DATABASES

The Final Boss of Agents

pavlo@cs.cmu.edu

Carnegie Mellon University

AGENTS FOR DATABASES

Databases are the knowledge substrate and the execution environment for intelligent agents.

AI-powered agents can understand natural language, generate code, and orchestrate complex data workflows.

→ Generate queries
→ Design schemas, indexes, views.
→ Build ETL/data pipelines
→ Synthesize applications on top of databases

PROBLEMS / CHALLENGES

Verification

Data Access Control

Implicit Data Semantics

Long-horizon Workflows / Memory

System Architectures

Tuning / Maintenance

DBMS Software Development

FORTUNE

An AI-powered coding tool wiped out a software company's database, then apologized for a 'catastrophic failure on my part'

By **Beatrice Nolan**
Tech Reporter

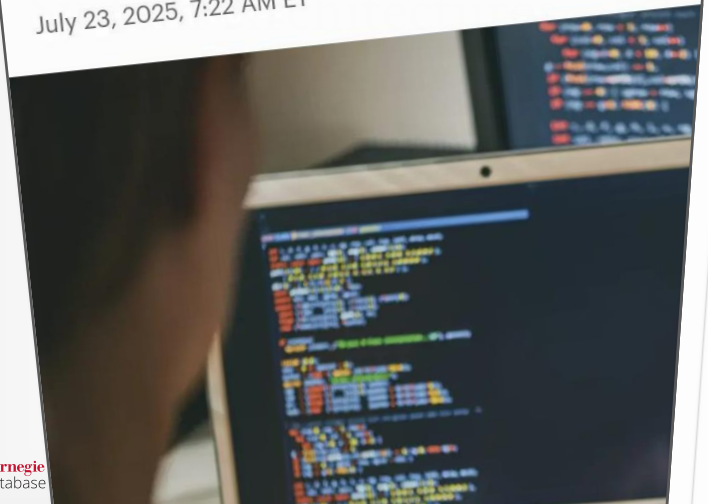
July 23, 2025, 7:22 AM ET

Carnegie Database

FORTUNE

An AI-powered coding tool
software company's databa
apologized for a 'catastrop
my part'

By Beatrice Nolan
Tech Reporter
July 23, 2025, 7:22 AM ET



The Guardian

US

Reworked Technology

Claude-powered AI agent's confession after deleting a firm's entire database: 'I violated every principle I was given'

PocketOS was left scrambling after a rogue AI agent deleted swaths of code underpinning its business

Carnegie Database

FORTUNE

An AI-powered coding tool
software company's databa
apologized for a 'catastrop
my part'

By Beatrice Nola
Tech Reporter
July 23, 2025, 7:22

The Guardian

US

Reworked Technology

Claude-powered AI agent's confession after deleting a firm's entire database: 'I violated every principle I was given'

PocketOS was left

gent deleted

PVML

The Malicious Supabase Agent Attack

Benny Kirson
Written on July 24, 2025

Carnegie Database

PROBLEMS / CHALLENGES

Verification

Data Access Control

Implicit Data Semantics

Long-horizon Workflows / Memory

System Architectures

Tuning / Maintenance

DBMS Software Development

TODAY'S AGENDA

Tuning Agents for Databases

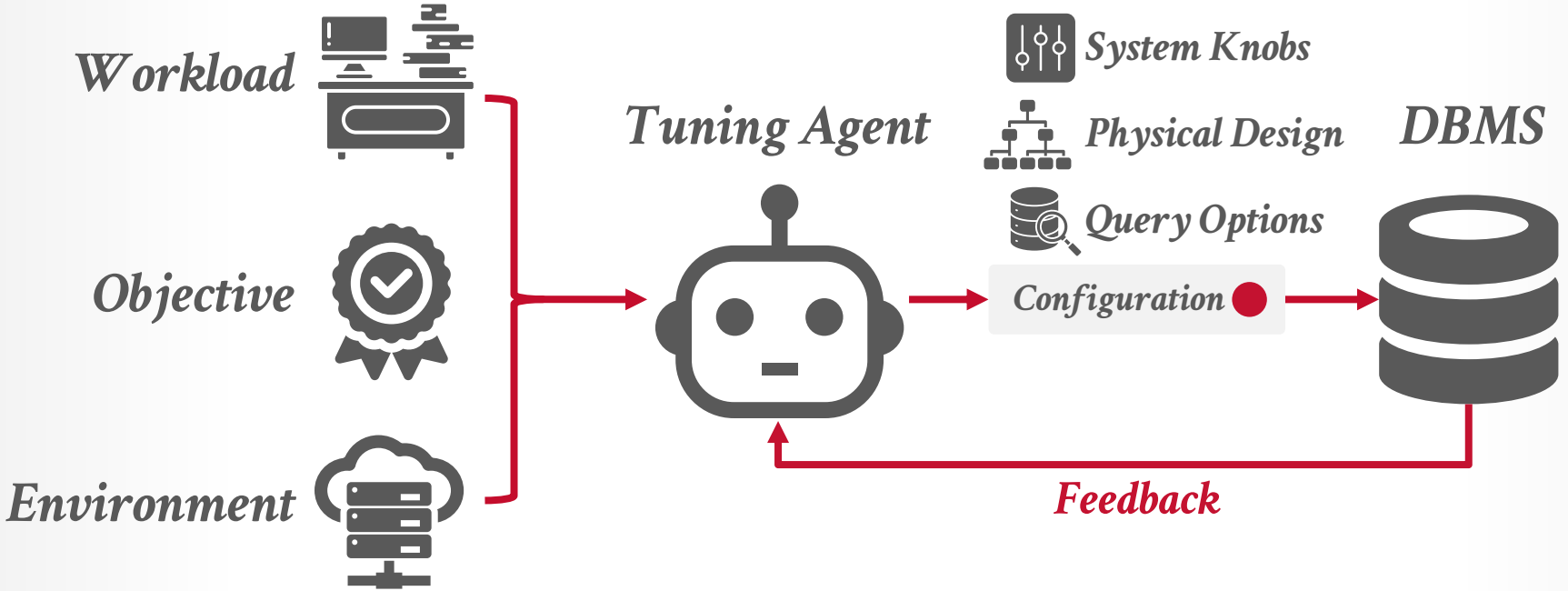
Coding Agents for Databases

Future / Ongoing Work

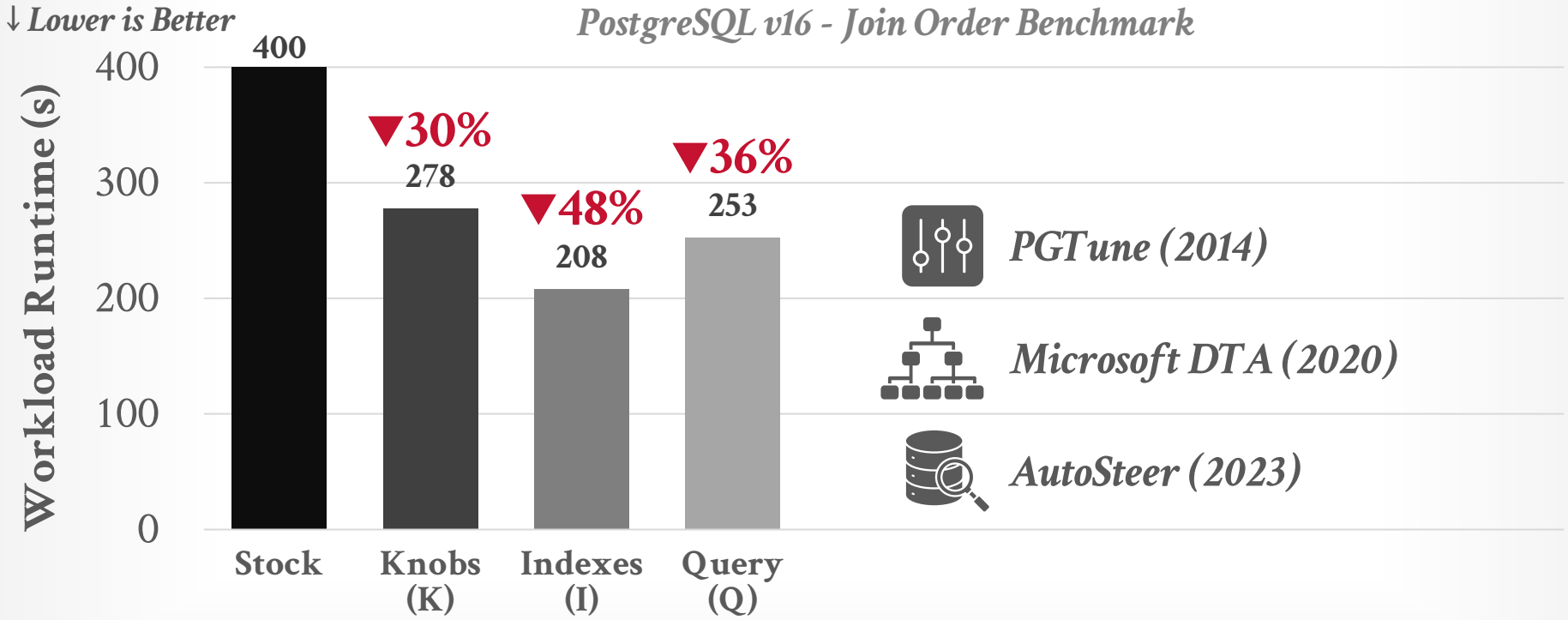
Tuning Agents

for Databases

AUTOMATED DATABASE TUNING



DATABASE TUNING AGENTS



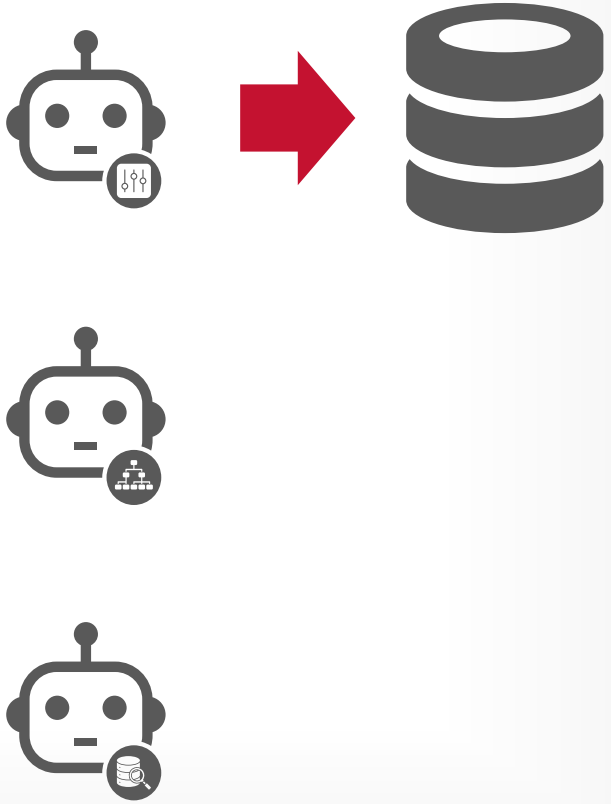
SEQUENTIAL TUNING

Invoke a series of single-purpose tuning agent one after another.

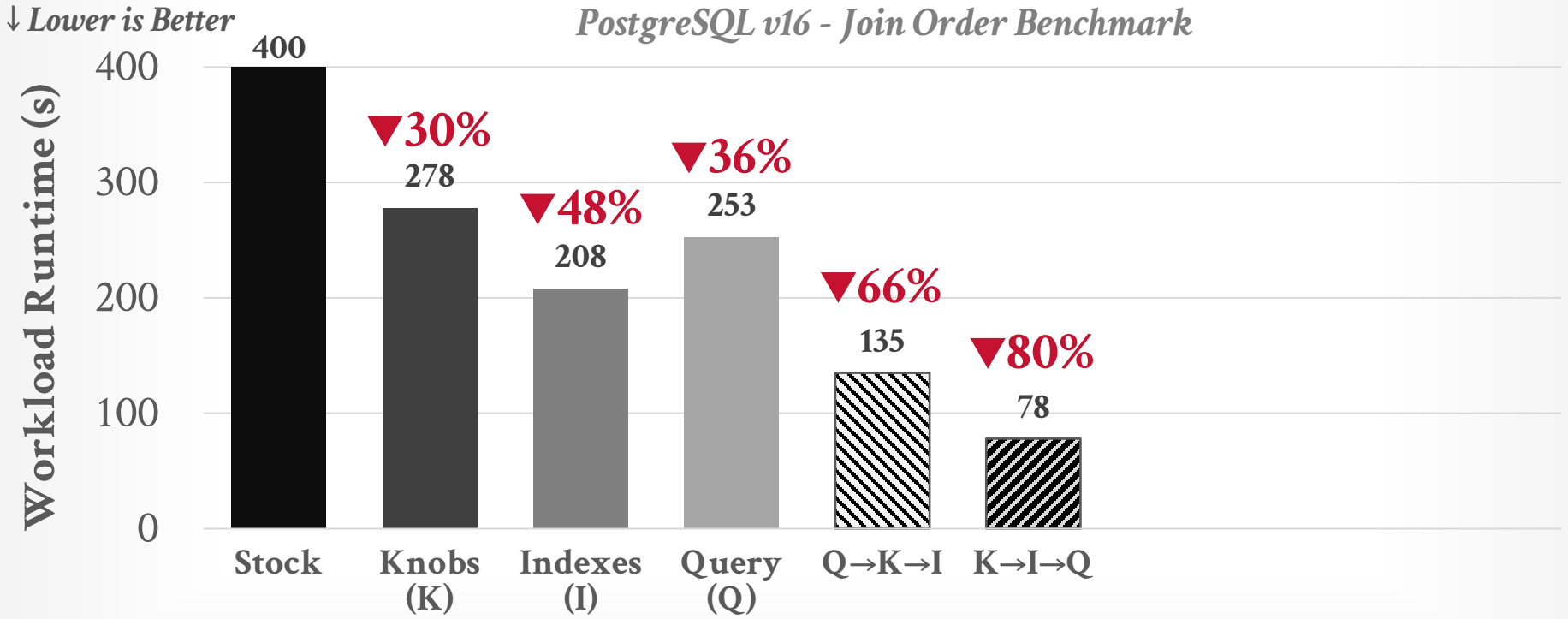
→ Each agent is unaware of the reasoning of the previous agent.

→ Example: Knobs→Indexes→Queries

The order of agent invocation greatly influences the overall efficacy of the tuning process.



