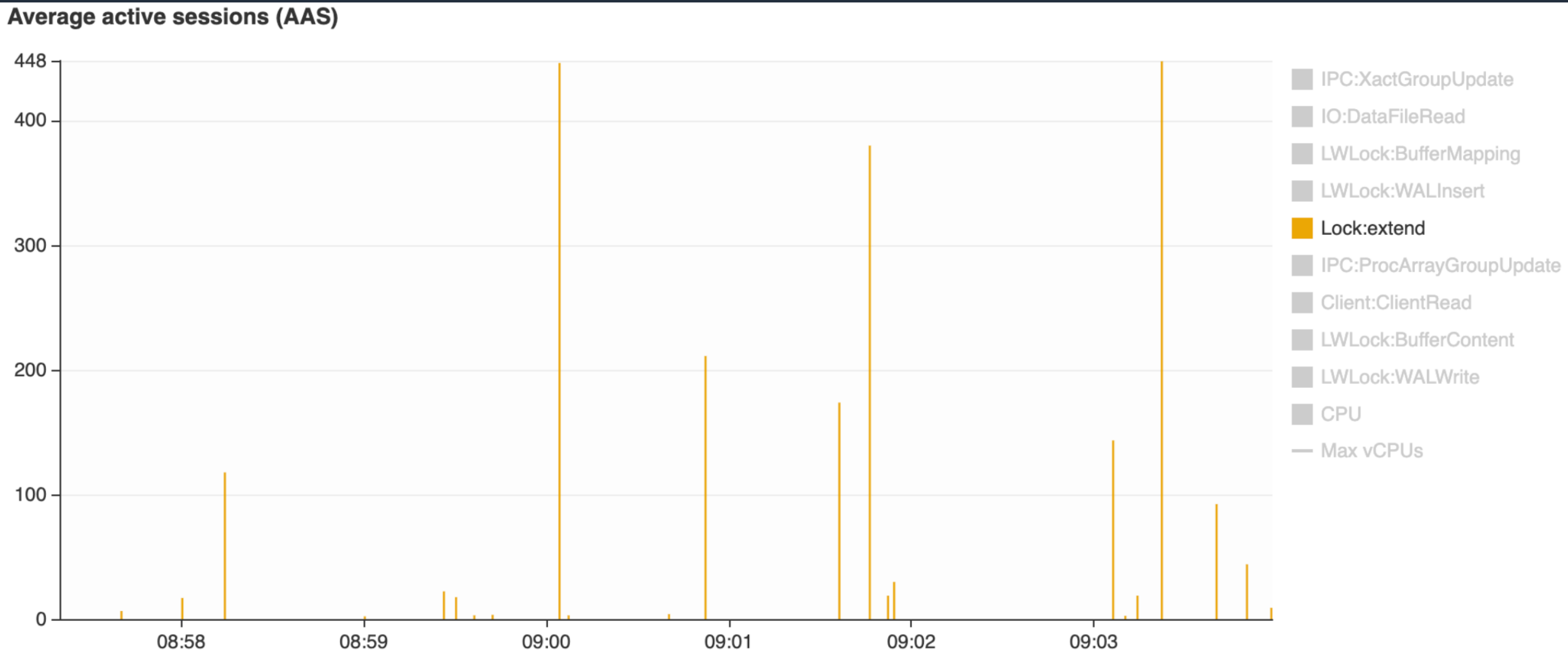
# Client:ClientRead



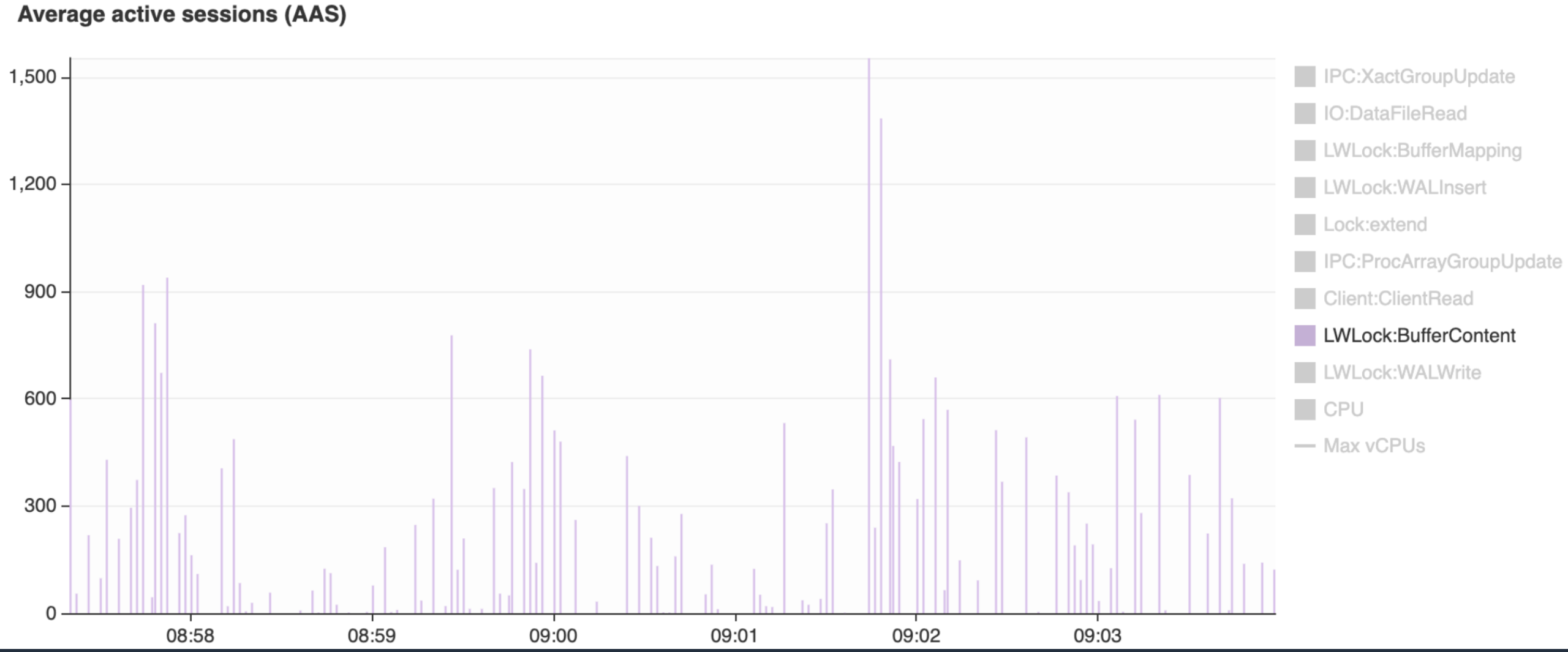
# IO:DataFileRead



