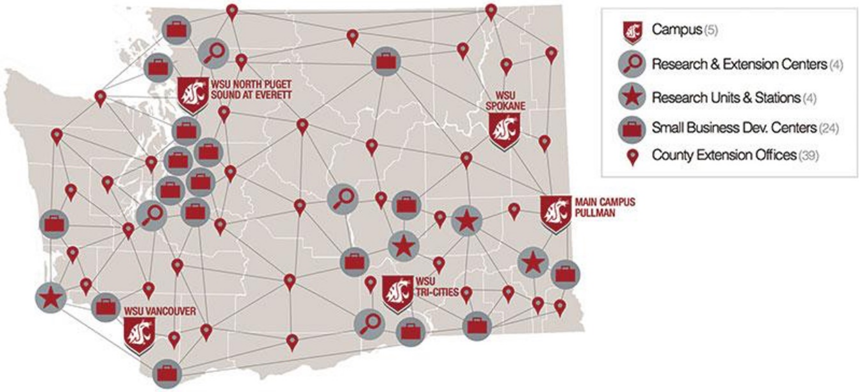
Introduction to ML Systems

2022 OxML Summer School – ML Fundamentals

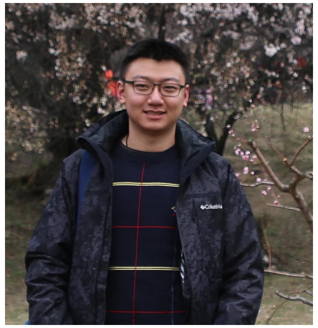
Dingwen Tao
Washington State University



Graduate Students





Research topics (not limited to):

- Big data management, analytics, visualization
- Large-scale machine/deep learning
- Heterogeneous computing (GPU/FPGA)
- Fault tolerance and resilience at extreme scale
- Energy-efficient computing
- Numerical algorithms, simulation & software

Undergraduate Students



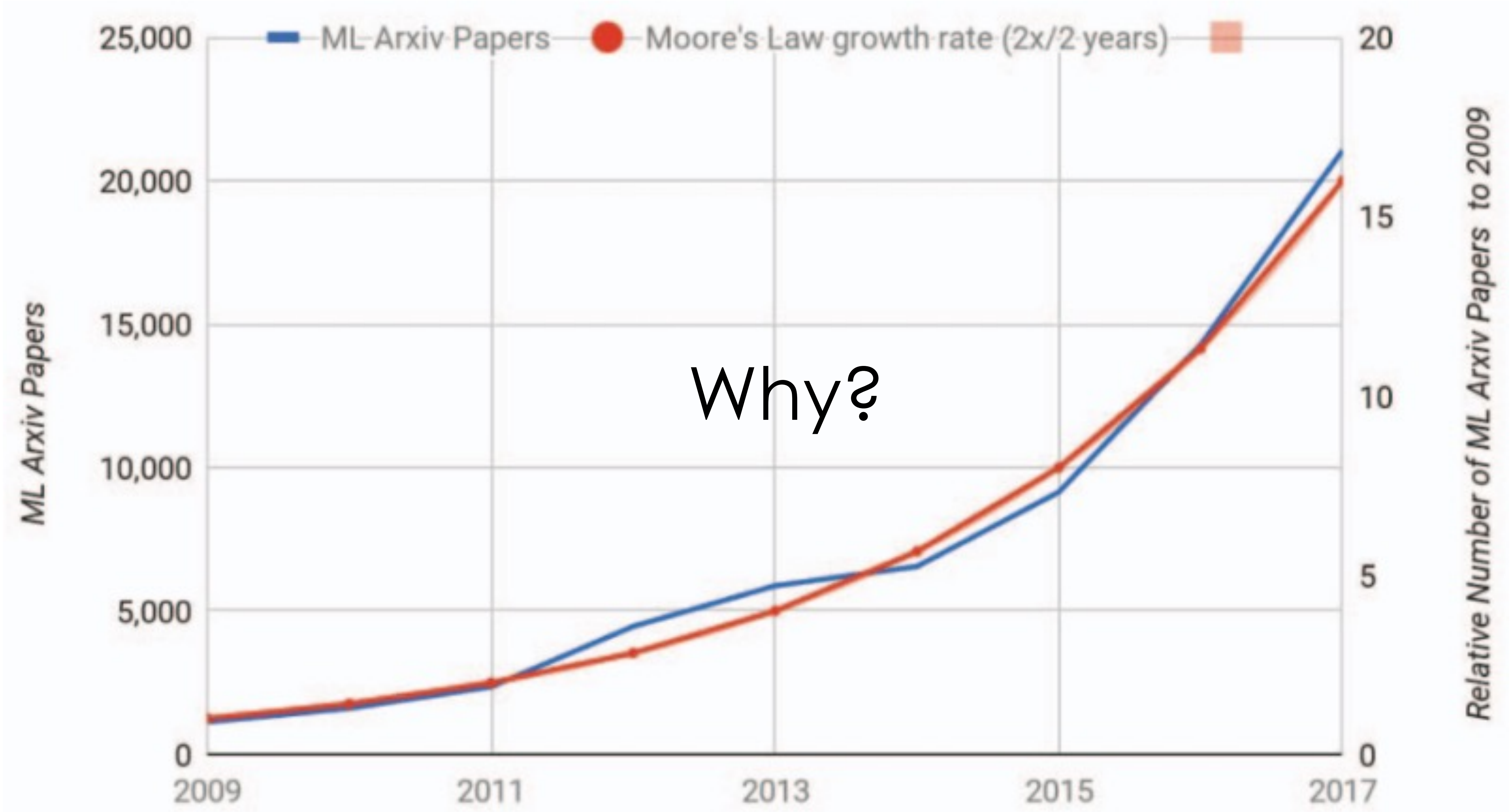



Thank You!

Today's Agenda

➢ Introduction to ML + Systems	14:00 – 14:10
➢ Key Trends in Hardware for ML	14:10 – 14:25
➢ Data Parallel Training & Its Challenges	14:25 – 15:15
➢ Break	15:15 – 15:30
➢ Pipeline Parallelism	15:30 – 15:45
➢ Model Parallelism	15:45 – 16:15
➢ Spatial Parallelism	16:15 – 16:25
➢ Summary & Close	16:25 – 16:30

R&D in ML and Systems is Exploding



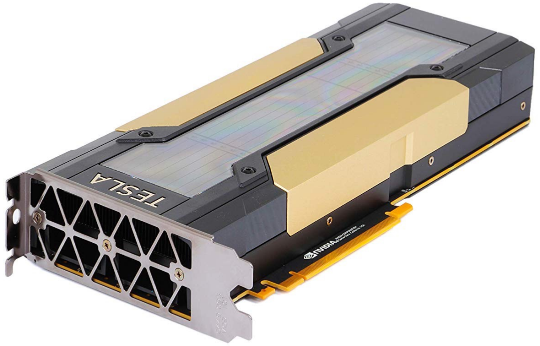
"A New Golden Age in Computer Architecture: Empowering the Machine-Learning Revolution",
https://ieeexplore.ieee.org/document/8259424

New Forces Driving AI Revolution

Data

Benchmarks

Compute



Abstractions

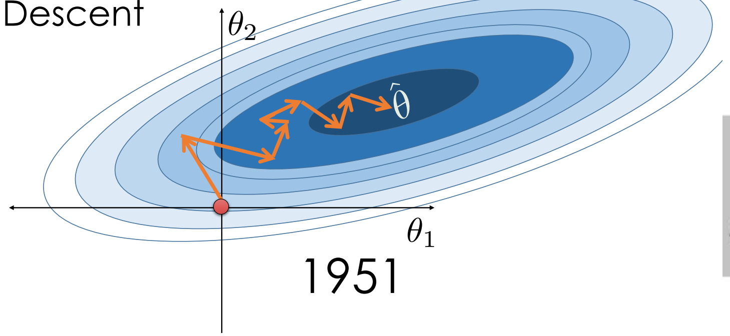
PyTorch

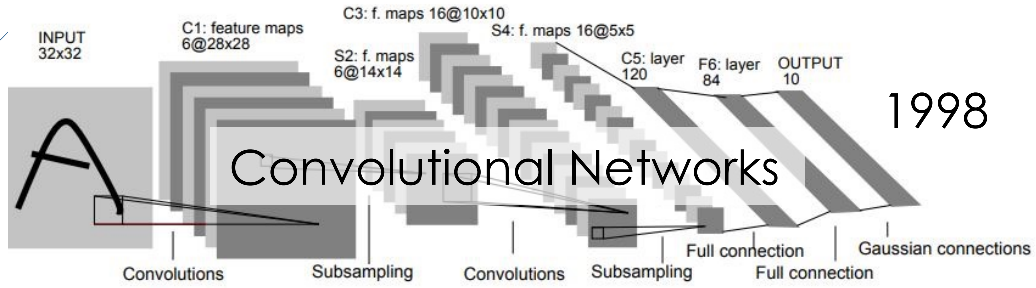
Caffe

TensorFlow

Advances in Algorithms and Models

Stochastic Gradient Descent





1998

Machine learning community has had an evolving focus on AI Systems

Fast Algorithms

Distributed Algorithms

Deep Learning Frameworks

Transformers Everywhere

2009 → 2022

ML for Systems

Machine Learning Frameworks

RL for Systems

Massive General Models

→ Integration of Communities

What defines good ML-Systems Research Today?

What is AI-Systems Research?

- Good AI and Systems research
  - Provides **insights to both communities**
  - **Builds on big ideas** in prior AI and Systems Research
- Leverages understanding of both domains
  - Studies **statistical and computational tradeoffs**
  - Identify **essential abstractions** to bridge AI and Systems
  - Reframes **systems problems as learning problems**
- More than just great open-source software!
  - But software impact often matters…

Kinds of AI-Systems Research

AI + Systems

Advances in **systems** are enabling substantial progress in **AI**

AI + Systems

Developing Systems for:

- Autonomous Vehicles
- Reinforcement Learning
- Secure Machine Learning
- Prediction Serving
- Experiment Management

Advancing AI

- Dynamic Neural Nets
- Prediction on Compressed Data
- Distributed Training
- Distributed Auto-ML

Advances in **AI** are being used to address fundamental challenges in **systems**.

AI + Systems

AI + Systems

- Reinforcement Learning for
  - Pandas code generation
  - SQL join planning
  - Network packet classification
  - Autoscaling
- Bandit Algorithms for radio link adaptation
- Wireless link quality estimation
- Multi-task learning for straggler mitigation
- VM Selection using Trees ..

Hardware for ML

Key Drivers for Neural Network Success

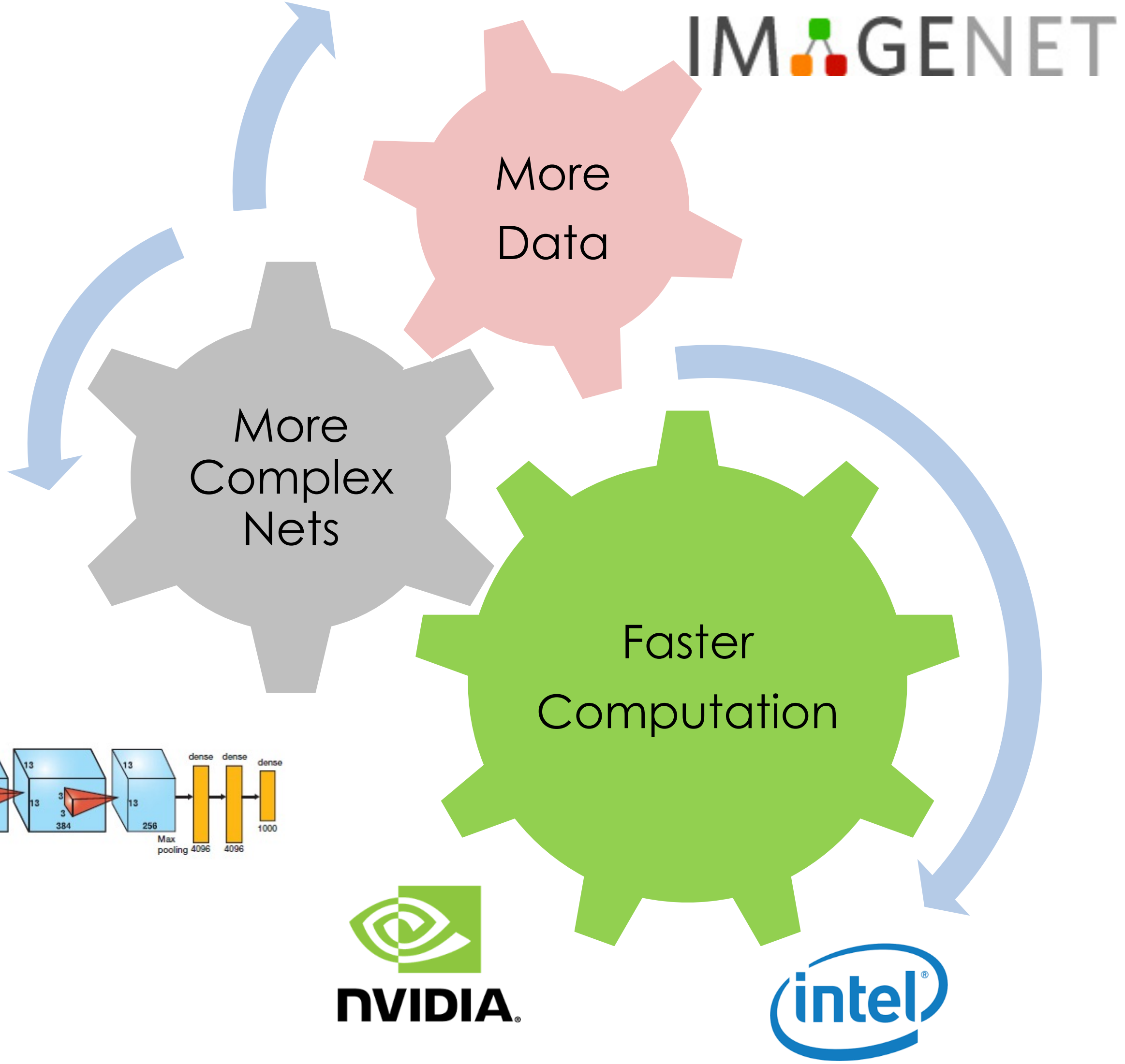
DARPA Neural Network Study Final Report (606 pages):

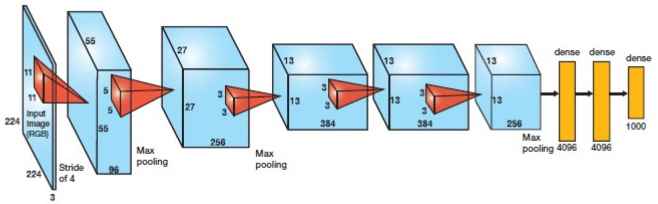
"After participating in this Study, my personal view is that **neural networks will provide the next major advance in computing technology."**

Dr. Jasper Lupo

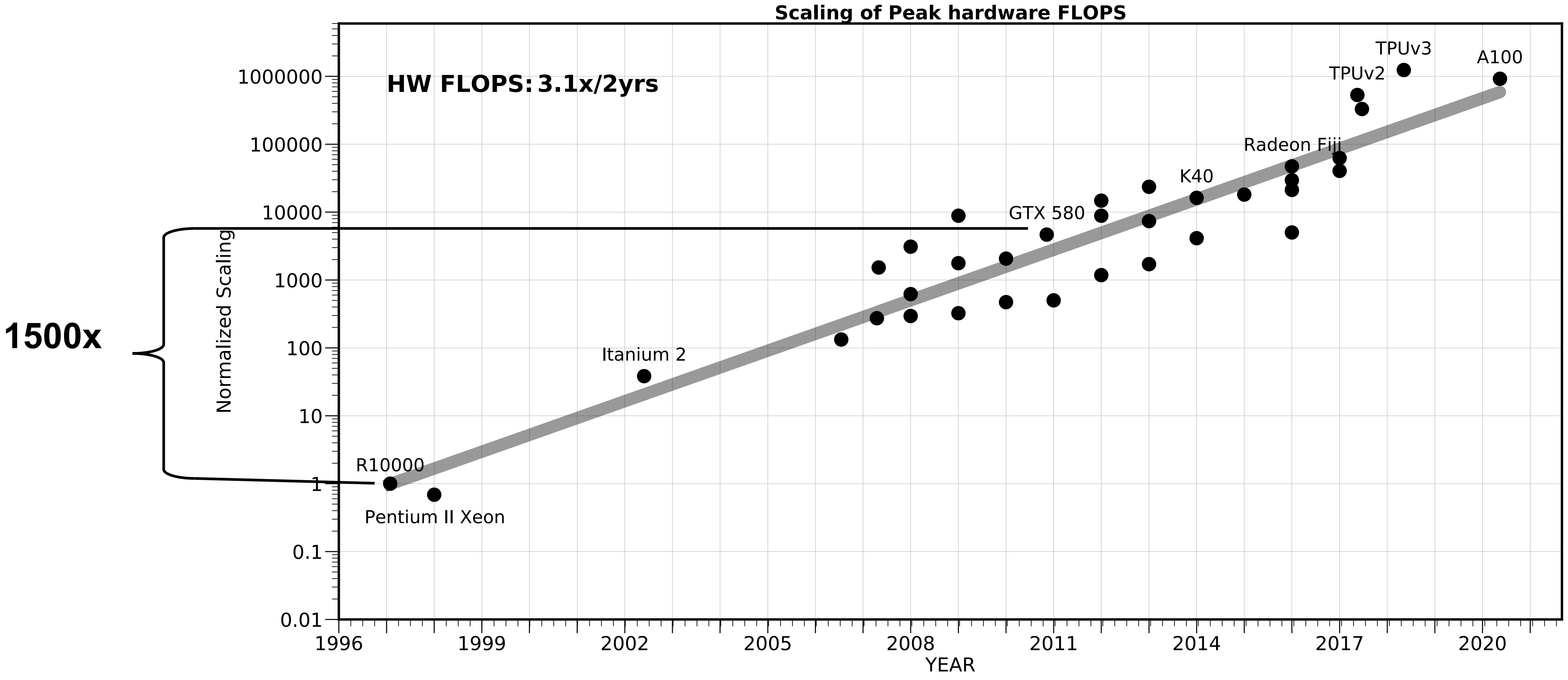
DARPA, Washington, DC

June, **1988**



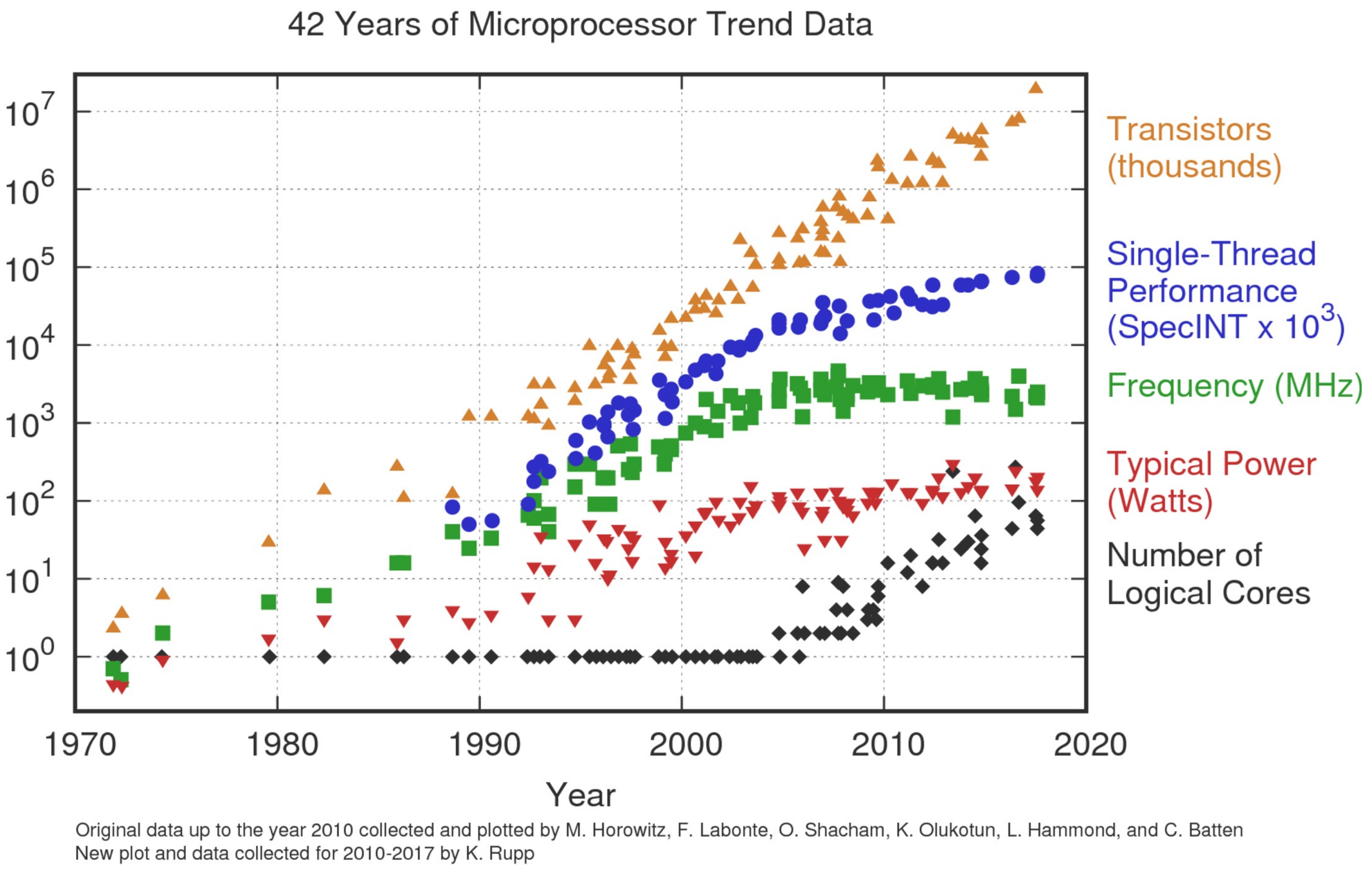


AlexNet vs Lenet5: 1000x More Compute



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

General Purpose Hardware Trend

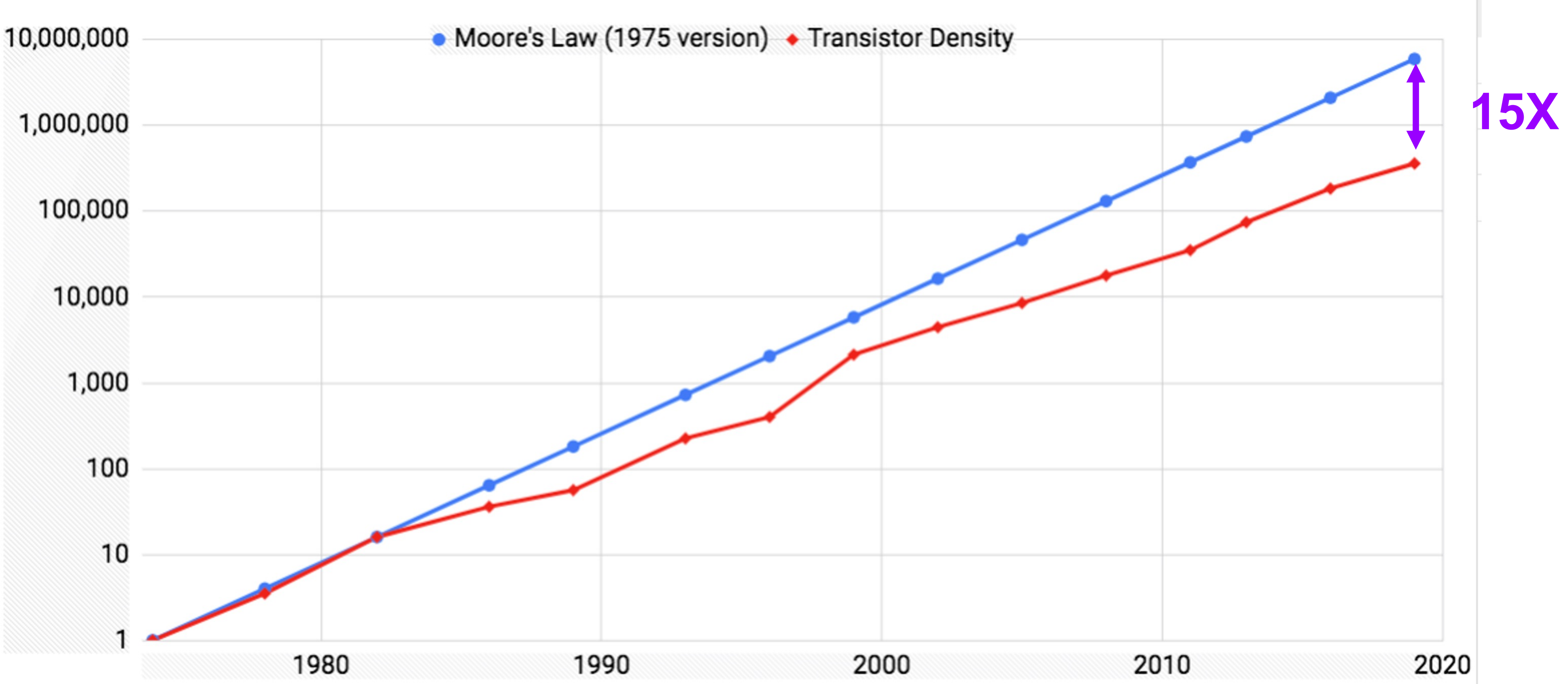


[42 Years of Microprocessor Trend Data](), Karl Rupp

Key Observations

- # Transistors still increasing
- Single Core Performance Plateauing
- End of Dennard Scaling
- Distributed Computing

Common Fallacy: Moore's Law is Dead (it's not)



Moore, Gordon E. "No exponential is forever: but 'Forever' can be delayed!" *Solid-State Circuits Conference, 2003.*

