

# 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?

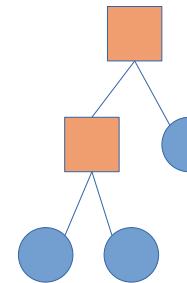
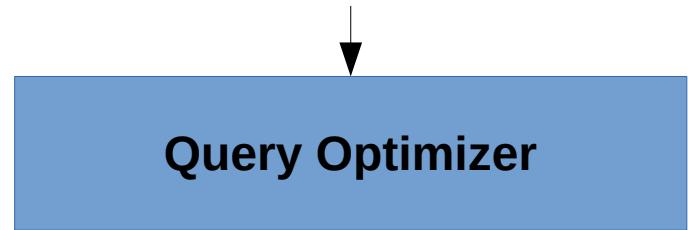
# This Talk

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

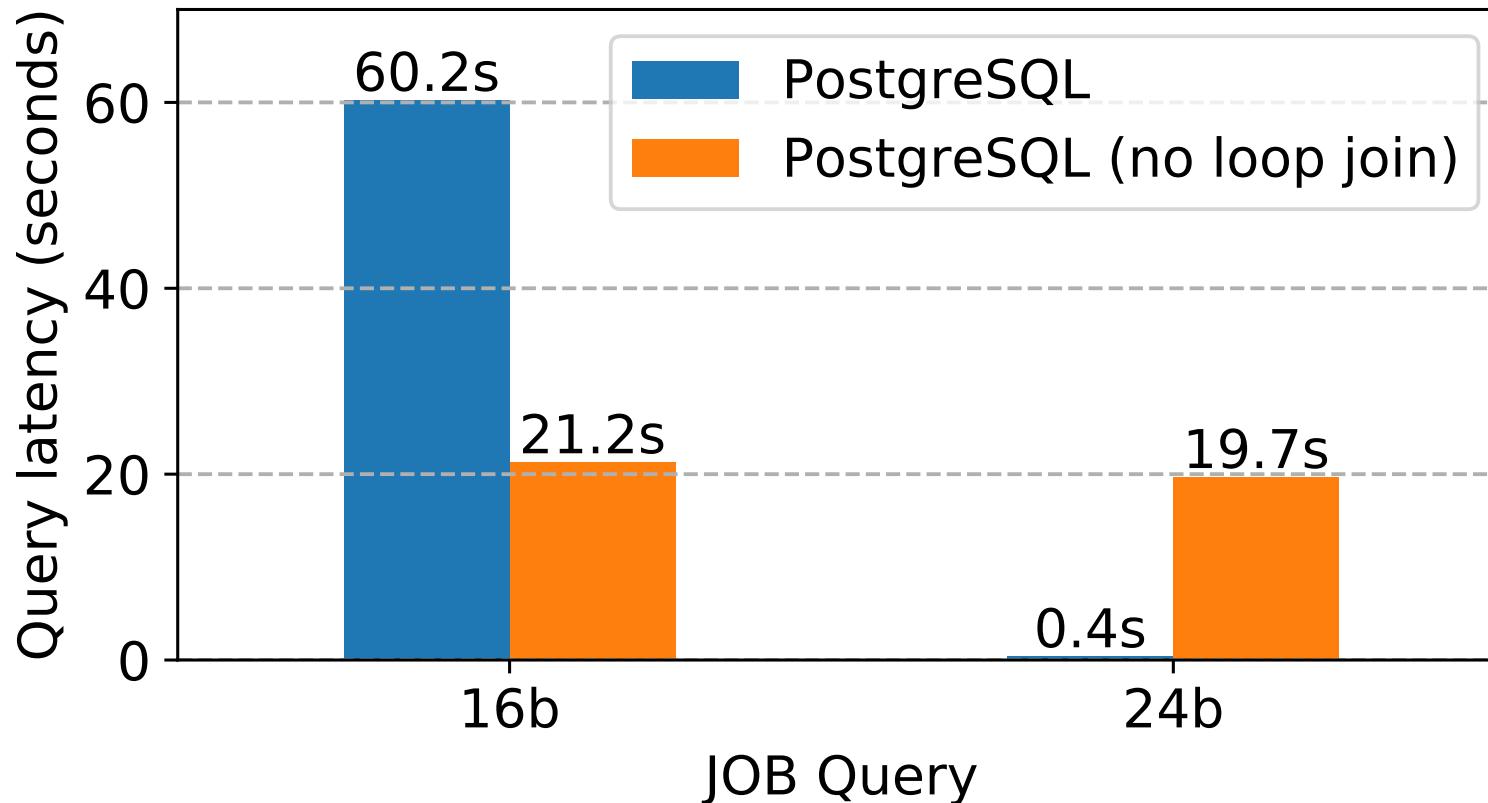
# QO is a huge effort

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

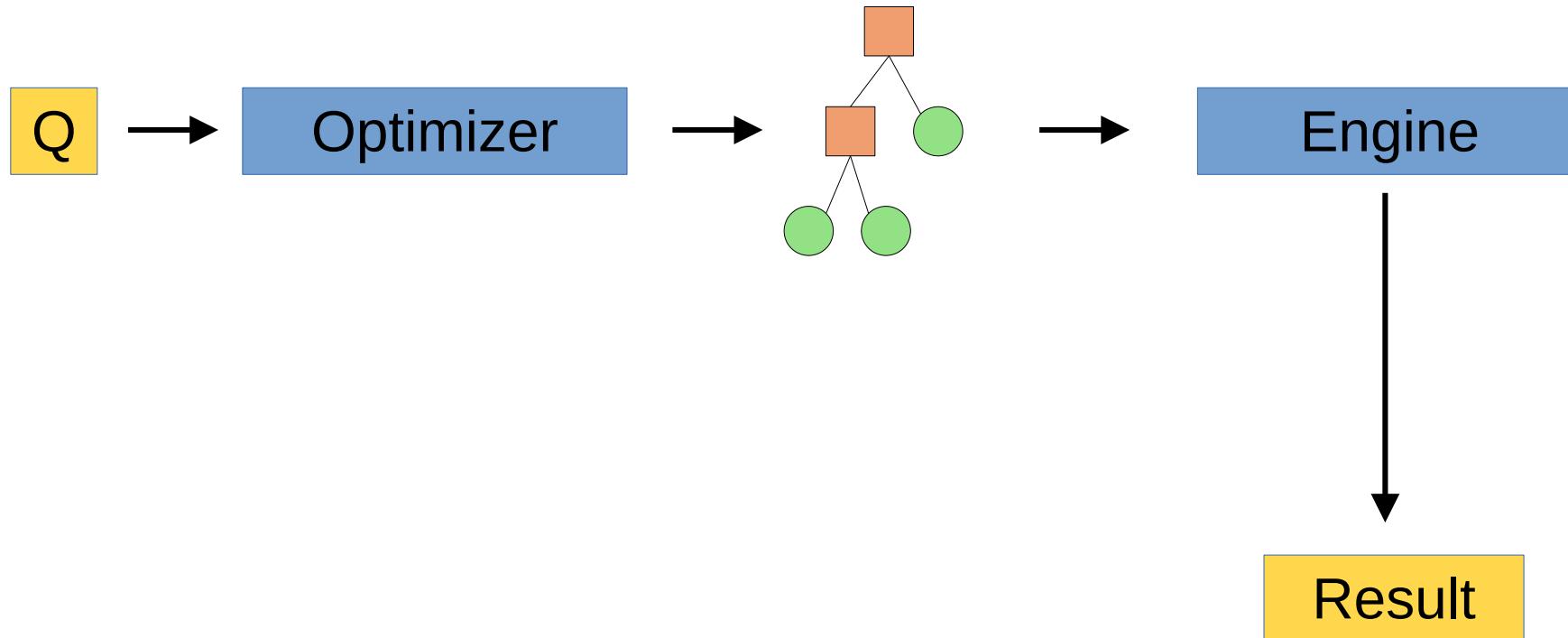
```
SELECT *  
FROM t1, t2 WHERE...
```



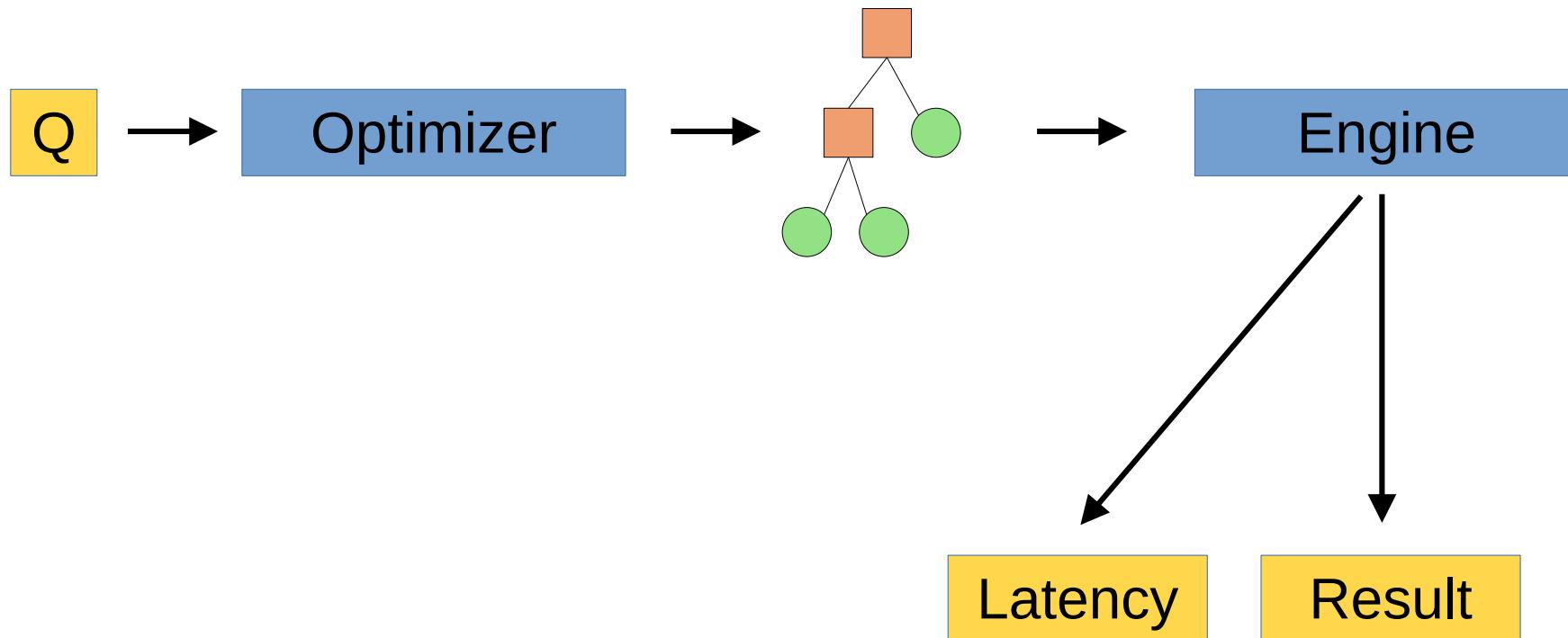
