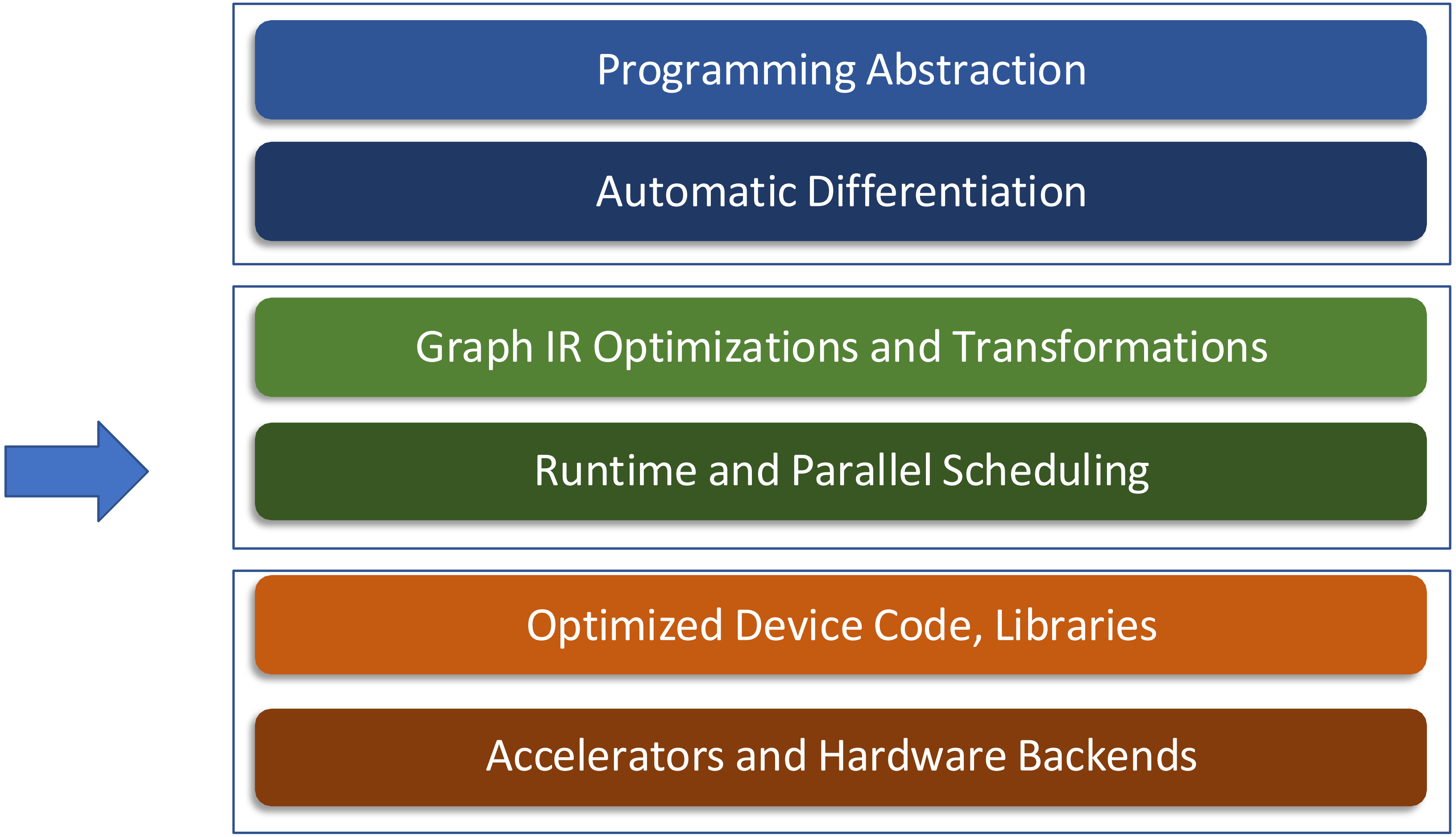
CMSC818Q: Special Topics in Cloud Networking and Computing