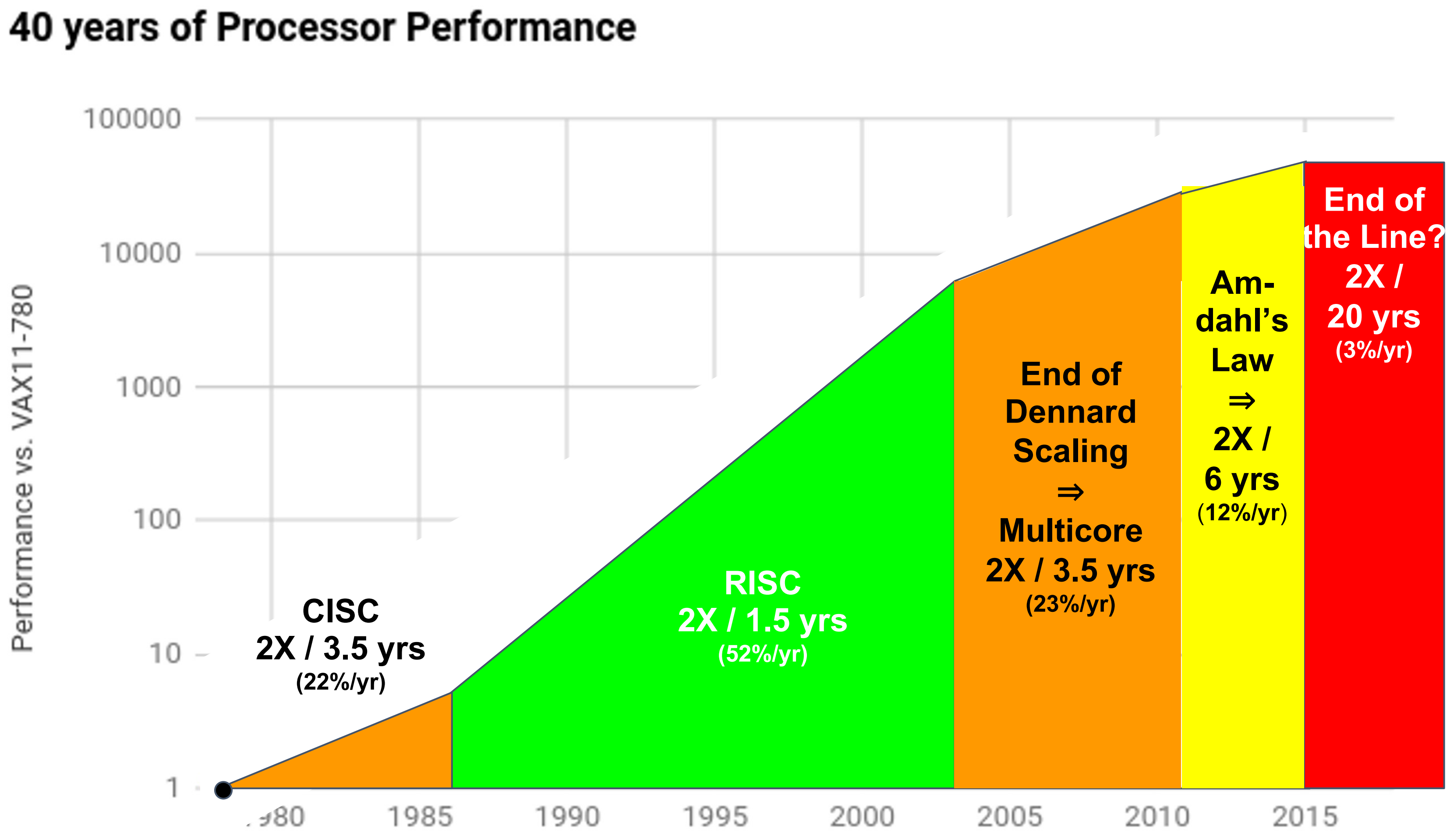
Slide from Prof. Patterson

It is becoming increasingly difficult to push the boundary

Building a 3nm fab costs around $20B. This is still economical given the $600B ARR for the semi-conductor industry, but it is questionable how much farther we can push the limit.

Source: high end performance packaging 3d/2.5d integration report, Yole, Development, 2020.

But It has Slowed Down

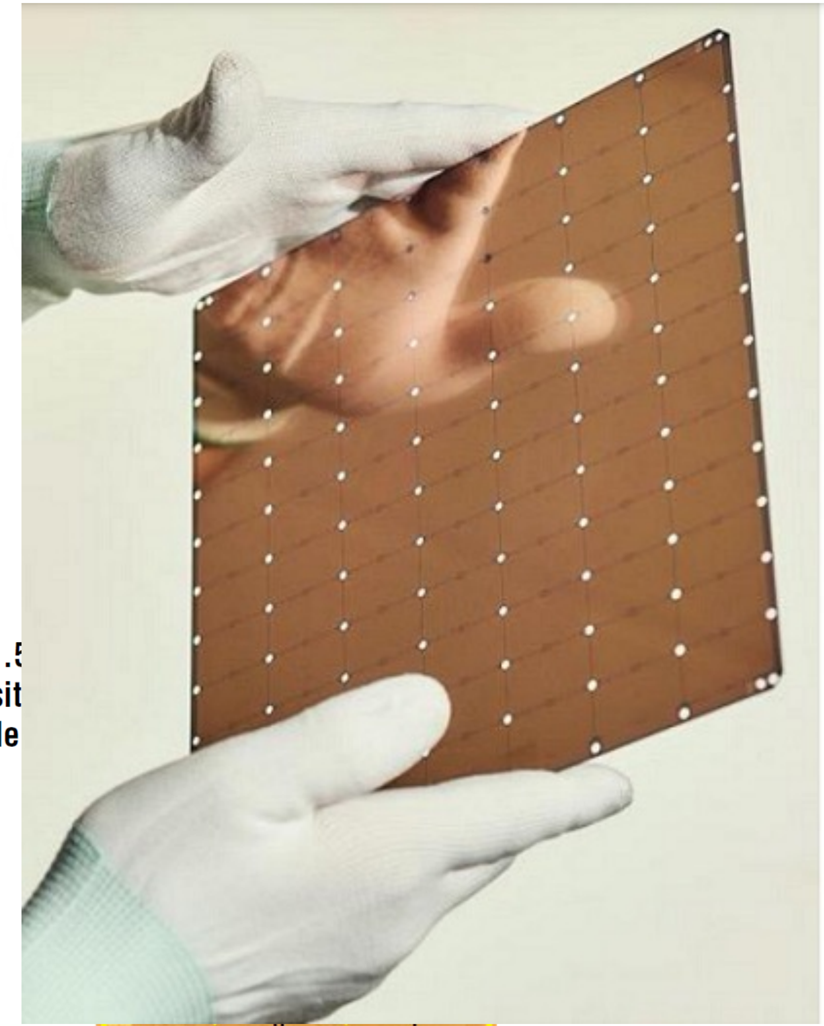


Based on SPECintCPU. Source: John Hennessy and David Patterson, Computer Architecture: A Quantitative Approach, 6/e. 2018

Domain Specific Accelerators

- John Hennessy and David Patterson, "A New Golden Age for Computer Architecture," *Communications of the ACM*, February 2019

Domain Specific Accelerators



Cerebras Wafer-Scale Engine

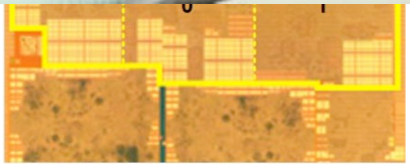
	Gen1 WSE	Gen2 WSE
Fabrication process	16 nm	7 nm
Silicon area	46,225 mm²	46,225 mm²
Transistors	1.2 Trillion	2.6 Trillion
AI-optimized cores	400,000	850,000
Memory on-chip	18 GB	40 GB
Memory bandwidth	9 PB/s	20 PB/s
Fabric bandwidth	100 Pb/s	220 Pb/s

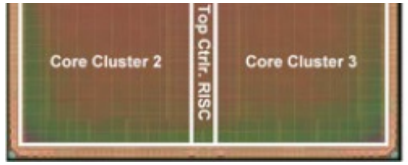
IOPs/mW

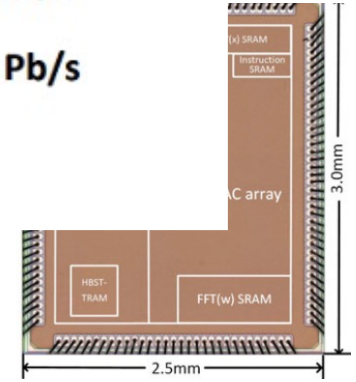
PS/mW

7.1 An 11.5
Sparsit
Mobile

-to-12b Unified Neural-
Circulant-Enabled
with 8.1× Higher
Based 2D Data-Reuse







AI Chip Landscape

basicmi.github.io/AI-chip

Tech Giants/Systems

Google
Microsoft
aws
IBM
facebook
HUAWEI
Baidu 百度
Alibaba Group 阿里巴巴集团
FUJITSU
NOKIA
TOSHIBA
Hewlett Packard Enterprise
DELL

IC Vender/Fabless

intel
SAMSUNG
NVIDIA
QUALCOMM
AMD
NXP
ST
XILINX
MEDIATEK
BROADCOM
MARVELL
Rockchip 瑞芯微电子

IP/Design Service

arm
SYNOPSYS
Imagination
cadence
CEVA
VeriSilicon
alchip
GUC
FARADAY
eSilicon

Startup in China

Cambricon 寒武纪科技
地平线 Horizon Robotics
BITMAIN
intellifusion 云天励飞
ChipIntelli
Think Force
Canaan
云知声 Unisound
AISPEECH 思必驰 专注人性化的智能语音
Rokid
肇观电子 NextVPU
Enflame

Startup Worldwide

cerebras
Wave Computing
Graphcore
habana
SambaNova SYSTEMS
thinci
LIGHTELLIGENCE
HAILO Empowering Intelligence
KALRAY
Tenstorrent
MYTHIC
Preferred Networks
brainchip
PEZY Computing
GREENWAVES TECHNOLOGIES
AIMOTIVE
KONIKU
Tachyum
flexlogix
SYNTIANT
gyrfalcon technology
NOVUMIND

more on https://basicmi.github.io/AI-Chip/

Compiler

TensorFlow
PyTorch
NVIDIA TensorRT
tvm
nGraph Compiler stack (Beta)
plaidML

Benchmarks

MLPerf
AI - Benchmark
AI Matrix.
AIIA

Designing an accelerator

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

The remaining 20%: SW development + Full system design

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

https://eecs.wsu.edu/~dtao/download/Distributed-DL-PyTorch-Zhang.pdf

3) Deploy at scale? Distributed Deep Learning

Naveen Kumar (Google)

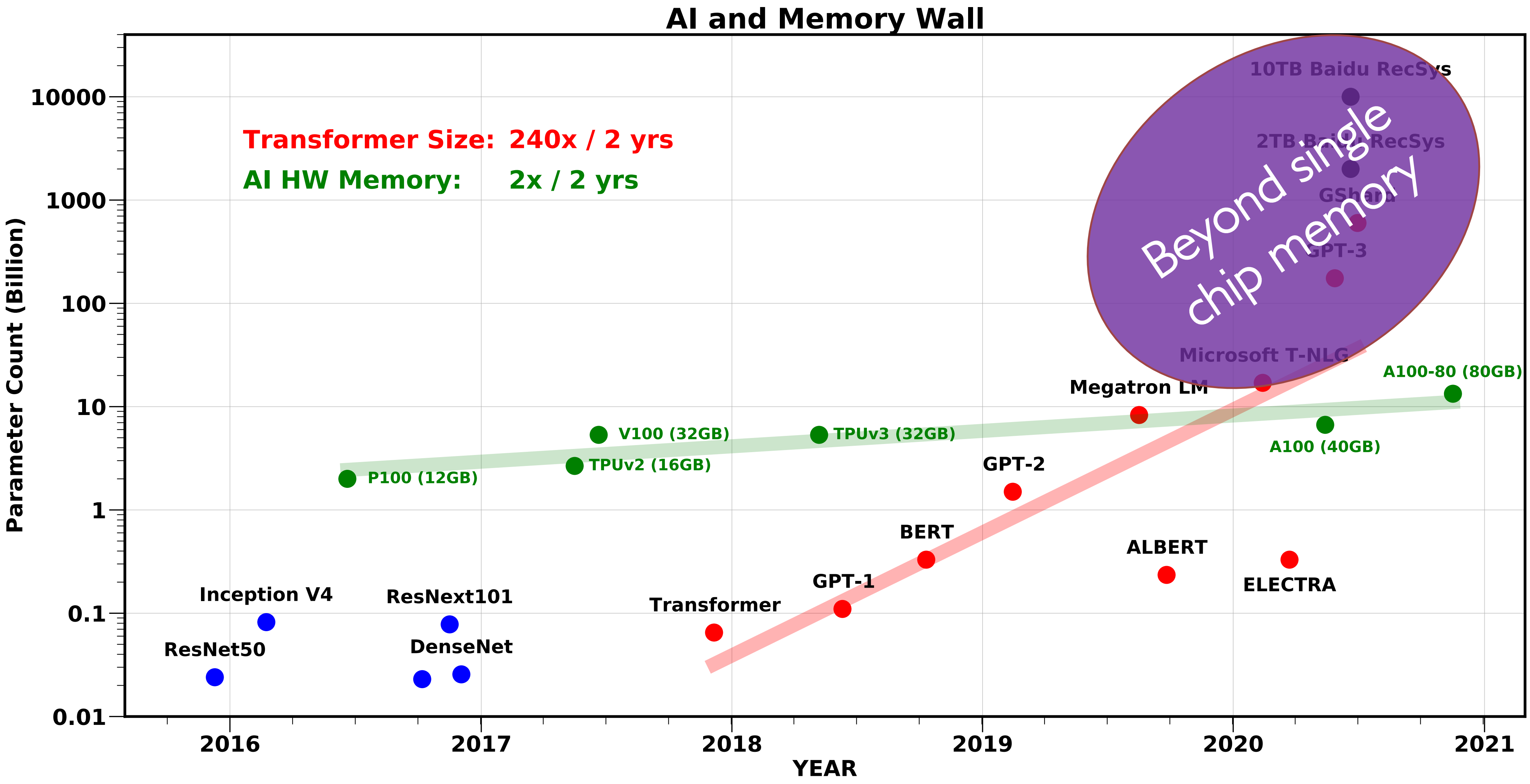
Distributed Deep Learning

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

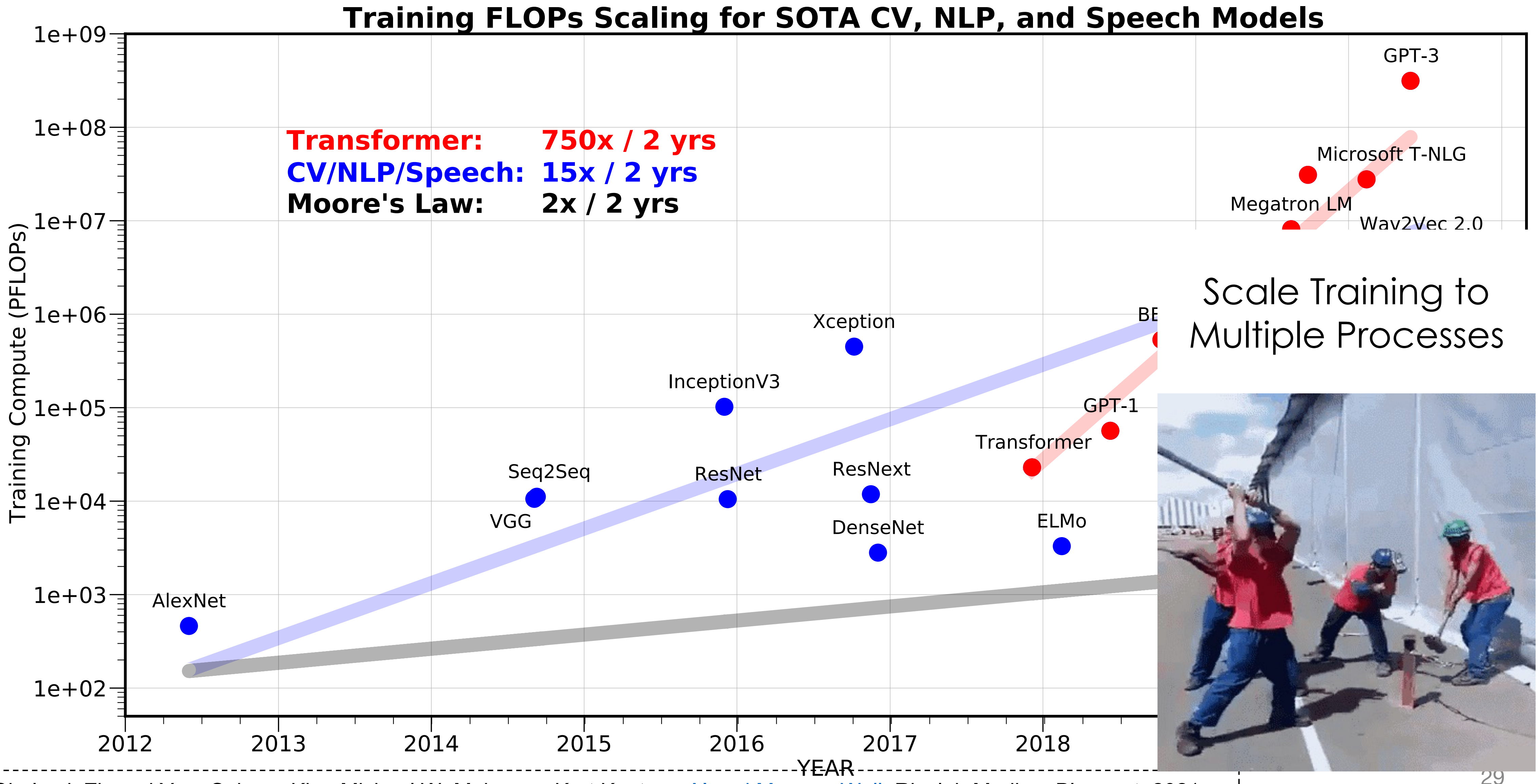
*Very simplified definition.

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

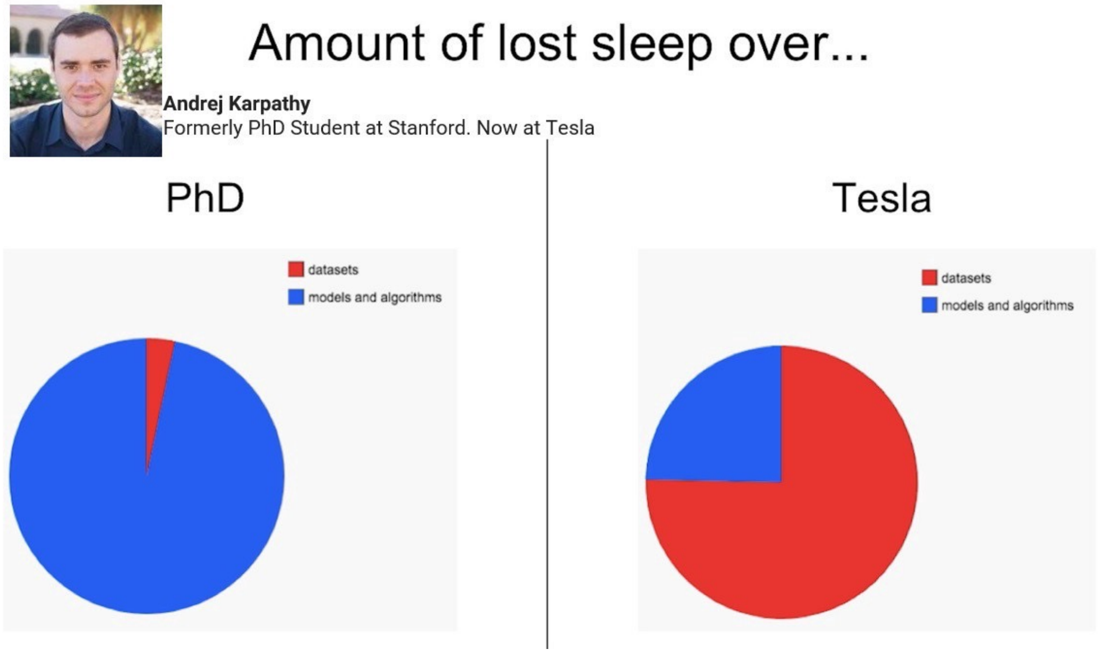
- More data *often** dominates models and algorithms


EXPERT OPINION

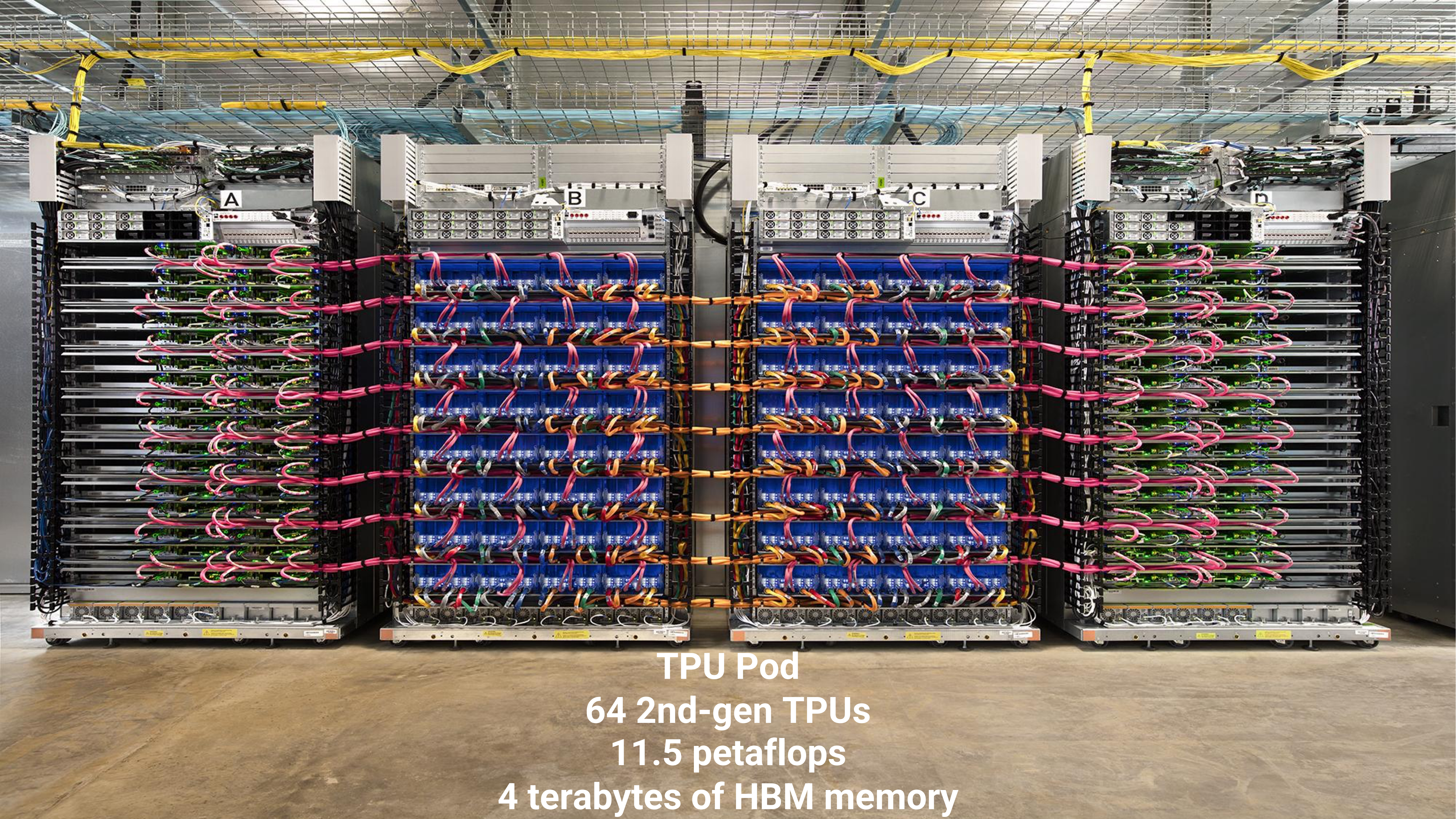
Contact Editor: **Brian Brannon,** bbrannon@computer.org

The Unreasonable Effectiveness of Data

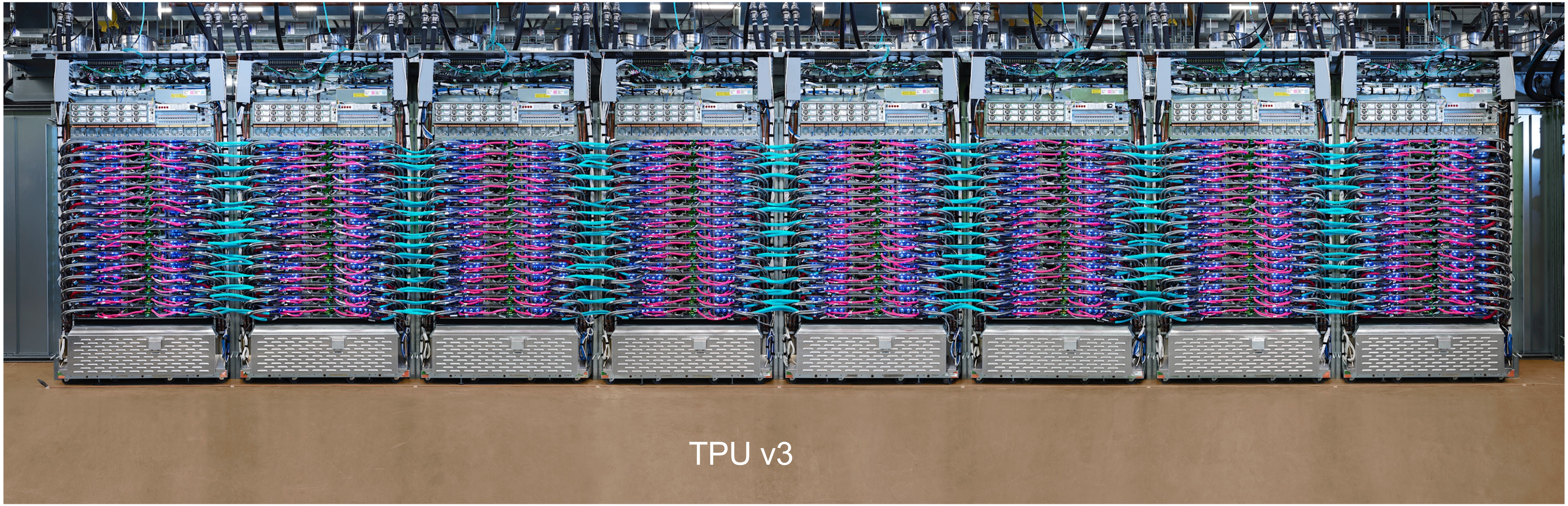
Alon Halevy, Peter Norvig, and Fernando Pereira, *Google*



