Intro to LSM & B-Trees for Devs & DevOps

adyen

engineered
for ambition

DB Storage Engine (DevOps)

Heap (Un-Ordered)

LSM (Ordered)

Who am I?

Dave Pitts - Database Engineer - Adyen (Madrid)

adyen

engineered
for ambition

A typical week at Adyen?

Work From Home

chill/focus time

Office

human connection (CSR)

Traditional DB Storage (Devs)

Data Page (aka Heap) - UNORDERED

Data
Pages

Row Structure:

Primary Key

Field 1

Field 2

Field 3

...

8Kb - variable row counts

B-tree Pages - ORDERED

PK (OLTP)

Secondary Indexes (RI & OLAP)

B-Tree

Leaf Structure (array)

(Key, DataPage Pointer)

8Kb - 100s of children (PK)

I love 💚 pgbench

* Simple to use and highly adaptable bench marking tool

* --foreign-keys for FK constraints between tables

* scale factor 1, 2, 4, 8 ...
-s 1 of 1 i.e. 100,000 accounts
-s 2 of 2 i.e. 200,000 accounts
-s 4 of 4 i.e. 400,000 accounts
-s 8 of 8 i.e. 800,000 accounts

FKs & Secondary Index?

```
      constraint_name       |    child_table    | fk_columns |                suggested_index
----------------------------+-------------------+------------+---------------------------------------
 pgbench_accounts_bid_fkey | pgbench_accounts | bid         | CREATE INDEX ON pgbench_accounts(bid);
 pgbench_history_aid_fkey  | pgbench_history  | aid         | CREATE INDEX ON pgbench_history(aid);
 pgbench_history_bid_fkey  | pgbench_history  | bid         | CREATE INDEX ON pgbench_history(bid);
 pgbench_history_tid_fkey  | pgbench_history  | tid         | CREATE INDEX ON pgbench_history(tid);
 pgbench_tellers_bid_fkey  | pgbench_tellers  | bid         | CREATE INDEX ON pgbench_tellers(bid);
(5 rows)
```

Unindexed foreign keys
https://wiki.postgresql.org/wiki/Unindexed_foreign_keys

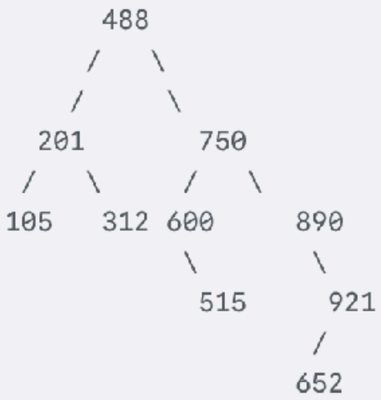
pgbench - sf1

```
sf1=# \d pgbench_accounts
              Table "public.pgbench_accounts"
  Column  |     Type      | Collation | Nullable | Default
----------+---------------+-----------+----------+---------
 aid      | integer       |           | not null |
 bid      | integer       |           |          |
 abalance | integer       |           |          |
 filler   | character(84) |           |          |
Indexes:
    "pgbench_accounts_pkey" PRIMARY KEY, btree (aid)
    "accounts_bid" btree (bid)
Foreign-key constraints:
    "pgbench_accounts_bid_fkey" FOREIGN KEY (bid) REFERENCES pgbench_branches(bid)
Referenced by:
    TABLE "pgbench_history" CONSTRAINT "pgbench_history_aid_fkey" FOREIGN KEY (aid) REFERENCES pgbench_accounts(aid)
```

Regular binary tree

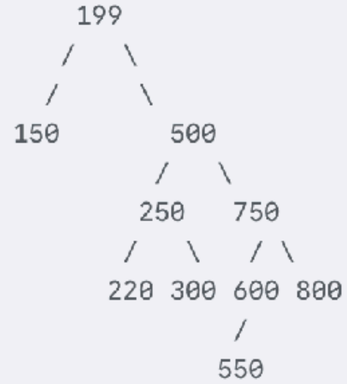
Example 1

Numbers: 488, 201, 750, 105, 312, 600, 890, 515, 652, 921



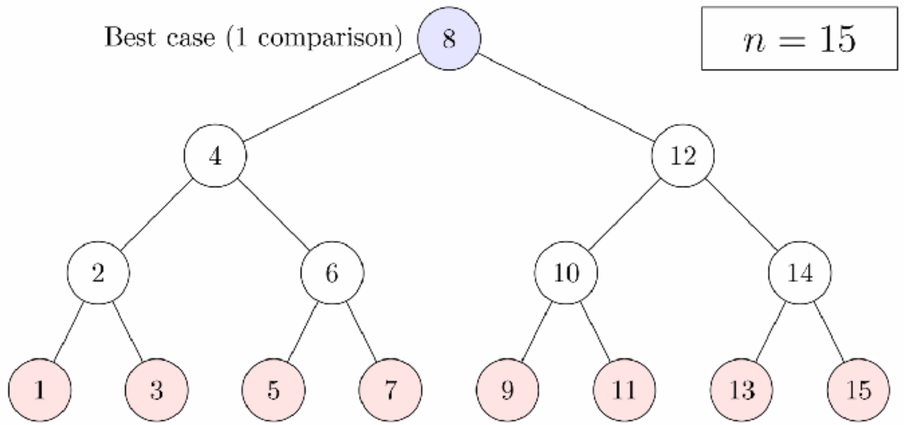
Example 2

Numbers: 199, 150, 500, 250, 750, 220, 300, 600, 800, 550



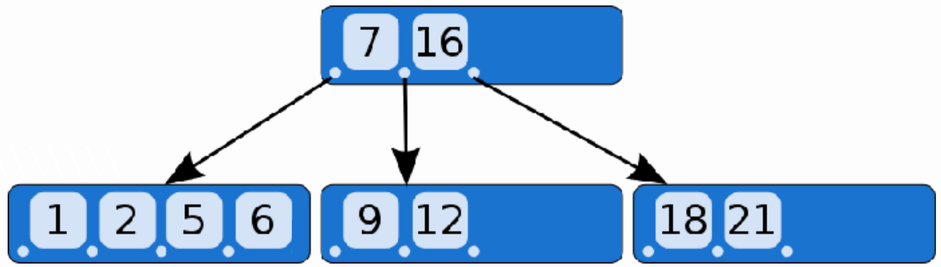
1970s balance tree fan-out (flat)

Binary-search tree



Worst case
$\lfloor \log_2(n) + 1 \rfloor$
comparisons)

Balanced Tree (f = fanout)



$\log_2(n)$ vs $\log_f(n)$
f = 4 or 200?

self-balancing algorithms 💪

What The Fanout?

Imagine a tree whose canopy is the world itself! (F=200?)

1. Fanout and Exponential Growth

- Suppose each B-tree node can branch to **F** children (the fanout).
- At depth 1 (the root), the tree covers at most **F** entries.
- At depth 2, it covers **F × F = F^2** entries.
- At depth 3, it covers **F × F × F = F^3** entries.
- In general, at depth *d*, the tree can cover up to **F^d entries**.
- That's **exponential growth** — every time you add a level, the capacity multiplies.

How Deep Is My Balanced Tree?

