



Running a Query Optimizer Advisor in Production

What we Learned
(and What the Model didn't)

Matteo Interlandi

HPTS 2024

Crash Course on **Steering Query Optimizers using ML**

Goal

Share our experience on running a ML-based Query Optimizer Advisor at scale

What you will learn (Outline)

- How query hints are used to steer query optimizers toward better plans
- How to automate hint generation using ML
- How to scale it over hundreds of thousands of jobs
- Key lessons learned

For more Details

Research Data Management Track Paper
SIGMOD '21, June 20–25, 2021, Virtual Event, China

Bao: Making Learned Query Optimization Practical

Ryan Marcus
MIT & Intel Labs
rymarcus@csail.mit.edu

Parinaz Negi
MIT
pnegi@csail.mit.edu

Hongzi Mao
MIT
hmao@csail.mit.edu

Nehme Tathai
MIT & Intel Labs
tathai@csail.mit.edu

Mohammad Alizadeh
MIT
alizadeh@csail.mit.edu

Tim Kraska
MIT
tkraska@csail.mit.edu

ABSTRACT
Recent efforts applying machine learning techniques to query optimization have shown great promise for practical gains due to substantive performance overhead. Stability is a key challenge, and prior work performance. Motivated by these difficulties, we introduce Bao (the learned optimizer). Bao takes advantage of the machine built to existing query optimizers by providing per-query optimization hints. Bao combines modern neural network architectures with Thompson sampling, a well-studied reinforcement learning algorithm. As a result, Bao automatically learns to maintain and adapt to changes in query workloads, data, and schemas. Experimentally, Bao demonstrates that Bao can quickly learn strategies that improve end-to-end query execution performance, including full latency, for several workloads containing long-running queries. In cloud environments, we show that Bao can offer both reduced costs and better performance compared with a commercial system.

CCS CONCEPTS
Information Systems → Query optimization.

KEYWORDS
Query optimization, machine learning, reinforcement learning, ACO, Reinforcement Learning, Best-First Search, Parinaz Negi, Hongzi Mao, Nehme Tathai, Mohammad Alizadeh, and Tim Kraska. Bao: Making Learned Query Optimization Practical. In Proceedings of the ACM International Conference on Management of Data (SIGMOD '21), June 20–25, 2021, Virtual Event, China. ACM, New York, NY, USA, 11 pages. <https://doi.org/10.1145/3458188>

1 INTRODUCTION
Query optimization is an important task for database management systems. Despite decades of study [26], the most important elements of query optimization – cardinality estimation and cost modeling – have proven difficult to model [15]. Several works have applied machine learning techniques to these subproblems [17, 40, 41, 51, 53, 55, 73, 76]. While all of these prior solutions demonstrate promising results, we argue that none of the techniques are yet practical, as they suffer from several fundamental problems.

In the best of our knowledge, Bao (Bao’s optimizer) is the first learned optimizer that overcomes the aforementioned problems. Bao is fully integrated into PostgreSQL as an extension, and can be easily installed without the need for a reversible PostgreSQL. The database administrator (DBA) just needs to develop an optimizer model, and even has the option to selectively turn on the learned optimizer on or off for specific queries.

<https://arxiv.org/abs/2106.02021>

<https://doi.org/10.1145/3458188>

<https://arxiv.org/abs/2106.02021>

175

Industrial Track Paper
SIGMOD '21, June 20–25, 2021, Virtual Event, China

Steering Query Optimizers: A Practical Take on Big Data Workloads

Parinaz Negi¹, Matteo Interlandi², Ryan Marcus³, Mohammad Alizadeh¹, Tim Kraska¹
Marc Friedman⁴, Aksh Jindal⁴
MIT¹, Microsoft², Intel Labs³, USA⁴

	Workloads			Total
	A	B	C	
3 Jobs	49K	11K	48K	1.06K
4 Change Streams	48K	13K	21K	80.3K
4 Change Streams	29K	8K	11.8K	54.8K
4 Change rate operators	12K	87	1.8K	16.35K

ABSTRACT
In recent years, there has been tremendous interest in research that applies machine learning to database systems. Being one of the most complex components of a DBMS, query optimizers could benefit from adaptive policies that can learn automatically from the data and the query workload. Recent research has brought up novel ideas towards a learned query optimizer, however these ideas have not been evaluated on a commercial query processor or on large-scale, real-world workloads. In this paper, we take the approach used by Marcus et al. in Bao and adapt it to SCOPF, a big data system used in production at Microsoft. Along with the idea of multiple per-query optimization hints, we propose a novel multi-step training process that allows for the introduction of the concept of a “steering operator”. We describe a pipeline consisting of different rule configurations for steering each query workload, data, or schema and make matters worse. Cardinality estimates based on supervised learning must be obtained when data changes, or risk becoming stale. Several proposed reinforcement learning techniques assume that both the workload and the schema remain constant, and require complex engineering when this is not the case [48, 53, 73, 76].

1 INTRODUCTION
Query optimization is an important task for database management systems. Despite decades of study [26], the most important elements of query optimization – cardinality estimation and cost modeling – have proven difficult to model [15]. Several works have applied machine learning techniques to these subproblems [17, 40, 41, 51, 53, 55, 73, 76]. While all of these prior solutions demonstrate promising results, we argue that none of the techniques are yet practical, as they suffer from several fundamental problems.

In the best of our knowledge, Bao (Bao’s optimizer) is the first learned optimizer that overcomes the aforementioned problems. Bao is fully integrated into PostgreSQL as an extension, and can be easily installed without the need for a reversible PostgreSQL. The database administrator (DBA) just needs to develop an optimizer model, and even has the option to selectively turn on the learned optimizer on or off for specific queries.

<https://arxiv.org/abs/2106.02021>

<https://doi.org/10.1145/3458188>

<https://arxiv.org/abs/2106.02021>

175

Industrial Track Paper
SIGMOD '21, June 20–25, 2021, Virtual Event, China

Deploying a Steered Query Optimizer in Production at Microsoft

Wangda Zhang, Matteo Interlandi, Paul Mirino, Shri Ghan, Natin Chandrahari, Karim Lee, Marc Friedman, Rafal Holt, Hiran Patel, Aksh Jindal
Microsoft
ga-adviser@microsoft.com

ABSTRACT
Modern analytical workloads are highly heterogeneous and naturally complex, making generic query optimizers ineffective for many situations and scenarios. As a result, it is important to steer the optimizer to scenarios of query workloads in production. We conduct a series of experiments to evaluate a query optimizer for workloads in production, and make improvements to the way we solve several optimization challenges including making cost estimates more accurate, making cardinality estimates more precise, and making execution performance estimates more accurate. We describe a pipeline consisting of different rule configurations for steering each query workload, data, or schema and make matters worse. Cardinality estimates based on supervised learning must be obtained when data changes, or risk becoming stale. Several proposed reinforcement learning techniques assume that both the workload and the schema remain constant, and require complex engineering when this is not the case [48, 53, 73, 76].

1 INTRODUCTION
Query optimization is an important task for database management systems. Despite decades of study [26], the most important elements of query optimization – cardinality estimation and cost modeling – have proven difficult to model [15]. Several works have applied machine learning techniques to these subproblems [17, 40, 41, 51, 53, 55, 73, 76]. While all of these prior solutions demonstrate promising results, we argue that none of the techniques are yet practical, as they suffer from several fundamental problems.

In the best of our knowledge, Bao (Bao’s optimizer) is the first learned optimizer that overcomes the aforementioned problems. Bao is fully integrated into PostgreSQL as an extension, and can be easily installed without the need for a reversible PostgreSQL. The database administrator (DBA) just needs to develop an optimizer model, and even has the option to selectively turn on the learned optimizer on or off for specific queries.

<https://arxiv.org/abs/2106.02021>

<https://doi.org/10.1145/3458188>

<https://arxiv.org/abs/2106.02021>

175

Industrial Track Paper
SIGMOD '21, June 20–25, 2021, Virtual Event, China

CONDITIONALLY RISK-AVERSE CONTEXTUAL BANDITS

Mónika Faruq
Yale University (Yale) and Intel Labs
mfaruq@cs.yale.edu, et al.

Paul Mirino
Microsoft Research
pmirino@microsoft.com

Wangda Zhang
Microsoft Research
wangzhang@microsoft.com

ABSTRACT
Contextual bandits with average-case statistical guarantees are inadequate in risk-averse situations because they might trade off degraded worst-case behavior for better average performance. Deploying a risk-averse contextual bandit is challenging because reinforcement learning algorithms are not sensitive to the entire distribution of rewards, necessitating we exhibit the first risk-averse contextual bandit algorithm with an online regret guarantee. We conduct experiments from diverse scenarios where worst-case outcomes should be avoided, from dynamic pricing, inventory management, and self-tuning software, including a production classic data processing system.

1 Introduction
Contextual bandits [Auer et al., 2002; Langford and Zhang, 2007] are a mature technology with numerous applications; however, adoption has been most aggressive in recommendation scenarios [Bennett and El Ghemal, 2009], where the worst-case outcome is user annoyance. At the other extreme are medical and defense scenarios where worst-case outcomes are literally fatal. In between are scenarios of interest because they should be avoided, e.g., logistics, finance, and self-tuning software, where the term *catastrophe* highlights the inadequacy of average case performance guarantees in real-world applications [Mansour et al., 2021]. These scenarios demand risk-averse, i.e., decisions should sacrifice average performance in order to avoid worst-case outcomes, and incorporating risk aversion into contextual bandits would facilitate adoption. More generally, risk aversion is essential for making informed decisions that align with the risk preferences of the decision maker by balancing the potential benefits and risks of a particular action.

This paper solves risk-averse decision making for contextual bandits via reduction to regression, resulting in the first risk-averse contextual bandit algorithm with an online regret guarantee. The regret guarantee applies over adversarially chosen context sequences and includes the exploration choices made by the algorithm. The approach utilizes utility (rather than bounded) functions, allows for arbitrary action spaces, introduces a compressed overhead relative to the risk-neutral setting, introduces statistical overhead directly related to the desired level of risk-aversion, with no overhead in the risk-neutral limit, and compares with other innovations within the Decision-Estimation framework [Foster et al., 2021], e.g., linear representability [Zhu and Mironov, 2022].

We make the following contributions:

- We explain the problem setting (Section 2) with careful definitions which facilitate the application of theory and reveal the unique status of Section 3.
- We state the resulting algorithm (Section 3), which arise via application of the Estimation-to-Decision framework [Foster et al., 2021].
- We discuss the theoretical utility of explicit rules for algorithm design over more commonly used risk measures VaR and CVaR (Section 2 and 3).
- We provide an empirical support for the technique via diverse scenarios (Section 4). Empirically, tail control is proportionally inexpensive relative to average-case algorithms, justifying the efficacy of average-case guarantees in the self-tuning software for the literature.