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




NVIDIA.
NVIDI
Selene

Cambridge-1

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

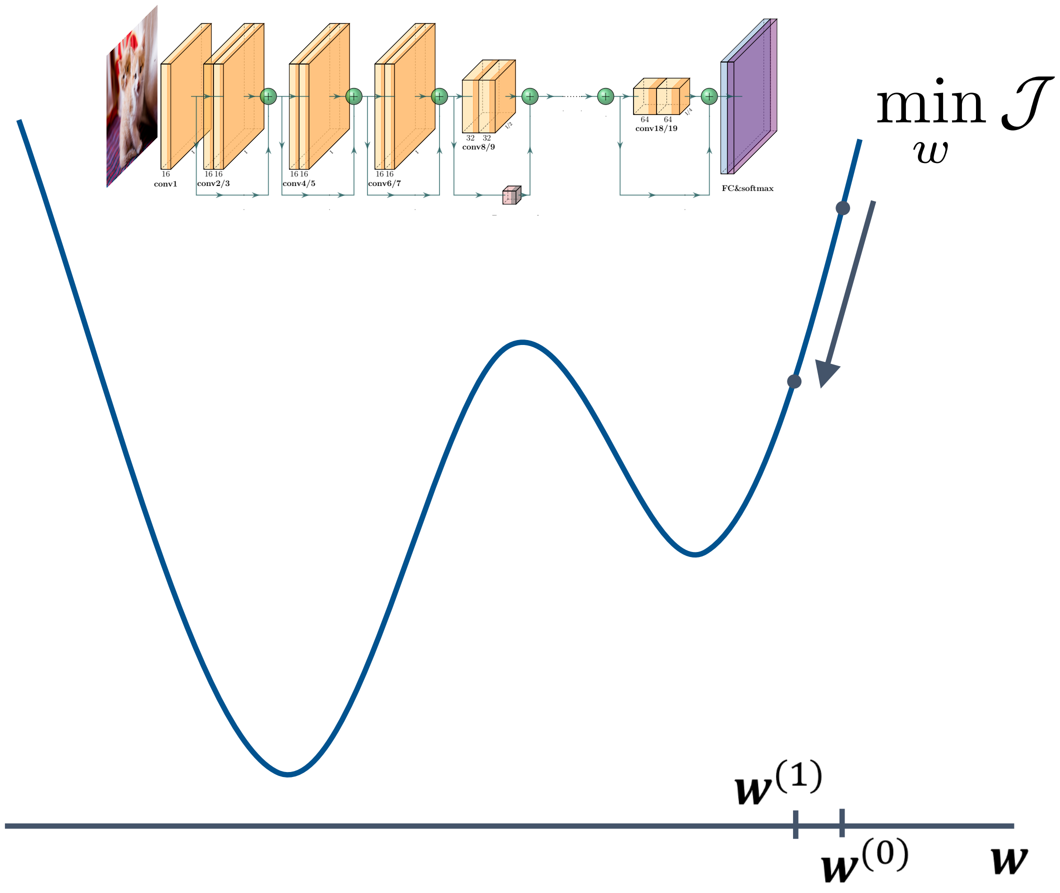
Throughput
System
Property

**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



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

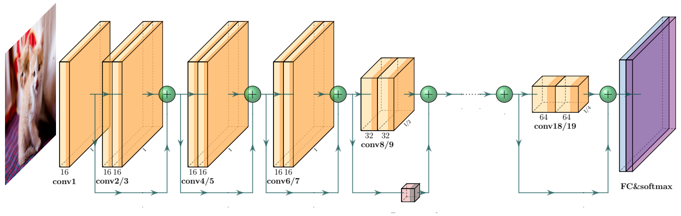
$$w^1 = w^0 - \alpha \underbrace{\frac{\partial \mathcal{J}(w^0)}{\partial w}}_{\Delta w}$$

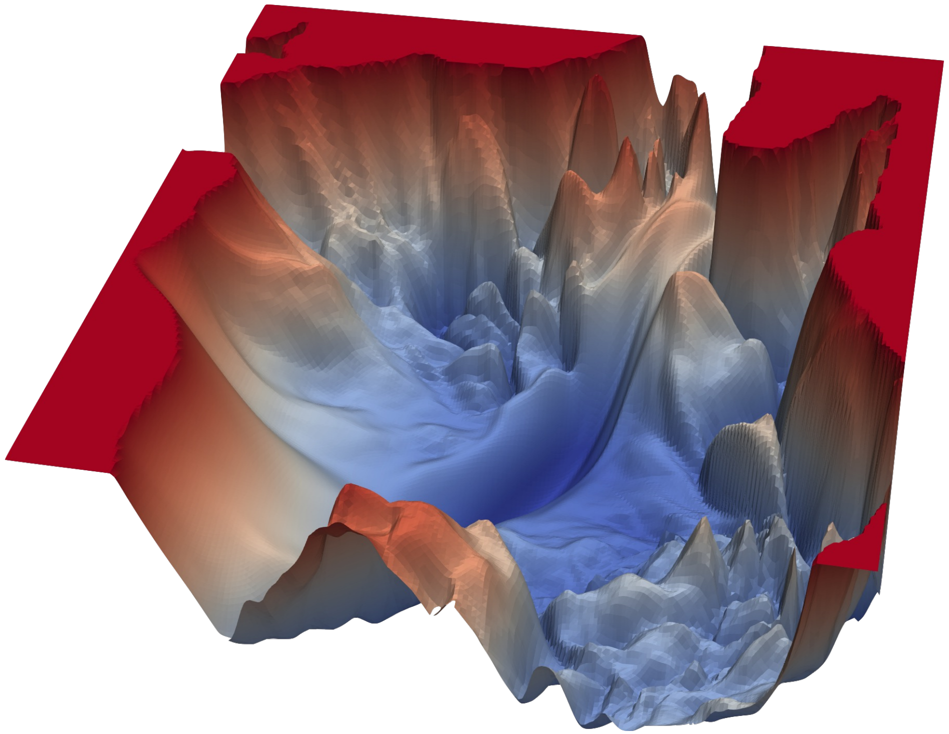
Learning rate

Two key elements:

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

Stochastic Gradient Descent





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

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

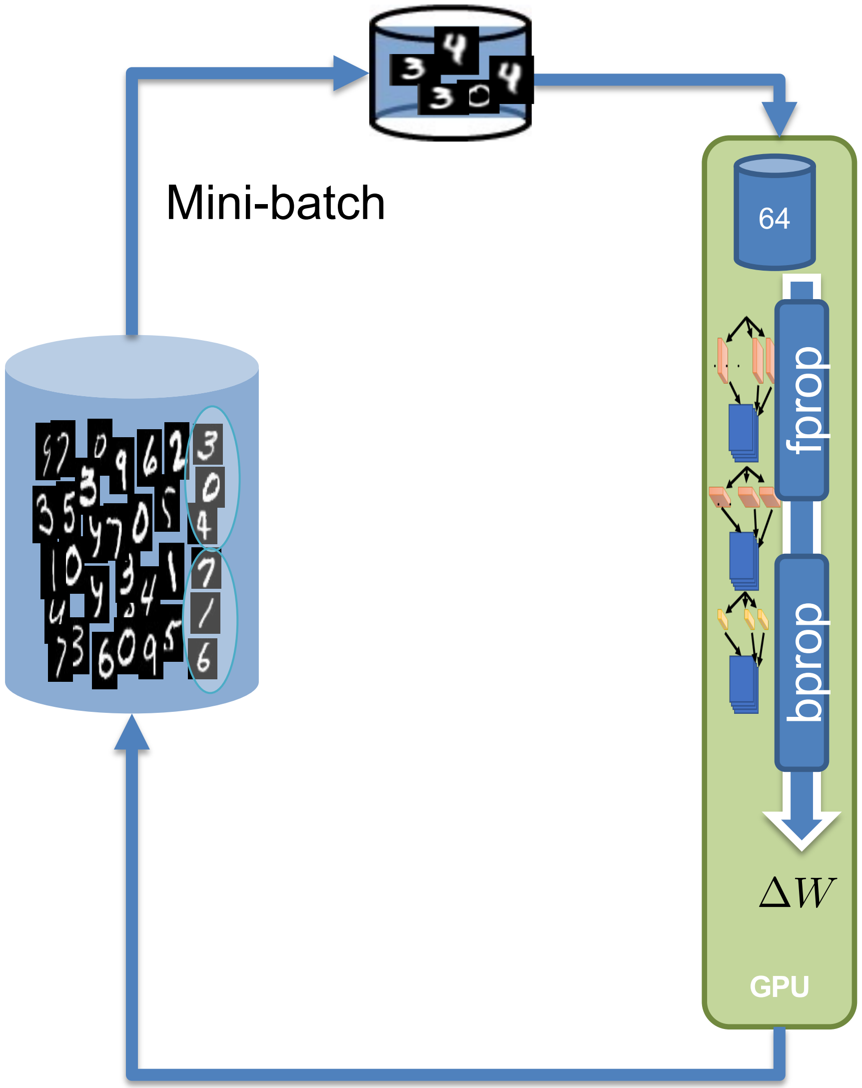
Learning rate

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.*

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

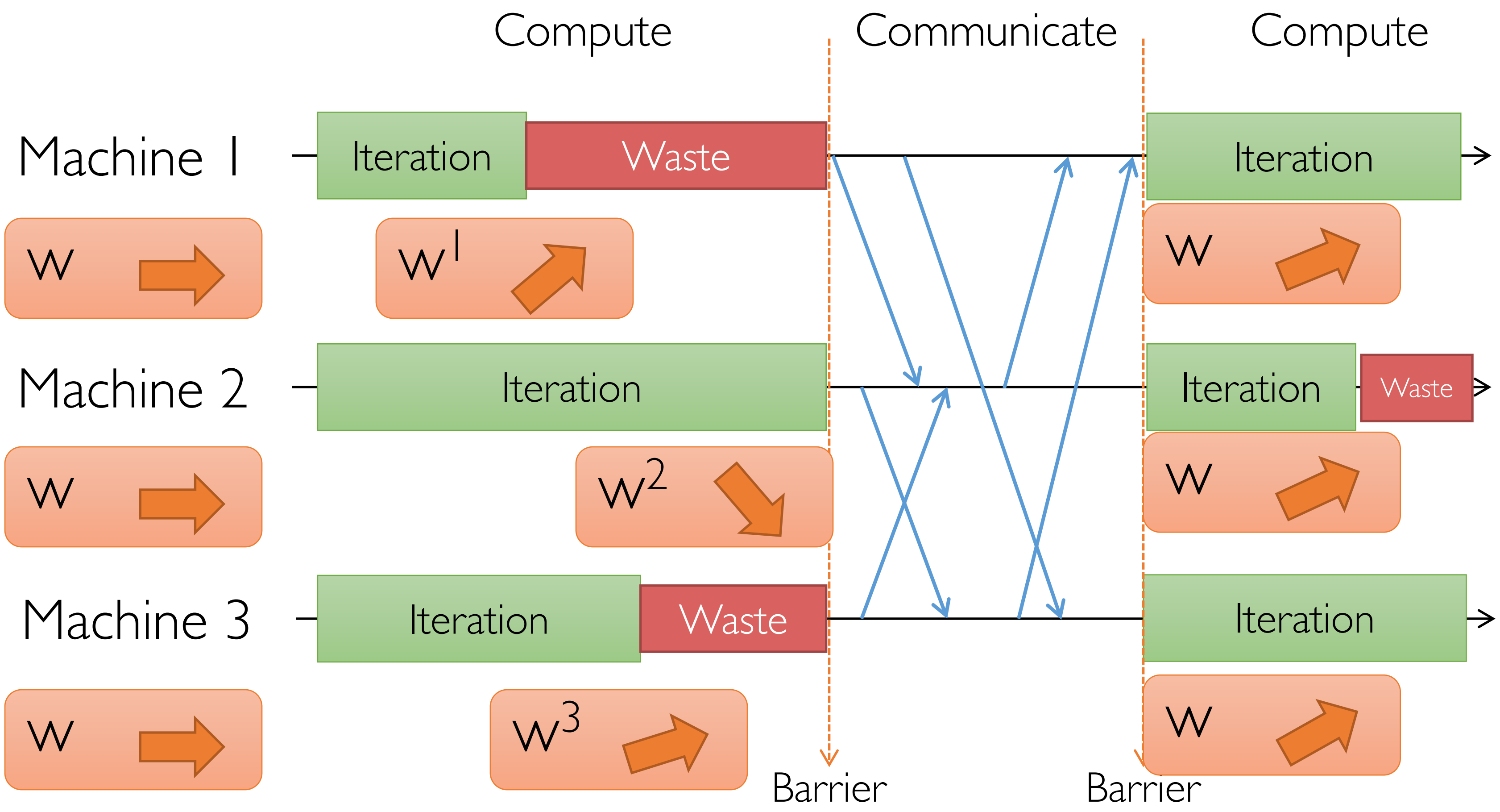
Model Parallelism: *Break the model and distribute processing of every layer to multiple PEs*

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

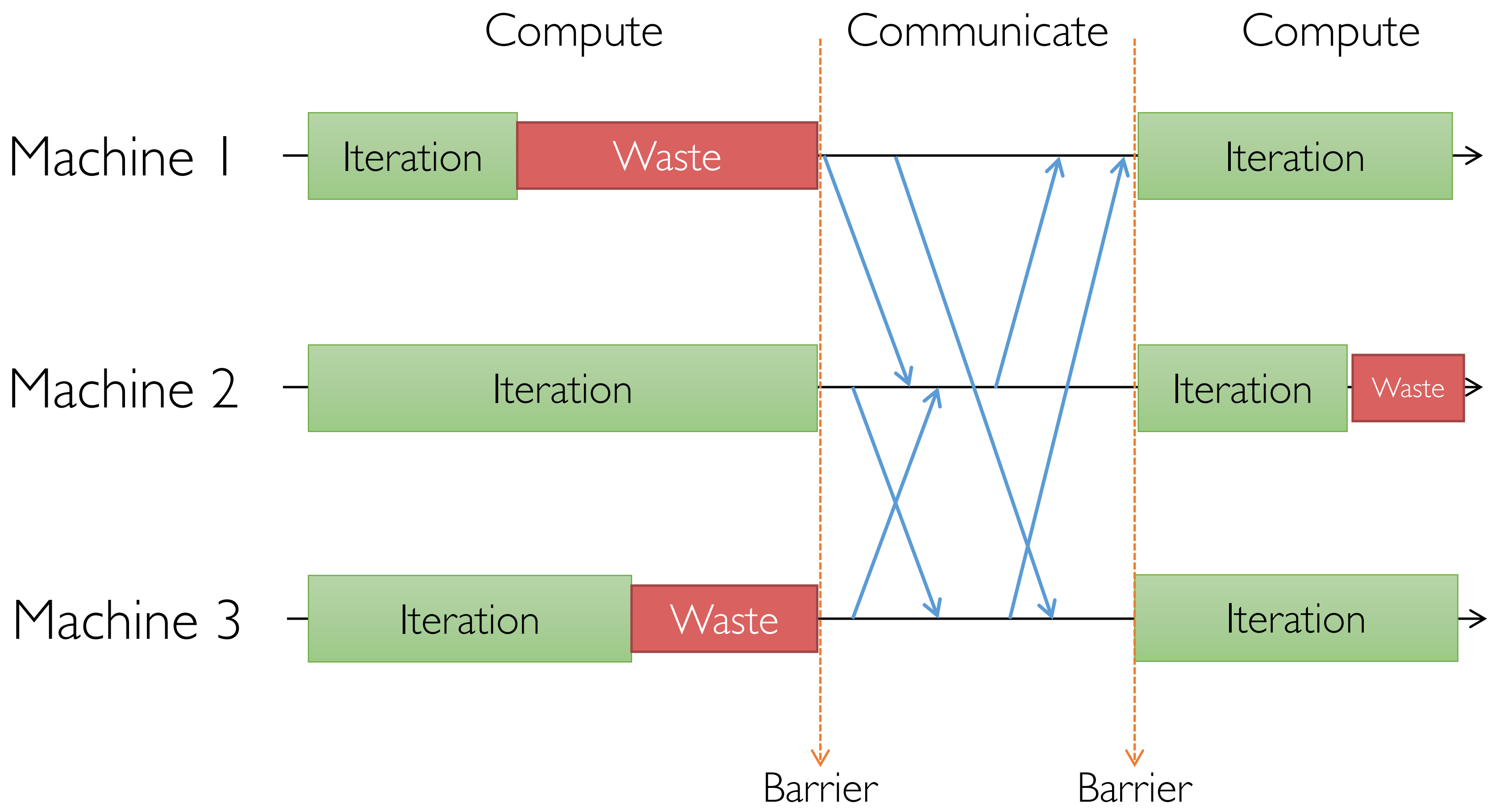
For either approach it is also possible to use ***synchronous*** *or* ***asynchronous*** *updates*

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

Bulk Synchronous Parallel (BSP) Execution

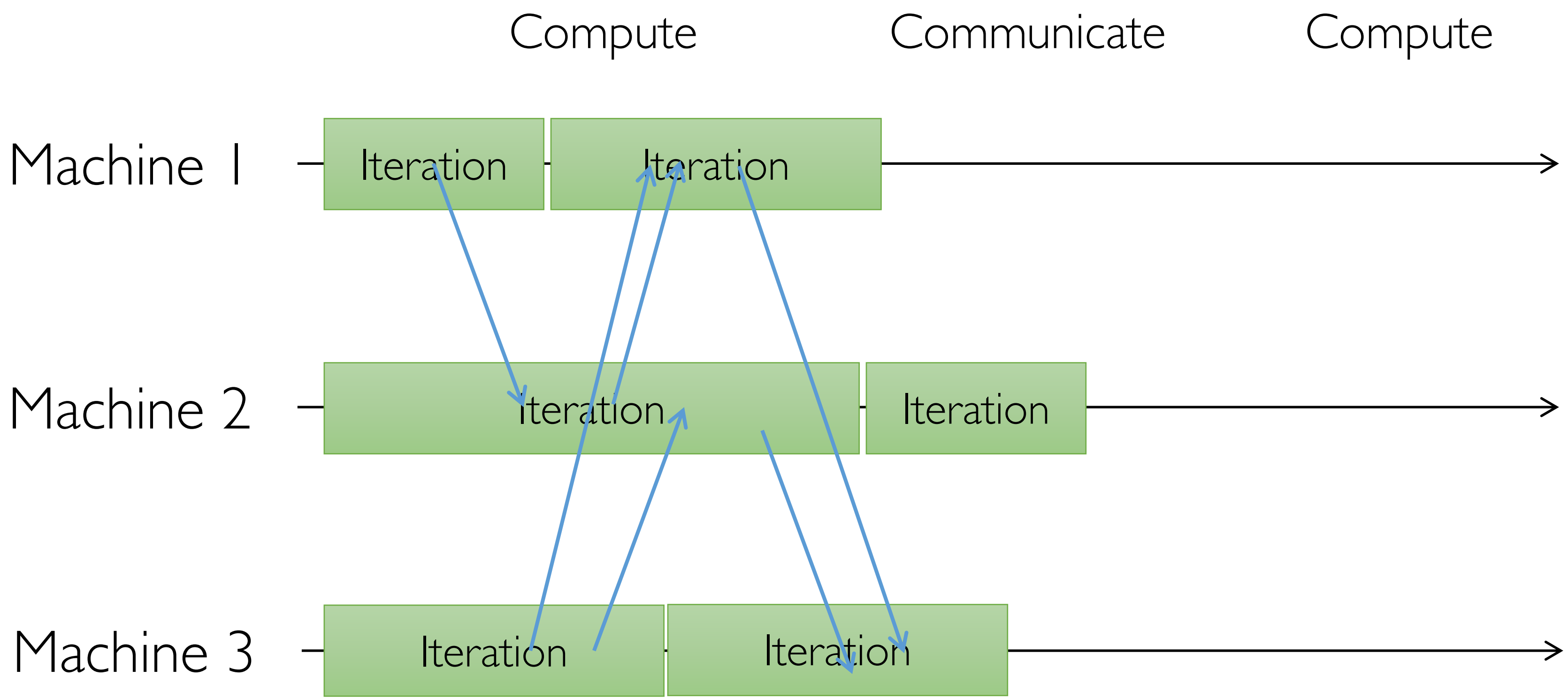


Bulk Synchronous Parallel (BSP) Execution



Enable more frequent coordination on parameter values

Asynchronous Execution

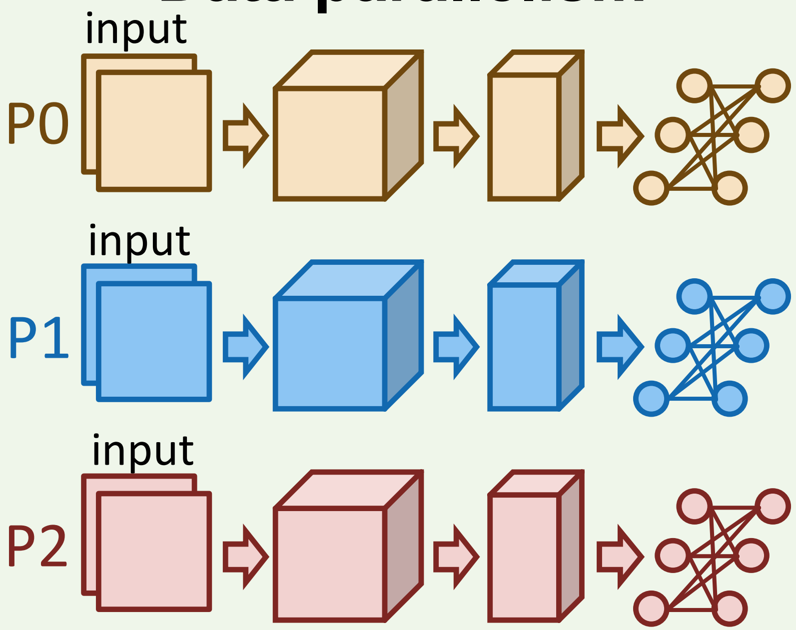


Enable more frequent coordination on parameter values, but often results in generalization loss. Today we will only focus on synchronous training.

