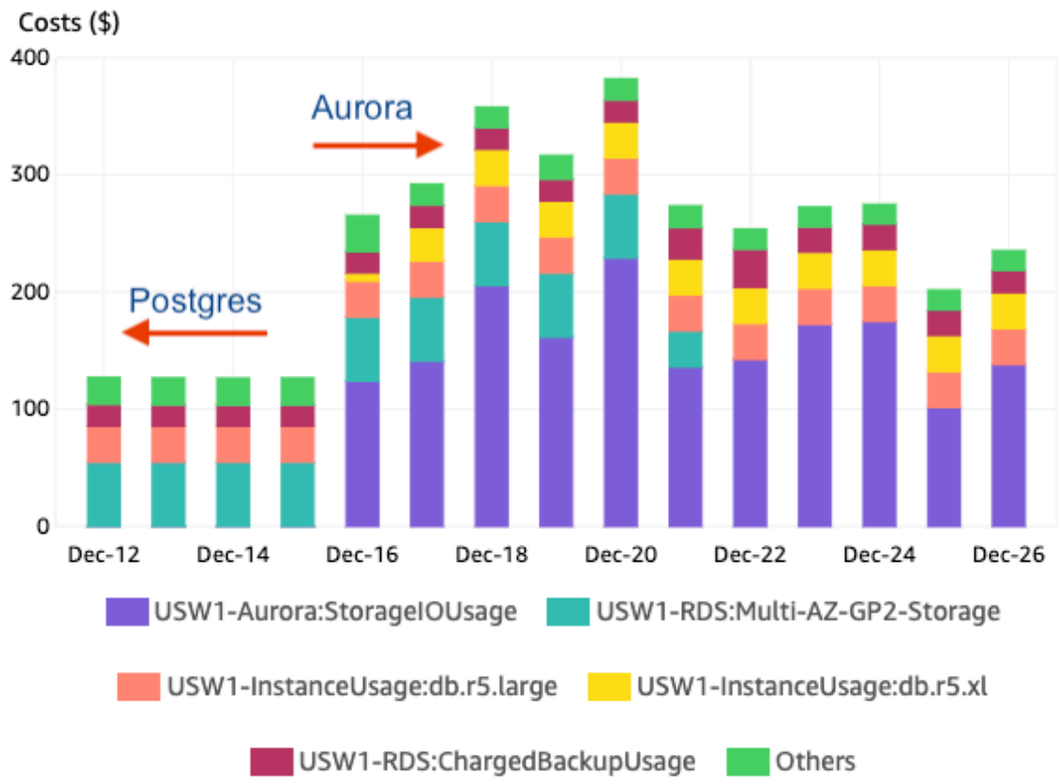
Optimizing Postgres I/O Performance and Costs

@LukasFittl

Why does I/O matter?



“Migrating to Aurora: easy except the bill"

pganalyze

What We'll Talk About Today

1. I/Os, Small and Large

2. The Role of Shared Buffers and the WAL

3. A Tale of Checkpoints and Full Page Writes

4. Amazon Aurora Is Different (But Not Always Better)

5. Breaking Down Your Query Workload Into I/Os

6. Indexes and Write Amplification

7. VACUUM Tuning and Table Bloat

pganalyze

I/Os, Small and Large

2020

1ns

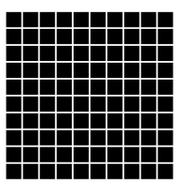
L1 cache reference: 1ns

Branch mispredict: 3ns

L2 cache reference: 4ns

Mutex lock/unlock: 17ns

100ns =



Main memory reference: 100ns

1,000ns ≈ 1µs

Compress 1KB wth Zippy: 2,000ns ≈ 2µs

10,000ns ≈ 10µs =

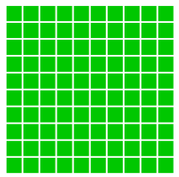
Send 2,000 bytes over commodity network: 22ns

SSD random read: 16,000ns ≈ 16µs

Read 1,000,000 bytes sequentially from memory: 2,000ns ≈ 2µs

Round trip in same datacenter: 500,000ns ≈ 500µs

1,000,000ns = 1ms =

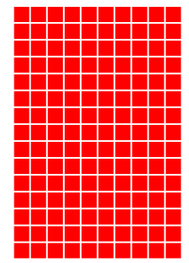


Read 1,000,000 bytes sequentially from SSD: 31,000ns ≈ 31µs

Disk seek: 2,000,000ns ≈ 2ms

Read 1,000,000 bytes sequentially from disk: 625,000ns ≈ 625µs

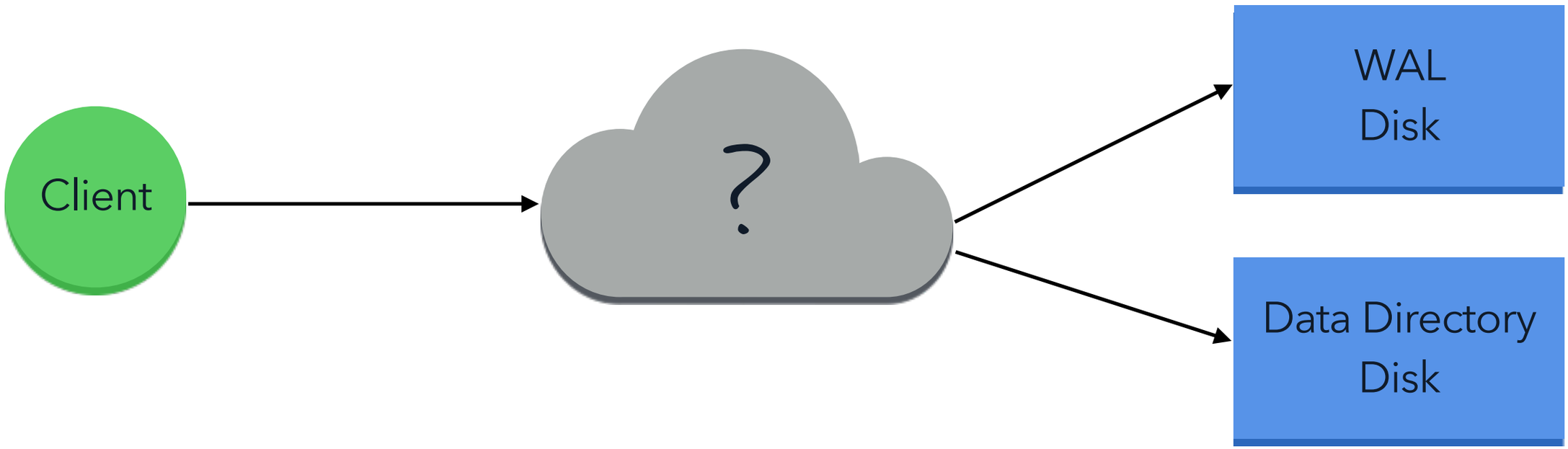
Packet roundtrip CA to Netherlands: 150,000,000ns ≈ 150ms



Latency Numbers Every Programmer Should Know

Write I/O

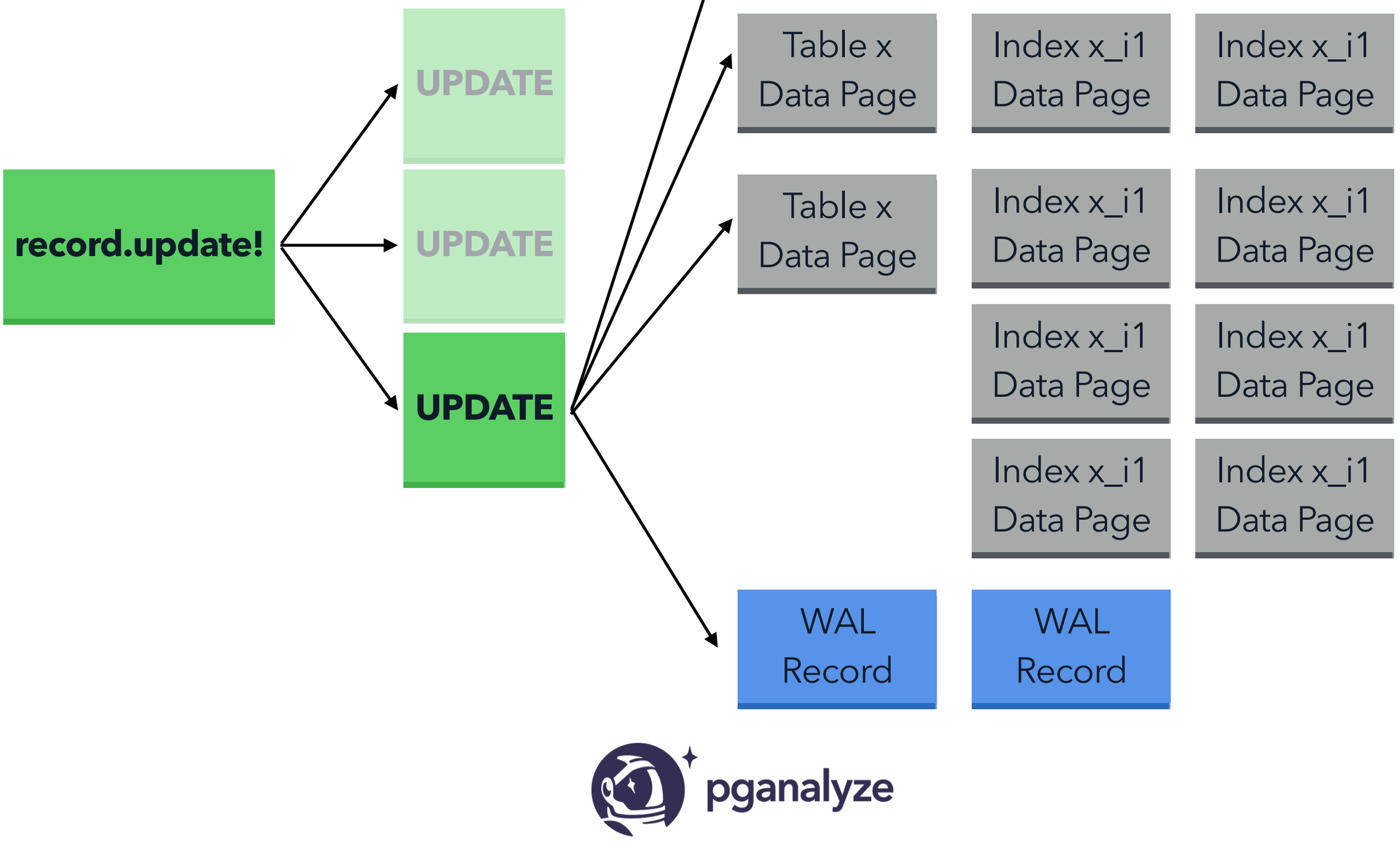
Change that needs to be written to disk



🔧 Change buffer sizes, flush intervals, reduce index writes, etc.

