# JANUS: Fast and Flexible Deep Learning via Symbolic Graph Execution of Imperative Programs

**Eunji Jeong**, Sungwoo Cho, Gyeong-In Yu,
Joo Seong Jeong, Dong-Jin Shin, Byung-Gon Chun

SEOUL NATIONAL UNIVERSITY

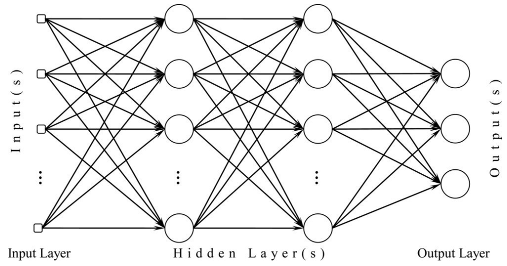
nsdi '19

SYSML 2019

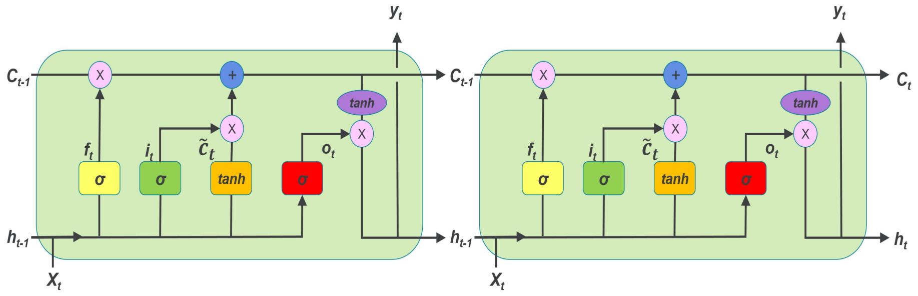
**Demo**

# Deep Learning (DL) Models

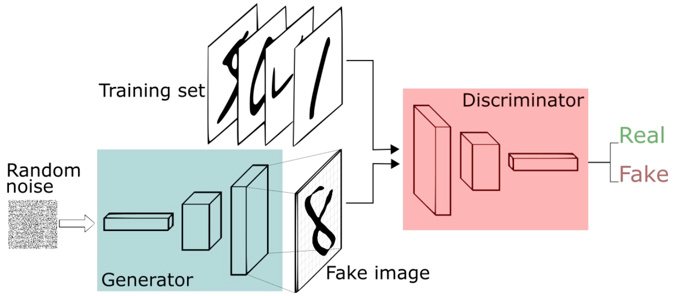
Images From:
http://www.mdpi.com/
https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/
Going Deeper with Convolutions, 2014, https://towardsdatascience.com/learn-how-recurrent-neural-networks-work-84e975feaaf7
Short-Term Load Forecasting Using EMD-LSTM Neural Networks with a Xgboost Algorithm for Feature Importance Evaluation, Energies 2017
https://skymind.ai/wiki/generative-adversarial-network-gan
https://en.wikipedia.org/wiki/Reinforcement_learning
https://medium.com/@Petuum/intro-to-dynamic-neural-networks-and-dynet-67694b18cb23



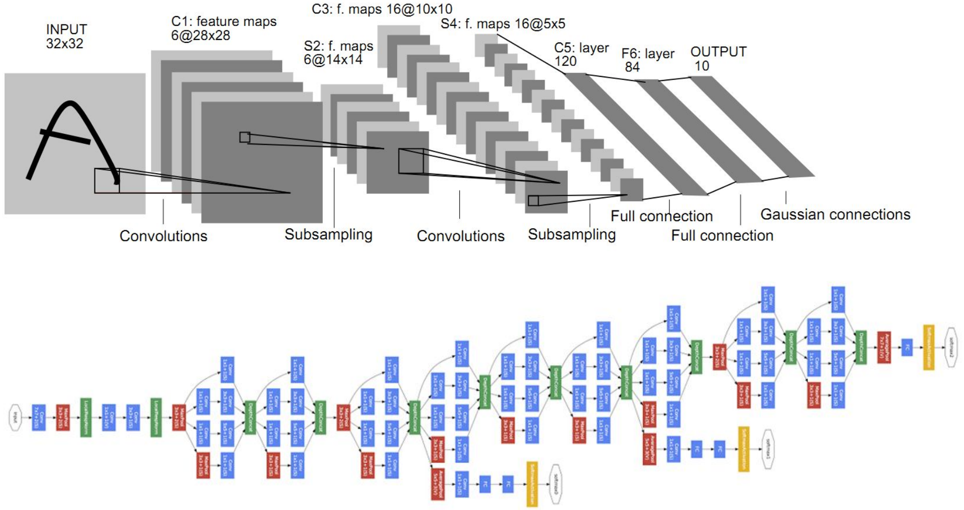

Multilayer Perceptron



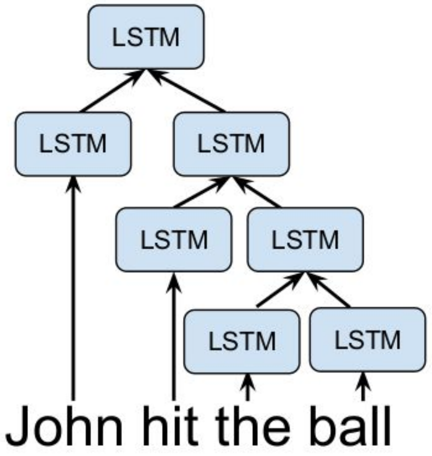

Recurrent Neural Networks



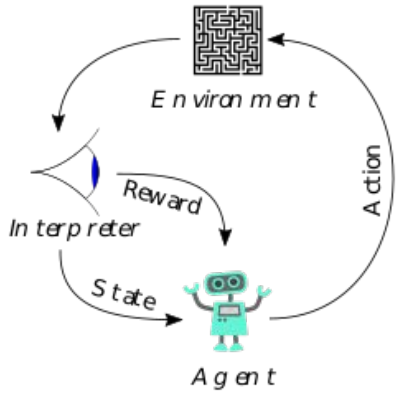

Generative Adversarial Networks




Convolutional Neural Networks




Recursive Neural Networks




Deep Reinforcement Learning Models

# Deep Learning (DL) Models

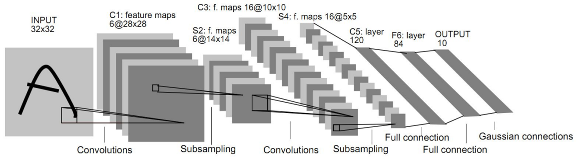
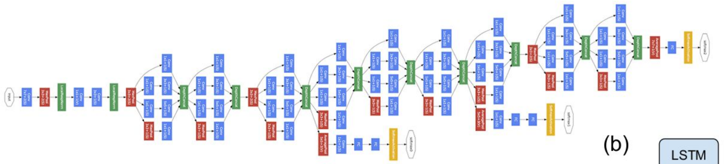
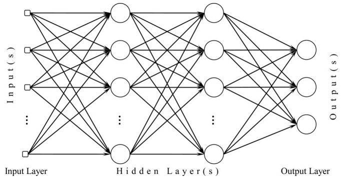
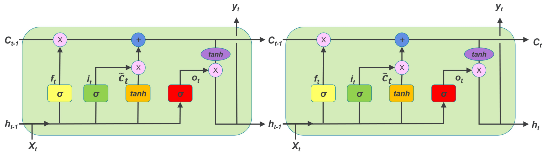
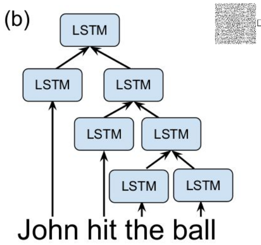


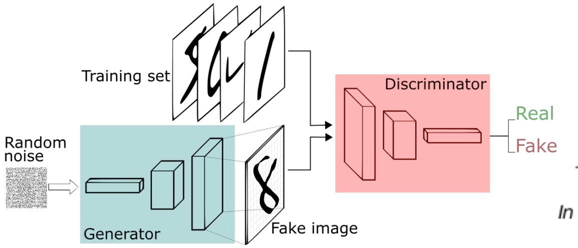

Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

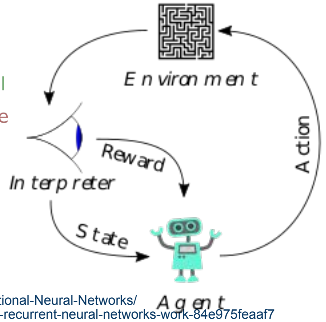




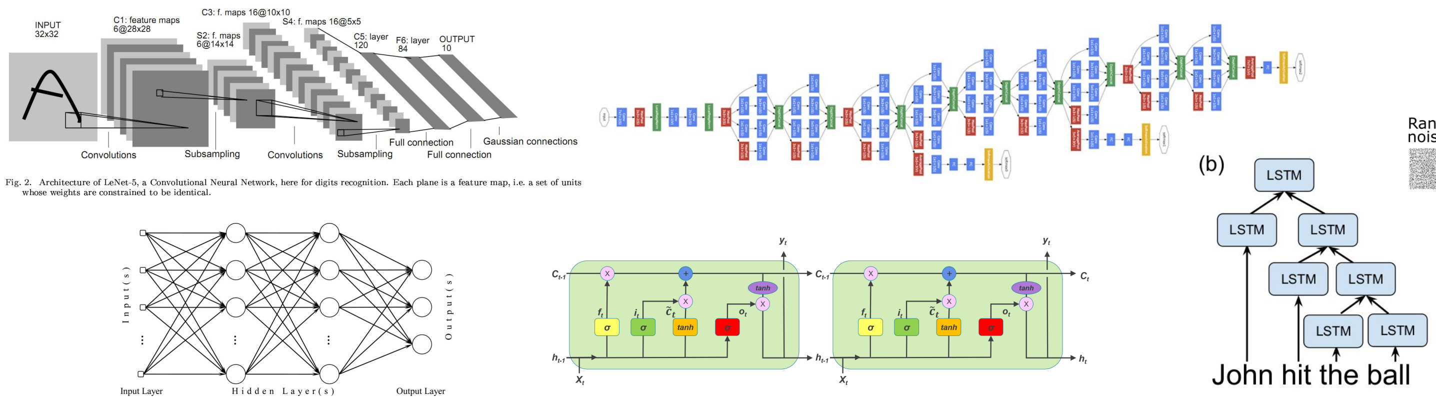



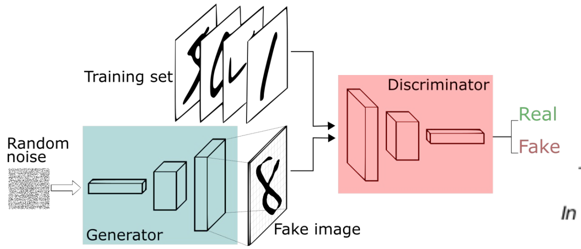



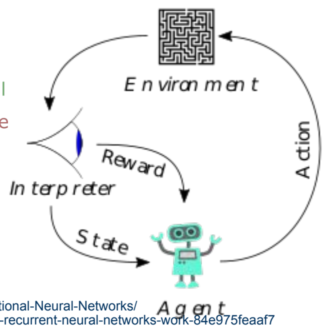




Images From:
http://www.mdpi.com/
https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/
Going Deeper with Convolutions, 2014, https://towardsdatascience.com/learn-how-recurrent-neural-networks-work-84e975feaaf7
Short-Term Load Forecasting Using EMD-LSTM Neural Networks with a Xgboost Algorithm for Feature Importance Evaluation, Energies 2017
https://skymind.ai/wiki/generative-adversarial-network-gan
https://en.wikipedia.org/wiki/Reinforcement_learning
https://medium.com/@Petuum/intro-to-dynamic-neural-networks-and-dynet-67694b18cb23

# Deep Learning (DL) Frameworks



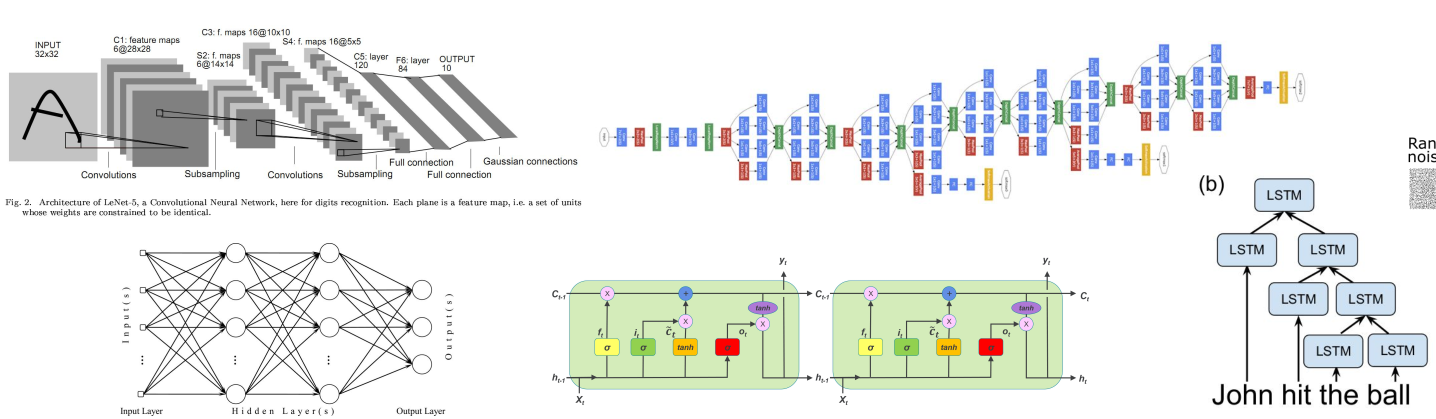



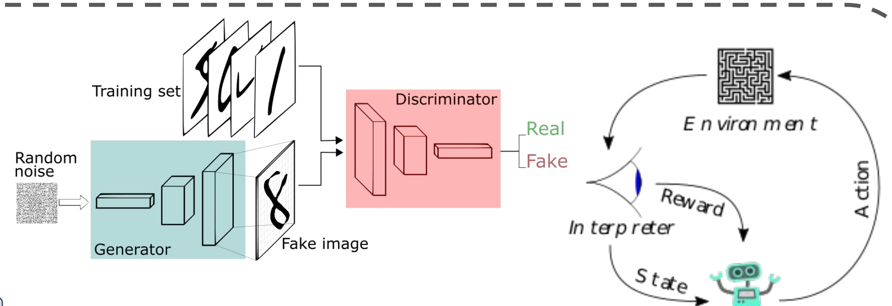

Images From:
http://www.mdpi.com/
https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/
Going Deeper with Convolutions, 2014, https://towardsdatascience.com/learn-how-recurrent-neural-networks-work-84e975feaaf7
Short-Term Load Forecasting Using EMD-LSTM Neural Networks with a Xgboost Algorithm for Feature Importance Evaluation, Energies 2017
https://skymind.ai/wiki/generative-adversarial-network-gan
https://en.wikipedia.org/wiki/Reinforcement_learning
https://medium.com/@Petuum/intro-to-dynamic-neural-networks-and-dynet-67694b18cb23

**Define & Execute**

TensorFlow Google
mxnet
Microsoft CNTK
Caffe2
julia
tvm
Chainer
ONNX
Keras
PyTorch facebook
dy/net
R
theano

# Deep Learning (DL) Frameworks




Images From:
http://www.mdpi.com/
https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/
Going Deeper with Convolutions, 2014, https://towardsdatascience.com/learn-how-recurrent-neural-networks-work-84e975feaaf7
Short-Term Load Forecasting Using EMD-LSTM Neural Networks with a Xgboost Algorithm for Feature Importance Evaluation, Energies 2017
https://skymind.ai/wiki/generative-adversarial-network-gan
https://en.wikipedia.org/wiki/Reinforcement_learning
https://medium.com/@Petuum/intro-to-dynamic-neural-networks-and-dynet-67694b18cb23

TensorFlow Google
mxnet
theano
Microsoft CNTK
tvm
Caffe2

PyTorch facebook
TensorFlow 2.0
mxnet imperative
dy/net
R
julia
Chainer

# Today’s Talk

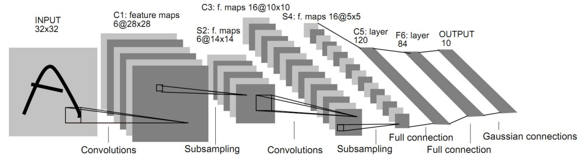
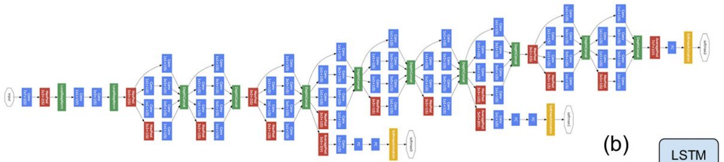
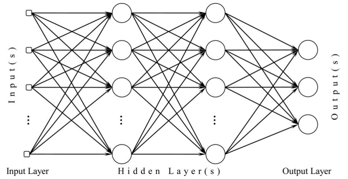
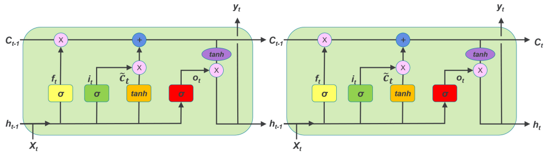
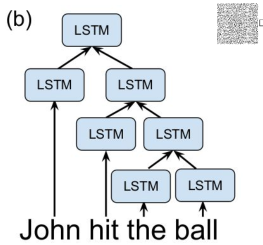


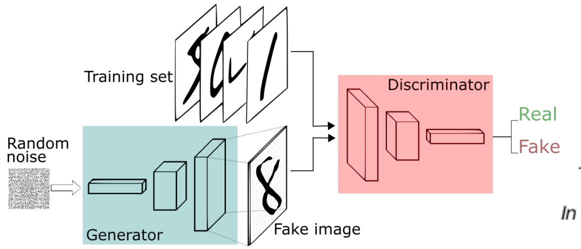

Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

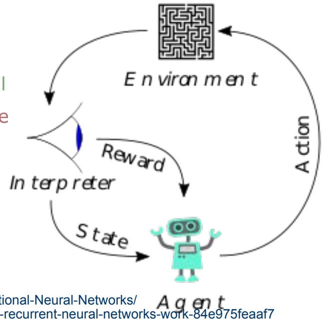







Images From:
http://www.mdpi.com/
https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/
Going Deeper with Convolutions, 2014, https://towardsdatascience.com/learn-how-recurrent-neural-networks-work-84e975feaaf7
Short-Term Load Forecasting Using EMD-LSTM Neural Networks with a Xgboost Algorithm for Feature Importance Evaluation, Energies 2017
https://skymind.ai/wiki/generative-adversarial-network-gan
https://en.wikipedia.org/wiki/Reinforcement_learning
https://medium.com/@Petuum/intro-to-dynamic-neural-networks-and-dynet-67694b18cb23

## JANUS

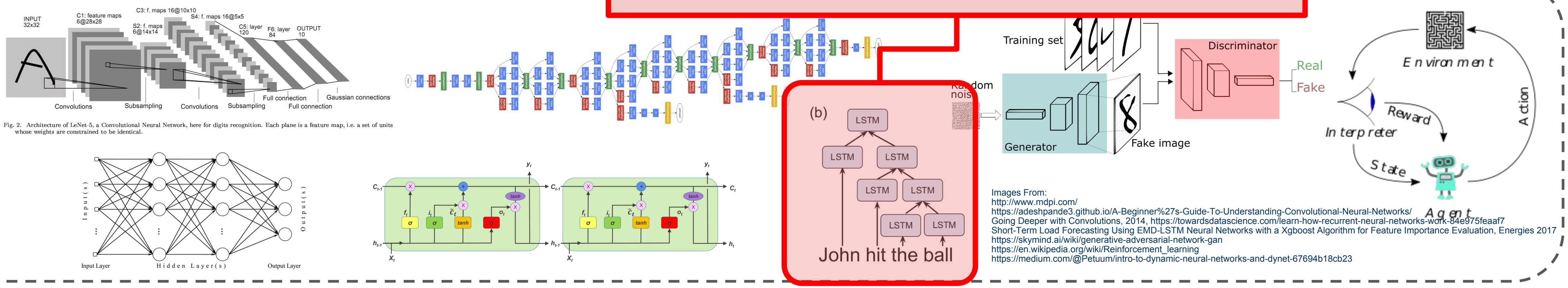
(NSDI 2019)

TensorFlow Google
mxnet
theano
Microsoft CNTK
tvm
Caffe2

PyTorch facebook
TensorFlow 2.0
mxnet imperative
dynet
R
julia
Chainer

# Today's Talk

## Recursive Neural Networks
(EuroSys 2018)




Images From:
http://www.mdpi.com/
https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/
Going Deeper with Convolutions, 2014, https://towardsdatascience.com/learn-how-recurrent-neural-networks-work-84e975feaaf7
Short-Term Load Forecasting Using EMD-LSTM Neural Networks with a Xgboost Algorithm for Feature Importance Evaluation, Energies 2017
https://skymind.ai/wiki/generative-adversarial-network-gan
https://en.wikipedia.org/wiki/Reinforcement_learning
https://medium.com/@Petuum/intro-to-dynamic-neural-networks-and-dynet-67694b18cb23

## JANUS
(NSDI 2019, SysML 2019)

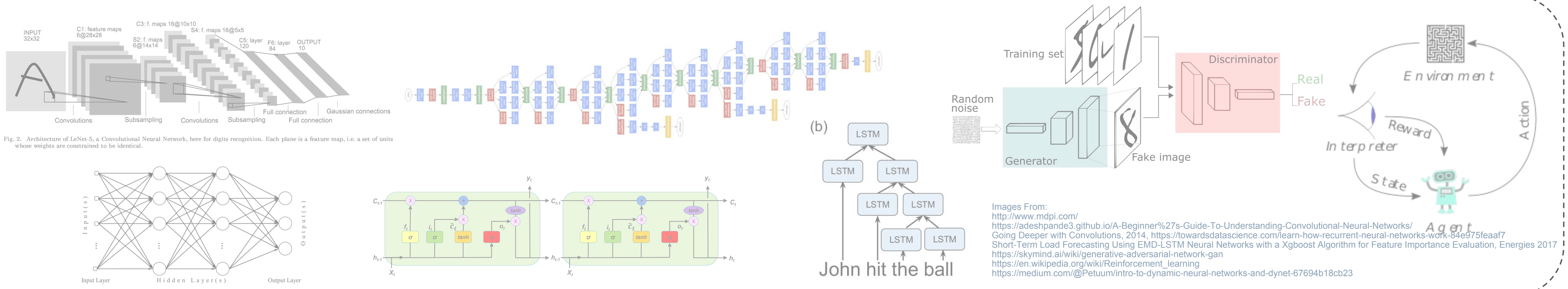
TensorFlow Google
mxnet
theano
Microsoft CNTK
tvm
Caffe2

PyTorch facebook
TensorFlow 2.0
mxnet imperative
dy/net
R
julia
Chainer

# Outline

- **JANUS**
- How to handle Recursive Neural Networks?
- On-going Works

# Two Paradigms




Images From:
http://www.mdpi.com/
https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/
Going Deeper with Convolutions, 2014, https://towardsdatascience.com/learn-how-recurrent-neural-networks-work-84e975feaaf7
Short-Term Load Forecasting Using EMD-LSTM Neural Networks with a Xgboost Algorithm for Feature Importance Evaluation, Energies 2017
https://skymind.ai/wiki/generative-adversarial-network-gan
https://en.wikipedia.org/wiki/Reinforcement_learning
https://medium.com/@Petuum/intro-to-dynamic-neural-networks-and-dynet-67694b18cb23

## Symbolic DL Frameworks

TensorFlow Google
mxnet
theano
Microsoft CNTK
tvm
Caffe2

## Imperative DL Frameworks

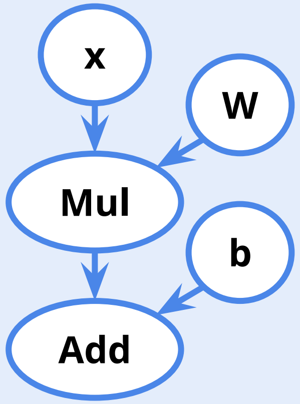
PyTorch facebook
TensorFlow 2.0
mxnet imperative
dy/net
R
julia
Chainer

# Two Paradigms

## Symbolic DL Frameworks

✓ Build a Symbolic Graph
✓ Execute the Graph

```
def build_graph(g):
  x = g.input(float)
  linear = g.add(g.mul(W, x), b)

build_graph(graph)
run_graph(graph, x_data)
```




Google TensorFlow
Apache mxnet
facebook Caffe2
Microsoft CNTK

## Imperative DL Frameworks

✓ Directly Execute the Computations

```
def linear(x):
  return W * x + b
linear(x_data)
```

facebook PyTorch
Google TensorFlow 2.0
Apache mxnet imperative
dynet
Chainer

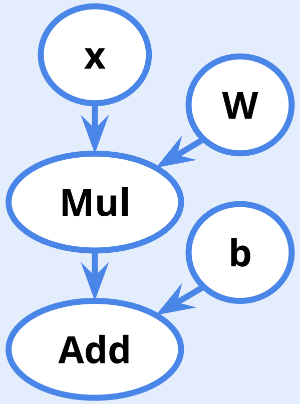
# Two Paradigms

## Symbolic DL Frameworks

✓ Build a Symbolic Graph

✓ Execute the Graph

```
def build_graph(g):
  x = g.input(float)
  linear = g.add(g.mul(W, x), b)

build_graph(graph)
run_graph(graph, x_data)
```




Google TensorFlow · Apache mxnet · facebook Caffe2 · Microsoft CNTK

## Imperative DL Frameworks

✓ Directly Execute the Computations

```
def linear(x):
  return W * x + b
linear(x_data)
```

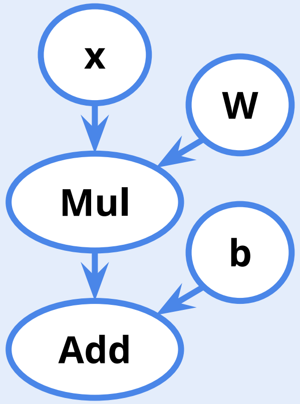
facebook PyTorch · Google TensorFlow 2.0 · Apache mxnet imperative · dynet · Chainer

# Two Paradigms

## Symbolic DL Frameworks

- ✓ Build a Symbolic Graph
- ✓ Execute the Graph

```
def build_graph(g):
  x = g.input(float)
  linear = g.add(g.mul(W, x), b)

build_graph(graph)
run_graph(graph, x_data)
```




Google TensorFlow · Apache mxnet · facebook Caffe2 · Microsoft CNTK

## Imperative DL Frameworks

- ✓ Directly Execute the Computations

```
def linear(x):
  return W * x + b
linear(x_data)
```

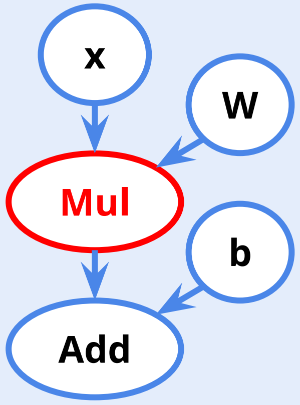
facebook PyTorch · Google TensorFlow 2.0 · Apache mxnet imperative · dynet · Chainer

# Two Paradigms

## Symbolic DL Frameworks

- ✓ Build a Symbolic Graph
- ✓ Execute the Graph

```
def build_graph(g):
  x = g.input(float)
  linear = g.add(g.mul(W, x), b)

build_graph(graph)
run_graph(graph, x_data)
```




Google TensorFlow · Apache mxnet · facebook Caffe2 · Microsoft CNTK

## Imperative DL Frameworks

- ✓ Directly Execute the Computations

```
def linear(x):
  return W * x + b
linear(x_data)
```

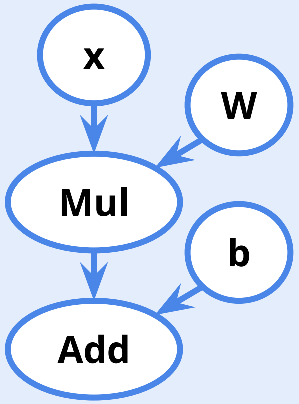
facebook PyTorch · Google TensorFlow 2.0 · Apache mxnet imperative · dy/net · Chainer

# Two Paradigms

## Symbolic DL Frameworks

✓ Build a Symbolic Graph
✓ Execute the Graph

```
def build_graph(g):
  x = g.input(float)
  linear = g.add(g.mul(W, x), b)

build_graph(graph)
run_graph(graph, x_data)
```




Google TensorFlow
APACHE SOFTWARE FOUNDATION mxnet
facebook Caffe2
Microsoft CNTK

## Imperative DL Frameworks

✓ Directly Execute the Computations

```
def linear(x):
  return W * x + b
linear(x_data)
```

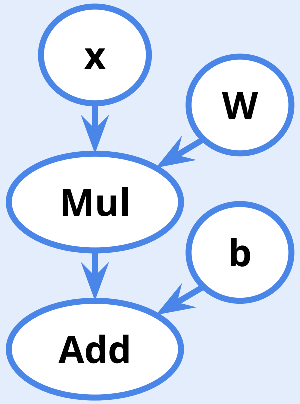
facebook PyTorch
Google TensorFlow 2.0
APACHE SOFTWARE FOUNDATION mxnet imperative
dy/net
Chainer

# Two Paradigms

## Symbolic DL Frameworks

- ✓ Build a Symbolic Graph
- ✓ Execute the Graph

```
def build_graph(g):
  x = g.input(float)
  linear = g.add(g.mul(W, x), b)

build_graph(graph)
run_graph(graph, x_data)
```




Google TensorFlow · Apache mxnet · facebook Caffe2 · Microsoft CNTK

## Imperative DL Frameworks

- ✓ Directly Execute the Computations

```
def linear(x):
  return W * x + b
linear(x_data)
```

facebook PyTorch · Google TensorFlow 2.0 · Apache mxnet imperative · dynet · Chainer

## Pros & Cons



| | Symbolic DL Frameworks | Imperative DL Frameworks |
|---|---|---|
| **Pros** | + Easy to Optimize<br>+ Compiler Optimization<br>+ Parallel Execution of Operations<br>+ Deploy on GPU, Cluster, Mobile, ... | + Direct Execution:<br>Easy to Program & Debug |
| **Cons** | - Decoupled View:<br>Hard to Program & Debug | - Hard to Optimize |

# Pros & Cons

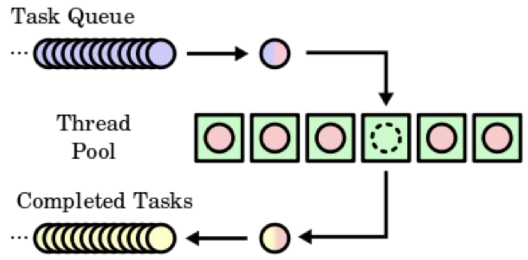
Performance | Programmability

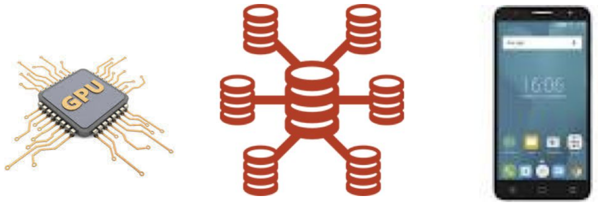
| | Symbolic DL Frameworks | Imperative DL Frameworks |
|---|---|---|
| **Pros** | + **Easy to Optimize**<br>+ **Compiler Optimization**<br>+ **Parallel Execution of Operations**<br>+ **Deploy on GPU, Cluster, Mobile,...** | + Direct Execution:<br>Easy to Program & Debug |
| **Cons** | - Decoupled View:<br>Hard to Program & Debug | - **Hard to Optimize** |

# Pros & Cons

Performance | Programmability

| | Symbolic DL Frameworks | eworks |
|---|---|---|
| **Pros** | + **Easy to Optimize**<br>+ **Compiler Optimization**<br>+ **Parallel Execution of Operations**<br>+ **Deploy on GPU, Cluster, Mobile,…** | + Direct Execution:<br>bug |
| **Cons** | - Decoupled View:<br>Hard to Program & Debug | - |



# Pros & Cons

Performance | Programmability

| | Symbolic DL Frameworks | Imperative DL Frameworks |
|---|---|---|
| **Pros** | + **Easy to Optimize**<br>+ **Compiler Optimization**<br>+ **Parallel Execution of Operations**<br>+ **Deploy on GPU, Cluster, Mobile,…** | + Direct Execution:<br>Easy to Program & Debug |
| **Cons** | - Decoupled View:<br>Hard to Program & Debug | - **Hard to Optimize** |

# Pros & Cons

Performance | **Programmability**

| | Symbolic DL Frameworks | Imperative DL Frameworks |
|---|---|---|
| **Pros** | + Easy to Optimize<br>+ Compiler Optimization<br>+ Parallel Execution of Operations<br>+ Deploy on GPU, Cluster, Mobile, … | **+ Direct Execution: Easy to Program & Debug** |
| **Cons** | **- Decoupled View: Hard to Program & Debug** | - Hard to Optimize |

# Pros & Cons

Performance | **Programmability**

| | Symbolic DL Frameworks | Imperative DL Frameworks |
|---|---|---|
| **Pros** | + Easy to Optimize<br>  + Compiler Optimization<br>  + Parallel Execution of Operations<br>  + Deploy on GPU, Cluster, Mobile, … | **+ Direct Execution: Easy to Program & Debug** |
| **Cons** | **- Decoupled View: Hard to Program & Debug** | - Hard to Optimize |

# What People Want Is...

Performance | Programmability

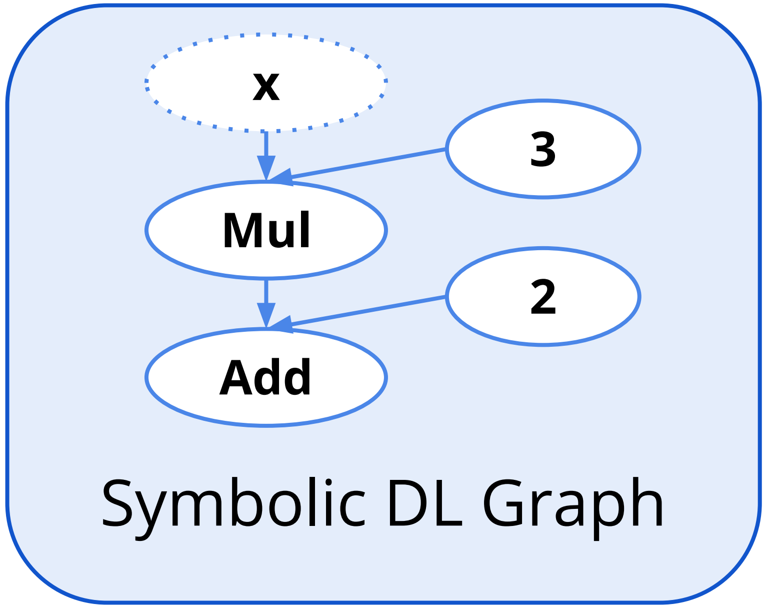
| | Symbolic DL Frameworks | Imperative DL Frameworks |
|---|---|---|
| **Pros** | + Easy to Optimize<br>+ Compiler Optimization<br>+ Parallel Execution of Operations<br>+ Deploy on GPU, Cluster, Mobile,... | + Direct Execution:<br>Easy to Program & Debug |
| **Cons** | - Decoupled View:<br>Hard to Program & Debug | - Hard to Optimize |

# JANUS: Combining the Best of Both Worlds

Imperative DL Program

```
def foo(x):
  prod = mul(3, x)
  return add(prod, 2)
```

**Transparent Conversion**




***"Easy Programmability"***

***"High Performance"***

# JANUS: Combining the Best of Both Worlds

- 11 models in 5 major neural network categories:
    - Convolutional Neural Networks (**CNN**) LeNet, ResNet-50, Inception-v3
    - Recurrent Neural Networks (**RNN**) LSTM, LM
    - Recursive Neural Networks (**TreeNN**) TreeRNN, TreeLSTM
    - Generative Adversarial Networks (**GAN**) GAN, PIX2PIX
    - Deep Reinforcement Learning (**DRL**) A3C, PPO

- **Up to 47.6x speedup** compared to imperative DL framework, **comparable performance (within 4%)** to symbolic DL framework with unmodified imperative DL programs

# Outline

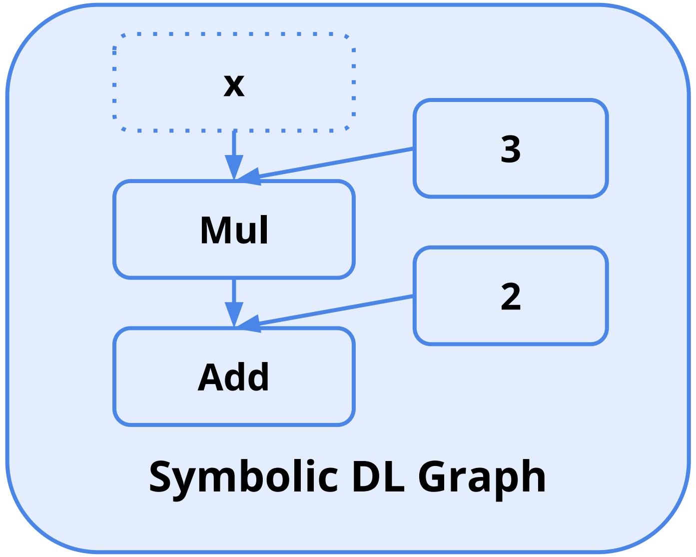
- **JANUS**
    - Approach
    - **Challenges**
    - Our Solution
    - Evaluation
- How to handle Recursive Neural Networks?
- On-going Works

# Challenges in Graph Conversion

**Imperative DL Program**

```
def foo(x):
  tmp = mul(3, x)
  return add(tmp, 2)
```

**Transparent Conversion**



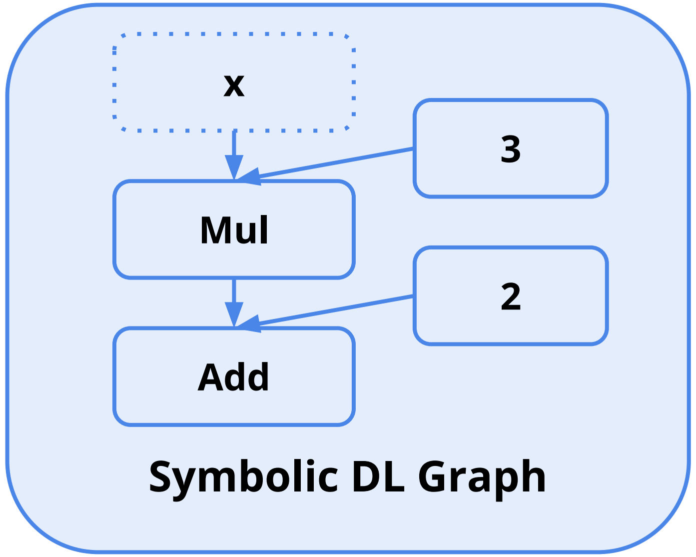
# Challenges in Graph Conversion

**De-facto Standard Language for DL Programming**

**Imperative Python DL Program**

```
def foo(x):
  tmp = mul(3, x)
  return add(tmp, 2)
```

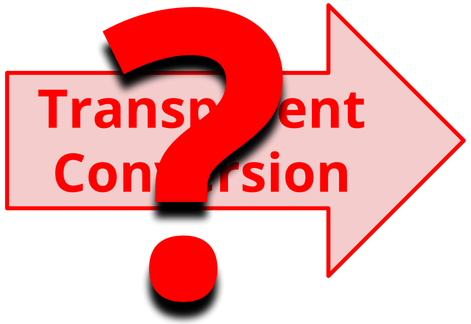
**Transparent Conversion**

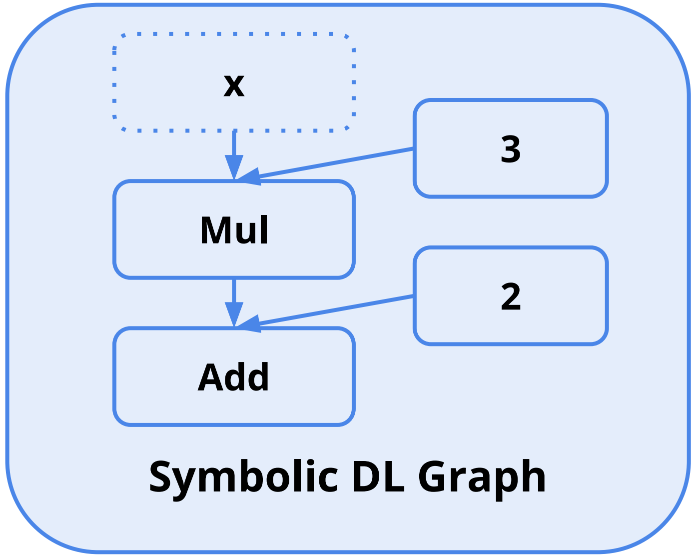


# Challenges in Graph Conversion

**Imperative Python DL Program**

```
def foo(x):
  tmp = mul(3, x)
  return add(tmp, 2)
```



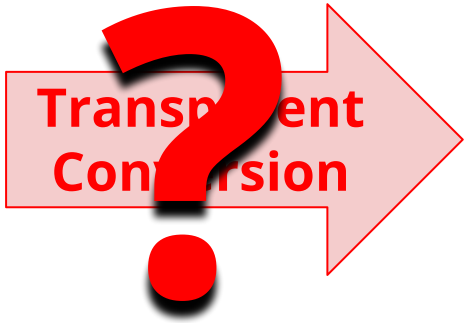



# Discrepancy between Python Programs and DL Graphs

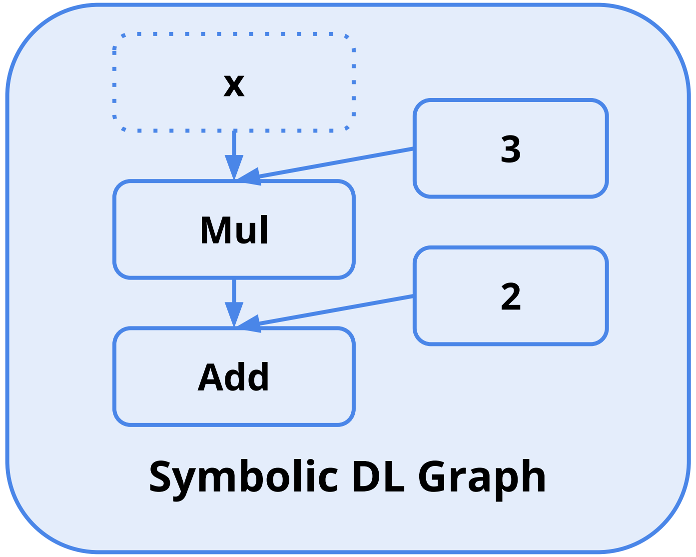
**"Dynamic"**

**Imperative Python DL Program**

```
def foo(x):
  tmp = mul(3, x)
  return add(tmp, 2)
```



**"Static"**

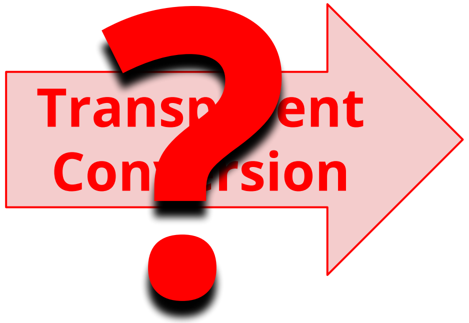


# Discrepancy between Python Programs and DL Graphs

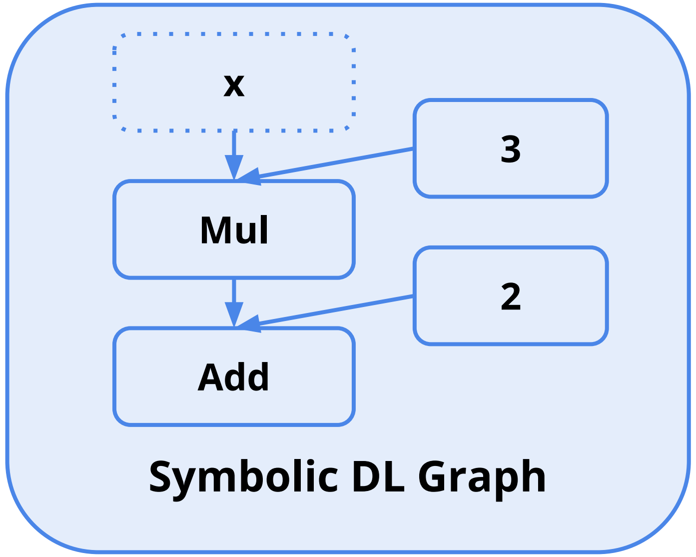
*"Dynamic"*

**Imperative Python DL Program**

```
def foo(x):
  tmp = mul(3, x)
  return add(tmp, 2)
```



*"Static"*




**Characteristics**

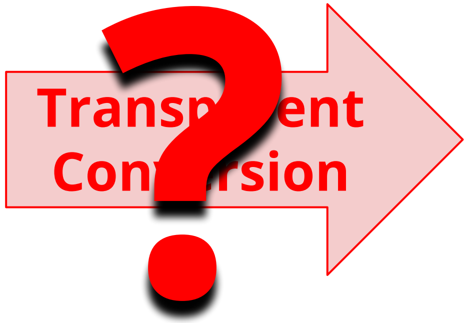
- **determined at runtime**
- **change at runtime**

# Discrepancy between Python Programs and DL Graphs

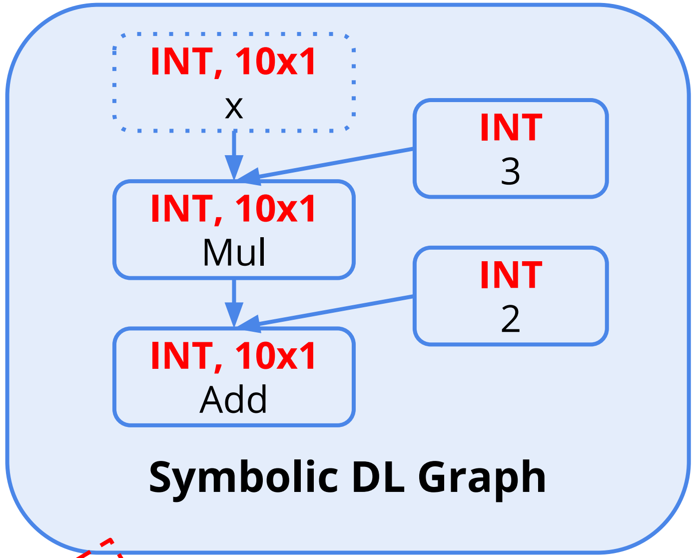
***"Dynamic"***

**Imperative Python DL Program**

```
def foo(x):
  tmp = mul(3, x)
  return add(tmp, 2)
```




***"Static"***



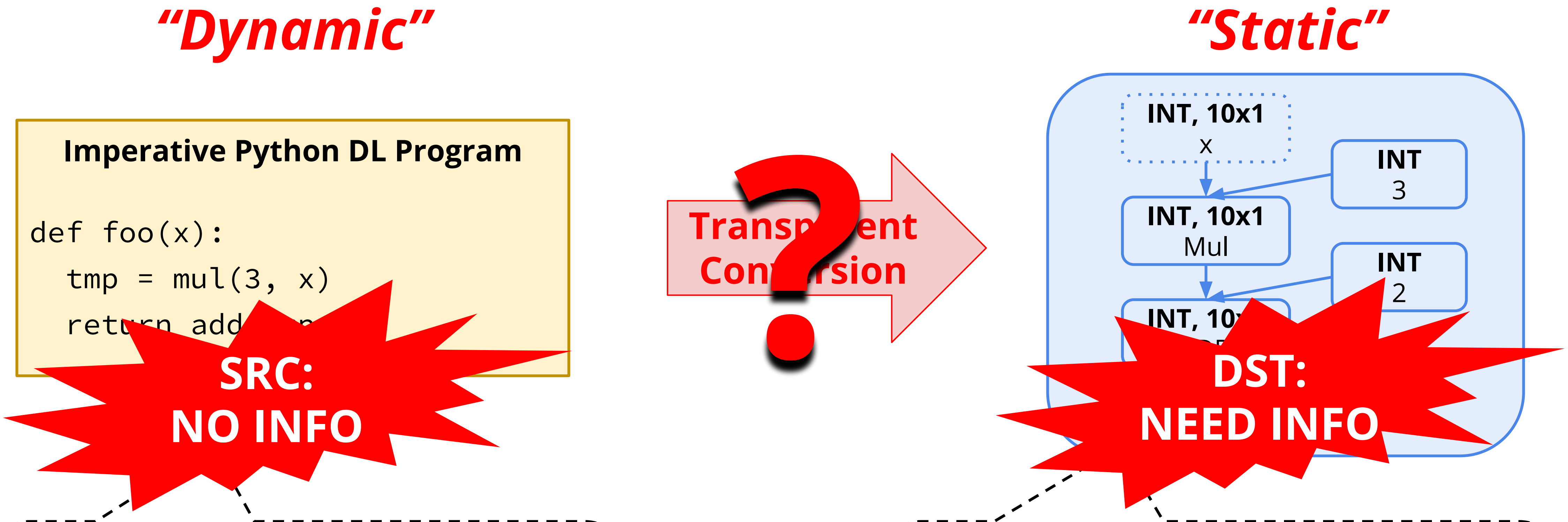

**Characteristics**

- **determined at runtime**
- **change at runtime**

**Characteristics**

- **must be given when building a graph**

# Discrepancy between Python Programs and DL Graphs




**Characteristics**

- **determined at runtime**
- **change at runtime**

**Characteristics**

- **must be given when building a graph**

## Example: Recurrent Neural Network (RNN)

```python
class RNNModel(object):
  def __call__(self, sequence):
    state = self.state
    outputs = []
    for item in sequence:
      state = rnn_cell(state, item)
      outputs += [state]
    self.state = state
    return compute_loss(outputs)

for sequence in sequences:
  optimize(lambda: model(sequence))
```

**Dynamic Features of Python**

√ Dynamic Control Flow

√ Dynamic Types

√ Impure Functions

?

