Introduction to TensorFlow

TensorFlow:
Large-Scale Machine Learning on Heterogeneous Distributed Systems

(Preliminary White Paper, November 9, 2015)

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng
Google Research*

Abstract

TensorFlow [1] is an interface for expressing machine learning algorithms, and an implementation for executing such algorithms. A computation expressed using TensorFlow can be executed with little or no change on a wide variety of heterogeneous systems, ranging from mobile devices such as phones

sequence prediction [47], move selection for Go [34], pedestrian detection [2], reinforcement learning [38], and other areas [17, 5]. In addition, often in close collaboration with the Google Brain team, more than 50 teams at Google and other Alphabet companies have deployed deep neural networks using DistBelief in a wide variety

http://tensorflow.org/whitepaper2015.pdf

TensorFlow: A system for large-scale machine learning

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean,
Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur,
Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker,
Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng

Google Brain

OSDI 2016

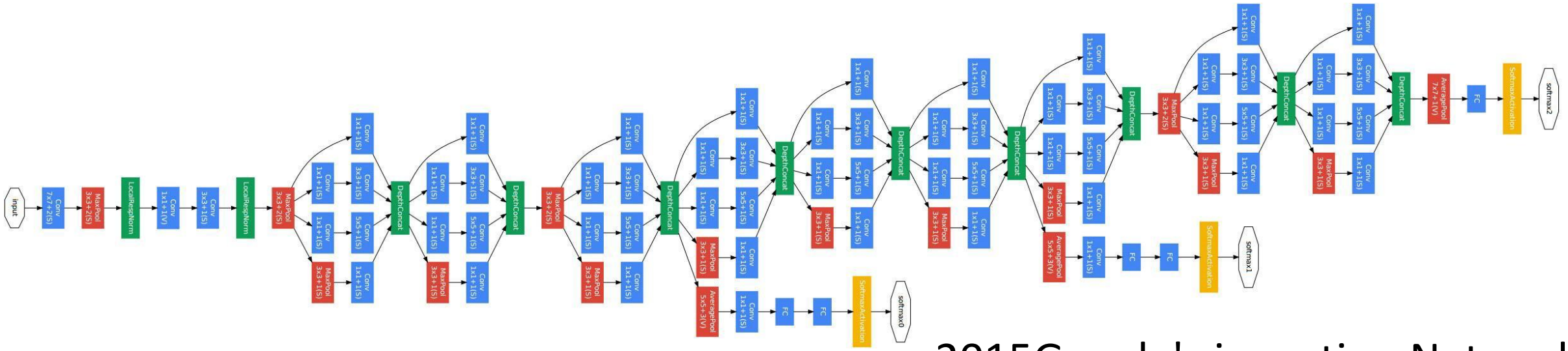
Outline

- What is TensorFlow?
- Why did we create TensorFlow?
- How does TensorFlow work?
- Parallelisms: Data and Model
- Fault Tolerance
- Wrapping up

- Architecture in comparison with Mapreduce

- Fast, flexible, and scalable open-source machine learning library
- One system for research and production
- Runs on CPU, GPU, TPU, and Mobile
- Apache 2.0 license

Machine learning gets complex quickly



2015Google's inception Network

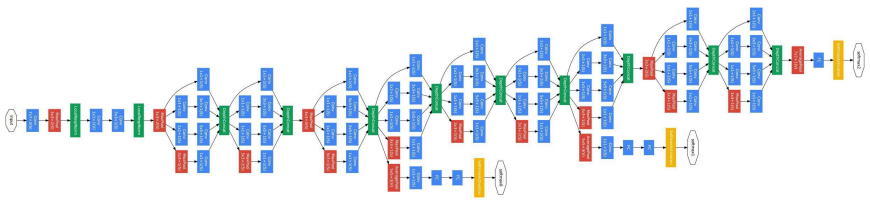
Modeling complexity

Machine learning gets complex quickly

Distributed System

Heterogenous System

TensorFlow Handles Complexity



Modeling complexity

Distributed System

Heterogenous System

Under the Hood

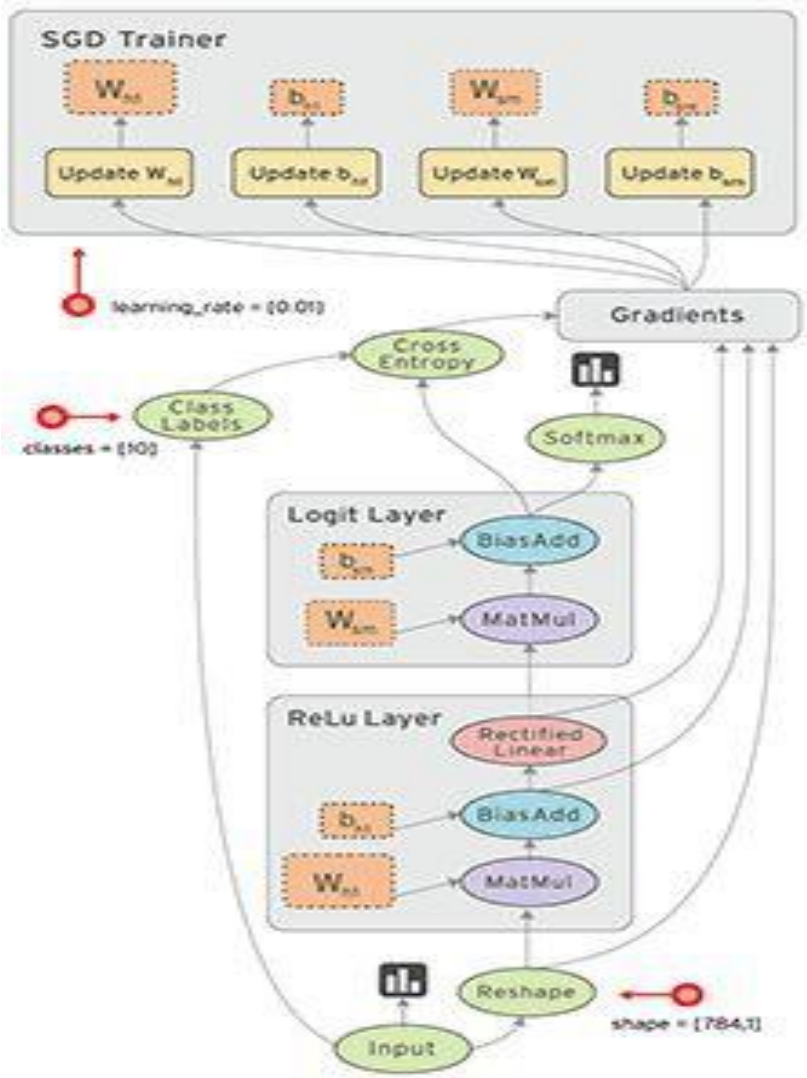
A multidimensional array.

A graph of operations.

The TensorFlow Graph

Computation is defined as a graph

- Graph is defined in high-level language (Python)
- Graph is compiled and optimized
- Graph is executed (in parts or fully) on available low level devices (CPU, GPU, TPU)
- Nodes represent computations and state
- Data (tensors) flow along edges

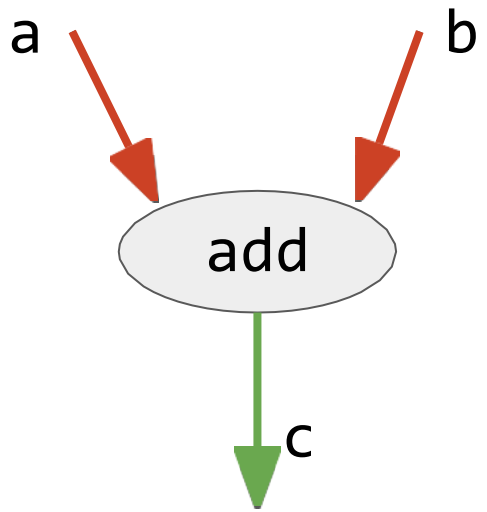


Build a graph; then run it.

