TENSORFLOW

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng, Google Brain

ARPIT AGGARWAL (2016MCS2653)

RAJAT GUPTA (2016MCS2673)

Credits: Jeff Dean

OVERVIEW

- INTRODUCTION
- BACKGROUND & MOTIVATION
  - DistBelief
  - Design Principles
  - Related Work
- Execution Model
- Extensibility Case Studies
- Implementation
- Evaluation - Caffe, Torch

TENSORFLOW - ABSTRACT

- A machine learning system that operates at large scale and in heterogeneous environments.
- Uses dataflow graphs to represent computation, shared state, and the operations that mutate that state.
- Maps the nodes of a dataflow graph across many machines in a cluster, and within a machine across multiple computational devices, including multicore CPUs, general- purpose GPUs, and Tensor Processing Units (TPUs).

INTRODUCTION

- DistBelief - first generation machine learning system.

- Uses GPUs for fast training of models.

- Dataflow graph to represent computation and state on which algorithm operates.

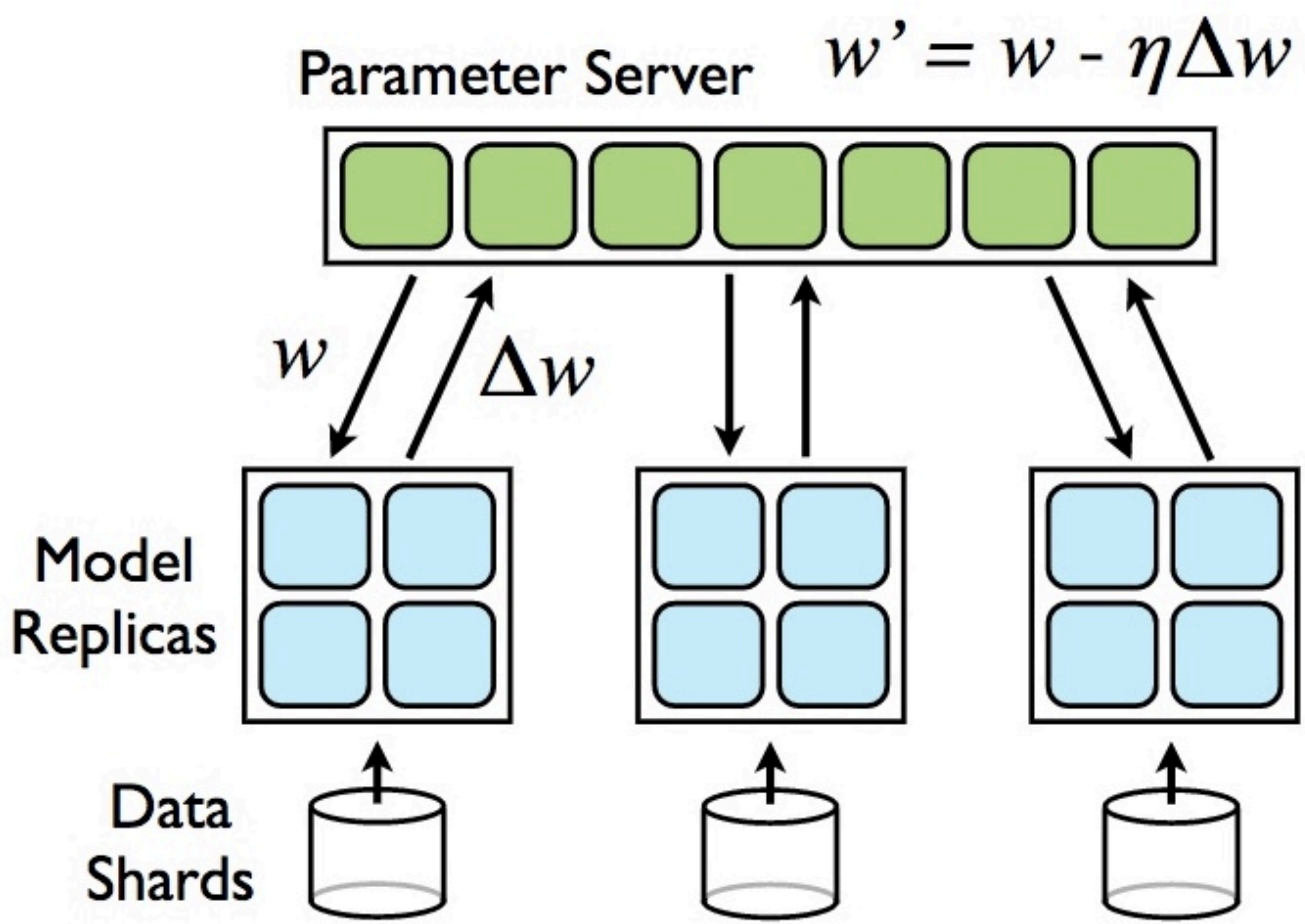
- Parameter servers (low-level efficiency).

BACKGROUND

DISTBELIEF

- Parameter Servers
- DAG structure and backpropogation
- Data Parallelism
- Ad-hoc for Deep Neural Network Algorithms.

