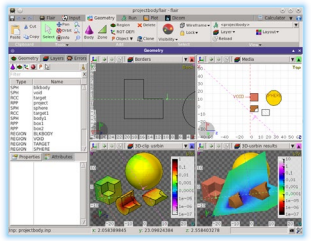
Diagnosing and Managing Performance in Data Systems

Markos Markakis

August 17, 2026

Complex needs mean complex data systems



Scientific

AI

GPUs

Memory Pooling

Novel Applications

Operational Demands

Hardware Advances

Cloud & Managed Services

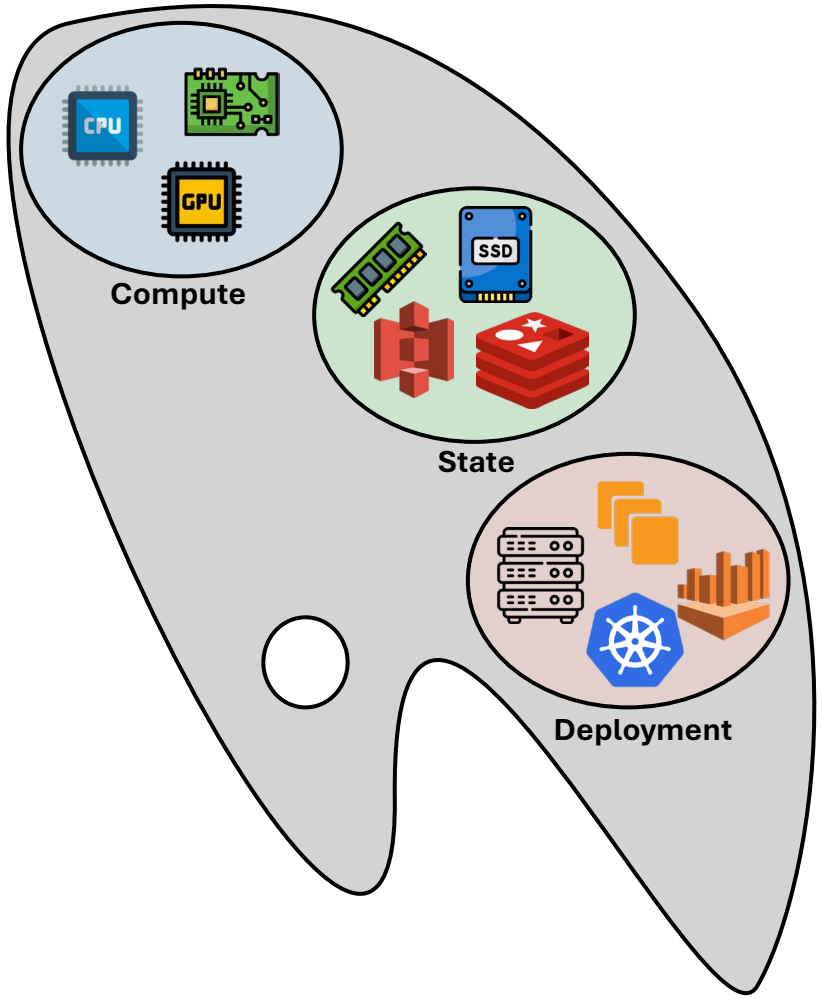
Traffic Management

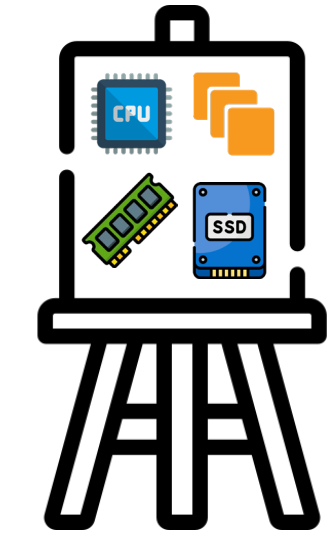
Identity & Access

Containers

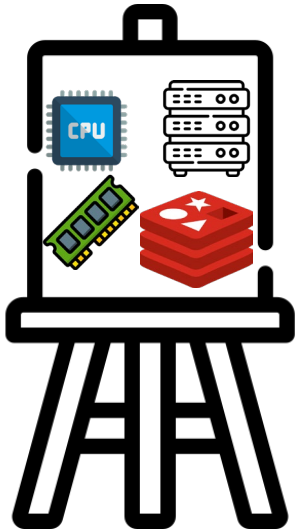
Serverless

Complexity can unlock flexibility…

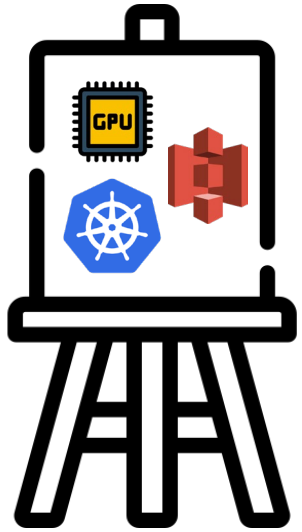




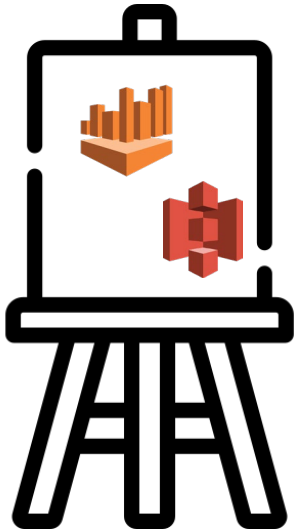
General-purpose



Low latency

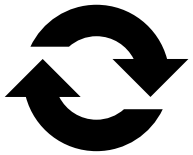


High Throughput

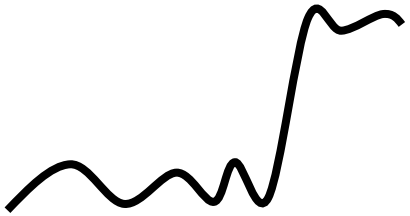


Bursty

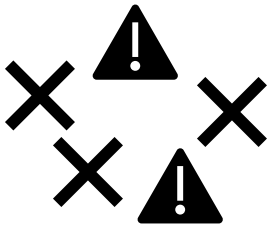
…but it makes remaining efficient challenging



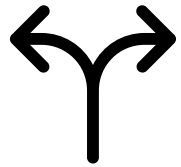
Updates/Bugs



Latency Spikes



Errors



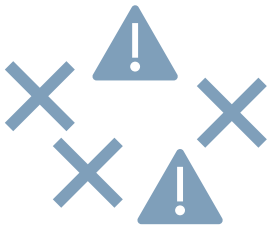
Balancing

Costly Idle Capacity

Scaling

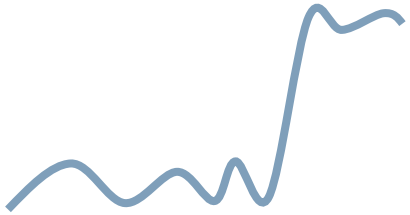
Two related needs: diagnosis & management

Diagnosis



Errors

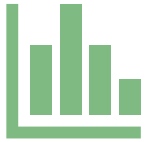
Costly Idle Capacity



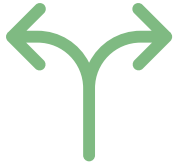
Latency Spikes

Management

Updates / Bugs

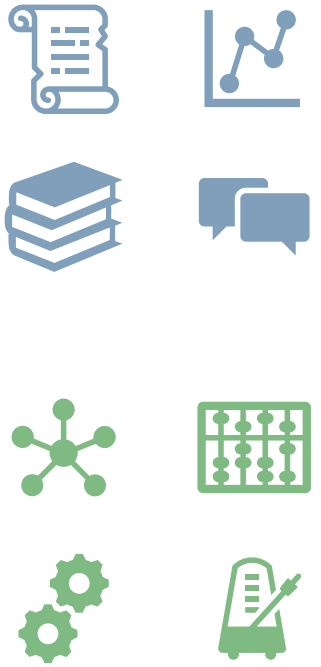


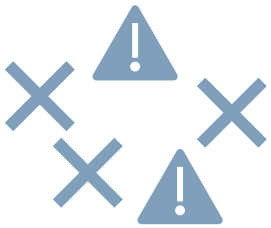
Scaling



Balancing

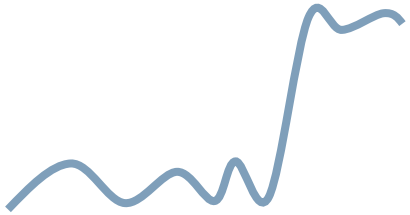
Across needs, data and levers are low-level





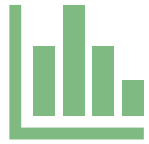
Errors

Costly Idle Capacity



Latency Spikes

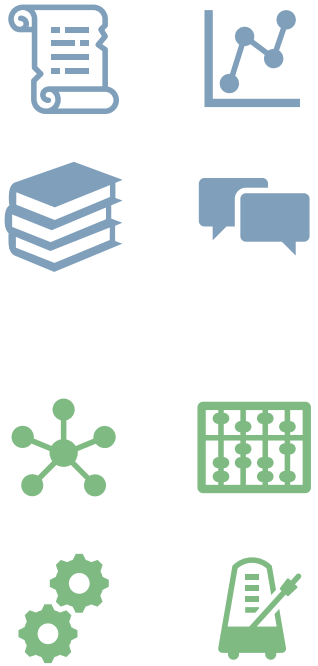
Updates / Bugs

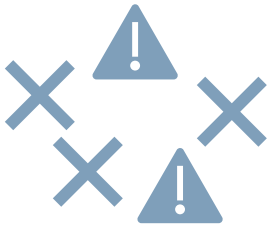


Scaling

Balancing

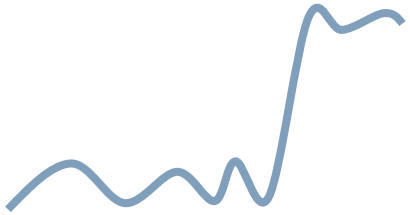
But operators reason at a high level





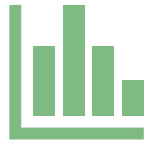
Errors

Costly Idle Capacity



Latency Spikes

Updates / Bugs



Scaling

Balancing

Why are queries slow?

Cheaper instance?

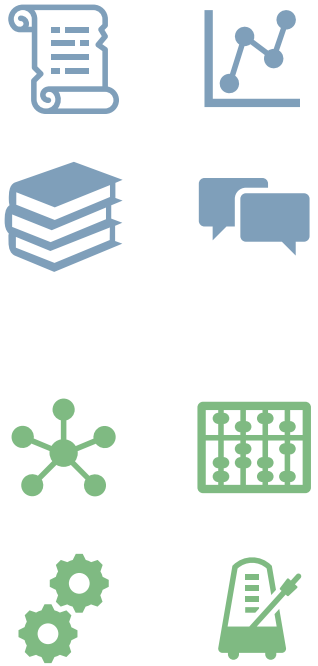
Bug or noise?