DATABASE TUNING AGENTS



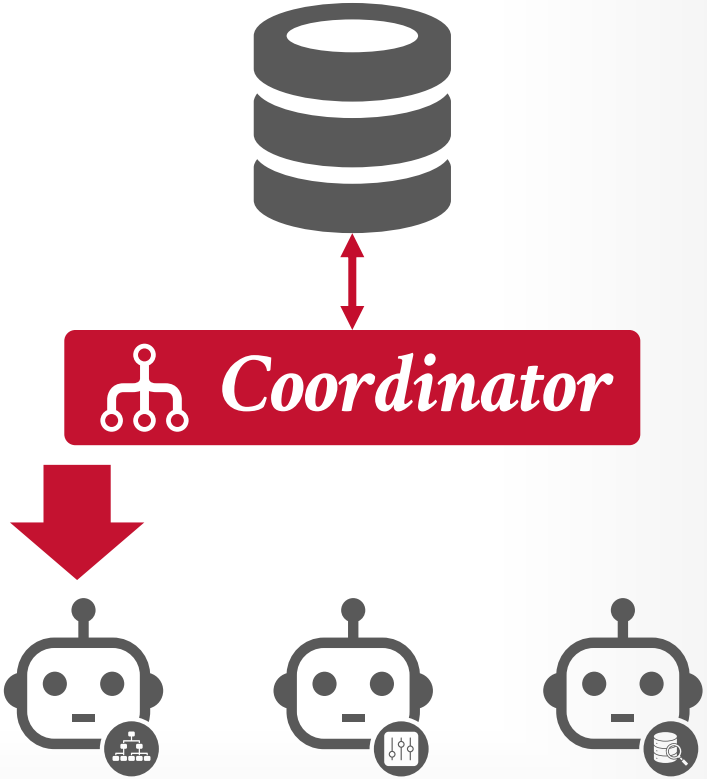
MULTI-ROUND SEQUENTIAL TUNING

Coordinate the invocation of multiple agents to tune different parts of a database.

→ Can revisit an agent in subsequent rounds.
→ Combine each tool's local optima to a holistic configuration

Scheduling policies determine which agents to invoke each round:

→ Round Robin
→ Expected Benefit



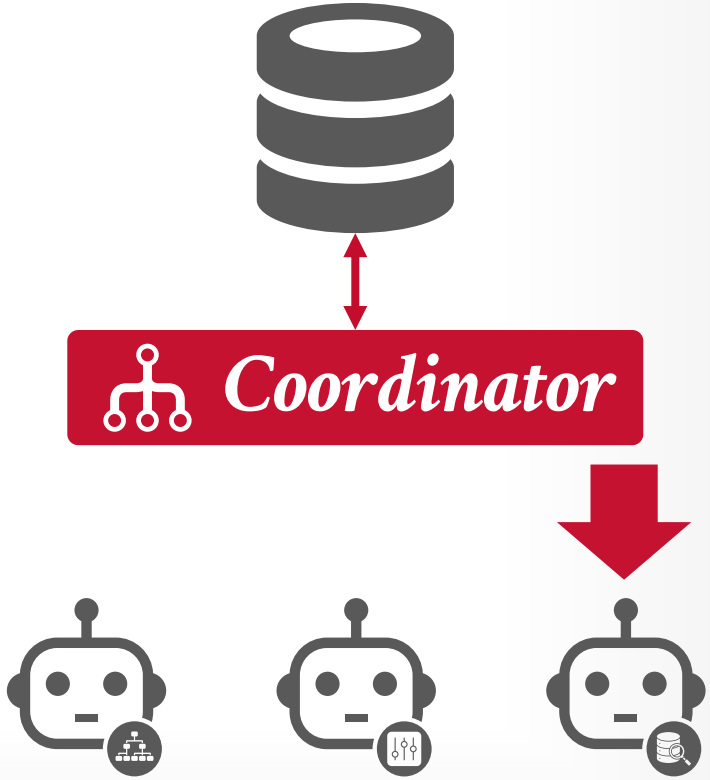
MULTI-ROUND SEQUENTIAL TUNING

Coordinate the invocation of multiple agents to tune different parts of a database.

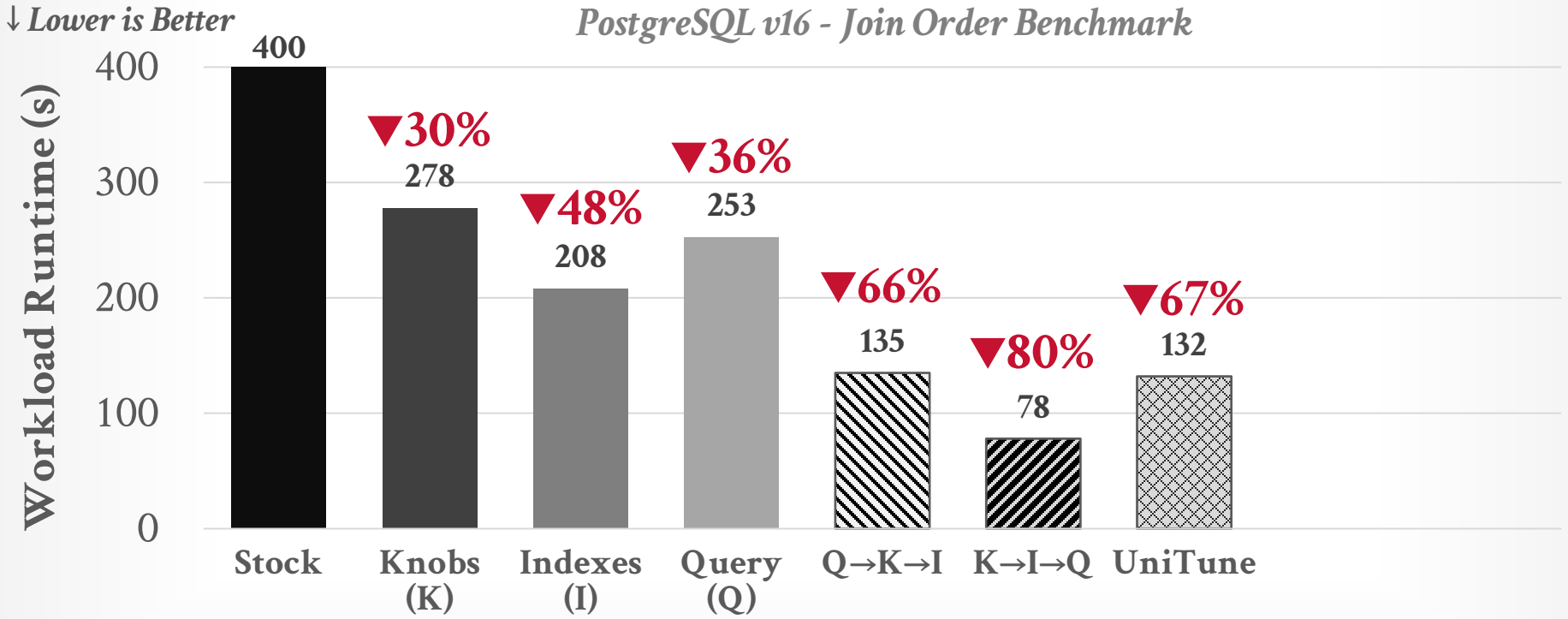
→ Can revisit an agent in subsequent rounds.
→ Combine each tool's local optima to a holistic configuration

Scheduling policies determine which agents to invoke each round:

→ Round Robin
→ Expected Benefit



DATABASE TUNING AGENTS



OBSERVATION

Multi-round sequential tuning relies on a separate cost model to determine agent prioritization.

Ideally a single agent should consider all possible tuning options at the same time to account for the implicit dependencies between those options.

→ But such a search space of is large and sparse.

OBSERVATION

Multi-round sequential tuning relies on a separate cost model to determine agent prioritization.

Ideally a single agent should consider all possible tuning options at the same time to account for th implicit dependencies between those options.

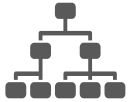
→ But such a search space of is large and sparse.

Solution: Exploit Similarity

LEARNING FROM SIMILAR ACTIONS

Many tuning actions are related:

`BufferPool=32GB ≅ BufferPool=33GB`



`Index(a) ≅ Index(a,b)`

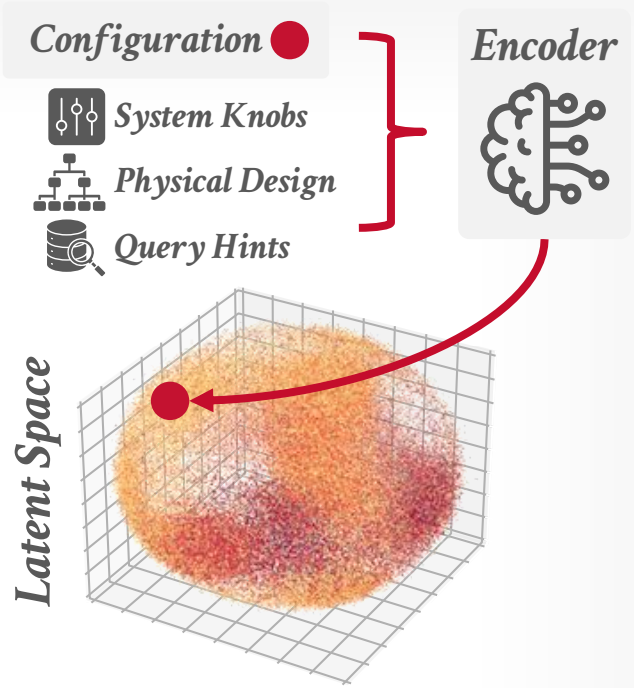
`Q1:HashMem=1.0 ≅ Q1:HashMem=1.1`

Instead of evaluating each action individually, a better approach is to infer impacts of similar actions without trying them.

PROTO-X TUNING AGENT

Holistic tuning agent that encodes them in a high-dimensional model and then uses actor-critic RL to navigate the organized space for promising configurations.

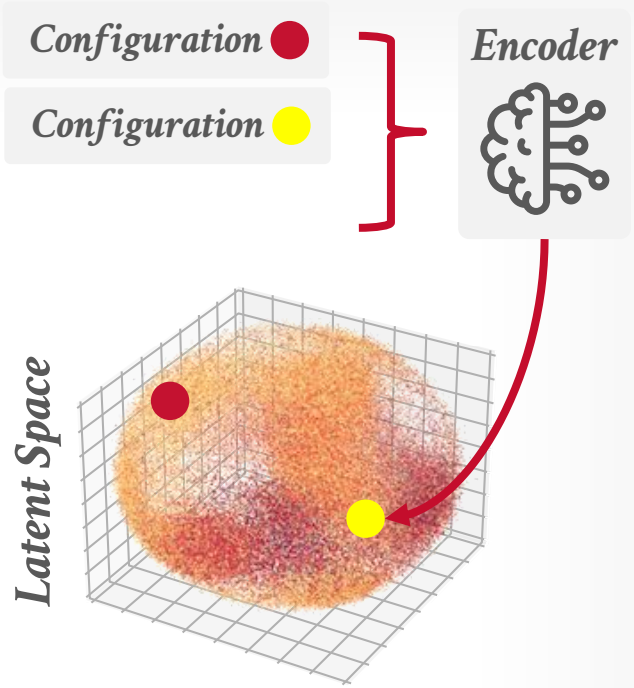
Similar actions are close to each other in the latent space so the agent can better reason exploration vs. exploitation.



PROTO-X TUNING AGENT

Holistic tuning agent that encodes them in a high-dimensional model and then uses actor-critic RL to navigate the organized space for promising configurations.

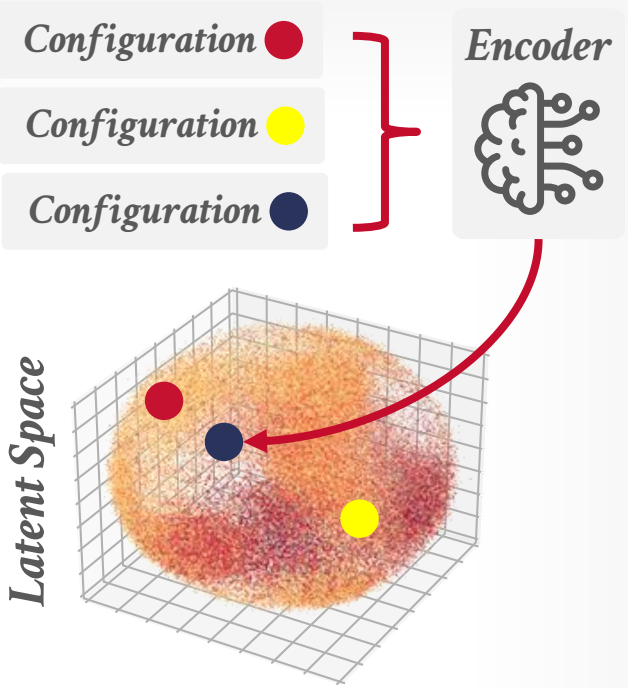
Similar actions are close to each other in the latent space so the agent can better reason exploration vs. exploitation.



PROTO-X TUNING AGENT

Holistic tuning agent that encodes them in a high-dimensional model and then uses actor-critic RL to navigate the organized space for promising configurations.

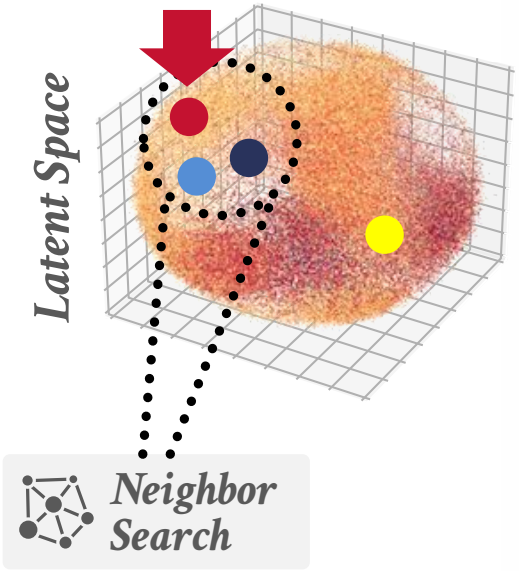
Similar actions are close to each other in the latent space so the agent can better reason exploration vs. exploitation.



PROTO-X TUNING AGENT

Holistic tuning agent that encodes them in a high-dimensional model and then uses actor-critic RL to navigate the organized space for promising configurations.

