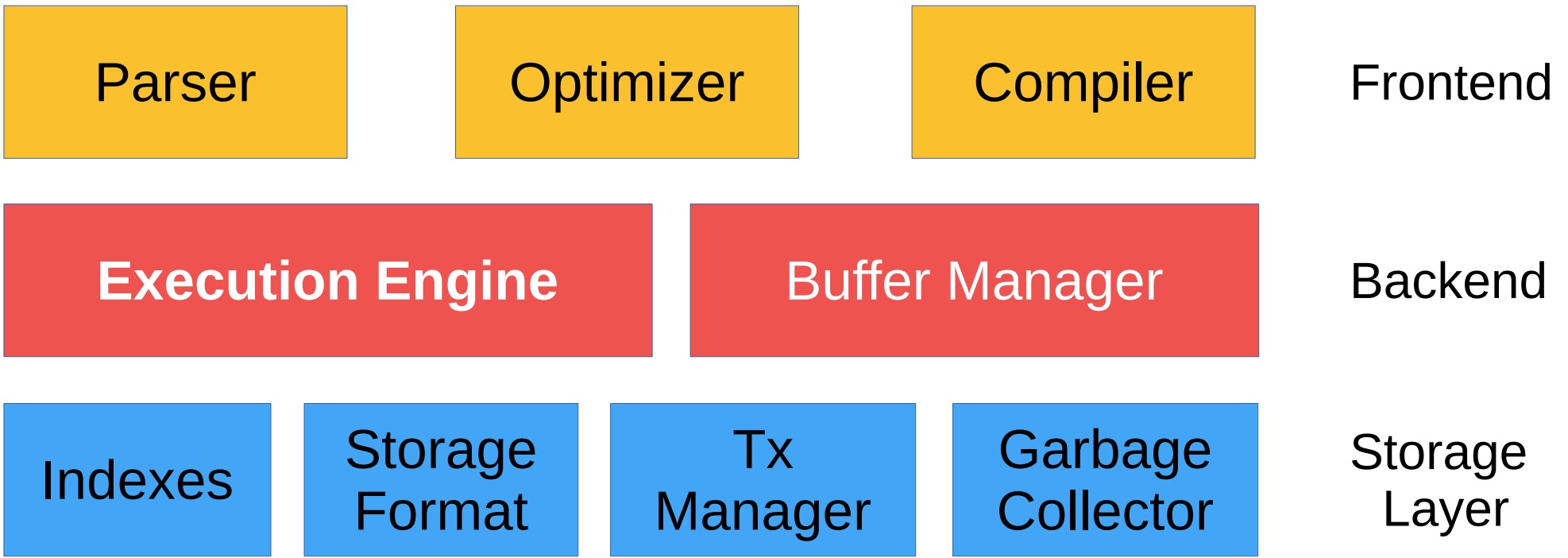
Query Execution & Data Layout

CIS 6500

Ryan Marcus

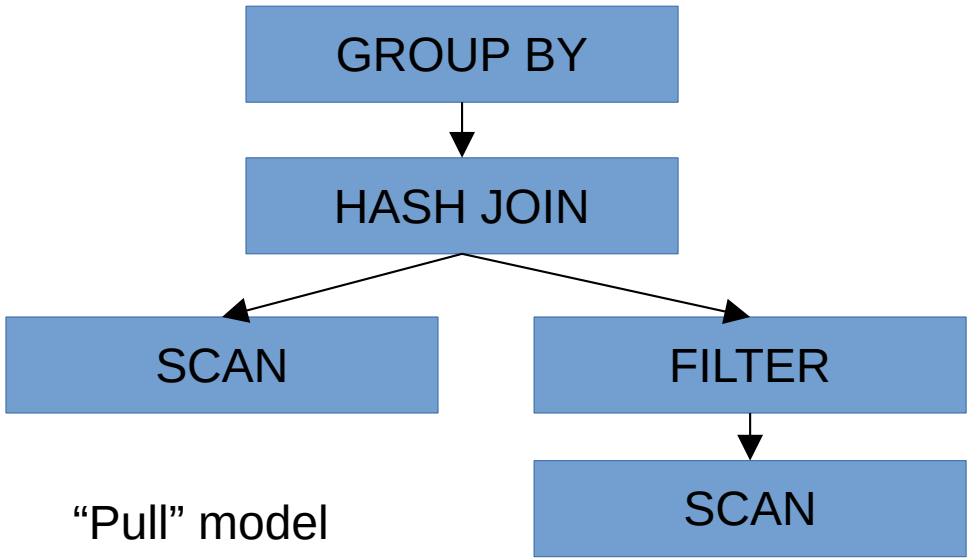
Did you fill this out? https://rm.cab/papers

DB Architecture



Execution Engine

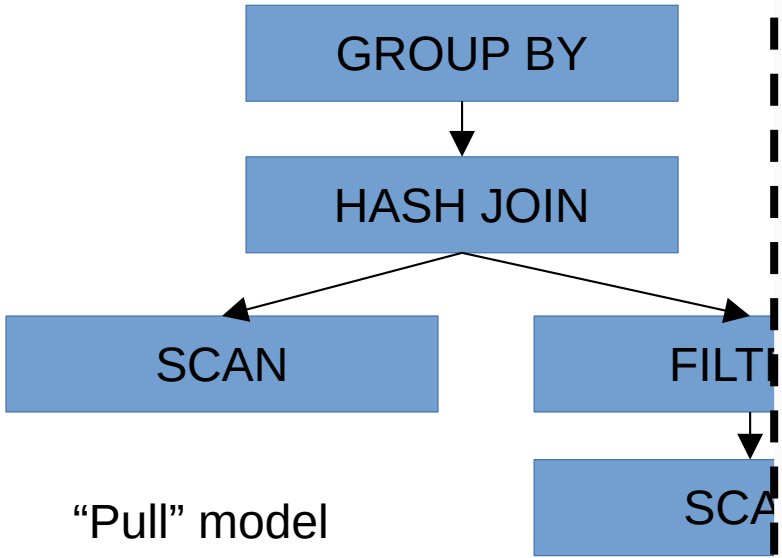
- Given a plan, actually process the bytes



“Pull” model

Execution Engine

- Given a plan, ac

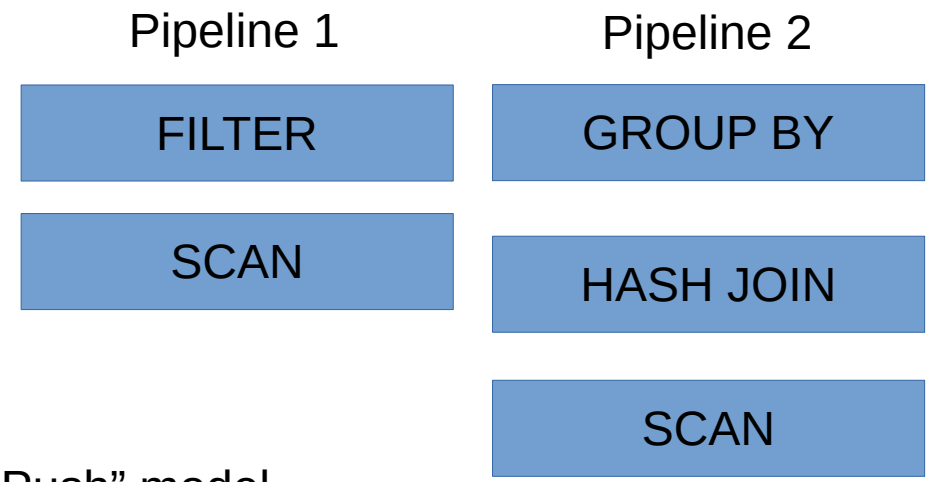


"Pull" model

```
class Filter implements Operator {
  Operator input;
  Predicate p;

  bool hasNext() {
    return input.hasNext();
  }

  Row next() {
    while (true) {
      Row r = input.next();
      if (p.check(r)) return r;
    }

    return null;
  }
}
```

Execution Engine

- Given a plan, actually process the bytes



“Push” model

Execution Engine

- Given a plan, ac

Pipeline 1

Pipe

FILTER

GRO

SCAN

HAS

S

"Push" model

```
function pipeline(Source src,
                  Sink dst,
                  Filter f,
                  Transform t) {

  for (Row r : src) {
    if (f != null && !f.check(r))
      continue;

    List<Row> rows = [r];
    if (t != null)
      rows = t.transform(r)

    for (Row r : rows) {
      dst.sink(r)
    }
  }
}
```

Execution Engine

- Interpreted engines
  - Generally “pull” model
  - Tuple-at-a-time or vectorized
- Compiling engines
  - Pipelined / generally “push” model

Simple
Extendable

Performance

Execution Engine Operations

- Filtering
- Joining
- Aggregating

Filtering

- Naive filtering – exactly what you think it is

C = “Penguin”

Input

A	B	C
5	Alpha	Horse
5	Beta	Cat
5	Gamma	Penguin
8	Delta	Penguin

Output

A	B	C
5	Gamma	Penguin
8	Delta	Penguin

Filtering

- Bitmap filtering – mark, don’t copy

A = 5

Input

A	B	C
5	Alpha	Horse
5	Beta	Cat
5	Gamma	Penguin
8	Delta	Penguin

Output

Filter
1
1
1
0

Filtering

- Late materialization – only read what you need

B = “Beta”

Input

A	B	C
5	Alpha	Horse
5	Beta	Cat
5	Gamma	Penguin
8	Delta	Penguin

Filtering

- Late materialization – only read what you need

B = “Beta”

Input

A	B	C
5	Alpha	Horse
5	Beta	Cat
5	Gamma	Penguin
8	Delta	Penguin

Output

Row	A	B	C
2	5	Beta	Cat

Filtering

- Late materialization – only read what you need

B = “Beta”

Input

A	B	C
5	Alpha	Horse
5	Beta	Cat
5	Gamma	Penguin
8	Delta	Penguin

Output

A	B	C
5	Beta	Cat

Filtering

- Which strategy is best depends on selectivity

Very selective	Late materialization
Selective	Copy or bitmap
Not selective	Bitmap

Joins

Name	Dept
Ryan	CIS
Zack	CIS
Boon	CIS
Susan	CIS
Martha	Psych

⋈

Dept	Admin	Admin email
CIS	Cheryl	che@...
Psych	Martin	mar@...

=

Name	Dept	Admin	Admin email
Ryan	CIS	Cheryl	che@...
Zack	CIS	Cheryl	che@...
Boon	CIS	Cheryl	che@...
Susan	CIS	Cheryl	che@...
Martha	Psych	Martin	mar@...

