Deep Learning With TensorFlow

Yonghui Wu

Agenda

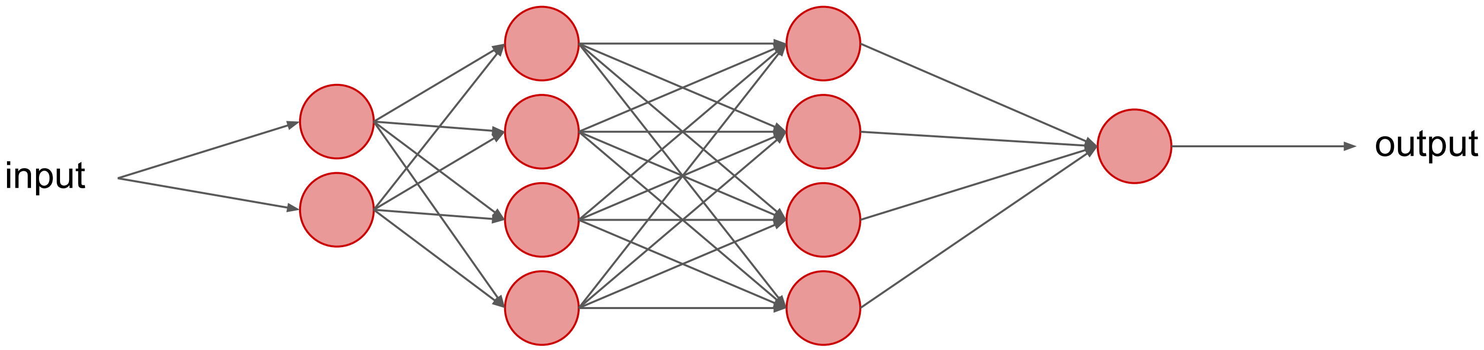
- Intro to deep learning
- TensorFlow from 10000 feet up
- TensorFlow key concepts
- Concluding remarks

Intro to deep learning

Deep learning

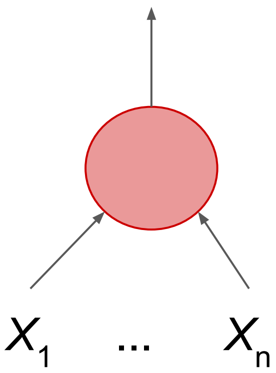
Deep learning is a branch of artificial intelligence that is based on neural networks

- loosely inspired by the brain
- built from simple, *trainable* functions.



Primitives: the neuron

$y = F(w_1 x_1 + ... + w_n x_n + b)$

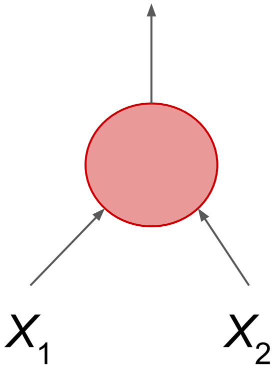


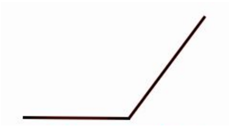
inputs

- $w_1 ... w_n$ are weights,
- b is a bias,
- weights and biases are parameters,
- F is a “differentiable” non-linear function.

Primitives: the neuron — example

$$y = \max(0, x_1 - 1.2x_2 + 0.1)$$





$n = 2$

$w_1 = 1 \quad w_2 = -1.2$

$b = 0.1$

$F(x) = \max(0, x)$ "Relu"

A modern network [Szegedy et al., 2016]

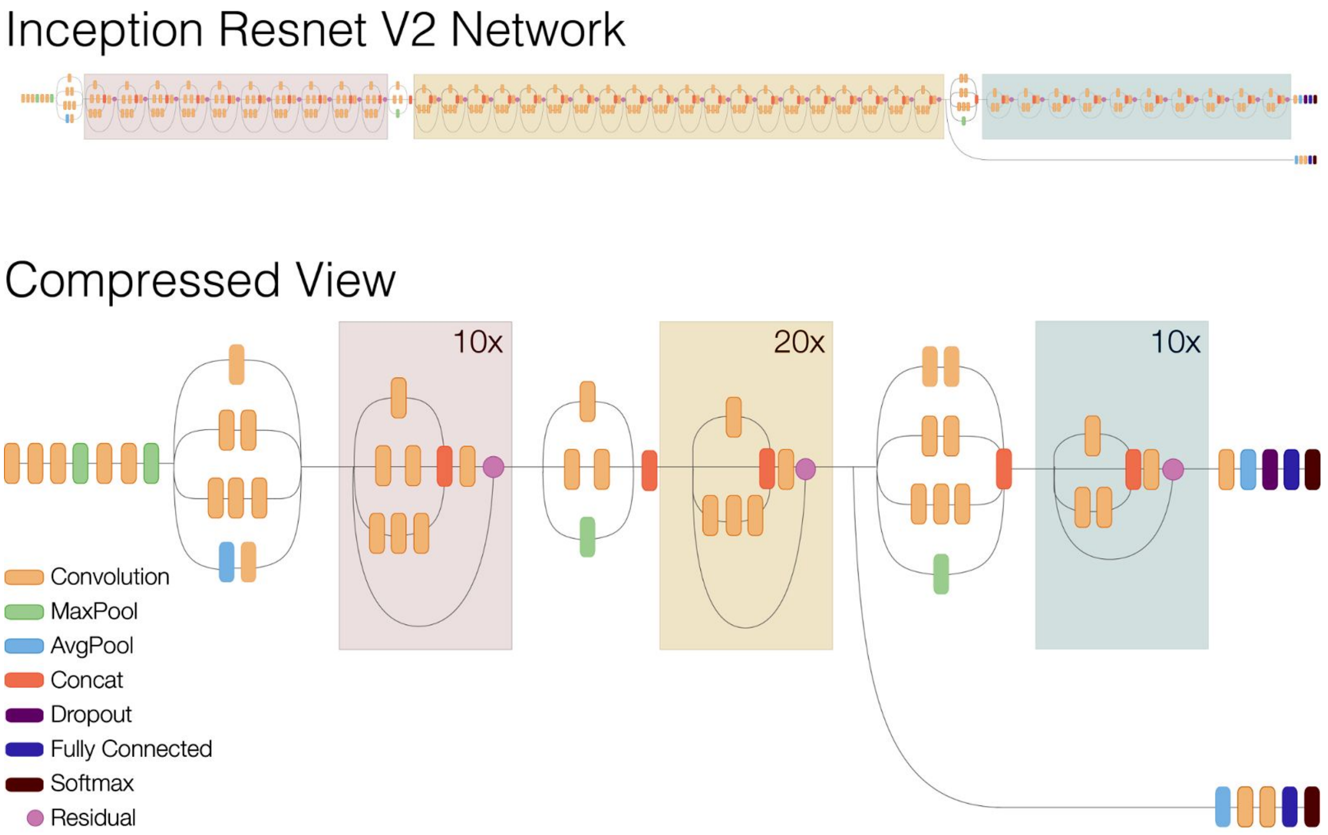


Image:
Fernanda Viégas

A learning algorithm

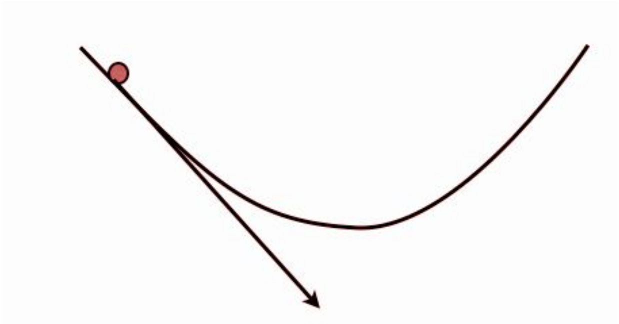
Given training examples "(input, output)" pairs

While not done:

1. Pick a batch of random training example (*X*, *Y*)
2. Run the neural network on *X, generate Y'*
3. Compute a loss by comparing Y' to Y
4. Adjust model parameters to reduce the error (the "loss")

Gradient descent

- Compute gradient of the loss with respects to params in the neural network.
- new_params = old_params - alpha * gradient
- alpha is the learning rate



Deep learning is conceptually very simple

- That is it, that is all the key elements of deep learning
- Typical procedure to solve a problem with deep learning
    - Define a network suitable to the problem one would like to solve.
    - Define a loss function.
    - Initialize the network with small random number
    - Iteratively adjust network params following some optimization algorithm until loss doesn’t decrease any more

The renaissance of deep learning

- The concept and main algorithm were actually invented 1960's ~ 1980's
- It gained popularity only very recently
  - AlexNet won ImageNet 2012 competition, beating runner up by as much as 10% in top-5 accuracy
- People attributed the renaissance of NN to:
  - Large and Larger datasets
  - Advance in computing power

Some attractive properties of deep learning

- Applicable across many domains.
- With a fairly simple conceptual core.
  - Back propagation
  - SGD
  - Neuron
- Benefits from having lots of data.
  - Often requires little data curation.
  - Tolerates inconsistent data.

More attractive properties of deep learning

- Model gets better with more data, more compute
  - Data and compute and sometimes easy to get
- Requires architectural choices but no detailed design of algorithms and data representations.
  - Can learn intriguing and powerful data representation

Deep Learning

Universal Machine Learning

...that works better than the alternatives!

Current State-of-the-art in:

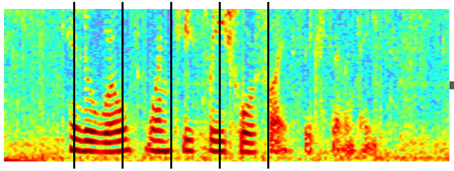
Speech Recognition
Image Recognition
Machine Translation
Molecular Activity Prediction
Road Hazard Detection
Optical Character Recognition
...

Rapid Progress in Image Recognition

Team	Year	Place	Error (top-5)	Params
XRCE (pre-neural-net explosion)	2011	1st	25.8%	
Supervision (AlexNet)	2012	1st	16.4%	60M
Clarifai	2013	1st	11.7%	65M
MSRA	2014	3rd	7.35%	
VGG	2014	2nd	7.32%	180M
GoogLeNet (Inception)	2014	1st	6.66%	5M
Andrej Karpathy (human)	2014	N/A	5.1%	100 trillion?
BN-Inception (Arxiv)	2015	N/A	4.9%	13M
Inception-v3 (Arxiv)	2015	N/A	3.46%	25M

ImageNet classification challenge

Speech Recognition



Acoustic Input

Deep
Recurrent
Neural Network

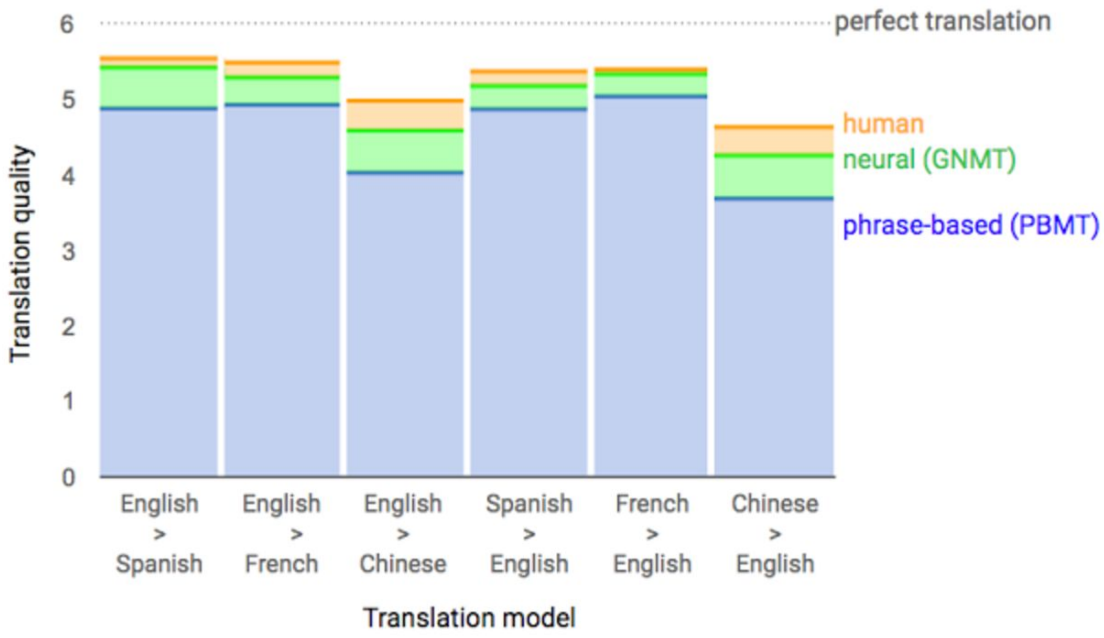
"How cold is it outside?"

Text Output

Neural nets are rapidly replacing previous technologies

Google Research Blog - August 2012, August 2015

Machine Translation



GNMT significantly reduced the gap between in translation quality between MT and human

AlphaGo

AlphaGo dominated the Game of Go

TensorFlow from 10000 feet up

https://tensorflow.org/

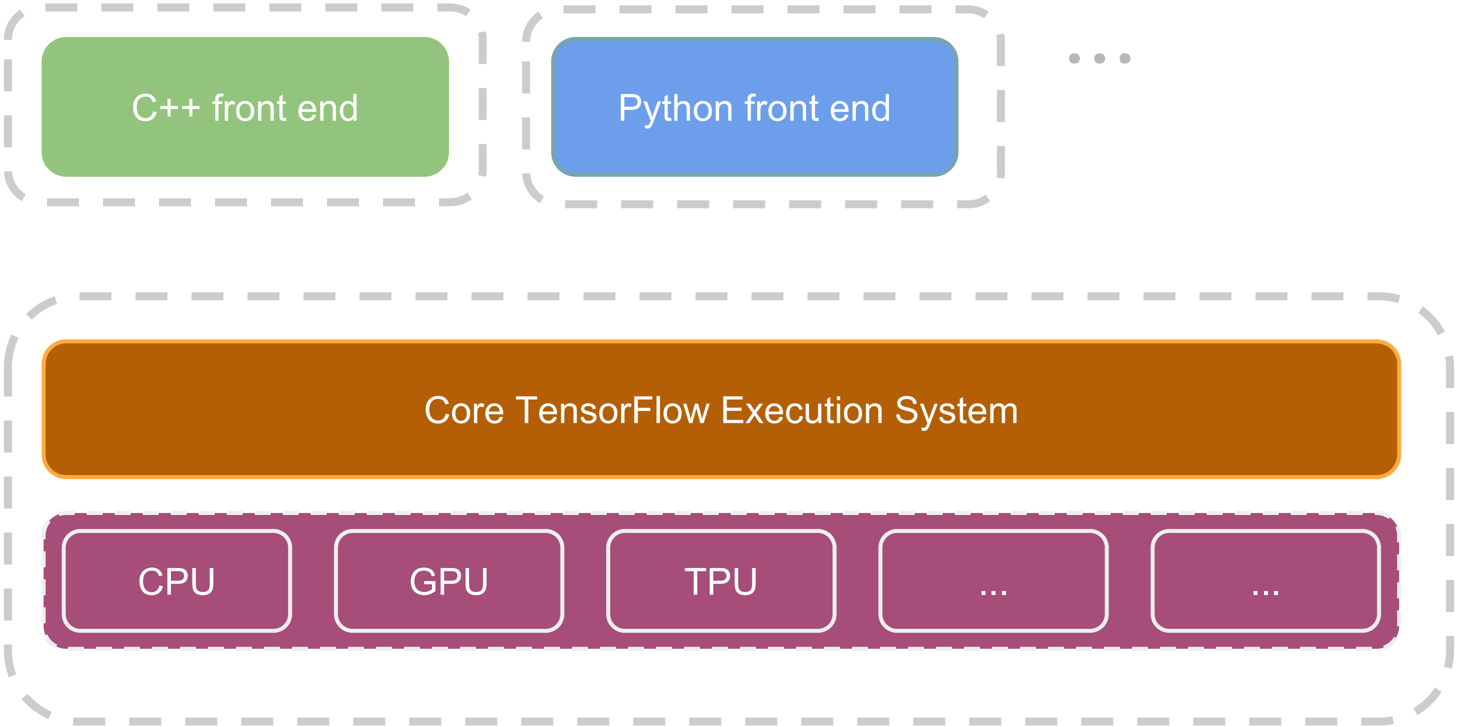
and

https://github.com/tensorflow/tensorflow

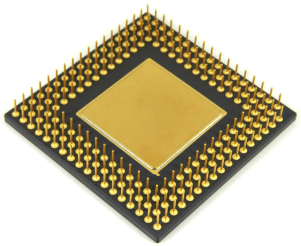
Software for machine learning and particularly for deep learning.