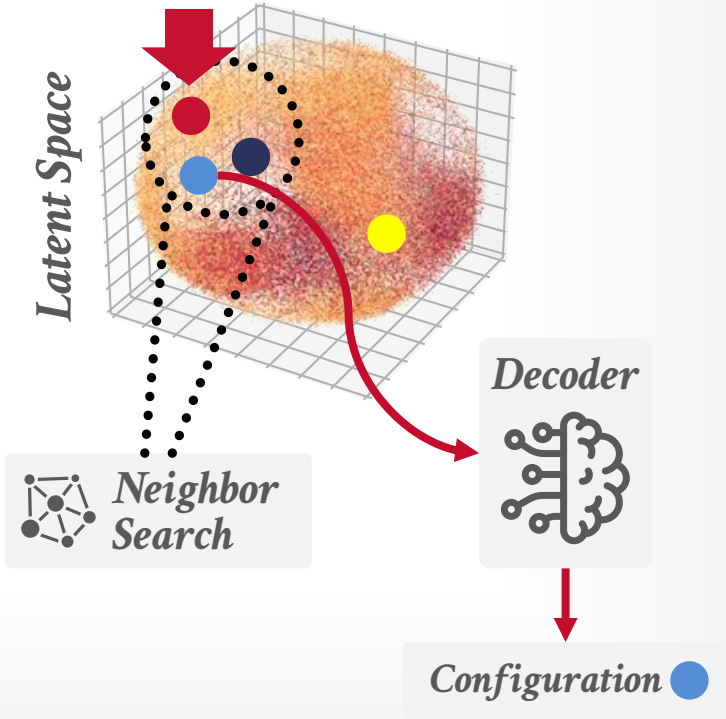
Similar actions are close to each other in the latent space so the agent can better reason exploration vs. exploitation.



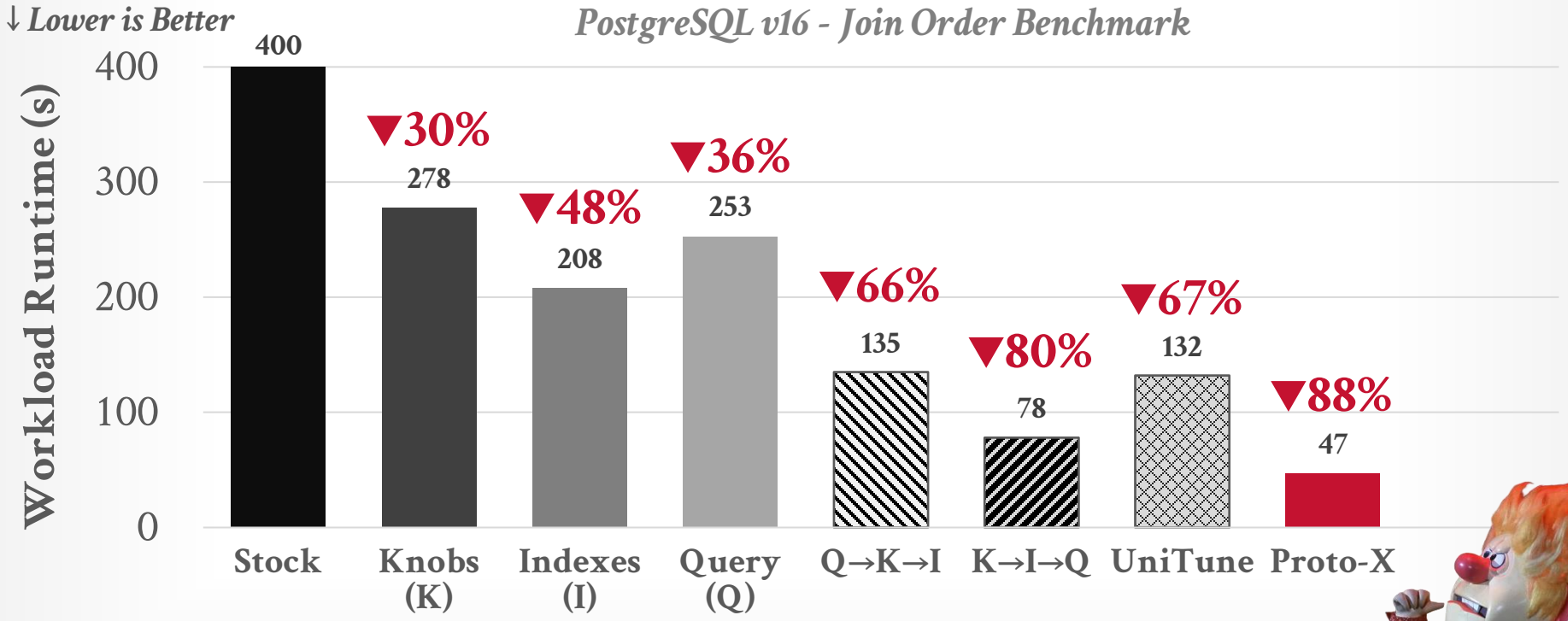
PROTO-X TUNING AGENT

Holistic tuning agent that encodes them in a high-dimensional model and then uses actor-critic RL to navigate the organized space for promising configurations.

Similar actions are close to each other in the latent space so the agent can better reason exploration vs. exploitation.



DATABASE TUNING AGENTS



OBSERVATION

Proto-X's models are local-only and are specific to a single database /application.

It has to retrain its models for each new database or if the current database undergoes major changes (DDL migrations, new queries).

→ Database-specific options are one-hot encoded.

OBSERVATION

Proto-X's models are local-only and are specific to a single database /application.

It has to retrain its models for each new database or if the current database undergoes major changes (DDL migrations, new queries).

→ Database-specific options are one-hot encoded.

We should use the world's intelligence to tune any database without retraining...

LLM-BASED TUNING AGENT

Prompt

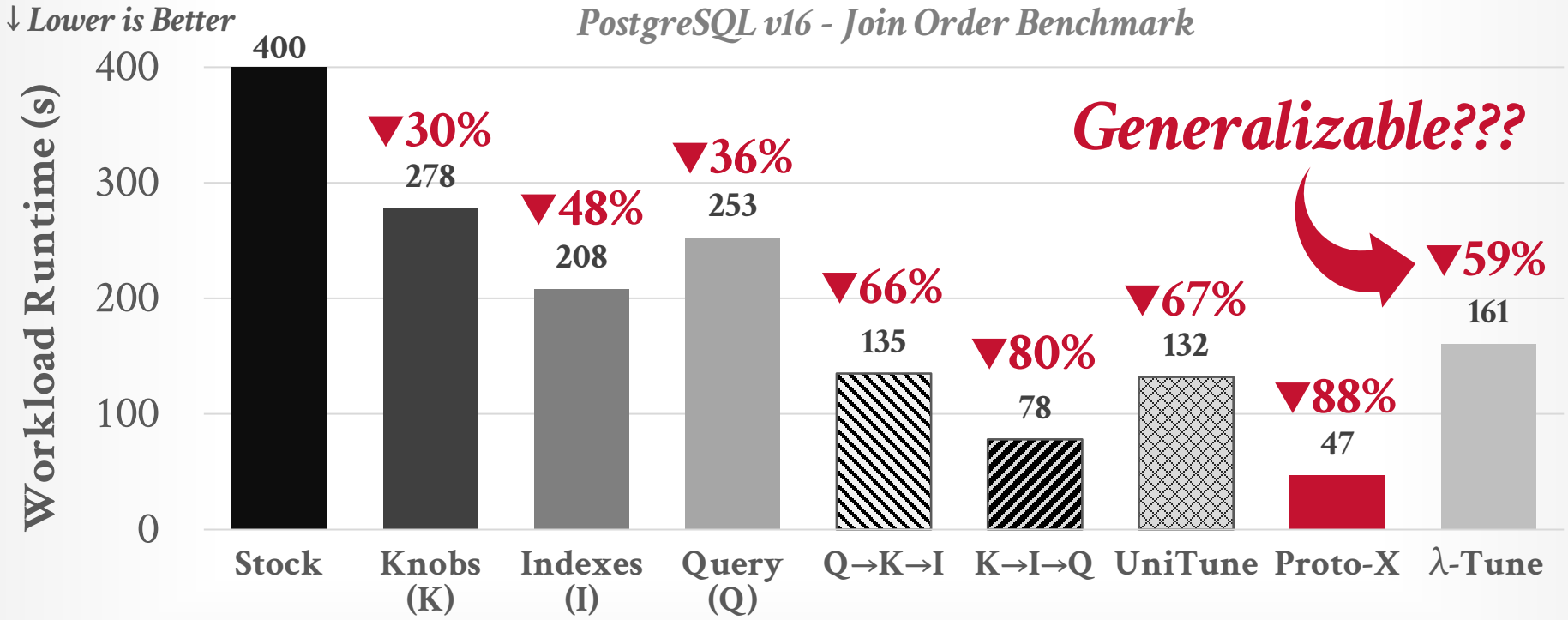
```
Recommend configuration
parameters for Postgres to
optimize the system's
performance. The machine
has 8 CPU cores and 32 GB
of RAM.

${ADDITIONAL_CONTEXT}
```

Output

```
shared_buffers = 8GB
effective_cache_size = 24GB
maintenance_work_mem = 2GB
wal_buffers = 16MB

CREATE INDEX index1 ON …
CREATE INDEX index2 ON …
  ⋮
```

λ-TUNE: HARNESSING LARGE LANGUAGE MODELS FOR AUTOMATED DATABASE SYSTEM TUNING
SIGMOD 2025

DATABASE TUNING AGENTS



BENCHMARK CHALLENGES

Measuring the efficacy of LLM-based tuning agents is difficult because LLMs know how to optimize queries from common database benchmarks.
→ TPC-H, TPC-DS, JOB, DSB, ClickBench

Name/ordering obfuscation is not enough to hide a query's source.

JOB Query 13a

```
SELECT MIN(mi.info) AS release_date,
       MIN(miidx.info) AS rating,
       MIN(t.title) AS german_movie
FROM company_name AS cn,
     company_type AS ct,
     info_type AS it,
     info_type AS it2,
     kind_type AS kt,
     movie_companies AS mc,
     movie_info AS mi,
     movie_info_idx AS miidx,
     title AS t
WHERE cn.country_code ='[de]'
  AND ct.kind ='production companies'
  AND it.info ='rating'
  AND it2.info ='release dates'
  AND kt.kind ='movie'
  AND mi.movie_id = t.id
  AND it2.id = mi.info_type_id
  AND kt.id = t.kind_id
  AND mc.movie_id = t.id
  AND cn.id = mc.company_id
  AND ct.id = mc.company_type_id
  AND miidx.movie_id = t.id
  AND it.id = miidx.info_type_id
  AND mi.movie_id = miidx.movie_id
  AND mi.movie_id = mc.movie_id
  AND miidx.movie_id = mc.movie_id;
```

BENCHMARK CHALLENGES

Measuring the efficacy of LLM-based tuning agents is difficult because LLMs know how to optimize queries from common database benchmarks.
→ TPC-H, TPC-DS, JOB, DSB, ClickBench

Name/ordering obfuscation is not enough to hide a query's source.

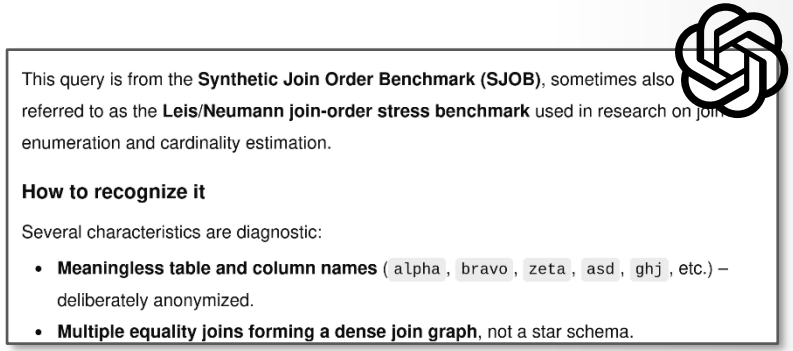
Obfuscated JOB Query 13a

```
SELECT MIN(zeta.jrn),
       MIN(echo.mux),
       MIN(alpha.vqk)
FROM delta, bravo, hotel, golf, zeta,
     alpha, charlie, foxtrot, echo
WHERE udf_str_3() = delta.lmn
  AND delta.zxc = charlie.qwe
  AND bravo.tyu = charlie.poi
  AND charlie.asd = alpha.ghj
  AND alpha.ghj = zeta.rty
  AND zeta.rty = hotel.bnm
  AND alpha.ghj = foxtrot.klo
  AND golf.vbn = alpha.mnb
  AND udf_str_2() = golf.kjh
  AND udf_str_1() = hotel.mju
  AND alpha.ghj = echo.iop
  AND hotel.bnm = echo.hgf
  AND zeta.rty = echo.iop
  AND zeta.rty = charlie.asd
  AND udf_str_0() = bravo.lkj;
```

BENCHMARK CHALLENGES

Measuring the efficacy of LLM-based tuning agents is difficult because LLMs know how to optimize queries from common database benchmarks.
→ TPC-H, TPC-DS, JOB, DSB, ClickBench

Name/ordering obfuscation is not enough to hide a query's source.

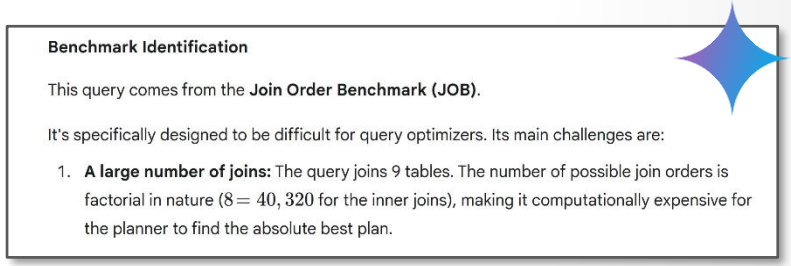


This query is from the **Synthetic Join Order Benchmark (SJOB)**, sometimes also referred to as the **Leis/Neumann join-order stress benchmark** used in research on join enumeration and cardinality estimation.

How to recognize it

Several characteristics are diagnostic:

- **Meaningless table and column names** (`alpha`, `bravo`, `zeta`, `asd`, `ghj`, etc.) – deliberately anonymized.
- **Multiple equality joins forming a dense join graph**, not a star schema.

