AI-Systems
Distributed Deep Learning (Part I)
(294-162)

Amir Gholami & Joseph E. Gonzalez

Acknowledgments

Many slides from Prof. Kurt Keutzer, Pallas Group

Agenda for Today

- 1:10-2:00: Preliminary Lecture on Parallel Training
- 2:00-2:45: PC Meeting Discussions
- 2:45-3:00: Break
- 3:00-4:00: Guest Lecture by Prof. Sophia Shao

Remaining slides from Last Lecture

Designing an accelerator

1) Accelerators are Only the First 80% of the Problem

The other 80%: Full system design

The remaining 200%: SW development

2) HW design shouldn't be about what can be built, rather what can be programmed

Stay tuned for the Lecture on AI Frameworks

3) Deploy at scale? Today's lecture

Research Challenges (if you are interested definition checkout EE 290 course by Prof. Shao)

Abstraction Levels, DSLs	HW Definition Languages
Coarse Grained vs Fine Grain Acceleration	Co-Design Methodology
Programming Languages, Programming Models	Datacenter / Design, Deployment
Optimization Techniques, Runtimes	Datacenter / Heterogeneity

Naveen Kumar (Google)

The right dataflow, precision, and many other parameters heavily depend on the workload.

If you were to design a HW today, it would at least take **~2years** for its tape out, with a cost of **~$500M** (estimate for 3nm)

It must remain relevant through ~5 years to justify the huge upfront investment

What do you think is the right workload for the future to bet on?

Distributed Deep Learning

Objectives For Today

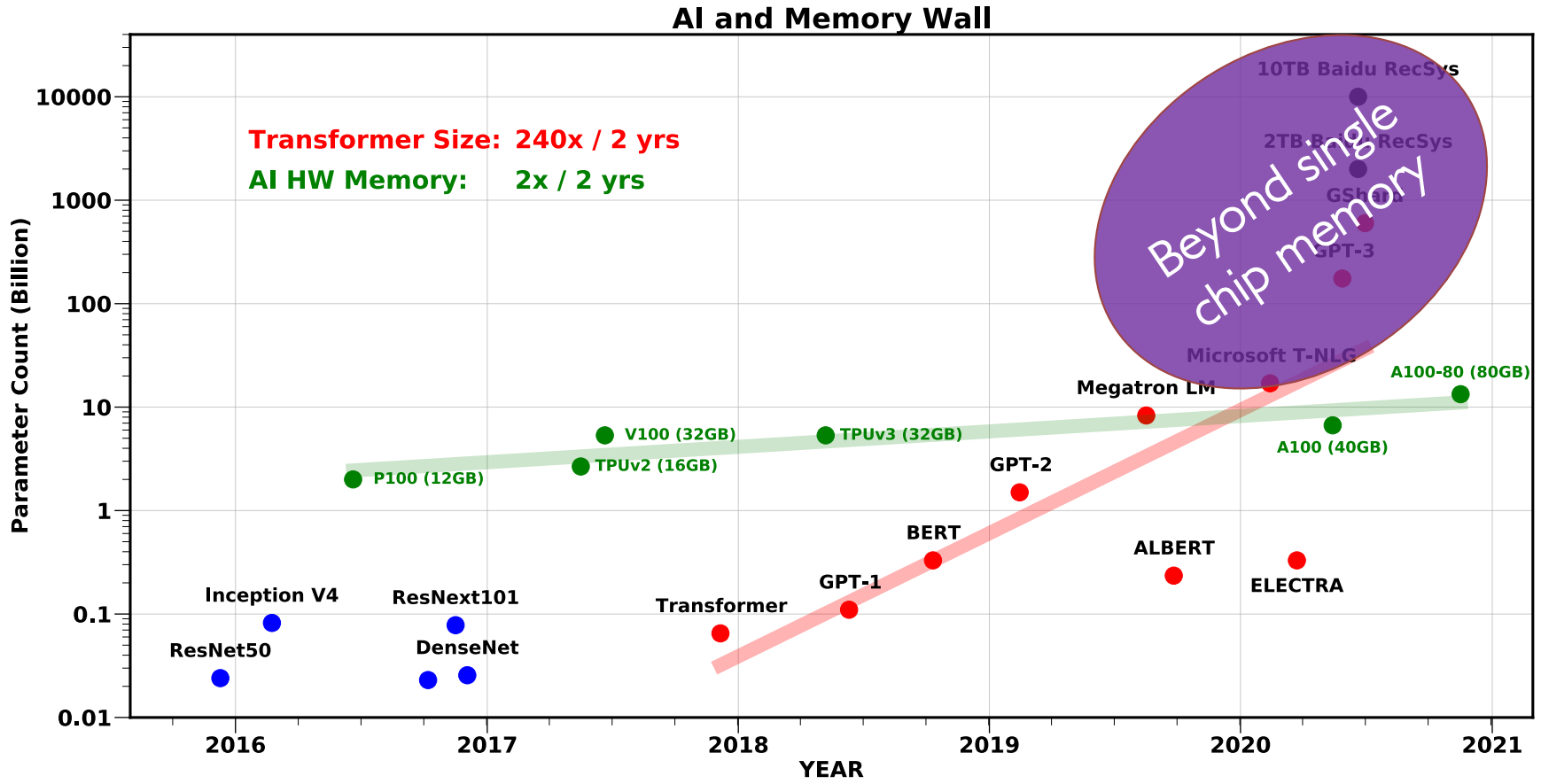
- Data Parallel Training and its Challenges
- Communication Complexity Analysis
- Model Parallel Training (Next Lecture)
- Memory Efficient Methods for Training Large Models (Next Lecture)

Distributed Training: What is it? & Why?

- **Distributed Training* ~** Training across multiple devices
  - Different local and remote memory speeds / network
- Why do we need distributed training?
  - **Additional memory** (memory bandwidth) for larger model
    - "Need" to store weights + activations
  - Faster training by leveraging **parallel computation**
  - Reduce or eliminate **data movement**
    - Privacy → Federated Learning
    - Limited bandwidth to edge devices
    - **(stay tuned for 04/25 lecture)**

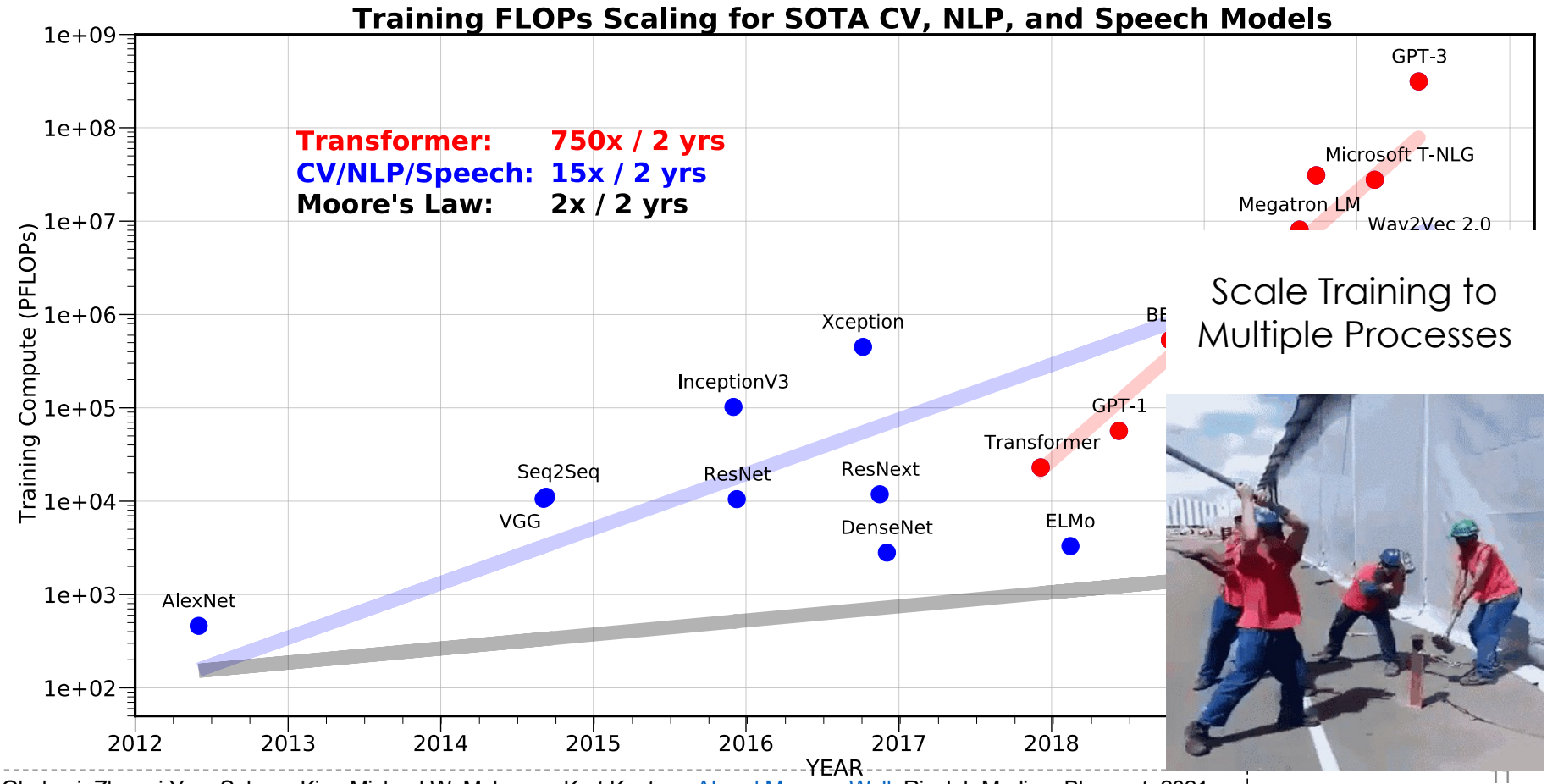
*Very simplified definition.

Training Large Models



Amir Gholami, Zhewei Yao, Sehoon Kim, Michael W. Mahoney, Kurt Keutzer, AI and Memory Wall, Riselab Medium Blogpost, 2021.

Faster Processing

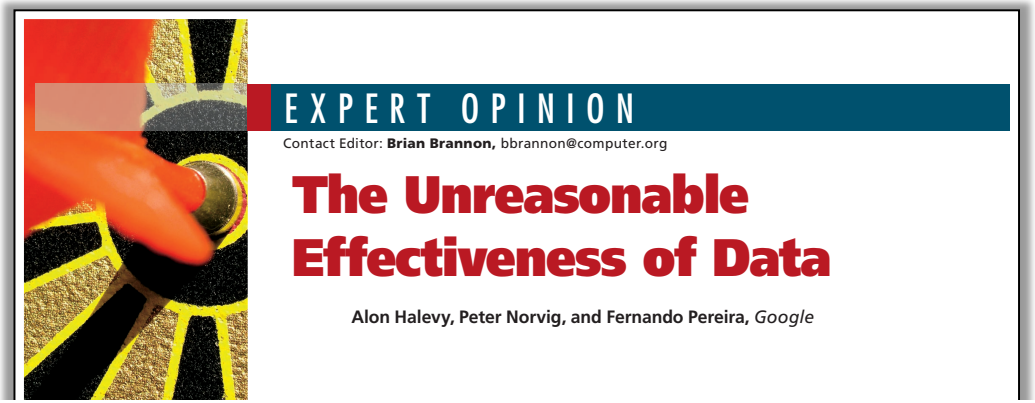


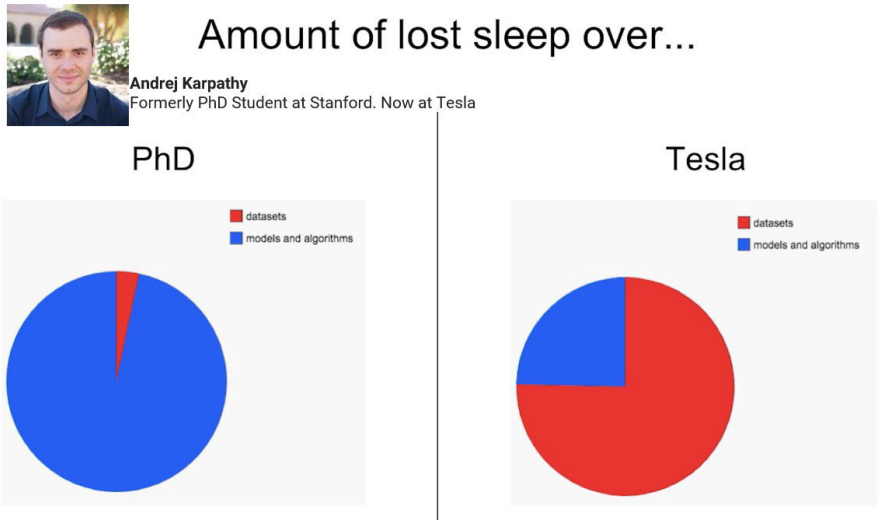
Scale Training to Multiple Processes

Amir Gholami, Zhewei Yao, Sehoon Kim, Michael W. Mahoney, Kurt Keutzer, AI and Memory Wall, Riselab Medium Blogpost, 2021.