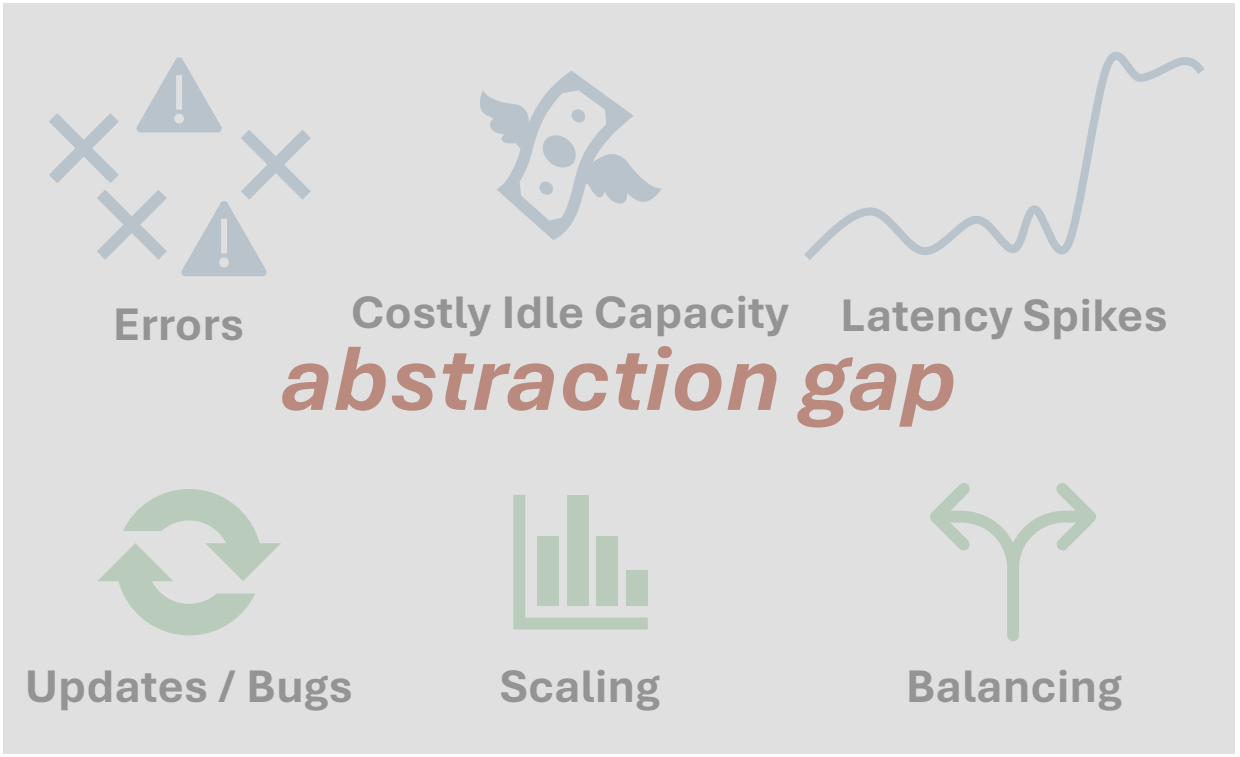
Always < 10 s

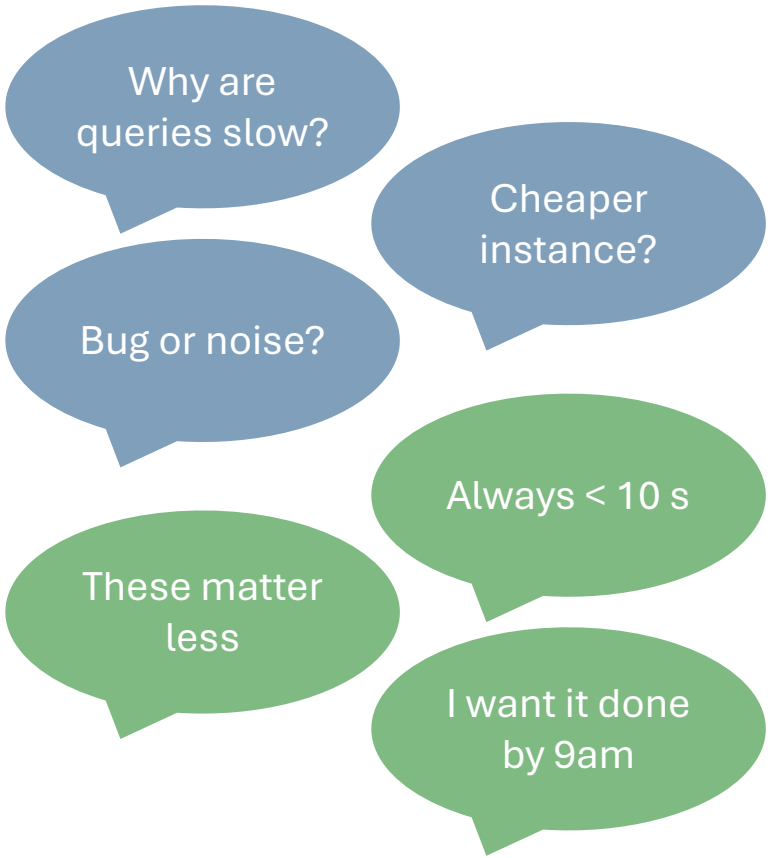
These matter less

I want it done by 9am

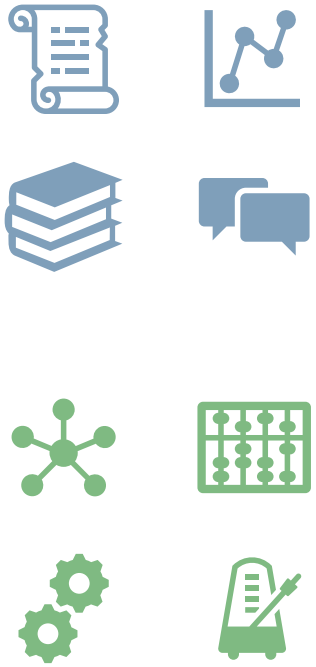
Systems & operators face an abstraction gap

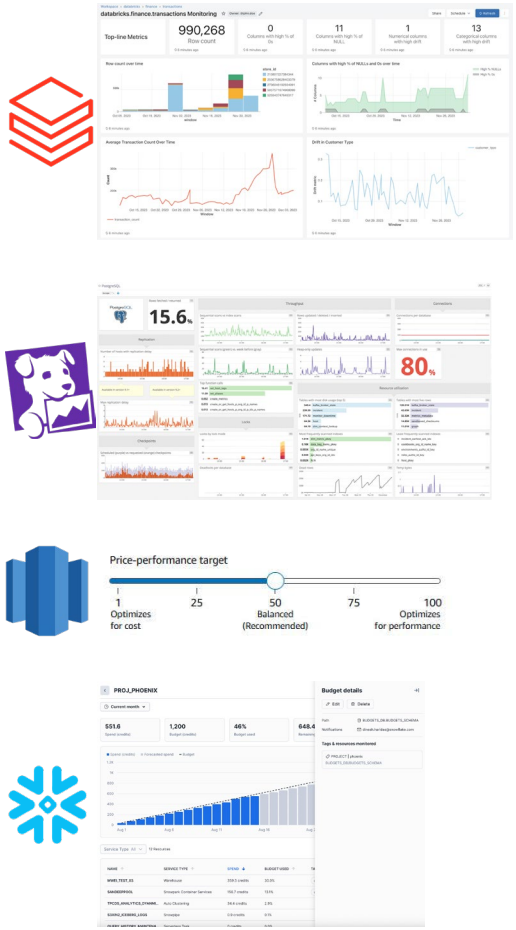






Prior work has mitigated some of the gap





abstraction gap

Why are queries slow?

Cheaper instance?

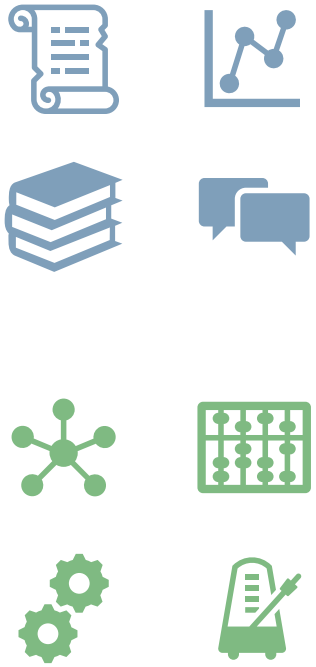
Bug or noise?

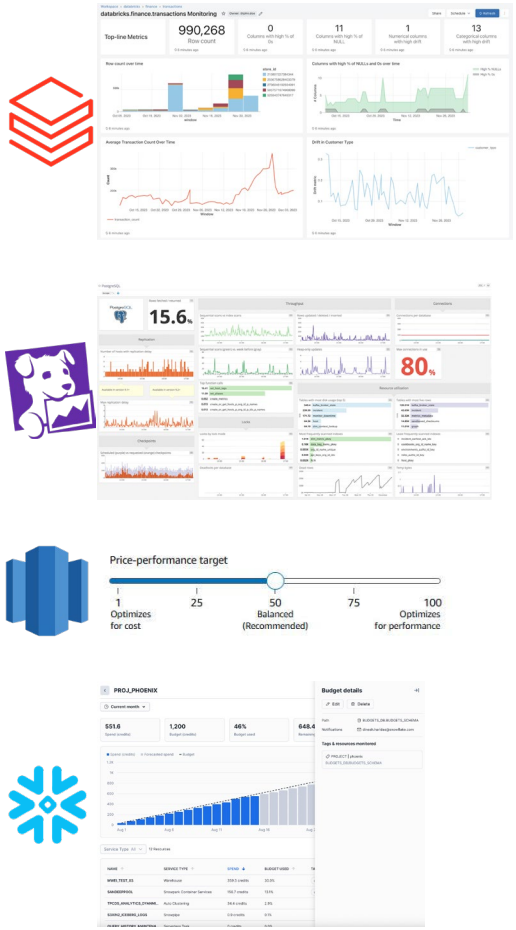
Always < 10 s

These matter less

I want it done by 9am

This thesis helps close it further





LOGos

abstraction gap

AutoSLO

Why are queries slow?

Cheaper instance?

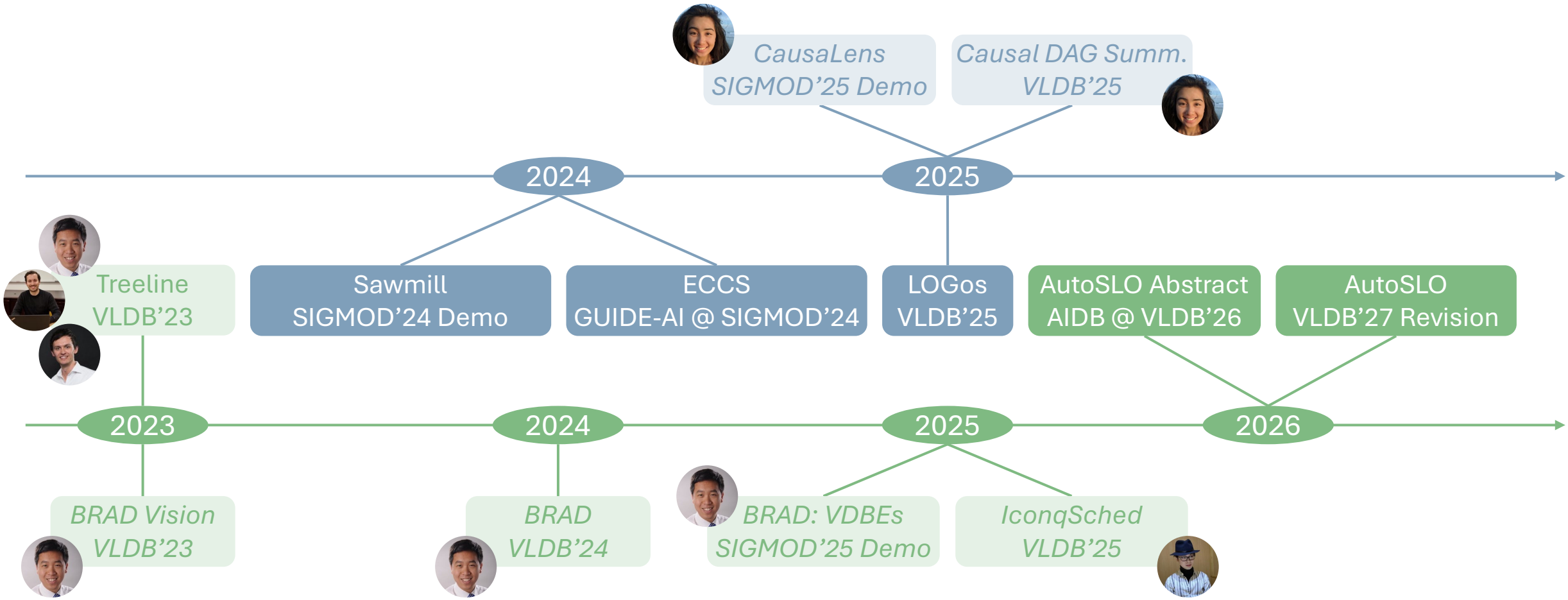
Bug or noise?

Always < 10 s

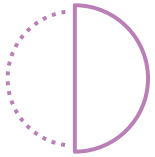
These matter less

I want it done by 9am

Bringing together two lines of work



Two different domains, three shared principles



Scope modeling by intent

Complexity = hard to model fully

Model just enough to act

Refine models using feedback

Models will be imperfect

Continuously integrate feedback

Prioritize effort by impact

Cannot respond to everything

Right amount of effort, right time

```
2023-11-06 16:34:40.811 EST [ 65495bf0.179a 3/2462 ] postgres@tpcds1 LOG:  statement: SET work_mem = 128;
2023-11-06 16:34:40.812 EST [ 65495bf0.179a 3/2463 ] postgres@tpcds1 LOG:  statement: SET max_parallel_workers = 1;
2023-11-06 16:34:40.812 EST [ 65495bf0.179a 3/2464 ] postgres@tpcds1 LOG:  statement: SET effective_cache_size = 262144;
2023-11-06 16:34:40.812 EST [ 65495bf0.179a 3/2465 ] postgres@tpcds1 LOG:  statement: SET maintenance_work_mem = 32768;
2023-11-06 16:34:40.813 EST [ 65495bf0.179a 3/2466 ] postgres@tpcds1 LOG:  statement: SET random_page_cost = 4;
2023-11-06 16:34:40.813 EST [ 65495bf0.179a 3/2467 ] postgres@tpcds1 LOG:  statement: SET seq_page_cost = 1;
2023-11-06 16:34:40.820 EST [ 65495bf0.179a 3/2468 ] postgres@tpcds1 LOG:  statement: -- Filename: query080.sql
2023-11-06 16:34:53.850 EST [ 65495bf0.179a 3/2468 ] postgres@tpcds1 LOG:  duration: 13029.853 ms
...
2023-11-07 08:15:48.440 EST [ 654a3884.49b7 3/3278 ] postgres@tpcds1 LOG:  statement: SET work_mem = 128;
2023-11-07 08:15:48.441 EST [ 654a3884.49b7 3/3279 ] postgres@tpcds1 LOG:  statement: SET max_parallel_workers = 1;
2023-11-07 08:15:48.441 EST [ 654a3884.49b7 3/3280 ] postgres@tpcds1 LOG:  statement: SET effective_cache_size = 262144;
2023-11-07 08:15:48.441 EST [ 654a3884.49b7 3/3281 ] postgres@tpcds1 LOG:  statement: SET maintenance_work_mem = 65536;
2023-11-07 08:15:48.442 EST [ 654a3884.49b7 3/3282 ] postgres@tpcds1 LOG:  statement: SET random_page_cost = 4;
2023-11-07 08:15:48.442 EST [ 654a3884.49b7 3/3283 ] postgres@tpcds1 LOG:  statement: SET seq_page_cost = 1;
2023-11-07 08:15:48.443 EST [ 654a3884.49b7 3/3284 ] postgres@tpcds1 LOG:  statement: -- Filename: query080.sql
2023-11-07 08:15:49.446 EST [ 654a3884.49b7 3/3284 ] postgres@tpcds1 LOG:  duration: 1003.228 ms
...
```

How can we go from this to correct quantitative conclusions?