Now some really fun maths (optional) with logs to base 200 (i.e. F=200)

2. Logarithms and Tree Height

- If you know the number of rows NN and fanout F you can ask:
  How many levels (depth) does the tree need to hold N rows?

  - Solve for *d* in:

$$F^d \geq N$$

  Take logarithms:

$$d \geq \log_F(N)$$

- That's why lookups in a B-tree are **O(log n)**: the tree height grows only logarithmically with the number of rows.

Postgres Example

Real-world application: One billion rows, just four levels of depth!

- Assume:
  - Fanout ≈ **200** (typical for 8 KB pages with small keys).
  - Rows indexed: **1 billion (10^9)**.
- Tree depth needed:

$$d \approx \log_{200}(10^9)$$

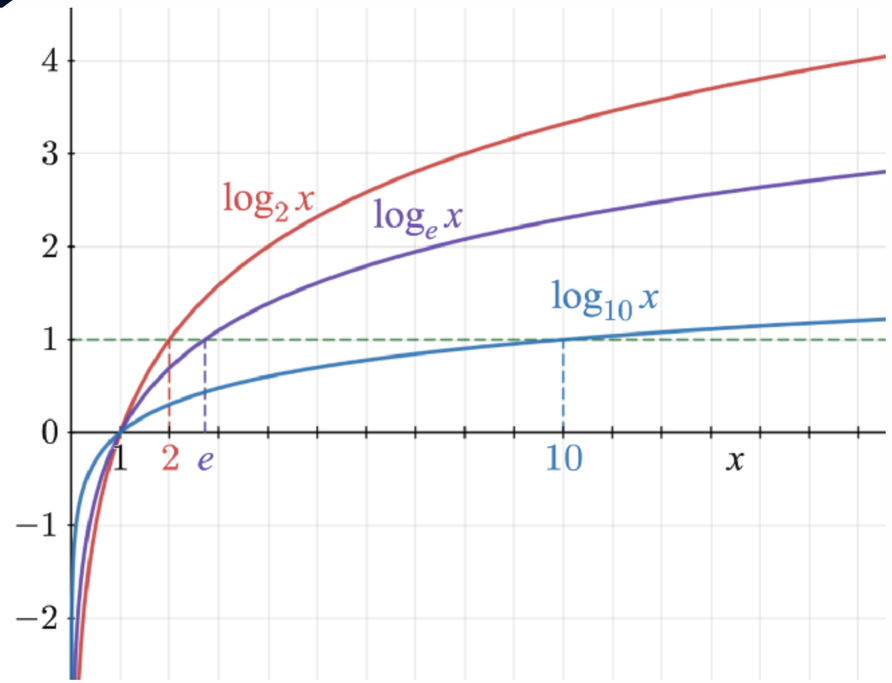
- Change base:

$$d = \frac{\log_{10}(10^9)}{\log_{10}(200)} = \frac{9}{2.3} \approx 4$$

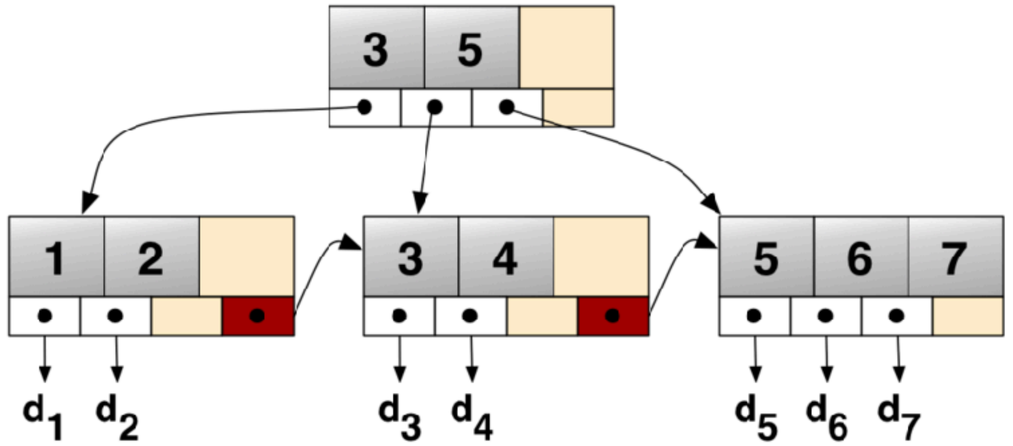
- Just **4 levels deep**!
  That means even with a **billion rows**, a lookup requires at most 4 page reads (root → internal → leaf → row).

Powerful maths and B+ trees

Log base 200 - “very flat”



B+Tree (linked list)



“where pk between 1 and 7” (ASC/DESC)

Btree Reads-per-Select

Scale Factor 4 (400,000 rows) has 4 Read (3 Index Pages & 1 Data Page)

```
bench_ff100_sf4=# explain (analyze,buffers) select * from pgbench_accounts where aid = 32928;
                                                          QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------
 Index Scan using pgbench_accounts_pkey on pgbench_accounts  (cost=0.42..8.44 rows=1 width=97) (actual time=0.527..0.528 rows=1 loops=1)
   Index Cond: (aid = 32928)
   Buffers: shared read=4
 Planning:
   Buffers: shared hit=62 read=8 dirtied=1
 Planning Time: 17.107 ms
 Execution Time: 0.573 ms
(7 rows)
```

Btree Reads-per-Select

Scale Factor 8 (800,000 rows) has 4 Read (still 3 Index Pages & 1 Data Page)

```
bench_ff100_sf8=# explain (analyze,buffers) select * from pgbench_accounts where aid = 32928;
                                                          QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------
 Index Scan using pgbench_accounts_pkey on pgbench_accounts  (cost=0.42..8.44 rows=1 width=97) (actual time=0.709..0.711 rows=1 loops=1)
   Index Cond: (aid = 32928)
   Buffers: shared read=4
 Planning:
   Buffers: shared hit=62 read=8 dirtied=3
 Planning Time: 10.082 ms
 Execution Time: 0.978 ms
(7 rows)
```

Btree Reads-per-Select

Keep on doubling (3 Index Pages & 1 Data Page)

bench_ff100_sf16, bench_ff100_sf32, bench_ff100_sf64 ….

