

Next Steps for Learned Query Optimization

Ryan Marcus
University of Pennsylvania

Slides: <https://rm.cab/nedb26>

This Talk

- Why **learn** a query optimizer?
- Lessons learned **deploying learned QOs**
- **LimeQO**, an offline learned query optimizer
- The end of RL for QO?

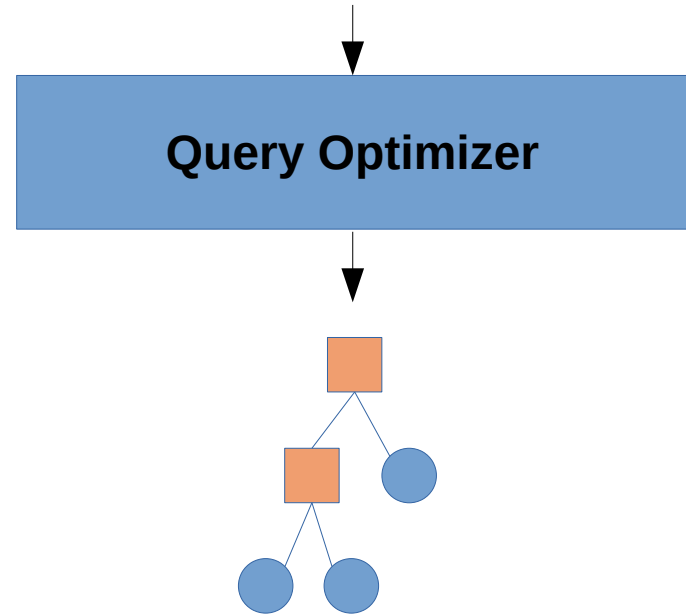
This Talk

- Why **learn** a query optimizer?
- Lessons learned **deploying learned QOs**
- **LimeQO**, an offline learned query optimizer
- The end of RL for QO?

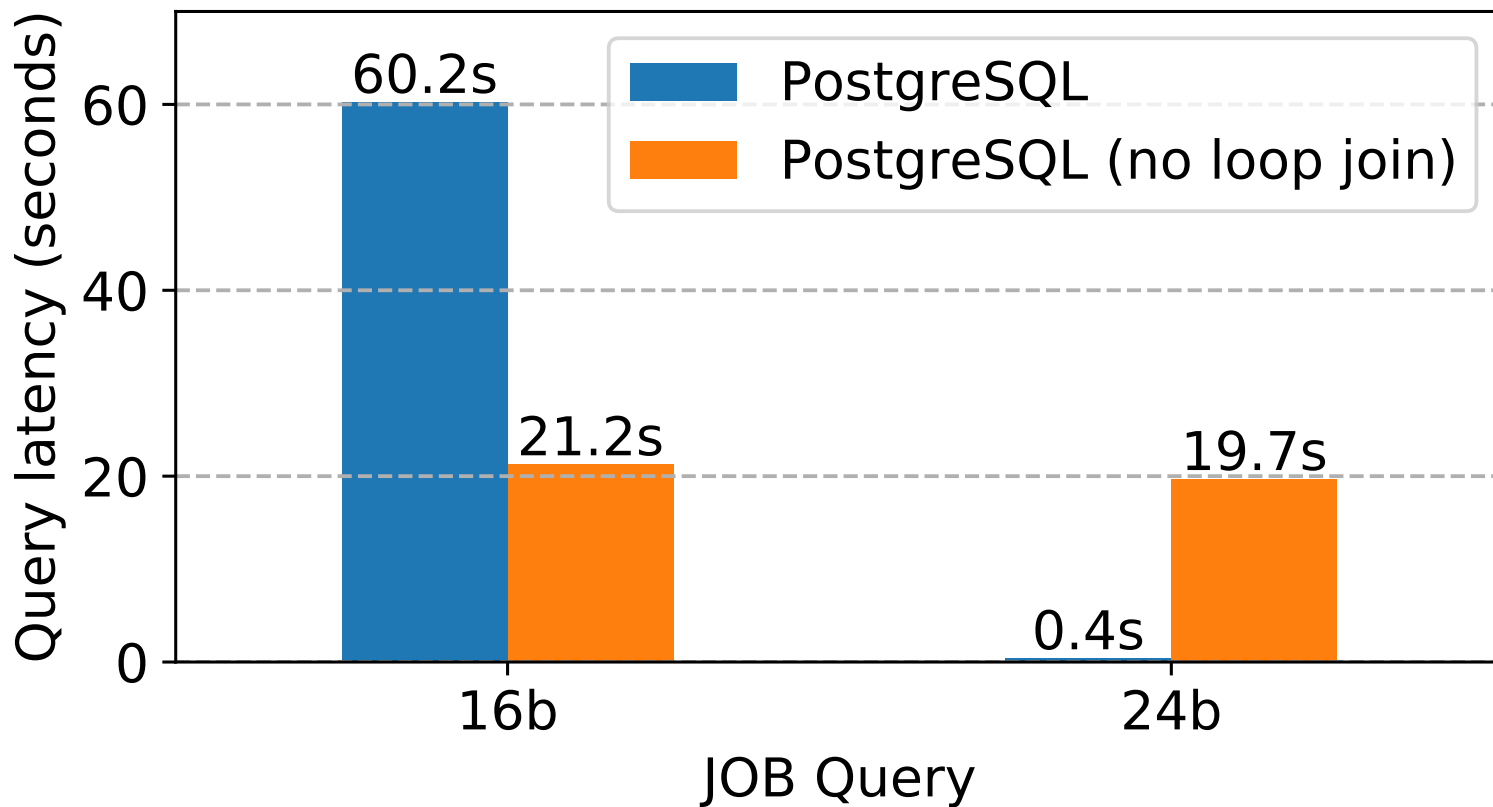
QO is a huge effort

- Transform SQL into a query plan
- 42K LOC in PG12
- 1M+ SQL Server
- 45-55 FTEs, Oracle (~ \$5mil/year)