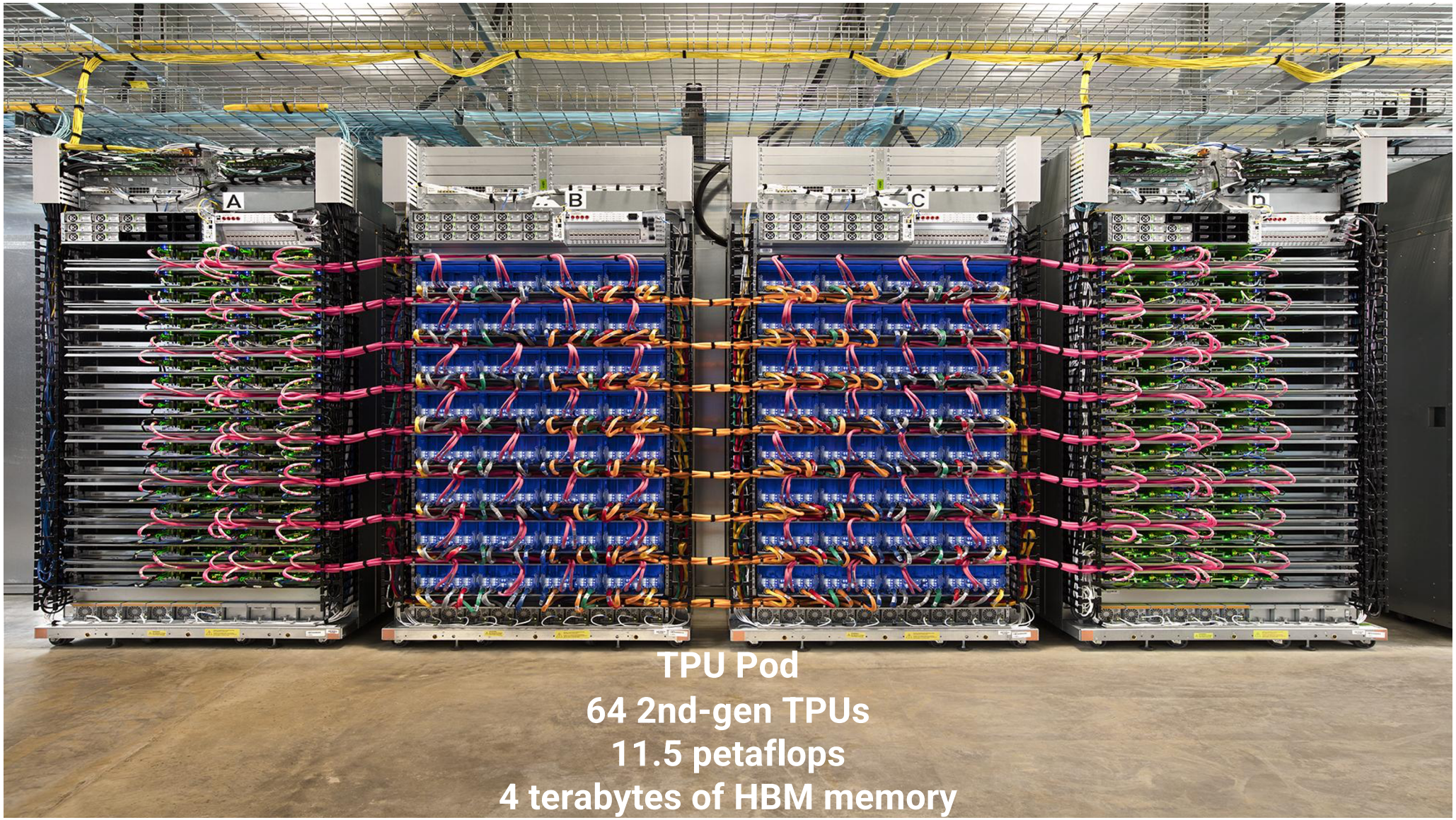
On Dataset Size and Learning

- 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

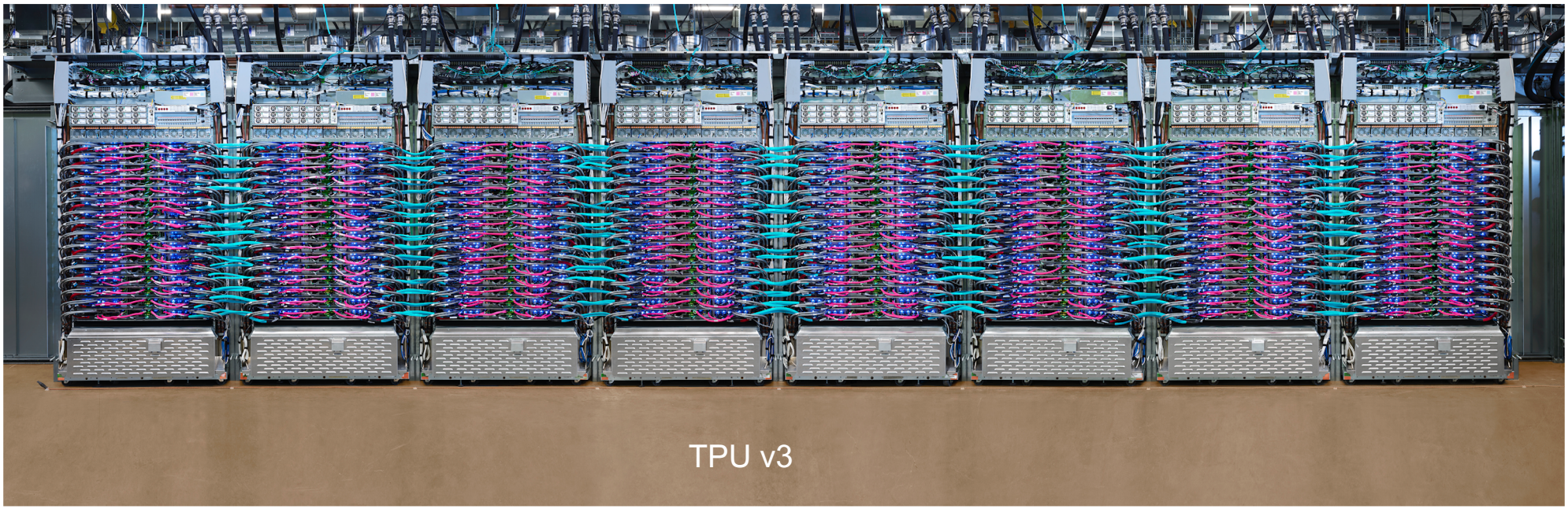




Example:
Scale is TPU's Primary Value Proposition


A
B
C
D
TPU Pod
64 2nd-gen TPUs
11.5 petaflops
4 terabytes of HBM memory

TPUv3



Ideal Metric of Success for Efficient Training

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

Convergence
Machine Learning
Property

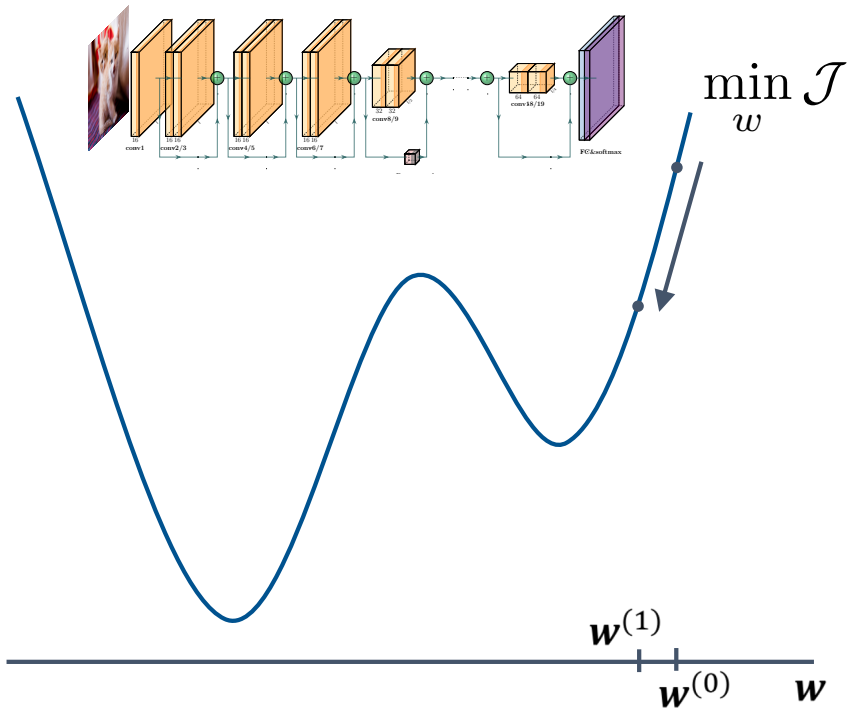
Throughput
System
Property

**Somewhat of a simplistic linear model. As we will later see there are many more moving parts to this*

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)?*

Gradient Descent



$$\min_w \mathcal{J}(w) = \frac{1}{N}\sum_{i=1}^{N} cost(w, x_i)$$

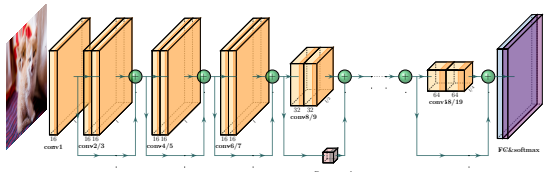
$$w^1 = w^0 - \alpha \underbrace{\frac{\partial \mathcal{J}(w^0)}{\partial w}}_{\Delta w}$$

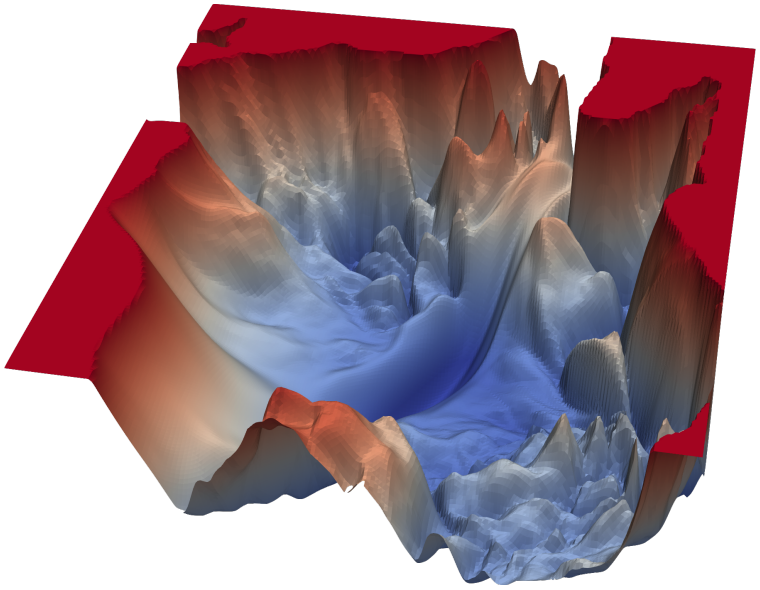
Learning rate

Two key elements:

- The computed gradient: the direction
- The learning rate: how big a step do we take?

Stochastic Gradient Descent





$$\min_w \mathcal{J}(w) = \frac{1}{N}\sum_{i=1}^{N} cost(w, x_i)$$

$$w^1 = w^0 - \underbrace{\frac{\alpha}{B}\sum_{i=1}^{B}\frac{\partial \mathcal{J}(w^0)}{\partial w}}_{\Delta w}$$

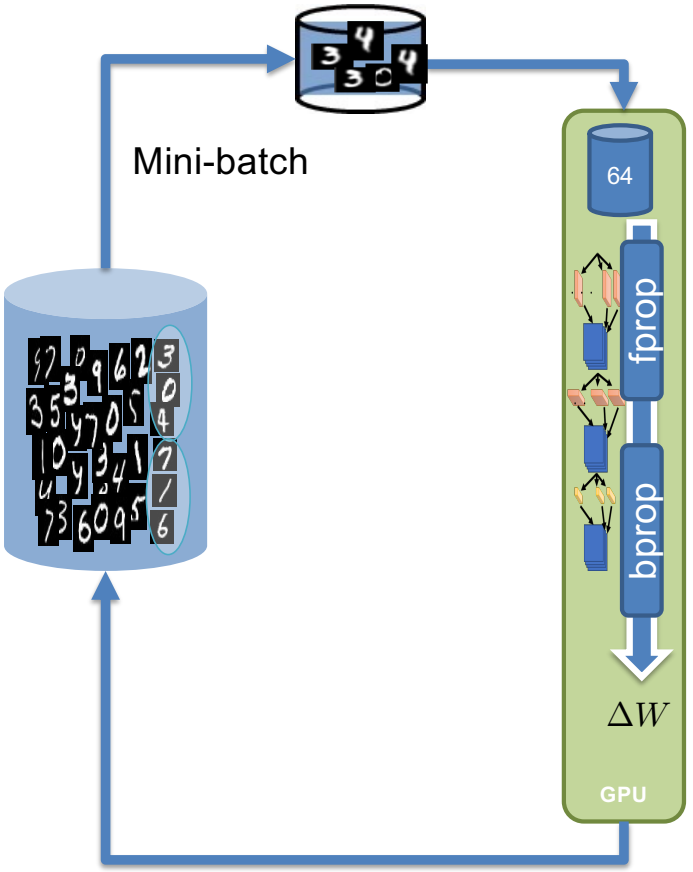
Learning rate

Two key elements:

- The computed gradient: the direction
- The learning rate: how big a step do we take?

Synchronous Stochastic Gradient Descent

In every iteration of SGD we load a **random mini-batch of training** data, and compute the gradient.



$$\min_w \mathcal{J}(w) = \frac{1}{N}\sum_{i=1}^{N} cost(w, x_i)$$

$$w^1 = w^0 \underbrace{- \frac{\alpha}{B}\sum_{i=1}^{B} \frac{\partial \mathcal{J}(w^0)}{\partial w}}_{\Delta w}$$

Parallelization Opportunities

Data Parallelism: *Distribute the processing of data to multiple PEs.*

$$w^1 = w^0 - \frac{\alpha}{B} \sum_{i=1}^{B} \frac{\partial \mathcal{J}(w^0)}{\partial w}$$

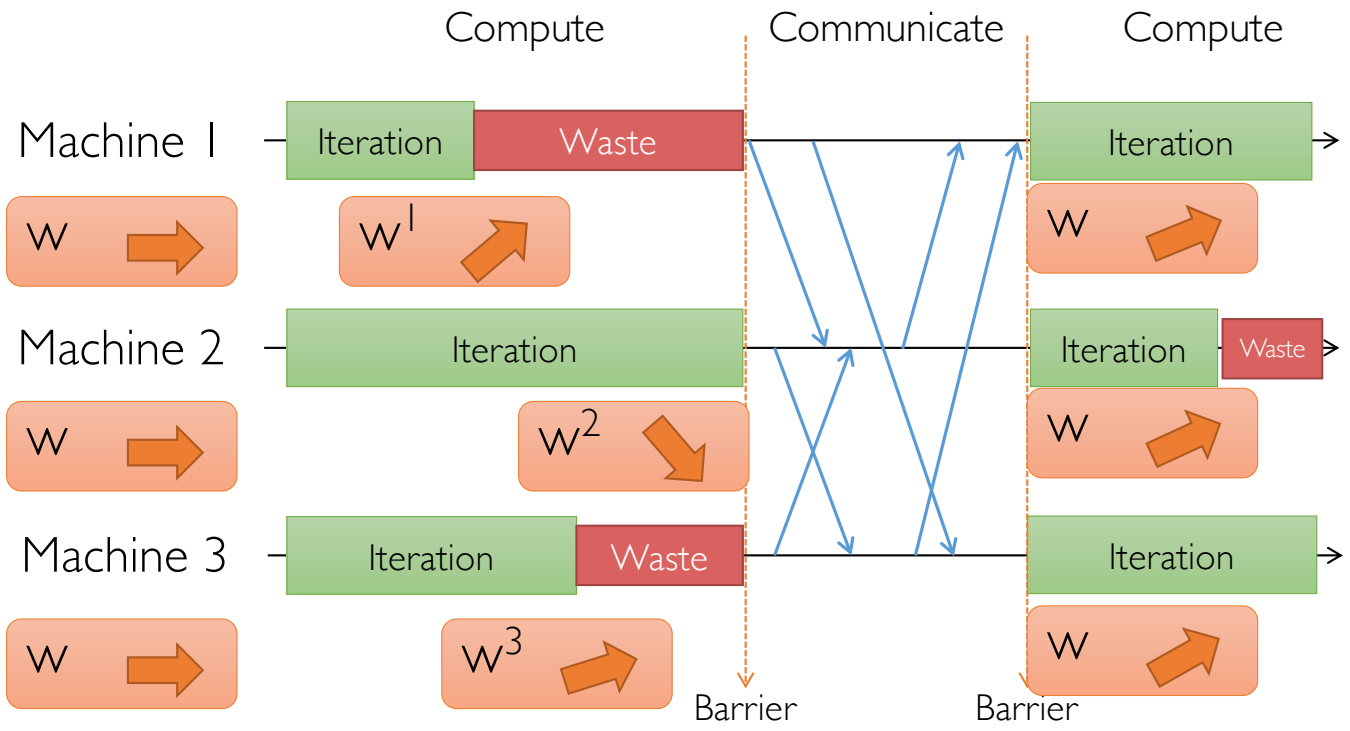
Model Parallelism: *Break the model and distribute processing of every layer to multiple PEs*

$$w^1 = w^0 - \frac{\alpha}{B} \sum_{i=1}^{B} \frac{\partial \mathcal{J}(w^0)}{\partial w}$$

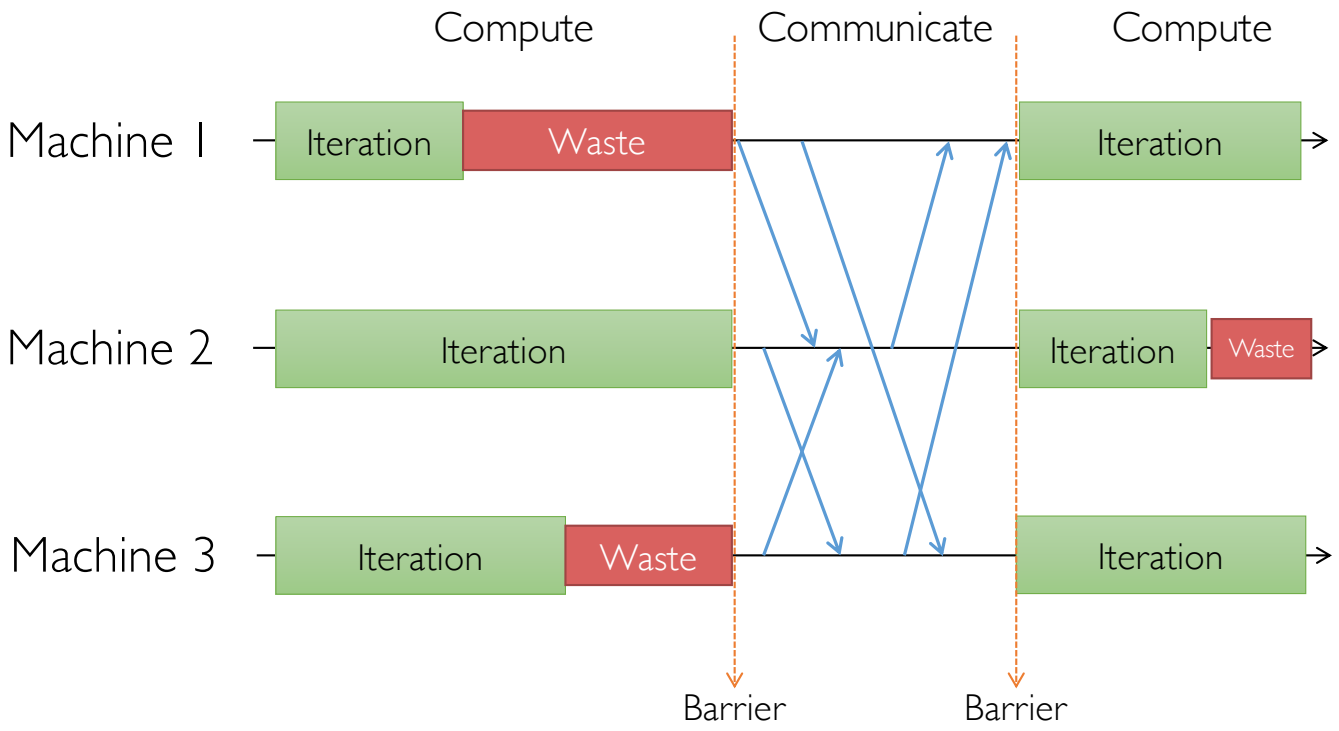
For either approach it is also possible to use ***synchronous*** *or* ***asynchronous*** *updates*

$$w^1 = w^0 - \frac{\alpha}{B} \sum_{i=1}^{B} \frac{\partial \mathcal{J}(w^0)}{\partial w}$$

Bulk Synchronous Parallel (BSP) Execution

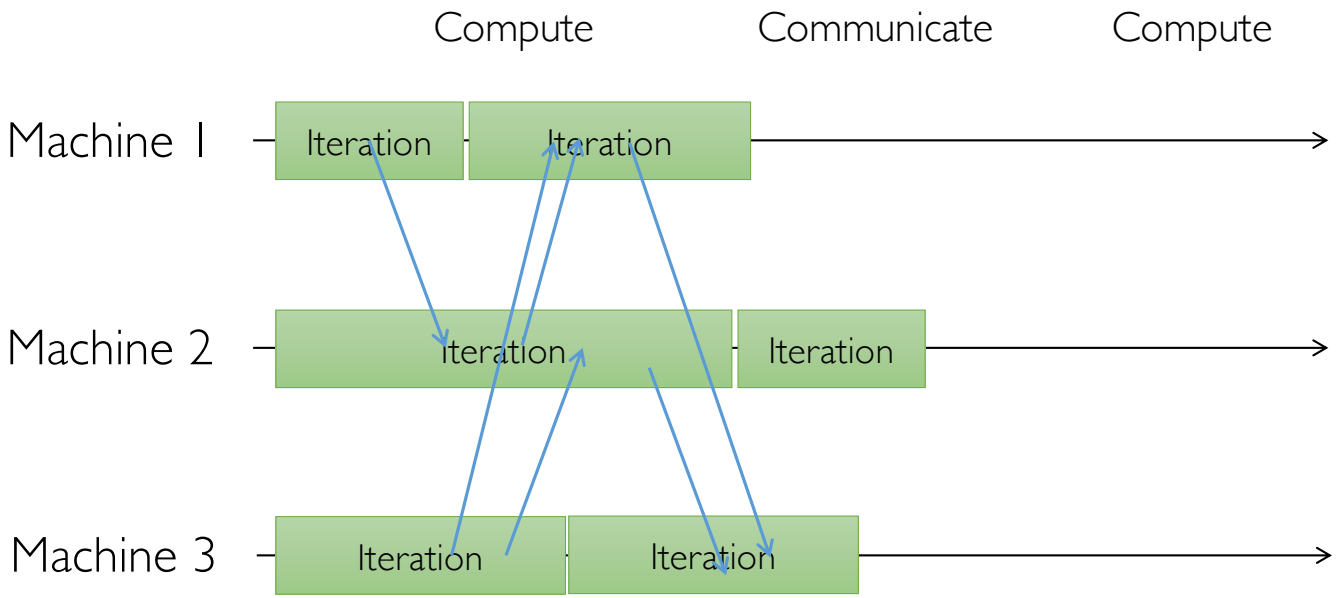


Bulk Synchronous Parallel (BSP) Execution



Enable more frequent coordination on parameter values

Asynchronous Execution



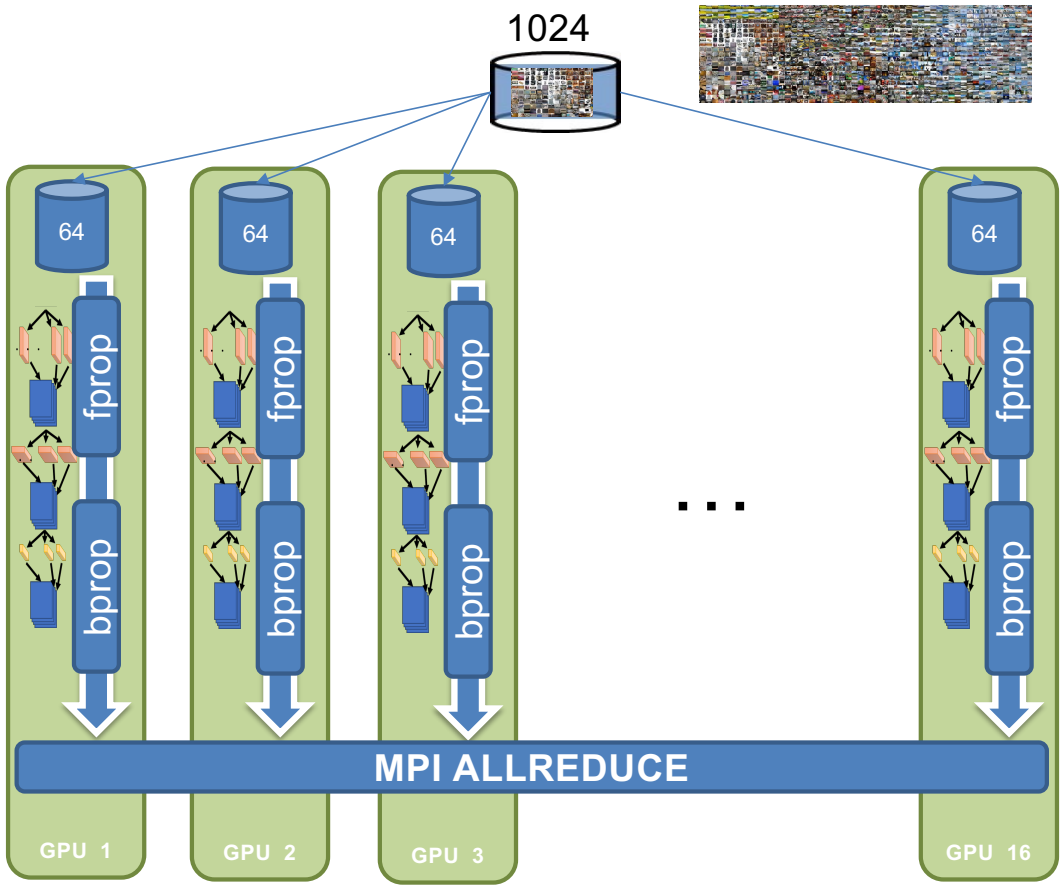
Enable more frequent coordination on parameter values, but often results in generalization loss. Today we will only focus on synchronous training.

