Cortex: A Compiler For Recursive Deep Learning Models

Pratik Fegade[1],
Tianqi Chen[1,2], Phillip B. Gibbons[1], Todd C. Mowry[1]

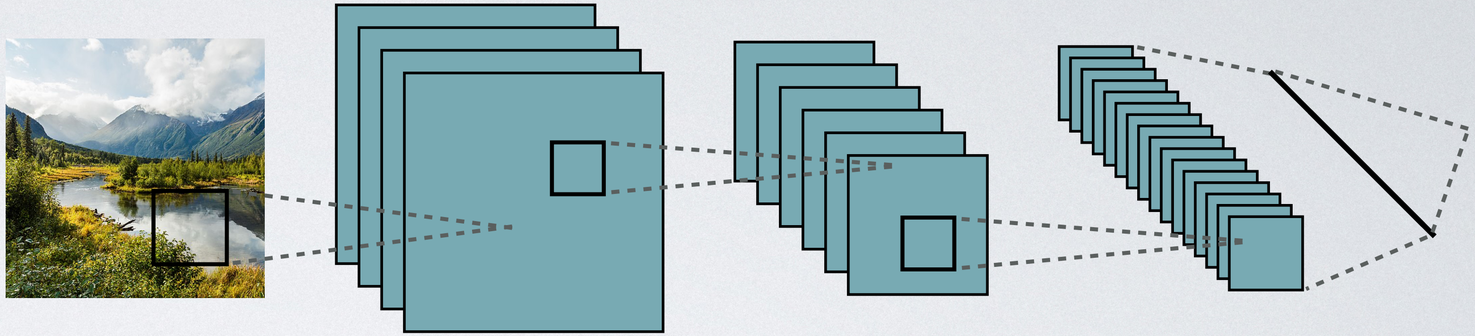
[1]Carnegie Mellon University [2]OctoML

Carnegie Mellon University
Computer Science Department

catalyst

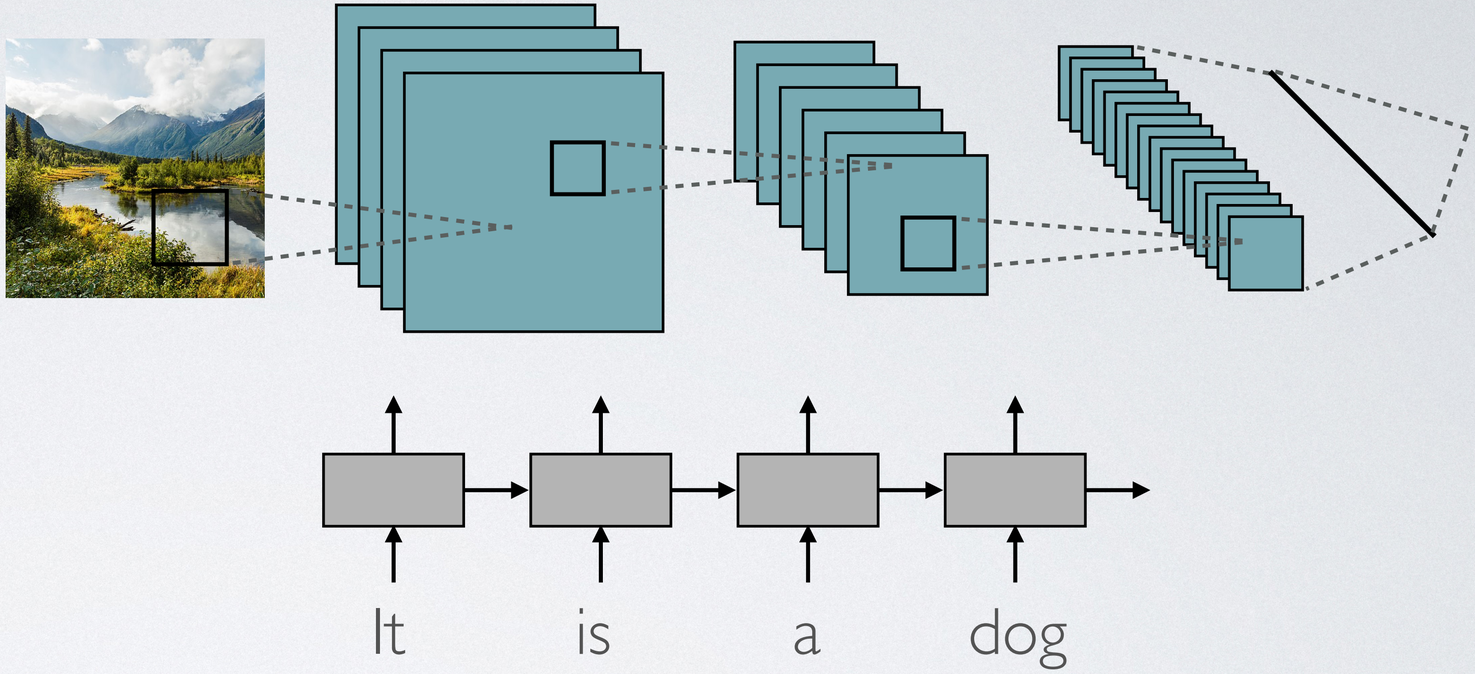
Deep Learning Models Often Mirror Input Structure

- Convolutional neural networks



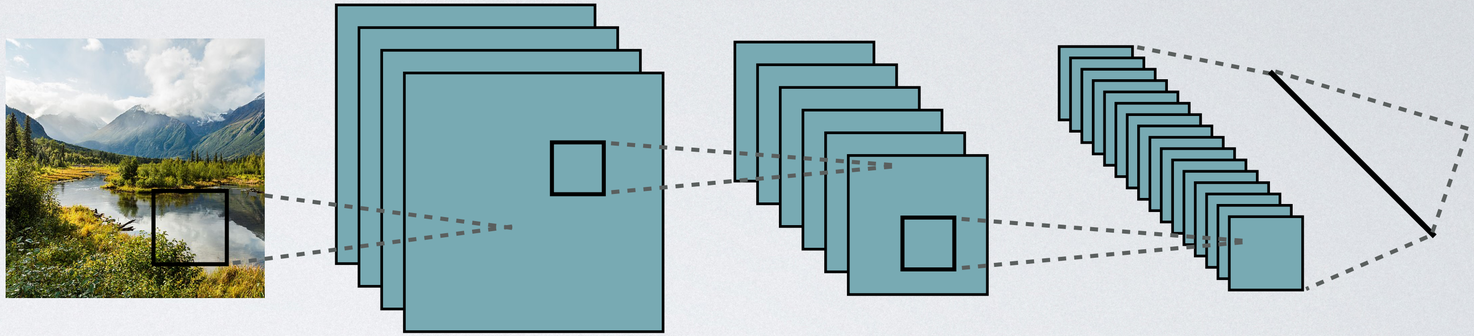
Deep Learning Models Often Mirror Input Structure

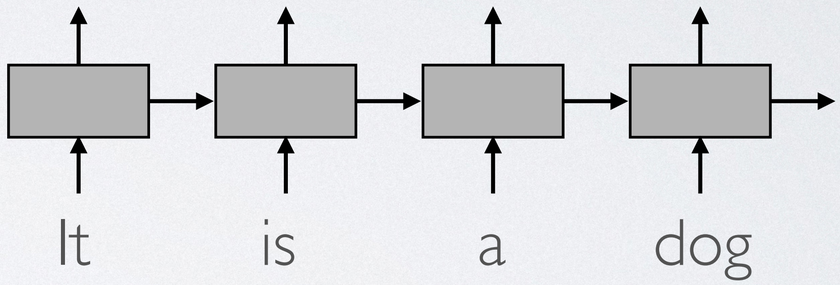
- Convolutional neural networks
- Recurrent neural networks

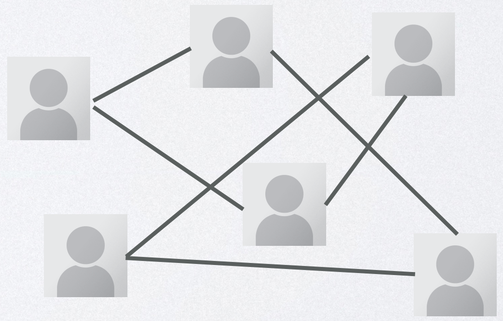


Deep Learning Models Often Mirror Input Structure

- Convolutional neural networks
- Recurrent neural networks
- Graph neural networks







Recursive Input Domains Lead to Recursive Models

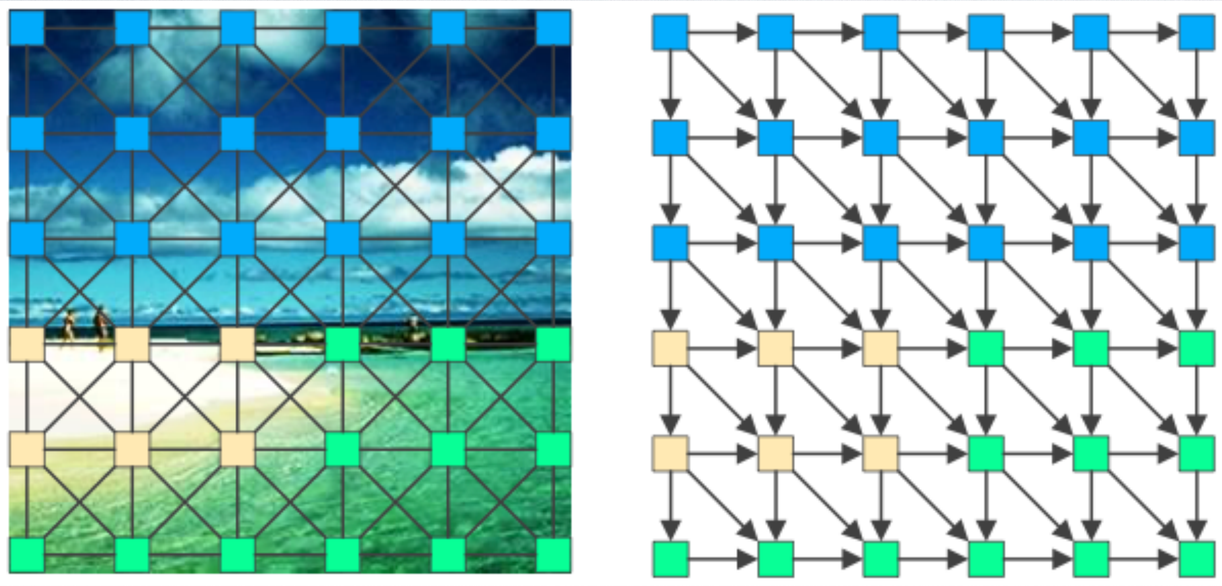


Image segmentation using DAG-RNN

Recursive Input Domains Lead to Recursive Models

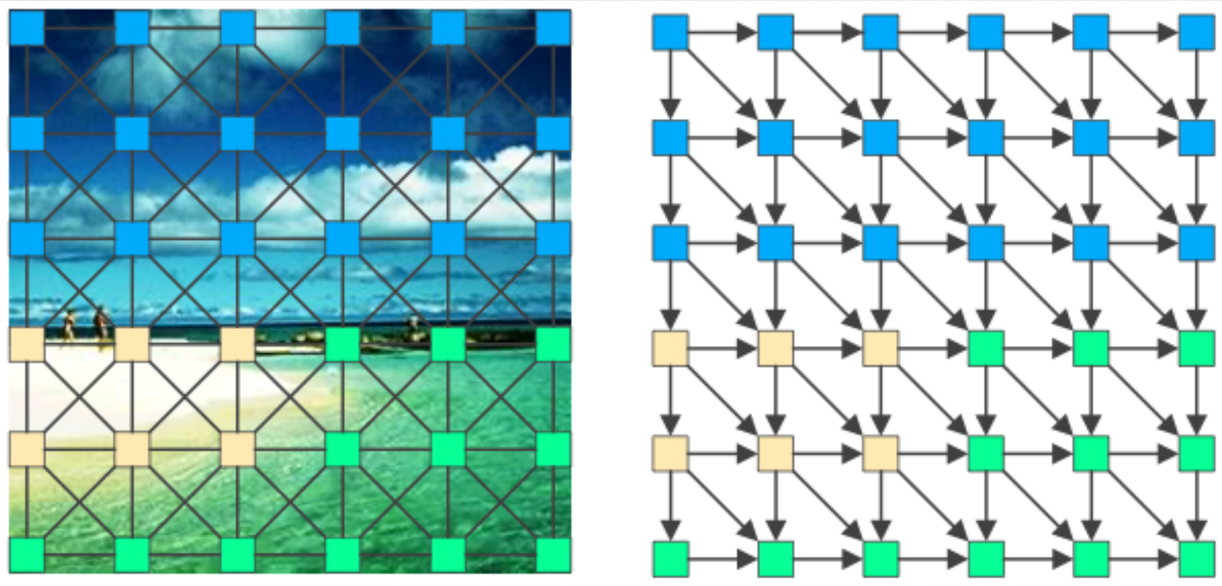
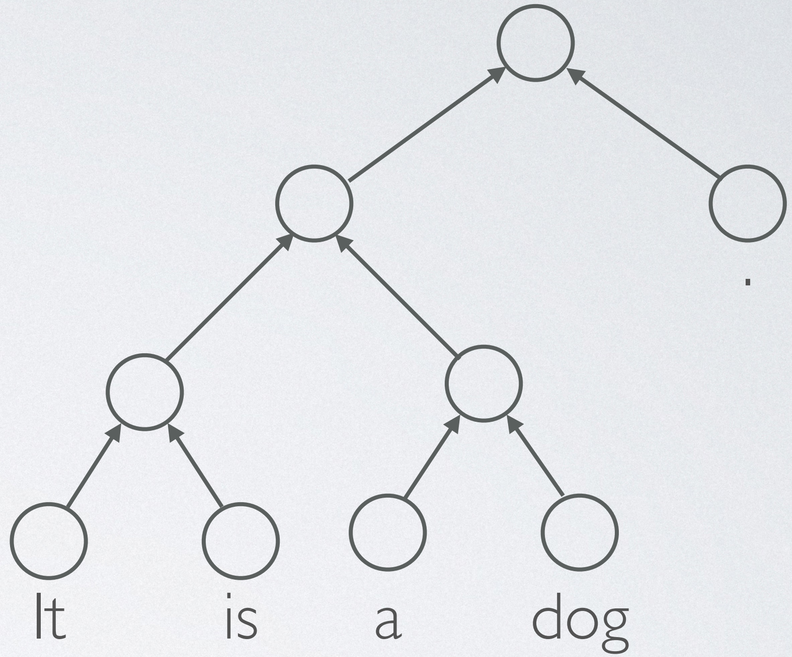


Image segmentation using DAG-RNN



Semantic text classification using TreeLSTM

Outline

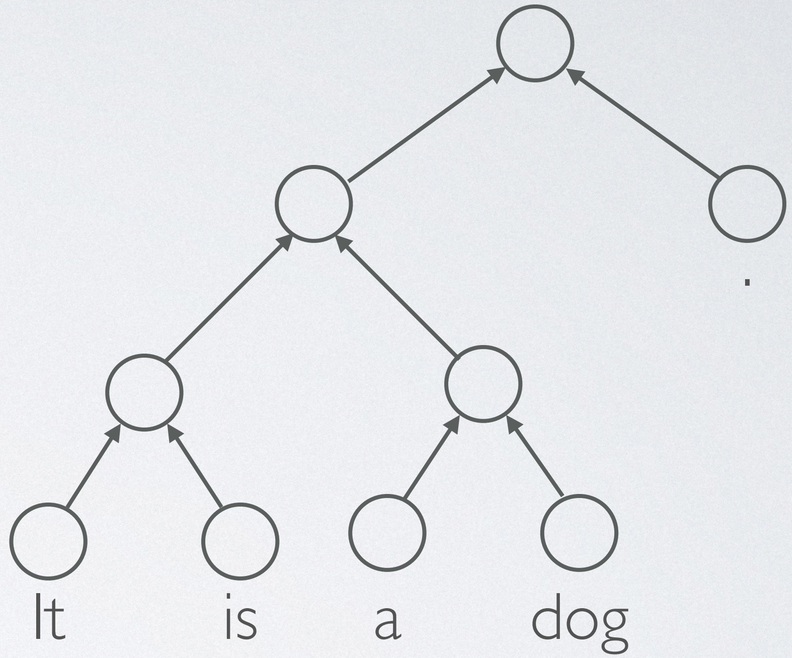
- Motivation: Inefficiencies in Execution of Recursive Models
- Cortex: Our Compiler Based Solution
  - Recursive Lowering
  - Loop IR Lowering
- Evaluation
- Conclusion

Outline

- **Motivation: Inefficiencies in Execution of Recursive Models**
- Cortex: Our Compiler Based Solution
  - Recursive Lowering
  - Loop IR Lowering
- Evaluation
- Conclusion

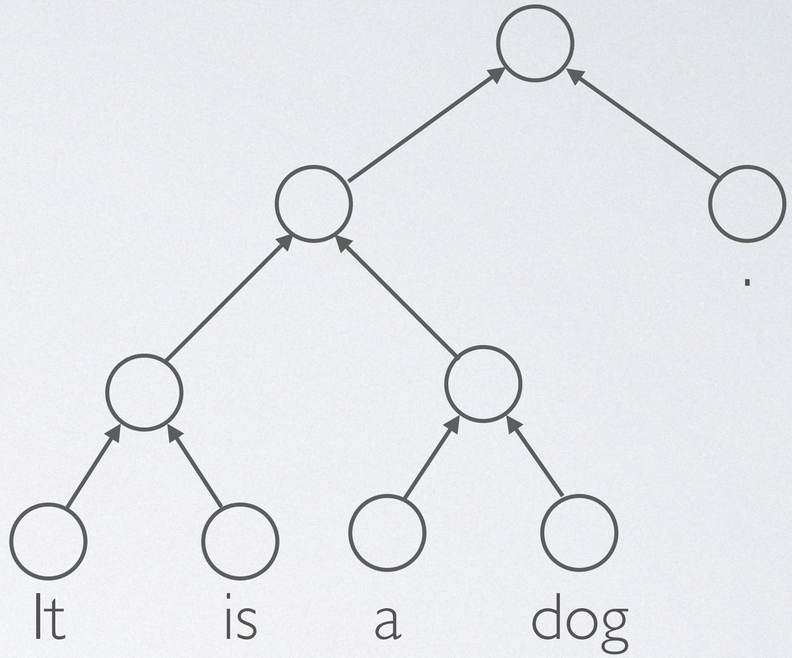
Running Example: Simple TreeFC Model

```
# lh, rh, Emb, words are tensors
def treeFC(n):
```



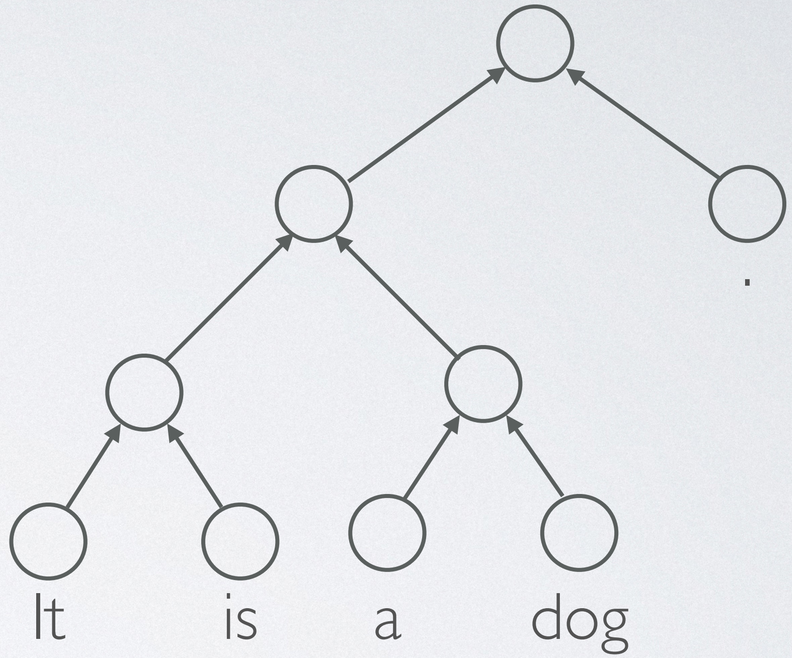
Running Example: Simple TreeFC Model

```
# lh, rh, Emb, words are tensors
def treeFC(n):
  if isleaf(n):
    return Emb[words[n]]
```



