Machine Learning for Query Optimization:
A Few Interesting Results
& Thousands of Practical Barriers

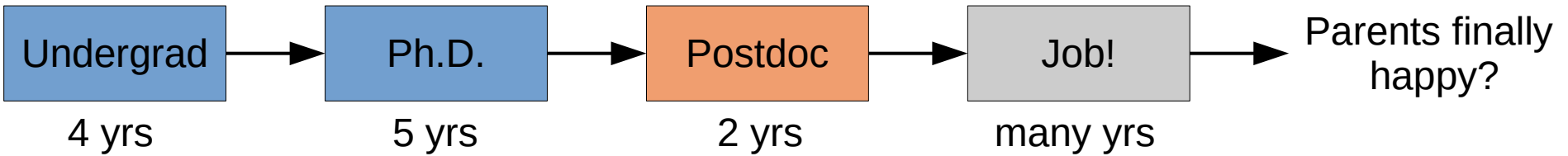
Ryan Marcus (MIT)
ryanmarcus@csail.mit.edu
Twitter: @RyanMarcus

These slides: https://ryan.cab/bu20

Machine Learning for Query Optimization:
A Few Interesting Results
& Thousands of Practical Barriers

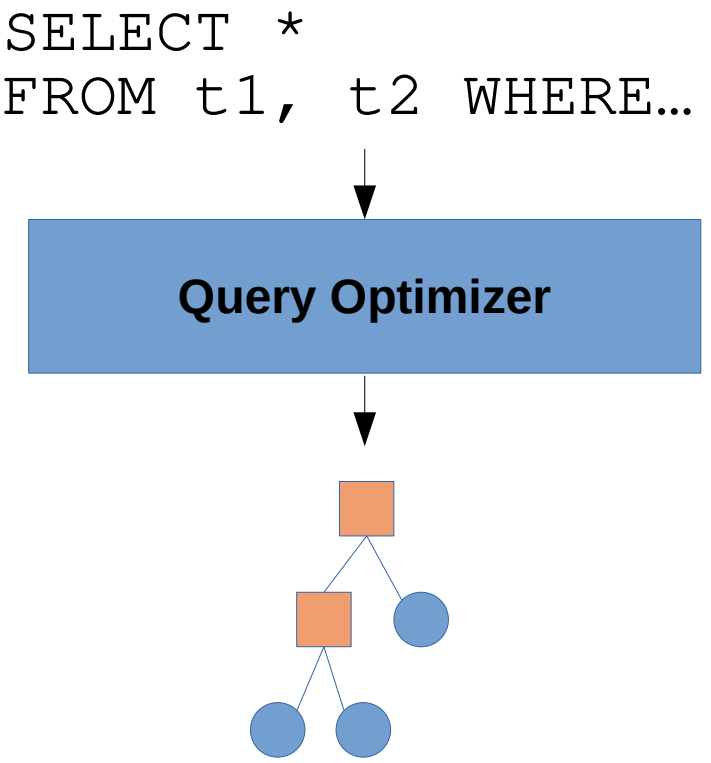
Ryan Marcus (MIT)
ryanmarcus@csail.mit.edu
Twitter: @RyanMarcus

These slides: https://ryan.cab/bu20



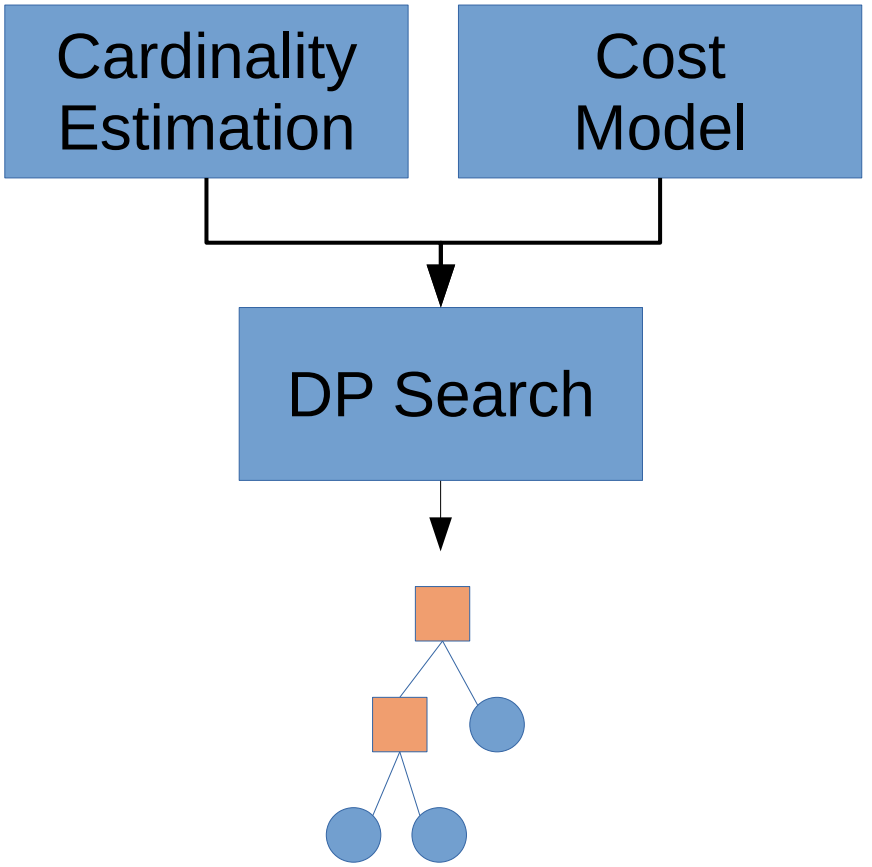
Query Optimizers

- 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



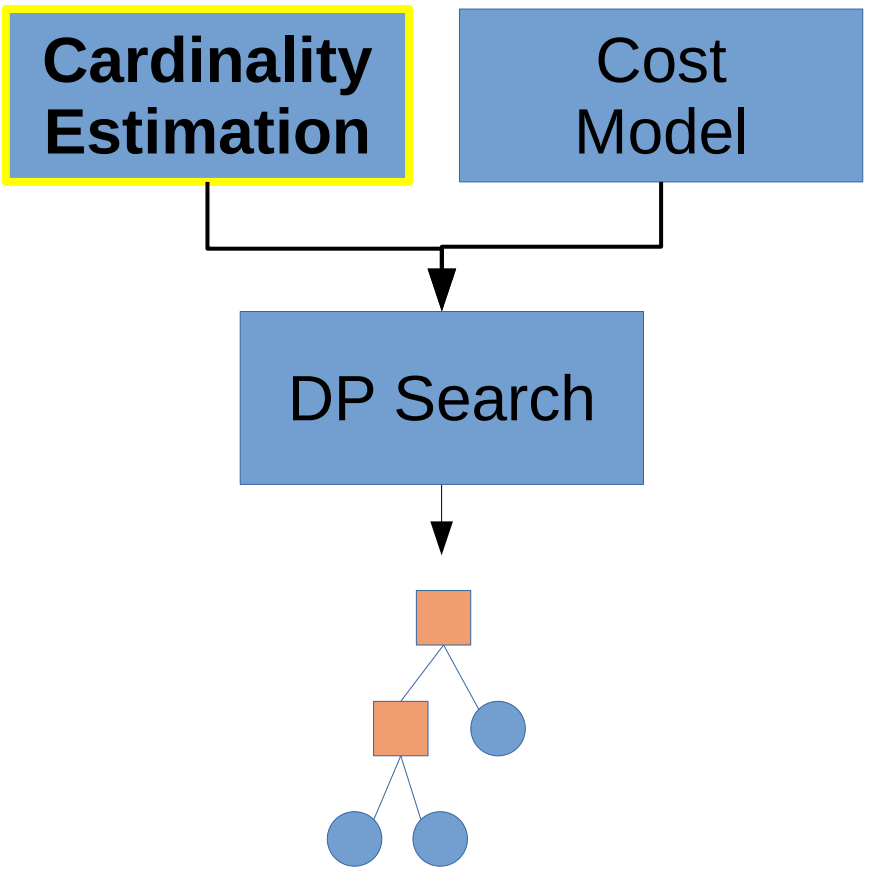
Classic Query Optimizers

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



Classic Query Optimizers

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

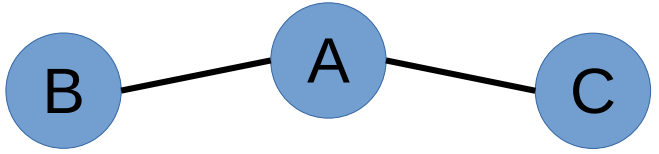


Cardinality Estimation

```
SELECT * FROM A, B, C WHERE
A.c1 = B.c1 AND A.c2 = C.c2;
```

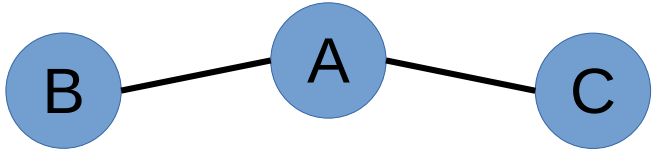
Cardinality Estimation

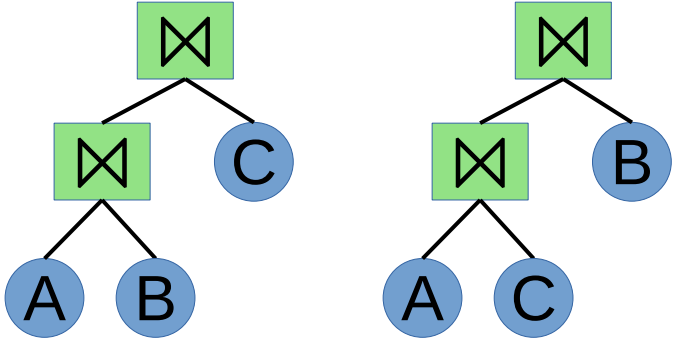
```
SELECT * FROM A, B, C WHERE
A.c1 = B.c1 AND A.c2 = C.c2;
```



Cardinality Estimation

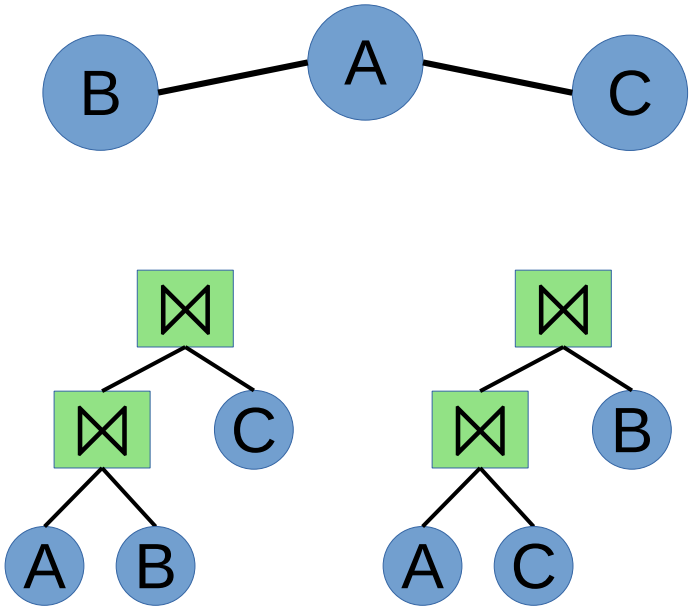
```
SELECT * FROM A, B, C WHERE
A.c1 = B.c1 AND A.c2 = C.c2;
```





Cardinality Estimation

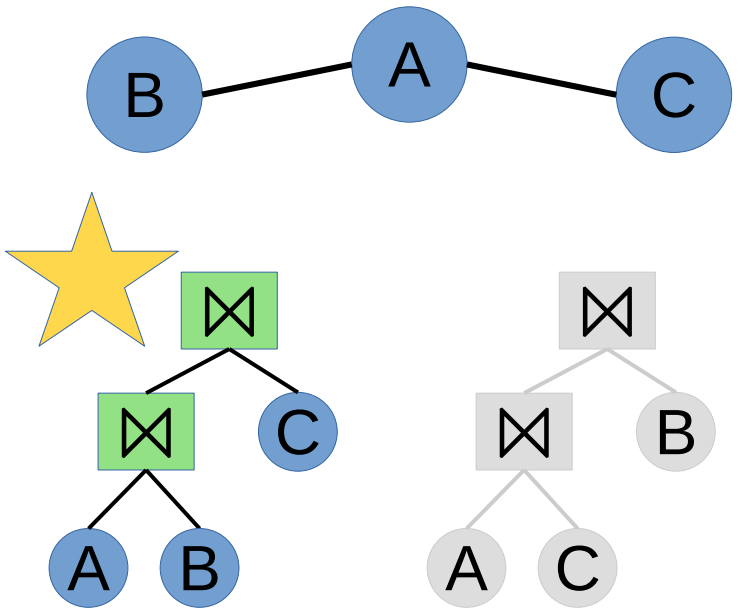
```
SELECT * FROM A, B, C WHERE
A.c1 = B.c1 AND A.c2 = C.c2;
```



Estimator	True
A ⋈ B	8
A ⋈ C	15

Cardinality Estimation

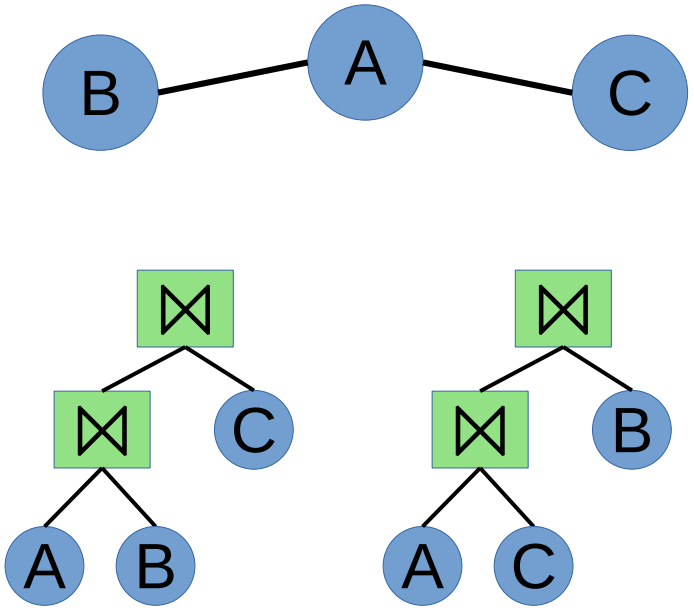
```
SELECT * FROM A, B, C WHERE
A.c1 = B.c1 AND A.c2 = C.c2;
```



Estimator	True
A ⋈ B	8
A ⋈ C	15

Cardinality Estimation

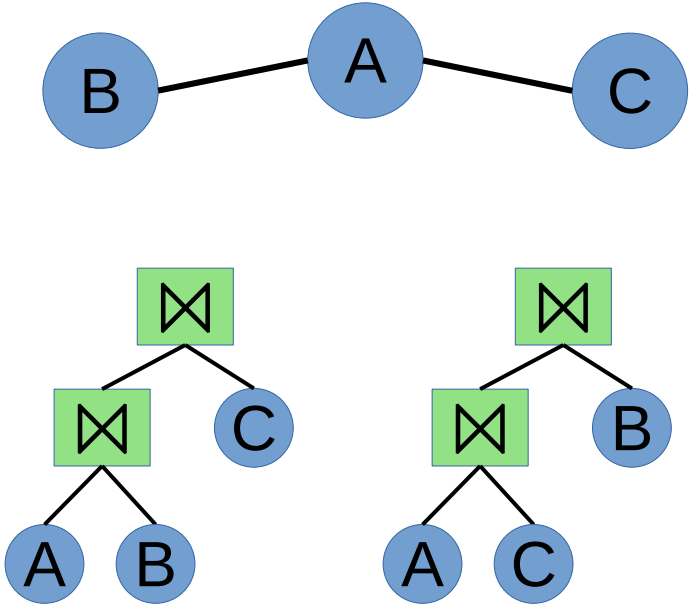
```
SELECT * FROM A, B, C WHERE
A.c1 = B.c1 AND A.c2 = C.c2;
```



Estimator	True	Est 1
A ⋈ B	8	4
A ⋈ C	15	9

Cardinality Estimation

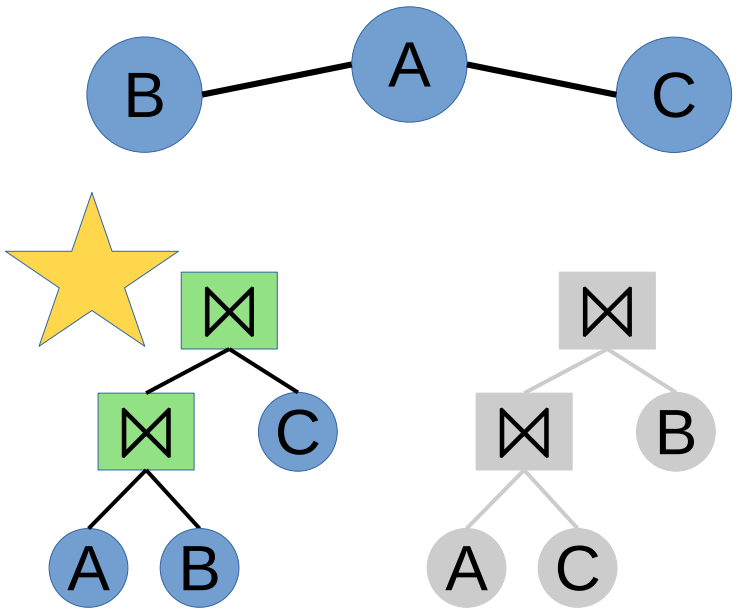
```
SELECT * FROM A, B, C WHERE
A.c1 = B.c1 AND A.c2 = C.c2;
```



Estimator	True	Est 1
A ⋈ B	8	4
A ⋈ C	15	9
L_2	0	$4^2 + 6^2$

Cardinality Estimation

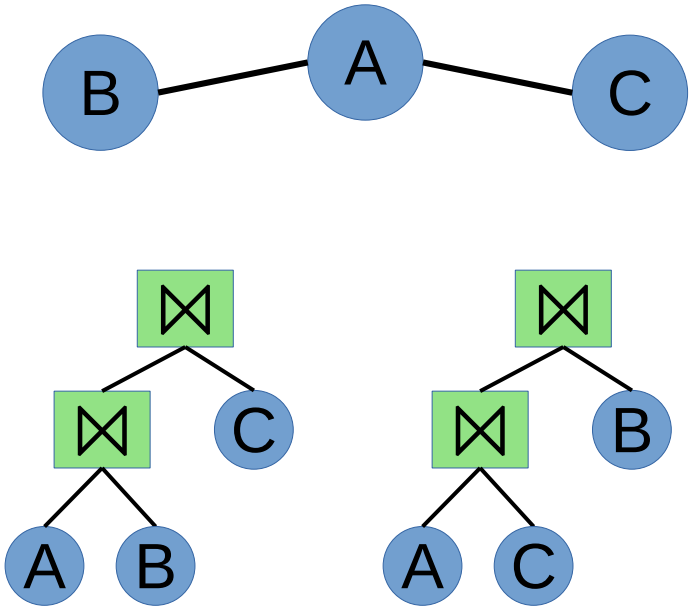
```
SELECT * FROM A, B, C WHERE
A.c1 = B.c1 AND A.c2 = C.c2;
```



Estimator	True	Est 1
A ⋈ B	8	4
A ⋈ C	15	9
L_2	0	$4^2 + 6^2$

Cardinality Estimation

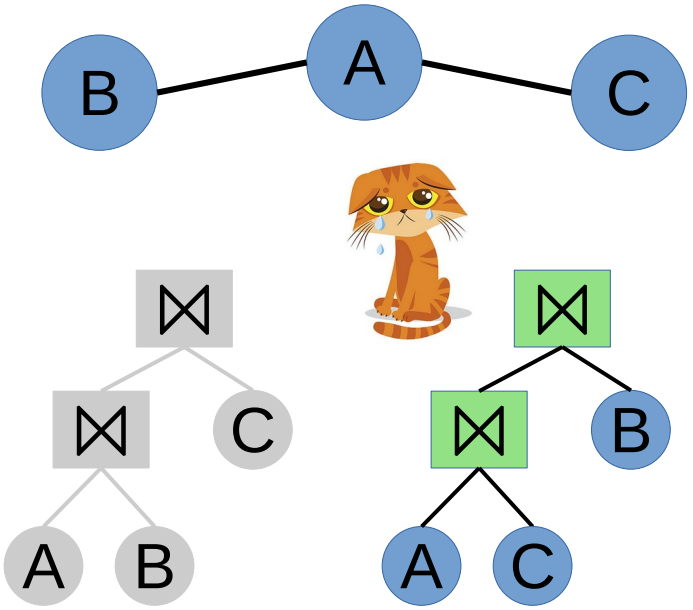
```
SELECT * FROM A, B, C WHERE
A.c1 = B.c1 AND A.c2 = C.c2;
```



Estimator	True	Est 1	Est 2
$A \bowtie B$	8	4	12
$A \bowtie C$	15	9	11
L_2	0	$4^2 + 6^2$	$4^2 + 4^2$

Cardinality Estimation

```
SELECT * FROM A, B, C WHERE
A.c1 = B.c1 AND A.c2 = C.c2;
```

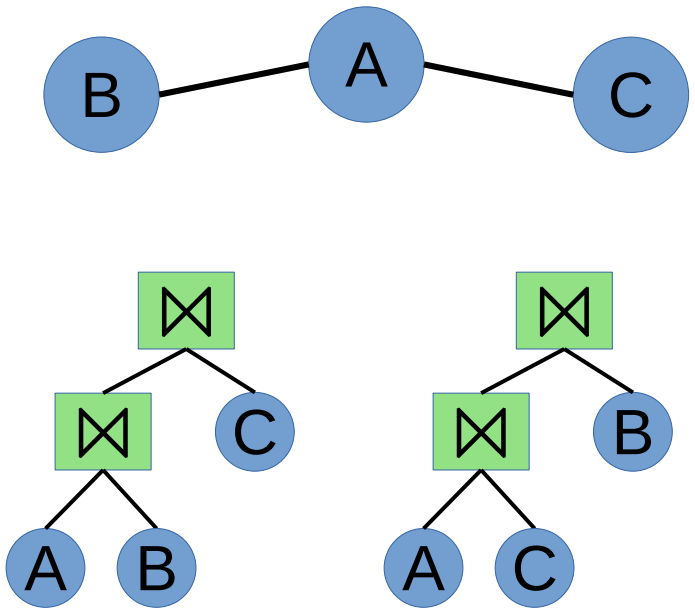


Estimator	True	Est 1	Est 2
A ⋈ B	8	4	12
A ⋈ C	15	9	11
L_2	0	$4^2 + 6^2$	$4^2 + 4^2$

Cardinality Estimation

```
SELECT * FROM A, B, C WHERE
A.c1 = B.c1 AND A.c2 = C.c2;
```

Higher error, better plan!