Synchronous Data Parallel

Synchronous Data Parallelism

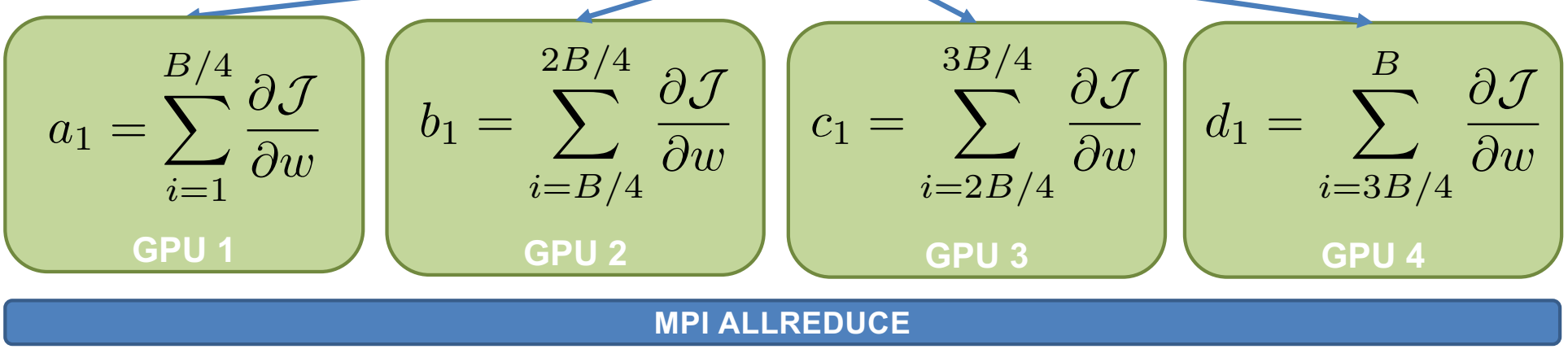
- Compute the entire model on each processor
- Distribute the batch evenly across each processor:
  - 1024 batch distributed over 16 PEs: 64 images per GPU
- Communicate gradient updates through **allreduce**

$$w^1 = w^0 - \frac{\alpha}{B} \sum_{i=1}^{B} \frac{\partial \mathcal{J}(w^0)}{\partial w}$$



All Reduce

$$w^1 = w^0 - \frac{\alpha}{B} \sum_{i=1}^{B} \frac{\partial \mathcal{J}(w^0)}{\partial w}$$

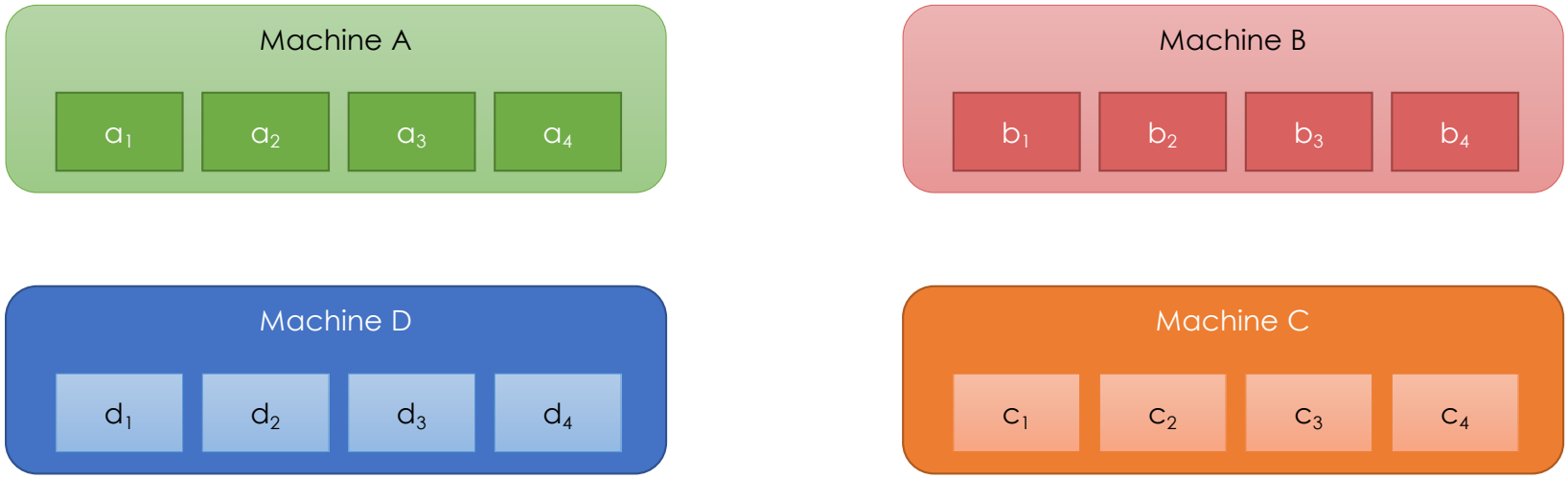


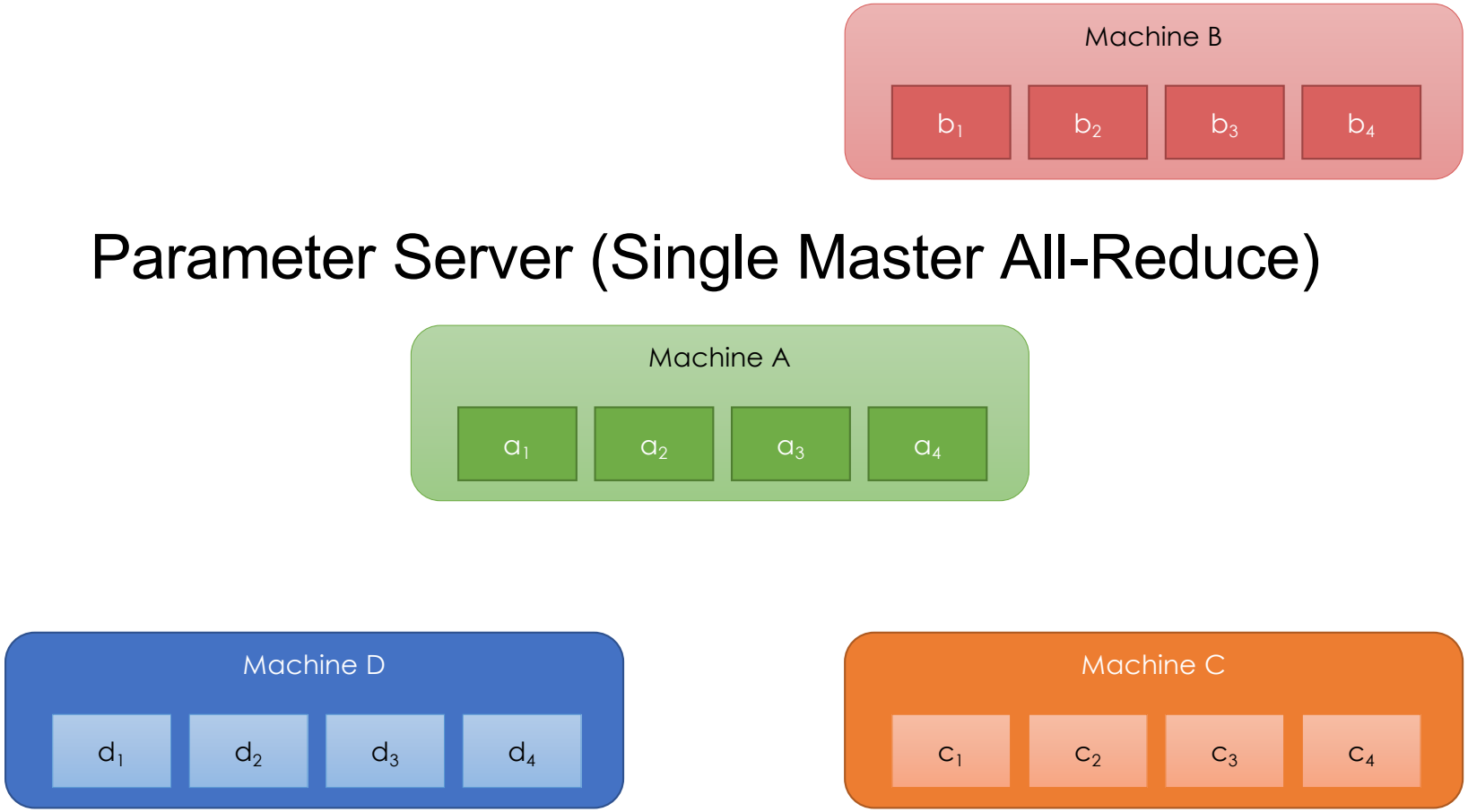
$$\sum_{i=1}^{B} \frac{\partial \mathcal{J}}{\partial w} = a_1 + b_1 + c_1 + d_1$$

All Reduce

There are many different all reduce algorithms, each with their own trade offs.

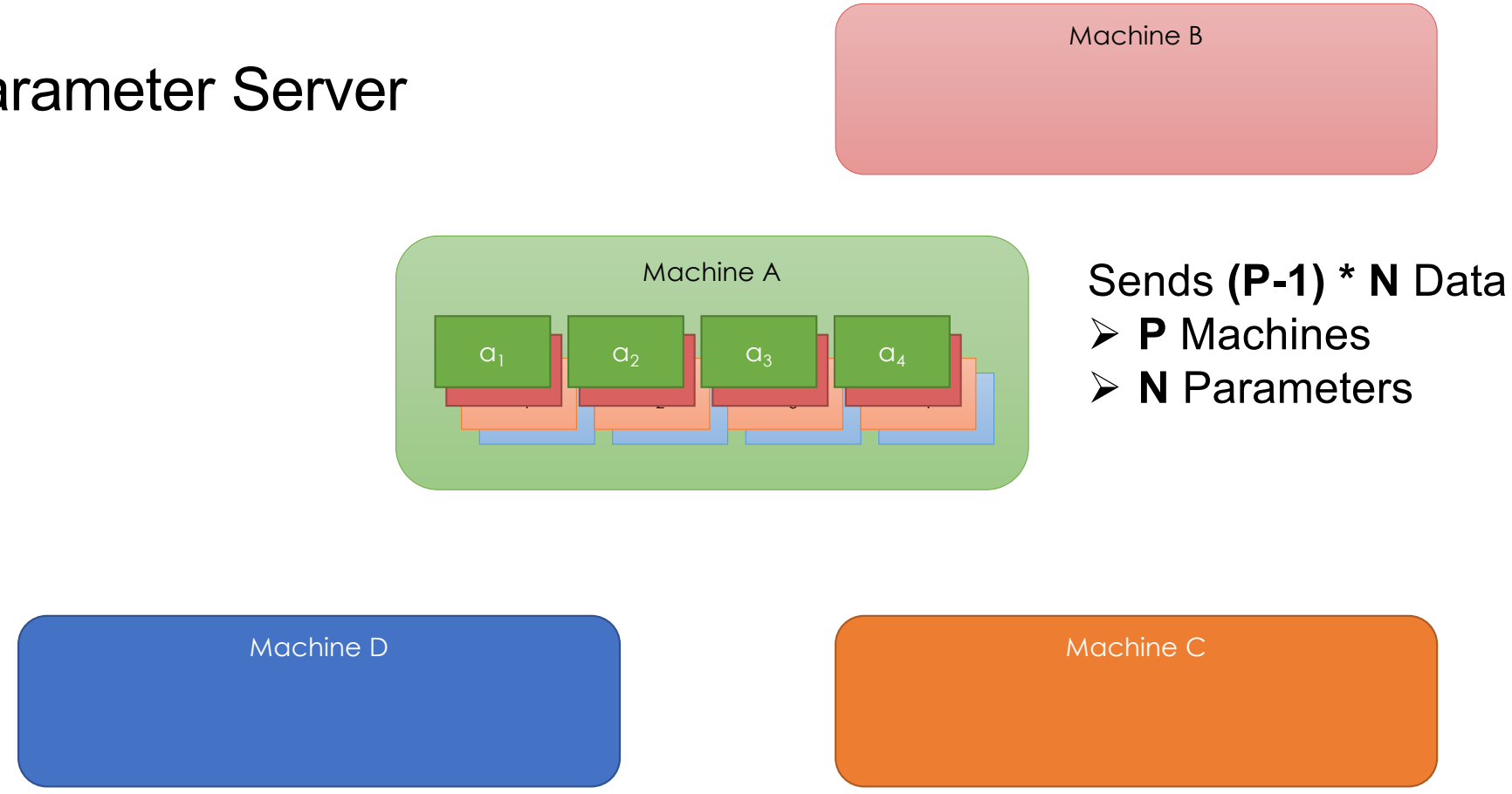
For simplicity, assume our model has 4 layers, and is trained on P=4 machines





Parameter Server (Single Master All-Reduce)

