AI-Systems
Distributed Training

Joseph E. Gonzalez
Co-director of the RISE Lab
jegonzal@cs.berkeley.edu

What is? & Why? Distributed Training

- **Distributed Training* ~** Training across multiple devices
  - Different local and remote memory speeds / network
- Why do we need distributed training?
  - Faster training by leveraging **parallel computation**
  - **Additional memory** (memory bandwidth) for larger model
    - "Need" to store weights + activations
  - Reduce or eliminate **data movement**
    - Privacy → Federated Learning
    - Limited bandwidth to edge devices
  - Need to process all the data?

*Very simplified definition.

On Dataset Size and Learning

- Data is a a resource! (e.g., like processors and memory)
  - Is having lots of processors a problem?
- You don’t have to use all the data!
  - Though using more data can often help
- More data *often** dominates models and algorithms

EXPERT OPINION

Contact Editor: **Brian Brannon,** bbrannon@computer.org

The Unreasonable Effectiveness of Data

Alon Halevy, Peter Norvig, and Fernando Pereira, *Google*

*More data also supports more sophisticated models and algorithms.

What are the Metrics of Success?

- **Marketing Team:** Maximize number of GPUs/CPUs used
  - A bad metric … why?
- **Machine Learning:** Minimize passes through the training data
  - Easy to measure, but not complete … why?
- **Systems:** minimize time to complete a pass through the training data
  - Easy to measure, but not complete … why?

Ideal Metric of Success

$$\left(\frac{\text{“Learning”}}{\text{Second}}\right) = \left(\frac{\text{“Learning”}}{\text{Record}}\right) \times \left(\frac{\text{Record}}{\text{Second}}\right)$$

Convergence
Machine Learning
Property

Throughput
System
Property

Metrics of Success

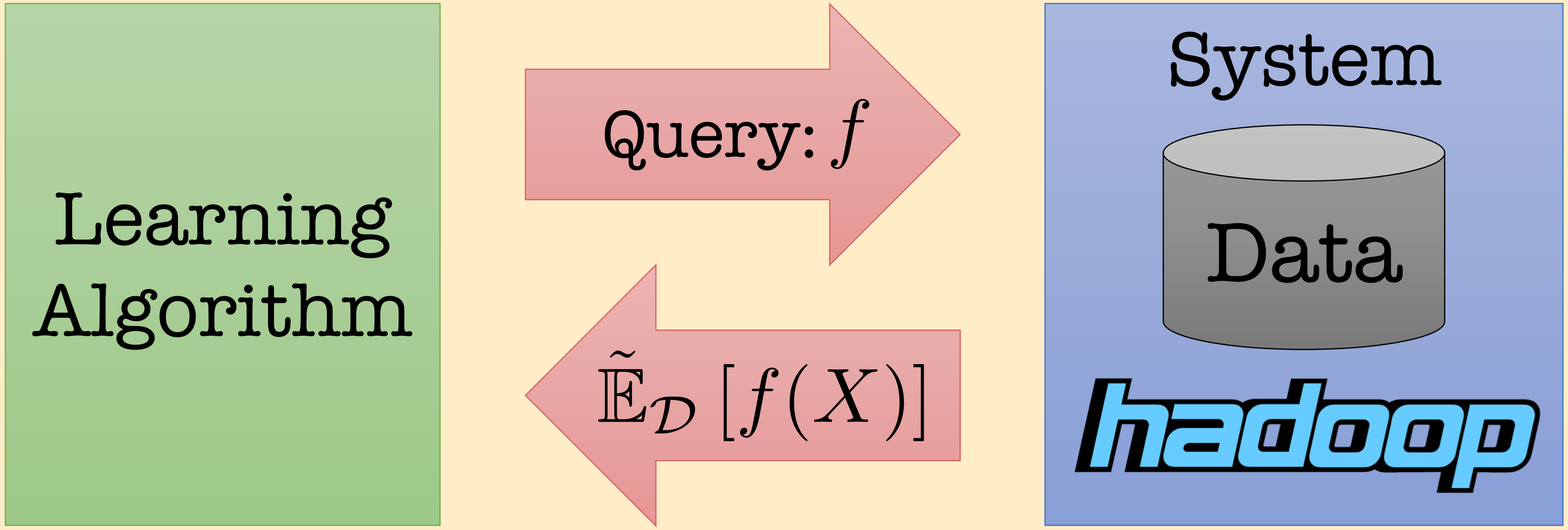
- Minimize training time to "*best model*"
  - Best model measured in terms of test error
- Other Concerns?
  - **Complexity**: *Does the approach introduce additional training complexity (e.g., hyper-parameters)*
  - **Stability**: *How consistently does the system train the model?*
  - **Cost:** *Will obtaining a faster solution cost more money (power)?*

The Early Days….

Map-Reduce for Distributed Training

Learning by Distributed Aggregation

Learning from Statistics (Aggregation)*



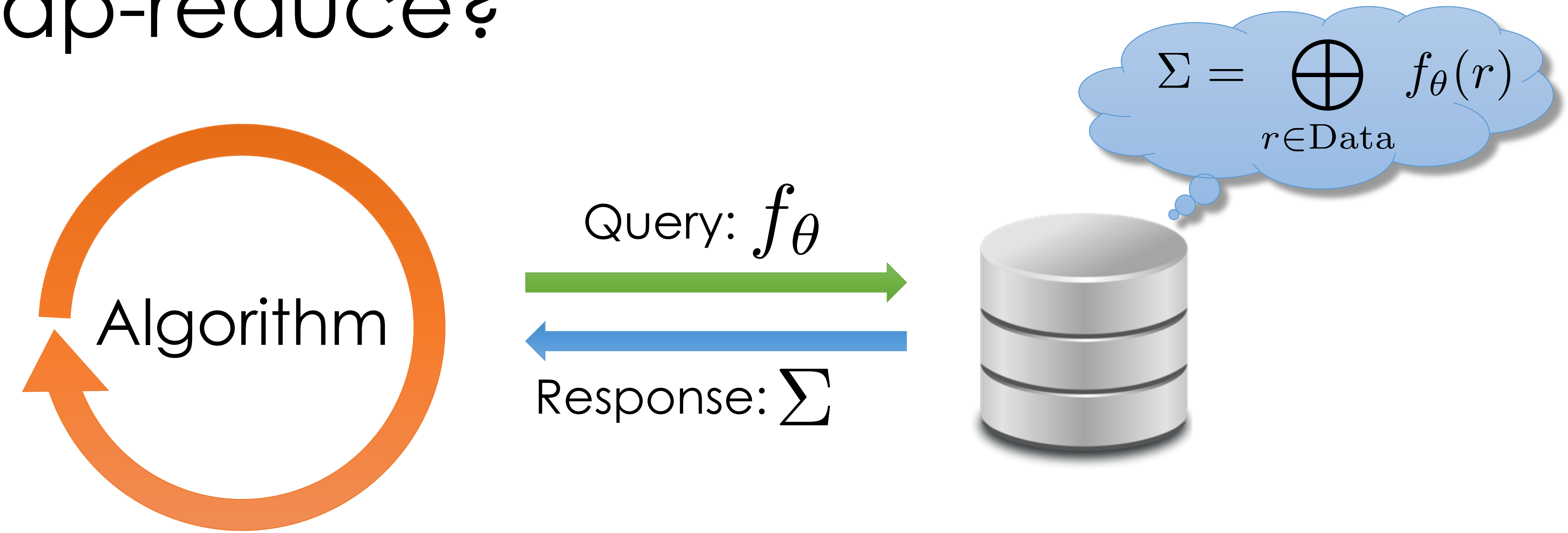
- D. Caragea et al., *A Framework for Learning from Distributed Data Using Sufficient Statistics and Its Application to Learning Decision Trees*. Int. J. Hybrid Intell. Syst. 2004
- Chu et al., *Map-Reduce for Machine Learning on Multicore*. NIPS'06.

*next set of slides are old!

Can we compute

$$\hat{\theta} = (X^T X)^{-1} X^T Y$$

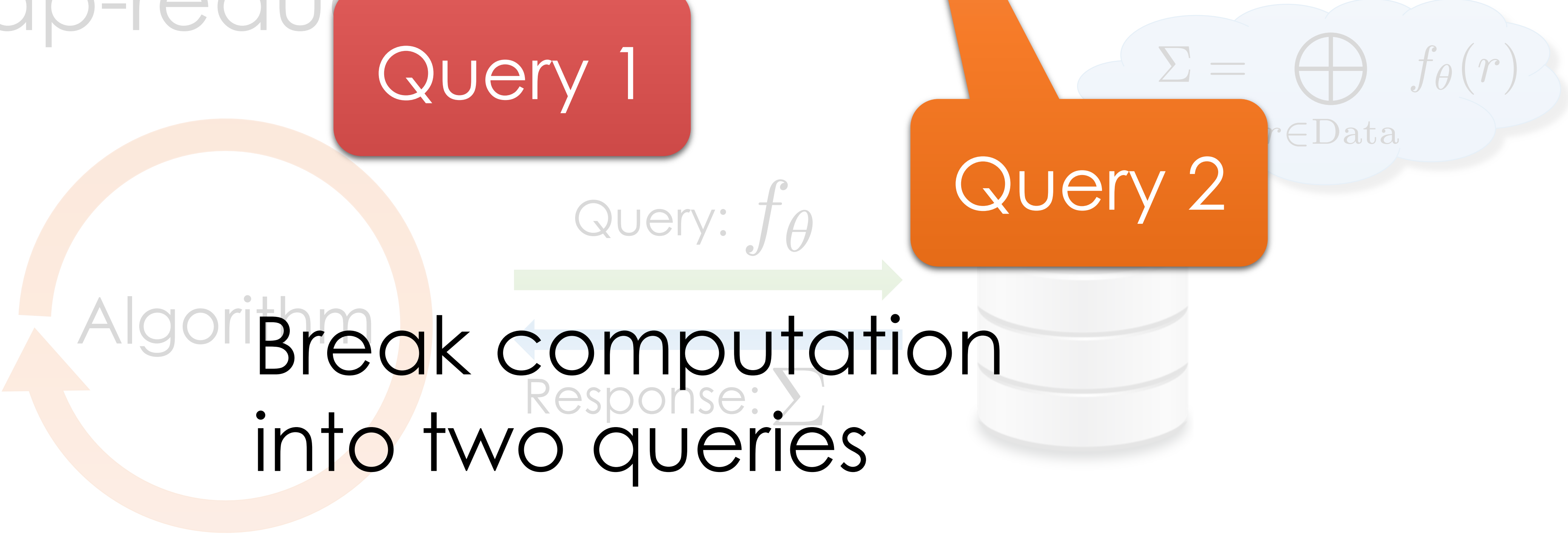
using the statistical query pattern in map-reduce?



Can we compute

$$\hat{\theta} = (\underbrace{X^T X}_{\text{Query 1}})^{-1} \underbrace{X^T Y}_{\text{Query 2}}$$

using the statistical query pattern in map-reduce?



Break computation into two queries

Cost Analysis

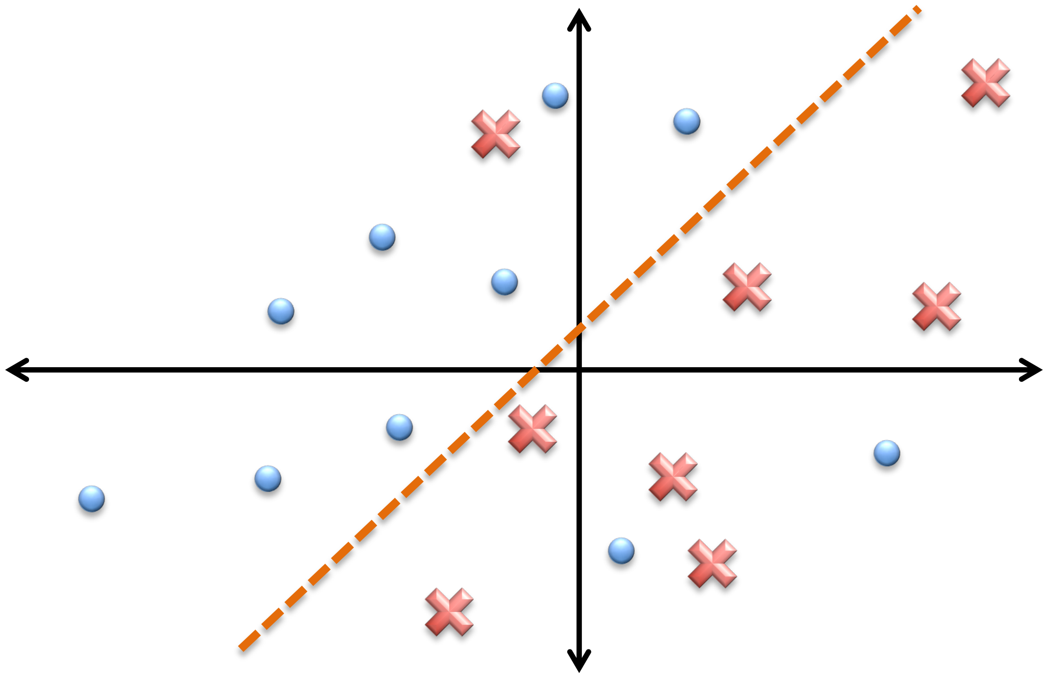
$$\hat{\theta} = (X^T X)^{-1} X^T Y$$

Comutation	Cost
$A = X^T X$	$\Rightarrow O(np^2)$
$C = X^T Y$	$\Rightarrow O(np)$
$B = A^{-1}$	$\Rightarrow O(p^3)$
BC	$\Rightarrow O(p^2)$

When **n >> p** we want to distribute this computation

What about

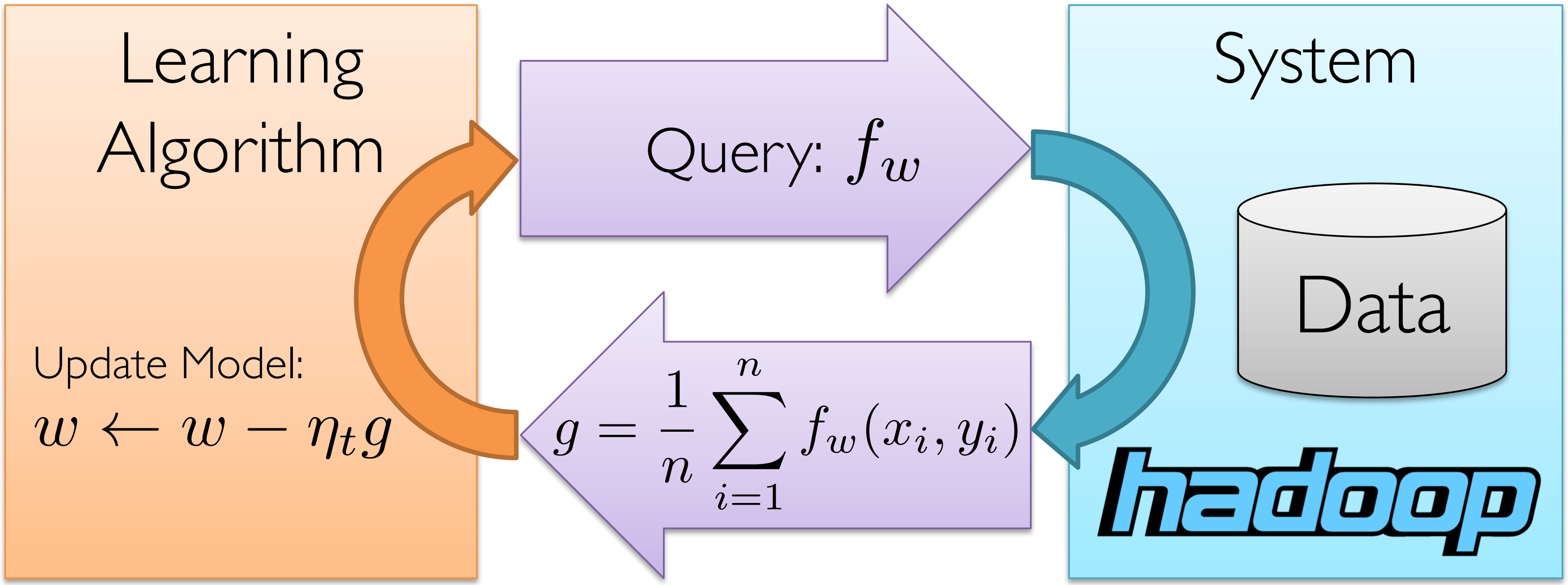
Logistic Regression using **Gradient Descent?**



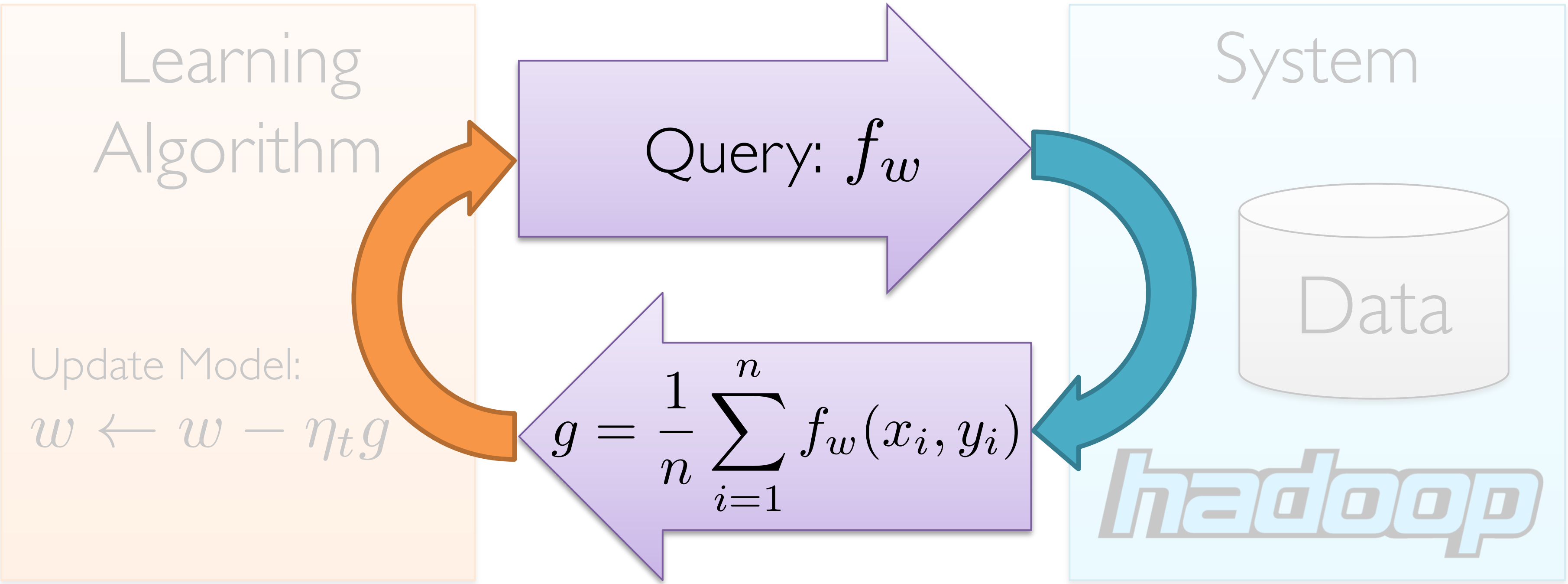
Logistic Regression in Map-Reduce

Gradient descent:

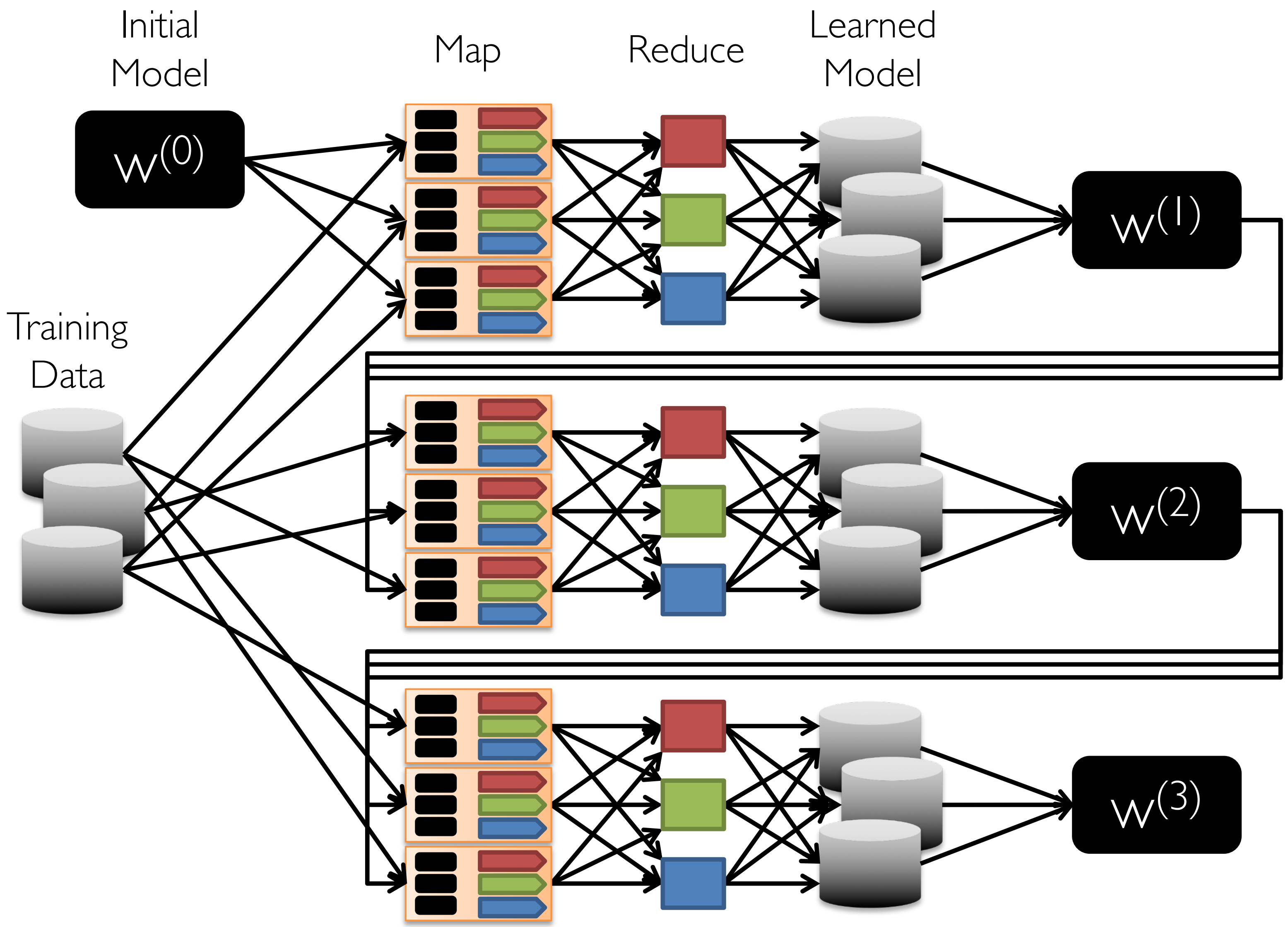
$$f_w(x, y) = \nabla \log L(y, h_w(x))$$



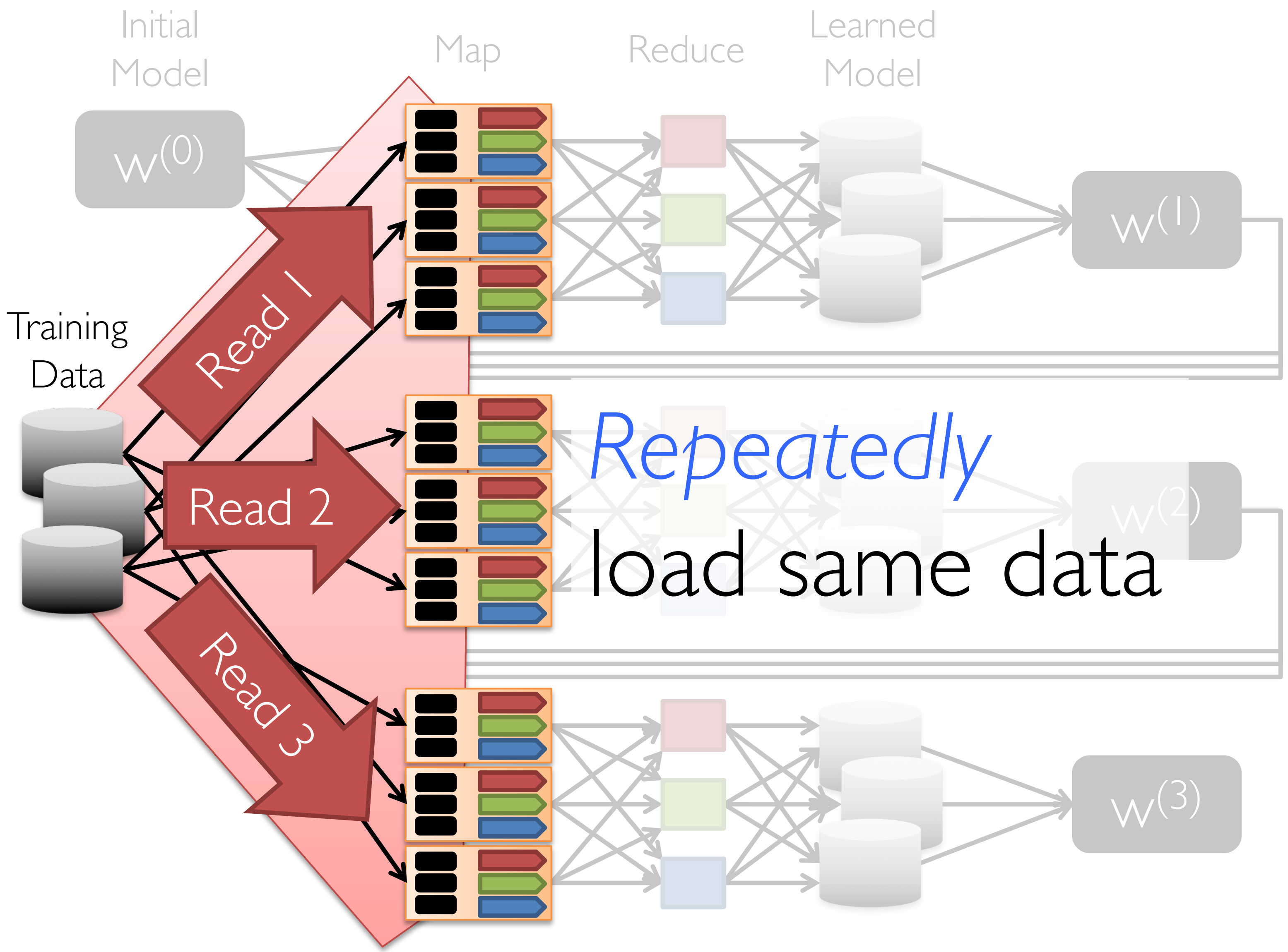
Map-Reduce is not optimized for iteration and multi-stage computation