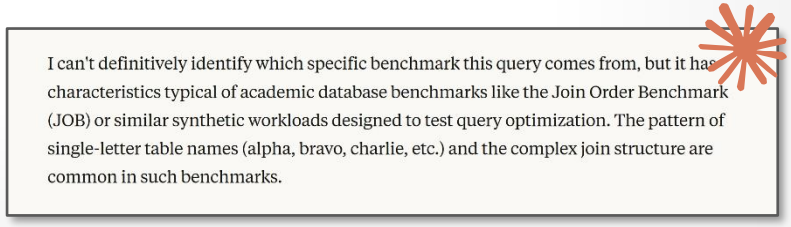


Benchmark Identification

This query comes from the **Join Order Benchmark (JOB)**.

It's specifically designed to be difficult for query optimizers. Its main challenges are:

1. **A large number of joins:** The query joins 9 tables. The number of possible join orders is factorial in nature ($8 = 40,320$ for the inner joins), making it computationally expensive for the planner to find the absolute best plan.

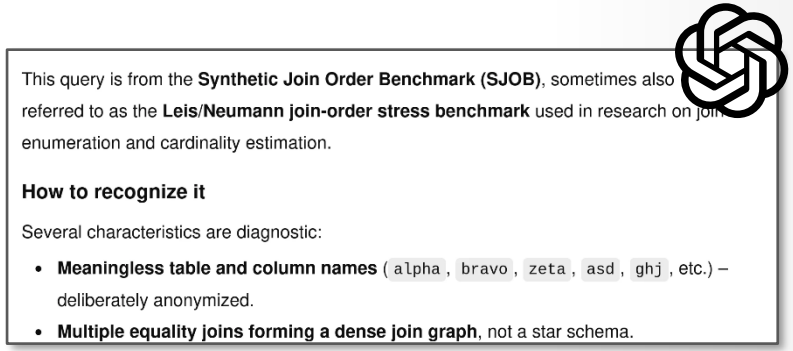


I can't definitively identify which specific benchmark this query comes from, but it has characteristics typical of academic database benchmarks like the Join Order Benchmark (JOB) or similar synthetic workloads designed to test query optimization. The pattern of single-letter table names (alpha, bravo, charlie, etc.) and the complex join structure are common in such benchmarks.

BENCHMARK CHALLENGES

Measuring the efficacy of LLM-based tuning agents is difficult because LLMs know how to optimize queries from common database benchmarks.
→ TPC-H, TPC-DS, JOB, DSB, ClickBench

Name/ordering obfuscation is not enough to hide a query's source.

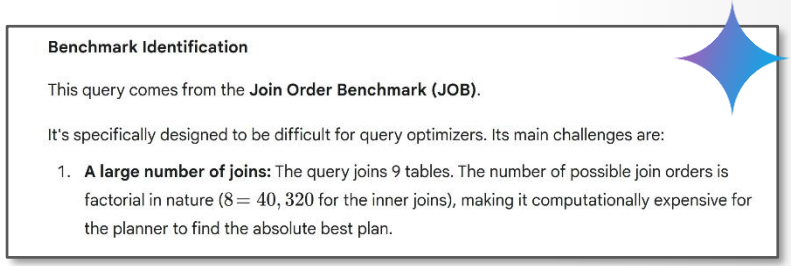


This query is from the **Synthetic Join Order Benchmark (SJOB)**, sometimes also referred to as the **Leis/Neumann join-order stress benchmark** used in research on join enumeration and cardinality estimation.

How to recognize it

Several characteristics are diagnostic:

- **Meaningless table and column names** (`alpha`, `bravo`, `zeta`, `asd`, `ghj`, etc.) – deliberately anonymized.
- **Multiple equality joins forming a dense join graph**, not a star schema.

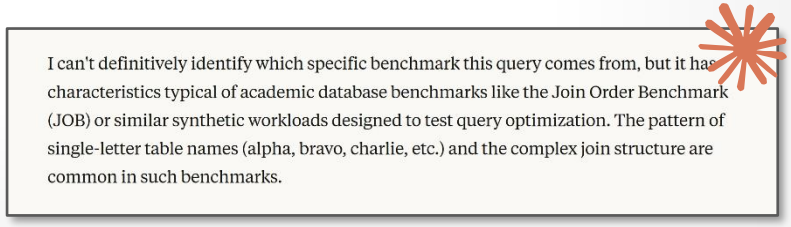


Benchmark Identification

This query comes from the **Join Order Benchmark (JOB)**.

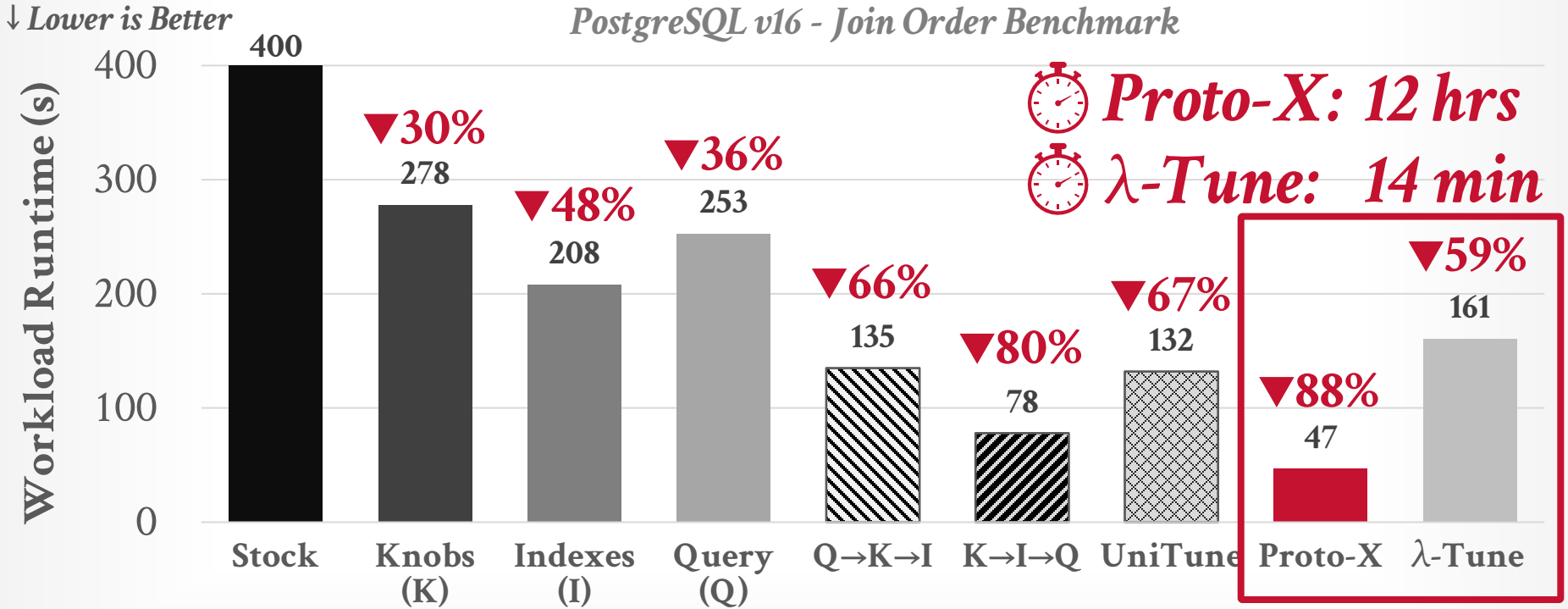
It's specifically designed to be difficult for query optimizers. Its main challenges are:

1. **A large number of joins:** The query joins 9 tables. The number of possible join orders is factorial in nature ($8 = 40,320$ for the inner joins), making it computationally expensive for the planner to find the absolute best plan.



I can't definitively identify which specific benchmark this query comes from, but it has characteristics typical of academic database benchmarks like the Join Order Benchmark (JOB) or similar synthetic workloads designed to test query optimization. The pattern of single-letter table names (alpha, bravo, charlie, etc.) and the complex join structure are common in such benchmarks.

DATABASE TUNING AGENTS



AGENT BOOSTING

Use LLMs to discover and extract relevant information to accelerate specialized tuning agents.

Bootstrap the initial configuration to a tuning agent so that it does not start from scratch each time.

→ Facilitates knowledge transfer as a database / application evolve over time.

→ Also supports transfer for tuning decisions across workloads, environments, and versions.

THIS IS GOING TO SOUND CRAZY, BUT WHAT IF WE USED LARGE-LANGUAGE MODELS TO BOOST AUTOMATIC DATABASE TUNING ALGORITHMS BY LEVERAGING PRIOR HISTORY? WE WILL FIND BETTER CONFIGURATIONS MORE QUICKLY THAN RETRAINING FROM SCRATCH!

SIGMOD 2026

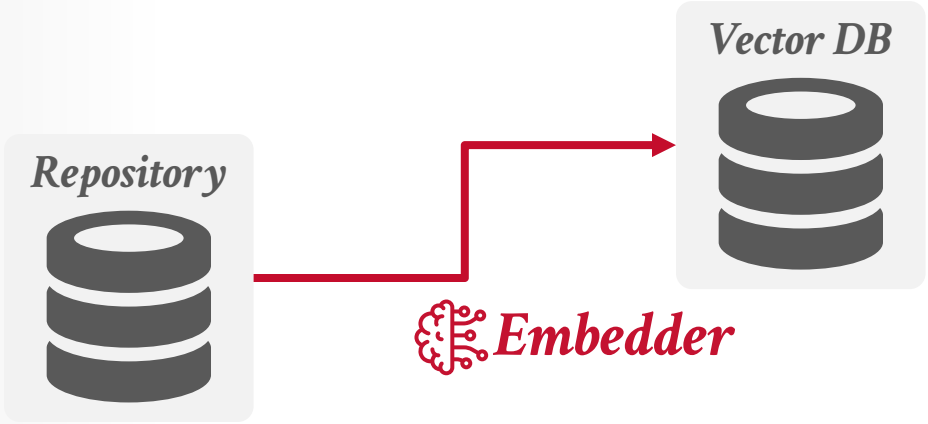
AGENT BOOSTING

Repository

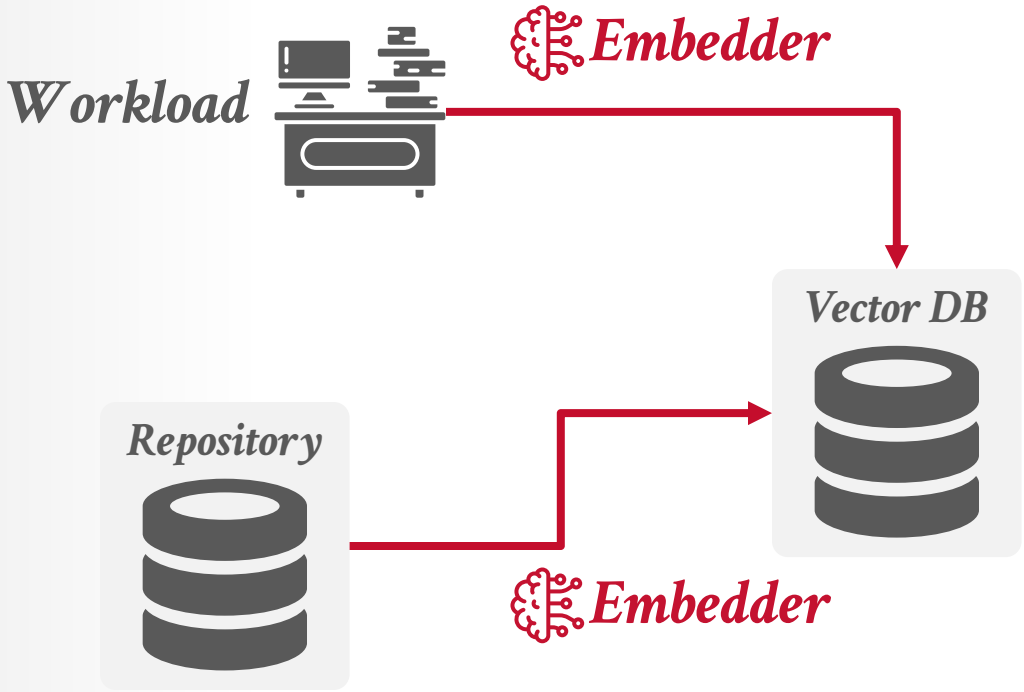
Schematic

```
Metadata:
TABLE ${tbl_name} [${tbl_stats}] (
  ${col_name} ${col_type} (${col_stats})
  INDEX ${index_definition}
)

Query:
${sql_or_query_plan}
```

