Distributed TensorFlow

Stony Brook University

CSE545, Spring 2019

Goals

- Understand TensorFlow as a data workflow system.
  - Know the key components of TensorFlow.
  - Understand the key concepts of *distributed* TensorFlow.
- Execute basic distributed tensorflow program.
- Establish a foundation to distribute deep learning models:
  - Convolutional Neural Networks
  - Recurrent Neural Network (or LSTM, GRU)

TensorFlow

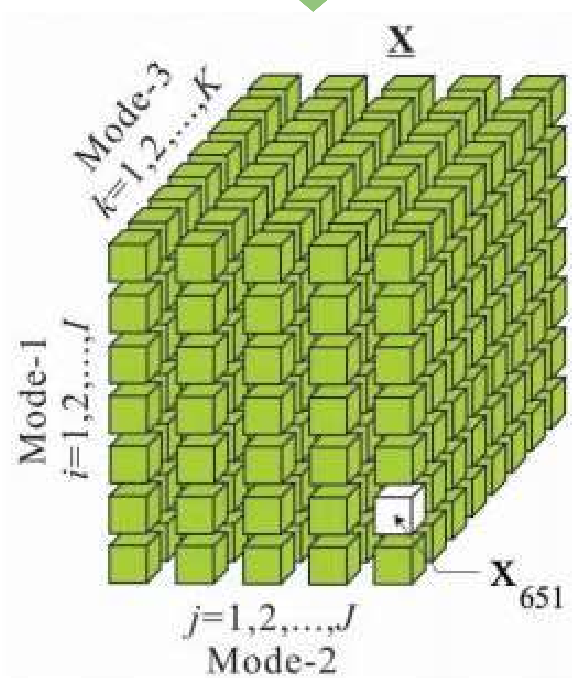
A workflow system catered to numerical computation.

One view: Like Spark, but uses *tensors* instead of *RDDs*.

TensorFlow

A workflow system catered to numerical computation.

One view: Like Spark, but uses *tensors* instead of *RDDs*.



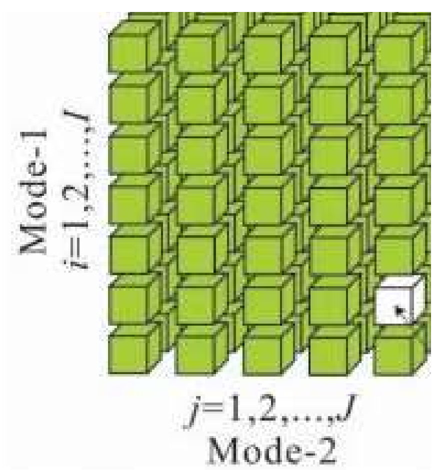
A multi-dimensional matrix

(i.stack.imgur.com)

TensorFlow

A workflow system catered to numerical computation.

One view: Like Spark, but uses *tensors* instead of *RDDs*.



(i.stack.imgur.com)

A 2-d tensor is just a matrix.

- 1-d: vector
- 0-d: a constant / scalar

Note: Linguistic ambiguity:
Dimensions of a Tensor =/=
Dimensions of a Matrix

TensorFlow

A workflow system catered to numerical computation.

One view: Like Spark, but uses *tensors* instead of *RDDs*.

Examples > 2-d :
Image definitions in terms of RGB per pixel
    Image[*row*][*column*][*rgb*]

Subject, Verb, Object representation of language:
    Counts[verb][subject][object]

TensorFlow

A workflow system catered to numerical computation.

One view: Like Spark, but uses *tensors* instead of *RDDs*.

Technically, less abstract than *RDDs* which could hold tensors as well as many other data structures (dictionaries/HashMaps, Trees, ...etc…).

Then, why TensorFlow?

TensorFlow

Efficient, high-level built-in **linear algebra** and **machine learning optimization** *operations* (i.e. transformations).

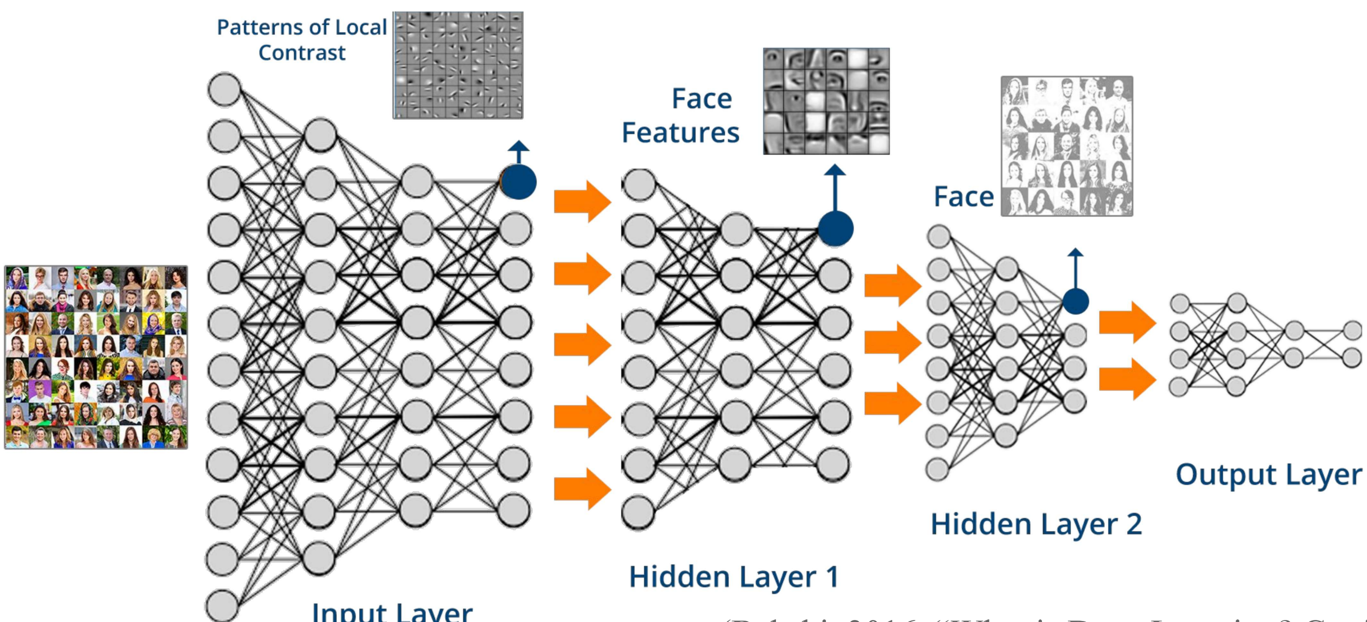
enables complex models, like deep learning

Then, why TensorFlow?

TensorFlow

Efficient, high-level built-in **linear algebra** and **machine learning** *operations*.

enables complex models, like deep learning



(Bakshi, 2016, "What is Deep Learning? Getting Started With Deep Learning")

TensorFlow

Efficient, high-level built-in **linear algebra** and **machine learning** *operations*.

```
import tensorflow as tf

b = tf.Variable(tf.zeros([100]))                    # 100-d vector, init to zeroes
W = tf.Variable(tf.random_uniform([784,100],-1,1)) # 784x100 matrix w/rnd vals
x = tf.placeholder(name="x")                        # Placeholder for input
relu = tf.nn.relu(tf.matmul(W, x) + b)              # Relu(Wx+b)
C = [...]                                           # Cost computed as a function
                                                    # of Relu

s = tf.Session()
for step in xrange(0, 10):
  input = ...construct 100-D input array ...        # Create 100-d vector for input
  result = s.run(C, feed_dict={x: input})           # Fetch cost, feeding x=input
  print step, result
```

(Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., ... & Ghemawat, S. (2016). Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*.)