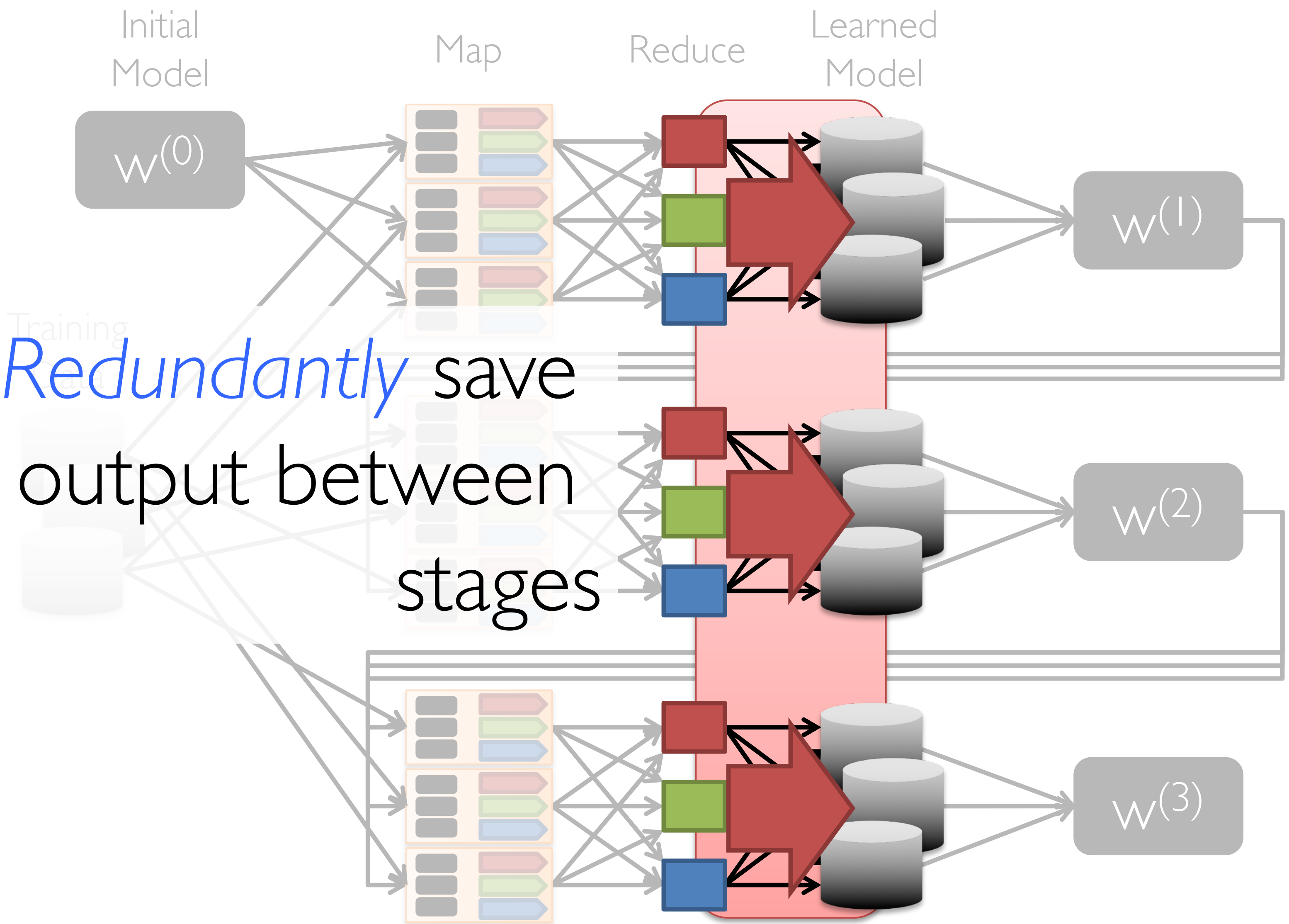
Iteration in Map-Reduce



Cost of Iteration in Map-Reduce



Cost of Iteration in Map-Reduce



Redundantly save output between stages

Spark

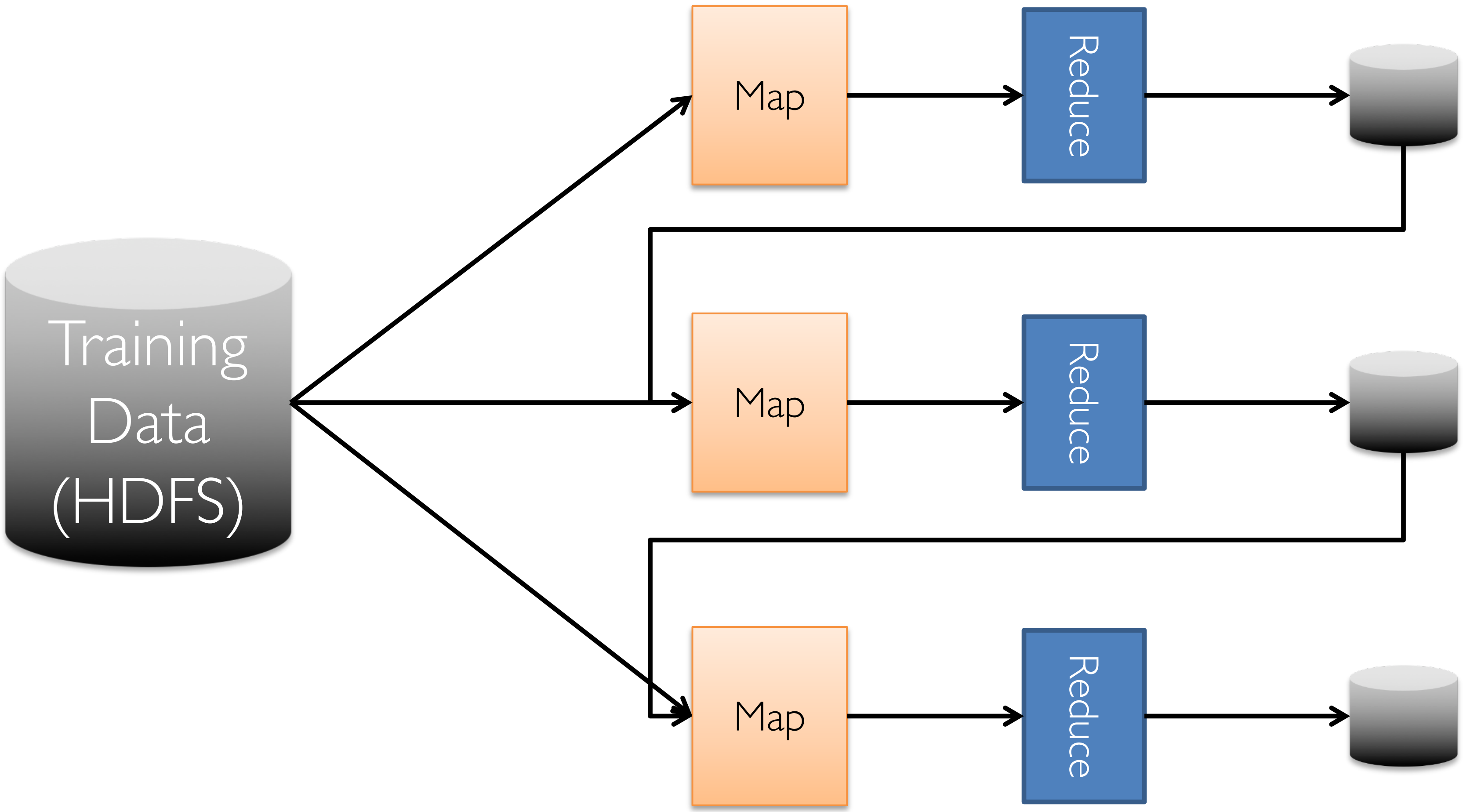
Iteration and Multi-stage computation

In-Memory Dataflow System

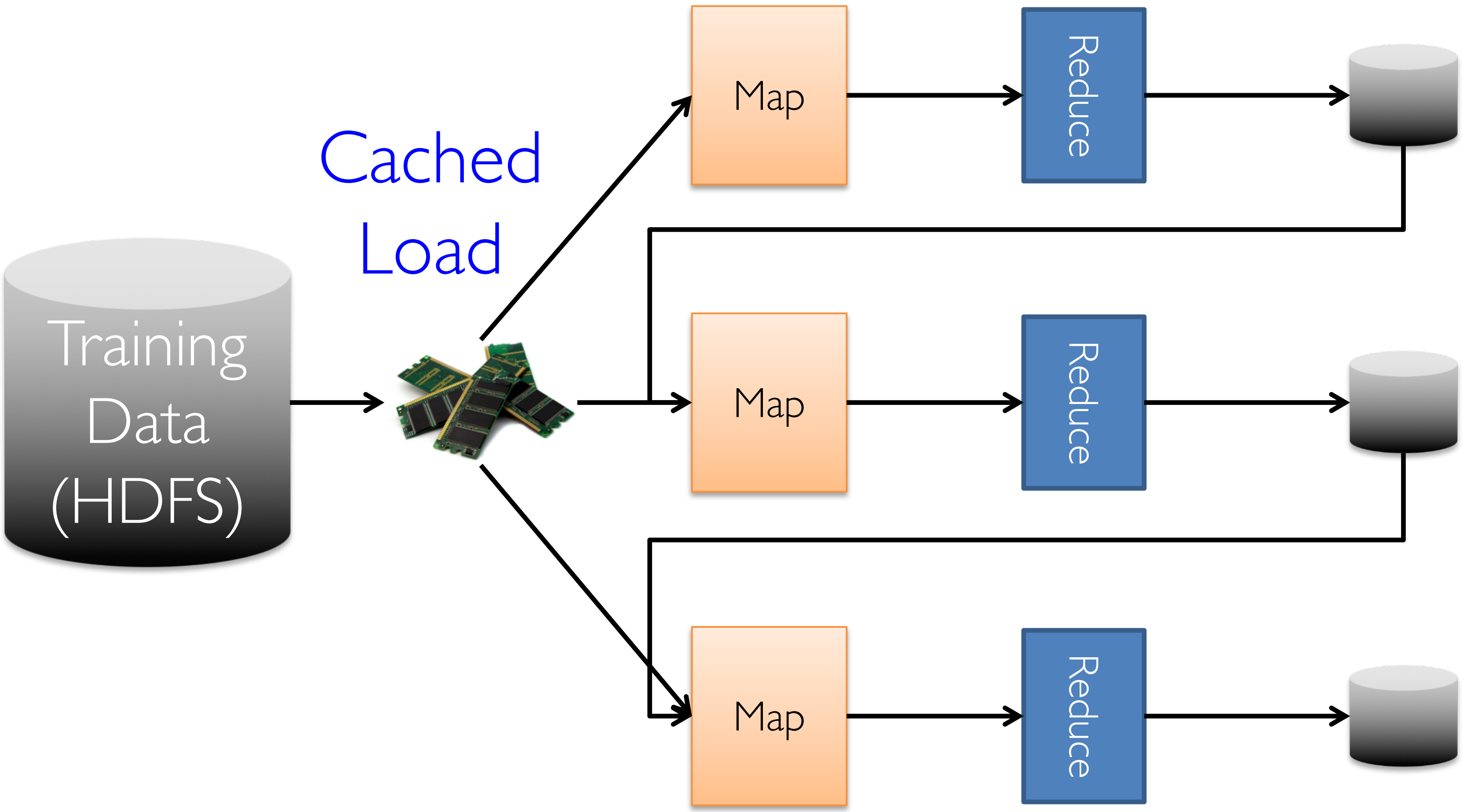
M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. *Spark: cluster computing with working sets.* HotCloud'10

M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M.J. Franklin, S. Shenker, I. Stoica. *Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing*, NSDI 2012

Dataflow View

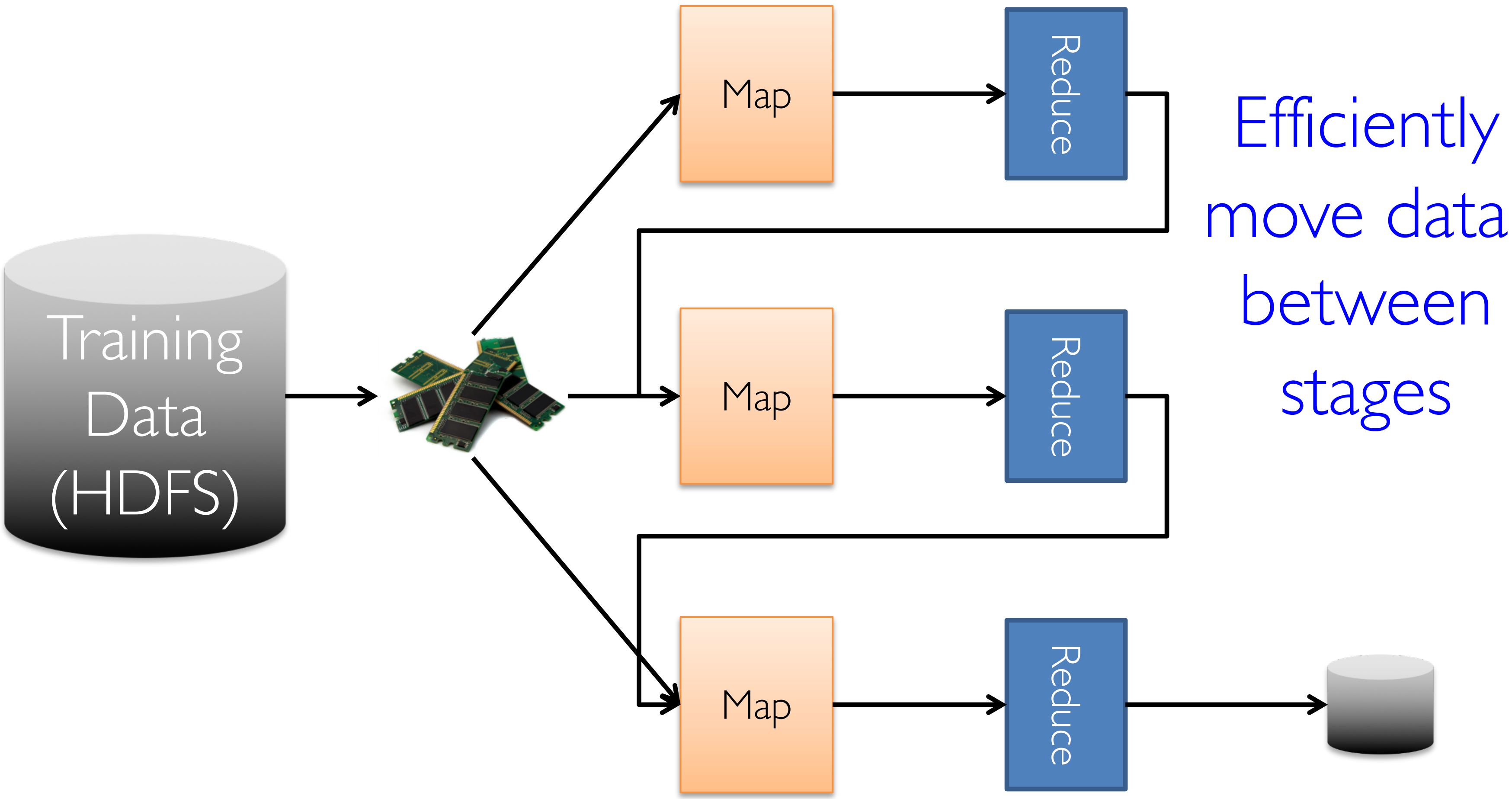


Memory Opt. Dataflow



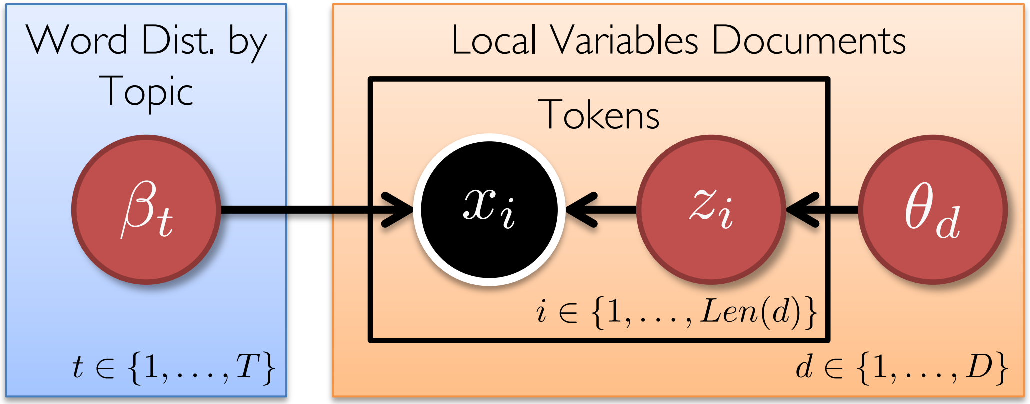
10-100× faster than network and disk

Memory Opt. Dataflow View



Statistical Inference in Large Latent Variable Models

- Large topic models associated variables with each word and document



- Not a good fit for BSP model

Scalable Inference in Latent Variable Models

Amr Ahmed, Mohamed Aly, Joseph Gonzalez,* Shravan Narayanamurthy, Alexander Smola
Yahoo! Research, Santa Clara, CA, USA
{amahmed, aly, jegonzal, shravanm, smola}@yahoo-inc.com

ABSTRACT

General Terms

Algorithms, Experimentation, Performance

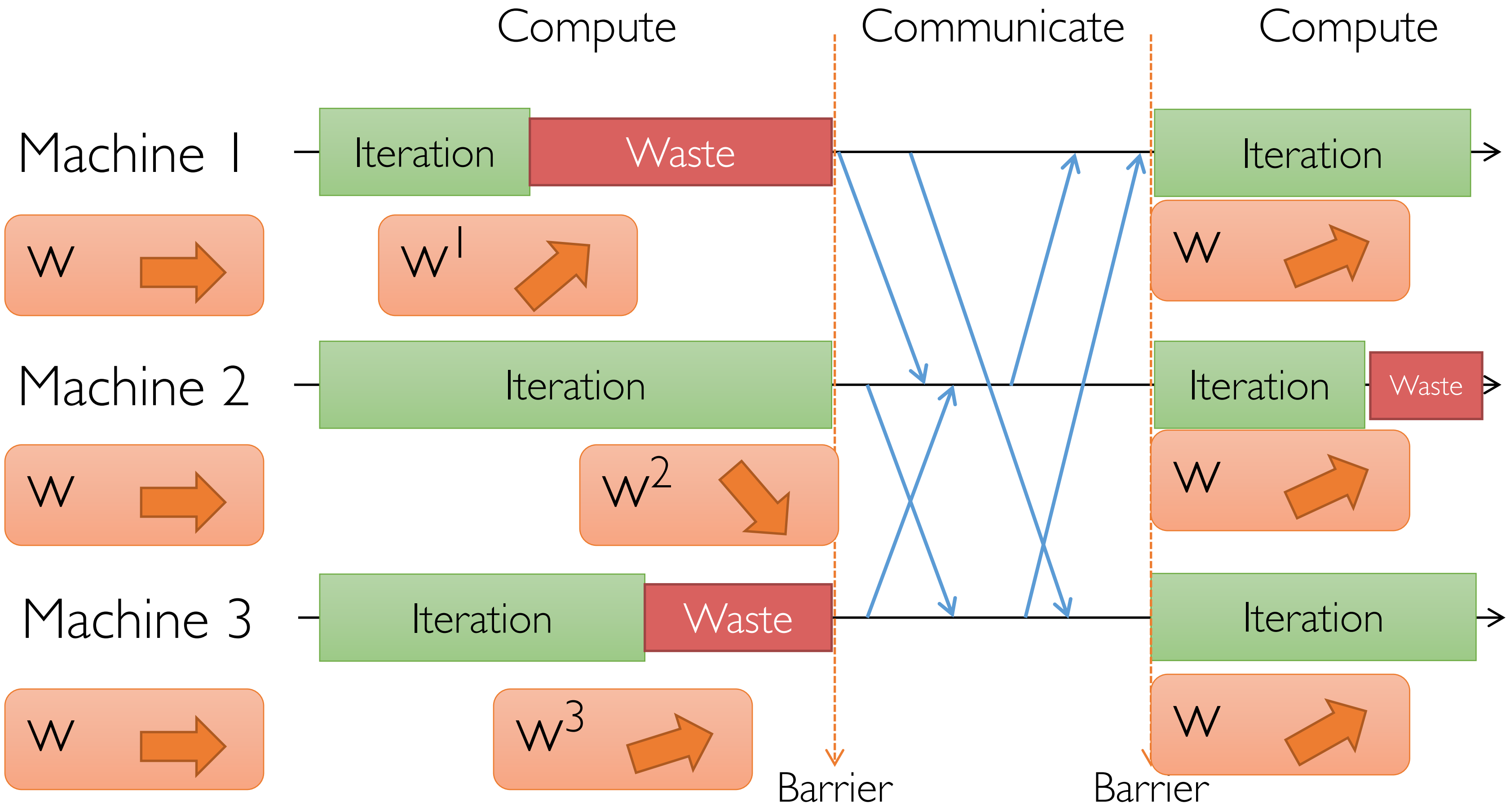
Keywords

Inference, Graphical Models, Large-scale Systems, Latent Models.

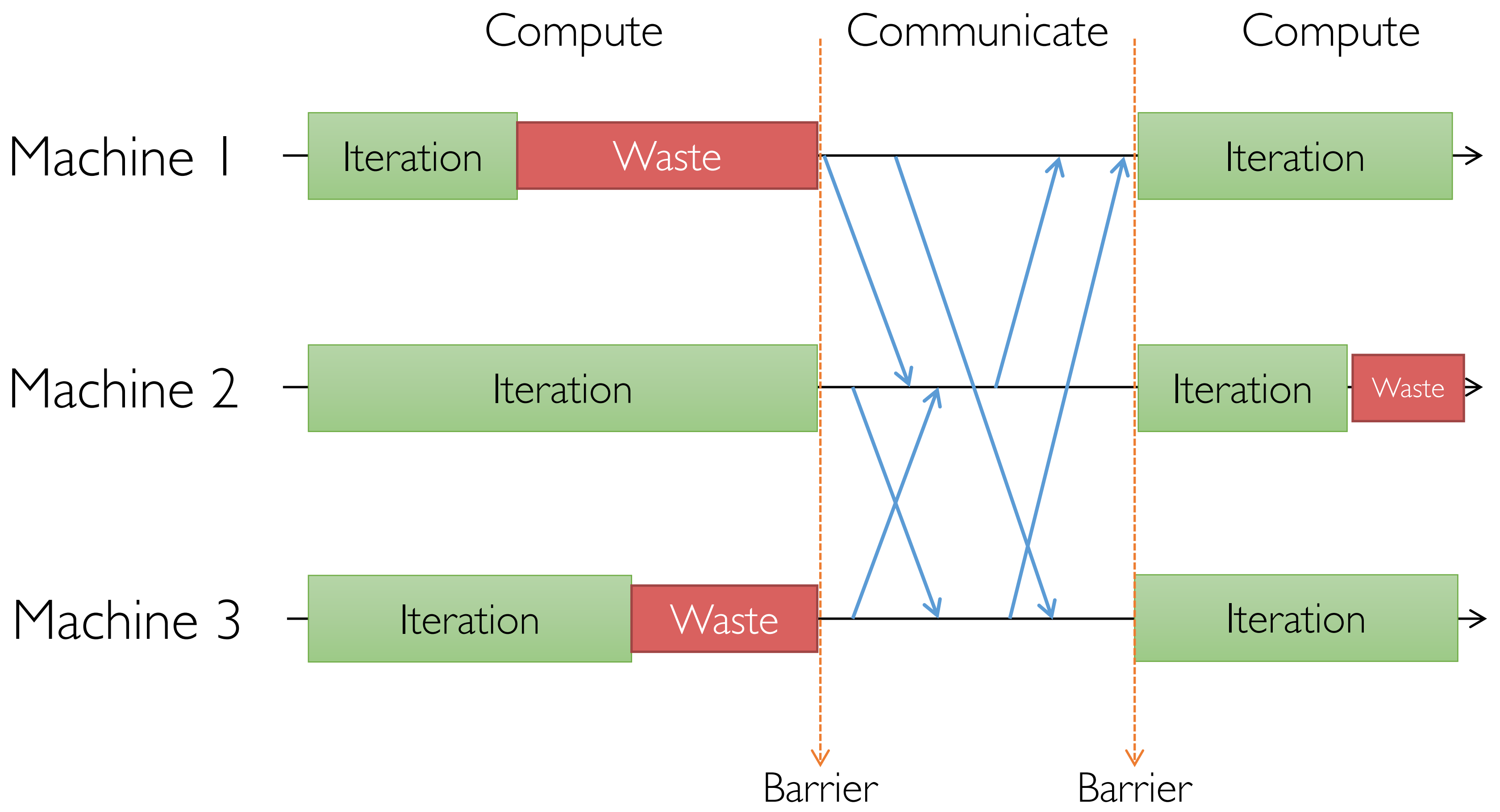
1. INTRODUCTION

Categories and Subject Descriptors

Bulk Synchronous Parallel (BSP) Execution

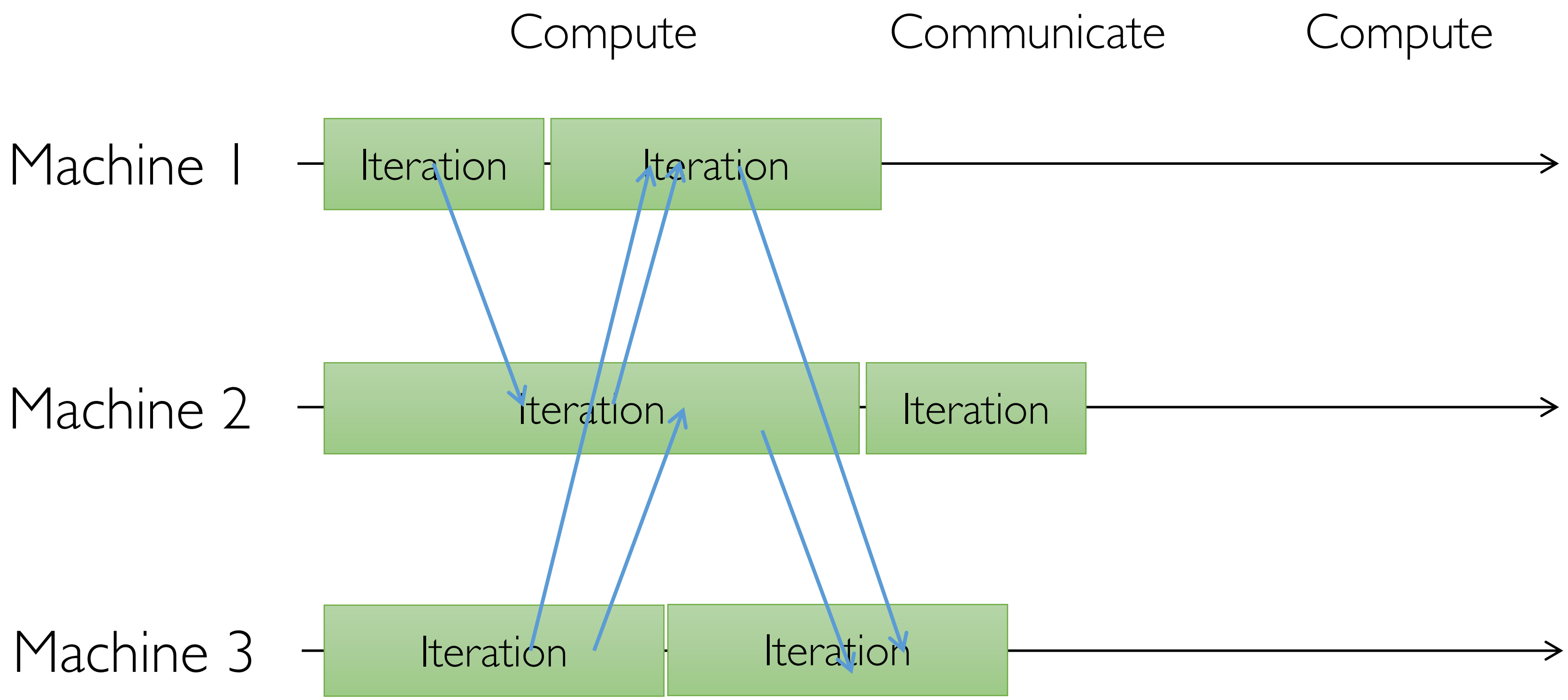


Asynchronous Execution



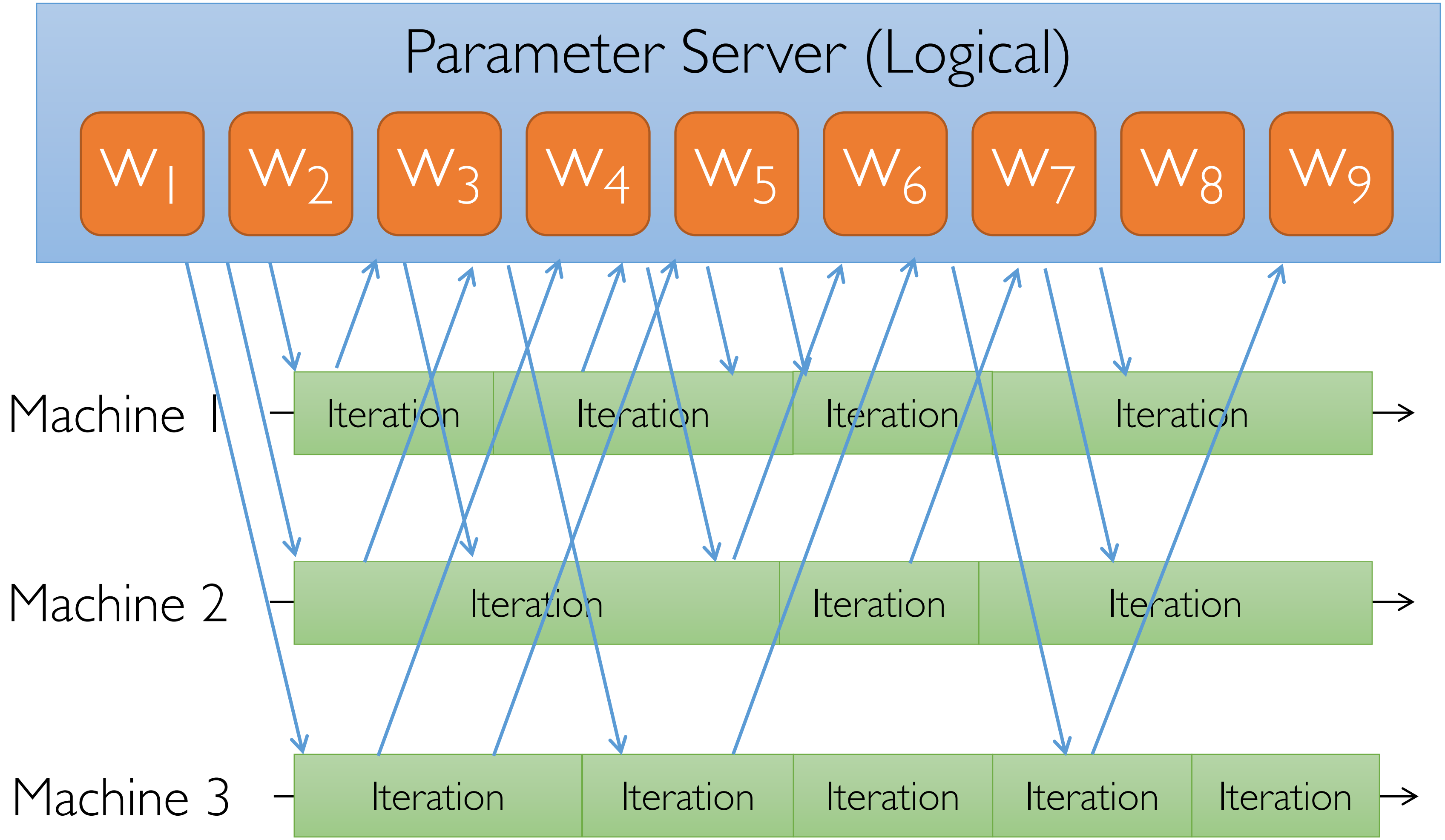
Enable more frequent coordination on parameter values

Asynchronous Execution



Enable more frequent coordination on parameter values

Asynchronous Execution



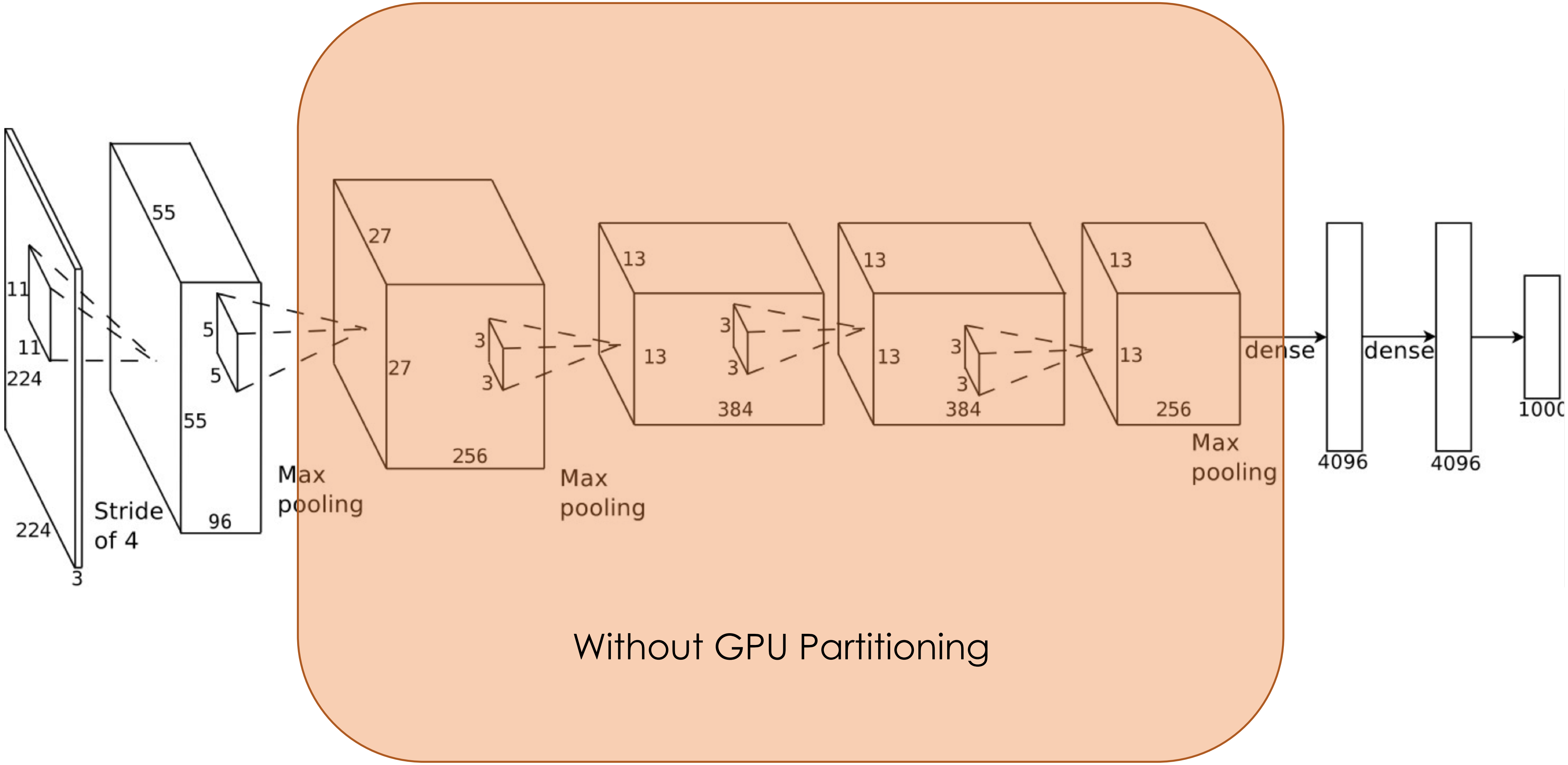
AlexNet

ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky, Illya Sutskever, Geoffrey E. Hinton

TL;DR; This paper describe the deep convolutional architecture, training techniques, and system innovations that resulted in the winning entry for the ILSVRC-2012 Benchmark. This model substantially outperformed the next best model that year.

