Steering Query Optimizers: A Practical Take on Big Data Workloads

Parimarjan Negi, Matteo Interlandi, Ryan Marcus,
Mohammad Alizadeh, Tim Kraska, Marc Friedman,
Alekh Jindal

Overview **Bao**

Main Idea (Bao, [1]): Introduce learning to the PostgreSQL optimizer by adaptively disabling some rules (e.g., disable nested loop joins)

Easily adapted to new optimizer that allows hints to disable rules, and utilizes the search and infrastructure of the existing optimizer

[1] Bao: Learning to Steer Query Optimizers (SIGMOD 2021). *Ryan Marcus, Parimarjan Negi, Hongzi Mao, Nesime Tatbul, Mohammad Alizadeh, Tim Kraska*

Overview **Differences to Bao**

Apply Bao's idea in ***SCOPE***, Microsoft's internal big data query processor.

A lot more rules **(200+)**

Very different optimizer and execution environment

Large real world workloads
(Execute **1000s of hours of jobs** with different rules)

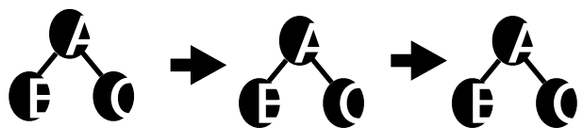
Intuition

Job

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

Optimizer

Plan



Intuition **Rule Based Optimizer**

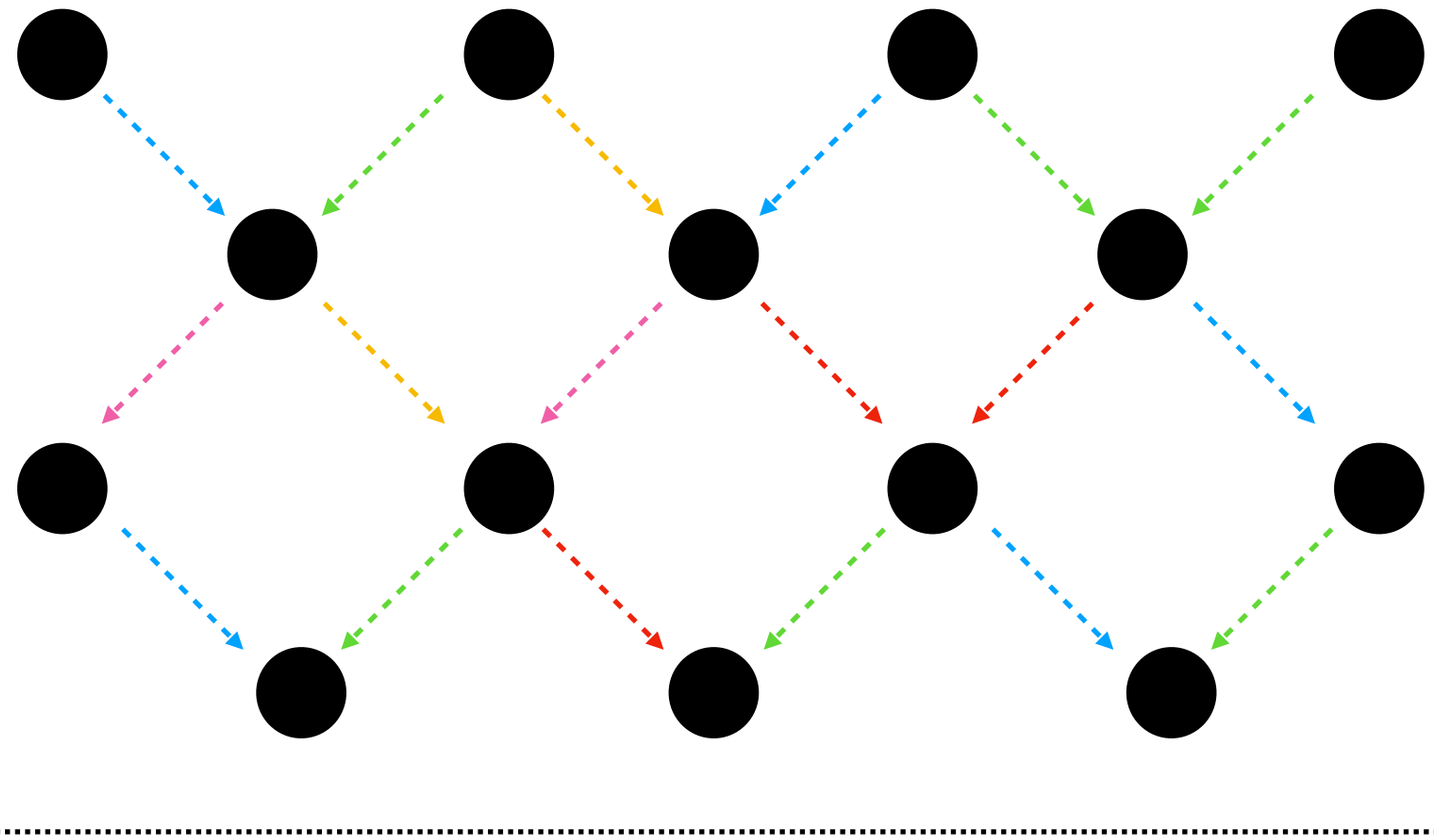
Job

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

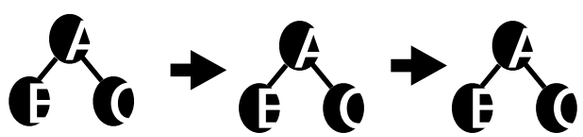
Rules

200+

Optimizer



Plan



Toy Optimizer: Rules used to transform query graph

Intuition **Cheapest Plan**

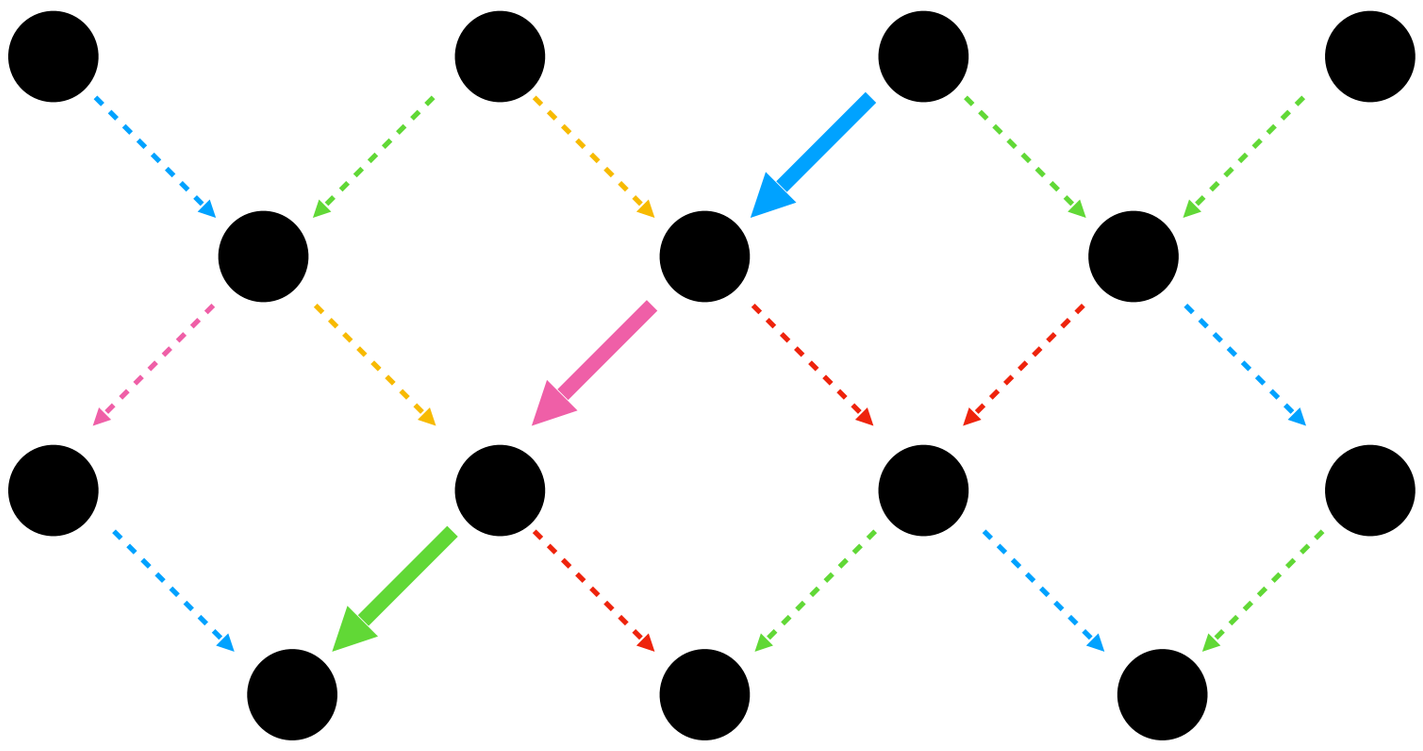
Job

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

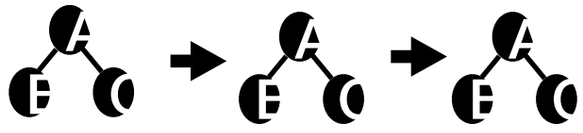
Rules

200+

Optimizer



Plan



Intuition **Rule Signature**

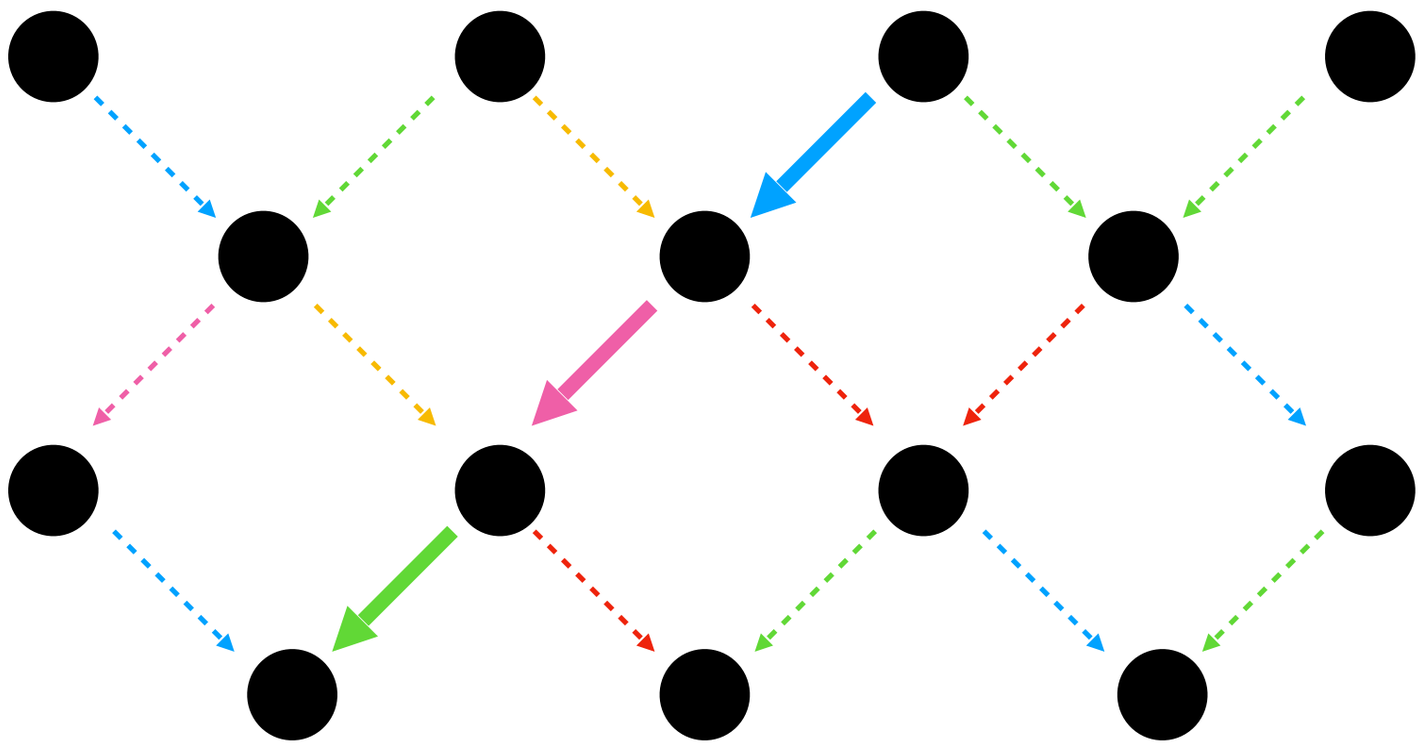
Job

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

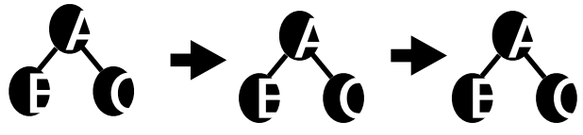
Rules

200+

Optimizer



Plan



Which rules actually used for final plan?

