Distributed TensorFlow

CSE545 - Spring 2020

Stony Brook University

H. Andrew Schwartz

Big Data Analytics, The Class

Goal: Generalizations
A *model* or *summarization* of the data.

Data Frameworks

- Hadoop File System ✓
- Spark ✓
- Streaming ✓
- MapReduce ✓
- Tensorflow

Algorithms and Analyses

- Similarity Search
- Hypothesis Testing
- Graph Analysis
- Recommendation Systems
- Deep Learning

Limitations of Spark

Spark is fast for being so flexible

- Fast: RDDs in memory + Lazy evaluation: optimized chain of operations.
- Flexible: Many transformations -- can contain any custom code.

Limitations of Spark

Spark is fast for being so flexible

- Fast: RDDs in memory + Lazy evaluation: optimized chain of operations.
- Flexible: Many transformations -- can contain any custom code.

However:

- Hadoop MapReduce can still be better for extreme IO, data that will not fit in memory across cluster.

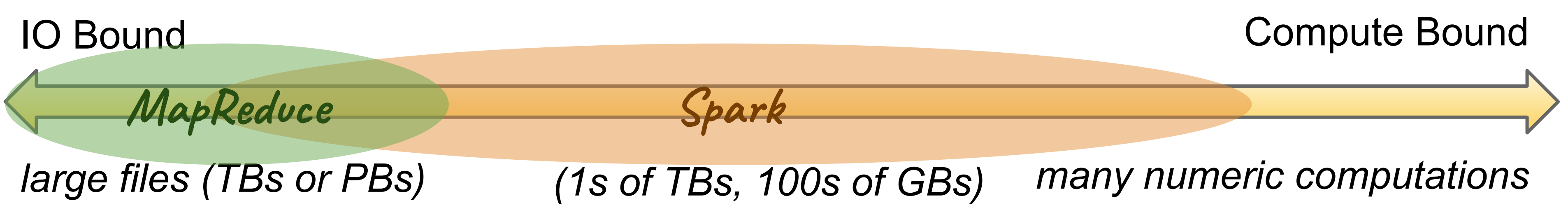
Limitations of Spark

Spark is fast for being so flexible

- Fast: RDDs in memory + Lazy evaluation: optimized chain of operations.
- Flexible: Many transformations -- can contain any custom code.

However:

- Hadoop MapReduce can still be better for extreme IO, data that will not fit in memory across cluster.



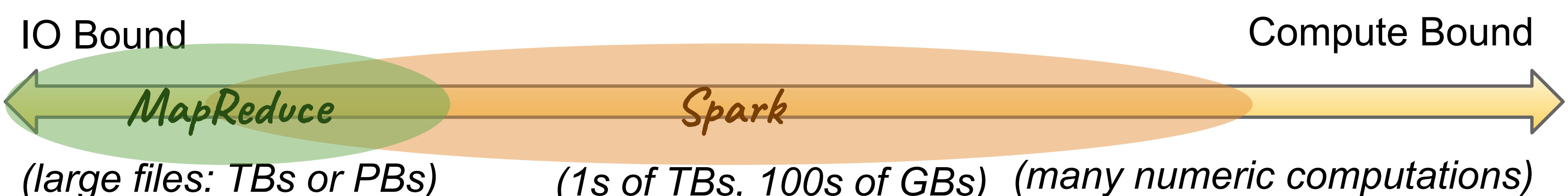
Limitations of Spark

Spark is fast for being so flexible

- Fast: RDDs in memory + Lazy evaluation: optimized chain of operations.
- Flexible: Many transformations -- can contain any custom code.

However:

- Hadoop MapReduce can still be better for extreme IO, data that will not fit in memory across cluster.
- Modern machine learning (esp. Deep learning), a common big data task, requires heavy numeric computation.



* this is the subjective approximation of the instructor as of February 2020. A lot of factors at play.

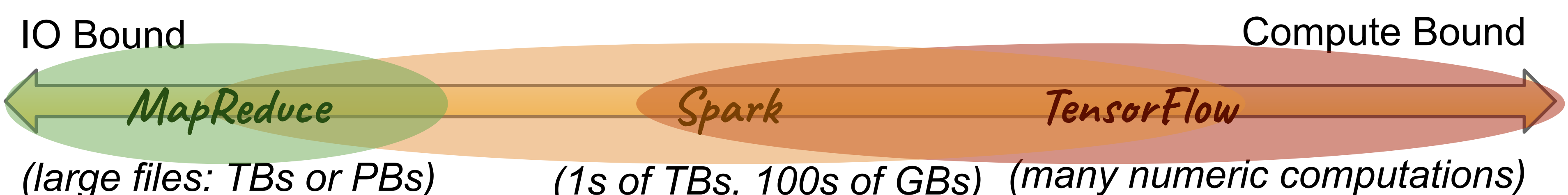
Limitations of Spark

Spark is fast for being so flexible

- Fast: RDDs in memory + Lazy evaluation: optimized chain of operations.
- Flexible: Many transformations -- can contain any custom code.

However:

- Hadoop MapReduce can still be better for extreme IO, data that will not fit in memory across cluster.
- Modern machine learning (esp. Deep learning), a common big data task, requires heavy numeric computation.



* this is the subjective approximation of the instructor as of February 2020. A lot of factors at play.

Learning Objectives

- 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)

What is TensorFlow?

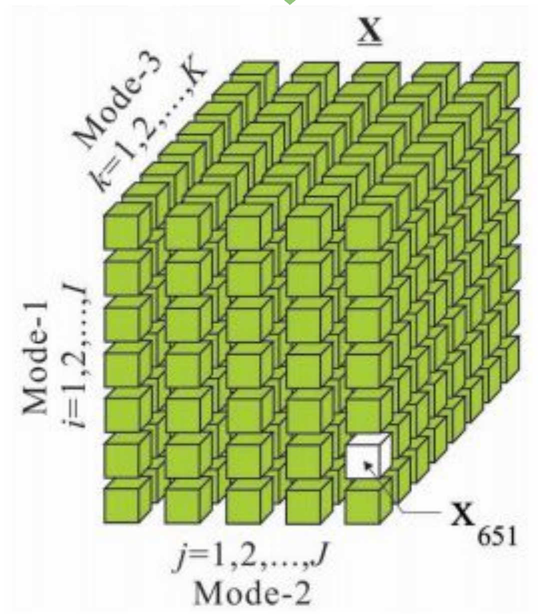
A workflow system catered to numerical computation.

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

What is **Tensor**Flow?

A workflow system catered to numerical computation.

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



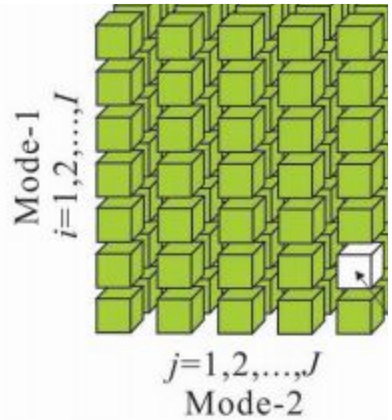
A multi-dimensional matrix

(i.stack.imgur.com)

What is **Tensor**Flow?

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

What is **Tensor**Flow?

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]

What is **Tensor**Flow?

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?

What is TensorFlow?

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

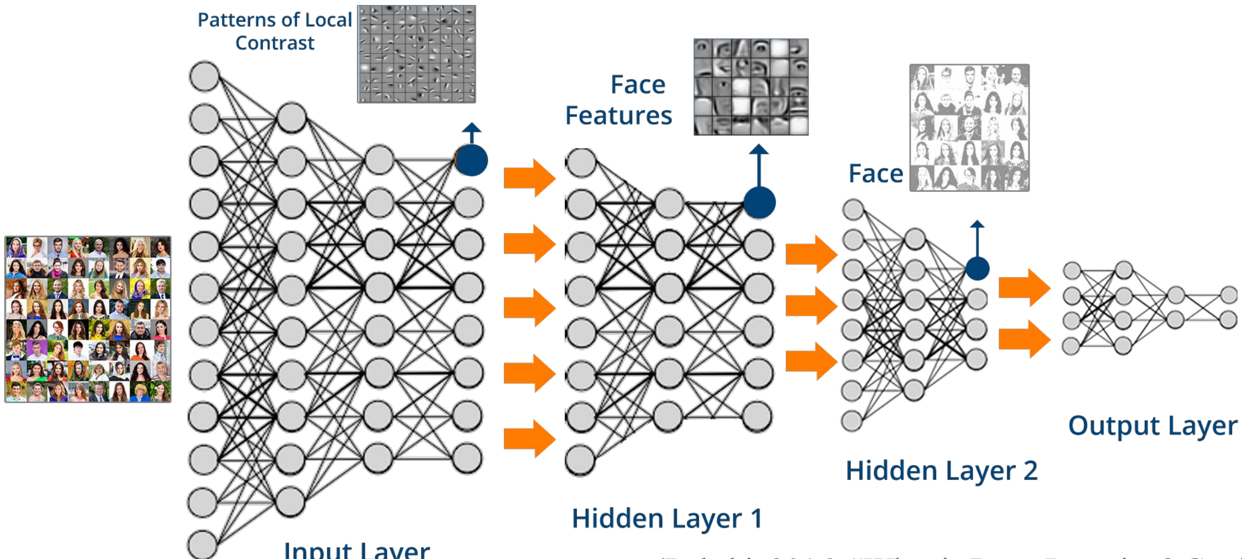
enables complex models, like deep learning

Then, why TensorFlow?

What is TensorFlow?

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

enables complex models, like deep learning



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

What is 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*.)