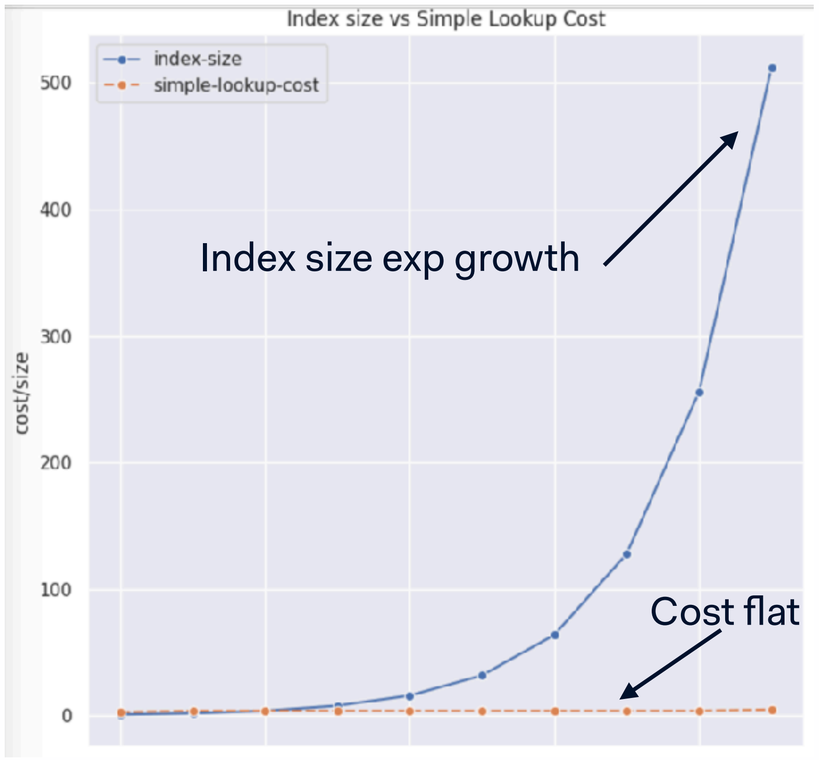
Btree Reads-per-Select

Scale Factor 512 (5.12 million rows) has 5 Read (4 Index Pages & 1 Data Page)

```
bench_ff100_sf512=# explain (analyze,buffers) select * from pgbench_accounts where aid = 32928;
                                                          QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------
 Index Scan using pgbench_accounts_pkey on pgbench_accounts  (cost=0.56..8.58 rows=1 width=97) (actual time=0.080..0.083 rows=1 loops=1)
   Index Cond: (aid = 32928)
   Buffers: shared hit=5
 Planning Time: 0.256 ms
 Execution Time: 0.212 ms
(5 rows)
```

Very efficient read structures

Size vs Lookups



DB Storage Engine (DevOps)

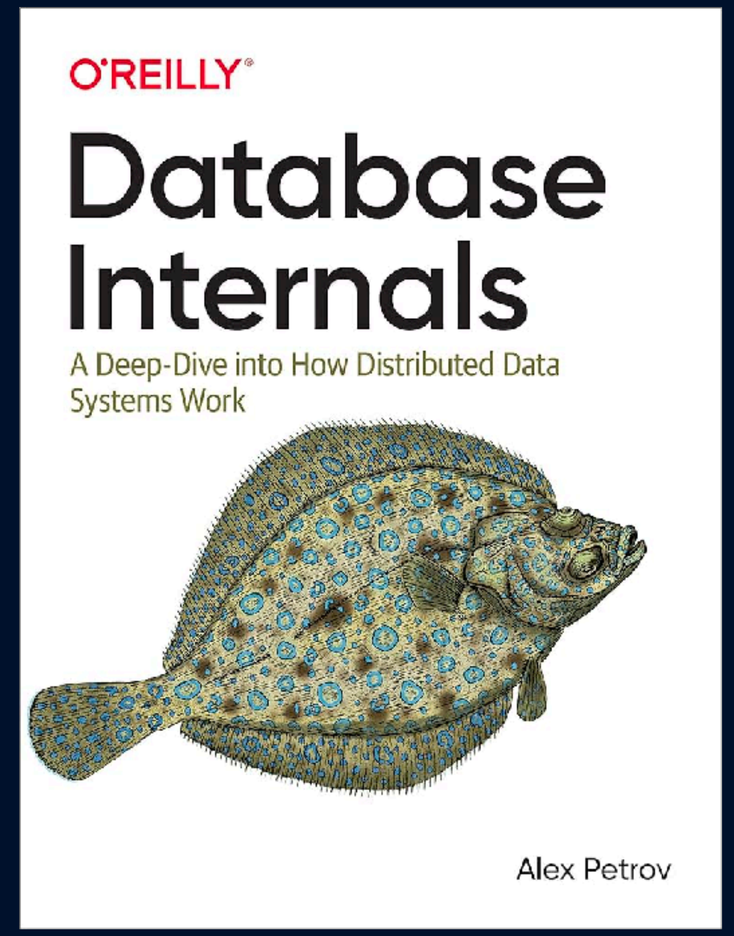
Heap (Un-Ordered)

LSM (Ordered)

Immutable Log Structures

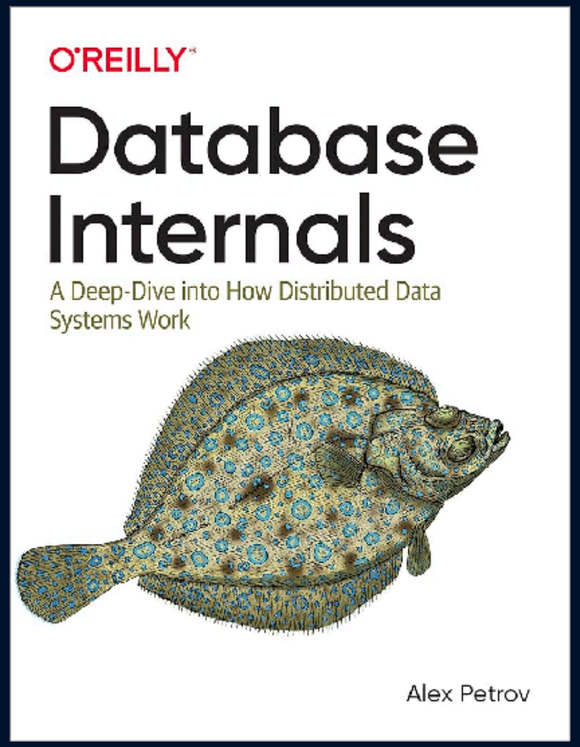
"Accountants don't use erasers or they end up in jail"

Key	Value
K1	AAA,BBB,CCC
K2	AAA,BBB
K3	AAA,DDD
K4	AAA,2,01/01/2015
K5	3,ZZZ,5623



Database Internals

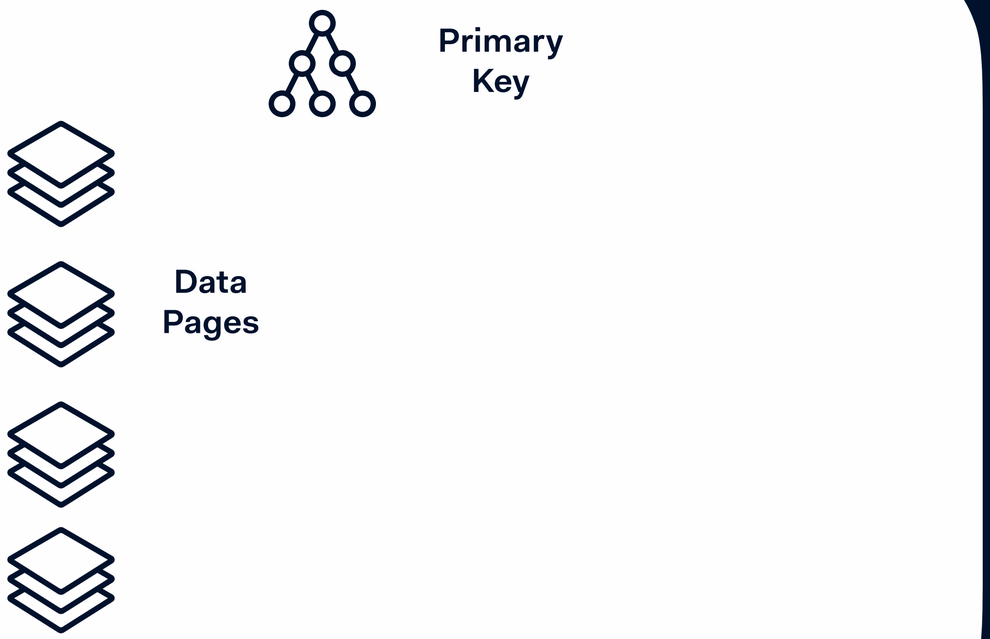
"A very fine line between indexes and tables”
"b-trees are heavily read optimized ... lsm-trees are heavily write optimized"