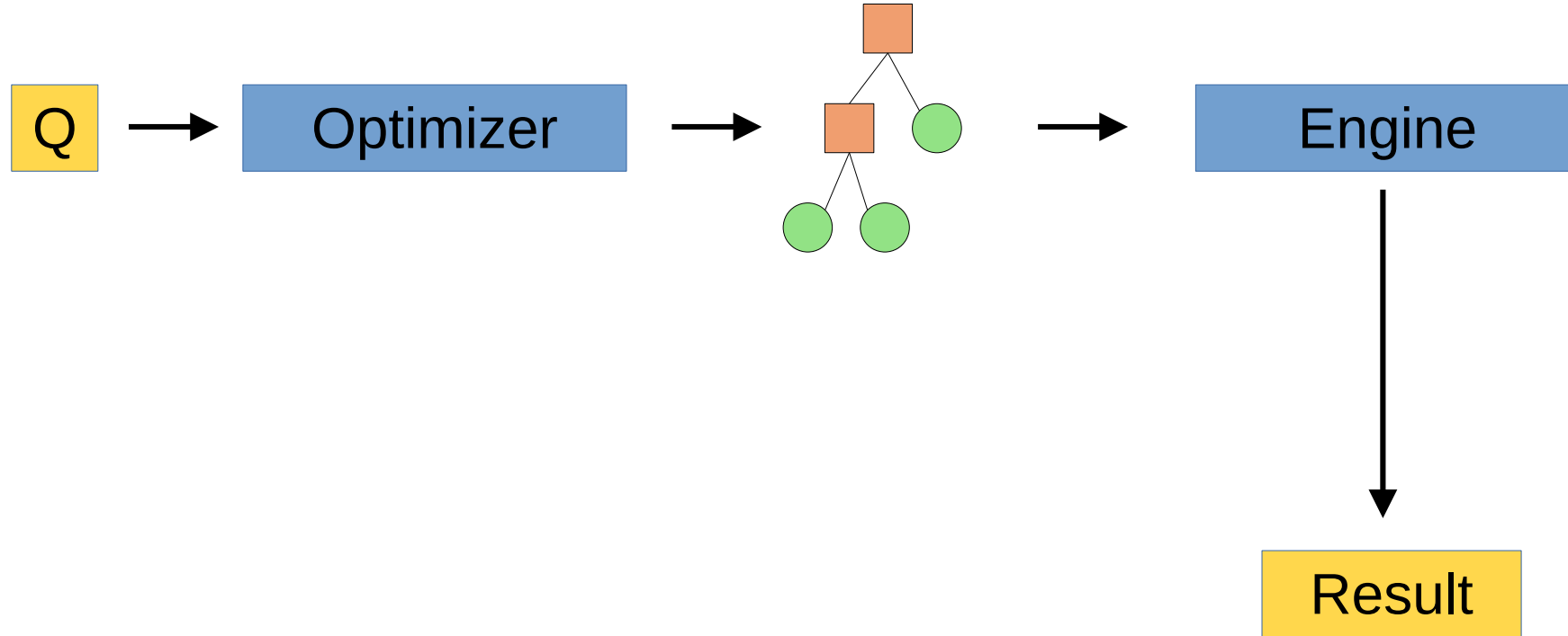
```
SELECT *  
FROM t1, t2 WHERE...
```



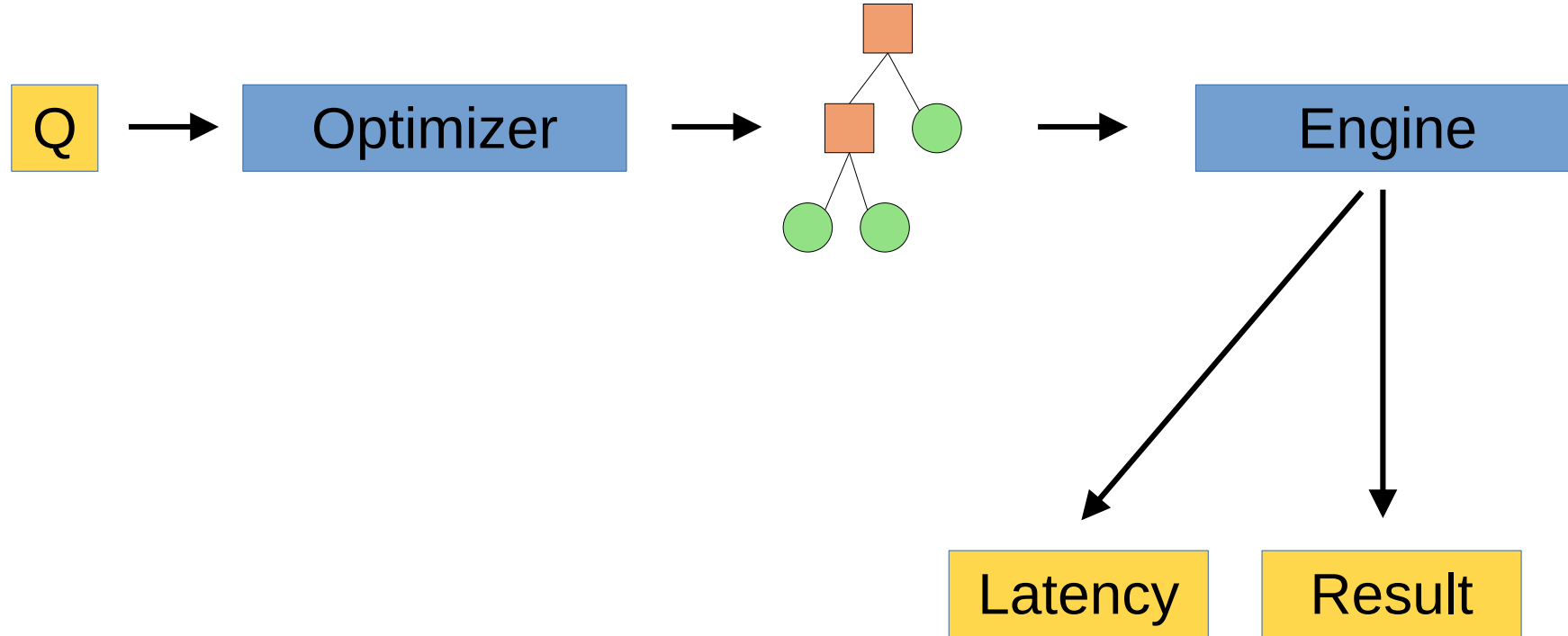
QOs aren't that good



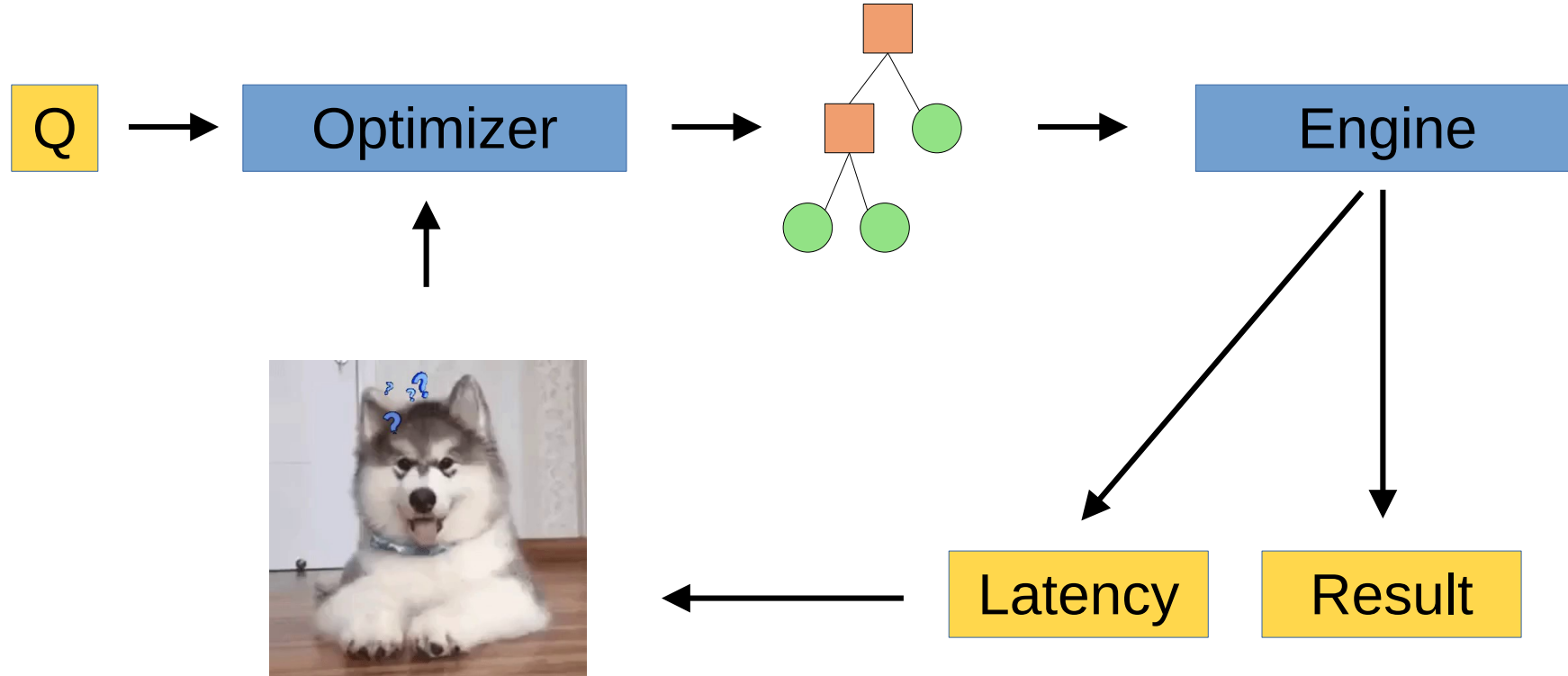