**Correctness & Performance**

of Graph Execution

# RNN Example

Dynamic Control Flow | Dynamic Types | Impure Function

```
class RNNModel(object):
  def __call__(self, sequence):
    state = self.state
    outputs = []
    for item in sequence:
      state = rnn_cell(state, item)
      outputs += [state]
    self.state = state
    return compute_loss(outputs)

for sequence in sequences:
  optimize(lambda: model(sequence))
```

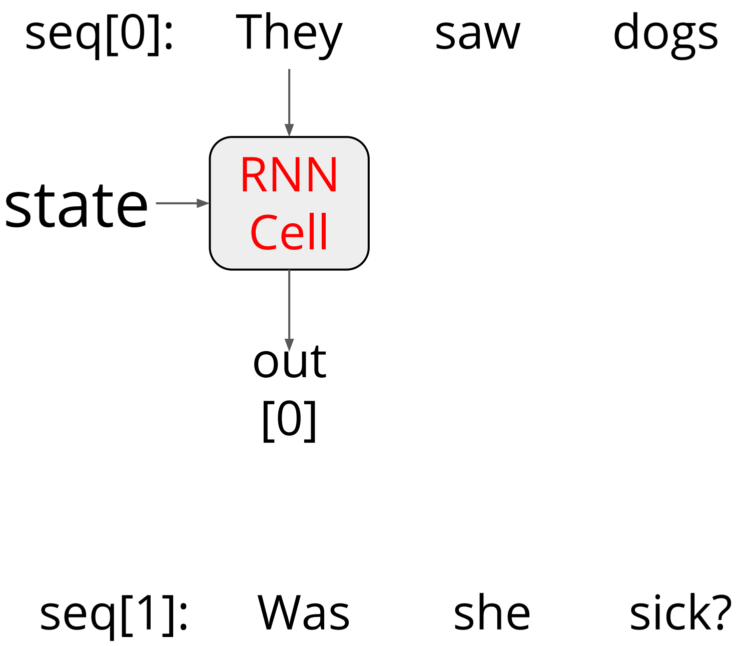
seq[0]: They saw dogs

seq[1]: Was she sick?

## RNN Example

Dynamic Control Flow | Dynamic Types | Impure Function

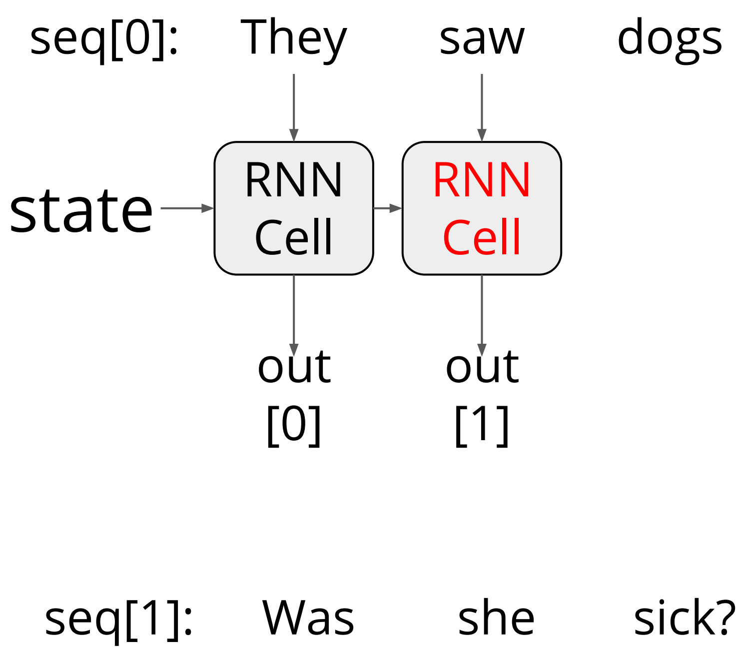
```
class RNNModel(object):
  def __call__(self, sequence):
    state = self.state
    outputs = []
    for item in sequence:
      state = rnn_cell(state, item)
      outputs += [state]
    self.state = state
    return compute_loss(outputs)

for sequence in sequences:
  optimize(lambda: model(sequence))
```



# RNN Example

**Dynamic Control Flow** | Dynamic Types | Impure Function

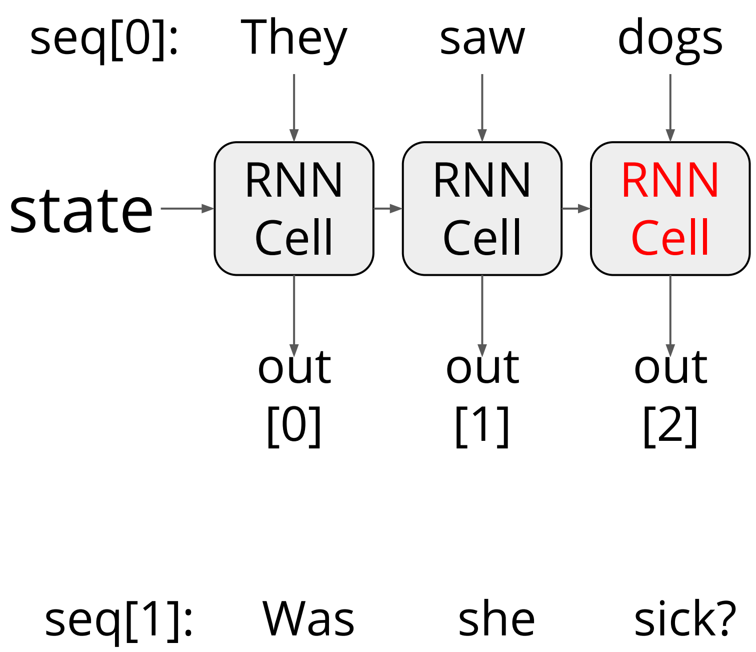
```
class RNNModel(object):
  def __call__(self, sequence):
    state = self.state
    outputs = []
    for item in sequence:
      state = rnn_cell(state, item)
      outputs += [state]
    self.state = state
    return compute_loss(outputs)

for sequence in sequences:
  optimize(lambda: model(sequence))
```



# RNN Example



```
class RNNModel(object):
  def __call__(self, sequence):
    state = self.state
    outputs = []
    for item in sequence:
      state = rnn_cell(state, item)
      outputs += [state]
    self.state = state
    return compute_loss(outputs)

for sequence in sequences:
  optimize(lambda: model(sequence))
```



# RNN Example

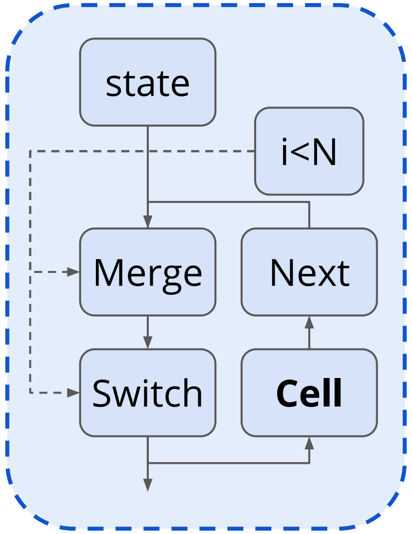
Dynamic Control Flow | Dynamic Types | Impure Function

```
class RNNModel(object):
  def __call__(self, sequence):
    state = self.state
    outputs = []
    for item in sequence:
      state = rnn_cell(state, item)
      outputs += [state]
    self.state = state
    return compute_loss(outputs)

for sequence in sequences:
  optimize(lambda: model(sequence))
```

# RNN Example

Dynamic Control Flow | Dynamic Types | Impure Function

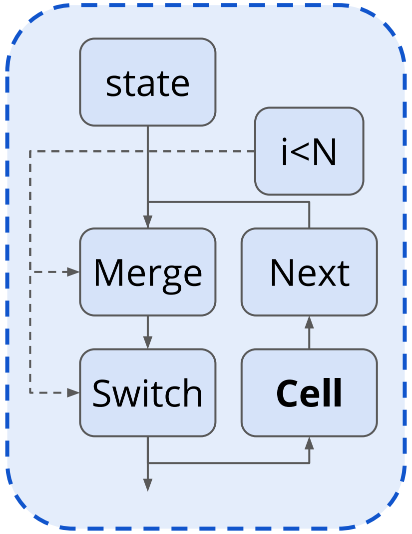
```
class RNNModel(object):
  def __call__(self, sequence):
    state = self.state
    outputs = []
    for item in sequence:
      state = rnn_cell(state, item)
      outputs += [state]
    self.state = state
    return compute_loss(outputs)

for sequence in sequences:
  optimize(lambda: model(sequence))
```



# RNN Example

**Dynamic Control Flow** | Dynamic Types | Impure Function

```
class RNNModel(object):
  def __call__(self, sequence):
    state = self.state
    outputs = []
    for item in sequence:
      state = rnn_cell(state, item)
      outputs += [state]
    self.state = state
    return compute_loss(outputs)

for sequence in sequences:
  optimize(lambda: model(sequence))
```

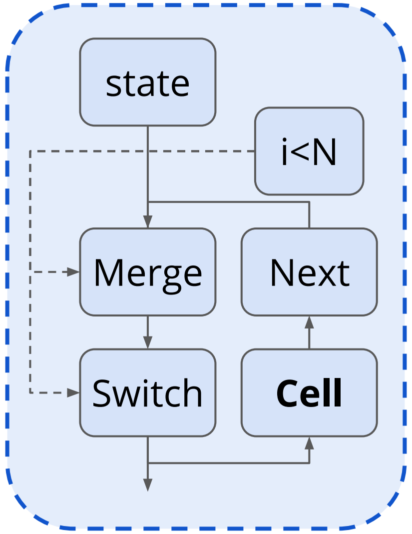



- Correct☺

# RNN Example

**Dynamic Control Flow** | Dynamic Types | Impure Function

```
class RNNModel(object):
  def __call__(self, sequence):
    state = self.state
    outputs = []
    for item in sequence:
      state = rnn_cell(state, item)
      outputs += [state]
    self.state = state
    return compute_loss(outputs)

for sequence in sequences:
  optimize(lambda: model(sequence))
```

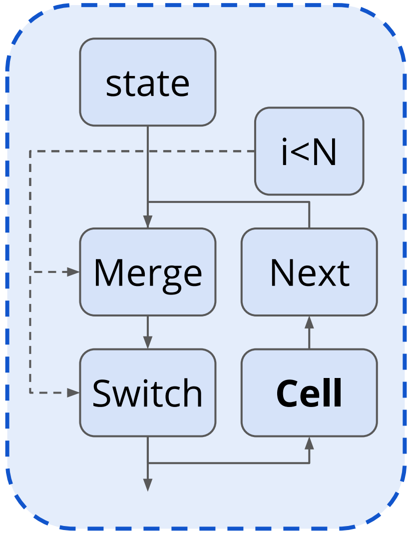



- Correct ☺
- Slow ☹

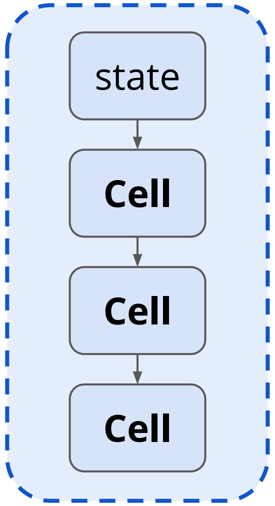
# RNN Example

Dynamic Control Flow | Dynamic Types | Impure Function

```
class RNNModel(object):
  def __call__(self, sequence):
    state = self.state
    outputs = []
    for item in sequence:
      state = rnn_cell(state, item)
      outputs += [state]
    self.state = state
    return compute_loss(outputs)

for sequence in sequences:
  optimize(lambda: model(sequence))
```




- Correct ☺
- Slow ☹

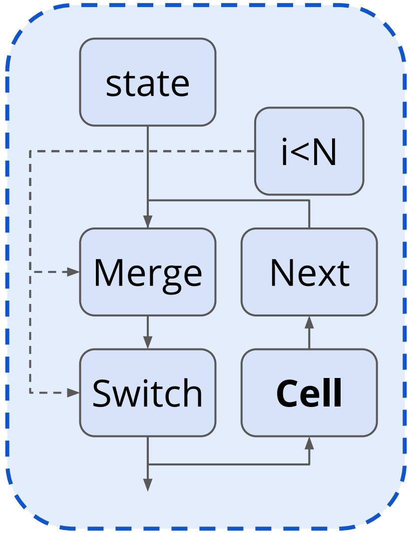



- Fast ☺

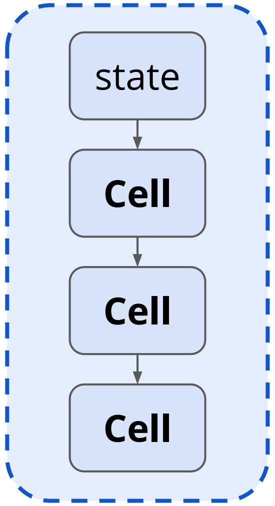
# RNN Example

**Dynamic Control Flow** | Dynamic Types | Impure Function

```python
class RNNModel(object):
  def __call__(self, sequence):
    state = self.state
    outputs = []
    for item in sequence:
      state = rnn_cell(state, item)
      outputs += [state]
    self.state = state
    return compute_loss(outputs)

for sequence in sequences:
  optimize(lambda: model(sequence))
```




- Correct ☺
- Slow ☹

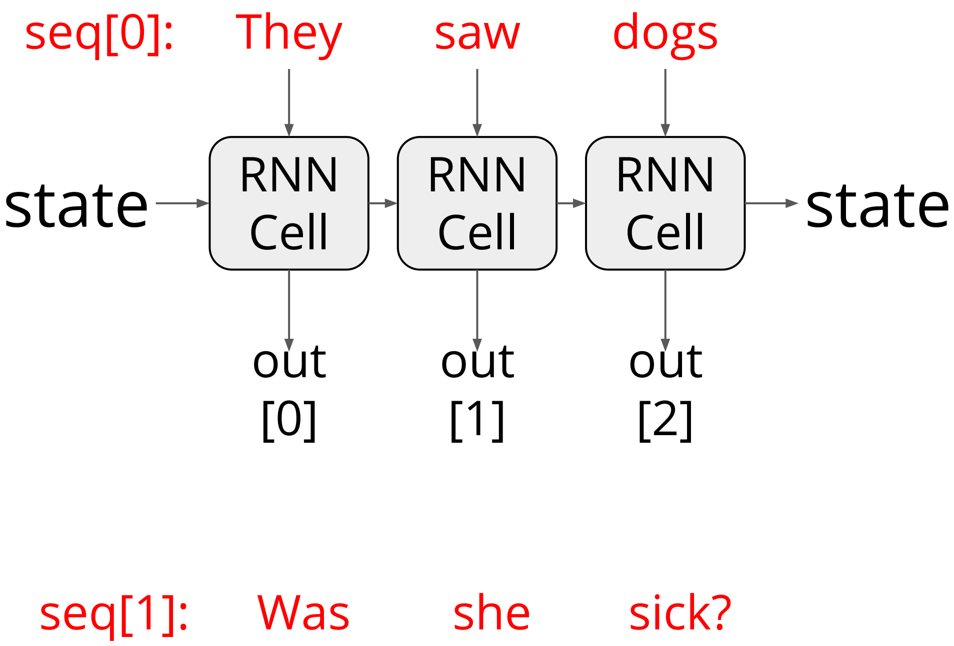



- Fast ☺
- Incorrect ☹
- Need Info ☹

## RNN Example

Dynamic Control Flow | **Dynamic Types** | Impure Function

```python
class RNNModel(object):
    def __call__(self, sequence):
        state = self.state
        outputs = []
        for item in sequence:
            state = rnn_cell(state, item)
            outputs += [state]
        self.state = state
        return compute_loss(outputs)

for sequence in sequences:
    optimize(lambda: model(sequence))
```



# RNN Example

Dynamic Control Flow | **Dynamic Types** | Impure Function

```
class RNNModel(object):
  def __call__(self, sequence):
    state = self.state
    outputs = []
    for item in sequence:
      state = rnn_cell(state, item)
      outputs += [state]
    self.state = state
    return compute_loss(outputs)

for sequence in sequences:
  optimize(lambda: model(sequence))
```

**PlaceHolder**
type: **int**
shape: **?**

**PlaceHolder**
type: **float**
shape: **?**

⋮

- Correct ☺
- Inefficient ☹

# RNN Example



```
class RNNModel(object):
  def __call__(self, sequence):
    state = self.state
    outputs = []
    for item in sequence:
      state = rnn_cell(state, item)
      outputs += [state]
    self.state = state
    return compute_loss(outputs)

for sequence in sequences:
  optimize(lambda: model(sequence))
```

**PlaceHolder**
type: **int**
shape: **?**

**PlaceHolder**
type: **float**
shape: **?**

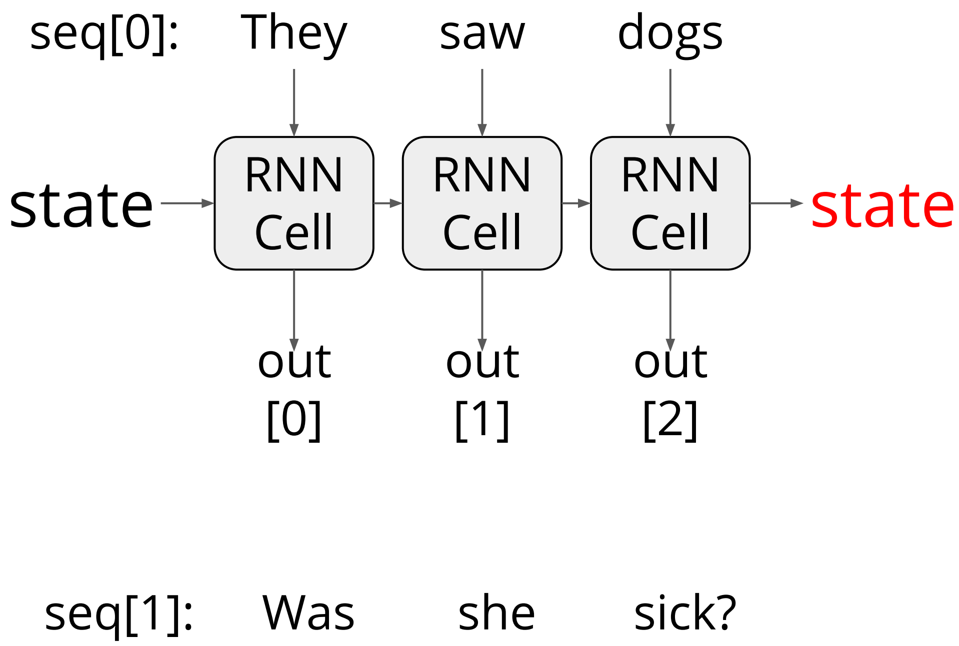
⋮

- Correct ☺
- Inefficient ☹

**PlaceHolder**
type: **int**
shape: **(3x128)**

- Fast ☺
- Incorrect ☹
- Need Info ☹

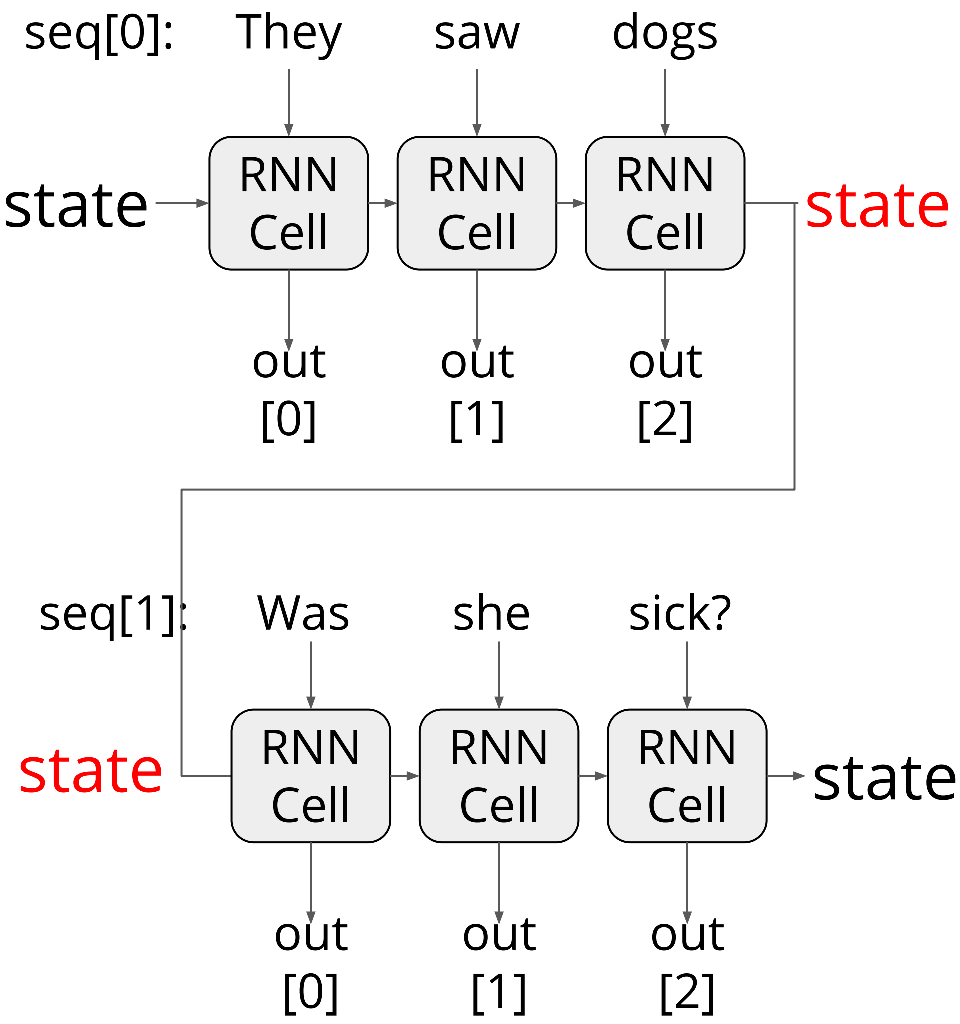
```python
class RNNModel(object):
  def __call__(self, sequence):
    state = self.state
    outputs = []
    for item in sequence:
      state = rnn_cell(state, item)
      outputs += [state]
    self.state = state
    return compute_loss(outputs)

for sequence in sequences:
  optimize(lambda: model(sequence))
```



# RNN Example

Dynamic Control Flow | Dynamic Types | **Impure Function**

```
class RNNModel(object):
  def __call__(self, sequence):
    state = self.state
    outputs = []
    for item in sequence:
      state = rnn_cell(state, item)
      outputs += [state]
    self.state = state
    return compute_loss(outputs)

for sequence in sequences:
  optimize(lambda: model(sequence))
```



# RNN Example

Dynamic Control Flow | Dynamic Types | **Impure Function**

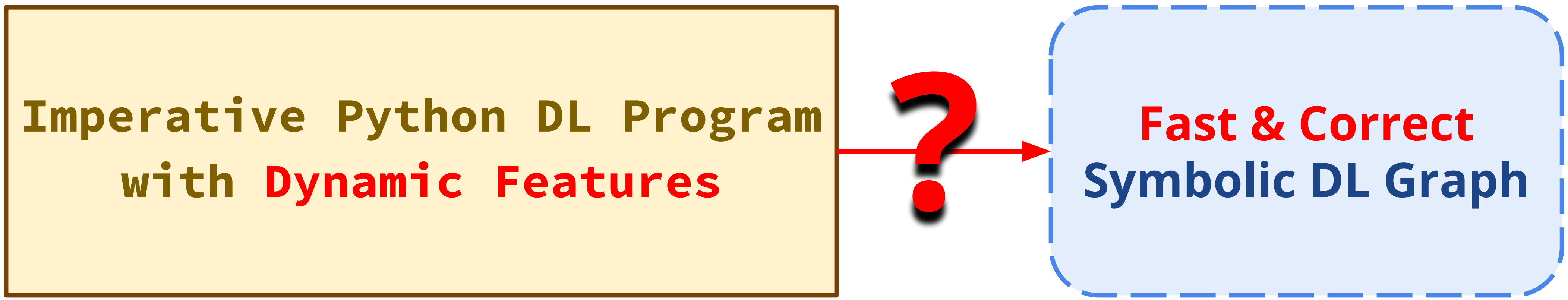
```
class RNNModel(object):
  def __call__(self, sequence):
    state = self.state
    outputs = []
    for item in sequence:
      state = rnn_cell(state, item)
      outputs += [state]
    self.state = state
    return compute_loss(outputs)

for sequence in sequences:
  optimize(lambda: model(sequence))
```

?

# Challenge Summary

**Challenge:**

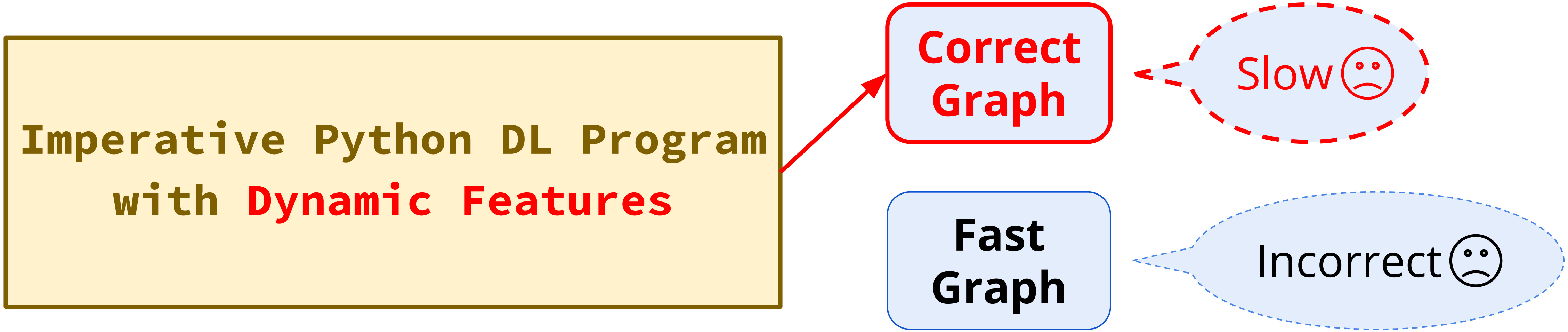
**achieving Correctness & Performance at the same time**



# Challenge Summary

**Challenge:**

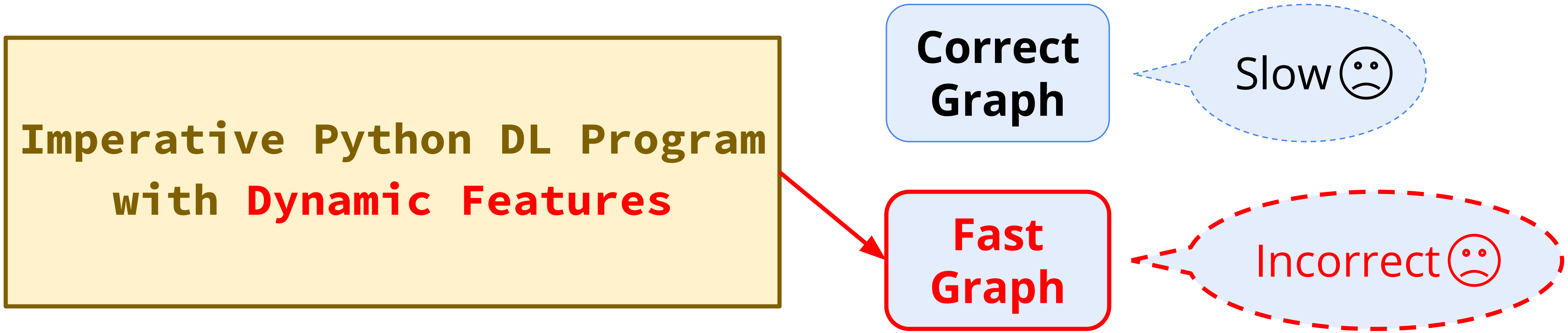
**achieving Correctness & Performance at the same time**



# Challenge Summary

**Challenge:**

**achieving Correctness & Performance at the same time**



# Outline

- **JANUS**
    - Approach
    - Challenges
    - **Our Solution**
    - Evaluation
- How to handle Recursive Neural Networks?
- On-going Works

# Solution: Speculative Graph Generation and Execution

- Goal: Correctness & Performance

- [Performance] **Speculatively Specialize the Graph**
  - Make reasonable assumptions based on the execution history (***Profiling***)
  - Run specialized graph (Common Case)

- [Correctness] **Validate Assumptions**
  - ***Fallback*** if an assumption is broken (Rare Case)

# Overall Workflow on JANUS

**Fast Path**
**(Common Case)**

Correct Path
(Rare Case)

Imperative DL Program

```
for item in sequence:
  state = rnn(state, item)
  outputs += [state]
```

Imperative Executor

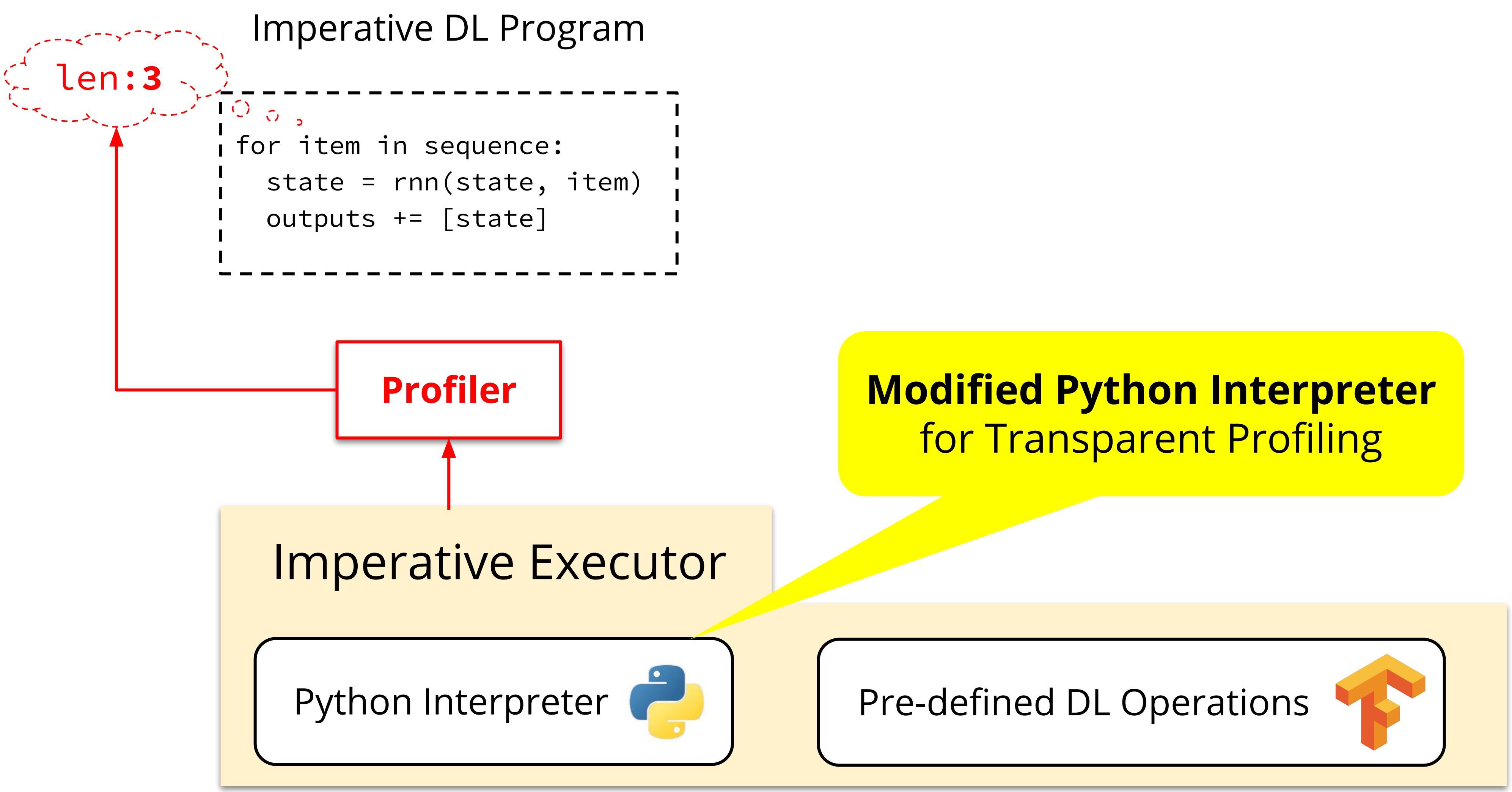
Python Interpreter

Pre-defined DL Operations

# Overall Workflow on JANUS

**Fast Path**
**(Common Case)**

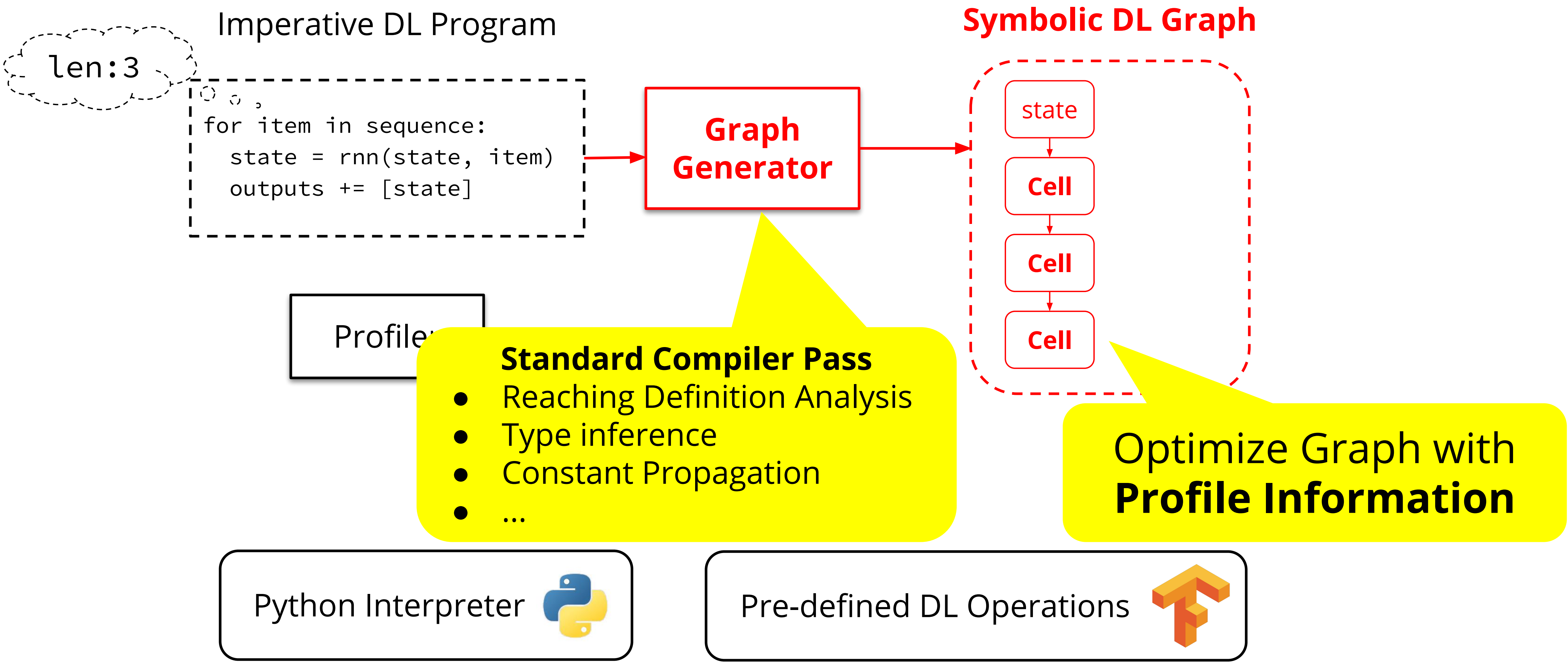
Correct Path
(Rare Case)



# Overall Workflow on JANUS

**Fast Path**
**(Common Case)**

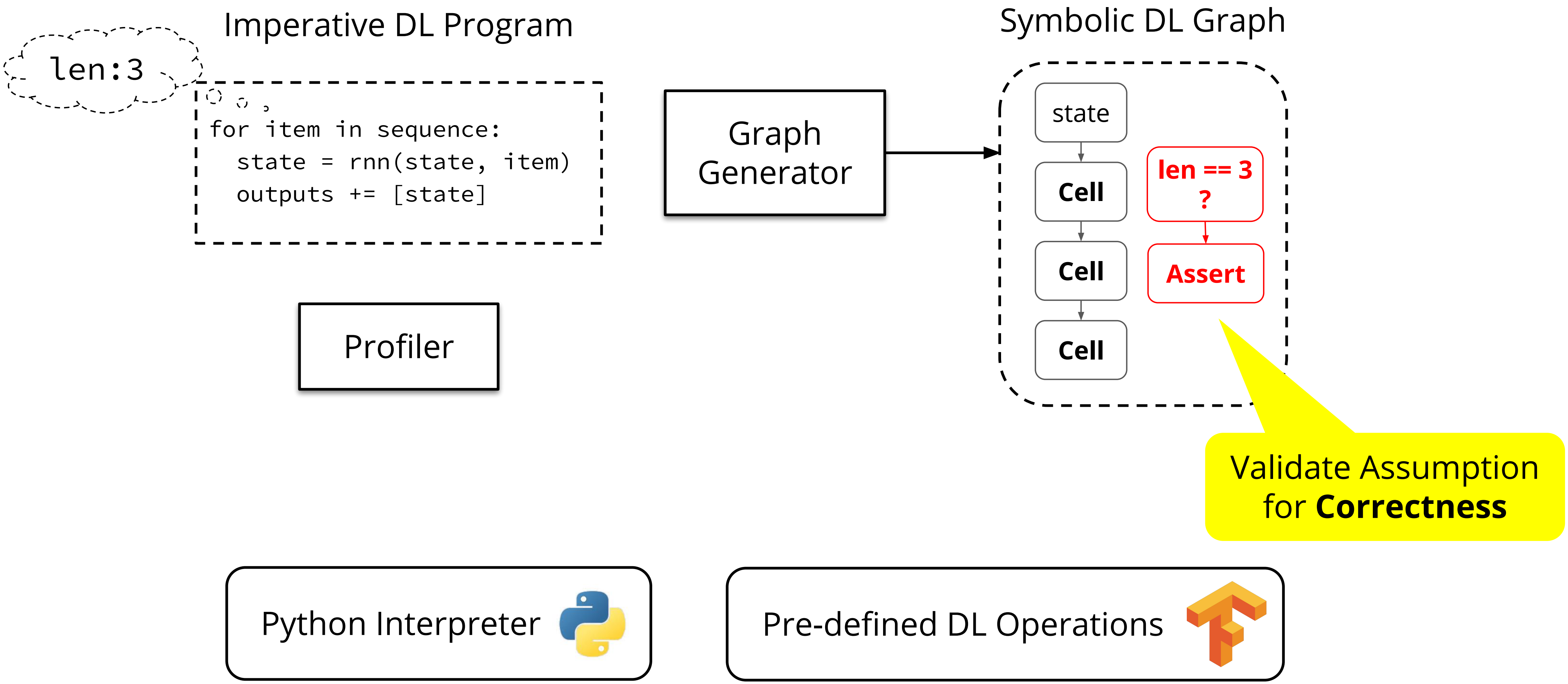
Correct Path
(Rare Case)