pganalyze

Write I/O



pganalyze

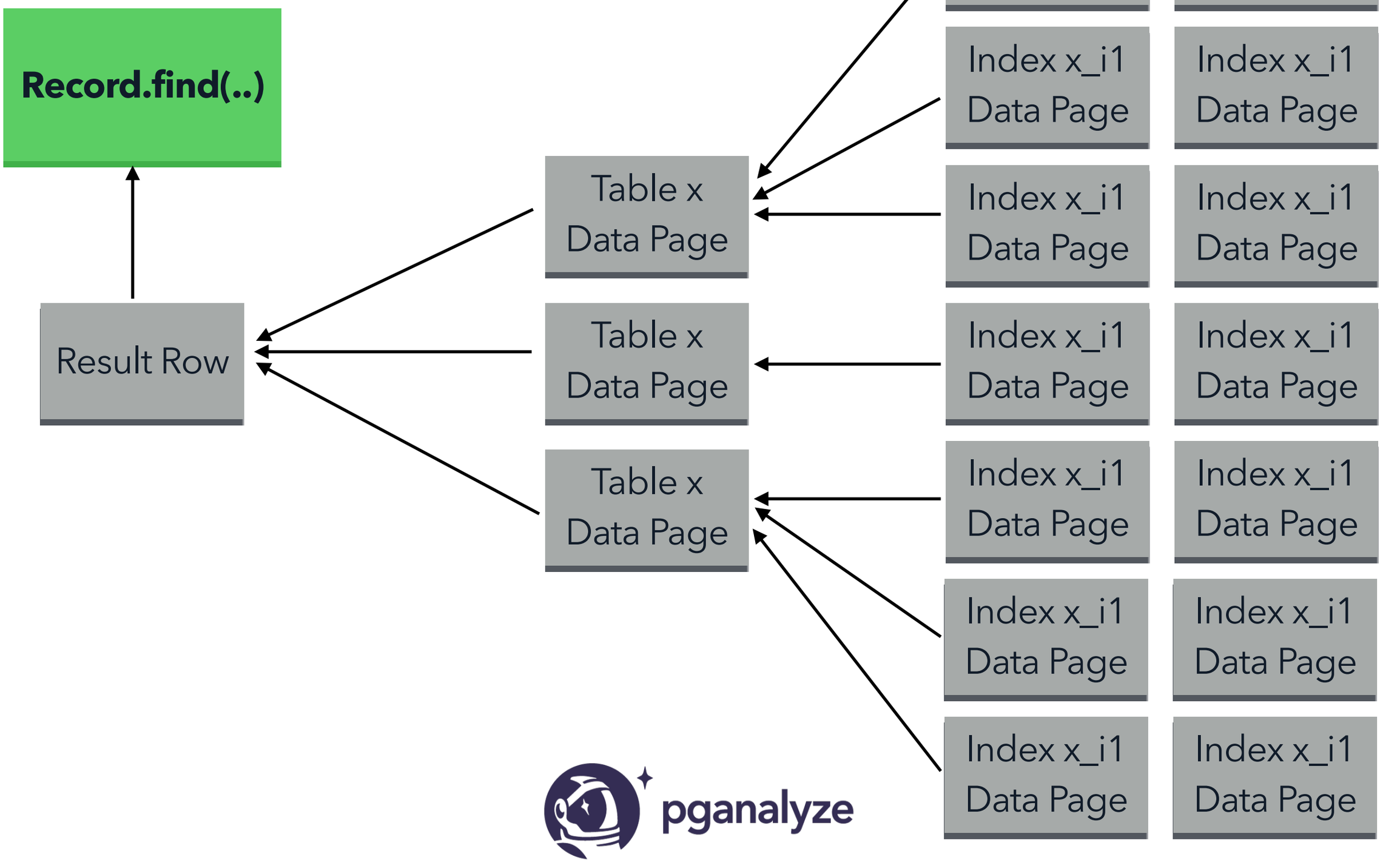
Read I/O

Provide a certain query result to the client

🔧 Improve parallelism, improve indexes, improve caching

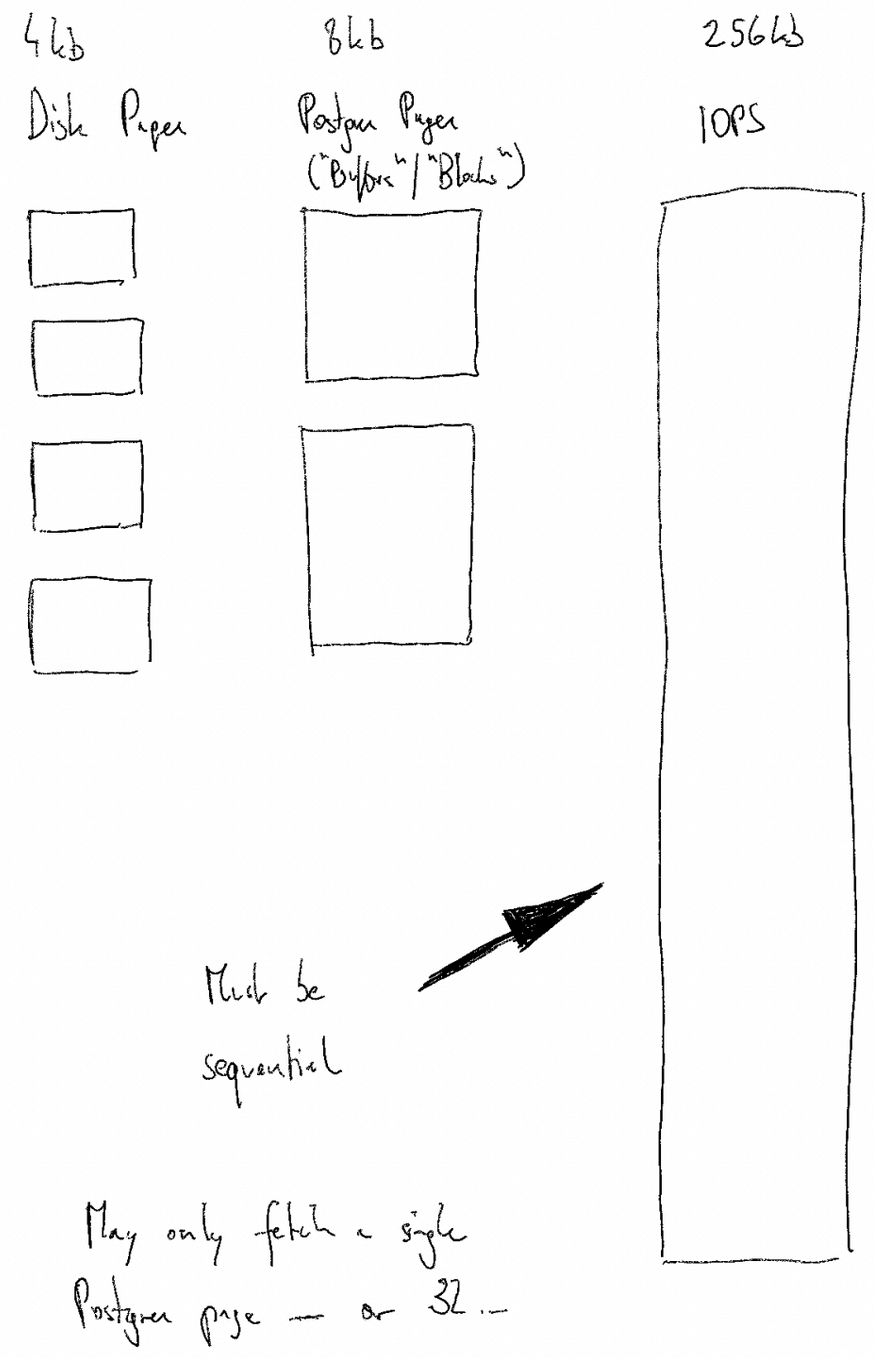
pganalyze

Read I/O

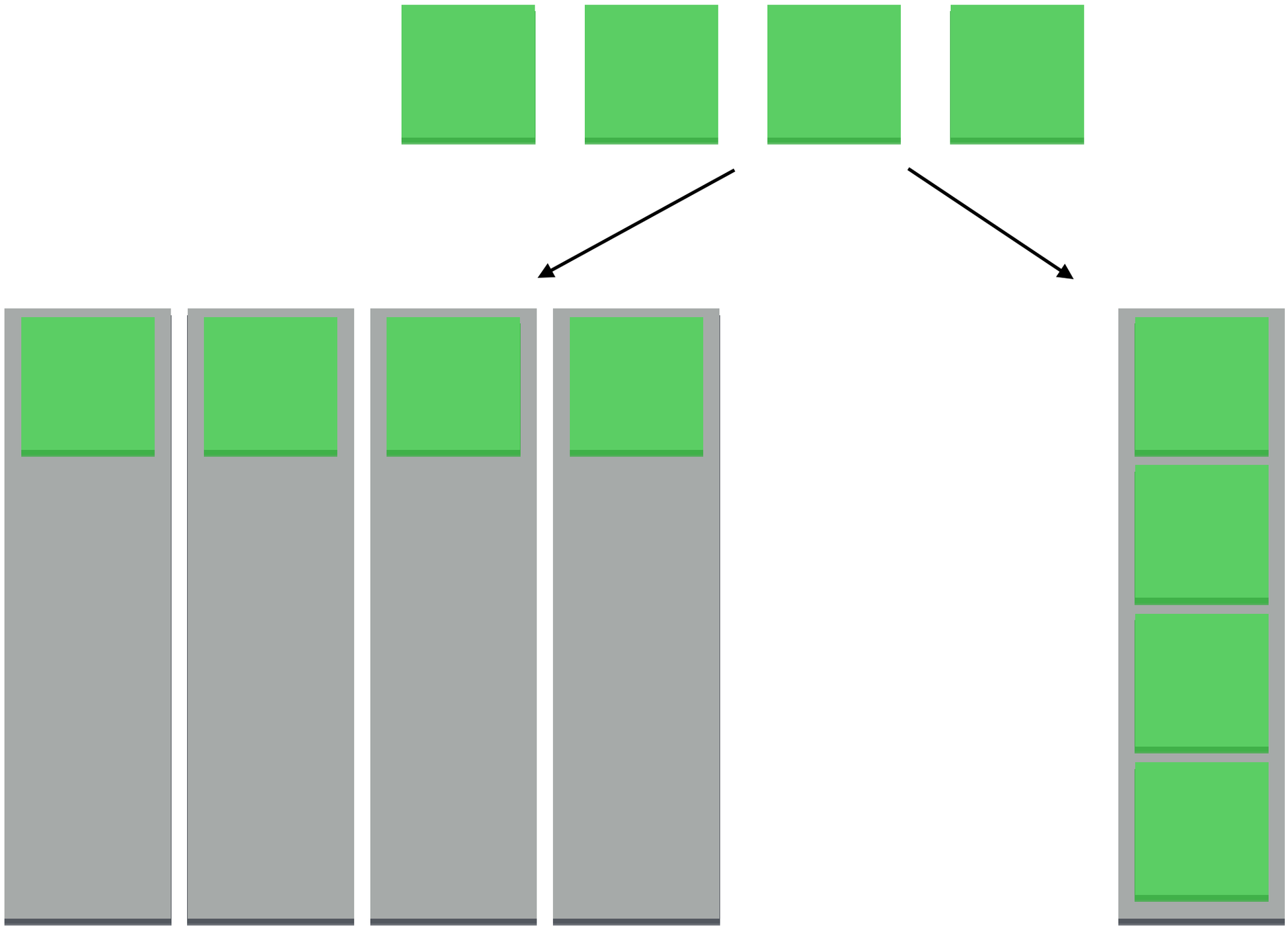

Record.find(..)
Result Row
Table x
Data Page
Table x
Data Page
Table x
Data Page
Index x_i1
Data Page
Index x_i1
Data Page
Index x_i1
Data Page
Index x_i1
Data Page
Index x_i1
Data Page
Index x_i1
Data Page
Index x_i1
Data Page
Index x_i1
Data Page
Index x_i1
Data Page
Index x_i1
Data Page
Index x_i1
Data Page
Index x_i1
Data Page
pganalyze

I/Os Can Contain More Than 1 Postgres Page

1 Postgres Page
= 2 Disk Pages (usually)



pganalyze



$X (PIOPS)

1/4 of $X (PIOPS)

pganalyze

To create more efficient I/O, we want to get **more work done with a single I/O operation**

The best I/O operation is

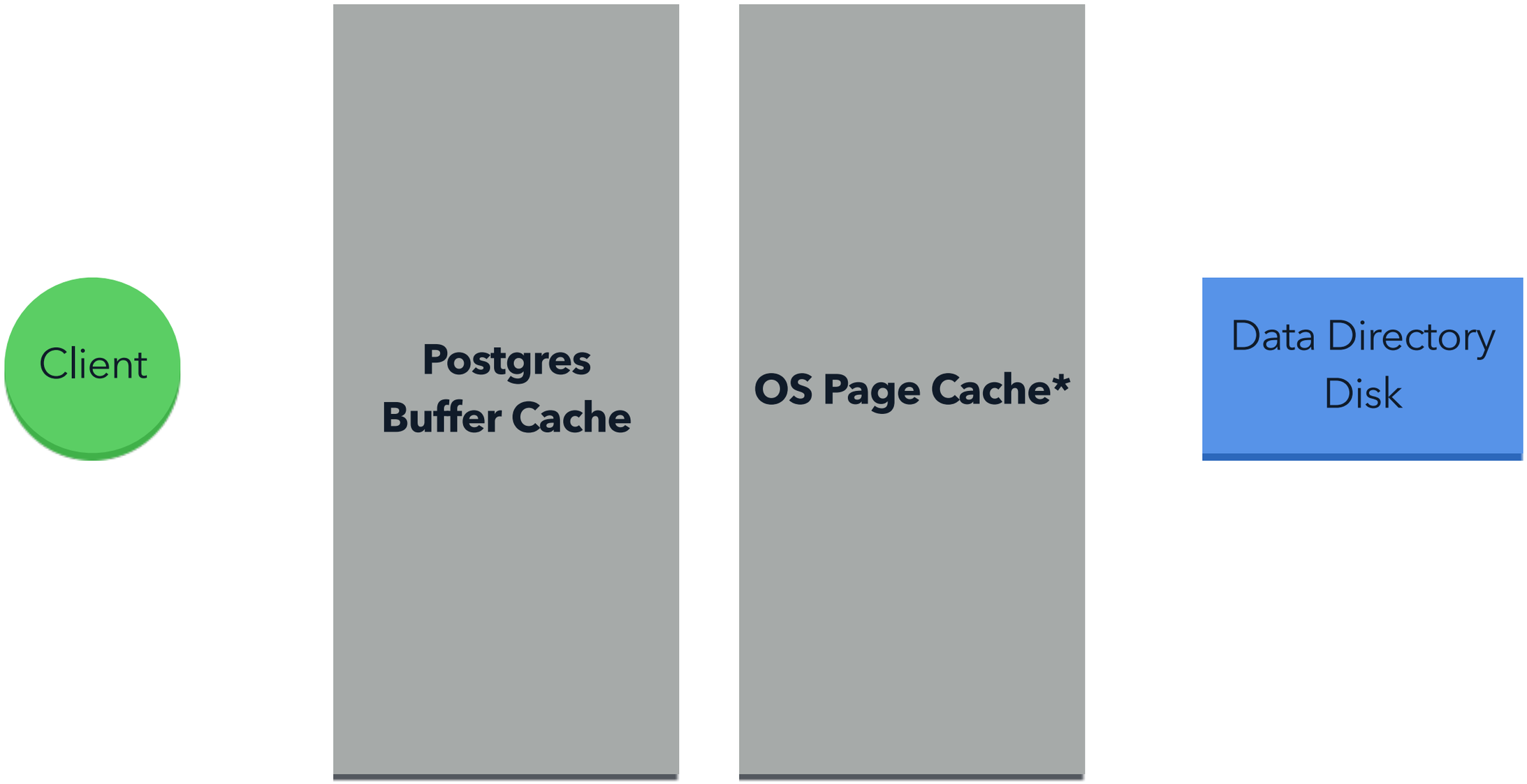
the one that didn't happen.

(e.g. because we avoided table bloat, thus not even having a particular data page)

pganalyze

The Role of Shared Buffers and the WAL

The Two Caches That Matter

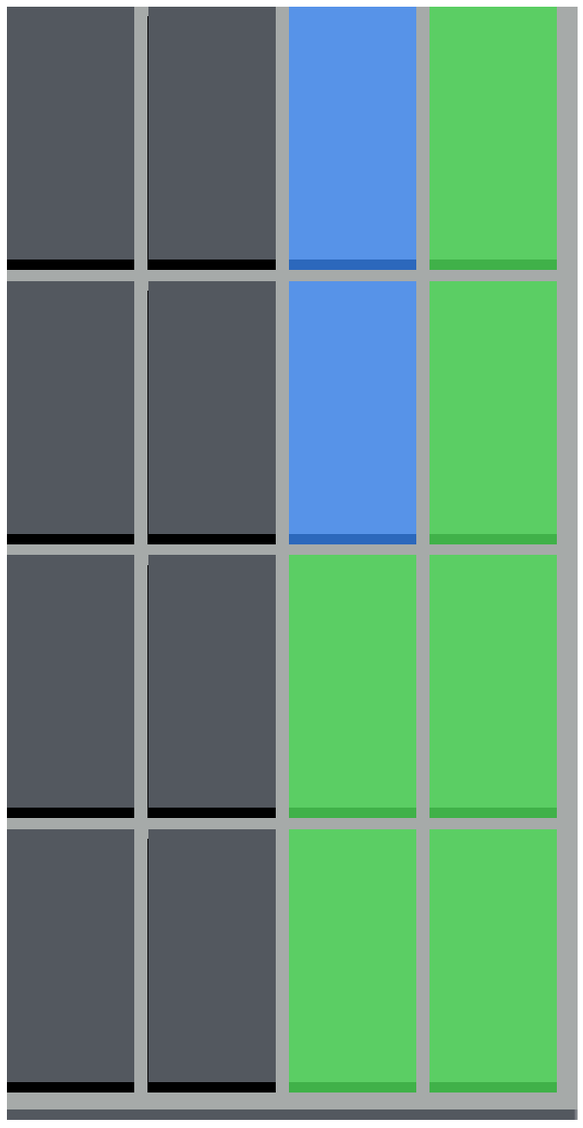


pganalyze

* except on Aurora

Postgres Shared Buffers

8kB Page



e.g. 128kB shared_buffers

Pinned Buffer
Actively being read/used by a backend

Dirty Buffer
Previously written to, not yet persisted to disk

Unpinned Buffer
Unused buffer, may contain cached data

pganalyze

Postgres Shared Buffers

What?

Reference count (pin count) is increased to avoid contents being replaced by others,
additional buffer locks may be taken (e.g. to replace content),
buffer is used (read and/or written to) and pin count decreased once work is done.

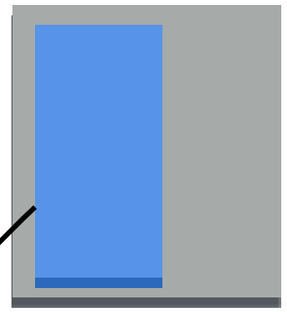
Who can do it?

1. Individual backends trying to read data / use cache data
2. Background processes (for similar reasons)

pganalyze

shared_buffers are not (just) a cache, they are essential for writes.

1. Remember changed page in shared_buffers



2. Remember changed rows (or full page) in wal_buffers

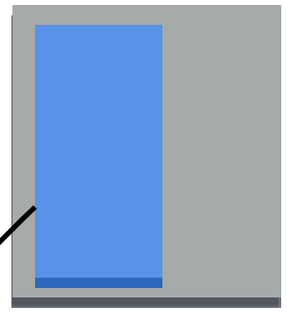
3. Persist to WAL

4. Persist to Data Directory

pganalyze

shared_buffers are not (just) a cache, they are essential for writes.

1. Remember changed page in shared_buffers



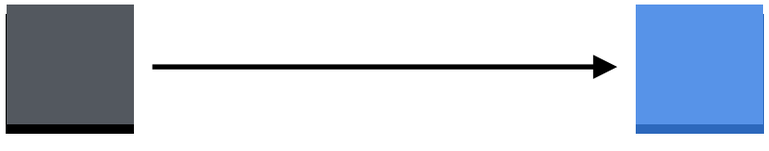
2. Remember changed rows (or full page) in wal_buffers

3. Persist to WAL

4. Persist to Data Directory

pganalyze

Postgres Shared Buffers



Pinned Buffer

Dirty Buffer

What?

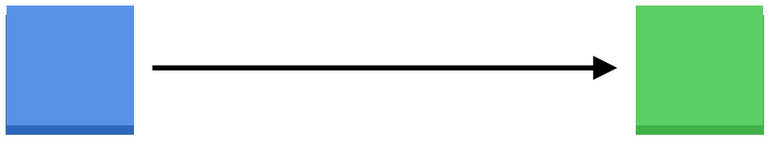
Backend (or other processes) changed the data in the buffer (= “dirtied” it) and that data now needs to be persisted to disk.

Who can do it?

1. Individual backends trying to write data
2. VACUUM cleaning up a table

pganalyze

Postgres Shared Buffers



Dirty Buffer Unpinned Buffer

What?

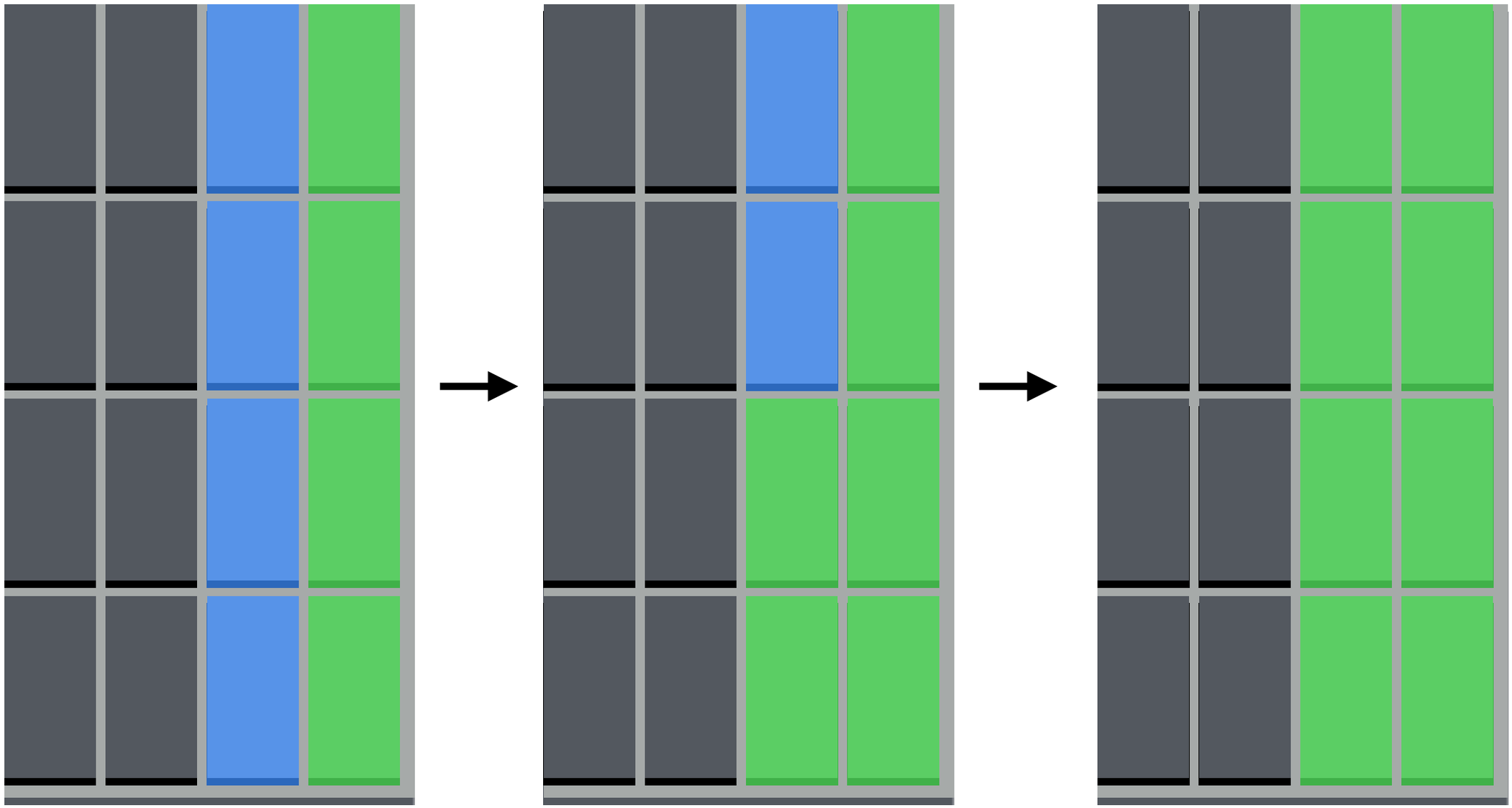
Write data from memory to OS page cache, and potentially flush OS page cache to disk.

Who can do it?