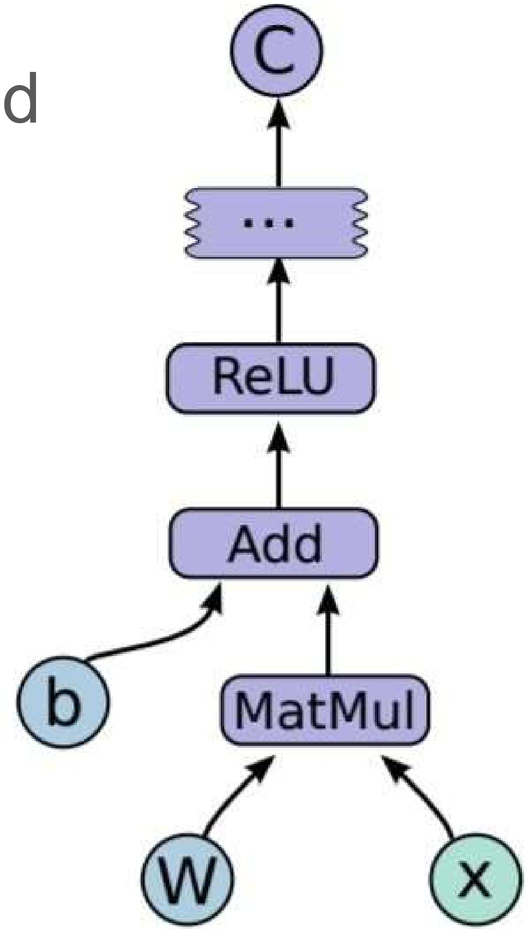
Tensor**Flow**

Operations on tensors are often conceptualized as **graphs**:

```
import tensorflow as tf

b = tf.Variable(tf.zeros([100]))
W = tf.Variable(tf.random_uniform([784,100],-1,1))
x = tf.placeholder(name="x")
relu = tf.nn.relu(tf.matmul(W, x) + b)
C = [...]

s = tf.Session()
for step in xrange(0, 10):
  input = ...construct 100-D input array ...
  result = s.run(C, feed_dict={x: input})
  print step, result
```



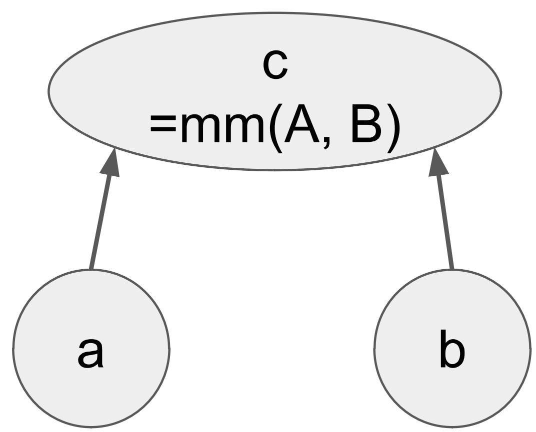
(Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., ... & Ghemawat, S. (2016). Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*.)

Tensor**Flow**

Operations on tensors are often conceptualized as graphs:

A simpler example:

c = tensorflow.matmul(a, b)

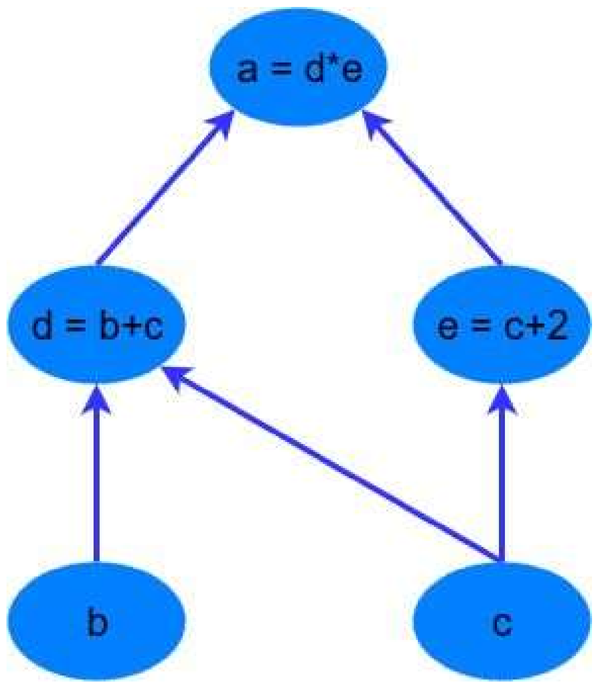


Tensor**Flow**

Operations on tensors are often conceptualized as graphs:

example:

$d = b + c$
$e = c + 2$
$a = d * e$



(Adventures in Machine Learning. *Python TensorFlow Tutorial*, 2017)

* technically, *operations* that work with tensors.

Ingredients of a TensorFlow

tensors*
variables - persistent mutable tensors
constants - constant
placeholders - from data

operations
an abstract computation
(e.g. matrix multiply, add)
executed by device *kernels*

graph

session
defines the environment in which operations *run*.
(like a Spark context)

devices
the specific devices (cpus or gpus) on which to run the session.

* technically, *operations* that work with tensors.

Ingredients of a TensorFlow

tensors*
variables - persistent mutable tensors
constants - constant
placeholders - from data

operations
an abstract computation (e.g. matrix multiply, add) executed by device *kernels*

- tf.Variable(initial_value, name)
- tf.constant(value, type, name)
- tf.placeholder(type, shape, name)

graph

session
defines the environment in which operations *run*.
(like a Spark context)

devices
the specific devices (cpus or gpus) on which to run the session.

Operations

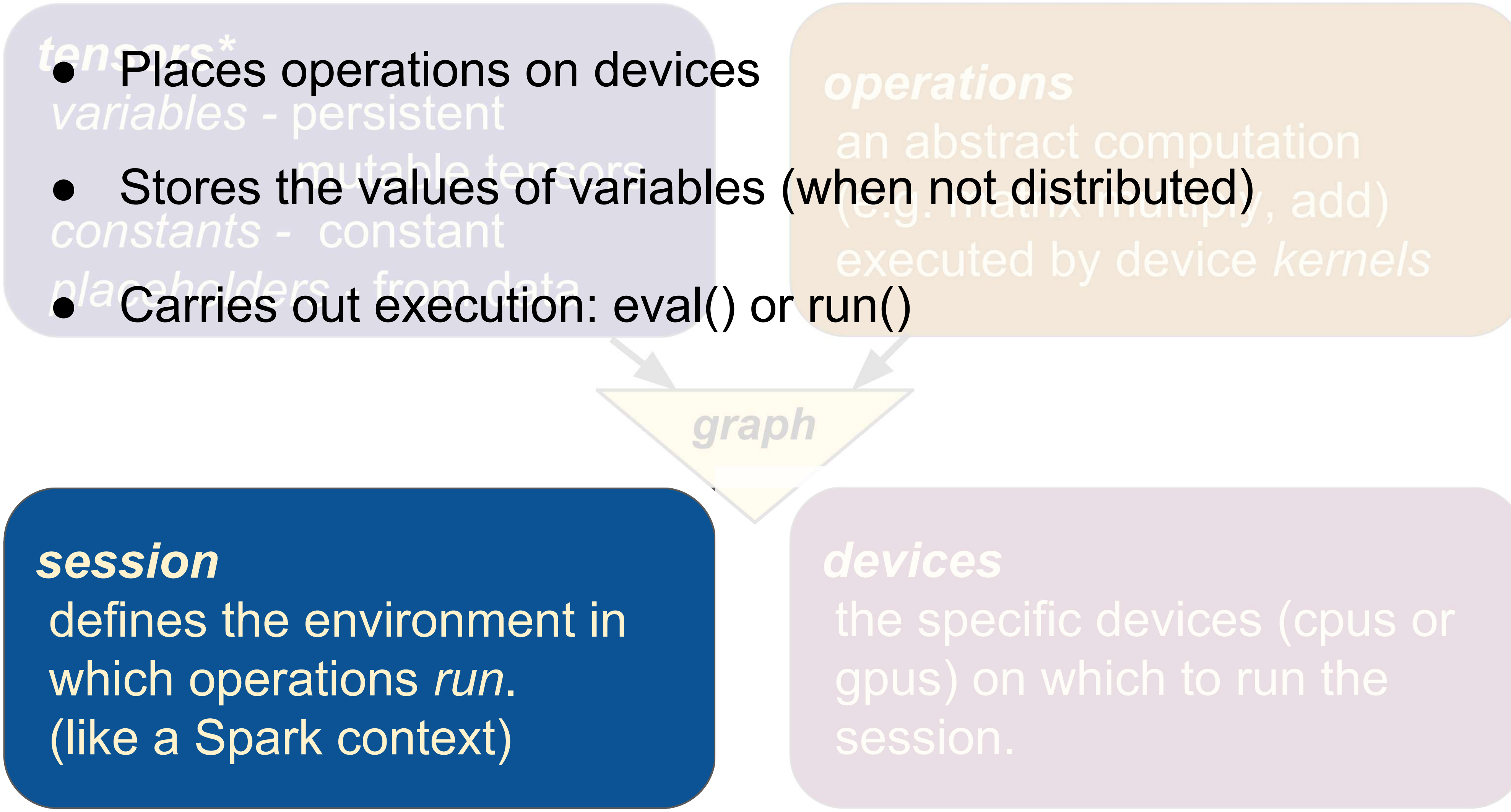
tensors*
variables - persistent
mutable tensors
constants - constant
placeholders - from data

operations
an abstract computation
(e.g. matrix multiply, add)
executed by device *kernels*

Category	**Examples**
Element-wise mathematical operations	Add, Sub, Mul, Div, Exp, Log, Greater, Less, Equal, ...
Array operations	Concat, Slice, Split, Constant, Rank, Shape, Shuffle, ...
Matrix operations	MatMul, MatrixInverse, MatrixDeterminant, ...
Stateful operations	Variable, Assign, AssignAdd, ...
Neural-net building blocks	SoftMax, Sigmoid, ReLU, Convolution2D, MaxPool, ...
Checkpointing operations	Save, Restore
Queue and synchronization operations	Enqueue, Dequeue, MutexAcquire, MutexRelease, ...
Control flow operations	Merge, Switch, Enter, Leave, NextIteration

Sessions

- Places operations on devices
- Stores the values of variables (when not distributed)
- Carries out execution: eval() or run()



* technically, *operations* that work with tensors.

