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

Deep Learning (DL) Frameworks

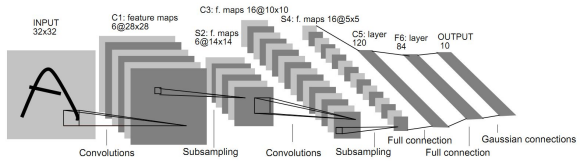
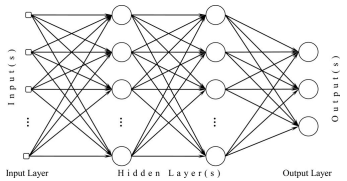
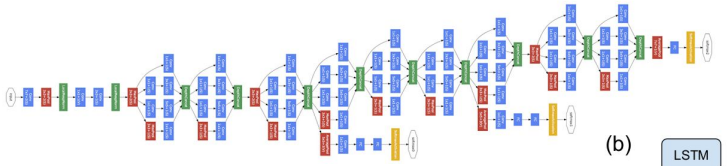
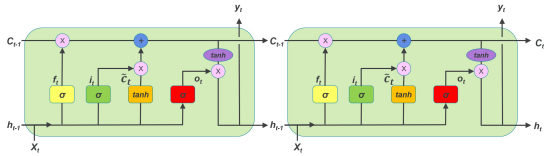
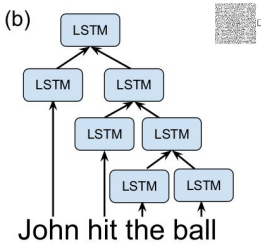


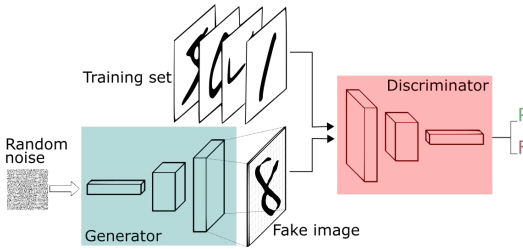
Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

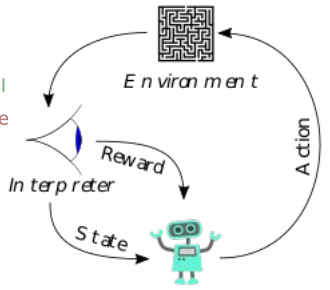












Images From:
http://www.mdpi.com/
https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/
Going Deeper with Convolutions, 2014, https://towardsdatascience.com/learn-how-recurrent-neural-networks-work-84e975feaaf7
Short-Term Load Forecasting Using EMD-LSTM Neural 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

Define **Execute**

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

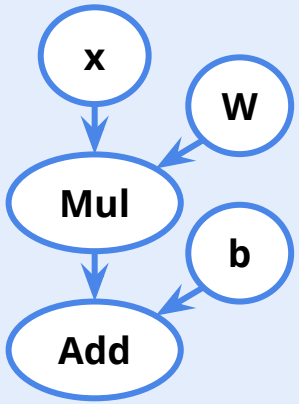
Two Paradigms

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

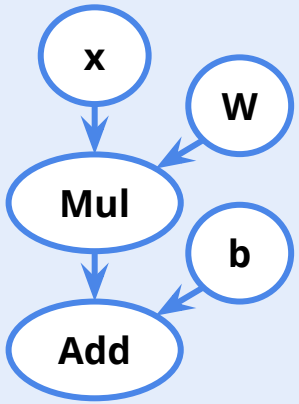
Symbolic DL Frameworks

✓ 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 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

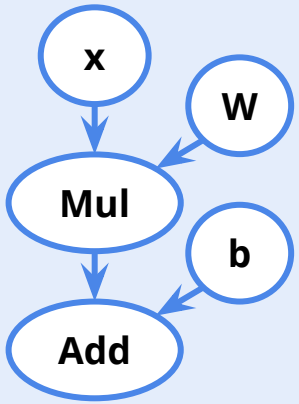
Two Paradigms

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 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
gluon
Chainer

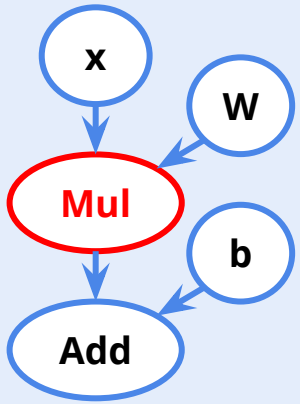
Two Paradigms

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

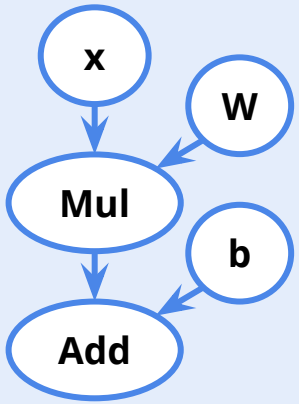
Symbolic DL Frameworks

✓ 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 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

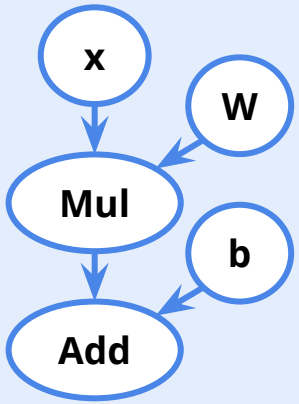
Symbolic DL Frameworks

✓ 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 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

Pros & Cons

	Symbolic DL Frameworks	Imperative DL Frameworks

Pros	+ Easy to Optimize
	  + Compiler Optimization
	  + Parallel Execution of Operations
	  + 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**
	  + **Compiler Optimization**
	  + **Parallel Execution of Operations**
	  + **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

Pros

+ **Easy to Optimize**
  + **Compiler Optimization**
  + **Parallel Execution of Operations**
  + **Deploy on GPU, Cluster, Mobile,...**

Cons

- Decoupled View:
  Hard to Program & Debug

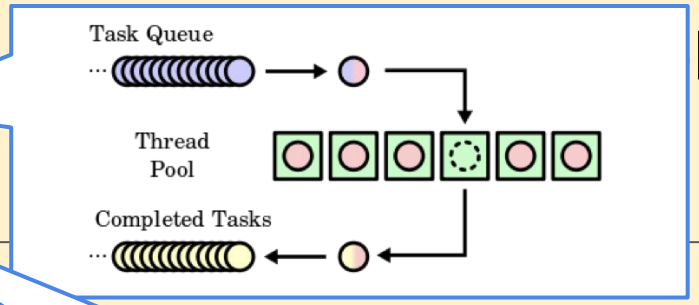
eworks

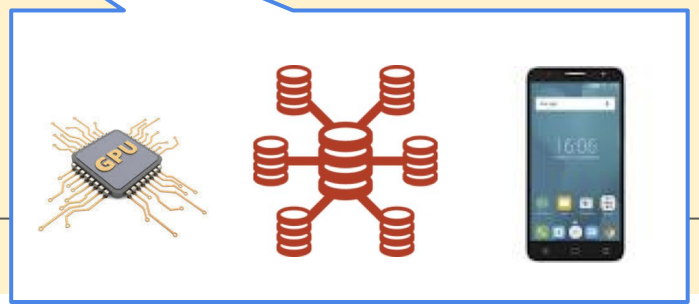
+ Direct Execution:

bug

-







Pros & Cons

Performance | Programmability

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

Pros & Cons

	Symbolic DL Frameworks	Imperative DL Frameworks

Pros	+ Easy to Optimize
	  + Compiler Optimization
	  + Parallel Execution of Operations
	  + 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
	  + Compiler Optimization
	  + Parallel Execution of Operations
	  + 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**
	  + **Compiler Optimization**
	  + **Parallel Execution of Operations**
	  + **Deploy on GPU, Cluster, Mobile,...**
		+ **Direct Execution:**
		Easy to Program & Debug

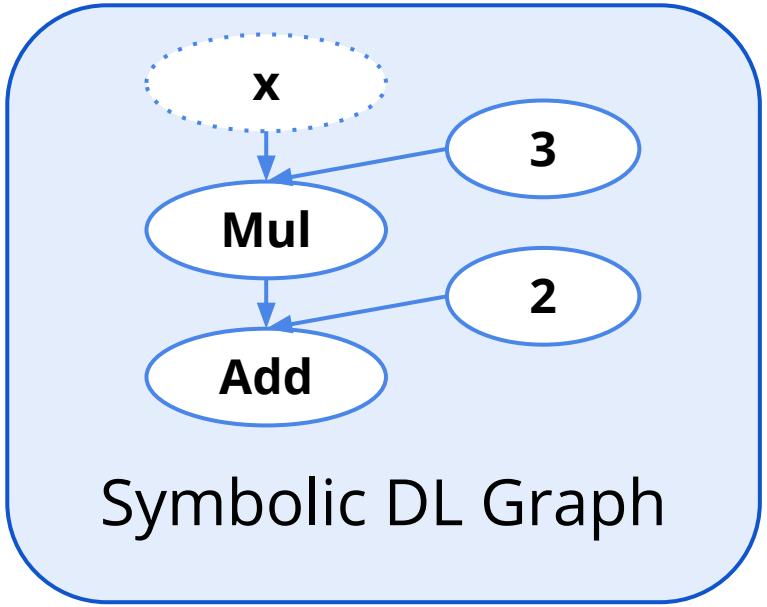
Cons	- Decoupled View: Hard to Program & Debug	- Hard to Optimize

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

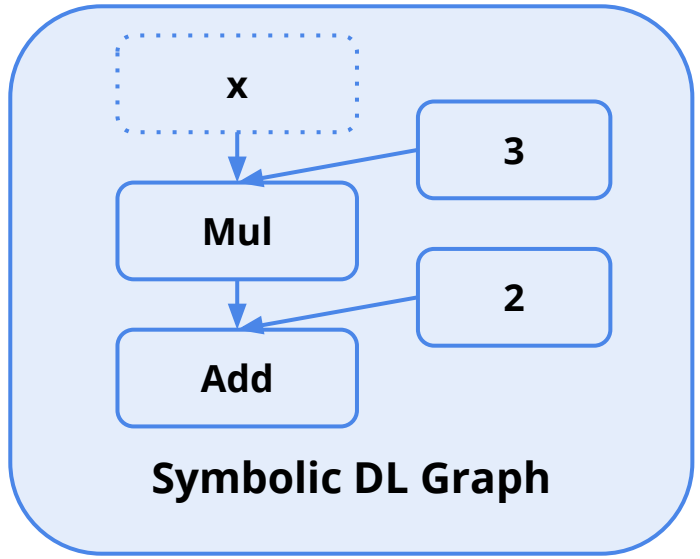
- Approach
- **Challenges**
- JANUS
- Evaluation

Challenges in Graph Conversion

Imperative DL Program

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

Transparent Conversion



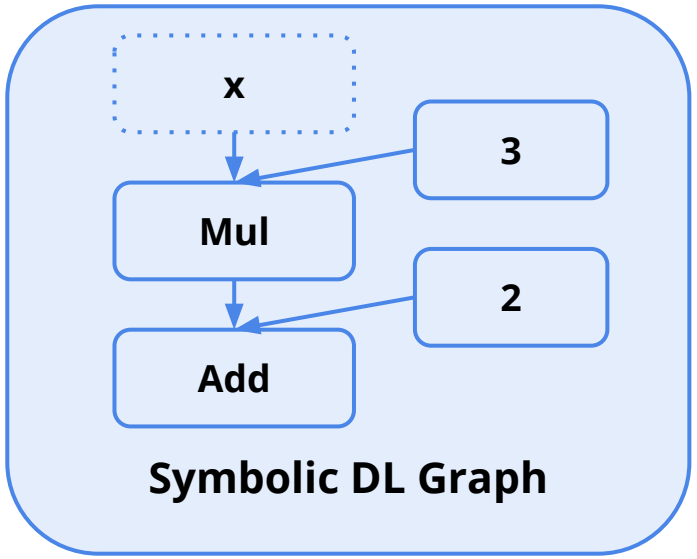
Challenges in Graph Conversion

De-facto Standard Language for DL Programming

Imperative Python DL Program

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

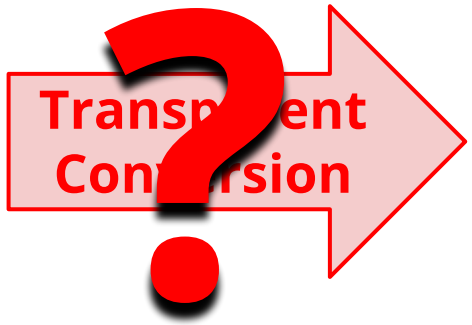
Transparent Conversion

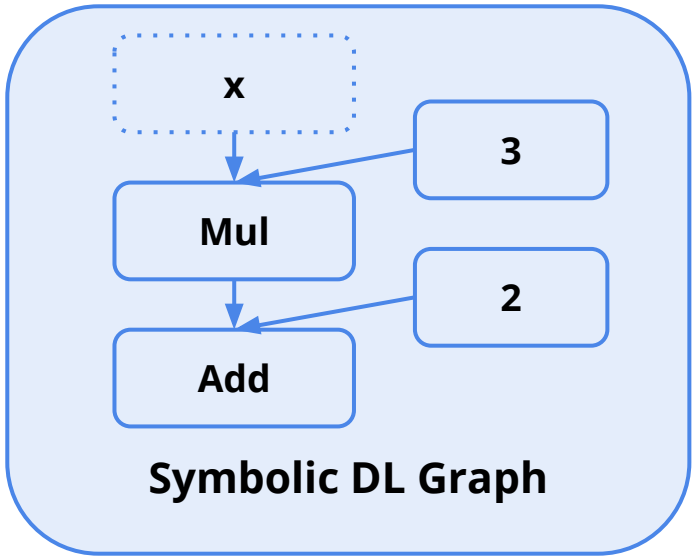


Challenges in Graph Conversion

Imperative Python DL Program

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



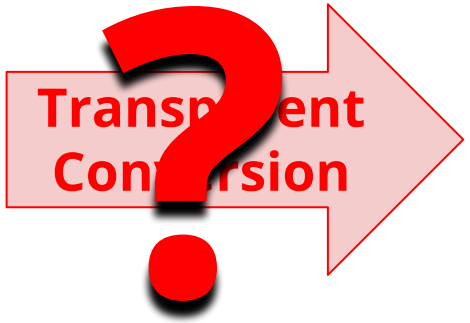


Discrepancy between Python Programs and DL Graphs

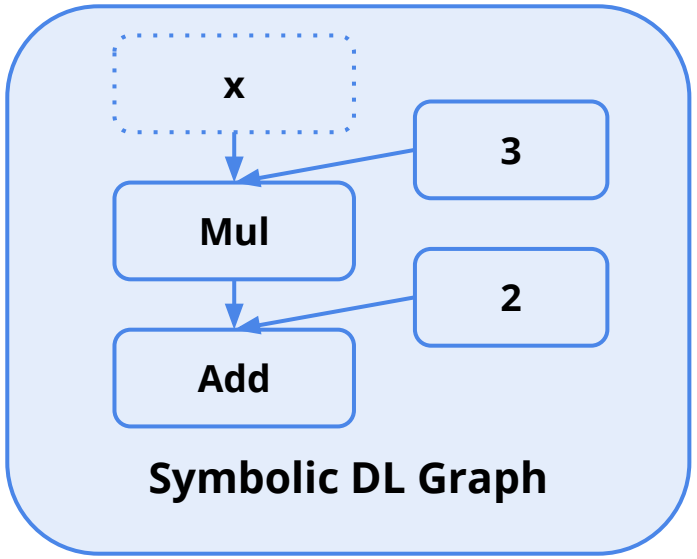
"Dynamic"

Imperative Python DL Program

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



"Static"



Discrepancy between Python Programs and DL Graphs

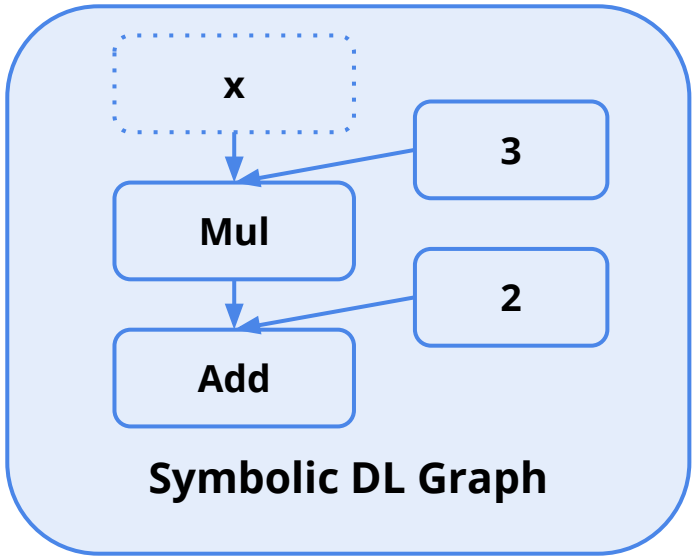
"Dynamic"

Imperative Python DL Program

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

Transparent Conversion ?