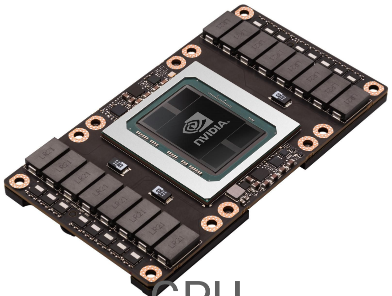
First open-source release in November 2015, under an Apache 2.0 license.

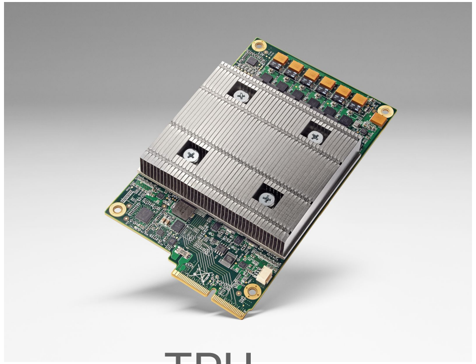
System structure



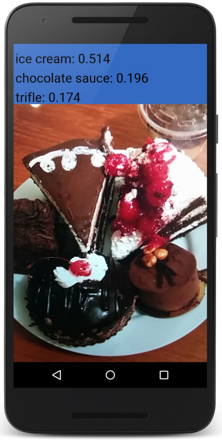
TensorFlow supports many platforms


CPU

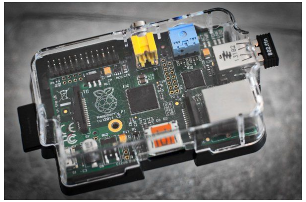

GPU


TPU

iOS



Android


Raspberry Pi

… and many languages

python™

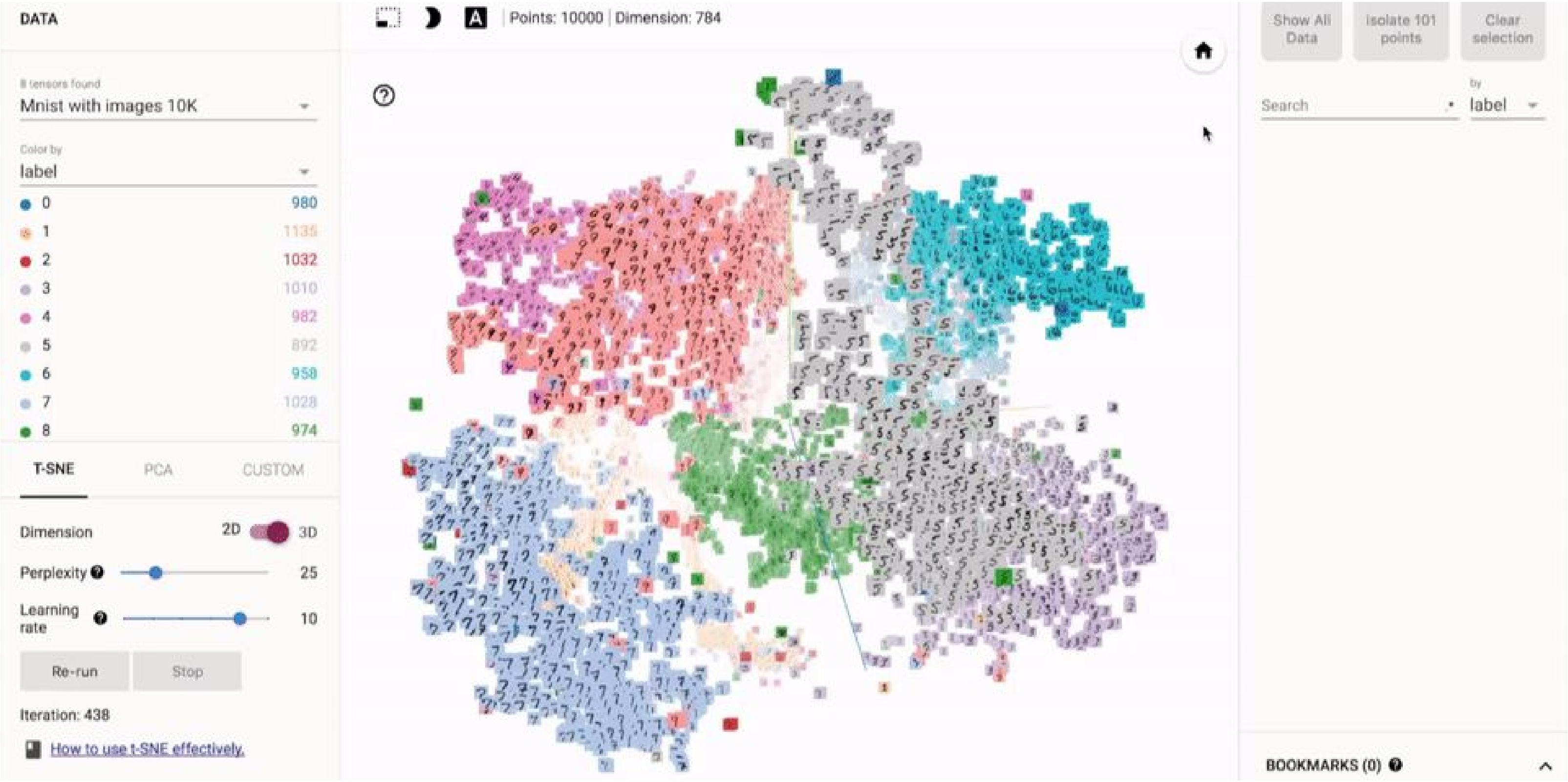
C++

Java

Go

Haskell

TensorFlow provides great tools like TensorBoard



v0.5
Initial Release
Nov ‘15

Dec ‘15
v0.6
Faster on GPUs;
Python 3.3+

v0.7
TensorFlow Serving
Feb ‘16

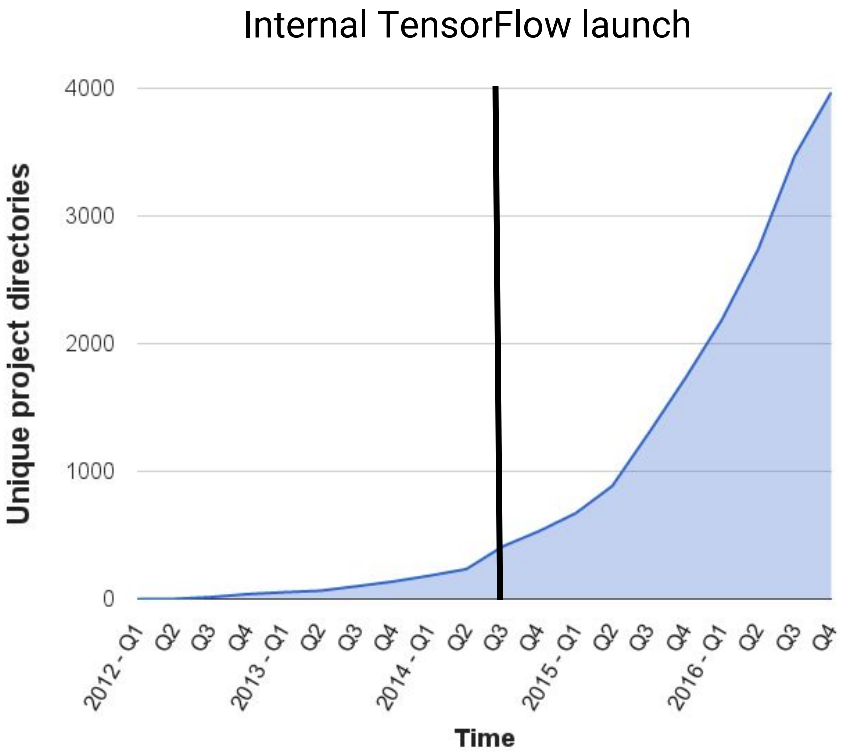
Apr ‘16
v0.8
Distributed
TensorFlow

v0.9
iOS; Mac GPU
Jun ‘16

Nov ’16
v0.12
Windows;
Embedding Visualizer

1.0
58x speedup
(on 64 GPUs for Inception v3)
Feb ‘17

of Google directories containing model description files

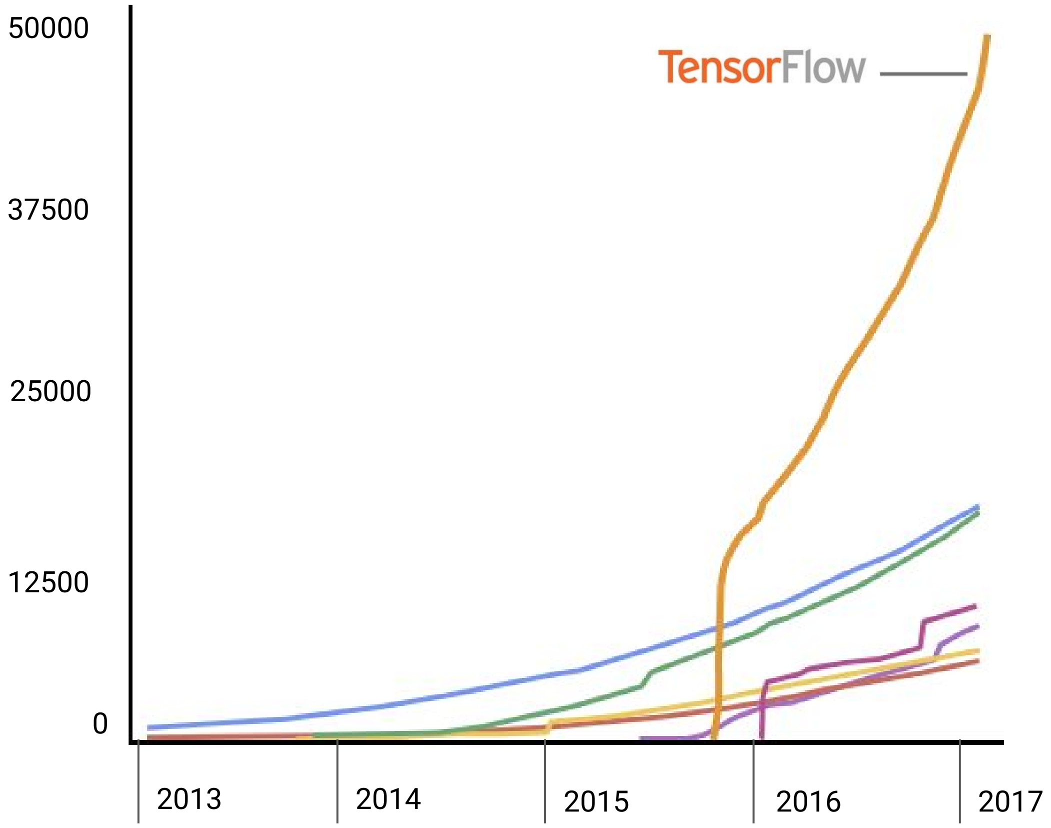


Production use in many areas:

Search
Gmail
Translate
Maps
Android
Photos
Speech
YouTube
Play
… many others ...