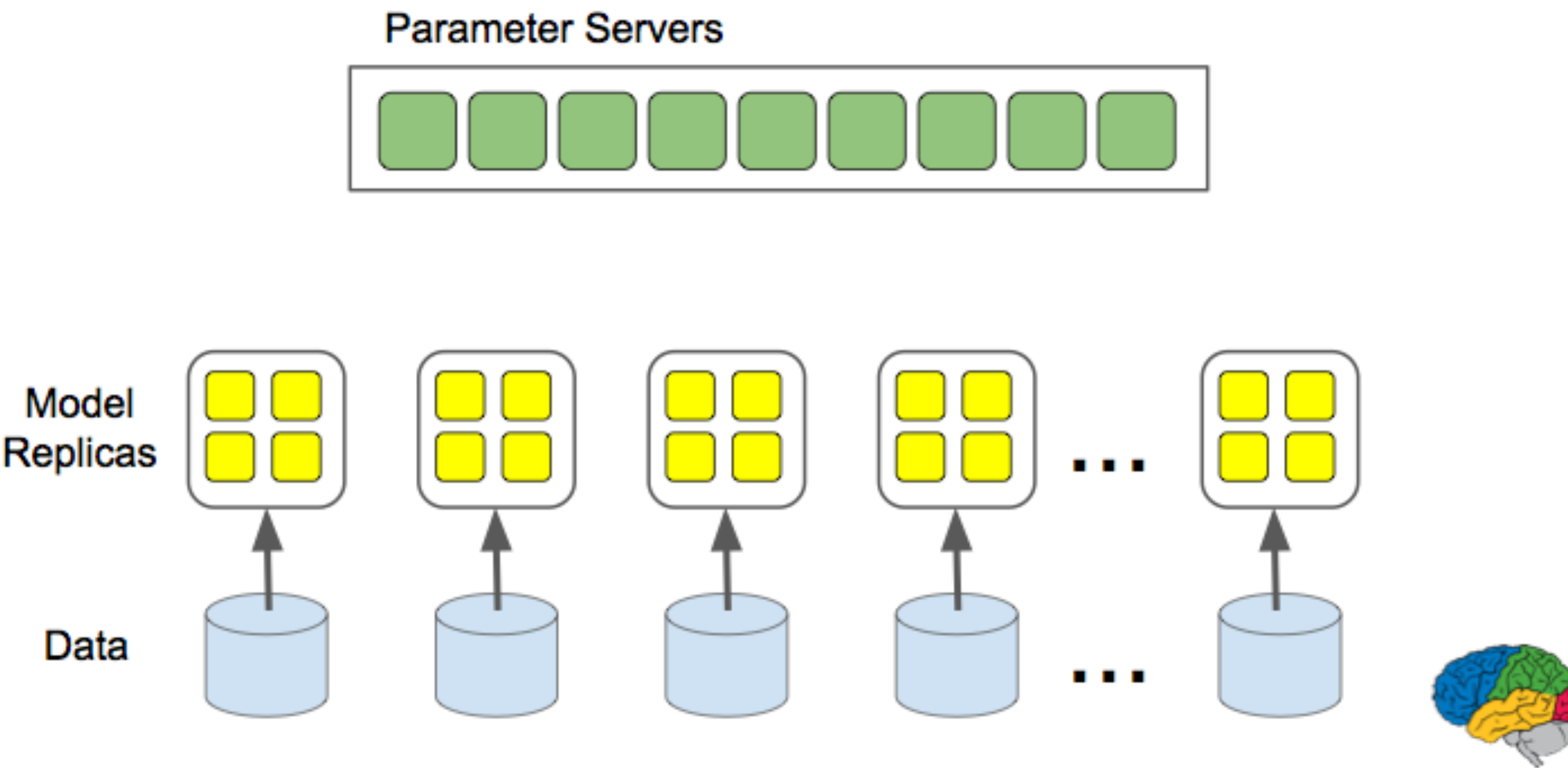
PARAMETER SERVER



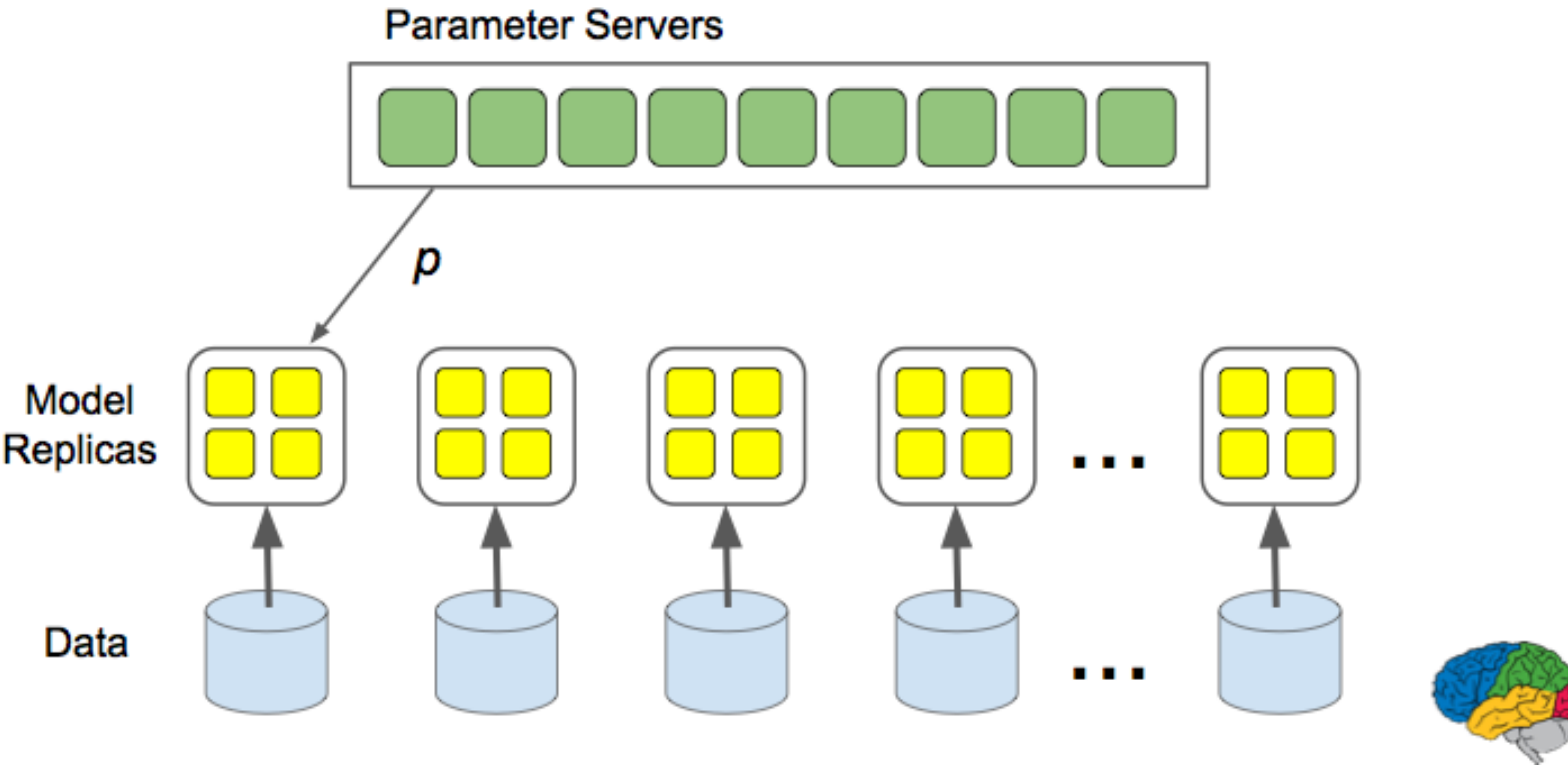
Data Parallelism

- 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

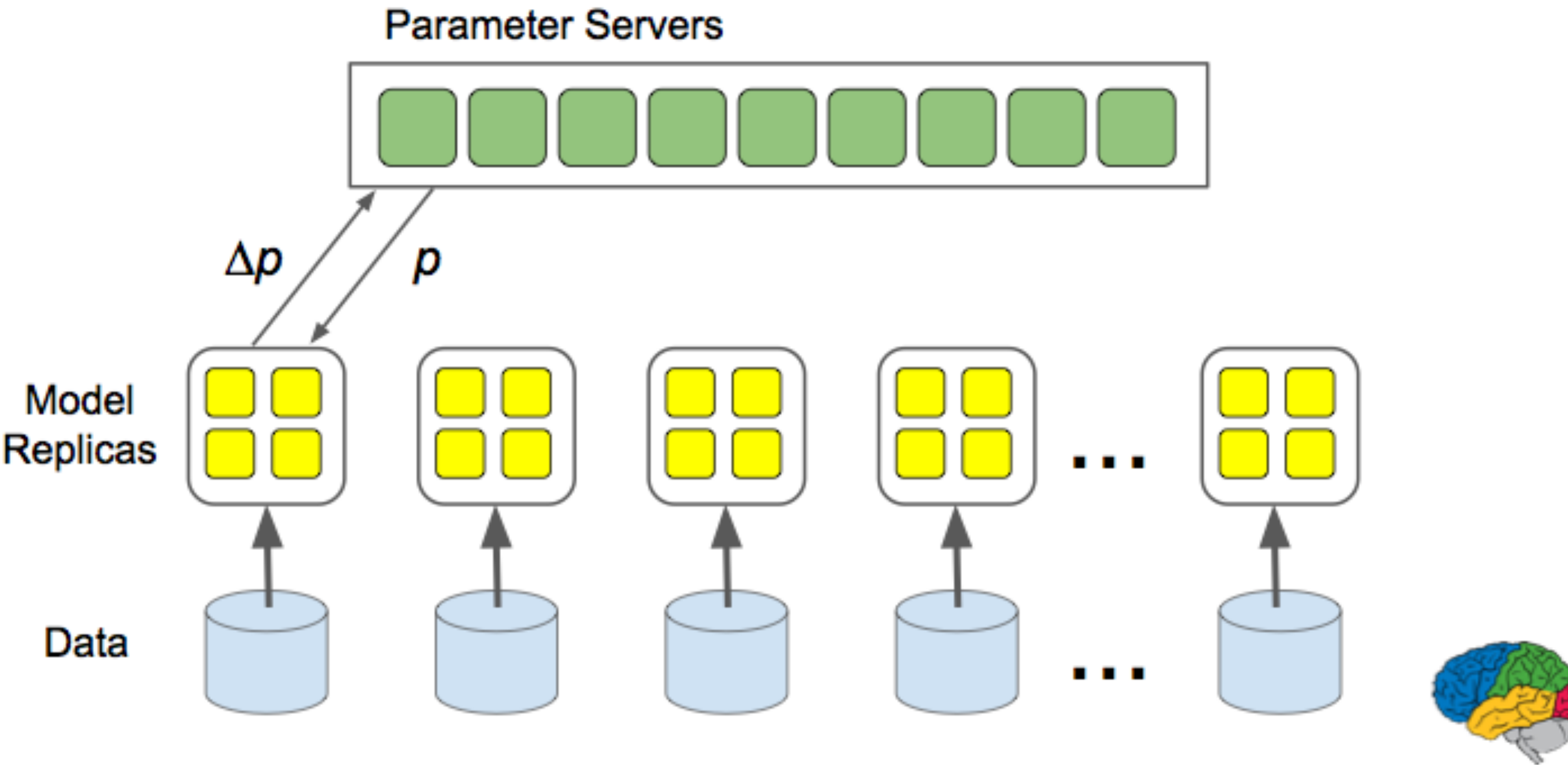
Data Parallelism



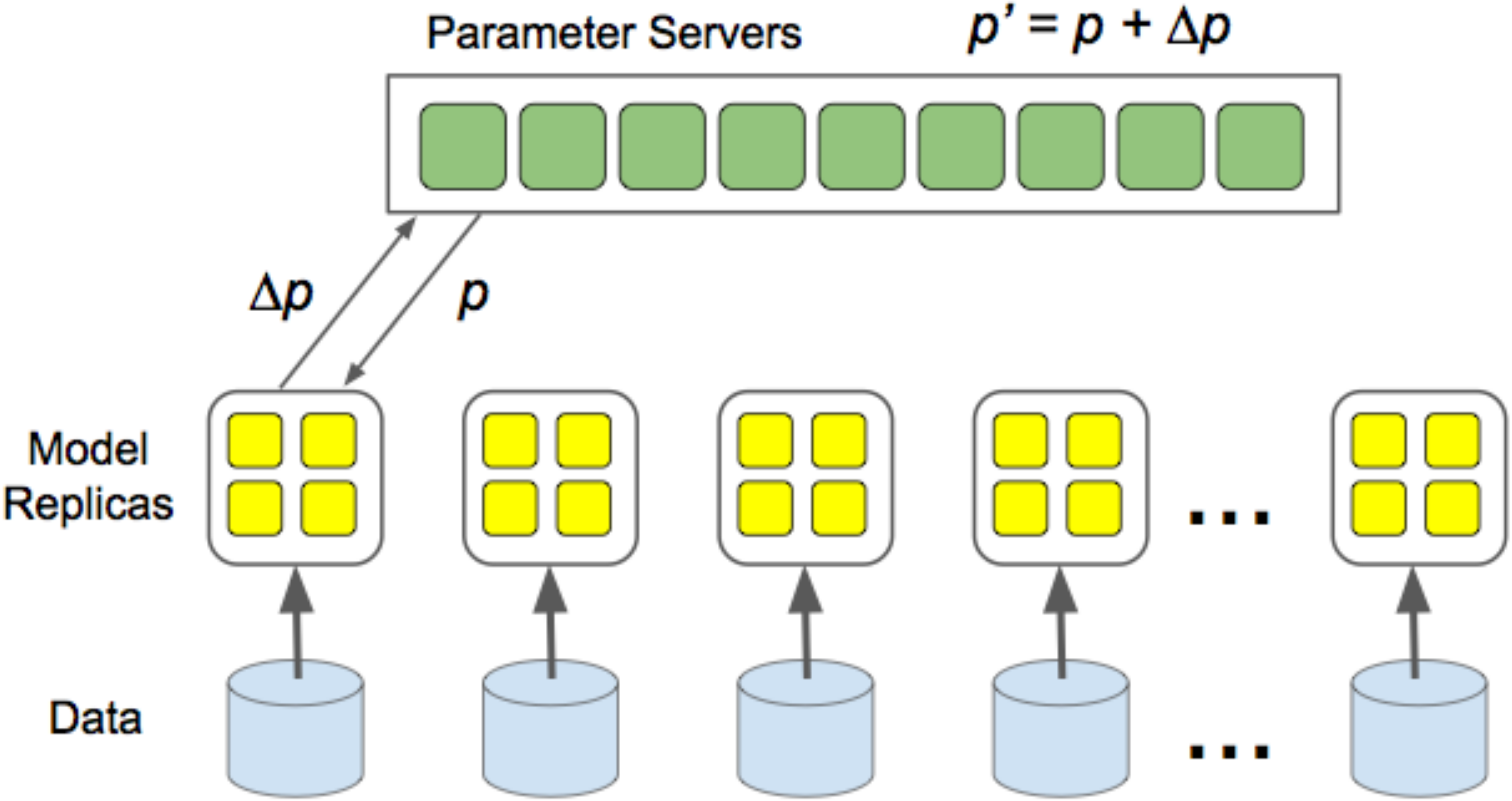
Data Parallelism



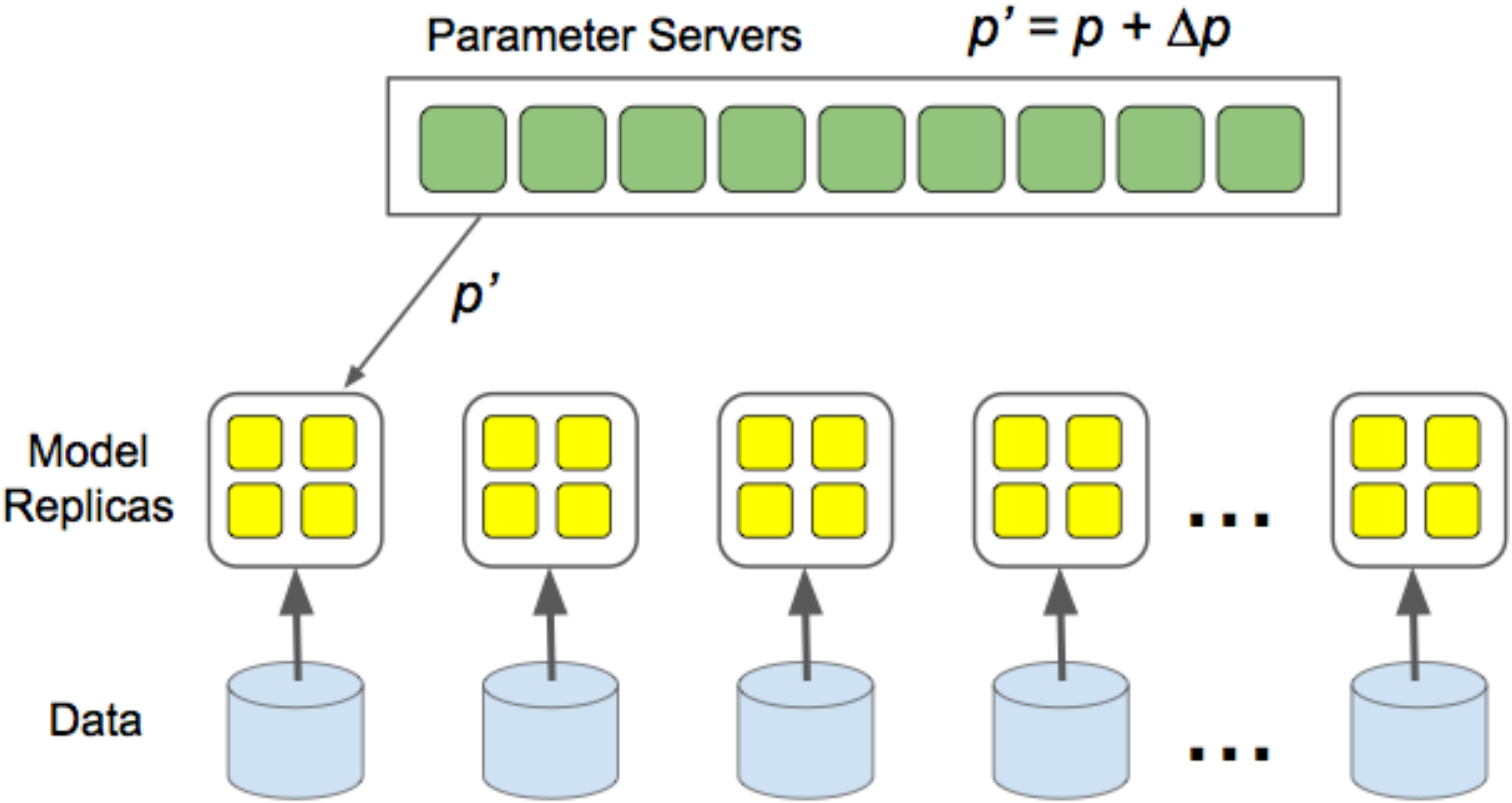
Data Parallelism



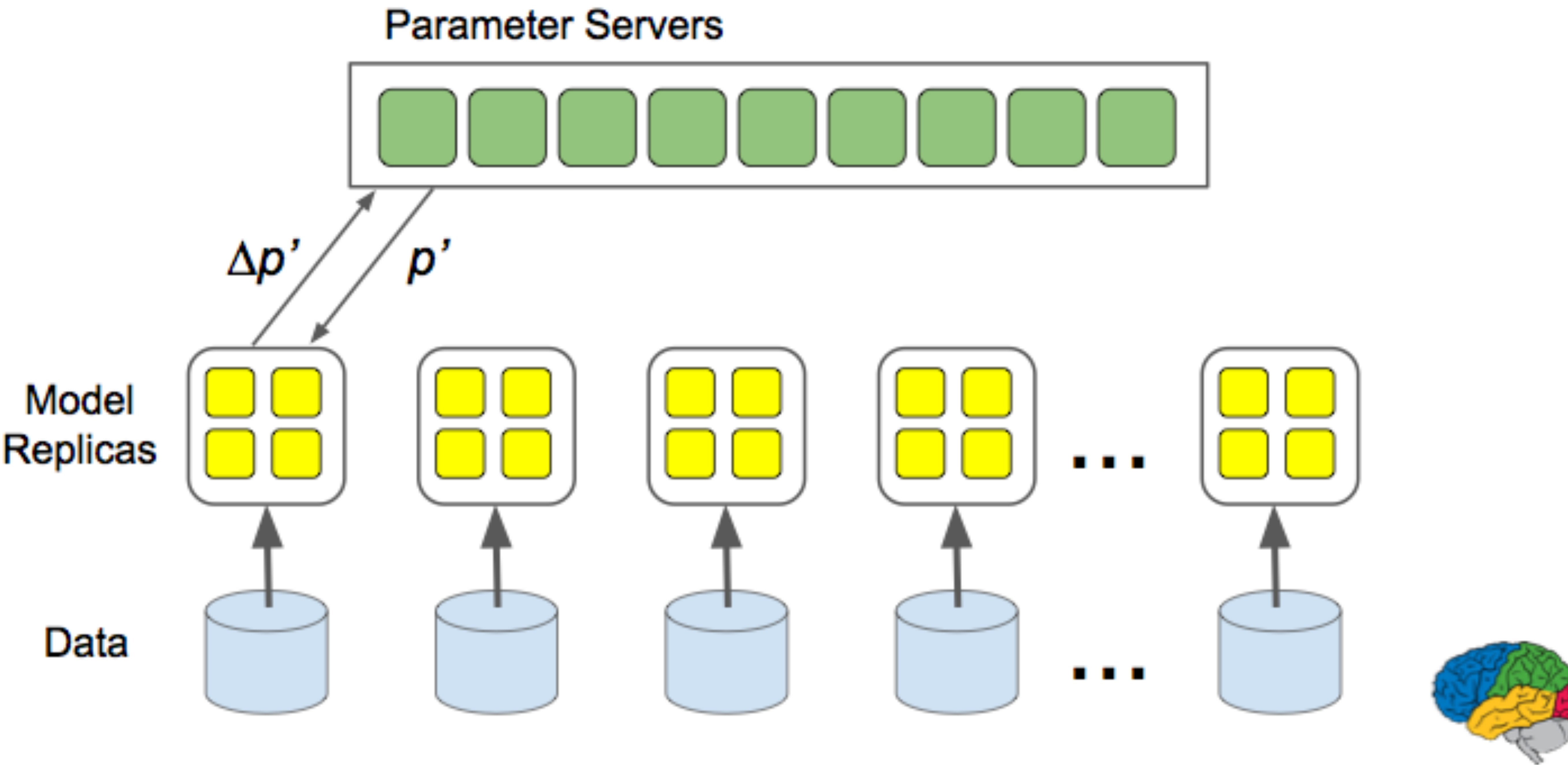
Data Parallelism



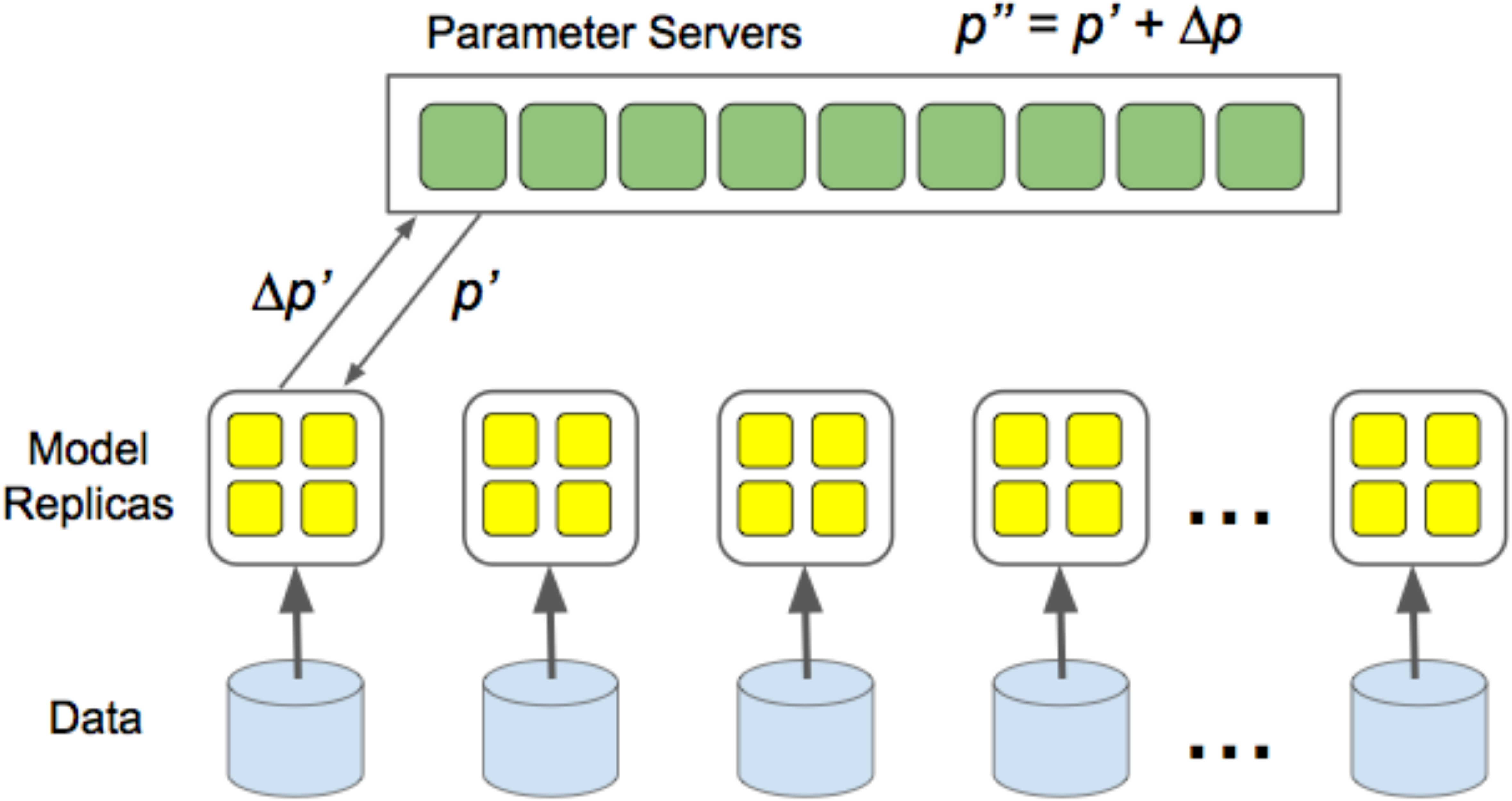
Data Parallelism



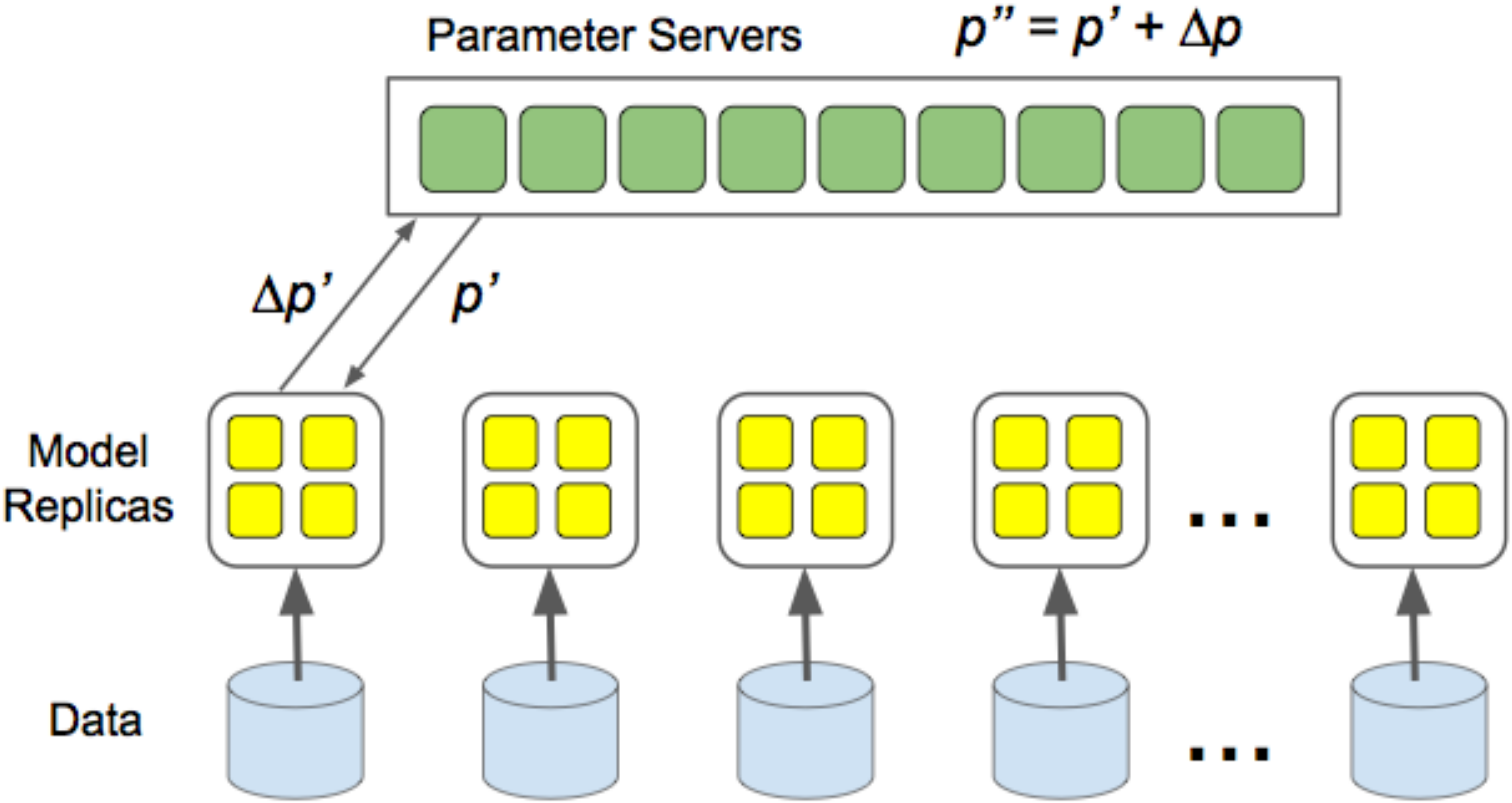
Data Parallelism



Data Parallelism



Data Parallelism

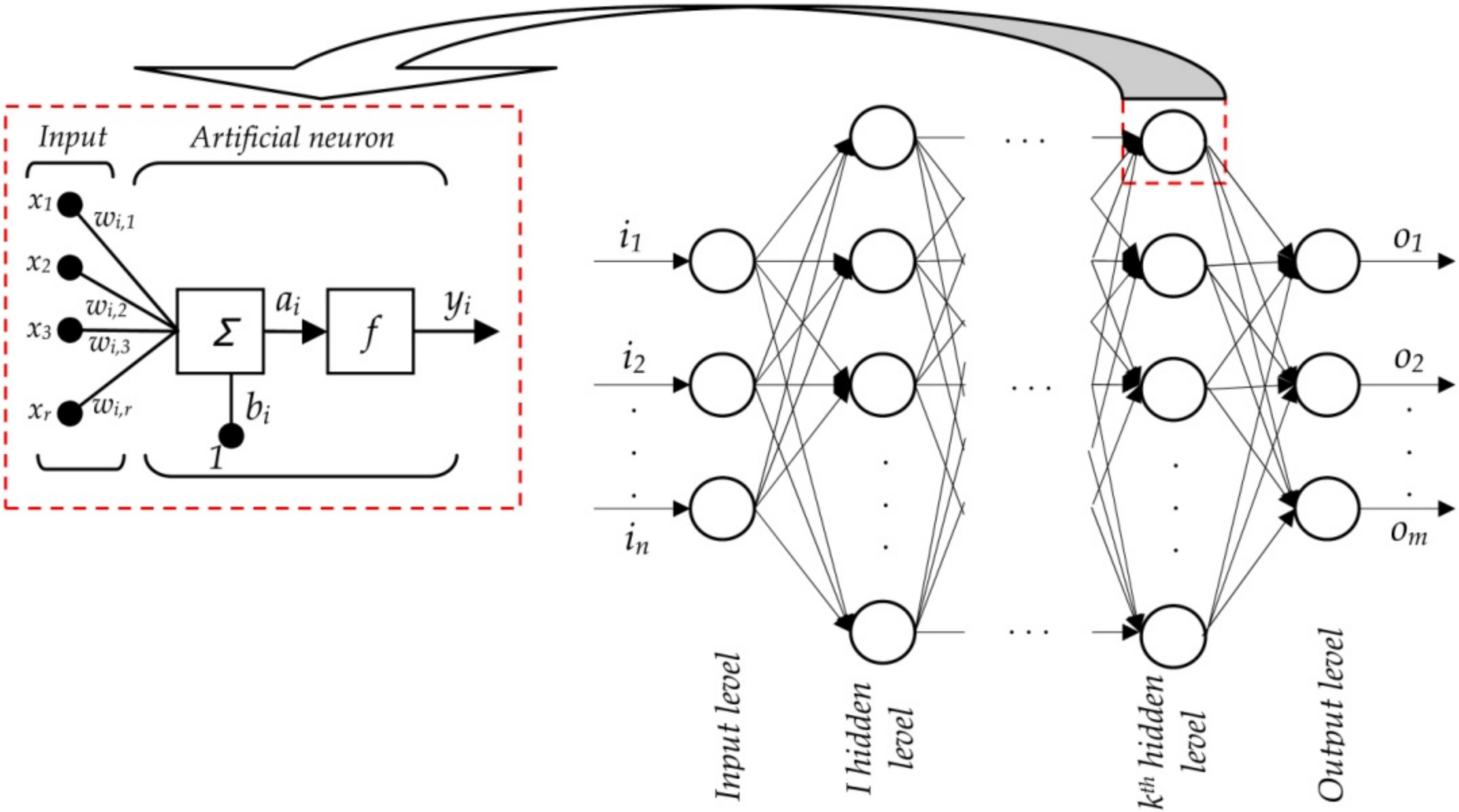


TRAINING MODEL

- Parameters - Weight matrix and Bias Vector
- Objective - minimise Loss Function.

Loss Function : g(f(Actual Value) - f(Predicted Value))

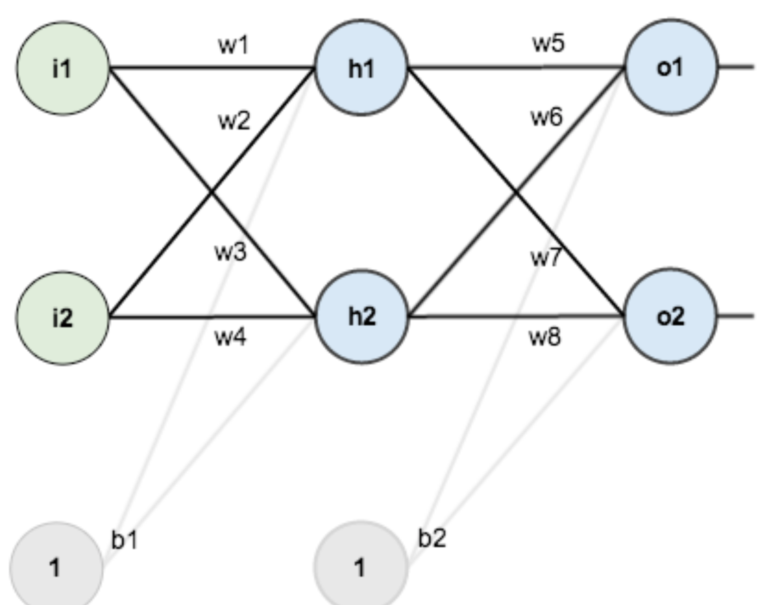
NEURAL NETWORK & LOSS FUNCTION

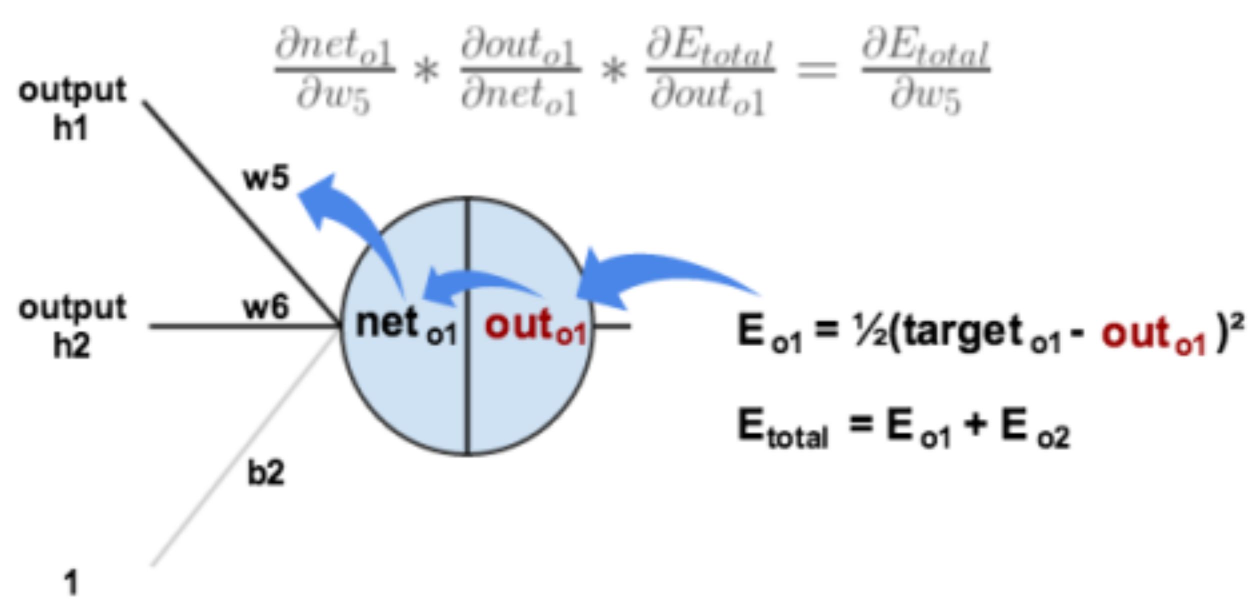


BACKPROPAGATION

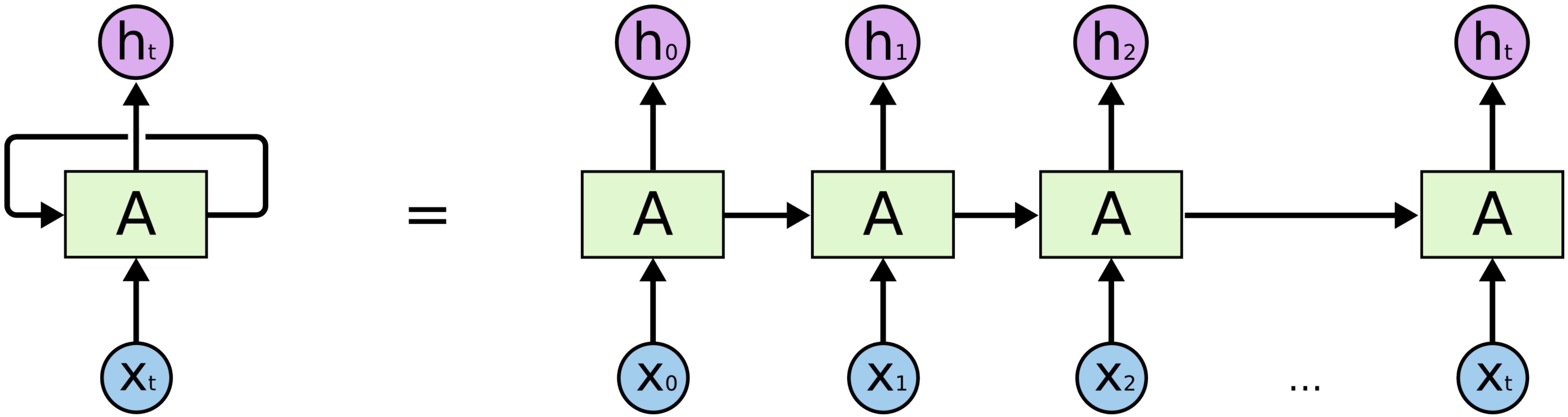
Back propagation is commonly used by the gradient descent optimisation algorithm to adjust the weight of neurones by calculating the gradient of the loss function.