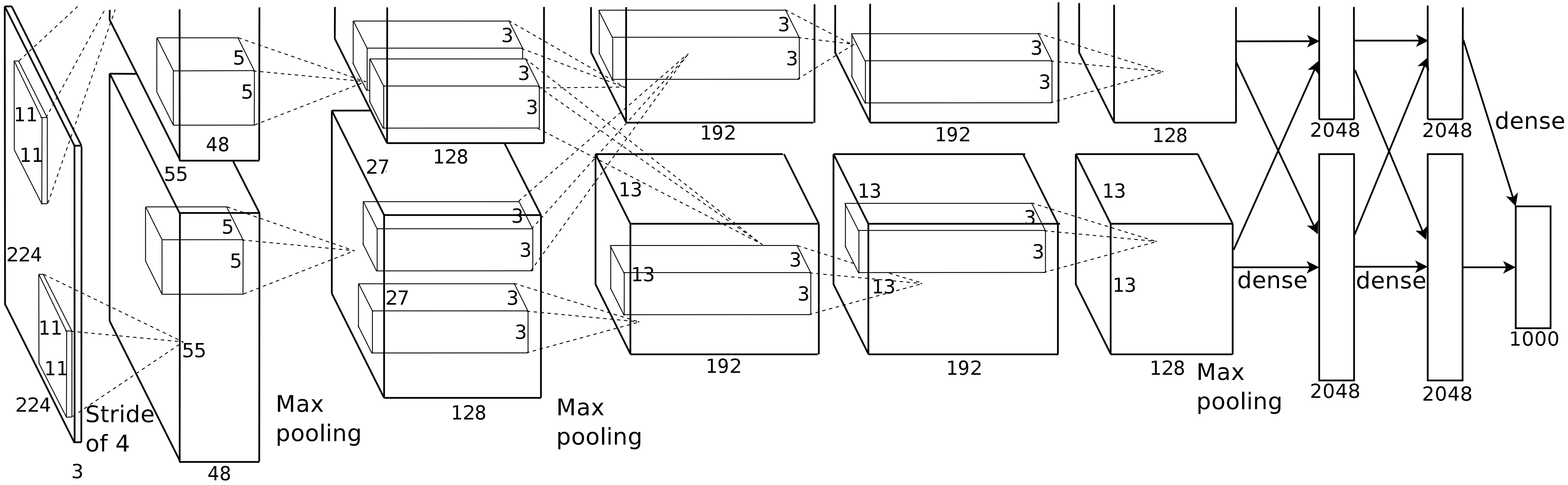
The AlexNet* Architecture



*Posthumously Named

The **Actual** AlexNet* Architecture

from the paper



*Posthumously Named

Training on Multiple GPUs

- Limited by GPU **memory** using Nvidia GTX 580 (3GB RAM)
  - 60M Parameters ~ **240 MB**
  - Need to cache activation maps for backpropagation
    - Batch size = 128
    - 128 * (227*227*3 + 55*55*96*2 + 96*27*27*2 + 256*27*27*2 + 256*13*13*2 + 13*13*384*2 + 256*13*13 + 6*6*256 + 4096 + 4096 + 1000) *4 Bytes ~ **782MB Activations**
    - That is assuming no overhead and single precision values
- Tuned splitting across GPUS to balance communication and computation

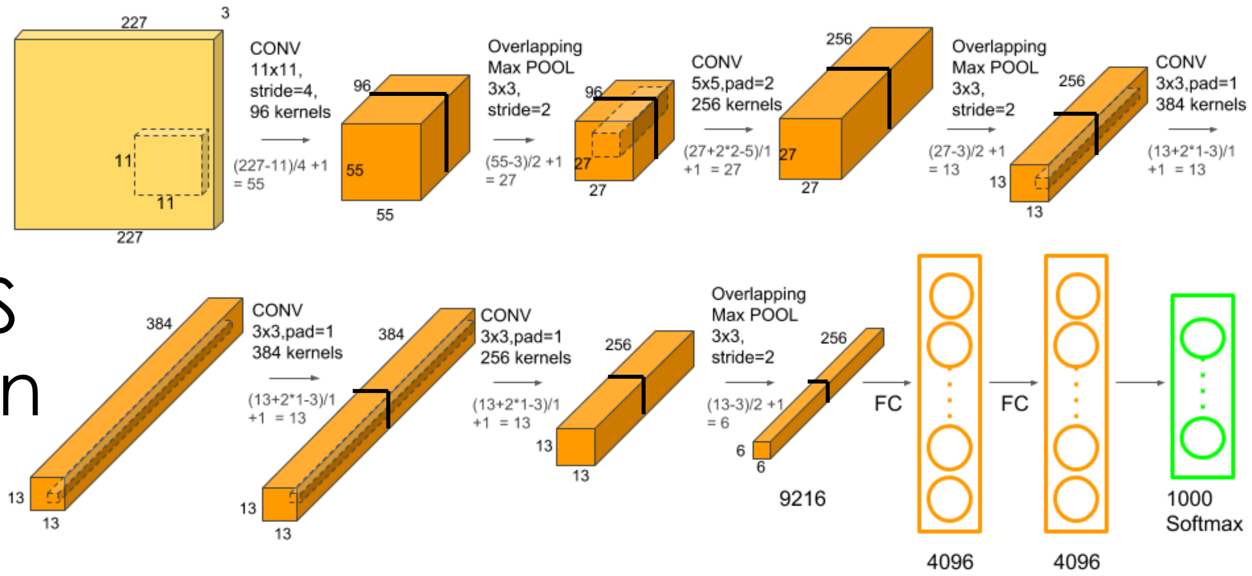


Image from https://neurohive.io/en/popular-networks/alexnet-imagenet-classification-with-deep-convolutional-neural-networks/

Interesting Consequence of Partitioned Training

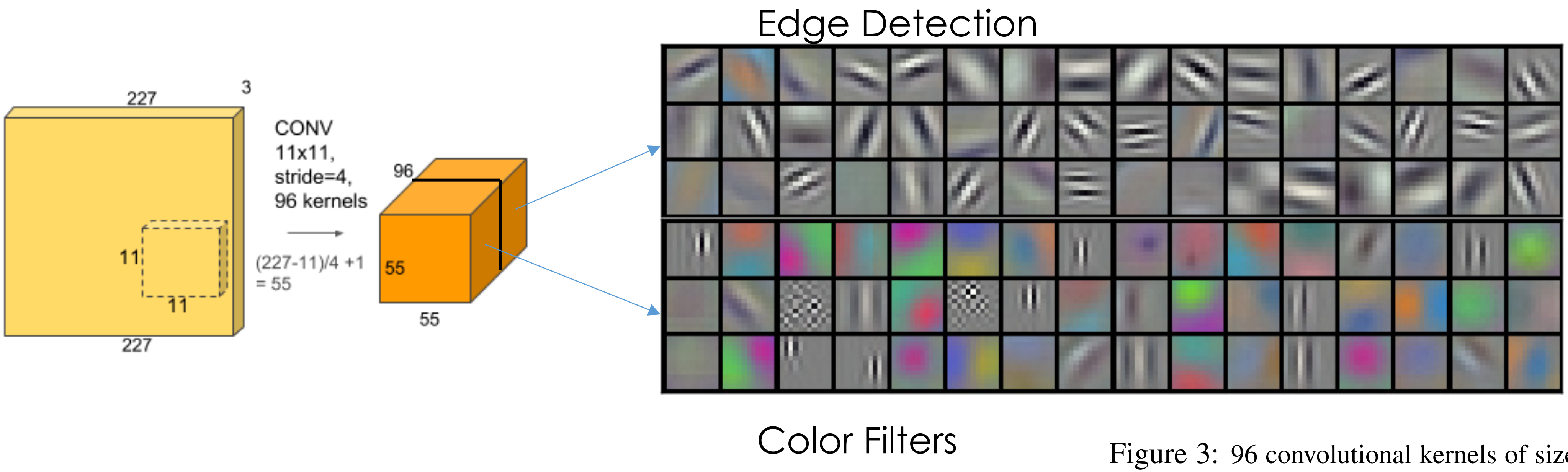
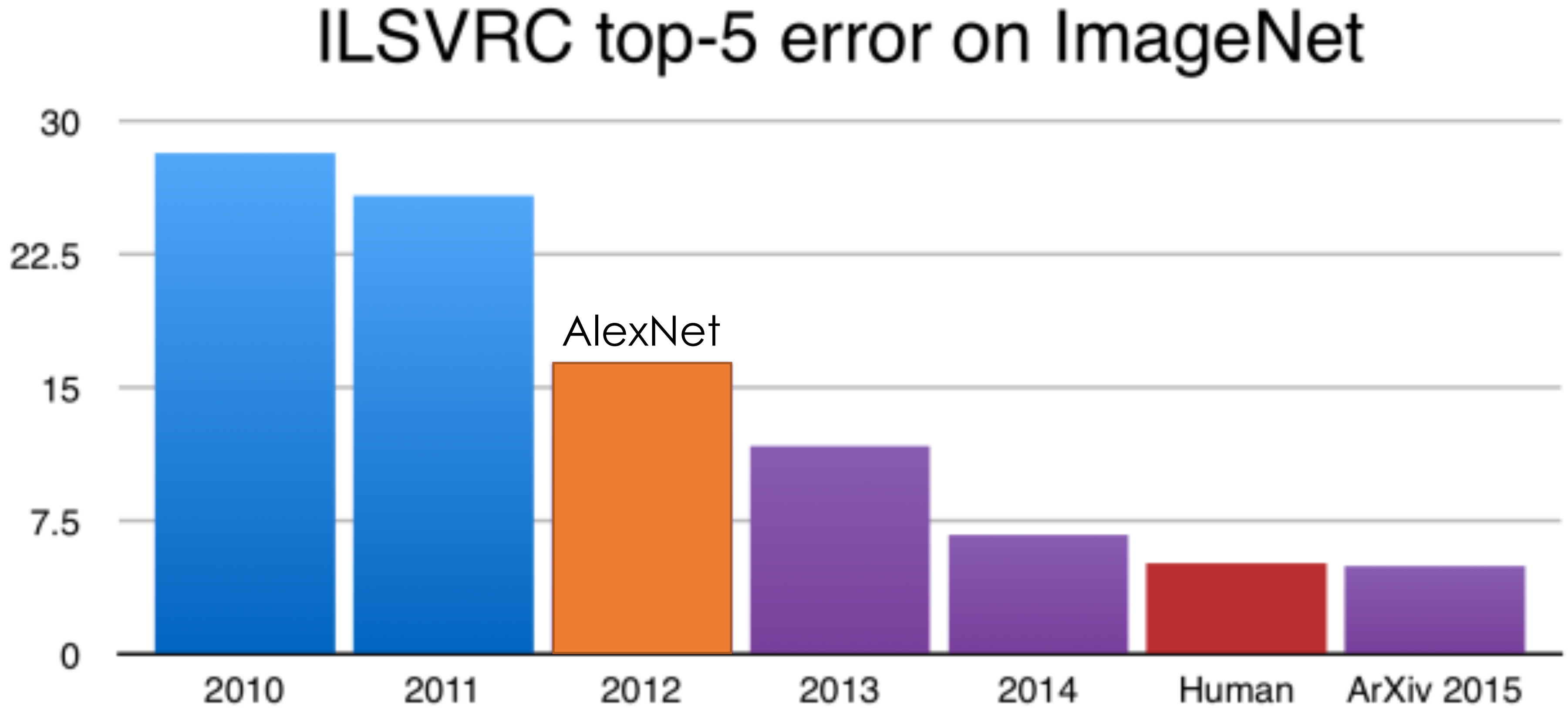


Figure 3: 96 convolutional kernels of size $11\times11\times3$ learned by the first convolutional layer on the $224\times224\times3$ input images. The top 48 kernels were learned on GPU 1 while the bottom 48 kernels were learned on GPU 2. See Section 6.1 for details.

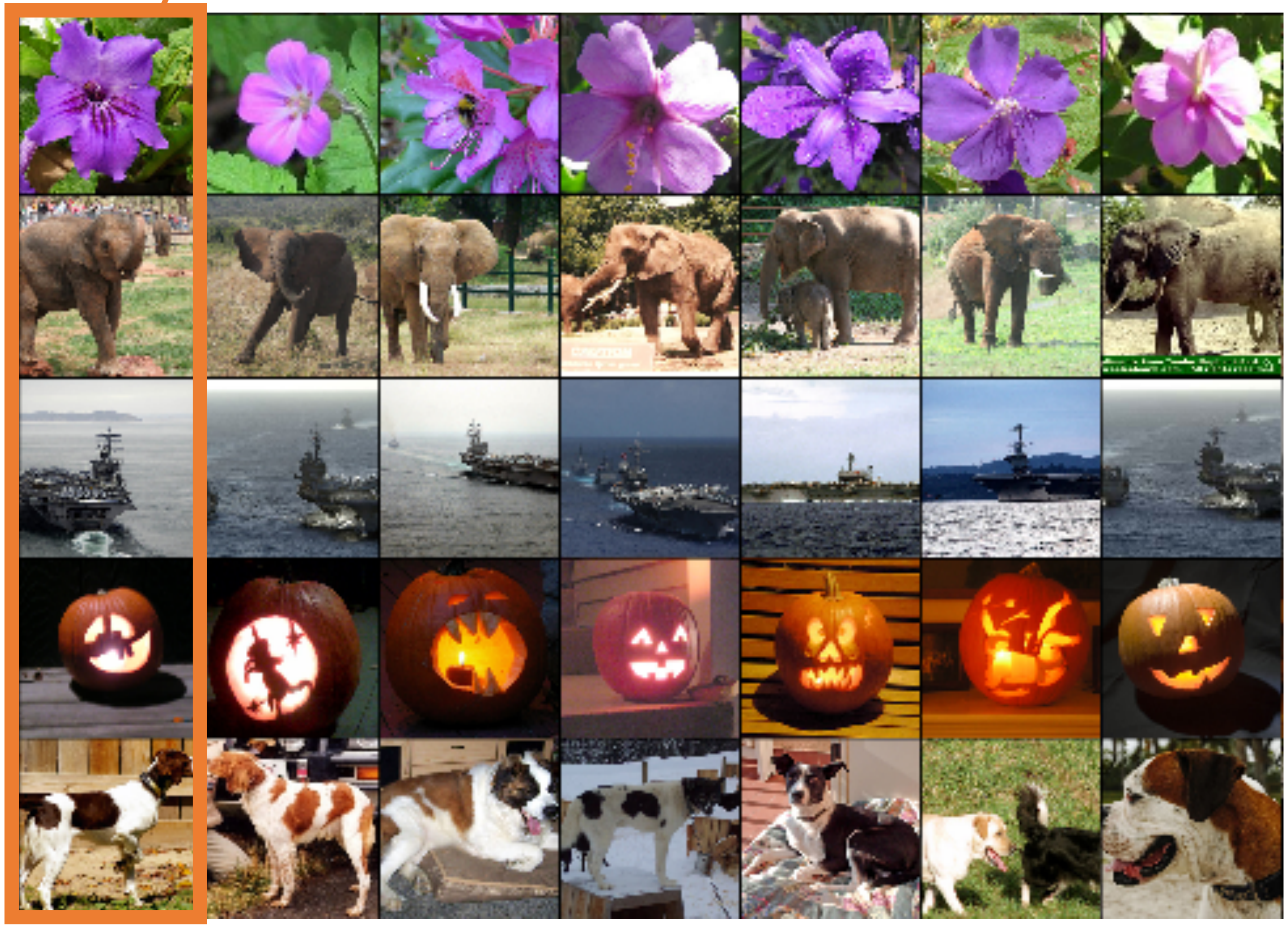
Put into historical context



Good Embeddings …

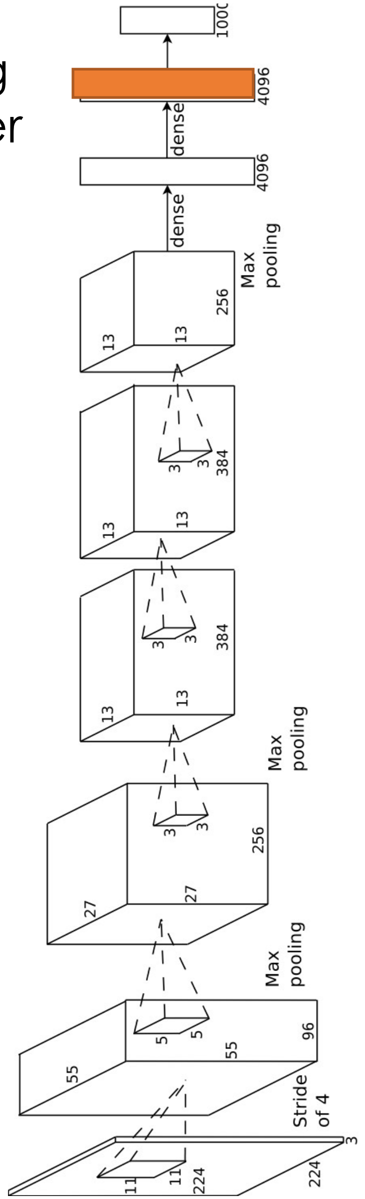
This will later be the foundation of **many** papers

Query



Images with largest dot product with query

Embedding Layer



DistBelief

Large Scale Distributed Deep Networks

Described the system for the 2012 ICML Paper

NIPS 2012 (Same Year as AlexNet)

Large Scale Distributed Deep Networks

Jeffrey Dean, Greg S. Corrado, Rajat Monga, Kai Chen,
Matthieu Devin, Quoc V. Le, Mark Z. Mao, Marc'Aurelio Ranzato,
Andrew Senior, Paul Tucker, Ke Yang, Andrew Y. Ng
{jeff, gcorrado}@google.com
Google Inc., Mountain View, CA

Abstract

Recent work in unsupervised feature learning and deep learning has shown that being able to train large models can dramatically improve performance. In this paper, we consider the problem of training a deep network with billions of parameters using tens of thousands of CPU cores. We have developed a software framework called *DistBelief* that can utilize computing clusters with thousands of machines to train large models. Within this framework, we have developed two algorithms for large-scale distributed training: (i) Downpour SGD, an asynchronous stochastic gradient descent procedure supporting a large number of model replicas, and (ii) Sandblaster, a framework that supports a variety of distributed batch optimization procedures, including a distributed implementation of L-BFGS. Downpour SGD and Sandblaster L-BFGS both increase the scale and speed of deep network ing. We have successfully used our system to train a deep network previously reported in the literature, and achieves state-of- ImageNet, a visual object recognition task with 16 gories. We show that these same technique of a more modestly- sized deep net vice. Although we focus o to training large neura gradient-based machine

Label

DistBelief

1 Introduction

Deep learning and unsupervised feat in many practical applications. State-of-the-art performanc domains, ranging from speech recognition [1, 2], visual object recogn ocessing [5, 6].

It has also been observed that increasin cale of deep learning, with respect to the number of training examples, the number of model parameters, or both, can drastically improve ultimate classification accuracy [3, 4, 7]. These results have led to a surge of interest in scaling up the training and inference algorithms used for these models [8] and in improving applicable optimization procedures [7, 9]. The use of GPUs [1, 2, 3, 8] is a significant advance in recent years that makes

Building High-level Features Using Large Scale Unsupervised Learning

Quoc V. Le
Marc'Aurelio Ranzato
Rajat Monga
Matthieu Devin
Kai Chen
Greg S. Corrado
Jeff Dean
Andrew Y. Ng

Abstrac

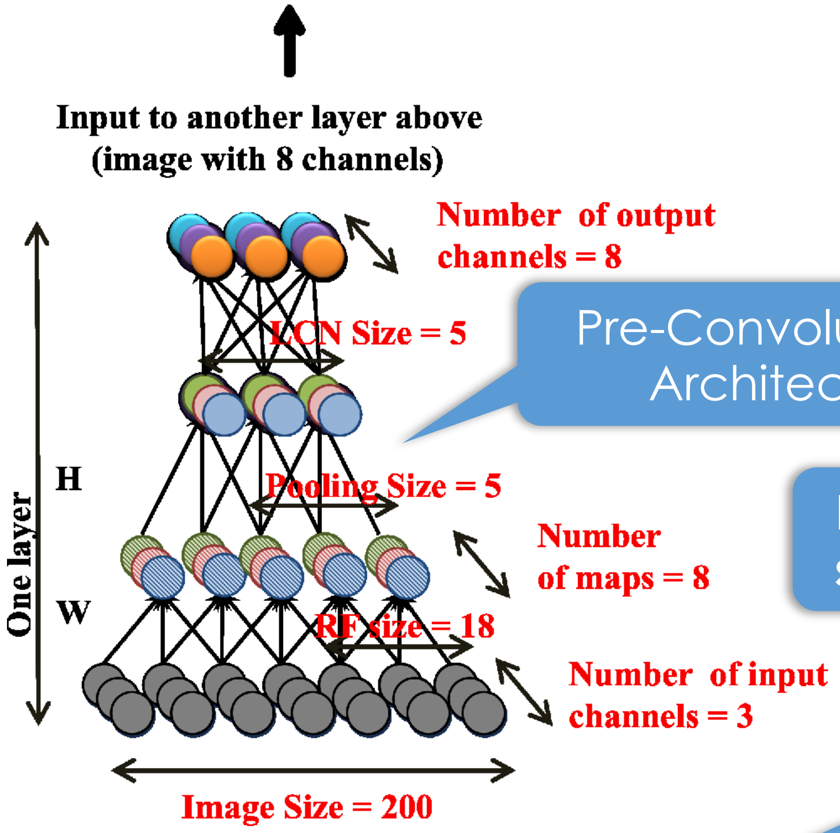
We consider the problem level, class-specific featu only unlabeled data. F possible to learn a face d unlabeled images using To answer this, we train connected sparse autoenc and local contrast normal dataset of images (the lion connections, the dataset has 10 million

but current experimental evidence suggests the possibility that some neurons in the temporal cortex are

Building High-Level Features Using Large Scale Unsupervised Learning

ICML 2012

9x "deep"



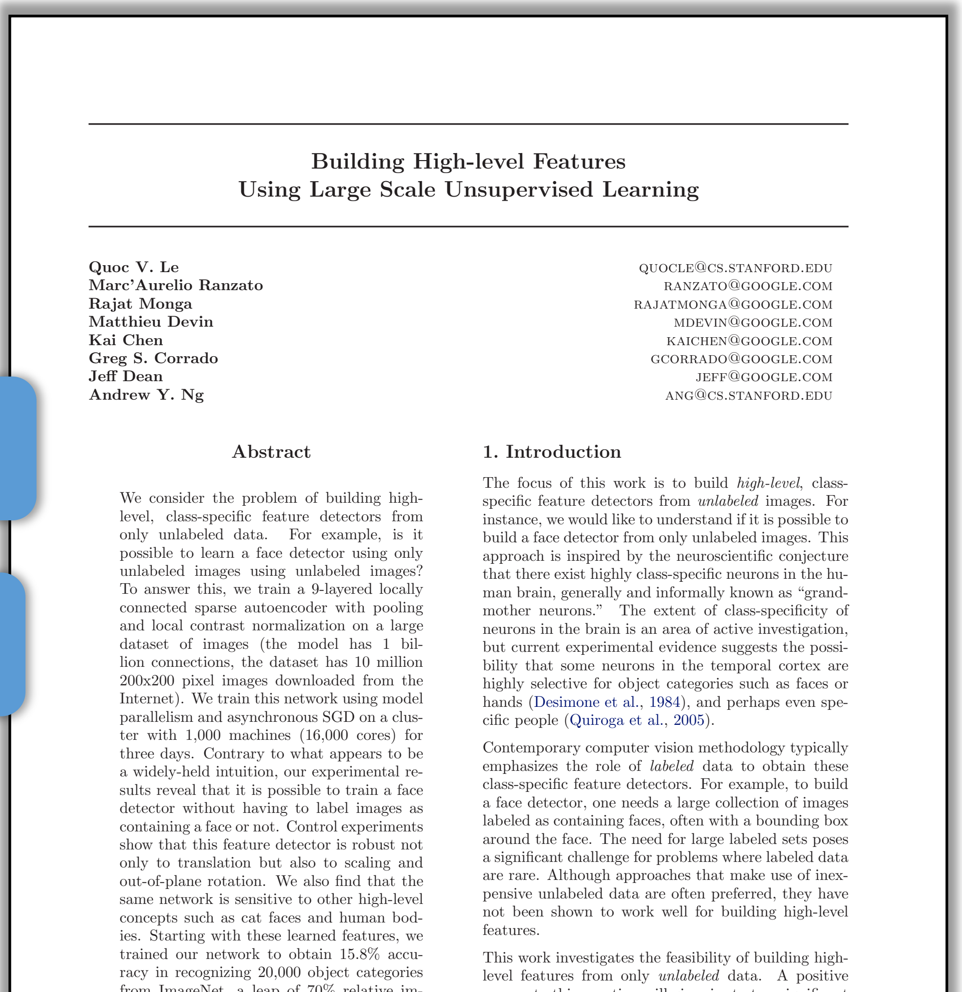
Pre-Convolutional* Architecture

Each Neuron has **separate** weights

sparse connectivity

30x bigger than other deep nets.

~1 billion parameters!



Building High-level Features
Using Large Scale Unsupervised Learning

Quoc V. Le QUOCLE@CS.STANFORD.EDU
Marc'Aurelio Ranzato RANZATO@GOOGLE.COM
Rajat Monga RAJATMONGA@GOOGLE.COM
Matthieu Devin MDEVIN@GOOGLE.COM
Kai Chen KAICHEN@GOOGLE.COM
Greg S. Corrado GCORRADO@GOOGLE.COM
Jeff Dean JEFF@GOOGLE.COM
Andrew Y. Ng ANG@CS.STANFORD.EDU

Abstract

We consider the problem of building high-level, class-specific feature detectors from only unlabeled data. For example, is it possible to learn a face detector using only unlabeled images using unlabeled images? To answer this, we train a 9-layered locally connected sparse autoencoder with pooling and local contrast normalization on a large dataset of images (the model has 1 billion connections, the dataset has 10 million 200x200 pixel images downloaded from the Internet). We train this network using model parallelism and asynchronous SGD on a cluster with 1,000 machines (16,000 cores) for three days. Contrary to what appears to be a widely-held intuition, our experimental results reveal that it is possible to train a face detector without having to label images as containing a face or not. Control experiments show that this feature detector is robust not only to translation but also to scaling and out-of-plane rotation. We also find that the same network is sensitive to other high-level concepts such as cat faces and human bodies. Starting with these learned features, we trained our network to obtain 15.8% accuracy in recognizing 20,000 object categories from ImageNet, a leap of 70% relative im-

1. Introduction

The focus of this work is to build *high-level*, class-specific feature detectors from *unlabeled* images. For instance, we would like to understand if it is possible to build a face detector from only unlabeled images. This approach is inspired by the neuroscientific conjecture that there exist highly class-specific neurons in the human brain, generally and informally known as "grandmother neurons." The extent of class-specificity of neurons in the brain is an area of active investigation, but current experimental evidence suggests the possibility that some neurons in the temporal cortex are highly selective for object categories such as faces or hands (Desimone et al., 1984), and perhaps even specific people (Quiroga et al., 2005).

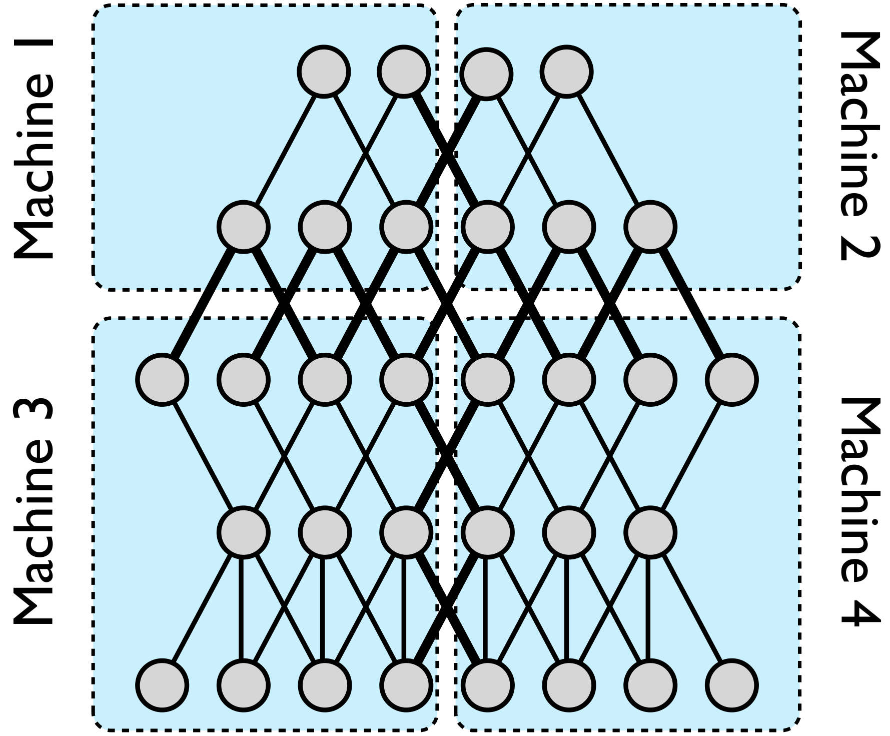
Contemporary computer vision methodology typically emphasizes the role of *labeled* data to obtain these class-specific feature detectors. For example, to build a face detector, one needs a large collection of images labeled as containing faces, often with a bounding box around the face. The need for large labeled sets poses a significant challenge for problems where labeled data are rare. Although approaches that make use of inexpensive unlabeled data are often preferred, they have not been shown to work well for building high-level features.