# Overall Workflow on JANUS

**Fast Path**
**(Common Case)**

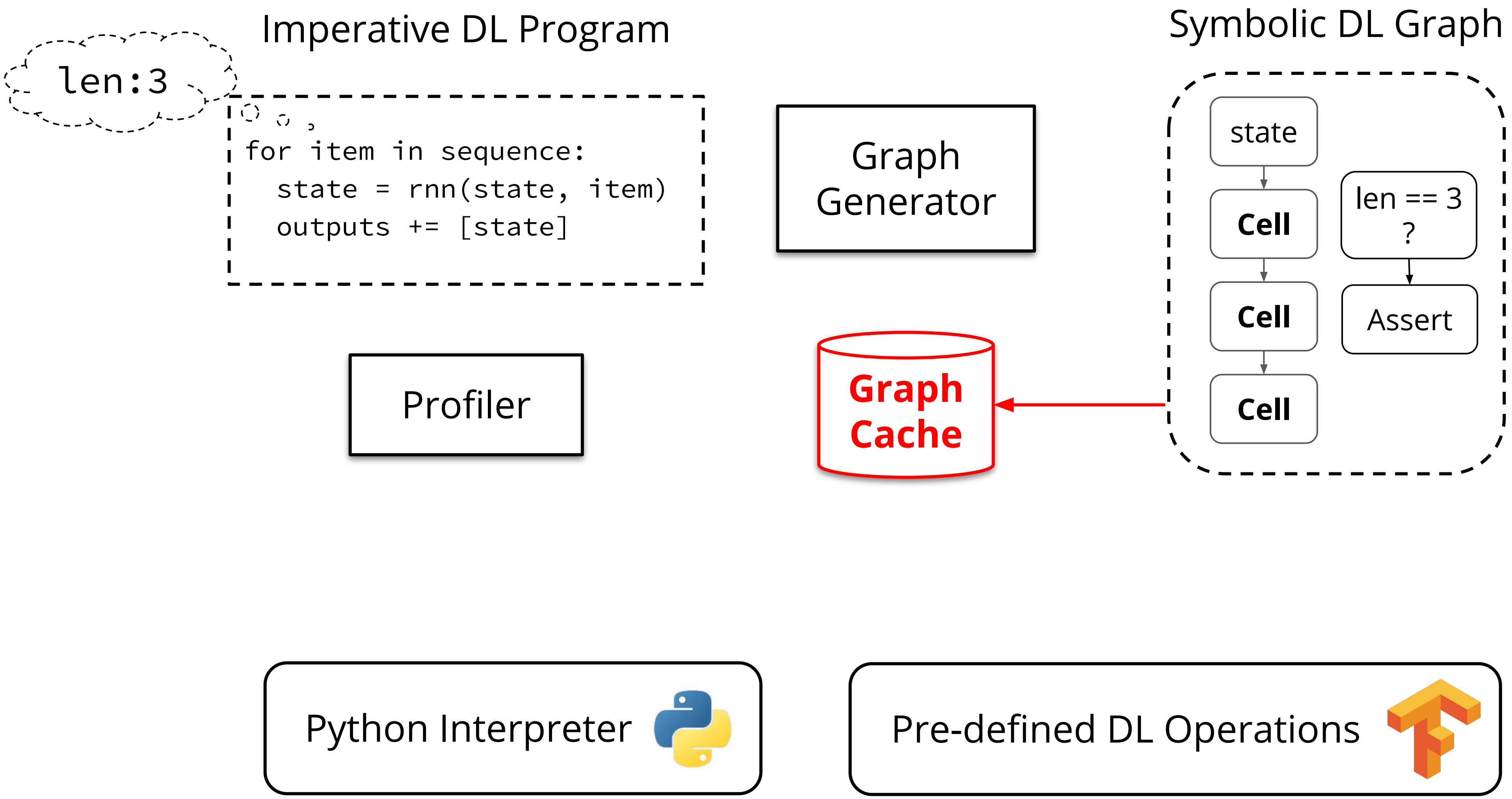
Correct Path
(Rare Case)



# Overall Workflow on JANUS

**Fast Path**
**(Common Case)**

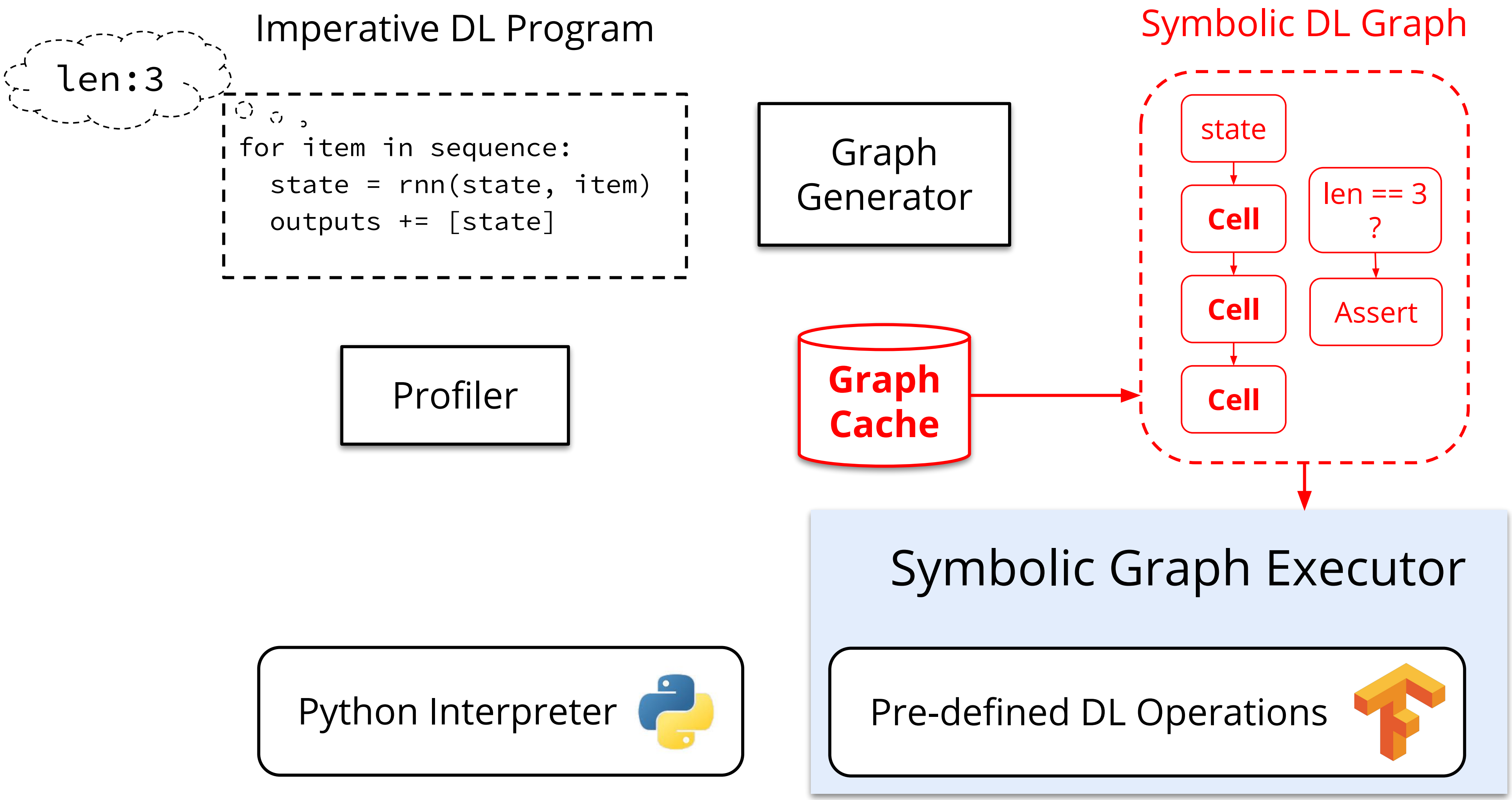
Correct Path
(Rare Case)



# Overall Workflow on JANUS

**Fast Path (Common Case)**

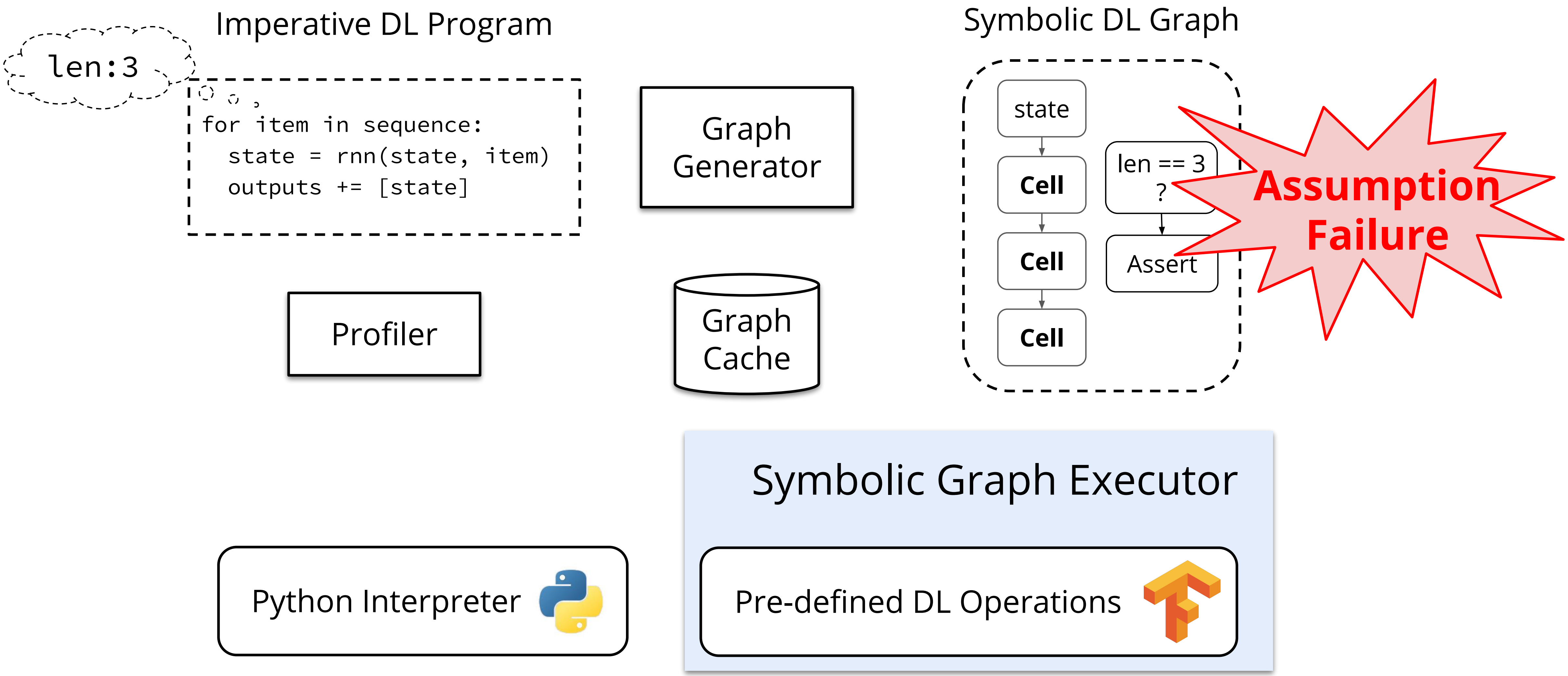
Correct Path (Rare Case)



# Overall Workflow on JANUS

Fast Path
(Common Case)

**Correct Path**
**(Rare Case)**



# Overall Workflow on JANUS

Fast Path
(Common Case)

**Correct Path**
**(Rare Case)**

Imperative DL Program

len:3

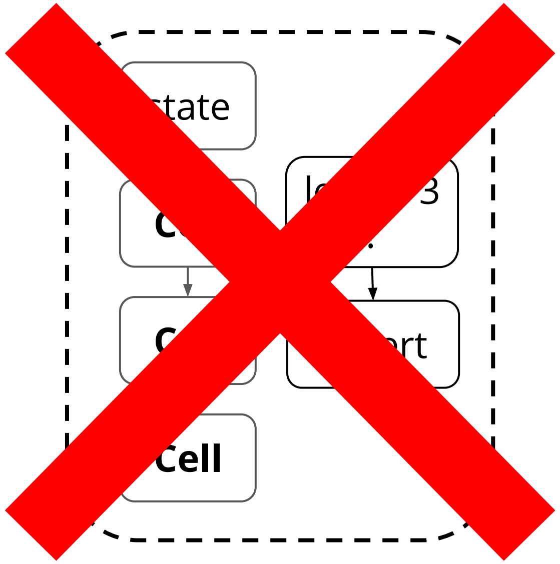
```
for item in sequence:
  state = rnn(state, item)
  outputs += [state]
```

Graph Generator

Profiler

Graph Cache

Symbolic DL Graph




Symbolic Graph Executor

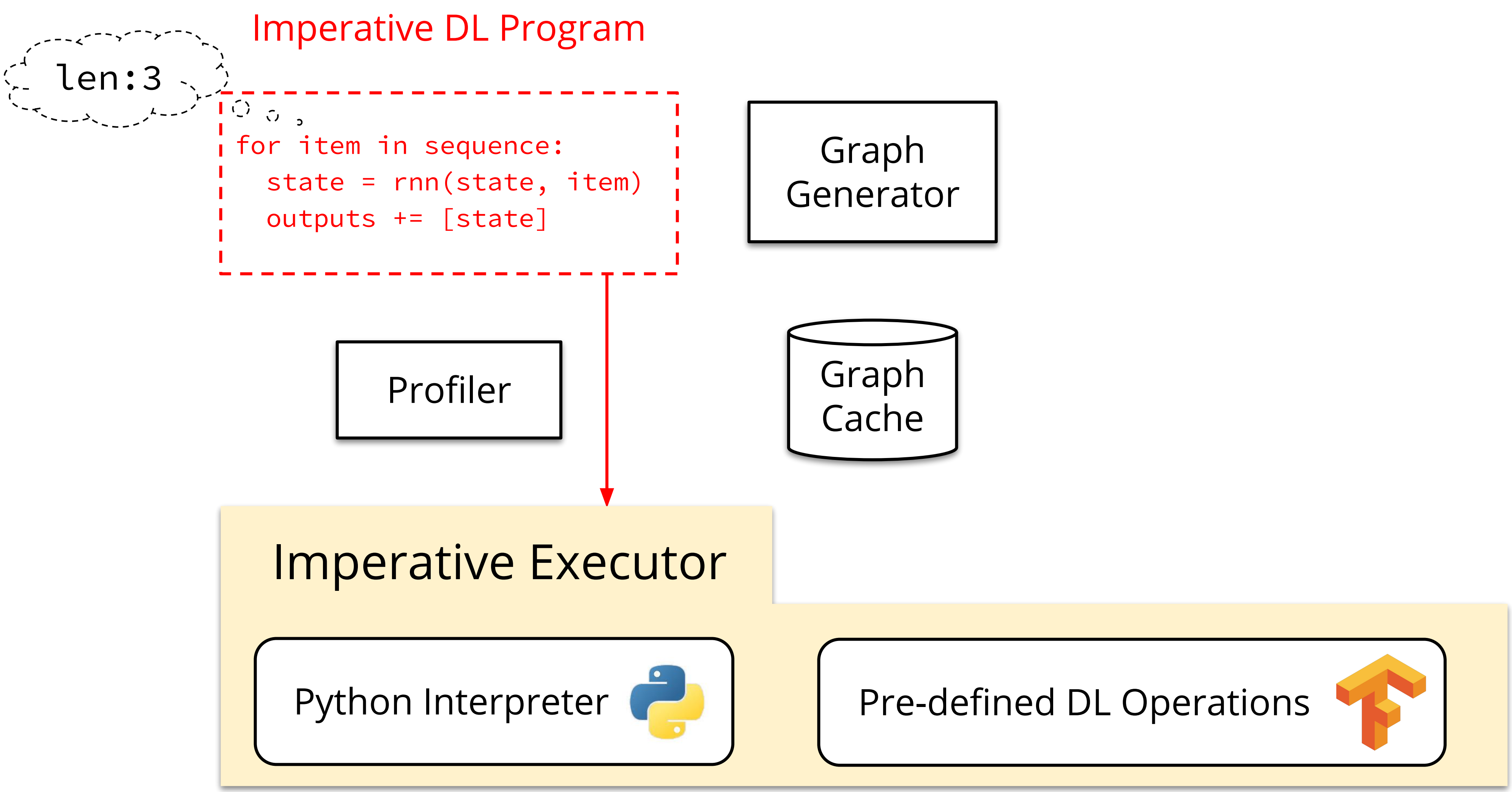
Pre-defined DL Operations

Python Interpreter

# Overall Workflow on JANUS

Fast Path
(Common Case)

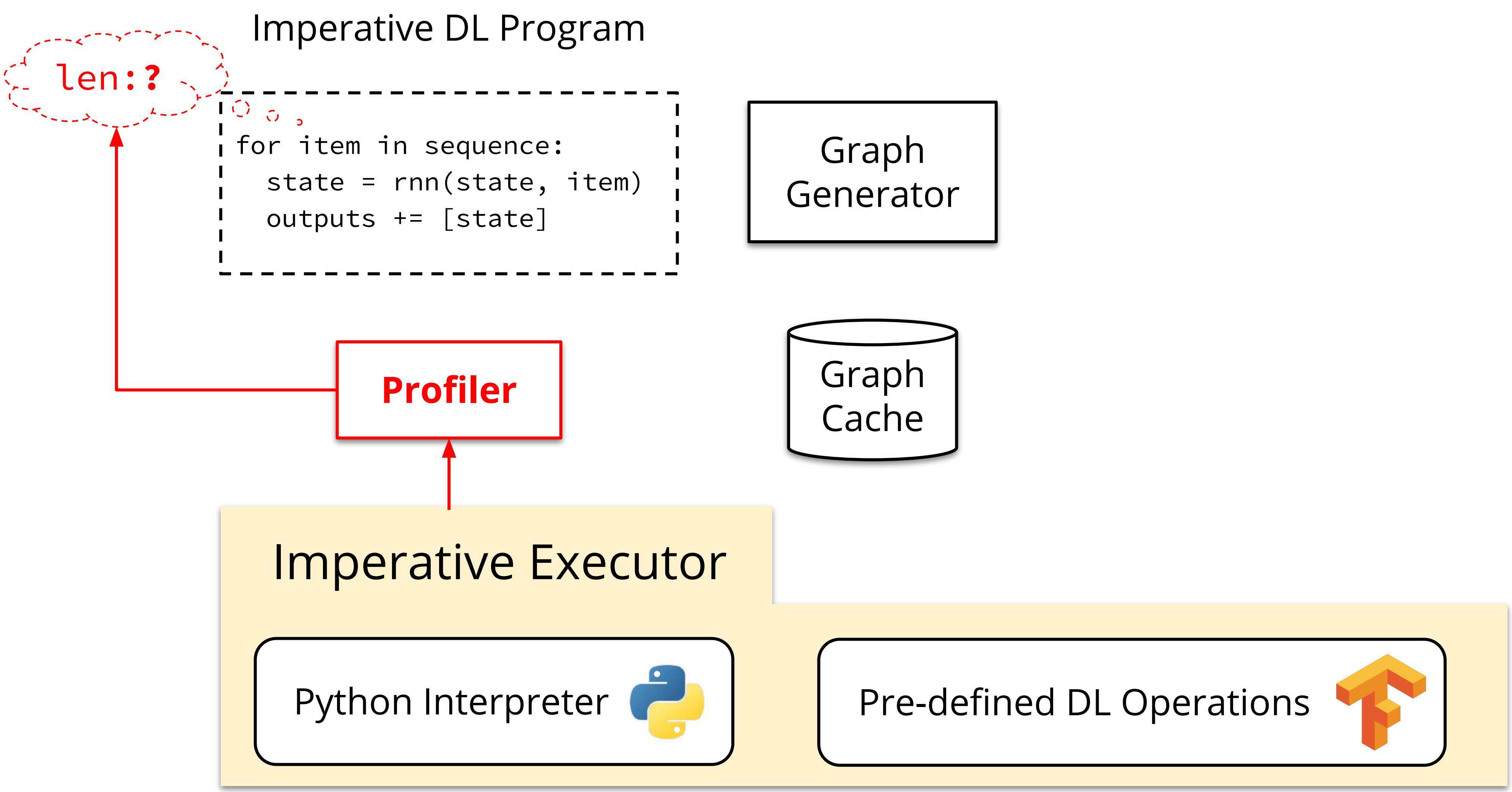
**Correct Path**
**(Rare Case)**



# Overall Workflow on JANUS

Fast Path (Common Case)

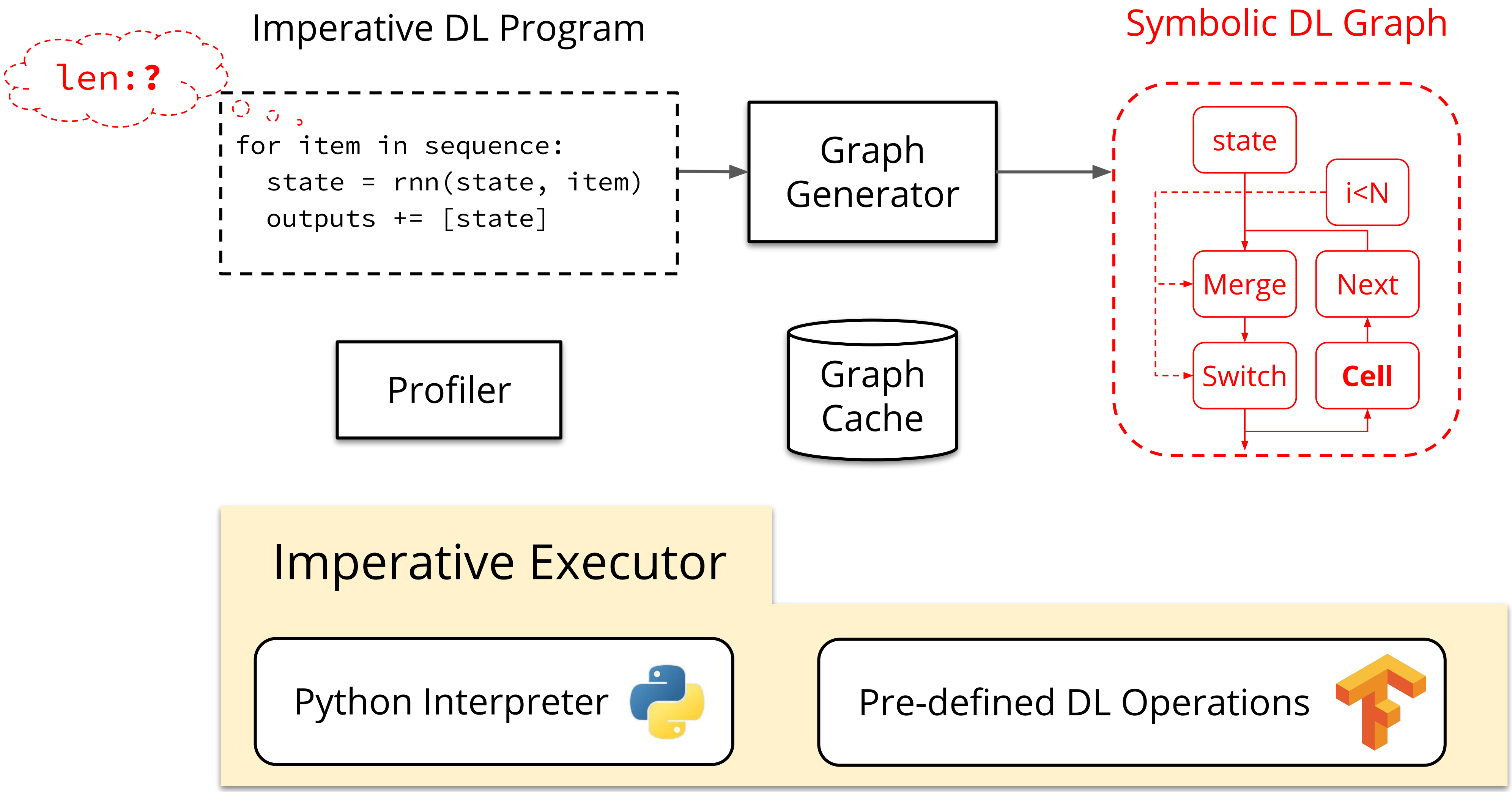
**Correct Path (Rare Case)**



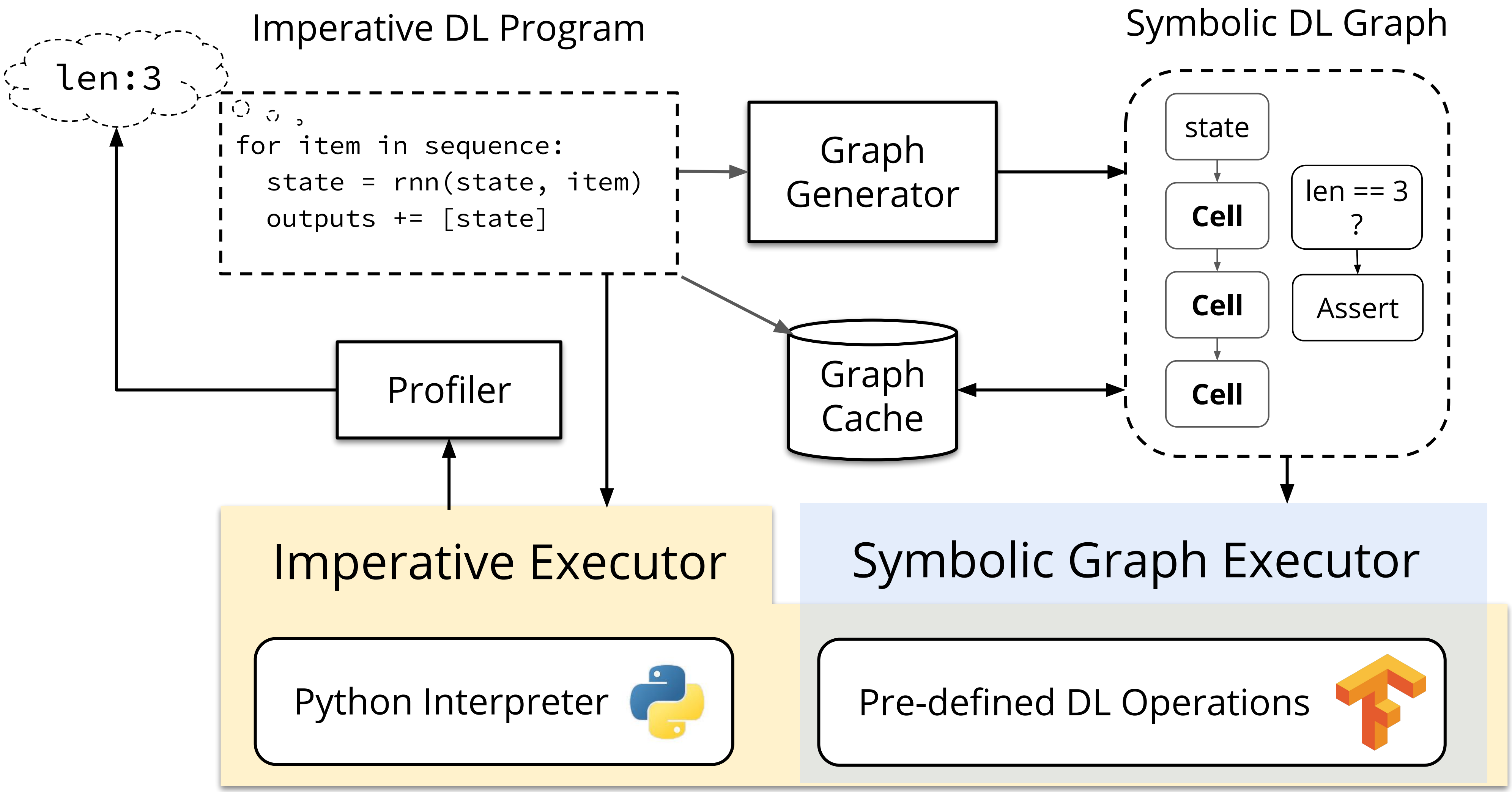
# Overall Workflow on JANUS

Fast Path
(Common Case)

**Correct Path**
**(Rare Case)**



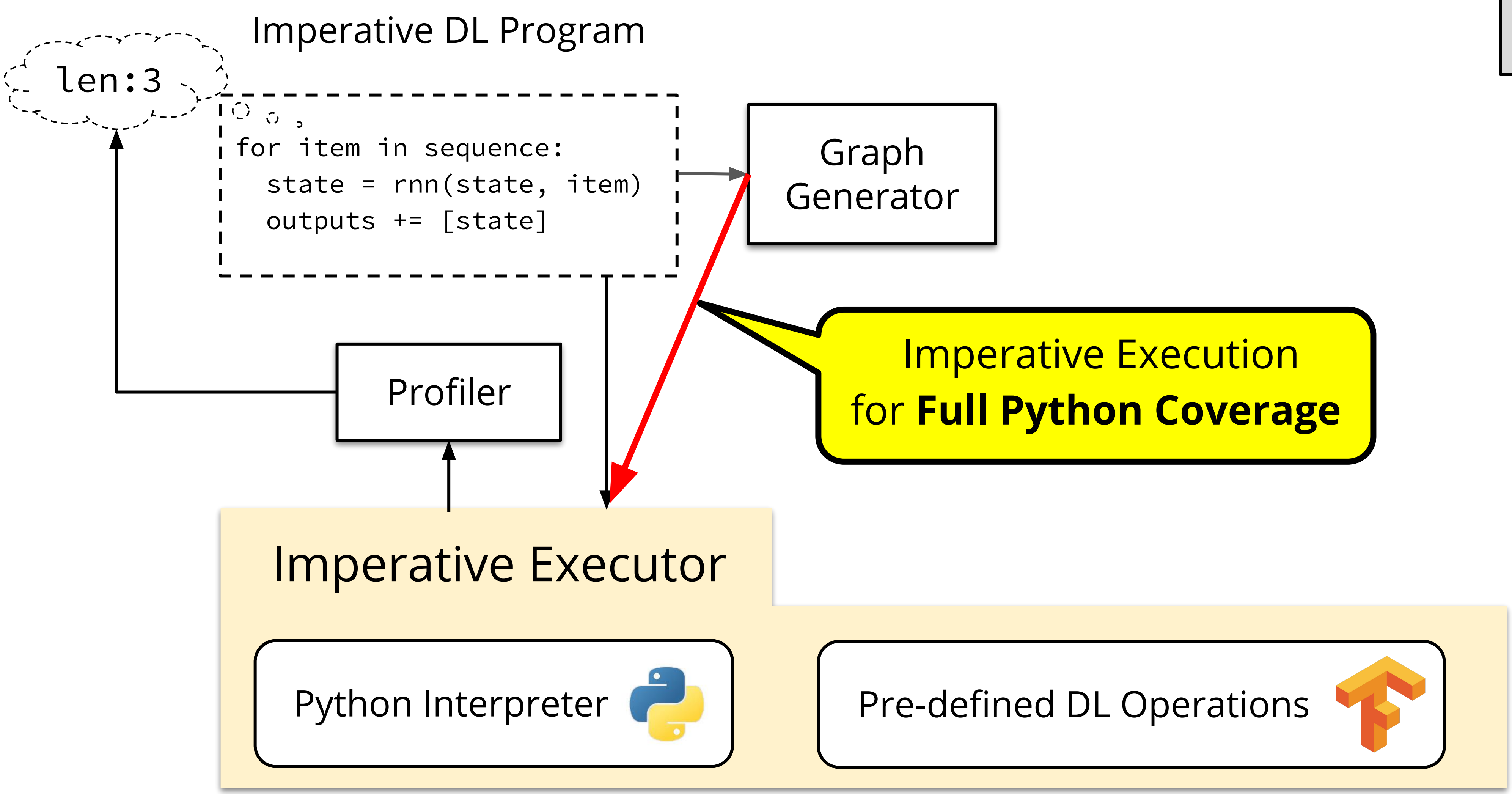
# Overall Workflow on JANUS



# Additional System Aspects

Python Coverage | Global State Consistency

See our paper for more details!



# Additional System Aspects

Python Coverage

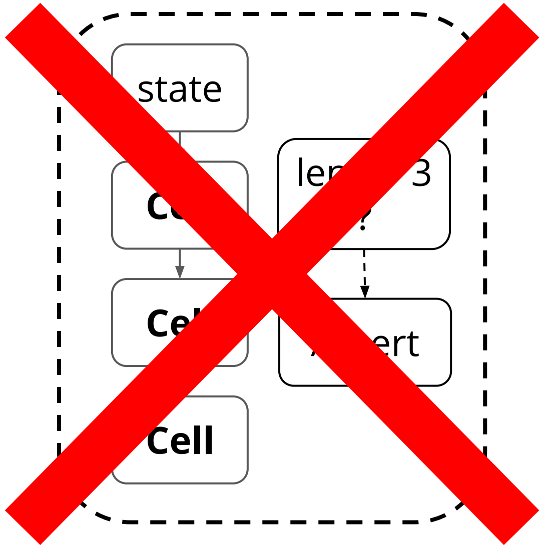
**Global State Consistency**

**Imperative DL Program**

```
for item in sequence:
  state = rnn(state, item)
  outputs += [state]
```

**Fallback**

Symbolic DL Graph




Imperative Executor

Python Interpreter

Symbolic Graph Executor

Pre-defined DL Operations

# Additional System Aspects

Python Coverage

**Global State Consistency**

No Problem

***"Pure"***

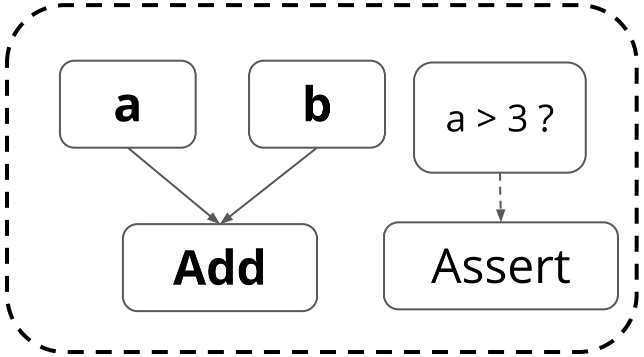
**Imperative DL Program**

```
def foo(a, b):
  if a > 3:
    return a + b
```

**Fallback**

***"Pure"***

**Symbolic DL Graph**




Imperative Executor

Python Interpreter

Symbolic Graph Executor

Pre-defined DL Operations

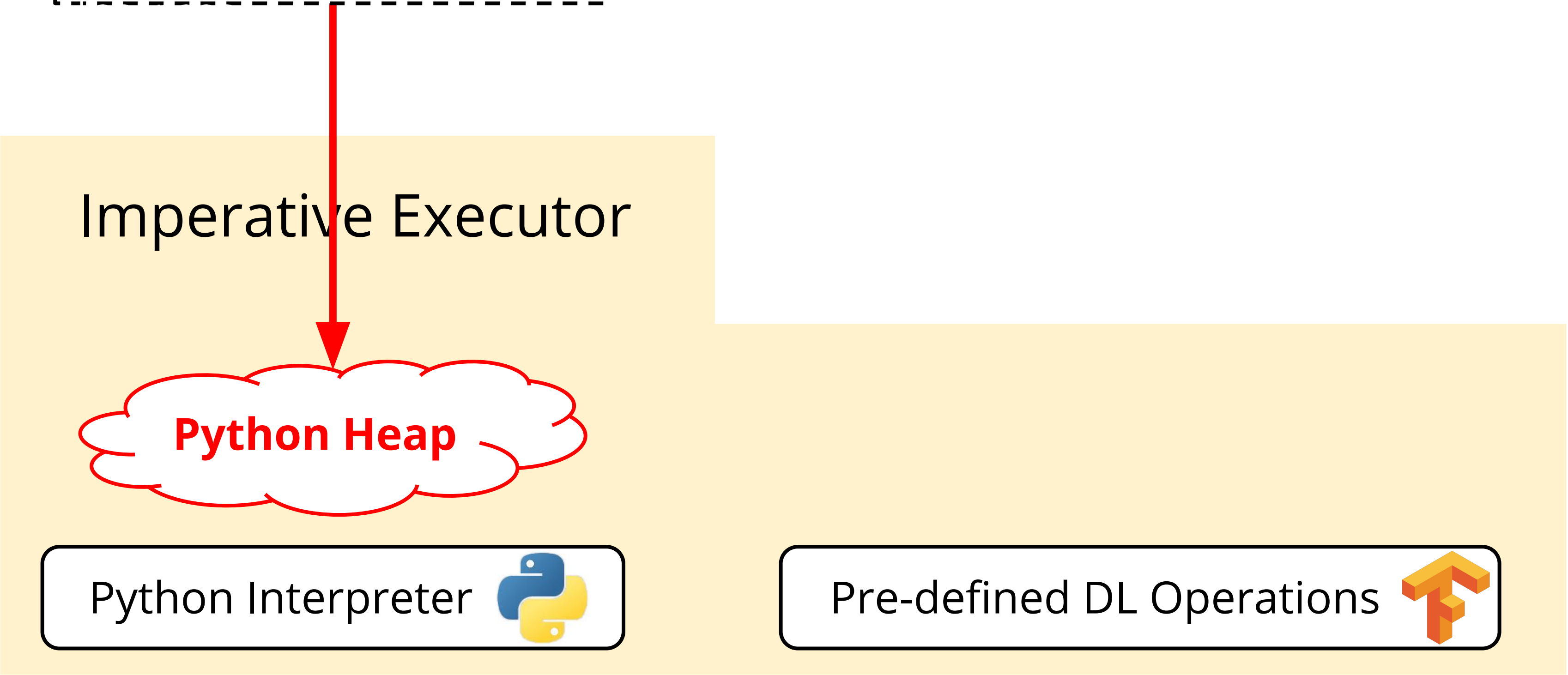
# Additional System Aspects

Python Coverage | **Global State Consistency**

***"Impure"***

**Imperative DL Program**

```
def foo(obj):
 obj.data = value
```



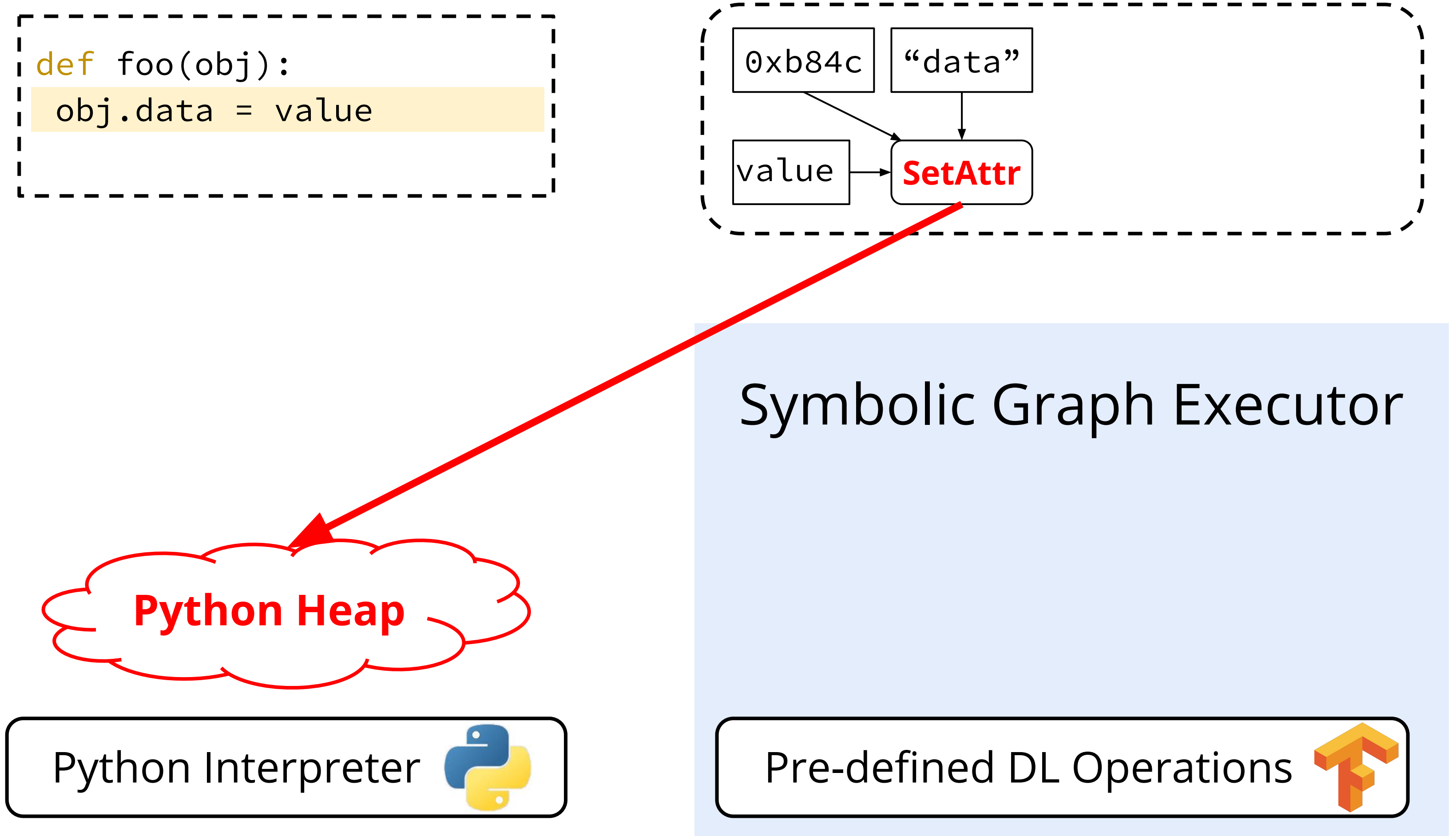
# Additional System Aspects

**Global State Consistency**

*"Impure"*
**Imperative DL Program**

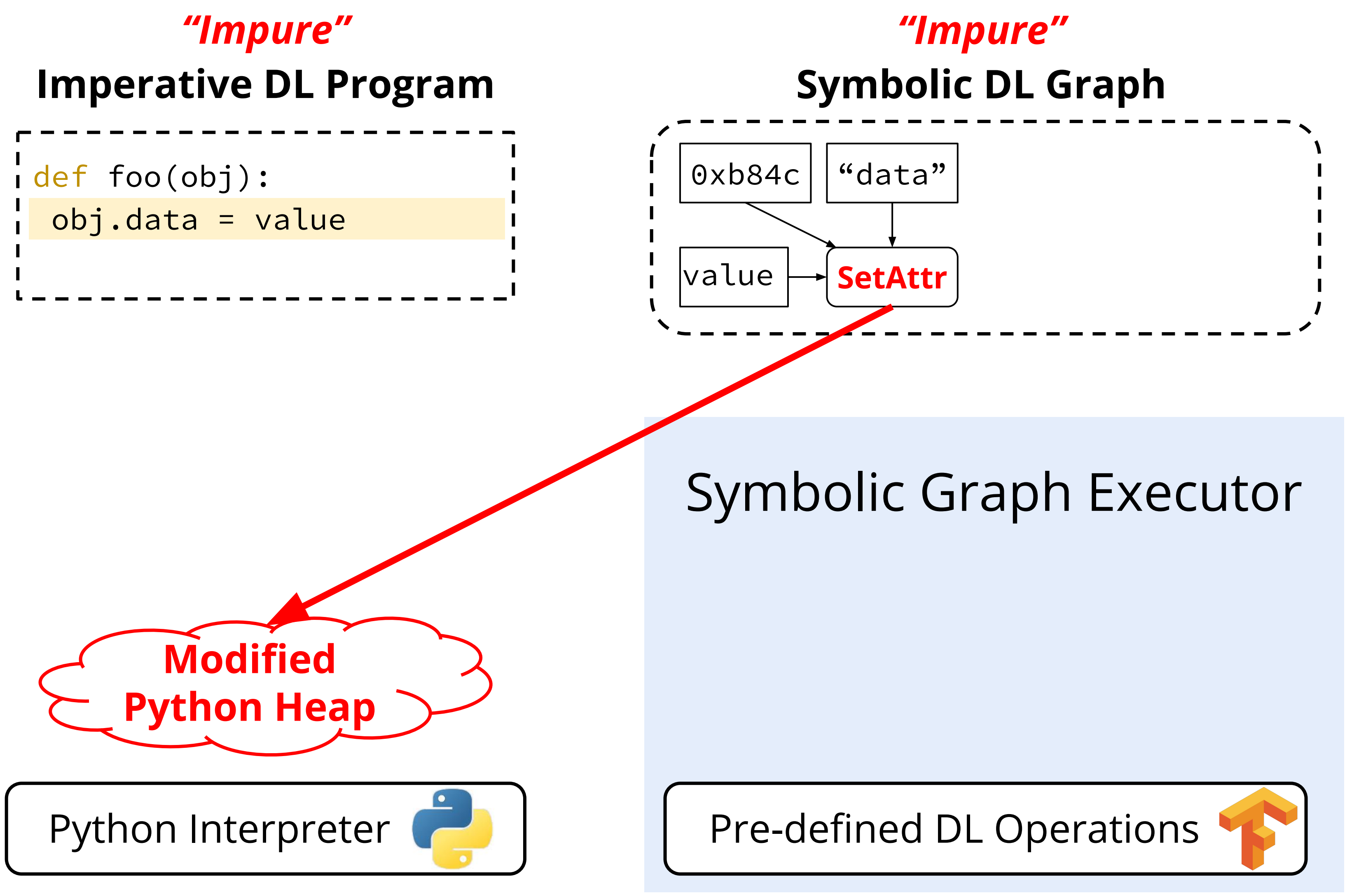
```
def foo(obj):
  obj.data = value
```