Research use for:

100s of projects and papers



50K+

TensorFlow

GitHub Star Count

TensorFlow key concepts

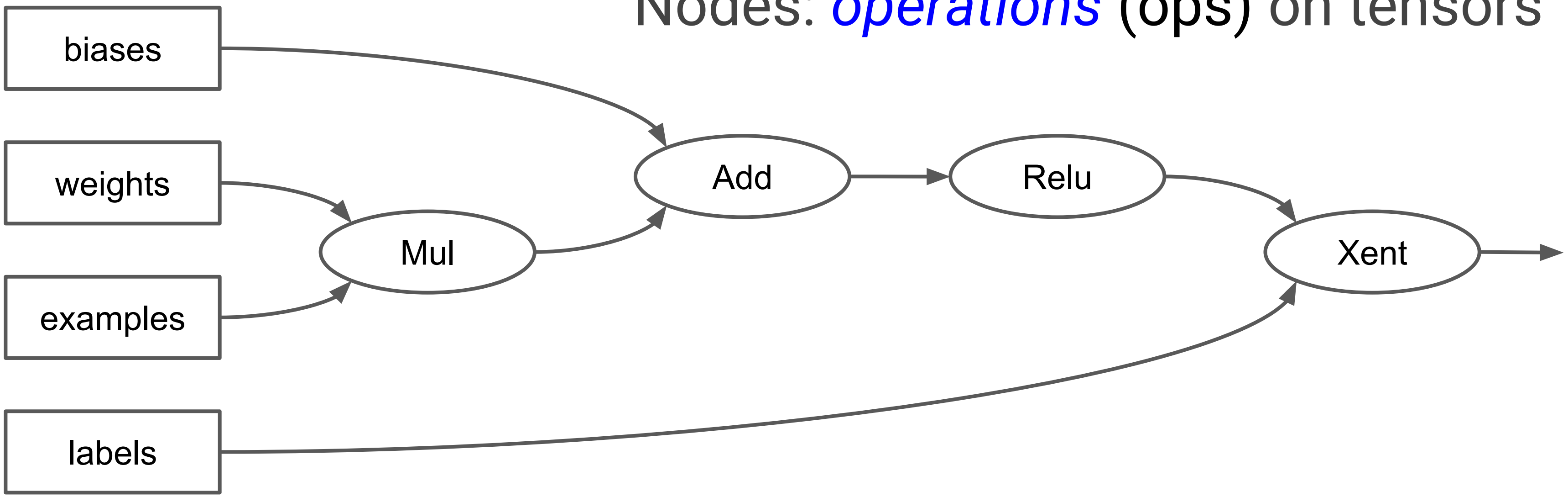
TensorFlow key concepts

- Data flow graph
- Distributed computing
- Model parallelism and data parallelism
- Control flows
- Functions

Dataflow graph

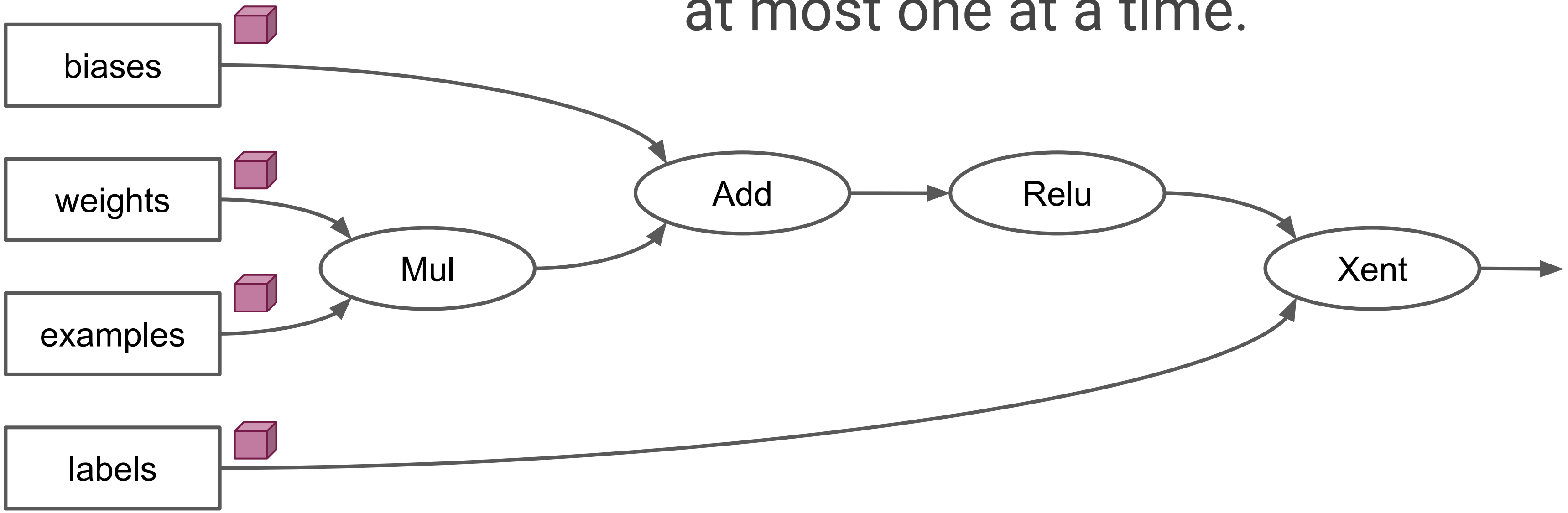
Data: *tensors* (N-dimensional arrays)

Nodes: *operations* (ops) on tensors



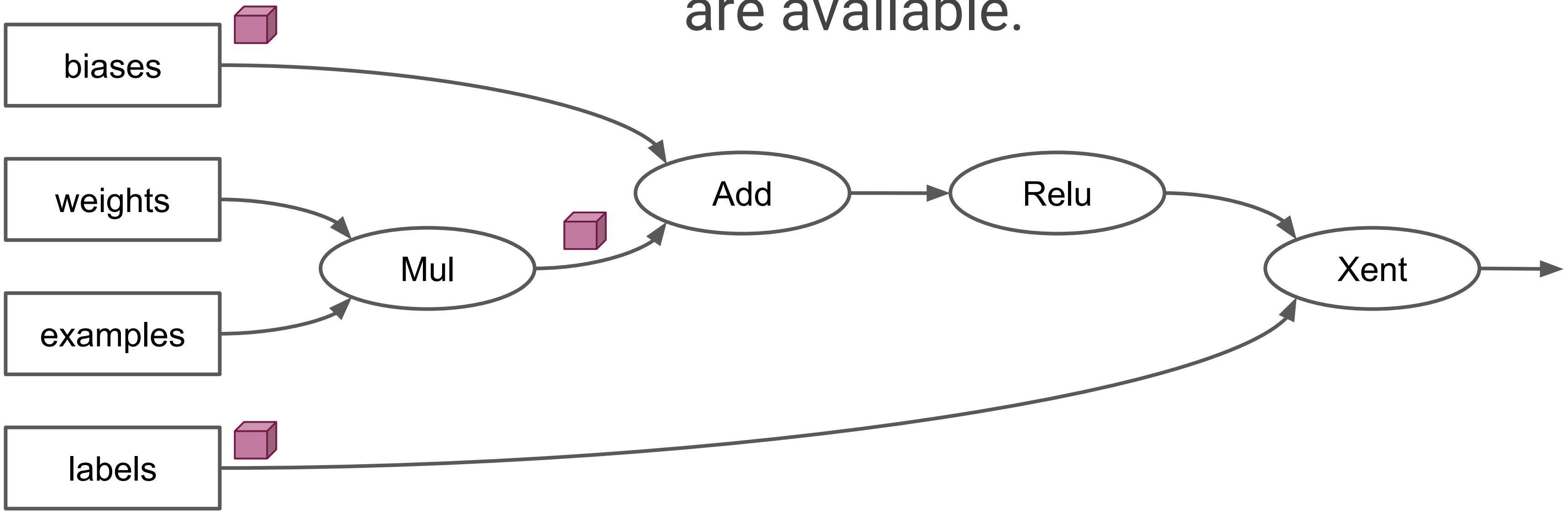
Dataflow computation: basic semantics

Edges hold tensors,
at most one at a time.



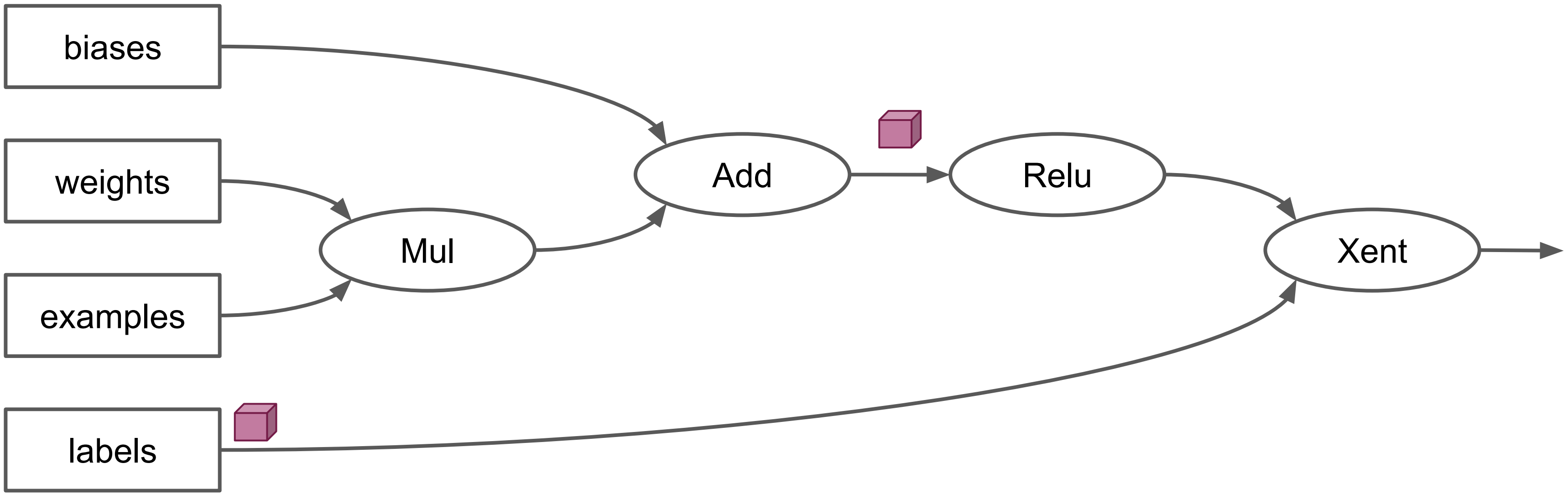
Dataflow computation: basic semantics

A node may fire when all its inputs are available.



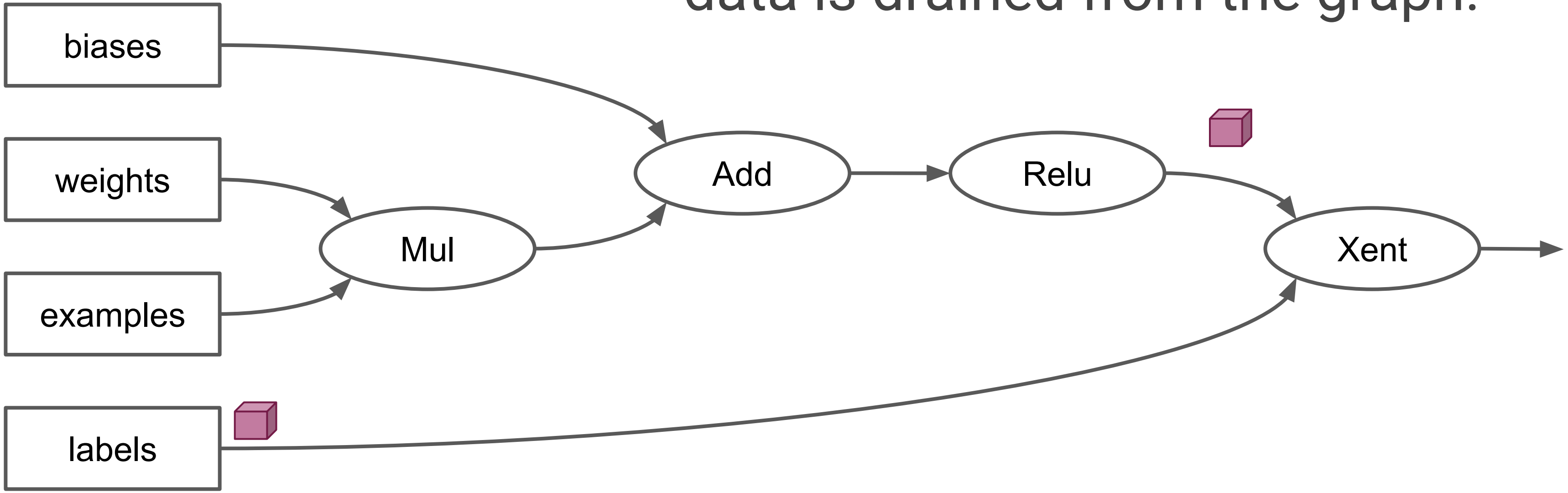
Dataflow computation: basic semantics

After firing, the node is done.



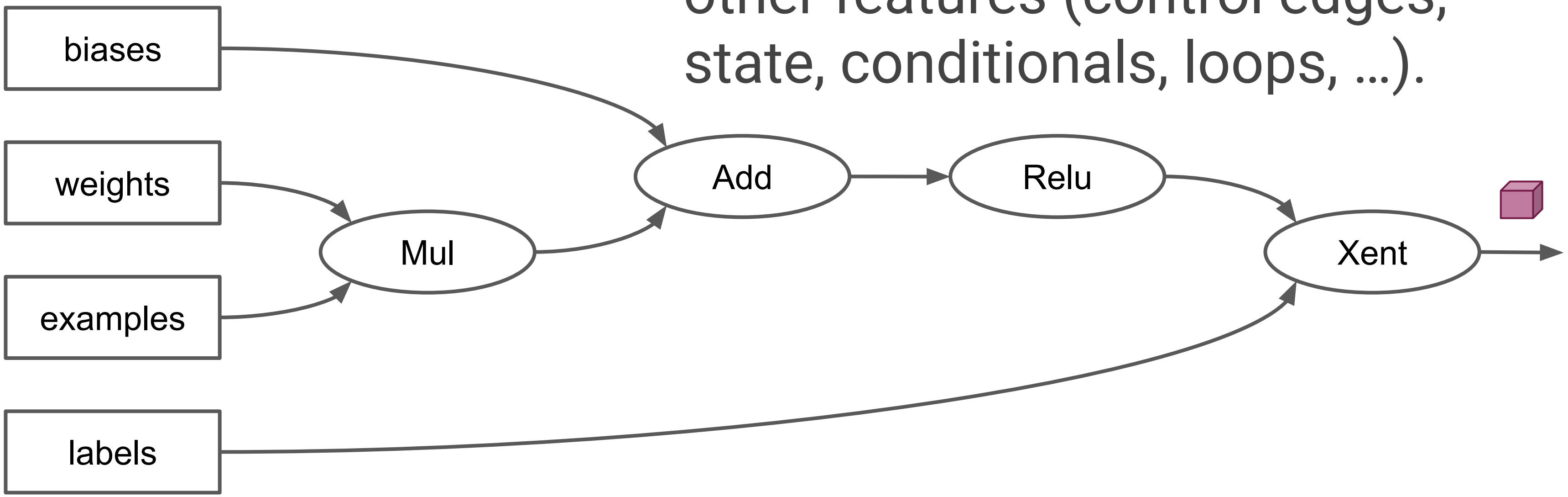
Dataflow computation: basic semantics

Computation ends when all data is drained from the graph.

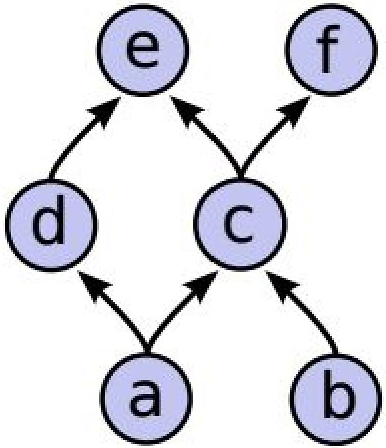


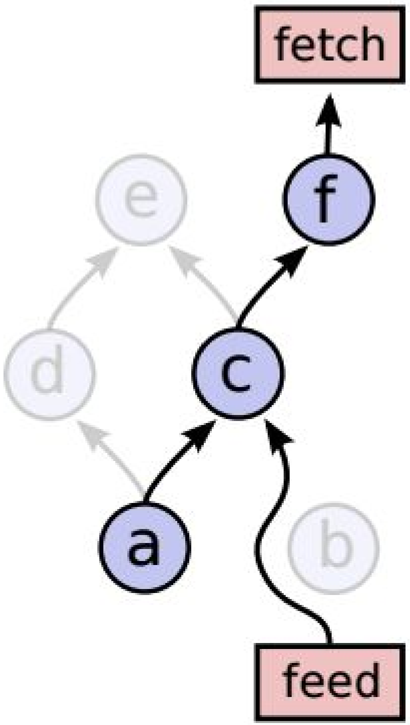
Dataflow computation: basic semantics

This simple model extends to other features (control edges, state, conditionals, loops, ...).



