Machine Learning for Query Optimization

Ryan Marcus
University of Pennsylvania

Slides: `https://rm.cab/bu23`

Learned Systems

- Claim: hardware & app diversity growing faster than we can design systems
- Approach: build instance-optimized, learned systems that automatically…
  - *Invent* new approaches to solving the user's problem
  - *Adapt* to the user's workload & hardware
  - Understands the user's *intention*

Gottschlich et al. "The three pillars of machine programming." MAPL@SIGPLAN. 2018.

This Talk

- Why learn a query optimizer?
- Brief description of **Neo**, our first attempt.
- Key ideas in **Bao**, our second attempt.

Query Optimization

- Essentially: translate complex requests for information into fast programs.
- Ex: find all movies with Scarlett Johansson produced by Sony

ACTOR

a_id	name	YOB
1	Scarl	84
2	BradP	63
3	JonTr	54
…		

FILM

f_id	name	RAT
1	Her	86
2	Aveng	81
3	PulpF	93
…		

COMPANY

c_id	name
1	Sony
2	Fox
3	MGM
…	

APPEARS_IN

a_id	f_id
1	1
1	2
3	3
…	

PRODUCED

c_id	f_id
1	1
2	2
3	2
…	

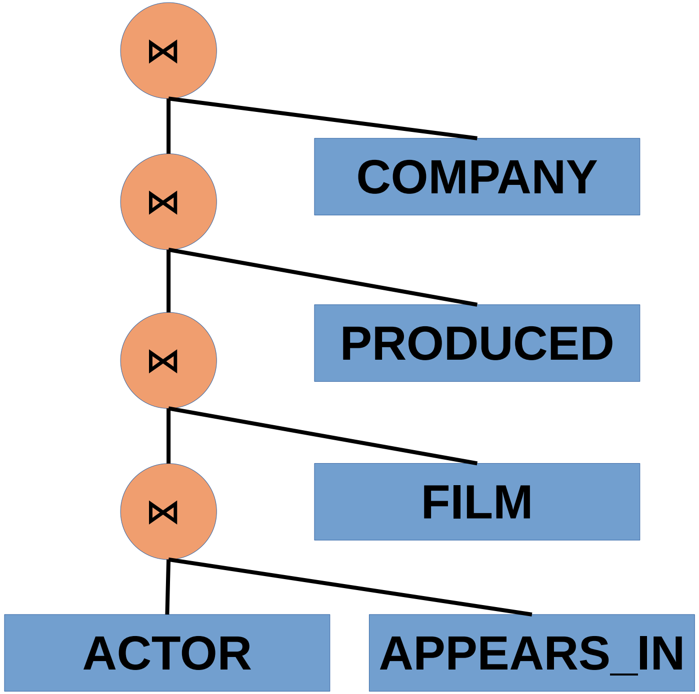
Query Optimization

- Find all movies with Scarlett Johansson produced by Sony.
- *Logically,* what we want is to filter:

$$\text{ACTOR} \bowtie \text{APPEARS\_IN} \bowtie \text{FILM} \bowtie \text{PRODUCED} \bowtie \text{COMPANY}$$

- Physically, I have a lot of options...

Query Optimization



ACTOR

a_id	name	YOB
1	Scarl	84
2	BradP	63
3	JonTr	54
…		

FILM

f_id	name	RAT
1	Her	86
2	Aveng	81
3	PulpF	93
…		

COMPANY

c_id	name
1	Sony
2	Fox
3	MGM
…	

APPEARS_IN

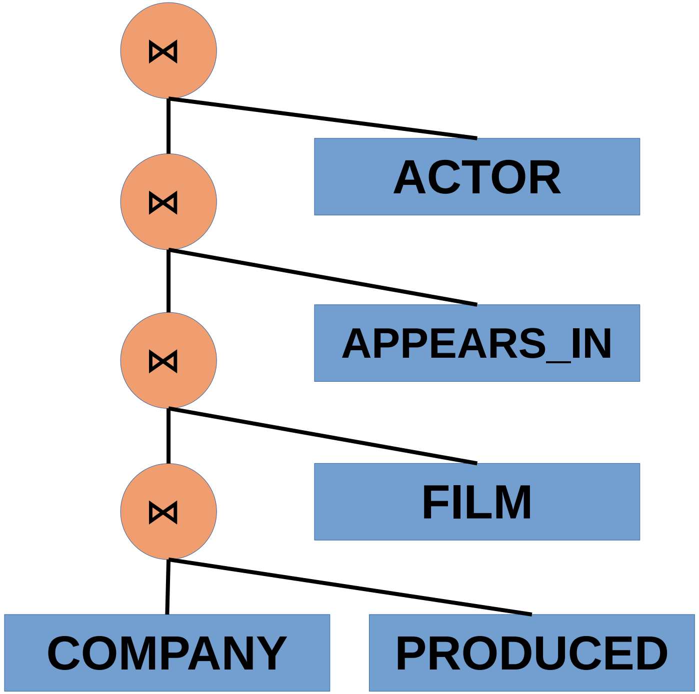
a_id	f_id
1	1
1	2
3	3
…	

PRODUCED

c_id	f_id
1	1
2	2
3	2
…	

Find all movies with SJ. Then, filter those by movies produced by Sony.

Query Optimization



ACTOR

a_id	name	YOB
1	Scarl	84
2	BradP	63
3	JonTr	54
…		

FILM

f_id	name	RAT
1	Her	86
2	Aveng	81
3	PulpF	93
…		

COMPANY

c_id	name
1	Sony
2	Fox
3	MGM
…	

APPEARS_IN

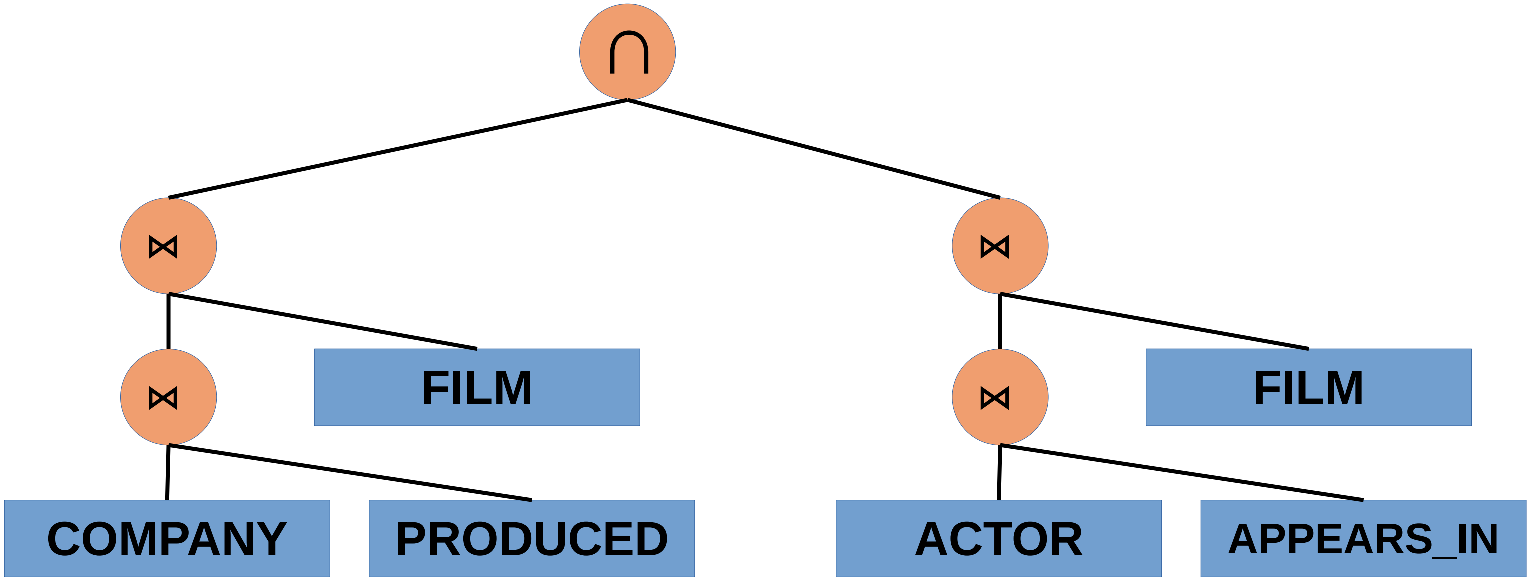
a_id	f_id
1	1
1	2
3	3
…	

PRODUCED

c_id	f_id
1	1
2	2
3	2
…	

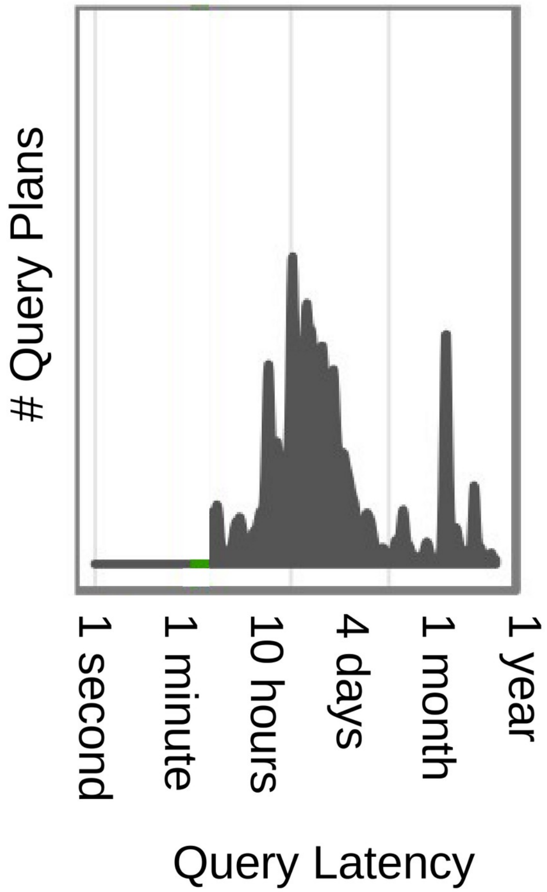
Find all Sony movies. Then, filter by those movies with SJ.

Query Optimization



Find all Sony movies, find all SJ movies, take the intersection.

Query Optimization

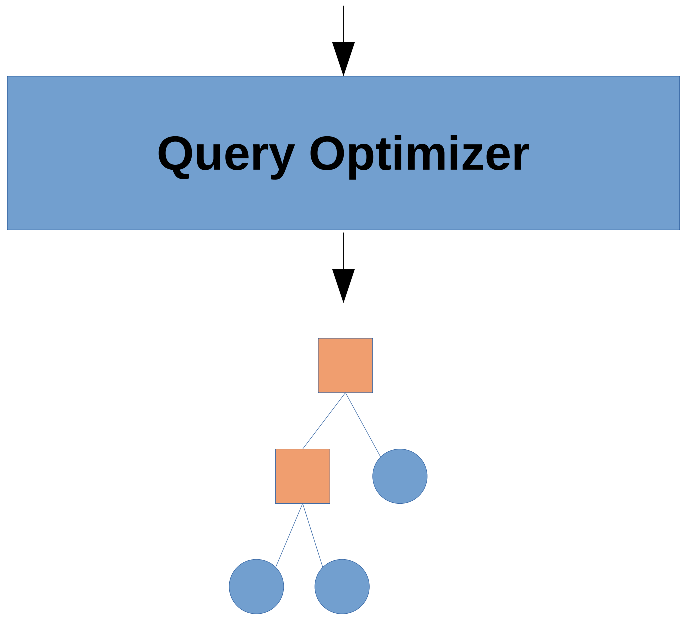


- # plans follows Catalan numbers
- At n = 19, more than 2^32 plans

Classic Query Optimizers

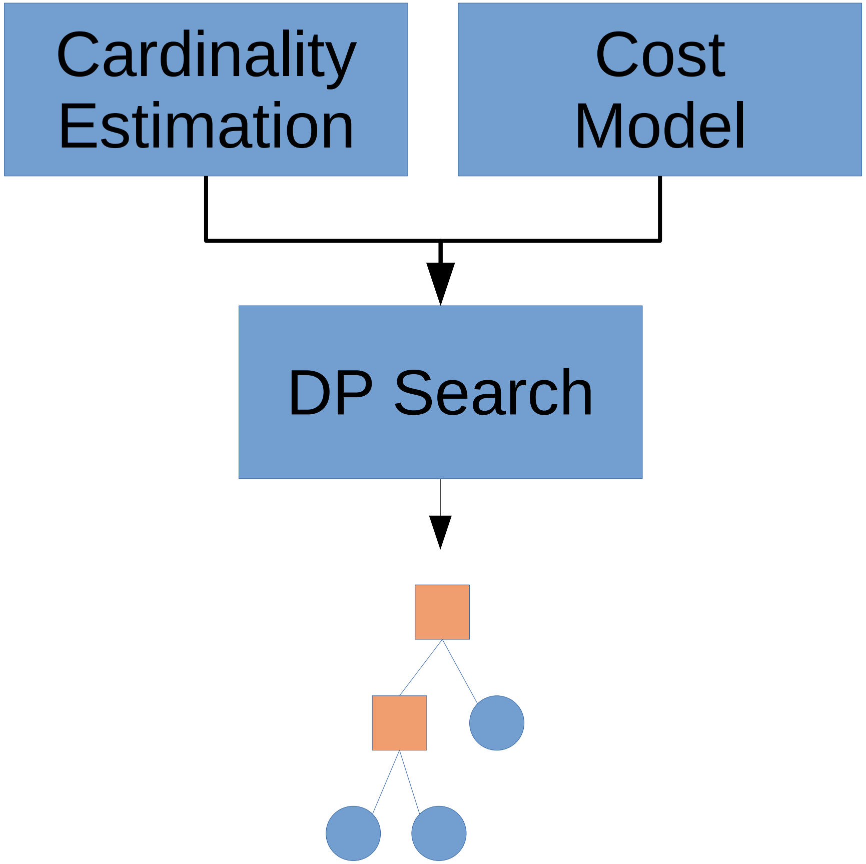
- Transform SQL into a query plan
- HUGE effort!
  - 42K LOC in PG
  - 1M+ SQL Server
  - 45-55 FTEs, Oracle (~ $5mil/year)
- Requires *per DB* tuning
  - PG: 15% bump
  - Oracle: 22% bump
  - SQL Server: 18% bump

```
SELECT *
FROM t1, t2 WHERE…
```

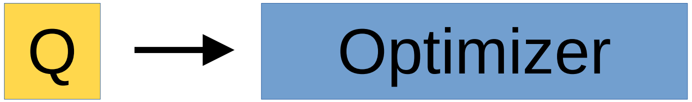


Classic Query Optimizers

- Cardinality estimation models
  - Histograms
  - Uniformity
  - MFVs
- Cost models
  - Polynomials
  - Hand tuned
- DP Search
  - NP-Hard

