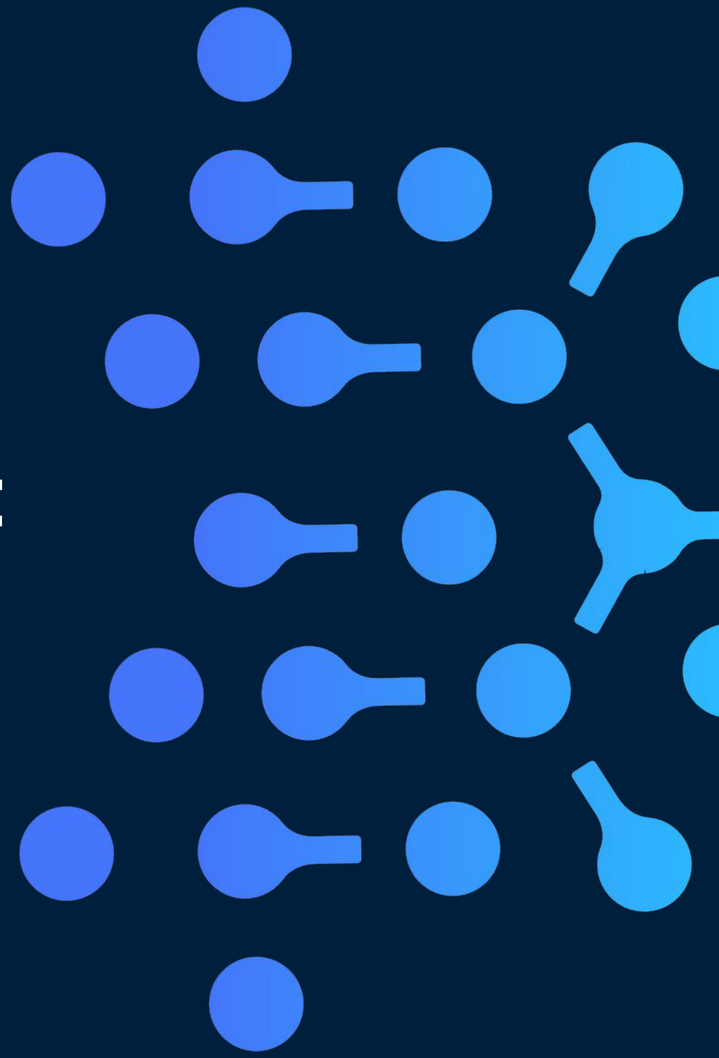
dbtune

Why your PostgreSQL tuning guide might be wrong (and what to do about it)

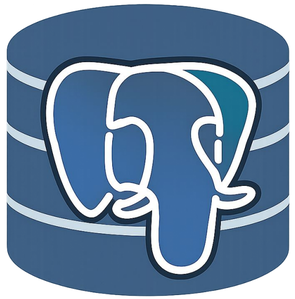
Mohsin Ejaz
Senior DevOps Engineer



About me

- Nearly two decades in DevOps, QA, and Release Management with PostgreSQL
- 17 years at EnterpriseDB
- Specializing in CI/CD, automation, and database benchmarking
- Currently: DevOps,Pre Sales and Customer success – focusing on reliability and performance
- Focus: Ensuring database performance isn't just a "best case" scenario, but a reliable, automated reality for global enterprises

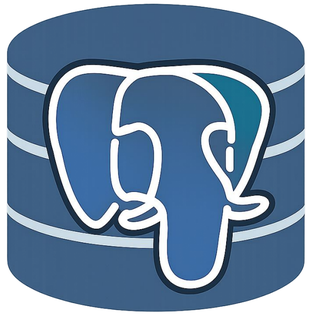
Same PostgreSQL version. Same workload.



PostgreSQL

~ 6ms

?

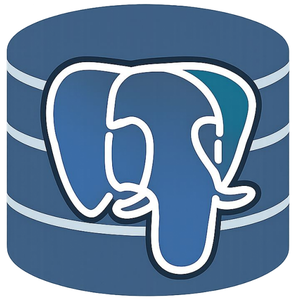


PostgreSQL

~ 0.06 ms

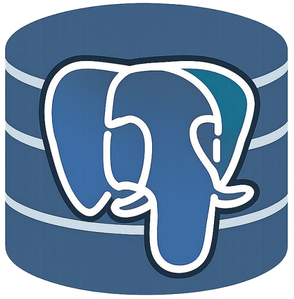
Same PostgreSQL.