Joins

- Skew makes processing joins hard.
  - 1 row can turn into N rows
  - N rows can turn into 0 rows
- Think about how various methods handle skew
- Three ways: loops, hash, merge.

Loop Join

- For each row of the left side table, find matches in the right side table.
  - Can use an index on the right side table.

$$LoopCost = |L| \times \log |R|$$

Hash Join

- Build a hash table of the left (“build”) side, search the hash table for each row on the right (“probe”) side

$$HashCost = |L| + |R|$$

Merge Join

- Sort the left and right sides, then find matches via merge.
  - Can sometimes use an index to help sort

$$MergeCost = |L| \log |L| + |R| \log |R| + |L| + |R|$$

Join Choice

	Cost	Memory	Parallelism
Loop Join	$\|L\| \times \log \|R\|$	1	$\text{Over } \|L\|$
Hash Join	$\|L\| + \|R\|$	$\|L\| \times (2 - LF)$	$\text{Over } \|R\|$
Merge Join	$\|L\| \log \|L\| + \|R\| \log \|R\| + \|L\| + \|R\|$	1	Sorts
Merge Join (presort)	$\|L\| + \|R\|$	1	???

Aggregation

MIN(B) GROUP BY A

K	V

Hash-based

A	B
5	6
5	2
6	3
6	4
5	5
2	1

Aggregation

MIN(B) GROUP BY A

K	V
5	6

Hash-based

A	B
5	6
5	2
6	3
6	4
5	5
2	1

Aggregation

MIN(B) GROUP BY A

K	V
5	2

Hash-based

A	B
5	6
5	2
6	3
6	4
5	5
2	1

Aggregation

MIN(B) GROUP BY A

K	V
5	2
6	3

Hash-based

A	B
5	6
5	2
6	3
6	4
5	5
2	1

Aggregation

MIN(B) GROUP BY A

K	V
5	2
6	3

Hash-based

A	B
5	6
5	2
6	3
6	4
5	5
2	1

Aggregation

MIN(B) GROUP BY A

K	V
5	2
6	3

Hash-based

A	B
5	6
5	2
6	3
6	4
5	5
2	1

Aggregation

MIN(B) GROUP BY A

K	V
5	2
6	3
2	1

Hash-based

A	B
5	6
5	2
6	3
6	4
5	5
2	1

Aggregation

MIN(B) GROUP BY A

K	V

Sort-based

A	B
2	1
5	5
5	2
5	6
6	3
6	4

Aggregation

MIN(B) GROUP BY A

K	V
2	1

Sort-based

A	B
2	1
5	5
5	2
5	6
6	3
6	4

Aggregation

MIN(B) GROUP BY A

K	V
2	1
5	2

Sort-based

A	B
2	1
5	5
5	2
5	6
6	3
6	4

Aggregation

MIN(B) GROUP BY A

K	V
2	1
5	2
6	3

Sort-based

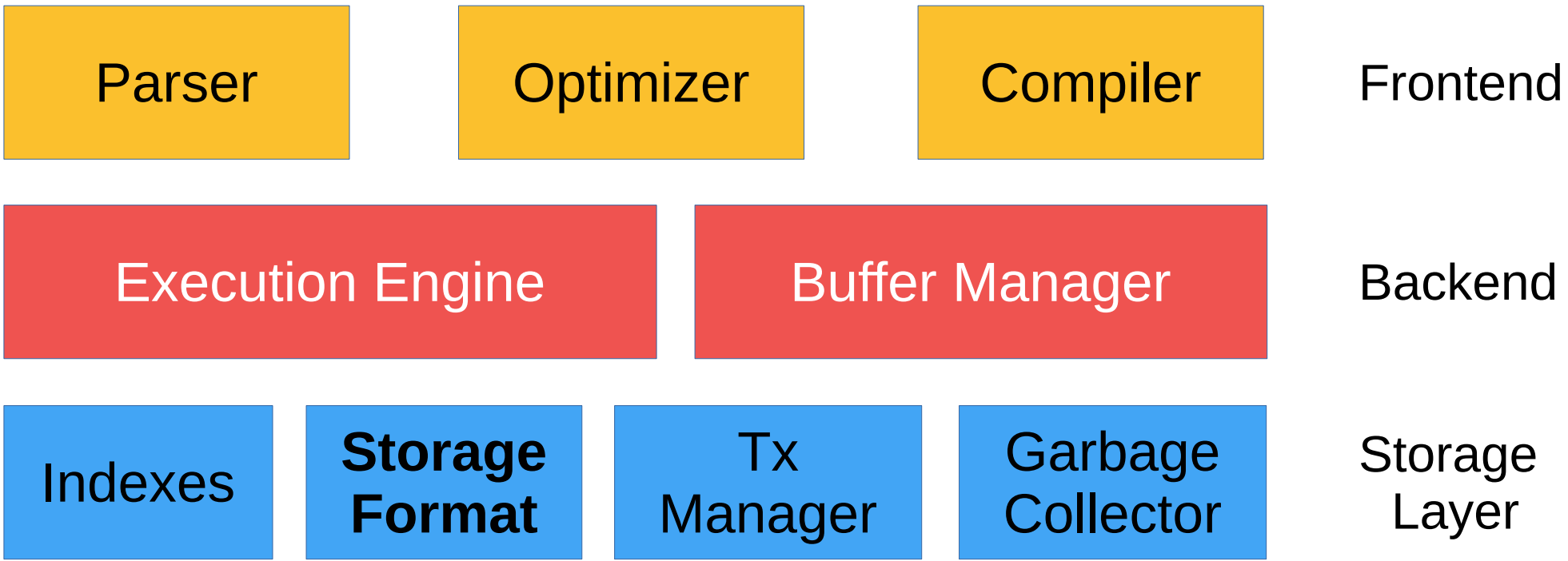
A	B
2	1
5	5
5	2
5	6
6	3
6	4

Aggregation

- Sort-based
- Works well when data is already sorted
- Helpful when there are many keys
  - Incremental output
  - Low memory usage
  - Output already sorted

- Hash-based
- Generally the fastest method
  - Parallelism
- Must store every key in memory

DB Architecture



Storage Format

- Data layout impacts read sizes

A	B	C	D
Alpha	1	Red	UPenn
Beta	2	Black	Cornell
Gamma	3	Blue	Harvard
Delta	4	Purple	Dartmouth

```
SELECT B, C FROM T;
```

Storage Format

- Data layout impacts read sizes

A	B	C	D
Alpha	1	Red	UPenn
Beta	2	Black	Cornell
Gamma	3	Blue	Harvard
Delta	4	Purple	Dartmouth

Row order

```
SELECT B, C FROM T;
```

4 blocks

Storage Format

- Data layout impacts read sizes

A	B	C	D
Alpha	1	Red	UPenn
Beta	2	Black	Cornell
Gamma	3	Blue	Harvard
Delta	4	Purple	Dartmouth

Column order

```
SELECT B, C FROM T;
```

2 blocks

Storage Format

- Data layout impacts read sizes

A	B	C	D
Alpha	1	Red	UPenn
Beta	2	Black	Cornell
Gamma	3	Blue	Harvard
Delta	4	Purple	Dartmouth

```
INSERT INTO T
(A, B, C, D)
VALUES (Omega, 5,
Pink, Yale)
```

Storage Format

- Data layout impacts read sizes

A	B	C	D
Alpha	1	Red	UPenn
Beta	2	Black	Cornell
Gamma	3	Blue	Harvard
Delta	4	Purple	Dartmouth

Row order

```
INSERT INTO T
(A, B, C, D)
VALUES (Omega, 5,
Pink, Yale)
```

1 block

Storage Format

- Data layout impacts read sizes

A	B	C	D
Alpha	1	Red	UPenn
Beta	2	Black	Cornell
Gamma	3	Blue	Harvard
Delta	4	Purple	Dartmouth

Column order

```
INSERT INTO T
(A, B, C, D)
VALUES (Omega, 5,
Pink, Yale)
```

4 blocks

Column Groups

- Data layout impacts read sizes

A	B	C	D
Alpha	1	Red	UPenn
Beta	2	Black	Cornell
Gamma	3	Blue	Harvard
Delta	4	Purple	Dartmouth

Row groups

```
INSERT INTO T
(A, B, C, D)
VALUES (Omega, 5,
Pink, Yale)
```

3 blocks

Storage Format

A
5
5
5
5
7
7
7

Normal

A
4x 5
3x 7

Run-length

A
5
0
0
0
2
0
0

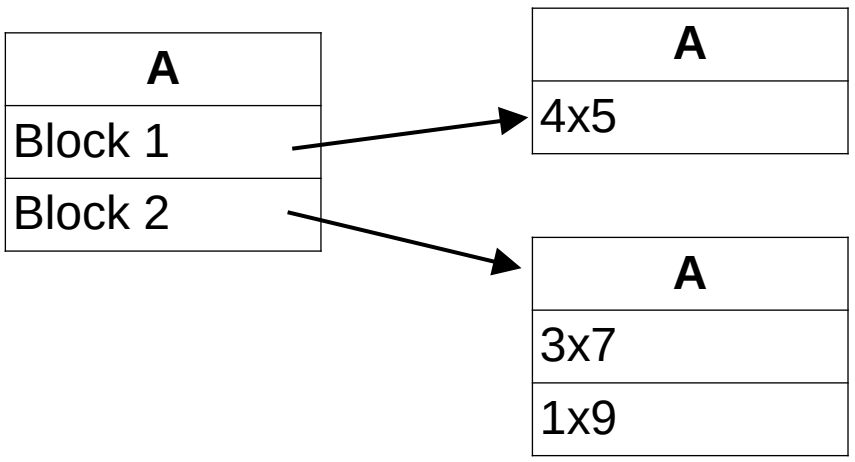
Delta

Blocking

- Random access on uncompressed data is easy

A
5
5
5
5
7
7
7
9

Normal



Block layout

Blocking

- Smaller blocks?
  - Finer-grained access 👍
  - Less efficient compression 👎
  - More metadata overhead 👎
- Larger blocks?
  - Coarser-grained access 👎
  - Higher-throughput scans 👍

https://rm.cab/seminaralt

Type of Compression

- Heuristic compression: look at patterns in data, come up with an algorithm to make it smaller
  - Run-length encoding, delta encoding
- Entropy compression: fundamental algorithms over streams of bytes
  - Huffman trees, arithmetic encoding, succinct data structures

Heuristic Compression

- “Just kind of seems like a good idea”
- Run length encoding
- Delta encoding

A
4x 5
3x 7

Run-length

A
5
0
0
0
2
0
0

Delta

Heuristic Compression

- “Just kind of seems like a good idea”
- Dictionary encoding

A
San Francisco
Philadelphia
Philadelphia
Albuquerque
Albuquerque
San Francisco
Philadelphia

D(A)
Albuquerque
Philadelphia
San Francisco

A
2
1
1
0
0
2
1

Sortable!