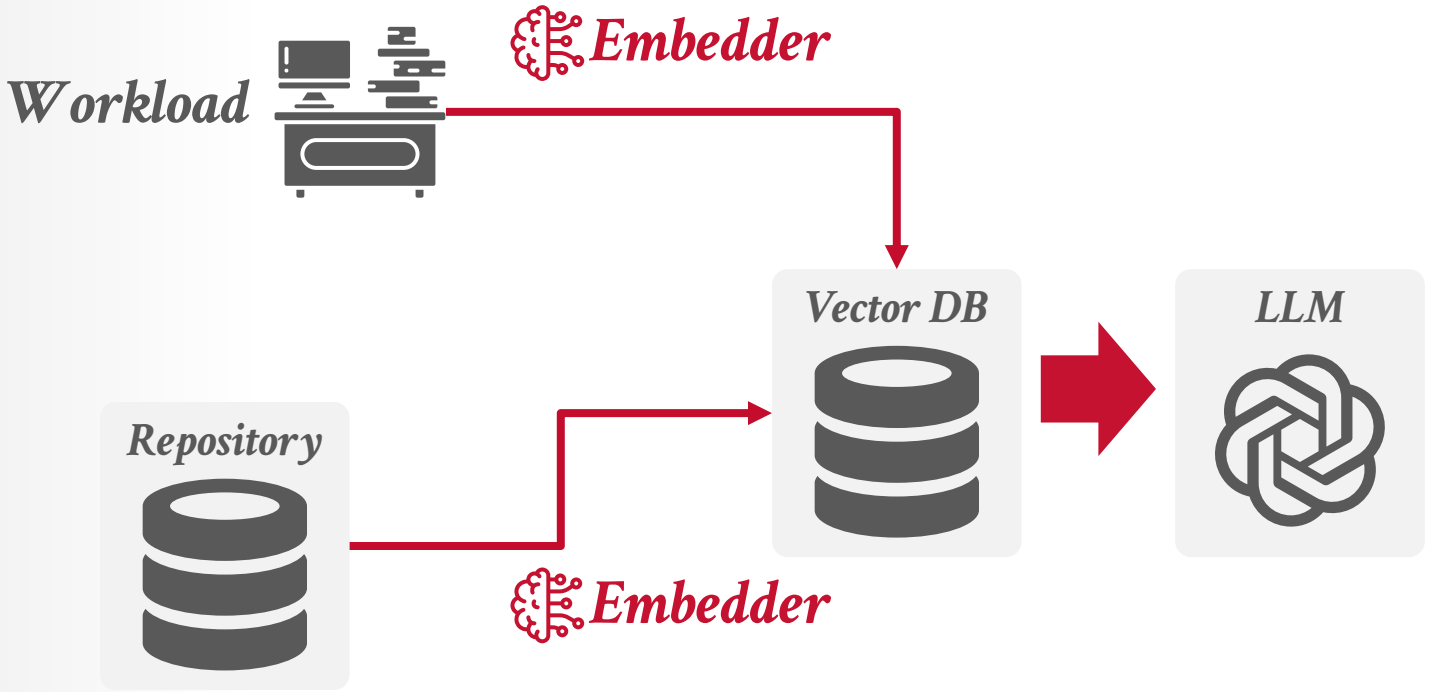
AGENT BOOSTING



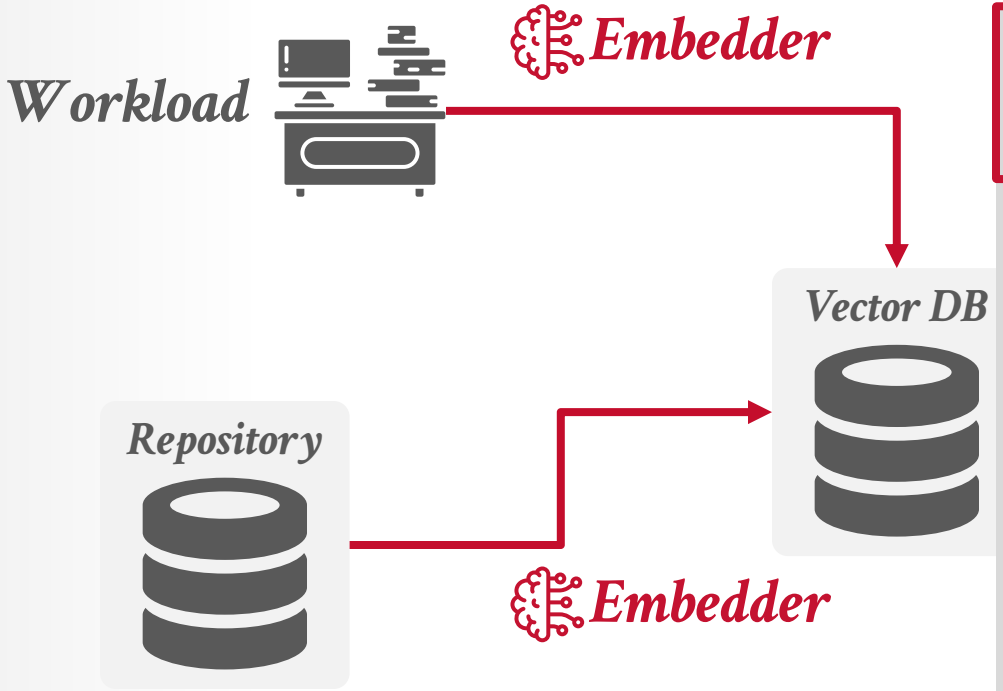
AGENT BOOSTING



AGENT BOOSTING



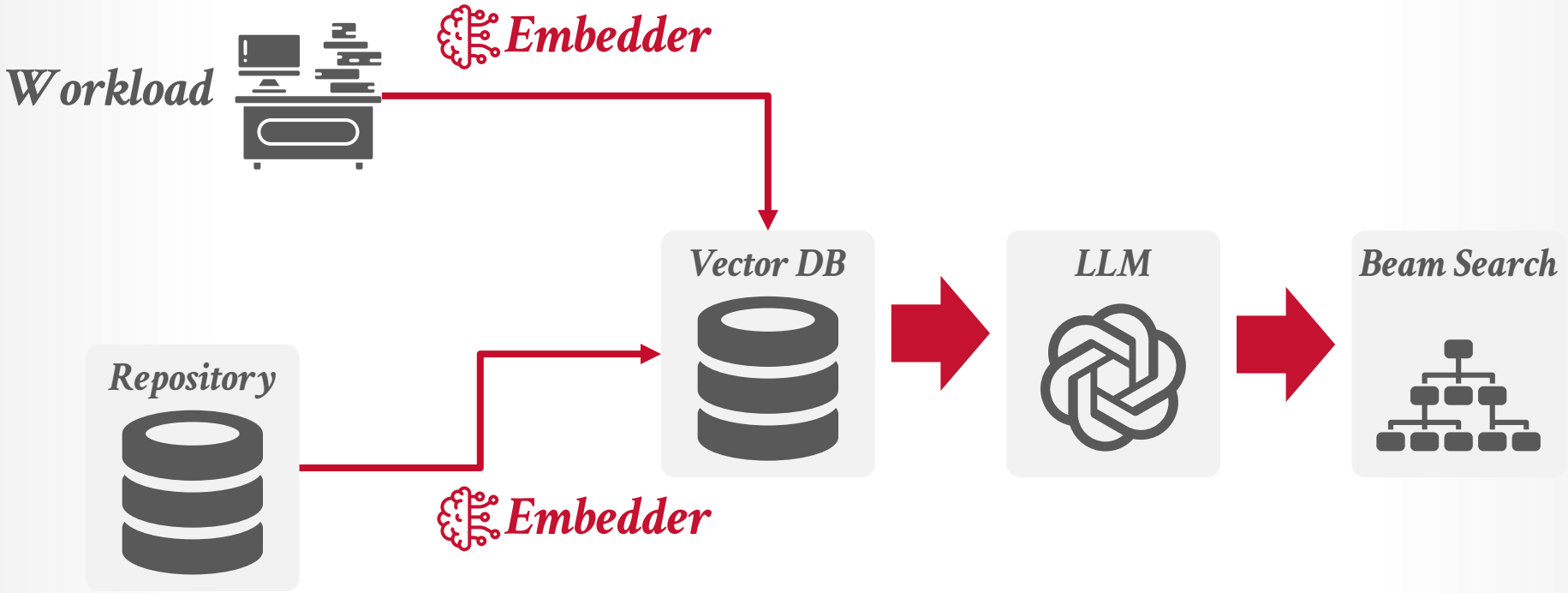
AGENT BOOSTING



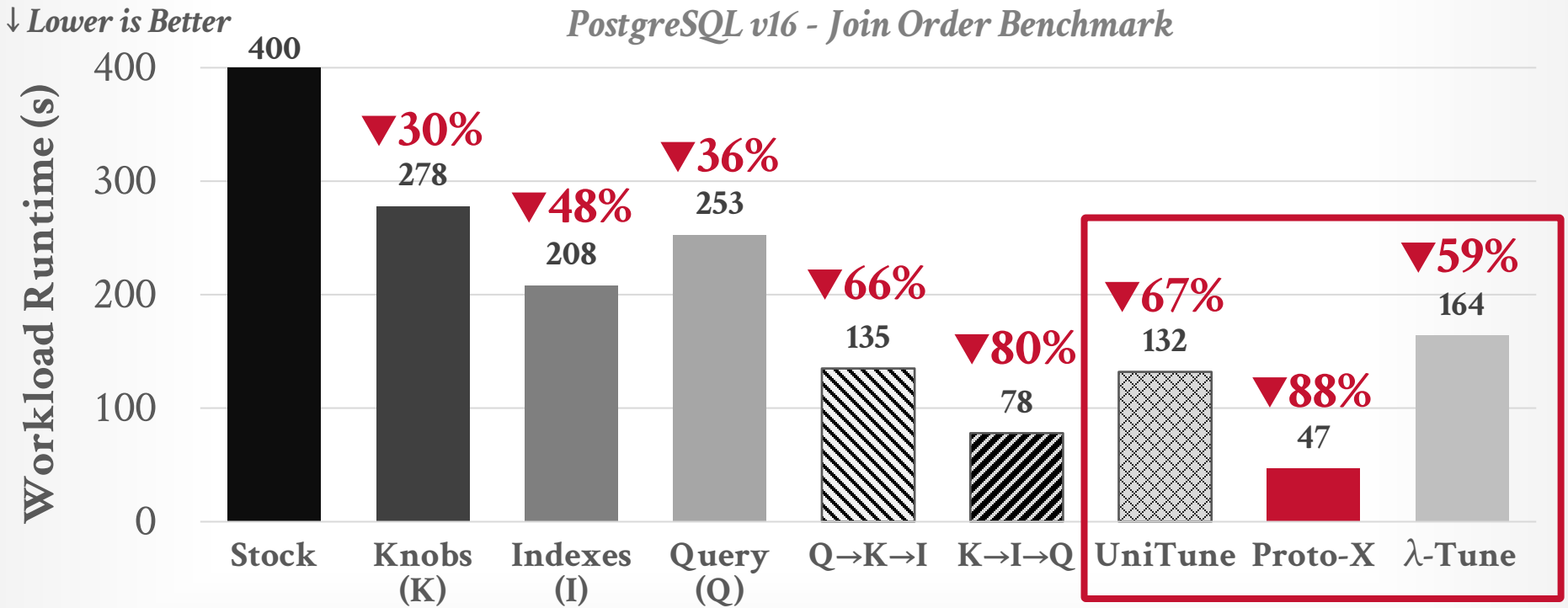
Prompt

```
You are the world's best DBA for ${SYSTEM}
that wants to make databases as fast as
possible because you don't want to disappoint
your parents.

These are the ranges of the system knobs set
in the system. ${SYSTEM_KNOB_RANGES}

You are provided with several reference
configurations from similar query runs.
${REFERENCES}

**Instruction**
1. Analyze the below *TARGET*
2. Remember you are optimizing ${SYSTEM}
3. Think carefully, consult references, make
   suggestions
4. Out the requested information in *JSON*
```

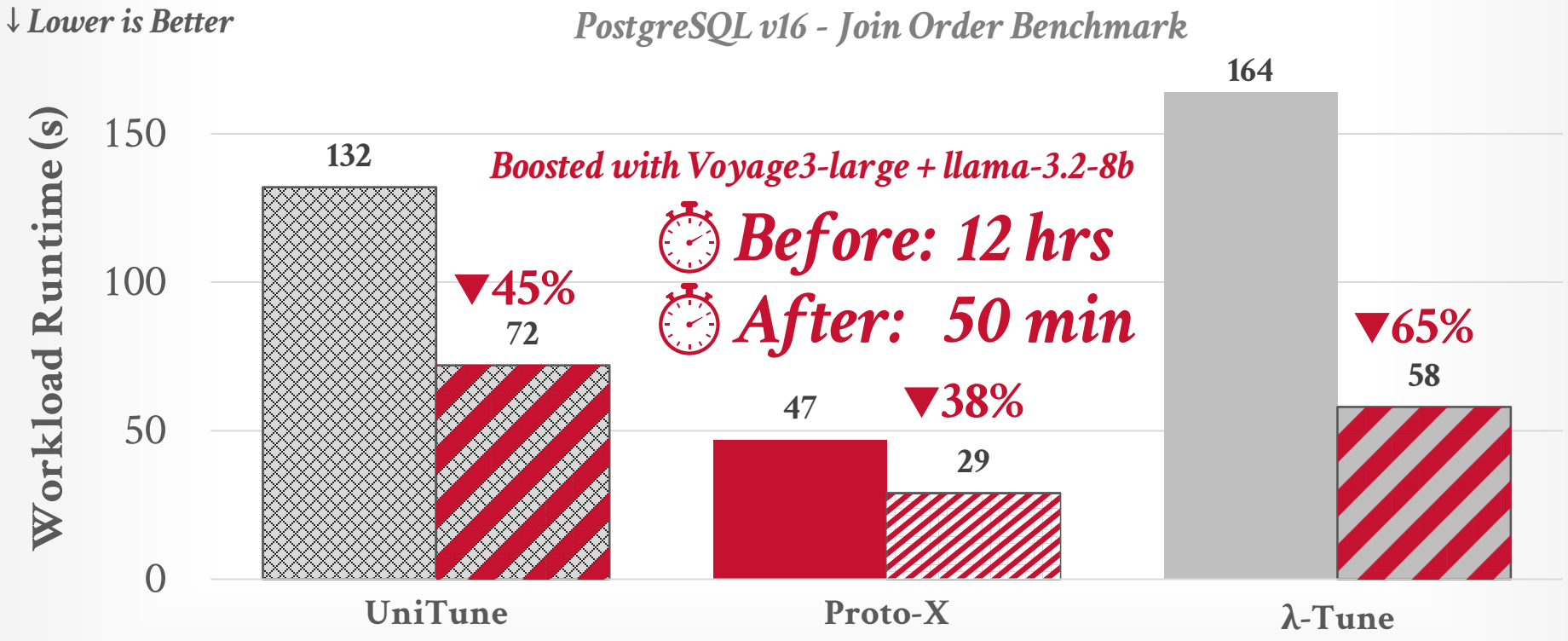
AGENT BOOSTING



AGENT BOOSTING



AGENT BOOSTING



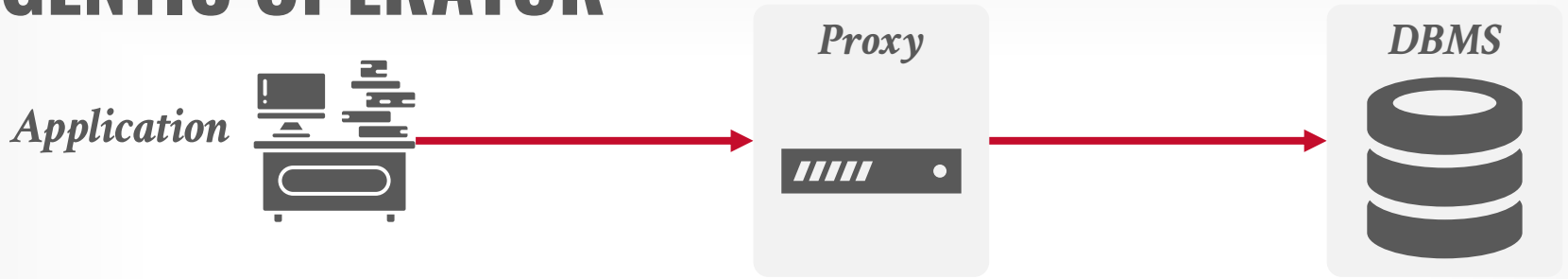
OBSERVATION

None of the approaches we discussed today handle lifecycle management issues of running a production database fleet.