Different defaults



Upstream PostgreSQL

JIT : ON



Managed PostgreSQL

JIT : OFF

What you will learn today

Three patterns

1. Different starting points
2. Infrastructure changes the math
3. Interactions & amplifications

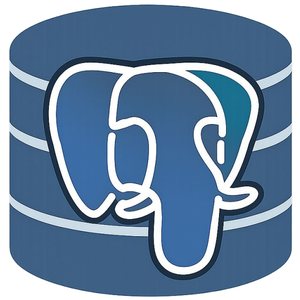
Practical takeaways

- How to test on YOUR System
- Infrastructure checklist
- Data-driven tuning workflow
- Tools you can use today

Pattern 1

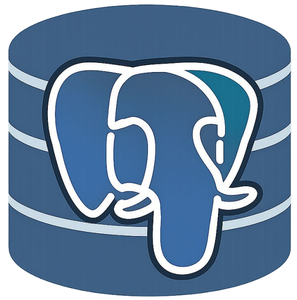
Different starting points

Example: JIT defaults in managed PostgreSQL



Upstream PostgreSQL

JIT : ON

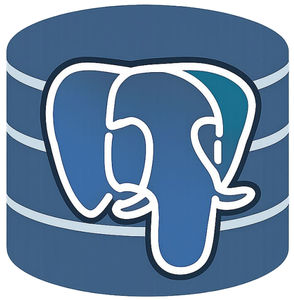


Managed PostgreSQL

(RDS)