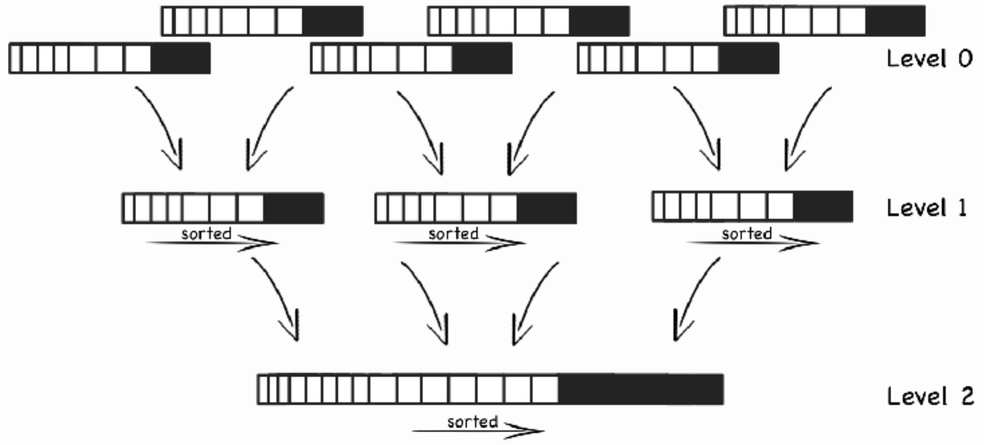
- SE Radio 417: Alex Petrov: Distributed Databases

Data Pages + PK ≃ LSM

Paged (aka Heap)



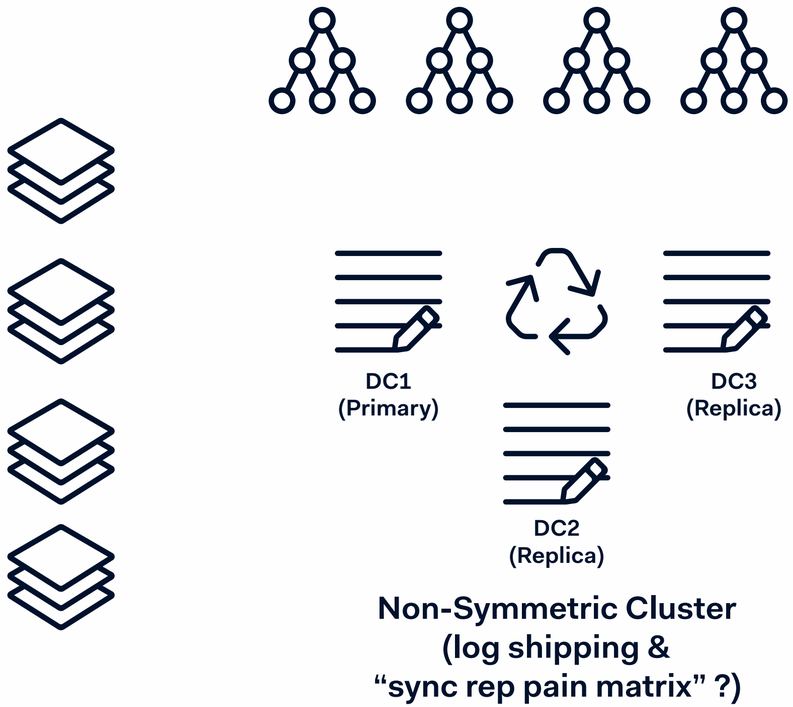
LSM Based (key-value)



Compaction continues creating fewer, larger and larger files

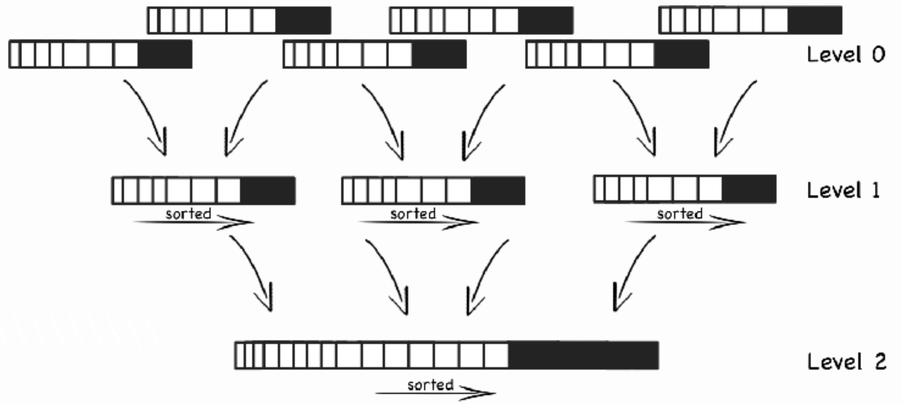
Replication Models?

Paged (aka Heap)

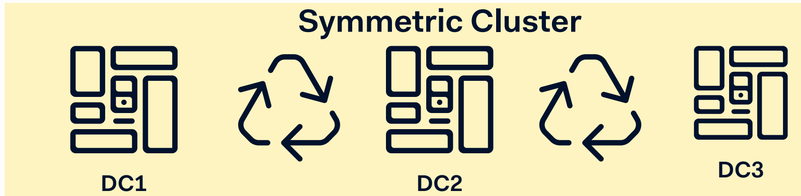


Non-Symmetric Cluster
(log shipping &
“sync rep pain matrix” ?)

Log Based (Quorum)



Compaction continues creating fewer, larger and larger files



RAFT (ligher/tighter real time consistency)

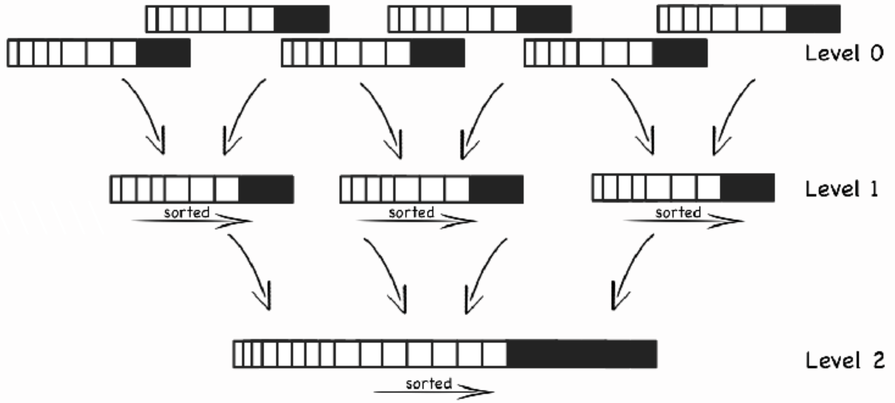
Reads & Bloom Filters

* Probabilistic (set-membership) data structure ... 1-2% false positives ?
* Based on multiple hash functions - feed into a compact bit array.
* Good for YES/NO question? My data is not in this file, but **maybe** it is in this files
* Read Amplification?
* Hashing does not support Range Scans, only atomic look ups!?