The error is calculated at the output and distributed back through the network layers.





RNN & ADVERSARIAL NETWORKS



DISTBELIEF-LIMITATIONS

- Defining New Layers
- Refining the Training Algorithms
    - Newer Versions of SGD required changes in Parameter server Implementation. e.g.
    - get() and put() interface were not sufficient for all optimisation methods. e.g.
- Defining new Training Algorithms
    - Not sufficient for advanced models such as RNN, adversarial networks, reinforcement learning models.
- Main Purpose for Training Deep Neural Networks.
- Difficult to Deploy trained models to different Platforms such as Mobiles

DISTBELIEF-LIMITATIONS

- Parameter update rules not the same programming model as the rest of the system.
- Separate code for parameter servers vs. rest of system.
- Lacked uniformity & was more complicated.

DESIGN PRINCIPLES

- Dataflow graphs of primitive operators
- Deferred execution
- Common abstraction for heterogeneous accelerators

DATAFLOW GRAPHS

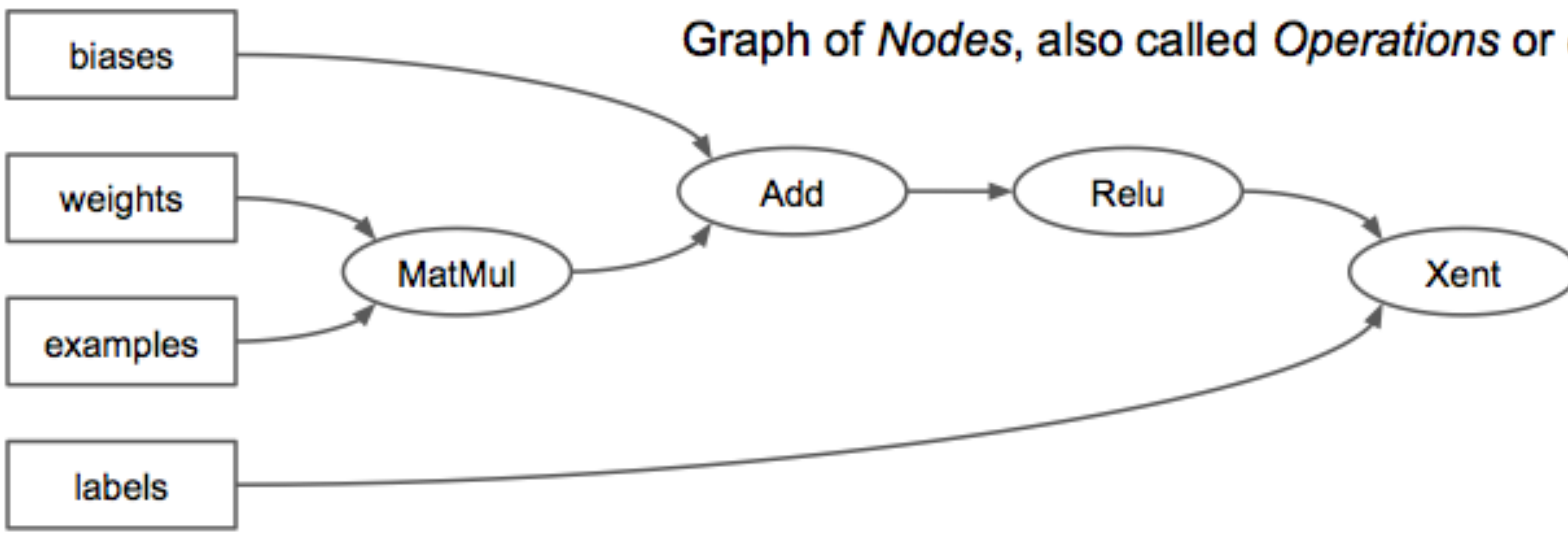
- A directed graph that shows the data dependencies between a number of functions.

- Consume data from input ports and produce data to its output ports.