Synchronous Data Parallel

Parallel and distributed training

Data parallelism



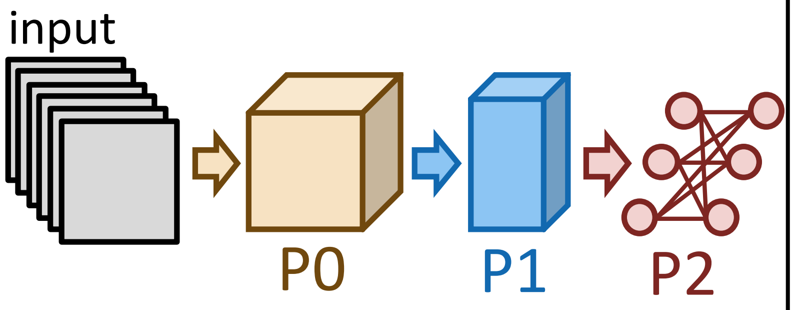
Pros:

a. Easy to realize

Cons:

a. Not work for large models
b. High allreduce overhead

Pipeline parallelism



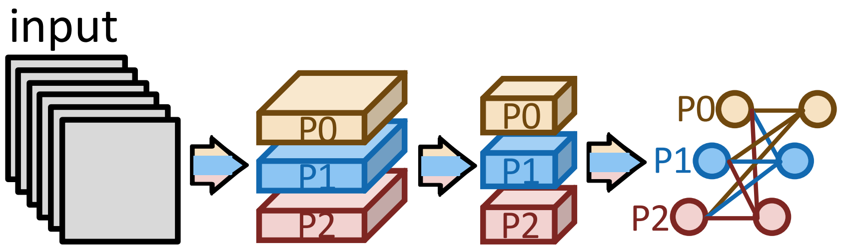
Pros:

a. Make large model training feasible
b. No collective, only P2P

Cons:

a. Bubbles in pipeline
b. Removing bubbles leads to stale weights

Model parallelism



Pros:

a. Make large model training feasible

Cons:

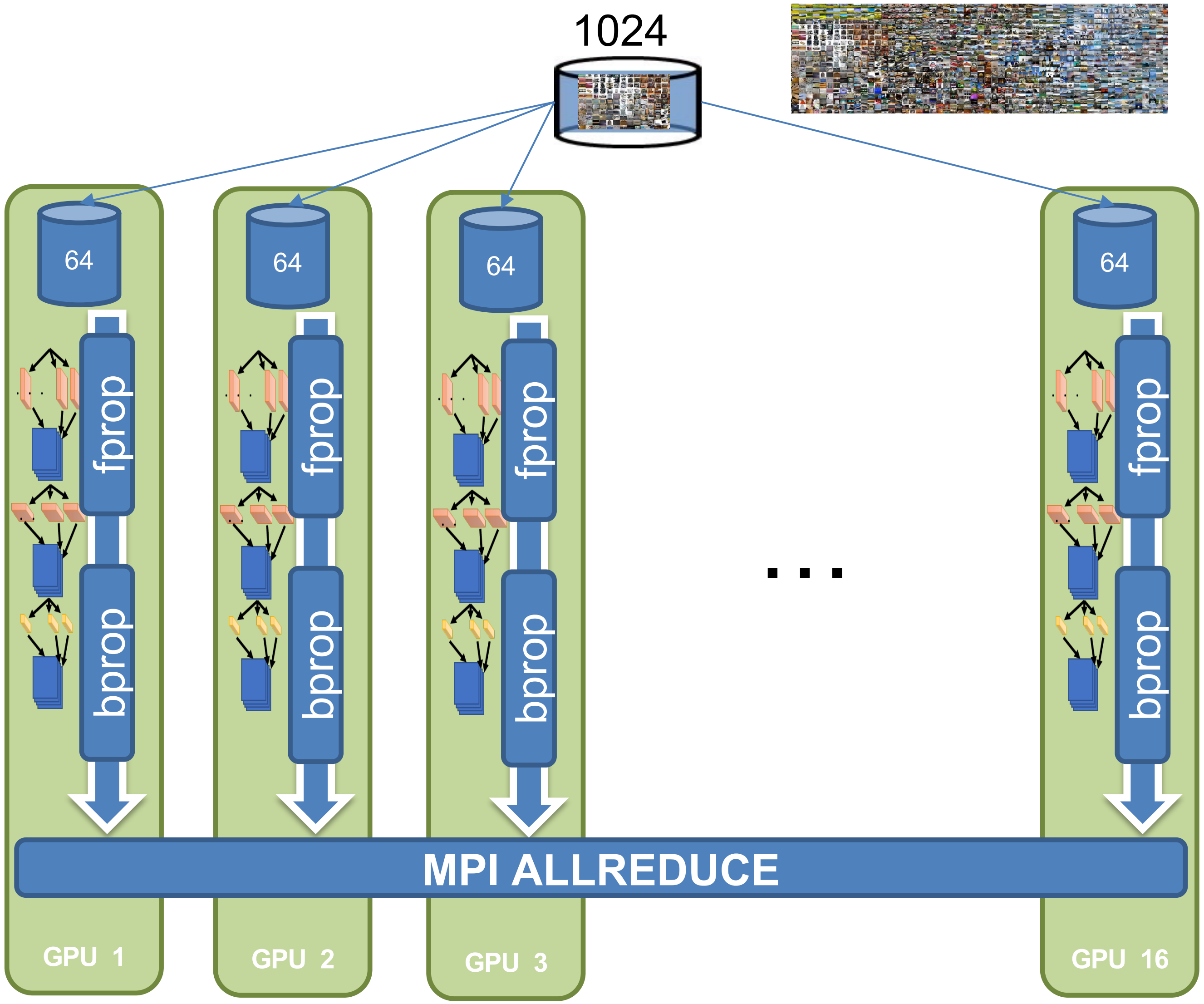
b. Communication for each operator (or each layer)

Slide: Courtesy of Shigang Li

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

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

$$a_1 = \sum_{i=1}^{B/4} \frac{\partial \mathcal{J}}{\partial w}$$

GPU 1

$$b_1 = \sum_{i=B/4}^{2B/4} \frac{\partial \mathcal{J}}{\partial w}$$

GPU 2

$$c_1 = \sum_{i=2B/4}^{3B/4} \frac{\partial \mathcal{J}}{\partial w}$$

GPU 3

$$d_1 = \sum_{i=3B/4}^{B} \frac{\partial \mathcal{J}}{\partial w}$$

GPU 4

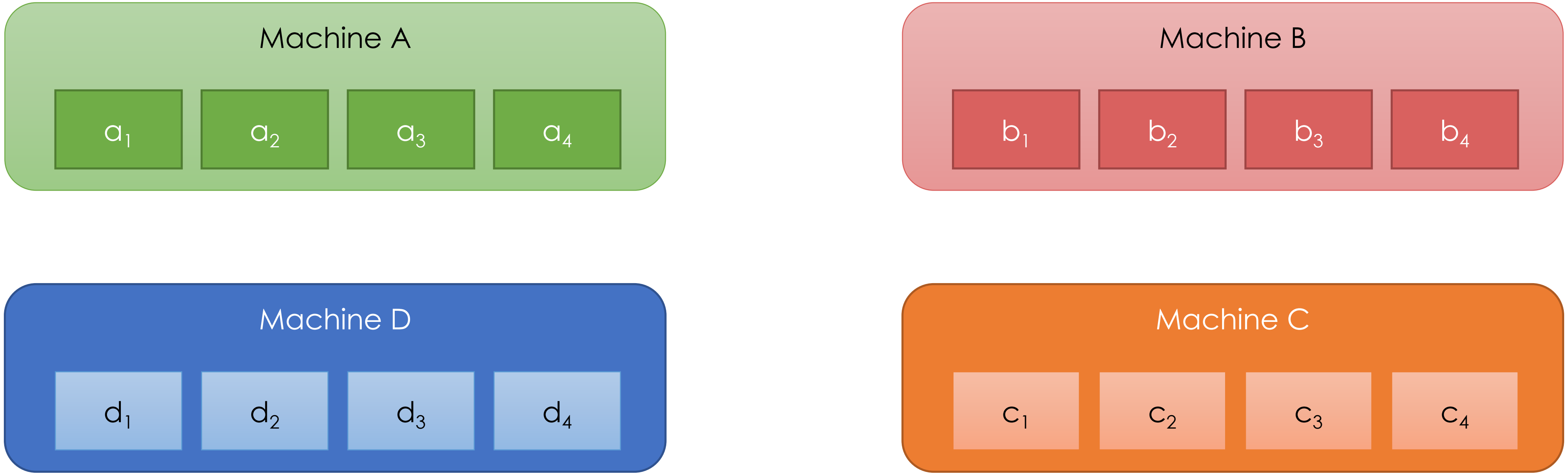
MPI ALLREDUCE

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

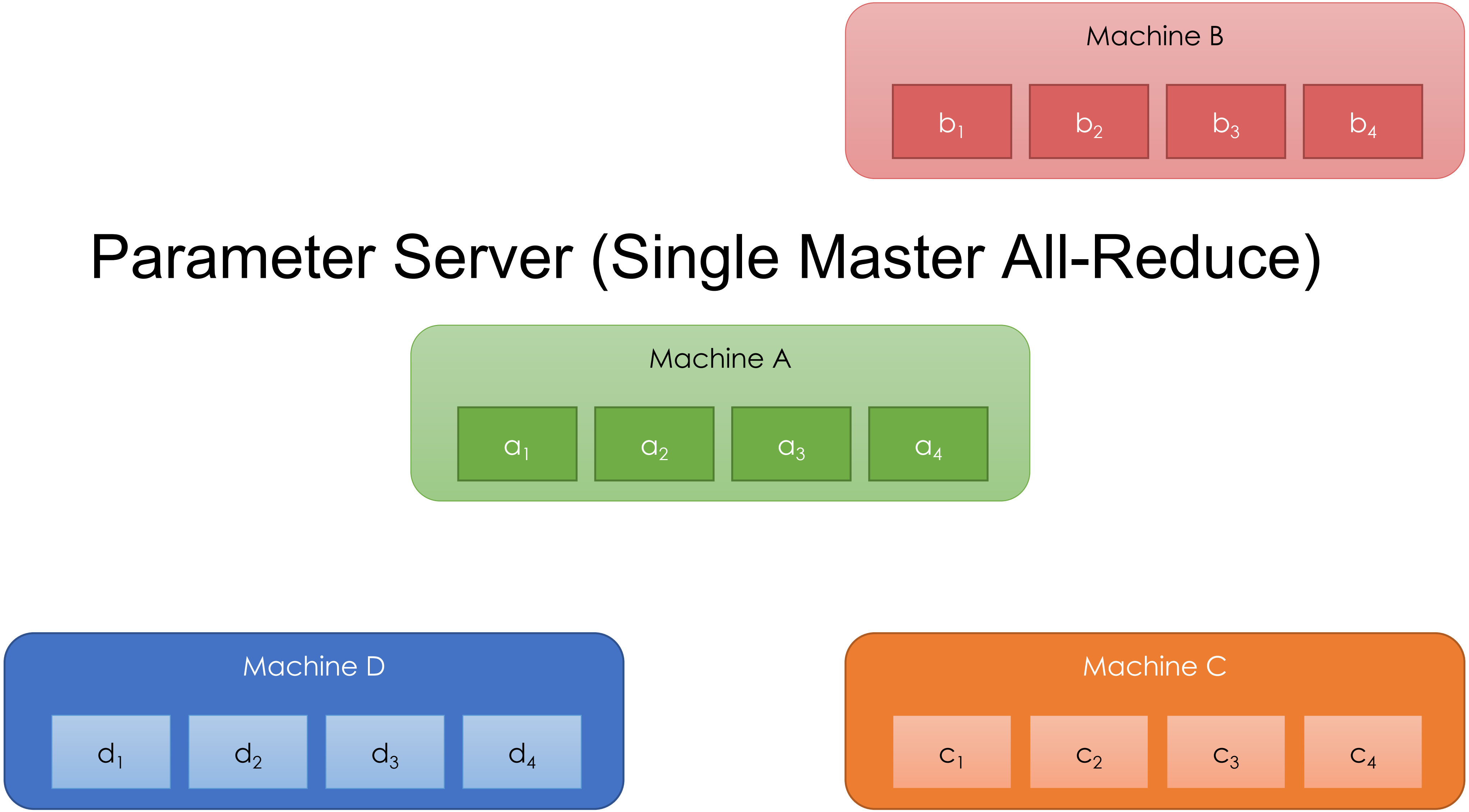
All Reduce

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

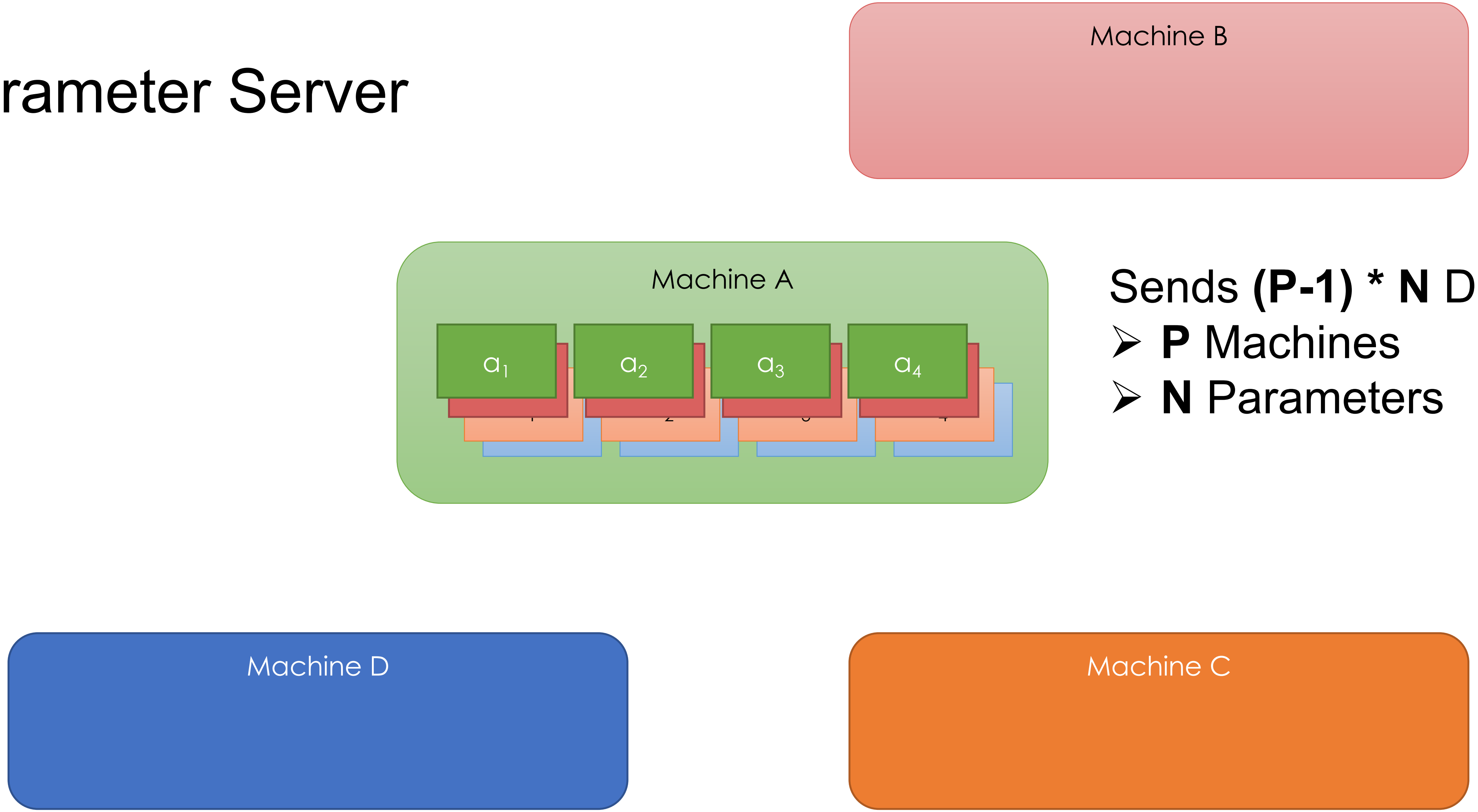
For simplicity, assume our model has 4 layers, and is trained on P=4 machines



Parameter Server (Single Master All-Reduce)



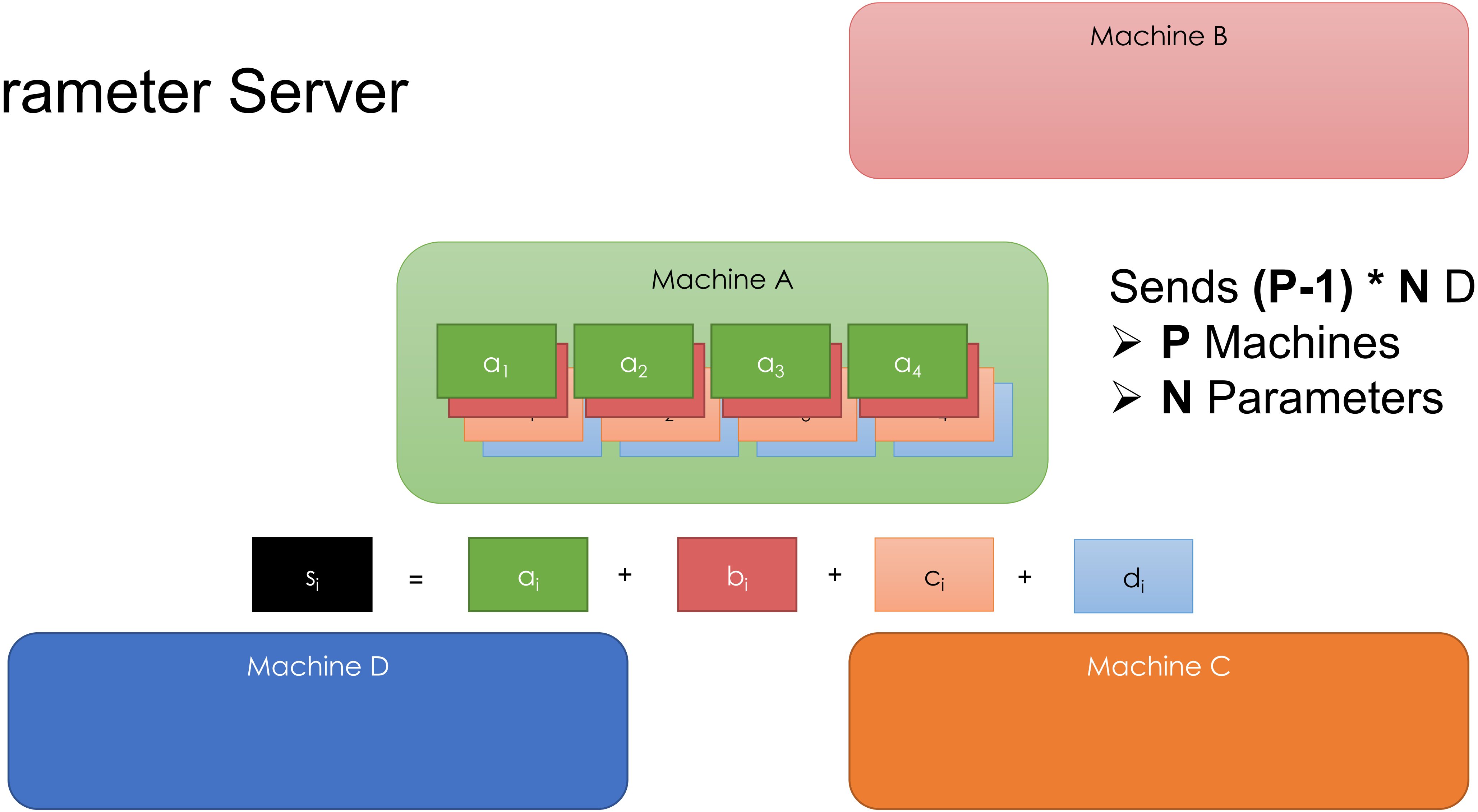
Parameter Server



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

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

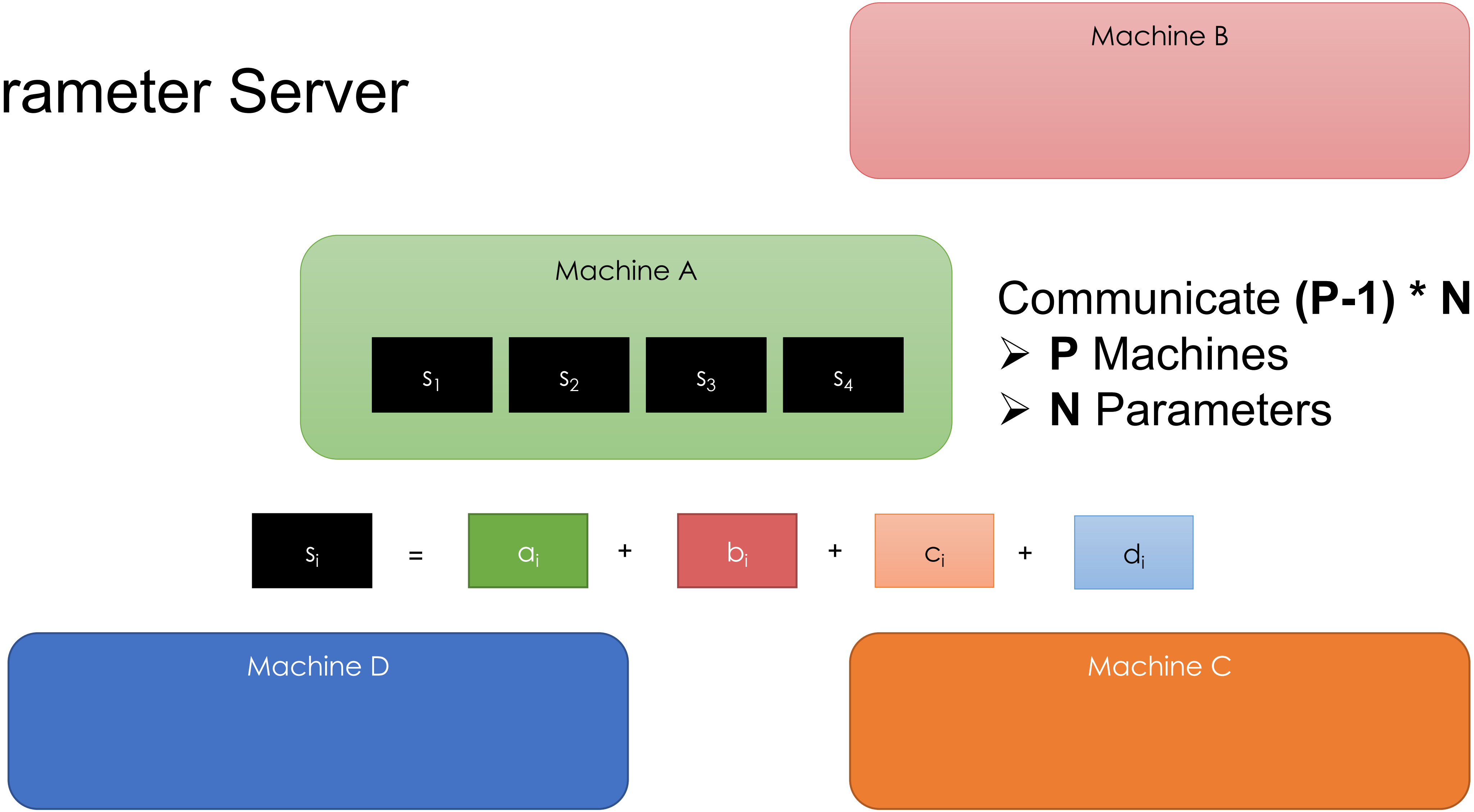
Parameter Server



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

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

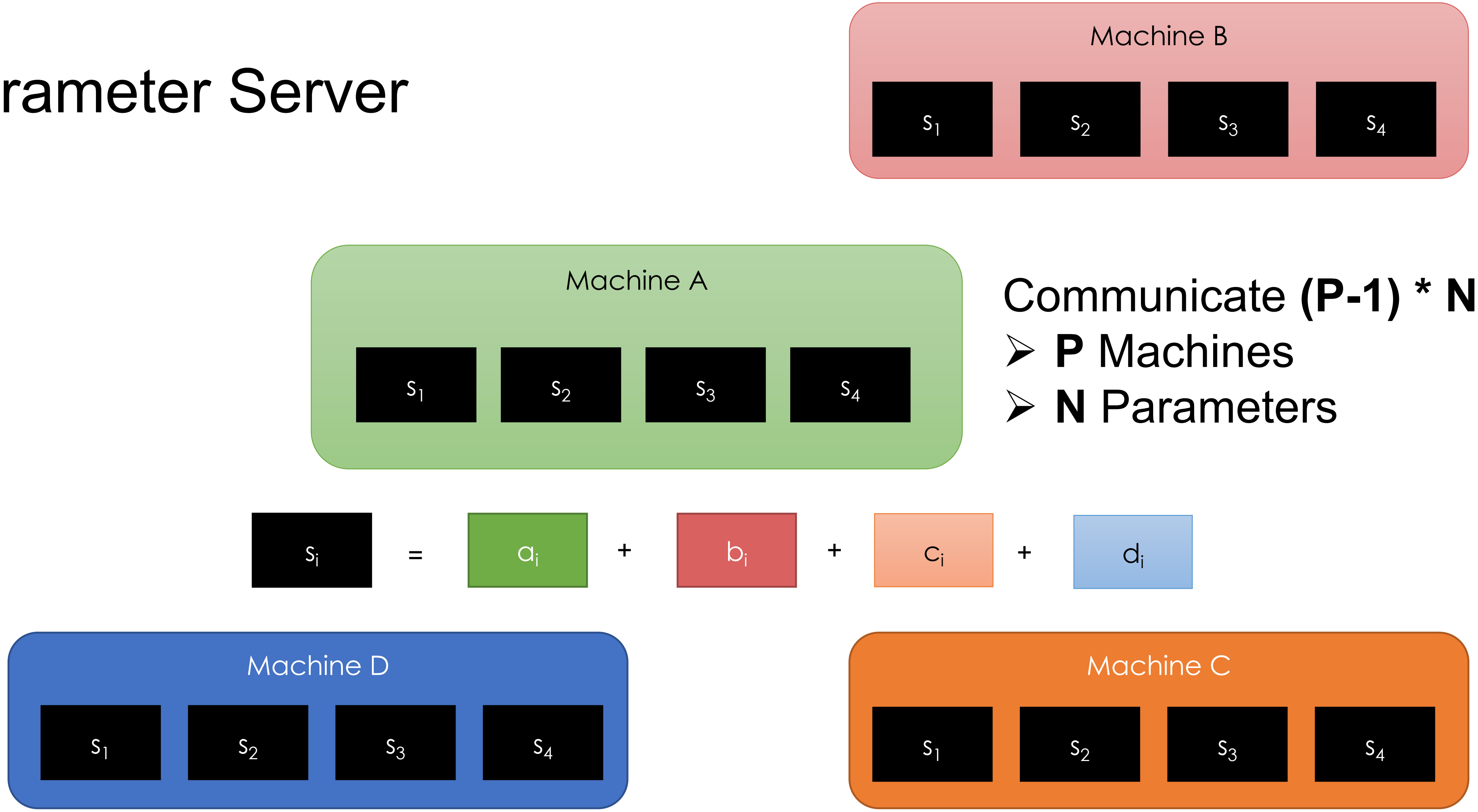
Parameter Server



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

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

Parameter Server

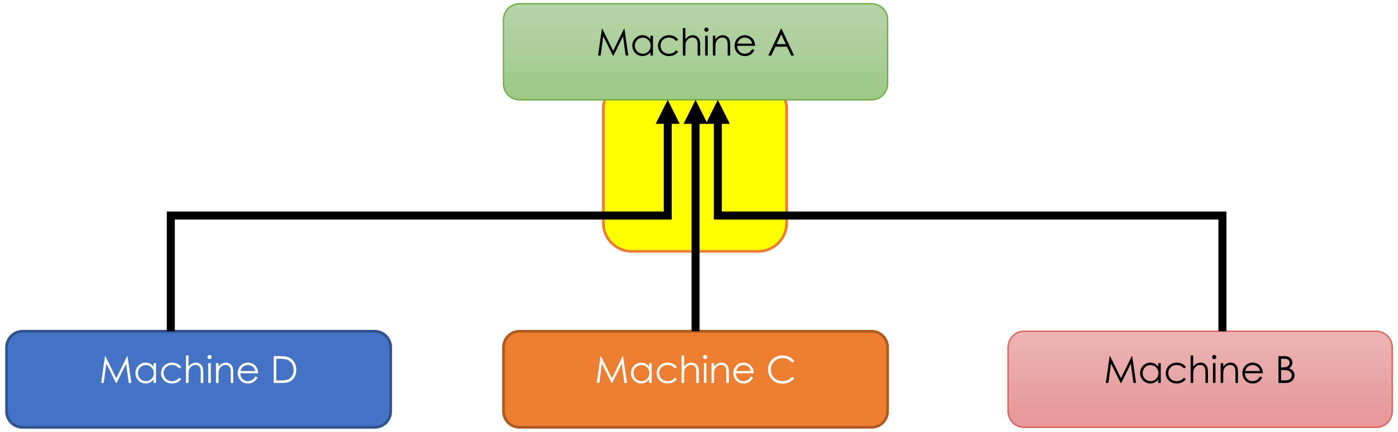


Communicate **(P-1)** * **N** Data *2

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

Parameter Server

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



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

Parameter Server All Reduce

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

Compute local sum on each machine

$$s_i = a_i + b_i + c_i + d_i$$

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**

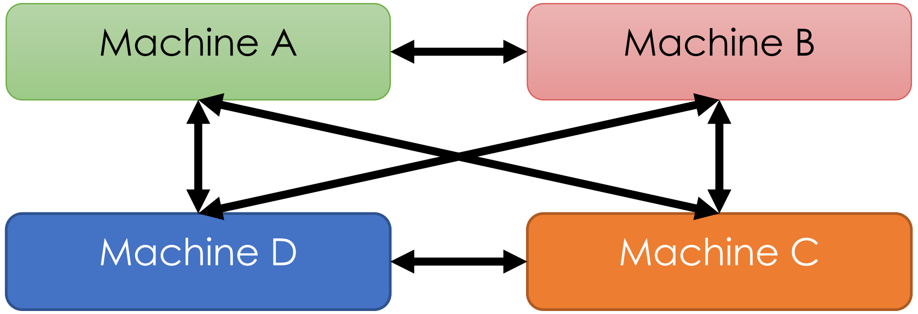
Total Communication per machine:

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

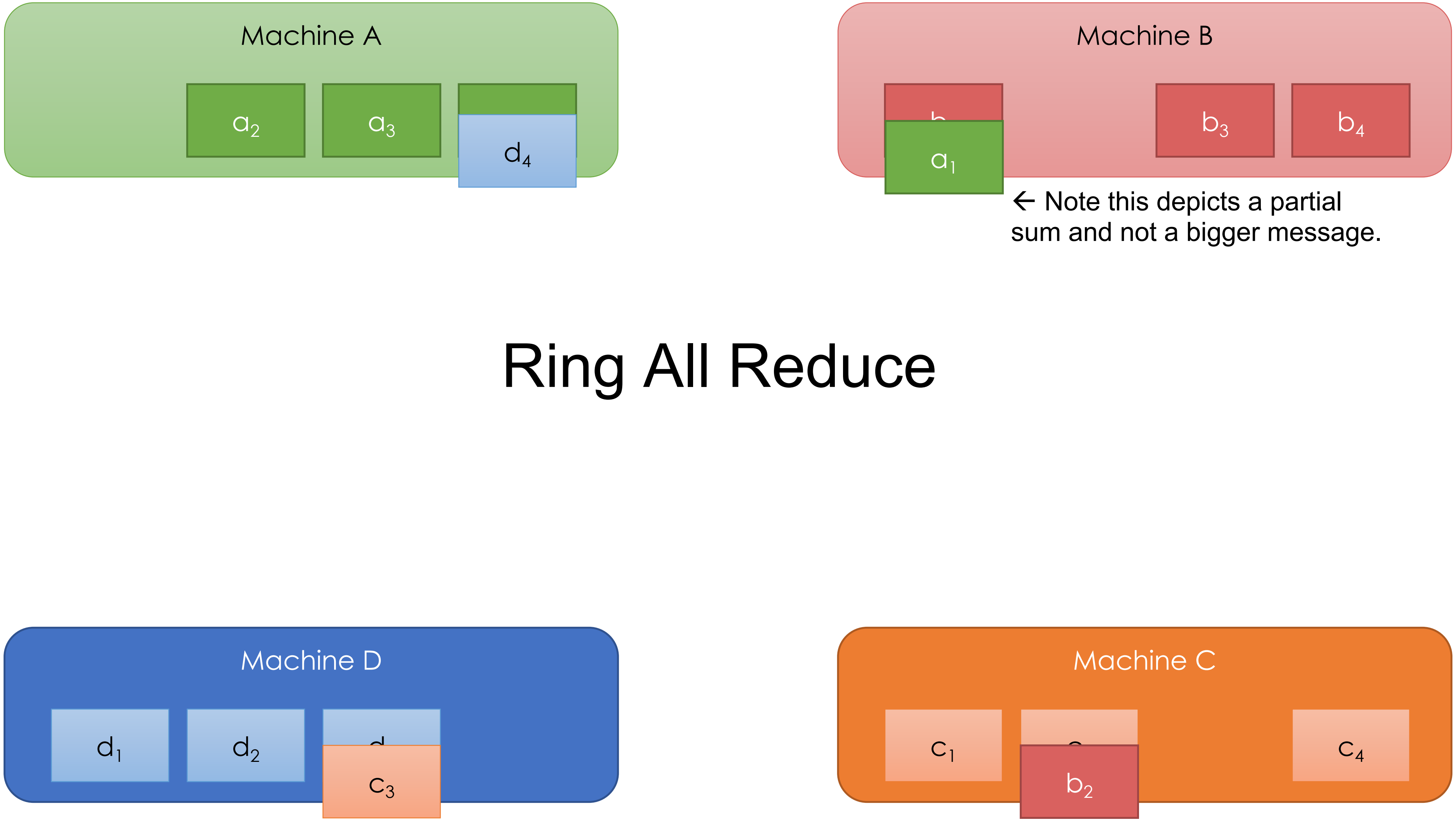


- Same **high fan-in** (P-1)
- **Reduced** Inbound Bandwidth = 2*(P-1)N/P
  - Previously 2*(P-1)*N for the parameter server

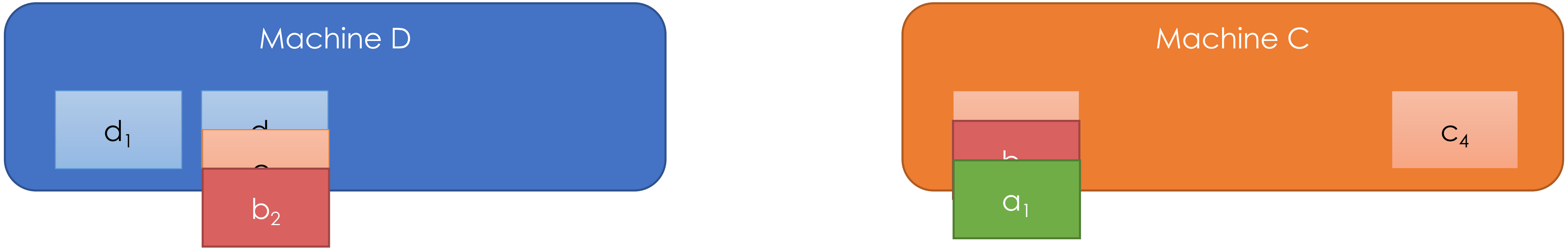
Ring All Reduce

Send messages in a ring to reduce fan-in.

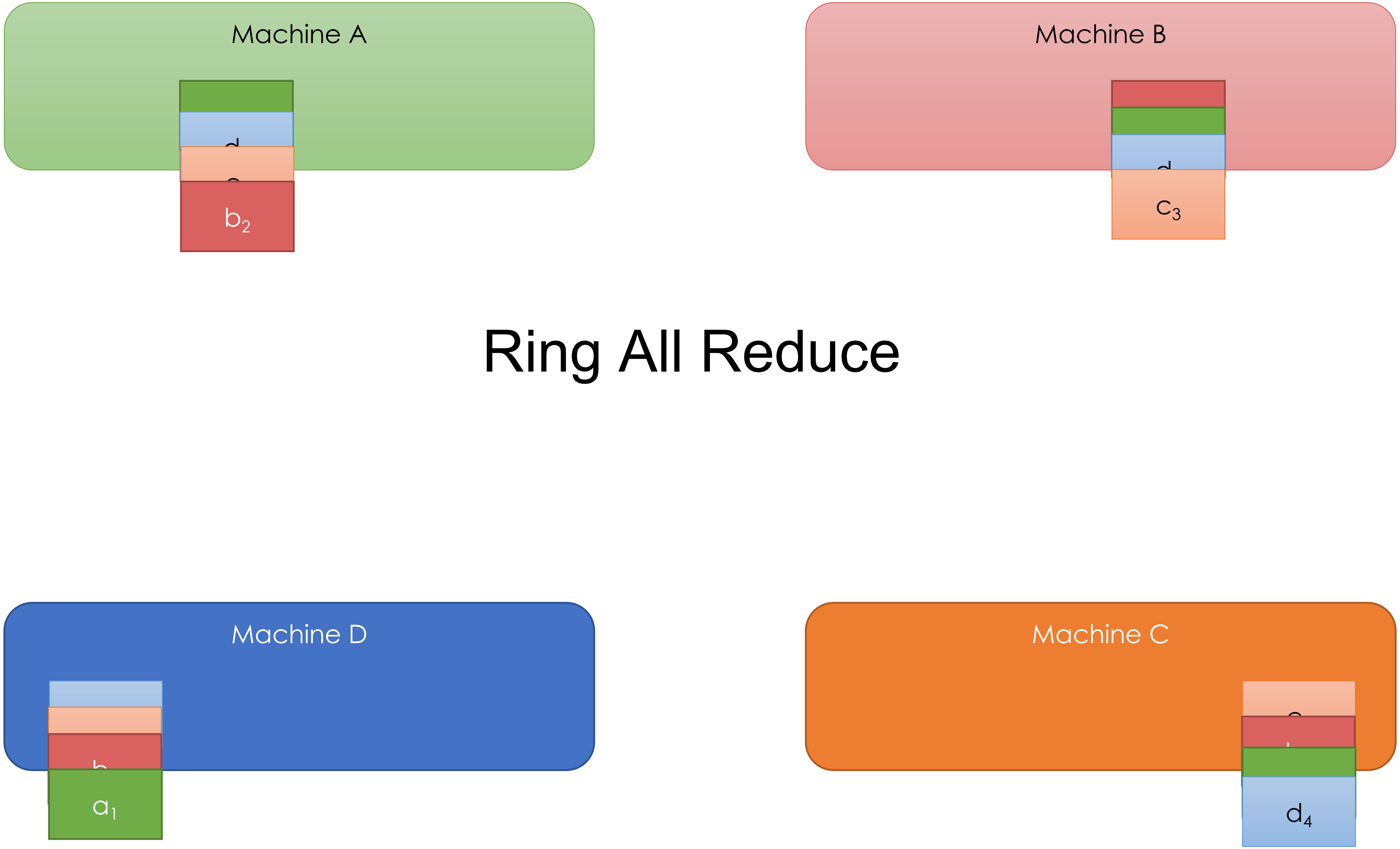
Ring All Reduce



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!)

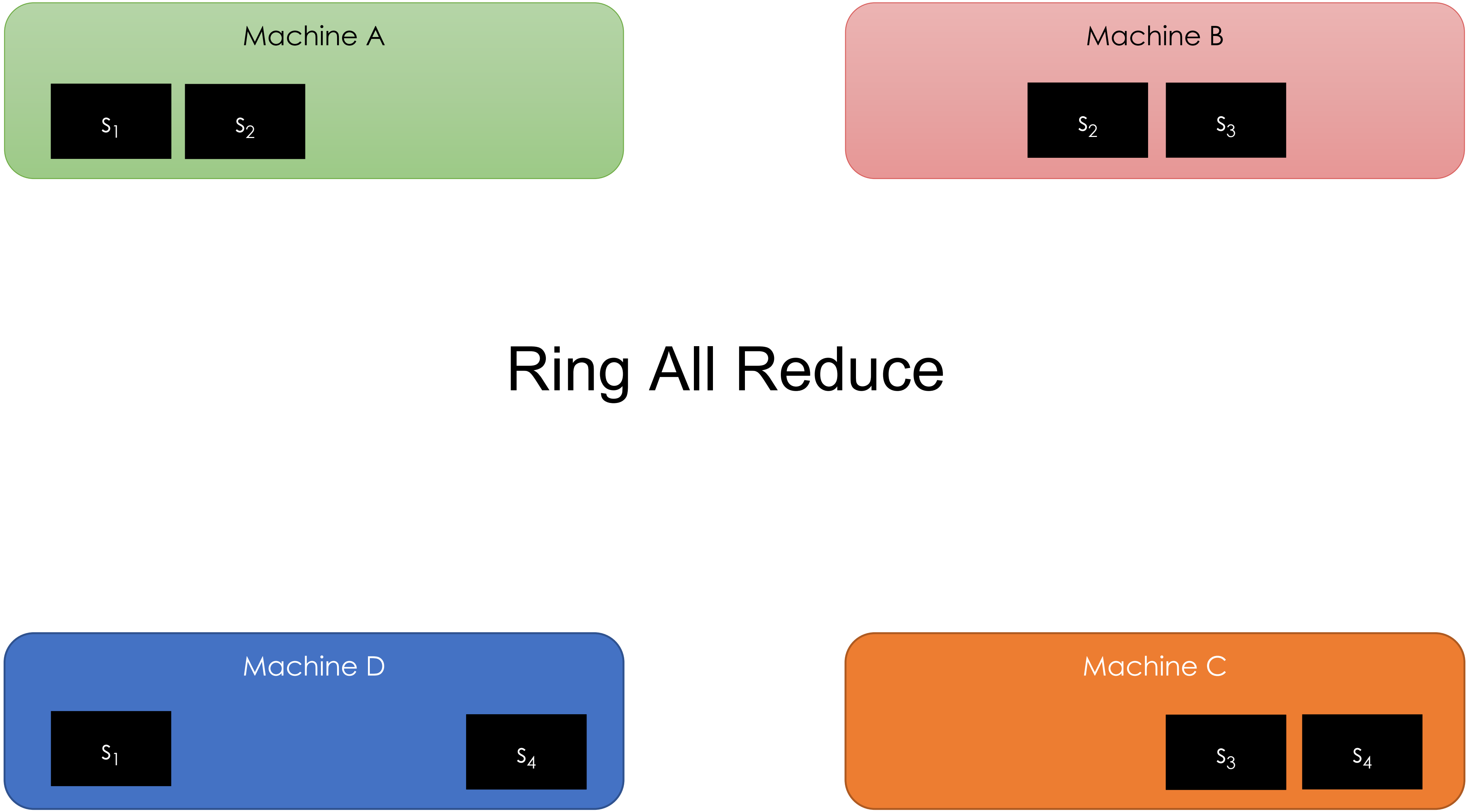
- **Fan-in Per Round:**
  - **1** (doesn't depend on P)

Ring All Reduce

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

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

Ring All Reduce



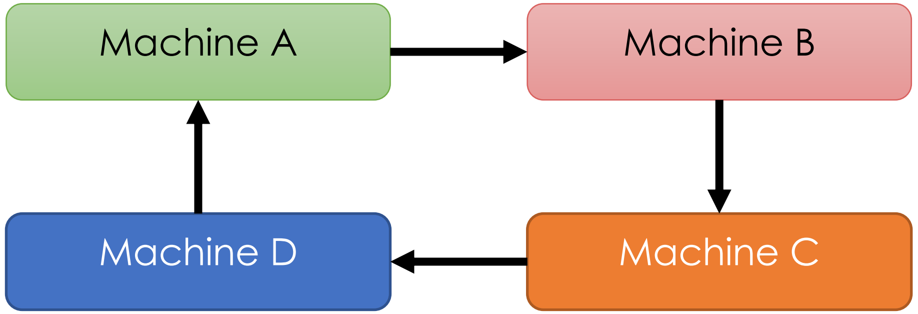
Ring All Reduce

Ring All Reduce


Machine A
S1
S2
S3
S4
Machine B
S1
S2
S3
S4
Ring All Reduce
Machine D
S1
S2
S3
S4
Machine C
S1
S2
S3
S4

Ring All-Reduce

- Simplified communication topology with low fan-in

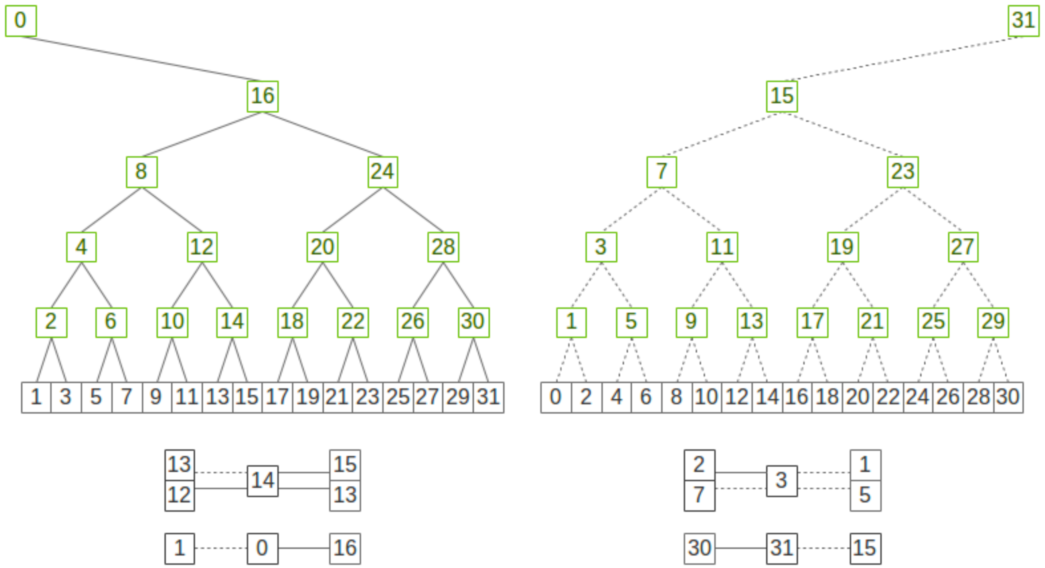


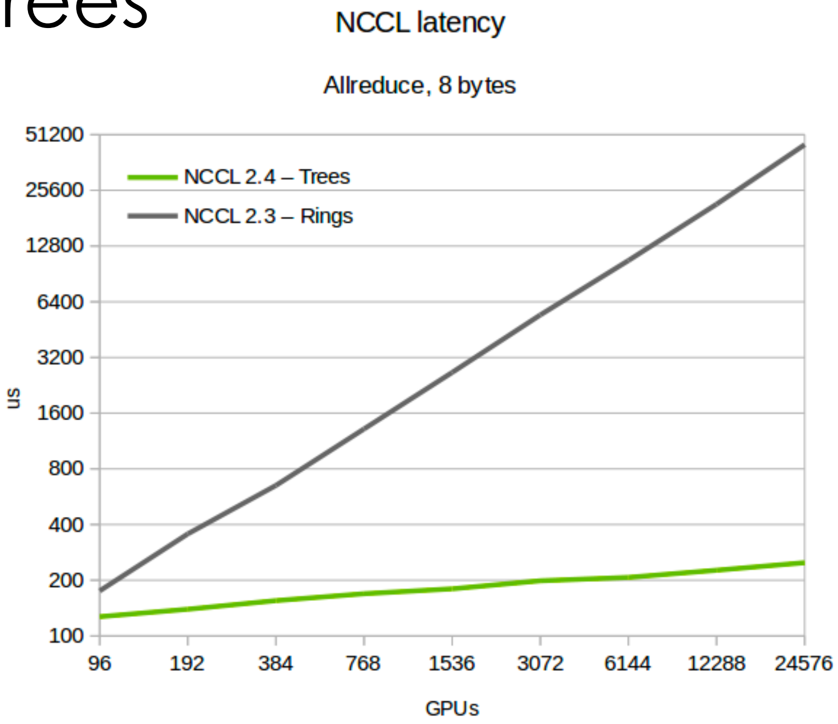
- Overall communication
  - Same total communication: **2*(P-1)*N**, but evenly distributed
  - Each Machine communicates 2*(P-1)N/P (almost independent of P)
  - **Fan-in** is constant (doesn't depend on P)

- **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

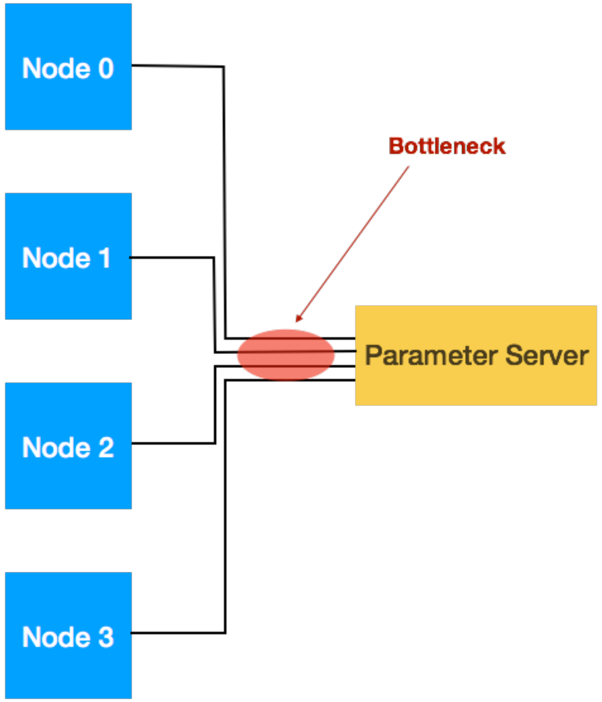
https://devblogs.nvidia.com/massively-scale-deep-learning-training-nccl-2-4/

Complexity Summary

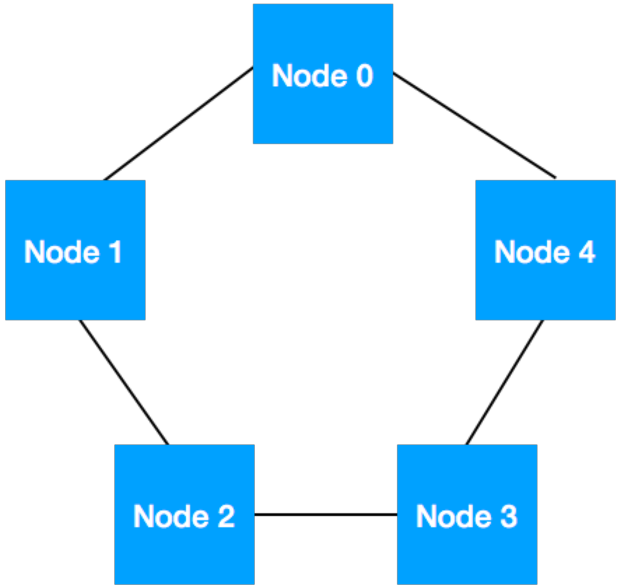
$$T_{comm} = (\alpha + PN\beta)$$

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

α latency
β bandwidth
N message size
P #processes



Parameter Server



Ring All-reduce

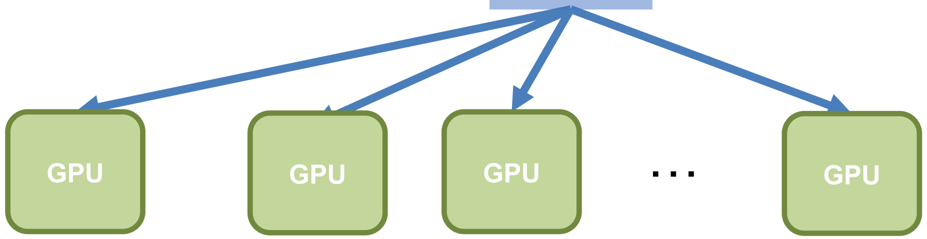
Great Reference: T. Rajeev, R. Rabenseifner, and W. Gropp. "Optimization of collective communication operations in MPICH." *The International Journal of High Performance Computing Applications*, 2005.

