TensorFlow and Clipper (Lecture 24, cs262a)

Ali Ghodsi and Ion Stoica,
UC Berkeley
April 18, 2018

Today's lecture

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

(https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf)

Crankshaw et al., "Clipper: A Low-Latency Online Prediction Serving System", NSDI 2017

(https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/crankshaw)

A short history of Neural Networks

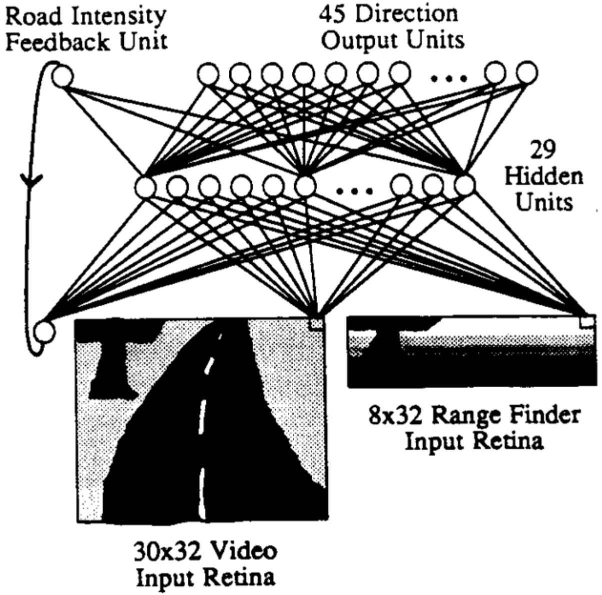
1957: Perceptron (Frank Rosenblatt): one layer network neural network

1959: first neural network to solve a real world problem, i.e., eliminates echoes on phone lines (Widrow & Hoff)

1988: Backpropagation (Rumelhart, Hinton, Williams): learning a multi-layered network

A short history of NNs

1989: ALVINN: autonomous driving car using NN (CMU)

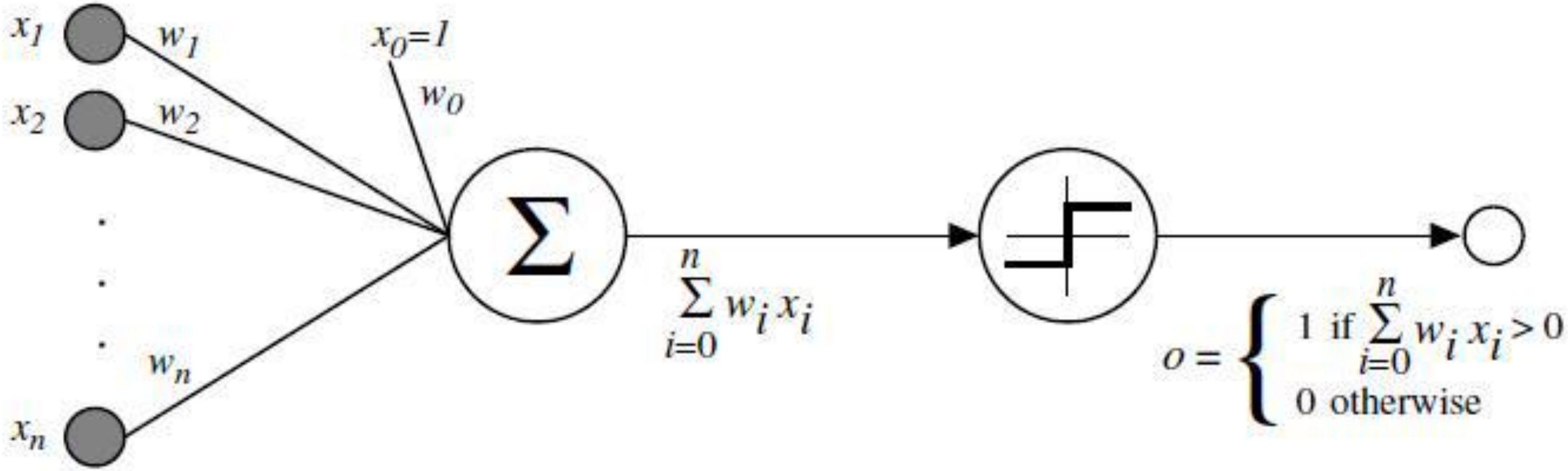


1989: (LeCun) Successful application to recognize handwritten ZIP codes on mail using a “deep” network

2010s: near-human capabilities for image recognition, speech recognition, and language translation

Perceptron

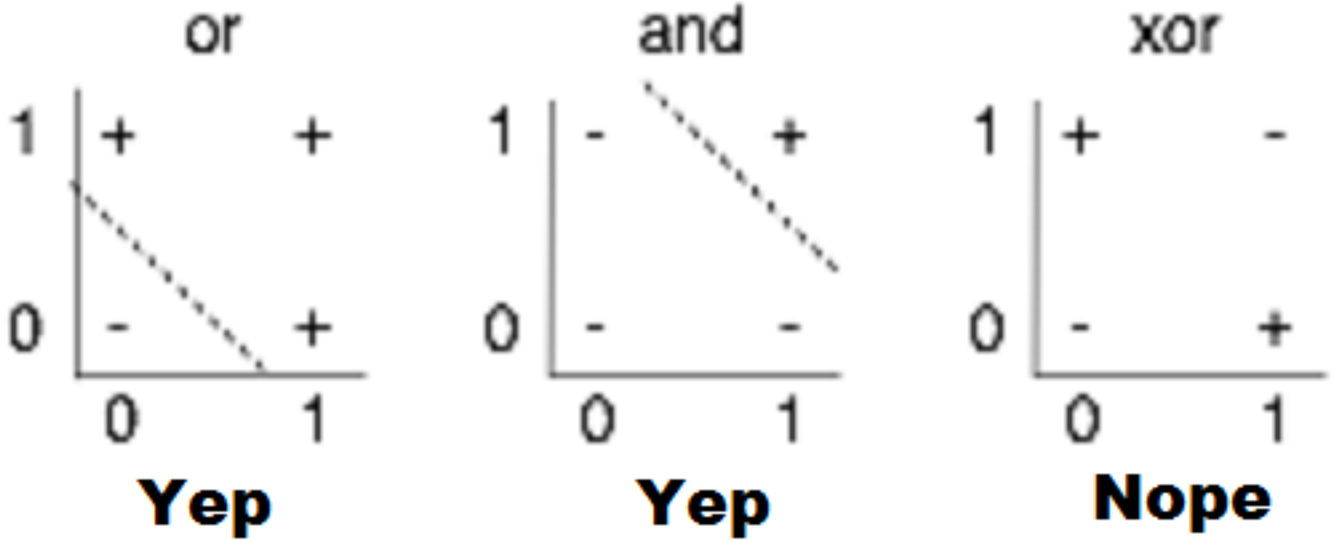
Invented by Frank Rosenblatt (1957): simplified mathematical model of how the neurons in our brains operate