Causal inference to the rescue!

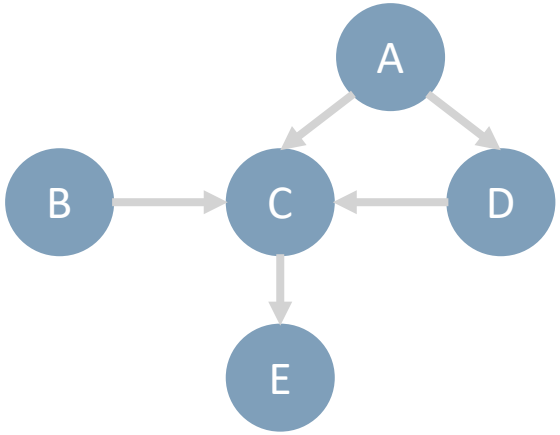
- Statistical framework for drawing unbiased cause-effect conclusions.
- Relationships are expressed through average treatment effects:

$$ATE(T, Y) = E[Y|do(T = 1)] - E[Y|do(T = 0)]$$

- Ideal setting: run a controlled experiment to sidestep confounding.
- But logs are observational data – can't go back and "do".
- To avoid confounding, we must instead adjust for the right variables.

What to adjust for? Look at causal graph

- For each variable in the dataset, create a node.
- For each potential direct causal relationship, create a directed edge.
- Key assumption:
  - Each node is fully determined by its parents.
  - Conditionally independent of everything else, given its parents.
- Causal discovery: automatically build a causal graph from data.



Causal discovery over log data is non-trivial

- **Functional dependencies**

  The same event creates dependent log entries.

- **Large number of variables**

  Many off-the-shelf algorithms scale exponentially.

- **Biased data collection**

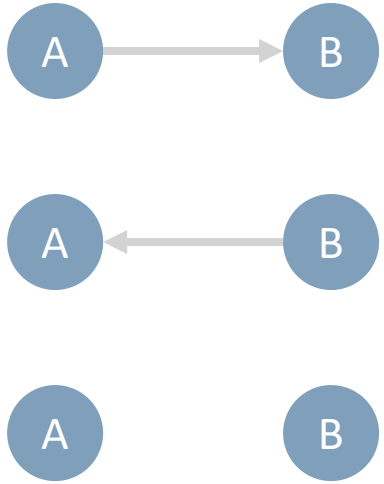
  Failure cases relatively rare, effect can be “drowned out”.

Table 1: Causal discovery on the PostgreSQL dataset. ✓ and ● indicate a non-empty and empty causal graph, respectively; ▲ indicates a 30-minute timeout; and ✗ indicates an error.

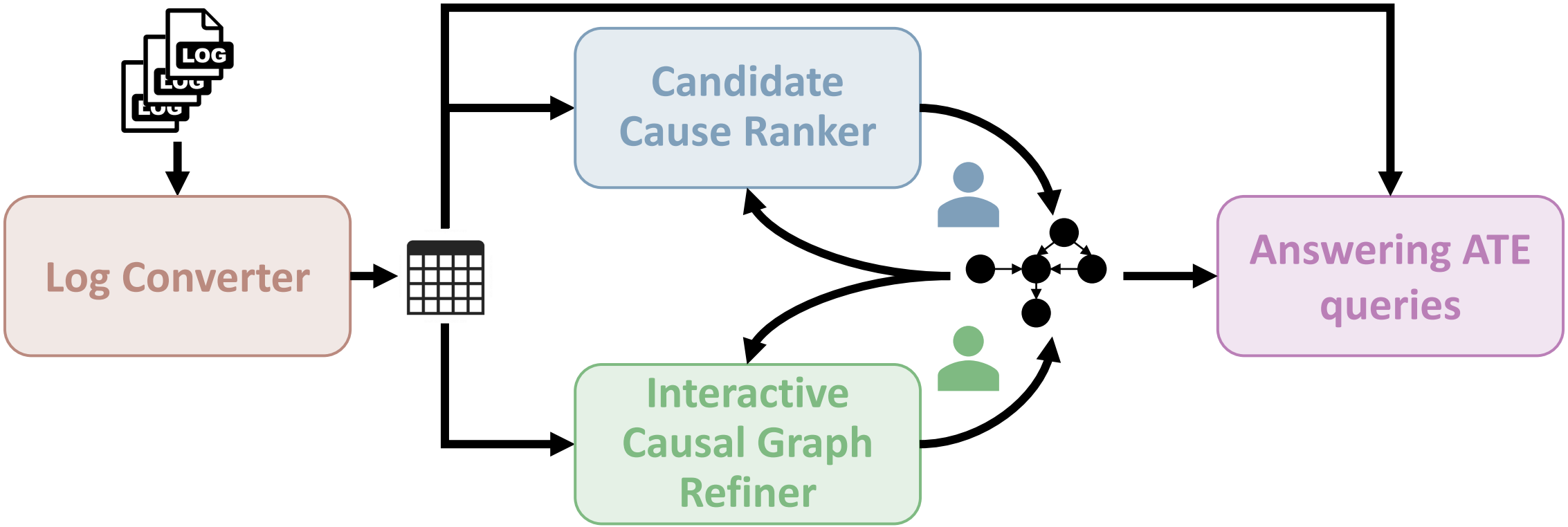
Alg.	Independence Test / Method	Result	Alg.	Scoring Function	Result
PC [93]	fisherz	✗	GIN [107]	kci	▲
	mv_fisherz	✗		hsic	▲
	mc_fisherz	✗	GRaSP [52]	CV_general	✓
	kci	▲		marginal_general	✗
	chisq	✓		CV_multi	✗
	gsq	●		marginal_multi	✗
	d_separation	✗		BIC	✗
FCI [94]	fisherz	✗		BDeu	✓
	kci	▲	GES [15]	CV_general	▲
	chisq	✓		marginal_general	✗
	gsq	●		CV_multi	▲
LiNGAM [90]		✗		marginal_multi	✗
				BIC	✗
Exact	dp [92]	✗		BIC_from_cov	✗
	astar [111]	✗		BDeu	✓

LOGos: a human-in-the-loop approach

- Fully automatic causal discovery did not work.
- Manually creating a causal graph from a set of variables V would require $O(|V|^2)$ judgments.
- A judgment for outputs whether an edge should exist and in which direction.
- Only solicit most important judgments in order of decreasing impact on the ATE.



Three main components work together



LOGos reflects our design principles

Scope modeling by intent

Only build the part of the causal graph that matters

Refine models using feedback

Solicit judgments about the presence of causal relationships

Prioritize effort by impact

Solicit judgments in decreasing order of ATE impact

Diagnosing Performance with LOGos

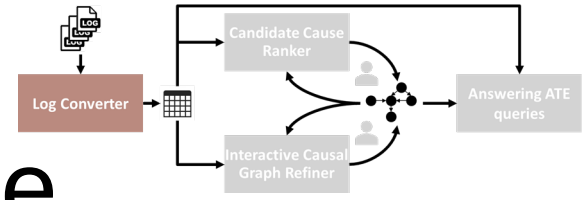
- Introduction
- **Technical Deep Dive**
- Results

Managing Performance with AutoSLO

- Introduction
- Technical Deep Dive
- Results

Synthesis & Research Outlook

- Case Study
- Concurrent Trends
- Future Research Directions

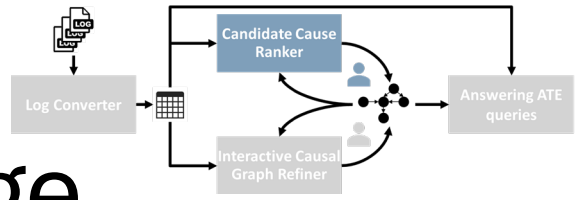


We must first obtain log data as a table

- Causal unit: user-defined unit of analysis.
- Aggregate variables within each causal unit.
- Pick aggregate function to maximize empirical entropy.
- Intuition: want causal units to “look different”.
- More details in the thesis.

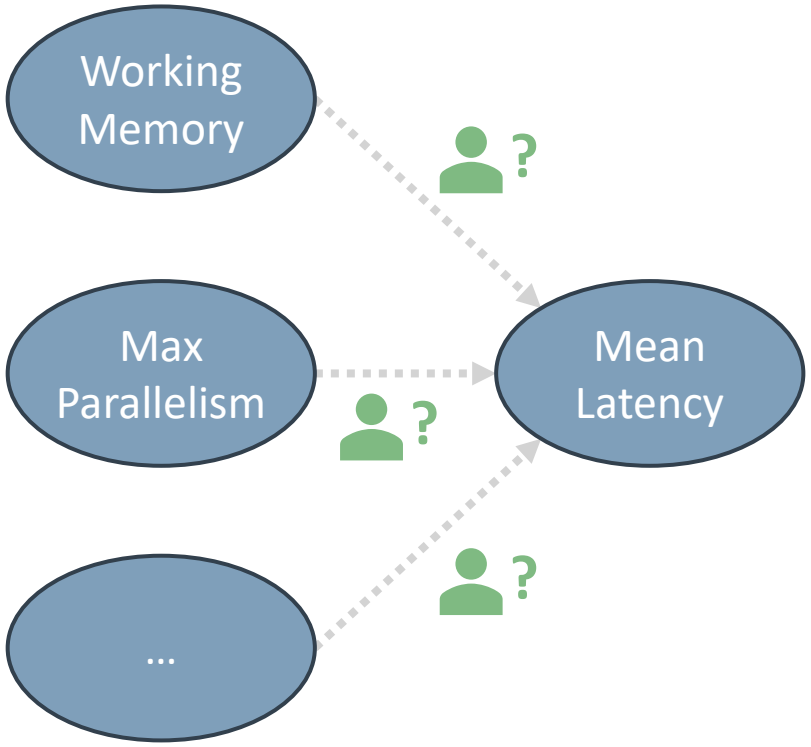
```
2023-11-06 16:34:40.811 EST [ 65495bf0.179a 3/2462 ] postgres@tpcds1 LOG:  statement: SET work_mem = 128;
2023-11-06 16:34:40.812 EST [ 65495bf0.179a 3/2463 ] postgres@tpcds1 LOG:  statement: SET max_parallel_workers = 1;
2023-11-06 16:34:40.812 EST [ 65495bf0.179a 3/2464 ] postgres@tpcds1 LOG:  statement: SET effective_cache_size = 262144;
2023-11-06 16:34:40.812 EST [ 65495bf0.179a 3/2465 ] postgres@tpcds1 LOG:  statement: SET maintenance_work_mem = 32768;
2023-11-06 16:34:40.813 EST [ 65495bf0.179a 3/2466 ] postgres@tpcds1 LOG:  statement: SET random_page_cost = 4;
2023-11-06 16:34:40.813 EST [ 65495bf0.179a 3/2467 ] postgres@tpcds1 LOG:  statement: SET seq_page_cost = 1;
2023-11-06 16:34:40.820 EST [ 65495bf0.179a 3/2468 ] postgres@tpcds1 LOG:  statement: -- Filename: query080.sql
2023-11-06 16:34:53.850 EST [ 65495bf0.179a 3/2468 ] postgres@tpcds1 LOG:  duration: 13029.853 ms
...
2023-11-07 08:15:48.440 EST [ 654a3884.49b7 3/3278 ] postgres@tpcds1 LOG:  statement: SET work_mem = 128;
2023-11-07 08:15:48.441 EST [ 654a3884.49b7 3/3279 ] postgres@tpcds1 LOG:  statement: SET max_parallel_workers = 1;
2023-11-07 08:15:48.441 EST [ 654a3884.49b7 3/3280 ] postgres@tpcds1 LOG:  statement: SET effective_cache_size = 262144;
2023-11-07 08:15:48.441 EST [ 654a3884.49b7 3/3281 ] postgres@tpcds1 LOG:  statement: SET maintenance_work_mem = 65536;
2023-11-07 08:15:48.442 EST [ 654a3884.49b7 3/3282 ] postgres@tpcds1 LOG:  statement: SET random_page_cost = 4;
2023-11-07 08:15:48.442 EST [ 654a3884.49b7 3/3283 ] postgres@tpcds1 LOG:  statement: SET seq_page_cost = 1;
2023-11-07 08:15:48.443 EST [ 654a3884.49b7 3/3284 ] postgres@tpcds1 LOG:  statement: -- Filename: query080.sql
2023-11-07 08:15:49.446 EST [ 654a3884.49b7 3/3284 ] postgres@tpcds1 LOG:  duration: 1003.228 ms
...
```

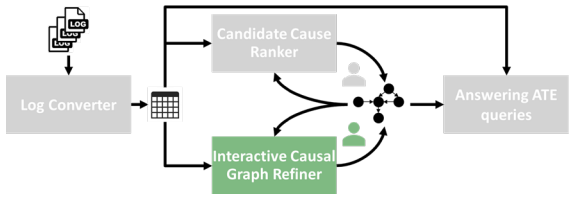
Session ID	Mode work_mem	...	Mean duration
65495bf0.179a	128	...	...
654a3884.49b7	128	...	...
...	...	...	..