This work investigates the feasibility of building high-level features from only *unlabeled* data. A positive

*This pre-dates AlexNet but is two decades after LeNet.

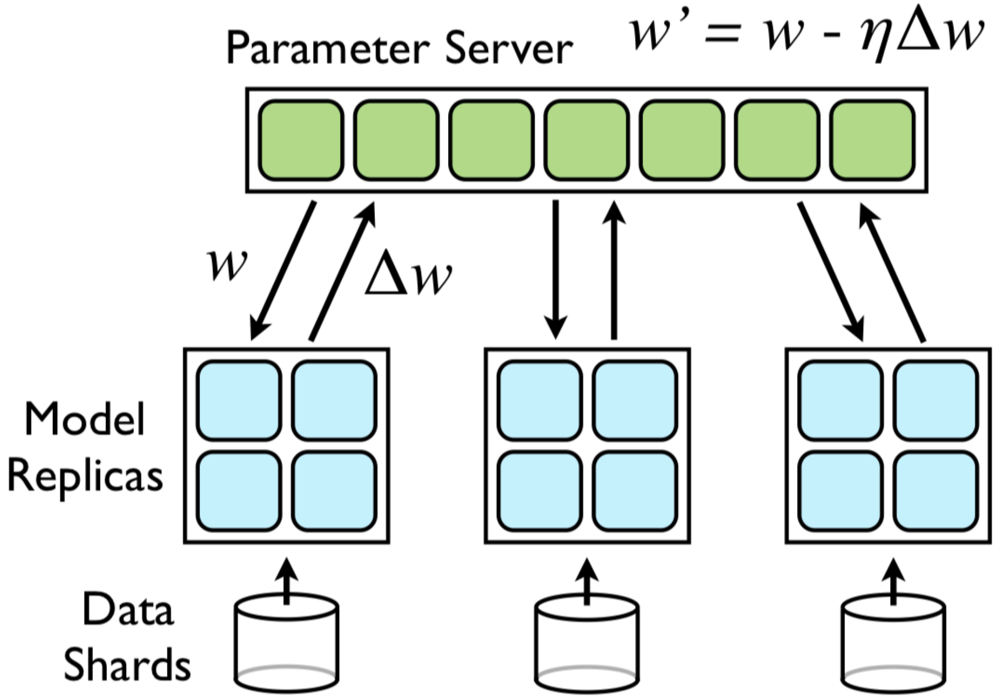
Combine Model and Data Parallelism

Model Parallelism



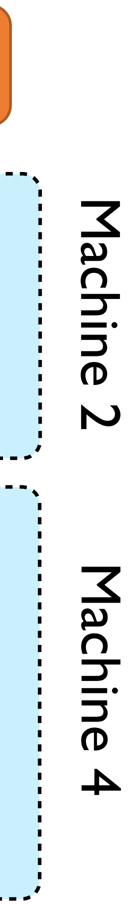
This appears in earlier work on graph systems …

Data Parallelism

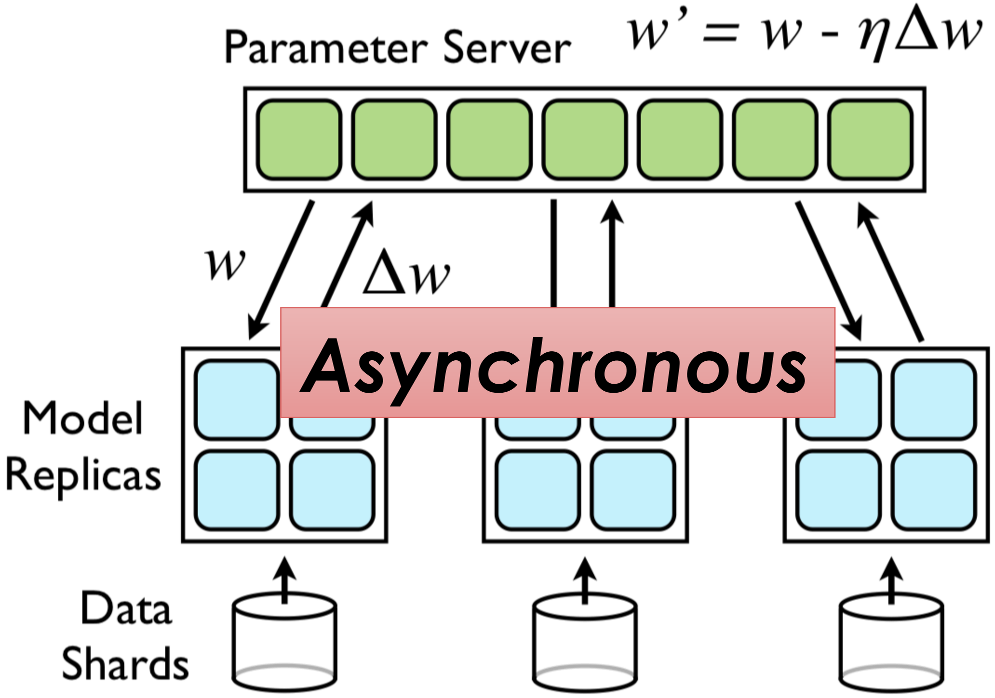


Downpour SGD

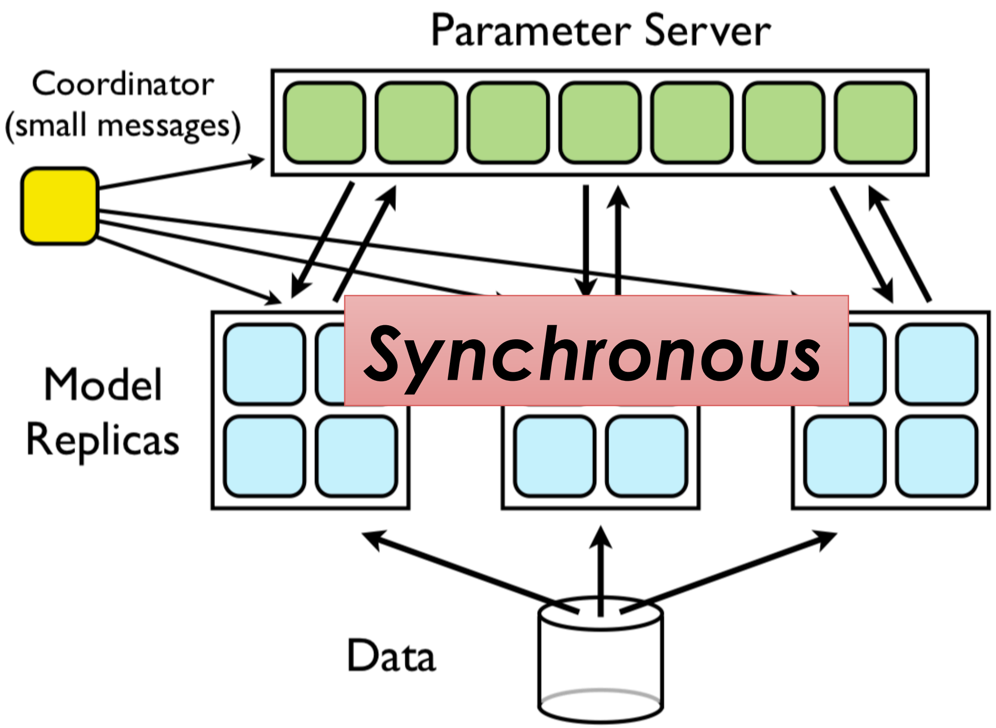
Combine Model and Data Parallelism



Data Parallelism

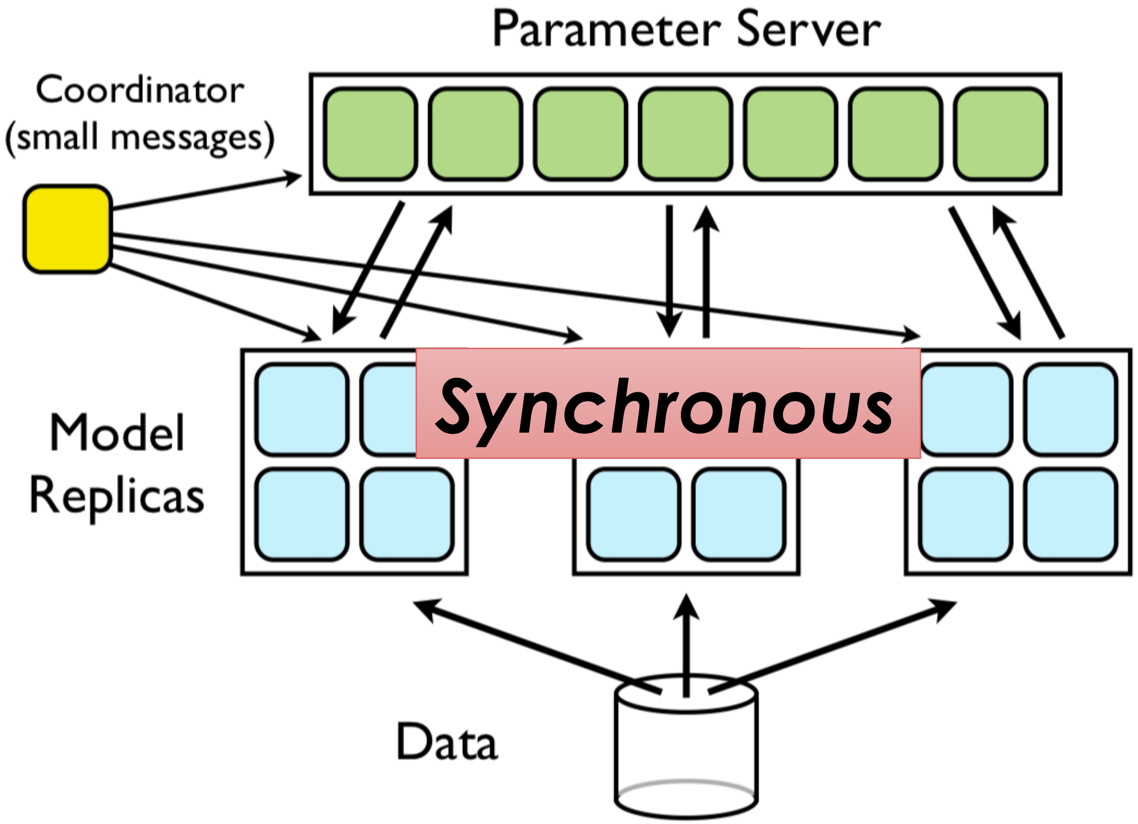


Downpour SGD



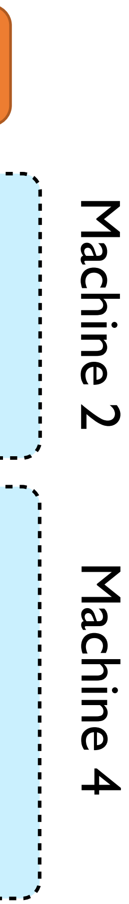
Sandblaster L-BFGS

Sandblaster L-BFGS

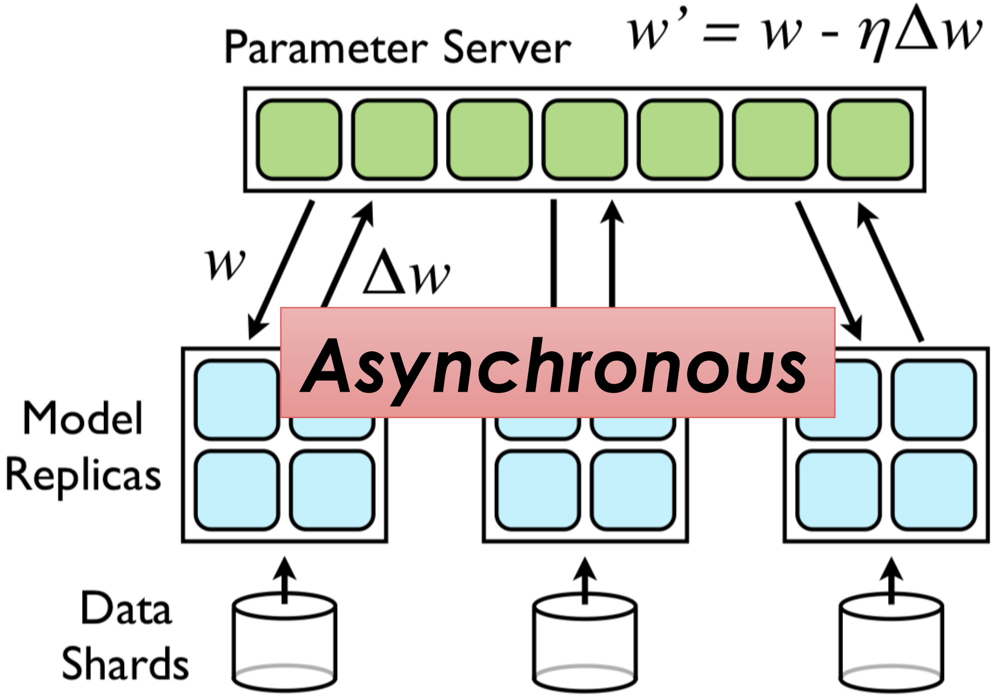


- L-BFGS
  - Commonly used for convex opt. problems
  - Requires repeated scans of all data
  - Robust, minimal tuning
- Naturally fits map-reduce pattern
- **Innovations:**
  - accumulate gradients and store outputs in a sharded key value store (parameter server)
  - Tiny tasks + backup tasks to mitigate stragglers

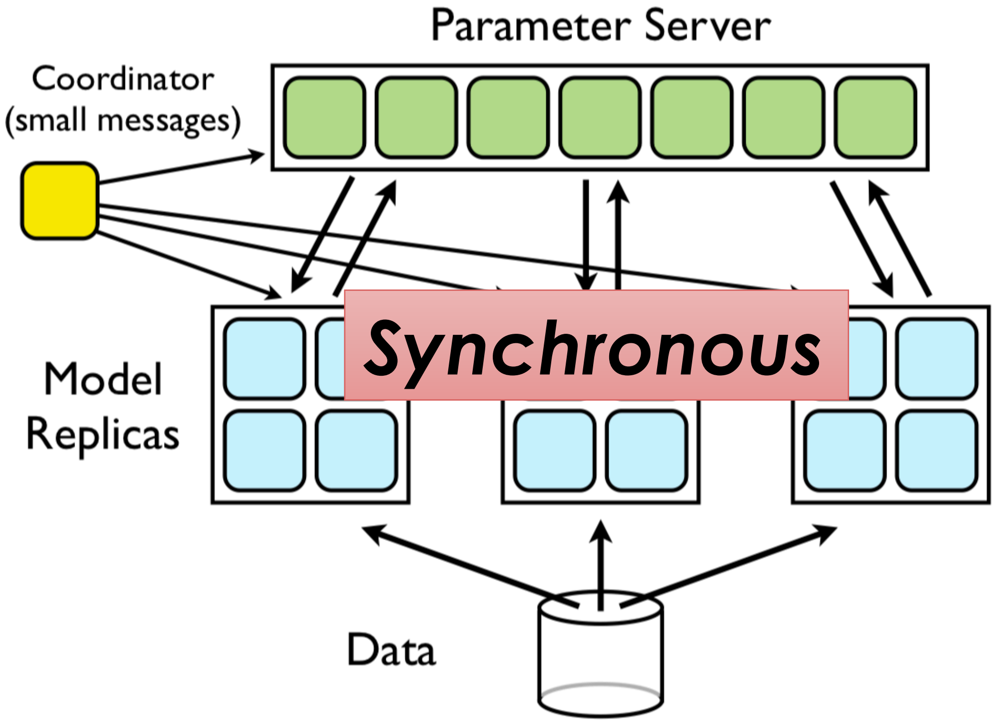
Combine Model and Data Parallelism



Data Parallelism



Downpour SGD

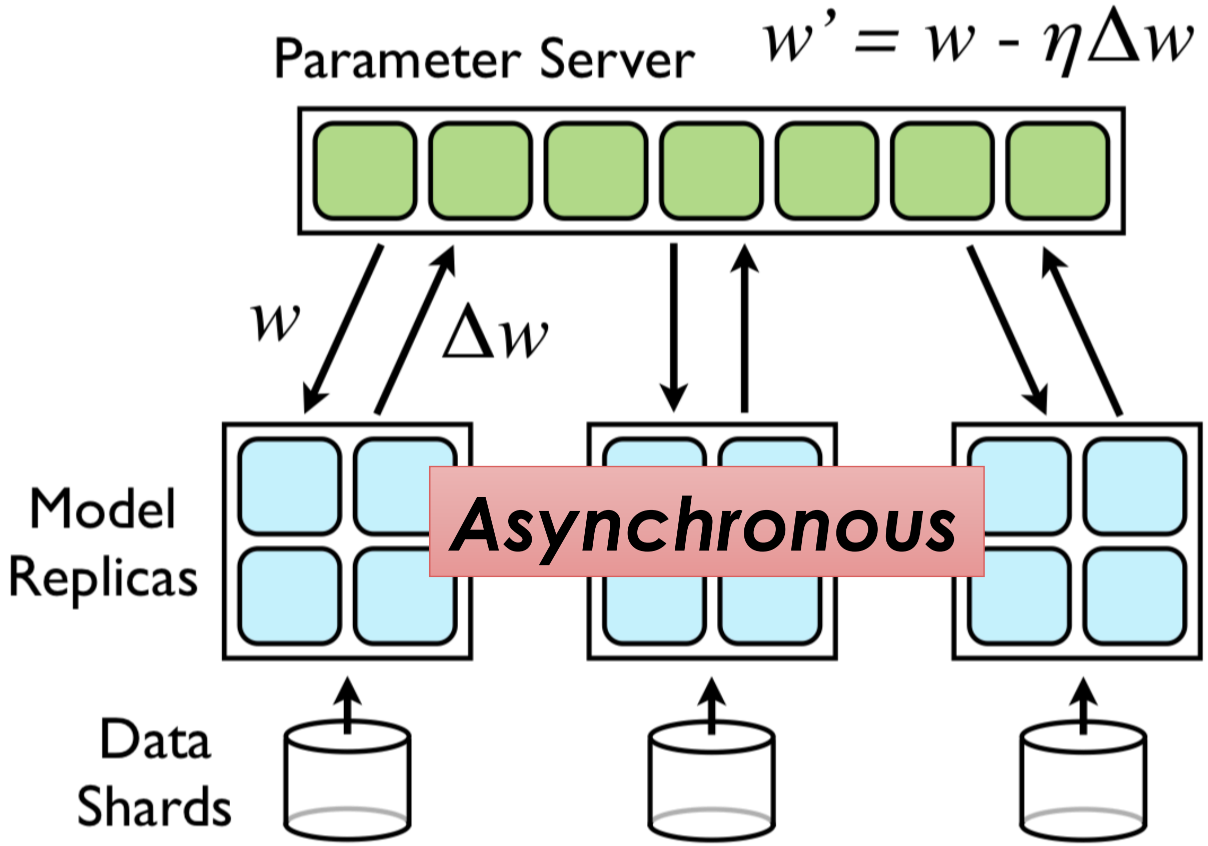


Sandblaster L-BFGS

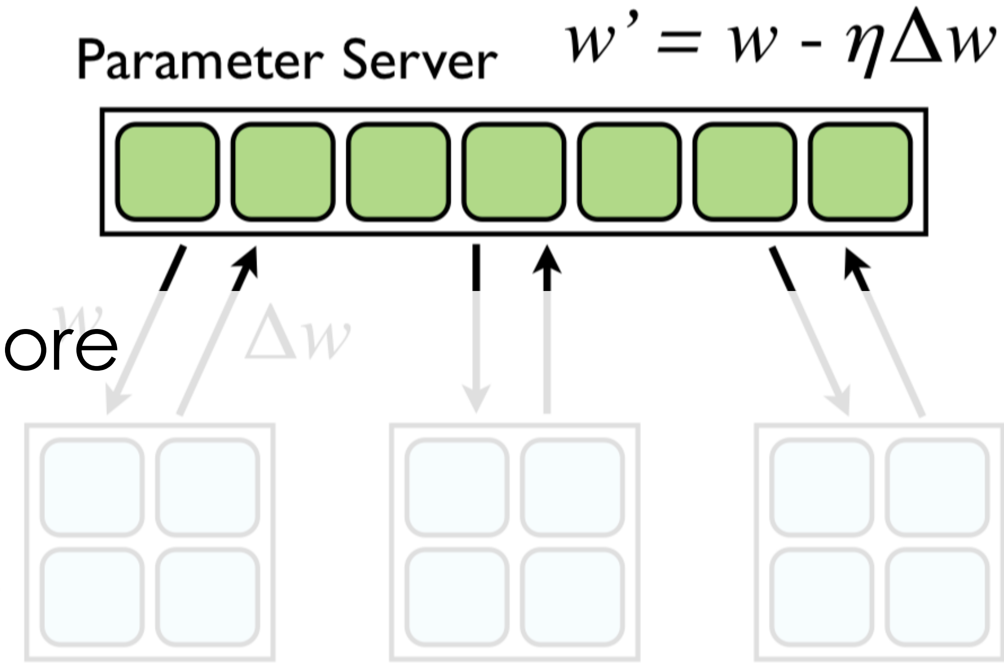
Downpour SGD

Claimed Innovations

- Parameter Server
- Combine model and data parallelism in an async. execution.
- Adagrad stabilization
- Warmstarting

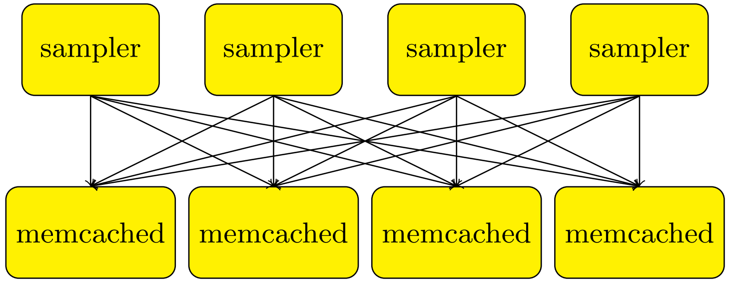


Parameter Servers

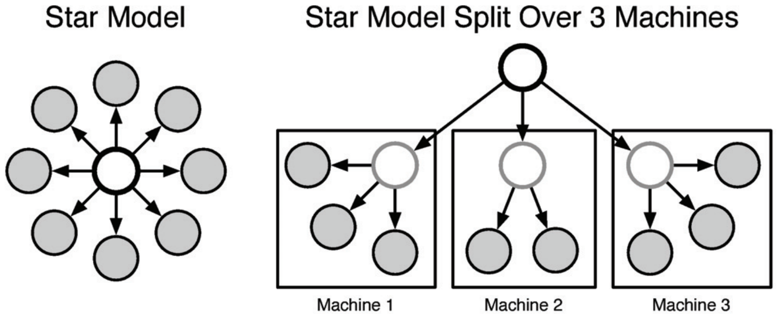


- Essentially a **sharded** key-value store
  - support for put, get, **add**
- Idea appears in earlier papers:

"An Architecture for Parallel Topic Models", Smola and Narayanamruthy. (VLDB'10)



"Scalable Inference in Latent Variable Models", Ahmed, Aly, Gonzalez, Narayanamruthy, and Smola. (WSDM'12)

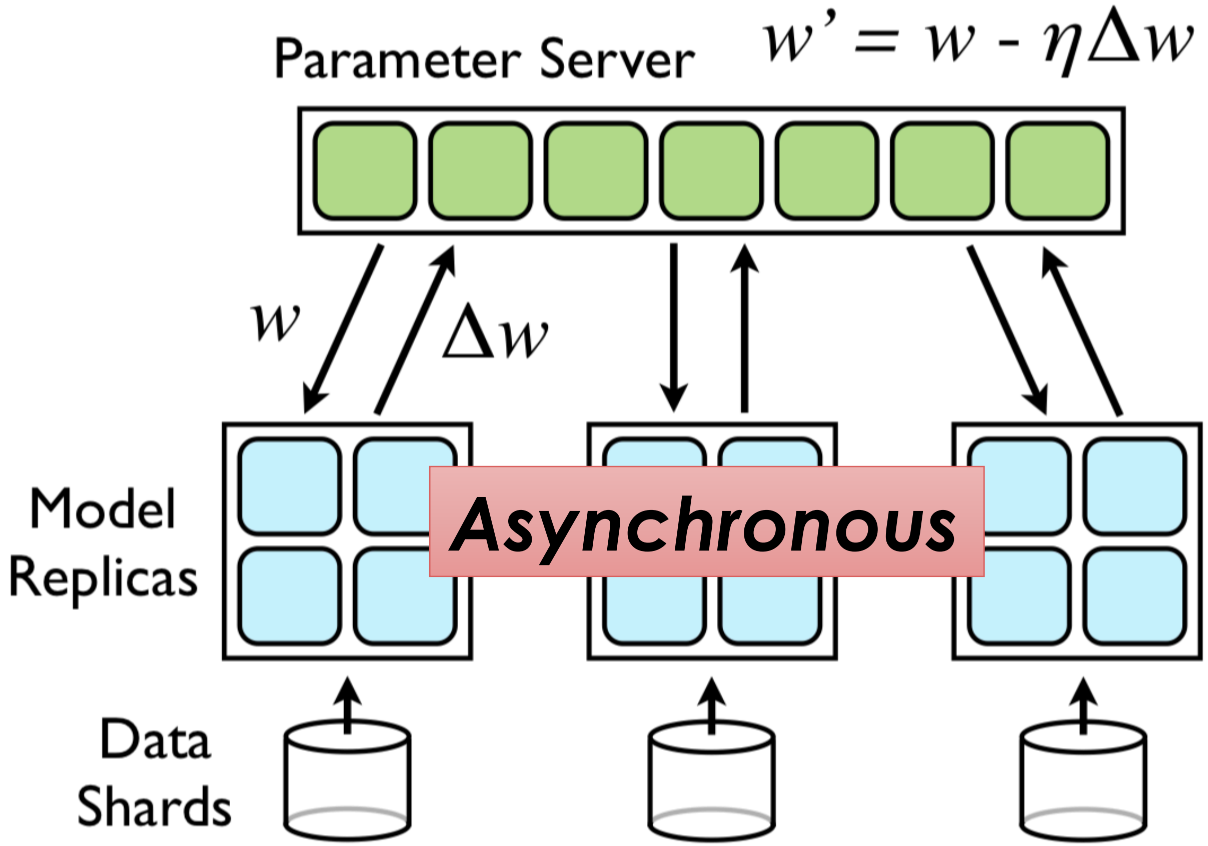


DistBelief was probably the first paper to call a sharded key-value store a Parameter Server.

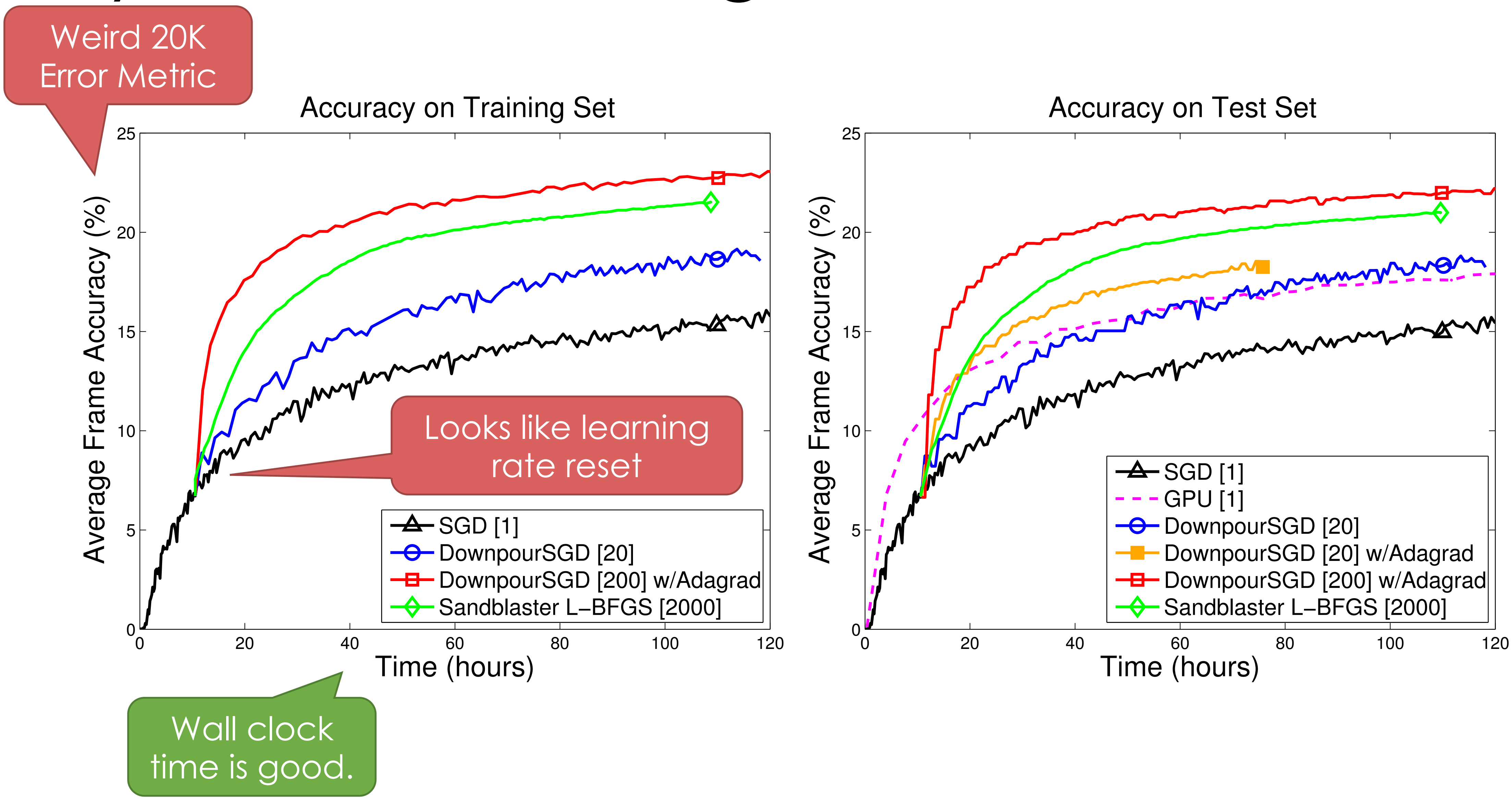
Downpour SGD

Claimed Innovations

- Parameter Server
- Combine model and data parallelism in an **async. execution.**
- Adagrad stabilization
- Warmstarting



Key Results: Training and Test Error



Why are they in the NY Times

- Trained a 1.7 billion parameter model (30x larger than state-of-the-art) (was it necessary?)
- Using 16,000 cores (efficiently?)
- Achieves 15.8 accuracy on ImageNet 20K (70% improvement over state of the art).
  - Non-standard benchmark
- Qualitatively interesting results

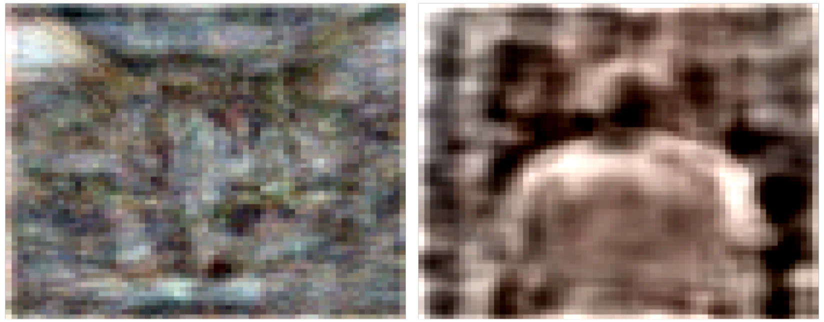


Figure 6. Visualization of the cat face neuron (left) and human body neuron (right).