Feeding and Fetching






```
Run(input={"b": ...}, outputs={"f:0"})
```

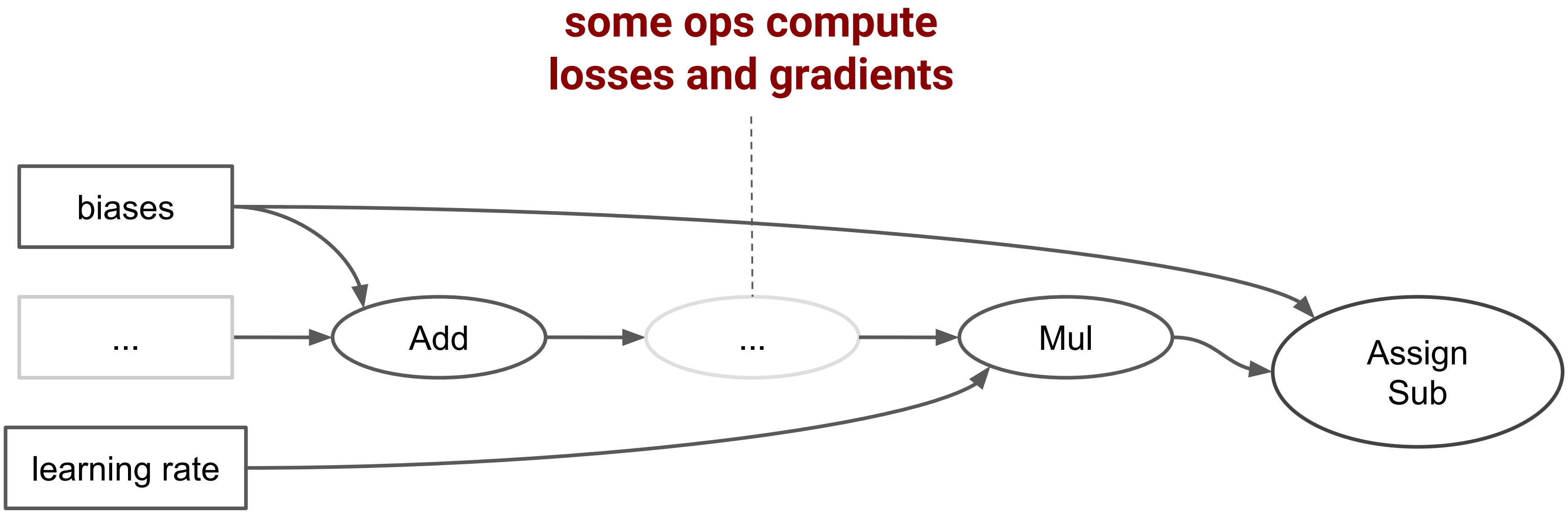
Python API -- Forward Computation

```
import tensorflow as tf
examples = tf.placeholder(tf.float32)
labels = tf.placeholder(tf.int32)
weight = tf.Variable(tf.zeros([784,10]))
bias = tf.Variable(tf.zeros([10]))
y = tf.matmul(examples, weight) + bias
loss = tf.cross_entropy(y, labels)
```

Python API -- Backward computation

```
weights_grad, bias_grad = tf.gradients(loss, [weights, bias])
new_weights = tf.assign_sub(weight, lr * weights_grad)
new_bias = tf.assign_sub(bias, lr * bias_grad)
train_op = tf.group([new_weights, new_bias])
```

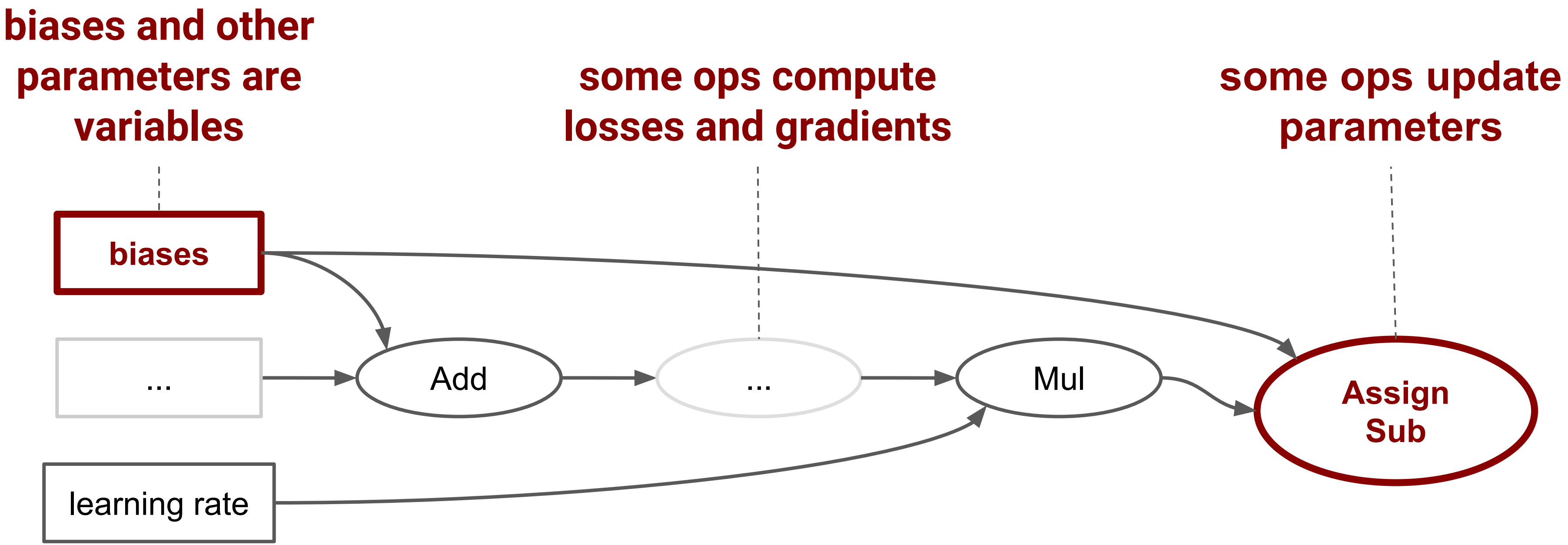
Dataflow computation *also “backward”*



Stateless ops vs stateful ops

- Two types of TensorFlow ops
  - Stateless ops
  - Stateful ops
- Stateless ops
  - Pure functions
  - Execution of the ops has no side effects
  - Output is a deterministic function of the input
  - Most mathematical ops, like MatMul, BiasAdd, Conv and etc are stateless ops
- Stateful ops
  - Not pure function
  - Op has states that persist across session runs
  - Variables are stateful ops

Dataflow computation *with state*



Graph definition vs graph execution

- Graph definition: define the data flow graph
  - Forward subgraph, backward subgraph, and a train subgraph
  - Only specifies how computations should be done
  - Typically done in some frontend, e.g. Python
- Graph execution is completely separate from graph definition
  - The frontend sends GraphDef to the backend
  - Backend will analyze the graph, and will carry out a few rounds of optimization:
    - constant folding, common subexpression elimination and etc
  - Graph execution is done on the backend
  - Actually executes necessary ops in a graph on real data
    - E.g. update the model params
  - Graph execution is typically driven by a frontend as well

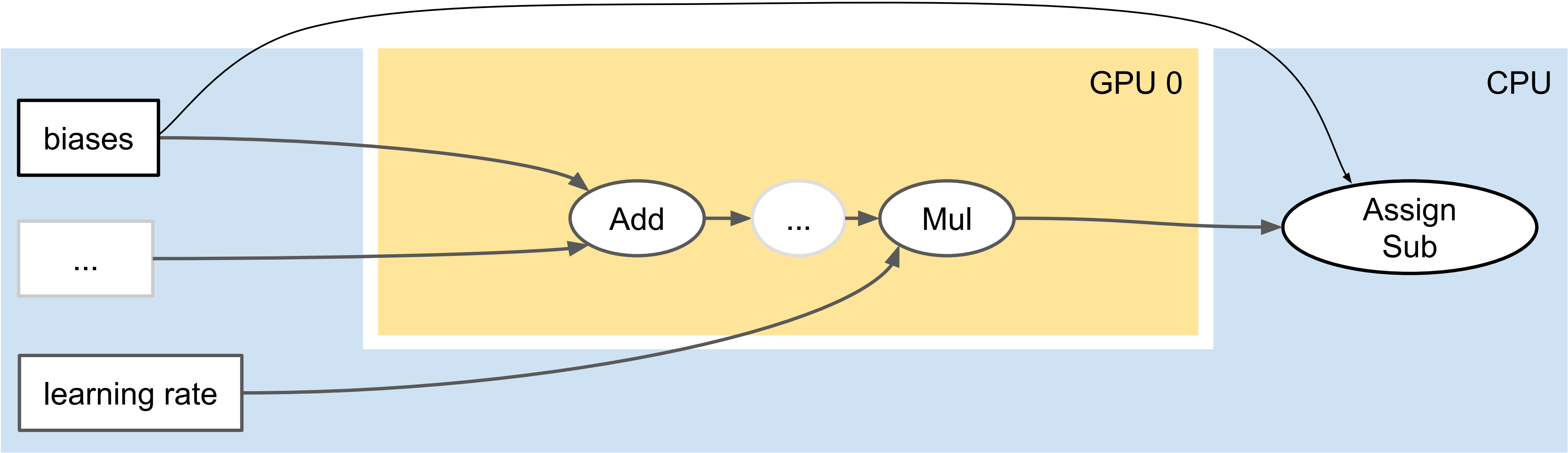
Python API - Drive the training loop

```
init = tf.initialize_all_variables()
sess = tf.Session()
sess.run(init)
for i in range(1000):
  batch_xs, batch_ys = mnist.train.next_batch(100)
  sess.run(train_op, feed_dict={examples: batch_xs, labels:
batch_ys})
```

Distributed execution of TF graph

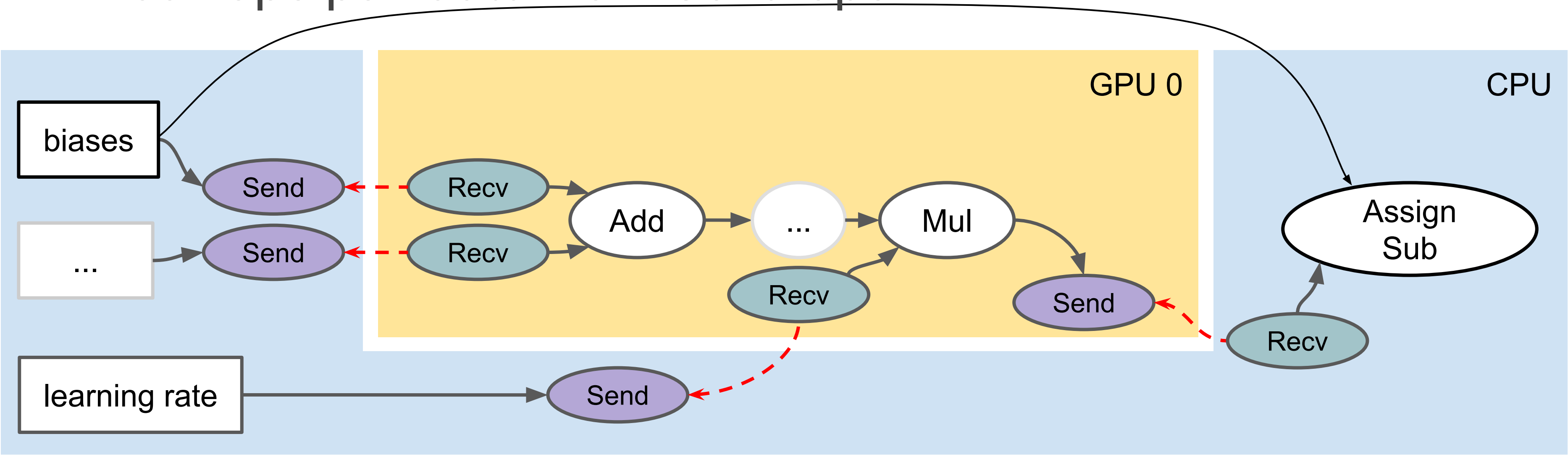
- With TensorFlow, you can easily distribute graph execution to multiple devices, and/or multiple machines
- It requires very minimal code change
- TensorFlow will automatically insert nodes to the graph to enable distributed execution of the graph

Distributed dataflow computation



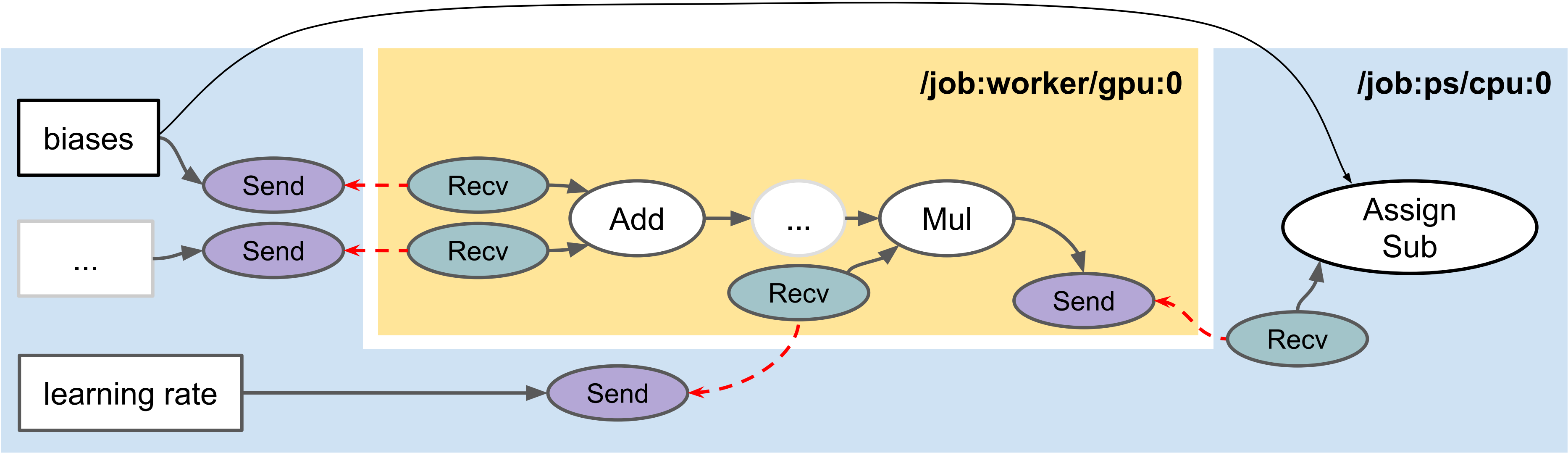
Distributed dataflow computation

- TensorFlow adds *Send/Recv* ops to transport tensors.
- *Recv* ops pull data from *Send* ops.