Ingredients of a TensorFlow

tensors*

variables - persistent mutable tensors

constants - constant

placeholders - from data

operations

an abstract computation (e.g. matrix multiply, add) executed by device *kernels*

graph

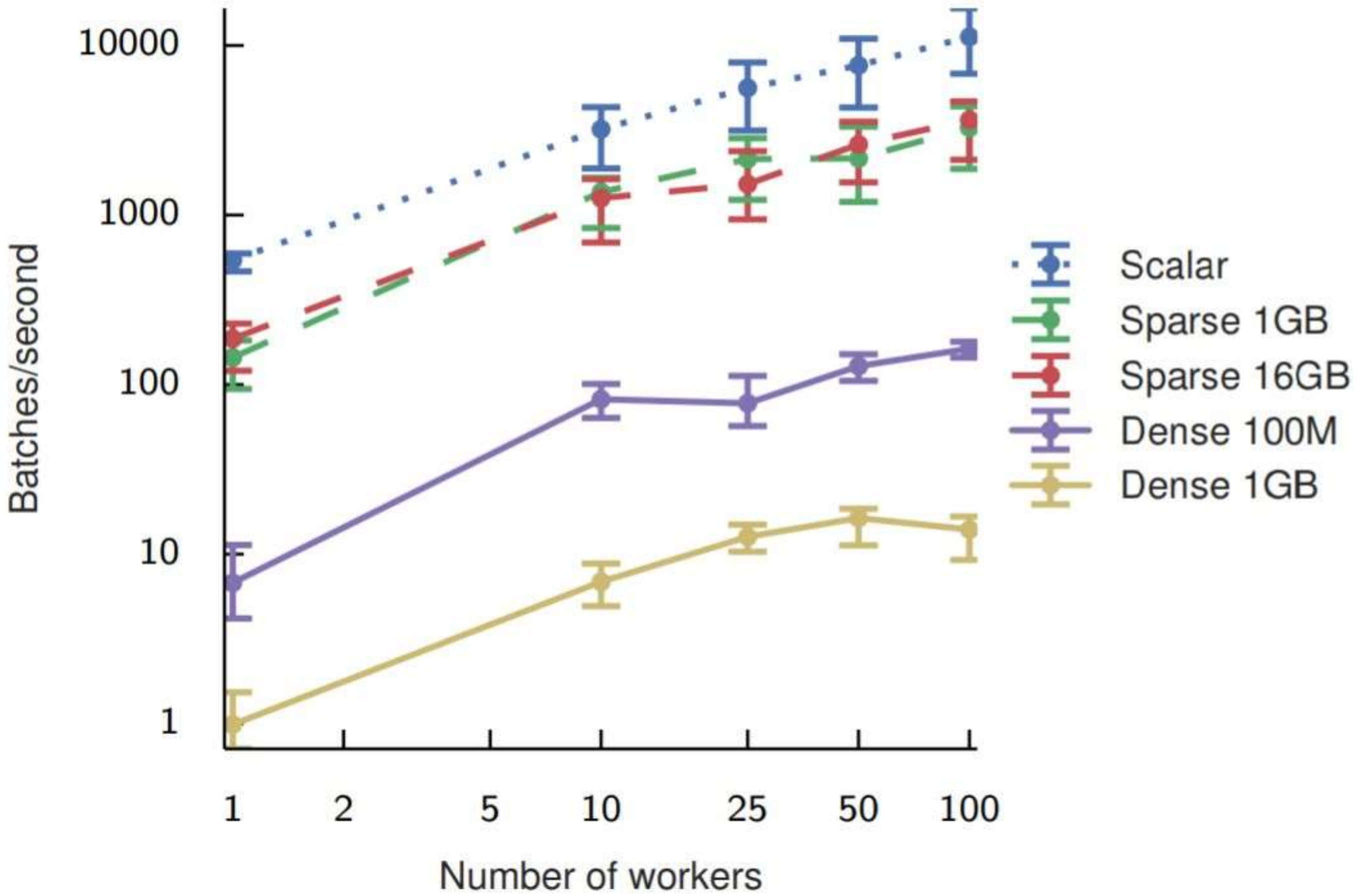
session

defines the environment in which operations *run*. (like a Spark context)

devices

the specific devices (cpus or gpus) on which to run the session.

Distributed TensorFlow



Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... & Kudlur, M. (2016, November). TensorFlow: A System for Large-Scale Machine Learning. In *OSDI* (Vol. 16, pp. 265-283).

Distributed TensorFlow

Distributed:

- Locally: Across processors (cpus, gpus, tpus)
- Across a Cluster: Multiple machine with multiple processors

Distributed TensorFlow

Distributed:

- Locally: Across processors (cpus, gpus, tpus)
- Across a Cluster: Multiple machine with multiple processors

Parallelisms:

- Data Parallelism: All nodes doing same thing on different subsets of data
- Graph/Model Parallelism: Different portions of model on different devices

Distributed TensorFlow

Distributed:

- Locally: Across processors (cpus, gpus, tpus)
- Across a Cluster: Multiple machine with multiple processors

Parallelisms:

- Data Parallelism: All nodes doing same thing on different subsets of data
- Graph/Model Parallelism: Different portions of model on different devices

Model Updates:

- Asynchronous Parameter Server
- Synchronous AllReduce (doesn’t work with Model Parallelism)

Distributed TensorFlow

Distributed:

- Locally: Across processors (cpus, gpus, tpus)
- Across a Cluster: Multiple machine with multiple processors

Parallelisms:

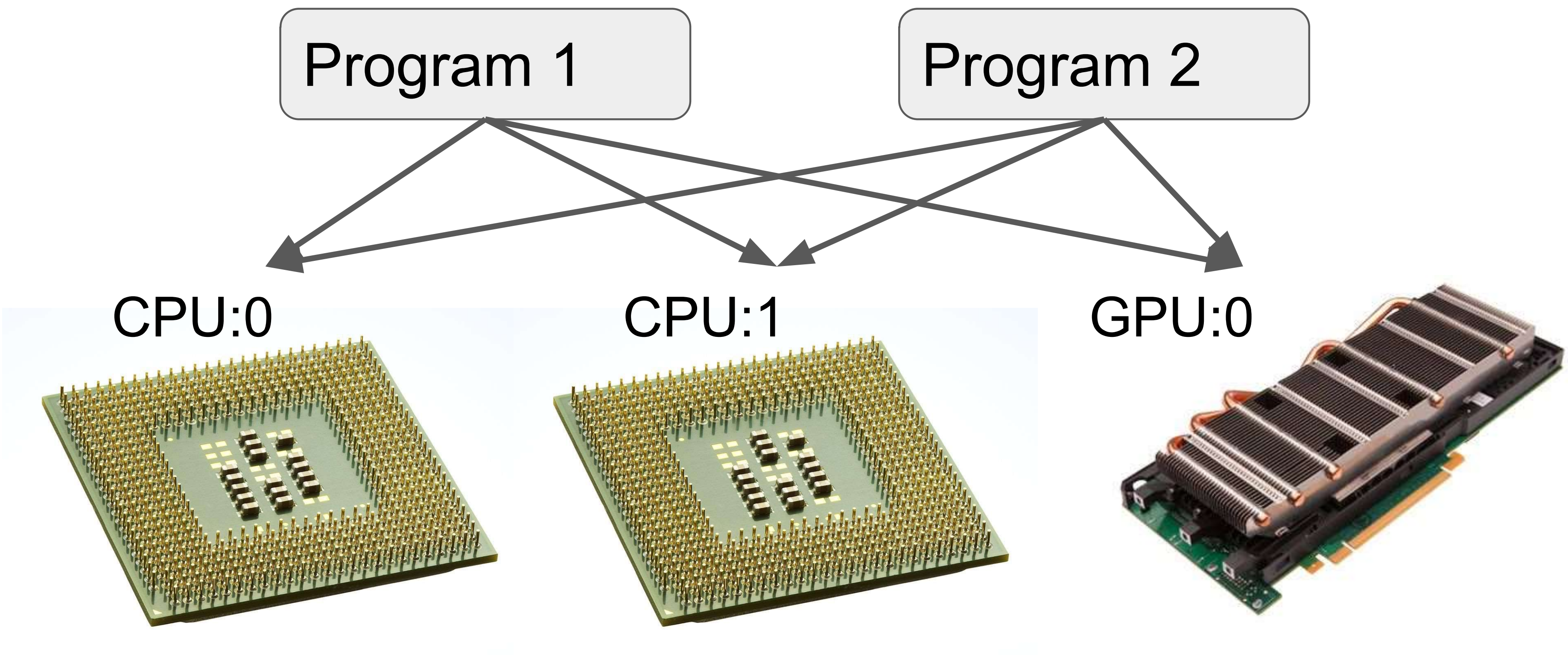
- Data Parallelism: All nodes doing same thing on different subsets of data
- Graph/Model Parallelism: Different portions of model on different devices