From: http://www.andreykurenkov.com/writing/ai/a-brief-history-of-neural-nets-and-deep-learning/

Perceptron

Could implement AND, OR, but not XOR

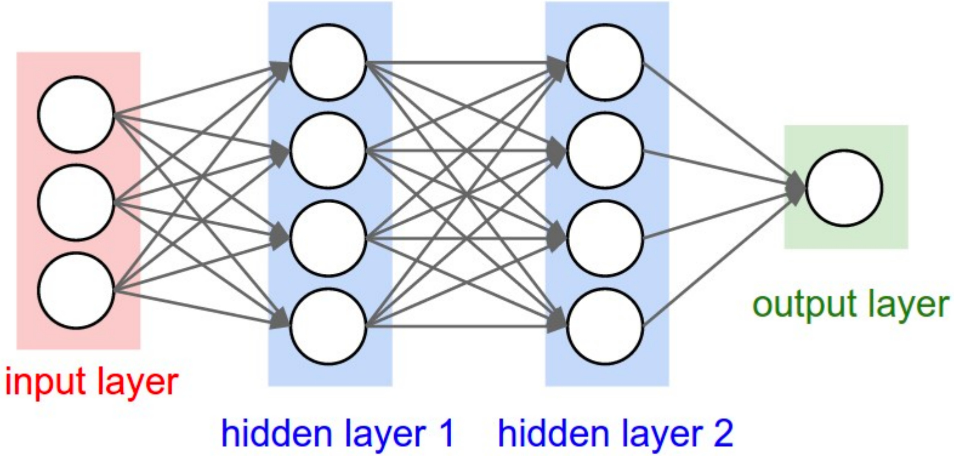


From: http://www.andreykurenkov.com/writing/ai/a-brief-history-of-neural-nets-and-deep-learning/

Hidden layers

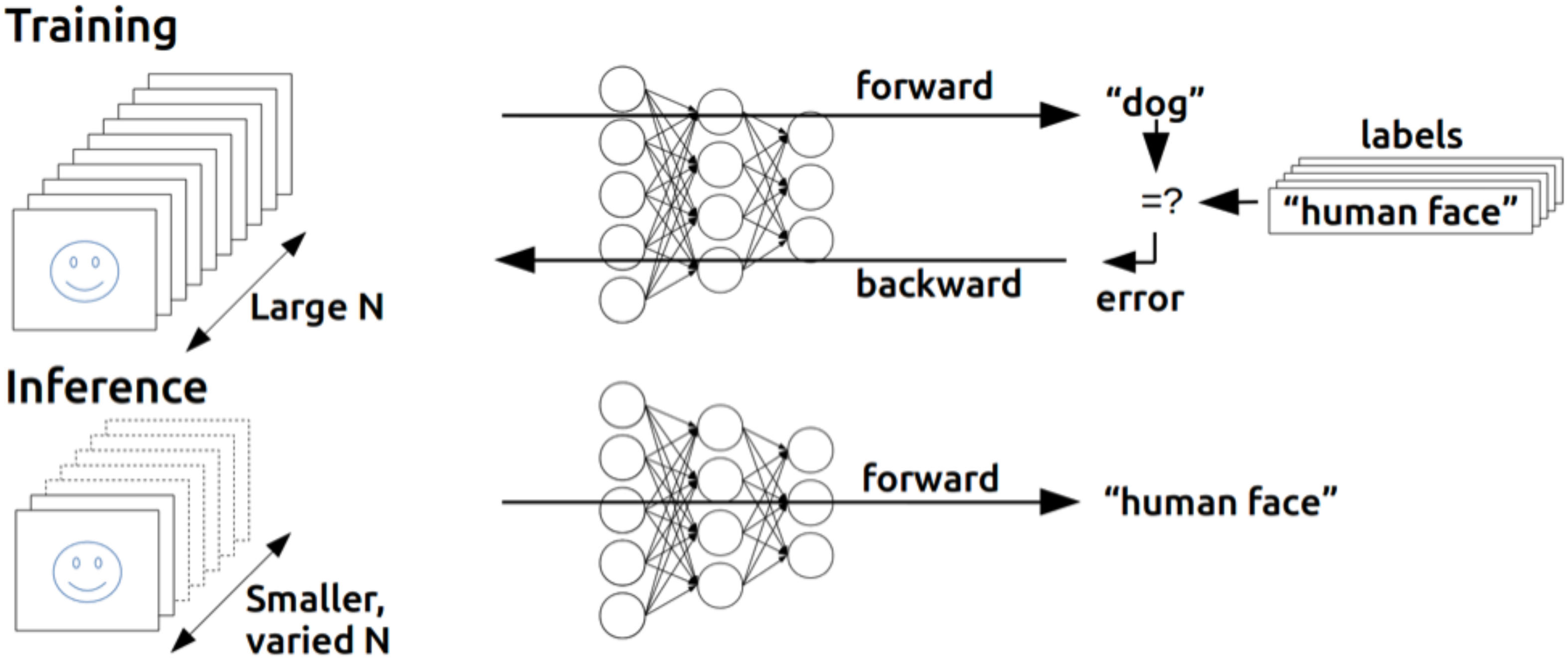
Hidden layers can find **features** within the data and allow following layers to operate on those features

- Can implement XOR



From: http://www.andreykurenkov.com/writing/ai/a-brief-history-of-neural-nets-and-deep-learning/

Learning: Backpropagation



From: http://www.andreykurenkov.com/writing/ai/a-brief-history-of-neural-nets-and-deep-learning/

Context (circa 2015)

Deep learning already claiming big successes

Team	Year	Place	Error (top-5)
XRCE (pre-neural-net explosion)	2011	1st	25.8%
Supervision (AlexNet)	2012	1st	16.4%
Clarifai	2013	1st	11.7%
GoogLeNet (Inception)	2014	1st	6.66%
Andrej Karpathy (human)	2014	N/A	5.1%
BN-Inception (Arxiv)	2015	N/A	4.9%
Inception-v3 (Arxiv)	2015	N/A	3.46%