Visualizing LSM trees

Pyramid Build Blocks

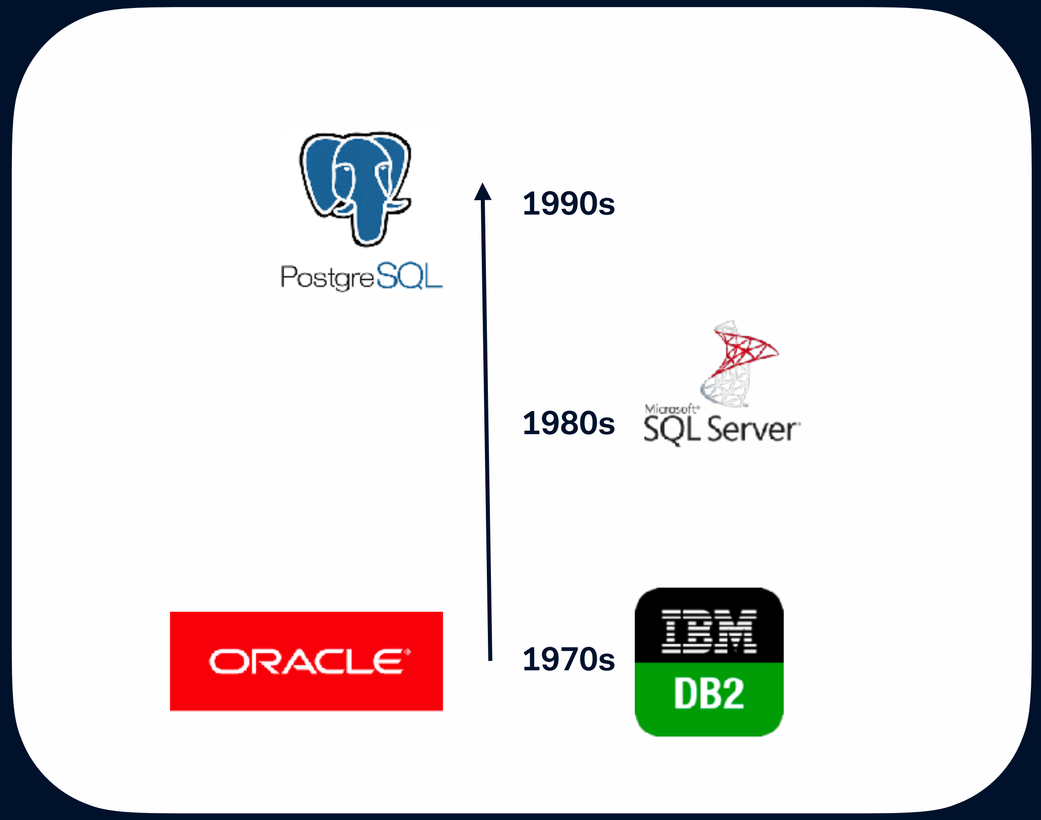
SSD table - triangulate



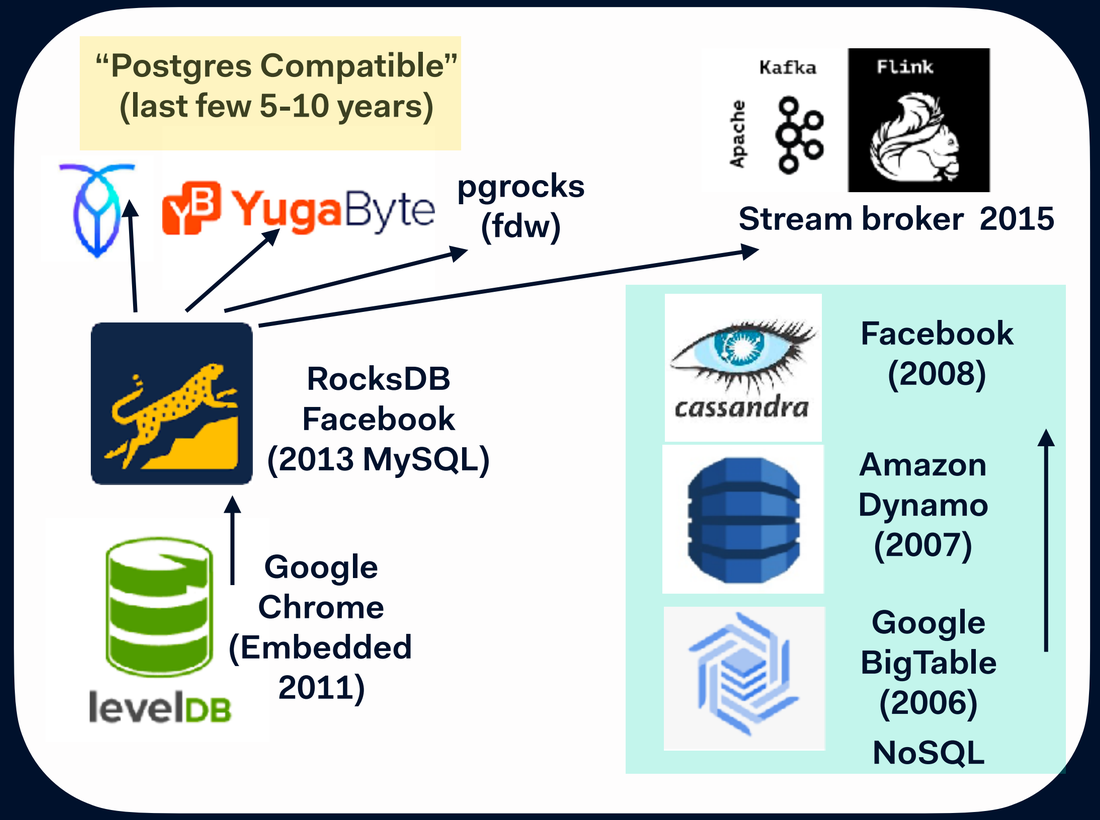
upto 5 or 6 levels and x10 growth?