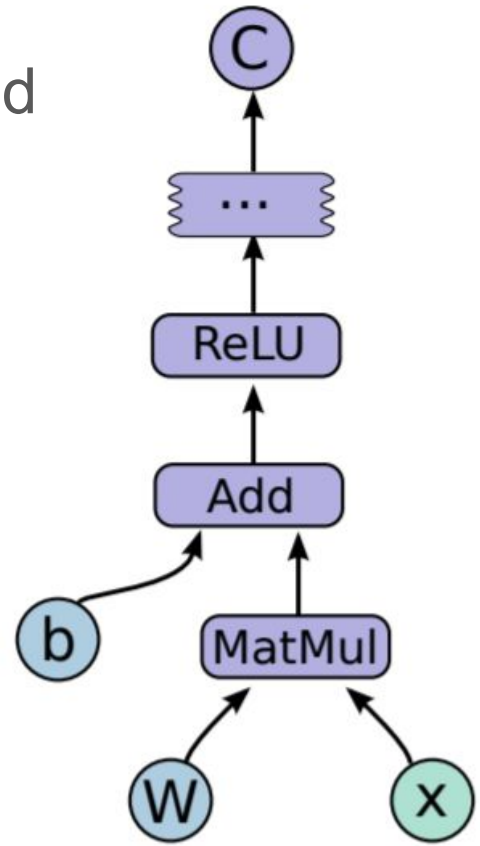
Tensor**Flow**

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
```



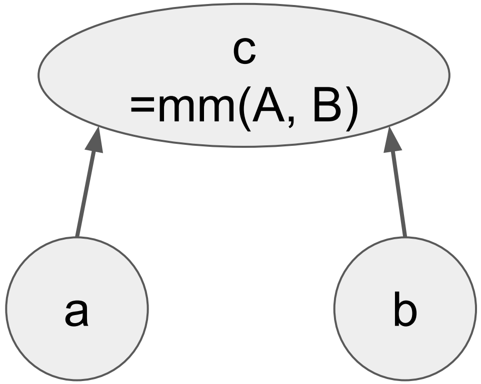
(Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., ... & Ghemawat, S. (2016). Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*.)

TensorFlow

Operations on tensors are often conceptualized as graphs:

A simpler example:

c = tensorflow.matmul(a, b)

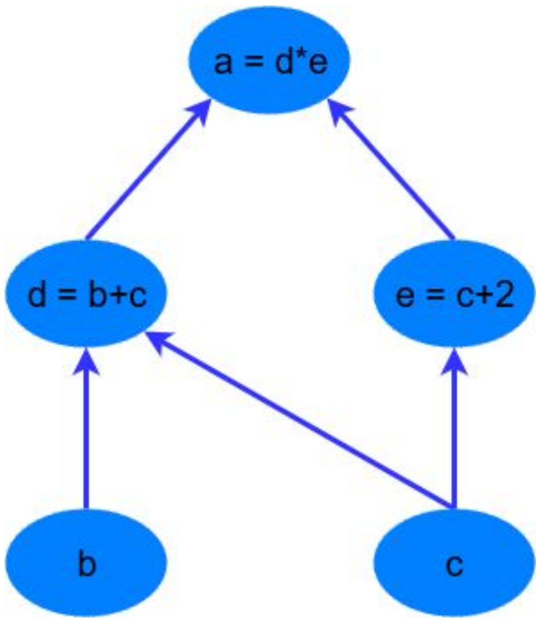


TensorFlow

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)

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*

* technically, still *operations*

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.

Ingredients of a TensorFlow

tensors*

variables - persistent mutable tensors

constants - constant

placeholders - from data

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

operations

an abstract computation (e.g. matrix multiply, add)

executed by device *kernels*

* technically, still *operations*

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.

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*

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

Ingredients of a TensorFlow

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

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.

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

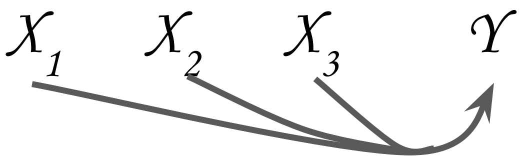
Typical use-case: (Supervised Machine Learning)

Determine weights, $\mathcal{W}$, of a function, f, such that ε is minimized: $f(\mathcal{X} | \mathcal{W}) = \mathcal{Y} + \varepsilon$

Distributed TensorFlow

Typical use-case:

Determine weights, $\mathcal{W}$, of a function, f, such that ε is minimized: $f(X \mid \mathcal{W}) = Y + \varepsilon$



Distributed TensorFlow

Typical use-case:

Determine weights, $\mathcal{W}$, of a function, f, such that ε is minimized: $f(\mathcal{X}\,|\,\mathcal{W}) = \mathcal{Y} + \varepsilon$

$X_1 \quad X_2 \quad X_3 \qquad Y$

$X_1 \quad X_2 \quad X_3 \quad X_4 \quad X_5 \quad X_6$

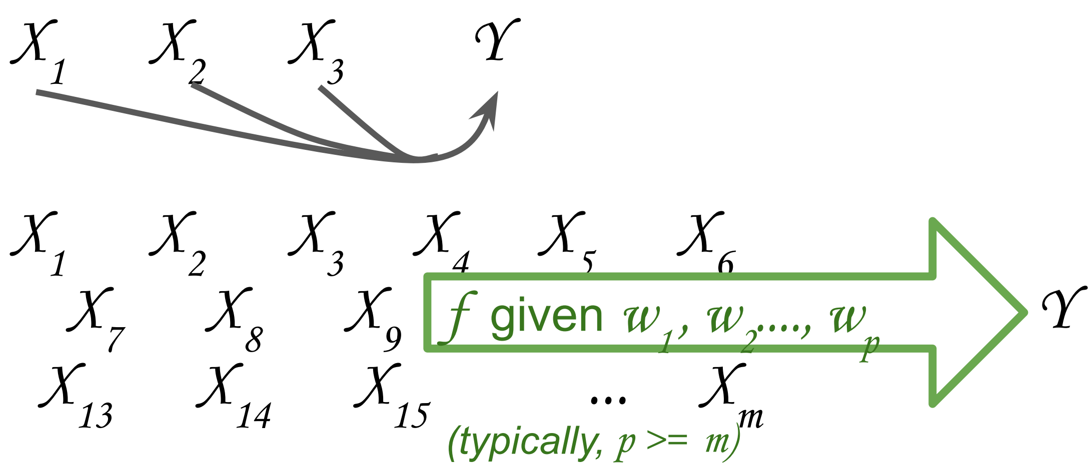
$X_7 \quad X_8 \quad X_9 \quad X_{10} \quad X_{11} \quad X_{12} \qquad\qquad Y$

$X_{13} \quad X_{14} \quad X_{15} \quad \ldots \quad X_m$

Distributed TensorFlow

Typical use-case:

Determine weights, $\mathcal{W}$, of a function, f, such that ε is minimized: $f(\mathcal{X} \mid \mathcal{W}) = \mathcal{Y} + \varepsilon$



Distributed TensorFlow

Typical use-case:

Determine weights, W, of a function, f, such that $|\varepsilon|$ is minimized:

$$f(X|W) = \hat{Y}$$
$$Y = (X|W) + \varepsilon$$
$$Y = \hat{Y} + \varepsilon$$
$$\varepsilon = \hat{Y} - Y$$

$X_1 \quad X_2 \quad X_3 \quad \rightarrow \quad Y$

$X_1 \quad X_2 \quad X_3 \quad X_4 \quad X_5 \quad X_6$
$X_7 \quad X_8 \quad X_9 \quad \xrightarrow{f \text{ given } w_1, w_2, \ldots, w_p} \quad Y$
$X_{13} \quad X_{14} \quad X_{15} \quad \ldots \quad X_m$

(typically, $p >= m$)

Distributed TensorFlow

Typical use-case:

Determine weights, $\mathcal{W}$, of a function, f, such that ε is minimized: f

$$F(X|W) = \hat{Y}$$
$$Y = (X|W) + \varepsilon$$
$$Y = \hat{Y} + \varepsilon$$
$$\varepsilon = \hat{Y} - Y$$

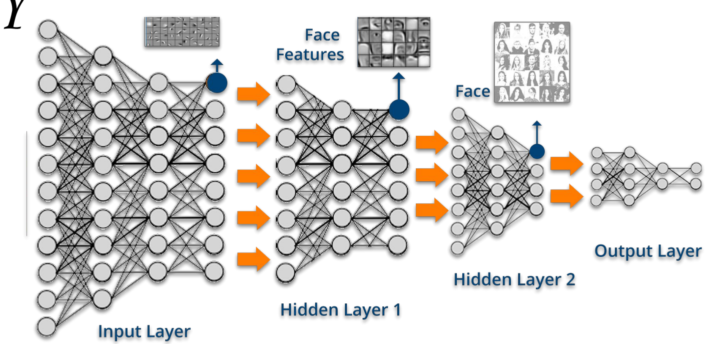
X_1 X_2 X_3 $\rightarrow$ Y

Typically, very complex!

X_1 X_2 X_3 X_4 X_5 X_6
X_7 X_8 X_9
X_{13} X_{14} X_{15} … X_m

f given $w_1, w_2, \ldots, w_p$ $\Rightarrow$ Y

(typically, $p >= m$)



Distributed TensorFlow

$$F(X|W) = \hat{Y}$$
$$Y = (X|W) + \varepsilon$$
$$Y = \hat{Y} + \varepsilon$$
$$\varepsilon = \hat{Y} - Y$$

Typical use-case:

Determine weights, $\mathcal{W}$, of a function, f, such that ε is minimized: f

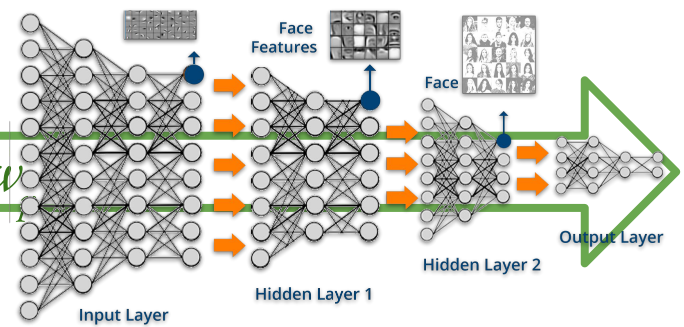
$\mathcal{W}$ determined through *gradient descent*:

back propagating error across the network that defines f.

$X_1^{(1)}$ X_2 X_3 X_4 X_5 X_6
X_7 X_8 X_9
X_{13} X_{14} X_{15} … X_m

f given $w_1, w_2, \ldots, w$

(typically, $p >= m$)



$Y^{(1)}$

Distributed TensorFlow

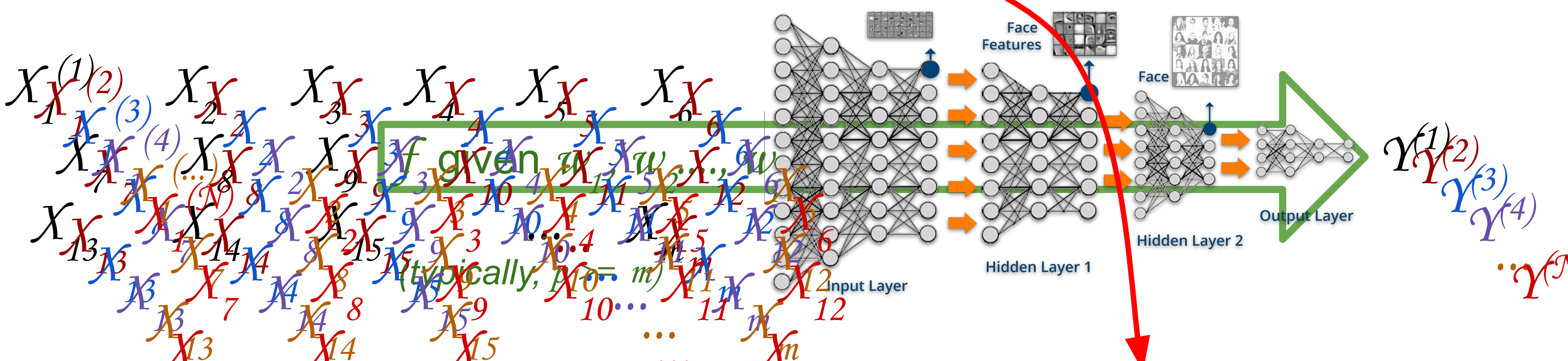
Typical use-case:

Determine weights, $\mathcal{W}$, of a function, f, such that ε is minimized: f

$$f(X|W) = \hat{Y}$$
$$Y = (X|W) + \varepsilon$$
$$Y = \hat{Y} + \varepsilon$$
$$\varepsilon = \hat{Y} - Y$$

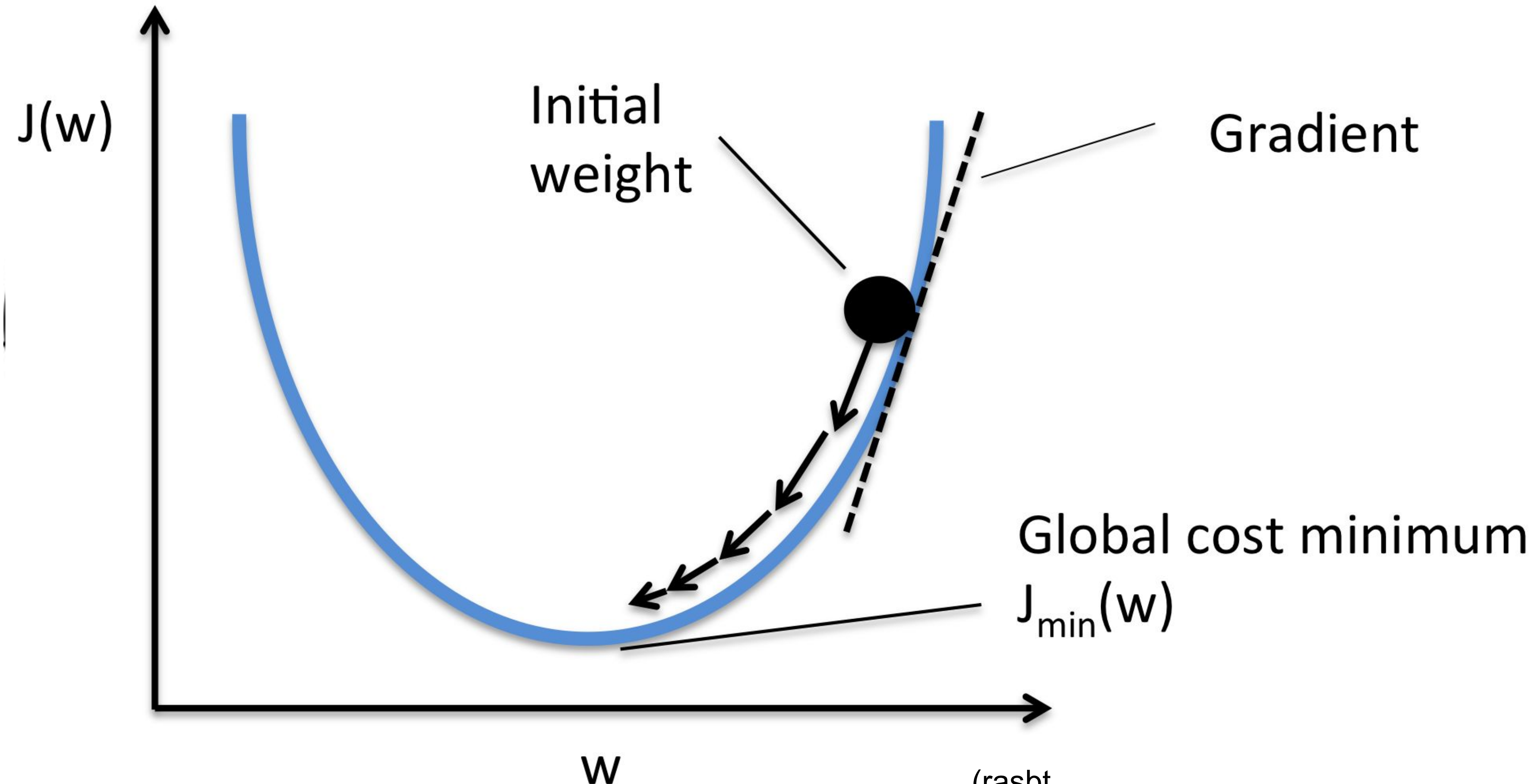
$\mathcal{W}$ determined through *gradient descent*:

back propagating error across the network that defines f.



minimizes ε on ***N*** training examples

Weights Derived from Gradients



(rasbt,

Weights Derived from Gradients

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

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

Weights 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:

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

Weights 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))

Weights Derived from Gradients

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

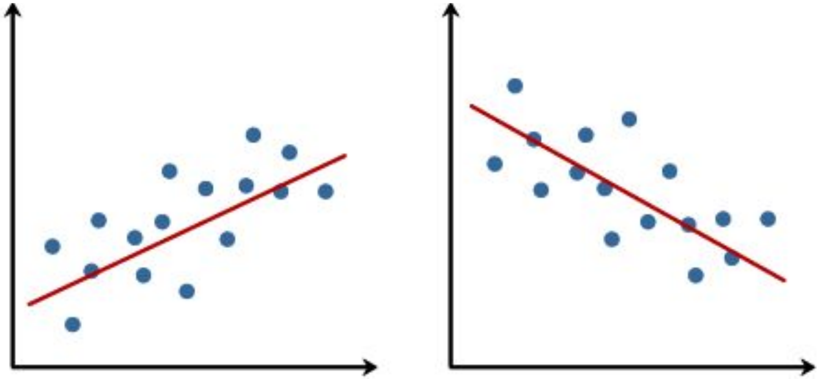
$$\hat{\beta} = argmin_{\beta}\{\sum_{i}^{N}(y_i - \hat{y}_i)^2\}$$

matrix multiply

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

Thus:

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



Weights 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? $\beta_{new} = \beta_{prev} - \alpha * \text{grad}$

Weights Derived from Gradients

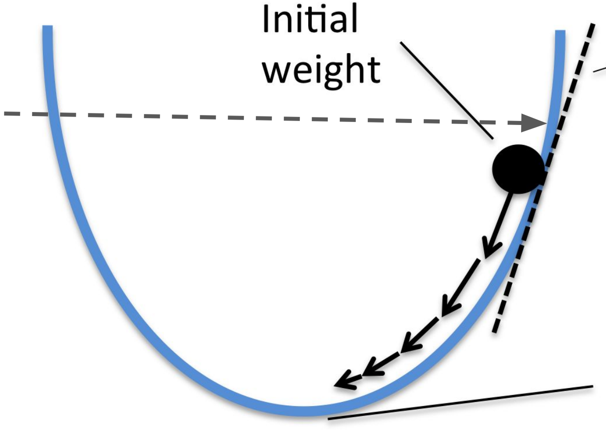
Linear Regression: Trying to find "betas" that minimize:

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

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



Weights 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:

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

