Neo: A Learned Query Optimizer

Ryan Marcus, Parimarjan Negi, Hongzi Mao, Chi Zhang,
Mohammad Alizadeh, Tim Kraska,
Olga Papaemmanouil, Nesime Tatbul

ryanmarcus@csail.mit.edu
Twitter: @RyanMarcus

Paper: http://rm.cab/neo

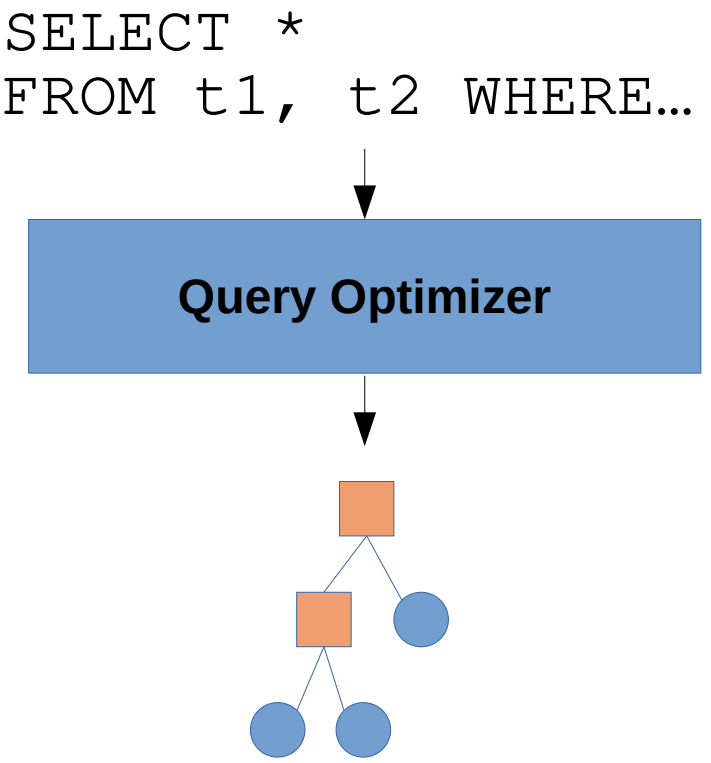
Slides: http://rm.cab/neovldb19

DSAIL
Data Systems and AI Lab

Microsoft

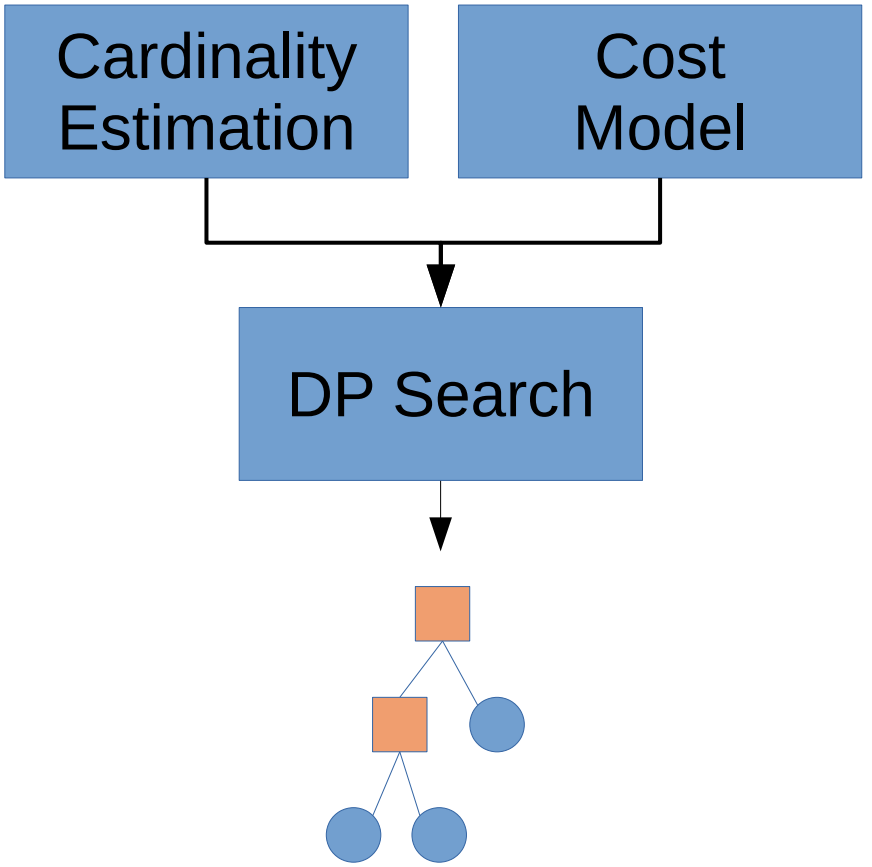
Query Optimizers

- 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

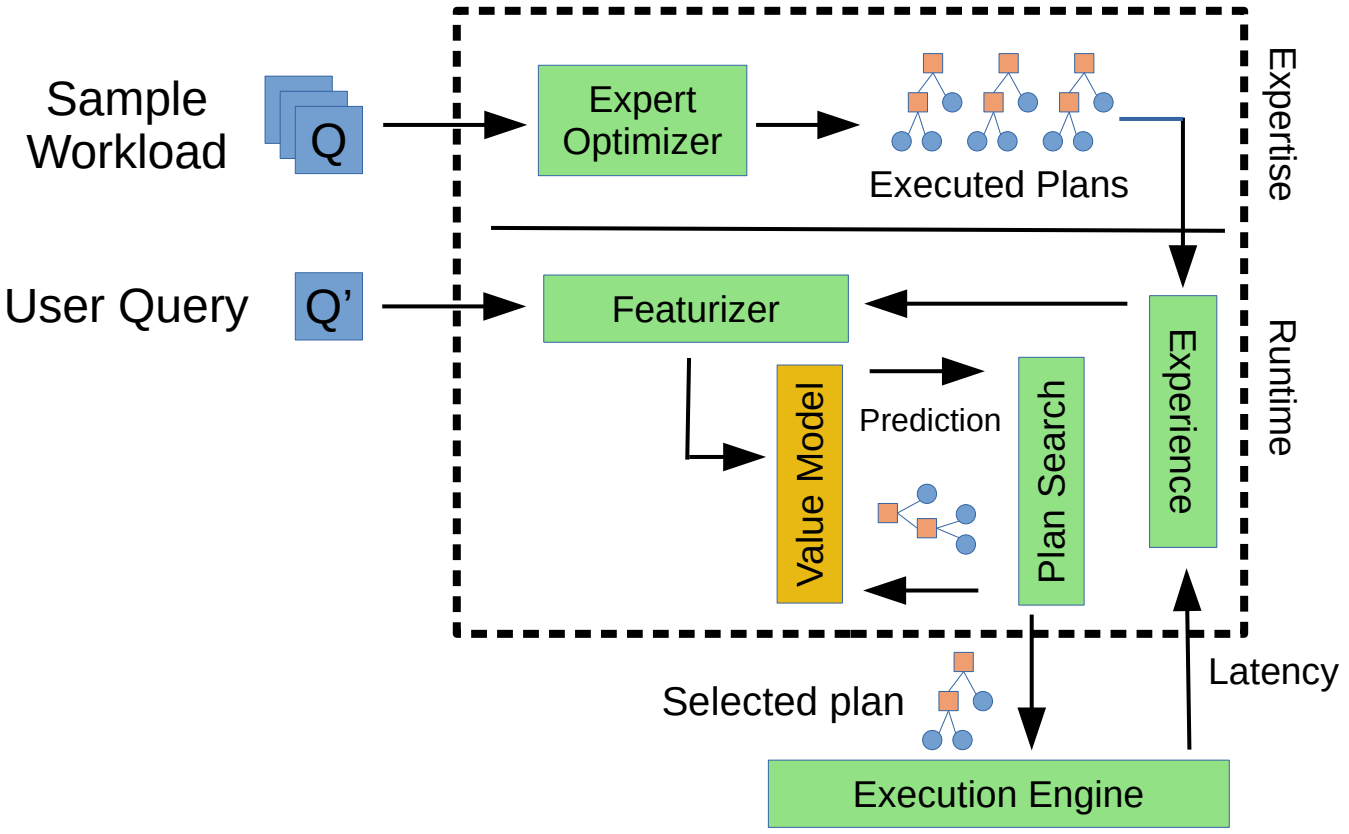
- Cardinality estimation models
  - Histograms
  - Uniformity
  - MFVs
- Cost models
  - Polynomials
  - Hand tuned
- DP Search
  - NP-Hard



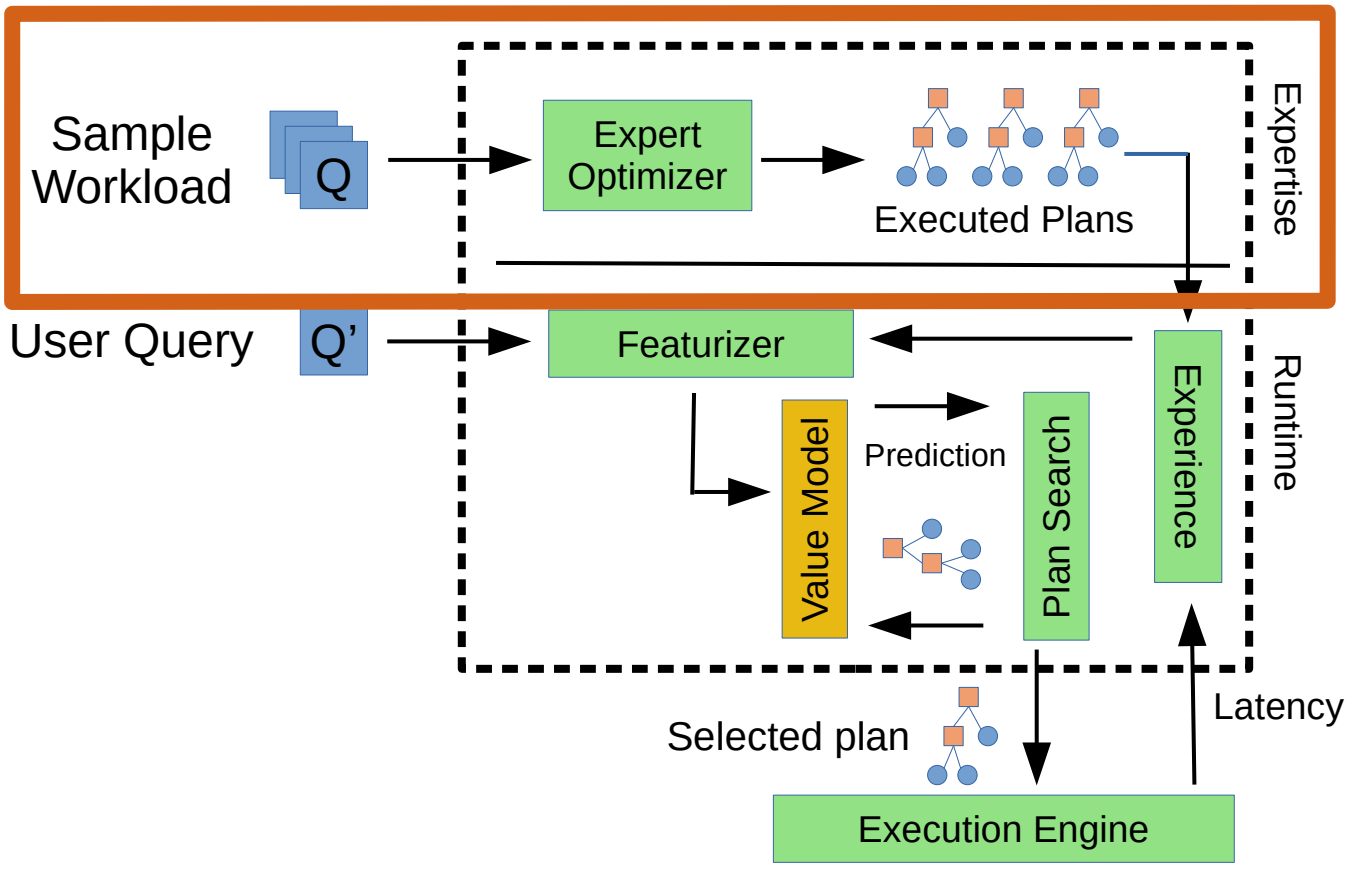
Neo

- 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

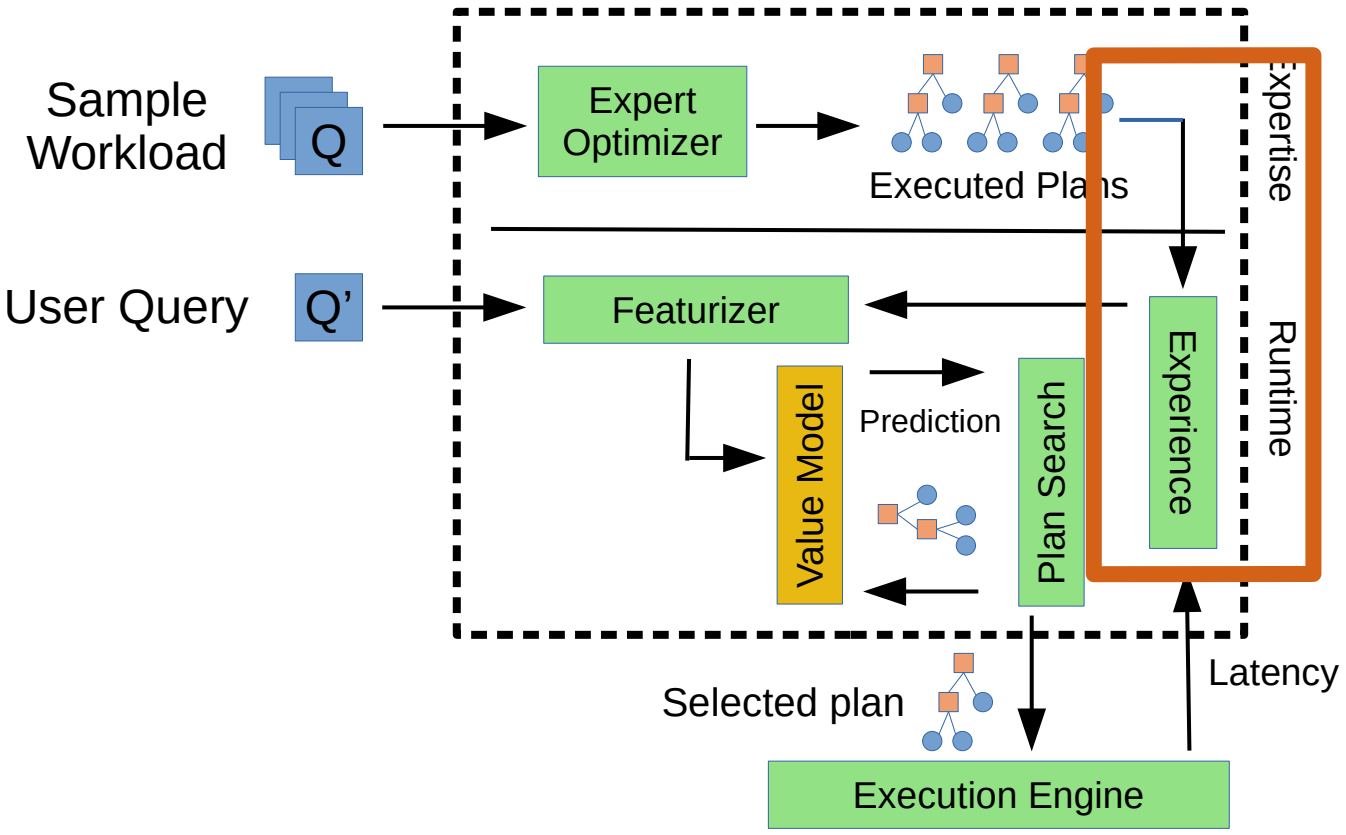

Sample Workload
Q
Expert Optimizer
Executed Plans
Expertise
User Query
Q'
Featurizer
Value Model
Prediction
Plan Search
Experience
Runtime
Selected plan
Latency
Execution Engine

Neo



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

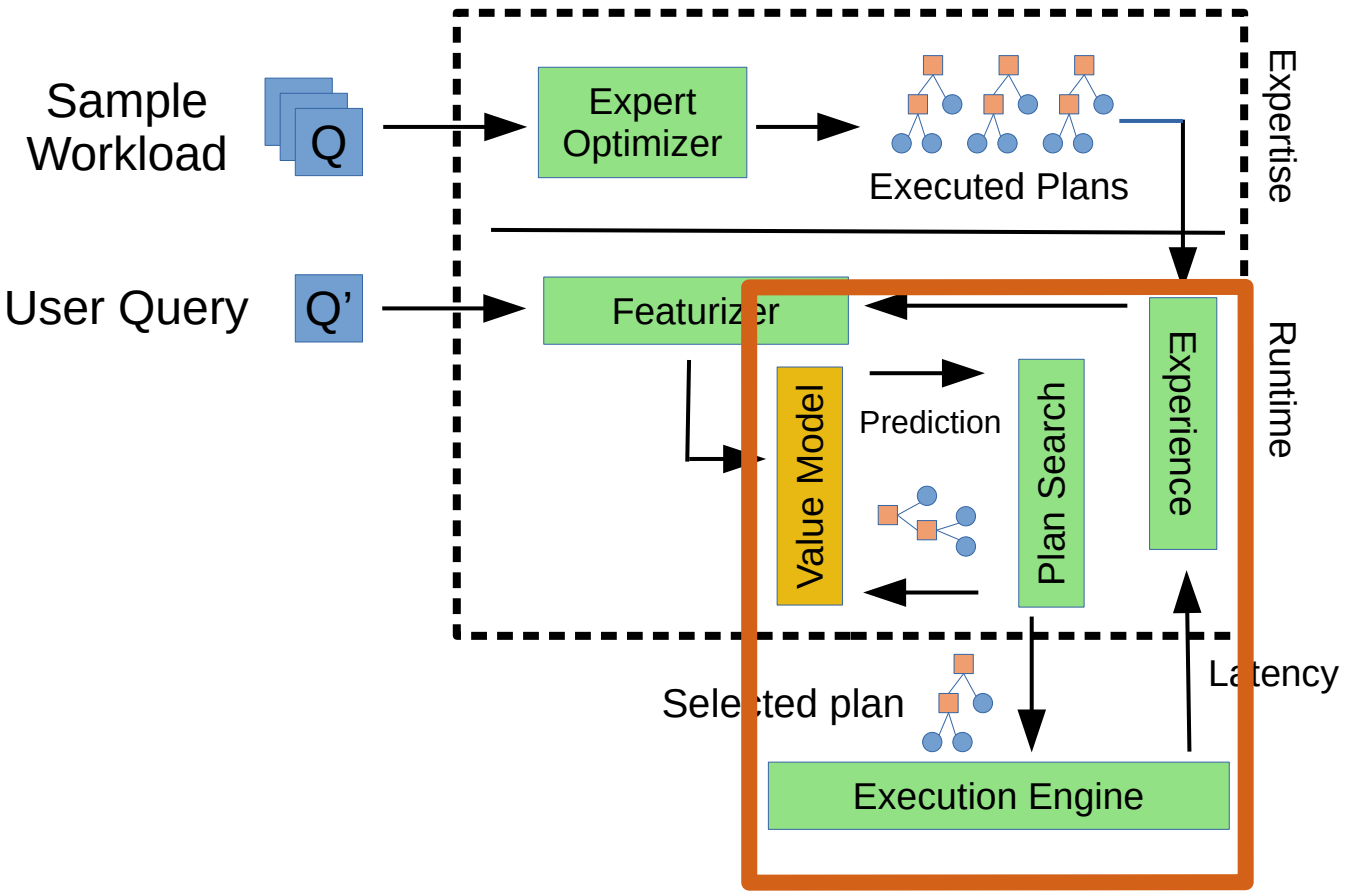
Neo



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.