Implementations - old & new

Paged/Heap (old world)



LSM (new world)

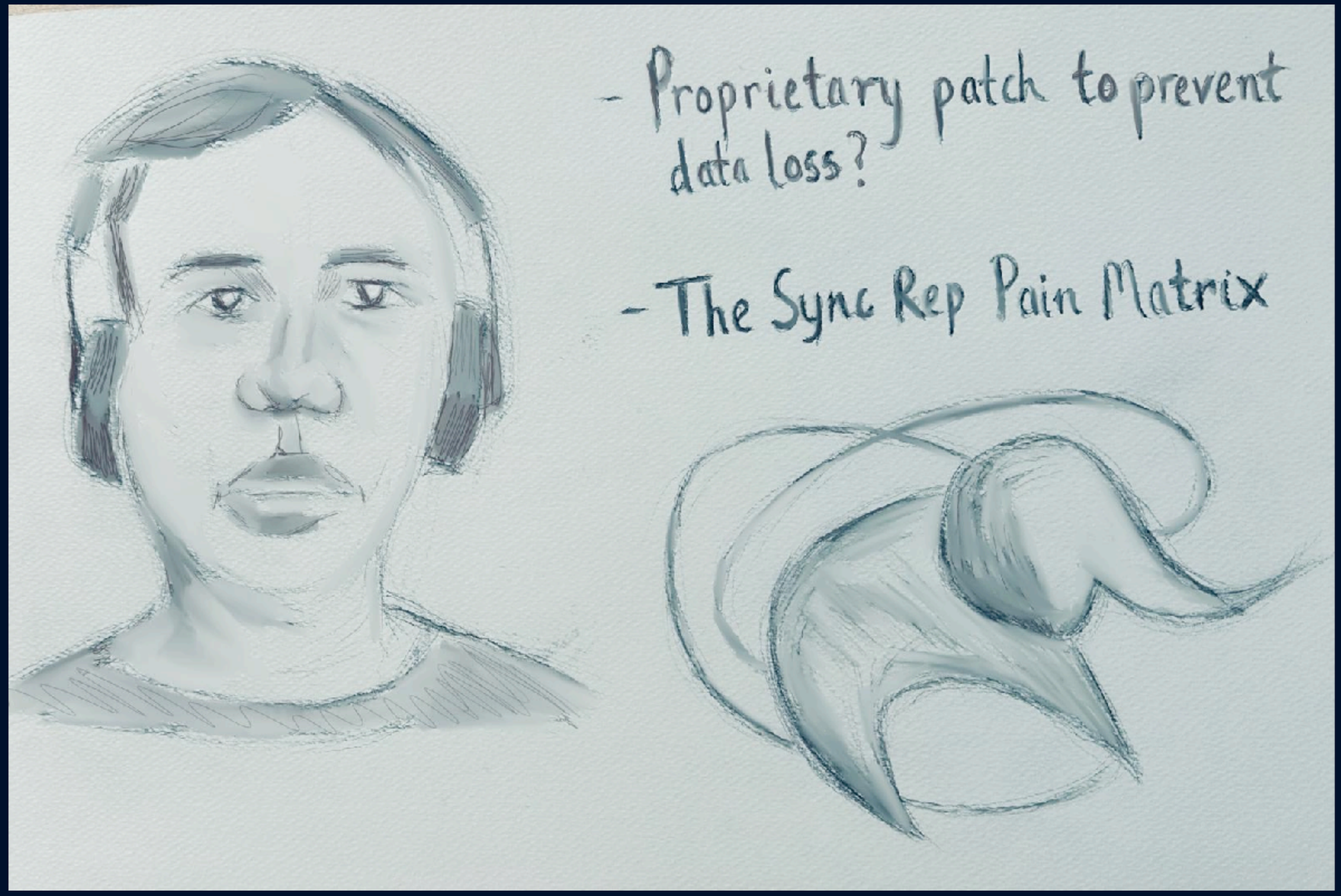


Flink LSM-tree synergy?

Flink's state access pattern is *append + point lookup* dominated:

Operation type	Frequency	Pattern
INSERT / PUT	Very frequent	New keys (e.g., new user/session/window).
UPDATE	Frequent but small	Overwrites of existing keys (e.g., count += 1). LSM handles this by appending a new version — not in-place modification.
DELETE	Moderate	Expiring state, old windows — handled via tombstones and compaction.
READ	Moderate	Random point lookups (e.g., lookup current counter).

Kukushkin & PG Community Pain



PostgreSQL High Availability Poker

Custom DB Topologies

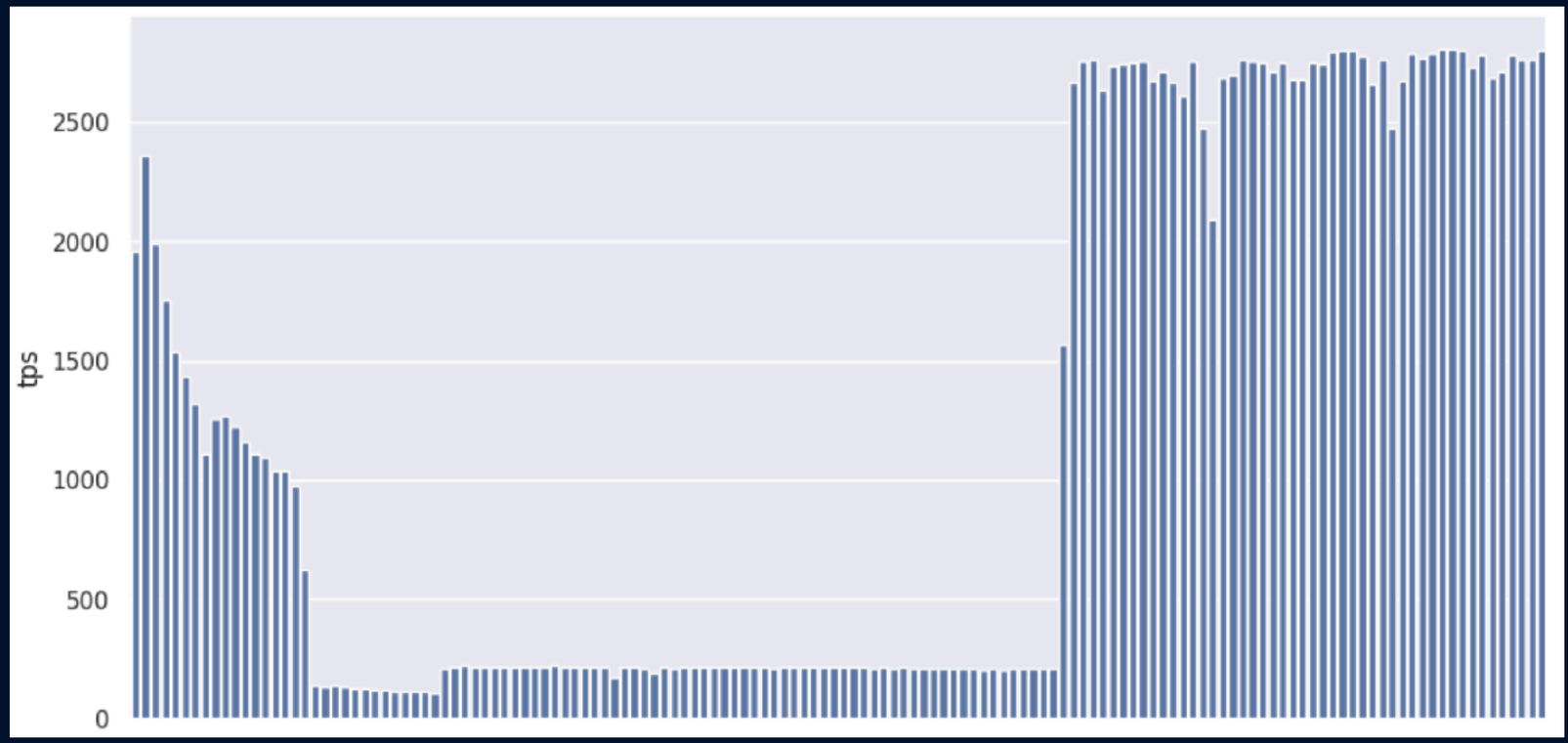
* Community PG Standalone - bread & butter
* Community PG Async Replicas - bread & butter
* Community PG Sync Replicas - bread & butter
* “PG Compatible” Highly distributed (LSM)
  - great for uptime (but use sparingly?)

pgbench against LSM trees?