"Static"



Characteristics

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

Discrepancy between Python Programs and DL Graphs

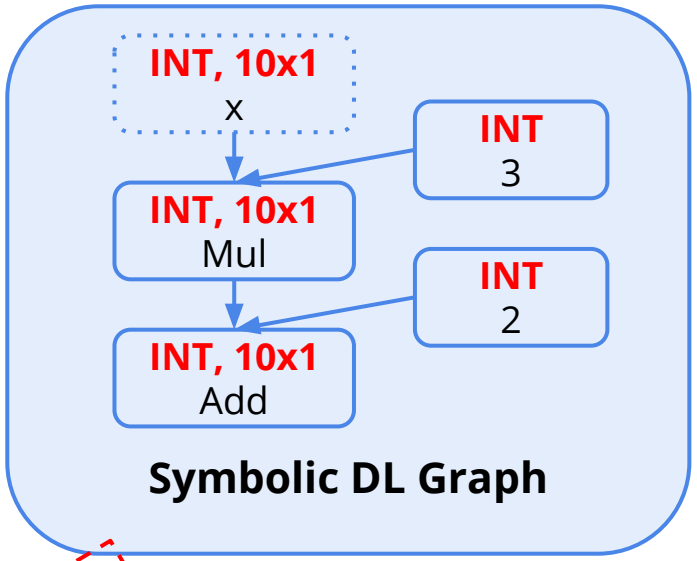
"Dynamic"

Imperative Python DL Program

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

Transparent Conversion ?

"Static"



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

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

Discrepancy between Python Programs and DL Graphs

"Dynamic"

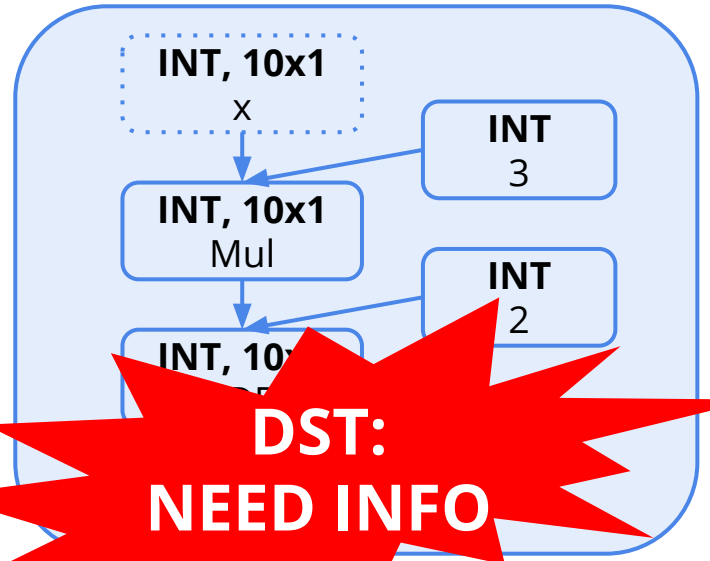
Imperative Python DL Program

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

SRC: NO INFO

Transparent Conversion

"Static"



DST: NEED INFO

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

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

Example: Recurrent Neural Network (RNN)

```
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

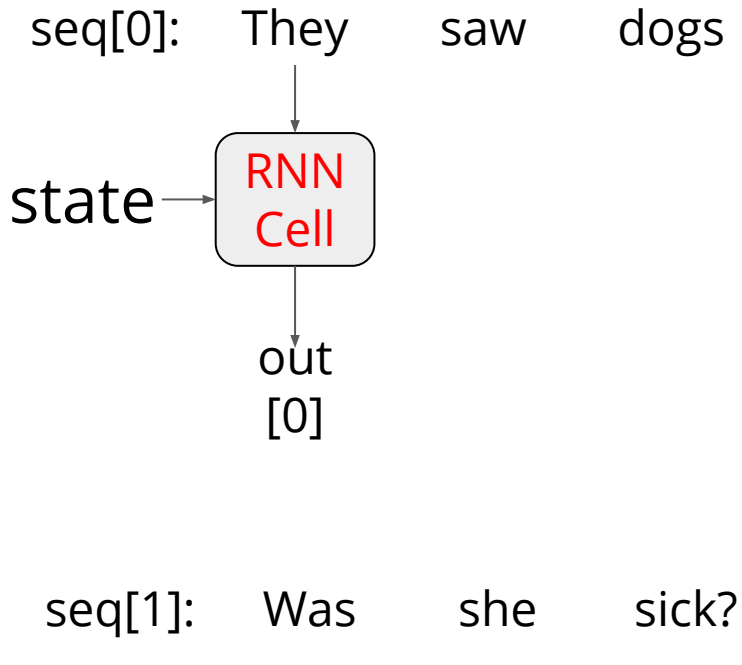
seq[1]: Was she sick?

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))
```

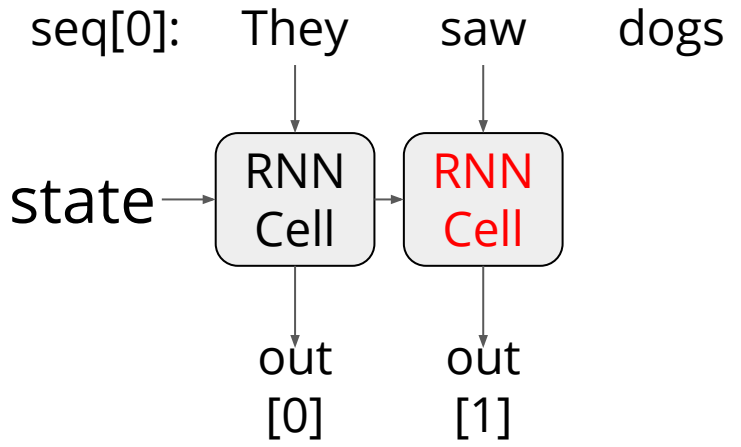


RNN Example

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))
```

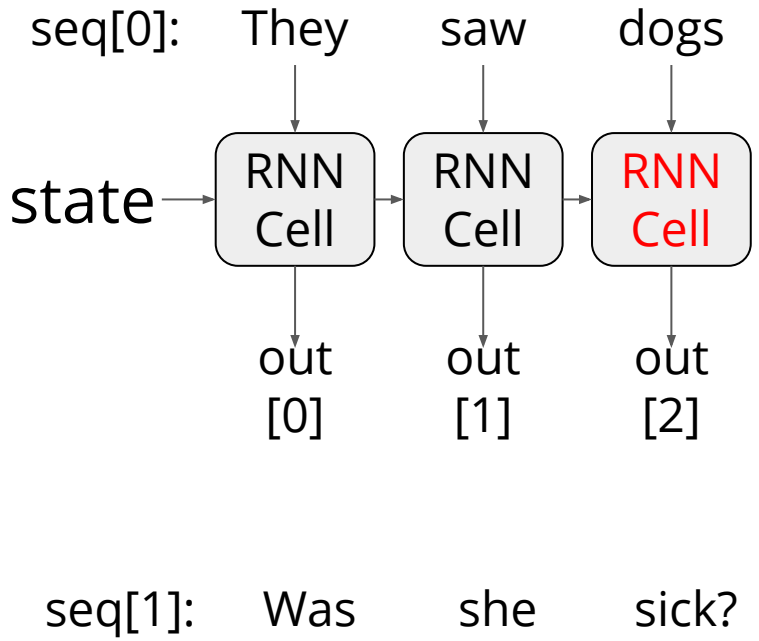


RNN Example

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)

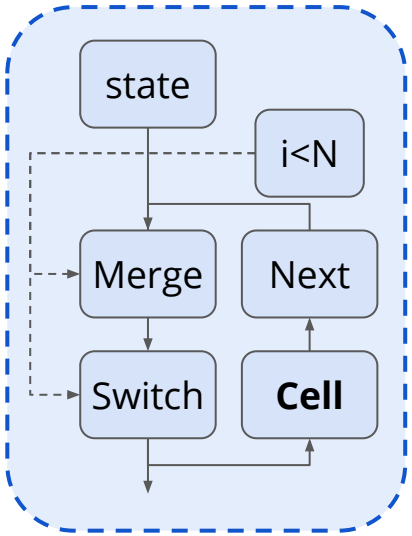
for sequence in sequences:
  optimize(lambda: model(sequence))
```

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))
```

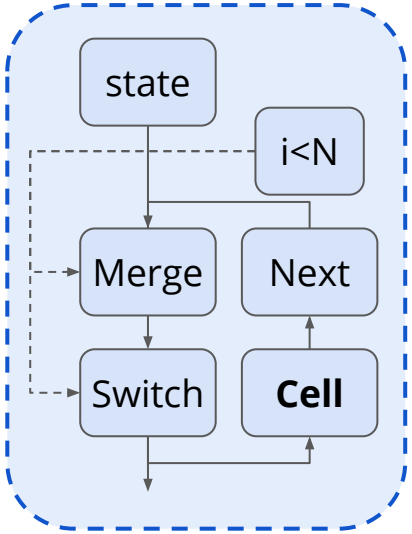


RNN Example

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))
```



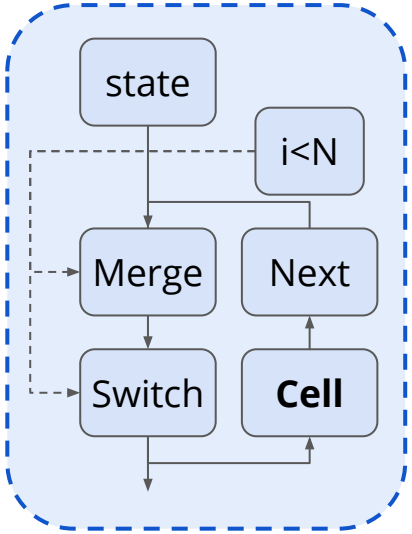
- Correct ☺

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))
```



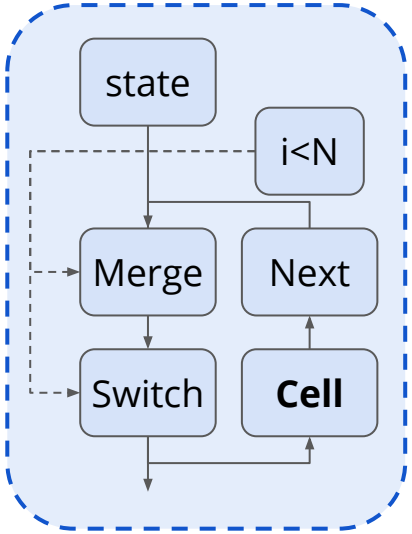
- Correct ☺
- Slow ☹

RNN Example

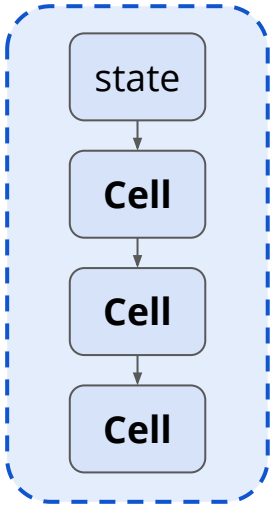
Dynamic Control Flow | Dynamic Types | Impure Function

```
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 ☹



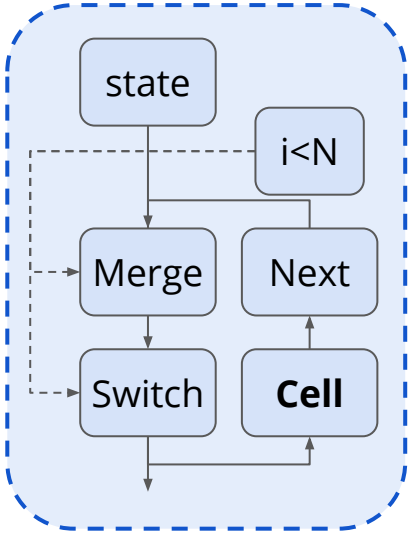
- Fast ☺

RNN Example

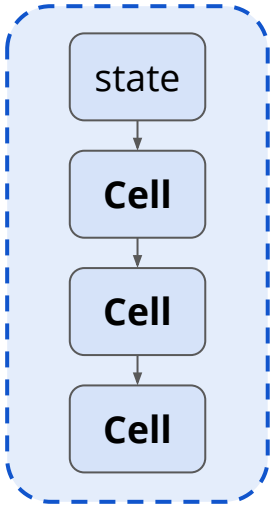
Dynamic Control Flow | Dynamic Types | Impure Function

```
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 ☹

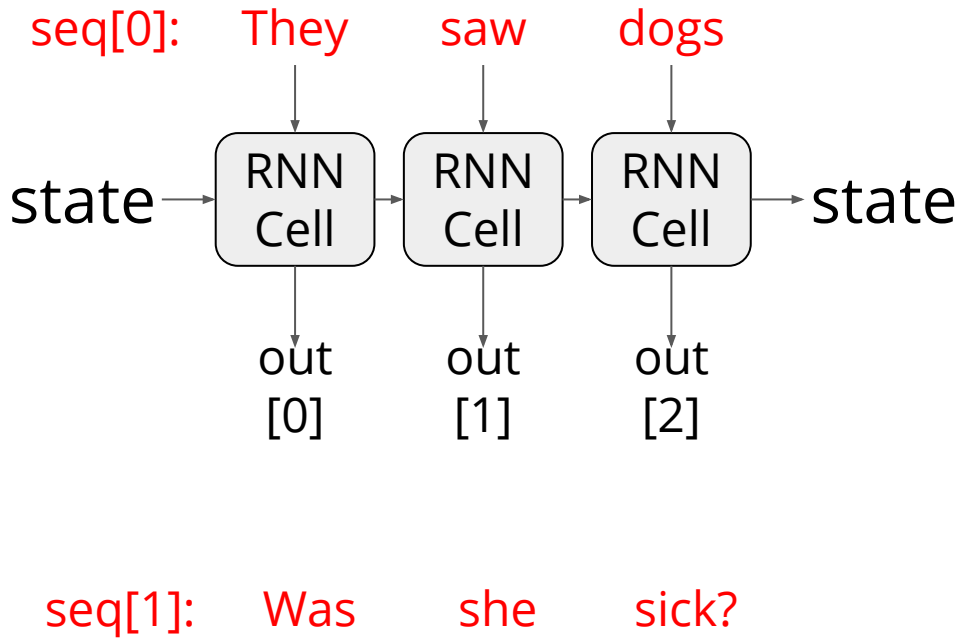


- Fast ☺
- Incorrect ☹
- Need Info ☹

RNN Example


```
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))
```

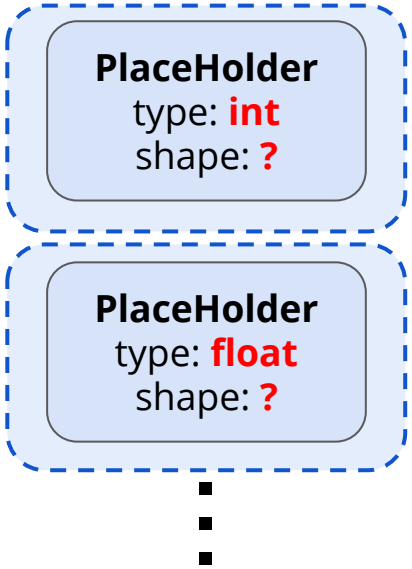


RNN Example

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 ☺
- Inefficient ☹

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: **?**

⋮

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

- Correct ☺
- Inefficient ☹

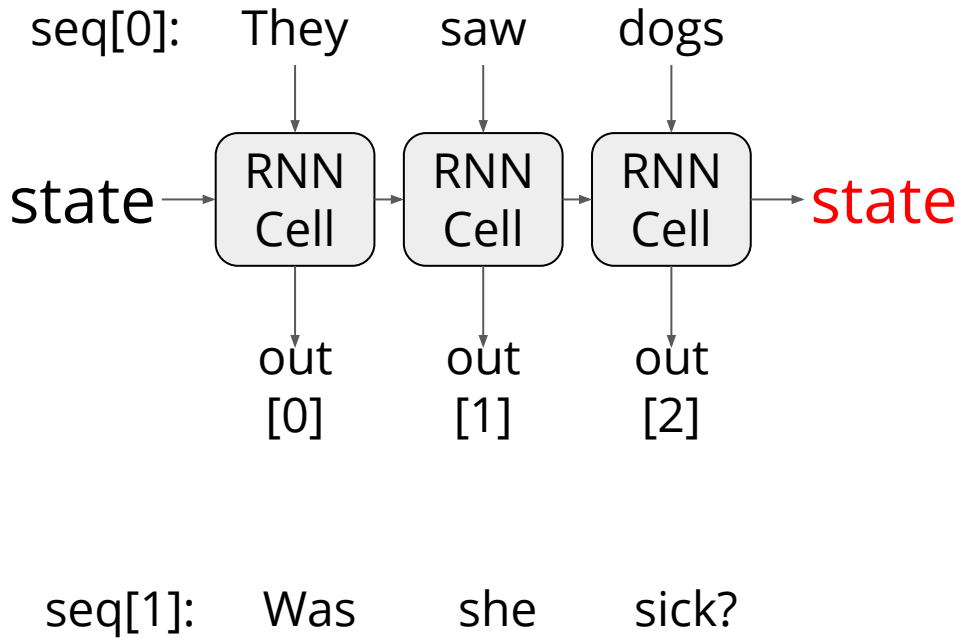
- Fast ☺
- Incorrect ☹
- Need Info ☹

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))
```