LOGos ranks candidate causes to judge

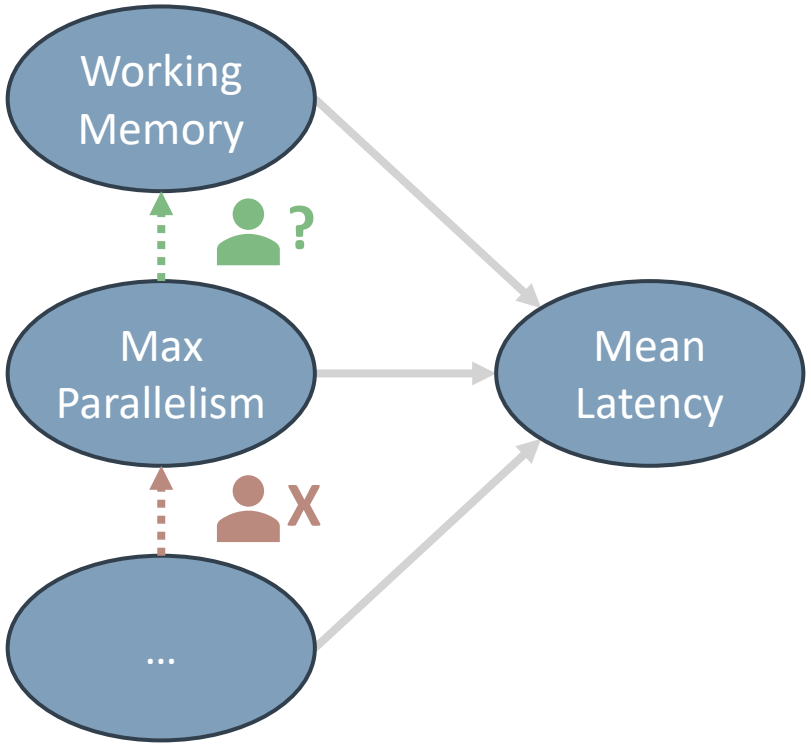
- Specify a target variable O – e.g. mean latency.
- Goal: Find candidate causes of O and solicit user judgments about them.
- Intuition: Find the few reliably related variables
- Algorithm sketch:
  - Sparsely regress O using LASSO to find candidate graph parents.
  - Compute pairwise regression coefficients and sort by p-values.
  - Prompt user for judgments in that order.

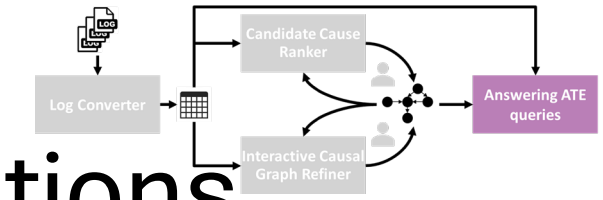




The graph can be refined interactively

- Goal: Which edge edit would most impact the ATE of interest? Have the user judge it.
- Intuition: Avoid judging graph edits irrelevant to the ATE of interest.
- Algorithm sketch:
  - From the current graph, recover the *adjustment set*.
  - For each single-variable adjustment set change, map it to graph edge edits.
  - Rank these alternatives by their impact on the ATE.



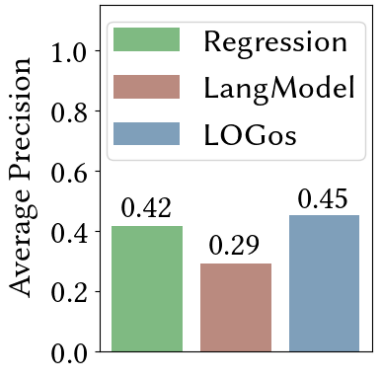


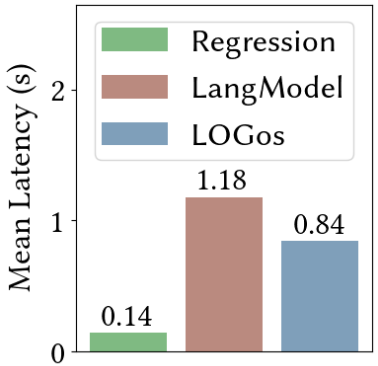
We evaluate LOGos across log collections

- No open-source log dataset with labels of what went wrong.
- We construct three datasets to evaluate on:
  - PostgreSQL (Real-world):
    - Run TPC-DS for various knob values.
  - Proprietary (Real-world, post-processed):
    - Tweak HTTP log from real application to fail with some probability.
  - XYZ (Synthetic):
    - Synthetic variables X, Y, Z among many, noisy others.

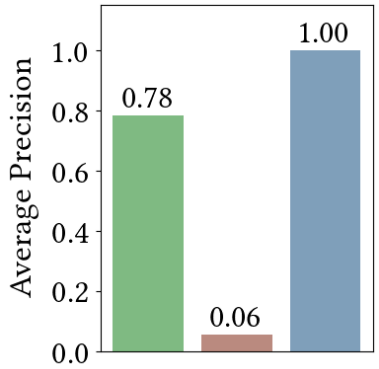
LOGos ranks the true causes highly and is fast