Heuristic Compression

- Text compression – LZSS encoding.

I_AM_SAM_SAM_I_AM_THAT_SAM_I_AM

Message types:
0 <LENGTH> <LITERAL>
1 <OFFSET> <LENGTH>

Written	Produced	Output
0 6 I_AM_S	I_AM_S	I_AM_S
1 -4 3	AM_	I_AM_SAM_
1 -4 4	SAM_	I_AM_SAM_SAM_
1 -13 5	I_AM_	I_AM_SAM_SAM_I_AM
0 5 THAT_	THAT_	I_AM_SAM_SAM_I_AM_THAT_
1 -14 8	SAM_I_AM	I_AM_SAM_SAM_I_AM_THAT_SAM_I_AM

Entropy Compression

$$H(x) = -\sum_{c \in A} P(c) \log_2 P(c)$$

$$\left[-\frac{1}{4} \log_2 \left(\frac{1}{4} \right) \right] = 2 \text{ bits}$$

Character	Probability
Red	25%
Blue	25%
Green	25%
Yellow	25%

00 → Red
01 → Blue
10 → Green
11 → Yellow

The absolute *best* you can do… given the assumptions.

Entropy Compression

$$H(x) = -\sum_{c \in A} P(c) \log_2 P(c)$$

Character	Probability
Red	80%
Blue	5%
Green	5%
Yellow	10%

Entropy is...

$$-(.80 \log_2 0.8 + 2(0.05 \log_2 0.05) + 0.1 \log_2 0.1)$$

= 1.02 bits

… but how to design the actual code?

Huffman Trees

Character	Probability
Red	80%
Blue	5%
Green	5%
Yellow	10%

Make a weighted node for each character

Repeat until there is only one node:

Select the two nodes with lowest weight
Combine them into a tree.

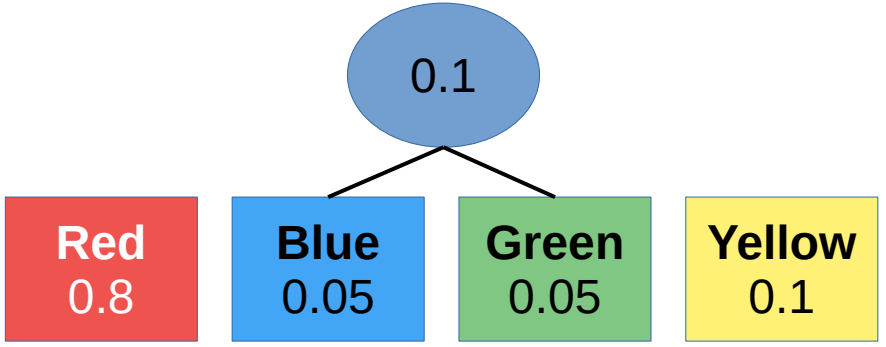
Huffman Trees

Character	Probability
Red	80%
Blue	5%
Green	5%
Yellow	10%

Make a weighted node for each character

Repeat until there is only one node:

Select the two nodes with lowest weight
Combine them into a tree.



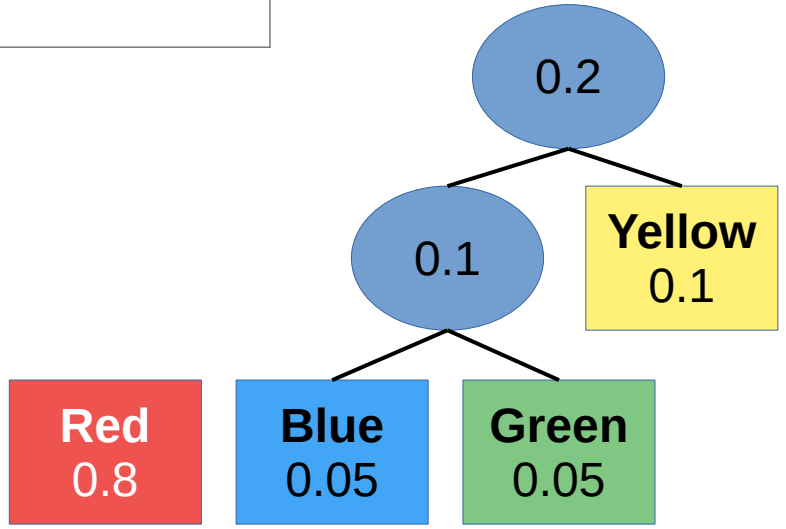
Huffman Trees

Character	Probability
Red	80%
Blue	5%
Green	5%
Yellow	10%

Make a weighted node for each character

Repeat until there is only one node:

Select the two nodes with lowest weight
Combine them into a tree.



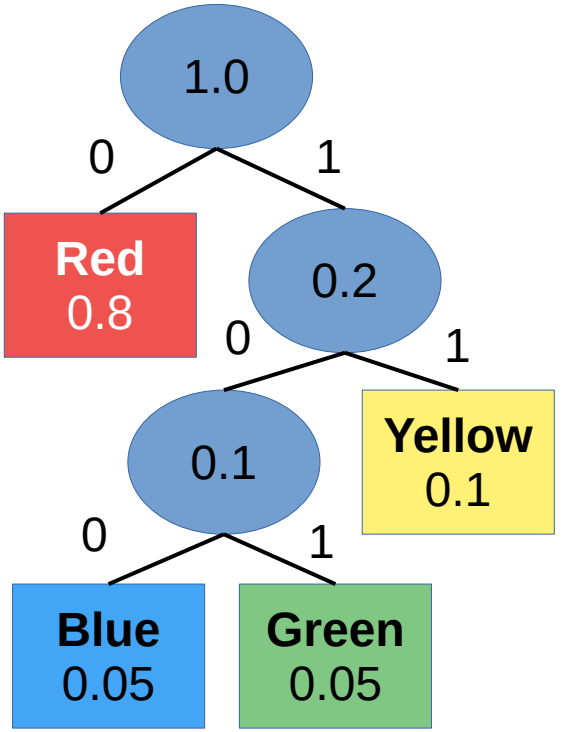
Huffman Trees

Character	Probability
Red	80%
Blue	5%
Green	5%
Yellow	10%

Red → 0
Yellow → 11
Blue → 100
Green → 101

0.8 * 1 bit
+ 0.1 * 2 bits
+ 0.05 * 3 bits
+ 0.05 * 3 bits
= 1.3 bits

Prefix free!



Huffman Tree

- Optimality: Huffman tree achieves minimal entropy iff probabilities are *Dyadic*

$$2^x \quad x \in \{-1, -2, \dots\}$$

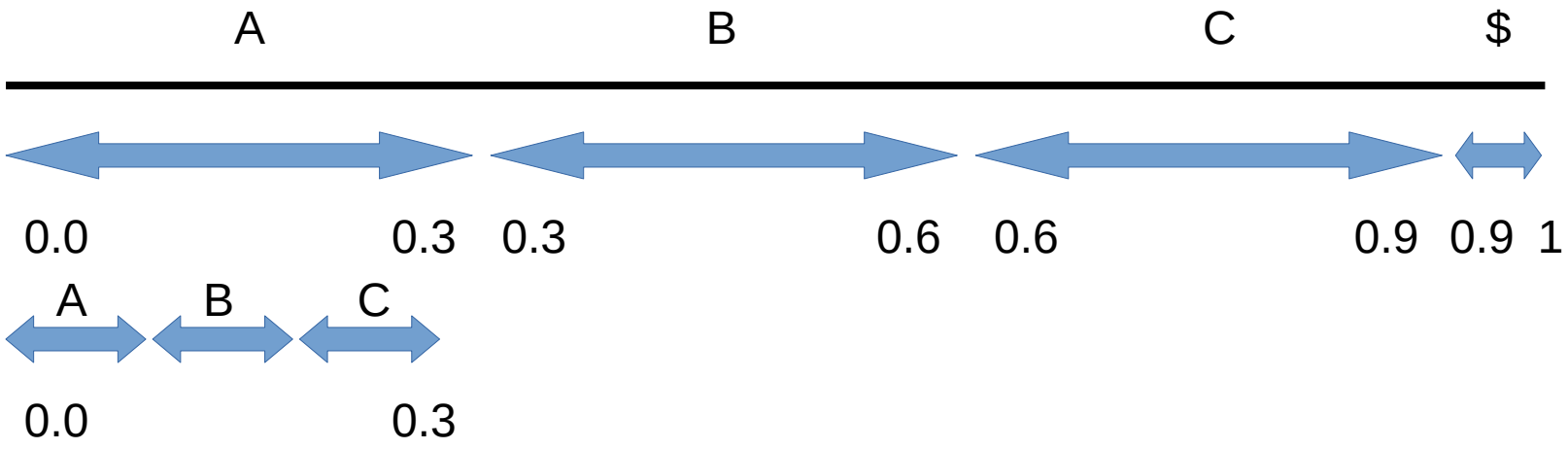
- Huffman codes can be arbitrarily bad when the most popular symbol has p > 0.5.
  - Can never assign less than 1 bit to a symbol

Entropy Compression

- What about non-uniform distributions?
  - “u” after “q”
  - Huffman bigrams!
- In general, what if I know P(c | current message)?