Imagenet challenge classification task

From: http://www.wsdm-conference.org/2016/slides/WSDM2016-Jeff-Dean.pdfz

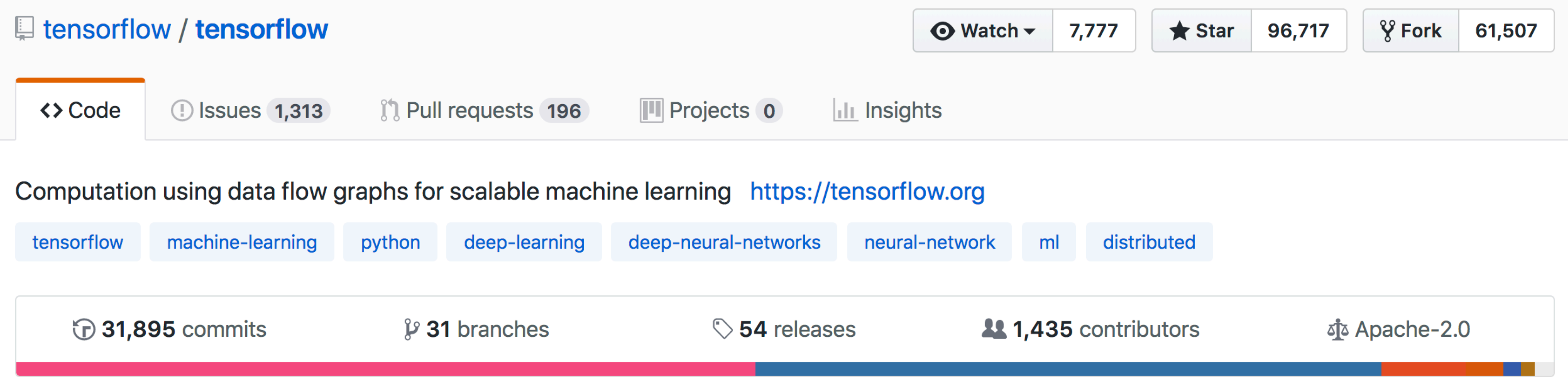
Context (circa 2015)

Deep learning already claiming big successes

Number of developers/researchers exploding

A “zoo” of tools and libraries, some of questionable quality…

What is TensorFlow?



Open source library for numerical computation using **data flow graphs**

Developed by Google Brain Team to conduct machine learning research

- Based on DisBelief used internally at Google since 2011

“TensorFlow is an interface for expressing machine learning algorithms, and an implementation for executing such algorithms”

What is TensorFlow

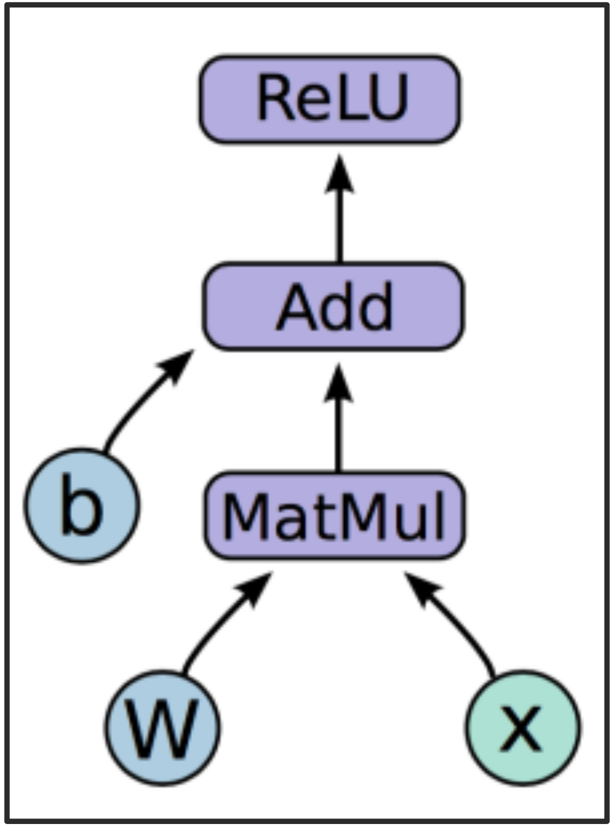
Key idea: express a numeric computation as a **graph**

Graph nodes are **operations** with any number of inputs and outputs

Graph edges are **tensors** which flow between nodes

Programming model

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

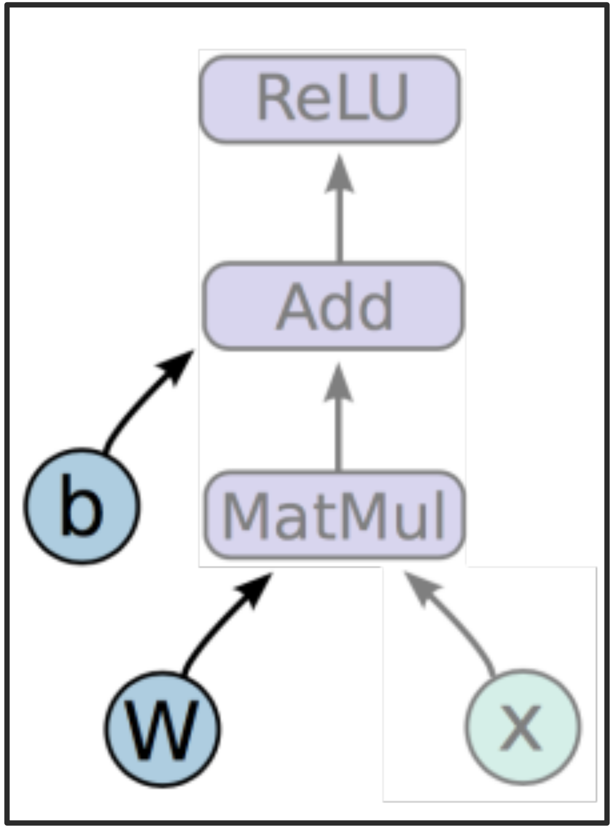


Programming model

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

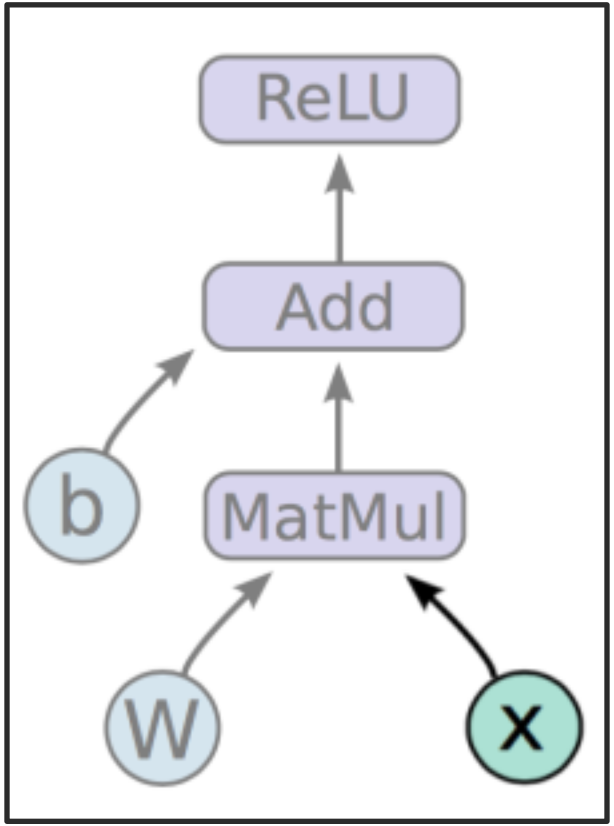
Variables are stateful nodes which output their current value. State is retained across multiple executions of a graph