1. Background writer (bgwriter)
2. Checkpointer
3. Individual backend that needs an unpinned buffer 🚨

pganalyze

Background Writer



Background Writer

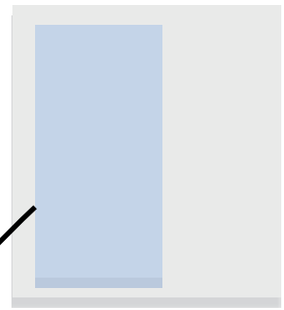
Tune to run more often for busy workloads
=> **reduce bgwriter_delay**

If background worker doesn't do its job in time,
Individual queries might write dirty buffers before they can read.
(pg_stat_statements.shared_blks_written)

But: If it runs too often, you'll create additional I/O.
(a dirty page can be "re-used" for a write within the same checkpoint)

Persisting Changes to the WAL

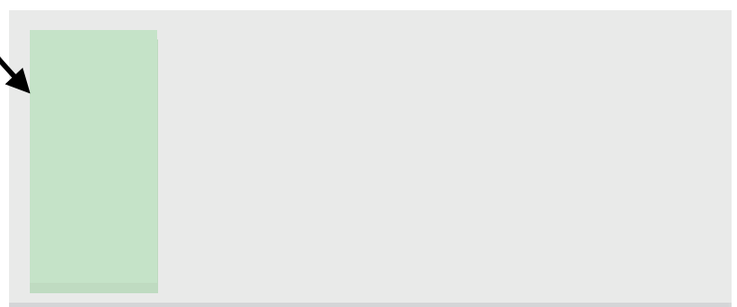
1. Remember changed page in shared_buffers



2. Remember changed rows (or full page) in wal_buffers

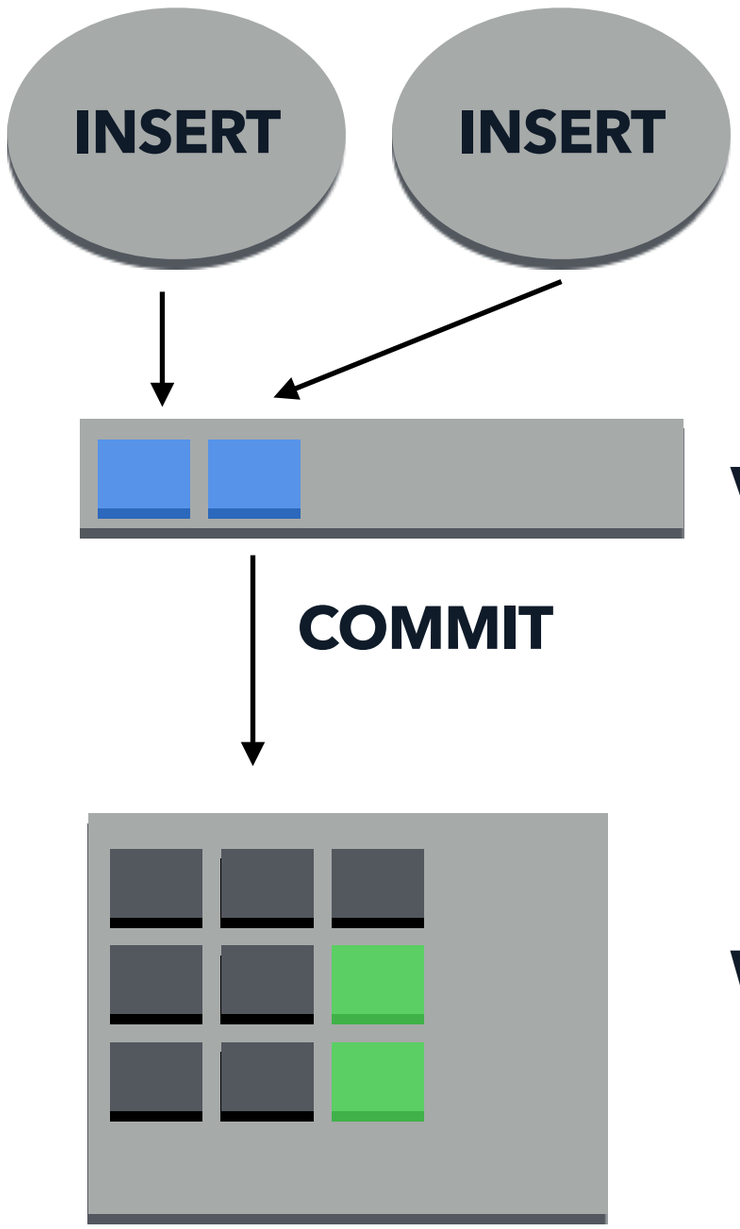
3. Persist to WAL

4. Persist to Data Directory



pganalyze

wal_buffers



wal_buffers (e.g. 16 MB)

WAL file (e.g. 64 MB)

pganalyze

wal_buffers = -1

- Default is automatically set to max(shared_buffers / 32, wal_segment_size)
- If you have very big transactions (or are using synchronous_commit = off), increasing wal_buffers to a multiple of wal_segment_size can be helpful

wal_buffers = -1

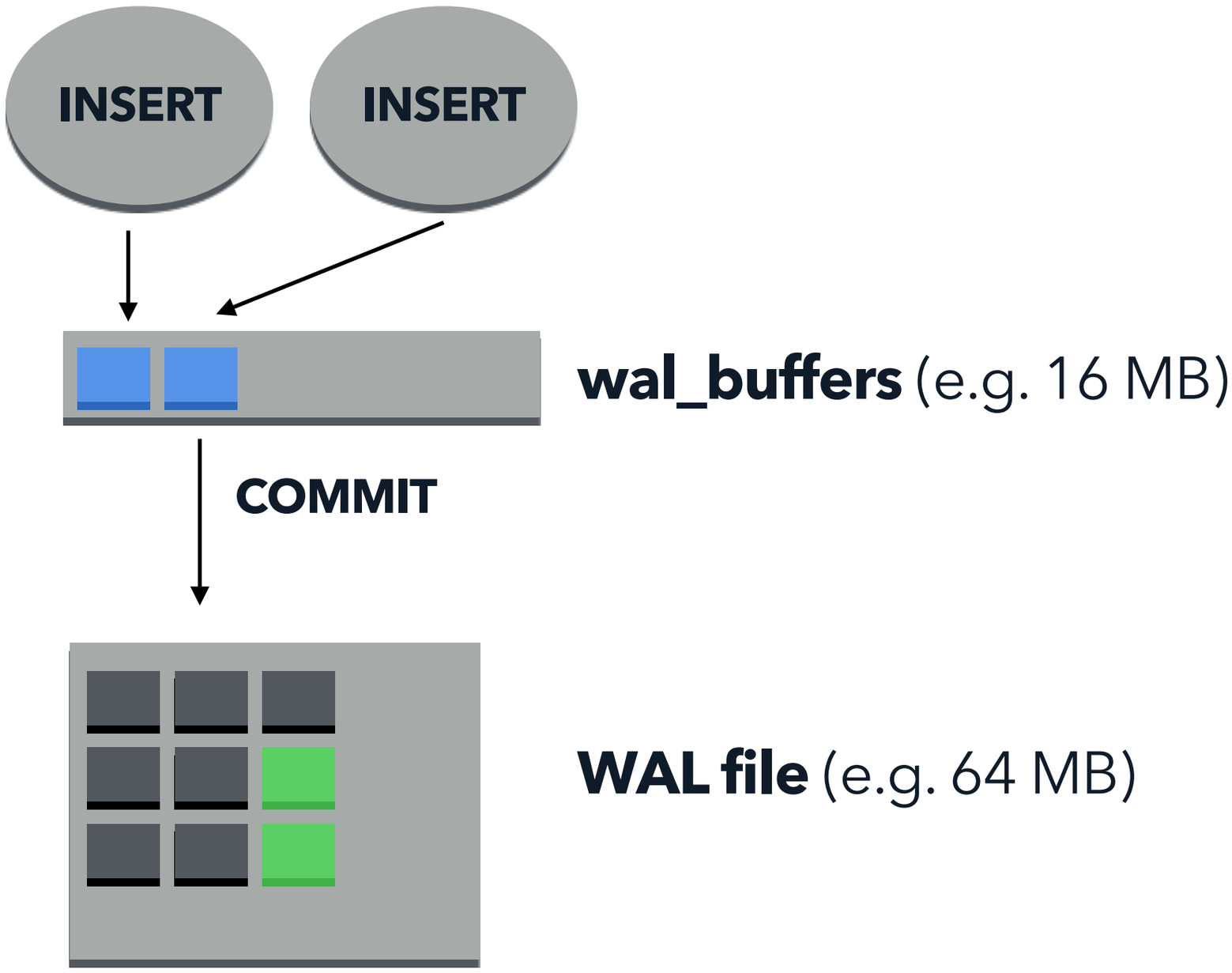
Write-Ahead Log / Settings

Setting	Current Value	Default Value	Source
commit_delay	0	0	default
commit_siblings	5	5	default
fsync	on	on	default
full_page_writes	on	on	default
synchronous_commit	on	on	configuration file
wal_buffers	16 MB	-1	override
wal_compression	off	off	default
wal_level	replica	replica	configuration file
wal_log_hints	on	off	configuration file
wal_sync_method	fdatasync	fdatasync	default
wal_writer_delay	200 ms	200 ms	default
wal_writer_flush_after	1 MB	1 MB	default

synchronous_commit = off

(sometimes)

synchronous_commit = on

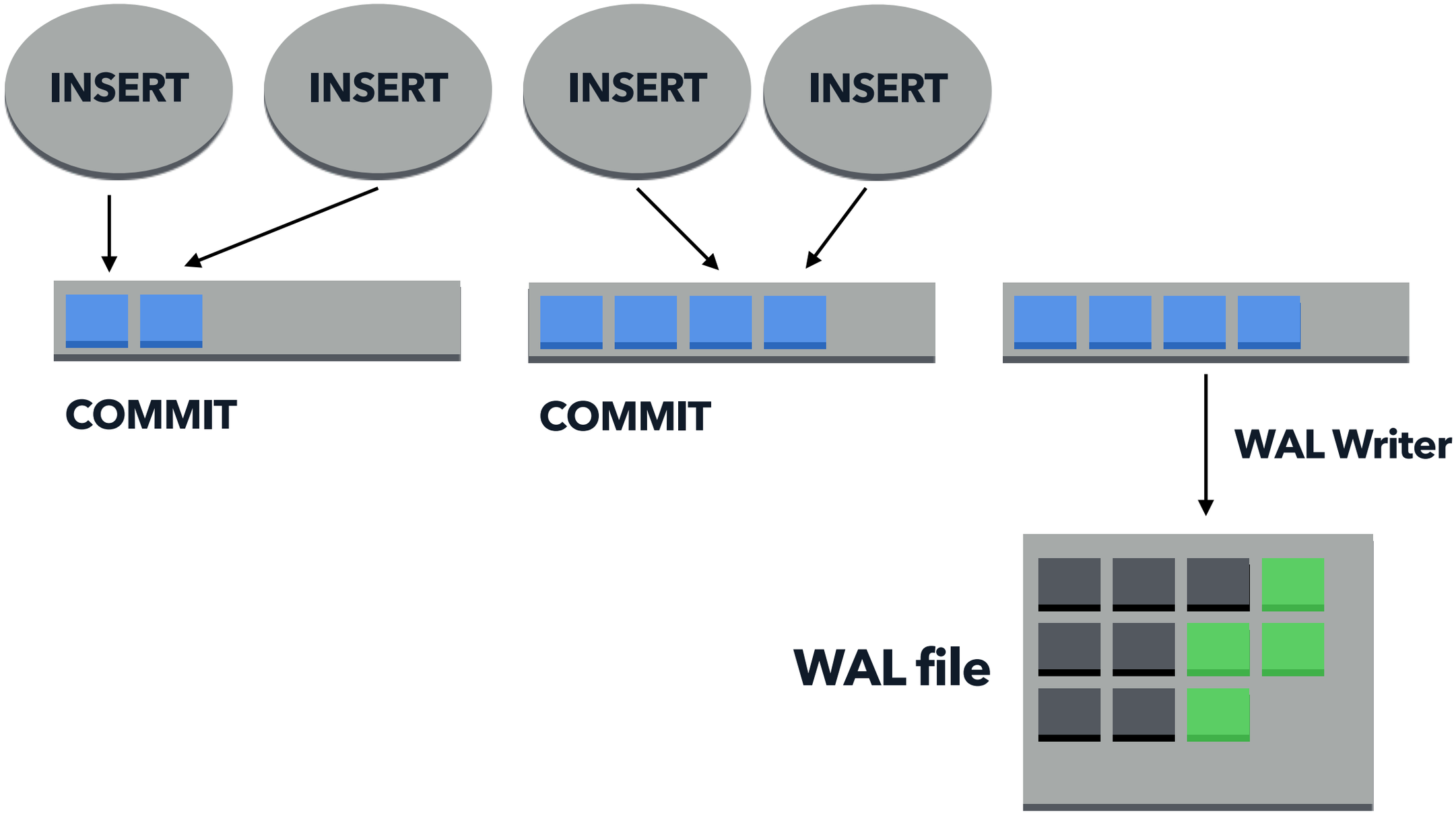


wal_buffers (e.g. 16 MB)

WAL file (e.g. 64 MB)

pganalyze

synchronous_commit = off



pganalyze

synchronous_commit = off

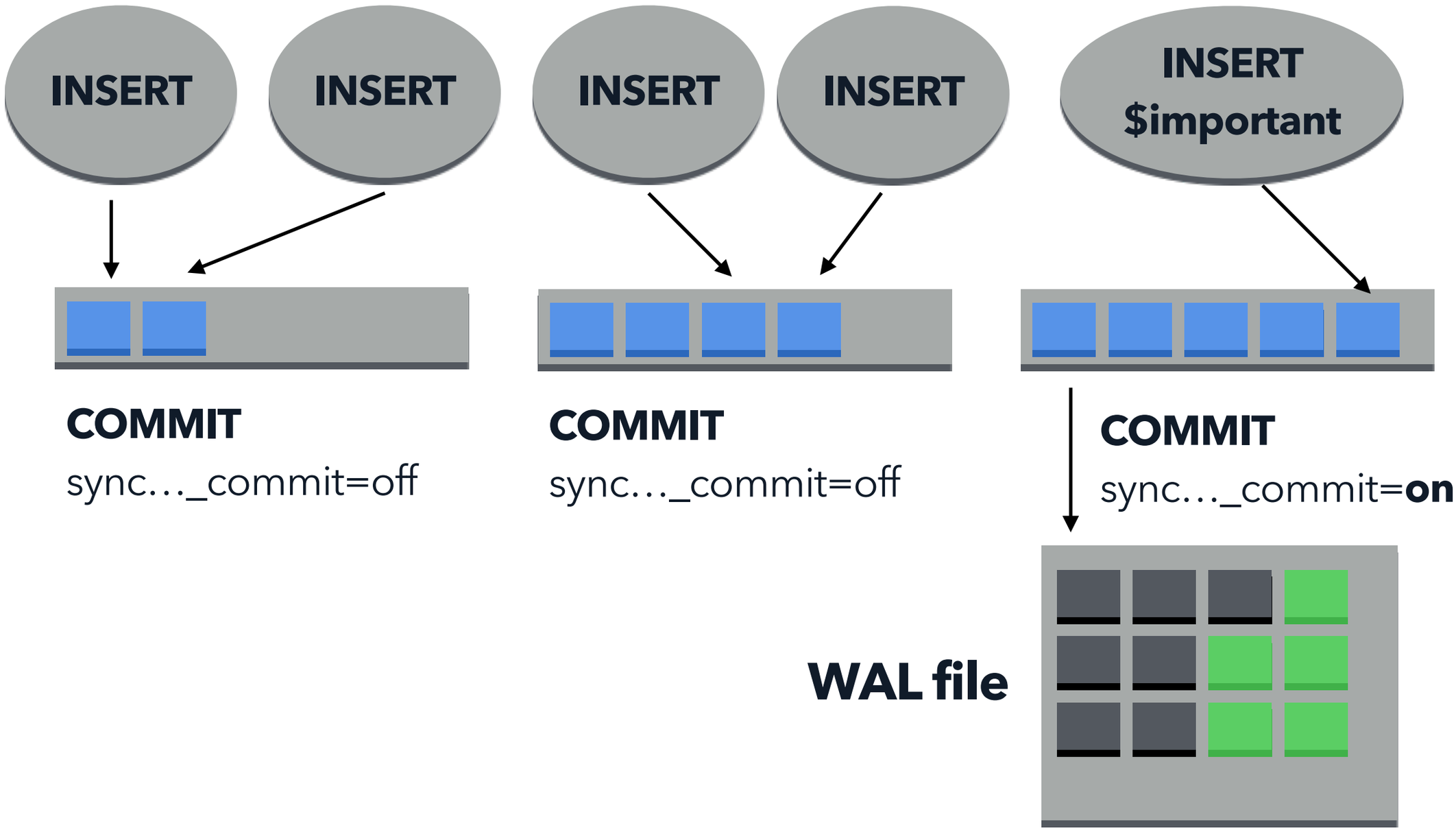
! You may loose data not yet persisted to WAL in a crash.

The database will be consistent
(just missing that most recent data),
no risk of corruption.

"This parameter **can be changed at any time**; the behavior for any one transaction is determined by the setting in effect when it commits.

It is therefore possible, and useful, to **have some transactions commit synchronously and others asynchronously**."

synchronous_commit = [SET per transaction]



pganalyze

If you make heavy use of
synchronous_commit = off ...