*"Impure"*
**Symbolic DL Graph**



# Additional System Aspects

Python Coverage | **Global State Consistency**

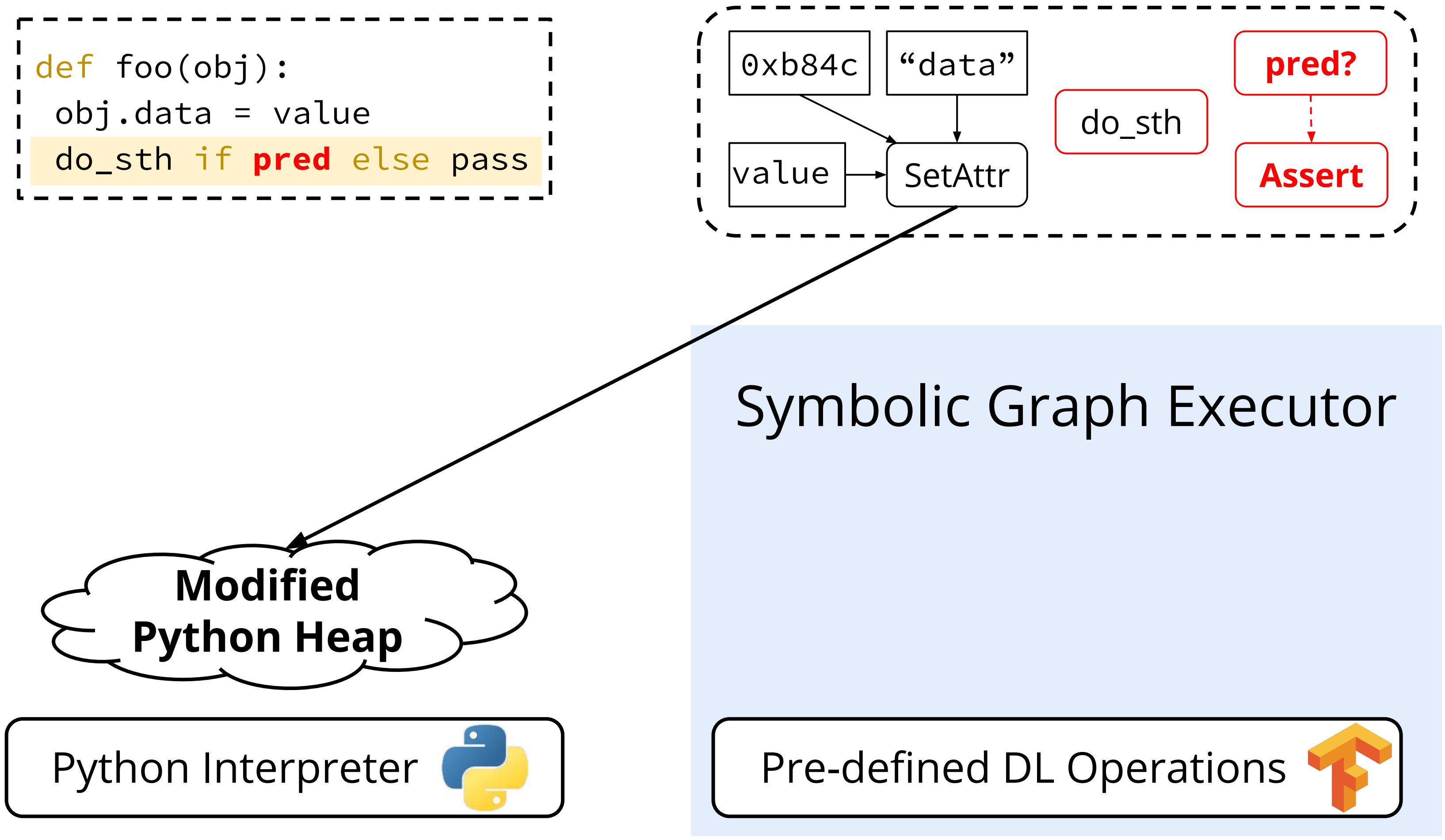


# Additional System Aspects

*"Impure"*
Imperative DL Program

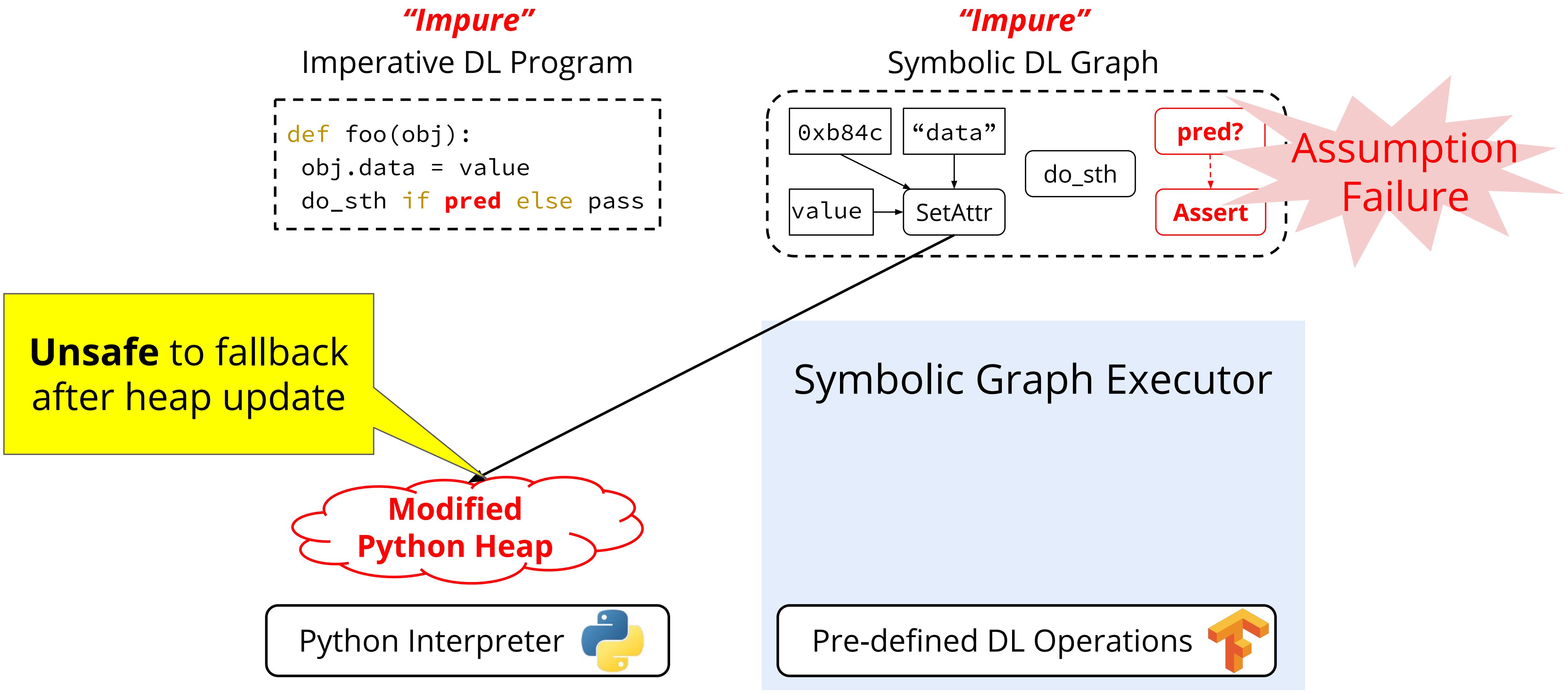
*"Impure"*
Symbolic DL Graph

```
def foo(obj):
  obj.data = value
  do_sth if pred else pass
```



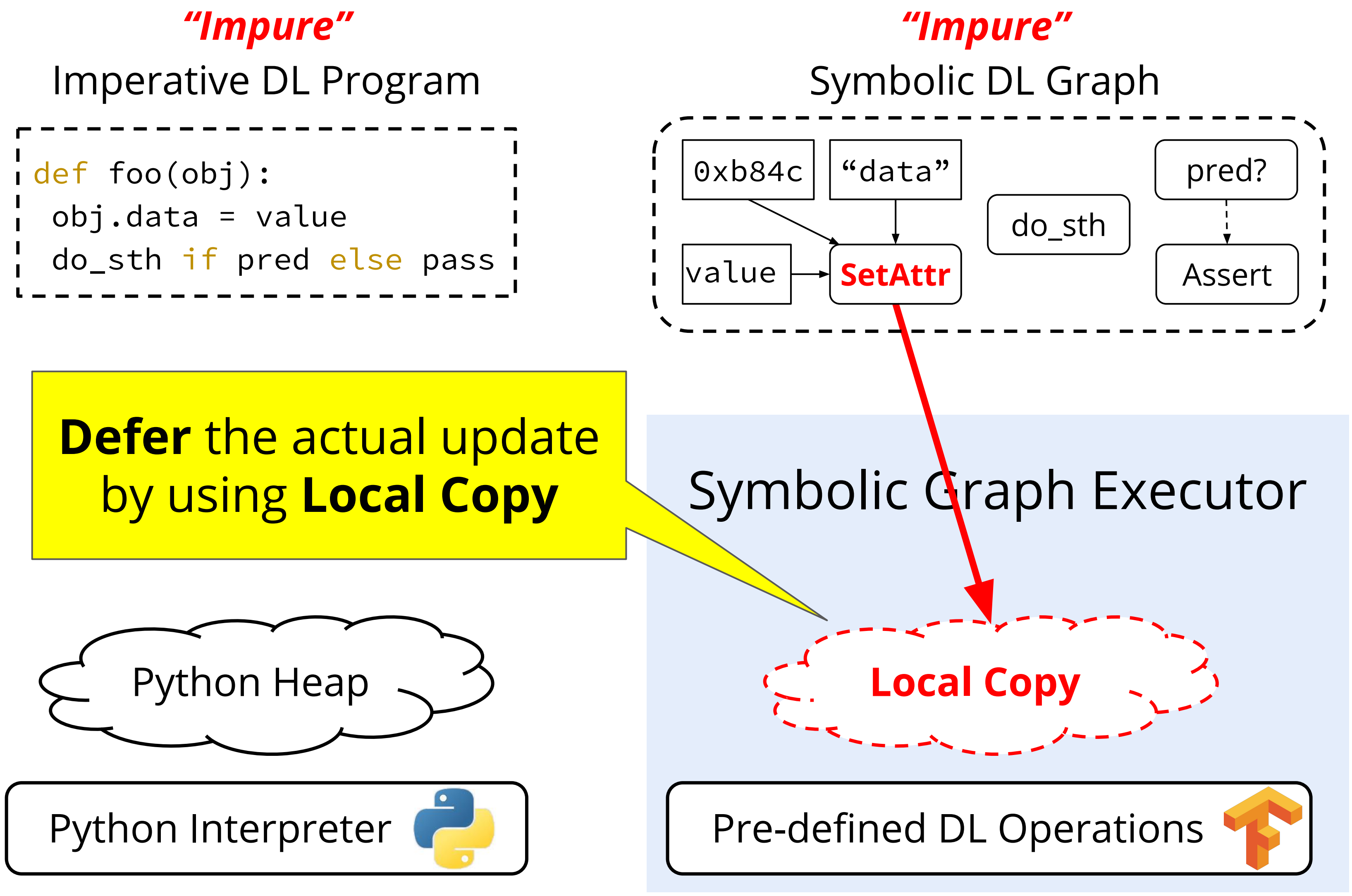
# Additional System Aspects

Python Coverage | **Global State Consistency**



# Additional System Aspects





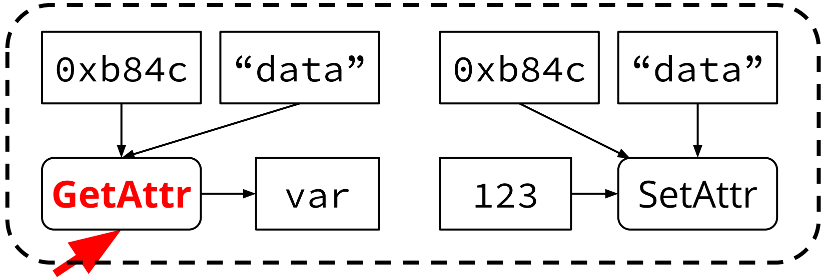
# Additional System Aspects

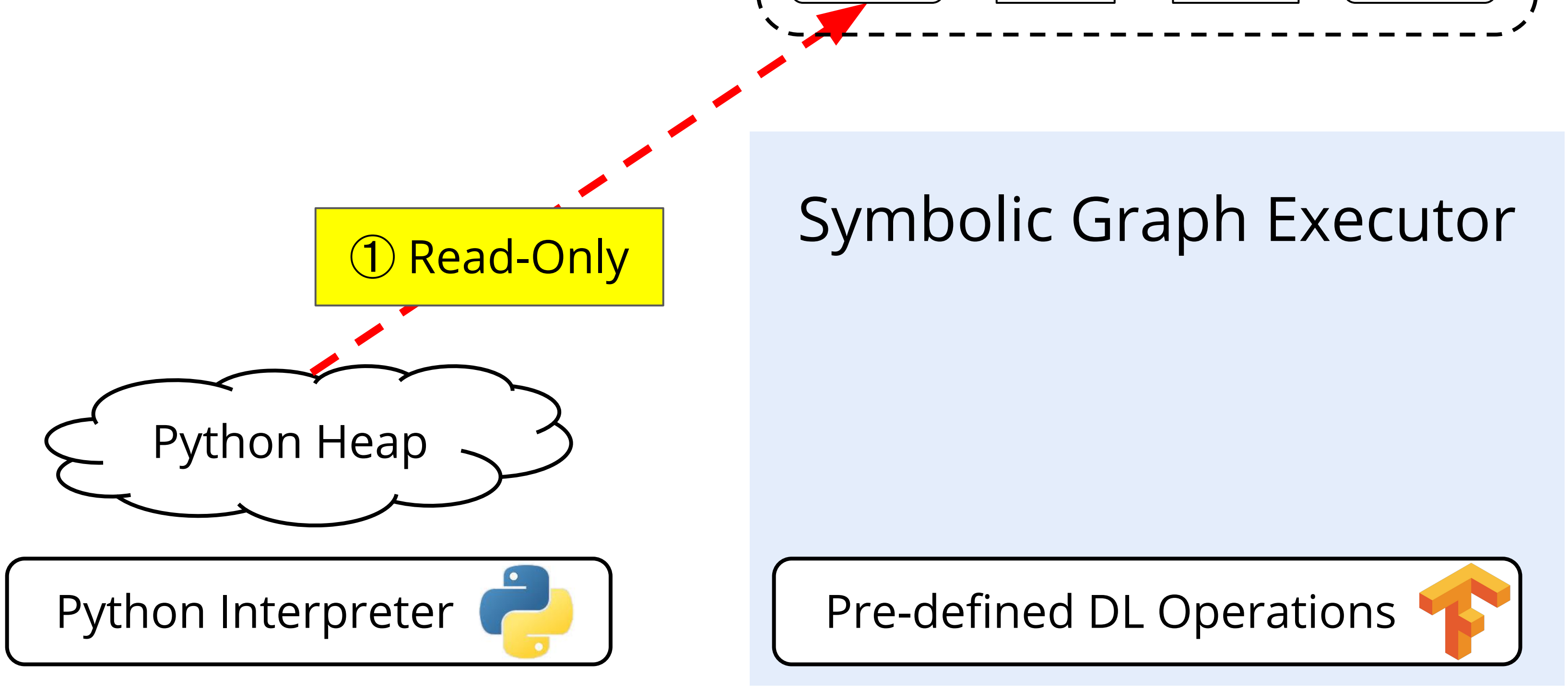
Python Coverage | **Global State Consistency**

***"Impure"***
Imperative DL Program

```python
def foo(obj):
  var = obj.data
  obj.data = 123
```

***"Impure"***
Symbolic DL Graph



# Additional System Aspects

Python Coverage

**Global State Consistency**

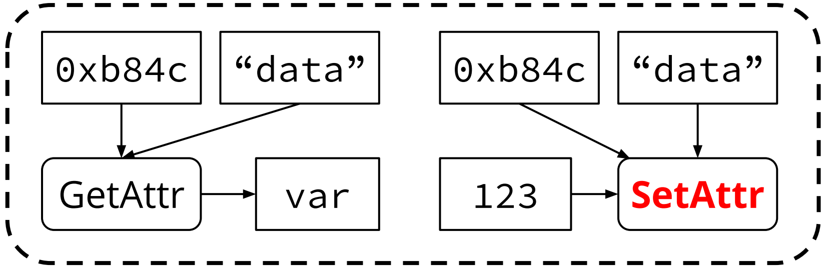
***"Impure"***

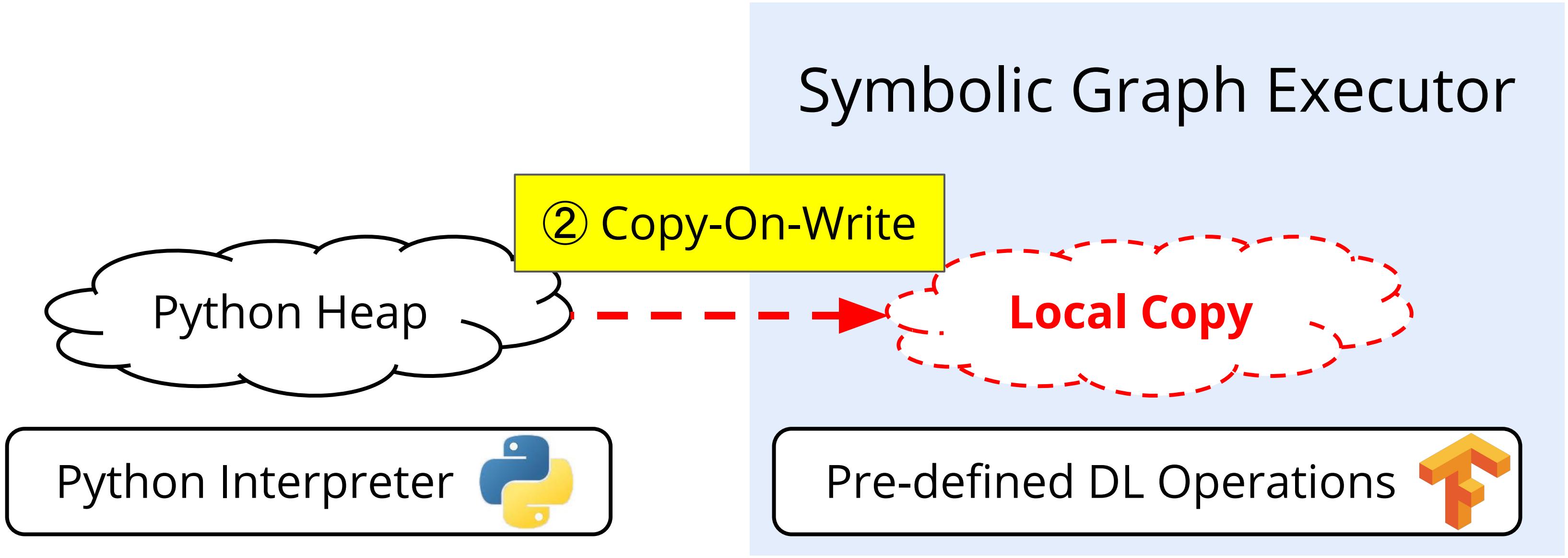
Imperative DL Program

```
def foo(obj):
  var = obj.data
  obj.data = 123
```

***"Impure"***

Symbolic DL Graph



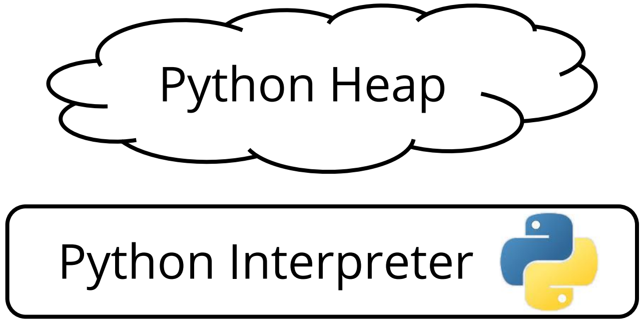



# Additional System Aspects

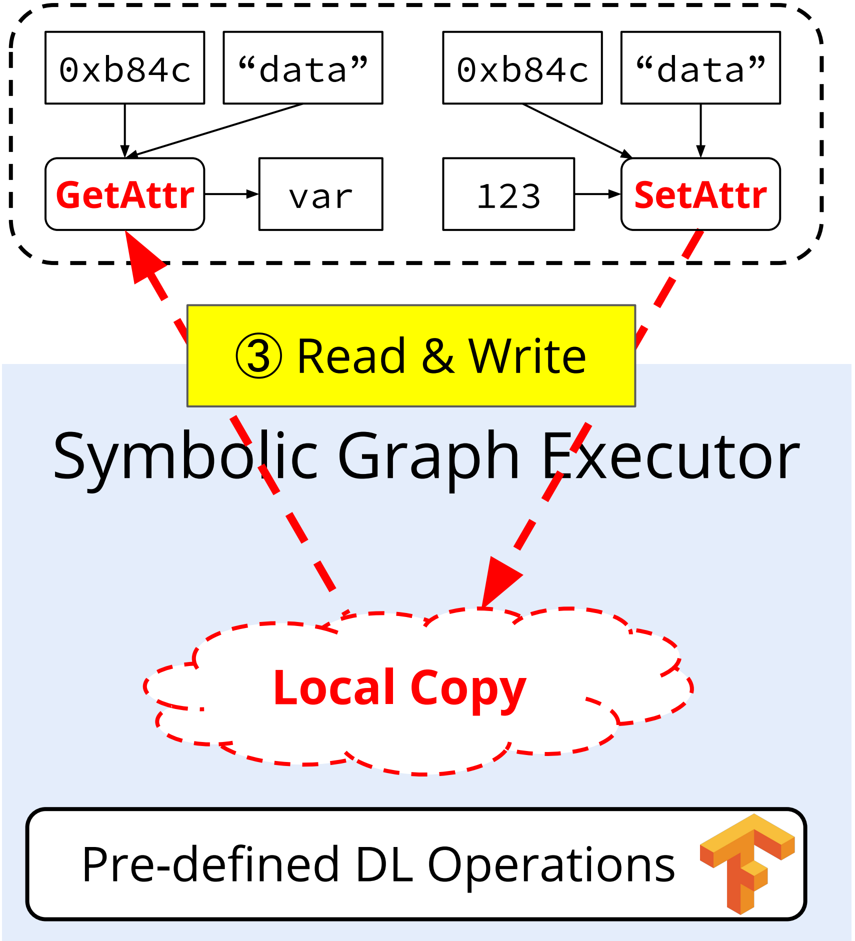
Python Coverage | **Global State Consistency**

***"Impure"***
Imperative DL Program

```
def foo(obj):
  var = obj.data
  obj.data = 123
```




***"Impure"***
Symbolic DL Graph



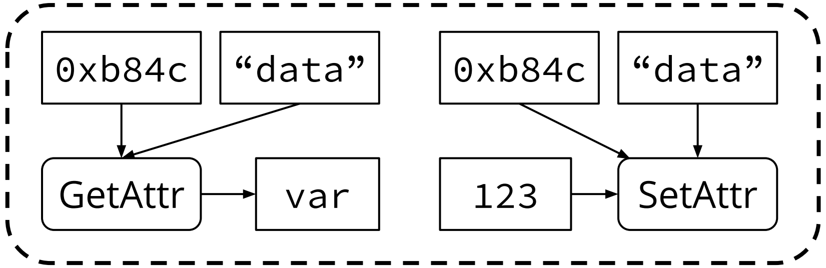
# Additional System Aspects

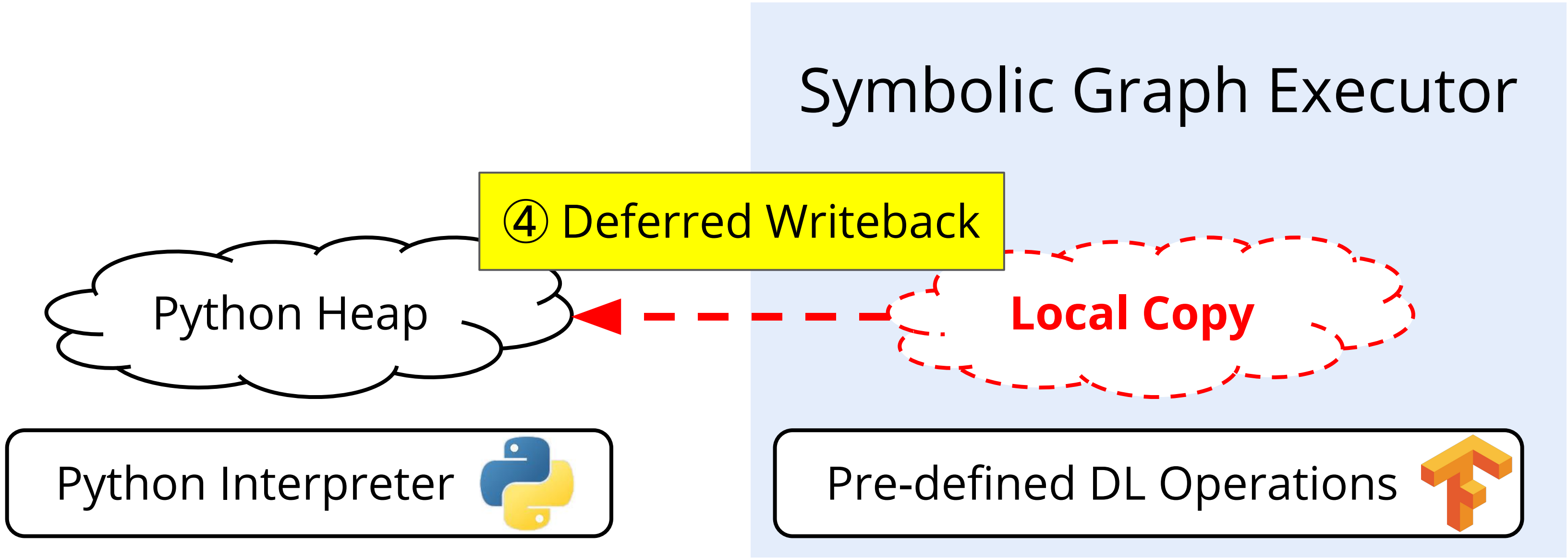
Python Coverage | **Global State Consistency**

***"Impure"***
Imperative DL Program

```
def foo(obj):
  var = obj.data
  obj.data = 123
```

***"Impure"***
Symbolic DL Graph



# Implementation

- **JANUS**: 4700 LoC

- Implemented on top of **TensorFlow** 1.8.0
  - Symbolic Graph Executor: TensorFlow
  - Imperative Executor: TensorFlow Eager
  - Modification: 771 LoC (custom operations, execution model, ...)

- and also on top of **CPython** 3.5.2
  - Modification: 1096 LoC (for transparent, non-intrusive profiling, ...)

# Outline

- **JANUS**
  - Approach
  - Challenges
  - Our Solution
  - **Evaluation**
- How to handle Recursive Neural Networks?
- On-going Works

# Evaluation Setup: Frameworks & Environments

- **Frameworks**
  - JANUS — Implemented on top of TensorFlow
  - Symbolic — TensorFlow
  - Imperative — TensorFlow Eager

- **Hardware & Software Setup**
  - 6 machines connected via Mellanox ConnectX-4 cards w/ 100Gbps InfiniBand
  - Each machine w/ 2x(`Intel Xeon E5-2695`)+6x(`NVIDIA GeForce Titan Xp`)
  - Ubuntu 16.04, TensorFlow 1.8.0, CUDA 9.0
  - Horovod 0.12.1, NCCL v2.1, OpenMPI v3.0.0

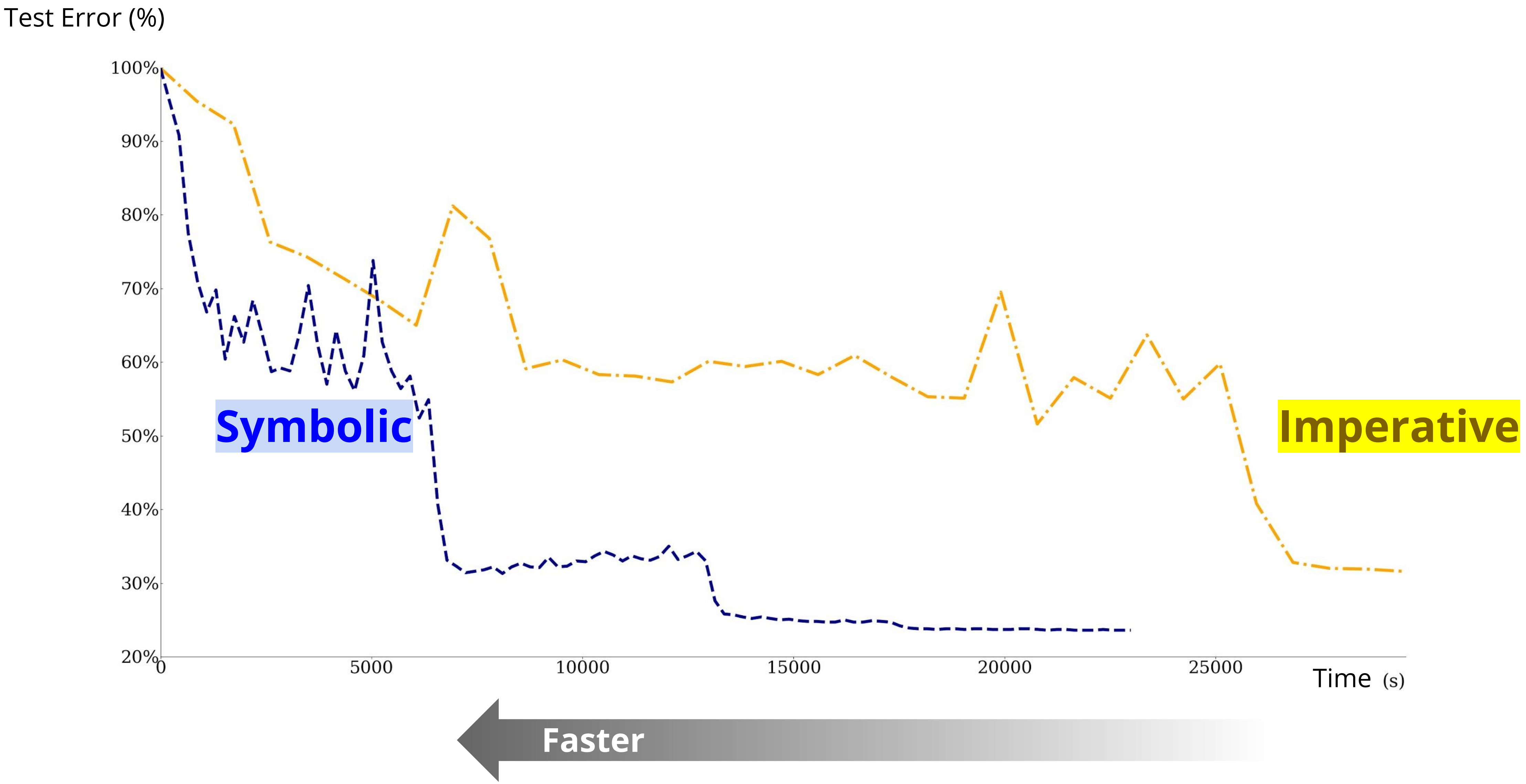
# Evaluation Setup: Applications

11 models in 5 categories using various dynamic characteristics of Python

| | |
|---|---|
| • Convolutional Neural Networks (**CNN**) | LeNet, ResNet-50, Inception-v3 |
| • Recurrent Neural Networks (**RNN**) | LSTM, LM |
| • Recursive Neural Networks (**TreeNN**) | TreeRNN, TreeLSTM |
| • Deep Reinforcement Learning (**DRL**) | A3C, PPO |
| • Generative Adversarial Networks (**GAN**) | AN, PIX2PIX |