DATAFLOW GRAPHS

Computation is a dataflow graph

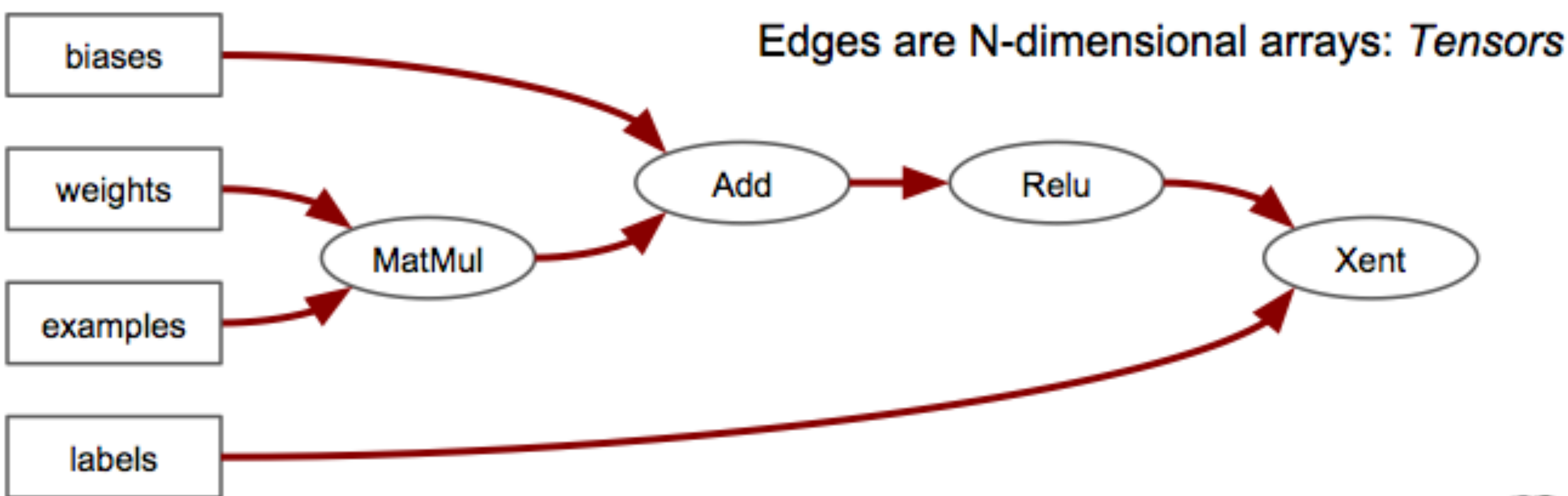
Graph of *Nodes*, also called *Operations* or *ops.*



DATAFLOW GRAPHS

Computation is a dataflow graph

with tensors



DATAFLOW GRAPHS OF PRIMITIVE OPERATORS

- DistBelief:
  - Complex layers , Rigid Structure (c++ classes)
  - Not helpful for research purposes.
- TF:
  - Easier to work on new and add new layers, every mathematical operation is represented as node.
  - Allows to define separate gradient for each layer.
  - Mutable state and updation policy as nodes, which helps in defining new rules.

Deferred execution

TWO PHASES:

First Phase:

- Defines program as dataflow graph where nodes are mapped to input data and variables that define the state.
- Performs intermediate optimisations before execution.

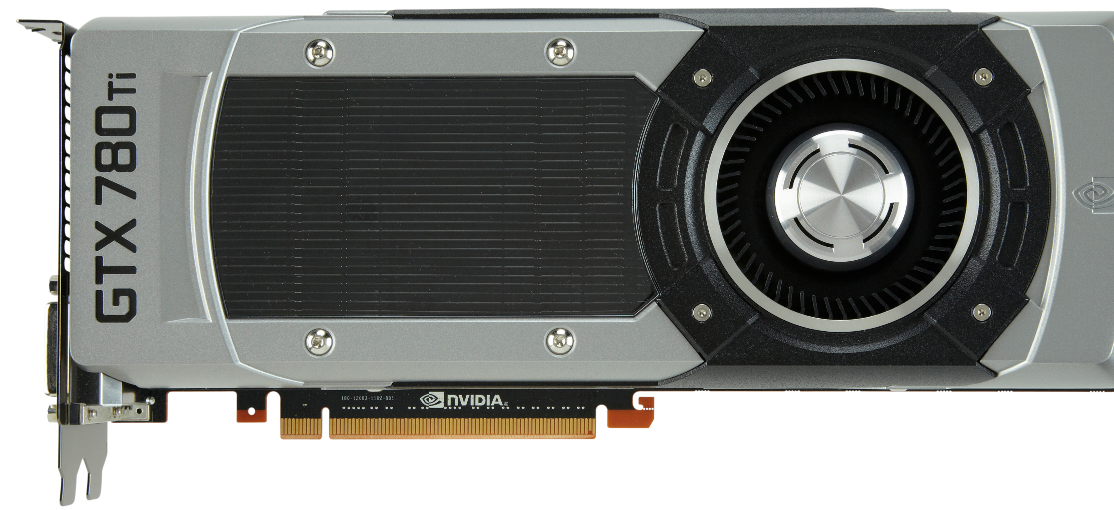
Second Phase:

- Execution of Program on multiple available devices.
- Defer the execution till entire program is available, which helps in better utilisation of resources.

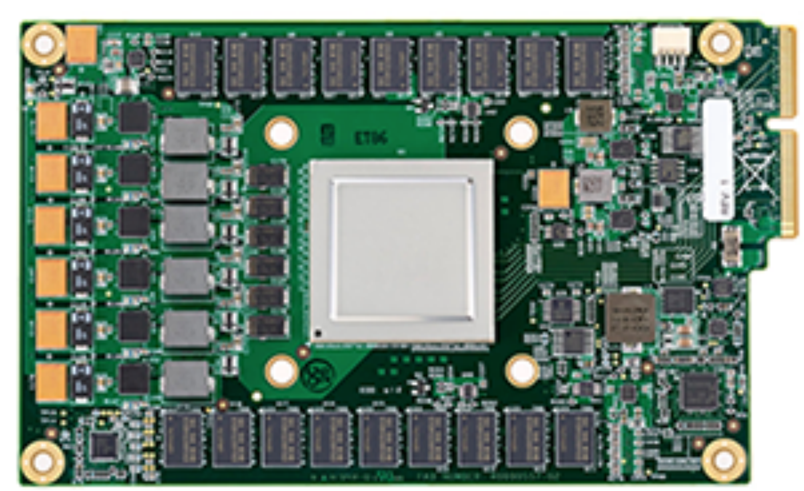
COMMON ABSTRACTION FOR HETEROGENEOUS ACCELERATORS

- CPU - Central Processing Unit

- GPU - Graphical Processing Unit



- TPU - Tensor Processing Unit



COMMON ABSTRACTION FOR HETEROGENEOUS ACCELERATORS

Common abstraction for multiple device

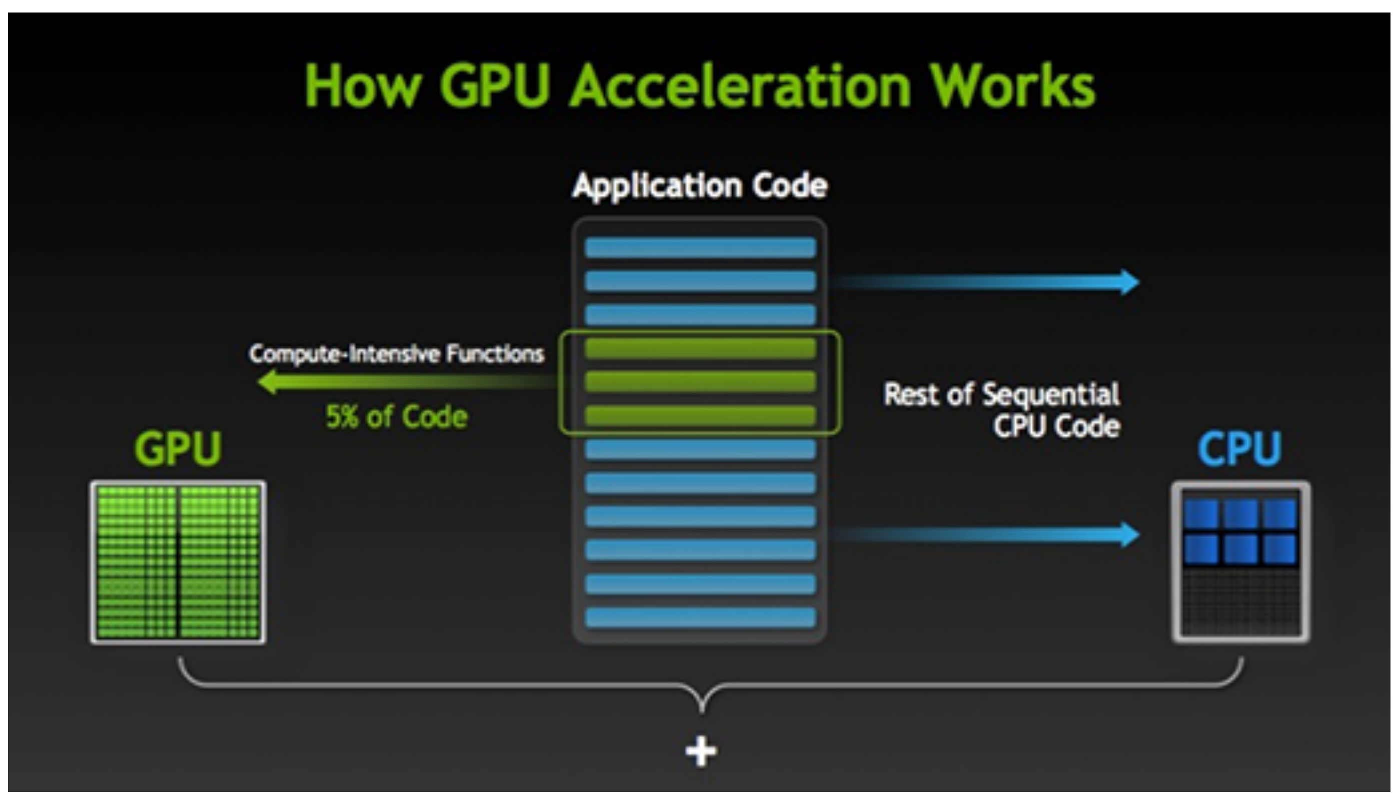
A device must have methods for

- Issuing kernels for execution
- Allocating memory for input and output
- Transferring buffers to and from host memory.

GPU

A graphics processing unit (GPU) is a computer chip that performs rapid mathematical calculations, primarily for the purpose of rendering images.

A CPU consists of a few cores optimised for sequential serial processing while a GPU has a massively parallel architecture consisting of thousands of smaller, more efficient cores designed for handling multiple tasks simultaneously.



TPU

Tensor Processing Unit

Custom machine learning ASIC

In production use for >16 months: used on every search query, used for AlphaGo match, many other uses, ...

See Google Cloud Platform blog: [Google supercharges machine learning tasks with TPU custom chip](), by Norm Jouppi, May, 2016

GENERATIVE ADVERSARIAL NETWORKS

Two agents:

Generative model: Generate new samples that are as similar as the data

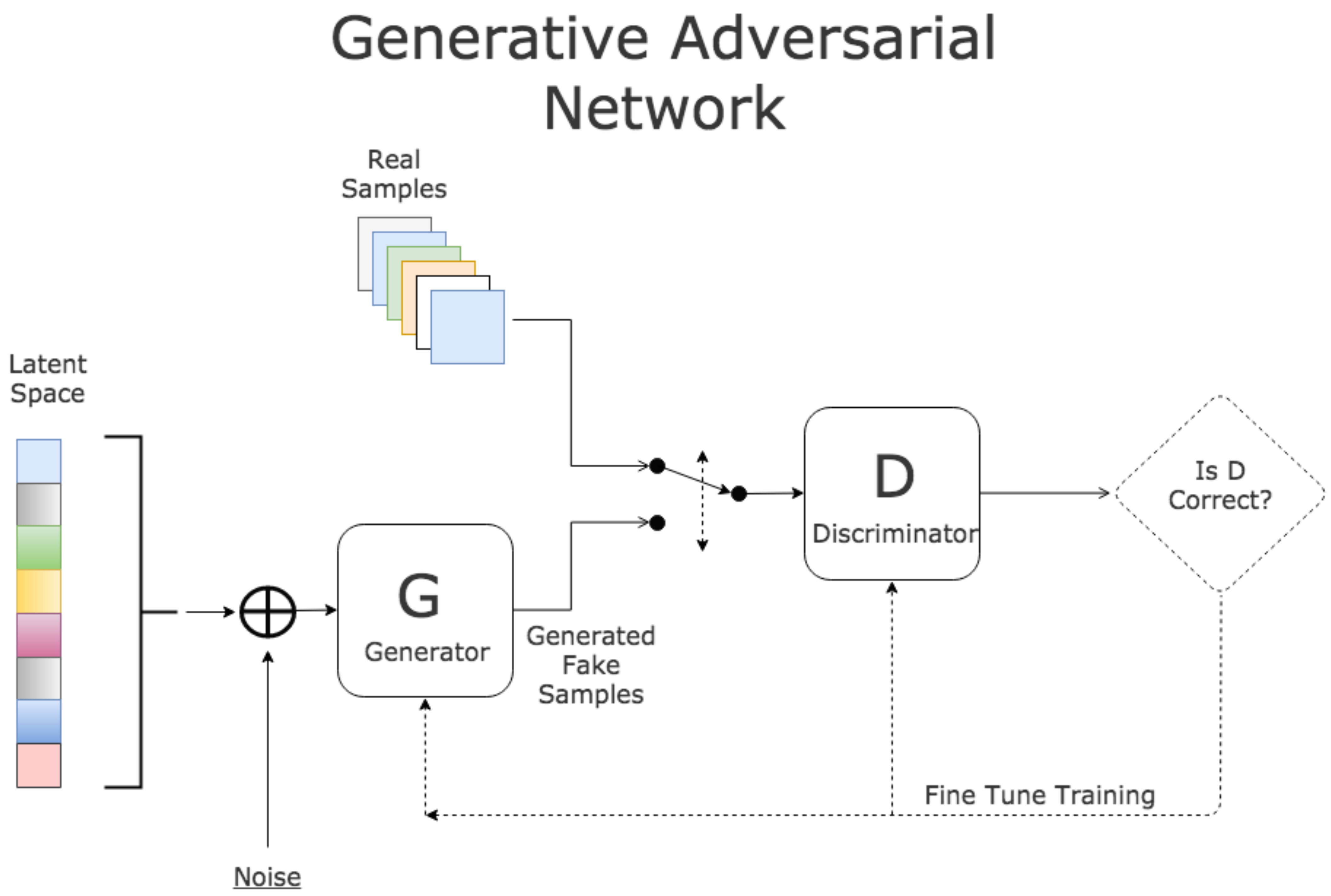
Discriminative model: Distinguish samples in disguise

TRAINING GANS: TWO-PLAYER GAME

Generator network : try to fool the discriminator by generating real-looking images

Discriminator network : try to distinguish between real and fake images

GAN



Generative Adversarial
Network
Real
Samples
Latent
Space
D
Discriminator
Is D
Correct?
G
Generator
Generated
Fake
Samples
Fine Tune Training
Noise

TENSOR

A Tensor is a generalisation of vectors and matrices to potentially higher dimensions. Internally, TensorFlow represents tensors as n-dimensional arrays of base datatypes.

- `Tensor` object:
  - Partially defined computation.
  - Build a graph of `Tensor` objects, calculating dependency and further execution to get desired result.
- All Tensors are dense.
- Efficient memory usage and communication such RDMA, GPU-GPU transfer.

RELATED WORK

- Single machine frameworks
- Batch flow networks
- Parameter Servers

SINGLE MACHINE FRAMEWORKS

- CAFFE
  - Framework for training specified neural network.
  - Easy to compose models.
  - Difficult to add new layers or optimisers.
  - Similar to DistBelief.
- THEANO
  - Similar to TensorFlow.
  - DataFlow Graph Representation.
  - But single machine.

SINGLE MACHINE FRAMEWORKS

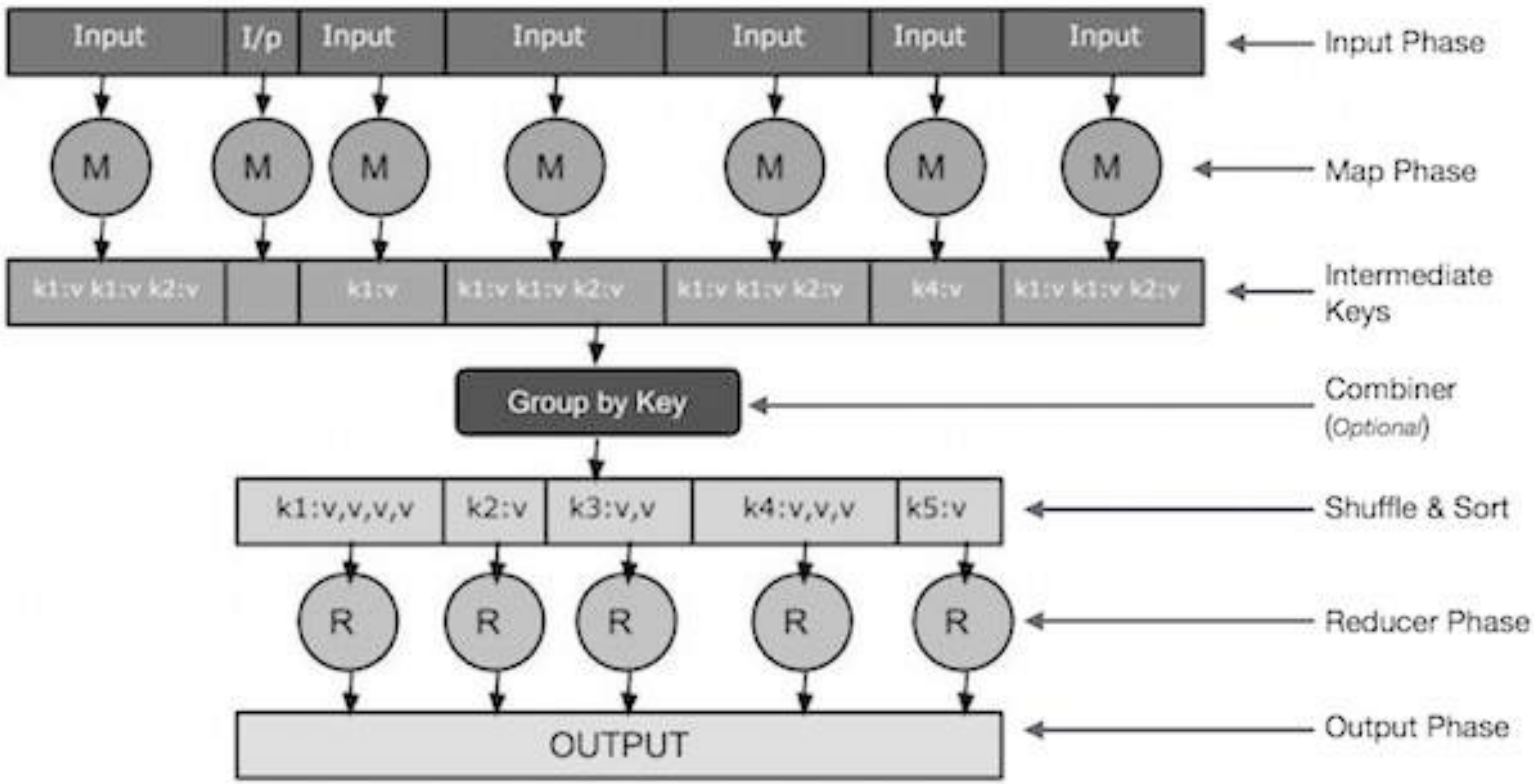
- TORCH
    - Imperative Programming model
    - Fine grained control of execution and memory execution.
    - Lacks dataflow graph for portable representation

MAP REDUCE

The MapReduce algorithm contains two important tasks, namely Map and Reduce.