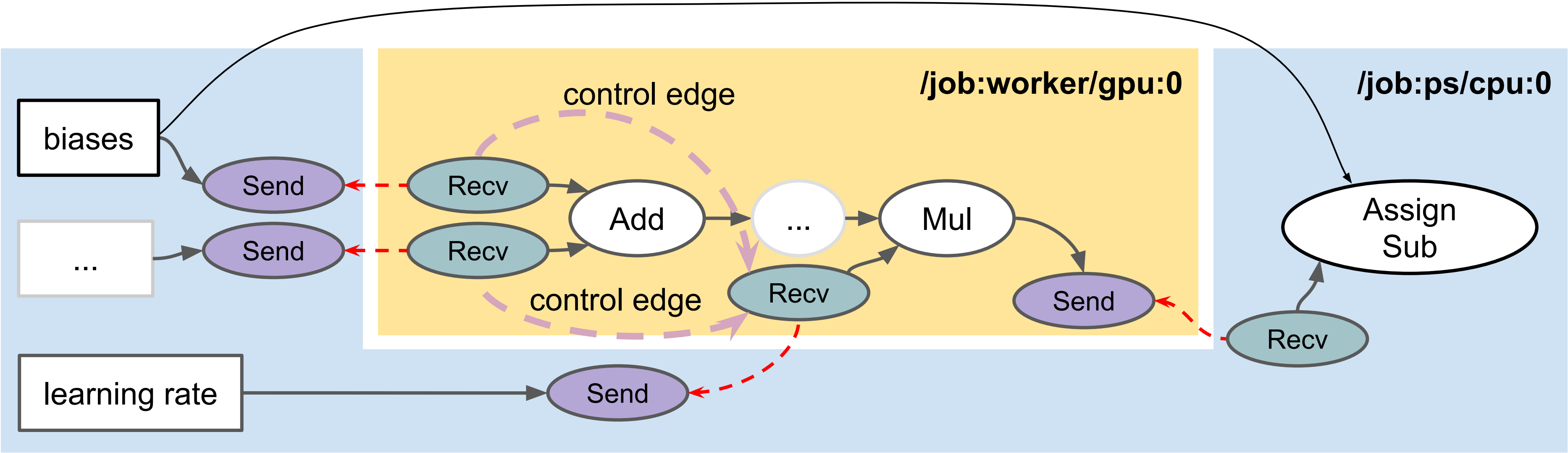
Distributed dataflow computation

- Communication across machines is abstracted just like cross-device communication within a machine.



Control edges for dataflow scheduling

- Control edges impose additional execution ordering.
- Critical-path analysis informs their addition.



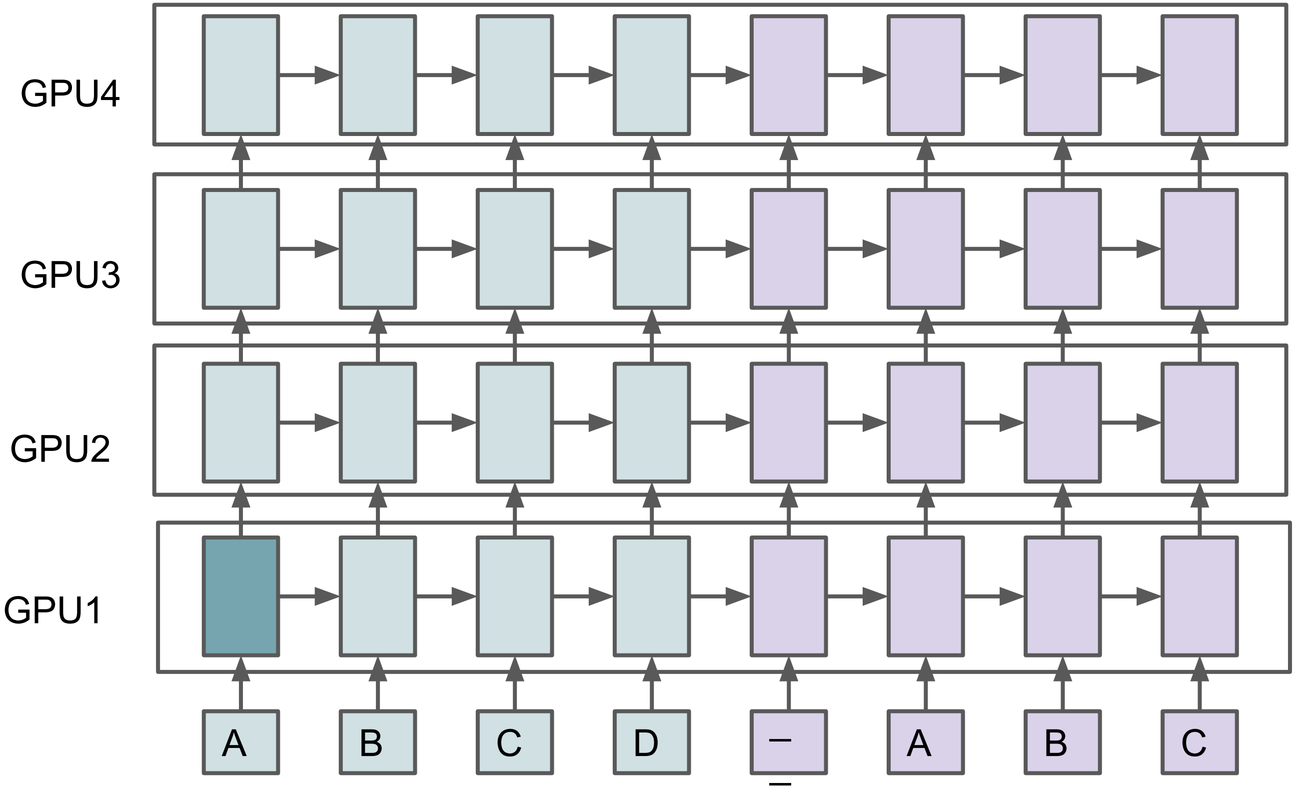
Python API - Distributed computation

Stacked LSTMs on one single devide

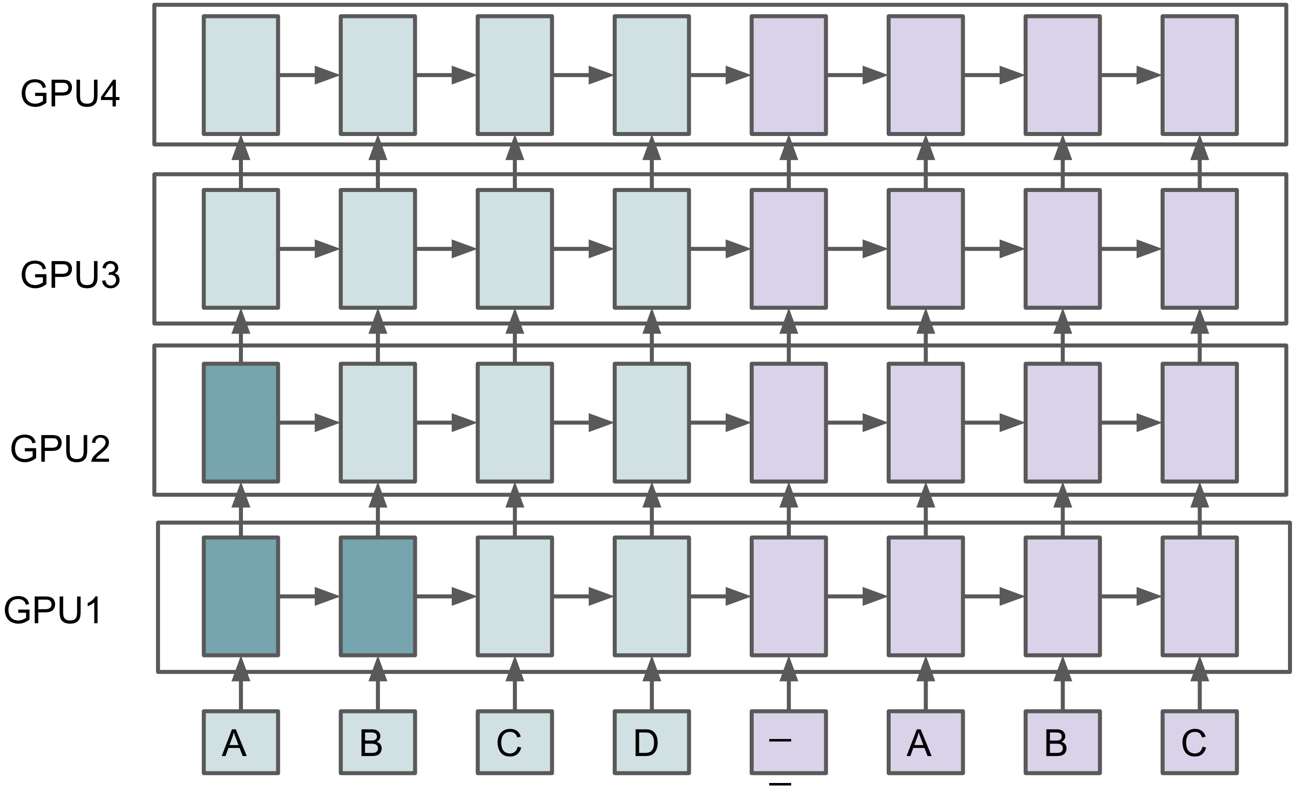
```
for i in range(8):
 for d in range(4): # d is depth
     input = x[i] if d is 0 else m[d-1]
     m[d], c[d] = LSTMCell(
           input, mprev[d], cprev[d])
     mprev[d] = m[d]
     cprev[d] = c[d]
```