Data Parallel Training Complexity Analysis

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

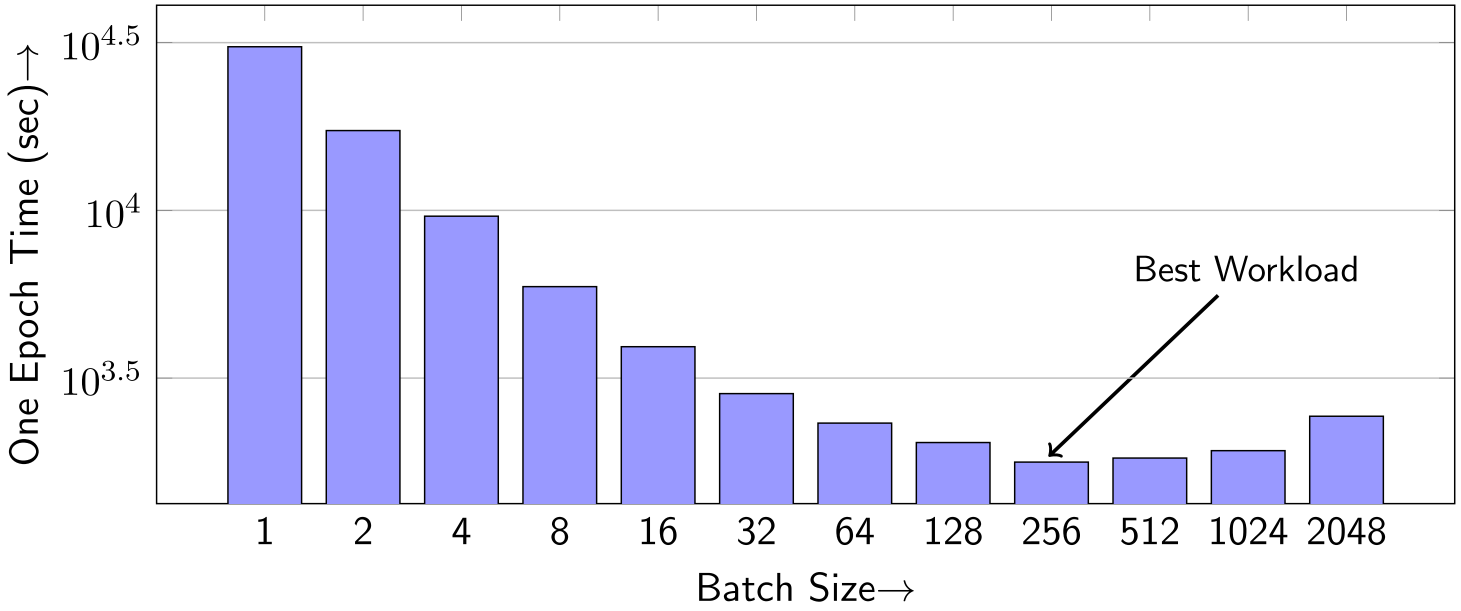
$$T_{comm}(batch) = 2\sum_{i=0}^{L}\left(\alpha(P-1) + \beta\frac{P-1}{P}|W_i|\right)$$

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



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 speed up

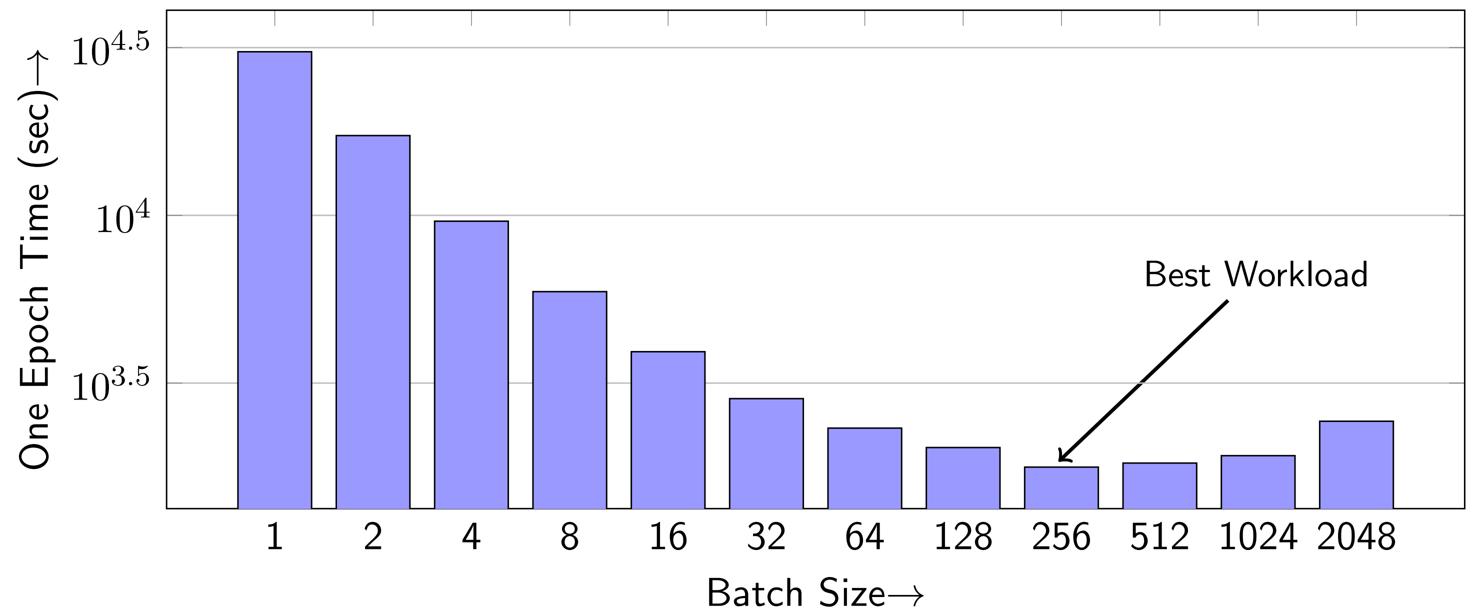


One epoch training time of AlexNet computed on an Intel KNL system

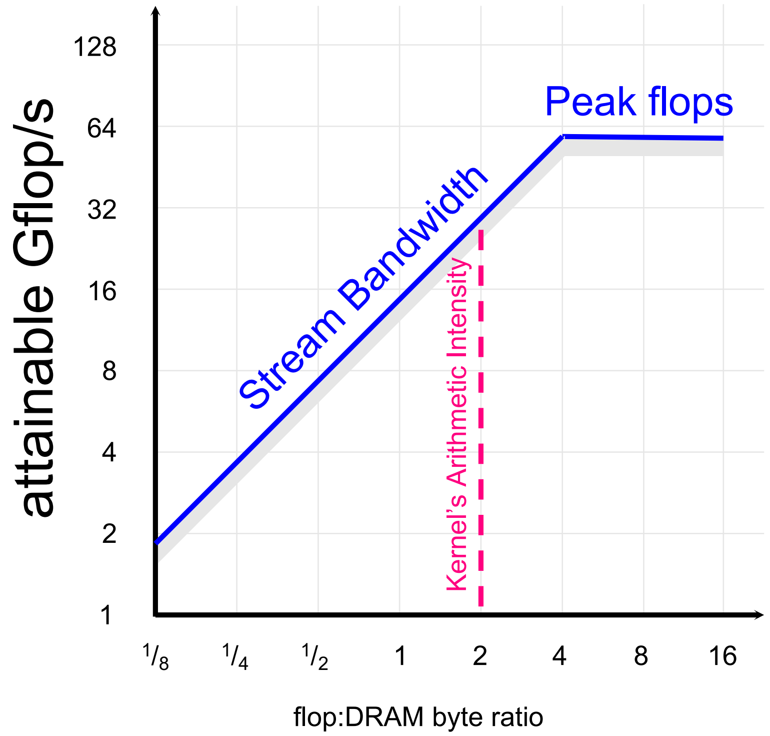
- Why does this happen?
  - Remember roofline model?

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 speed up

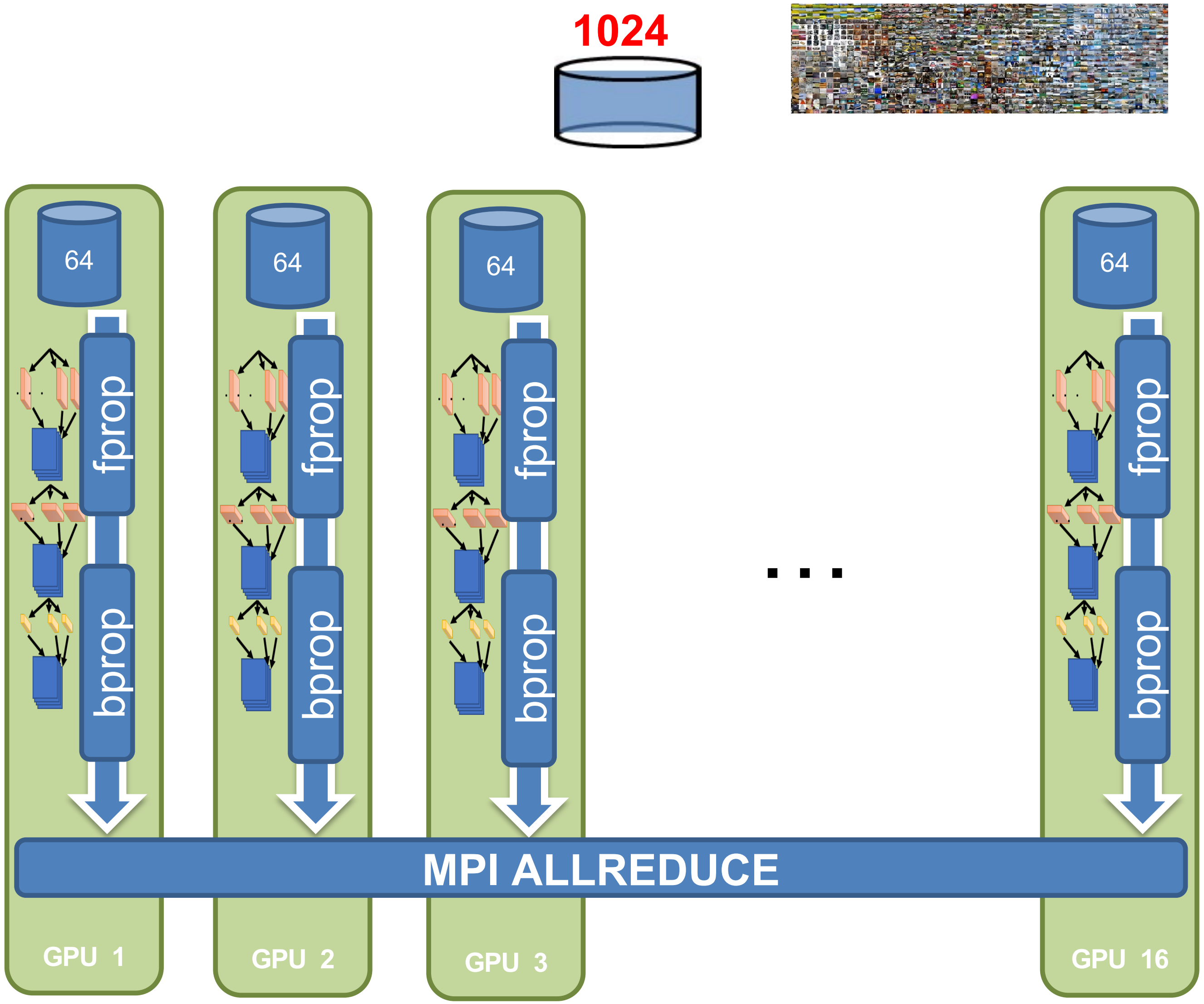


One epoch training time of AlexNet computed on an Intel KNL system

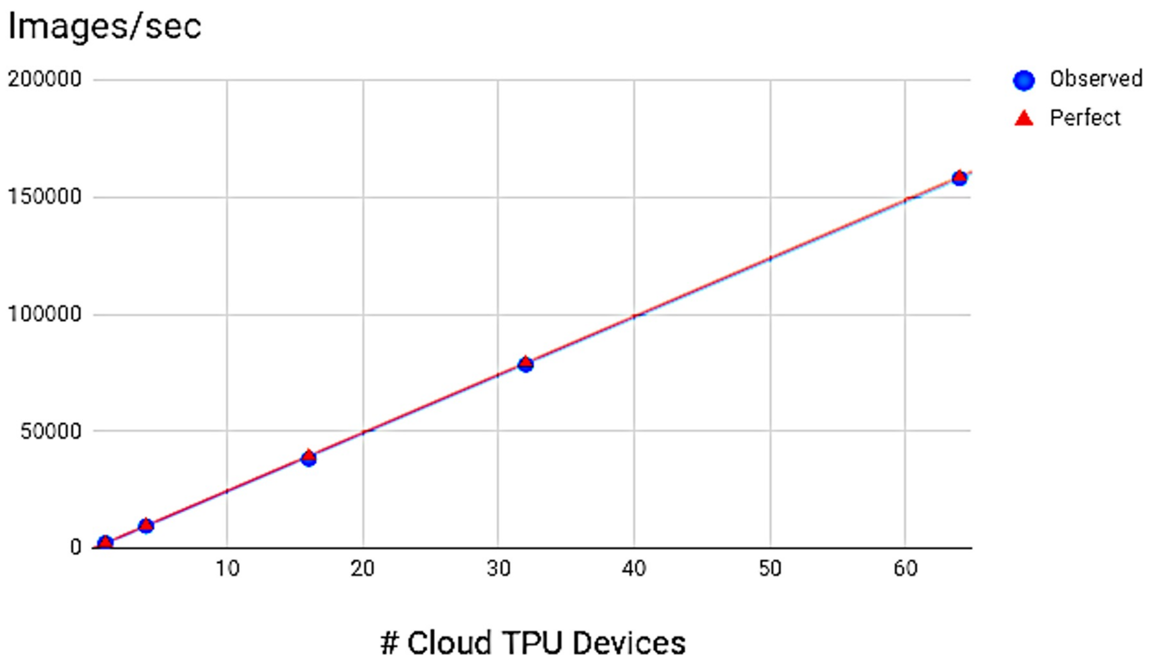


Scaling Data Parallel Training

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

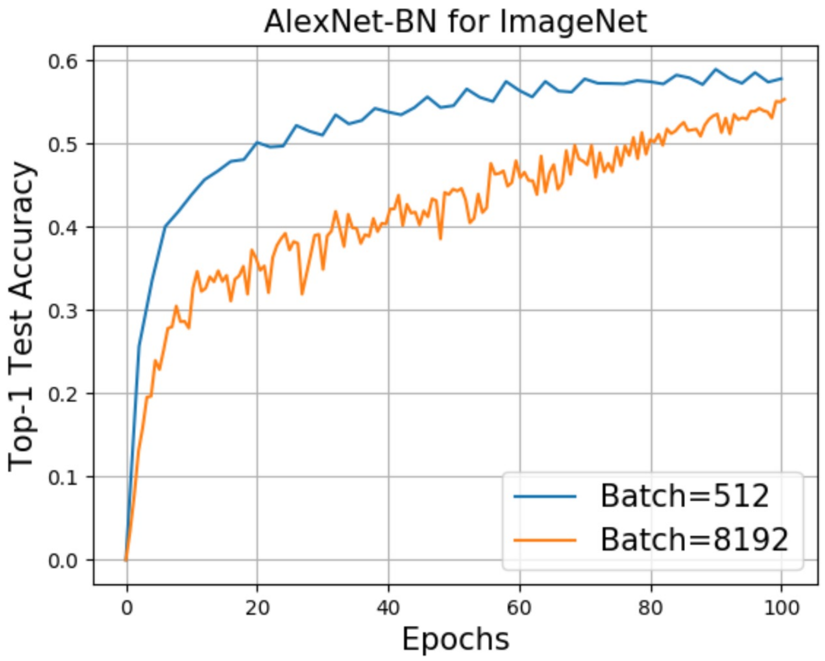
- Motivation for larger batch sizes
  - More opportunities for parallelism → but is it useful?
  - Recall (1/n variance reduction):

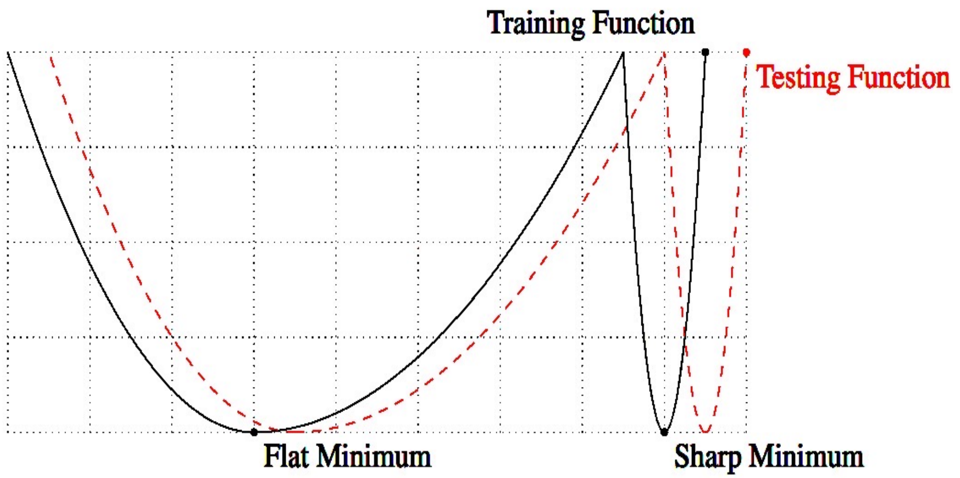
$$\frac{1}{n}\sum_{i=1}\nabla_\theta \mathbf{L}(y_i, f(x_i;\theta)) \approx \frac{1}{|\mathcal{B}|}\sum_{i\in\mathcal{B}}\nabla_\theta \mathbf{L}(y_i, f(x_i;\theta))$$

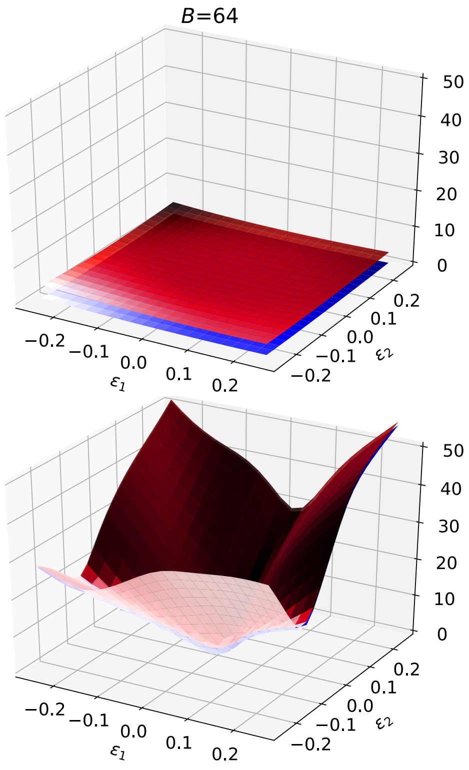
- Is a variance reduction helpful?
  - Only if it let's you take bigger steps (move faster)
  - Does it affect the final prediction accuracy?

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"

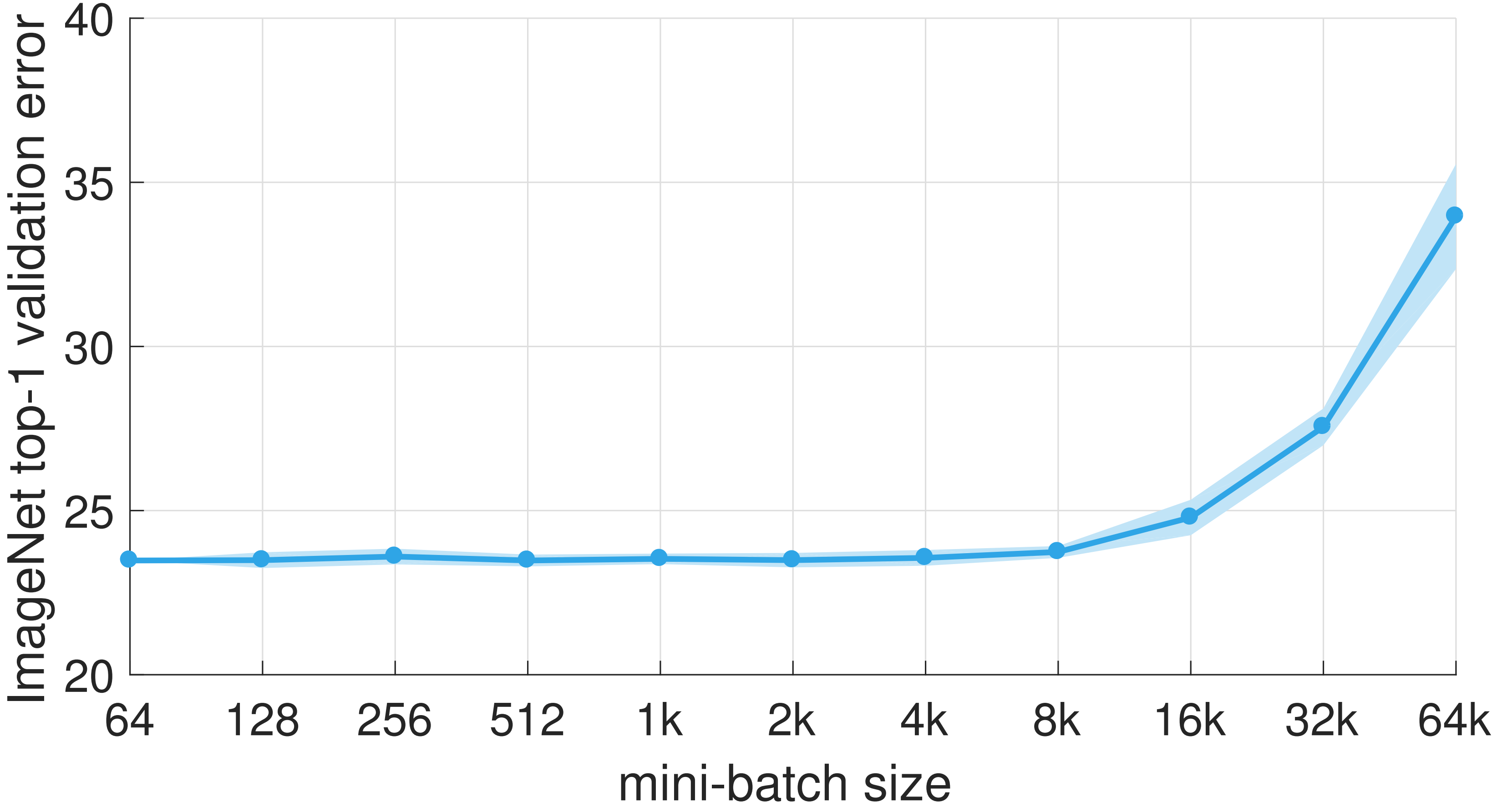






Keskar et al., On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima, ICLR'16.
Z. Yao, A. Gholami, Q. Lei, K. Keutzer, M. Mahoney. Hessian-based Analysis of Large Batch Training and Robustness to Adversaries, NeurIPS'18.
Ginsburg, Boris, Igor Gitman, and Yang You. "Large Batch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

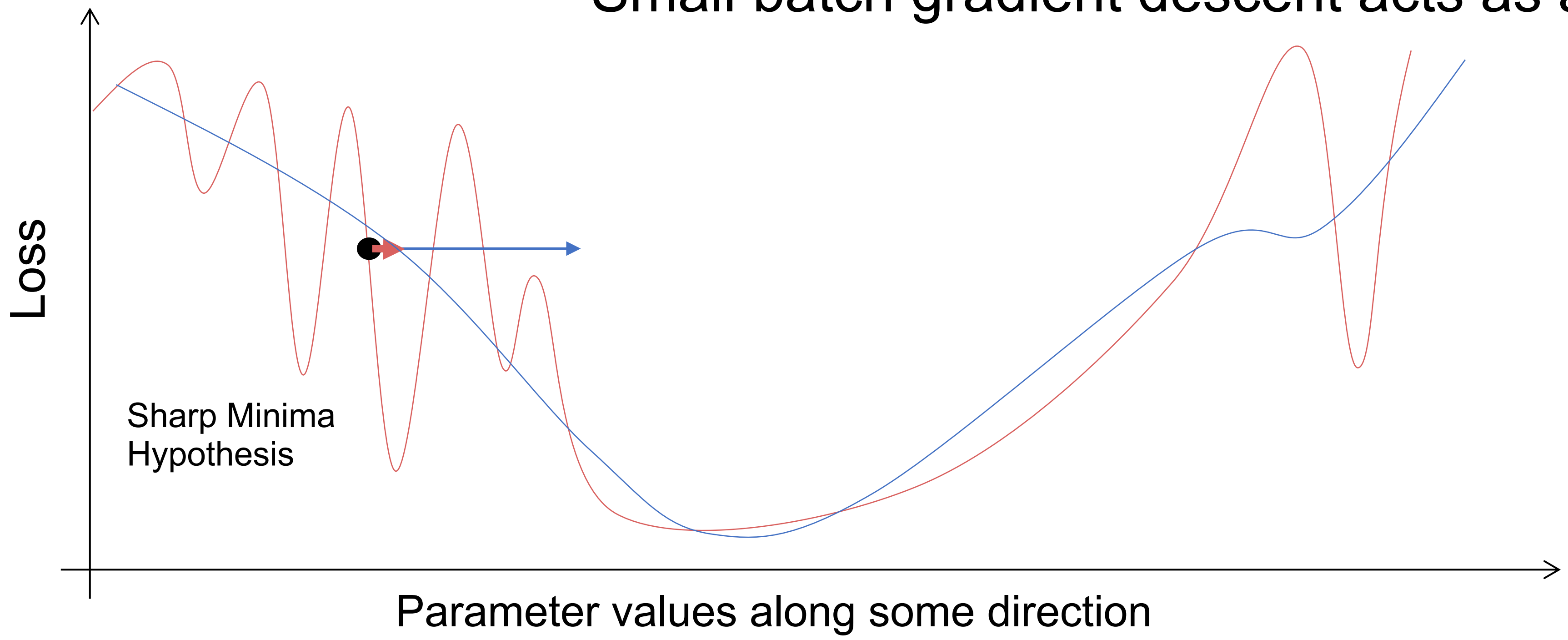
Objective function

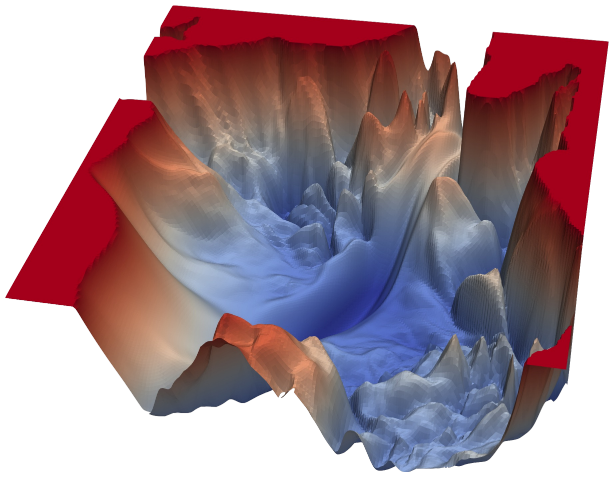
$$L(\theta) = \frac{1}{N}\sum_{i=1}^{N} l(x_i, y_i, \theta)$$

