LLM Sys

Deep Learning Framework Design

Lei Li

Carnegie Mellon University
Language Technologies Institute

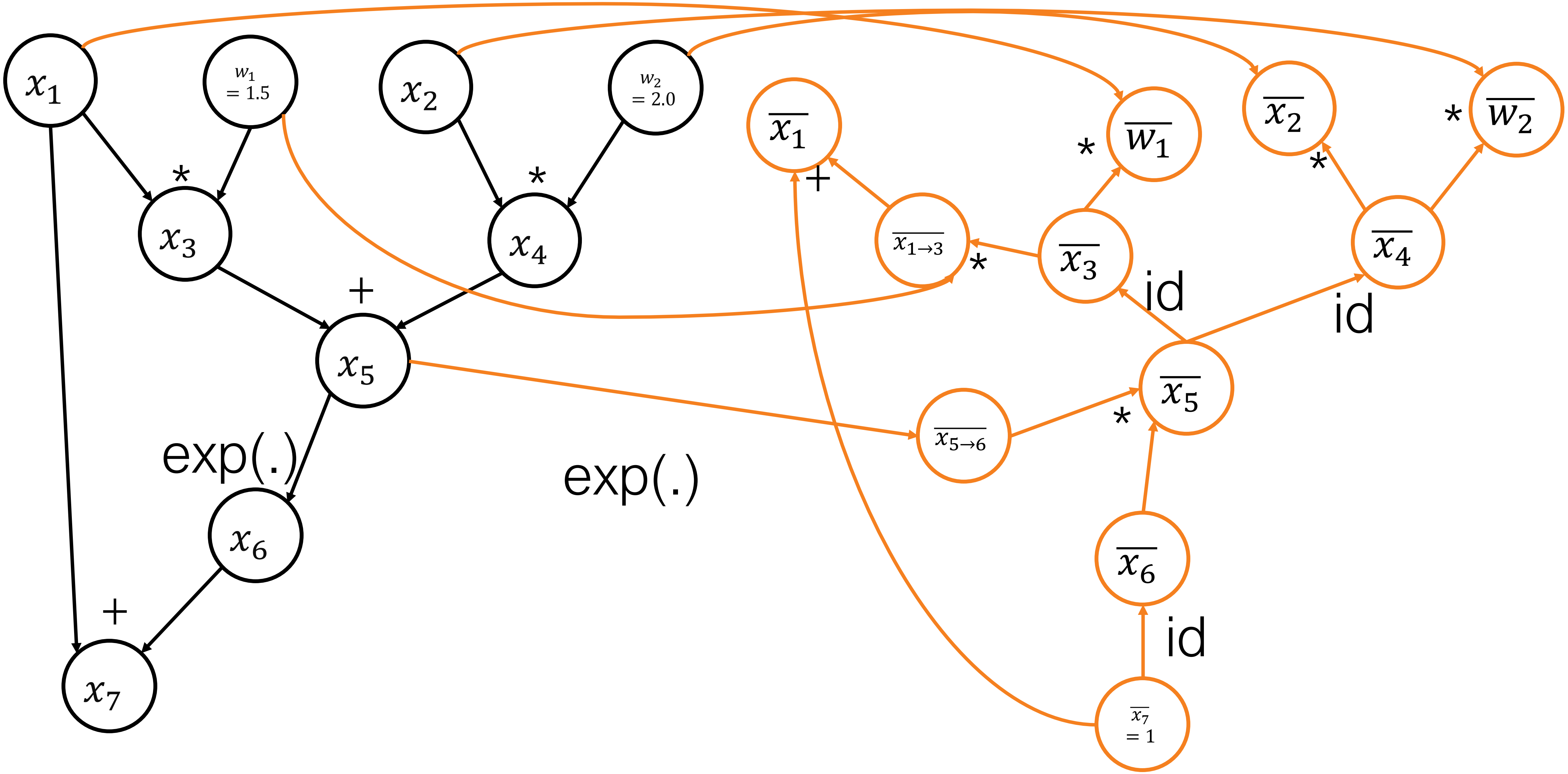
Recap

- Learning parameters of an NN needs gradient calculation
- Computation Graph
    - to perform computation: topological traversal along the DAG
- Auto Differentiation
    - building backward computation graph for gradient calculation

$x1 = 3, x2 = 0.5$

$y = x1 + \exp(1.5 * x1 + 2.0 * x2)$

Backward Computation Graph



Today’s Topic

- How to design a deep learning framework
  - Design ideas in TensorFlow
    - Abadi et al., “TensorFlow: A System for Large-Scale Machine Learning”, OSDI 2016
  - Basic Graph node types in Tensorflow/Pytorch
  - Overall design principles
- Hands-on practice to implement a mini-tensorflow
- Execution in Tensorflow

Deep Learning Frameworks (also for LLMs)

- expressive to specify any neural networks
  - support future custom operators/layers
- productive for ML engineers
  - hide low-level details (no need to write cuda)
  - automatic differentiation (no need to derive gradient calculation manually)
- efficient in large-scale training and inference
  - automatically scale to data and model size
  - automatic hardware acceleration

Deep Learning Programming Framework

- Formulate machine learning computation using data flow graphs (data moving around a computation graph)

- TensorFlow is an interface for expressing machine learning algorithms and an implementation for executing such algorithms

- PyTorch is a programming framework for tensor computation, deep learning, and auto differentiation

Aspect	PyTorch	TensorFlow	JAX	NumPy
Primary Use	Deep learning	Deep learning	numerical and ML computing	numerical computing
Programming Paradigm	Dynamic (eager execution)	Static (Graph mode, or Eager)	Functional transformations	Procedural
Autograd	dynamic comp graph	static comp graph	Functional-based with grad/jit	Not available
Hardware Support	CPU, GPU, TPU	CPU, GPU, TPU	CPU, GPU, TPU	CPU only
Ease of Use	Pythonic	a bit learning curve	Pythonic and functional	Very easy, native python
Ecosystem	PyTorch Lightning, TorchVision	TensorBoard, TensorFlow Extended	integrates with NumPy	NA
Parallelism	Multi-GPU with DataParallel or DDP	Multi-GPU/TPU via tf.distribute	Multi-GPU/TPU via pmap	No parallelism

TensorFlow

- Key idea: express a numeric computation as a computation graph
  - following what we described in last lecture
- Graph nodes are operations with any number of inputs and outputs
- Graph edges are tensors which flow between nodes
  - tensor: multidimensional array

Data as a Tensor

- A tensor is a multi-dimensional array. generalization to vector and matrix

tf.constant([[1, 2], [3, 4]])

is a 2x2 tensor with element type int32

tf.Tensor([[2 3] [4 5]], shape=(2, 2), dtype=int32)

