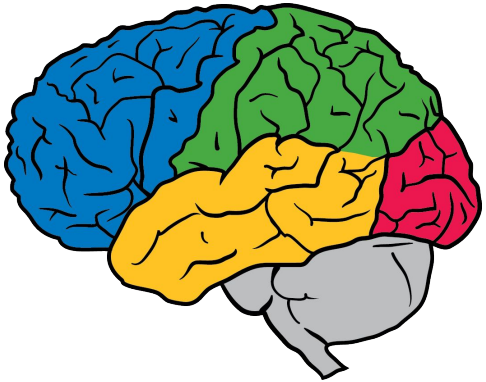


TensorFlow: A System for Machine Learning on Heterogeneous Systems

Jeff Dean
Google

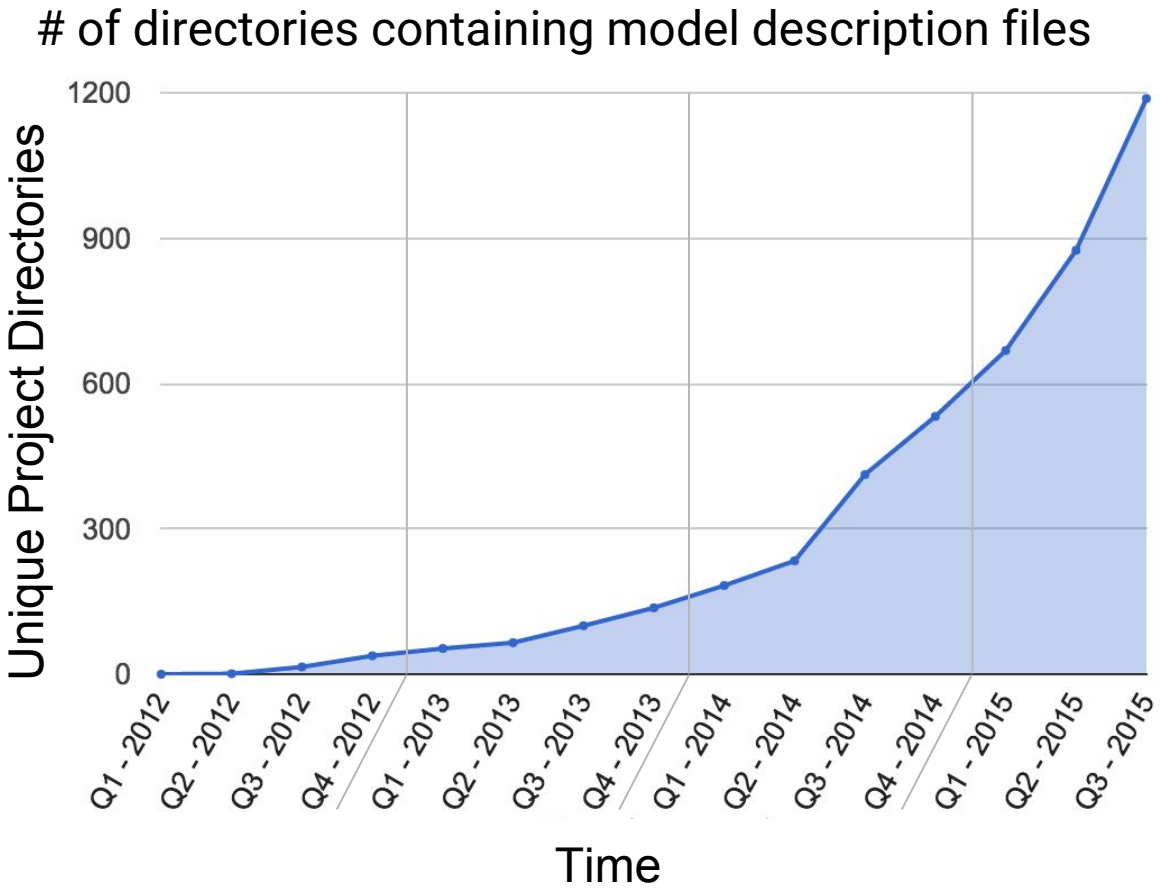
Google Brain team in collaboration with many other teams

Google Brain Team

- **Mission:** Develop advanced AI techniques and make them useful for people

- Strong mix of pure research, applied research, and computer systems building

Growing Use of Deep Learning at Google



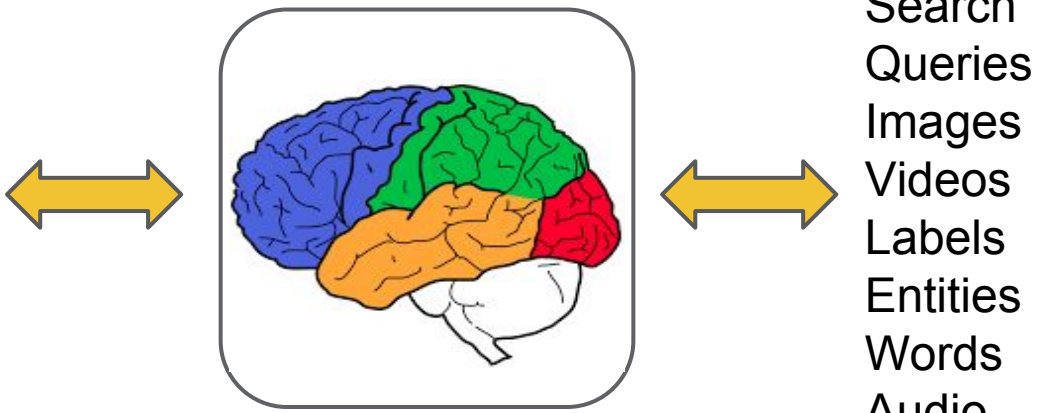
Across many products/areas:

Android
Apps
drug discovery
Gmail
Image understanding
Maps
Natural language understanding
Photos
Robotics research
Speech
Translation
YouTube
… many others ...

Deep Learning

Universal Machine Learning

Speech
Text
Search
Queries
Images
Videos
Labels
Entities
Words
Audio
Features



Speech
Text
Search
Queries
Images
Videos
Labels
Entities
Words
Audio
Features

What do you want in a machine learning system?

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

TensorFlow:
Second Generation Deep Learning System

TensorFlow

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

Open sourced single-machine TensorFlow on Monday, Nov. 9th

- Flexible Apache 2.0 open source licensing
- Updates for distributed implementation coming soon

http://tensorflow.org/

Version: master

MNIST For ML Beginners

- The MNIST Data
- Softmax Regressions
- Implementing the Regression
- Training
- Evaluating Our Model

Deep MNIST for Experts

- Setup
  - Load MNIST Data
  - Start TensorFlow InteractiveSession
- Build a Softmax Regression Model
  - Placeholders
  - Variables
  - Predicted Class and Cost Function
- Train the Model
  - Evaluate the Model
- Build a Multilayer Convolutional Network
  - Weight Initialization
  - Convolution and Pooling
  - First Convolutional Layer
  - Second Convolutional Layer
  - Densely Connected Layer
  - Readout Layer
  - Train and Evaluate the Model

TensorFlow Mechanics 101

- Tutorial Files
- Prepare the Data

TensorFlow Mechanics 101

This is a technical tutorial, where we walk you through the details of using TensorFlow infrastructure to train models at scale. We use again MNIST as the example.

View Tutorial

Convolutional Neural Networks

An introduction to convolutional neural networks using the CIFAR-10 data set. Convolutional neural nets are particularly tailored to images, since they exploit translation invariance to yield more compact and effective representations of visual content.

View Tutorial

Vector Representations of Words

This tutorial motivates why it is useful to learn to represent words as vectors (called word embeddings). It introduces the word2vec model as an efficient method for learning embeddings. It also covers the high-level details behind noise-contrastive training methods (the biggest recent advance in training embeddings).

View Tutorial

Recurrent Neural Networks

An introduction to RNNs, wherein we train an LSTM network to predict the next word in an English sentence. (A task sometimes called language modeling.)

View Tutorial

Sequence-to-Sequence Models

A follow on to the RNN tutorial, where we assemble a sequence-to-sequence model for machine translation. You will learn to build your own English-to-French translator, entirely machine learned, end-to-end.

View Tutorial

Motivations

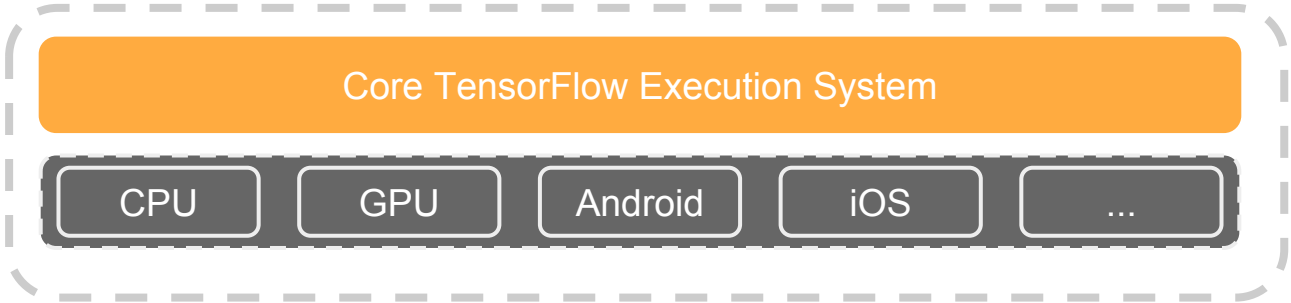
DistBelief (1st system):

- 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

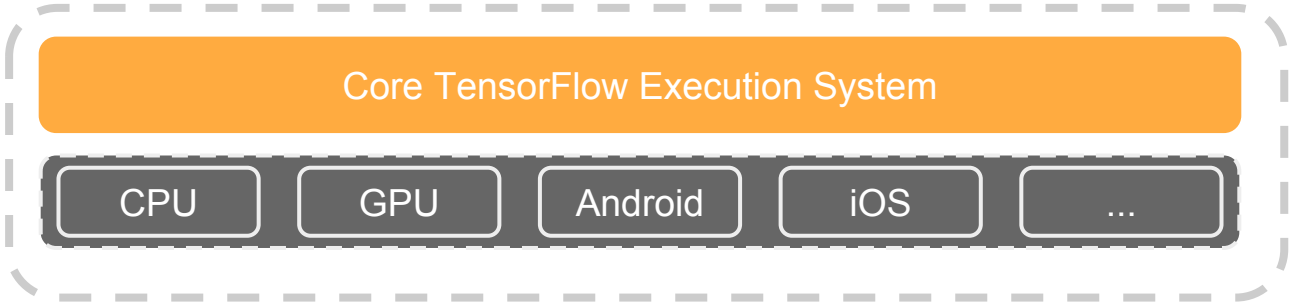
TensorFlow: Expressing High-Level ML Computations

- Core in C++



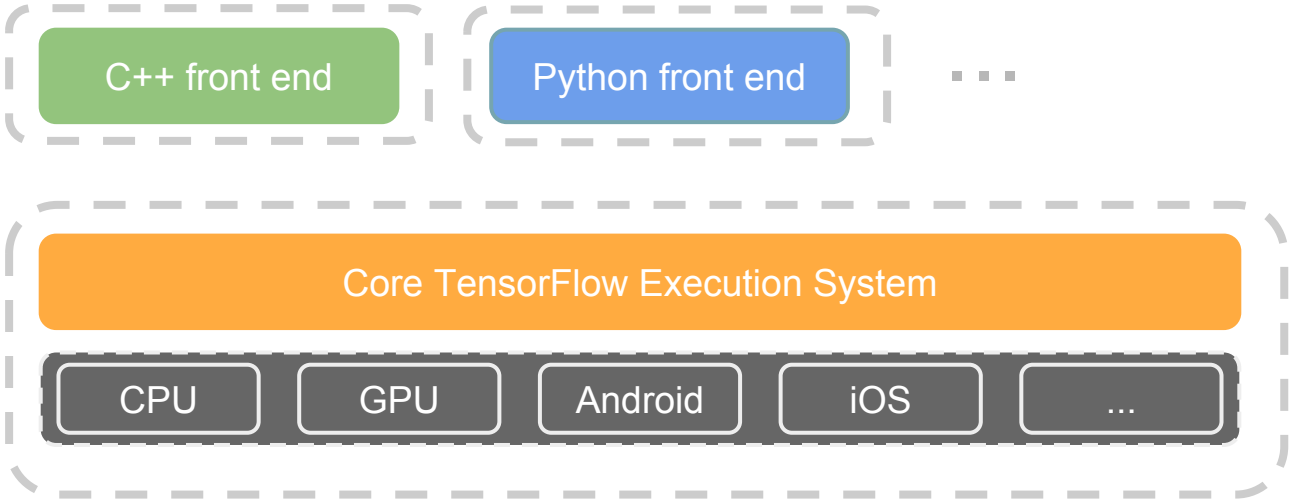
TensorFlow: Expressing High-Level ML Computations

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



TensorFlow: Expressing High-Level ML Computations

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



Portable

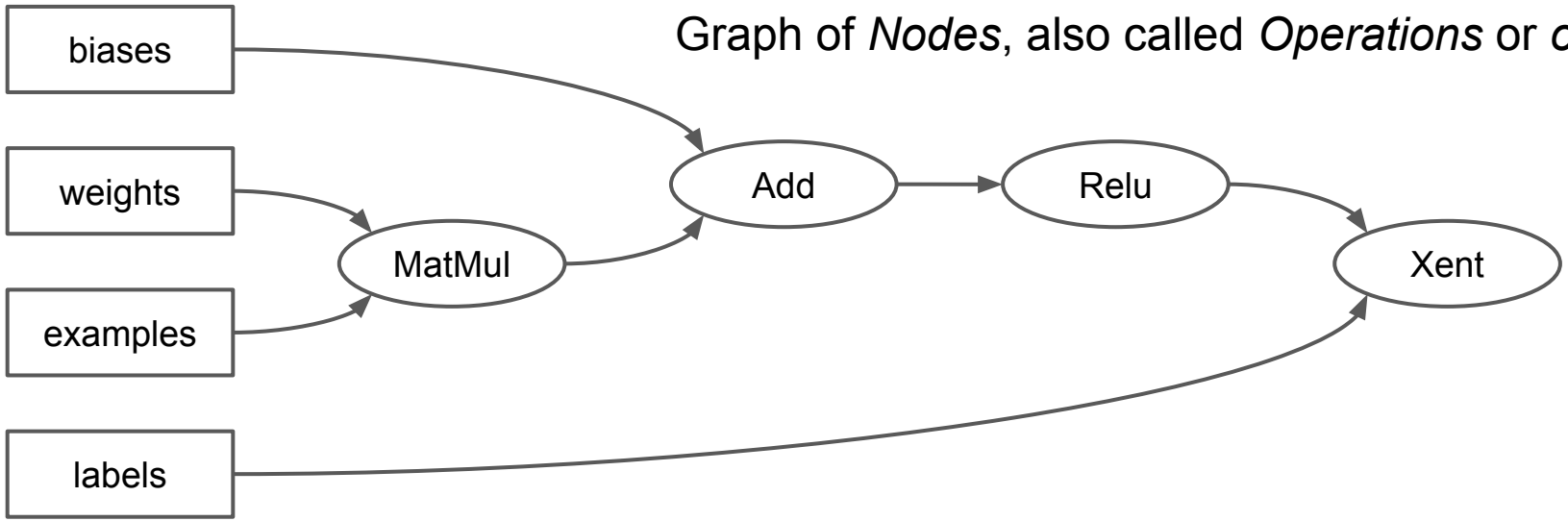
Automatically runs models on range of platforms:

from **phones** ...