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

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

- 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…

Data Parallelism Summary

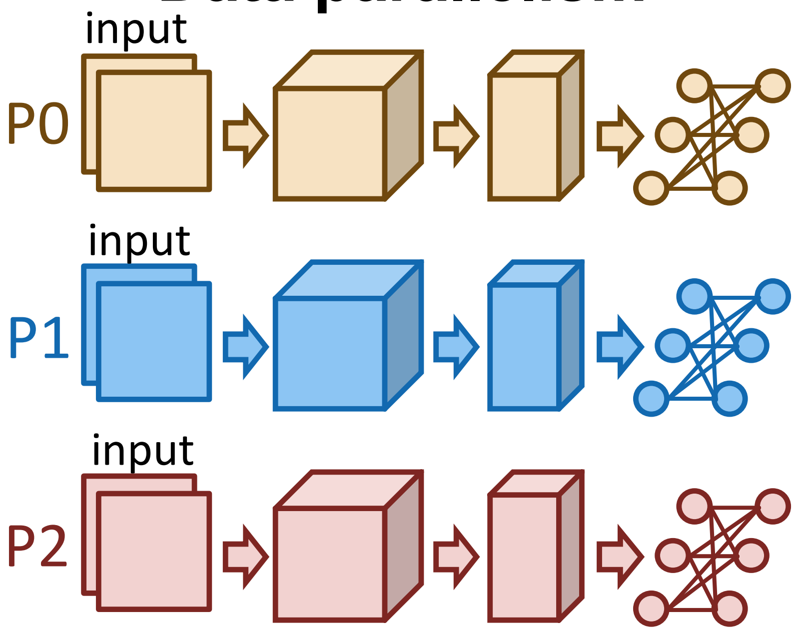
- An efficient parallel training method where the comm time is independent of processors with ring allreduce
- Very easy to implement. Only requires allreduce operation before updating parameters
- Very challenging to scale. Using large batch training is not an option as it hurts generalization performance.
  - Existing solutions often require a lot of tuning (outside of ResNet-50 on ImageNet)
- Does not work for large models such as GPT-3 which are too large to fit in one GPU
- Processes are never idle

Pipeline Parallelism

Really a form of model parallelism

Parallel and distributed training

Data parallelism



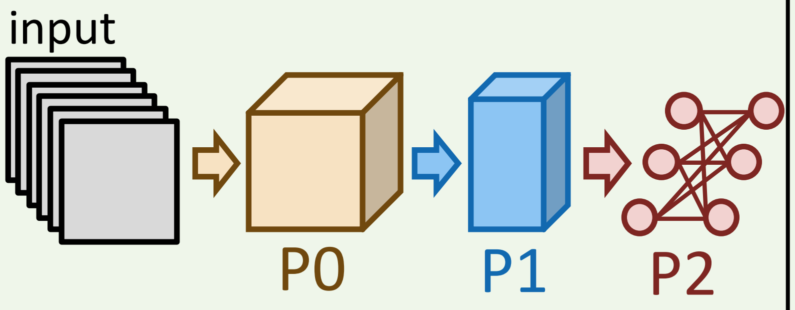
Pros:

a. Easy to realize

Cons:

a. Not work for large models
b. High allreduce overhead

Pipeline parallelism



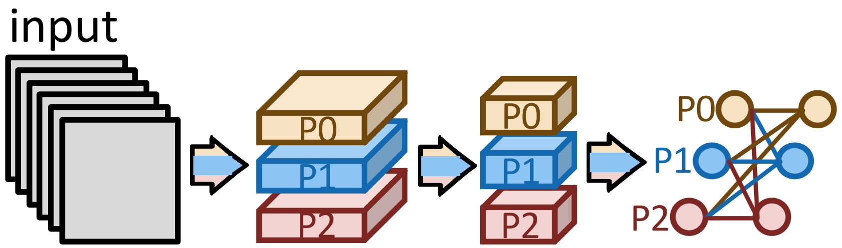
Pros:

a. Make large model training feasible
b. No collective, only P2P

Cons:

a. Bubbles in pipeline
b. Removing bubbles leads to stale weights

Model parallelism



Pros:

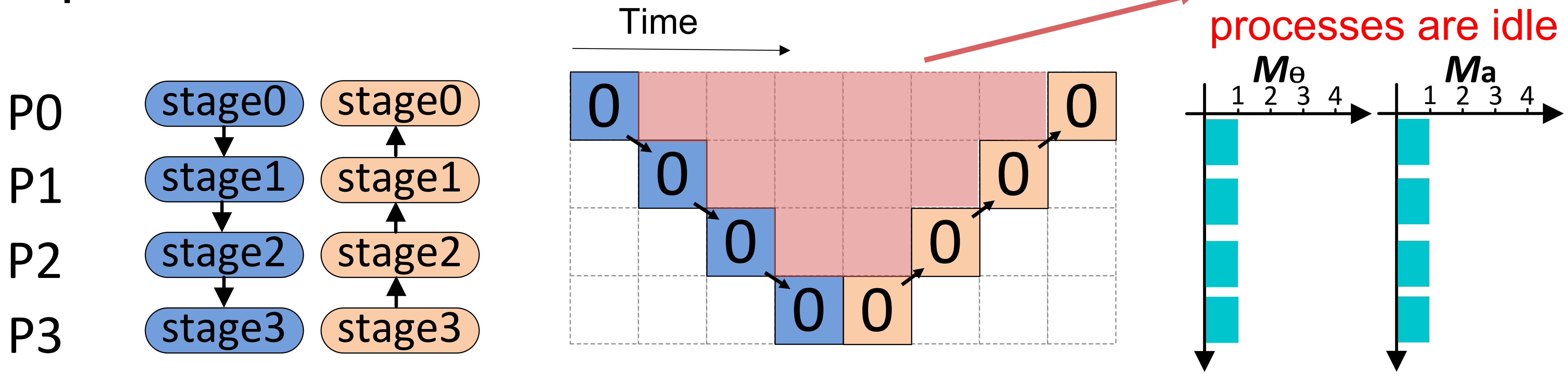
a. Make large model training feasible

Cons:

b. Communication for each operator (or each layer)

Slide: Courtesy of Shigang Li

Pipeline Parallelism



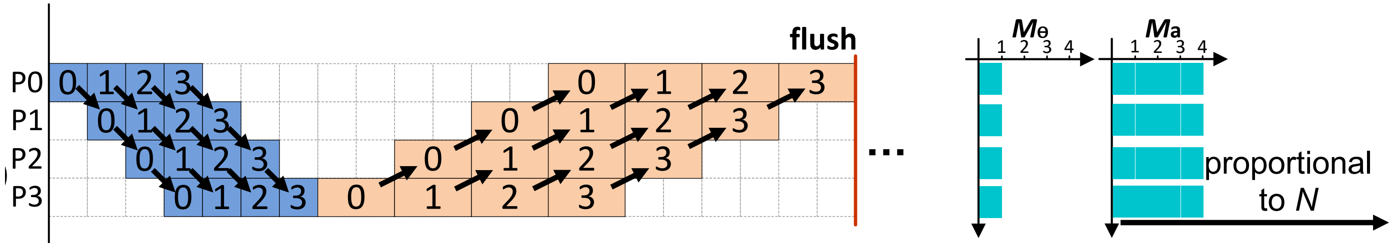
Bubble

x x Forward and backward passes of ***model replica*0** for micro-batch **x**

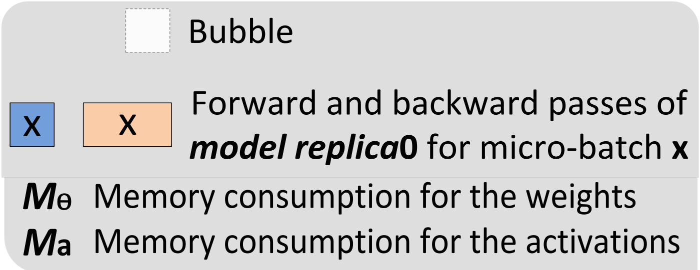
***M*θ** Memory consumption for the weights

***M*a** Memory consumption for the activations

GPipe [NeurIPS'19]: Reduce Bubble with Micro-Batching

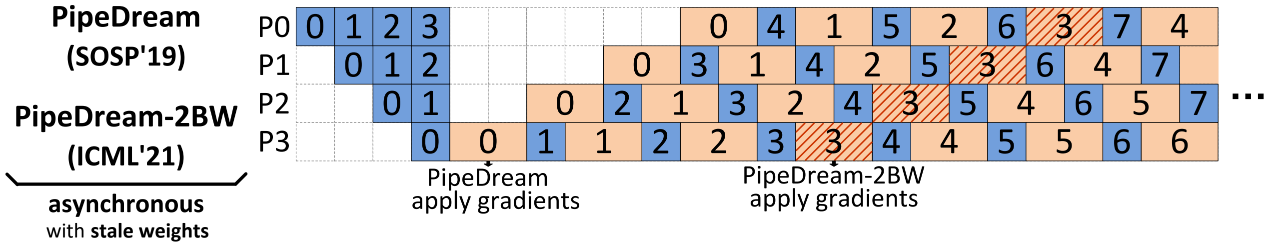


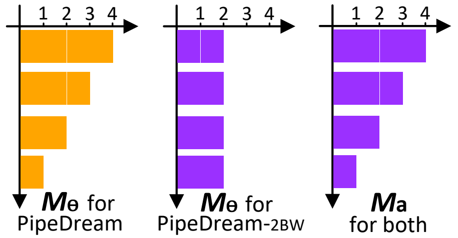
- GPipe reduces the bubble size by breaking the batch size into smaller pieces to reduce the idle time of the processes
- Pro: Reduces bubble size in an easy to implement manner
- Con: Significantly increases activation memory



Slide: Courtesy of Shigang Li

PipeDream[SOSP'19]: Use Async Updates to remove Bubble





- Pipedream uses asynchronous training: Avoid any idling by always doing a forward/backward pass irrespective of stale gradients/weights
- Pro: No bubble
- Con: As with other async methods this does affect model accuracy and convergence, and as such has not been adopted in industry.

Slide: Courtesy of Shigang Li

Asynchronous Methods

- General advice: Training methods that adversely affect generalization are not adopted, unless there is a 10x speed improvement.

- Otherwise, there are so many moving parts that can go wrong in training NNs, that most often practitioners stay away from async methods unless absolutely necessary
  - For example training very large rec systems.

Pipeline Parallelism Summary

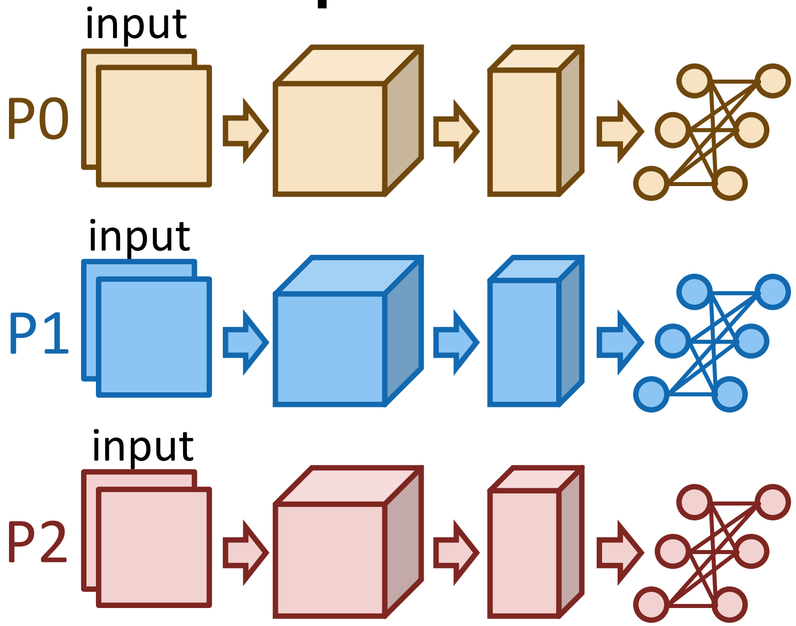
- Slightly more involved algorithm than data parallel method but with the advantage of only requiring point to point communication
- Ideal for large scale training to thousands of processes where point-to-point communication is much cheaper than collective operations such as allreduce or all-gather
- Requires special handling of bubble that results in idle processes

Model Parallelism

AKA Operator Parallelism

Parallel and distributed training

Data parallelism



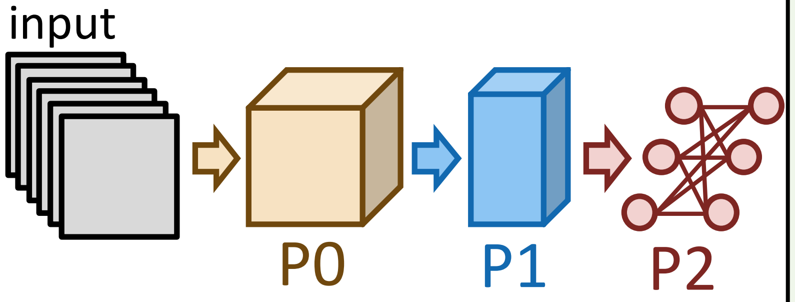
Pros:

a. Easy to realize

Cons:

a. Not work for large models
b. High allreduce overhead

Pipeline parallelism



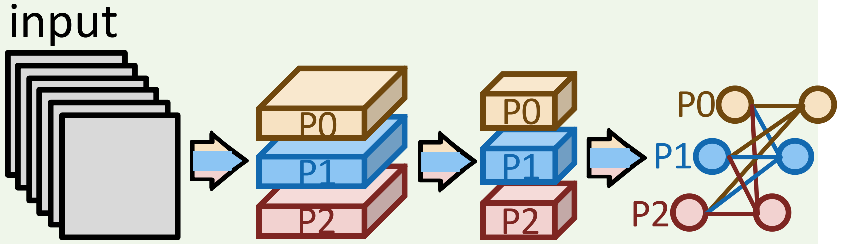
Pros:

a. Make large model training feasible
b. No collective, only P2P

Cons:

a. Bubbles in pipeline
b. Removing bubbles leads to stale weights

Model parallelism



Pros:

a. Make large model training feasible

Cons:

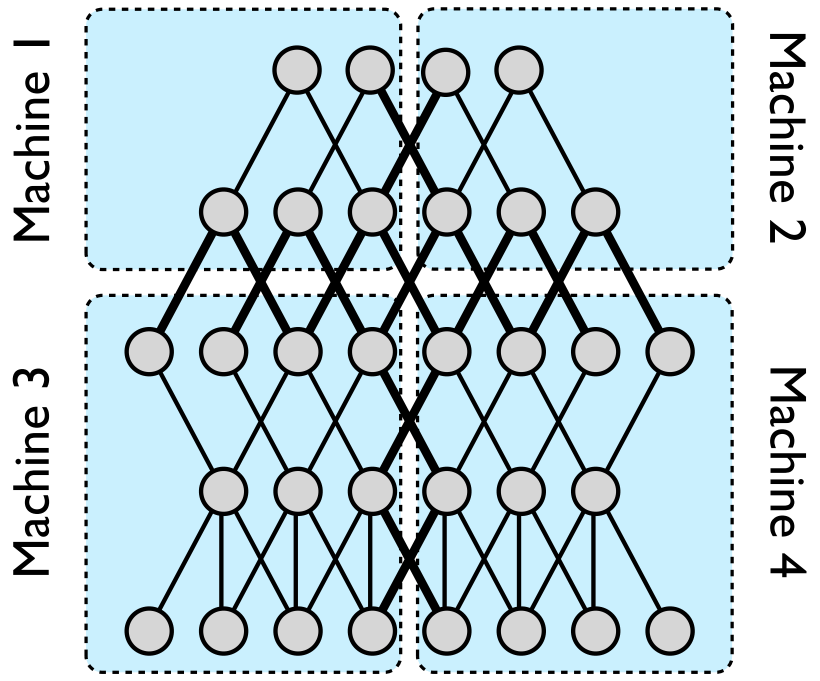
b. Communication for each operator (or each layer)

Slide: Courtesy of Shigang Li

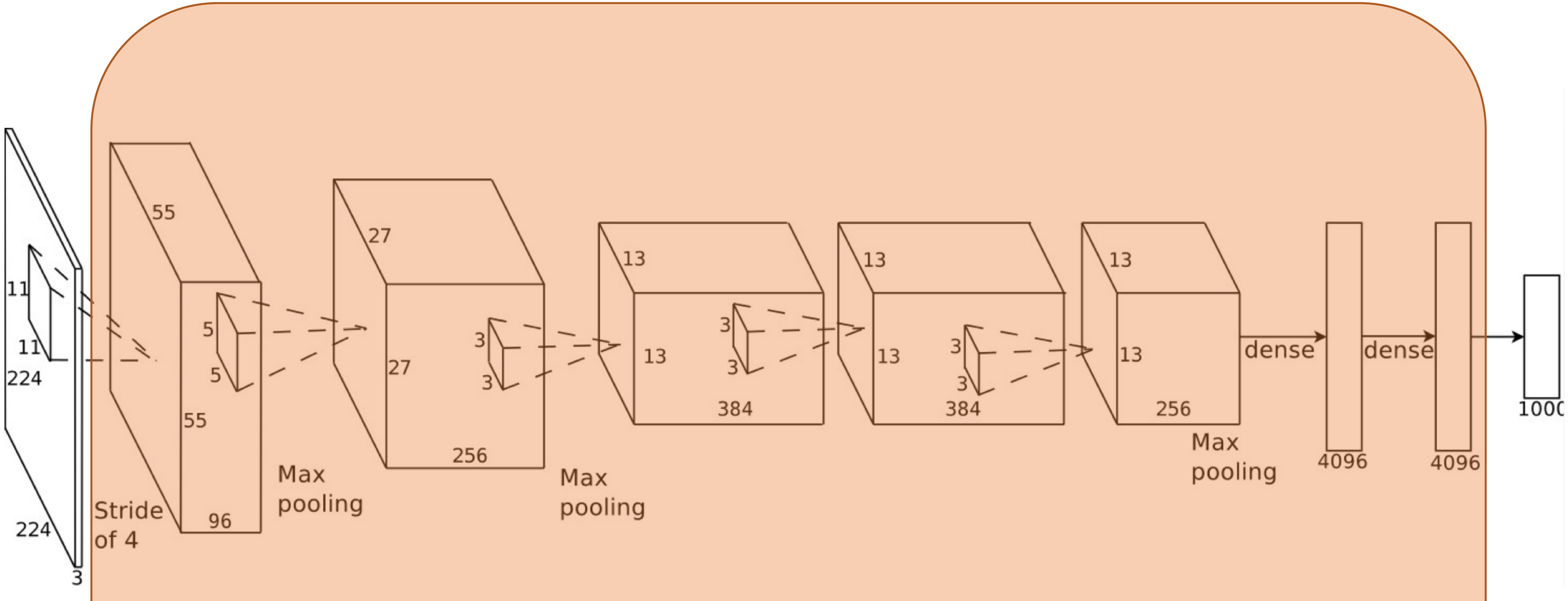
Model Parallelism

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 across layers
    - Only one set of layers active a time → poor work balance
    - This is basically pipeline parallelism
  - Split individual layers
    - which dimension?
      - Weights or spatial → depends on operation



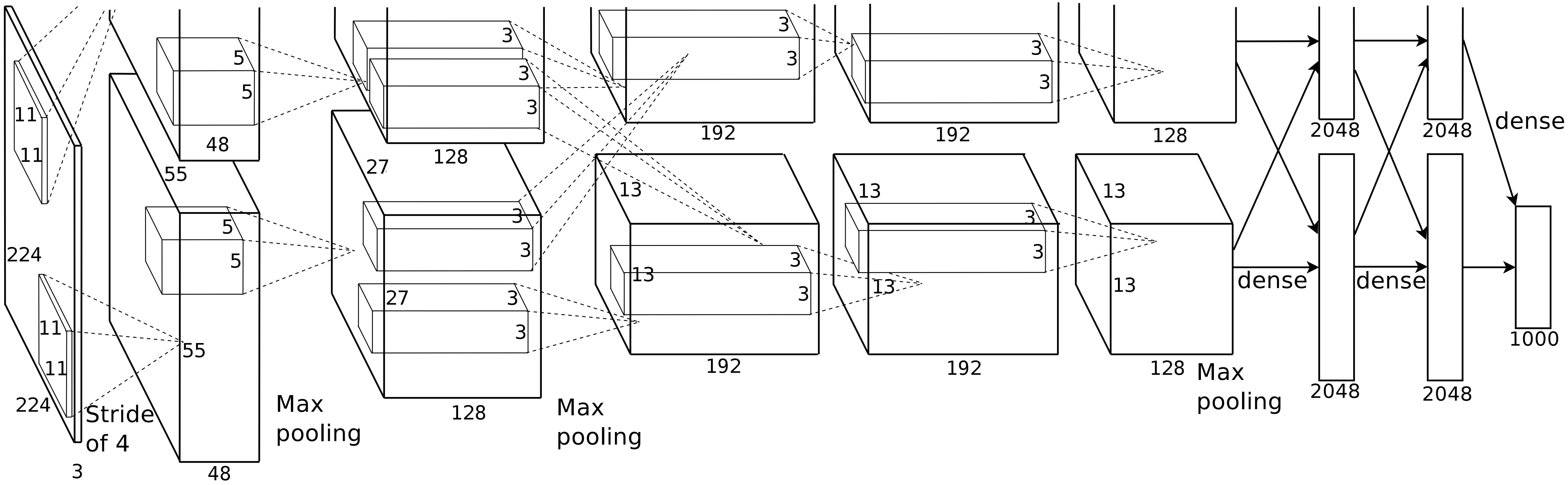
The AlexNet Architecture



Without GPU Partitioning

The **Actual** AlexNet Architecture

From the paper “ImageNet Classification with Deep Convolutional Neural Networks”



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

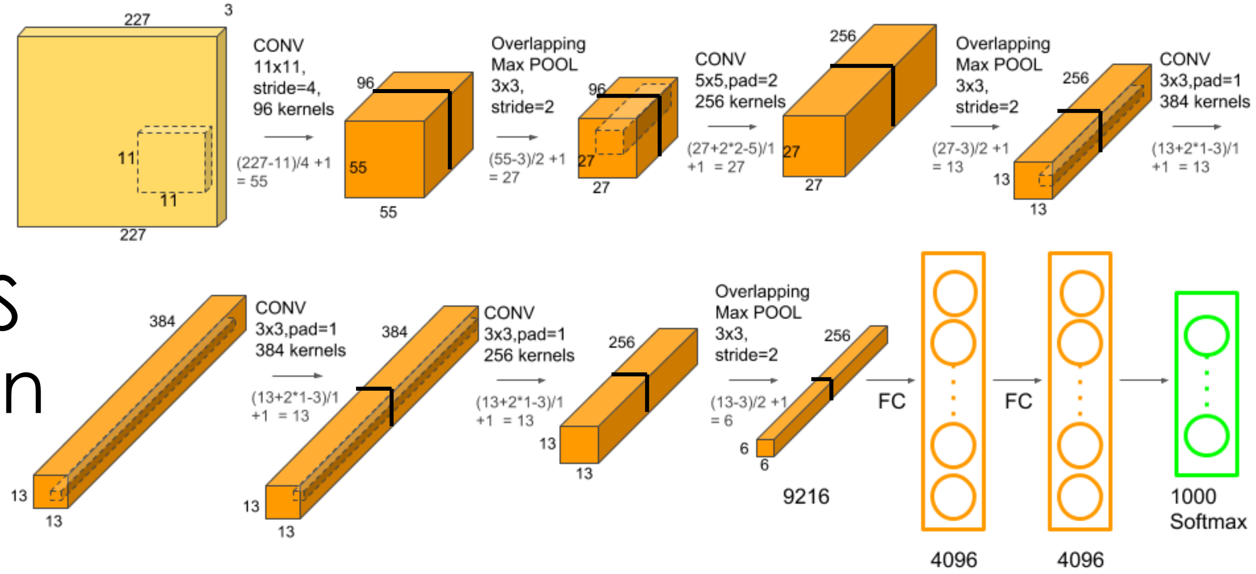
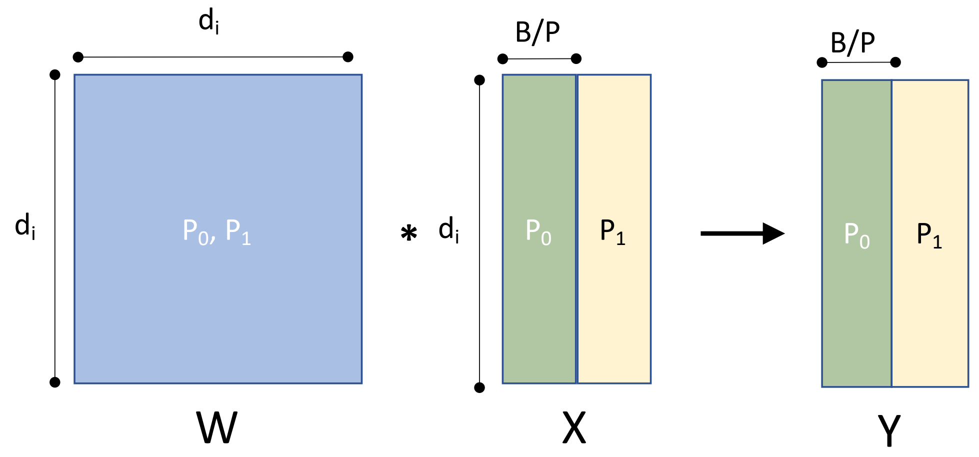


Image from https://neurohive.io/en/popular-networks/alexnet-imagenet-classification-with-deep-convolutional-neural-networks/