Long-term Impact

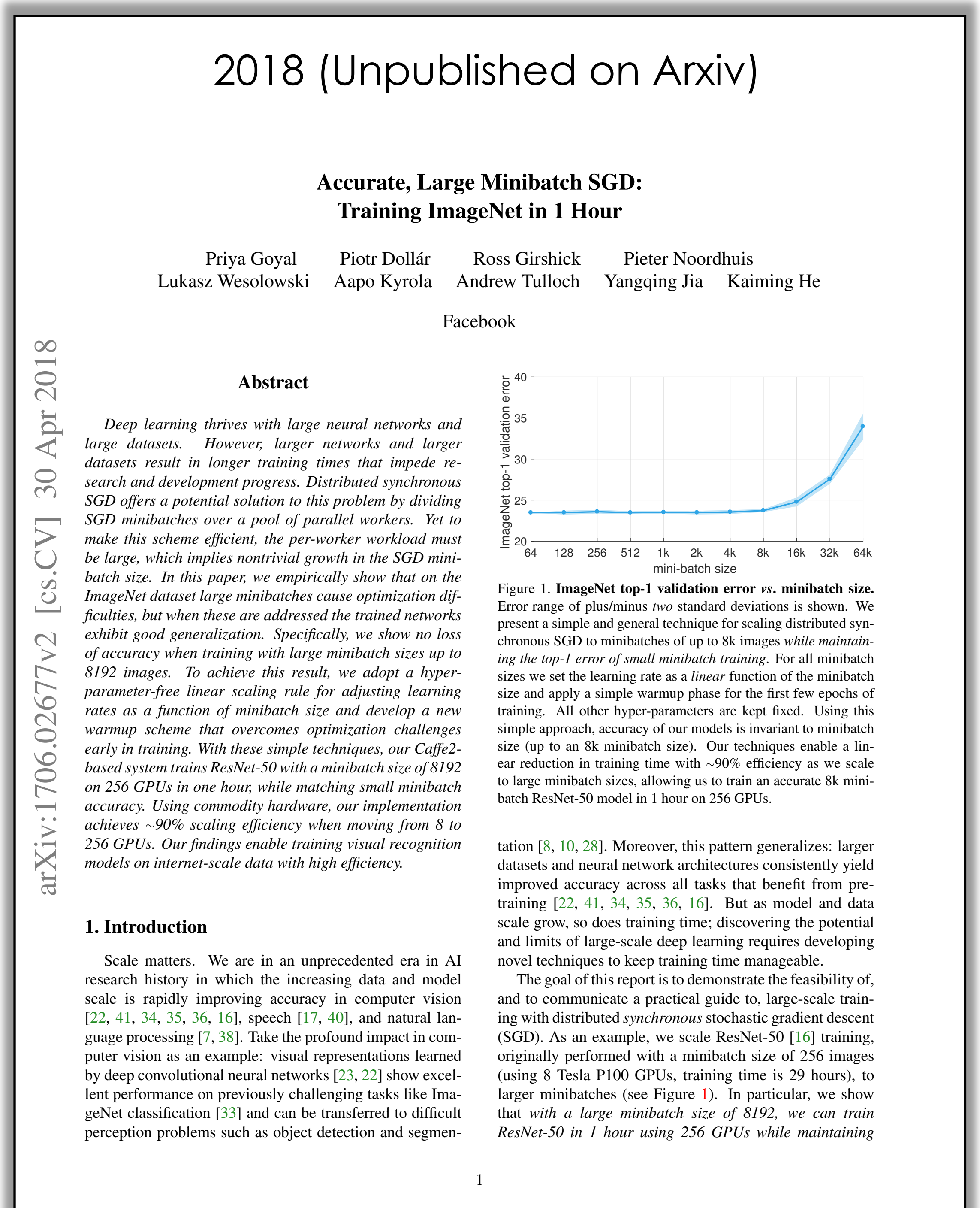
- The **parameter server** appears in many later machine learning systems
- Downpour (**Asynchronous**) SGD has been largely **replaced by synchronous systems** for supervised training
  - Asynchrony is still popular in RL research
    - Why?
- Model parallelism is still used for large language models
  - Predated this work
- The neural network architectures studied here have been largely replaced by convolutional networks

More recent large-scale training

- Generated a lot of press
  - Surpassed by

  Fast.ai: "*Now anyone can train ImageNet in 18 minutes for $40.*" blog post

- Popularized linear learning rate scaling



2018 (Unpublished on Arxiv)

Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour

Priya Goyal Piotr Dollár Ross Girshick Pieter Noordhuis
Lukasz Wesolowski Aapo Kyrola Andrew Tulloch Yangqing Jia Kaiming He

Facebook

arXiv:1706.02677v2 [cs.CV] 30 Apr 2018

Abstract

Deep learning thrives with large neural networks and large datasets. However, larger networks and larger datasets result in longer training times that impede research and development progress. Distributed synchronous SGD offers a potential solution to this problem by dividing SGD minibatches over a pool of parallel workers. Yet to make this scheme efficient, the per-worker workload must be large, which implies nontrivial growth in the SGD minibatch size. In this paper, we empirically show that on the ImageNet dataset large minibatches cause optimization difficulties, but when these are addressed the trained networks exhibit good generalization. Specifically, we show no loss of accuracy when training with large minibatch sizes up to 8192 images. To achieve this result, we adopt a hyper-parameter-free linear scaling rule for adjusting learning rates as a function of minibatch size and develop a new warmup scheme that overcomes optimization challenges early in training. With these simple techniques, our Caffe2-based system trains ResNet-50 with a minibatch size of 8192 on 256 GPUs in one hour, while matching small minibatch accuracy. Using commodity hardware, our implementation achieves ~90% scaling efficiency when moving from 8 to 256 GPUs. Our findings enable training visual recognition models on internet-scale data with high efficiency.

1. Introduction

Scale matters. We are in an unprecedented era in AI research history in which the increasing data and model scale is rapidly improving accuracy in computer vision [22, 41, 34, 35, 36, 16], speech [17, 40], and natural language processing [7, 38]. Take the profound impact in computer vision as an example: visual representations learned by deep convolutional neural networks [23, 22] show excellent performance on previously challenging tasks like ImageNet classification [33] and can be transferred to difficult perception problems such as object detection and segmentation [8, 10, 28]. Moreover, this pattern generalizes: larger datasets and neural network architectures consistently yield improved accuracy across all tasks that benefit from pre-training [22, 41, 34, 35, 36, 16]. But as model and data scale grow, so does training time; discovering the potential and limits of large-scale deep learning requires developing novel techniques to keep training time manageable.

The goal of this report is to demonstrate the feasibility of, and to communicate a practical guide to, large-scale training with distributed *synchronous* stochastic gradient descent (SGD). As an example, we scale ResNet-50 [16] training, originally performed with a minibatch size of 256 images (using 8 Tesla P100 GPUs, training time is 29 hours), to larger minibatches (see Figure 1). In particular, we show that *with a large minibatch size of 8192, we can train ResNet-50 in 1 hour using 256 GPUs while maintaining*

Figure 1. **ImageNet top-1 validation error *vs.* minibatch size.** Error range of plus/minus *two* standard deviations is shown. We present a simple and general technique for scaling distributed synchronous SGD to minibatches of up to 8k images *while maintaining the top-1 error of small minibatch training*. For all minibatch sizes we set the learning rate as a *linear* function of the minibatch size and apply a simple warmup phase for the first few epochs of training. All other hyper-parameters are kept fixed. Using this simple approach, accuracy of our models is invariant to minibatch size (up to an 8k minibatch size). Our techniques enable a linear reduction in training time with ~90% efficiency as we scale to large minibatch sizes, allowing us to train an accurate 8k minibatch ResNet-50 model in 1 hour on 256 GPUs.

1

Contrasting to the first paper

- **Synchronous** SGD
  - Much of the recent work has focused on synchronous setting
  - Easier to reason about
- Focus exclusively on data parallelism: **batch-size scaling**
- Focuses on the **generalization gap problem**

How do you distribute SGD?

Stochastic Gradient Descent

$\theta^{(0)} \leftarrow$ initial vector (random, zeros …)

For t from 0 to convergence:

$$\mathcal{B} \sim \text{ Random subset of indices}$$

$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \eta_t \left(\frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \nabla_\theta \mathbf{L}(y_i, f(x_i; \theta)) \Big|_{\theta = \theta^{(t)}} \right)$$

Data Parallelism

Slow? (~150ms)
Depending on size of B

Batch Size Scaling

- Increase the batch size by adding machines

$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \hat{\eta} \left(\frac{1}{k} \sum_{j=1}^{k} \frac{1}{|\mathcal{B}_j|} \sum_{i \in \mathcal{B}_j} \nabla_\theta \mathbf{L}(y_i, f(x_i; \theta)) \Big|_{\theta = \theta^{(t)}} \right)$$

- Each server processes a fixed batch size (e.g., n=32)
- As more servers are added (k) the effective overall batch size increases linearly
- Why do these additional servers help?

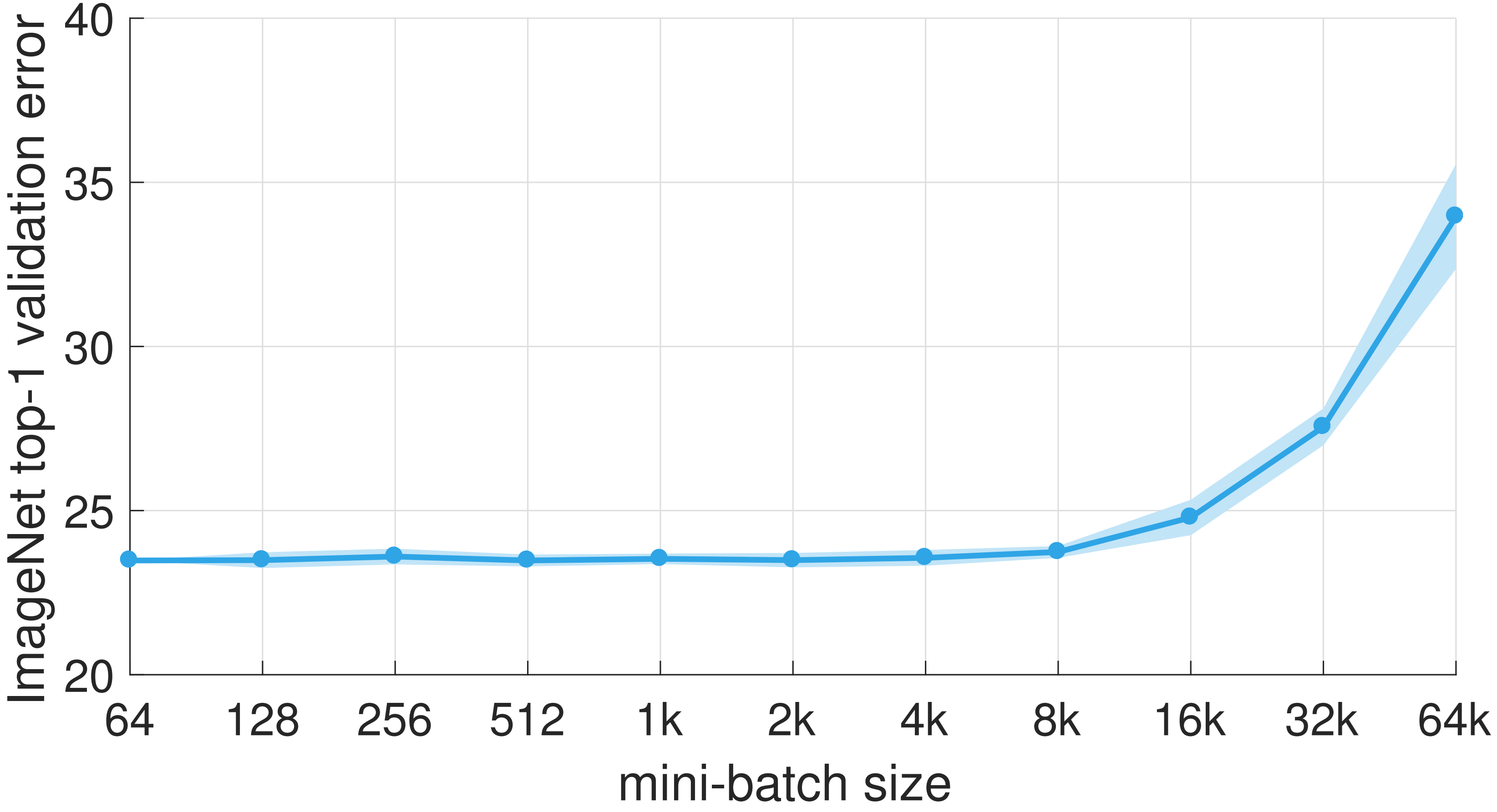
Bigger isn't Always Better

- Motivation for larger batch sizes
  - More opportunities for parallelism → but is it useful?
  - Recall (1/n variance reduction):

$$\frac{1}{n}\sum_{i=1}^{n}\nabla_\theta \mathbf{L}(y_i, f(x_i;\theta)) \approx \frac{1}{|\mathcal{B}|}\sum_{i\in\mathcal{B}}\nabla_\theta \mathbf{L}(y_i, f(x_i;\theta))$$

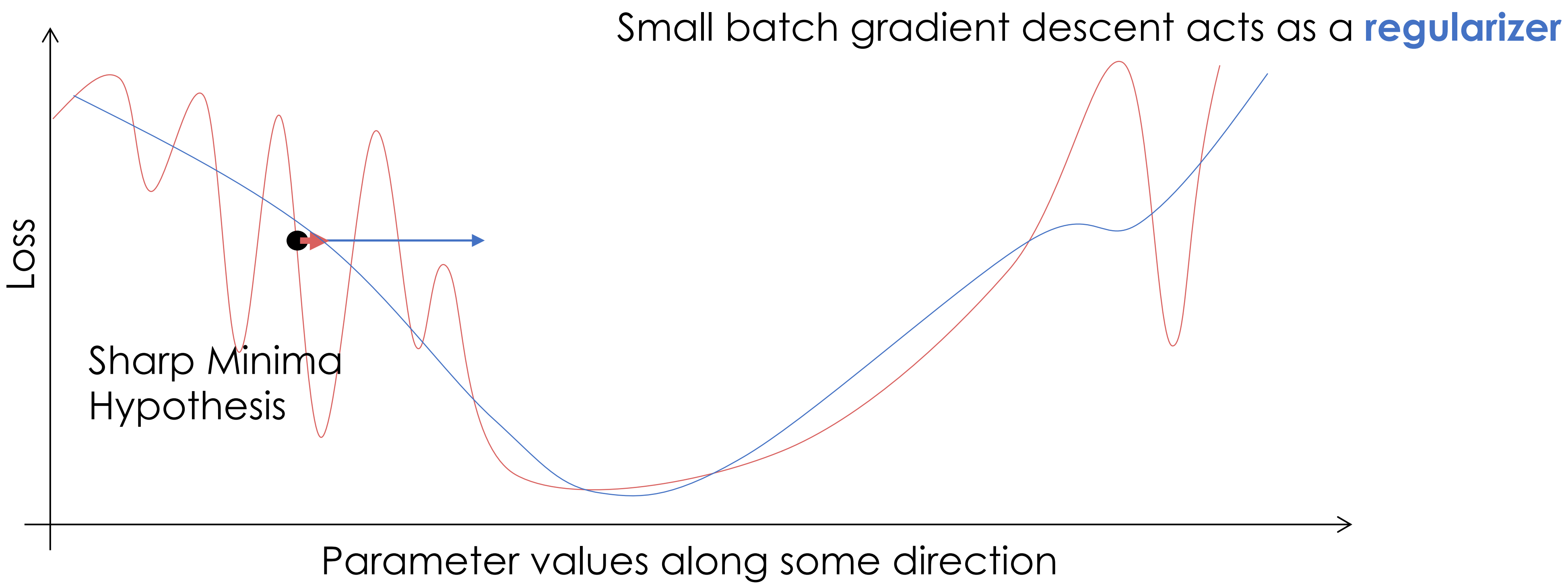
- Is a variance reduction helpful?
  - Only if it let's you take bigger steps (move faster)
  - Does it affect the final prediction accuracy?

Generalization Gap Problem



Larger batch sizes harm generalization performance.

Rough “Intuition”



Key problem: *Addressing the generalization gap for large batch sizes.*

Solution: Linear Scaling Rule

- Scale the learning rate linearly with the batch size

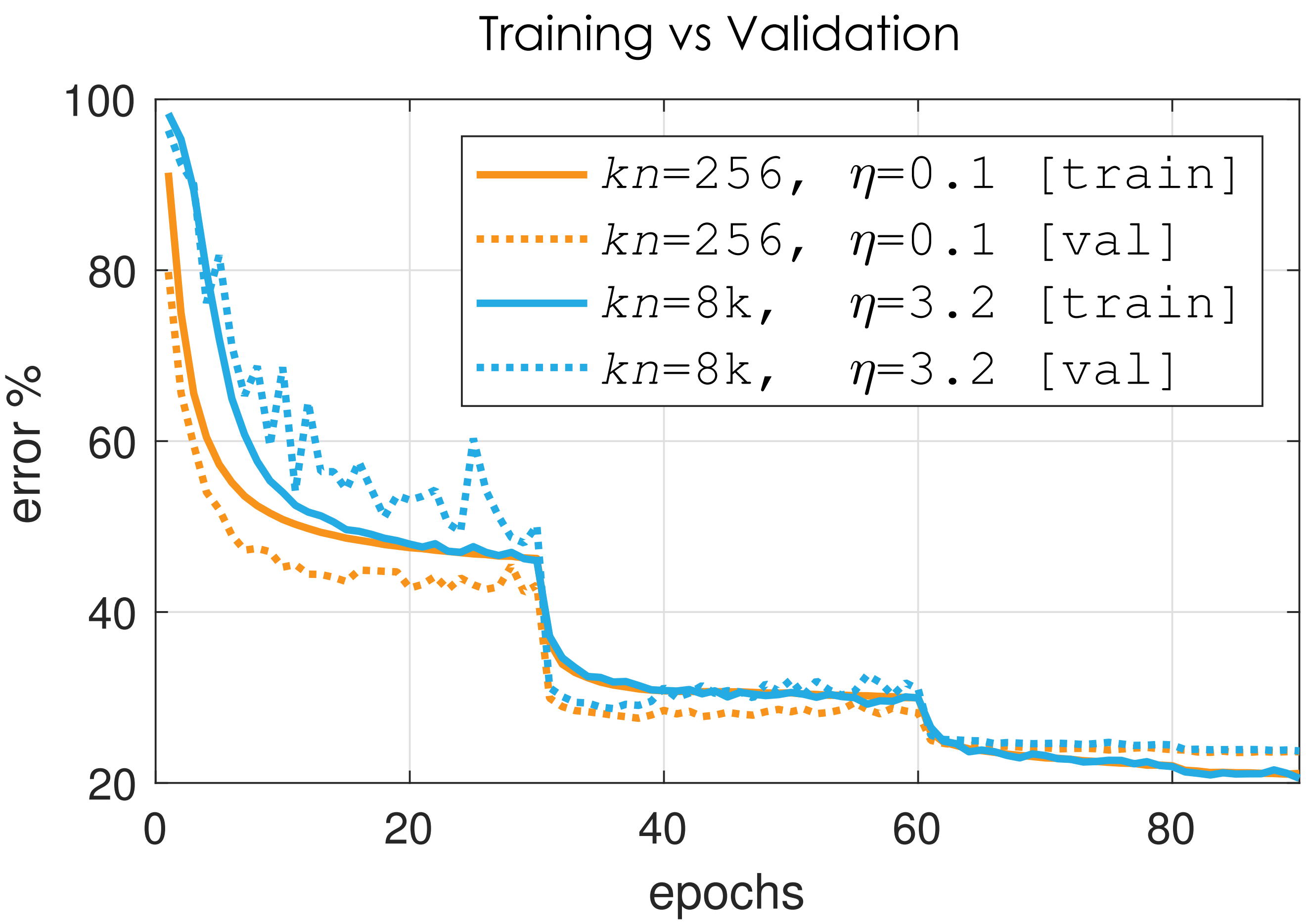
$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \underbrace{\hat{\eta}}_{=\eta k} \left(\frac{1}{k} \sum_{j=1}^{k} \frac{1}{|\mathcal{B}_j|} \sum_{i \in \mathcal{B}_j} \nabla_\theta \mathbf{L}(y_i, f(x_i;\theta)) \Big|_{\theta=\theta^{(t)}} \right)$$

- Addresses generalization performance by **taking larger steps** (also improves training convergence)
- **Sub-problem:** *Large learning rates can be destabilizing in the beginning.* Why?
  - **Gradual warmup solution:** increase learning rate scaling from constant to linear in first few epochs
  - Doesn't help for very large k…

Other Details

- **Independent Batch Norm**: Batch norm calculation applies only to local batch size (n).
- **All-Reduce:** Recursive halving and doubling algorithm
  - Used instead of popular ring reduction (fewer rounds)
- **Gloo** a library for efficient collective communications
- **Big Basin GPU Servers:** Designed for deep learning workloads
  - Analysis of communication requirements → latency bound
- **No discussion on straggler or fault-tolerance**
  - **Why?!**

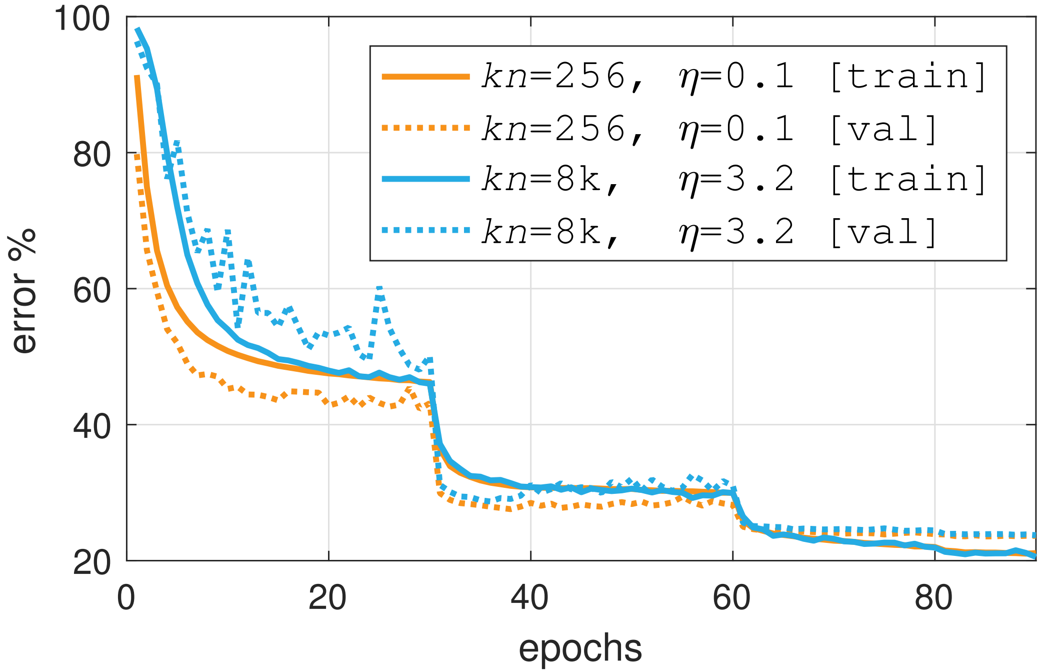
Key Results

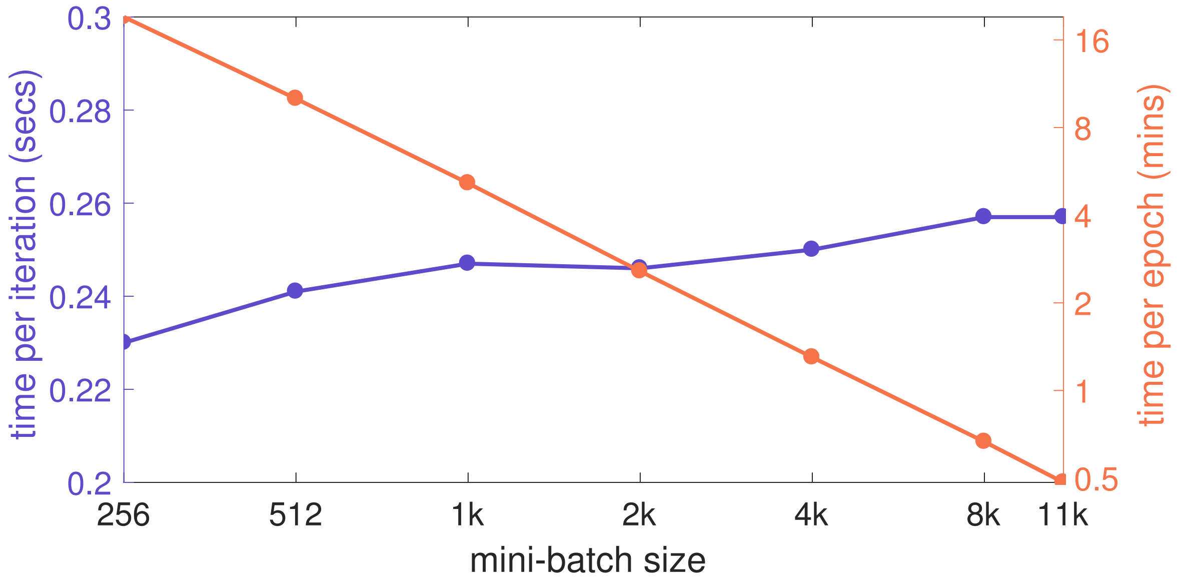


All curves closely match using the linear scaling rule.

Note learning rate schedule drops.

Key Results





$$\left(\frac{\text{“Learning”}}{\text{Epoch}} \right)$$

Machine Learning

$$\left(\frac{\text{Epoch}}{\text{Second}} \right)$$

System

Key Results

- Train ResNet-50 to state-of-the-art on 256 GPUs in 1 hour
  - 90% scaling efficiency

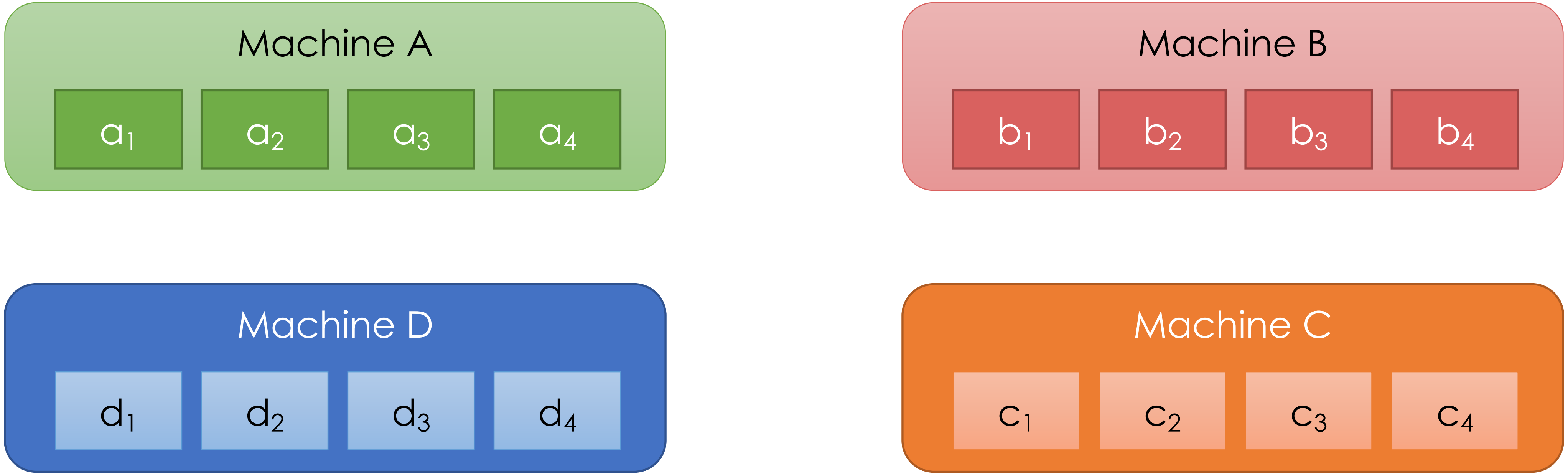
- Fairly careful study of the linear scaling rule
  - Observed limits to linear scaling do not depend on dataset size
  - Cannot scale parallelism with dataset size

All-Reduce

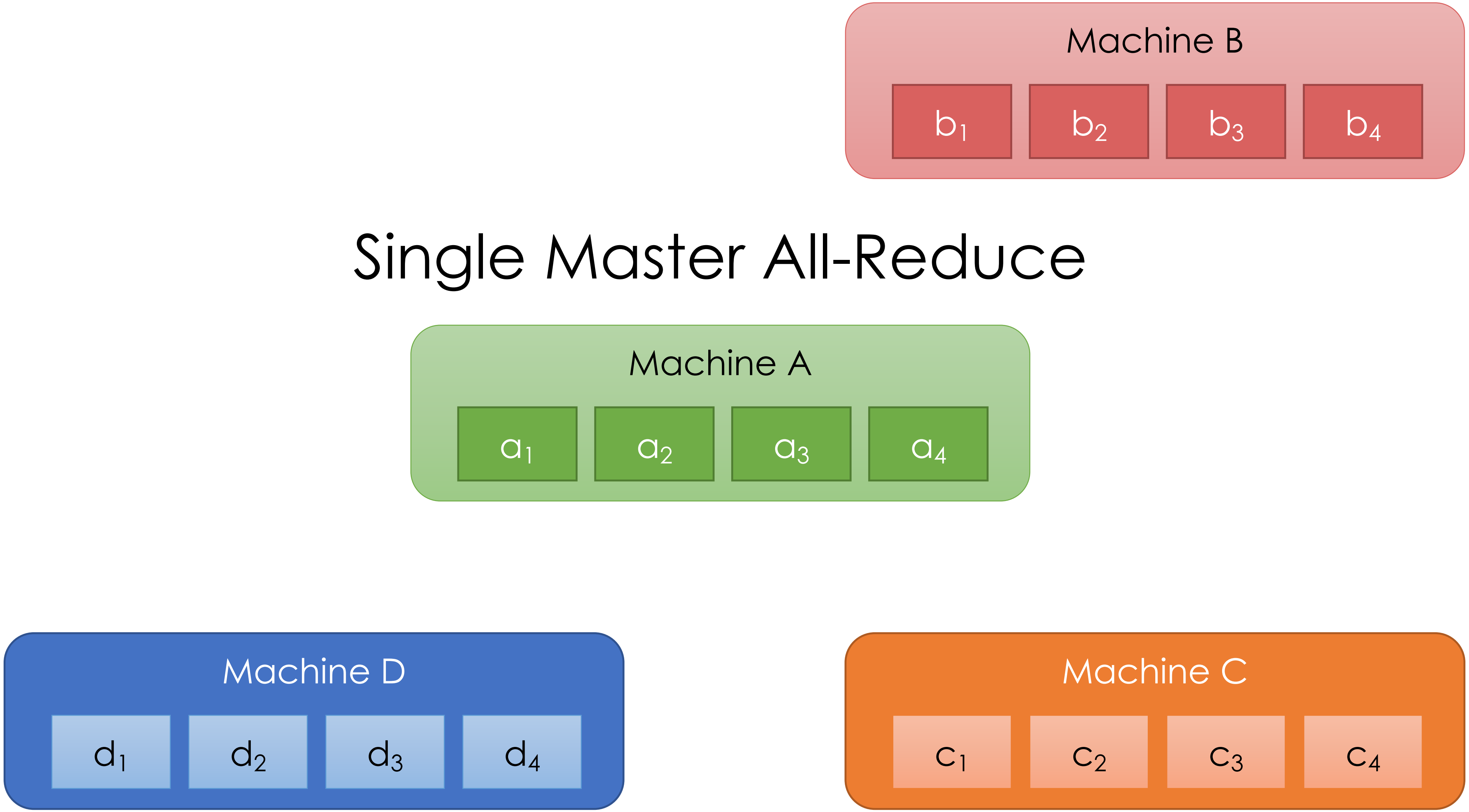
All Reduce

Mechanism to sum and distribute data across machines.

