TensorFlow

ZHAW Workshop

Effrosyni Kokiopoulou, Luciano Sbaiz

www.tensorflow.org

15 September 2016


Google Research Europe


MACHINE LEARNING
MACHINE PERCEPTION
NATURAL LANGUAGE UNDERSTANDING
We are working on big problems

Deep Learning

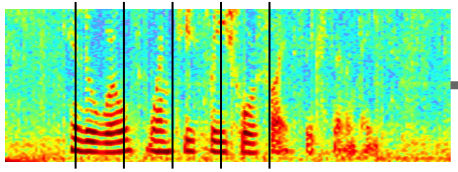
Universal Machine Learning

...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
...

Speech Recognition



Acoustic Input

Deep
Recurrent
Neural Network

"How cold is
it outside?"

Text Output

Reduced word errors by more than 30%

Google Research Blog - August 2012, August 2015

Combined Vision + Translation

IS THIS A

CAT or DOG?

CAT DOG

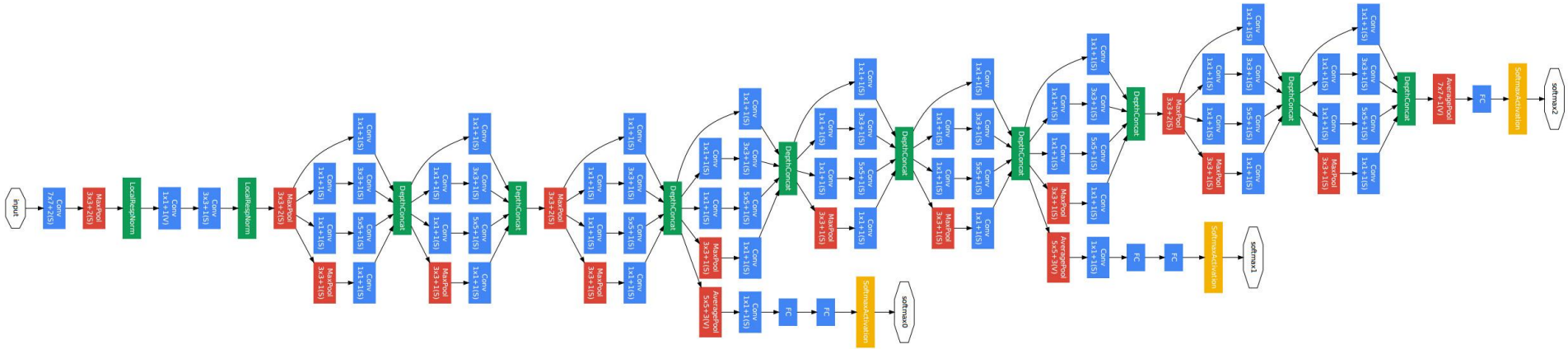
OUTPUT LAYER

ACTIVATED NEURONS

INPUT LAYER

DEEP *neural* NETWORK

Real Models Can Be Complex (GoogLeNet, 2015)



Going Deeper with Convolutions

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich

ArXiv 2014, CVPR 2015

Motivations: Key Things in a ML Framework

- **Ease of expression**: for lots of crazy ML ideas/algorithms
- **Portability**: can run on wide variety of platforms
- **Scalability**: can run experiments quickly
- **Reproducibility**: easy to share and reproduce research
- **Production readiness**: go from research to real products

Motivations: History

DistBelief (1st system):

- Great for scalability, and production training of basic kinds of models
- Not as flexible as we wanted for research purposes

Better understanding of problem space allowed us to make some dramatic simplifications

Just better, almost by all means!!

If we like it, wouldn't the rest of the world like it, too?

Open sourced TensorFlow on Monday, Nov. 9th 2015

- Flexible Apache 2.0 licensing
- Distributed implementation released with TensorFlow 0.8

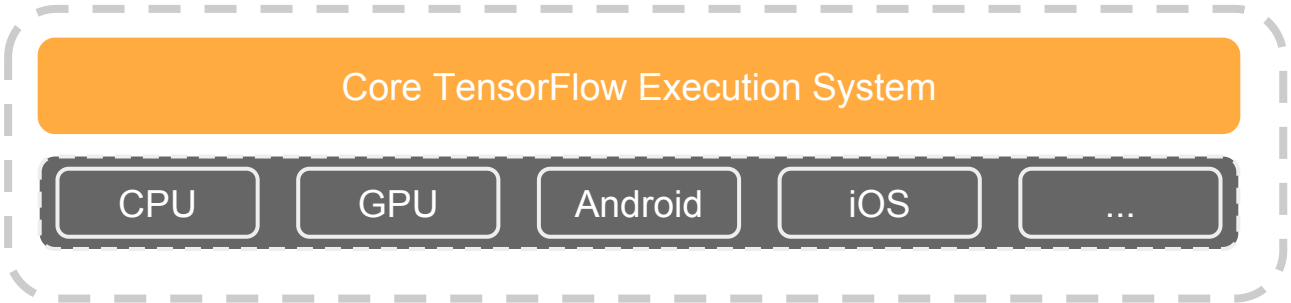
http://tensorflow.org/
https://github.com/tensorflow

Outline

- TensorFlow Overview
- Model’s Representation in TensorFlow: Graph
  - Introduction to graph
  - Its placement and execution
- Notebooks
  - Introduction: tensors, graphs, sessions, optimization
  - Dense features: MNIST
  - Sparse features: income prediction
  - Convolutional networks
- Performance Boost with Large-scale Distributed Systems
  - Model parallelism
  - Data parallelism
- Interesting Projects from TensorFlow Community

Overview

- Core in C++
- Supports different backends.



Runs on Variety of Platforms

Automatically runs models on range of platforms:

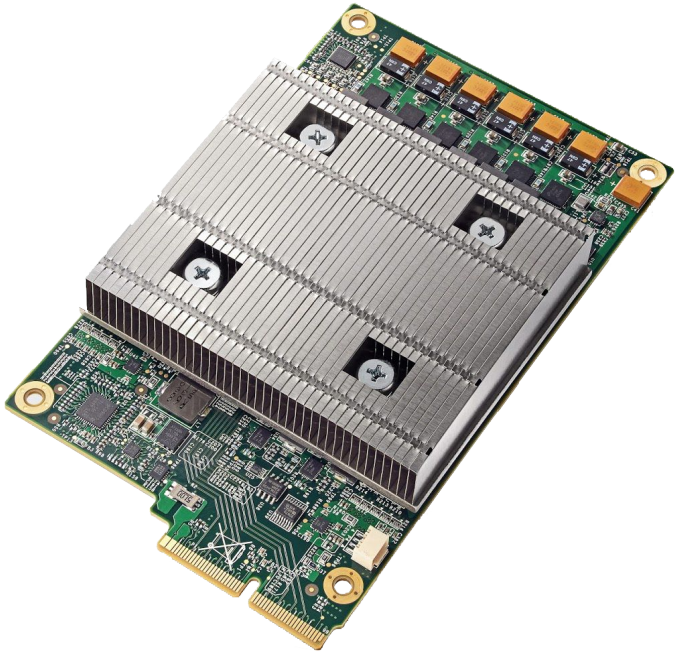
from **phones** ...

to **single machines** (CPU and/or GPUs) …

to **distributed systems** of many 100s of GPU cards

Tensor Processing Unit (TPU)

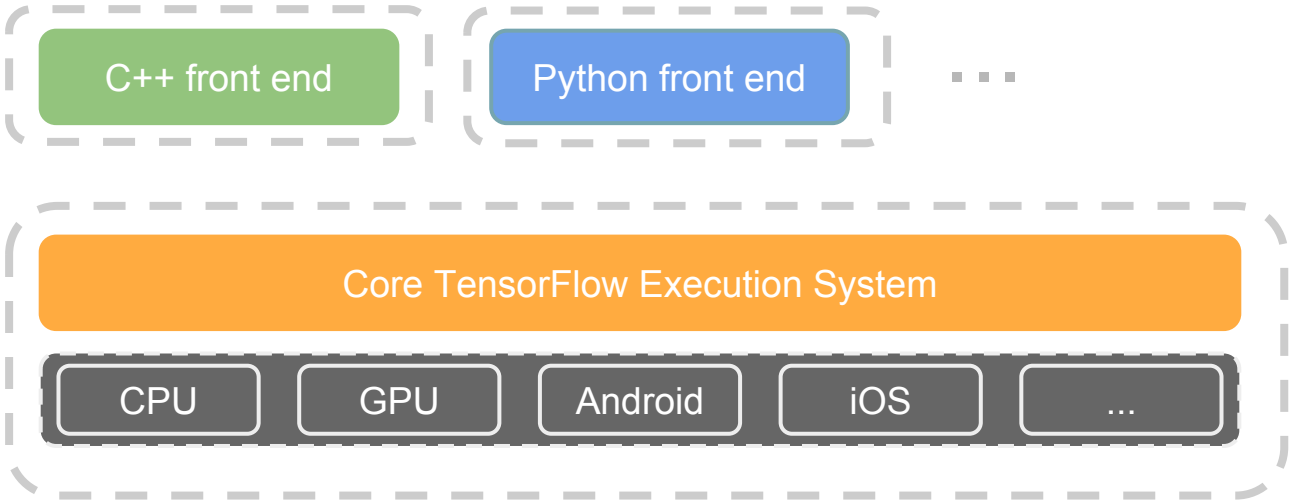
Custom machine learning ASIC



In production use for >14 months: used on every search query, used for AlphaGo match, ...

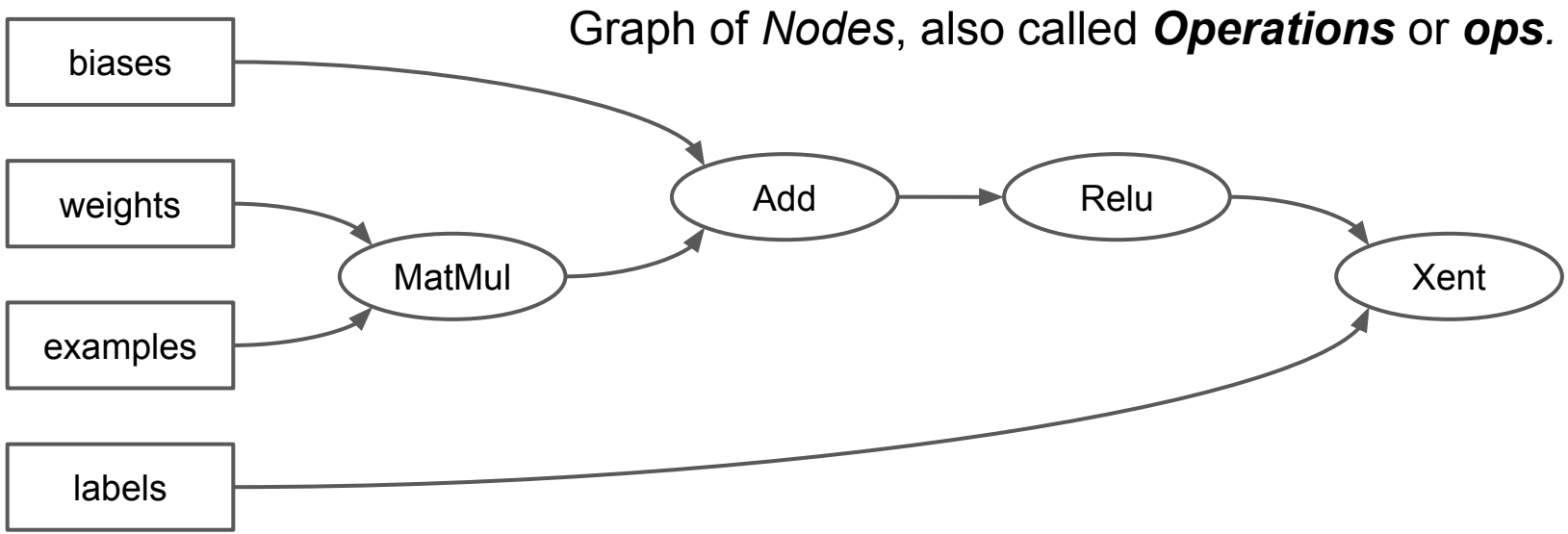
Front Ends

- Different front ends for specifying/driving the computation
  - Python and C++ today, easy to add more



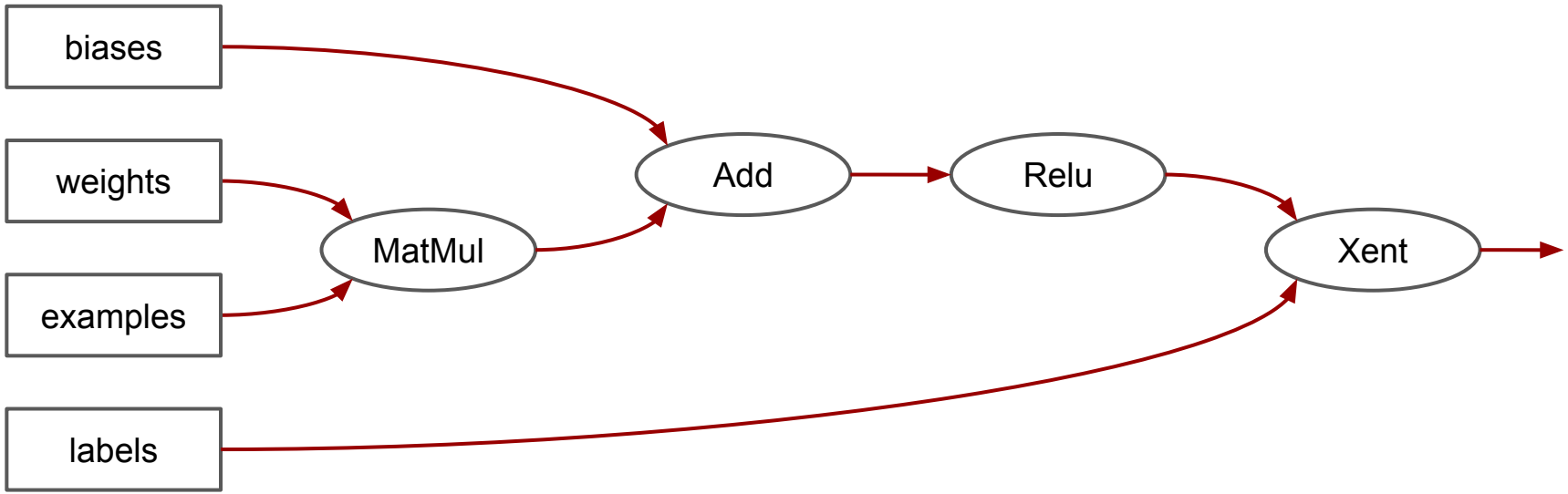
Models Are Graphs in TensorFlow

- Computation is expressed as a dataflow graph



Tensors, They Flow

- Edges are N-dimensional Arrays: Tensors



Python API: Model Construction

- Build a graph computing a neural net inference.

```
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data

mnist = input_data.read_data_sets('MNIST_data', one_hot=True)
x = tf.placeholder("float", shape=[None, 784])
W = tf.Variable(tf.zeros([784,10]))
b = tf.Variable(tf.zeros([10]))
y = tf.nn.softmax(tf.matmul(x, W) + b)
```