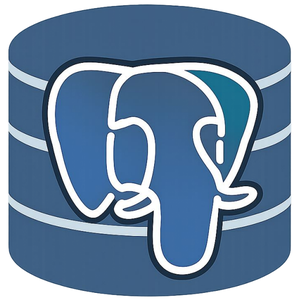
JIT : OFF

Example: Aurora specific behavior



Upstream PostgreSQL

Standard optimizer



Managed PostgreSQL

Custom optimizer

Aurora has a custom optimizer based on benchmarking & telemetry

Pattern 1 – What to do

Know your starting point

1. **SHOW ALL;**

   Know your real baseline

2. **Compare with upstream defaults**

   Not assumptions - actual defaults

3. **Read provider documentation**

   Defaults reflect intent

Pattern 2

Infrastructure changes the math

Storage changes I/O cost

Local storage

Predictable Latency

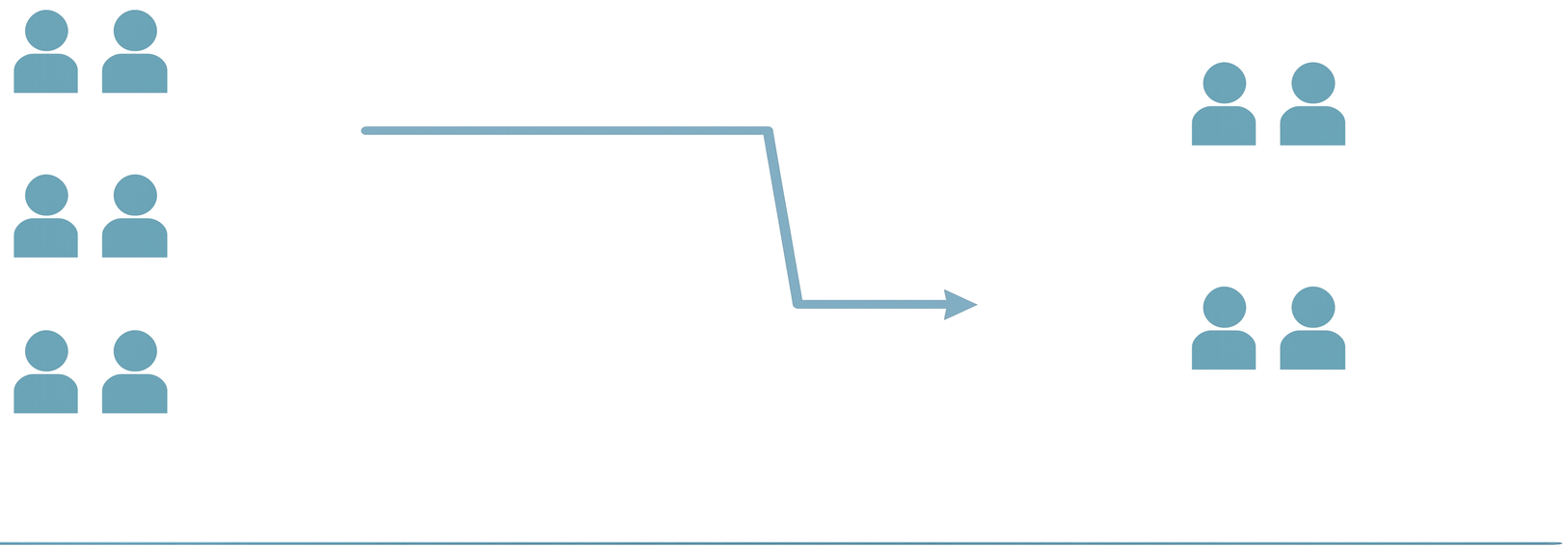
Network - attached storage

Added latency & variability

Storage variability breaks cost assumptions

- Same parameter
- Same PostgreSQL
- Different infrastructure
- Different math

Compute changes parallelism math

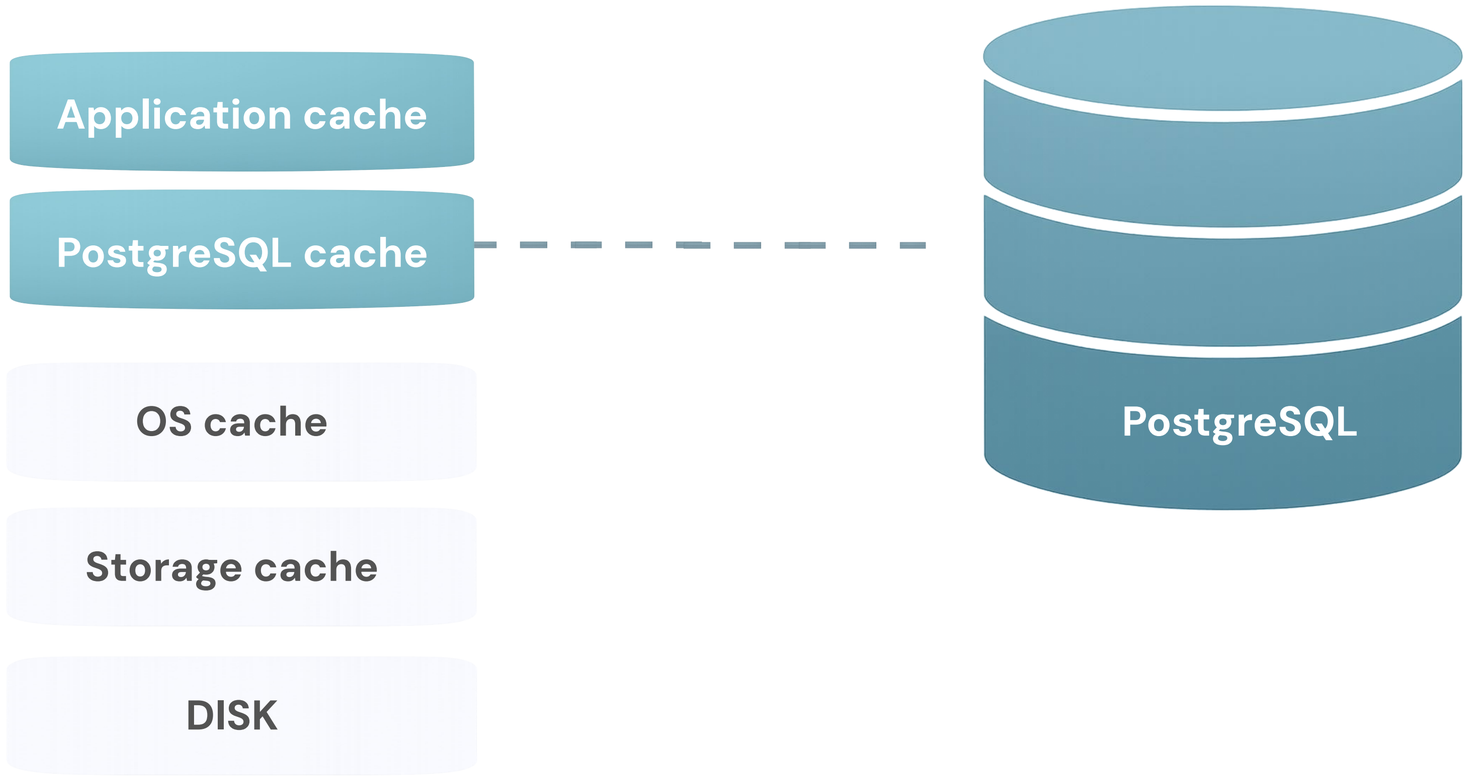


Compute - What to do

Tune for CPU stability

- Is CPU availability consistent or variable ?
- Observe saturation during real workload spikes
- Validate parallel plan under stress
- Tune for worst-case, not best-case conditions