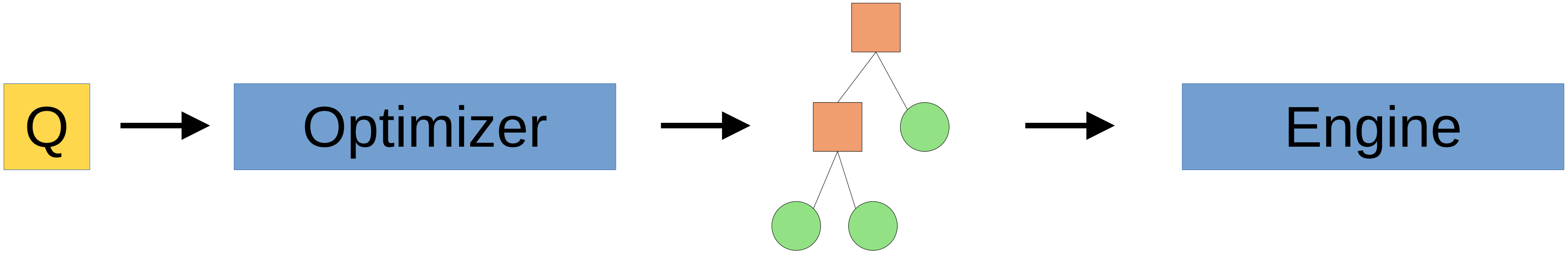


Query Optimization

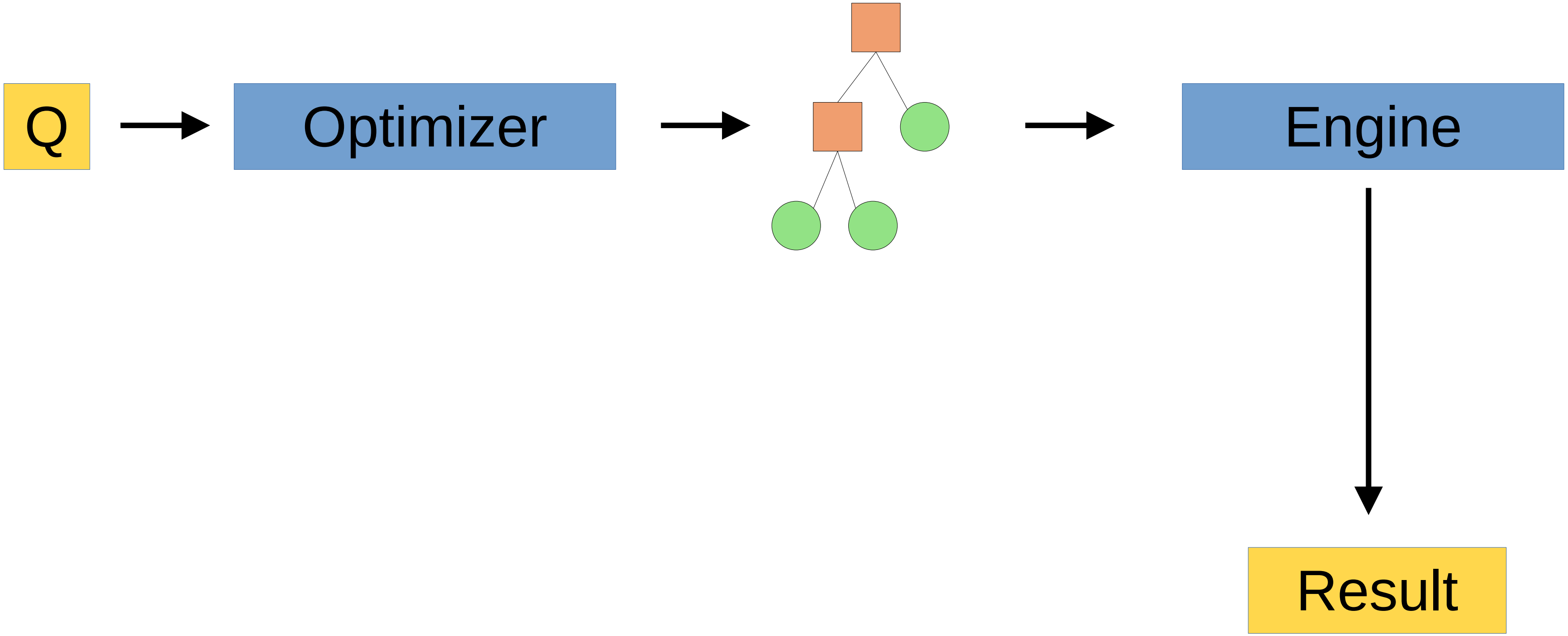


Query Optimization

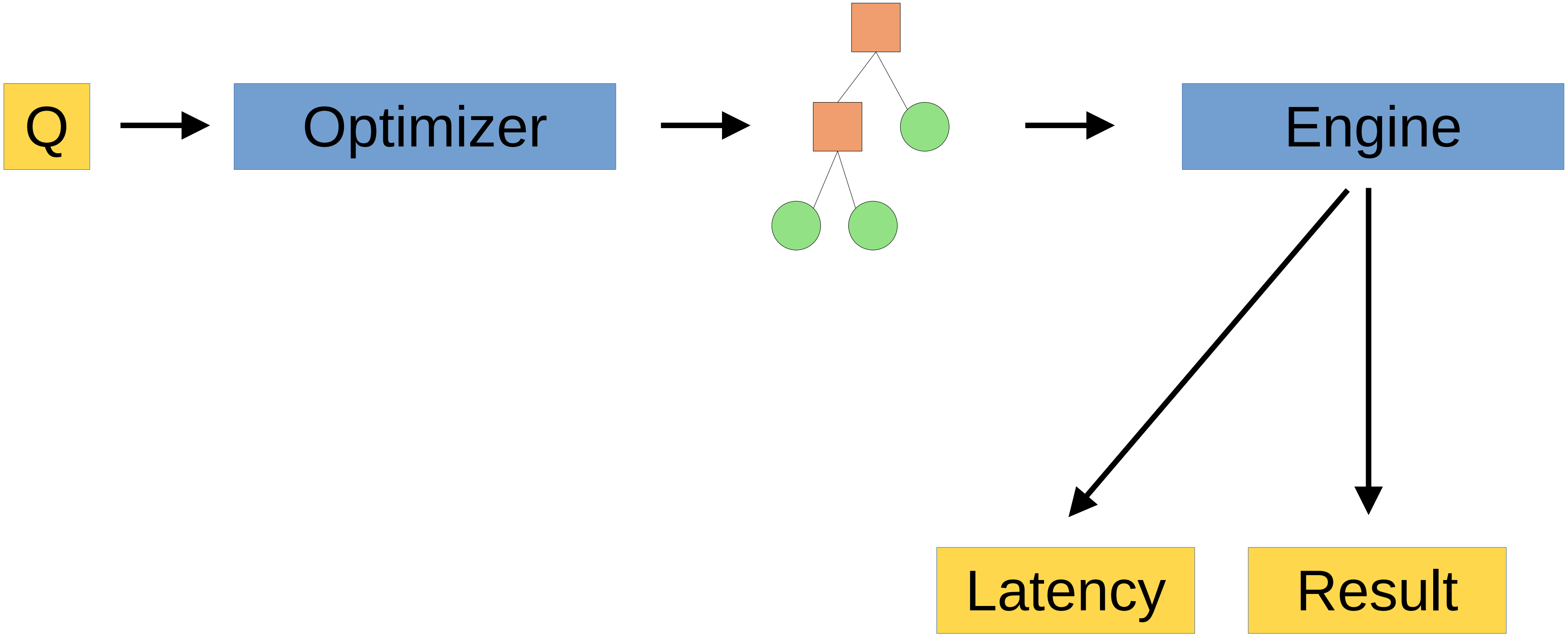
Query Optimization



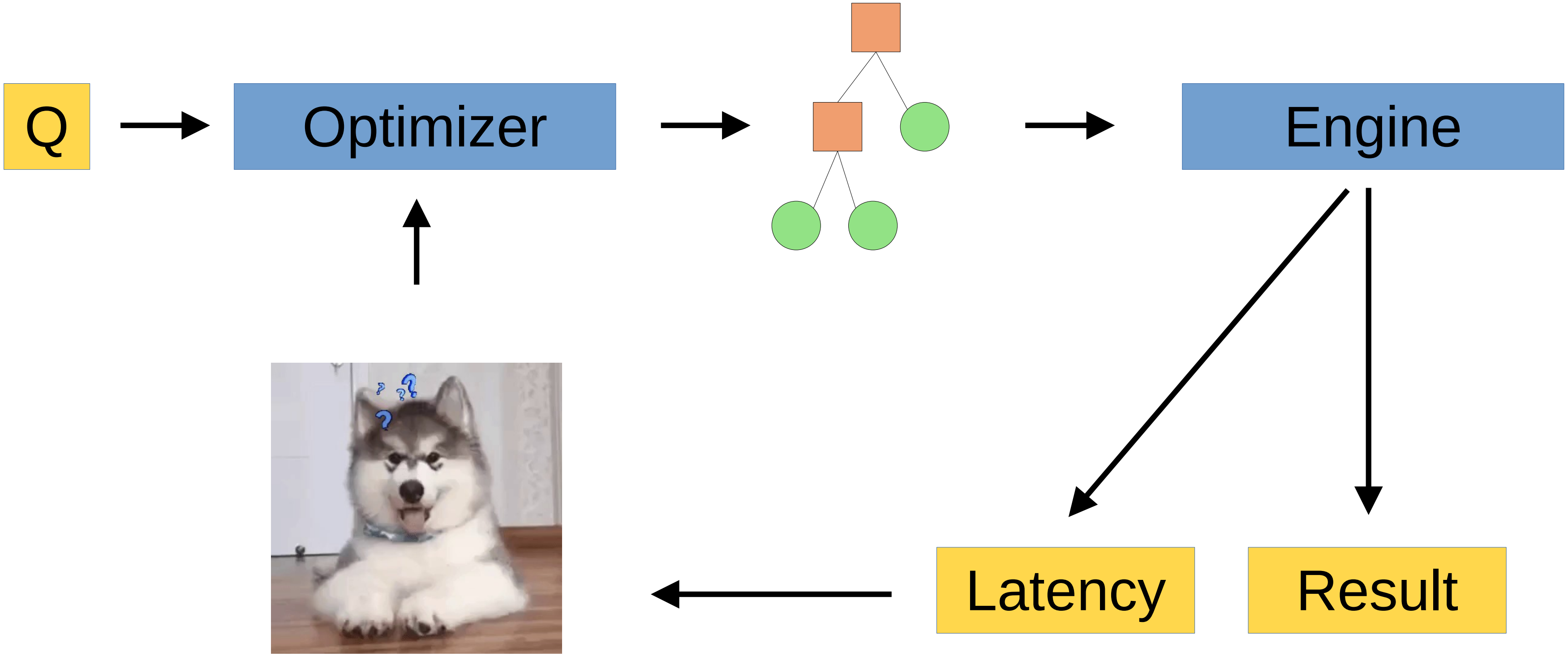
Query Optimization



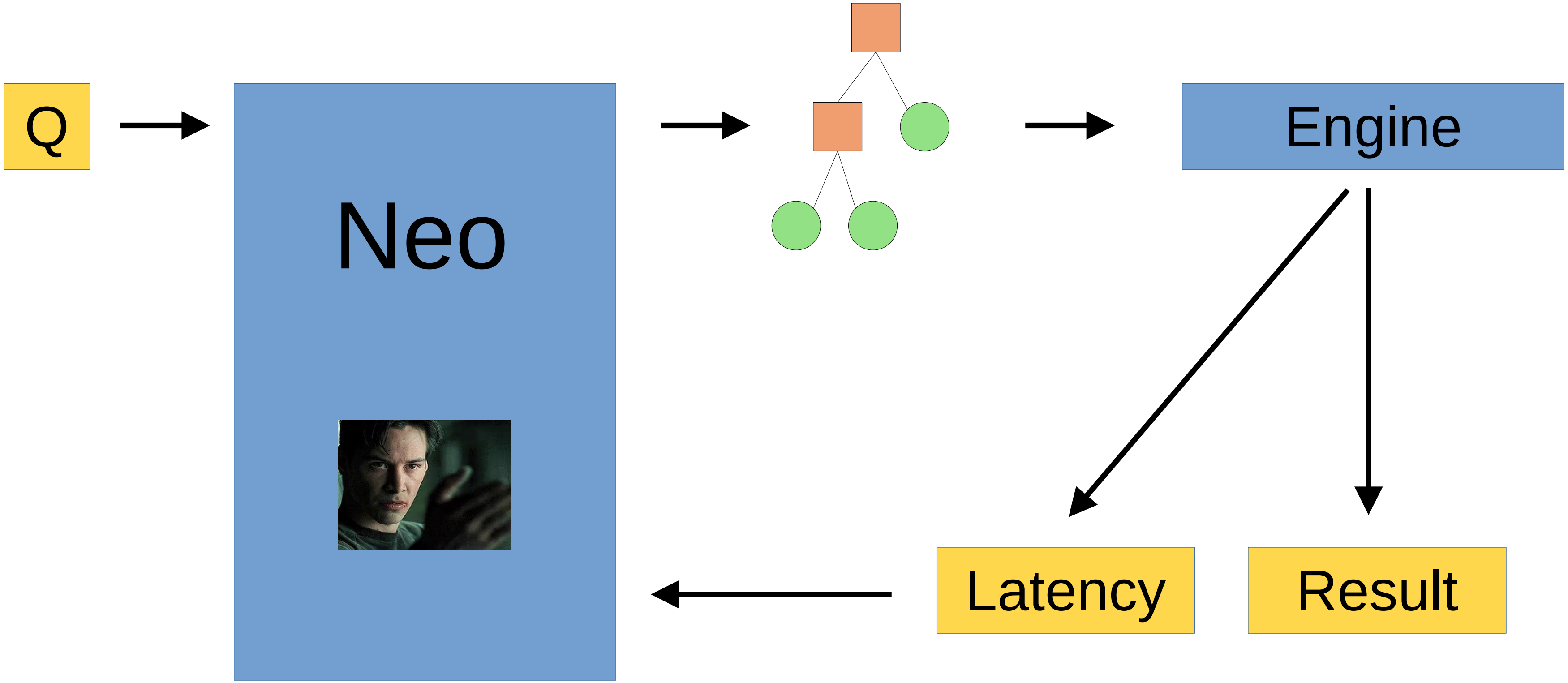
Query Optimization



Query Optimization



Previous Approach: Neo

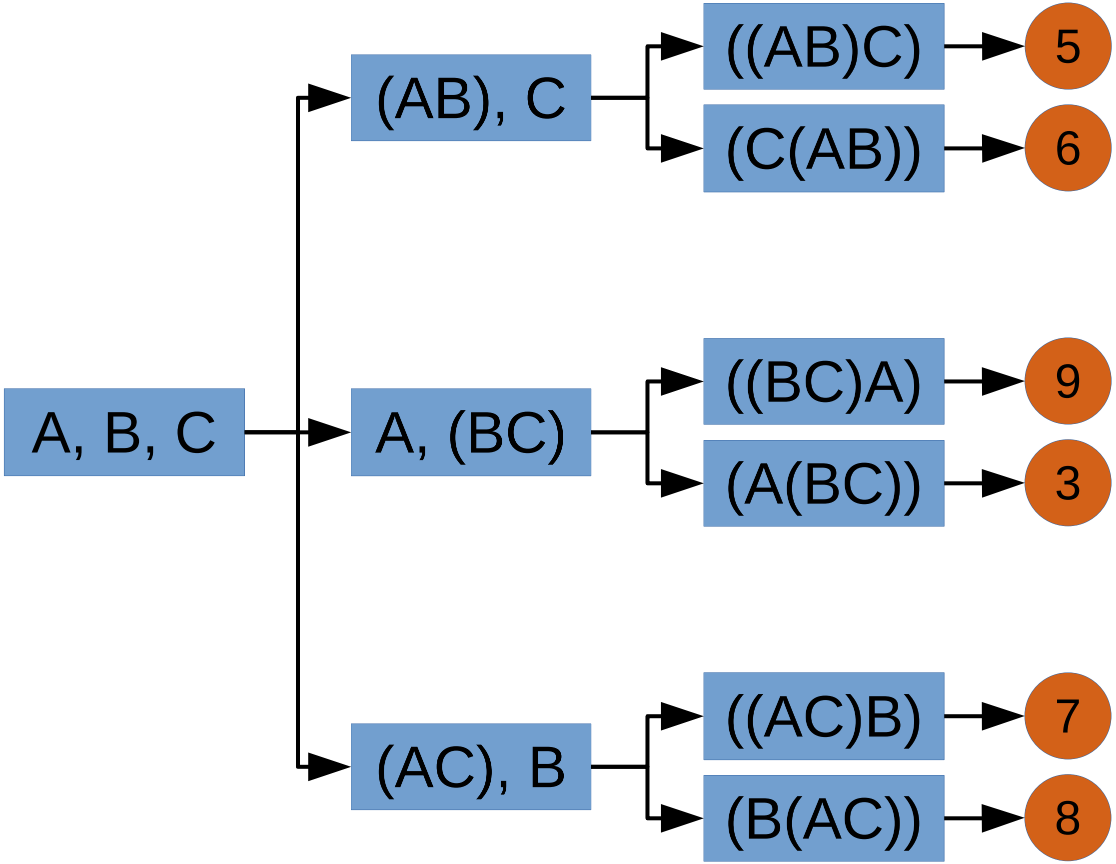


Neo: A Learned Query Optimizer. VLDB '19

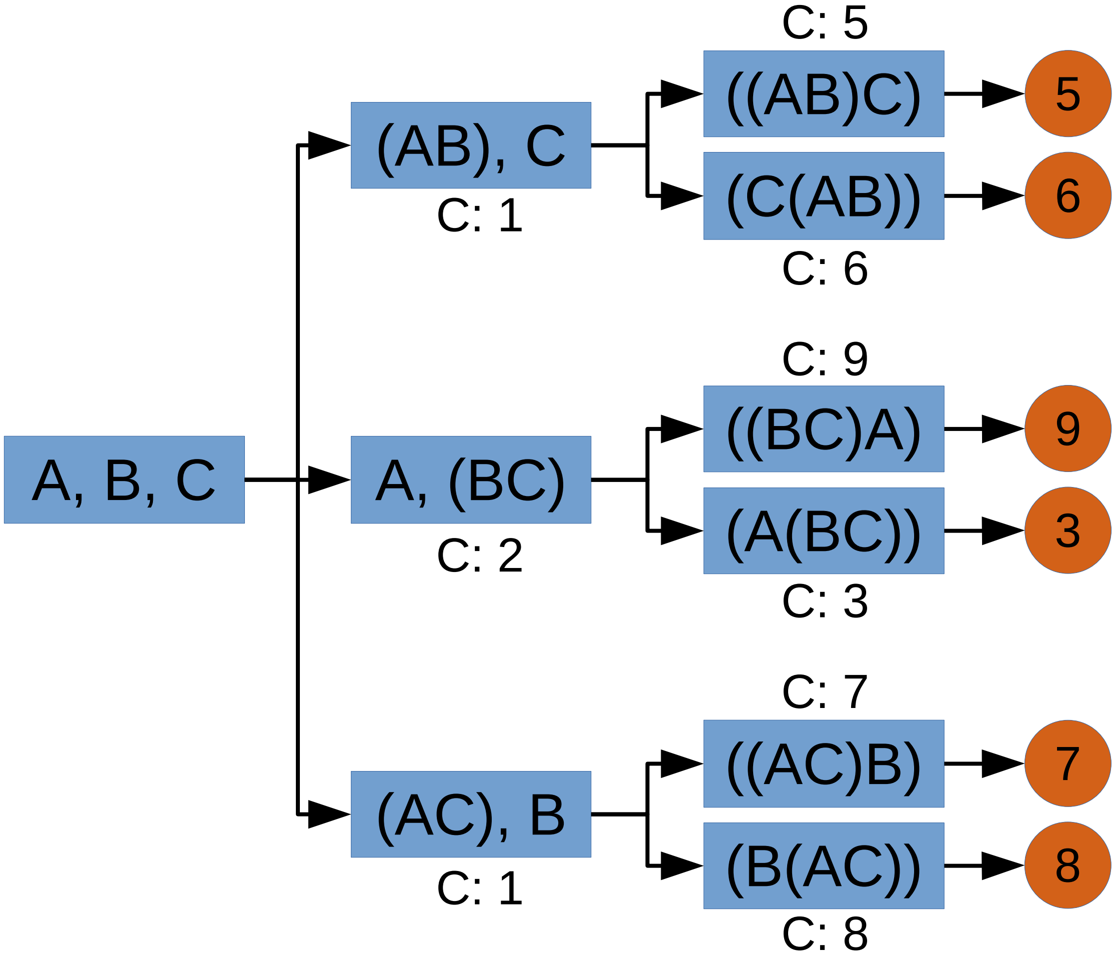
Neo

- Neo is first to show we can have *all learned everything*.
  - No cost models, cardinality estimation or exponential search.
- Deep connection between DRL and standard query optimization techniques
- Beat commercial systems