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

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

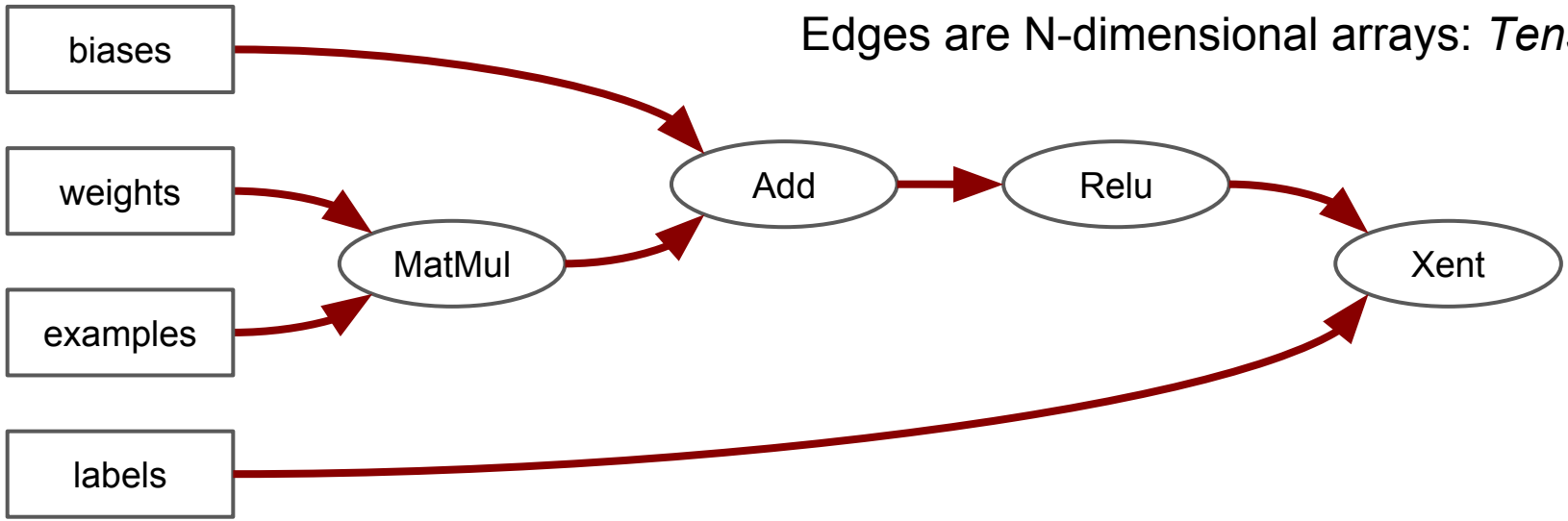
Computation is a dataflow graph



Graph of *Nodes*, also called *Operations* or *ops.*

Computation is a dataflow graph

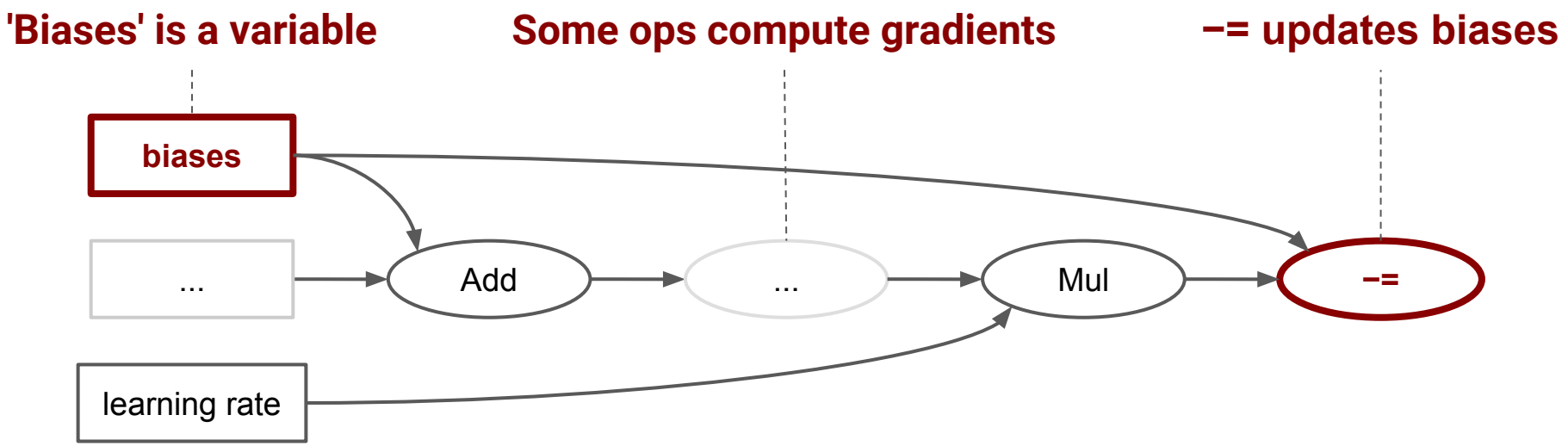
with tensors



Edges are N-dimensional arrays: *Tensors*

Computation is a dataflow graph

with state



Automatic Differentiation

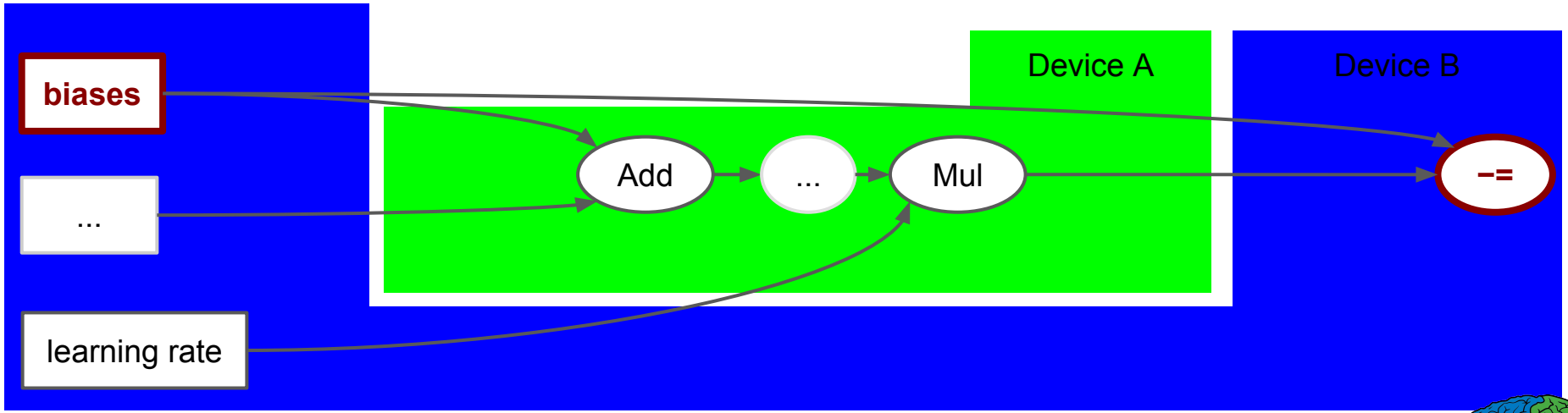
Similar to Theano, TensorFlow can automatically calculate symbolic gradients of variables w.r.t. loss function.

```
# Minimize the mean squared errors.
loss = tf.reduce_mean(tf.square(y-predict - y_expected))
optimizer = tf.train.GradientDescentOptimizer(0.01)
train = optimizer.minimize(loss)
```

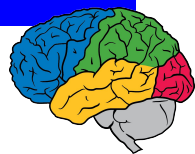
Much easier to express complex and train complex models

Computation is a dataflow graph

distributed

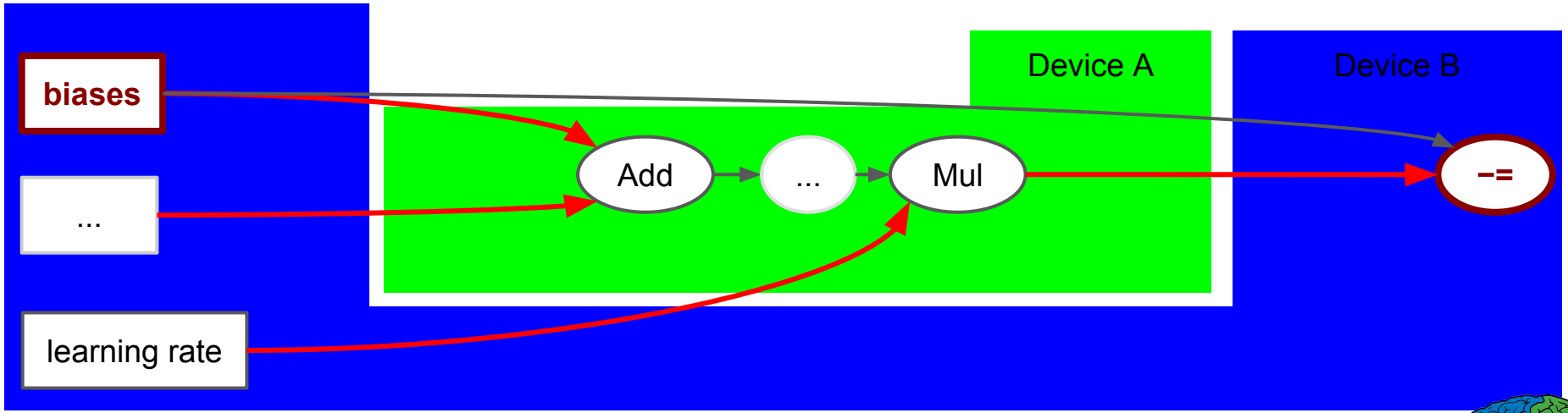


Devices: Processes, Machines, GPUs, etc

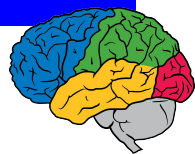


Send and Receive Nodes

distributed

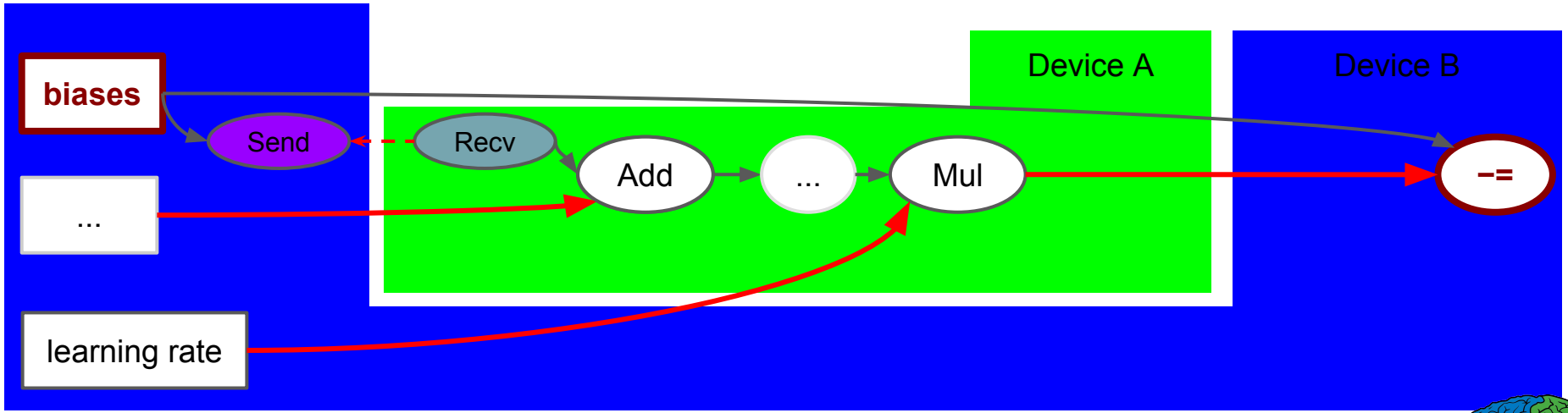


Devices: Processes, Machines, GPUs, etc

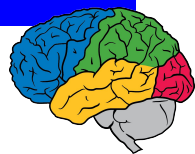


Send and Receive Nodes

distributed

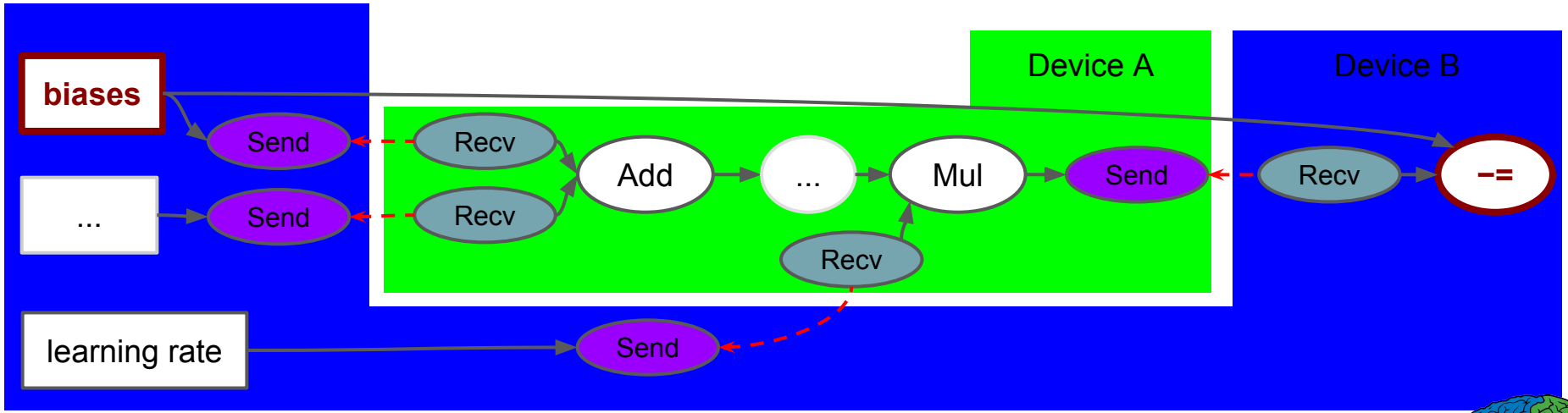


Devices: Processes, Machines, GPUs, etc

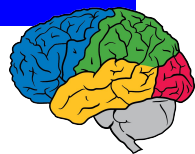


Send and Receive Nodes

distributed



Devices: Processes, Machines, GPUs, etc



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**

Extensible

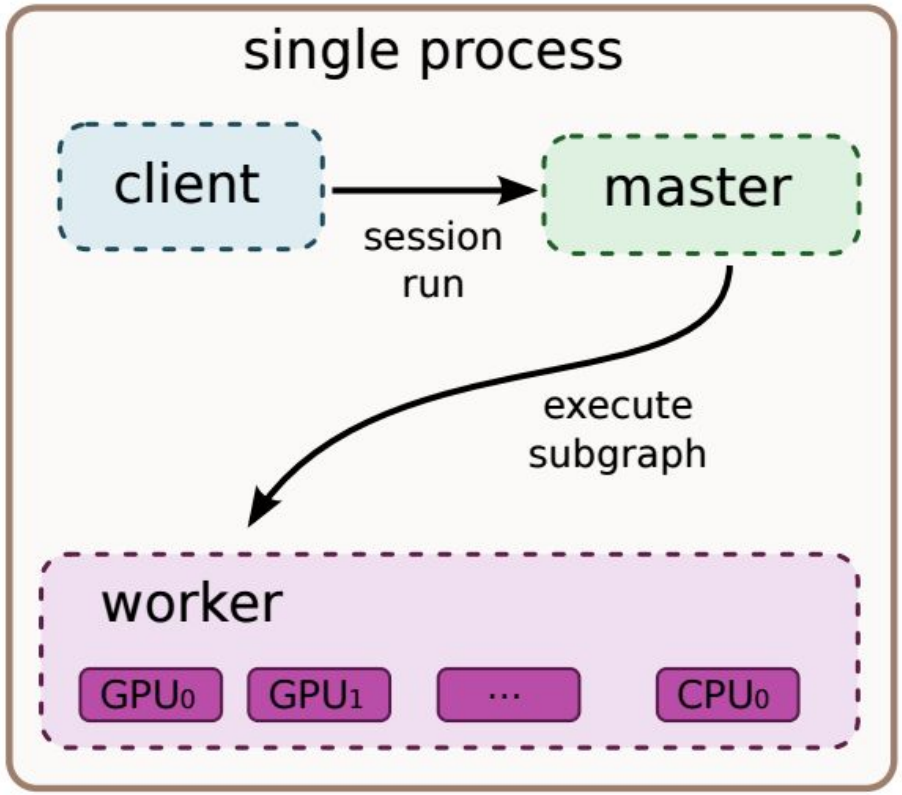
- Core system defines a number of standard ***operations*** and ***kernels*** (device-specific implementations of operations)
- Easy to define new operators and/or kernels

Session Interface

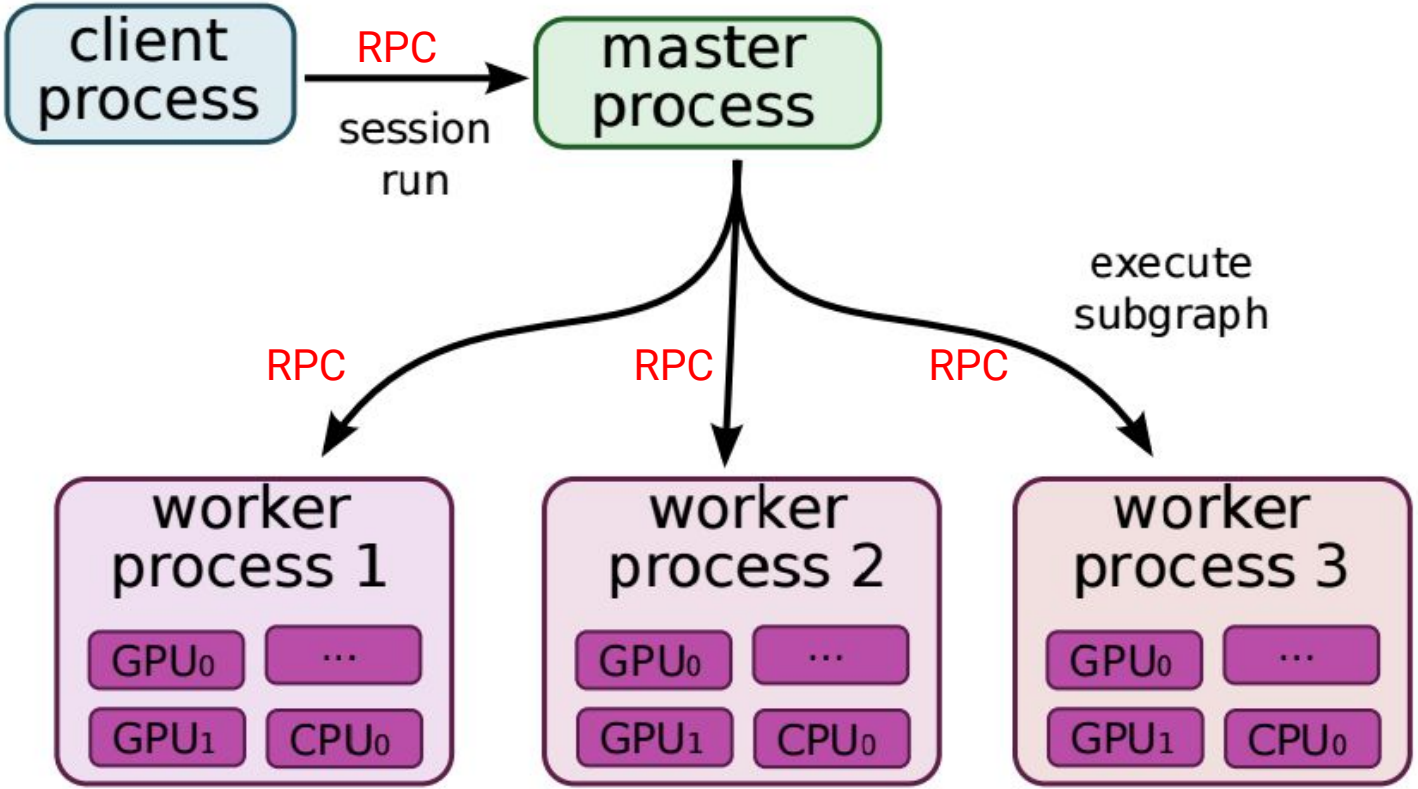
- `Extend`: add nodes to computation graph
- `Run`: execute an arbitrary subgraph
  - optionally feeding in Tensor inputs and retrieving Tensor output

Typically, setup a graph with one or a few `Extend` calls and then `Run` it thousands or millions or times