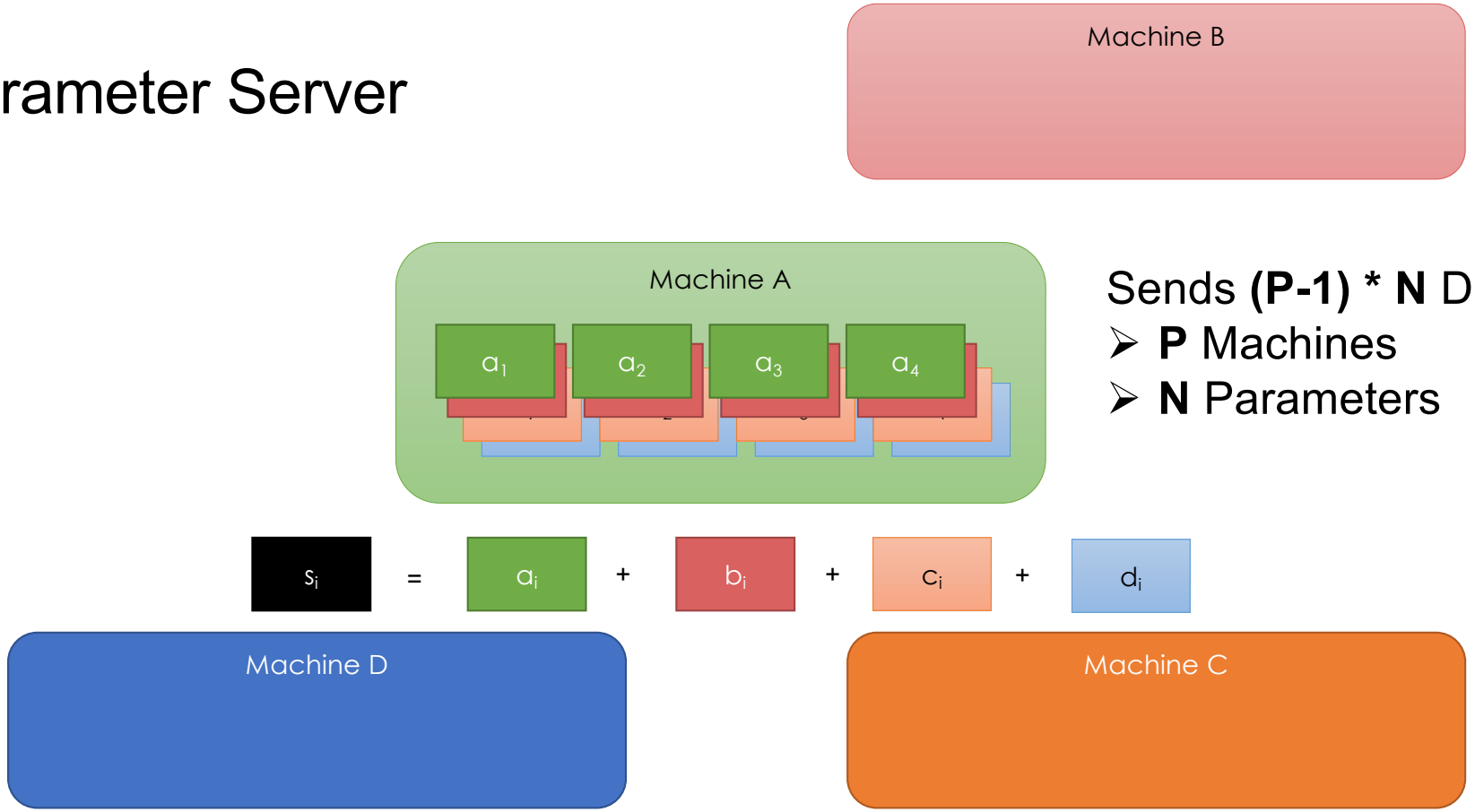
Parameter Server



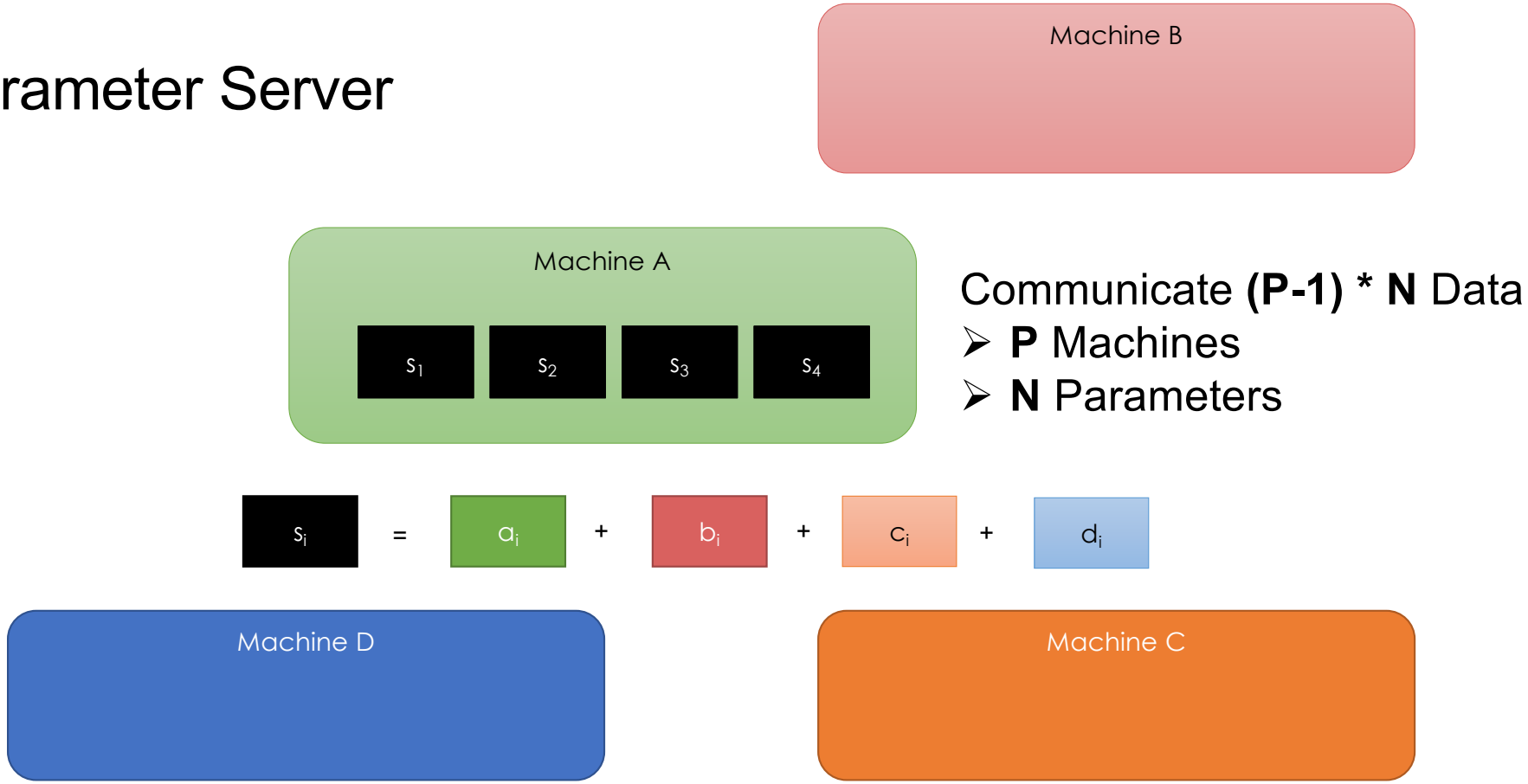
Sends **(P-1) * N** Data

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

Parameter Server



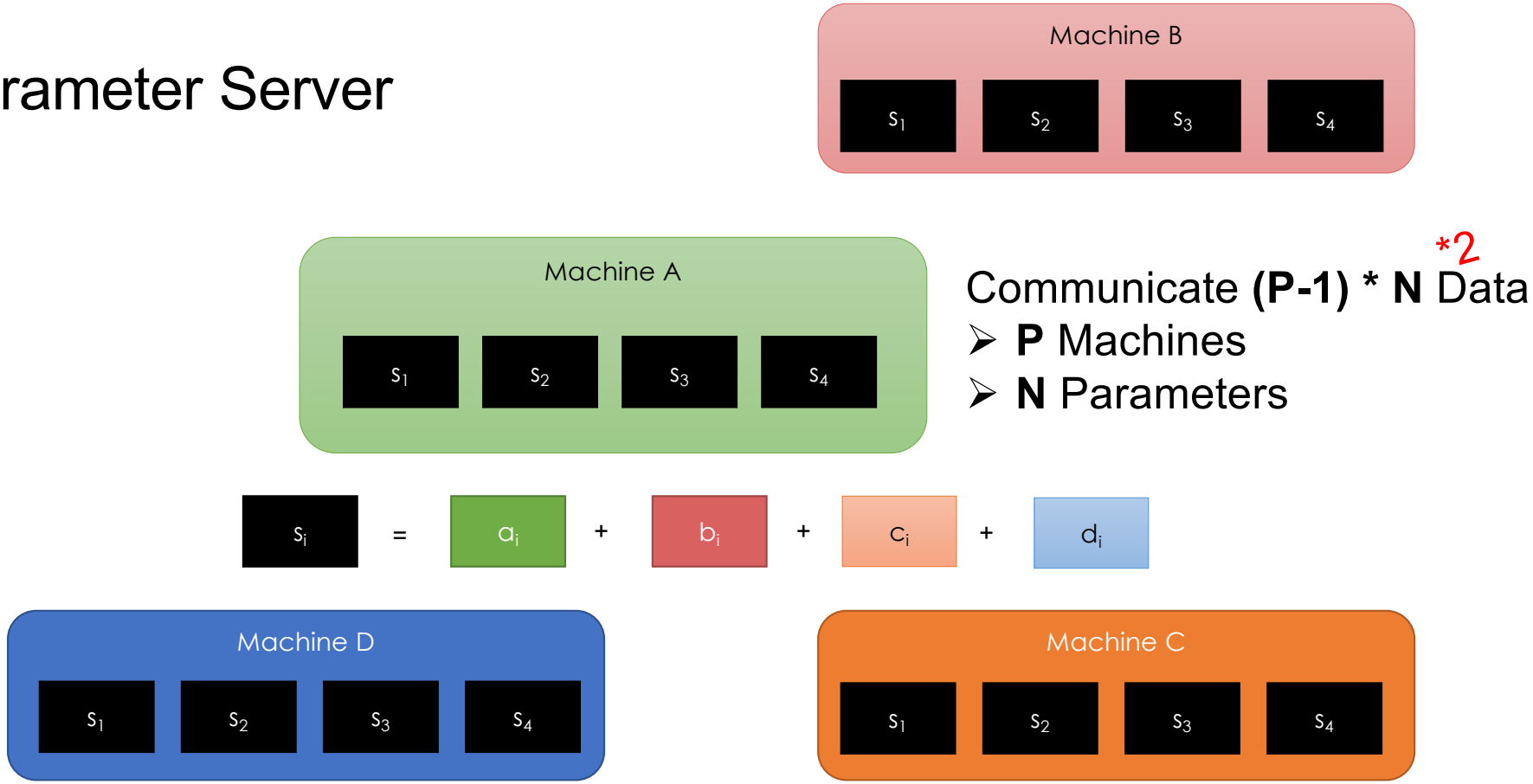
Parameter Server



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

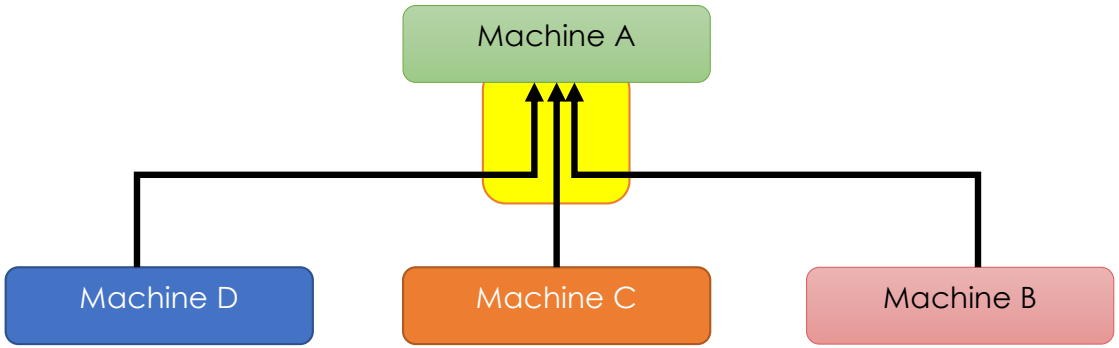
- **P** Machines
- **N** Parameters

Parameter Server

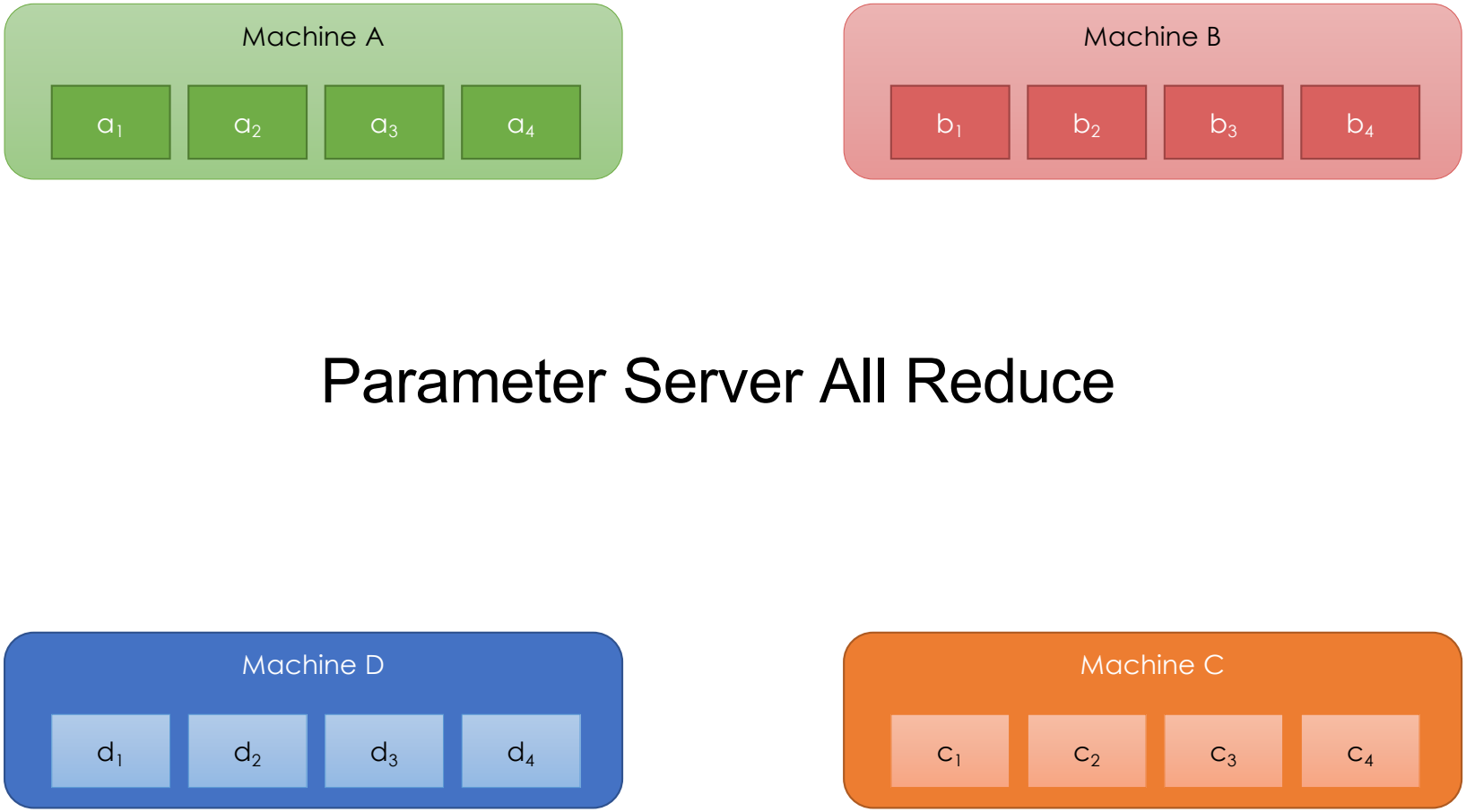


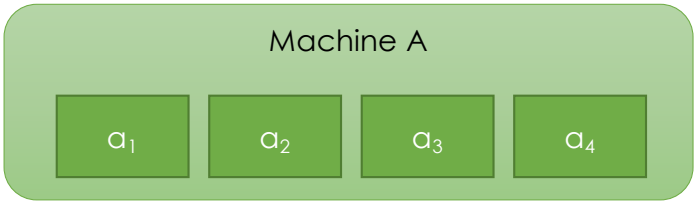
Parameter Server

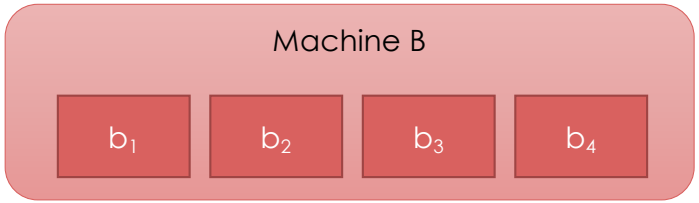
Comm **(P-1) * N** Data *2
- **P** Machines
- **N** Parameters



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

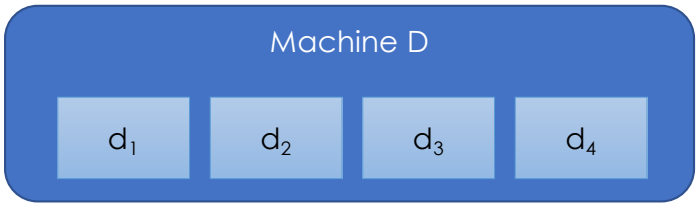

Machine A
a_1
a_2
a_3
a_4
Machine B
b_1
b_2
b_3
b_4
Parameter Server All Reduce
Machine D
d_1
d_2
d_3
d_4
Machine C
c_1
c_2
c_3
c_4

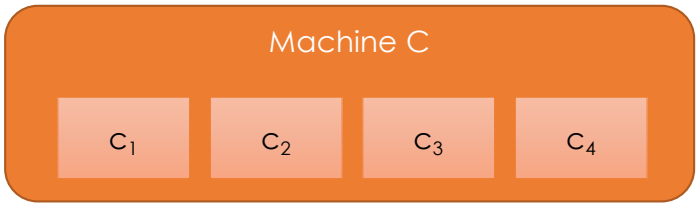


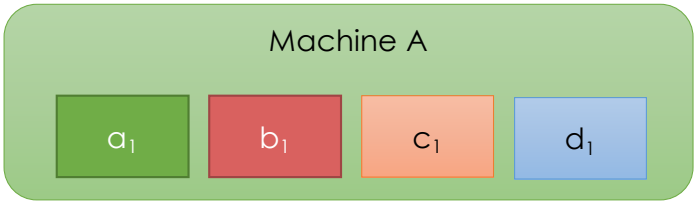


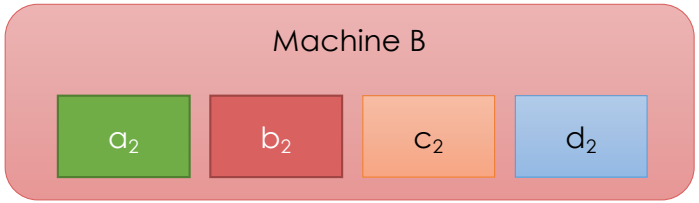
Send each entry to parameter server for that entry.

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





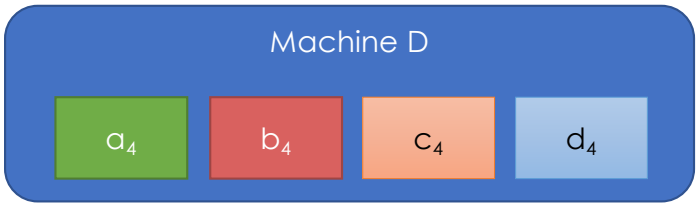


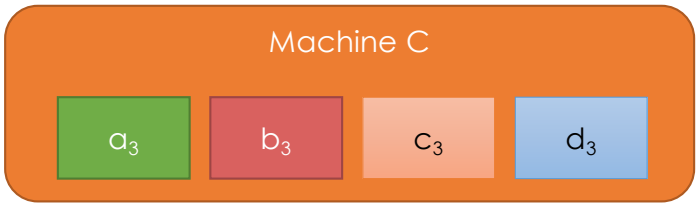


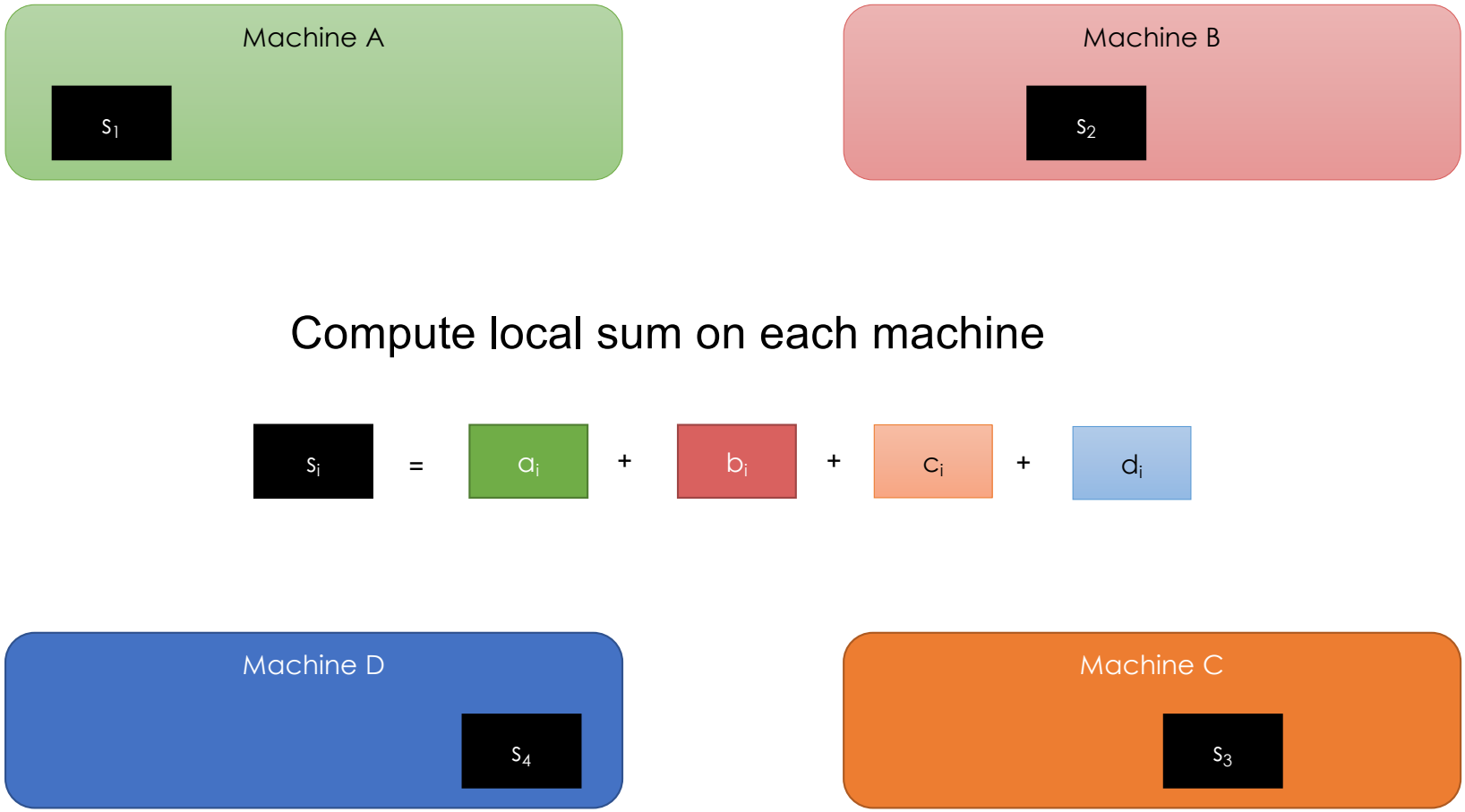
Each machine sends N/P data to all other machines.

