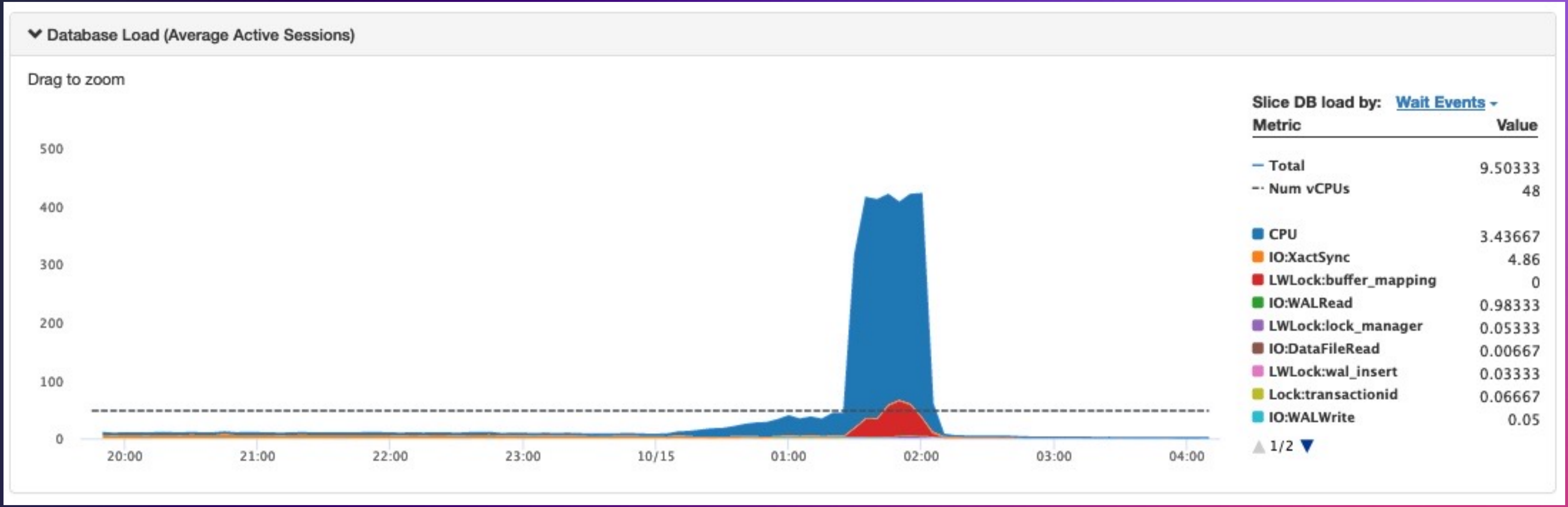
Fixing Broken Plans

Help the planner do it’s job!

David Rader (he/him)

Sr. Mgr Database Engineering
AWS RDS and Aurora

Our database was fine, then a plan flipped!



Database Load (Average Active Sessions)

Drag to zoom

Slice DB load by: Wait Events

Metric	Value
— Total	9.50333
-· Num vCPUs	48
🟦 CPU	3.43667
🟧 IO:XactSync	4.86
🟥 LWLock:buffer_mapping	0
🟩 IO:WALRead	0.98333
🟪 LWLock:lock_manager	0.05333
🟫 IO:DataFileRead	0.00667
■ LWLock:wal_insert	0.03333
■ Lock:transactionid	0.06667
■ IO:WALWrite	0.05

Help the planner do its job!

Parameters

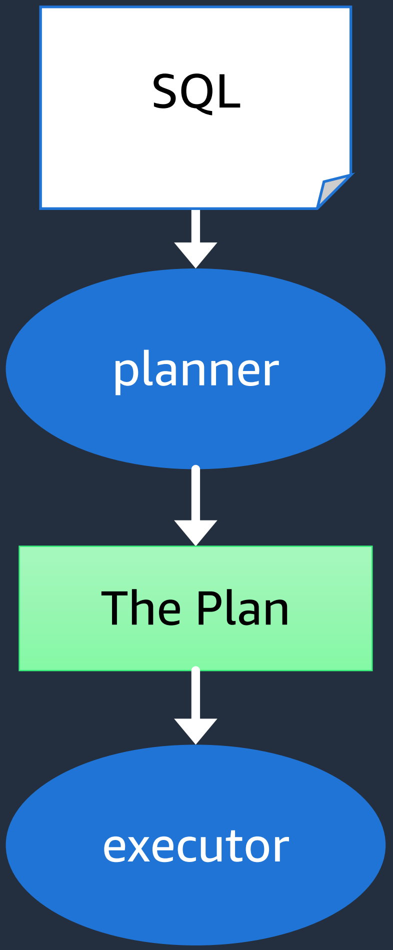
Statistics

Hints

Query Plan Management

The Planner's job

Convert your SQL to the **estimated** least cost plan to execute your query



- Access method per table
- Join order (every combination)
- Join methods

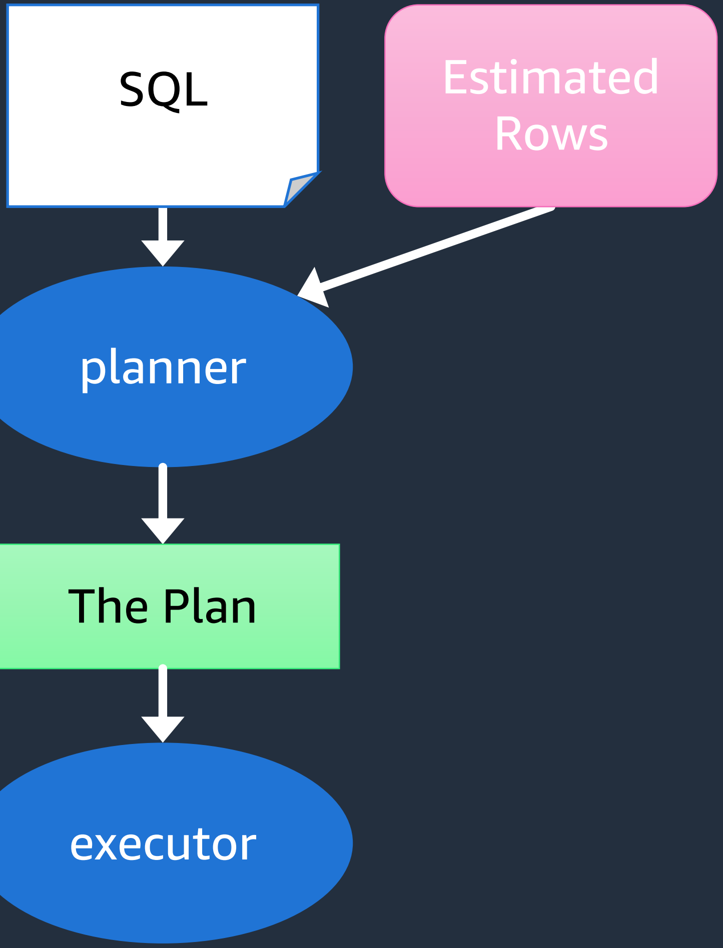
Sample plan and estimated costs

```
=> explain select *
from boarding_pass bp
     left join boarding_pass_details d
     on bp.pass_id = d.boarding_pass_id
where pass_id between 10 and 50;

                                 QUERY PLAN
------------------------------------------------------------------------
 Hash Right Join  (cost=128.01..1521595.27 rows=93 width=120)
   Hash Cond: (d.boarding_pass_id = bp.pass_id)
   ->  Seq Scan on boarding_pass_details d  (cost=0.00..1367611.00
rows=58611900 width=80)
   ->  Hash  (cost=127.51..127.51 rows=40 width=40)
         ->  Index Scan using boarding_pass_pkey on boarding_pass bp
 (cost=0.44..127.51 rows=40 width=40)
               Index Cond: ((pass_id >= 10) AND (pass_id <= 50))
(6 rows)
```