Rule Signature

Intuition **Rule Signature**

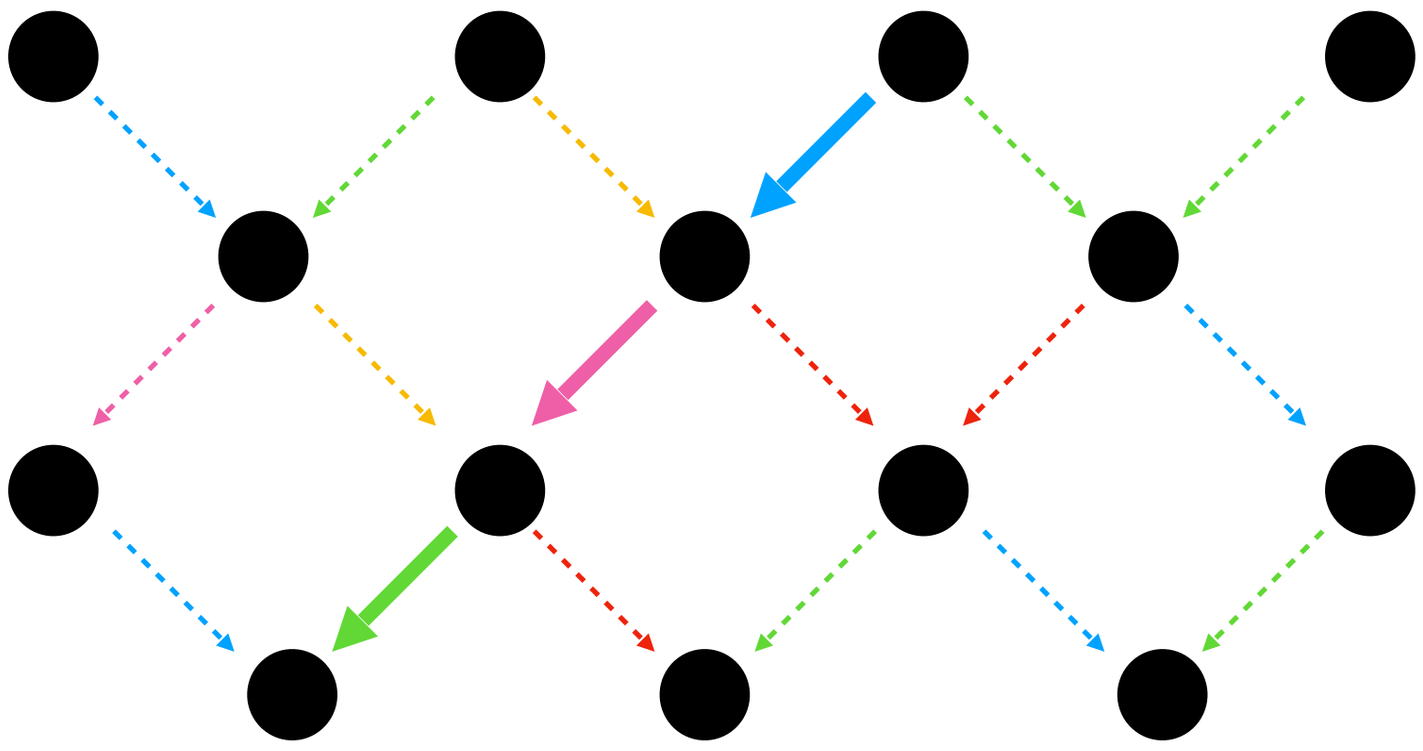
Job

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

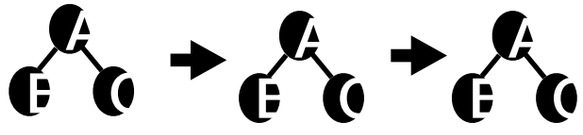
Rules

200+

Optimizer



Plan



Helps find:
interesting rules, similar jobs

Rule Signature

Intuition **What if cheapest plan is bad?**

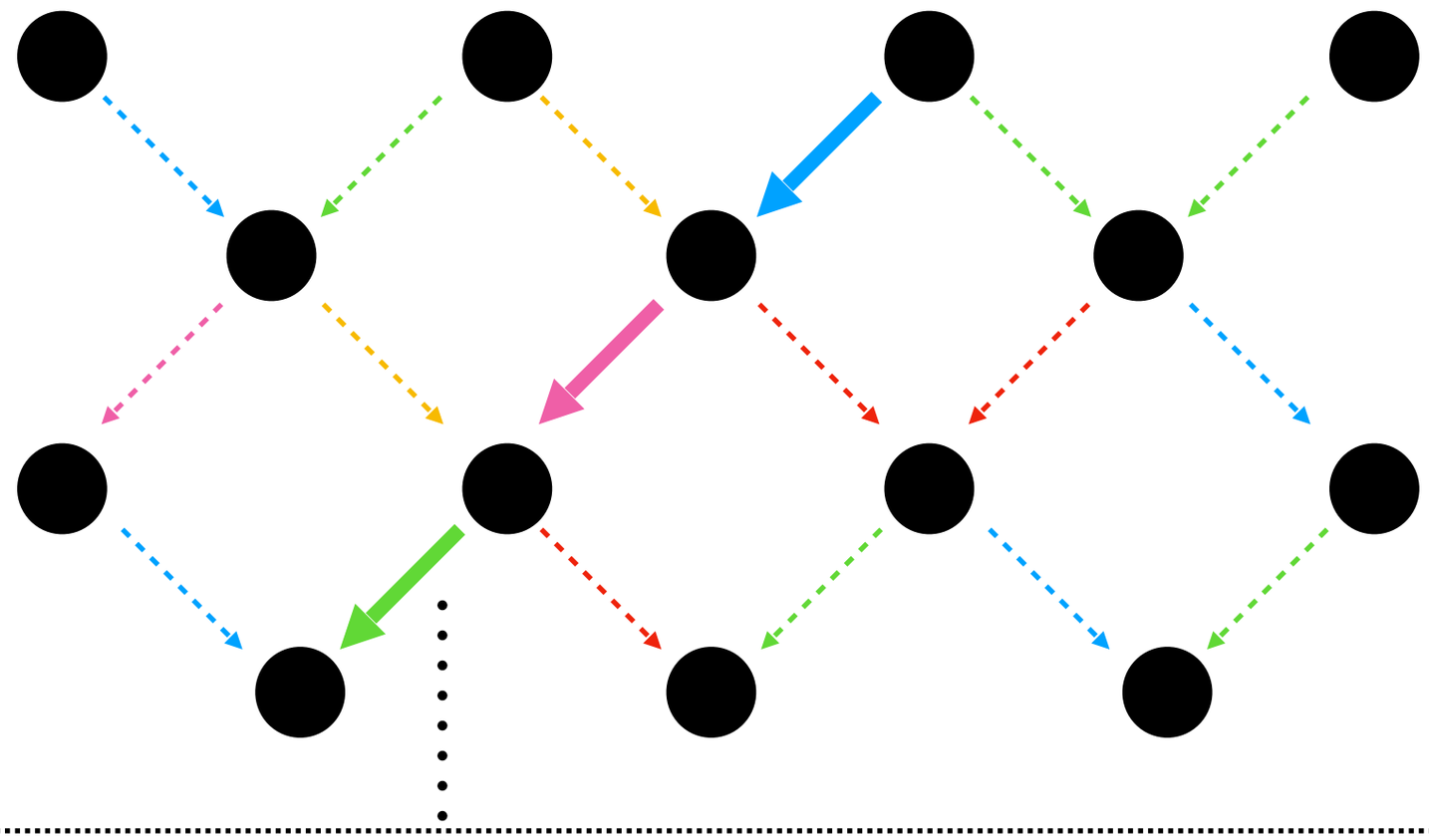
Job

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

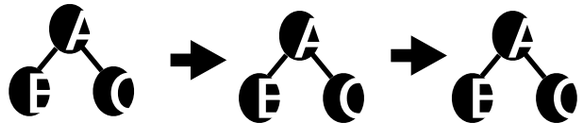
Rules

200+

Optimizer



Plan



Actually bad due to
mistakes in cardinalities, costs,
other assumptions

Hard to fix!