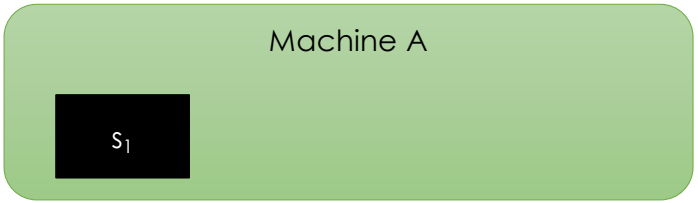
(P-1) * N/P

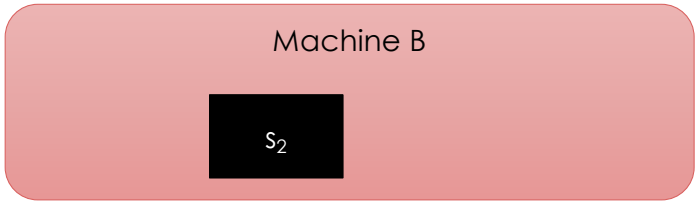
- **P** Machines
- **N** Parameters





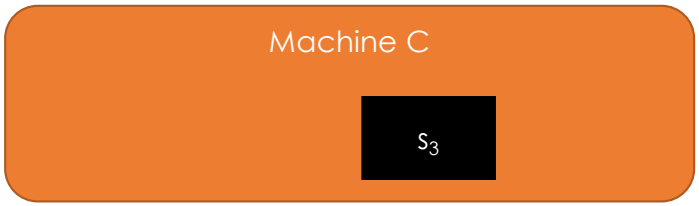

Machine A
s_1
Machine B
s_2
Compute local sum on each machine
s_i = a_i + b_i + c_i + d_i
Machine D
s_4
Machine C
s_3



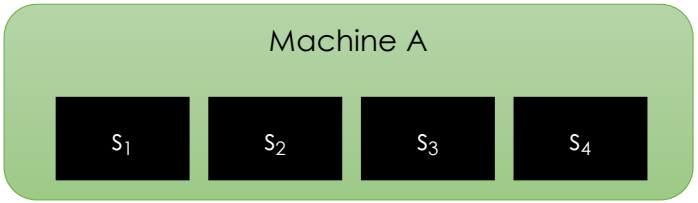


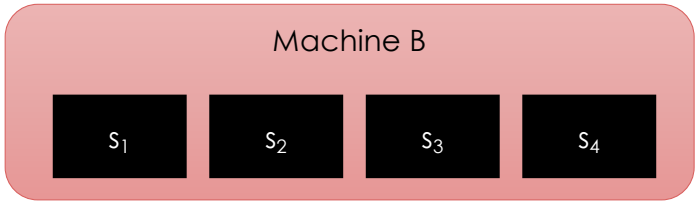
Each machine broadcasts* the sum (N/P data size) to all other machines.
(P-1) * N/P

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



*** Technically All Gather based on MPI communication definition**

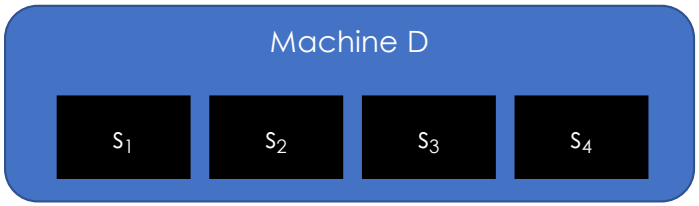


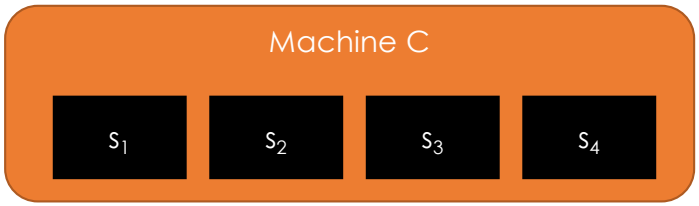


Total Communication per machine:

2* (P-1) * N/P (roughly independent of P)

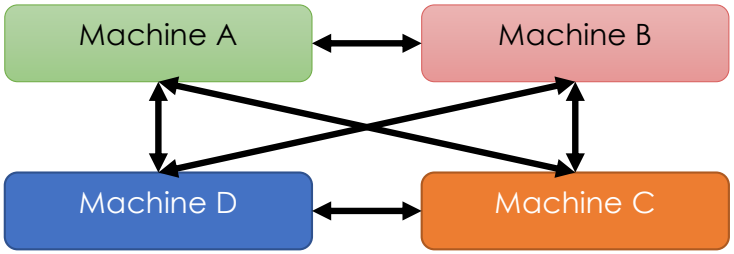
- **P** Machines
- **N** Parameters



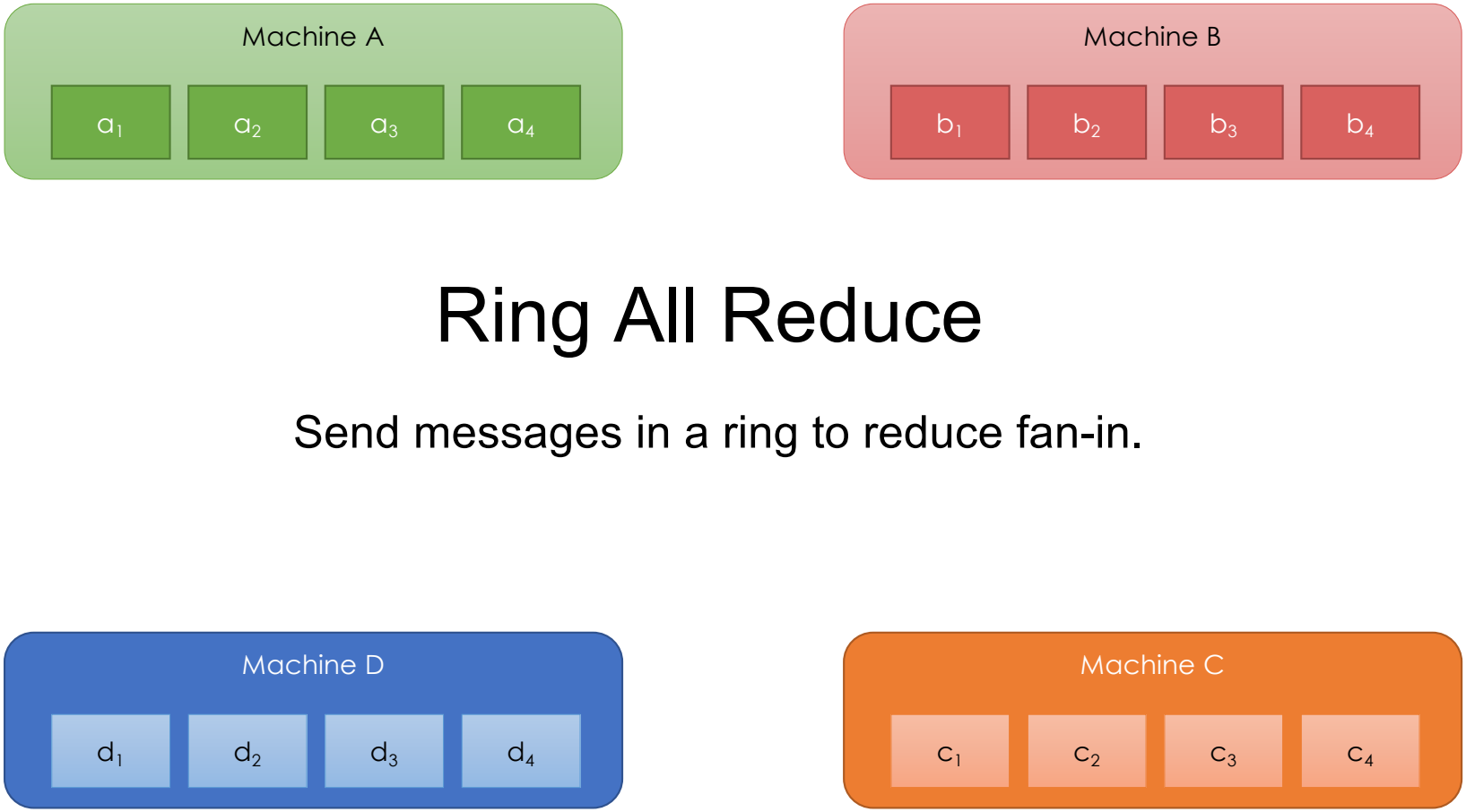


Parameter Server All-Reduce

- Same amount of total data transmitted as before, but spread evenly across all machines instead of just one

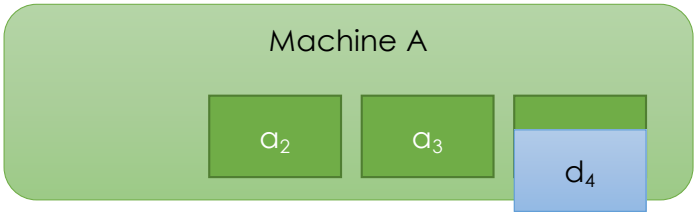


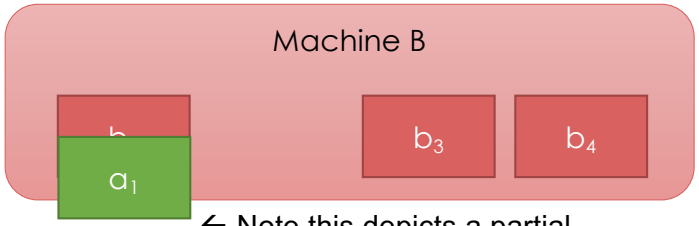
- Same **high fan-in** (P-1)
- **Reduced** Inbound Bandwidth = 2*(P-1)N/P
  - Previously 2*(P-1)*N for the parameter server



Ring All Reduce

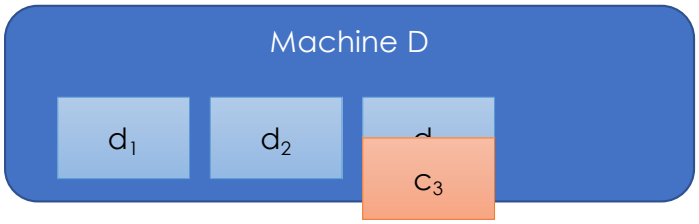
Send messages in a ring to reduce fan-in.

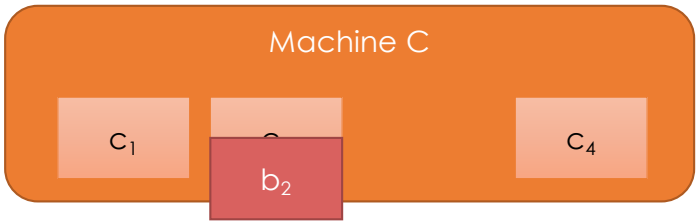




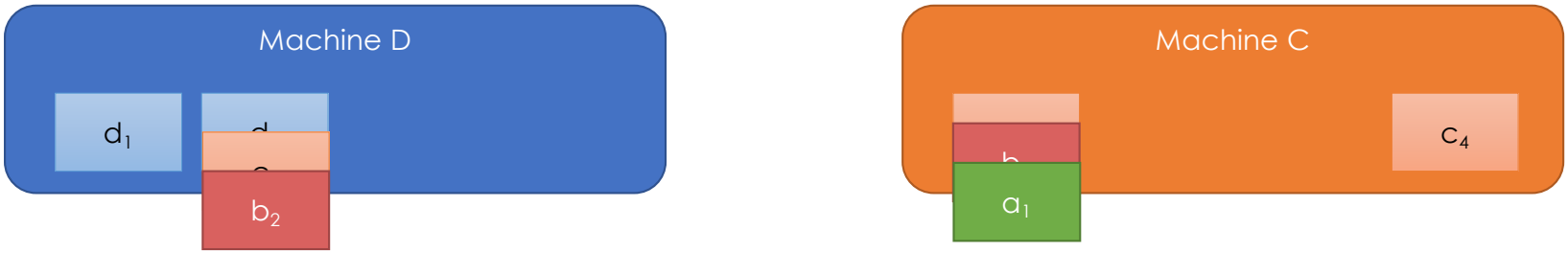
← Note this depicts a partial sum and not a bigger message.

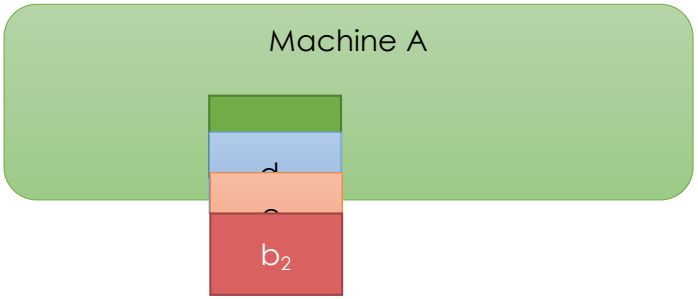
Ring All Reduce

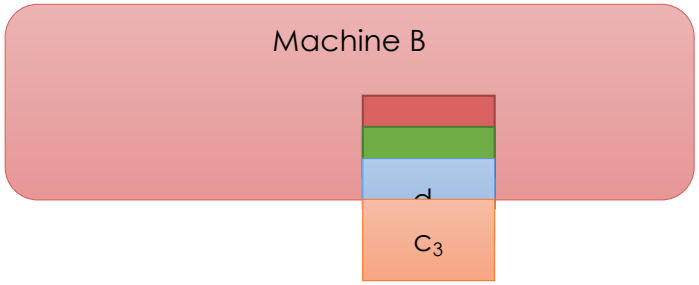




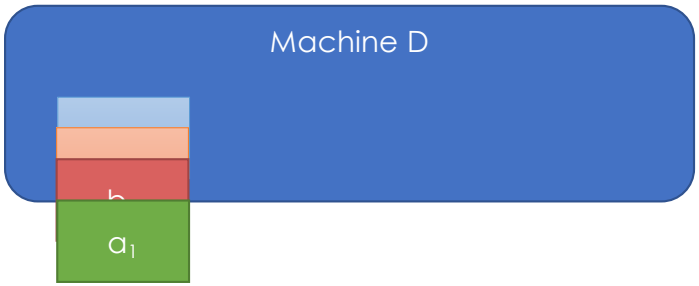
Ring All Reduce

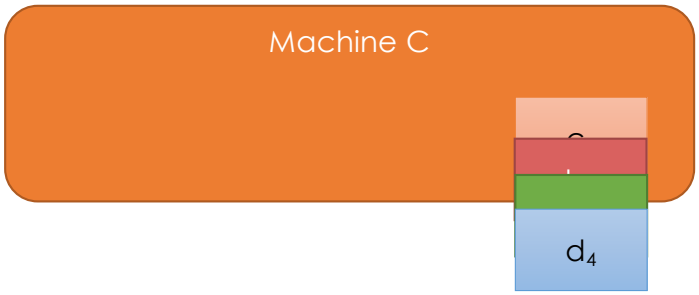


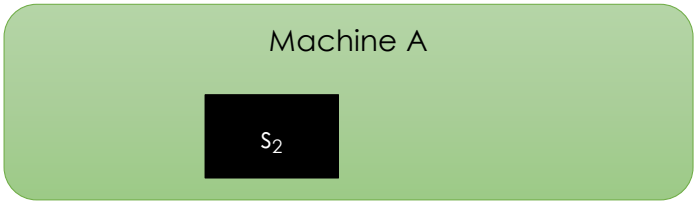


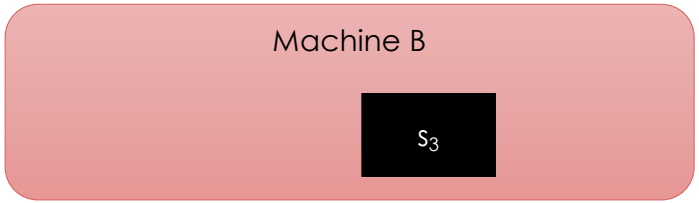


Ring All Reduce





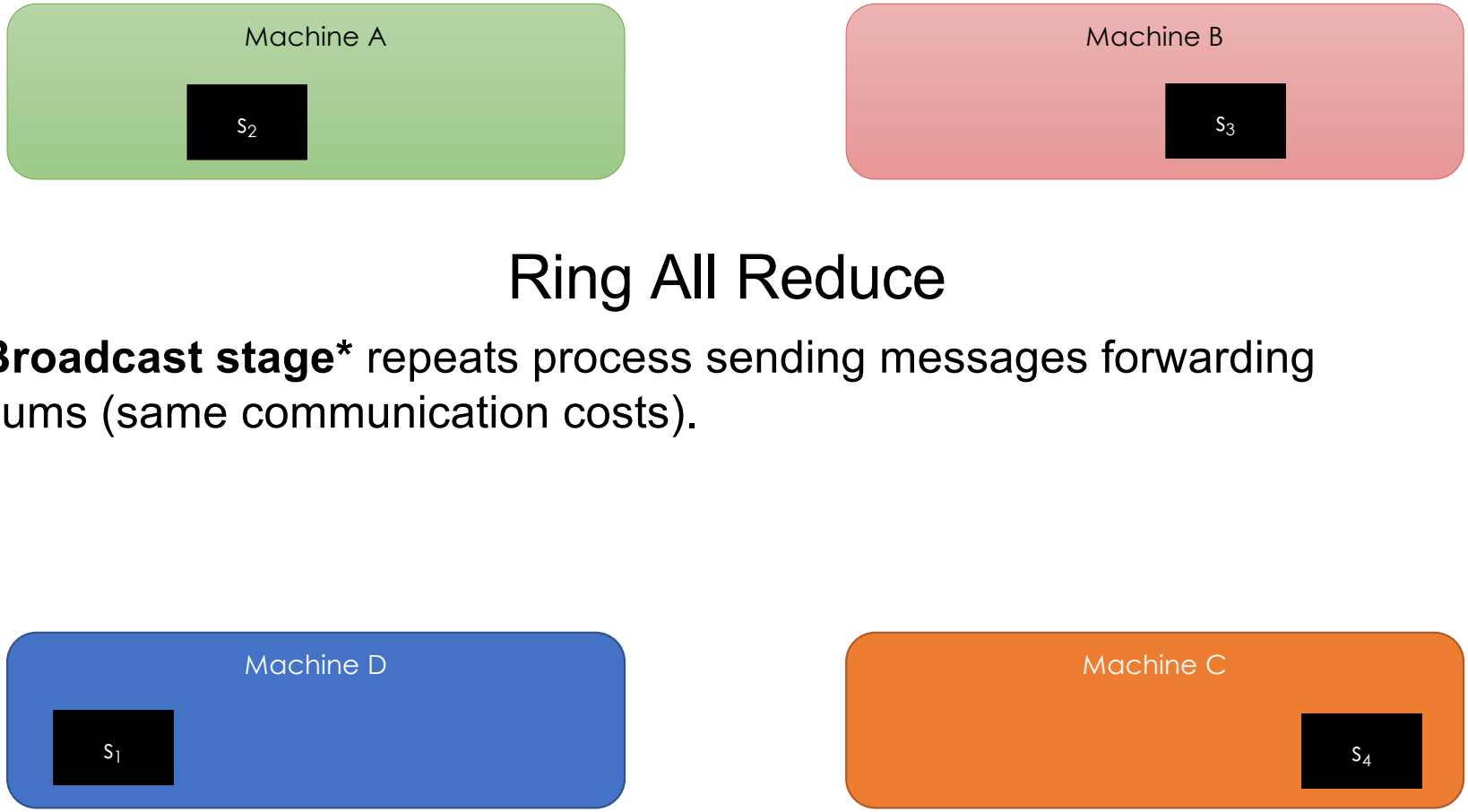




Ring All Reduce

Each machine sends N/P data to next machine each of (p-1) rounds:
(P-1) * N/P (doesn't depend on P!)