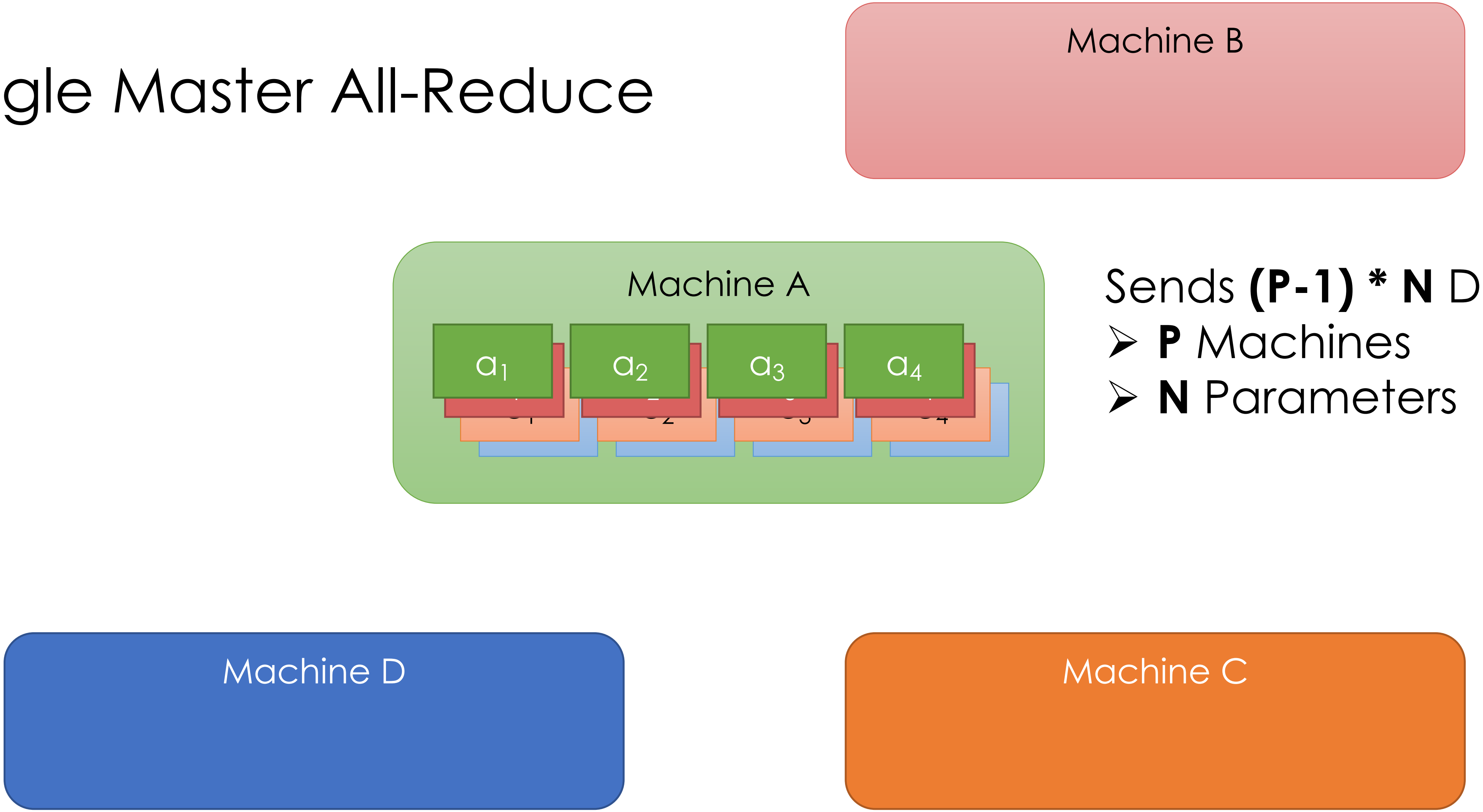
- ➢ Used to sum and distribute the gradient



Single Master All-Reduce



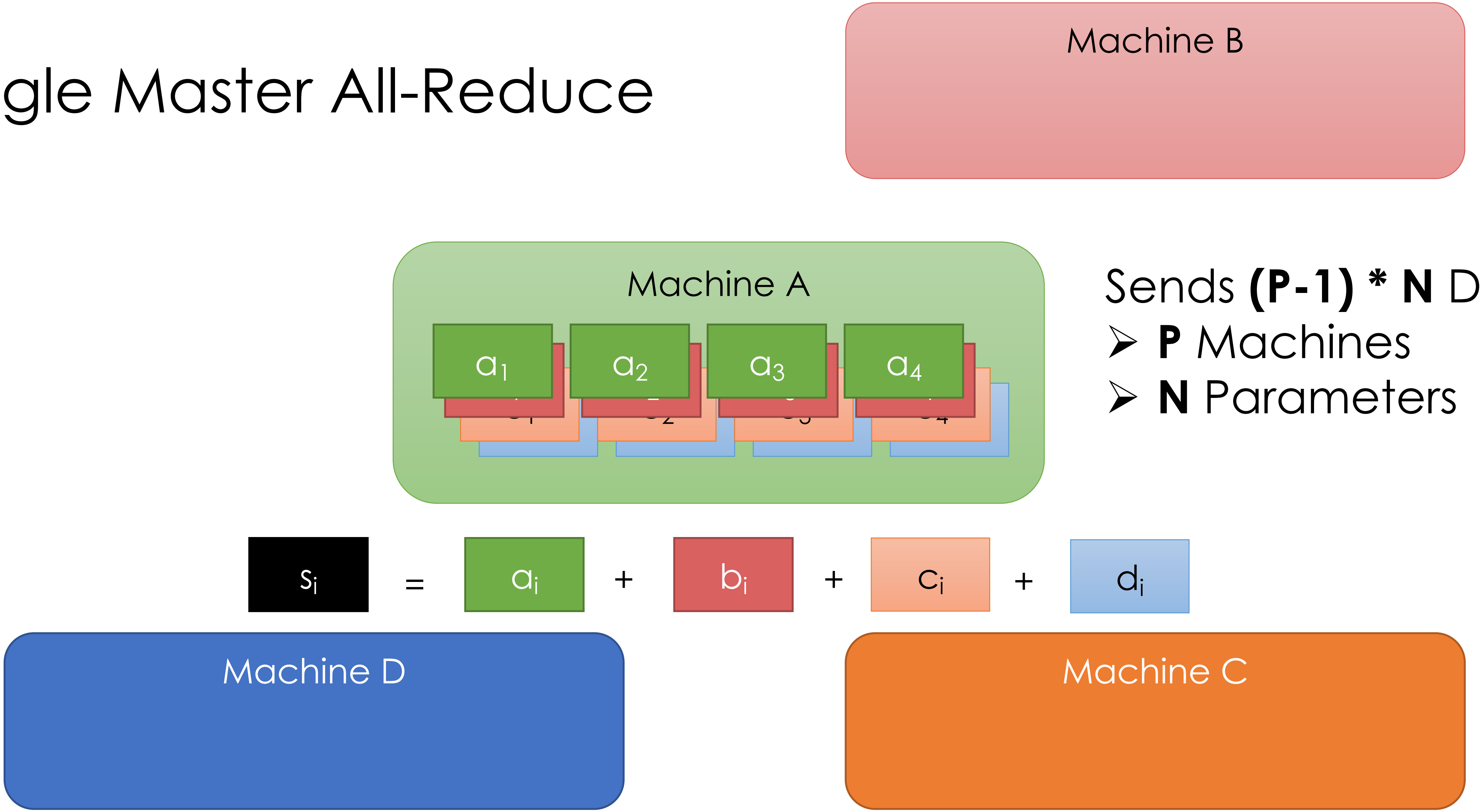
Single Master All-Reduce



Sends **(P-1) * N** Data

- **P** Machines
- **N** Parameters

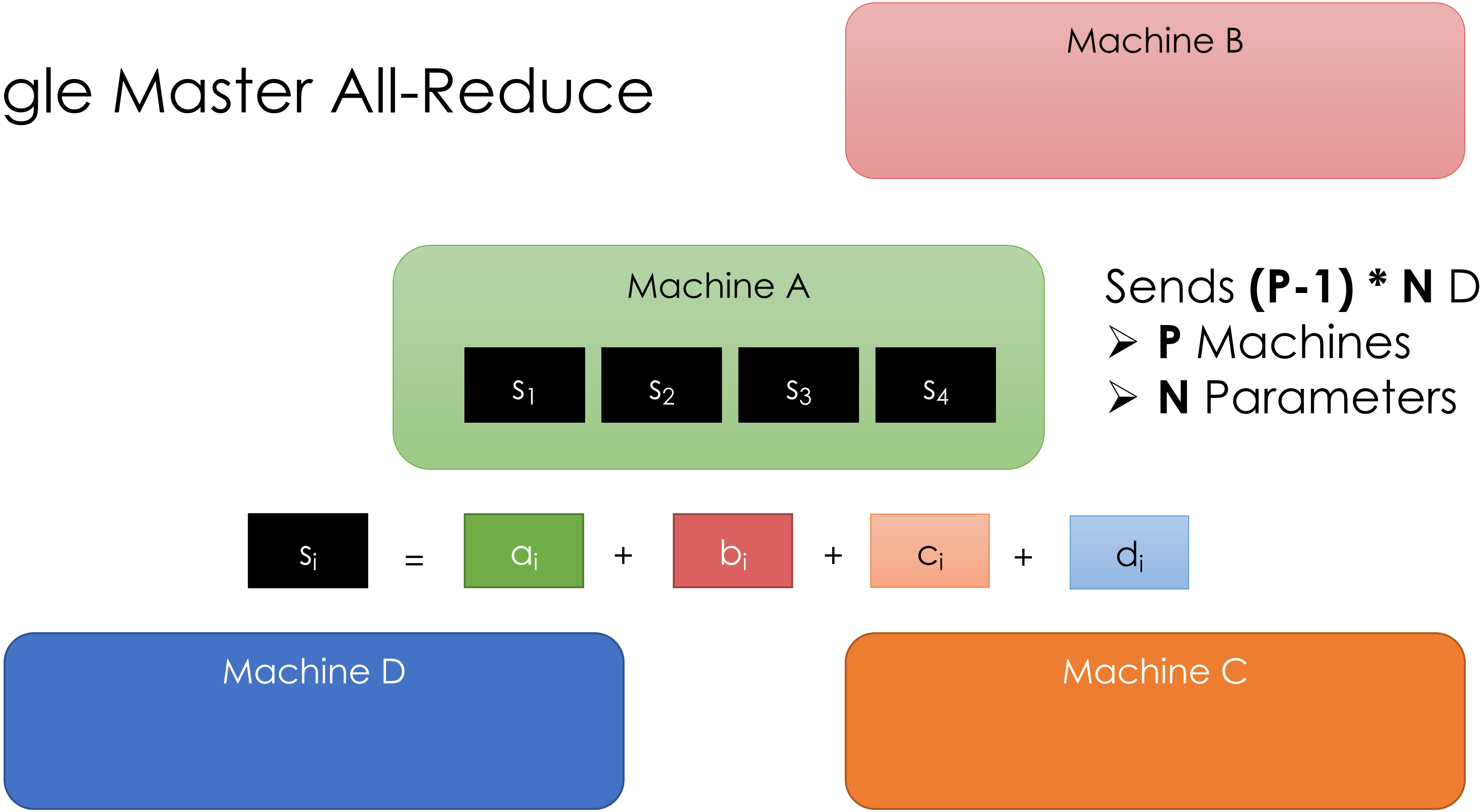
Single Master All-Reduce



Sends **(P-1) * N** Data

- **P** Machines
- **N** Parameters

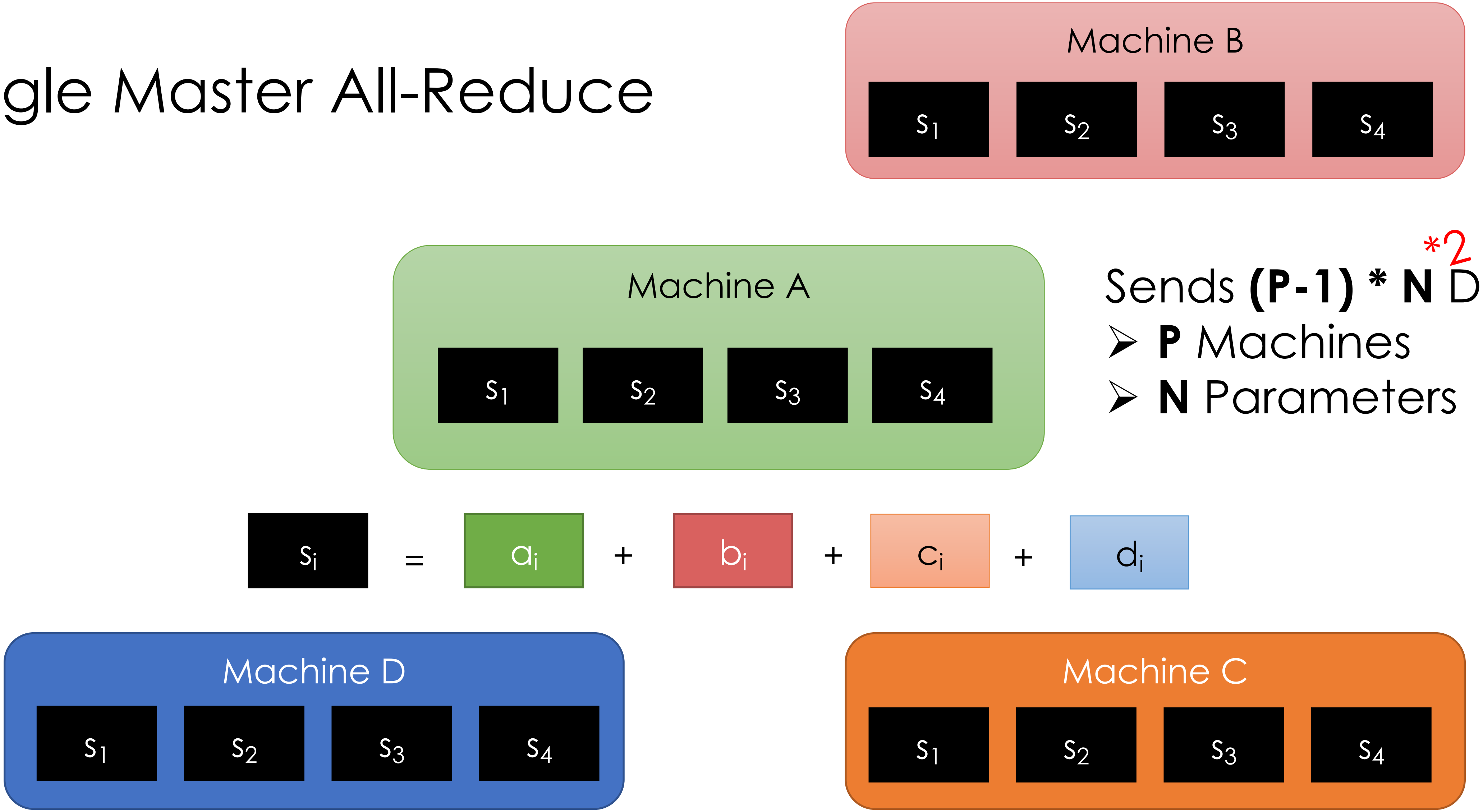
Single Master All-Reduce



Sends **(P-1) * N** Data

- **P** Machines
- **N** Parameters

Single Master All-Reduce

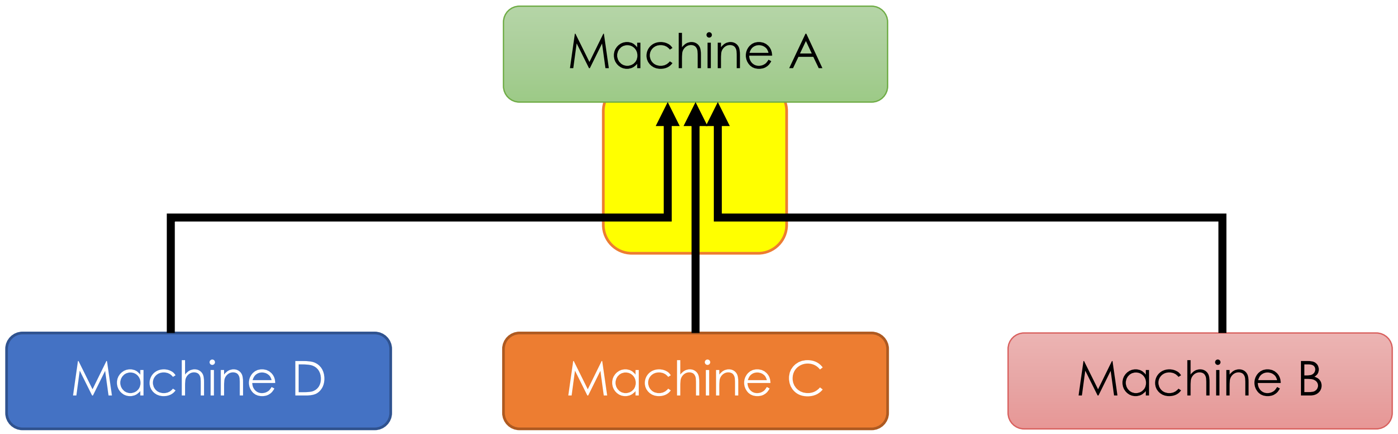


Sends **(P-1) * N** *2 Data

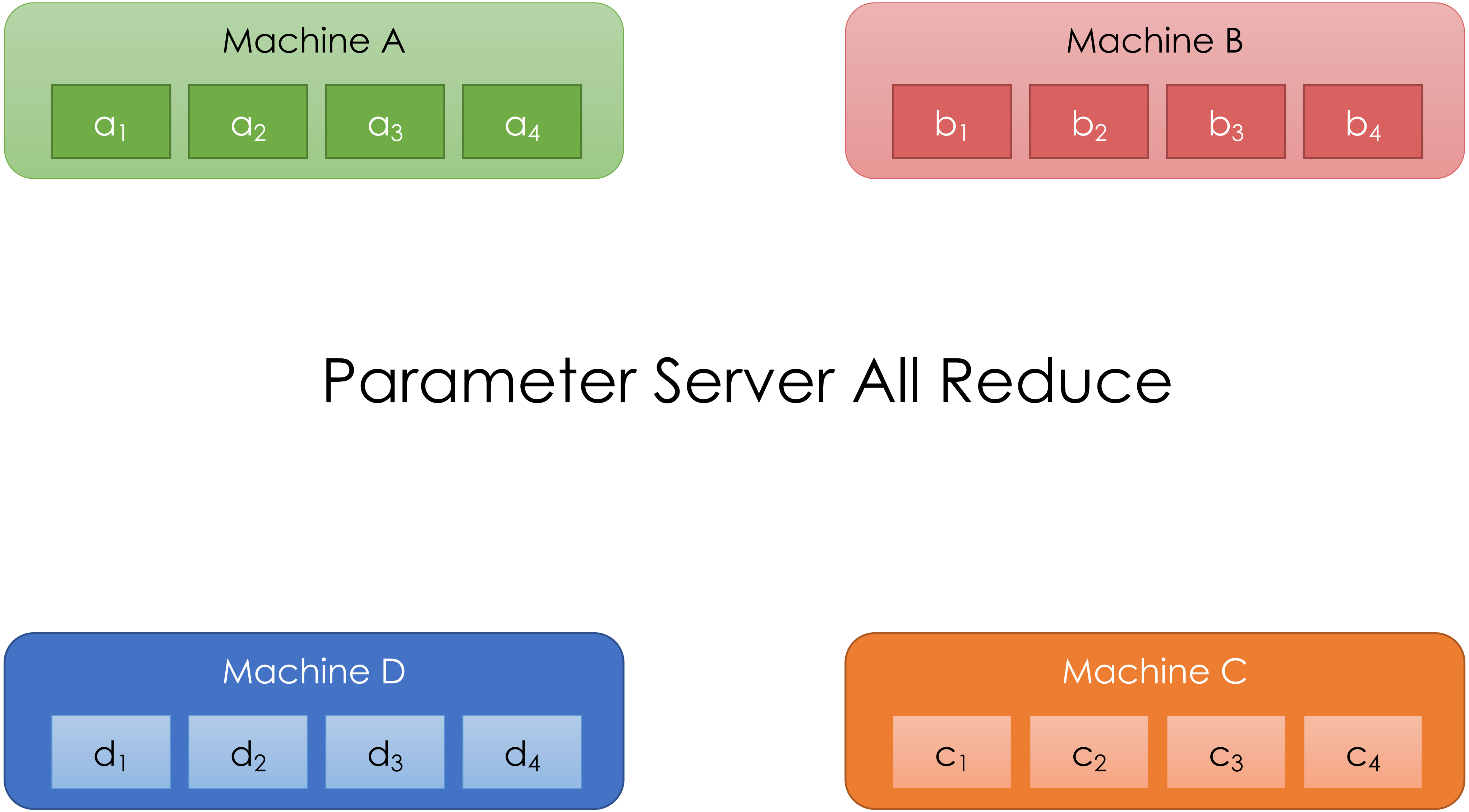
- **P** Machines
- **N** Parameters

Single Master All-Reduce

*2
Sends **(P-1) * N** Data
- ➢ **P** Machines
- ➢ **N** Parameters



Issues?
- ➢ High **fan-in** on Machine A
- ➢ **(P-1) * N Bandwidth** for Machine A


Machine A
a_1 a_2 a_3 a_4
Machine B
b_1 b_2 b_3 b_4
Parameter Server All Reduce
Machine D
d_1 d_2 d_3 d_4
Machine C
c_1 c_2 c_3 c_4

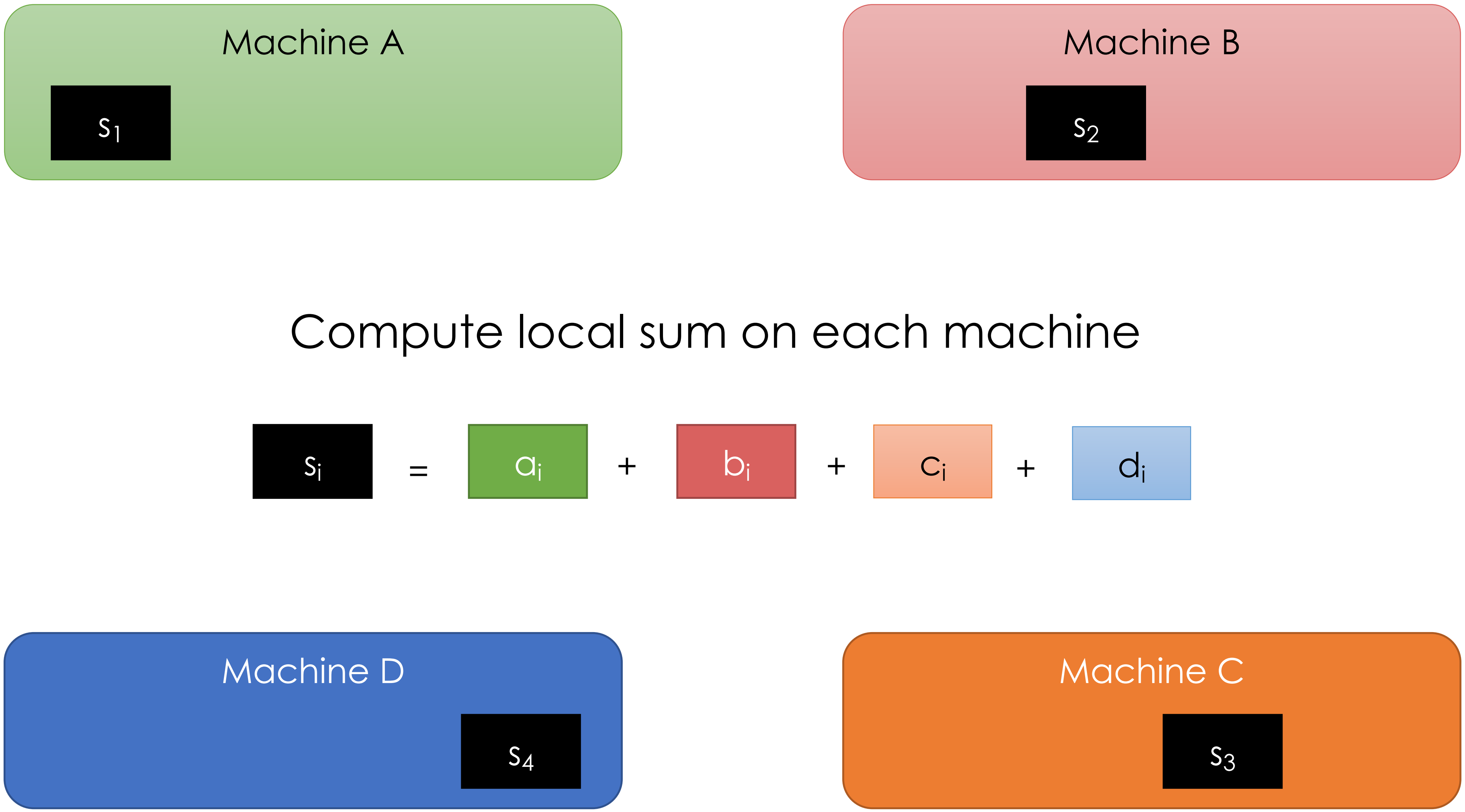
Send each entry to parameter server for that entry.