This Talk

- 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 ← **Just an overview today.**
    - Watch masters. Emulate masters. Surpass masters.
  - Learn embeddings ← **See paper for details.** ☺
    - No histograms, no exception lists. Learned models.

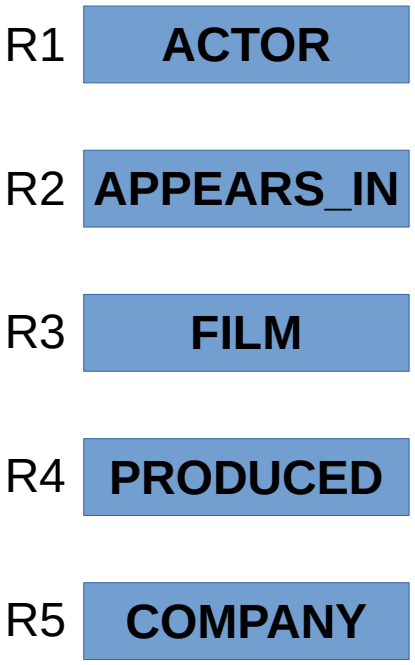
Query Optimization as an MDP

- 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

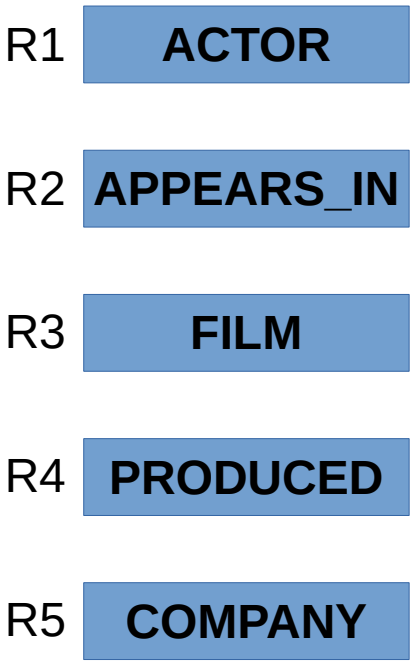


(ACTORS ⋈ APPEARS_IN ⋈ FILM ⋈ PRODUCED ⋈ COMPANY)

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



(ACTORS ⋈ APPEARS_IN ⋈ FILM ⋈ PRODUCED ⋈ COMPANY)

Query Optimization as an MDP

Actions available:

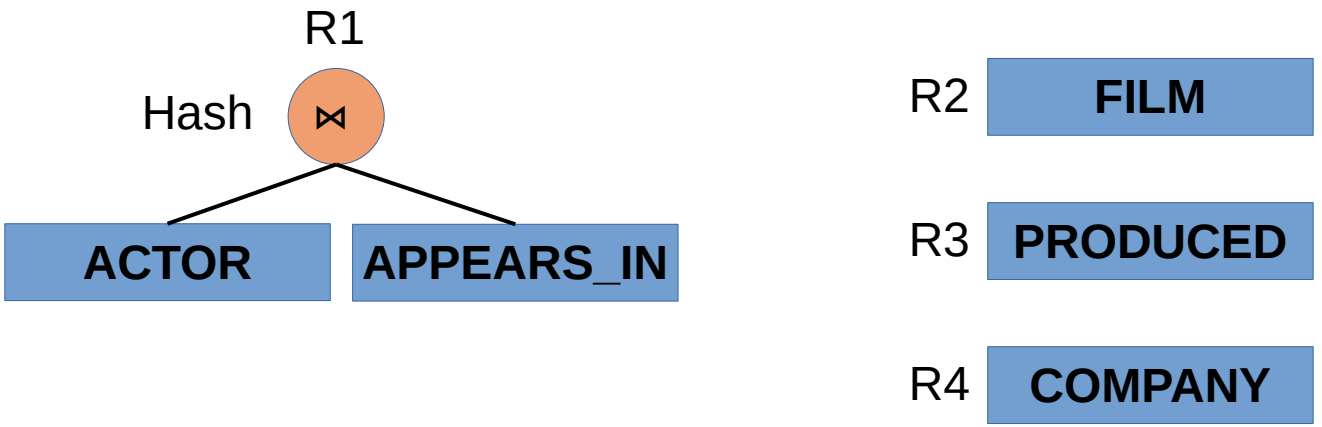
1. **(R1, R2) Hash**
2. (R1, R2) Sort
3. (R2, R3) Hash
4. (R2, R3) Sort
5. (R3, R4) Hash
6. (R3, R4) Sort
7. (R4, R5) Hash
8. (R4, R5) Sort

(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

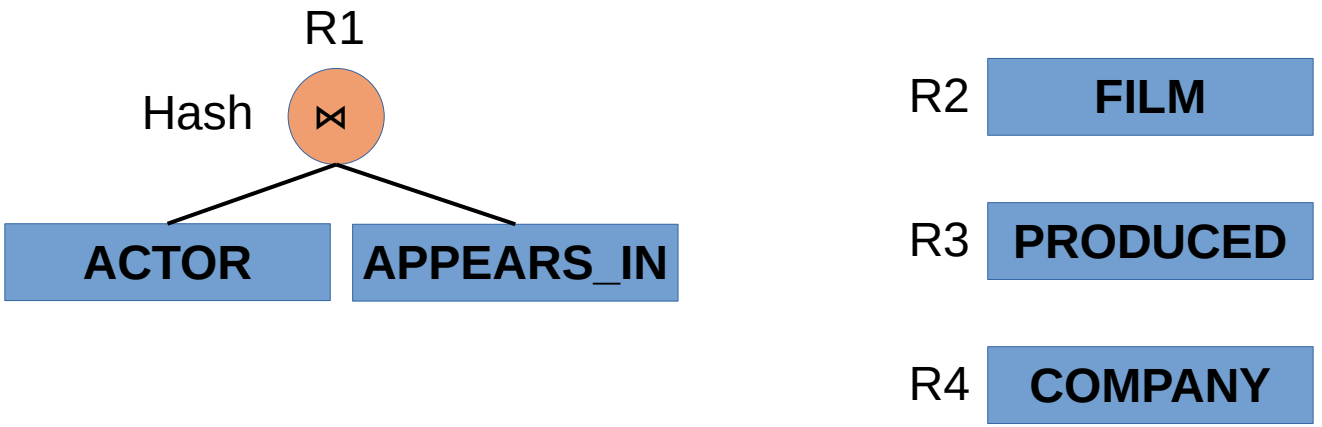


(ACTORS ⋈ APPEARS_IN ⋈ FILM ⋈ PRODUCED ⋈ COMPANY)

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

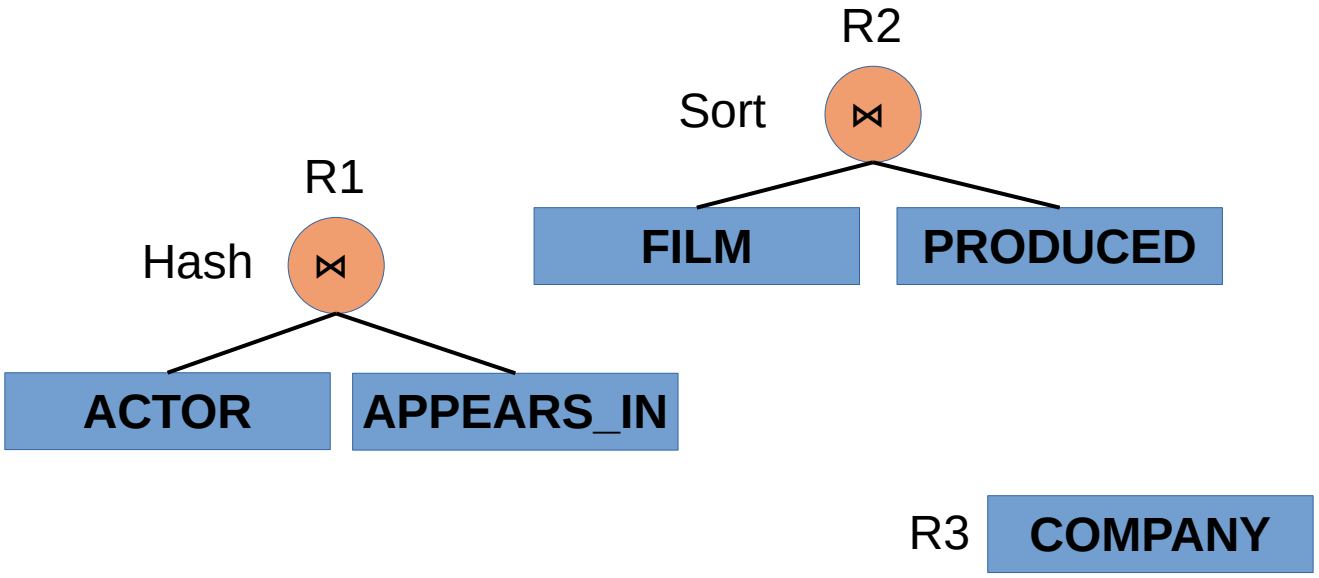


(ACTORS ⋈ APPEARS_IN ⋈ FILM ⋈ PRODUCED ⋈ COMPANY)

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

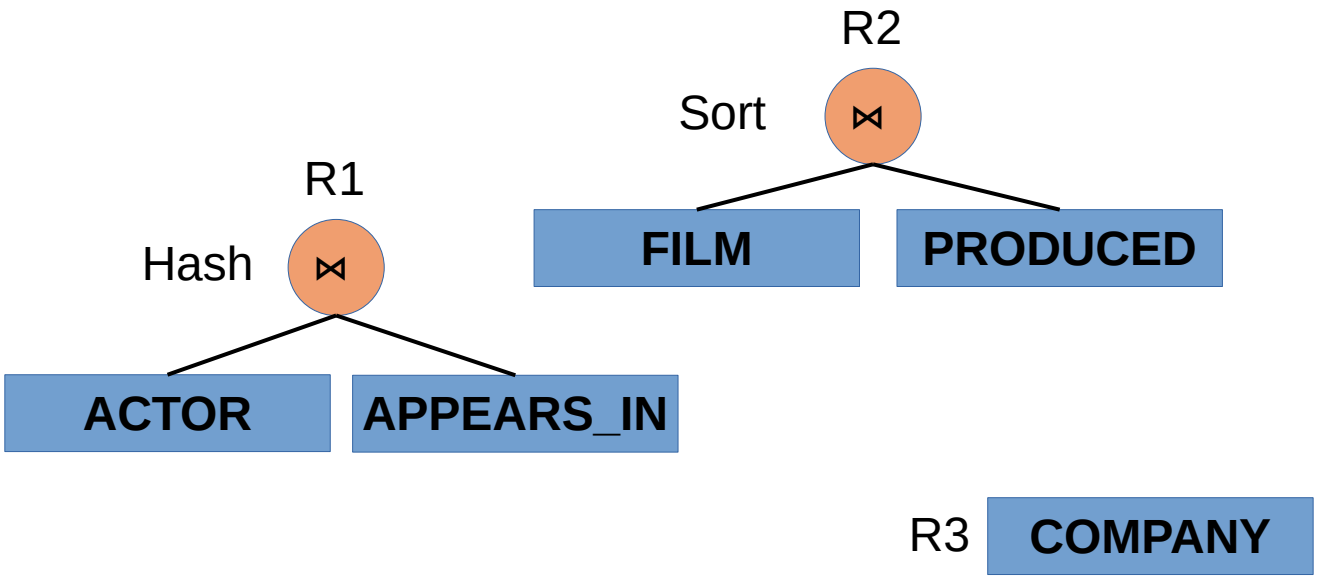


(ACTORS ⋈ APPEARS_IN ⋈ FILM ⋈ PRODUCED ⋈ COMPANY)

Query Optimization as an MDP

Actions available:

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

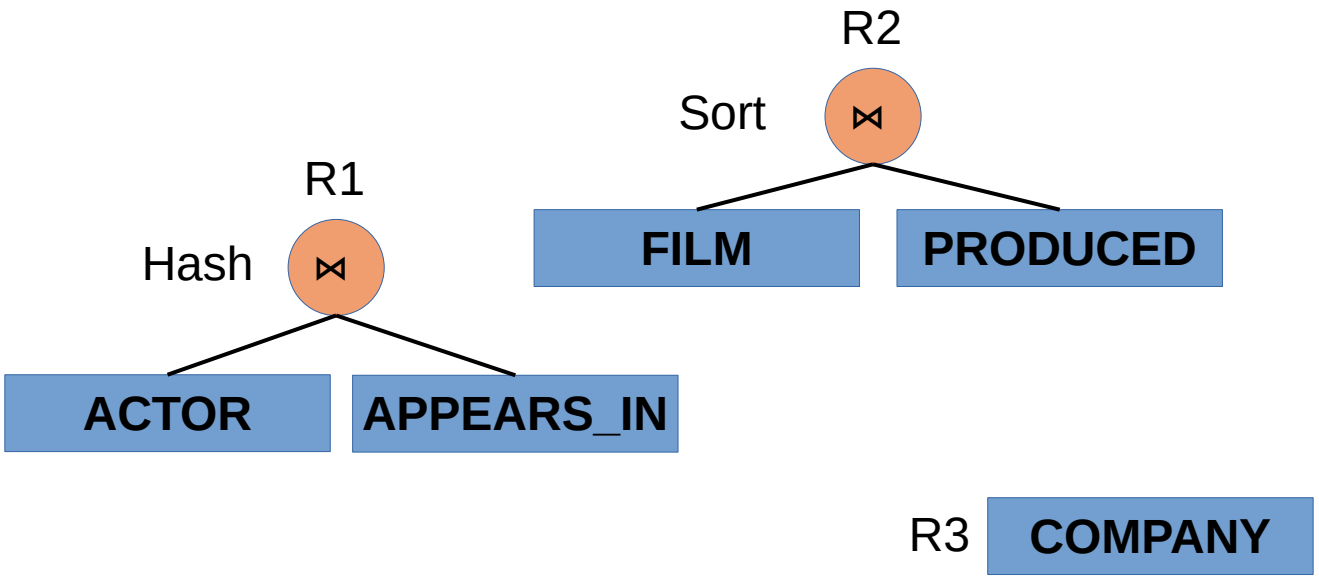


(ACTORS ⋈ APPEARS_IN ⋈ FILM ⋈ PRODUCED ⋈ COMPANY)

Query Optimization as an MDP

Actions available:

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

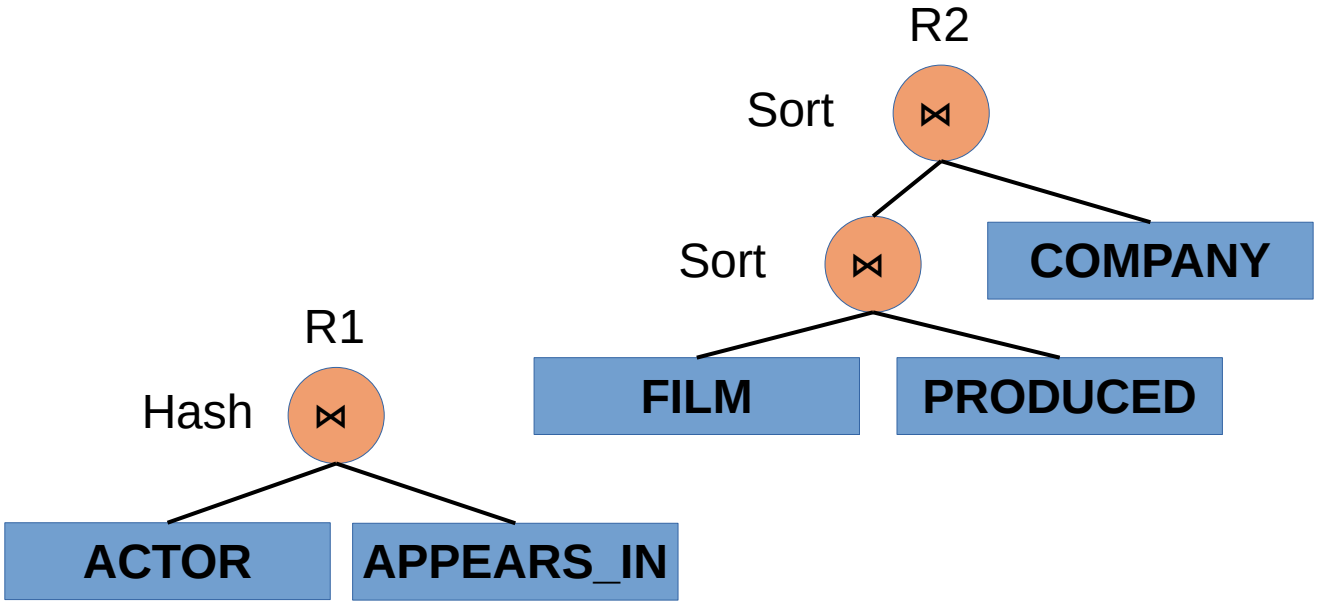


(ACTORS ⋈ APPEARS_IN ⋈ FILM ⋈ PRODUCED ⋈ COMPANY)

Query Optimization as an MDP

Actions available:

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

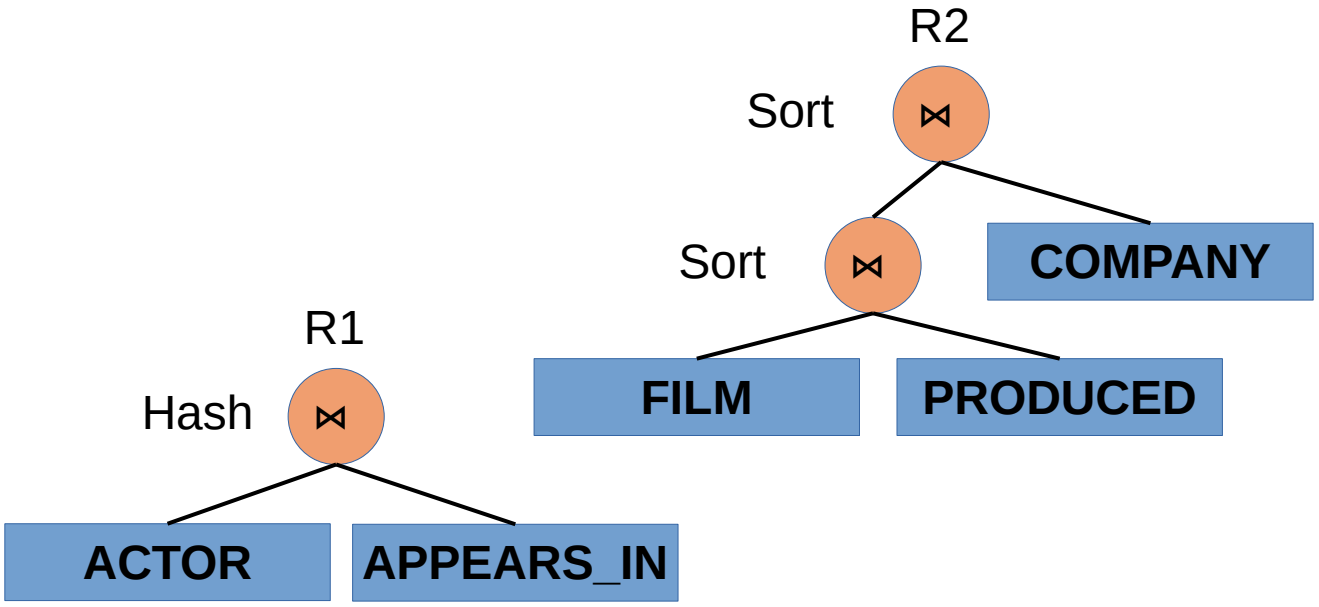


(ACTORS ⋈ APPEARS_IN ⋈ FILM ⋈ PRODUCED ⋈ COMPANY)

Query Optimization as an MDP

Actions available:

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

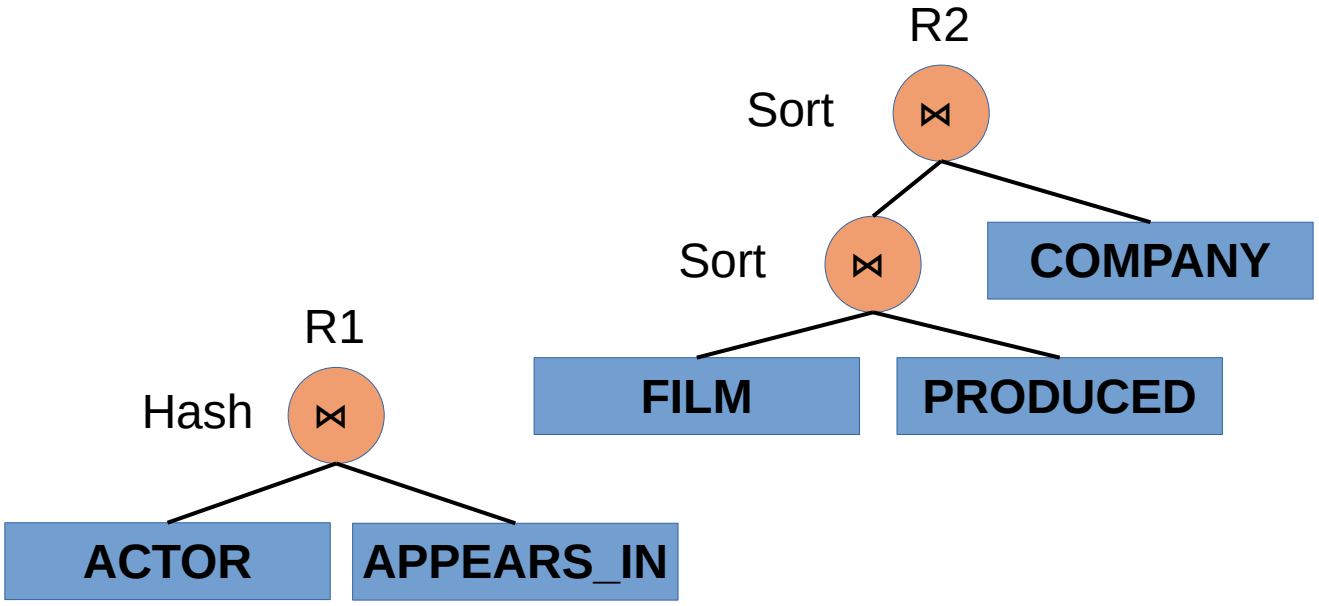


(ACTORS ⋈ APPEARS_IN ⋈ FILM ⋈ PRODUCED ⋈ COMPANY)

Query Optimization as an MDP

Actions available:

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

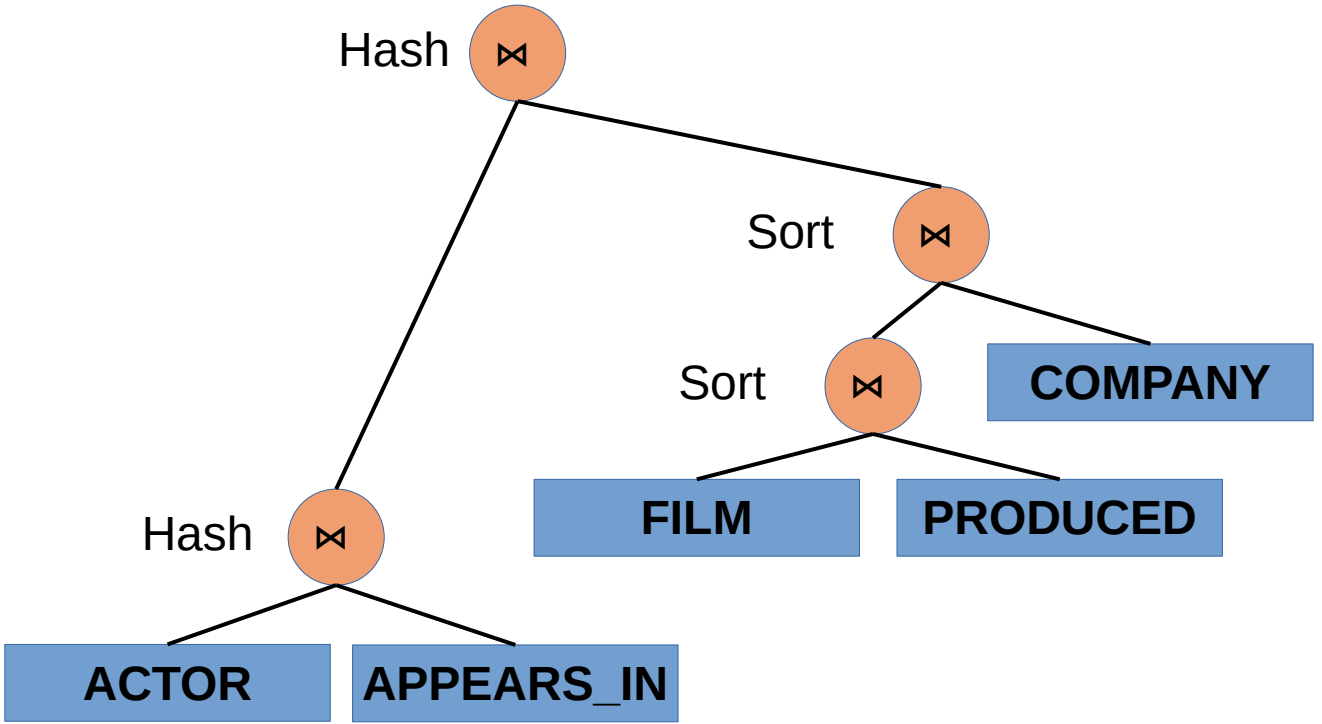


(ACTORS ⋈ APPEARS_IN ⋈ FILM ⋈ PRODUCED ⋈ COMPANY)

Query Optimization as an MDP

Actions available:

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



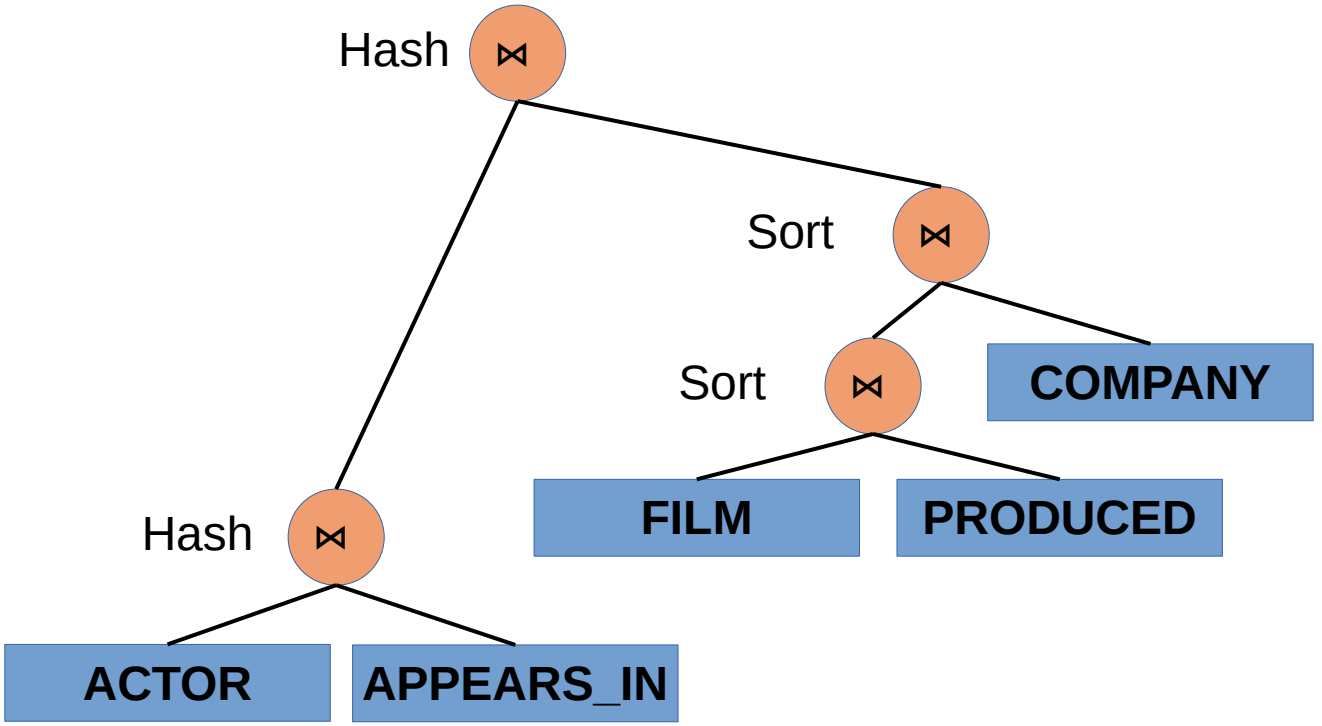
(ACTORS ⋈ APPEARS_IN ⋈ FILM ⋈ PRODUCED ⋈ COMPANY)

Query Optimization as an MDP

Every previous state had reward 0

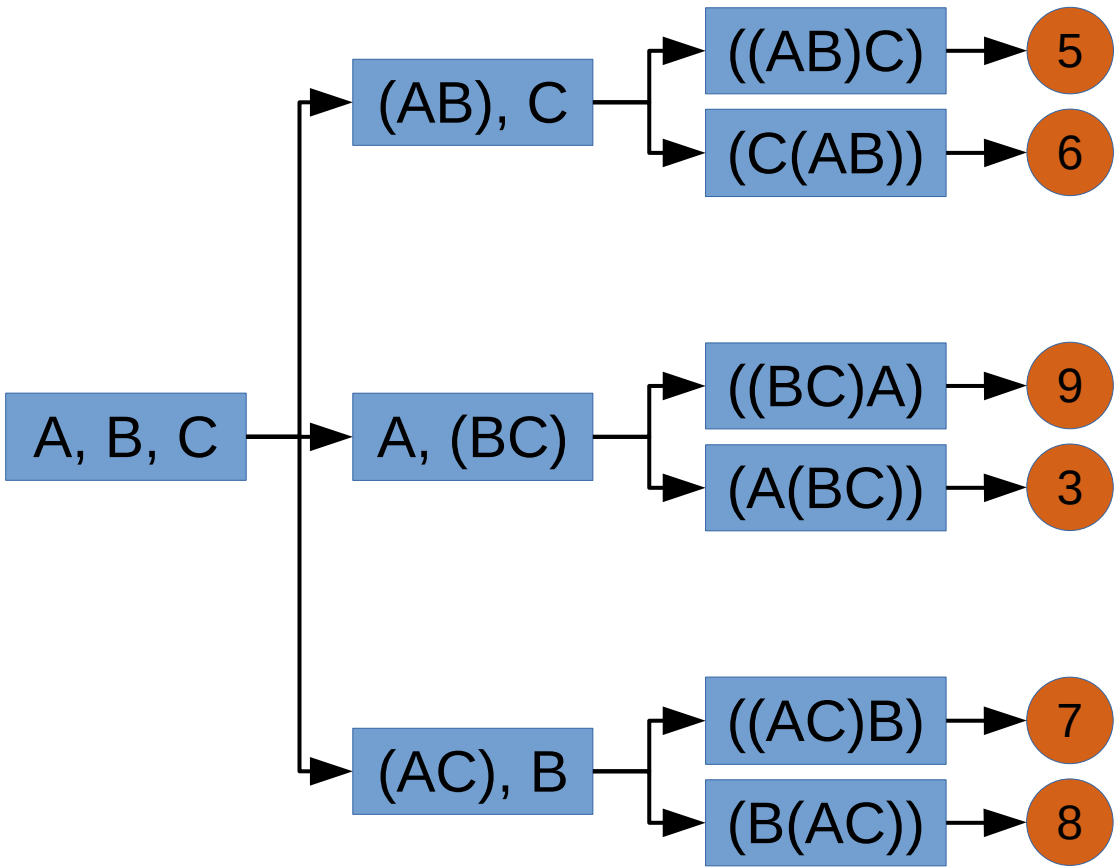
Now, we execute the program and record the latency.

Reward is -latency.

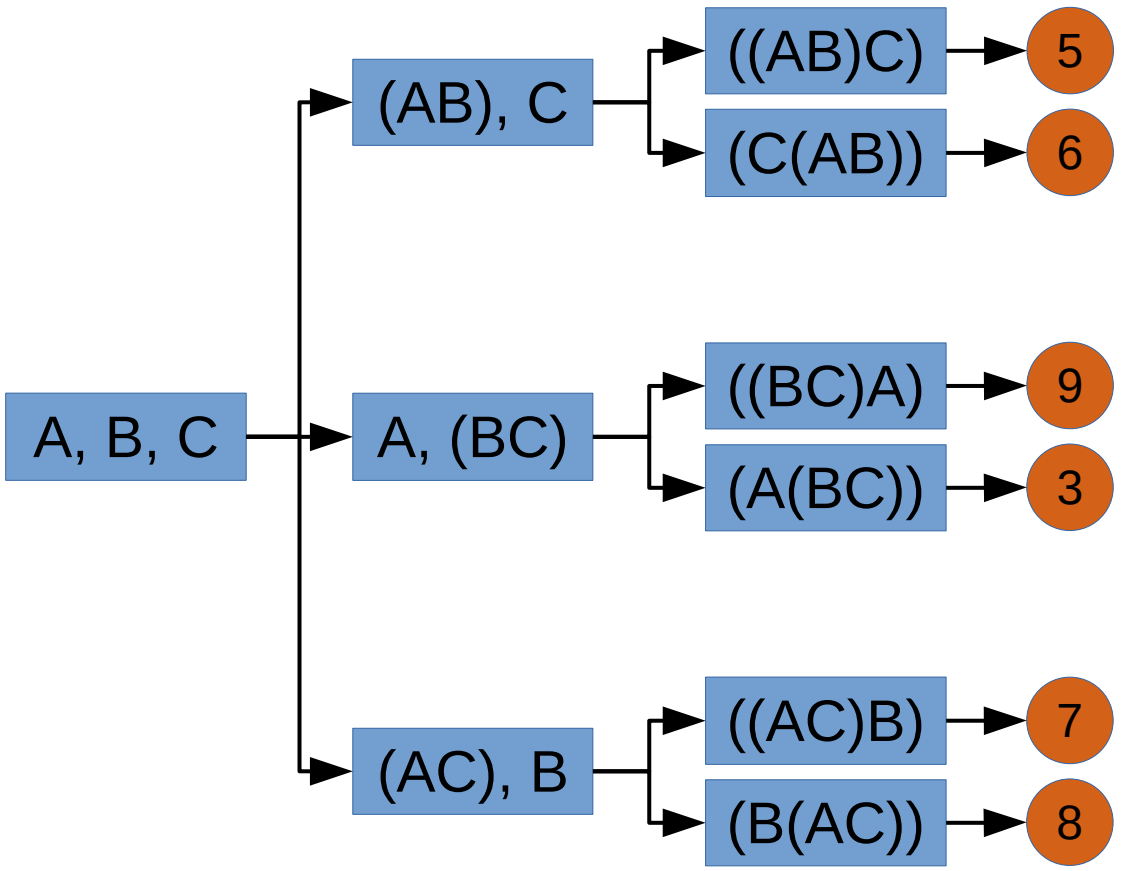


(ACTORS ⋈ APPEARS_IN ⋈ FILM ⋈ PRODUCED ⋈ COMPANY)