(mostly parameters)



Programming model

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



Placeholders are nodes whose value is fed in at execution time

(inputs, labels, …)

Programming model

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

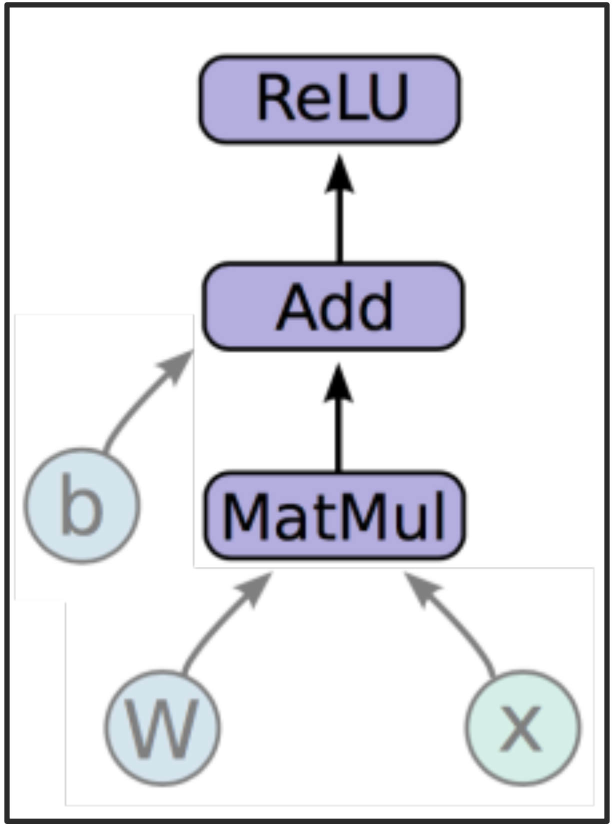
Mathematical operations:

MatMul: Multiply two matrices

Add: Add elementwise

ReLU: Activate with elementwise rectified linear function

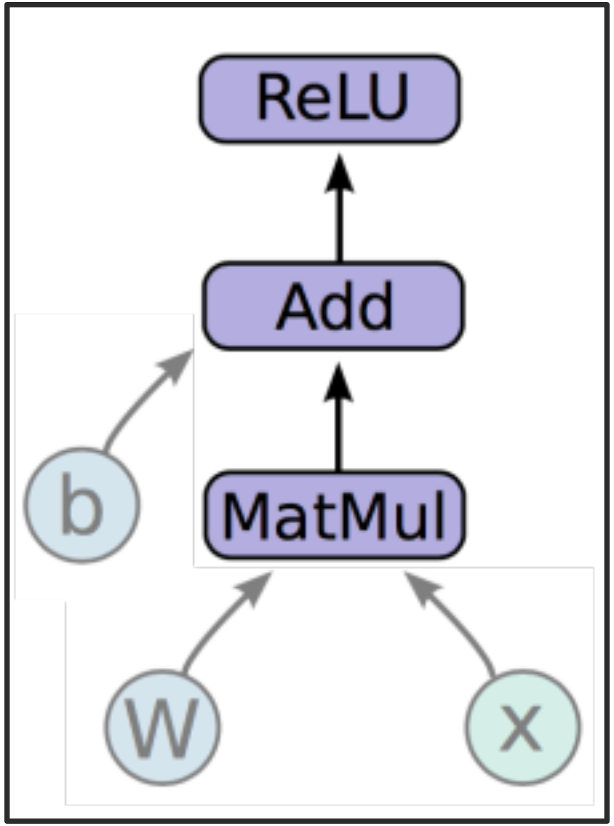
$$ReLu(x) = \begin{cases} 0, x <= 0 \\ x, x > 0 \end{cases}$$



Code

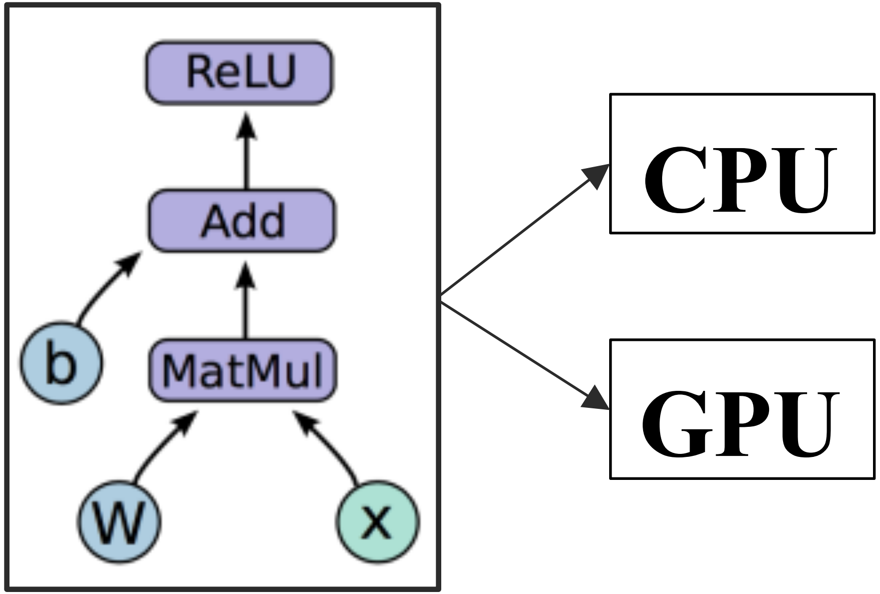
```python
import tensorflow as tf

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)
```

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



Running the graph

Deploy graph with a **session**: a binding to a particular execution context (e.g. CPU, GPU)



End-to-end

So far:

- Built a **graph** using **variables** and **placeholders**
- Deploy the graph onto a **session**, i.e., **execution environment**