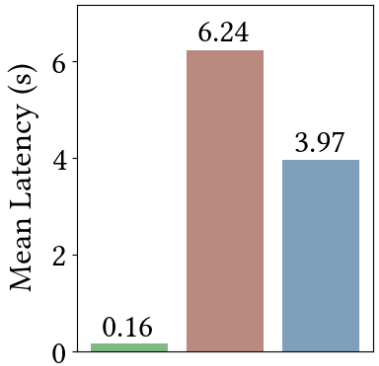
PostgreSQL



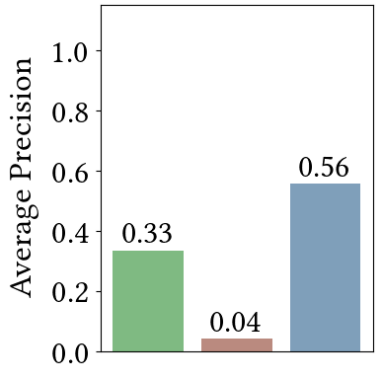


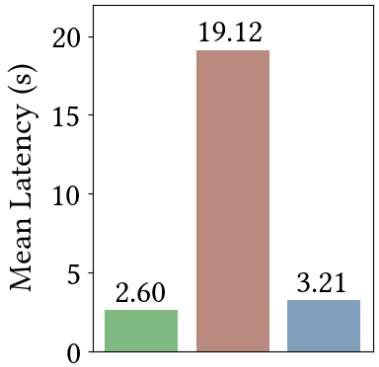
Proprietary



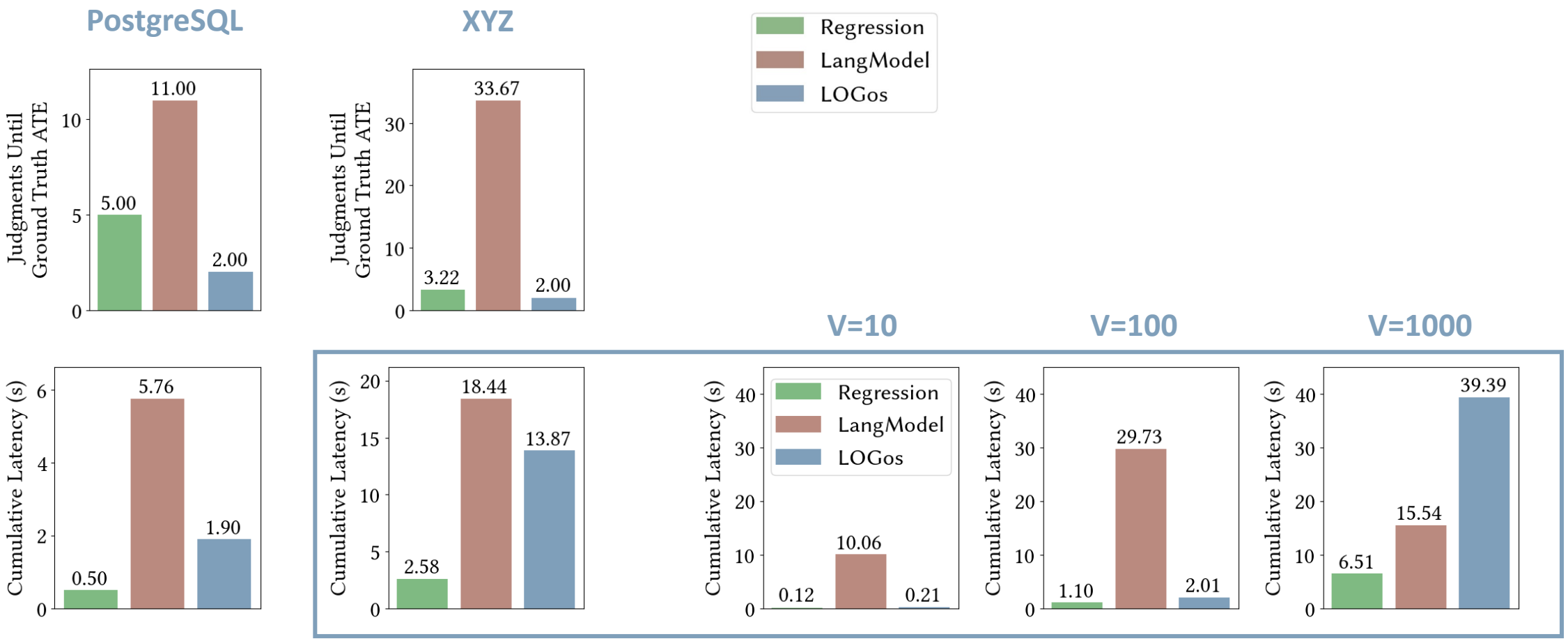


XYZ





Graph refinement reduces # judgments



Key Takeaway

LOGos can help engineers draw principled causal conclusions from logs faster

Diagnosing Performance with LOGos

- Introduction
- Technical Deep Dive
- Results

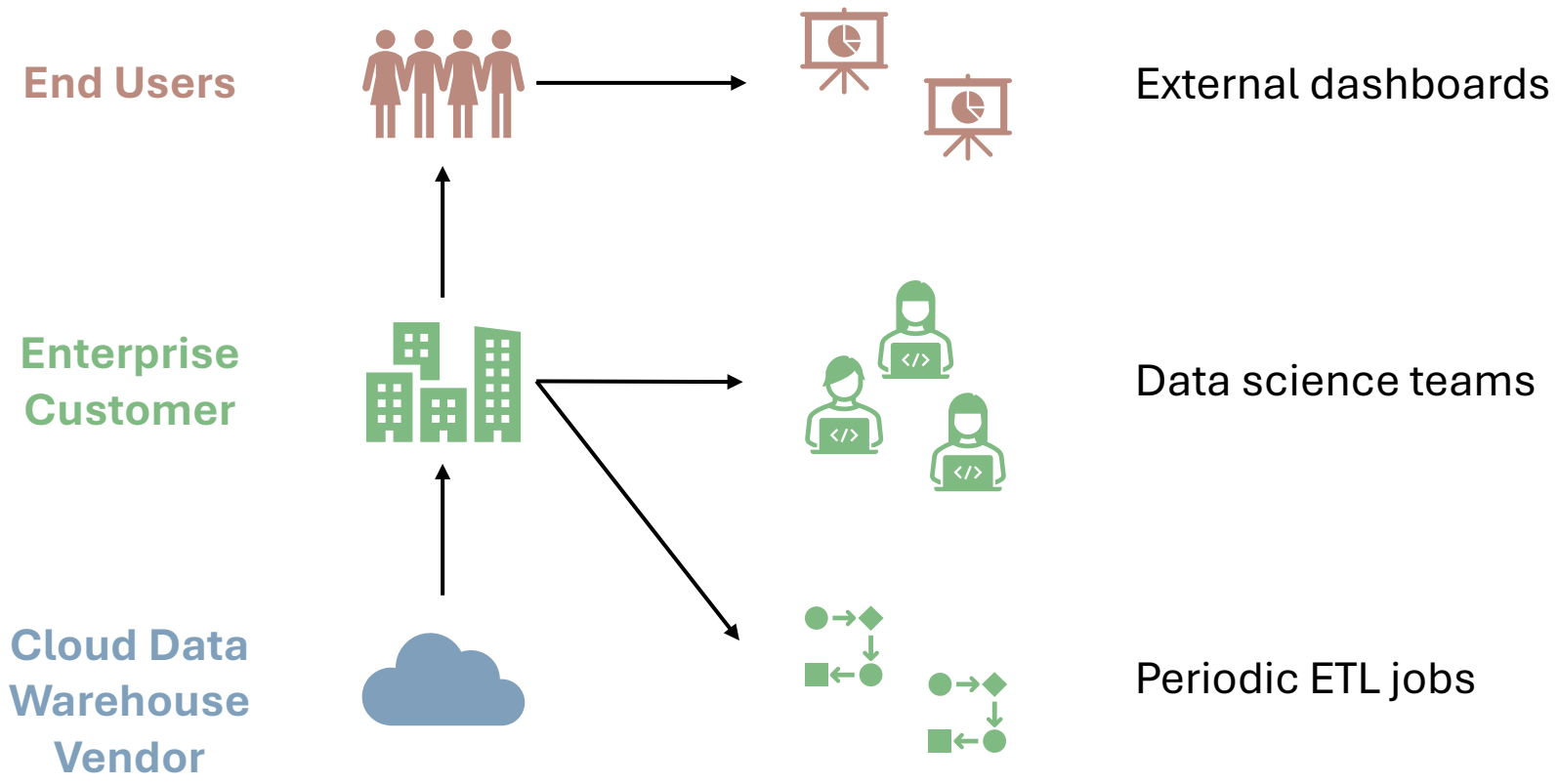
Managing Performance with AutoSLO

- **Introduction**
- Technical Deep Dive
- Results

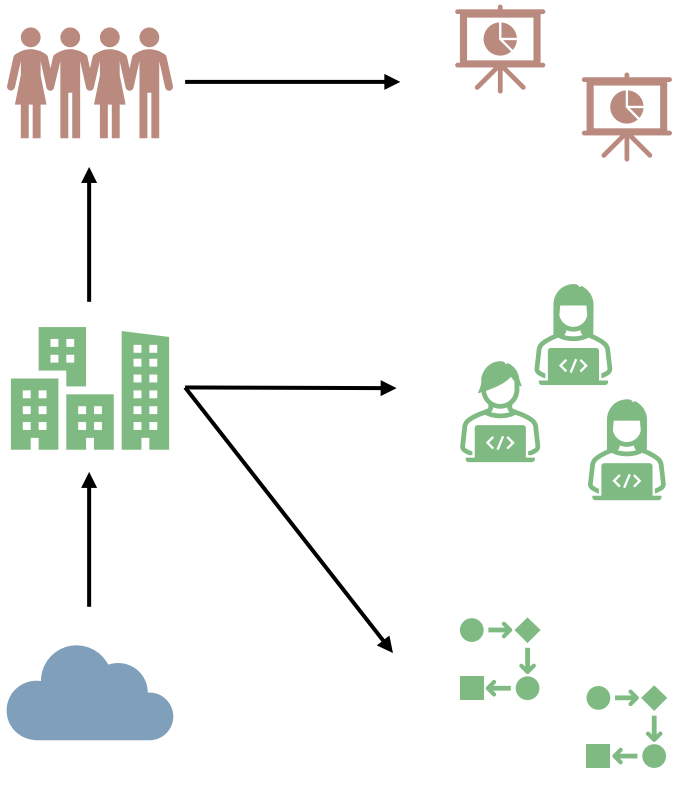
Synthesis & Research Outlook

- Case Study
- Concurrent Trends
- Future Research Directions

Cloud data warehouses see many workloads



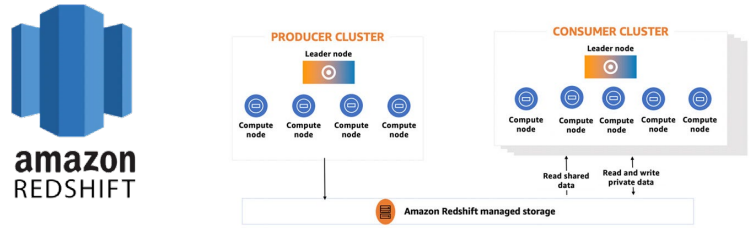
Each workload has a different latency SLO

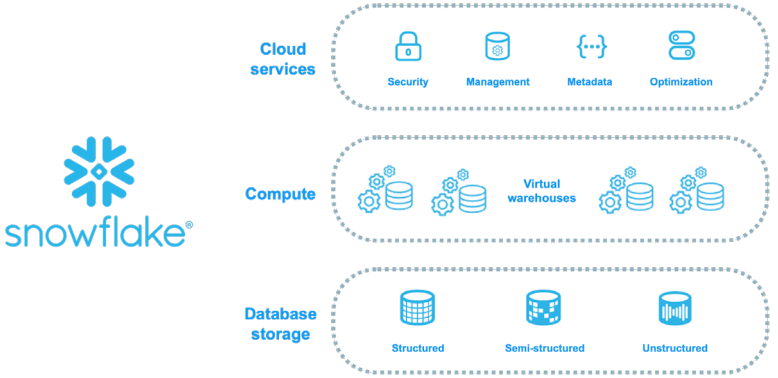


External dashboards	A few seconds
Data science teams	Seconds to minutes
Periodic ETL jobs	Minutes to hours

Compute-storage separation can help isolate

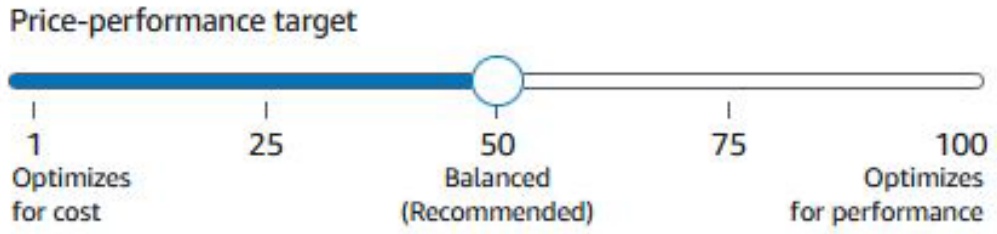
- Modern cloud data warehouses let multiple compute clusters query the same data.
- Offers a crude way to meet latency SLOs through performance isolation:
  - Map each workload to a separate cluster.
  - Manage each of these clusters separately.





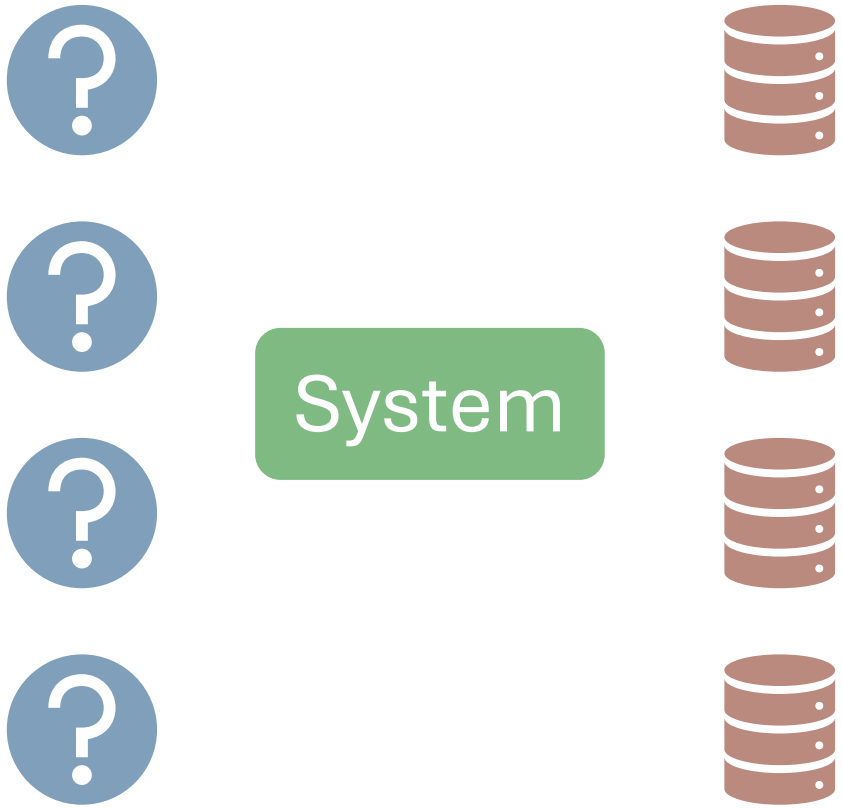
Meeting SLOs using separation is only implicit

- Per-workload clusters must be tuned independently until performance is acceptable.
  - Too small: SLO violations
  - Too large: unnecessary costs
- In practice, there is a level where SLO violation rate is “low enough”
  - Will not keep spending to lower it further.
- Current auto-tuning solutions are not driven by explicit SLOs.
- Even if we tune each cluster, intra-workload contention is not managed.



What if we don’t silo each workload?

- “Workloads” are not a necessary intermediate level. In practice:
  - There is a stream of queries
  - Each query has a latency SLO
  - The SLO may be inherited from somewhere, but it does not matter.
- On the other end, we can have one or more clusters running.
- We can then individually route each query to meet its SLO.



Managing this setup requires 3 key behaviors

- Some workload elements are repeating/predictable; the system should plan resources to execute them.
- Online variability cannot be eliminated; the system should adjust resources to it.
- Each query is now routed individually; the system can and should react to concurrent load to limit SLO violations.

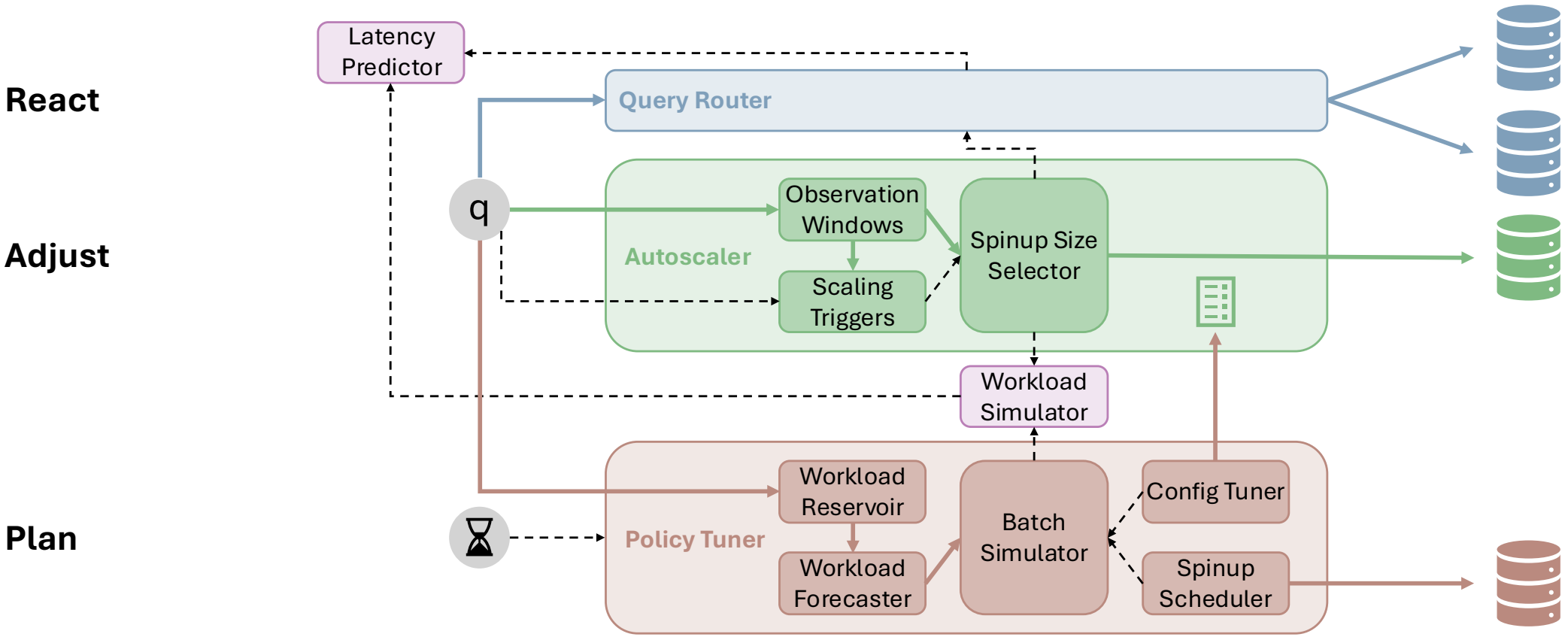
Key Desired Behavior	**Plan** resources based on history	**Adjust** resources based on demand	**React** to concurrent live load
Timescale	≈ 24 hrs	≈ 5 mins	≈ 1 s

No prior work offers all 3 key behaviors

- Some works only plan for a given workload but cannot handle deviations.
- Other works can only adjust online, but SLO violations can accrue by the time new resources spin up.
- Across the board, no work can react, because no work treats SLOs as a first-class input.