Arithmetic encoding

- Turn any message into a single number

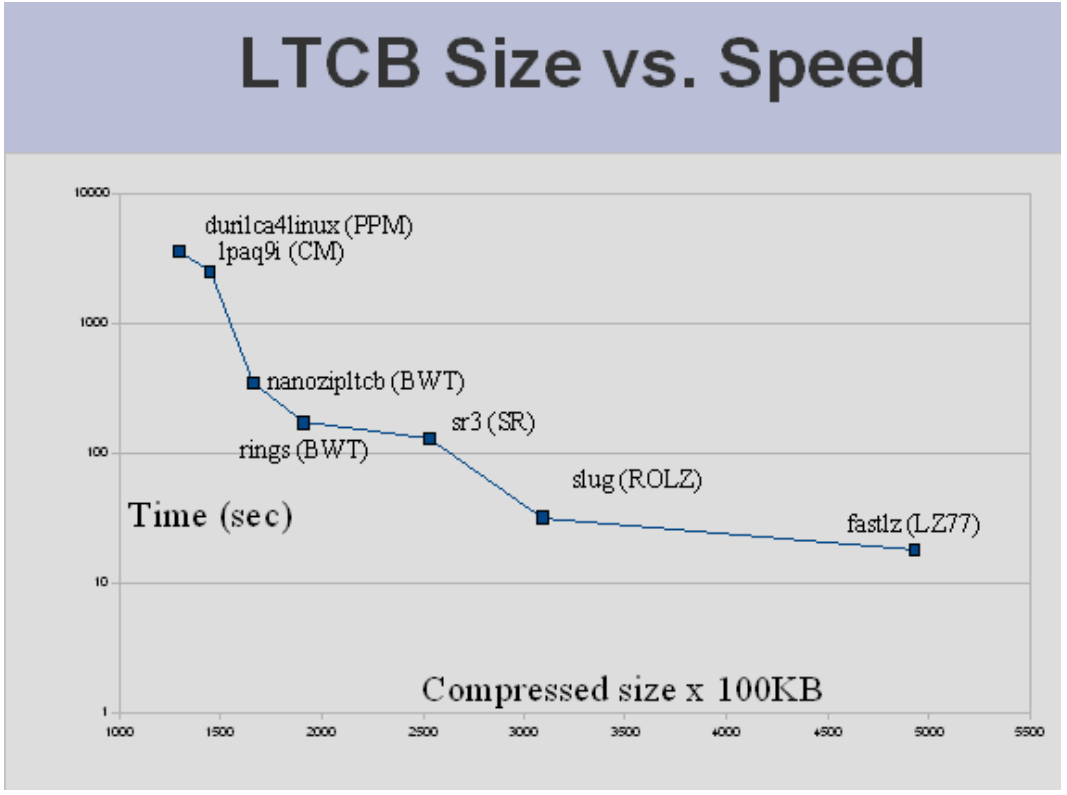


Combining Heuristic + Entropy

- LVSS + Huffman Tree = Gzip
- Dictionary + Arithmetic = Zstd
- Common in database systems:
  - Huffman + Dictionary: “compressed dictionary”
  - Huffman + delta encoding
  - Zstd for maximum compression

Decompression Speeds

- Smaller is cheaper, but sometimes slower

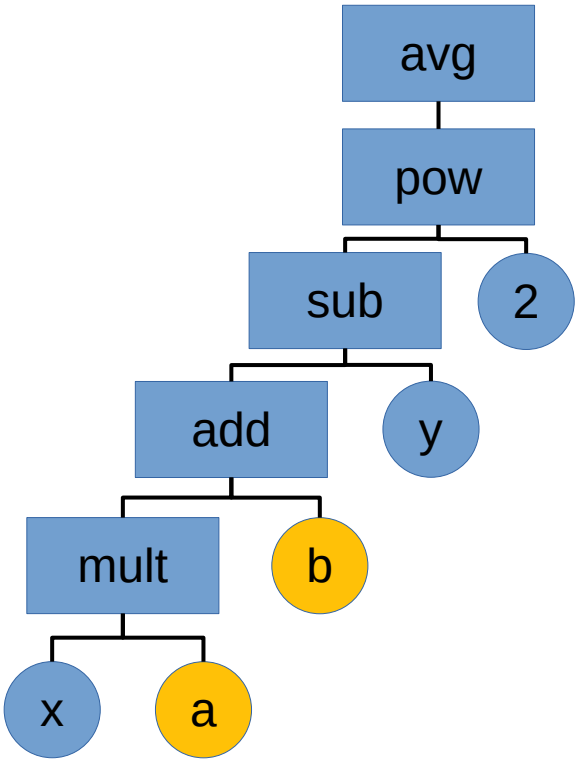


Deep Learning

- PyTorch can differentiate functions for us

```
[12] x = torch.Tensor([1, 3])
     y = torch.Tensor([2, 5])
     a = torch.nn.Parameter(torch.Tensor([1]))
     b = torch.nn.Parameter(torch.Tensor([1]))

     loss = (x * a + b - y).pow(2).mean()
     loss
```

```
tensor(0.5000, grad_fn=<MeanBackward0>)
```



Deep Learning

```
lenet = nn.Sequential(
    nn.Conv2d(1, 6, 2),
    nn.AvgPool2d(4),
    nn.Sigmoid(),
    nn.Conv2d(6, 16, 2),
    nn.AvgPool2d(4),
    nn.Sigmoid(),
    nn.Flatten(),
    nn.Linear(16, 120),
    nn.Sigmoid(),
    nn.Linear(120, 84),
    nn.Sigmoid(),
    nn.Linear(84, 10)
)
```

```
[205] lenet(imgs).shape
```

```
torch.Size([10, 10])
```

Each layer applied one after the other

Input features, output features, kernel size

4x4 average pooling

16 features to 120 features

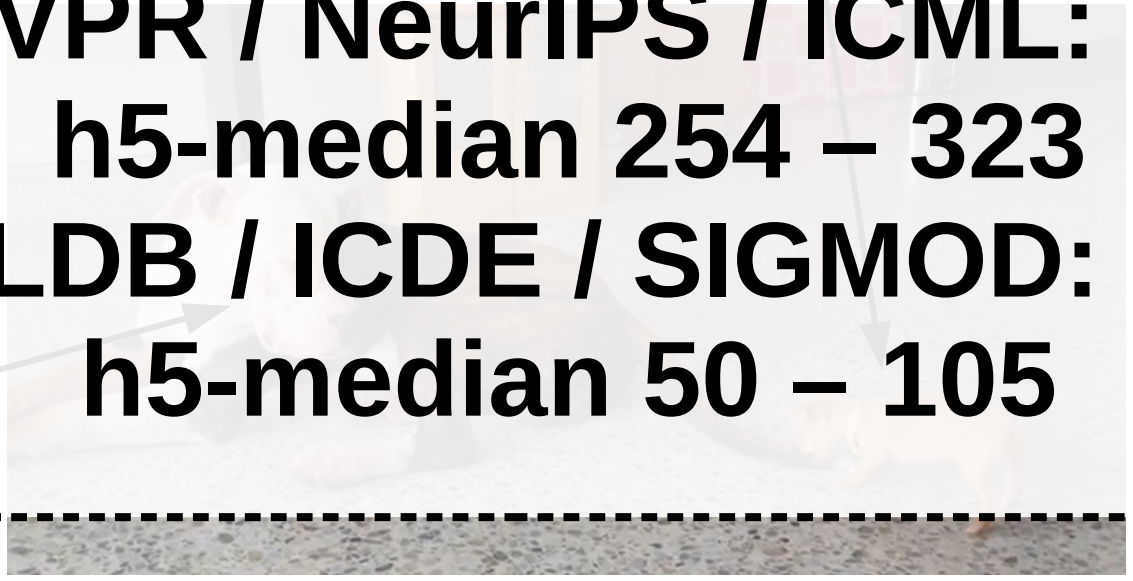
Final output: one value per class

Deep Learning for QO

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

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

Deep Learning for QO



CVPR / NeurIPS / ICML:
h5-median 254 – 323
VLDB / ICDE / SIGMOD:
h5-median 50 – 105

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

Deep Learning for QO

- Recent groundswell of research
- R/L based approaches:
  - Feb/Mar ‘18:
    - Marcus et al. (ReJOIN)
    - Ortiz et al.
  - Aug ‘18:
    - Krishnan et al. (DQ)
  - Now: Neo, VLDB ‘19
- Cardinality based approaches:
  - 2015: Liu et al.
  - 2019: Kipf et al. (MSCN)
  - Now:
    - group by (Kipf et al.)
    - local models (Woltmann et al.)
    - … and more!

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 QO

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

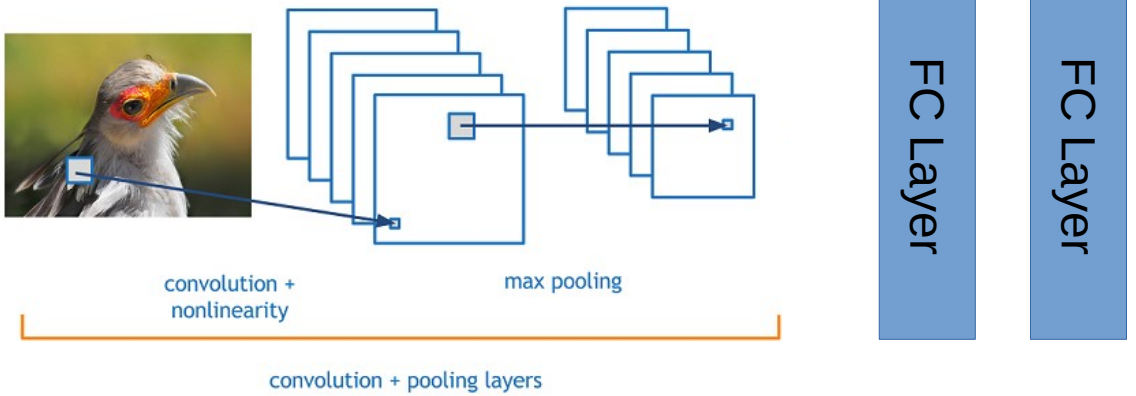
* my work

What makes DL *good?*

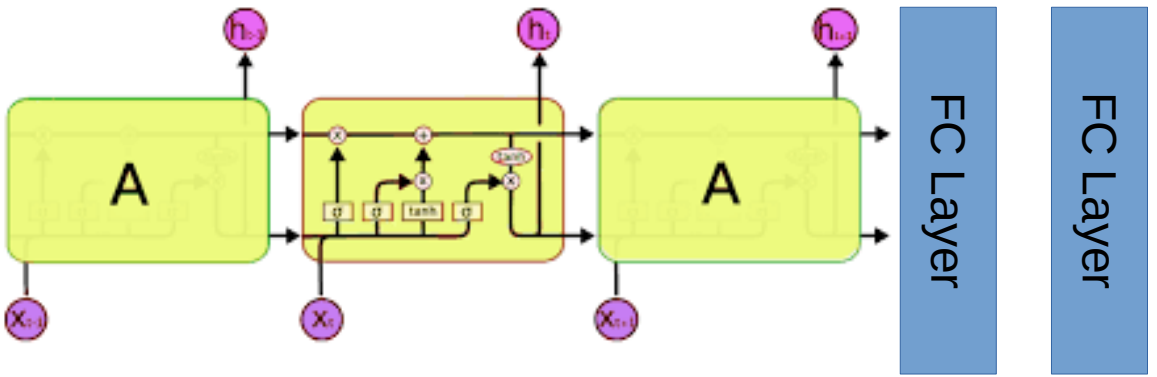
- Biggest DL success stories:
  - Computer vision (CV)
  - Natural language processing (NLP)