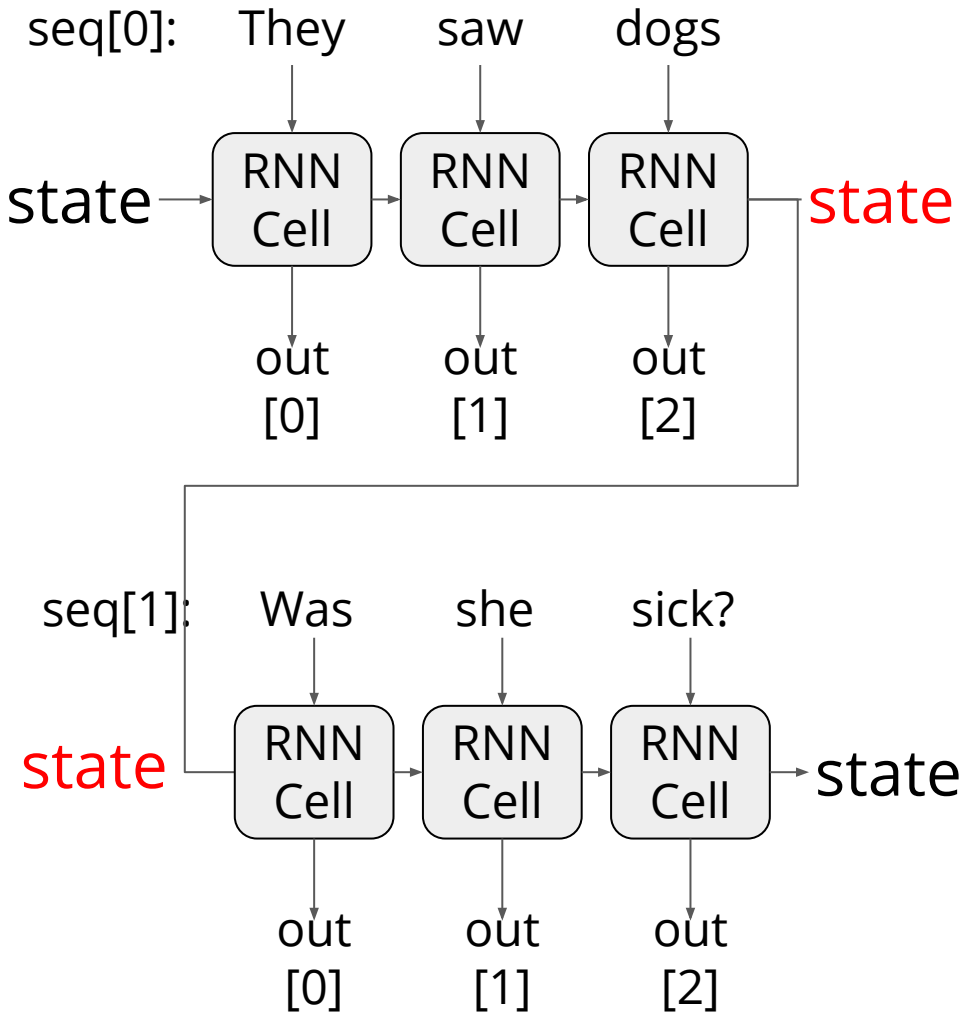


RNN Example

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)