Pytorch:

```
torch.tensor([[1., 2.], [3., 4.]])
```

Computation Graph in Tensorflow

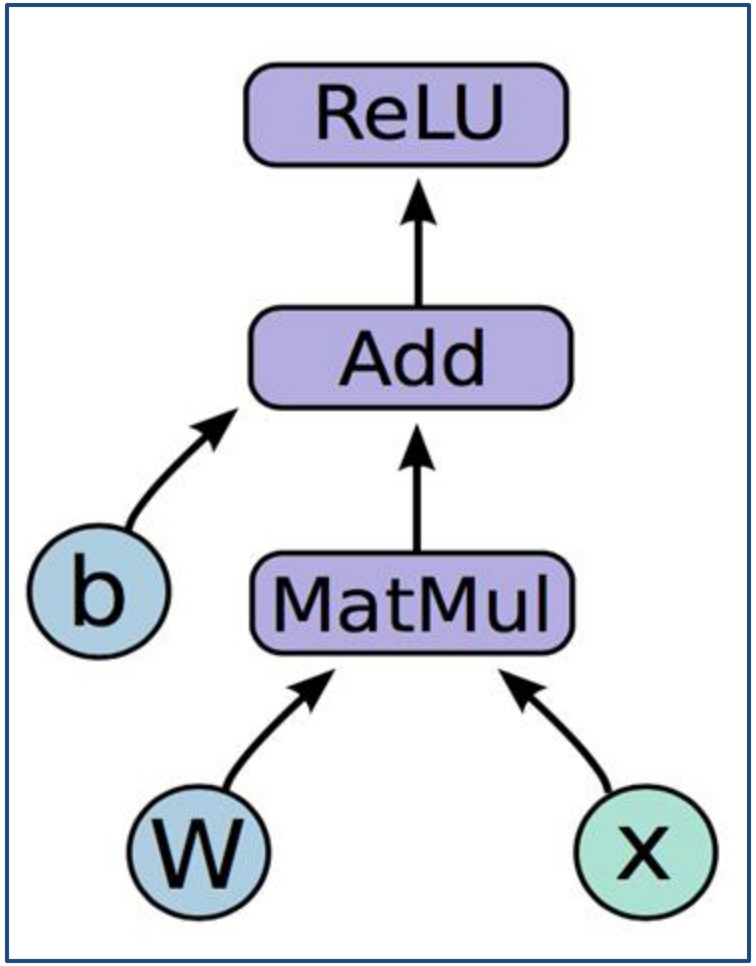
$$h = RELU(Wx + b)$$

```
import tensorflow as tf

b = tf.Variable(tf.zeros((100,)))
W = tf.Variable(tf.random_uniform((784, 100), -1, 1))

x = tf.placeholder(tf.float32, (1, 784))

h = tf.nn.relu(tf.matmul(x, W) + b)
```

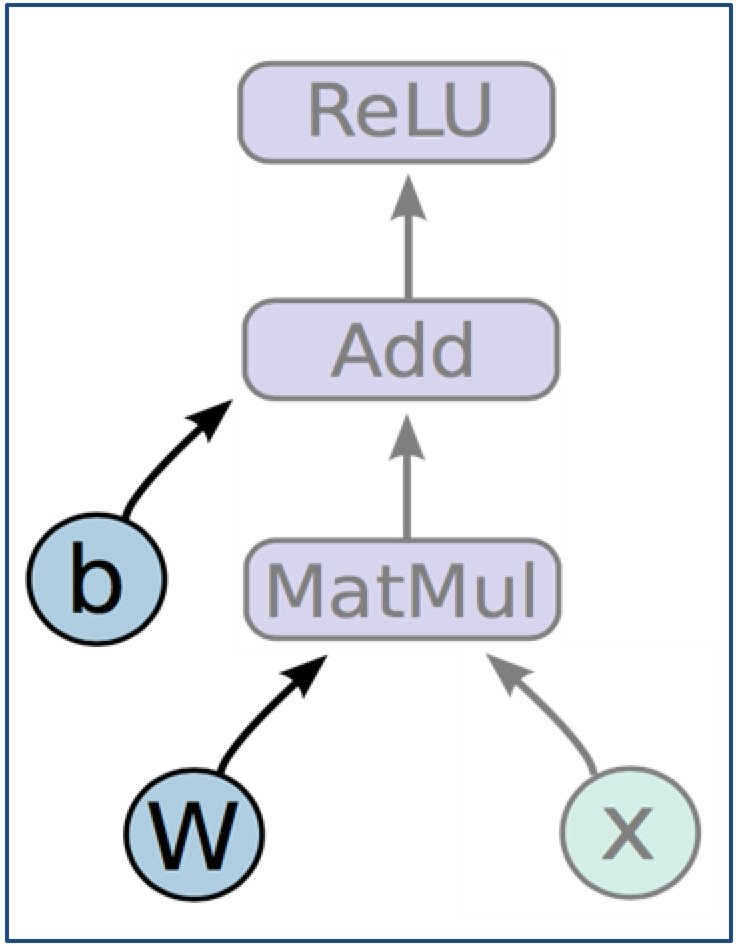


Variable Node

$$h = RELU(Wx + b)$$

```
b = tf.Variable(tf.zeros((100,)))
tf.Variable(initial_value=None,
  trainable=None,
  name=None)
```

- Variables are stateful nodes which output their current value.
- State is retained across multiple executions of a graph
- mostly parameters

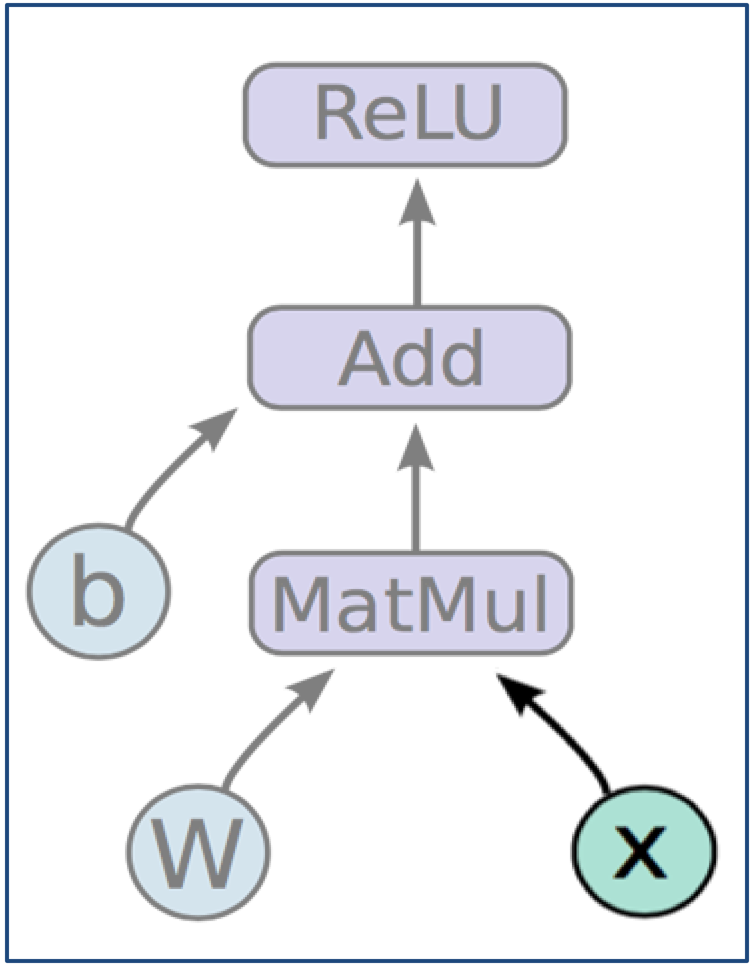


Placeholder Node (Tensorflow v1)

$$h = RELU(Wx + b)$$

```
x = tf.placeholder(tf.float32, (1, 784))
```

- Represent Inputs, Labels, …
- value is fed in at execution time
- No need to explicitly define Placeholder in Tensorflow v2