[arXiv:2210.13573v2](https://arxiv.org/abs/2210.13573v2) [stat.ML] 8 Jul 2023

Steering Query Optimizers

The Intuition

Intuition **Rule Based Optimizer**

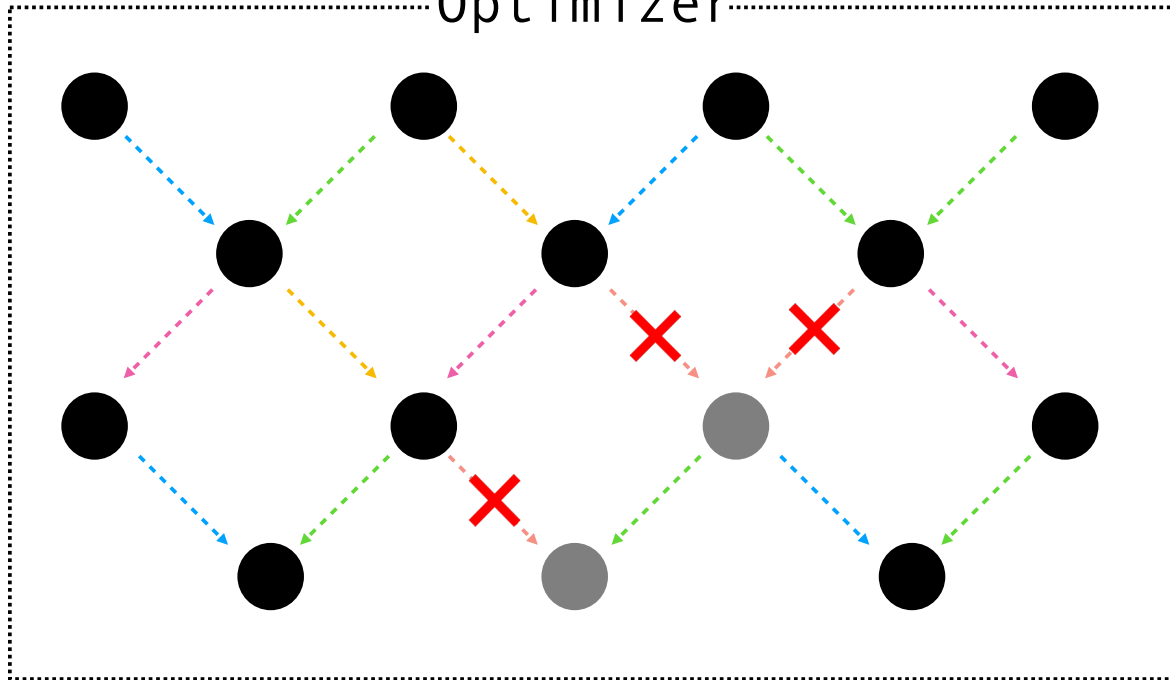
Query (job)

```
in0 = SELECT * FROM sales WHERE ...;  
in1 = SELECT * FROM product WHERE ...;  
SELECT A, COUNT(*) FROM in0 JOIN in1  
GROUP BY ...;
```

Rules



Optimizer



plan

Search



rule

Optimizer 10,000 feet POV:

- *Rules used to transform query graph*
- *Rule configuration dictates the search space*

Intuition **Cheapest Plan**

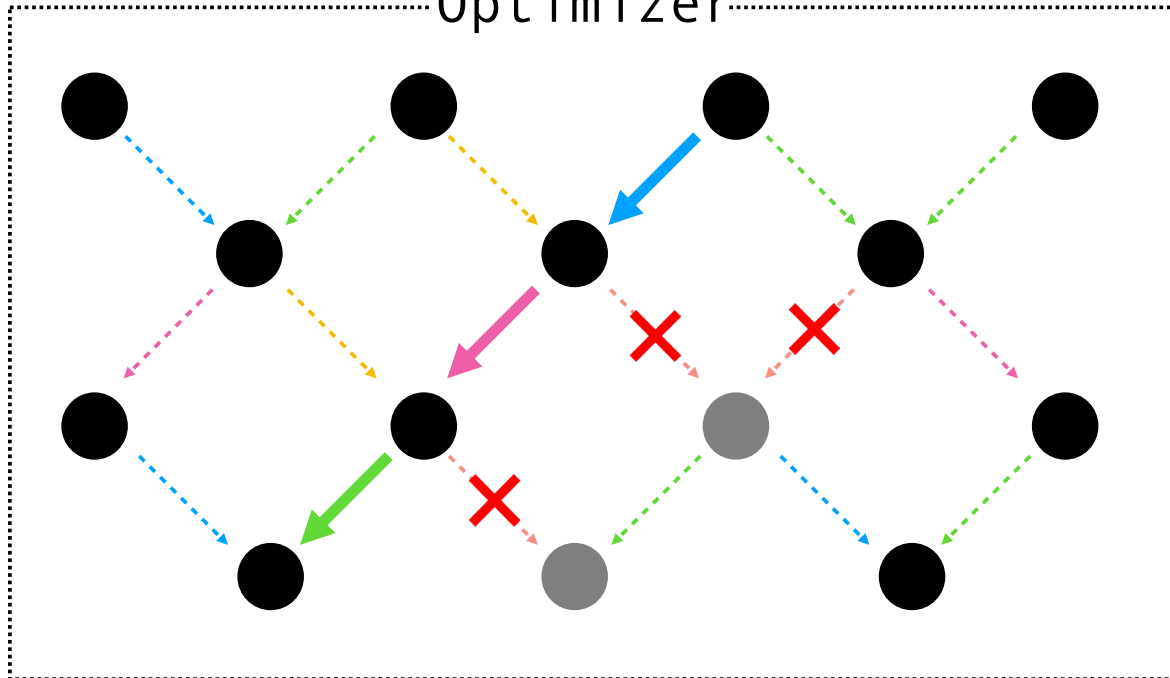
Query (job)

```
in0 = SELECT * FROM sales WHERE ...;  
in1 = SELECT * FROM product WHERE ...;  
SELECT A, COUNT(*) FROM in0 JOIN in1  
GROUP BY ...;
```

Rules



Optimizer



Search



Optimizer 10,000 feet POV:

- *Rules used to transform query graph*
- *Rule configuration dictates the search space*

Intuition **What if cheapest plan is not optimal?**

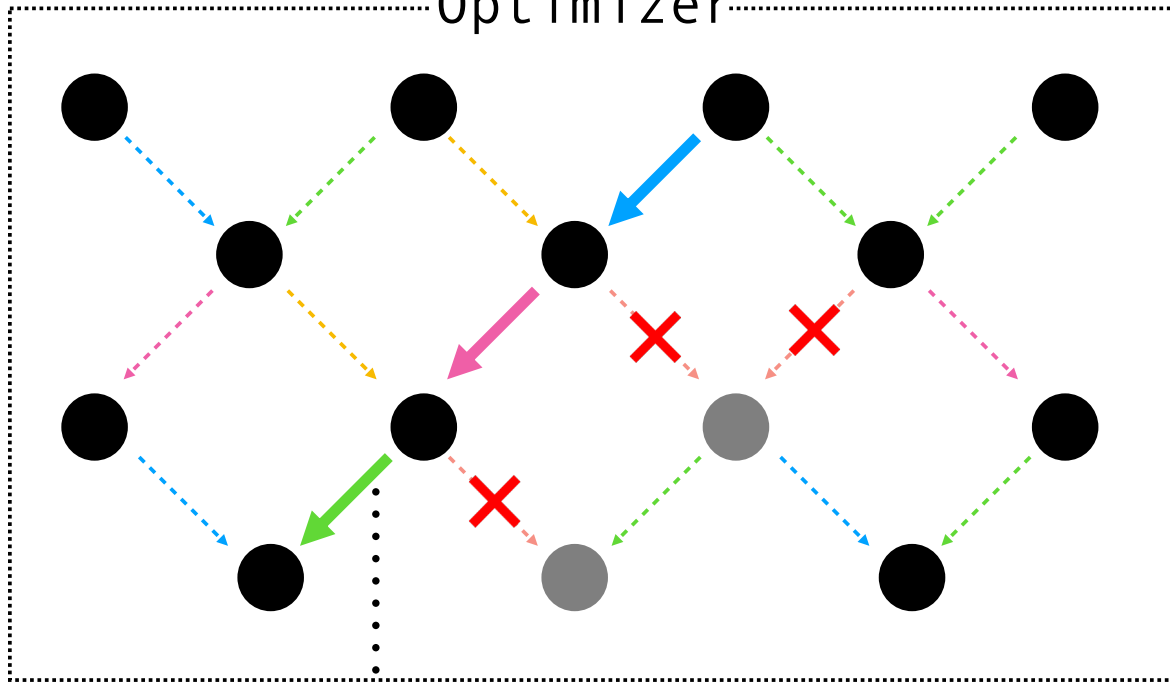
Query (job)

```
in0 = SELECT * FROM sales WHERE ...;  
in1 = SELECT * FROM product WHERE ...;  
SELECT A, COUNT(*) FROM in0 JOIN in1  
GROUP BY ...;
```

Rules



Optimizer



Search



Not optimal due to mistakes in the *heuristics*,
cardinalities, *costs*, other assumptions

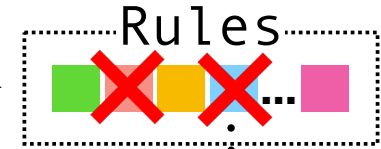
Intuition **Disabling Rules**

Query (job)

```
in0 = SELECT * FROM sales WHERE ...;  
in1 = SELECT * FROM product WHERE ...;  
SELECT A, COUNT(*) FROM in0 JOIN in1  
GROUP BY ...;
```

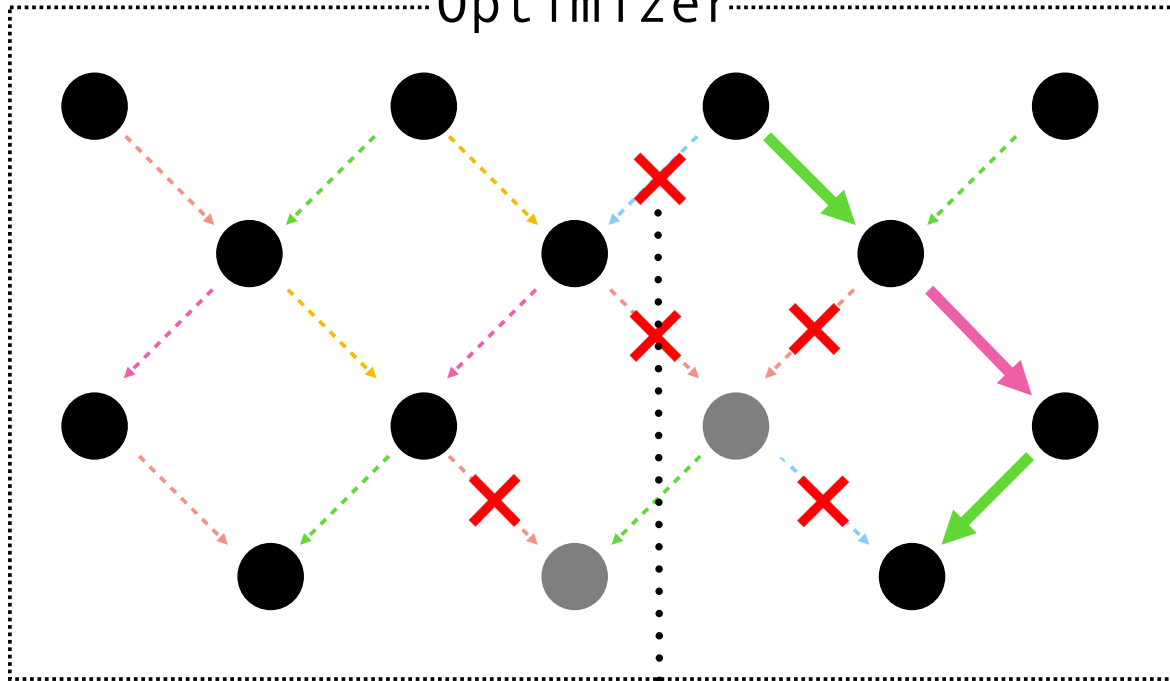


Rule Configuration
// optimizer hint disabling rule 20
-optFlags DR(20);



Disable rule

Optimizer



Block bad path

Search



Intuition **Enabling Rules**

Query (job)