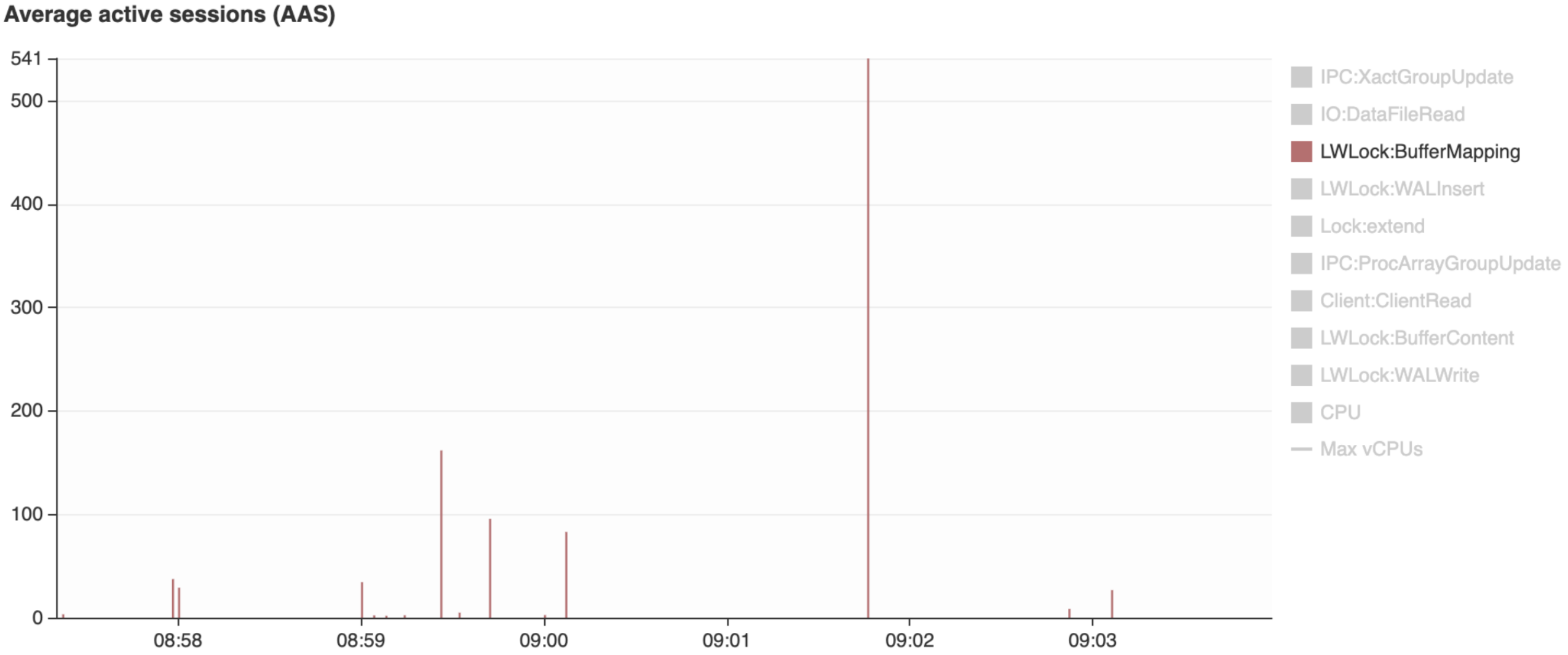
# Lock:Extend



# LwLock:BufferContent

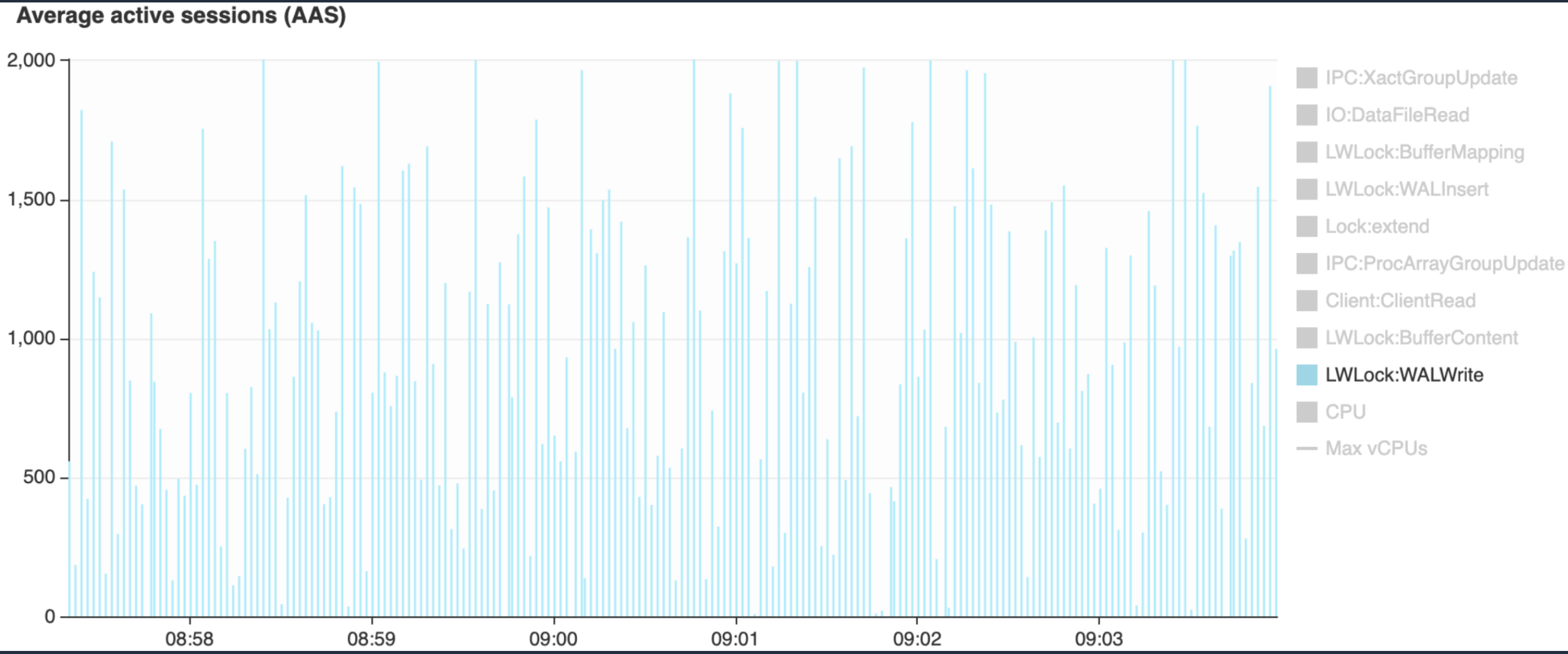


# LwLock:BufferMapping