Rule Signature

Intuition **Disabling Rules**

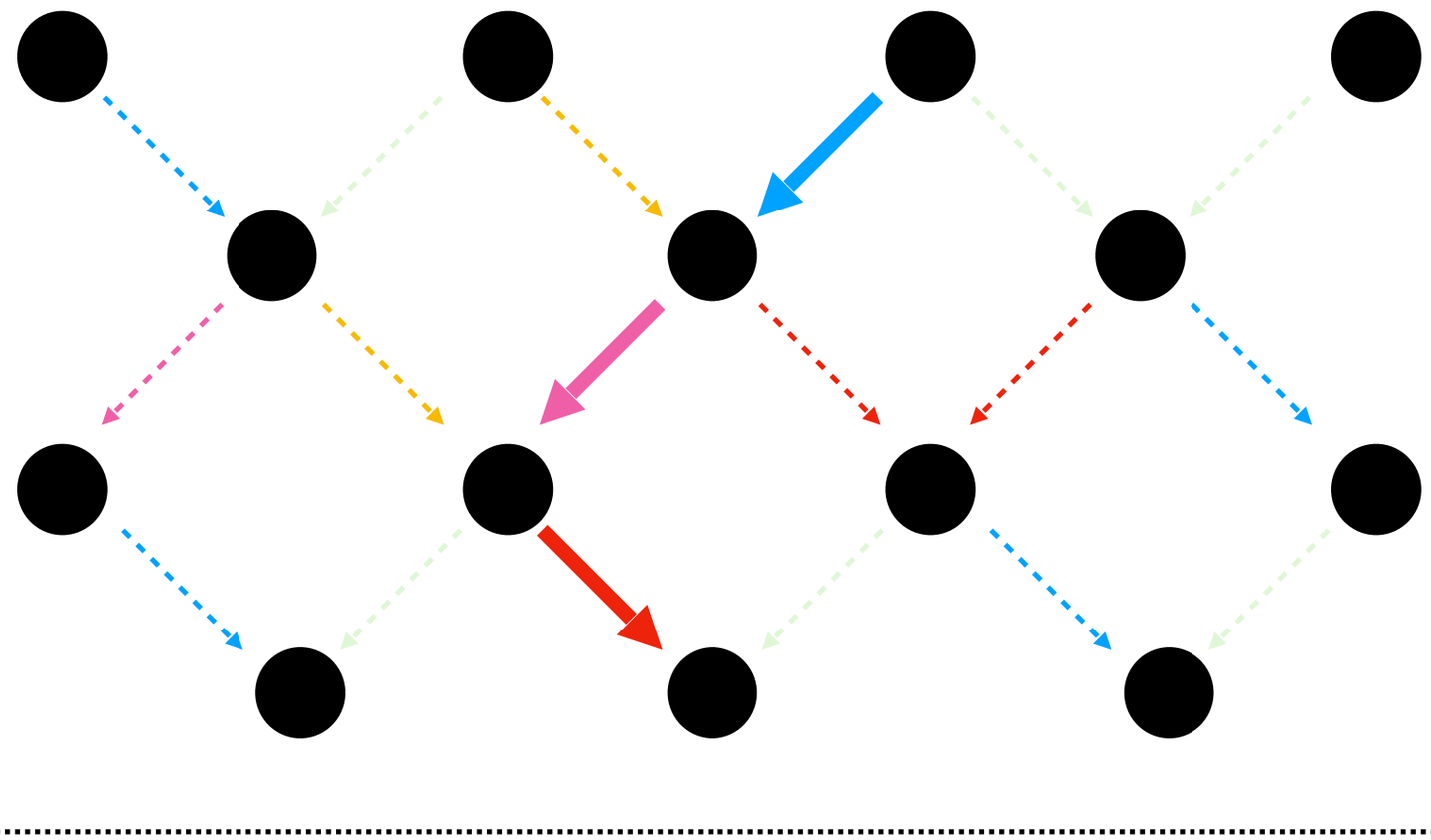
Job

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

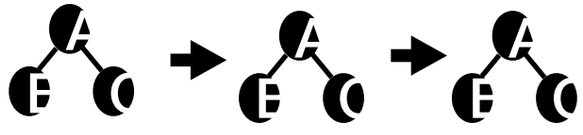
Rules

200+

Optimizer



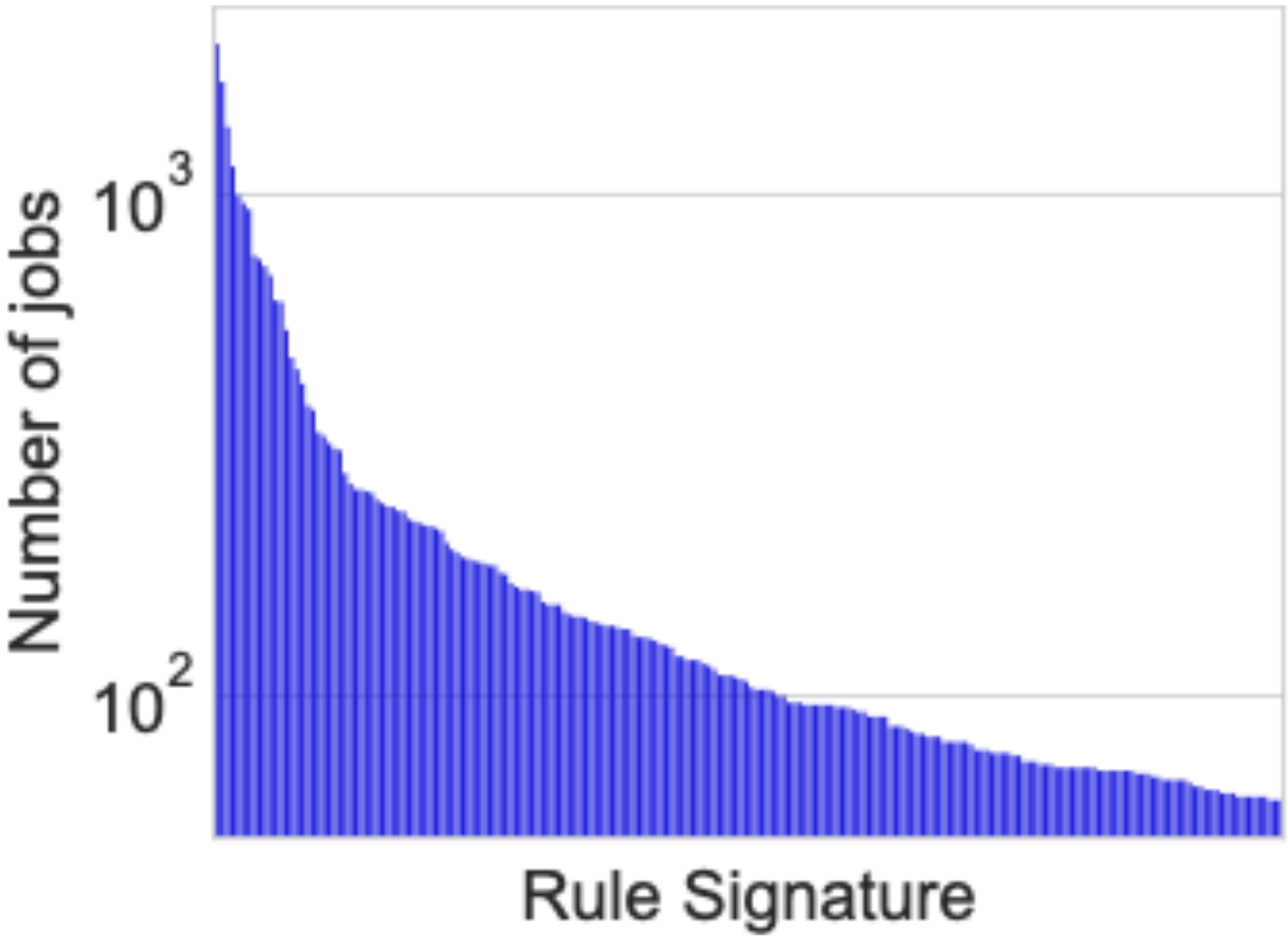
Plan



Block bad path.
Steer optimizer to better path

Rule Signature

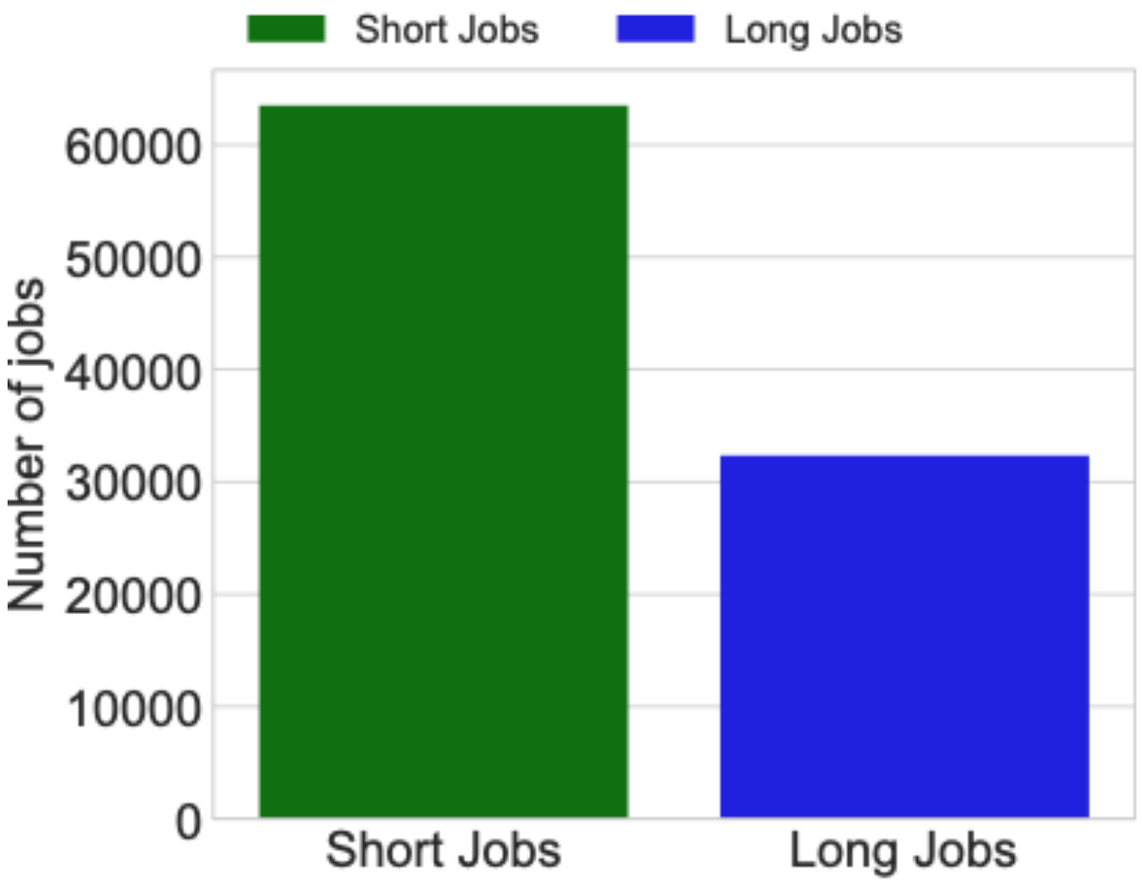
Exploration **Rule Signature**



Technically, 2^256 potential rule signatures. But it has a lot of structure!