QOs are leaving info on the table



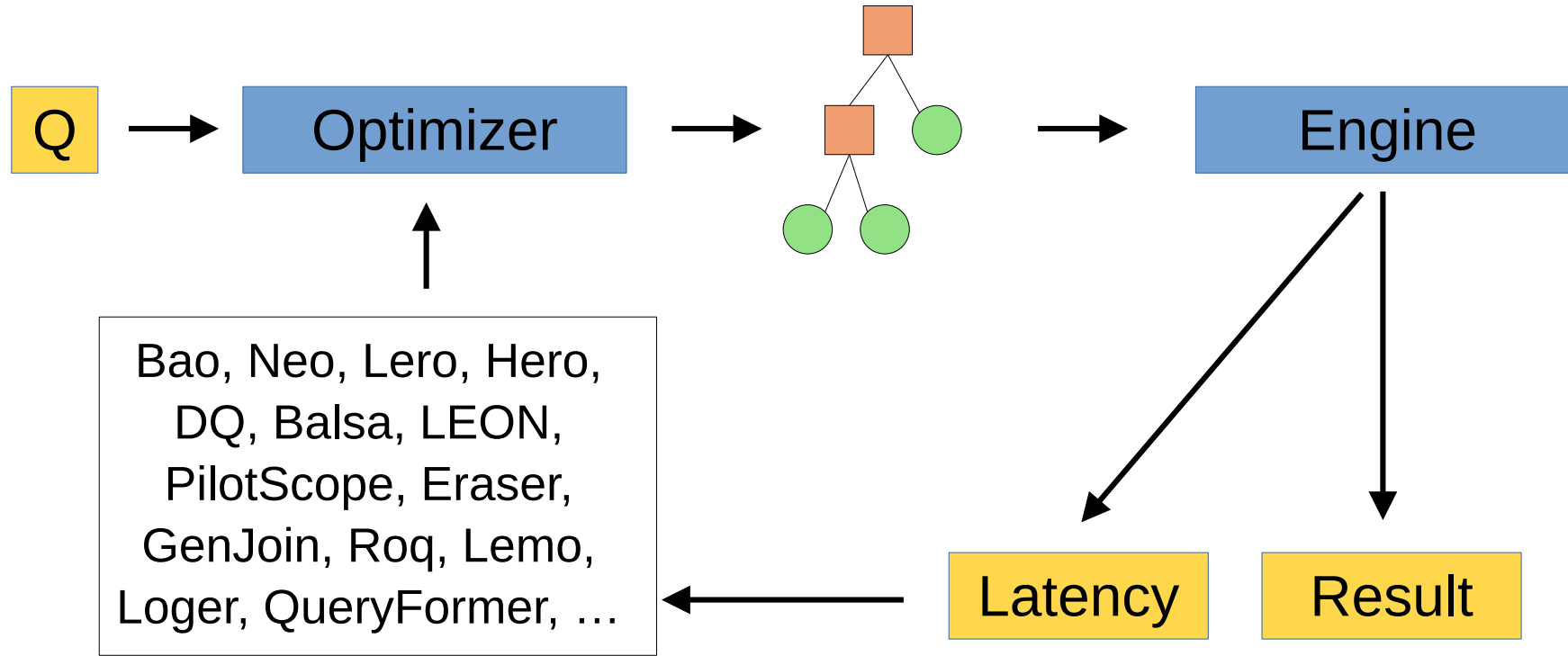
QOs are leaving info on the table



QOs are leaving info on the table



QOs are leaving info on the table



At least 29 others!