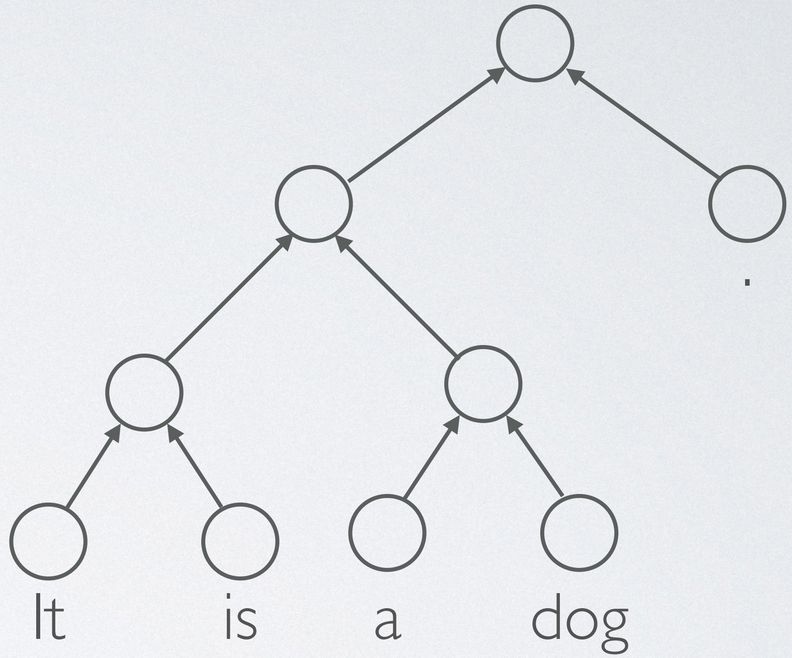
Running Example: Simple TreeFC Model

```
# lh, rh, Emb, words are tensors
def treeFC(n):
  if isleaf(n):
    return Emb[words[n]]
  else:
    lh = treeFC(n.left)
    rh = treeFC(n.right)
```



Running Example: Simple TreeFC Model

```
# lh, rh, Emb, words are tensors
def treeFC(n):
  if isleaf(n):
    return Emb[words[n]]
  else:
    lh = treeFC(n.left)
    rh = treeFC(n.right)
    return W * (lh + rh)
```



Current Frameworks: Interleaved Control Flow and Tensor Algebra

CPU execution GPU execution

```
def treeFC(n):
  if isleaf(n):
    return Emb[words[n]]
  else:
    lh = treeFC(n.left)
    rh = treeFC(n.right)
    add =  lh + rh
    return  W * add
```

Current Frameworks: Interleaved Control Flow and Tensor Algebra

CPU execution GPU execution

```
def treeFC(n):
  if isleaf(n):
    return Emb[words[n]]
  else:
    lh = treeFC(n.left)
    rh = treeFC(n.right)
    add =  lh + rh
    return  W * add
```

Current Frameworks: Interleaved Control Flow and Tensor Algebra

CPU execution GPU execution

```
def treeFC(n):
  if isleaf(n):
    return Emb[words[n]]
  else:
    lh = treeFC(n.left)
    rh = treeFC(n.right)
    add =  lh + rh
    return  W * add
```

Current Frameworks: Interleaved Control Flow and Tensor Algebra

CPU execution GPU execution

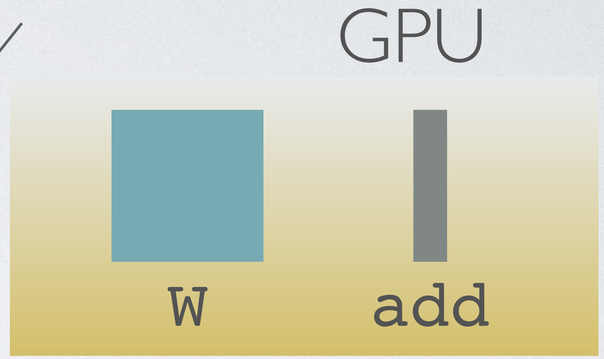
```
def treeFC(n):
  if isleaf(n):
    return Emb[words[n]]
  else:
    lh = treeFC(n.left)
    rh = treeFC(n.right)
    add = lh + rh
    return W * add
```

Current Frameworks: Interleaved Control Flow and Tensor Algebra

CPU execution GPU execution

```
def treeFC(n):
  if isleaf(n):
    return Emb[words[n]]
  else:
    lh = treeFC(n.left)
    rh = treeFC(n.right)
    add = lh + rh
    return W * add
```

cuDNN
kernel call



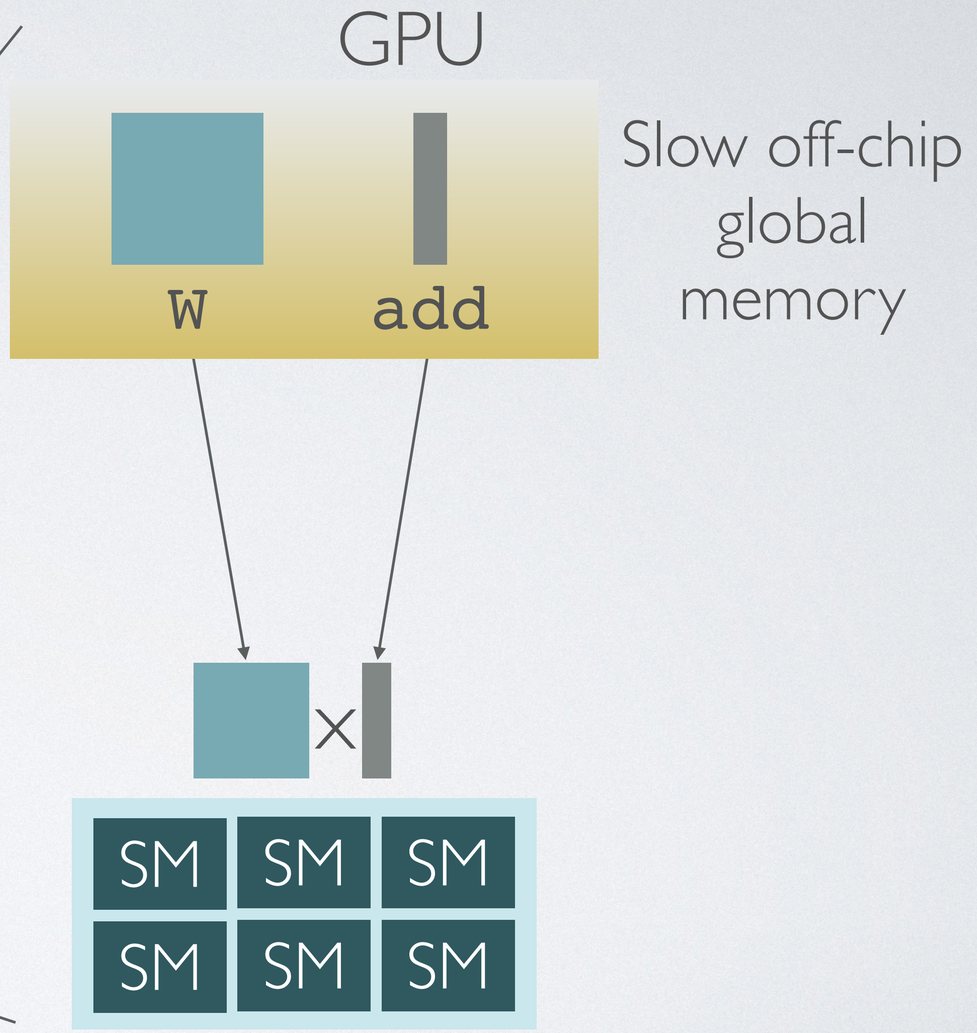
Slow off-chip
global
memory

Current Frameworks: Interleaved Control Flow and Tensor Algebra

CPU execution GPU execution

```
def treeFC(n):
  if isleaf(n):
    return Emb[words[n]]
  else:
    lh = treeFC(n.left)
    rh = treeFC(n.right)
    add = lh + rh
    return W * add
```

cuDNN
kernel call



Desired Execution With Aggressive Kernel Fusion

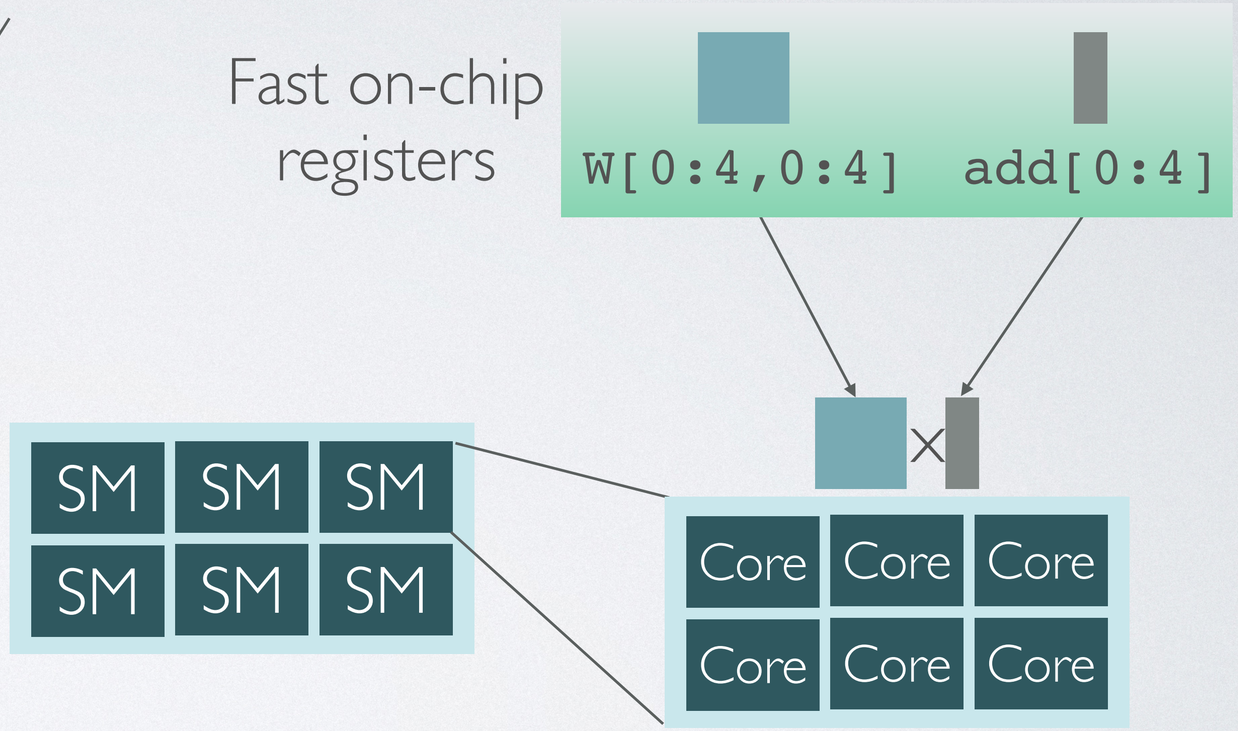
CPU execution GPU execution

```
def treeFC(n):
  if isleaf(n):
    return Emb[words[n]]
  else:
    lh = treeFC(n.left)
    rh = treeFC(n.right)
    add =  lh + rh
    return  W * add
```

Desired Execution With Aggressive Kernel Fusion

CPU execution GPU execution

```
def treeFC(n):
  if isleaf(n):
    return Emb[words[n]]
  else:
    lh = treeFC(n.left)
    rh = treeFC(n.right)
    add =  lh + rh
    return  W * add
```



Challenges With Aggressive Kernel Fusion

- Executing recursive control flow efficiently on accelerators

Challenges With Aggressive Kernel Fusion

- Executing recursive control flow efficiently on accelerators
- Exploit the data reuse effectively

Challenges With Aggressive Kernel Fusion

- Executing recursive control flow efficiently on accelerators
- Exploit the data reuse effectively
- Can no longer use vendor libraries

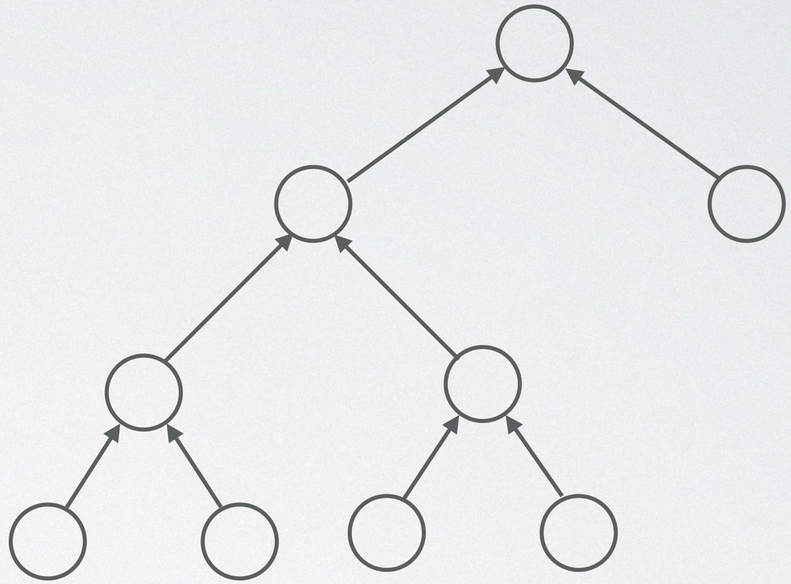
Outline

- Motivation: Inefficiencies in Execution of Recursive Models
- **Cortex: Our Compiler Based Solution**
  - Recursive Lowering
  - Loop IR Lowering
- Evaluation
- Conclusion

Control Flow Depends Only on Input Structure

- Starting from the leaves, move up the tree towards the root

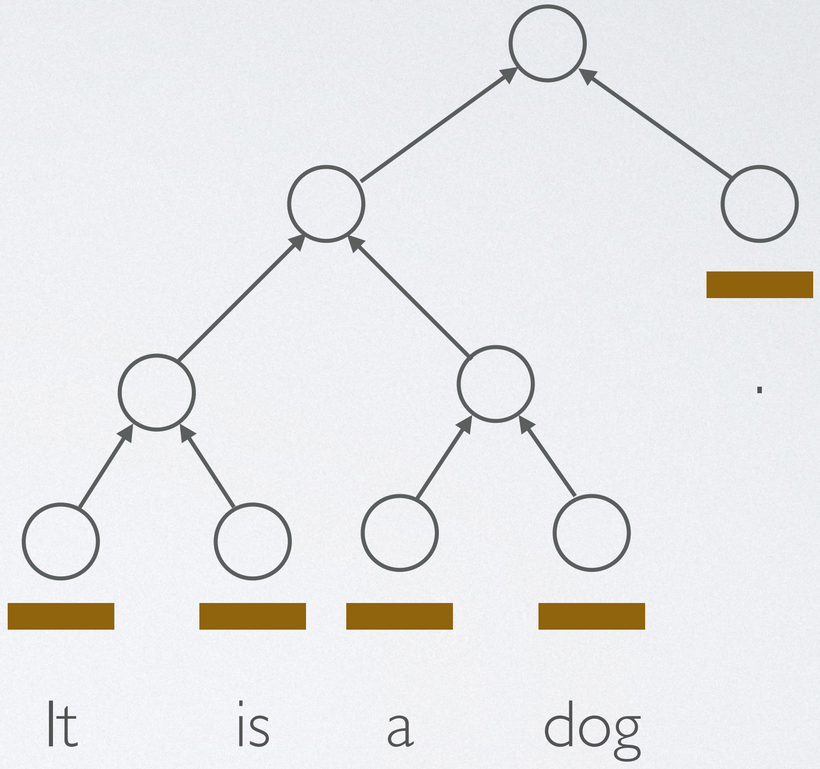
```
# lh, rh, Emb, W are tensors
def treeFC(n):
  if isleaf(n):
    return Emb[words[n]]
  else:
    lh = treeFC(n.left)
    rh = treeFC(n.right)
    return W*(lh + rh)
```



Control Flow Depends Only on Input Structure

- Starting from the leaves, move up the tree towards the root

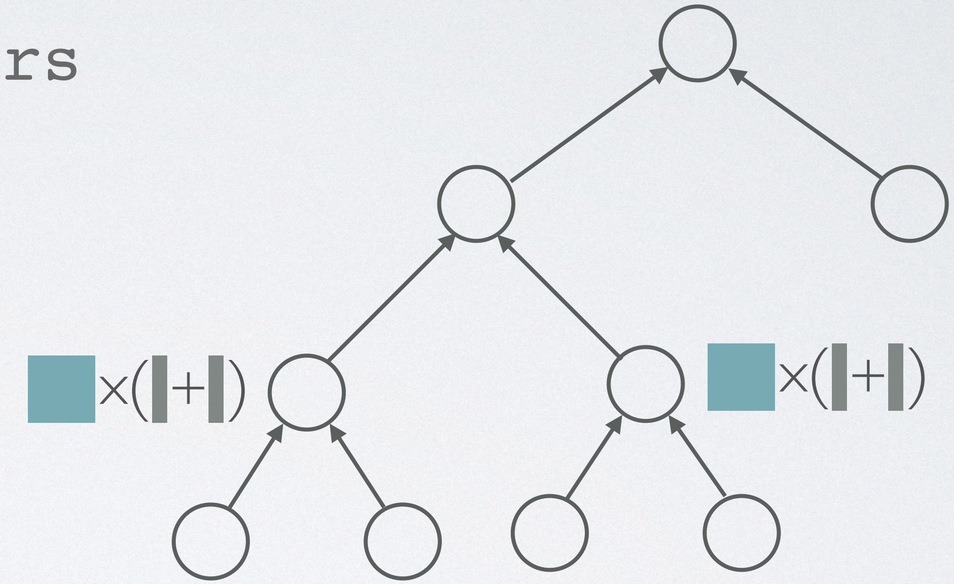
```
# lh, rh, Emb, W are tensors
def treeFC(n):
  if isleaf(n):
    return Emb[words[n]]
  else:
    lh = treeFC(n.left)
    rh = treeFC(n.right)
    return W*(lh + rh)
```



Control Flow Depends Only on Input Structure

- Starting from the leaves, move up the tree towards the root

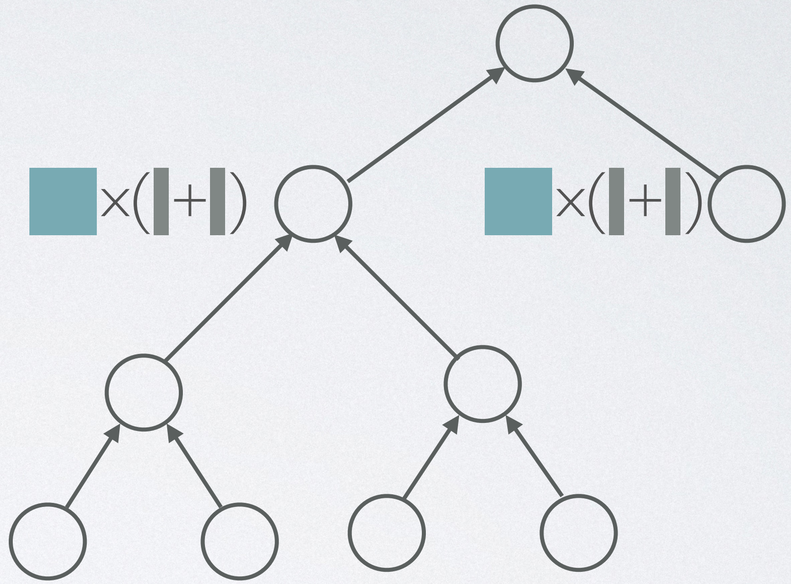
```
# lh, rh, Emb, W are tensors
def treeFC(n):
  if isleaf(n):
    return Emb[words[n]]
  else:
    lh = treeFC(n.left)
    rh = treeFC(n.right)
    return W*(lh + rh)
```