Sample plan and estimated costs

```
=> explain select *
from boarding_pass bp
     left join boarding_pass_details d
     on bp.pass_id = d.boarding_pass_id
where pass_id between 10 and 50;

                                 QUERY PLAN
------------------------------------------------------------------------

 Hash Right Join  (cost=128.01..1521595.27 rows=93 width=120)
   Hash Cond: (d.boarding_pass_id = bp.pass_id)
   ->  Seq Scan on boarding_pass_details d  (cost=0.00..1367611.00
rows=58611900 width=80)
   ->  Hash  (cost=127.51..127.51 rows=40 width=40)
         ->  Index Scan using boarding_pass_pkey on boarding_pass bp
 (cost=0.44..127.51 rows=40 width=40)
               Index Cond: ((pass_id >= 10) AND (pass_id <= 50))
(6 rows)
```

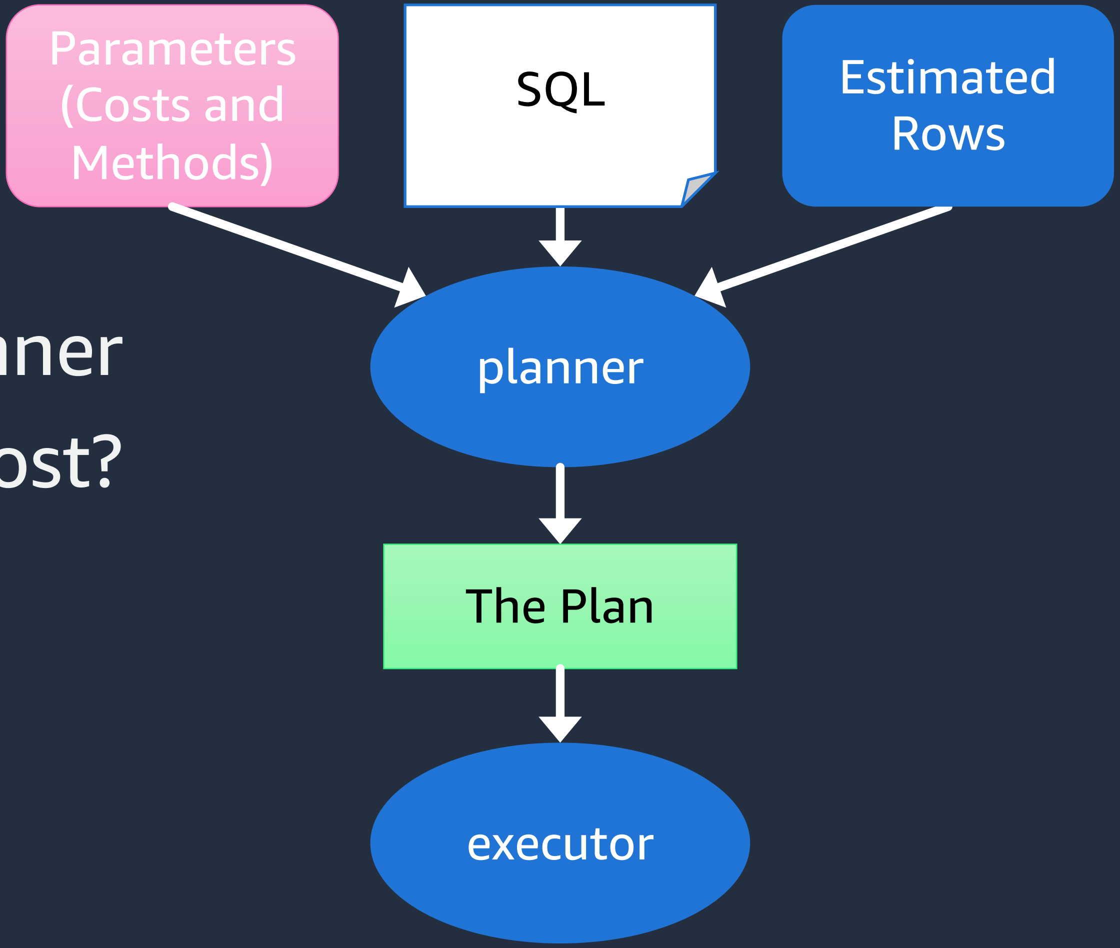
How does the planner estimate the cost?



Estimated rows

```
=> explain select *
from boarding_pass bp
     left join boarding_pass_details d
     on bp.pass_id = d.boarding_pass_id
where pass_id between 10 and 50;

                                 QUERY PLAN
------------------------------------------------------------------------
 Hash Right Join  (cost=128.01..1521595.27 rows=93 width=120)
   Hash Cond: (d.boarding_pass_id = bp.pass_id)
   ->  Seq Scan on boarding_pass_details d  (cost=0.00..1367611.00
rows=58611900 width=80)
   ->  Hash  (cost=127.51..127.51 rows=40 width=40)
         ->  Index Scan using boarding_pass_pkey on boarding_pass bp
 (cost=0.44..127.51 rows=40 width=40)
               Index Cond: ((pass_id >= 10) AND (pass_id <= 50))
(6 rows)
```

How does the planner **estimate** the cost?



Cost Parameters

seq_page_cost – Cost per page read in order (1.0)

random_page_cost - Lower value favors index scan **<=**

cpu_tuple_cost – Relative cost for processing vs IO

parallel_setup_cost – Cost to start parallel workers

…. and more:

https://www.postgresql.org/docs/current/runtime-config-query.html#RUNTIME-CONFIG-QUERY-CONSTANTS

`enable_` Parameters

Access m

(sequ x, bitmap index)

Jo thods

(ne loop, hash, me

Aggreg moize, and lot

https://www.p g-query.html#RUNTIME-CONFIG-QUERY-ENABLE

Other parameters to consider

`effective_cache_size` – Estimate pages in memory

`work_mem` – Memory per sort/hash operation

`max_parallel_workers_per_gather –` OLTP? Set 0

`geqo_threshold` (genetic query optimization)