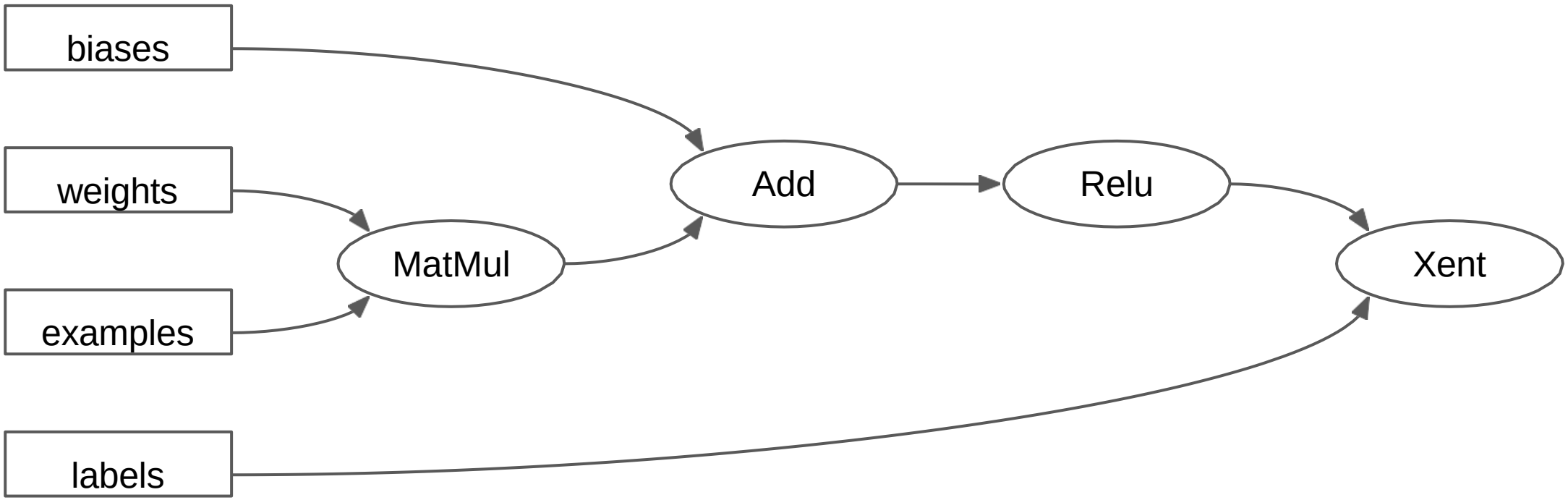
```
...
c = tf.add(a, b)


...

session = tf.Session()
value_of_c = session.run(c, {a=1, b=2})
```

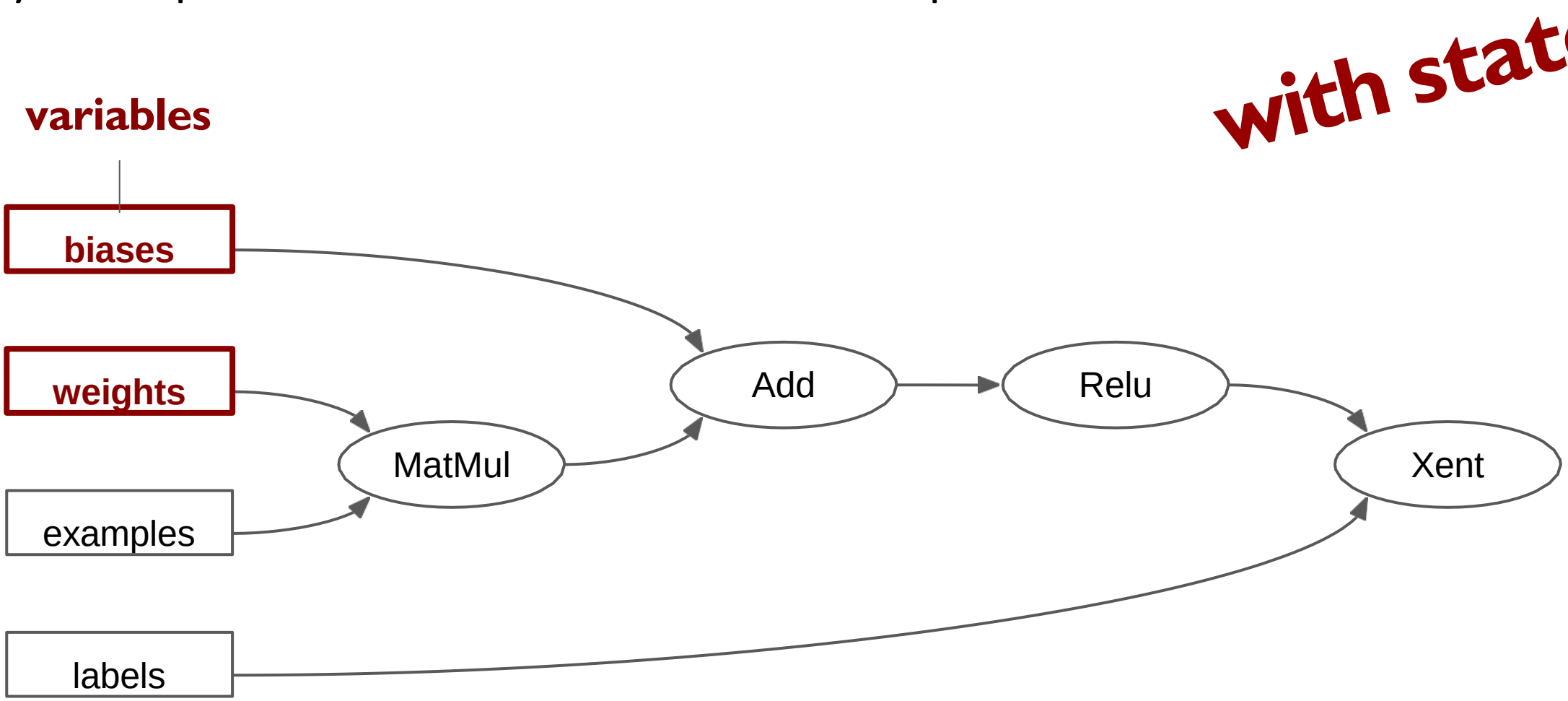


Any Computation is a TensorFlow Graph



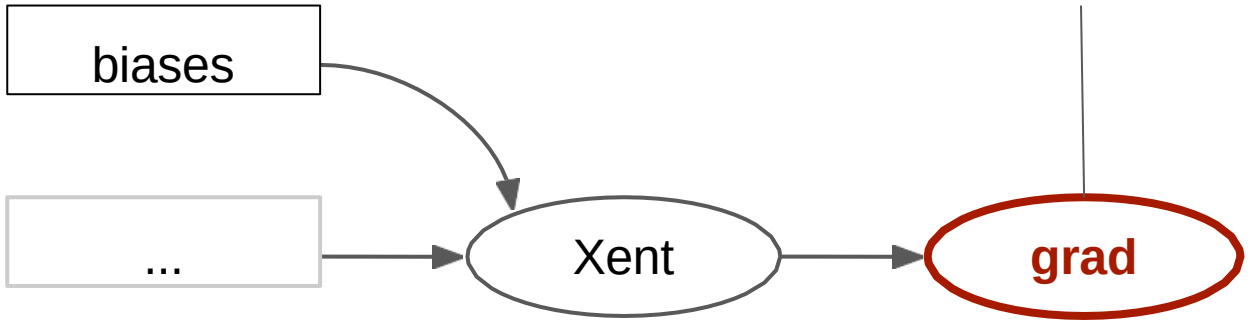
A single neural network layer; a primitive linear classifier

Any Computation is a TensorFlow Graph



Automatic Differentiation

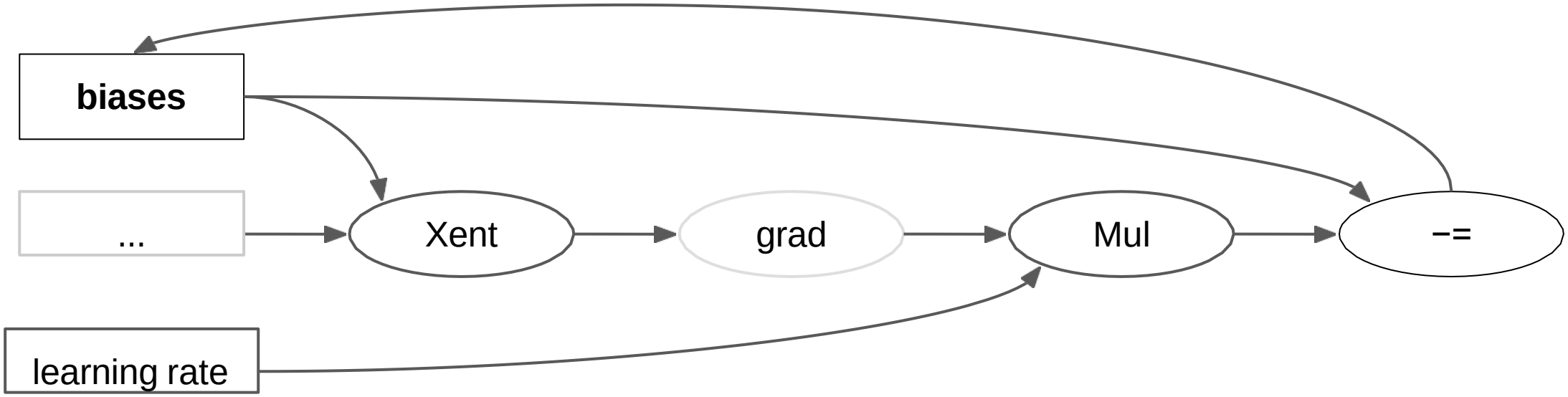
Automatically add ops which compute gradients for variables