Model Updates:

- Asynchronous Parameter Server
- Synchronous AllReduce (doesn’t work with Model Parallelism)

Local Distribution

Multiple devices on single machine

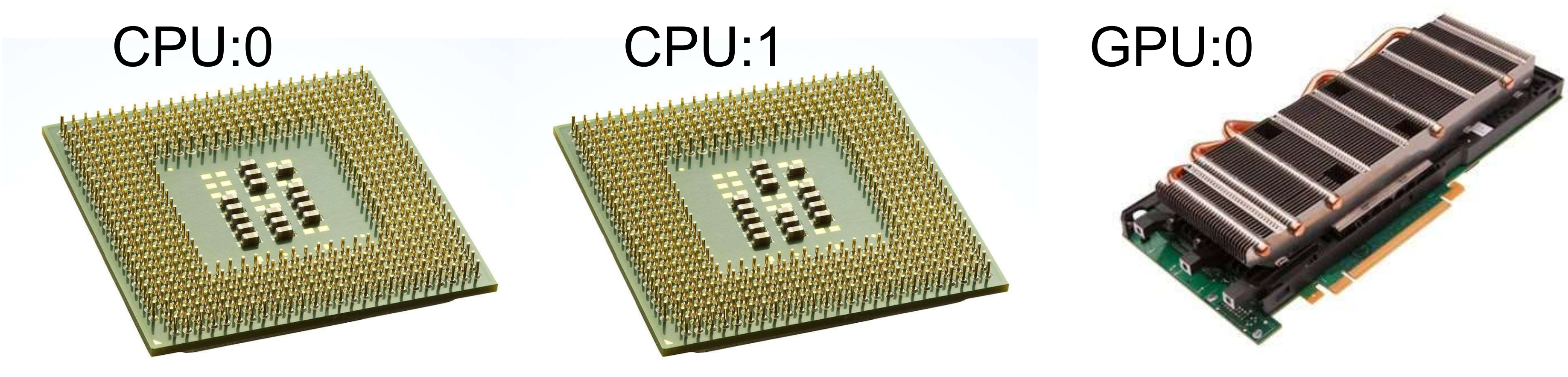


Local Distribution

Multiple devices on single machine

```
with tf.device("/cpu:1")
   beta=tf.Variable(...)
```

```
with tf.device("/gpu:0")
   y_pred=tf.matmul(beta,X)
```

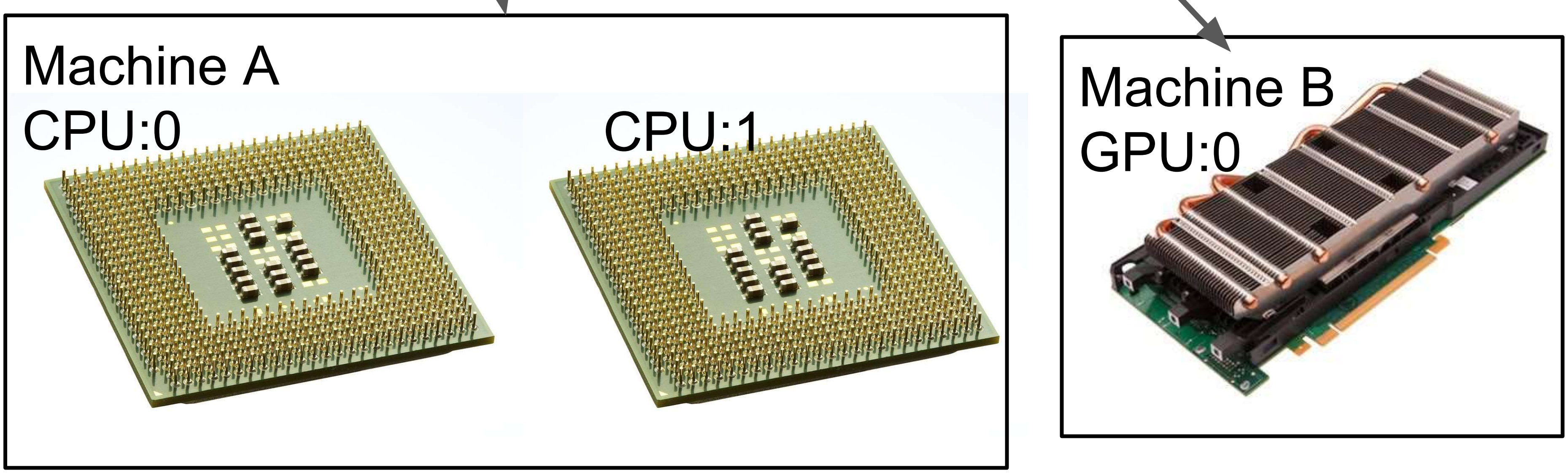


Cluster Distribution

Multiple devices on multiple machines

```
with tf.device("/cpu:1")
   beta=tf.Variable(...)
```

```
with tf.device("/gpu:0")
   y_pred=tf.matmul(beta,X)
```



Distributed TensorFlow

Distributed:

- Locally: Across processors (cpus, gpus, tpus)
- Across a Cluster: Multiple machine with multiple processors

Parallelisms:

- Data Parallelism: All nodes doing same thing on different subsets of data
- Graph/Model Parallelism: Different portions of model on different devices

Model Updates:

- Asynchronous Parameter Server
- Synchronous AllReduce (doesn’t work with Model Parallelism)

Distributed TensorFlow

Distributed:

- Locally: Across processors (cpus, gpus, tpus)
- Across a Cluster: Multiple machine with multiple processors

Parallelisms:

- Data Parallelism: All nodes doing same thing on different subsets of data
- Graph/Model Parallelism: Different portions of model on different devices

Model Updates:

- Asynchronous Parameter Server
- Synchronous AllReduce (doesn’t work with Model Parallelism)

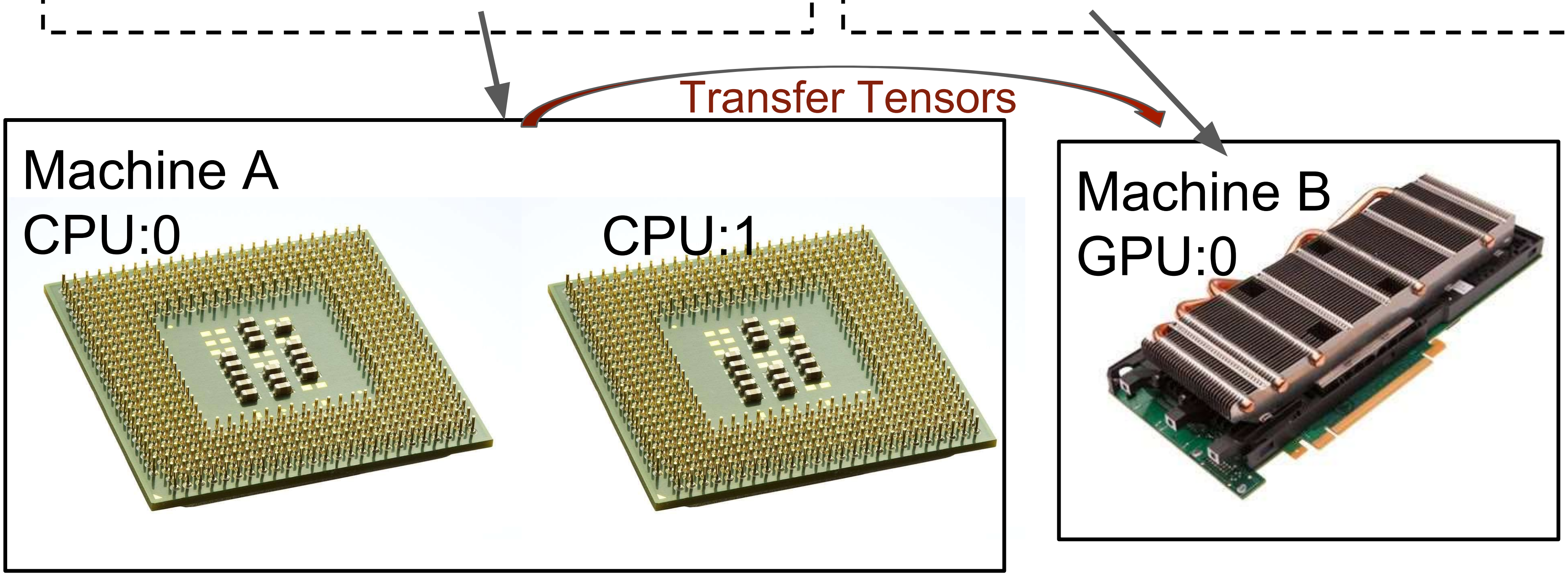
Cluster Distribution

Model Parallelism

Multiple devices on multiple machines

```
with tf.device("/cpu:1")
   beta=tf.Variable(...)
```

```
with tf.device("/gpu:0")
   y_pred=tf.matmul(beta,X)
```