Distributed Training

Instructor: Alan Liu

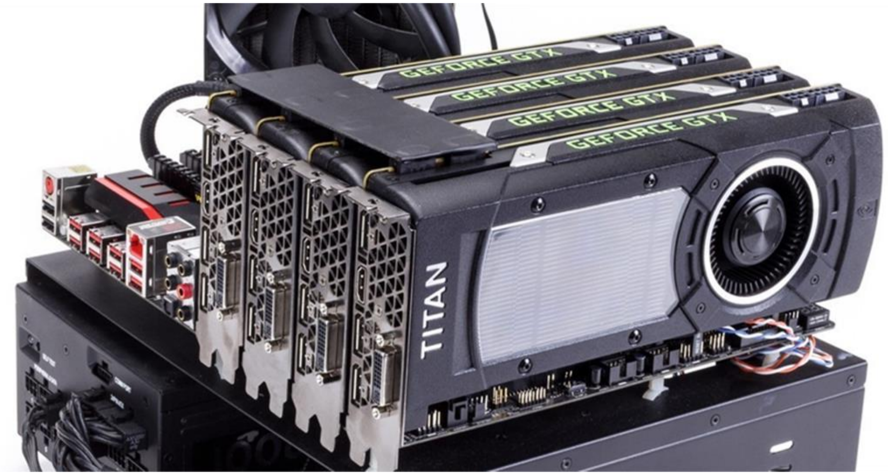
UNIVERSITY OF MARYLAND 1856
DEPARTMENT OF COMPUTER SCIENCE

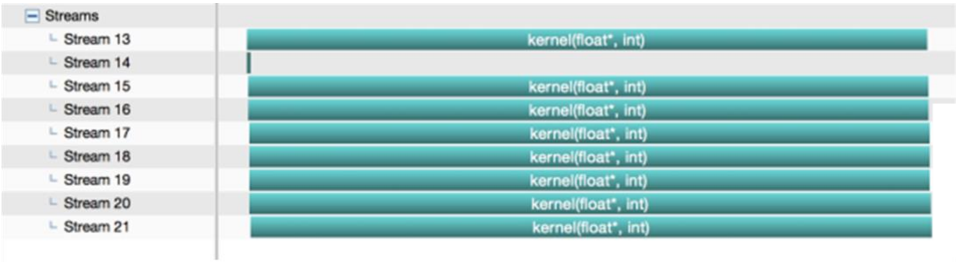
A Typical Deep Learning System Stack



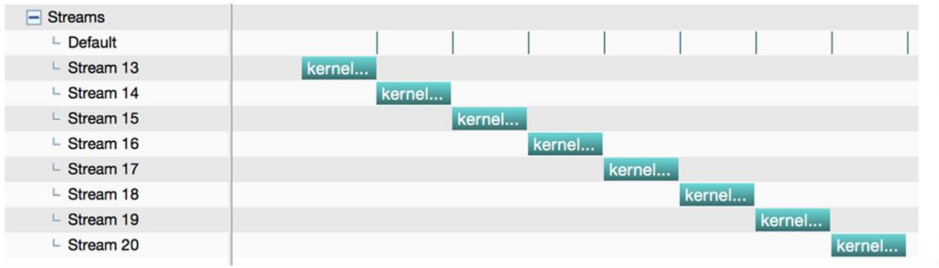
Parallelization Problem

- Parallel execution of concurrent kernels
- Overlap compute and data transfer





Parallel over multiple streams