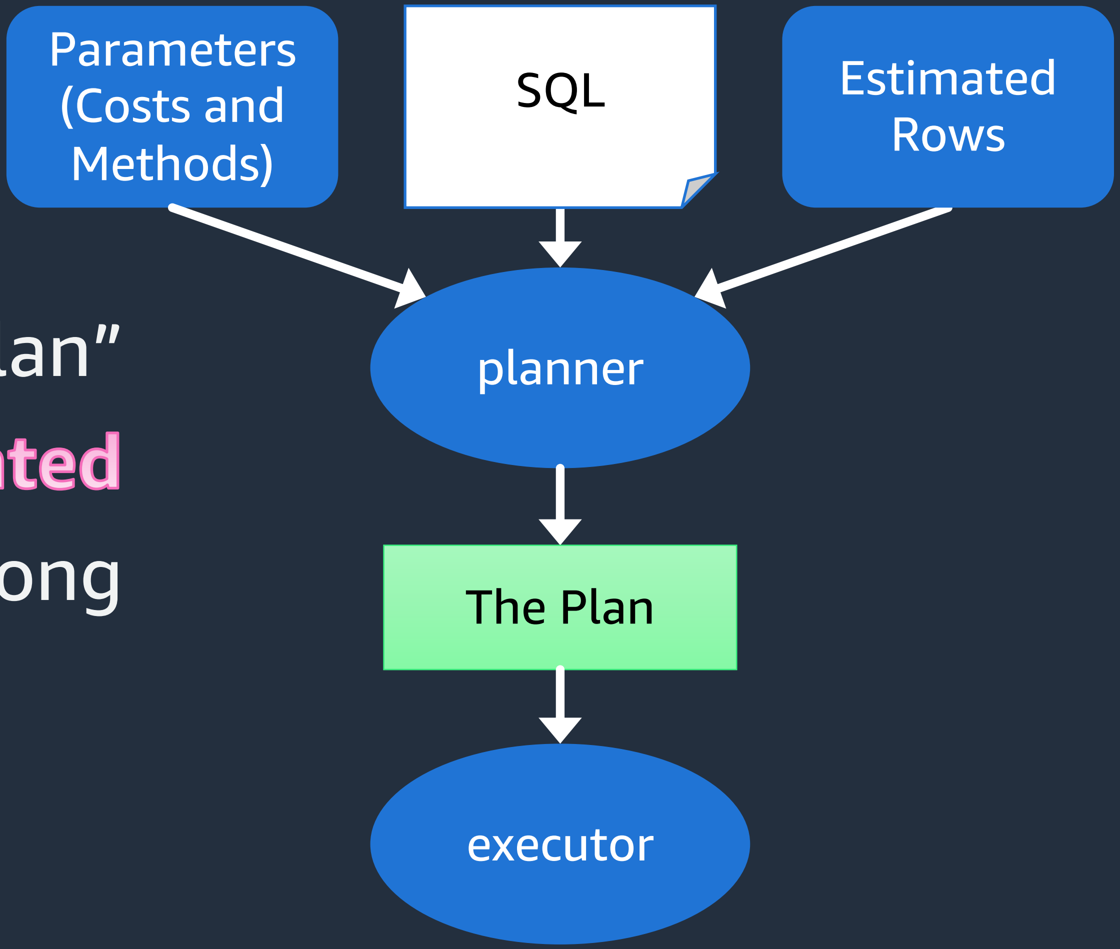
`plan_cache_mode –` prepared statements

“The query was fast the first 5 times, then slowed down.”

If the planner chooses a “bad plan”

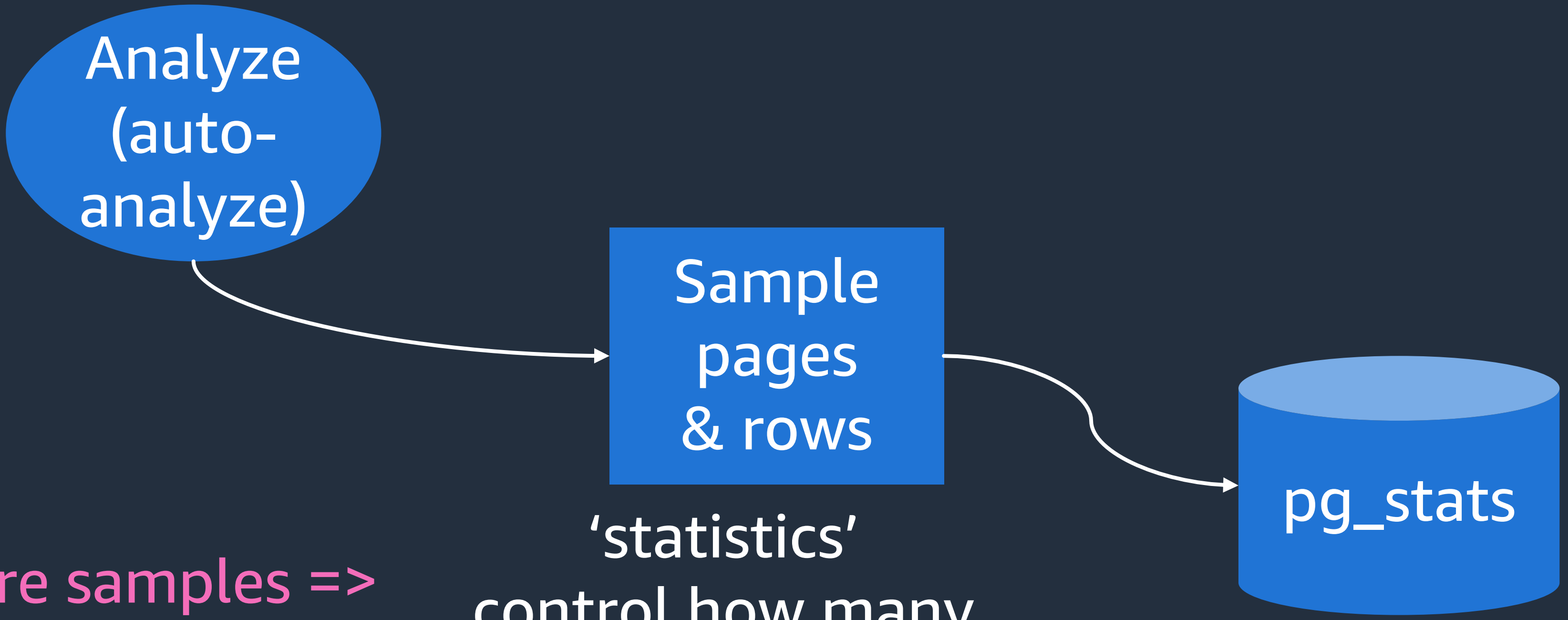
It means the **estimated**

Cost was wrong



Fixing Statistics to Estimate Rows

Statistics for estimating frequencies



‘statistics’
control how many

Per Attribute:

- most common values
- null fraction
- distinct count
- histograms

more samples => longer analyze, longer planning

Used by the planner at plan time

Beware! - Stats on large tables

Can have bad estimates for distinct values and histograms

Increase `default_statistics_target`

default for all tables, all statistics objects.

Set the statistics target for a single column

```
alter table alter column set statistics {target}
```

(Increases samples for entire table)

Or -- Set n_distinct yourself!

```
alter table alter column [COL] set n_distinct=[n or -ratio]
```

Extended Statistics (Multivariate)

If two or more of your columns are related to each.

(think height and weight)

```
CREATE STATISTICS (kind) ON (col1, col2) FROM tbl;
```

Must create explicitly
Populated (and updated) by analyze/autoanalyze

bad stats => bad plans

The most important part of query planning.

Major version upgrade – run ANALYZE

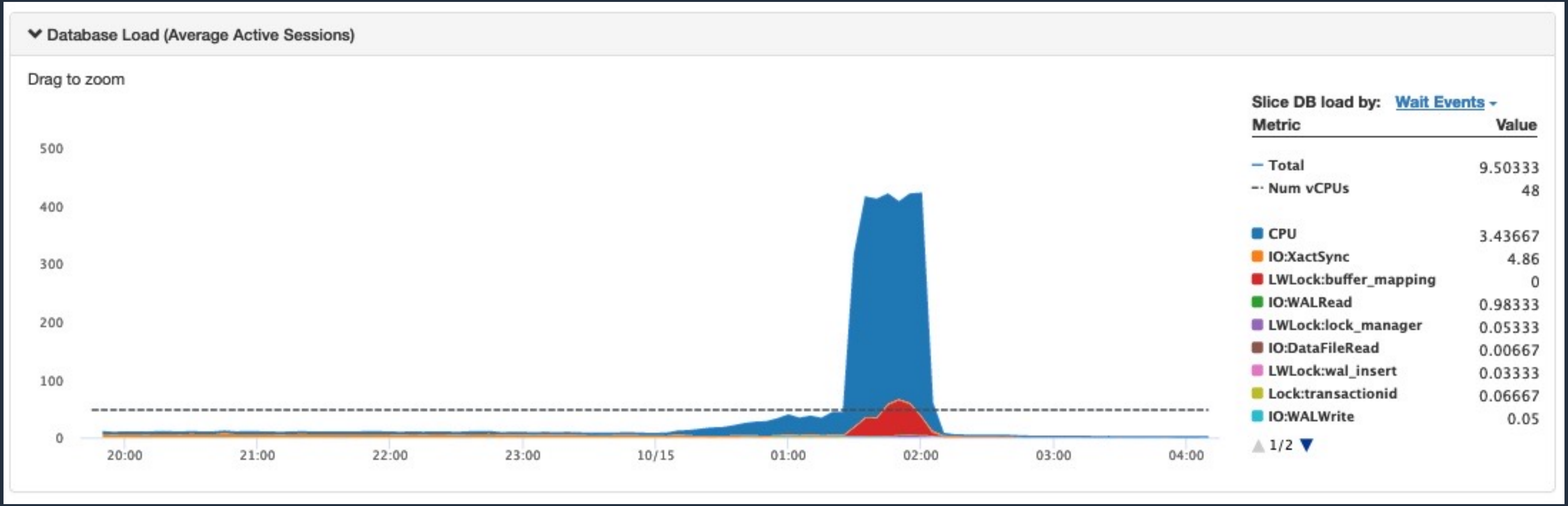
Leave autovacuum turned on

Check n_distinct on large tables

Watch out for (rapidly) changing data distributions

Like a new partition

Daily partitions!