Caching breaks planner assumptions



Caching - What to do

Tune for what PostgreSQL actually sees

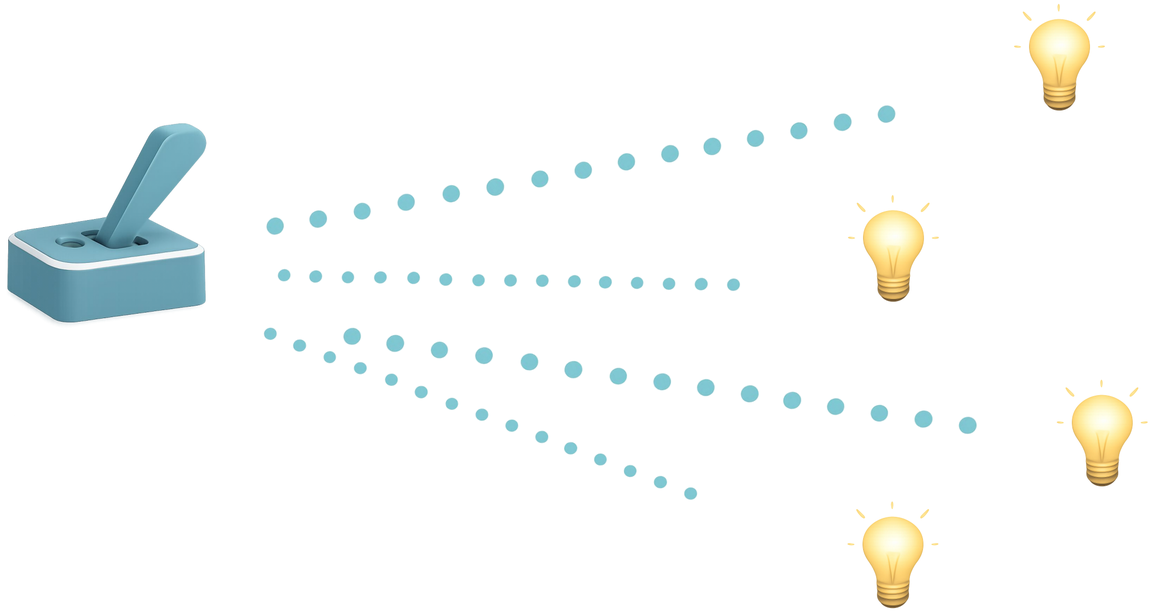
- Don’t tune blindly for read latency if reads are absorbed elsewhere.
- Focus tuning on what actually hits PostgreSQL: writes, cache misses, maintenance operations.
- Validate assumptions with metrics, not intuition.