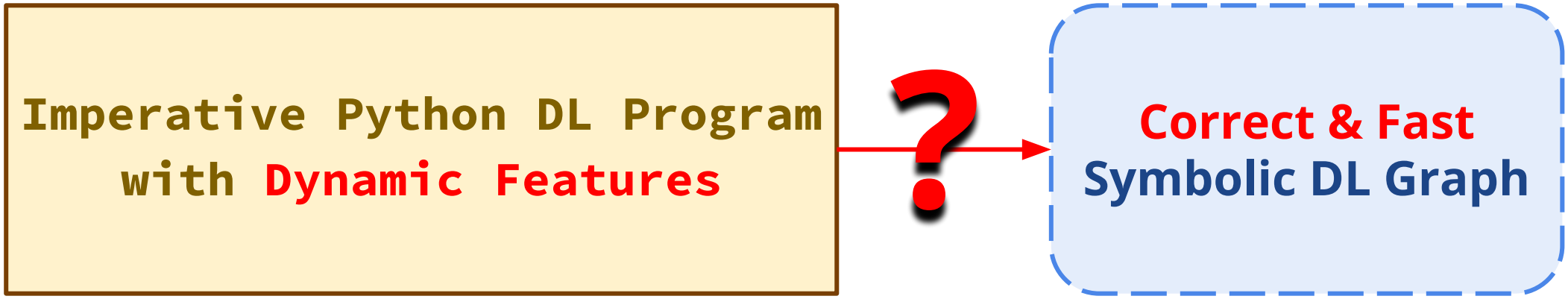
for sequence in sequences:
  optimize(lambda: model(sequence))
```

?

Challenge Summary

Challenge:

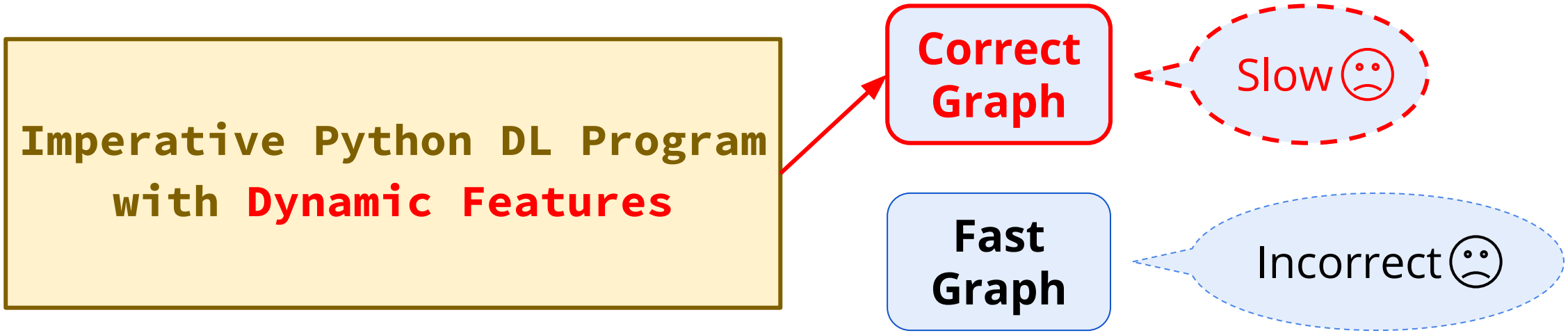
achieving Correctness & Performance at the same time



Challenge Summary

Challenge:

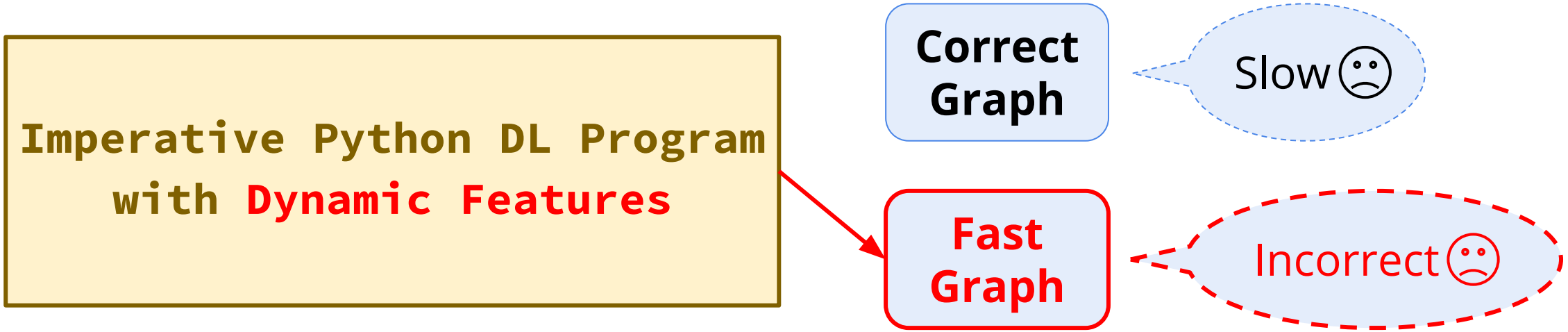
achieving Correctness & Performance at the same time



Challenge Summary

Challenge:

achieving Correctness & Performance at the same time



Outline

- Approach
- Challenges
- **JANUS**
- Evaluation

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)

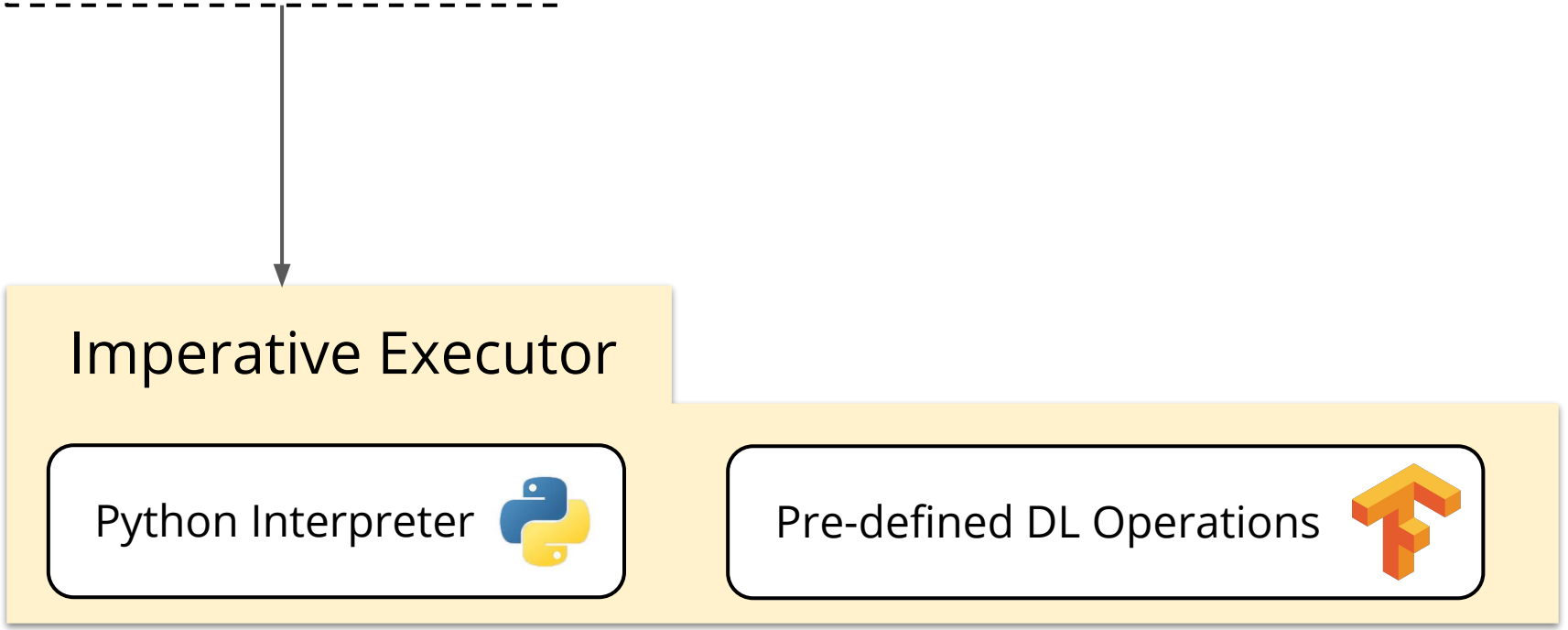
Overall Workflow on JANUS

Fast Path
(Common Case)

Correct Path
(Rare Case)

Imperative DL Program

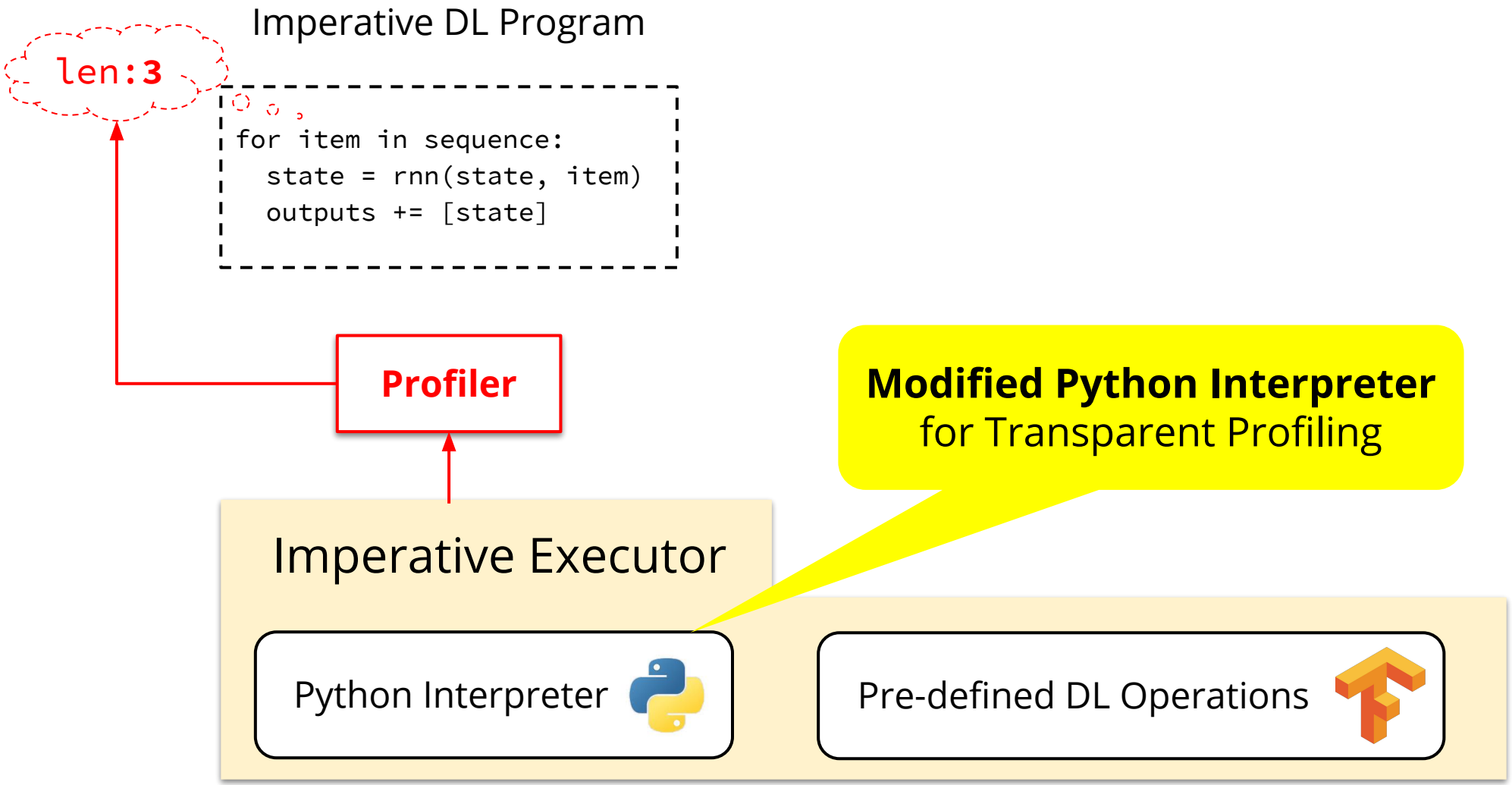
```
for item in sequence:
  state = rnn(state, item)
  outputs += [state]
```



Overall Workflow on JANUS

Fast Path
(Common Case)

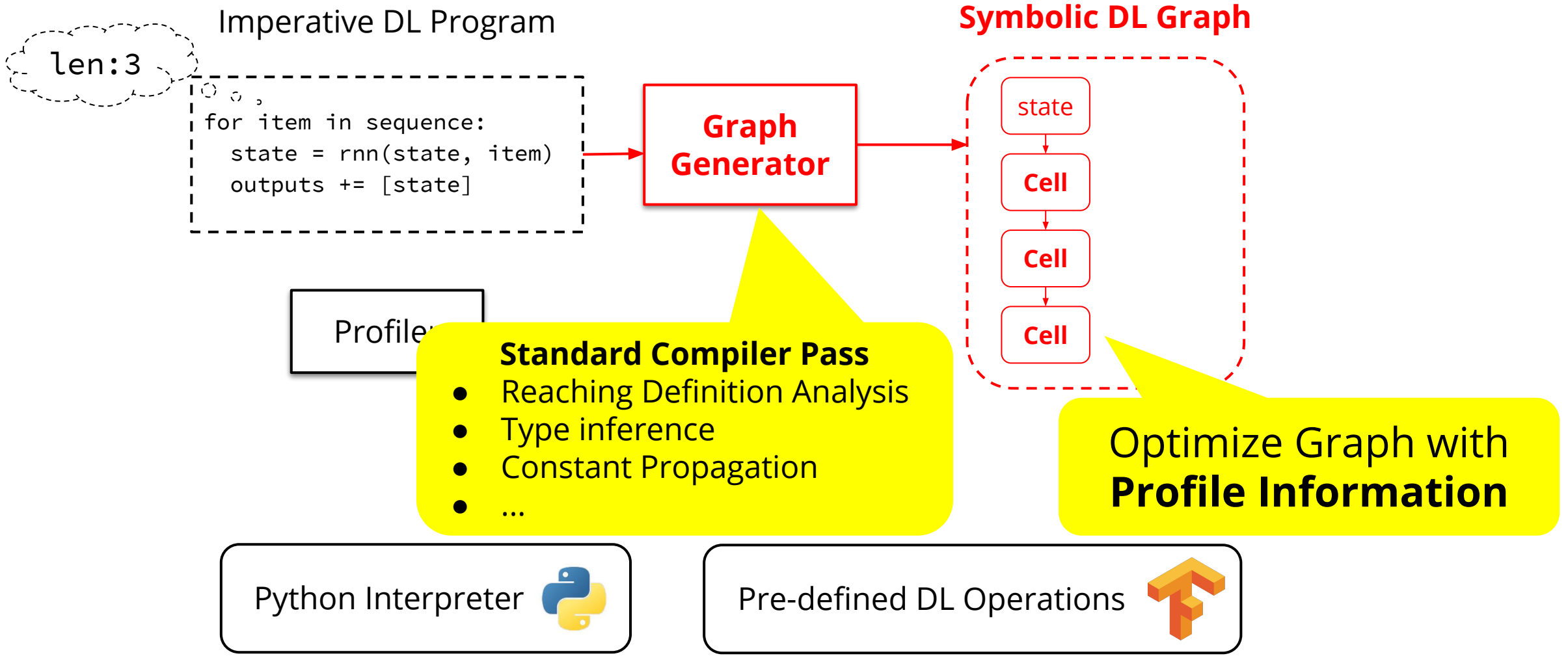
Correct Path
(Rare Case)



Overall Workflow on JANUS

Fast Path
(Common Case)