This Talk

- Why **learn** a query optimizer?
- Lessons learned **deploying learned QOs**
- **LimeQO**, an offline learned query optimizer
- The end of RL for QO?

The initial (failed) pitch

- “Put this cool RL stuff into your QO!” – Us
- “... no.” – basically everyone we talked to



Sample Inefficiency
Even an hour of
startup time kills POCs



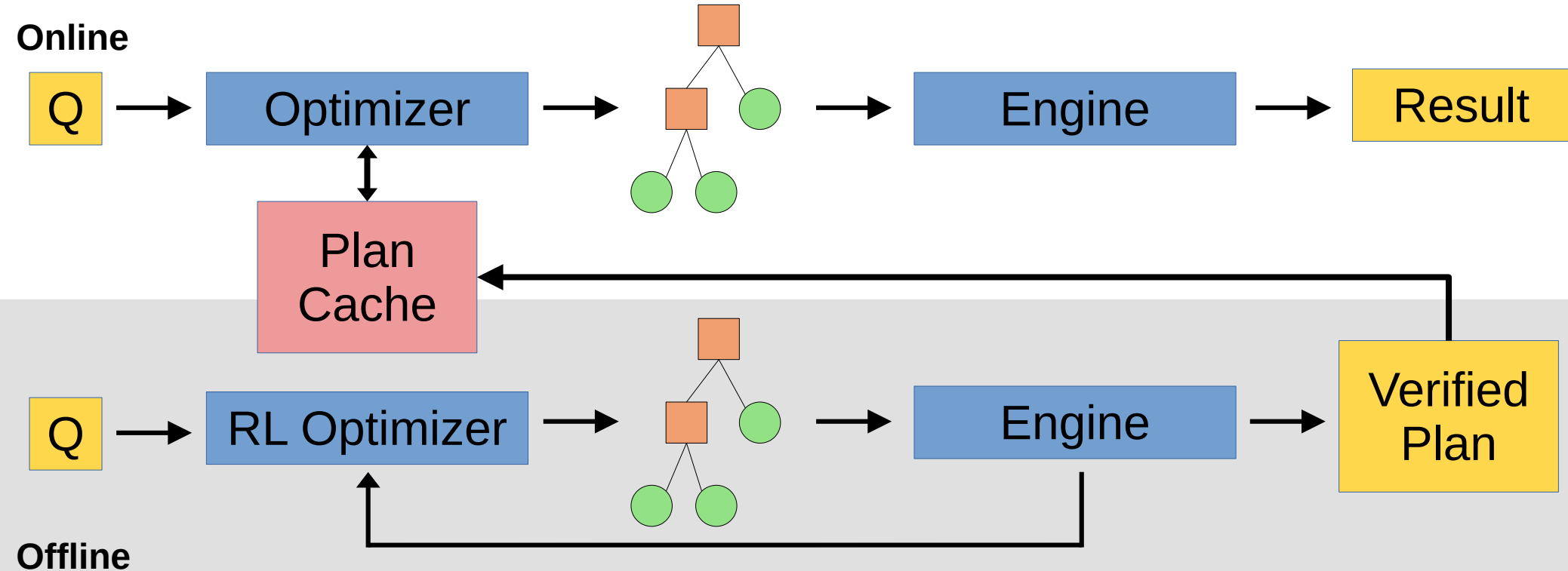
Exploration → Regressions
Unpredictable query
slowdowns turn into 3AM
pages.



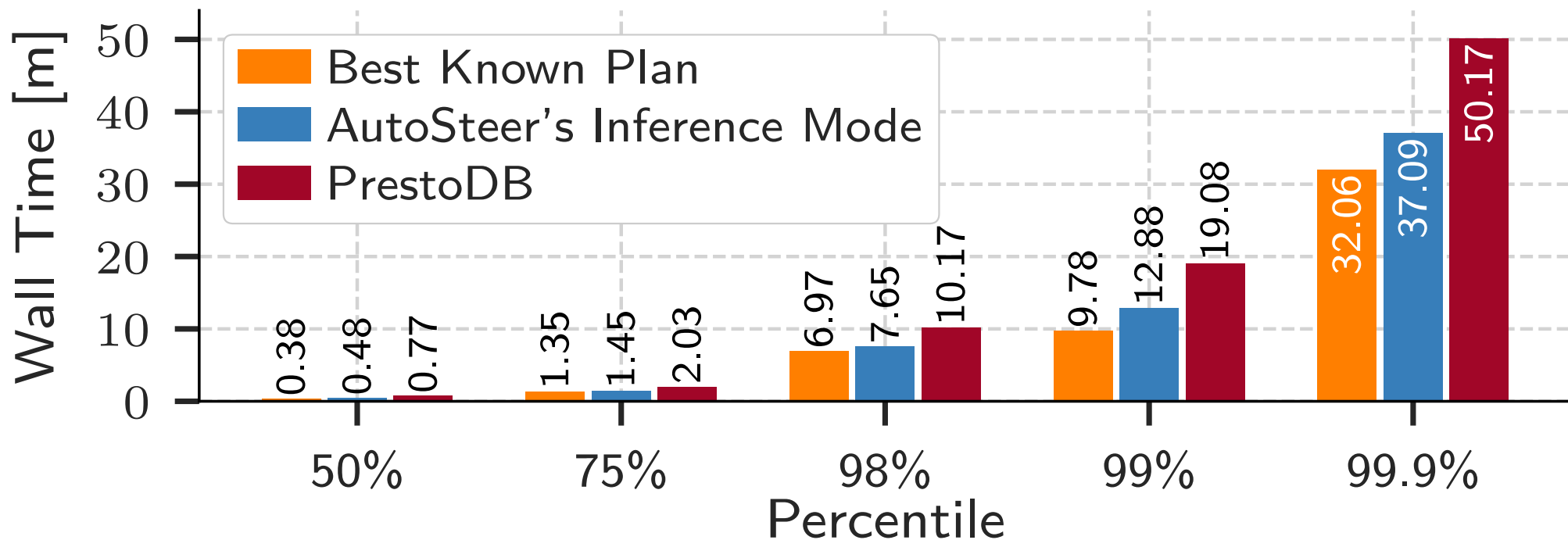
Hot-path Complexity
Putting RL components
(complex) into a QO
(complex) is scary

What ended up landing

- Use RL *offline*, cache good plans, reuse live.