Weights Derived from Gradients

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

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

2. Gradient descent solution

(Mirrors many parameter optimization problems.)

1. Matrix Solution:

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

Weights Derived from Gradients

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

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

<u>Gradient descent</u> needs to solve.
(Mirrors many parameter optimization problems.)

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

Weights Derived from Gradients

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

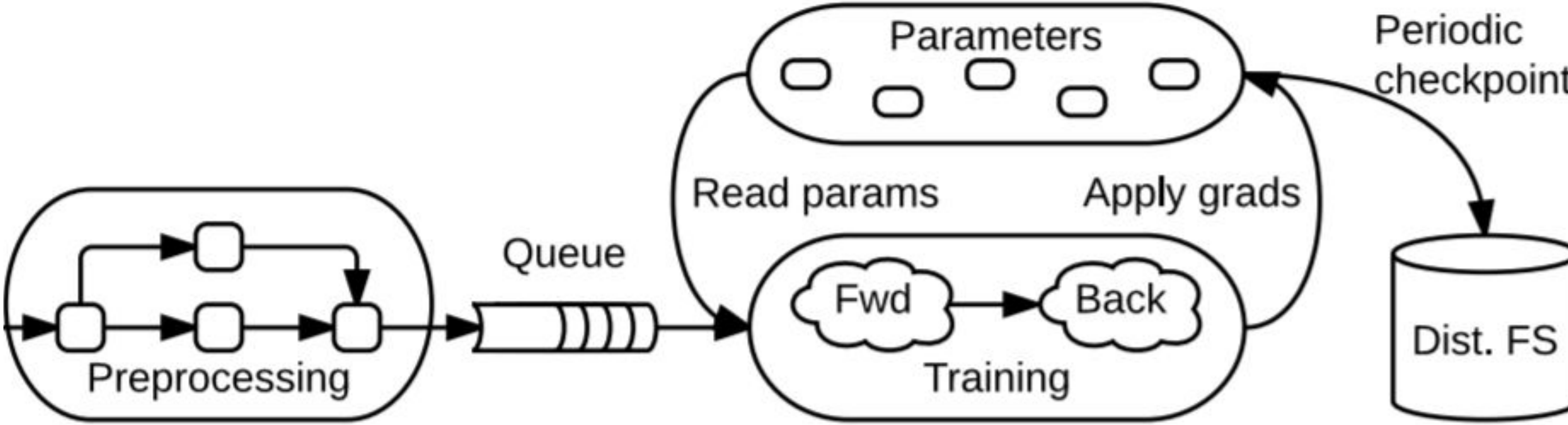
$$\hat{\beta}^{ridge} = argmin_{\beta}\{\boxed{\sum_{i=1}^{N}(y_i - \sum_{j=1}^{m} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{m} \beta_j^2\}}$$

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

TensorFlow has built-in ability to derive gradients given a $\boxed{\text{cost function}}$.

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

Weights Derived from Gradients



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

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

Options for distribution

1. **Distribute copies of entire dataset**
   a. Train over all with different hyperparameters
   b. Train different folds per worker node.

Options for distribution

1. **Distribute copies of entire dataset**
    a. Train over all with different parameters
    b. Train different folds per worker node.

2. **Distribute data**
    a. Each node finds parameters for subset of data
    b. Needs mechanism for updating parameters
        i. Centralized parameter server
        ii. Distributed All-Reduce

Options for distribution

1. **Distribute copies of entire dataset**
   a. Train over all with different parameters
   b. Train different folds per worker node.

2. **Distribute data**
   a. Each node finds parameters for subset of data
   b. Needs mechanism for updating parameters
      i. Centralized parameter server
      ii. Distributed All-Reduce

3. **Distribute model or individual operations** (e.g. matrix multiply)

Options for distribution

1. **Distribute copies of entire dataset**
   a. Train over all with different parameters
   b. Train different folds per worker node.

   Pro: Easy; Good for compute-bound; Con: Requires data fit in worker memories

2. **Distribute data**
   a. Each node finds parameters for subset of data
   b. Needs mechanism for updating parameters
      i. Centralized parameter server
      ii. Distributed All-Reduce

   Pro: Flexible to all situations; Con: Optimizing for subset is suboptimal

3. **Distribute model or individual operations** (e.g. matrix multiply)

   Pro: Parameters can be localized Con: High communication for transferring Intermediar data.

Options for distribution

Done often in practice. Not talked about much because it's mostly as easy as it sounds.

1. **Distribute copies of entire dataset**
   a. Train over all with different parameters
   b. Train different folds per worker node.

   Pro: Easy; Good for compute-bound; Con: Requires data fit in worker memories

2. **Distribute data**
   a. Each node finds parameters for subset of data
   b. Needs mechanism for updating parameters
      i. Centralized parameter server
      ii. Distributed All-Reduce

   Pro: Flexible to all situations; Con: Optimizing for subset is suboptimal

3. **Distribute model or individual operations** (e.g. matrix multiply)

   Pro: Parameters can be localized Con: High communication for transferring Intermediar data.

Options for distribution

Done often in practice. Not talked about much because it's mostly as easy as it sounds.

1. **Distribute copies of entire dataset**
   a. Train over all with different parameters
   b. Train different folds per worker node.

   Pro: Easy; Good for compute-bound; Con: Requires data fit in worker memories

2. **Distribute data**
   a. Each node finds parameters for subset of data
   b. Needs mechanism for updating parameters
      i. Centralized parameter server
      ii. Distributed All-Reduce

   Preferred method for big data or very complex models (i.e. models with many internal parameters).

   Pro: Flexible to all situations; Con: Optimizing for subset is suboptimal

3. **Distribute model or individual operations** (e.g. matrix multiply)

   Pro: Parameters can be localized Con: High communication for transferring Intermediar data.

Options for distribution

Done often in practice. Not talked about much because it's mostly as easy as it sounds.

1. **Distribute copies of entire dataset**
    a. Train over all with different parameters
    b. Train different folds per worker node.

    Pro: Easy; Good for compute-bound; Con: Requires data fit in worker memories

2. **Distribute data**
    a. Each node finds parameters for subset of data
    b. Needs mechanism for updating parameters
        i. Centralized parameter server
        ii. Distributed All-Reduce

    Data Parellelism

    Pro: Flexible to all situations; Con: Optimizing for subset is suboptimal

Preferred method for big data or very complex models (i.e. models with many internal parameters).

3. **Distribute model or individual operations** (e.g. matrix multiply)

    Model Parellelism

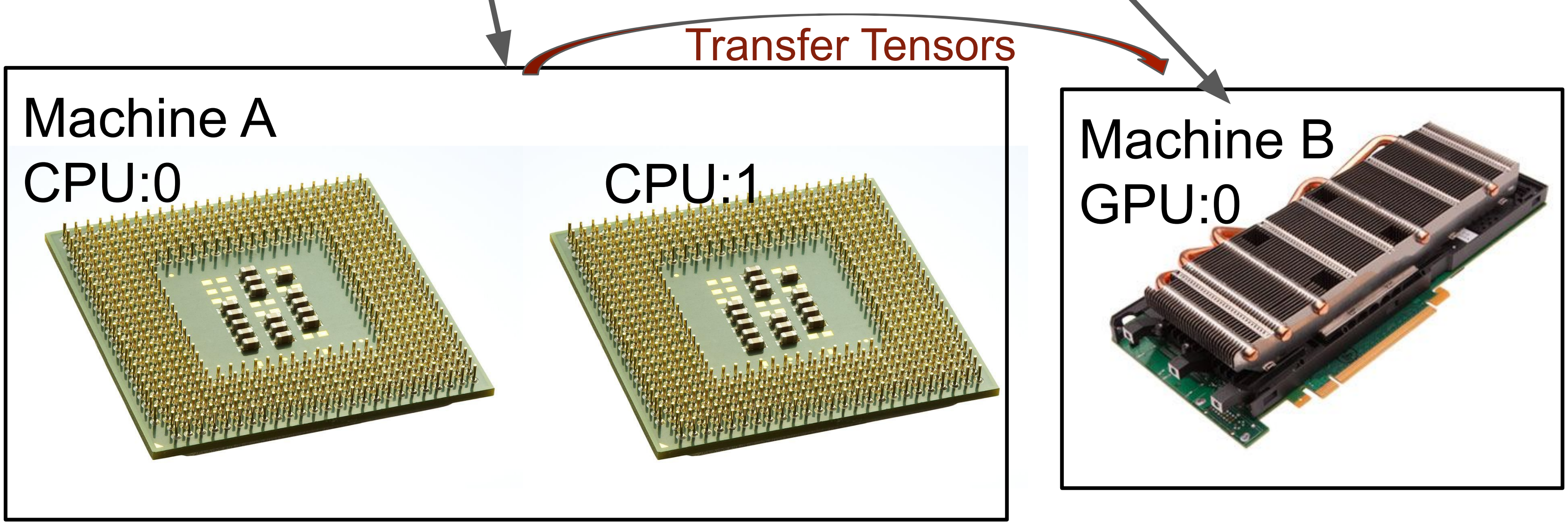
    Pro: Parameters can be localized Con: High communication for transferring Intermediar data.

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)
```