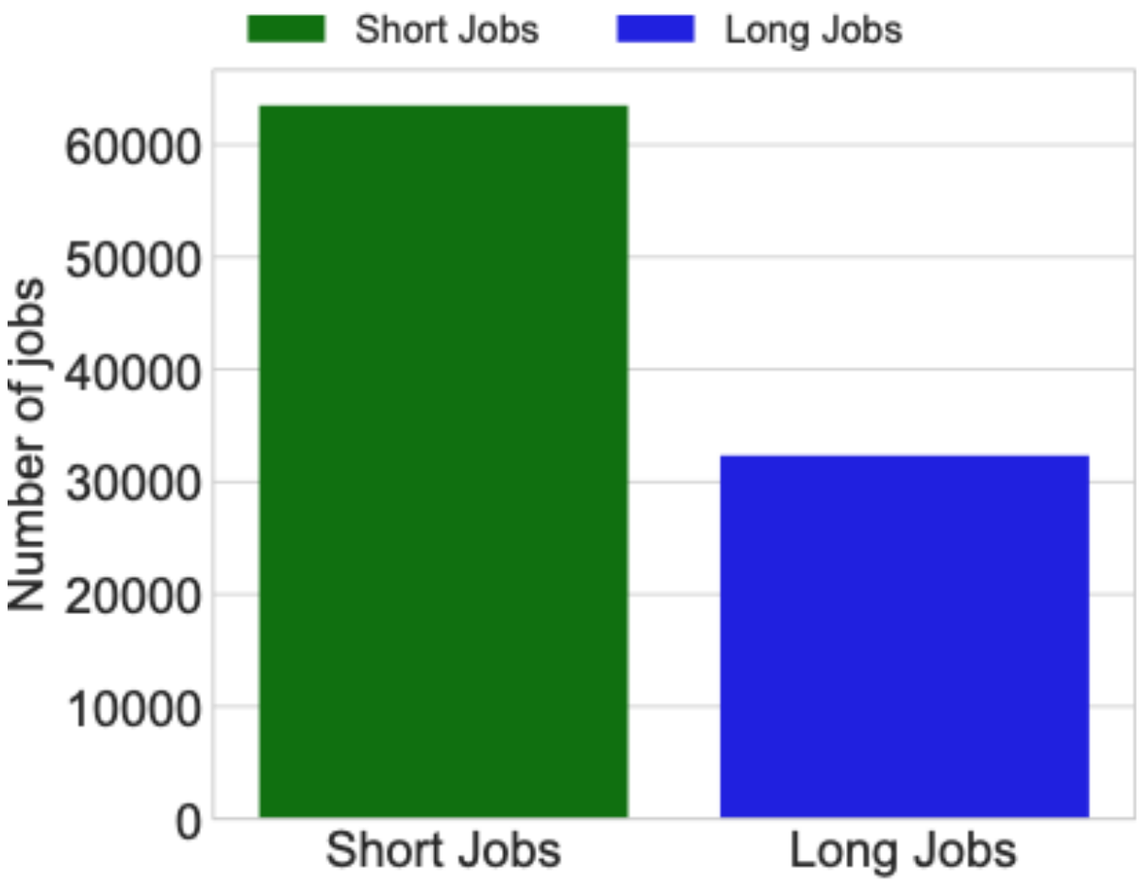
Exploration **Which jobs to try I**

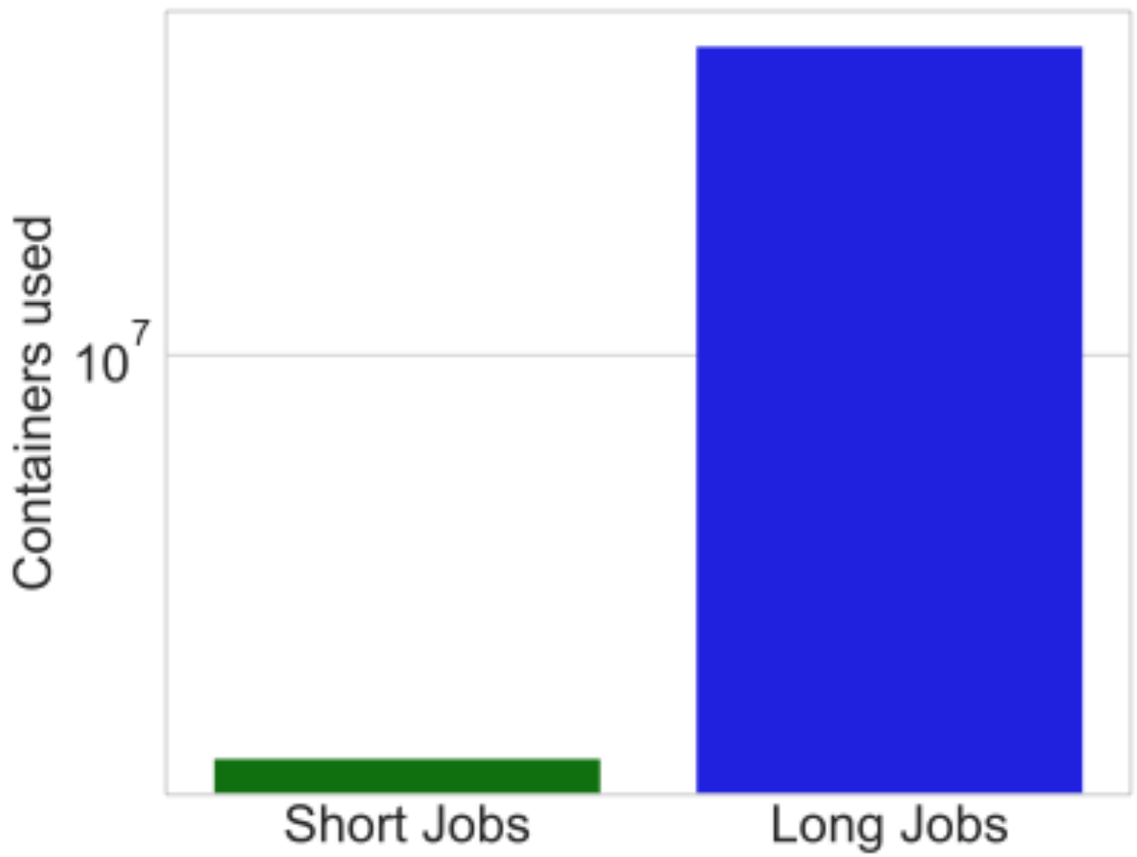
Long, i.e. > 5 minutes



Exploration **Which jobs to try I**

Long, i.e. > 5 minutes

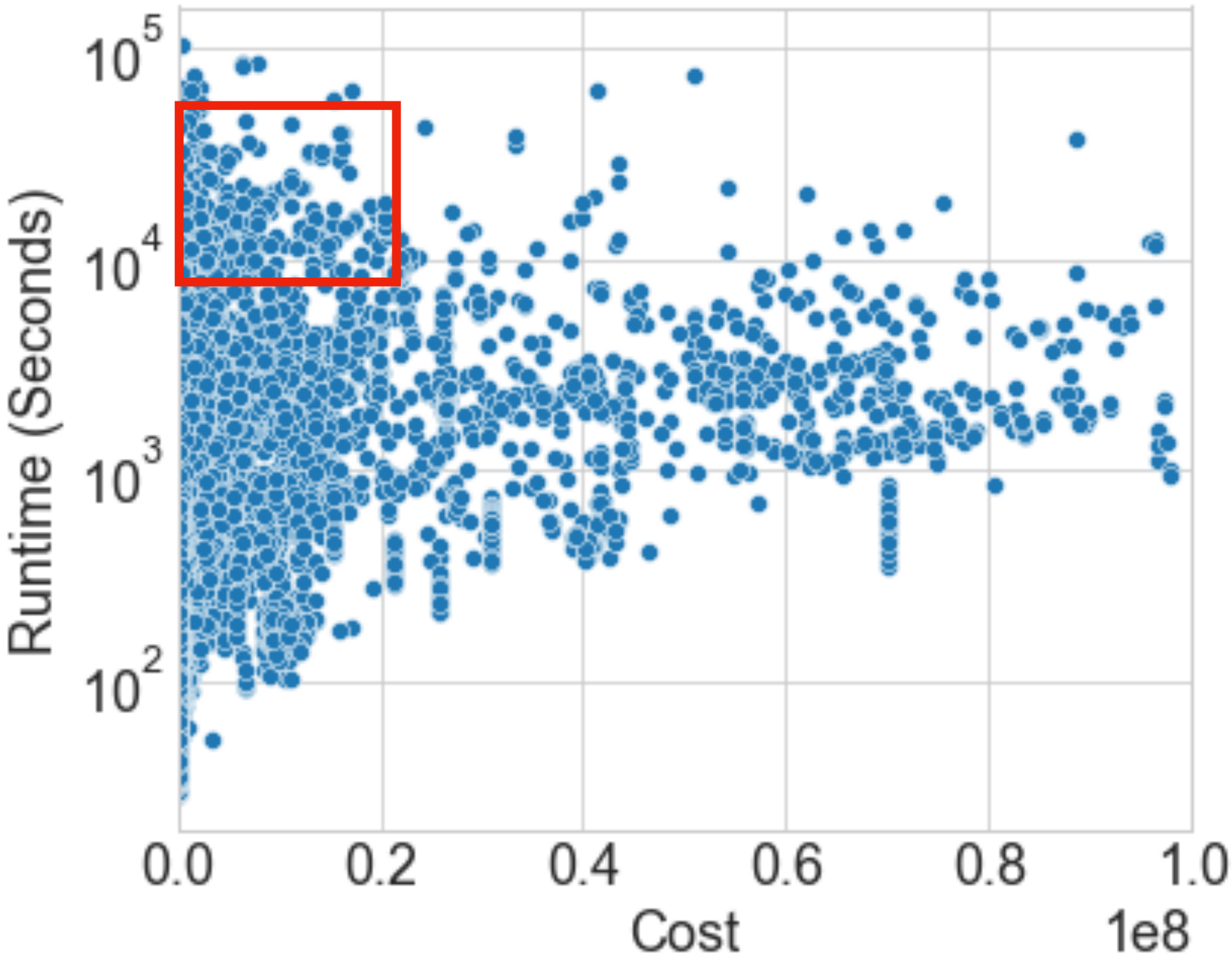




Long jobs use almost all resources!