Deep Reinforcement Learning



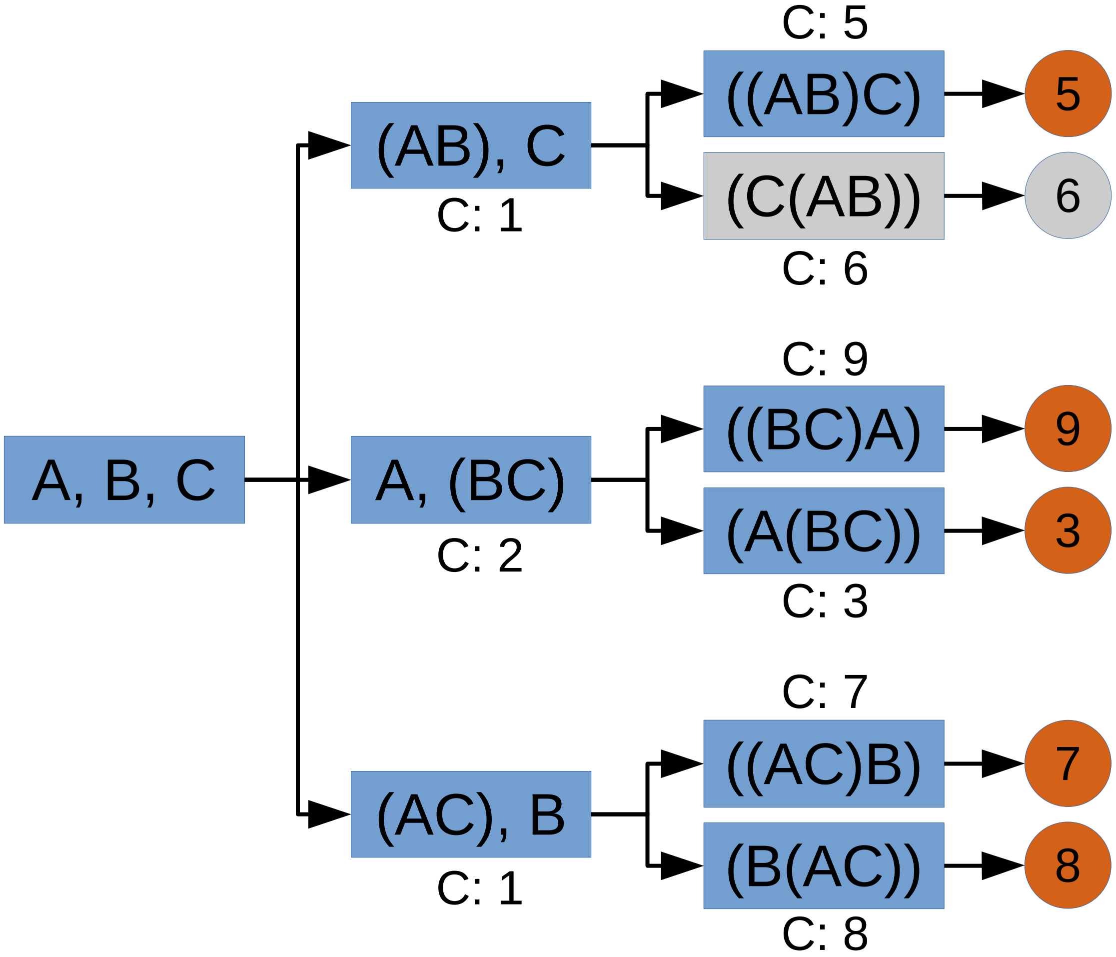
Deep Reinforcement Learning



Traditional Cost Model

A cost function C which estimates the intermediate cost of plan

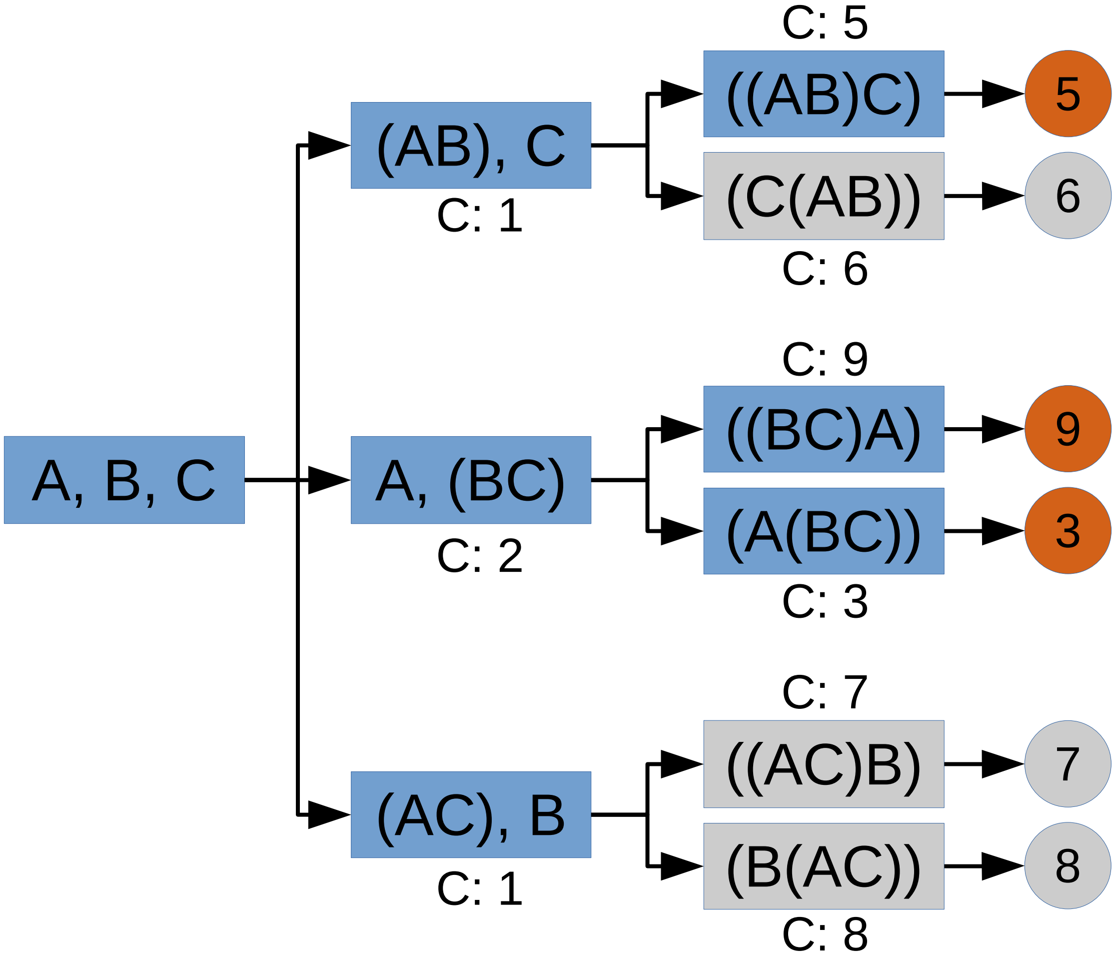
Deep Reinforcement Learning



<u>Traditional Cost Model</u>

A cost function C which estimates the intermediate cost of plan

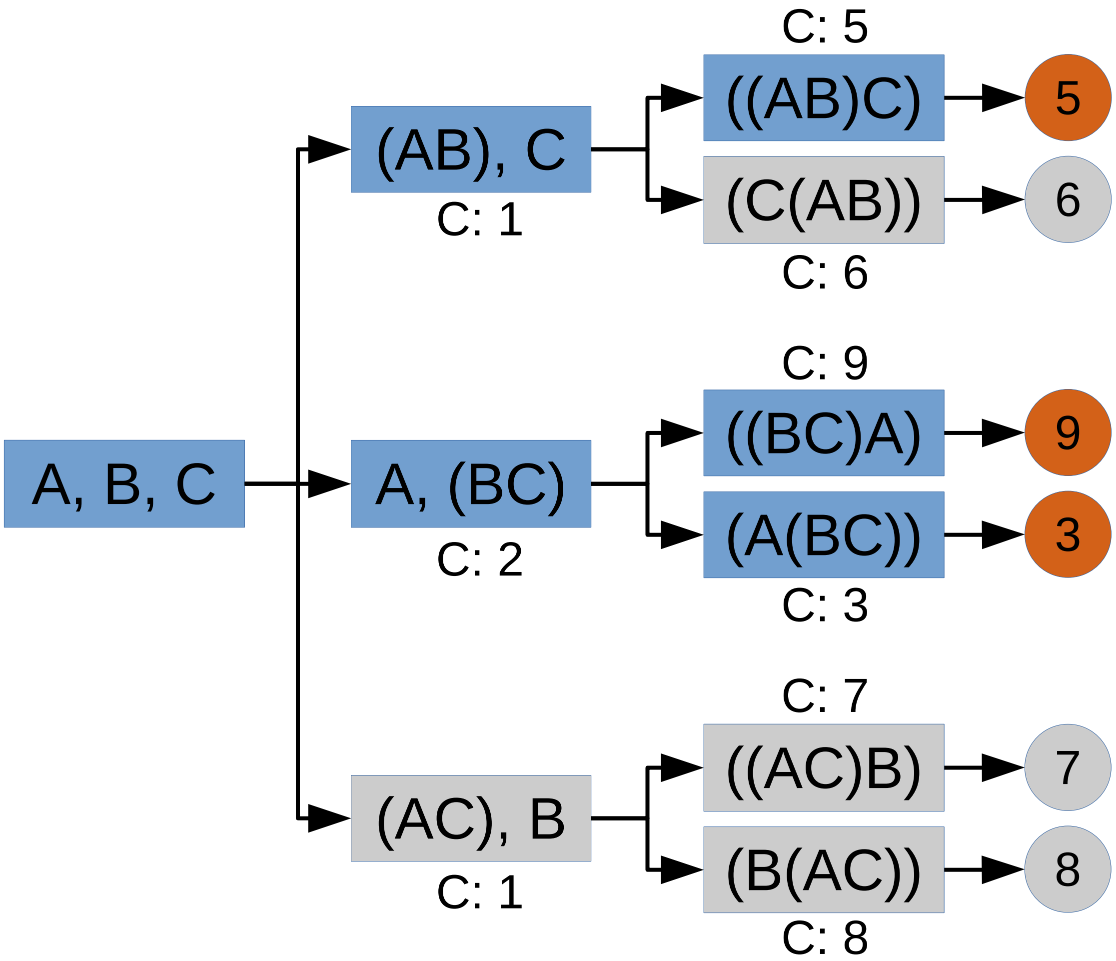
Deep Reinforcement Learning



Traditional Cost Model

A cost function C which estimates the intermediate cost of plan

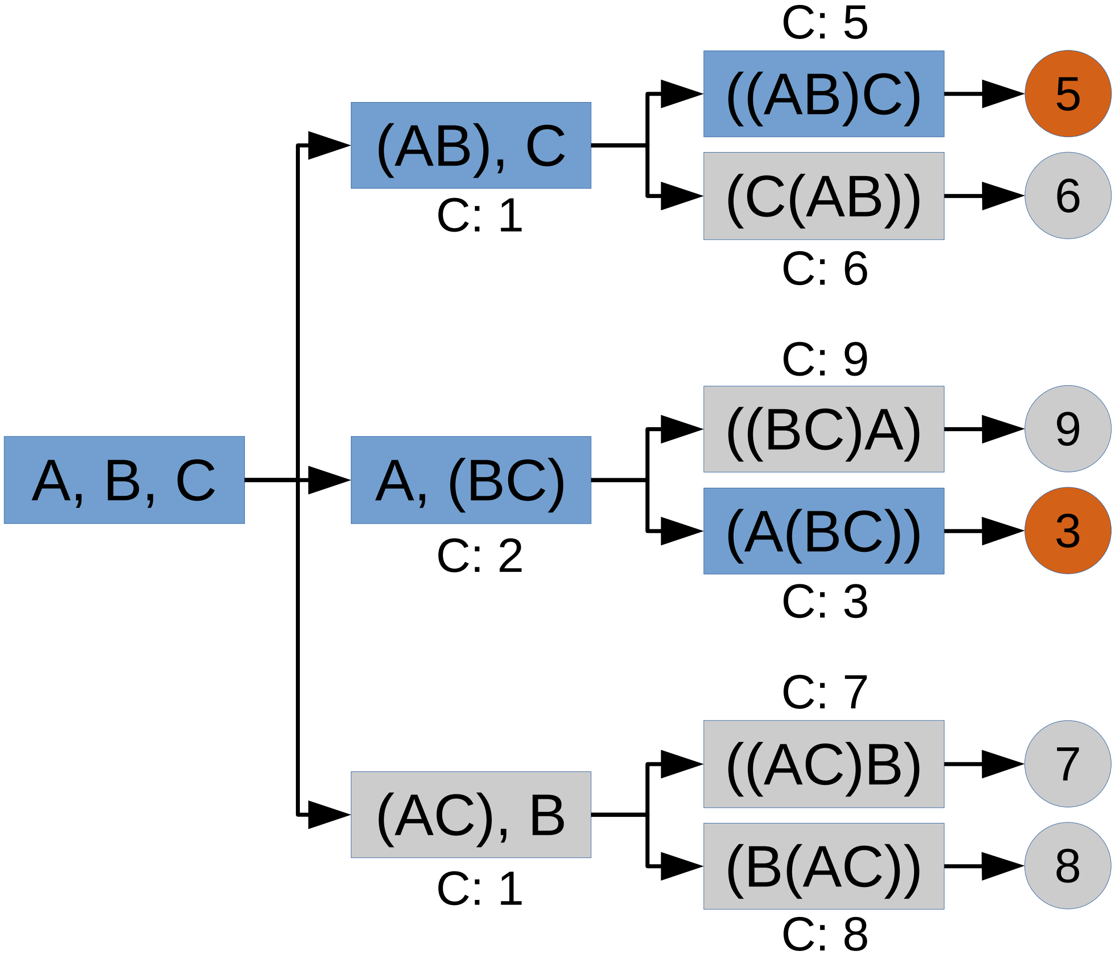
Deep Reinforcement Learning



<u>Traditional Cost Model</u>

A cost function C which estimates the intermediate cost of plan

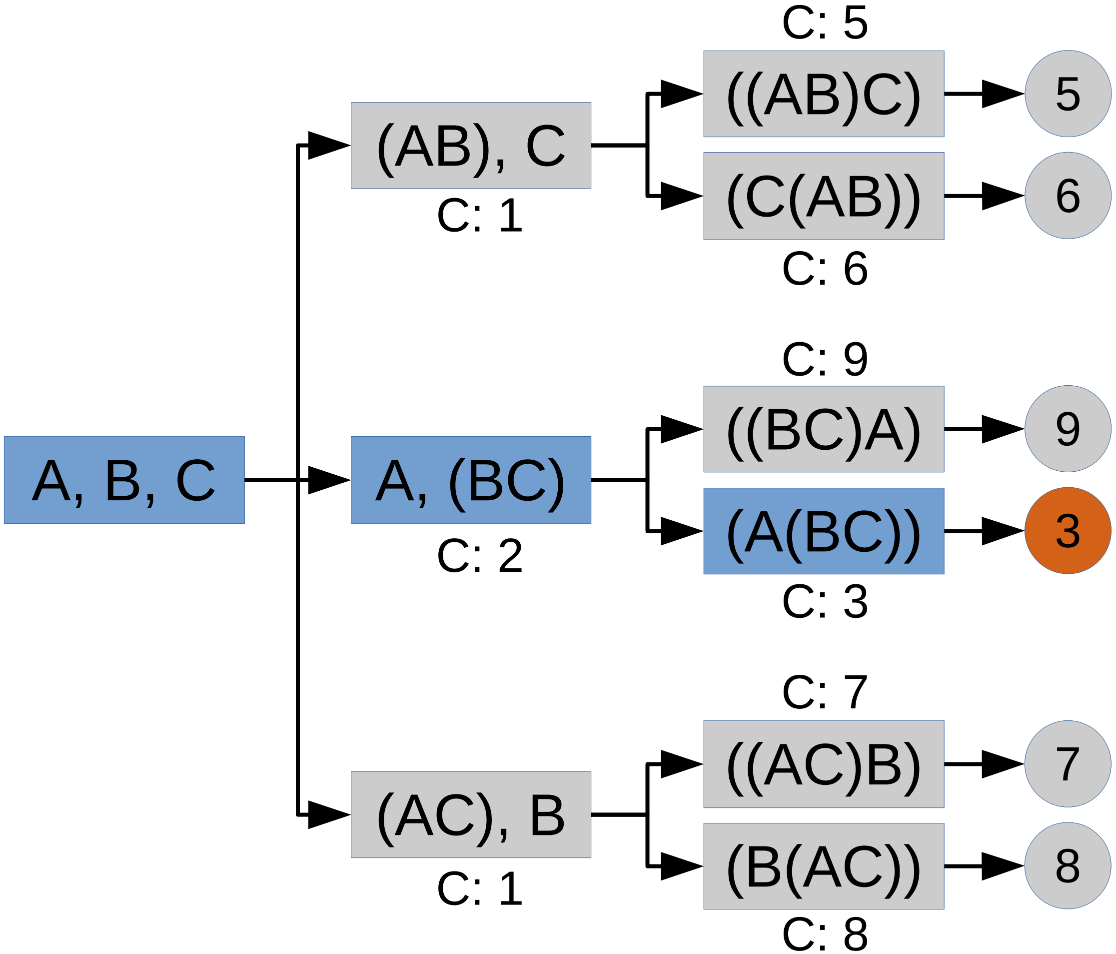
Deep Reinforcement Learning



<u>Traditional Cost Model</u>

A cost function C which estimates the intermediate cost of plan

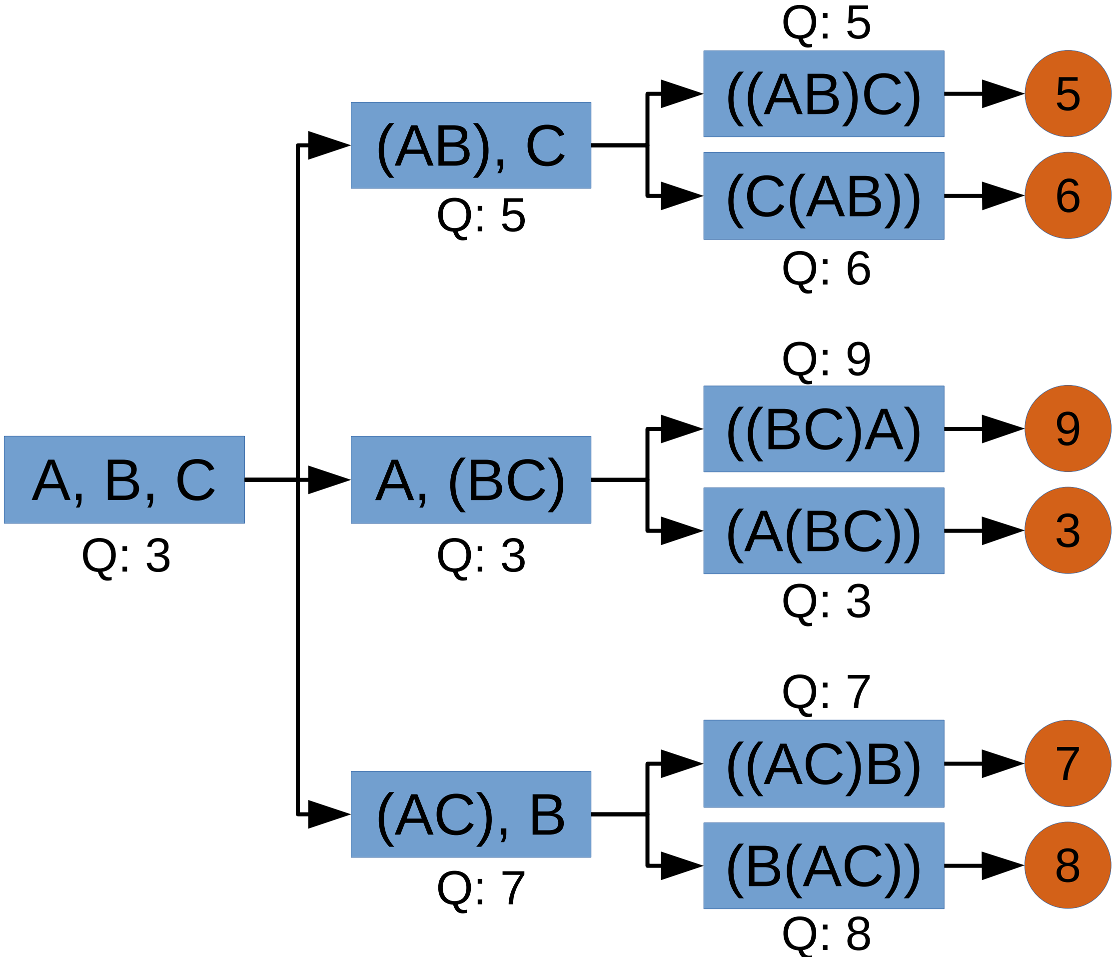
Deep Reinforcement Learning



<u>Traditional Cost Model</u>

A cost function C which estimates the intermediate cost of plan

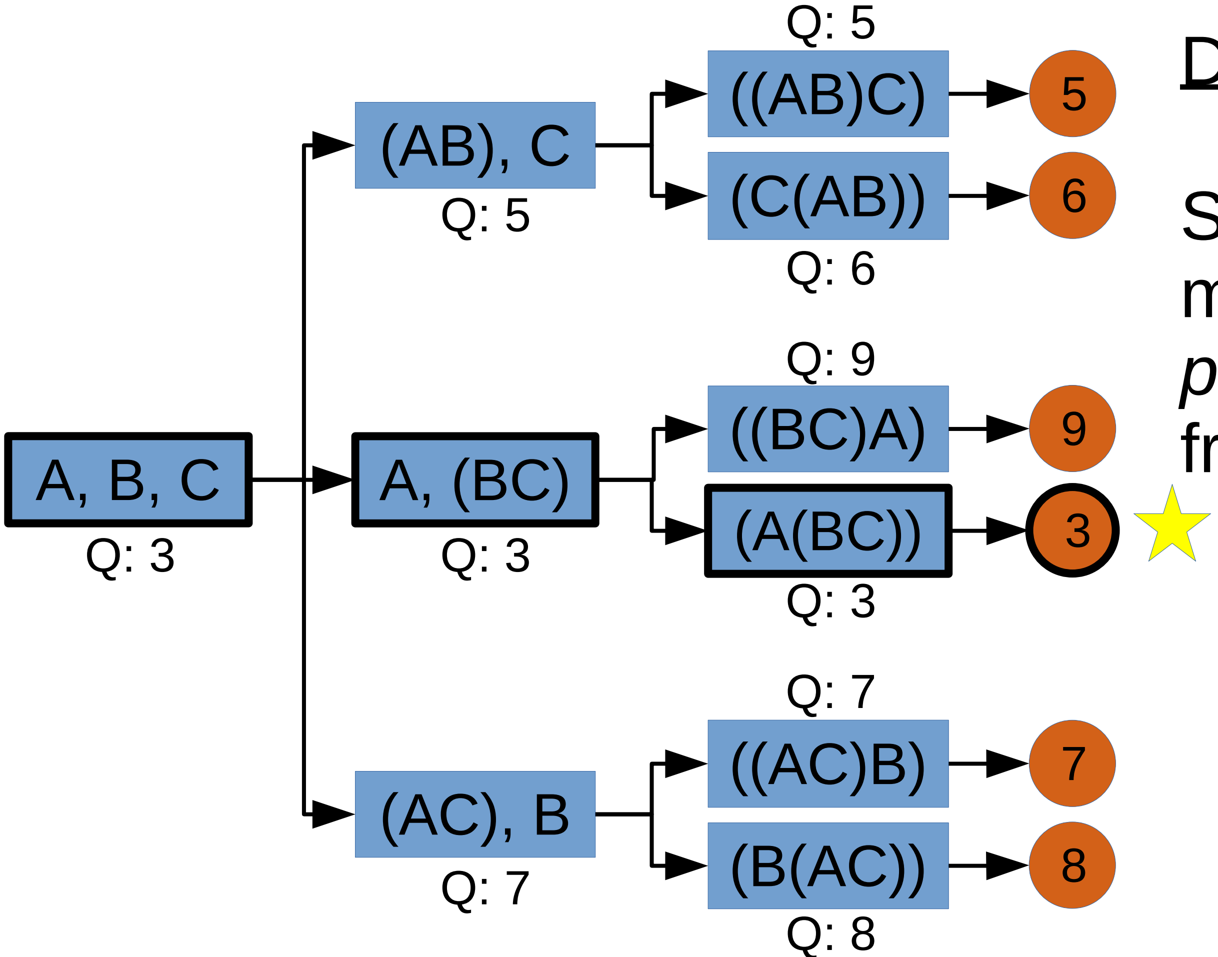
Deep Reinforcement Learning



<u>Deep reinforcement learning</u>

Supp. an oracle Q(·) which maps each state to the *best possible latency achievable* from that state.

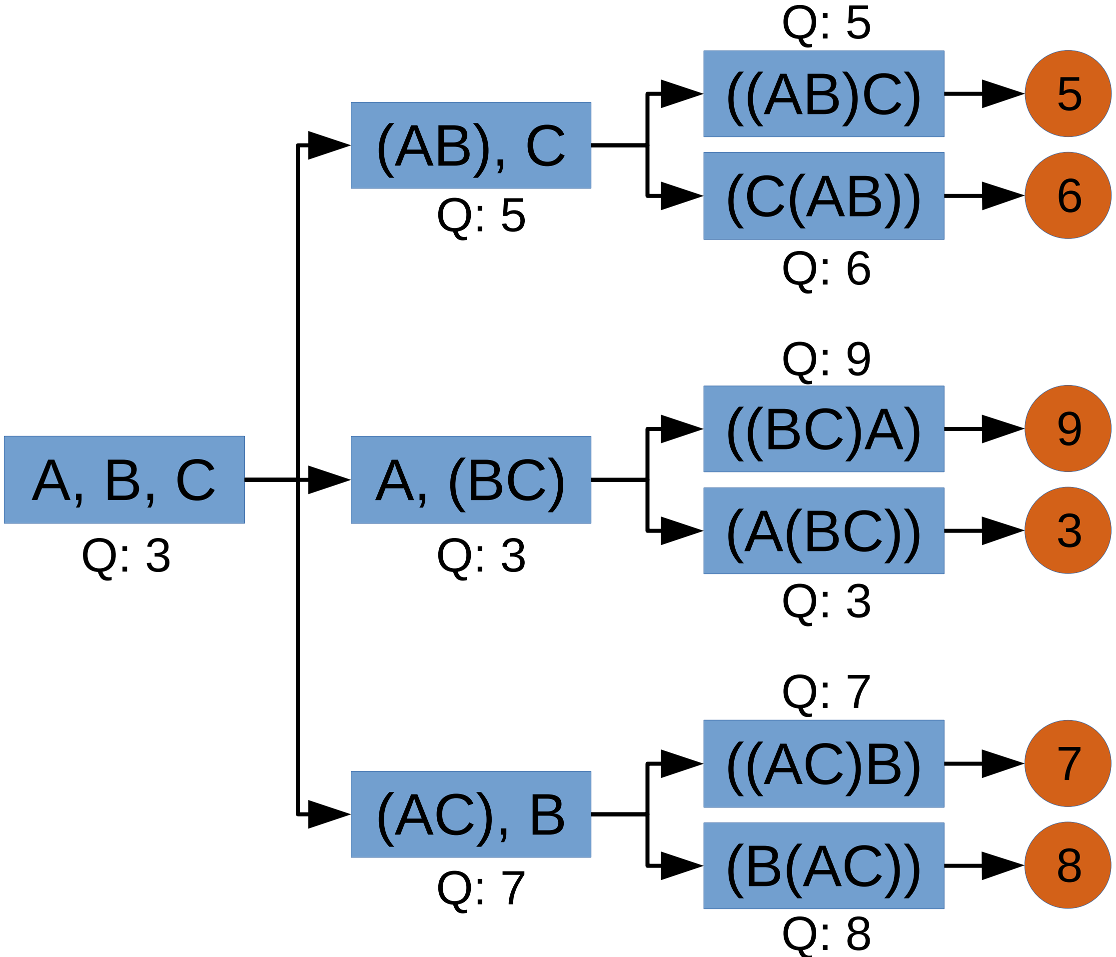
Deep Reinforcement Learning



<u>Deep reinforcement learning</u>

Supp. an oracle Q(·) which maps each state to the *best possible latency achievable* from that state.