Database Load (Average Active Sessions)

Drag to zoom

Slice DB load by: Wait Events

Metric	Value
— Total	9.50333
-- Num vCPUs	48
■ CPU	3.43667
■ IO:XactSync	4.86
■ LWLock:buffer_mapping	0
■ IO:WALRead	0.98333
■ LWLock:lock_manager	0.05333
■ IO:DataFileRead	0.00667
■ LWLock:wal_insert	0.03333
■ Lock:transactionid	0.06667
■ IO:WALWrite	0.05

▲ 1/2 ▼

Can I tell the planner what to do?

Partially – with hints

pg_hint_plan extension

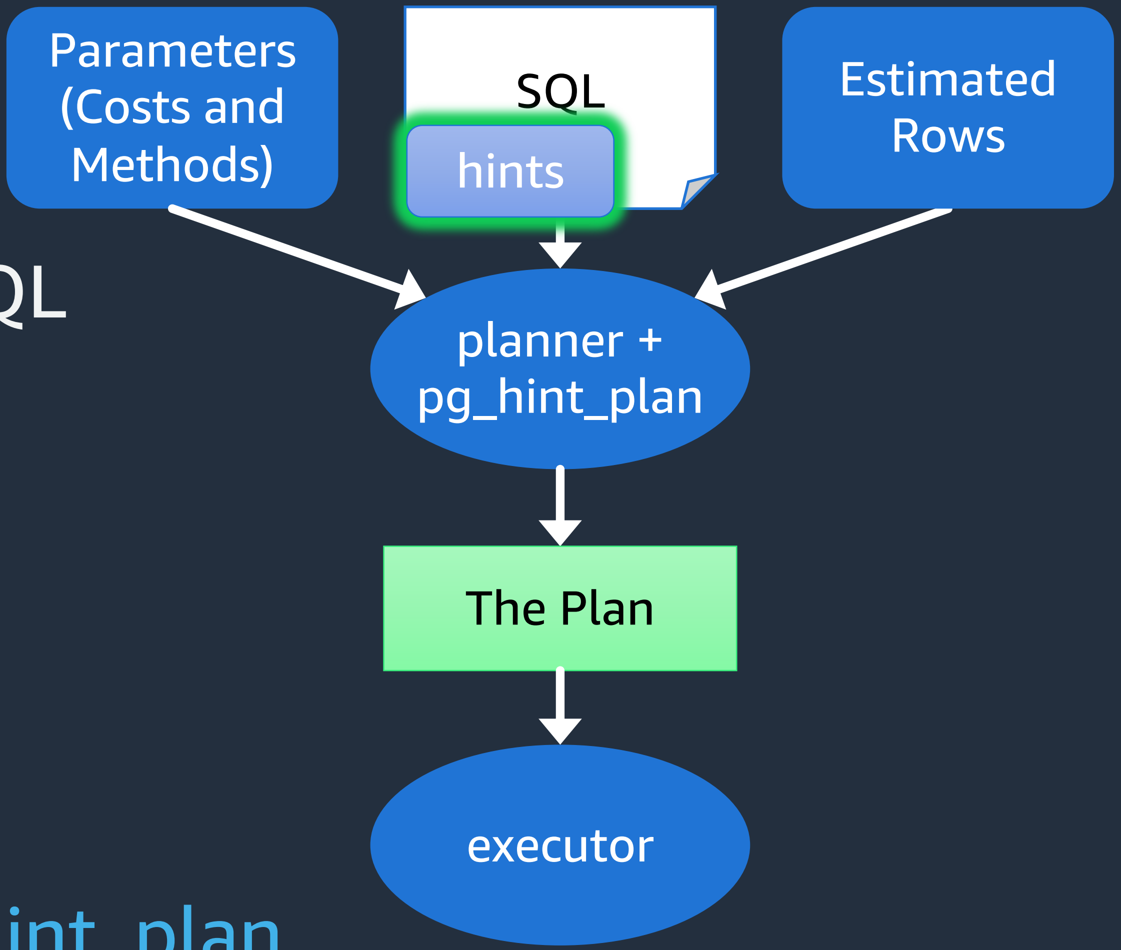
Hints are (one) comment in the SQL

/*+ IndexScan(tbl_a a_pk) */

/*+ NestLoop(tbl_a tbl_b) */

/*+ Leading((b a)) */

https://github.com/ossc-db/pg_hint_plan



Setup

```
=> show shared_preload_libraries;
                shared_preload_libraries
---------------------------------------------------
 rdsutils,pg_stat_statements,pg_hint_plan
(1 row)
```

Example

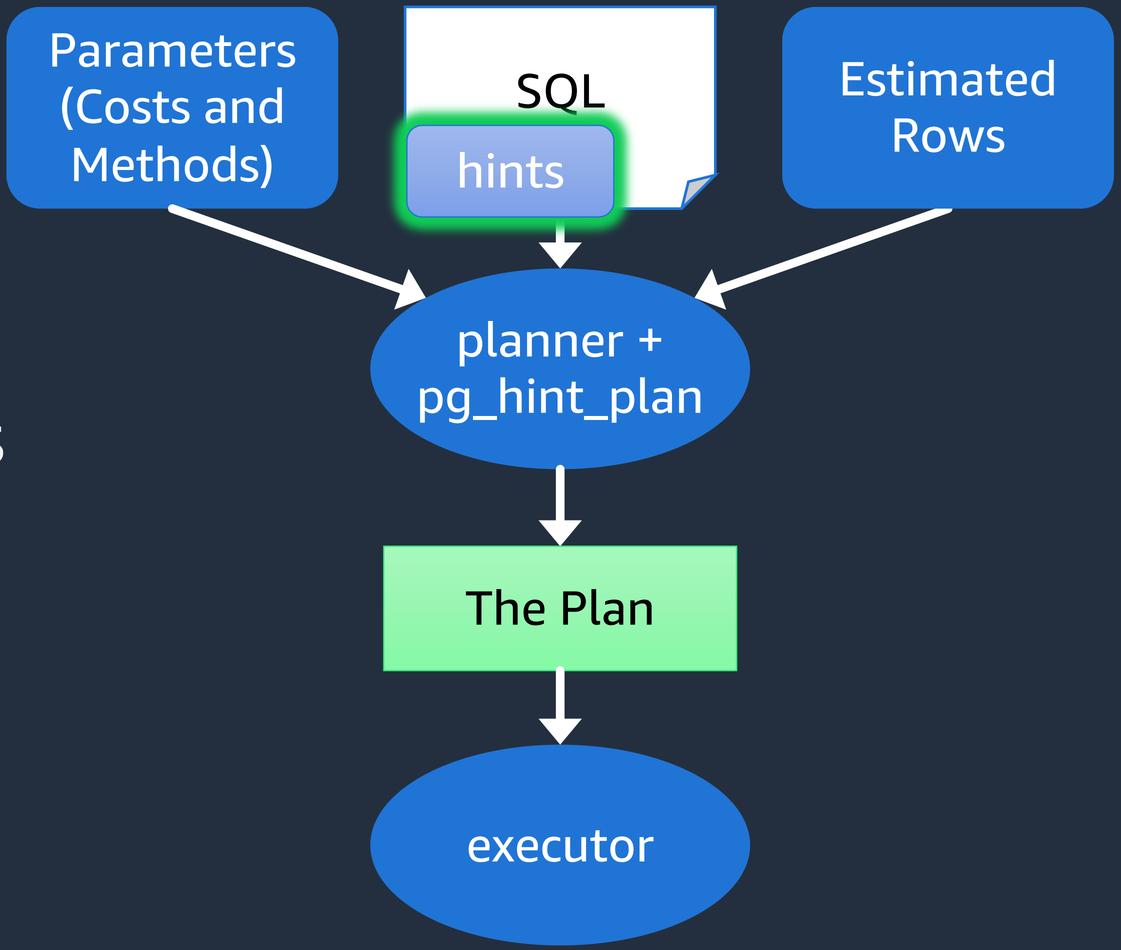
```
=> explain select * from boarding_pass_details where
boarding_pass_id between 10 and 17000000;

                    QUERY PLAN
---------------------------------------------------
 Seq Scan on boarding_pass_details
(cost=0.00..5100000.00 rows=169356713 width=80)
   Filter: ((boarding_pass_id >= 10) AND
(boarding_pass_id <= 17000000))
(2 rows)
```

Example

```
=> explain /*+ IndexScan(boarding_pass_details) */
select * from boarding_pass_details where
boarding_pass_id between 10 and 17000000;

            QUERY PLAN
----------------------------------------------------
 Index Scan using boarding_pass_details_pass_id on
boarding_pass_details  (cost=0.57..6426635.83
rows=169356713 width=80)
   Index Cond: ((boarding_pass_id >= 10) AND
(boarding_pass_id <= 17000000))
(2 rows)
```

How do hints work?