Key Desired Behavior	**Plan** resources based on history	**Adjust** resources based on demand	**React** to concurrent live load
Timescale	≈ 24 hrs	≈ 5 mins	≈ 1 s
ResTune	✓ *Replay-based knob tuning on replica*	✗	✗
WiseDB	✓ *Tree model for fixed per-template routing*	✗	✗
Das et al.	✗	✓ *Container sizing w/ token bucket*	✗
SLAOrches-trator	✗	✓ *Utilization- or simulation-based pool resizing*	✗
BRAD	✗	✓ *Blueprint beam search*	✗
Auto-WLM	✗	✓ *Queueing-based horizontal scaling*	✗
RAIS	✓ *Multi-forecast parameter tuning*	✓ *Queueing-based horizontal scaling*	✗

AutoSLO assigns a key component to each



AutoSLO reflects our design principles

Scope modeling by intent

Gray-box latency predictor is good enough if SLOs are met

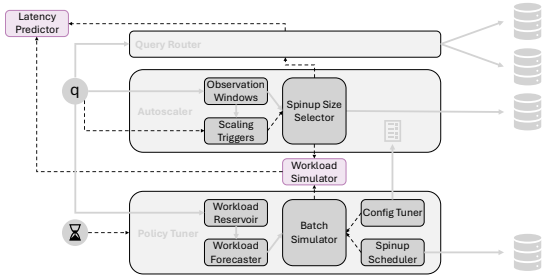
Refine models using feedback

Do not only rely on planning; adjust and react online

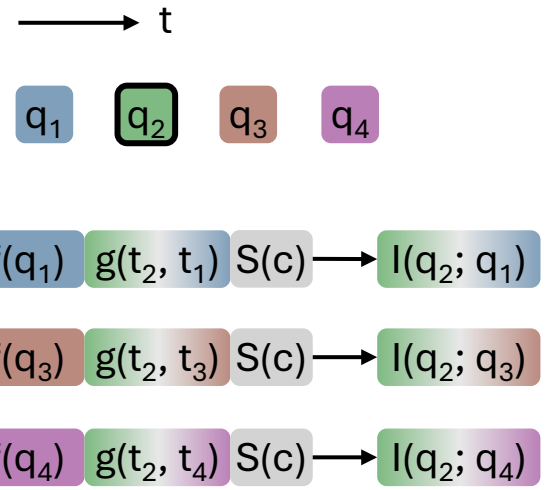
Prioritize effort by impact

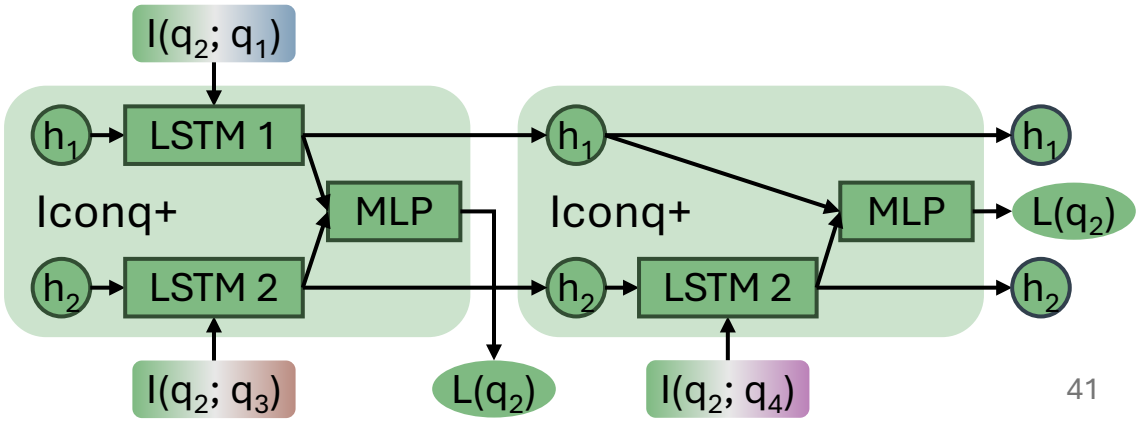
Match component fidelity and latency to each timescale

The Latency Predictor implements Iconq+ based on Iconq

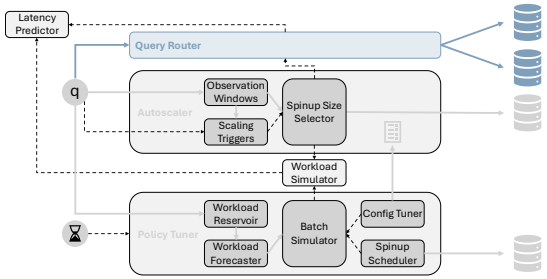


- Original Iconq:
  - Featurize each query individually
  - Create joint vectors with each neighbor.
  - Ingest through forward/backward LSTM.
- Iconq+ includes 3 modifications:
  - Cluster size features.
  - Training on censored observations.
  - Incremental predictions through forward neighbor ingestion.

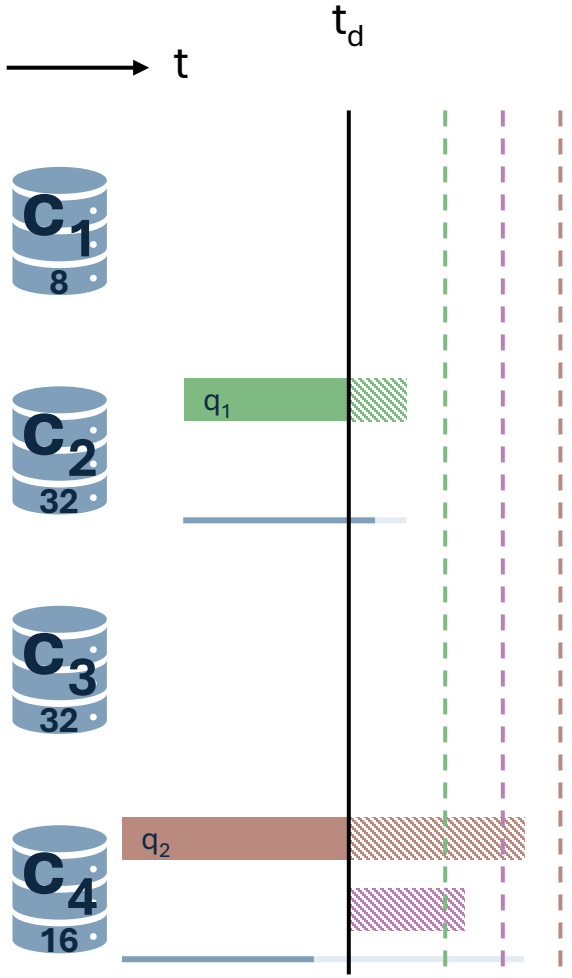




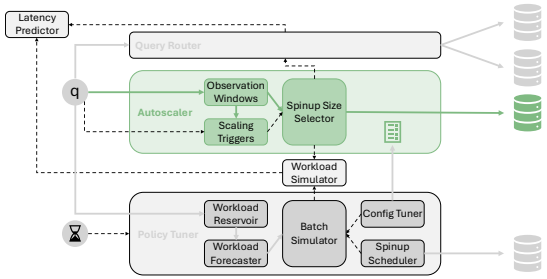
The Query Router balances SLO risk and cost



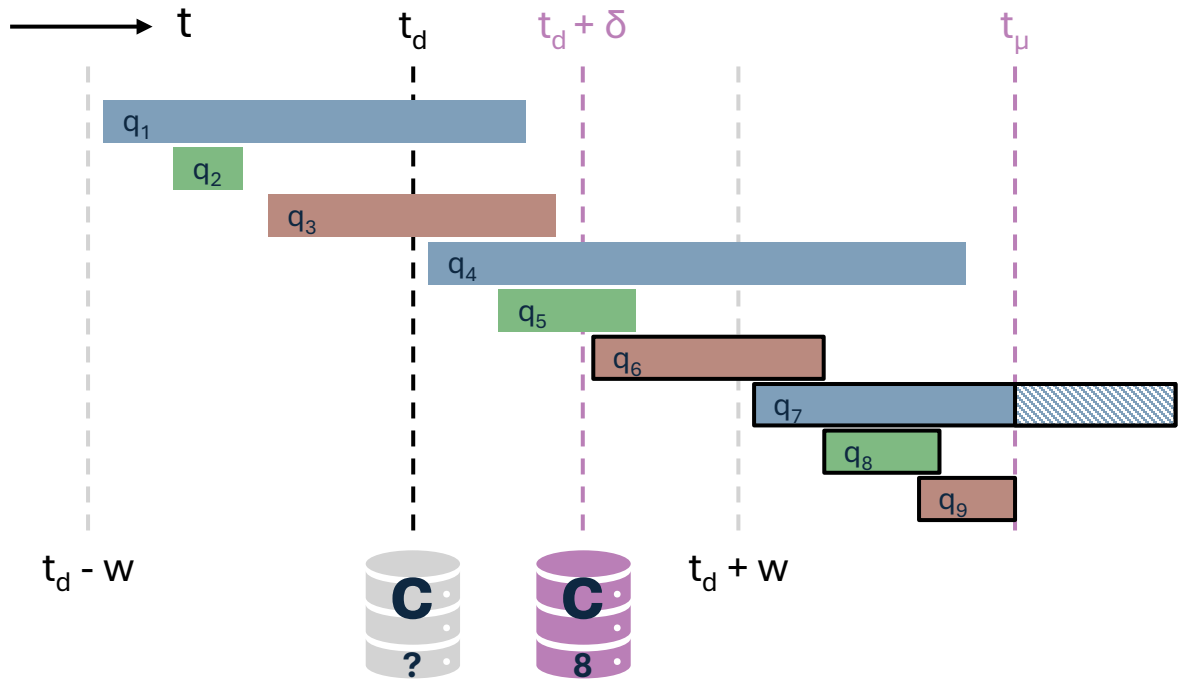
- Assume an incoming query q.
- Goal: Route q while meeting SLOs.
- Intuition: Select based on fewest SLO violations, breaking ties by cost.
- Algorithm sketch:
  - Per active cluster, hypothesize placement and calculate:
    - SLO adherence for new query.
    - SLO adherence for old queries.
    - Cost impact on the cluster.