Deep Reinforcement Learning



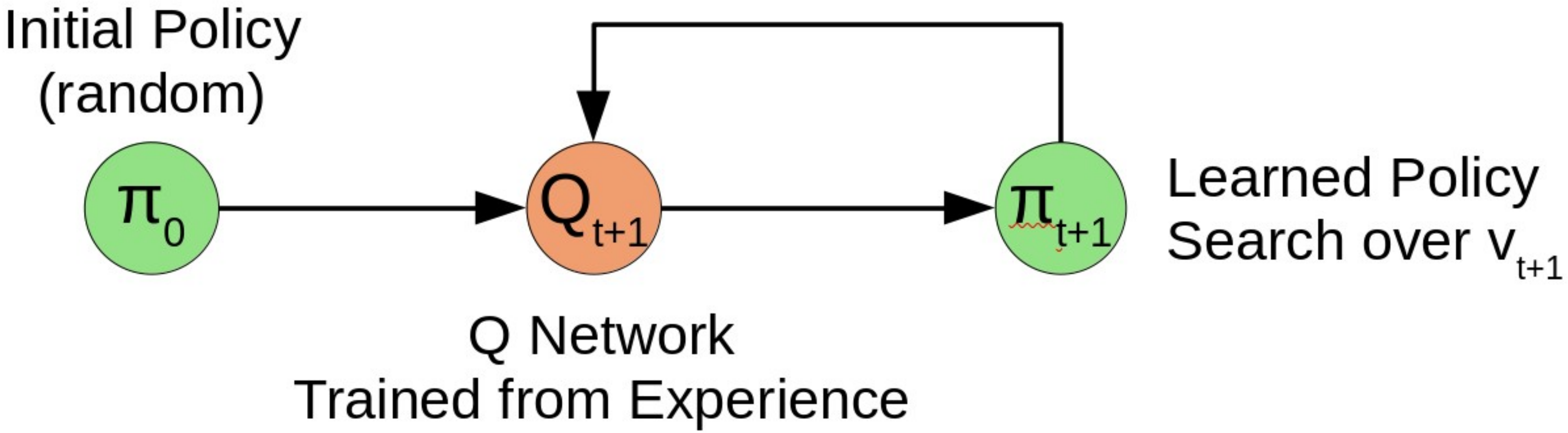
<u>Deep reinforcement learning</u>

Supp. an oracle Q(·) which maps each state to the *best possible latency achievable* from that state.

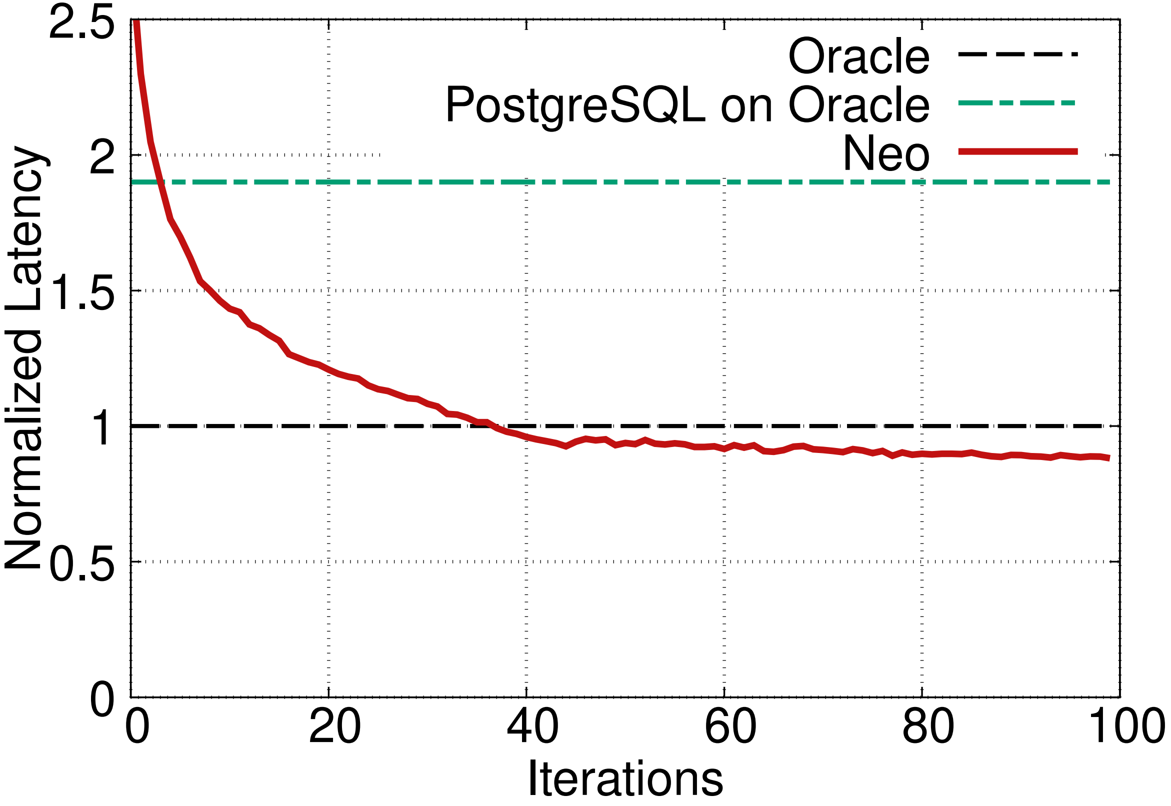
Of course, there's no Q(·).

… so we will learn an approximation, Q̂

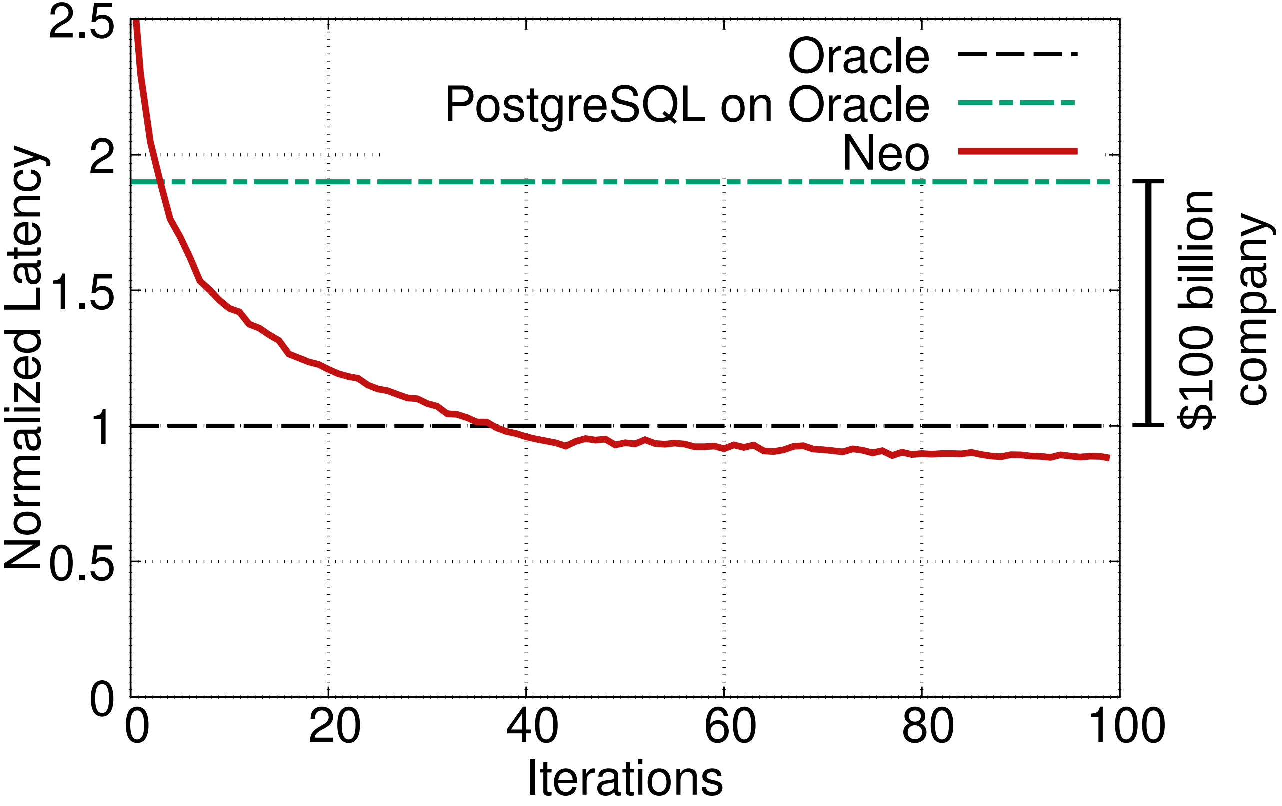
Value Iteration



Neo

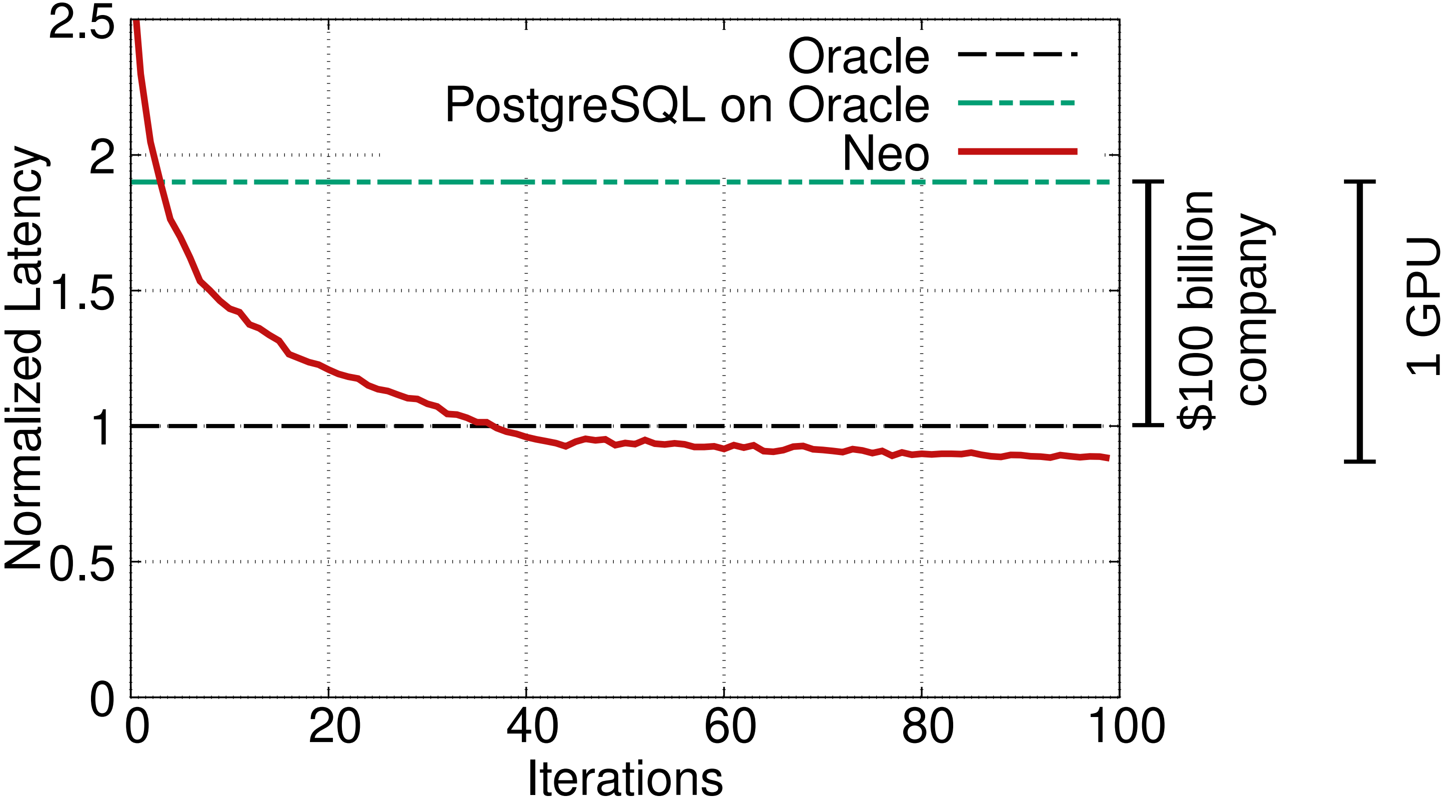

Oracle
PostgreSQL on Oracle
Neo
Normalized Latency
Iterations
2.5
2
1.5
1
0.5
0
0
20
40
60
80
100

Neo



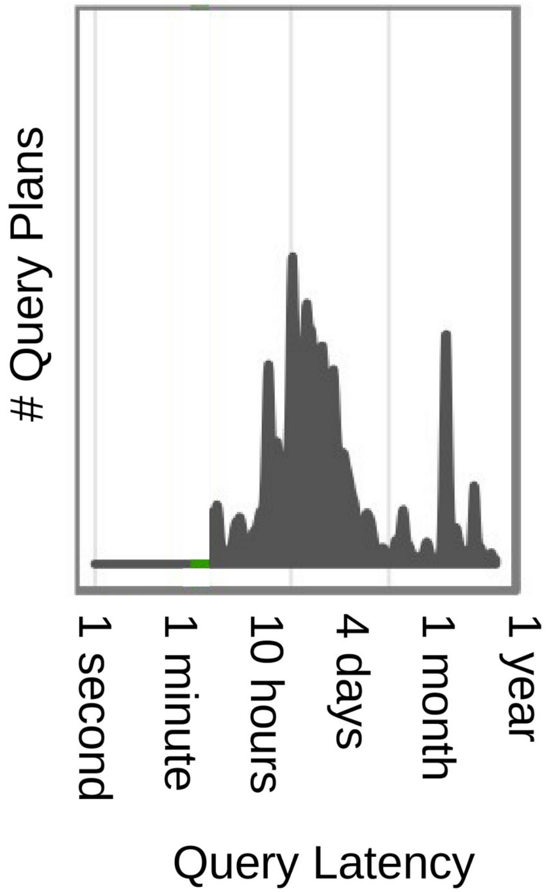
Oracle
PostgreSQL on Oracle
Neo
Normalized Latency
Iterations
0
0.5
1
1.5
2
2.5
0
20
40
60
80
100
$100 billion company

Neo



Oracle
PostgreSQL on Oracle
Neo
Normalized Latency
Iterations
2.5
2
1.5
1
0.5
0
0
20
40
60
80
100
$100 billion
company
1 GPU

This Ain't Mario



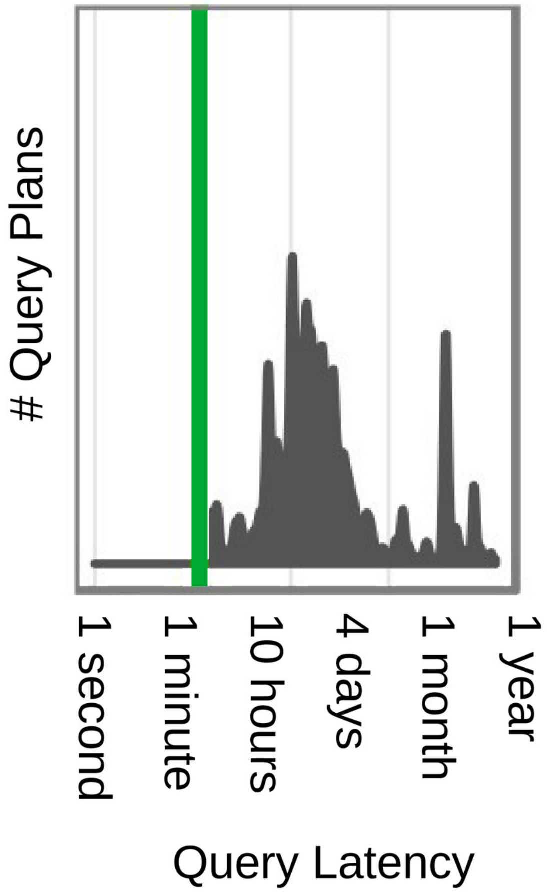
Neo worked great *on average…*

But sometimes picked terrible plans.

Unlike most RL problems, doing worse takes longer. Makes sample inefficient methods even worse.

How do traditional optimizers compare?

This Ain't Mario



Neo worked great *on average…*

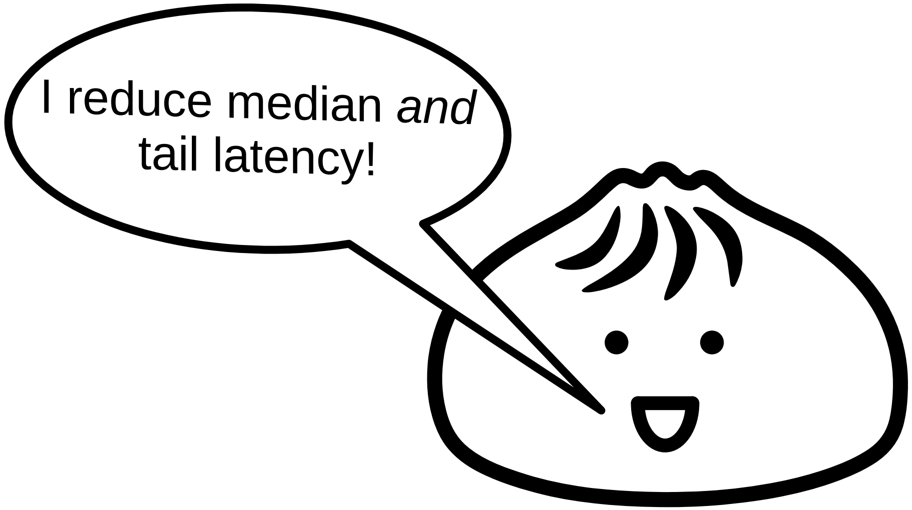
But sometimes picked terrible plans.

Unlike most RL problems, doing worse takes longer. Makes sample inefficient methods even worse.

How do traditional optimizers compare?

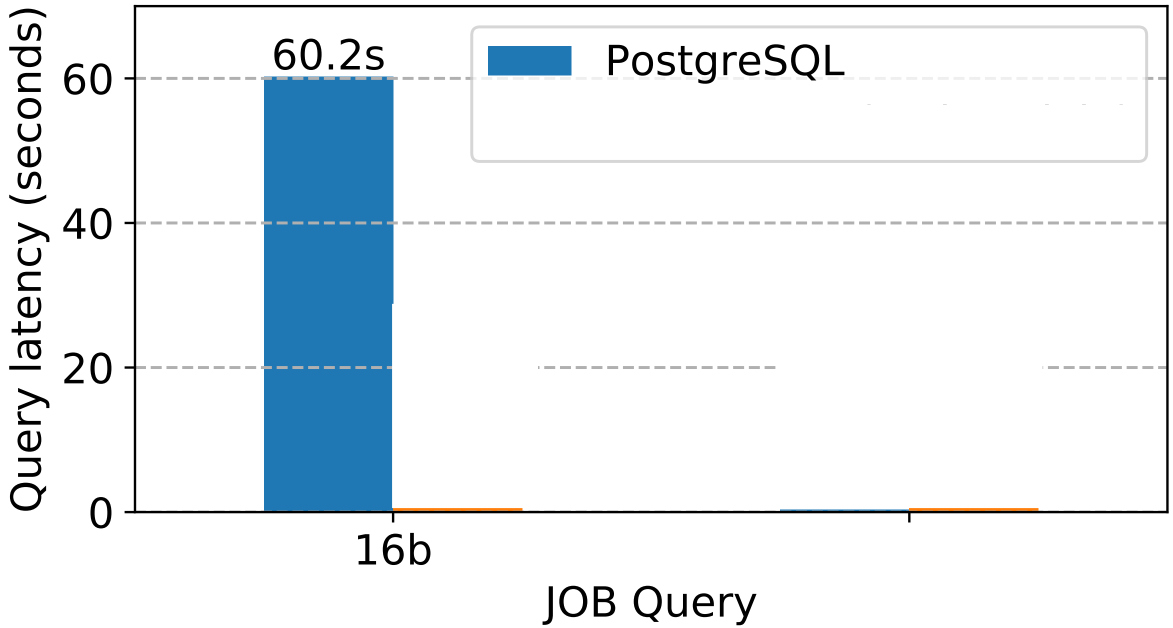
Introducing Bao

- Bao: Bandit optimizer
- By *steering* a traditional query optimizer, Bao:
  - Outperforms after *1 hour*
  - Reduces 99% latency
  - Adapts to changes in workload, schema, and data.



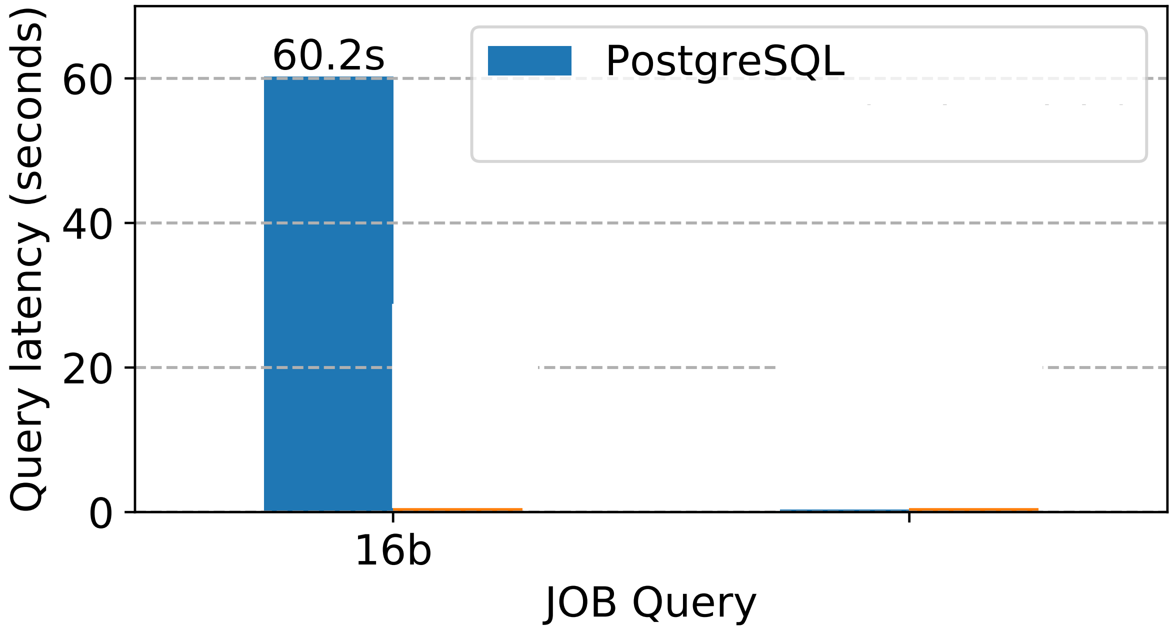
Query Hints

Slow query.