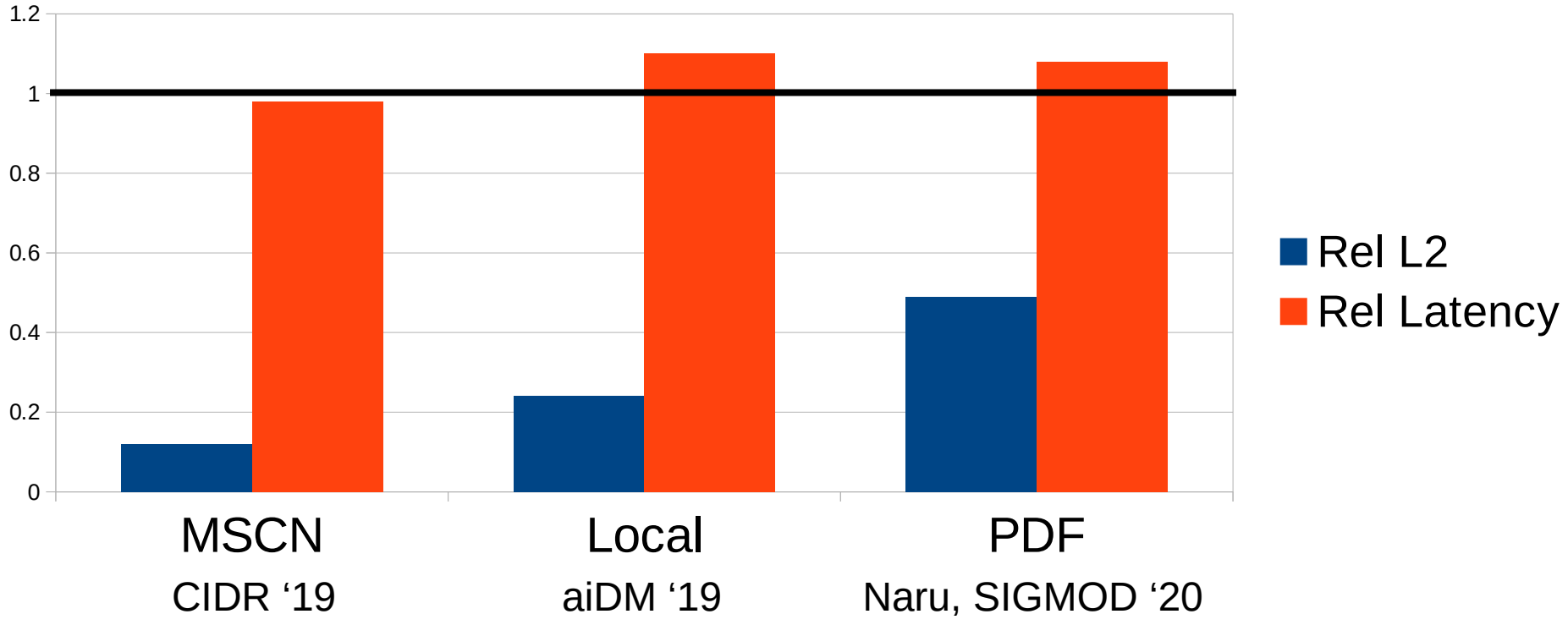


Estimator	True	Est 1	Est 2
A ⋈ B	8	4	12
A ⋈ C	15	9	11
L_2	0	$4^2 + 6^2$	$4^2 + 4^2$

Learning QO?

- Cardinality estimation
  - Can we use ML to predict the cardinalities?
  - Yes, but limited impact on QO.

Cardinality Estimation



Learning QO?

- Cardinality estimation
  - Can we use ML to predict the cardinalities?
  - Yes, but limited impact on QO.
- End-to-end QO
  - Possible, but harder.
  - Requires some background.

DB + ML

“but there’s no worst-case bound!”
- the DB community

“can’t hear you over
all my grant money”
- the ML community

The Deep Learning Mythos

Fear, Uncertainty, Doubt.
It’s a black box.

Magic
Crazy results from Arch-Mage Goodfellow

Deep Learning for QO

- Recent groundswell of research
- RL based approaches:
  - Feb/Mar ‘18:
    - Marcus et al. (ReJOIN)
    - Ortiz et al.
  - Aug ‘18:
    - Krishnan et al. (DQ)
  - Aug ‘19:
    - Neo, VLDB
- Cardinality based approaches:
  - 2015: Liu et al.
  - 2019: Kipf et al. (MSCN)
  - Aug ‘19:
    - Group by (Kipf et al.)
    - Local models (Woltmann et al.)
    - Shared clouds (Wu et al.)

Deep Learning for QO

- *Many* of these works represent a “just add deep learning and stir” approach.
- Characterized by:
  - Fully-connected neural networks
  - Train / test set leakage
  - Off-the-shelf RL or regression loss functions
  - Little to no integration with a broader DB
  - No evaluation of *actual query performance*
- Examples: ReJOIN*****, MSCN, Learned State Representations, operator embeddings*****, DQ, CardNet, and probably many more...

* my work, and I’ll be the first to admit that I drank the Kool-Aid.

Deep Learning for QO

- *Many* of these works represent a “just add deep learning and stir” approach.
- Characterized by:
  - Fully-connected neural networks
  - Train / test set leakage
  - Off-the-shelf RL or regression loss functions
  - Little to no integration with a broader DB
  - No evaluation of *actual query performance*

} The 5 deadly sins of ML for DB

- Examples: ReJOIN*****, MSCN, Learned State Representations, operator embeddings*****, DQ, CardNet, and probably many more...

* my work, and I’ll be the first to admit that I drank the Kool-Aid.

Deep Learning for QO

- *Many* of these works represent a “just add deep learning and stir” approach.
- Characterized by:
  - **Fully-connected neural networks**
  - Train / test set leakage
  - Off-the-shelf RL or regression loss functions
  - Little to no integration with a broader DB
  - No evaluation of *actual query performance*

The 5 deadly sins of ML for DB

- Examples: ReJOIN*****, MSCN, Learned State Representations, operator embeddings*****, DQ, CardNet, and probably many more...

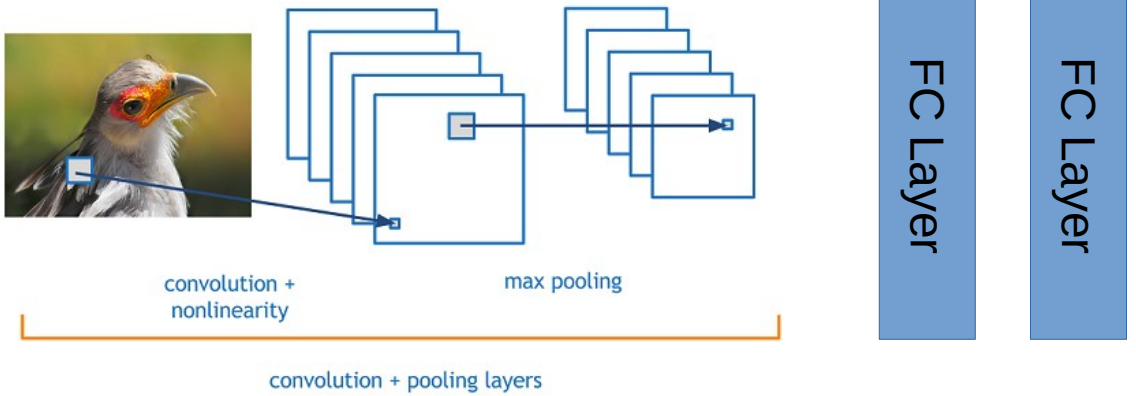
* my work, and I’ll be the first to admit that I drank the Kool-Aid.

What makes DL *good?*

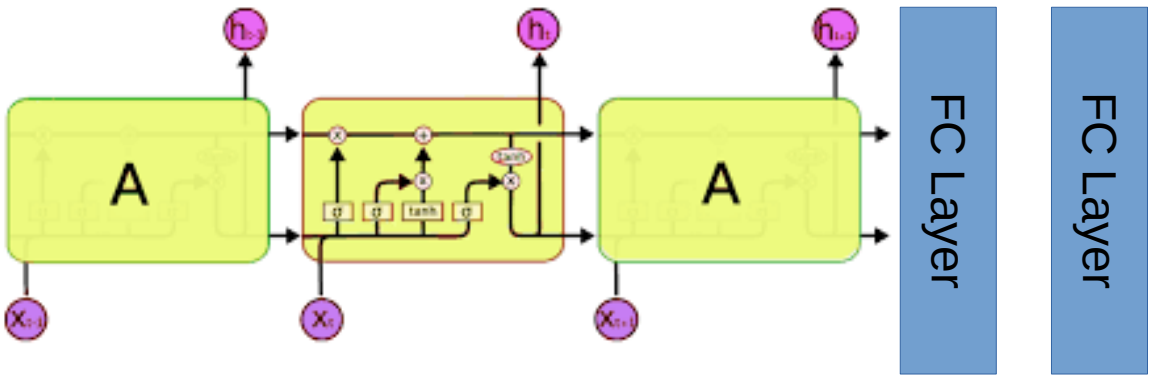
- Biggest DL success stories:
  - Computer vision (CV)
  - Natural language processing (NLP)

What makes DL *good?*

Convolution Neural Networks (CNNs)



Long short-term Memory Networks (LSTM)



What makes DL *good?*

- To build a NN to *recognize objects in images*, we modeled the low-level structure used by *the human eye*.
  - LeCun et al., and indirectly, a Turing award
- To build a NN to *recognize words in speech,* we modeled the low-level structure used by *the human prefrontal cortex*.
  - Graves et al.

- To build a NN to ***{perform a task}***, we modeled the low-level structure used by ***{expert system known to perform well}***.

What makes DL *good?*

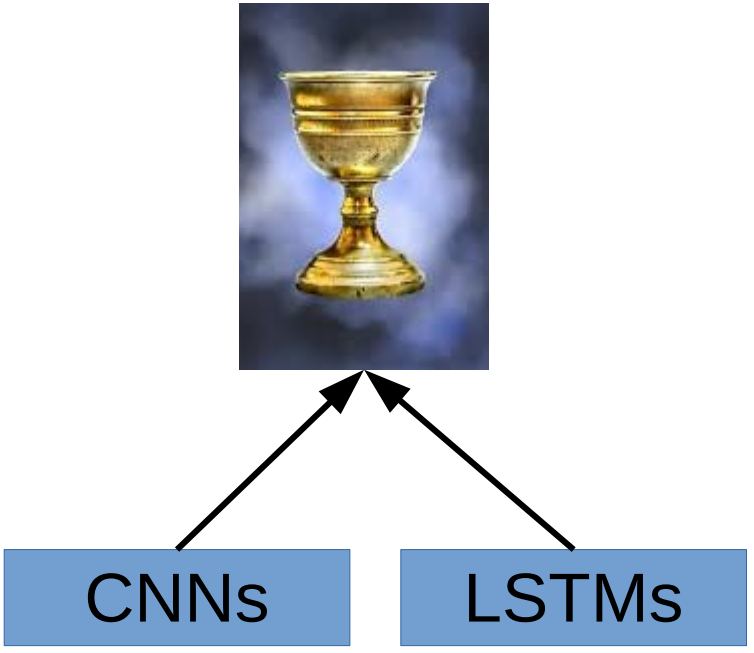
- To build a NN to *recognize objects in images*, we modeled the low-level structure used by *the human eye*.
  - LeCun et al., and indirectly, a Turing award
- To build a NN to *recognize words in speech,* we modeled the low-level structure used by *the human prefrontal cortex*.
  - Graves et al.

- To build a NN to ***{perform a task}***, we modeled the low-level structure used by ***{expert system known to perform well}***.

INDUCTIVE BIAS

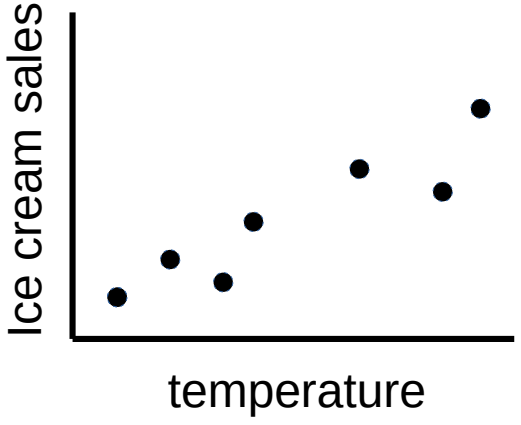
What makes DL *good?*

- As systems researchers, we *hate* complex, problem-specific solutions
  - We *love* throwing out complexity and generalizing.
- Our first response when we see all these different architectures?
  - Can’t we be more general?



What makes DL *good?*

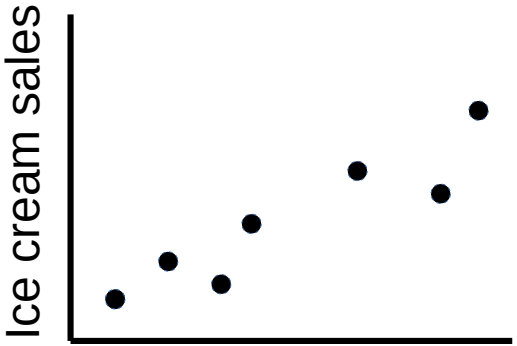
- “But wait! Hornik et al. showed that fully-connected layers can represent *any arbitrary function*!” - Someone not good at deep learning.



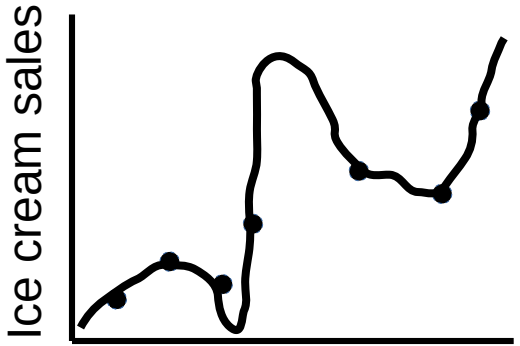
Original Data

What makes DL *good?*

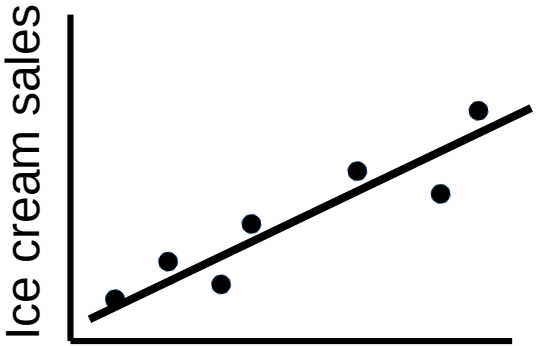
- “But wait! Hornik et al. showed that fully-connected layers can represent *any arbitrary function*!” - Someone not good at deep learning.



Original Data



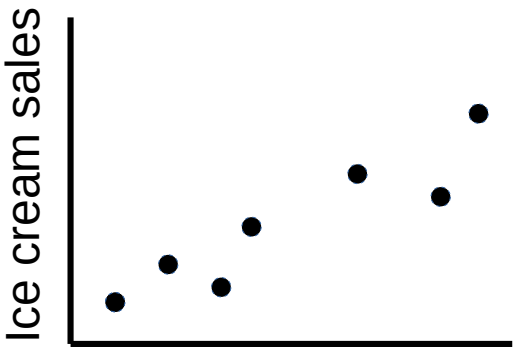
Arbitrary Function A



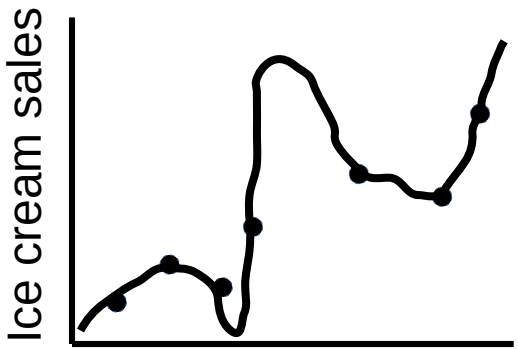
Arbitrary Function B

What makes DL *good?*

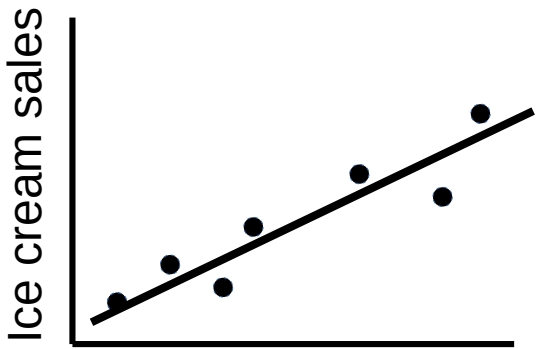
- You don’t want an *arbitrary* function that fits.
- You want a *generalizable* function that fits.



Original Data



Arbitrary Function A



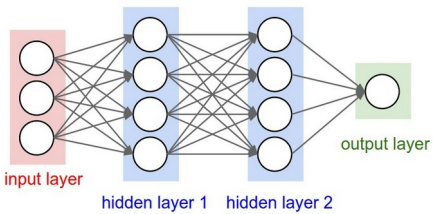
Arbitrary Function B

What makes DL *good?*

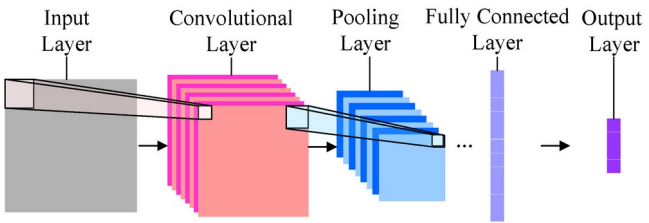
- Old school ML: regularization. Deep learning: just SGD!
- How do we know if our function will generalize?
- Inductive bias
  - One way: model it after a (biological) expert system that generalizes.
  - Generally: *constrain* the type of function that can be learned based on *prior knowledge* of the problem at hand.

Inductive Bias

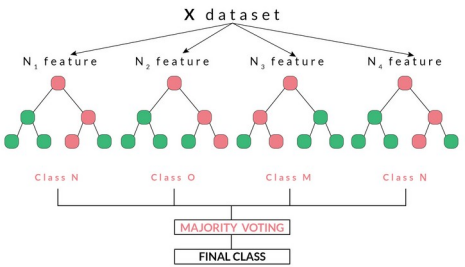
- Rank MNIST performance of these algorithms.



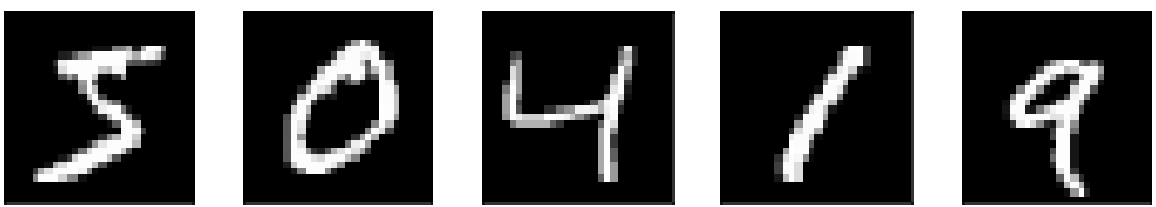
Fully-connected NN



Convolution NN



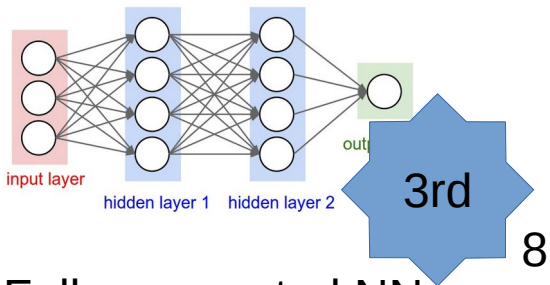
Random Forest



Fixed parameter budget ~5k, ReLU, large hyperparameter grid search

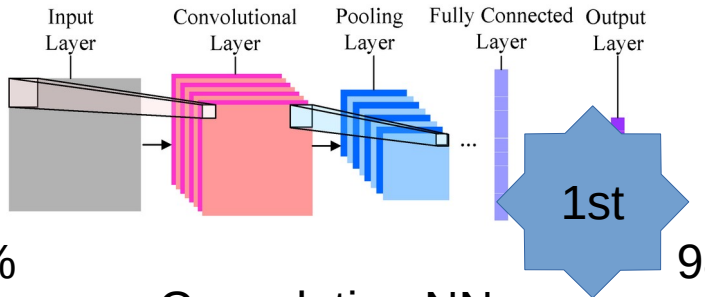
Inductive Bias

- Rank MNIST performance of these algorithms.