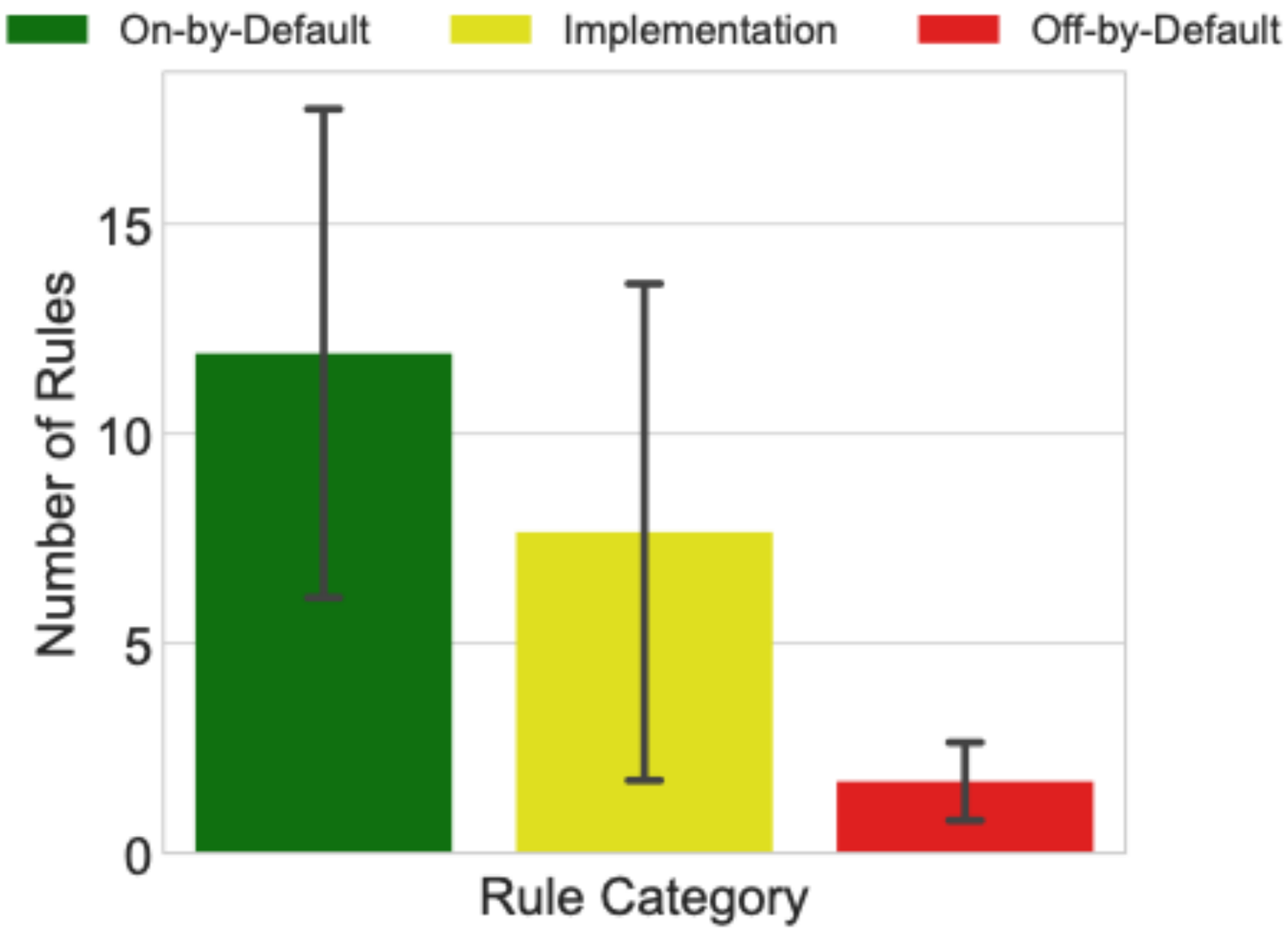
Exploration **Which jobs to try II**

Low cost, but high runtime --- suggests cost model is wrong



Exploration **Which rules to try I**

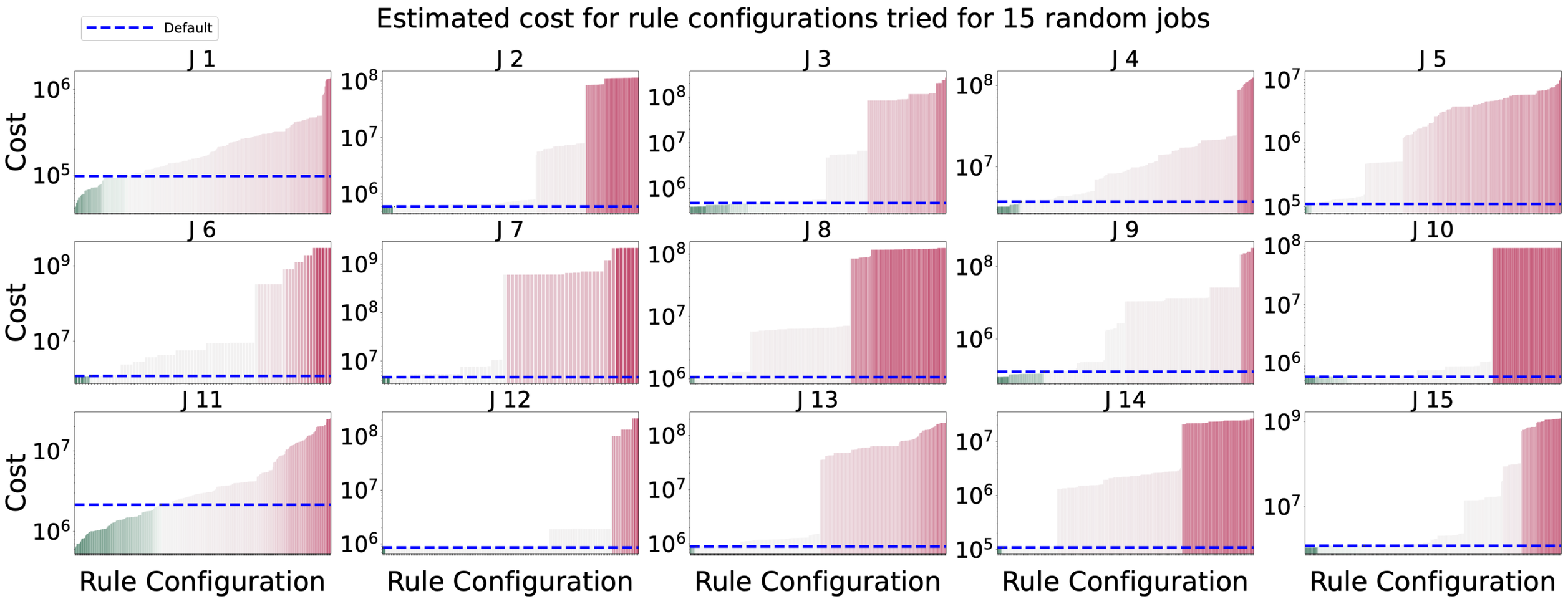
255 total rules. Use rule signatures to estimate interesting set of rules for a job



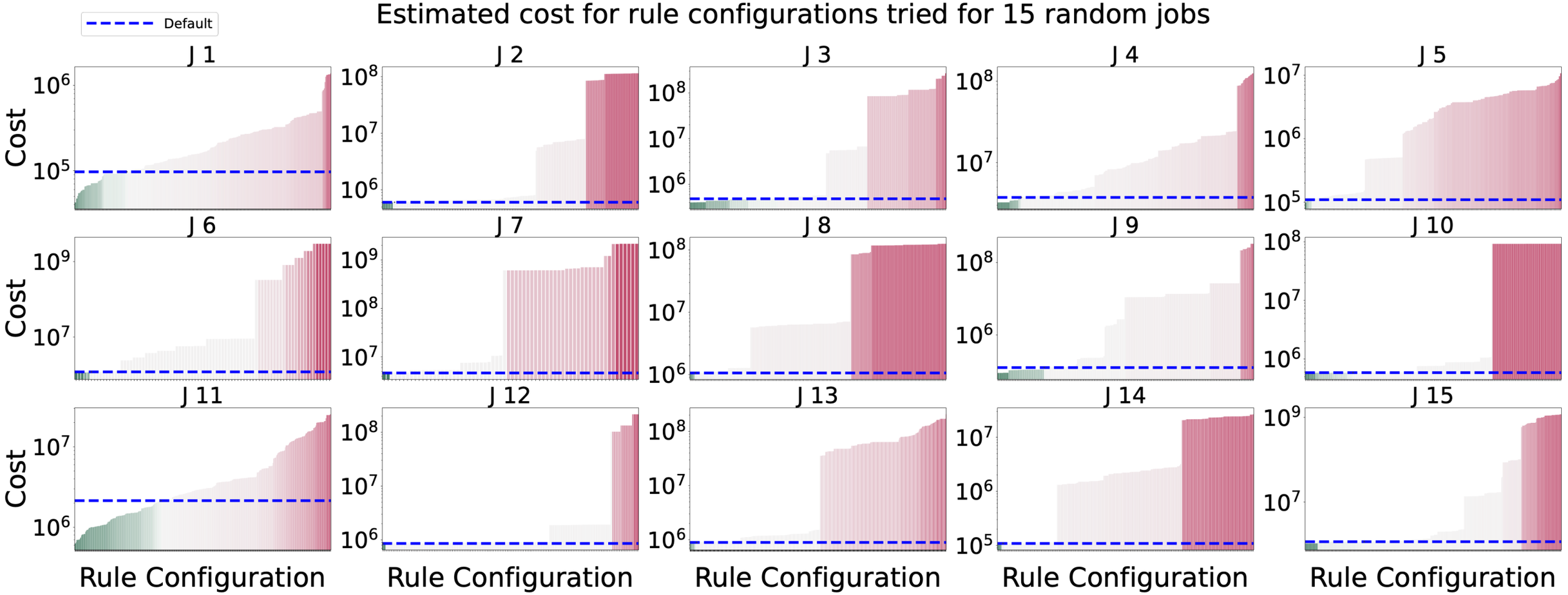
Heuristic: *Try to divide rules into potentially independent categories*