- Consider lowering **wal_writer_delay**
  (to write WAL more frequently, avoiding flushes during individual commits)
- Increase **wal_buffers** to a multiple of wal_segment_size

wal_segment_size & min_wal_size

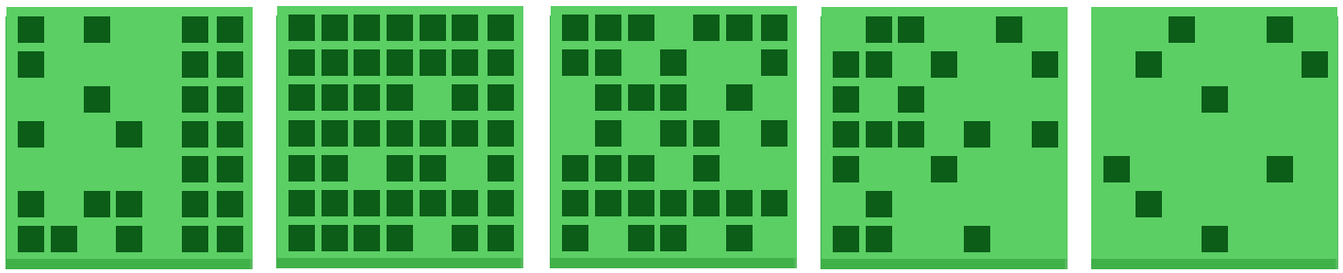
- Default of 16 MB is too small (too much churn with new files)
- Consider at least 64 MB
- If going for a much larger value, ensure min_wal_size is sufficient
  (to keep old files around, avoiding expensive initialization of new files)

pganalyze

A Tale of Checkpoints And Full Page Writes

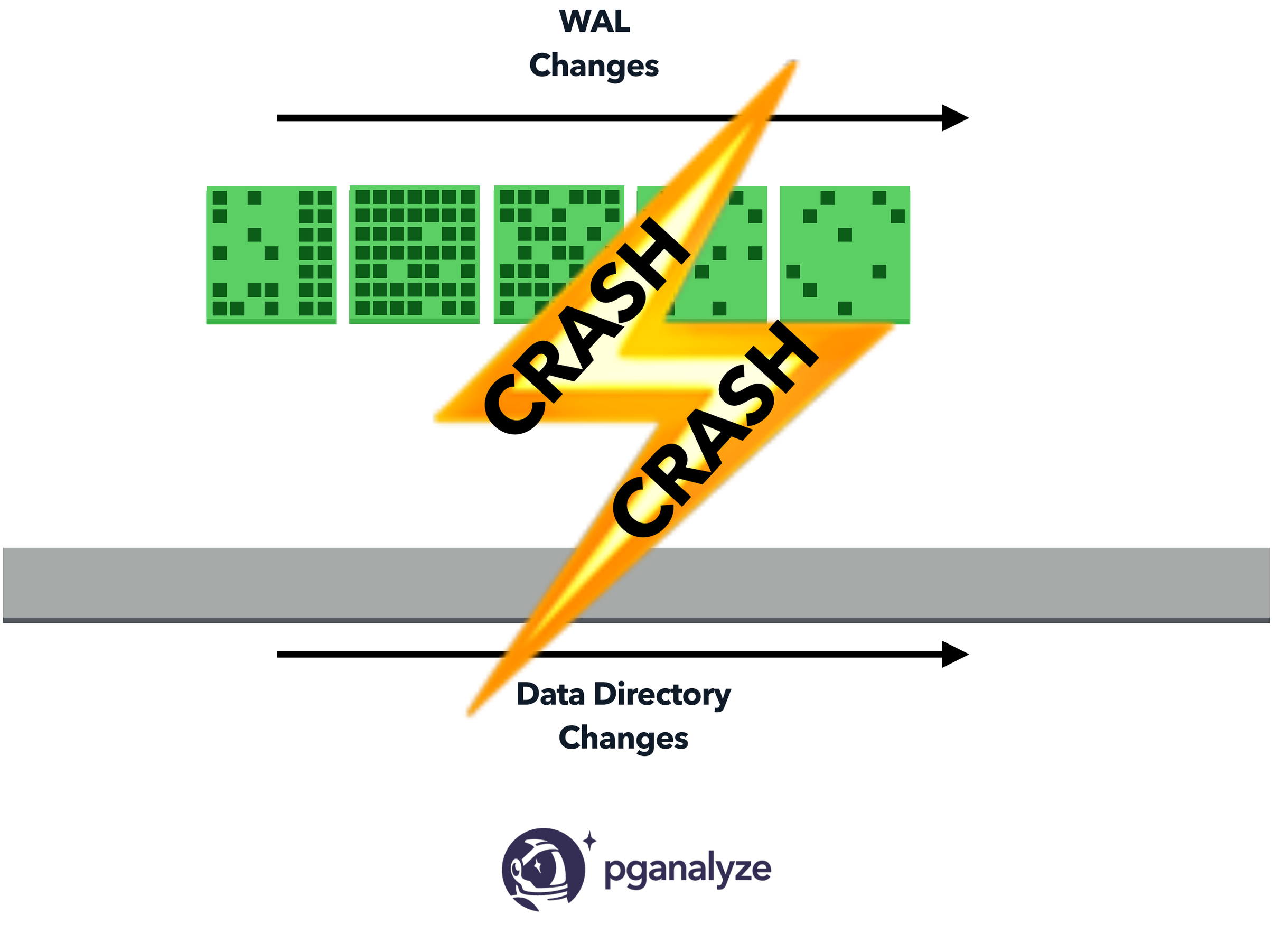
Why do we have checkpoints?

WAL
Changes



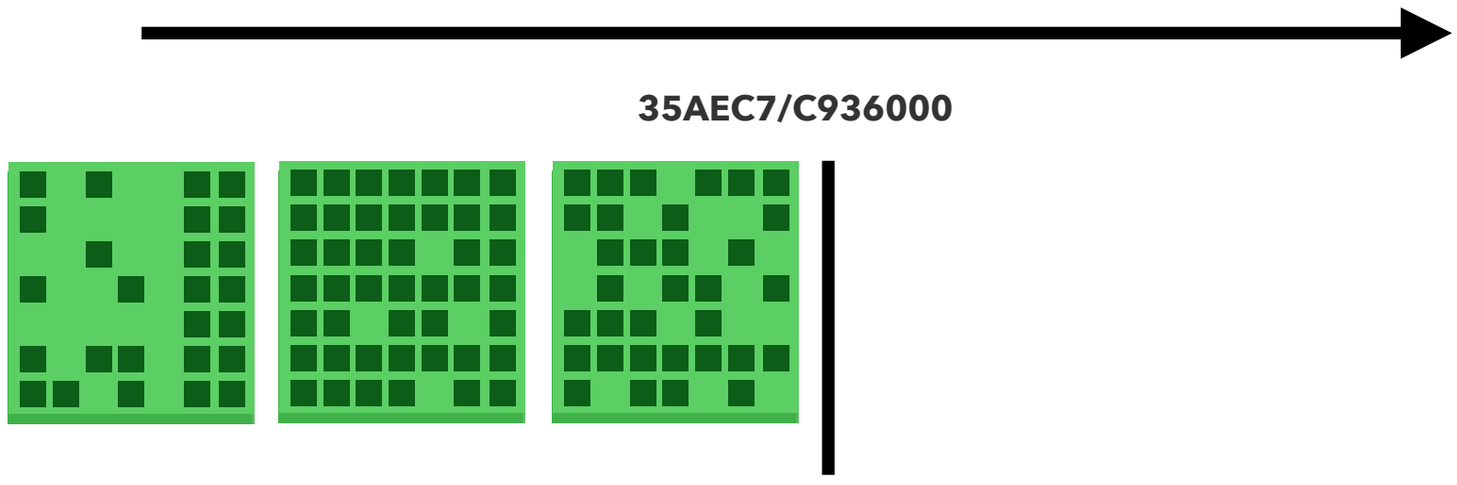
Data Directory
Changes

pganalyze



pganalyze

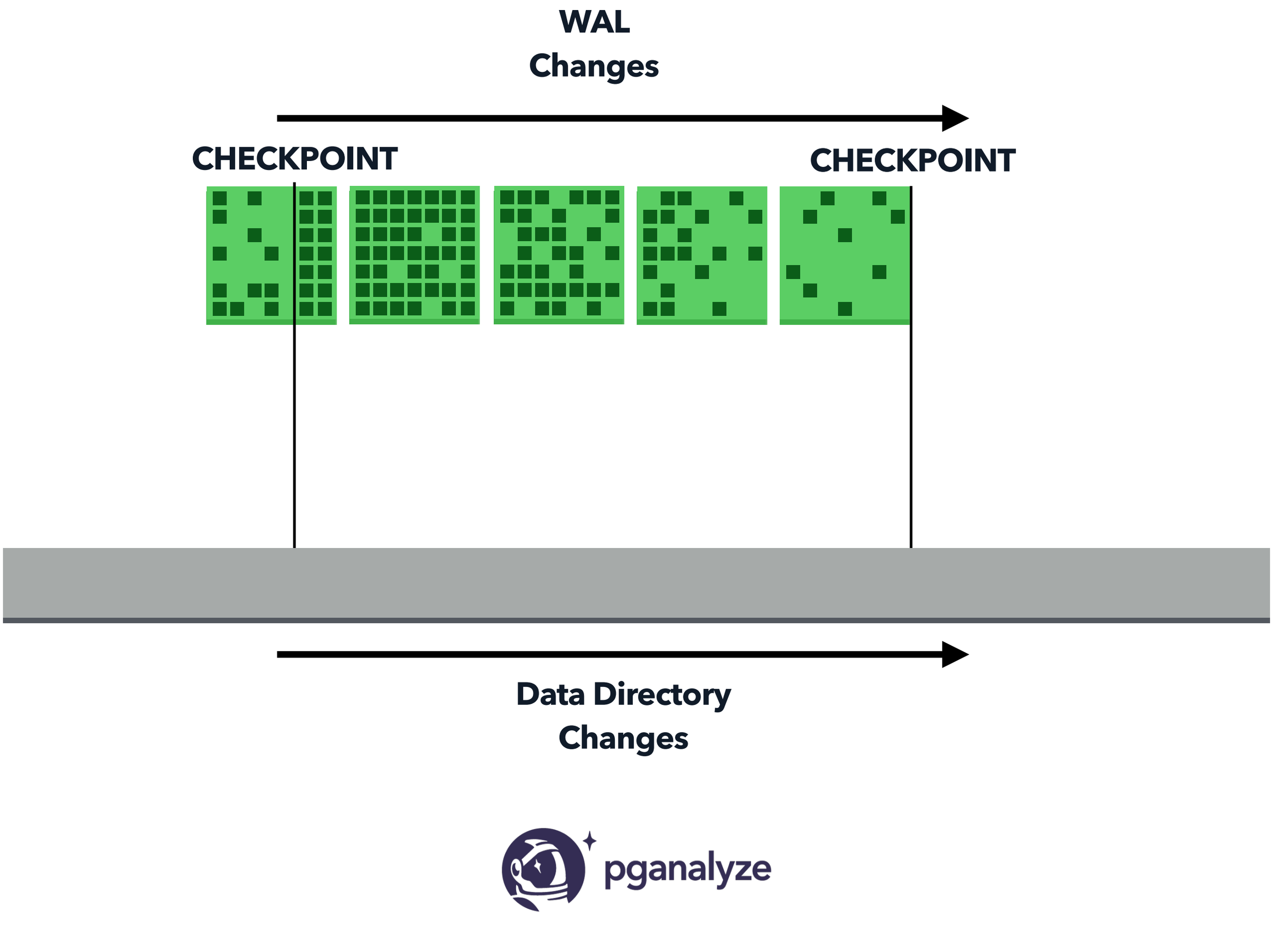
WAL
Changes



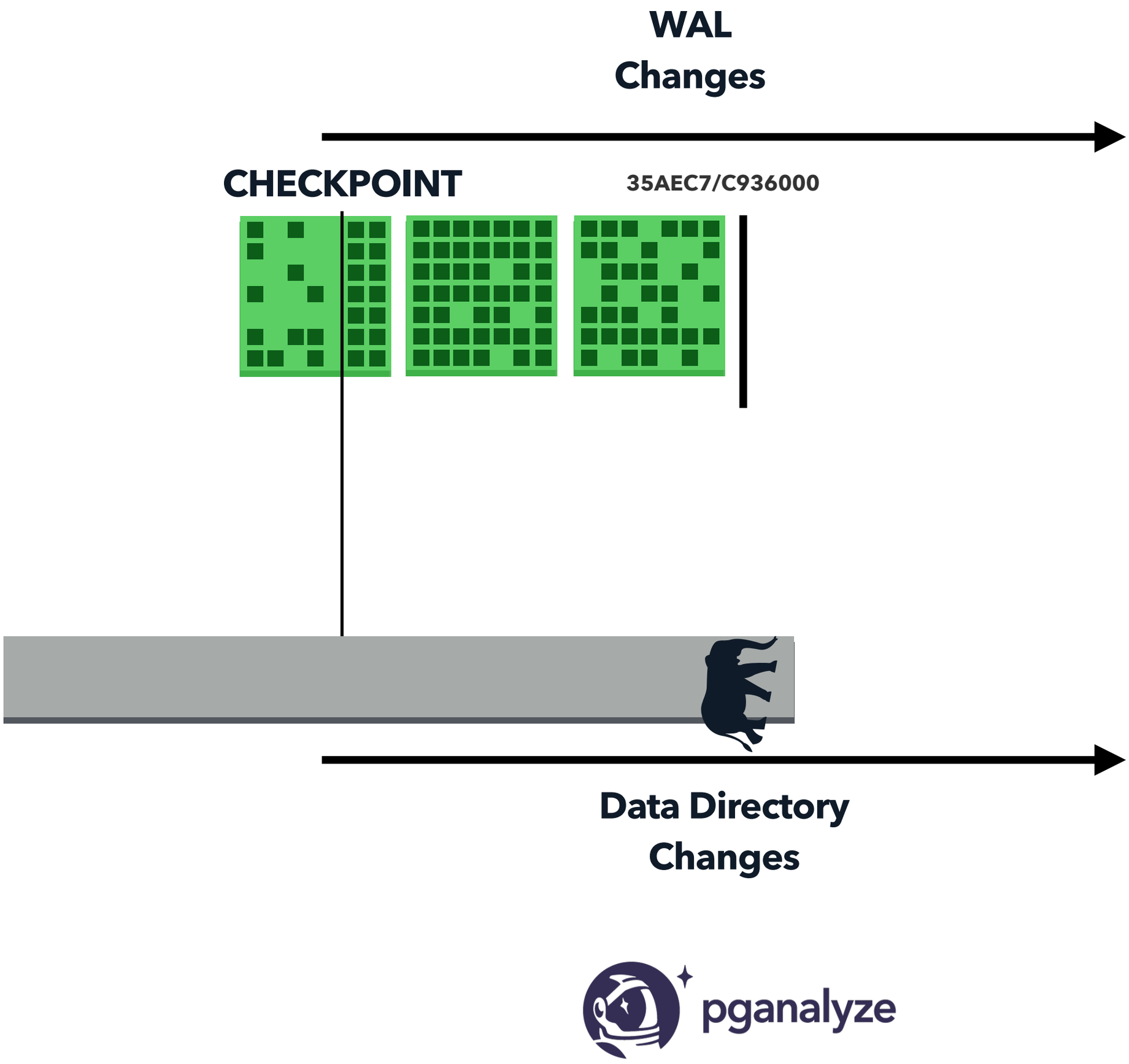
Data Directory
Changes

What is the state of
the data directory?!
(How far did it get synchronized?)