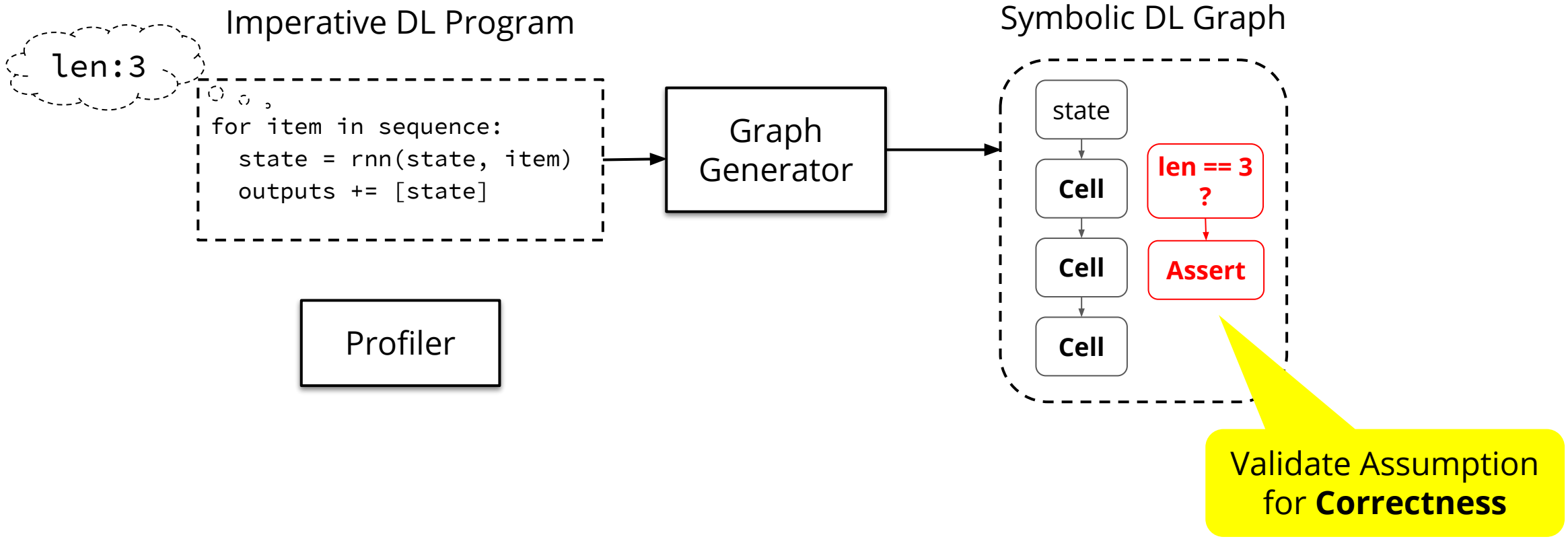
Correct Path
(Rare Case)



Overall Workflow on JANUS

Fast Path
(Common Case)

Correct Path
(Rare Case)



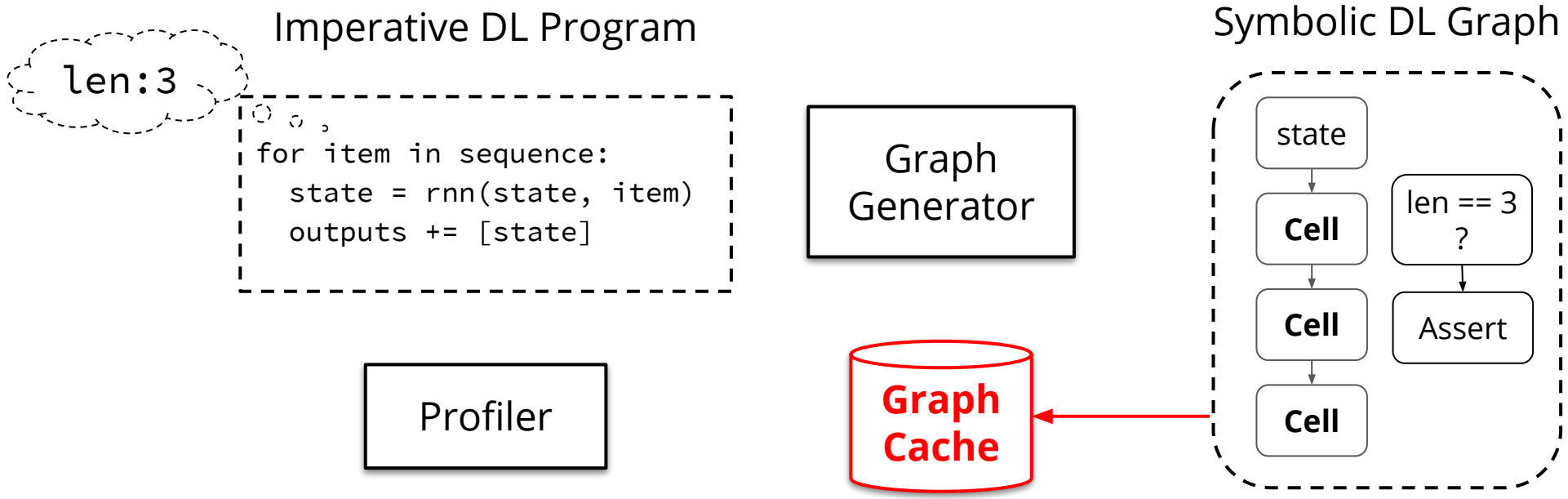
Python Interpreter

Pre-defined DL Operations

Overall Workflow on JANUS

Fast Path
(Common Case)

Correct Path
(Rare Case)



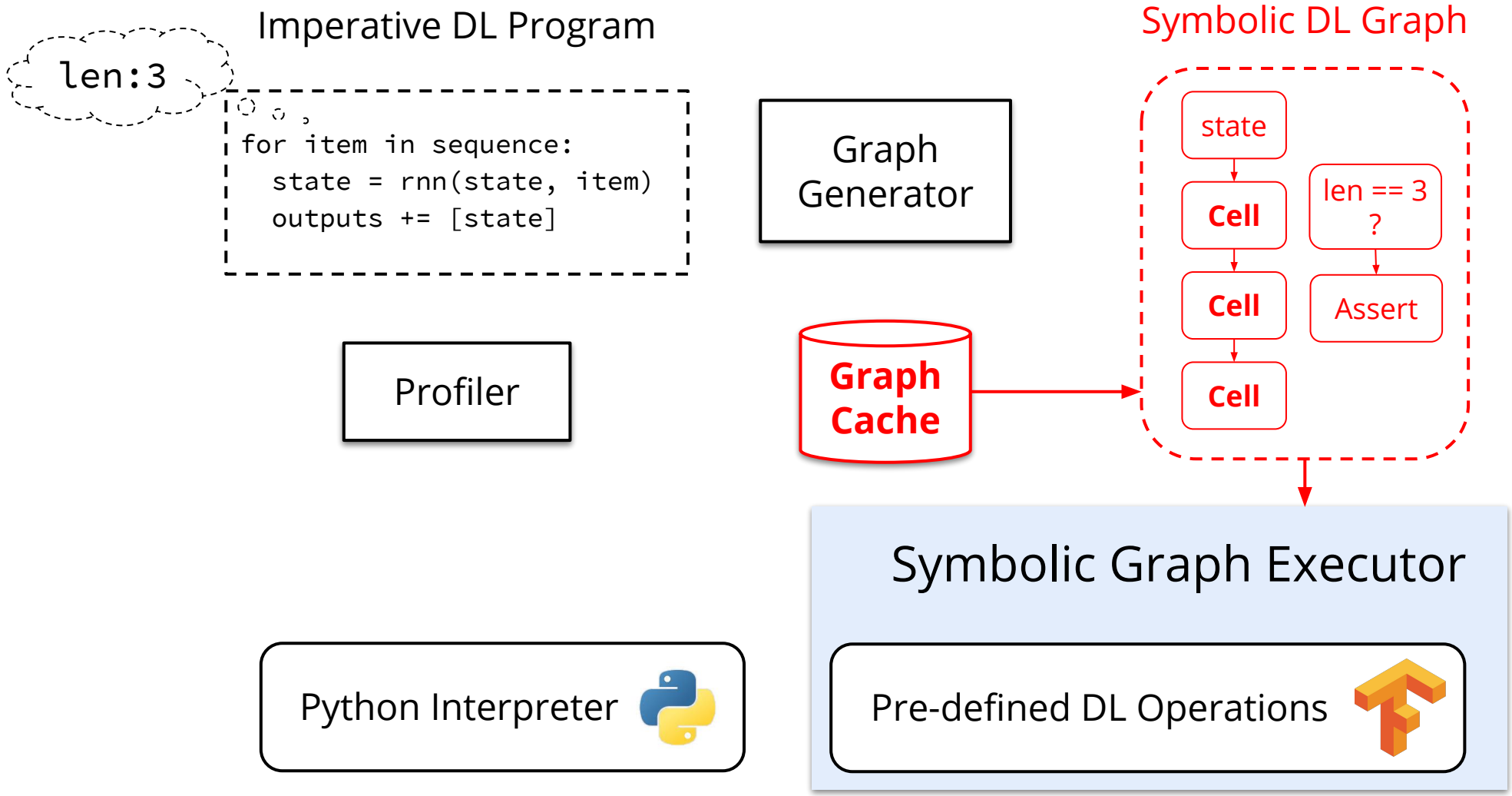
Python Interpreter

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)

Imperative DL Program

len:3

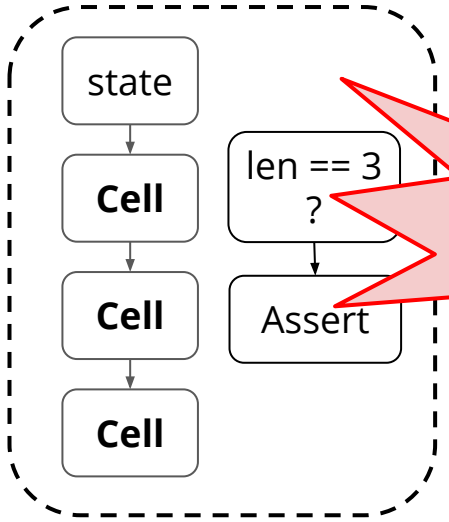
```
for item in sequence:
  state = rnn(state, item)
  outputs += [state]
```

Graph Generator

Profiler

Graph Cache

Symbolic DL Graph



Assumption Failure

Python Interpreter

Symbolic Graph Executor

Pre-defined DL Operations

Overall Workflow on JANUS

Fast Path
(Common Case)

Correct Path
(Rare Case)

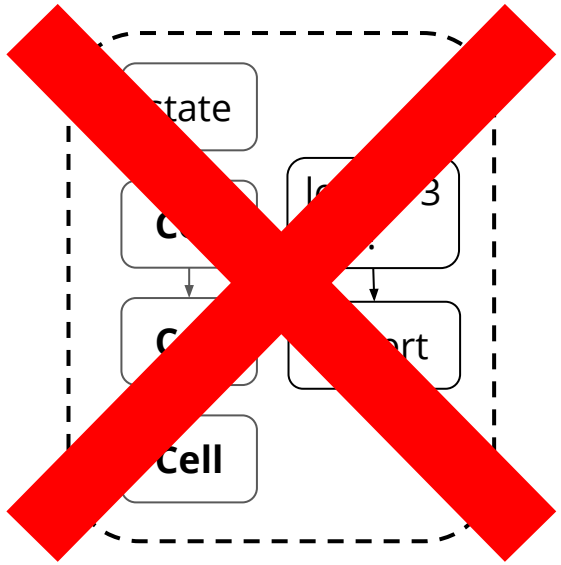
Imperative DL Program

len:3

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

Graph Generator

Symbolic DL Graph



Profiler

Graph Cache