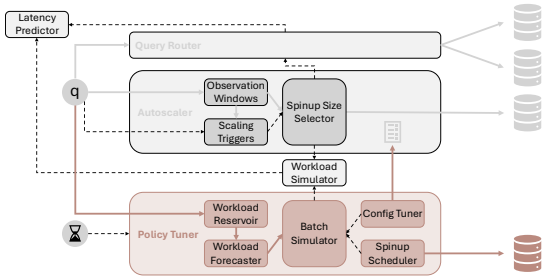
The Autoscaler relies on forward simulation



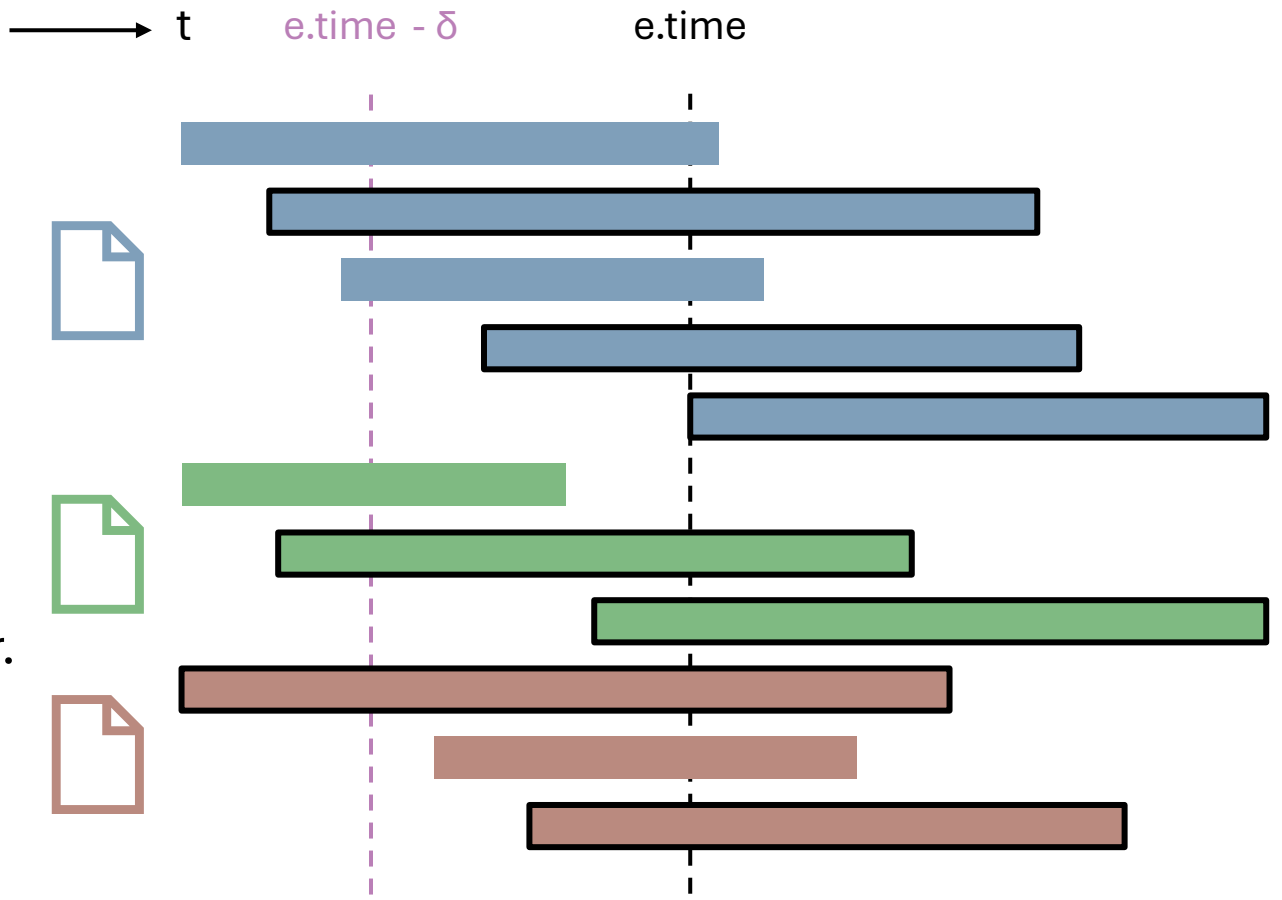
- Goal: Decide what size cluster to spin up.
- Intuition: Hypothesize different sizes, run simulation forward and compute benefit.
- Algorithm Sketch:
  - Collect rolling recent arrival window.
  - Hypothesize a new cluster request.
  - Simulate arrival window until cluster is expected to be available.
  - Continue simulating until μ queries that saw the new cluster have finished executing.
  - Also account for the SLO adherence/cost of remaining queries at that point.
  - Repeat and pick the cluster size with the fewest SLO violations, breaking ties by cost.



The Policy Tuner plans from multiple forecasts



- Goal: Schedule cluster spinups likely to be needed, to avoid reacting online.
- Intuition: Use several forecasts to reliably identify contention peaks.
- Algorithm sketch:
  - Simulate multiple forecasts.
  - Find when at least k (e.g. 3) have a high SLO violation rate for running queries (e.g. at least 0.6).
  - Schedule a cluster spinup shortly earlier.
  - Pick size to avoid the most violations (break ties by cost).
  - Then also optimize Autoscaler settings.



Diagnosing Performance with LOGos

- Introduction
- Technical Deep Dive
- Results

Managing Performance with AutoSLO

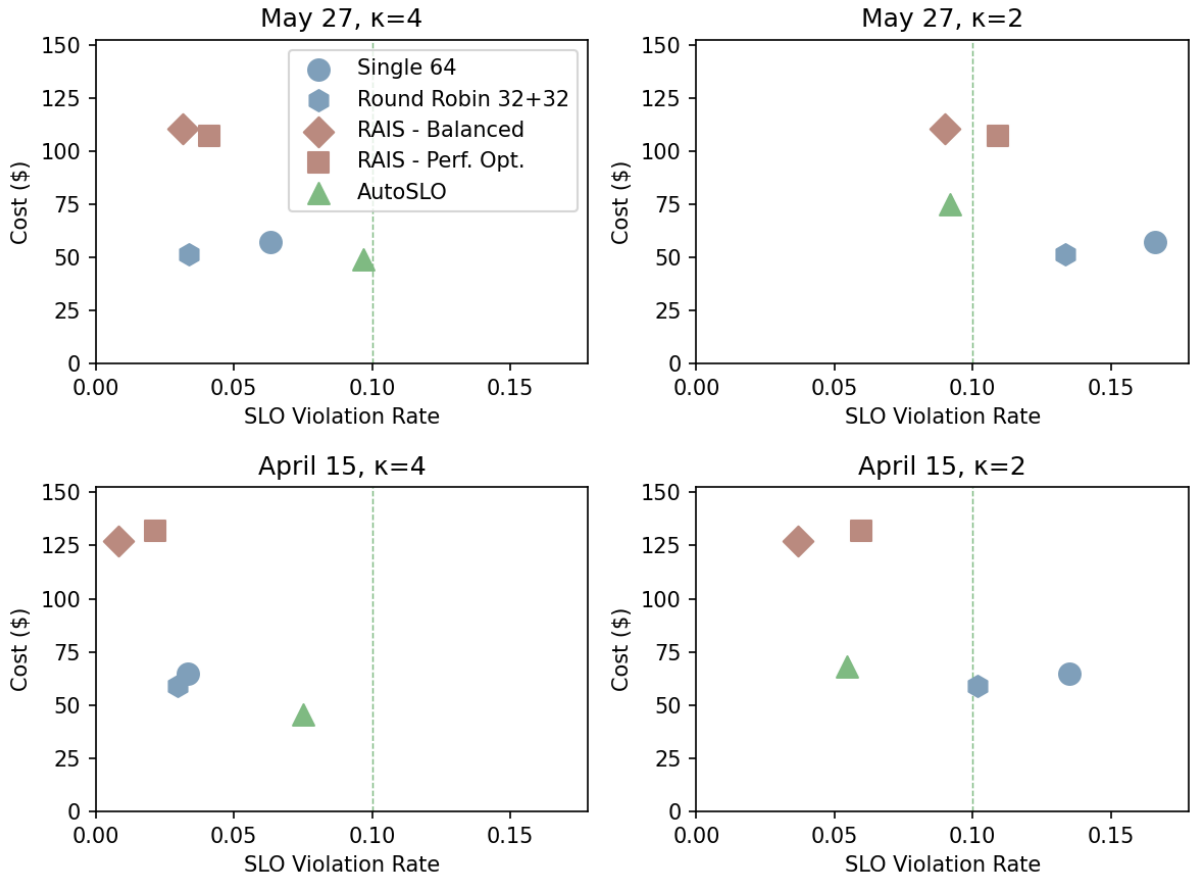
- Introduction
- Technical Deep Dive
- **Results**

Synthesis & Research Outlook

- Case Study
- Concurrent Trends
- Future Research Directions

AutoSLO effectively meets SLOs end-to-end

- Manual single- or dual-cluster configurations are cheap but ineffective as SLOs tighten.
- Current autoscaling with RAIS is ~2x expensive.
- Meets SLOs more often, but cannot control it when κ=2.
- AutoSLO is always the cheapest method that meets the SLO target.
- It reduces cost by a mean of 26.4% vs. per-scenario next-best.
- Compared to the only baseline that always meets the target, it reduces cost by a mean of 49.6%.

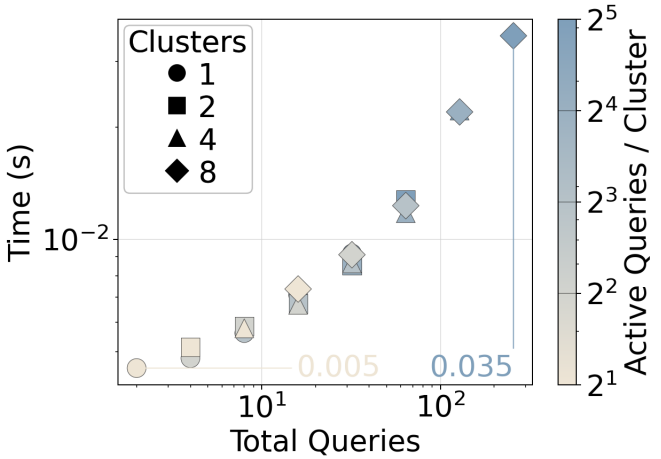


Each component is individually effective

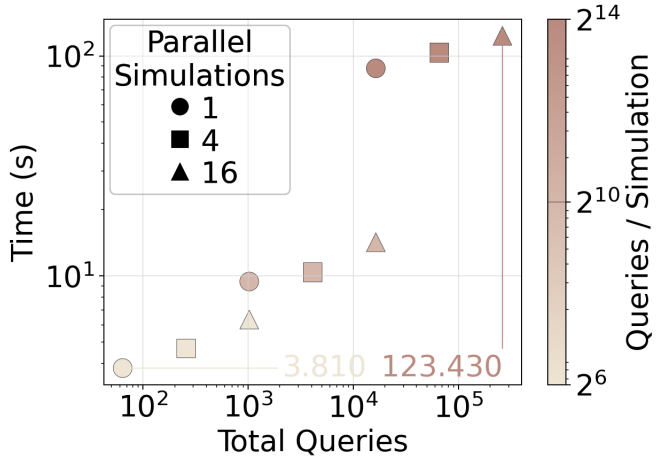
- Policy Tuner
  One day of workload history reduces SLO violation rate by a mean of 44.6%.
- Query Router
  Reduces SLO violation rate by a mean of 47.8% while also lowering cost.
- Spinup Size Selector
  Reduces SLO violation rate by a mean of 93.7% while increasing cost by <12%.
- Spinup Trigger
  Reduces SLO violation rate by a mean of 54.6% for fixed Spinup Size Selector.
- More details in the thesis.