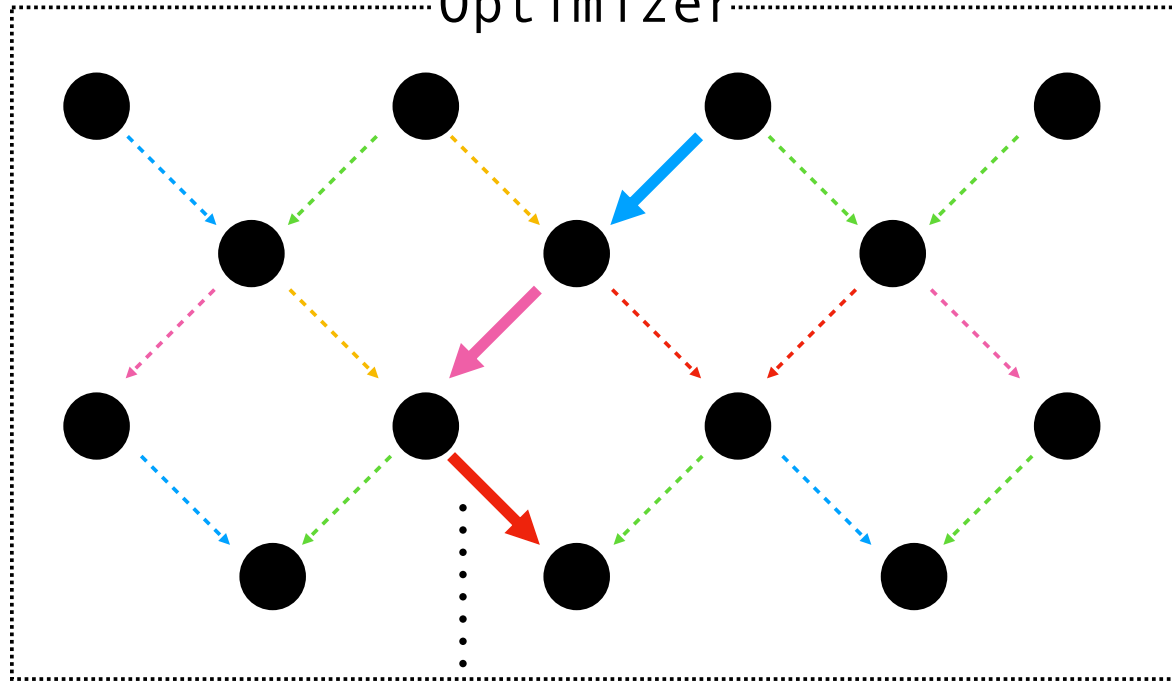
```
in0 = SELECT * FROM sales WHERE ...;  
in1 = SELECT * FROM product WHERE ...;  
SELECT A, COUNT(*) FROM in0 JOIN in1  
GROUP BY ...;
```

Rules



Enable rule

Optimizer



Search



Enable previously unreachable
search space

Intuition **Steering Query Optimizer**

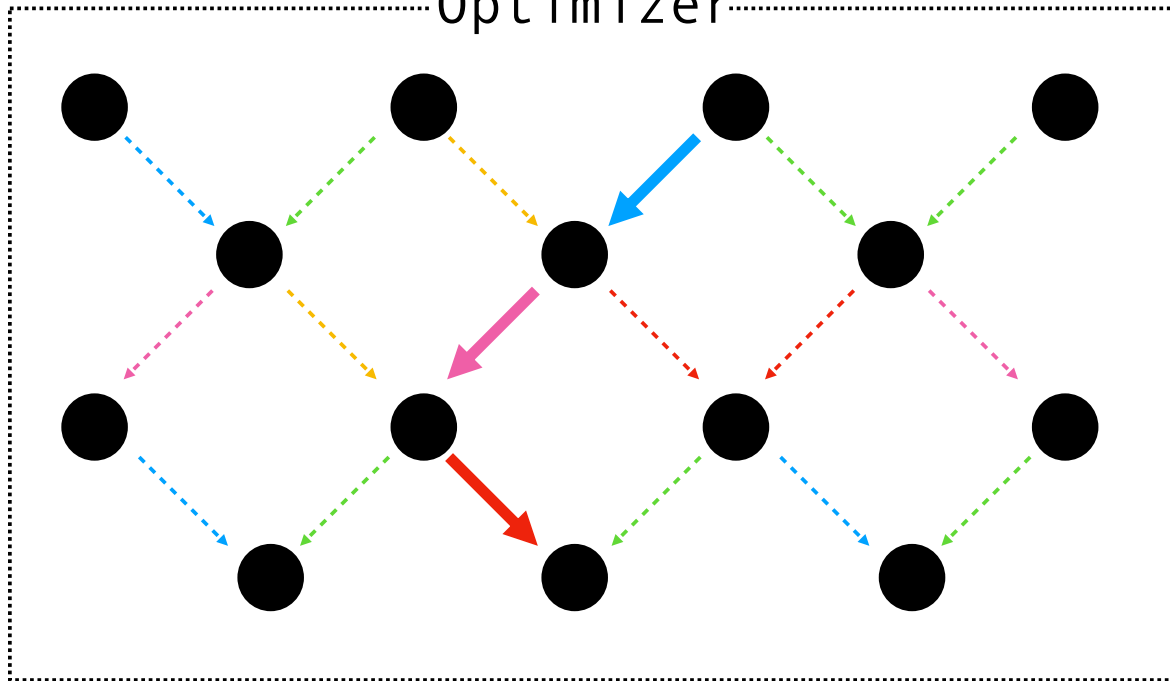
Query (job)

```
in0 = SELECT * FROM sales WHERE ...;  
in1 = SELECT * FROM product WHERE ...;  
SELECT A, COUNT(*) FROM in0 JOIN in1  
GROUP BY ...;
```

Rules



Optimizer



Search



Scope users already use rule hints to fine tune queries:

- Up to ~9% of jobs contain user-provided rule hints
- Difficult to tune, requiring a lot of expertise and experimentation
- *Can we automate this at scale?*

Steering Query Optimizers

Core Techniques

Core Techniques **Rule Signature**

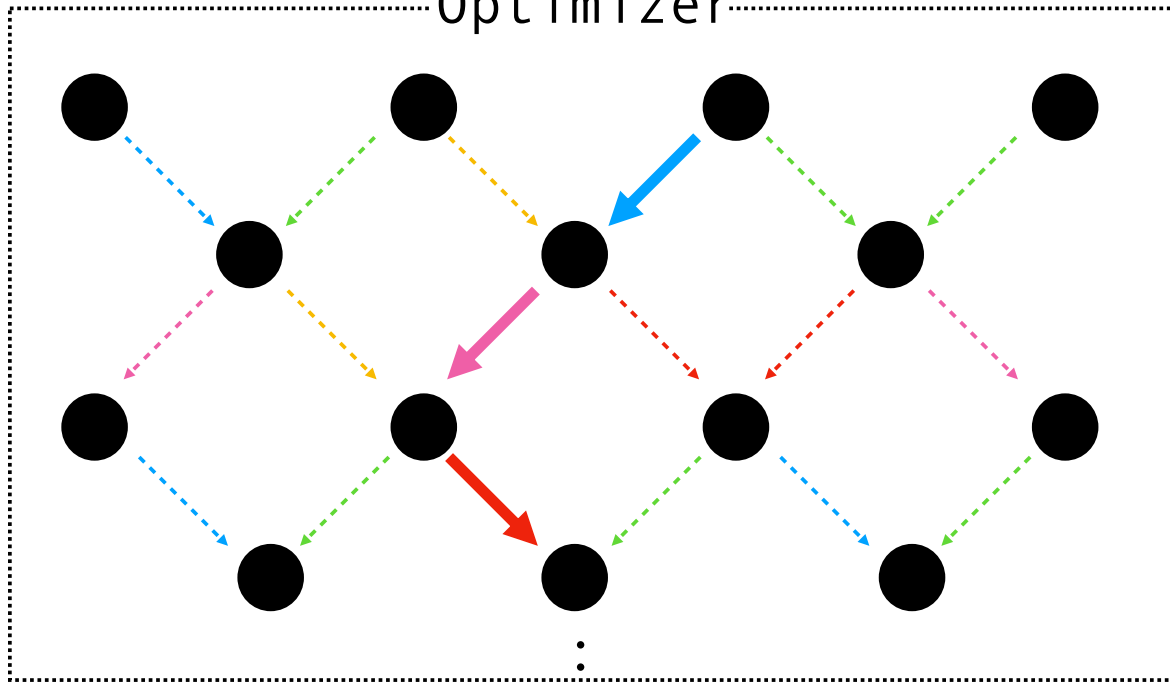
Query (job)

```
in0 = SELECT * FROM sales WHERE ...;  
in1 = SELECT * FROM product WHERE ...;  
SELECT A, COUNT(*) FROM in0 JOIN in1  
GROUP BY ...;
```

Rules



Optimizer



Search



Rule Signature



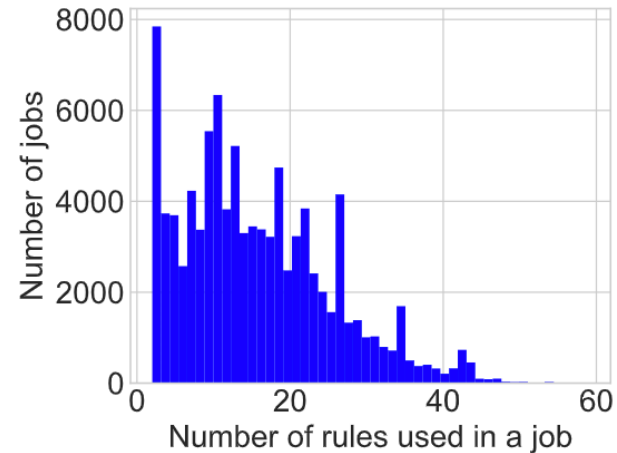
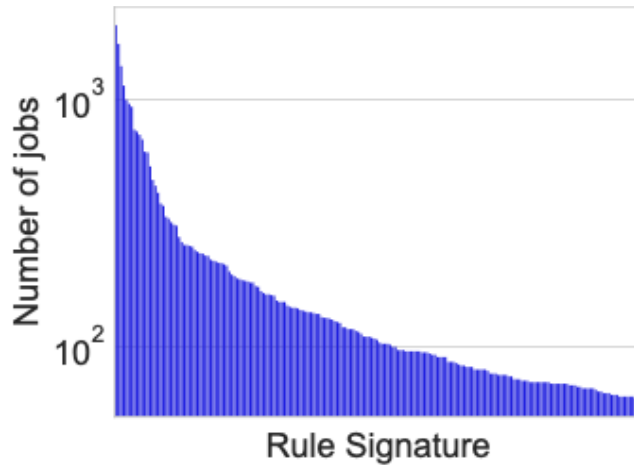
Define which rule is actually used to reach the final query plan

Core Techniques **Rule Signature**

Query (job)

```
in0 = SELECT * FROM sales WHERE ...;  
in1 = SELECT * FROM product WHERE ...;  
SELECT A, COUNT(*) FROM in0 JOIN in1  
GROUP BY ...;
```

Rules



2^{256} potential rule signatures. But it has a lot of structure!