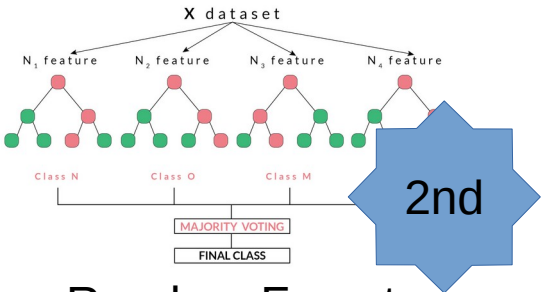
86%

Fully-connected NN



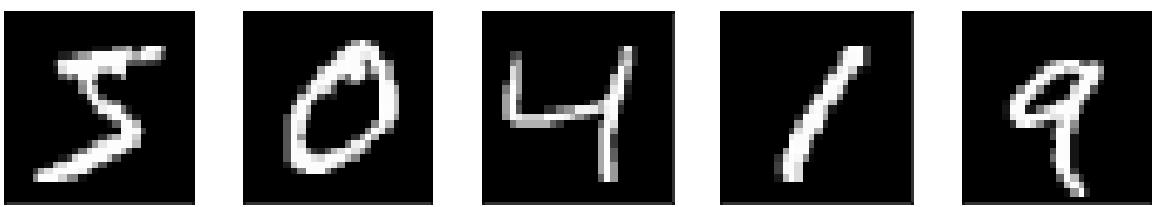
98%

Convolution NN



Random Forest

92%



Fixed parameter budget ~5k, ReLU, large hyperparameter grid search

Inductive Bias

- Conv nets have a *strong inductive bias* borrowed from biological visual systems.
- Random forests have a bias towards conditional sparsity – given one pixel, it's neighbor likely isn't that telling.
- Fully connected NNs assign every feature a weight no matter what – biased towards sensitive/simple functions
  - … sort of. For details see Valle-Perez et al., https://arxiv.org/abs/1805.08522
- **Deep learning is *fantastic* if and only if the inductive bias of the model matches reality.**

Inductive Bias

- Deep learning hype
  - End-to-end
  - Automatic feature engineering
  - Great generalization
- Deep learning reality:
  - All these things.
  - **BUT** you have to get the model architecture right!

Inductive Bias

- What about database systems?
  - Convolutional neural networks are to computer vision as _____ is to database systems?
- Obviously, I don’t have the answer.
- But here’s an idea: tree convolution!

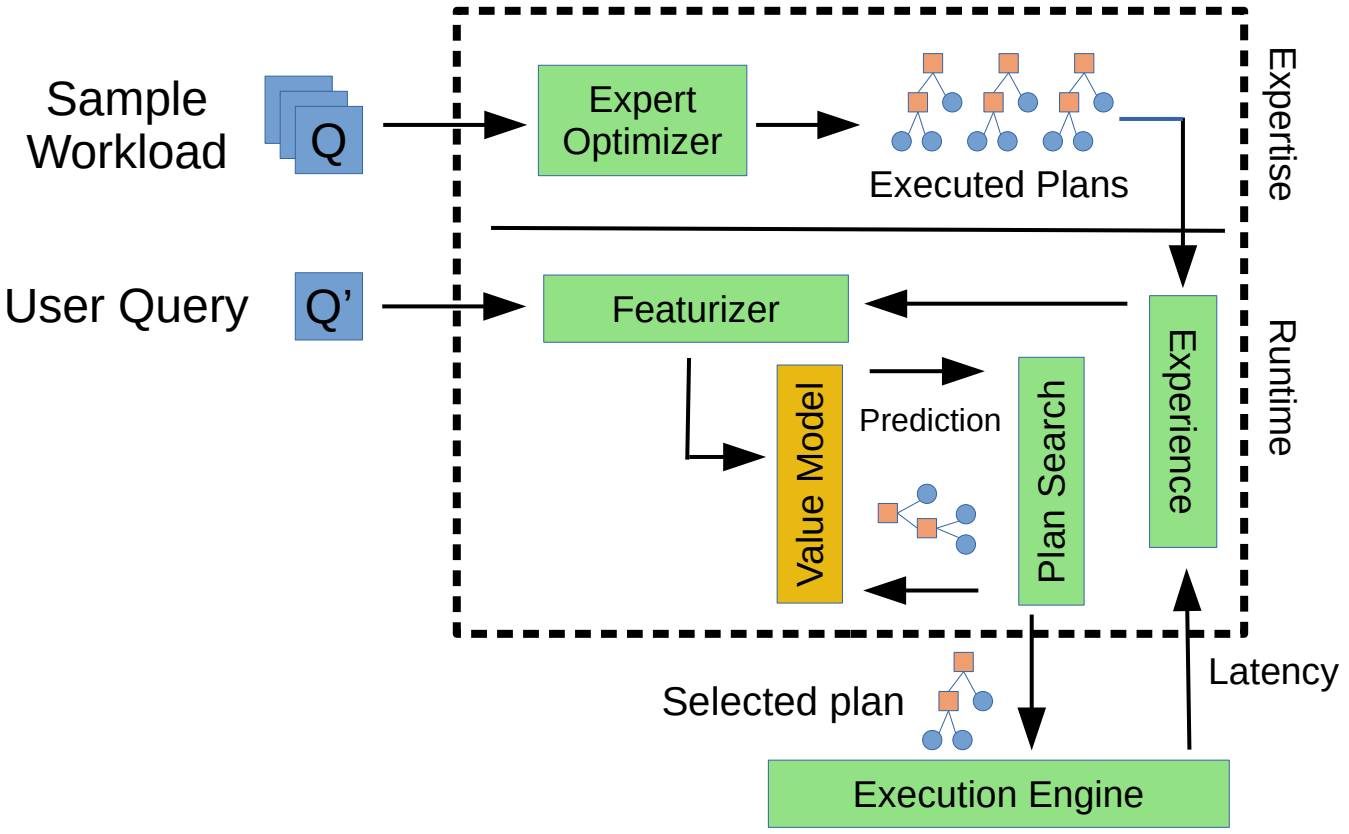
Neo: A Learned Query Optimizer

- Joint work
  - Parimarjan Negi, Hongzi Mao, Chi Zhang, Mohammad Alizadeh, Tim Kraska, Olga Papaemmanouil, Nesime Tatbul.
- An early prototype of what a deep learning powered optimizer might look like

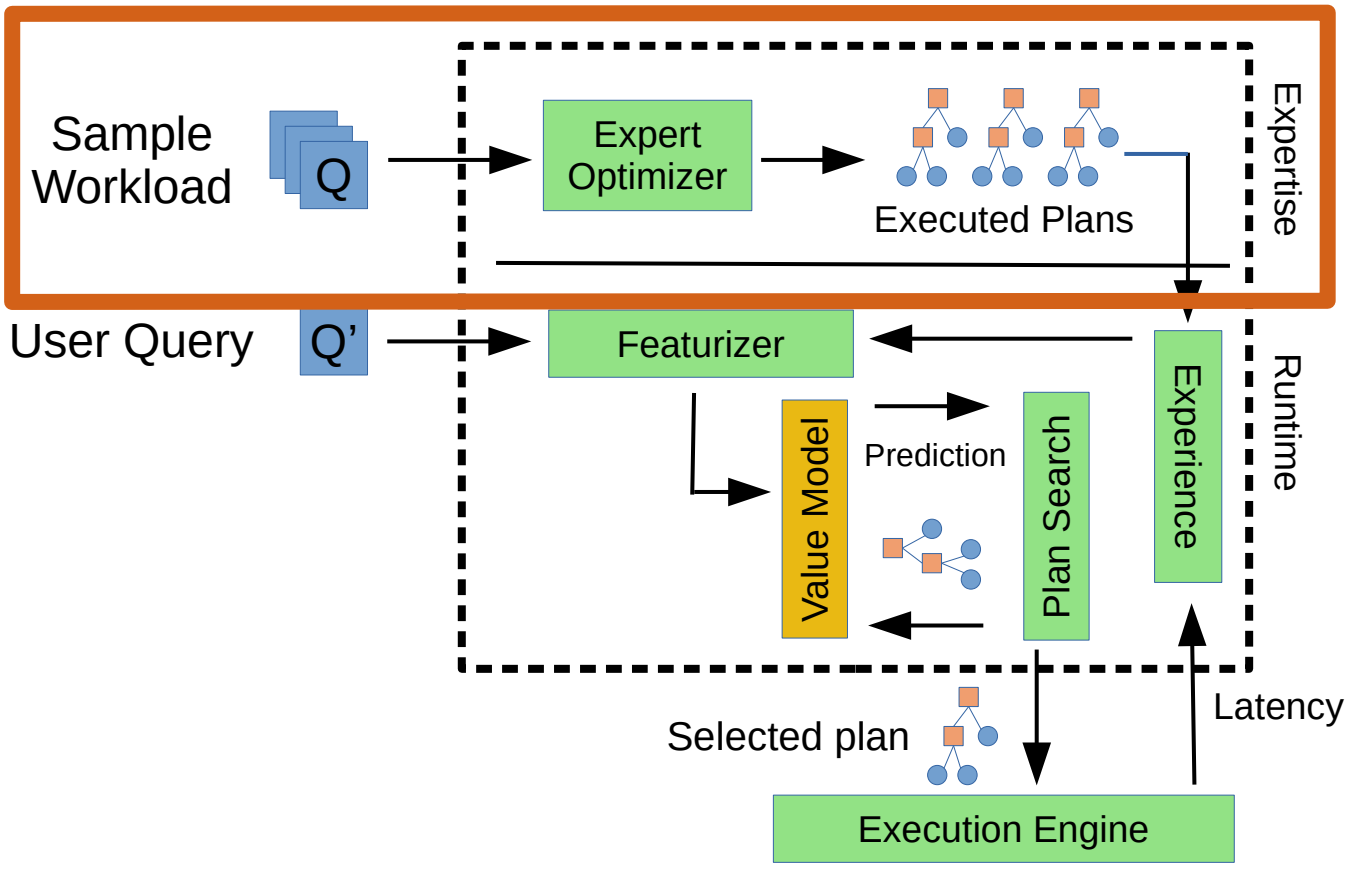
Neo

- No cost models, cardinality estimation or exponential search.
  - Previous: can replace each with a learned system *in isolation*
    - Unclear benefit on query latency
  - Neo is first to show we can have *all learned everything*.
    - Optimizing query latency directly, end-to-end
- Automatic per-DB tuning
  - Adaption to the user’s workflow and data
- Headline result: matches or exceeds the performance of SOTA query optimizers within 24hrs of training.

Neo

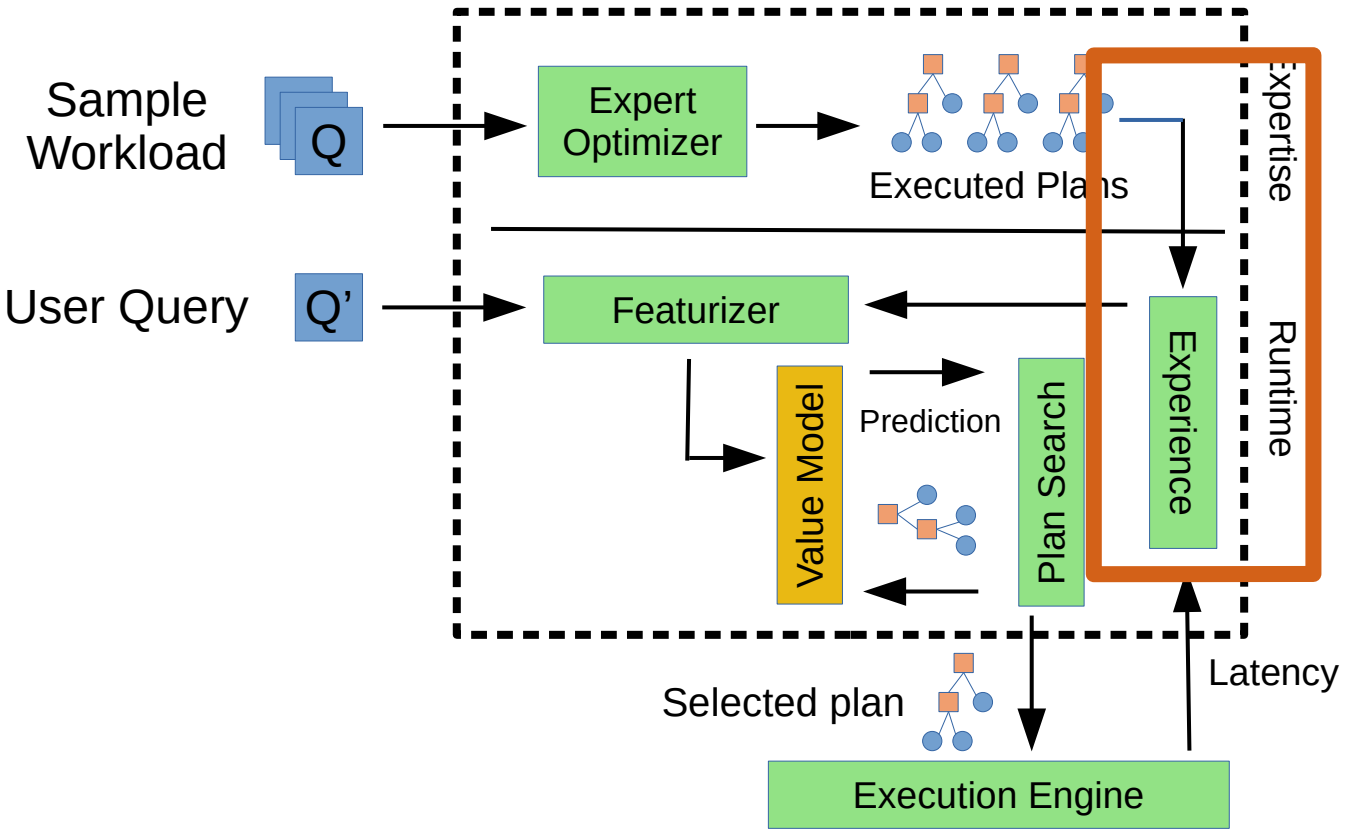

Sample Workload
Q
Expert Optimizer
Executed Plans
Expertise
User Query
Q'
Featurizer
Value Model
Prediction
Plan Search
Experience
Runtime
Selected plan
Latency
Execution Engine

Neo



1. We observe how a basic query optimizer handles a sample workload.

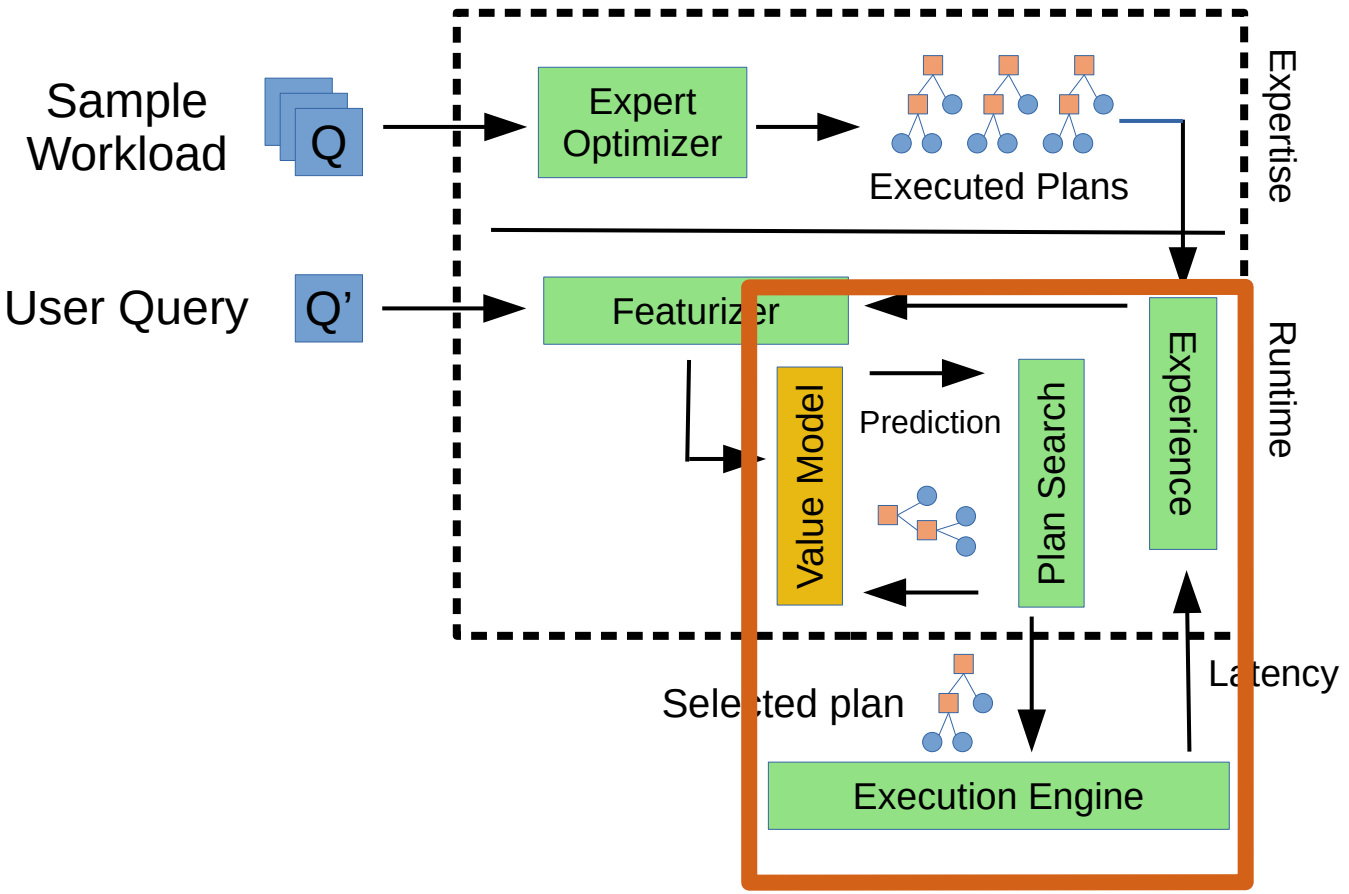
Neo



1. We observe how a basic query optimizer handles a sample workload.

2. We train a combination of a *value model* and a *plan search* module to emulate the expert.

Neo



1. We observe how a basic query optimizer handles a sample workload.

2. We train a combination of a *value model* and a *plan search* module to emulate the expert.

3. We create a feedback loop where we refine the value model using real query latency on user-submitted queries.

Neo

- First, how to represent QO as an RL problem? (MDP)
- Neo is designed around three principles:
  - Find the right inductive bias
    - Fully-connected neural networks? Never heard of ‘em.
  - Learning from demonstration
    - Watch masters. Emulate masters. Surpass masters.
  - Learn embeddings
    - No histograms, no exception lists. Learned models.

Just an overview today.