Each component is efficient for its timescale

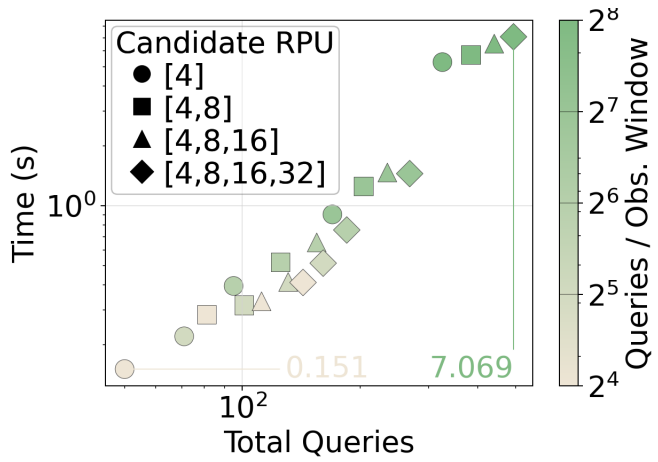
Route



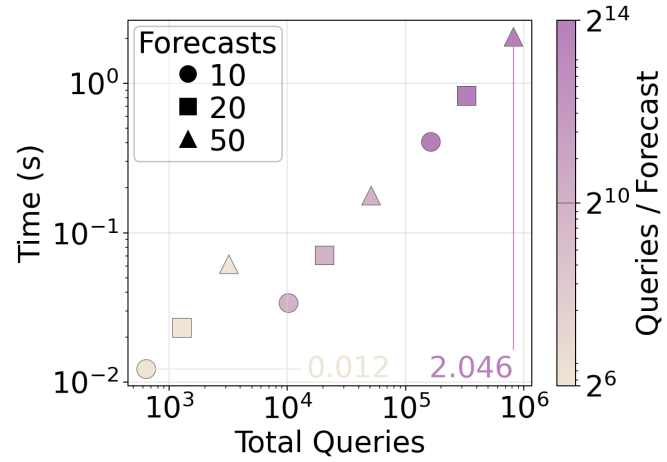
SimulateBatch



FindBestSpinupSize

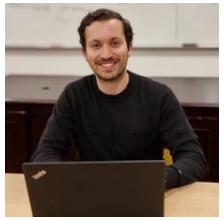


FindGoodSpinupTime



Key Takeaway

AutoSLO can scale resources and route queries to meet latency SLOs cheaply



Diagnosing Performance with LOGos

- Introduction
- Technical Deep Dive
- Results

Managing Performance with AutoSLO

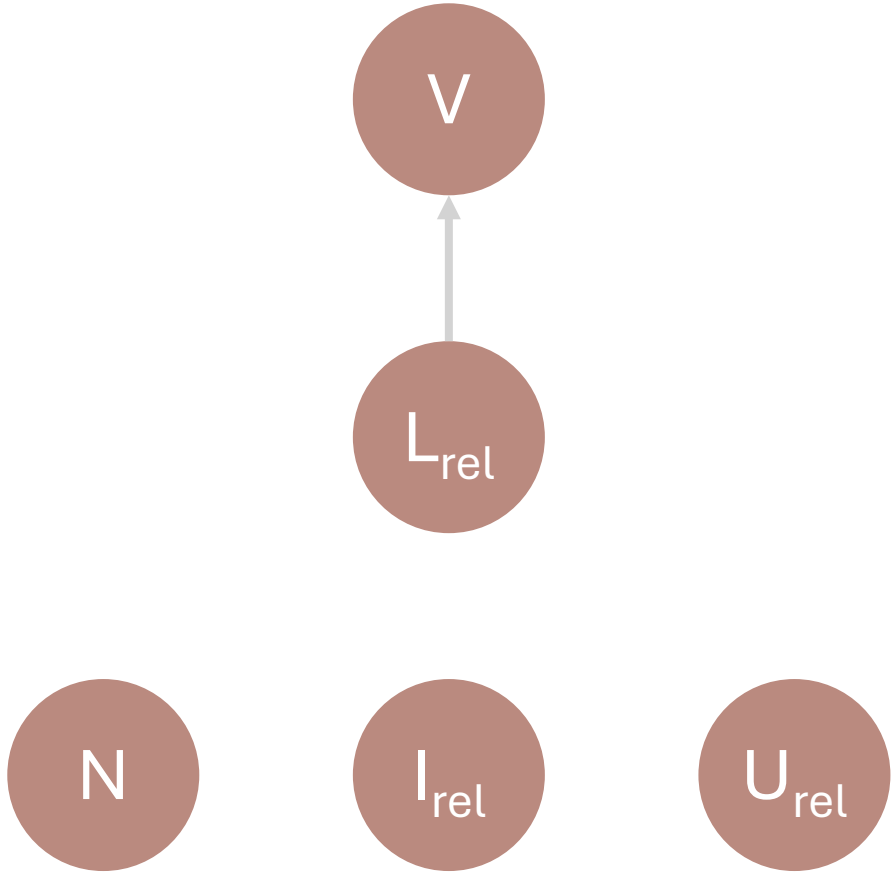
- Introduction
- Technical Deep Dive
- Results

Synthesis & Research Outlook

- **Case Study**
- Concurrent Trends
- Future Research Directions

Applying LOGos to logs from AutoSLO

- Take the logs from the April 15, κ=4 run from earlier: 17,399 records, 1,356 queries.
- What causes SLO violations?
  - …high latency-to-SLO ratio
  - Functional dependencies
- Okay, what causes *that*?
  - # neighbors at route time
  - Initial latency prediction over SLO
  - Updated latency prediction over SLO



The candidate causes reflect AutoSLO's design

- # neighbors at route time

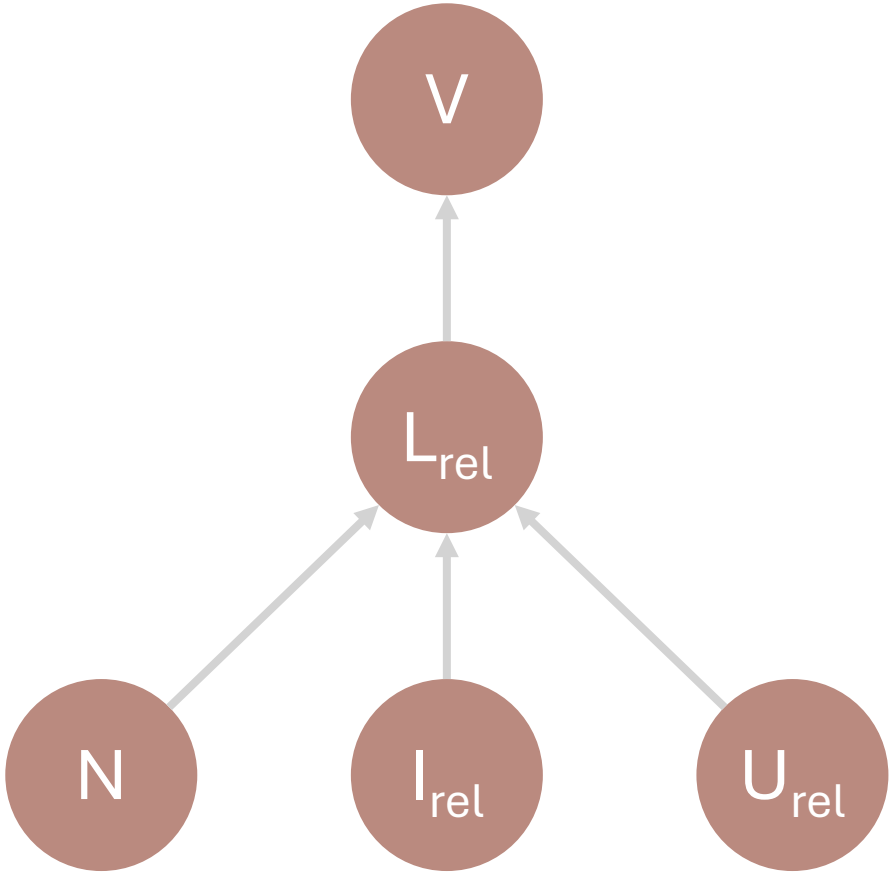
  Directly cause contention that influences SLO adherence.

- Initial latency prediction over SLO

  Determines attractiveness of the cluster for future routing.

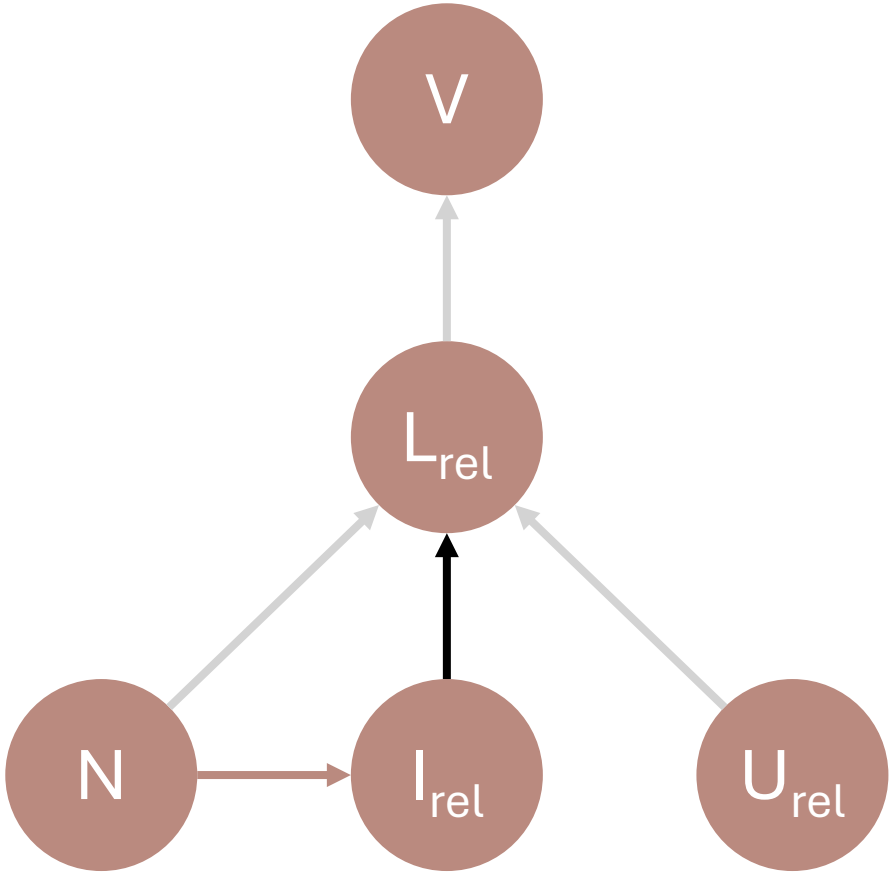
- Updated latency prediction over SLO

  Influential in the same way as the initial prediction.



The dual influence of routing-time neighbors

- We use LOGos to calculate ATE(I_{rel}, L_{rel}) -> small value > 0.
- We then use LOGos to investigate confounders.
- It asks us: *does N cause I_{rel}?*
- Neighbors at route time indeed affect the initial latency prediction using Iconq+.
- Adjusting for this dramatically raises the ATE(I_{rel}, L_{rel}).
- We remove the feedback loop that discourages routing future queries here.



Small case study, but offers three sanity checks

- Among thousands of log messages, LOGos quickly narrows it down to a few salient variables.
- Variables around routing decisions play the most direct role in SLO violations.
- Contention is a strong confounding factor, underscoring the importance of a contention-aware Latency Predictor.

Diagnosing Performance with LOGos

- Introduction
- Technical Deep Dive
- Results

Managing Performance with AutoSLO

- Introduction
- Technical Deep Dive
- Results

Synthesis & Research Outlook

- Case Study
- **Concurrent Trends**
- Future Research Directions

Expanding performance diagnosis

Broader coverage

TracEx

Analyze and compare execution traces across different DBMSes.

RCRank

Use multi-modal evidence (e.g. CPU and memory utilization, query plans) to diagnose slow queries.

DBDoctor

Use eBPF to collect event-driven metric observations.

LOGos

Flexibly use log variables and their aggregates, not just templates.

More robust detection

BALANCE

Bayesian multi-collinear feature selection over operational timeseries.

RT-Log

Transferable log anomaly detection by learning environment-invariant log representations.

Yan et al.

Siamese Discrepancy Network learns to identify rarely observed anomalies.

LOGos

Human-in-the-loop causal inference for model pieces that actually matter.

Abstracting away performance management

More decoupling & abstraction

Amazon Aurora DSQL

From managed Postgres, to autoscaling, to full resource disaggregation.

Virtual Database Engines

Full DBMS abstraction; only specify required dialect, performance and freshness.

AutoSLO

Leverages decoupling to optimize over a multi-cluster data warehouse.

Cost-performance inputs

Zhang et al.

"Cost intelligence" and the challenges for realizing it.

Tao

Decompose plans into "tasklets", schedule them for SLO compliance.

Accordion

Expose intra-query parallelism as a tunable knob during execution.

AutoSLO

Elevates quantitative latency SLOs to first-class inputs and objectives.

Integrating AI-driven reasoning

Diagnostic tools

Panda, D-Bot, Andromeda

Integrate public documentation and bug reports with scenario-specific signals (e.g. telemetry) to compose diagnoses.

DBAIOps

Create a knowledge graph, combine it with anomaly detection models & LLM reasoning.

SysInsight

Also use statically analyzed source code.

LOGos

Interactive mechanism for extracting causal conclusions; AI-driven systems can also apply it to the information they are using.

Tuning tools

GPTuner

Use LLMs to build search space from documentation; navigate with Bayesian methods.

λ-Tune

Use LLMs to generate the best set of configurations to compare in practice.

AgentTune

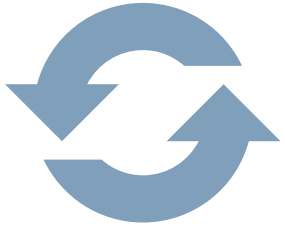
Decompose knob tuning into subtasks, assign each to a sub-agent.

AutoSLO

Efficient SLO control loop; can also become part of AI-designed or AI-operated systems.

Maintaining diagnostic representations online

Deal with selection bias in log generation



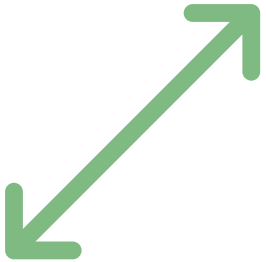
Maintain the causal graph online

Integrate multi-modal signals/metrics

Meeting diverse objectives

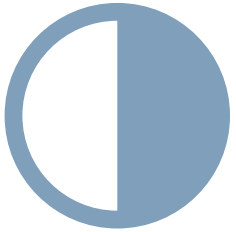
Support different SLO types

Suggest and refine SLOs



Expand the action space

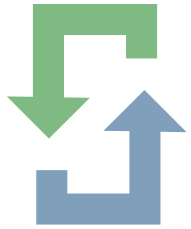
Operating in uncertain environments



Support conditional ATEs

Quantify and leverage uncertainty

Integrating causal diagnosis and management

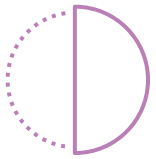


Make workload management causal

Inform AutoSLO design using causal conclusions

Refine the causal model while meeting SLOs

Diagnosing and Managing Performance in Data Systems



Scope modeling by intent

Refine models using feedback

Prioritize effort by impact

LOGos can help engineers draw principled causal conclusions from logs faster

AutoSLO can scale resources and route queries to meet latency SLOs cheaply