Search

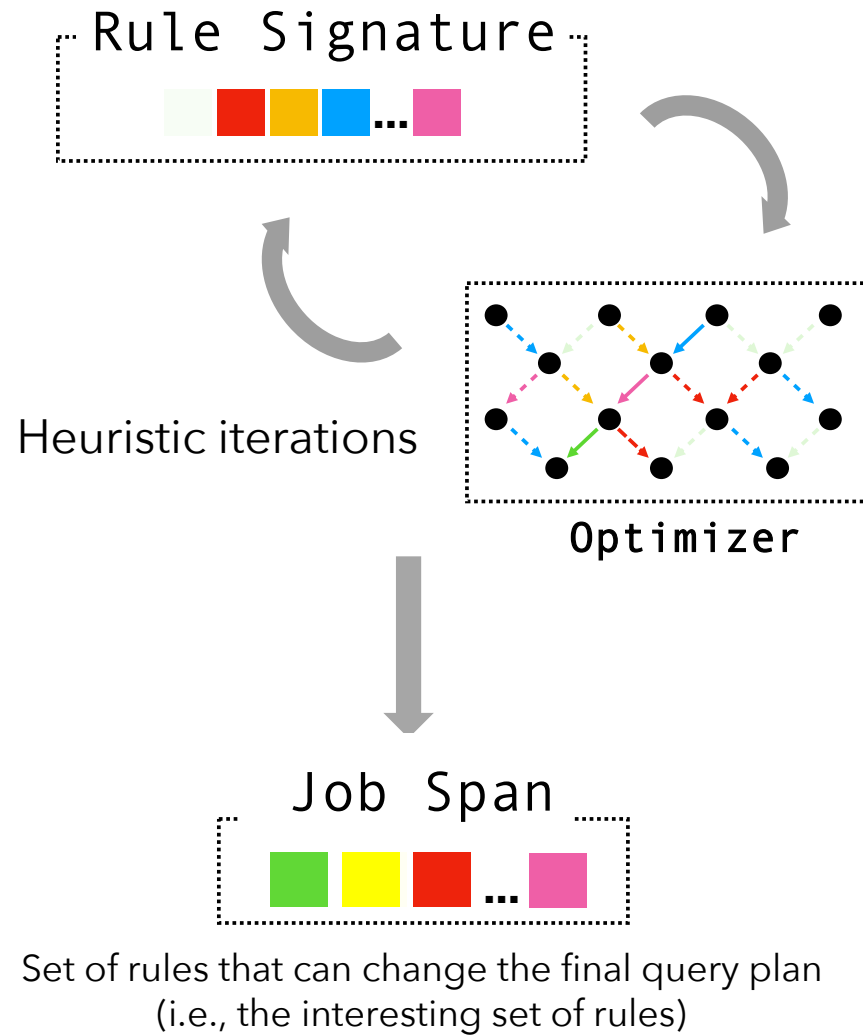


Rule Signature



Define which rule is actually used to reach the final query plan

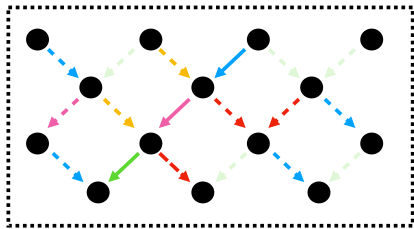
Core Techniques **Job Span**



Steering Query Optimizers

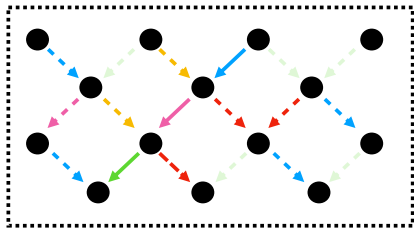
The Automation

Automation **As RL Problem**

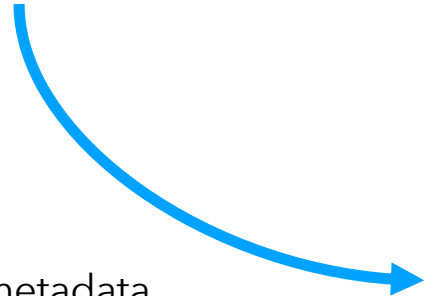


Optimizer

Automation **As RL Problem**



Optimizer

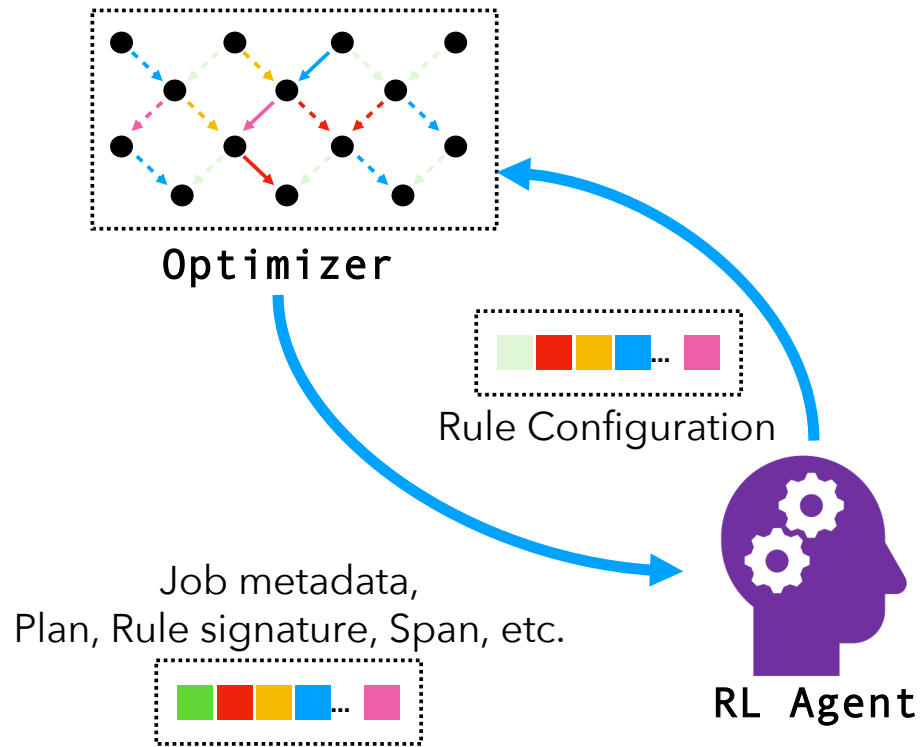


Job metadata,
Plan, Rule signature, Span, etc.