See paper for details. ☺

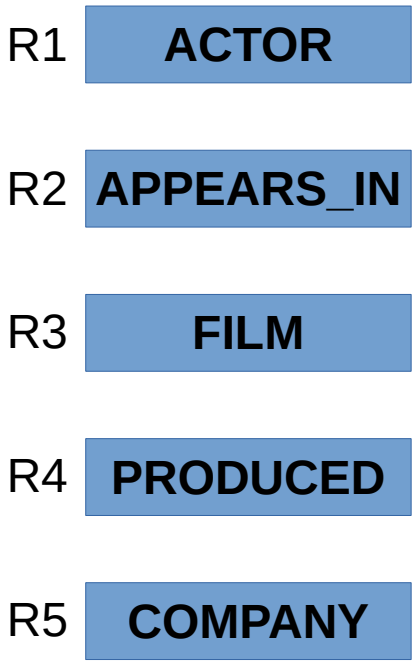
Query Optimization as an MDP

- DB assumptions
  - Binary query plan trees
  - Non-distributed
  - Fixed # join operators
  - Equi-joins only

Query Optimization as an MDP

Actions available:

1. (R1, R2) Hash
2. (R1, R2) Sort
3. (R2, R3) Hash
4. (R2, R3) Sort
5. (R3, R4) Hash
6. (R3, R4) Sort
7. (R4, R5) Hash
8. (R4, R5) Sort



(ACTORS ⋈ APPEARS_IN ⋈ FILM ⋈ PRODUCED ⋈ COMPANY)

Query Optimization as an MDP

Actions available:

1. **(R1, R2) Hash**
2. (R1, R2) Sort
3. (R2, R3) Hash
4. (R2, R3) Sort
5. (R3, R4) Hash
6. (R3, R4) Sort
7. (R4, R5) Hash
8. (R4, R5) Sort

R1 **ACTOR**

R2 **APPEARS_IN**

R3 **FILM**

R4 **PRODUCED**

R5 **COMPANY**

(ACTORS ⋈ APPEARS_IN ⋈ FILM ⋈ PRODUCED ⋈ COMPANY)

Query Optimization as an MDP

Actions available:

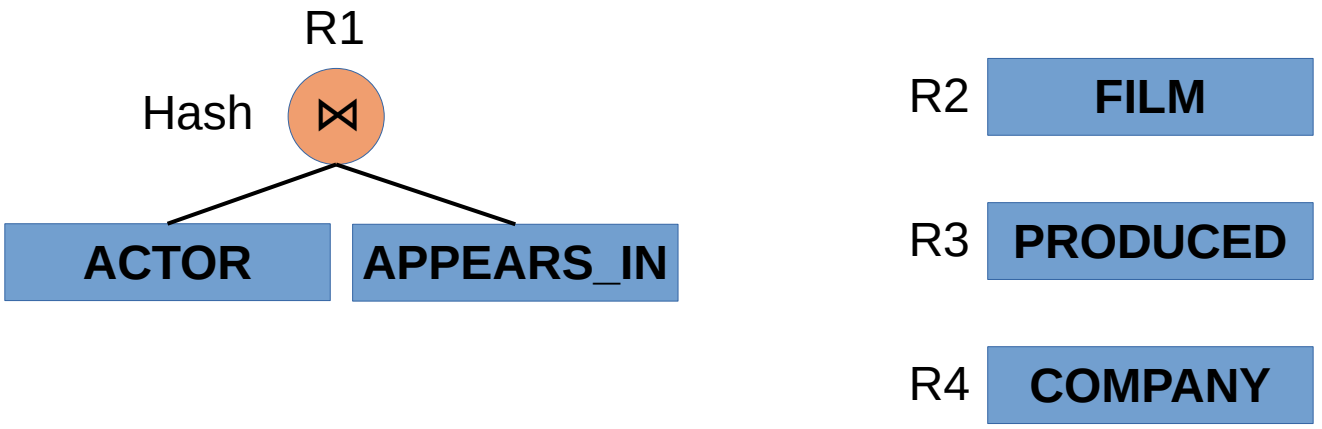
1. **(R1, R2) Hash**
2. (R1, R2) Sort
3. (R2, R3) Hash
4. (R2, R3) Sort
5. (R3, R4) Hash
6. (R3, R4) Sort
7. (R4, R5) Hash
8. (R4, R5) Sort

(ACTORS ⋈ APPEARS_IN ⋈ FILM ⋈ PRODUCED ⋈ COMPANY)

Query Optimization as an MDP

Actions available:

1. (R1, R2) Hash
2. (R1, R2) Sort
3. (R2, R3) Hash
4. (R2, R3) Sort
5. (R3, R4) Hash
6. (R3, R4) Sort

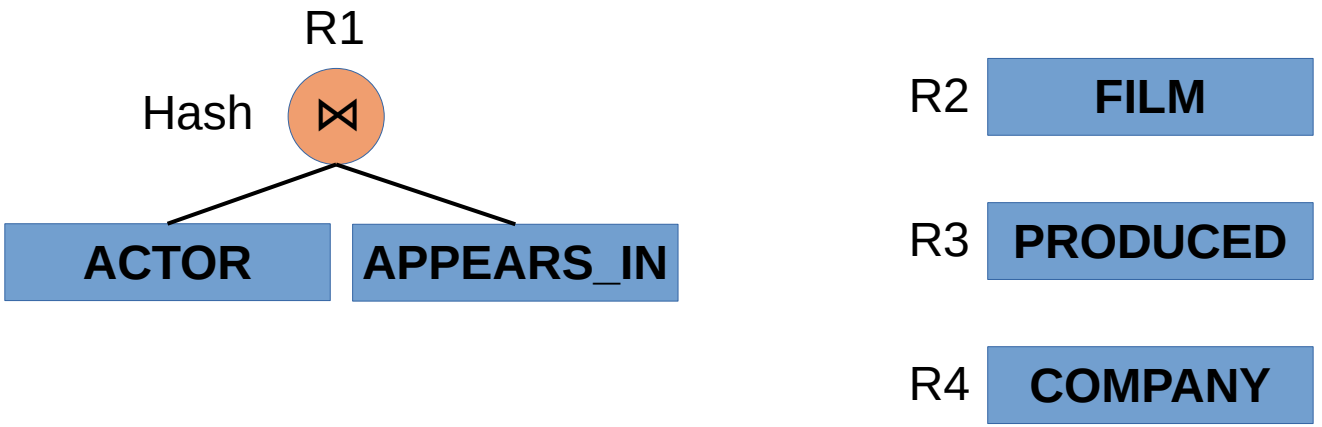


(ACTORS ⋈ APPEARS_IN ⋈ FILM ⋈ PRODUCED ⋈ COMPANY)

Query Optimization as an MDP

Actions available:

1. (R1, R2) Hash
2. (R1, R2) Sort
3. (R2, R3) Hash
4. **(R2, R3) Sort**
5. (R3, R4) Hash
6. (R3, R4) Sort

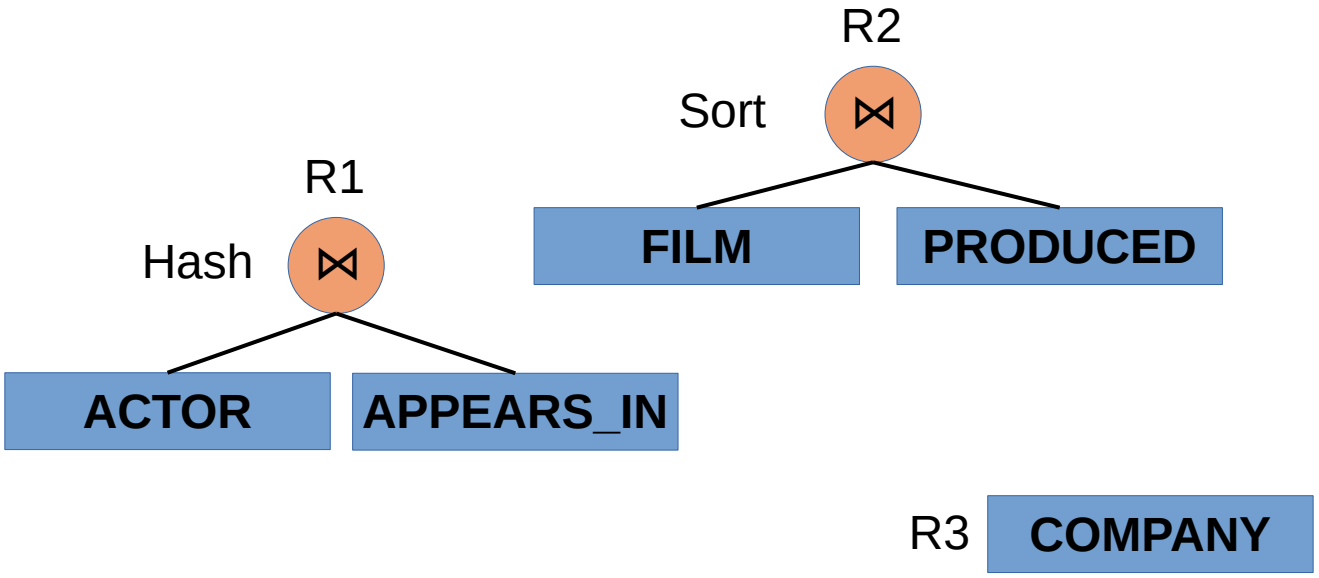


(ACTORS ⋈ APPEARS_IN ⋈ FILM ⋈ PRODUCED ⋈ COMPANY)

Query Optimization as an MDP

Actions available:

1. (R1, R2) Hash
2. (R1, R2) Sort
3. (R2, R3) Hash
4. **(R2, R3) Sort**
5. (R3, R4) Hash
6. (R3, R4) Sort

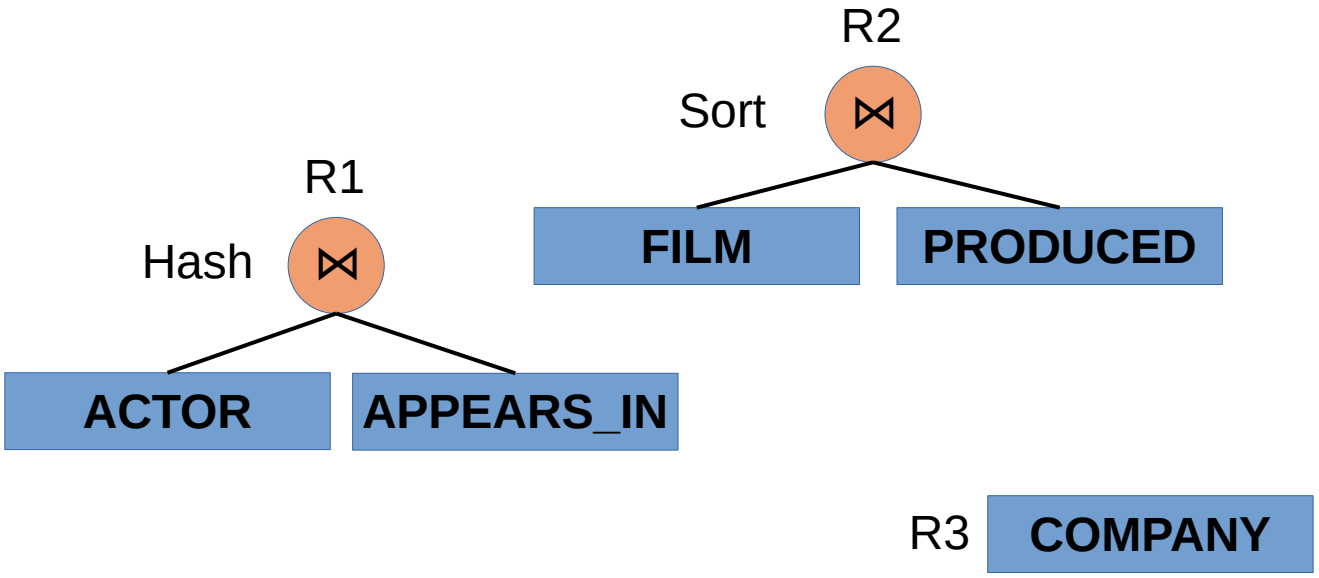


(ACTORS ⋈ APPEARS_IN ⋈ FILM ⋈ PRODUCED ⋈ COMPANY)

Query Optimization as an MDP

Actions available:

1. (R1, R2) Hash
2. (R1, R2) Sort
3. (R2, R3) Hash
4. (R2, R3) Sort

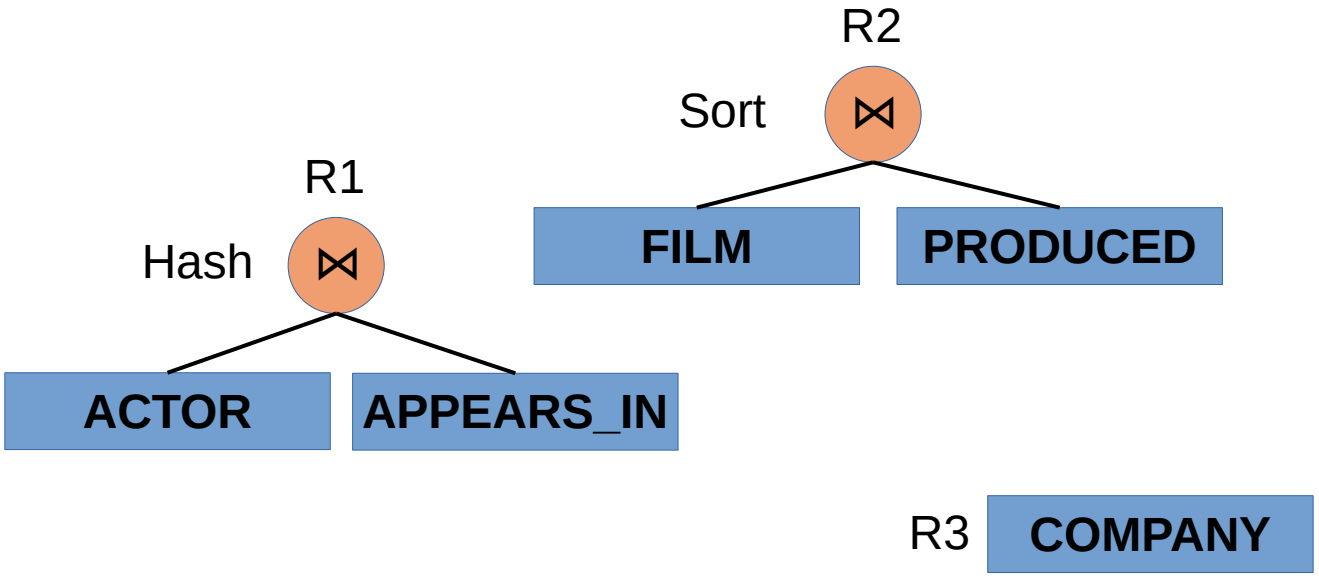


(ACTORS ⋈ APPEARS_IN ⋈ FILM ⋈ PRODUCED ⋈ COMPANY)

Query Optimization as an MDP

Actions available:

1. (R1, R2) Hash
2. (R1, R2) Sort
3. (R2, R3) Hash
4. **(R2, R3) Sort**

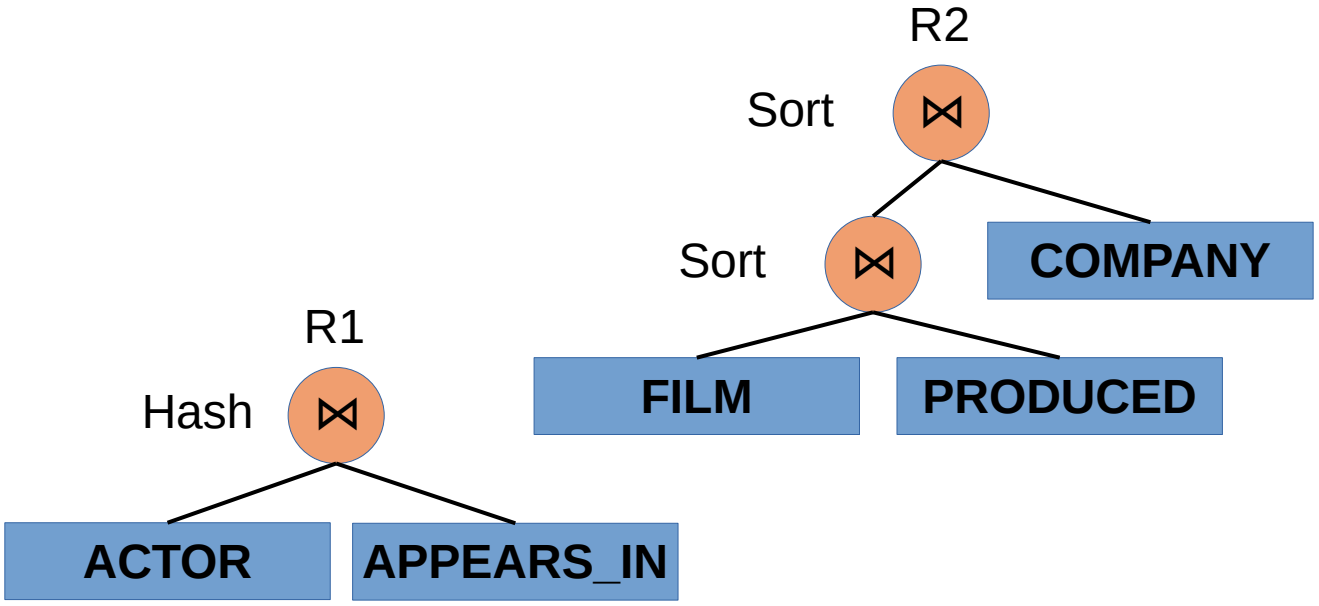


(ACTORS ⋈ APPEARS_IN ⋈ FILM ⋈ PRODUCED ⋈ COMPANY)

Query Optimization as an MDP

Actions available:

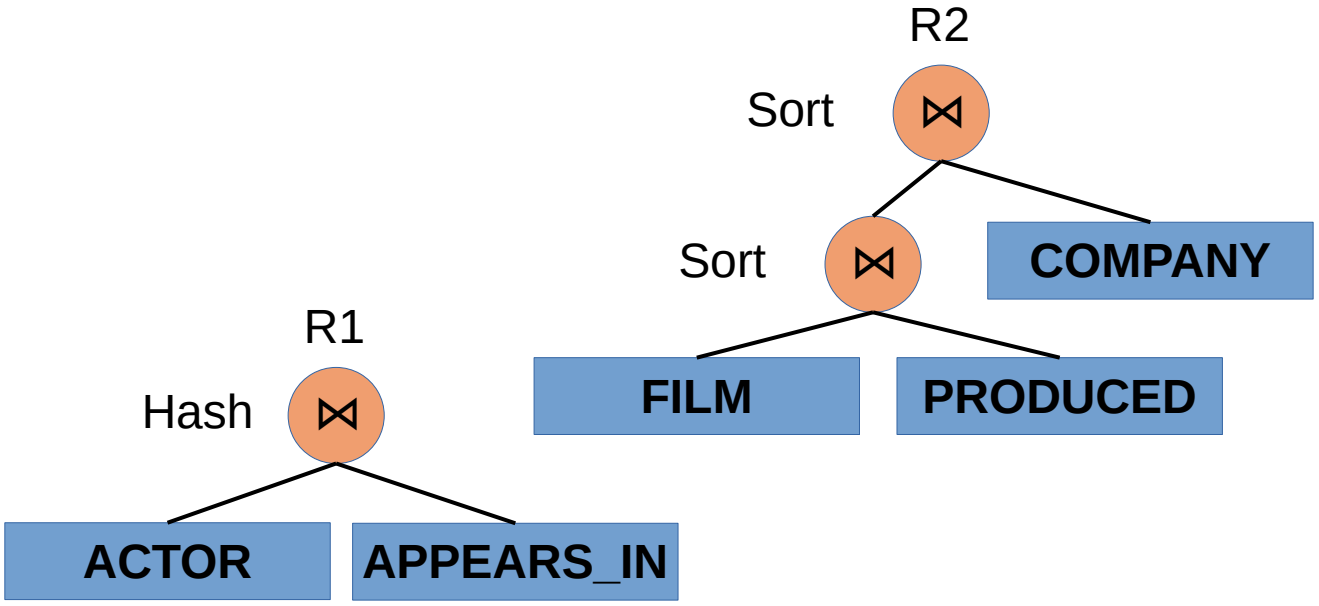
1. (R1, R2) Hash
2. (R1, R2) Sort
3. (R2, R3) Hash
4. **(R2, R3) Sort**



(ACTORS ⋈ APPEARS_IN ⋈ FILM ⋈ PRODUCED ⋈ COMPANY)

Query Optimization as an MDP

Actions available:
1. (R1, R2) Hash
2. (R1, R2) Sort

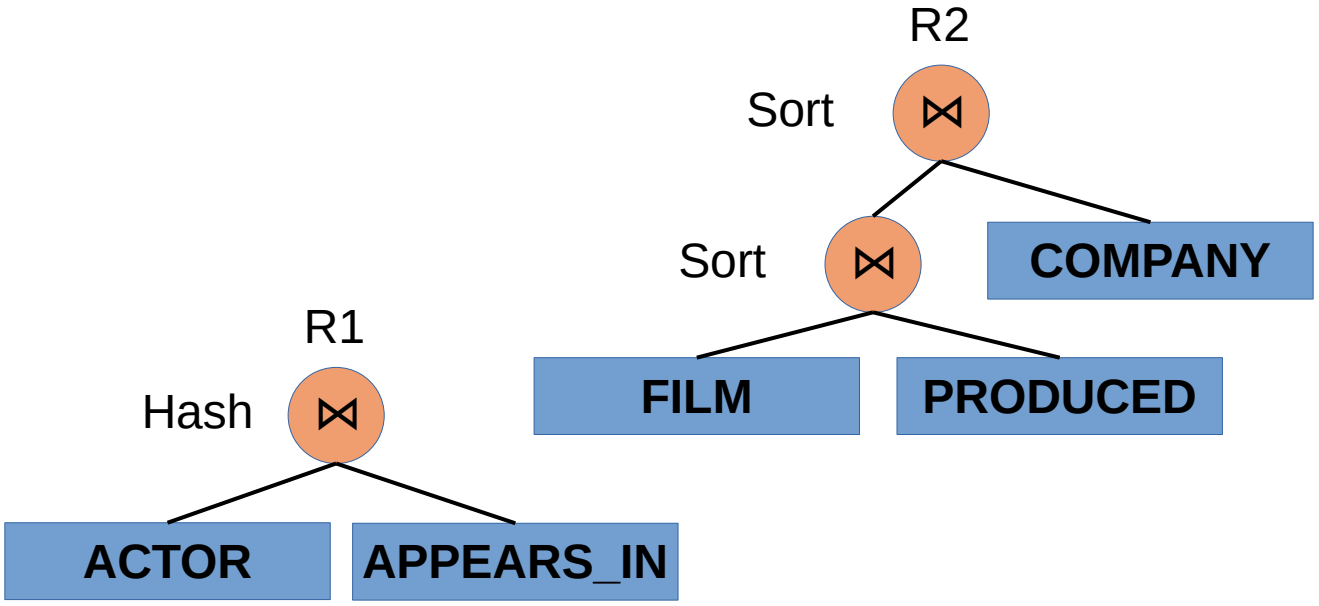


(ACTORS ⋈ APPEARS_IN ⋈ FILM ⋈ PRODUCED ⋈ COMPANY)

Query Optimization as an MDP

Actions available:

1. **(R1, R2) Hash**
2. (R1, R2) Sort

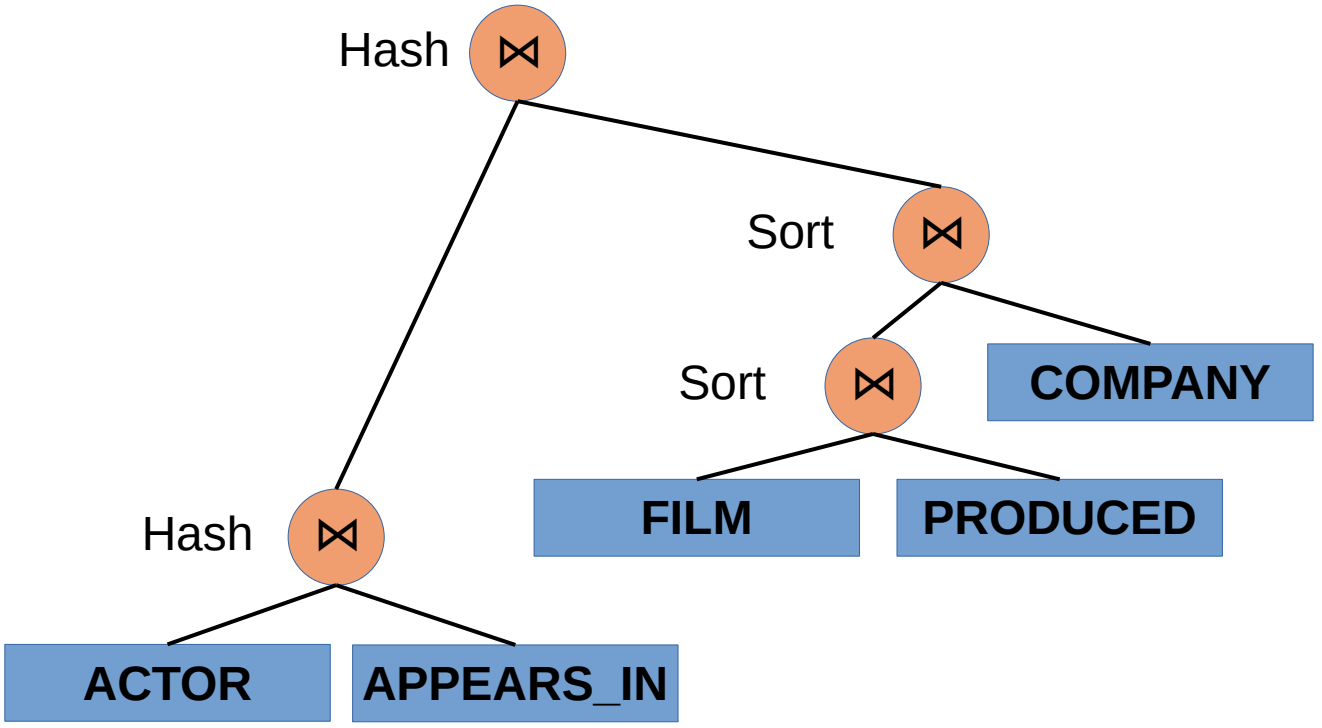


(ACTORS ⋈ APPEARS_IN ⋈ FILM ⋈ PRODUCED ⋈ COMPANY)

Query Optimization as an MDP

Actions available:

1. **(R1, R2) Hash**
2. (R1, R2) Sort



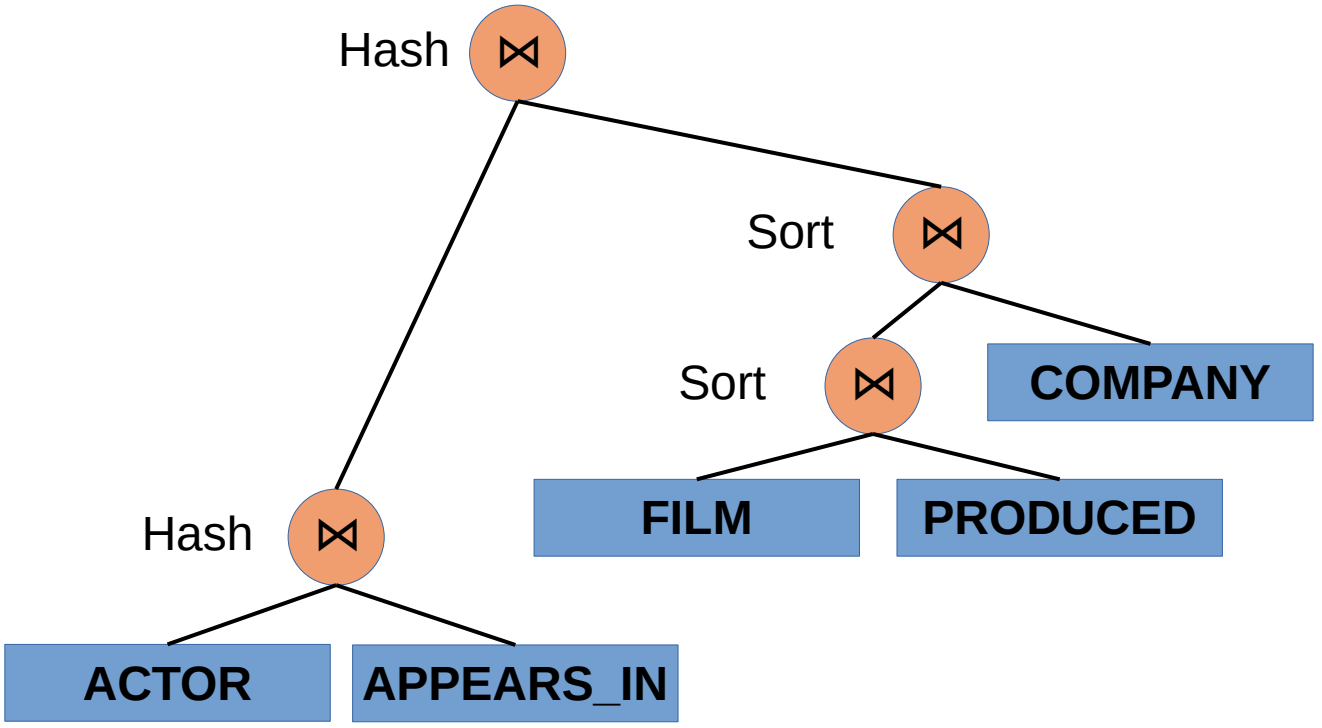
(ACTORS ⋈ APPEARS_IN ⋈ FILM ⋈ PRODUCED ⋈ COMPANY)

Query Optimization as an MDP

Every previous state had reward 0

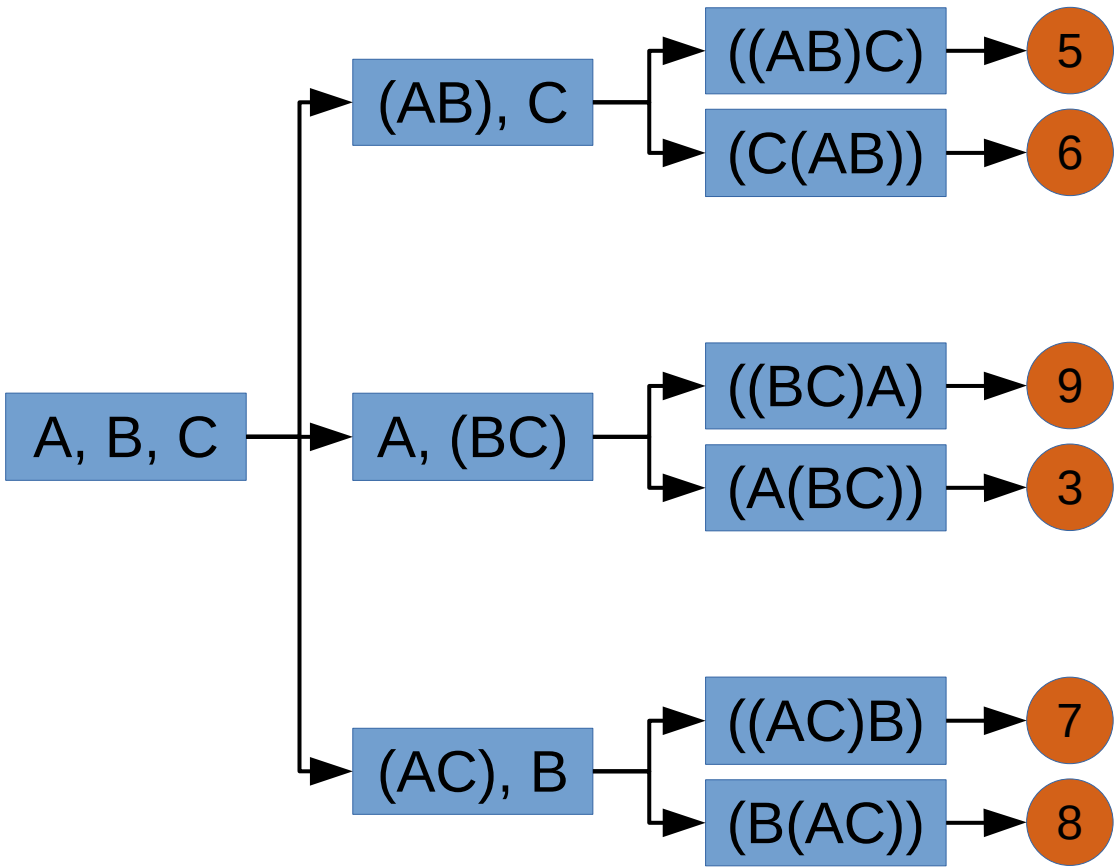
Now, we execute the program and record the latency.

Reward is -latency.

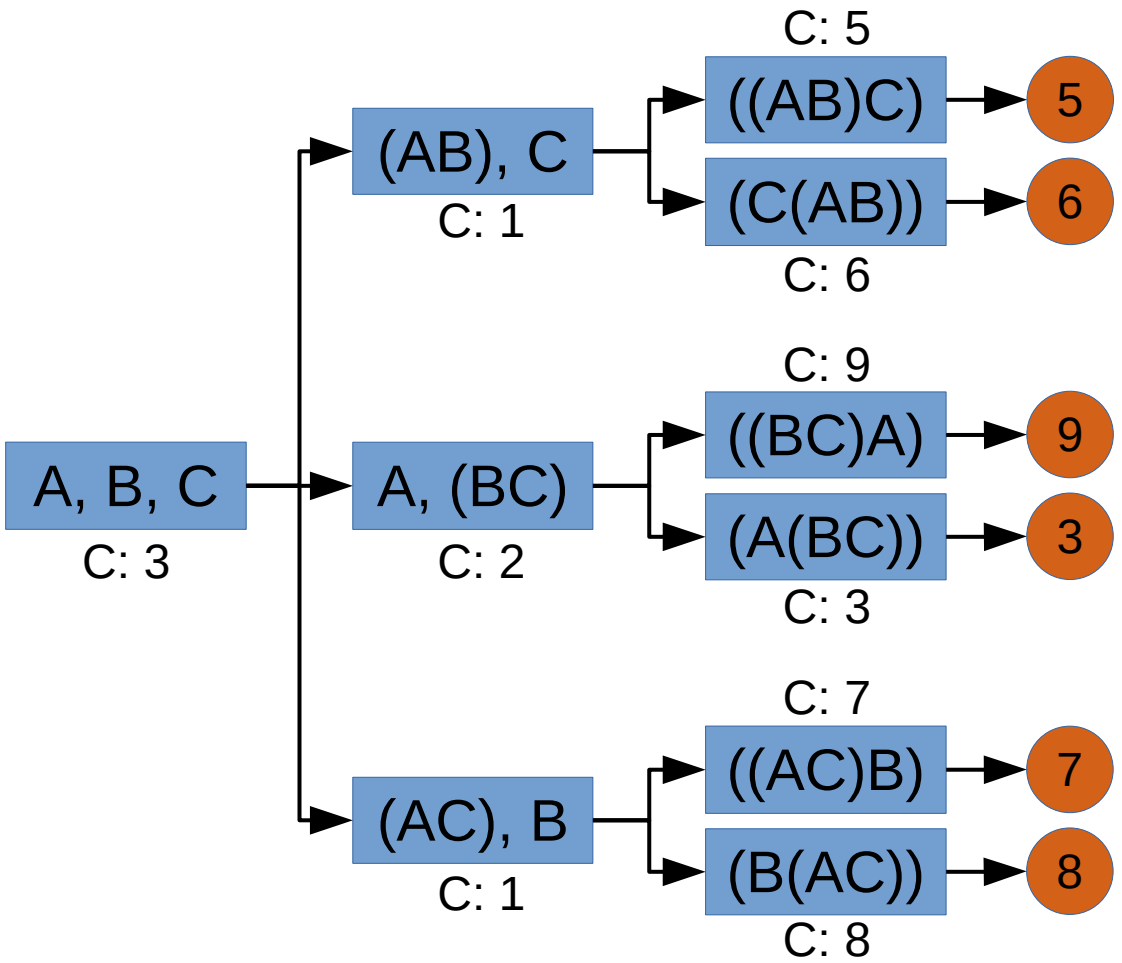


(ACTORS ⋈ APPEARS_IN ⋈ FILM ⋈ PRODUCED ⋈ COMPANY)