Symbolic Graph Executor

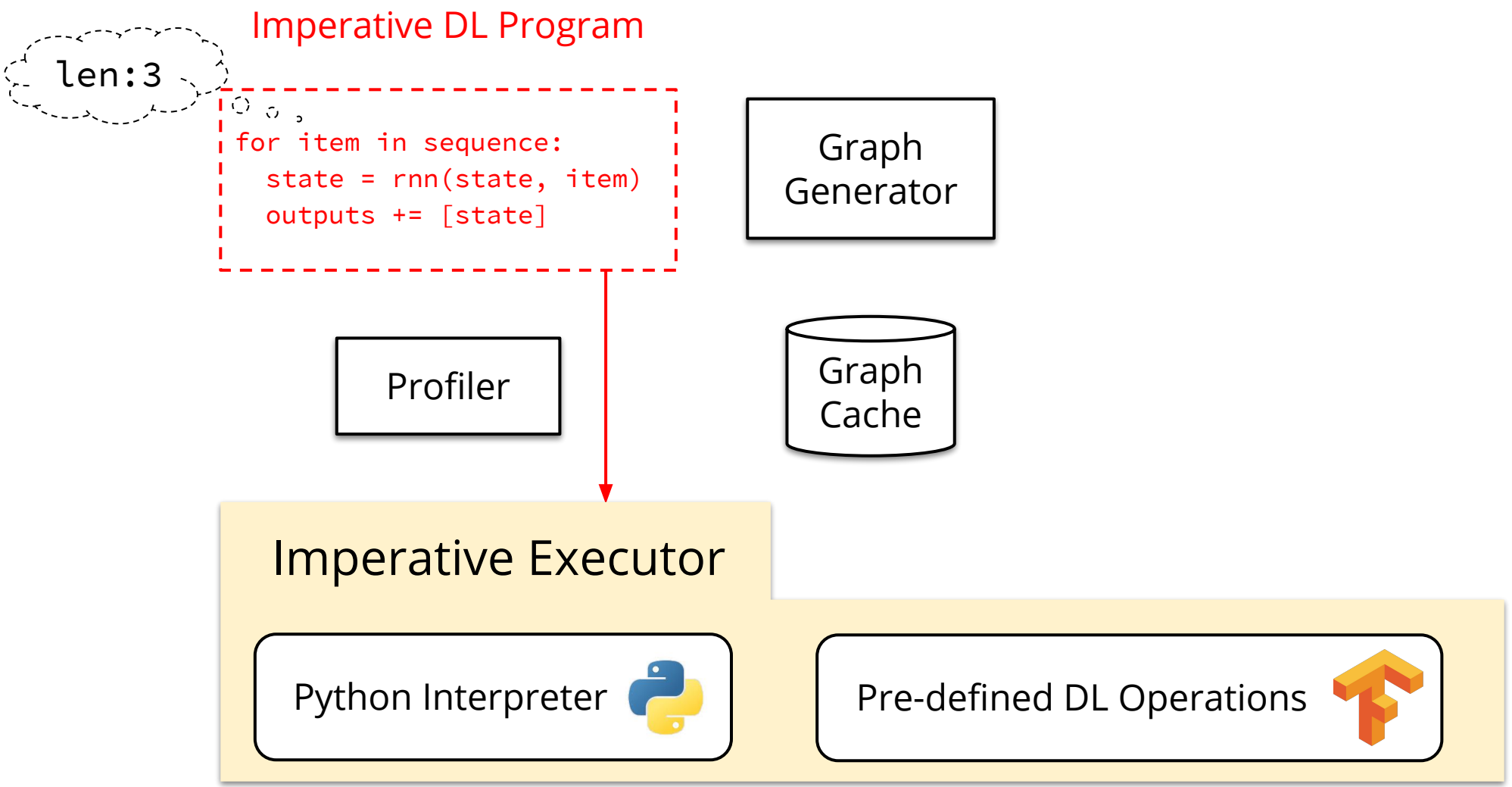
Python Interpreter

Pre-defined DL Operations

Overall Workflow on JANUS

Fast Path
(Common Case)

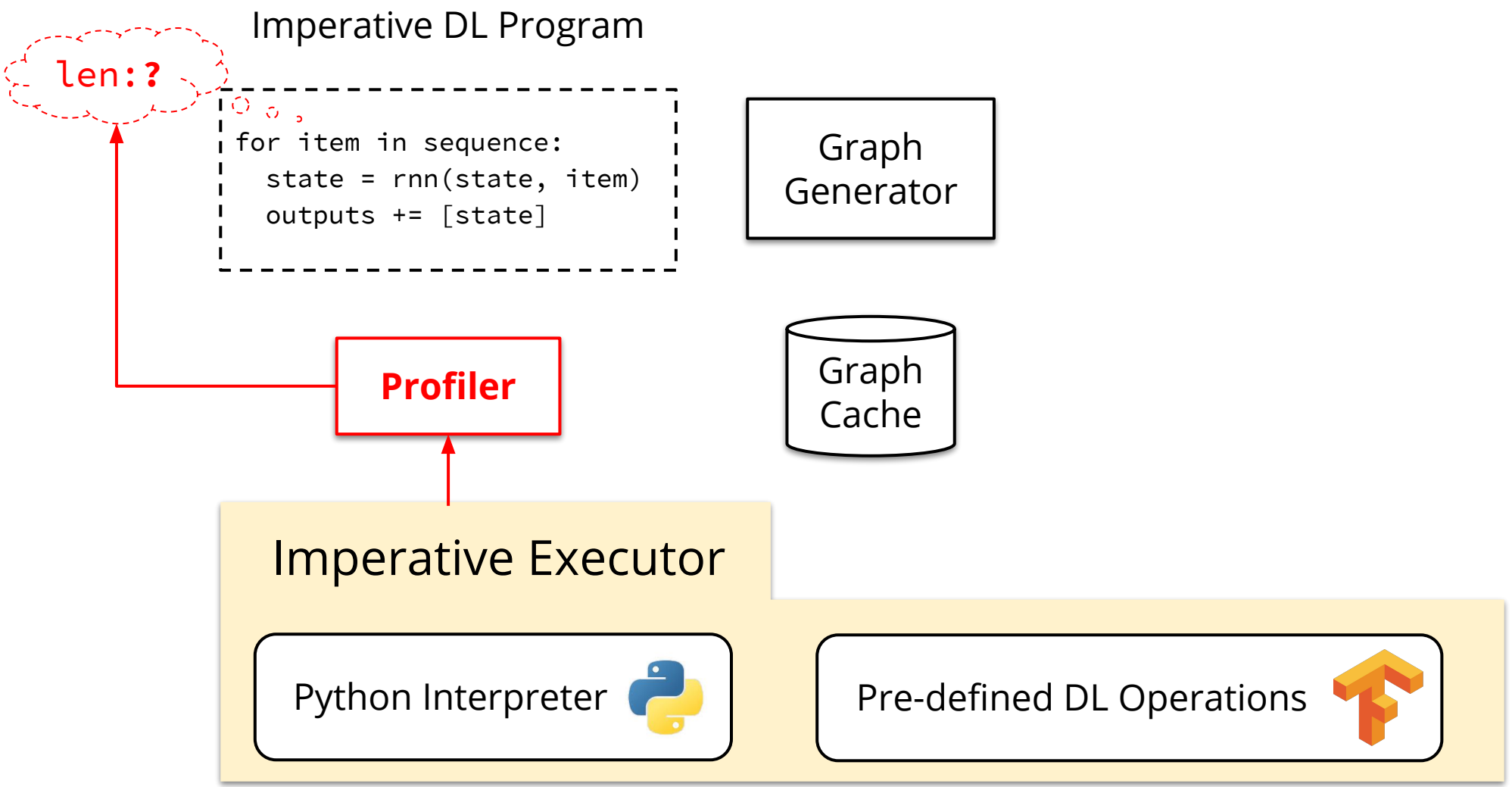
Correct Path
(Rare Case)



Overall Workflow on JANUS

Fast Path
(Common Case)

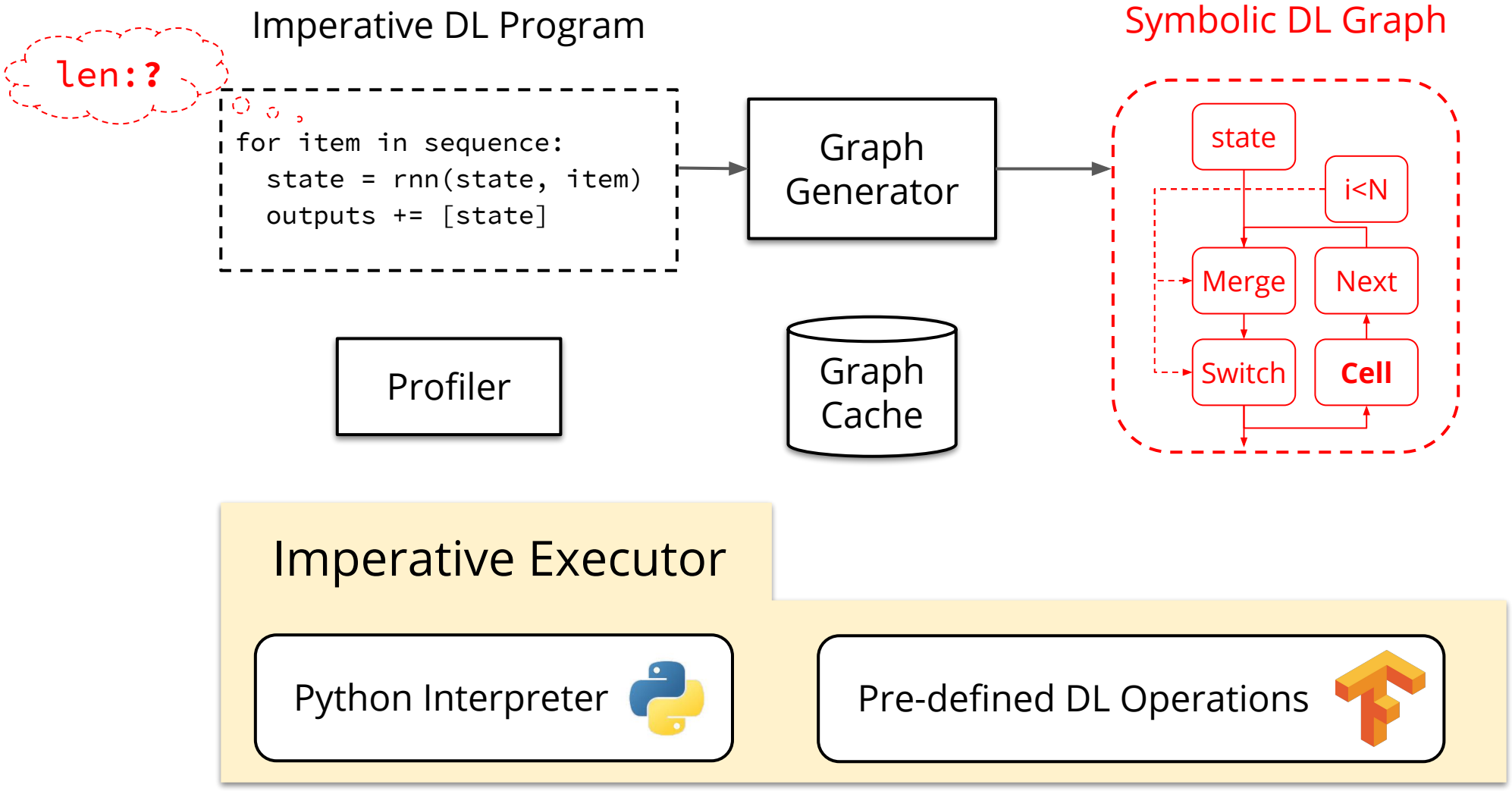
Correct Path
(Rare Case)



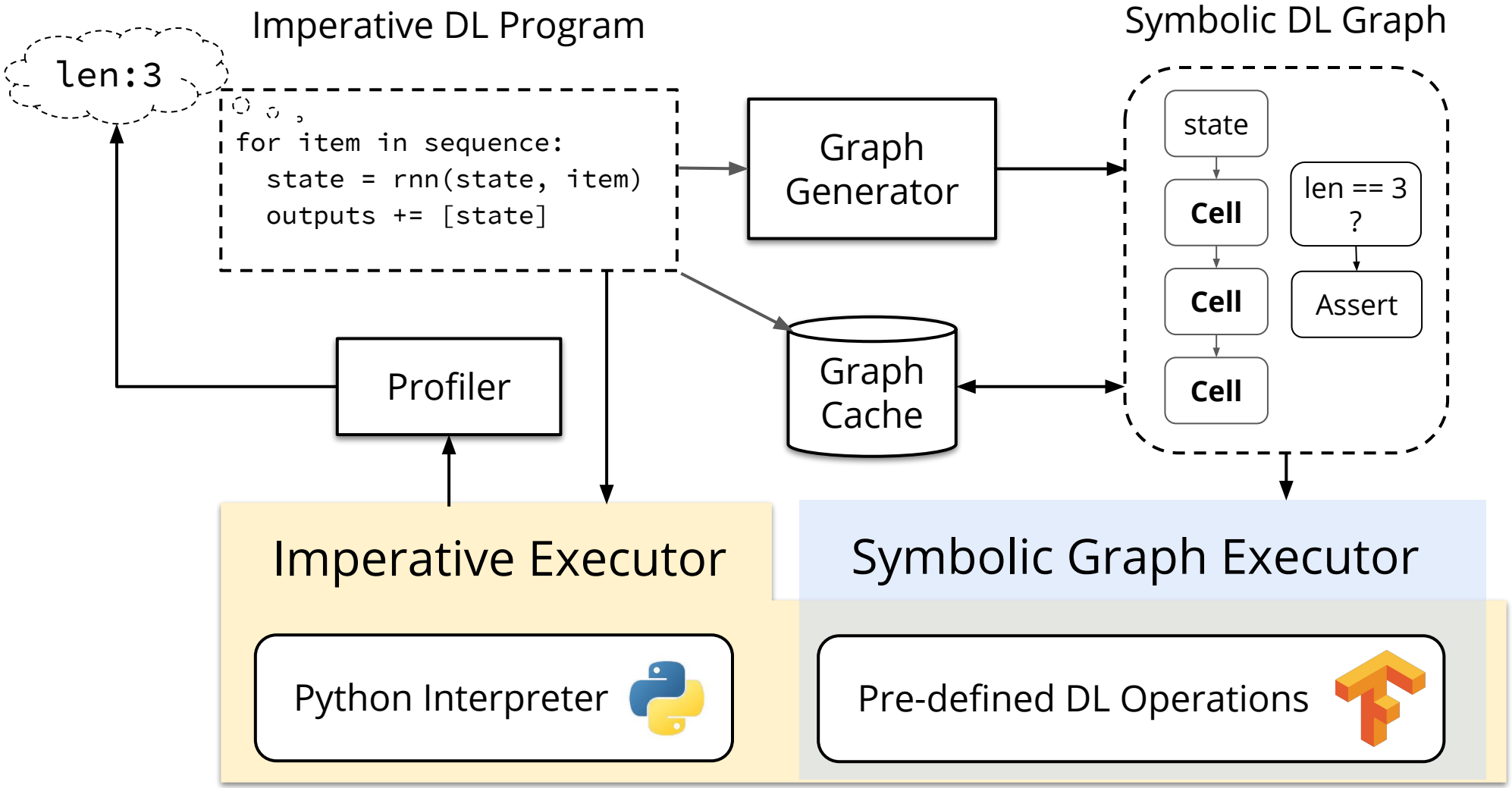
Overall Workflow on JANUS

Fast Path
(Common Case)

Correct Path
(Rare Case)



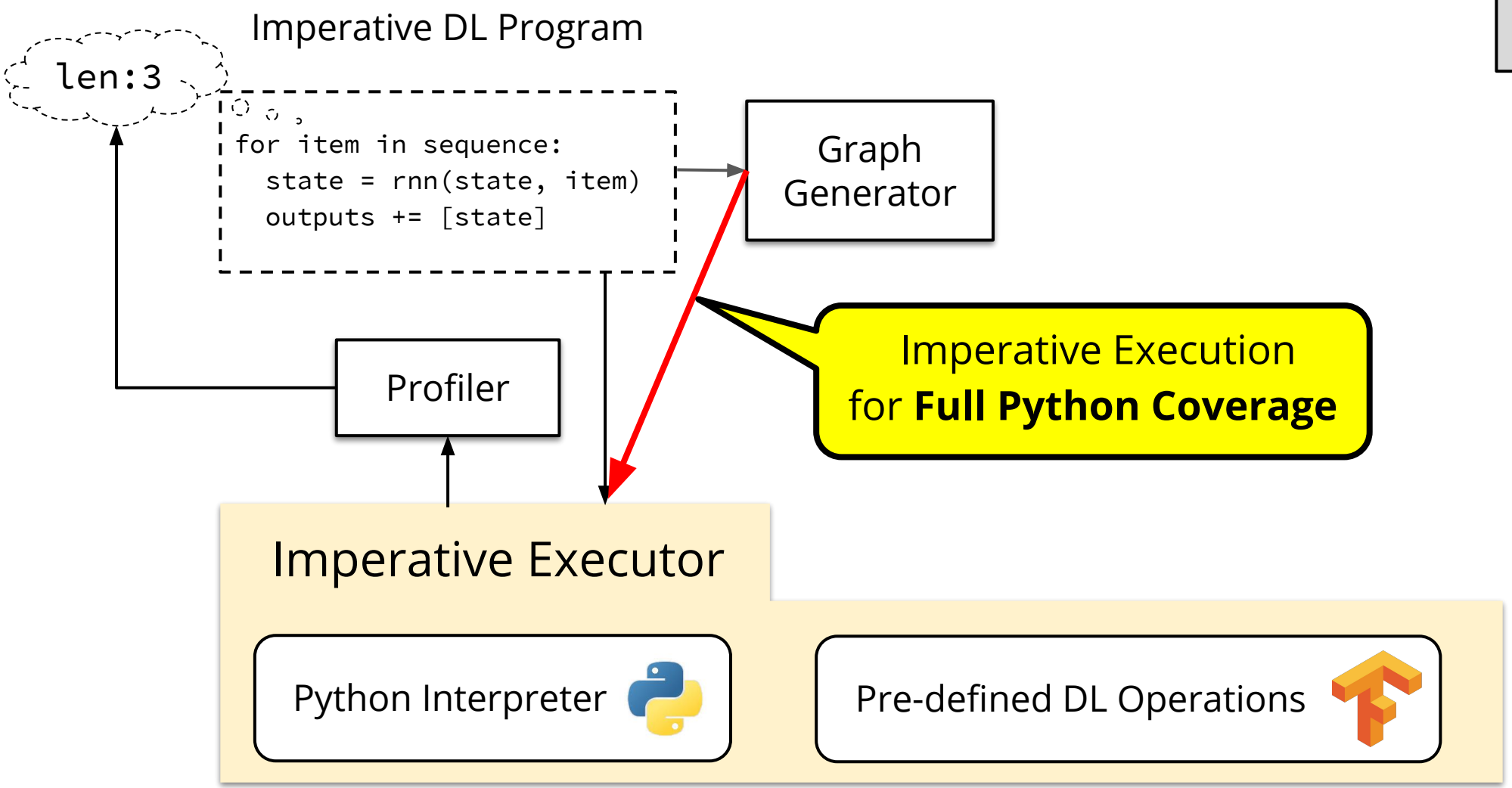
Overall Workflow on JANUS



Additional System Aspects

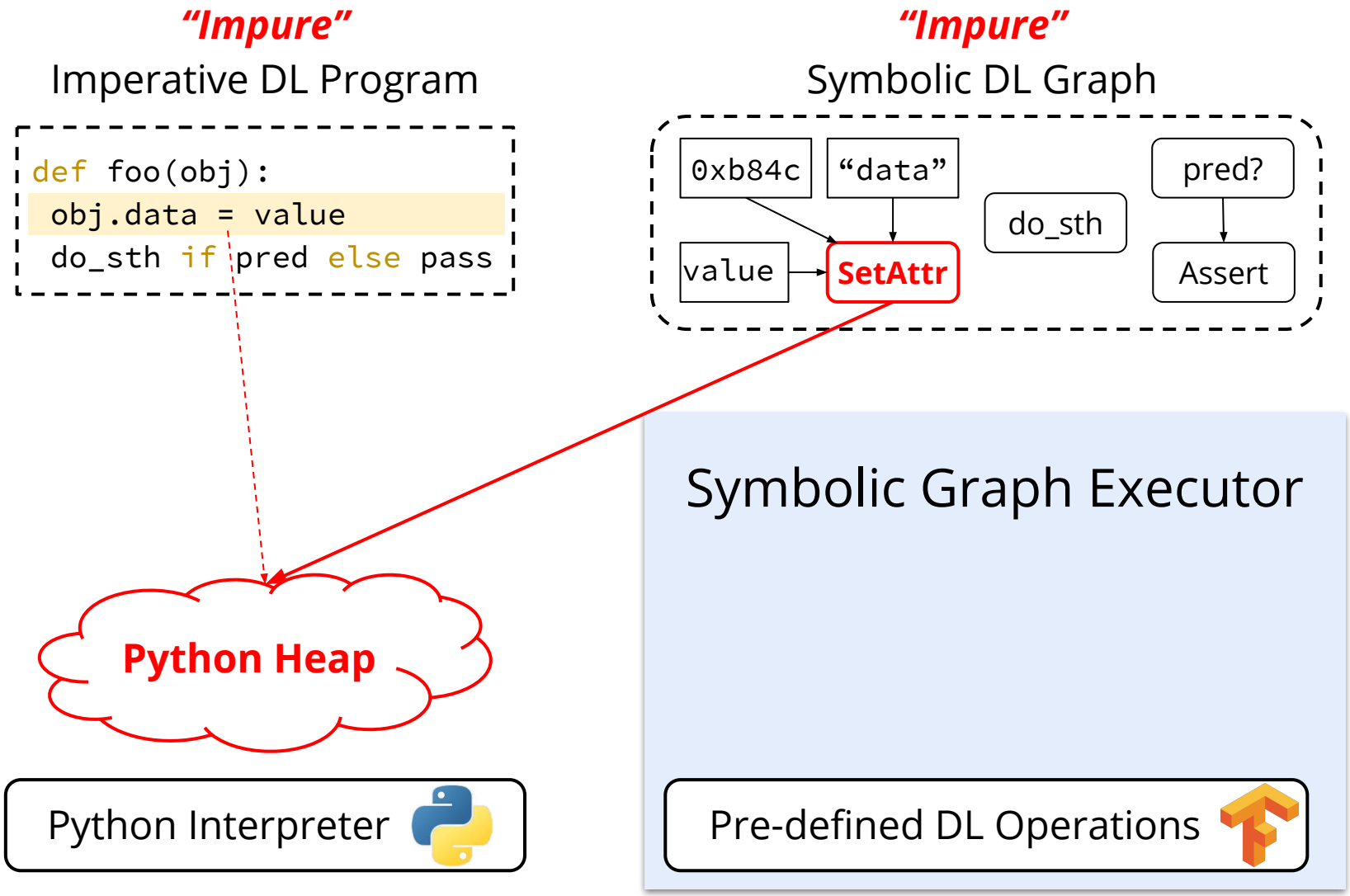
Python Coverage | Global State Consistency

See our paper for more details!



Additional System Aspects

Python Coverage | **Global State Consistency**



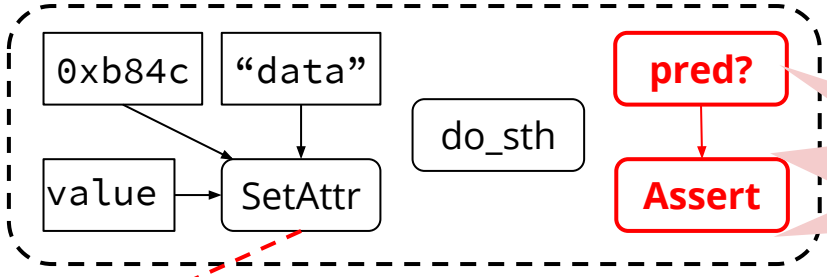
Additional System Aspects

Python Coverage | **Global State Consistency**

Impure
Imperative DL Program

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

Impure
Symbolic DL Graph