Problem Solved?



Problem Not Solved

- Offline execution time *is a resource*
 - ... and therefore, as DB folks, we must optimize it!
- Given X hours of offline exploration time, maximize the improvement to my workload latency

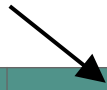
This Talk

- Why **learn** a query optimizer?
- Lessons learned **deploying learned QOs**
- **LimeQO**, an offline learned query optimizer
- The end of RL for QO?



Joint work with Zixuan Yi,
Yao Tian, and Zack Ives

Default plan



	H_1	H_2	H_3	H_4	...	H_k
Q_1						
Q_2						
Q_3						
...						
Q_n						

Default plan

	H_1	H_2	H_3	H_4	...	H_k
Q_1	23					
Q_2	21					
Q_3	18					
...	...					
Q_n	22					



AutoSteer tests one hint per query offline

Default plan

	H_1	H_2	H_3	H_4	...	H_k
Q_1	23					43
Q_2	21					
Q_3	18		10		8	
...	...					
Q_n	22	32				



AutoSteer tests one hint per query offline

Default plan

	H_1	H_2	H_3	H_4	...	H_k
Q_1	23					43
Q_2	21				8	
Q_3	18		10			
...	...					
Q_n	22	32				



Some hints don't work out (slower)

Default plan

	H_1	H_2	H_3	H_4	...	H_k
Q_1	23					43
Q_2	21				8	
Q_3	18		10			
...	...					
Q_n	22	32				



Some hints do work out (faster)

Default plan

	H_1	H_2	H_3	H_4	...	H_k
Q_1	23					43
Q_2	21				8	
Q_3	18		10			
...	...					
Q_n	22	32				

Offline time used: $43 + 8 + 10 + 32 = 93s$

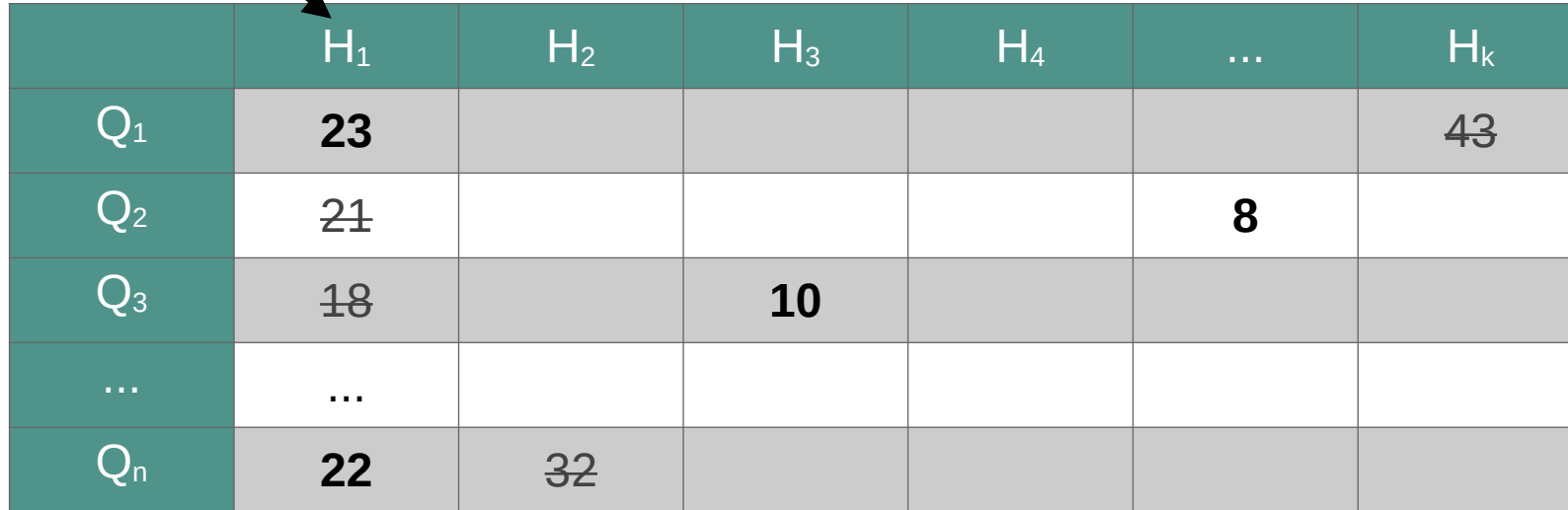
Default plan

	H_1	H_2	H_3	H_4	...	H_k
Q_1	23					43
Q_2	21				8	
Q_3	18		10			
...	...					
Q_n	22	32				

Offline time used: $43 + 8 + 10 + 32 = 93s$
Reduction in query time: $(21 - 8) + (18 - 19) = 12s$

Key insight!

Default plan



	H ₁	H ₂	H ₃	H ₄	...	H _k
Q ₁	23					43
Q ₂	21				8	
Q ₃	18		10			
...	...					
Q _n	22	32				

We want to **minimize** offline time used and **maximize** reduction in query time!

Offline time used: $43 + 8 + 10 + 32 = 93\text{s}$

Reduction in query time: $(21 - 8) + (18 - 19) = 12\text{s}$

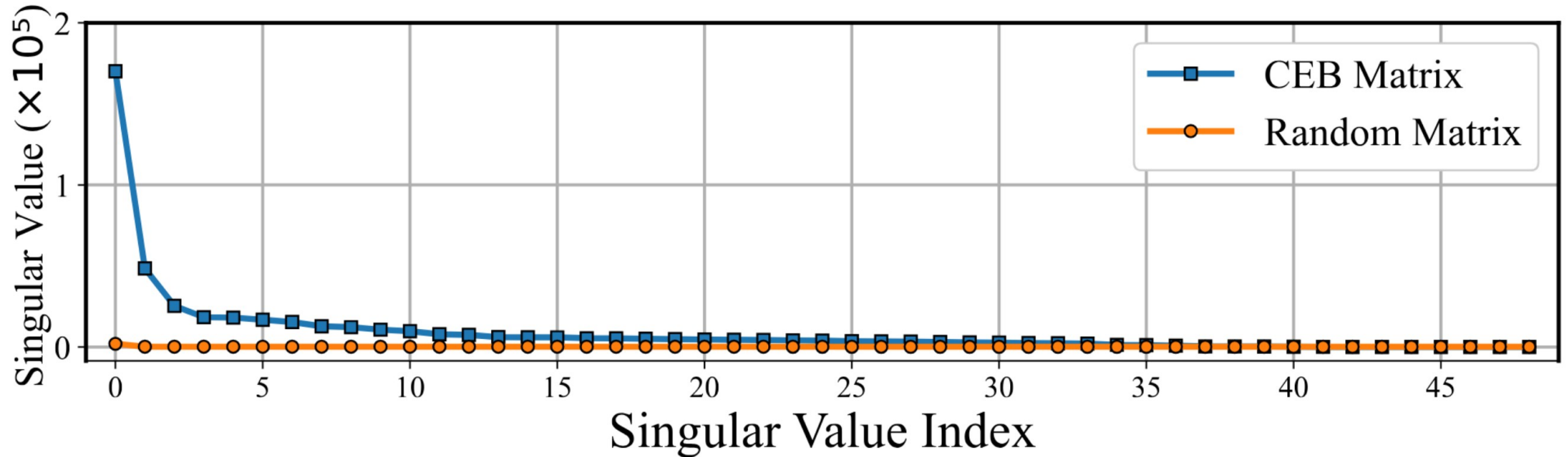
	H_1	H_2	H_3	H_4	...	H_k
Q_1	23					43
Q_2	21				8	
Q_3	18		10			
...	...					
Q_n	22	32				