Control Flow Depends Only on Input Structure

- Starting from the leaves, move up the tree towards the root

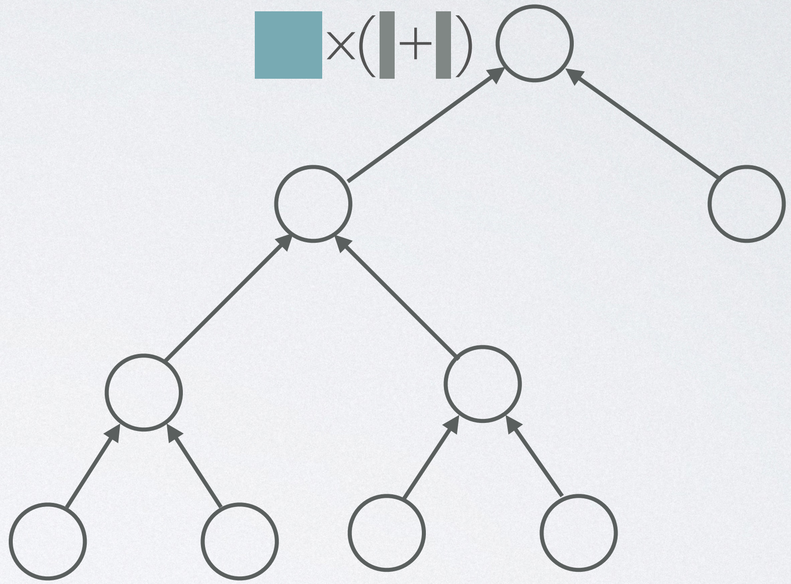
```
# lh, rh, Emb, W are tensors
def treeFC(n):
  if isleaf(n):
    return Emb[words[n]]
  else:
    lh = treeFC(n.left)
    rh = treeFC(n.right)
    return W*(lh + rh)
```



Control Flow Depends Only on Input Structure

- Starting from the leaves, move up the tree towards the root

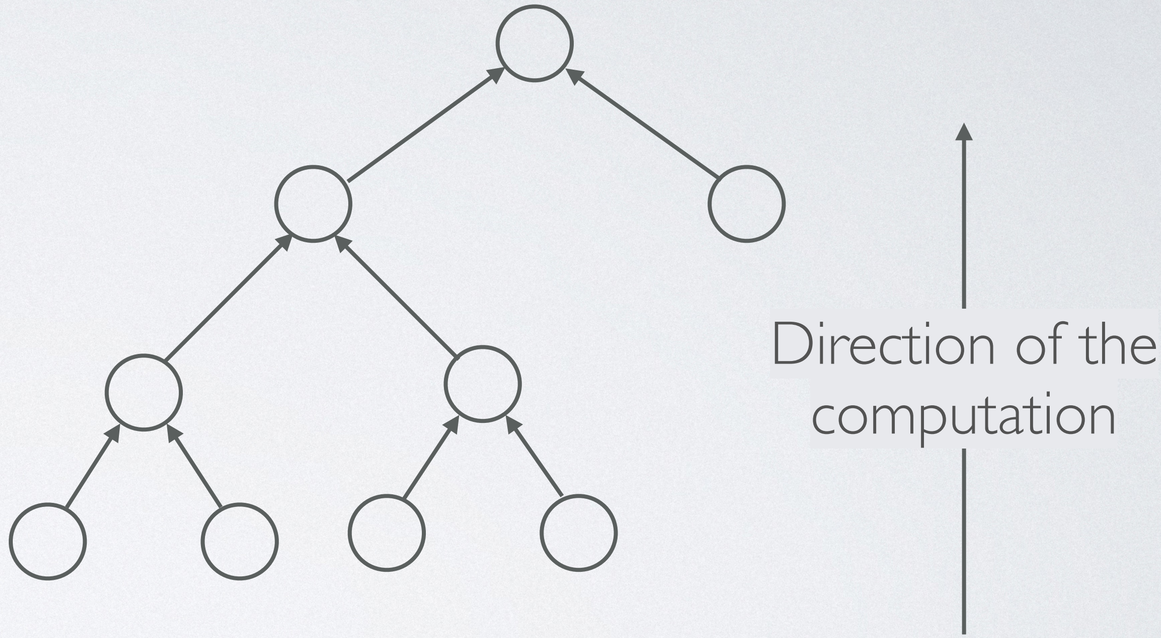
```
# lh, rh, Emb, W are tensors
def treeFC(n):
  if isleaf(n):
    return Emb[words[n]]
  else:
    lh = treeFC(n.left)
    rh = treeFC(n.right)
    return W*(lh + rh)
```



Control Flow Depends Only on Input Structure

- Starting from the leaves, move up the tree towards the root

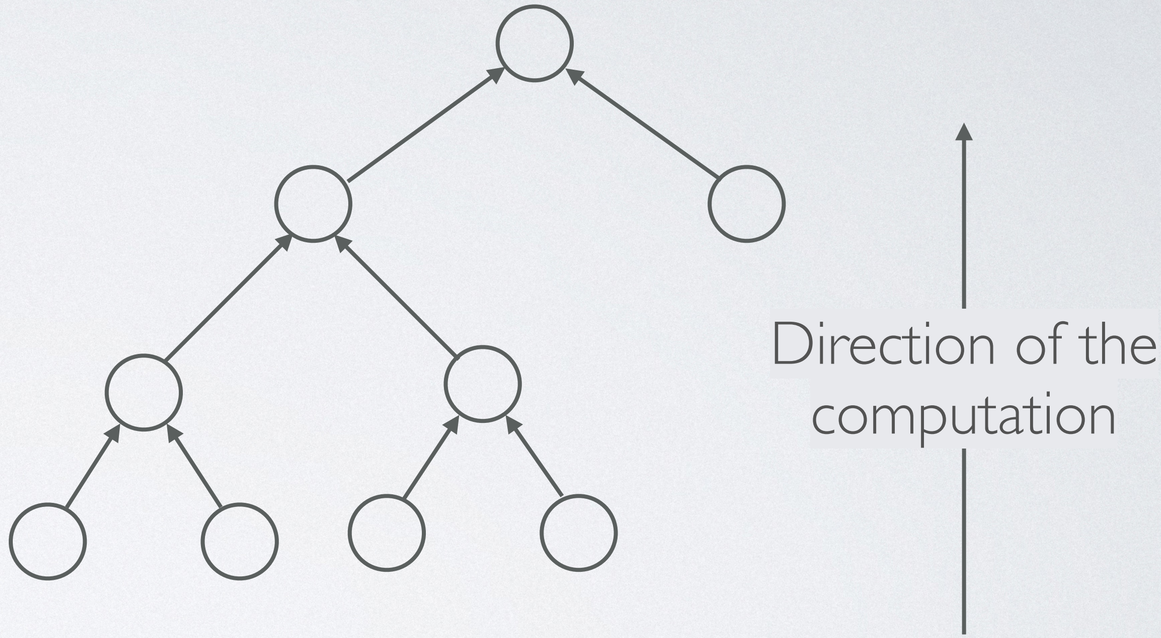
```
# lh, rh, Emb, W are tensors
def treeFC(n):
  if isleaf(n):
    return Emb[words[n]]
  else:
    lh = treeFC(n.left)
    rh = treeFC(n.right)
    return W*(lh + rh)
```



Control Flow Depends Only on Input Structure

- Starting from the leaves, move up the tree towards the root

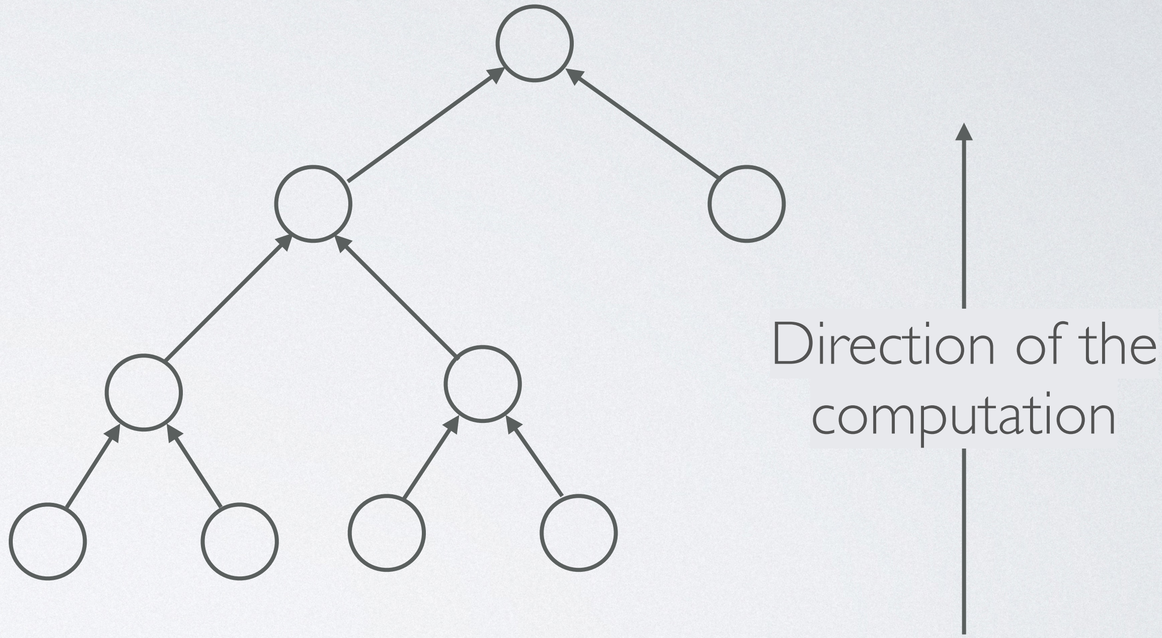
```
# lh, rh, Emb, W are tensors
def treeFC(n):
    if isleaf(n):
        return Emb[words[n]]
    else:
        lh = treeFC(n.left)
        rh = treeFC(n.right)
        return W*(lh + rh)
```



Control Flow Depends Only on Input Structure

- Starting from the leaves, move up the tree towards the root

```
# lh, rh, Emb, W are tensors
def treeFC(n):
  if isleaf(n):
    return Emb[words[n]]
  else:
    lh = treeFC(n.left)
    rh = treeFC(n.right)
    return W*(lh + rh)
```



Data Structure Linearization in Cortex

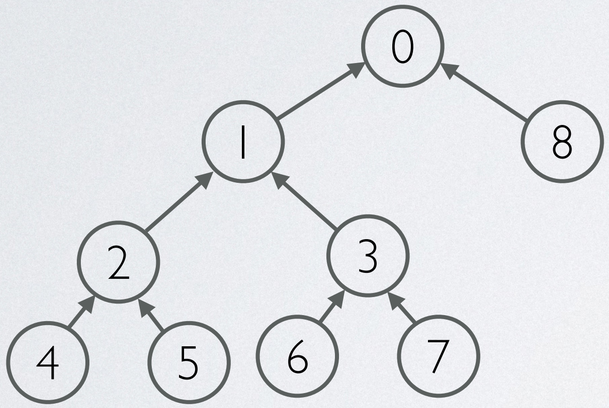
- Control flow depends only on the input structure

Data Structure Linearization in Cortex

- Control flow depends only on the input structure
- Given the input structure, node processing order is fixed

Data Structure Linearization in Cortex

- Control flow depends only on the input structure
- Given the input structure, node processing order is fixed

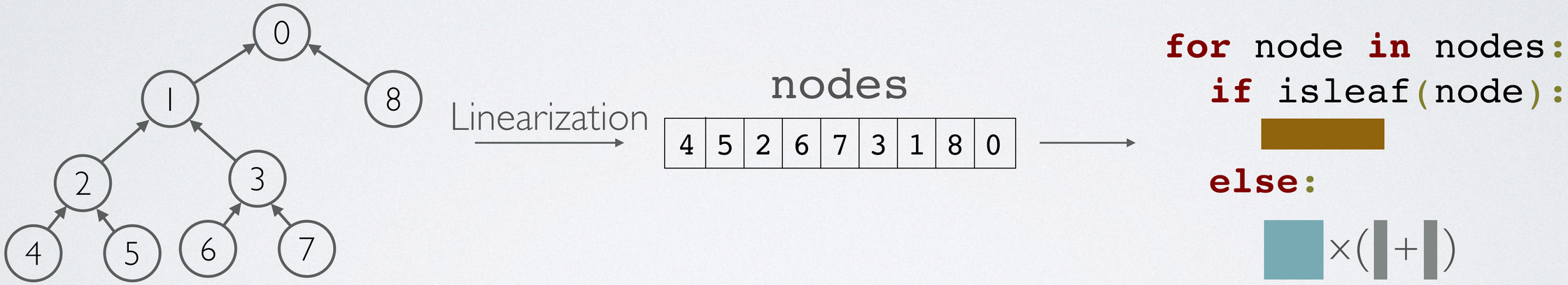


Data Structure Linearization in Cortex

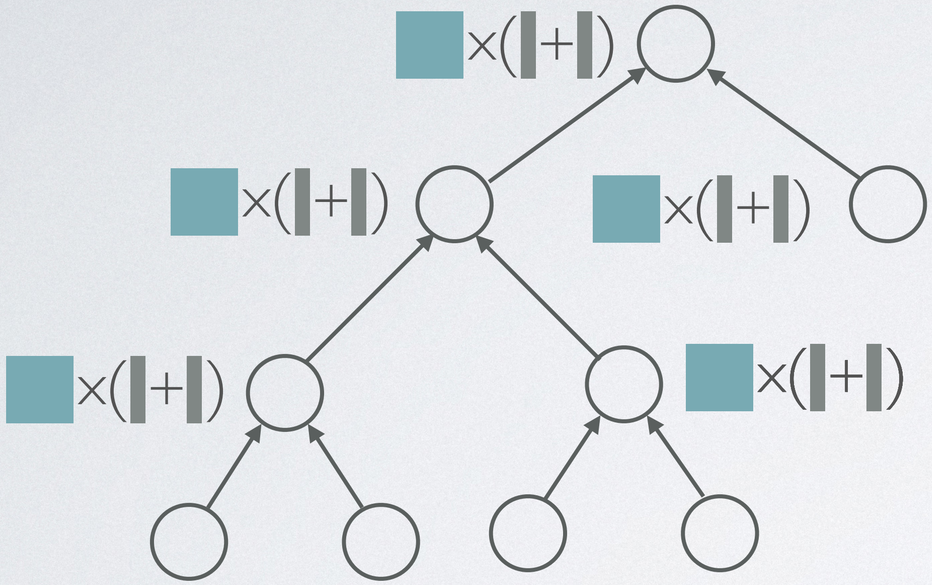
- Control flow depends only on the input structure
- Given the input structure, node processing order is fixed

Data Structure Linearization in Cortex

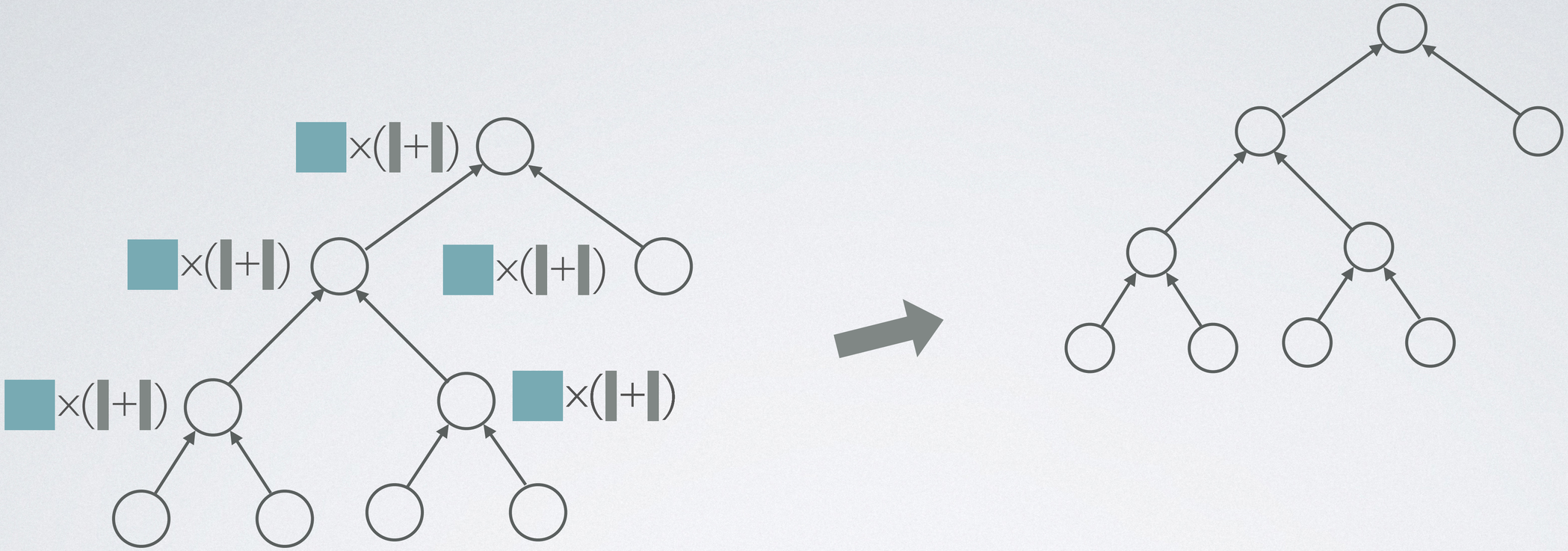
- Control flow depends only on the input structure
- Given the input structure, node processing order is fixed



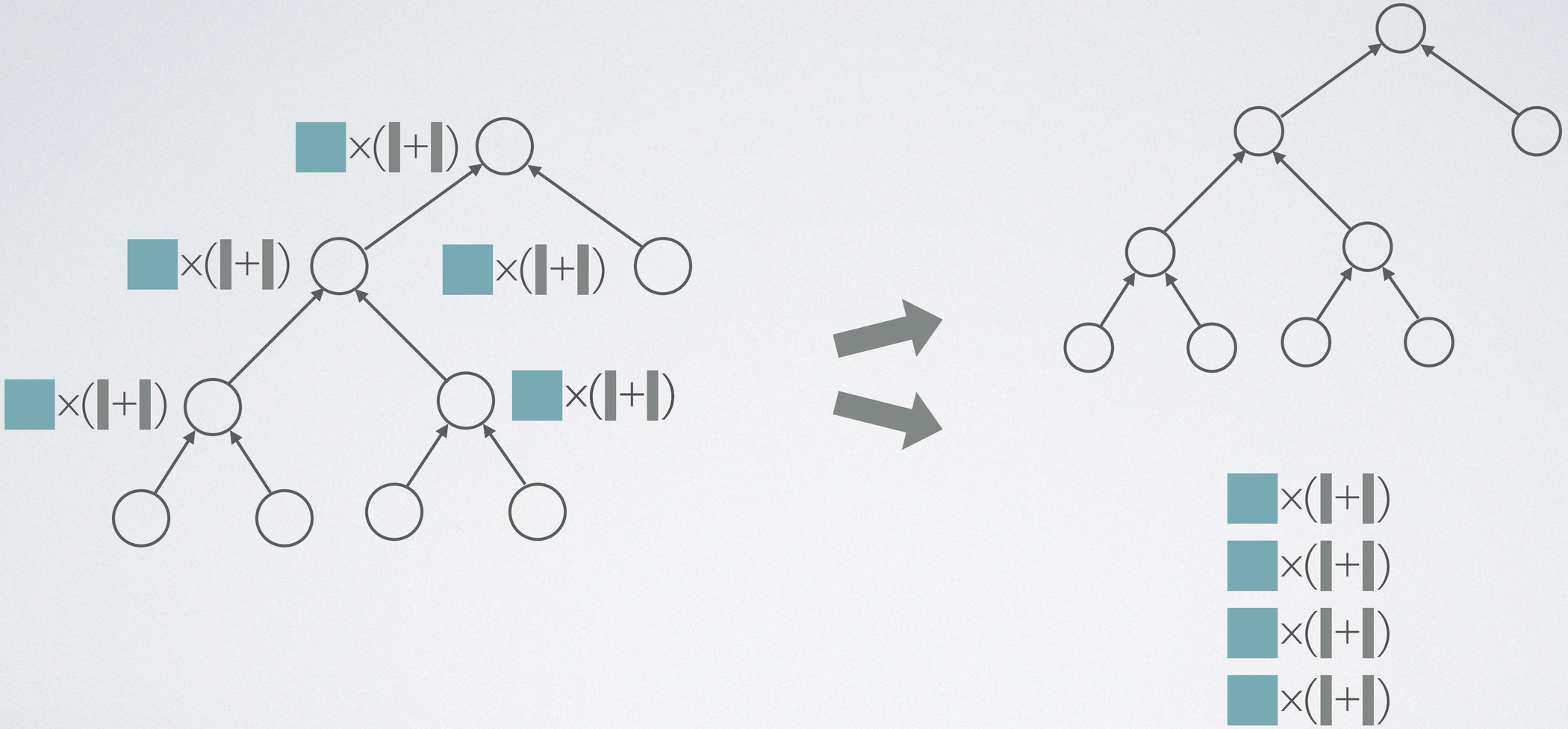
Separation of Control Flow and Tensor Computations