- The Map task takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key-value pairs).

- The Reduce task takes the output from the Map as an input and combines those data tuples (key-value pairs) into a smaller set of tuples.



Input
I/p
Input
Input
Input
Input
Input
Input Phase
M
Map Phase
k1:v k1:v k2:v
k1:v
k1:v k1:v k2:v
k1:v k1:v k2:v
k4:v
k1:v k1:v k2:v
Intermediate Keys
Group by Key
Combiner (Optional)
k1:v,v,v,v
k2:v
k3:v,v
k4:v,v,v
k5:v
Shuffle & Sort
R
Reducer Phase
OUTPUT
Output Phase

BATCH DATAFLOW SYSTEMS

- Map Reduce Technique
- DryadLINQ
  - Uses high-level query language
  - Powerful than Map-Reduce
- SPARK
  - Extends DryadLINQ
  - Catches computed data for iterative ML algorithms
  - Limited to Immutable Input data and all subcomputations to be deterministic to handle cluster failure.

PARAMETER SERVERS

- Project Adam - CNN
- Li - fault tolerance, models, elastic rescaling.
- GeePS - Parameter server specialised for use with GPUs.
- MXNet - Uses dataflow graph and parameter servers

  Uses some function to aggregate the updates

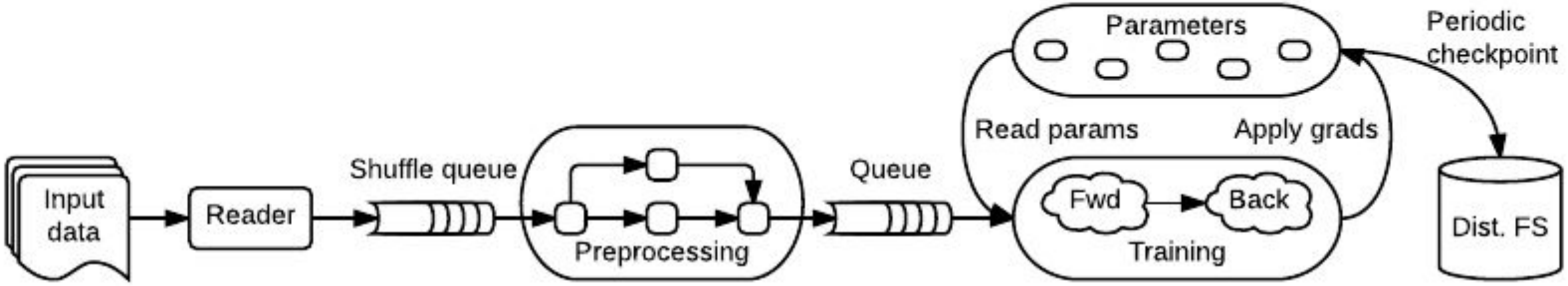
  Doesn’t allow sparse gradient updates within a single value.

TENSORFLOW EXECUTION MODEL

TensorFlow Execution Model

Data Flow graph

- Uses single dataflow graph to represent all computation and states including
  - Individual mathematical operations
  - Parameters
  - Parameter update rules
  - Input preprocessing



Data Flow graph

Expresses communication between subgraph explicitly

Benefit?

Easy to execute independent computations in parallel

Easy to partition computations across multiple devices

Updating parameters in case of very large models

Parameter server approach

- Make in-place updates to very large parameters
- Propagate those updates to parallel training steps as quiclky as possible

Tensorflow approach

- Dataflow with mutable states
- Added flexibility
- Able to experiment with different optimizations algorithms, consistency schemes, parallel strategies

Dataflow Graph Elements

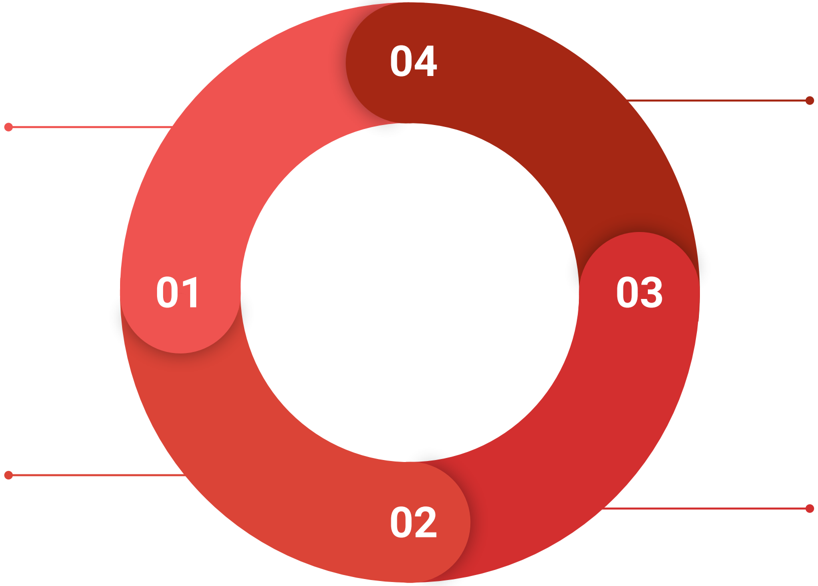
1. Vertex - unit of computation
2. Edge - output from, or input to a vertex

Tensors

- Values that flow along edges
- N-dimensional arrays
- Dense

Operations

- Takes m>=0 tensors as input
- Produces n>=0 tensors as output
- Has named type
- Eg. Const, MatMul, Assign.



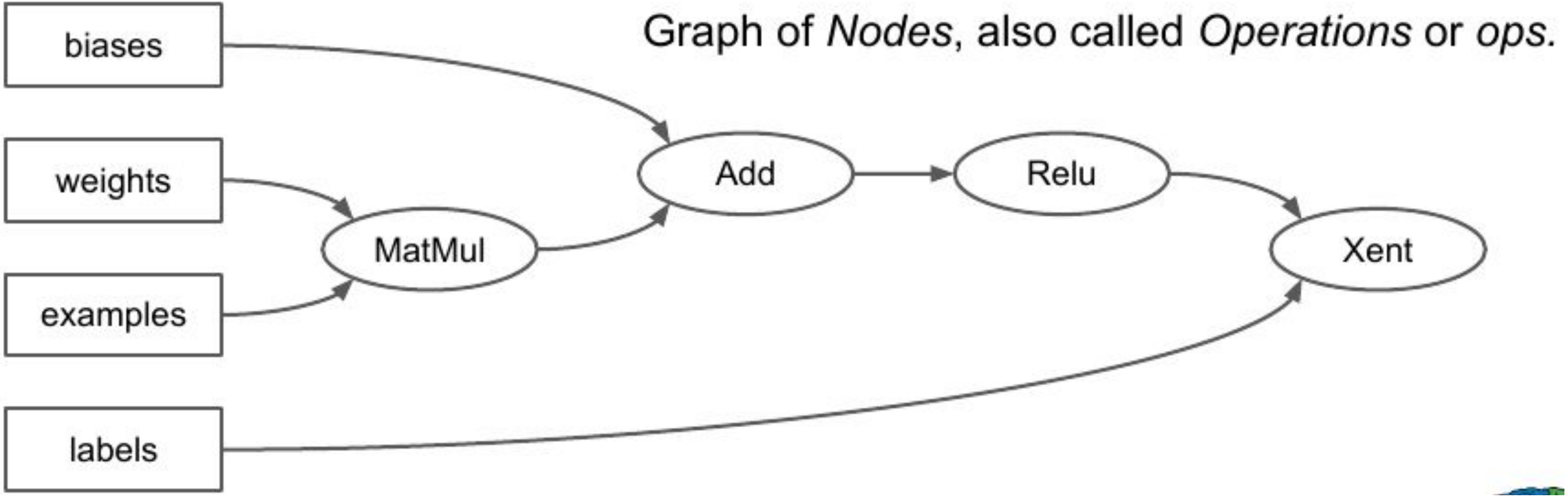
Variables (Stateful operations)

- Variable operation owns a mutable buffer used to store shared parameters
- No inputs
- Produces reference handle

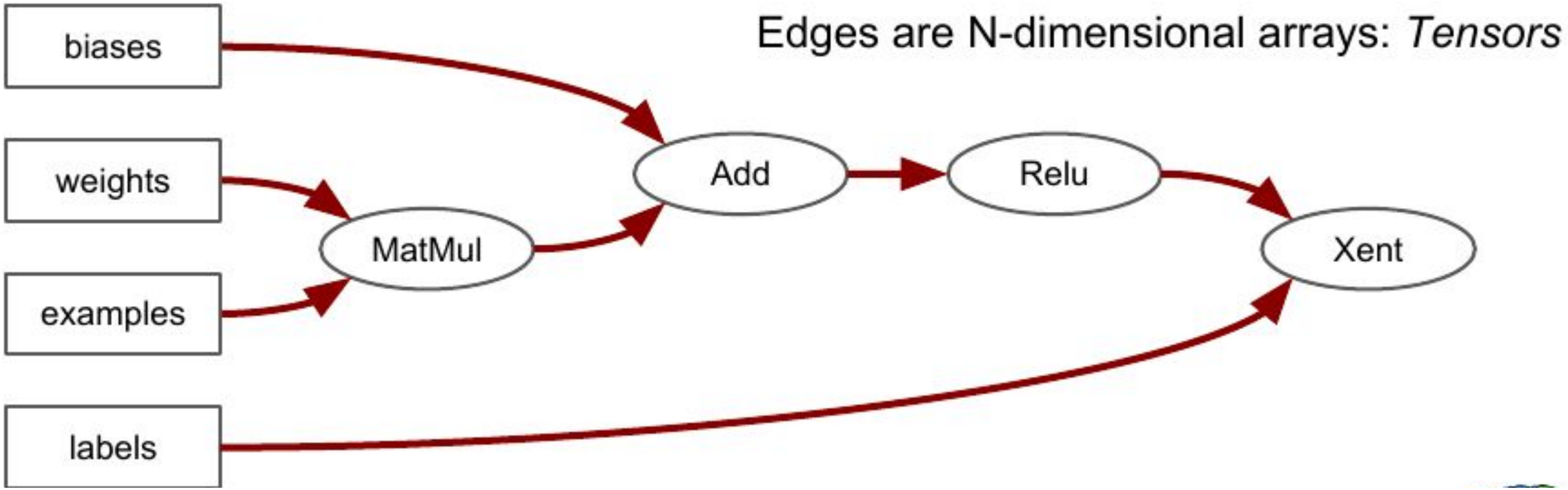
Queues (Stateful operations)

- Several queue implementations supported
- Allows concurrent access
- Produces reference handle
- Supports synchronization

Computation is a dataflow graph



Computation is a dataflow graph



Dataflow Graph Elements

Since tensors are dense, Tensorflow offers two alternatives for sparse data

- Encode data into variable-length string elements of dense tensor
- Use tuple of dense tensors
  - n-D sparse tensor with m nonzero elements
  - Co-ordinate list as m*n matrix of coordinates
  - m length vector of values

Partial and Concurrent Execution

- API for executing graph allows client to specify declaratively the subgraph that should be executed
- Runtime prunes the graph to contain the necessary set of operations
- Tensorflow supports multiple concurrent steps on the same graph
- Co-ordination can be done through queues
- By default concurrent executions of a subgraph run asynchronously
- Asynchrony makes it straightforward to use machine learning algorithms with weak consistency
- Checkpointing subgraph runs periodically for fault tolerance.

Distributed Execution

- Simplified by dataflow as communication between subcomputations explicit
- Allows a program to be deployed on
  - Cluster of GPUs for training
  - Cluster of TPUs for serving
  - Cellphone for mobile inference
- Tensorflow runtime places operations on devices subject to implicit and explicit constraints on graph
- Device responsible for executing a kernel for each operation assigned to it
- Tensorflow allows multiple kernels to be registered for single operation
- For many operations, such as element-wise operators (Add, Sub, etc.), a single kernel implementation is compiled for CPU and GPU using different compilers.

Distributed Execution

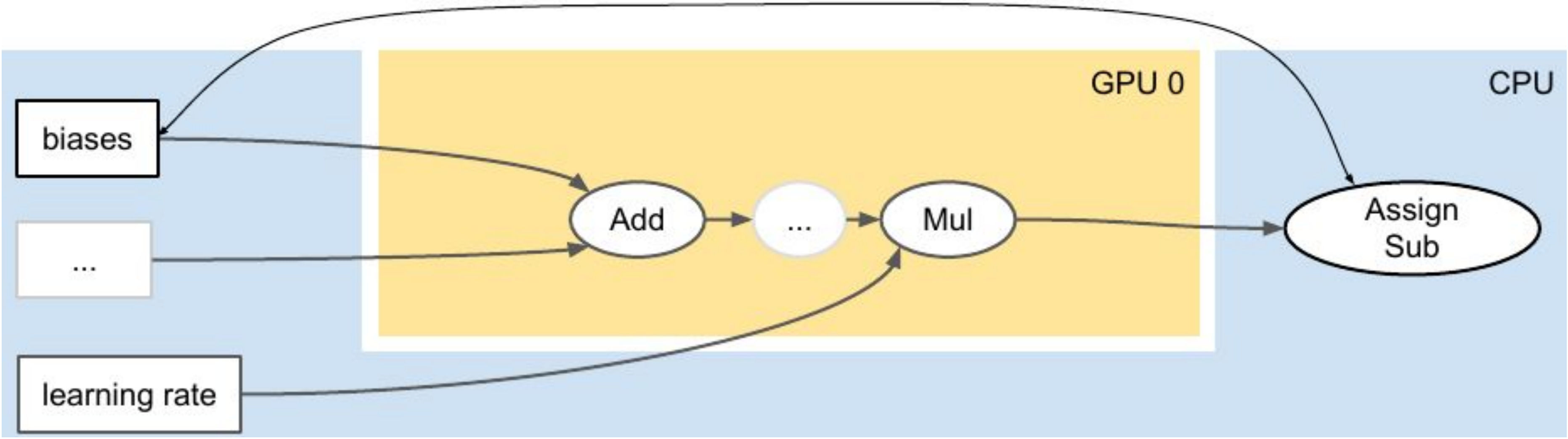
Placement algorithm

- Computes feasible set of devices for each operation
- Calculates set of operations that must be colocated.
- Selects a satisfying device for each colocation group
- Respects implicit colocation constraints (each stateful op and its state on same device)
- Custom device preferences allowed