Mathematical Operations

$$h = RELU(Wx + b)$$

tf.linalg.matmul(a, b): multiply two matrices

tf.math.add(a, b): Add elementwise

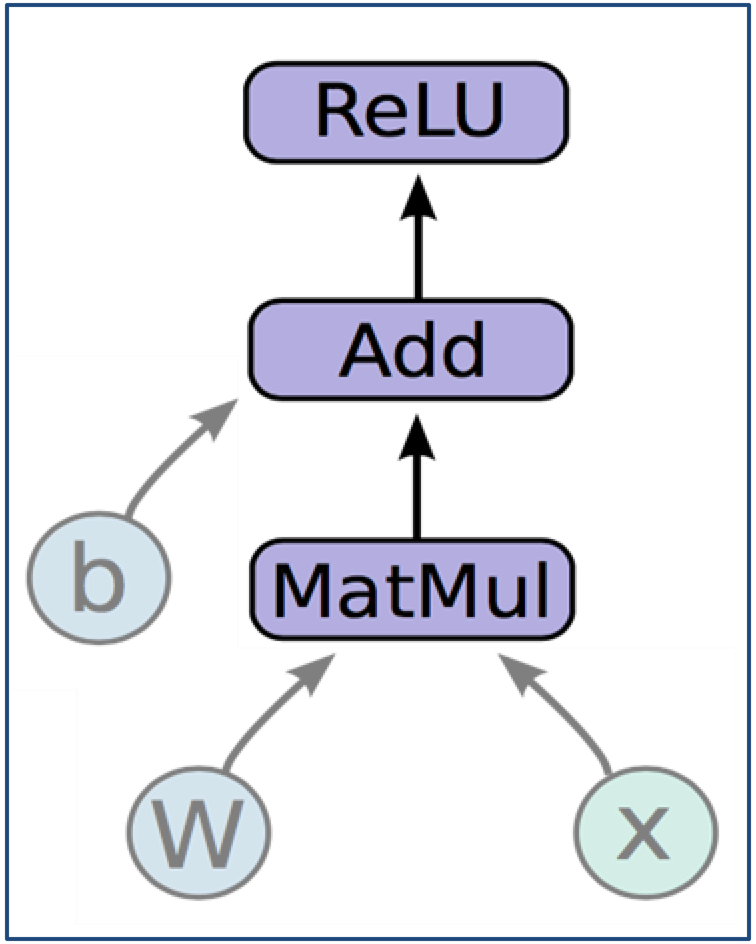
tf.nn.relu(a): Activate with elementwise rectified linear function $ReLu(x) = \begin{cases} 0, x <= 0 \\ x, x > 0 \end{cases}$

Pytorch:

torch.matmul(a, b)

torch.add(a, b)

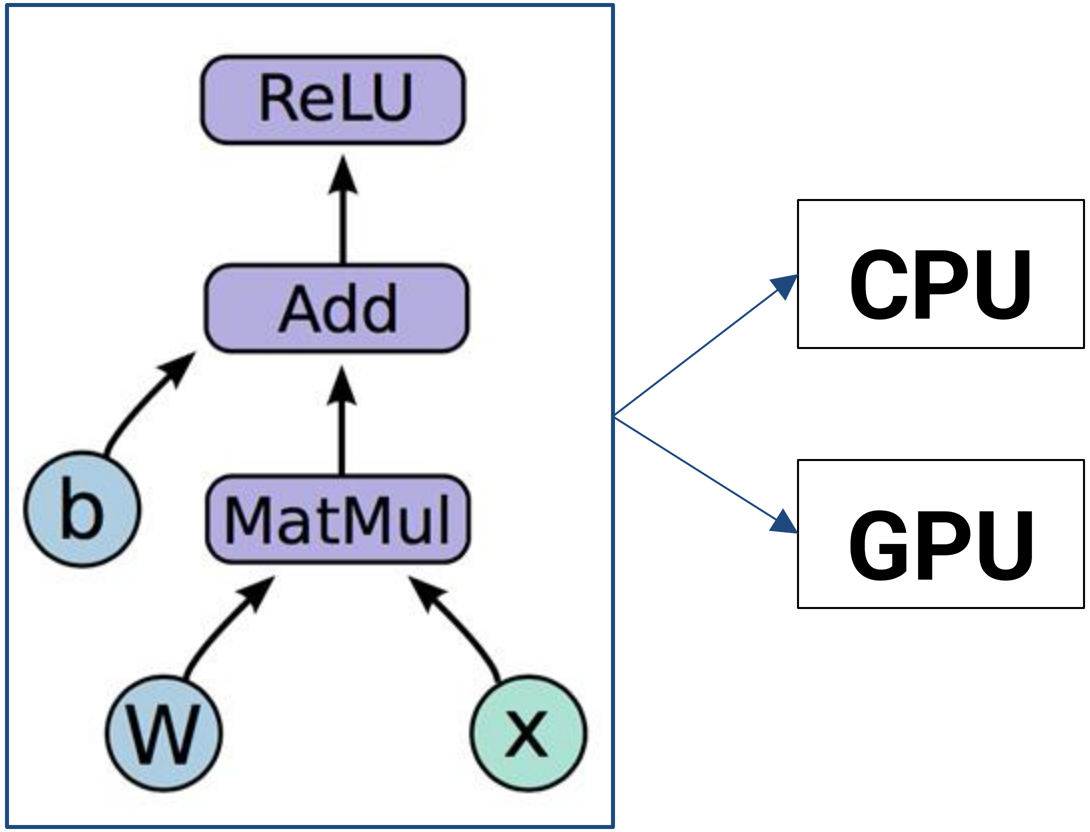
torch.nn.ReLU(a)



Running the Graph

In TF v1, to deploy graph with a session: a binding to a particular execution context (e.g. CPU, GPU)

```
with tf.Session() as s:
  …
  s.run()
```



Defining Loss

- Use **placeholder** for **labels**
- Build loss node using labels and **prediction**

```
prediction = tf.nn.softmax(...)  #Output of neural network
label = tf.placeholder(tf.float32, [100, 10])

cross_entropy = -tf.reduce_sum(label * tf.log(prediction), axis=1)
```

Gradient Computation

```
train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)
```

- tf.train.GradientDescentOptimizer is an Optimizer object
- tf.train.GradientDescentOptimizer(lr).minimize(cross_entropy) adds optimization operation to computation graph
- TensorFlow graph nodes have attached gradient operations
- Gradient with respect to parameters computed with Auto Differentiation (recall previous lecture)