Next: train model

- Define loss function
- Compute gradients

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: Backpropagation

```
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 **backpropagation … automatically**

Design Principles

Dataflow graphs of primitive operators

Deferred execution (two phases)

1. Define program i.e., symbolic dataflow graph w/ placeholders
2. Executes optimized version of program on set of available devices

Common abstraction for heterogeneous accelerators

1. Issue a kernel for execution
2. Allocate memory for inputs and outputs
3. Transfer buffers to and from host memory

Dynamic Flow Control

Problem: support ML algos that contain conditional and iterative control flow, e.g.

- Recurrent Neural Networks (RNNs)
- Long-Short Term Memory (LSTM)

Solution: Add conditional (if statement) and iterative (while loop) programming constructs

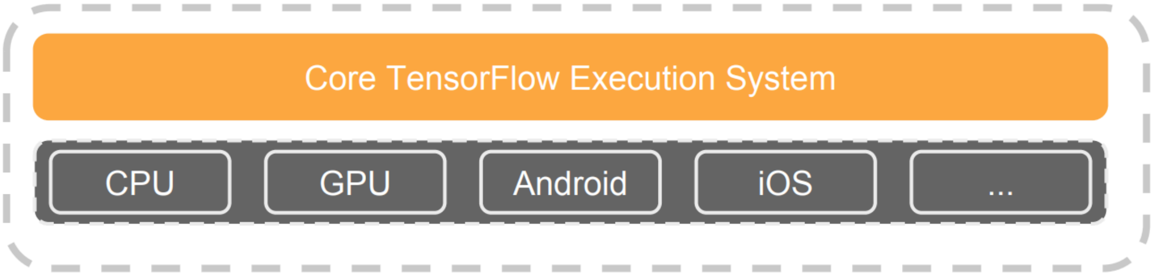
TensorFlow high-level architecture

Core in C++

- Very low overhead

Different front ends for specifying/driving the computation

- Python and C++ today, easy to add more



From: http://www.wsdm-conference.org/2016/slides/WSDM2016-Jeff-Dean.pdf

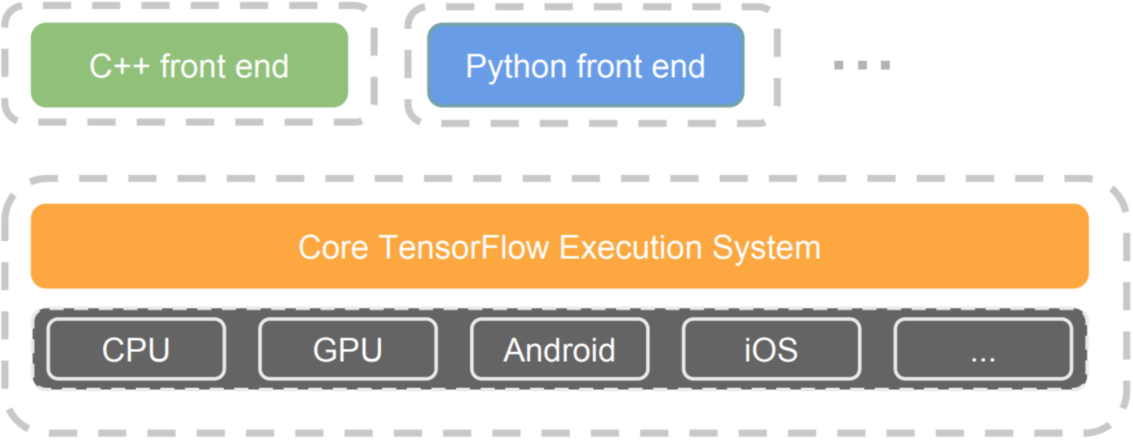
TensorFlow architecture

Core in C++

- Very low overhead

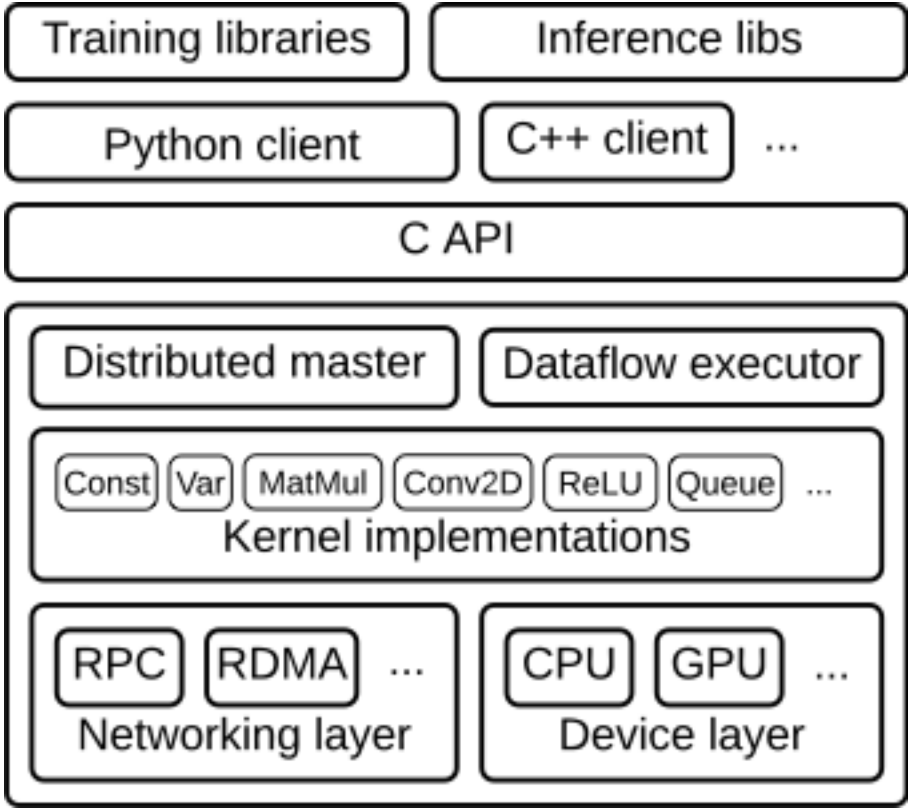
Different front ends for specifying/driving the computation

- Python and C++ today, easy to add more



From: http://www.wsdm-conference.org/2016/slides/WSDM2016-Jeff-Dean.pdf

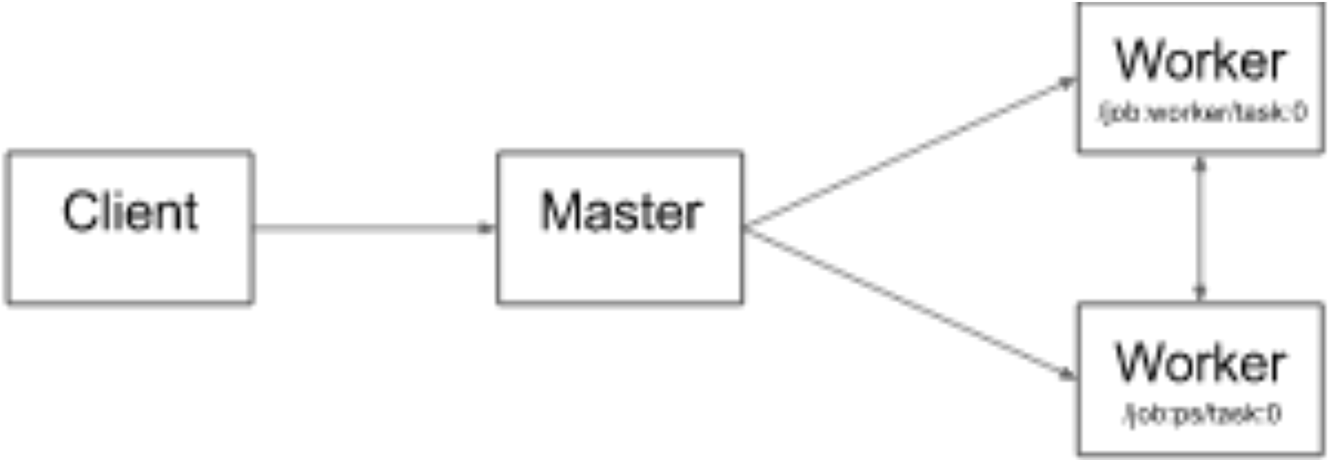
Detailed architecture