Distributed Execution

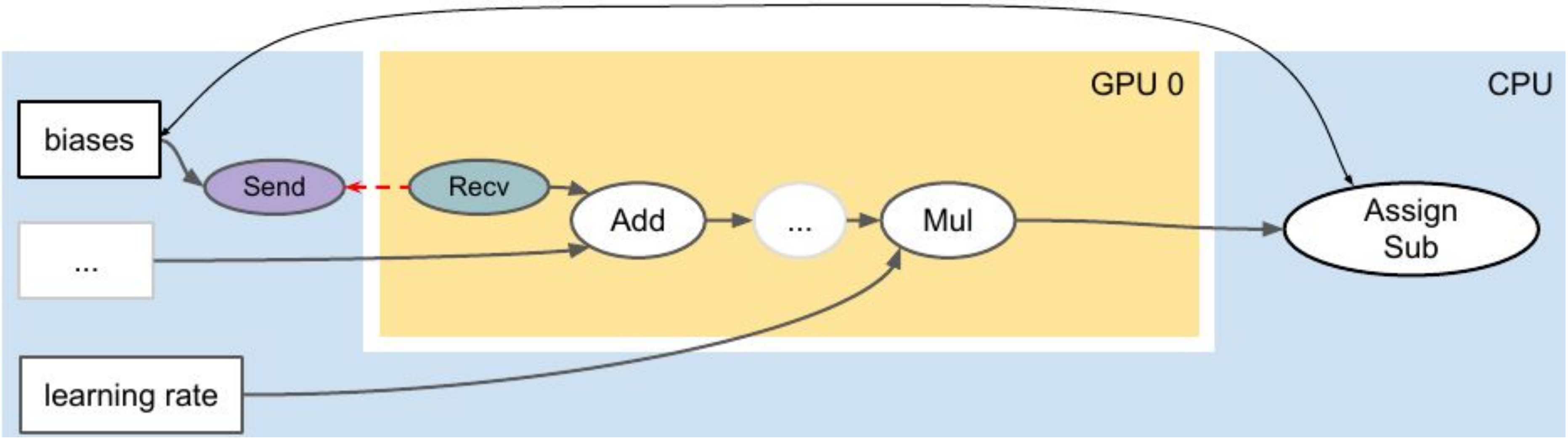
- Simple heuristics yield adequate performance for novice users
- Expert users can optimize performance by manually placing operations to balance the computation, memory, and network requirements across multiple tasks and multiple devices within those tasks

Distributed Execution



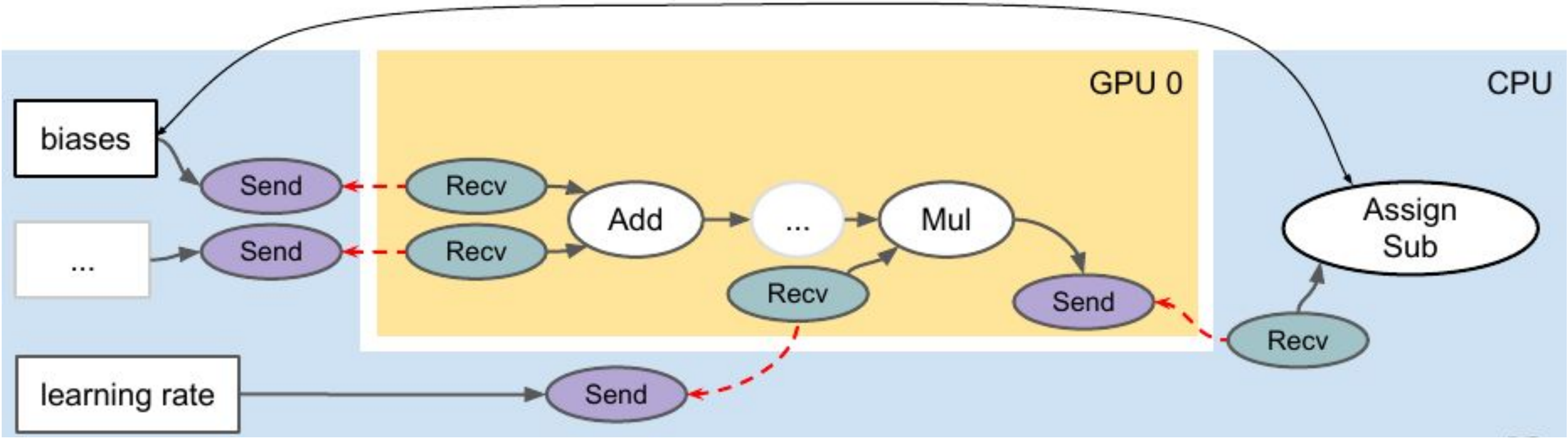
Assign devices to ops

- TensorFlow inserts Send/Recv Ops to transport tensors across devices
- Recv ops pull data from Send ops



Assign devices to ops

- TensorFlow inserts Send/Recv Ops to transport tensors across devices
- Recv ops pull data from Send ops



Send and Receive Implementations

- Send transmits its single input to a specified device as soon as the tensor is available, using a rendezvous key to name the value
- Recv has a single output, and blocks until the value for a specified rendezvous key is available locally, before producing that value

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

Distributed Execution

- Once the graph for a step has been pruned, placed, and partitioned, its subgraphs are cached in their respective devices.
- A client session maintains the mapping from step definitions to cached subgraphs, so that a distributed step on a large graph can be initiated with one small message to each participating task.
- This model favors static, reusable graphs, but it can support dynamic computations using dynamic control flow.

Dynamic Control Flow

- Tensorflow supports advanced machine learning algorithms that contain conditional and iterative control flow (RNN) (generate predictions from sequential data)
- Core of RNN is a recurrence relation
- Dynamic control flow enables iteration over subsequences that have variable lengths
- Doesn’t unroll the computation to the length of the longest sequence

Dynamic Control Flow

```
input = ...   # A sequence of tensors
state = 0     # Initial state
w = ...       # Trainable weights

for i in range(len(input)):
  state, out[i] = f(state, w, input[i])
```

- Added conditional and iterative programming constructs in dataflow graph itself
- Above primitives used to build higher-order constructs, such as map(), fold() and scan()

Dynamic Control Flow

- Uses Switch and Merge from classic dynamic dataflow architectures

Switch

- Demultiplexer
- Takes input data and a control input
- Switch output not taken receives a special dead value.

Merge

- Multiplexer
- Forwards atmost 1 non-dead signal to output
- Produces dead signal if both input dead signal

Dynamic Control Flow

- While loop is much more complicated
- operators to ensure loop is well formed
  - Enter
  - Exit
  - NextInstruction
- The execution of iterations can overlap
- TensorFlow can also partition conditional branches and loop bodies across multiple devices and processes.
- The partitioning step adds logic to coordinate the start and termination of each iteration on each device, and to decide the termination of the loop.

Extensible

- Core system defines a number of standard operations

and kernels (device-specific implementations of

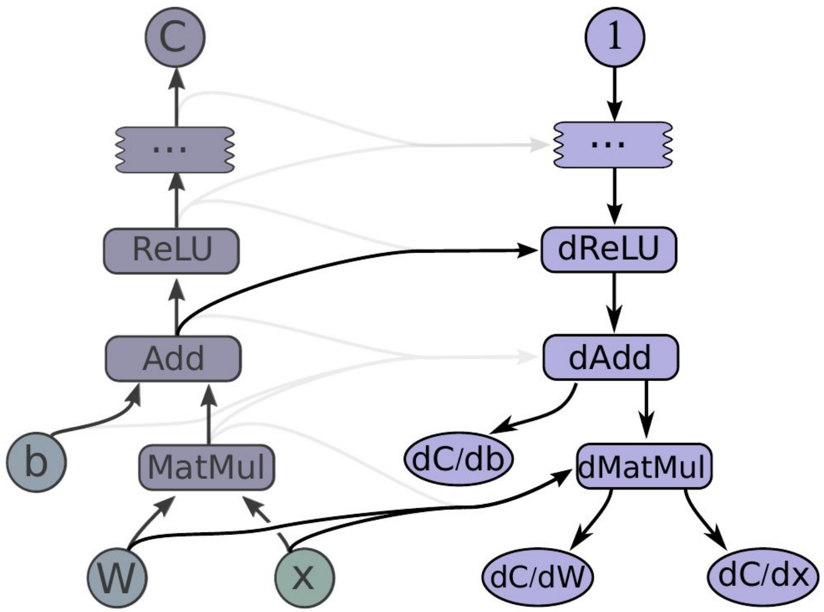
operations)

- Easy to define new operators and/or kernels

Differentiation and Optimization

Differentiation :

- Library differentiates a symbolic expression for a loss function and produces a new symbolic expression representing the gradients.
- Given a neural network as a composition of layers and a loss function, the library will automatically derive the backpropagation code.

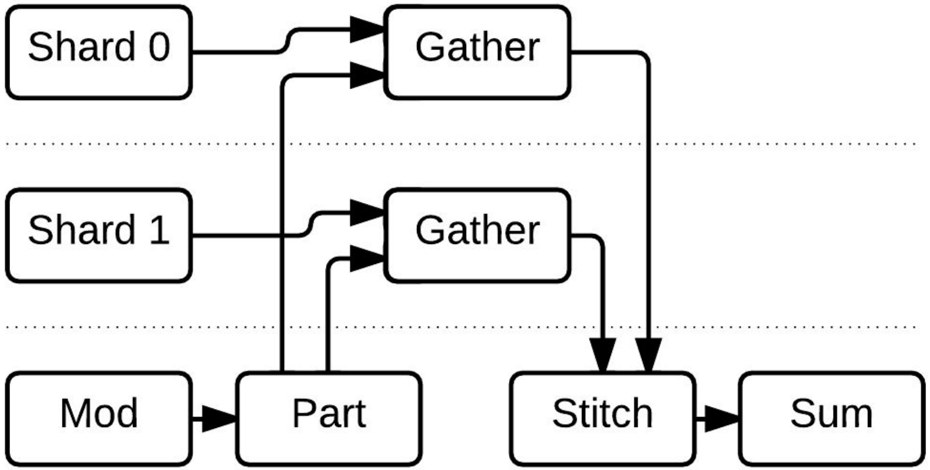


Differentiation and Optimization

- Wide range of optimization algorithms provided
- W’ = W - a * dL/dW
    - Parameter server can implement SGD using -= as write operation
- New optimizations can be built using Variable operations and primitive mathematical operations without modifying the underlying system

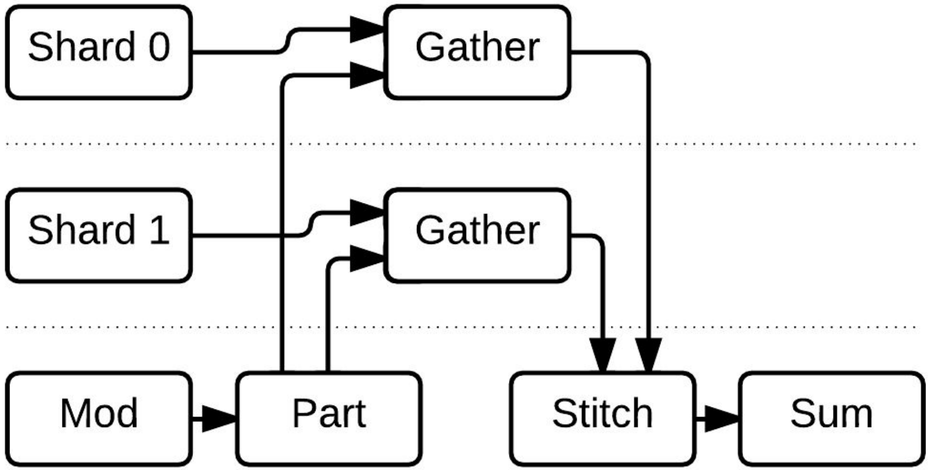
Training very large models

- b -> sparse vectors
- n -> number of words in dictionary
- n * d embedding matrix
- b * d dense matrix representation
- n * d can be very large
- Implement sparse embedding matrix as composition of primitive operations



Training very large models

- Gather : extracts a sparse set of rows form a tensor
- Part : divides incoming indices into variable-sized tensors that contain indices destined for each shard
- Stitch - reassembles partial results into single tensor
- Each operation has its corresponding gradient, so it supports automatic differentiation



Exploiting Model Parallelism

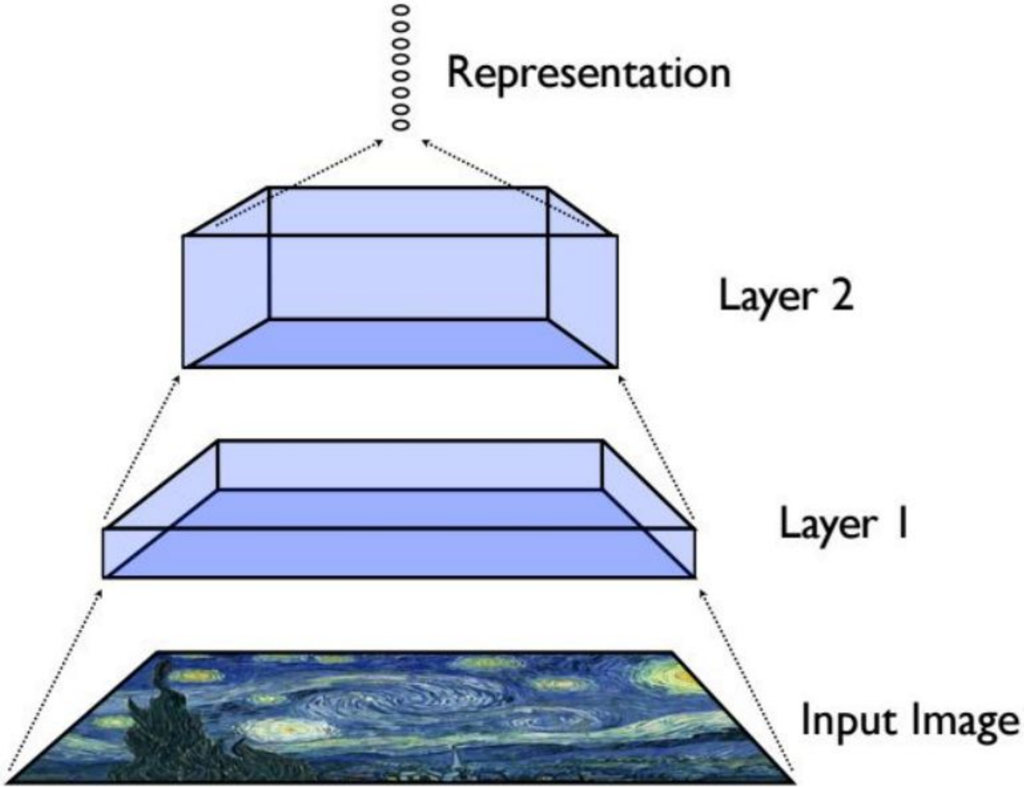
On a single core: Instruction parallelism (SIMD). Pretty much free.

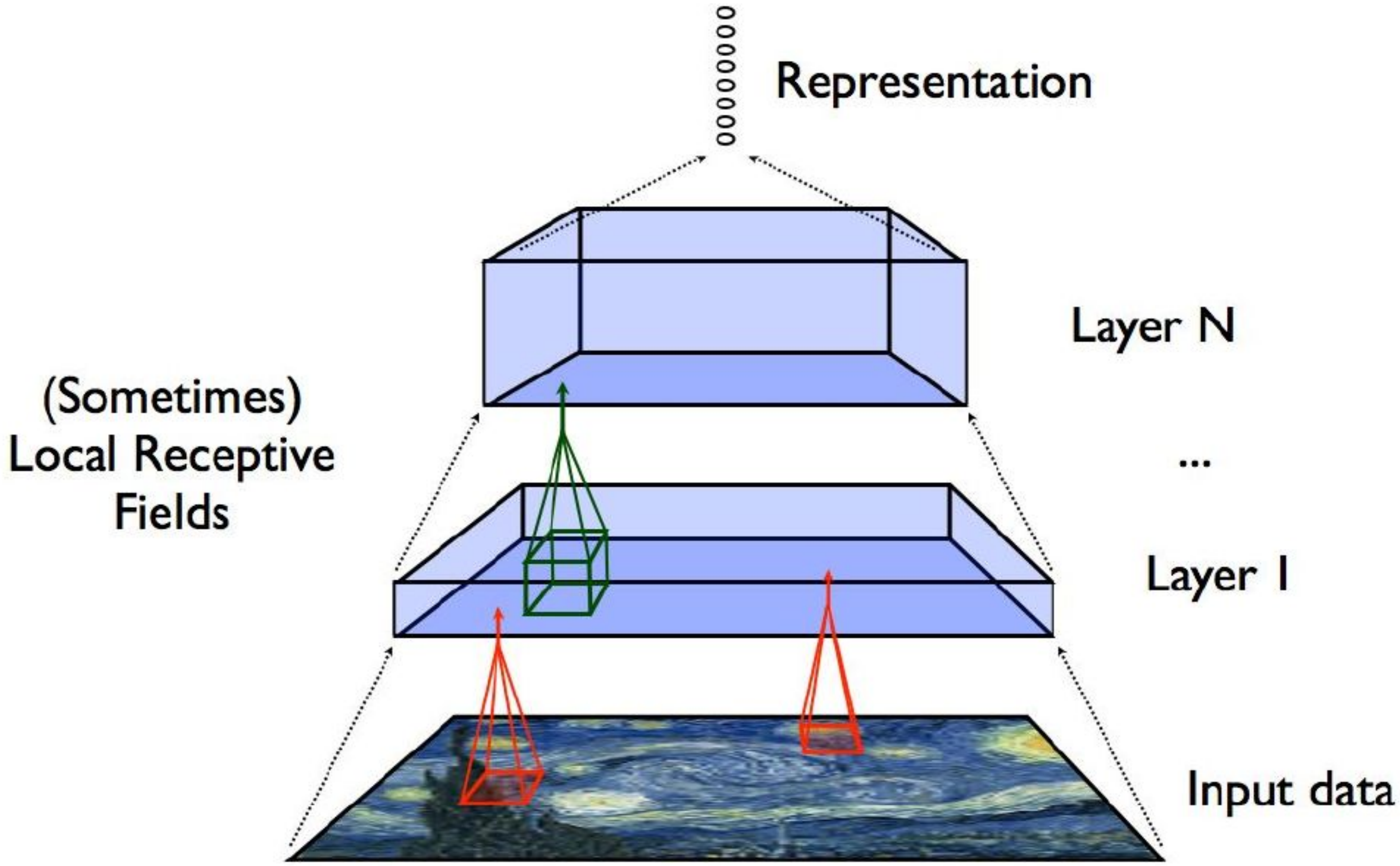
Across cores: thread parallelism. Almost free, unless across sockets, in which case inter-socket bandwidth matters (QPI on Intel).