# QOs aren't that good



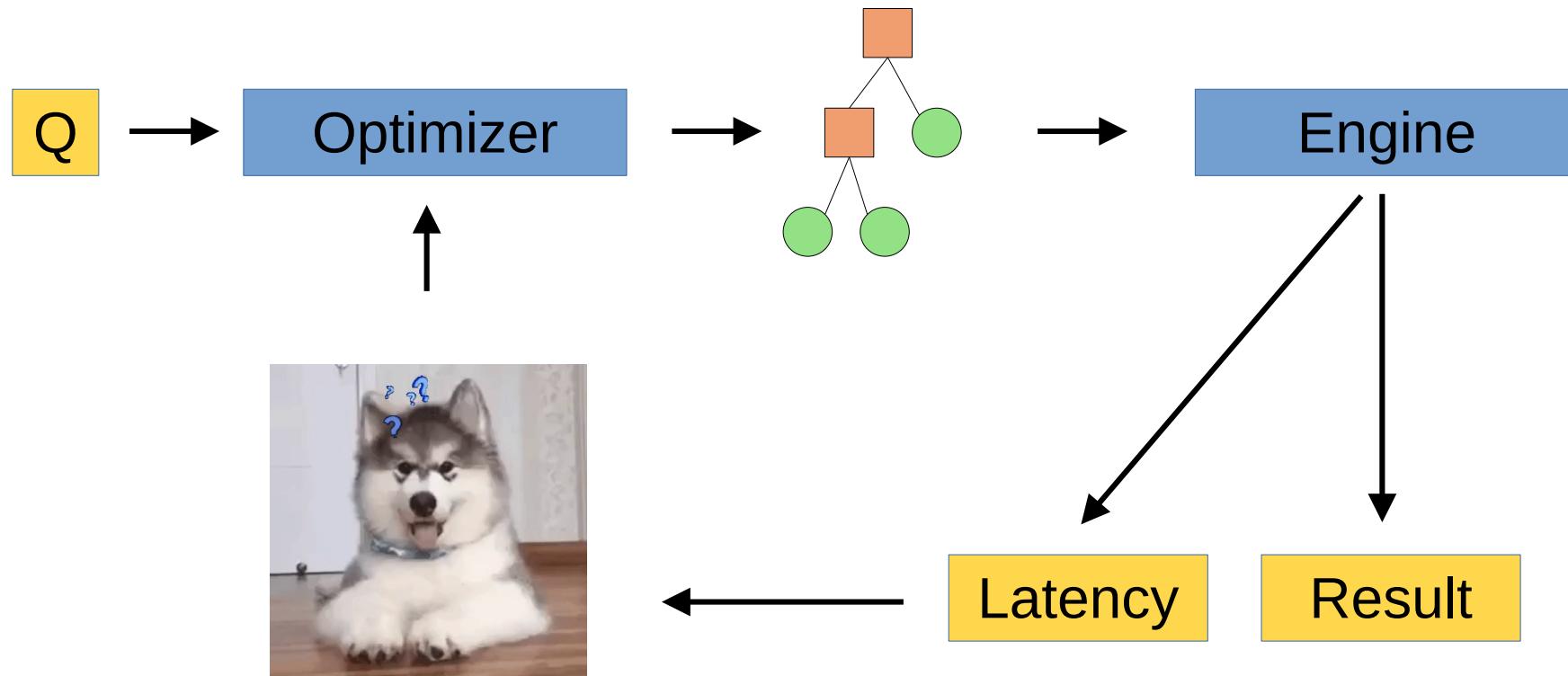
# QOs are leaving info on the table



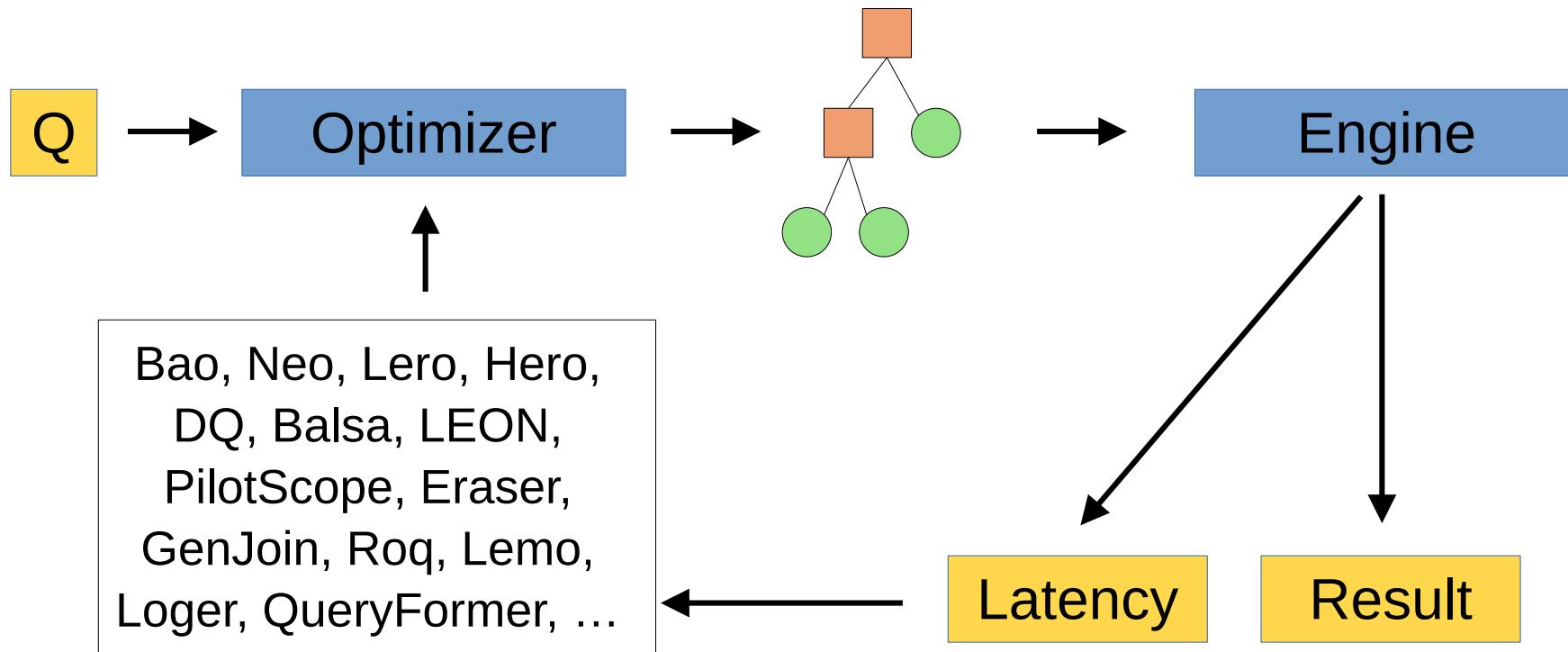
# QOs are leaving info on the table



# QOs are leaving info on the table