Today’s Topic

- How to design a deep learning framework
  - Design ideas in TensorFlow
    - Abadi et al., “TensorFlow: A System for Large-Scale Machine Learning”, OSDI 2016
  - Basic Graph node types in Tensorflow/Pytorch
  - Overall design principles
- Hands-on practice to implement a mini-tensorflow
- Execution in Tensorflow

Core TensorFlow Constructs

- All nodes return tensors
- How a node computes is indistinguishable to TensorFlow
- In TF v1: metaprogramming - constructing the graph for the real computation. No computation occurs yet!
- TF v2 has eager mode, the computation is applied immediately (essentially constructing the graph and apply the computation)

Design Principles

- Dataflow graphs of primitive operators
- Deferred execution (two phases)
  1. Define program i.e., symbolic dataflow graph w/ placeholders, essentially constructing the computation graph
  2. Executes optimized version of program on set of available devices

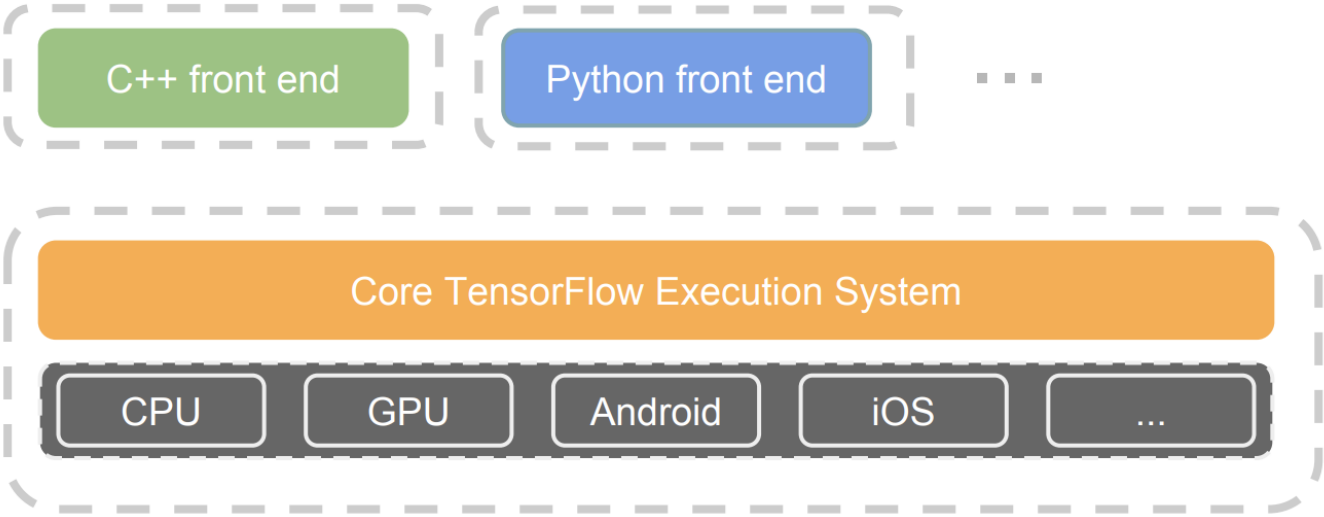
Dynamic Flow Control

- Problem: support ML algos that contain conditional and iterative control flow, e.g.
  - Recurrent Neural Networks (RNNs) and LSTMs
  - Autoregressive decoder
- Solution: Add conditional (if statement) and iterative (while loop) programming constructs

TensorFlow Architecture

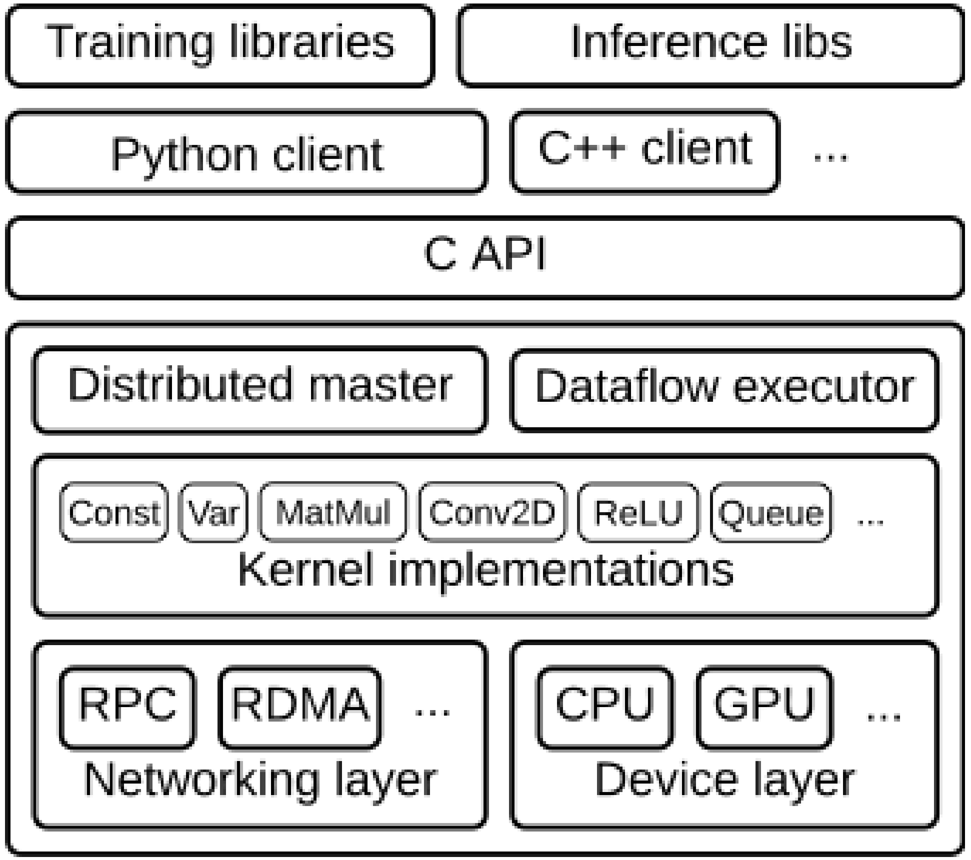
- Core in C++
  - Very low overhead

- Different front ends for specifying/driving the computation
  - Python and C++, easy to add more



TensorFlow Implementation

- Semi-interpreted
- Call GPU kernel per primitive operation
- Can batch operations with custom C++
- Basic type-safety within dataflow graph (error at graph construction time)



https://www.tensorflow.org/extend/architecture

Code Practice: Implement Computation Graph

https://github.com/llmsystem/llmsys_code_examples/tree/main/mini_tensorflow

Please follow the instructions and fill in the code in

https://github.com/llmsystem/llmsys_code_examples/blob/main/mini_tensorflow/mini_tensorflow.ipynb

The full code is provided in

https://github.com/llmsystem/llmsys_code_examples/blob/main/mini_tensorflow/mini_tensorflow_full.ipynb