Any Computation is a TensorFlow Graph

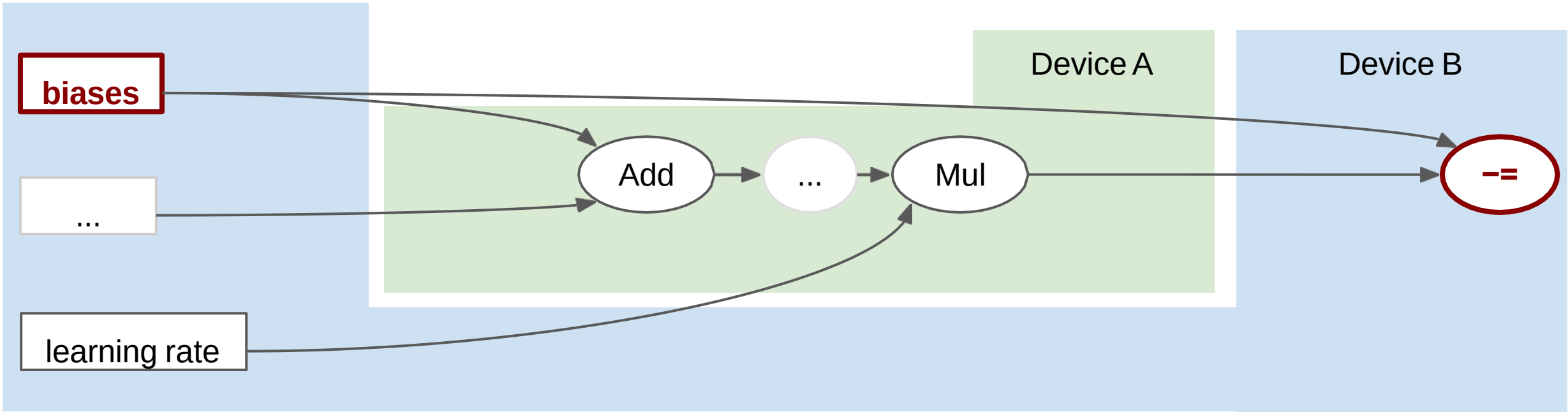
Simple gradient descent:

with state



Any Computation is a TensorFlow Graph

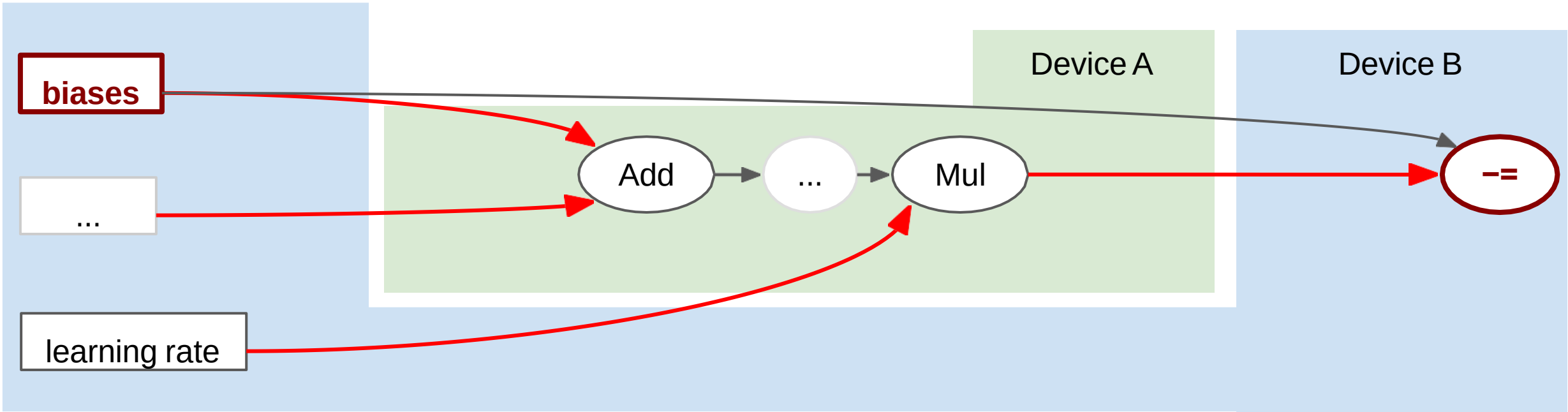
distributed



Devices: Processes, Machines, CPUs, GPUs, TPUs, etc

Send and Receive Nodes

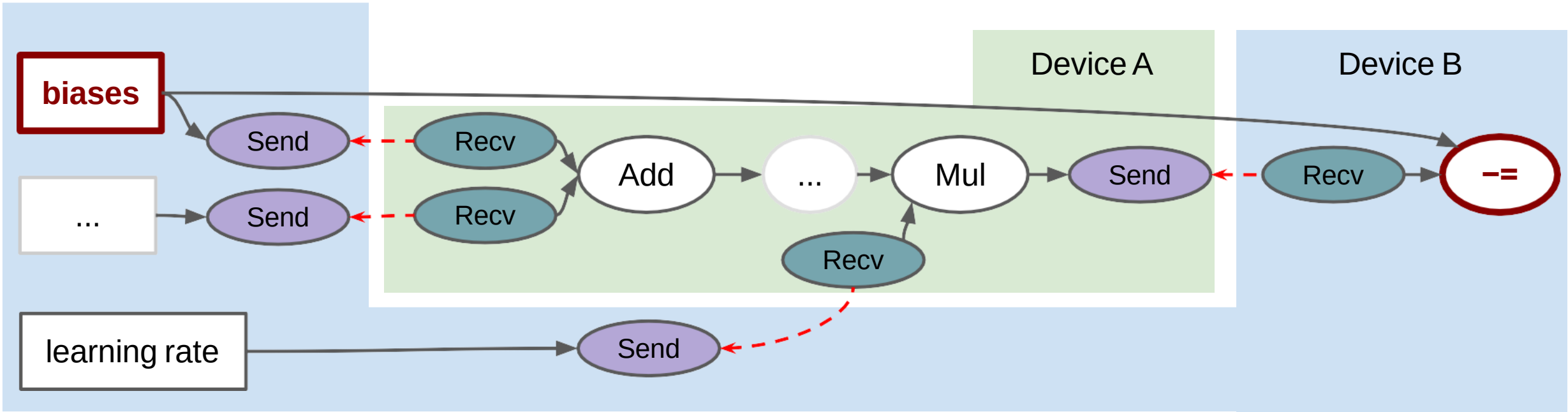
distributed



Devices: Processes, Machines, CPUs, GPUs, TPUs, etc

Send and Receive Nodes

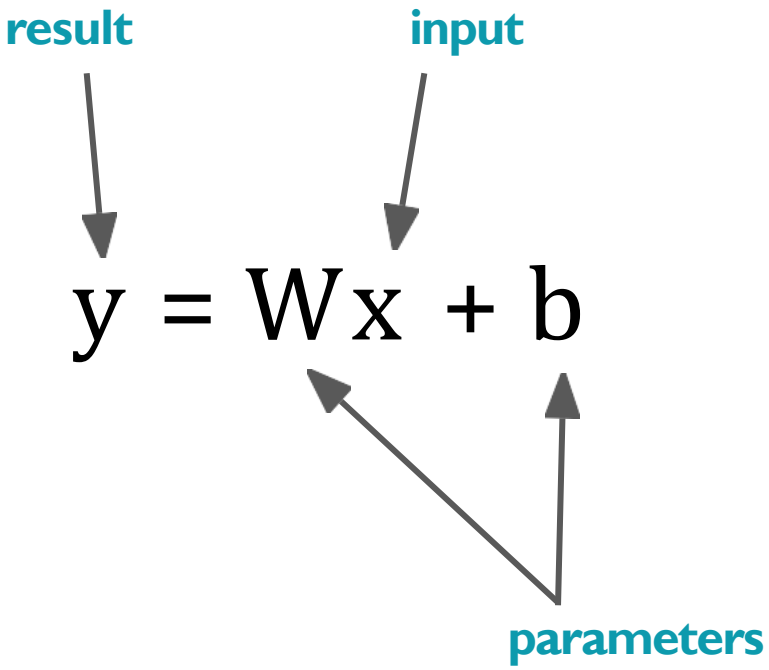
distributed

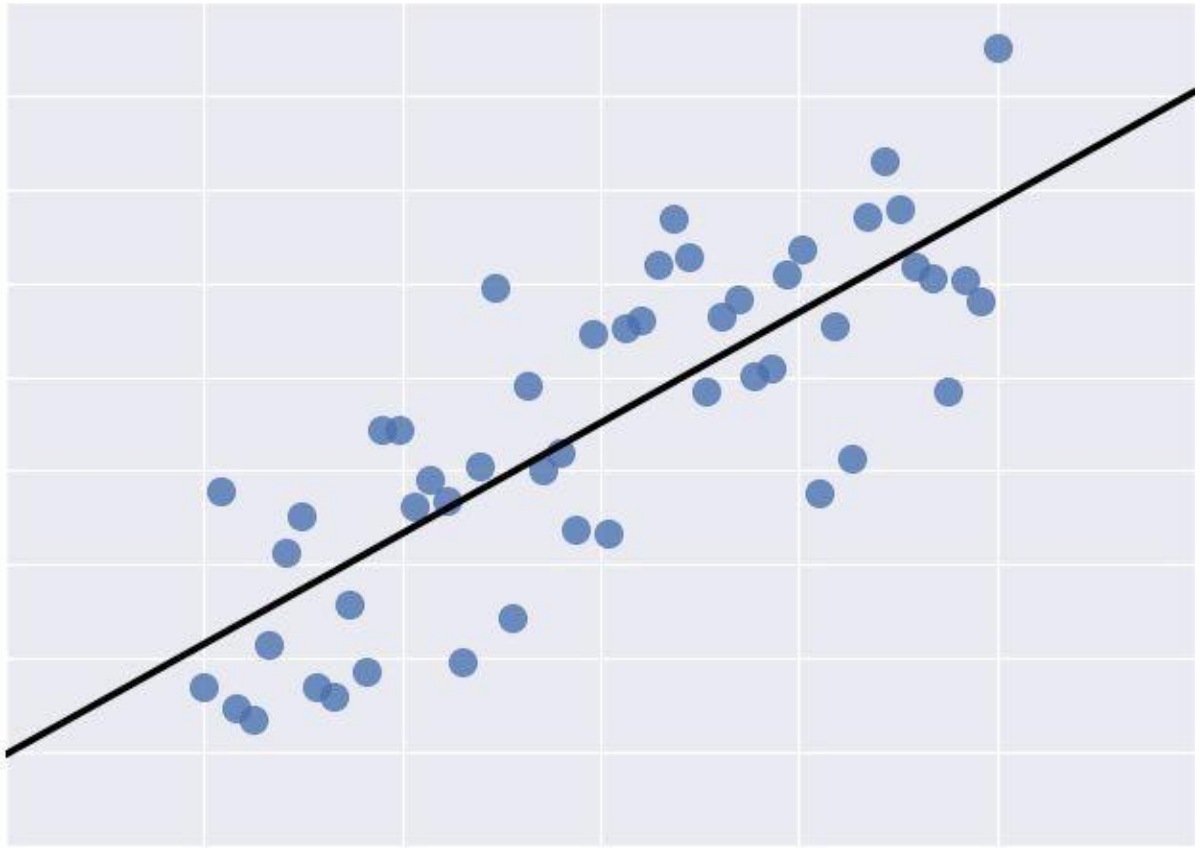


Devices: Processes, Machines, CPUs, GPUs, TPUs, etc

From perspective of Linear Regression

Linear Regression





What are we trying to do?

Mystery equation: `y = 0.1 * x + 0.3 + noise`

Model: `y = W * x + b`

Objective: Given enough `(x, y)` value samples, figure out the value of `W` and `b`.

y = Wx + b in TensorFlow

```
import tensorflow as tf
```

y = Wx + b in TensorFlow

```
import tensorflow as tf

x = tf.placeholder(shape=[None],
                   dtype=tf.float32, name=”x”)
```

y = Wx + b in TensorFlow

```
import tensorflow as tf

x = tf.placeholder(shape=[None],
                   dtype=tf.float32, name=”x”)

W = tf.get_variable(shape=[], name=”W”)
```

y = Wx + b in TensorFlow

```
import tensorflow as tf

x = tf.placeholder(shape=[None],
                   dtype=tf.float32, name=”x”)

W = tf.get_variable(shape=[], name=”W”)

b = tf.get_variable(shape=[], name=”b”)
```

y = Wx + b in TensorFlow

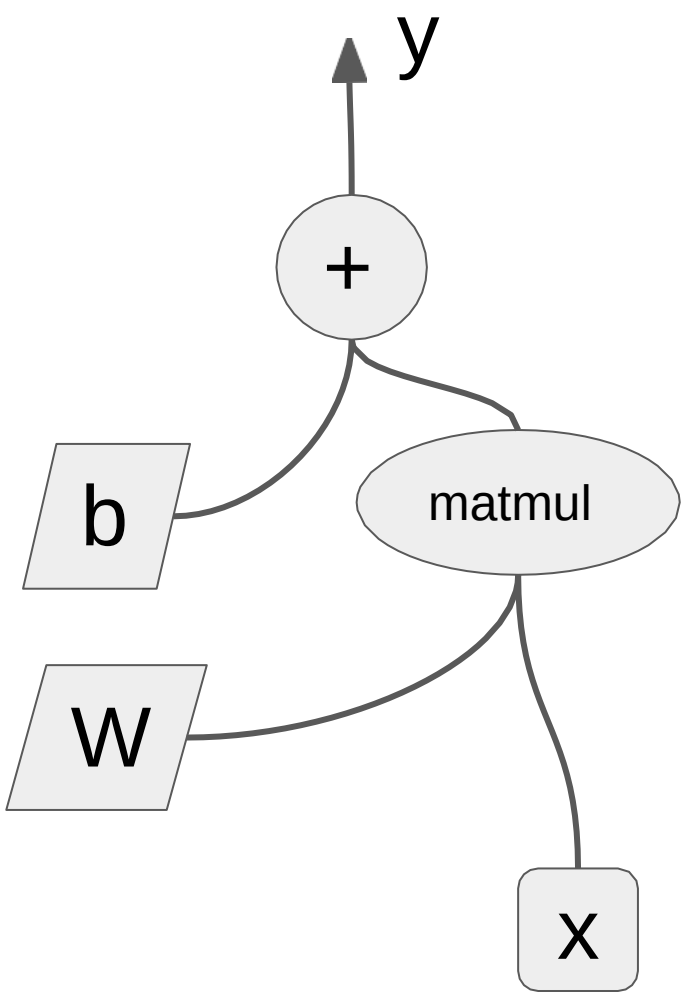
```
import tensorflow as tf

x = tf.placeholder(shape=[None],
                   dtype=tf.float32, name="x")

W = tf.get_variable(shape=[], name="W")

b = tf.get_variable(shape=[], name="b")

y = W * x + b
```



Variables Must be Initialized

Collects all variable initializers

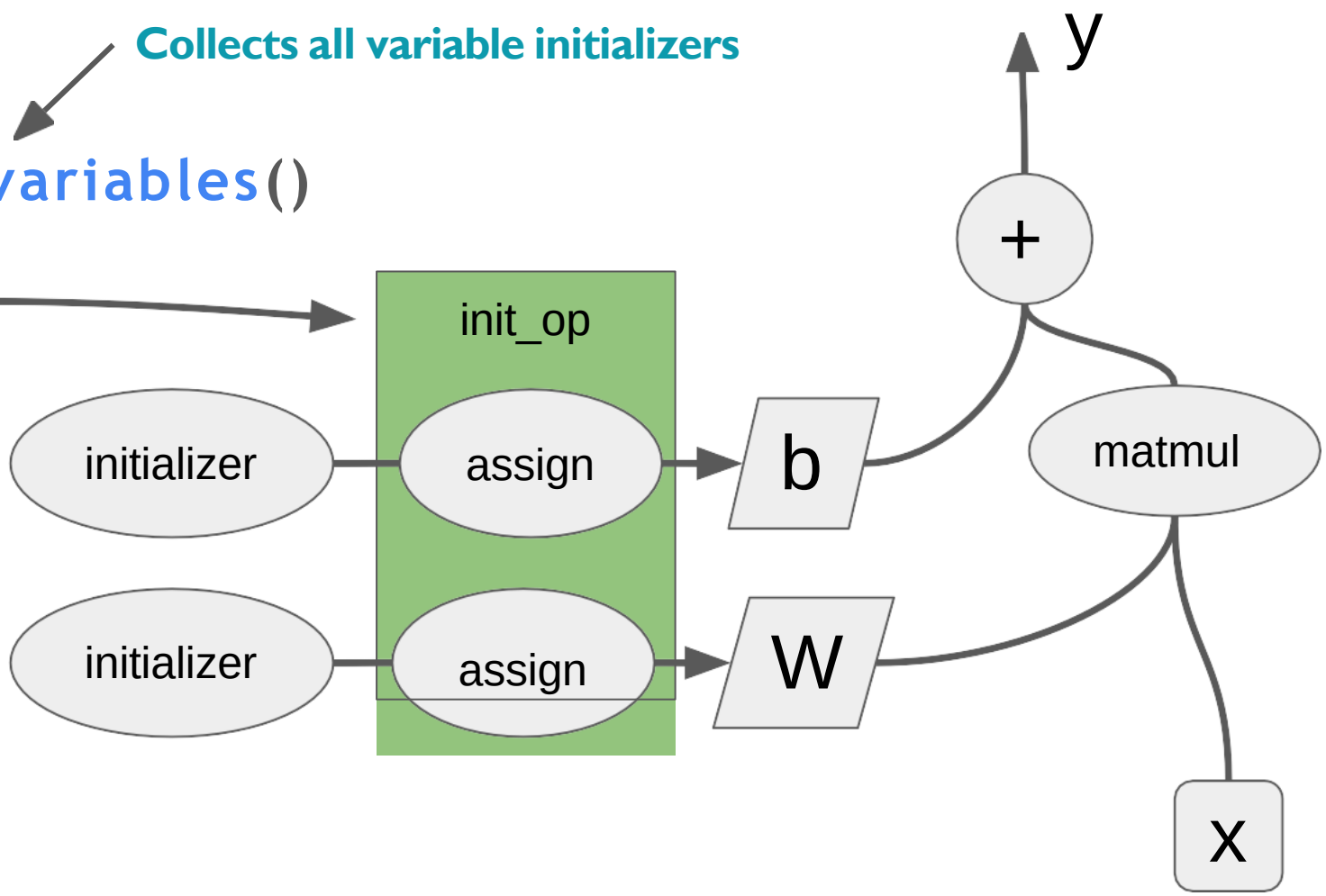
```
init_op = tf.initialize_all_variables()
```