Exploration **Which rules to try II**

Heuristic: *Randomly enable / disable the interesting set of rules for the job to generate up to 1000 tries, and compile them*



Exploration **Which rules to try II**



Plans with different rules, apples to oranges comparison!

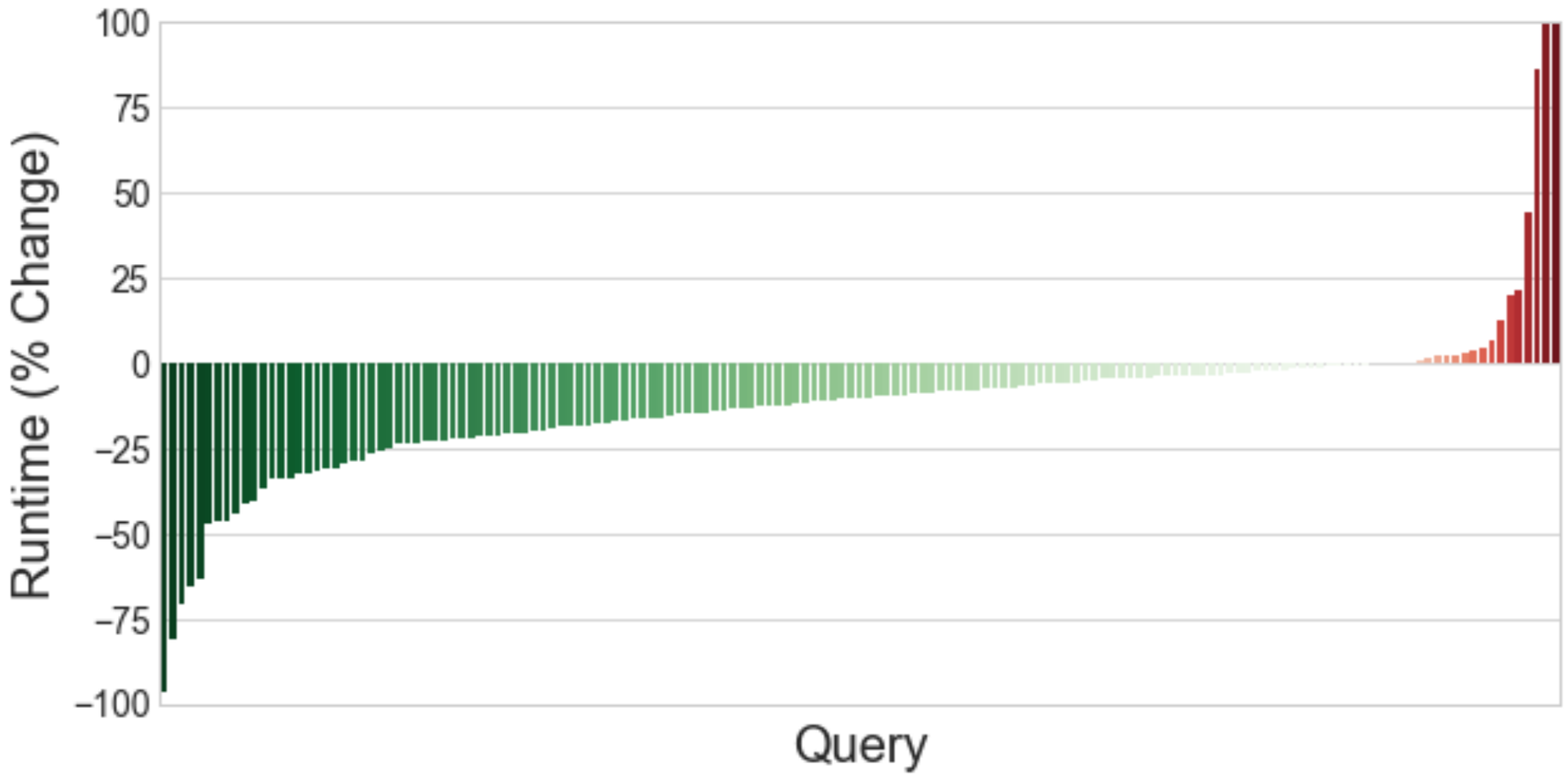
Cheaper plans by disabling rules because costs depend on rules (e.g., rules for parallelism, or cardinalities)