Query Hints

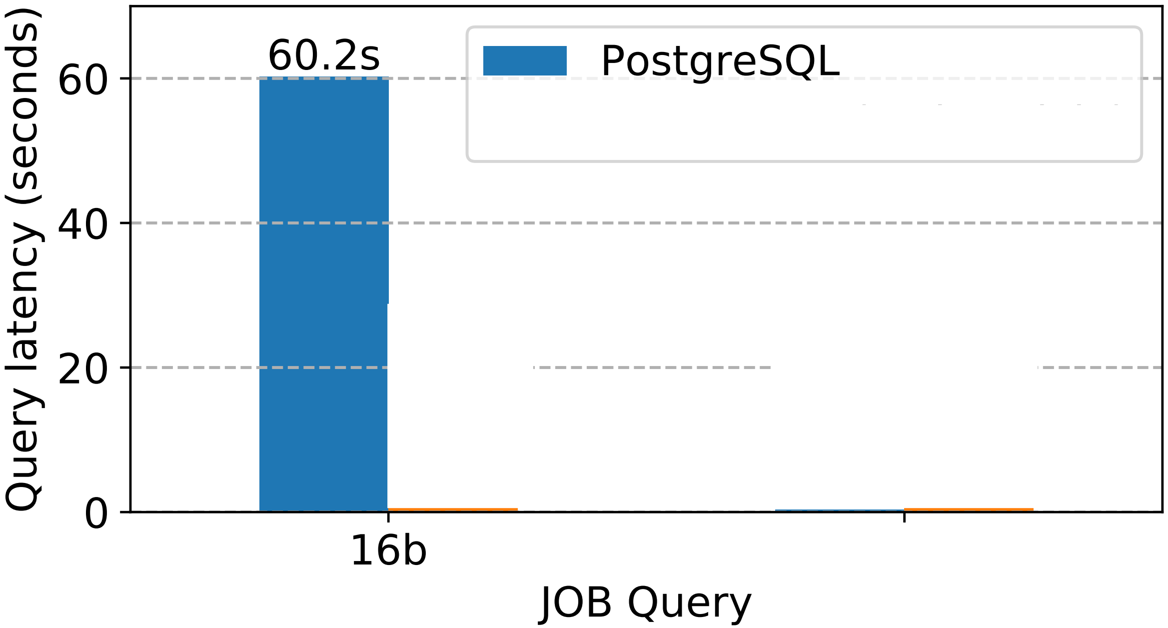
Slow query. Run EXPLAIN.
> Loop join plan,
> Low selectivity



Query Hints

Slow query. Run EXPLAIN.
> Loop join plan,
> Low selectivity

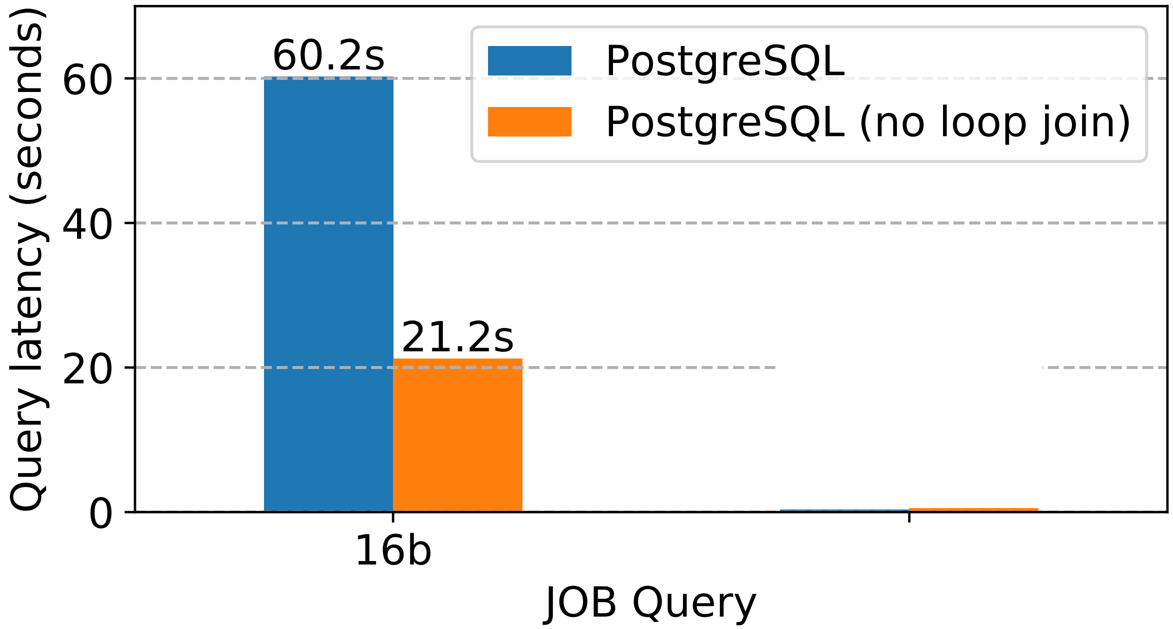
Try disabling loop join
> ...



Query Hints

Slow query. Run EXPLAIN.
> Loop join plan,
> Low selectivity

Try disabling loop join
> Huge improvement

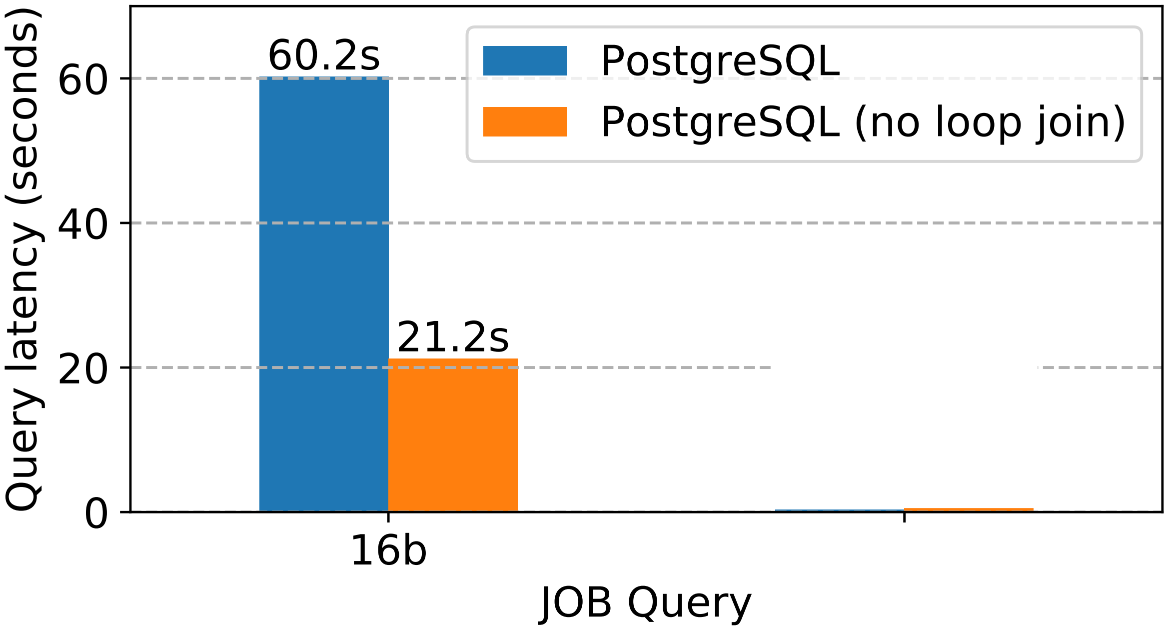


Query Hints

Slow query. Run EXPLAIN.
> Loop join plan,
> Low selectivity

Try disabling loop join
> Huge improvement

Apply this hint globally
> …

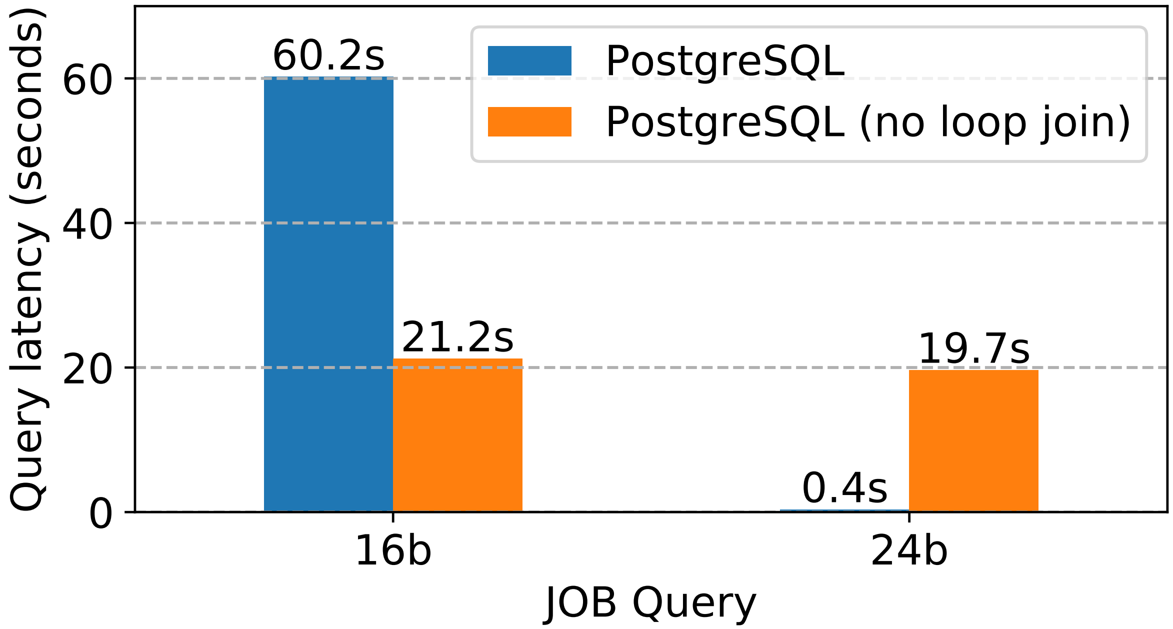


Query Hints

Slow query. Run EXPLAIN.
> Loop join plan,
> Low selectivity

Try disabling loop join
> Huge improvement

Apply this hint globally
> … other regressions

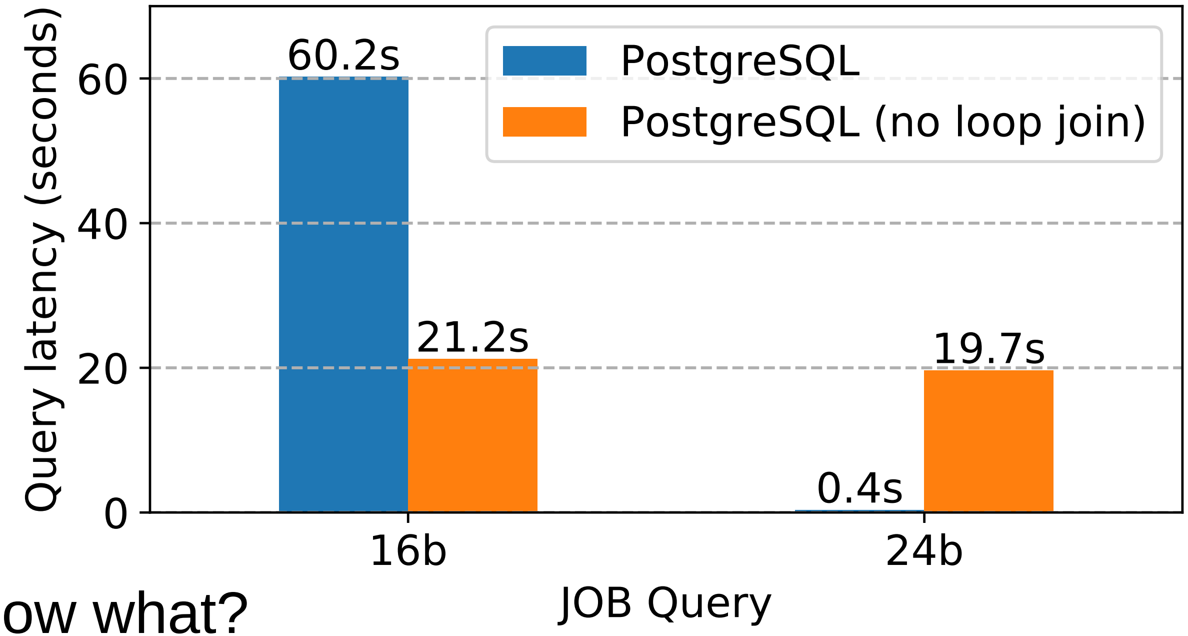


Query Hints

Slow query. Run EXPLAIN.
> Loop join plan,
> Low selectivity

Try disabling loop join
> Huge improvement

Apply this hint globally
> … other regressions



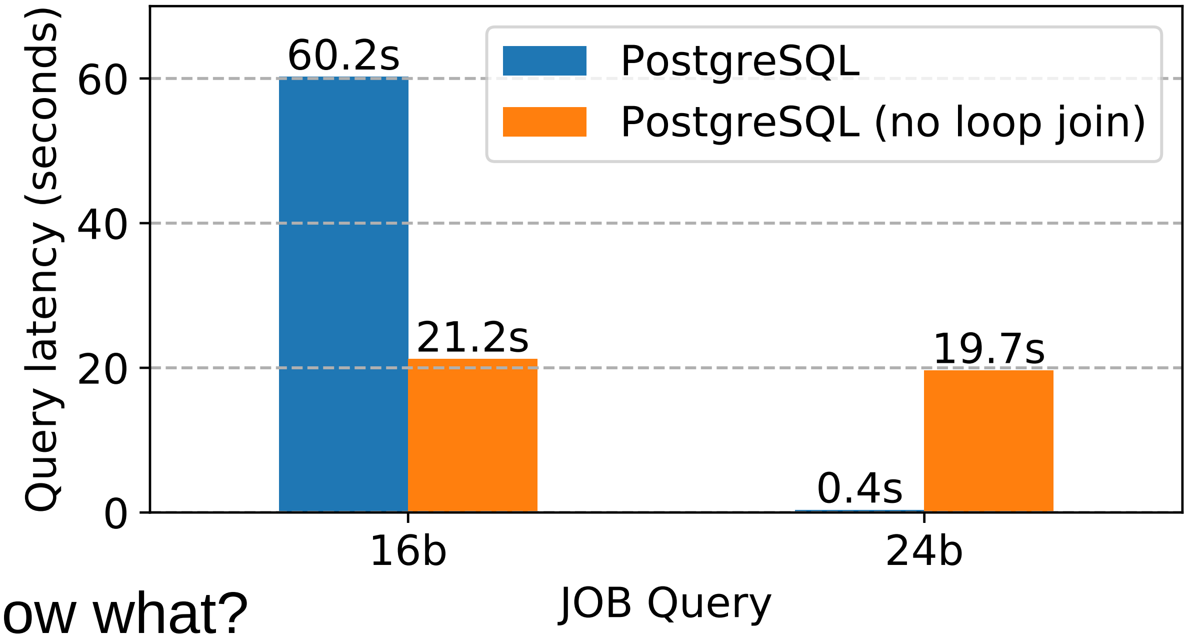
Undo that, need local hints. Now what?

Query Hints

Slow query. Run EXPLAIN.
> Loop join plan,
> Low selectivity

Try disabling loop join
> Huge improvement

Apply this hint globally
> … other regressions



Undo that, need local hints. Now what?

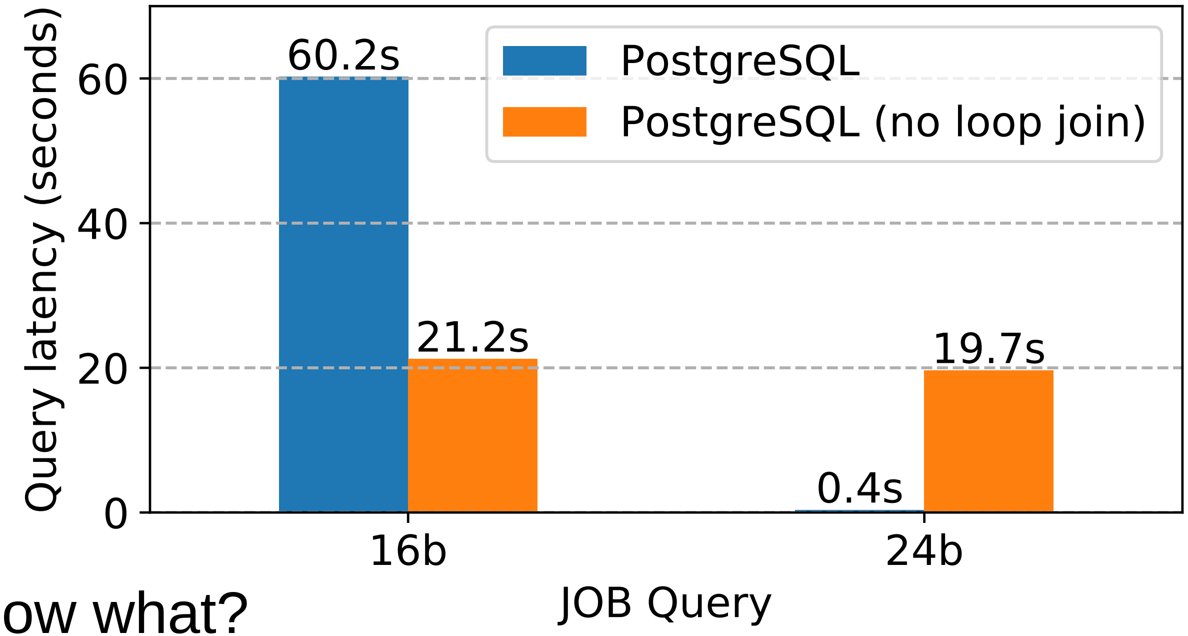
Opt 1: Apply the hint to every instance of the query

Query Hints

Slow query. Run EXPLAIN.
> Loop join plan,
> Low selectivity

Try disabling loop join
> Huge improvement

Apply this hint globally
> … other regressions



Undo that, need local hints. Now what?

Opt 1: Apply the hint to every instance of the query

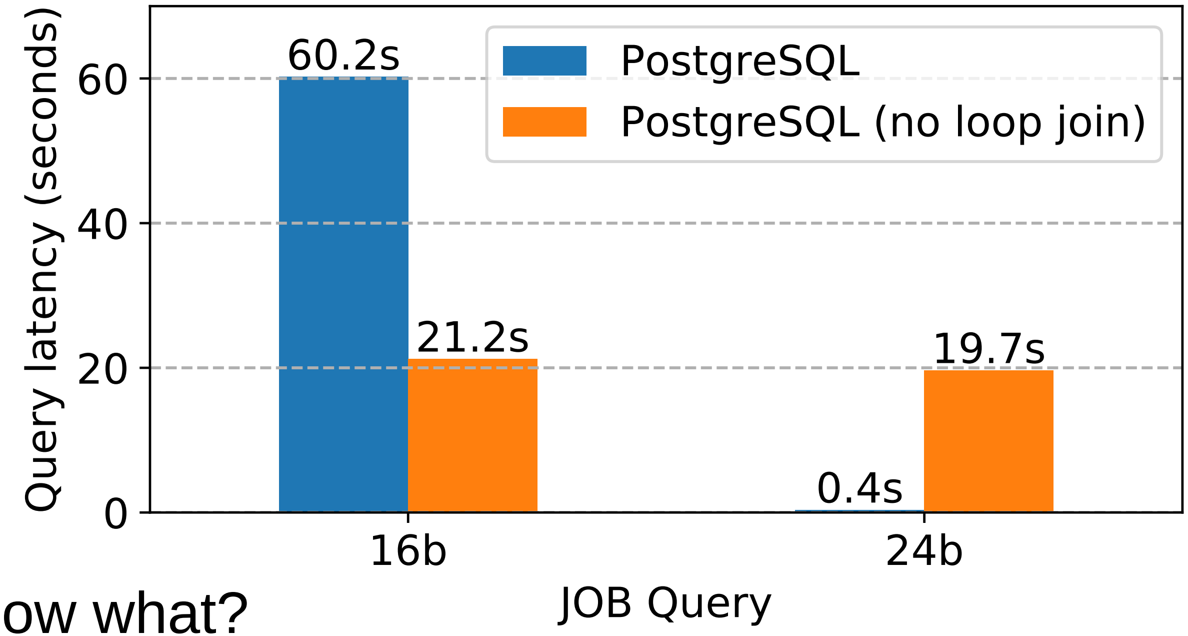
Opt 2: Set as default, find regressions, add hints to those queries

Query Hints

Slow query. Run EXPLAIN.
> Loop join plan,
> Low selectivity

Try disabling loop join
> Huge improvement

Apply this hint globally
> … other regressions



Undo that, need local hints. Now what?