Python API: Easy Backprop

- Automatically add ops to calculate symbolic gradients of variables w.r.t. loss function.
- Apply these gradients with an optimization algorithm

```
y_ = tf.placeholder(tf.float32, [None, 10])
cross_entropy = -tf.reduce_sum(y_ * tf.log(y))
opt = tf.train.GradientDescentOptimizer(0.01)
train_op = opt.minimize(cross_entropy)
```

Python API: Easy to Run

- Launch the graph and run the training ops in a 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={x: batch_xs, y_: batch_ys})
```

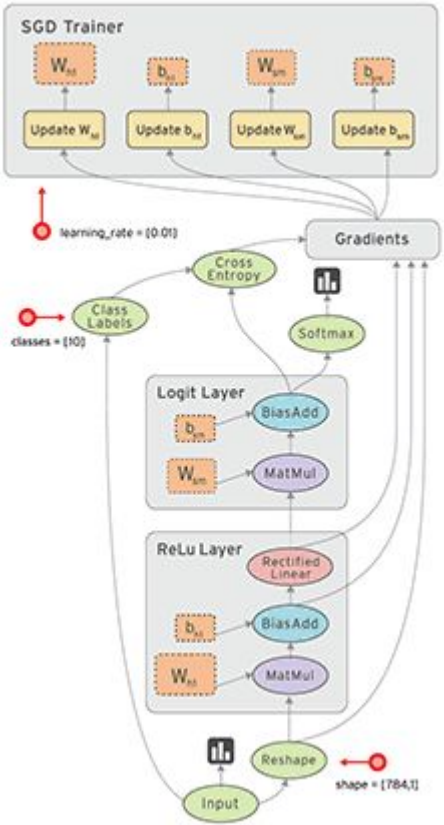
Already Many Ops, Easy to Add More

- Core system defines a number of standard ***operations*** and ***kernels*** (device-specific implementations of operations)
- Easy to define new operators and/or kernels

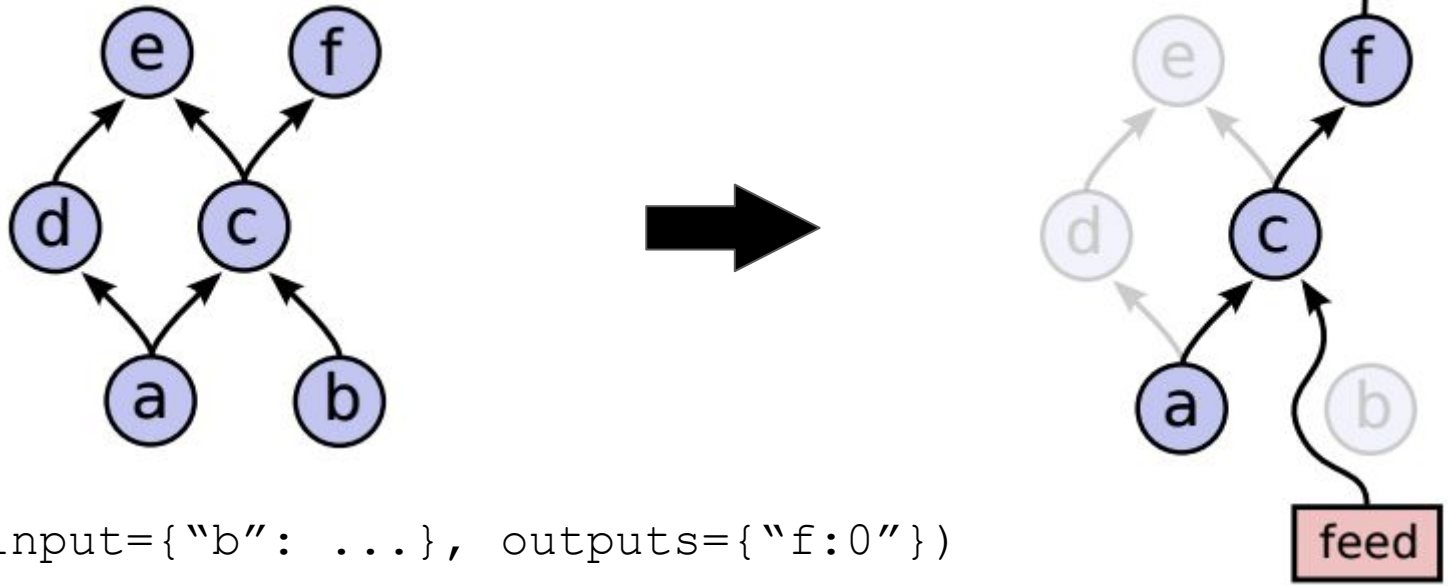
Let’s move to the first notebook!

How Is Graph Executed:

- TensorFlow computes set of Ops that must run to compute the outputs
- Ops execute, in parallel, as soon as their inputs are available
- In practice often very complex with 100s or 1000s of nodes and edges



Feeding and Fetching

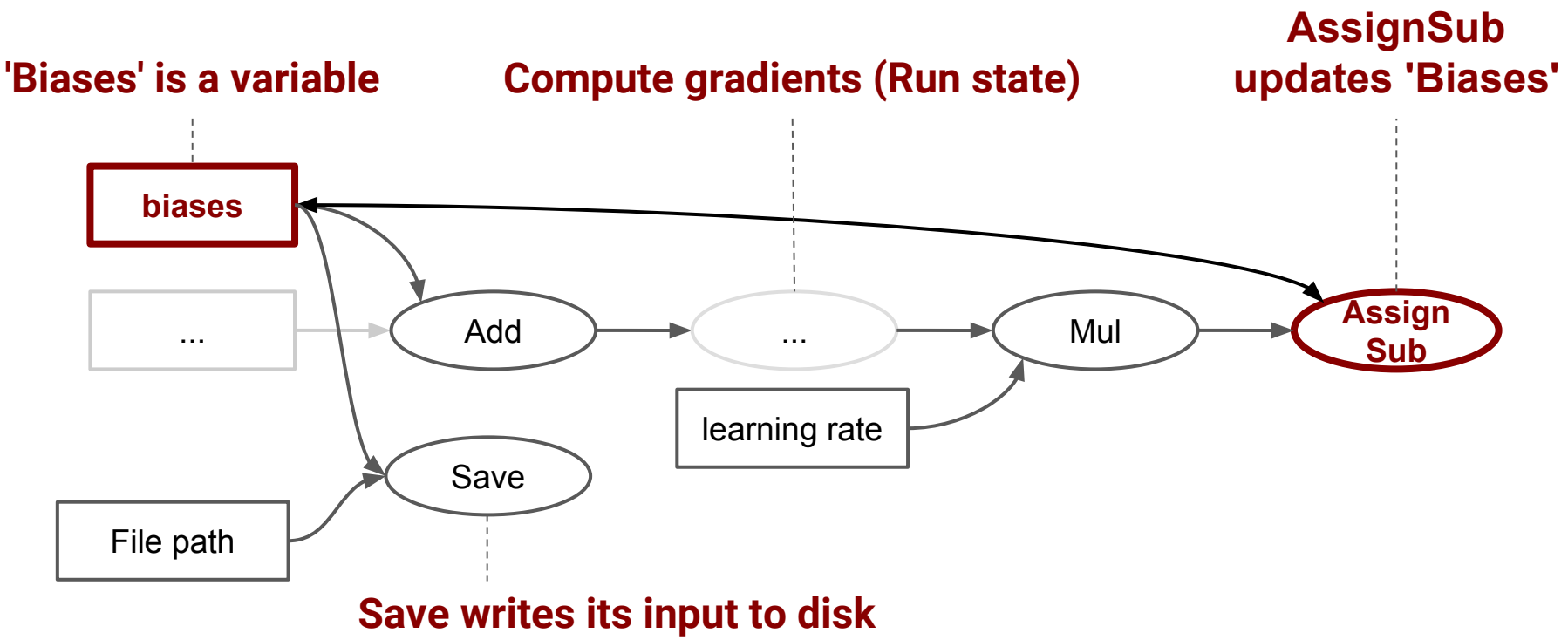