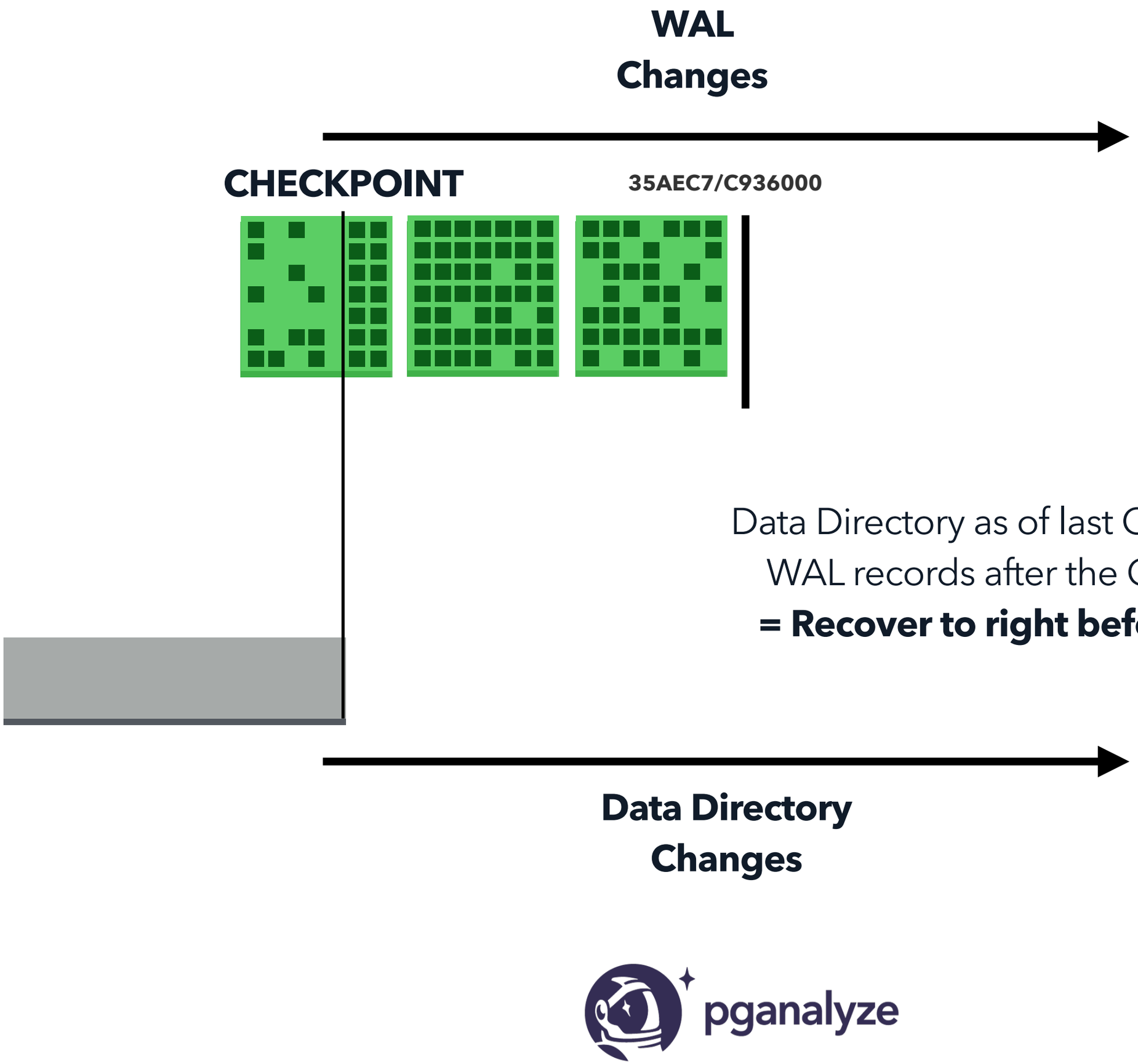
pganalyze


WAL
Changes
CHECKPOINT
CHECKPOINT
Data Directory
Changes

pganalyze

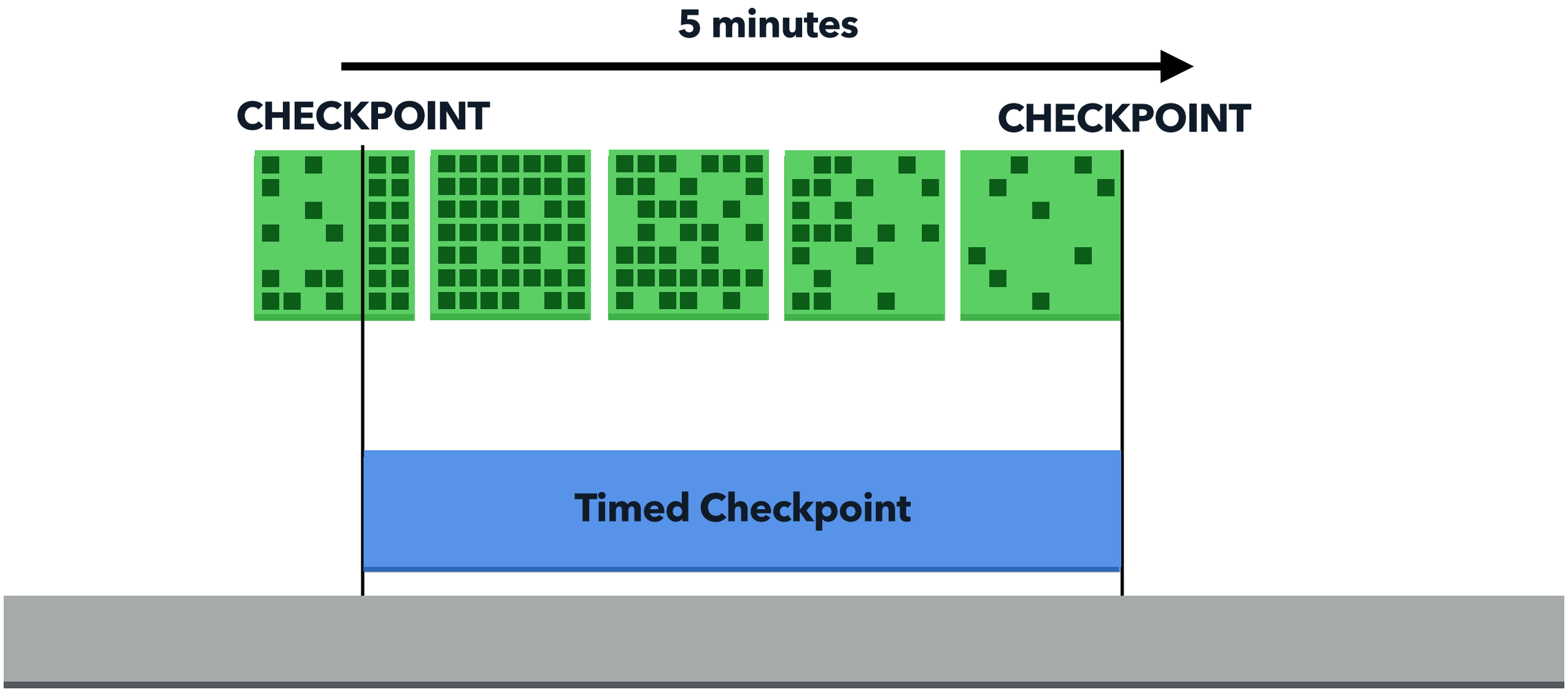

WAL
Changes
CHECKPOINT
35AEC7/C936000
Data Directory
Changes

pganalyze


WAL
Changes
CHECKPOINT
35AEC7/C936000
Data Directory as of last CHECKPOINT +
WAL records after the CHECKPOINT
= Recover to right before the crash
Data Directory
Changes

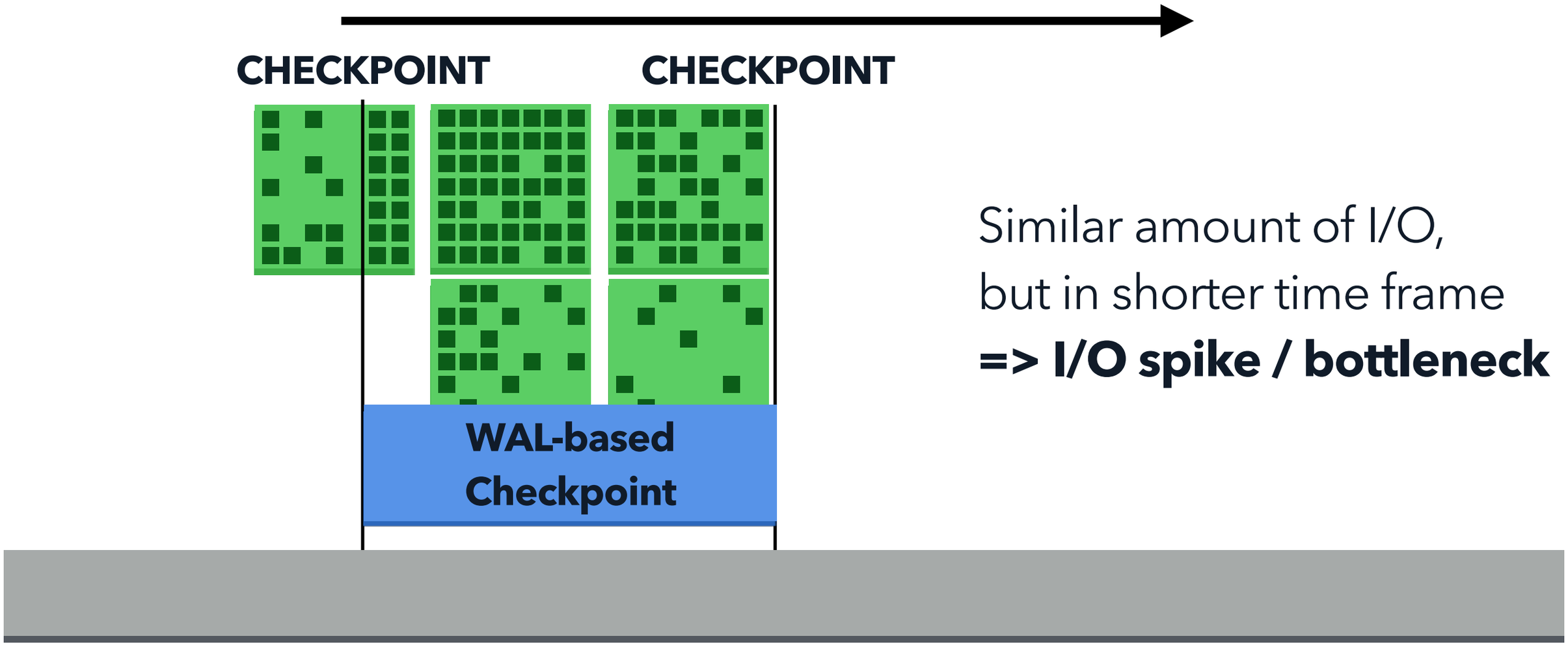
pganalyze

Timed vs WAL-based Checkpoints



checkpoint_timeout = 300s

pganalyze



Similar amount of I/O,
but in shorter time frame
=> I/O spike / bottleneck

max_wal_size = 1GB

pganalyze

Write-Ahead Log / Checkpoints

Setting	Current Value	Default Value	Source
checkpoint_completion_target	0.9	0.9	configuration file
checkpoint_flush_after	256 KB	256 KB	default
checkpoint_timeout	300 s	300 s	default
checkpoint_warning	30 s	30 s	default
max_wal_size	50 GB	1 GB	configuration file
min_wal_size	192 MB	80 MB	configuration file

Log Insights - W40: Checkpoint starting

Checkpoint to synchronize the data directory has started

Learn more

Postgres Server Log

Show all log events

Reason for Checkpoint Starting:

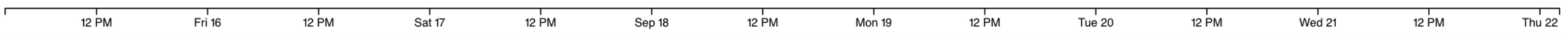
1826 × time

1 × wal

Related Config Parameters:

```
checkpoint_completion_target = 0.9
checkpoint_timeout = 300 s
max_wal_size = 50 GB
```

Jump to time:



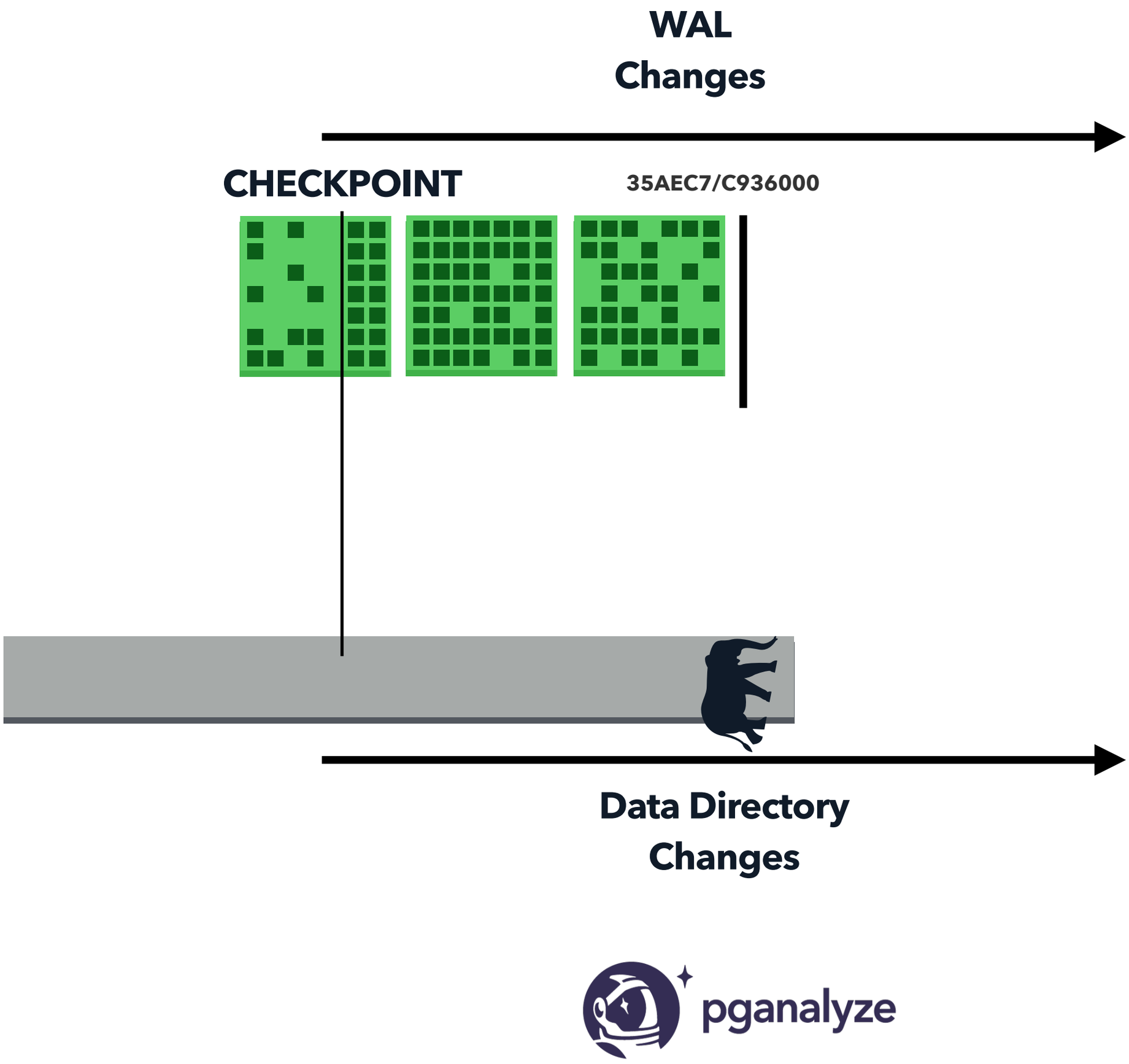

```
Sep 22 02:04:27am PDT W40 4498 LOG: checkpoint starting: time
Sep 22 01:59:27am PDT W40 4498 LOG: checkpoint starting: time
Sep 22 01:54:27am PDT W40 4498 LOG: checkpoint starting: time
Sep 22 01:49:27am PDT W40 4498 LOG: checkpoint starting: time
Sep 22 01:44:27am PDT W40 4498 LOG: checkpoint starting: time
Sep 22 01:39:27am PDT W40 4498 LOG: checkpoint starting: time
Sep 22 01:34:27am PDT W40 4498 LOG: checkpoint starting: time
Sep 22 01:29:27am PDT W40 4498 LOG: checkpoint starting: time
Sep 22 01:24:27am PDT W40 4498 LOG: checkpoint starting: time
Sep 22 01:19:27am PDT W40 4498 LOG: checkpoint starting: time
Sep 22 01:14:27am PDT W40 4498 LOG: checkpoint starting: time
Sep 22 01:09:27am PDT W40 4498 LOG: checkpoint starting: time
Sep 22 01:04:27am PDT W40 4498 LOG: checkpoint starting: time
```