Hints increase the estimated costs of **other** options



Do I have to change app SQL?

No!

Use hint_plan.hints table

Hint for a normalized sql statement (? Instead of parameters)

```
=> create extension pg_hint_plan;
CREATE EXTENSION
=> set pg_hint_plan.enable_hint_table=1;
SET
```

Using hints table

```
=>insert into hint_plan.hints (norm_query_string,
application_name, hints)
values
  ('explain select * from boarding_pass_details where
boarding_pass_id between ? and ?;'
  , 'psql'
  , 'SeqScan(boarding_pass_details)'
  );
```

Using hints table

```
=> explain select * from boarding_pass_details where
boarding_pass_id between 10 and 1000000;
                                 QUERY PLAN
----------------------------------------------------------------
 Gather  (cost=1000.00..4522536.90 rows=9965369 width=80)
   Workers Planned: 2
   ->  Parallel Seq Scan on boarding_pass_details
(cost=0.00..3525000.00 rows=4152237 width=80)
         Filter: ((boarding_pass_id >= 10) AND (boarding_pass_id
<= 1000000))
(4 rows)
```

Read the docs carefully!

Table names in hints are case sensitive

Must match what is in pg_class.relname - not in the query

If you alias a table, use the alias in the hint

If you use the same table twice (subselects), find the auto-generated alias in the explain to use in your hint

Debugging hints – turn on log messages

```
=> set pg_hint_plan.debug_print=verbose;
SET
=> set client_min_messages = log;
SET
```

Can I tell the planner which plan to run?

Yes – with QPM

Query Plan Management

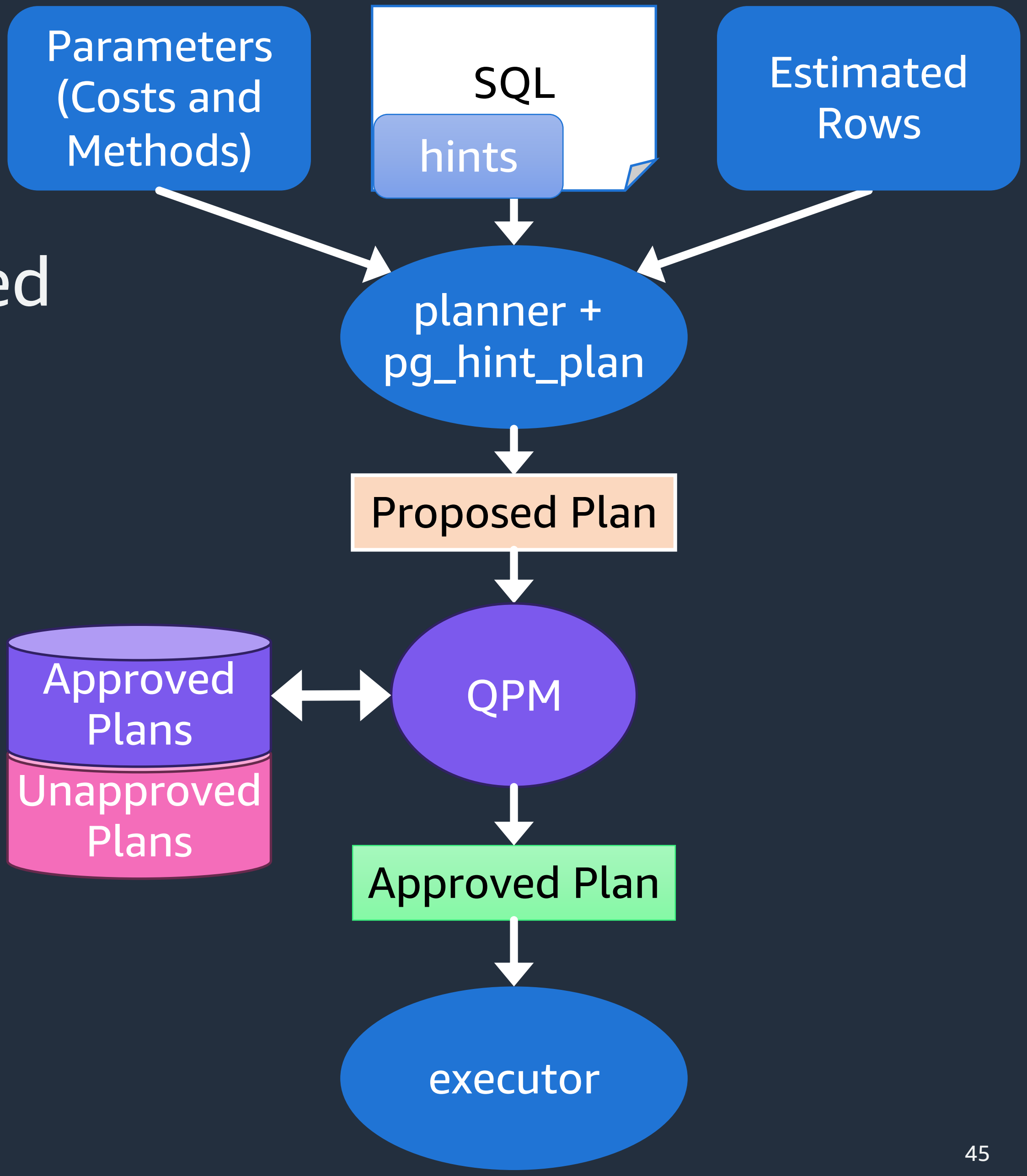
Enforce that only a “known” approved plan is run

If proposed plan is approved, run it!