```
Run(input={"b": ...}, outputs={"f:0"})
```

Where Are My Tensors? (Allocation and Duration)

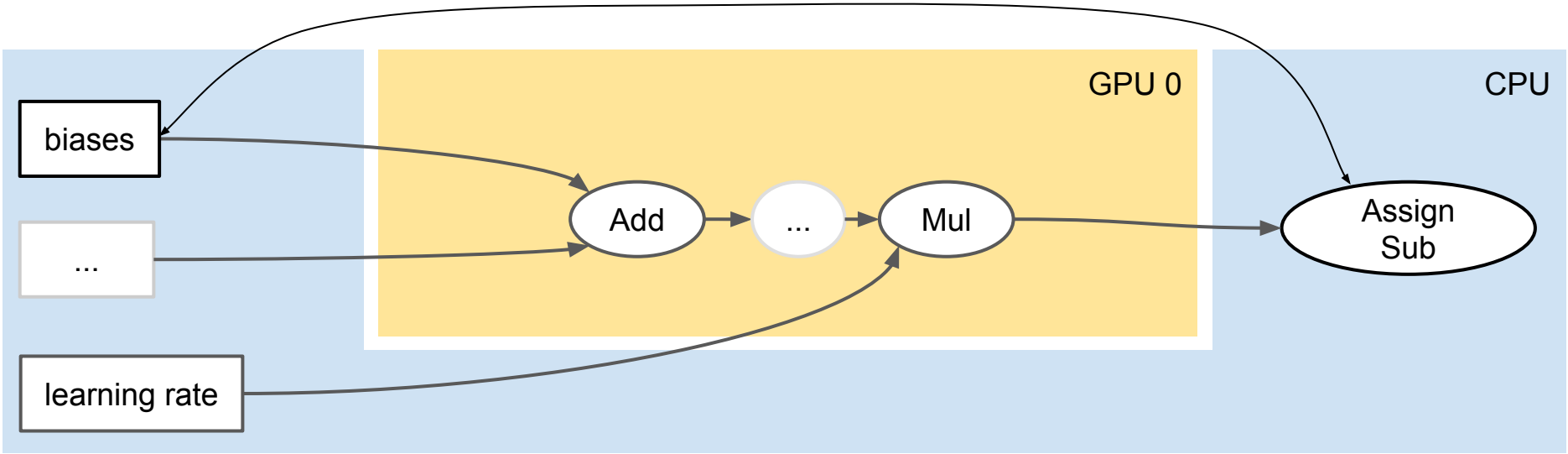
- **Run State**
  - Duration: *Run* call
  - Tensors allocated for ops output, freed after all consuming ops have run
- **Session State**
  - Duration: Session
  - Tensors held by special ops: Variable, Queue
- **Persistent State**
  - Duration: On disk
  - Read/Written by special ops: Save, Restore

Variables: Used for Trained Parameters



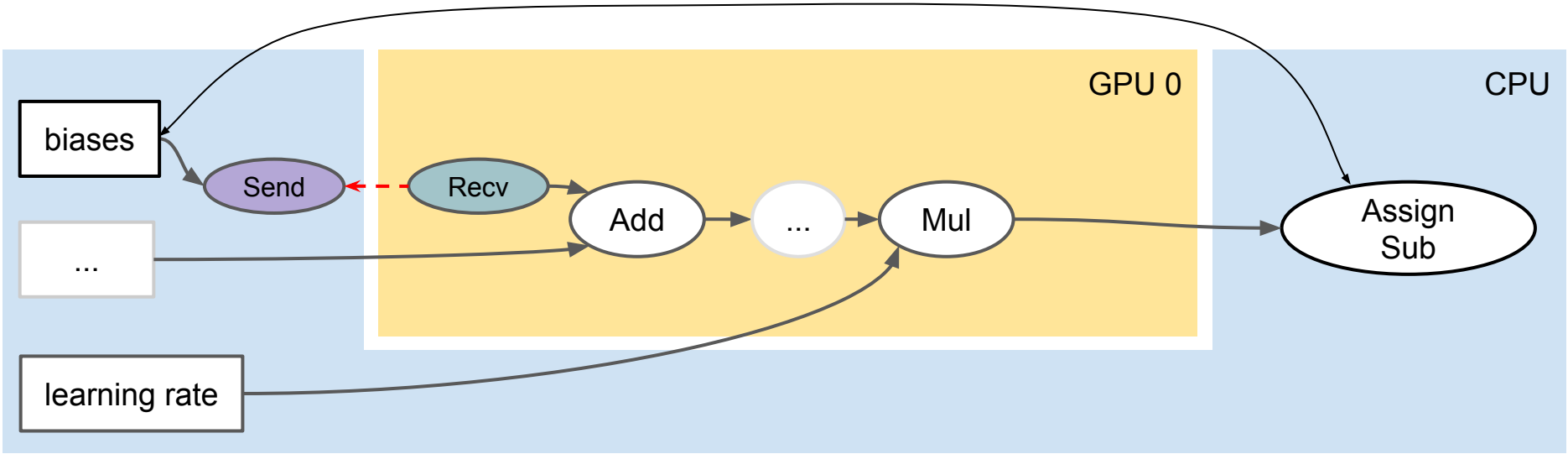
Where Are My Ops? (Placement with >1 Devices)

- Ops are assigned to devices for execution



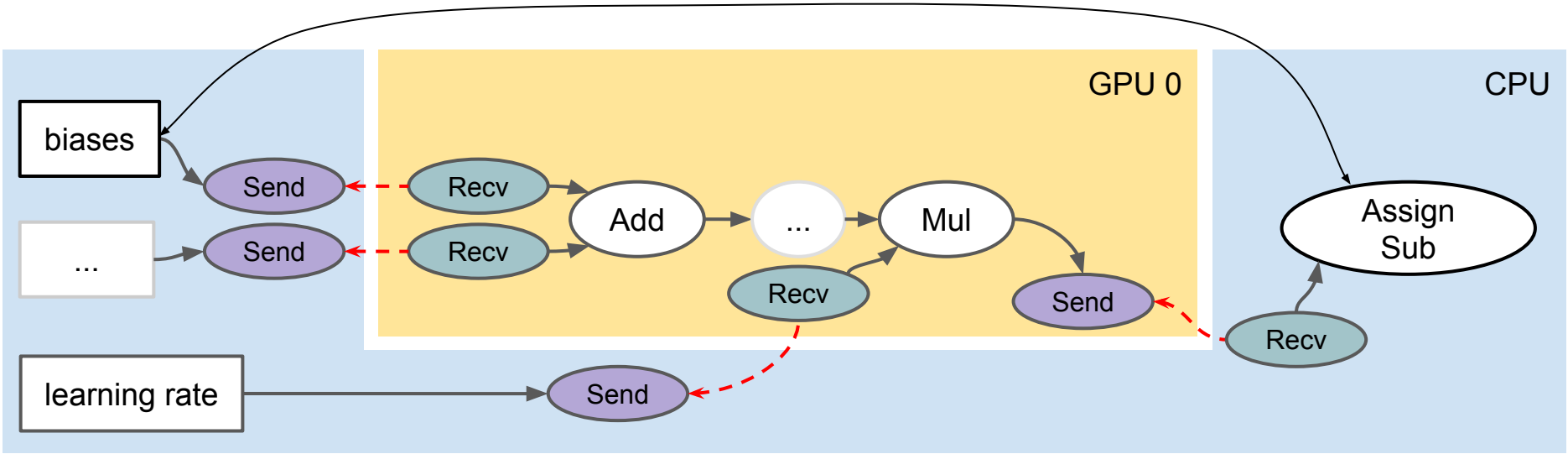
Ops Communicate Among Devices

- TensorFlow inserts Send/Recv Ops to transport tensors across devices



Ops Communicate Among Devices

- Recv ops pull data from Send ops

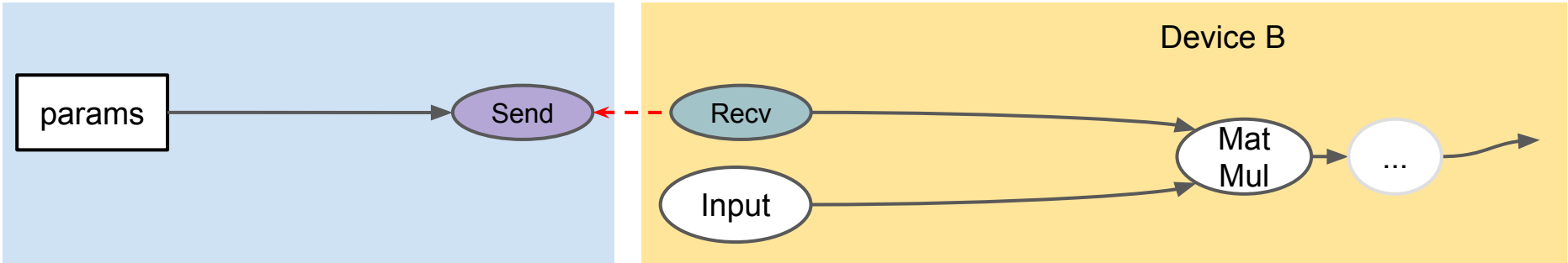


Send and Receive Implementations

- Different implementations depending on source/dest devices
- e.g. GPUs on same machine: **local GPU → GPU copy**
- e.g. CPUs on different machines: **cross-machine RPC**
- e.g. GPUs on different machines: **RDMA or RPC**

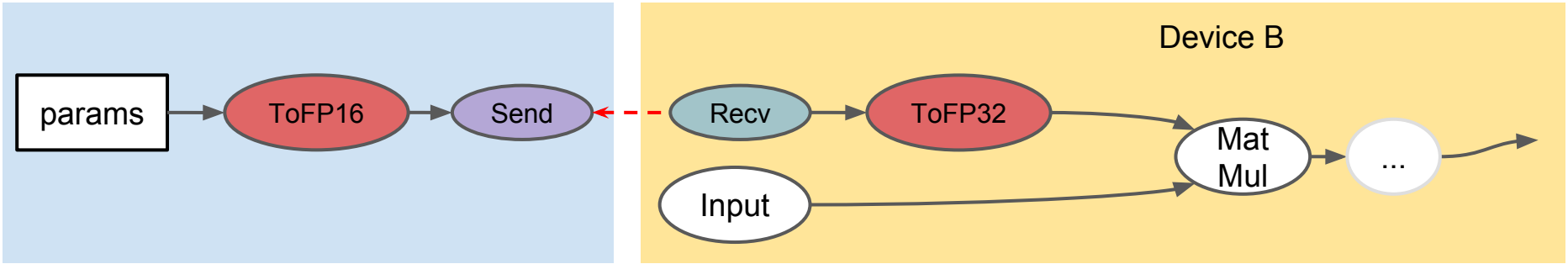
Optimization on Communication

- Transfer big tensors could be slow