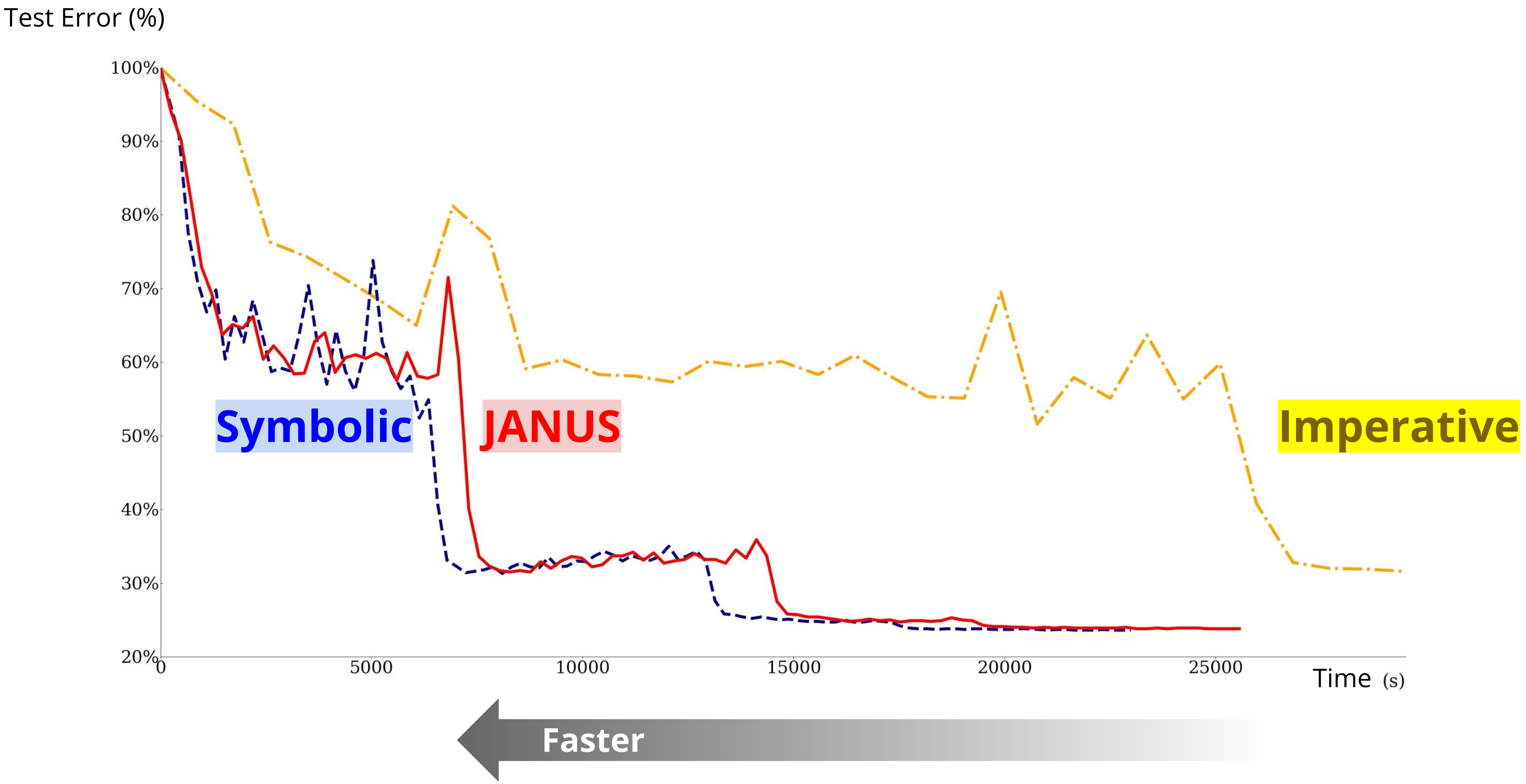
# ImageNet Test Error with ResNet50

36 GPUs



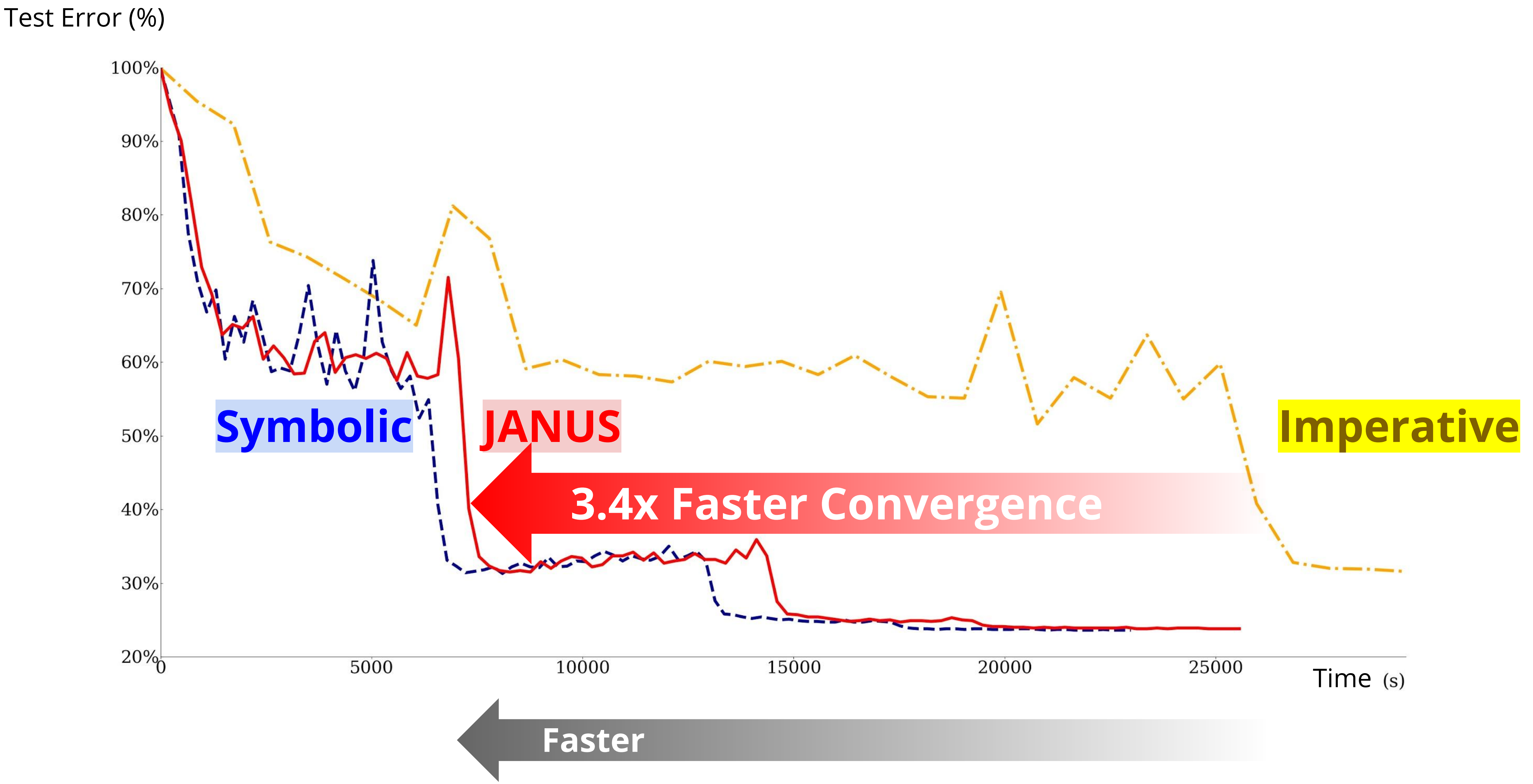
# ImageNet Test Error with ResNet50

36 GPUs



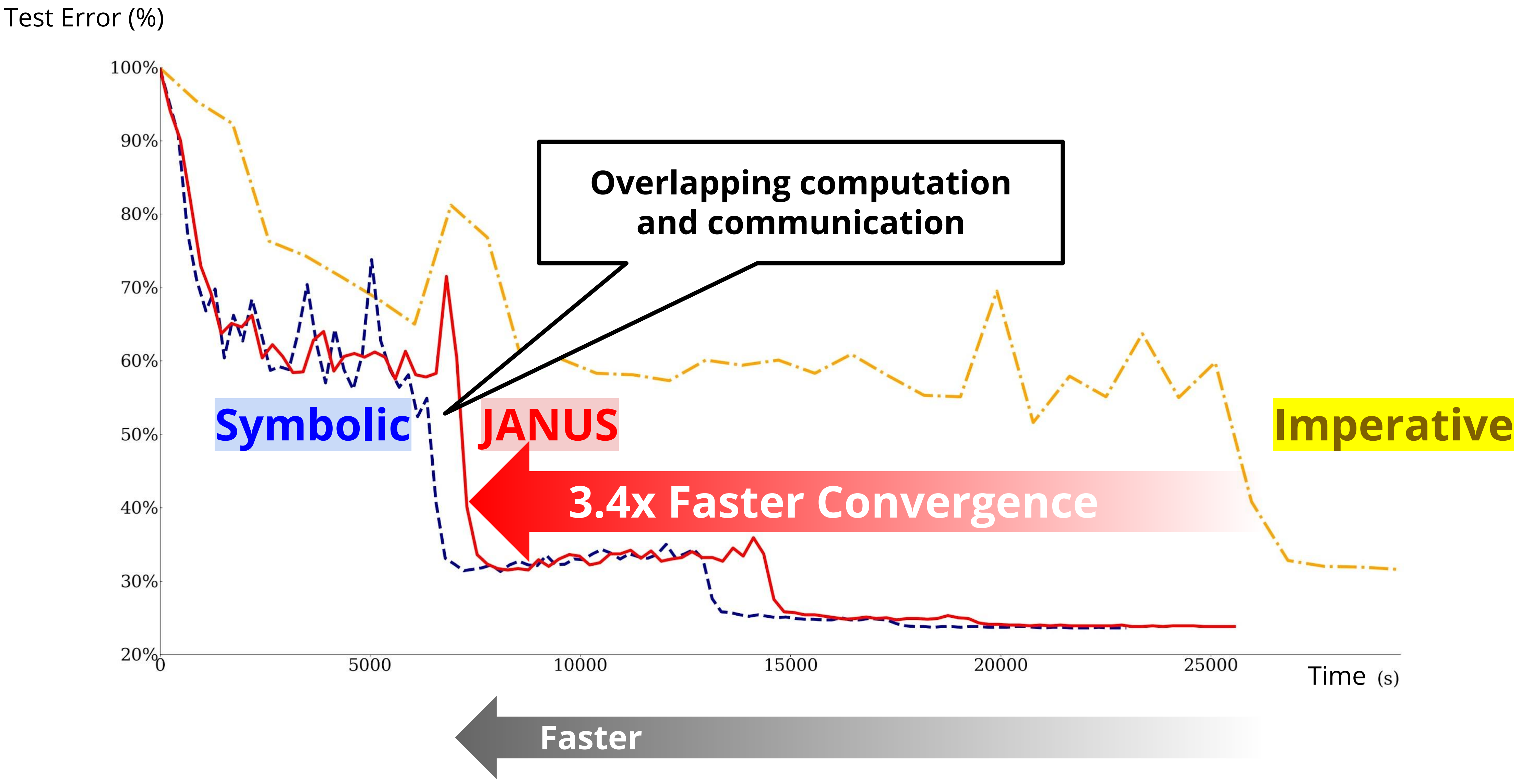
# ImageNet Test Error with ResNet50

36 GPUs

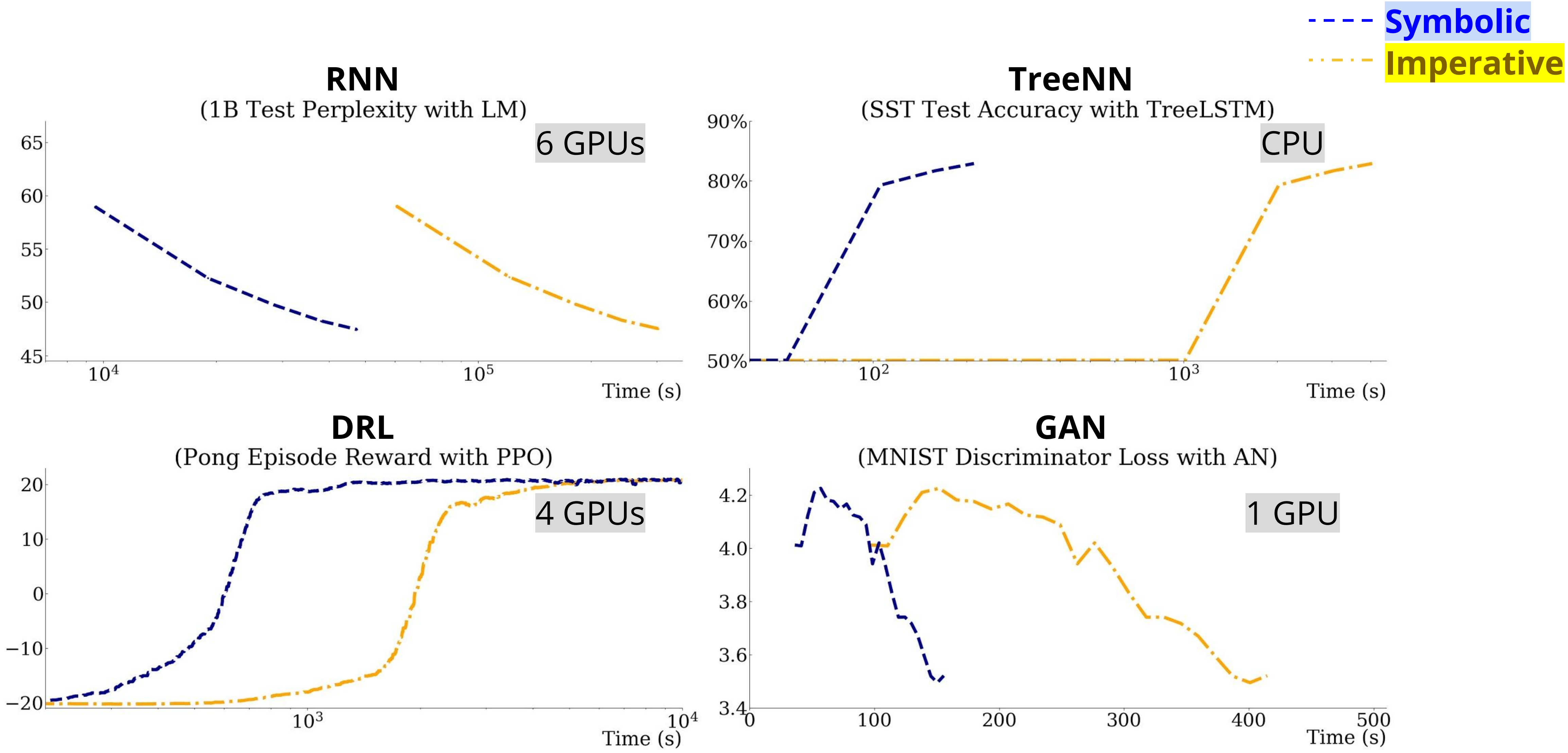


# ImageNet Test Error with ResNet50

36 GPUs

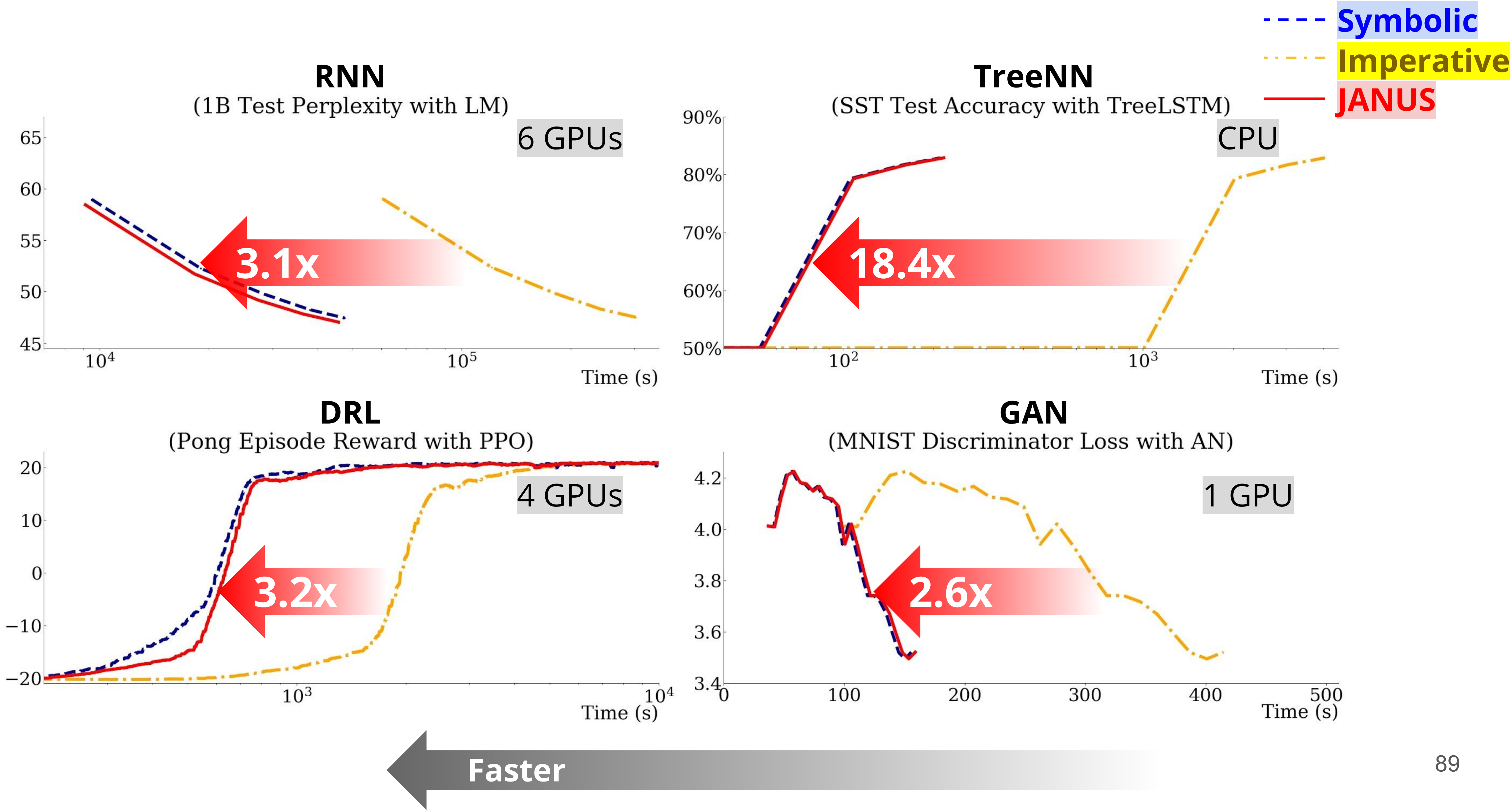


# Model Convergence



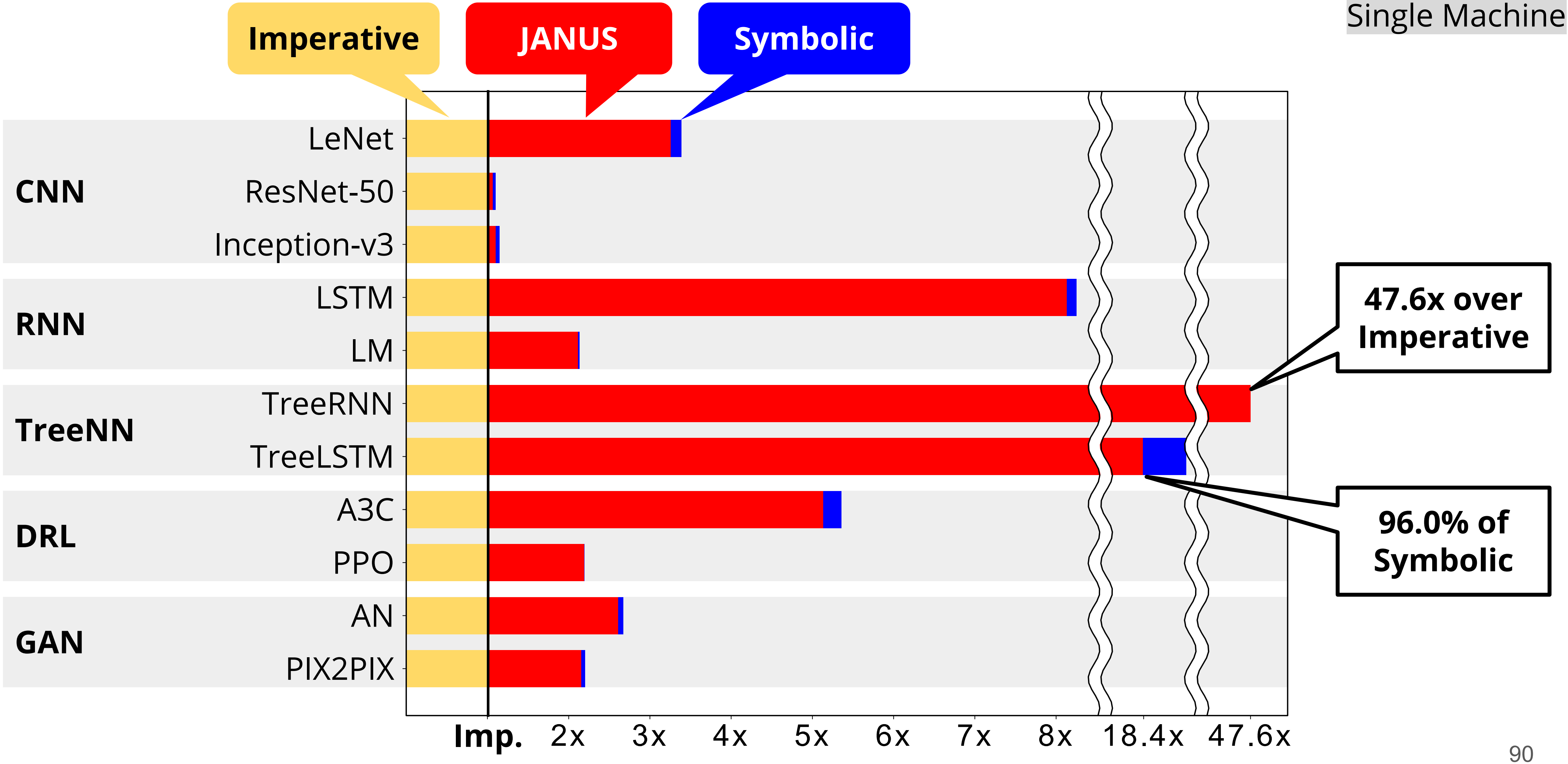

Faster

# Model Convergence

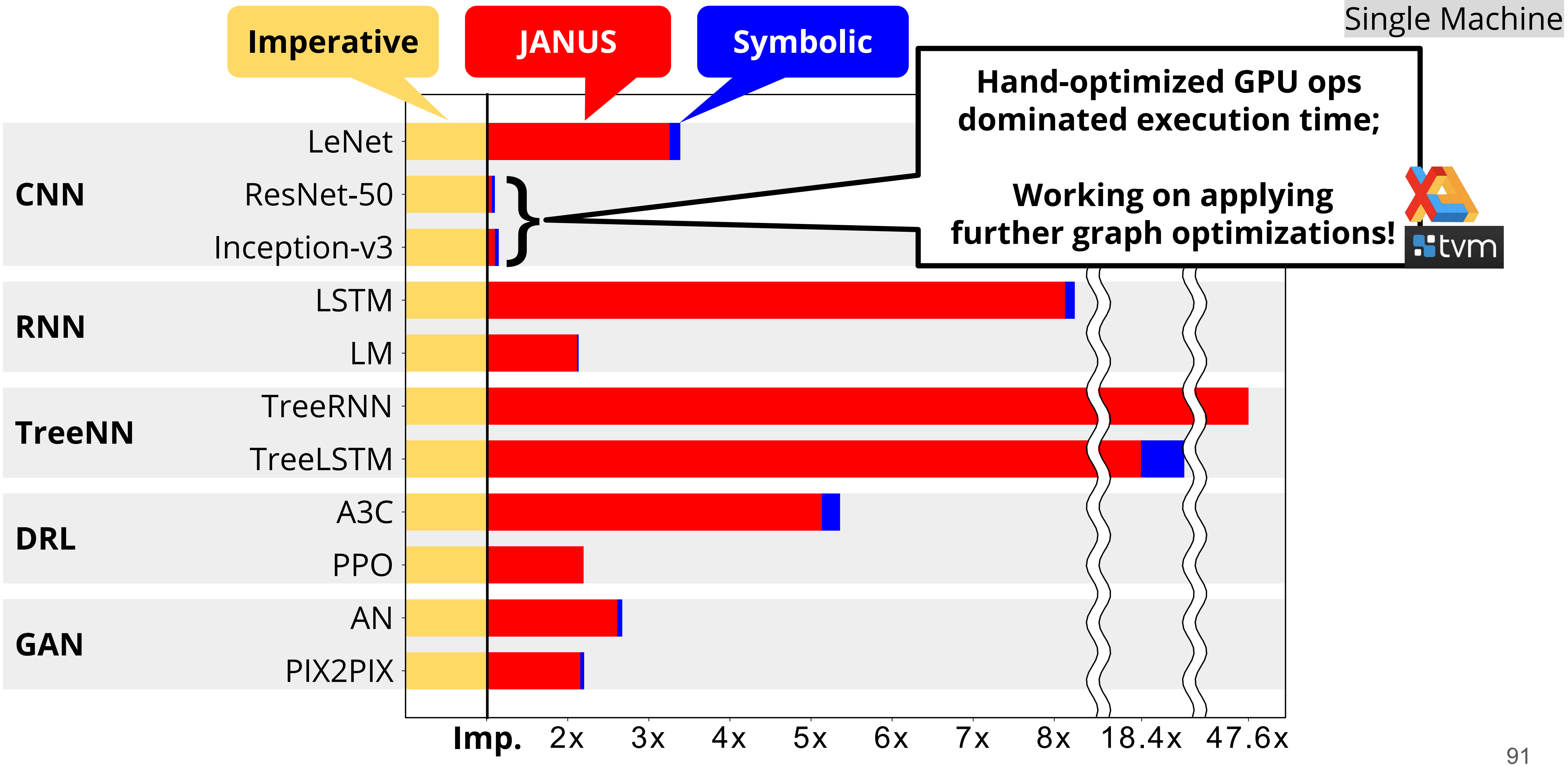


# Normalized Training Throughput

Single Machine

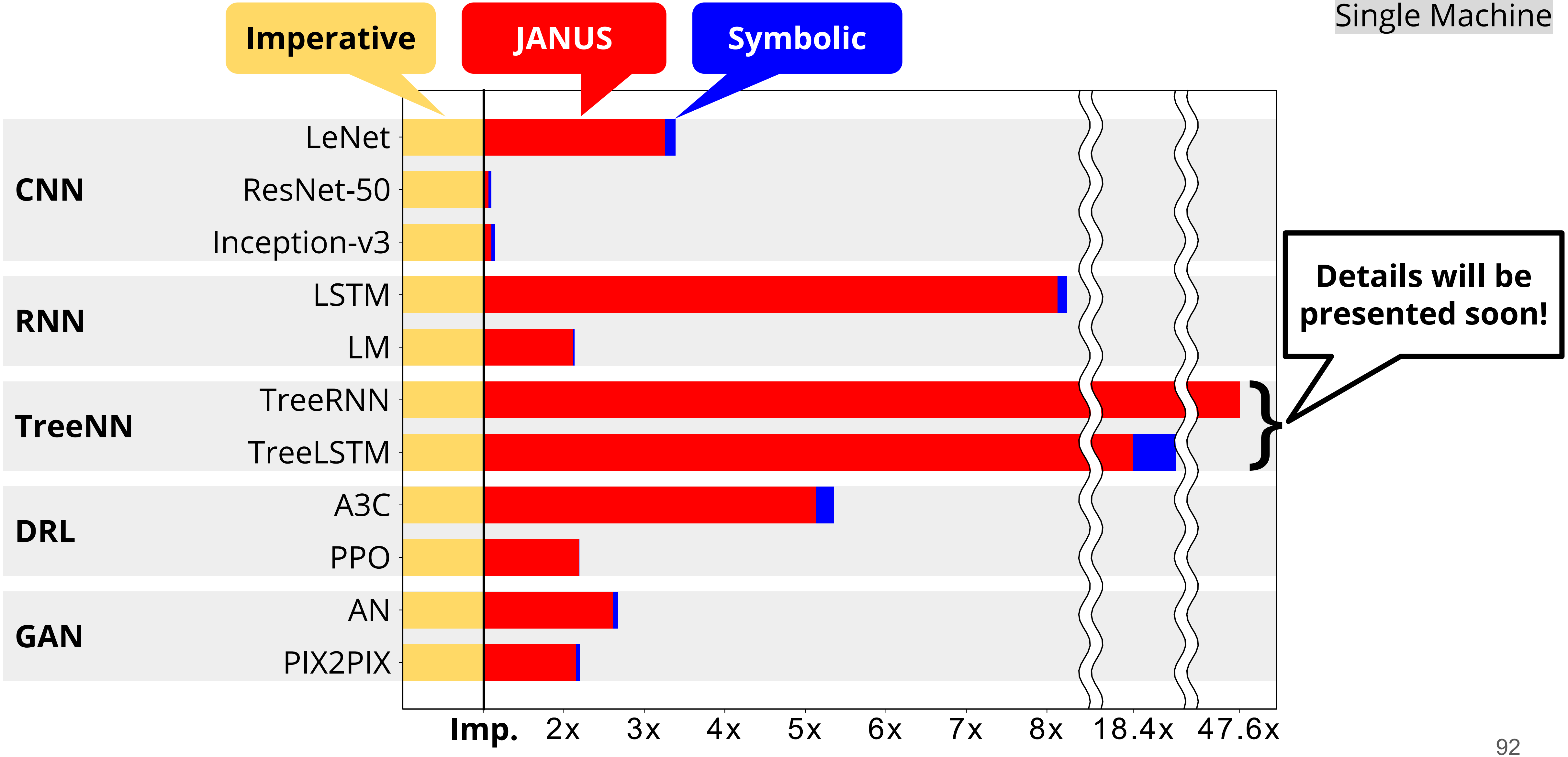


# Normalized Training Throughput



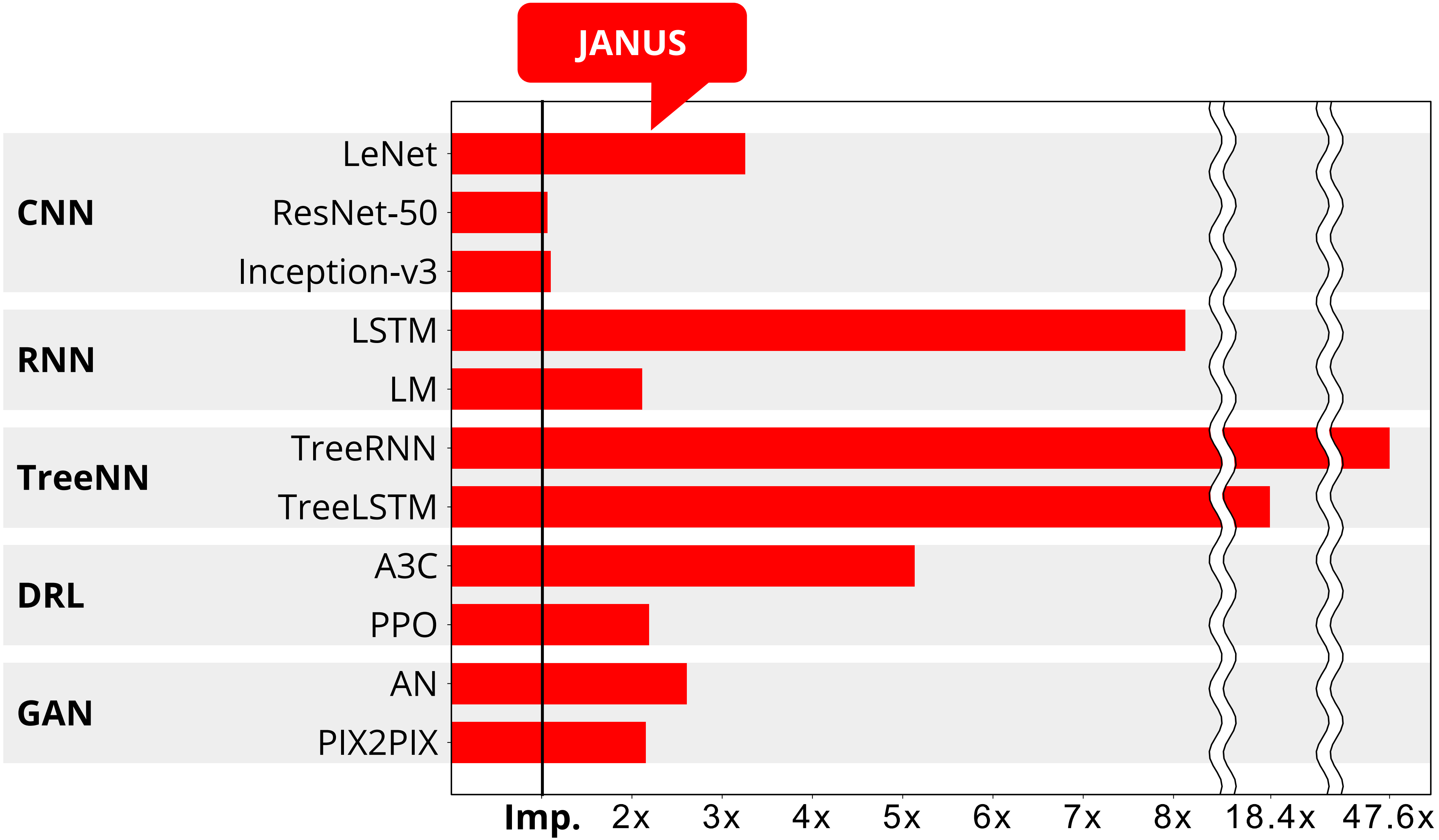
# Normalized Training Throughput

Single Machine



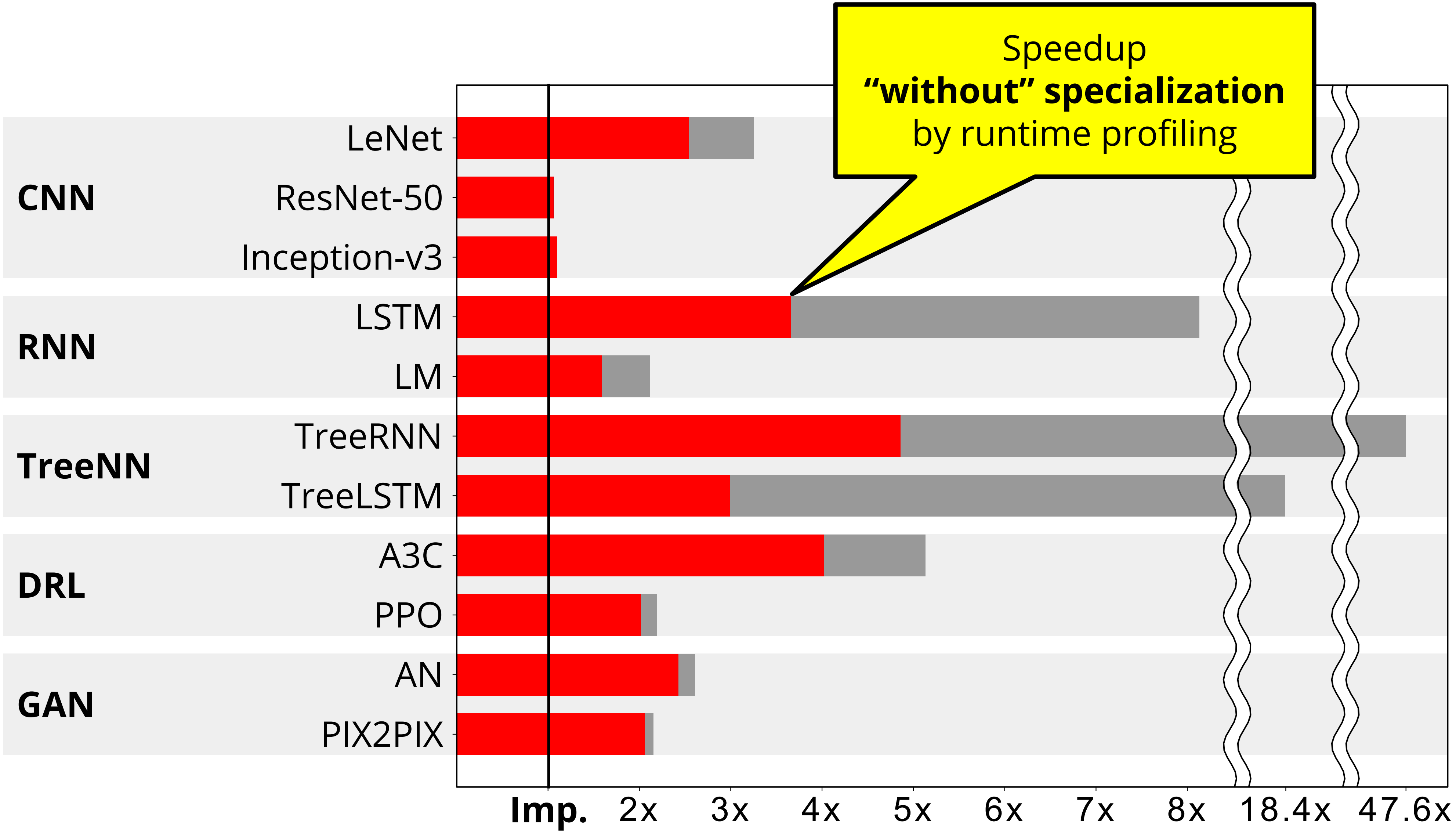
# JANUS Speedup over Imperative Execution

Single Machine



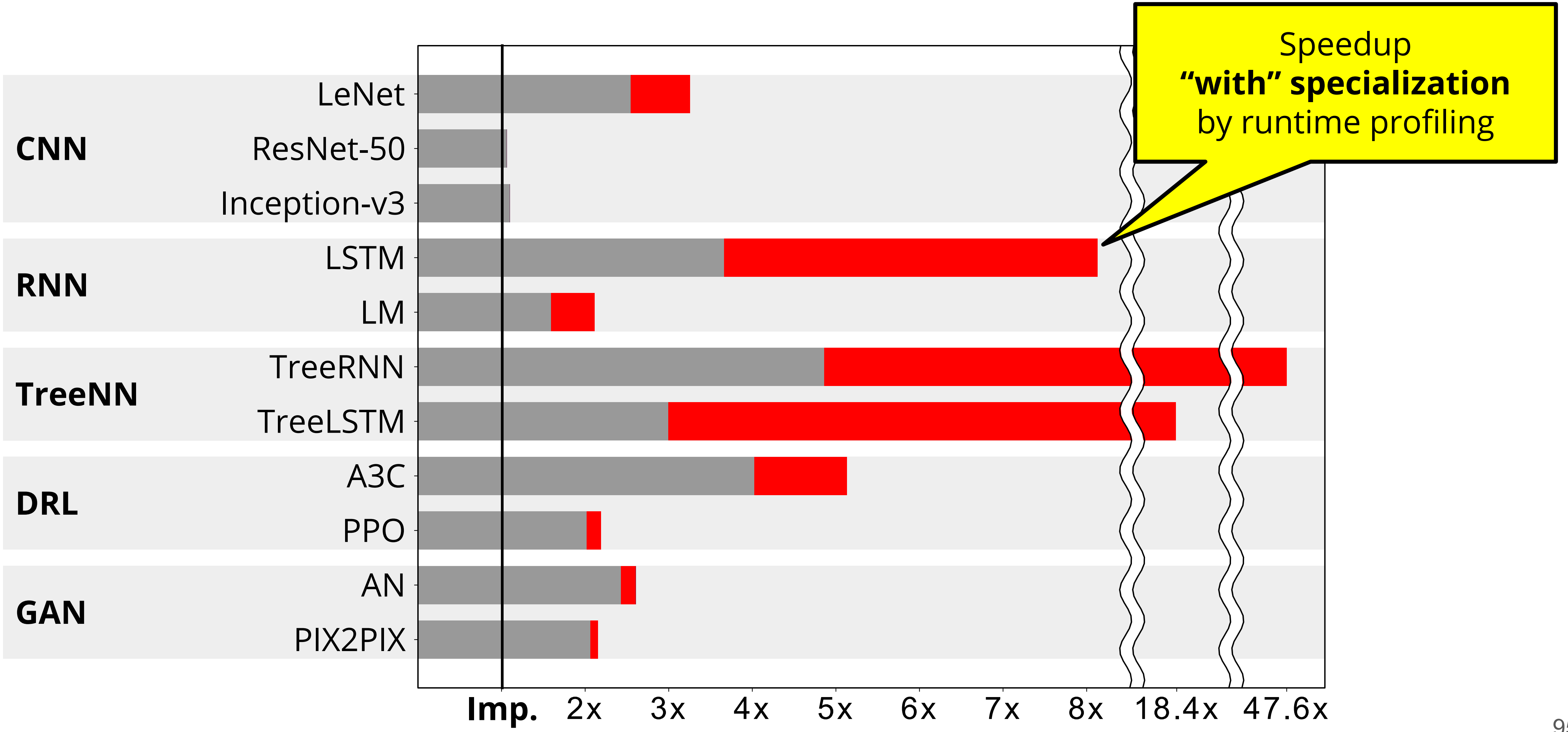
# JANUS Speedup over Imperative Execution

Single Machine



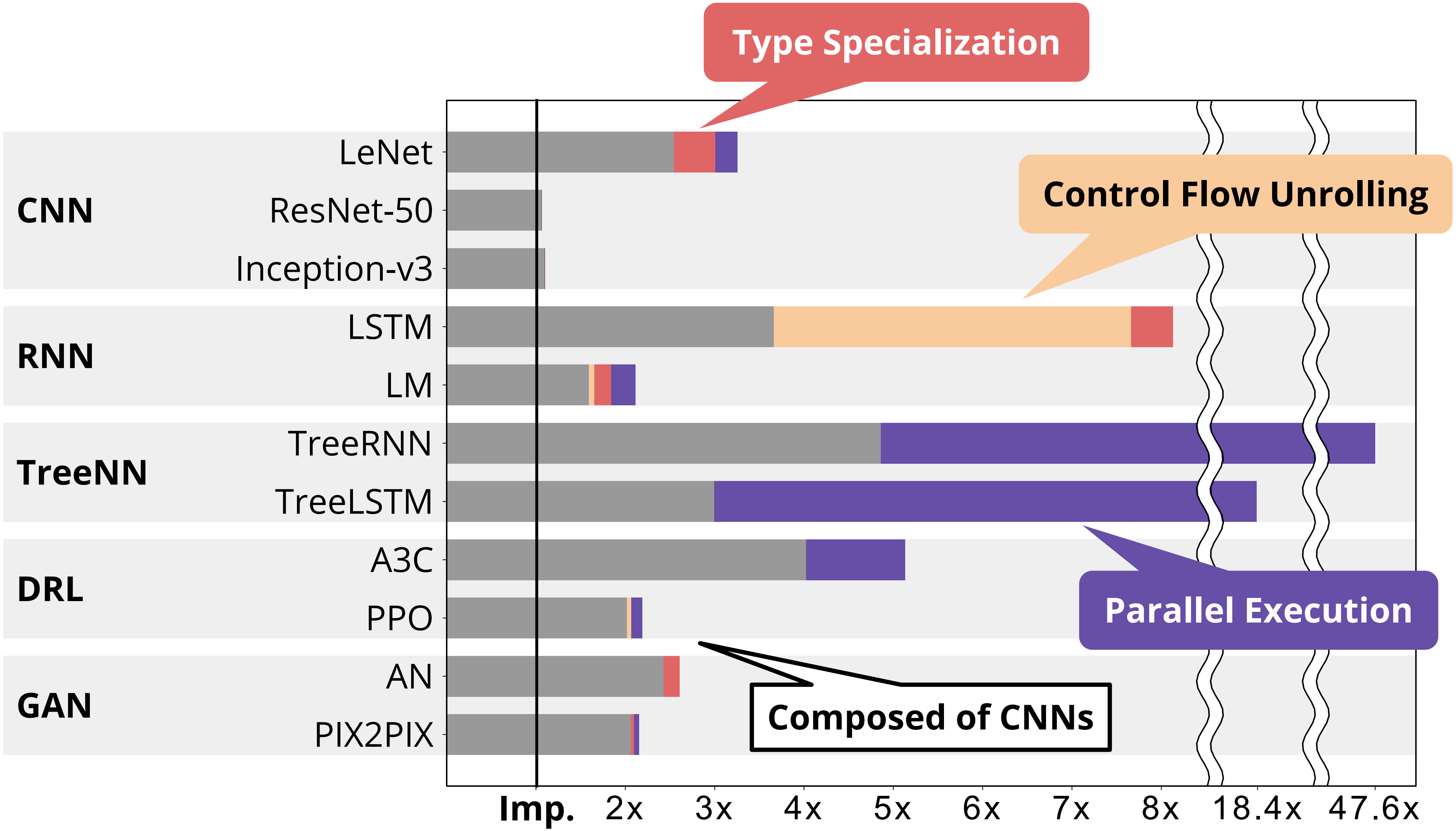
# JANUS Speedup over Imperative Execution

Single Machine



# JANUS Speedup over Imperative Execution

Single Machine



# Related Works

- Imperative to symbolic: one-shot converters
  - TensorFlow: defun, AutoGraph, Swift for TensorFlow, JAX, ...
  - PyTorch JIT trace, script
  - MXNet Gluon

- Cannot handle the dynamic semantics of Python **correctly** & **efficiently**

# JANUS: Summary

- Programmability and debuggability of imperative DL frameworks
  with the performance of symbolic DL frameworks

- Speculative graph generation and execution with runtime profiling

- Up to 47.6x speedup over imperative DL framework,
  within up to 4% difference compared to symbolic DL framework,
  while transparently and correctly executing imperative DL programs

# Outline

- JANUS
- **How to handle Recursive Neural Networks?**
- On-going Works

# Outline

- JANUS

- **How to handle Recursive Neural Networks?**

**Improving the Expressiveness of Deep Learning Frameworks with Recursion**

Eunji Jeong*
Seoul National University
ejjeong@snu.ac.kr

Joo Seong Jeong*
Seoul National University
joosjeong@snu.ac.kr

Soojeong Kim
Seoul National University
soojeong_kim@snu.ac.kr

Gyeong-In Yu
Seoul National University
gyeongin@snu.ac.kr

Byung-Gon Chun†
Seoul National University
bgchun@snu.ac.kr

**ABSTRACT**

Recursive neural networks have widely been used by researchers

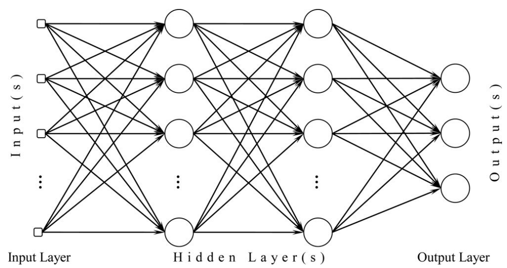
with Recursion. In *EuroSys '18: Thirteenth EuroSys Conference 2018, April 23–26, 2018, Porto, Portugal.* ACM, New York, NY, USA, 13 pages. https:

<EURO/SYS'18>

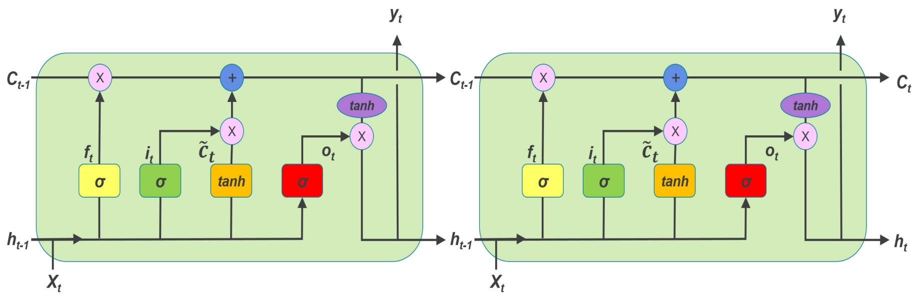
- On-going Works

# Recursive Neural Networks

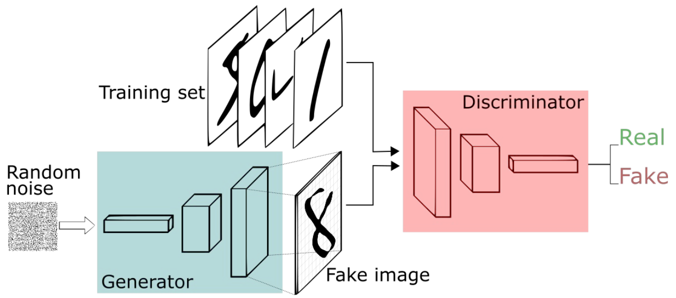
Images From:
http://www.mdpi.com/
https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/
Going Deeper with Convolutions, 2014, https://towardsdatascience.com/learn-how-recurrent-neural-networks-work-84e975feaaf7
Short-Term Load Forecasting Using EMD-LSTM Neural Networks with a Xgboost Algorithm for Feature Importance Evaluation, Energies 2017
https://skymind.ai/wiki/generative-adversarial-network-gan
https://en.wikipedia.org/wiki/Reinforcement_learning
https://medium.com/@Petuum/intro-to-dynamic-neural-networks-and-dynet-67694b18cb23



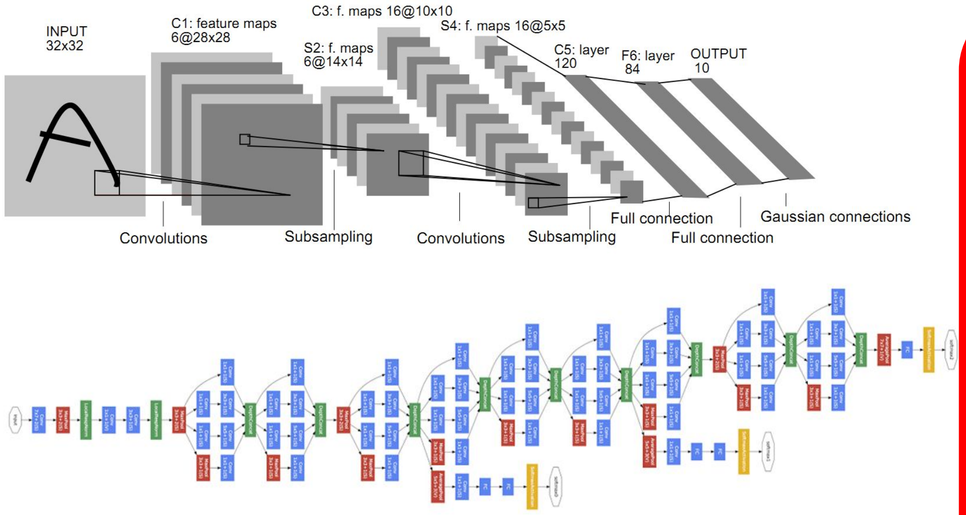

Multilayer Perceptron



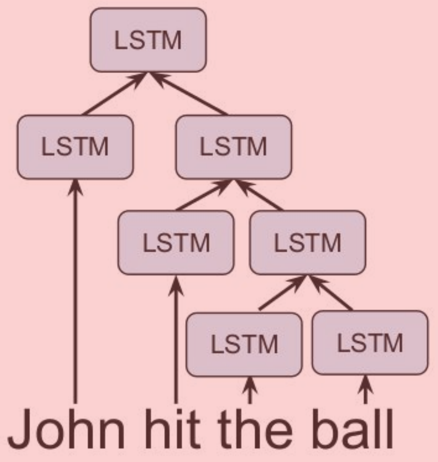

Recurrent Neural Networks



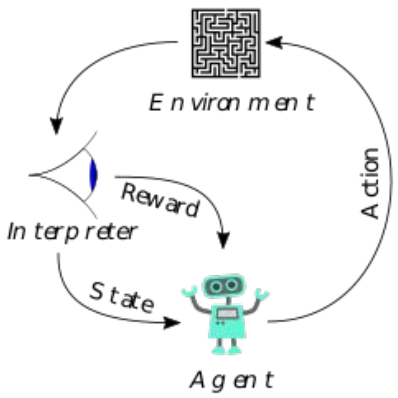

Generative Adversarial Networks




Convolutional Neural Networks



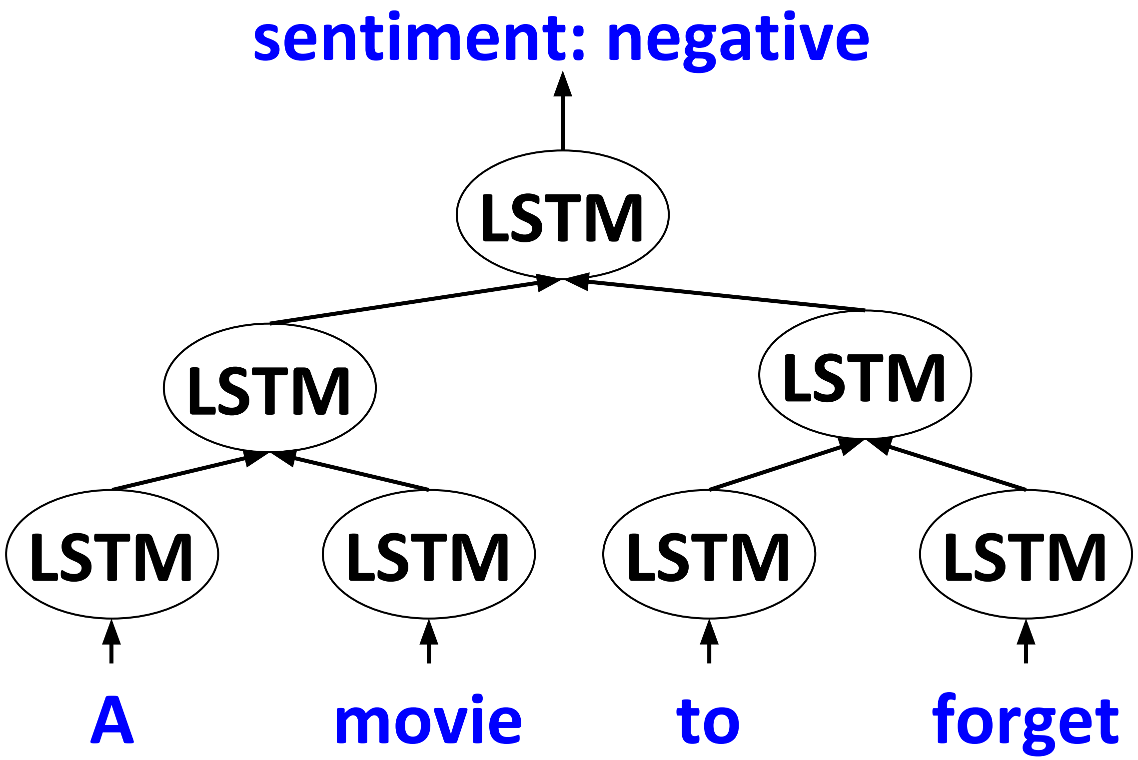

Recursive Neural Networks




Deep Reinforcement Learning Models

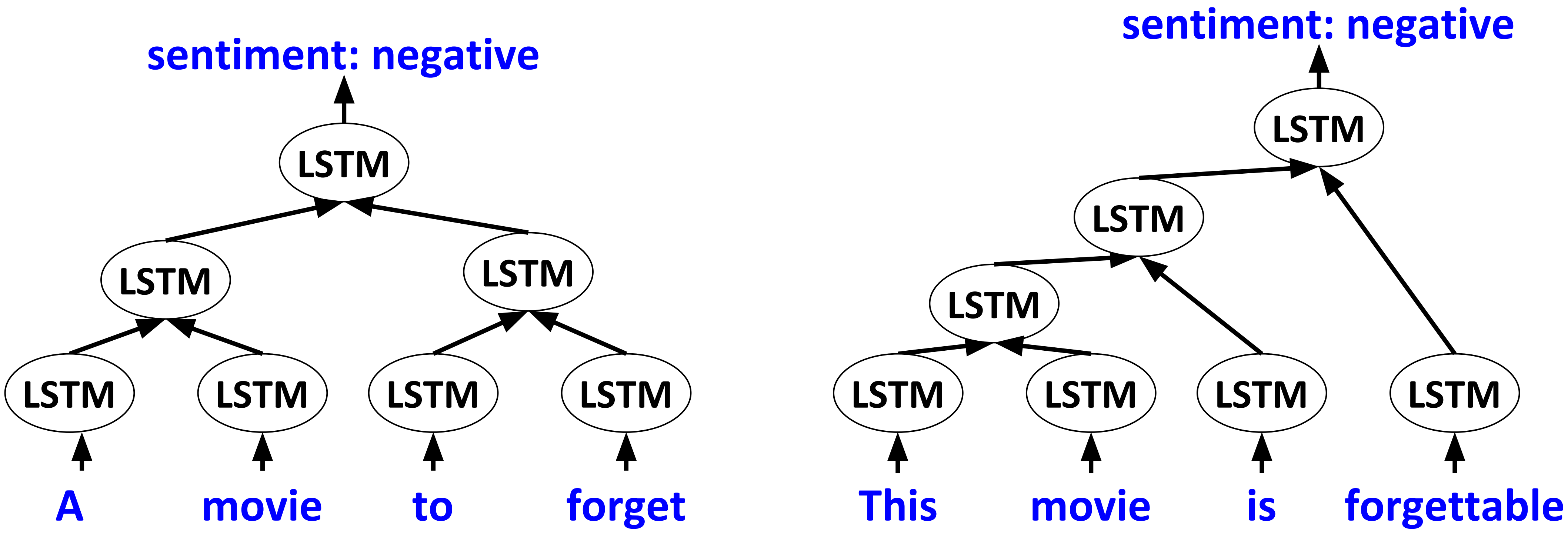
# Recursive Neural Networks

- Apply the same set of weights **recursively** over structured inputs
- Example: **TreeLSTM** → Sentiment analysis on sentence parse trees

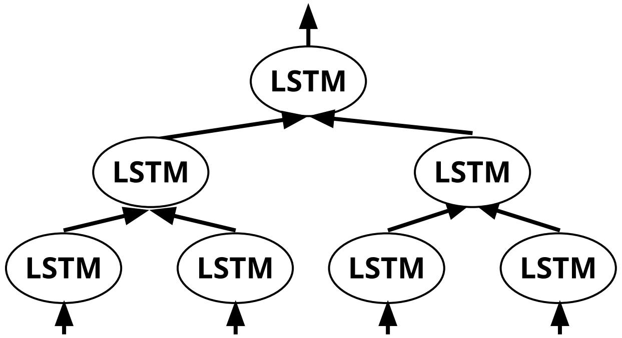


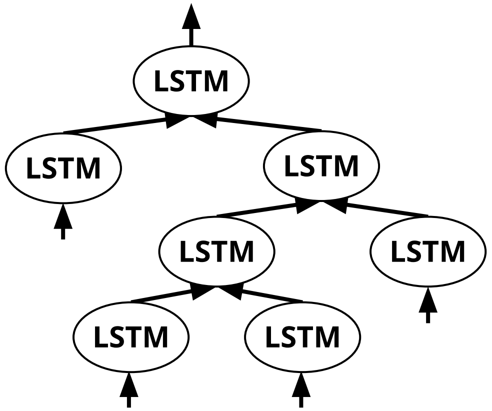
# Recursive Neural Networks

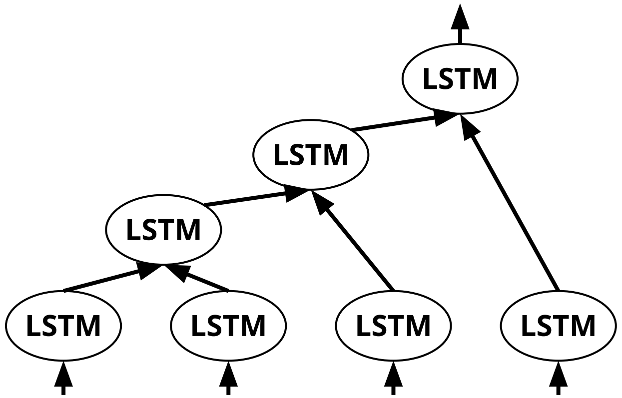
- Apply the same set of weights **recursively** over structured inputs
- Example: **TreeLSTM** → Sentiment analysis on sentence parse trees



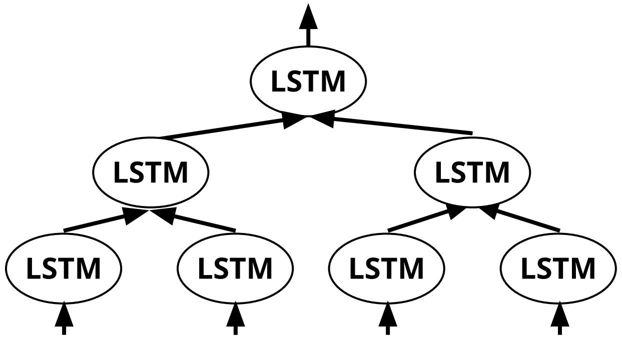
# How to Implement TreeLSTM?