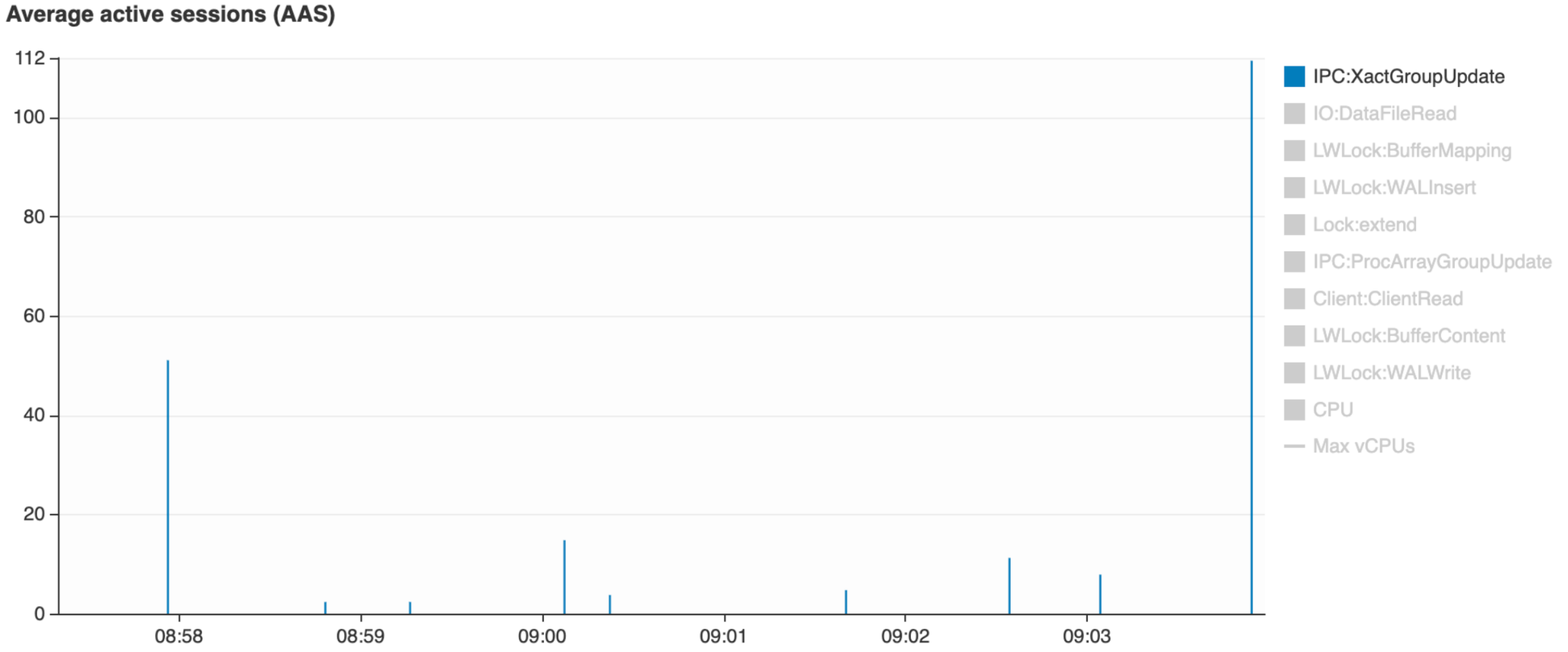


# LwLock:WALInsert

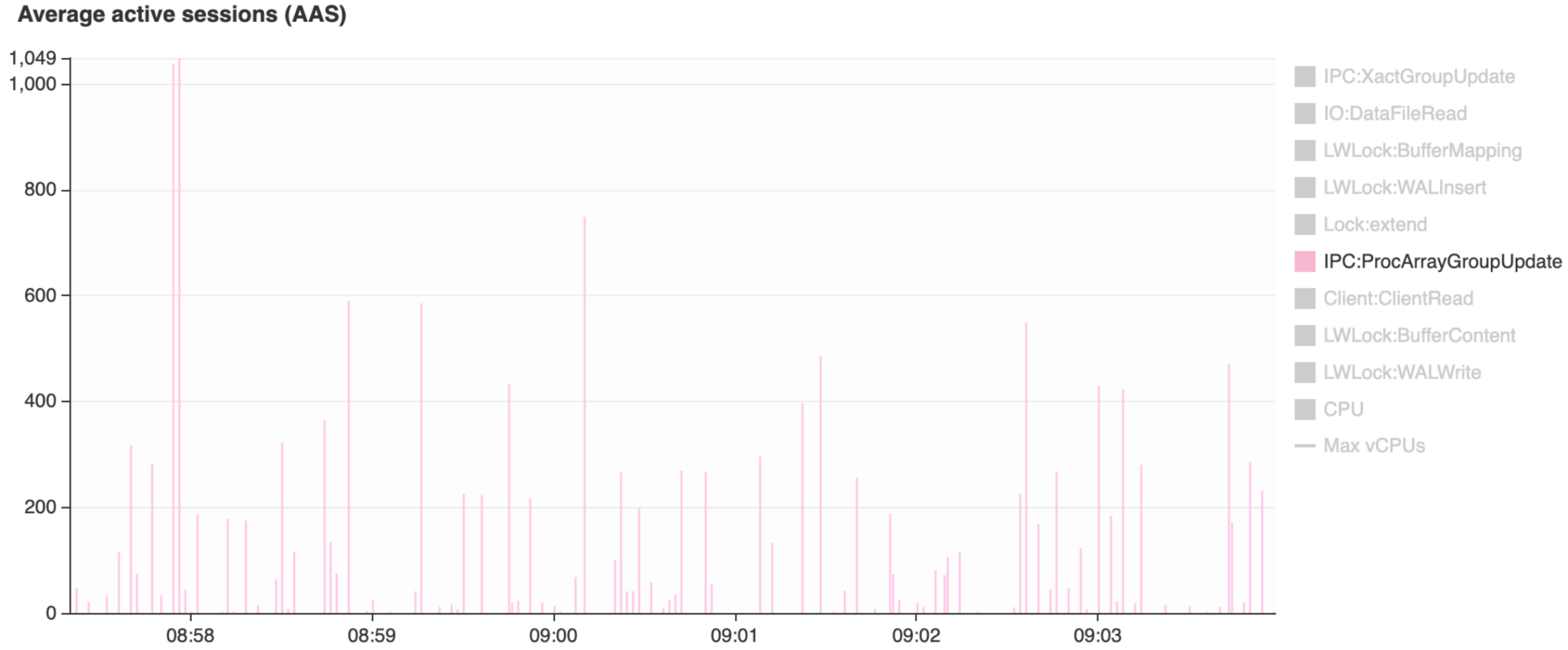
# LwLock:WALWrite



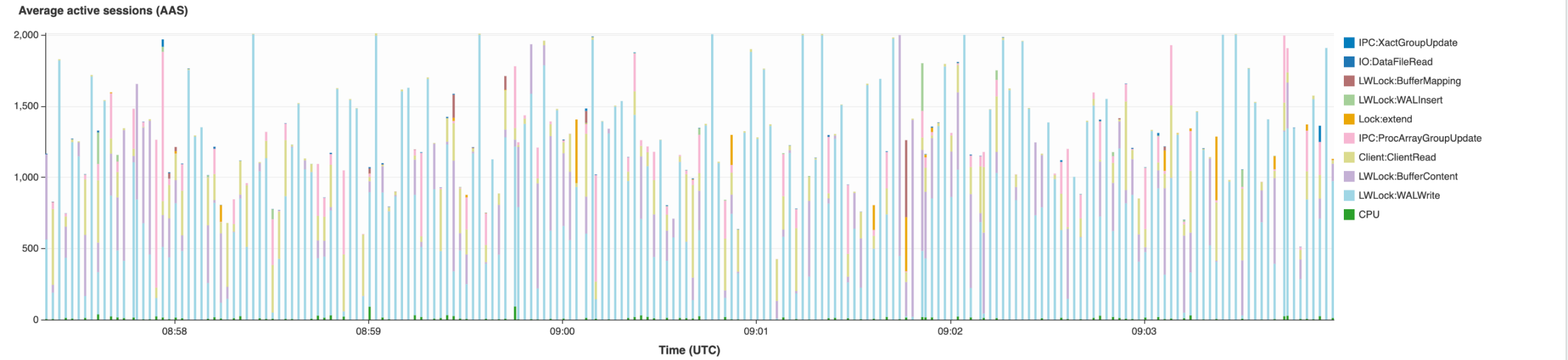
# IPC:XactGroupUpdate



# IPC: ProcArrayGroupUpdate



# Let’s review the top wait event again




Top wa… | …op users | Top session types | Top applications | Top databases

Legend

| | |
|---|---|
| CPU | 7.17 |
| LWLock:WALWrite | 892.93 |
| LWLock:BufferContent | 169.86 |
| Client:ClientRead | 129.26 |
| IPC:ProcArrayGroupUpdate | 84.90 |
| Lock:extend | 12.22 |
| LWLock:WALInsert | 5.32 |
| Other | 9.11 |