Pattern 2 summary

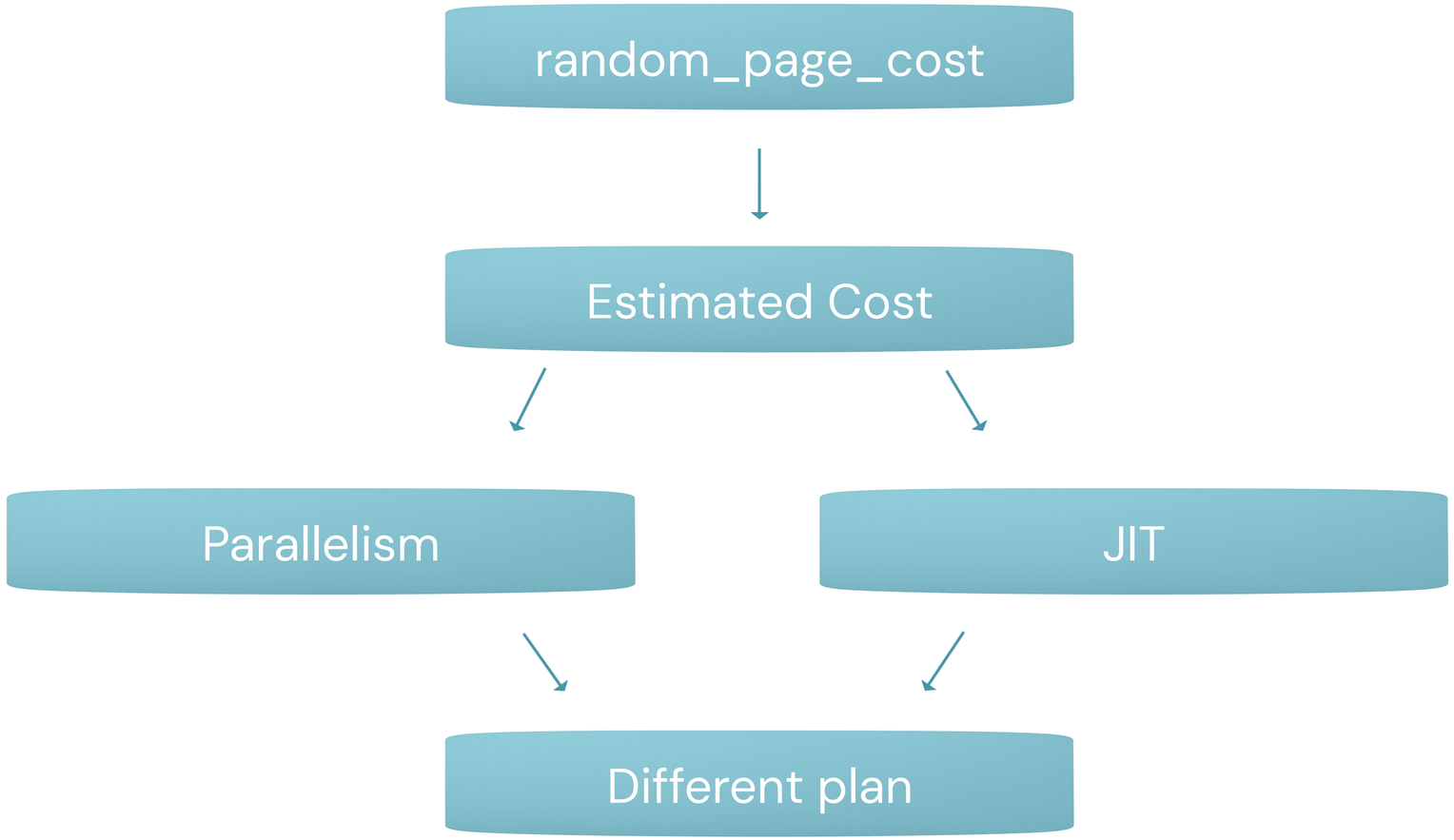
- Storage change I/O cost
- Compute changes CPI cost
- Caching hides reality