Makes an execution environment

```
sess = tf.Session()

sess.run(init_op)
```

Actually initialize the variables

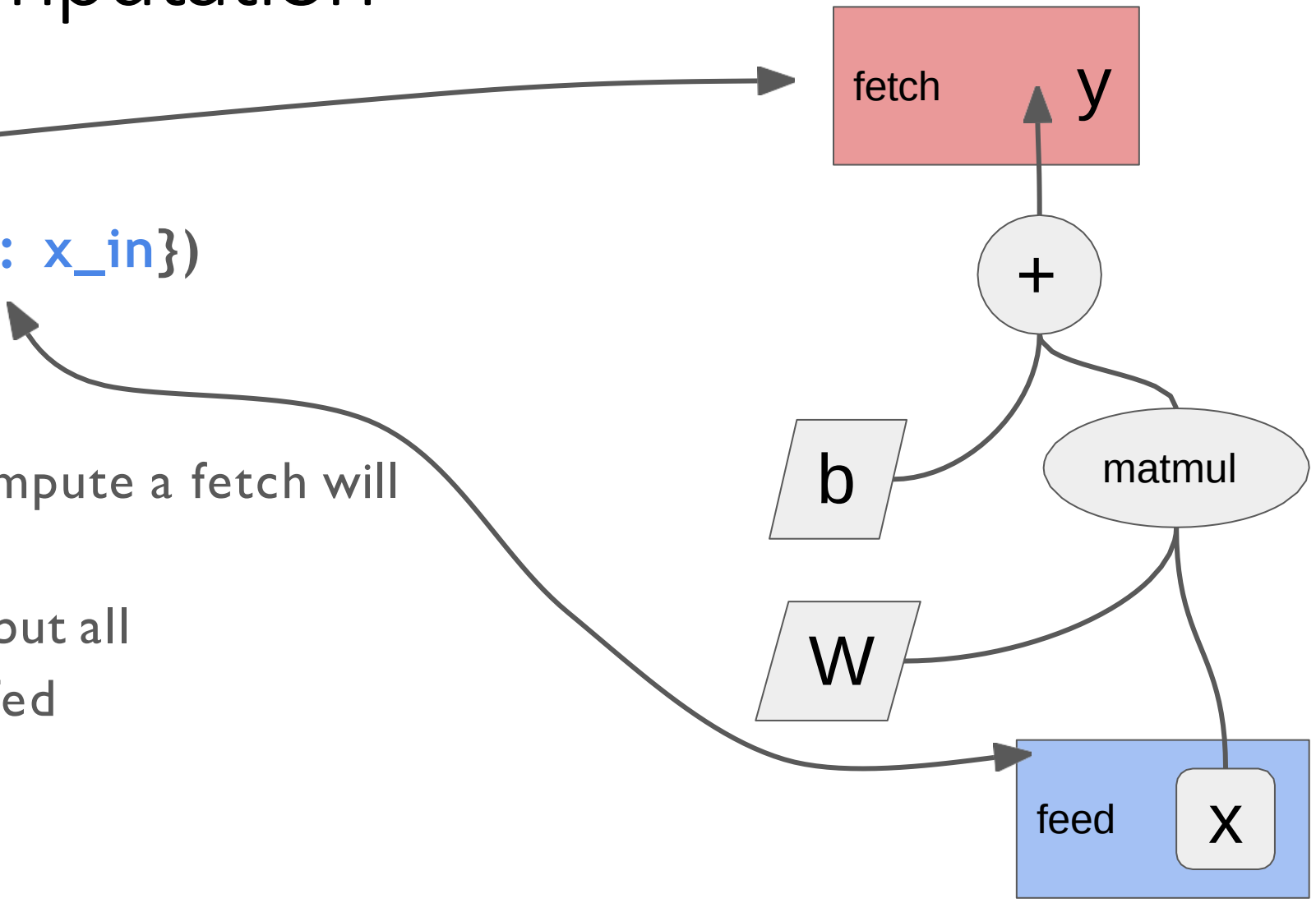


Running the Computation

```
x_in = 3

sess.run(y, feed_dict={x: x_in})
```

- Only what's used to compute a fetch will be evaluated
- All Tensors can be fed, but all placeholders must be fed



Putting it all together

```
import tensorflow as tf
x = tf.placeholder(shape=[None],
                   dtype=tf.float32,
                   name='x')
W = tf.get_variable(shape=[], name='W')
b = tf.get_variable(shape=[], name='b')
y = W * x + b

with tf.Session() as sess:

  sess.run(tf.initialize_all_variables())

  print(sess.run(y, feed_dict={x: x_in}))
```

Build the graph

Prepare execution environment

Initialize variables

Run the computation (usually often)

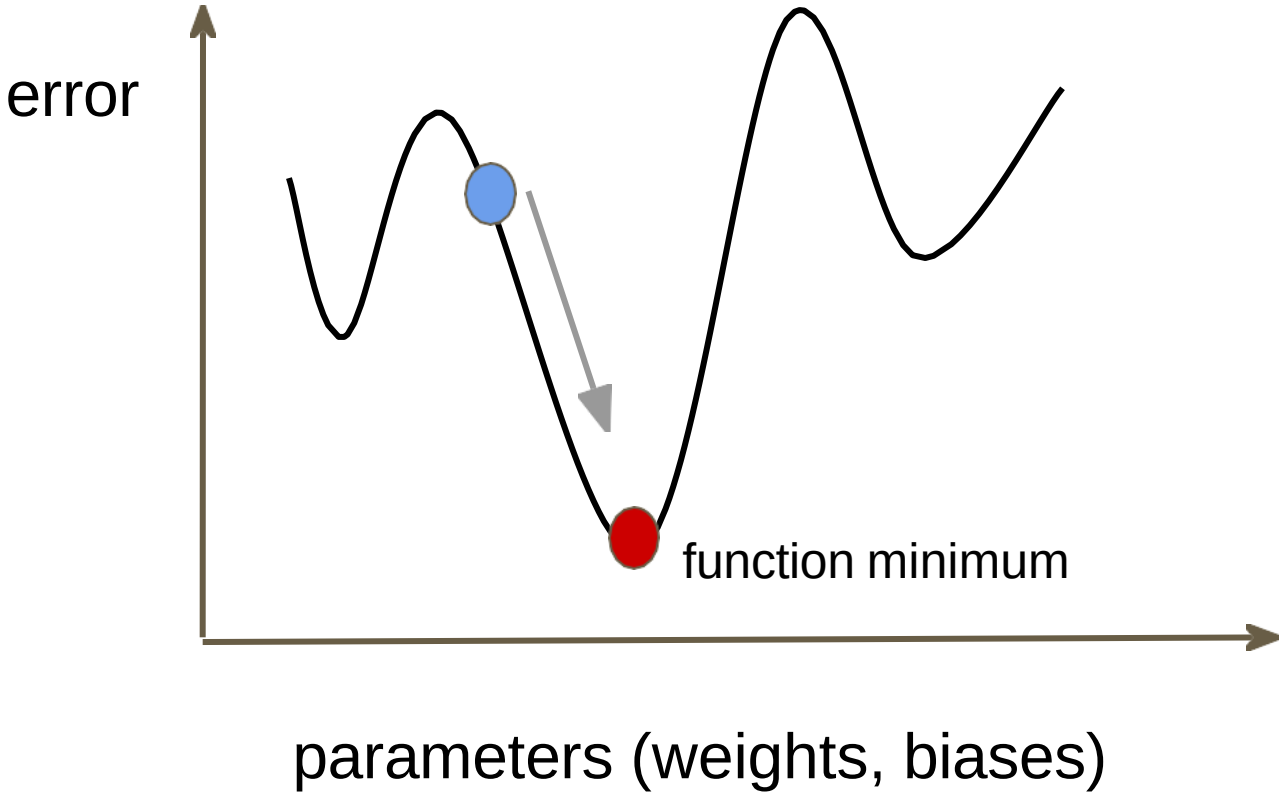
Define a Loss

Given **x, y** compute a loss, for instance:

$$L = (y - y_{label})^2$$

```
# create an operation that calculates loss.
loss = tf.reduce_mean(tf.square(y - y_data))
```

Minimize loss: optimizers




```
tf.train.AdadeltaOptimizer
tf.train.AdagradOptimizer
tf.train.AdagradDAOptimizer
tf.train.AdamOptimizer
...
```

Train

Feed (x, y_{label}) pairs and adjust W and b to decrease the loss.

$$W \leftarrow W - 1\,(\,dL/dW\,)$$

$$b \leftarrow b - 1\,(\,dL/db\,)$$

TensorFlow computes gradients automatically

```
# Create an optimizer
optimizer = tf.train.GradientDescentOptimizer(0.5)

# Create an operation that minimizes loss.
train = optimizer.minimize(loss)
```

Learning rate

Putting it all together

```
loss = tf.reduce_mean(tf.square(y - y_label))

optimizer = tf.train.GradientDescentOptimizer(0.5)

train = optimizer.minimize(loss)

with tf.Session() as sess:

  sess.run(tf.initialize_all_variables())

  for i in range(1000):

    sess.run(train, feed_dict={x: x_in[i],

                               y_label: y_in[i]})
```