Deep Reinforcement Learning



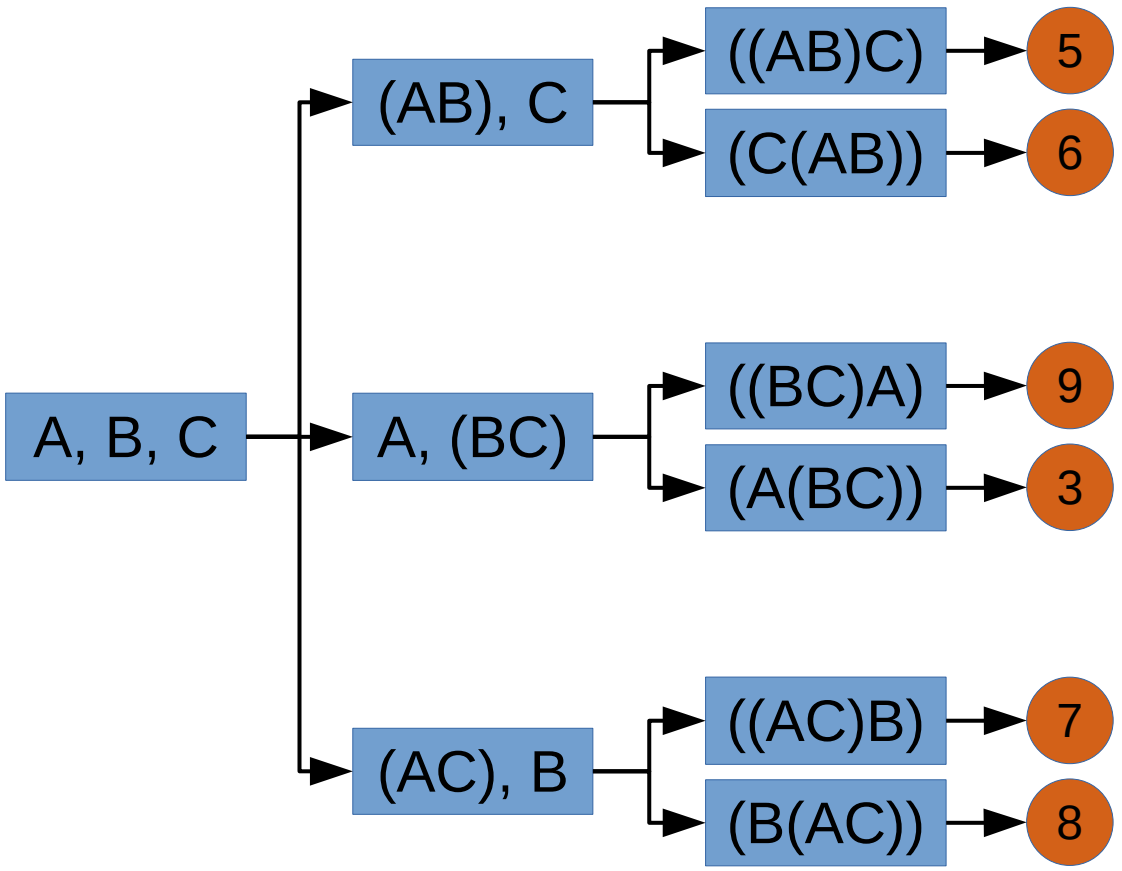
Deep Reinforcement Learning



Traditional Cost Model

A cost function C which estimates the intermediate cost of plan

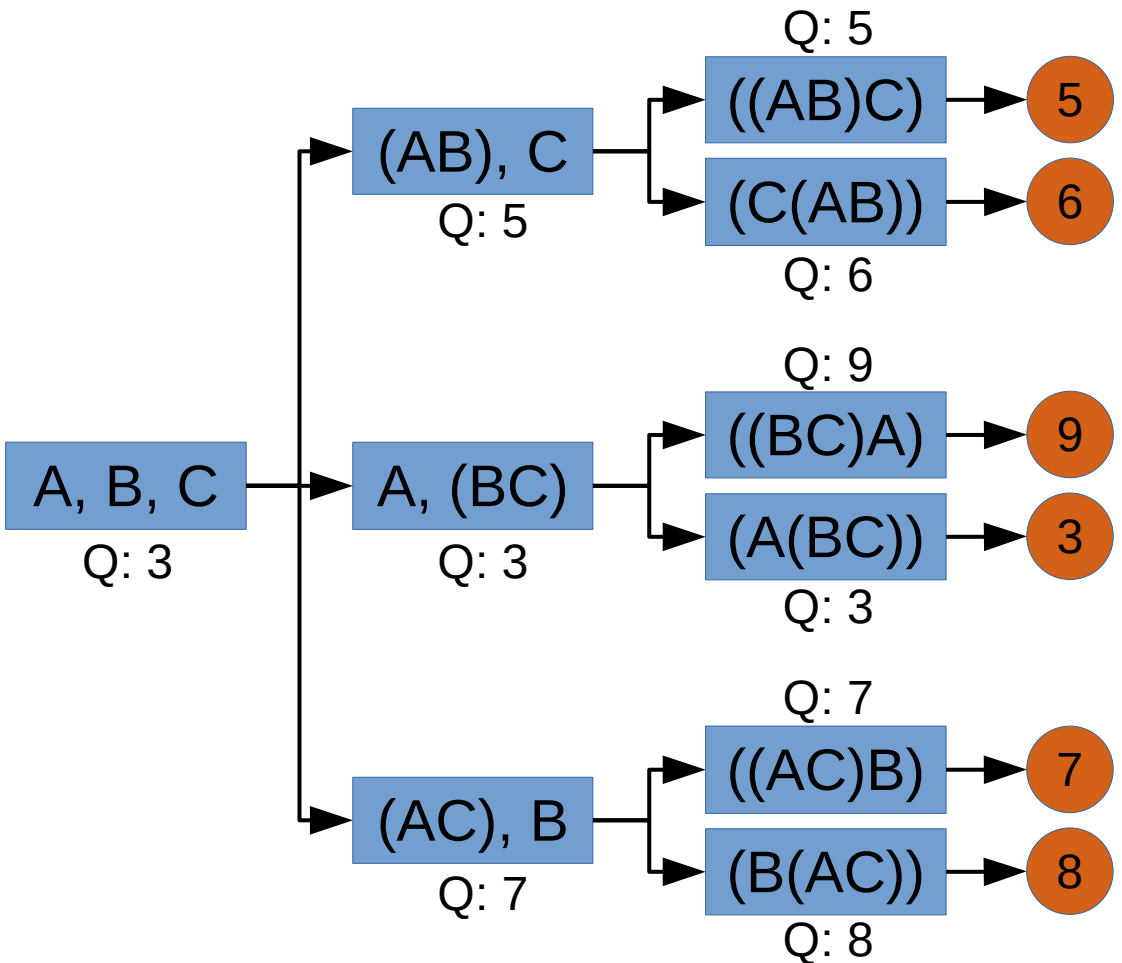
Deep Reinforcement Learning



Deep reinforcement learning

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

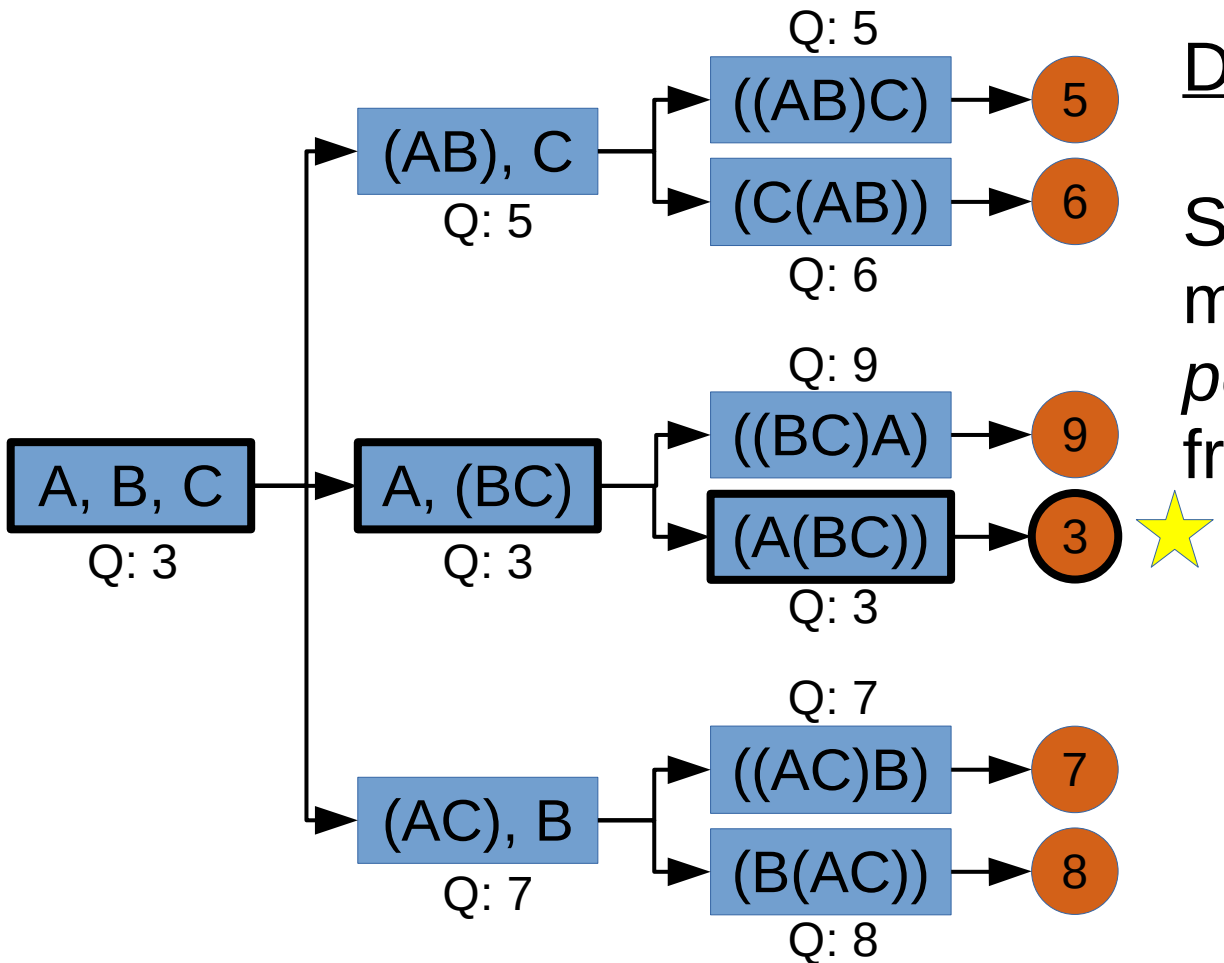
Deep Reinforcement Learning



Deep reinforcement learning

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

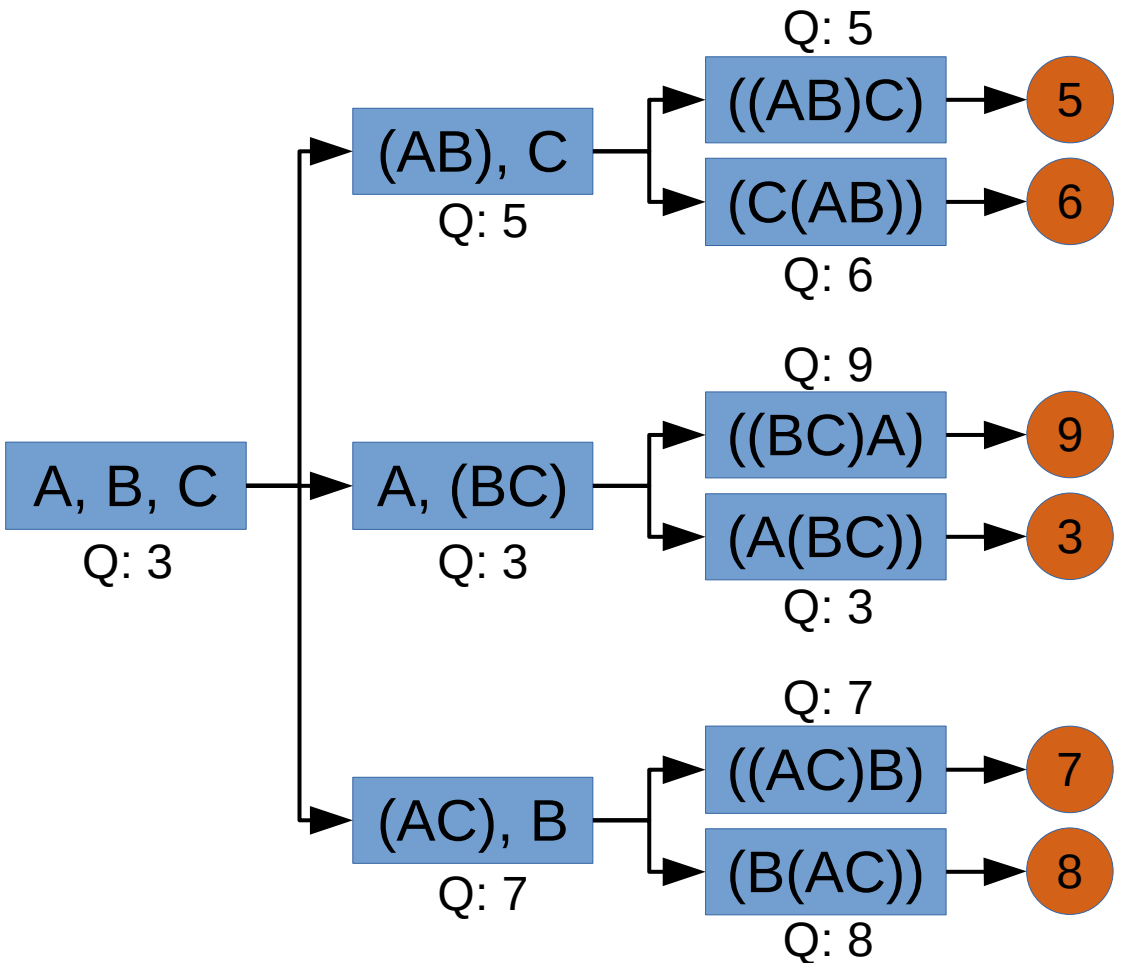
Deep Reinforcement Learning



<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



Deep reinforcement learning

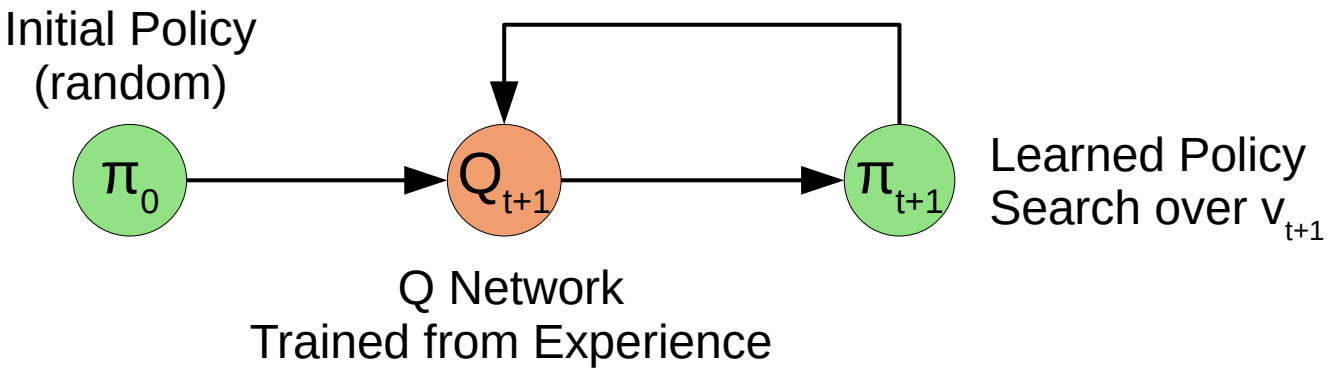
Supp. an oracle Q(·) which maps each state to the *best possible latency achievable* from that state.

Of course, there's no Q(·).

… so we will learn an approximation, $\hat{Q}$

Deep Reinforcement Learning

- Value iteration

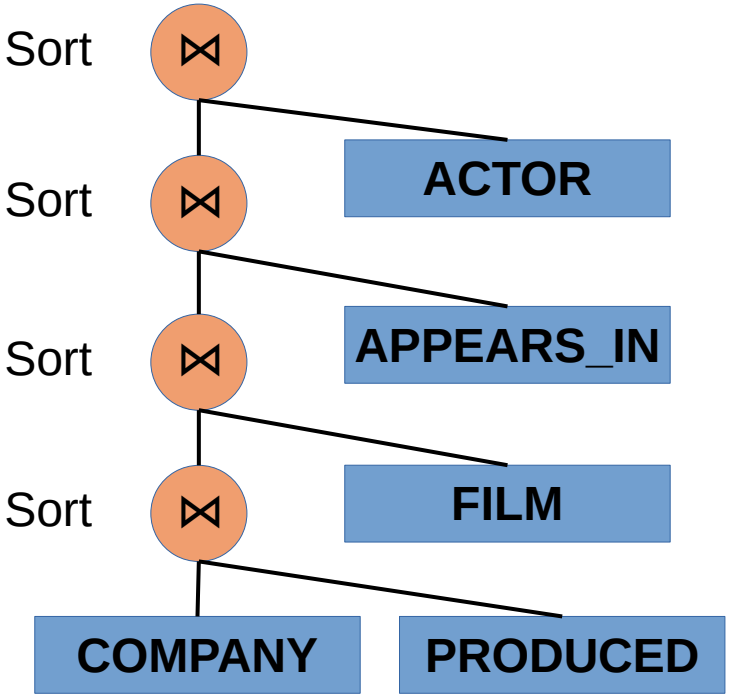


Inductive Bias

- How should we approximate the Q function?
- Option 1
  - Flatten the state into a vector
  - Use a fully connected neural network
- Not really how deep learning becomes successful
- Option 2
  - Try to find the right *inductive bias*
  - Build an intuitive network architecture

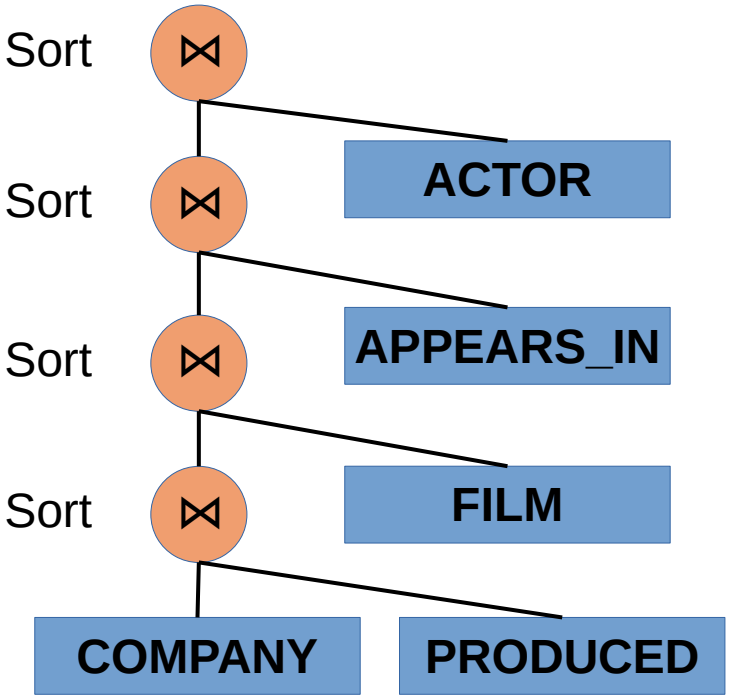
Tree Convolution

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



Tree Convolution

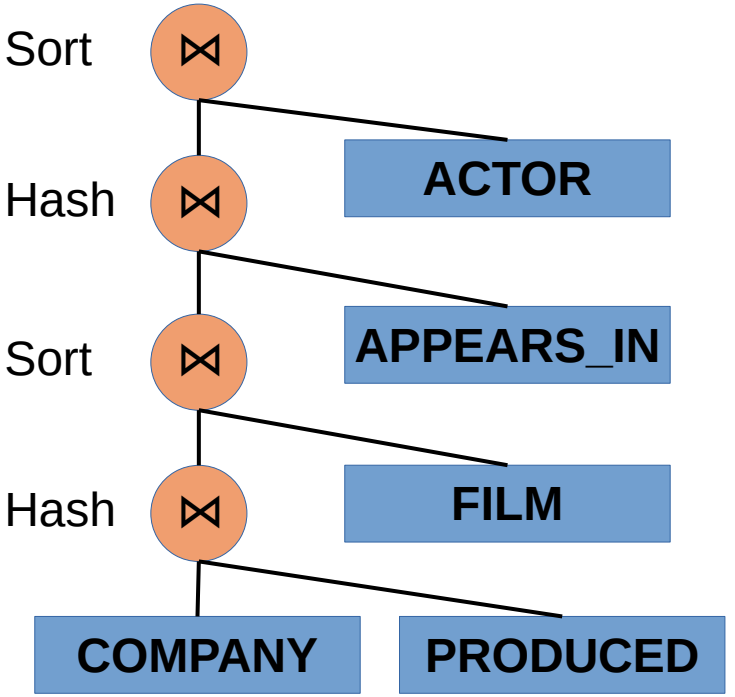
- How do we come up with a good inductive bias for query plans?



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

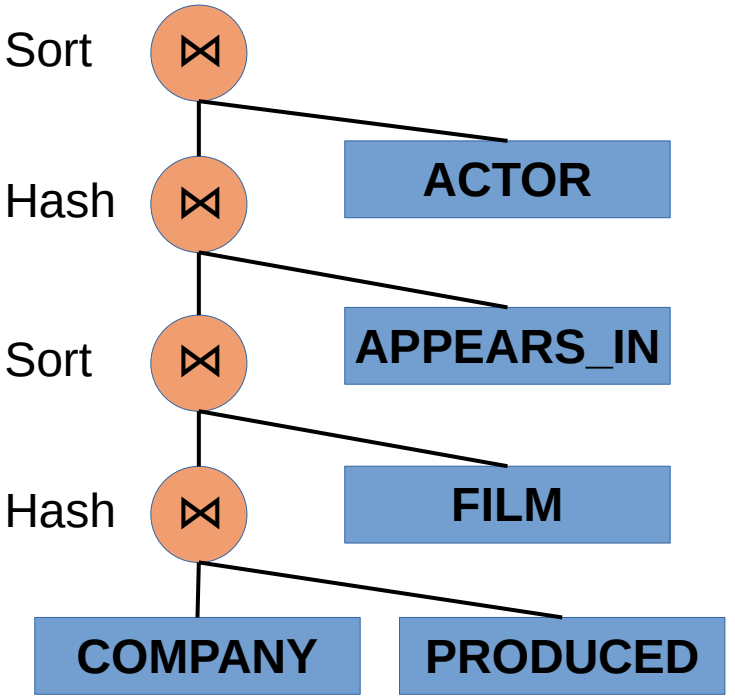
Tree Convolution

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



Tree Convolution

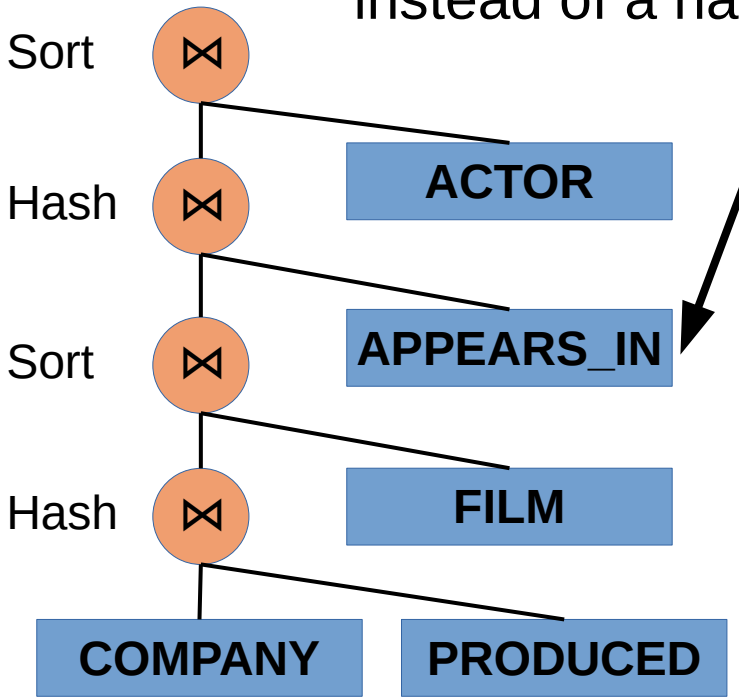
- How do we come up with a good inductive bias for query plans?



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

Tree Convolution

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

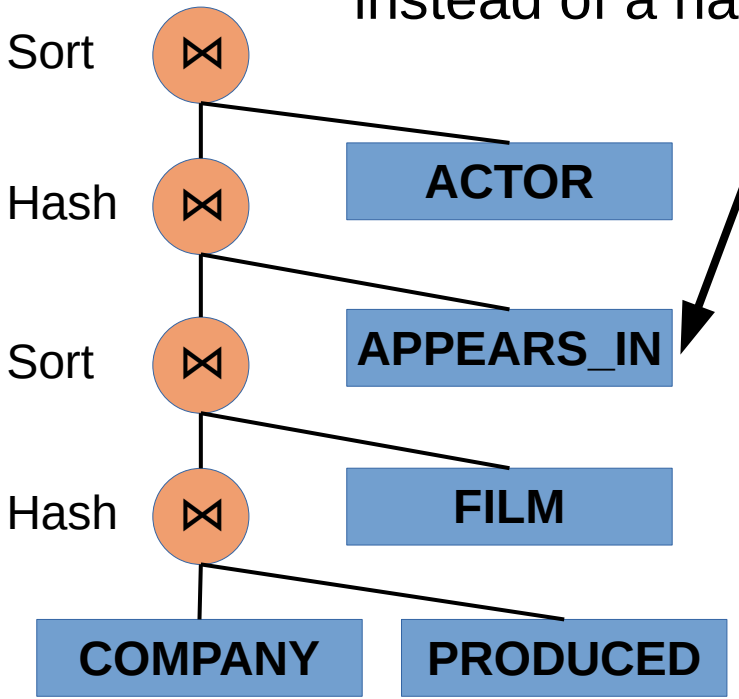


"APPEARS_IN" is presorted on disk – should use a sort instead of a hash.

"Hash then sort, 100% requires rehash or resort."

Tree Convolution

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

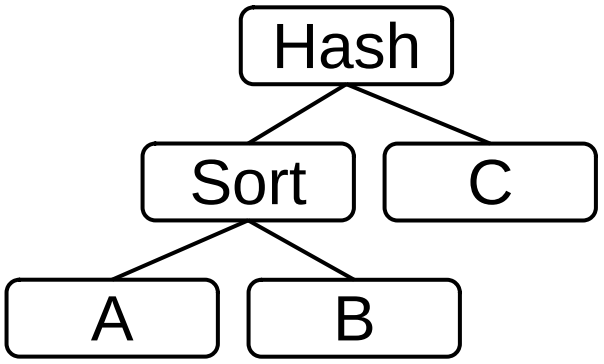


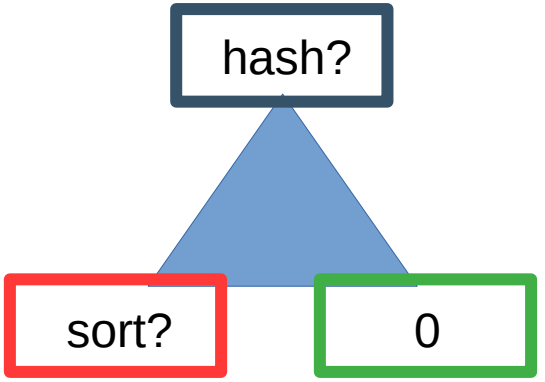
"APPEARS_IN" is presorted on disk – should use a sort instead of a hash.

"Hash then sort, 100% requires rehash or resort."

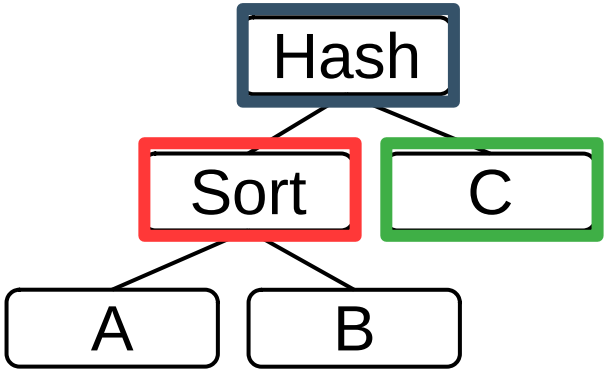
Experts examine *local structure* first, then look to higher level features.