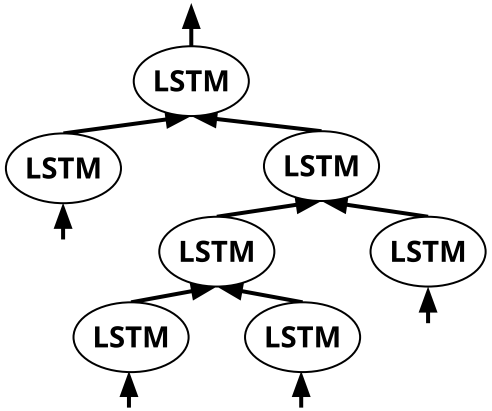


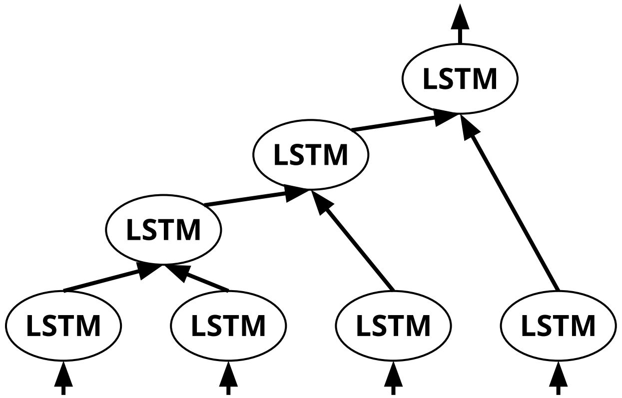



# How to Implement TreeLSTM?



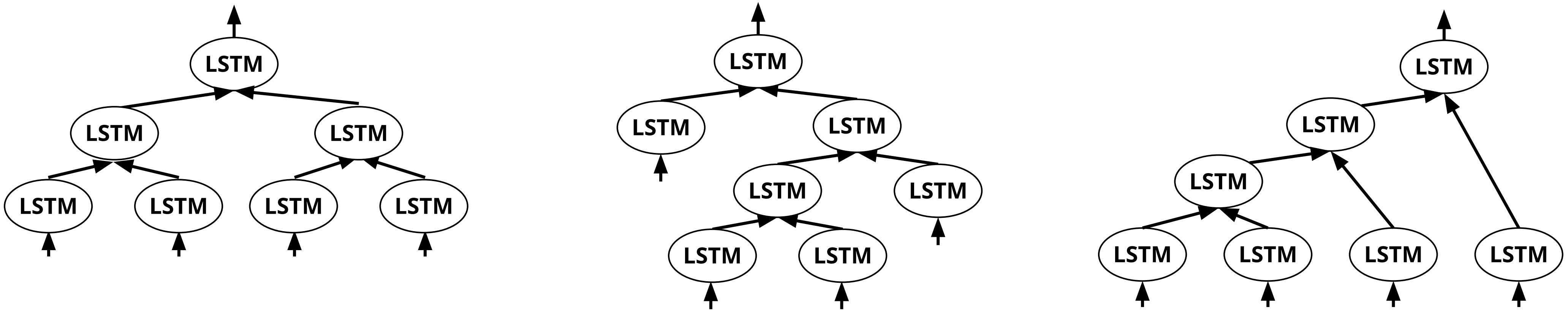




## Imperative Program

## Symbolic DL Graph

# How to Implement TreeLSTM?

**Imperative** Symbolic



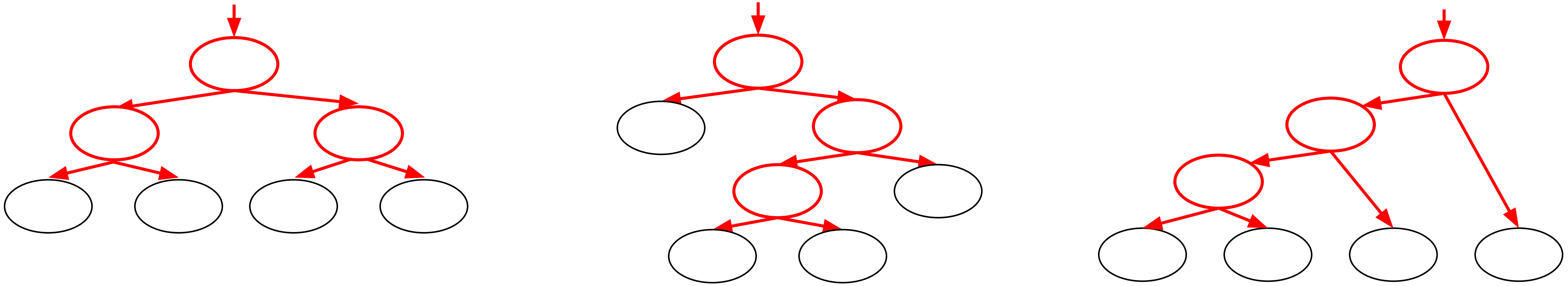

**Imperative Program**

```
def TreeLSTM(node):
   if node.is_leaf:
     return LSTM(node.word)
   else:
     lstate = TreeLSTM(node.left)
     rstate = TreeLSTM(node.right)
     return LSTM(lstate, rstate)
for tree in trees:
   TreeLSTM(tree)
```

**Symbolic DL Graph**

# How to Implement TreeLSTM?

Imperative | Symbolic



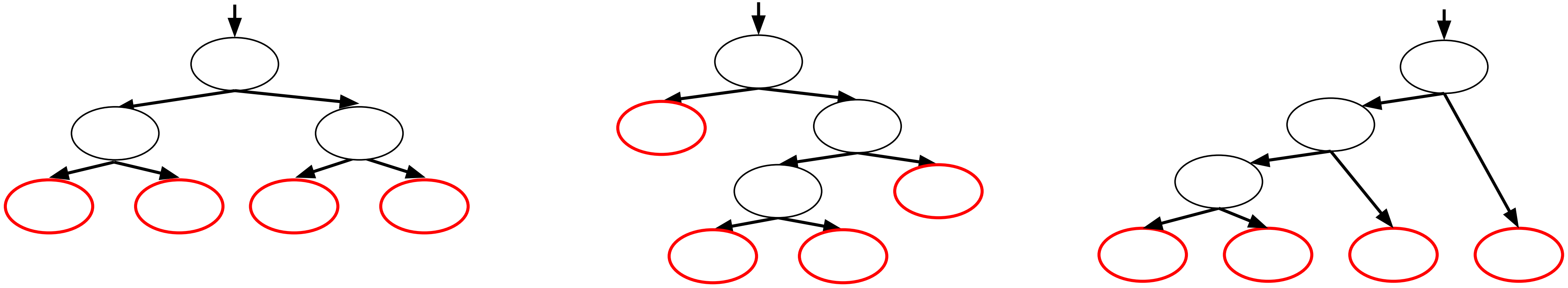
**Imperative Program**

```
def TreeLSTM(node):
    if node.is_leaf:
      return LSTM(node.word)
    else:
      lstate = TreeLSTM(node.left)
      rstate = TreeLSTM(node.right)
      return LSTM(lstate, rstate)
for tree in trees:
    TreeLSTM(tree)
```

Symbolic DL Graph

# How to Implement TreeLSTM?

**Imperative** Symbolic



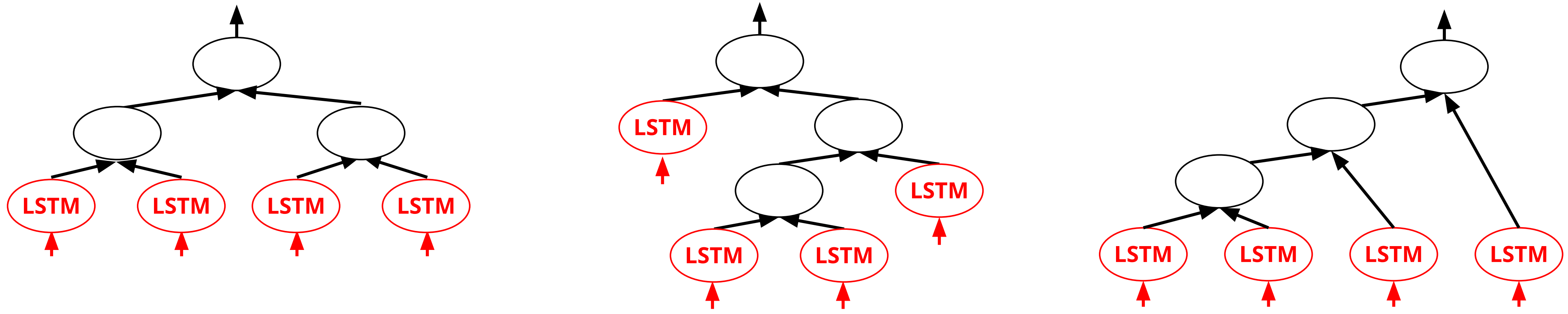
**Imperative Program**

```
def TreeLSTM(node):
    if node.is_leaf:
      return LSTM(node.word)
    else:
      lstate = TreeLSTM(node.left)
      rstate = TreeLSTM(node.right)
      return LSTM(lstate, rstate)
for tree in trees:
    TreeLSTM(tree)
```

**Symbolic DL Graph**

# How to Implement TreeLSTM?

Imperative | Symbolic



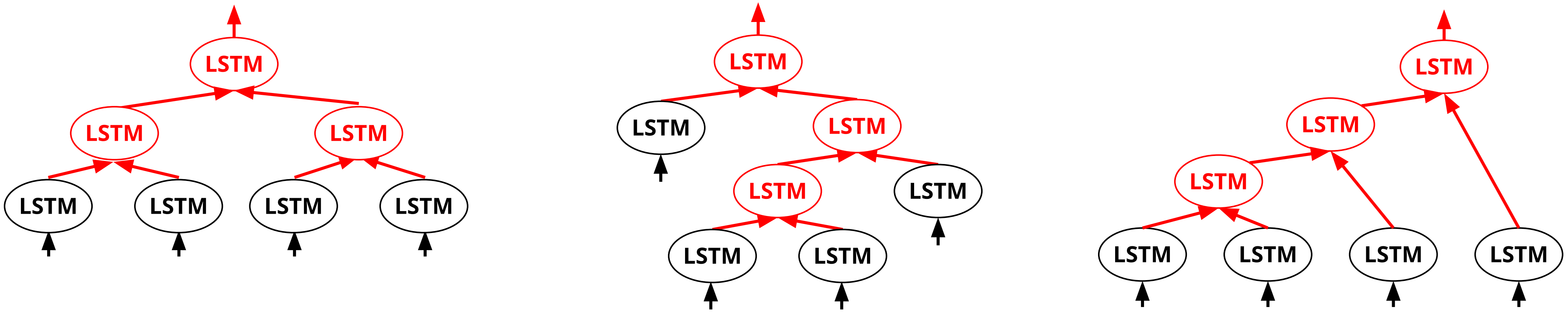

**Imperative Program**

```
def TreeLSTM(node):
    if node.is_leaf:
        return LSTM(node.word)
    else:
        lstate = TreeLSTM(node.left)
        rstate = TreeLSTM(node.right)
        return LSTM(lstate, rstate)
for tree in trees:
    TreeLSTM(tree)
```

**Symbolic DL Graph**

# How to Implement TreeLSTM?

**Imperative** Symbolic



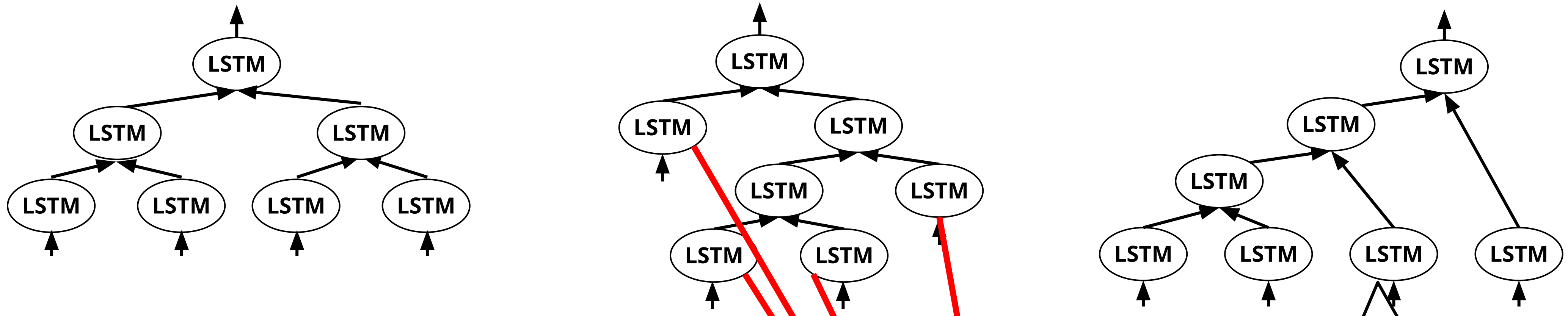

**Imperative Program**

```
def TreeLSTM(node):
   if node.is_leaf:
     return LSTM(node.word)
   else:
     lstate = TreeLSTM(node.left)
     rstate = TreeLSTM(node.right)
     return LSTM(lstate, rstate)
for tree in trees:
   TreeLSTM(tree)
```

Symbolic DL Graph

# How to Execute TreeLSTM?

Imperative | Symbolic

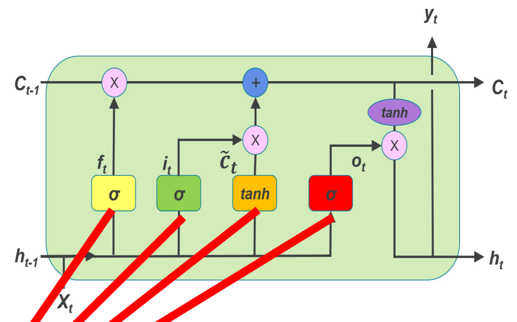



**Imperative Program**

```
def TreeLSTM(node):
    if node.is_leaf:
        return LSTM(node.word)
    else:
        lstate = TreeLSTM(node.left)
        rstate = TreeLSTM(node.right)
        return LSTM(lstate, rstate)
for tree in trees:
    TreeLSTM(tree)
```

Symbolic DL Gra

**LSTM**



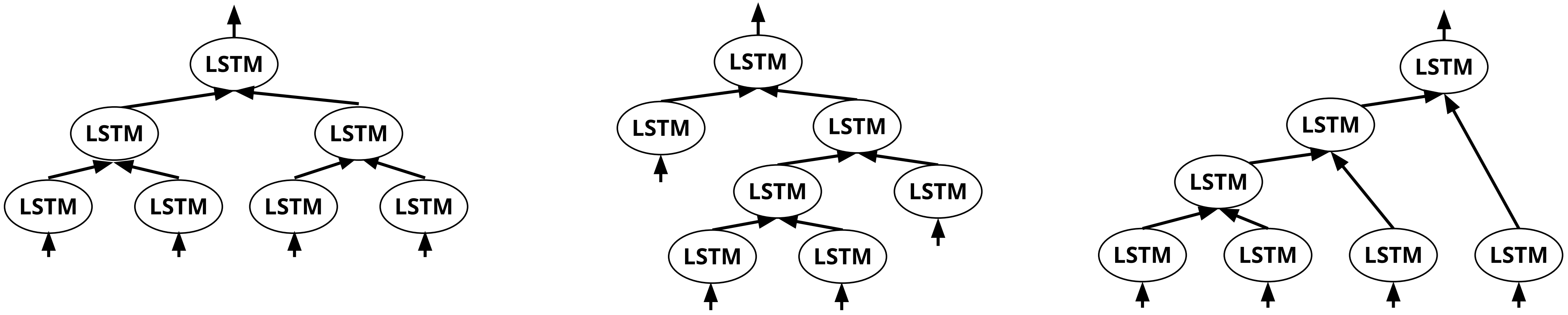
**multiple children of a tree**

**Parallelly Executable**

**multiple functions in a LSTM cell**

# How to Execute TreeLSTM?

Imperative | **Symbolic**




**Imperative Program**

```
def TreeLSTM(n de):
    if node.is   f:
        retur
    else:
        lstat                    .left)
        rstate = TreeLSTM(node.right)
        return LSTM(lstate, rstate)
for tree in trees:
    TreeLSTM(tree)
```

**No Parallelism**

**Symbolic DL Graph**

**New graph** for every sentence?
**High graph gen&opt overhead**

**Preferred?!**

How to represent
**all potential input structures**
in a **single graph?**

## Problem Statement

⚠ **Expressiveness**:
How to express **recursive** structures as a symbolic graph?

⚠ **Performance**:
How to exploit **parallelism**?

## Our Solution

✔ **Expressiveness**:
How to express **recursive** structures as a symbolic graph?

⇒ Abstractions for expressing **recursion** in symbolic DL graphs

✔ **Performance**:
How to exploit **parallelism**?

⇒ A System that executes such abstractions **in parallel**

Up to **30.2x** faster training, **147.9x** faster inference
(Implemented on top of TensorFlow, compared to PyTorch)

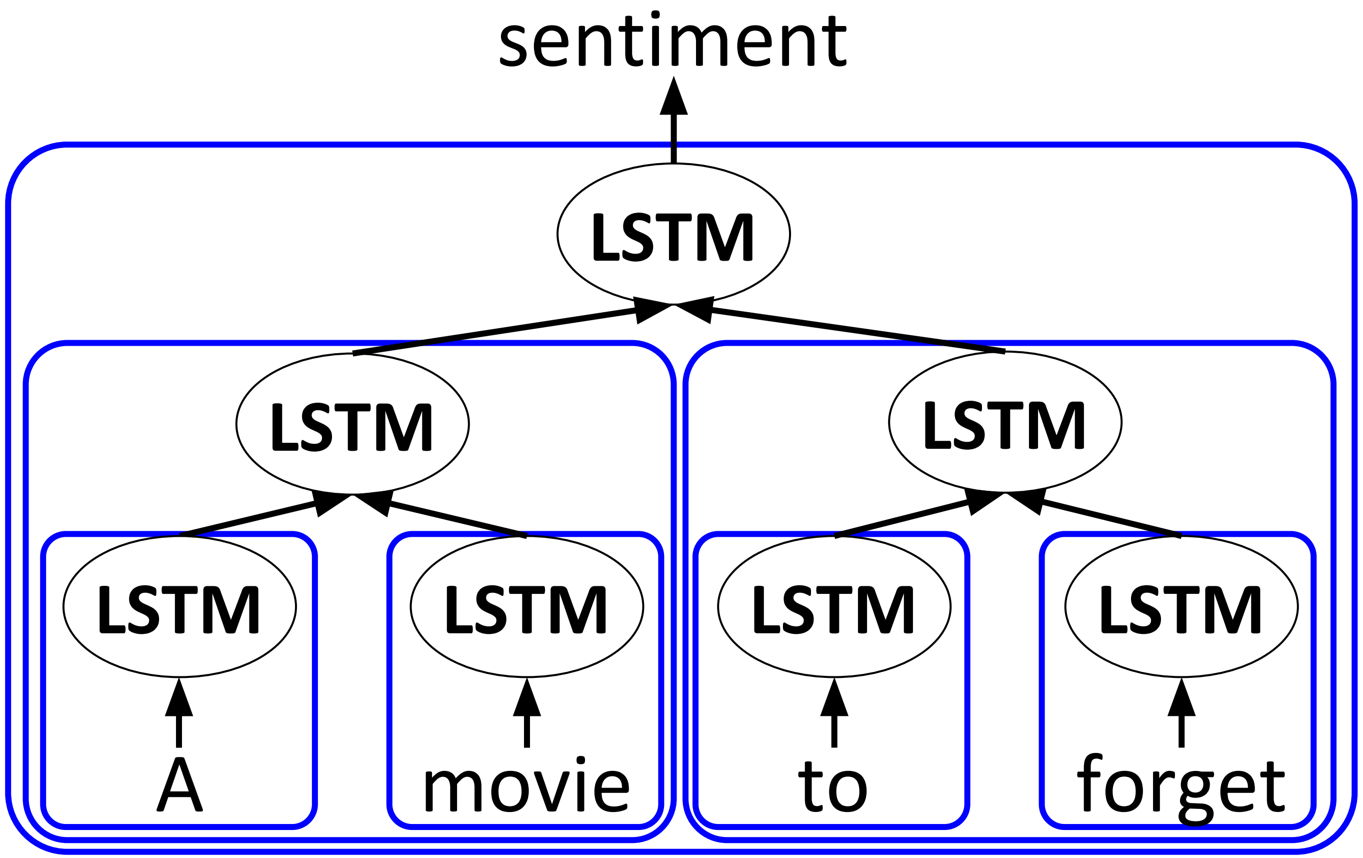
## Outline

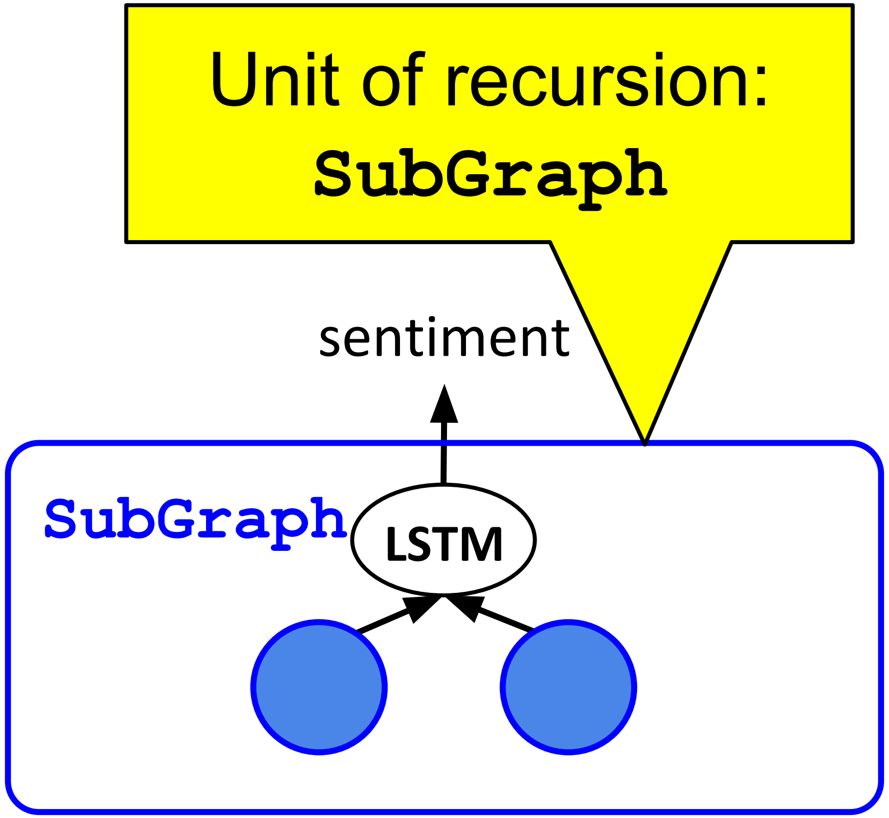
- JANUS
- How to handle Recursive Neural Networks?
  - Motivation
  - **New Abstractions**
  - Underlying System
  - TreeLSTM on JANUS
- On-going Works

# Abstractions for Recursion

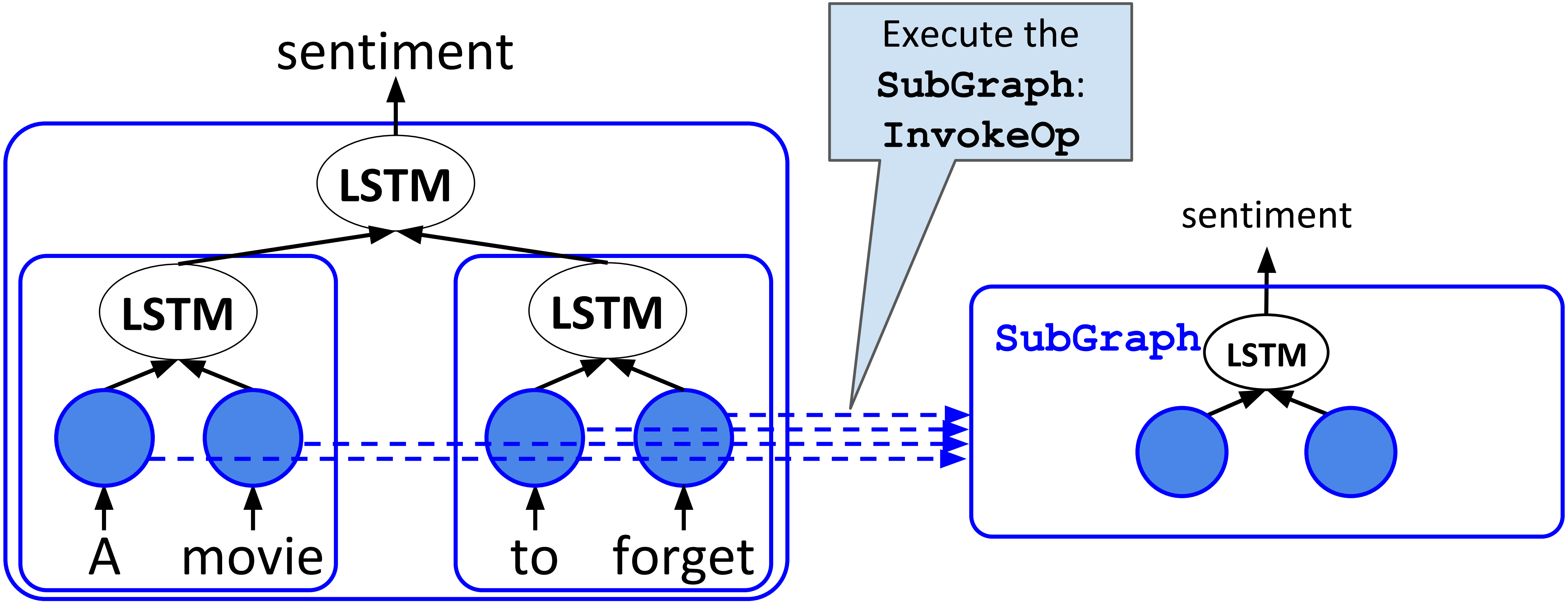
```
subgraph TreeLSTM(node):
  left = TreeLSTM(node.left)
  right = TreeLSTM(node.right)
  return LSTM(left, right)

root_sentiment = TreeLSTM(sentence)
```

# Abstractions for Recursion

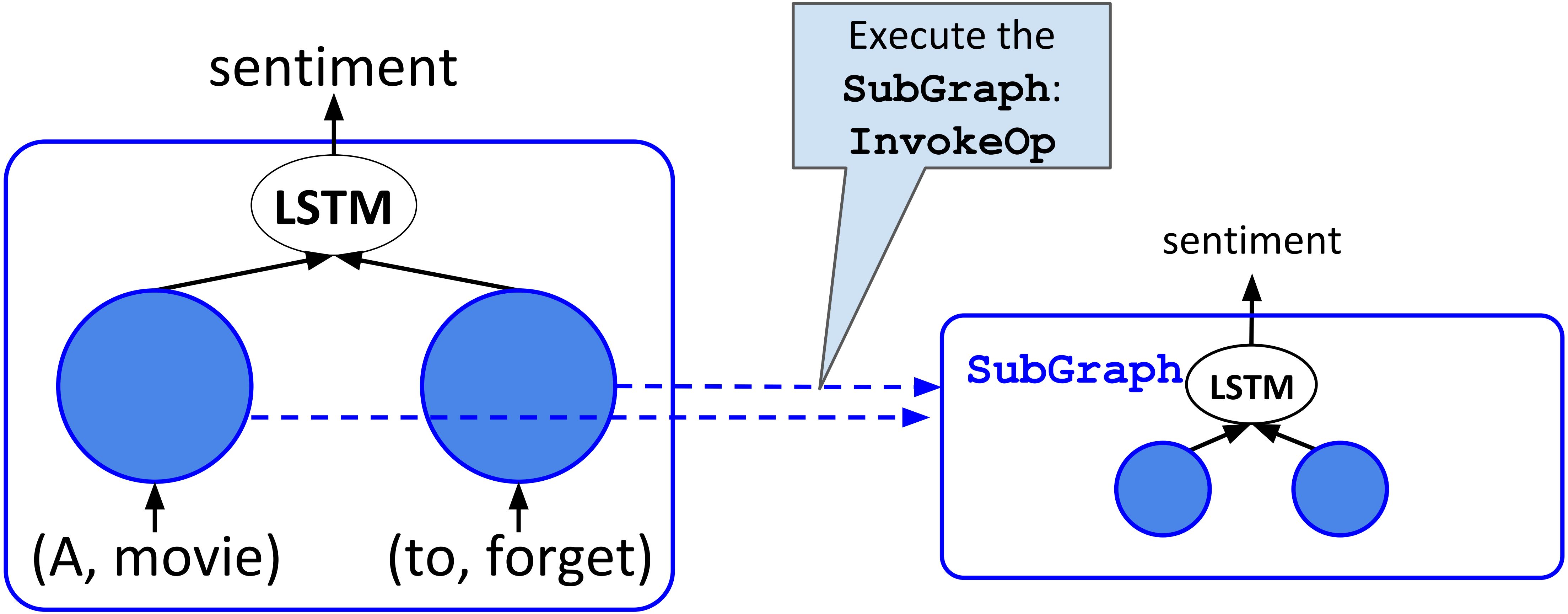


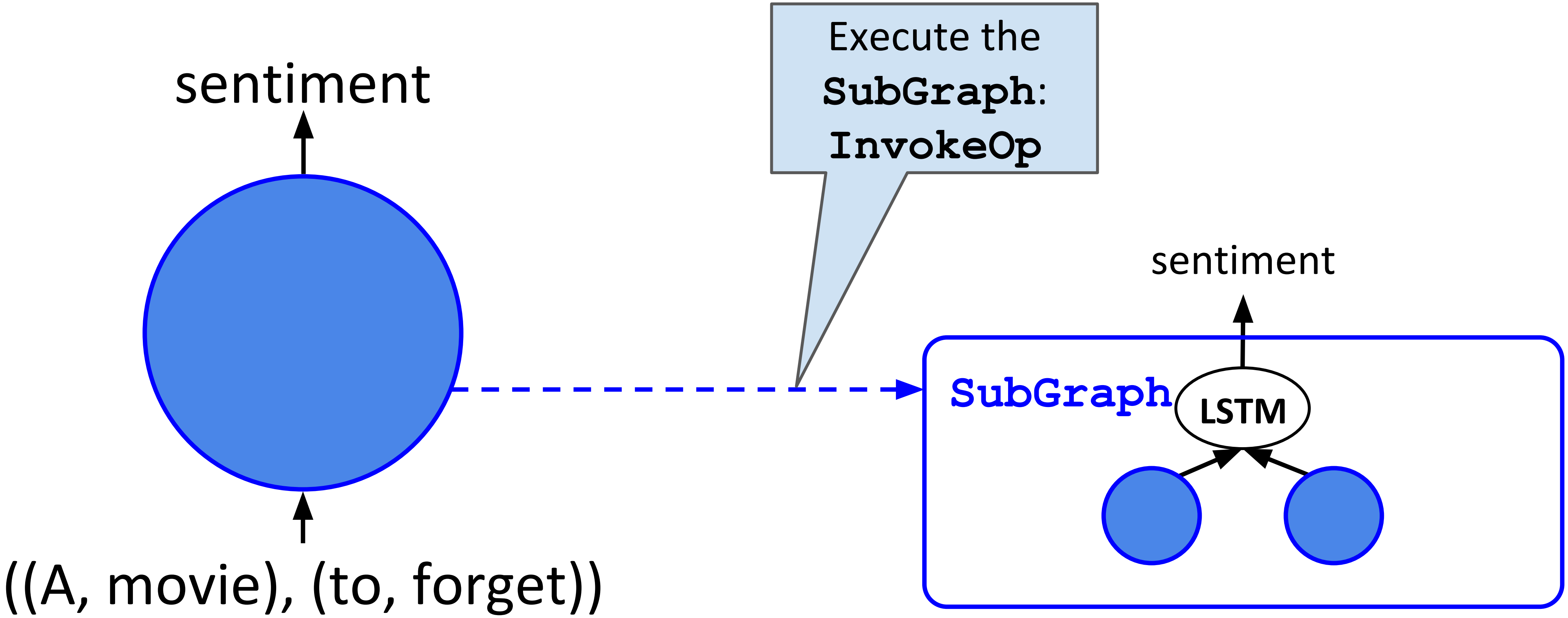
# Abstractions for Recursion



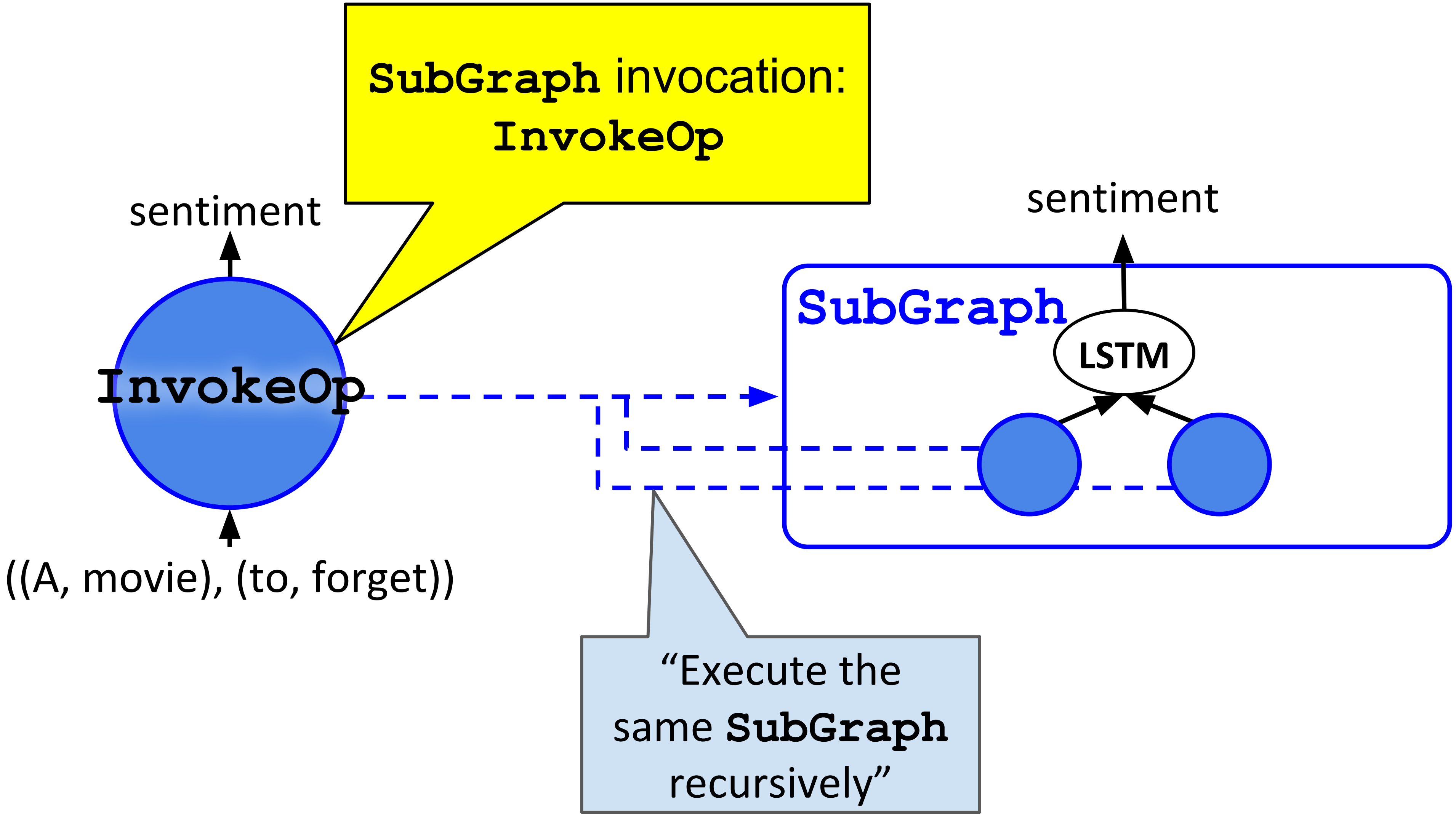
# Abstractions for Recursion



# Abstractions for Recursion

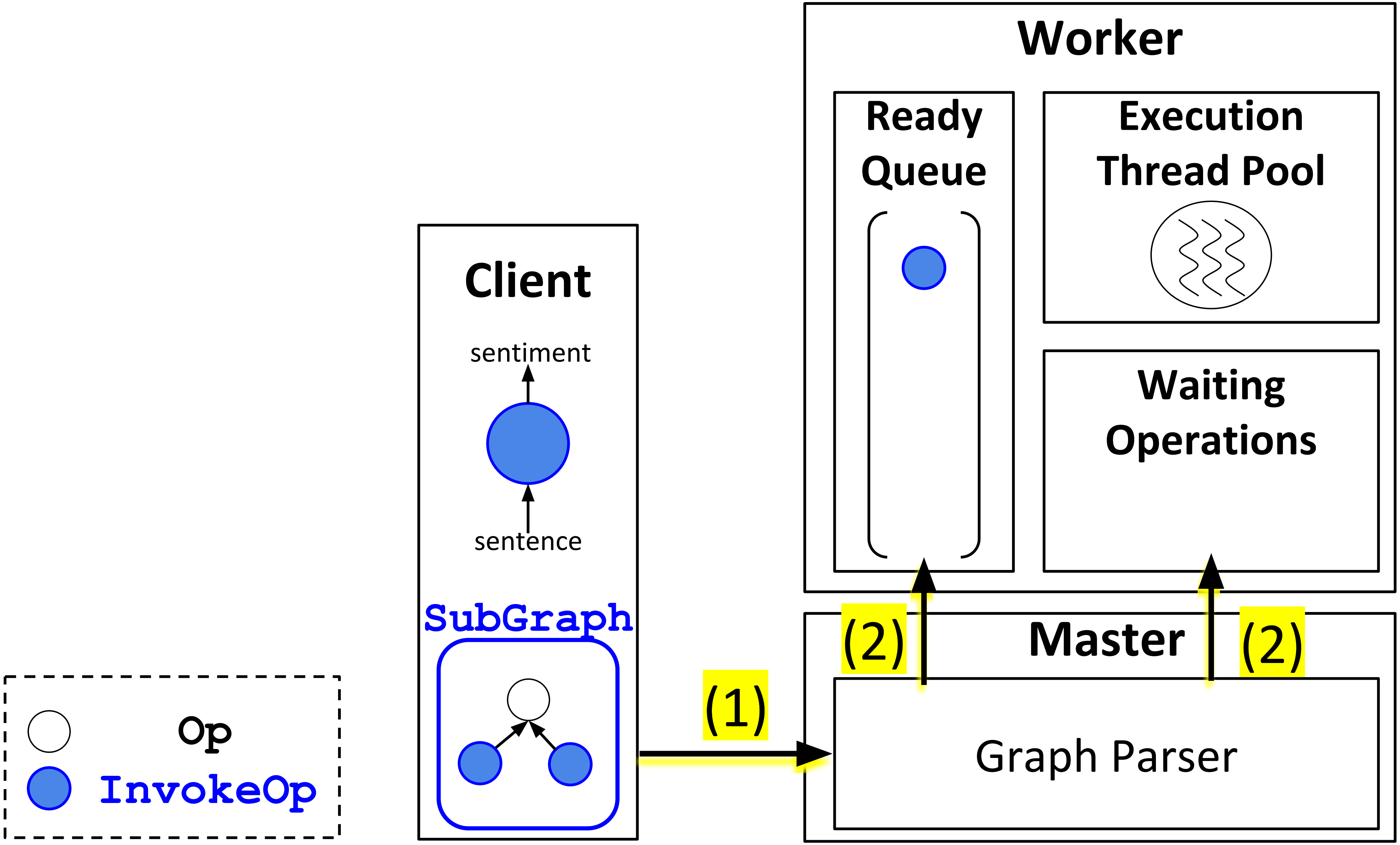


# Abstractions for Recursion



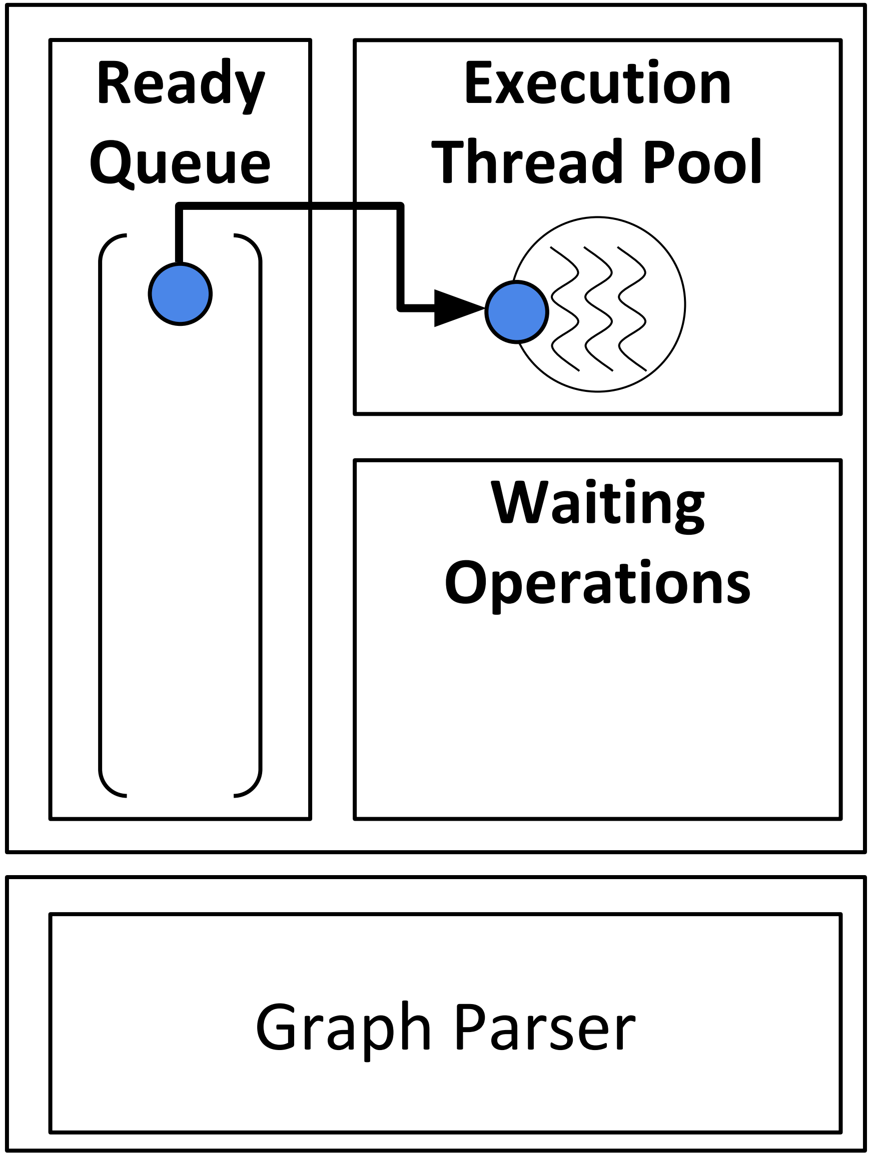
# Outline

- JANUS
- How to handle Recursive Neural Networks?
  - Motivation
  - New Abstractions
  - **Underlying System**
  - TreeLSTM on JANUS
- On-going Works