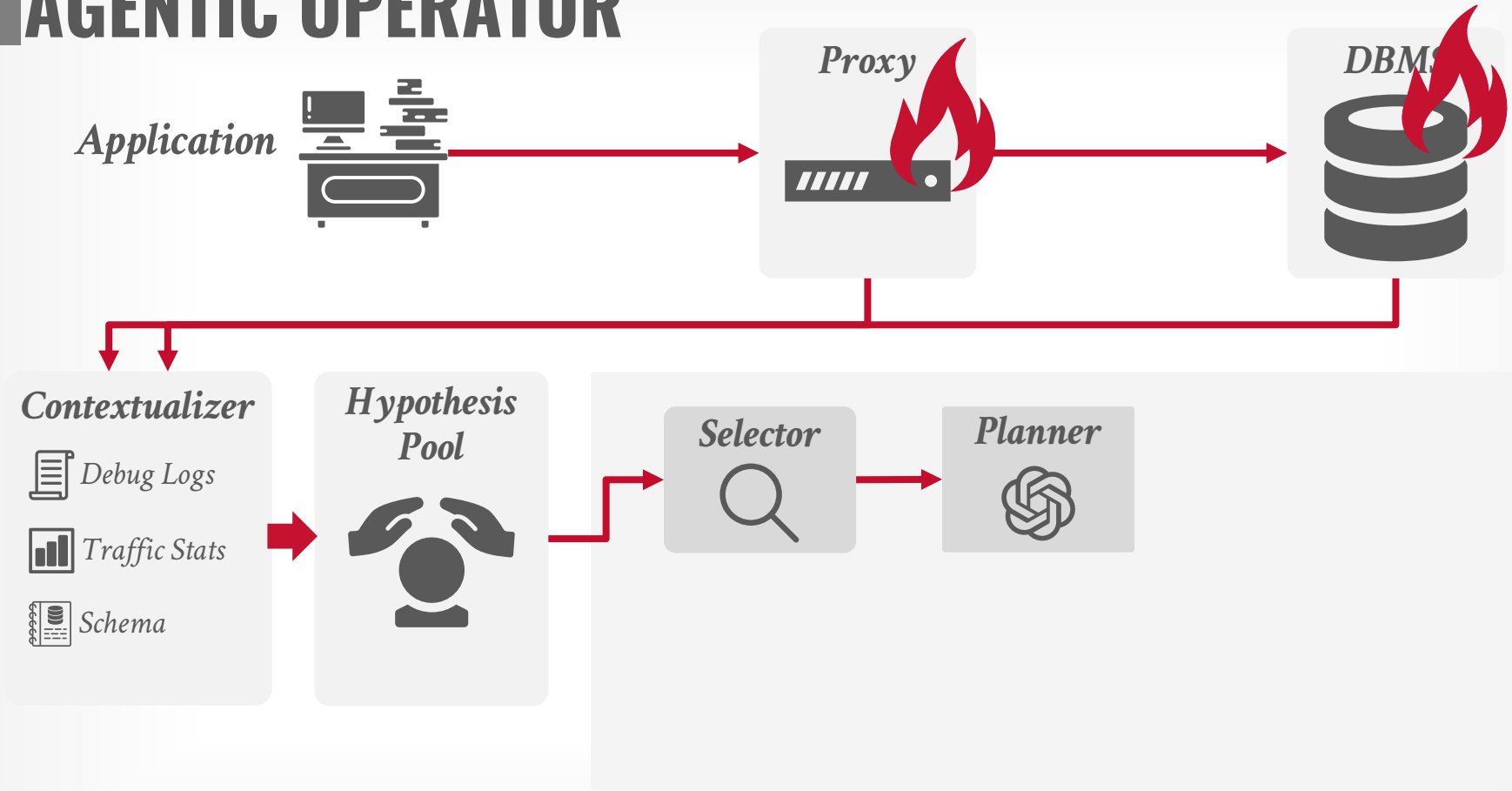
The agents do not consider temporal and relationship aspects of the choices they make.

→ Emergency vs. Long-term Care
→ Dependencies Between Instances
→ Black Swans vs. Scheduled Events

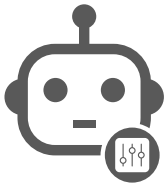
AGENTIC OPERATOR

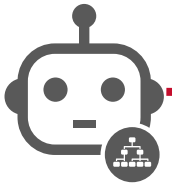


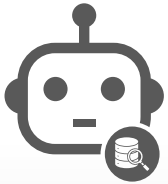
AGENTIC OPERATOR



AGENTIC OPERATOR







Planner

Agent Specification

```
Generate knob configuration…

Usage:
- Choose this when...
- Provides a one-off...

Estimated Runtime: Fast

Avoid:
- When holistic is needed
  (use tool-x instead).
- When disk trashing is
  high.
```

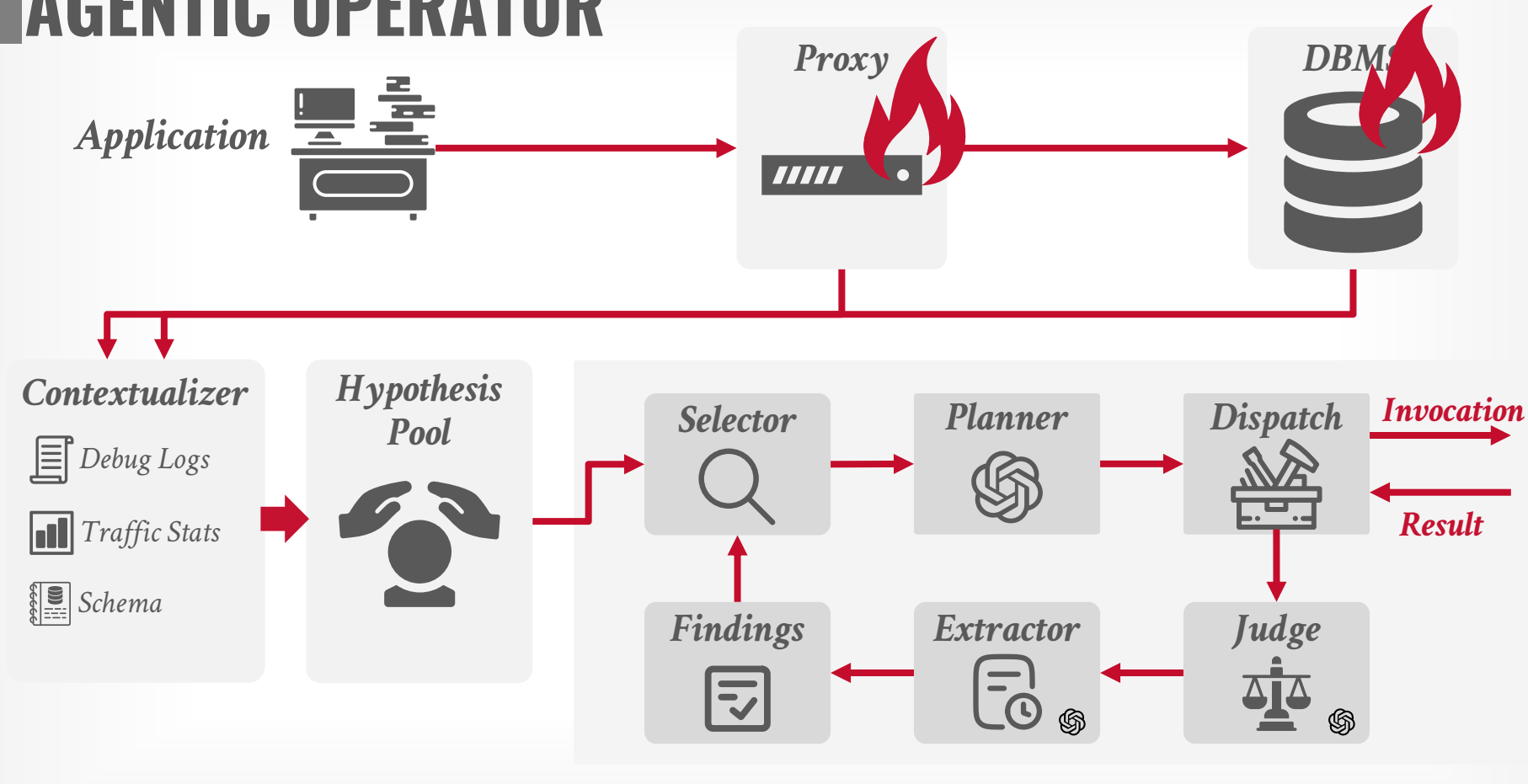
"Mission"

Scenarios To Use

Cost

Scenarios To Avoid

AGENTIC OPERATOR



Coding Agents

for Databases

CODING AGENTS FOR DATABASE DEVELOPMENT

There is significant adoption of agents to help build, debug, and optimize DBMSs.

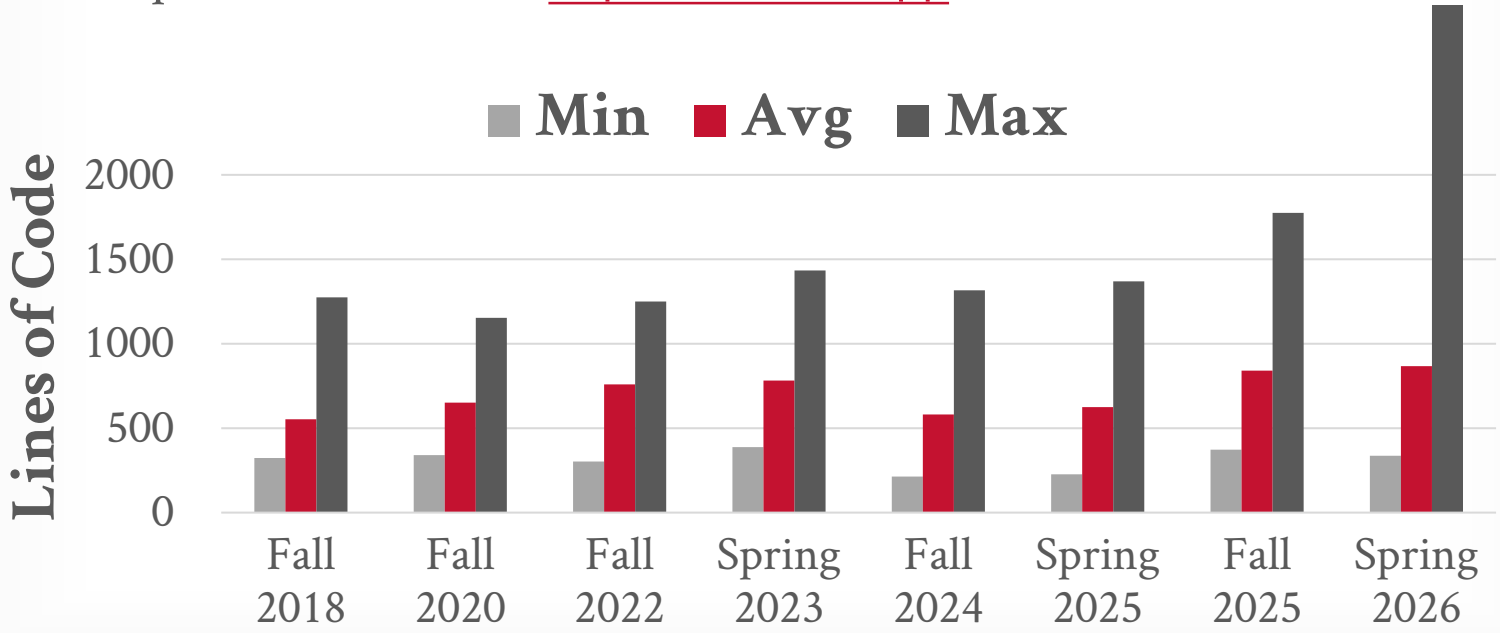
High-quality reference implementations for (almost) every DBMS components are available for model training.

→ Data Structures / Indexes
→ Storage Managers
→ Query Engines
→ Concurrency Control Protocols
→ Logging & Recovery

CMU-DB B+TREE PROJECT SUBMISSIONS

Student project submissions for **BusTub** DBMS.
→ Core Implementation File (`b_plus_tree.cpp`)

99% Increase!



CODING AGENTS FOR DATABASE DEVELOPMENT

There is significant adoption of agents to help build, debug, and optimize DBMSs.

High-quality reference implementations for (almost) every DBMS component are available for model training.

→ Data Structures / Indexes
→ Storage Managers
→ Query Engines
→ Concurrency Control Protocols
→ Logging & Recovery

CODING AGENTS FOR DATABASE DEVELOPMENT

There is significant adoption of agents to help build, debug, and optimize DBMSs.

High-quality reference implementations for (almost) every DBMS component are available for model training.

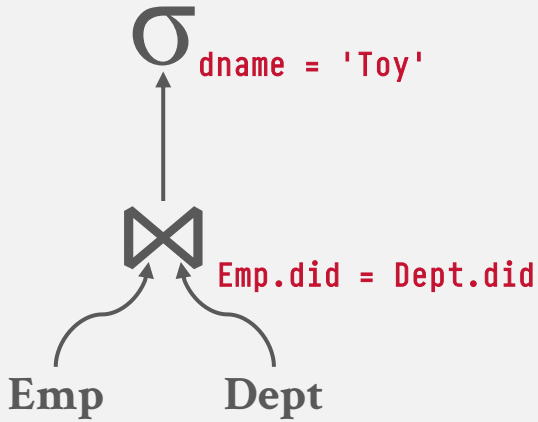
→ Data Structures / Indexes
→ Storage Managers
→ Query Engines
→ Concurrency Control Protocols
→ Logging & Recovery

QUERY OPTIMIZER / PLANNER

A DBMS's query optimizer is responsible for converting a query into an execution plan.

Typically comprised of four parts:

→ Intermediate Representations
→ Transformation Rules
→ Enumeration / Search Algorithms
→ Cost Model

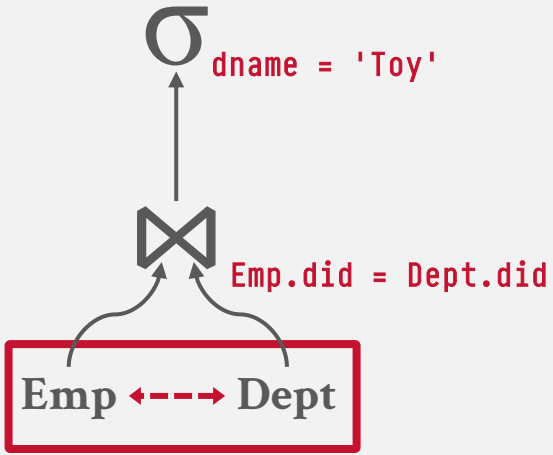


QUERY OPTIMIZER / PLANNER

A DBMS's query optimizer is responsible for converting a query into an execution plan.