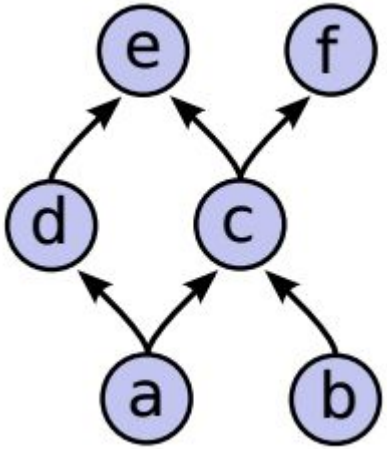
Single Process Configuration



Distributed Configuration

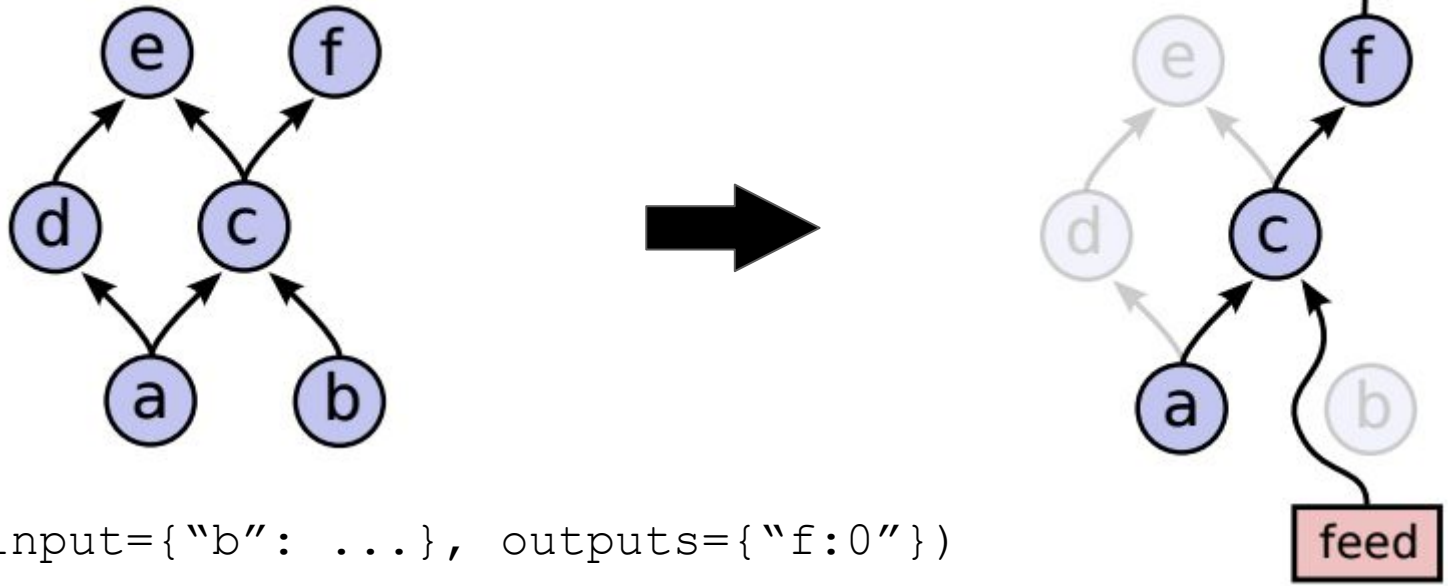


Feeding and Fetching




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

Feeding and Fetching




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

TensorFlow Single Device Performance

Initial measurements done by Soumith Chintala

Benchmark	Forward	Forward+Backward
AlexNet - cuDNNv3 on Torch (Soumith)	32 ms	96 ms
AlexNet - Neon (Soumith)	32 ms	101 ms
AlexNet - cuDNNv2 on Torch (Soumith)	70 ms	231 ms
AlexNet - cuDNNv2 on TensorFlow 0.5 (Soumith)	**96 ms**	**326 ms**

See https://github.com/soumith/convnet-benchmarks/issues/66

Two main factors:

(1) various overheads (nvcc doesn't like 64-bit tensor indices, etc.)

(2) versions of convolutional libraries being used (cuDNNv2 vs. v3, etc.)

TensorFlow Single Device Performance

Prong 1: Tackling sources of overhead

Benchmark	Forward	Forward+Backward
AlexNet - cuDNNv3 on Torch (Soumith)	32 ms	96 ms
AlexNet - Neon (Soumith)	32 ms	101 ms
AlexNet - cuDNNv2 on Torch (Soumith)	**70 ms**	**231 ms**
AlexNet - cuDNNv2 on TensorFlow 0.5 (Soumith)	96 ms	326 ms
AlexNet - cuDNNv2 on TensorFlow 0.5 (our machine)	97 ms	336 ms
AlexNet - cuDNNv2 on TensorFlow 0.6 (our machine: soon)	**70 ms (+39%)**	**230 ms (+31%)**

TensorFlow Single Device Performance

TODO: Release 0.6 this week improves speed to equivalent with other packages using cuDNNv2

Subsequent updates will upgrade to faster core libraries like cuDNN v3 (and/or the upcoming v4)

Also looking to improve memory usage

Single device performance important, but

….

biggest performance improvements come from large-scale distributed systems with model and data parallelism

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

Transition

- How do you do this at scale?
- How does TensorFlow make distributed training easy?

Model Parallelism

- Best way to decrease training time: **decrease the step time**
- Many models have lots of inherent parallelism
- Problem is distributing work so communication doesn't kill you
  - local connectivity (as found in CNNs)
  - towers with little or no connectivity between towers (e.g. AlexNet)
  - specialized parts of model active only for some examples

Exploiting Model Parallelism

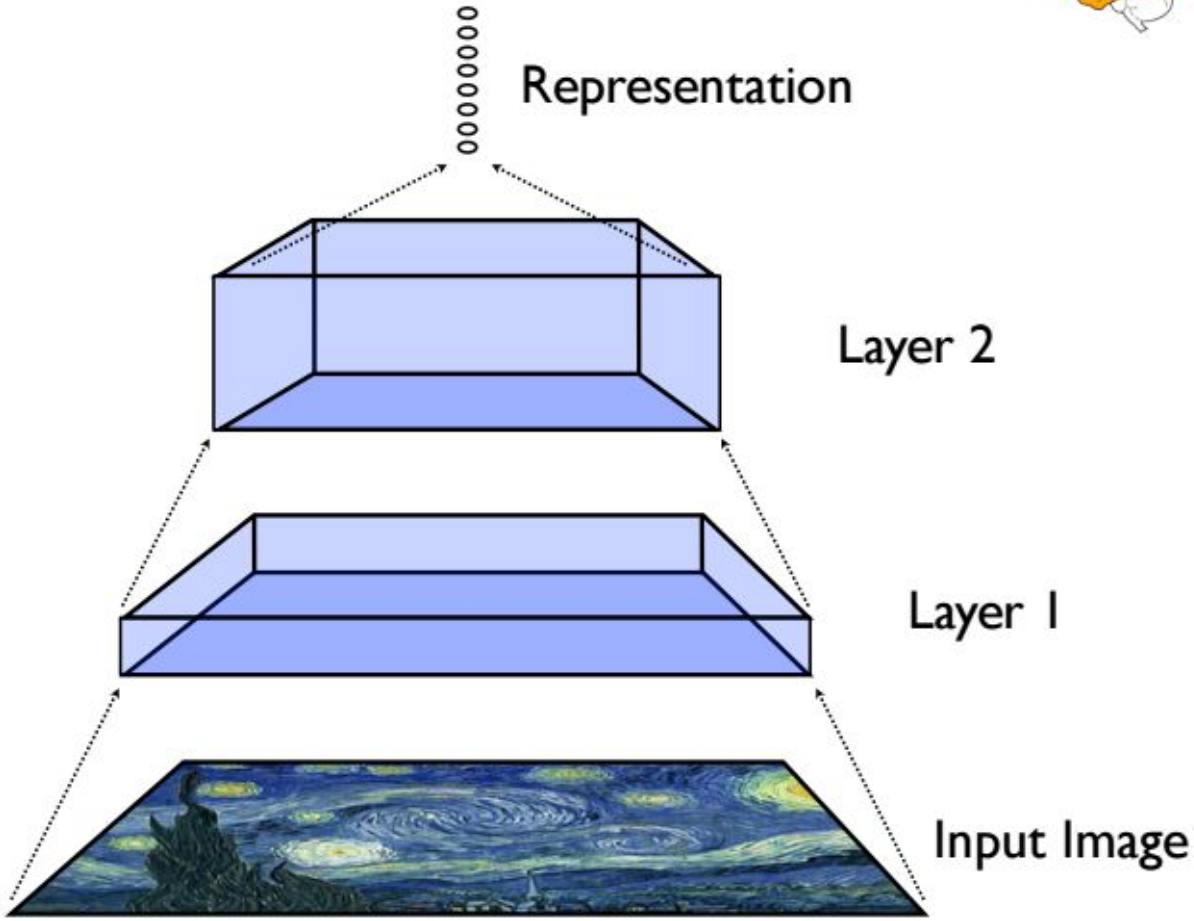
On a single core: Instruction parallelism (SIMD). Pretty much free.

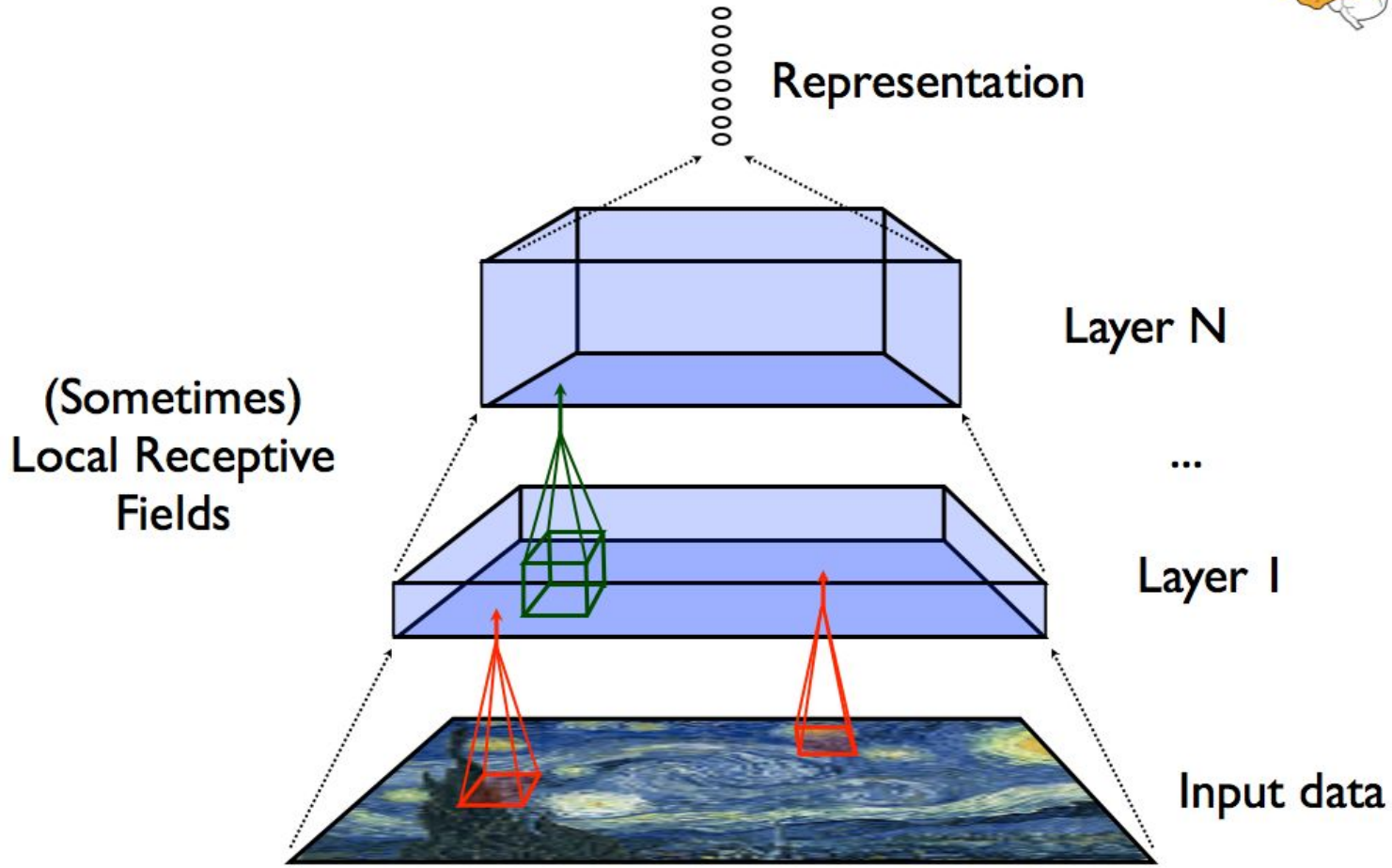
Across cores: thread parallelism. Almost free, unless across sockets, in which case inter-socket bandwidth matters (QPI on Intel).

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

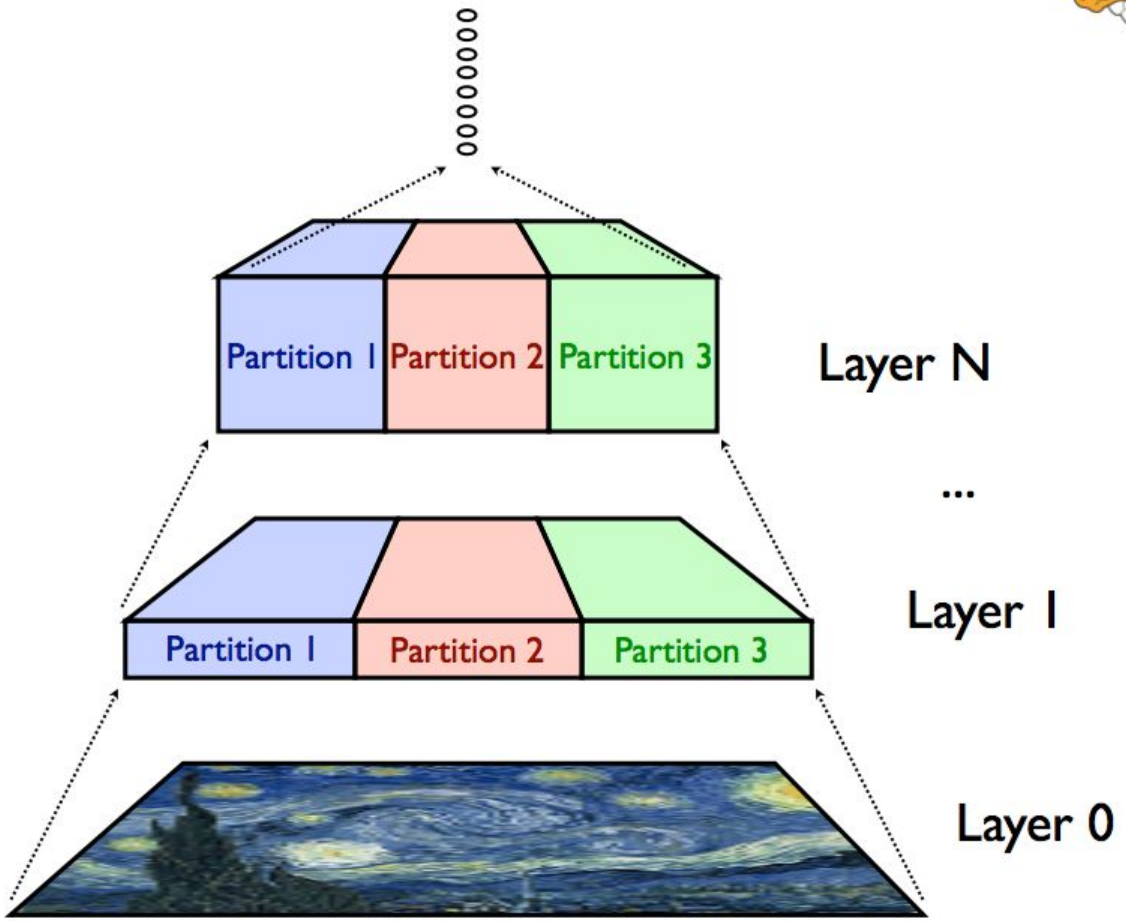
Across machines: limited by network bandwidth / latency