Cluster Distribution

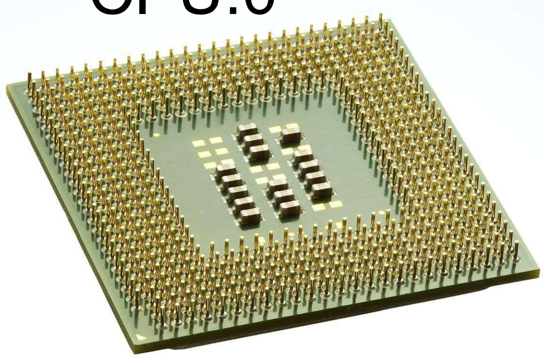
Data Parallelism

```
...
  beta=tf.Variable(...)
  pred=tf.matmul(beta,X)
```

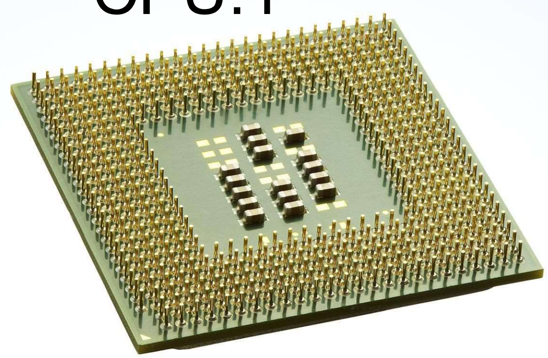
```
...
  beta=tf.Variable(...)
  pred=tf.matmul(beta,X)
```

```
...
  beta=tf.Variable(...)
  pred=tf.matmul(beta,X)
```

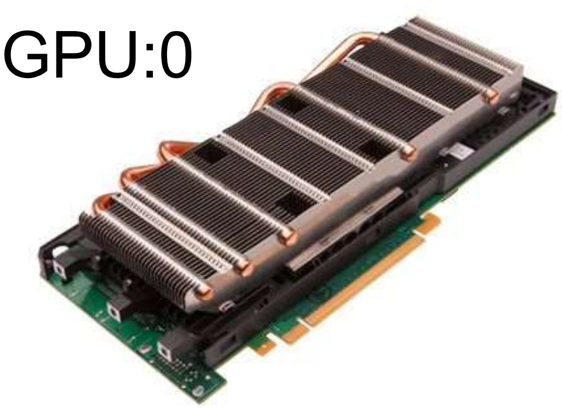
CPU:0



CPU:1



GPU:0



Distributed TensorFlow

Distributed:

- Locally: Across processors (cpus, gpus, tpus)
- Across a Cluster: Multiple machine with multiple processors

Parallelisms:

- Data Parallelism: All nodes doing same thing on different subsets of data
- Graph/Model Parallelism: Different portions of model on different devices

Model Updates:

- Asynchronous Parameter Server
- Synchronous AllReduce (doesn’t work with Model Parallelism)

Distributed TensorFlow

Distributed:

- Locally: Across processors (cpus, gpus, tpus)
- Across a Cluster: Multiple machine with multiple processors

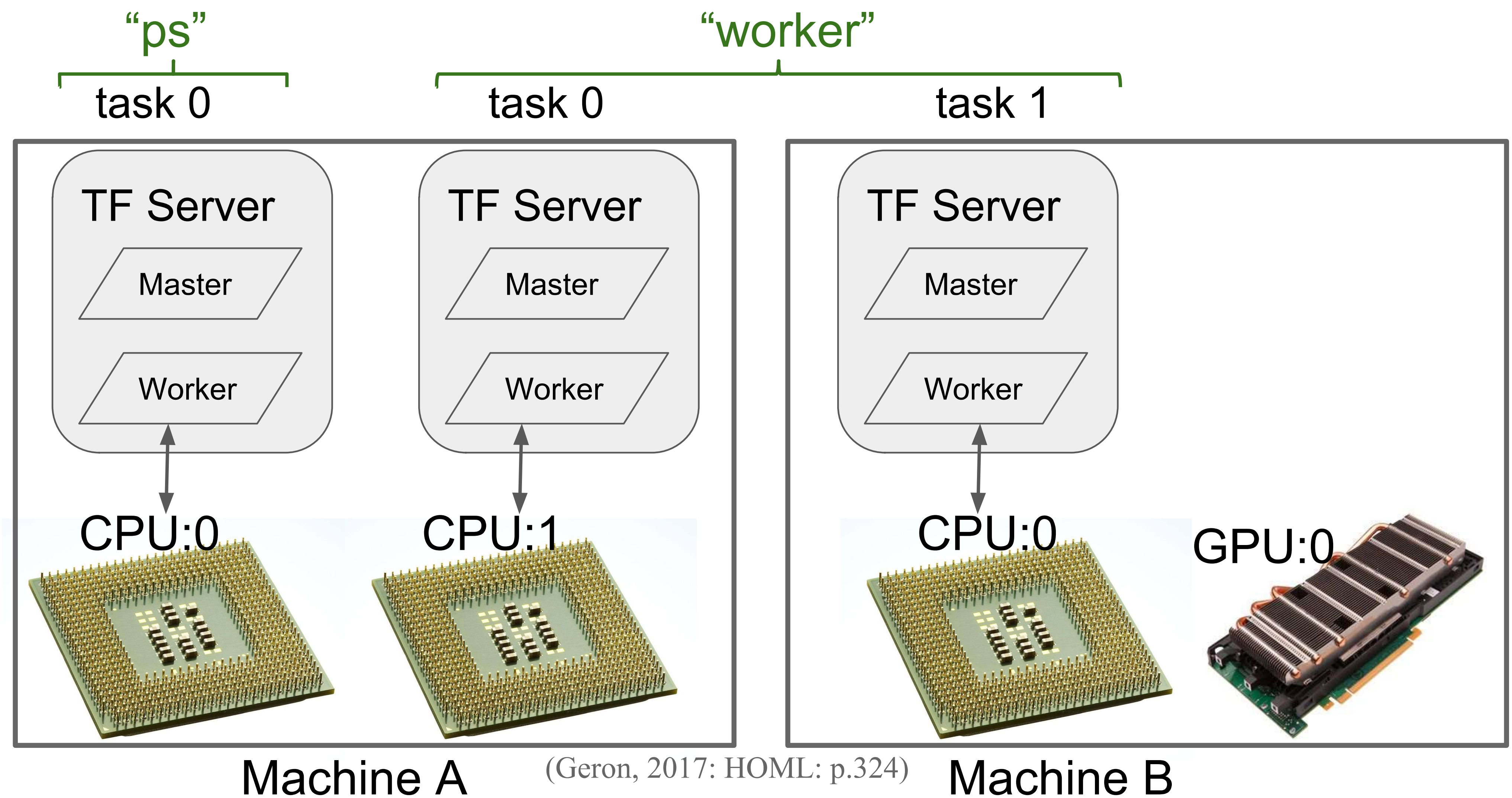
Parallelisms:

- Data Parallelism: All nodes doing same thing on different subsets of data
- Graph/Model Parallelism: Different portions of model on different devices

Model Updates:

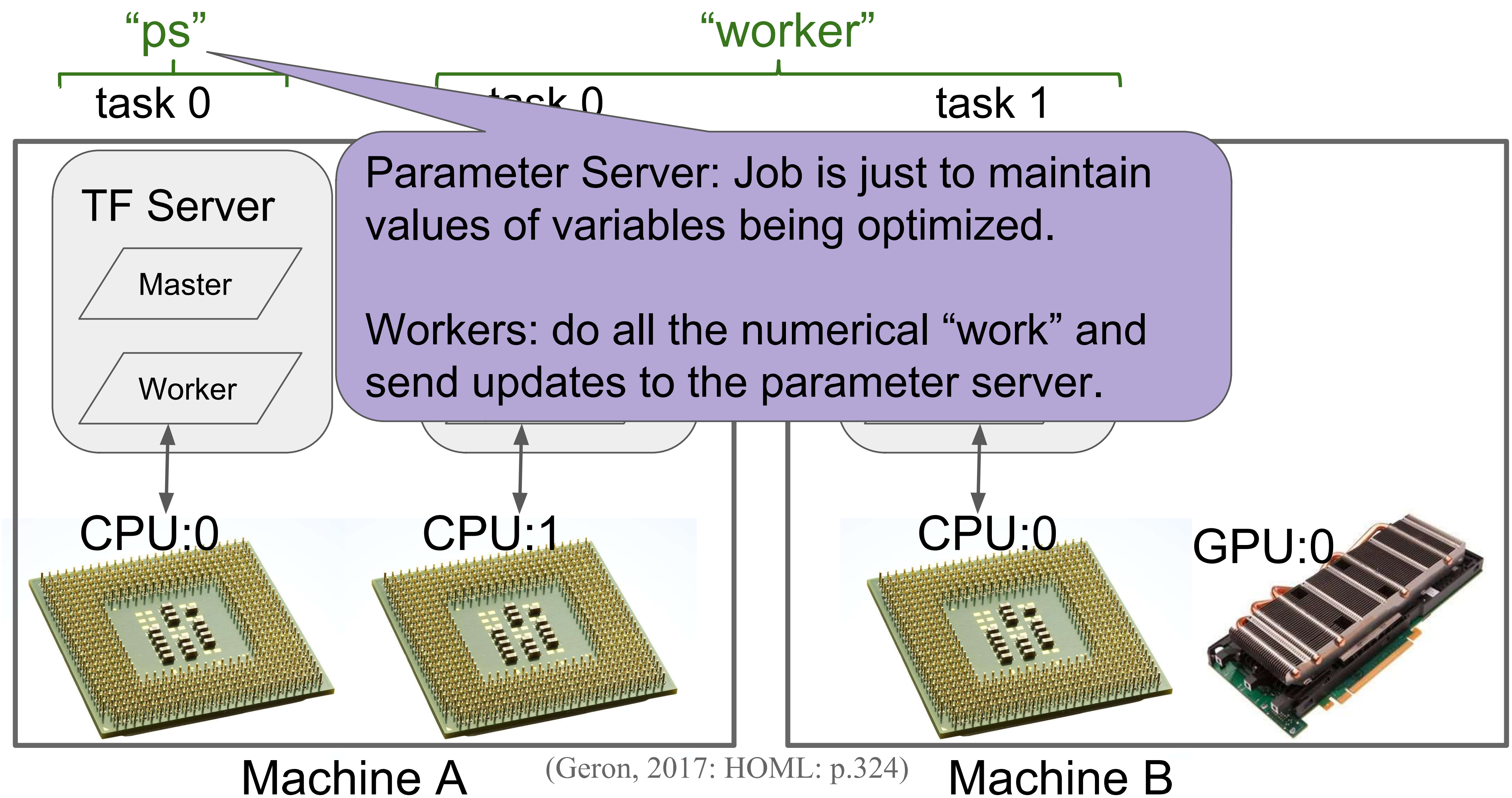
- Asynchronous Parameter Server
- Synchronous AllReduce (doesn’t work with Model Parallelism)

Asynchronous Parameter Server



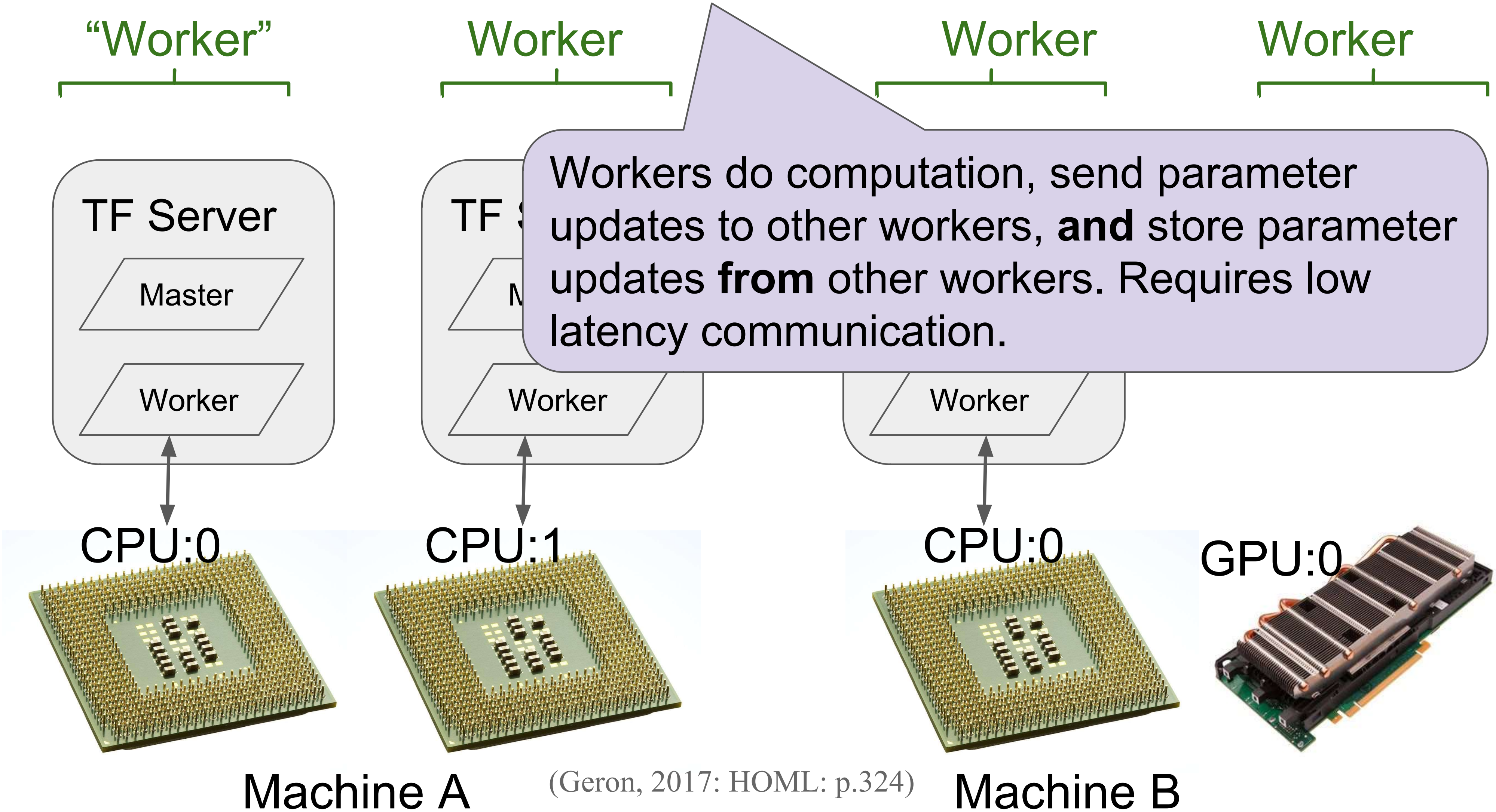
(Geron, 2017: HOML: p.324)

Asynchronous Parameter Server



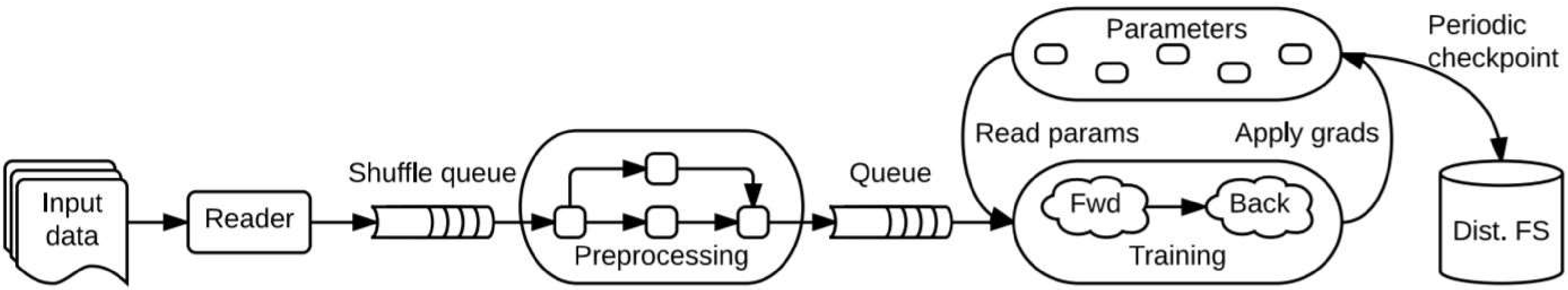
(Geron, 2017: HOML: p.324)