Deep Reinforcement Learning



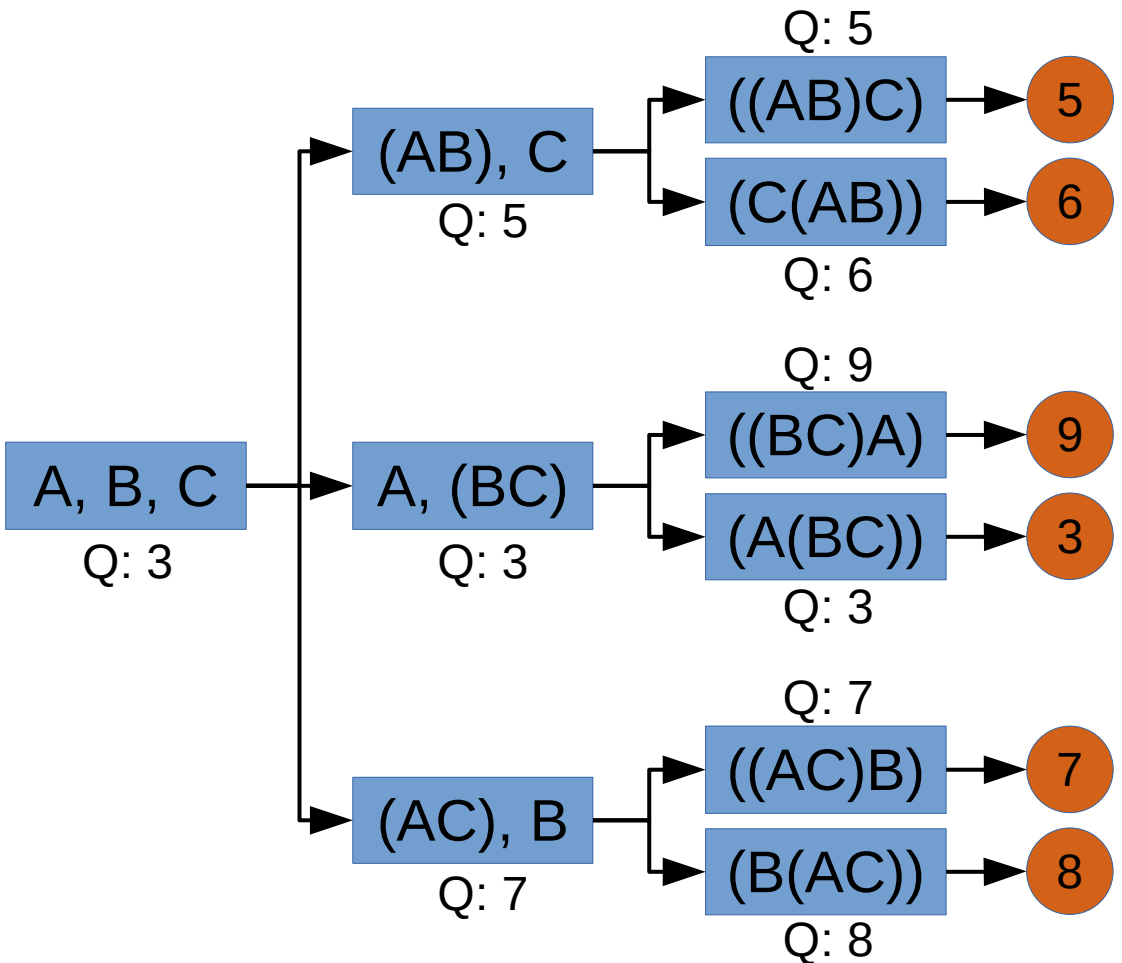
Deep Reinforcement Learning



Deep reinforcement learning

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

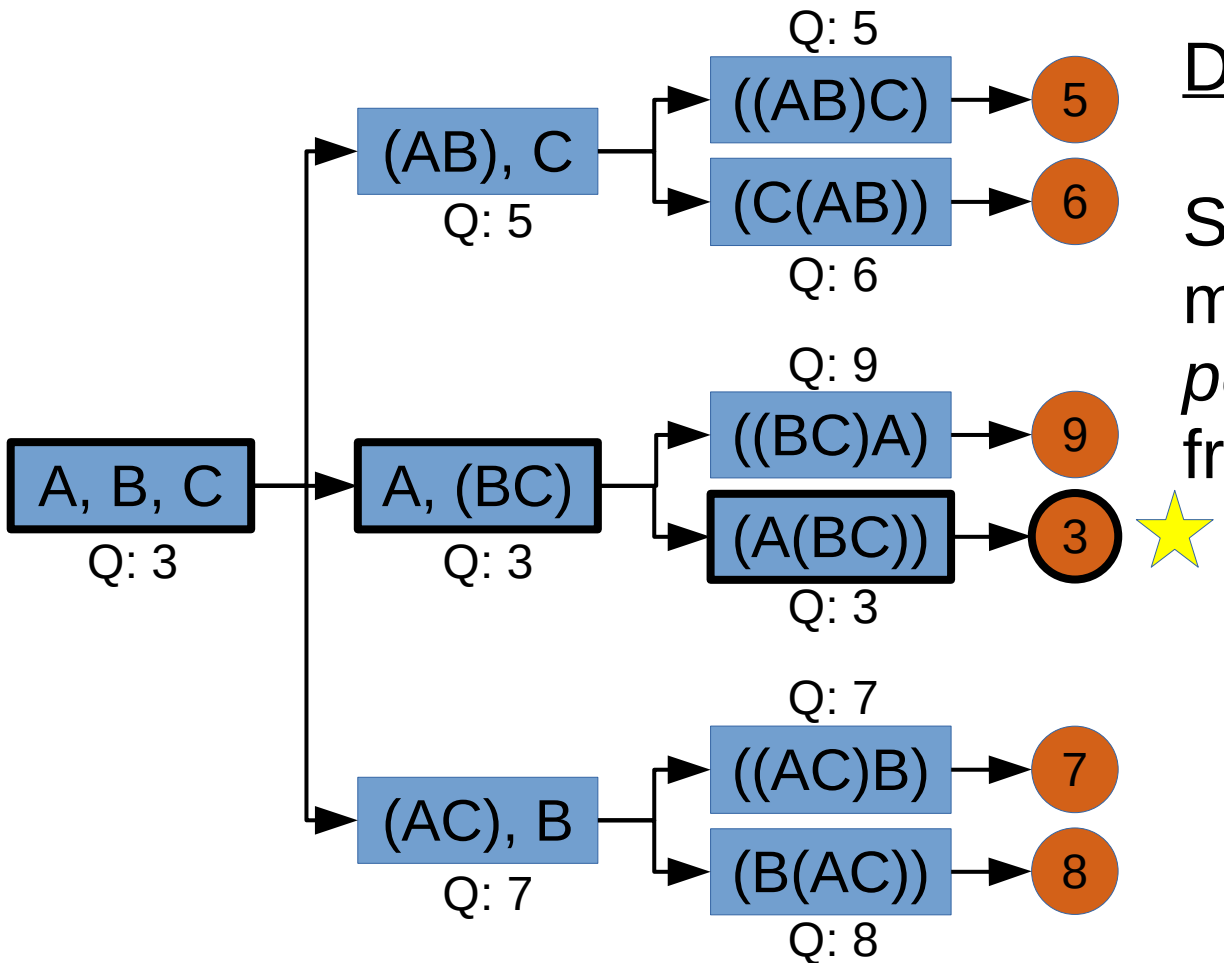
Deep Reinforcement Learning



Deep reinforcement learning

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

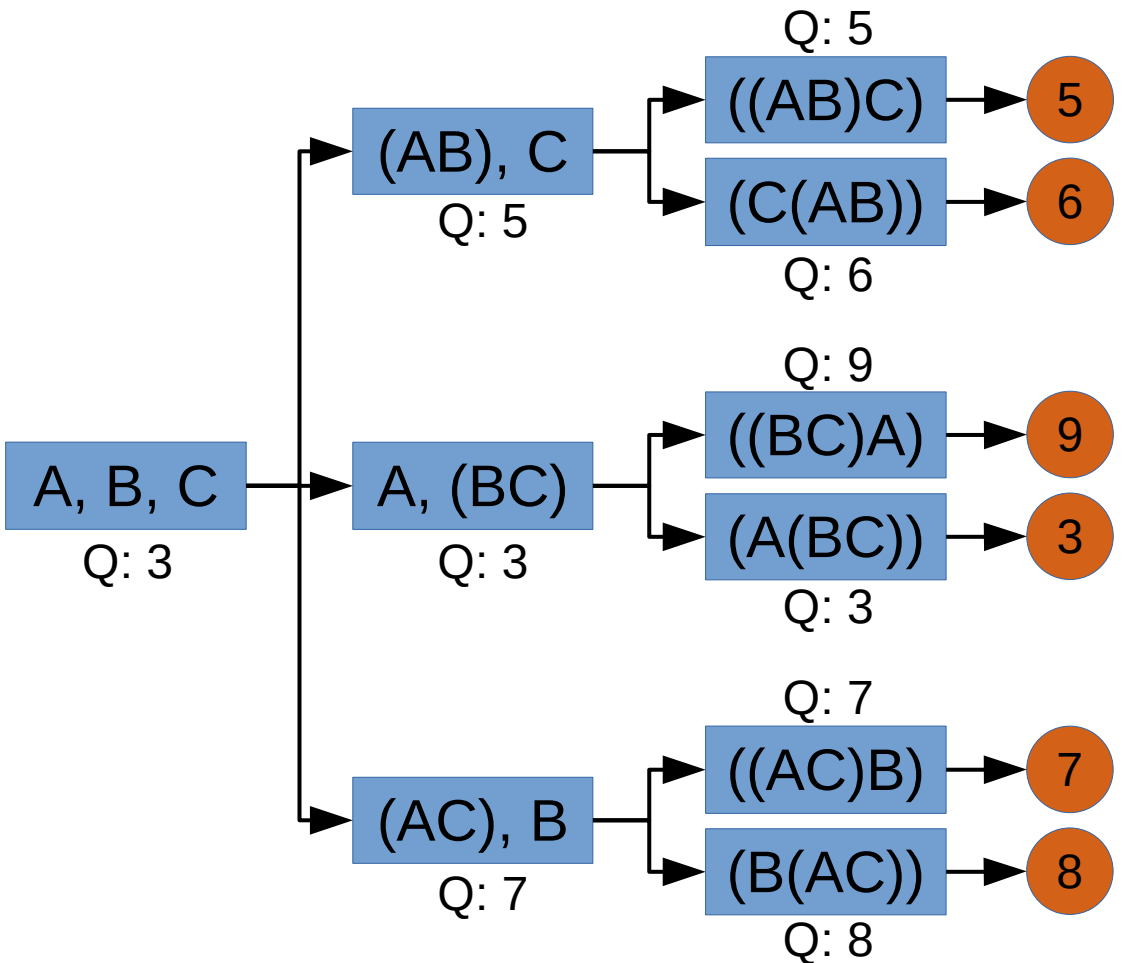
Deep Reinforcement Learning



<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



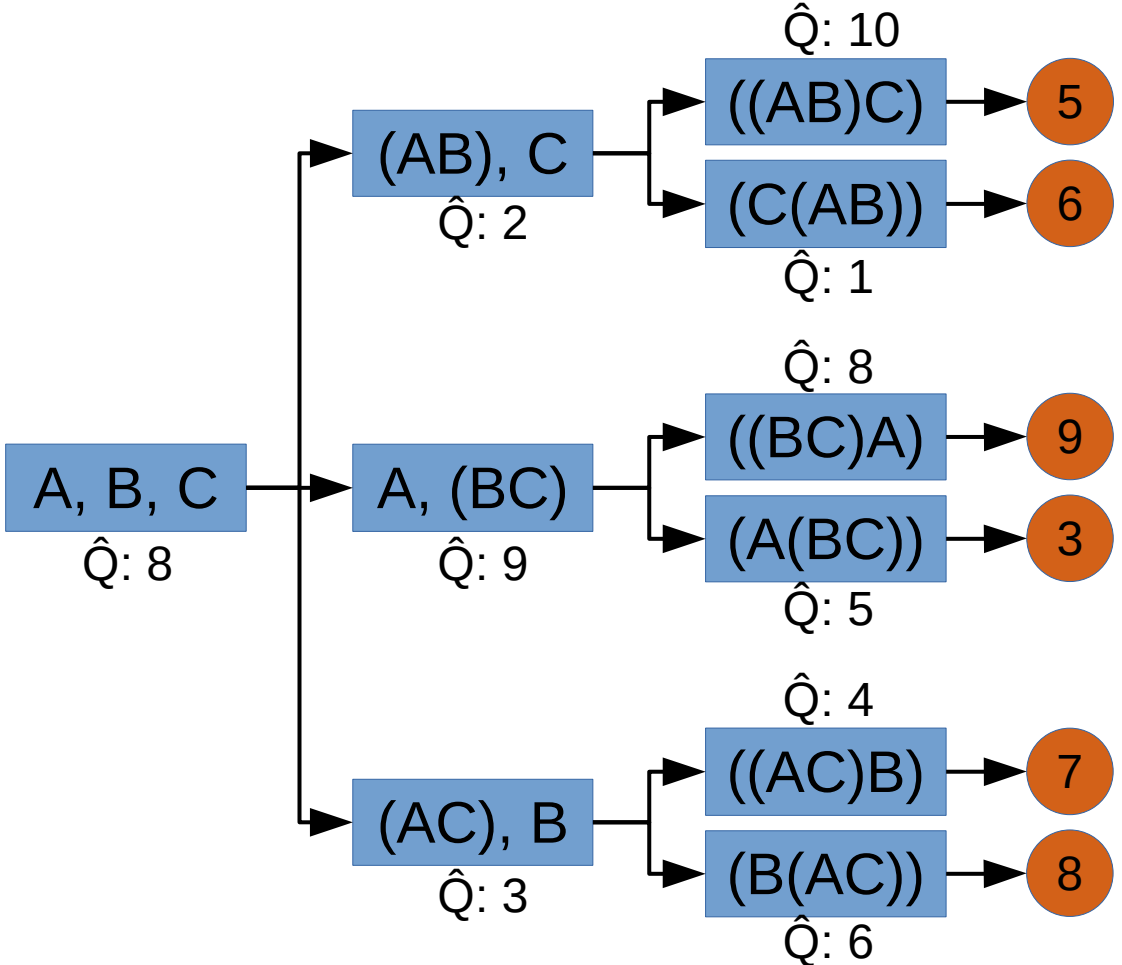
Deep reinforcement learning

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, $\hat{Q}$

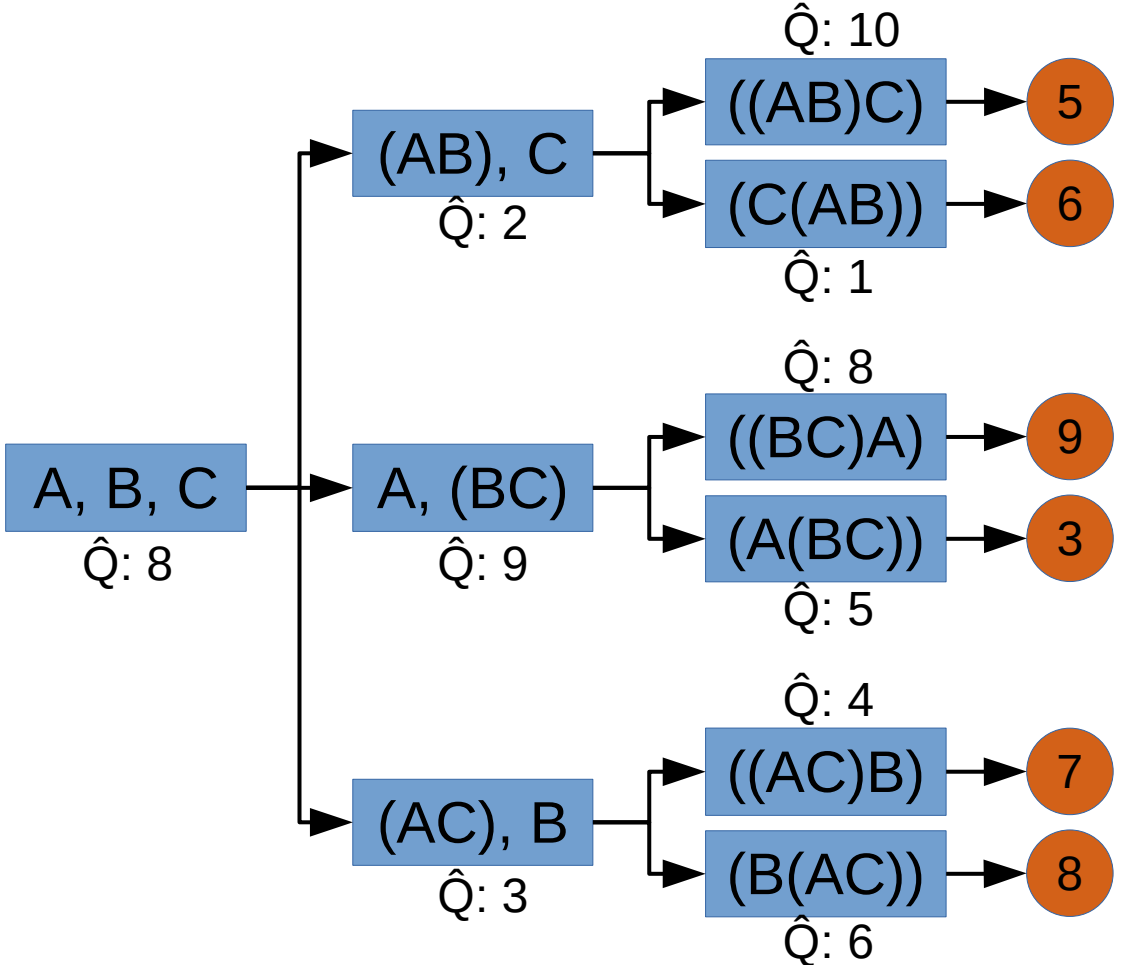
Deep Reinforcement Learning



<u>Deep reinforcement learning</u>

Approximation, $\hat{Q}$.
1) initialize $\hat{Q}_0$

Deep Reinforcement Learning



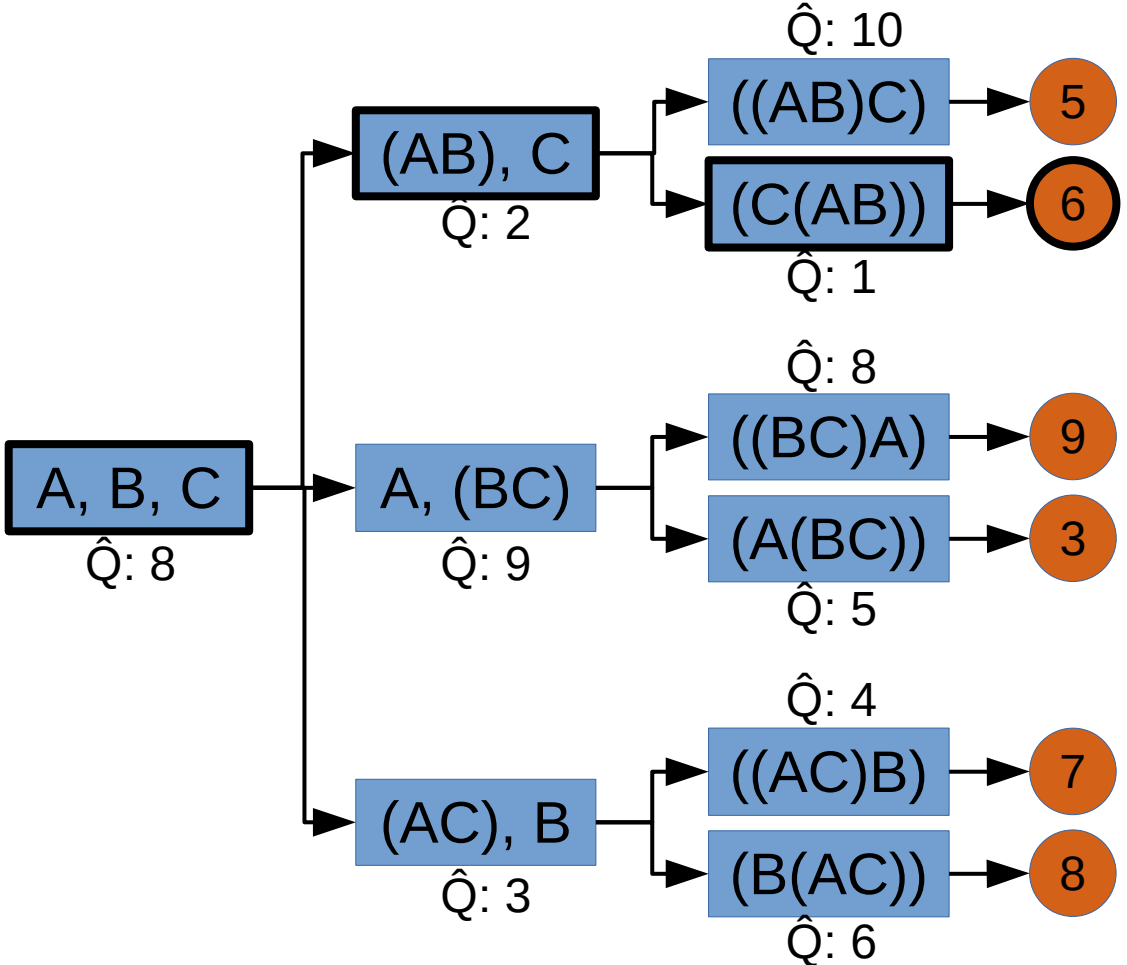
<u>Deep reinforcement learning</u>

Approximation, $\hat{Q}$.