Optimization on Communication

- Neural net training very tolerant of reduced precision
- e.g. drop precision to 16 bits across network
  - Not IEEE fp16, just mask off the last 16bits.



Placement is Tricky and Interesting

- Users can provide hints/constraints: *“place on gpu 2”* or *“place on task 7”*
- Want fastest graph execution possible:
  - Subject to memory limitations of devices
  - Subject to user-provided hints/constraints

Cool thought not yet explored: Use a neural net, possibly with some reinforcement learning, to help decide on node placement

Performance Boost with Large-scale Distributed Systems

Experiment Turnaround Time and Research Productivity

- **Minutes, Hours**:
  - **Interactive research! Instant gratification!**
- **1-4 days**
  - Tolerable
  - Interactivity replaced by running many experiments in parallel
- **1-4 weeks**
  - High value experiments only
  - Progress stalls
- **>1 month**
  - Don’t even try


Google | google.com/datacenters

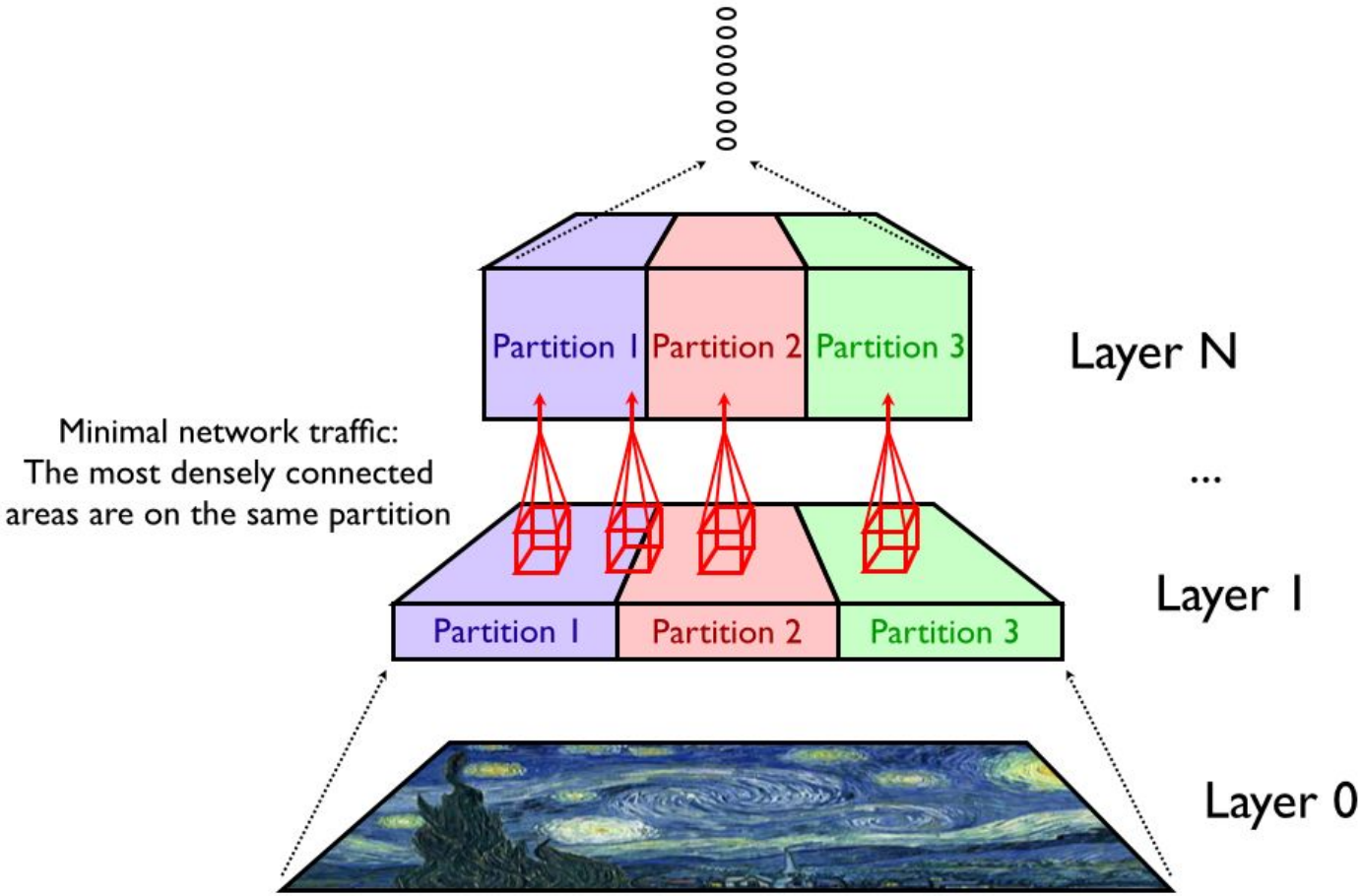
Scalability is Key to Faster Training

- How do you do machine learning at scale?
  - Model parallelism
  - Data parallelism
- How does TensorFlow make distributed training easy?

Model Parallelism