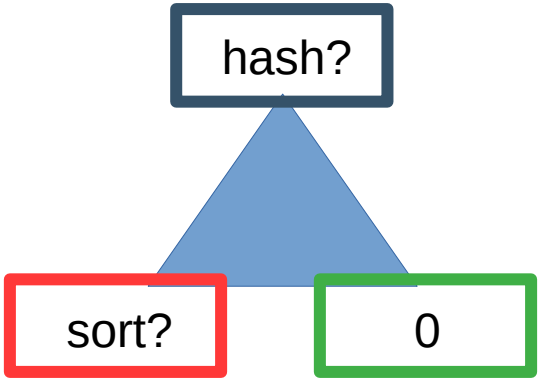
Tree Convolution



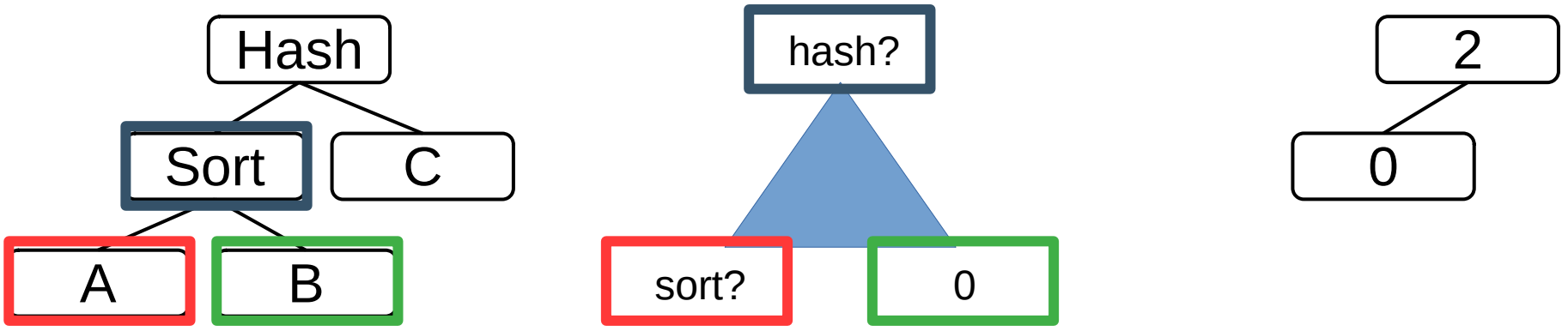


Tree Convolution

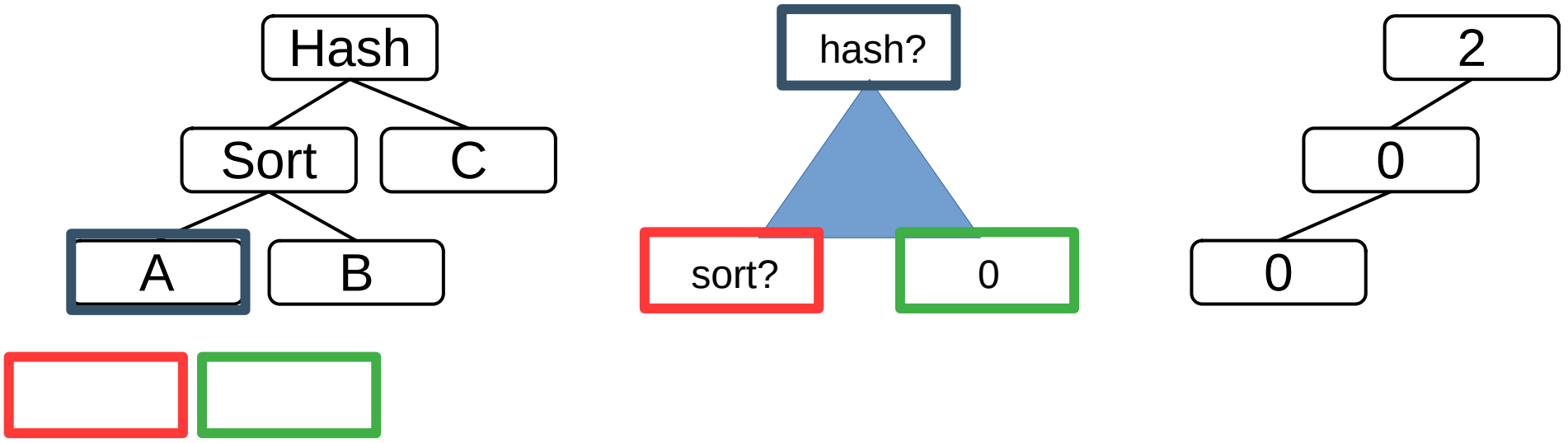




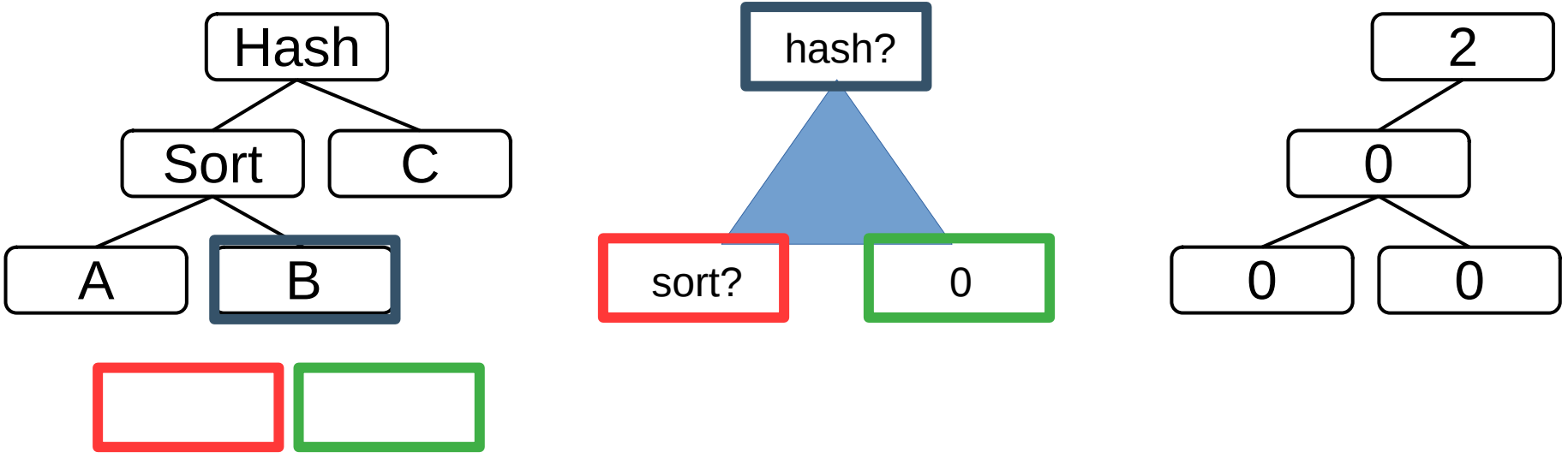
Tree Convolution



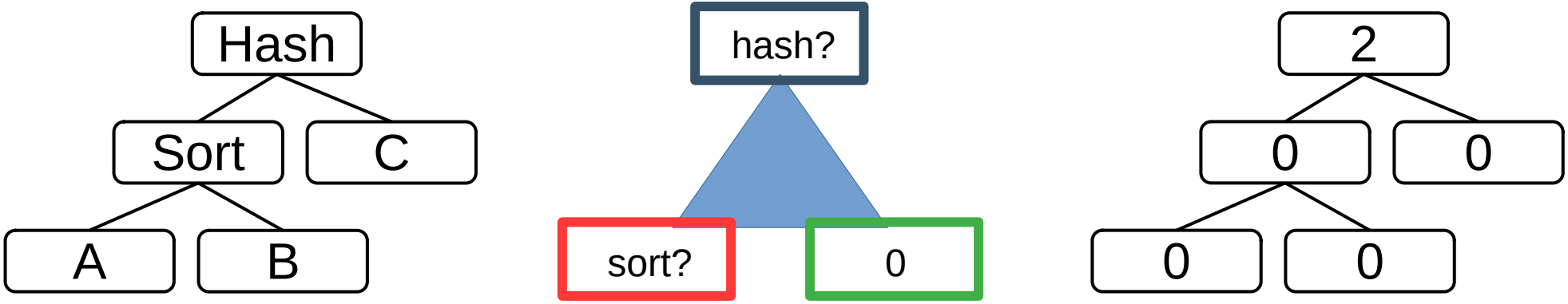
Tree Convolution



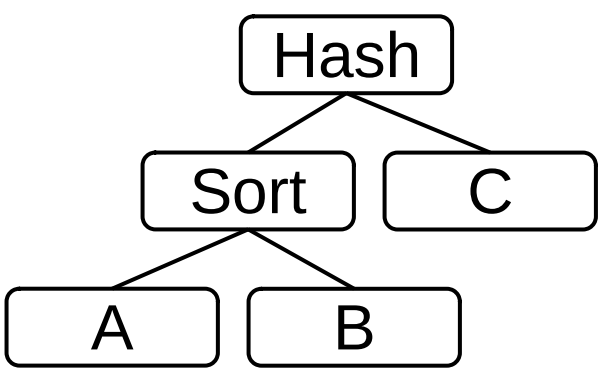
Tree Convolution

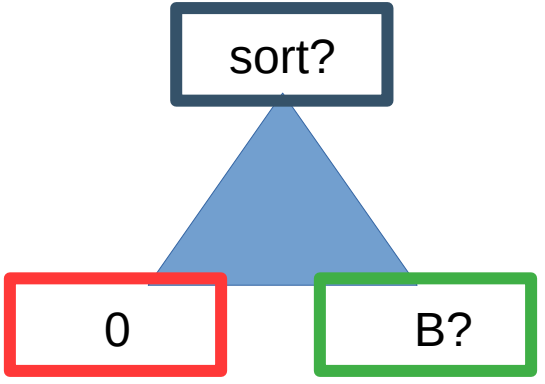


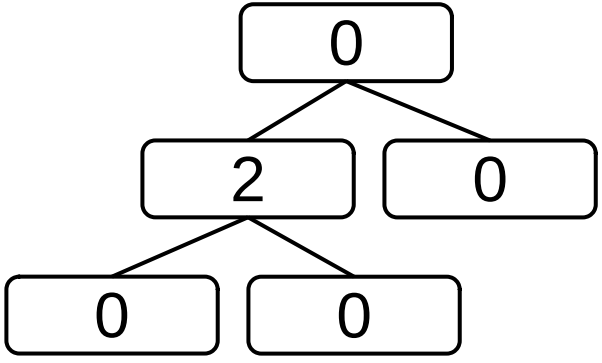
Tree Convolution



Tree Convolution



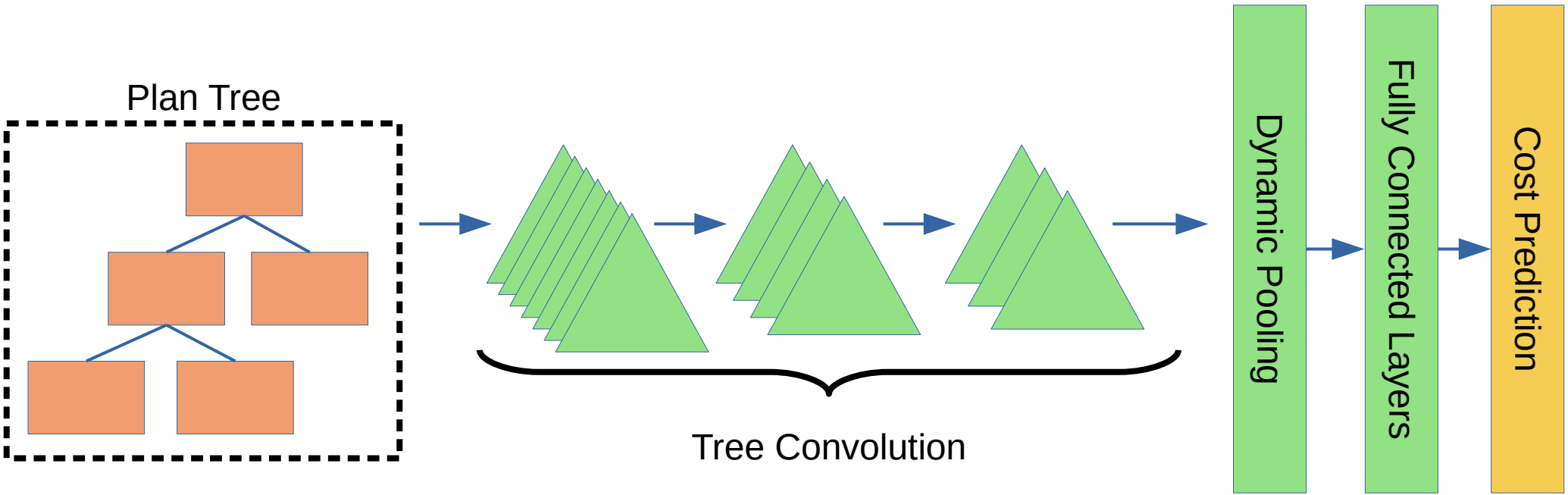




Tree Convolution

- Like image convolution, filter weights are:
  - Automatically learned
  - Stacked (to learn higher-level features)
- Efficiently vectorized on a GPU

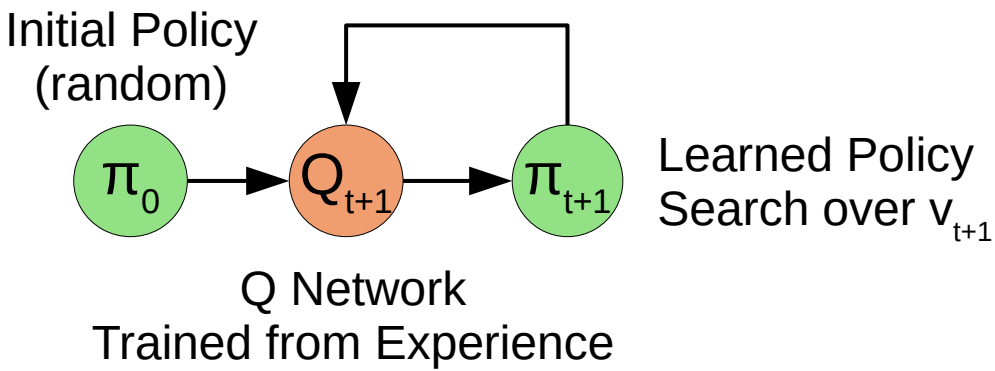
Neo



Value network architecture (used to approximate Q)

Random Policies

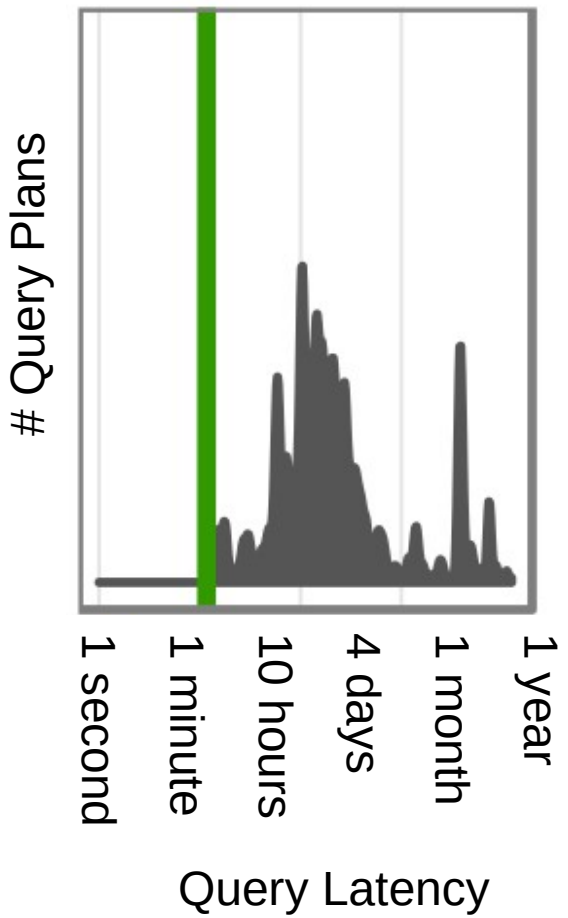
- DRL is very *sample inefficient*
  - You have to play for a long time before you get good.
- In QO, **doing worse takes longer!**
  - Cannot afford a random initial policy.



* not the exact histogram… credit to Leis et al.

Random Policies

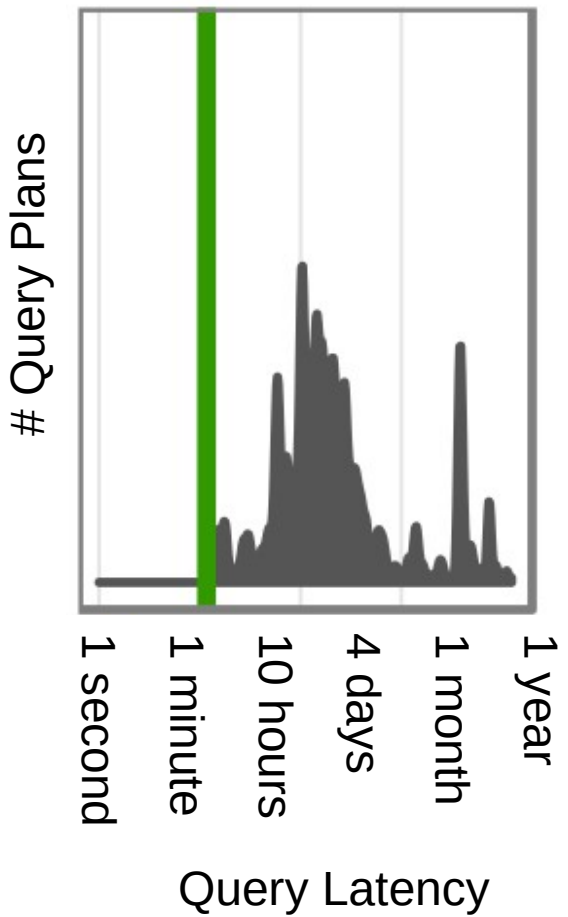
- DRL is very *sample inefficient*
  - You have to play for a long time before you get good.
- In QO, **doing worse takes longer!**
  - Cannot afford a random initial policy.



* not the exact histogram… credit to Leis et al.

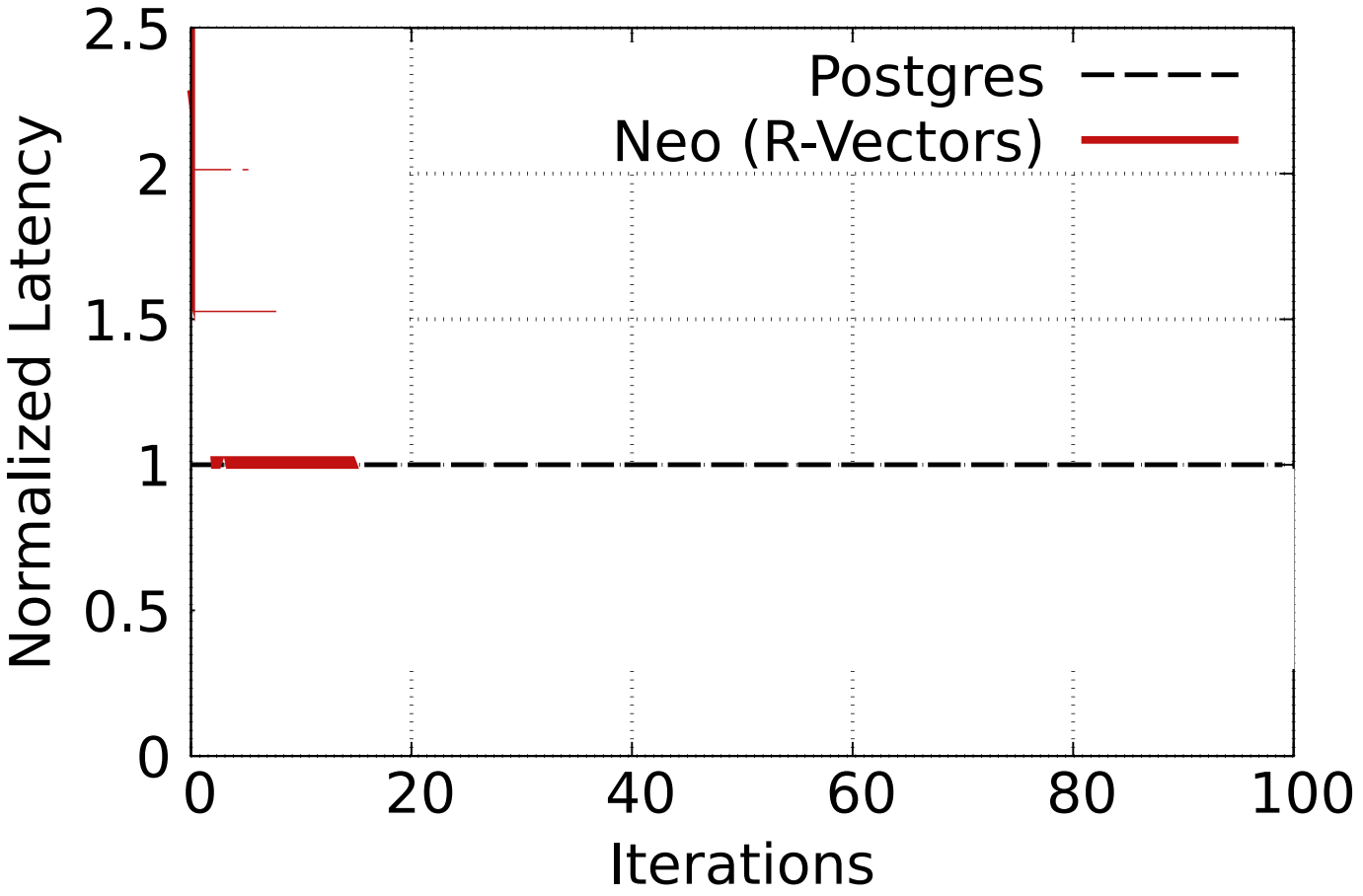
Random Policies

- Heuristic query optimizers have been around for a long time.
  - Some are very simple, like Selinger et al., ‘89
  - This is the green line.
- So instead of starting from random…
  - Use a simple heuristic system to bootstrap our policy.