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

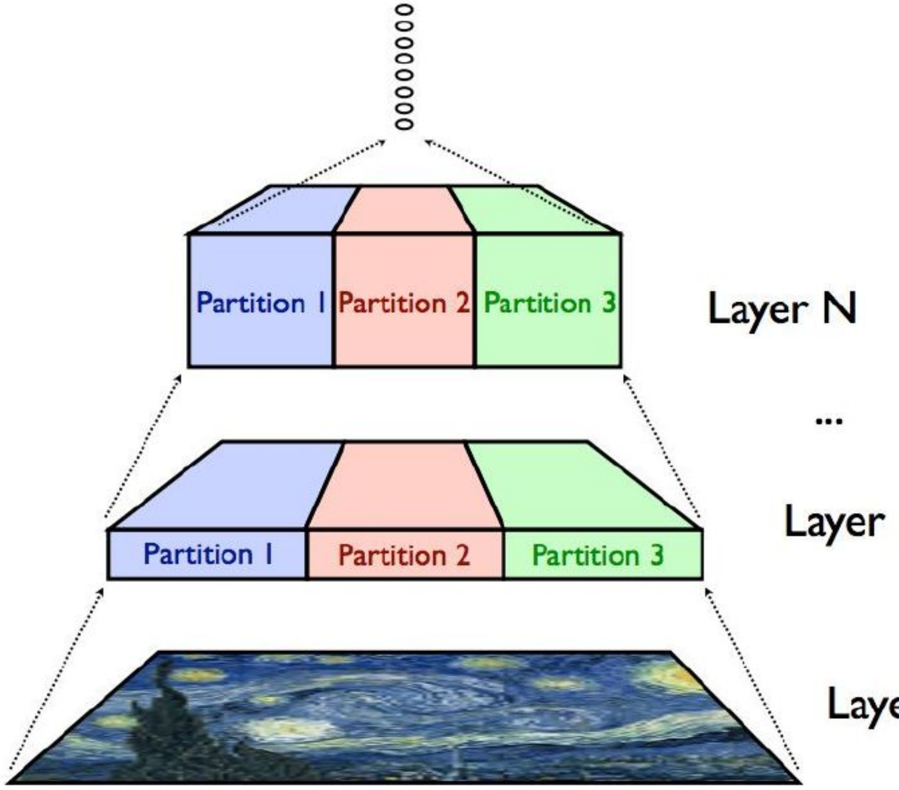
Across machines: limited by network bandwidth / latency

Model Parallelism

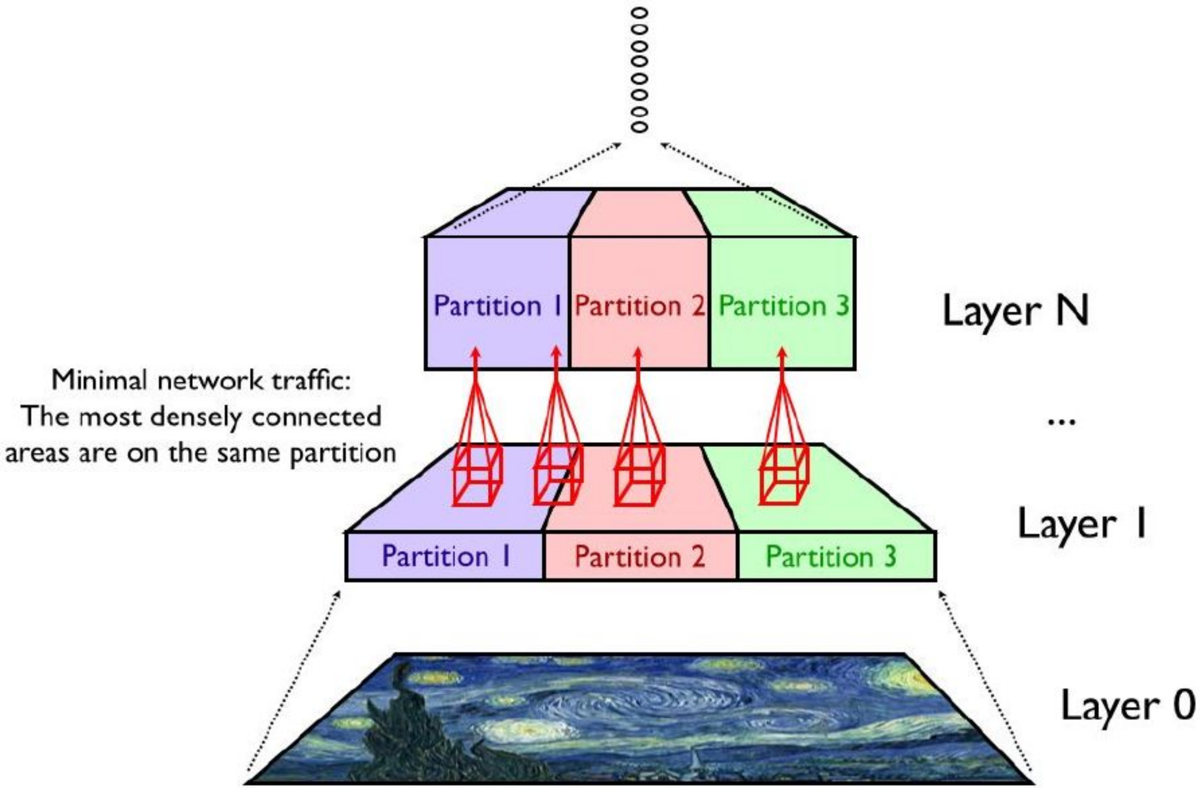



Representation
Layer N
(Sometimes)
Local Receptive
Fields
...
Layer 1
Input data

Model Parallelism : partition model across machines



Model Parallelism : partition model across machines



Fault Tolerance

- Training can take several days
- no guarantee for availability of the same resources for the duration of the training process.
- long-running TensorFlow job is likely to experience failure or pre-emption, and we require some form of fault tolerance.
- Implemented user level checkpointing
- Uses two operations in graph, Save and Restore
- 1 Save per task to maximise I/O bandwidth

Synchronous Replica Coordination

Asynchronous :

- SGD is robust to asynchrony
- Scalable because they maintain high throughput in presence of stragglers
- The increased throughput comes at the cost of using stale parameter values in training steps

Synchronous :

- Slow workers limit throughput
- Backup workers used to mitigate stragglers
- Take m out of n inputs

Data Parallelism Choices

Can do this Synchronously :

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

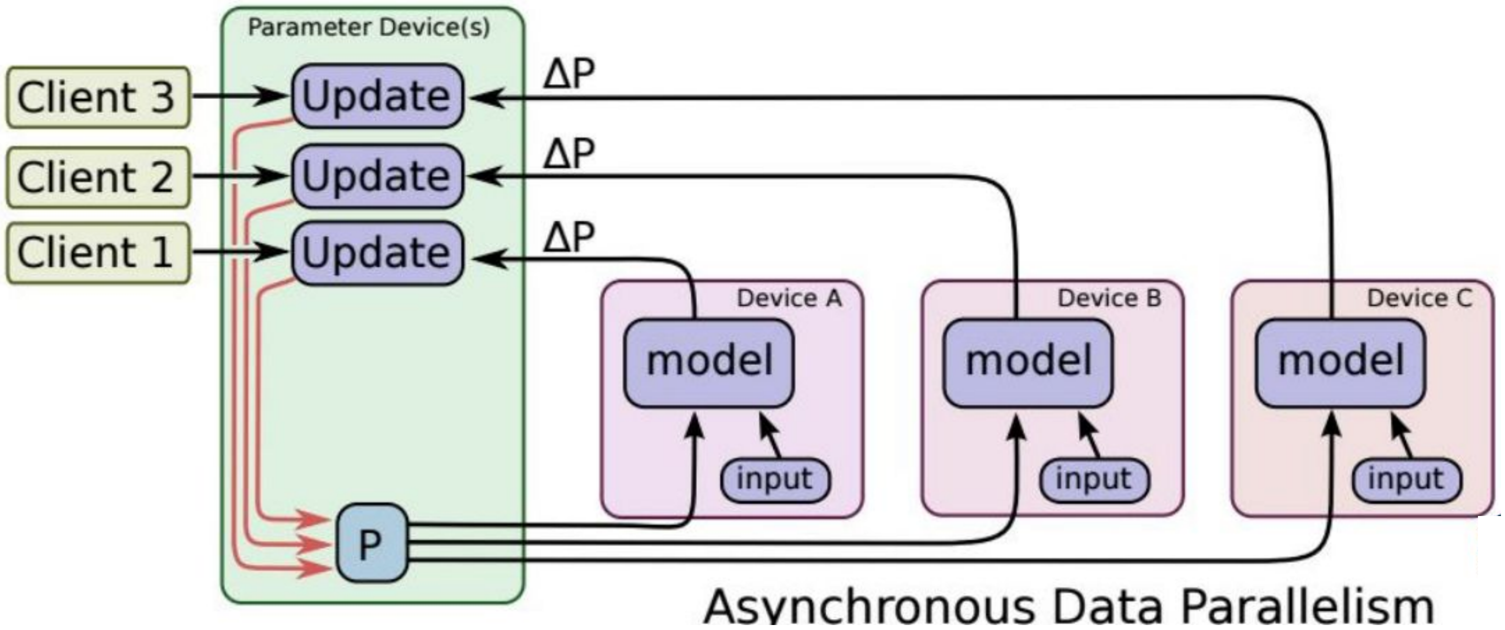
Can do this Asynchronously :

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

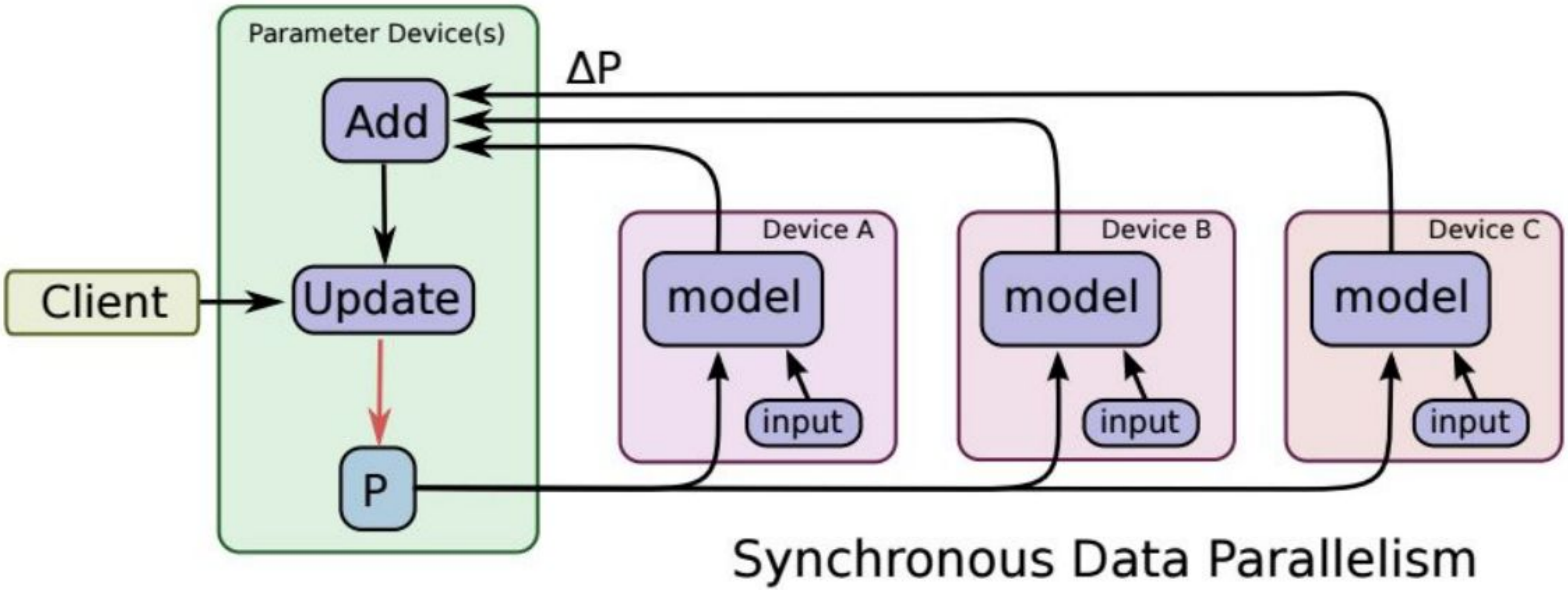
(or Hybrid : M asynchronous groups of N synchronous replicas)

Asynchronous Training

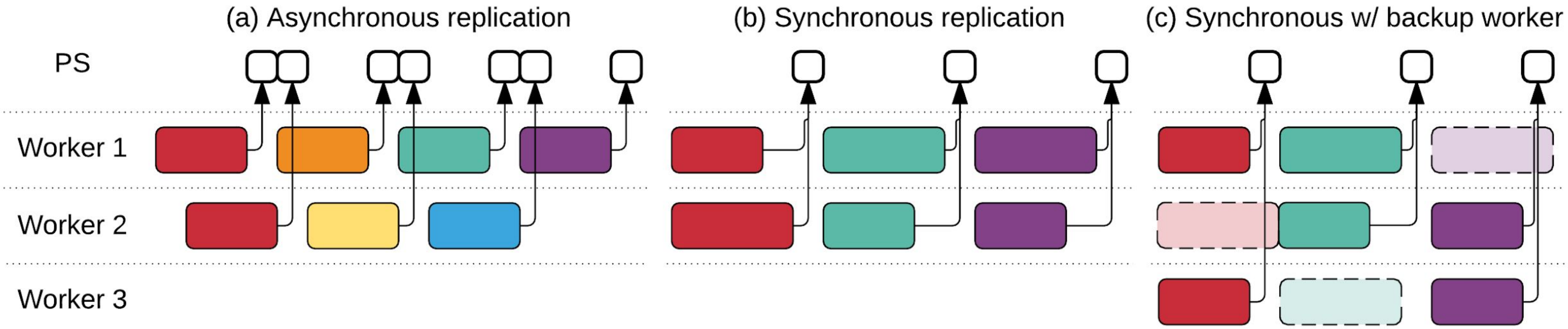
- Unlike DistBelief, no separate parameter server system:
  - Parameters are now just stateful nodes in the graph



Synchronous Training

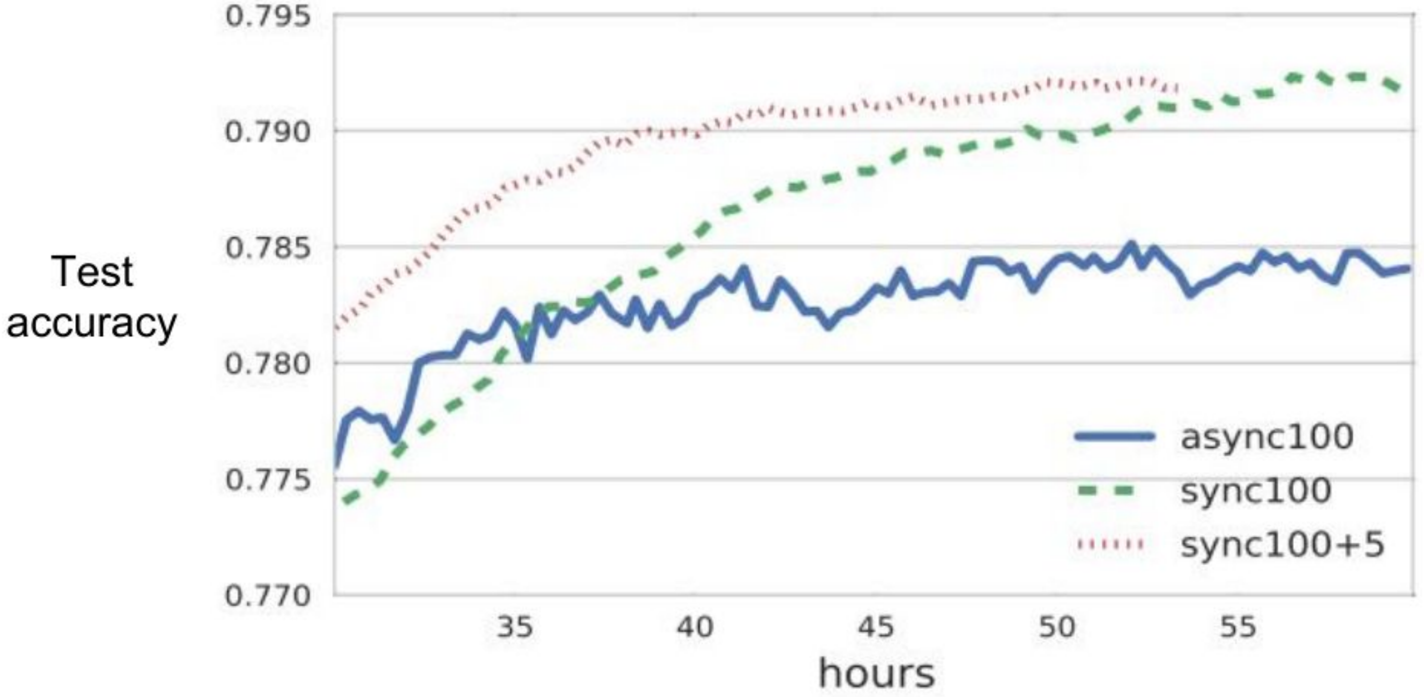


Synchronous Vs Asynchronous



- Three synchronization schemes for parallel SGD.
- Each color represents a different starting parameter value
- White square is a parameter update
- Dashed rectangle represents a backup worker whose result is discarded.