pganalyze

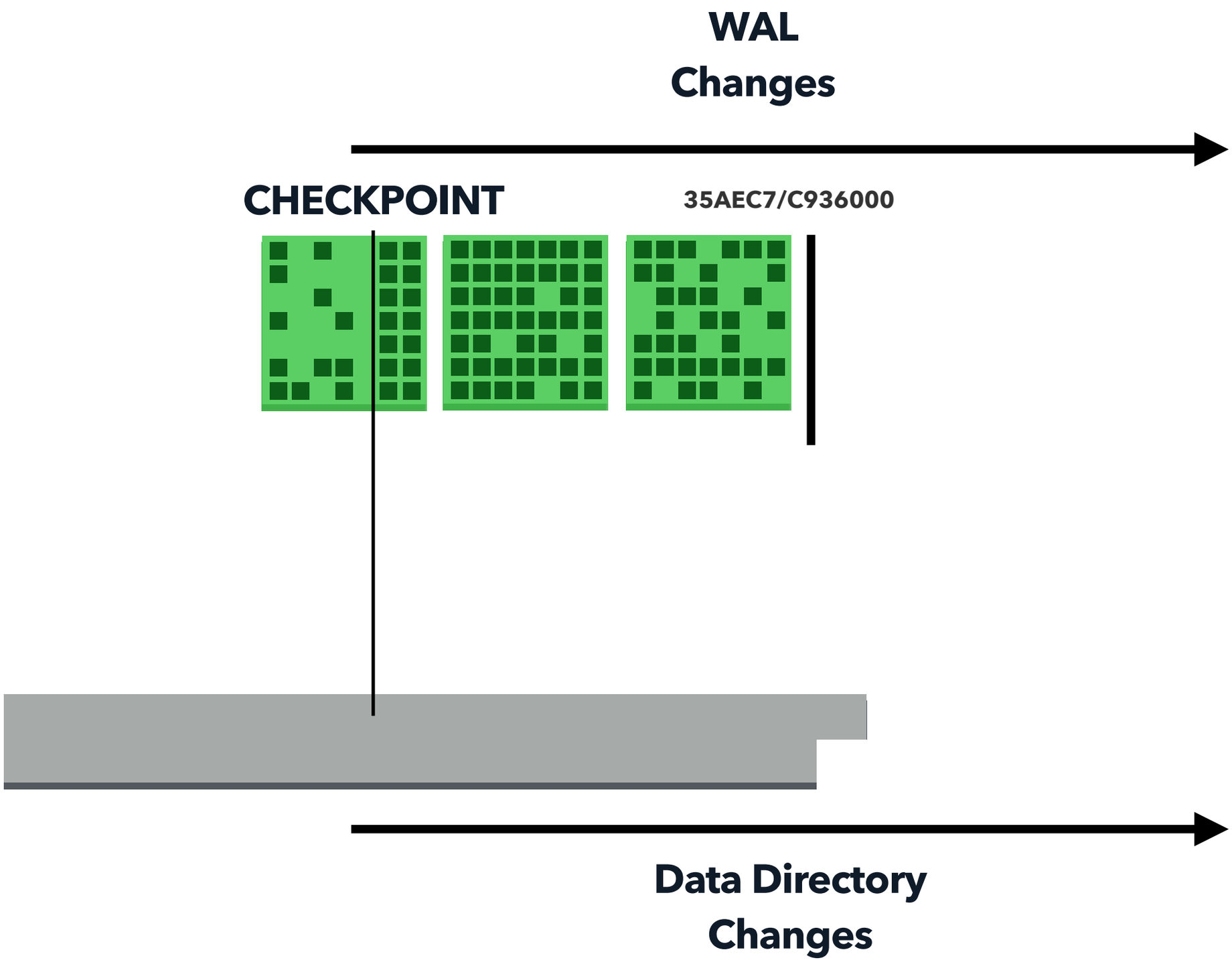
Full Page Writes

Full Page Images (FPIs) are big

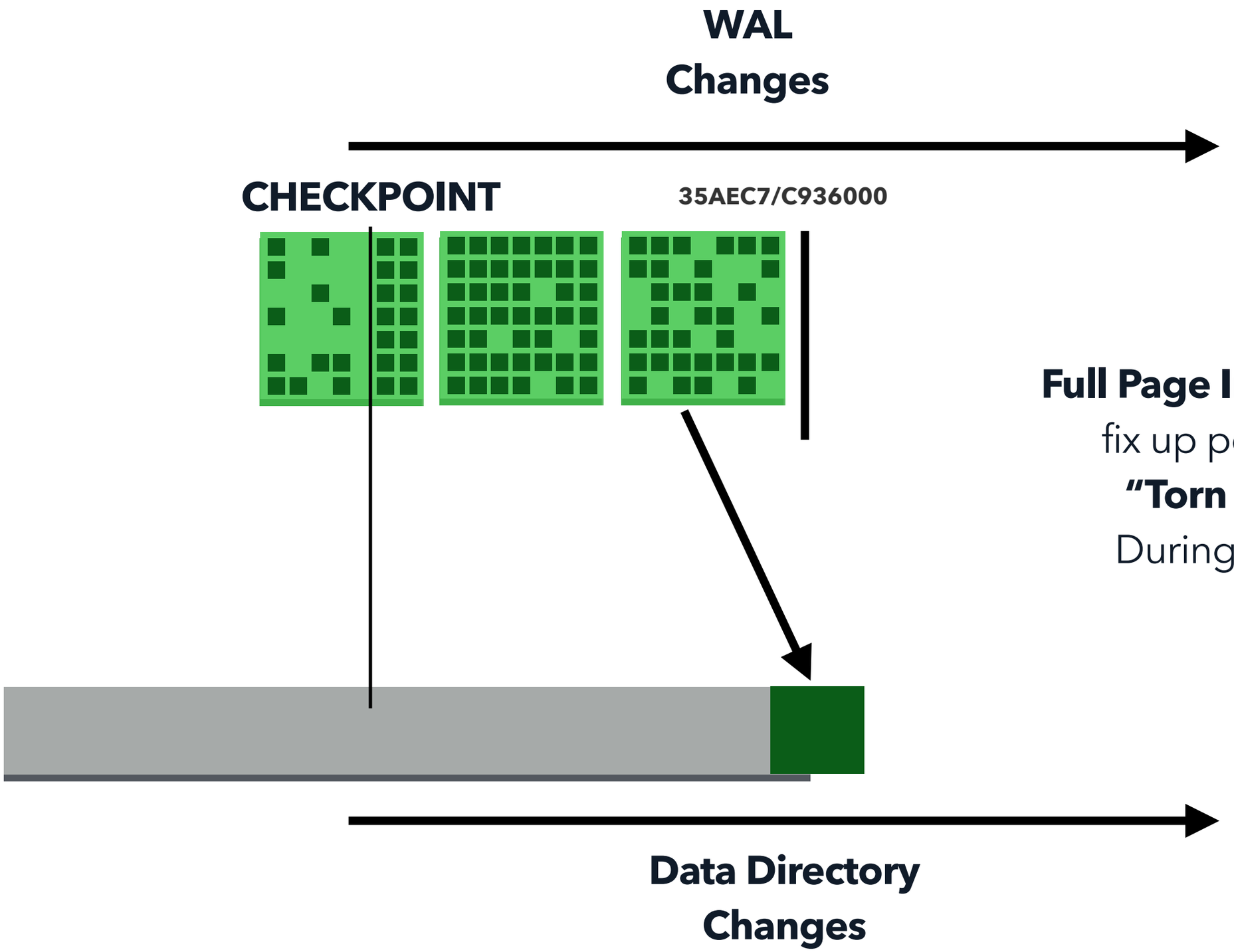
(a full 8kb page copied into the WAL)


WAL
Changes
CHECKPOINT
35AEC7/C936000
Data Directory
Changes

pganalyze

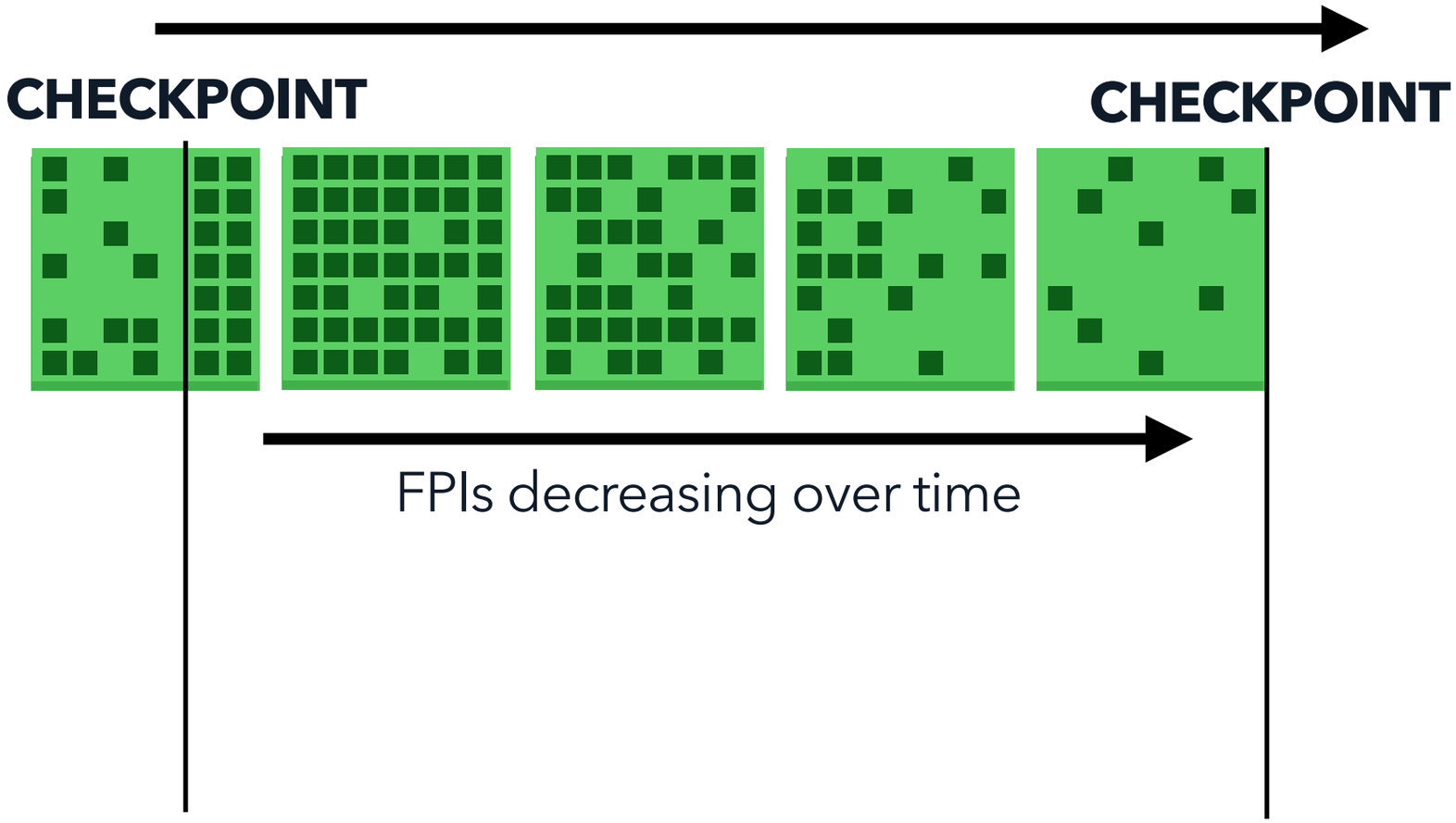

WAL
Changes
CHECKPOINT
35AEC7/C936000
Data Directory
Changes

pganalyze

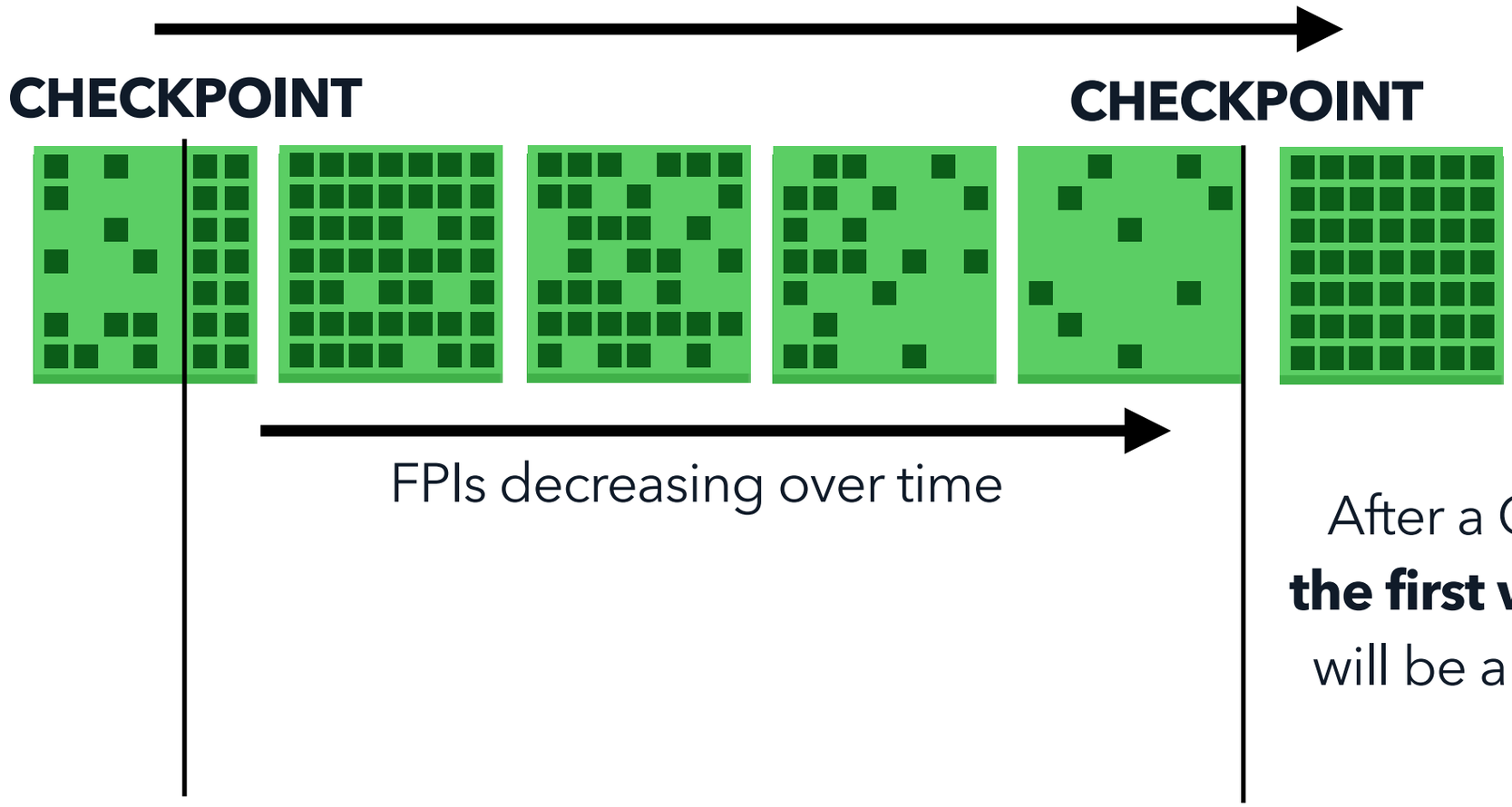


Full Page Images (FPI)
fix up potentially
“Torn Pages”
During a Crash

pganalyze



pganalyze



After a CHECKPOINT,
the first write to a page
will be a full page write

pganalyze

wal_compression = lz4 (with PG15+)

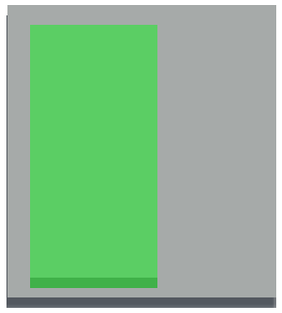
- Compresses FPIs in WAL with lz4 compression
- Reduces impact of having lots of full page writes after a checkpoint
- Before PG15 only pglz format was supported (high CPU overhead)

pganalyze

Amazon Aurora Is Different (But Not Always Better)

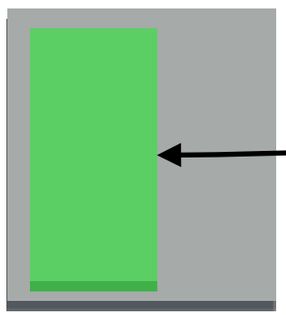
Amazon Aurora Is Different

1. Remember changed page in shared_buffers



2. Write out changed rows to WAL

3. Refresh Replicas
(their shared_buffers)



Aurora Log Write

pganalyze

Amazon Aurora Is Different

No Full Page Writes

"No Checkpoints"

(Heavily Modified Checkpointer

+ Background Writer)

Aurora Is Not Always Better

Both Read and Write IOPS are charged extra

Read I/Os are always charged per 8kB disk page

Read I/Os will be slower (sometimes)

Write I/Os are always charged per 4kB log record

Write I/O with synchronous_commit=on will be slower

"An **unoptimized SQL query** can incur higher I/Os as compared to an optimized query, because it needs to scan a lot of pages to get to the final query result.

Typically, **this is the most common cause for higher Aurora I/Os.**"

Amazon Aurora I/O Cost Optimization Methodology

pganalyze

Breaking Down Your Query Workload Into I/Os

Query Performance · pganalyze

staging.pganalyze.com/databases/23/queries

pganalyze

Server: prod-db-main (Amazon RDS)

Database: pgaweb

Last 24 hours

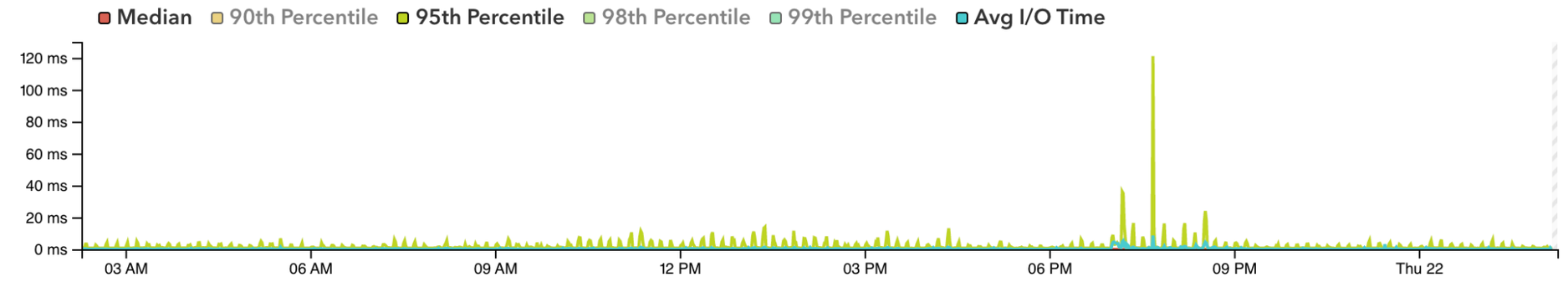
ORGANIZATION
pganalyze

- Dashboard
- Query Performance
- Index Advisor
- EXPLAIN Plans
- Schema Statistics
- Log Insights
- Connections
- VACUUM Activity
- Config Tuning
- System
- Alerts & Check-Up
- Settings

Lukas Fittl

Query Performance

Database Queries



SELECT | INSERT, UPDATE, DELETE | DDL & other | Compare to 7 days ago | Search...

QUERY	ROLE	AVG TIME (MS)	CALLS / MIN	% OF ALL I/O	% OF ALL RUNTIME
WITH total_times AS (...), table_queries AS (...)…	pgaweb_workers	127.10ms	183.89	15.35%	8.53%
WITH slow_queries AS (SELECT qs.database_id, qs.…	pgaweb_workers	3,510.92ms	4.55	9.91%	5.84%
INSERT INTO snapshots (...) SELECT ... RETURNING …	pgaweb_workers	0.83ms	16728.83	8.90%	5.06%
UPDATE queries q SET ... = ... FROM query_fingerp…	pgaweb_workers	5.76ms	2410.76	8.31%	5.07%
SELECT ... FROM query_stats_35d qs LEFT JOIN quer…	pgaweb_workers	4.46ms	2414.66	4.65%	3.93%
SELECT schema_indices.id, schema_indices.table_id…	pgaweb_workers	11.82ms	457.62	4.20%	1.97%
SELECT count(*) FROM queries WHERE queries.databa…	pgaweb_workers	1,795.24ms	2.75	4.13%	1.80%
COPY public.query_stats_35d_20220124(database_id,…	pgaweb_workers	14.51ms	2411.07	4.08%	12.78%
SELECT max(snapshots.received_at) FROM snapshots …	pgaweb_workers	290,554.33ms	0.03	3.78%	
INSERT INTO schema_table_scan_methods (...) VALUE…	pgaweb_workers	0.56ms	13584.80	3.47%	
COPY public.schema_index_stats_35d_20220124(schem…	pgaweb_workers	35.71ms	448.26	2.98%	5.85%

Get Help

pganalyze

Query #43839732 · pganalyze

staging.pganalyze.com/databases/23/queries/43839732

pganalyze

Server: prod-db-main (Amazon RDS)

Database: pgaweb

Last 24 hours

ORGANIZATION
pganalyze

- Dashboard
- Query Performance
- Index Advisor
- EXPLAIN Plans
- Schema Statistics
- Log Insights
- Connections
- VACUUM Activity
- Config Tuning
- System
- Alerts & Check-Up
- Settings

Lukas Fittl

```
WITH slow_queries AS ( SELECT qs.database_id, qs.fingerprint, qs....
```

Query #43839732 · Fingerprint: 7ed633e5f0d850da · Role: pgaweb_workers

Avg Time: 3,510.92ms

Calls Per Minute: 4.55 / min

Compare to 7 days ago

Overview | NEW Index Advisor 2 | Query Samples 5+ | EXPLAIN Plans 5+ | Query Tags 5+ | Log Entries 100+