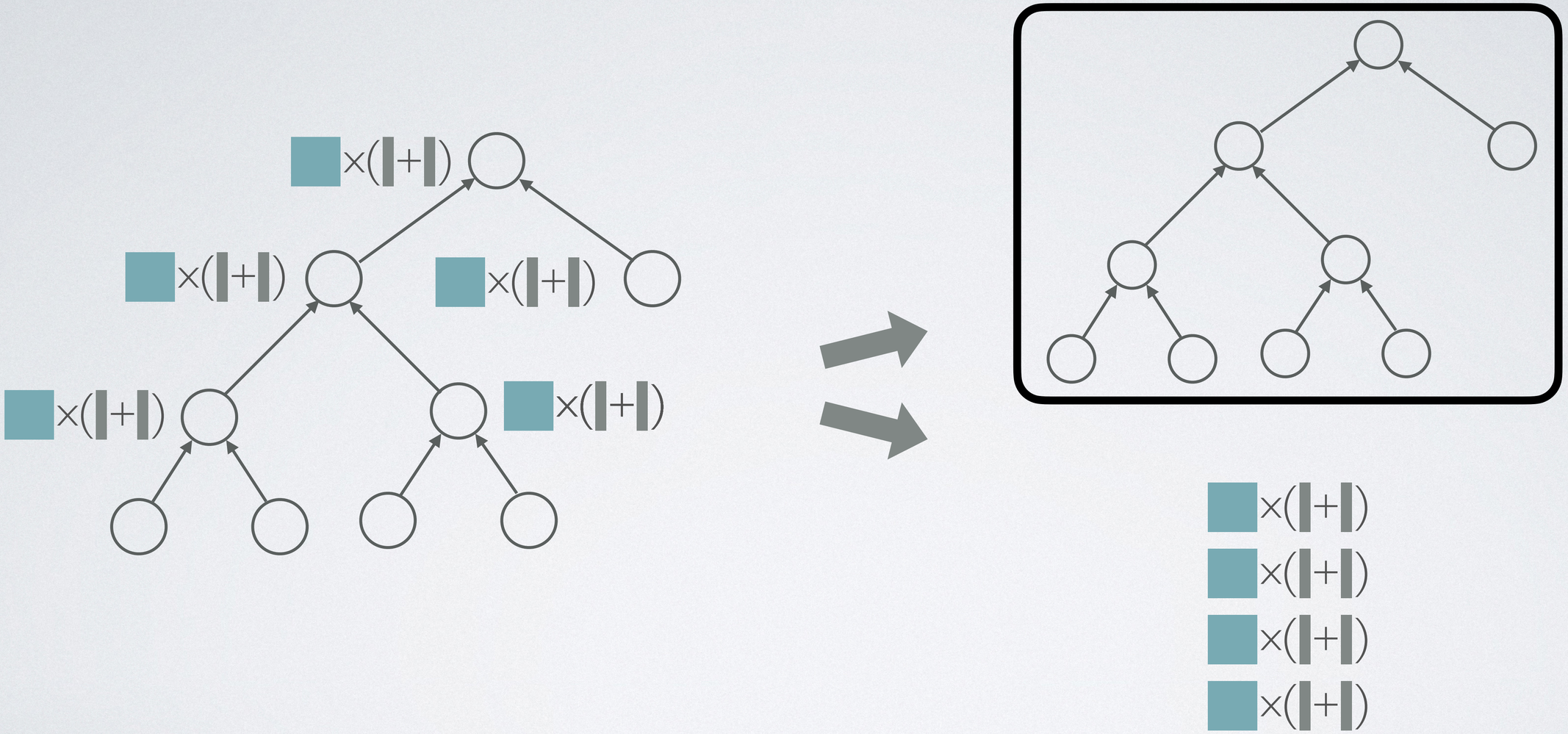
Separation of Control Flow and Tensor Computations



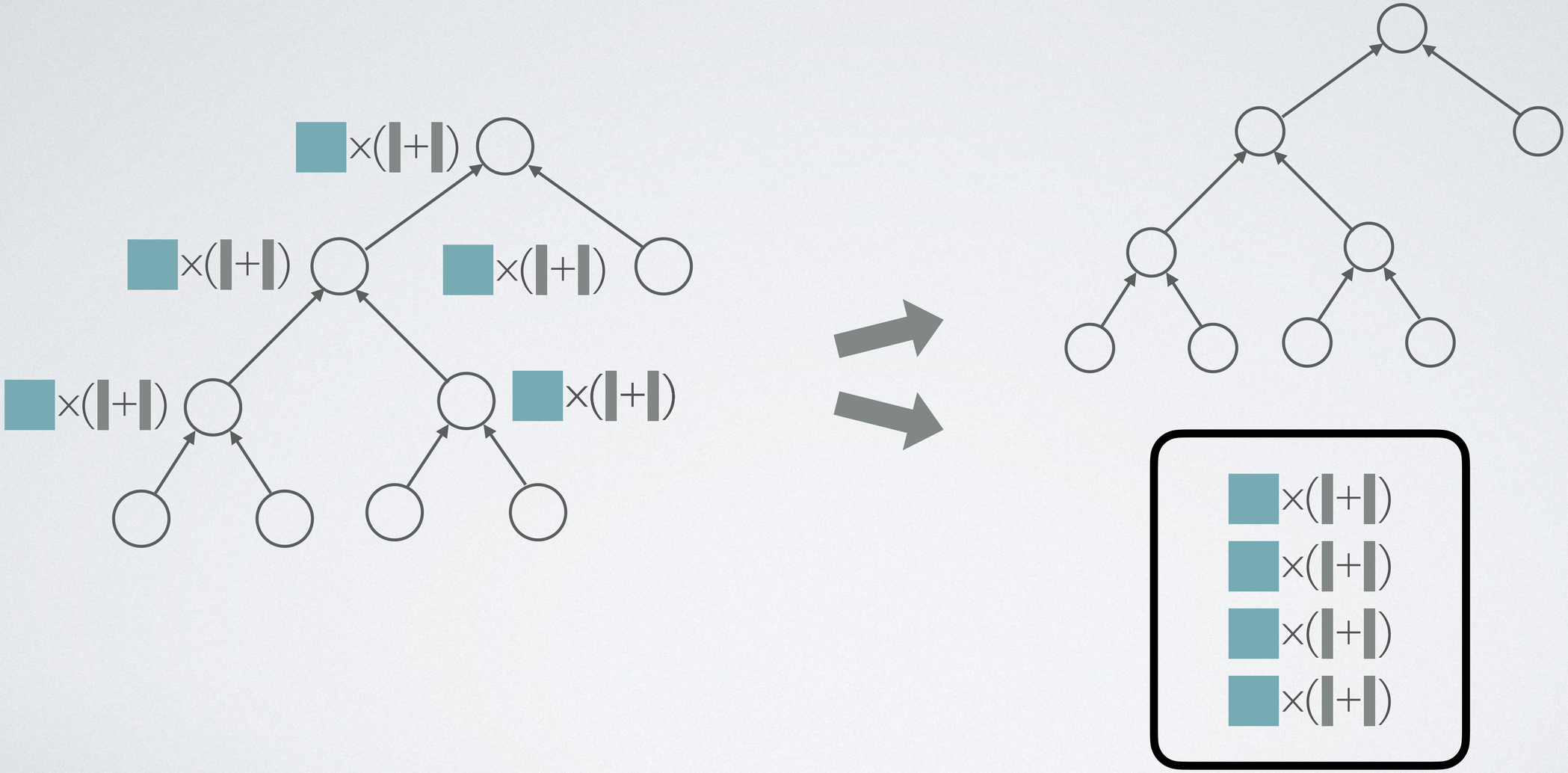
Separation of Control Flow and Tensor Computations



Separation of Control Flow and Tensor Computations



Separation of Control Flow and Tensor Computations



Desired Execution With Aggressive Kernel Fusion

Desired Execution With Aggressive Kernel Fusion

```
nodes = []
def treeFC(n):
  if not isleaf(n):
    treeFC(n.left)
    treeFC(n.right)
    nodes.append(n)
```

CPU execution

GPU execution

Desired Execution With Aggressive Kernel Fusion

```
nodes = []
def treeFC(n):
  if not isleaf(n):
    treeFC(n.left)
    treeFC(n.right)
    nodes.append(n)
```

```
for n in nodes:
  if isleaf(n):
    return Emb[words[n]]
  else:
    lh = treeFC(n.left)
    rh = treeFC(n.right)
    add =  lh + rh
    return  W * add
```

CPU execution

GPU execution

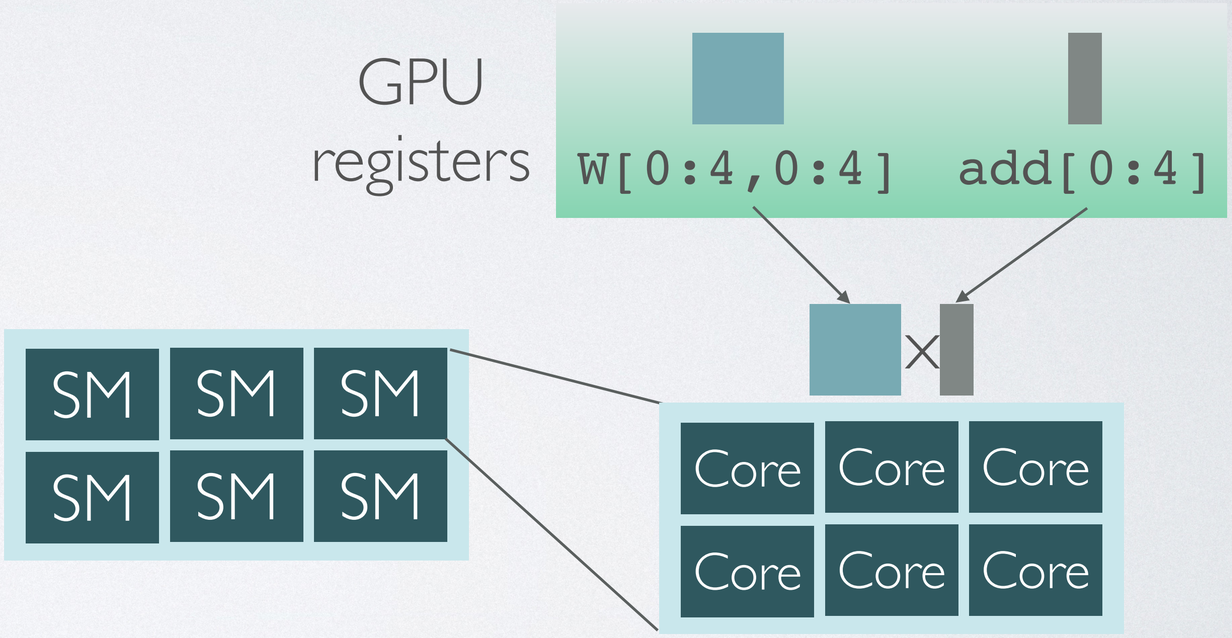
Desired Execution With Aggressive Kernel Fusion

```
nodes = []
def treeFC(n):
  if not isleaf(n):
    treeFC(n.left)
    treeFC(n.right)
    nodes.append(n)
```

```
for n in nodes:
  if isleaf(n):
    return Emb[words[n]]
  else:
    lh = treeFC(n.left)
    rh = treeFC(n.right)
    add =  lh + rh
    return  W * add
```