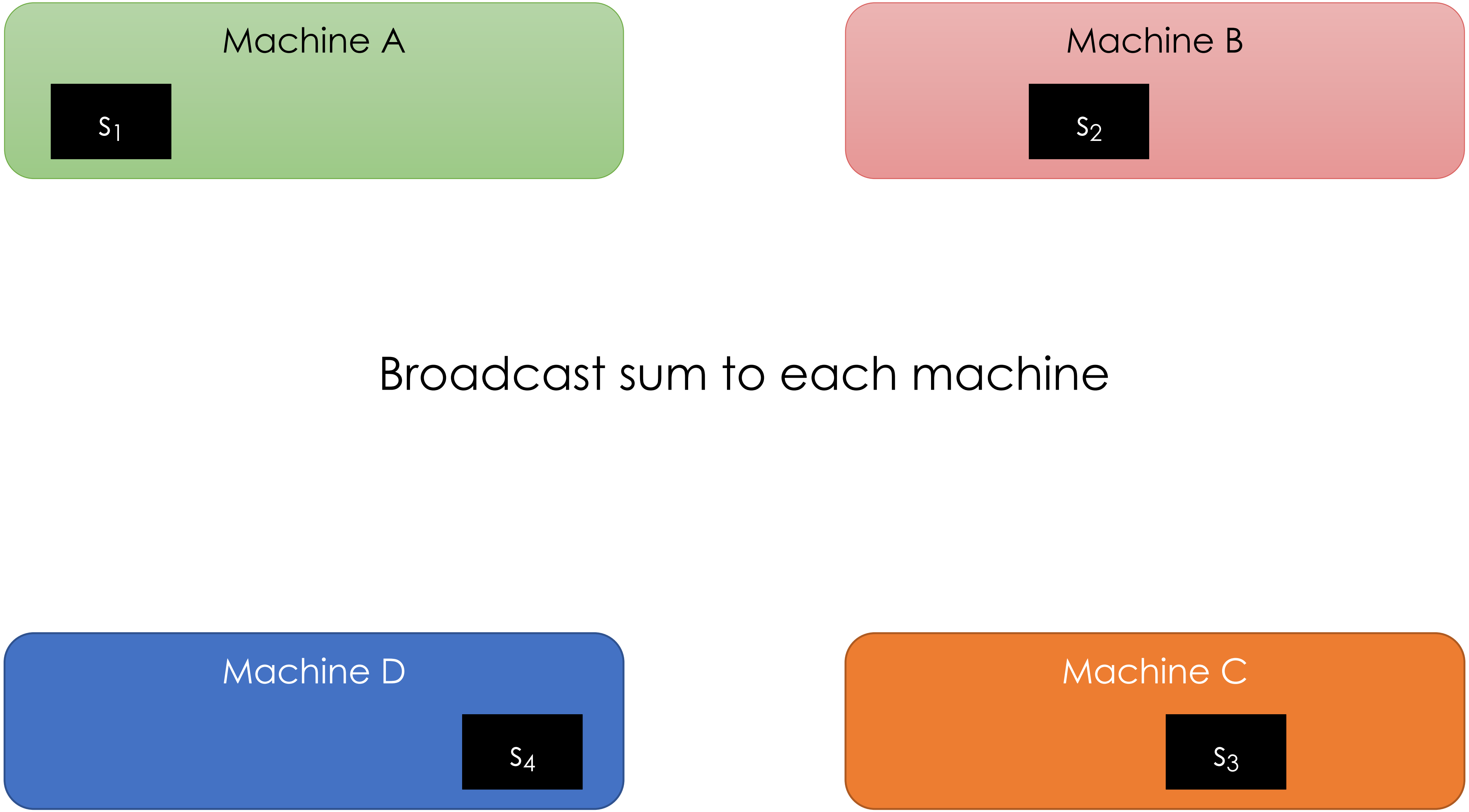
- Key 1 → A
- Key 2 → B
- Key 3 → C
- Key 4 → D

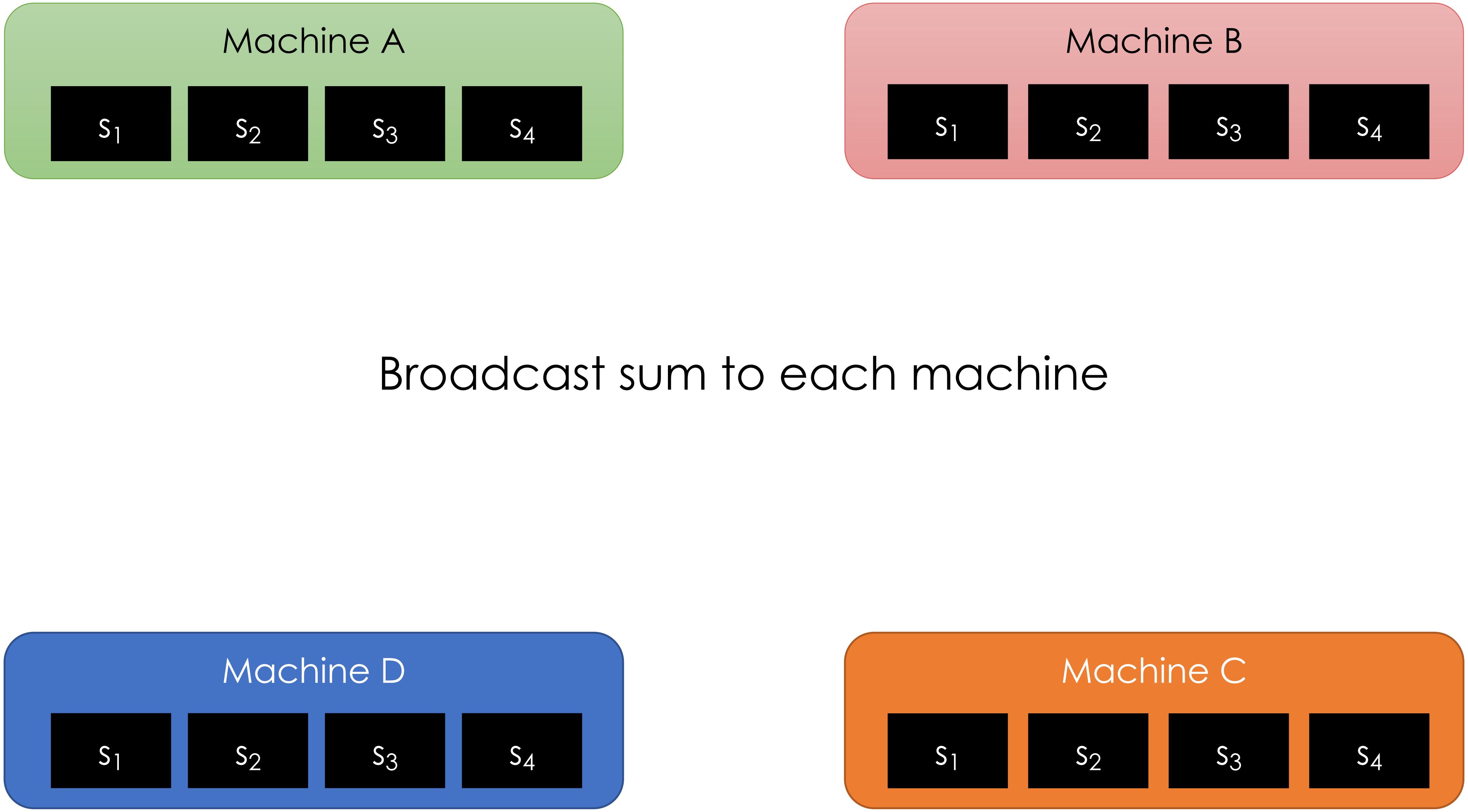
Each machine sends N/P data to all other machines.

P * (P-1) * N/P = (P-1) * N

- **P** Machines
- **N** Parameters


Machine A
s_1
Machine B
s_2
Compute local sum on each machine
$s_i = a_i + b_i + c_i + d_i$
Machine D
s_4
Machine C
s_3

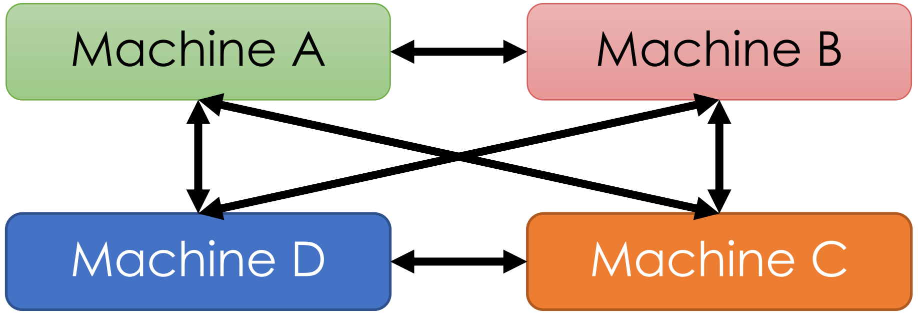

Machine A
s_1
Machine B
s_2
Broadcast sum to each machine
Machine D
s_4
Machine C
s_3



Machine A
S_1 S_2 S_3 S_4
Machine B
S_1 S_2 S_3 S_4
Broadcast sum to each machine
Machine D
S_1 S_2 S_3 S_4
Machine C
S_1 S_2 S_3 S_4

Parameter Server All-Reduce

- Same amount of data transmitted as before

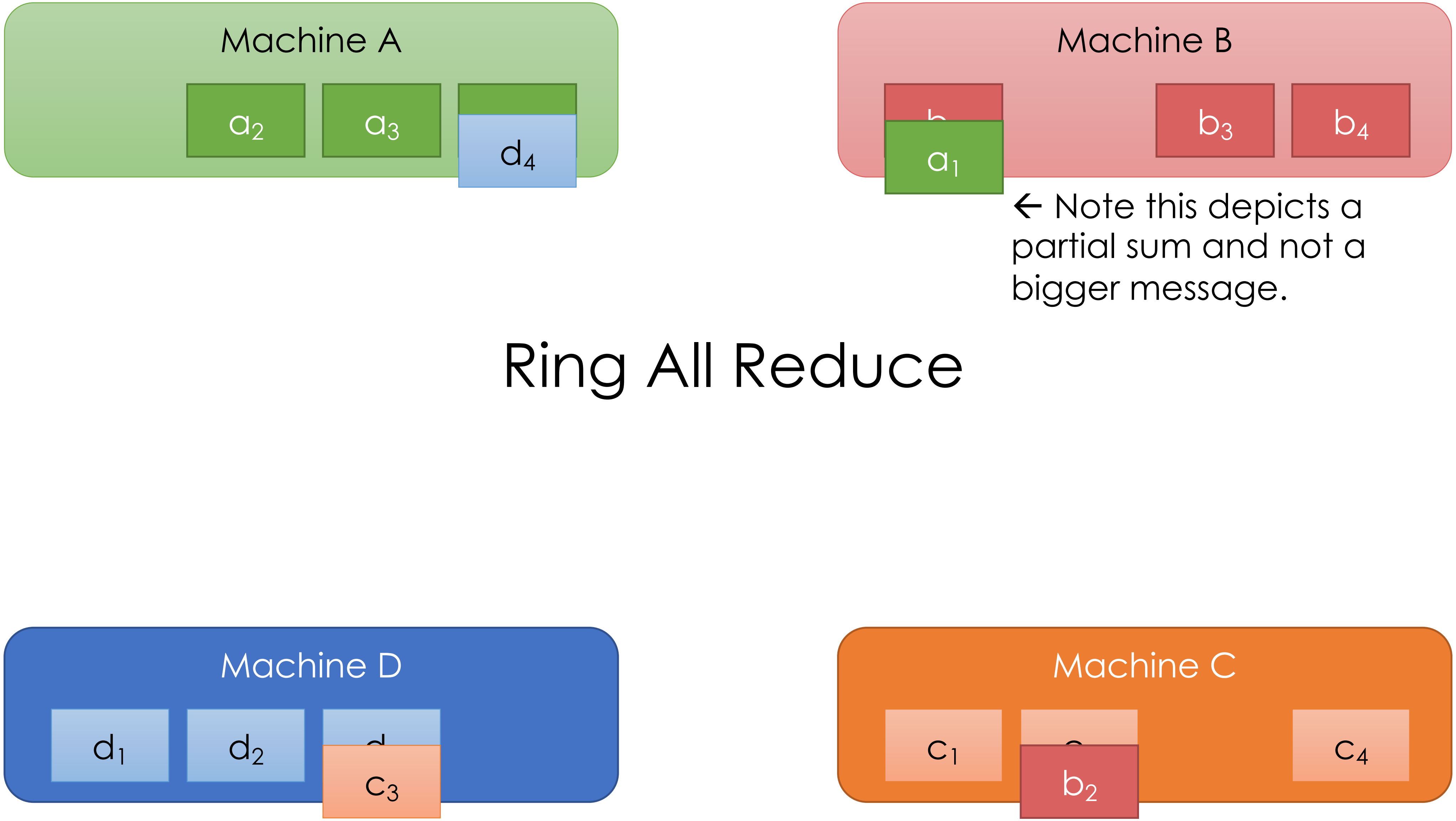


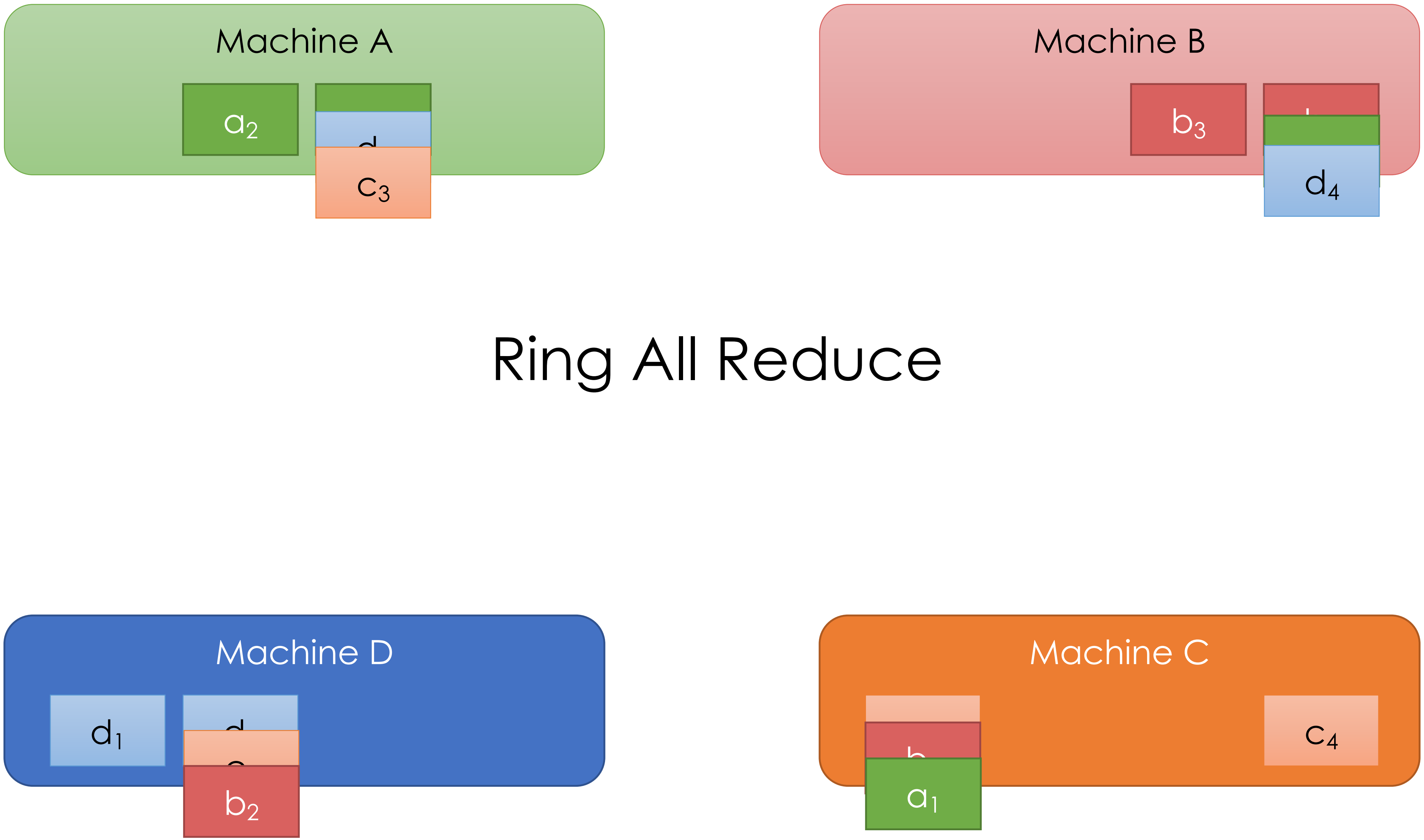
- Same **high fan-in** (P-1)
- **Reduced** Inbound Bandwidth = (P-1)N/P
  - Previously (P-1)*N

Ring All Reduce

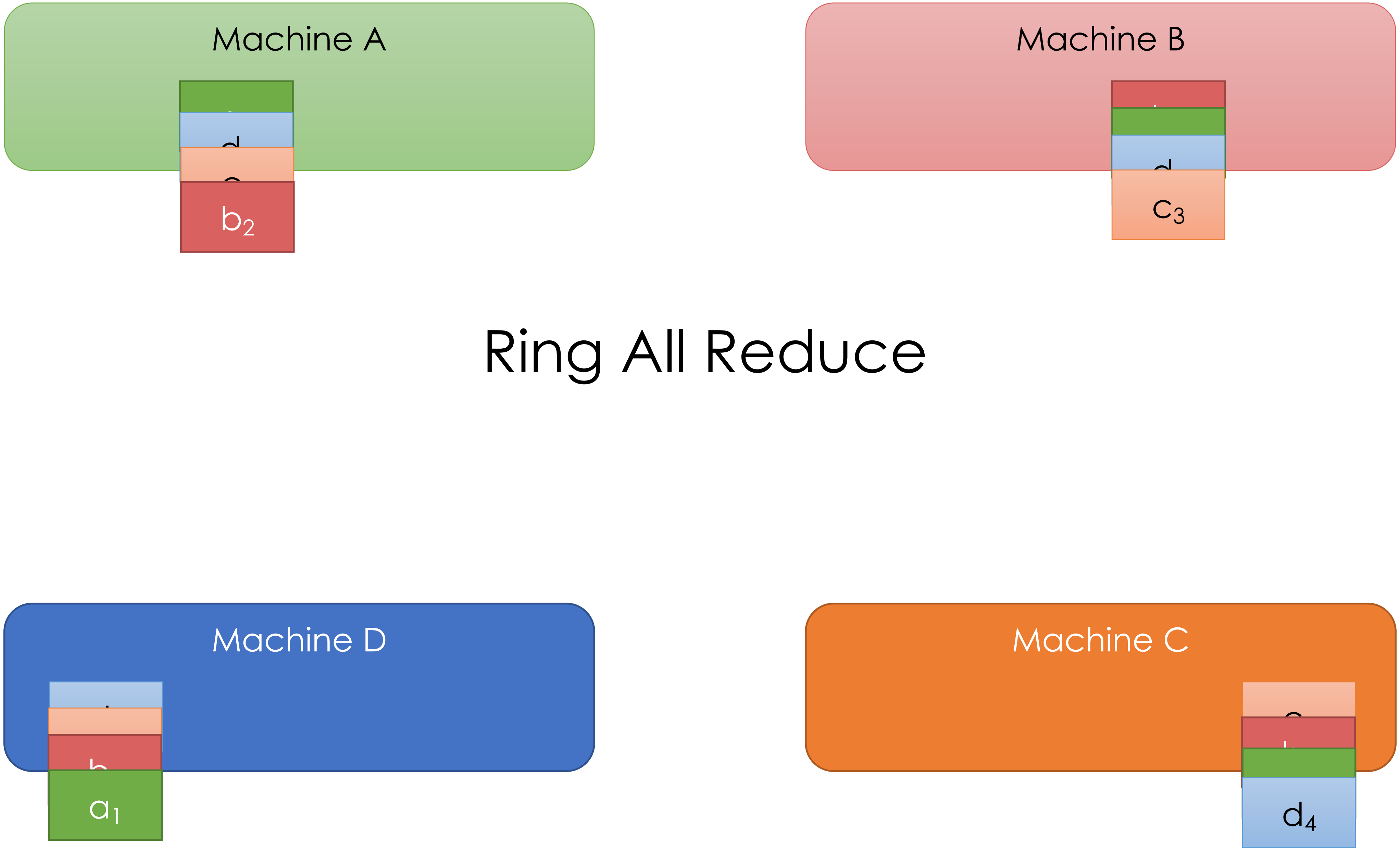
Send messages in a ring using to reduce fan-in.

Ring All Reduce




Machine A
a2
c3
Machine B
b3
d4
Ring All Reduce
Machine D
d1
b2
Machine C
c4
a1

Ring All Reduce



Machine A

S_2

Machine B

S_3

Ring All Reduce

Each machine sends N/P data to next machine each of (p-1) rounds:

(P-1) * P * N/P = (P-1) * N

- **Bandwidth** per round:
  - **P (N/P) = N** (doesn't depend on P)
- **Fan-in Per Round:**
  - 1 (doesn't depend on P)

Machine D

S_1

Machine C

S_4

Machine A

s_2

Machine B

s_3

Ring All Reduce

Broadcast stage repeats process sending messages forwarding sums (same communication costs).

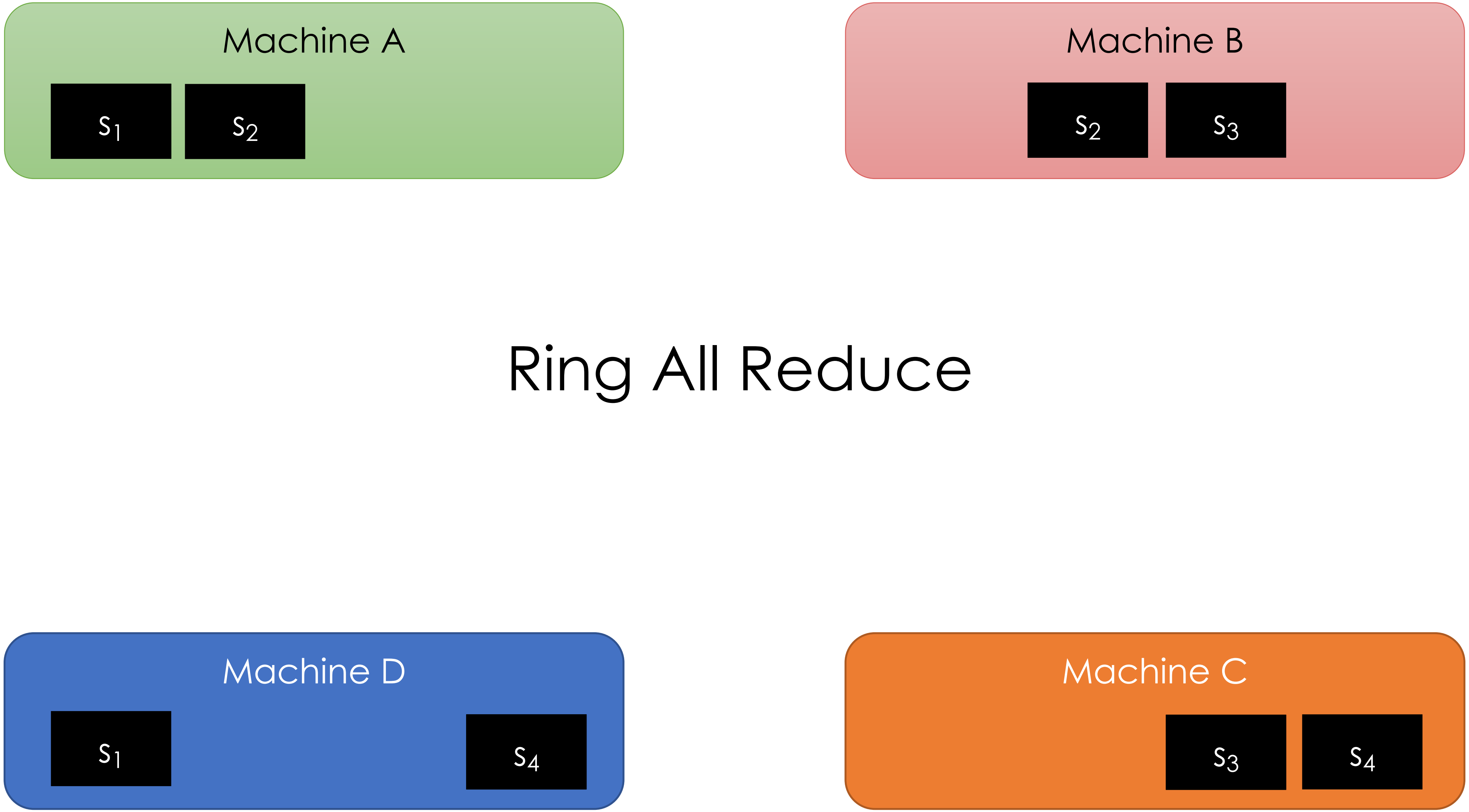
Machine D

s_1

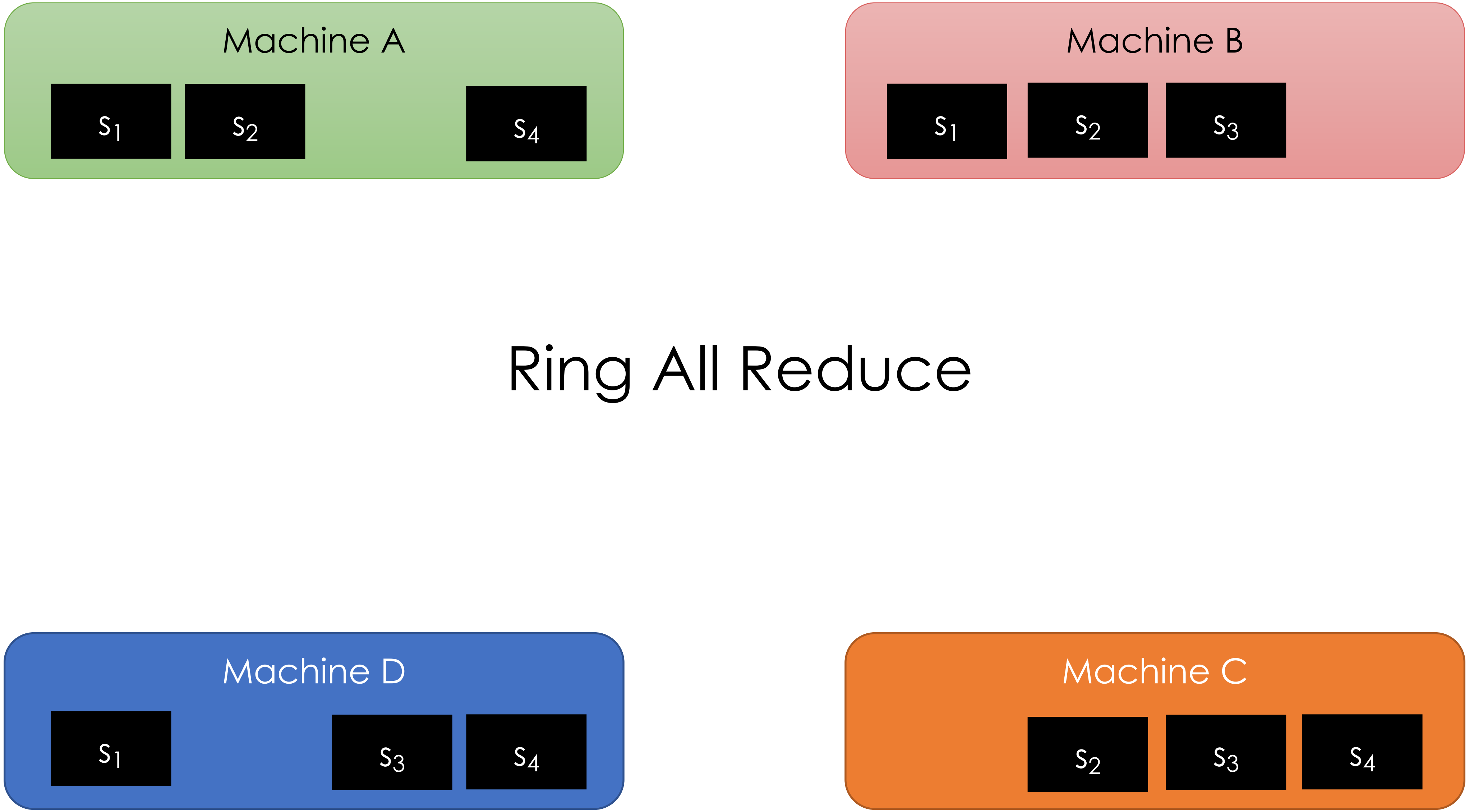
Machine C

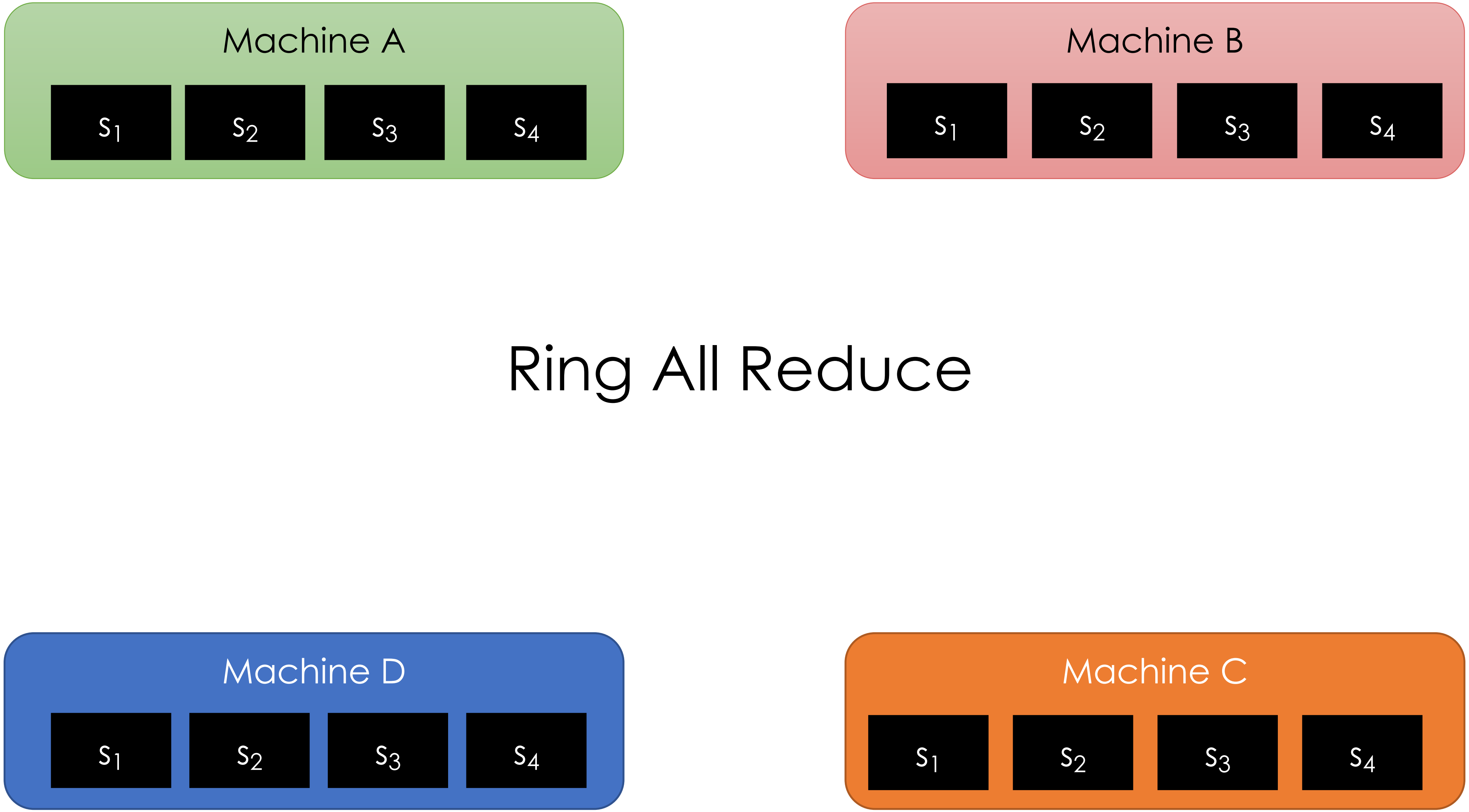
s_4

Ring All Reduce

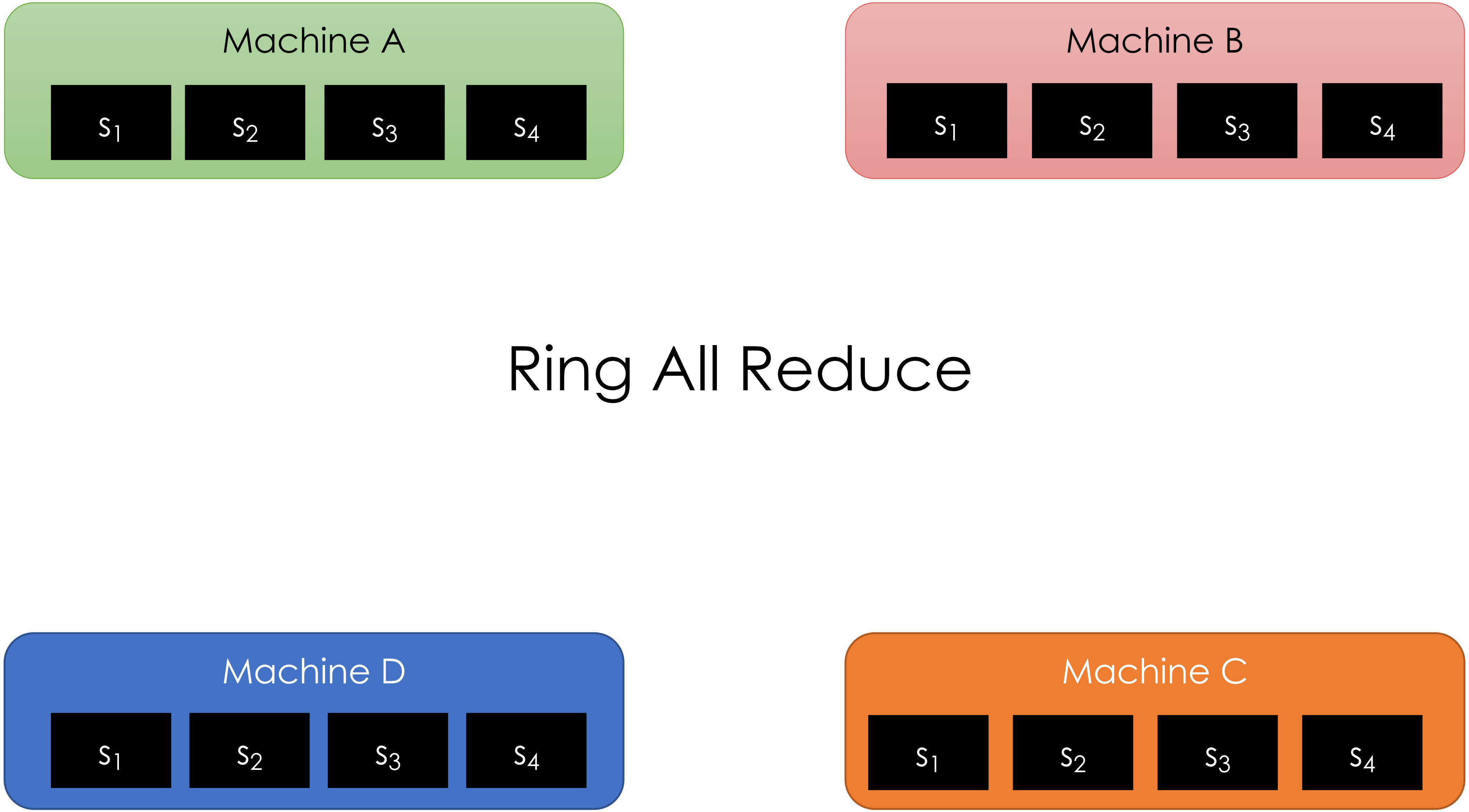


Ring All Reduce



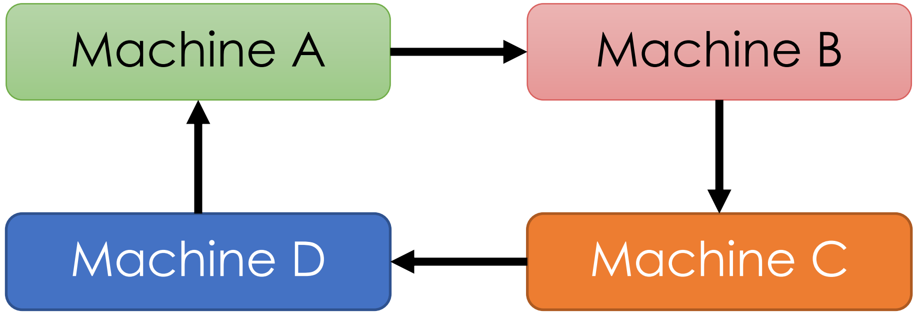

Machine A
S_1 S_2 S_3 S_4
Machine B
S_1 S_2 S_3 S_4
Ring All Reduce
Machine D
S_1 S_2 S_3 S_4
Machine C
S_1 S_2 S_3 S_4