Model Parallelism

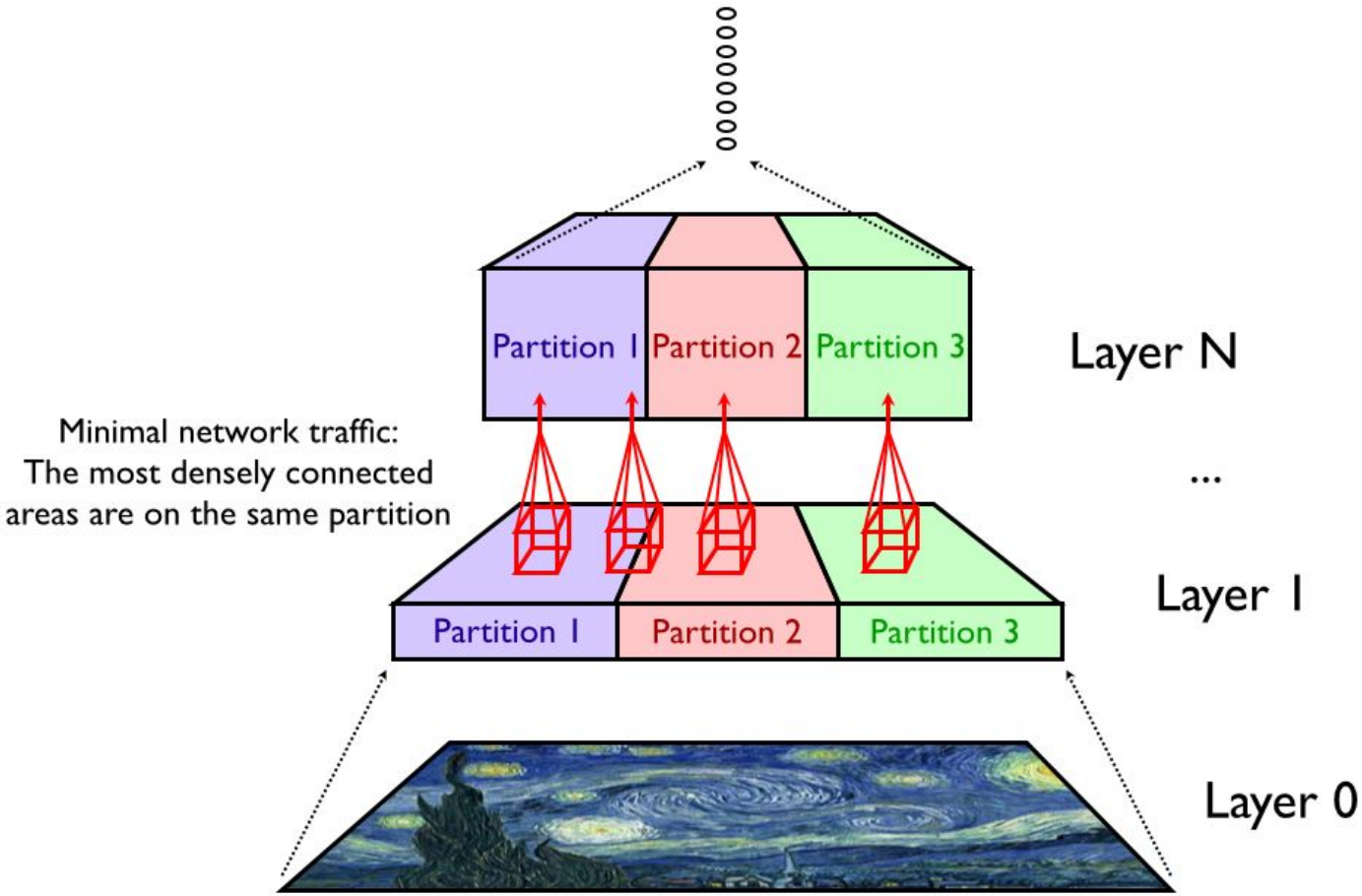



Representation
Layer N
...
Layer 1
Input data
(Sometimes)
Local Receptive
Fields

Model Parallelism: Partition model across machines



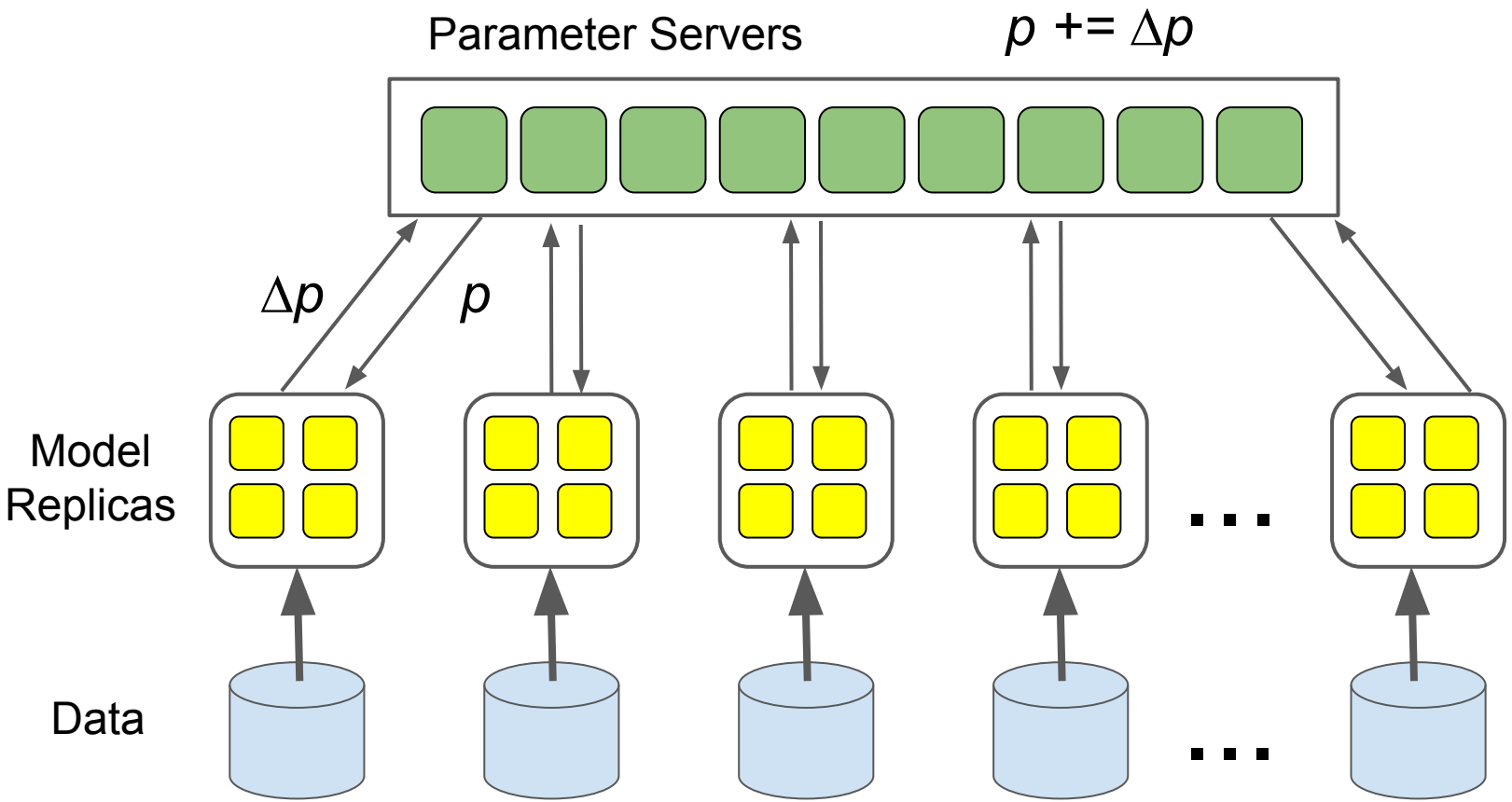
Model Parallelism: Partition model across machines



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

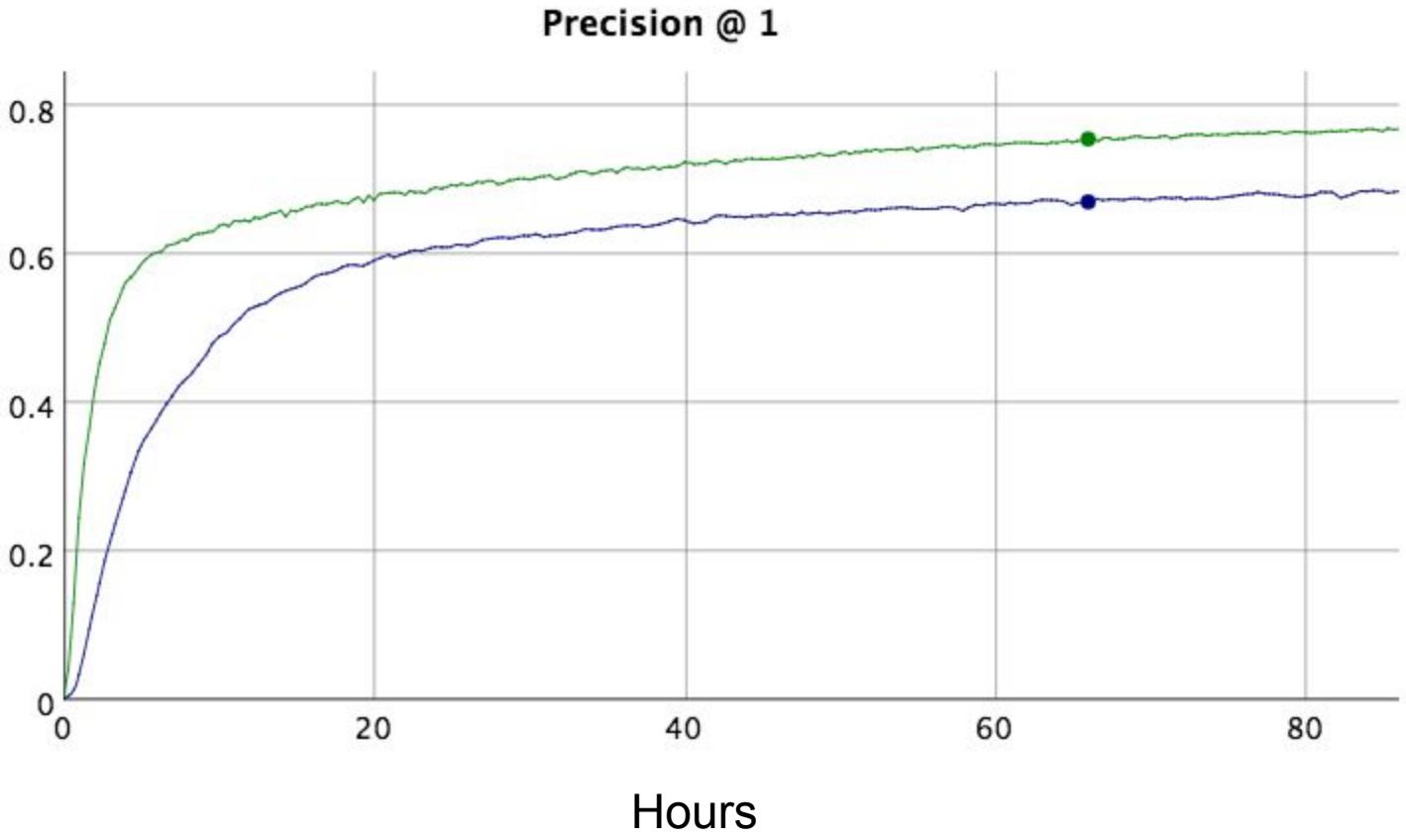
Data Parallelism



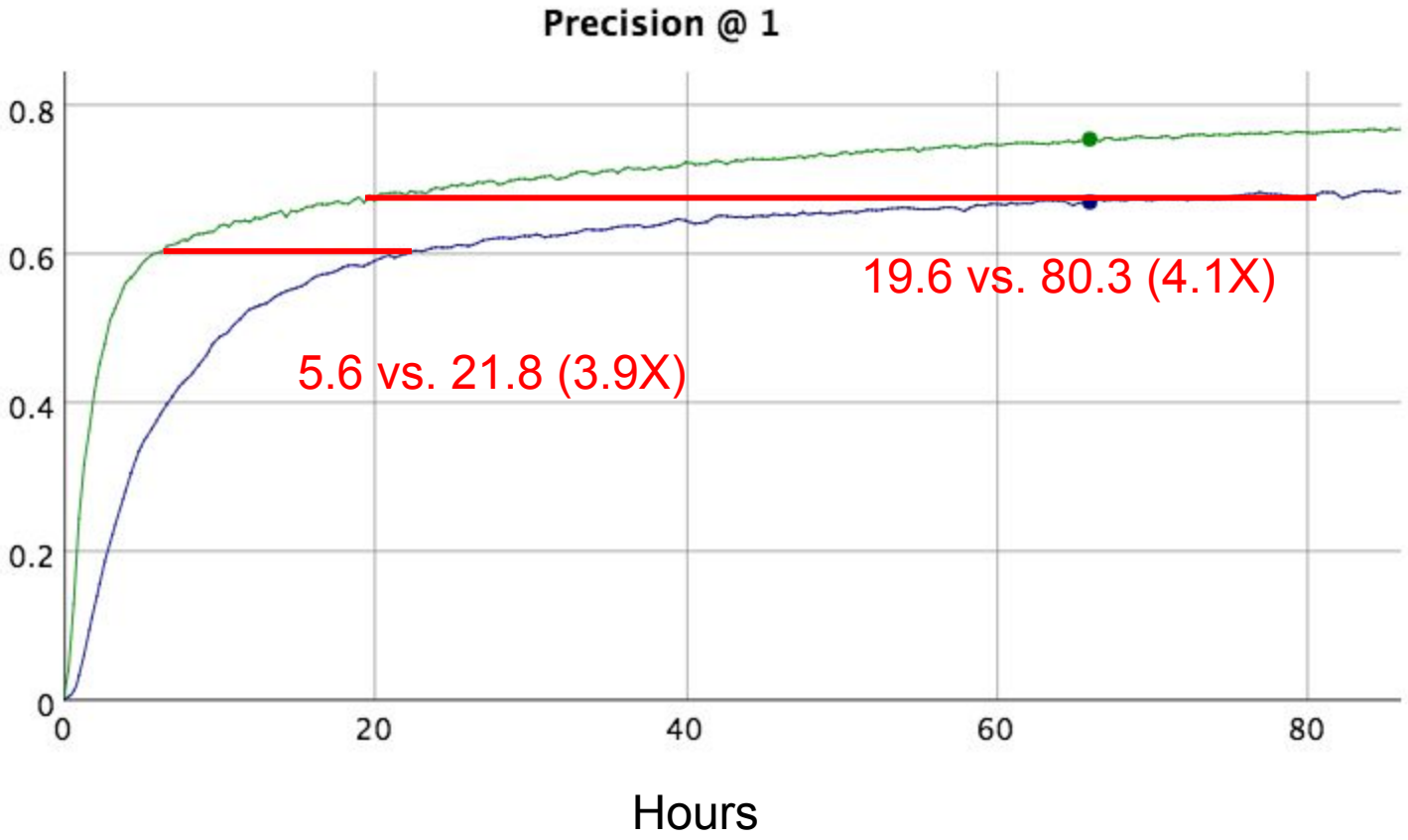
Success of Data Parallelism

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

10 vs 50 Replica Inception Synchronous Training



10 vs 50 Replica Inception Synchronous Training



Using TensorFlow for Parallelism

Trivial to express both model parallelism as well as data parallelism

- Very minimal changes to single device model code

Devices and Graph Placement

- Given a graph and set of devices, TensorFlow implementation must decide which device executes each node

Full and Partial Device Constraints (Hints)

Devices are named hierarchically:

```
/job:localhost/device:cpu:0
/job:worker/task:17/device:gpu:3
/job:parameters/task:4/device:cpu:0
```

Client can specify full or partial constraints for nodes in graph:

“Place this node on `/job:localhost/device:gpu:2`”

“Place this node on `/device:gpu:*`”

Placement Algorithm

Given hints, plus a cost model (node execution time estimates and Tensor size estimates), make placement decisions

- Current relatively simple greedy algorithm
- Active area of work

Example: LSTM [Hochreiter et al, 1997]

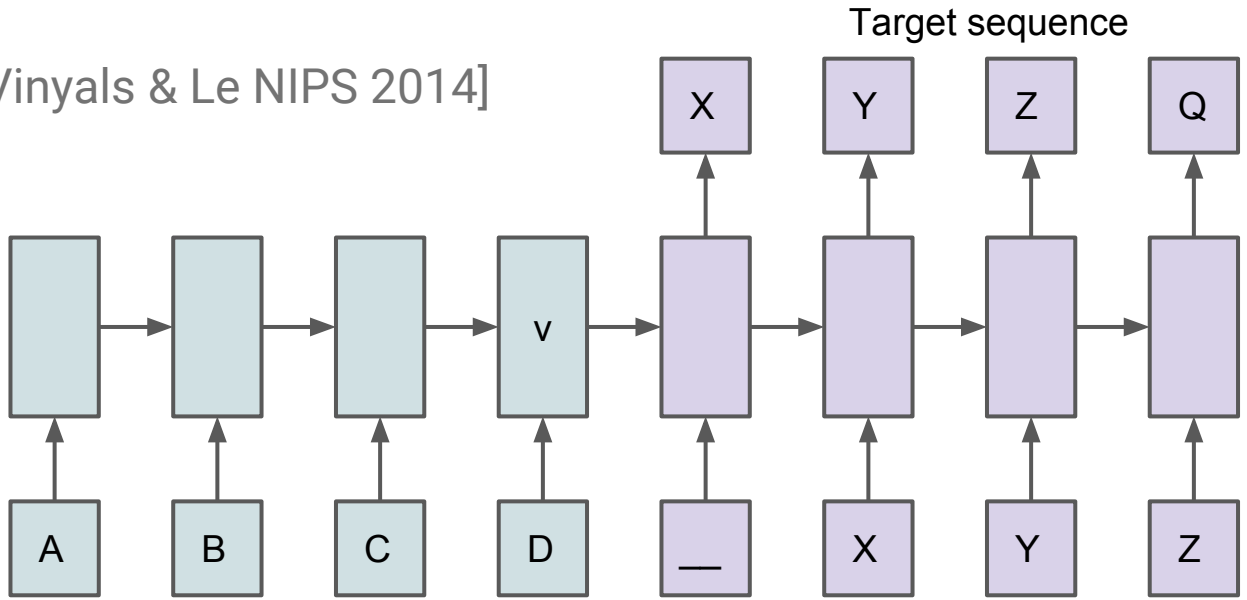
- From research paper to code

$$
\begin{aligned}
i_t &= W_{ix}x_t + W_{ih}h_{t-1} + b_i \\
j_t &= W_{jx}x_t + W_{jh}h_{t-1} + b_j \\
f_t &= W_{fx}x_t + W_{fh}h_{t-1} + b_f \\
o_t &= W_{ox}x_t + W_{oh}h_{t-1} + b_o \\
c_t &= \sigma(f_t) \odot c_{t-1} + \sigma(i_t) \odot \tanh(j_t) \\
h_t &= \sigma(o_t) \odot \tanh(c_t)
\end{aligned}
$$

```
def __call__(self, inputs, state, scope=None):
  """Long short-term memory cell (LSTM)."""
  with vs.variable_scope(scope or type(self).__name__):  # "BasicLSTMCell"
    # Parameters of gates are concatenated into one multiply for efficiency.
    c, h = array_ops.split(1, 2, state)
    concat = linear([inputs, h], 4 * self._num_units, True)

    # i = input_gate, j = new_input, f = forget_gate, o = output_gate
    i, j, f, o = array_ops.split(1, 4, concat)

    new_c = c * sigmoid(f + self._forget_bias) + sigmoid(i) * tanh(j)
    new_h = tanh(new_c) * sigmoid(o)

  return new_h, array_ops.concat(1, [new_c, new_h])
```

Sequence-to-Sequence Model

[Sutskever & Vinyals & Le NIPS 2014]



$$P(y_1, \ldots, y_{T'} | x_1, \ldots, x_T) = \prod_{t=1}^{T'} p(y_t | v, y_1, \ldots, y_{t-1})$$

Example: LSTM

```
for i in range(20):
    m, c = LSTMCell(x[i], mprev, cprev)
    mprev = m
    cprev = c
```