Pattern 3

Interactions and amplifications



Interactions: The cascade effect



Handling cascades safely

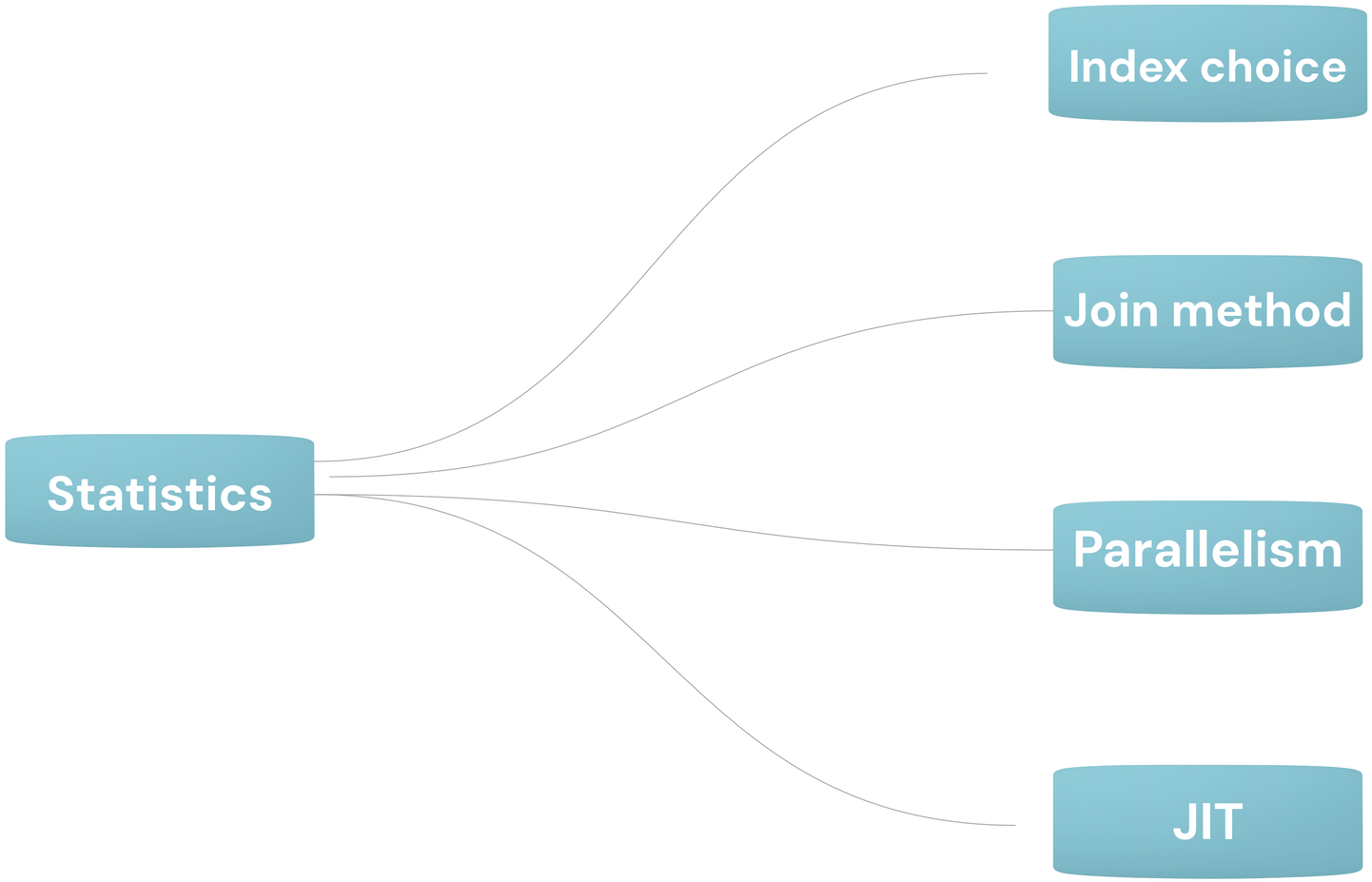
Don’t only ask: “Is this parameter good or bad?”

Ask: “What else does this unlock?”