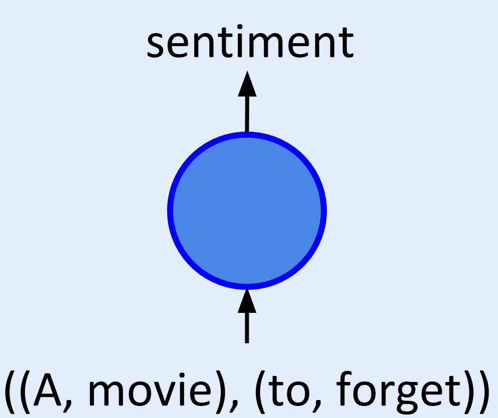

Worker
Ready Queue
Execution Thread Pool
Waiting Operations
Client
sentiment
sentence
SubGraph
(1)
(2)
Master
(2)
Graph Parser
Op
InvokeOp

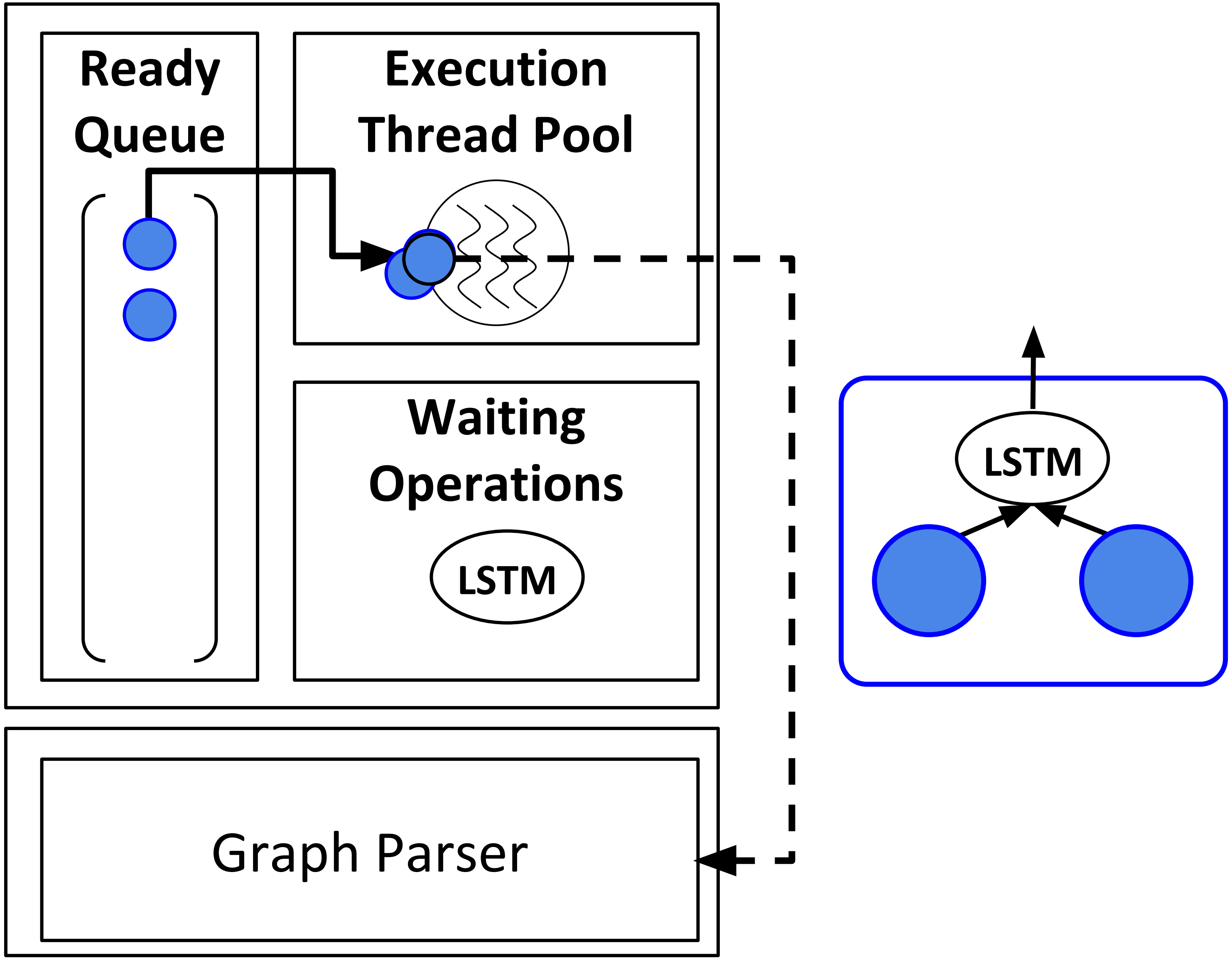
# Underlying System

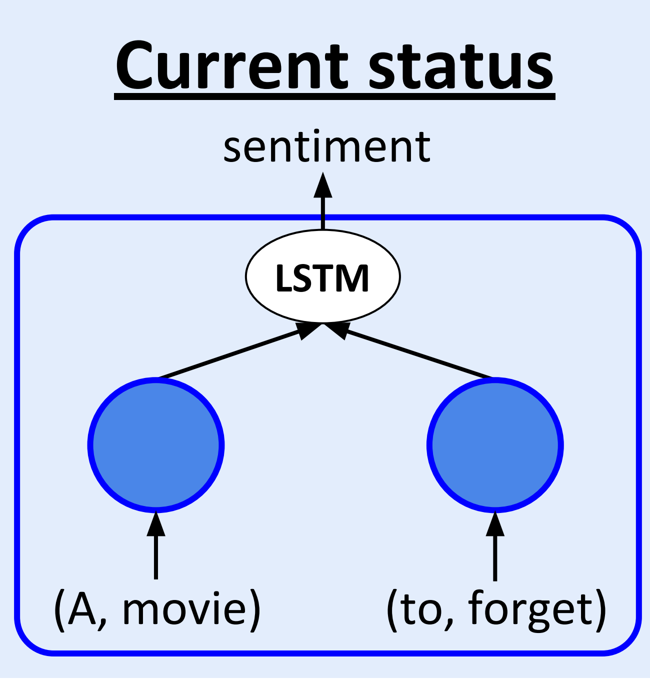


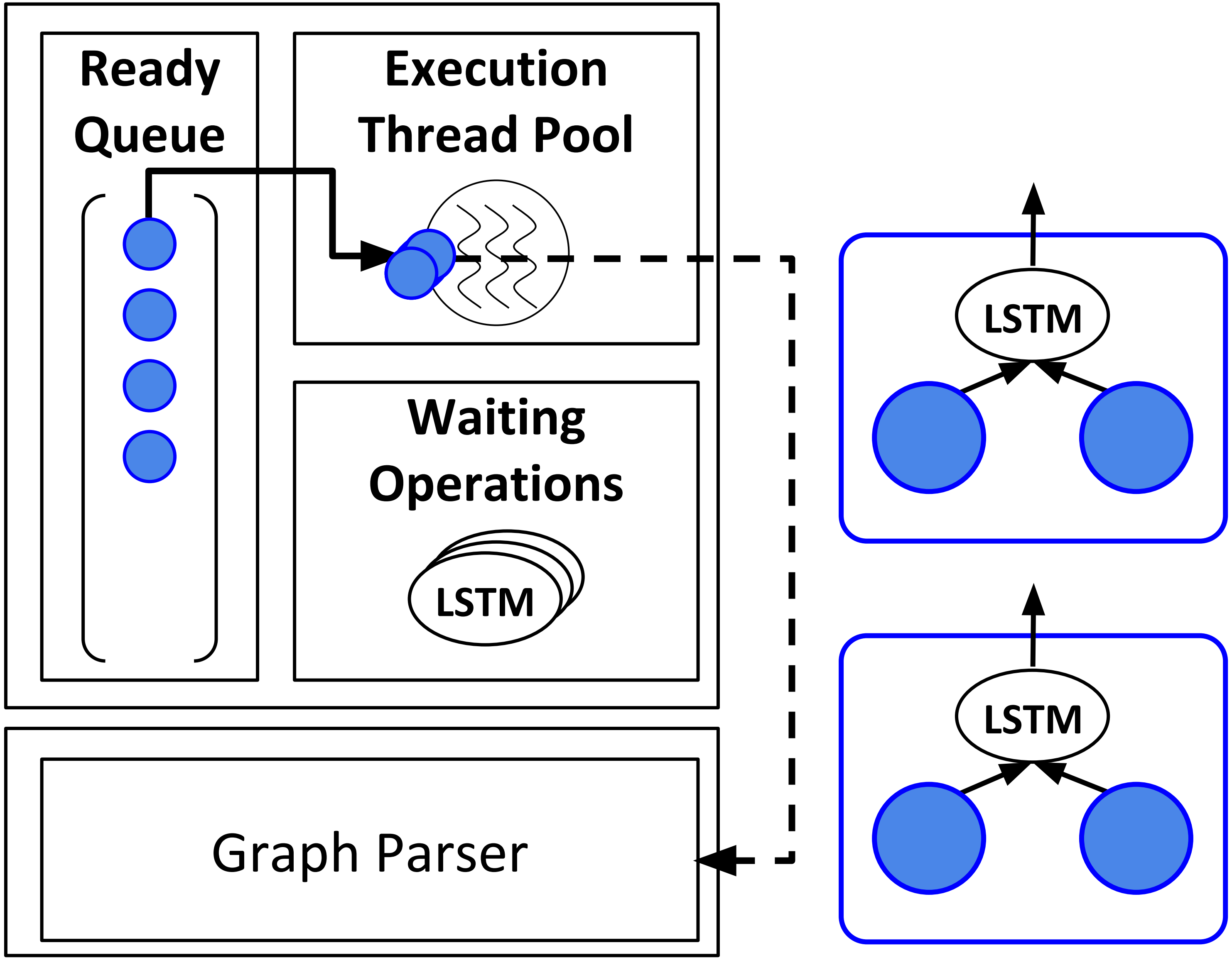


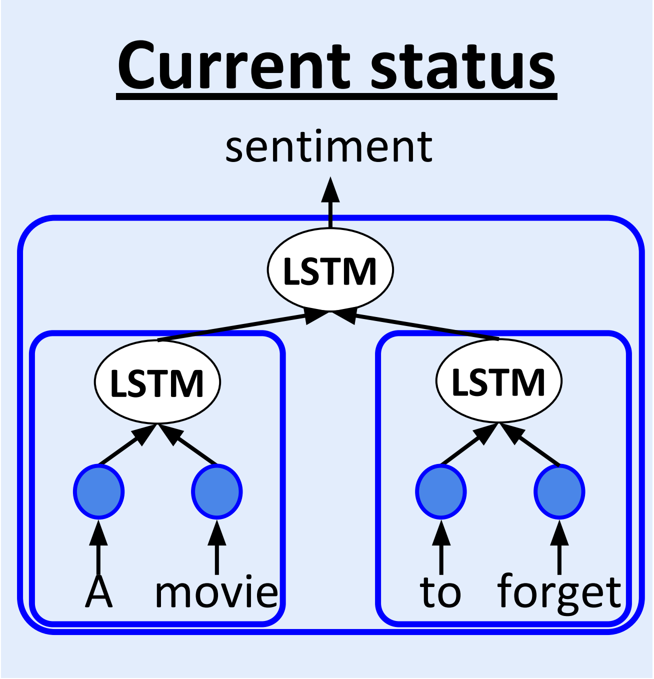

**Current status**

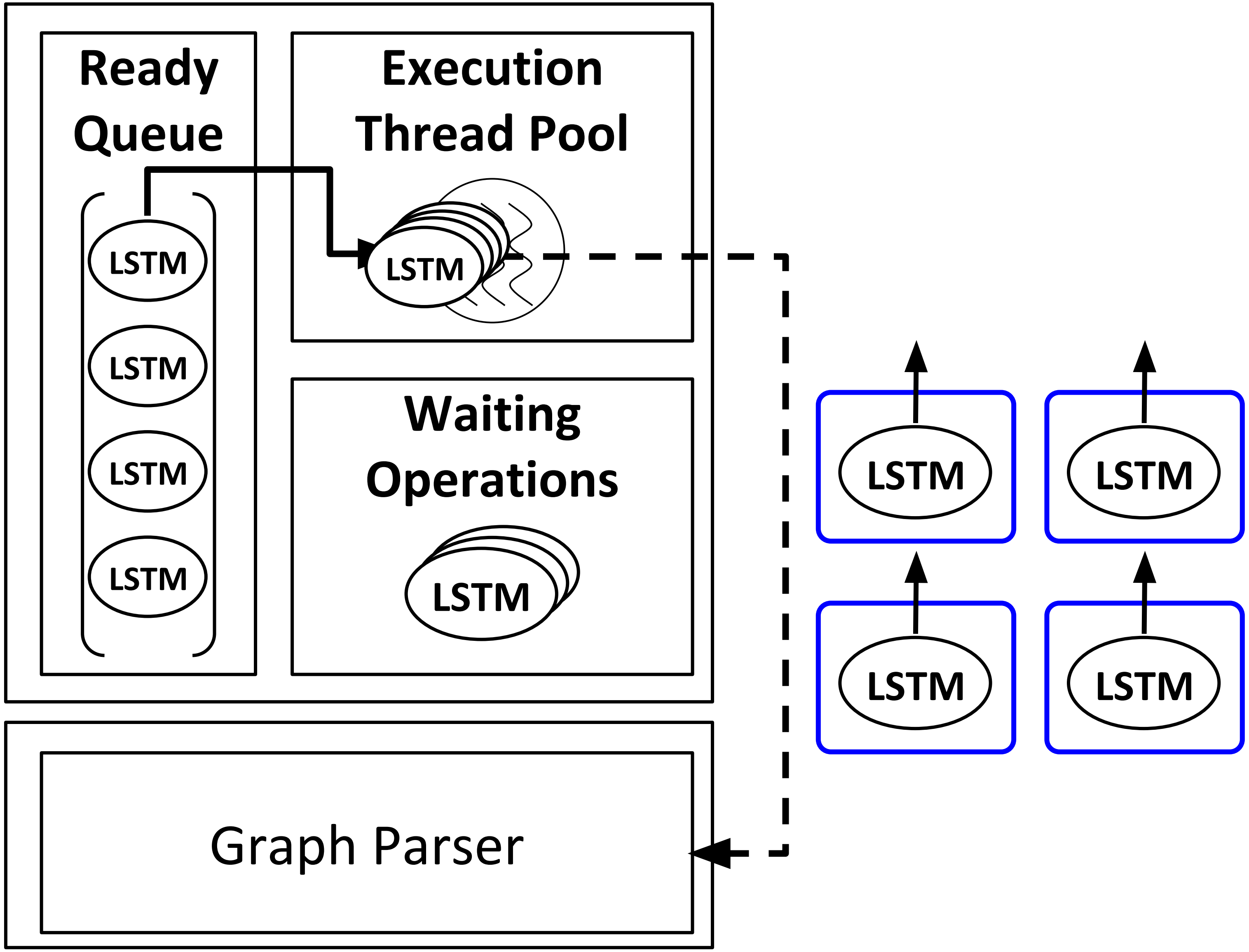


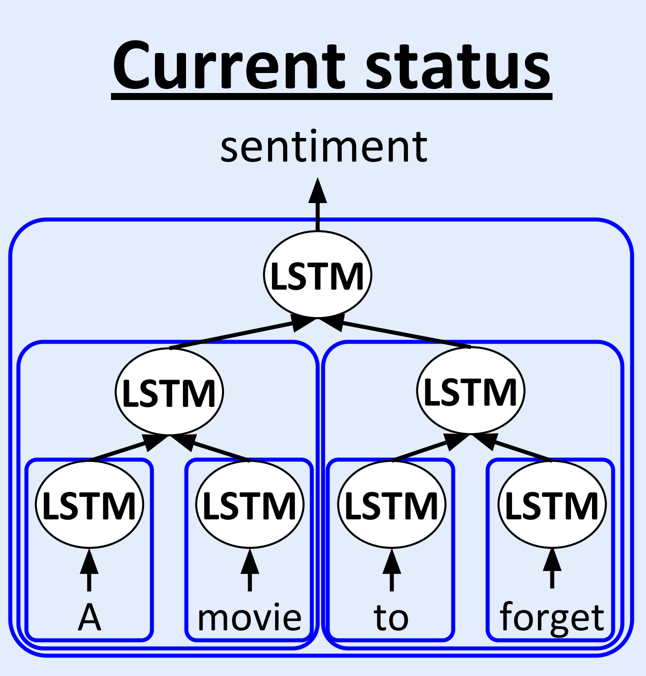

Ready
Queue
Execution
Thread Pool
Waiting
Operations
LSTM
Graph Parser
LSTM

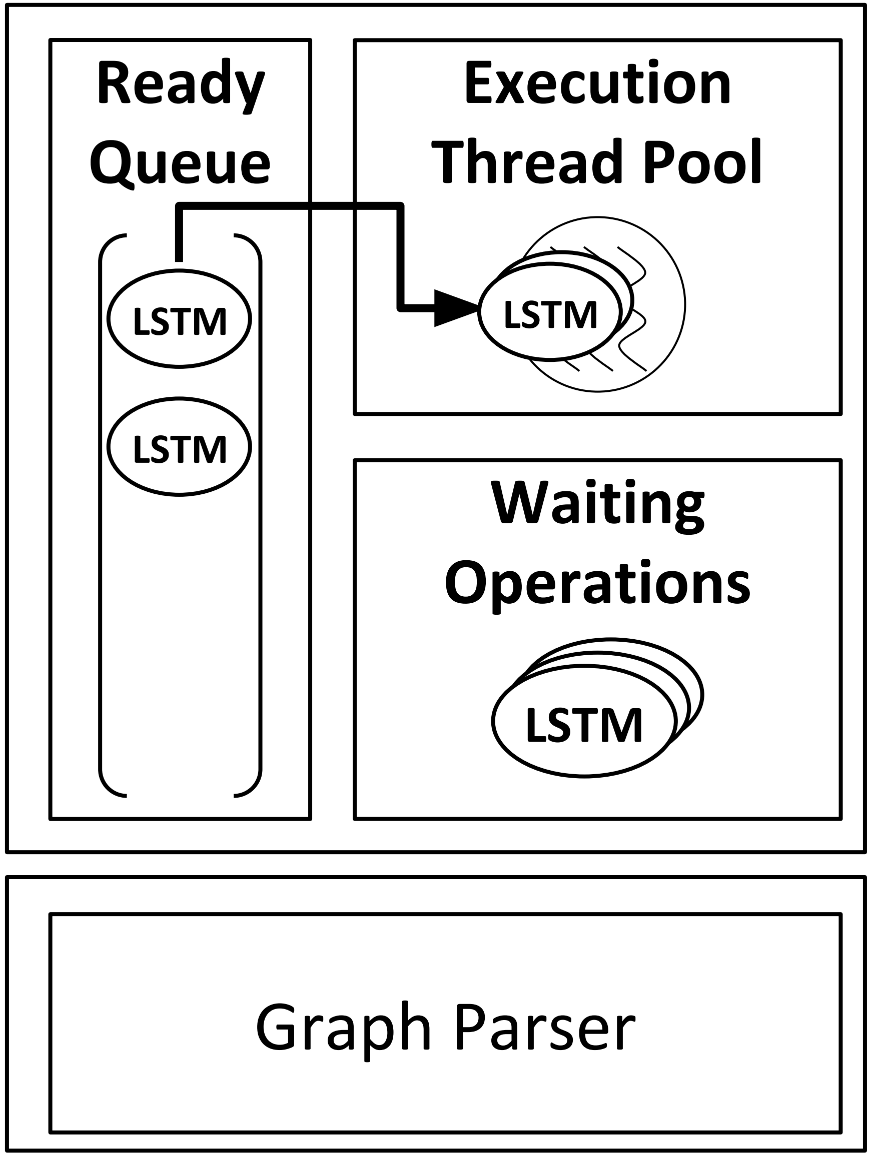

Current status
sentiment
LSTM
(A, movie)
(to, forget)

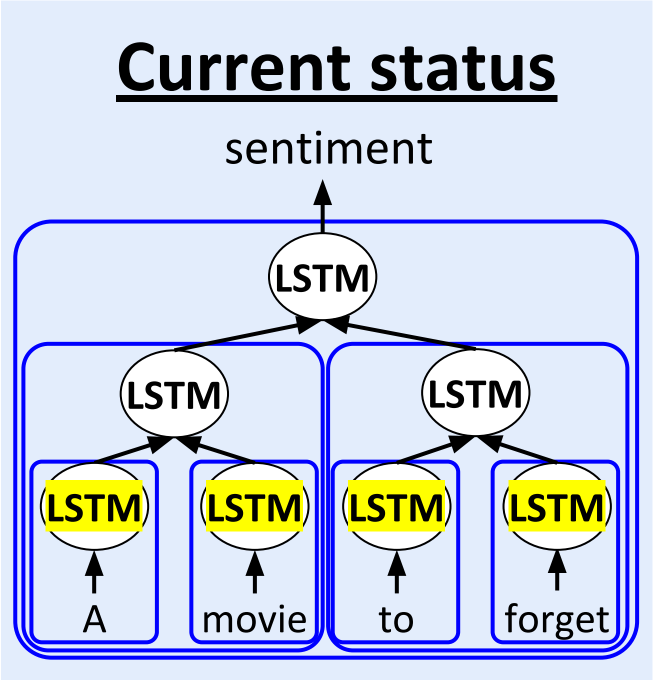

Ready
Queue
Execution
Thread Pool
Waiting
Operations
LSTM
Graph Parser
LSTM
LSTM


Current status
sentiment
LSTM
LSTM
LSTM
A
movie
to
forget


Ready
Queue
LSTM
LSTM
LSTM
LSTM
Execution
Thread Pool
LSTM
Waiting
Operations
LSTM
Graph Parser
LSTM
LSTM
LSTM
LSTM

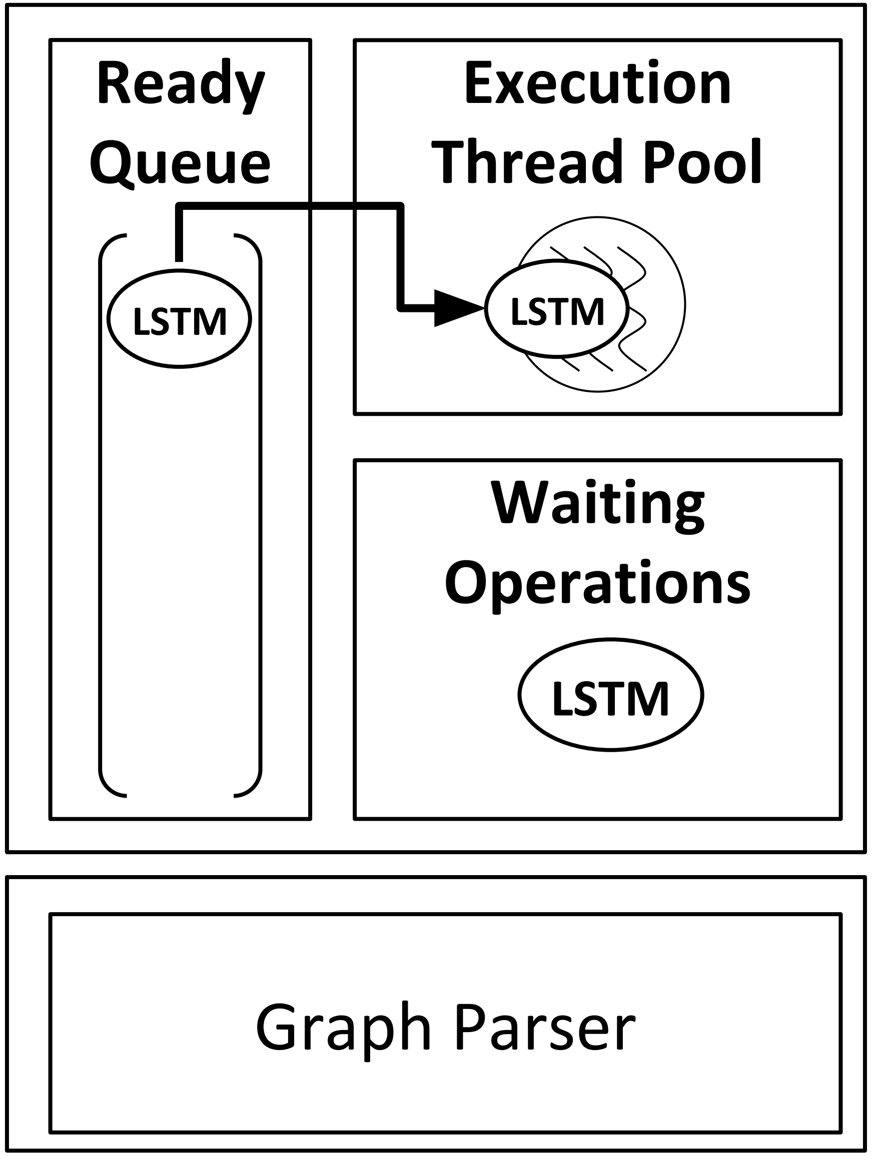

Current status
sentiment
LSTM
LSTM
LSTM
LSTM
LSTM
LSTM
LSTM
A
movie
to
forget

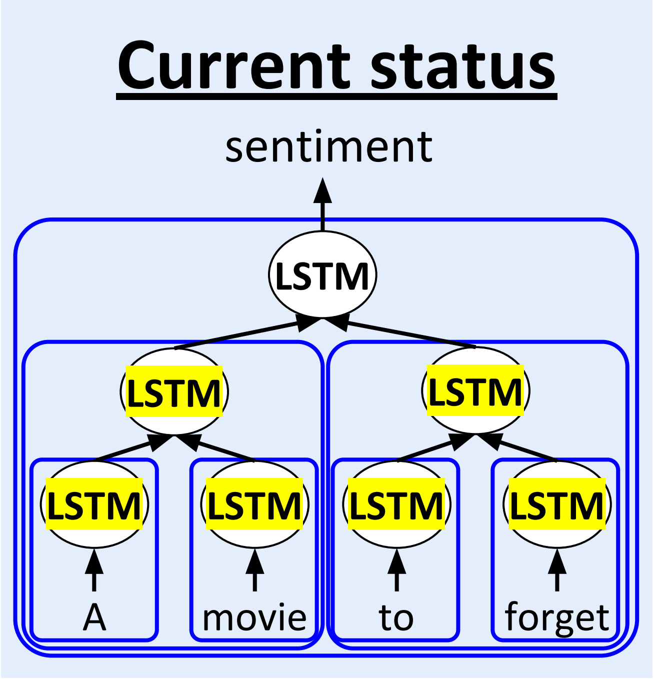

Ready Queue
LSTM
LSTM
Execution Thread Pool
LSTM
Waiting Operations
LSTM
Graph Parser

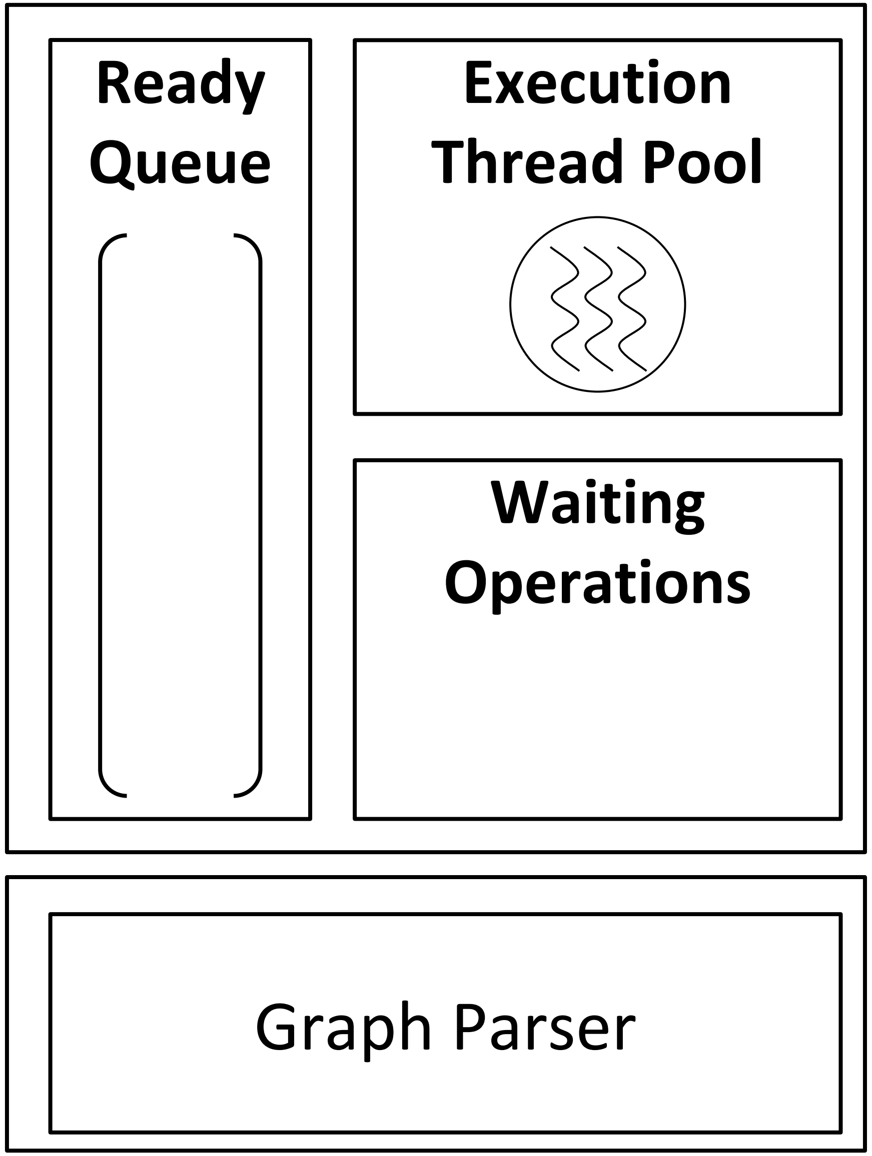

Current status
sentiment
LSTM
LSTM
LSTM
LSTM
LSTM
LSTM
LSTM
A
movie
to
forget

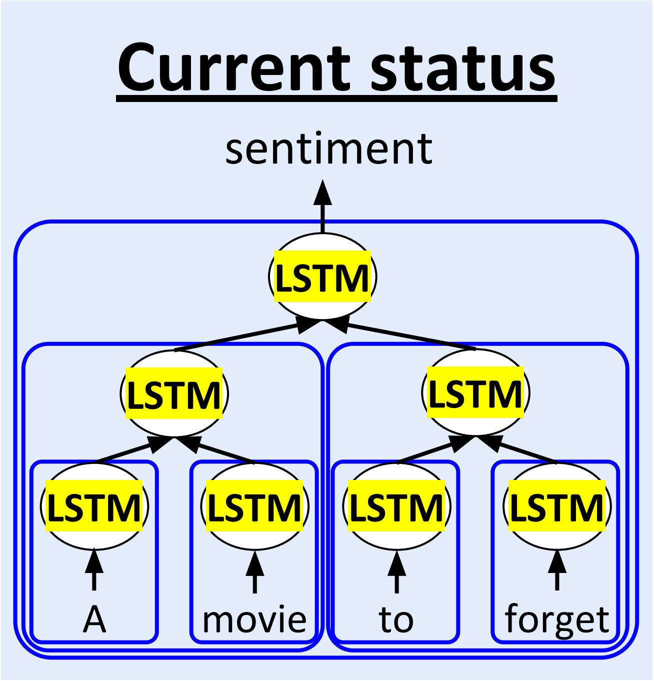
# Underlying System

Execution Model | Evaluation | Autodiff



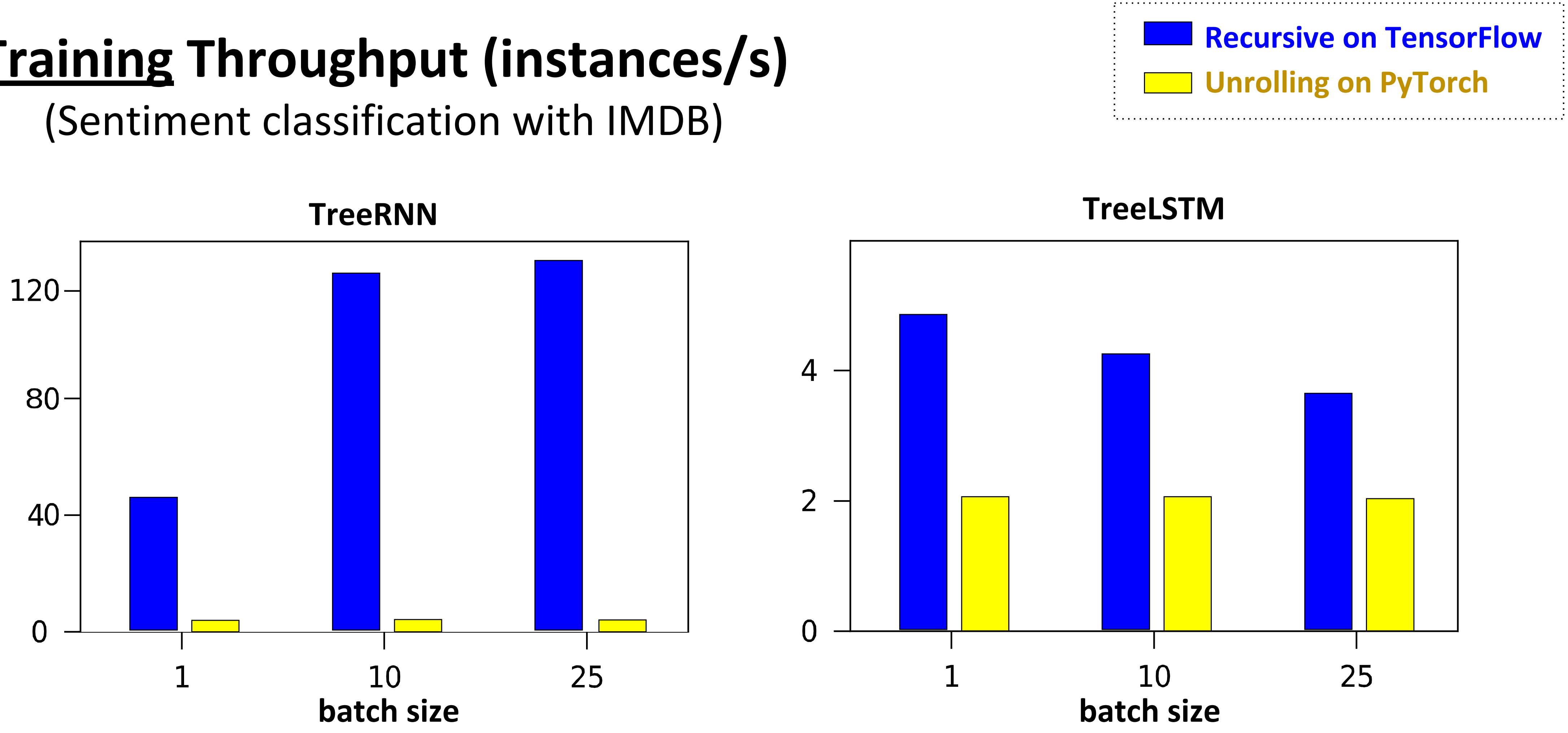




Ready Queue
Execution Thread Pool
Waiting Operations
Graph Parser


Current status
sentiment
LSTM
LSTM
LSTM
LSTM
LSTM
LSTM
LSTM
A
movie
to
forget

# Training Throughput (instances/s)

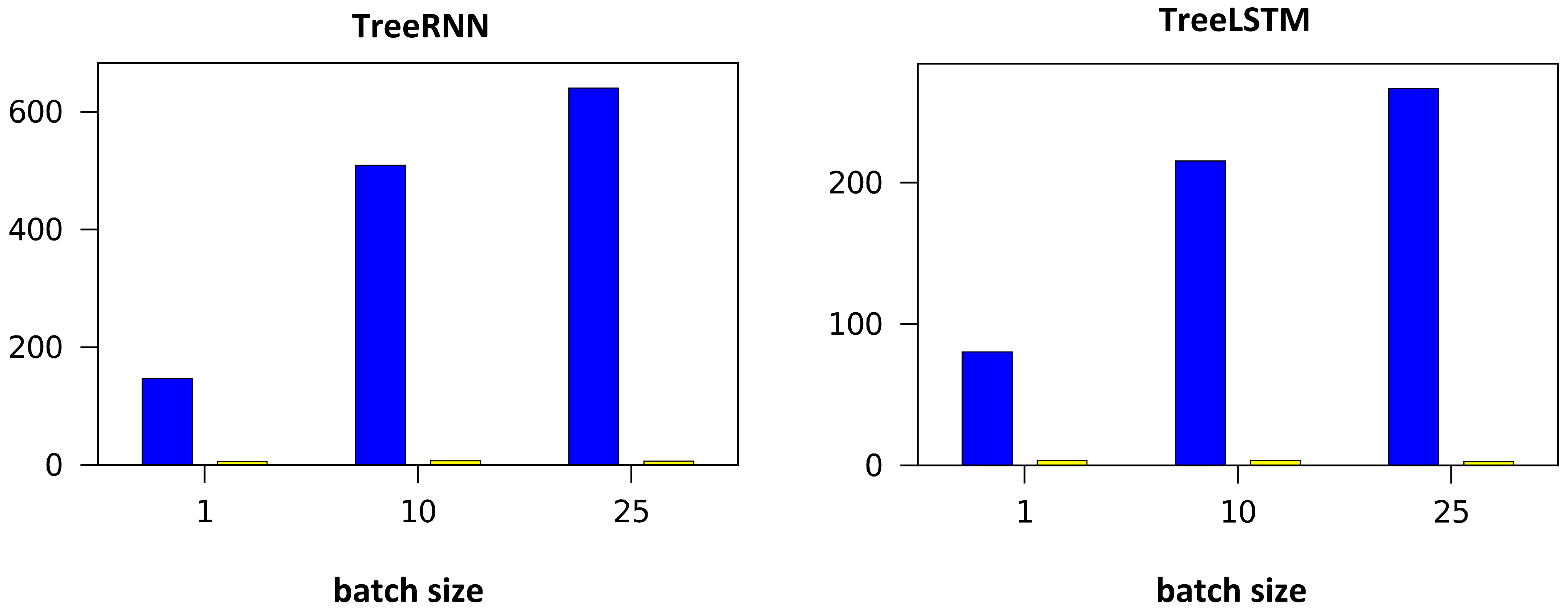
(Sentiment classification with IMDB)



# Inference Throughput (instances/s)

(Sentiment classification with IMDB)

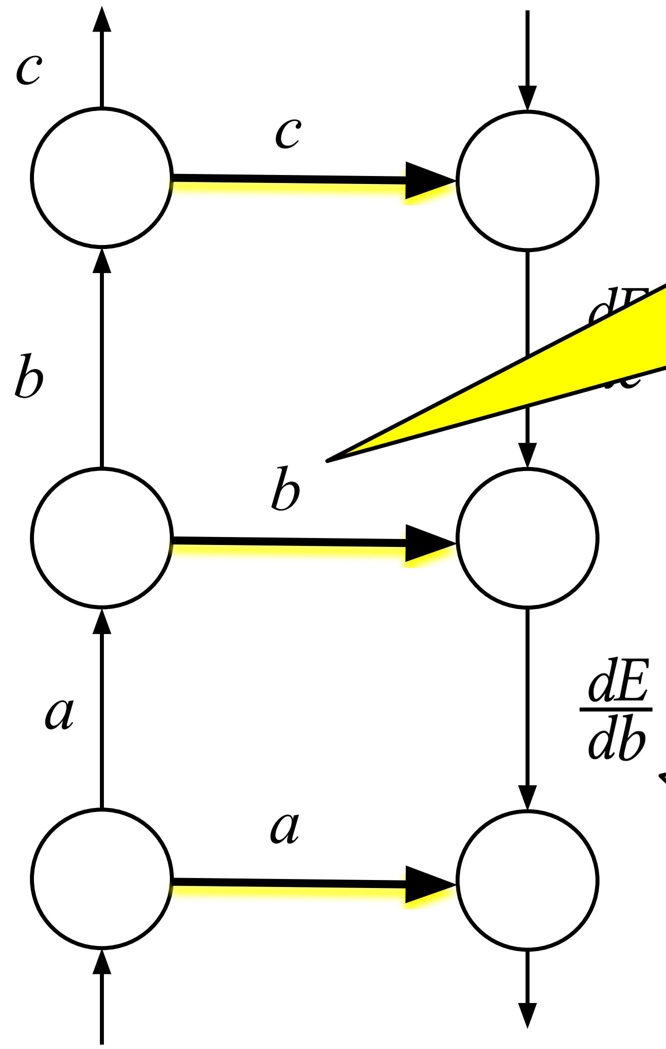
Recursive on TensorFlow
Unrolling on PyTorch



# Underlying System

Execution Model | Evaluation | **Autodiff**

“Feedforward”




“Backpropagation”