Example: Deep LSTM

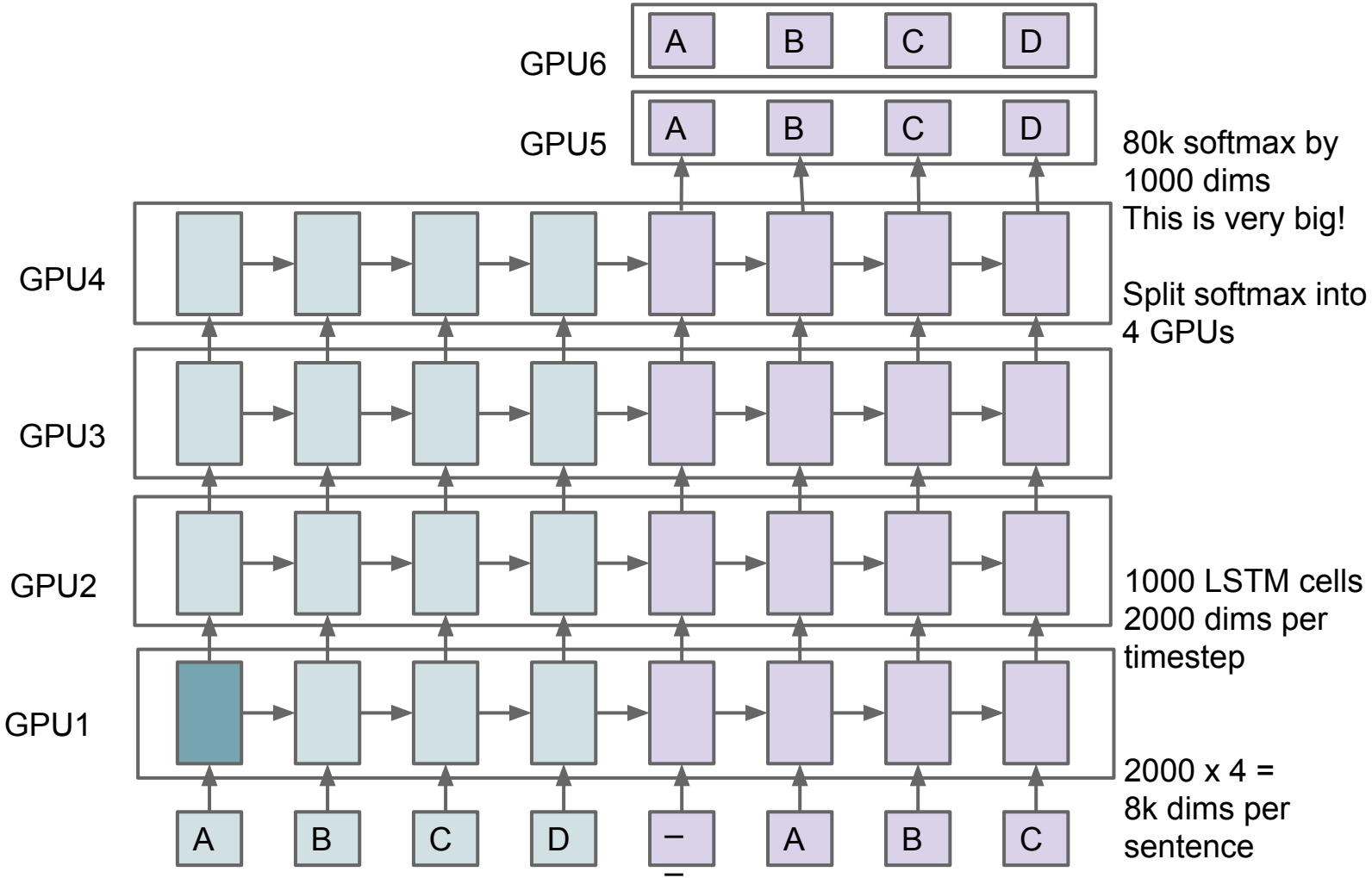
```
for i in range(20):
  for d in range(4): # d is depth
      input = x[i] if d is 0 else m[d-1]
      m[d], c[d] = LSTMCell(input, mprev[d], cprev[d])
      mprev[d] = m[d]
      cprev[d] = c[d]
```

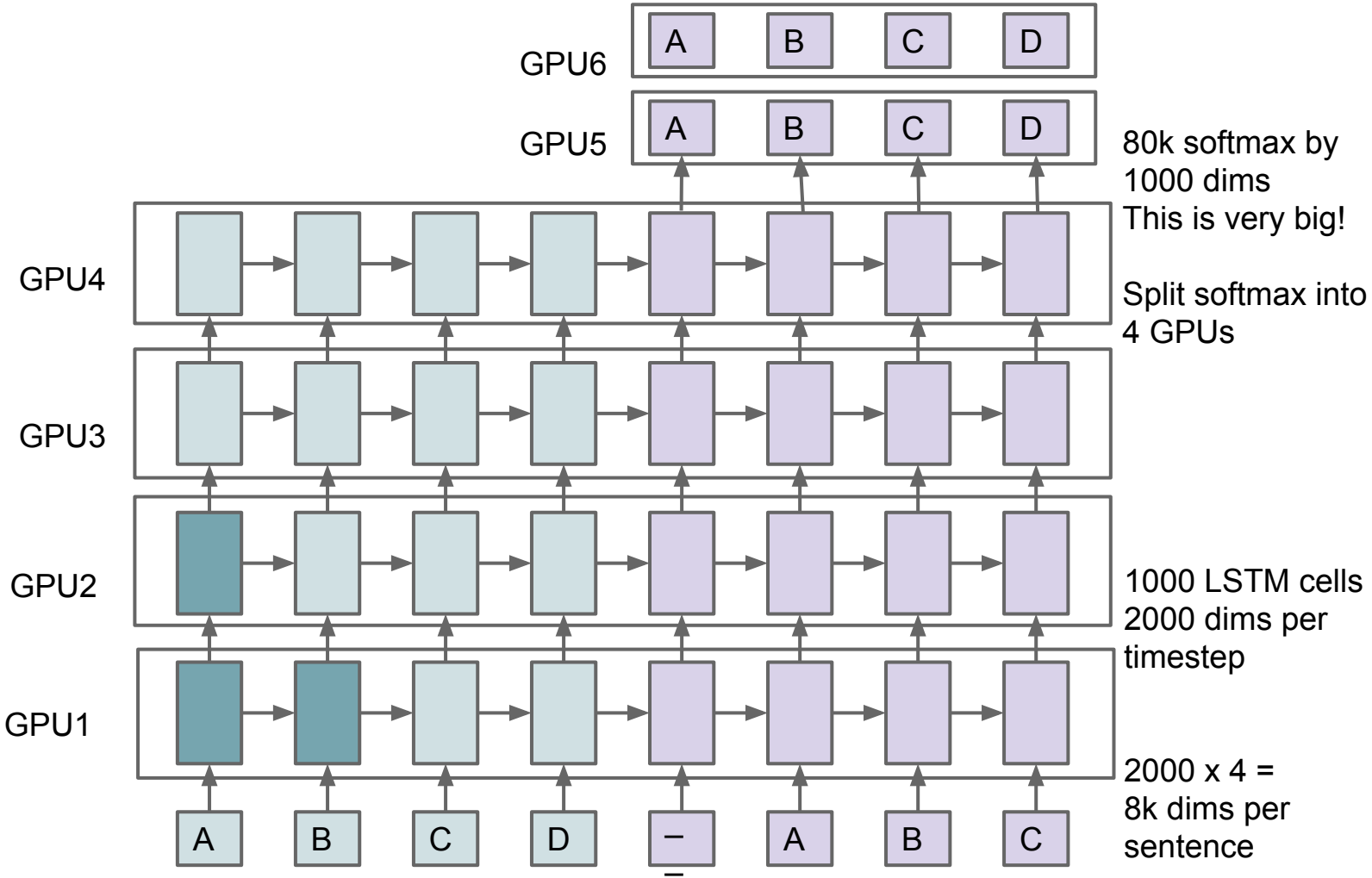
Example: Deep LSTM

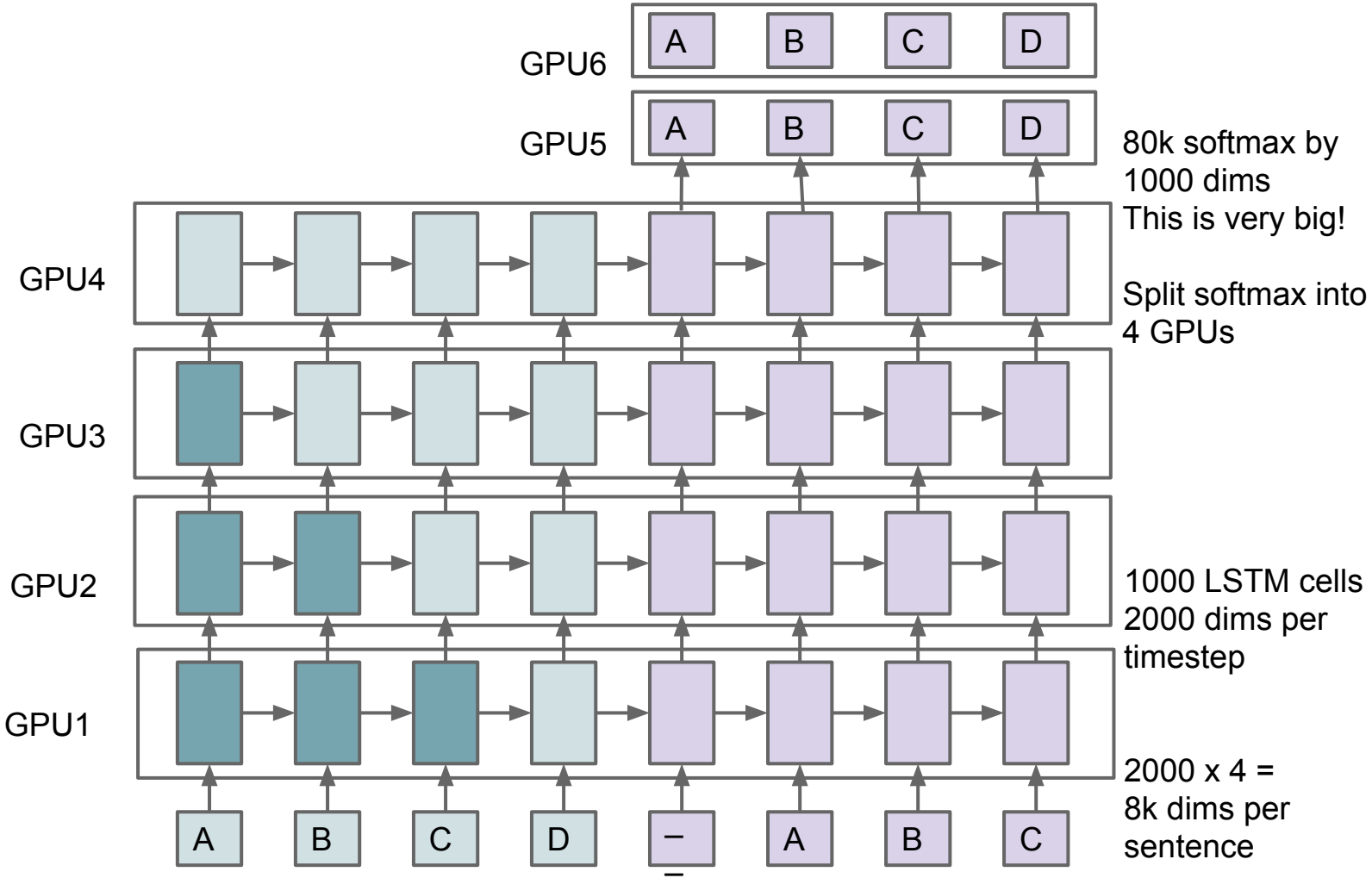
```
for i in range(20):
  for d in range(4): # d is depth
      input = x[i] if d is 0 else m[d-1]
      m[d], c[d] = LSTMCell(input, mprev[d], cprev[d])
      mprev[d] = m[d]
      cprev[d] = c[d]
```

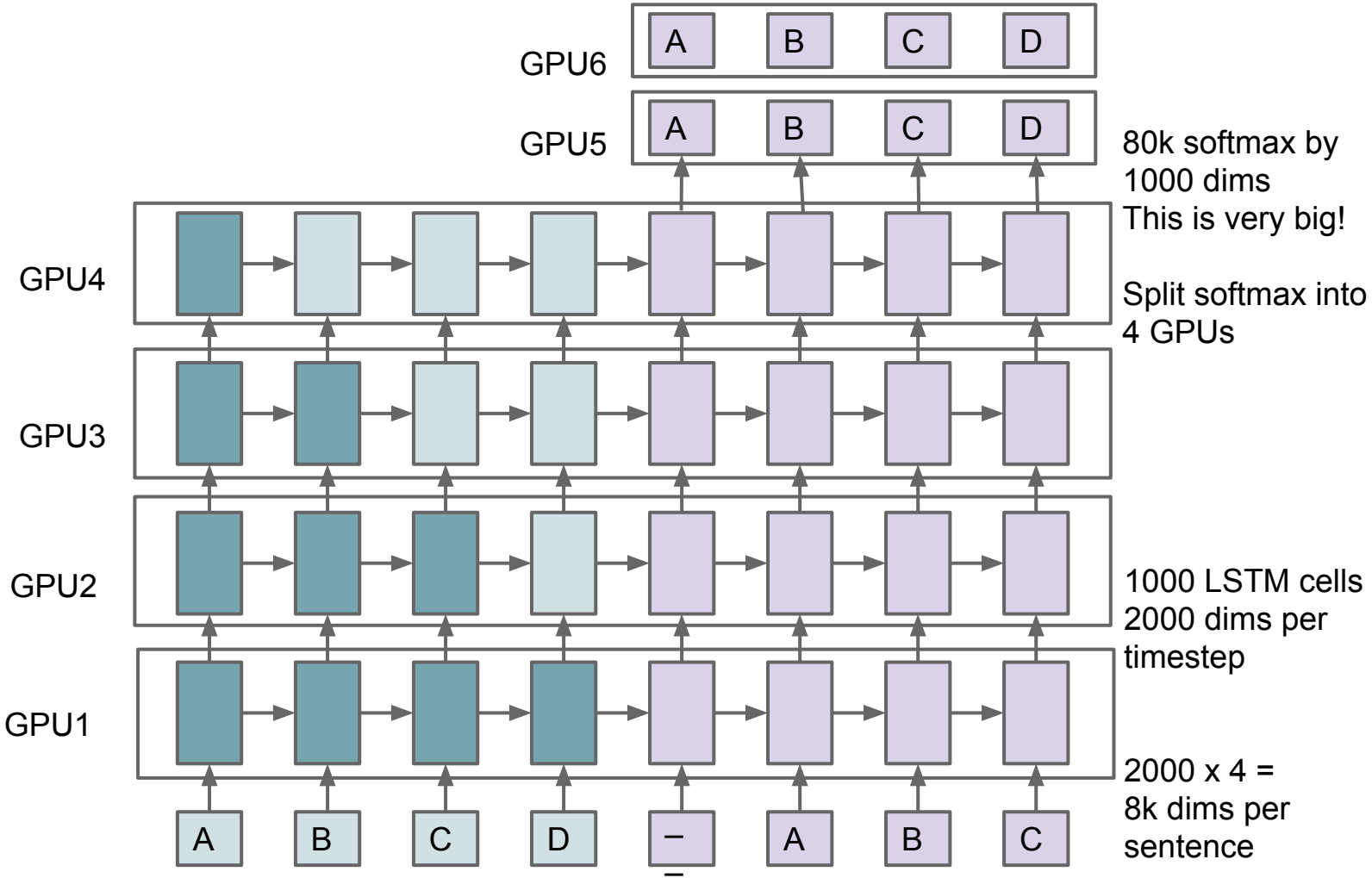
Example: Deep LSTM

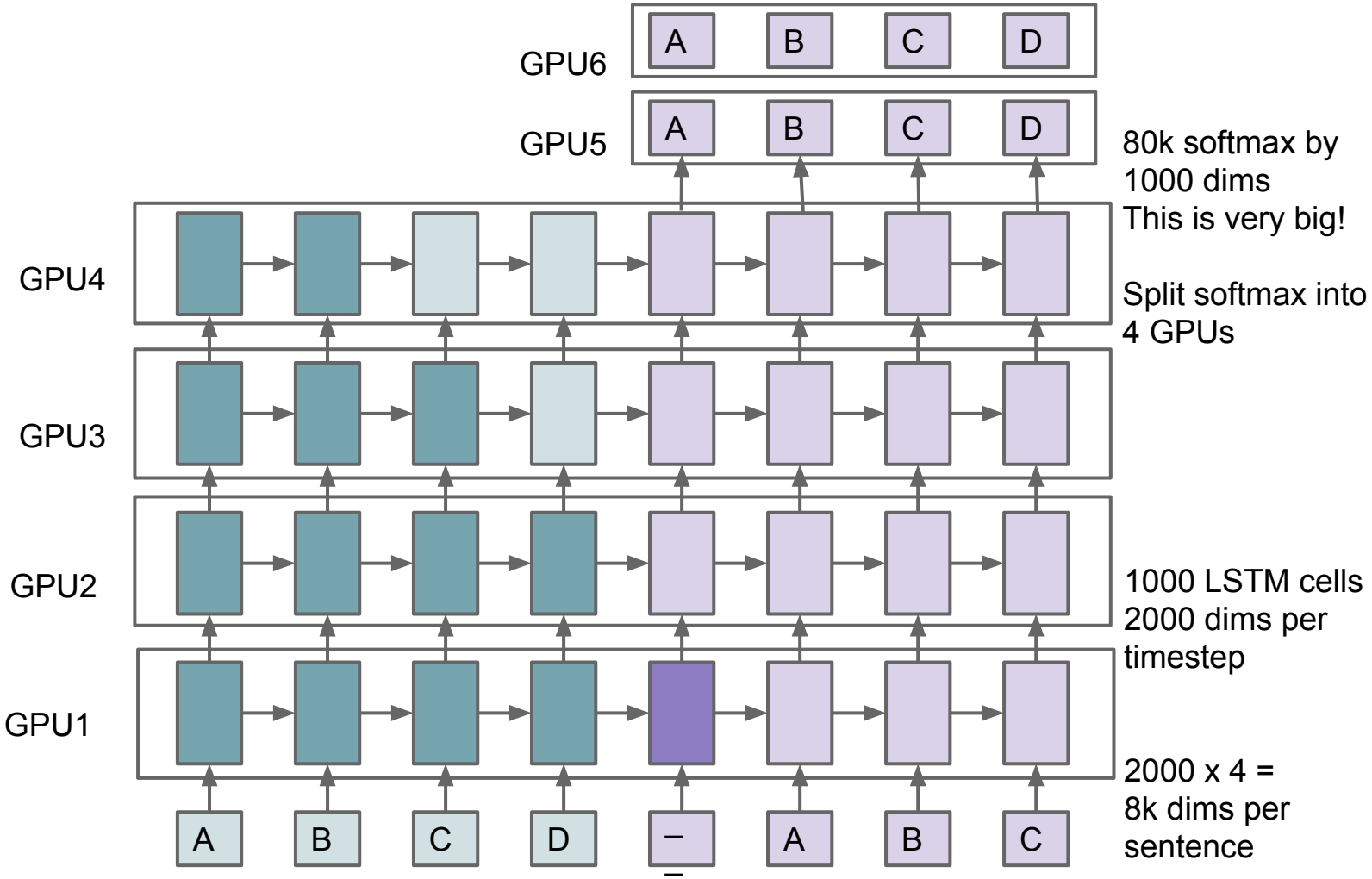
```
for i in range(20):
  for d in range(4): # d is depth
    with tf.device("/gpu:%d" % d):
      input = x[i] if d is 0 else m[d-1]
      m[d], c[d] = LSTMCell(input, mprev[d], cprev[d])
      mprev[d] = m[d]
      cprev[d] = c[d]
```