# QOs are leaving info on the table



# 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



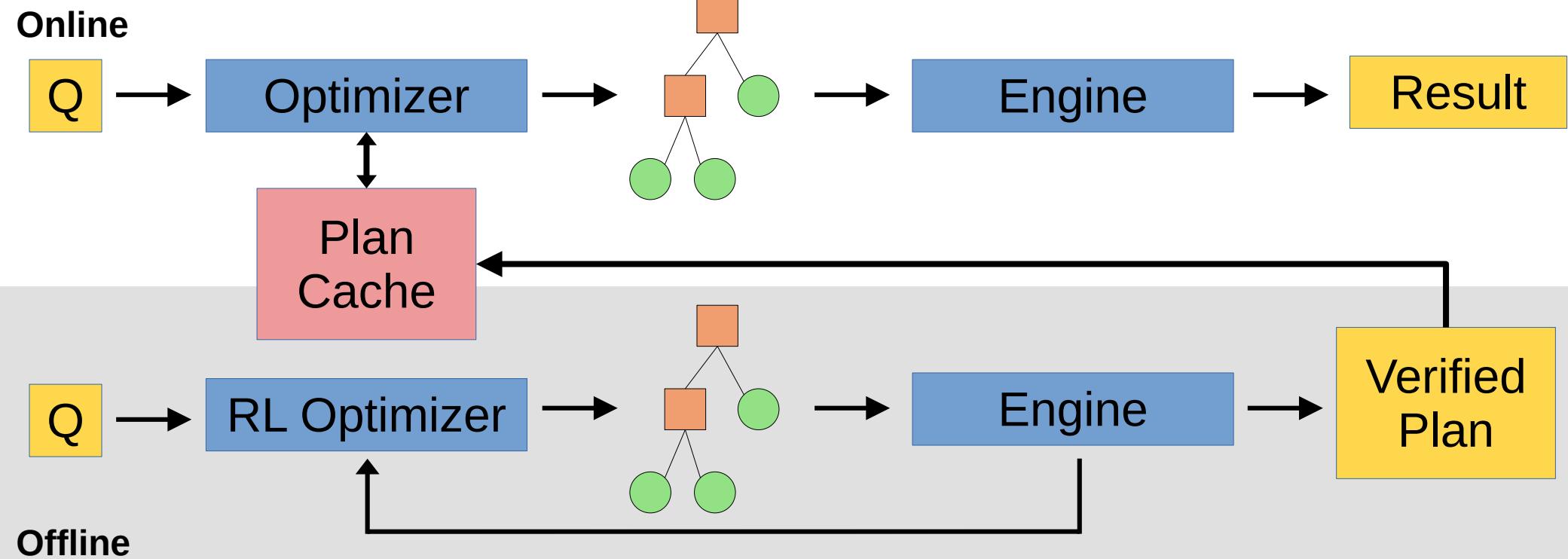
**Exploration → Regressions**  
Unpredictable query  
slowdowns turn into 3AM  
pages.