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?*

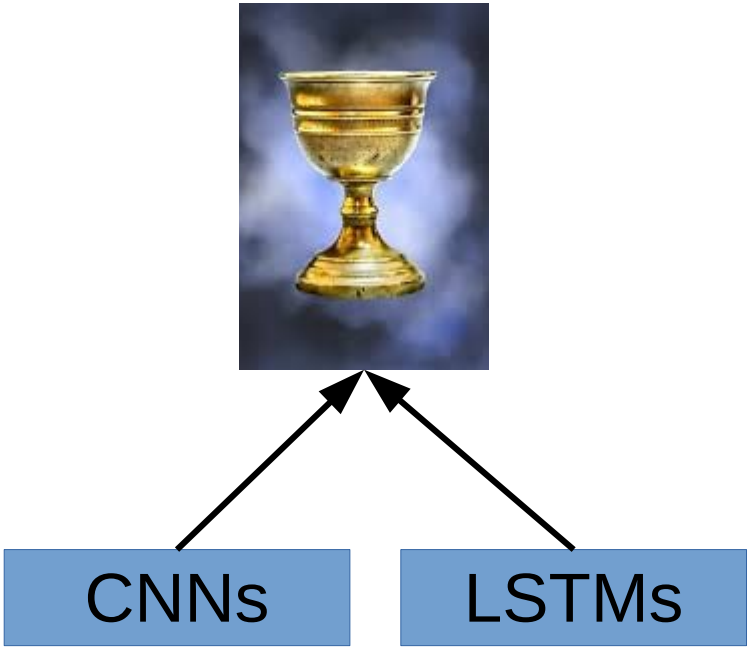
- To build a NN to *recognize objects in images*, we modeled the low-level structure used by *the human eye*.
  - LeCun et al., and indirectly, a Turing award
- To build a NN to *recognize words in speech,* we modeled the low-level structure used by *the human prefrontal cortex*.
  - Graves et al.

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

INDUCTIVE BIAS

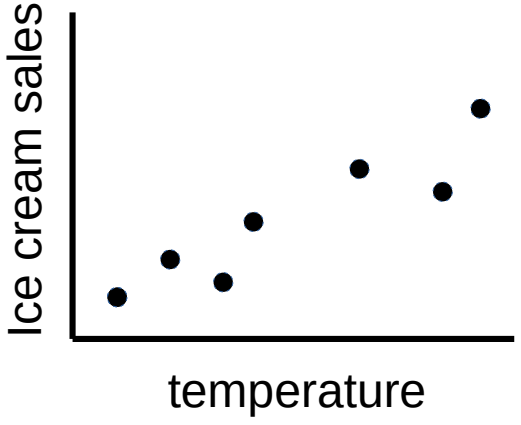
What makes DL *good?*

- 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?*

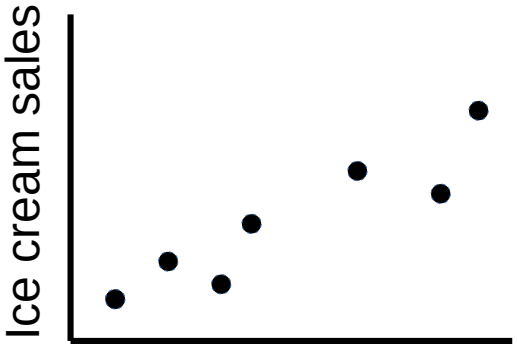
- “But wait! Hornik et al. showed that fully-connected layers can represent *any arbitrary function*!” - Someone not good at deep learning.



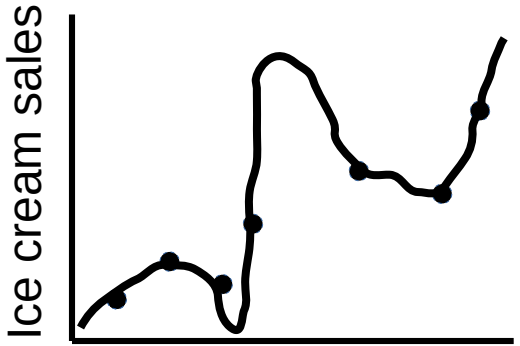
Original Data

What makes DL *good?*

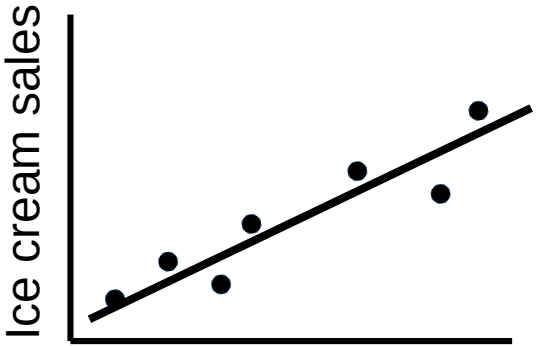
- “But wait! Hornik et al. showed that fully-connected layers can represent *any arbitrary function*!” - Someone not good at deep learning.



Original Data



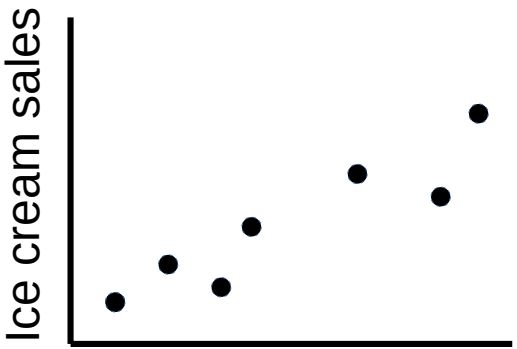
Arbitrary Function A



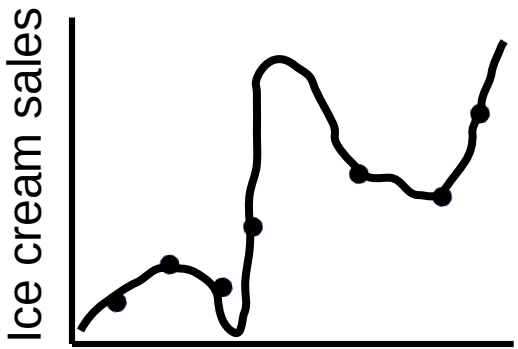
Arbitrary Function B

What makes DL *good?*

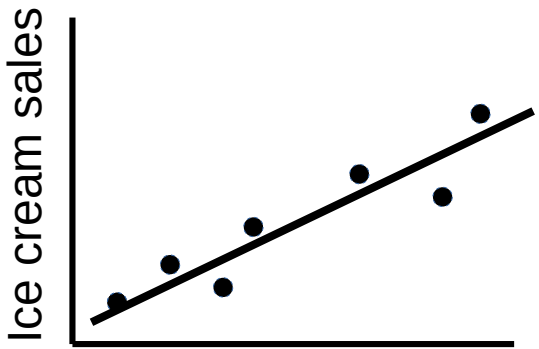
- You don’t want an *arbitrary* function that fits.
- You want a *generalizable* function that fits.



Original Data



Arbitrary Function A



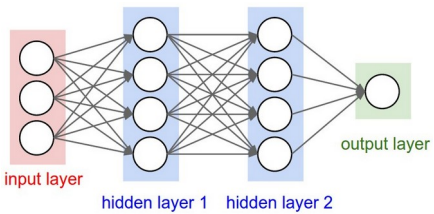
Arbitrary Function B

What makes DL *good?*

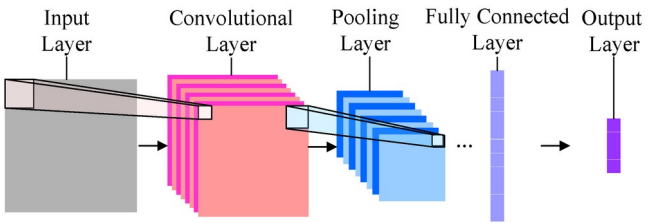
- Old school ML: regularization. Deep learning: just SGD!
- How do we know if our function will generalize?
- Inductive bias
  - One way: model it after a (biological) expert system that generalizes.
  - Generally: *constrain* the type of function that can be learned based on *prior knowledge* of the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

Inductive Bias

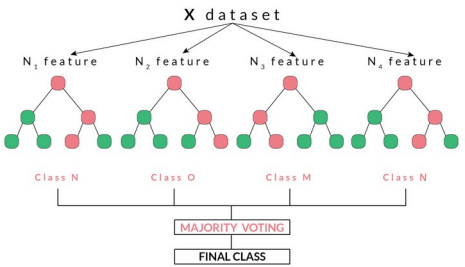
- Rank MNIST performance of these algorithms.



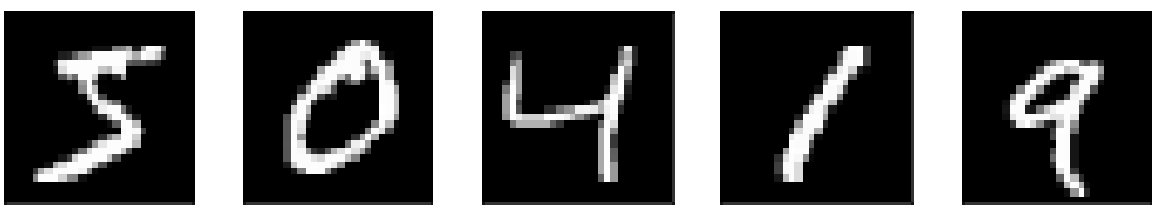
Fully-connected NN



Convolution NN



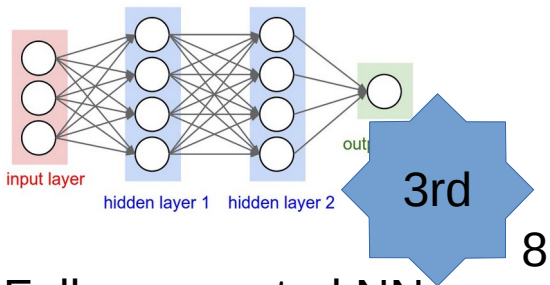
Random Forest



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

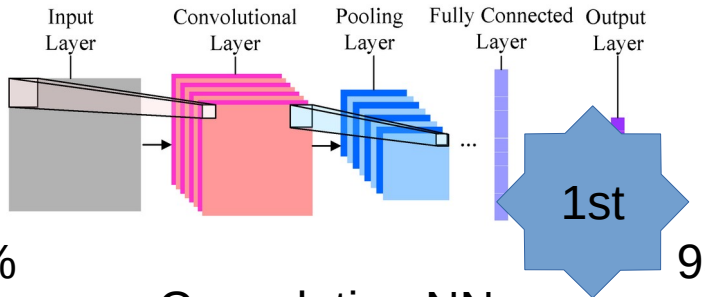
Inductive Bias

- Rank MNIST performance of these algorithms.