Today's Topic

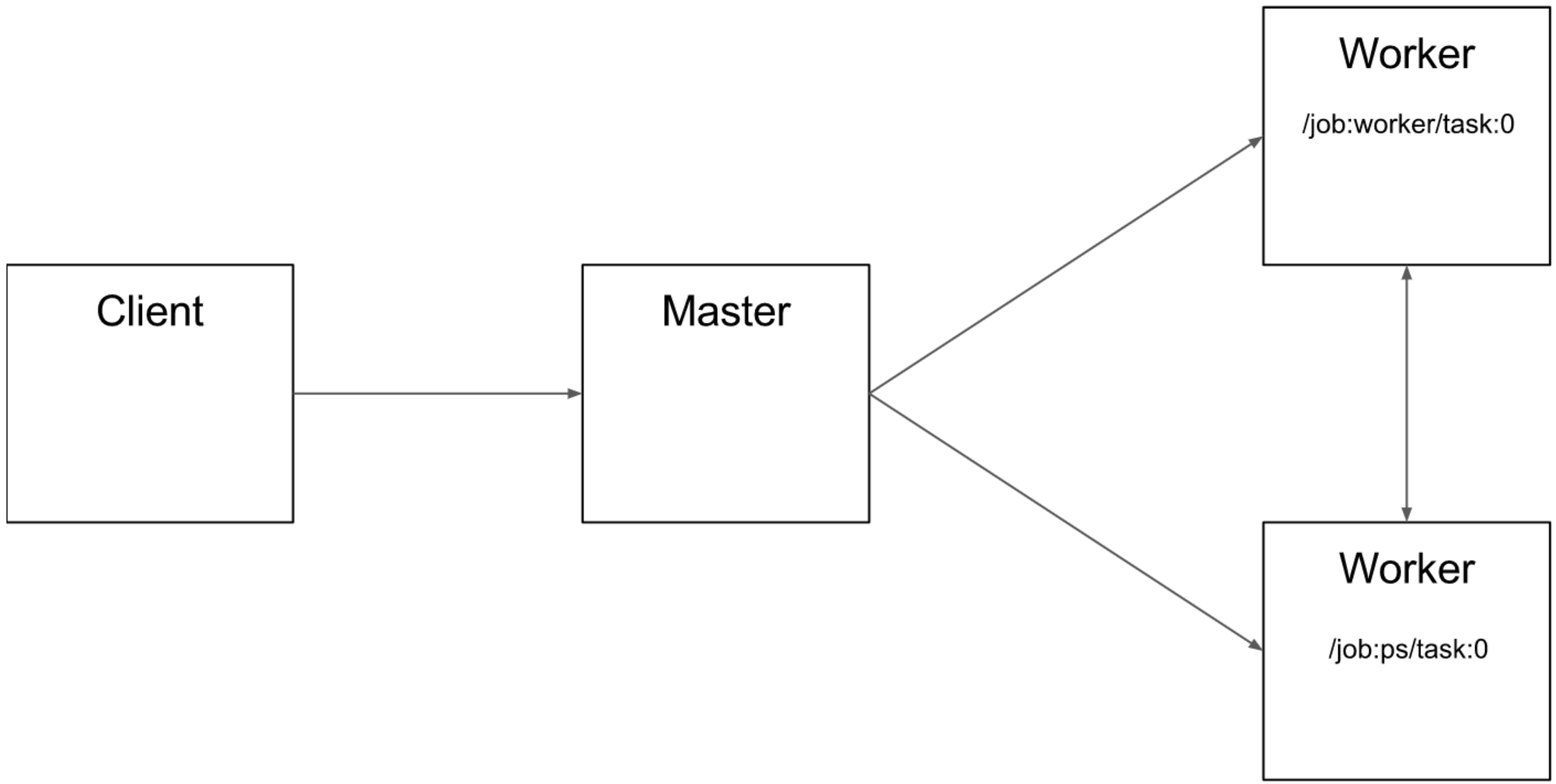
- How to design a deep learning framework
  - Design ideas in TensorFlow
    - Abadi et al., "TensorFlow: A System for Large-Scale Machine Learning", OSDI 2016
  - Basic Graph node types in Tensorflow/Pytorch
  - Overall design principles