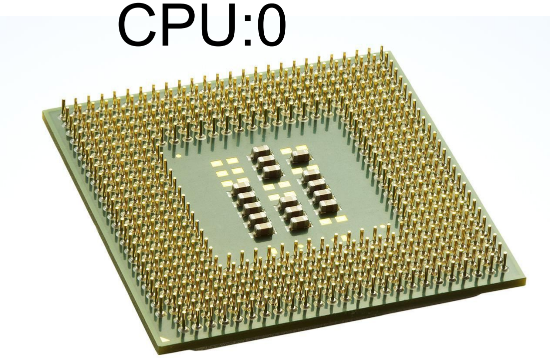
<u>Data</u> Parallelism

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

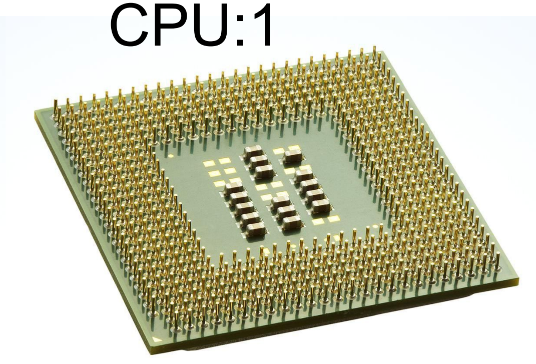
```
...
  beta=tf.Variable(...)
  pred=tf.matmul(beta,X)
```

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

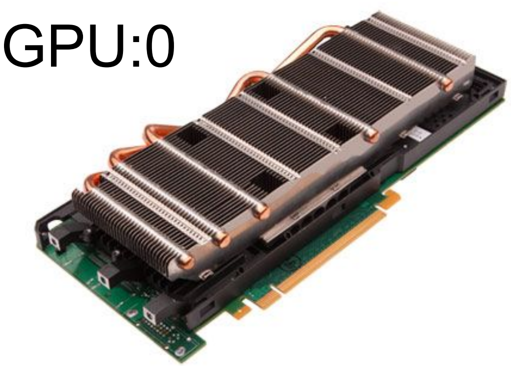
CPU:0



CPU:1



GPU:0



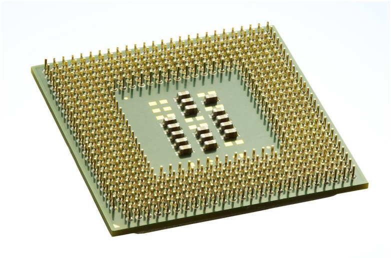
Data Parallelism

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

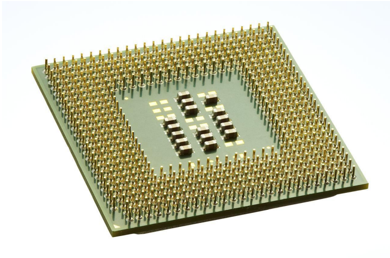
```
...
  beta=tf.Variable(...)
  pred=tf.matmul(beta,X)
```