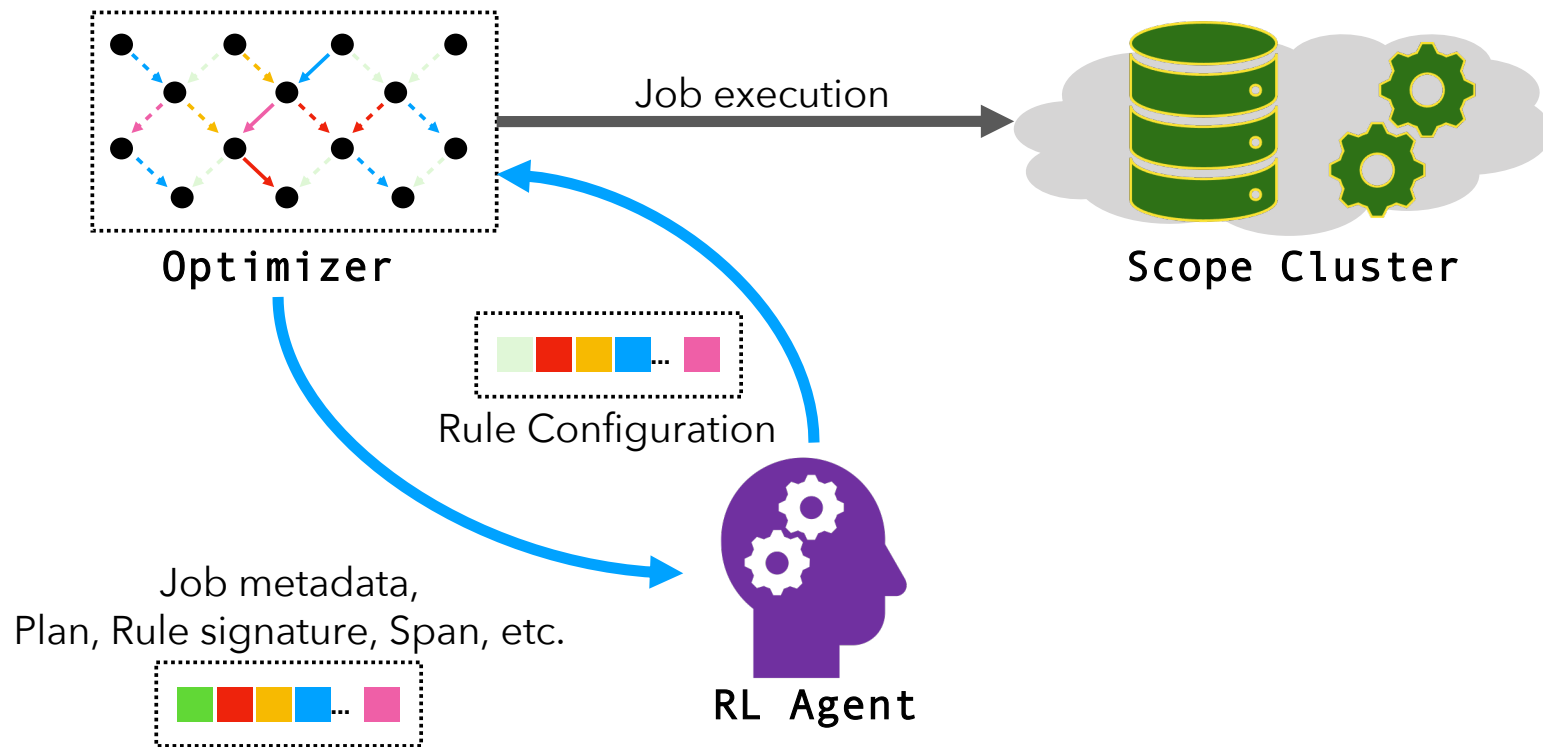


RL Agent

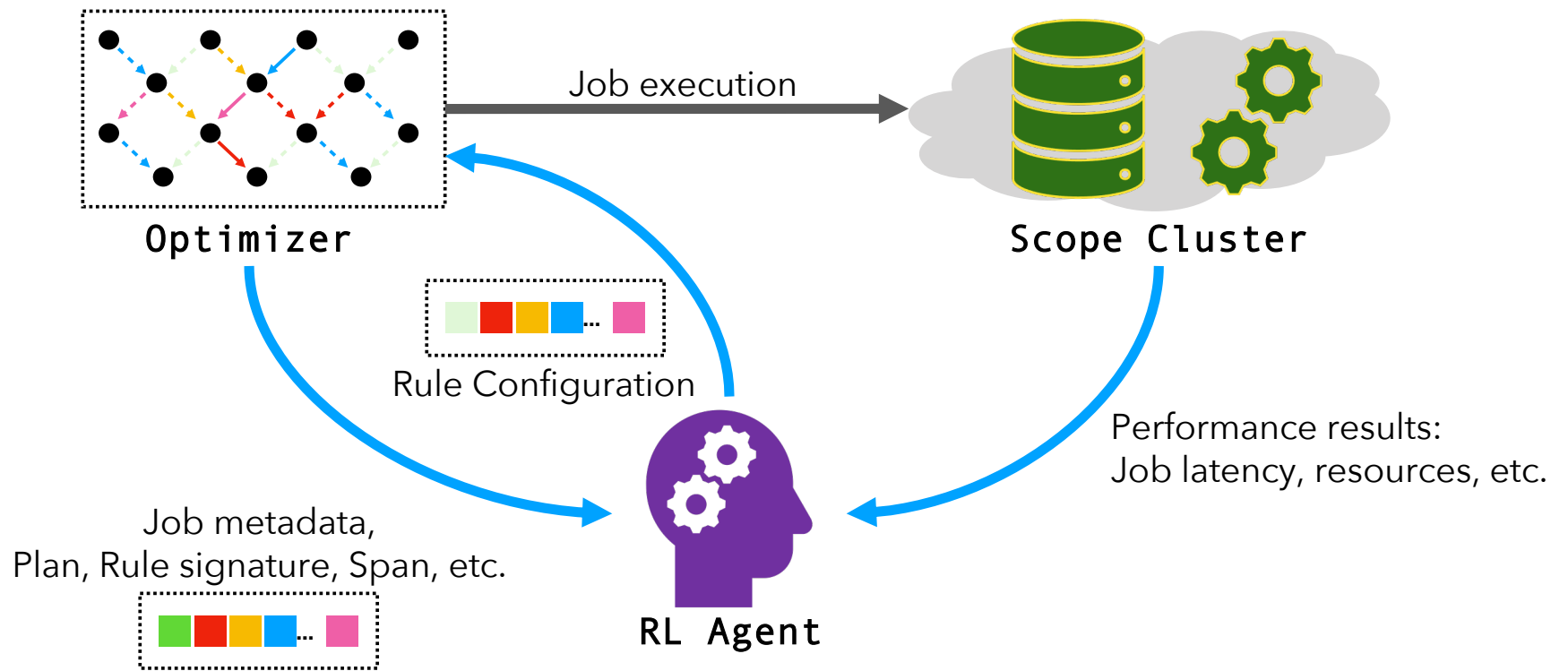
Automation **As** RL Problem



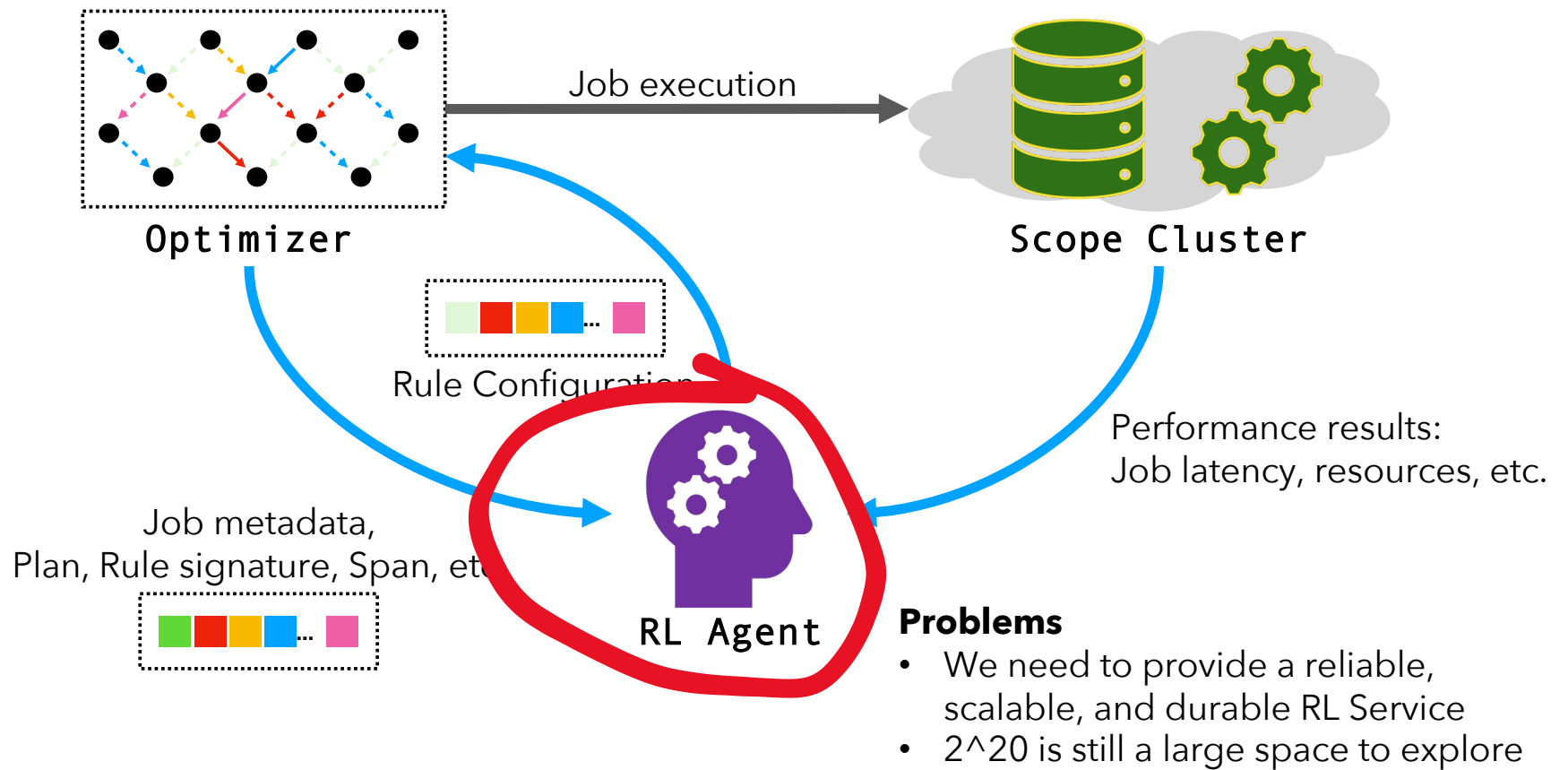
Automation **As** RL Problem



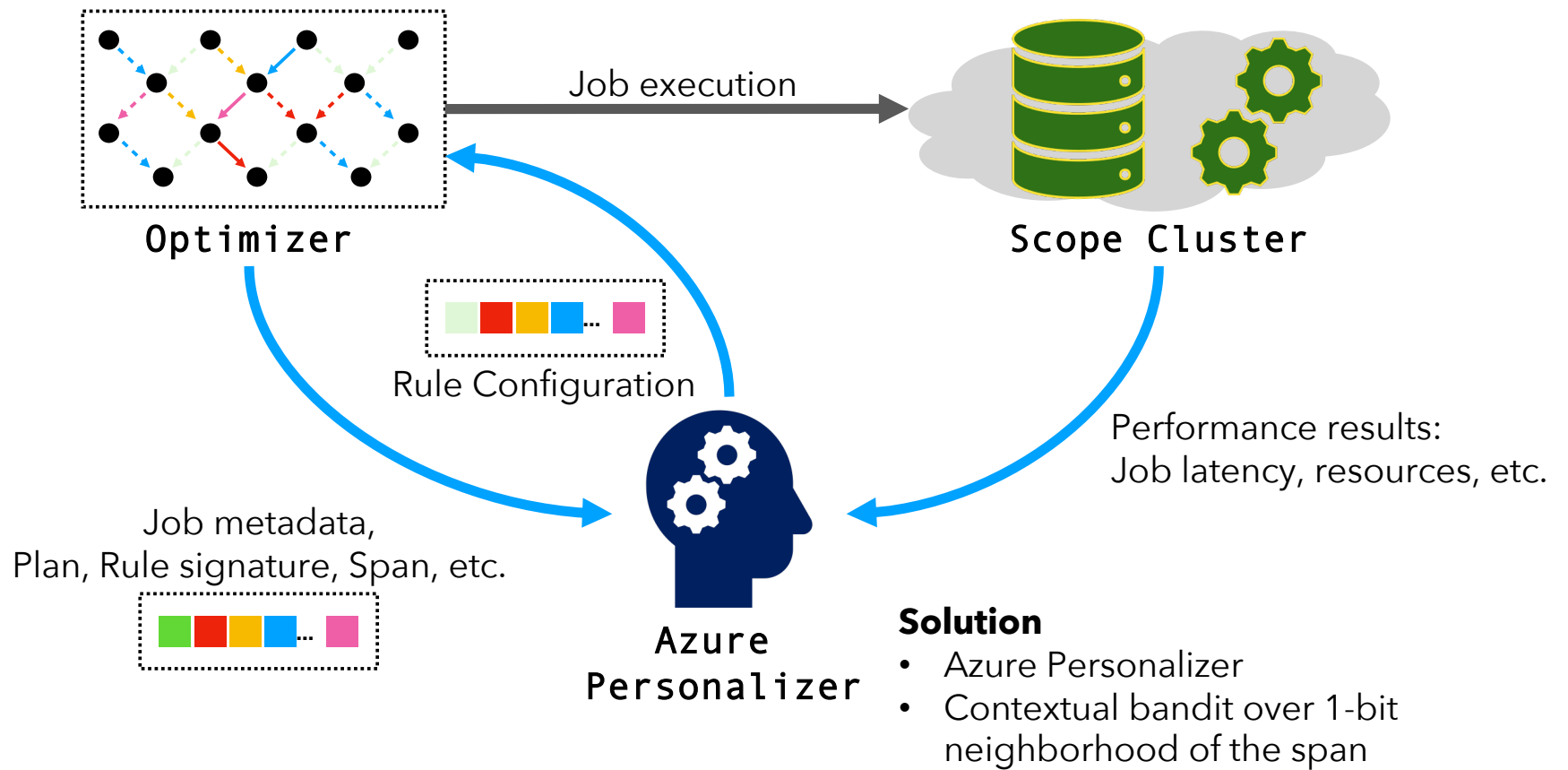
Automation **As RL Problem**



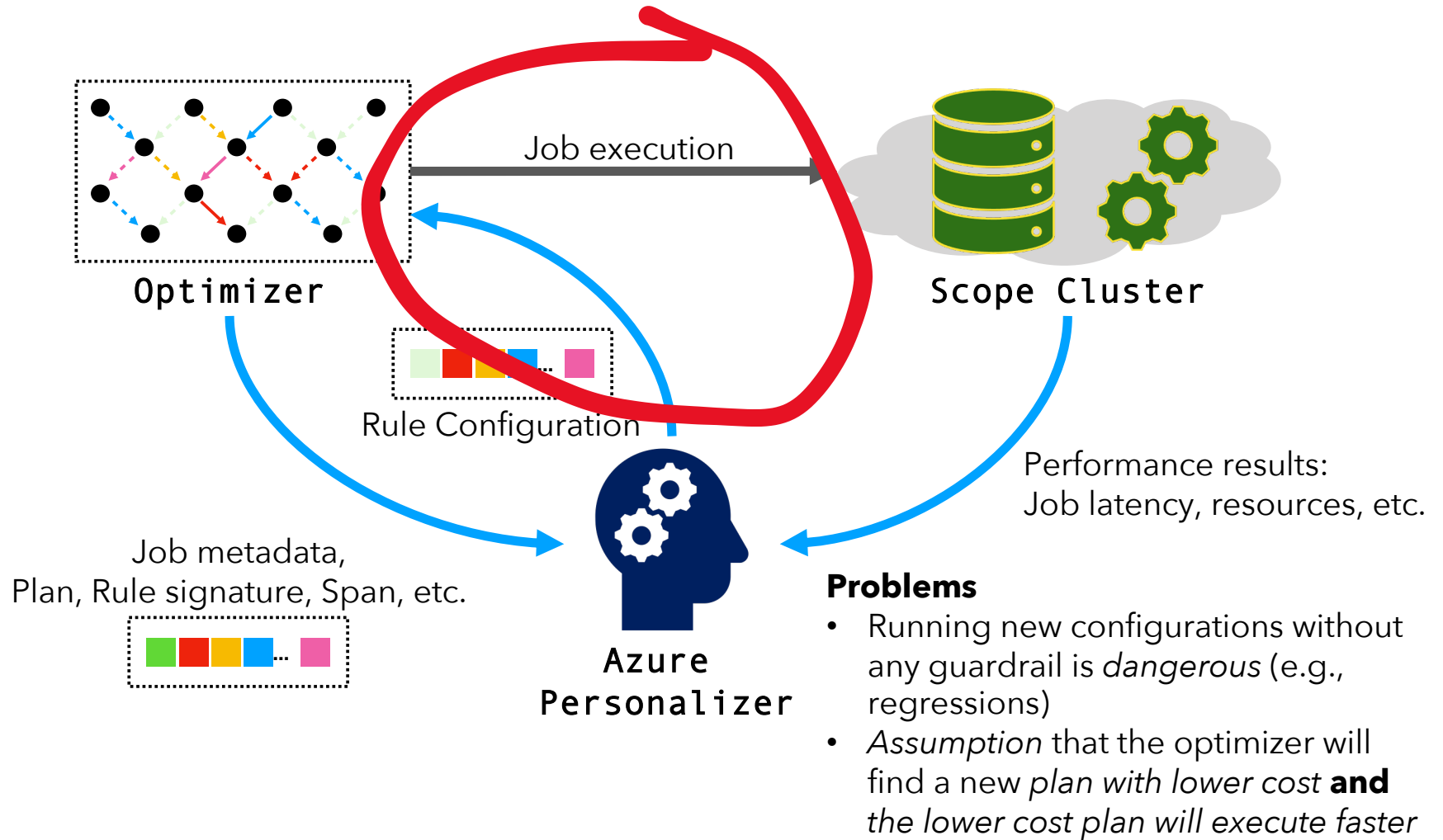