If not, QPM chooses the lowest estimated cost approved plan to run

You can have one or more approved plans per statement

When QPM sees a new (unknown) plan, it is saved for later

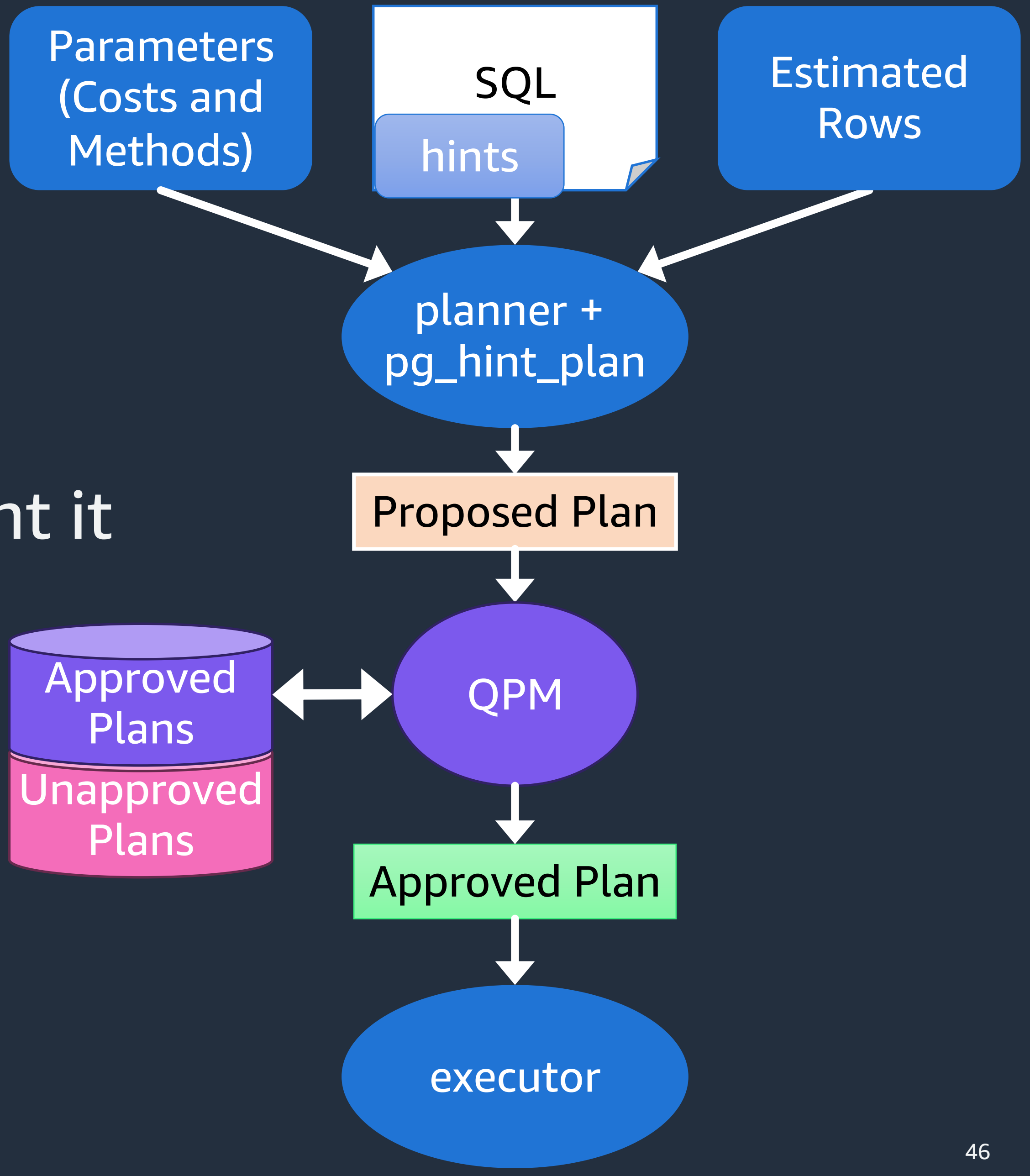


QPM – Reactive Mode

Manage one statement at a time

React when you see a “bad” plan

Mark that plan ‘Rejected’ – to prevent it from being used again



QPM – Proactive Mode

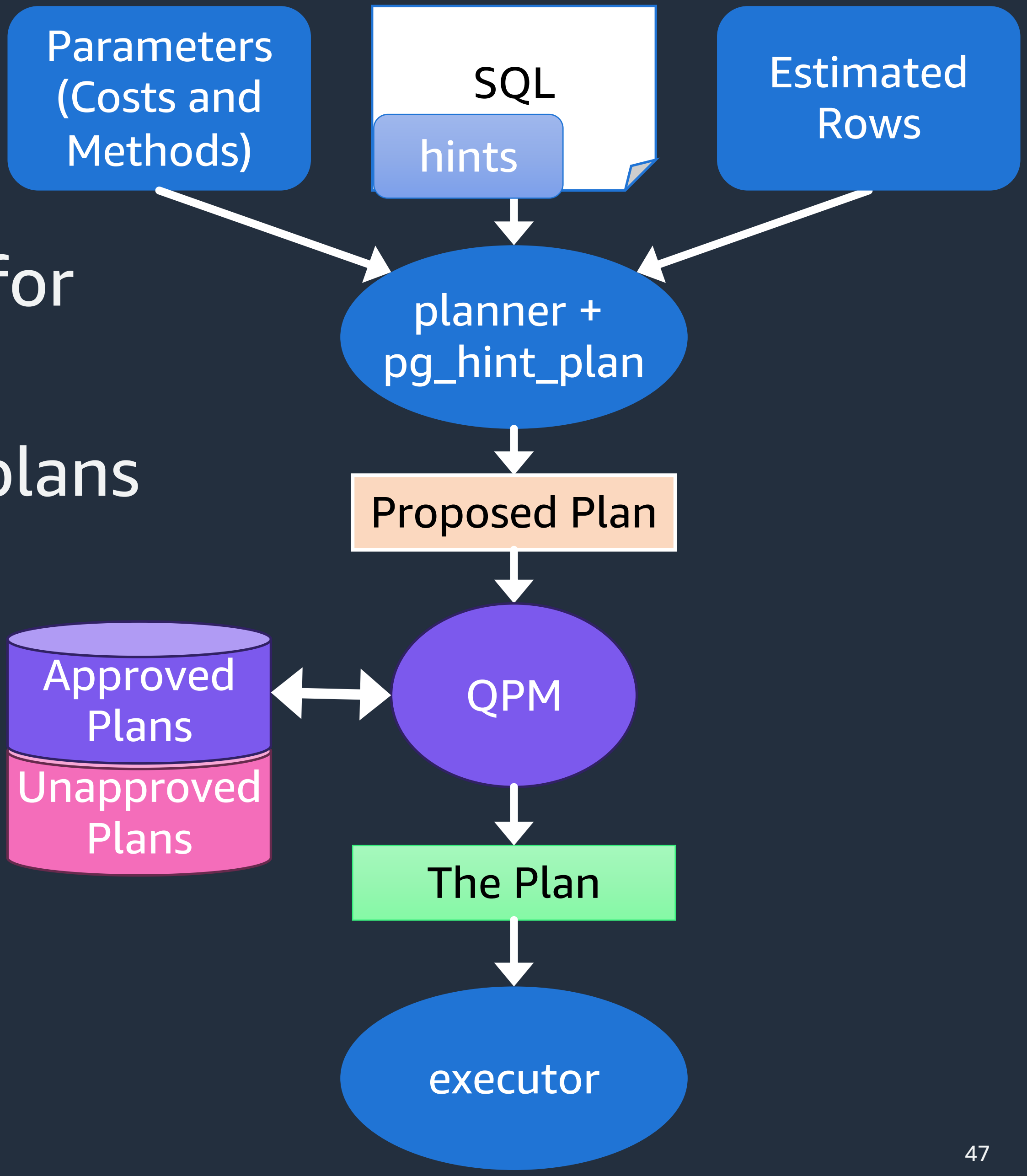
Capture and approve a set of plans for all statements (baseline)

Set QPM to Enforce only approved plans

Prevents plan flips

QPM saves new plans

Evolve baseline periodically to evaluate new plans



Setting up QPM

Set **rds.enable_plan_management** to “1” in cluster parameter group

```
=> create extension apg_plan_mgmt;
CREATE EXTENSION

-- (proactive) “automatic” to capture plans for statements executed 2+ times
=> set apg_plan_mgmt.capture_plan_baselines to automatic;
SET
-- (reactive) “manual” to capture individual statements and plans interactively
```

Capture your first plan

```
-- run your query at least 2 times
=> select * from boarding_pass_details where boarding_pass_id = 10;

=> select plan_hash, status, sql_text
from apg_plan_mgmt.dba_plans ;

-[ RECORD 1 ]-----+-----------------------------------------------------
------------
plan_hash         | -1757779097
status            | Approved
sql_text          | select * from boarding_pass_details where
boarding_pass_id = 10;
```