Typically comprised of four parts:

→ Intermediate Representations
→ Transformation Rules
→ Enumeration / Search Algorithms
→ Cost Model

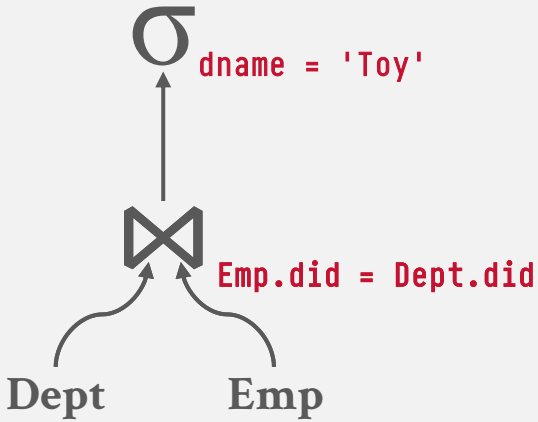


QUERY OPTIMIZER / PLANNER

A DBMS's query optimizer is responsible for converting a query into an execution plan.

Typically comprised of four parts:

→ Intermediate Representations
→ Transformation Rules
→ Enumeration / Search Algorithms
→ Cost Model

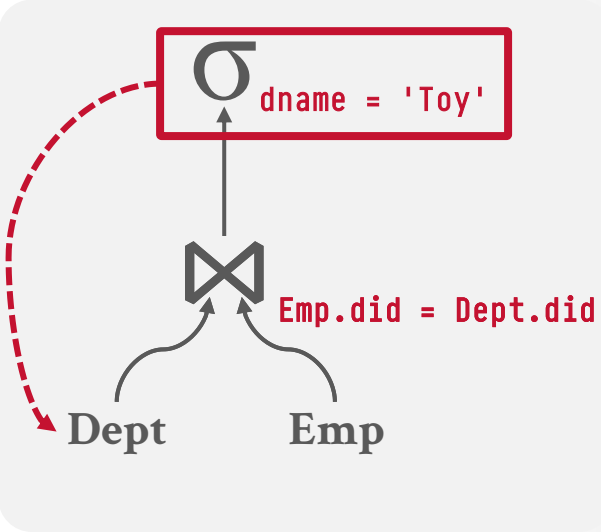


QUERY OPTIMIZER / PLANNER

A DBMS's query optimizer is responsible for converting a query into an execution plan.

Typically comprised of four parts:

→ Intermediate Representations
→ Transformation Rules
→ Enumeration / Search Algorithms
→ Cost Model

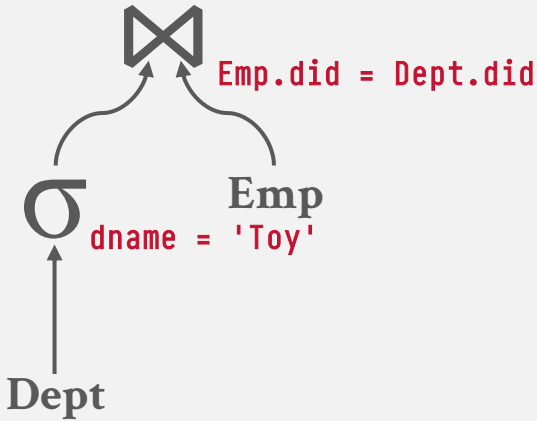


QUERY OPTIMIZER / PLANNER

A DBMS's query optimizer is responsible for converting a query into an execution plan.

Typically comprised of four parts:

→ Intermediate Representations
→ Transformation Rules
→ Enumeration / Search Algorithms
→ Cost Model



There are few open-source / public reference implementations of query optimizers.

APACHE calcite™
GREENPLUM ORCA
Velox

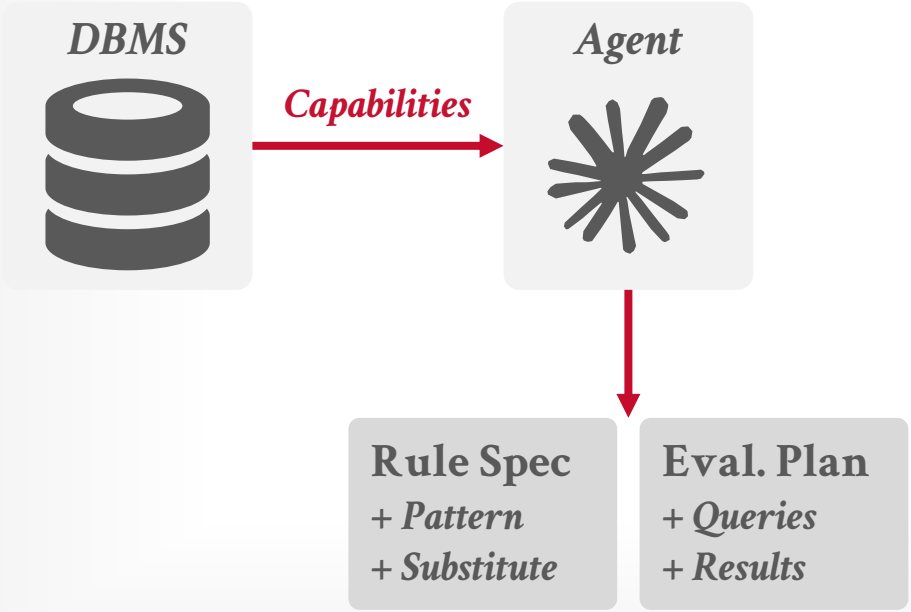
QUERY OPTIMIZER VIBE CODING

The scope and complexity of adding a new transformation rule is more difficult than other parts of the DBMS.

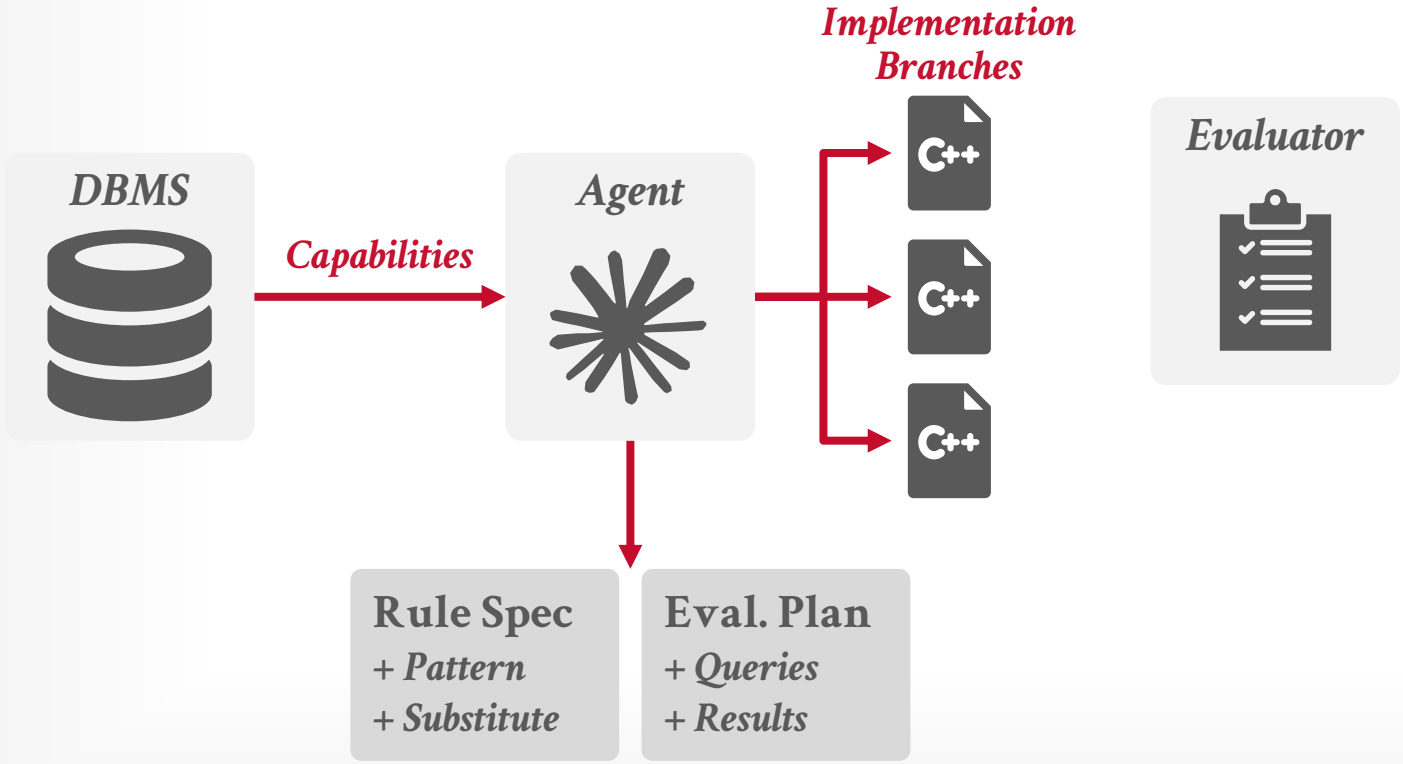
To create a new transformation rule:

→ Identify possible operators and their properties.
→ Synthesize rule (pattern + substitute) and evaluation plan.
→ Verify the rule is semantically correct.
→ Extend cost model to support new rule.
→ Validate runtime benefit.