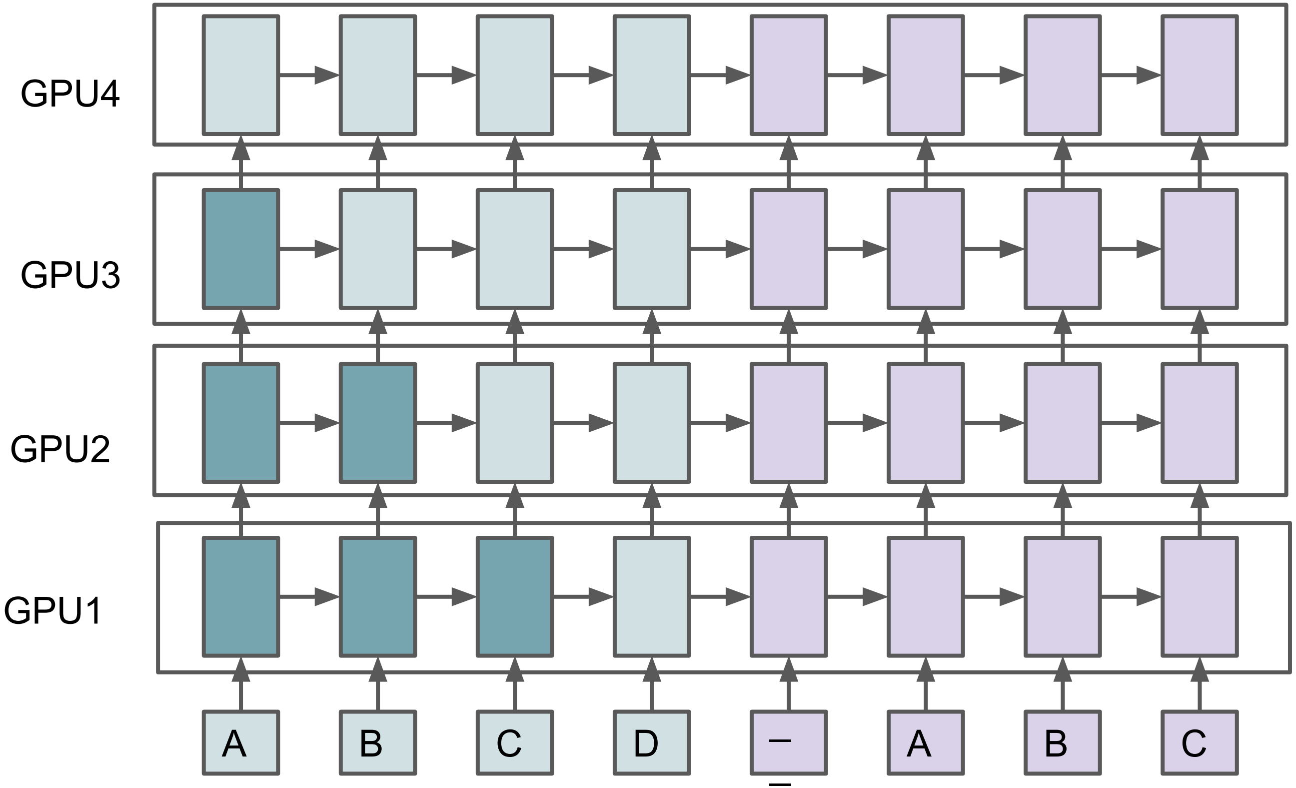
Stacked LSTMs on Multiple devices

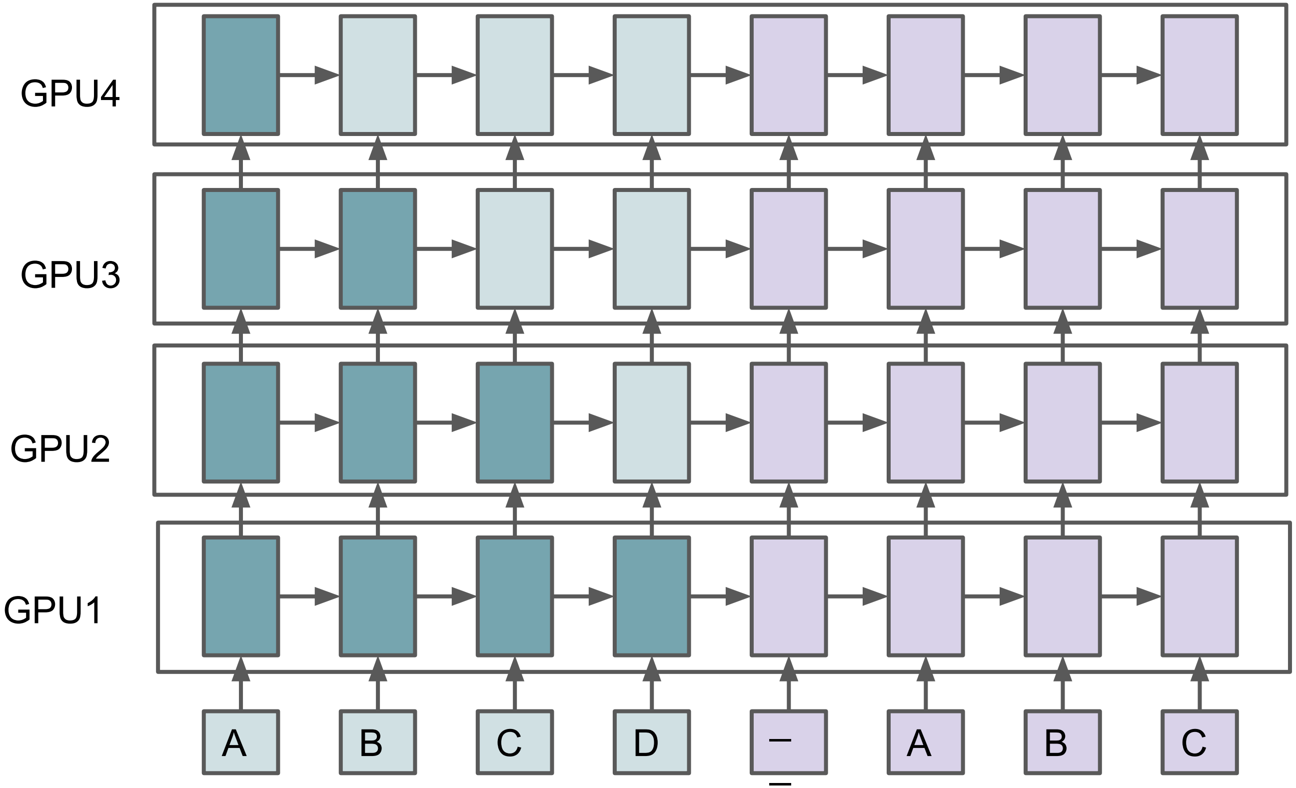
```
for i in range(8):
 for d in range(4): # d is depth
   with tf.device("/gpu:%d" % d):
     input = x[i] if d is 0 else m[d-1]
     m[d], c[d] = LSTMCell(
           input, mprev[d], cprev[d])
     mprev[d] = m[d]
     cprev[d] = c[d]
```