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

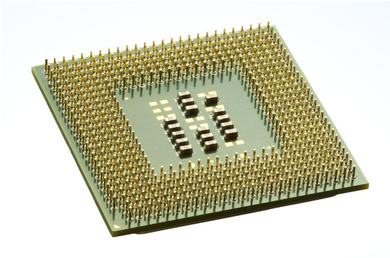
worker:0



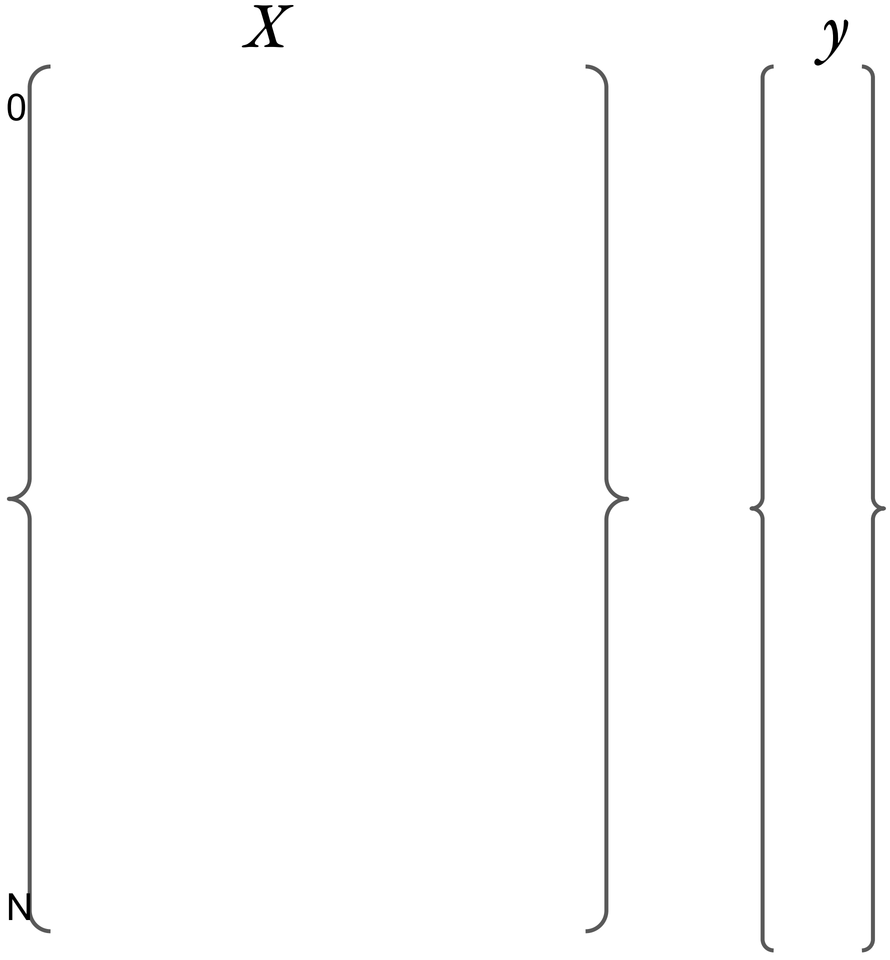
worker:1



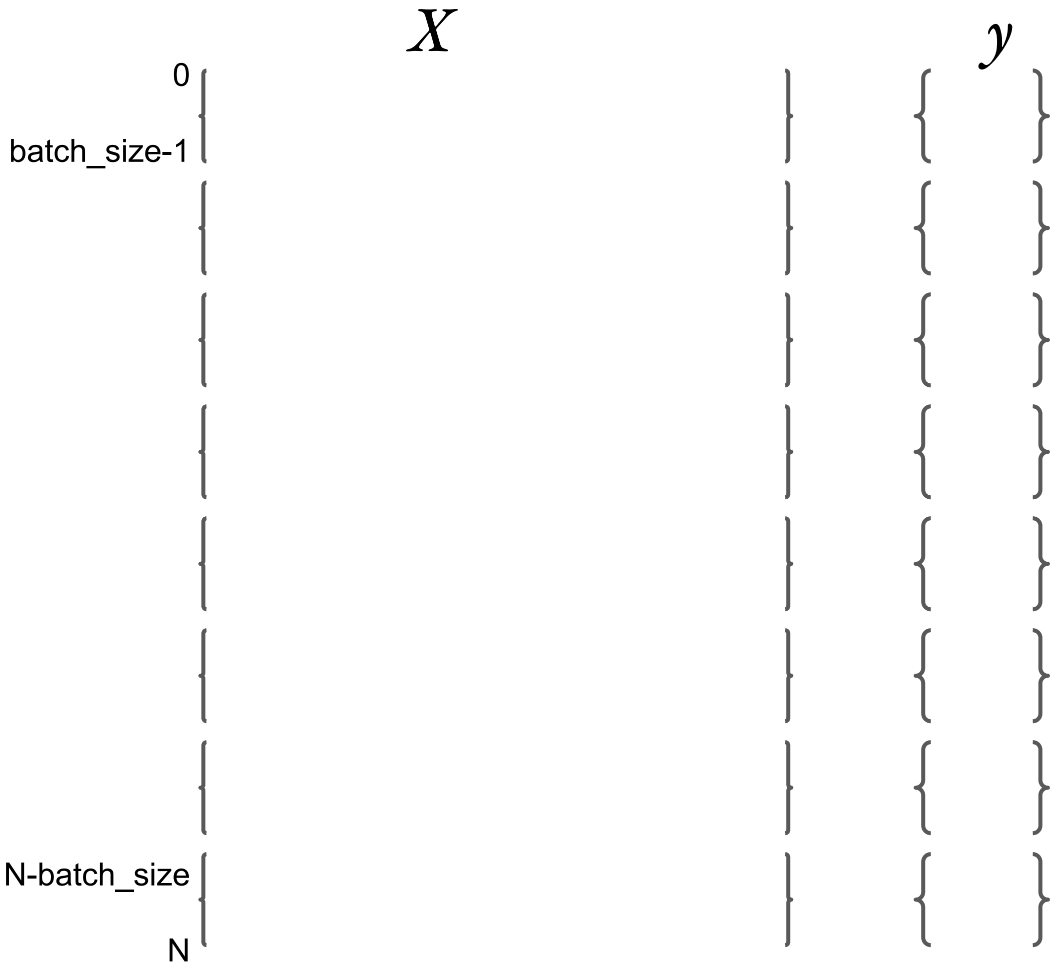
worker:2



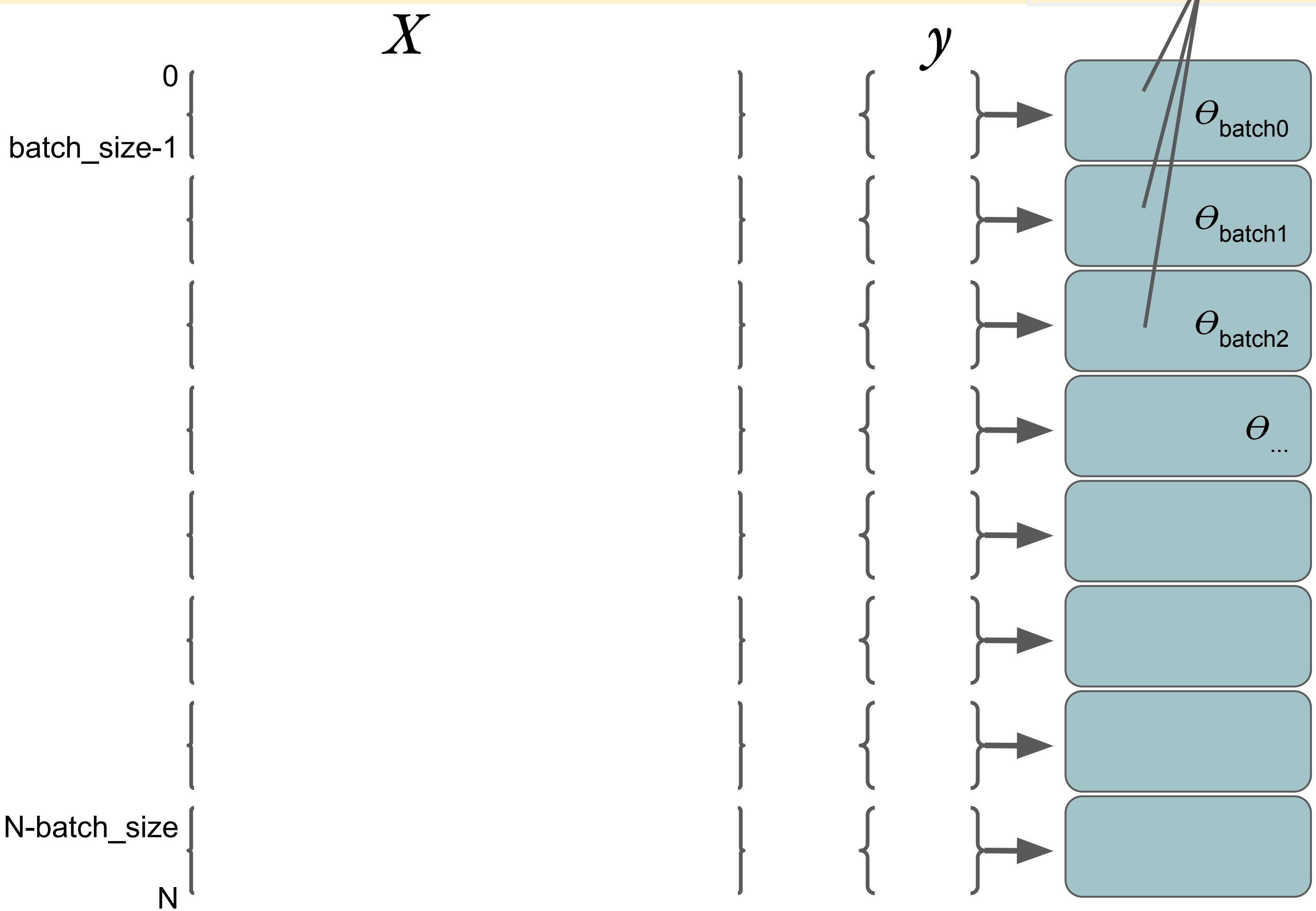
Distributing Data



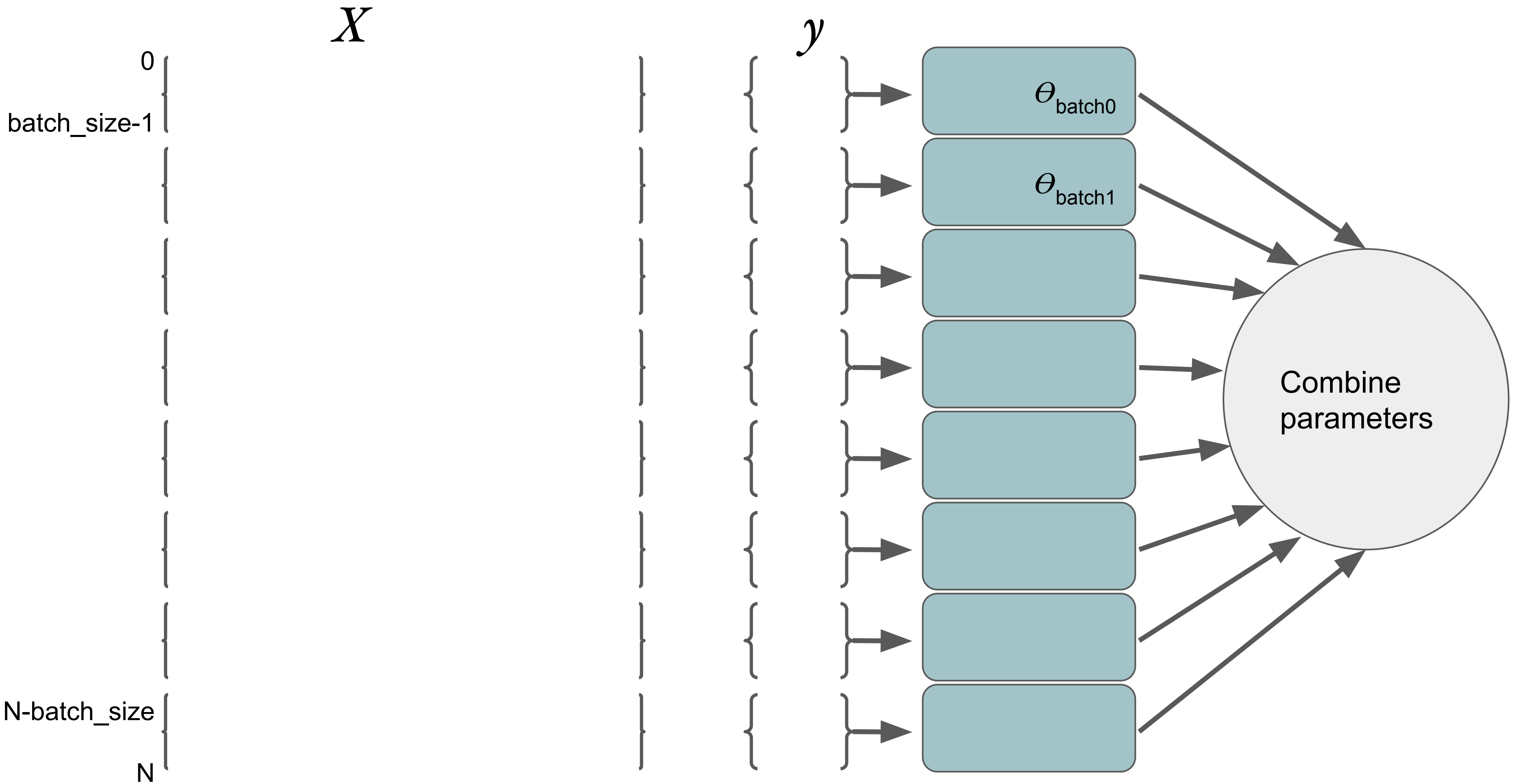
Distributing Data



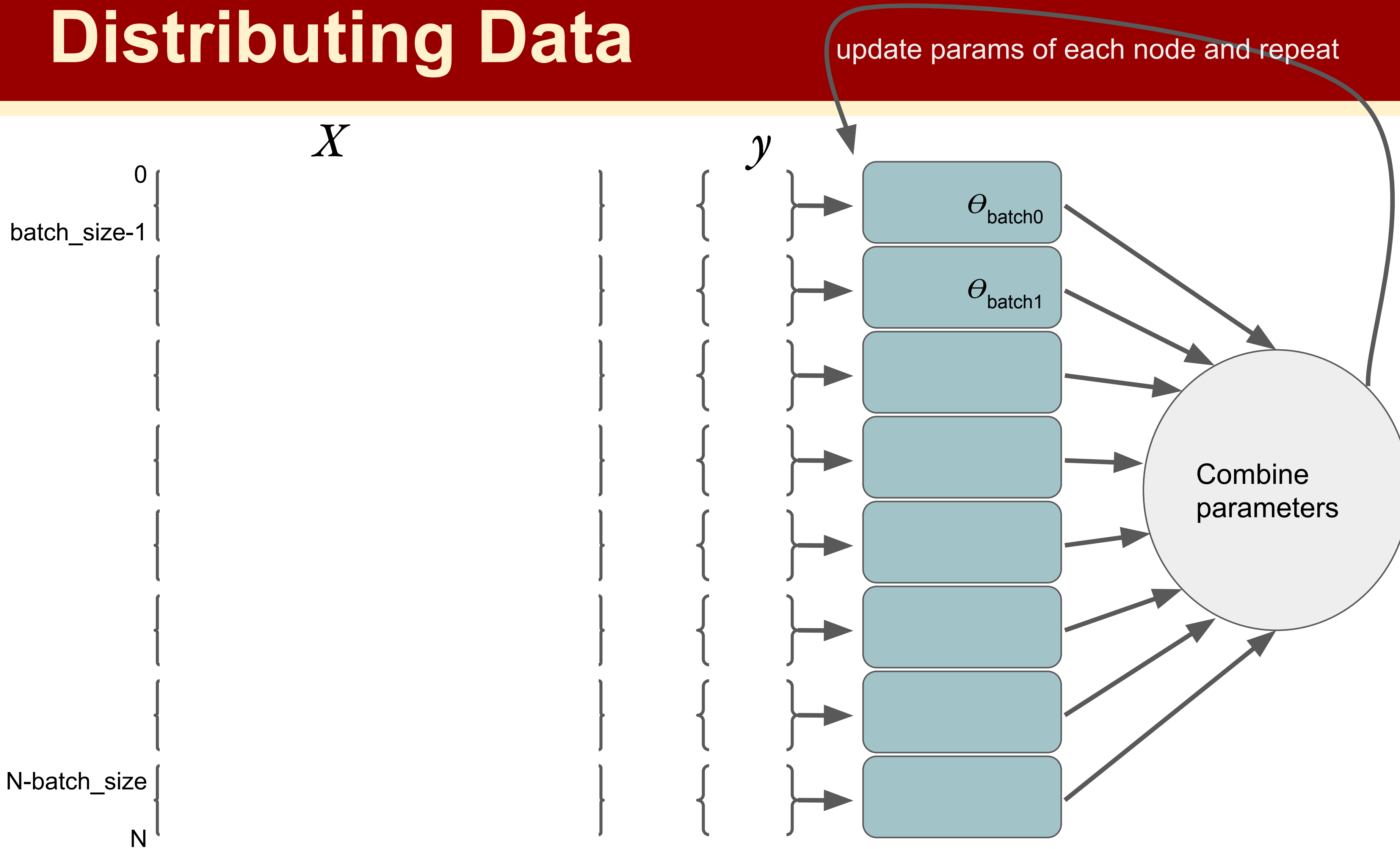
Distributing Data



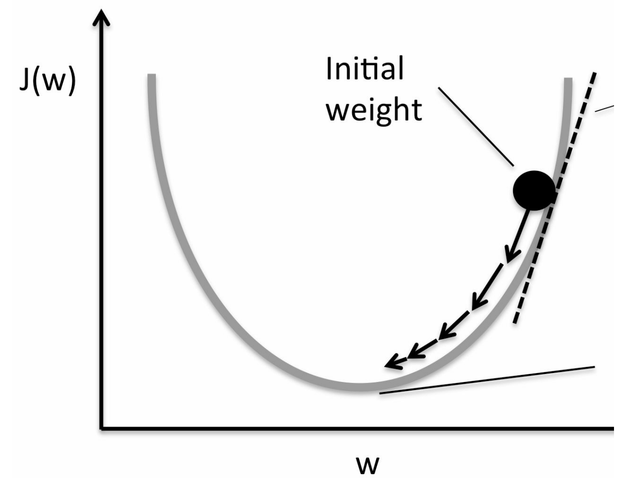
Distributing Data



Distributing Data



Gradient Descent for Linear Regression