Automation **As** RL Problem



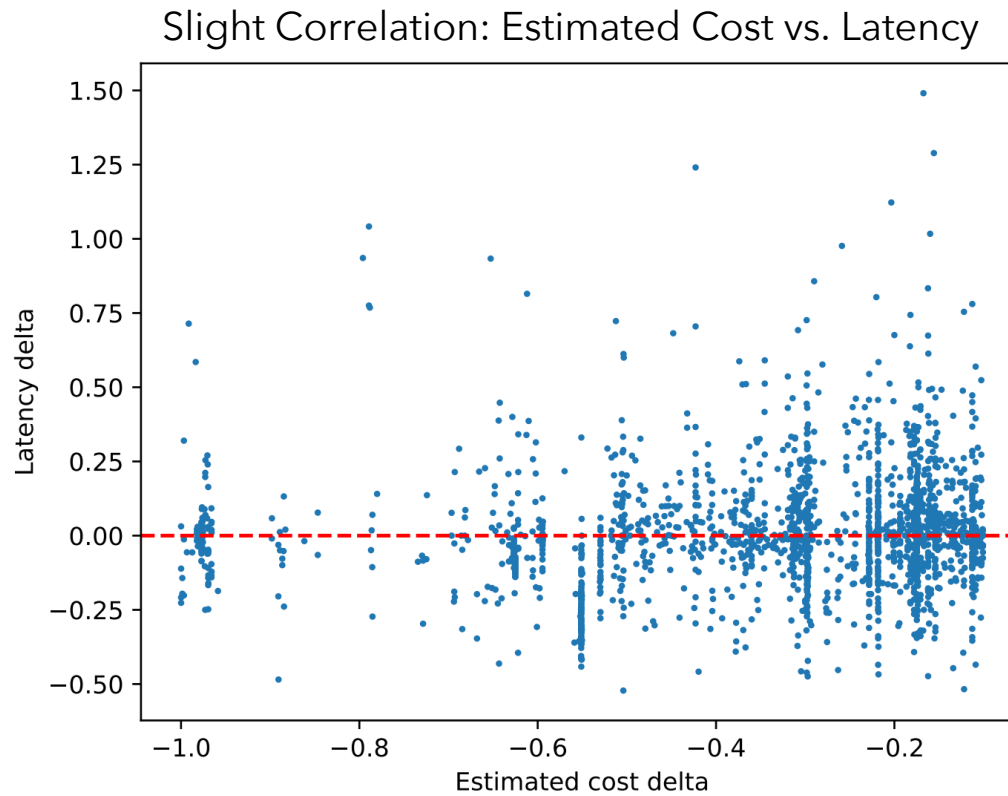
Automation **As RL Problem**



Automation **As RL Problem**



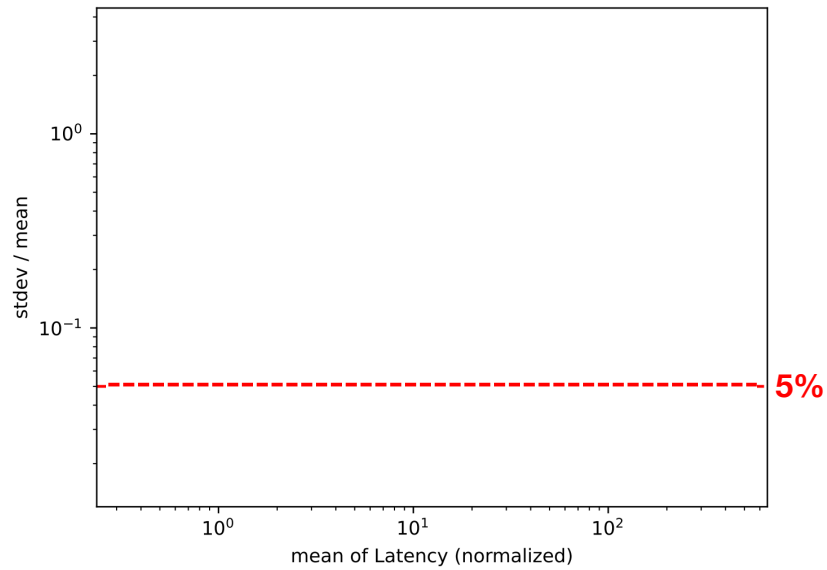
Automation **Lower Costs are not enough**



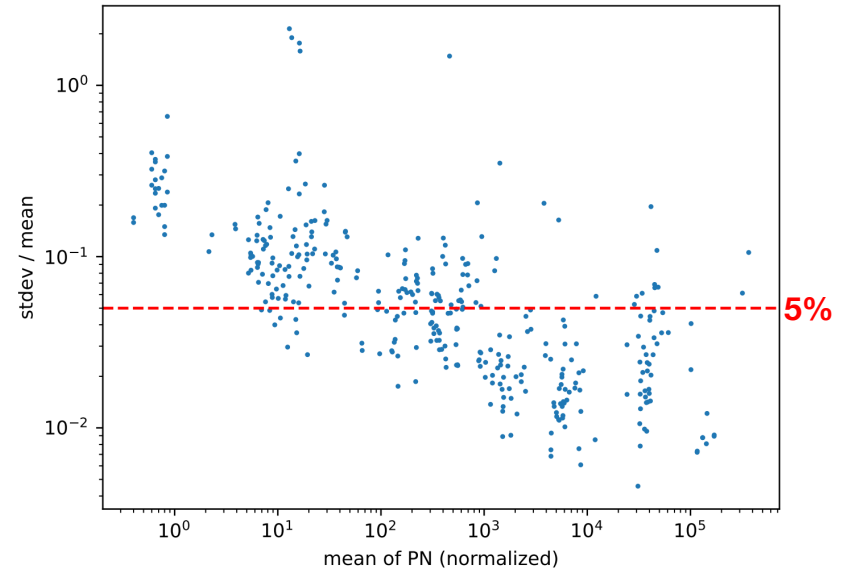
Conclusion: We need runtime information to avoid regressions. **But ...**