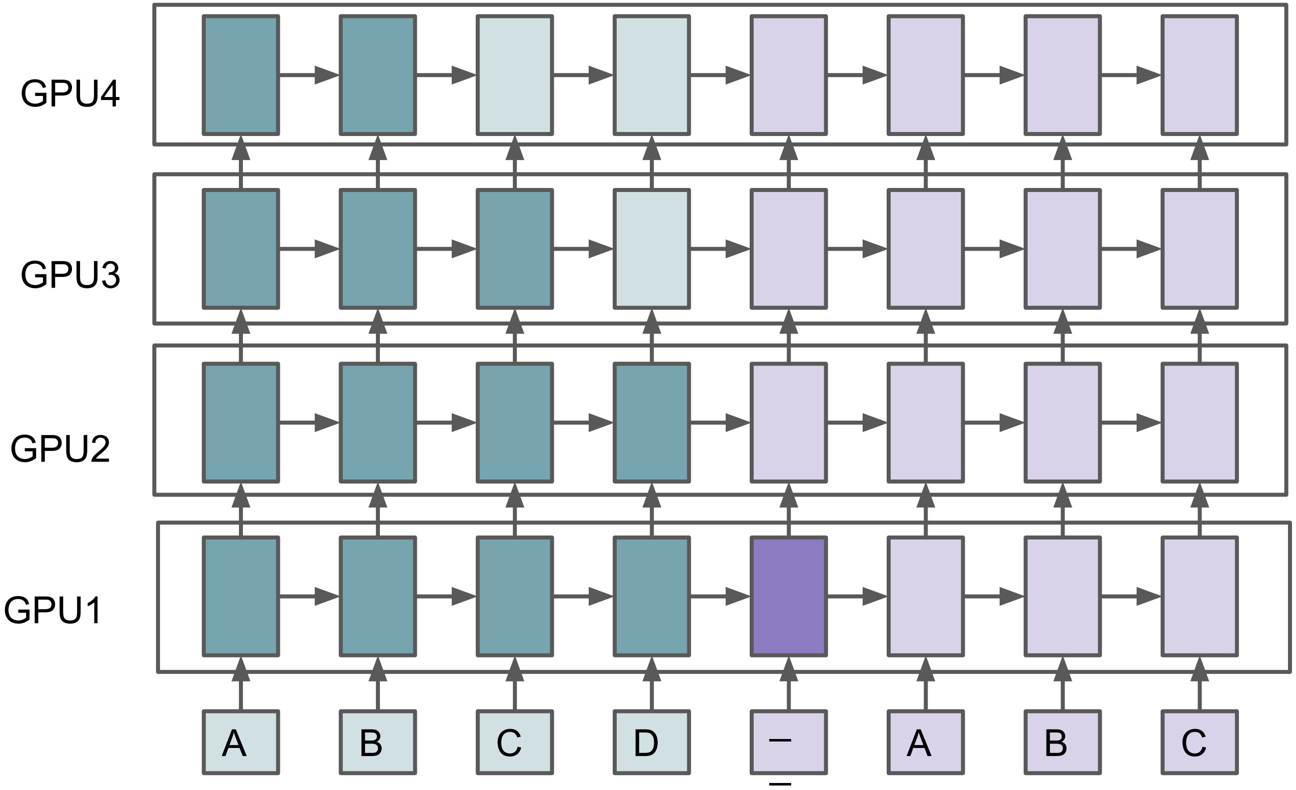


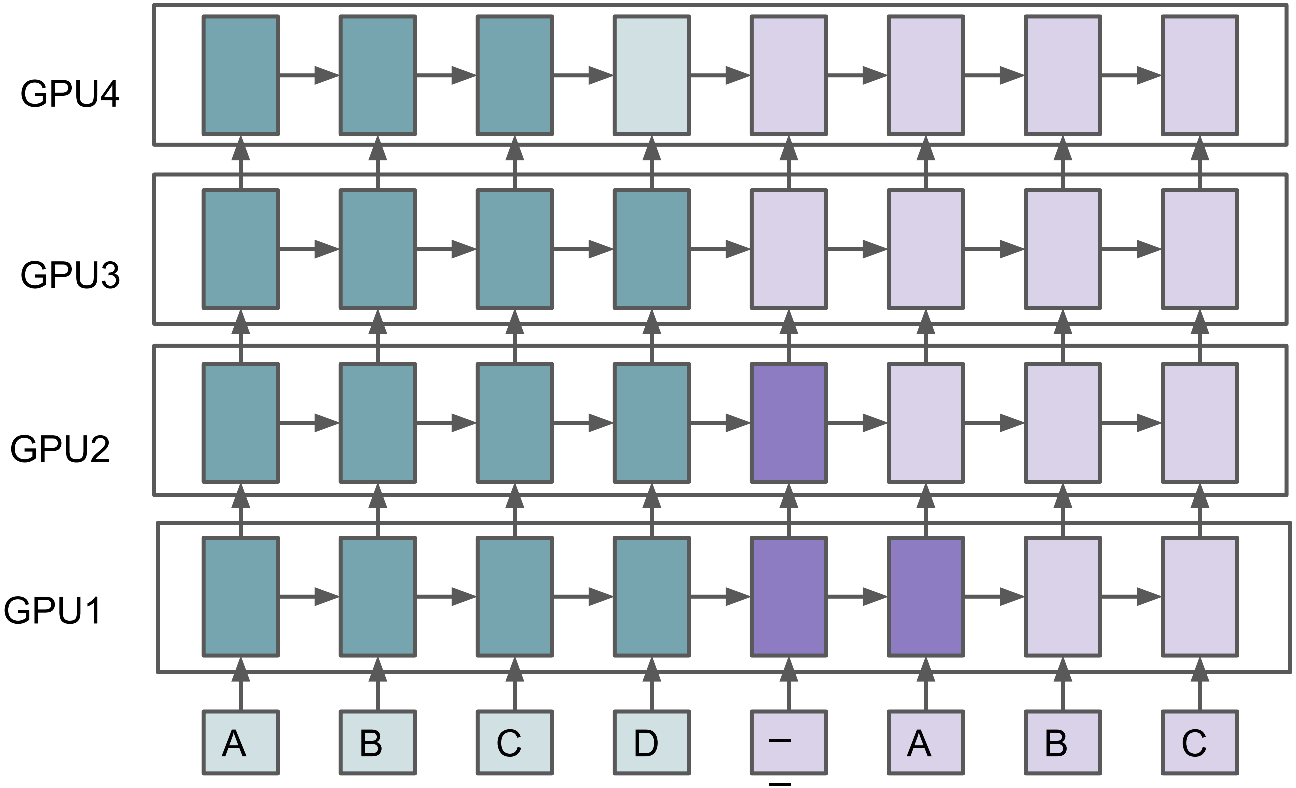
GPU4
GPU3
GPU2
GPU1
A
B
C
D
_
A
B
C

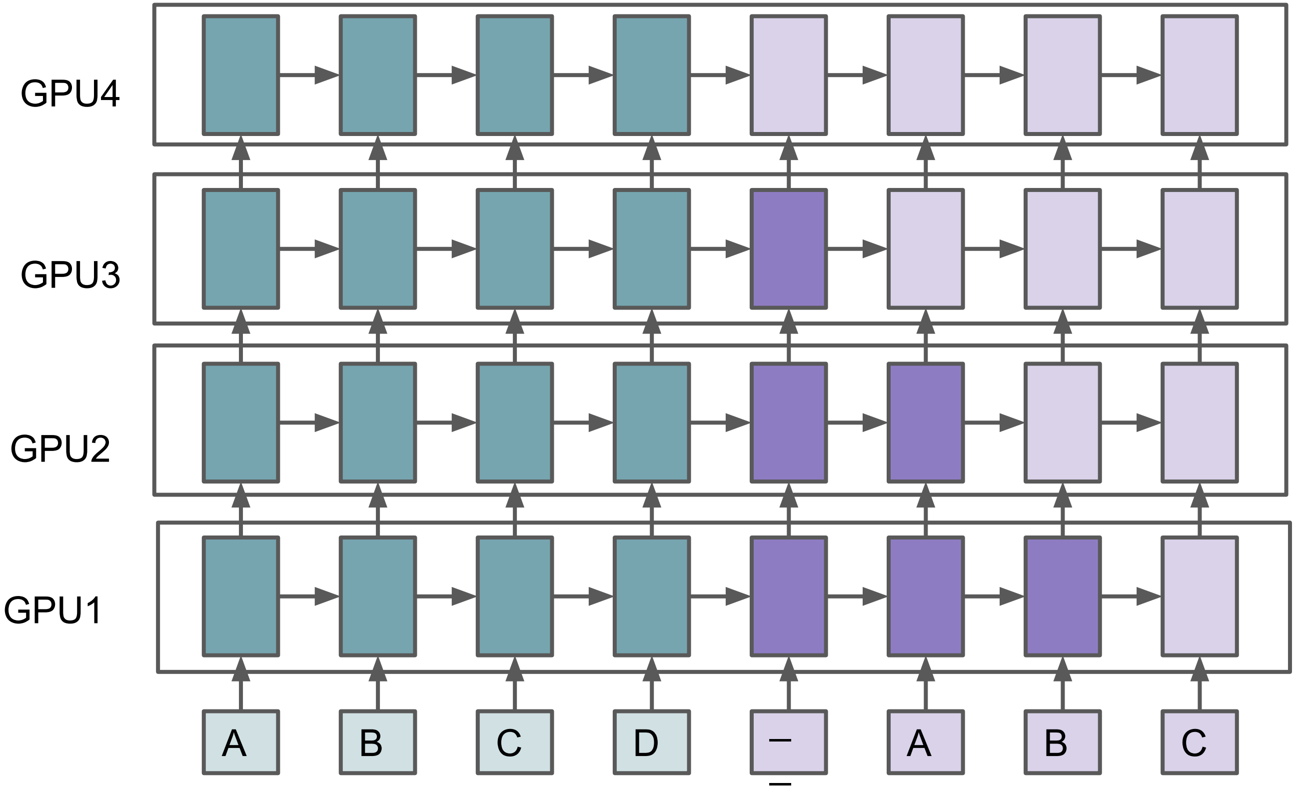

GPU4
GPU3
GPU2
GPU1
A
B
C
D
_
A
B
C

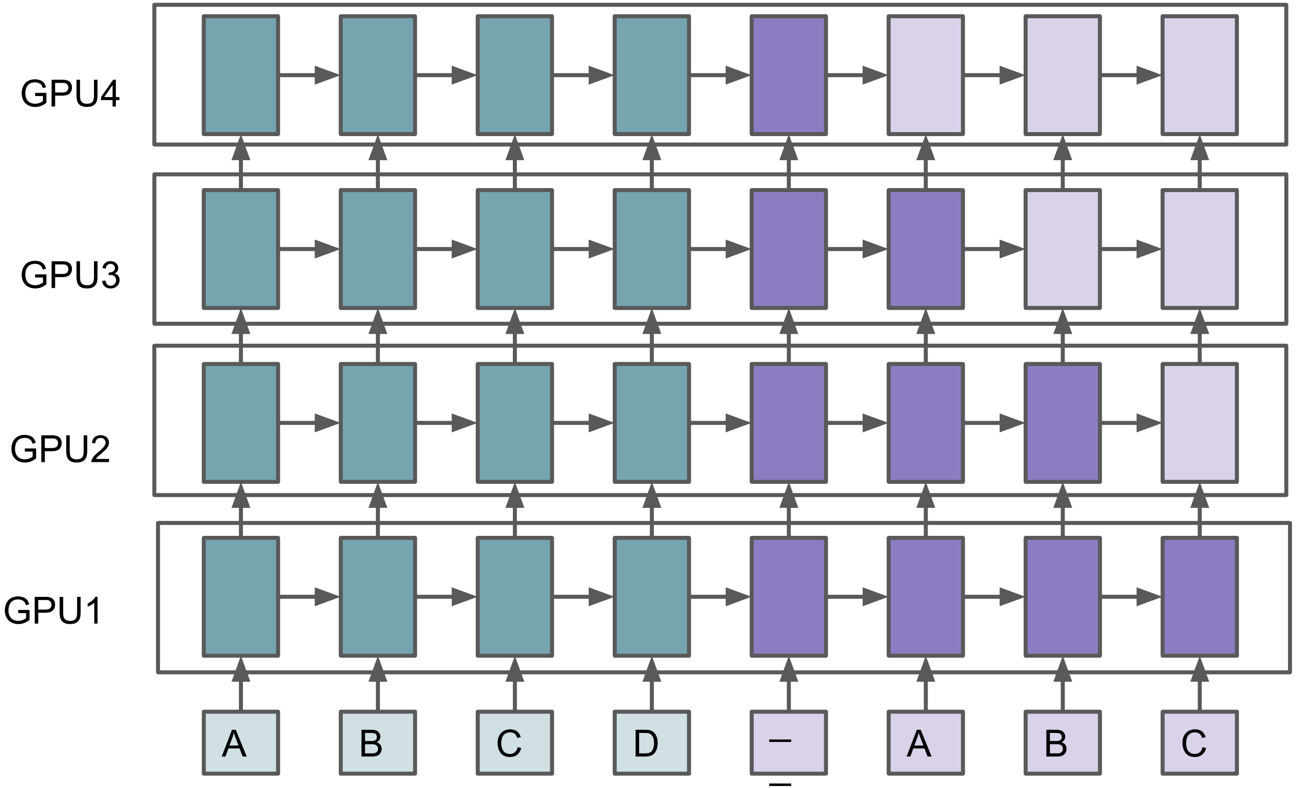

GPU4
GPU3
GPU2
GPU1
A
B
C
D
_
A
B
C

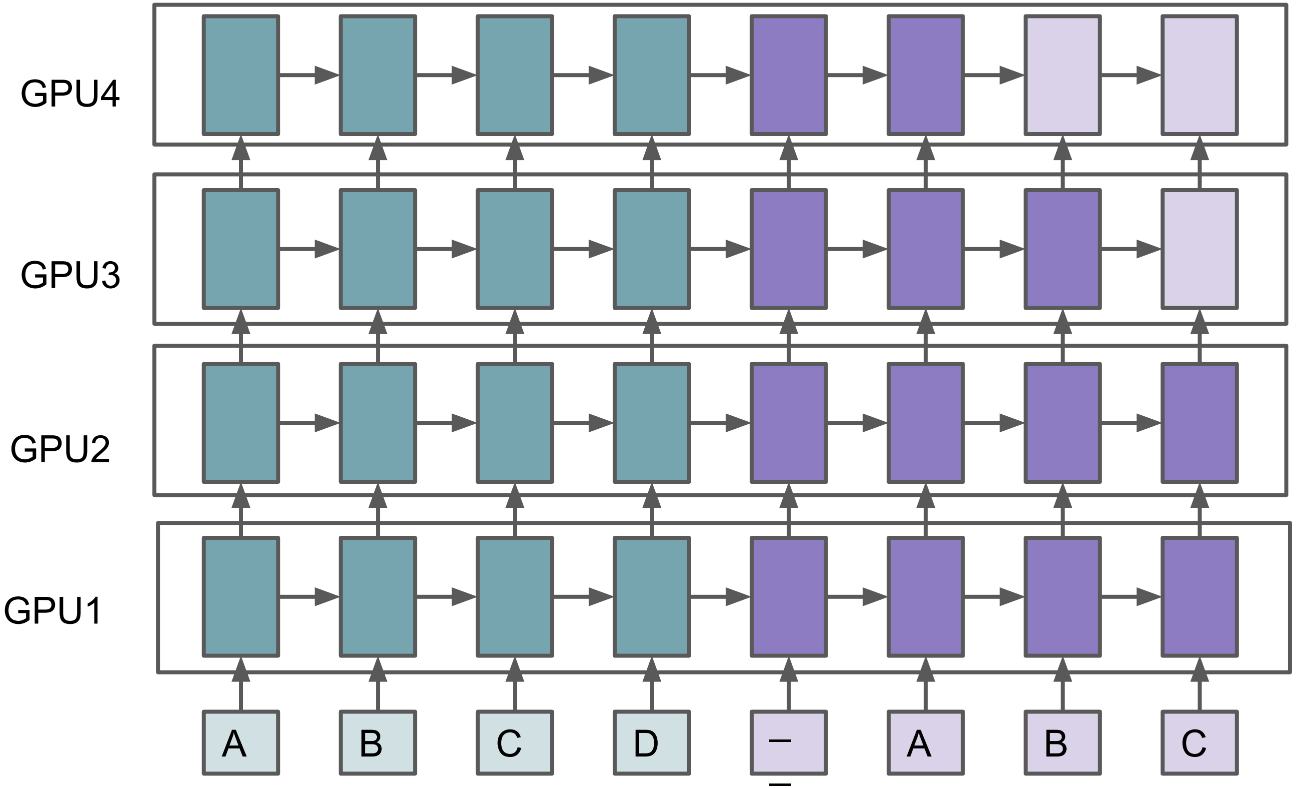

GPU4
GPU3
GPU2
GPU1
A
B
C
D
_
A
B
C

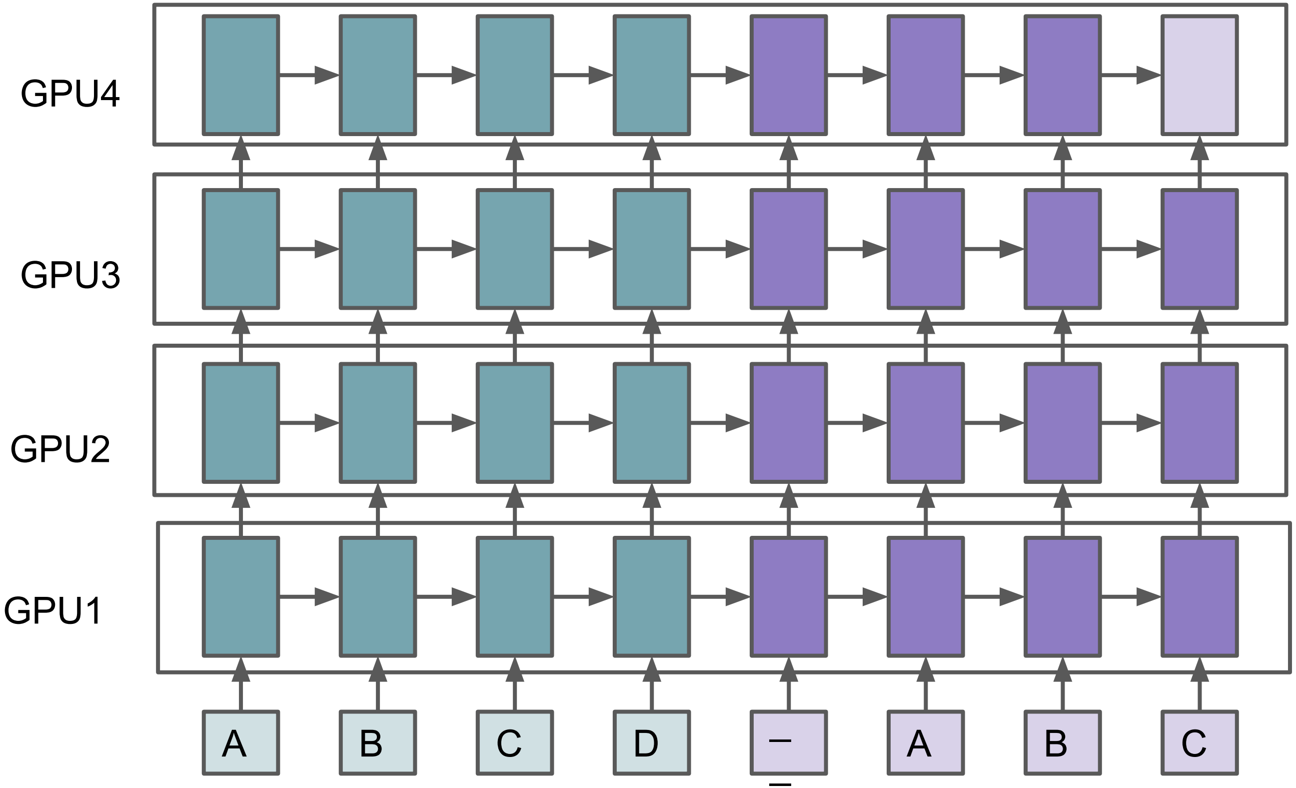

GPU4
GPU3
GPU2
GPU1
A
B
C
D
_
A
B
C

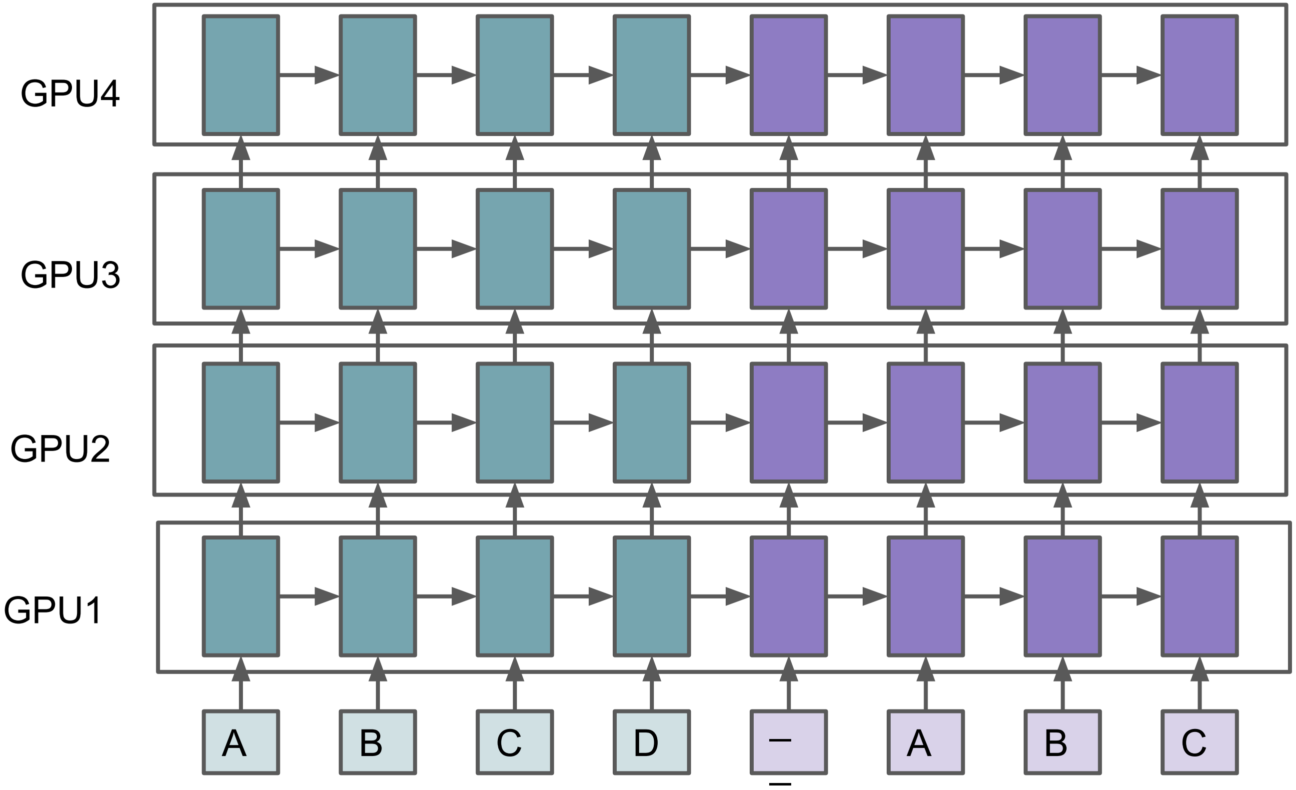

GPU4
GPU3
GPU2
GPU1
A
B
C
D
_
A
B
C


GPU4
GPU3
GPU2
GPU1
A
B
C
D
_
A
B
C


GPU4
GPU3
GPU2
GPU1
A
B
C
D
_
A
B
C


GPU4
GPU3
GPU2
GPU1
A
B
C
D
_
A
B
C

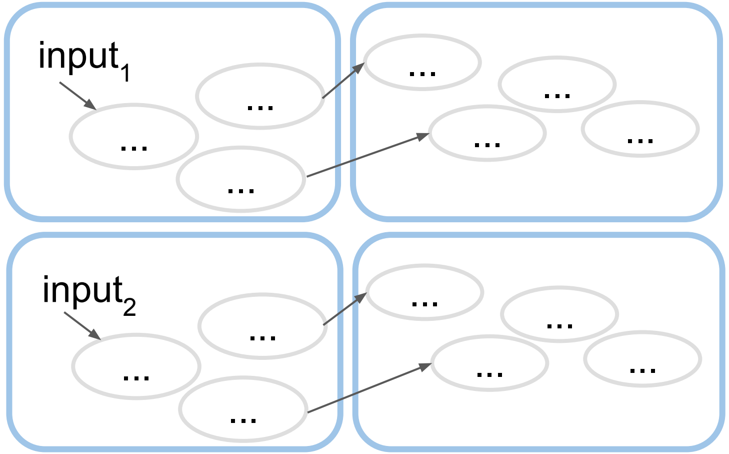

GPU4
GPU3
GPU2
GPU1
A
B
C
D
_
A
B
C


GPU4
GPU3
GPU2
GPU1
A
B
C
D
_
A
B
C

Parallelism

TensorFlow enables *model parallelism* and *data parallelism*:

- A graph can be split across several devices.
- Many graph replicas can process inputs in parallel.