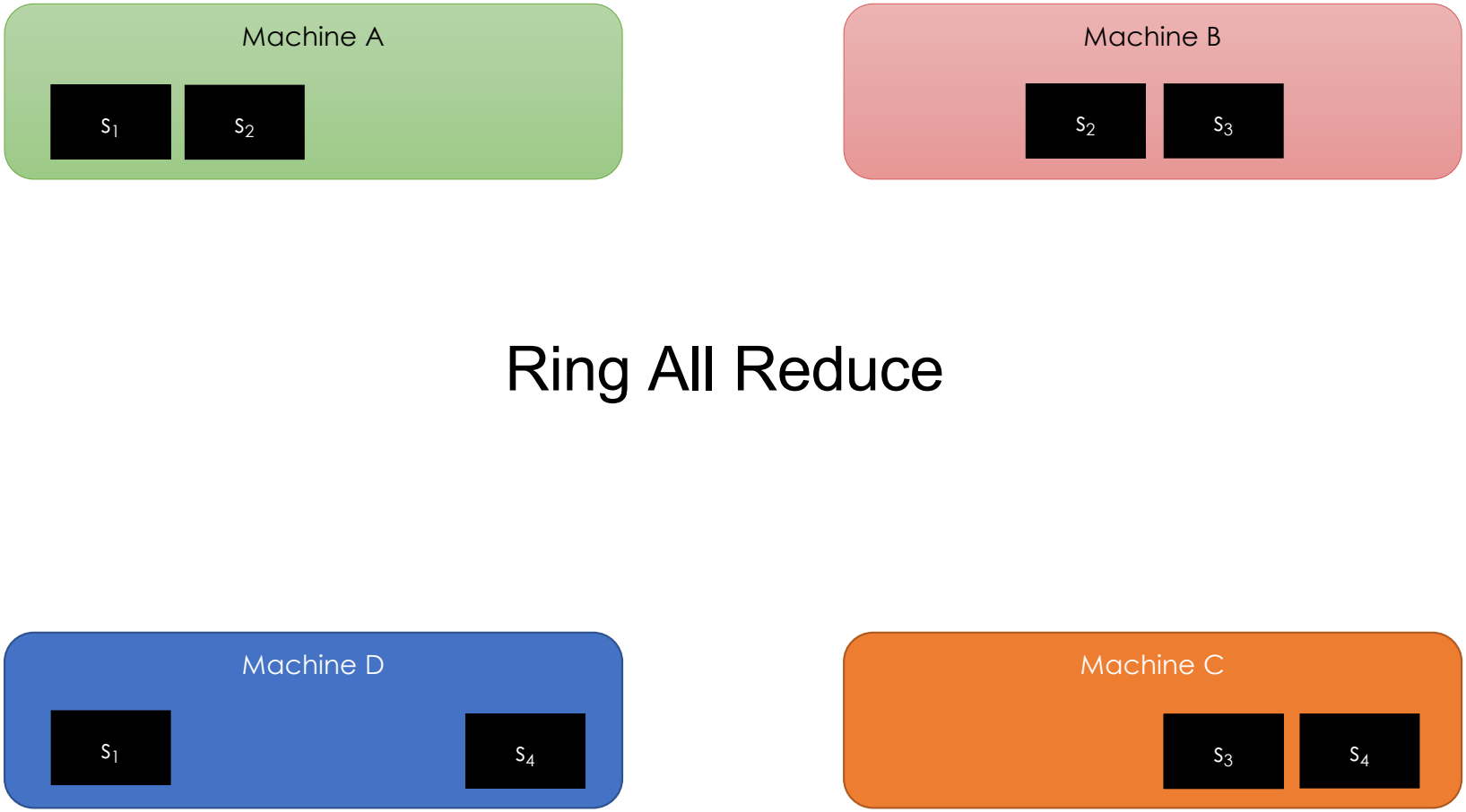
- **Fan-in Per Round:**
  - **1** (doesn't depend on P)

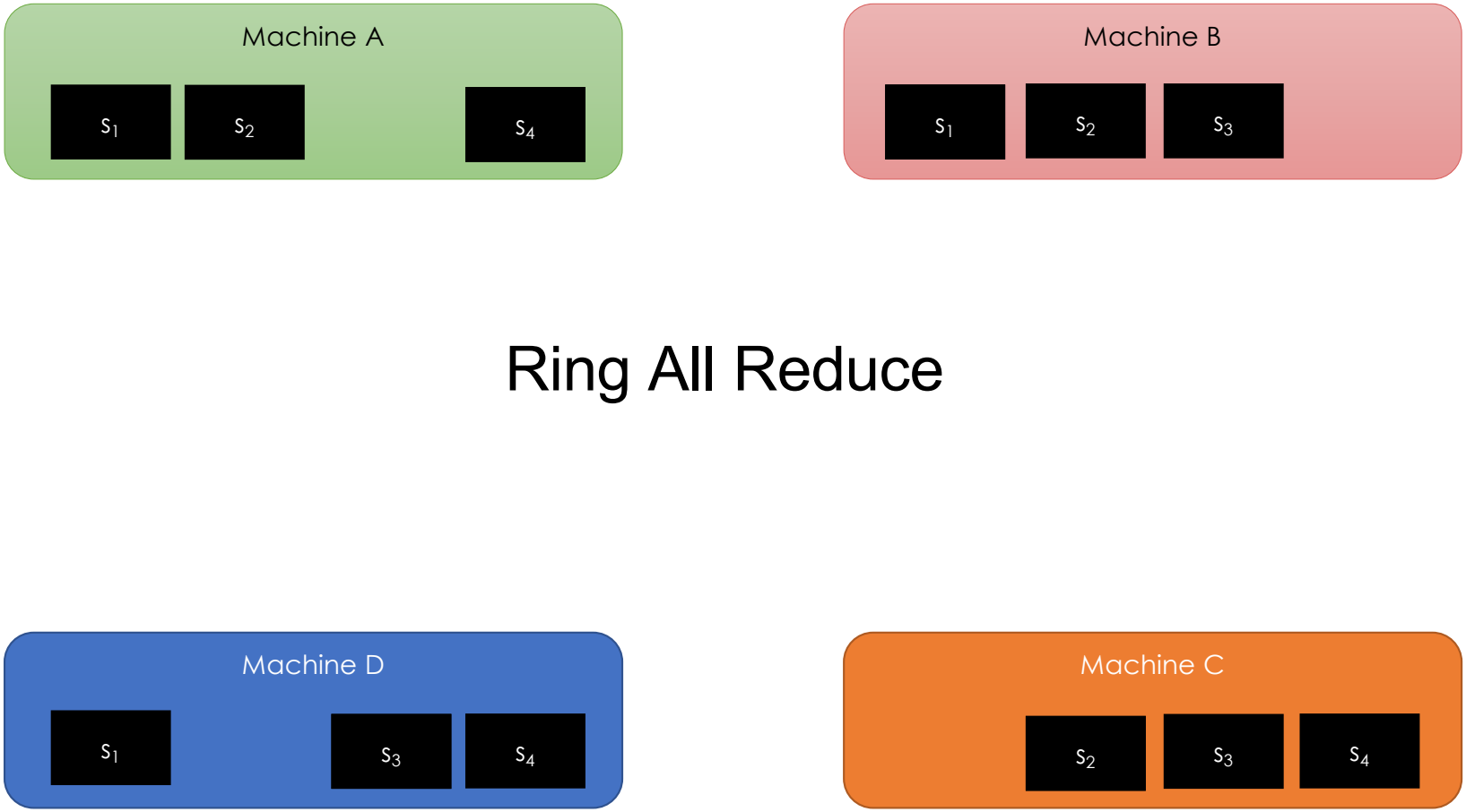


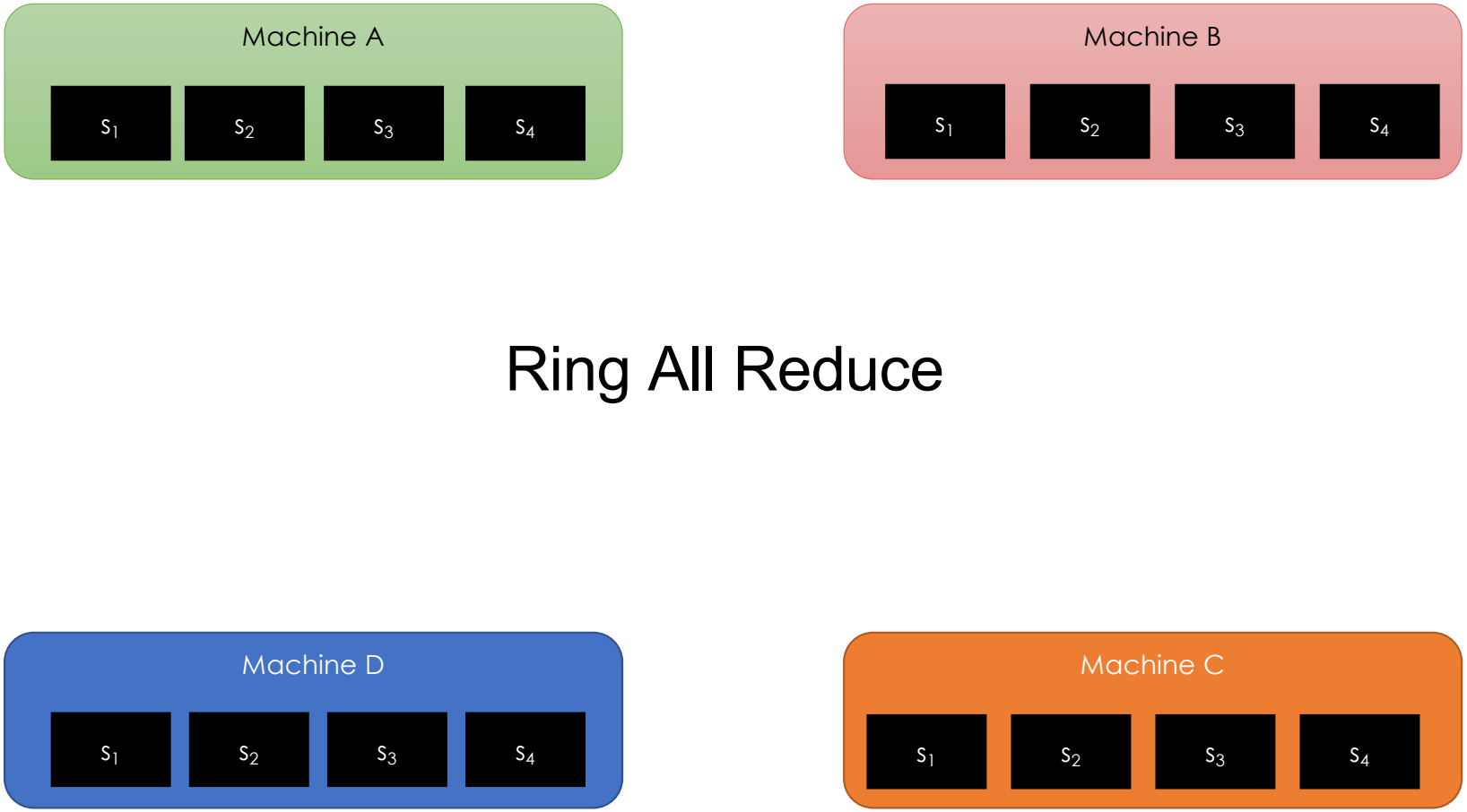
Ring All Reduce

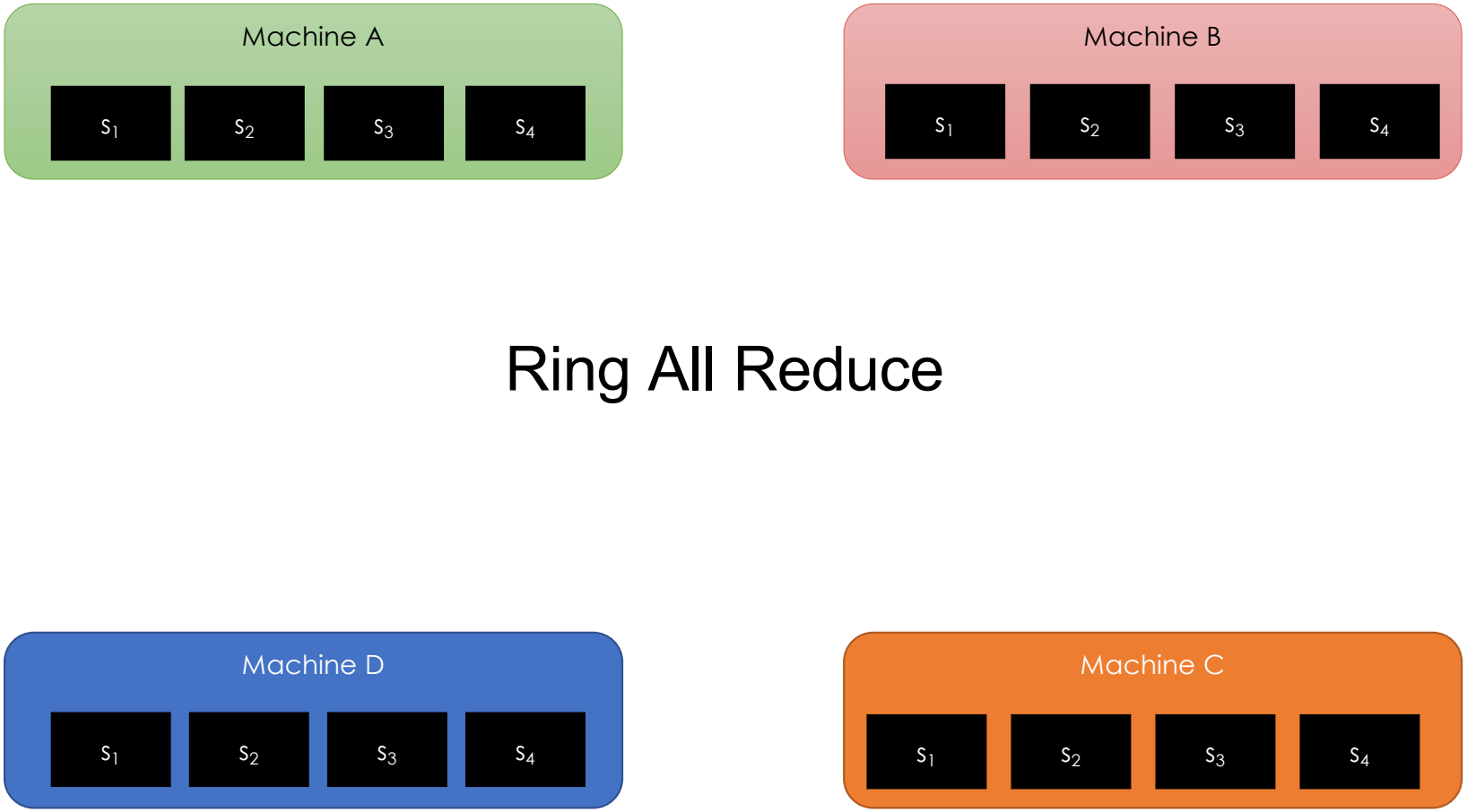
Broadcast stage* repeats process sending messages forwarding sums (same communication costs).