Assumption Failure

Unsafe to fallback after heap update

?

Modified Python Heap

Python Interpreter

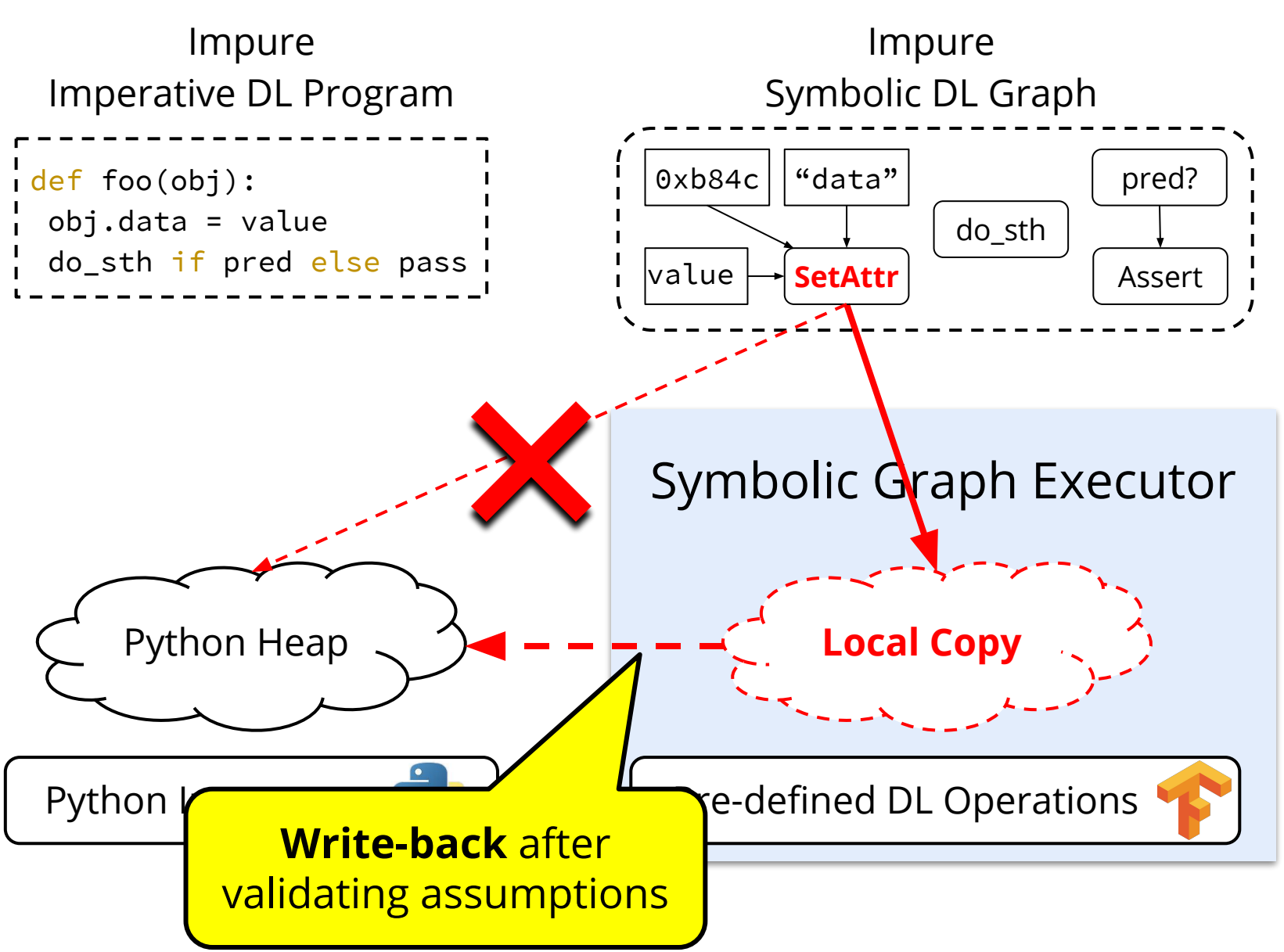
Symbolic Graph Executor

Pre-defined DL Operations

Additional System Aspects

Python Coverage | **Global State Consistency**

See our paper for more details!



Outline

- Approach
- Challenges
- JANUS
- **Evaluation**
  - √ Correctness and Performance
  - √ Breakdown of Performance Improvement

Evaluation Setup: Frameworks & Environments

Executors: TensorFlow (TF) + TF Eager (modified)
LoC: 4700 LoC / TF diff 771 LoC / CPython diff 1096 LoC

- **Frameworks**
  - **JANUS** Implemented on top of TensorFlow and CPython
  - **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

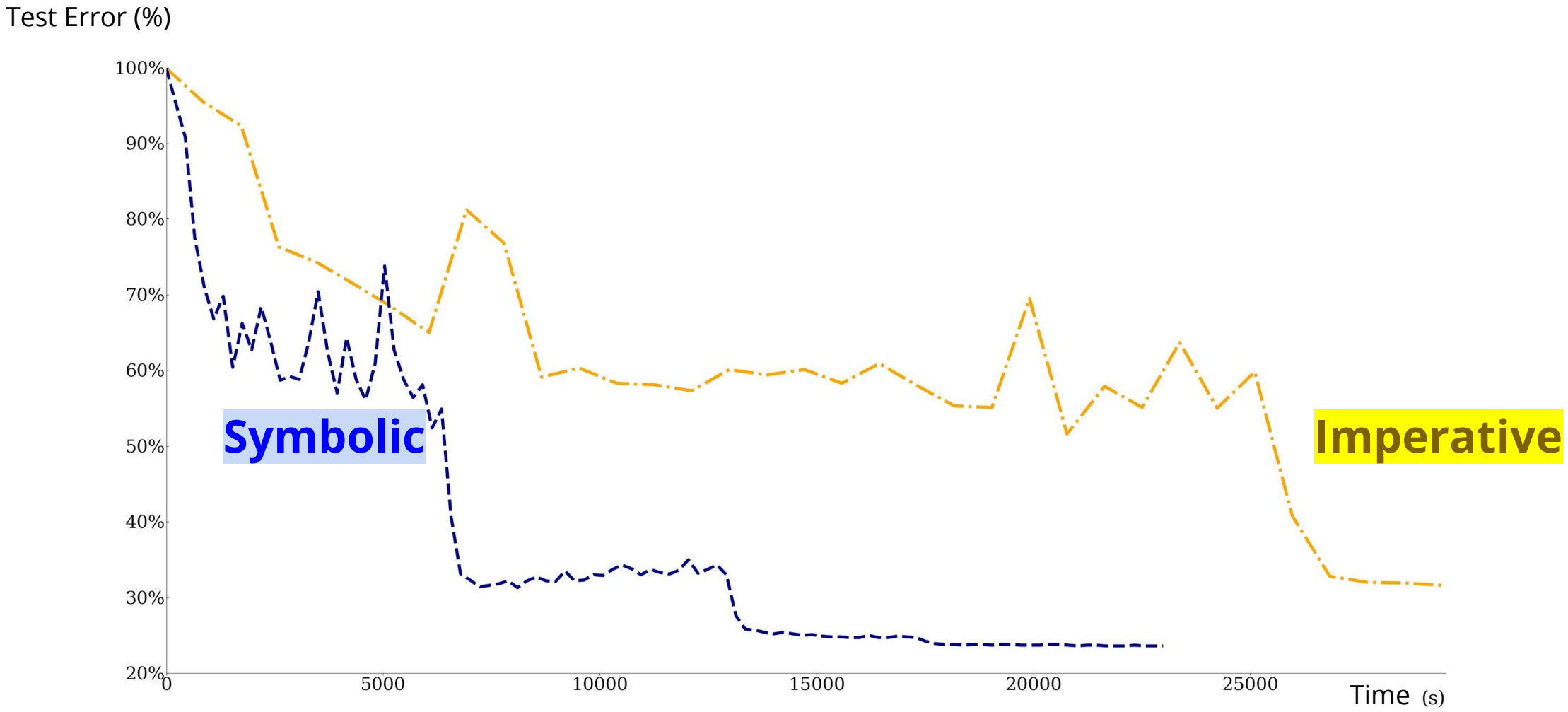
Evaluation Setup: Applications

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

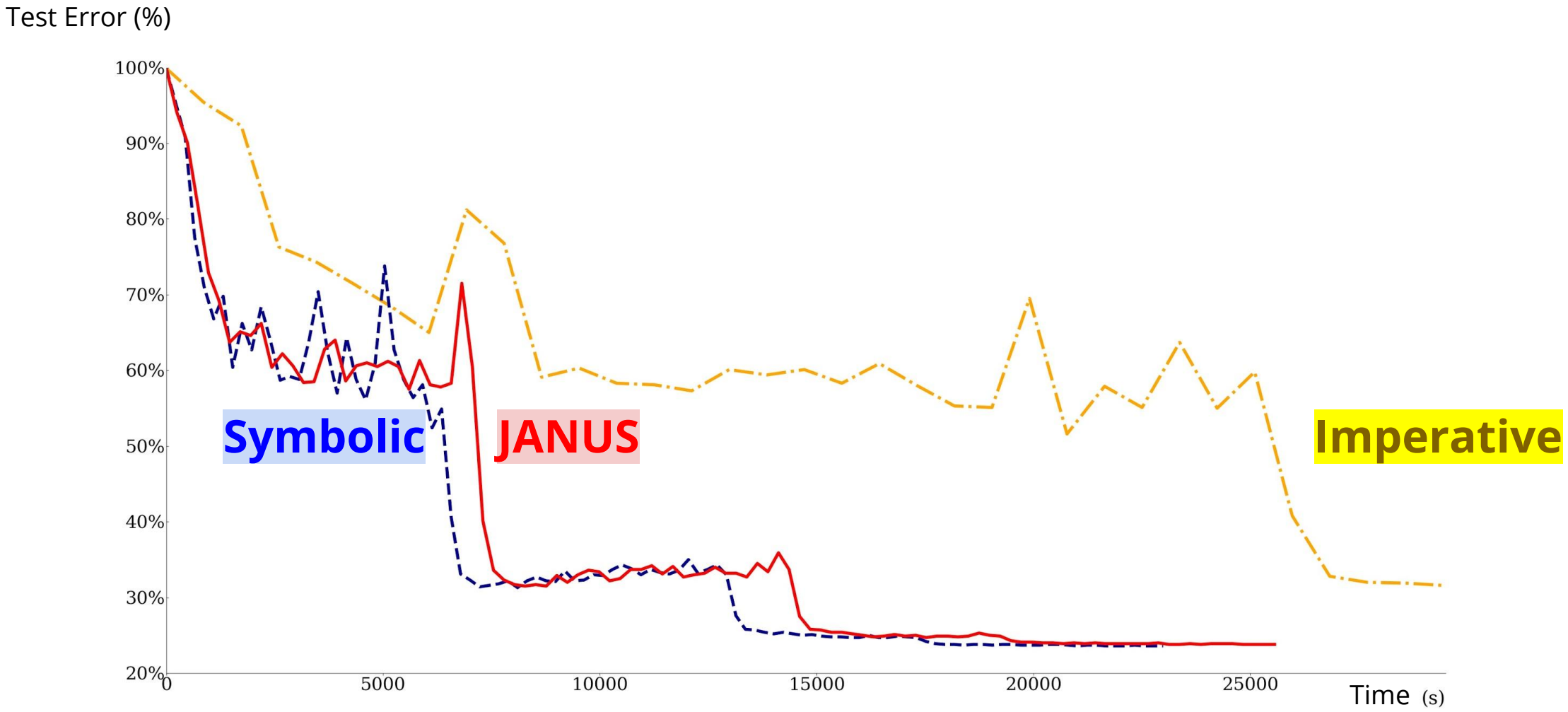
36 GPUs



Faster

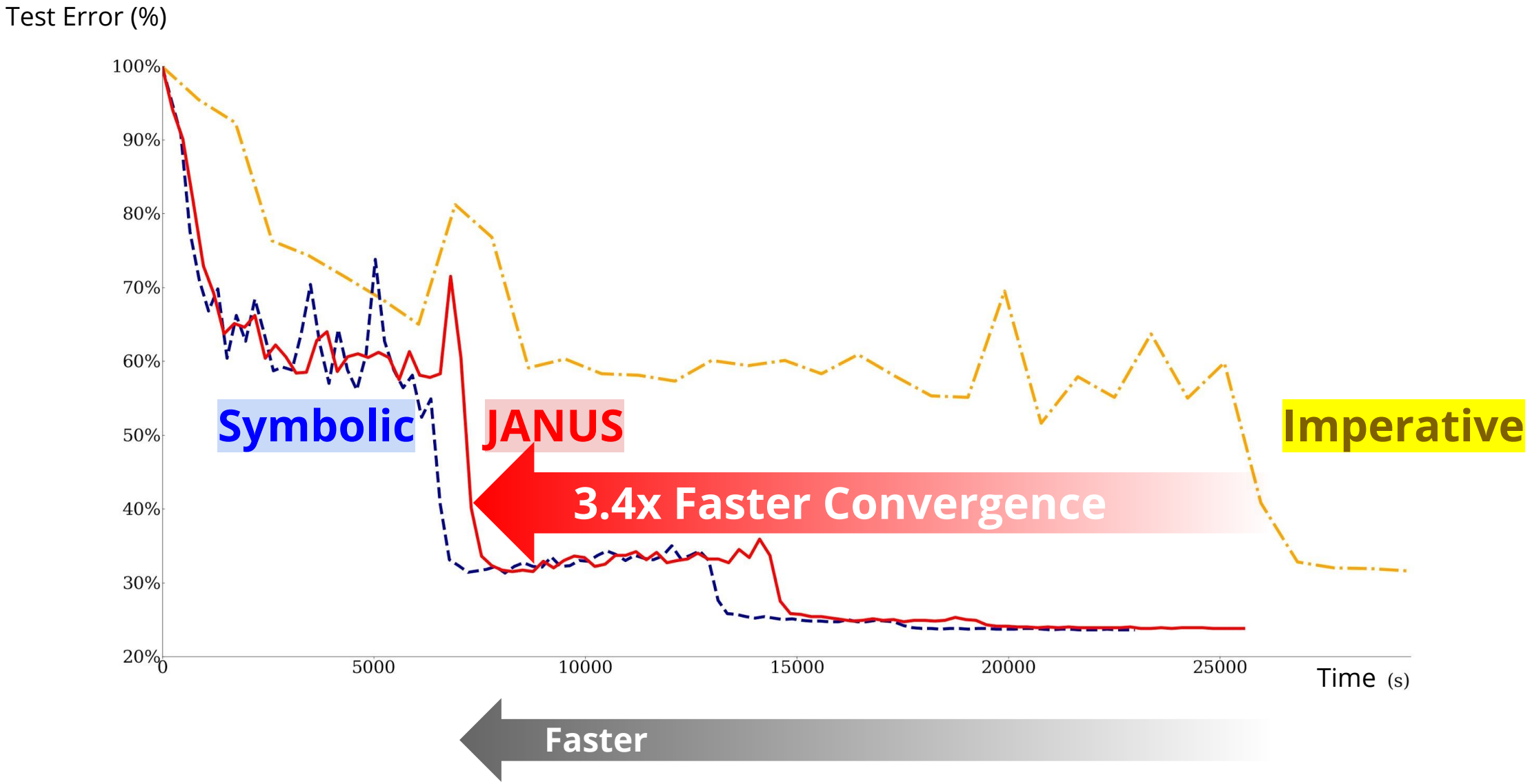
ImageNet Test Error with ResNet50

36 GPUs