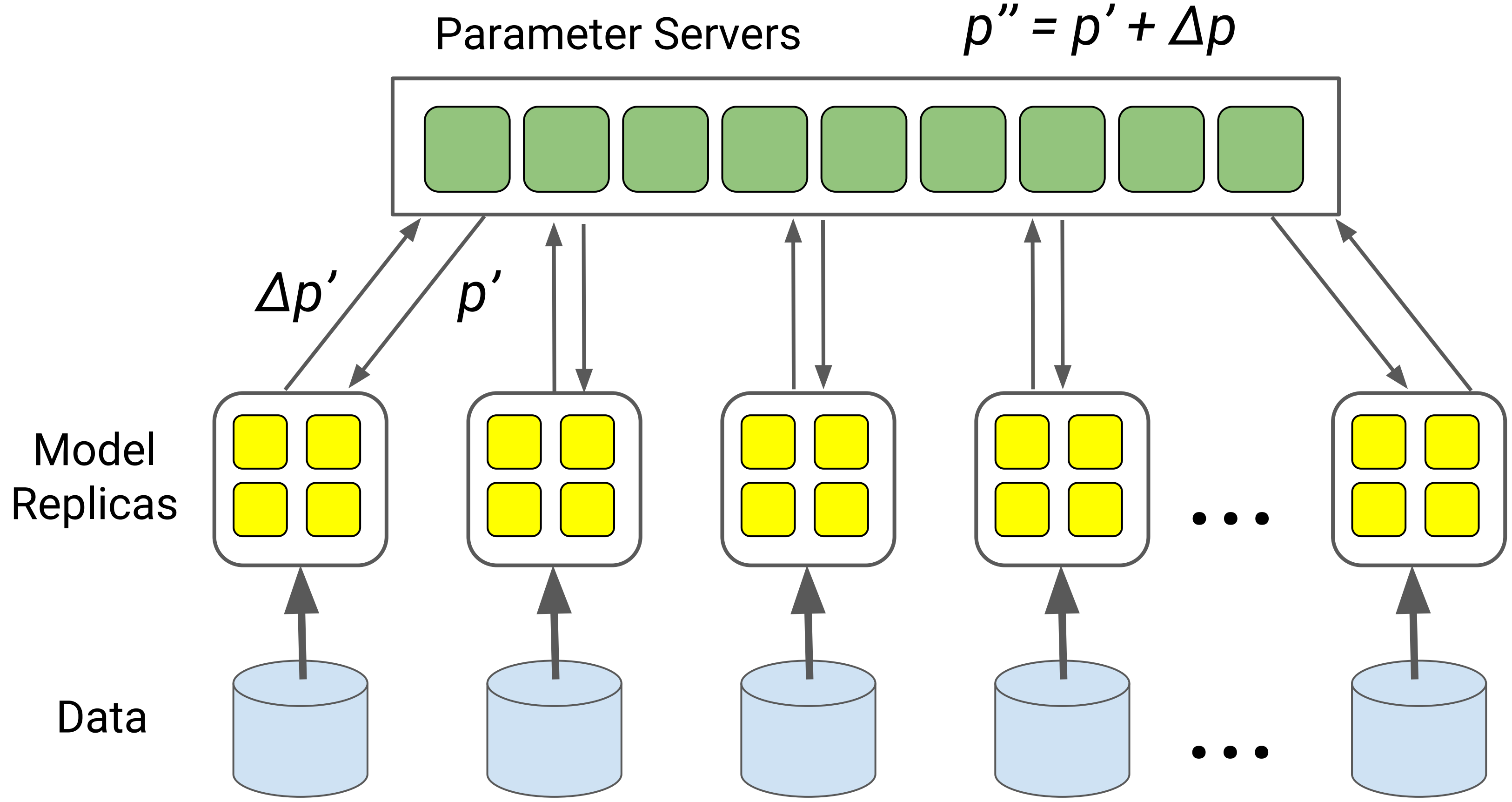
Exploiting Model Parallelism

- Different levels of model parallelism:
  - Across machines, across devices, across cores, instruction parallelism
- Across machines: limited by network bandwidth / latency
- Across devices: for GPUs, often limited by PCIe bandwidth.
- Across cores: thread parallelism. Almost free.
- On a single core: Instruction parallelism (SIMD). Just free.

Data Parallelism

- Use multiple model replicas to process different examples at the same time
  - All collaborate to update model state (parameters) in shared parameter server(s)
- Speedups depend highly on kind of model
  - Dense models: 10-40X speedup from 50 replicas
  - Sparse models:
    - support many more replicas
    - often can use as many as 1000 replicas

Data Parallelism



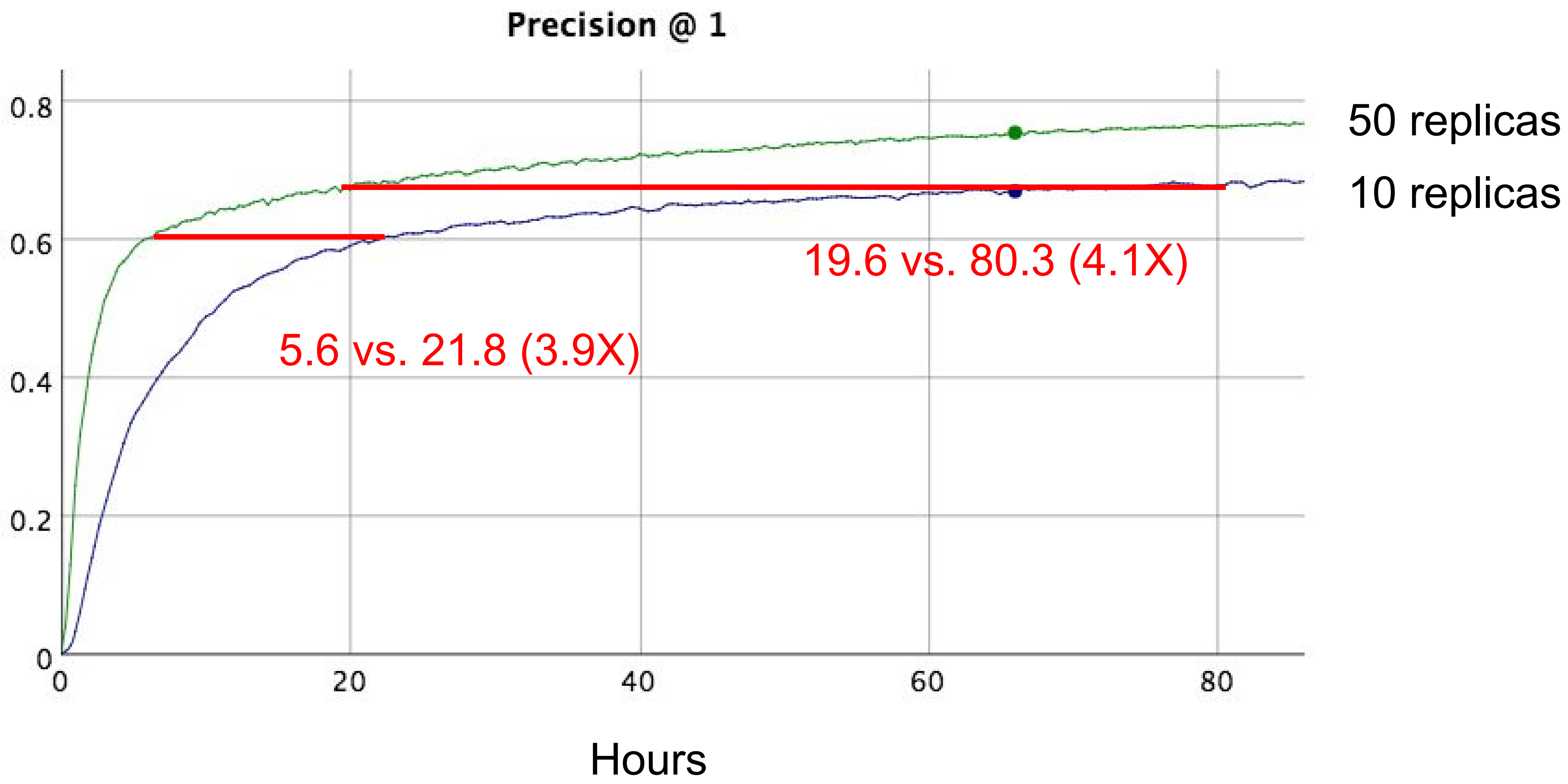
Data parallelism

- Params access and update can be synchronous and asynchronous
- In asynchronous model, this is the famous downpour SGD algorithm
  - No guarantee on the consistency of the params
- Synchronous params update usually works better
  - At the cost of training speech due to synchronous cost

Success of Data Parallelism

- Data parallelism is **really important** for many of Google's problems (very large datasets, large models):
  - RankBrain: 500 replicas
  - ImageNet Inception: 50 GPUs, ~40X speedup
  - SmartReply: 16 replicas, each with multiple GPUs
  - Language model on "One Billion Word": 32 GPUs

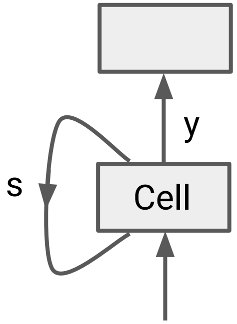
Image Model Training Time: 10 vs 50 GPUs



Control-flow constructs

Conditionals and iteration can be useful in building dynamic models.

For example, Recursive Neural Networks (RNNs) are widely used for speech recognition, language modeling, translation, image captioning, and more.

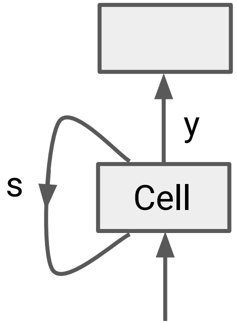



```
while step < len(seq):
    s, y = Cell(seq[step], s)
    ys = ys.append(y)
    step = step + 1
```

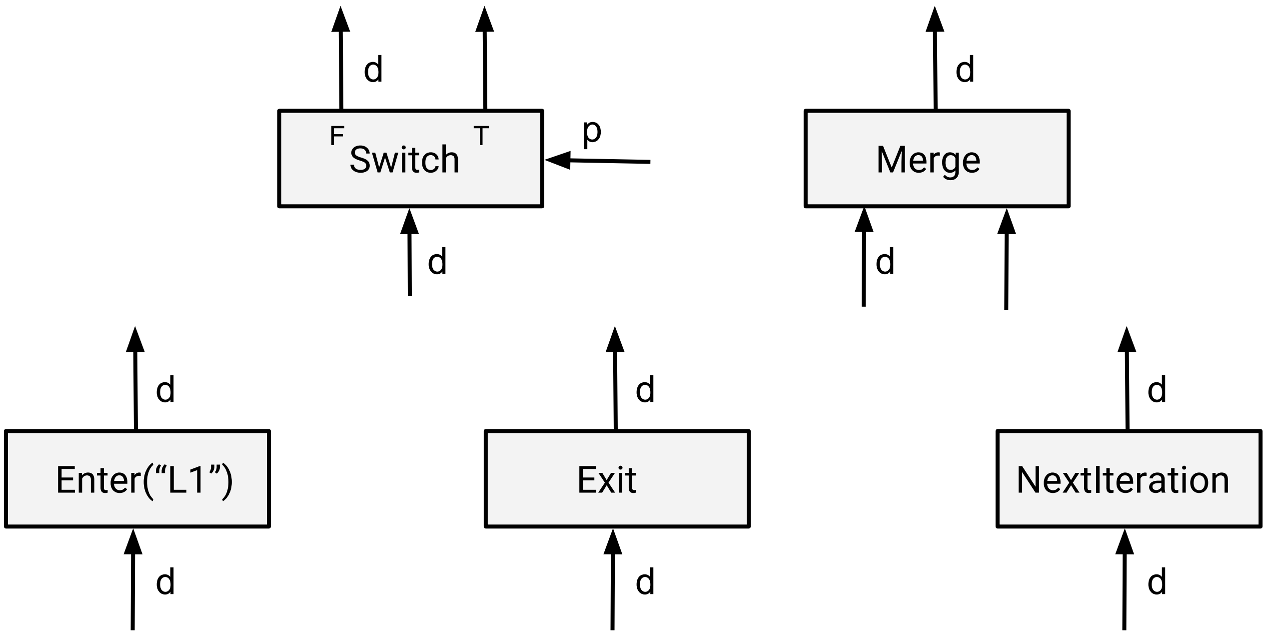
Control-flow constructs

We want:

- Fit with the computational model
- Parallel execution
  (sometimes of the iterations of a loop)
- Distributed execution
- Automatic generation of gradient code



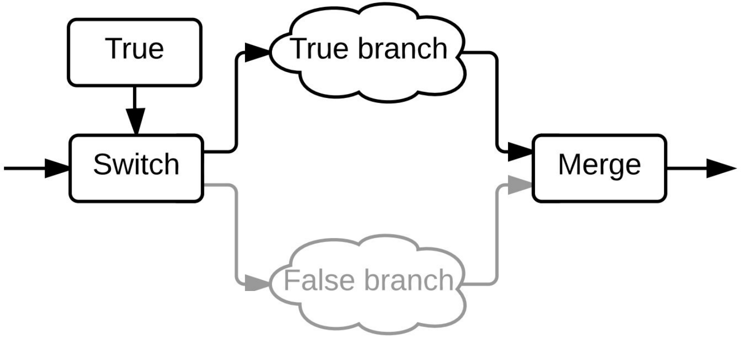
Dynamic dataflow architecture [Arvind et al., 1980s]



Execution contexts identify different invocations of the same node. New operations manipulate these contexts.

Python API - Condition

- Conditional execution of a sub-graph




```
x = tf.constant(2)
y = tf.constant(5)
def f1(): return tf.mul(x, 17)
def f2(): return tf.add(y, 23)
r = cond(tf.less(x, y), f1, f2)
```

Python API - While Loop

- Repeated execution of a sub-graph
- Very useful for RNN
- TF essentially a programming language

```
i = tf.constant(0)
c = lambda i: tf.less(i, 10)
b = lambda i: tf.add(i, 1)
r = tf.while_loop(c, b, [i])
```

TF is essentially a programming language

- With if, and while, TF is basically a programming langugage
- TF can easily extended with custom kernels
- Can be used to implement very complicated logic
    - E.g. BeamSearch can implemented entirely in TF

Python API - Functions

- You can define functions to combine primitive ops into a logical op

```
@function.Defun(*[self._dtype] * 2, func_name='BatchNorm')
def BatchNorm(h, gamma):
  # Uses a simple two-pass algorithm to compute mean and var for
  # numerical stability.
  mean = tf.reduce_mean(h, [0])
  var = tf.reduce_mean(tf.square(h - mean), [0])
  epsilon = tf.constant(1e-8)
  rstd = tf.rsqrt(var + epsilon)
  return gamma * (h - mean) * rstd
```

- TensorFlow executor treats functions as if they are primitive ops

Function benefits

- Reduce the size of the graphs
- Benefits of monolithic op, but without the hassle of writing the gradient functions.
- Reduce the memory usage
  - Intermediate results are being discarded
  - Could mean more computation during backprop.

A Few TensorFlow Community Examples

- DQN: github.com/nivwusquorum/tensorflow-deepq
- NeuralArt: github.com/woodrush/neural-art-tf
- Char RNN: github.com/sherjilozair/char-rnn-tensorflow
- Keras ported to TensorFlow: github.com/fchollet/keras
- Show and Tell: github.com/jazzsaxmafia/show_and_tell.tensorflow
- Mandarin translation: github.com/jikexueyuanwiki/tensorflow-zh

...

More examples related to NLP

- POS tagging: https://github.com/rockingdingo/deepnlp/tree/master/deepnlp/pos
- Syntaxnet: https://research.googleblog.com/2016/05/announcing-syntaxnet-worlds-most.html
- Seq2seq: https://opensource.googleblog.com/2017/04/tf-seq2seq-sequence-to-sequence-framework-in-tensorflow.html
- Many tutorials: https://tensorflow.google.cn/tutorials/
-

Concluding remarks

Conclusions

- **Ease of expression**:
  - Lots of crazy ML ideas are just different Graphs
  - Non-NN algorithms can also benefit if it maps to graph.
- **Portability**: can run on wide variety of platforms
- **Scalability**:
  - Easy to scale
  - Much faster training

Conclusions (cont.)

- Open Sourcing of TensorFlow
  - Rapid exchange of research ideas (we hope!)
  - Easy deployment of ML systems into products
  - TensorFlow community doing interesting things!