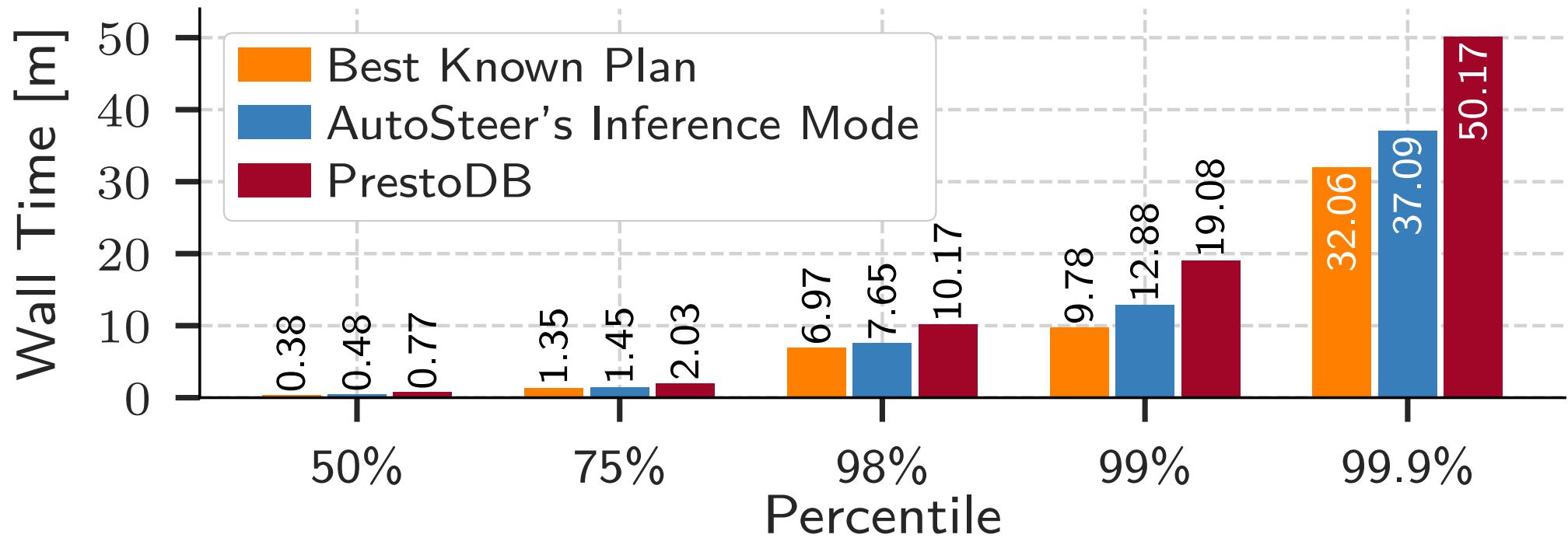
**Hot-path Complexity**  
Putting RL components  
(complex) into a QO  
(complex) is scary

# 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

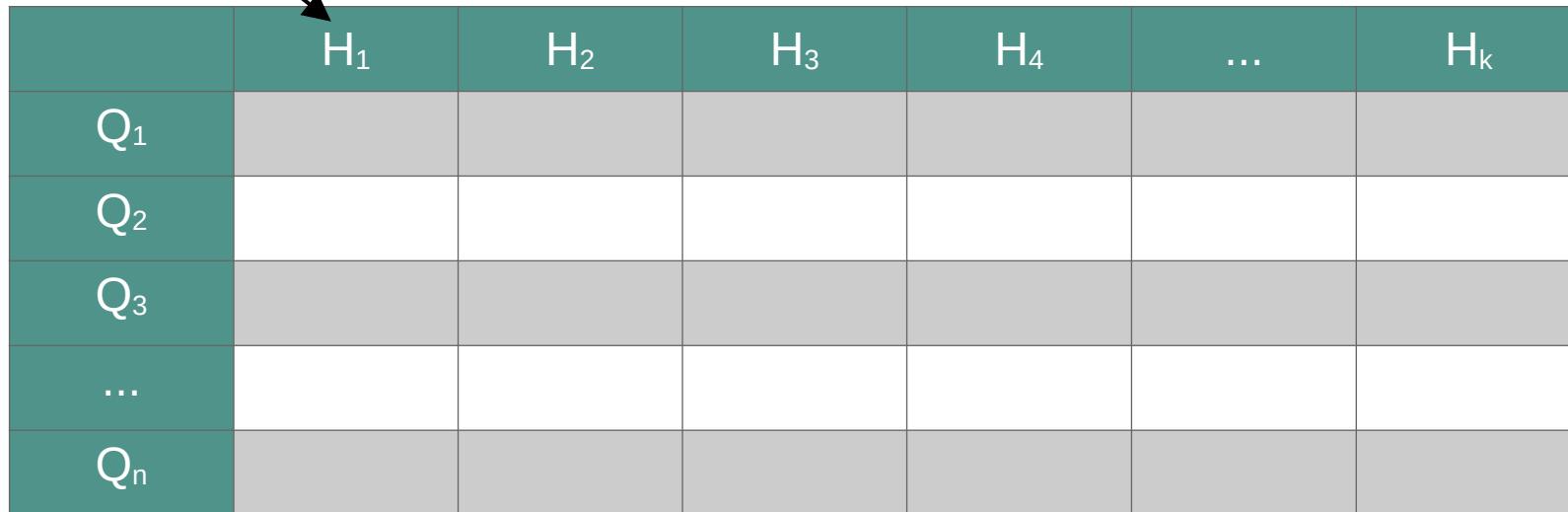
# This Talk

- 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					
...	...					
$Q_n$	22	32	10	8		