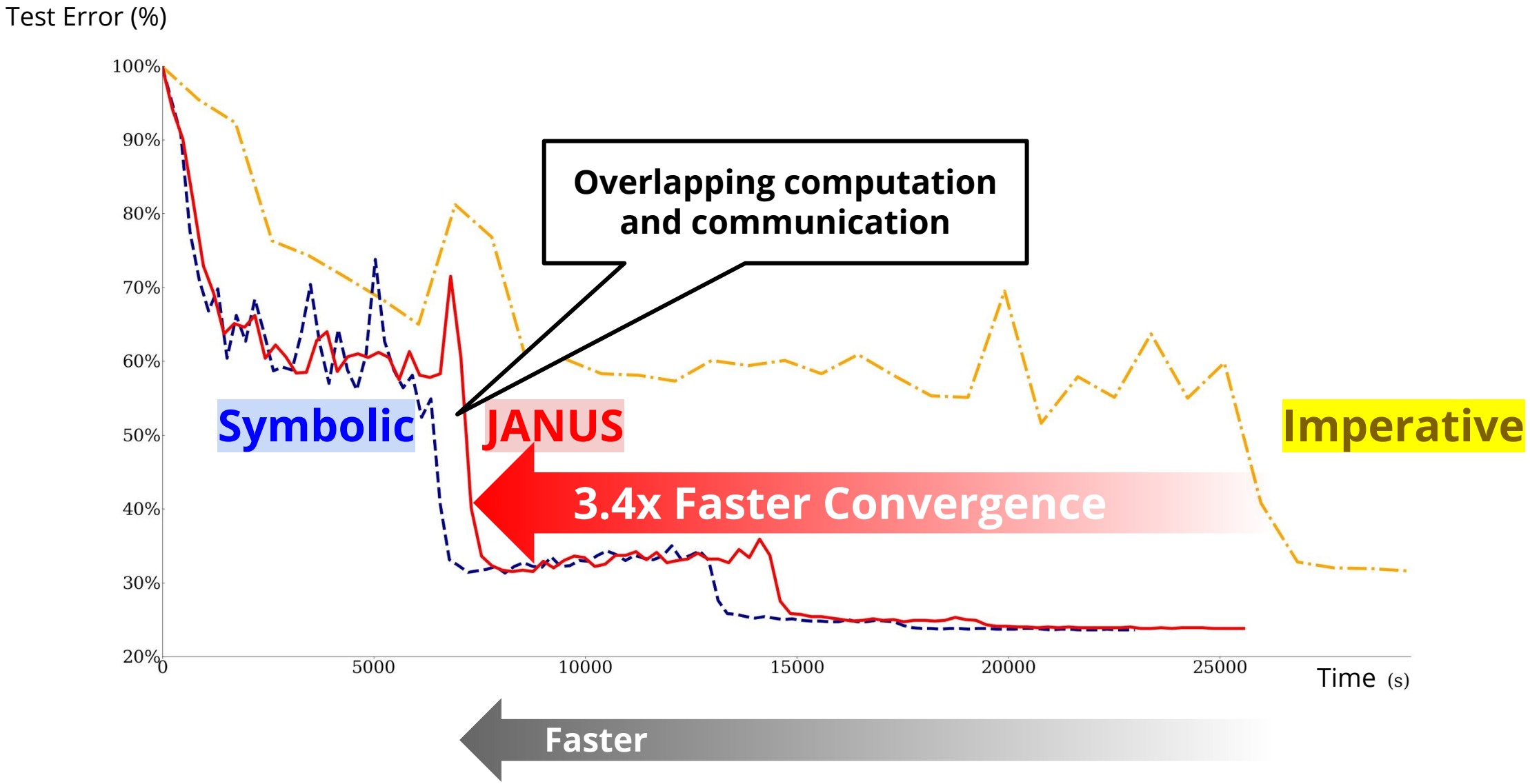
ImageNet Test Error with ResNet50

36 GPUs

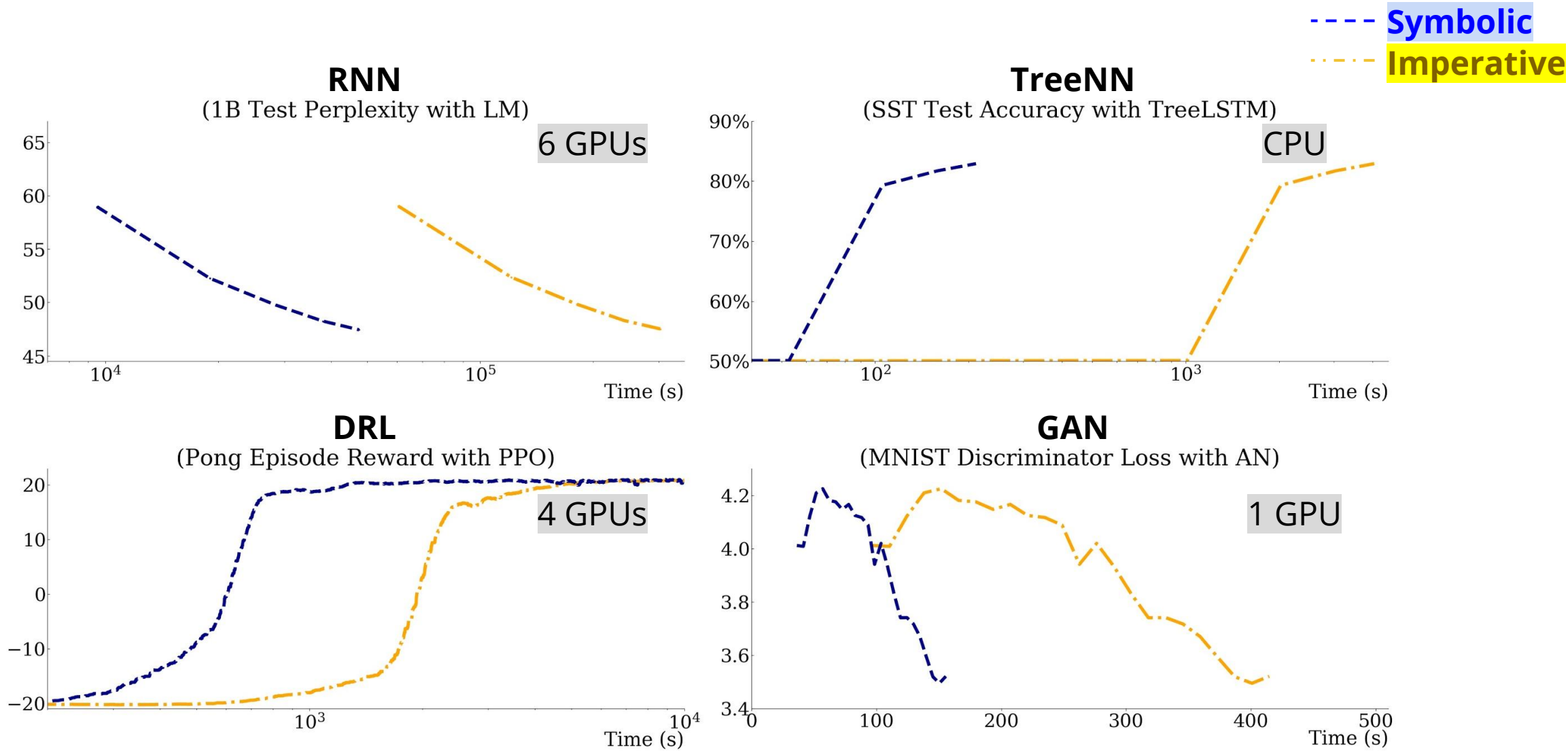


ImageNet Test Error with ResNet50

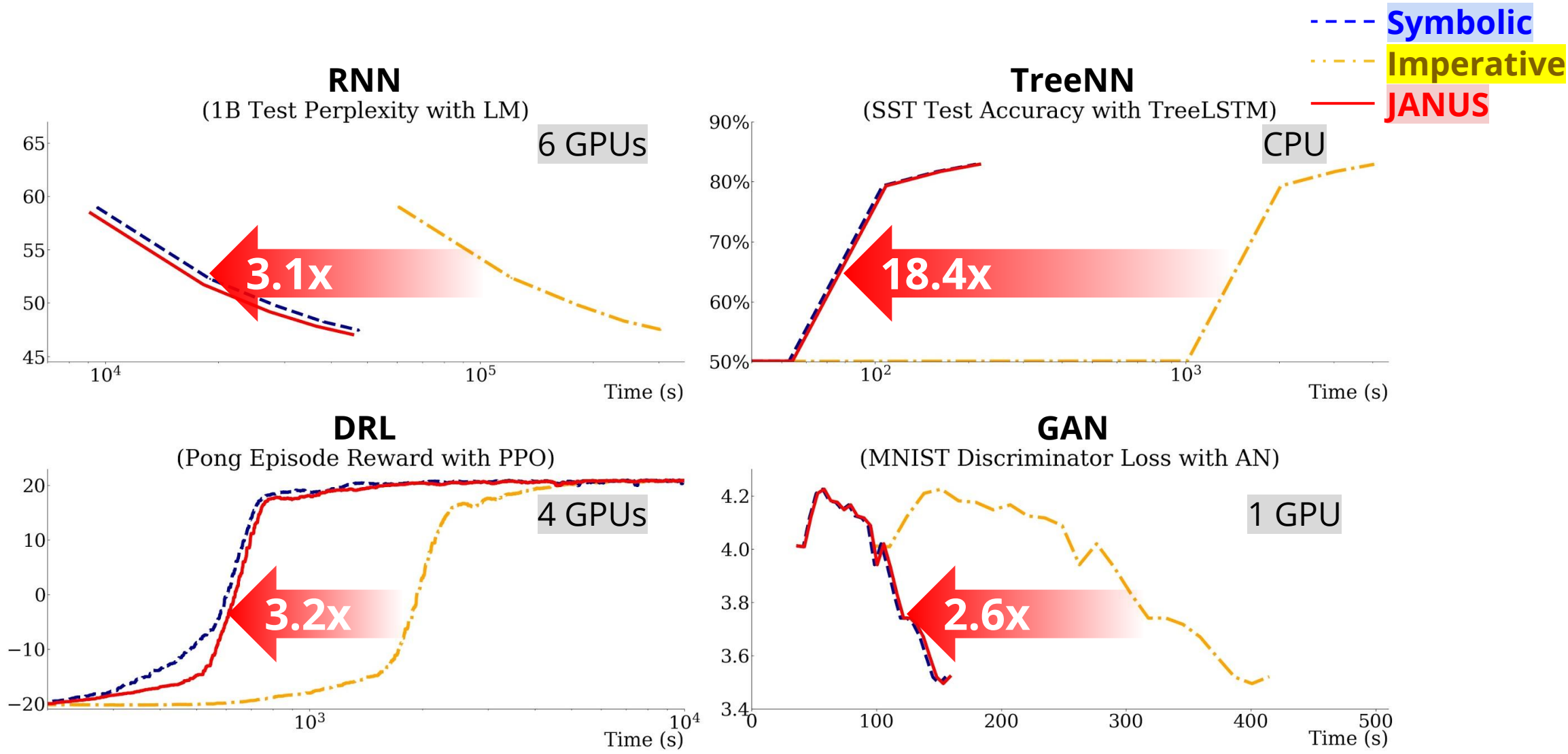
36 GPUs



Model Convergence

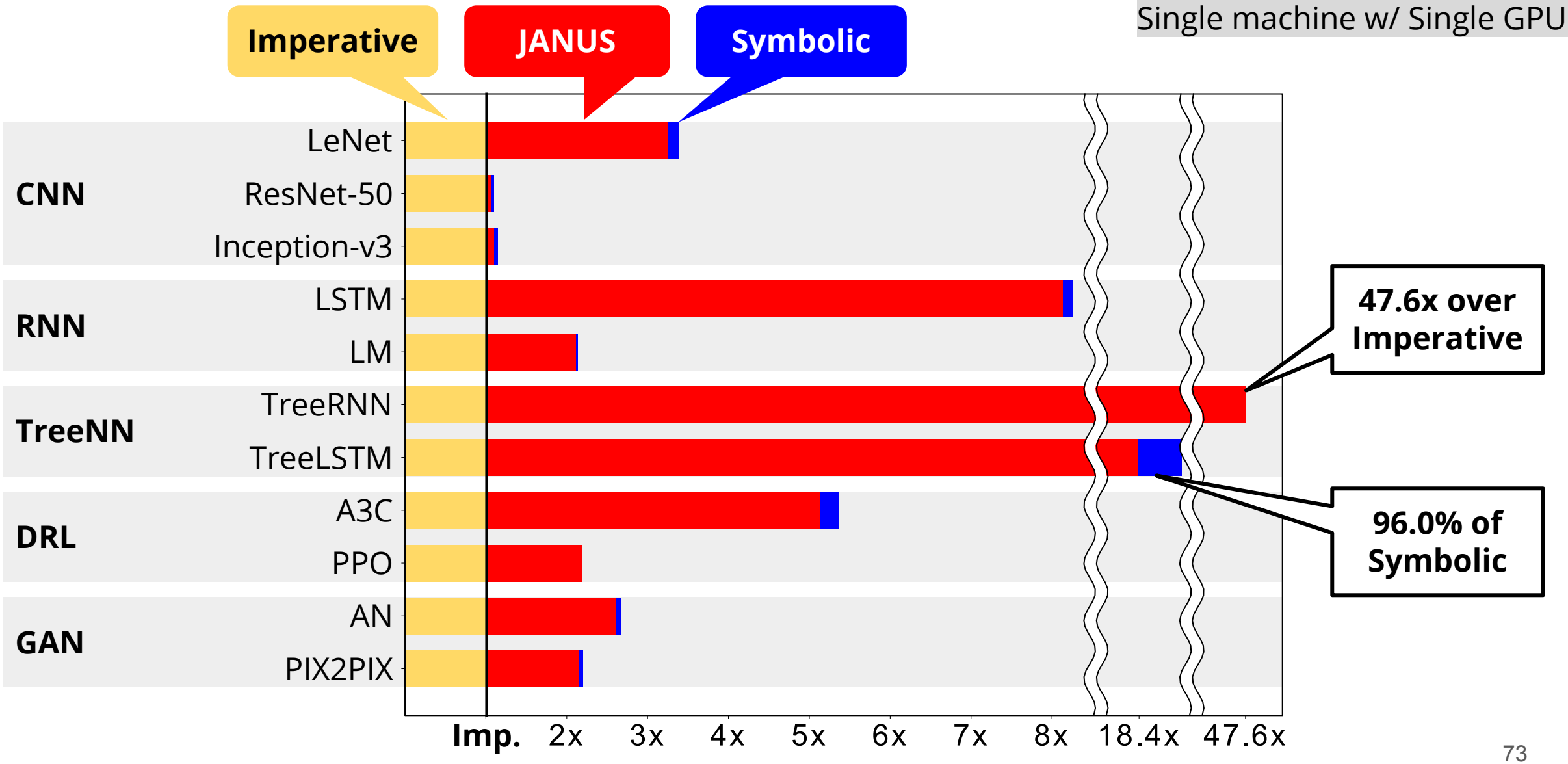


Model Convergence

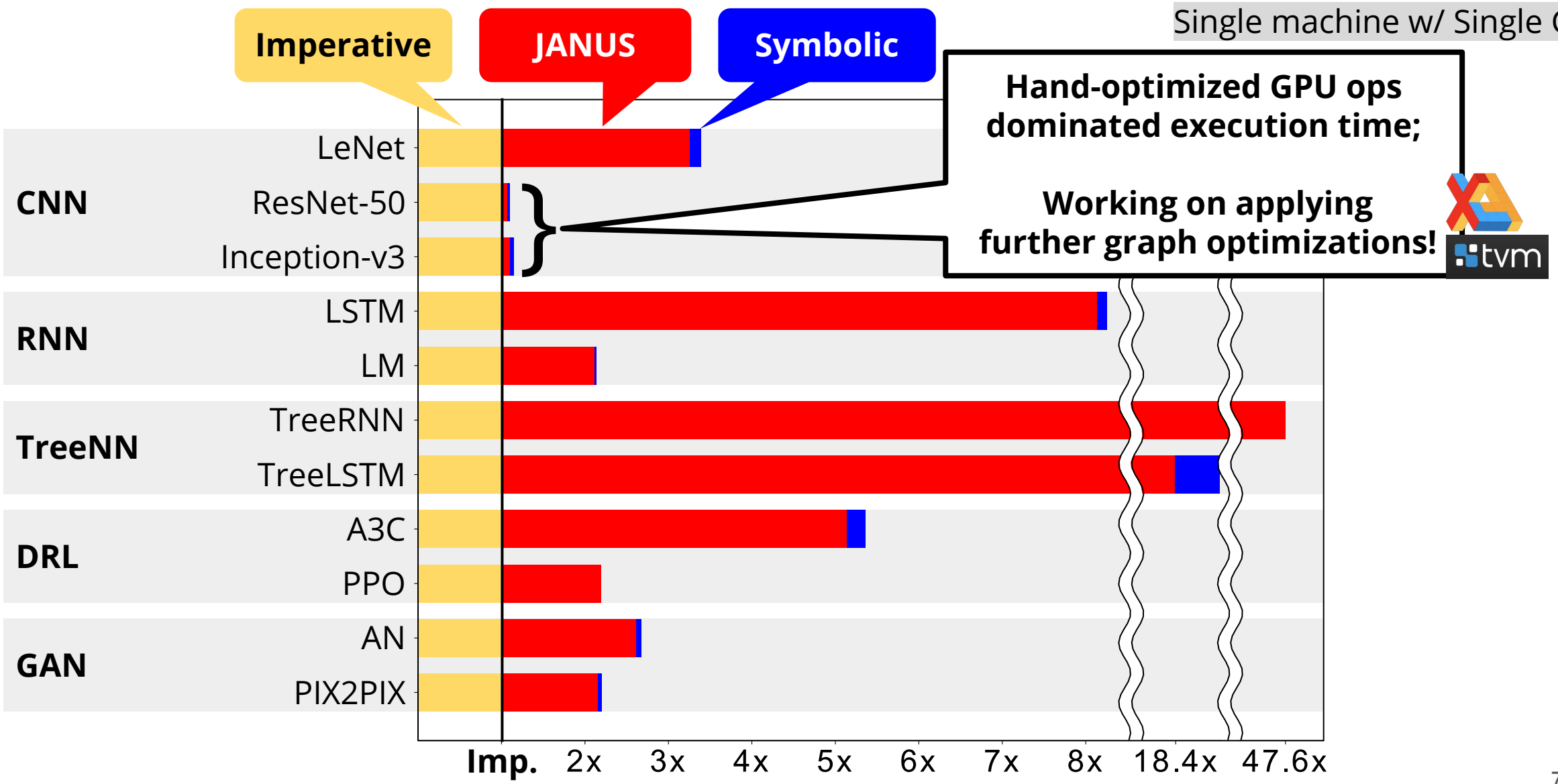


Faster

Normalized Training Throughput

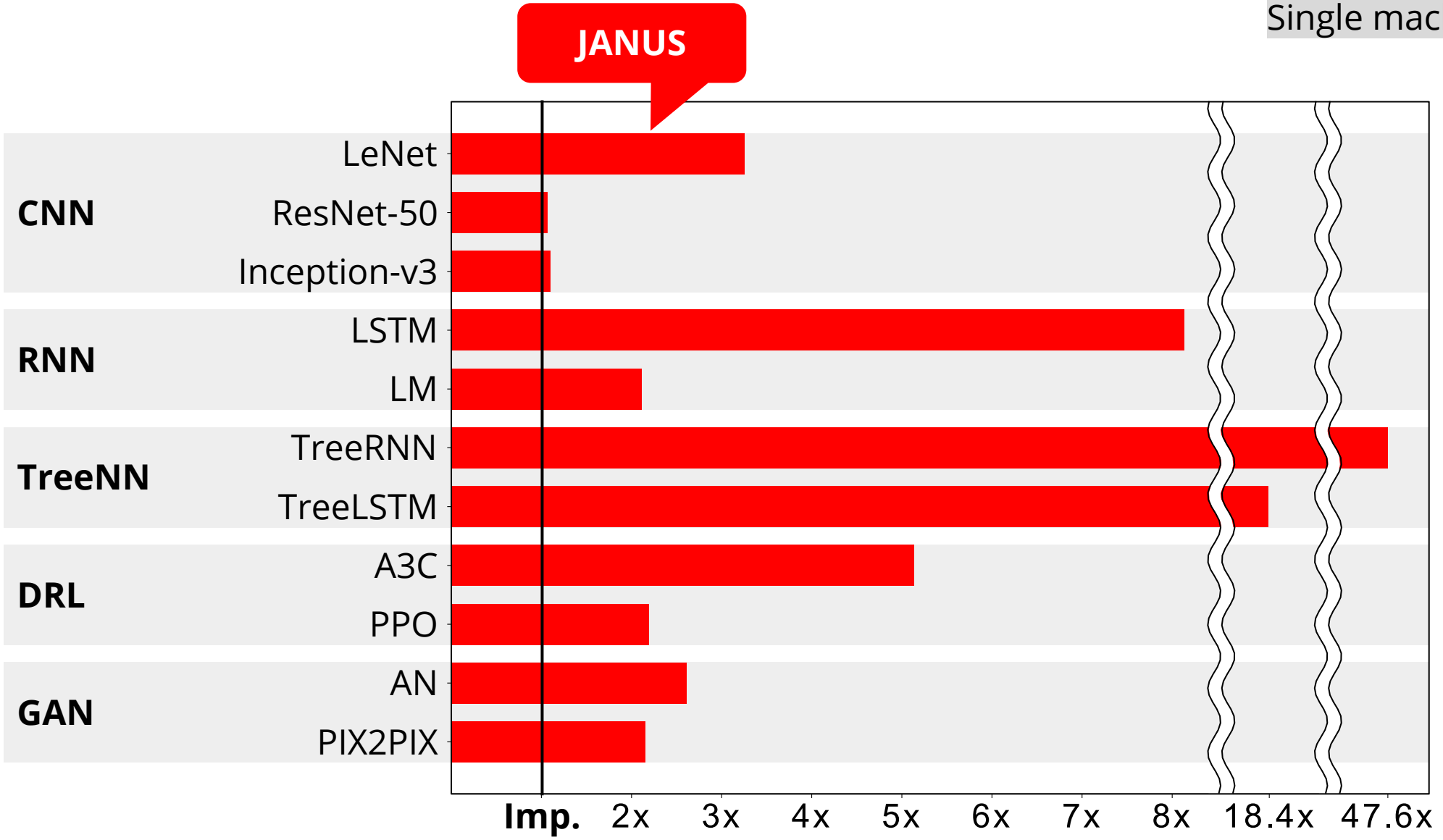


Normalized Training Throughput



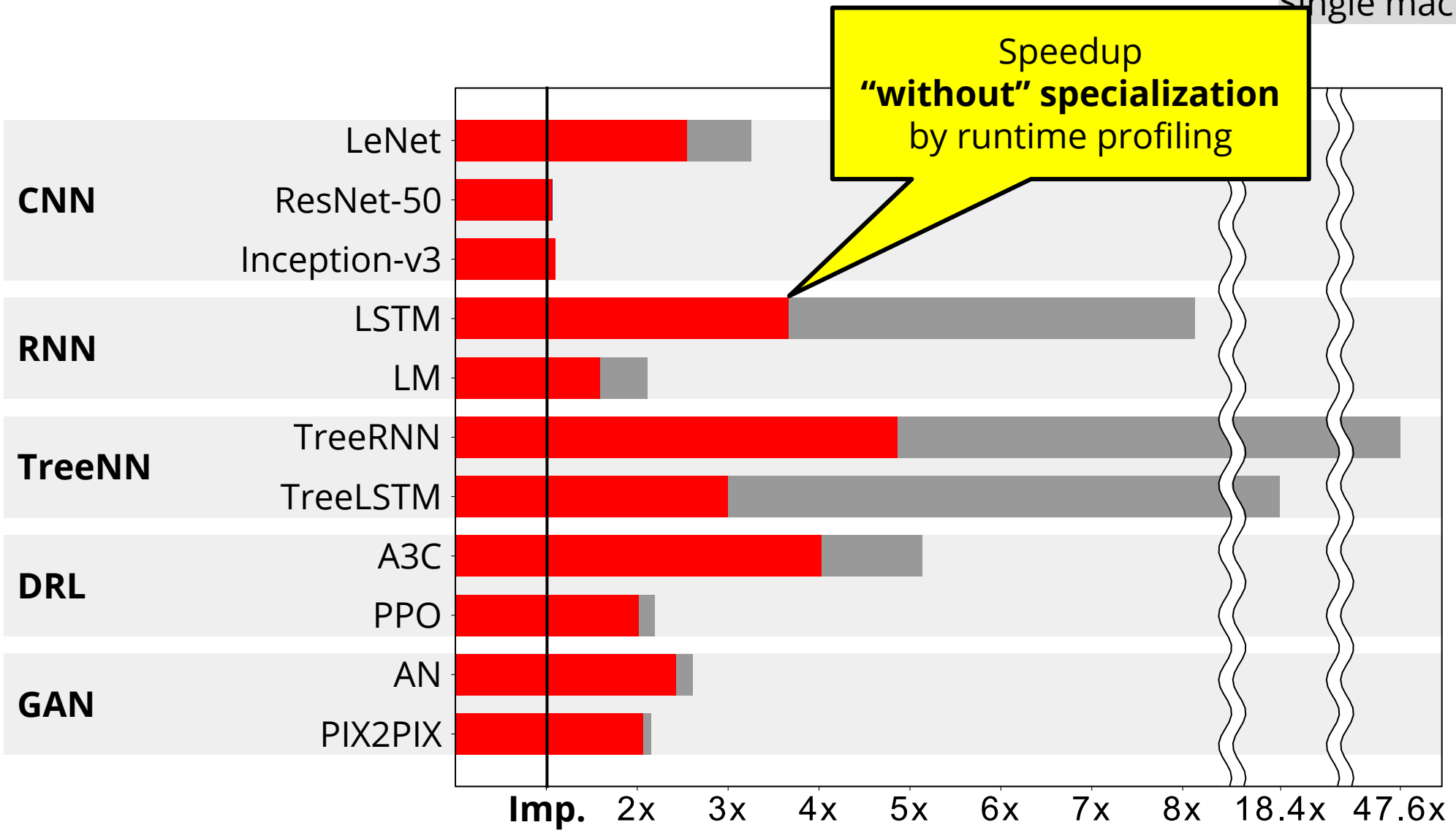
JANUS Speedup over Imperative Execution

Single machine w/ Single GPU



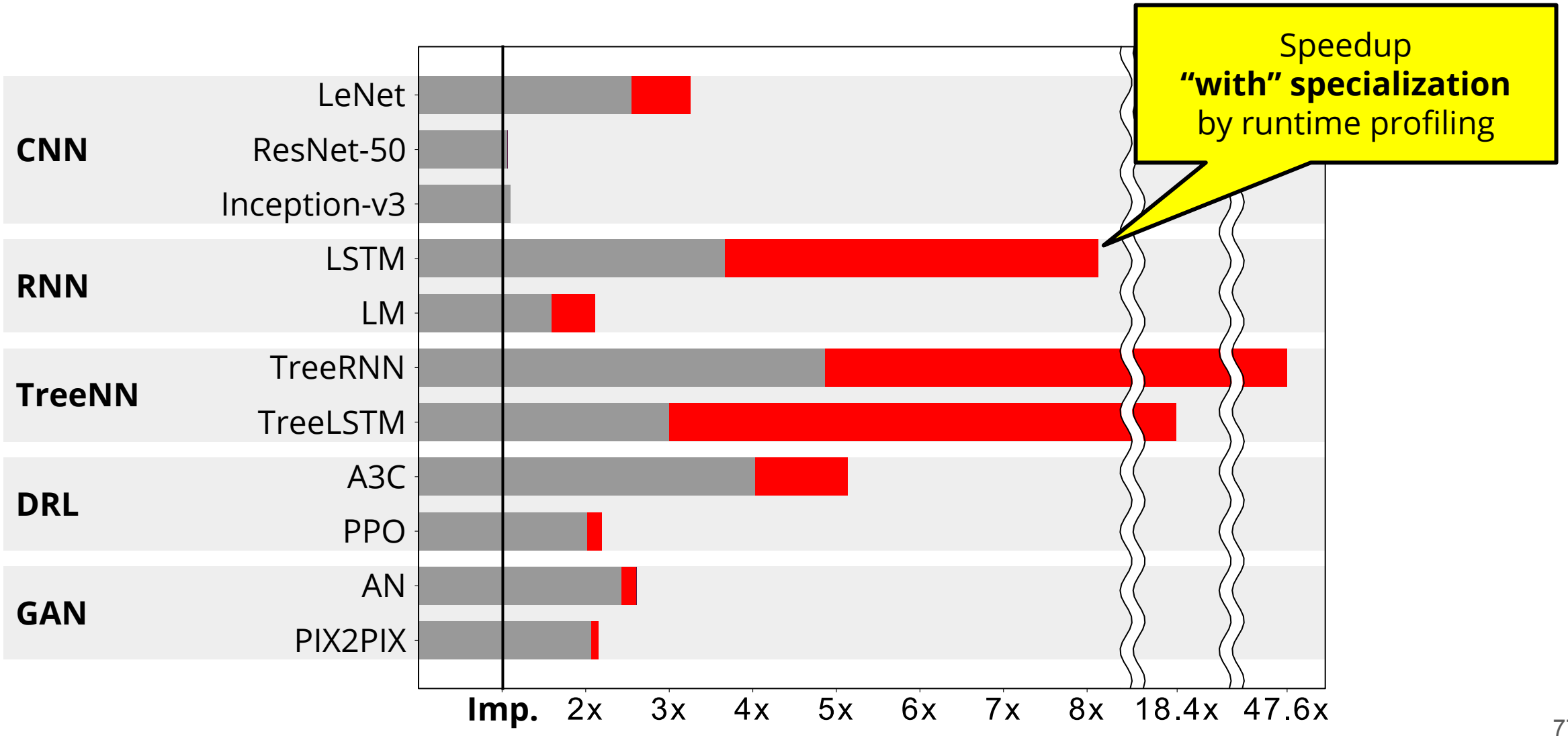
JANUS Speedup over Imperative Execution: Breakdown

Single machine w/ Single GPU

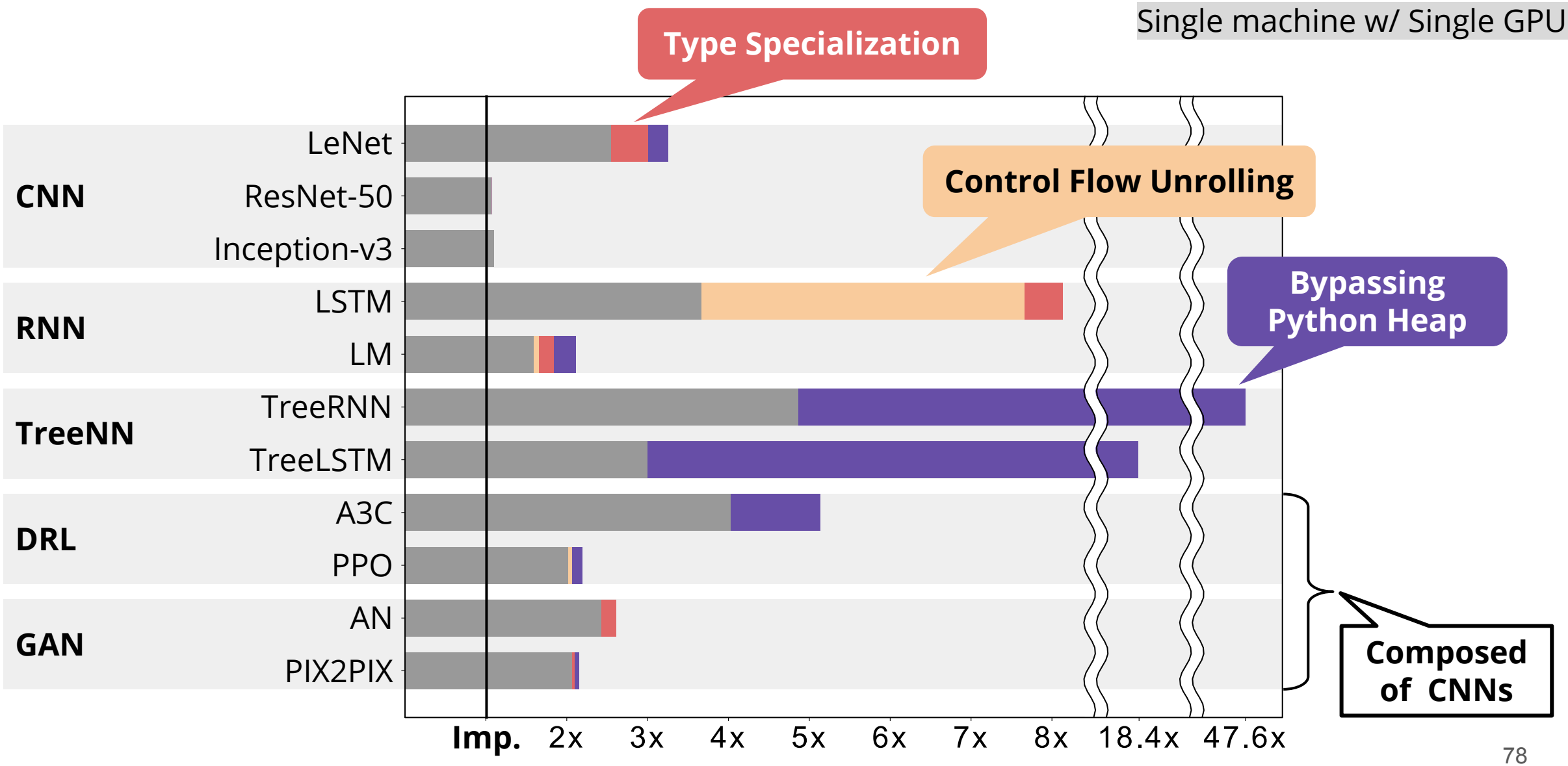


JANUS Speedup over Imperative Execution: Breakdown

Single machine w/ Single GPU



JANUS Speedup over Imperative Execution: Breakdown



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**

Conclusion

- 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