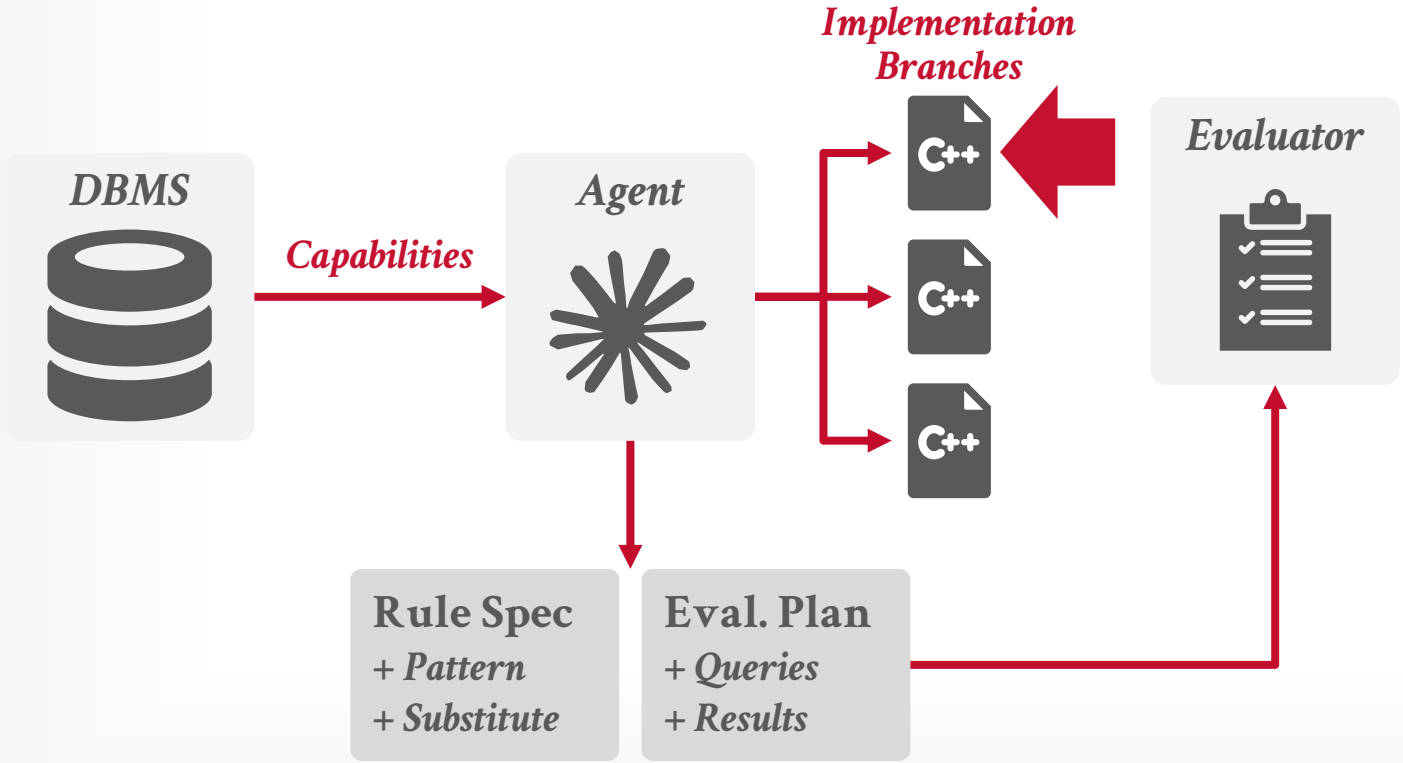
QUERY OPTIMIZER VIBE CODING



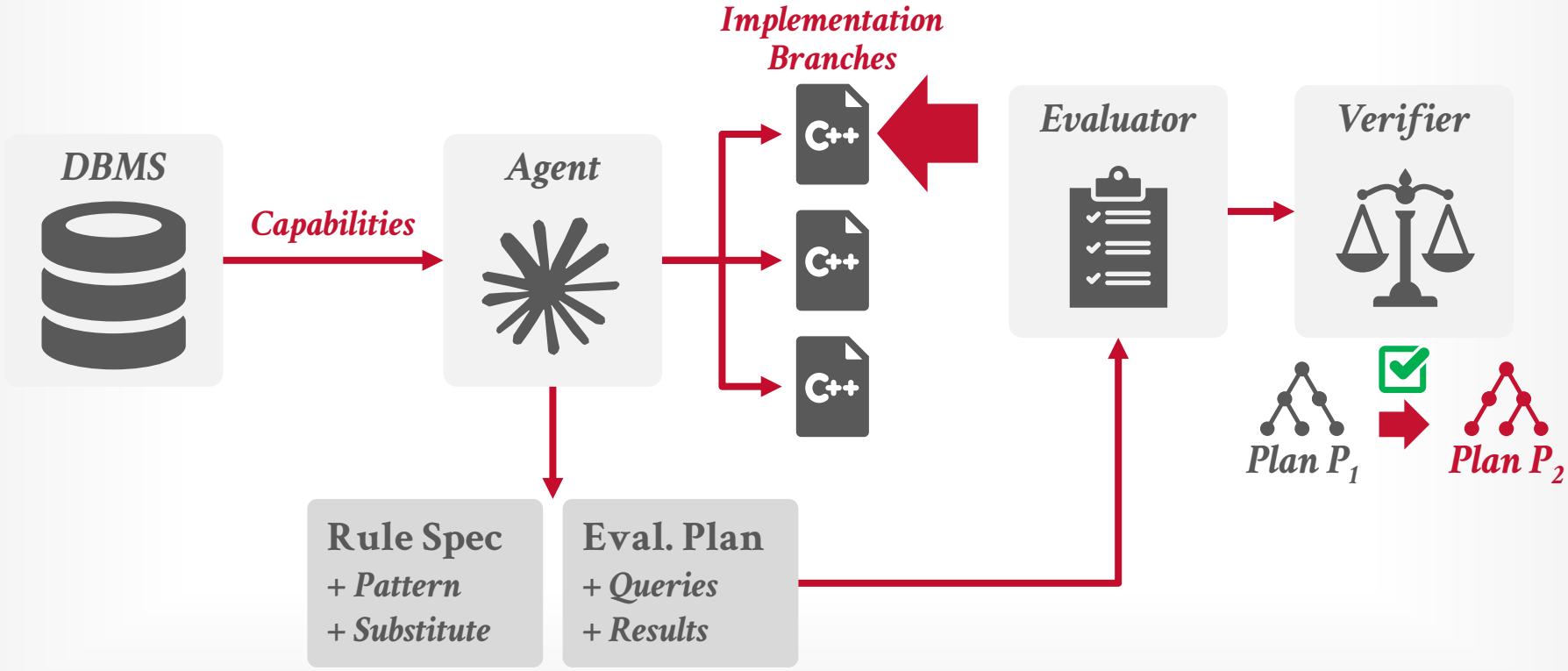
QUERY OPTIMIZER VIBE CODING



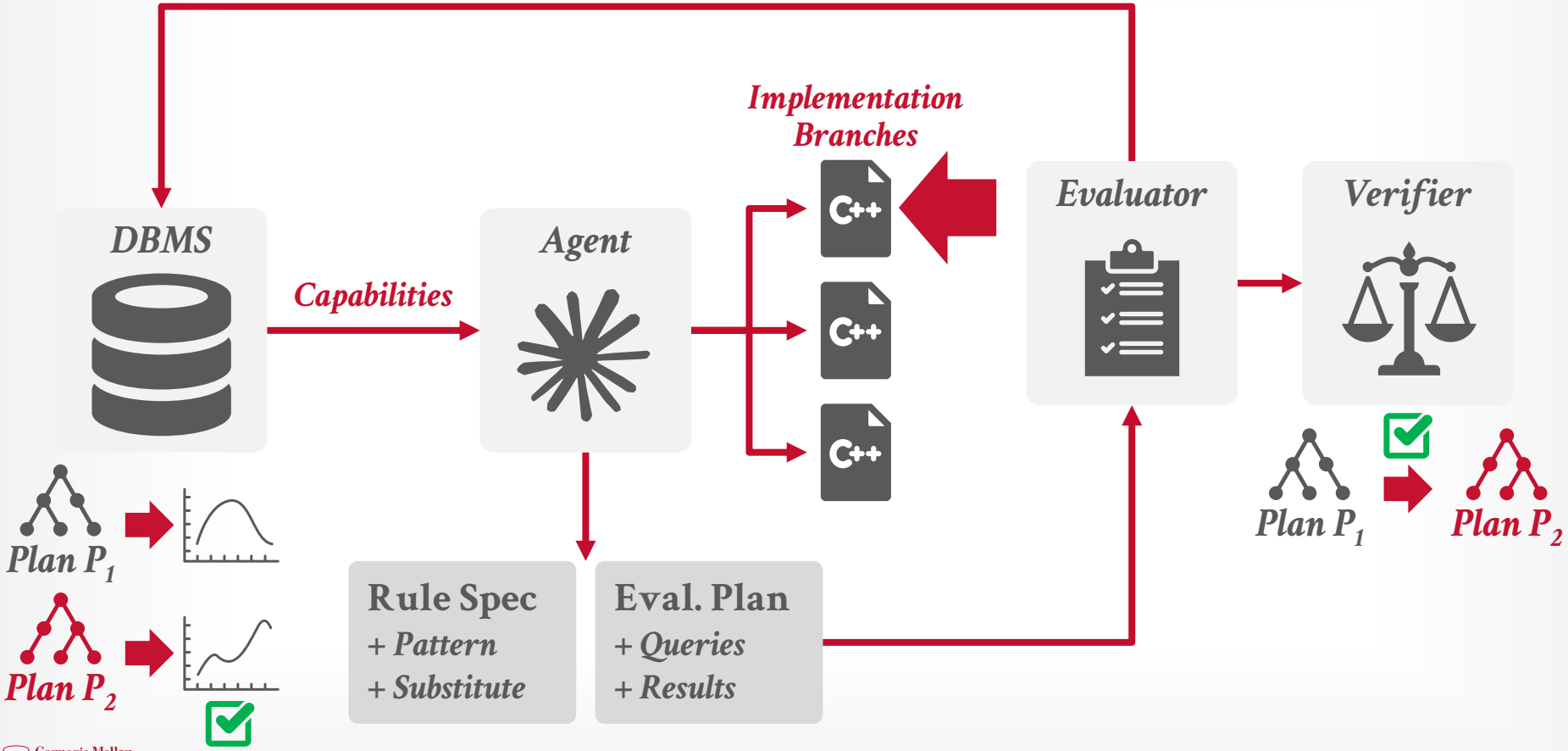
QUERY OPTIMIZER VIBE CODING



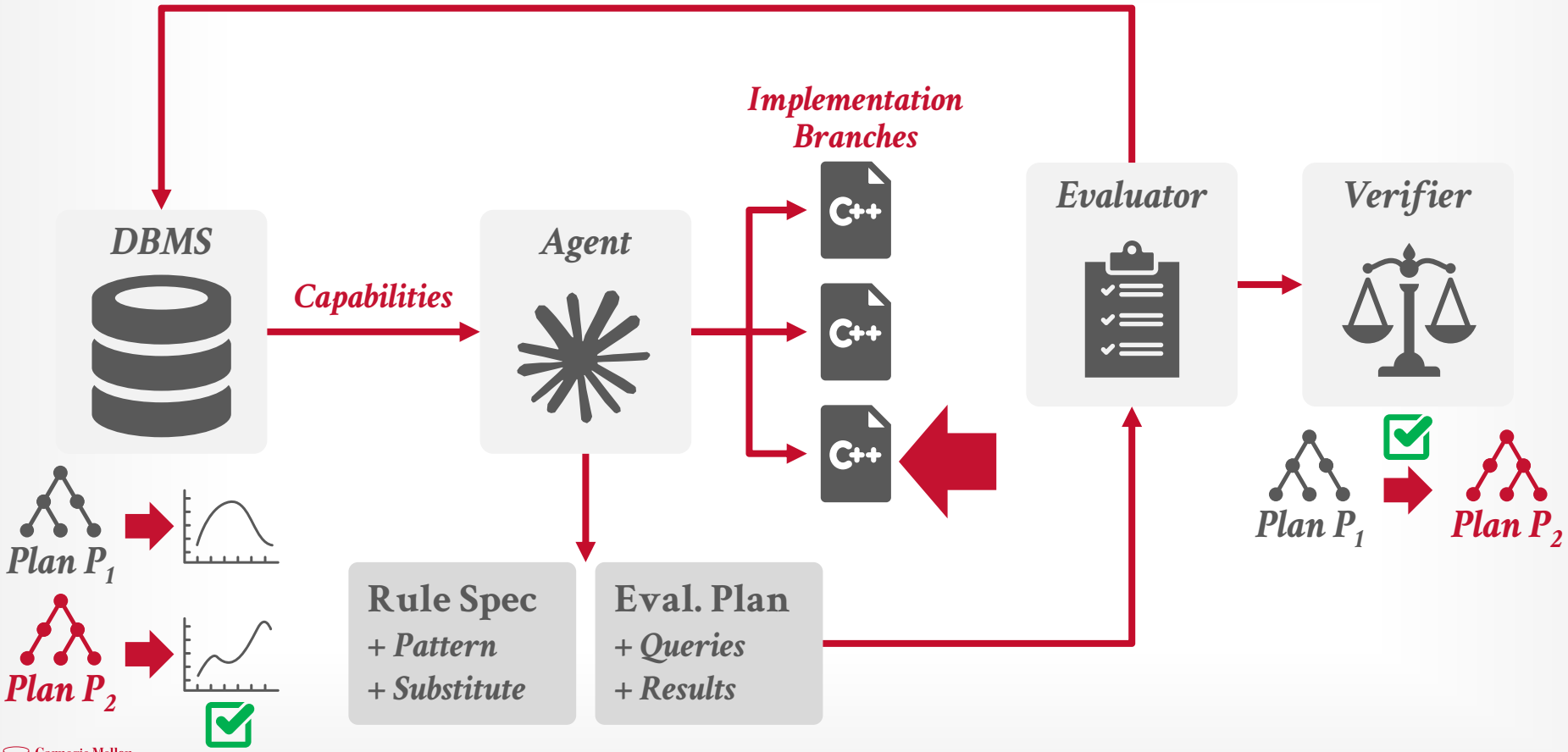
QUERY OPTIMIZER VIBE CODING



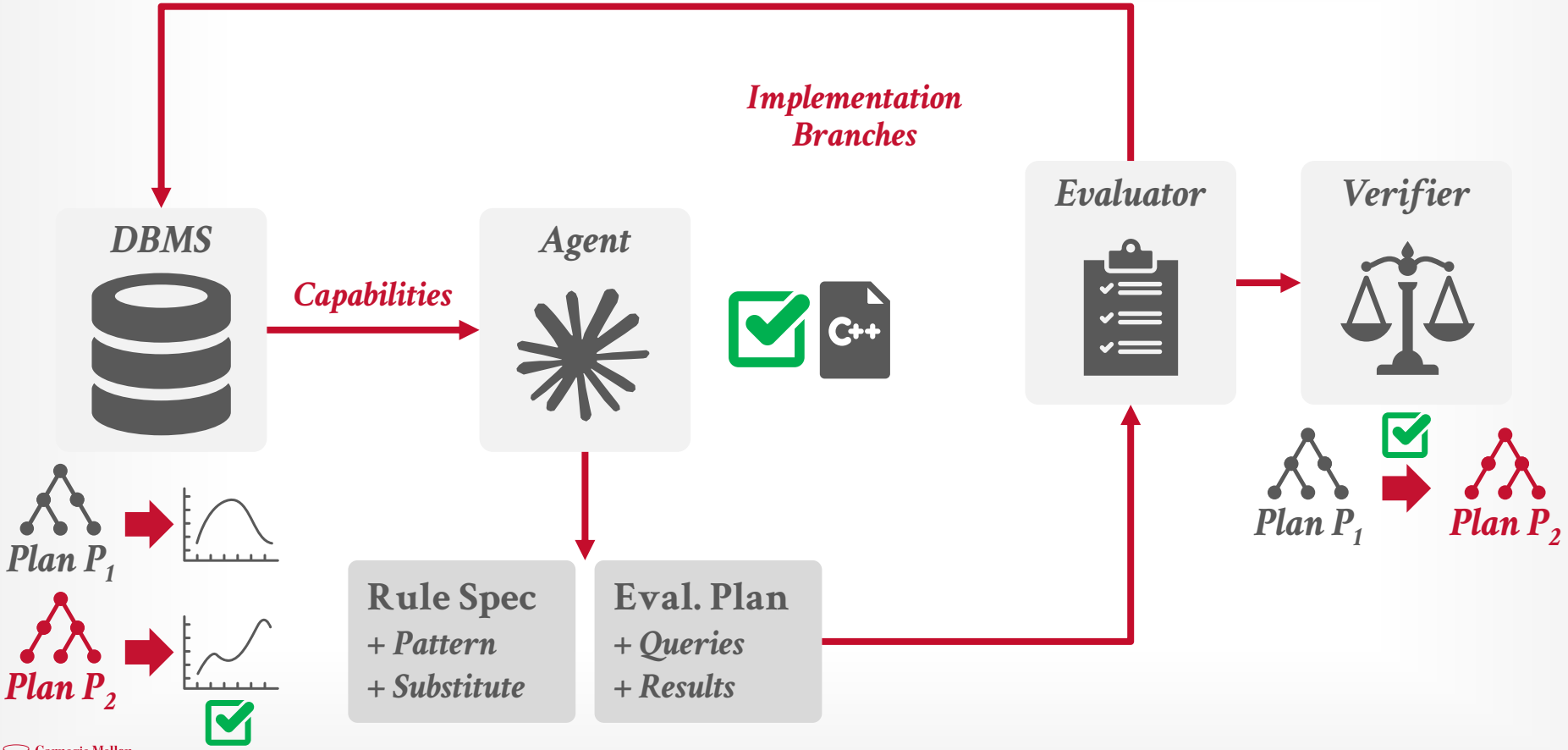
QUERY OPTIMIZER VIBE CODING



QUERY OPTIMIZER VIBE CODING



QUERY OPTIMIZER VIBE CODING



QUERY OPTIMIZER VIBE CODING

Blindly accepting what the LLM spits out is fraught with problems, but it is non-trivial to verify that two query plans are equivalent.

→ Examples: UW Cosette (2018), Berkeley QED Solver (2024), SJTU QLSolver (2023), Microsoft (2026)

We have successfully vibed two optimization passes from the Germans, but we manually verified them.

→ **DPHyp** (2008), **Unnesting v2** (2025)

Coding agents love to add special case code, but an optimizer should strive to be as general as possible.

QUERY OPTIMIZER VIBE CODING

TPC-H Query #7

```
SELECT EXTRACT(YEAR FROM l_shipdate) AS l_year, ...
  FROM supplier, lineitem, ...
```

Agent

```
struct PredicateSummary {
   lineage:     HashMap<Column, ValueRef>, // each output col→baseval
   eq_classes:  UnionFind<ValueRef>,       // a=b learned across plan
   ⋮
}
enum ValueRef { Base(Column), ExtractYear(Column), Derived(Column) }
```

CONCLUSION

Databases pose several significant challenges and pitfalls for AI agents.

Tuning Agents for Databases:

→ Combination of specialized algorithms controlled by reasoning models.

Coding Agents for Databases:

→ Frameworks that can automatically manipulate prompts for rapid implementation and testing of new components.

END

pavlo@cs.cmu.edu