- Best way to accelerate training: **decrease the step time**
- Many models have lots of inherent parallelism
- Minimizing communication according to computation
  - local connectivity (as found in CNNs)
  - towers with little or no connectivity between them (e.g. AlexNet)

Model Parallelism: Partition model across machines



Exploiting Model Parallelism

On a single core: Instruction parallelism (SIMD). Just free.

Across cores: thread parallelism. Almost free.

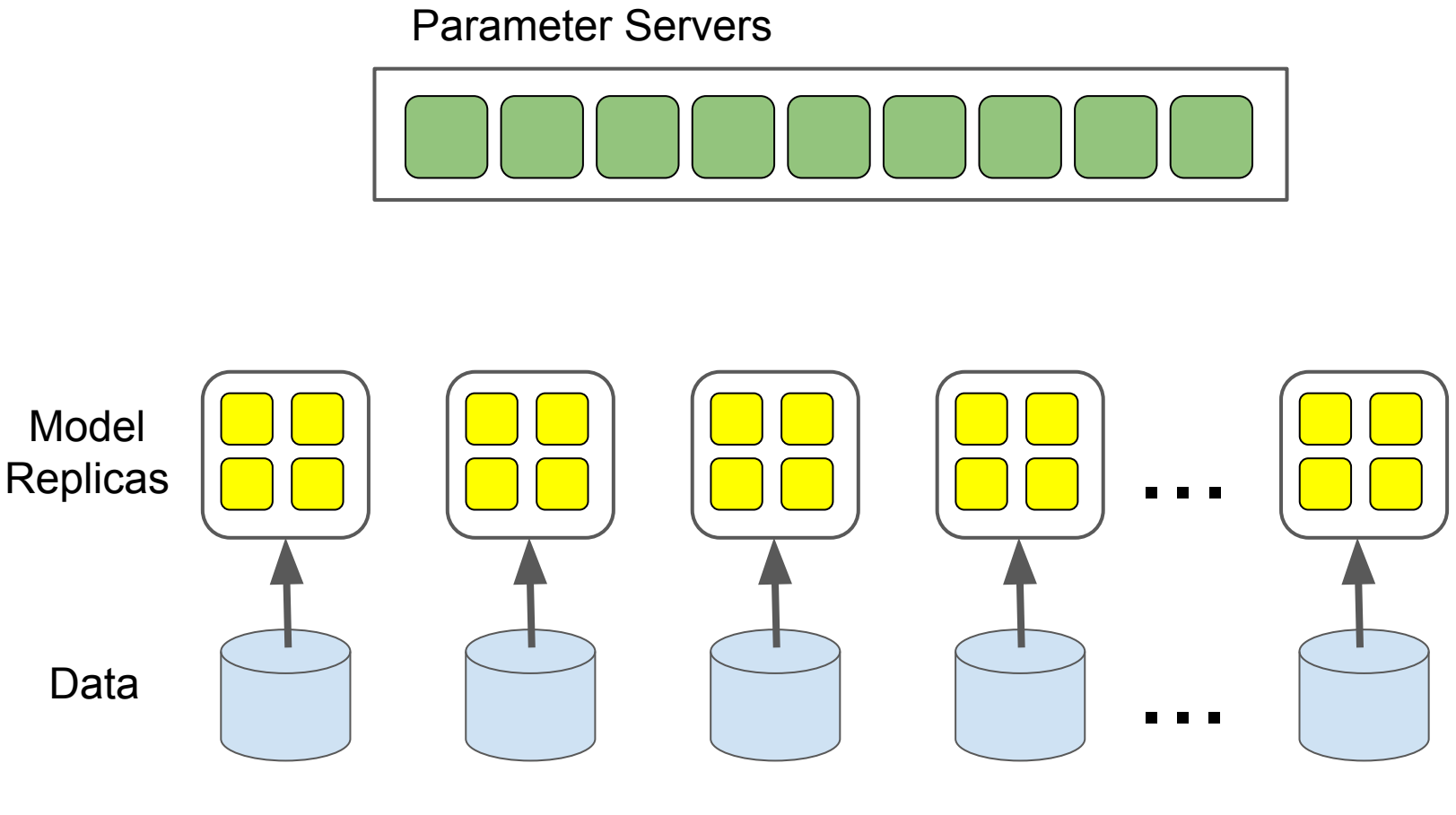
Across devices: for GPUs, often limited by PCIe bandwidth.

Across machines: limited by network bandwidth / latency

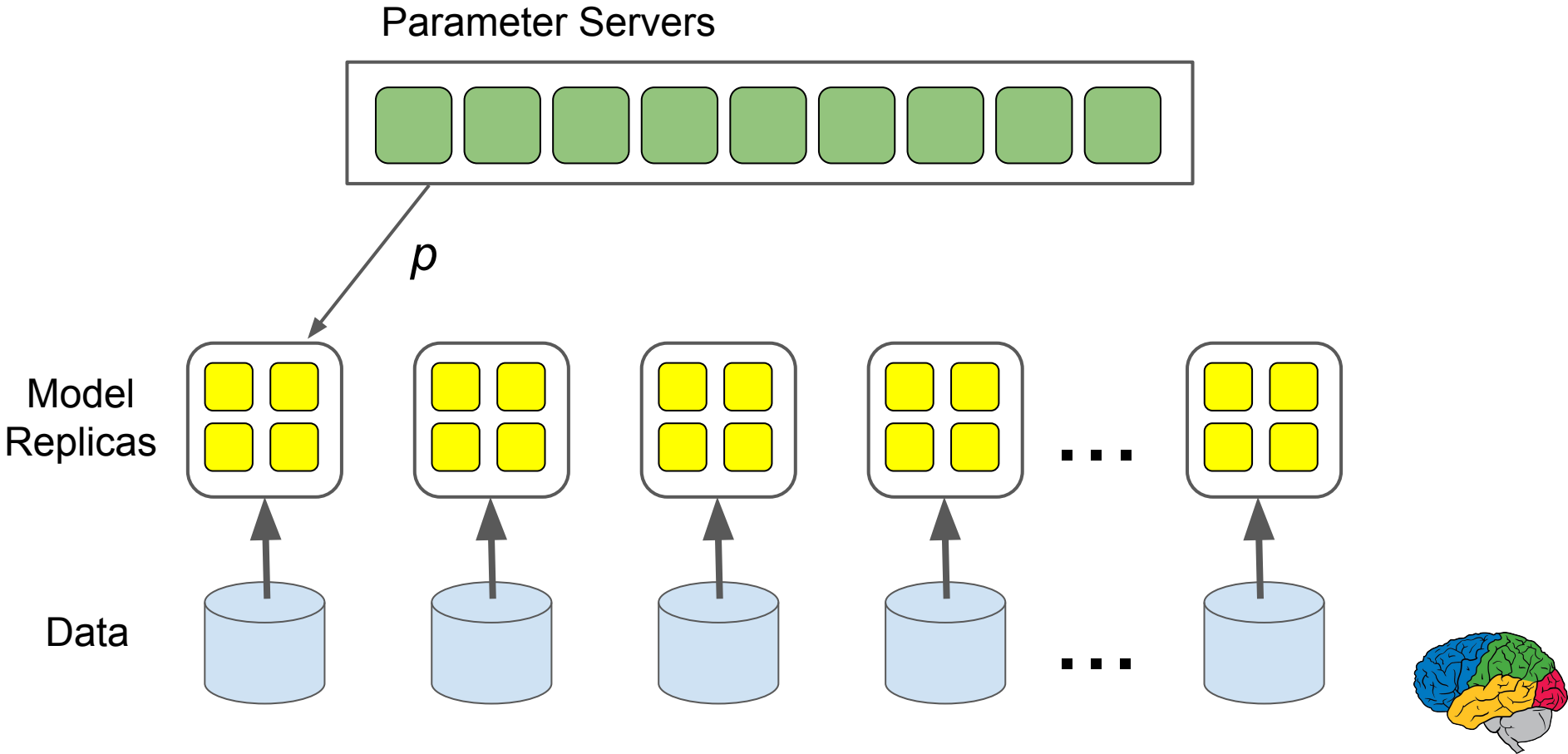
Data Parallelism

- 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

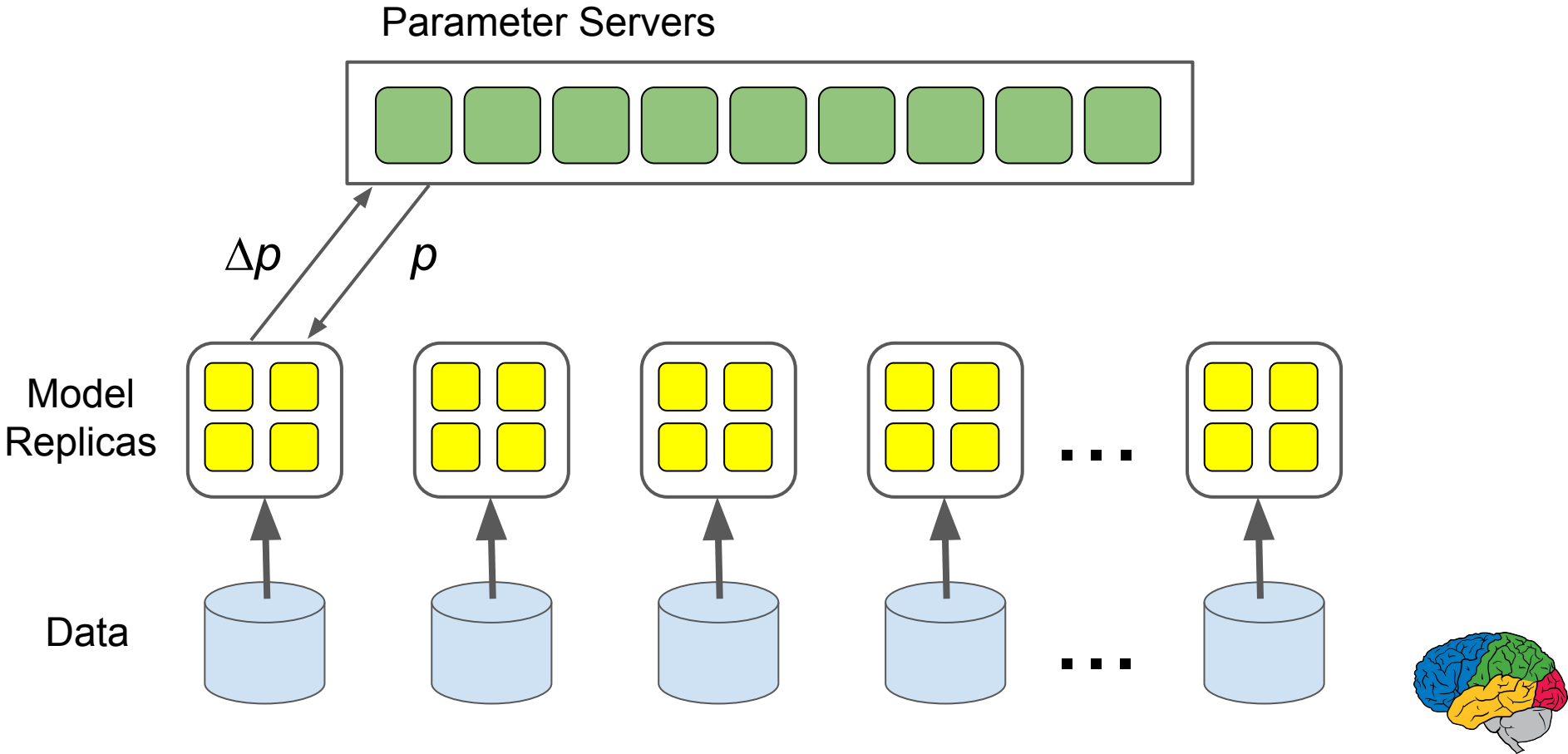
Data Parallelism



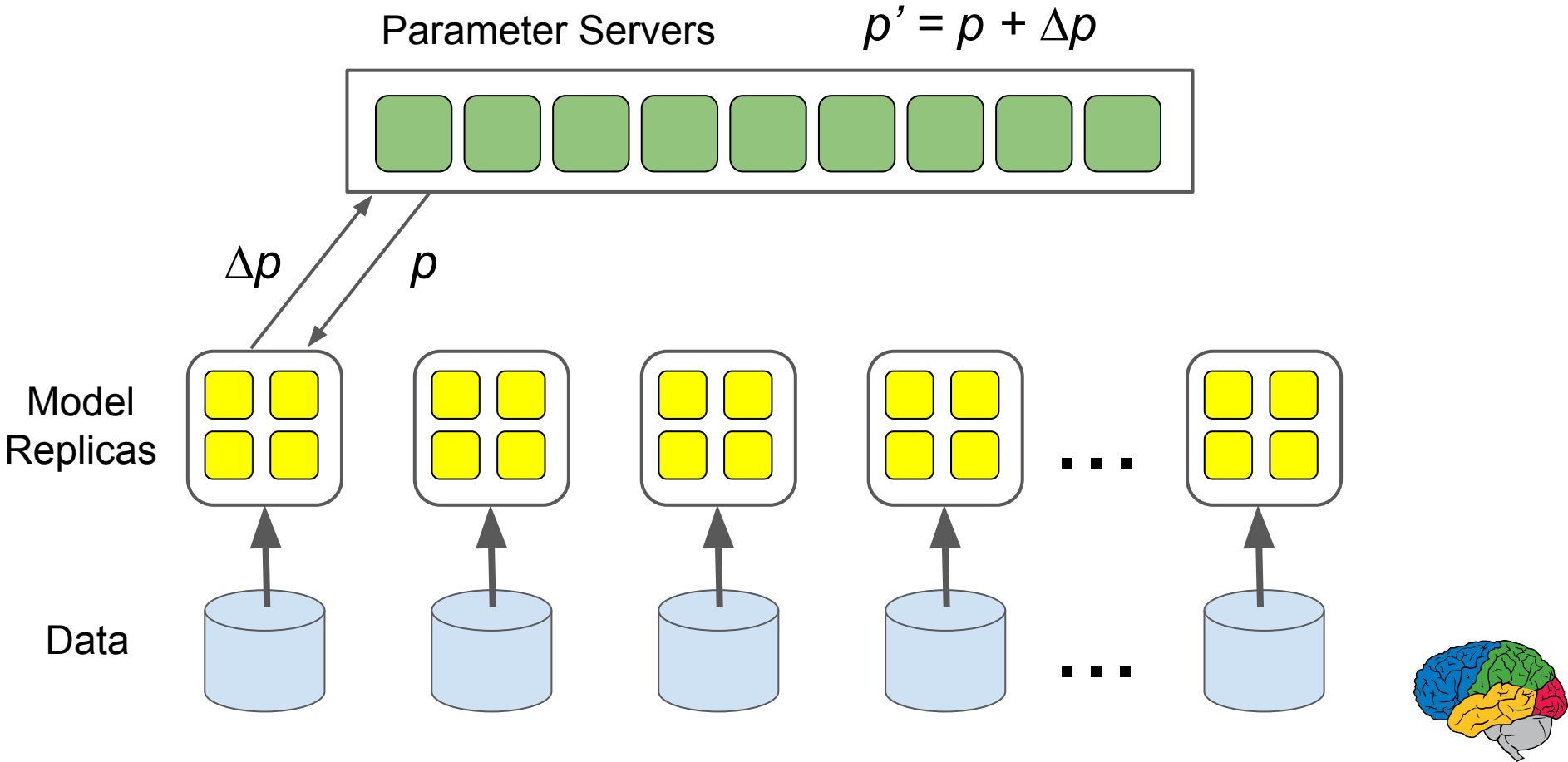
Data Parallelism



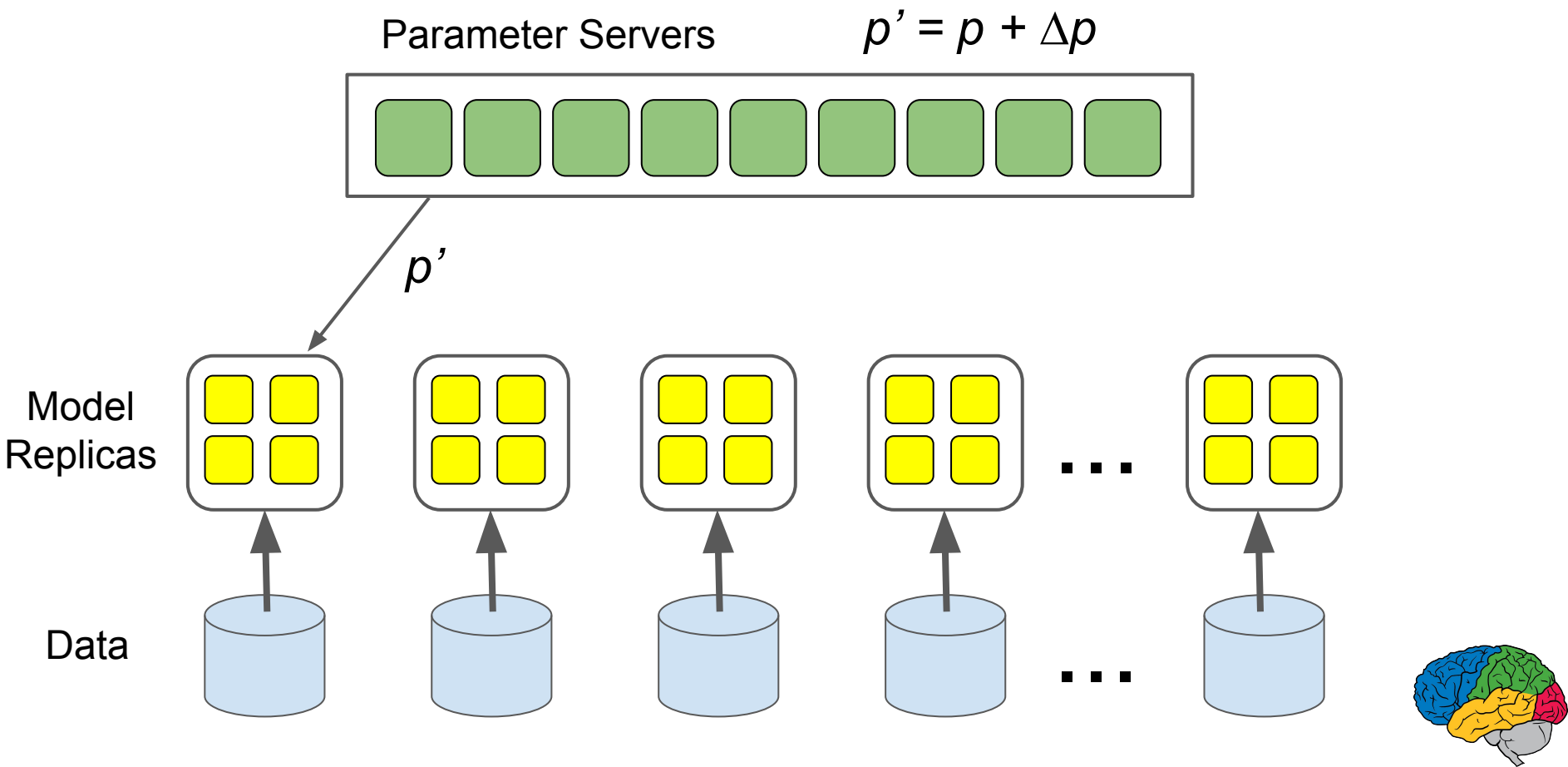
Data Parallelism



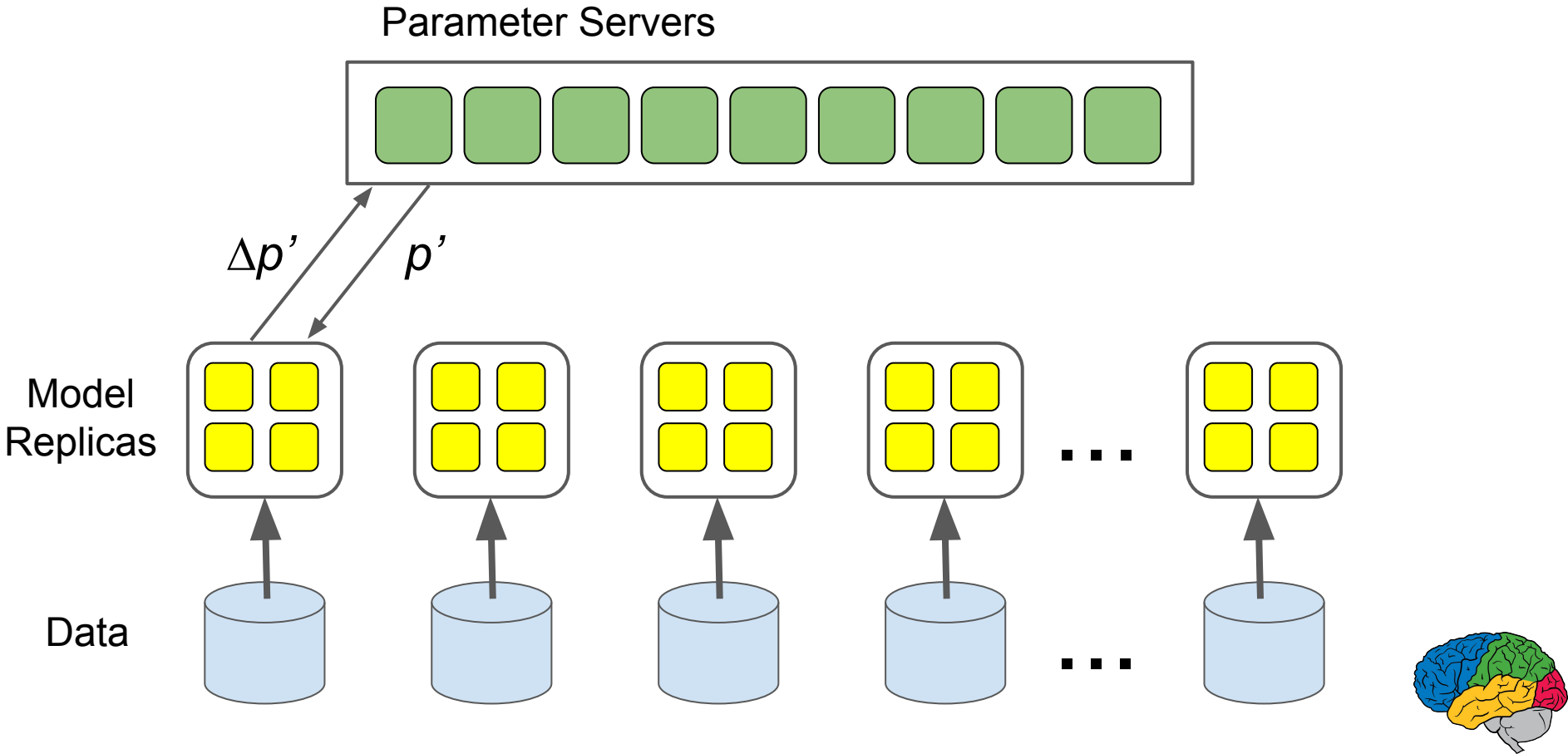
Data Parallelism



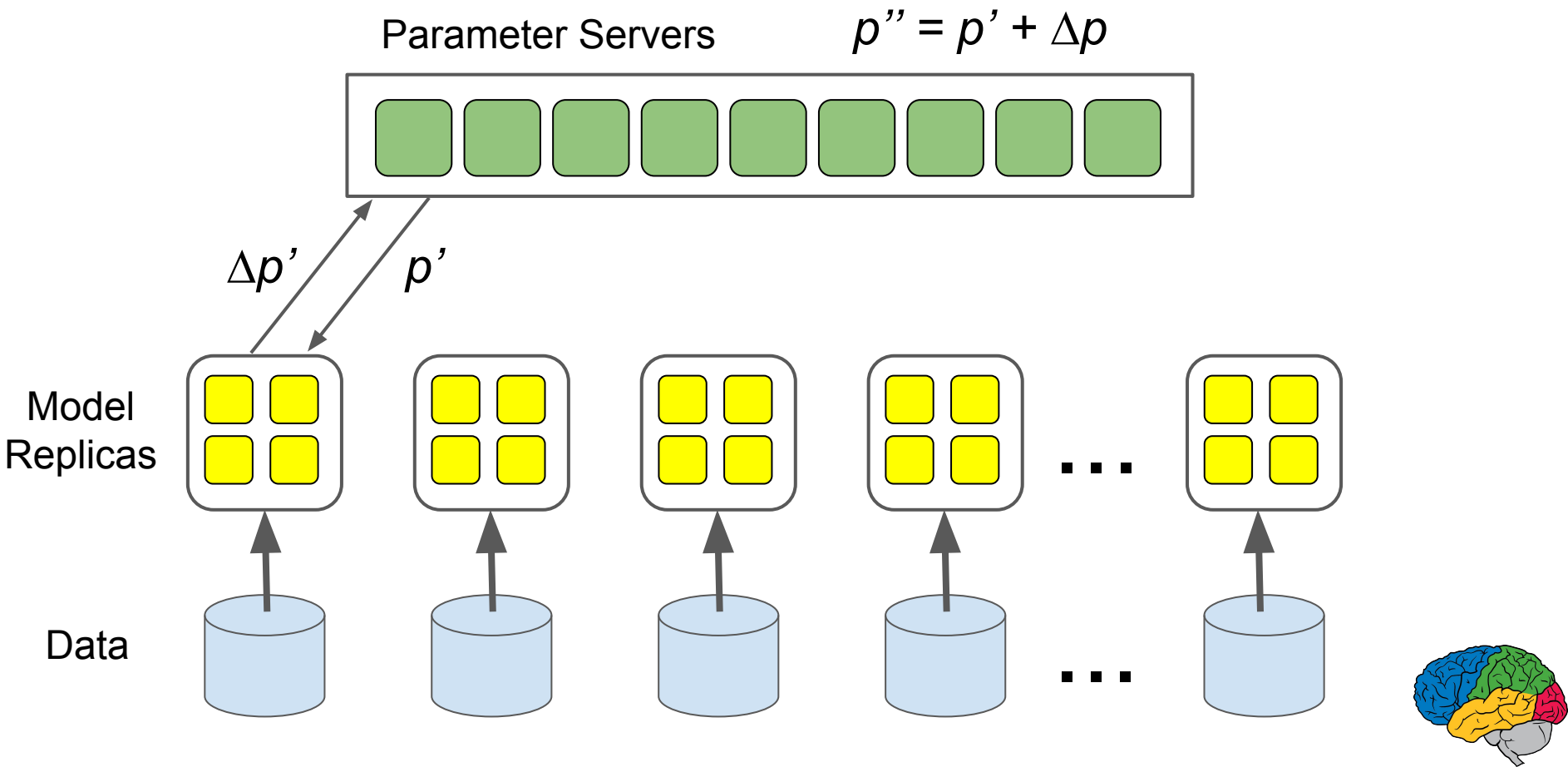
Data Parallelism



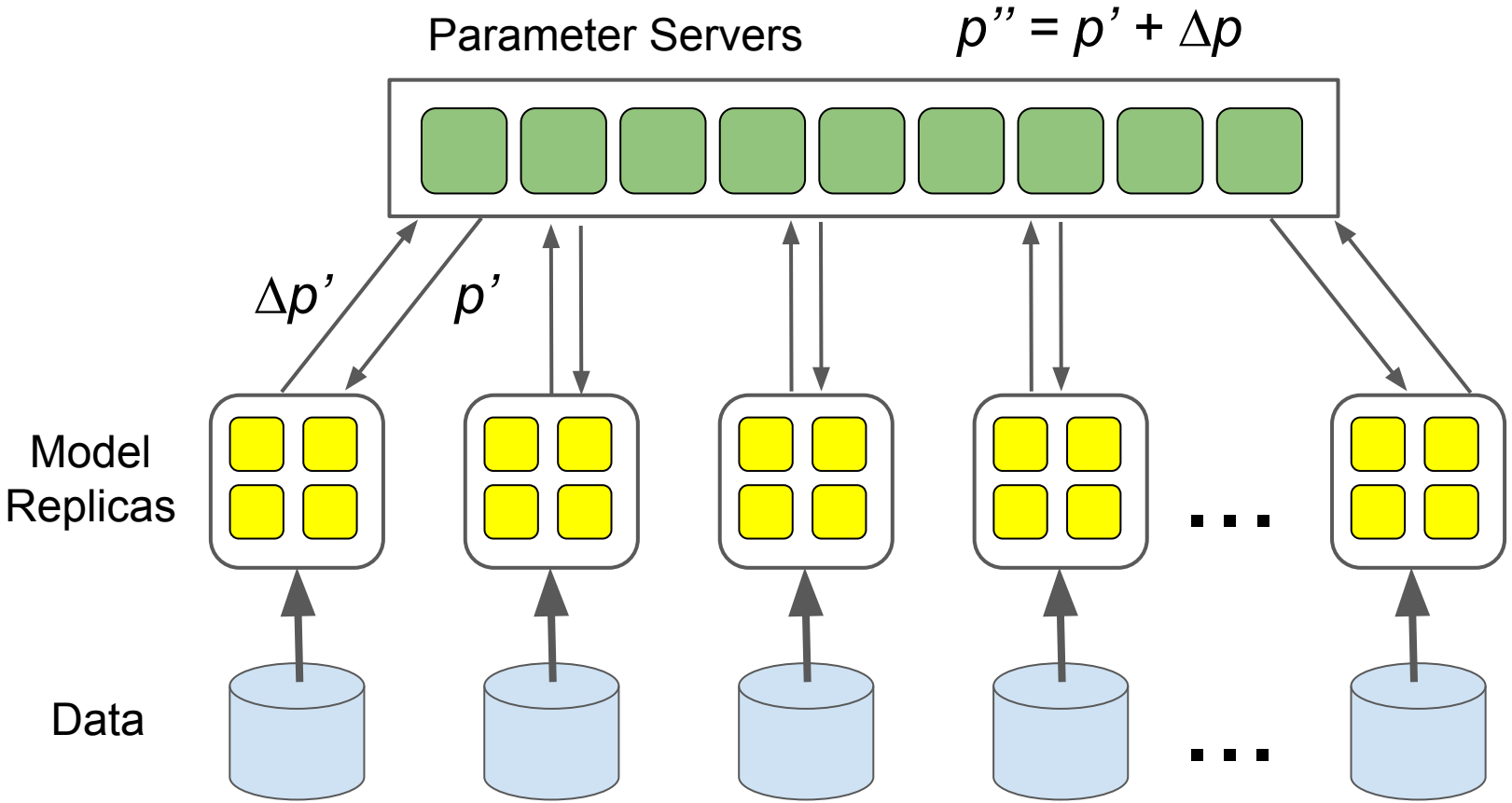
Data Parallelism



Data Parallelism



Data Parallelism



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