* not the exact histogram… credit to Leis et al.

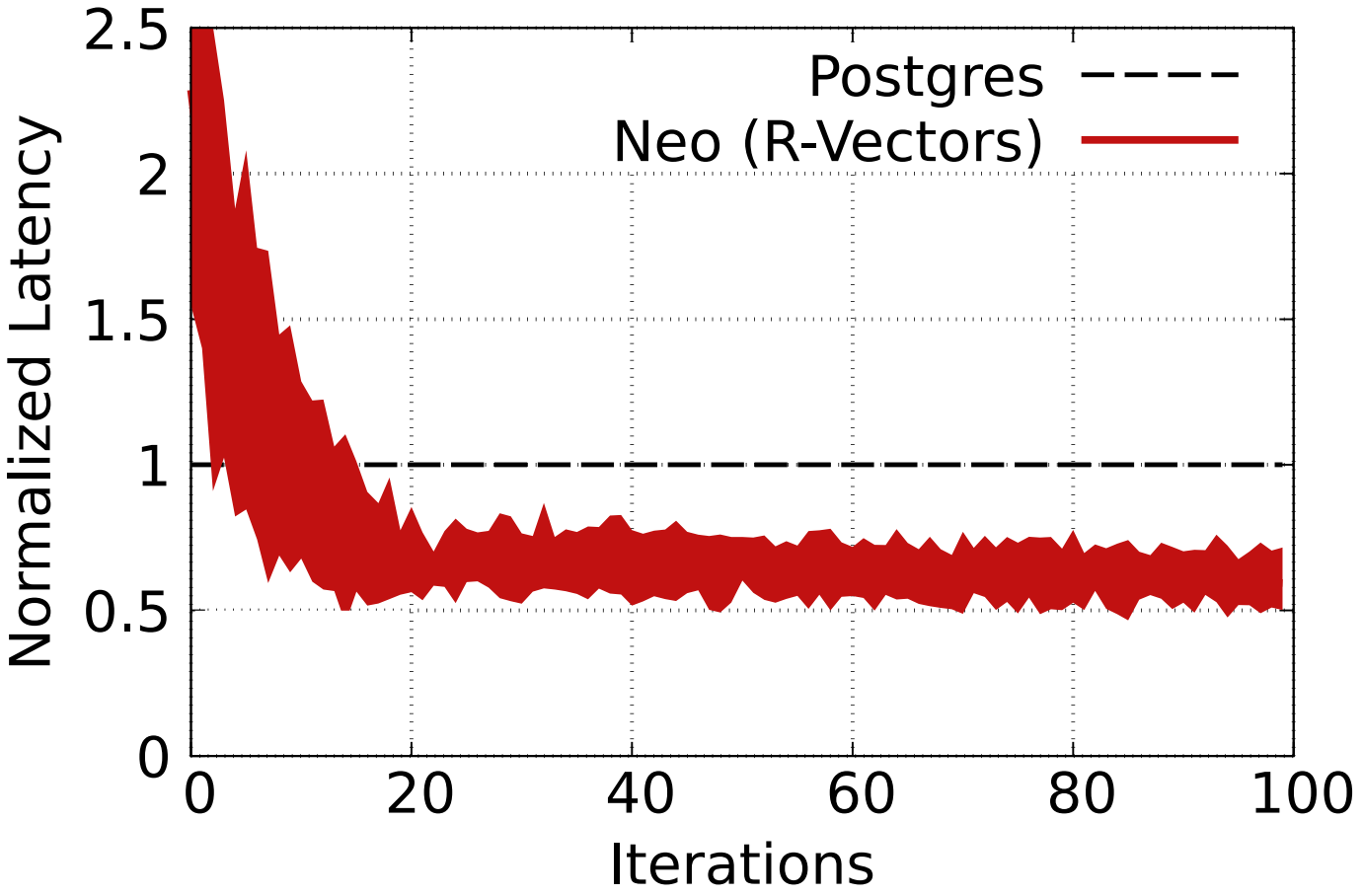
Experiments



1 = performance of PG optimizer

Neo trained with PG optimizer as expert on small sample beforehand.

Experiments

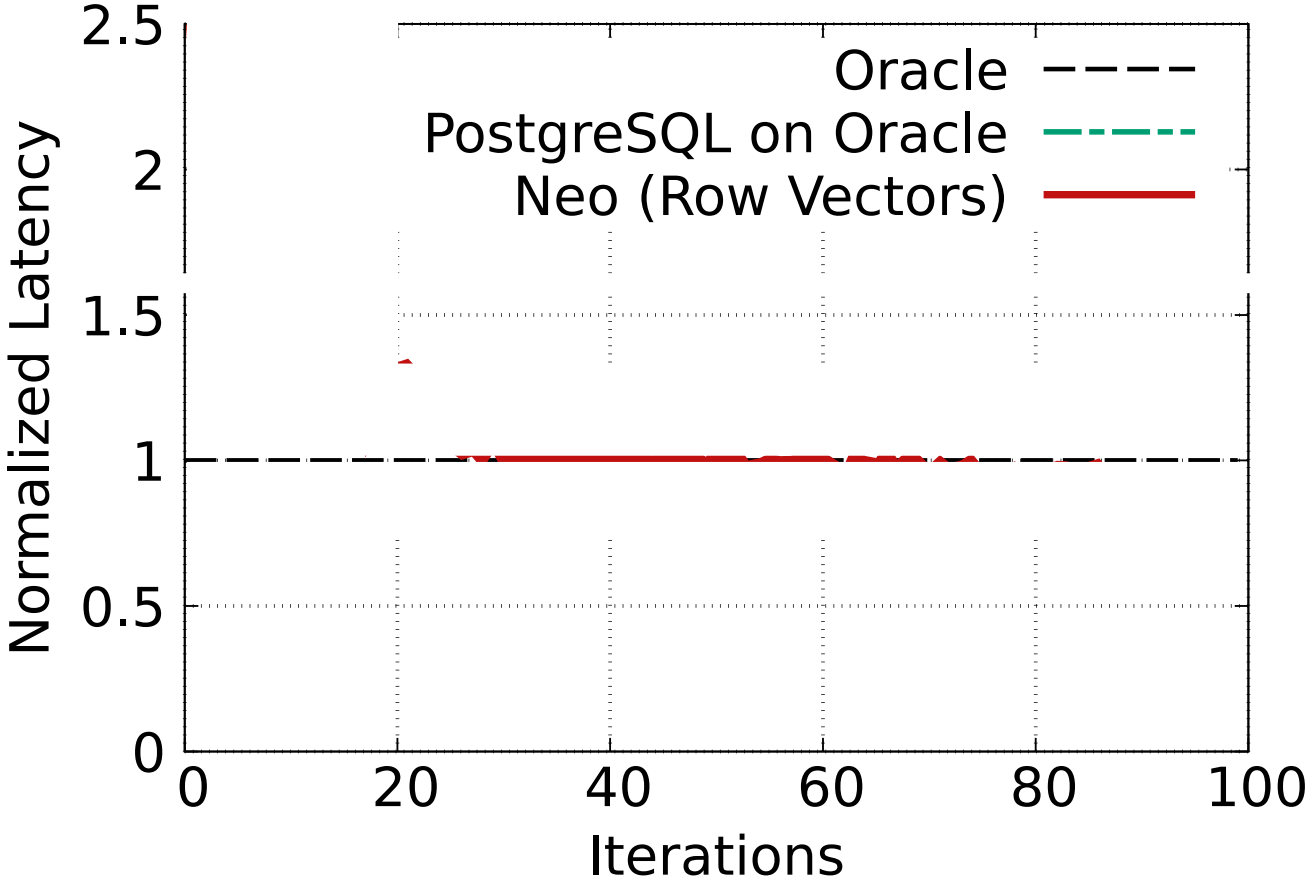


1 = performance of PG optimizer

Neo trained with PG optimizer as expert on small sample beforehand.

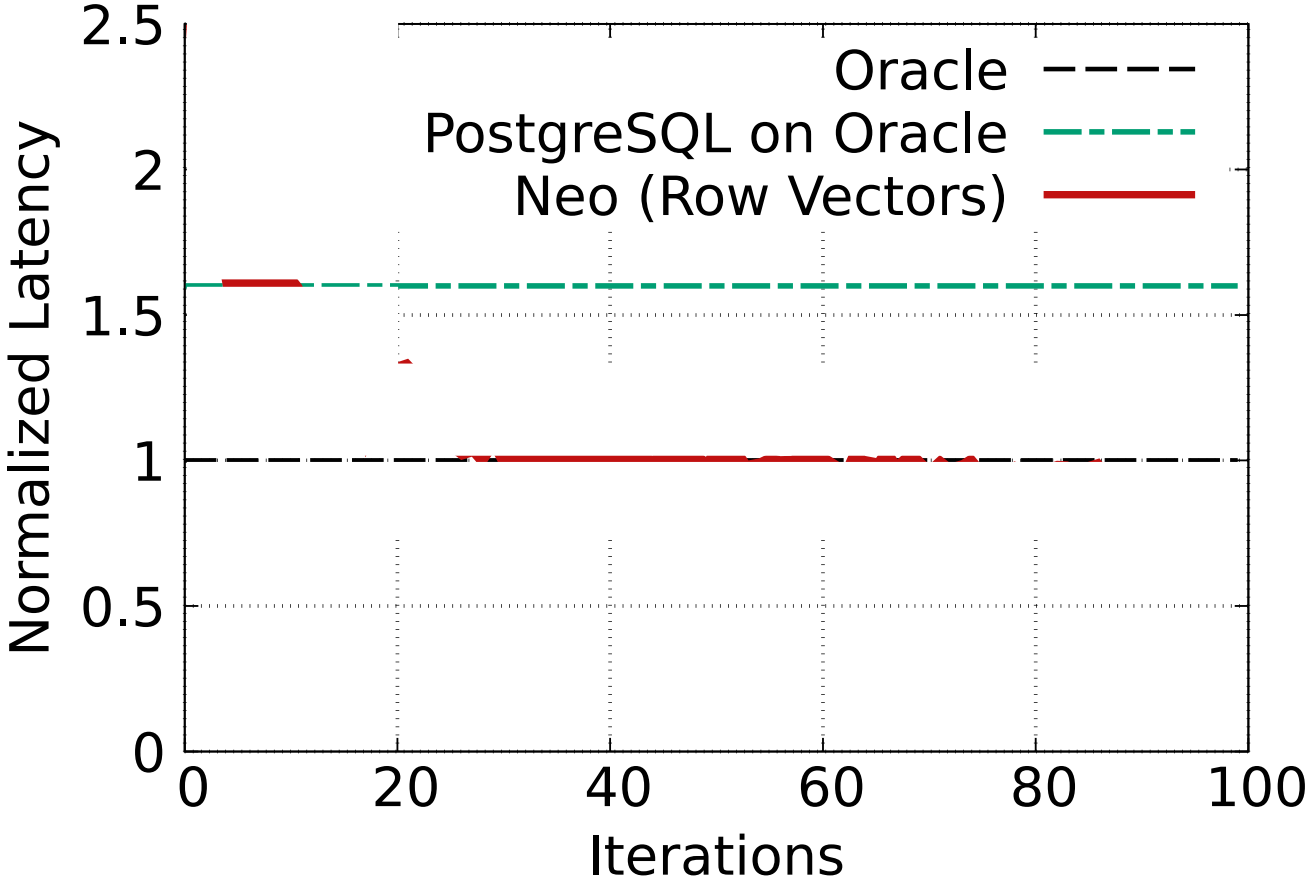
On test queries, Neo outperforms PG 15-25%.

Experiments



Black (1): performance of Oracle query optimizer

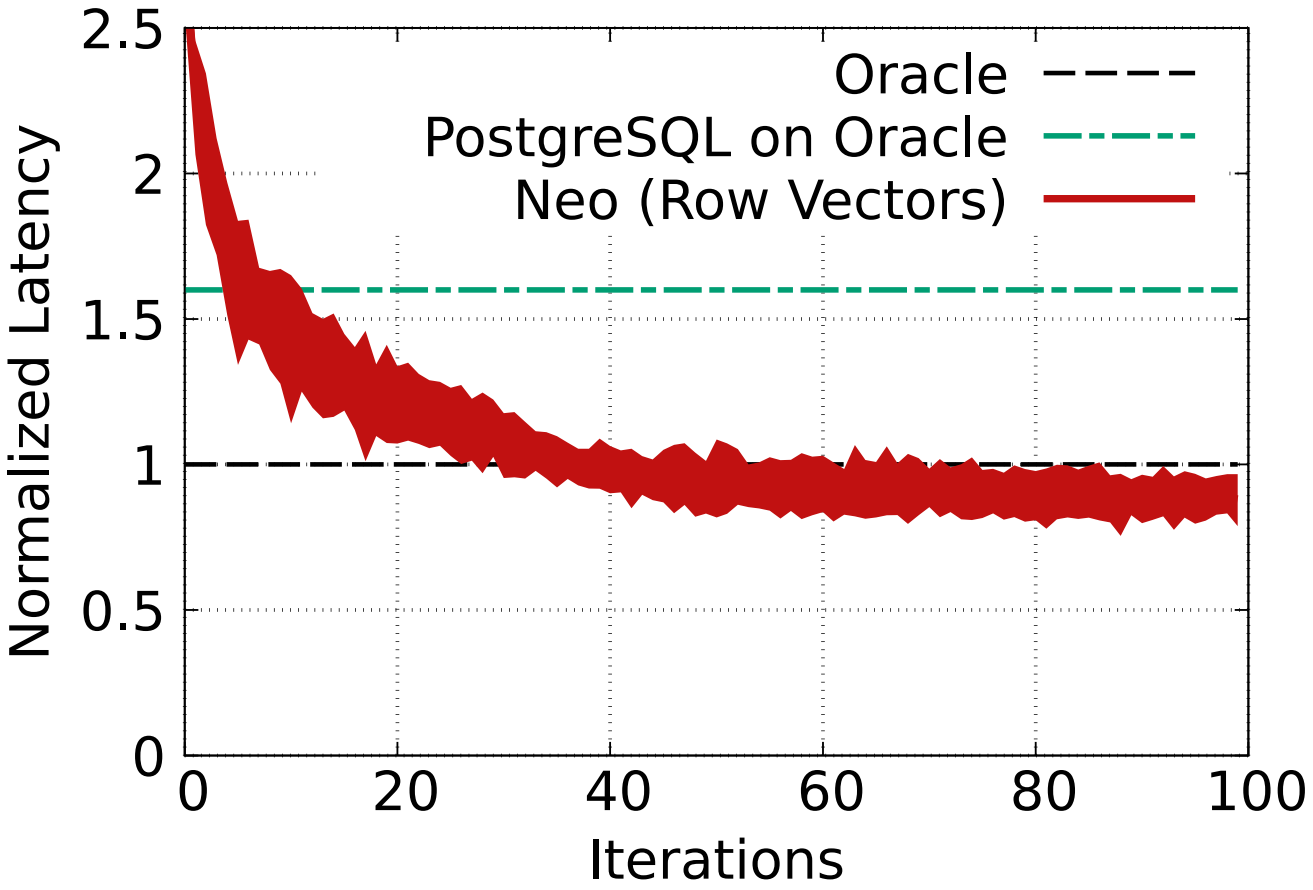
Experiments



Black (1): performance of Oracle query optimizer

Green (1.7): performance of PostgreSQL plans executed on Oracle

Experiments

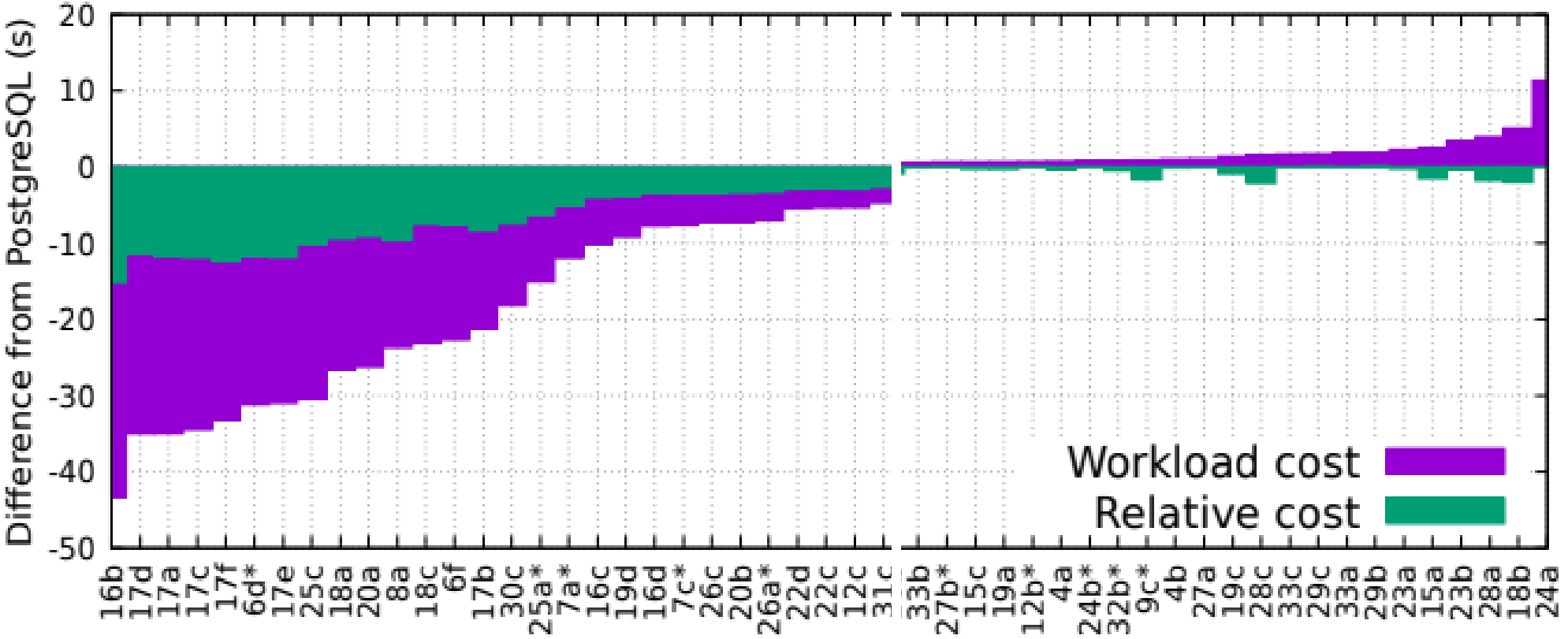


Black (1): performance of Oracle query optimizer

Green (1.7): performance of PostgreSQL plans executed on Oracle

Red: Performance of Neo over time

Experiments

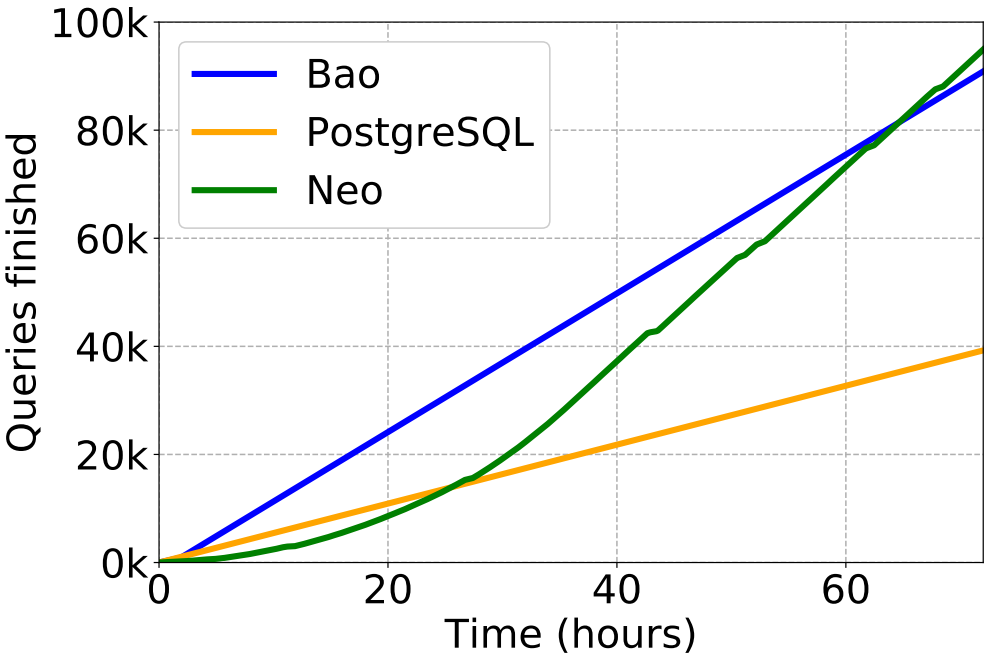


Conclusions

- Neo: first learned end-to-end optimizer
- Achieves performance on-par with SOTA commercial query optimizers
- Limitations & future work
  - Depends on an expert
  - Fixed schema
  - Concurrent queries

Coming Soonish

- Bao: Bandit Optimizer
- Current optimizers might not be good at picking the best plan, but they're **great at avoiding terrible plans.**
- Can we multiplex simple optimizers together using learning?



That's All

- These slides: https://ryan.cab/bu20
- Website: https://ryan.cab
- Me, on Twitter: @RyanMarcus
- Will find good inductive biases for food!
  - Current postdoc at CSAIL, looking for a position next year!
- Email: ryanmarcus@csail.mit.edu