- Hands-on practice to implement a mini-tensorflow

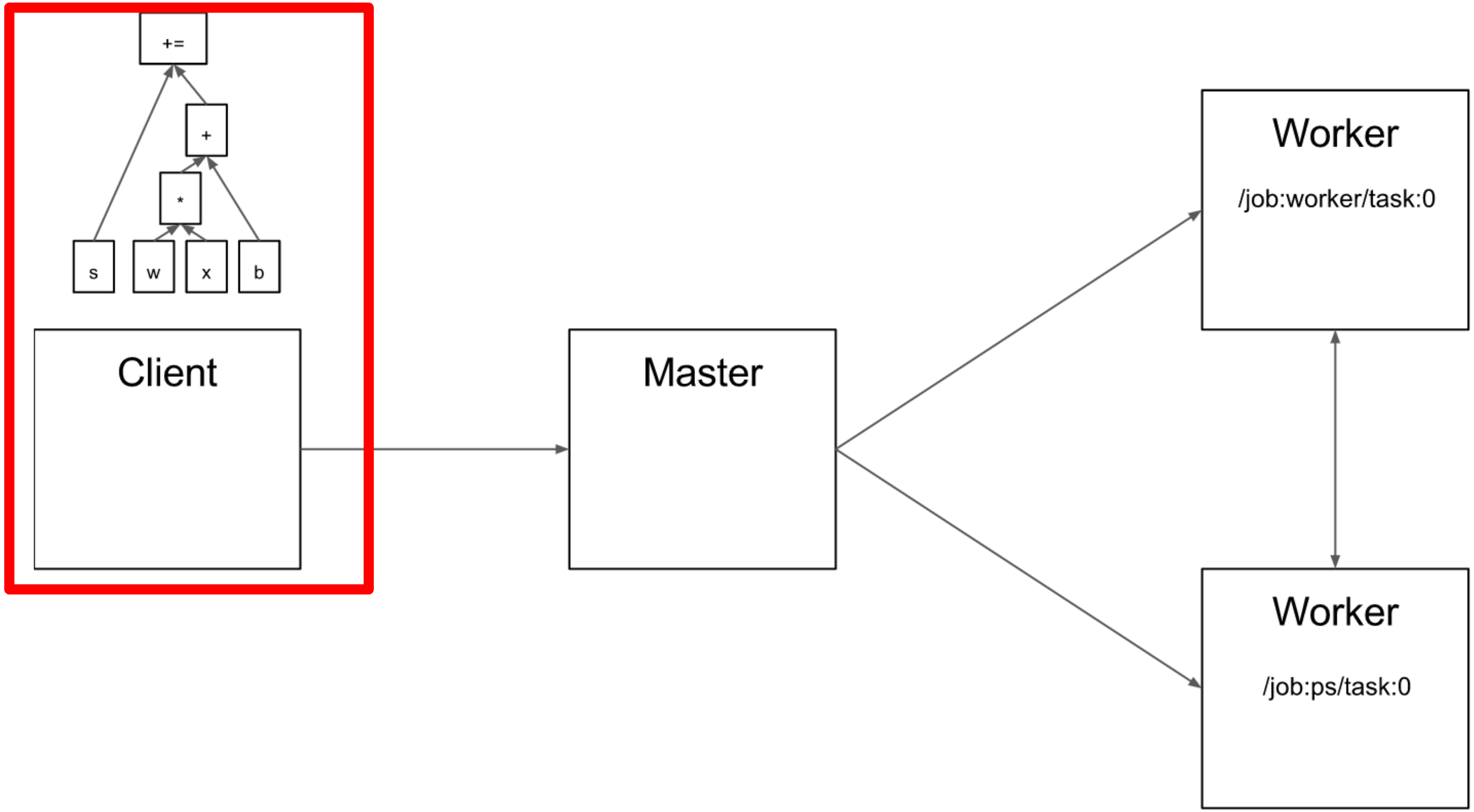
- Execution in Tensorflow

Tensorflow Execution Key Components

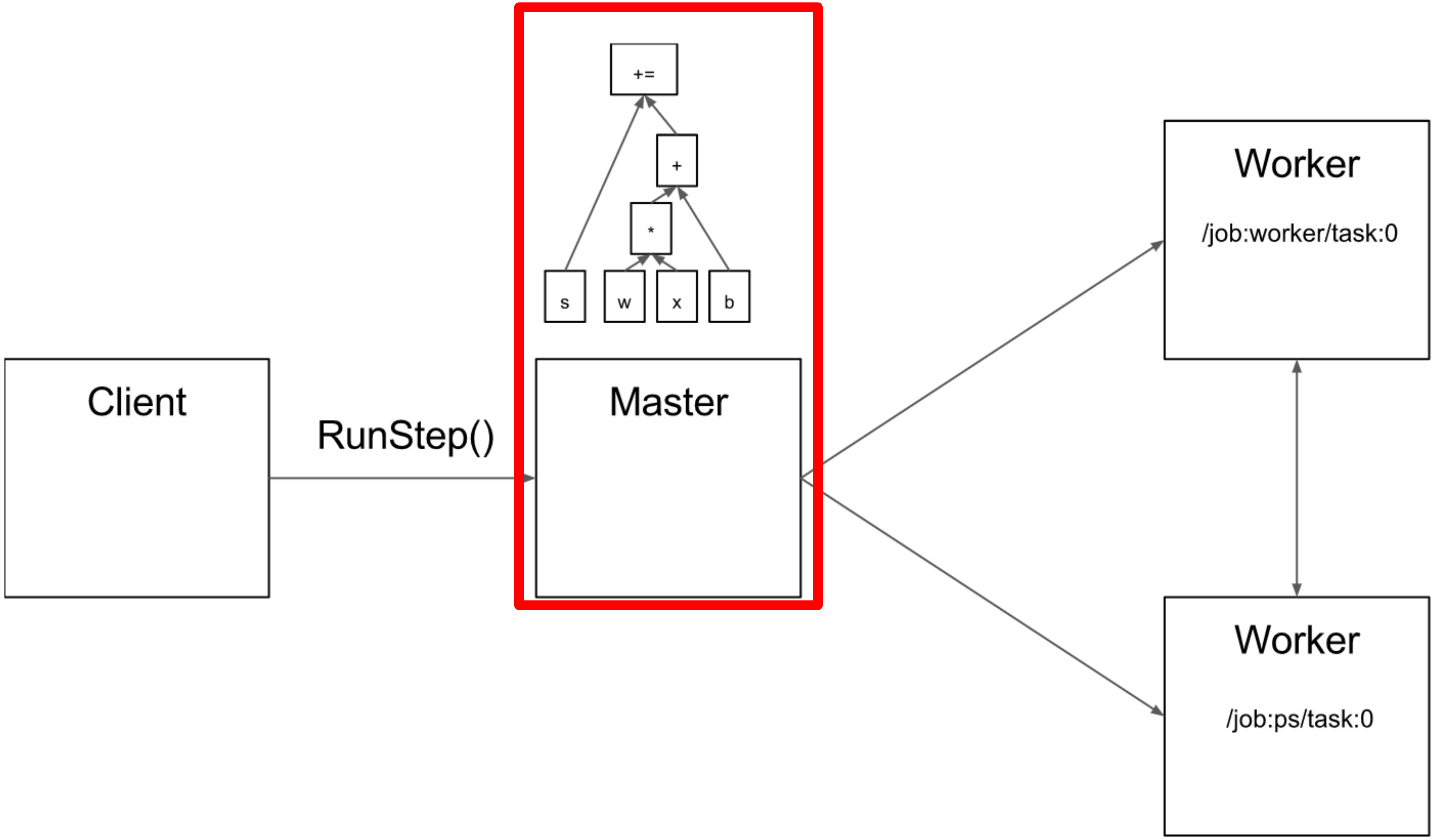
- Similar to MapReduce, Apache Hadoop, Apache Spark, …