1) initialize $\hat{Q}_0$

2) play 1 round with $\hat{Q}_0$

Deep Reinforcement Learning



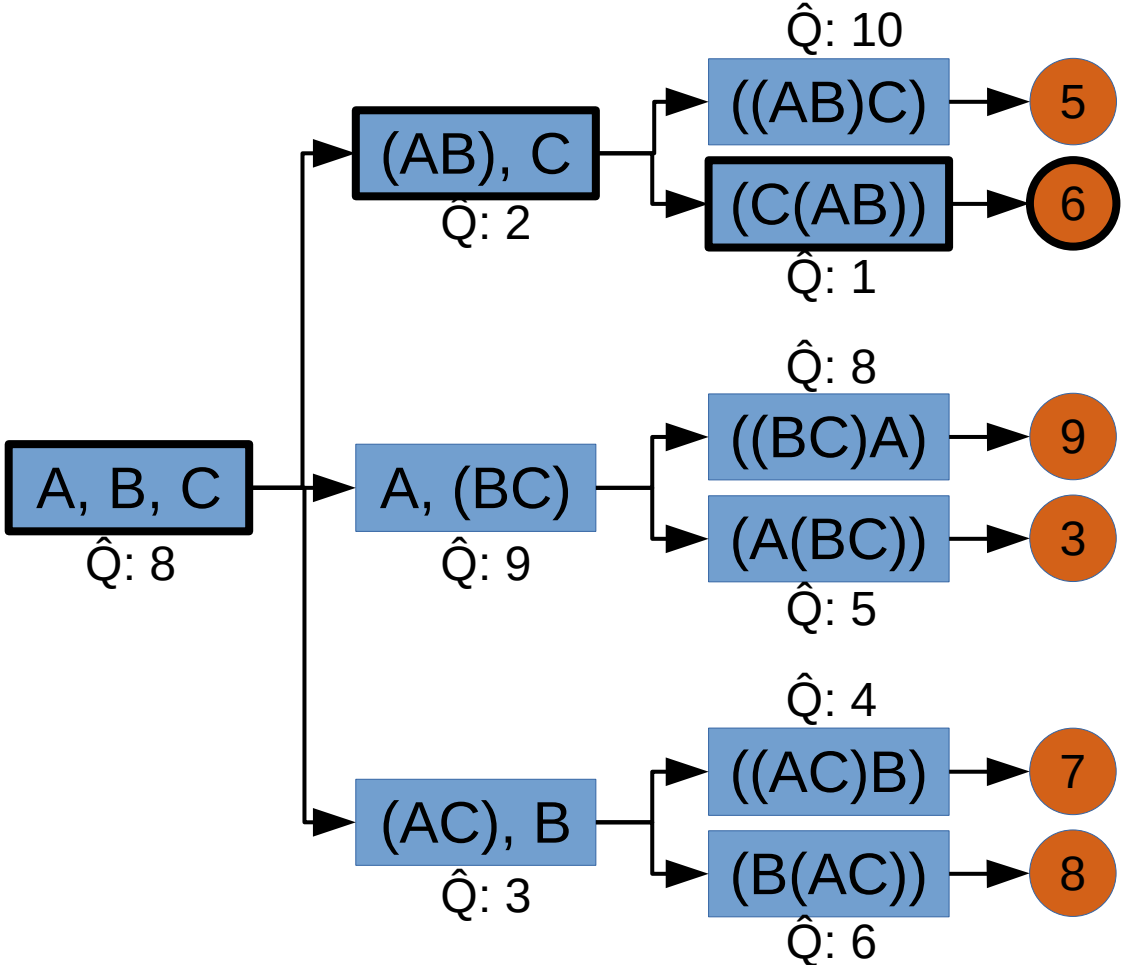
Deep reinforcement learning

Approximation, $\hat{Q}$.

1) initialize $\hat{Q}_0$

2) play 1 round with $\hat{Q}_0$

Deep Reinforcement Learning



Deep reinforcement learning

Approximation, $\hat{Q}$.

1) initialize $\hat{Q}_0$

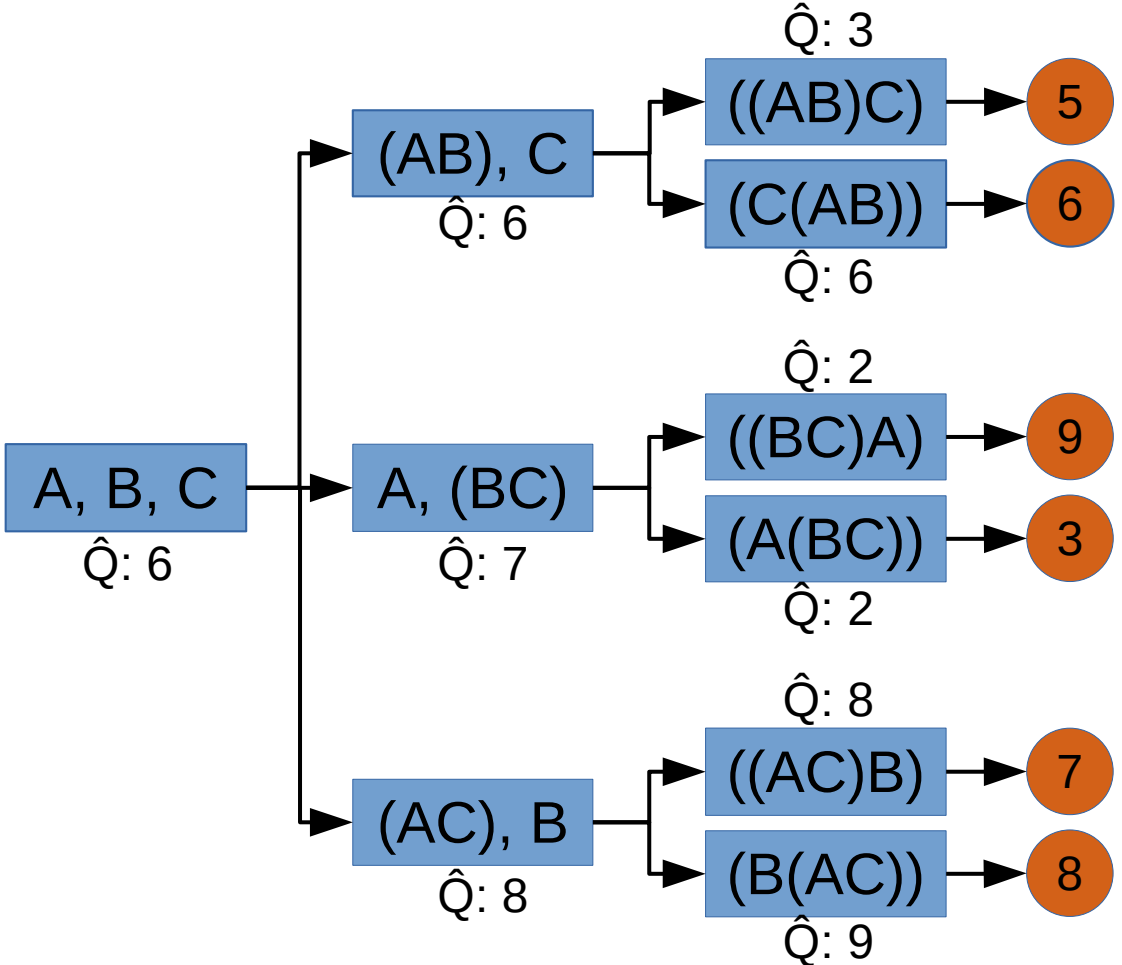
2) play 1 round with $\hat{Q}_0$

3) use obs. to train $\hat{Q}_1$

eg., $\hat{Q}_1((C(AB)) = 6$

$\hat{Q}_1(A, (BC)) = 6$

Deep Reinforcement Learning



<u>Deep reinforcement learning</u>

Approximation, $\hat{Q}$.

1) initialize $\hat{Q}_0$

2) play 1 round with $\hat{Q}_0$

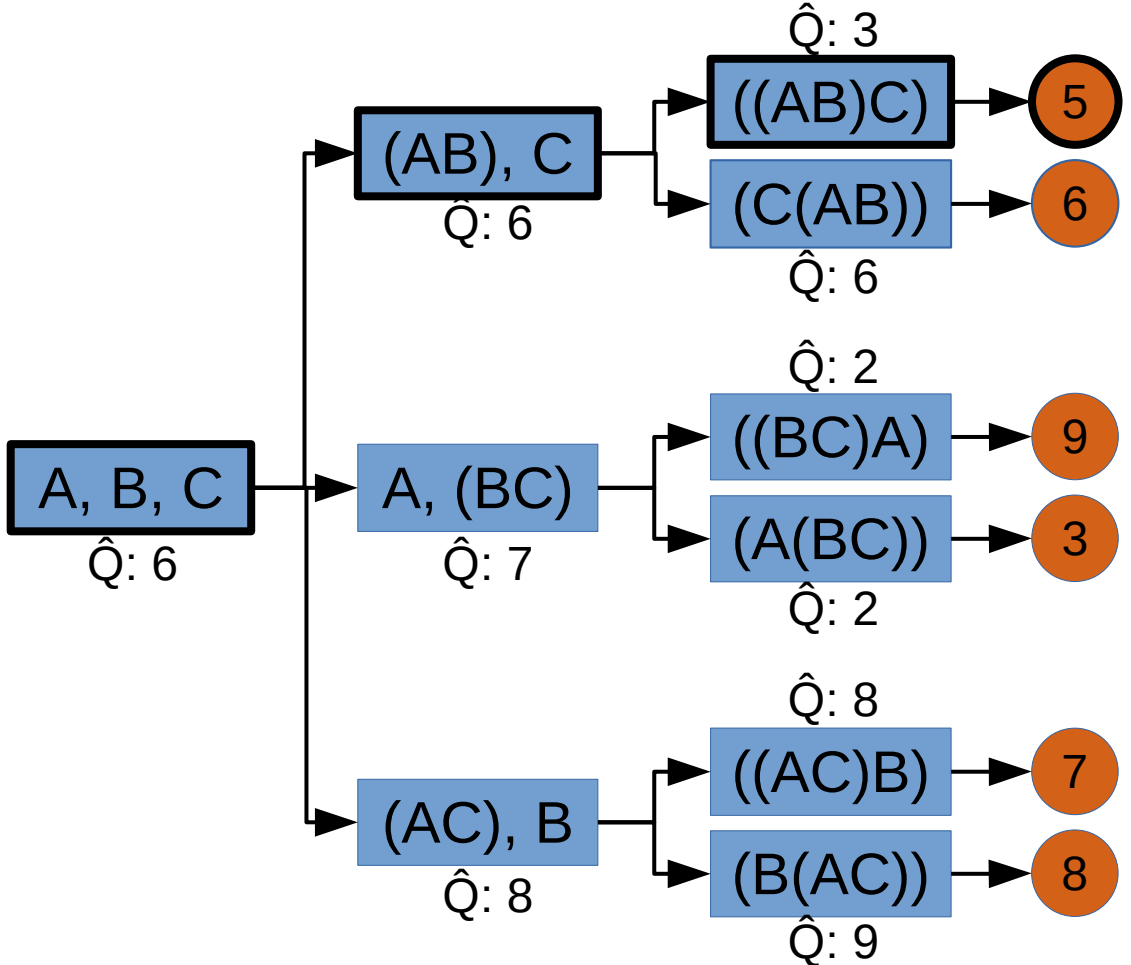
3) use obs. to train $\hat{Q}_1$

eg., $\hat{Q}_1((C(AB)) = 6$

$\hat{Q}_1(A, (BC)) = 6$

4) play 1 round with $\hat{Q}_1$

Deep Reinforcement Learning



<u>Deep reinforcement learning</u>

Approximation, $\hat{Q}$.

1) initialize $\hat{Q}_0$

2) play 1 round with $\hat{Q}_0$

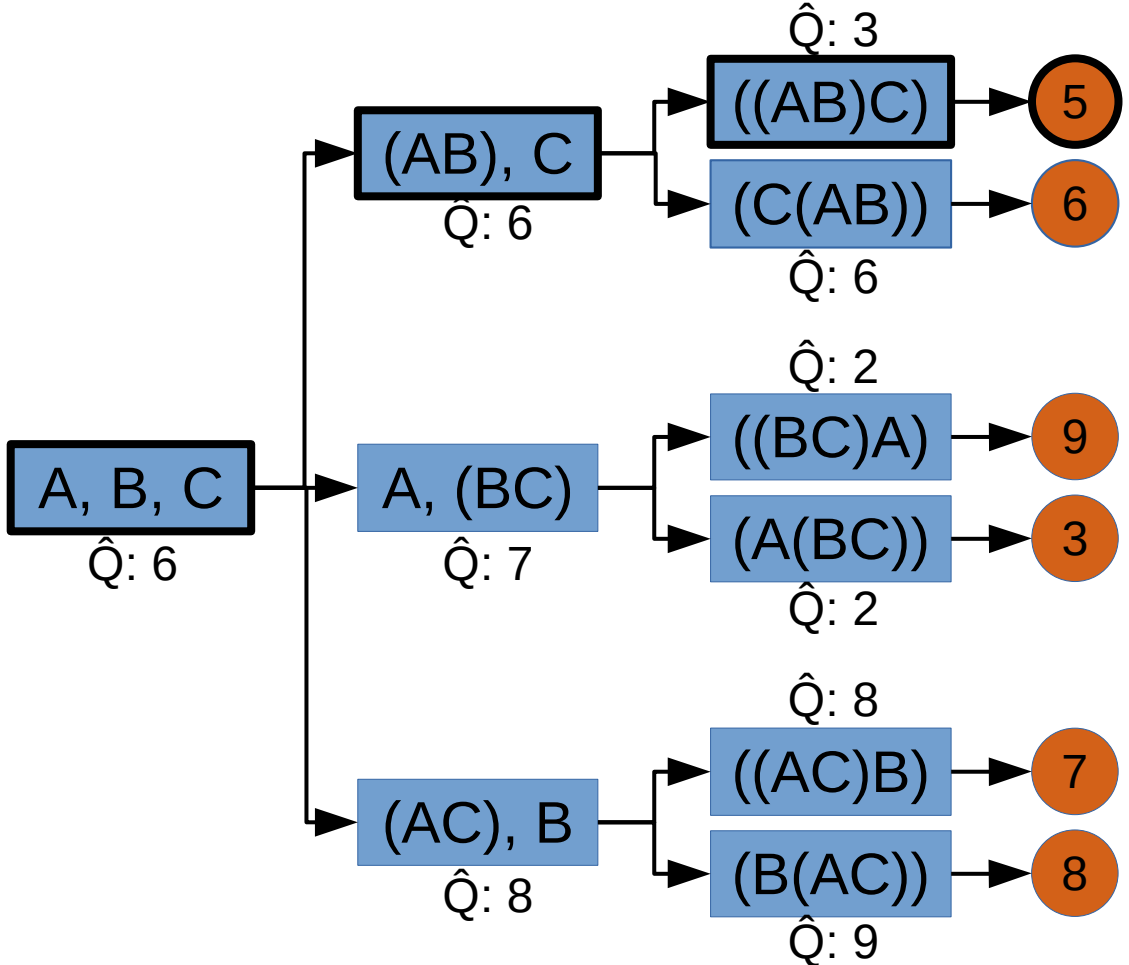
3) use obs. to train $\hat{Q}_1$

eg., $\hat{Q}_1((C(AB)) = 6$

$\hat{Q}_1(A, (BC)) = 6$

4) play 1 round with $\hat{Q}_1$

Deep Reinforcement Learning



Deep reinforcement learning

Approximation, $\hat{Q}$.