Exploration **Which rules to try II**

Heuristic: *Try executing 10 such "cheaper" plans to see if they are actually any good*

Contribution: **Disabling rules useful in SCOPE**

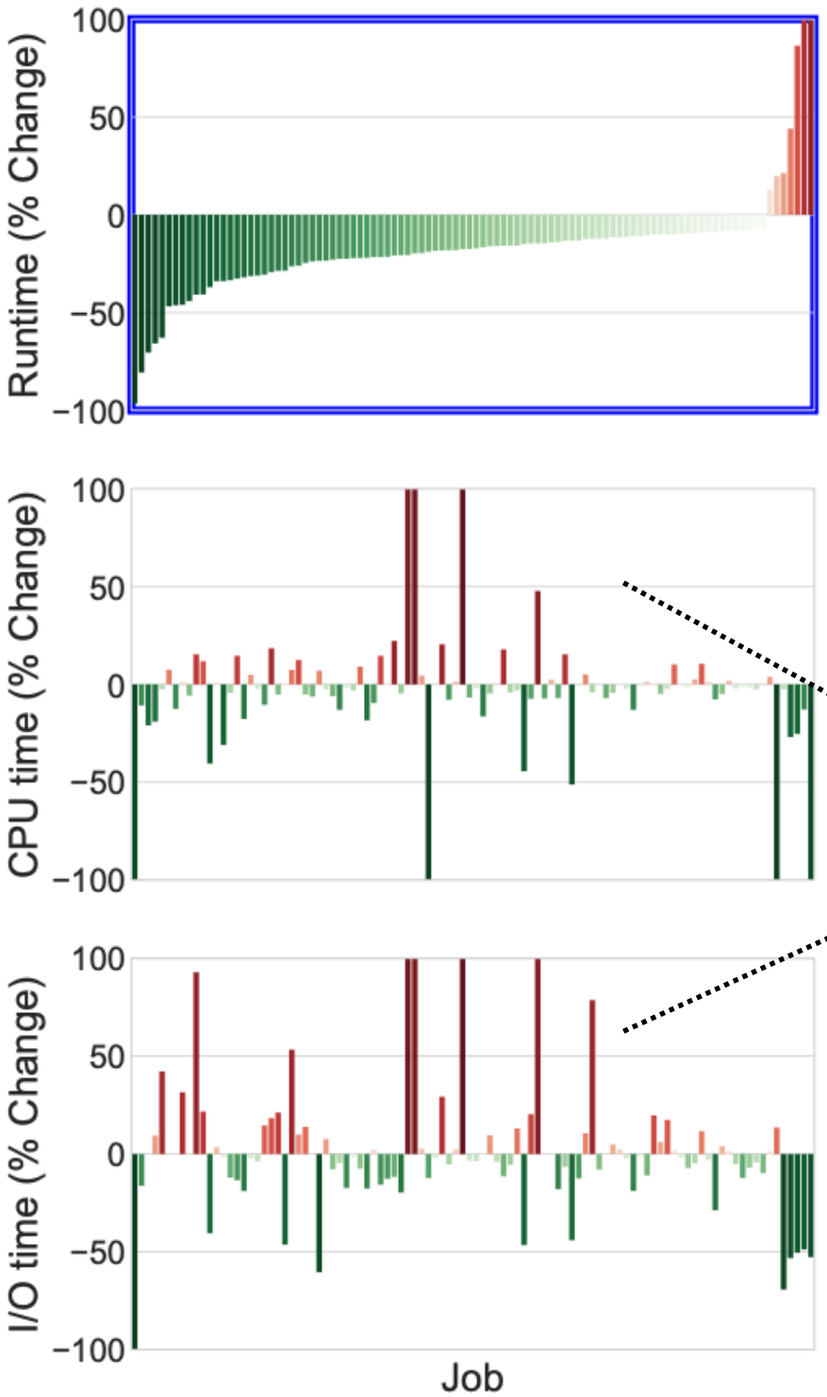
Each bar shows the query run time percentage improvement (green) or regression (red) from the best of the 10 configurations we execute



Requires a lot of experimentation to find useful rule configurations!

Contribution: **Different potential metrics**

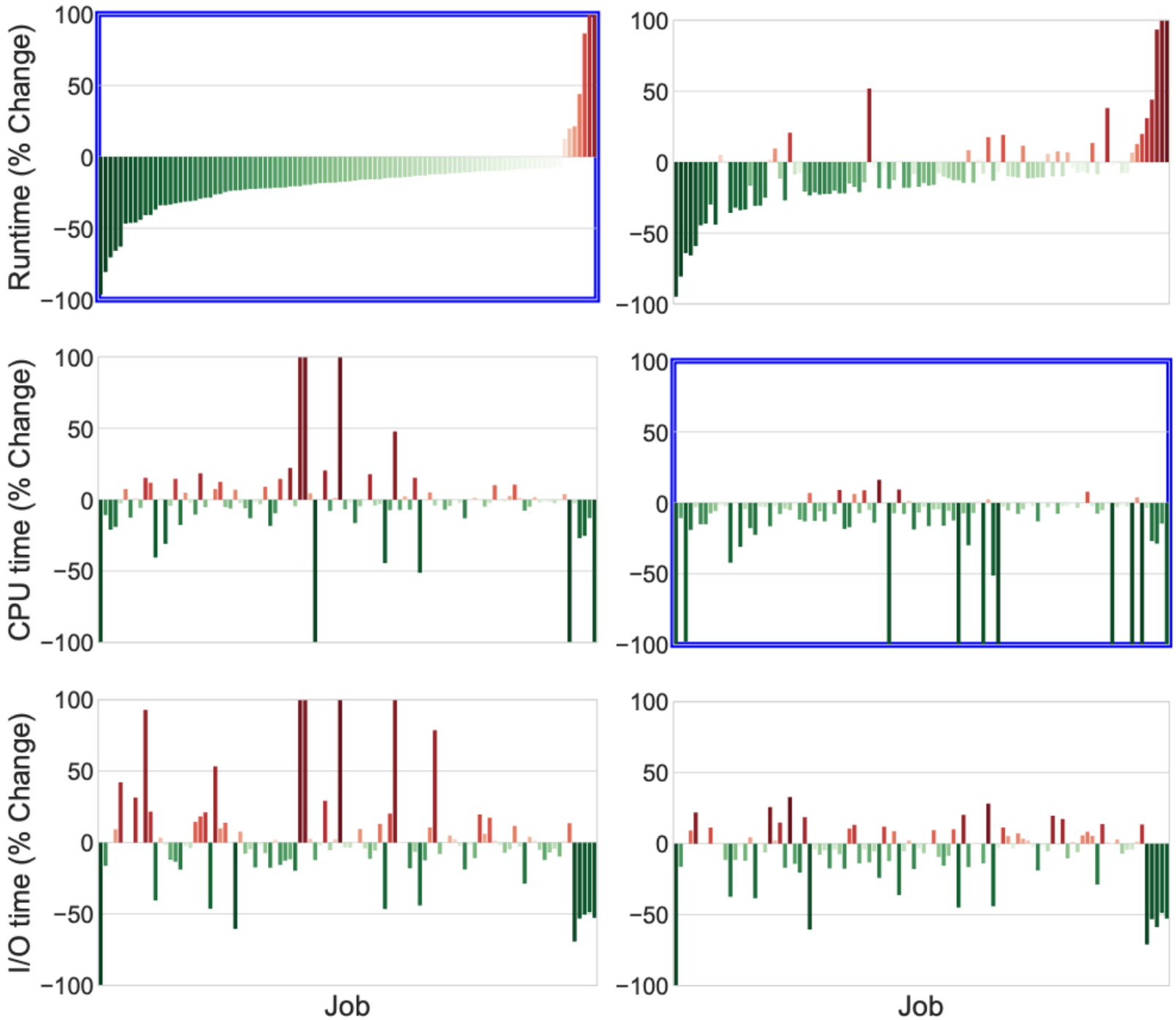
Choose best of the 10 rule configurations for **runtime**



Other metrics sometimes see regressions!