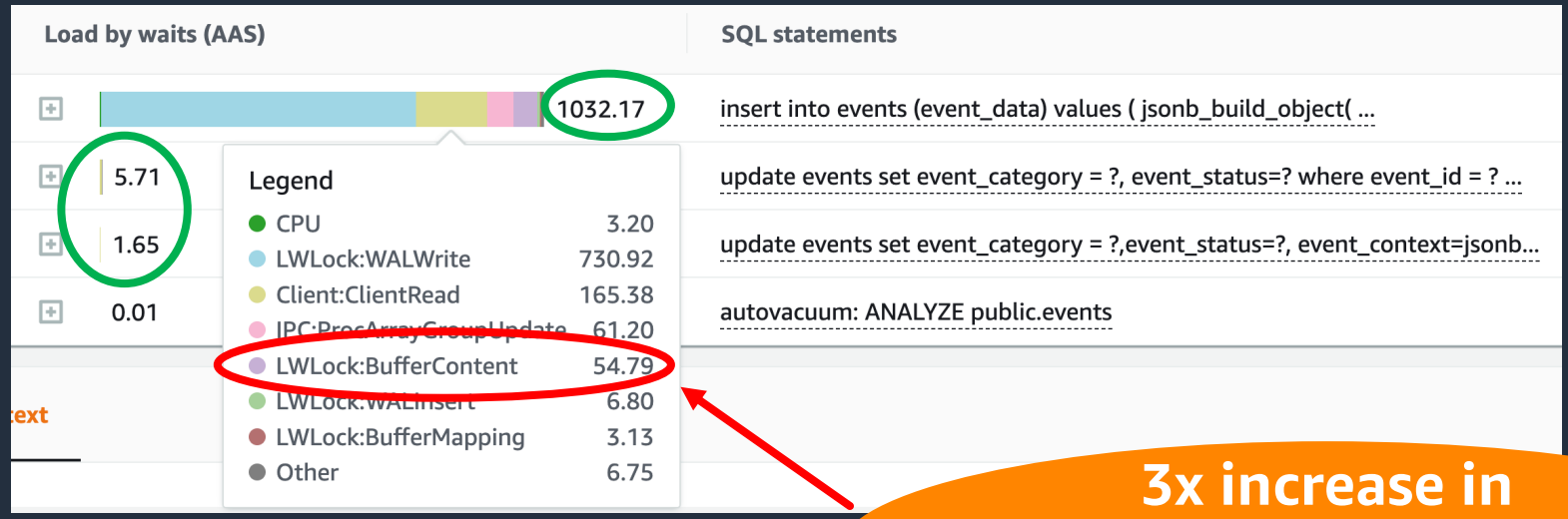
Top S…

1

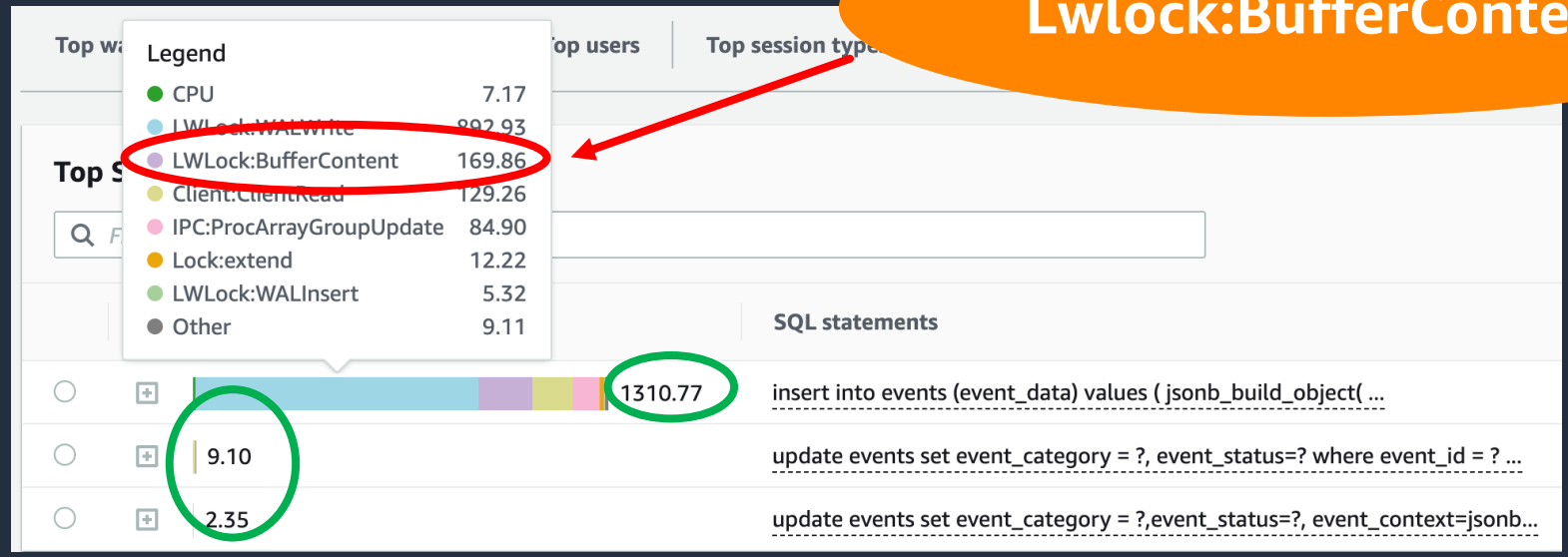
| | SQL statements | Calls/sec | Rows/sec | Blk dirty/… | Blk write… | Aae write | Avg laten… | Blk |
|---|---|---|---|---|---|---|---|---|
| 1310.77 | insert into events (event_data) values ( jsonb_build_object( … | 5197.14 | 5197.14 | 4804.44 | 373.03 | 0.00 | 22.91 | 0.0 |