Check-Up

3 new issues

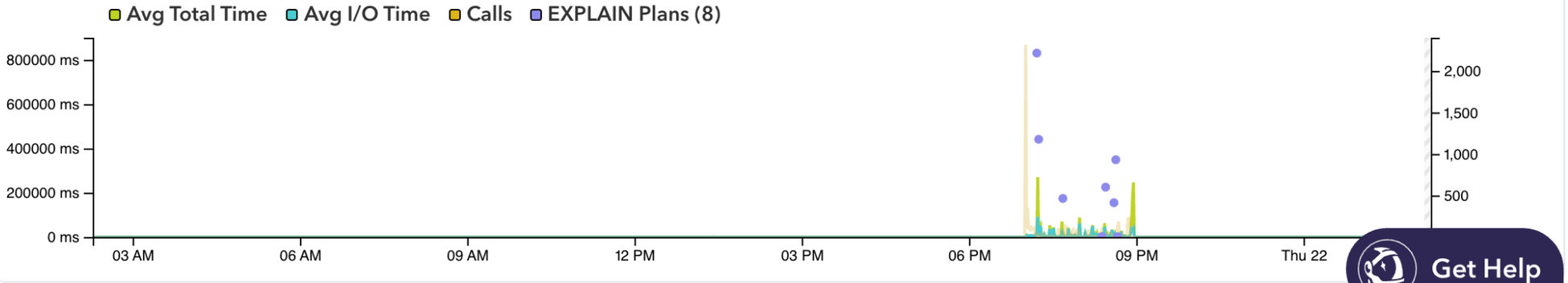
- Info Missing index on **public.query_stats_35d btree (collected_at, database_id)**
- Info Query **#43839732** takes 4472 ms on average (481614 ms max, 214 MB read from disk per call, 3848 calls in last 24h)
- Info Missing index on **public.query_samples_7d btree (postgres_role_id, database_id, query_fingerprint)**

SQL Statement

```
WITH slow_queries AS (
  SELECT qs.database_id, qs.fingerprint, qs.postgres_role_id, SUM(qs.total_time) / SUM(qs.calls) AS avg_time, SUM(qs.shar...
```

Show full query text

Avg Time & Calls



Get Help

Cache Hit Ratio

pganalyze

Query #43839732 · pganalyze

staging.pganalyze.com/databases/23/queries/43839732

pganalyze

Server: prod-db-main (Amazon RDS)

Database: pgaweb

Last 24 hours

ORGANIZATION
pganalyze

- Dashboard
- Query Performance
- Index Advisor
- EXPLAIN Plans
- Schema Statistics
- Log Insights
- Connections
- VACUUM Activity
- Config Tuning
- System
- Alerts & Check-Up
- Settings

Lukas Fittl

```
WITH slow_queries AS ( SELECT qs.database_id, qs.fingerprint, qs....
```

Query #43839732 · Fingerprint: 7ed633e5f0d850da · Role: pgaweb_workers

Avg Time: 3,510.92ms

Calls Per Minute: 4.55 / min

Compare to 7 days ago

Overview | NEW Index Advisor 2 | Query Samples 5+ | EXPLAIN Plans 5+ | Query Tags 5+ | Log Entries 100+

Check-Up

3 new issues

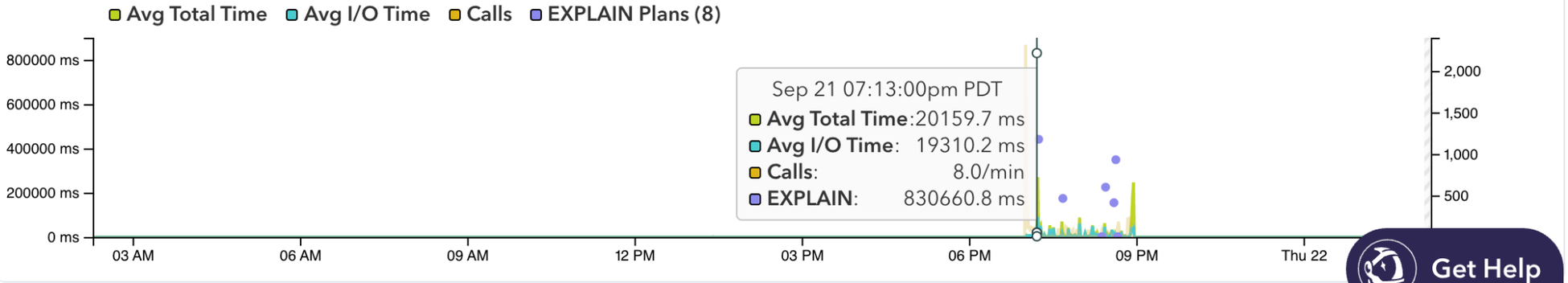
- Info Missing index on **public.query_stats_35d btree (collected_at, database_id)**
- Info Query **#43839732** takes 4472 ms on average (481614 ms max, 214 MB read from disk per call, 3848 calls in last 24h)
- Info Missing index on **public.query_samples_7d btree (postgres_role_id, database_id, query_fingerprint)**

SQL Statement

```
WITH slow_queries AS (
  SELECT qs.database_id, qs.fingerprint, qs.postgres_role_id, SUM(qs.total_time) / SUM(qs.calls) AS avg_time, SUM(qs.shar...
```

Show full query text

Avg Time & Calls



Get Help

Cache Hit Ratio

pganalyze

Query #43839732 · pganalyze

staging.pganalyze.com/databases/23/queries/43839732/explains/5f5jjymudbf6jck6fuvdchn4qy

pganalyze

ORGANIZATION
pganalyze

- Dashboard
- Query Performance
- Index Advisor
- EXPLAIN Plans
- Schema Statistics
- Log Insights
- Connections
- VACUUM Activity
- Config Tuning
- System
- Alerts & Check-Up
- Settings

Lukas Fittl

Server: prod-db-main (Amazon RDS)

Database: pgaweb

Last 24 hours

```
WITH slow_queries AS ( SELECT qs.database_id, qs.fingerprint, qs....
```

Query #43839732 · Fingerprint: 7ed633e5f0d850da · Role: pgaweb_workers

Avg Time: 3,510.92ms

Calls Per Minute: 4.55 / min

Compare to 7 days ago

Overview | NEW Index Advisor 2 | Query Samples 5+ | EXPLAIN Plans 5+ | Query Tags 5+ | Log Entries 100+

Check-Up

3 new issues

- Info Missing index on **public.query_stats_35d btree (collected_at, database_id)**
- Info Query **#43839732** takes 4472 ms on average (481614 ms max, 214 MB read from disk per call, 3848 calls in last 24h)
- Info Missing index on **public.query_samples_7d btree (postgres_role_id, database_id, query_fingerprint)**

EXPLAIN Plan

2022-09-21 07:13:54pm PDT · Fingerprint: a38dae4

830,660.75ms	168,610.33ms	18.6 GB	21,330,255
Runtime	I/O Read Time	Read From Disk	Total Est. Cost

✔ Analyze ✔ Verbose ✔ Costs ✔ Buffers ✖ Timing ✖ Summary

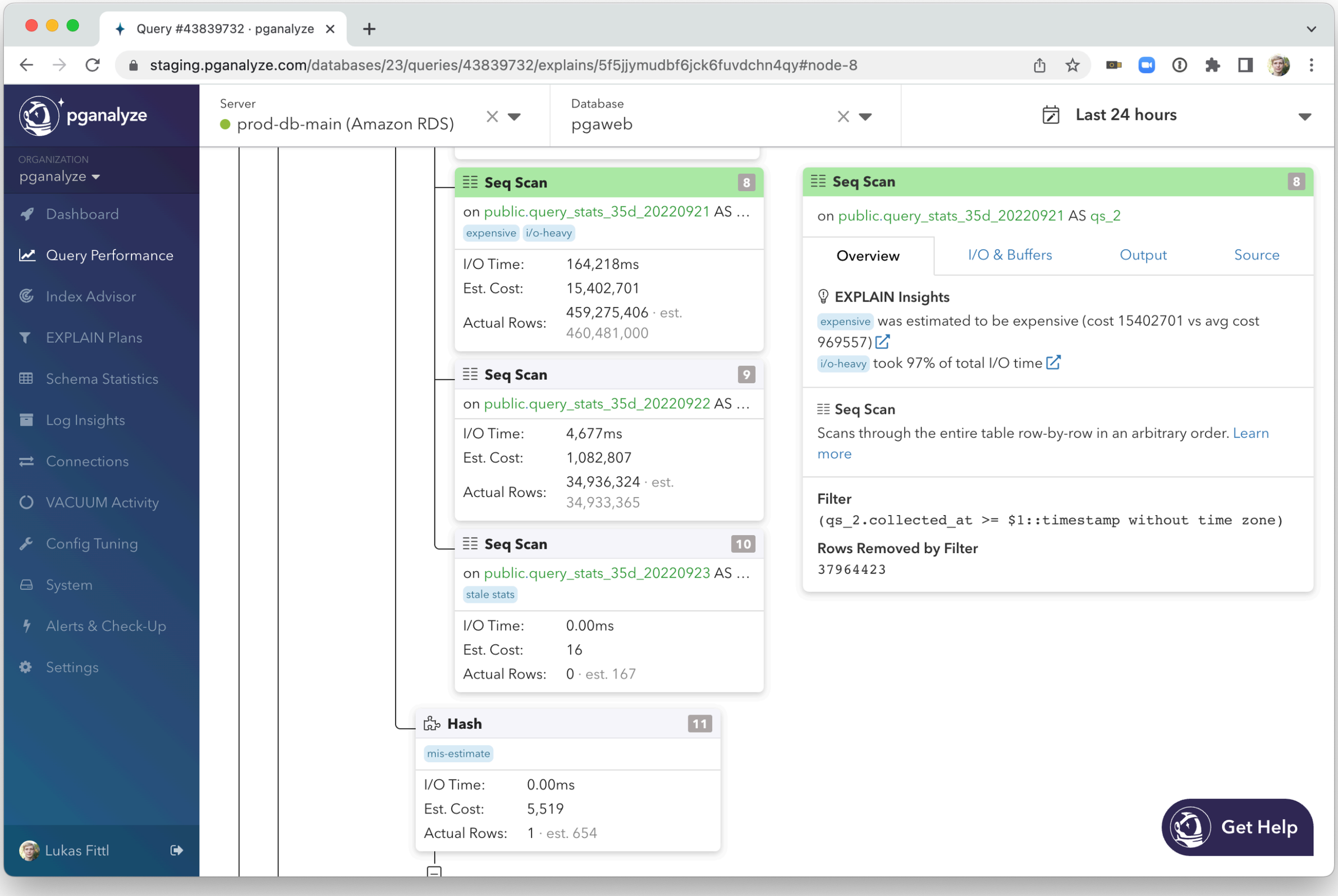
Compare Query Plans | View EXPLAIN Source

EXPLAIN Insights

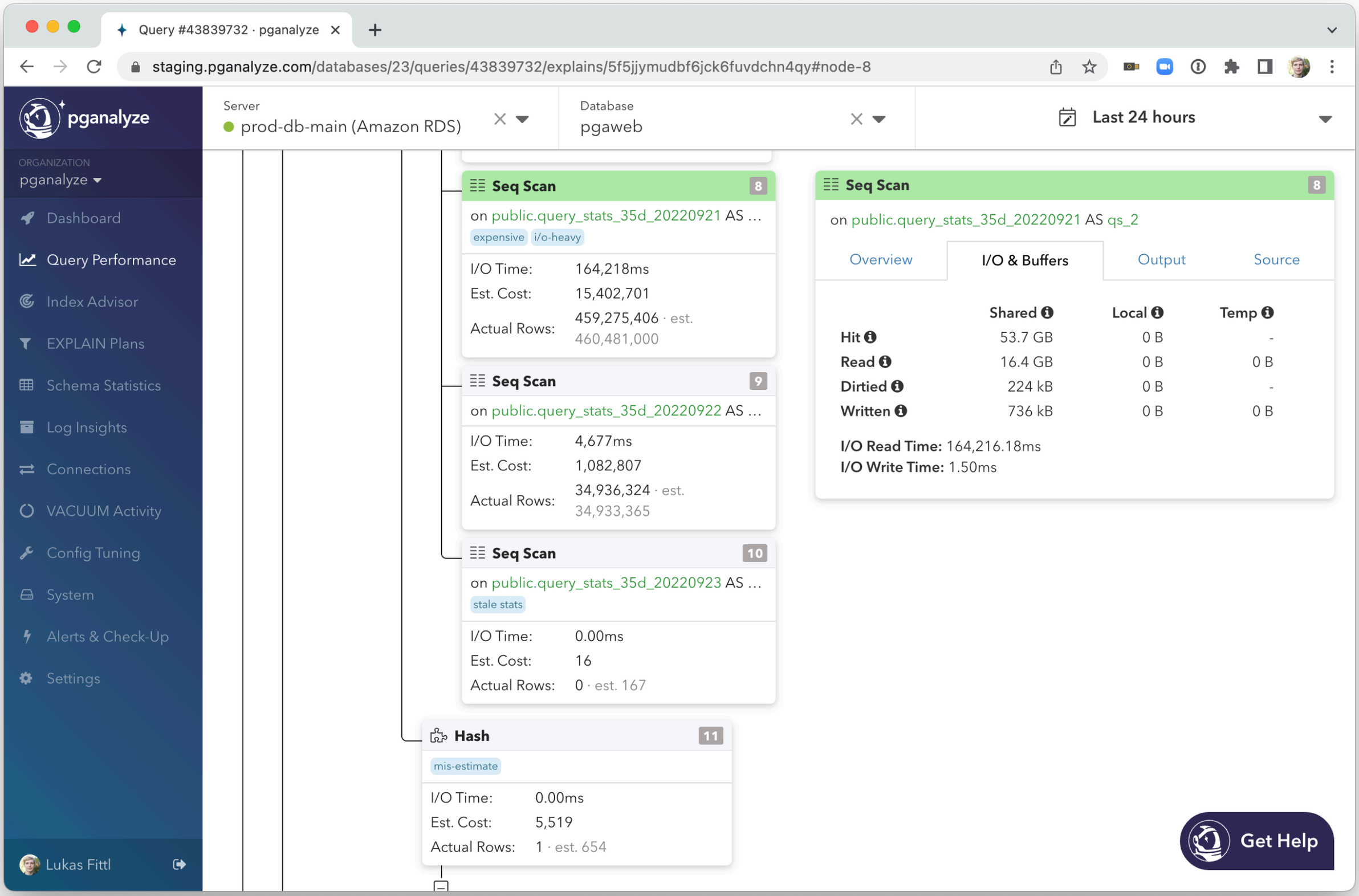
Plan node 6 was estimated to be expensive (cost 2477073 vs avg cost 969557)
Plan node 8 was estimated to be expensive (cost 15402701 vs avg cost 969557) and took 97% of total I/O time
Plan node 3 had rows over-estimated by 139376
Plan node 4 had rows over-estimated by 1254384
Plan node 5 had rows over-estimated by 1254384
Plan node 11 had rows over-estimated by a factor of 654
Plan node 12 had rows over-estimated by a factor of 654
Plan node 19 had rows over-estimated by 64975
Plan node 21 had rows over-estimated by 5083
Plan node 10 referenced a table that has not been analyzed recently
Plan node 14 referenced a table that has n… recently
Plan node 22 referenced a table that has not been analyzed

Get Help

pganalyze


Query #43839732 · pganalyze
staging.pganalyze.com/databases/23/queries/43839732/explains/5f5jjymudbf6jck6fuvdchn4qy#node-8
pganalyze
Server
prod-db-main (Amazon RDS)
Database
pgaweb
Last 24 hours
ORGANIZATION
pganalyze
Dashboard
Query Performance
Index Advisor
EXPLAIN Plans
Schema Statistics
Log Insights
Connections
VACUUM Activity
Config Tuning
System
Alerts & Check-Up
Settings
Lukas Fittl
Seq Scan 8
on public.query_stats_35d_20220921 AS ...
expensive i/o-heavy
I/O Time: 164,218ms
Est. Cost: 15,402,701
Actual Rows: 459,275,406 · est. 460,481,000
Seq Scan 9
on public.query_stats_35d_20220922 AS ...
I/O Time: 4,677ms
Est. Cost: 1,082,807
Actual Rows: 34,936,324 · est. 34,933,365
Seq Scan 10
on public.query_stats_35d_20220923 AS ...
stale stats
I/O Time: 0.00ms
Est. Cost: 16
Actual Rows: 0 · est. 167
Hash 11
mis-estimate
I/O Time: 0.00ms
Est. Cost: 5,519
Actual Rows: 1 · est. 654
Seq Scan 8
on public.query_stats_35d_20220921 AS qs_2
Overview
I/O & Buffers
Output
Source
EXPLAIN Insights
expensive was estimated to be expensive (cost 15402701 vs avg cost 969557)
i/o-heavy took 97% of total I/O time
Seq Scan
Scans through the entire table row-by-row in an arbitrary order. Learn more
Filter
(qs_2.collected_at >= $1::timestamp without time zone)
Rows Removed by Filter
37964423
Get Help

pganalyze


Query #43839732 · pganalyze
staging.pganalyze.com/databases/23/queries/43839732/explains/5f5jjymudbf6jck6fuvdchn4qy#node-8
pganalyze
Server
prod-db-main (Amazon RDS)
Database
pgaweb
Last 24 hours
ORGANIZATION
pganalyze
Dashboard
Query Performance
Index Advisor
EXPLAIN Plans
Schema Statistics
Log Insights
Connections
VACUUM Activity
Config Tuning
System
Alerts & Check-Up
Settings
Lukas Fittl
Seq Scan 8
on public.query_stats_35d_20220921 AS …
expensive i/o-heavy
I/O Time: 164,218ms
Est. Cost: 15,402,701
Actual Rows: 459,275,406 · est. 460,481,000
Seq Scan 9
on public.query_stats_35d_20220922 AS …
I/O Time: 4,677ms
Est. Cost: 1,082,807
Actual Rows: 34,936,324 · est. 34,933,365
Seq Scan 10
on public.query_stats_35d_20220923 AS …
stale stats
I/O Time: 0.00ms
Est. Cost: 16
Actual Rows: 0 · est. 167
Hash 11
mis-estimate
I/O Time: 0.00ms
Est. Cost: 5,519
Actual Rows: 1 · est. 654
Seq Scan 8
on public.query_stats_35d_20220921 AS qs_2
Overview
I/O & Buffers
Output
Source
Shared
Local
Temp
Hit 53.7 GB 0 B -
Read 16.4 GB 0 B 0 B
Dirtied 224 kB 0 B -
Written 736 kB 0 B 0 B
I/O Read Time: 164,216.18ms
I/O Write Time: 1.50ms
Get Help

pganalyze

Query #43839732 · pganalyze

staging.pganalyze.com/databases/23/queries/43839732/explains/5f5jjymudbf6jck6fuvdchn4qy#node-5

pganalyze

Server: prod-db-main (Amazon RDS)