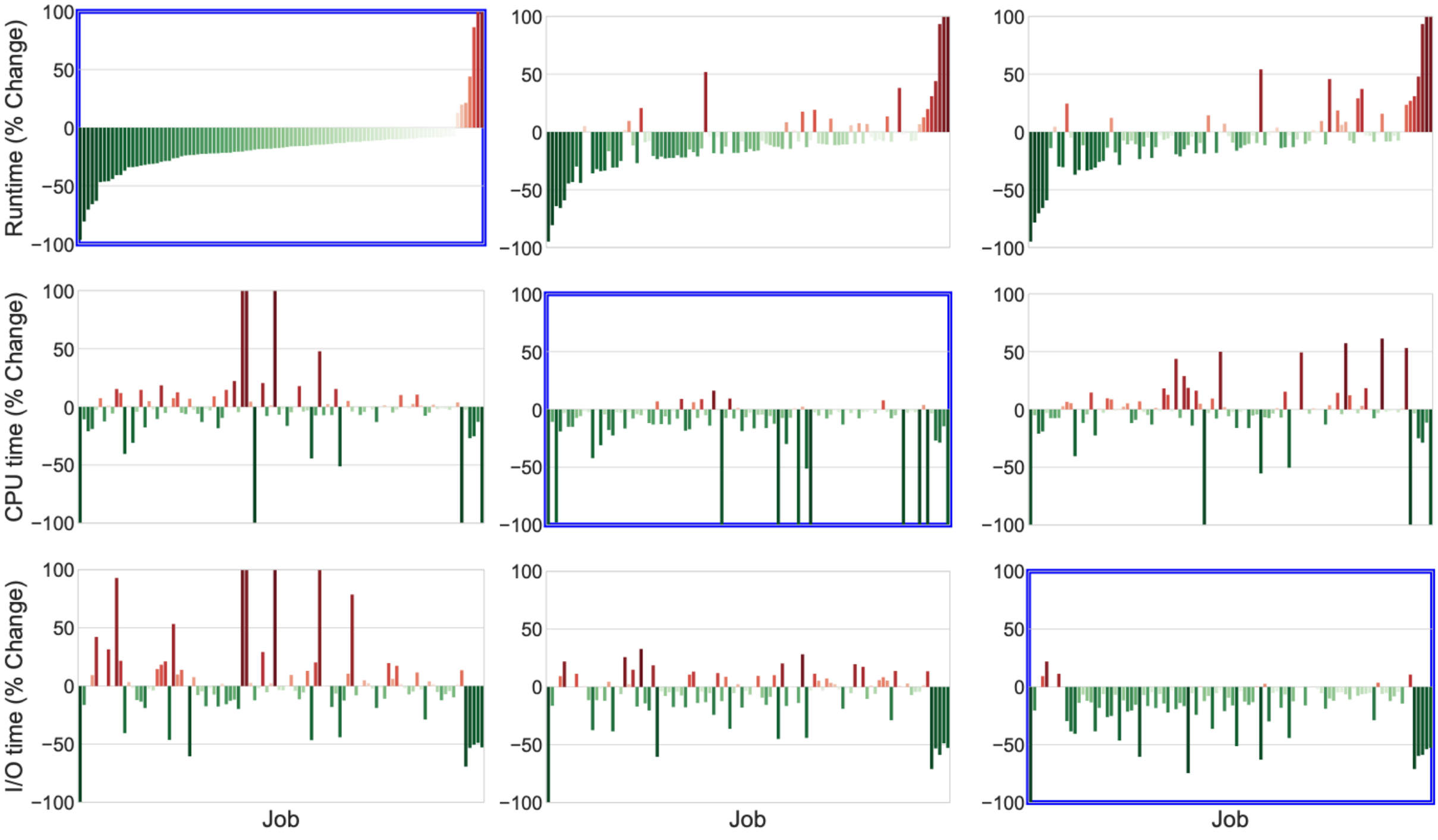
Contribution: **Different potential metrics**

Choose best of the 10 rule configurations for **CPU time**



Contribution: **Different potential metrics**

Choose best of the 10 rule configurations for **I/O time**



Learning: **Applying rules at compile time**

So far: try N potential rule configurations; see if any one gives improvement

Heuristic: if we see **similar job** in future,
apply rule configuration that improved runtime

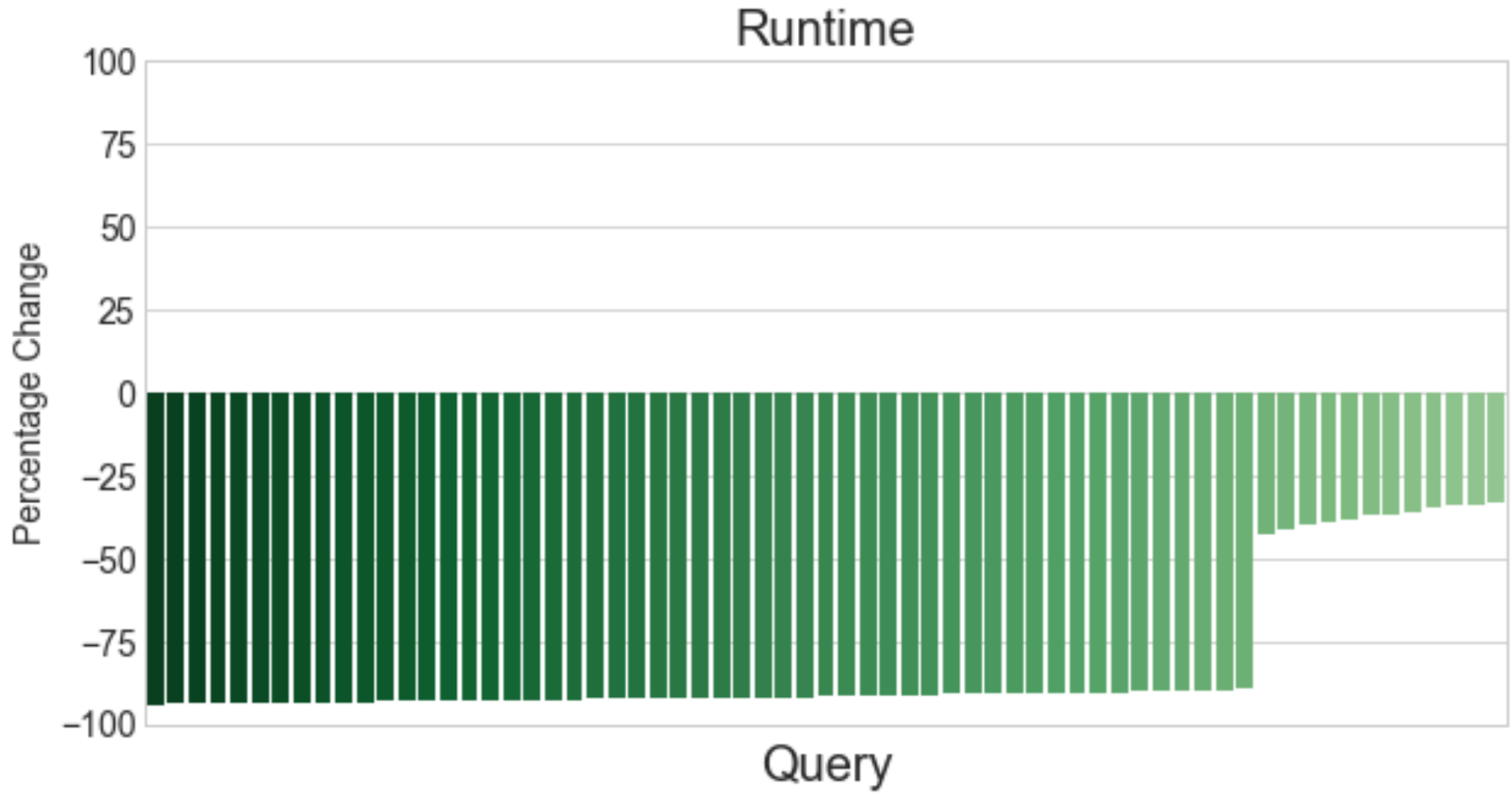
Learning: **Applying rules at compile time**

So far: try N potential rule configurations; see if any one gives improvement

Heuristic: if we see ~~**similar job**~~ **jobs with same rule signature** in future, apply rule configuration that improved runtime

Learning: **Just apply the best configuration?**

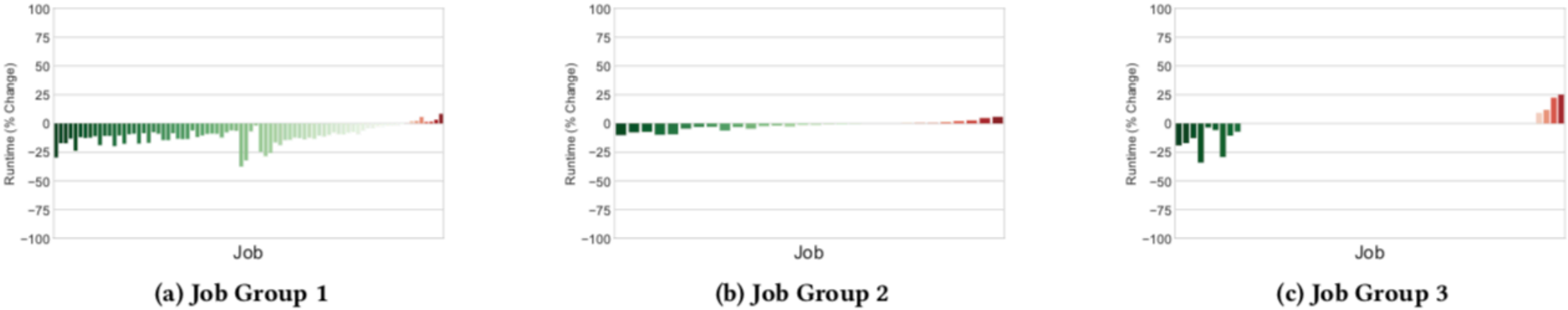
Case 1: E.g., of jobs with same rule signature, over 2 weeks, with consistently almost 90% improvements by simply applying a previously seen rule configuration



Rarer. Probably because very repetitive workload!

Learning: **Choosing from candidate rule configurations**

Case 2: Find set of interesting configurations, including the default, and use a model to choose one at compile time



(a) Job Group 1 (b) Job Group 2 (c) Job Group 3

All jobs in the group map to the same rule signature

Challenges

Creating job groups based on rule signatures not scalable.
Can only find the repetitive job patterns (e.g., templated queries)

Challenges

Creating job groups based on rule signatures not scalable.
Can only find the repetitive job patterns (e.g., templated queries)

Hard to interpret the new rule configurations
(we found them by randomly trying interesting configurations)

Challenges

Creating job groups based on rule signatures not scalable.
Can only find the repetitive job patterns (e.g., templated queries)

Hard to interpret the new rule configurations
(we found them by randomly trying interesting configurations)

Disabling rule for only sub-components of a job
(job consists of many SQL queries
chained together)

Thank You!