Before - After → Compare

- Plan Shape
- Parallelism
- JIT

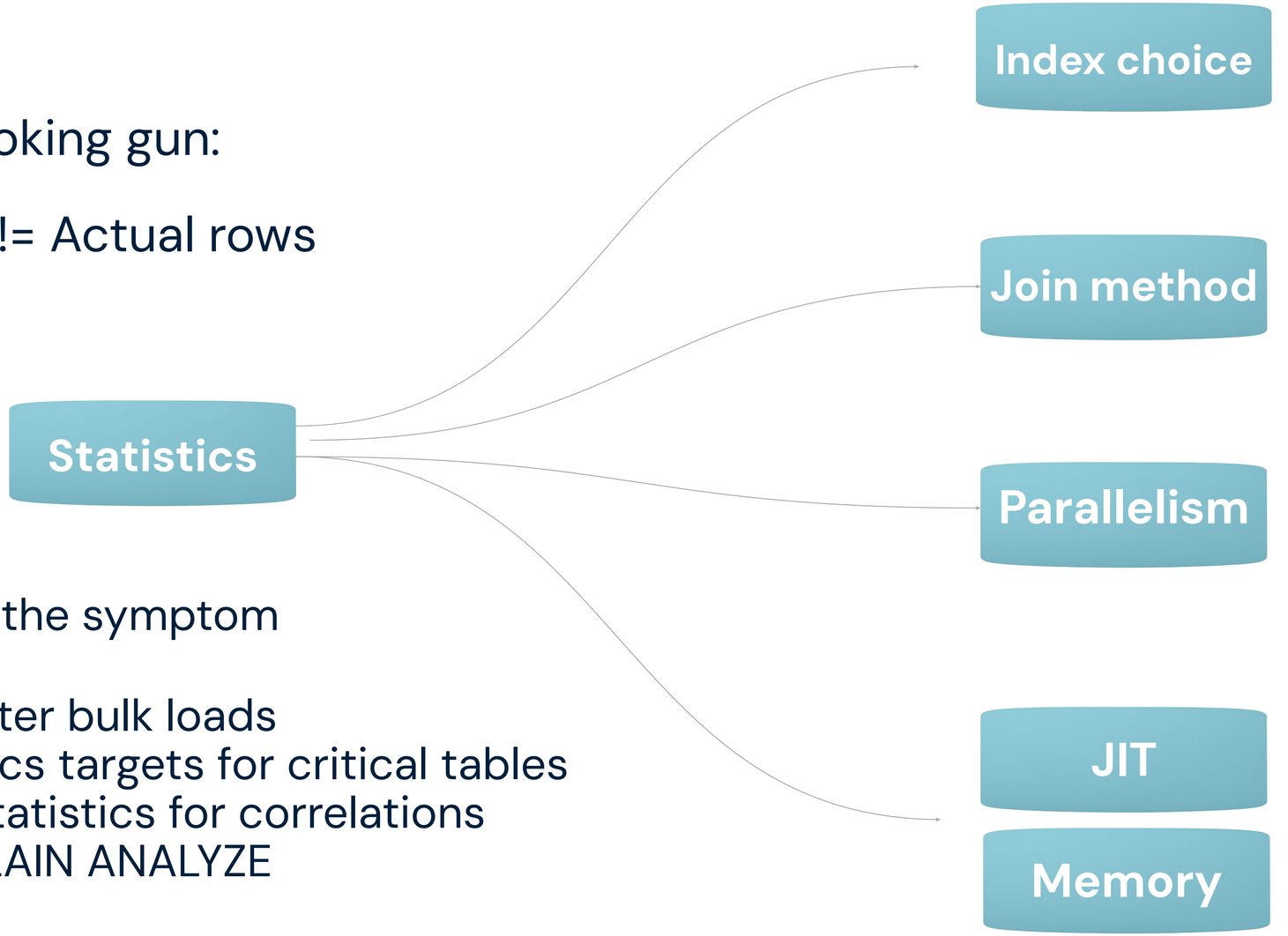
Statistics amplify everything



Handling statistical amplification

Look for the smoking gun:

Estimated rows != Actual rows



Fix the cause, not the symptom

- Run ANALYZE after bulk loads
- Increase statistics targets for critical tables
- Use extended statistics for correlations
- Verify with EXPLAIN ANALYZE

Pattern 3 – Emergent behaviour

Small changes

Large systems

Delayed effects

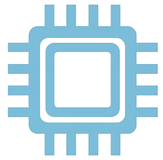
That's why tuning sometimes looks random.

It isn’t.

How to test on YOUR system-CHECKLIST

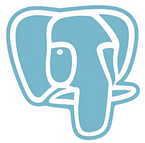
Storage

- Local or network - attached ?
- Stable or variable under load?



Compute

- Dedicated or burstable CPU?
- CPU limits (VMs, containers)



PostgreSQL

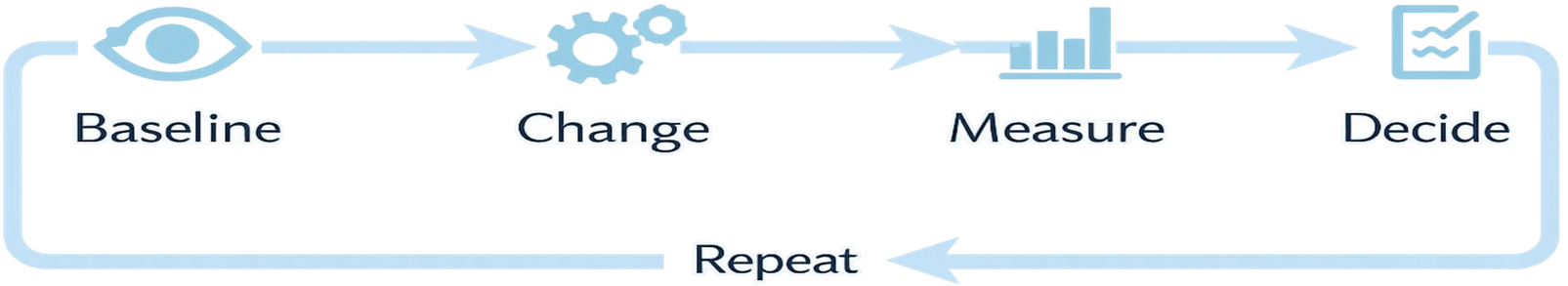
- Self hosted or managed?
- SHOW ALL; vs upstream defaults
- Provider docs = changed assumptions

Workload

- OLTP or analytics?
- Read heavy or write heavy?
- Caching above PostgreSQL?