# What changed: Slice top SQL by wait events

**Just before the performance degradation**

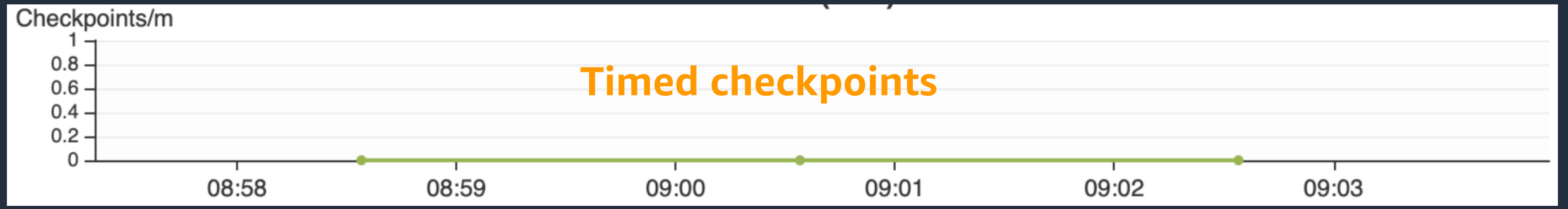


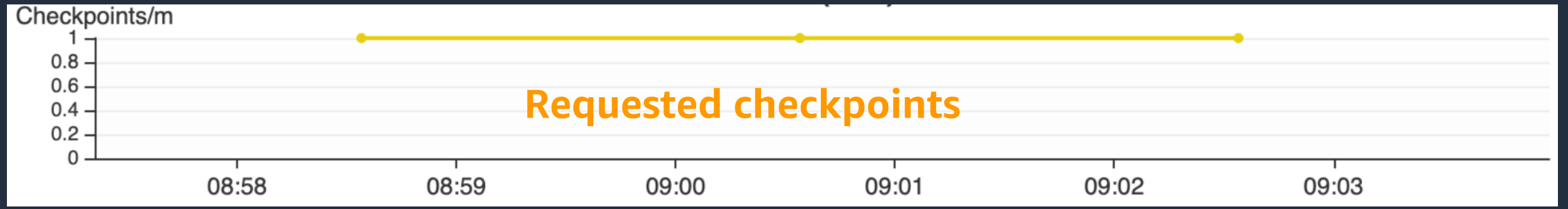

**During the performance degradation**




**3x increase in Lwlock:BufferContent**

# Checkpoint and data file writes




# Workload tuning

- Tune autovacuum
- Manage concurrency
- Reduce working set
- Tune fill factor
- Partitioning
- Reduce number of indexes

# Conclusion

## Wait events give great insights

- pg_stat_activity just gives a snapshot → like clicking a picture
- Take snapshots of pg_stat_activity → create a movie
- Compare and see what changed

## Capture additional metrics

- pg_stat_statements
- pg_stat_database
- pg_stat_user_*
- pg_locks
- pg_stat_bgwriter

## Enable logging

- log_min_duration_statement
- log_lock_waits
- log_checkpoints
- log_temp_files
- log_autovacuum_min_duration

# Thank you!

Sameer Kumar
kusme@amazon.com

Roneel Kumar
roneelk@amazon.com