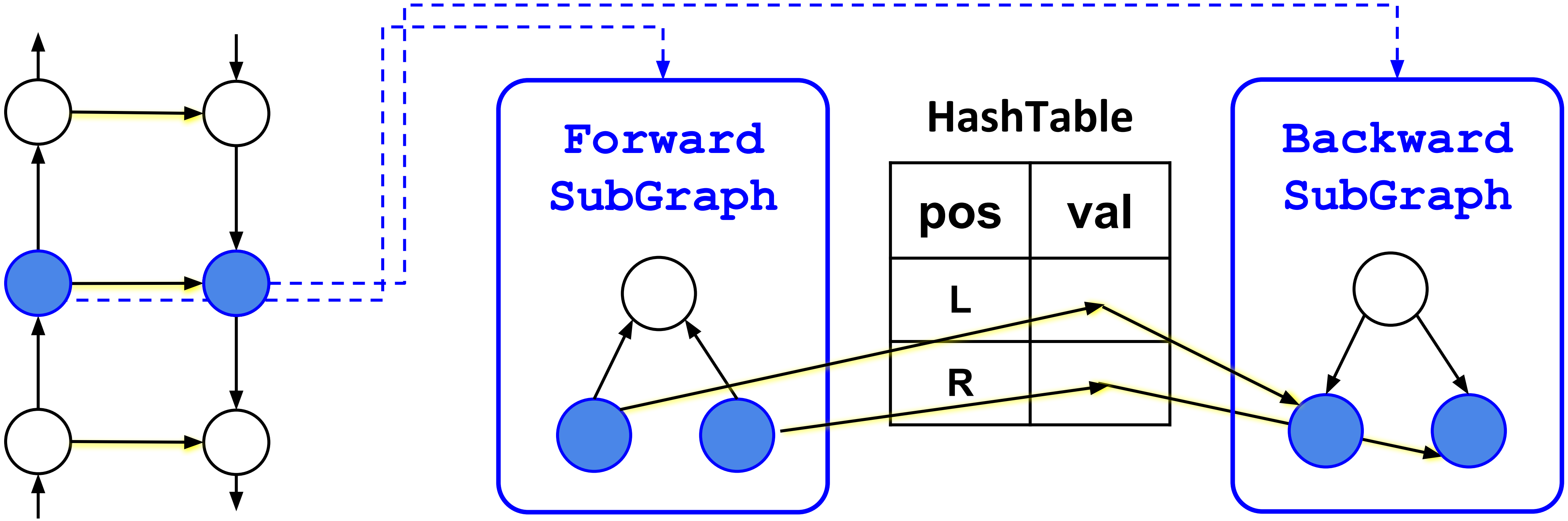
Forward computation values must be retained for backward computations

Framework builds back-propagation graph for user

**[Issue 1]** Building back-prop graph for `SubGraph`s and `InvokeOp`s

**[Issue 2]** Retaining the forward values with random execution order

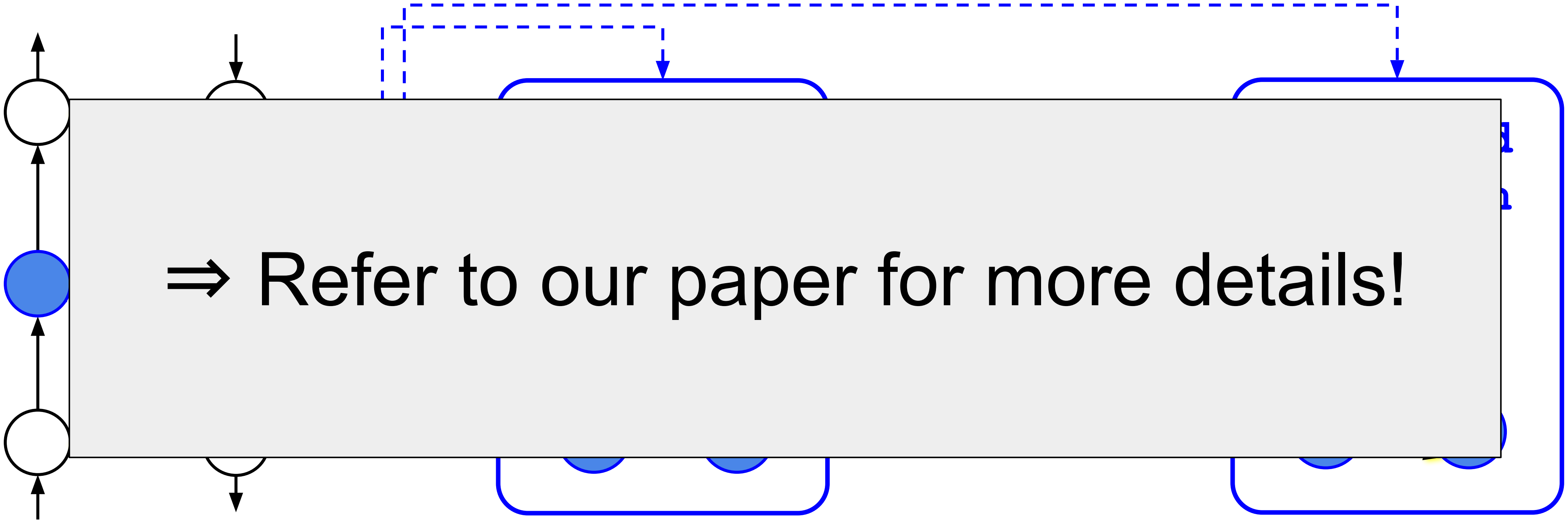
**[Solution]** ***Recursive*** backward SubGraph with ***hash tables***



**[Issue 1]** Building back-prop graph for `SubGraph`s and `InvokeOp`s

**[Issue 2]** Retaining the forward values with random execution order

**[Solution]** ***Recursive*** backward SubGraph with ***hash tables***



# Recursion for Symbolic DL Frameworks: Summary

- Improved expressiveness with abstractions SubGraphs and InvokeOps to program recursive neural networks

- Improved performance by recursively executing neural networks while exploiting parallelism

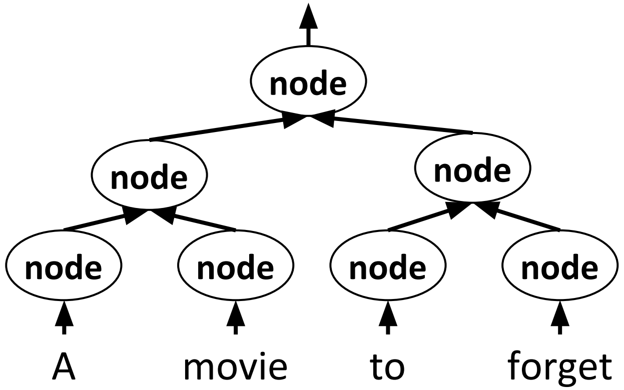
# Outline

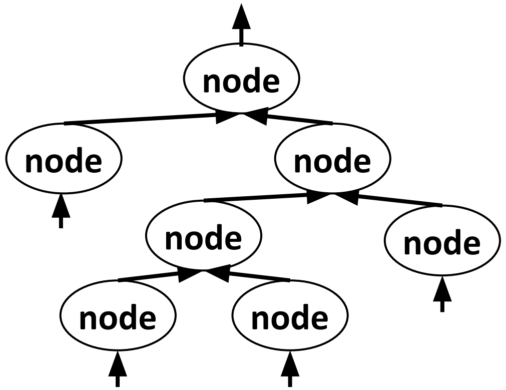
- JANUS
- How to handle Recursive Neural Networks?
  - Motivation
  - New Abstractions
  - Underlying System
  - **TreeLSTM on JANUS**
- On-going Works

# TreeLSTM on JANUS

Profiling | Gen. Graph | Run Graph

```
def TreeLSTM(node):
    if node.is_leaf:
        return LSTM(embed(node.word))
    else:
        lstate = TreeLSTM(node.left)
        rstate = TreeLSTM(node.right)
        return LSTM(lstate, rstate)

trees = parse(sentences)
for tree in trees:
    root_state = TreeLSTM(tree)
    sentiment = project(root_state)
```

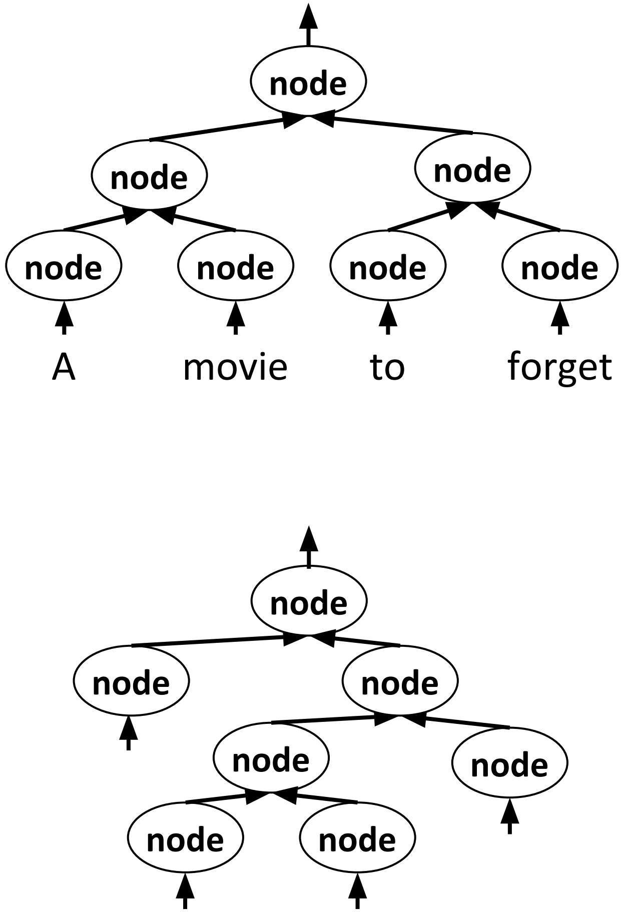


# TreeLSTM on JANUS

Profiling | Gen. Graph | Run Graph

```
def TreeLSTM(node):
    if node.is_leaf:
        return LSTM(embed(node.word))
    else:
        lstate = TreeLSTM(node.left)
        rstate = TreeLSTM(node.right)
        return LSTM(lstate, rstate)

trees = parse(sentences)
for tree in trees:
    root_state = TreeLSTM(tree)
    sentiment = project(root_state)
```



# TreeLSTM on JANUS

Profiling | Gen. Graph | Run Graph

python object

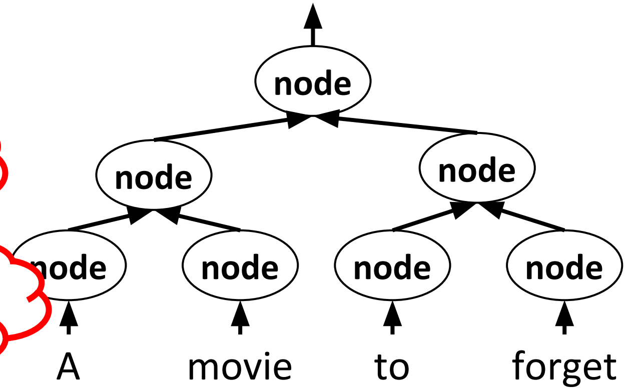
boolean

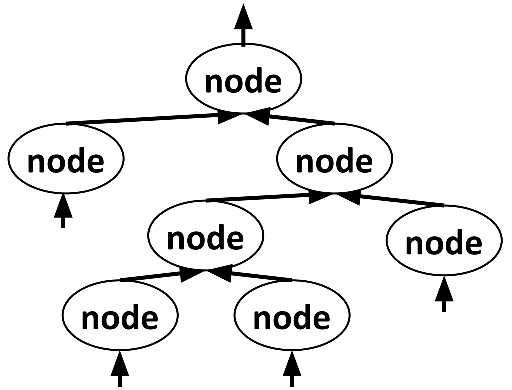
string

python object

python object

```
def TreeLSTM(node):
  if node.is_leaf:
    return LSTM(embed(node.word))
  else:
    lstate = TreeLSTM(node.left)
    rstate = TreeLSTM(node.right)
    return LSTM(lstate, rstate)

trees = parse(sentences)
for tree in trees:
  root_state = TreeLSTM(tree)
  sentiment = project(root_state)
```



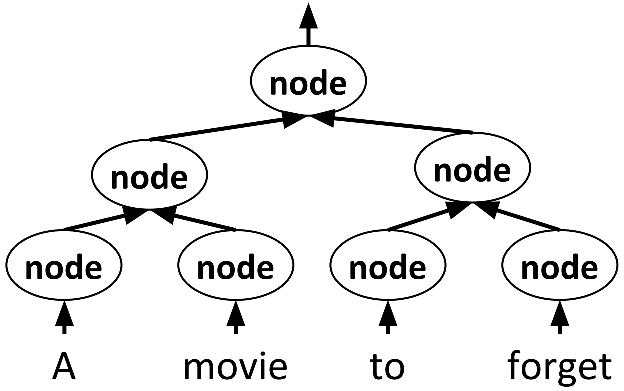
# TreeLSTM on JANUS

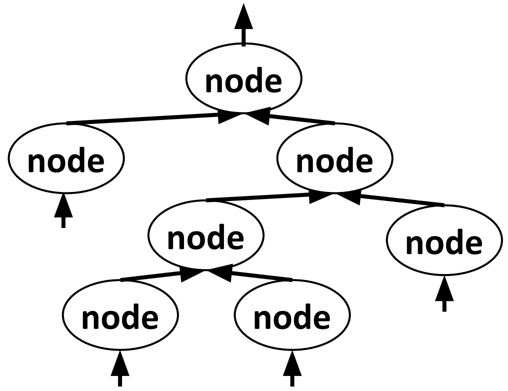


```
def TreeLSTM(node):
   if node.is_leaf:
      return LSTM(embed(n                    )
   else:
      lstate = TreeLSTM(no
      rstate = TreeLSTM(node.right)
      return LSTM(lstate, rstate)


trees = parse(sentences)
for tree in trees:
   root_state = TreeLSTM(tree)
   sentiment = project(root_state)
```

recursive call

recursive call

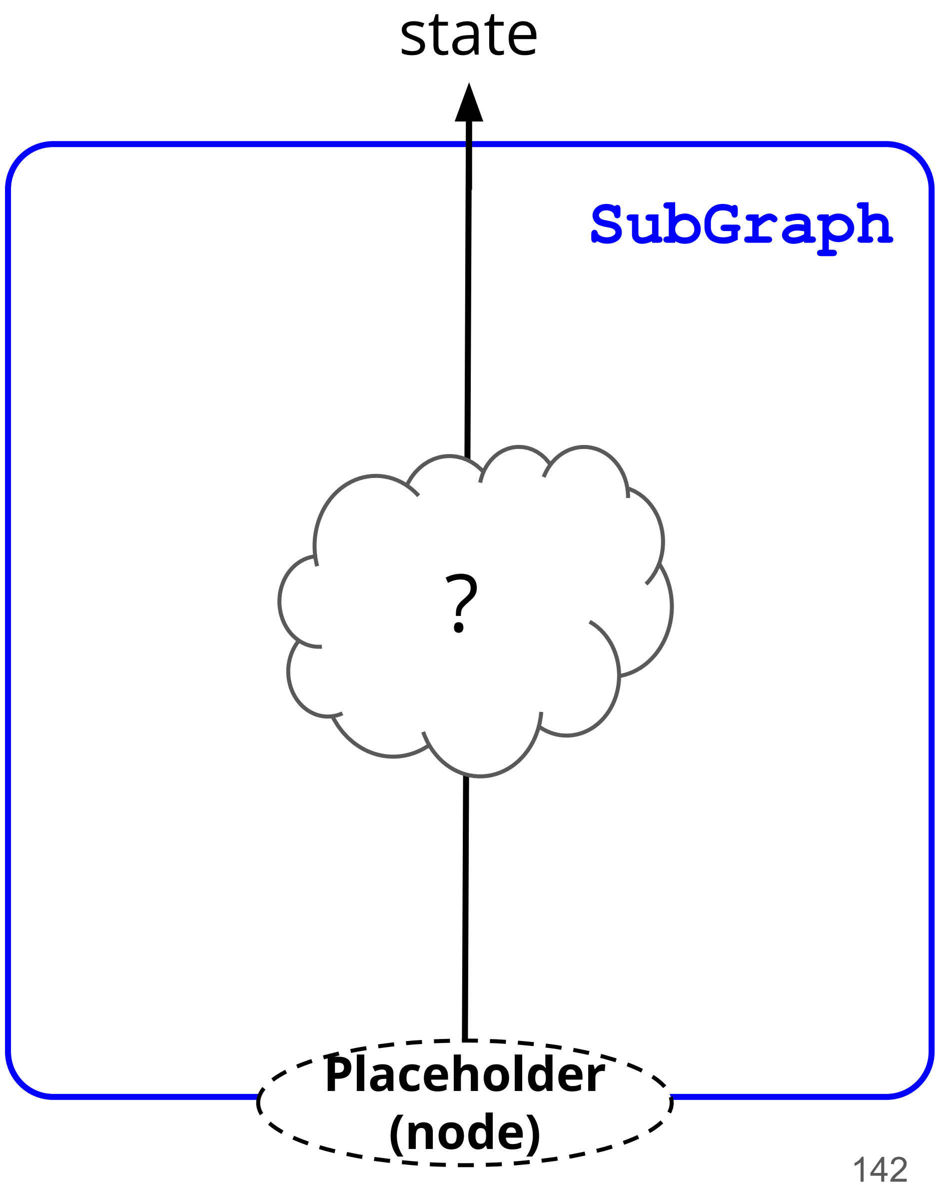


# TreeLSTM on JANUS

Profiling | **Gen. Graph** | Run Graph

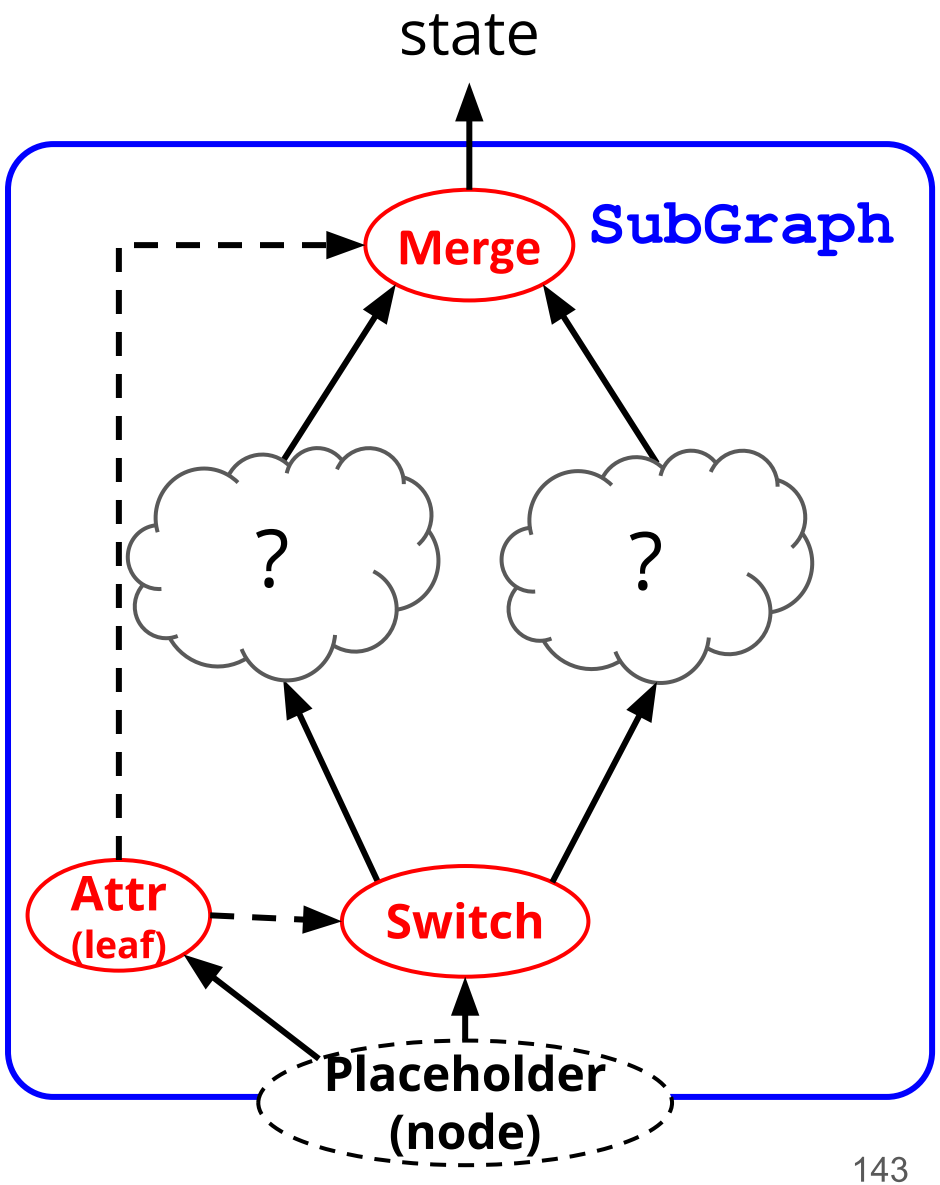
```
def TreeLSTM(node):
  if node.is_leaf:
    return LSTM(embed(node.word))
  else:
    lstate = TreeLSTM(node.left)
    rstate = TreeLSTM(node.right)
    return LSTM(lstate, rstate)

trees = parse(sentences)
for tree in trees:
  root_state = TreeLSTM(tree)
  sentiment = project(root_state)
```



# TreeLSTM on JANUS



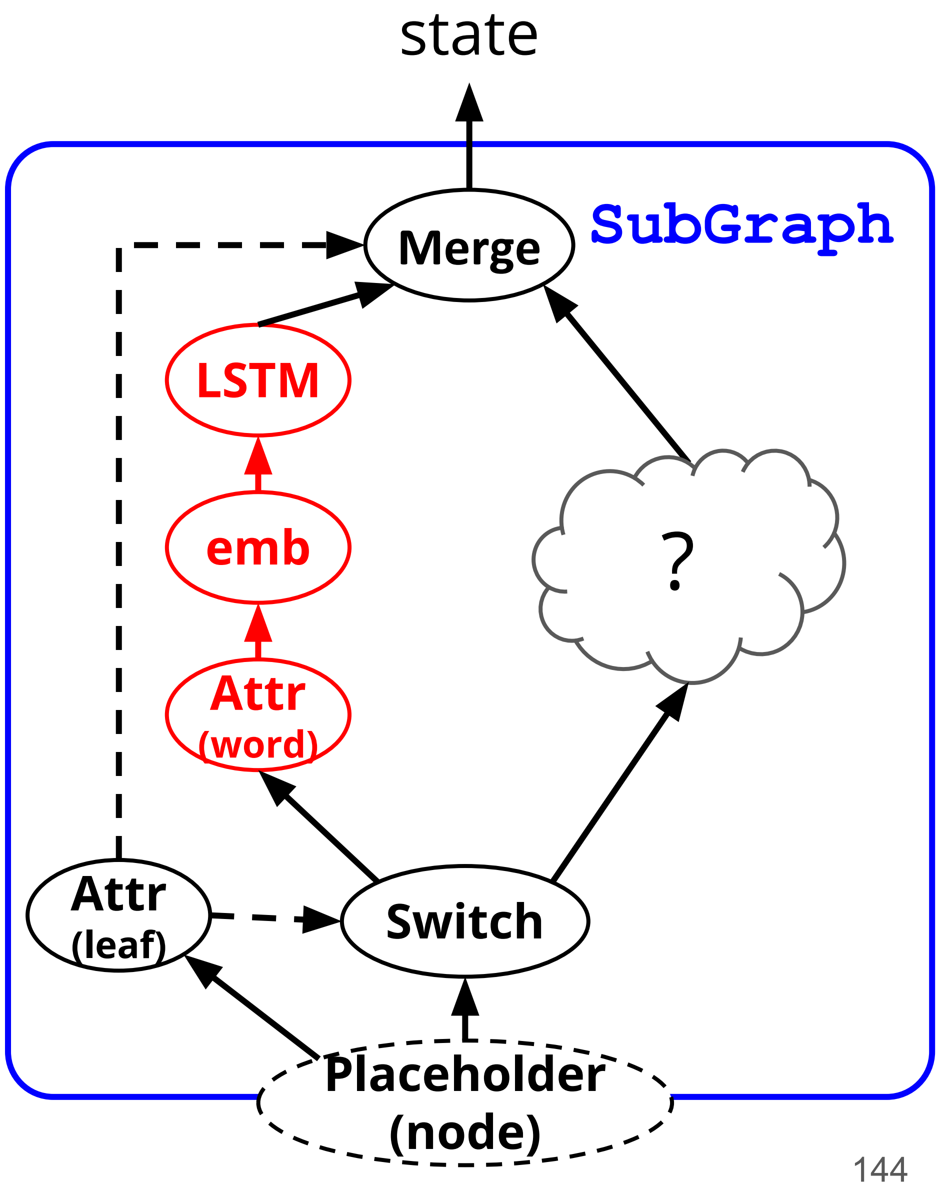
```
def TreeLSTM(node):
    if node.is_leaf:
        return LSTM(embed(node.word))
    else:
        lstate = TreeLSTM(node.left)
        rstate = TreeLSTM(node.right)
        return LSTM(lstate, rstate)

trees = parse(sentences)
for tree in trees:
    root_state = TreeLSTM(tree)
    sentiment = project(root_state)
```



# TreeLSTM on JANUS



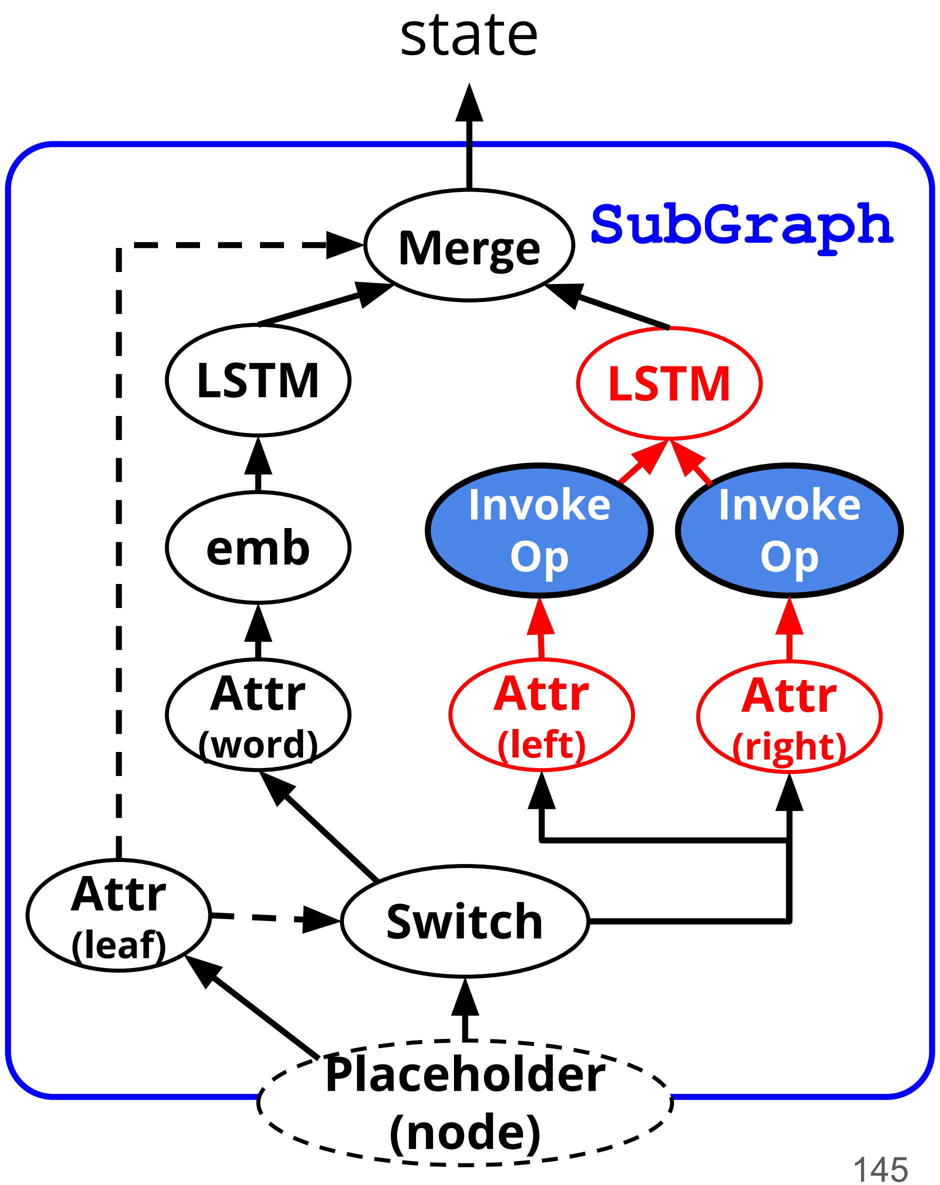
```
def TreeLSTM(node):
    if node.is_leaf:
        return LSTM(embed(node.word))
    else:
        lstate = TreeLSTM(node.left)
        rstate = TreeLSTM(node.right)
        return LSTM(lstate, rstate)

trees = parse(sentences)
for tree in trees:
    root_state = TreeLSTM(tree)
    sentiment = project(root_state)
```



# TreeLSTM on JANUS

Profiling | **Gen. Graph** | Run Graph

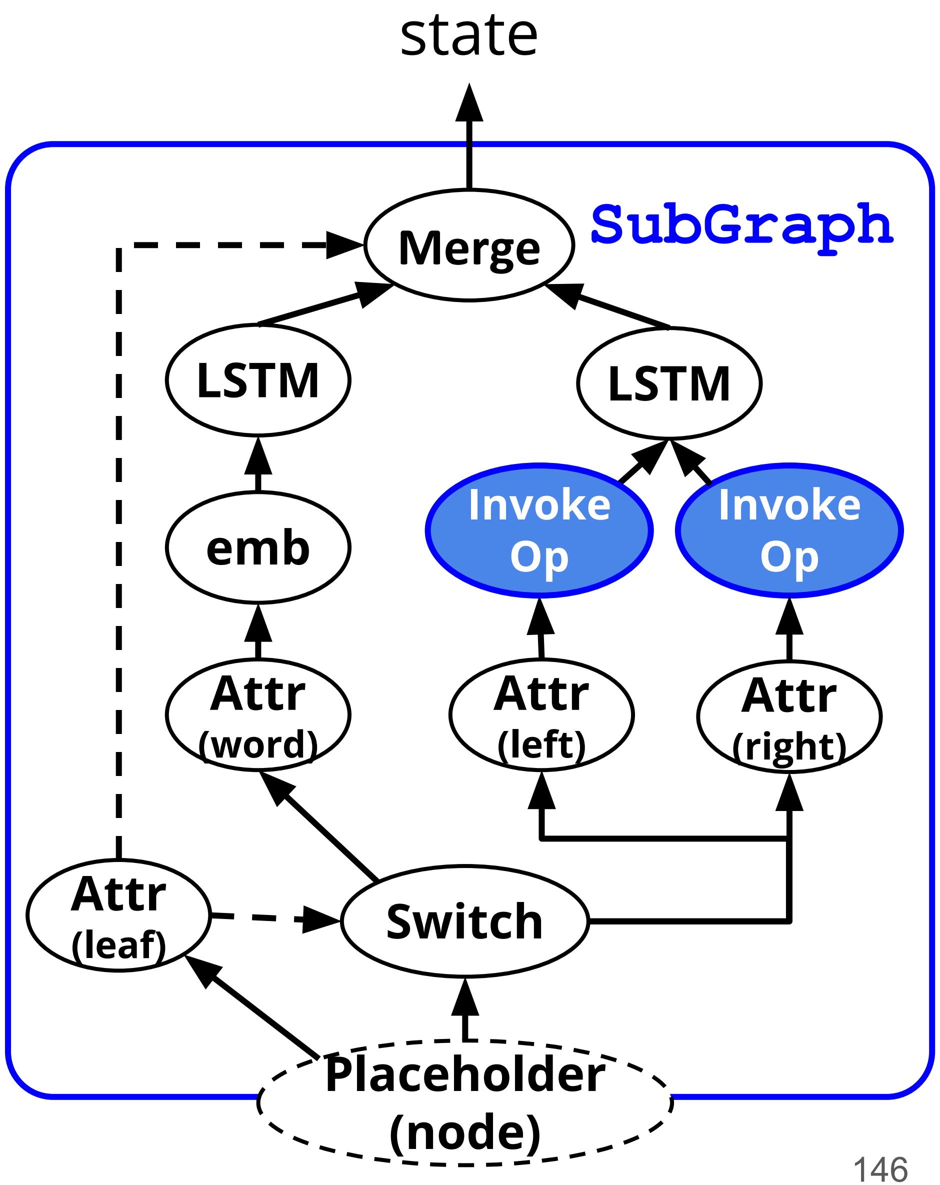
```
def TreeLSTM(node):
  if node.is_leaf:
    return LSTM(embed(node.word))
  else:
    lstate = TreeLSTM(node.left)
    rstate = TreeLSTM(node.right)
    return LSTM(lstate, rstate)

trees = parse(sentences)
for tree in trees:
  root_state = TreeLSTM(tree)
  sentiment = project(root_state)
```



# TreeLSTM on JANUS

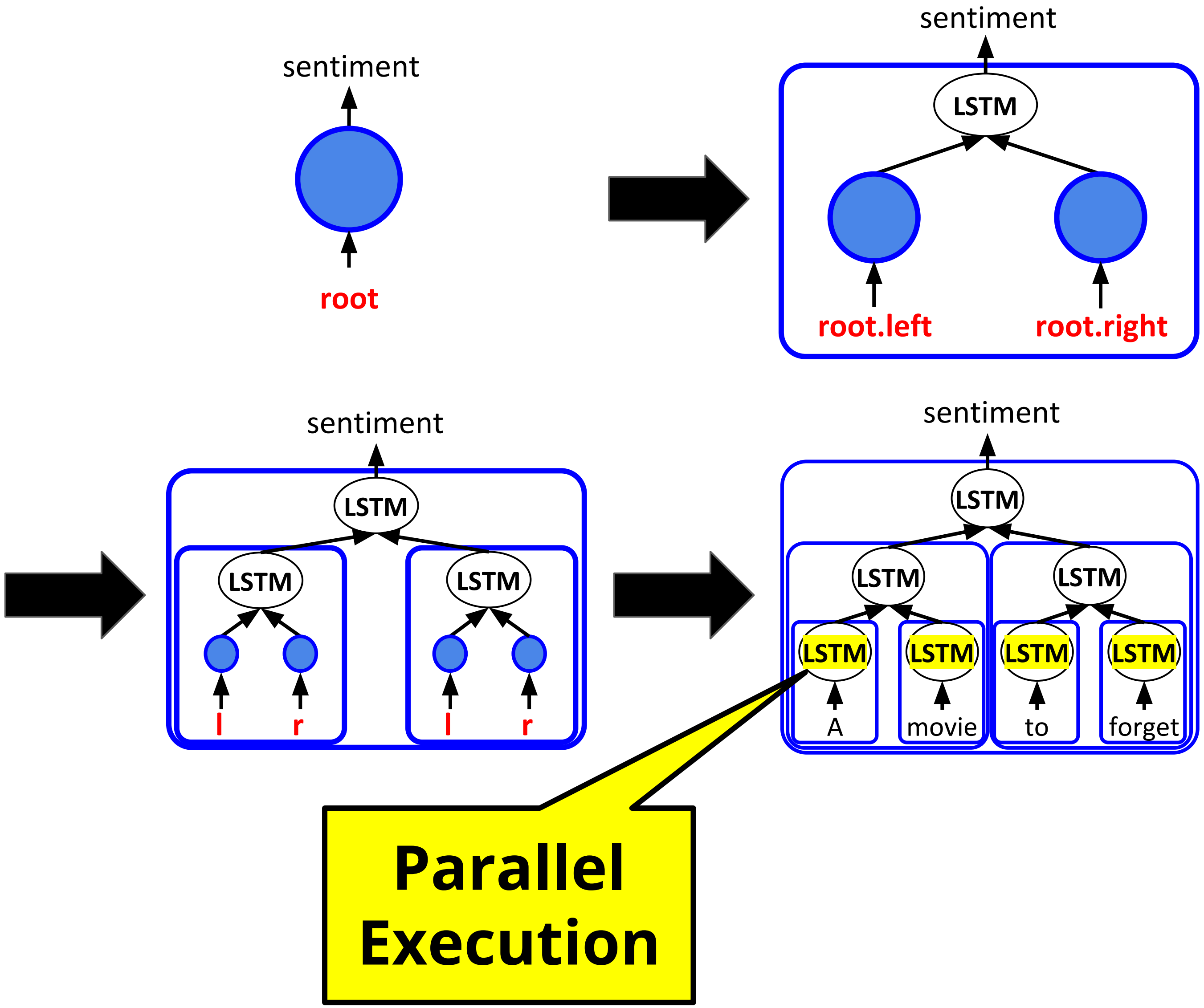
Profiling | Gen. Graph | **Run Graph**

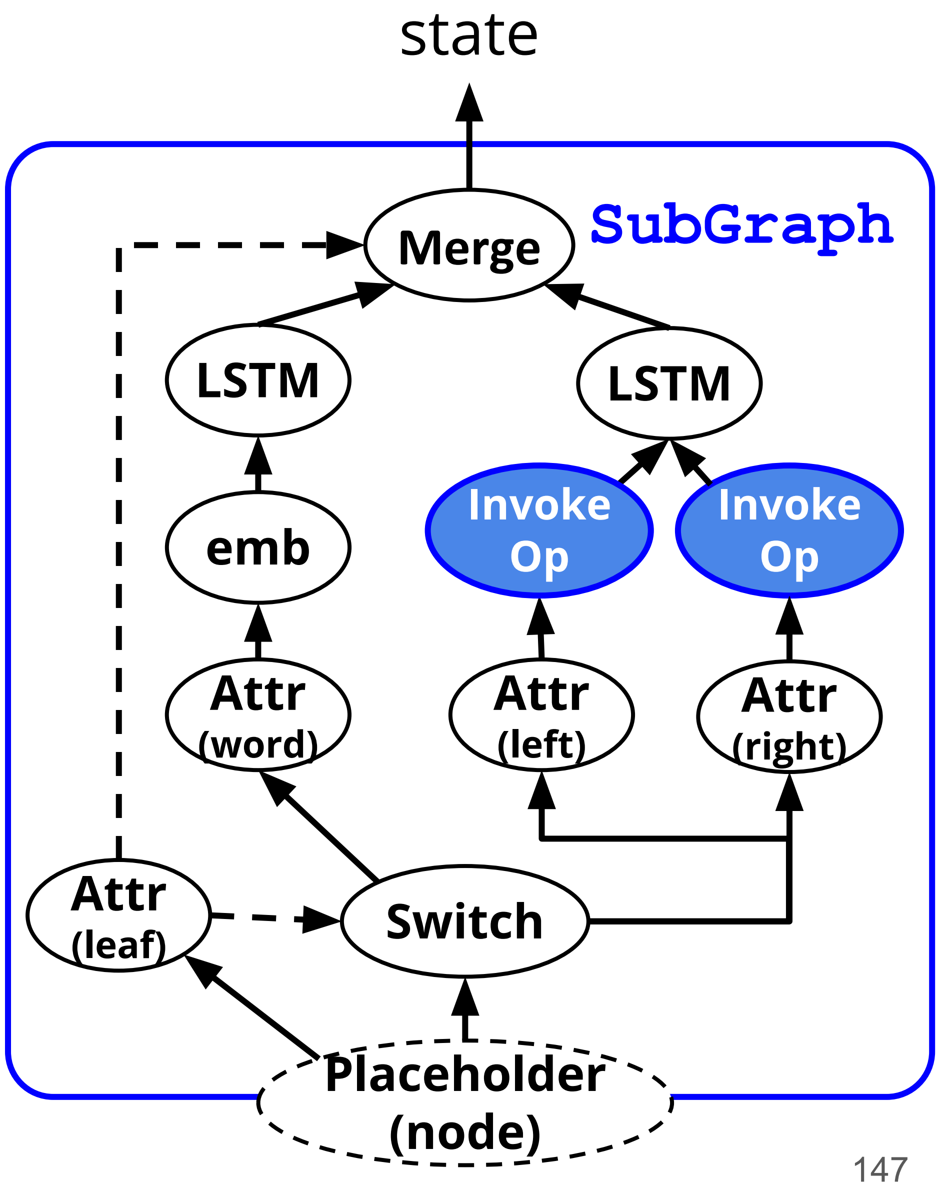
```
def TreeLSTM(node):
  if node.is_leaf:
    return LSTM(embed(node.word))
  else:
    lstate = TreeLSTM(node.left)
    rstate = TreeLSTM(node.right)
    return LSTM(lstate, rstate)

trees = parse(sentences)
for tree in trees:
  root_state = TreeLSTM(tree)
  sentiment = project(root_state)
```



# TreeLSTM on JANUS

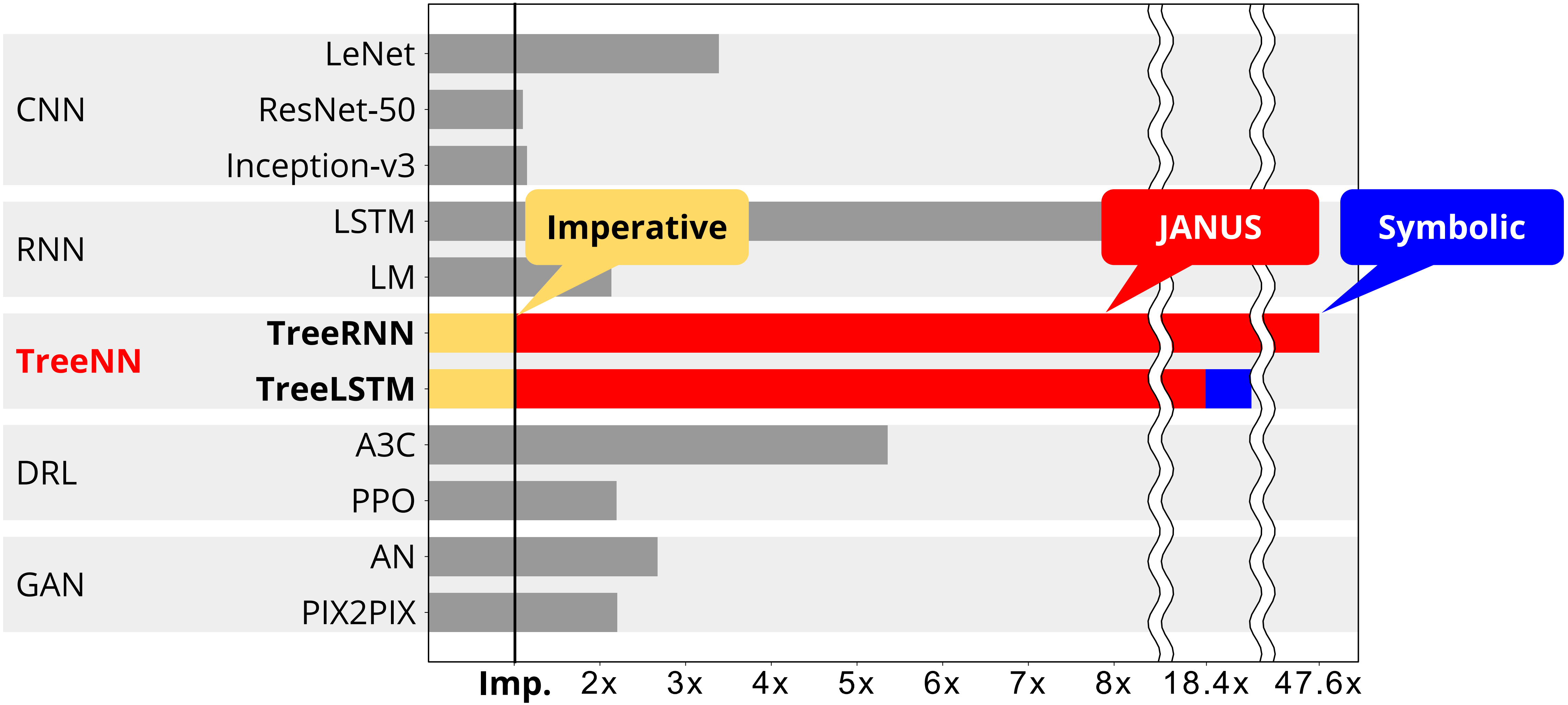
Profiling | Gen. Graph | **Run Graph**



# TreeLSTM on JANUS: Normalized Training Throughput

Single Machine



# Outline

- JANUS
- How to handle Recursive Neural Networks?
- **On-going Works**

# On-Going Works

- Open-Source
  - On top of TensorFlow 2.0
  - Collaboration with Google Brain TensorFlow AutoGraph team

- Improving JANUS
  - Transparent and fast profiler with un-modified Python interpreter
  - Integrate more powerful backend graph executors: TVM, XLA, ...

- Other Works
  - Parallax (EuroSys' 19): Sparsity-aware distributed training of DL models
  - Optimizing hyper-parameter optimization jobs for DL

# Thank You!