Step 2 — Measure on YOUR system

Treat tuning like science, not folklore



What to measure

Workload

- pg_stat_statements
- Top queries by total time
- Focus on top 5 (80/20 rule)

Queries

- EXPLAIN ANALYZE
- Estimated vs Actual rows
- Plan changes,parallelism,JIT

Infrastructure

- IOPS,CPU saturation
- Credit exhaustion
- PostgreSQL doesn’t see these– you must

Before —› Change —› After —› Decide

Tools (to support good thinking)

Query analysis

- pg_stat_statements
- EXPLAIN ANALYZE
- explain.dalibo.com
- pg_stat_plans

Infrastructure metrics

- CloudWatch / Azure monitor / Cloud monitoring
- Prometheus + Grafana
- pganalyze

Starting point

- PGTune
- SHOW ALL;
- Provider docs

Automation

- DBtune

Tools don’t replace thinking. They reduce blind spots.

Why the puzzle wasn’t a bug

Same PostgreSQL

Same workload

Very different performance

Three patterns

- ✓ Different starting points
- ✓ Infrastructure changes the math
- ✓ Interaction & Amplification

What good PostgreSQL tuning looks like

Not memorizing parameters

But building a feedback loop

Checklist

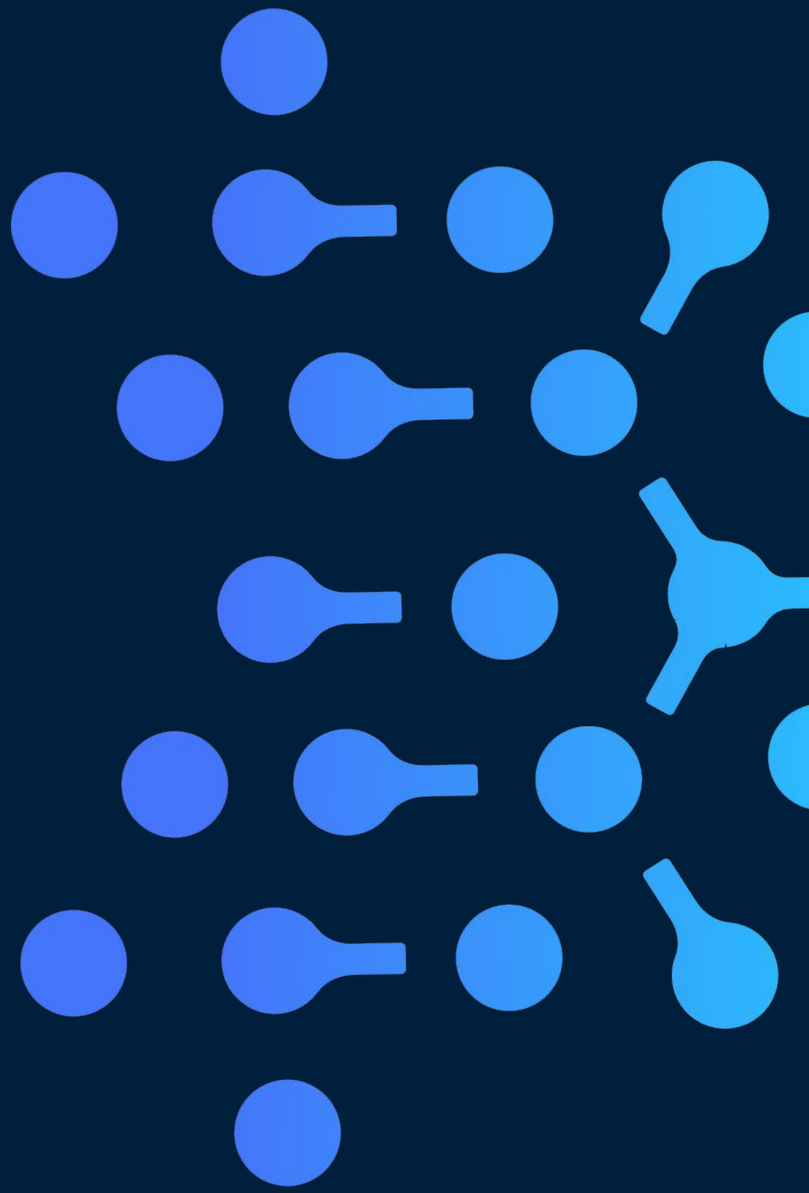
- ✓ Understand your system
- ✓ Form a hypothesis
- ✓ Change one thing
- ✓ Measure after
- ✓ Decide with data

Tuning guides are written for a system.

You are running a different one. Prove it on YOUR system

dbtune

Q&A



Thank you

Mohsin Ejaz

mohsin@dbtune.com