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

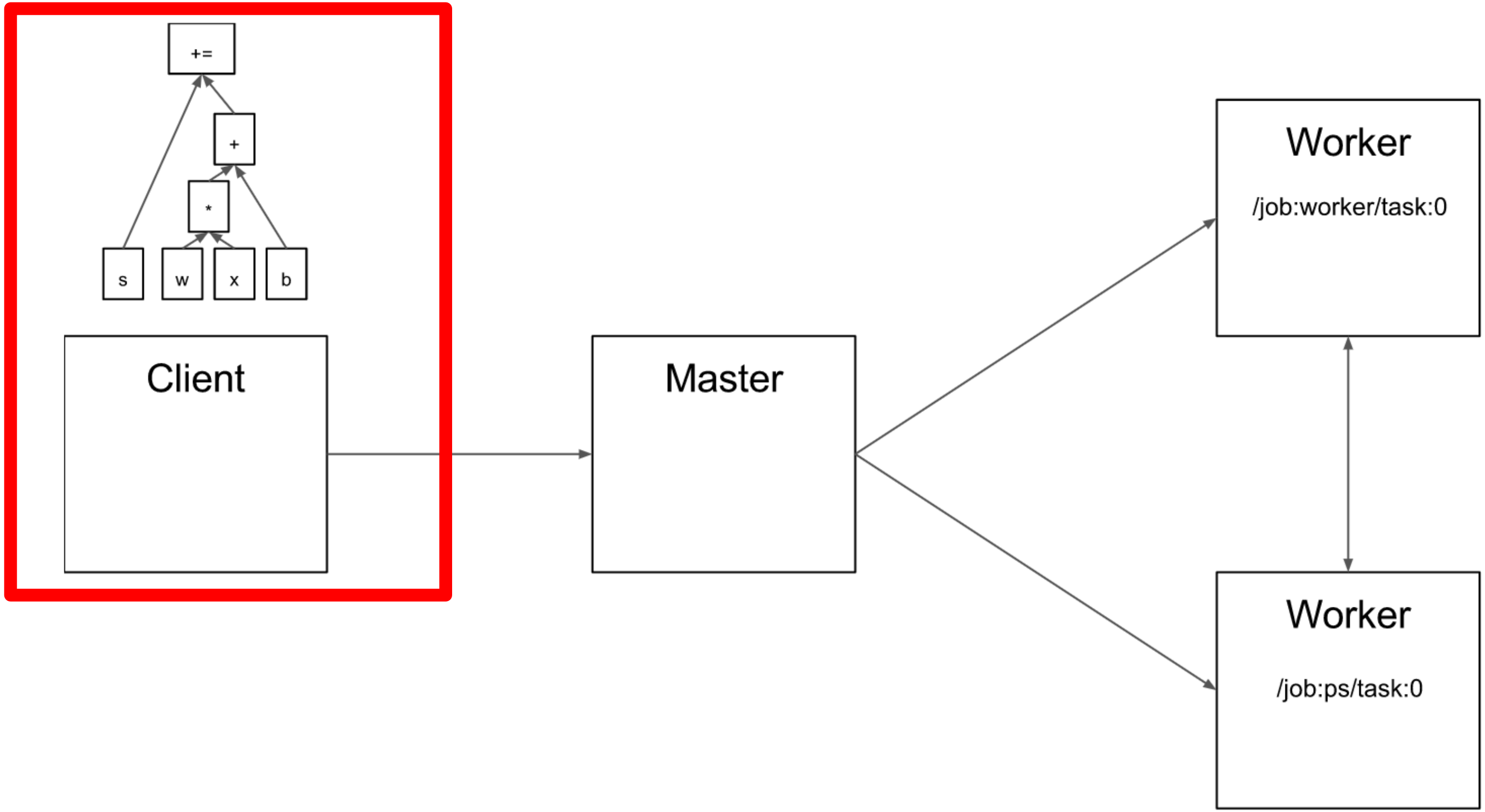
Key components

Similar to MapReduce, Apache Hadoop, Apache Spark, …



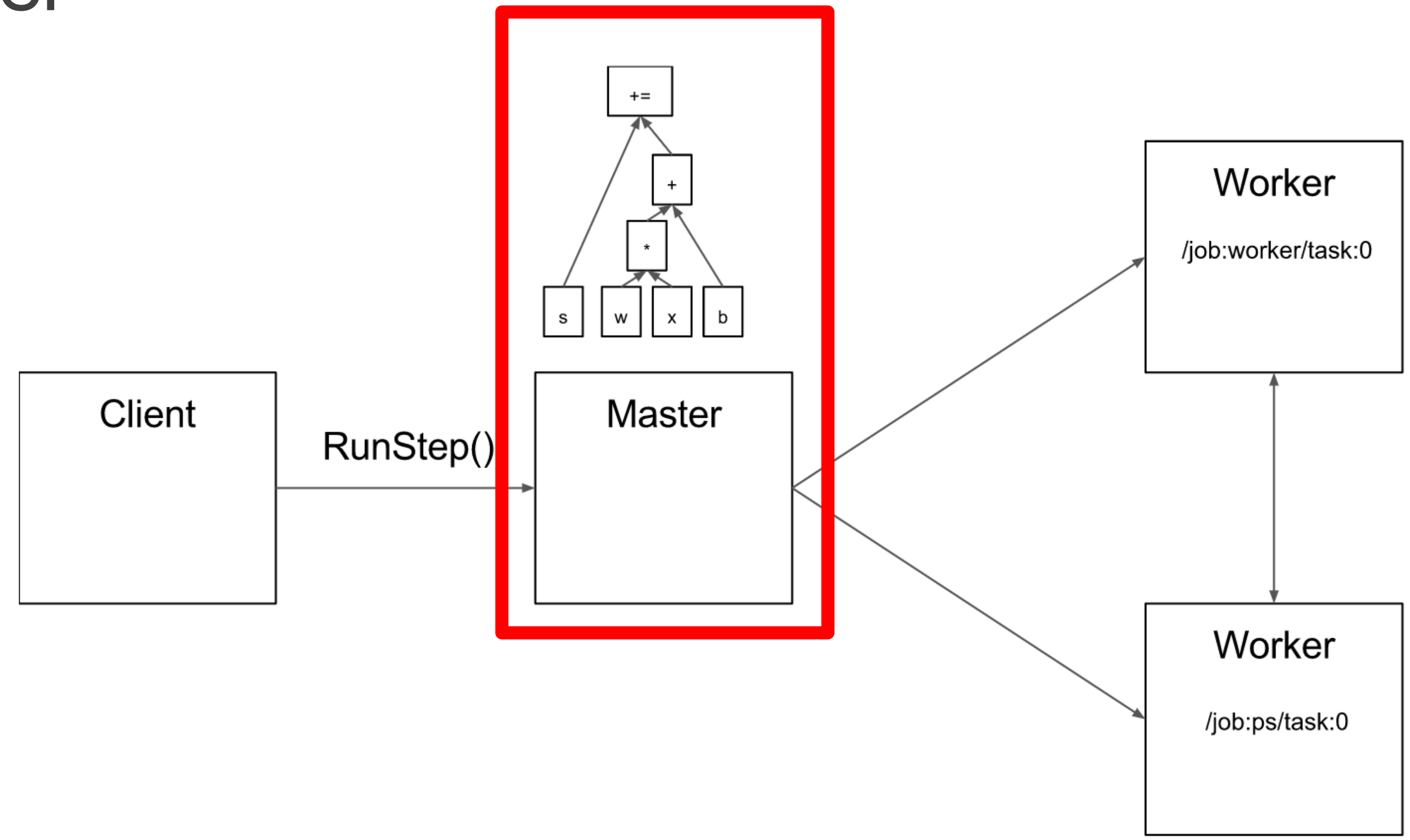
From: https://www.tensorflow.org/extend/architecture

Client



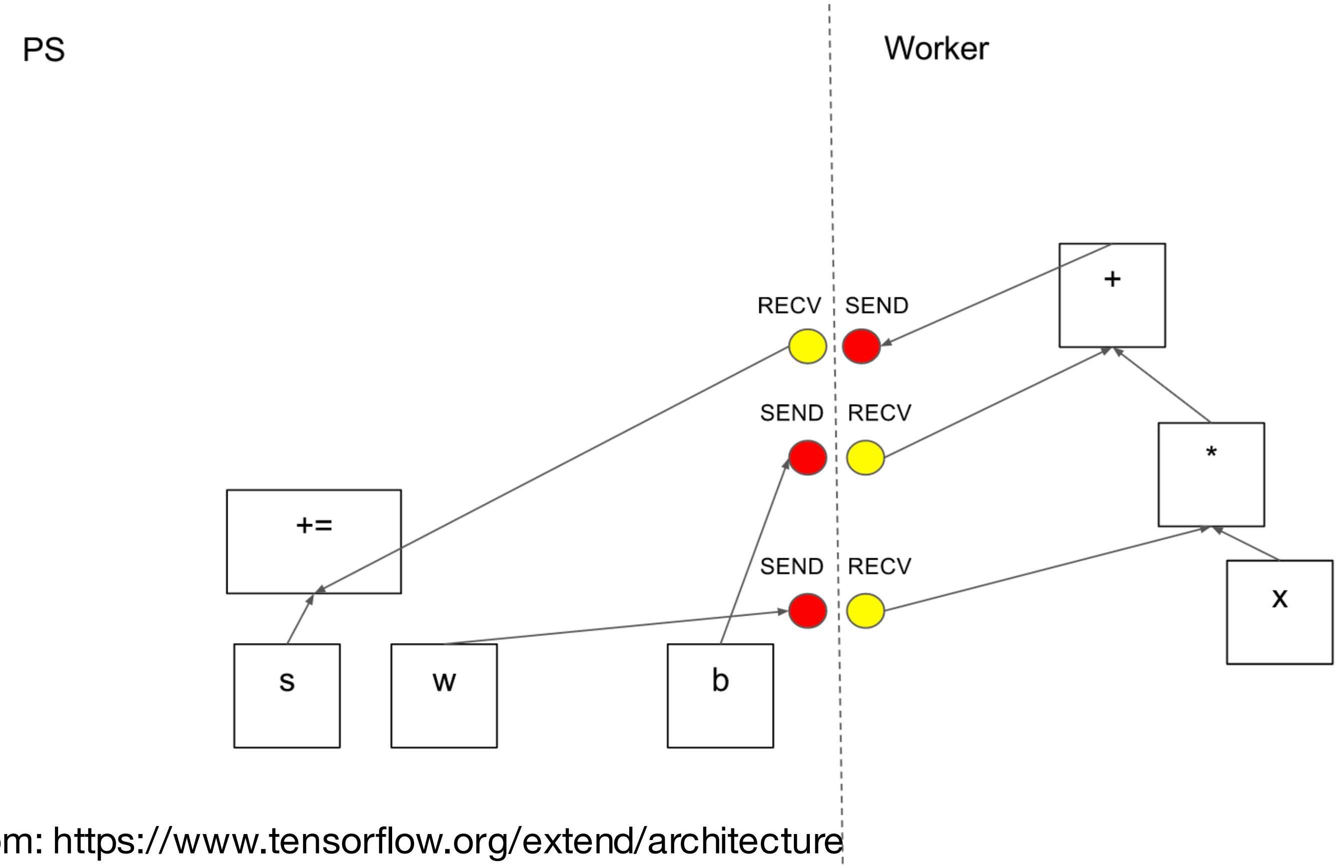
From: https://www.tensorflow.org/extend/architecture

Master



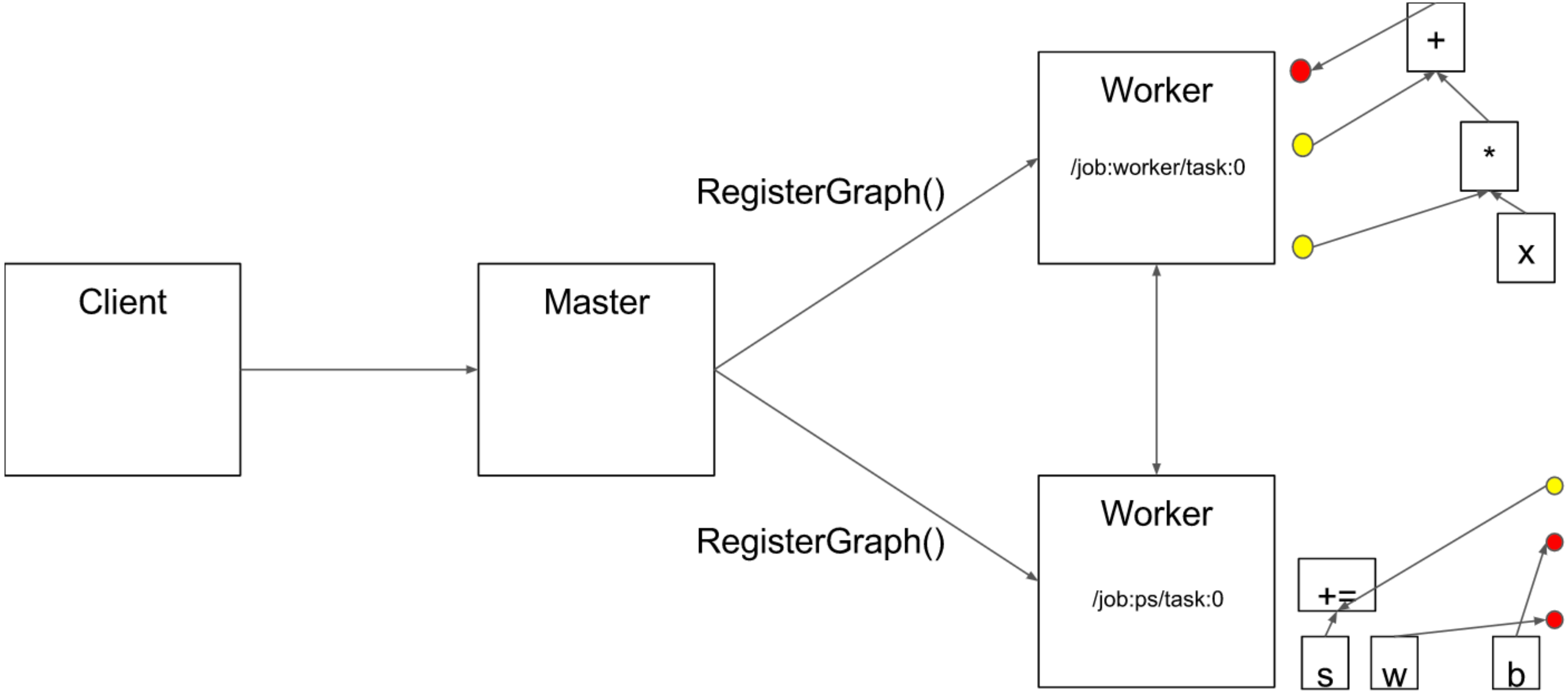
From: https://www.tensorflow.org/extend/architecture

Computation graph partition



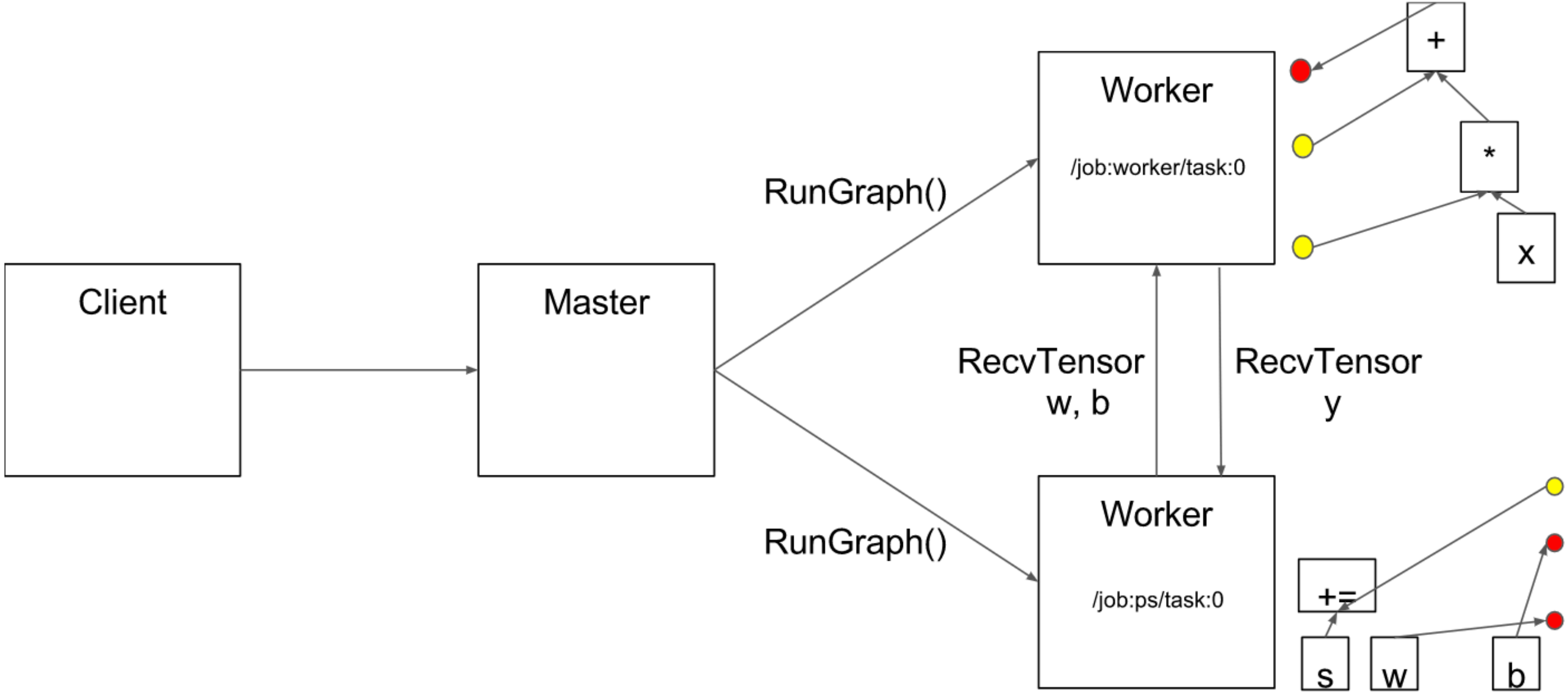
From: https://www.tensorflow.org/extend/architecture

Computation graph partition



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

Execution



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

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

Discussion

Eager vs. deferred (lazy) execution

Transparent vs. user-level fault tolerance

Easy of use

Discussion

	OpenMP/Cilk	MPI	MapReduce / Spark
Environment, Assumptions	Single node, multiple core, shared memory	Supercomputers Sophisticate programmers High performance Hard to scale hardware	Commodity clusters Java programmers Programmer productivity Easier, faster to scale up cluster
Computation Model	Fine-grained task parallelism	Message passing	Data flow / BSP
Strengths	Simplifies parallel programming on multi-cores	Can write very fast asynchronous code	Fault tolerance
Weaknesses	Still pretty complex, need to be careful about race conditions	Fault tolerance Easy to end up with non-deterministic code (if not using barriers)	Not as high performance as MPI