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					
...	...					
$Q_n$	22	32	10	8		



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					
$Q_3$	18					
...	...					
$Q_n$	22	32	10	8		



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					
$Q_3$	18					
...	...					
$Q_n$	22	32	10	8		

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					
$Q_3$	18					
...	...					
$Q_n$	22	32				

10

8

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

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

# 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:  $43 + 8 + 10 + 32 = 93s$

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

	$H_1$	$H_2$	$H_3$	$H_4$	...	$H_k$
$Q_1$	<b>23</b>					<b>43</b>
$Q_2$	<b>21</b>				<b>8</b>	
$Q_3$	<b>18</b>		<b>10</b>			
...	...					
$Q_n$	<b>22</b>	<b>32</b>				

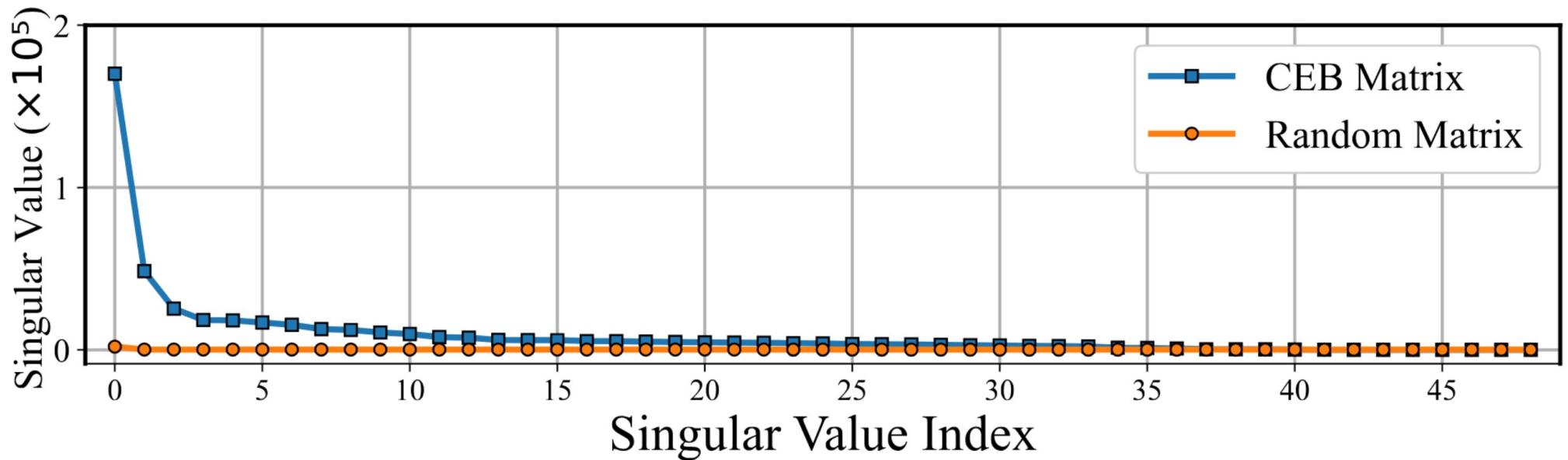
$$= \hat{W}$$

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 $W$

$$Q \times H = W$$

$N \times R$        $R \times K$        $N \times K$

$N$  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	$\approx 22$	$\approx 31$	$\approx 81$	$\approx \dots$	$\approx 62$
$Q_2$	21	$\approx 9$	$\approx 14$	$\approx 43$	$\approx \dots$	$\approx 11$
$Q_3$	18	$\approx 12$	$\approx 9$	$\approx 12$	$\approx \dots$	$\approx 4$
...	$\approx \dots$					
$Q_n$	22	$\approx 19$	$\approx 23$	$\approx 4$	$\approx \dots$	$\approx 16$

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	$\approx 22$	$\approx 31$	$\approx 81$	$\approx \dots$	$\approx 62$
$Q_2$	21	<b><math>\approx 9</math></b>	$\approx 14$	$\approx 43$	$\approx \dots$	$\approx 11$
$Q_3$	18	$\approx 12$	$\approx 9$	$\approx 12$	$\approx \dots$	<b><math>\approx 4</math></b>
...	$\approx \dots$	$\approx \dots$	$\approx \dots$	$\approx \dots$	$\approx \dots$	$\approx \dots$
$Q_n$	22	$\approx 19$	$\approx 23$	<b><math>\approx 4</math></b>	$\approx \dots$	$\approx 16$

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	$\approx 9$			...	
$Q_3$	18				...	$\approx 4$
...	...				...	
$Q_n$	22			$\approx 4$	...	

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			$> 22$	...	