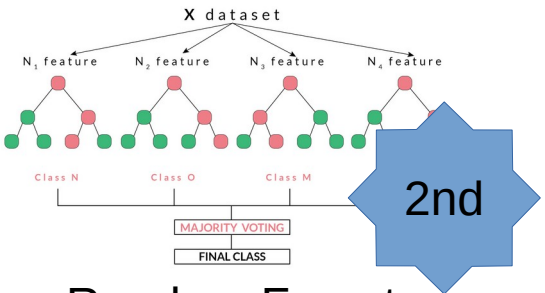
86%

Fully-connected NN



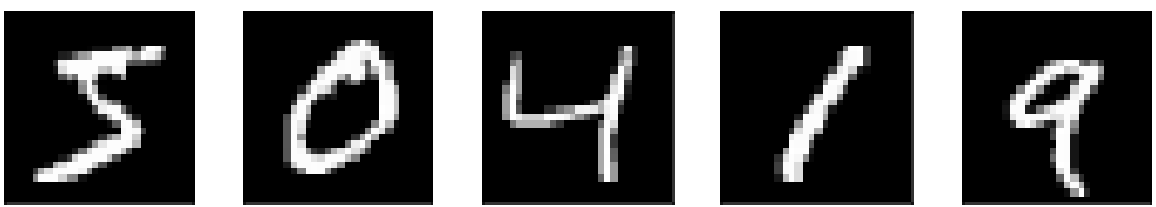
98%

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

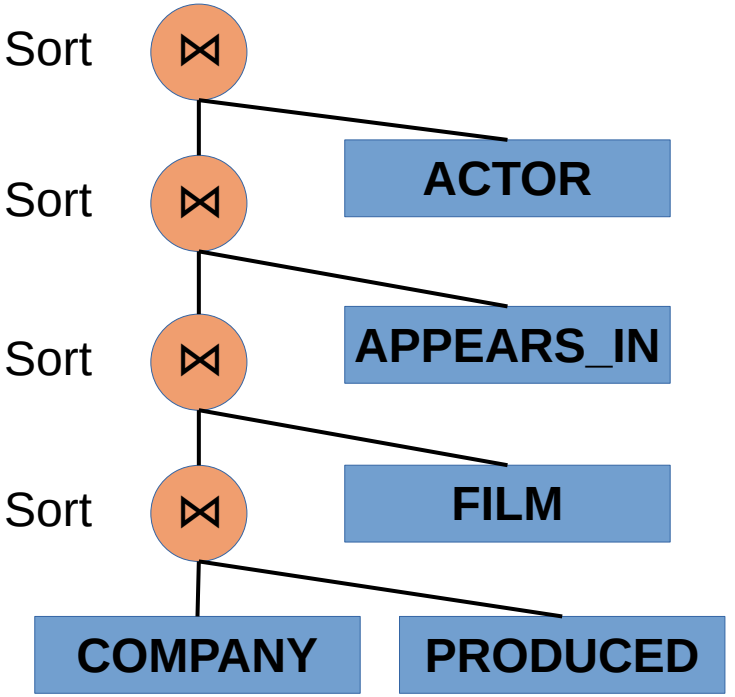
- Deep learning hype
  - End-to-end
  - Automatic feature engineering
  - Great generalization
- Deep learning reality:
  - All these things.
  - **BUT** you have to get the model architecture right!

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!

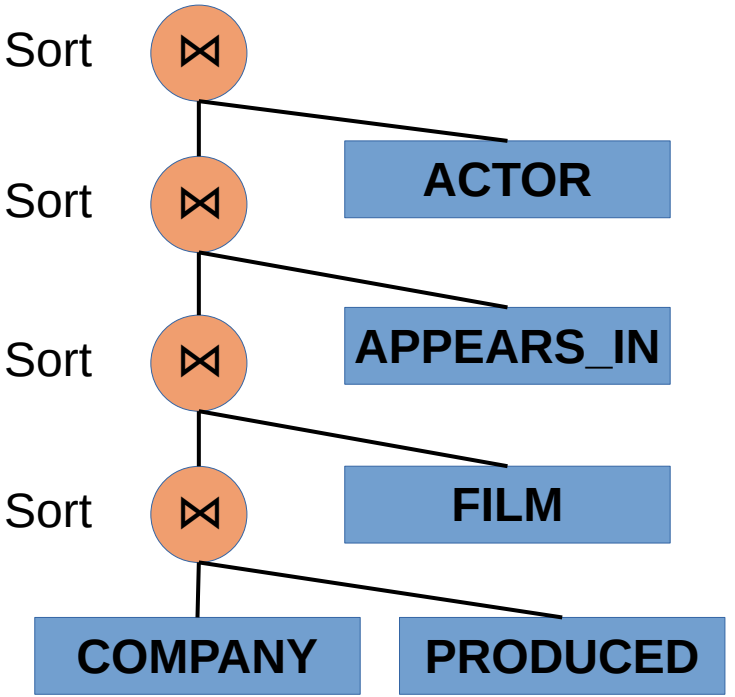
Tree Convolution

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



Tree Convolution

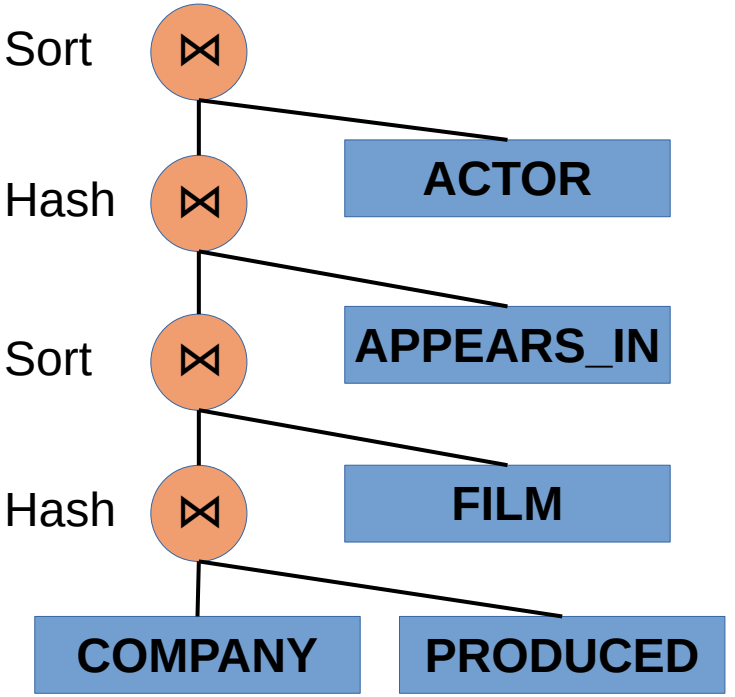
- How do we come up with a good inductive bias for query plans?



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

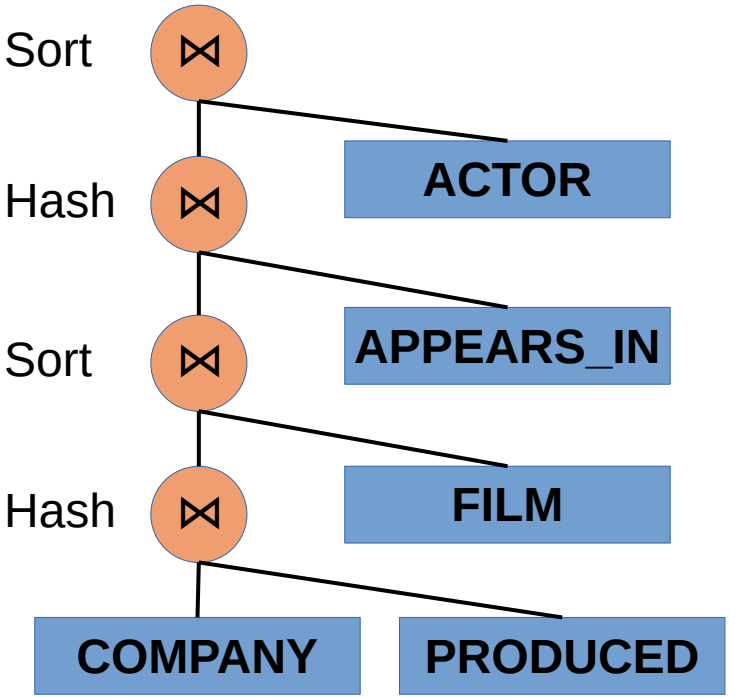
Tree Convolution

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



Tree Convolution

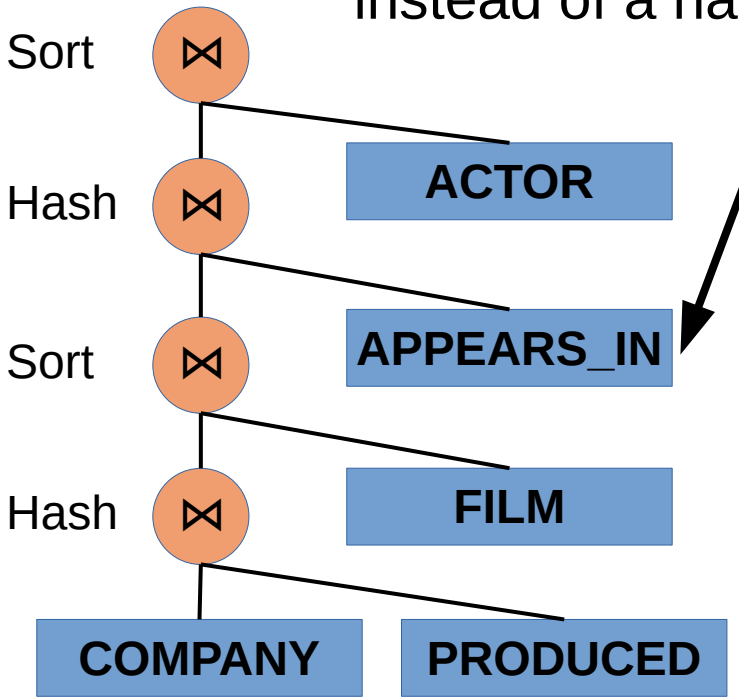
- How do we come up with a good inductive bias for query plans?