$= \hat{W}$
Workload
Matrix

\hat{W} is the partially observed workload matrix

W is the full workload matrix

Matrix Properties of W



→ the workload matrix has **low rank**

Matrix Properties of W

$$\begin{array}{ccccc} Q & \times & H & = & W \\ N \times R & & R \times K & & N \times K \end{array}$$

N is the number of queries in the workload

K is the number of possible hints

Why is W low rank?

→ Queries with similar performance on H_a are likely to have similar performance on H_b

LimeQO

- Given a partially observed \hat{W} , predict W

Default plan

	H_1	H_2	H_3	H_4	...	H_k
Q_1	23					
Q_2	21					
Q_3	18					
...
Q_n	22					

LimeQO



LimeQO

- Given a partially observed \hat{W} , predict W

Default plan

	H_1	H_2	H_3	H_4	...	H_k
Q_1	23	≈ 22	≈ 31	≈ 81	$\approx \dots$	≈ 62
Q_2	21	≈ 9	≈ 14	≈ 43	$\approx \dots$	≈ 11
Q_3	18	≈ 12	≈ 9	≈ 12	$\approx \dots$	≈ 4
...	$\approx \dots$	$\approx \dots$	$\approx \dots$	$\approx \dots$	$\approx \dots$	$\approx \dots$
Q_n	22	≈ 19	≈ 23	≈ 4	$\approx \dots$	≈ 16

LimeQO



Use matrix completion to guess the rest of the matrix

LimeQO

- Given a partially observed \hat{W} , predict W

Default plan

	H ₁	H ₂	H ₃	H ₄	...	H _k
Q ₁	23	≈ 22	≈ 31	≈ 81	≈ ...	≈ 62
Q ₂	21	≈ 9	≈ 14	≈ 43	≈ ...	≈ 11
Q ₃	18	≈ 12	≈ 9	≈ 12	≈ ...	≈ 4
...	≈ ...	≈ ...	≈ ...	≈ ...	≈ ...	≈ ...
Q _n	22	≈ 19	≈ 23	≈ 4	≈ ...	≈ 16

LimeQO



Identify candidates for exploration (consider cost and benefit)

LimeQO

- Given a partially observed \hat{W} , predict W

Default plan

	H_1	H_2	H_3	H_4	...	H_k
Q_1	23				...	
Q_2	21	≈ 9			...	
Q_3	18				...	≈ 4
...	
Q_n	22			≈ 4	...	

LimeQO



Identify candidates for exploration (consider cost and benefit)

LimeQO

- Given a partially observed \hat{W} , predict W

Default plan

	H ₁	H ₂	H ₃	H ₄	...	H _k
Q ₁	23				...	
Q ₂	21	18			...	
Q ₃	18				...	4
...	
Q _n	22			> 22	...	

Censored observation

LimeQO



LimeQO

- Given a partially observed \hat{W} , predict W

Default plan

	H ₁	H ₂	H ₃	H ₄	...	H _k
Q ₁	23	≈ 14	≈ 11	≈ 22	≈ ...	≈ 88
Q ₂	21	18	≈ 17	≈ 43	≈ ...	≈ 22
Q ₃	18	≈ 11	≈ 15	≈ 16	≈ ...	4
...	≈ ...	≈ ...	≈ ...	≈ ...	≈ ...	≈ ...
Q _n	22	≈ 22	≈ 8	> 22	≈ ...	≈ 24

LimeQO



Do matrix completion again with new values

LimeQO

- Given a partially observed \hat{W} , predict W

Default plan

	H ₁	H ₂	H ₃	H ₄	...	H _k
Q ₁	23	≈ 14	≈ 11	≈ 22	≈ ...	≈ 88
Q ₂	21	18	≈ 17	≈ 43	≈ ...	≈ 22
Q ₃	18	≈ 11	≈ 15	≈ 16	≈ ...	4
...	≈ ...	≈ ...	≈ ...	≈ ...	≈ ...	≈ ...
Q _n	22	≈ 22	≈ 11	> 22	≈ ...	≈ 8

LimeQO



Identify candidates for exploration (consider cost and benefit)

LimeQO

- Given a partially observed \hat{W} , predict W

Default plan

	H ₁	H ₂	H ₃	H ₄	...	H _k
Q ₁	23		≈ 11		...	
Q ₂	21	18	≈ 17		...	
Q ₃	18				...	4
...	
Q _n	22			> 22	...	≈ 8

LimeQO



Identify candidates for exploration (consider cost and benefit)