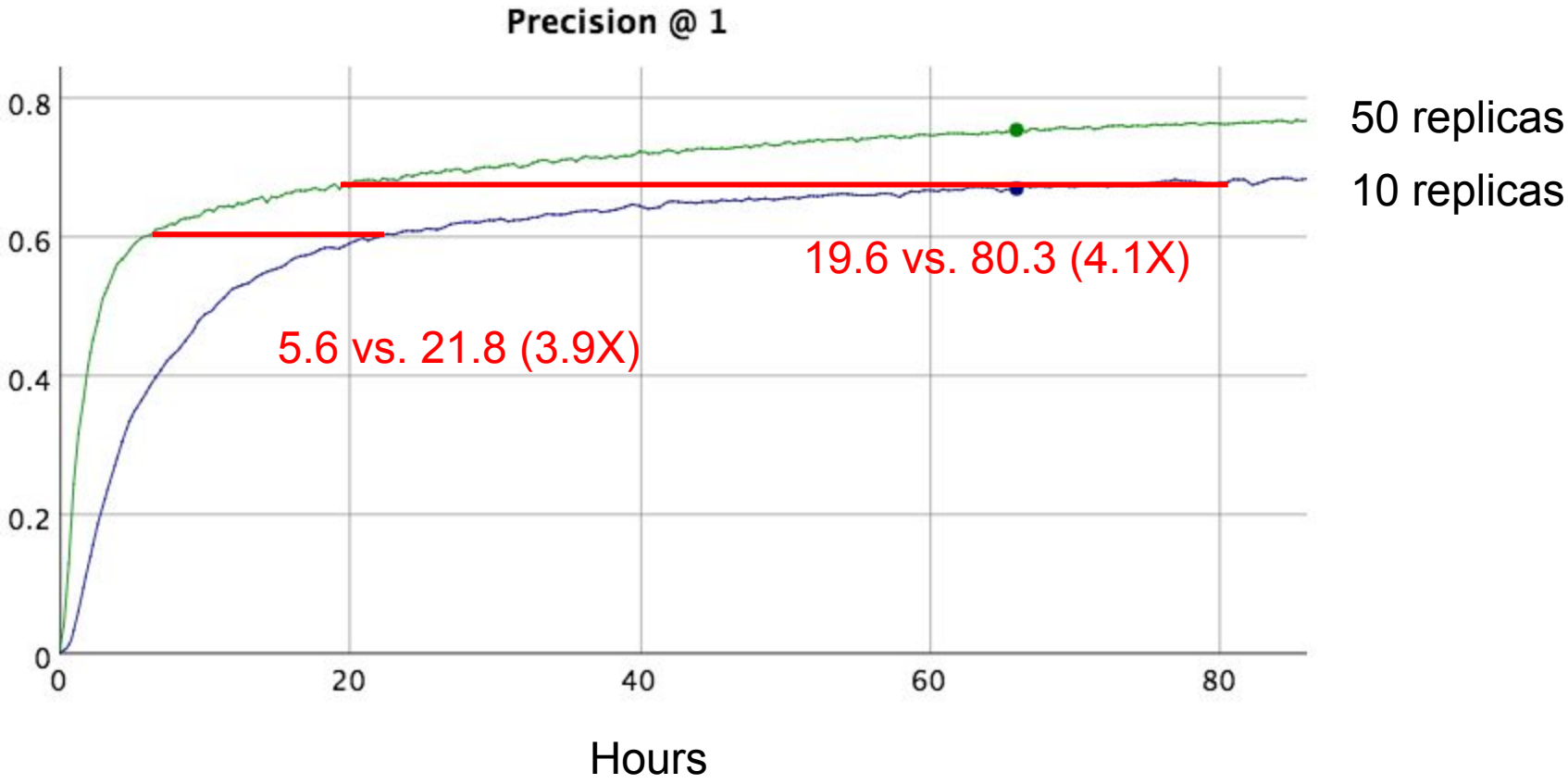
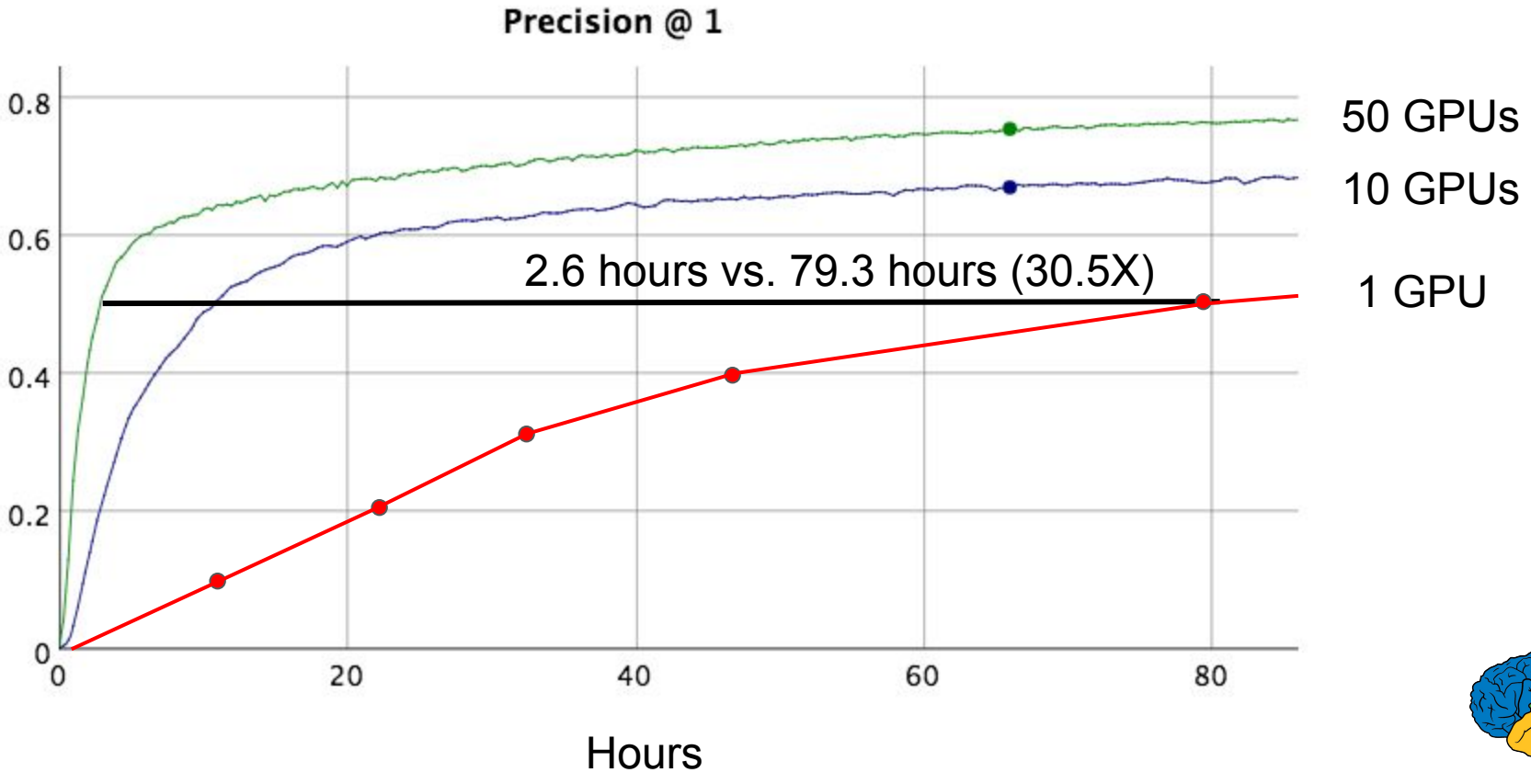


Image Model Training Time: VS 1 GPU



Easy to Go Distributed in TF

Trivial to express both model parallelism as well as data parallelism

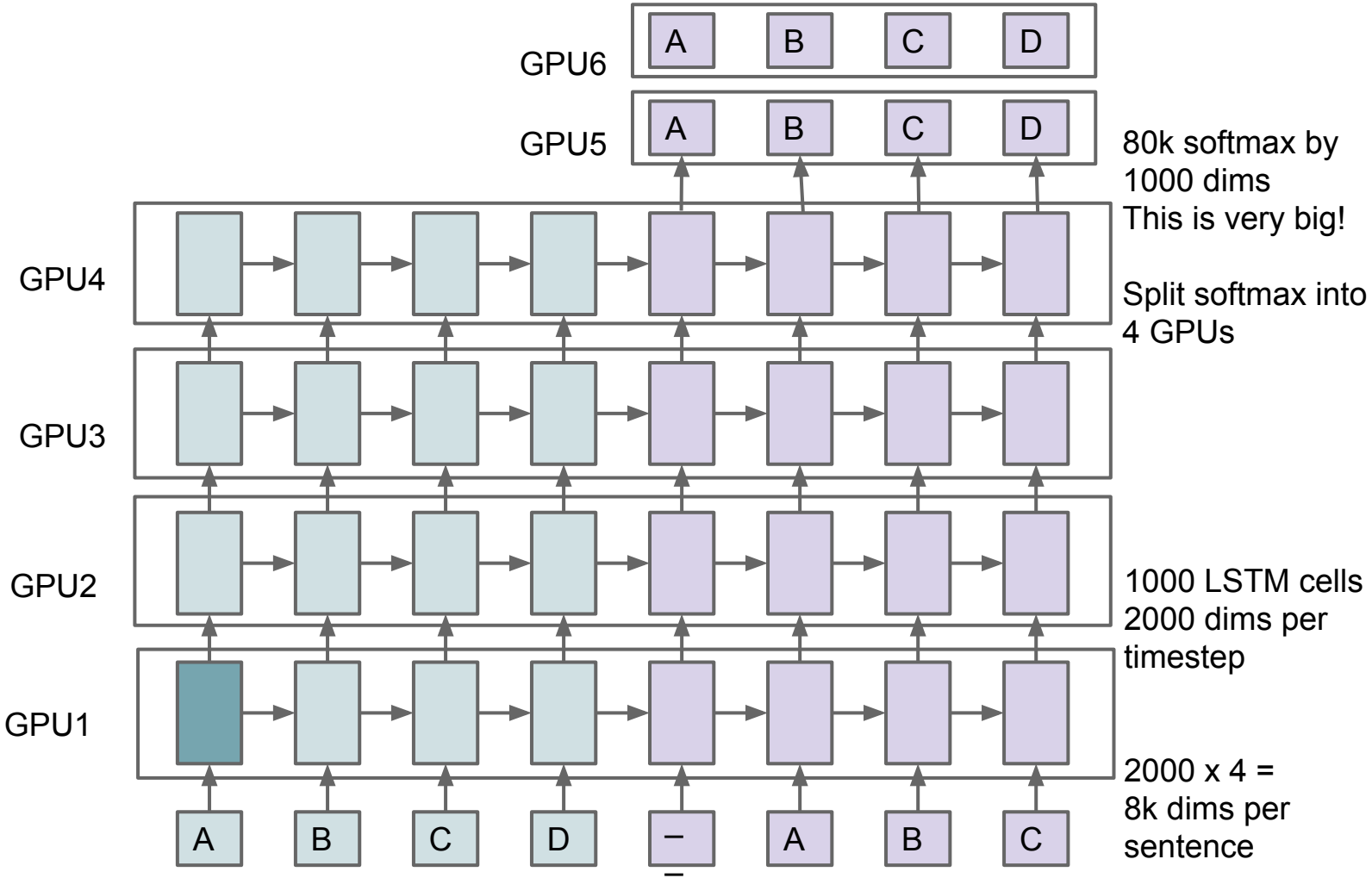
- Very minimal changes to single device model code

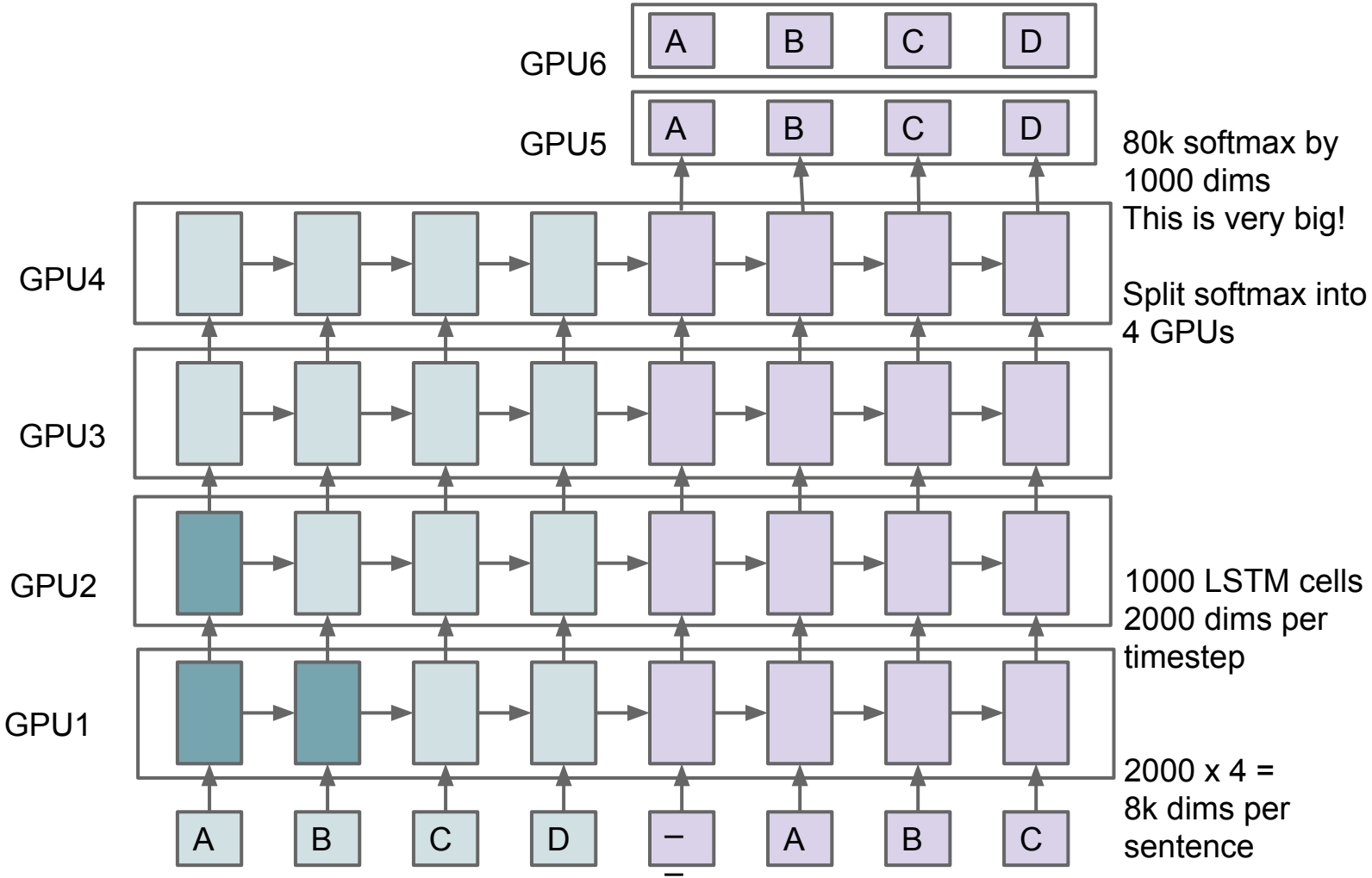
Single Device code (LSTM)

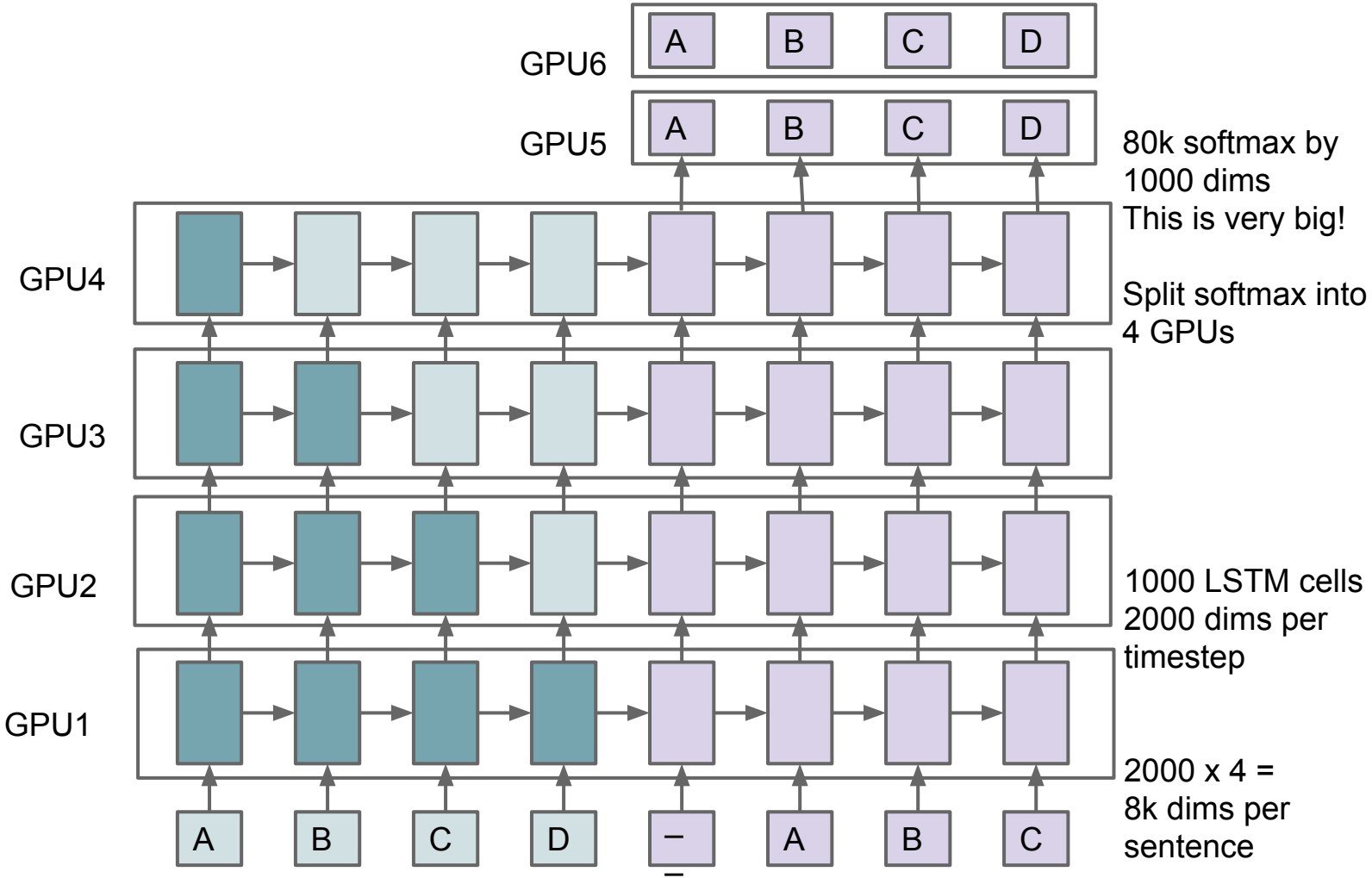
```
for i in range(8):
  for d in range(4): # d is depth
      input = x[i] if d is 0 else m[d-1]
      m[d], c[d] = LSTMCell(input, mprev[d], cprev[d])
      mprev[d] = m[d]
      cprev[d] = c[d]
```

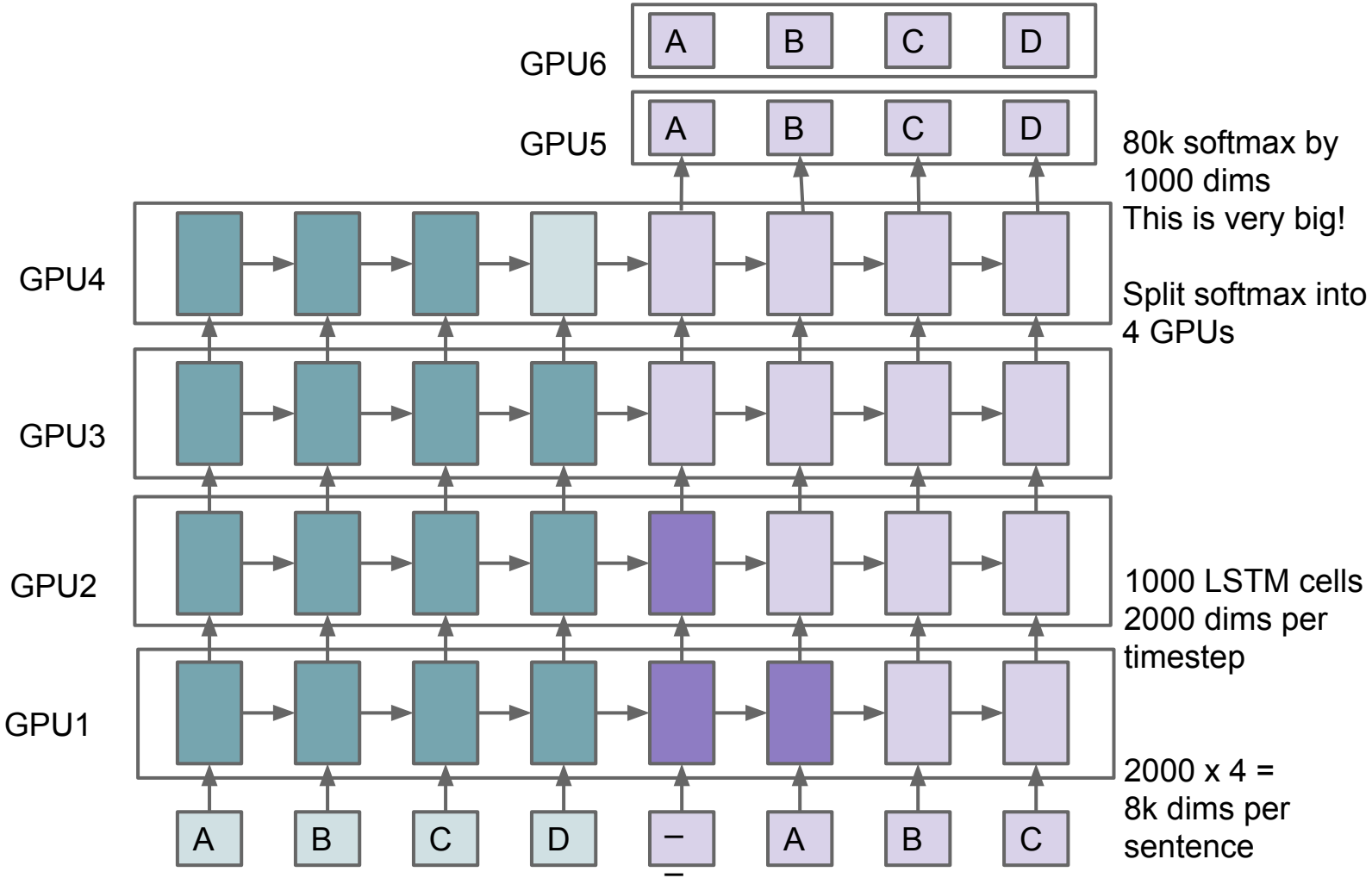
Model Parallelism: 1 line change

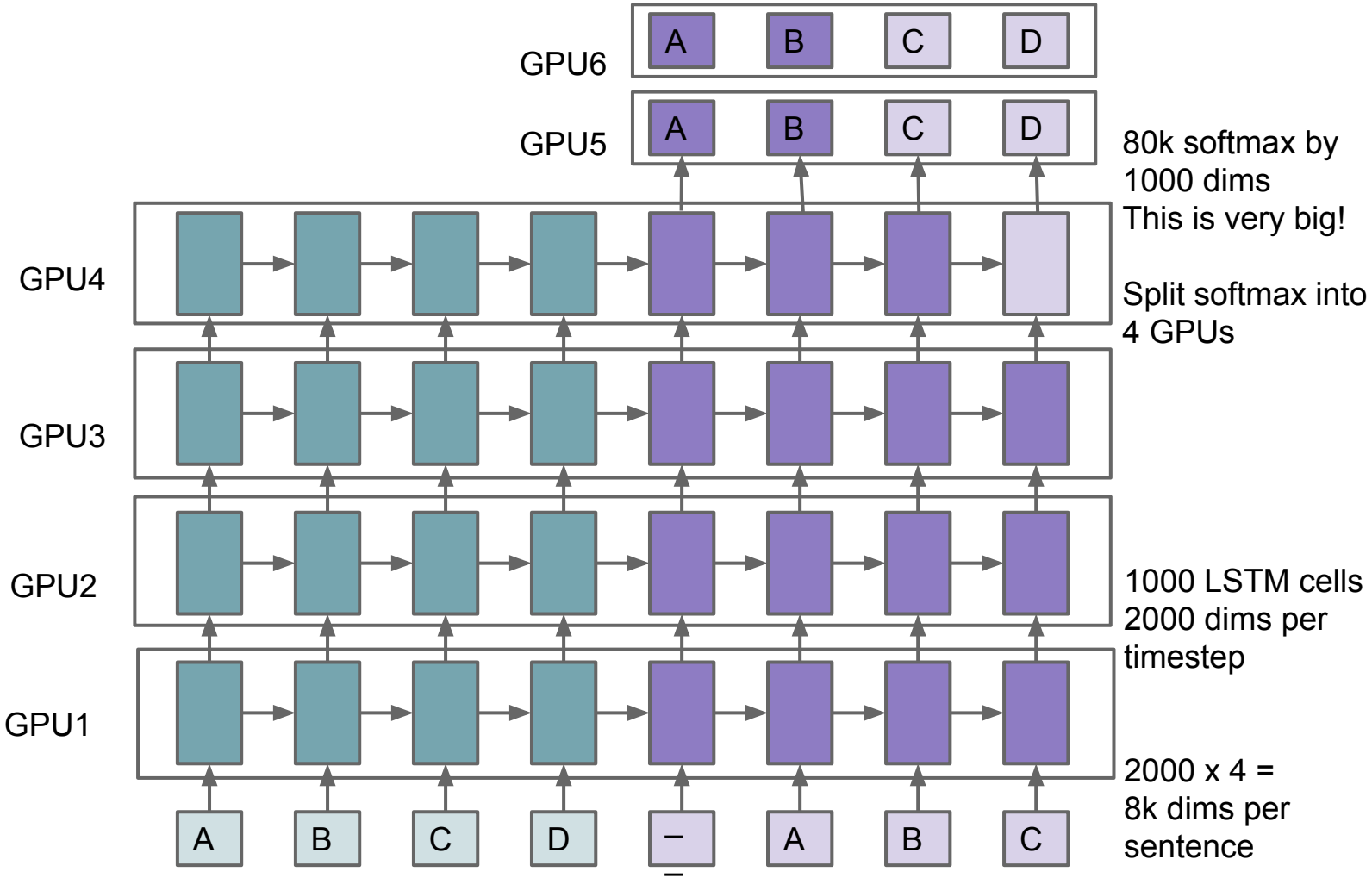
```
for i in range(8):
  for d in range(4): # d is depth
    with tf.device("/gpu:%d" % d):
      input = x[i] if d is 0 else m[d-1]
      m[d], c[d] = LSTMCell(input, mprev[d], cprev[d])
      mprev[d] = m[d]
      cprev[d] = c[d]
```

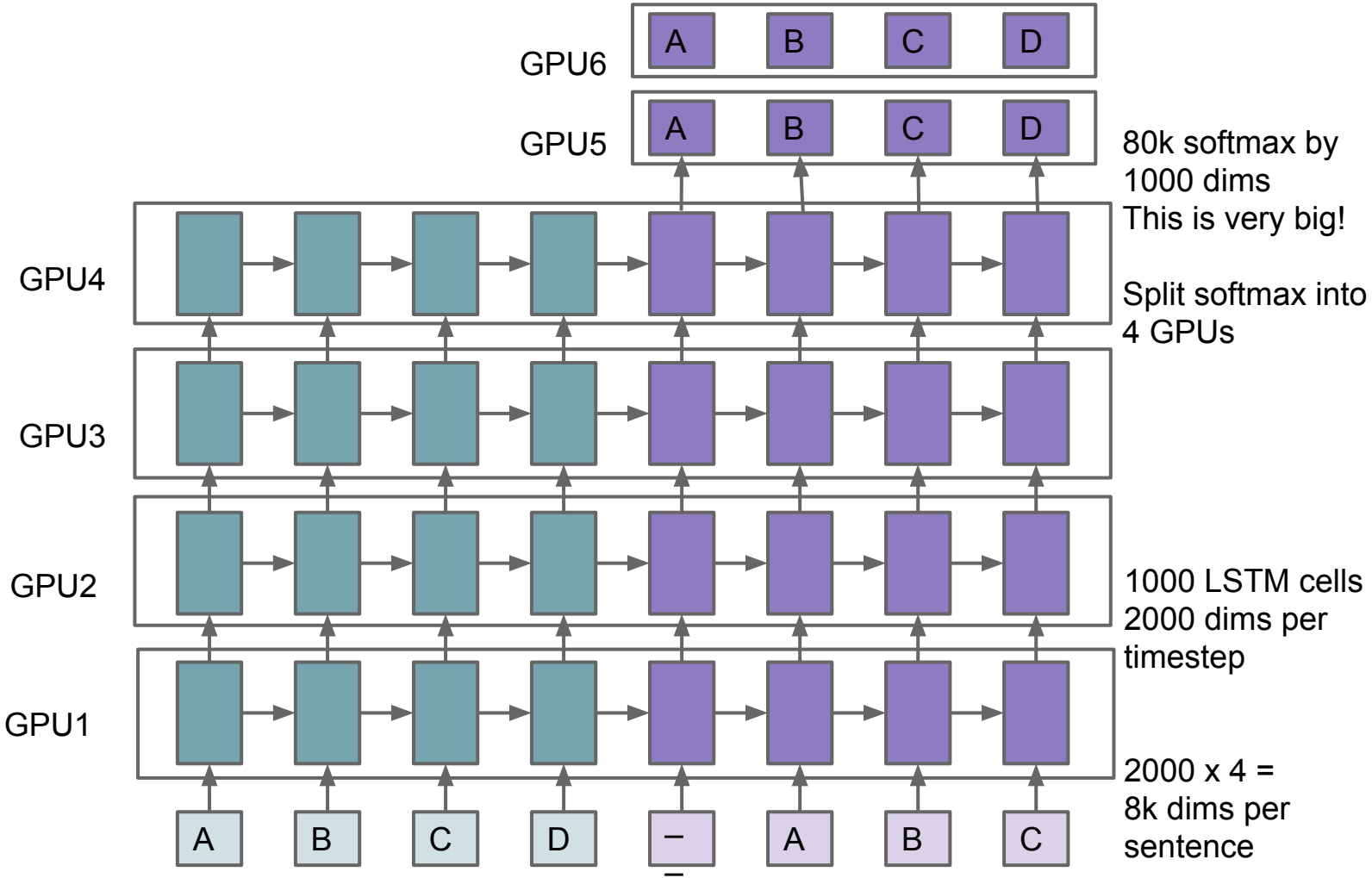

GPU6
A
B
C
D
GPU5
A
B
C
D
80k softmax by
1000 dims
This is very big!
GPU4
Split softmax into
4 GPUs
GPU3
GPU2
1000 LSTM cells
2000 dims per
timestep
GPU1
2000 x 4 =
8k dims per
sentence
A
B
C
D
–
A
B
C


GPU6
A
B
C
D
GPU5