Opt 1: Apply the hint to every instance of the query

Opt 2: Set as default, find regressions, add hints to those queries

Opt 3: Give up

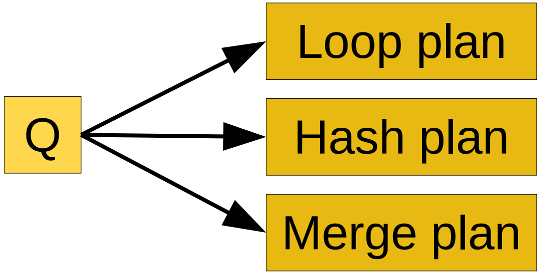
Bao

- Bao automatically determines the right hint to use.
- Consider different hints as *arms* in a *contextual multi-armed bandit*

Q

Bao

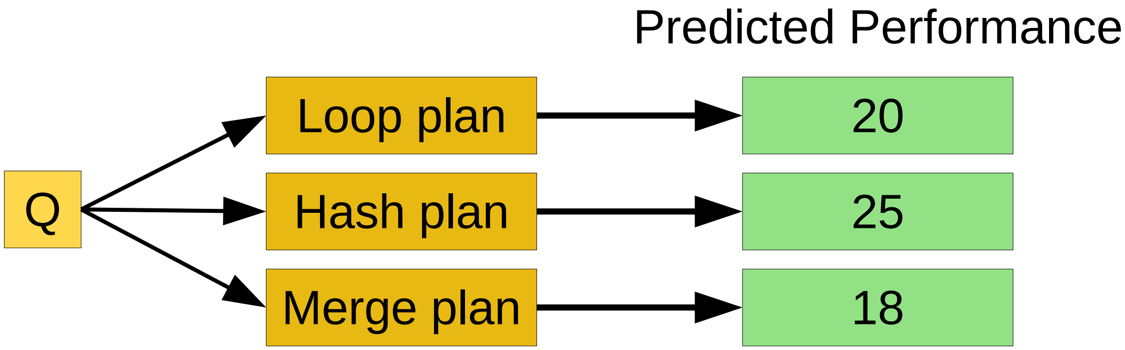
- Bao automatically determines the right hint to use.
- Consider different hints as *arms* in a *contextual multi-armed bandit*



Traditional Query Optimizer

Bao

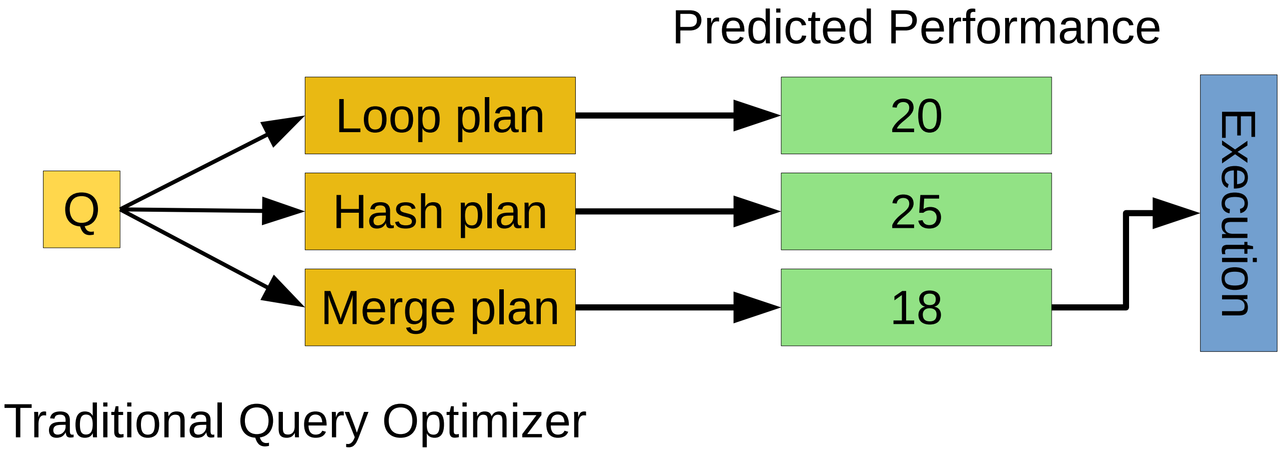
- Bao automatically determines the right hint to use.
- Consider different hints as *arms* in a *contextual multi-armed bandit*



Traditional Query Optimizer

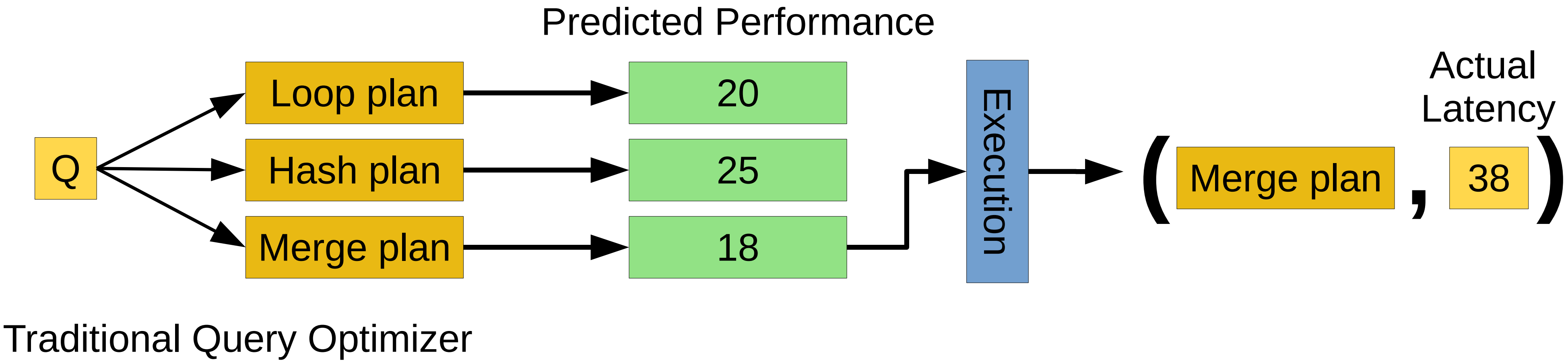
Bao

- Bao automatically determines the right hint to use.
- Consider different hints as *arms* in a *contextual multi-armed bandit*



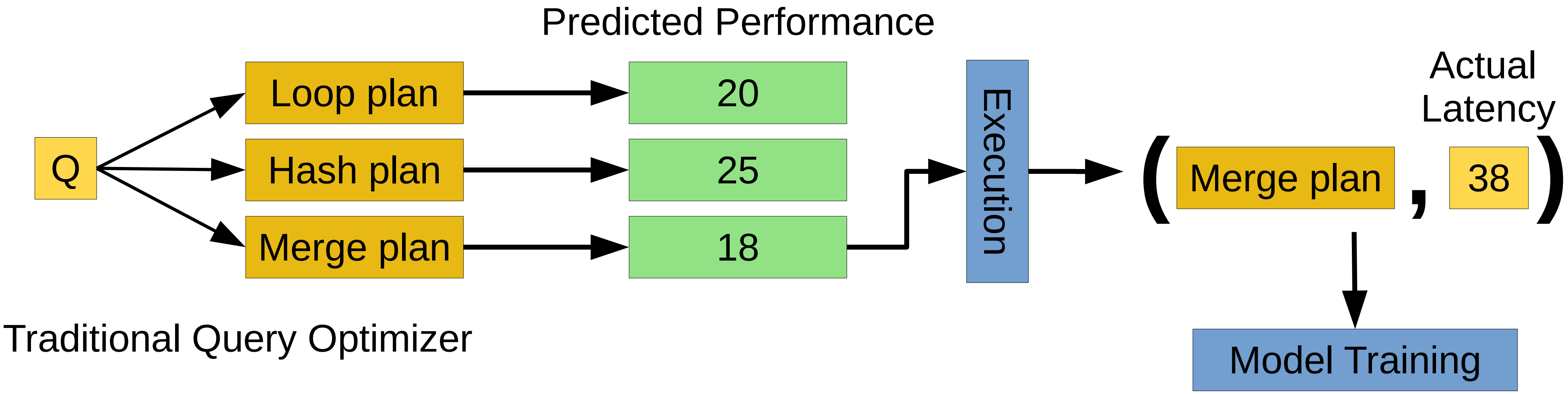
Bao

- Bao automatically determines the right hint to use.
- Consider different hints as *arms* in a *contextual multi-armed bandit*



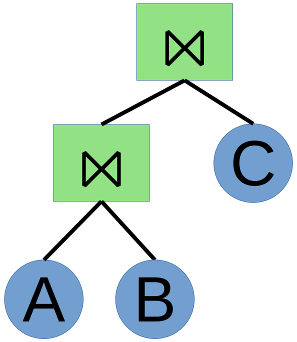
Bao

- Bao automatically determines the right hint to use.
- Consider different hints as *arms* in a *contextual multi-armed bandit*



Predictive Model

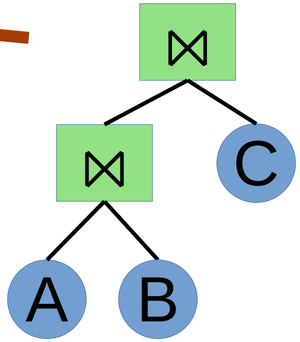
- Bao needs a good predictive model.
- Problem: Query plans have a tree structure.
- Solution: flatten the tree into a vector and engineer some features



Predictive Model

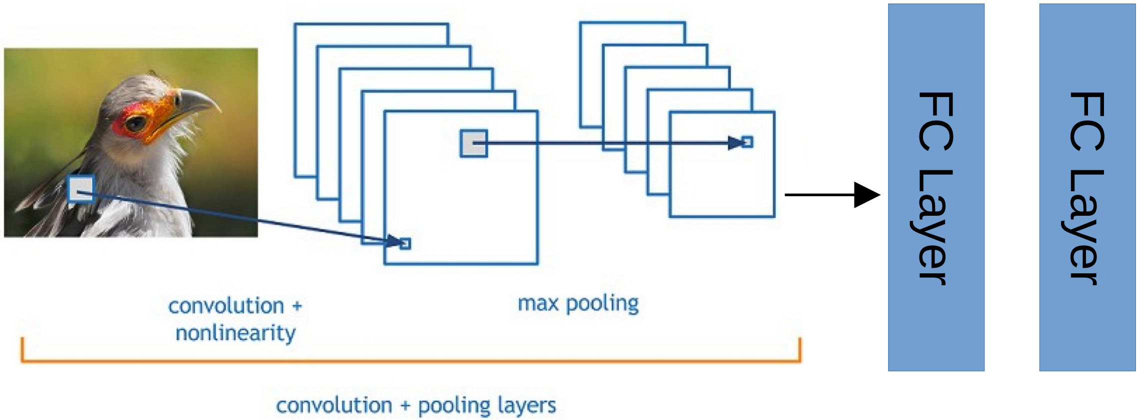
- Bao needs a good predictive model.
- Problem: Query plans have a tree structure.
- ~~Solution: flatten the tree into a vector and engineer some features~~

This is not normally how machine learning is effective.

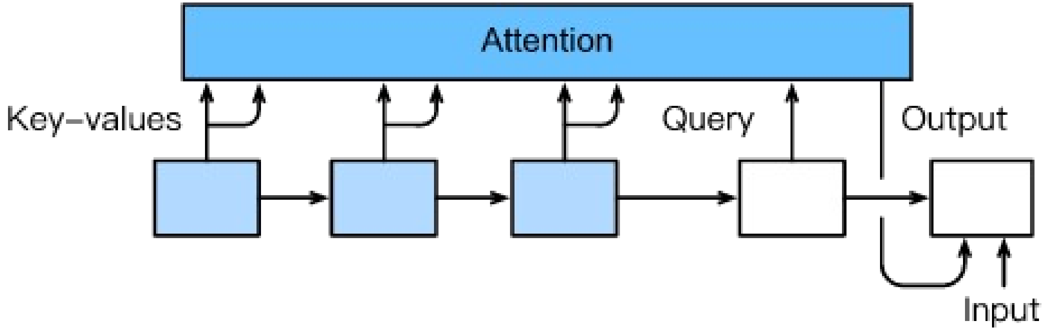


What makes ML *good?*

Convolution Neural Networks (CNNs)

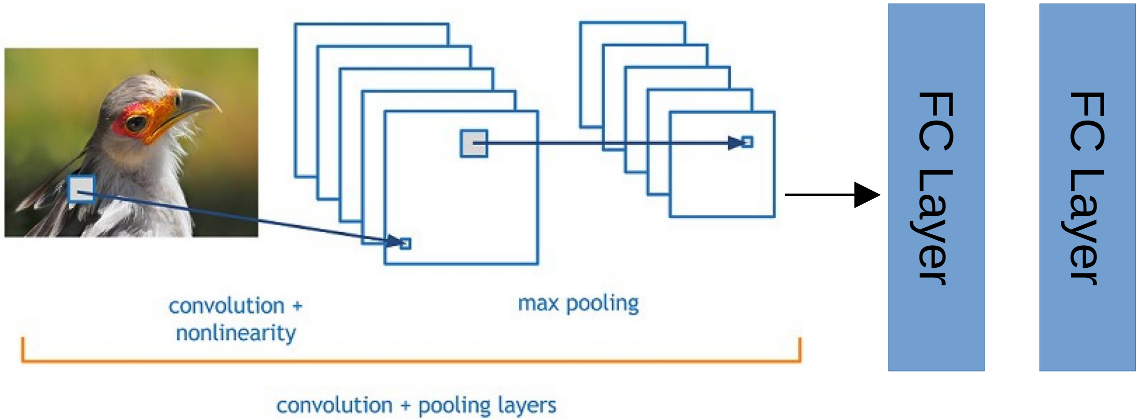


Attention mechanisms

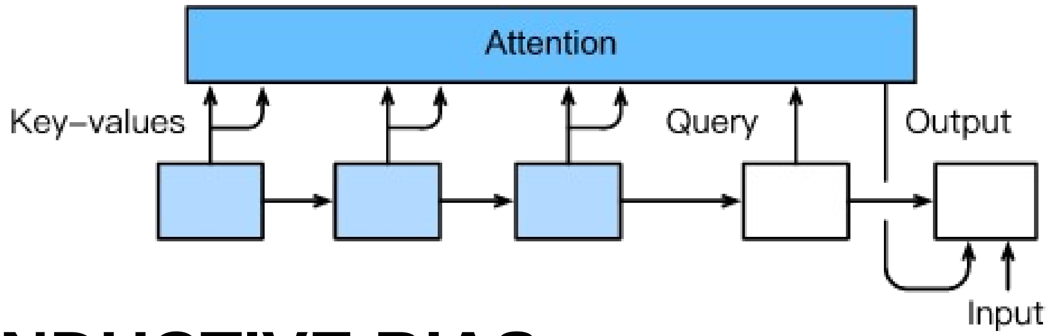


What makes ML *good?*

Convolution Neural Networks (CNNs)



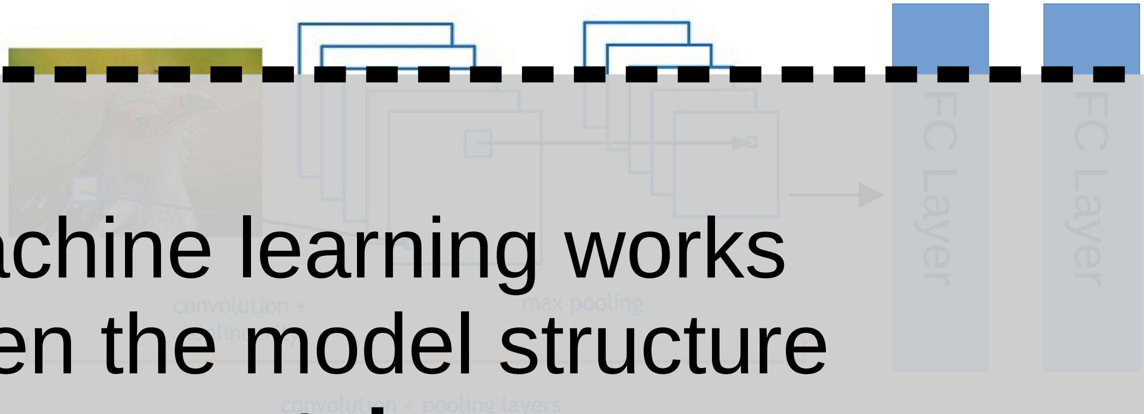
Attention mechanisms



INDUCTIVE BIAS

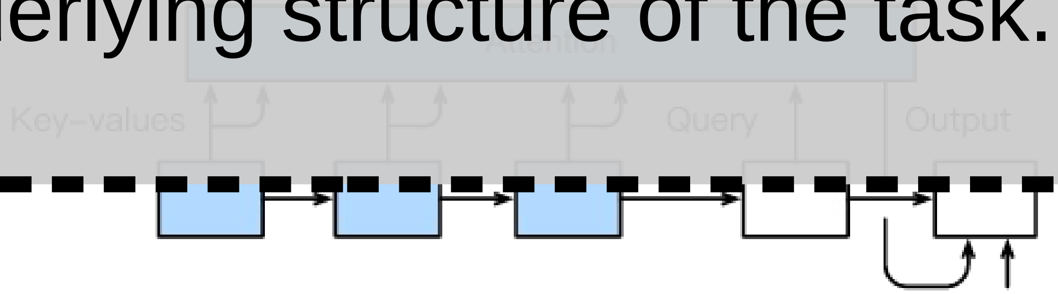
What makes ML *good?*

Convolution Neural Networks (CNNs)



Machine learning works
when the model structure
matches
the underlying structure of the task.

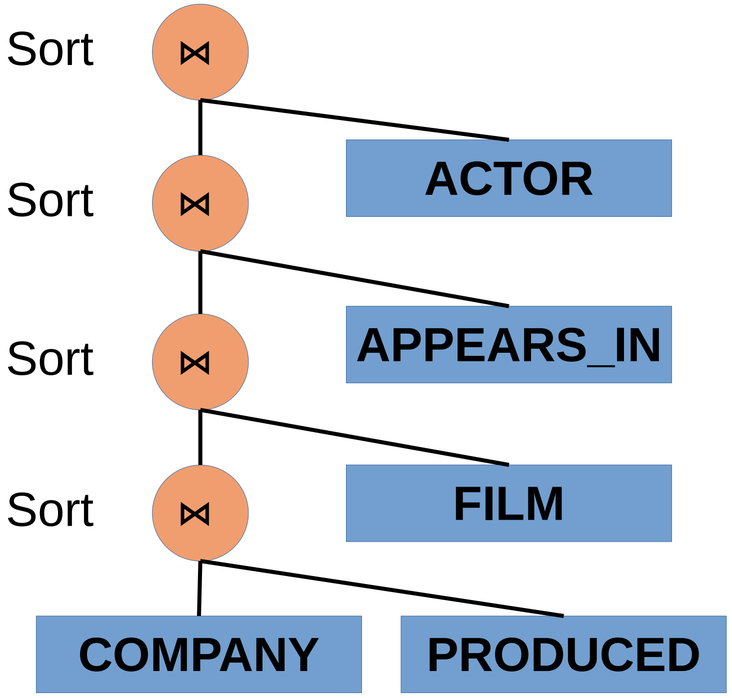
Attention mechanisms



INDUCTIVE BIAS

Tree Convolution

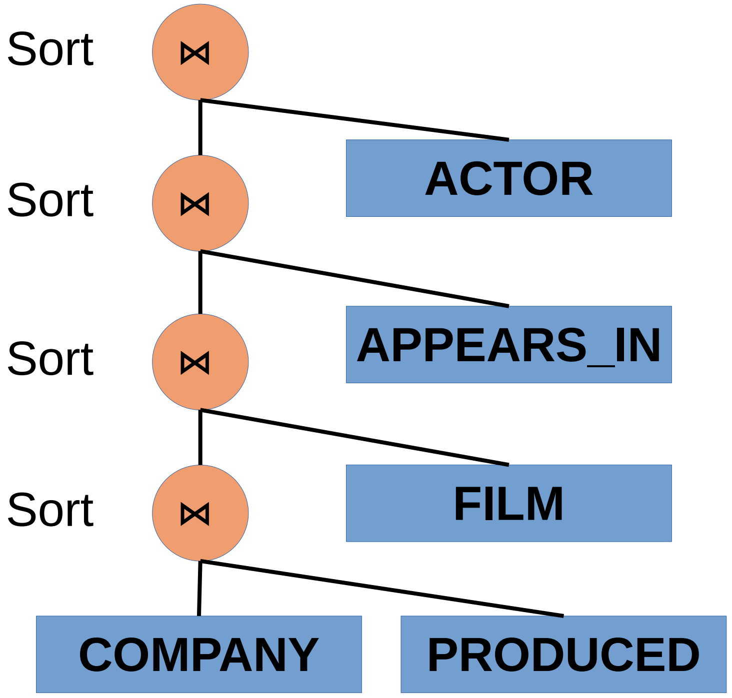
- How do we come up with a good inductive bias for query plans?



Tree Convolution

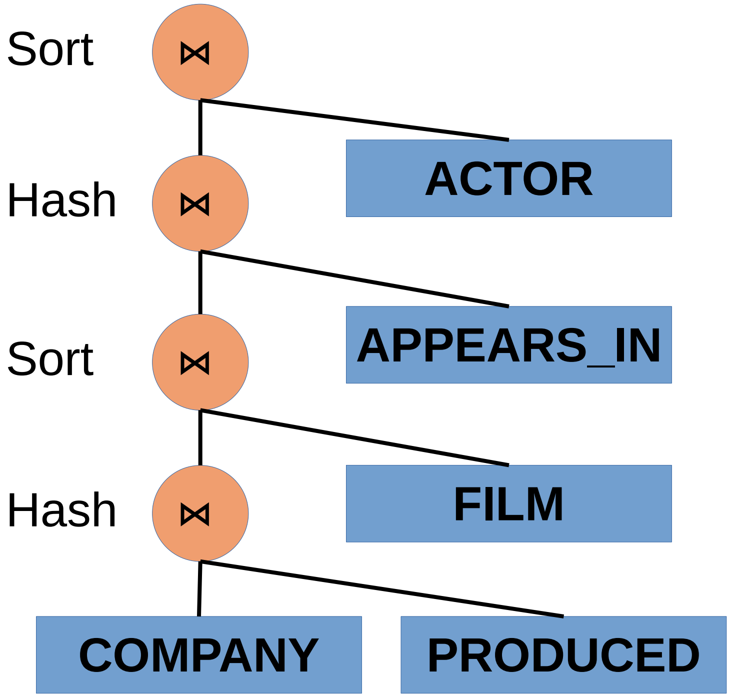
- How do we come up with a good inductive bias for query plans?

“Many stacked sort operators – possibly avoids a resort.”