Define a loss

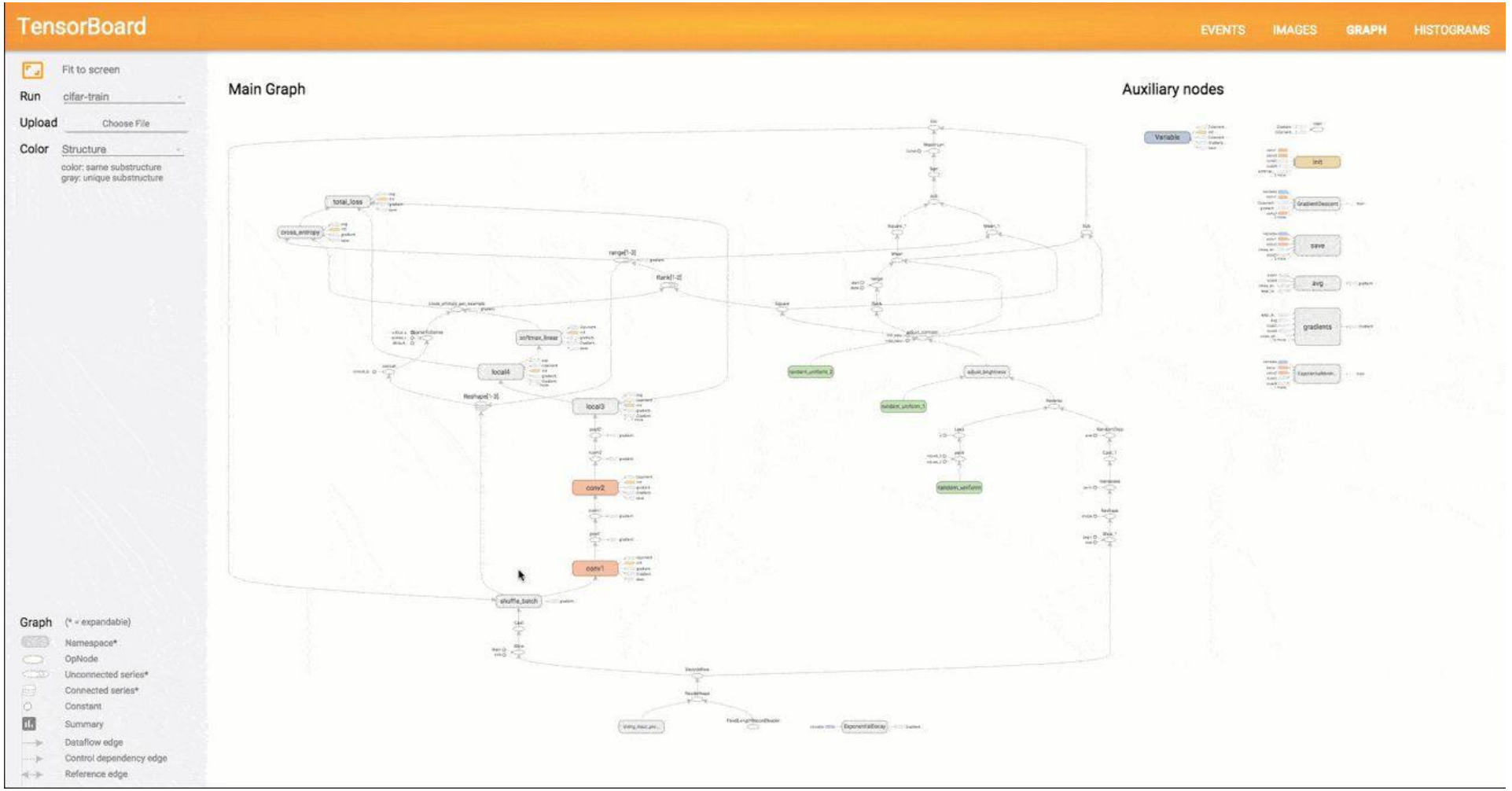
Create an optimizer

Op to minimize the loss

Initialize variables

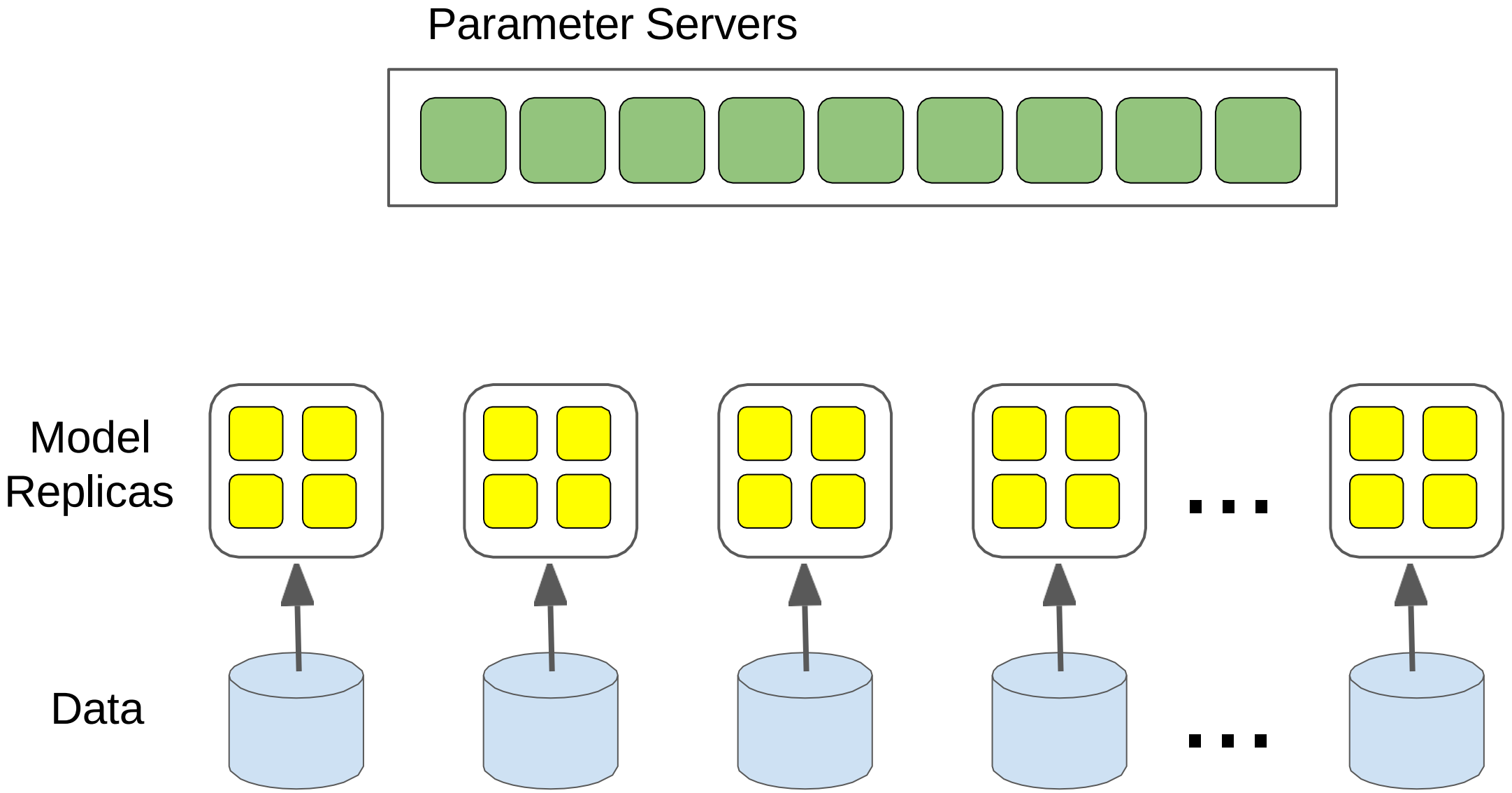
Iteratively run the training op

TensorBoard

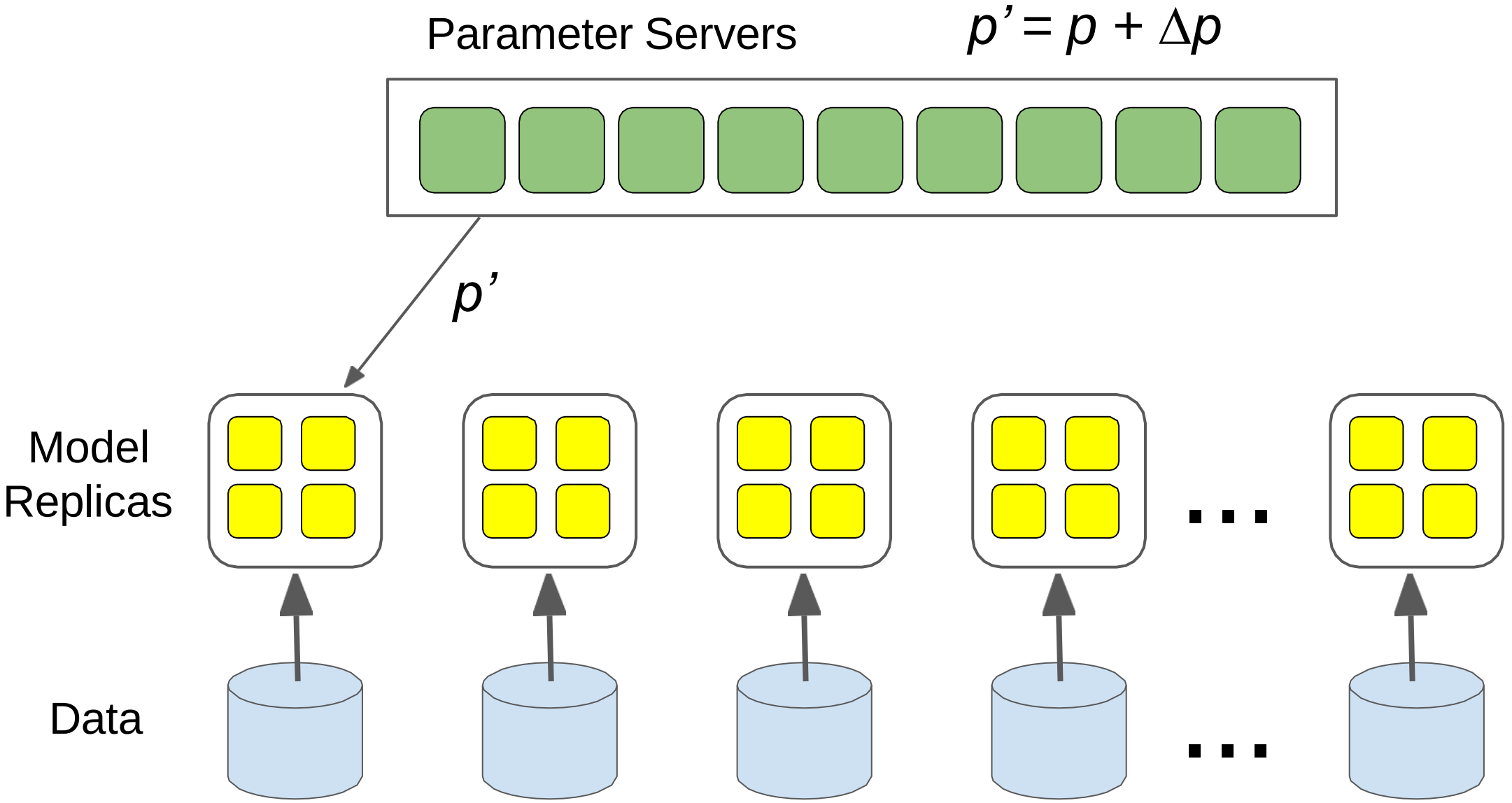


Parallelism

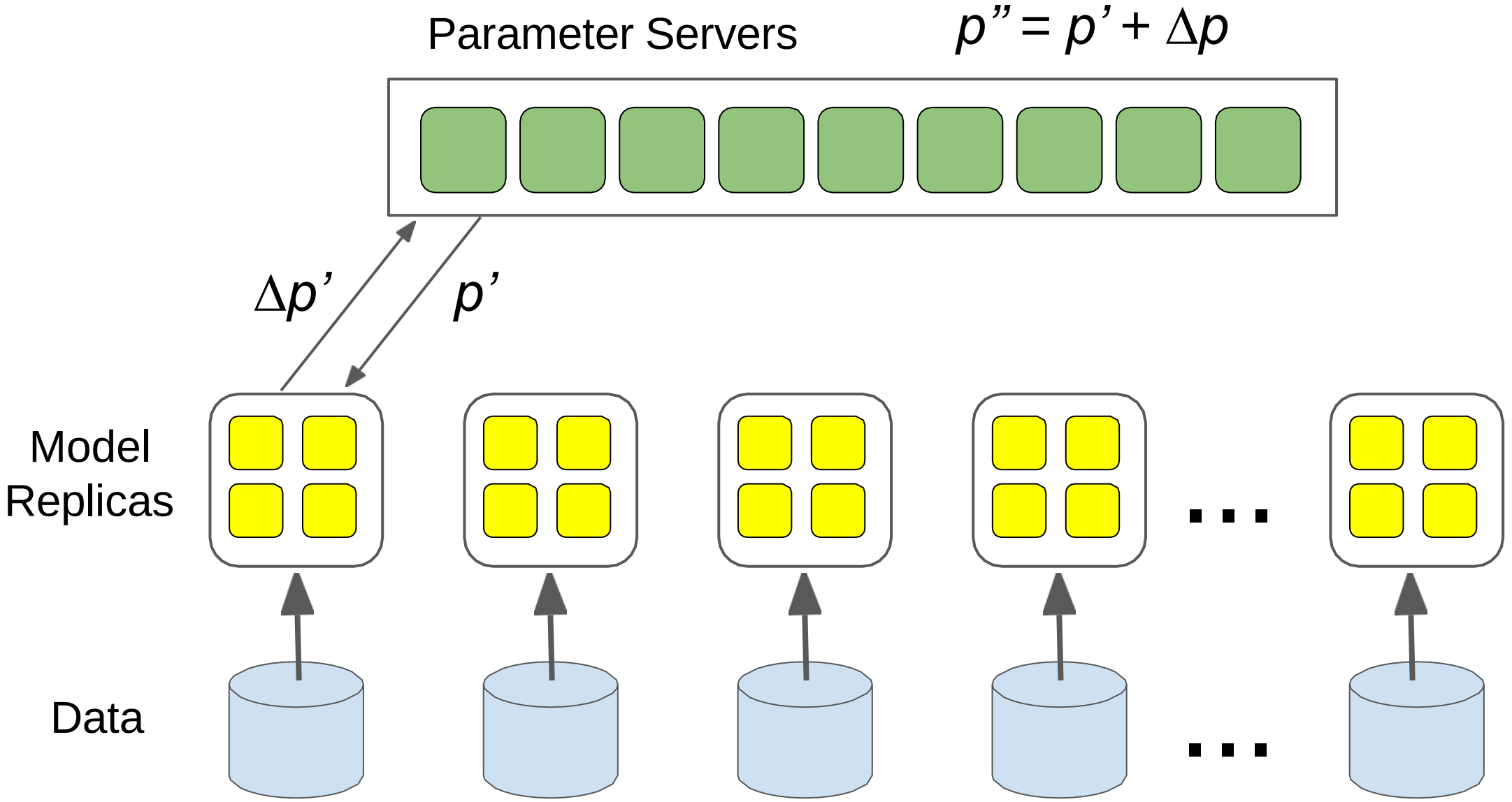
Data Parallelism



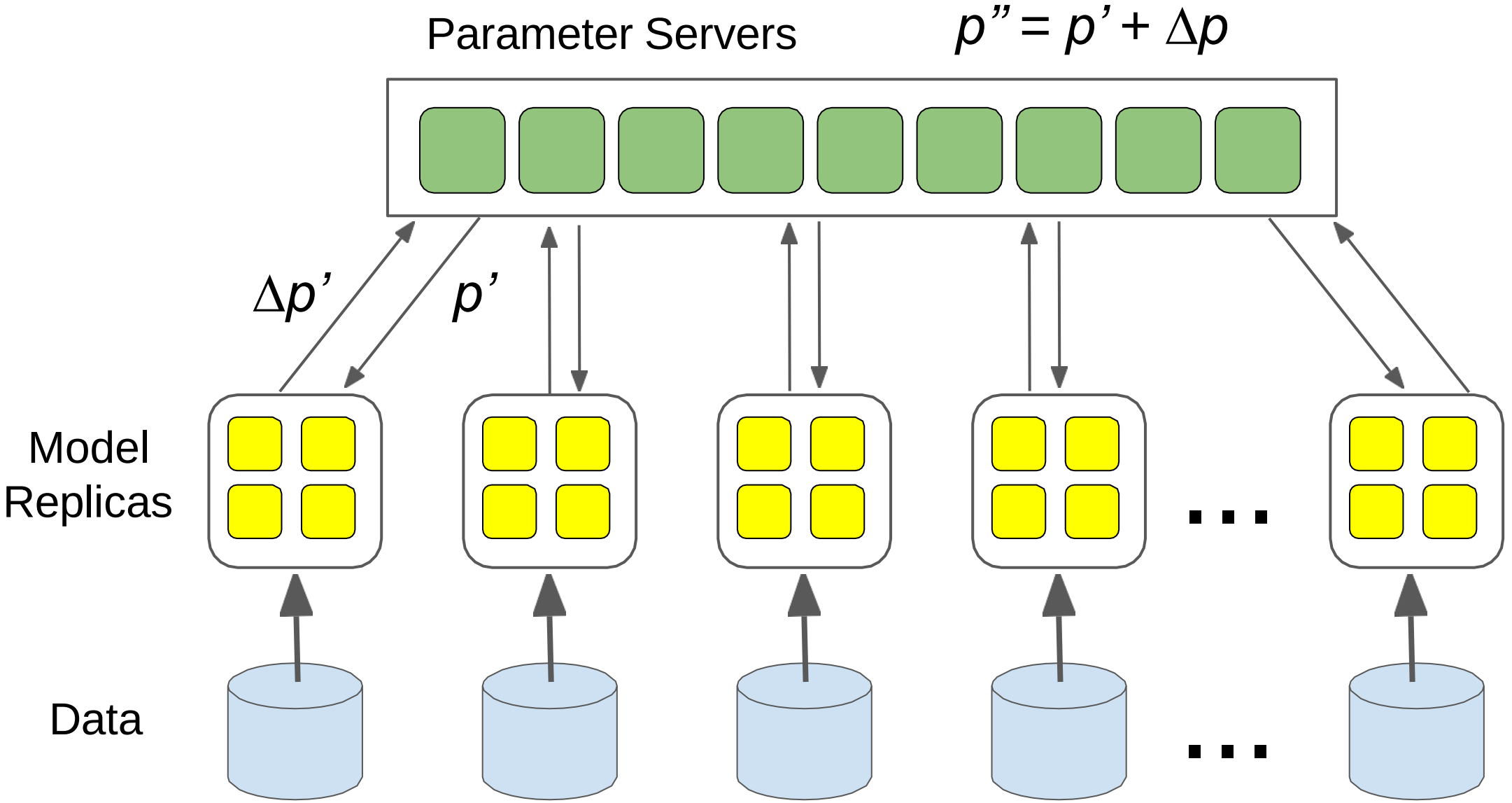
Data Parallelism