LimeQO



Censored observation

# LimeQO

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

Default plan

	$H_1$	$H_2$	$H_3$	$H_4$	...	$H_k$
$Q_1$	23	$\approx 14$	$\approx 11$	$\approx 22$	$\approx \dots$	$\approx 88$
$Q_2$	21	18	$\approx 17$	$\approx 43$	$\approx \dots$	$\approx 22$
$Q_3$	18	$\approx 11$	$\approx 15$	$\approx 16$	$\approx \dots$	4
$\dots$	$\approx \dots$					
$Q_n$	22	$\approx 22$	$\approx 8$	$> 22$	$\approx \dots$	$\approx 24$



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	$\approx 14$	$\approx 11$	$\approx 22$	$\approx \dots$	$\approx 88$
$Q_2$	21	18	$\approx 17$	$\approx 43$	$\approx \dots$	$\approx 22$
$Q_3$	18	$\approx 11$	$\approx 15$	$\approx 16$	$\approx \dots$	4
$\dots$	$\approx \dots$					
$Q_n$	22	$\approx 22$	$\approx 11$	$> 22$	$\approx \dots$	$\approx 8$



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		$\approx 11$		...	
$Q_2$	21	18	$\approx 17$		...	
$Q_3$	18				...	4
...	...	...	...	...	...	
$Q_n$	22			$> 22$	...	$\approx 8$



Identify candidates for exploration (consider cost and benefit)

# LimeQO

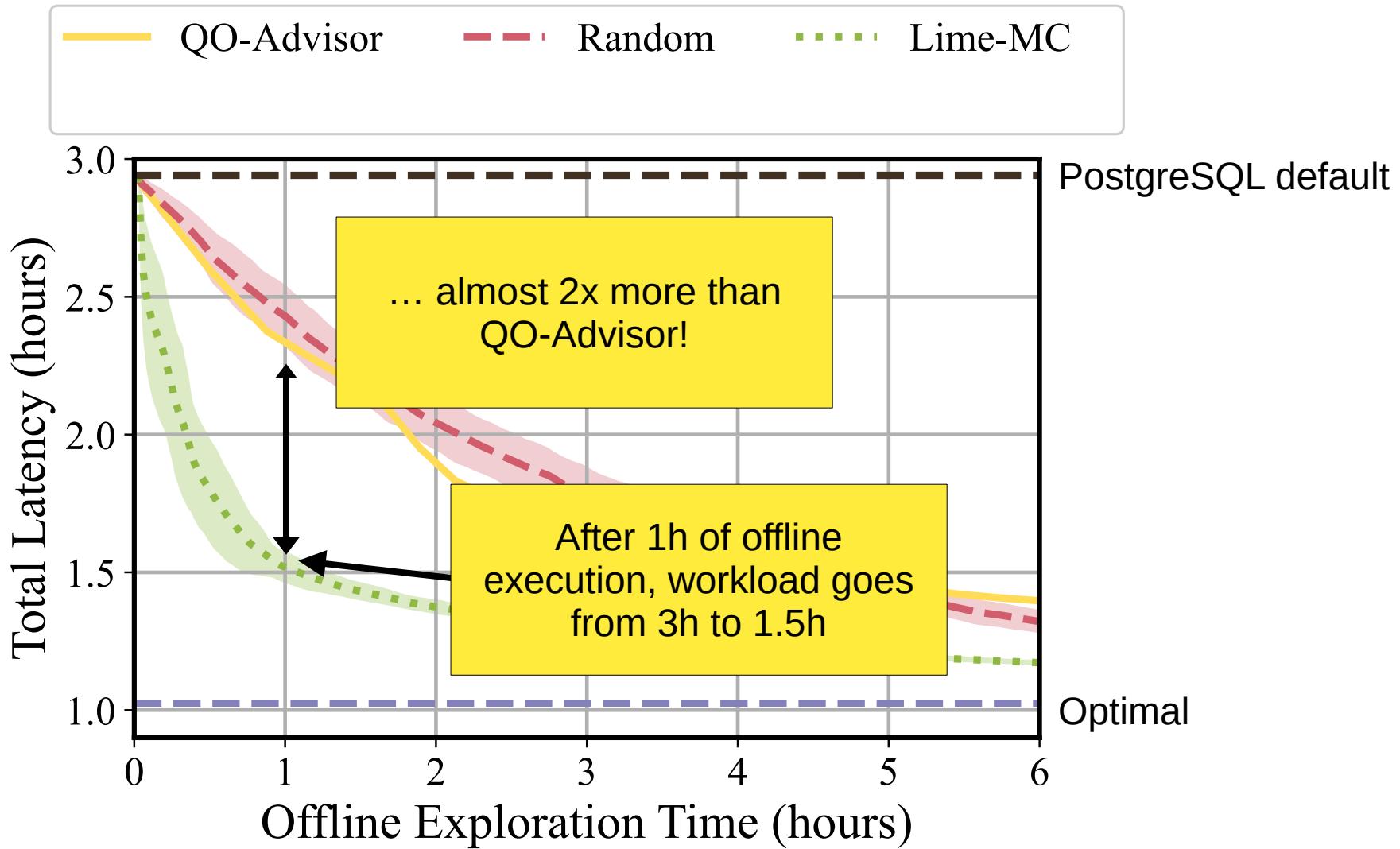
- Given a partially observed  $\hat{W}$ , predict  $W$