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

Tree Convolution

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

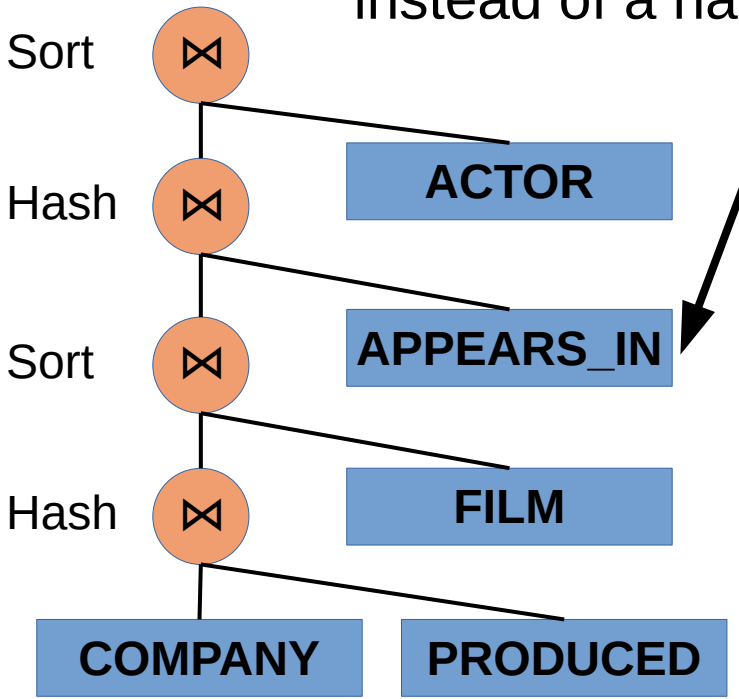


"APPEARS_IN" is presorted on disk – should use a sort instead of a hash.

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

Tree Convolution

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

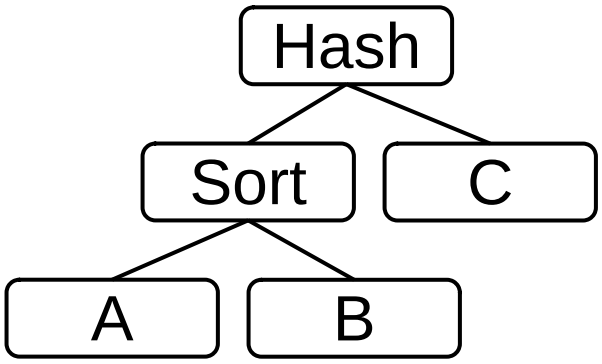


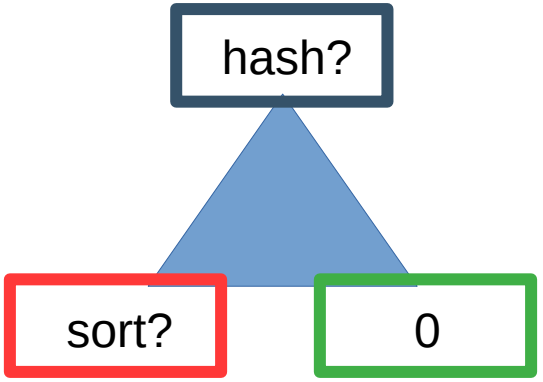
"APPEARS_IN" is presorted on disk – should use a sort instead of a hash.

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

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

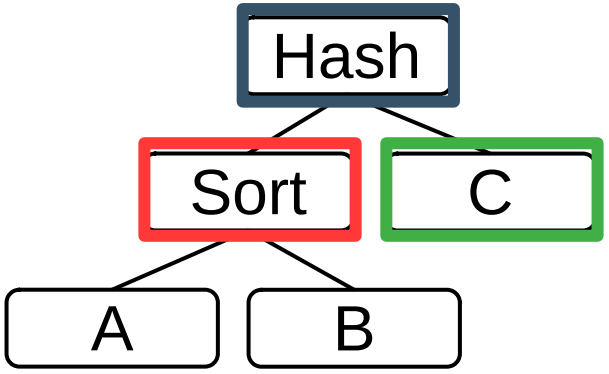
Tree Convolution

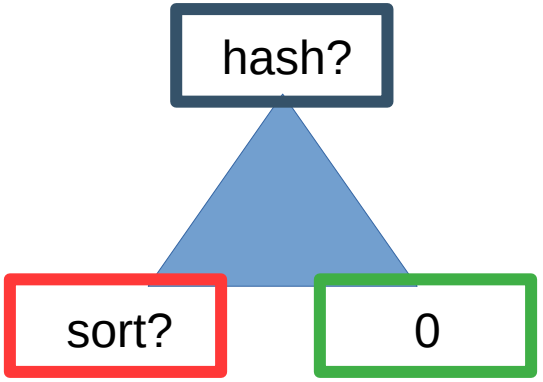




Detects a hash join on top of a sort join

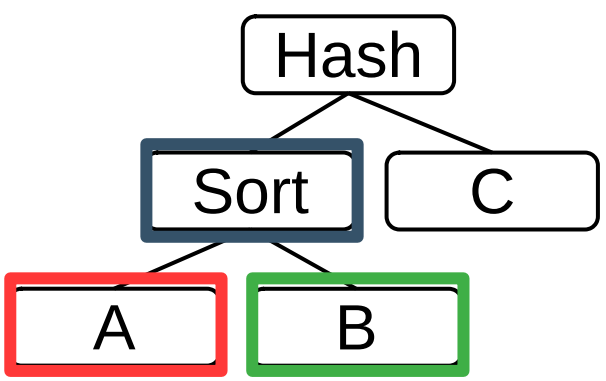
Tree Convolution

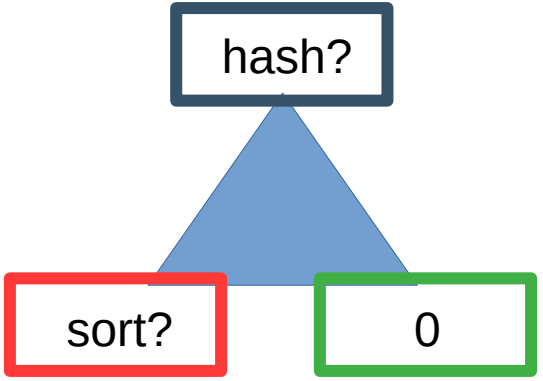


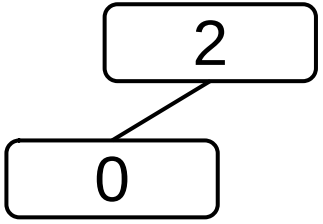


Detects a hash join on top of a sort join

Tree Convolution

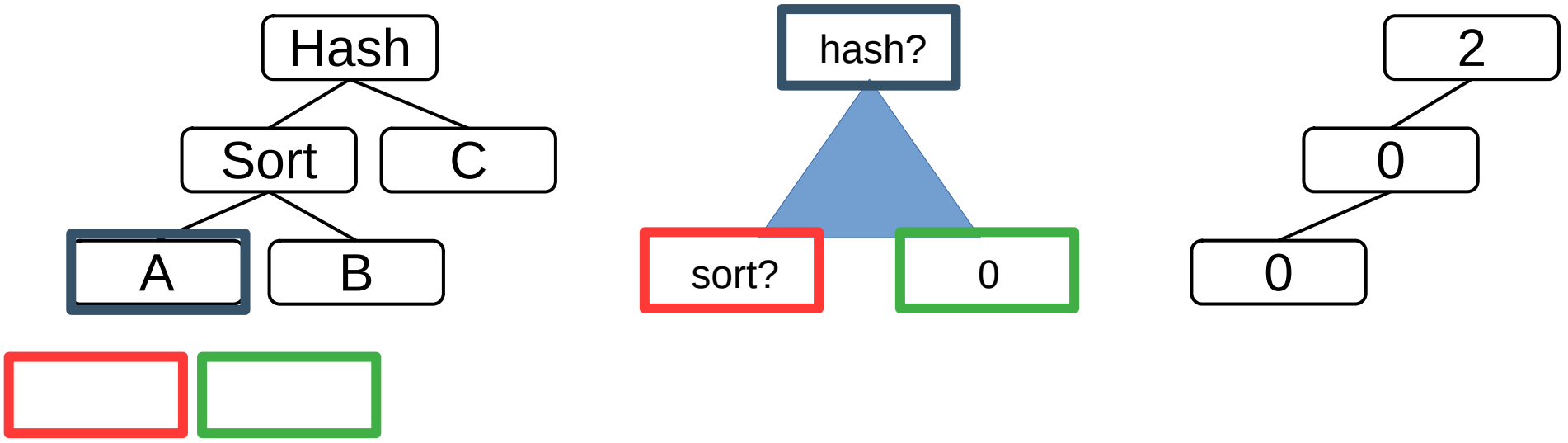






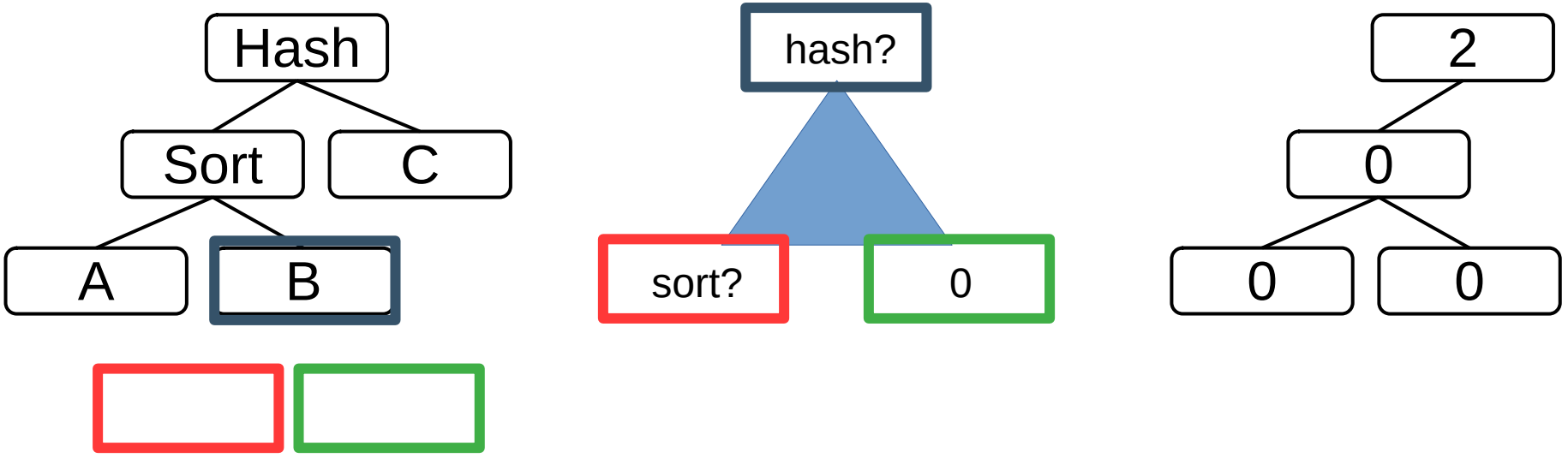
Detects a hash join on top of a sort join