Client

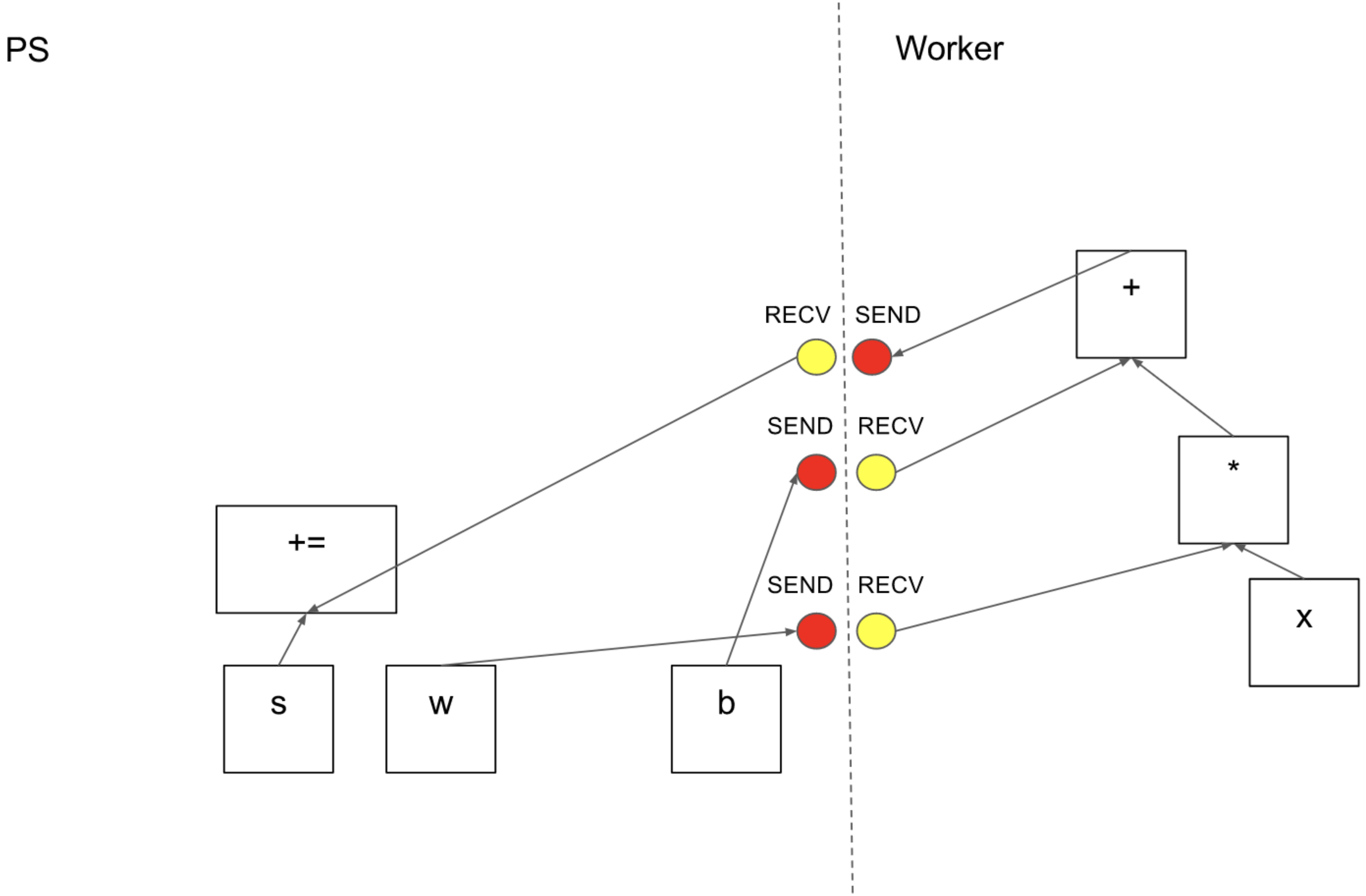

+=
+
*
s
w
x
b
Client
Master
Worker
/job:worker/task:0
Worker
/job:ps/task:0

Master

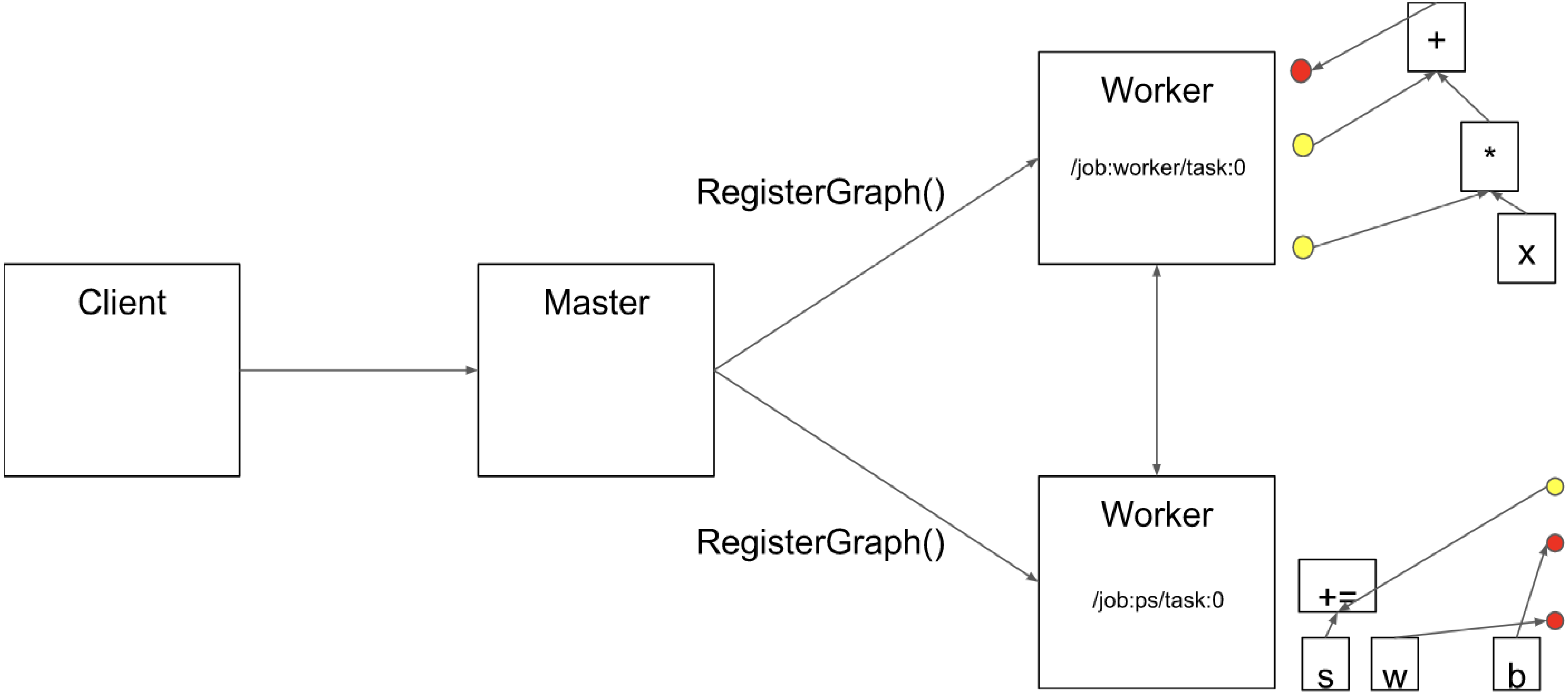


+=
+
*
s
w
x
b
Client
RunStep()
Master
Worker
/job:worker/task:0
Worker
/job:ps/task:0