CPU execution

GPU execution

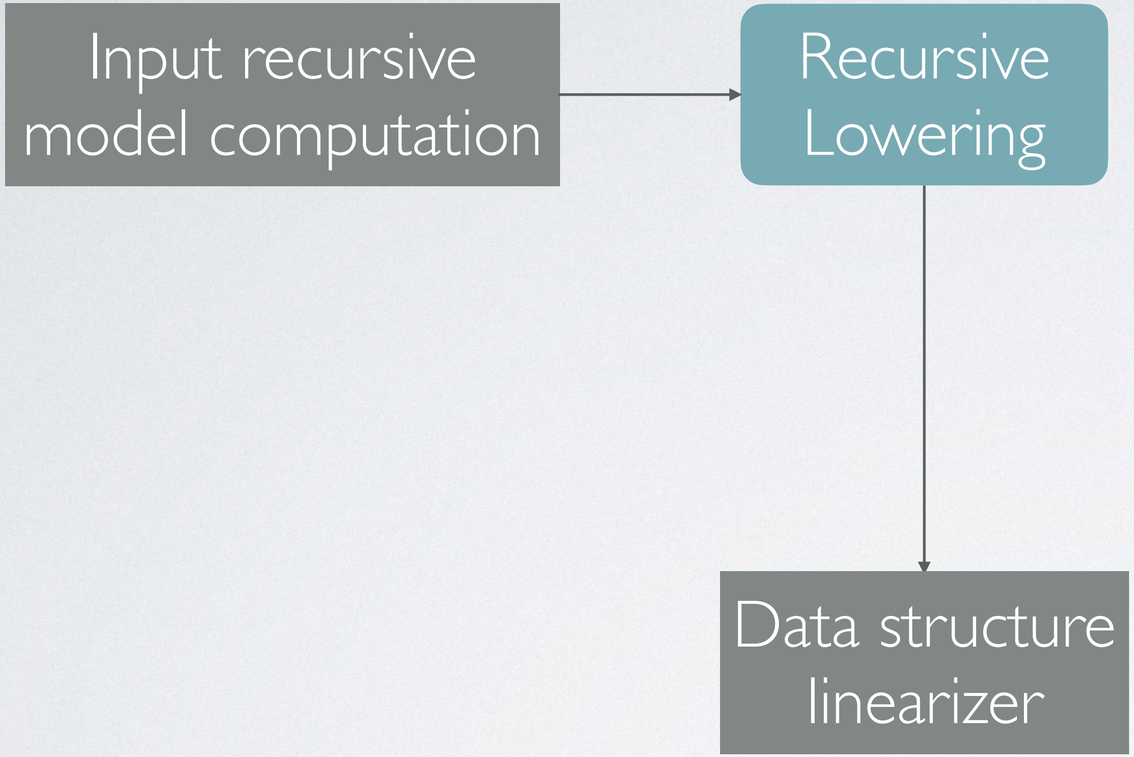


Overview of Cortex's Pipeline

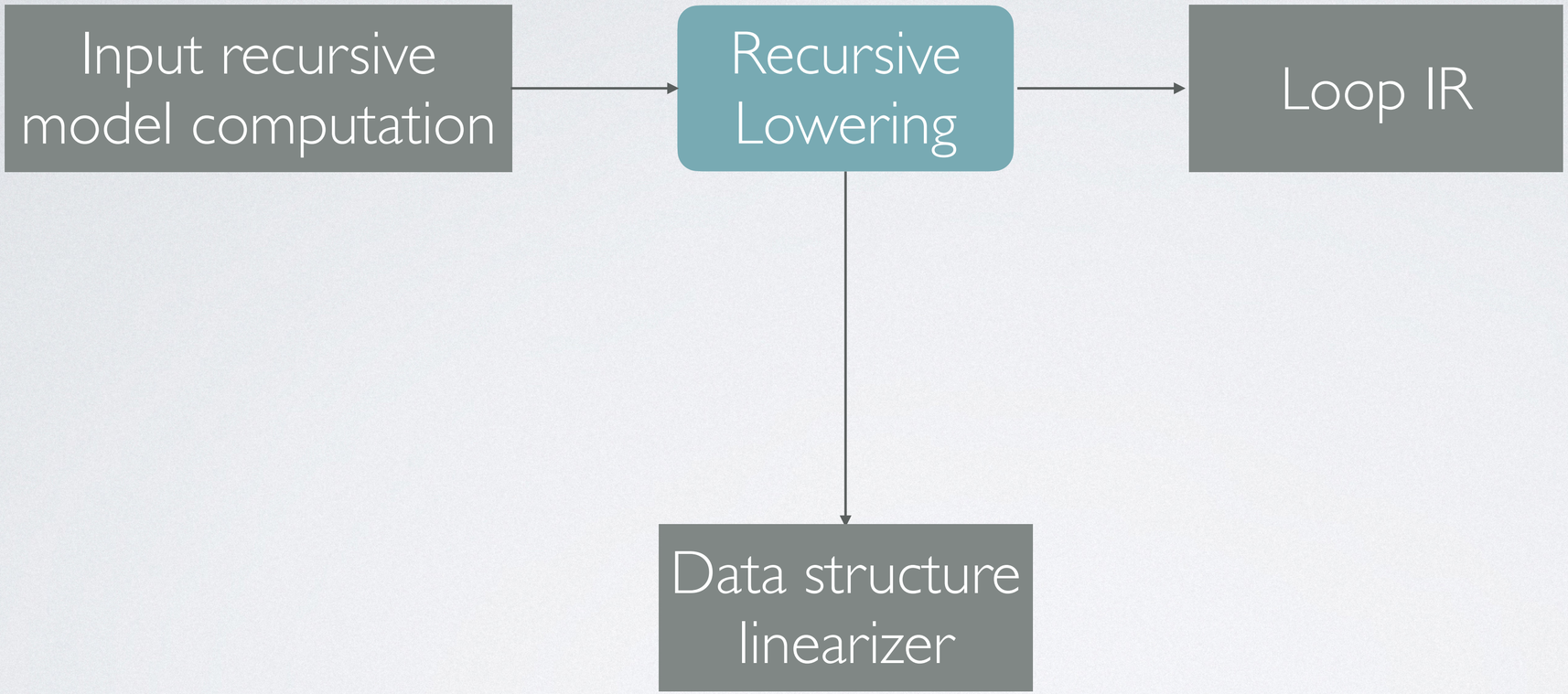
Overview of Cortex's Pipeline

Input recursive model computation

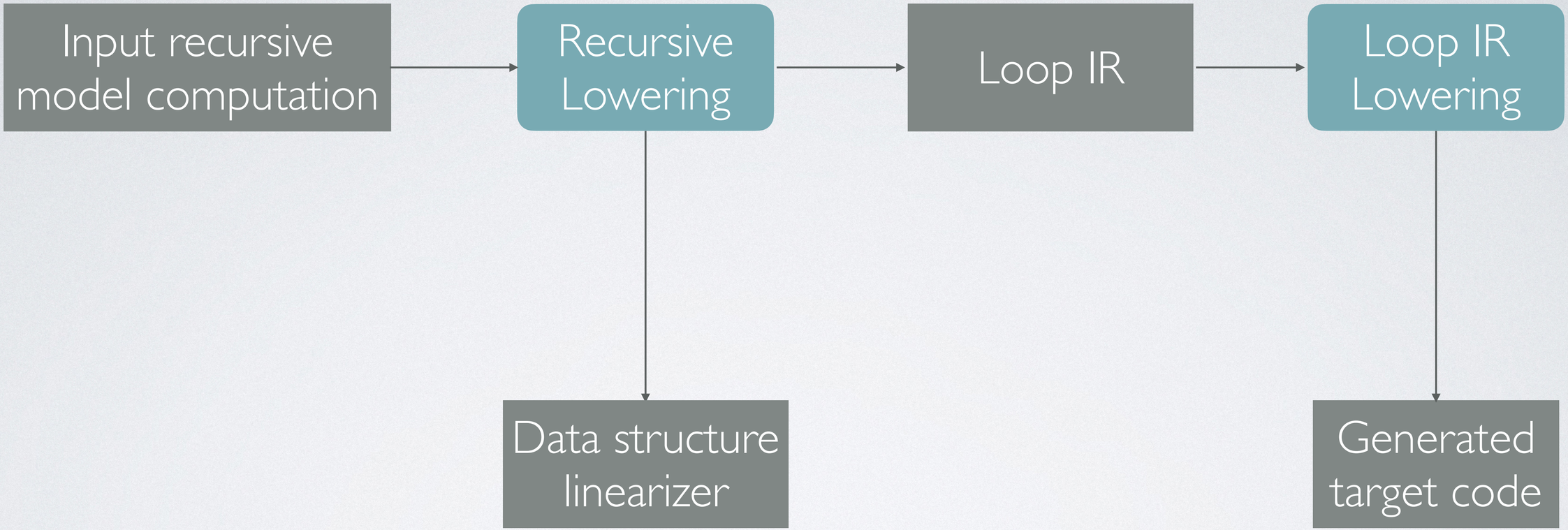
Overview of Cortex's Pipeline



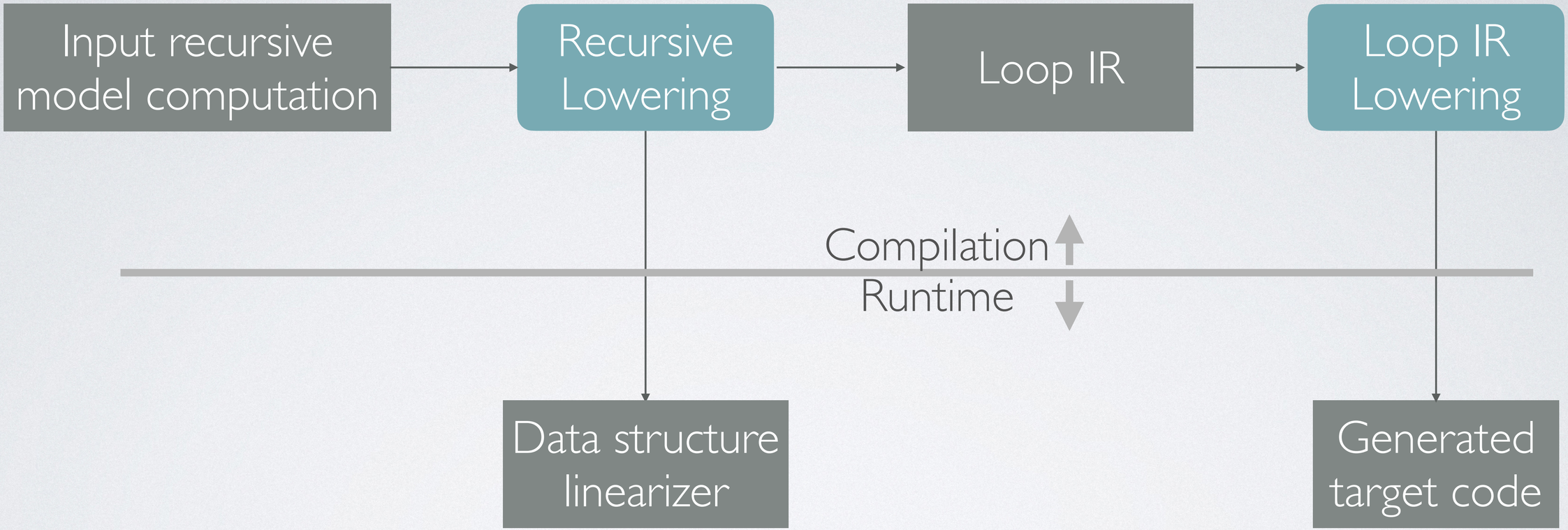
Overview of Cortex's Pipeline



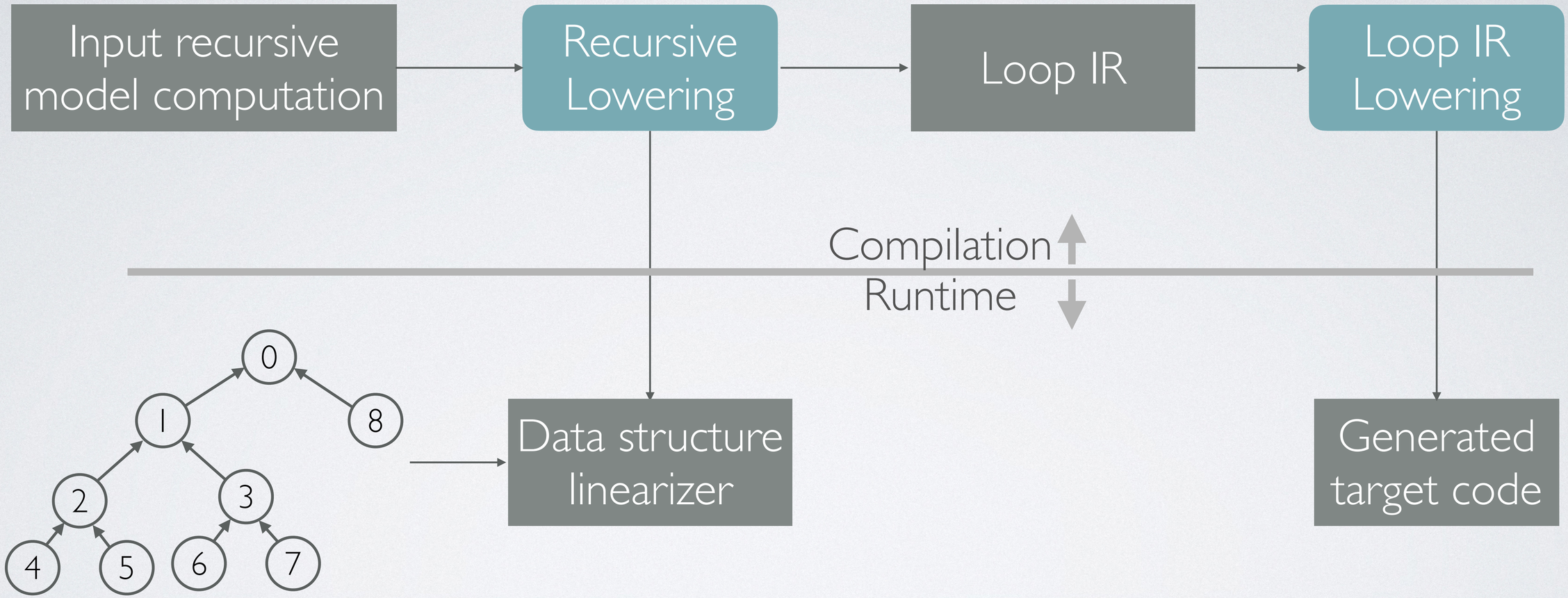
Overview of Cortex's Pipeline



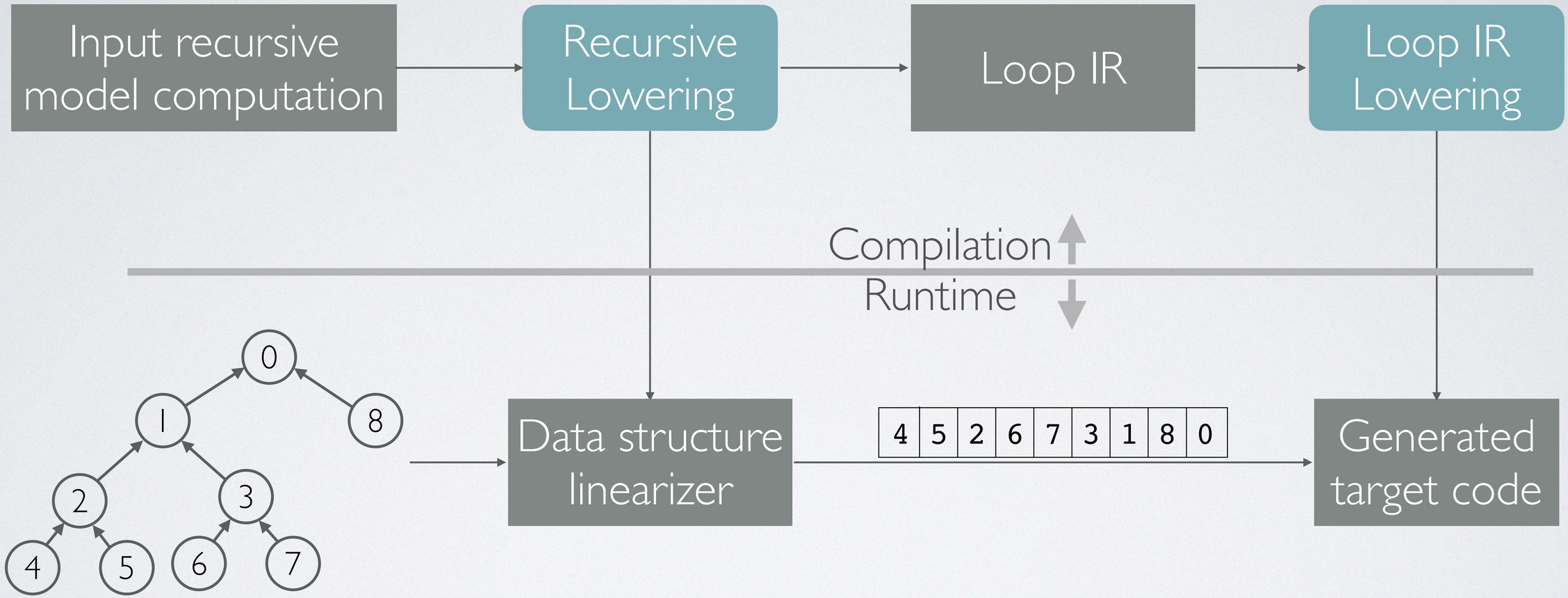
Overview of Cortex's Pipeline



Overview of Cortex's Pipeline



Overview of Cortex's Pipeline



Outline

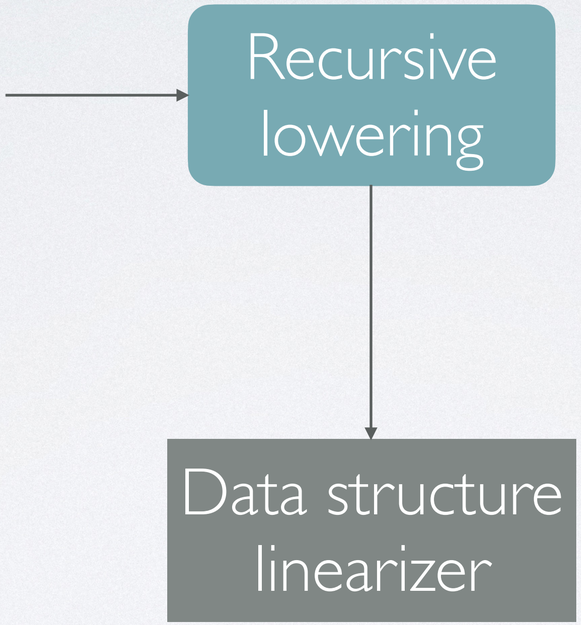
- Motivation: Inefficiencies in Execution of Recursive Models
- Cortex: Our Compiler Based Solution
  - **Recursive Lowering**
  - Loop IR Lowering
- Evaluation
- Conclusion

Recursion Is Lowered to Efficient Loops

```
def treeFC(n):
  if isleaf(n):
    return Emb[words[n]]
  else:
    lh = treeFC(n.left)
    rh = treeFC(n.right)
    return W * (lh + rh)
```

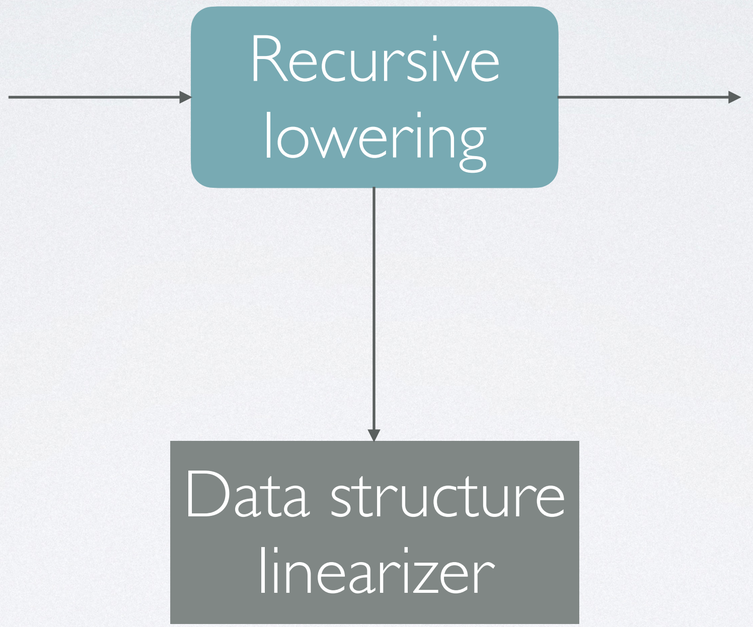
Recursion Is Lowered to Efficient Loops

```
def treeFC(n):
  if isleaf(n):
    return Emb[words[n]]
  else:
    lh = treeFC(n.left)
    rh = treeFC(n.right)
    return W * (lh + rh)
```



Recursion Is Lowered to Efficient Loops

```
def treeFC(n):
  if isleaf(n):
    return Emb[words[n]]
  else:
    lh = treeFC(n.left)
    rh = treeFC(n.right)
    return W * (lh + rh)
```




```
for n in nodes_order:
  if isleaf(n):
    res[n] = Emb[words[n]]
  else:
    lh = treeFC(n.left)
    rh = treeFC(n.right)
    res[n] = W * (lh + rh)
```

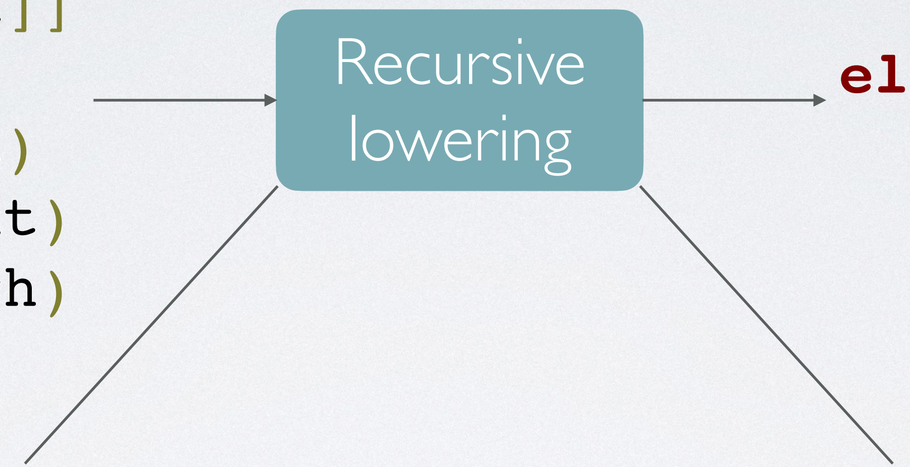
Recursion Is Lowered to Efficient Loops

```
def treeFC(n):
  if isleaf(n):
    return Emb[words[n]]
  else:
    lh = treeFC(n.left)
    rh = treeFC(n.right)
    return W * (lh + rh)
```


```
for n in nodes_order:
  if isleaf(n):
    res[n] = Emb[words[n]]
  else:
    lh = treeFC(n.left)
    rh = treeFC(n.right)
    res[n] = W * (lh + rh)
```

Recursion Is Lowered to Efficient Loops

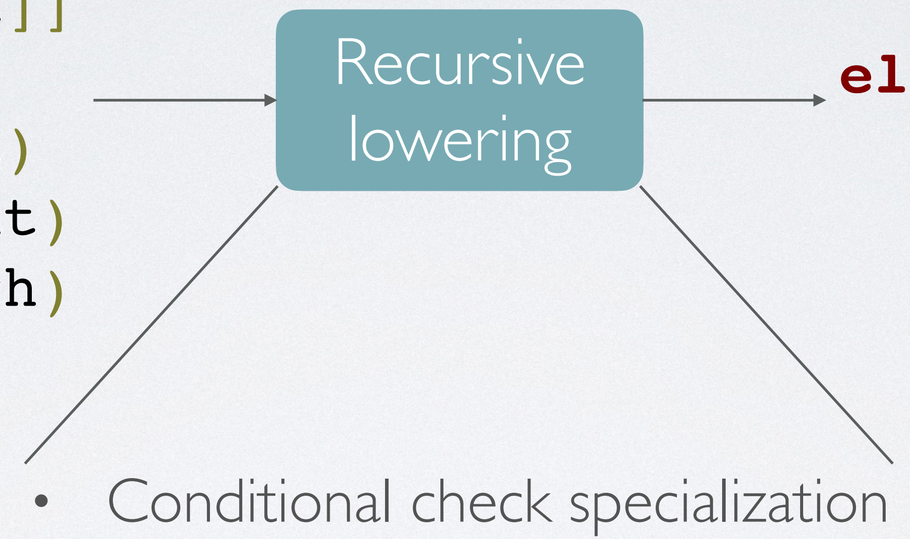
```
def treeFC(n):
  if isleaf(n):
    return Emb[words[n]]
  else:
    lh = treeFC(n.left)
    rh = treeFC(n.right)
    return W * (lh + rh)
```




```
for n in nodes_order:
  if isleaf(n):
    res[n] = Emb[words[n]]
  else:
    lh = treeFC(n.left)
    rh = treeFC(n.right)
    res[n] = W * (lh + rh)
```

Recursion Is Lowered to Efficient Loops

```
def treeFC(n):
  if isleaf(n):
    return Emb[words[n]]
  else:
    lh = treeFC(n.left)
    rh = treeFC(n.right)
    return W * (lh + rh)
```

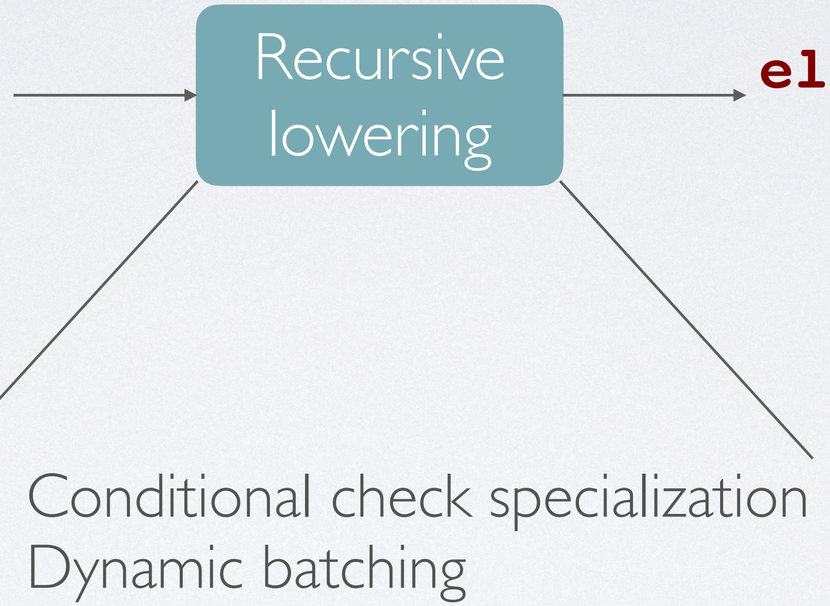



```
for n in nodes_order:
  if isleaf(n):
    res[n] = Emb[words[n]]
  else:
    lh = treeFC(n.left)
    rh = treeFC(n.right)
    res[n] = W * (lh + rh)
```