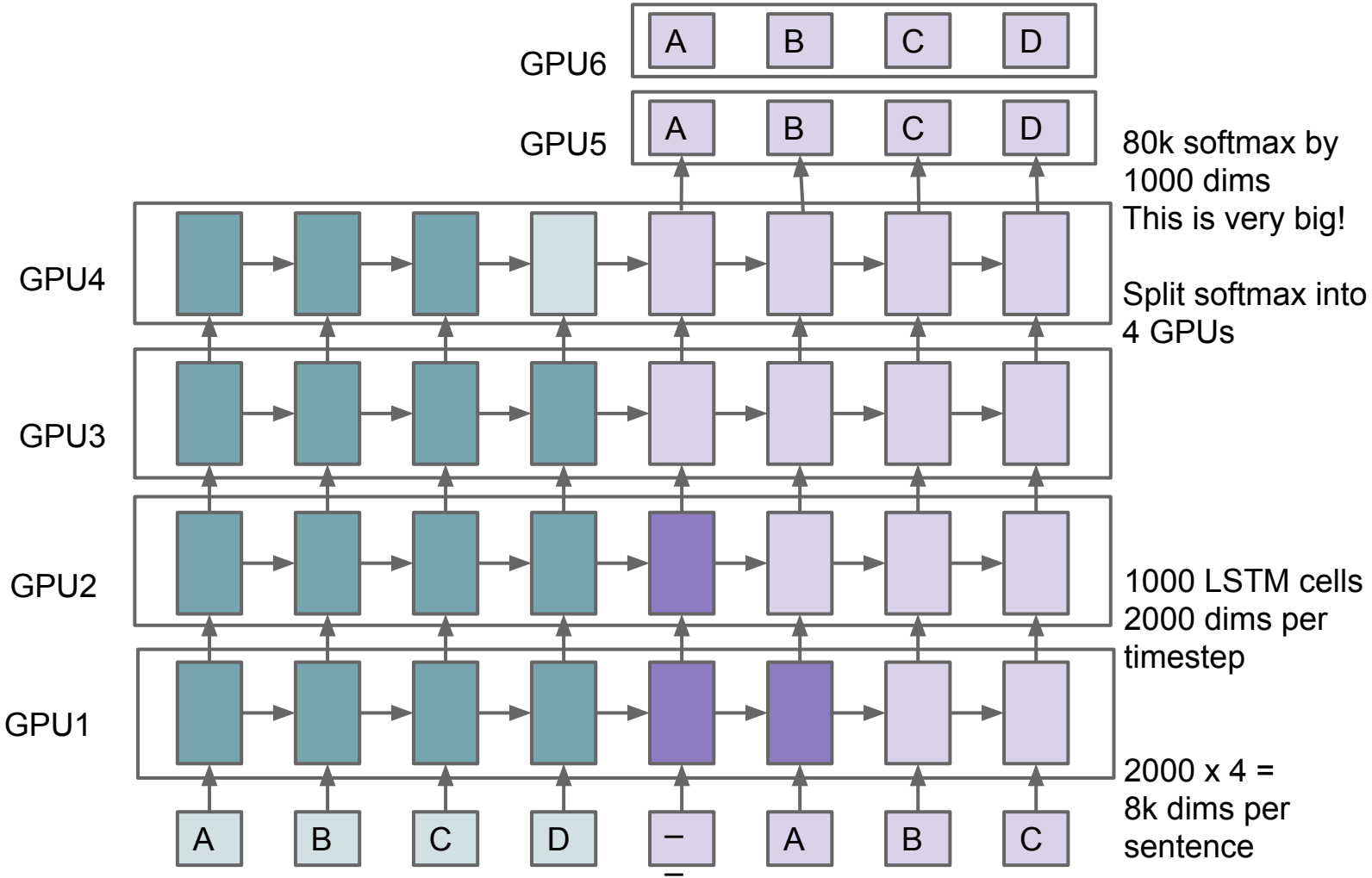

GPU6
A
B
C
D
GPU5
A
B
C
D
80k softmax by
1000 dims
This is very big!
GPU4
Split softmax into
4 GPUs
GPU3
GPU2
1000 LSTM cells
2000 dims per
timestep
GPU1
2000 x 4 =
8k dims per
sentence
A
B
C
D
–
A
B
C

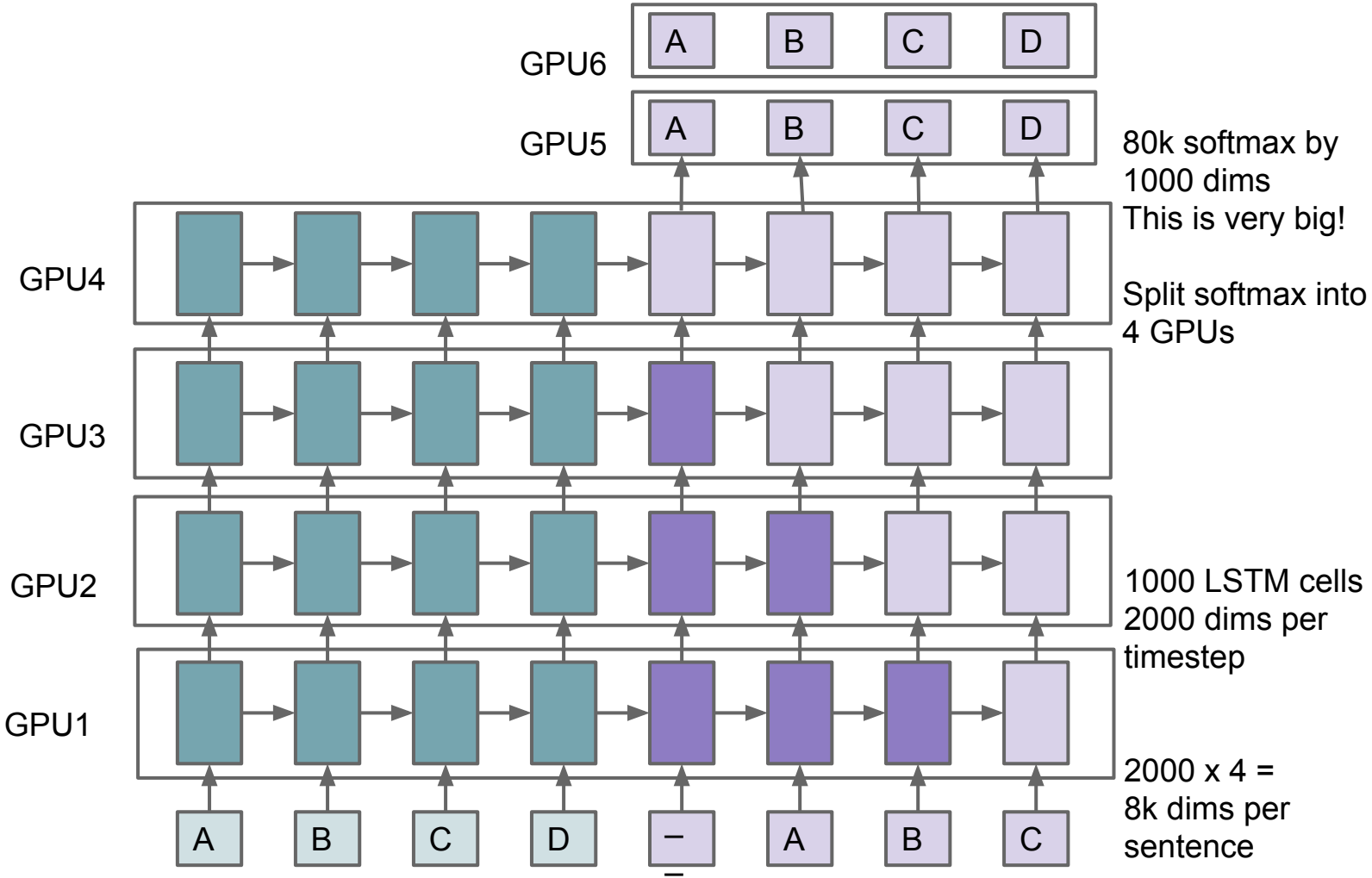

GPU6
A
B
C
D
GPU5
A
B
C
D
80k softmax by
1000 dims
This is very big!
GPU4
Split softmax into
4 GPUs
GPU3
GPU2
1000 LSTM cells
2000 dims per
timestep
GPU1
2000 x 4 =
8k dims per
sentence
A
B
C
D
–
A
B
C

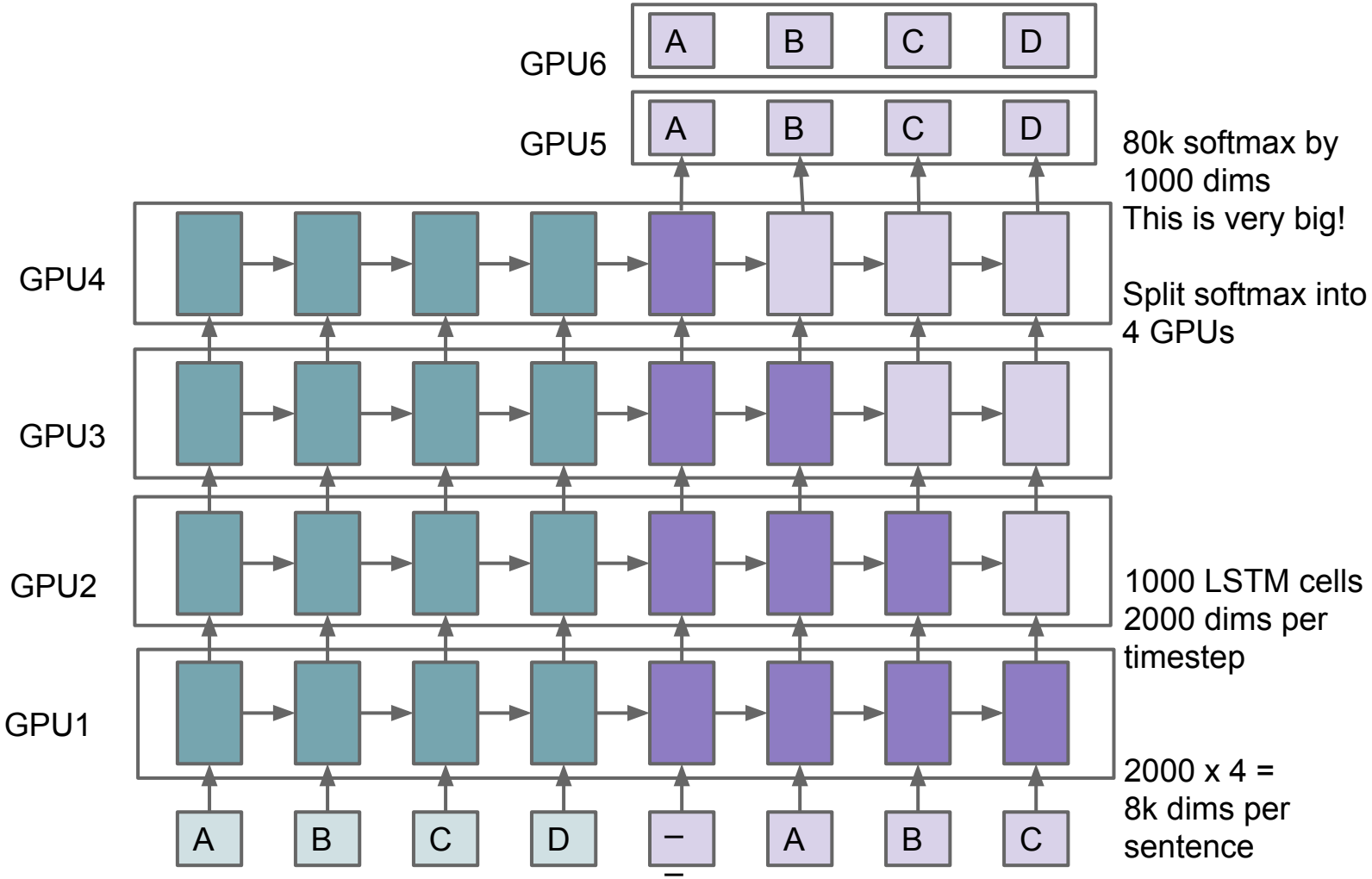

GPU6
GPU5
GPU4
GPU3
GPU2
GPU1
A
B
C
D
–
A
B
C
80k softmax by
1000 dims
This is very big!
Split softmax into
4 GPUs
1000 LSTM cells
2000 dims per
timestep
2000 x 4 =
8k dims per
sentence

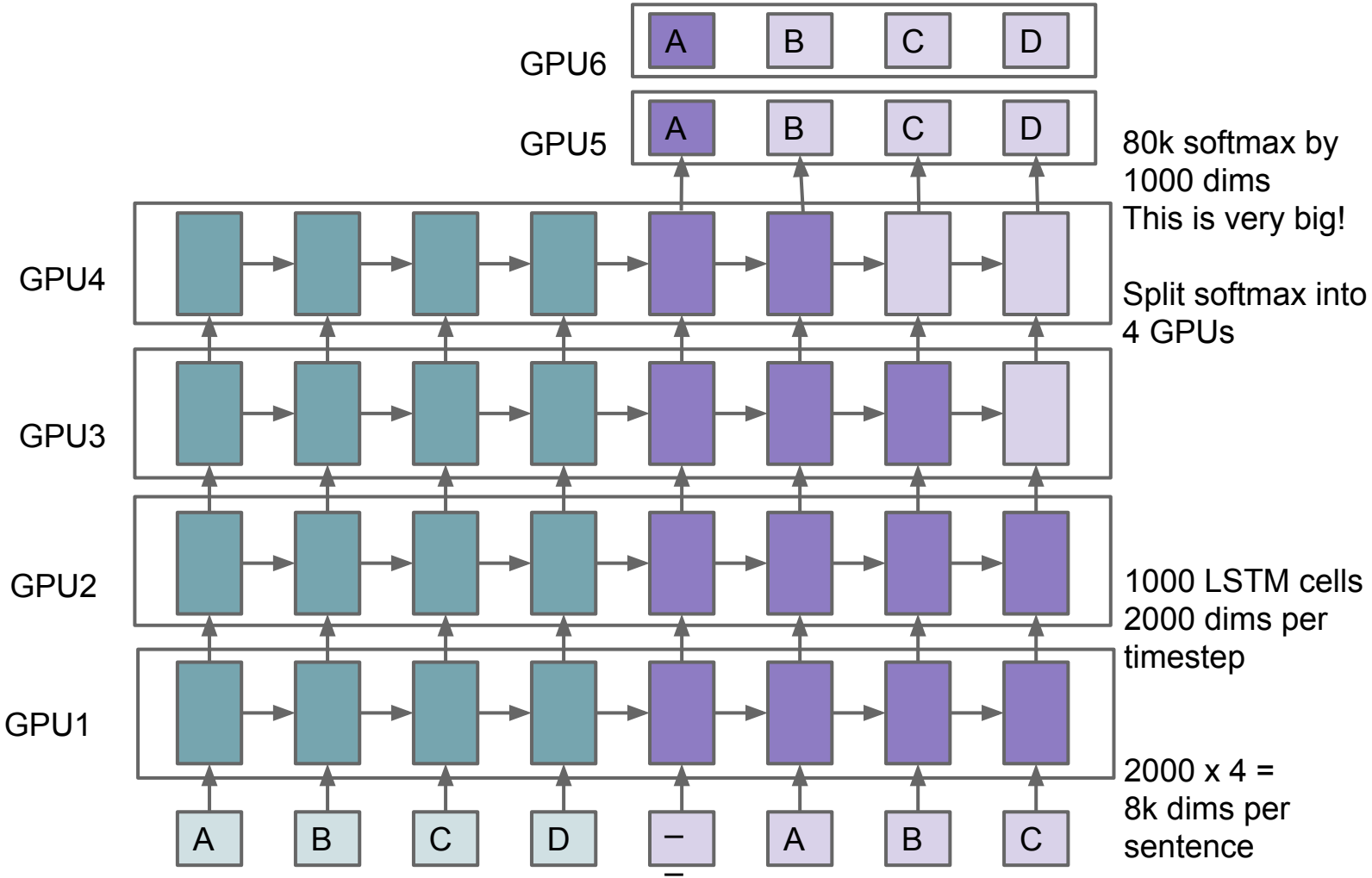

GPU6
A
B
C
D
GPU5
A
B
C
D
80k softmax by
1000 dims
This is very big!
GPU4
Split softmax into
4 GPUs
GPU3
GPU2
1000 LSTM cells
2000 dims per
timestep
GPU1
2000 x 4 =
8k dims per
sentence
A
B
C
D
–
A
B
C

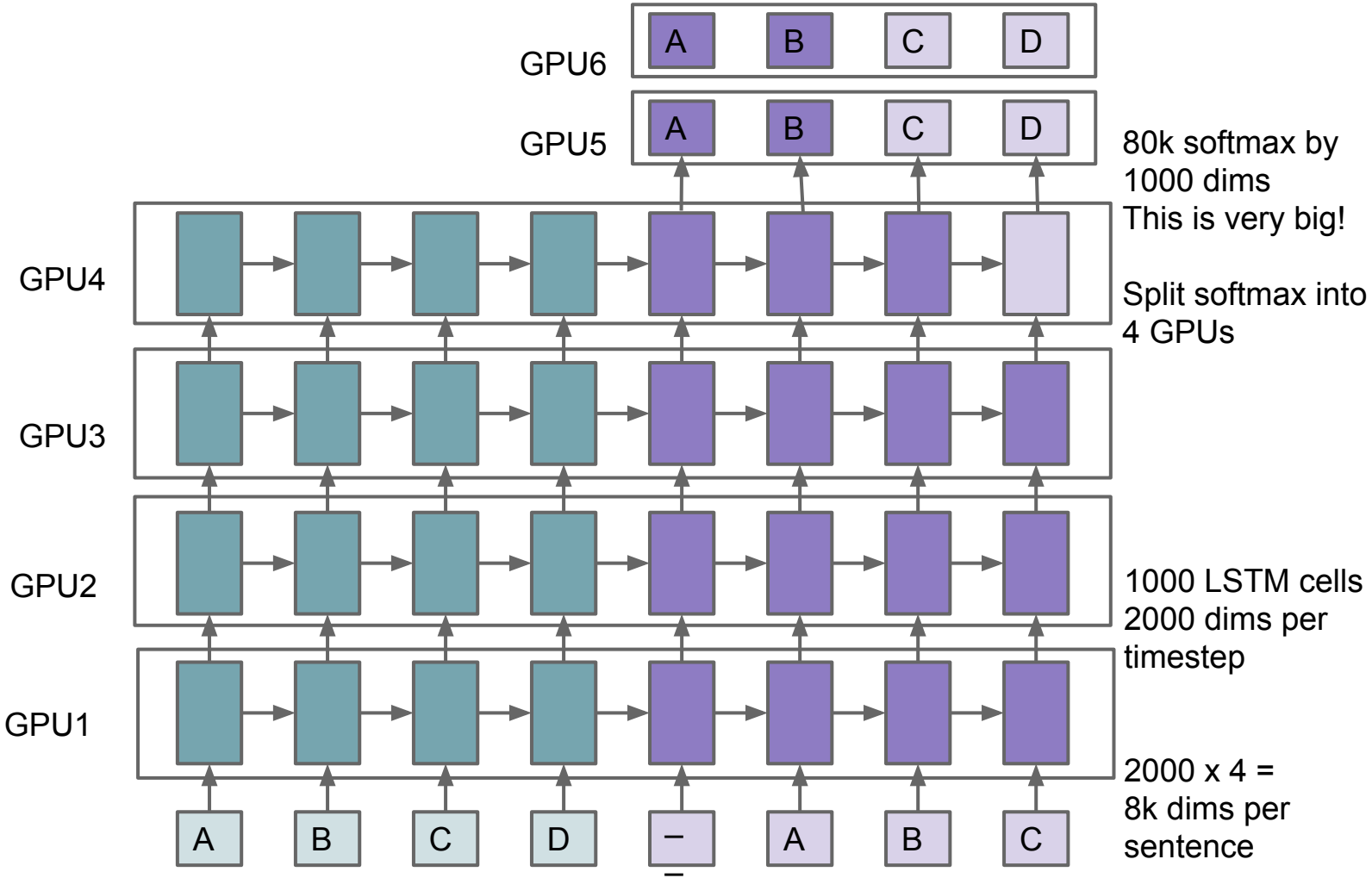

GPU6
GPU5
GPU4
GPU3
GPU2
GPU1
A
B
C
D
–
80k softmax by
1000 dims
This is very big!
Split softmax into
4 GPUs
1000 LSTM cells
2000 dims per
timestep
2000 x 4 =
8k dims per
sentence