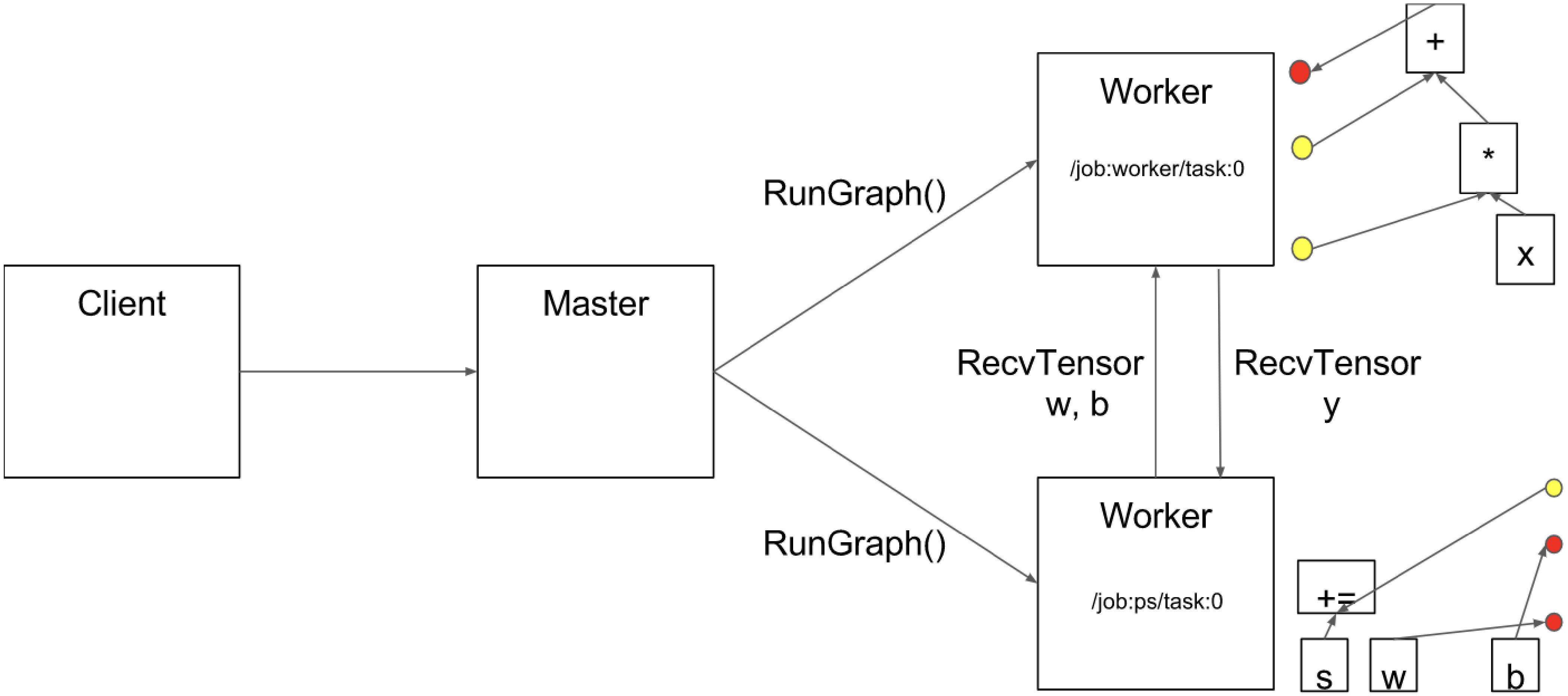
Computation Graph Partition



Computation Graph Partition



Execution


Client
Master
RunGraph()
RunGraph()
Worker
/job:worker/task:0
Worker
/job:ps/task:0
RecvTensor
w, b
RecvTensor
y
+
*
x
+=
s
w
b

Synchronous vs Asynchronous

- Determined by node: Queue nodes used for barriers
- Synchronous nearly as fast as asynchronous
- Default model is asynchronous

Fault Tolerance

- Assumptions:
  - Fine grain operations: “It is unlikely that tasks will fail so often that individual operations need fault tolerance” ;-)
  - “Many learning algorithms do not require strong consistency”
- Solution: user-level checkpointing (provides 2 ops)
  - save(): writes one or more tensors to a checkpoint file
  - restore(): reads one or more tensors from a checkpoint file

Performance

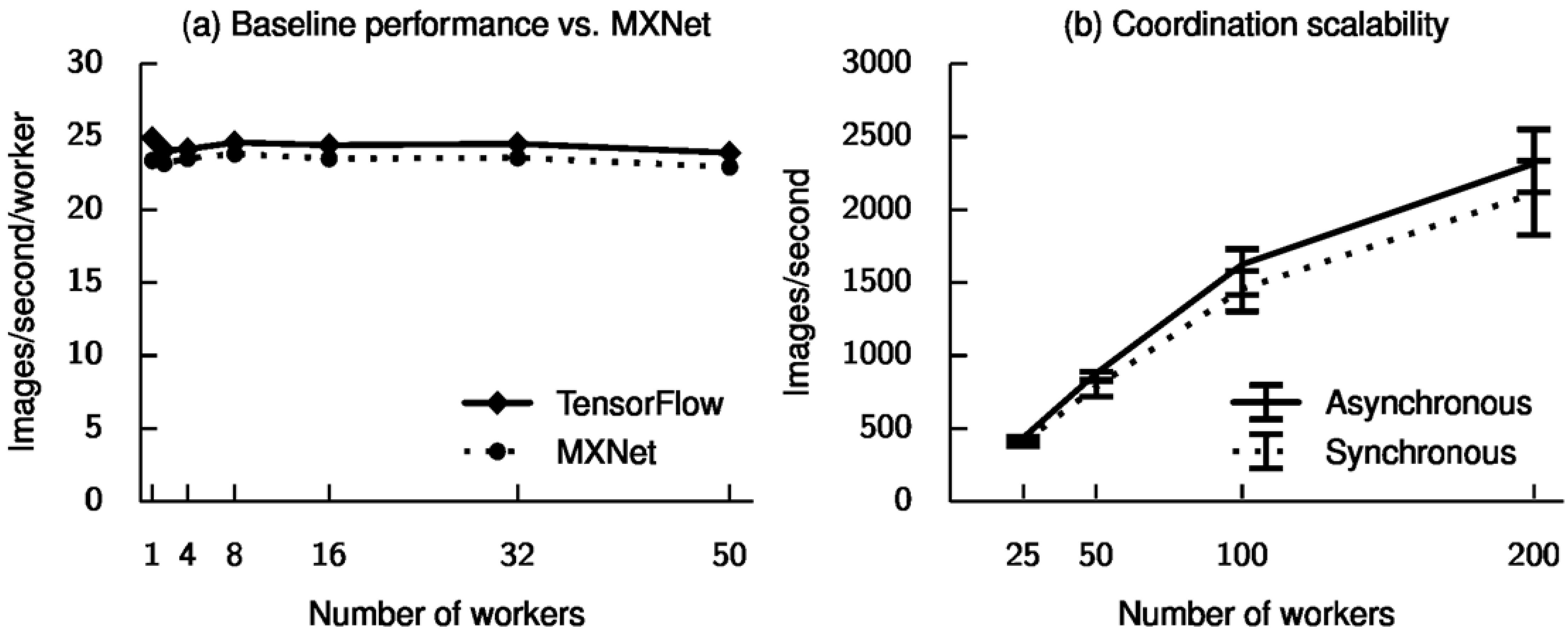
- Single Node

Library	Training step time (ms)			
	AlexNet	Overfeat	OxfordNet	GoogleNet
Caffe [38]	324	823	1068	1935
Neon [58]	87	**211**	**320**	**270**
Torch [17]	**81**	268	529	470
TensorFlow	**81**	279	540	445

Abadi et al., "TensorFlow: A System for Large-Scale Machine Learning", OSDI 2016

Performance

- Distributed Throughput



Abadi et al., "TensorFlow: A System for Large-Scale Machine Learning", OSDI 2016

Summary

- Key Contributions
  - Programmability/abstraction
  - Accessibility / ease of use

- Deferred execution:

  1. Define program i.e., symbolic dataflow graph w/ placeholders, essentially constructing the computation graph
  2. Executes (optimized) computation graph on set of available devices

Abadi et al., "TensorFlow: A System for Large-Scale Machine Learning", OSDI 2016