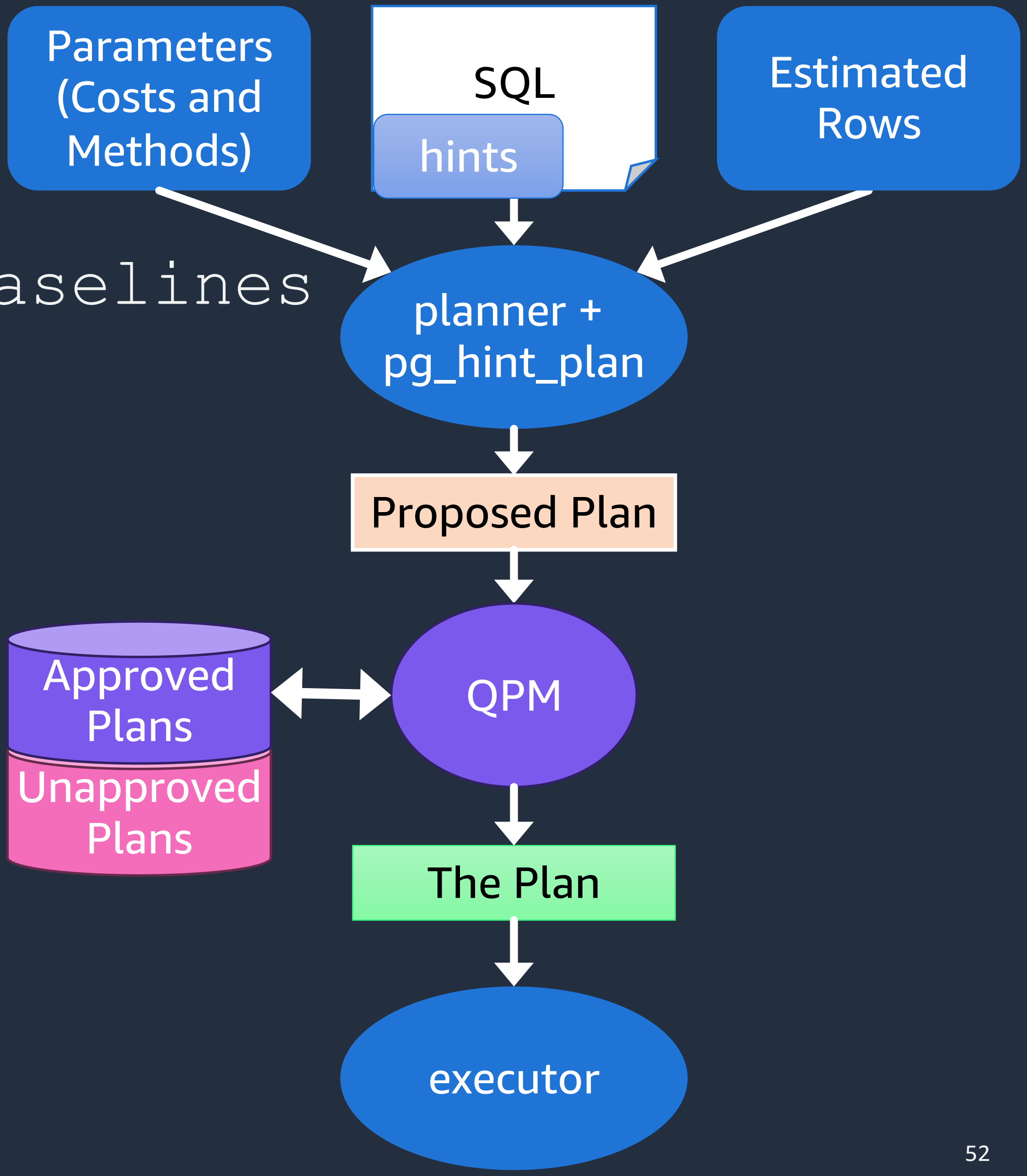
Plan Outlines

```
=> select plan_outline from apg_plan_mgmt.dba_plans
where plan_hash = -1757779097;

-[ RECORD 1 ]+--------------------------------------------
plan_outline | {                                          +
             |   "Fmt": "01.00",                          +
             |   "Outl": {                                +
             |     "Op": "IScan",                         +
             |     "QB": 1,                               +
             |     "S": "pgair",                   +
             |     "Idx": "boarding_pass_details_pass_id",+
             |     "Tbl": "boarding_pass_details",        +
             |     "Rid": 1                               +
             |   }                                        +
             | }
```

Enforcing plans

```
>set apg_plan_mgmt.use_plan_baselines
to ON;
SET
```

QPM will check Proposed Plan and switch to approved plan



Enforcing plans

```
=> /*+ SeqScan(boarding_pass_details) */ explain select * from
boarding_pass_details where boarding_pass_id = 10;

                        QUERY PLAN
------------------------------------------------------------
 Index Scan using boarding_pass_details_pass_id on boarding_pass_details
(cost=0.57..29.89 rows=704 width=80)
   Index Cond: (boarding_pass_id = 10)
 Note: An Approved plan was used instead of the minimum cost plan.
 SQL Hash: -1009835677, Plan Hash: -1757779097, Minimum Cost Plan Hash:
-1815128652
(4 rows)
```

Saved plans

```
=> select sql_hash, plan_hash, status, sql_text from
apg_plan_mgmt.dba_plans;
-[ RECORD 1 ]-------------------------------------------------
----
sql_hash  | -1009835677
plan_hash | -1757779097
status    | Approved
sql_text  | select * from boarding_pass_details where boarding_pass_id =
10;
-[ RECORD 2 ]-------------------------------------------------
----
sql_hash  | -1009835677
plan_hash | -1815128652
status    | Unapproved
sql_text  | select * from boarding_pass_details where boarding_pass_id =
10;
```

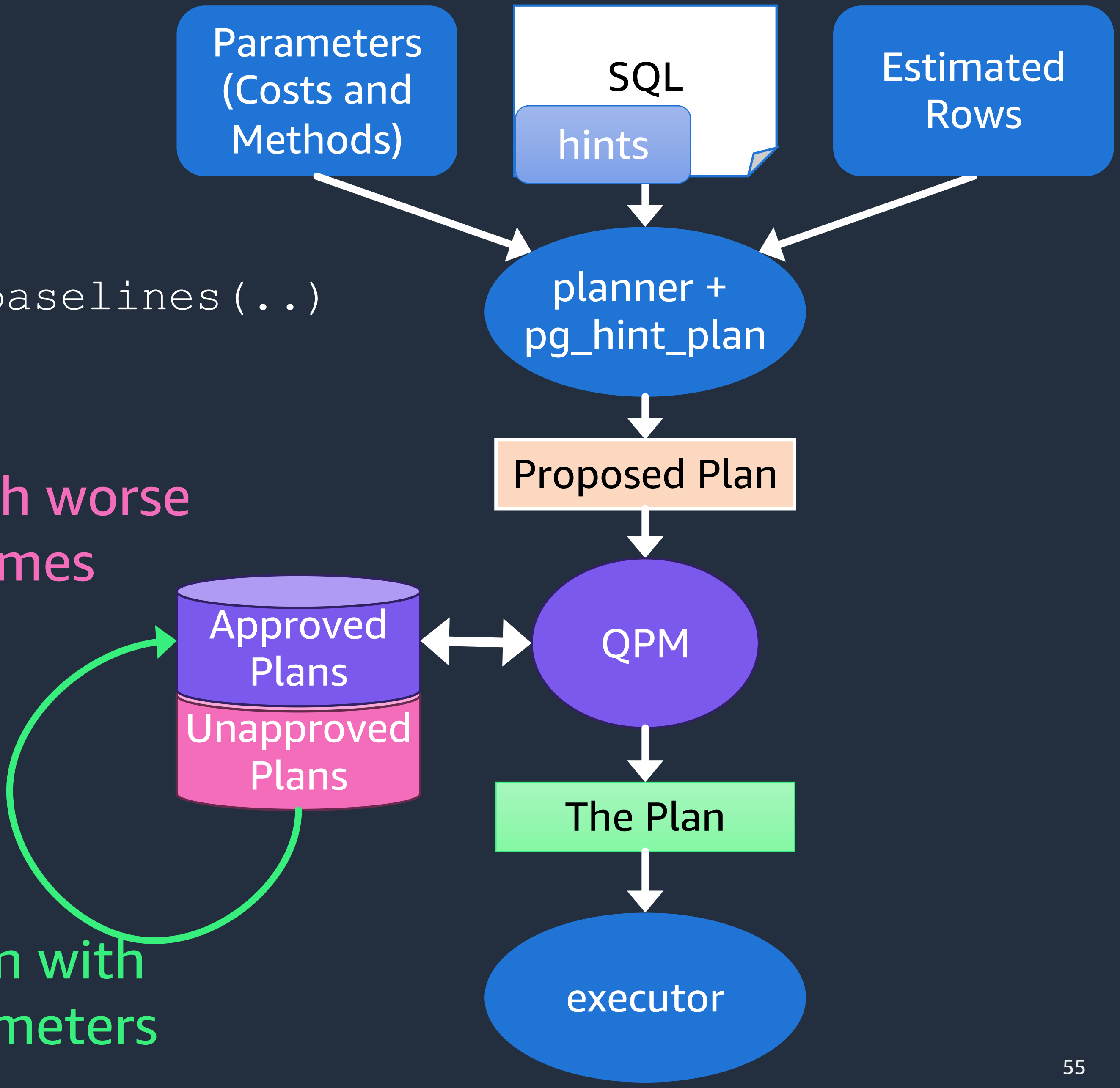
Evolving baselines

```
select apg_plan_mgmt.evolve_plan_baselines(..)
```