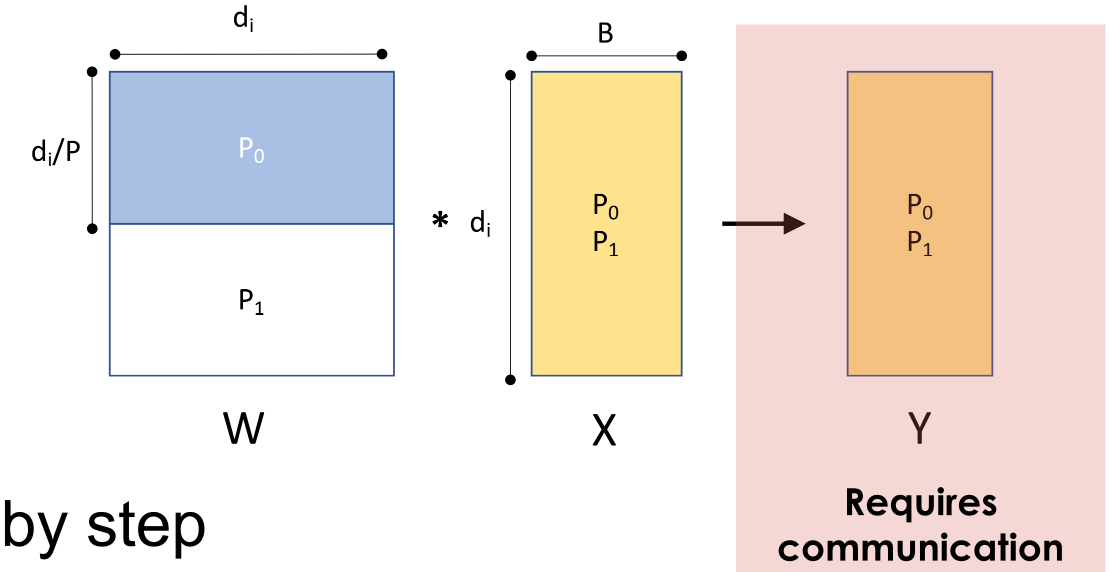
Model Parallelism: Comm Analysis

It helps to think of the operations in matrix form. Consider an FC layer

Data Parallelism: Partition input across different Processors (batch dimension)

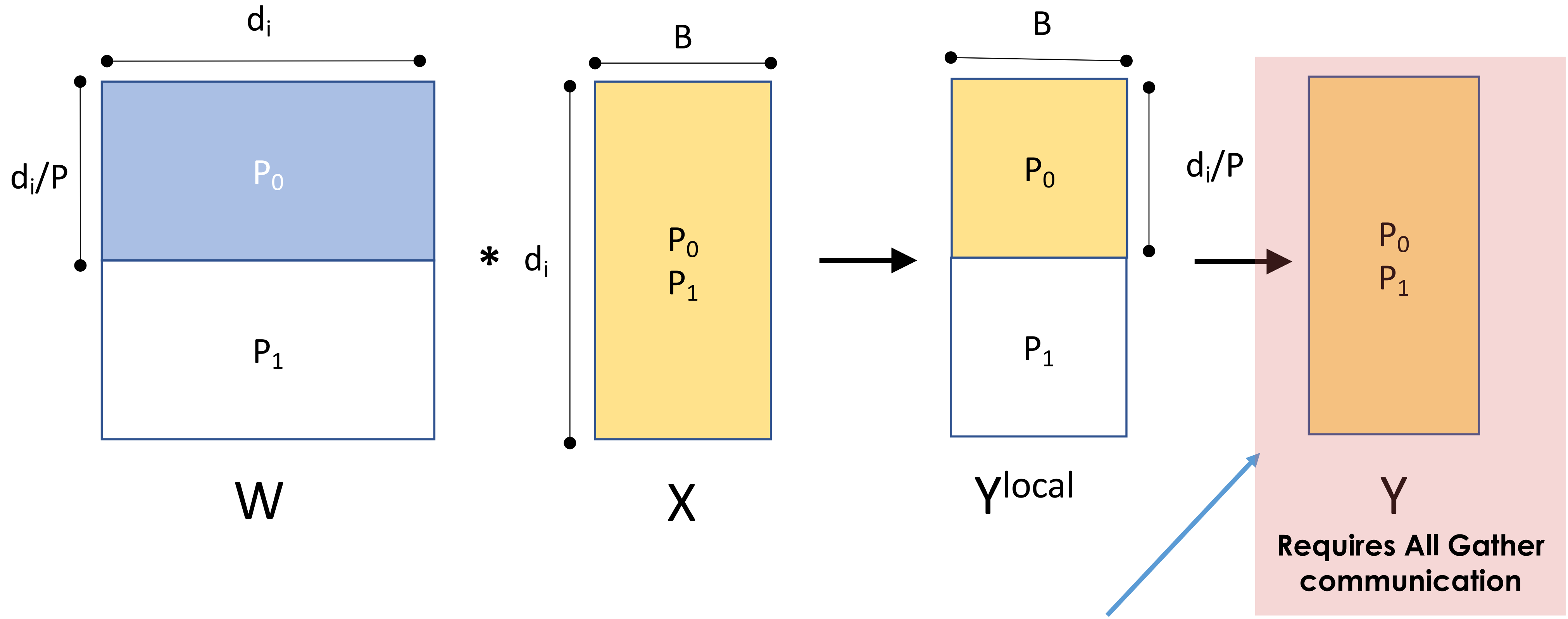


Model Parallelism: Partition weights across different Processes (W dimension)



Let's discuss the communication details, step by step

Comm Analysis: Forward Pass



- Requires an all gather communication so that all processes get each others activation data
- Same cost as all reduce without the 2x factor

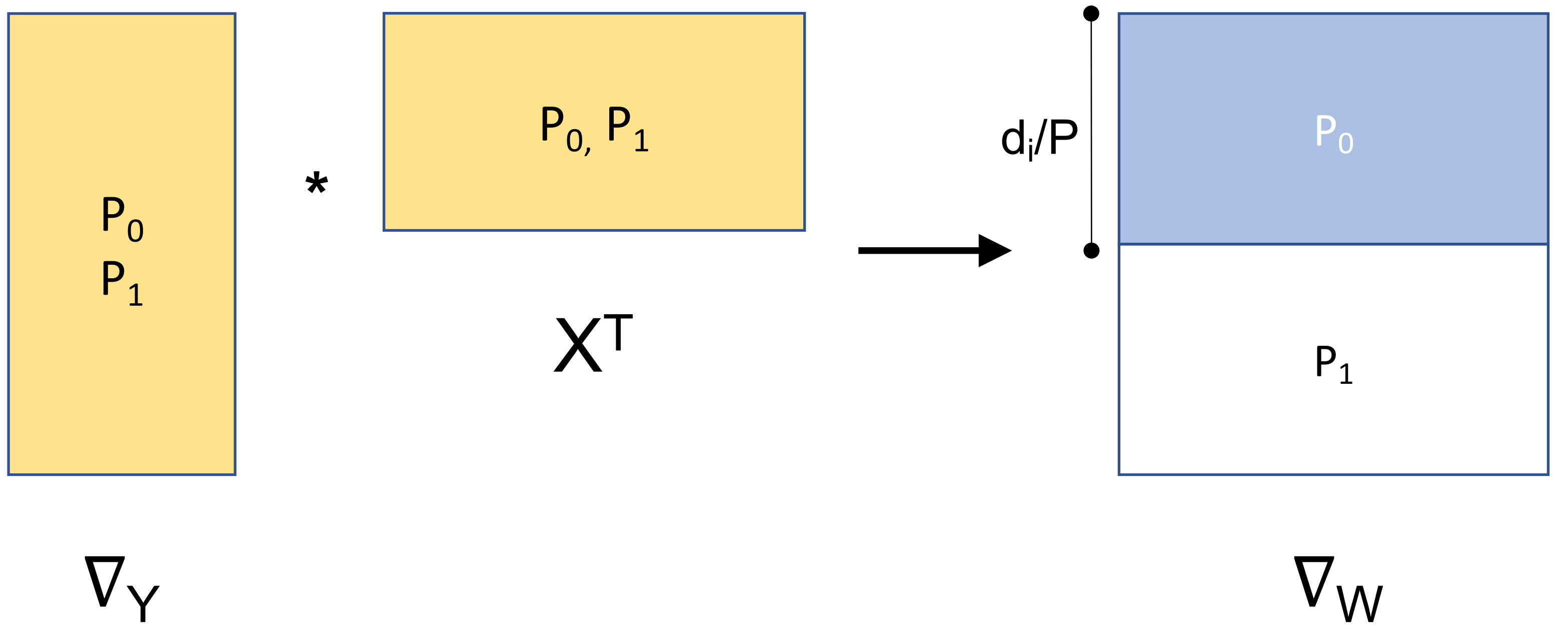
$$\sum_{i=1}^{L}\left(\beta(P-1)\frac{Bd_i}{P}\right)$$

* Ignoring latency term for notational simplicity

Backward Pass: Weights

$\nabla_Y * X^T = \nabla_W$

$W^T * \nabla_Y = \nabla_X$

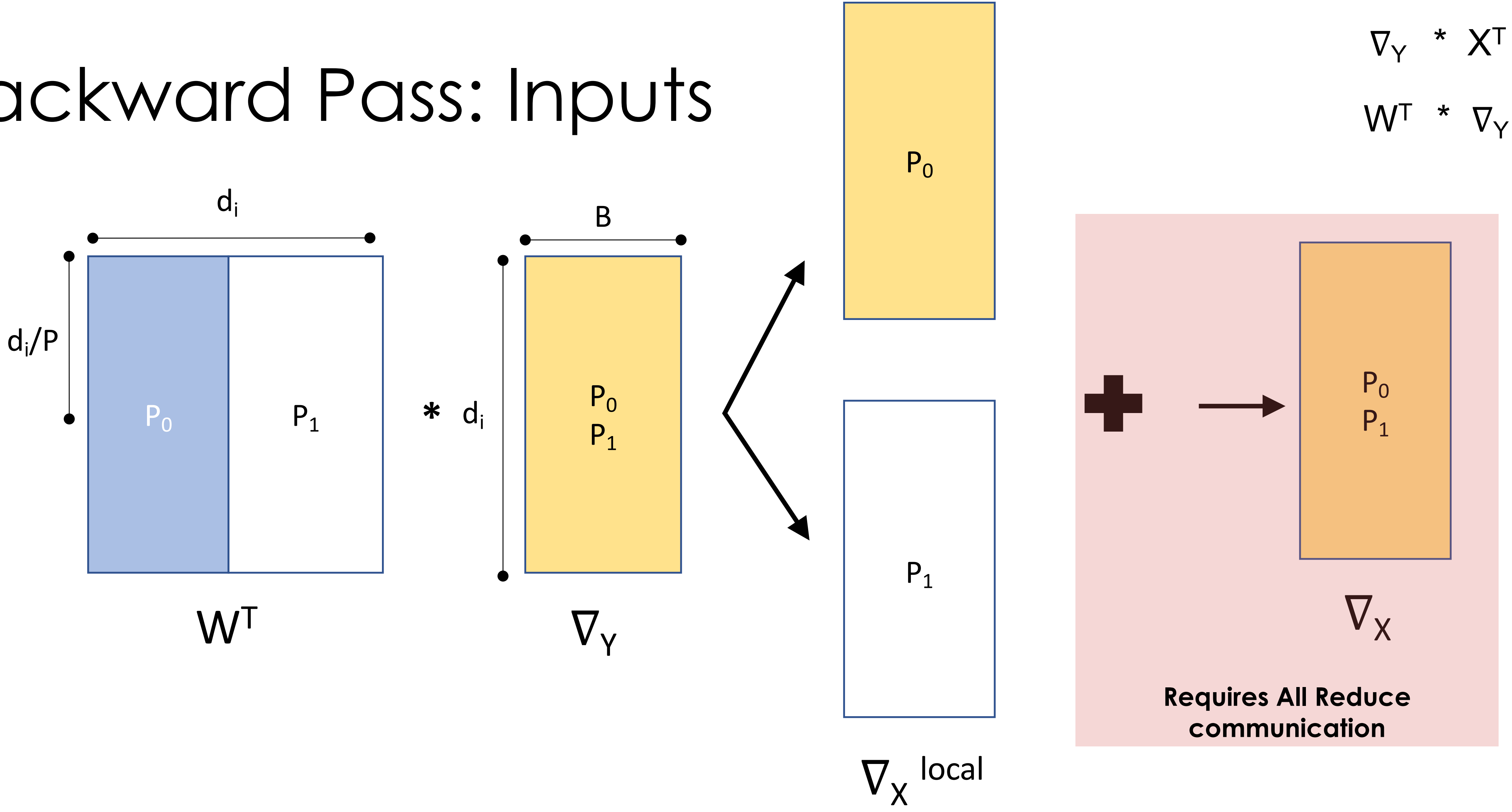


- No communication needed as every processor only needs the gradient of its own parameters
  - This makes model parallelism very effective for cases where the model size is large

Backward Pass: Inputs

$\nabla_Y * X^T = \nabla_W$

$W^T * \nabla_Y = \nabla_X$



- Aggregating activation delta requires an allreduce operation

$$2\sum_{i=2}^{L}\left(\beta(P-1)\frac{Bd_i}{P}\right)$$

Comm Complexity Analysis

In Model Parallelism we need two forms of communication:

1. All Gather operation so that all processors get all the activations
2. All reduce operation for backpropagating activation gradients

$$T_{comm}(model) = \underbrace{\sum_{i=1}^{L}\left(\beta(P-1)\frac{Bd_i}{P}\right)}_{\text{All Gather}} + \underbrace{2\sum_{i=2}^{L}\left(\beta(P-1)\frac{Bd_i}{P}\right)}_{\text{All Reduce}}$$

Model vs Data Parallelism?

- When does it make sense to use Model vs Data Parallelism?

$$T_{comm}(model) = \sum_{i=1}^{L}\left(\beta(P-1)\frac{Bd_i}{P}\right) + 2\sum_{i=2}^{L}\left(\beta(P-1)\frac{Bd_i}{P}\right)$$

$$T_{comm}(data) = \sum_{i=1}^{L}\left(\beta(P-1)\frac{d_i^2}{P}\right)$$

- Model parallelism reduces the quadratic complexity of d_i
  - It is useful for layers with very large weights $d_i \gg 1$

- It makes sense to use an integrated/hybrid data and model parallelism

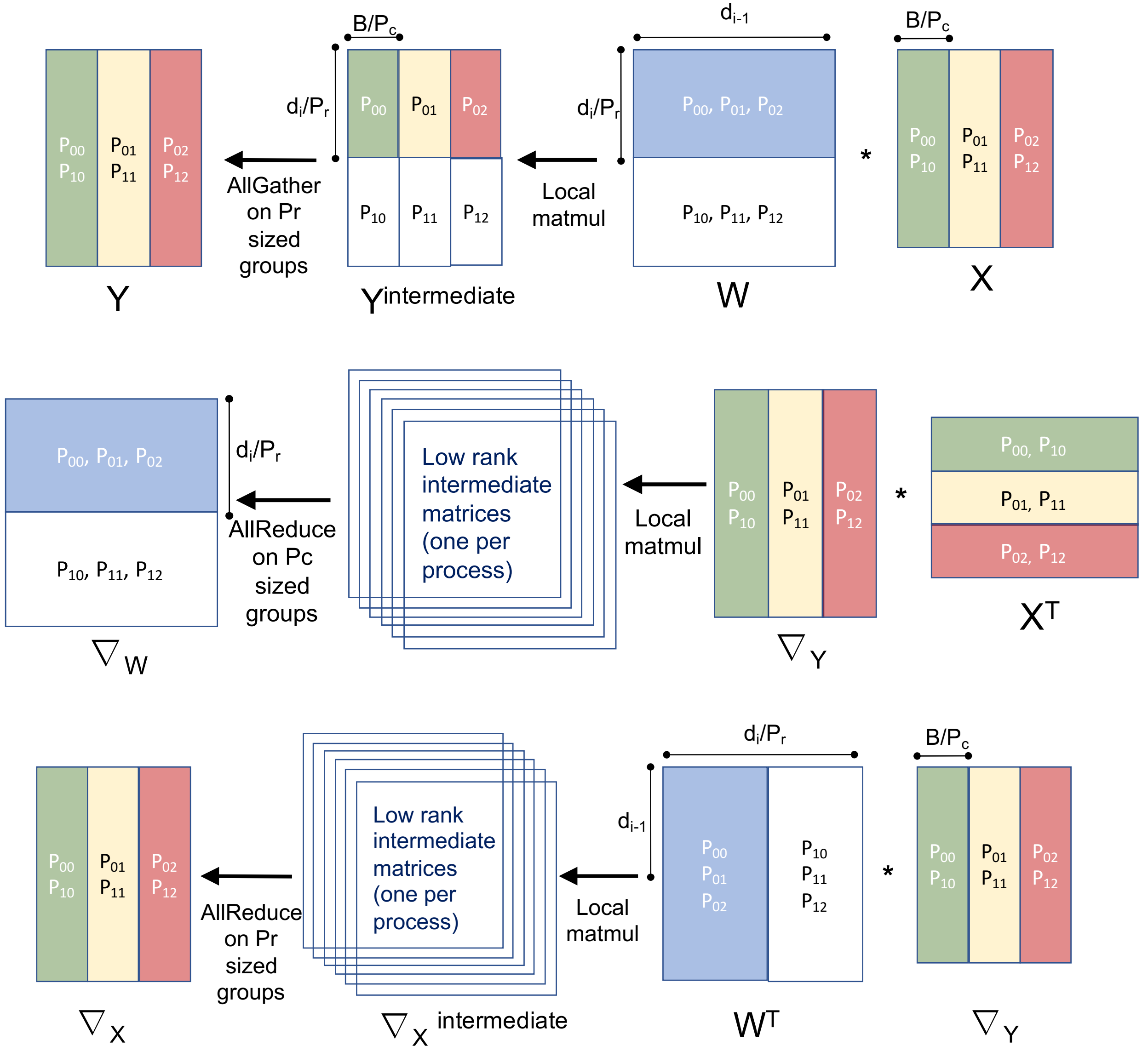
Gholami, Amir, Ariful Azad, Peter Jin, Kurt Keutzer, and Aydin Buluc. "Integrated model, batch, and domain parallelism in training neural networks." SPAA, 2018.

Model Parallelism Summary

- Has better comm complexity for **large** FC layers than Data parallel approach
- Makes training large models feasible by breaking it into smaller parts
- However, requires **blocking collective communication** during **both** forward pass (all gather), as well as backwards pass (all reduce)
- Slightly **harder to implement** than data/pipeline parallel

Integrated Model and Data Parallelism

For a linear graph we can find the optimal hybrid method for analyzing the communication complexity, coupled with hardware utilization [1]



Processes are 2D indexed: **$P = P_r \times P_c$**

[1] Gholami, Amir, Ariful Azad, Peter Jin, Kurt Keutzer, and Aydin Buluc. "Integrated model, batch, and domain parallelism in training neural networks." SPAA, 2018.

General Hybrid Methods

For a general computational graph we need to decide on:

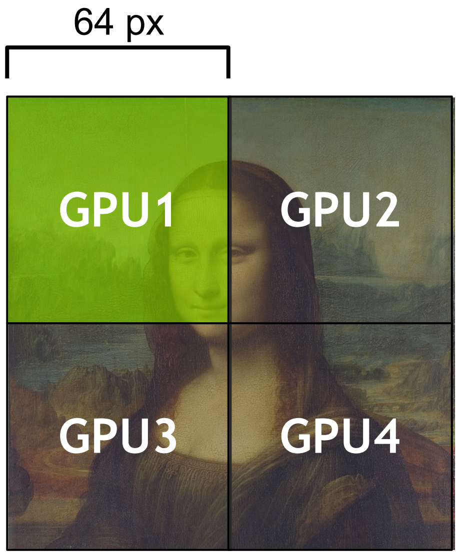
- How many processes to assign for DP
- Which axes to break the model: operator vs pipeline
- How to efficiently map the GPUs to the resulting execution graph
- …

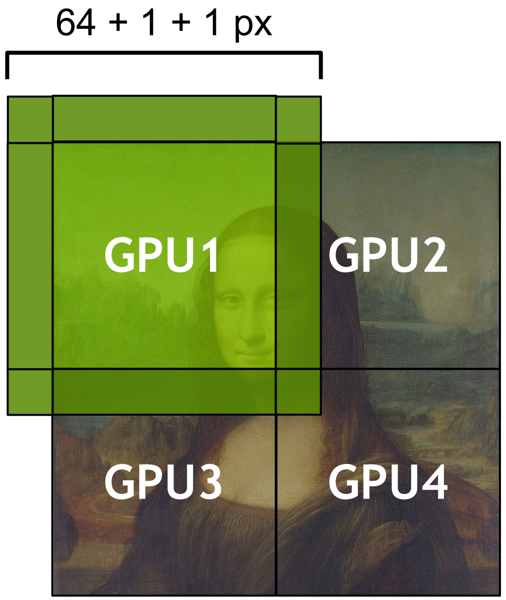
For a general non-linear graph this leads to a combinatorically large search space

Spatial Parallelism

Spatial Parallel Training

- The general idea is to break the input into smaller pieces and distribute the work among different processors
  - Need to exchange boundary points for spatial convolutions

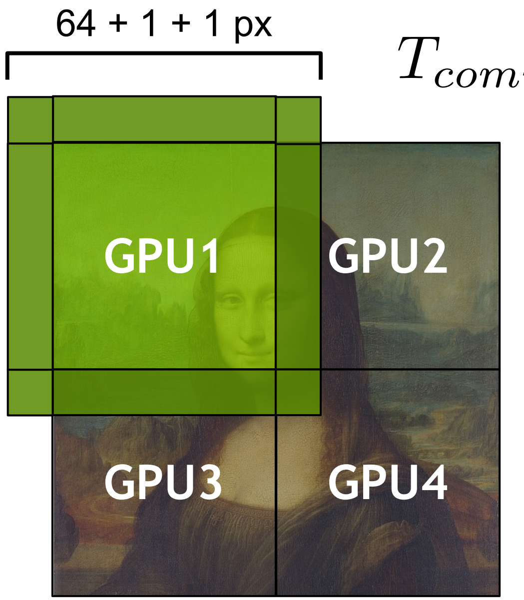




$$T_{comm}(domain) = \sum_{i=1}^{L} (\alpha + \beta B X_W^i X_C^i k_h^i / 2) + \sum_{i=1}^{L} (\alpha + \beta B Y_W^i Y_C^i k_w^i / 2) + 2 \sum_{i=1}^{L} \left(\alpha \log(P) + \beta \frac{P-1}{P} |W_i| \right)$$

Peter Jin, Boris Ginsburg, and Kurt Keutzer. "Spatially Parallel Convolutions" ICLR Workshop Track, 2018

Communication Complexity



$$T_{comm}(domain) = \sum_{i=0}^{L} (\alpha + \beta B X_W^i X_C^i k_h^i / 2)$$

Exchanging horizontal pixels

$$+ \sum_{i=0}^{L} (\alpha + \beta B Y_W^i Y_C^i k_w^i / 2)$$

Exchanging vertical pixels

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

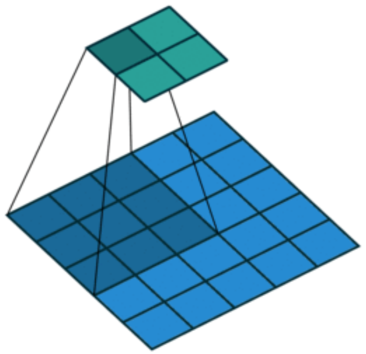
All reduce Cost (same as before)

Peter Jin, Boris Ginsburg, and Kurt Keutzer. "Spatially Parallel Convolutions" ICLR Workshop Track, 2018.
Gholami, Amir, Ariful Azad, Peter Jin, Kurt Keutzer, and Aydin Buluc. "Integrated model, batch, and domain parallelism in training neural networks." SPAA, 2018.

Useful for High Resolution Training

- Domain parallel scaling on V100 GPUs
  - 3x3 Conv, Batch=32, Channel=64

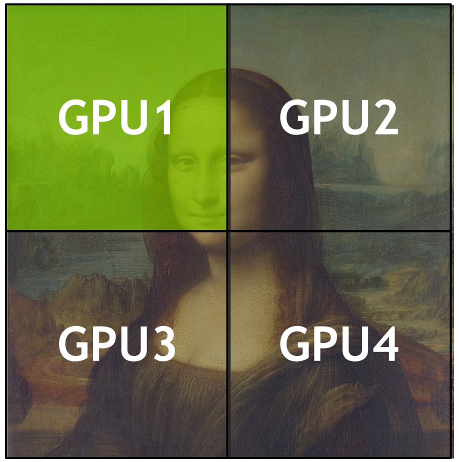
Resolution	GPUs	Fwd. wall-clock	Bwd. wall-clock
128×128	1	2.56 ms ($1.0\times$)	6.63 ms ($1.0\times$)
	2	1.52 ms ($1.7\times$)	3.50 ms ($1.9\times$)
	4	**1.23 ms ($2.1\times$)**	**2.33 ms ($2.8\times$)**
256×256	1	10.02 ms ($1.0\times$)	26.81 ms ($1.0\times$)
	2	5.34 ms ($1.9\times$)	11.79 ms ($2.3\times$)
	4	**3.11 ms ($3.2\times$)**	**6.96 ms ($3.9\times$)**
512×512	1	45.15 ms ($1.0\times$)	126.11 ms ($1.0\times$)
	2	20.18 ms ($2.2\times$)	60.15 ms ($2.1\times$)
	4	**10.65 ms ($4.2\times$)**	**26.76 ms ($4.7\times$)**



Peter Jin, Boris Ginsburg, and Kurt Keutzer. "Spatially Parallel Convolutions" ICLR Workshop Track, 2018
Figure from: Dumoulin, V., Visin, F.. A guide to convolution arithmetic for deep learning. *arXiv:1603.07285*, 2016.

Spatial Parallelism Summary

- A little harder to implement since you need to exchange the boundary points
- Only effective for high resolution input data
  - Limits the number of processors that can be effectively utilized



Acknowledgments

Many slides from

Prof. Joseph E. Gonzalez (UCB), Prof. Kurt Keutzer (UCB), Prof. Patterson (UCB), Michael Pellauer (Nvidia), Prof. Sophia Shao, Naveen Kumar (Google), Shigang Li (ETH)