1) initialize $\hat{Q}_0$
2) play 1 round with $\hat{Q}_0$
3) use obs. to train $\hat{Q}_1$

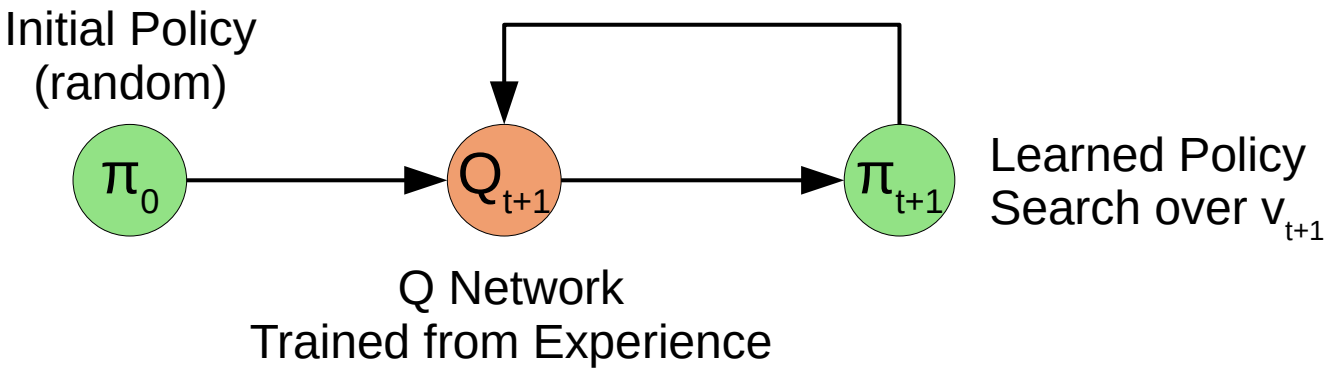
eg., $\hat{Q}_1((C(AB)) = 6$

$\hat{Q}_1(A, (BC)) = 6$

4) play 1 round with $\hat{Q}_1$
5) repeat

Deep Reinforcement Learning

- Value iteration

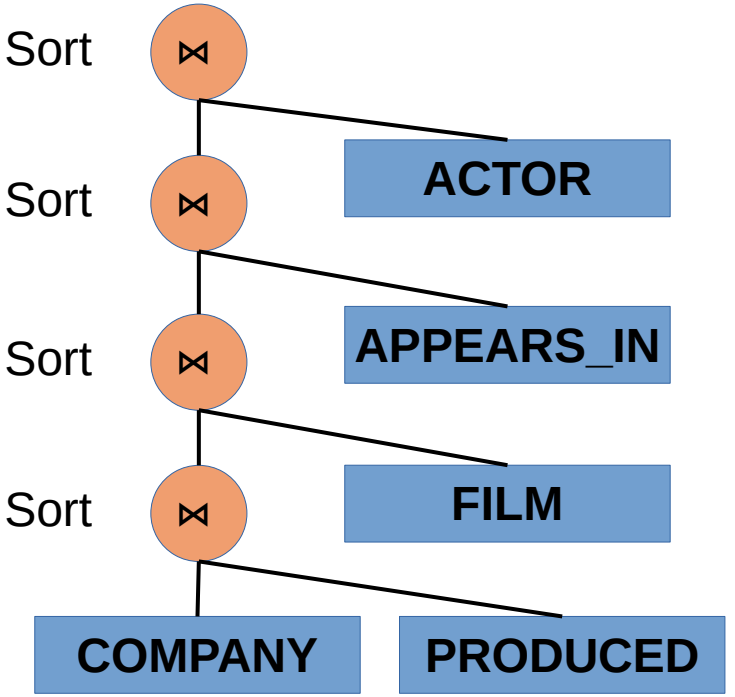


Inductive Bias

- 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

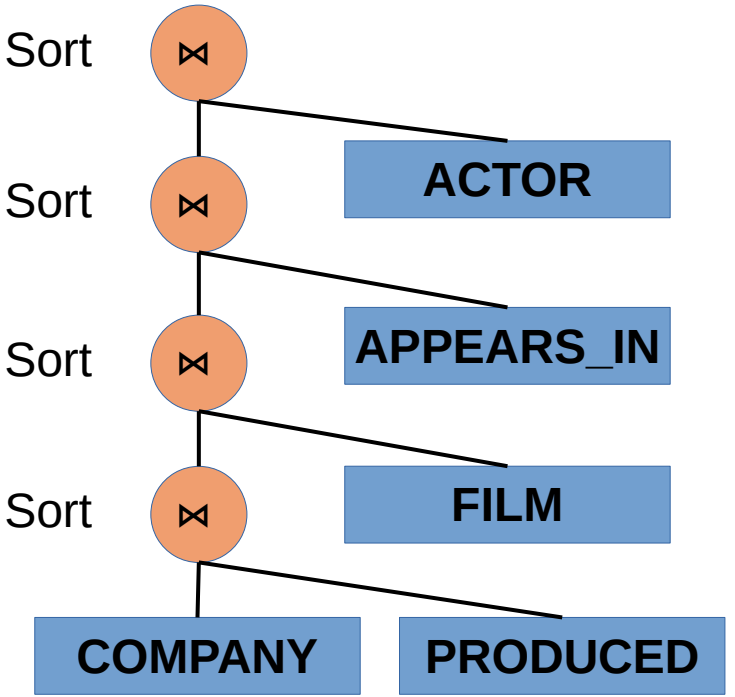
Tree Convolution

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



Tree Convolution

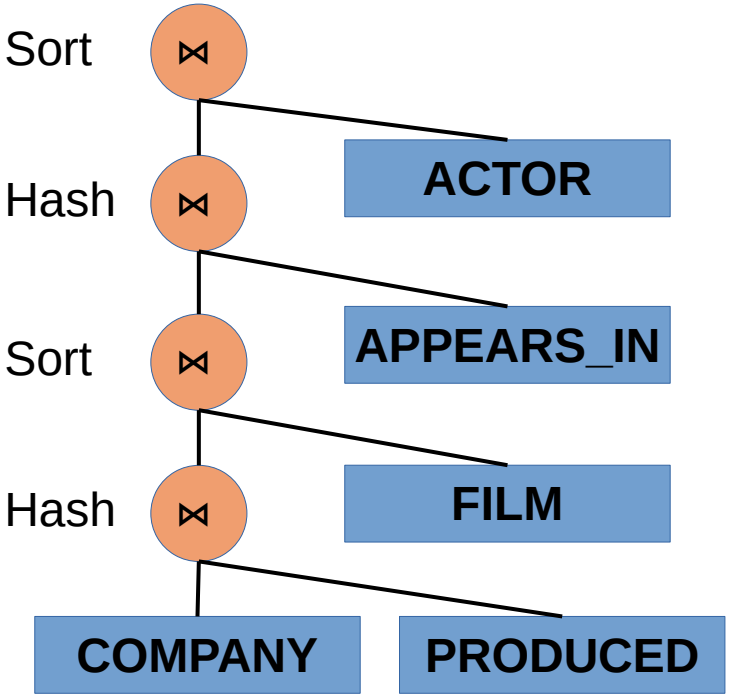
- How do we come up with a good inductive bias for query plans?



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

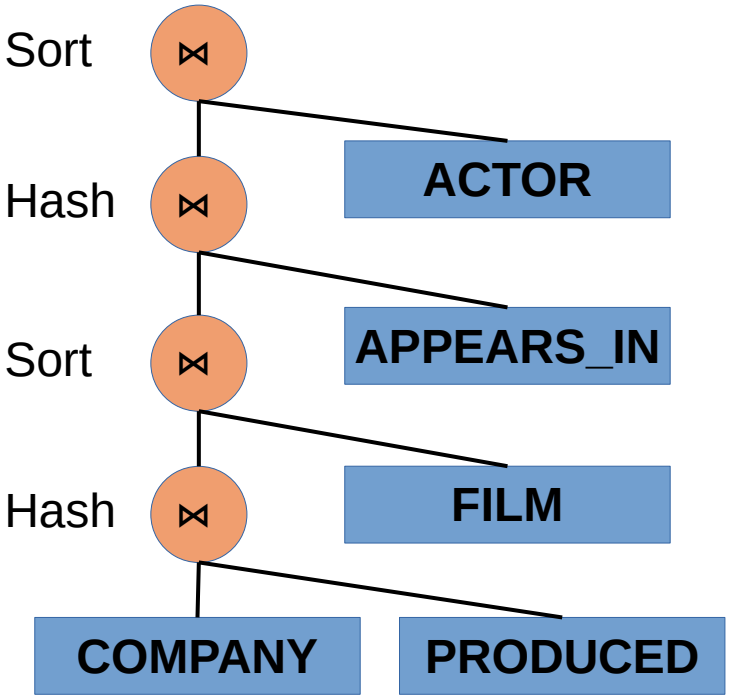
Tree Convolution

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



Tree Convolution

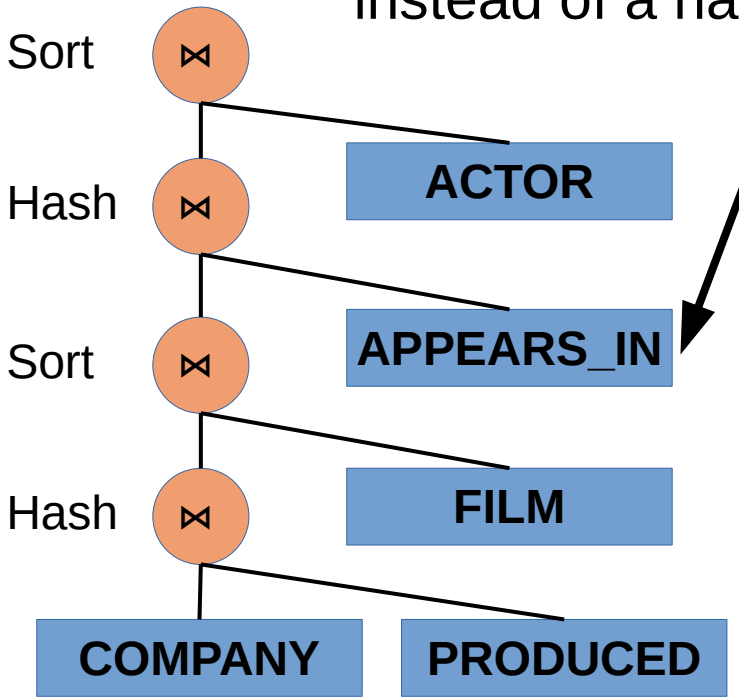
- How do we come up with a good inductive bias for query plans?



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

Tree Convolution

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

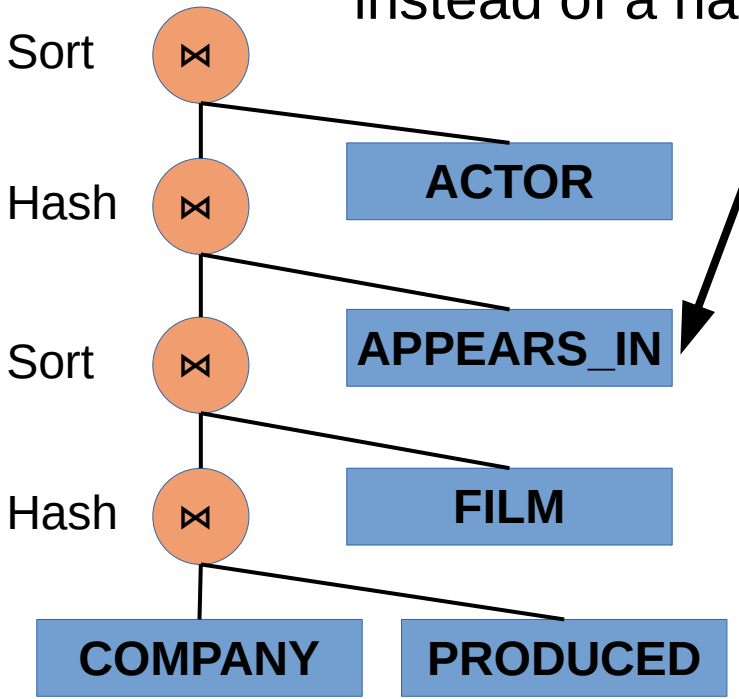


"APPEARS_IN" is presorted on disk – should use a sort instead of a hash.

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

Tree Convolution

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

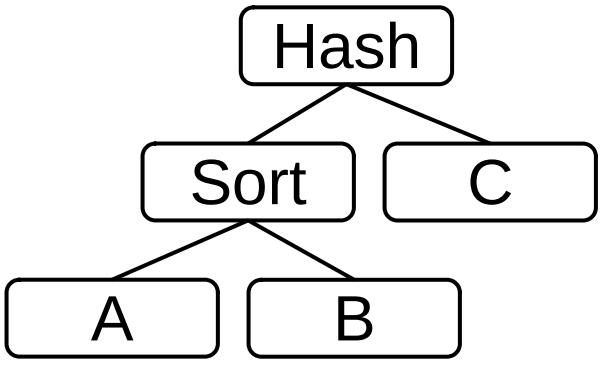


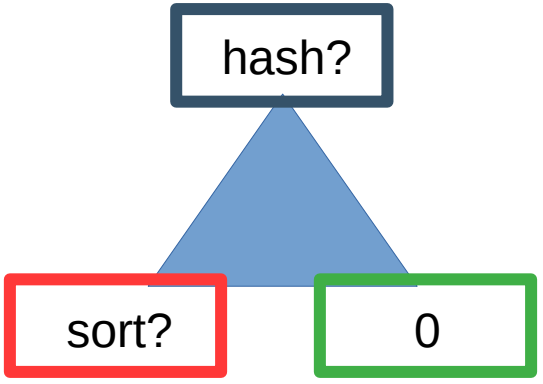
"APPEARS_IN" is presorted on disk – should use a sort instead of a hash.