Tree Convolution



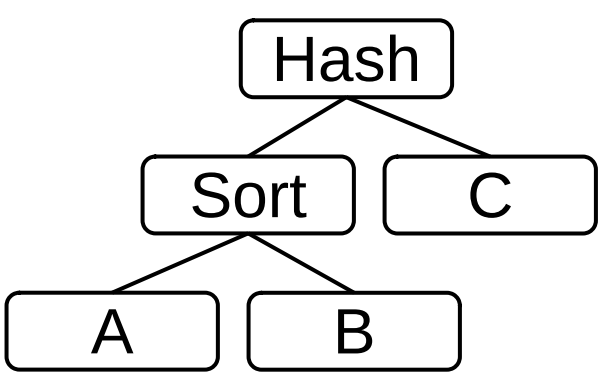
Detects a hash join on top of a sort join

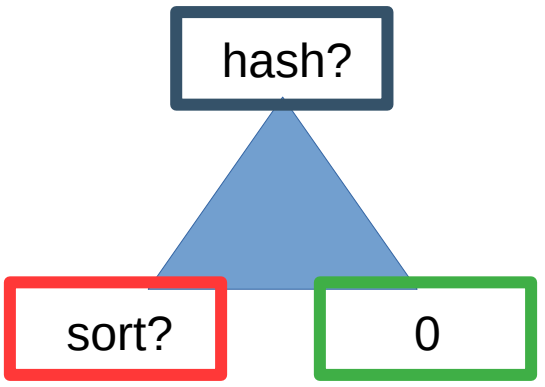
Tree Convolution

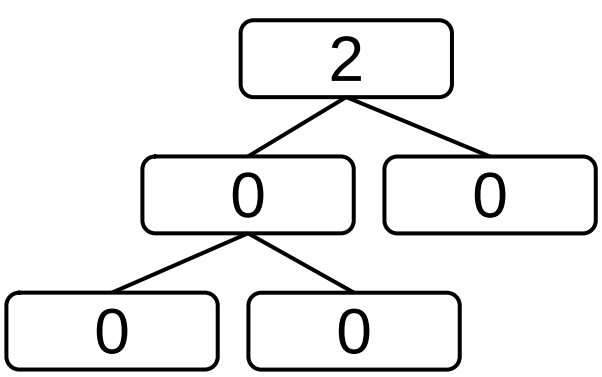


Detects a hash join on top of a sort join

Tree Convolution

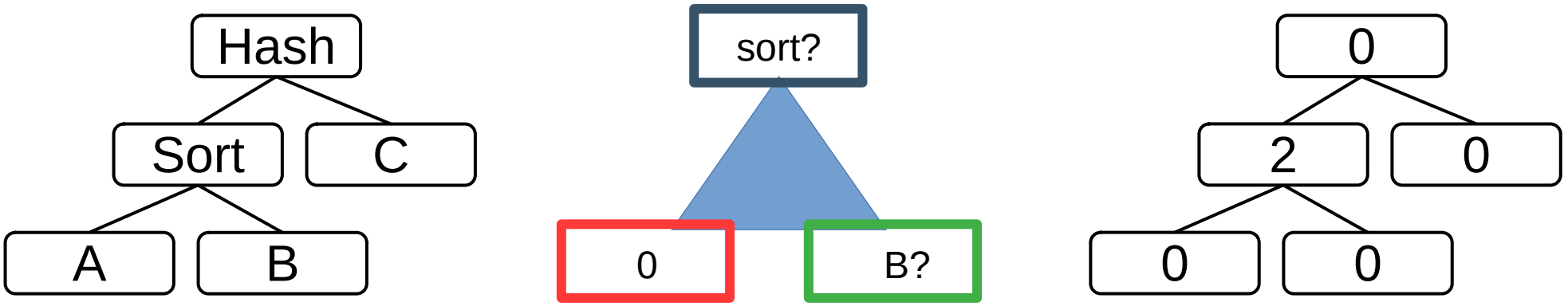






Detects a hash join on top of a sort join

Tree Convolution



Detects a sort-merge join with B on RHS

Tree Convolution

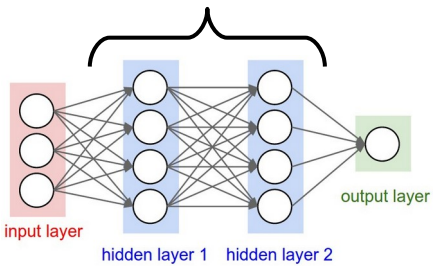
- Stackable / composable, just like image conv
- We've built an end-to-end, RL-powered optimizer around it
  - Neo: NEural Optimizer
  - See the talk on Thursday!
- *Almost any* tree convolution filter will be *close to* a semantically-relevant pattern

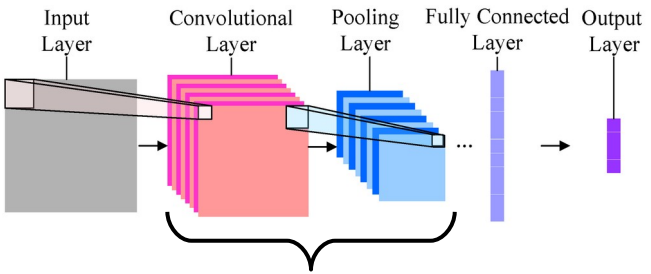
Random Weights Test

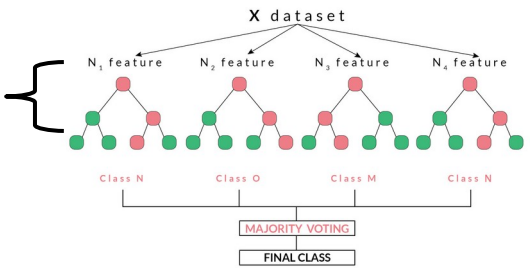
- Testing for inductive bias
  - “But Ryan, how do I know if a particular network architecture has a good inductive bias?” - Friendly advisor
  - “You can’t just assert your architecture has a good inductive bias and other systems do not! Where is the test, you bumbling fool?” - Reviewer #2
- Sufficient but not necessary condition:
  - The **random weights test**

Random Weights Test

- **The random weight test.**
  - Take your model, randomly initialize all weights but the last “layer.” Test performance.







sufficient but **not necessary**

Random Weights Test

	Fully Conn.	Conv	Random Forest
Only last layer	??????		
All layers	86%	98%	92%

(test accuracy, MNIST)

sufficient but **not necessary**

Random Weights Test

	Fully Conn.	Conv	Random Forest
Only last layer	45%	91%	74%
All layers	86%	98%	92%

(test accuracy, MNIST)

sufficient but **not necessary**

Random Weights Test

	Fully Conn.	Conv	Random Forest
Only last layer	45%	91%	74%
All layers	86%	98%	92%

(test accuracy, MNIST)

Big gap: bad bias.

Small gap: good bias.

sufficient but **not necessary**

Random Weights Test

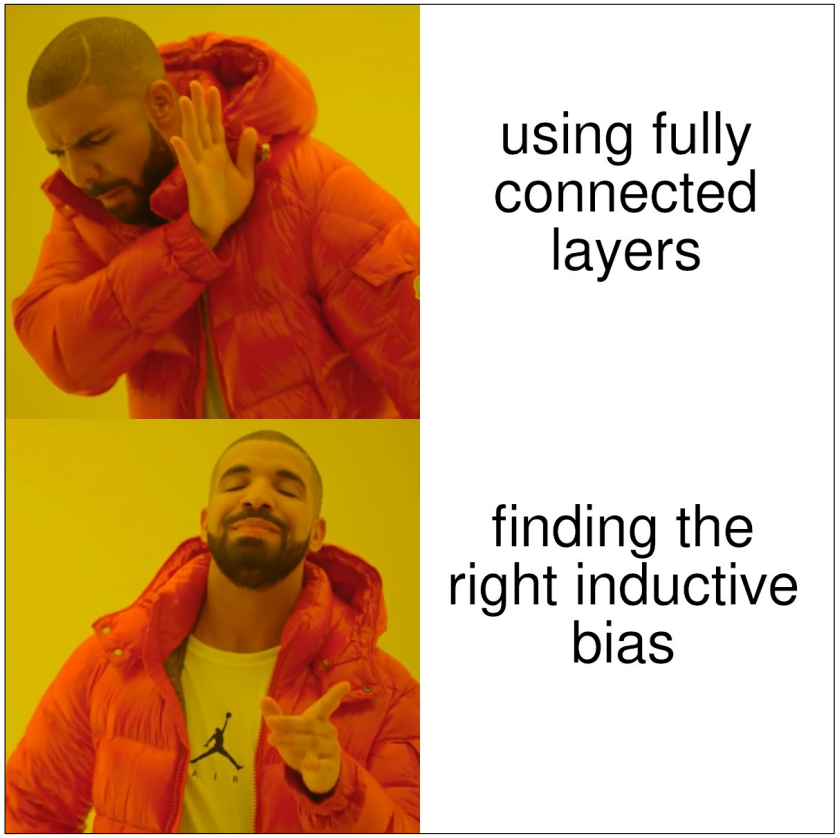
	Fully Conn.	Conv	Random Forest
Only last layer	45%	91%	74%
All layers	86%	98%	92%

(test accuracy, MNIST)

Even **randomly initialized convolution** layers **perform better** than **trained fully connected** layers!

sufficient but **not necessary**

Inductive Bias



- Deep learning success in CV and NLP due to **inductive bias**
- Fully-connected generality is normally **bad**
- Finding the right inductive bias for your problem → **huge performance gain**