*** Technically All Gather based on MPI communication definition**


Machine A
s_1
s_2
Machine B
s_2
s_3
Ring All Reduce
Machine D
s_1
s_4
Machine C
s_3
s_4

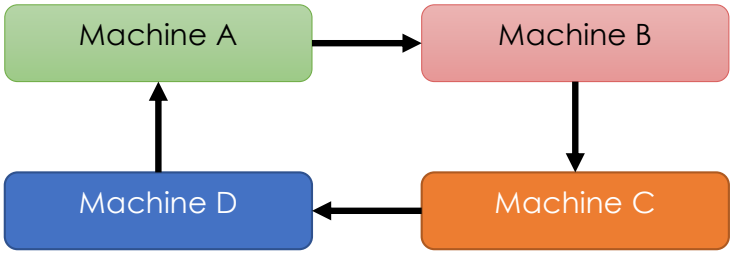

Machine A
s_1
s_2
s_4
Machine B
s_1
s_2
s_3
Ring All Reduce
Machine D
s_1
s_3
s_4
Machine C
s_2
s_3
s_4


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


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

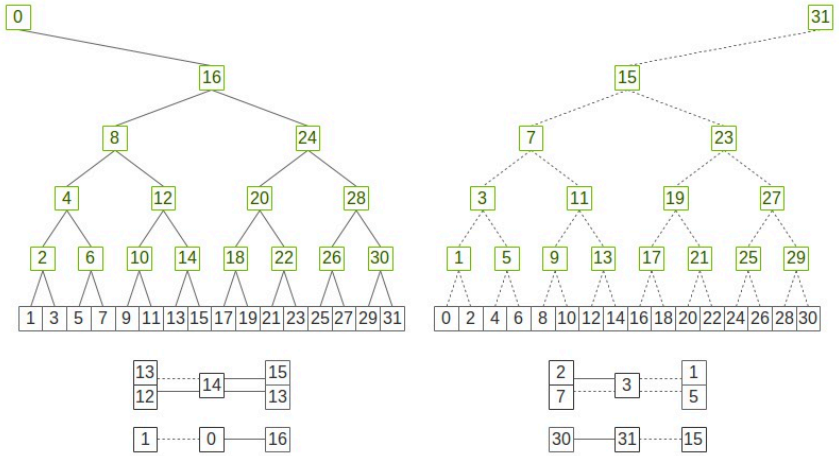
- Simplified communication topology with low fan-in

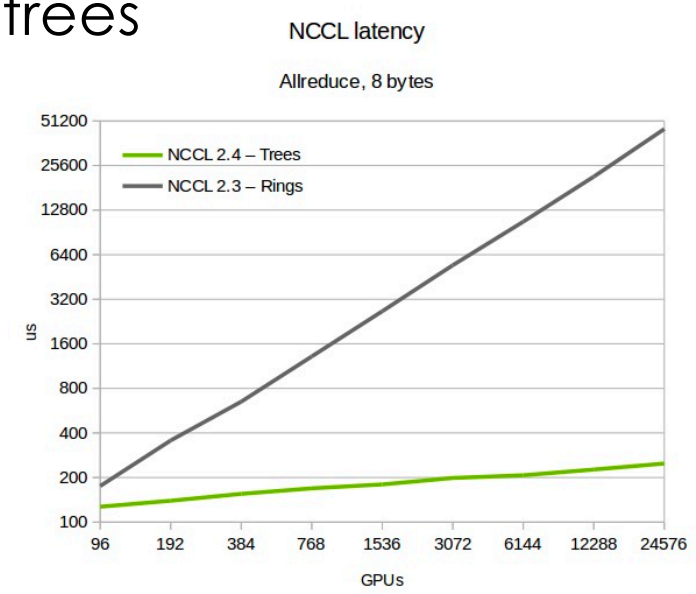


- Overall communication
  - Same total communication: **2*(P-1)*N**, but evenly distributed
  - Each Machine communicates 2*(P-1)N/P (almost independent of P)
  - **Fan-in** is constant (doesn't depend on P)
- **Issue:** Number of communication rounds (P-1)

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

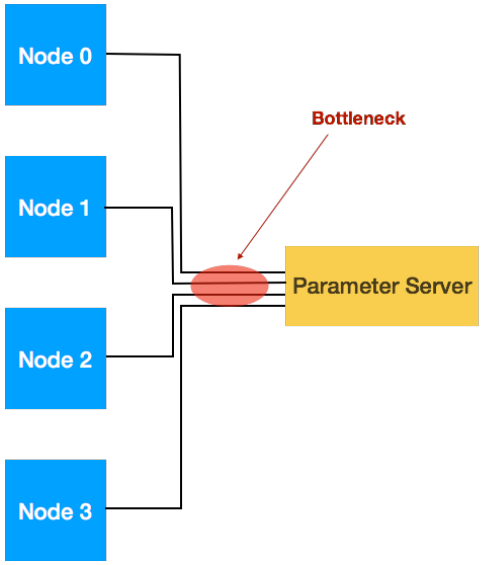
https://devblogs.nvidia.com/massively-scale-deep-learning-training-nccl-2-4/

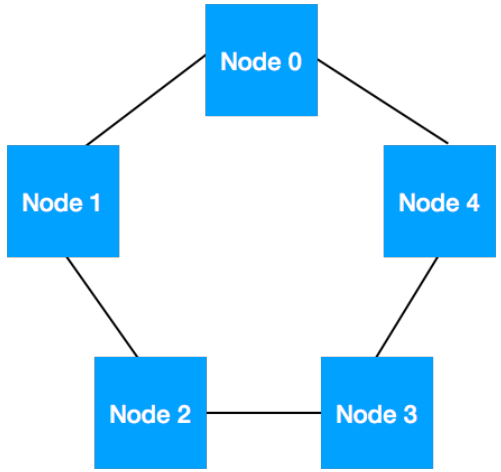
Complexity Summary

$$T_{comm} = (\alpha + PN\beta)$$

$$T_{comm} = 2((P-1)\alpha + \frac{P-1}{P}N\beta)$$

α latency
β bandwidth
N message size
P #processes





Parameter Server

Ring All-reduce

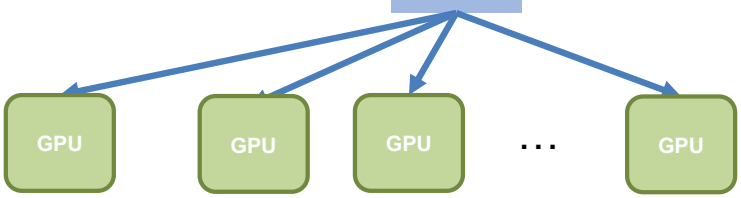
Great Reference: T. Rajeev, R. Rabenseifner, and W. Gropp. "Optimization of collective communication operations in MPICH." ***The International Journal of High Performance Computing Applications,*** **2005.**

Data Parallel Training Complexity Analysis

- Question: Comm time of ring allreduce is independent of the number of processors. So what limits scalability?

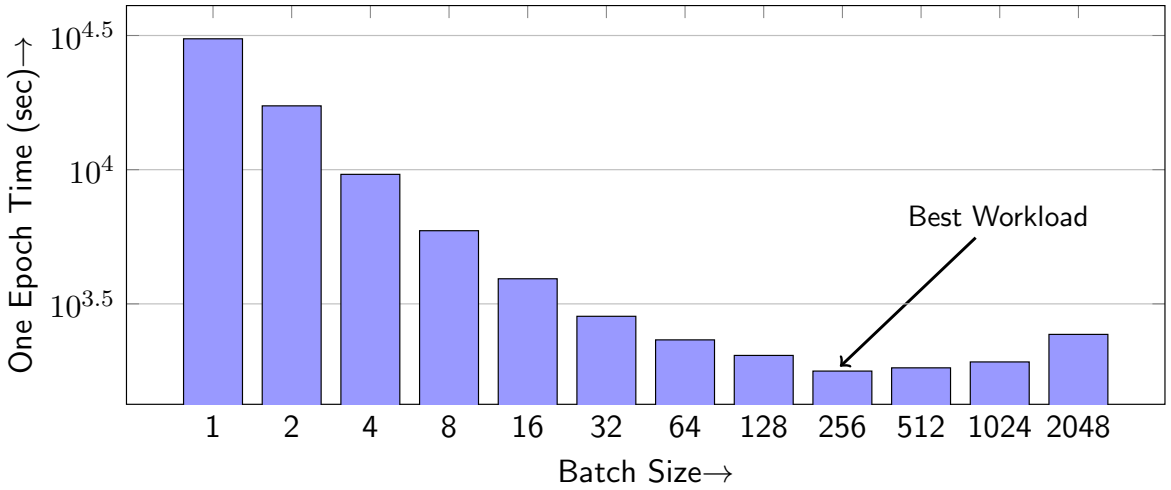
$$T_{comm}(batch) = 2\sum_{i=0}^{L}\left(\alpha(P-1) + \beta\frac{P-1}{P}|W_i|\right)$$

$$w^1 = w^0 - \frac{\alpha}{B}\sum_{i=1}^{B}\frac{\partial\mathcal{J}(w^0)}{\partial w}$$



Limits of Data Parallel Scaling

- The maximum limit of processors that you can use is P=B
- But this often leads to very low utilization of the hardware and would not yield any speed up

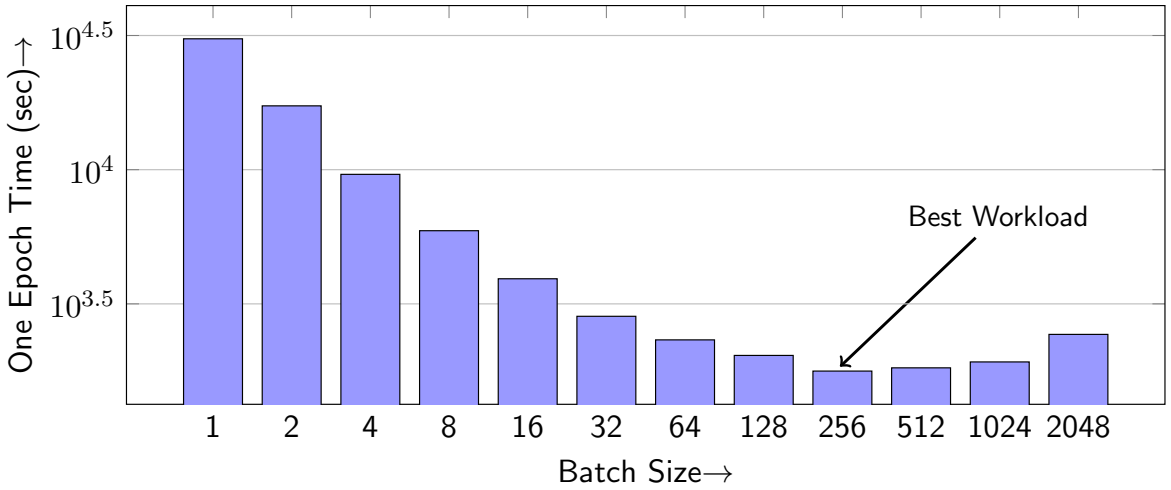


One epoch training time of AlexNet computed on an Intel KNL system

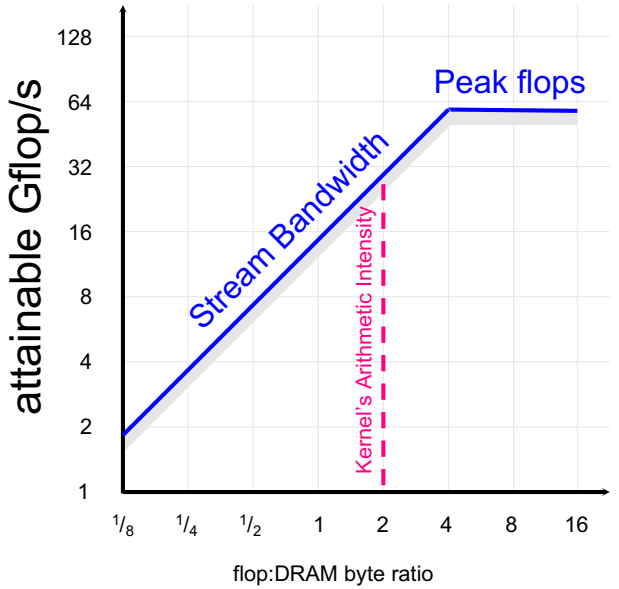
- Why does this happen?
  - Remember roofline model?

Limits of Data Parallel Scaling

- The maximum limit of processors that you can use is P=B
- But this often leads to very low utilization of the hardware and would not yield any speed up

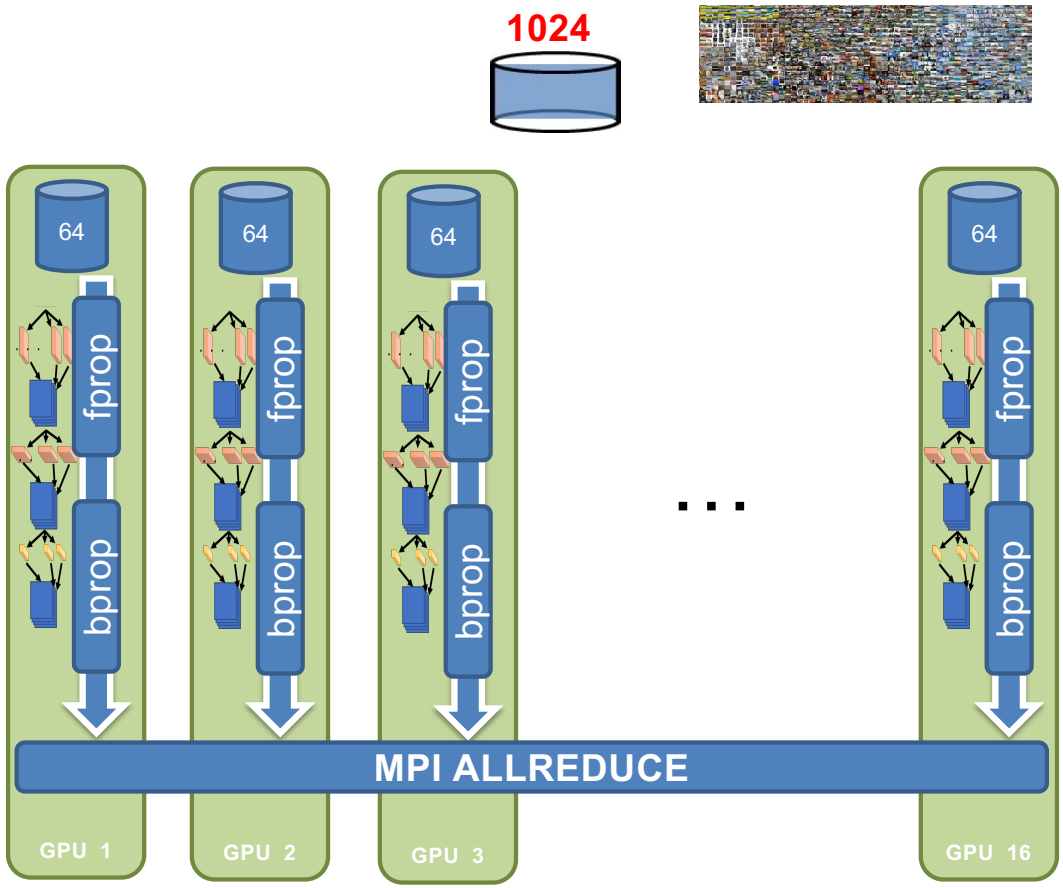


One epoch training time of AlexNet computed on an Intel KNL system

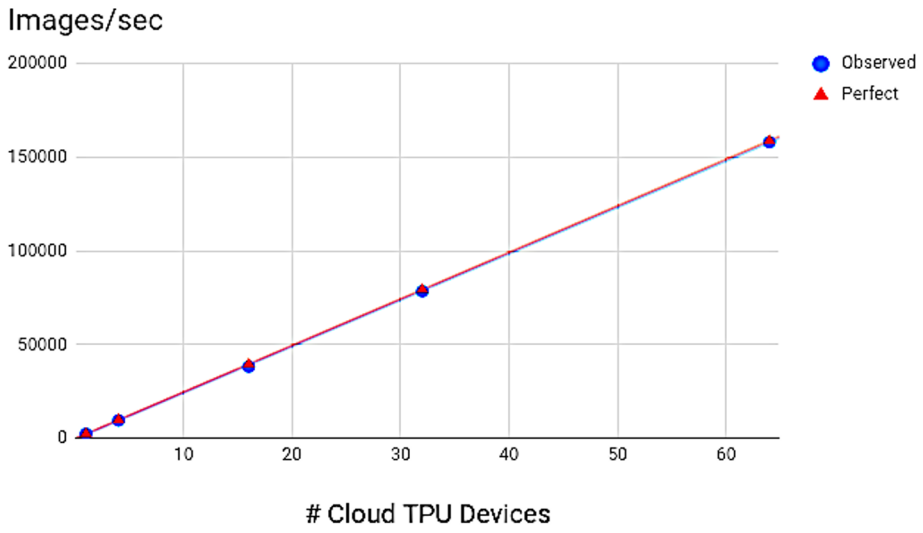


Scaling Data Parallel Training

If we want to keep scaling synchronous SGD then we have to keep increasing the batch size.



Naively increasing Batch size leads to perfect results but …



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

Convergence
Machine Learning
Property

Throughput
System
Property

Bigger isn't Always Better

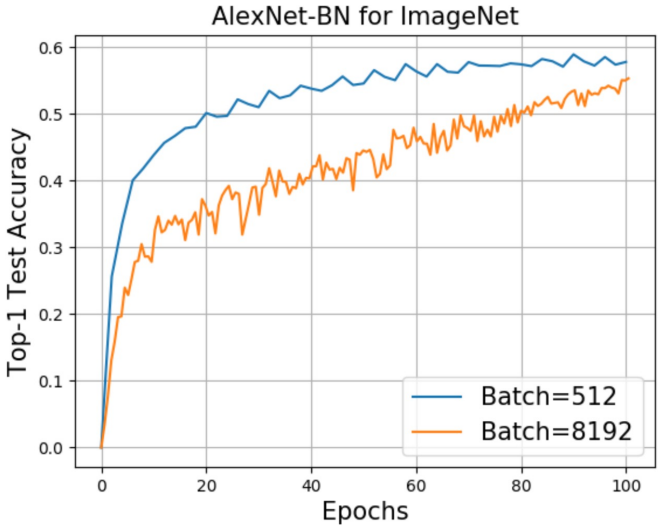
- Motivation for larger batch sizes
  - More opportunities for parallelism → but is it useful?
  - Recall (1/n variance reduction):

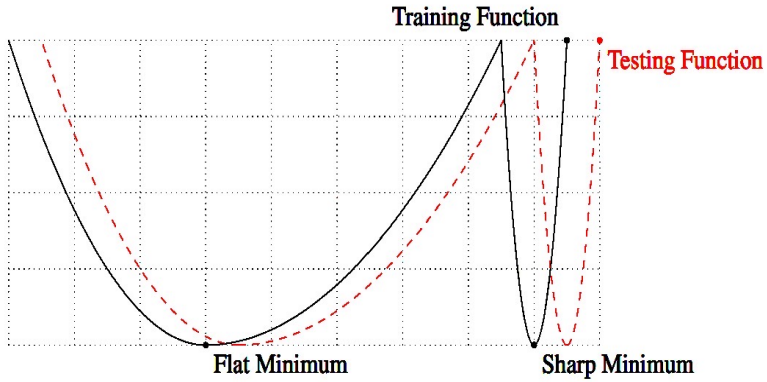
$$\frac{1}{n}\sum_{i=1}\nabla_\theta \mathbf{L}(y_i, f(x_i;\theta)) \approx \frac{1}{|\mathcal{B}|}\sum_{i\in\mathcal{B}}\nabla_\theta \mathbf{L}(y_i, f(x_i;\theta))$$

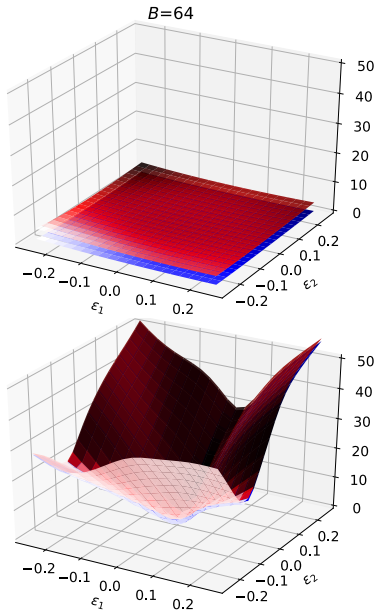
- Is a variance reduction helpful?
  - Only if it let's you take bigger steps (move faster)
  - Does it affect the final prediction accuracy?