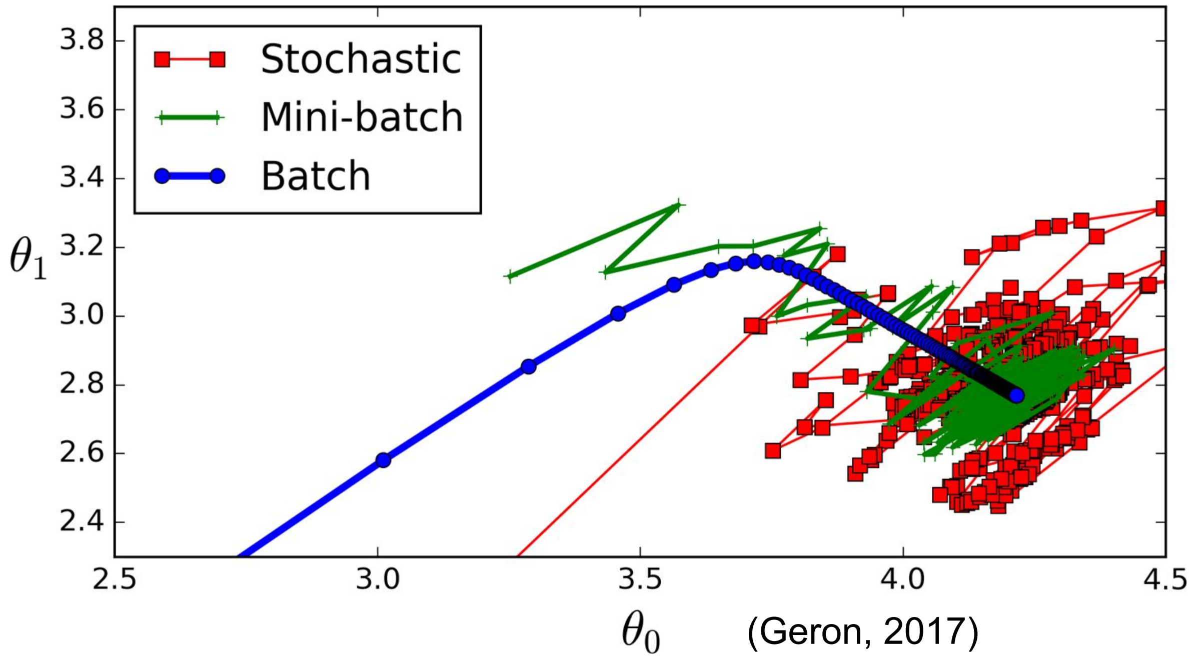
(Geron, 2017)

Gradient Descent for Linear Regression

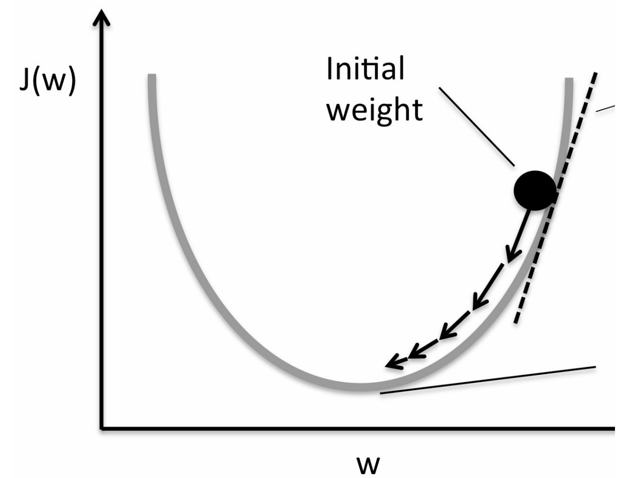
Batch Gradient Descent

Stochastic Gradient Descent: One example at a time

Mini-batch Gradient Descent: k examples at a time.



(Geron, 2017)

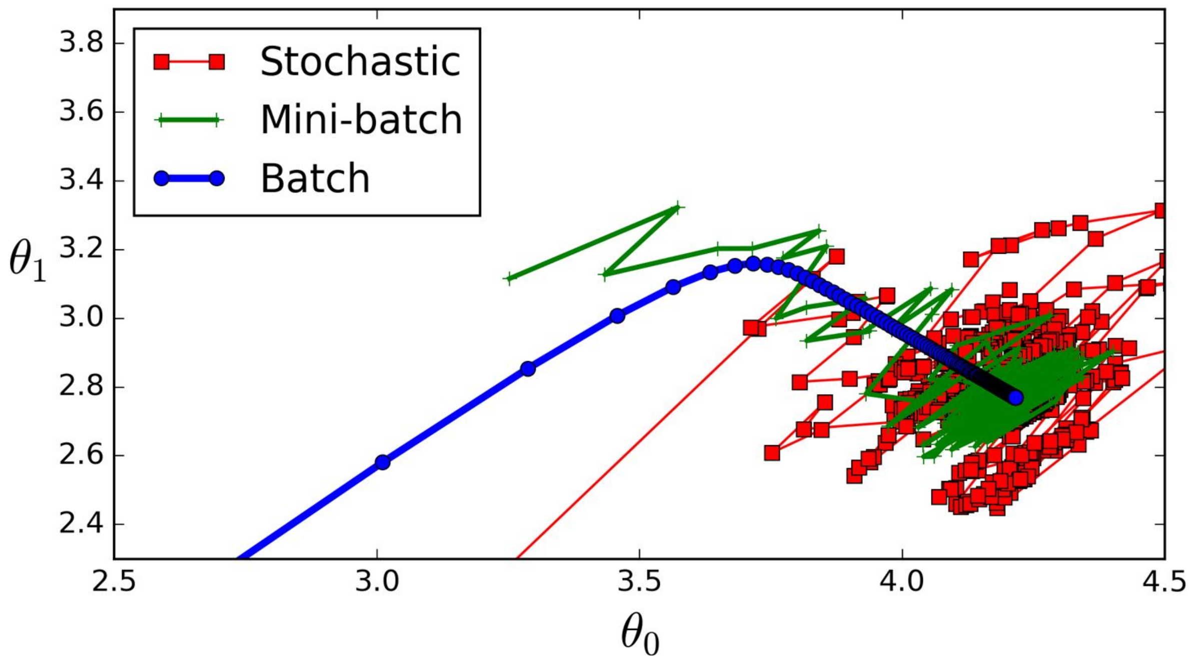


Gradient Descent for Linear Regression

Batch Gradient Descent

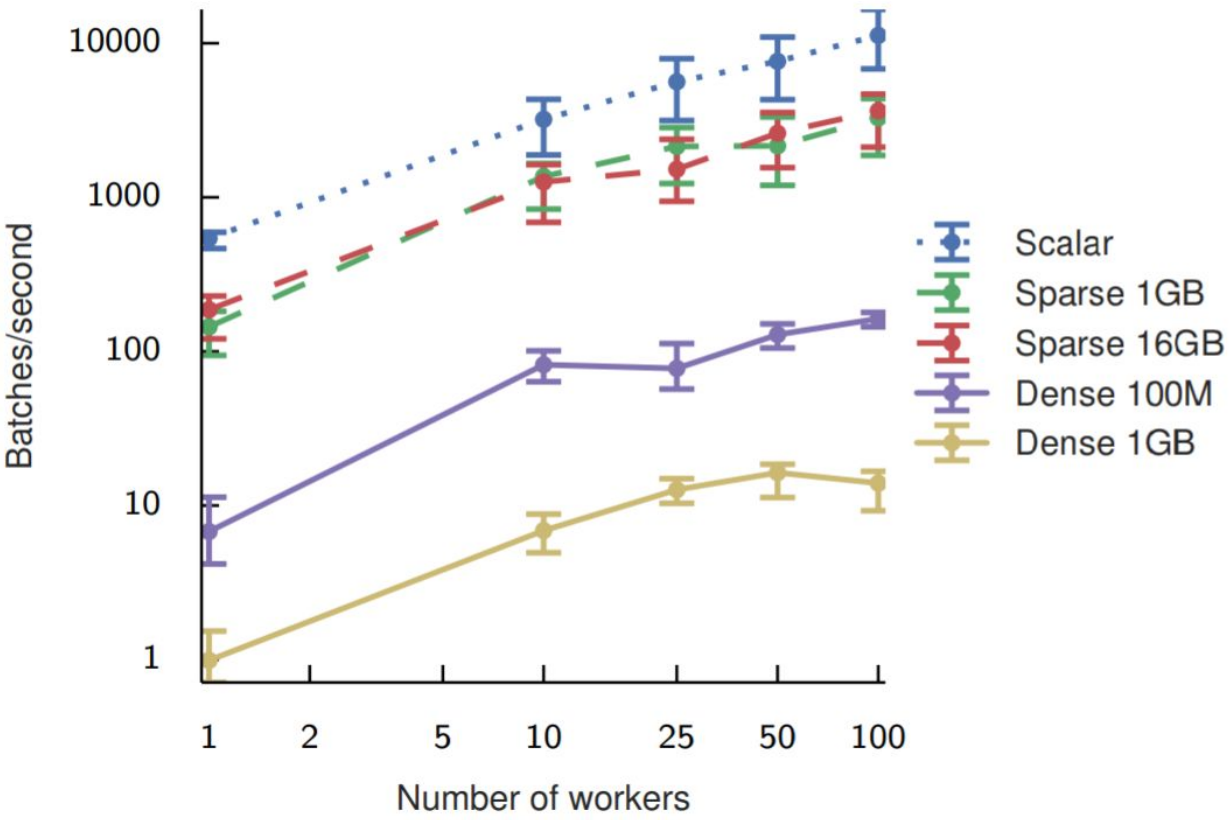
Stochastic Gradient Descent: One example at a time

Mini-batch Gradient Descent: k examples at a time.