Synchronous AllReduce



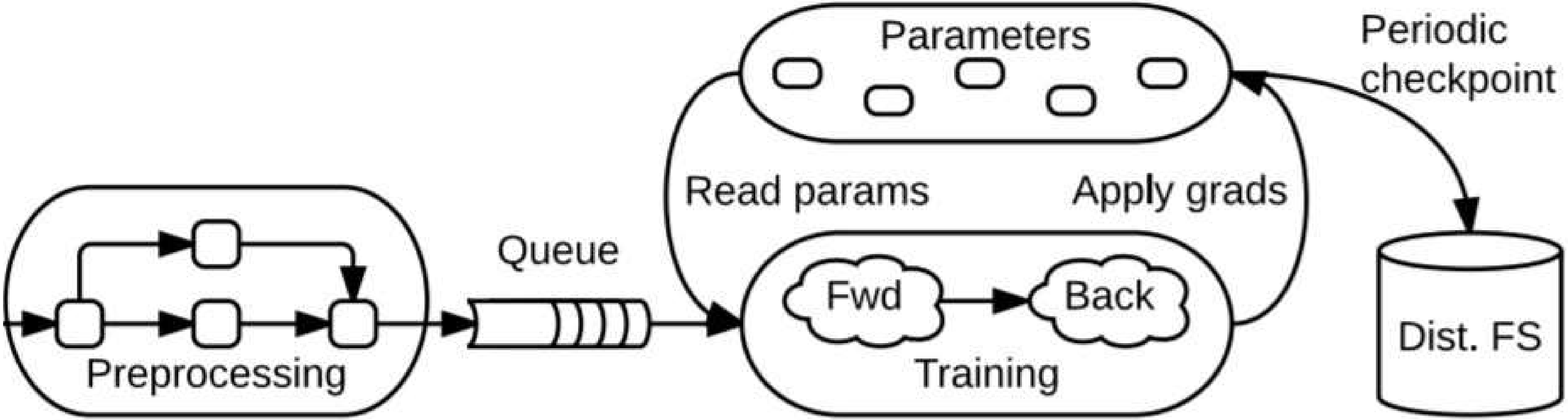
(Geron, 2017: HOML: p.324)

Distributed TensorFlow: Full Pipeline



Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... & Kudlur, M. (2016, November). TensorFlow: A System for Large-Scale Machine Learning. In *OSDI* (Vol. 16, pp. 265-283).

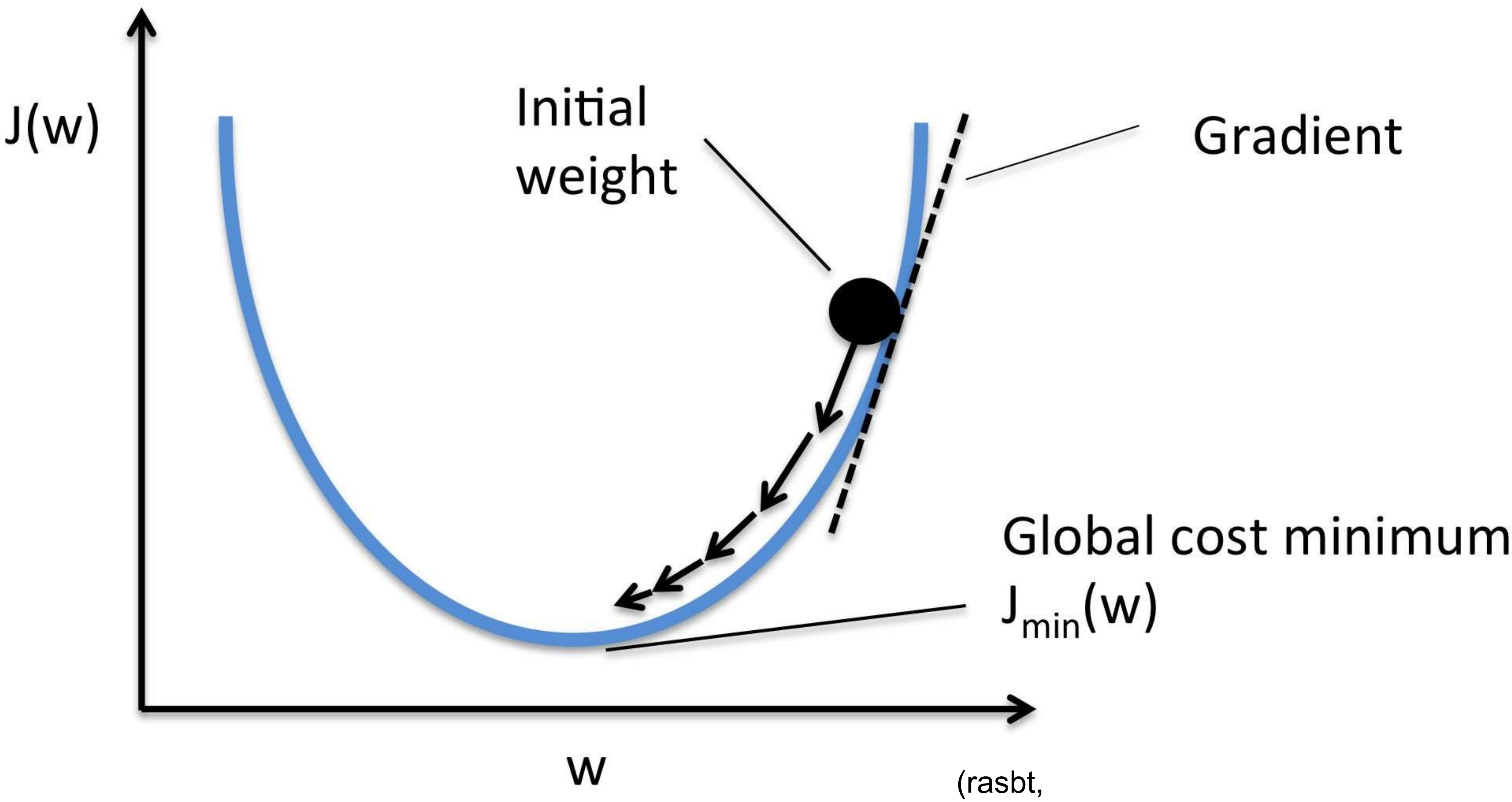
Parameters -- derived from **gradients**



TensorFlow has built-in ability to derive gradients given a cost function.

```
tf.gradients(cost, [params])
```

Parameters -- derived from **gradients**



(rasbt,

Parameters -- derived from **gradients.**

Linear Regression: Trying to find “betas” that minimize:

$$\hat{\beta} = argmin_{\beta}\{\sum_{i}^{N}(y_i - \hat{y}_i)^2\}$$

Parameters -- derived from **gradients.**

Linear Regression: Trying to find “betas” that minimize:

$$\hat{\beta} = argmin_{\beta}\{\sum_{i}^{N}(y_i - \hat{y}_i)^2\}$$

matrix multiply

$$\hat{y}_i = X_i\beta$$

Thus:

$$\hat{\beta} = argmin_{\beta}\{\sum_{i=0}^{N}(y_i - X_i\beta)^2\}$$

Parameters -- derived from **gradients.**

Linear Regression: Trying to find “betas” that minimize:

$$\hat{\beta} = argmin_{\beta}\{\sum_{i}^{N}(y_i - \hat{y}_i)^2\}$$

matrix multiply

$\hat{y}_i = X_i\beta$ Thus: $$\hat{\beta} = argmin_{\beta}\{\sum_{i=0}^{N}(y_i - X_i\beta)^2\}$$

In standard linear equation:

$$y = mx + b$$

$$\text{let } x' = x + [1, 1, ..., 1]_N^T$$
$$\text{then, } y = mx'$$

(if we add a column of 1s, mx + b is just matmul(m, x))

Parameters -- derived from **gradients.**

Linear Regression: Trying to find "betas" that minimize:

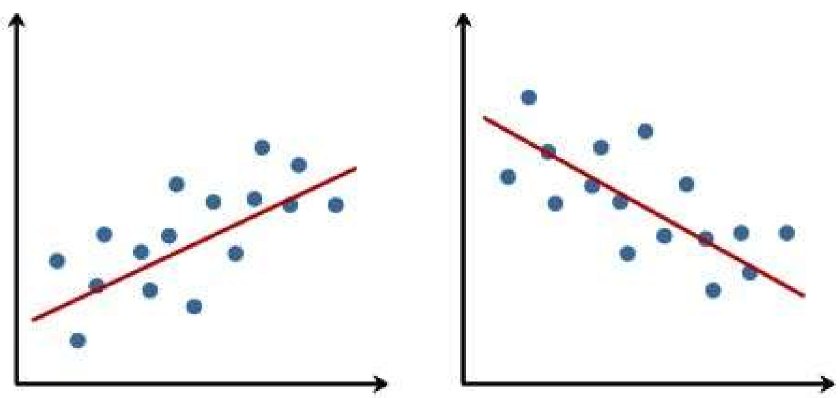
$$\hat{\beta} = argmin_{\beta}\{\sum_{i}^{N}(y_i - \hat{y}_i)^2\}$$

matrix multiply

$$\hat{y}_i = X_i\beta$$

Thus:

$$\hat{\beta} = argmin_{\beta}\{\sum_{i=0}^{N}(y_i - X_i\beta)^2\}$$



Parameters -- derived from **gradients.**

Linear Regression: Trying to find "betas" that minimize:

$$\hat{\beta} = argmin_{\beta}\{\sum_{i}^{N}(y_i - \hat{y}_i)^2\}$$

$\hat{y}_i = X_i\beta$ Thus: $$\hat{\beta} = argmin_{\beta}\{\sum_{i=0}^{N}(y_i - X_i\beta)^2\}$$

How to update? $\beta_{new} = \beta_{prev} - \alpha * \text{grad}$

Parameters -- derived from **gradients.**

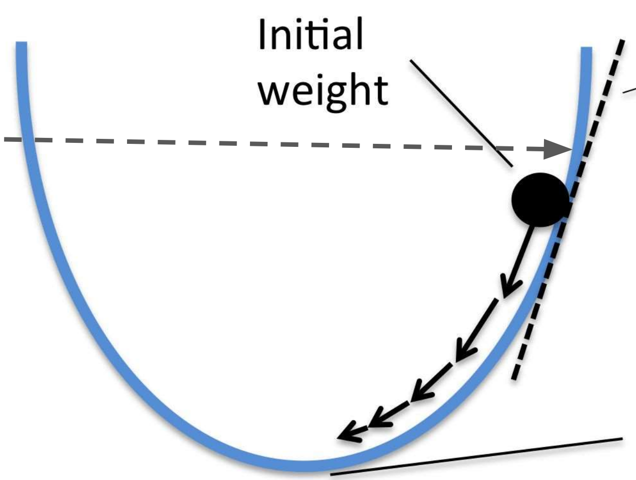
Linear Regression: Trying to find "betas" that minimize:

$$\hat{\beta} = argmin_{\beta}\{\sum_{i}^{N}(y_i - \hat{y}_i)^2\}$$

$\hat{y}_i = X_i\beta$ Thus: $$\hat{\beta} = argmin_{\beta}\{\sum_{i=0}^{N}(y_i - X_i\beta)^2\}$$

How to update? $\beta_{new} = \beta_{prev} - \alpha * \text{grad}$

(for gradient descent) "learning rate"



Parameters -- derived from **gradients.**

Ridge Regression (L2 Penalized linear regression, $\lambda||\beta||_2^2$)

$$\hat{\beta}^{ridge} = argmin_{\beta}\{\sum_{i=1}^{N}(y_i - \sum_{j=1}^{m} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{m} \beta_j^2\}$$

1. Matrix Solution:

$$\hat{\beta}^{ridge} = (X^T X + \lambda I)^{-1} X^T y$$

Demo

Ridge Regression (L2 Penalized linear regression, $\lambda||\beta||_2^2$)

$$\hat{\beta}^{ridge} = argmin_\beta\{\sum_{i=1}^{N}(y_i - \sum_{j=1}^{m} x_{ij}\beta_j)^2 + \lambda\sum_{j=1}^{m}\beta_j^2\}$$

2. Gradient descent solution
(Mirrors many parameter optimization problems.)

1. Matrix Solution:

$$\hat{\beta}^{ridge} = (X^TX + \lambda I)^{-1}X^Ty$$

Gradients

Ridge Regression (L2 Penalized linear regression, $\lambda||\beta||_2^2$)

$$\hat{\beta}^{ridge} = argmin_{\beta}\{\sum_{i=1}^{N}(y_i - \sum_{j=1}^{m} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{m} \beta_j^2\}$$

Gradient descent needs to solve.
(Mirrors many parameter optimization problems.)

TensorFlow has built-in ability to derive gradients given a cost function.

Gradients

Ridge Regression

(L2 Penalized linear regression, $\lambda||\beta||_2^2$)

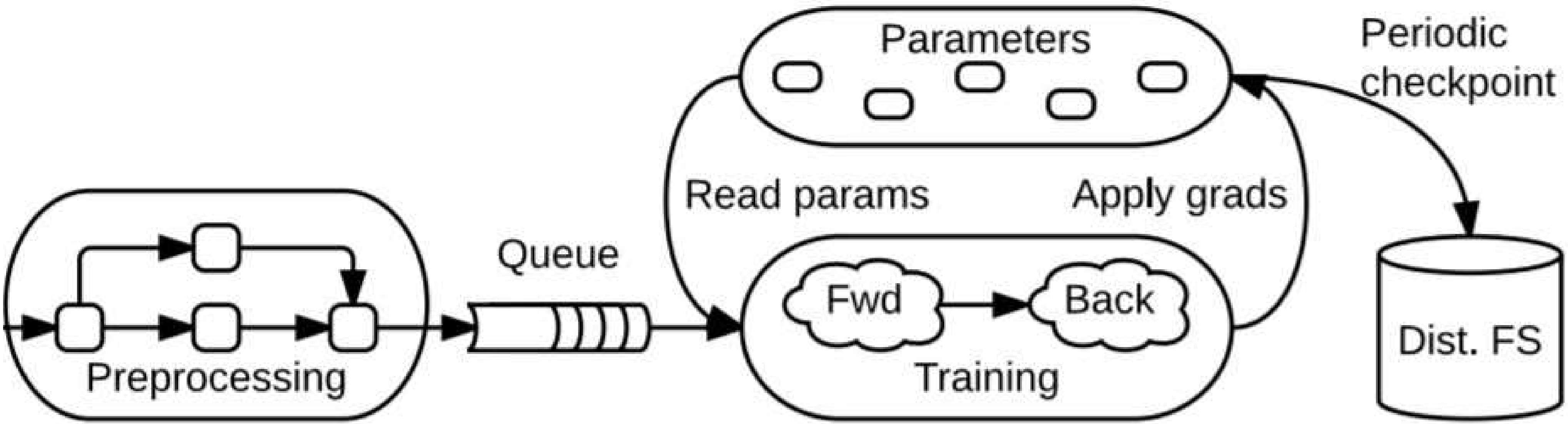
$$\hat{\beta}^{ridge} = argmin_\beta\{\sum_{i=1}^{N}(y_i - \sum_{j=1}^{m} x_{ij}\beta_j)^2 + \lambda\sum_{j=1}^{m}\beta_j^2\}$$

Gradient descent needs to solve.
(Mirrors many parameter optimization problems.)

TensorFlow has built-in ability to derive gradients given a cost function.

```
tf.gradients(cost, [params])
```

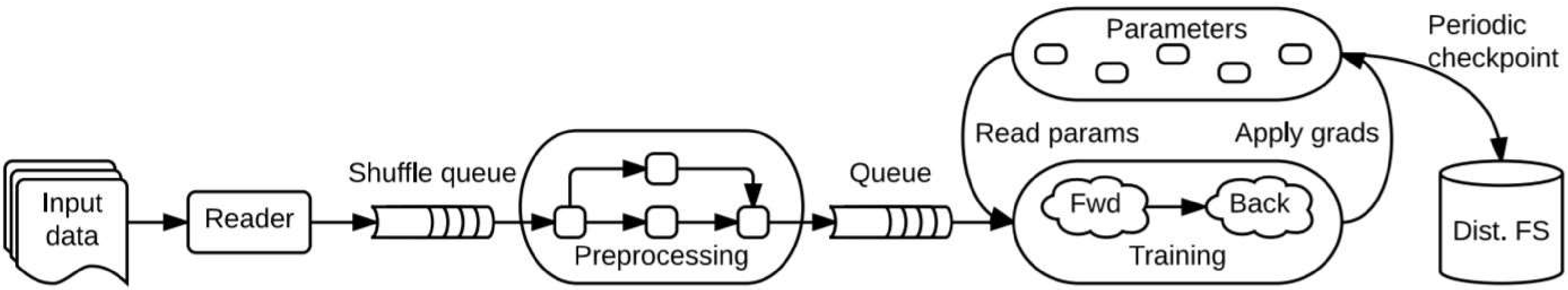
Gradients



TensorFlow has built-in ability to derive gradients given a cost function.

```
tf.gradients(cost, [params])
```

Distributed TensorFlow: Full Pipeline



Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... & Kudlur, M. (2016, November). TensorFlow: A System for Large-Scale Machine Learning. In *OSDI* (Vol. 16, pp. 265-283).