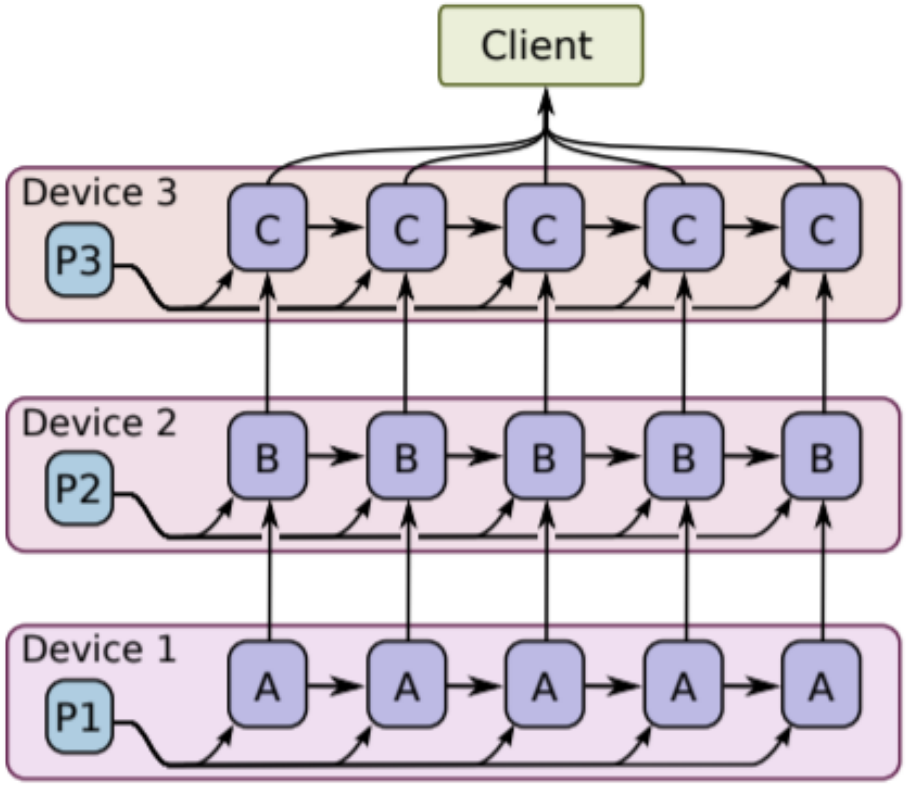
Data Parallelism



Data Parallelism



Model Parallelism

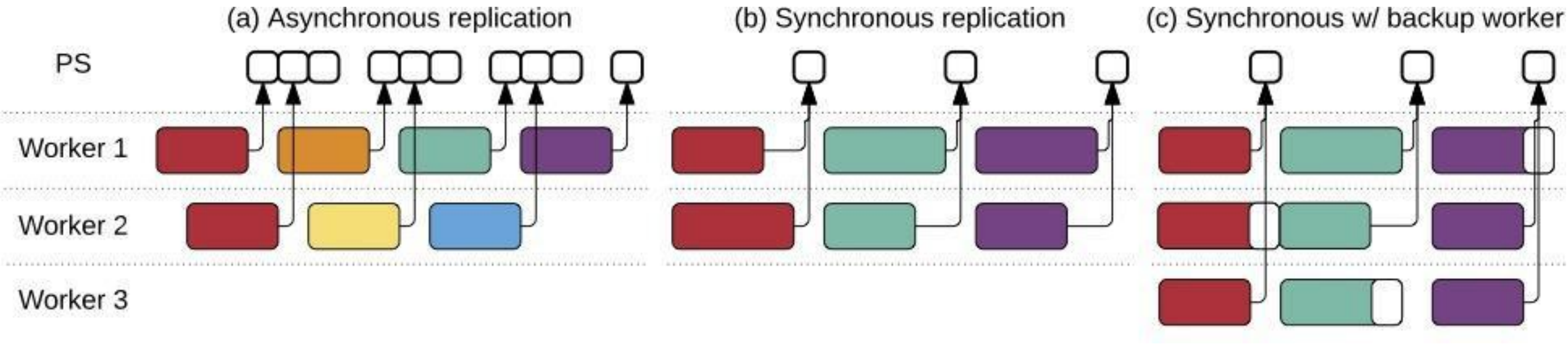


Fault Tolerance

- **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

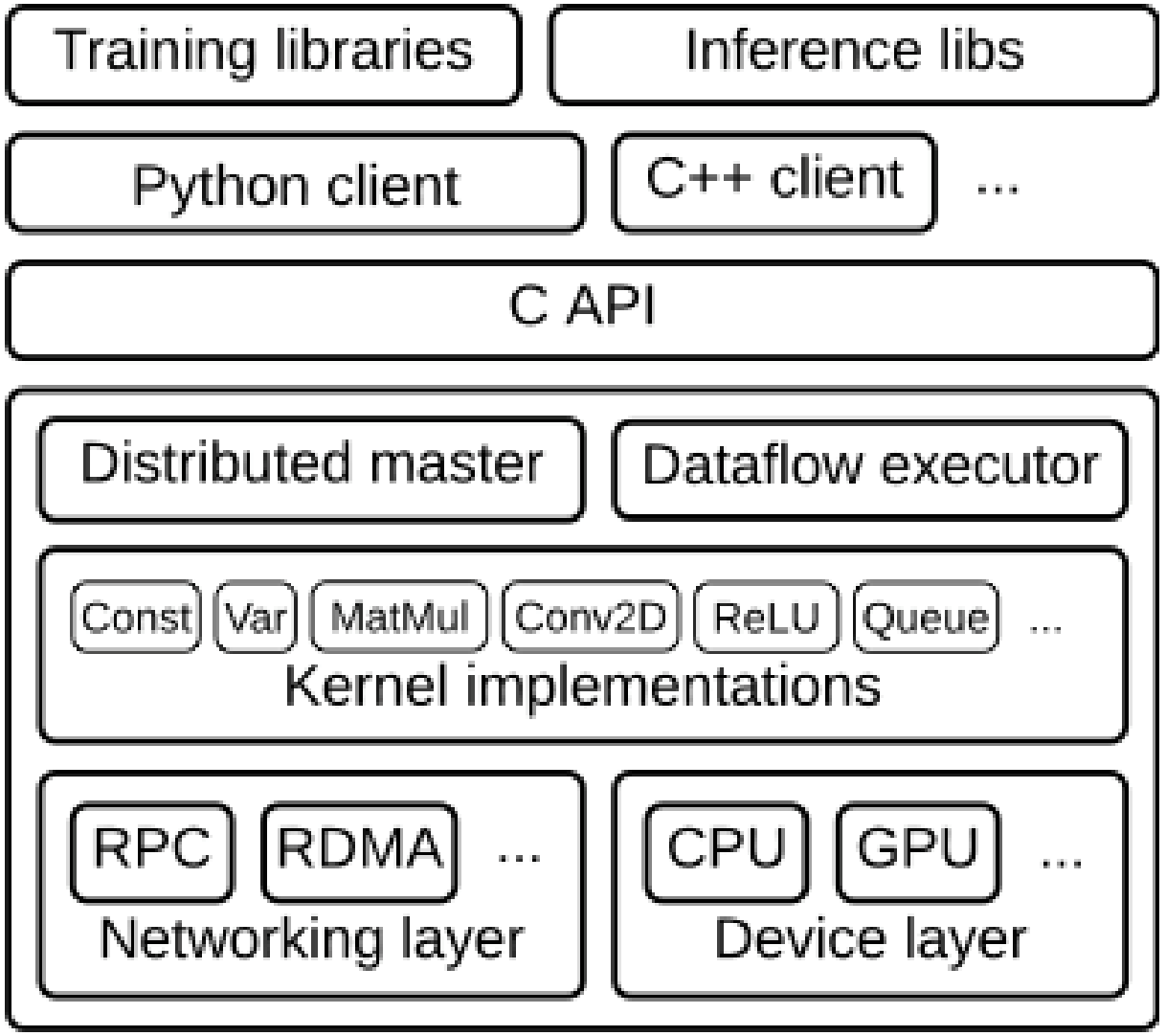
Distributed training mechanisms



Graph structure and low-level graph primitives (queues) allow us to play with synchronous vs. asynchronous update algorithms.

Architecture from perspective of MapReduce

Detailed architecture



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

Thank you!

Questions?

Reference Slides

https://learning.acm.org/binaries/content/assets/leaning-center/webinar-slides/2016/martinwicke_tensorflow_webinarslides.pdf

https://www.matroid.com/scaledml/slides/jeff.pdf