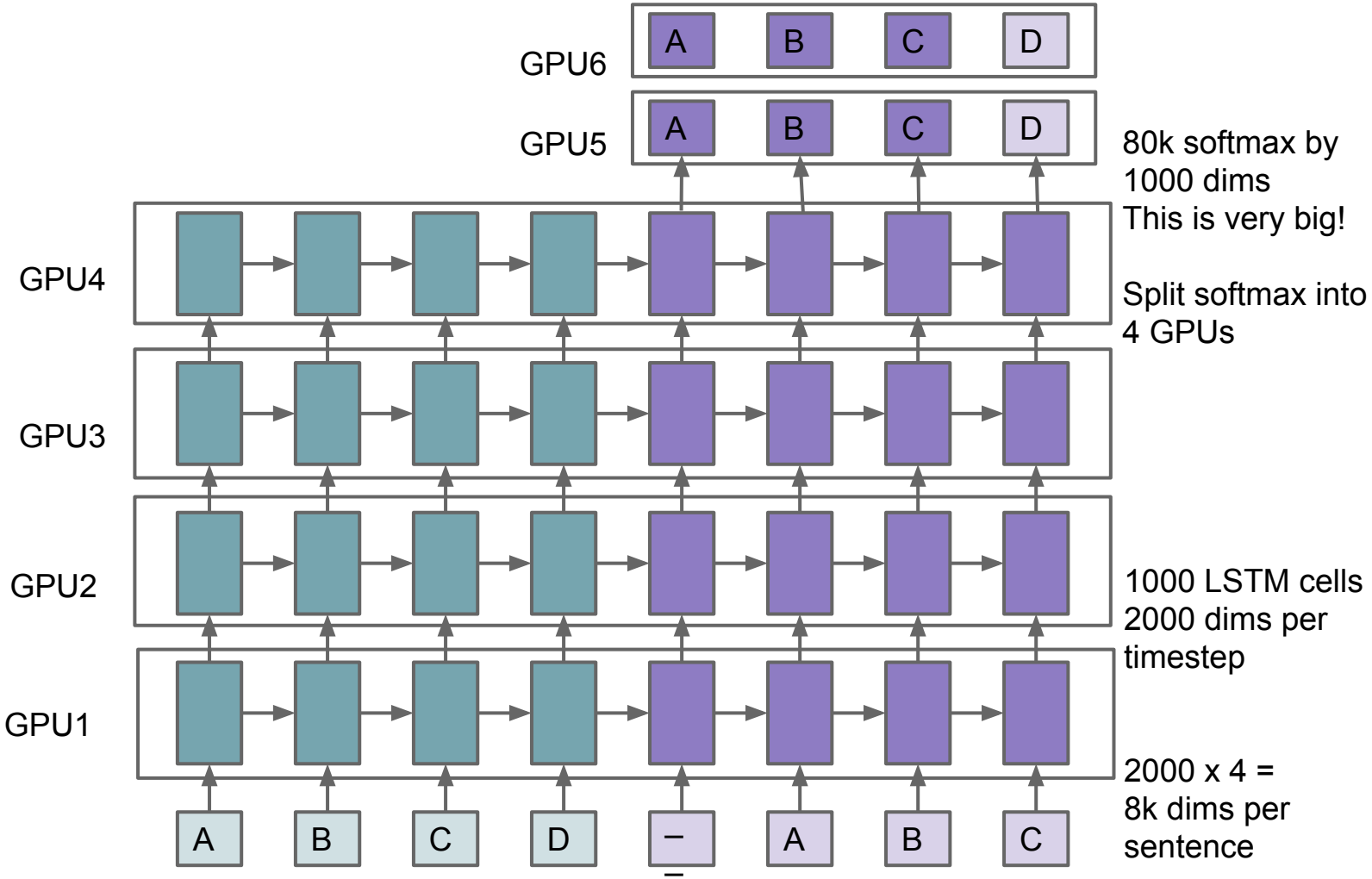

GPU6
A
B
C
D
GPU5
A
B
C
D
80k softmax by
1000 dims
This is very big!
GPU4
Split softmax into
4 GPUs
GPU3
GPU2
1000 LSTM cells
2000 dims per
timestep
GPU1
2000 x 4 =
8k dims per
sentence
A
B
C
D
–
A
B
C

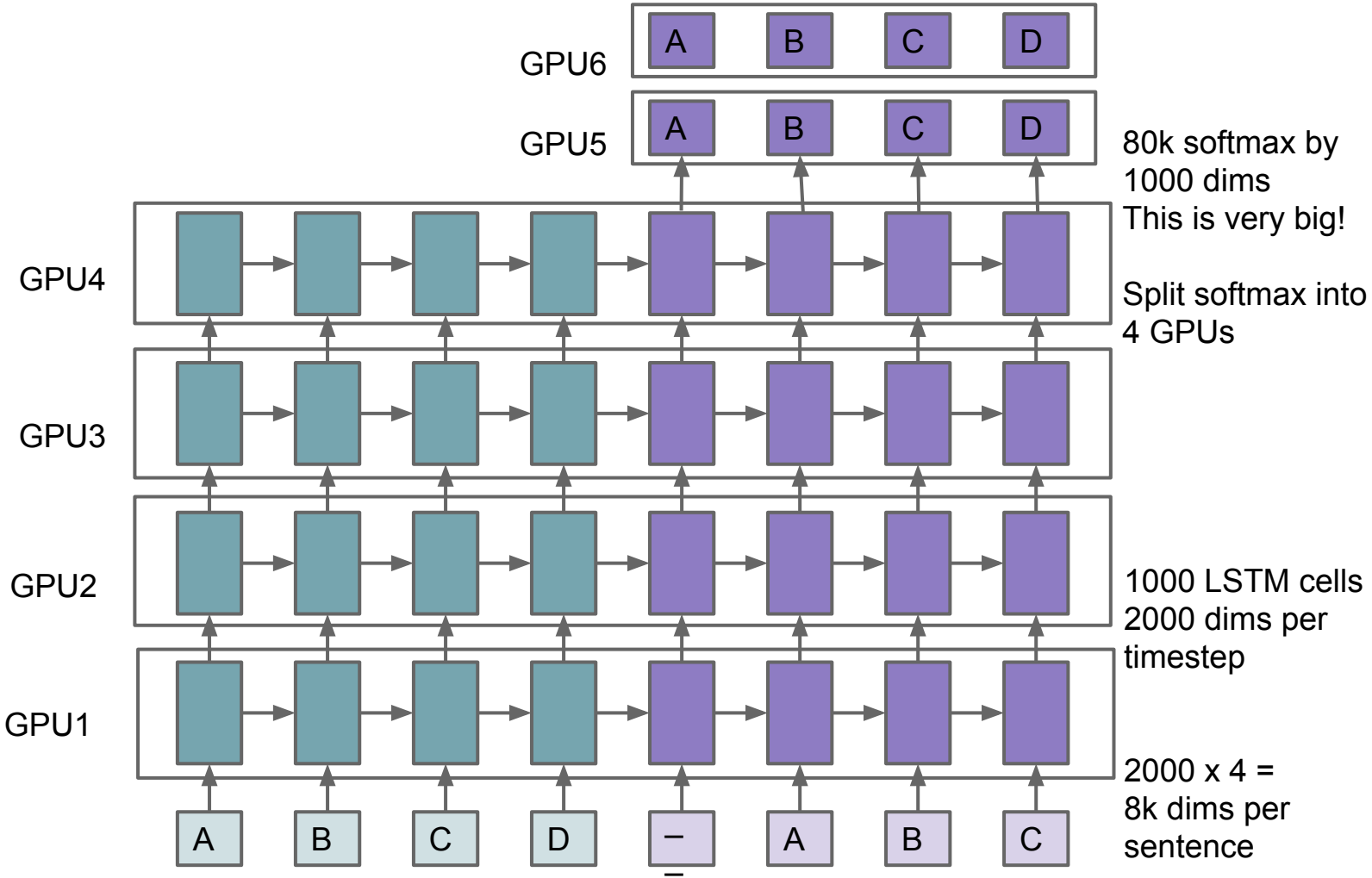

GPU6
A
B
C
D
GPU5
A
B
C
D
80k softmax by
1000 dims
This is very big!
GPU4
Split softmax into
4 GPUs
GPU3
GPU2
1000 LSTM cells
2000 dims per
timestep
GPU1
2000 x 4 =
8k dims per
sentence
A
B
C
D
–
A
B
C


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


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

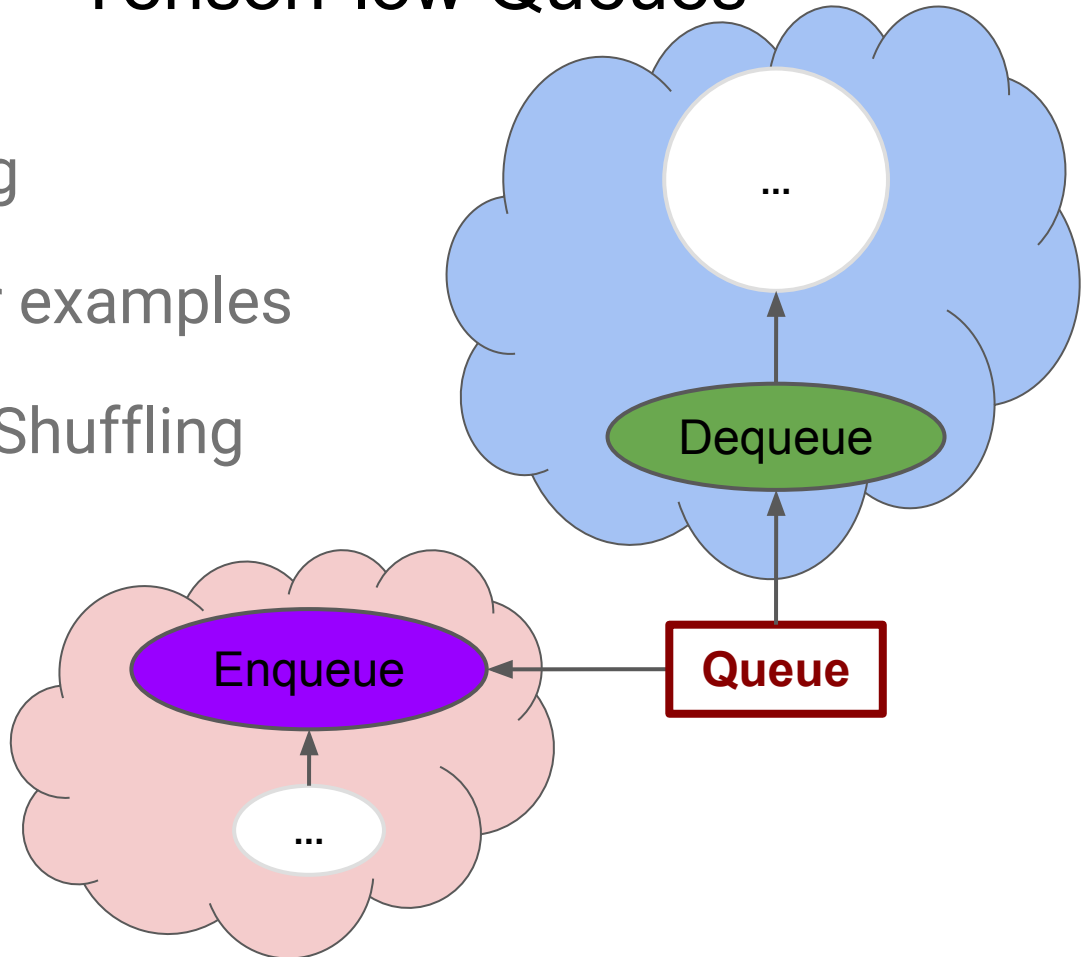

GPU6
A
B
C
D
GPU5
A
B
C
D
80k softmax by
1000 dims
This is very big!
GPU4
Split softmax into
4 GPUs
GPU3
GPU2
1000 LSTM cells
2000 dims per
timestep
GPU1
2000 x 4 =
8k dims per
sentence
A
B
C
D
–
A
B
C

TensorFlow Queues

Input prefetching

Grouping similar examples

Randomization/Shuffling



Example: Deep LSTMs

- Wrinkles
  - Bucket sentences by length using a queue per length
  - Dequeue when a full batch of same length has accumulated
  - N different graphs for different lengths
  - Alternative: while loop

Expressing Data Parallelism

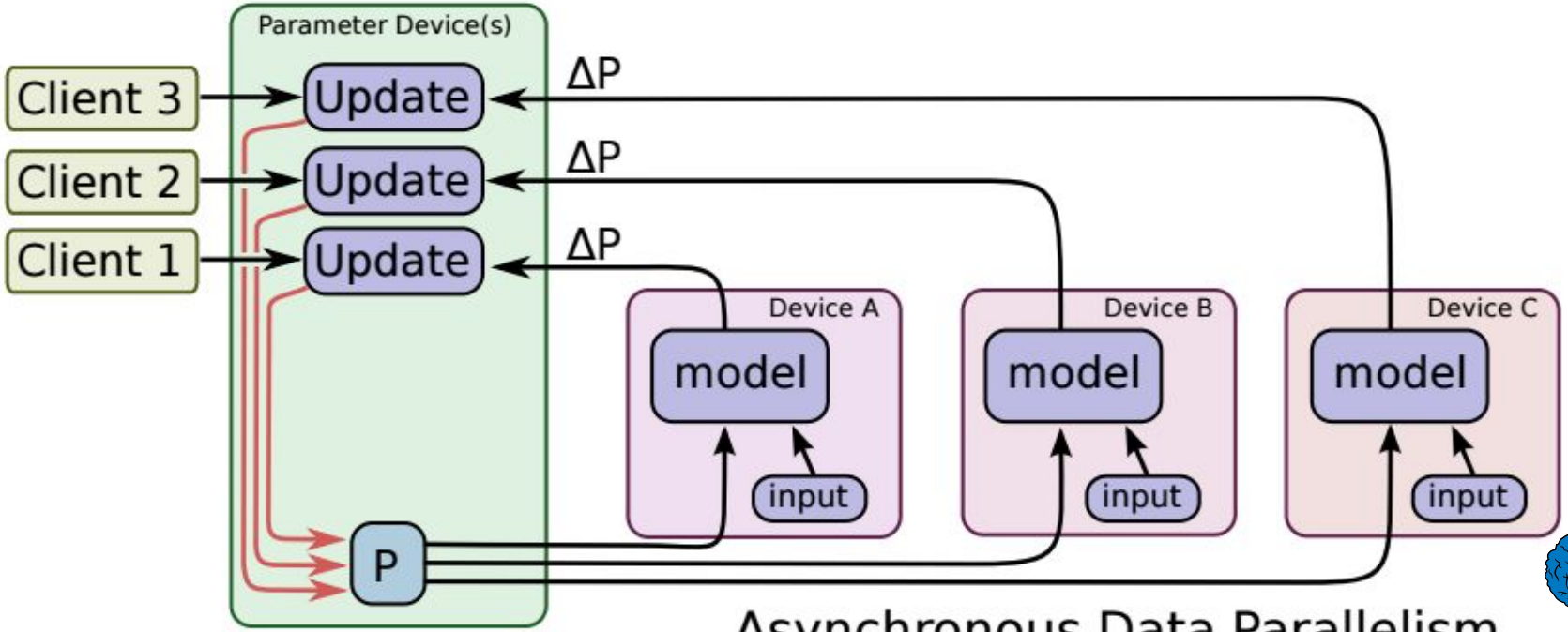
```
# We use the ReplicaDeviceSetter() device function to automatically
# assign Variables to the 'ps' jobs.
with tf.device(“/cpu:0”):
    # Create the Mnist model.
    model = MnistModel(batch_size=16, hidden_units=200)

    # Get an initialized, and possibly recovered 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])
      if local_step % 1000 == 0:
        print "step %d: %g" % (step, loss)
```