A
B
C
D
80k softmax by
1000 dims
This is very big!
GPU4
Split softmax into
4 GPUs
GPU3
GPU2
1000 LSTM cells
2000 dims per
timestep
GPU1
2000 x 4 =
8k dims per
sentence
A
B
C
D
–
A
B
C


GPU6
A
B
C
D
GPU5
A
B
C
D
80k softmax by
1000 dims
This is very big!
GPU4
Split softmax into
4 GPUs
GPU3
GPU2
1000 LSTM cells
2000 dims per
timestep
GPU1
2000 x 4 =
8k dims per
sentence
A
B
C
D
–
A
B
C


GPU6
A
B
C
D
GPU5
A
B
C
D
80k softmax by
1000 dims
This is very big!
GPU4
Split softmax into
4 GPUs
GPU3
GPU2
1000 LSTM cells
2000 dims per
timestep
GPU1
2000 x 4 =
8k dims per
sentence
A
B
C
D
–
A
B
C


GPU6
A
B
C
D
GPU5
A
B
C
D
80k softmax by
1000 dims
This is very big!
GPU4
Split softmax into
4 GPUs
GPU3
GPU2
1000 LSTM cells
2000 dims per
timestep
GPU1
2000 x 4 =
8k dims per
sentence
A
B
C
D
–
A
B
C


GPU6
A
B
C
D
GPU5
A
B
C
D
80k softmax by
1000 dims
This is very big!
GPU4
Split softmax into
4 GPUs
GPU3
GPU2
1000 LSTM cells
2000 dims per
timestep
GPU1
2000 x 4 =
8k dims per
sentence
A
B
C
D
–
A
B
C

Data Parallelism: Single Device code

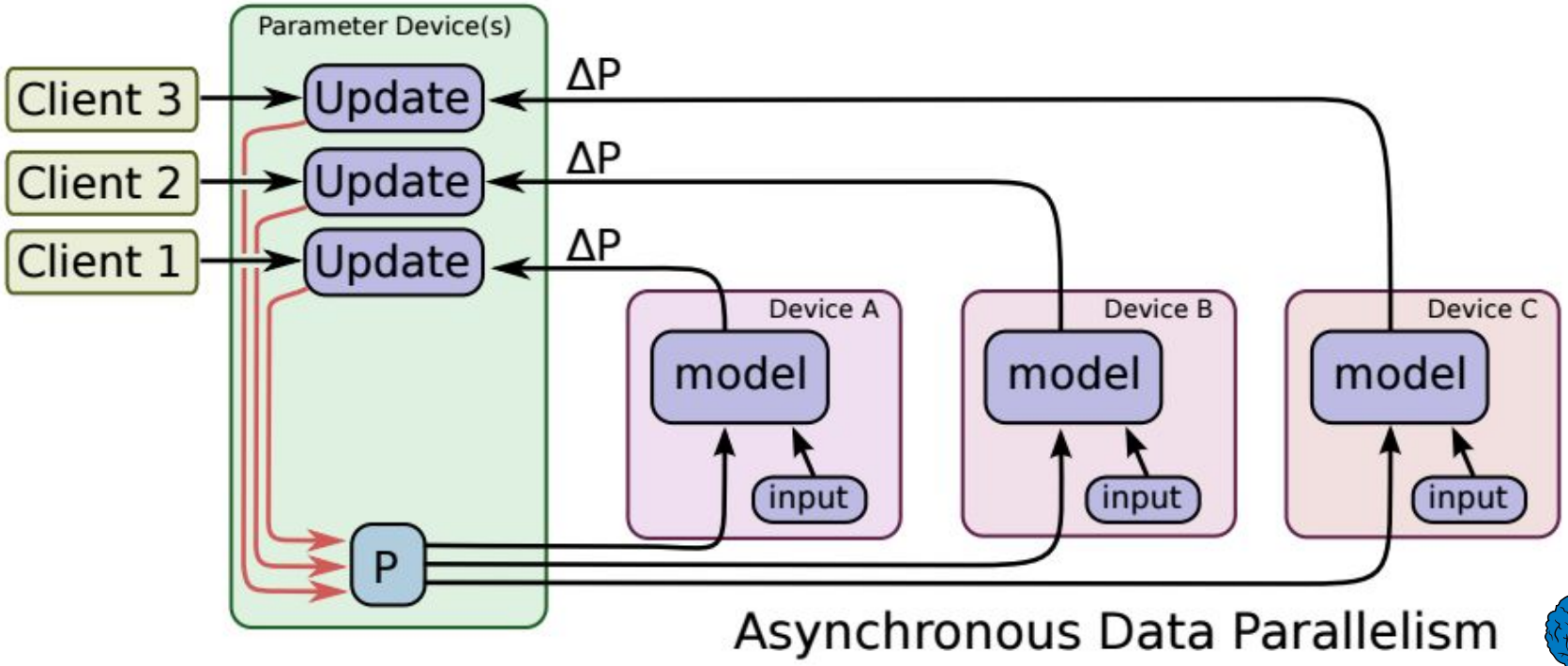
```
# Just assign everything to CPU0
with tf.device(“/cpu:0”):
    # Create the Mnist model.
    model = MnistModel(batch_size=16, hidden_units=200)