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

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

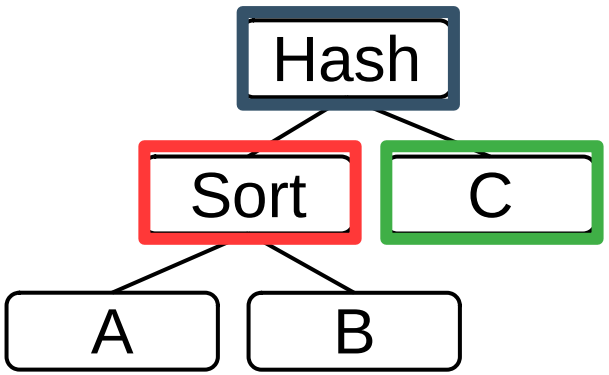
Tree Convolution

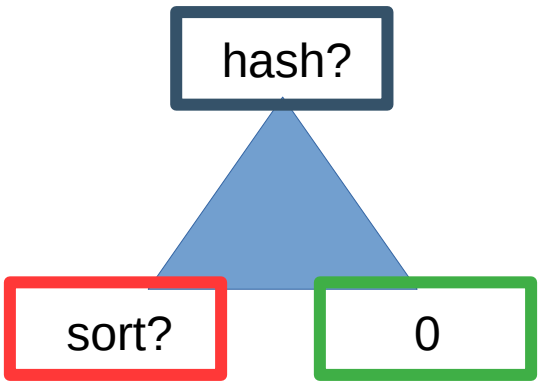




Detects a hash on top of a sort

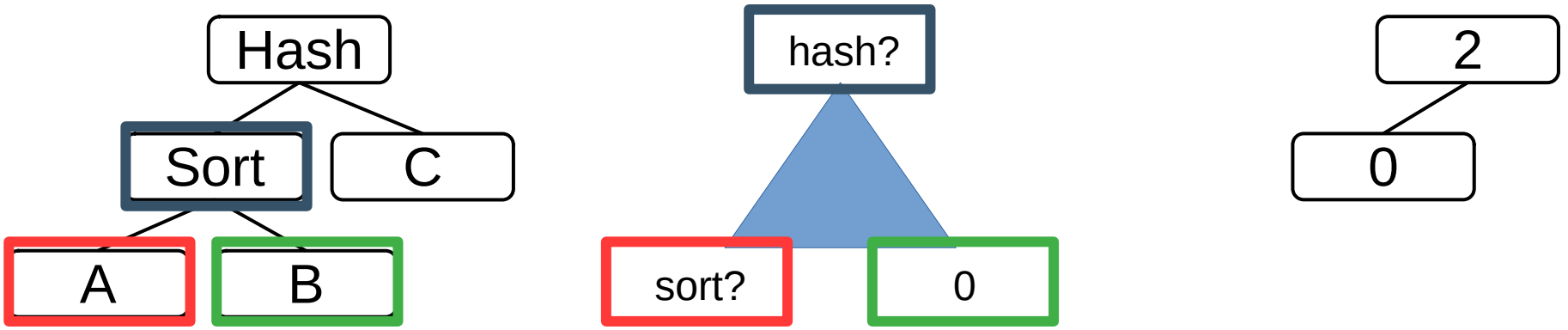
Tree Convolution





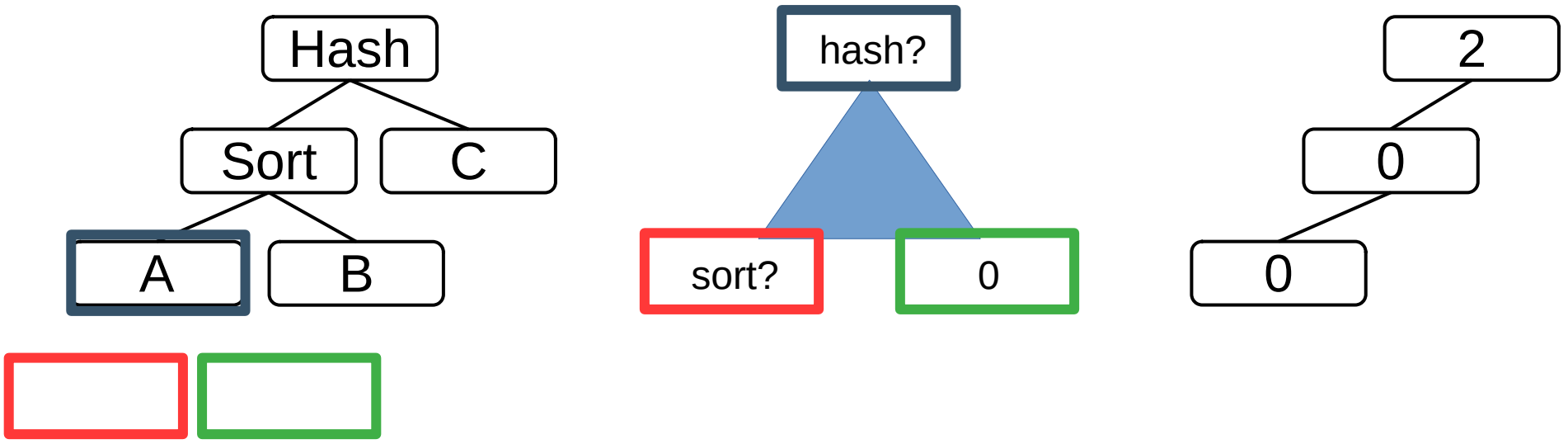
Detects a hash on top of a sort

Tree Convolution



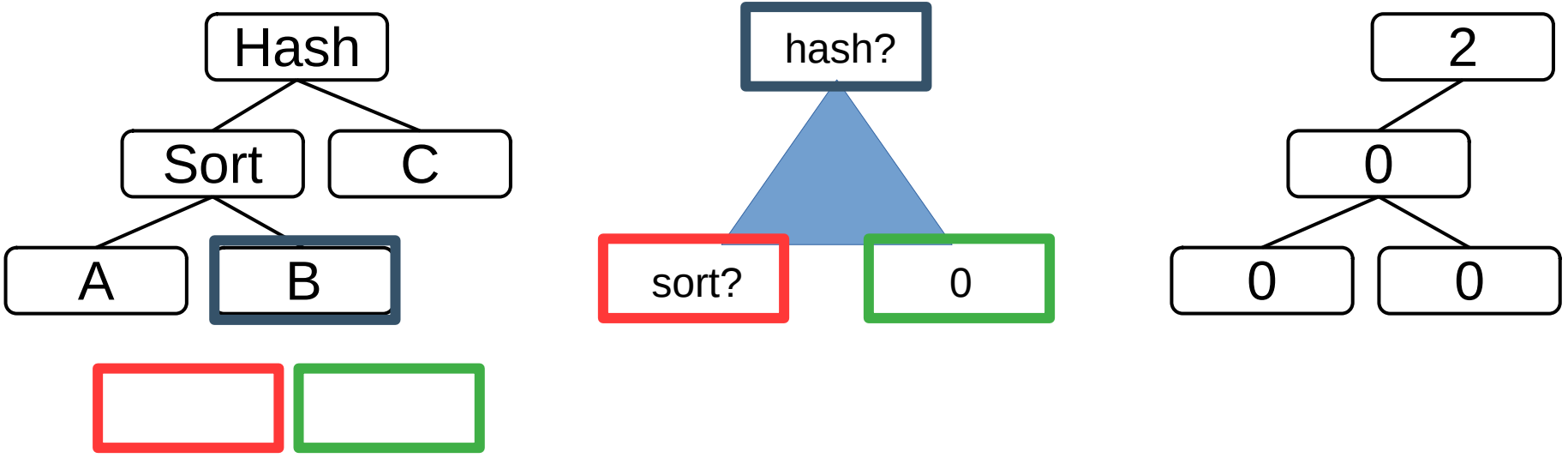
Detects a hash on top of a sort

Tree Convolution



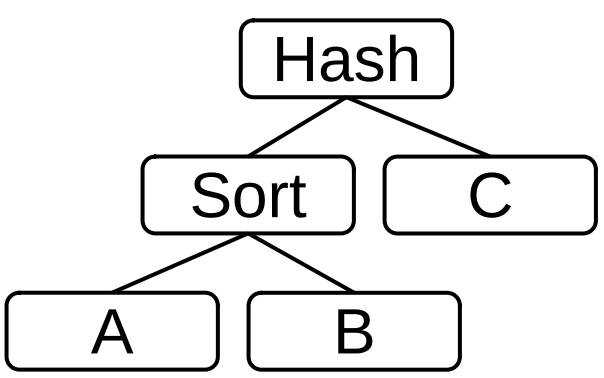
Detects a hash on top of a sort

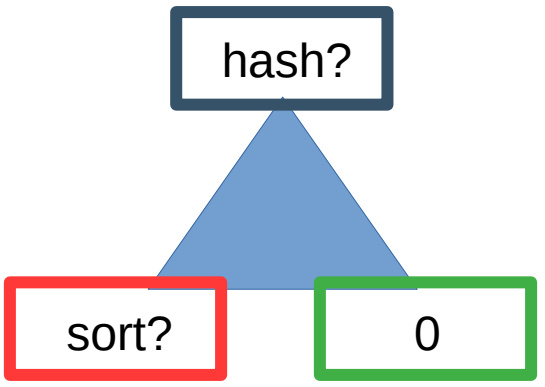
Tree Convolution

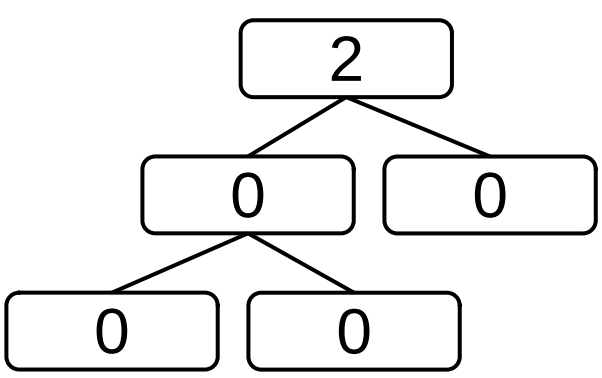


Detects a hash on top of a sort

Tree Convolution

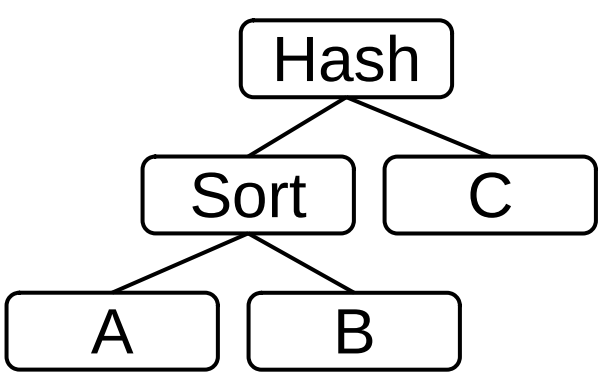


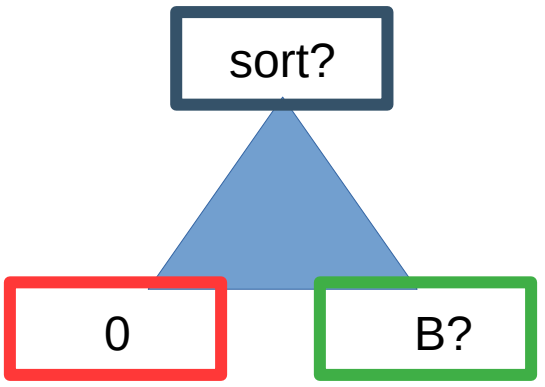


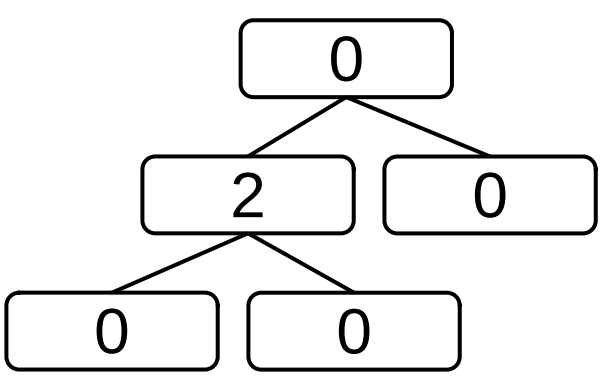


Detects a hash on top of a sort

Tree Convolution





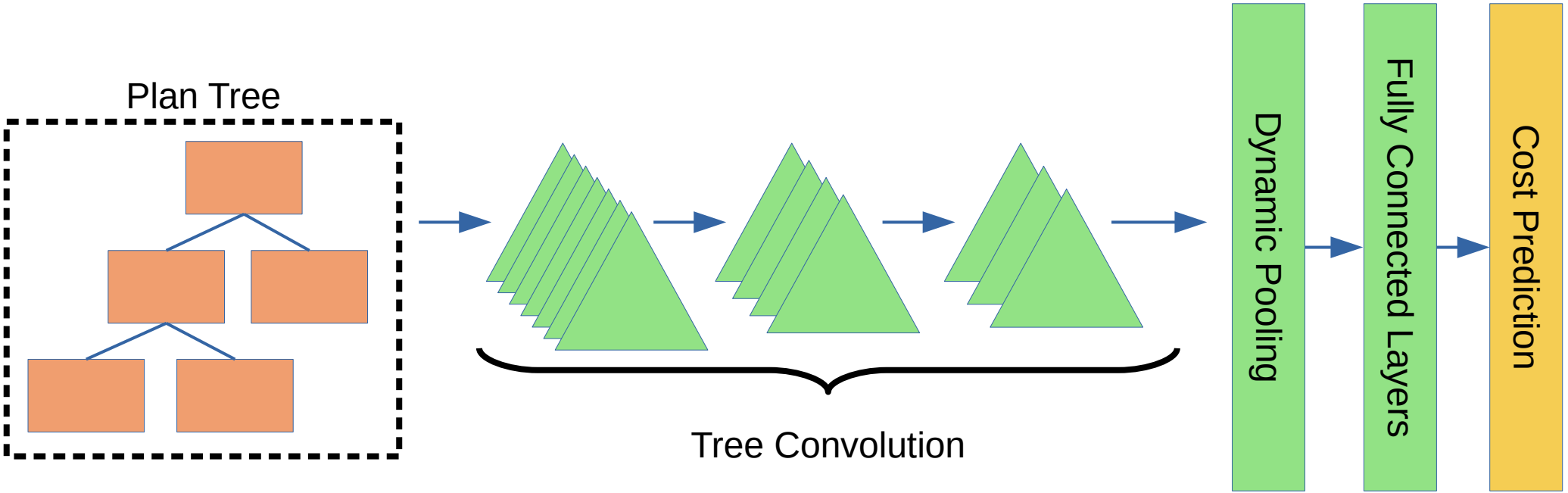


Detects a merge join with B on the right

Tree Convolution

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

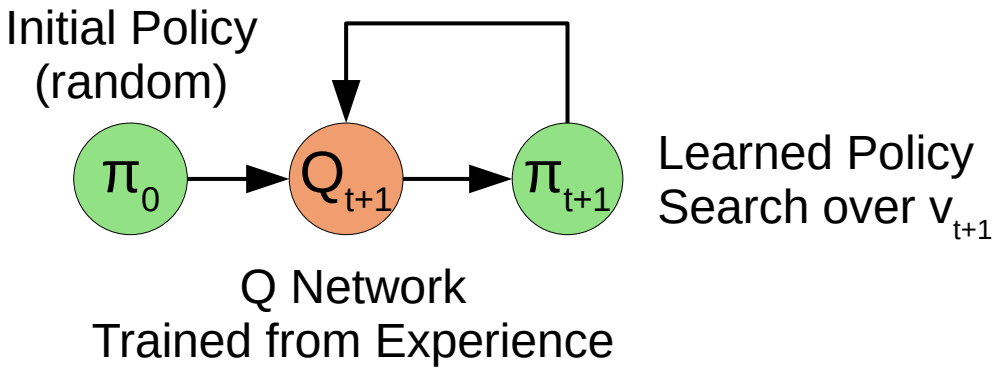
Neo



Value network architecture (used to approximate Q)

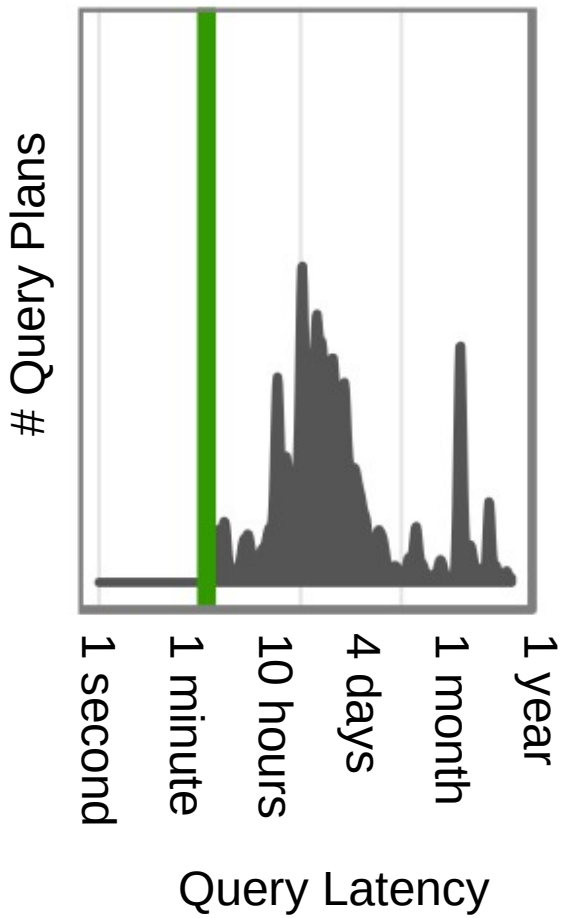
Random Policies

- 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.



Random Policies

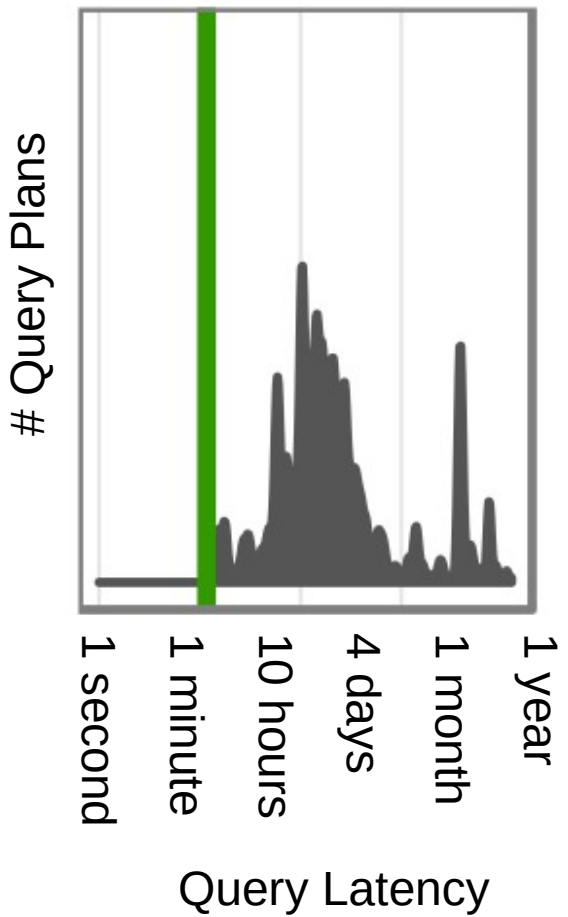
- DRL is very *sample inefficient*
  - You have to play for a long time before you get good.
- In QO, **doing worse takes longer!**
  - Cannot afford a random initial policy.



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

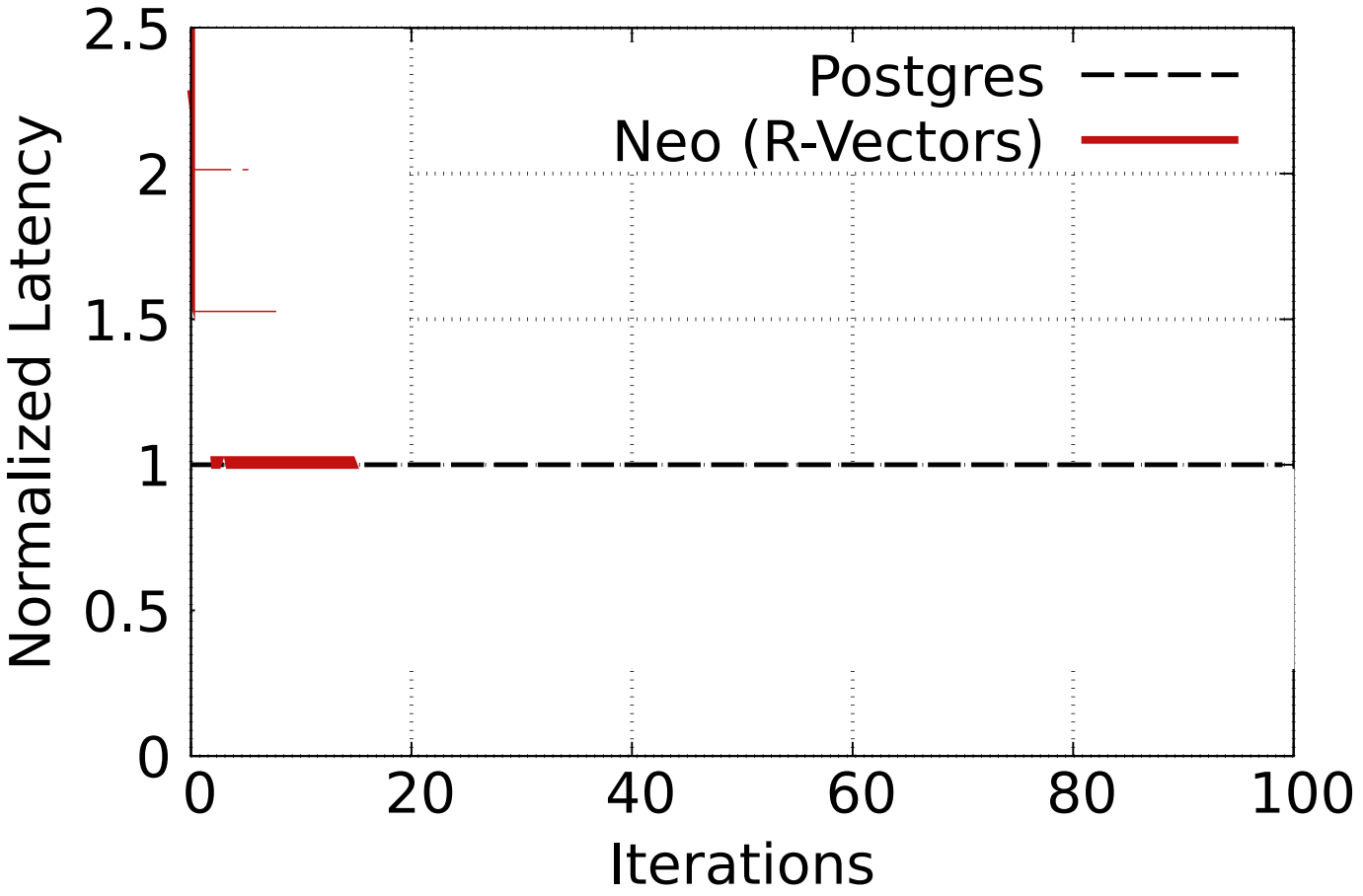
Random Policies

- 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.