Synchronous converges faster (time to accuracy)



- Synchronous updates (with backup workers) trains to higher accuracy faster
- Better scaling to more workers (less loss of accuracy)

Implementation

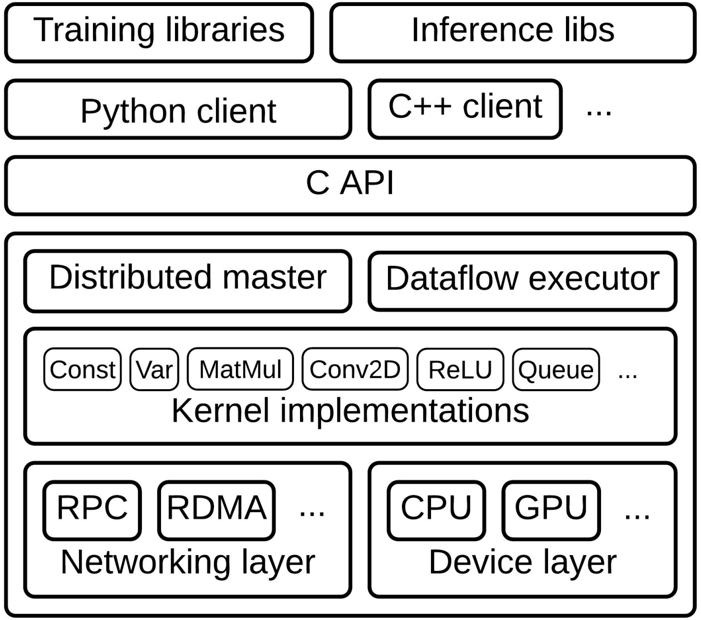
- Tensorflow runtime is a cross platform library
- C API separates user-level code in different languages from the core runtime.
- Runs on several operating systems including

x86 and ARM based CPU architectures

- Linux
- Mac OS X
- Windows
- Android, iOS

GPU microarchitectures

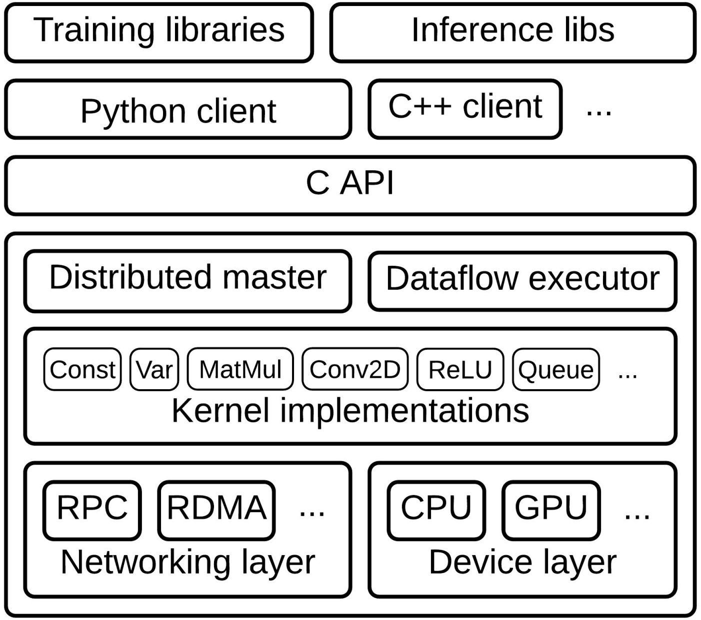
- Kepler
- Maxwell
- Pascal



Implementation

Distributed Master

- Translates user requests into execution across tasks
- Graph and step definitions -> Pruning -> Partitioning -> Subgraphs for each participating device
- Applies standard optimizations such as subexpression elimination
- Coordinates execution of optimized subgraphs across tasks

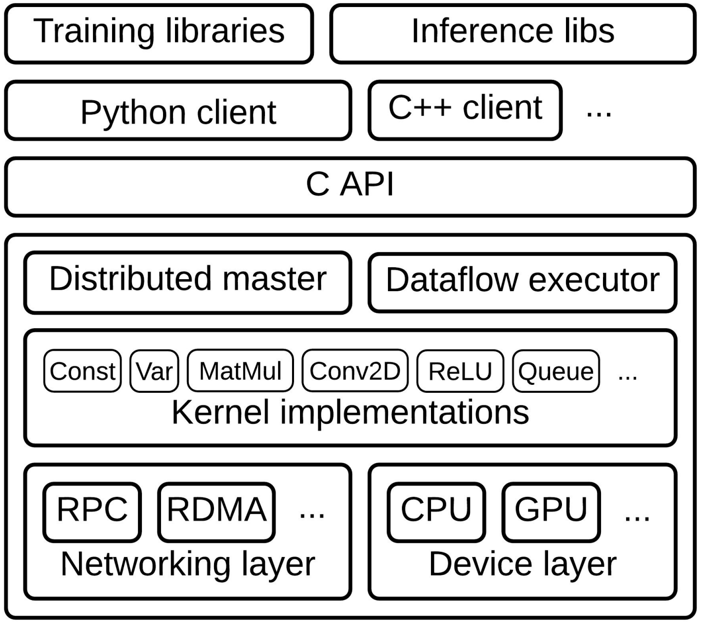


Implementation

Dataflow Executor

- Handles requests from master
- Schedules execution of kernels that comprise a local subgraph
- Dispatches kernels to local devices and runs in parallel when possible

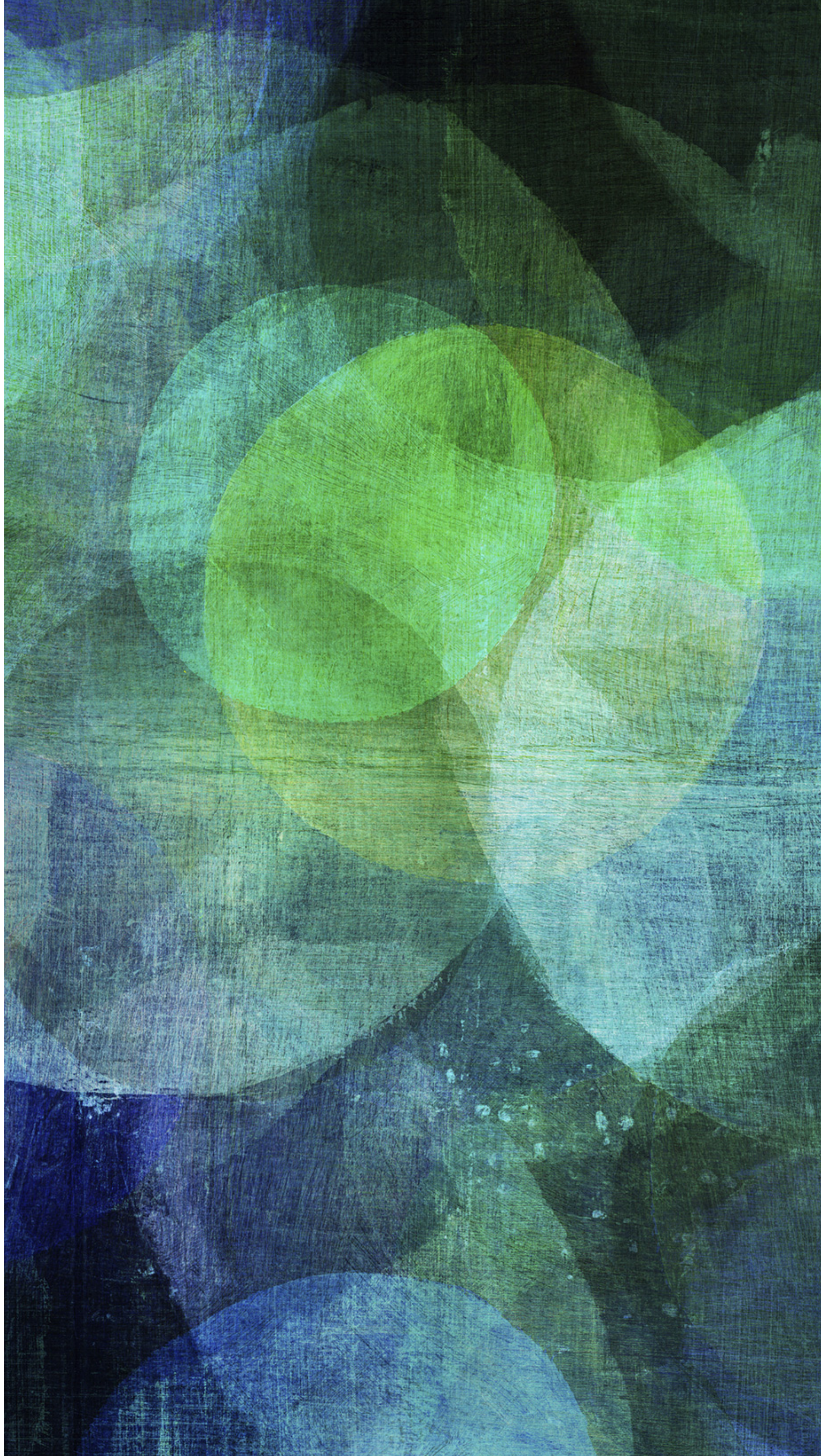
Current implementation can execute upto 10,000 subgraphs per second



Implementation

- Runtime supports over 200 standard operations including
  - Mathematical operations
  - Array manipulation operations
  - Control flow operations
  - State Management operations
- Implemented quantization for faster inference in environments such as mobile devices
- Users can register additional kernels written in C++
- Fused-kernels profitable for performance critical implementations such as Sigmoid and ReLU activation functions and their corresponding gradients

EVALUATION

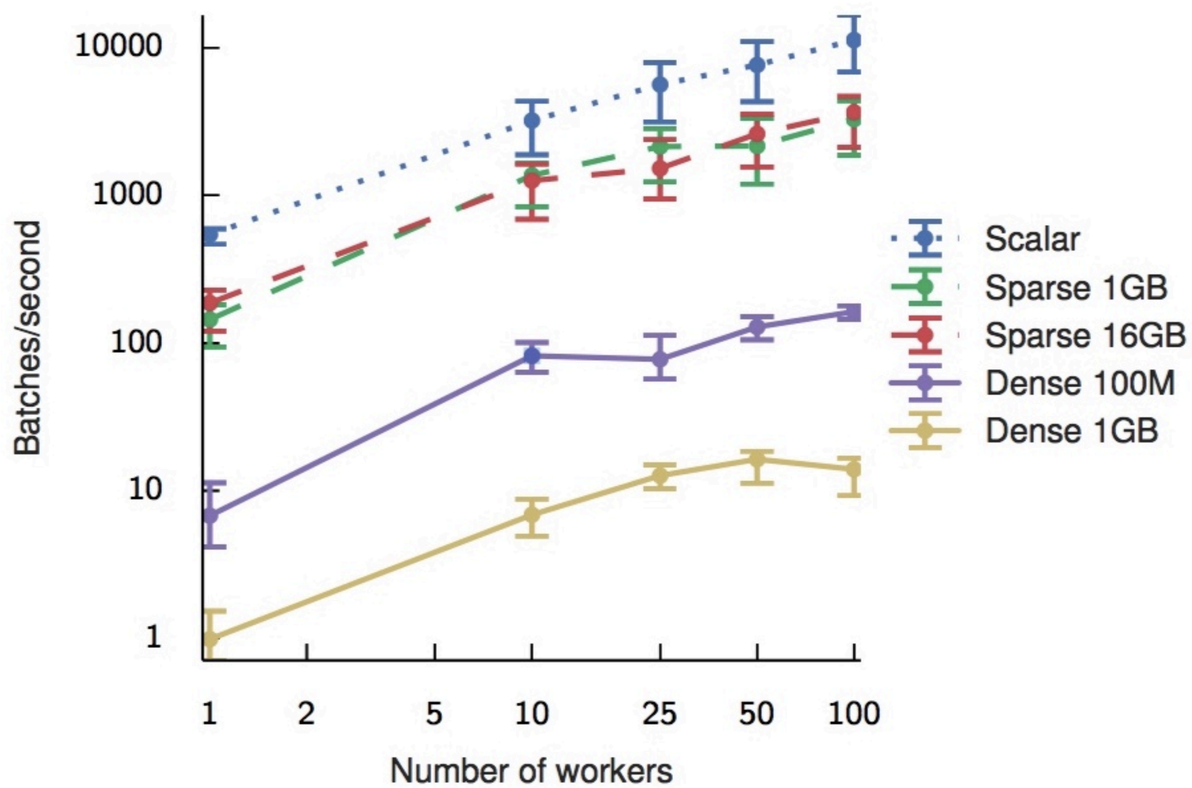


SINGLE MACHINE BENCHMARKS

- Single GPU
- Torch ~ Tensorflow
- Caffe uses open-source and simple libraries.
- Neon uses handwritten convolution kernels.

Library	Training step time (ms)			
	AlexNet	Overfeat	OxfordNet	GoogleNet
Caffe [38]	324	823	1068	1935
Neon [58]	87	**211**	**320**	**270**
Torch [17]	**81**	268	529	470
TensorFlow	**81**	279	540	445

SYNCHRONOUS REPLICA MICROBENCHMARK



Null training step - do trivial operation and send updates.

Scalar curve - single 4B key from all servers

Dense curve - fetches full model

IMAGE CLASSIFICATION - INCEPTION

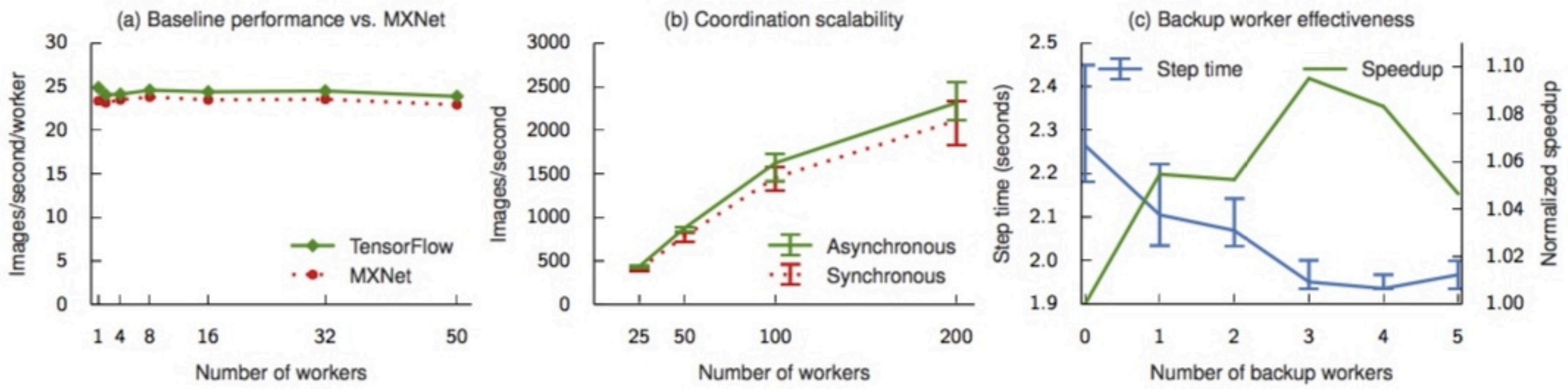


Figure 8: Results of the performance evaluation for Inception-v3 training (§6.3). (a) TensorFlow achieves slightly better throughput than MXNet for asynchronous training. (b) Asynchronous and synchronous training throughput increases with up to 200 workers. (c) Adding backup workers to a 50-worker training job can reduce the overall step time, and improve performance even when normalized for resource consumption.

- *MXNet and TensorFlow - Single GPU performance.*
- *Step time increases as worker increases*
- *Step time - 50 worker creates high contention, adding backup server decreases the step time.*

LANGUAGE MODELLING