Reject plans with worse execution times

Approve plans with better execution times

Run every plan with captured parameters



Evolving plans

```
=> SELECT apg_plan_mgmt.evolve_plan_baselines (
   sql_hash,
   plan_hash,
   min_speedup_factor := 1.1,
   action := 'approve'
)
FROM apg_plan_mgmt.dba_plans WHERE status = 'Unapproved’;

NOTICE:  [Unapproved] SQL Hash: -1009835677, Plan Hash: -1815128652, select *
from boarding_pass_details where boarding_pass_id = 10;
NOTICE:      Baseline   [Planning time 0.047 ms, Execution time 0.035 ms]
NOTICE:      Baseline+1 [Planning time 0.245 ms, Execution time 55907.218 ms]
NOTICE:      Total time benefit: -55907.381 ms, Execution time benefit: -
55907.183 ms, Estimated rows=704, Actual rows=10, Cost = 1000.00..3338570.40
-[ RECORD 1 ]---------+--
evolve_plan_baselines | 1
```

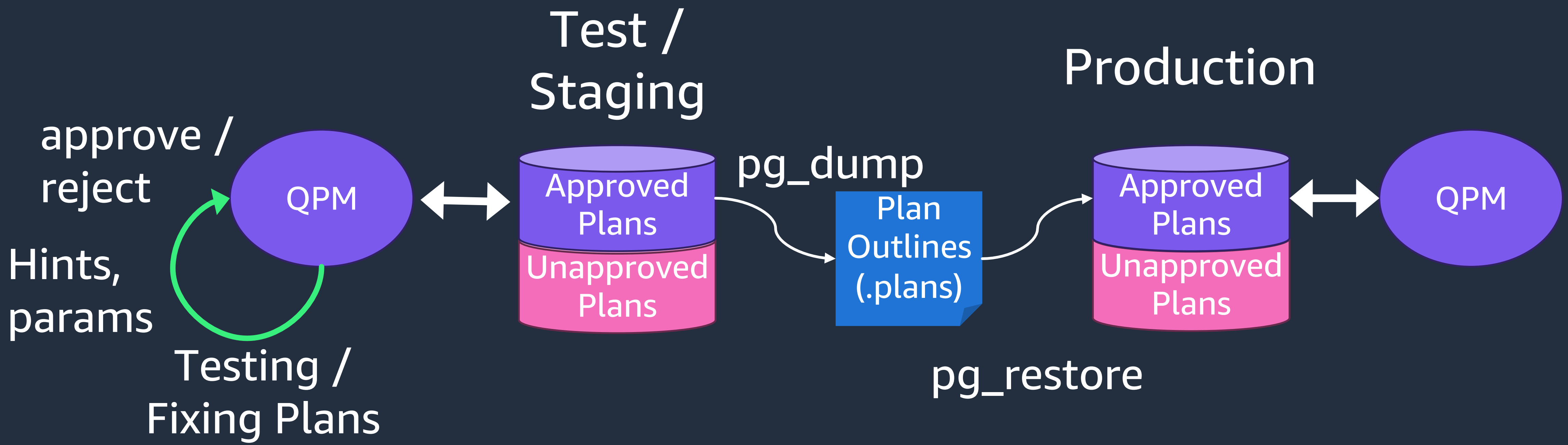
dba_plans after evolve baselines

```
=> select plan_hash, status, has_side_effects, planning_time_ms,
   execution_time_ms, cardinality_error, plan_created, last_used
from apg_plan_mgmt.dba_plans
where plan_hash = -1815128652;

-[ RECORD 1 ]-----+----------------------------
plan_hash         | -1815128652
status            | Unapproved
has_side_effects  | f
planning_time_ms  | 0.245
execution_time_ms | 55907.218
cardinality_error | 4.2541932631639967
plan_created      | 2023-08-30 12:06:51.540512
last_used         | 2023-08-30
```

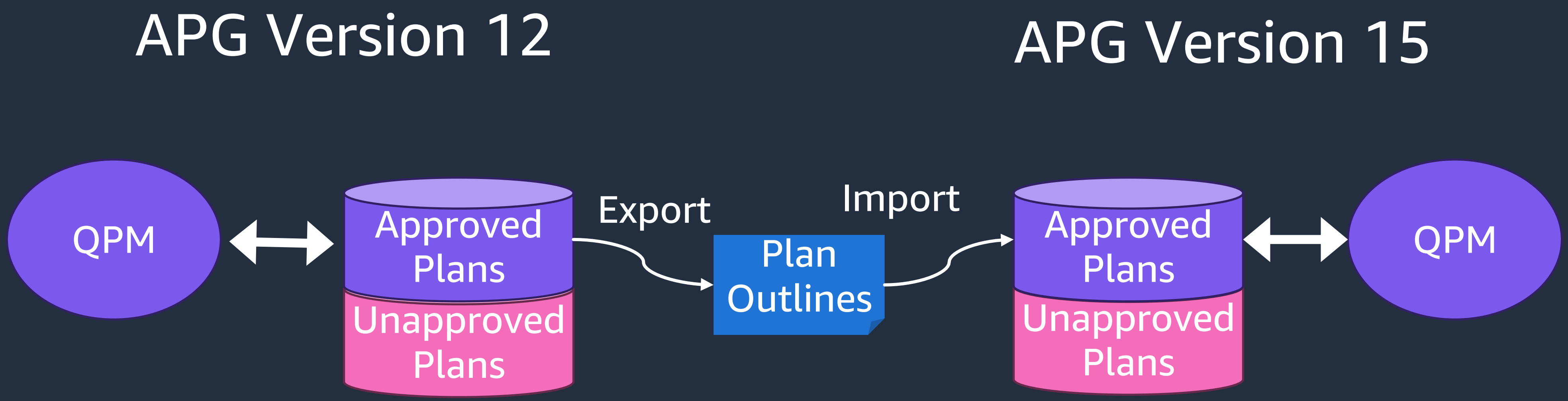
Reject specific plan

```
=> select apg_plan_mgmt.set_plan_status(-1009835677, -1815128652,
'rejected');
-[ RECORD 1 ]---+--
set_plan_status | 0

-[ RECORD 1 ]-----+---------------------------
plan_hash         | -1815128652
status            | Rejected
has_side_effects  | f
planning_time_ms  | 0.245
execution_time_ms | 55907.218
cardinality_error | 4.2541932631639967
plan_created      | 2023-08-30 12:06:51.540512
last_used         | 2023-08-30
```

Export/Import Plan Outlines



Major Version Upgrade



APG Version 12

APG Version 15

QPM

Approved Plans

Unapproved Plans

Export

Plan Outlines

Import

Approved Plans

Unapproved Plans

QPM

Fix bad plans – Help the planner do it’s job!

Fix your stats!

Tweak a couple of cost parameters

Use pg_hint_plan

Enforce plans with QPM

Questions?