- Conditional check specialization

Recursion Is Lowered to Efficient Loops

```
def treeFC(n):
  if isleaf(n):
    return Emb[words[n]]
  else:
    lh = treeFC(n.left)
    rh = treeFC(n.right)
    return W * (lh + rh)
```

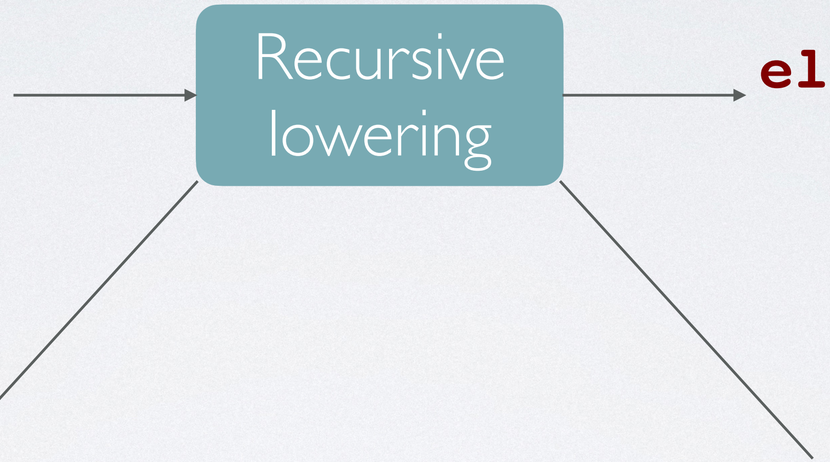



```
for n in nodes_order:
  if isleaf(n):
    res[n] = Emb[words[n]]
  else:
    lh = treeFC(n.left)
    rh = treeFC(n.right)
    res[n] = W * (lh + rh)
```

- Conditional check specialization
- Dynamic batching

Recursion Is Lowered to Efficient Loops

```
def treeFC(n):
  if isleaf(n):
    return Emb[words[n]]
  else:
    lh = treeFC(n.left)
    rh = treeFC(n.right)
    return W * (lh + rh)
```

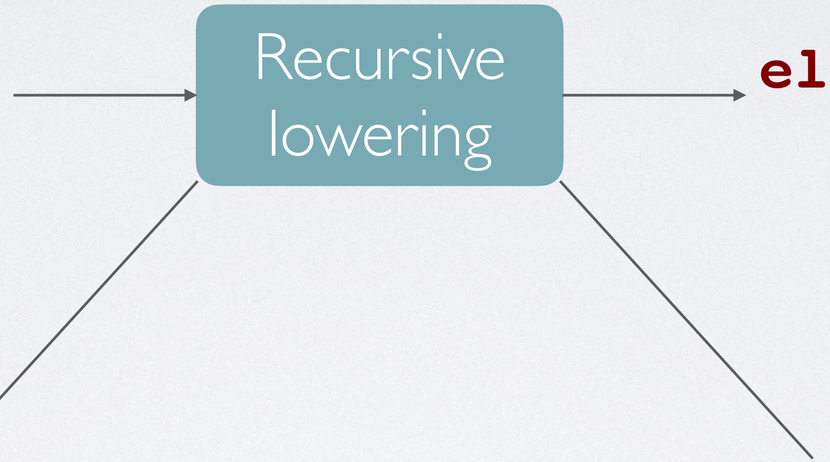



```
for n in nodes_order:
  if isleaf(n):
    res[n] = Emb[words[n]]
  else:
    lh = treeFC(n.left)
    rh = treeFC(n.right)
    res[n] = W * (lh + rh)
```

- Conditional check specialization
- Dynamic batching
- Recursive unrolling

Recursion Is Lowered to Efficient Loops

```
def treeFC(n):
  if isleaf(n):
    return Emb[words[n]]
  else:
    lh = treeFC(n.left)
    rh = treeFC(n.right)
    return W * (lh + rh)
```

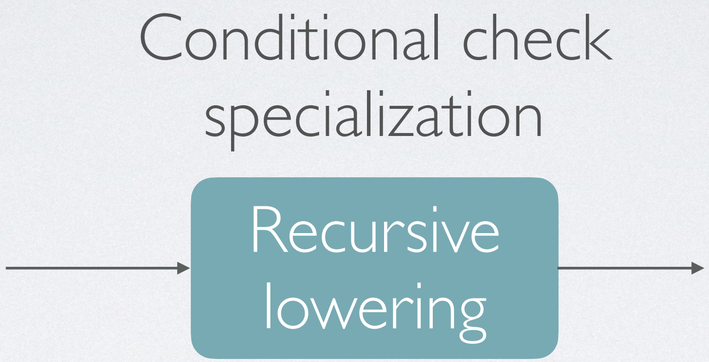



```
for n in nodes_order:
  if isleaf(n):
    res[n] = Emb[words[n]]
  else:
    lh = treeFC(n.left)
    rh = treeFC(n.right)
    res[n] = W * (lh + rh)
```

- Conditional check specialization
- Dynamic batching
- Recursive unrolling
- Recursive refactoring

Cortex Provides Recursive Scheduling Primitives

```
def treeFC(n):
  if isleaf(n):
    return Emb[words[n]]
  else:
    lh = treeFC(n.left)
    rh = treeFC(n.right)
    return W * (lh + rh)
```




```
for n in leaves:
  res[n] = Emb[words[n]]

for n in nodes:
  lh = treeFC(n.left)
  rh = treeFC(n.right)
  res[n] = W * (lh + rh)
```

Cortex Provides Recursive Scheduling Primitives

```
def treeFC(n):
  if isleaf(n):
    return Emb[words[n]]
  else:
    lh = treeFC(n.left)
    rh = treeFC(n.right)
    return W * (lh + rh)
```

Conditional check specialization

Recursive lowering

```
for n in leaves:
  res[n] = Emb[words[n]]

for n in nodes:
  lh = treeFC(n.left)
  rh = treeFC(n.right)
  res[n] = W * (lh + rh)
```

Cortex Provides Recursive Scheduling Primitives

```
def treeFC(n):
  if isleaf(n):
    return Emb[words[n]]
  else:
    lh = treeFC(n.left)
    rh = treeFC(n.right)
    return W * (lh + rh)
```

Conditional check specialization

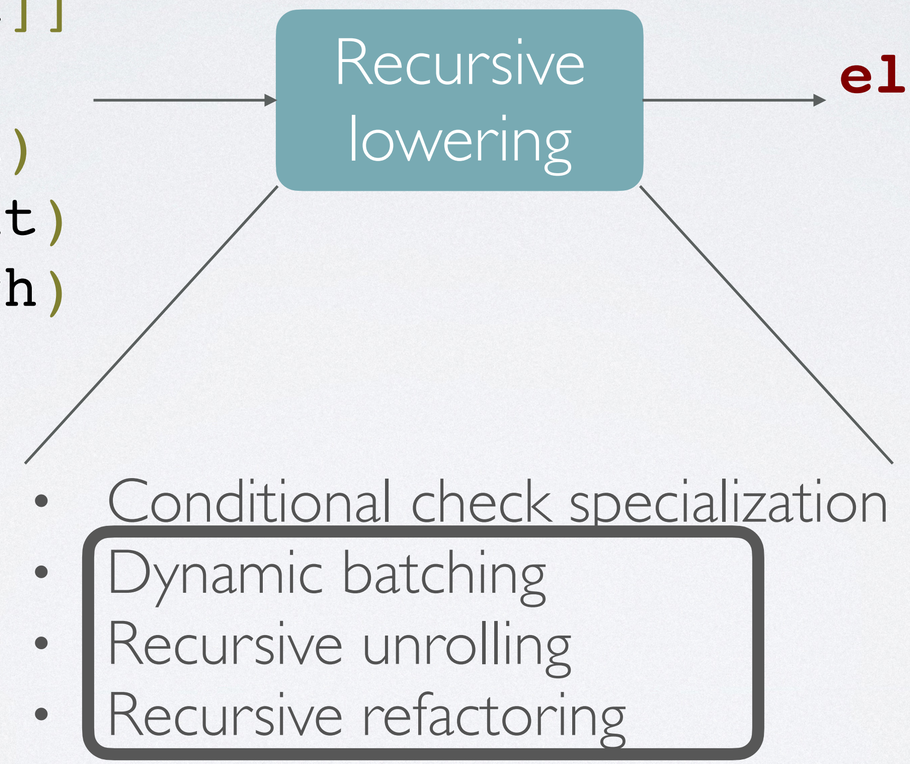
Recursive lowering

```
for n in leaves:
  res[n] = Emb[words[n]]
```

```
for n in nodes:
  lh = treeFC(n.left)
  rh = treeFC(n.right)
  res[n] = W * (lh + rh)
```

Recursion Is Lowered to Efficient Loops

```
def treeFC(n):
  if isleaf(n):
    return Emb[words[n]]
  else:
    lh = treeFC(n.left)
    rh = treeFC(n.right)
    return W * (lh + rh)
```



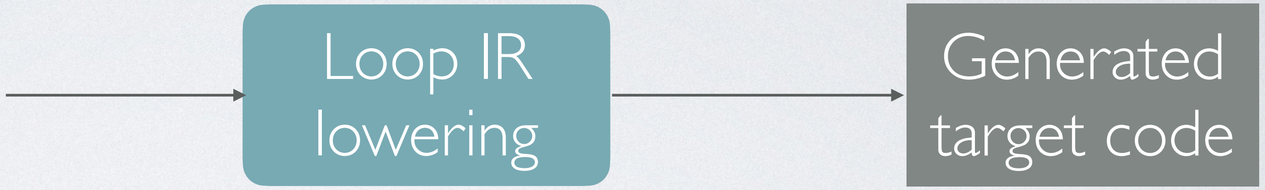

```
for n in nodes_order:
  if isleaf(n):
    res[n] = Emb[words[n]]
  else:
    lh = treeFC(n.left)
    rh = treeFC(n.right)
    res[n] = W * (lh + rh)
```

Outline

- Motivation: Inefficiencies in Execution of Recursive Models
- Cortex: Our Compiler Based Solution
  - Recursive Scheduling
  - **Loop IR Lowering**
- Evaluation
- Conclusion

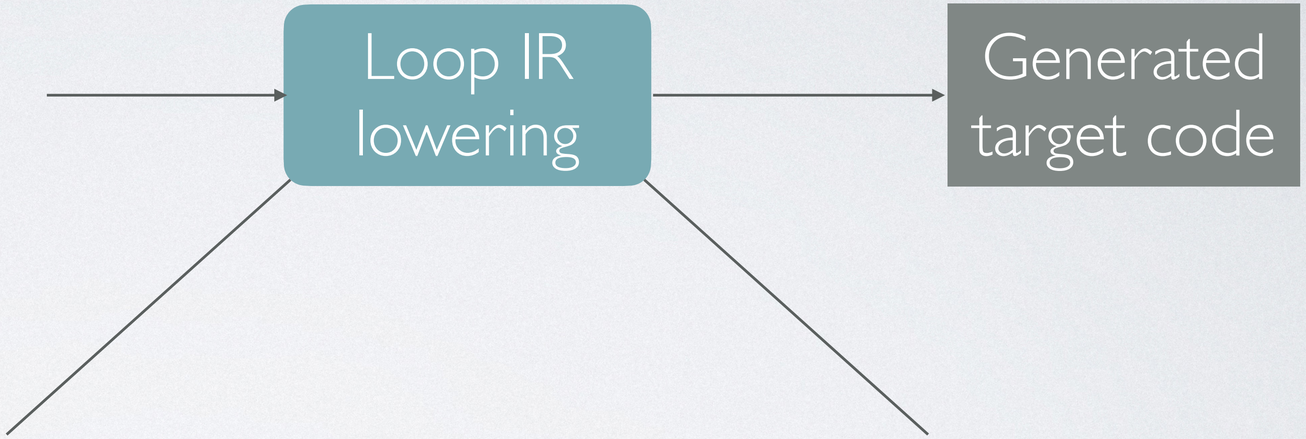
Loop IR Allows Tensor Compiler-Like Scheduling

```
for n in leaves:
  res[n] = Emb[words[n]]

for n in nodes:
  lh = treeFC(n.left)
  rh = treeFC(n.right)
  res[n] = W * (lh + rh)
```



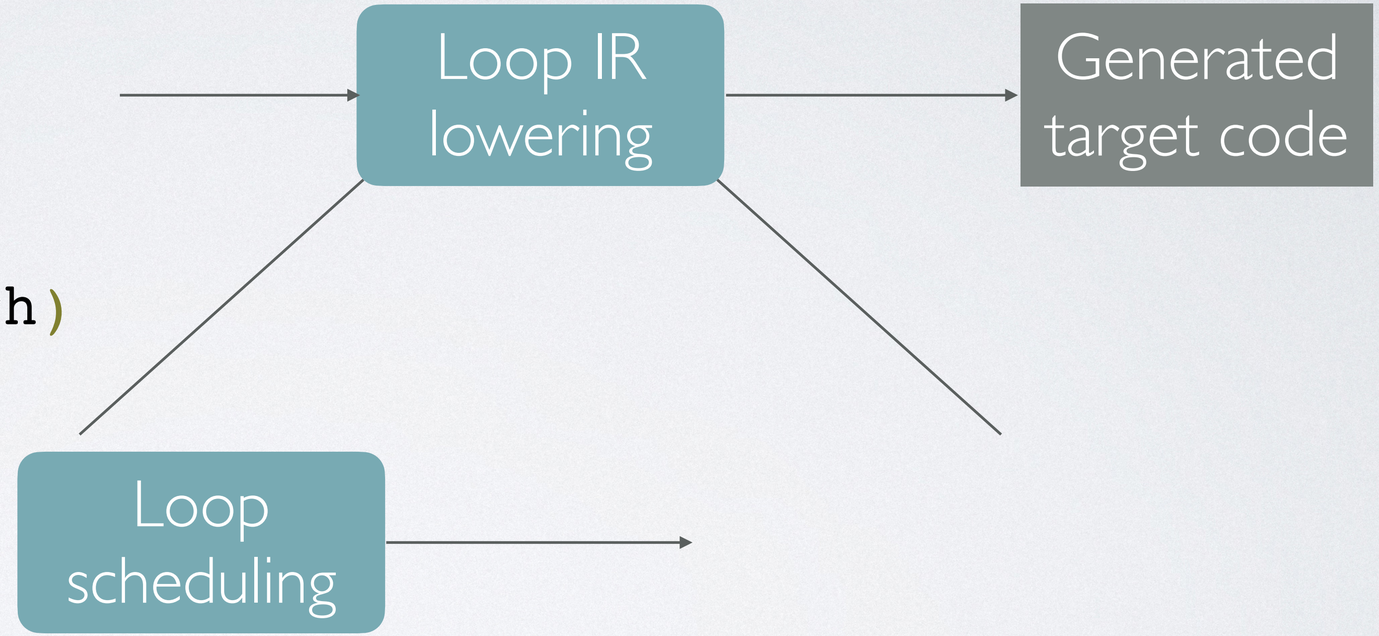
Loop IR Allows Tensor Compiler-Like Scheduling

```
for n in leaves:
  res[n] = Emb[words[n]]

for n in nodes:
  lh = treeFC(n.left)
  rh = treeFC(n.right)
  res[n] = W * (lh + rh)
```



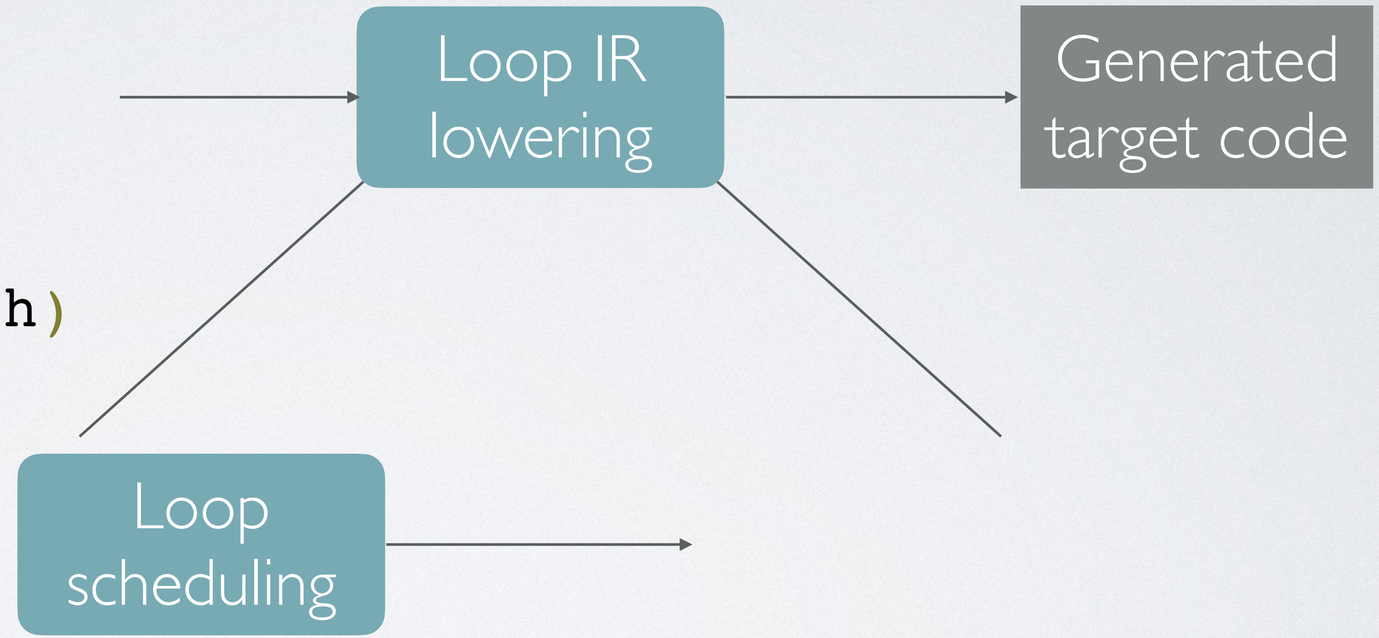
Loop IR Allows Tensor Compiler-Like Scheduling

```
for n in leaves:
  res[n] = Emb[words[n]]

for n in nodes:
  lh = treeFC(n.left)
  rh = treeFC(n.right)
  res[n] = W * (lh + rh)
```



Loop IR Allows Tensor Compiler-Like Scheduling

```
for n in leaves:
  res[n] = Emb[words[n]]

for n in nodes:
  lh = treeFC(n.left)
  rh = treeFC(n.right)
  res[n] = W * (lh + rh)
```