LimeQO

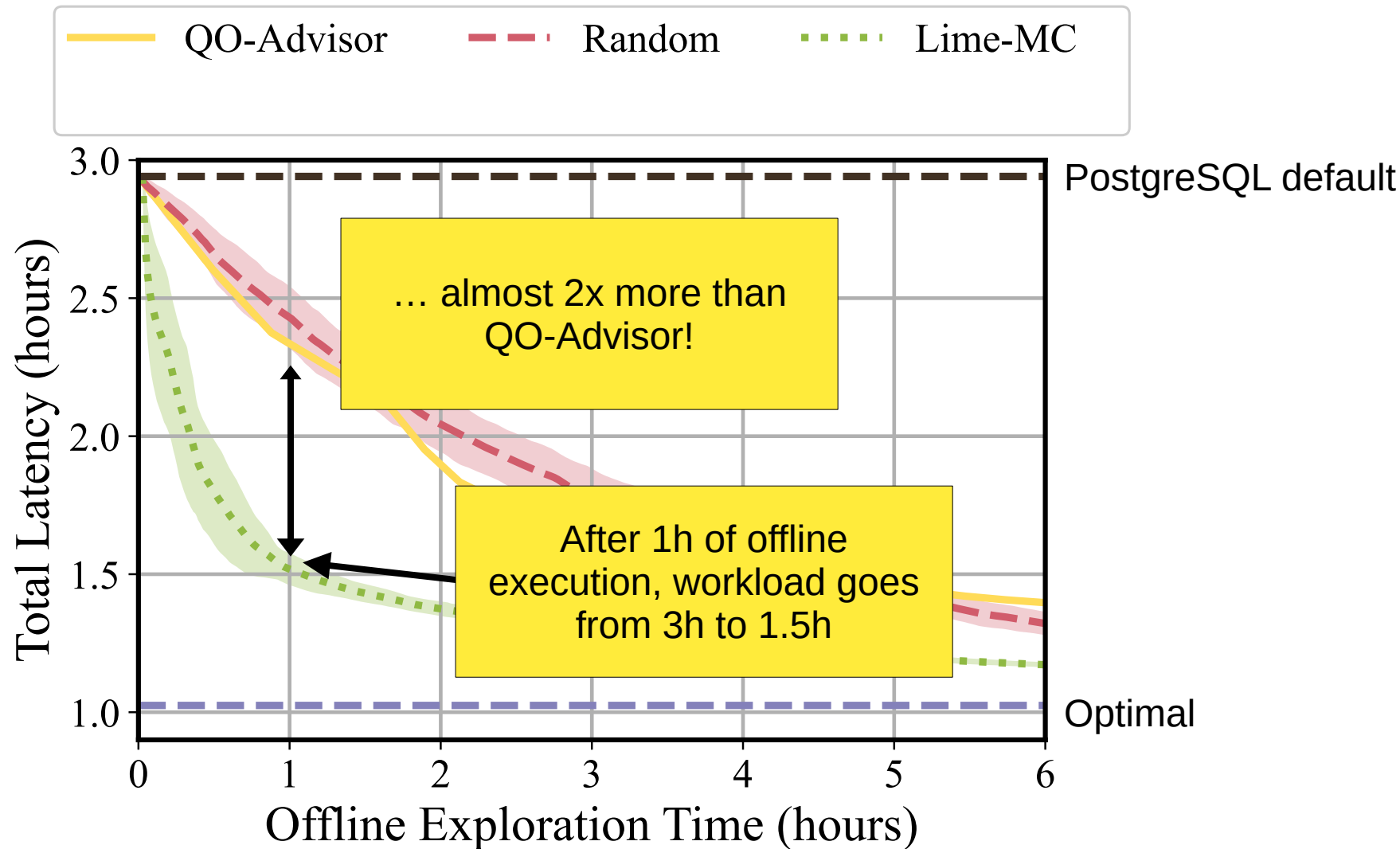
- Given a partially observed \hat{W} , predict W

Default plan

	H ₁	H ₂	H ₃	H ₄	...	H _k
Q ₁	23		> 23		...	
Q ₂	21	18	12		...	
Q ₃	18				...	4
...	
Q _n	22			> 22	...	14

LimeQO





Many more fun problems...

- Matrix completion with censored observations
- Which entries to explore?
 - → “acquisition function”
- Full experimental rundown
- <https://rm.cab/limeqo>



What about the query level?

- LimeQO is an offline optimizer for a workload
 - Core approach: transductive learning
 - Assumes each query has a small set of options
- BayesQO is an offline optimizer for a single query
 - Core approach: Bayes opt
 - Only works on one query at a time



This Talk

- Why **learn** a query optimizer?
- Lessons learned **deploying learned QOs**
- **LimeQO**, an offline learned query optimizer
- The end of RL for QO?

RL for QO Woes



Sample Inefficiency
Even an hour of
startup time kills POCs

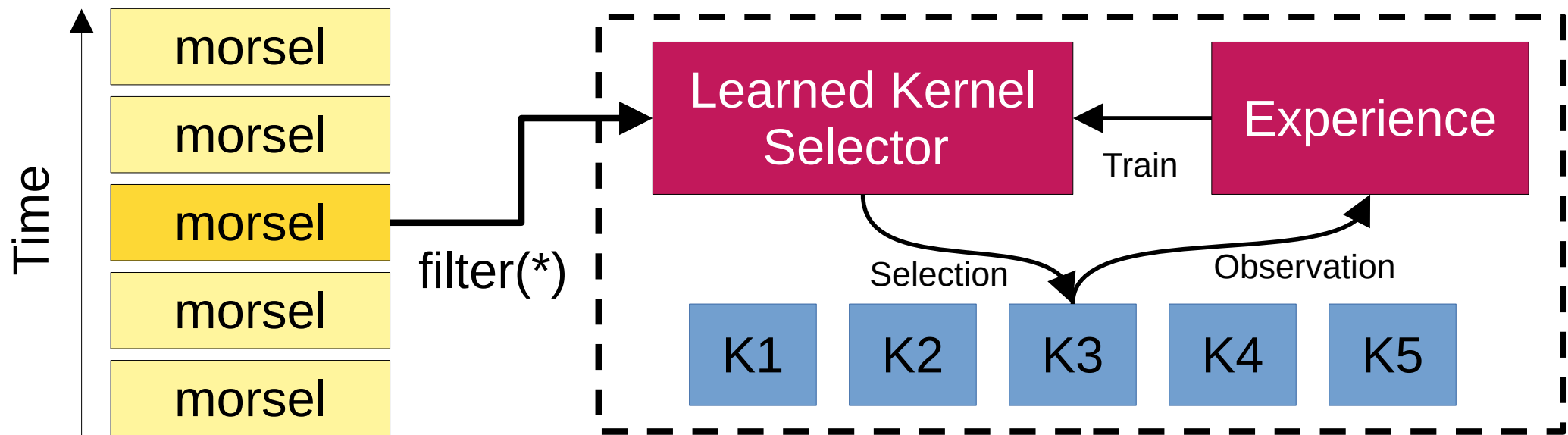


Exploration → Regressions
Unpredictable query
slowdowns turn into 3AM
pages.