(Geron, 2017)

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

discussed previously

Parallelisms:

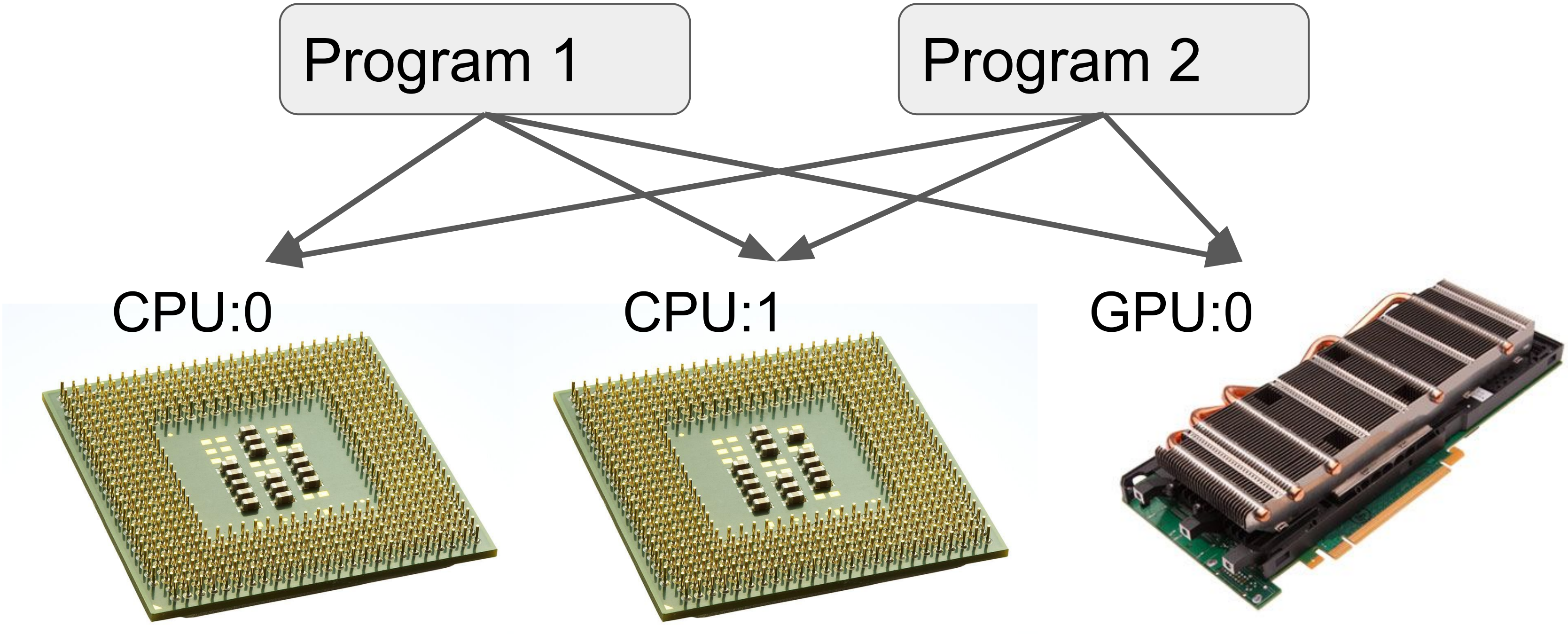
- Data Parallelism: All nodes doing same thing on different subsets of data
- Graph/Model Parallelism: Different portions of model on different devices

Model Updates:

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

Local Distribution

Multiple devices on single machine



Local Distribution

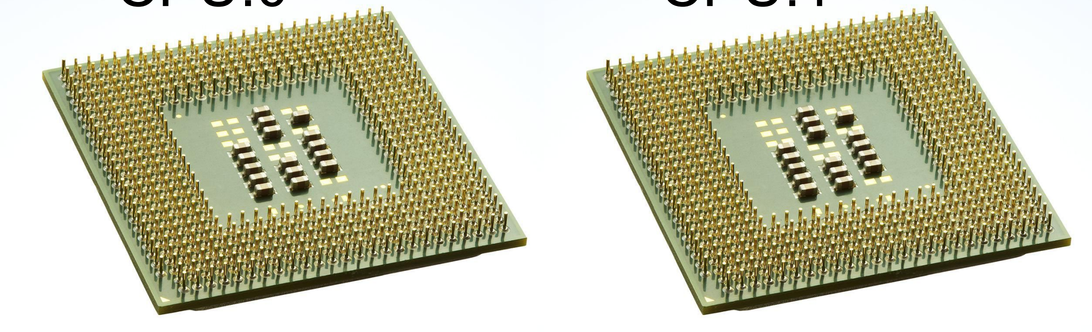
Multiple devices on single machine

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

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

CPU:0

CPU:1



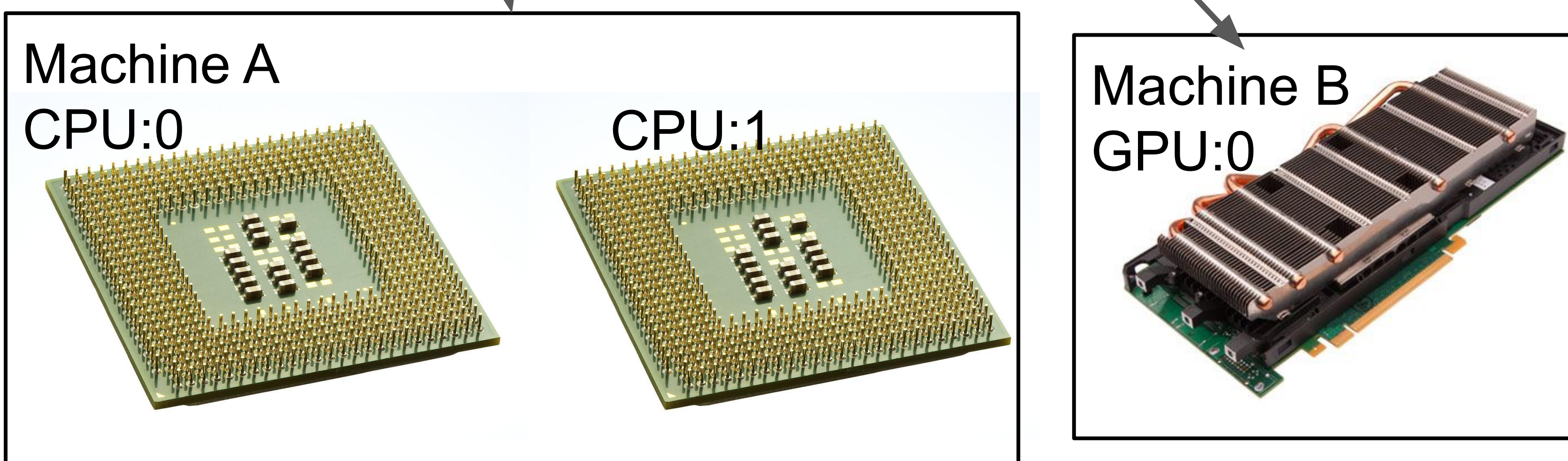
GPU:0

Local 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)

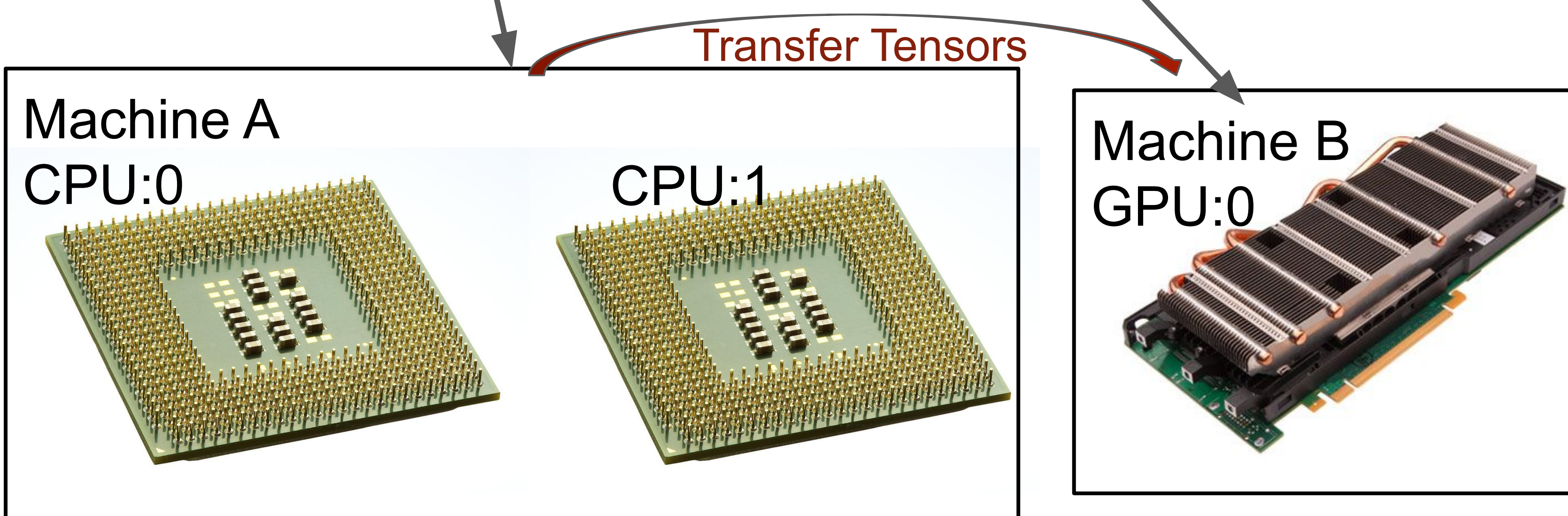
Parallelisms

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)
```



Parallelisms

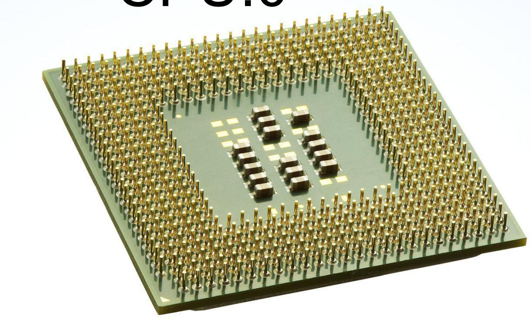
Data Parallelism

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

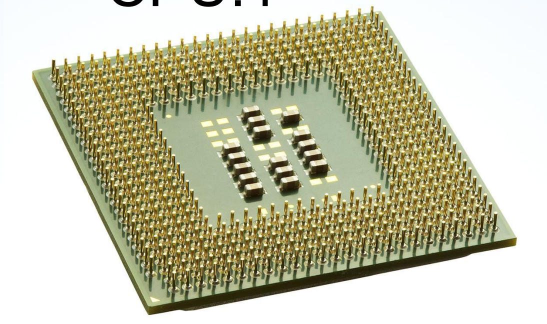
```
...
  beta=tf.Variable(...)
  pred=tf.matmul(beta,X)
```

```
...
  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

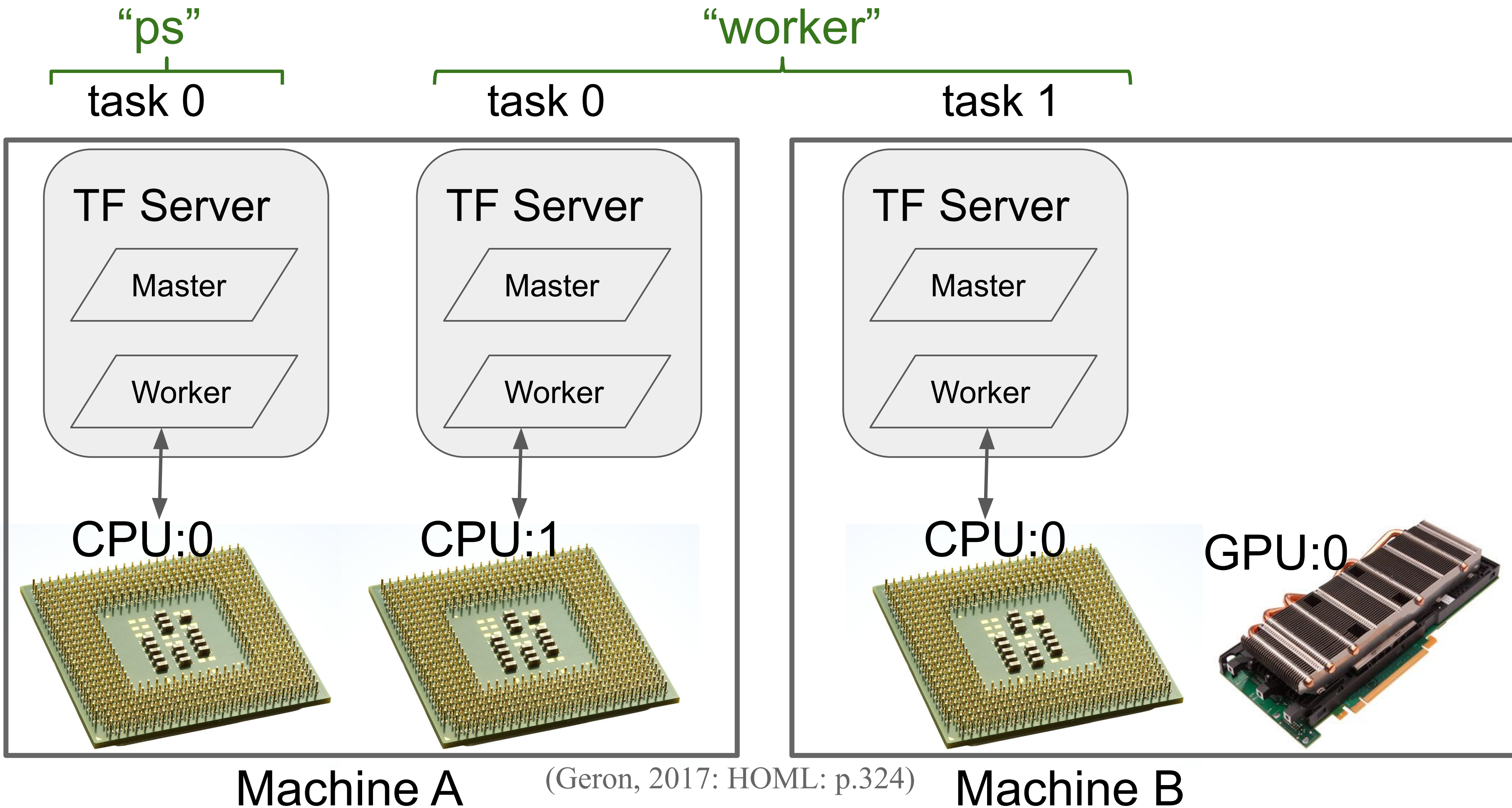
Parallelisms:

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

Model Updates:

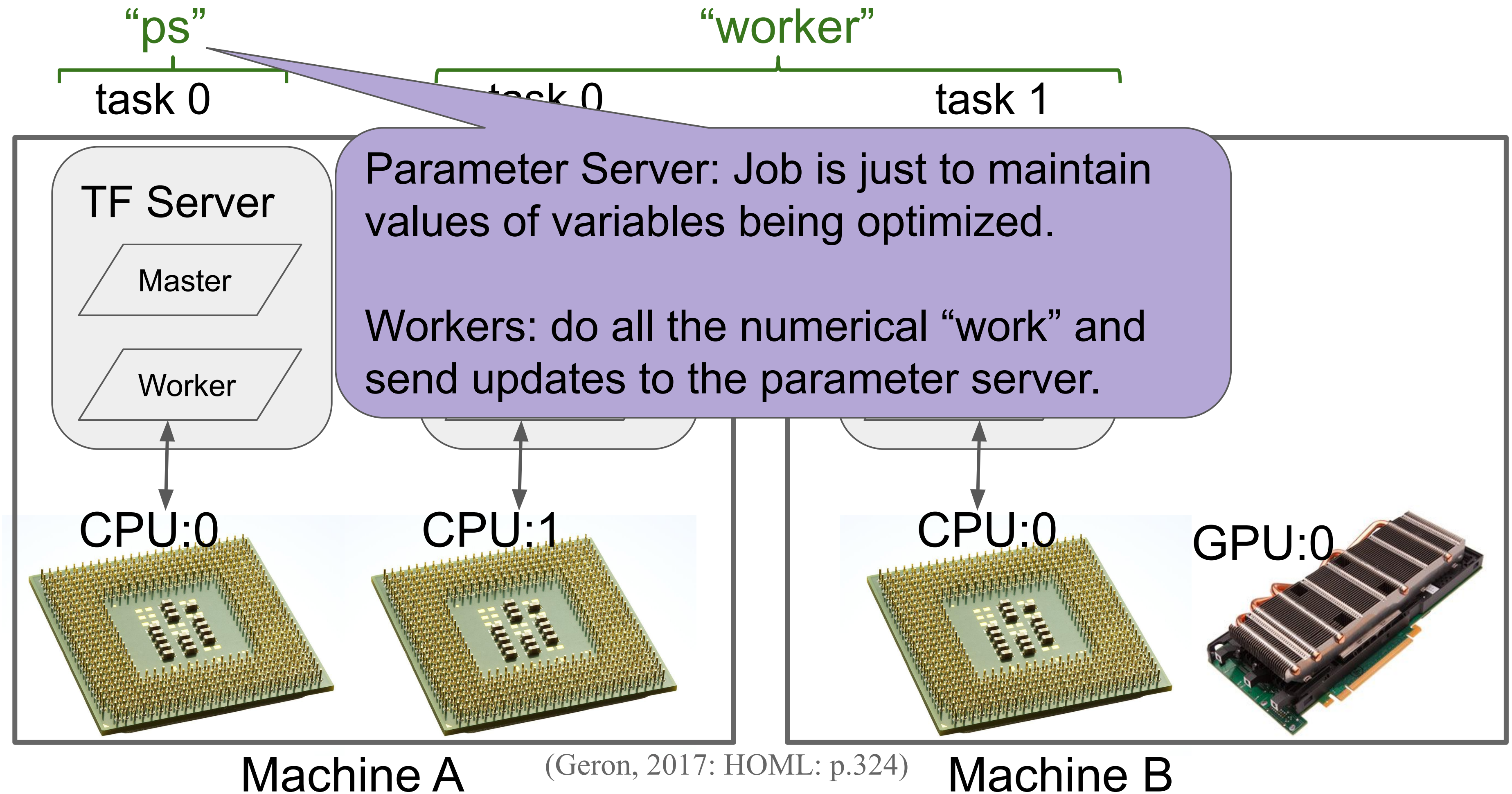
- Asynchronous Parameter Server
- Synchronous AllReduce (doesn't work with Model Parallelism)

Asynchronous Parameter Server



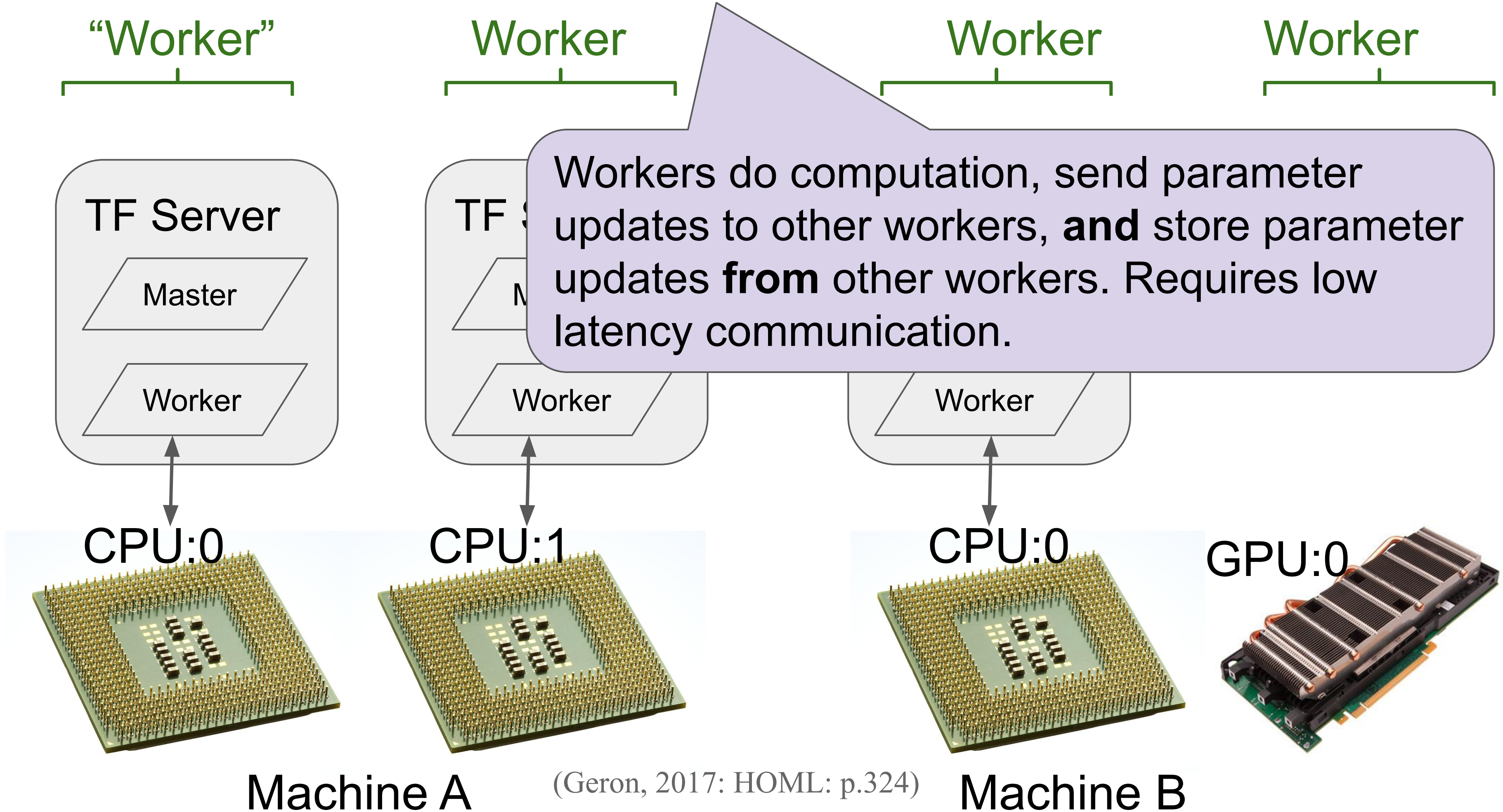
(Geron, 2017: HOML: p.324)

Asynchronous Parameter Server



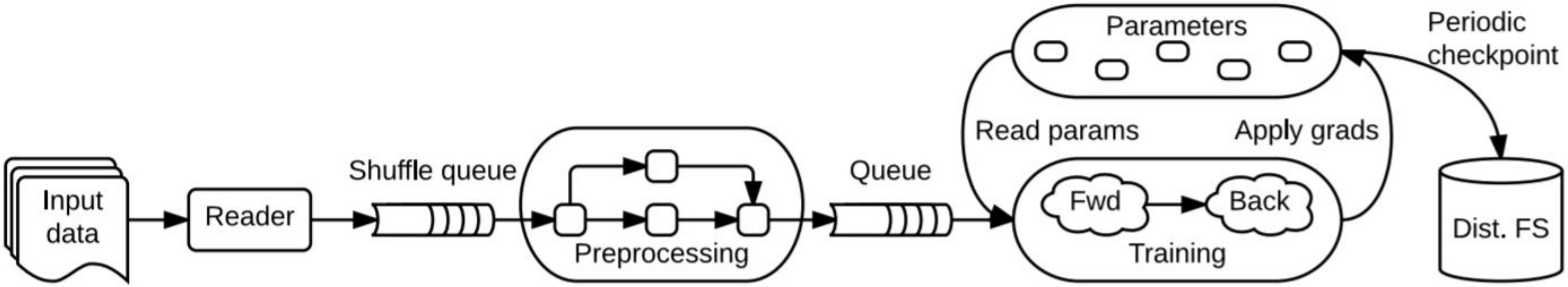
(Geron, 2017: HOML: p.324)

Synchronous All Reduce



(Geron, 2017: HOML: p.324)

Distributed TF: 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).

Summary

- TF is a workflow system, where records are always tensors
  - *operations* applied to tensors (as either Variables, constants, or placeholder)
- Optimized for numerical / linear algebra
  - automatically finds gradients
  - custom kernels for given devices
- “Easily” distributes
  - Within a single machine (local: many devices))
  - Across a cluster (many machines and devices)
  - Jobs broken up as parameter servers / workers makes coordination of data efficient