Database: pgaweb

Last 24 hours

ORGANIZATION
pganalyze

- Dashboard
- Query Performance
- Index Advisor
- EXPLAIN Plans
- Schema Statistics
- Log Insights
- Connections
- VACUUM Activity
- Config Tuning
- System
- Alerts & Check-Up
- Settings

Lukas Fittl

Hash Join 5

mis-estimate

I/O Time:	168,895ms
Est. Cost:	20,268,608
Actual Rows:	0 · est. 1,254,384

Append 6

expensive

I/O Time:	168,895ms
Est. Cost:	18,962,597
Actual Rows:	494,211,730 · est. 495,414,533

Seq Scan 7

on public.query_stats_35d AS qs_1

I/O Time:	0.00ms
Est. Cost:	0
Actual Rows:	0 · est. 1

Seq Scan 8

on public.query_stats_35d_20220921 AS …

expensive i/o-heavy

I/O Time:	164,218ms
Est. Cost:	15,402,701
Actual Rows:	459,275,406 · est. 460,481,000

Hash Join 5

Overview | I/O & Buffers | Output | Source

EXPLAIN Insights

mis-estimate had rows over-estimated by 1254384

Hash Join

Scans through the outer table, looking for matches in a hash table built from the inner table data and keyed by join key. Learn more

Hash Cond

```
(qs.database_id = databases.id)
```

Inner Unique

```
true
```

Join Type

```
Inner
```

Get Help

pganalyze

Query #43839732 · pganalyze

staging.pganalyze.com/databases/23/queries/43839732/explains/5f5jjymudbf6jck6fuvdchn4qy#node-11

pganalyze

Server: prod-db-main (Amazon RDS)

Database: pgaweb

Last 24 hours

ORGANIZATION
pganalyze

- Dashboard
- Query Performance
- Index Advisor
- EXPLAIN Plans
- Schema Statistics
- Log Insights
- Connections
- VACUUM Activity
- Config Tuning
- System
- Alerts & Check-Up
- Settings

Lukas Fittl

Seq Scan 10

on public.query_stats_35d_20220923 AS …

stale stats

I/O Time:	0.00ms
Est. Cost:	16
Actual Rows:	0 · est. 167

Hash 11

mis-estimate

I/O Time:	0.00ms
Est. Cost:	5,519
Actual Rows:	1 · est. 654

Bitmap Heap Scan 12

on public.databases

mis-estimate

I/O Time:	0.00ms
Est. Cost:	5,519
Actual Rows:	1 · est. 654

Bitmap Index Scan 13

using index_databases_on_server_id_and…

I/O Time:	0.00ms
Est. Cost:	178
Actual Rows:	4,367 · est. 4,434

Hash 11

Overview | I/O & Buffers | Output | Source

EXPLAIN Insights

mis-estimate had rows over-estimated by a factor of 654

Hash

Reads data into a hash table, where it can be looked up by the hash key.
Learn more

Hash Batches
1

Hash Buckets
1024

Original Hash Batches
1

Original Hash Buckets
1024

Peak Memory Usage
9

Get Help

pganalyze

In this case the root cause of increased I/O was a
mis-estimate by the planner,
causing a Hash Join to be preferred over a Nested Loop
(the latter would have allowed an index to be used)

The rest of this story is for another day..

But, what if …

this was on

Amazon Aurora?

Query #43839732 · pganalyze

staging.pganalyze.com/databases/23/queries/43839732/explains/5f5jjymudbf6jck6fuvdchn4qy#node-13

pganalyze

Server: prod-db-main (Amazon RDS)

Database: pgaweb

Last 24 hours

ORGANIZATION
pganalyze

- Dashboard
- Query Performance
- Index Advisor
- EXPLAIN Plans
- Schema Statistics
- Log Insights
- Connections
- VACUUM Activity
- Config Tuning
- System
- Alerts & Check-Up
- Settings

Lukas Fittl

EXPLAIN Plan

2022-09-21 07:13:54pm PDT · Fingerprint: a38dae4

830,660.75ms	168,610.33ms	18.6 GB	21,330,255
Runtime	I/O Read Time	Read From Disk	Total Est. Cost

✔ Analyze ✔ Verbose ✔ Costs ✔ Buffers ✖ Timing ✖ Summary

Compare Query Plans · View EXPLAIN Source

EXPLAIN Insights

Plan node 6 was estimated to be expensive (cost 2477073 vs avg cost 969557)
Plan node 8 was estimated to be expensive (cost 15402701 vs avg cost 969557) and took 97% of total I/O time
Plan node 3 had rows over-estimated by 139376
Plan node 4 had rows over-estimated by 1254384
Plan node 5 had rows over-estimated by 1254384
Plan node 11 had rows over-estimated by a factor of 654
Plan node 12 had rows over-estimated by a factor of 654
Plan node 19 had rows over-estimated by 64975
Plan node 21 had rows over-estimated by 5083
Plan node 10 referenced a table that has not been analyzed recently
Plan node 14 referenced a table that has not been analyzed recently
Plan node 22 referenced a table that has not been analyzed recently

```
WITH slow_queries AS (
SELECT qs.database_id,
qs.fingerprint,
qs.postgres_role_id,
SUM(qs.total_time) / SUM(qs.calls) AS avg_time,
SUM(qs.shar...
```

Show full query text

Nested Loop 1

I/O Time: 168,895ms
Est. Cost: 21,330,255

Bitmap Index Scan

using index_databases_on_server_id_and_datname

Get Help

pganalyze

18.6 GB = 2380 Pages of 8kb size

4.55 queries / minute

~ 200,000 queries / month

~ 500,000,000 page reads / month

~ 500,000,000 page reads / month

PAGE CONTENT

Database Instances

Database Storage and IOs

Storage Rate	$0.10 per GB-month
I/O Rate	$0.20 per 1 million requests

~ $100/month just for this one slow read query...

(if we were running this database on Aurora)

pganalyze

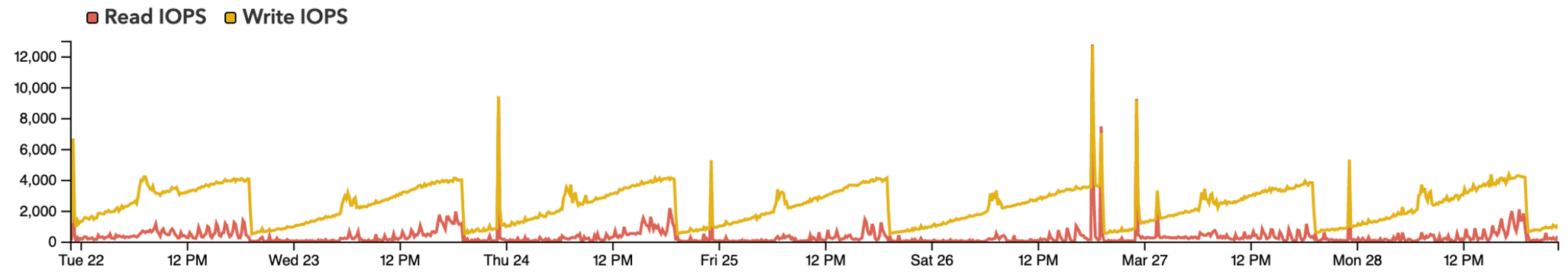
pganalyze

Indexes and Write Amplification

Bad Index Structure =

Lots Of Unnecessary Index Pages (and I/O!)

IOPS



pganalyze

```
CREATE TABLE public.log_line_stats_7d (
    log_line_id uuid DEFAULT public.gen_random_uuid() NOT NULL,
    server_id uuid NOT NULL,
    occurred_at_10min timestamp without time zone NOT NULL,
    occurred_at_1h timestamp without time zone NOT NULL,
    log_classification integer NOT NULL,
    database_id bigint,
    postgres_role_id uuid,
    schema_table_id bigint,
    query_fingerprint bytea
)
PARTITION BY RANGE (occurred_at_10min);
```

Use ULIDs for logs to reduce index write overhead #1942

Edit | Code

Merged **seanlinsley** merged 4 commits into `main` from `logs-ulid` on Apr 22

Conversation 14 | Commits 4 | Checks 4 | Files changed 5 | +182 −9

seanlinsley commented on Apr 11

https://brandur.org/nanoglyphs/026-ids#ulids
https://crates.io/crates/ulid

In the future we could further optimize storage by embedding `collected_at` inside the ULID, then extracting it with a SQL function when generating indexes and when reading from the table.

I'm not confident this will significantly improve the IO situation because there are much larger indexes than these, which one would expect take up more IO.

Indexes

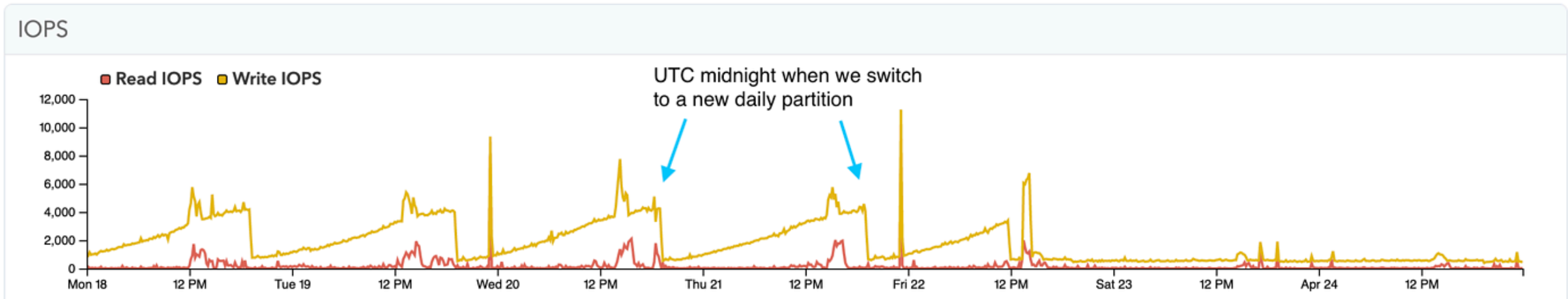
Name	Definition	Constraint	Valid?	First Seen	Size
log_lines_7d_20220407_pkey	btree (log_line_id)	PRIMARY KEY (log_line_id)	VALID	6 days ago	13.7 GB

Reviewers
lfittl
msakrejda

Assignees
No one—assign yourself

Labels
None yet

Projects



6,000 Write IOPS peak => ~1,000 Write IOPS peak

pganalyze

Unused (non-UNIQUE) Indexes =

Added write I/O without any benefit

Index Advisor

Overview | Missing Indexes | Unused Indexes

Total Data Size
6.5 TB
▲ 157.6 GB

Total Index Size
6 GB

Table Writes
852,414
per minute

Avg. Index Write Overhead
0.55
index bytes per table byte

Opportunities for Database (20)

Search...

IMPACT	TABLE	INDEX	INDEX SIZE	LAST USED	TABLE WRITES / MIN	INDEX WRITE OVERHEAD
▮▮▮▮▮	public.indexing_engine_runs_35d	indexing_engine_runs_35d_20220815_table_id_run_at_idx (+3…	37.3 MB	2022-08-17	0.000	-0.89
▮▮▯▯▯	public.snapshots	index_snapshots_on_snapshot_id	52.8 GB	2021-10-27	17,047.690	-0.26
▮▮▯▯▯	public.query_stats_35d	query_stats_35d_20220827_collected_at_idx (+1 more)	6.8 GB	2022-09-04	0.000	-0.23
▮▯▯▯▯	public.schema_table_options	index_schema_table_options_on_invalidated_at_snapshot_id	23.3 MB	2022-03-16	3.800	-0.17
▮▯▯▯▯	public.replication_stats	index_replication_stats_on_snapshot_id	1.1 GB	2022-08-29	225.997	-0.16
▮▯▯▯▯	public.replication_follower_stats	index_replication_follower_stats_on_snapshot_id	427.1 MB	2022-08-29	139.795	-0.15
▮▯▯▯▯	public.schema_columns	index_schema_columns_on_invalidated_at_snapshot_id	3.3 GB	2022-08-23	1,534.581	-0.13
▮▯▯▯▯	public.postgres_settings	index_postgres_settings_on_invalidated_at_snapshot_id	39.5 MB	2022-08-29	7.556	-0.12
▮▯▯▯▯	public.issue_states	index_issue_states_on_user_id	1.2 GB	2022-07-22	10.669	-0.10
▮▯▯▯▯	public.system_snapshots	index_system_snapshots_on_snapshot_id (+1 more)	381.3 MB	2021-10-27	0.000	-0.09
▮▯▯▯▯	public.invoices	index_invoices_on_organization_id	88 kB	2022-09-02	0.000	-0.07

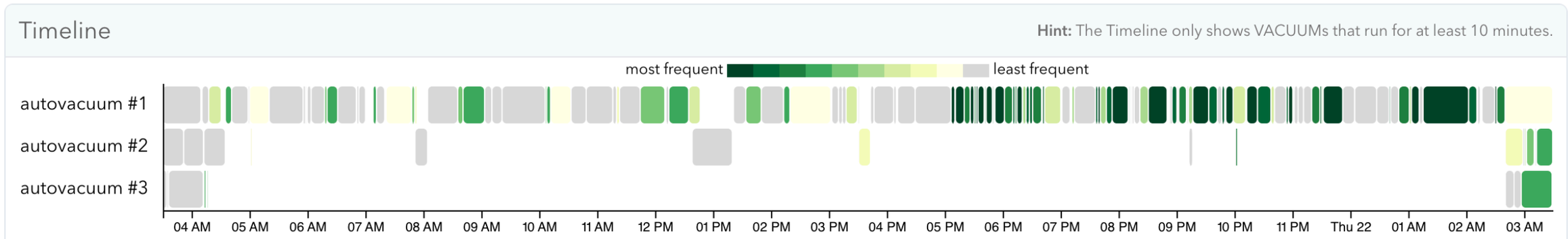
pganalyze

pganalyze

VACUUM Tuning and Table Bloat

VACUUM can directly cause I/O spikes

If a table hasn’t been vacuumed in a while,
autovacuum may schedule many
aggressive anti-wraparound VACUUMs



pganalyze

Make sure that large tables
get processed by autovacuum consistently
(e.g. by lowering the **autovacuum_scale_factor**)

Append only tables before PG13 may need manual VACUUM
(PG13+ has **autovacuum_vacuum_insert_scale_factor**)

On tables with a steady workload,
table bloat happens because
space was not reclaimed by VACUUM in time.

Root causes of autovacuum not doing its job

1. autovacuum can be too busy
(increase autovacuum_max_workers)

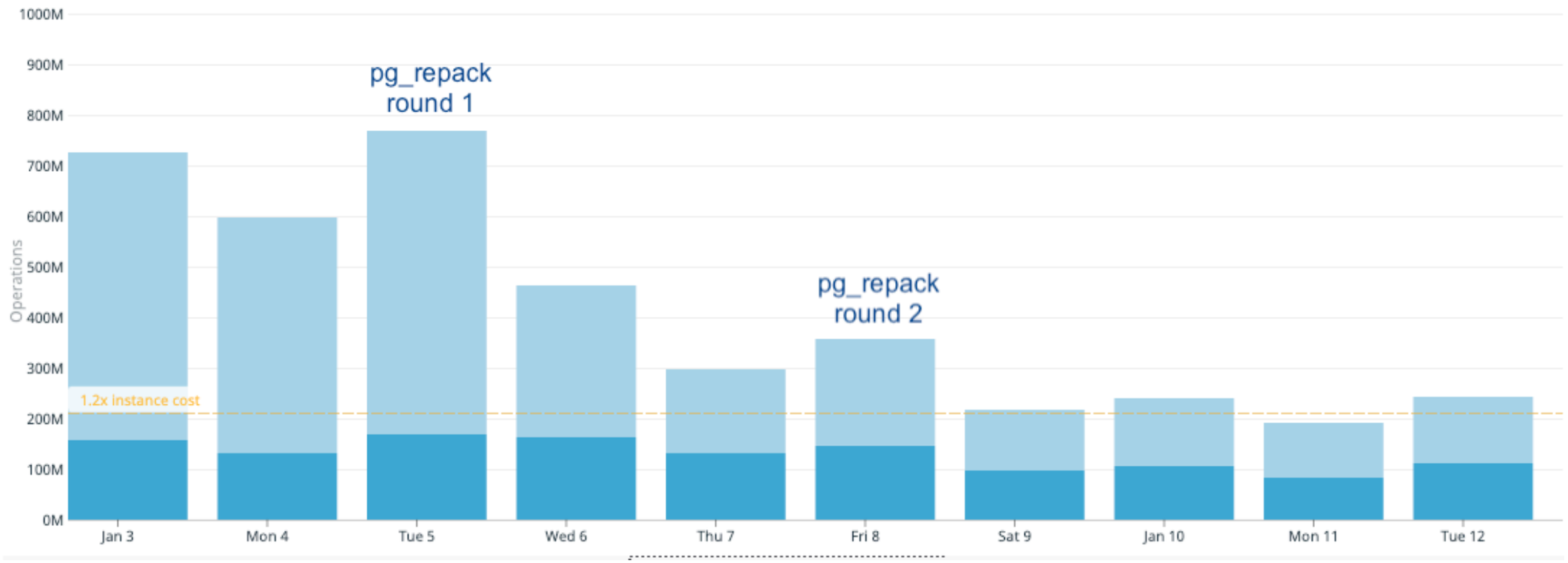
2. autovacuum can be too slow
(reduce autovacuum_vacuum_cost_delay)

3. autovacuum can be prevented from doing its work
(avoid long running transactions)

If table bloat happens:

Use **pg_repack** (aka VACUUM FULL without downtime)

Reduce Bloat, Tune Autovacuum => Save Money



“Migrating to Aurora: easy except the bill"

pganalyze

thanks!

Get a free trial of pganalyze

PGANALYZE.COM

Get free pganalyze eBooks and Postgres blog posts

PGANALYZE.COM/RESOURCES **PGANALYZE.COM/BLOG**

We will send you an email with a recording of this webinar tomorrow!

Feel free to get in touch with us at pganalyze.com/contact