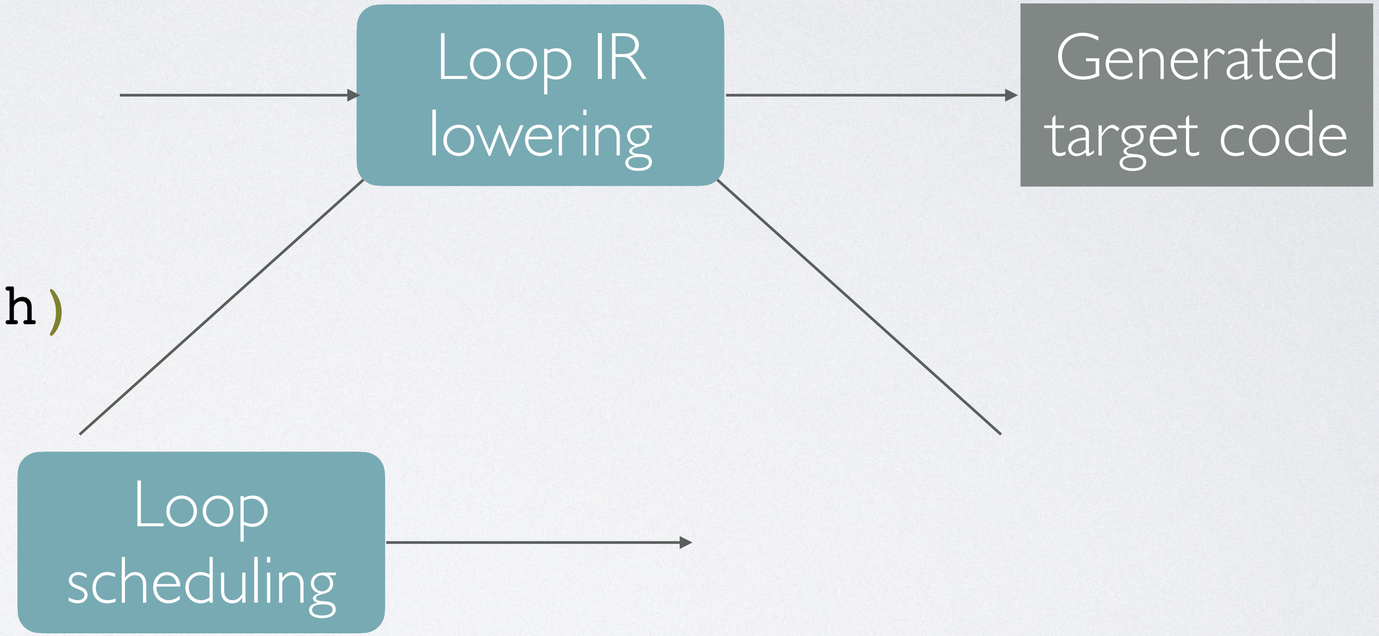


Scheduling primitives

- Loop tiling, unrolling, etc

Loop IR Allows Tensor Compiler-Like Scheduling

```
for n in leaves:
  res[n] = Emb[words[n]]

for n in nodes:
  lh = treeFC(n.left)
  rh = treeFC(n.right)
  res[n] = W * (lh + rh)
```

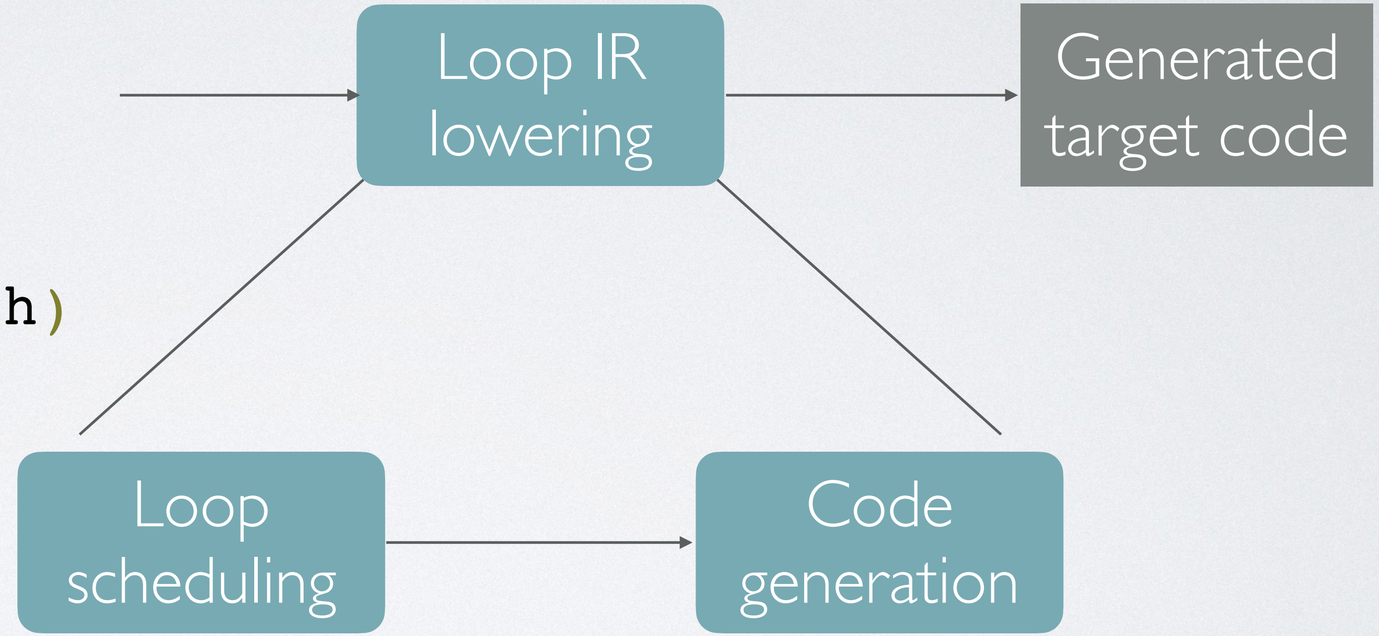


Scheduling primitives

- Loop tiling, unrolling, etc
- Caching in scratchpad

Loop IR Allows Tensor Compiler-Like Scheduling

```
for n in leaves:
  res[n] = Emb[words[n]]

for n in nodes:
  lh = treeFC(n.left)
  rh = treeFC(n.right)
  res[n] = W * (lh + rh)
```



Scheduling primitives

- Loop tiling, unrolling, etc
- Caching in scratchpad

Outline

- Motivation: Inefficiencies in Execution of Recursive Models
- Cortex: Our Compiler Based Solution
  - Recursive Lowering
  - Loop IR Lowering
- **Evaluation**
- Conclusion

Evaluation Setup

- Inference latencies measured for PyTorch, DyNet, Cavs and Cortex

Evaluation Setup

- Inference latencies measured for PyTorch, DyNet, Cavs and Cortex
- Models

Evaluation Setup

- Inference latencies measured for PyTorch, DyNet, Cavs and Cortex
- Models

Models	Dataset	Hidden Size
MV-RNN	Stanford treebank	64
TreeFC	Synthetic perfect binary trees of height 10	256
TreeGRU	Stanford treebank	256
TreeLSTM	Stanford treebank	256
DAG-RNN	Synthetic 10x10 DAGs	256

Evaluation Setup

- Inference latencies measured for PyTorch, DyNet, Cavs and Cortex
- Models

Models	Dataset	Hidden Size
MV-RNN	Stanford treebank	64
TreeFC	Synthetic perfect binary trees of height 10	256
TreeGRU	Stanford treebank	256
TreeLSTM	Stanford treebank	256
DAG-RNN	Synthetic 10x10 DAGs	256

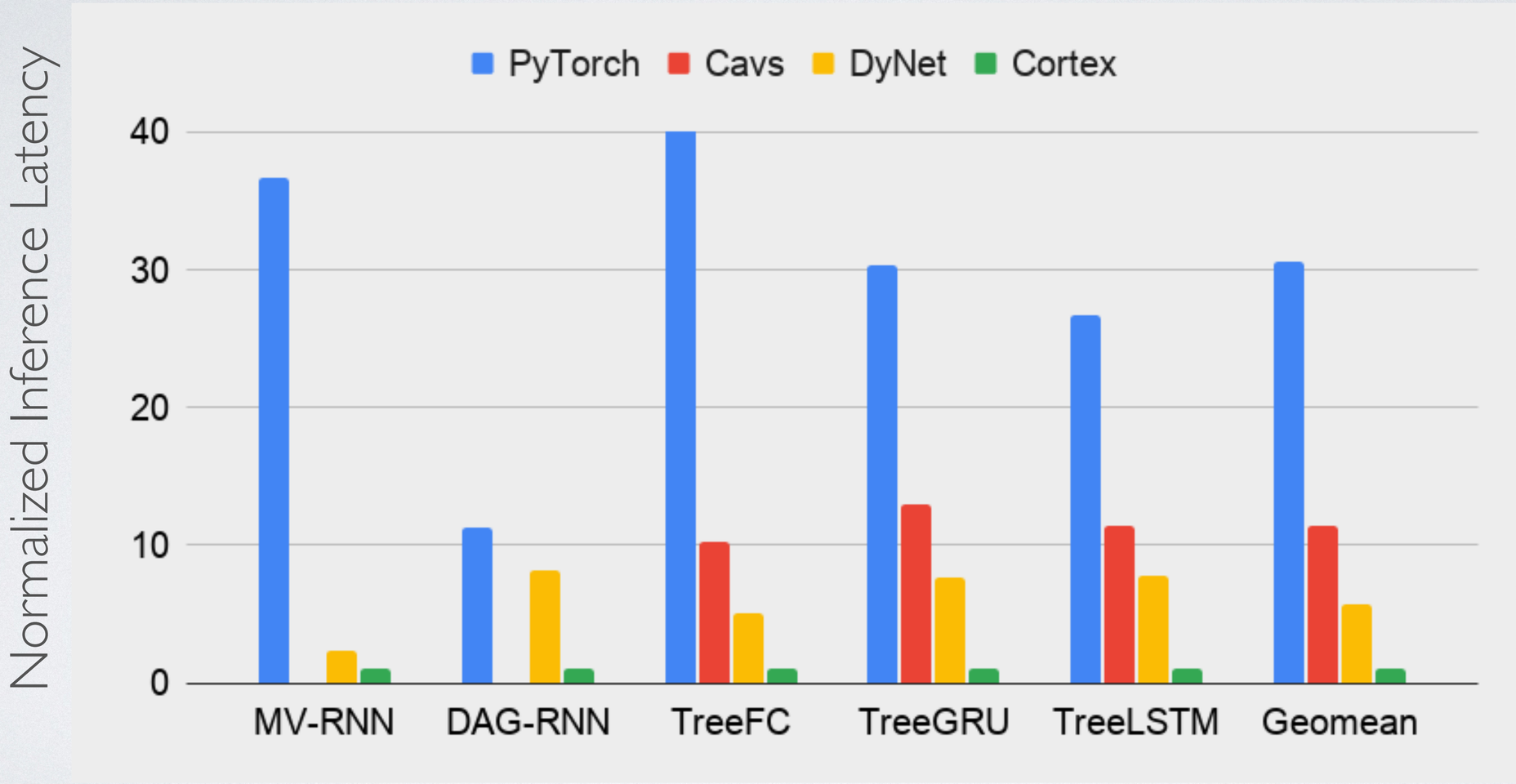
Evaluation Setup

- Inference latencies measured for PyTorch, DyNet, Cavs and Cortex
- Models

Models	Dataset	Hidden Size
MV-RNN	Stanford treebank	64
TreeFC	Synthetic perfect binary trees of height 10	256
TreeGRU	Stanford treebank	256
TreeLSTM	Stanford treebank	256
DAG-RNN	Synthetic 10x10 DAGs	256