Tree Convolution

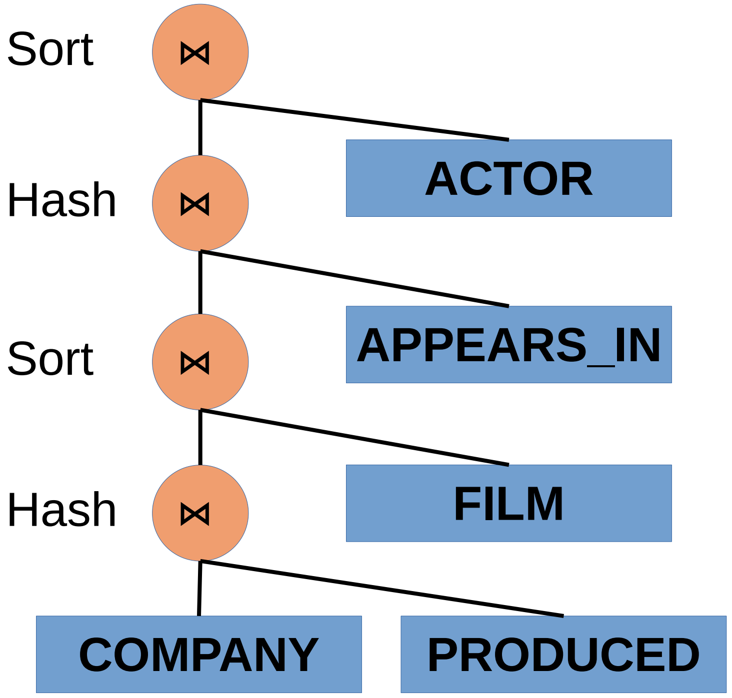
- How do we come up with a good inductive bias for query plans?



Tree Convolution

- How do we come up with a good inductive bias for query plans?

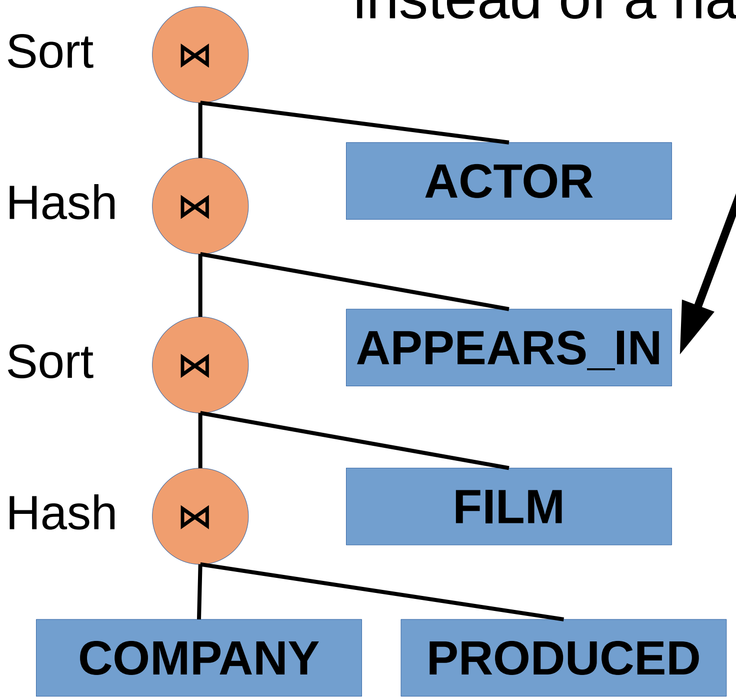
“Hash then sort, 100% requires rehash or resort.”



Tree Convolution

- How do we come up with a good inductive bias for query plans?

“APPEARS_IN” is presorted on disk – should use a sort instead of a hash.

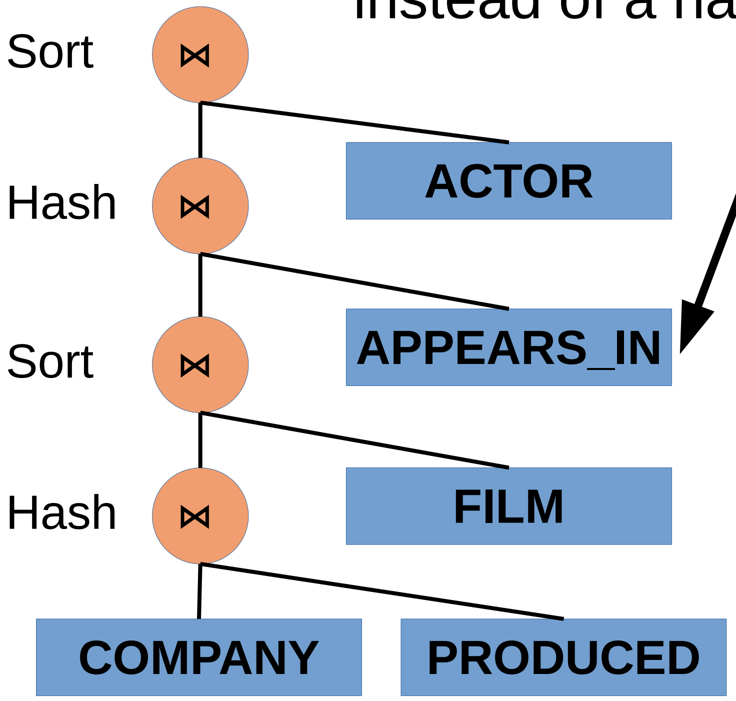


“Hash then sort, 100% requires rehash or resort.”

Tree Convolution

- How do we come up with a good inductive bias for query plans?

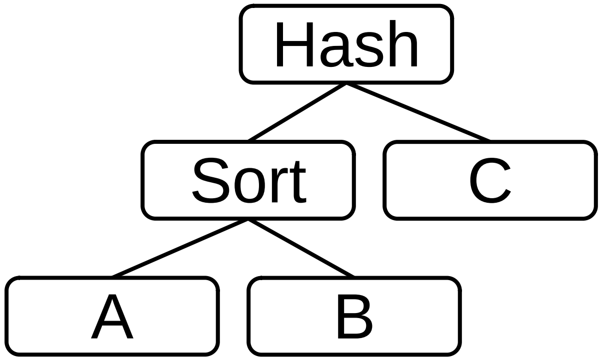
“APPEARS_IN” is presorted on disk – should use a sort instead of a hash.

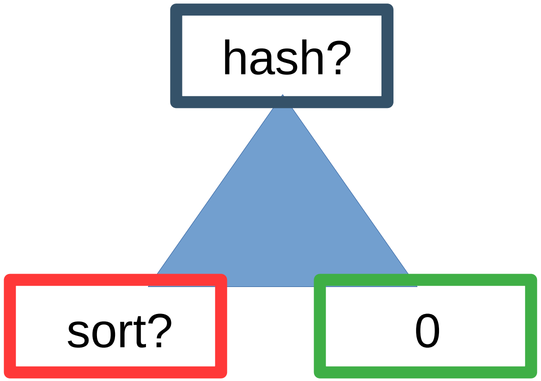


“Hash then sort, 100% requires rehash or resort.”

Experts examine *local structure* first, then look to higher level features.

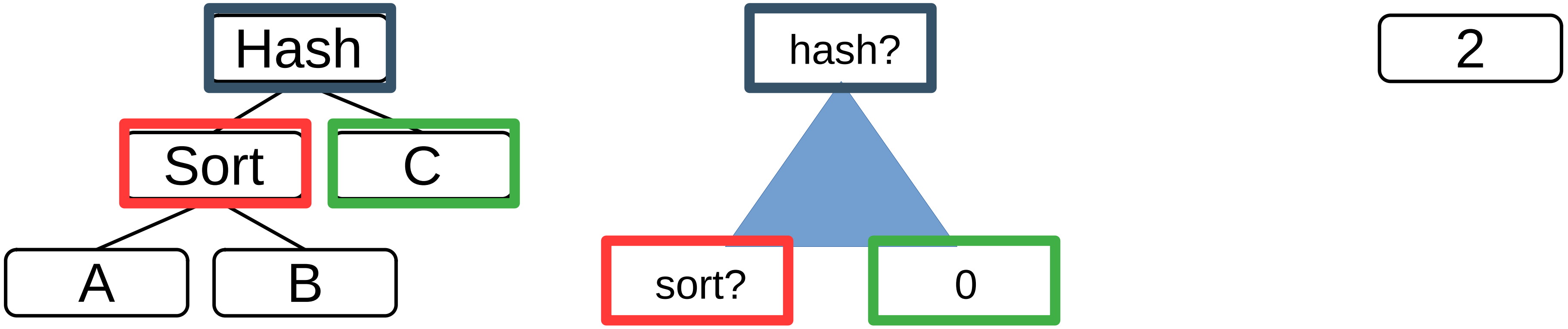
Tree Convolution





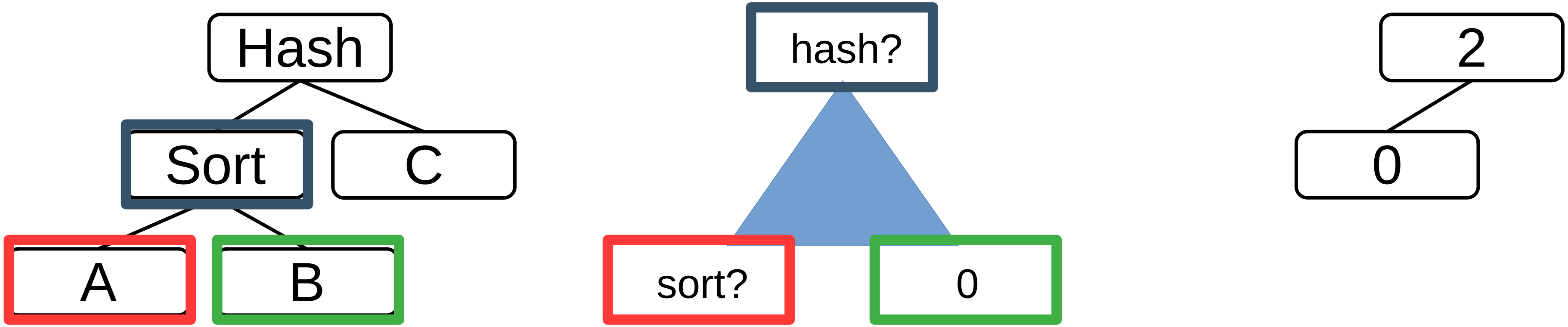
Detects a hash on top of a sort

Tree Convolution



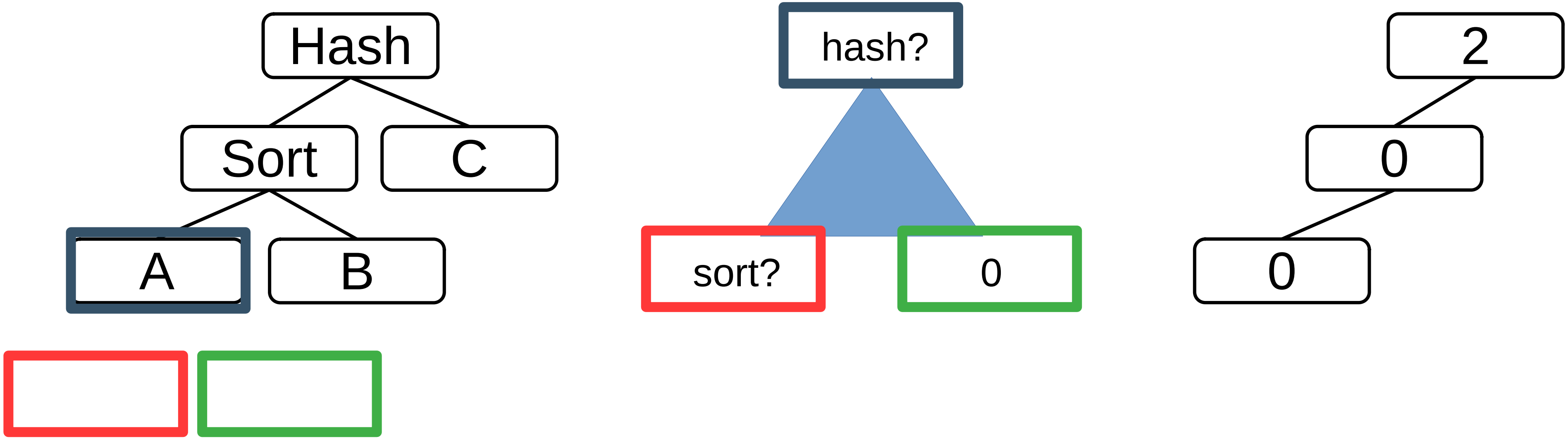
Detects a hash on top of a sort

Tree Convolution



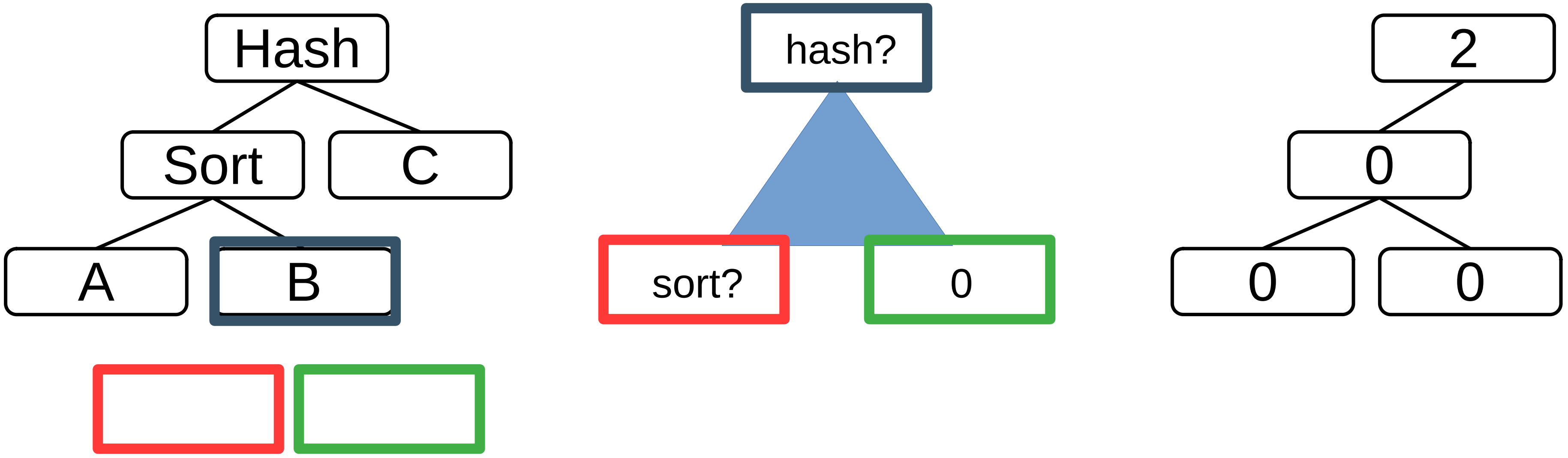
Detects a hash on top of a sort

Tree Convolution



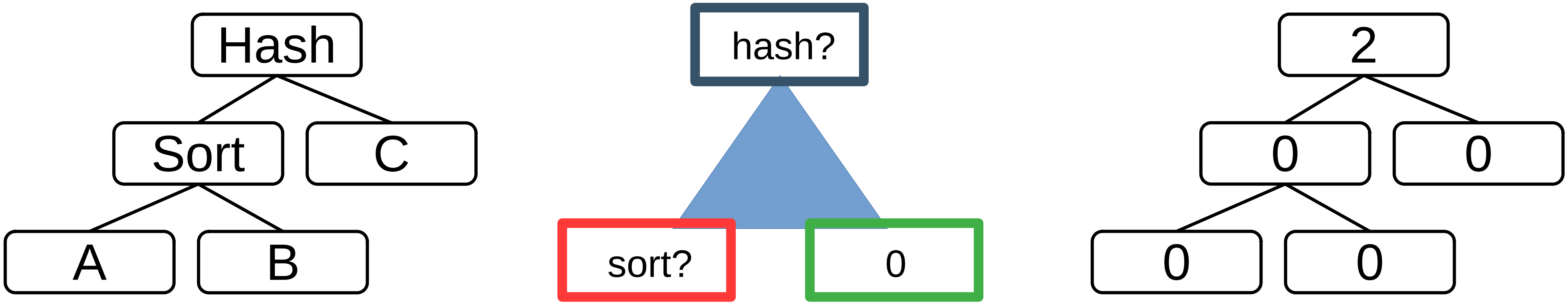
Detects a hash on top of a sort

Tree Convolution



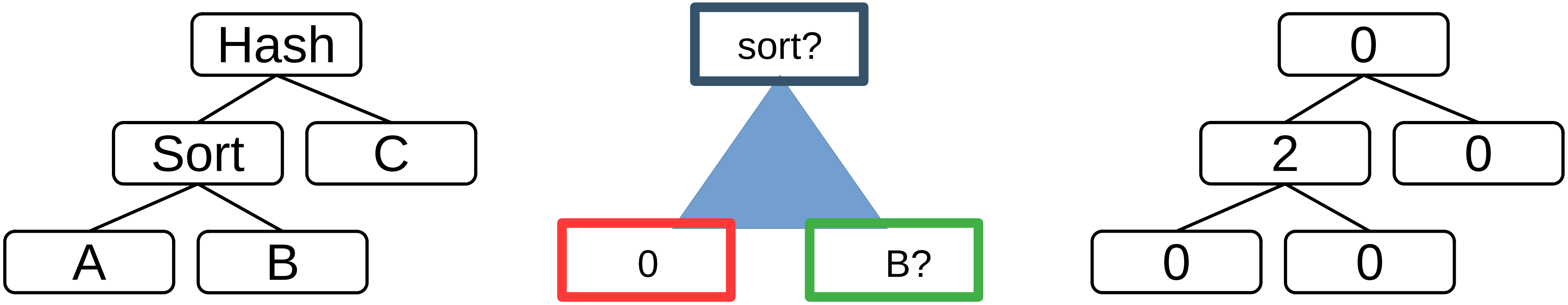
Detects a hash on top of a sort

Tree Convolution



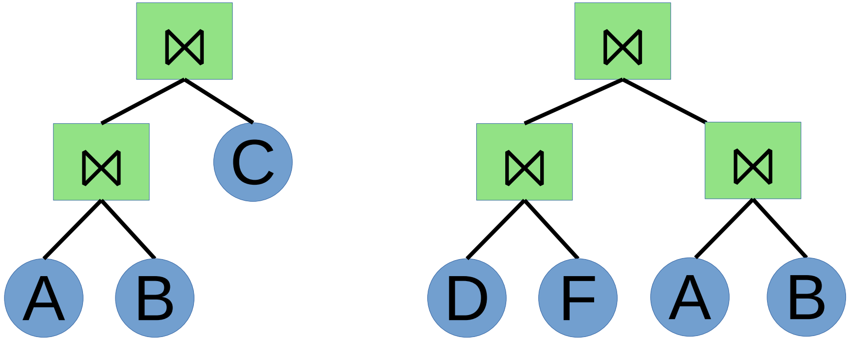
Detects a hash on top of a sort

Tree Convolution



Detects a merge join with B on the right

What makes ML *good*?



Fundamental structure is a query plan tree.

=> Tree convolution neural networks

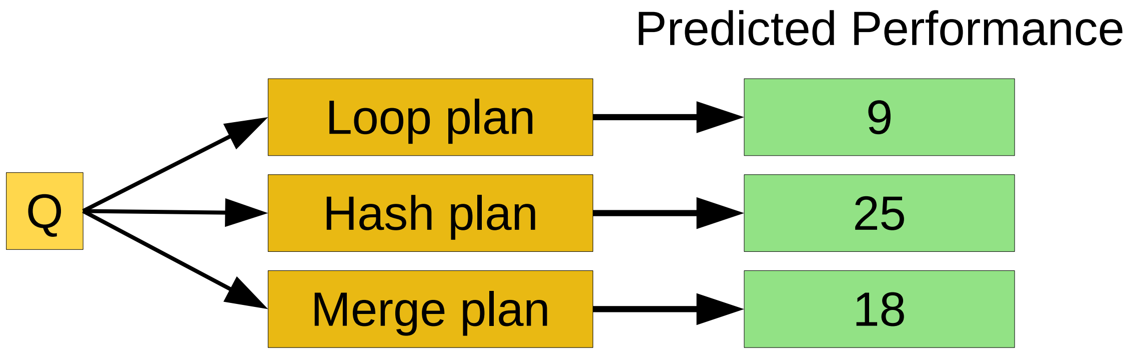
https://ryan.cab/neo
https://ryan.cab/treeconv

Training

- Option 1: Use a giant query log and train / continuously redeploy the model.
  - “Easy,” but doesn’t adapt.
- Option 2: Periodically retrain the model online using Thompson sampling.

Exploration & Exploitation

- How do we balance exploration (new policies) with exploitation (doing what we know works)?

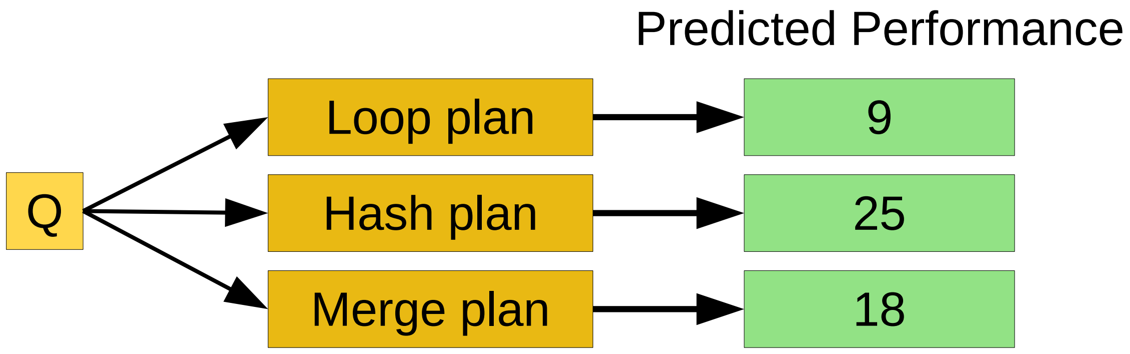


Traditional Query Optimizer

Exploration & Exploitation

- How do we balance exploration (new policies) with exploitation (doing what we know works)?