Automation **High Variance makes Learning Difficult**

Latency

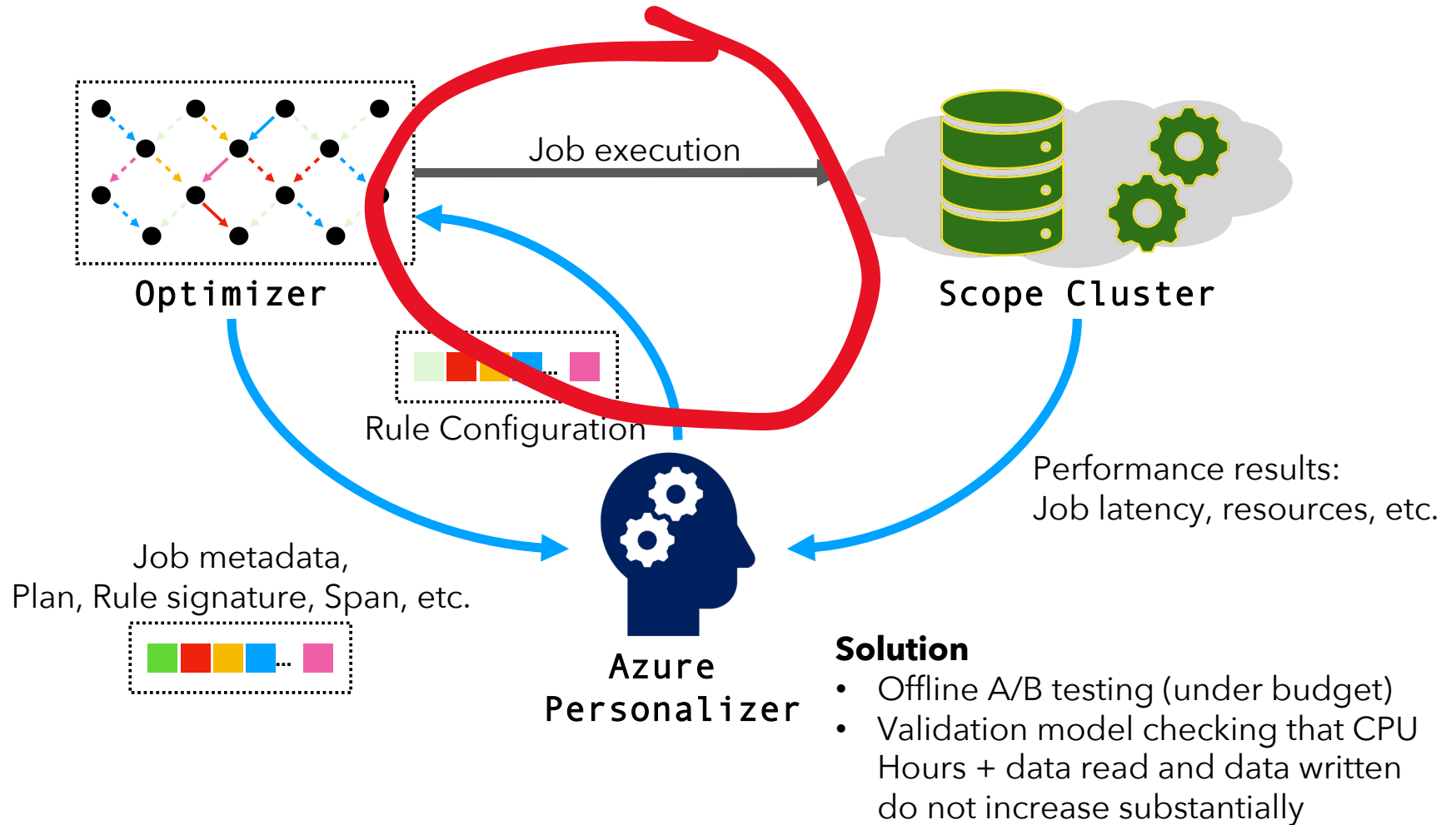


CPU Hours

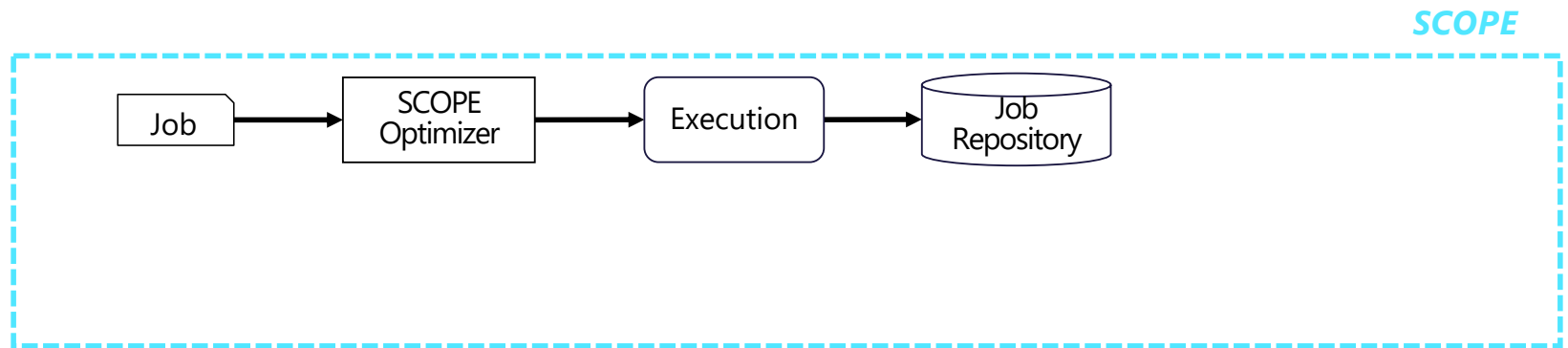


Run the same job on same data 10 times

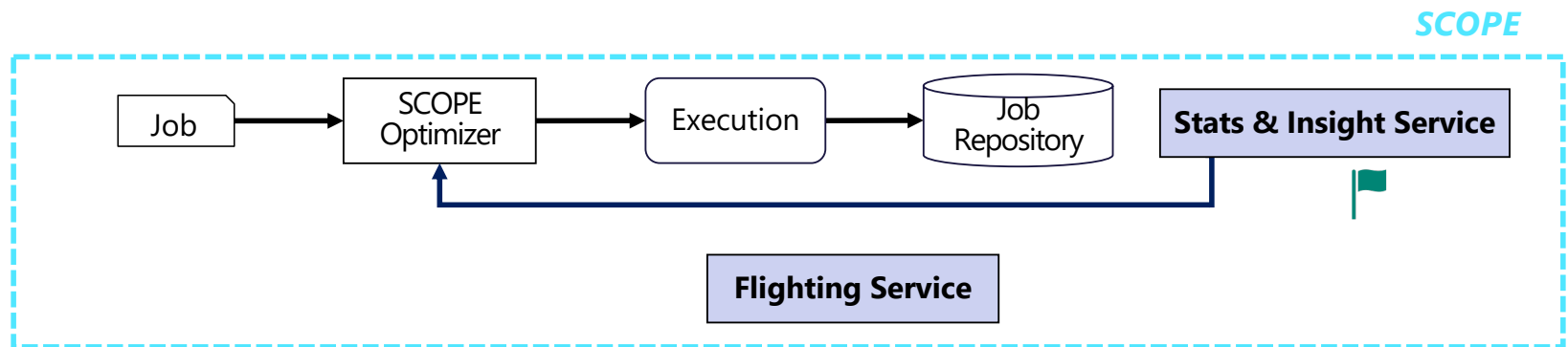
Automation **As RL Problem**





Automation **Let's make it Concrete**



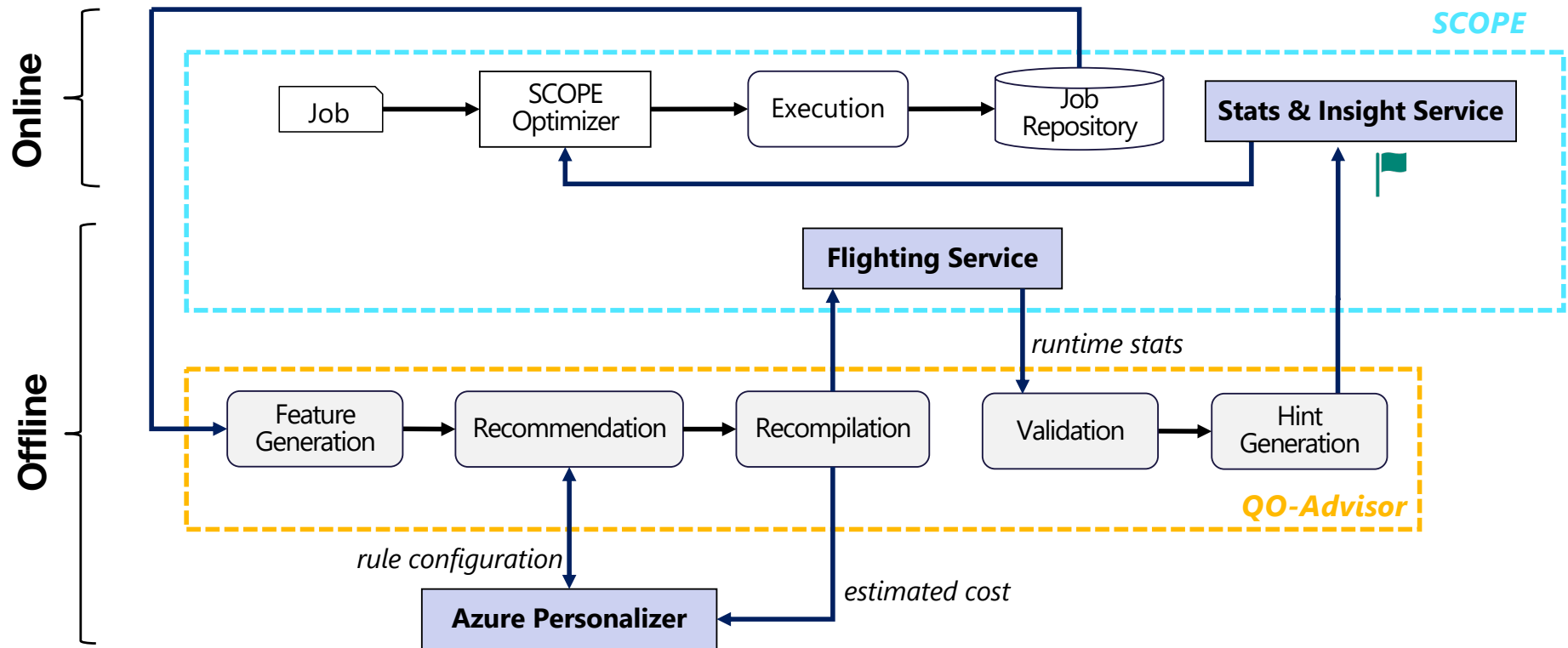
Automation **Let's make it Concrete**



Flying Service: A/B testing of different query plans

Stats & Insight Service: Serving machine learning models and hints (e.g.,  vs )

Automation **Let's make it Concrete**

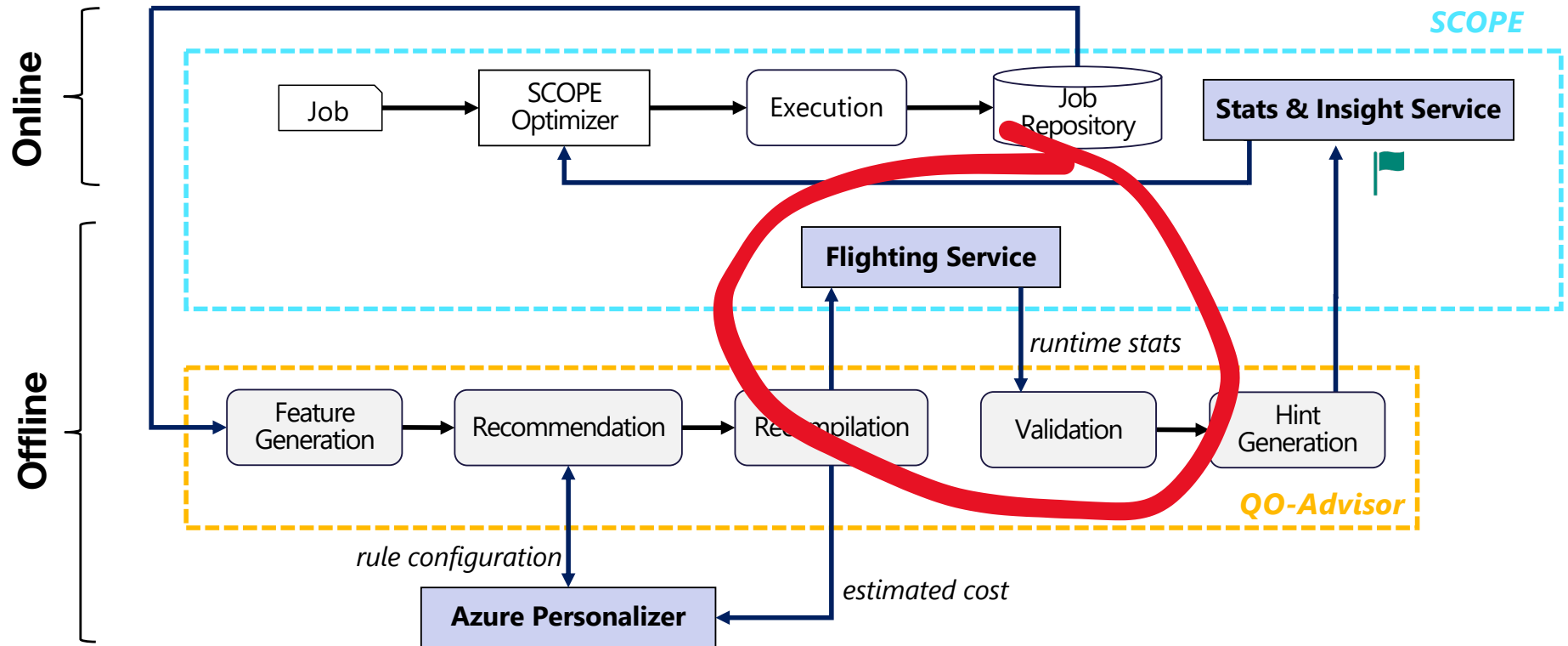


- Pipeline trained daily on ~5% of Scope jobs
- A pipeline run takes about 24h and about 500 vcores
- For jobs using QO-Advisor, 14% overall improvement in CPU Hours
- Generate ~200k events per day for Azure Personalizer