Problems with Large Batch Training

- Larger Batch leads to sub-optimal generalization
- A common belief is that large batch training gets attracted to "sharp minimas"

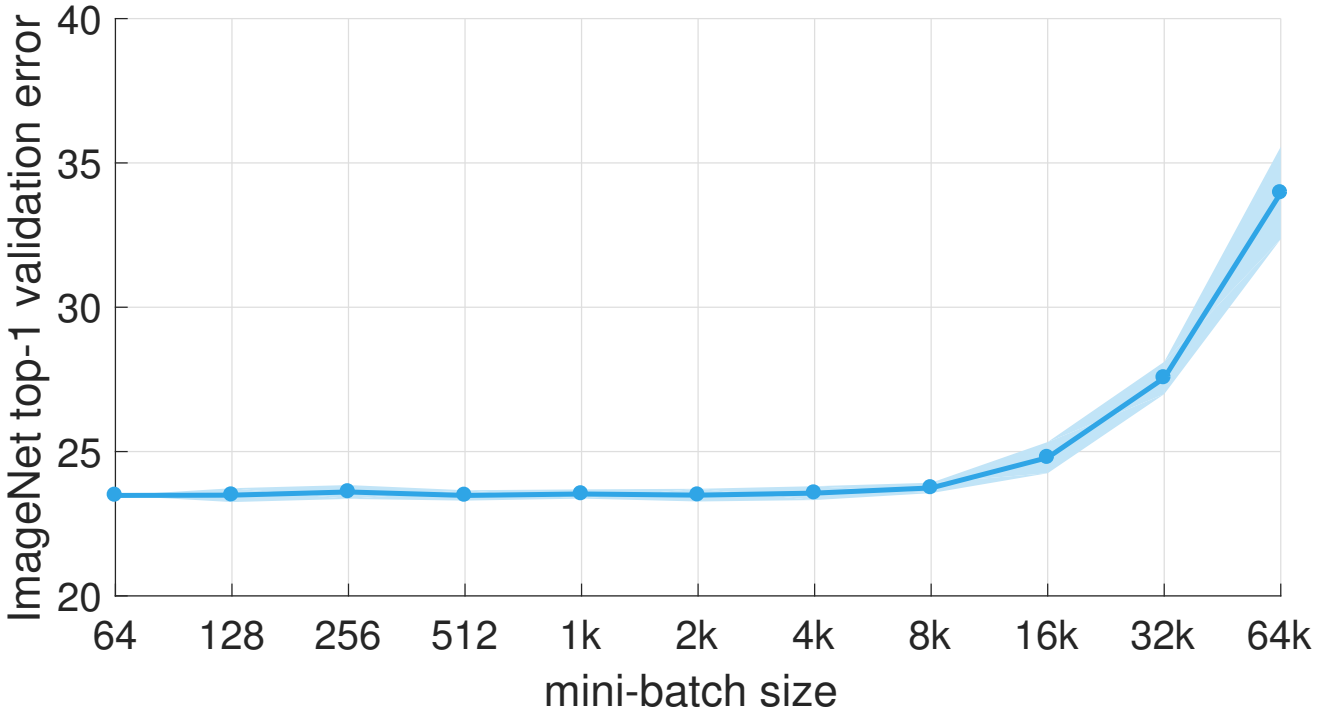






Keskar et al., On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima, ICLR'16.
Z. Yao, A. Gholami, Q. Lei, K. Keutzer, M. Mahoney. Hessian-based Analysis of Large Batch Training and Robustness to Adversaries, NeurIPS'18.
Ginsburg, Boris, Igor Gitman, and Yang You. "Large Batch Training of Convolutional Networks with LARS." arXiv:1708.03888, 2018.

Generalization Gap Problem



Larger batch sizes harm generalization performance.

Why? Large Batch Reduces Noise and may Get Trapped in Local Minima

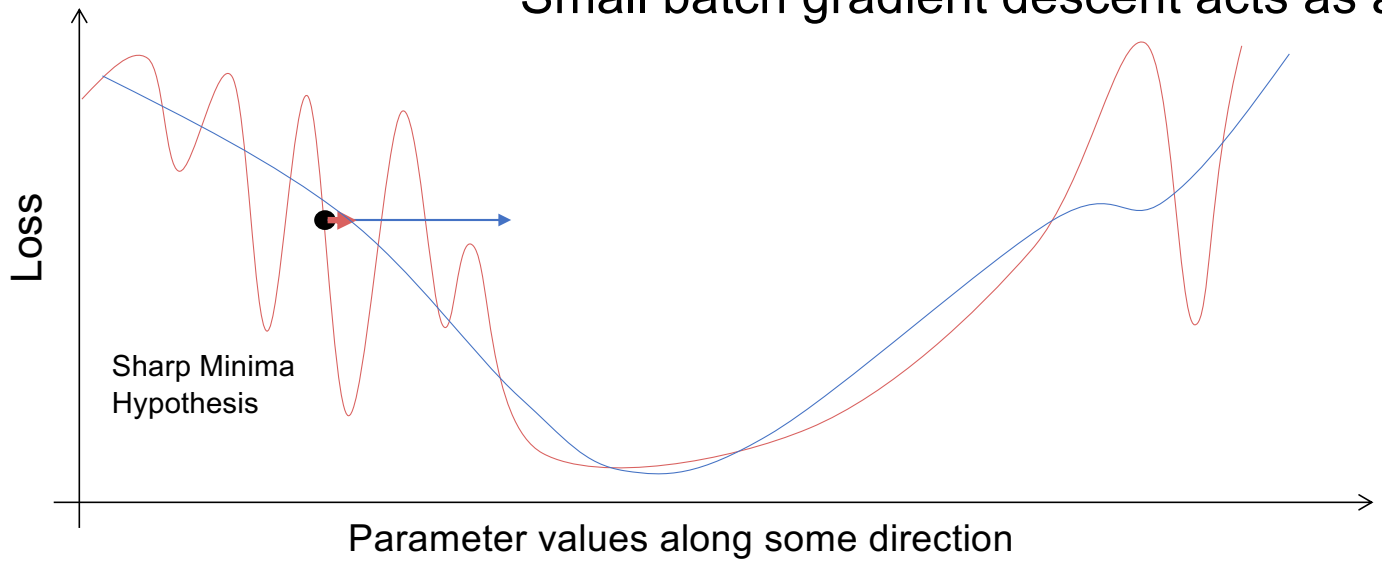
Objective function

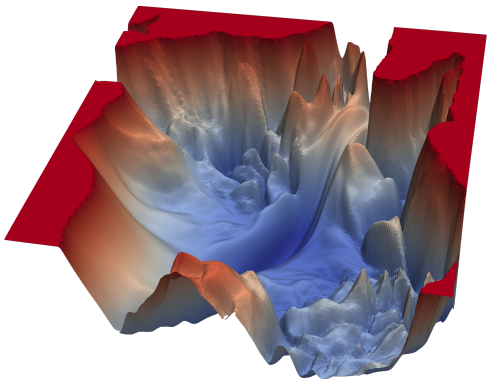
$$L(\theta) = \frac{1}{N}\sum_{i=1}^{N} l(x_i, y_i, \theta)$$

Update rule

$$\theta_{t+1} = \theta_t - \eta_t \frac{1}{|B|} \sum_{(x,y)\in B} \nabla_\theta l(x, y, \theta_t)$$

Small batch gradient descent acts as a **regularizer**





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

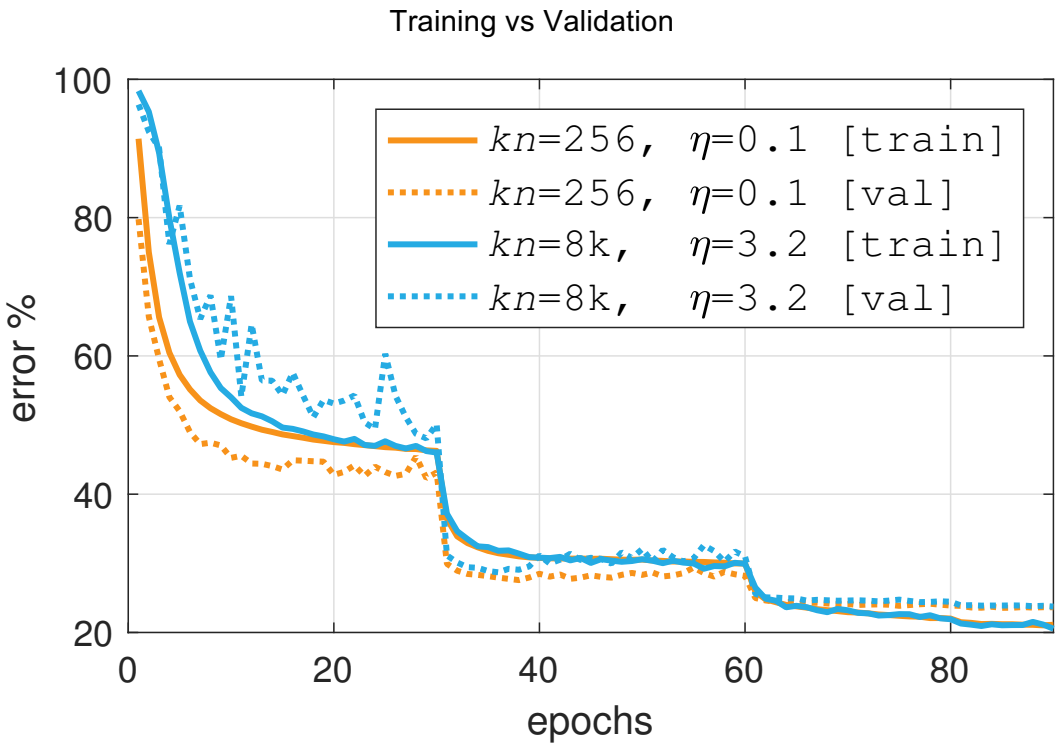
Solution: Linear Scaling Rule

- Scale the learning rate linearly with the batch size

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

- 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…

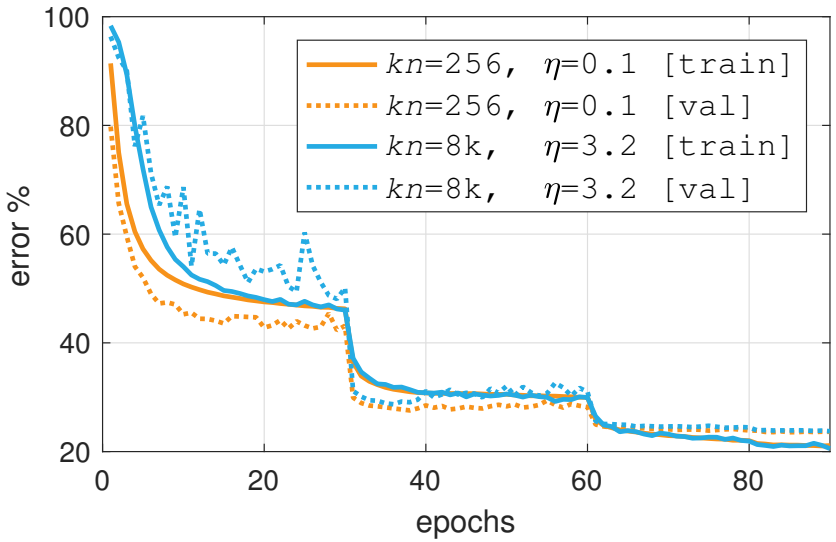
Key Results

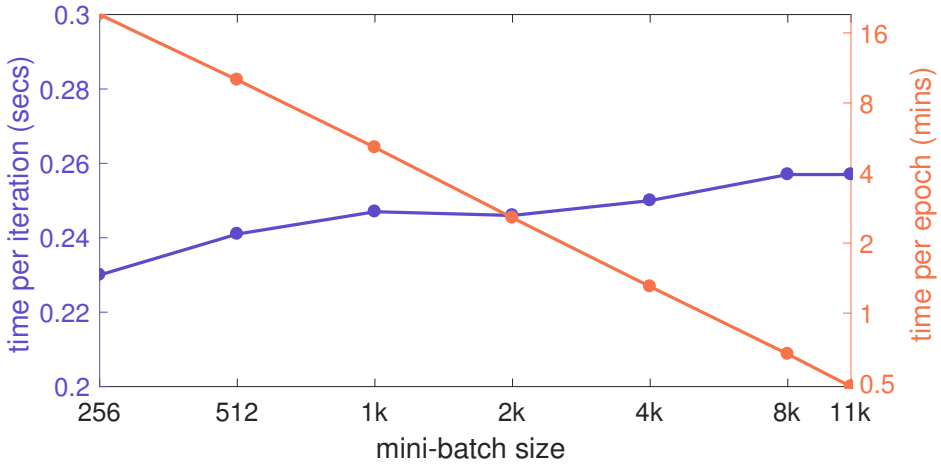


All curves closely match using the linear scaling rule.

Note learning rate schedule drops.

Key Results





$$\frac{\text{“Learning”}}{\text{Epoch}}$$

Machine Learning

$$\frac{\text{Epoch}}{\text{Second}}$$

System

Key Results

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

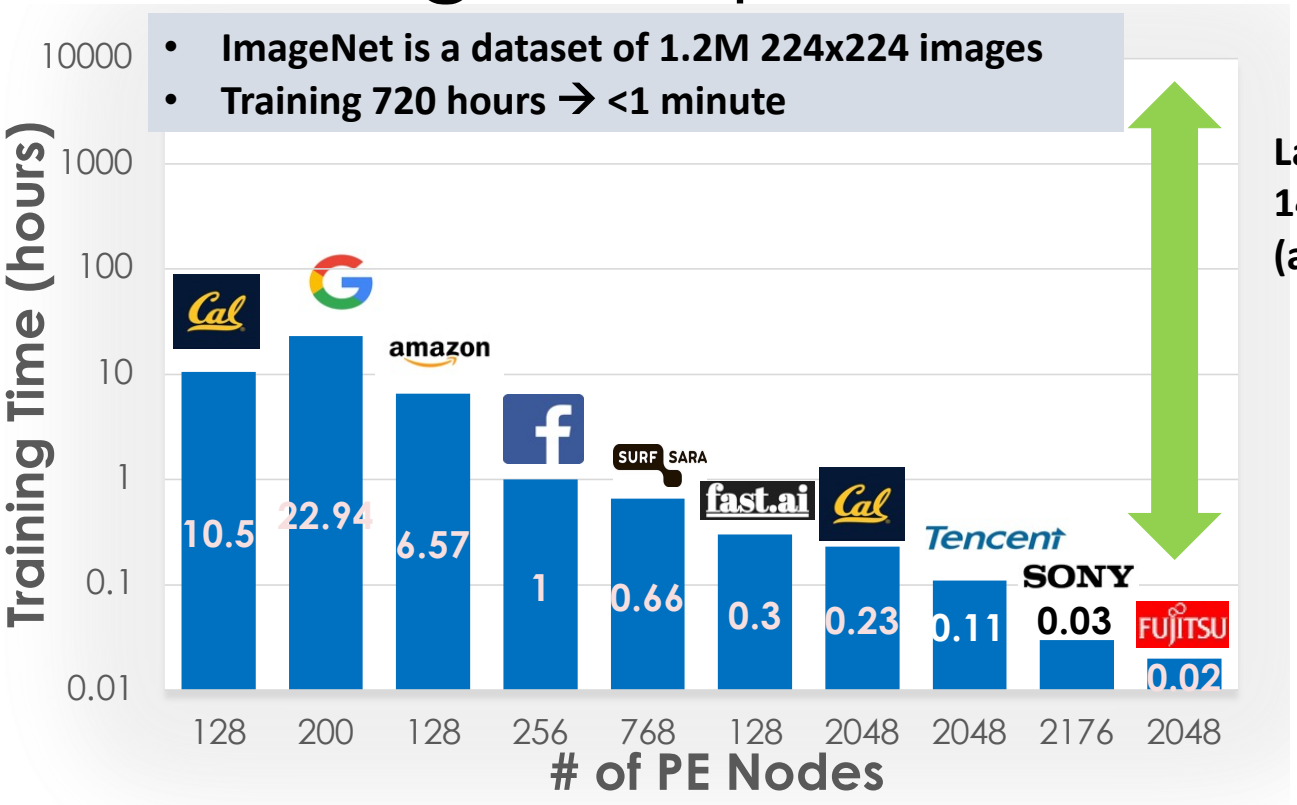
- Fairly careful study of the linear scaling rule
  - Observed limits to linear scaling do not depend on dataset size
  - But what is the limit?
    - You cannot indefinitely scale the learning rate …

Since then there has been a race to train ImageNet faster and several new large batch training methods have been developed (some with good foundation and some heuristics)

ImageNet Training Competition!

Iandola

Yang You



Latest record:
14 seconds / TPUv4
(as of Feb 2022)

- Iandola FN, Moskewicz MW, Ashraf K, Keutzer K. **FireCaffe**: near-linear acceleration of deep neural network training on compute clusters. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition 2016 (pp. 2592-2600).
- You Y, Zhang Z, Hsieh CJ, Demmel J, Keutzer K. **Imagenet training in minutes**. In Proceedings of the 47th International Conference on Parallel Processing 2018 Aug 13 (p. 1) ACM **(Best Paper Award)**

Very active area of research

Other papers to read if you are interested:

- Golmant, Noah, et al. "On the computational inefficiency of large batch sizes for stochastic gradient descent" **(Cal)**
- Shallue et al. "Measuring the Effects of Data Parallelism on Neural Network Training" **(Google)**
- You, Yang, et al. "Large batch optimization for deep learning: Training bert in 76 minutes." **(Cal)**

Next week's readings

Reading for Next Week

- Chimera: Efficiently Training Large-Scale Neural Networks with Bidirectional Pipelines [SC'21, Best Student Paper finalist]
  - A novel technique for pipeline parallel training with bidirectional computational flow to reduce the "bubble size".
- Efficient Large-Scale Language Model Training on GPU Clusters Using Megatron-LM [SC'21, Best Student Paper]
  - Large scale deployment of data, model, and pipeline parallelism to scale training of a 1T parameter transformer to 3K+ GPUs
- ZeRO-Infinity: Breaking the GPU Memory Wall for Extreme Scale Deep Learning [SC'21]
  - A novel method for increasing the maximum size of the model that can be trained on a GPU by leveraging NVMe.

Extra Suggested Reading

- DeepSpeed: Advancing MoE inference and training to power next-generation AI scale [Blog post]
- Large Scale Distributed Deep Networks [NeurIPS'12]
  - One of the first papers using (known) techniques applied to training large ML models at Google
- Gpipe: Efficient training of giant neural networks using pipeline parallelism [NeurIPS'19]
  - A micro-batching technique used for pipeline parallelism to reduce "bubble size" with synchronous SGD
- PipeDream: Fast and Efficient Pipeline Parallel DNN Training [SOSP'19]
  - Proposed an asynchronous method for reducing the "bubble size" of pipeline parallel training