Hot-path Complexity
Putting RL components
(complex) into a QO
(complex) is scary

Morsel-Driven Parallelism



Sample Inefficiency?

1k-2k+ morsels *per query*! ✓

Exploration → Regressions?

Incorrect decisions average out over the course of a single query! ✓

Hot-path Complexity?

The EE is often *less* complex than the QO, but is not simple. ⚠

Adversarial Benchmark Generation

Jeffrey Tao, Yimeng Zheng, Natalie Maus, Haydn Jones, Jacob Gardner, Ryan Marcus | DB@Penn

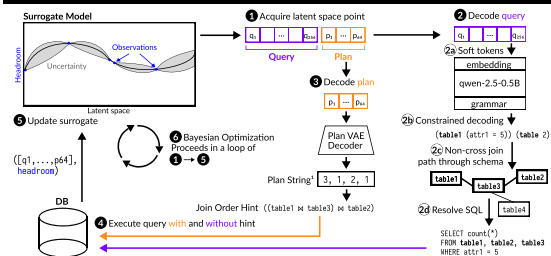


Figure 1. Our system generates a benchmark by searching the joint space of queries and plans using Bayesian Optimization.

Motivation

Benchmarks help us build high-performance systems. Recent SQL database benchmarks have focused on realism. But we may be over-indexing on optimizing what's already fast!

We propose a direct method¹ for generating maximally challenging benchmarks:

1. Propose potentially difficult queries
2. Use offline optimization to find faster plans
3. Maximize the DBMS's under-performance

We model this as a black-box optimization problem and leverage Bayesian optimization techniques. This allows us to directly find performance bugs within a given DBMS.

Results

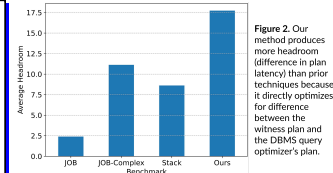


Figure 2. Our method produces more headroom (difference in plan latency) than prior techniques because it directly optimizes for difference between the witness plan and the DBMS query optimizer's plan.

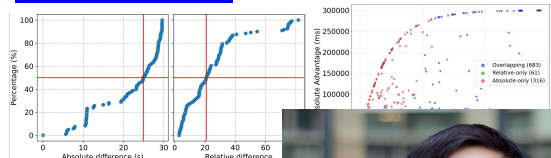
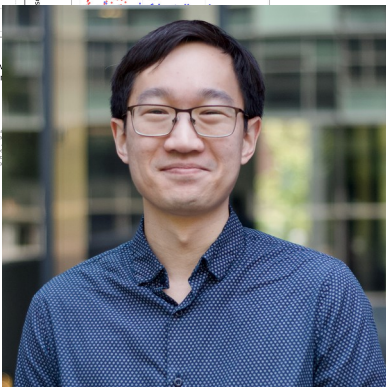


Figure 3. Left: We conduct optimization runs for absolute (DBMS - witness) difference > 1s. Right: Right: Both optimization targets find

Future Work

- Generate benchmarks on other DBMSes to establish generality of our technique
- Compare performance bugs across systems
- Investigate why the DBMS's plan differs from the witness

Jeff Tao



Adaptive Execution Engine Through Low Overhead RL

Zijie Zhao, Ryan Marcus



[WHO needs to be adaptive?]

Low-level kernels in modern data systems.

[Adaptive to WHAT?]

hardware, query, and data.

[OK, Anything else interesting besides applying {your_favourite_rl_algo} to this problem?]

New constraint: Decision must be **ultra-lightweight**, which need to be **invoked thousands times in a sec.**

New opportunity: Handle "what-if" problem by **selectively generating counterfactuals.**

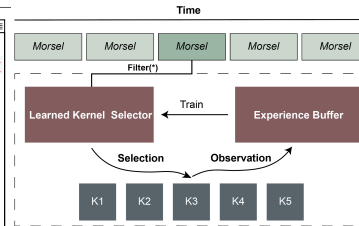
```
# Efficient kernel implementations
def index_iter(x: Morsel, alpha: float, beta: int) -> Morsel: ...
def slice_iter(x: Morsel, alpha: float, beta: int) -> Morsel: ...

class Filter(AdaptiveKernel):
    KERNELS = [index_iter, slice_iter]

    @classmethod
    def extract_features(cls, x: Morsel):
        # return features that will change the kernel selection
        return (x.selectivity, x.length)

    @classmethod
    def fallback(cls, features) -> int:
        # define a fallback option
        return 0

# Enjoy!
out = Filter.adaptive_execute(data, features, 0.1, 3)
```



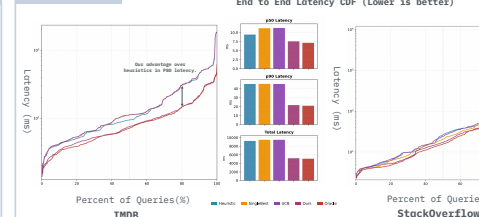
Learning by bootstrapping counterfactuals

Input: Query q , history $\mathcal{Q} = \{(x_i, y_i)\}_{i=1}^n$,

level α , bandwidth h , threshold N_{min} .

1. **Similarity weighting (RBF):** $w_i = \exp\left(-\frac{\|x_i - q\|^2}{h^2}\right)$
2. **Normalized sampling distribution:** $\tilde{w}_i = w_i / \sum_{j=1}^n w_j$
3. **Kernel-wise mean & variance (CLT):**
$$\hat{\mu}_k = \sum_{i=1}^n \tilde{w}_i y_{ik}, \hat{\sigma}_k^2 = \frac{1}{n} \left(\sum_{i=1}^n \tilde{w}_i y_{ik}^2 - \hat{\mu}_k^2 \right).$$
4. **Pairwise test (z-test):** Select $k^* \text{ s.t. for } k \neq k^*, \text{ reject } \mu_k^* \leq \mu_{k^*} \text{ at level } \alpha.$
5. **Decision rule:** if $N_{eff} = \frac{(\sum w_i)^2}{\sum w_i^2} > N_{min}$, exploit k^* , else evaluate all kernels and update \mathcal{Q}

Experiments



We generated 500 hundred queries using LLM on IMDB and StackOverflow Dataset, and measured end latencies. We compared our method against the **human defined heuristics**, the **vanilla UCB algorithm**, and the **optimal**. Our method constantly outperforms all other baselines and show near optimal performance.



Zijie Zhao

Get these slides: <https://rm.cab/nedb26>
My homepage: <https://ryanmarc.us>