Ring All Reduce



Ring All-Reduce

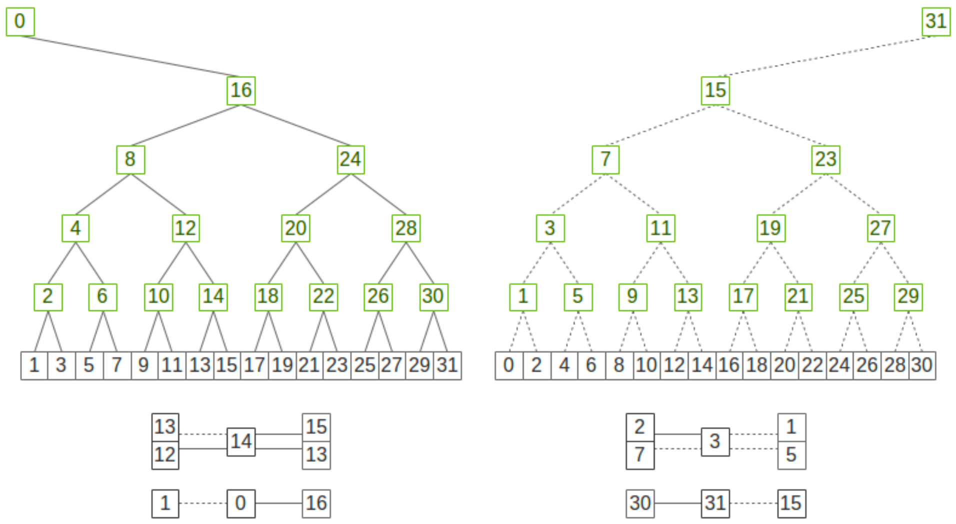
- Simplified communication topology with low fan-in

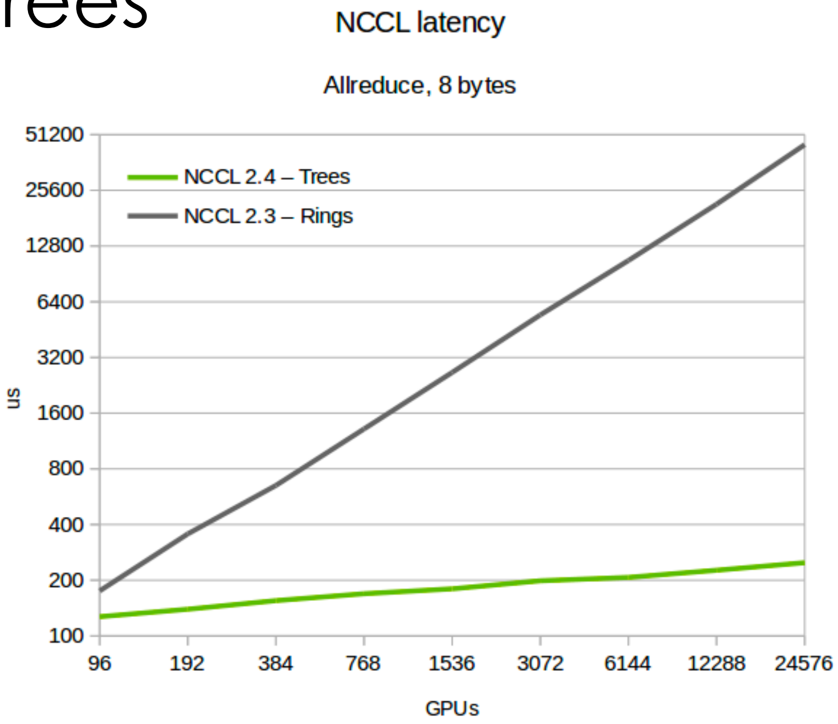


- Overall communication
  - Same total communication: **2*(P-1)*N**
  - **Bandwidth** per round (N) doesn't depend on P
  - **Fan-in** is constant (doesn't depend on P)
- **Issue:** Number of communication rounds (P-1)

Double Binary Tree All-Reduce

- ➢ Two overlaid binary reduction trees





- ➢ Double the fan-in → Log(p) rounds of communication
  - ➢ Currently used on Summit super-computer and latest NCCL

https://devblogs.nvidia.com/massively-scale-deep-learning-training-nccl-2-4/

Review:

Dimensions of Parallelism

Data Parallelism

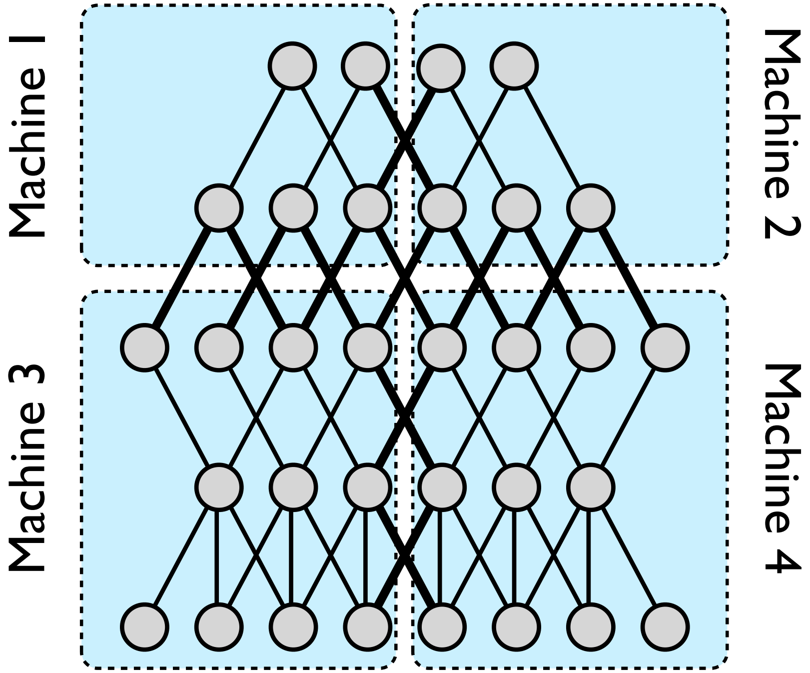
Parallelizing mini-batch gradient calculation with model replicated to all machines.

- Synchronous Execution (Most Common)
  - **Strengths:** deterministic, parallelism does not effect result
  - **Weaknesses:** need large batch sizes, frequent blocking comm., learning rate scaling, doesn't work with batch normalization
- Asynchronous Execution (Popular in Research)
  - **Strengths:** eliminate blocking and use background comm., batches don't need to span machines
  - **Weaknesses:** affects convergence (stability)
- Issues:
  - Model and activations must fit in each machine

Model Parallelism

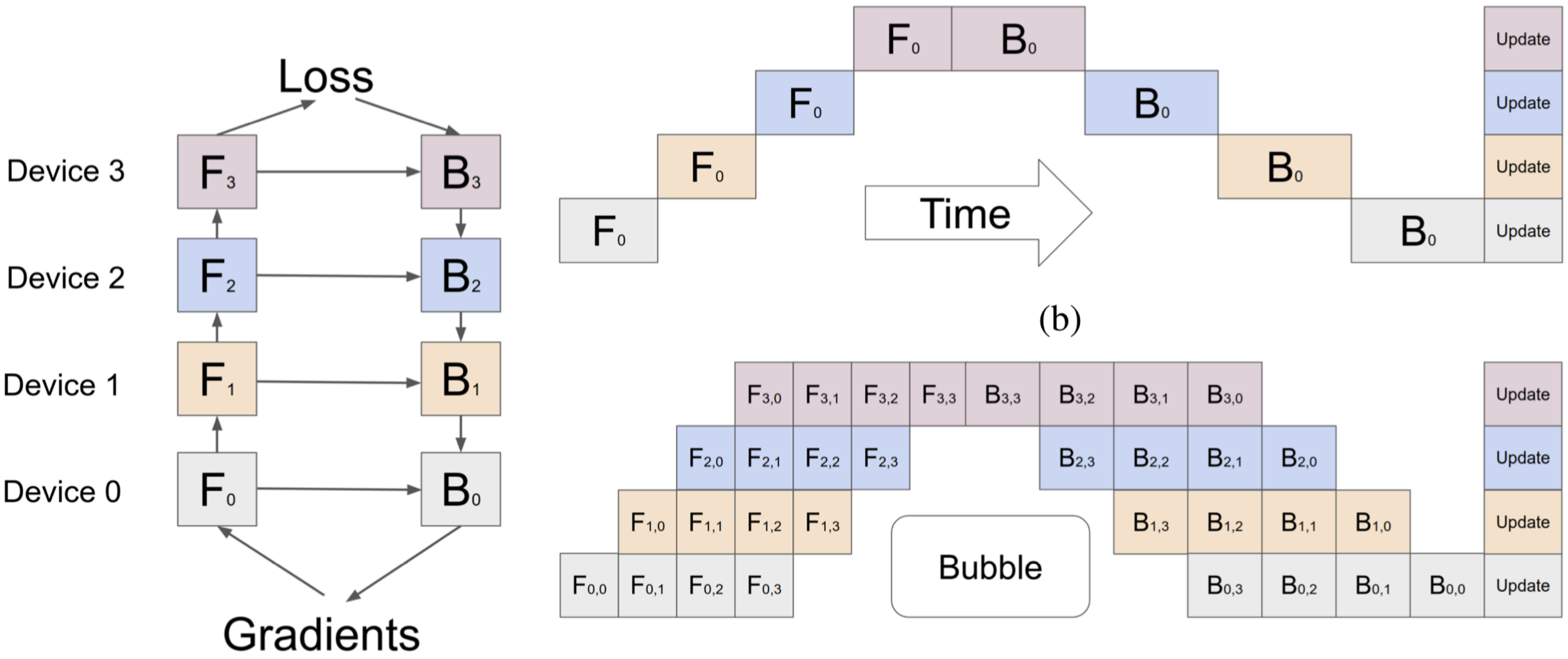
Divide the model across machines and replicate the data.

- ➢ Supports large models and activations
- ➢ Requires communication within single evaluation
- ➢ How to best divide a model?
  - ➢ Split individual layers
    - ➢ which dimension?
      - ➢ Batch or Spatial → depends on operation
  - ➢ Split across layers
    - ➢ Only one set of layers active a time → poor work balance
    - ➢ Soln: Pipelining Parallelism



Pipeline Parallelism

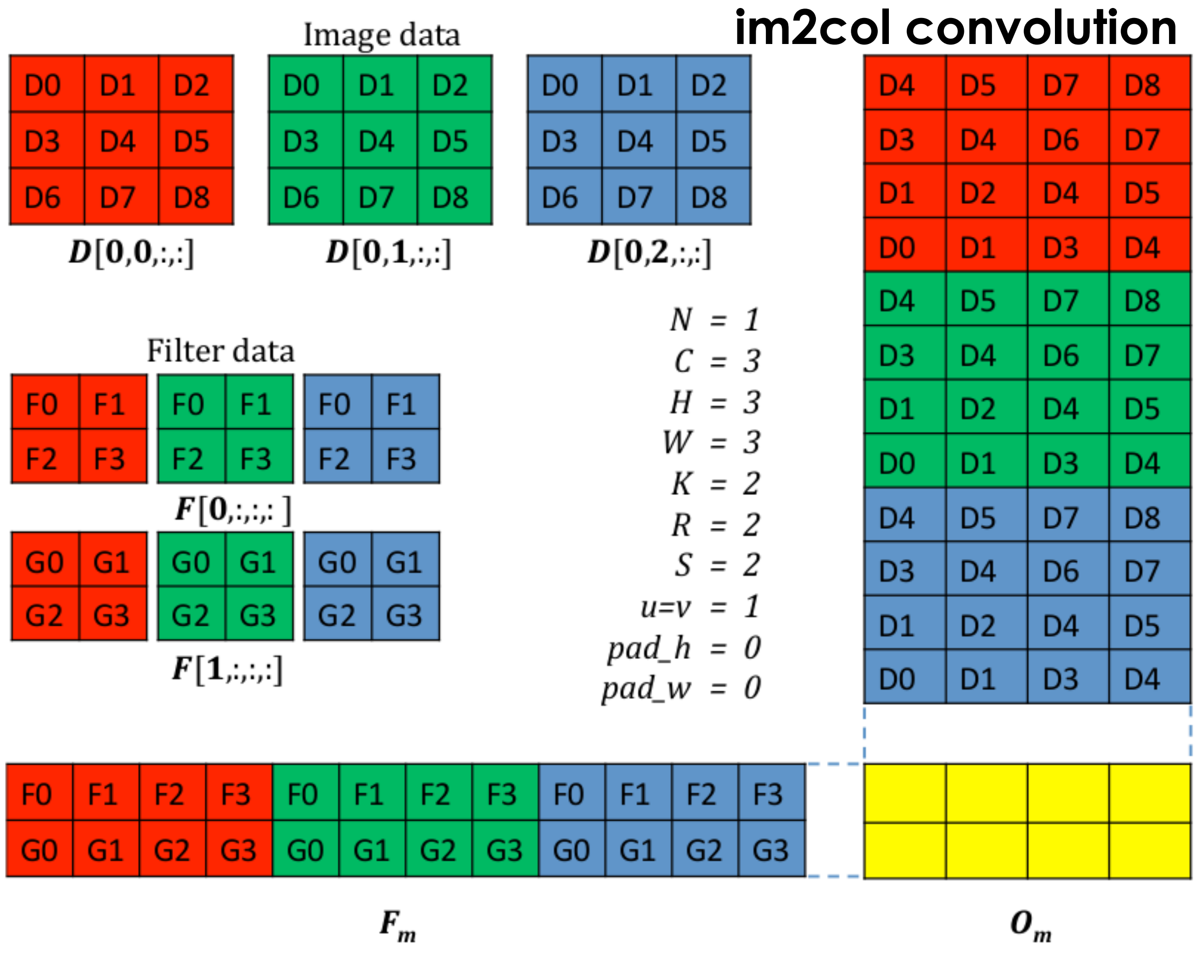
- ➢ Combine model and data parallelism to concurrently process multiple layers and batches.
  - ➢ Originally described in GPipe*



*[GPipe: Easy Scaling with Micro-Batch Pipeline Parallelism](#)

Operator Level Parallelism

- Exploiting the parallelism within linear algebra and convolution operations (a form of model parallelism)
- Multiple dimensions
  - Batch, spatial, time, …
- Typically cast operators as linear alg. routines and leverage optimizes BLAS libraries

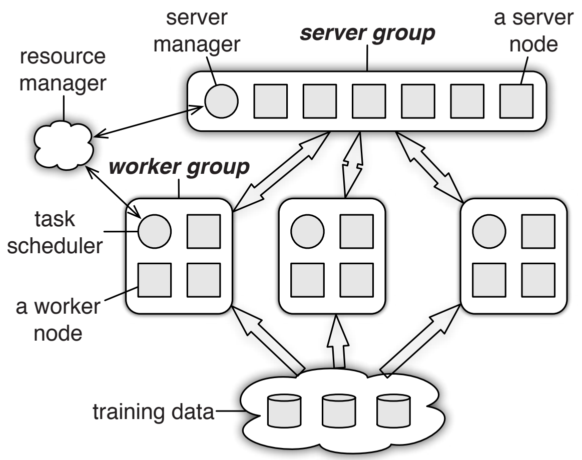


This weeks readings

Reading for the Week

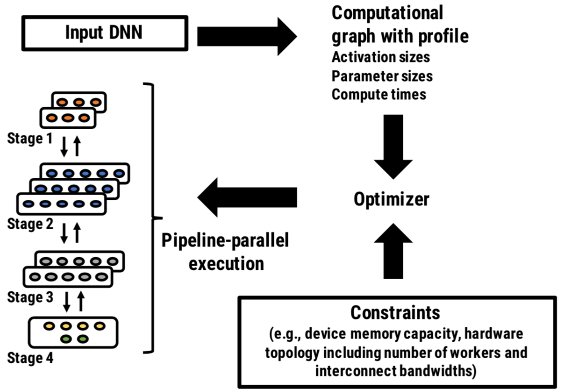
- Scaling Distributed Machine Learning with the Parameter Server (OSDI'14)
  - Paper describing the parameter server system
- PipeDream: Generalized Pipeline Parallelism for DNN Training (SOSP'19)
  - Latest paper exploring pipeline parallel training
- Adaptive Communication Strategies to Achieve the Best Error-Runtime Trade-off in Local-Update SGD (SysML'19)
  - Dynamic averaging approach to distributed training

Scaling Distributed Machine Learning with the Parameter Server (OSDI'14)



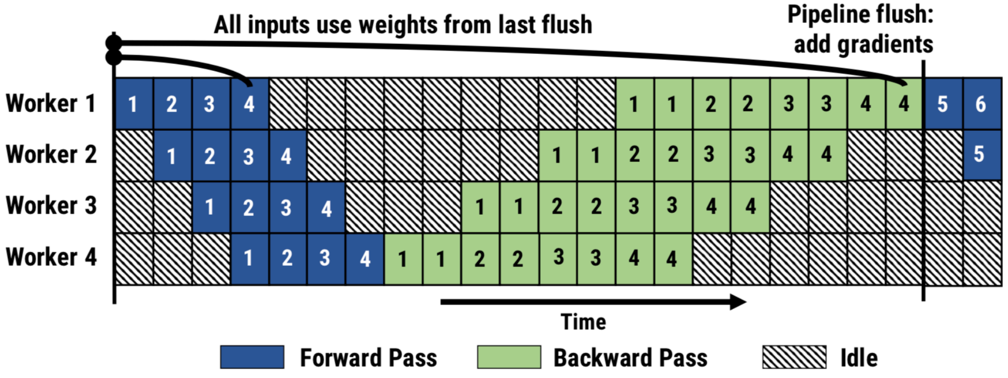
- Describes the key-value store customized for machine learning
  - Builds on earlier work in parameter servers
- **Additional Context:** focused on topic modeling and sparse regression
- **Key Ideas:** There are many ideas …
  - Keys – Value pairs with **linear algebra** semantics (e.g., get by range)
  - User defined **event handlers** on parameter servers and workers
  - Several different **consistency models**
  - **User defined filters** to determine when updates are communicated

PipeDream: Generalized Pipeline Parallelism for DNN Training (SOSP'19)

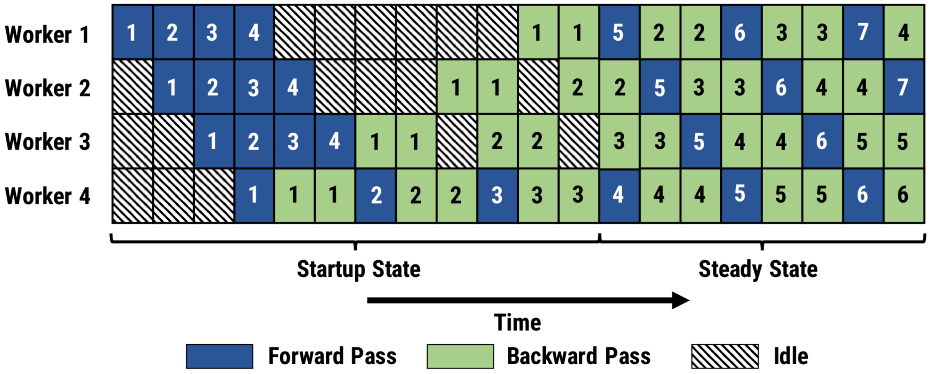


- Contemporaneously published with:
  - GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism (arXiv'18)
- **Key idea:** Leverage pipeline parallelism during training
  - **Automatically** constructs pipeline partition + schedule
  - Leverage **bounded staleness** + **versioned activations** to eliminate **bubbles**

GPipe



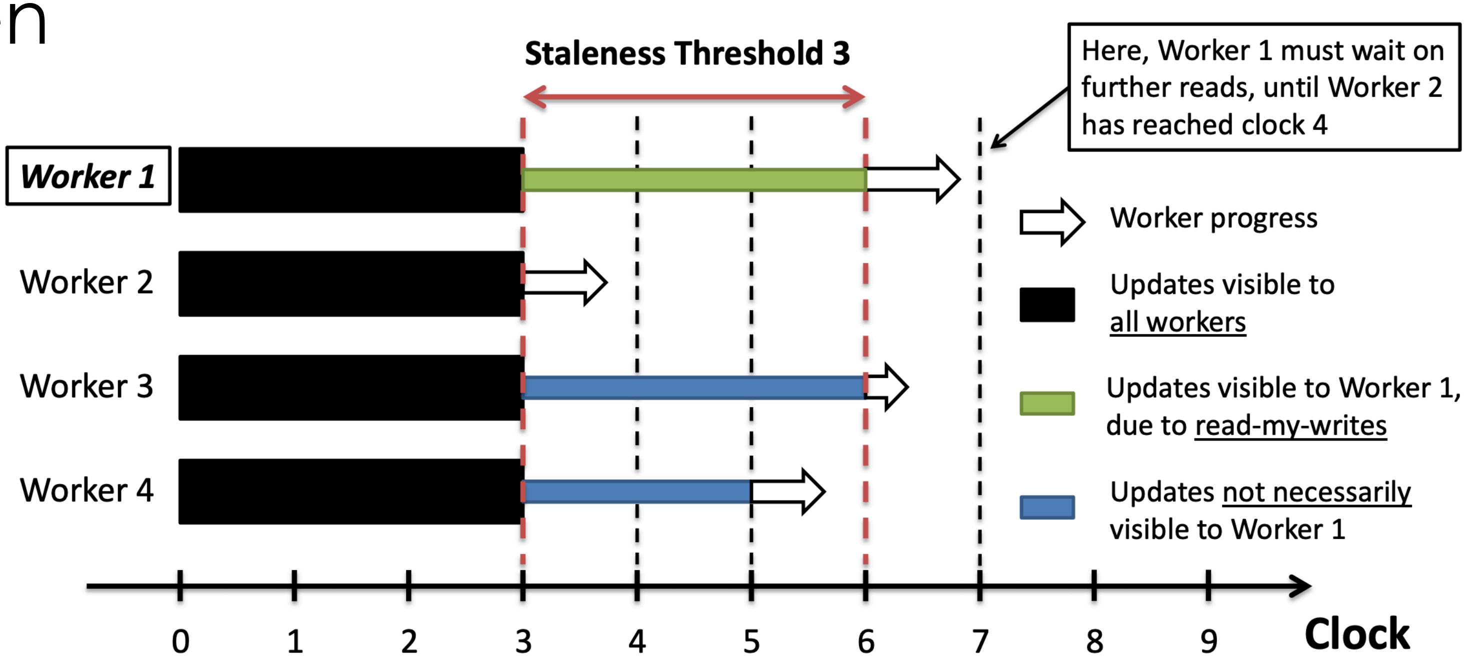
PipeDream



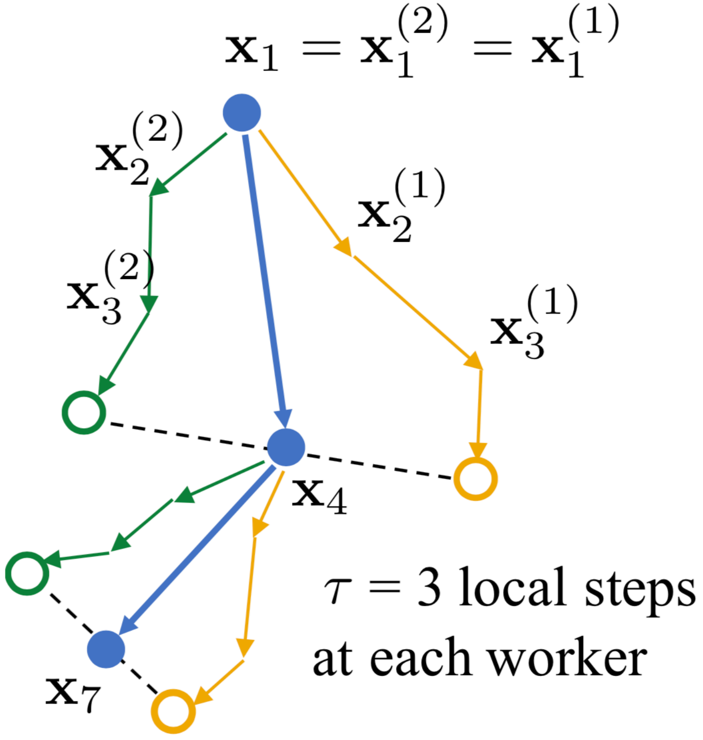
Bounded Staleness

- ➢ Developed as part of the parameter server work at CMU
  - ➢ More Effective Distributed ML via a Stale Synchronous Parallel Parameter Server (NIPS'13)
- ➢ Compromise between Hogwild and BSP
- ➢ Unclear implications for deep learning
  - ➢ Non-convex loss

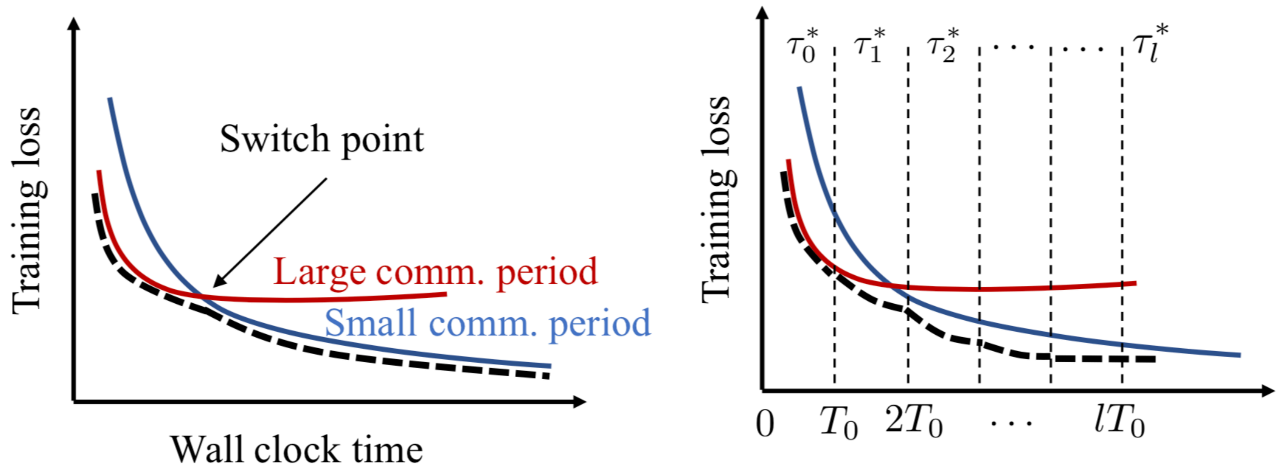
SSP: Bounded Staleness and Clocks



Adaptive Communication Strategies to Achieve the Best Error-Runtime Trade-off in Local-Update SGD (SysML'19)



- Studies Periodic Averaging SGD (PASGD)
- **Key Idea:** Change τ as algorithm converges



- More theoretical than previous reading
  - Theoretical results do not make convex assumptions!

Old Stuff