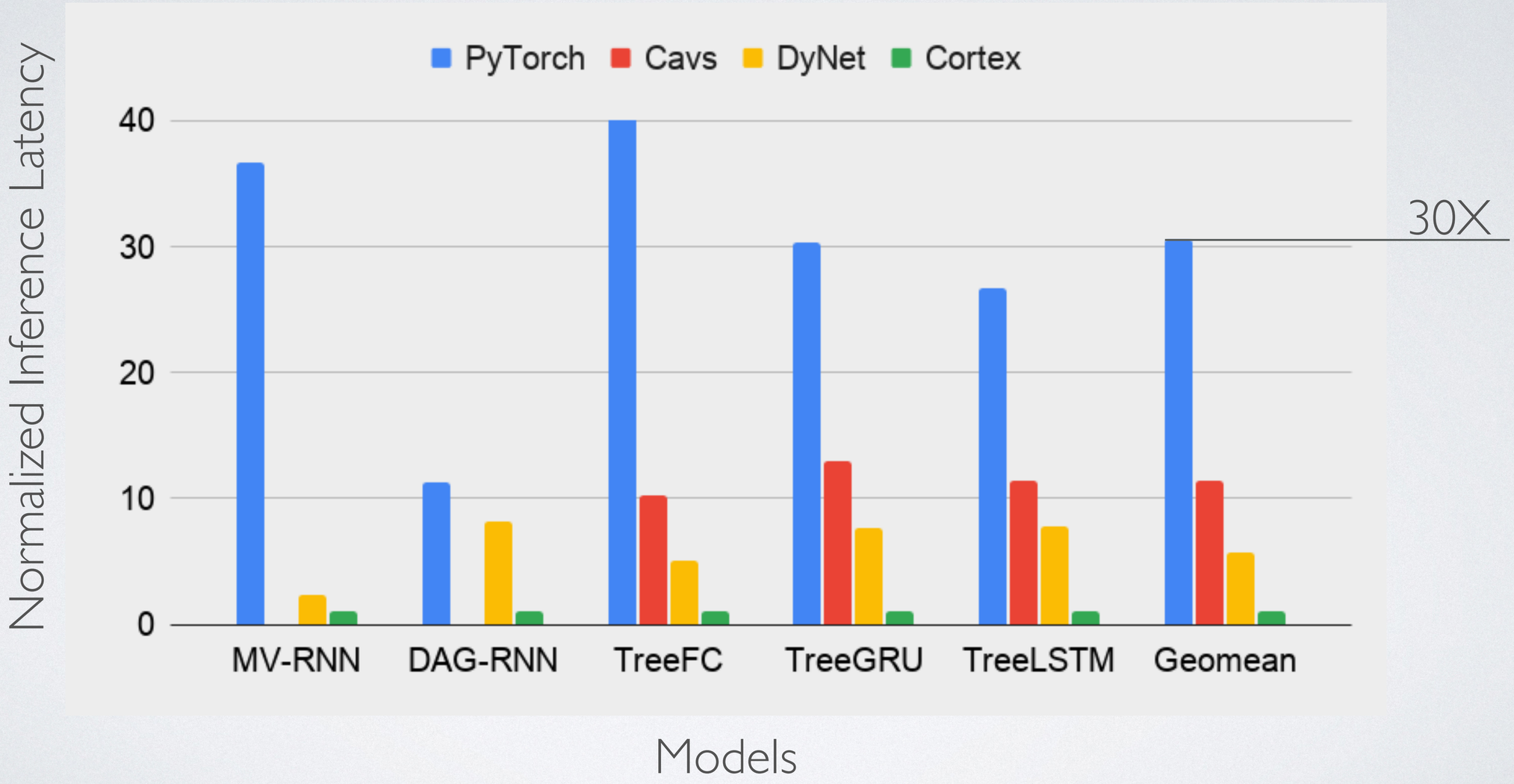
Inference Latencies on Nvidia V100 GPU

Lower is better



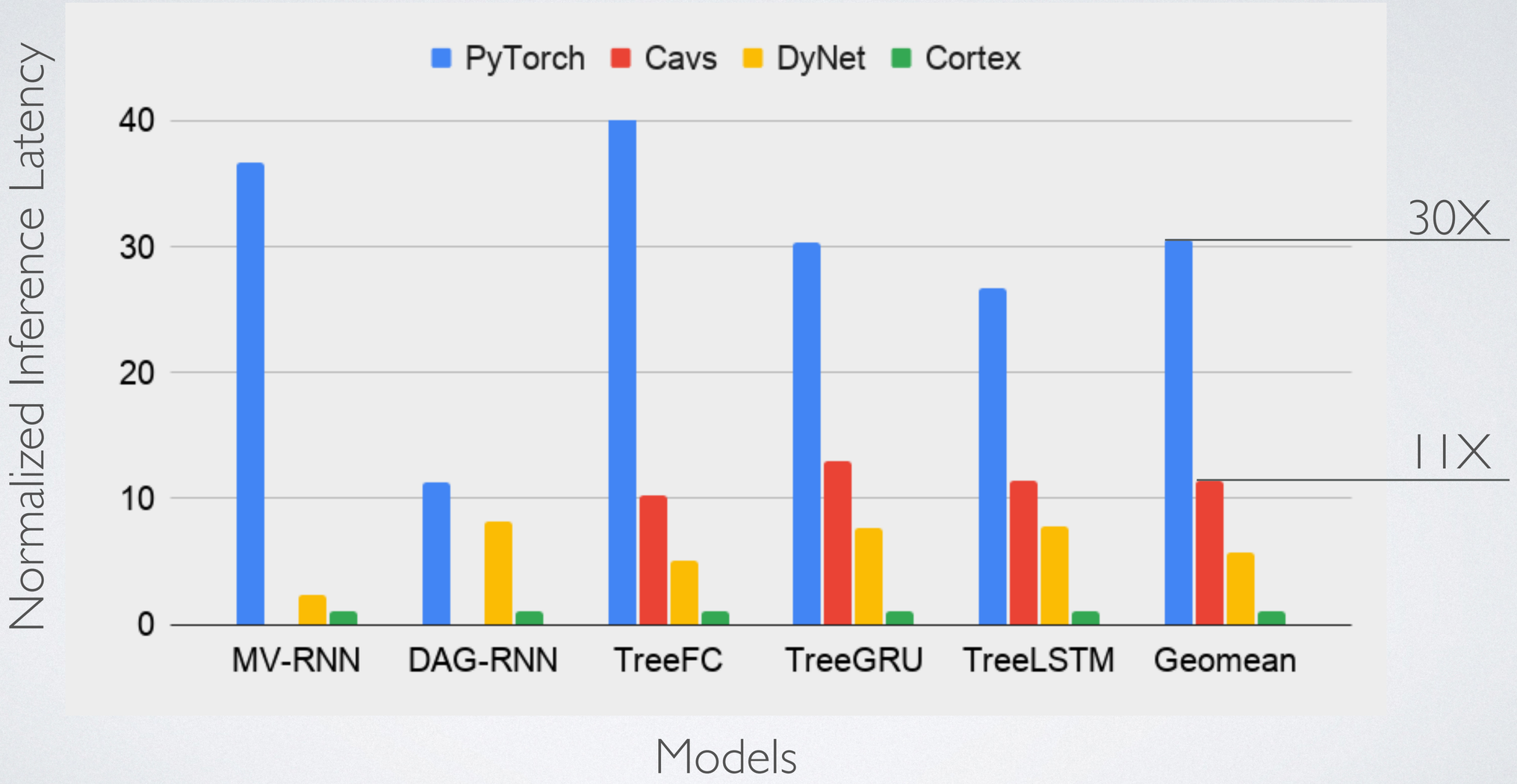
Inference Latencies on Nvidia V100 GPU

Lower is better



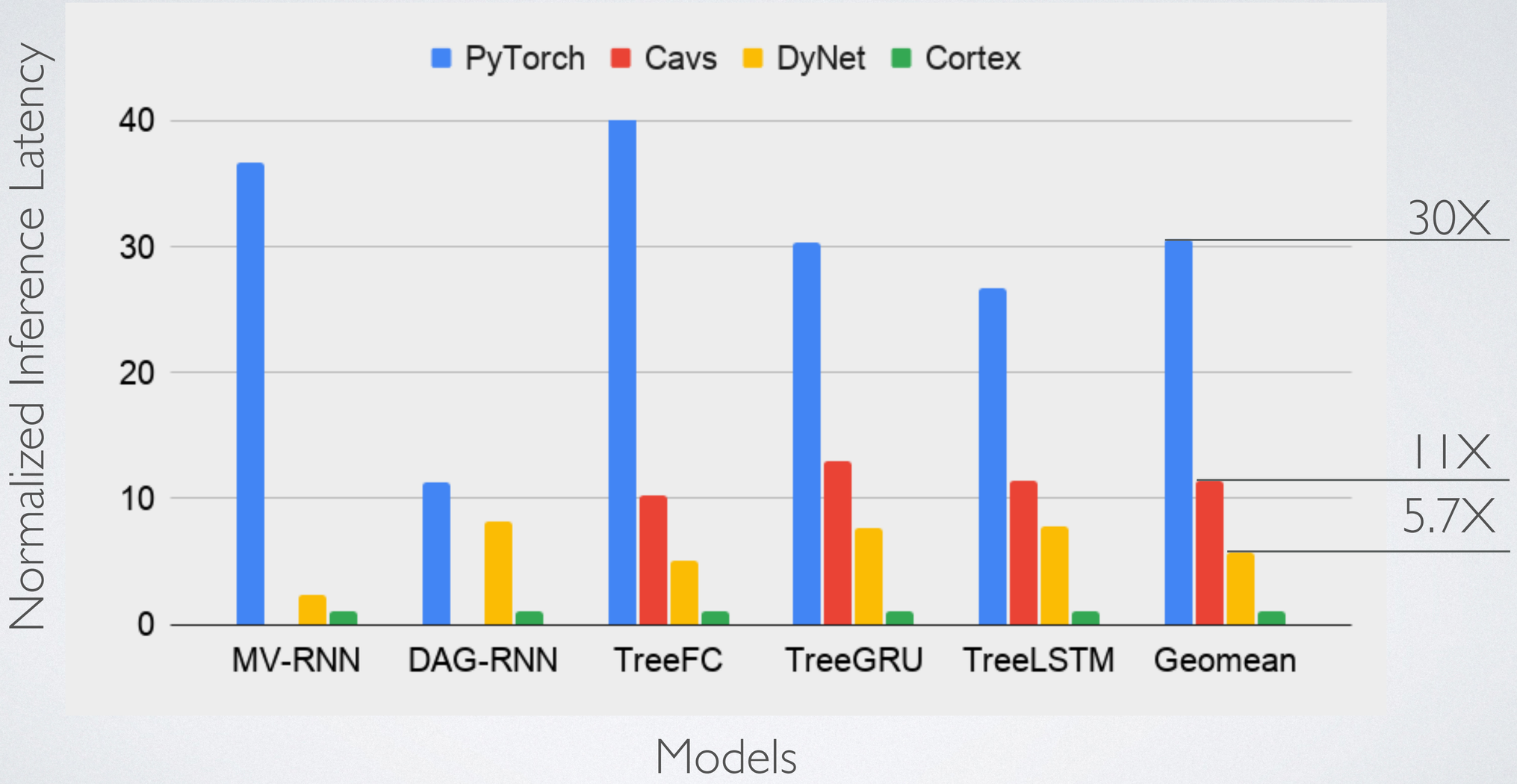
Inference Latencies on Nvidia V100 GPU

Lower is better



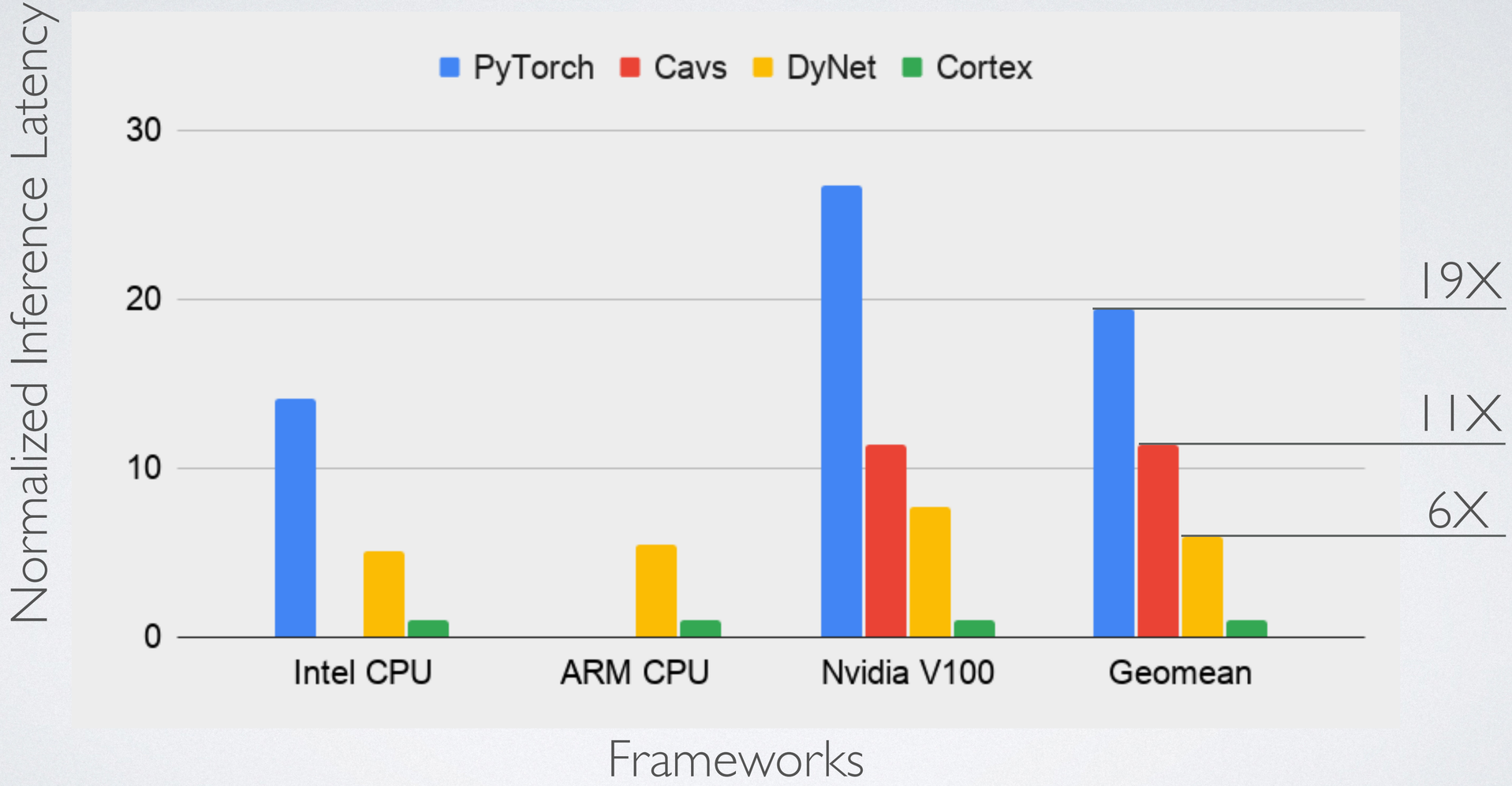
Inference Latencies on Nvidia V100 GPU

Lower is better



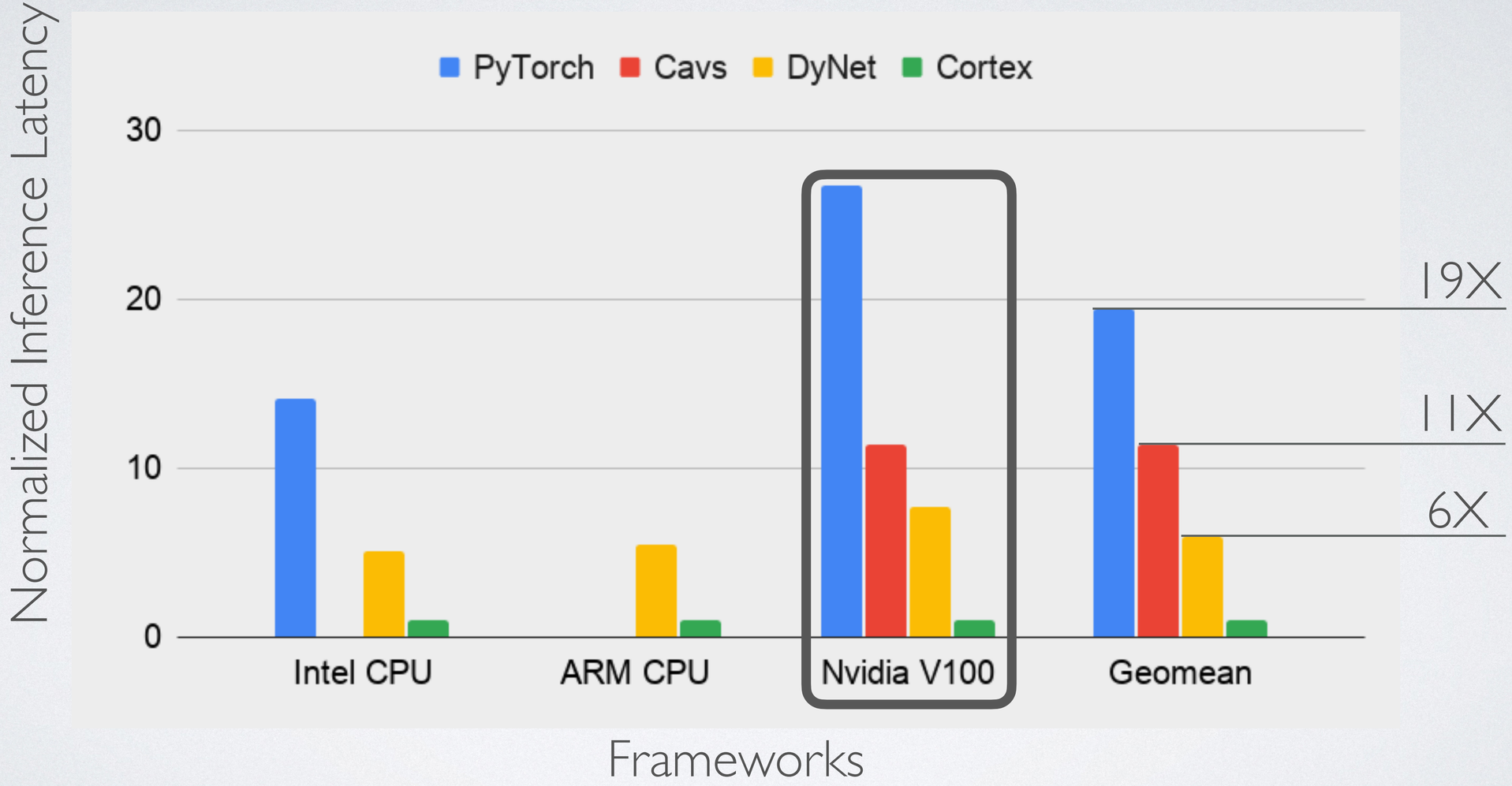
Inference Latencies for TreeLSTM Model

Lower is better



Inference Latencies for TreeLSTM Model

Lower is better



Cortex Has Low Scheduling Overheads

Cortex Has Low Scheduling Overheads

Framework	#Kernel calls
DyNet	389
Cavs	122
Cortex	1

Cortex Has Low Scheduling Overheads

Framework	#Kernel calls	Scheduling time (ms)
DyNet	389	3.03
Cavs	122	0.4
Cortex	1	0.01

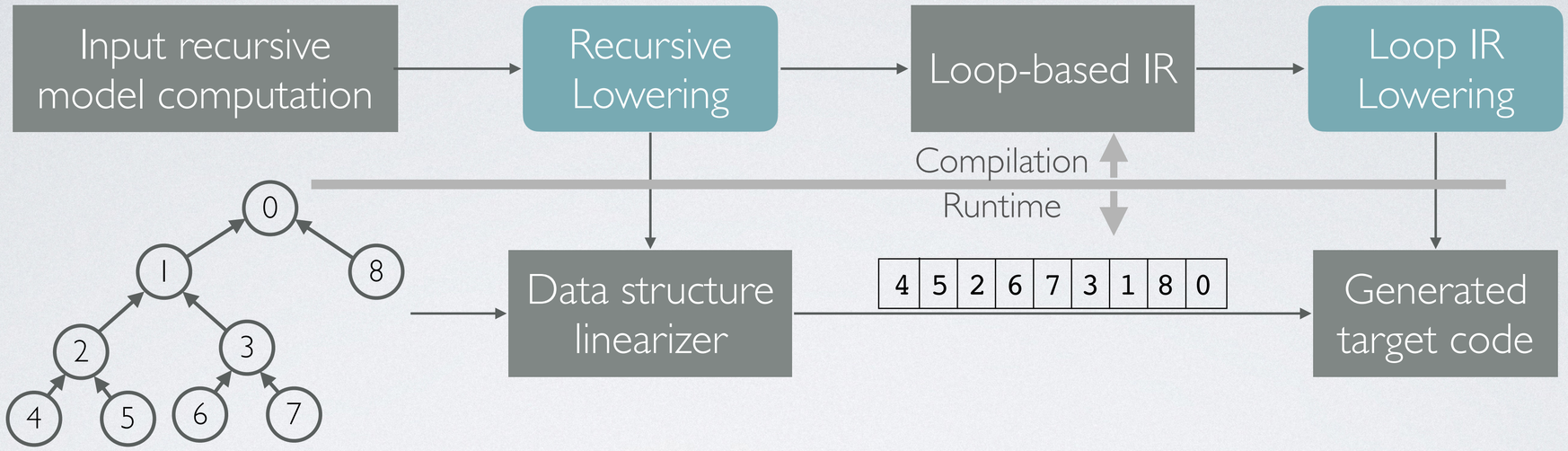
Cortex Has Low Scheduling Overheads

Framework	#Kernel calls	Scheduling time (ms)	Memory mgmt. time (ms)
DyNet	389	3.03	2.49
Cavs	122	0.4	2.01
Cortex	1	0.01	-

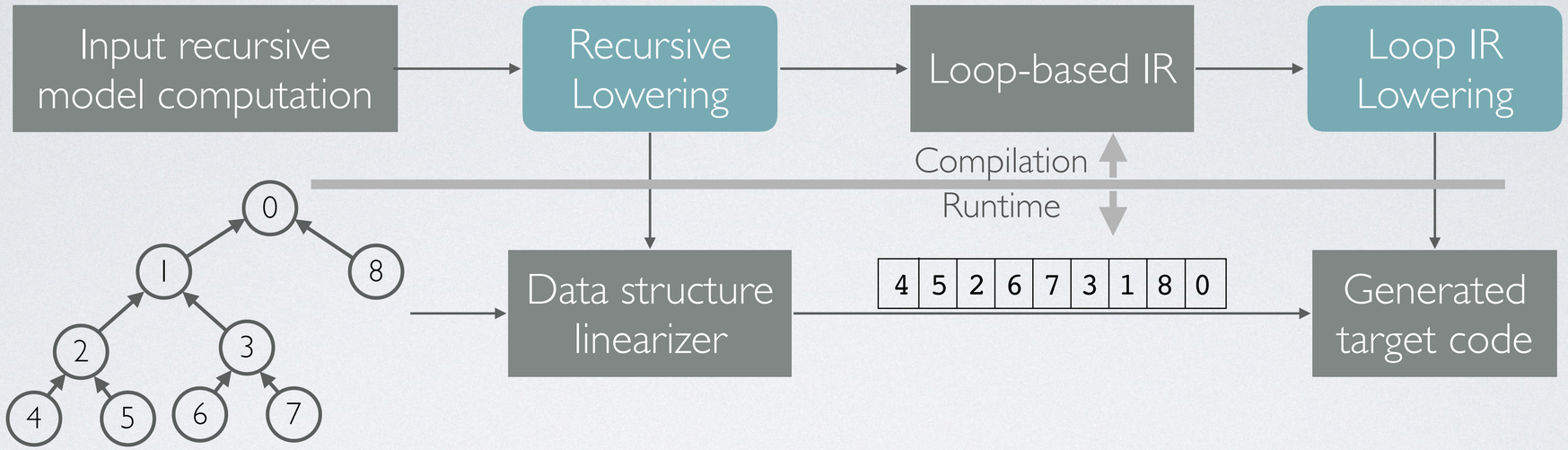
Outline

- Motivation: Inefficiencies in Execution of Recursive Models
- Cortex: Our Compiler Based Solution
  - Recursive Lowering
  - Loop IR Lowering
- Evaluation
- **Conclusion**

Conclusion

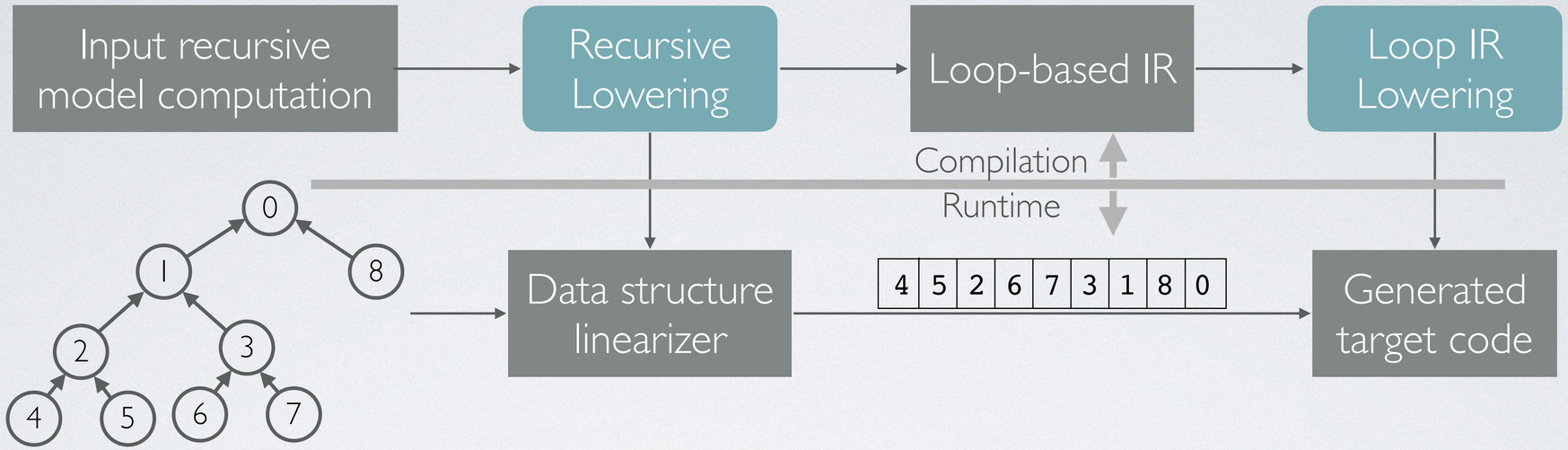

Input recursive model computation
Recursive Lowering
Loop-based IR
Loop IR Lowering
Compilation
Runtime
0
1
8
2
3
4
5
6
7
Data structure linearizer
4 5 2 6 7 3 1 8 0
Generated target code

Conclusion



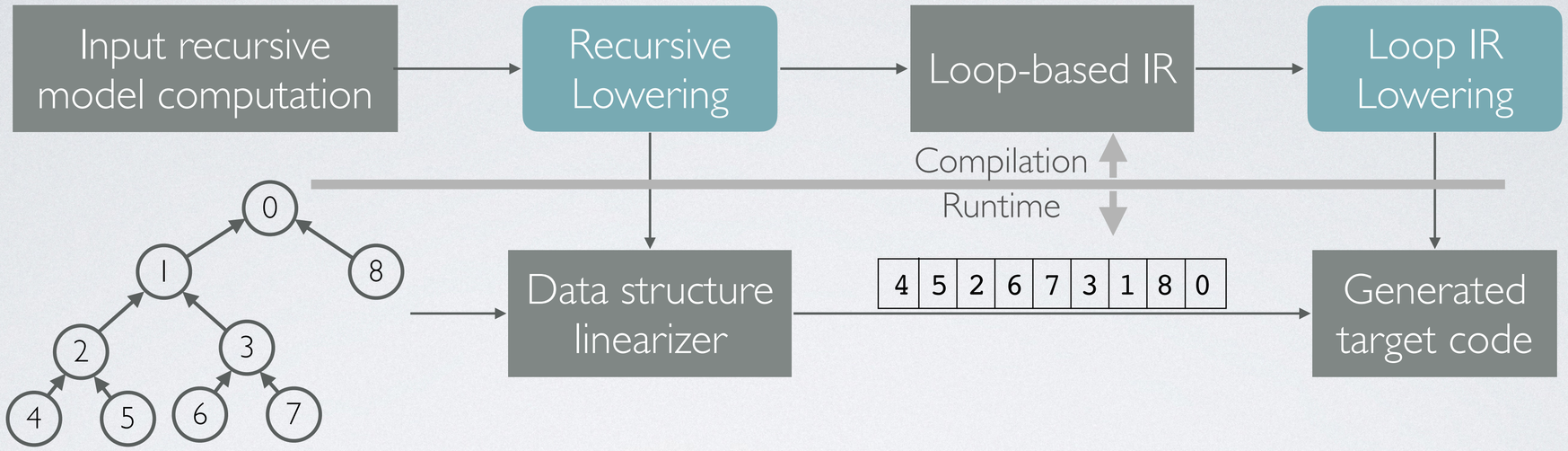
- Linearizes recursive input structures to allow for efficient loop-based code generation

Conclusion



- Linearizes recursive input structures to allow for efficient loop-based code generation
- Enables end-to-end optimizations with low overheads

Conclusion



- Linearizes recursive input structures to allow for efficient loop-based code generation
- Enables end-to-end optimizations with low overheads
- Achieves up to 14X faster inference on CPUs as well as GPUs