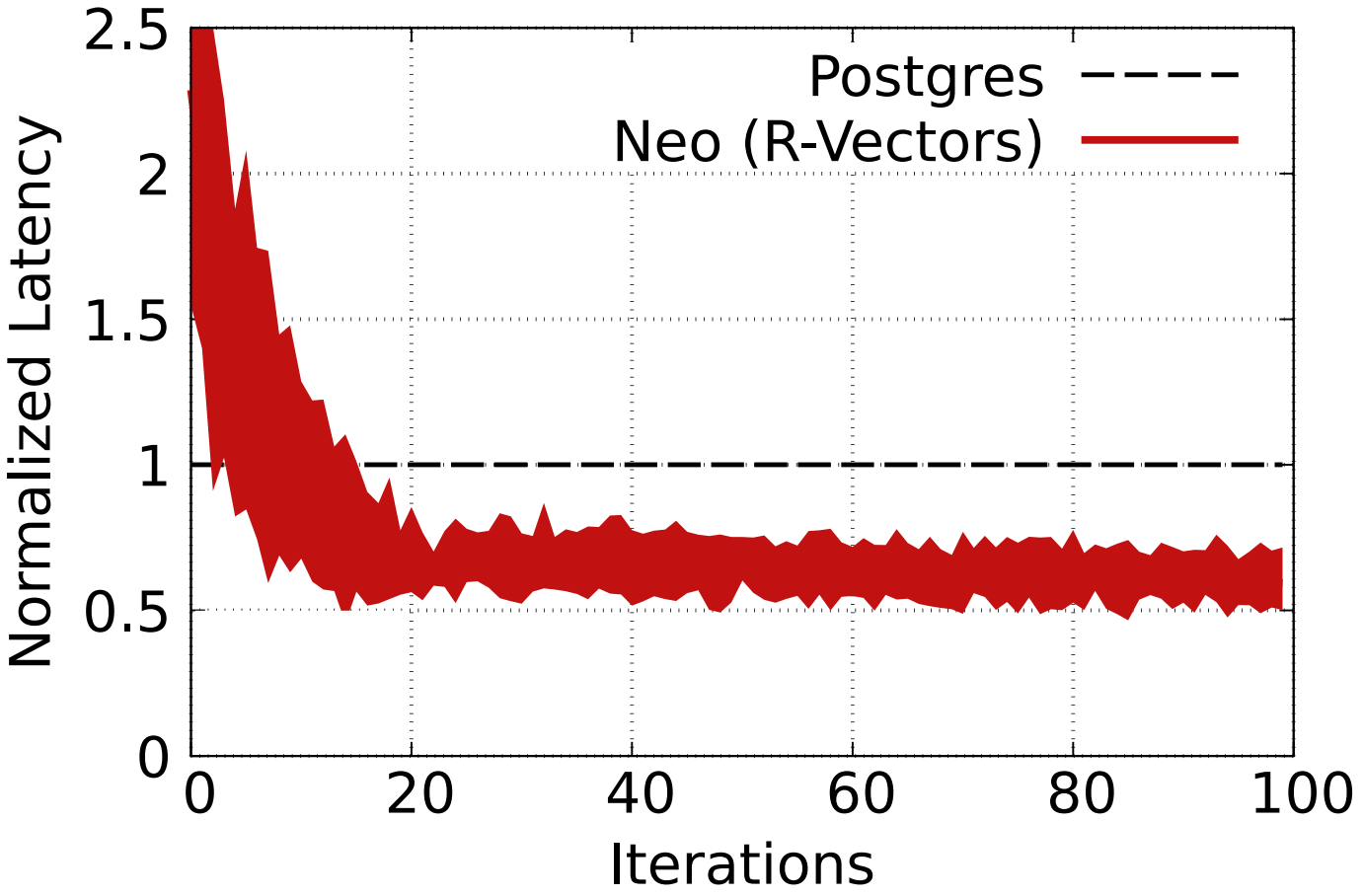
Experiments



1 = performance of PG optimizer

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

Experiments

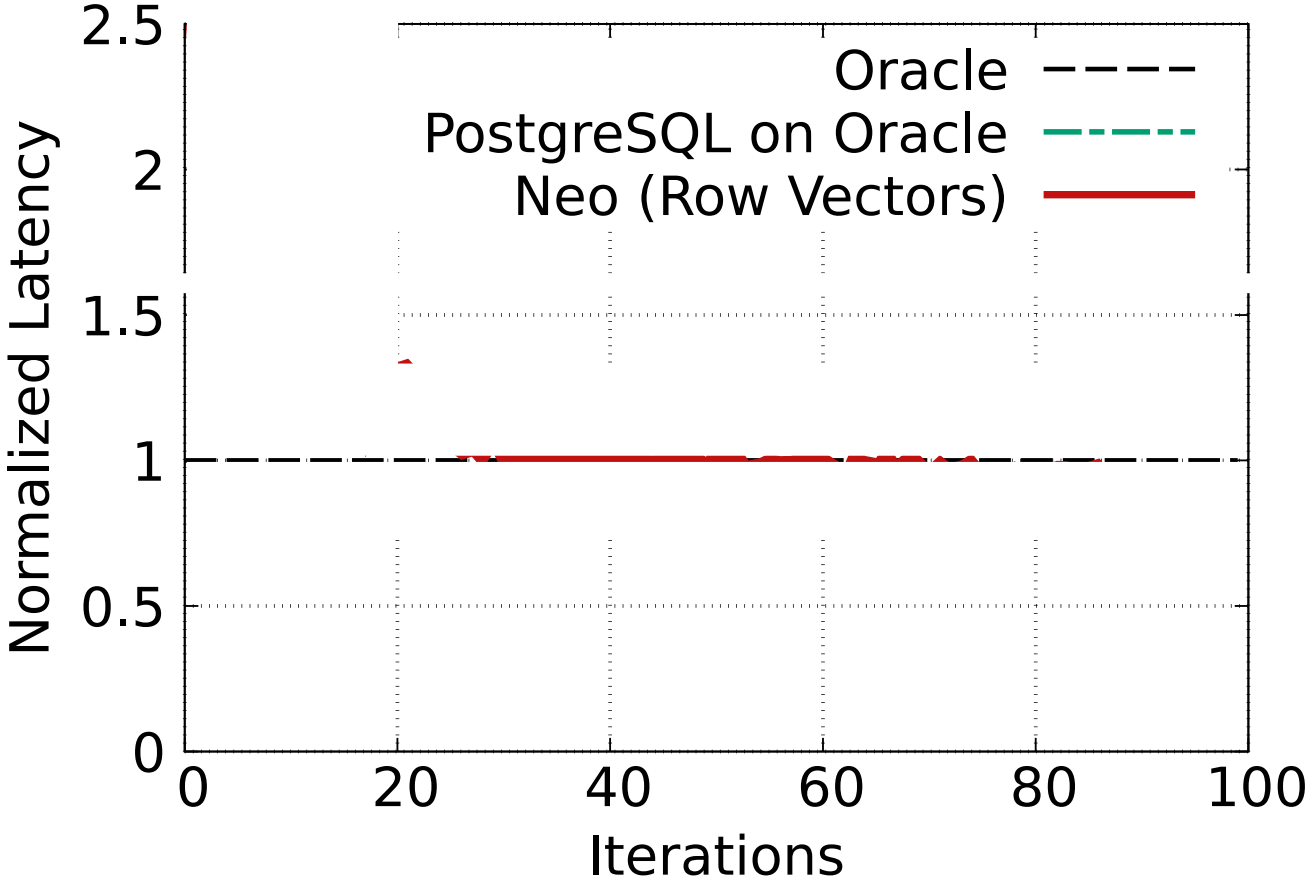


1 = performance of PG optimizer

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

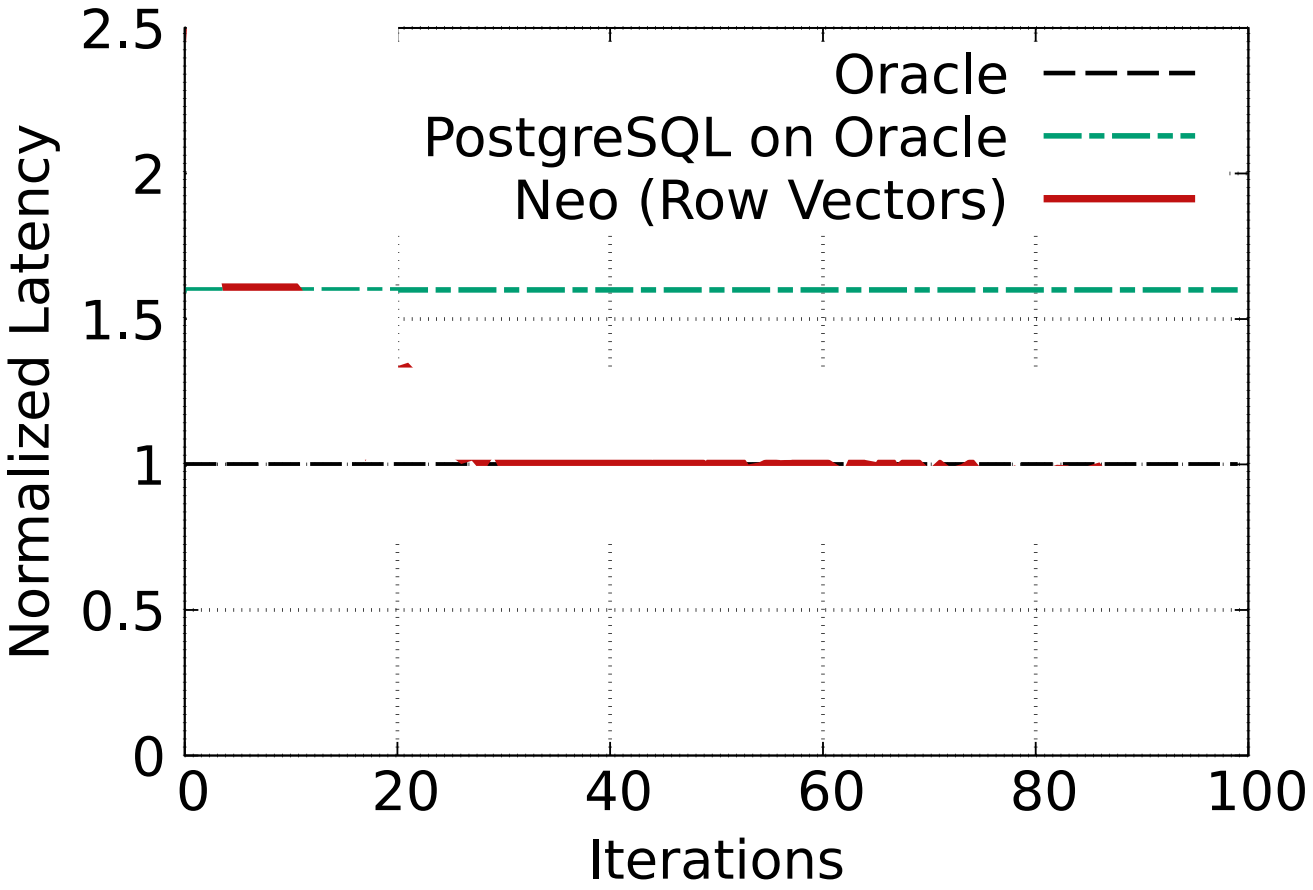
On test queries, Neo outperforms PG 15-25%.

Experiments



Black (1): performance of Oracle query optimizer

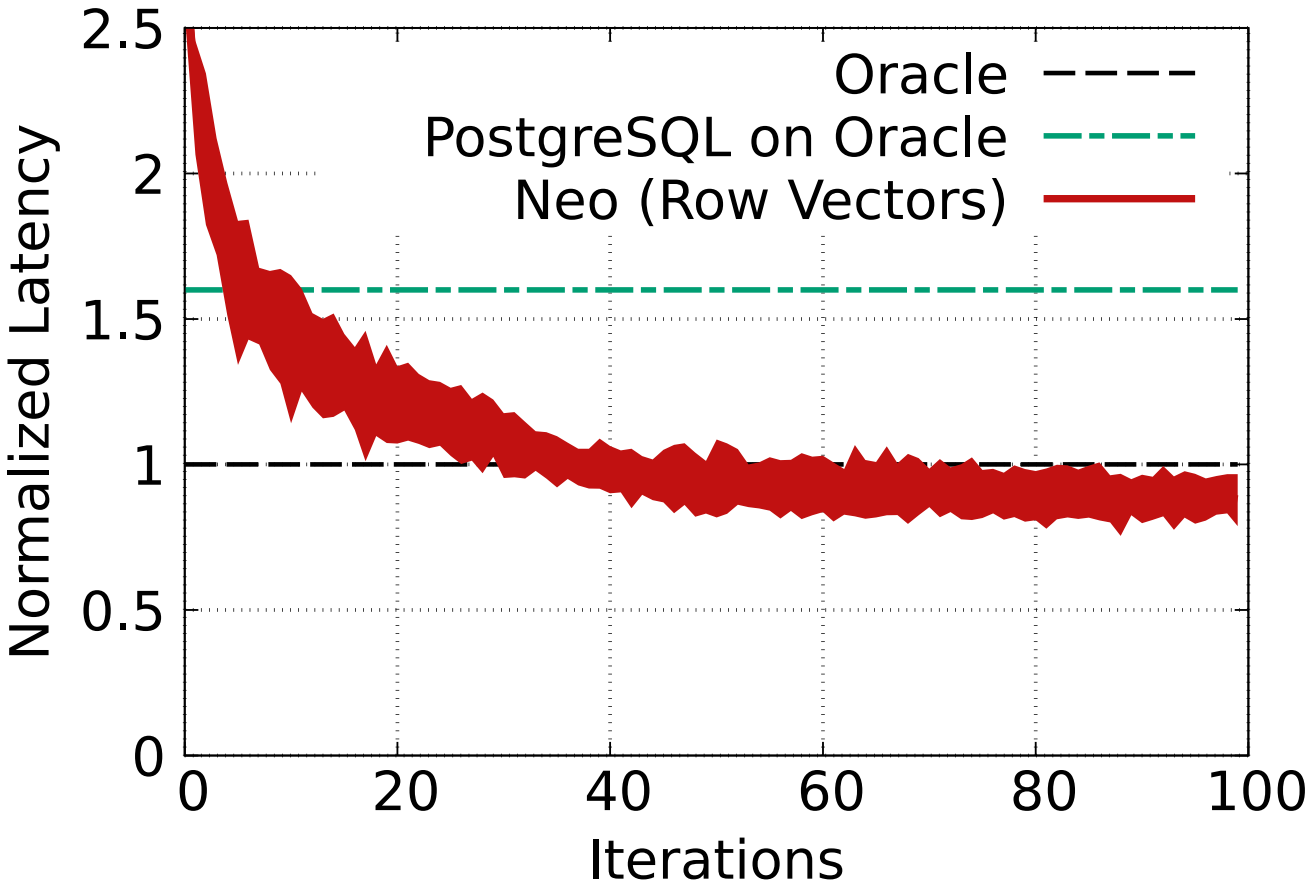
Experiments



Black (1): performance of Oracle query optimizer

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

Experiments

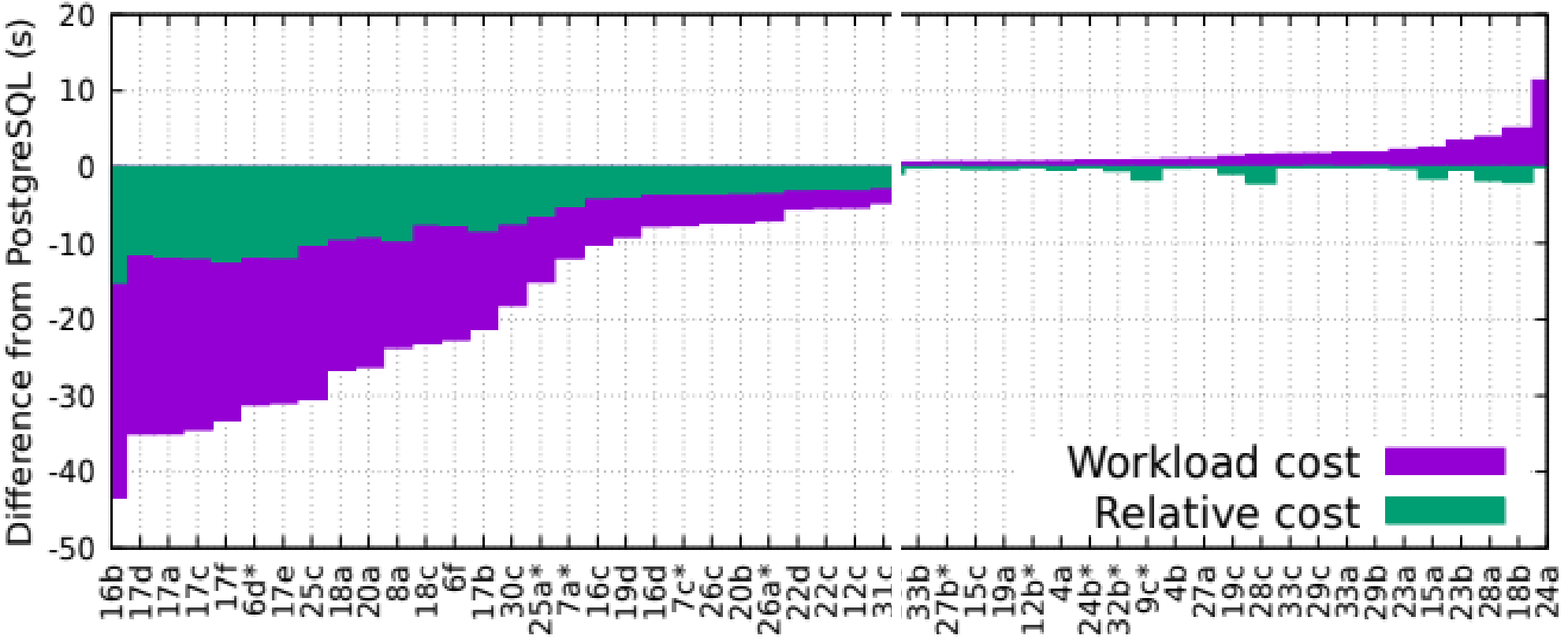


Black (1): performance of Oracle query optimizer

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

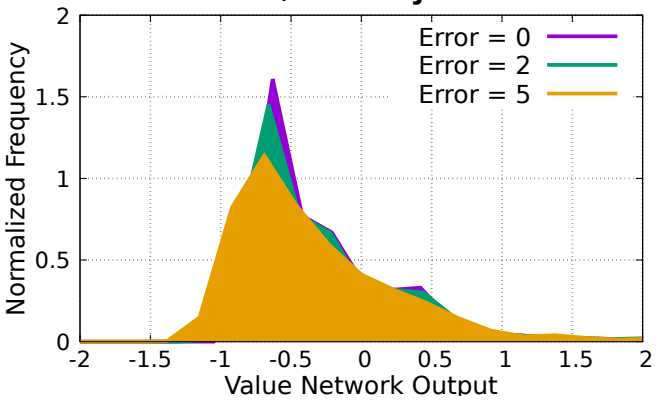
Red: Performance of Neo over time

Experiments

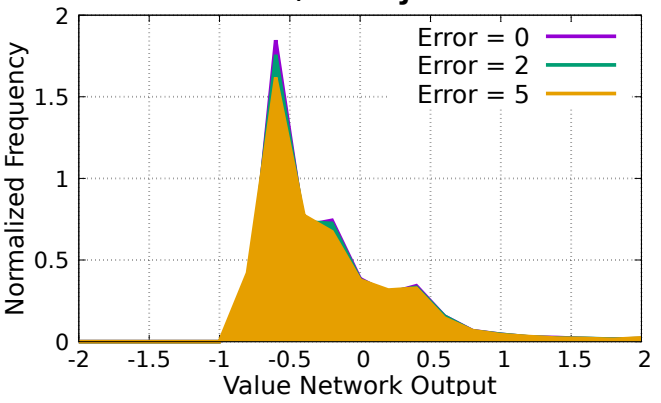


Experiments

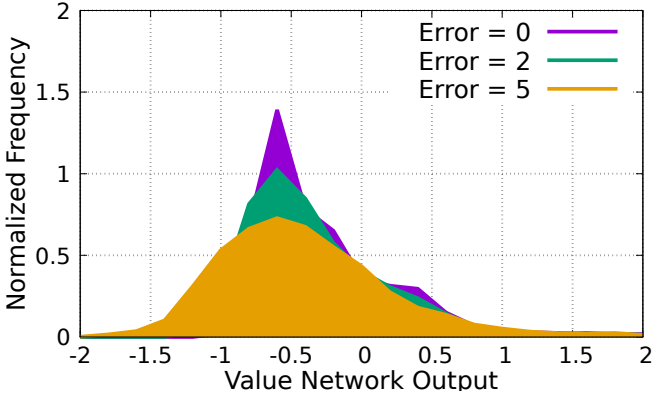
PG, <= 3 joins



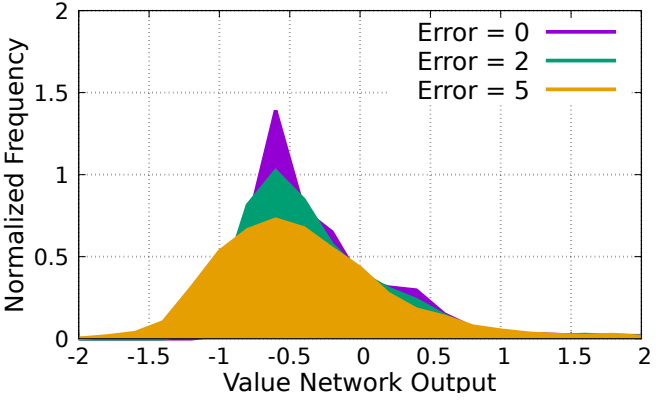
PG, > 3 joins



True card, <= 3 joins



True card, > 3 joins



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

That’s all!

- Neo: A Learned Query Optimizer
- A purely-learned policy with SOTA performance
- Me: Ryan Marcus (ryanmarcus@csail.mit.edu)
- Twitter: @RyanMarcus (web: http://rm.cab)
- These slides: http://rm.cab/neovldb19
- Paper: http://rm.cab/neo