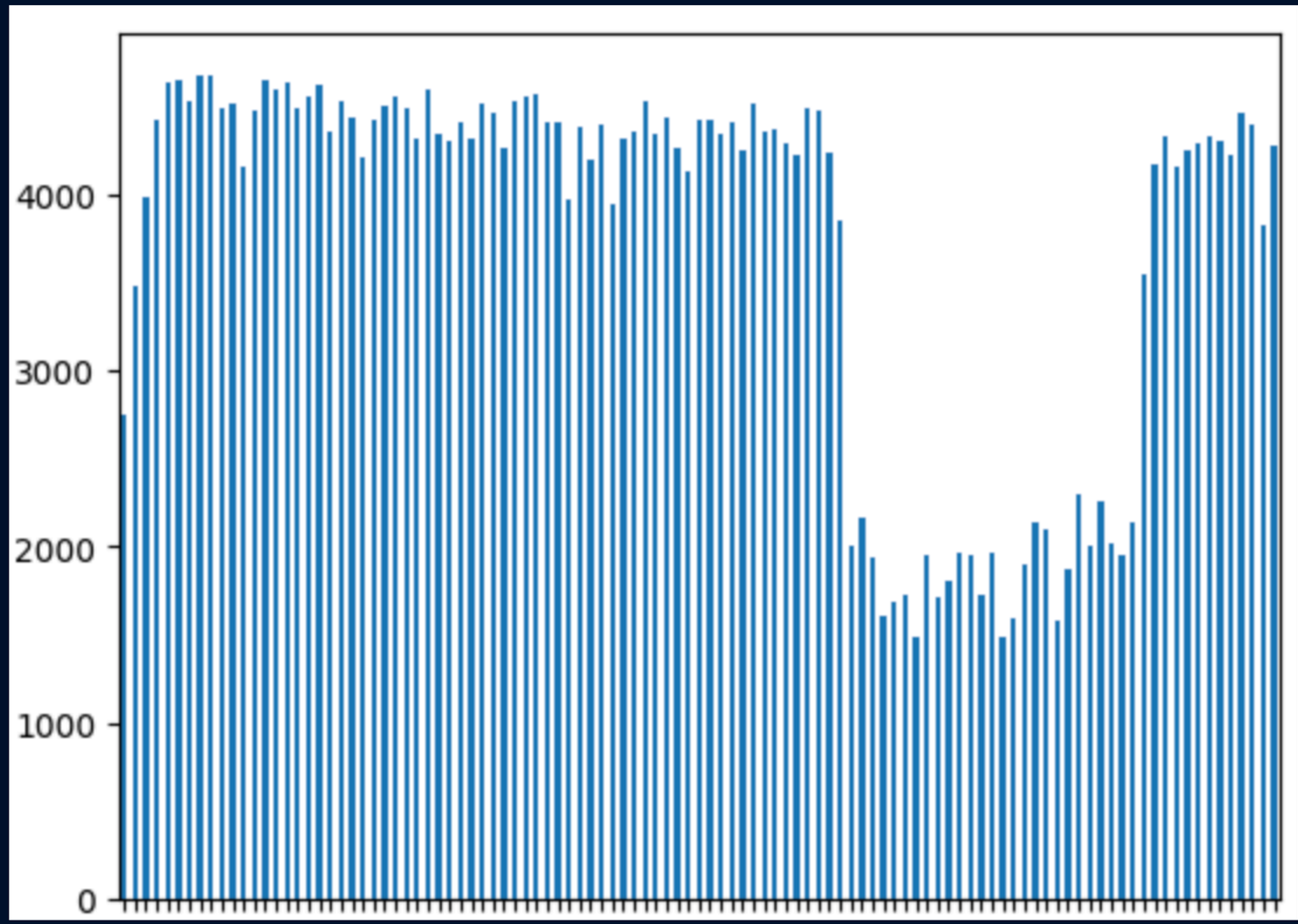
pgrocks (fdw)

Docker and pgbench scripts to test LSM trees on your (home/personal) laptop
https://github.com/dgapitts/pgday-paris-btree-lsm-demo

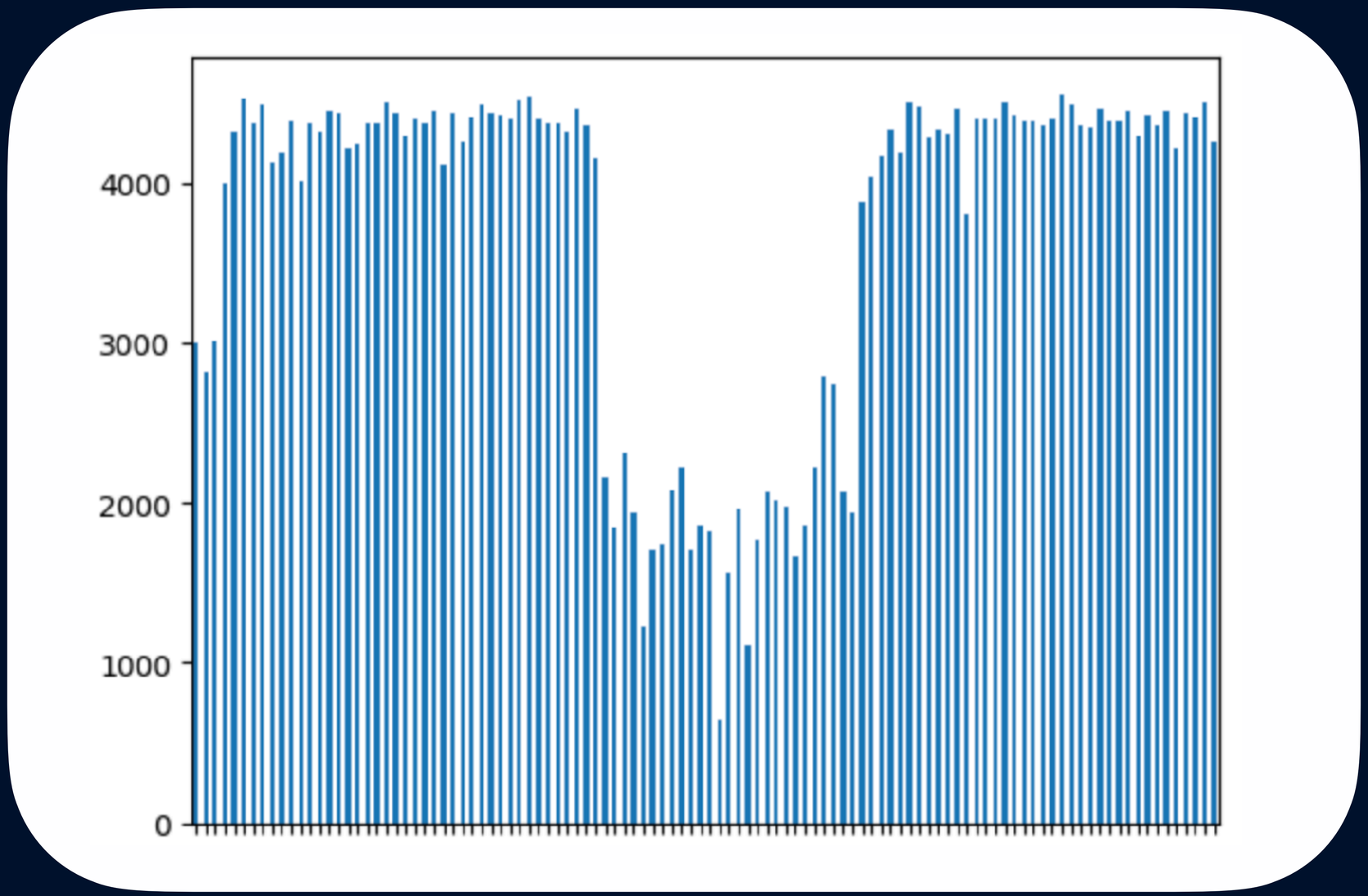
Long TXNs and TPS - pgrocks (fdw)