Exploitation: pick the lowest.

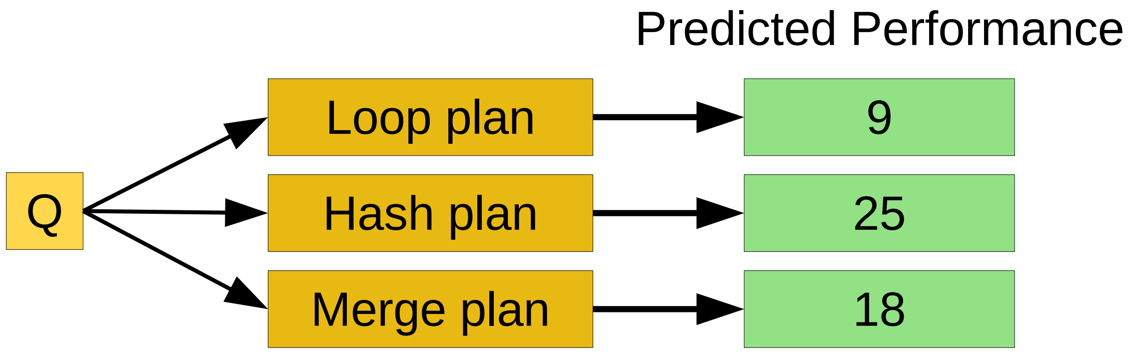


Traditional Query Optimizer

Exploration & Exploitation

- How do we balance exploration (new policies) with exploitation (doing what we know works)?

Exploitation: pick the lowest.
Exploration: choose randomly.

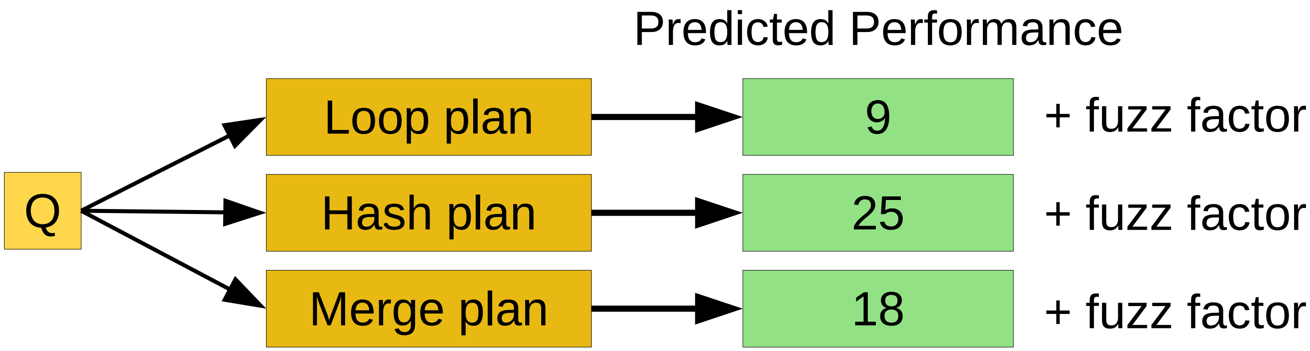


Traditional Query Optimizer

Exploration & Exploitation

- How do we balance exploration (new policies) with exploitation (doing what we know works)?

Exploitation: pick the lowest.
Exploration: choose randomly.



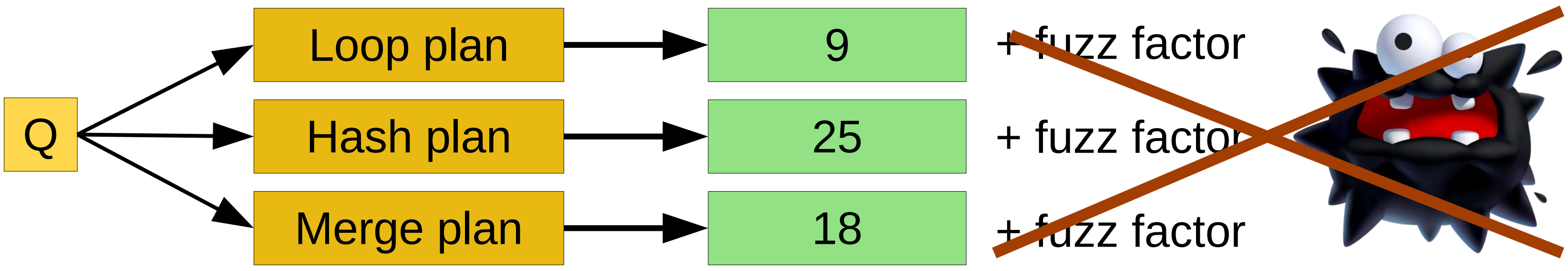
Traditional Query Optimizer

Exploration & Exploitation

- How do we balance exploration (new policies) with exploitation (doing what we know works)?

Exploitation: pick the lowest.
Exploration: choose randomly.

Predicted Performance



Traditional Query Optimizer

Exploration & Exploitation

- How do we balance exploration (new policies) with exploitation (doing what we know works)?

Exploitation: pick the lowest.
Exploration: choose randomly.

Predicted Performance

Loop plan	9	+ fuzz factor
Hash plan	25	+ fuzz factor
Merge plan	18	+ fuzz factor

Traditional Query Optimizer

Exploration & Exploitation

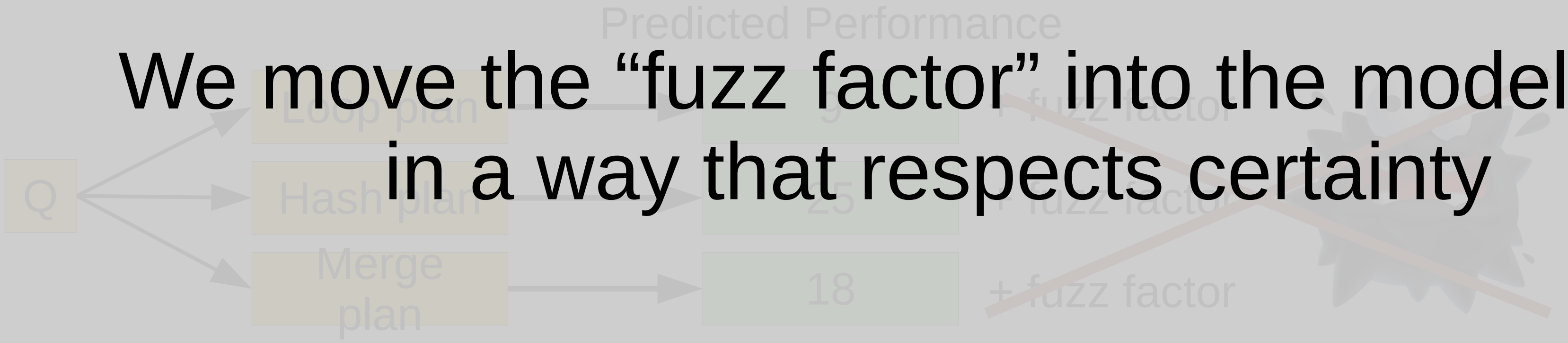
- How do we balance exploration (new policies) with exploitation (doing what we know works)?

With Thompson sampling, always pick the lowest.

BUT

We move the “fuzz factor” into the model itself in a way that respects certainty

Exploitation: pick the lowest.
Exploration: choose randomly.



Traditional Query Optimizer

Thompson Sampling

- An old, well-studied algorithm for balancing exploration and exploitation.

Thompson Sampling

- An old, well-studied algorithm for balancing exploration and exploitation.

Usual ML training
(exploitation)

model weights = E[P(model weights | data)]

Thompson Sampling

- An old, well-studied algorithm for balancing exploration and exploitation.

Usual ML training (exploitation)	model weights = E[P(model weights | data)]
Pick a random model (exploration)	model weights = sample P(model weights)

Thompson Sampling

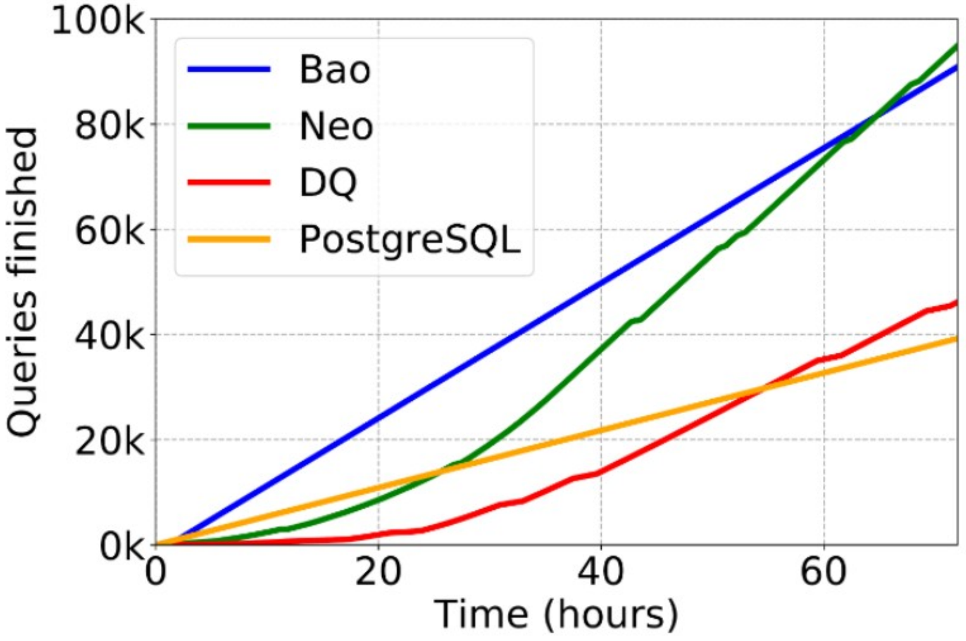
- An old, well-studied algorithm for balancing exploration and exploitation.

Usual ML training (exploitation)	model weights = E[P(model weights | data)]
Pick a random model (exploration)	model weights = sample P(model weights)
Sample model weights (Thompson Sampling)	model weights = sample P(model weights | data)

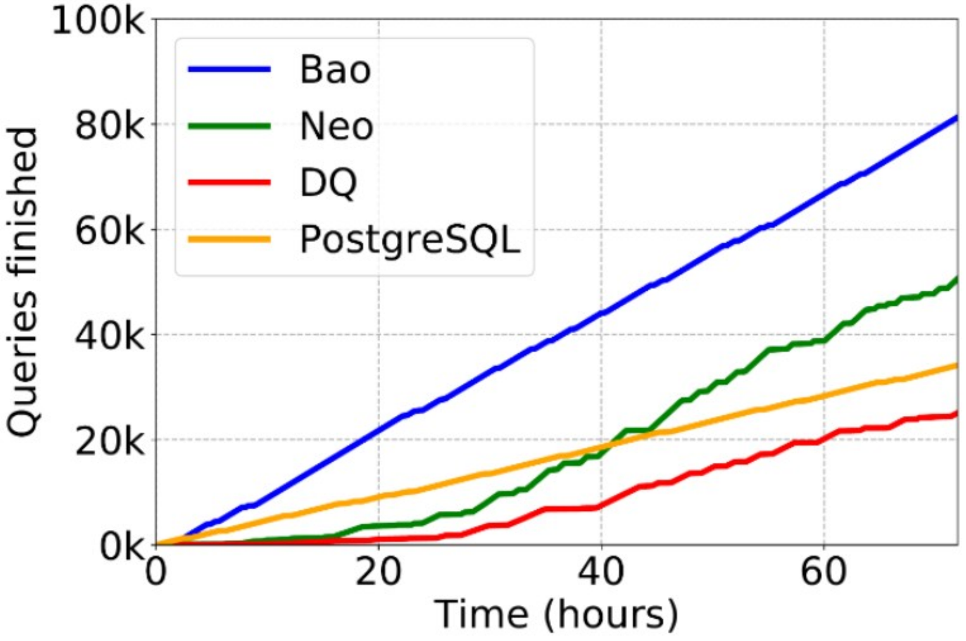
Extensibility

- New cardinality estimators can be easily added as features
  - Bao automatically incorporates them into decision making
- New optimization strategies can be added
  - Number of arms is exponential, so some care needed
  - Possibly easier than integrating into a traditional optimizer

Experimental Highlights

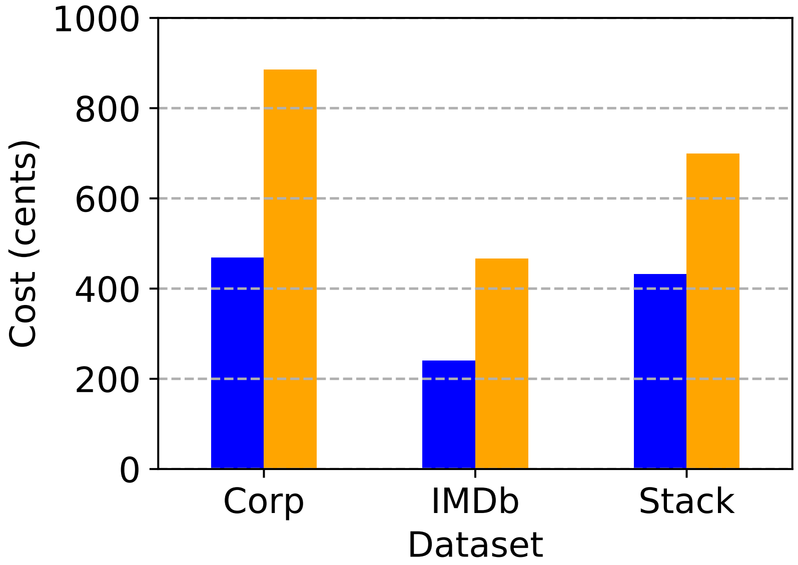


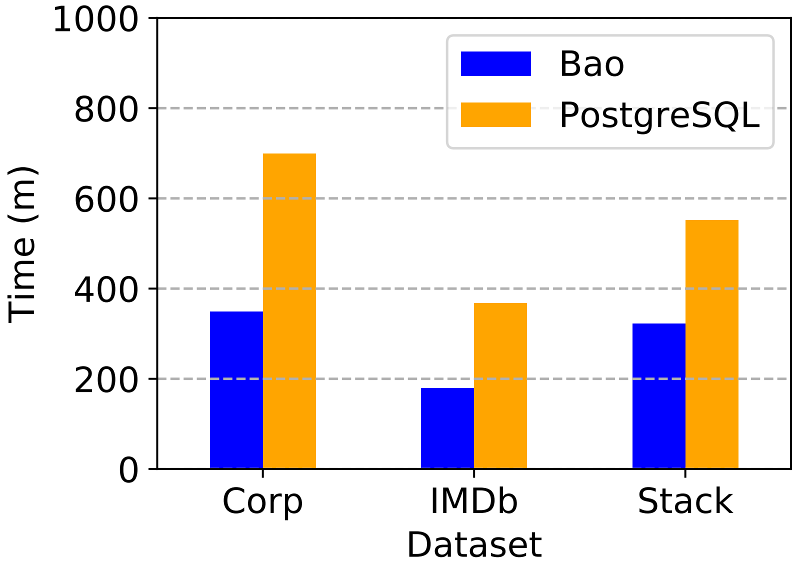
(a) Stable query workload



(b) Dynamic query workloads

Experiment Highlights



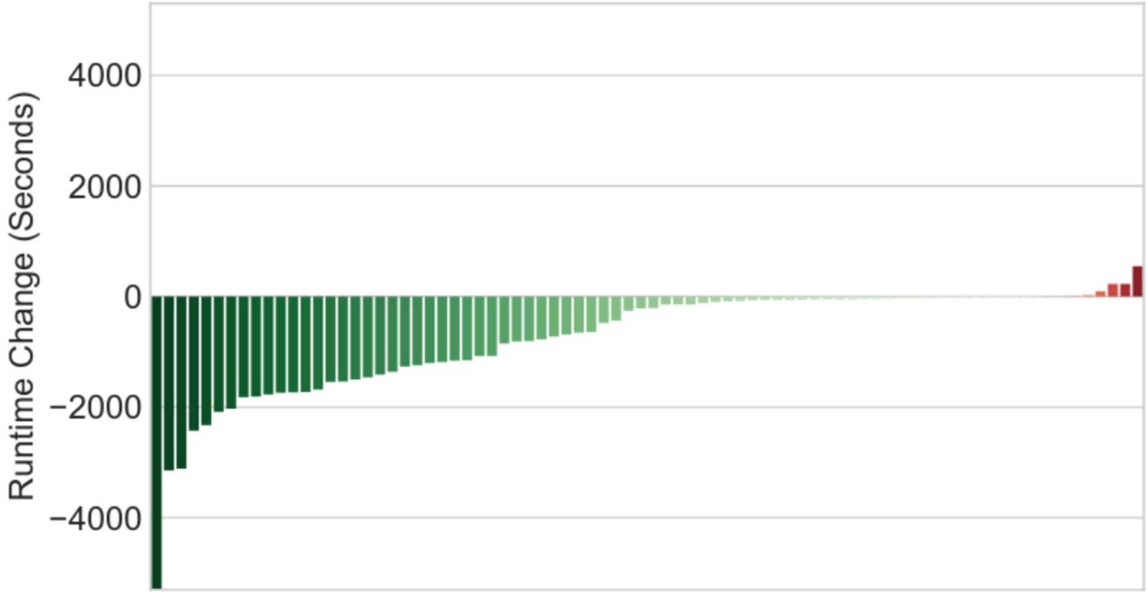


Bao handles dynamic workloads, schema, and data.

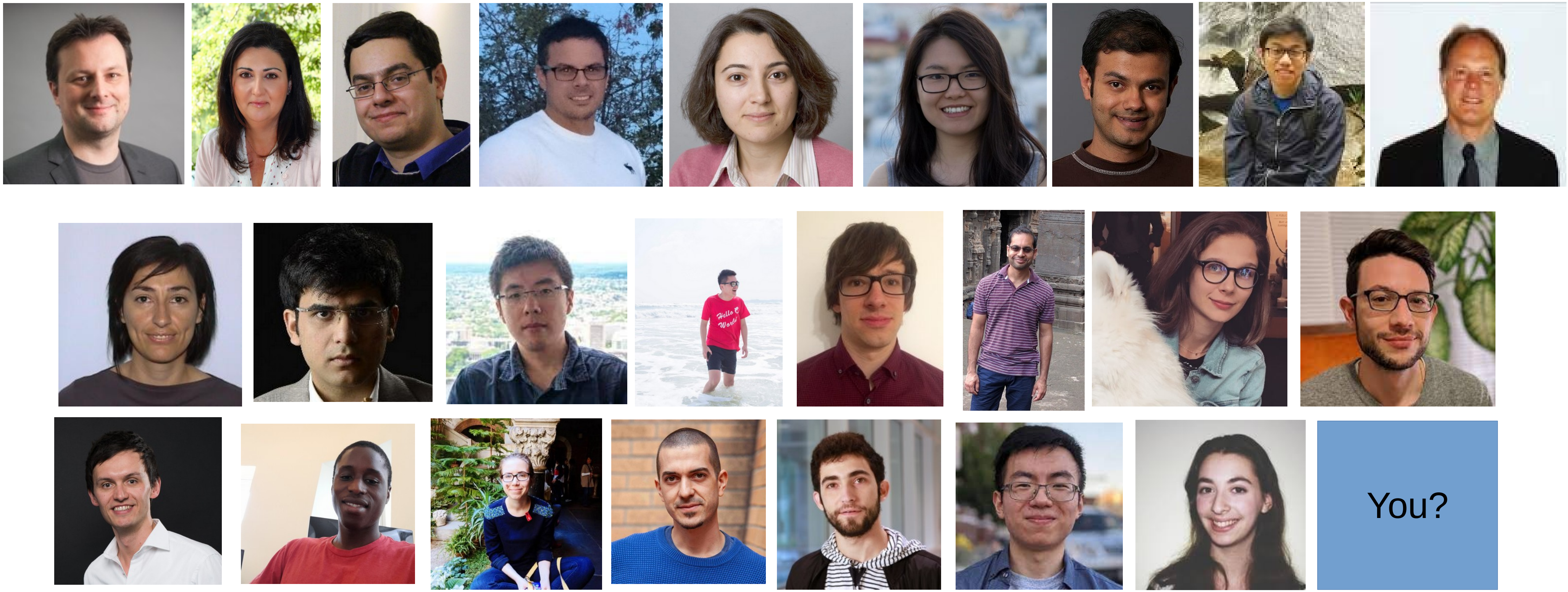
	Size	Queries	WL	Data	Schema
IMDb	7.2 GB	5000	Dynamic	Static	Static
Stack	100 GB	5000	Dynamic	Dynamic	Static
Corp	1 TB	2000	Dynamic	Static[a]	Dynamic

Microsoft SCOPE

- Bao adopted by Microsoft for SCOPE system
- 5+ PB analytic database



Collaborators



Tim Kraska, Olga Papaemmanouil, Mohammad Alizadeh, Justin Gottschlich, Nesime Tatbul, Bailu Ding, Sudipto Das, Wentao Wu, James Storer, Antonella DiLillo, Pari Negi, Hongzi Mao, Chi Zhang, Alex van Renen, Sanchit Misra, Nadia Chepurko, Emanuel Zgraggen, Andreas Kipf, Amadou Ngom, Sofiya Semenova, Raul Castro Fernandez, Solomon Garber, Jialin Ding, Anat Kleiman

Next Steps

- Ongoing collaboration: SageDB
  - Integrating many components under one roof
- Learned query “superoptimization”
  - Program synthesis and Bayes optimization
- Learned systems beyond on the RDBMS

Ryan Marcus

- Find me online: `https://ryanmarc.us`
- Twitter: @RyanMarcus
- Mastodon: RyanMarcus@discuss.systems
- Email: `rcmarcus@seas.upenn.edu`
- Actively seeking students and collaborators!
- Slides: `https://rm.cab/bu23`