Steering Query Optimizers

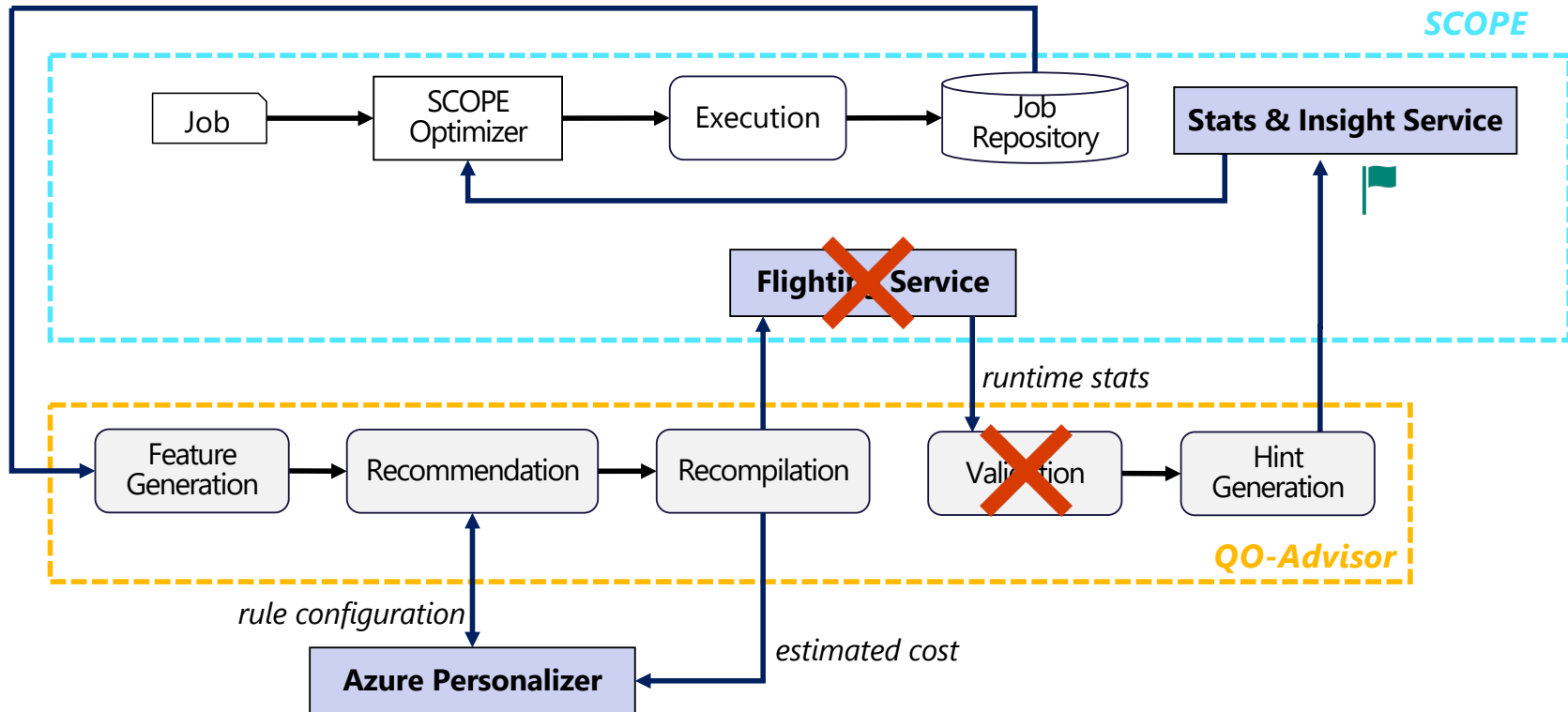
Scaling

Scaling **Flighting** Limitation

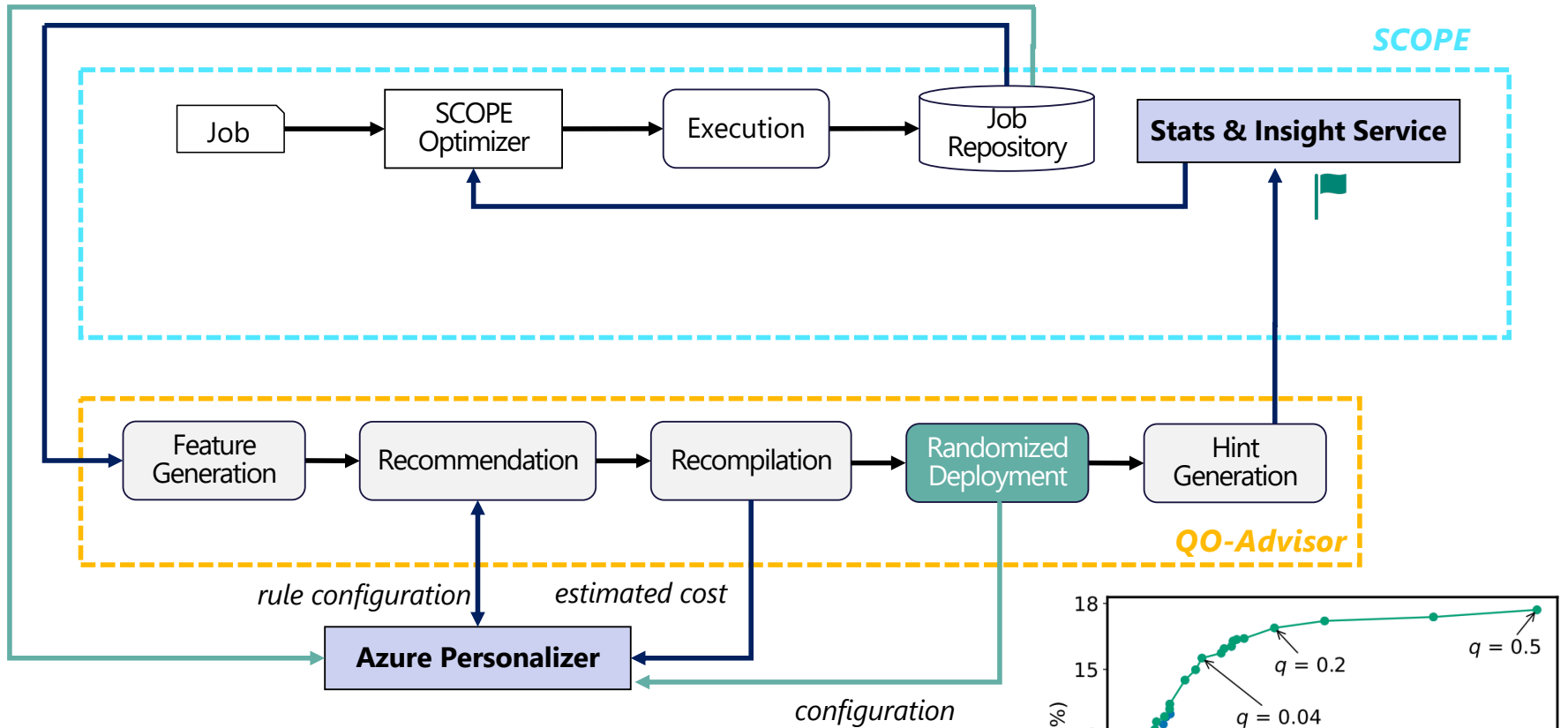


Flighting is limiting the number of jobs we can optimize
Can we replace offline validation with **online randomized A/B testing**?

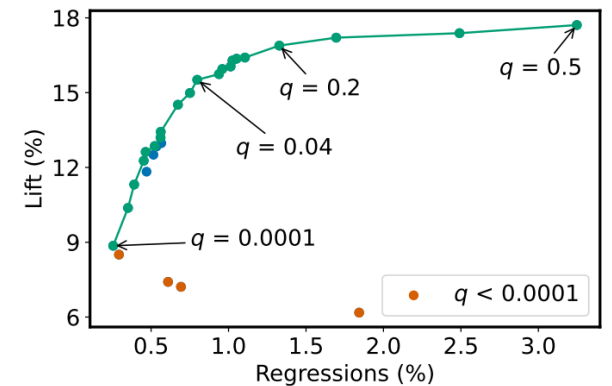
Scaling **Learning over Repeated Runs**



Scaling **Learning over Repeated Runs**

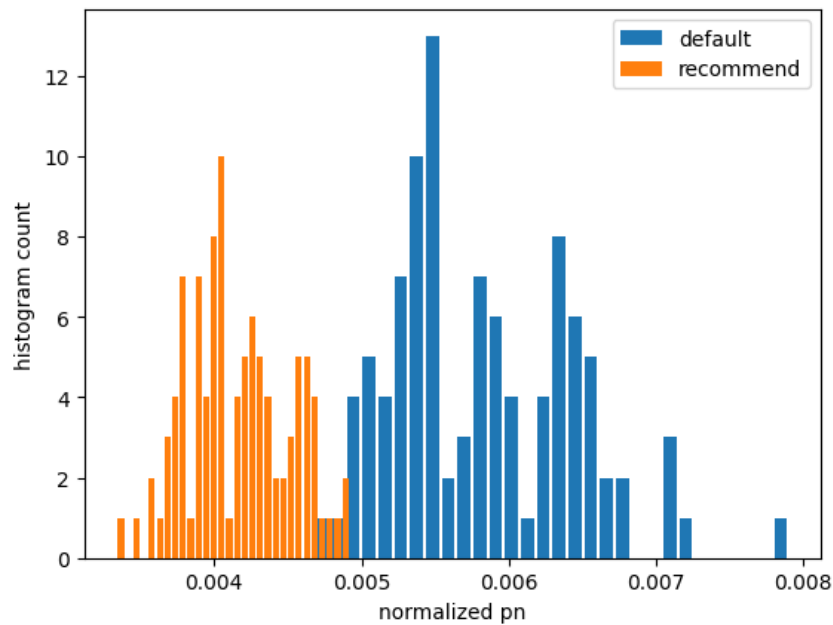


- Informed randomized deployment of configurations
- Risk-adverse contextual bandit



Scaling **Randomized Online A/B Testing Results**

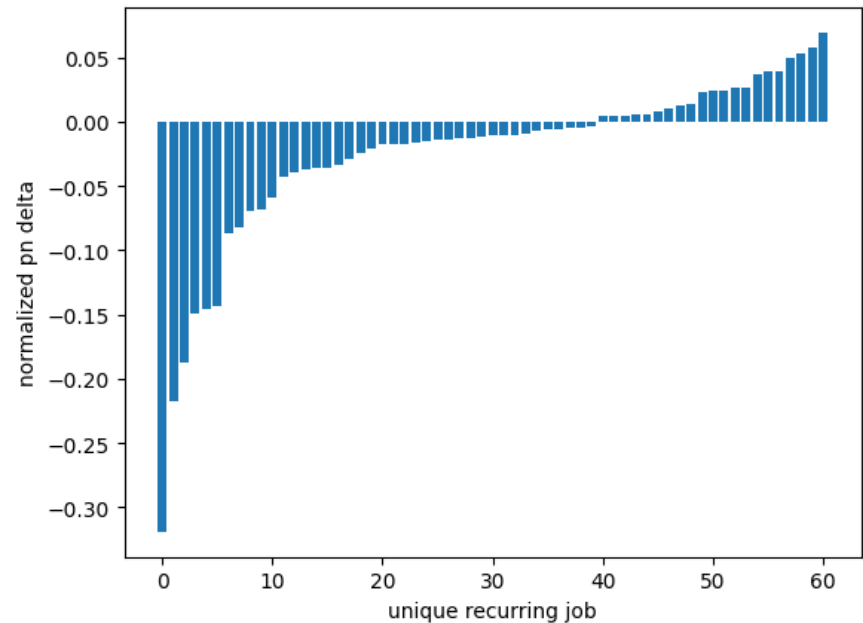
Performance of a cherry-picked job



$\text{normalized_pn} = \text{CPU_hours} / \text{input_data_size}$

Hint improves normalized PN
by ~25% for this particular job

Performance for each unique recurring job



“delta”: lower is better ($\text{PN_recommend} / \text{PN_default} - 1$)

Informed randomization + risk-adverse
algorithm lowers the chances of regressions

Scaling **Summary**

- *Pipeline runtime and resource utilizations*
 - From 24h to ~3h
 - Only uses vcores for recompilation and generating estimates

- *Job Coverage:*

- We train over all Scope jobs
- On average, apply hint on 4.84% of recurring jobs

→ Not as high as before due to risk-adverse algorithm

- *CPU Hours Improvement:*

- For jobs using QO-Advisor, overall improvement: 7.29%
- Small regressions for ~30% jobs

→ More regressions due to online exploration after removing offline validation step

- *We decided to **shut down the project after ~3 years***

- Total impact on Scope workload was less than expected
- Regressions were a major source of concern
- Azure Personalizer got discontinued

Conclusion/Key Takeaways

1. **This is a hard problem:** state-of-the-art optimizers are hard to beat on average
2. **Inaccurate cost estimates:** can be solved with validation (but it's expensive)
3. **But cannot really ignore cost estimates:** ML models make mistakes
4. **Noisy performance numbers:** ML models have hard time converging
5. **Sparse data:** This is the cost of safety
6. **Online exploration does not work well if you must avoid regressions:** Offline validation is better (but it's expensive)
7. **Featurization of database jobs:** The embedding-based approaches might have a great potential, but we didn't have the time to investigate this path

Thank You!

Wangda Zhang, Paul Mineiro, Shi Qiao, Nasim Ghazanfari, Karlen Lie, Marc Friedman, Rafah Hosn, Hiren Patel, Alekh Jindal, Parimarjan Negi, Ryan Marcus, Mohammad Alizadeh, Tim Kraska