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

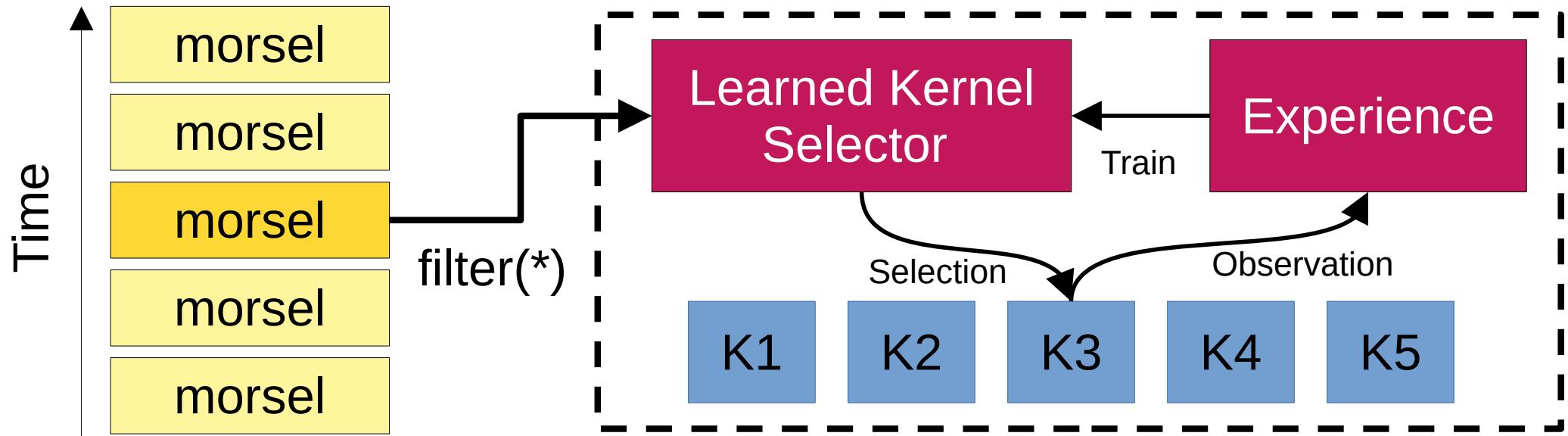


**Exploration → Regressions**  
Unpredictable query slowdowns turn into 3AM pages.



**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

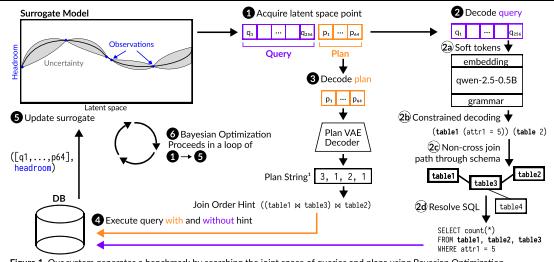


Figure 1. Our system generates a benchmark by searching the joint space of queries and plans using Bayesian Optimization.

Motivation	Results
<p>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!</p> <p>We propose a direct method<sup>1</sup> for generating maximally challenging benchmarks:</p> <ol style="list-style-type: none"> <li>1. Propose potentially difficult queries</li> <li>2. Use offline optimization to find faster plans</li> <li>3. Maximize the DBMS's under-performance</li> </ol> <p>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.</p>	<p>Average Headroom</p> <p>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's query optimizer's plan.</p> <p>Percentage (%)</p>

Figure 3. Left: We conduct optimization runs for absolute (DBMS - with absolute difference > 1s. 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



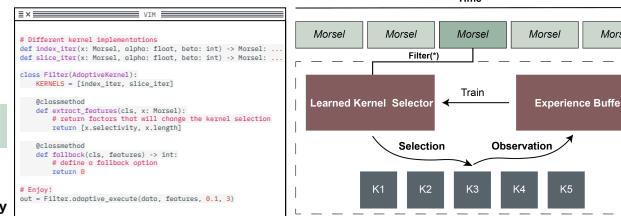
Jeff Tao

## Adaptive Execution Engine Through Low Overhead RL

Zijie Zhao, Ryan Marcus



Time



[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\_favorite\_rl\_algo} to this problem?]

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

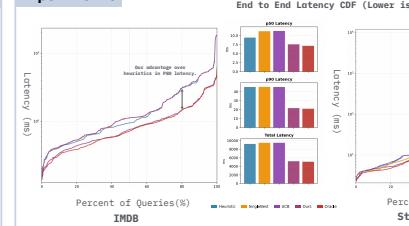
New opportunity: Handle "what-if" problem by selectively generating counterfactuals.

### Learning by bootstrapping counterfactuals

Input: Query  $q$ , history  $\mathcal{D} = \{(x, y)\}_{i=1}^n$ , level  $\alpha$ , 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):**  
 $\hat{\mu}_k = \sum_{i=1}^n \tilde{w}_i y_{i,k}$ ,  $\hat{\sigma}_k^2 = \frac{1}{n} \left( \sum_{i=1}^n \tilde{w}_i y_{i,k}^2 - \hat{\mu}_k^2 \right)$ .
4. **Pairwise test (z-test):**  
Select  $k^*$  s.t. for all  $k \neq k^*$ ,  $\hat{\mu}_i^* \leq \hat{\mu}_k$  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  $\mathcal{D}$

### Experiments



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



Zijie Zhao

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