    # Get an session.
    sess = tf.Session()

    # Train the model.
    for local_step in xrange(FLAGS.max_steps):
      _, loss, step = sess.run([model.train_op, model.loss, model.global_step])
```

Data Parallelism: 5 lines Change

```
# We use the ReplicaDeviceSetter() device function to automatically
# assign Variables to the 'ps' jobs.
with tf.device(tf.ReplicaDeviceSetter(parameter_devices=10)):
    # Create the Mnist model.
    model = MnistModel(batch_size=16, hidden_units=200)

    # Create a Supervisor.  It will take care of initialization, summaries,
    # checkpoints, and recovery. When multiple replicas of this program are running,
    # the first one, identified by --task=0 is the 'chief' supervisor (e.g., initialization, saving)
    supervisor = tf.Supervisor(is_chief=(FLAGS.task == 0), saver=model.saver)

    # Get an initialized, and possibly recovered session.
    sess = supervisor.PrepareSession(FLAGS.master_job)

    # Train the model.
    for local_step in xrange(int32_max):
      _, loss, step = sess.run([model.train_op, model.loss, model.global_step])
      if step >= FLAGS.max_steps:
        break
```

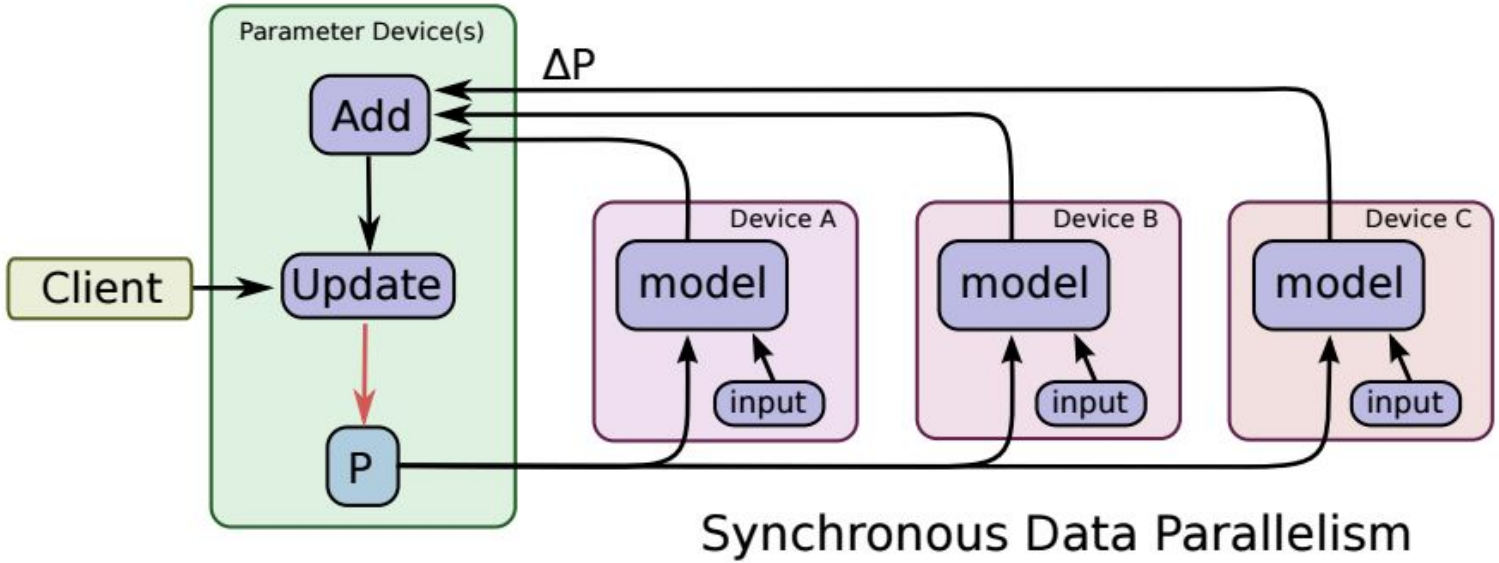
Asynchronous Training

- Each replica updates the parameters directly.



Synchronous Training

- Parameters are only updated when enough ***not-stale*** gradients are collected.



Data Parallelism Choices

Synchronously:

- **N replicas** equivalent to an **N times larger batch size (batch_norm still local)**
- Pro: No gradient staleness
- Con: Less fault tolerant (requires some recovery if any single machine fails)

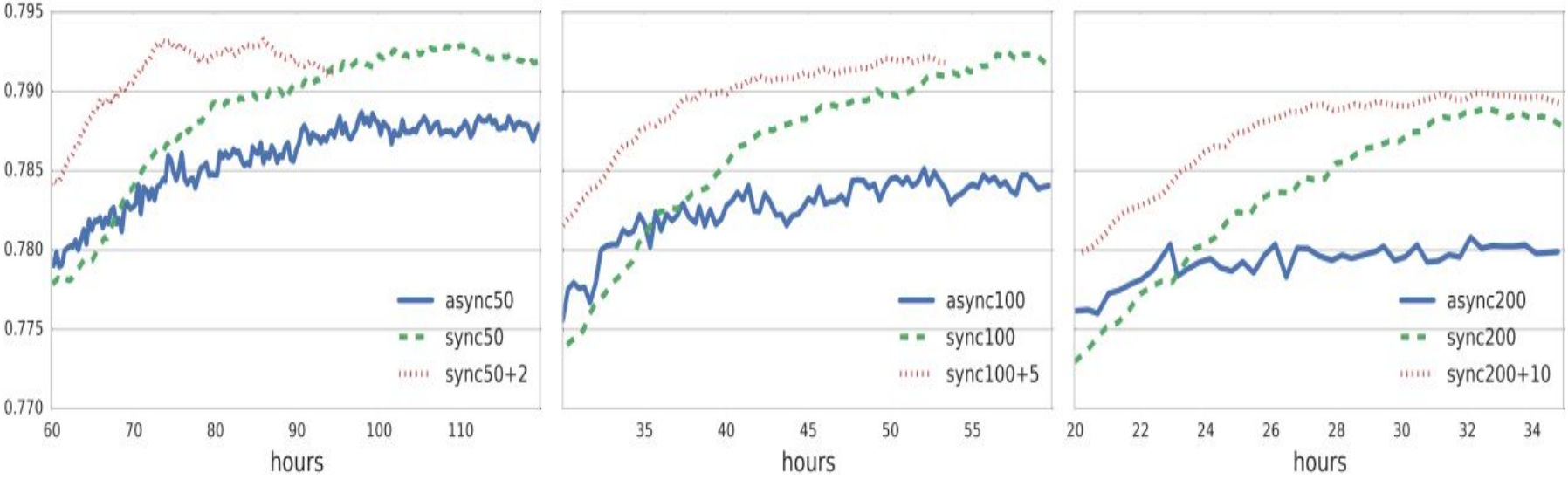
Asynchronously:

- Pro: Relatively fault tolerant (failure in model replica doesn't block other replicas)
- Con: Gradient staleness means each gradient less effective

(Or **hybrid**: M asynchronous groups of N synchronous replicas)

Sync VS Async on Inception (paper link)

- Sync could have both better models and faster training
- Sync could scale better



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!

A Few TensorFlow Community Examples

- 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

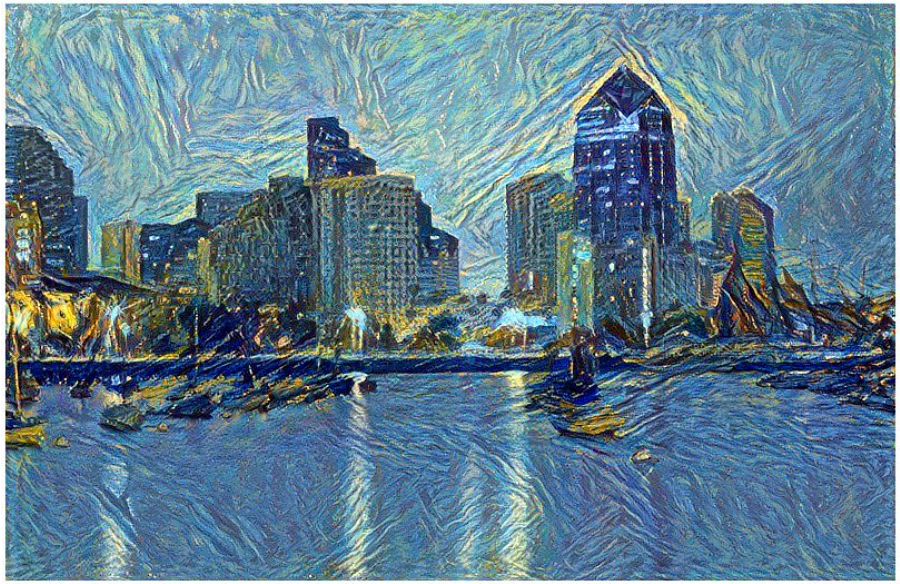
...

github.com/woodrush/neural-art-tf

"Neural Art" in TensorFlow

An implementation of "A neural algorithm of Artistic style" in TensorFlow, for

- Introductory, hackable demos for TensorFlow, and
- Demonstrating the use of importing various Caffe cnn models (VGG and illustration2vec) in TF.



github.com/sherjilozair/char-rnn-tensorflow

char-rnn-tensorflow

Multi-layer Recurrent Neural Networks (LSTM, RNN) for character-level language models in Python using Tensorflow.

Inspired from Andrej Karpathy's char-rnn.

Requirements

- Tensorflow

Basic Usage

To train with default parameters on the tinyshakespeare corpus, run `python train.py`.

To sample from a checkpointed model, `python sample.py`.

github.com/fchollet/keras

Keras: Deep Learning library for Theano and TensorFlow

You have just found Keras.

Keras is a minimalist, highly modular neural networks library, written in Python and capable of running either on top of either TensorFlow or Theano. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research.

Use Keras if you need a deep learning library that:

- allows for easy and fast prototyping (through total modularity, minimalism, and extensibility).
- supports both convolutional networks and recurrent networks, as well as combinations of the two.
- supports arbitrary connectivity schemes (including multi-input and multi-output training).
- runs seamlessly on CPU and GPU.

Read the documentation at Keras.io.

Keras is compatible with: - **Python 2.7-3.5** with the Theano backend - **Python 2.7** with the TensorFlow backend

github.com/jazzsaxmafia/show_and_tell.tensorflow

Neural Caption Generator

- Implementation of "Show and Tell" http://arxiv.org/abs/1411.4555
  - Borrowed some code and ideas from Andrej Karpathy's NeuralTalk.
- You need flickr30k data (images and annotations)

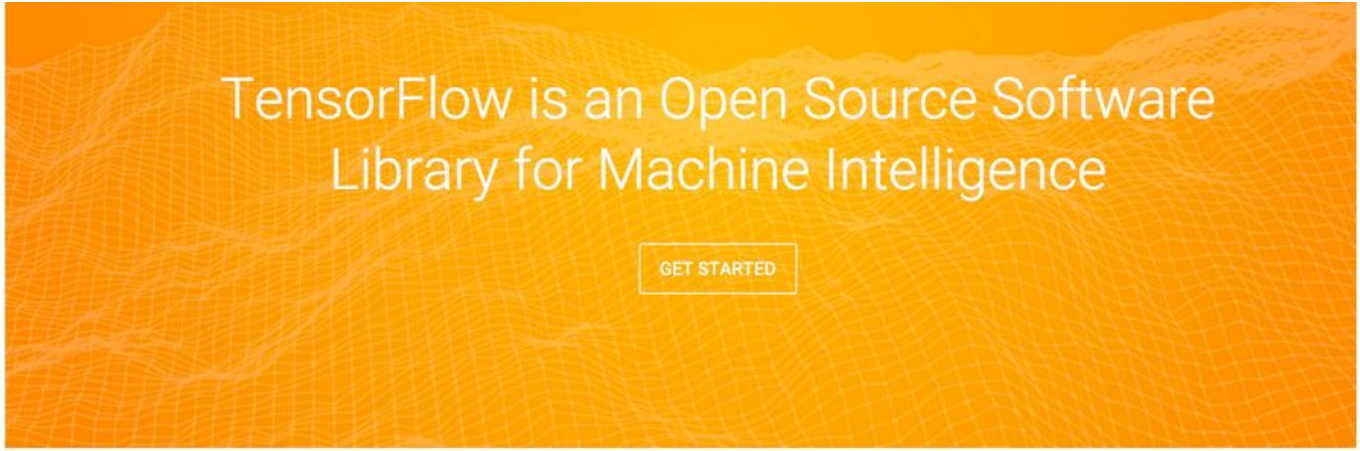
Code

- make_flickr_dataset.py : Extracting feats of flickr30k images, and save them in './data/feats.npy'
- model_tensorflow.py : TensorFlow Version
- model_theano.py : Theano Version

Usage

- Flickr30k Dataset Download
- Extract VGG Featues of Flicker30k images (make_flickr_dataset.py)
- Train: run train() in model_tensorflow.py or model_theano.py
- Test: run test() in model_tensorflow.py or model_theano.py.
  - parameters: VGG FC7 feature of test image, trained model path

github.com/jikexueyuanwiki/tensorflow-zh



你正在翻译的项目可能会比 Android 系统更加深远地影响着世界！

缘起

2015年11月9日，Google 官方在其博客上称，Google Research 宣布推出第二代机器学习系统 TensorFlow，针对先前的 DistBelief 的短板有了各方面的加强，更重要的是，它是开源的，任何人都可以用。

机器学习作为人工智能的一种类型，可以让软件根据大量的数据来对未来的情况进行阐述或预判。如今，领先的科技巨头无不在机器学习下予以极大投入。Facebook、苹果、微软，甚至国内的百度。Google 自然也在其中。「TensorFlow」是 Google 多年以来内部的机器学习系统。如今，Google 正在将此系统成为开源系统，并将此系统的参数公布给业界工程师、学者和拥有大量编程能力的技术人员，这意味着什么呢?

Further Reading

http://tensorflow.org/ https://github.com/tensorflow

(Clickable links in bibliography)

- Dean, *et al.*, *Large Scale Distributed Deep Networks*, NIPS 2012
- TensorFlow: A System for Large-Scale Machine Learning
- Sutskever, *et al.*, *Sequence to Sequence Learning with Neural Networks*, NIPS 2014
- Szegedy, *et al.*, Rethinking the Inception Architecture for Computer Vision, CVPR 2016

Research in Brain:

research.google.com/pubs/BrainTeam.html

We are hiring!!

Brain: g.co/brain

Brain Residency Program: g.co/brainresidency

Questions?