Expressing Data Parallelism

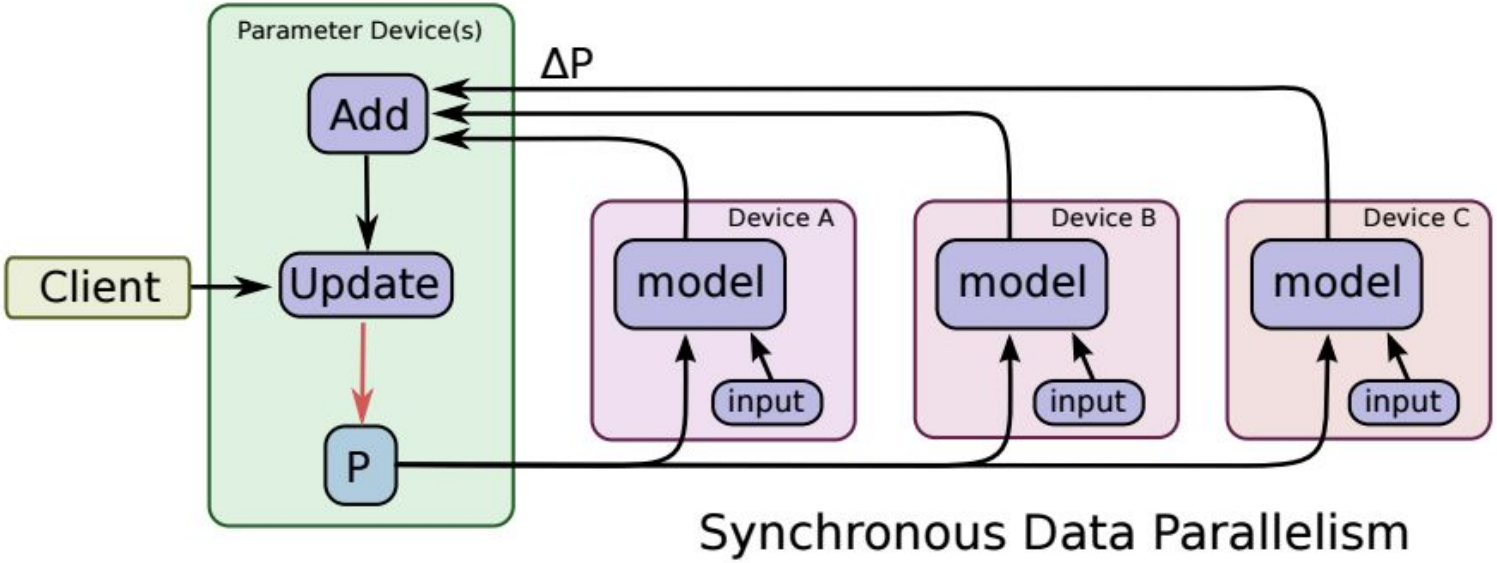
```
# We use the ReplicaDeviceSetter() device function to automatically
# assign Variables to the 'ps' jobs.
with tf.device(tf.ReplicaDeviceSetter(parameter_devices=10)):
    # Create the Mnist model.
    model = MnistModel(batch_size=16, hidden_units=200)

    # 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
      if local_step % 1000 == 0:
        print "step %d: %g" % (step, loss)
```

Asynchronous Training

- Unlike DistBelief, no separate parameter server system:
  - Parameters are now just stateful nodes in the graph

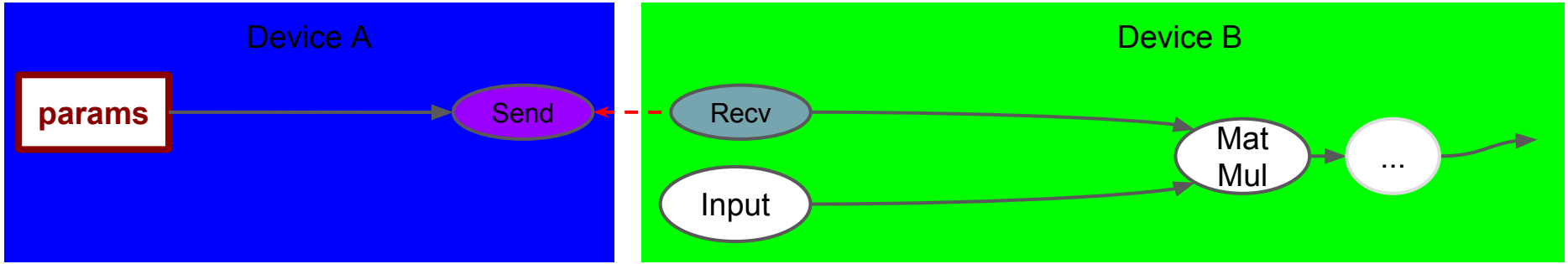


Synchronous Variant



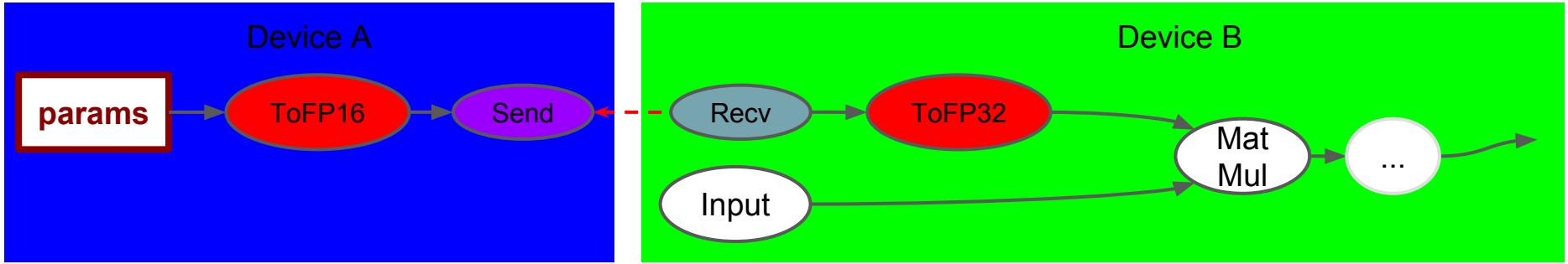
Network Optimizations

- Neural net training very tolerant of reduced precision
- e.g. drop precision to 16 bits across network



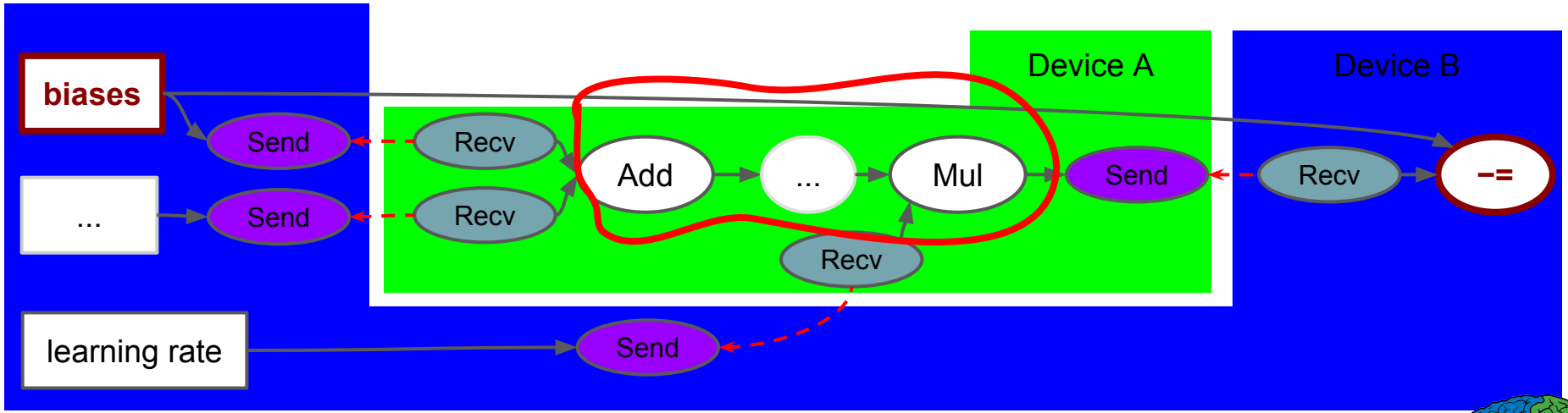
Network Optimizations

- Neural net training very tolerant of reduced precision
- e.g. drop precision to 16 bits across network



Subgraph Compiler

- Compile small subgraphs together to generate optimized routine
- Dynamic compiler with caching so sizes are known



Devices: Processes, Machines, GPUs, etc

Quantization for Inference

- Need even less precision for inference
- 8-bit fixed point works well, but many ways of quantizing
- Critical for things like mobile devices
  - w/quantization, high-end smart phone can run Inception model at >6 frames per second (fps)

Open Source Status for Distributed TensorFlow

Multi GPU in single machine already in open source release

- See 4-GPU CIFAR10 training example in repository

Distributed implementation coming soon:

- GitHub tracking issue: github.com/tensorflow/tensorflow/issues/23

Concluding Remarks

- Model and Data Parallelism enable great ML work:
  - Neural Machine Translation: ~6x speedup on 8 GPUs
  - Inception / Imagenet: ~40x speedup on 50 GPUs
  - RankBrain: ~300X speedup on 500 machines
- A variety of different parallelization schemes are easy to express in TensorFlow

Concluding Remarks

- 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

- 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

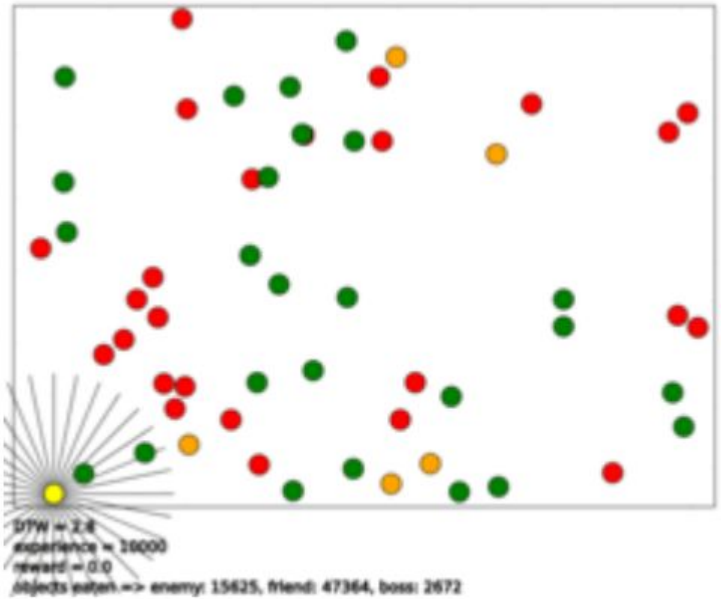
...

github.com/nivwusquorum/tensorflow-deepq

Reinforcement Learning using Tensor Flow

Quick start

Check out Karpathy game in `notebooks` folder.



The image above depicts a strategy learned by the DeepQ controller. Available actions are accelerating top, bottom, left or right. The reward signal is +1 for the green fellas, -1 for red and -5 for orange.

github.com/woodrush/neural-art-tf

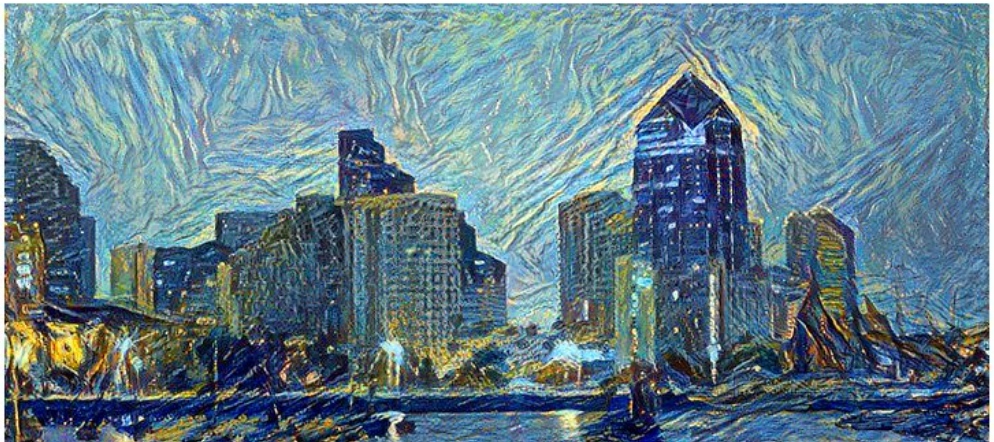
"Neural Art" in TensorFlow

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.

In this work, I put effort in putting the code simple as possible, for being a good introductory code to TF. For this reason, I also implemented very basic uses of TensorBoard (the visualizer). I also aimed on demonstrating the use of importing various Caffe models from *.caffemodel files into TensorFlow, especially models that seemed not to be imported by anybody yet in TF (as far as I know). Based on https://github.com/ethereon/caffe-tensorflow, I modified the importer so that it can import illustration2vec (http://illustration2vec.net/), which is another CNN available as a Caffe model. Using different CNNs yields different results, which reflects the characteristics of the model.

In the Neural Art problem setting, the weights of the CNN are fixed, and the input image into the CNN is the only "trainable" variable, making the code easy to understand (the optimized/trained image is the output image). I hope this example serves as a good introduction to TensorFlow as well as for entertainment purposes.



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)

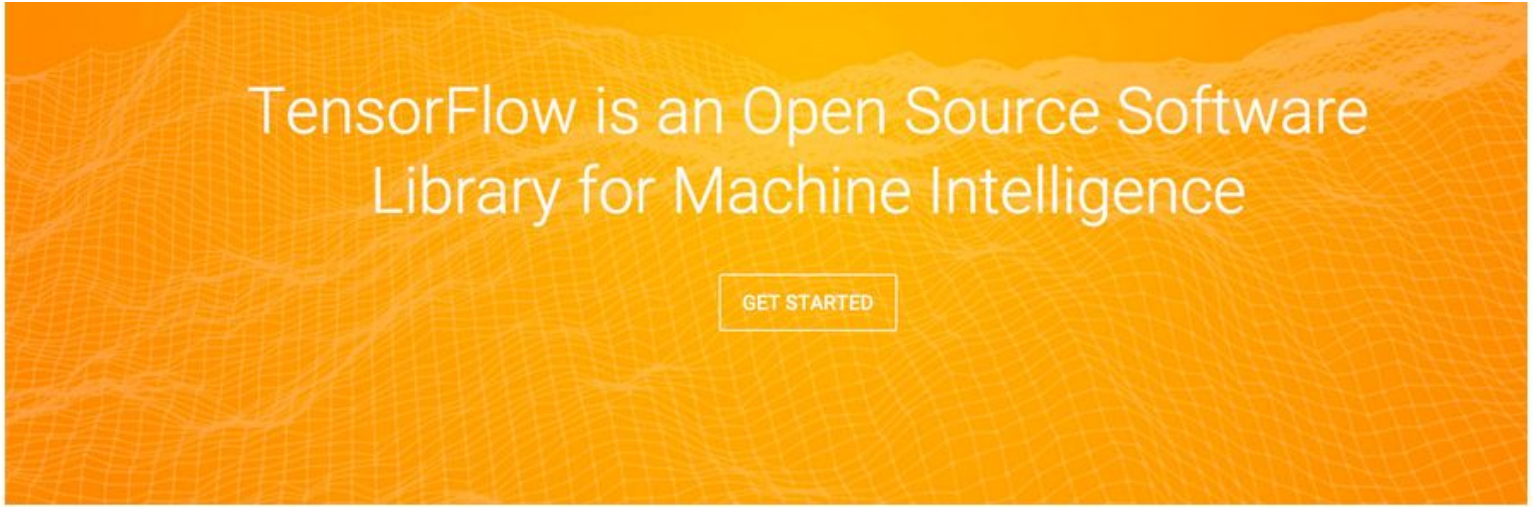
Code

- 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

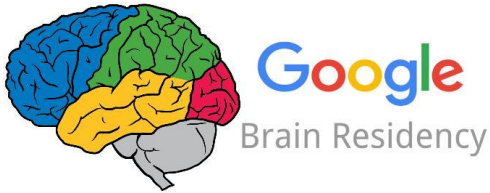


你正在翻译的项目可能会比 Android 系统更加深远地影响着世界！

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

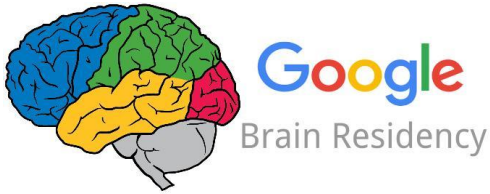


Google Brain Residency Program

New one year immersion program in deep learning research

Learn to conduct deep learning research w/experts in our team

- Fixed one-year employment with salary, benefits, ...
- Goal after one year is to have conducted several research projects
- Interesting problems, TensorFlow, and access to computational resources



Google Brain Residency Program

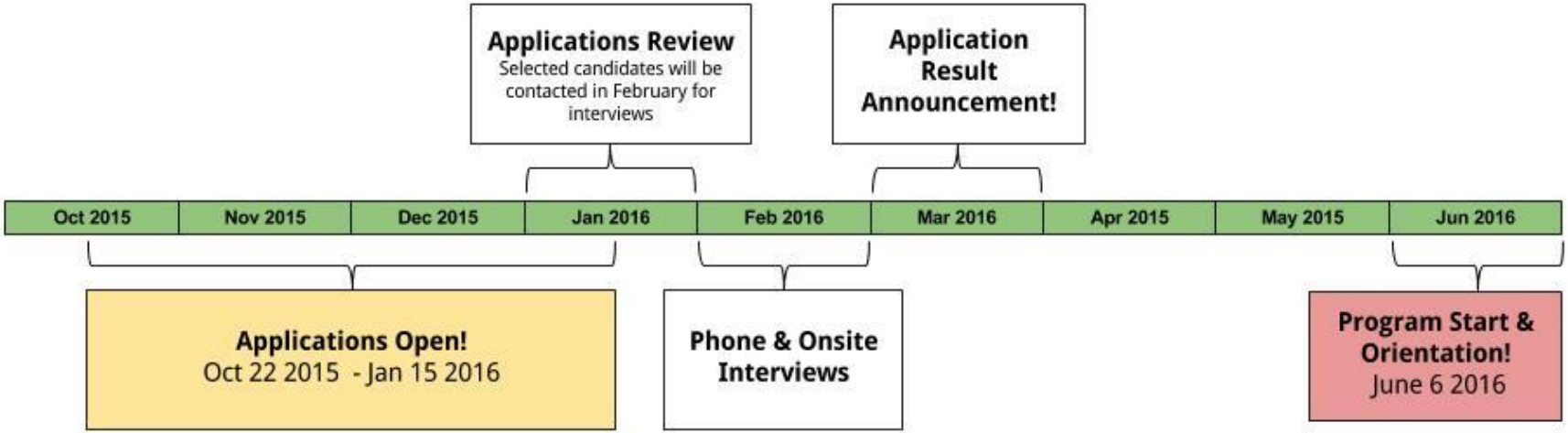
Who should apply?

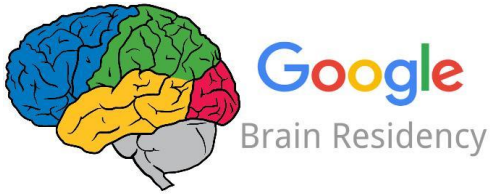
- people with BSc, MSc or PhD, ideally in CS, mathematics or statistics
- completed coursework in calculus, linear algebra, and probability, or equiv.
- programming experience
- motivated, hard working, and have a strong interest in deep learning

Google Brain Residency Program

Program Application & Timeline

DEADLINE: January 15, 2016





Google Brain Residency Program

For more information:

g.co/brainresidency

Contact us:

brain-residency@google.com