Drop Column - pgrocks (fdw)



Truncate Table - pgrocks (fdw)



Actual 4 vs Expected 17,103?

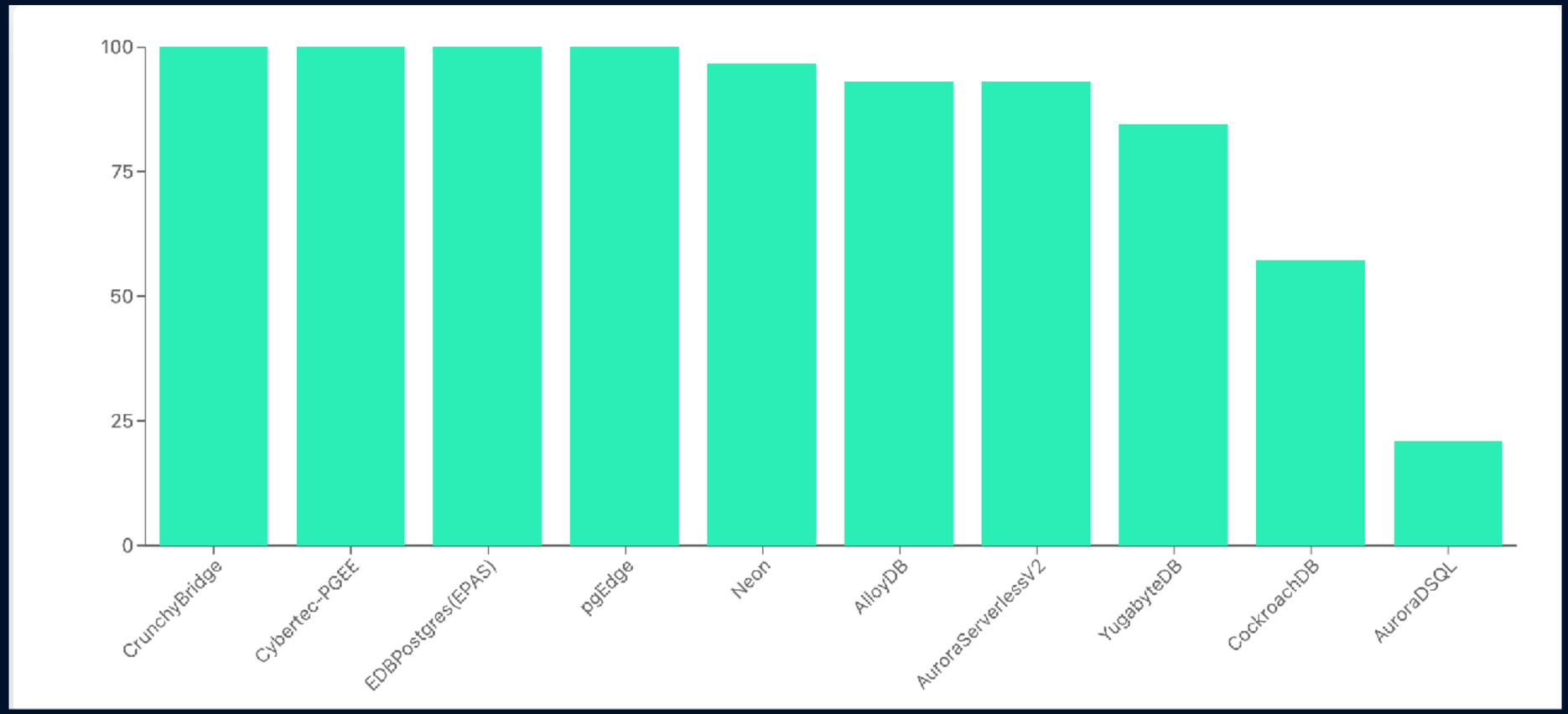
```
bench1=# analyze big_table;
analyze big_table;
WARNING:  'analyze' is a beta feature!
LINE 1: analyze big_table;
        ^
HINT:  Set 'ysql_beta_features' yb-tserver gflag to true to suppress the warning for all beta features.
ANALYZE
Time: 5743.785 ms (00:05.744)
```

```
bench1=# explain (analyze,buffers) select filler from big_table where id = 80000;
explain (analyze,buffers) select filler from big_table where id = 80000;
                                                              QUERY PLAN
_____________________________________________________________________________________________________________________
 Index Scan using big_table_id on big_table  (cost=0.00..2099.12 rows=17103 width=218)
                                                                                (actual time=8.996..9.050 rows=4 loops=1)
   Index Cond: (id = 80000)
 Planning Time: 7.983 ms
 Execution Time: 10.065 ms
 Peak Memory Usage: 8 kB
(5 rows)
```

adyen

engineered
for ambition

pgscorecard.com ?



Are LSM trees a niche use case?

Challenges

* Monitoring - this is really hard!!
* Compaction vs Vacuum (think ora-1555)
* Read amplification (mitigated by bloom filters)
* OLTP write heavy ✅ but may not OLTP update/read ❓ and not classic OLAP/HTAP ❌ (no index range scans?)

Further Tests

* Update Heavy 0%, 20%, 40%, 80%, 100% analysis
* Hot Key ranges, frequent updated / key customers
* Long Duration tests, do not just test daily jobs but monthly and annual too!

A few links

Microsoft Posette "Myths and Truths about Synchronous Replication in PostgreSQL" Alexander Kukushkin (hard DBA stuff)
https://www.youtube.com/watch?v=PFn9qRGzTMc

"PostgreSQL High Availability Poker" Adyen DEV training (from pgDay Lowlands)
https://www.youtube.com/watch?v=oEjj6ofjxpo

Software Engineering Radio E417 - Alex Petrov on LSMs trees (contributor to Cassandra)
https://se-radio.net/2020/07/episode-417-alex-petrov-on-database-storage-engines/

Docker and pgbench scripts to test LSM trees on your laptop
https://github.com/dgapitts/pgday-paris-btree-lsm-demo

pgDay Paris and Amsterdam Open Source Data Infra meetup
https://www.youtube.com/watch?v=TMKKZwWYY6A

Q & A

Me

Jobs

Code (WIP / scratch)

adyen

engineered
for ambition