Serial execution

Objectives For Today

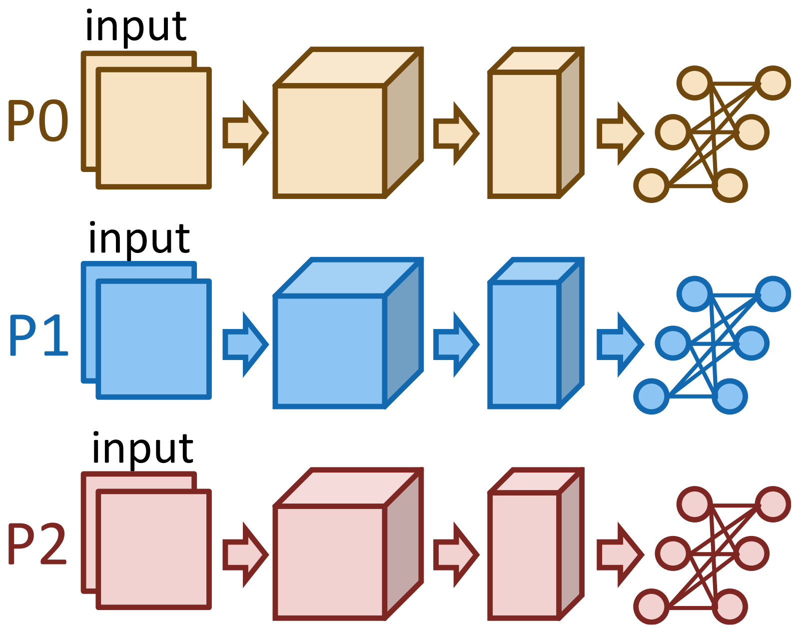
Challenges with Data Parallel Training

Model Parallelism

Pipeline Parallelism

Parallel and distributed training

Data parallelism



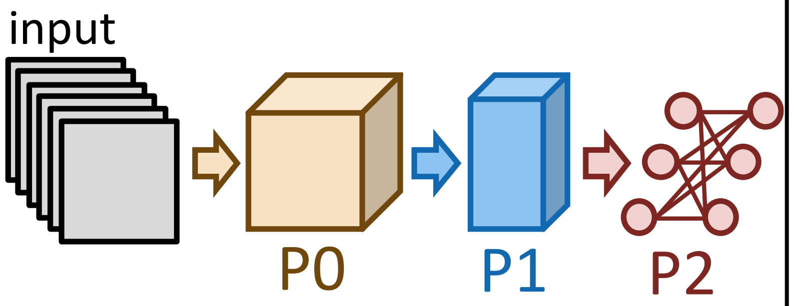
Pros:

a. Easy to realize

Cons:

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

Pipeline parallelism



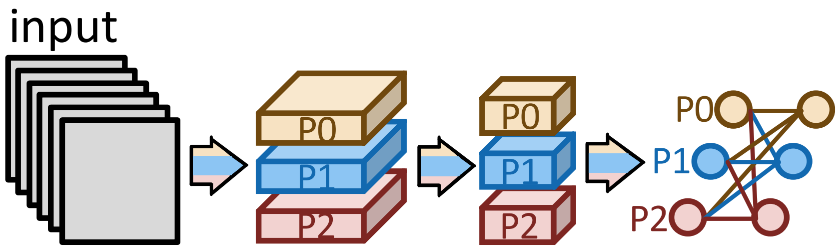
Pros:

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

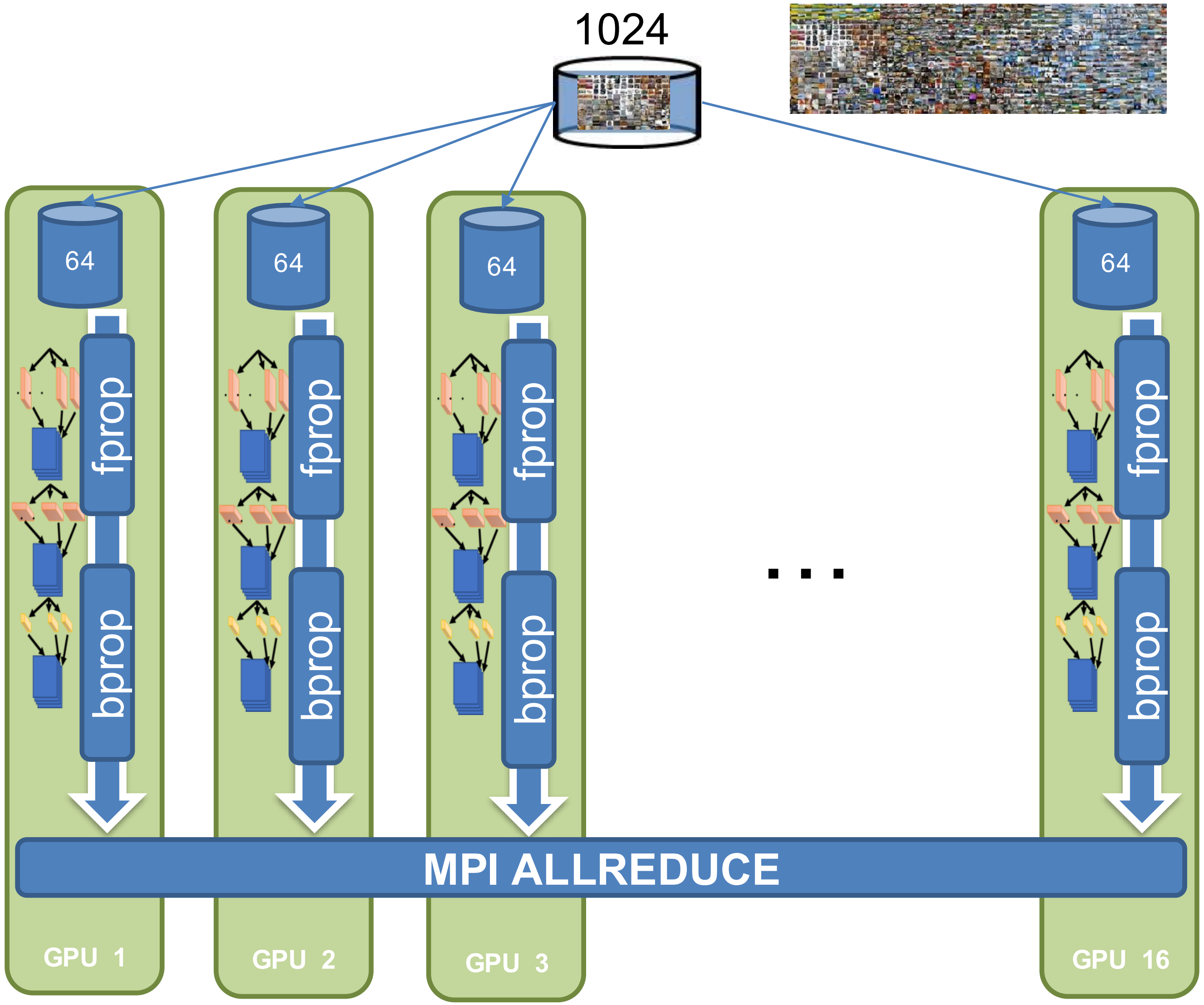
Cons:

b. Communication for each operator (or each layer)

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

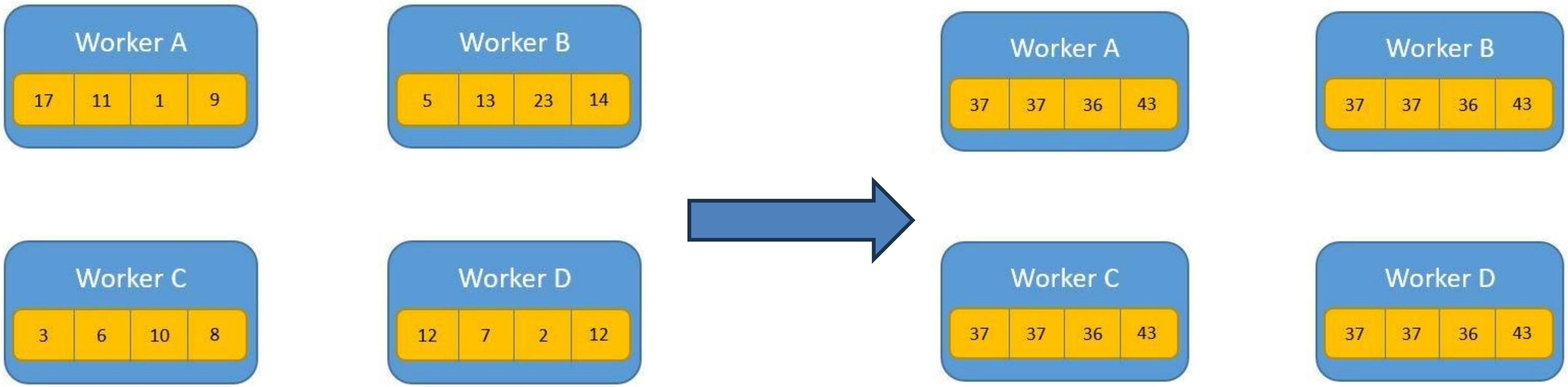
$$d_1 = \sum_{i=3B/4}^{B} \frac{\partial \mathcal{J}}{\partial w}$$

GPU 4

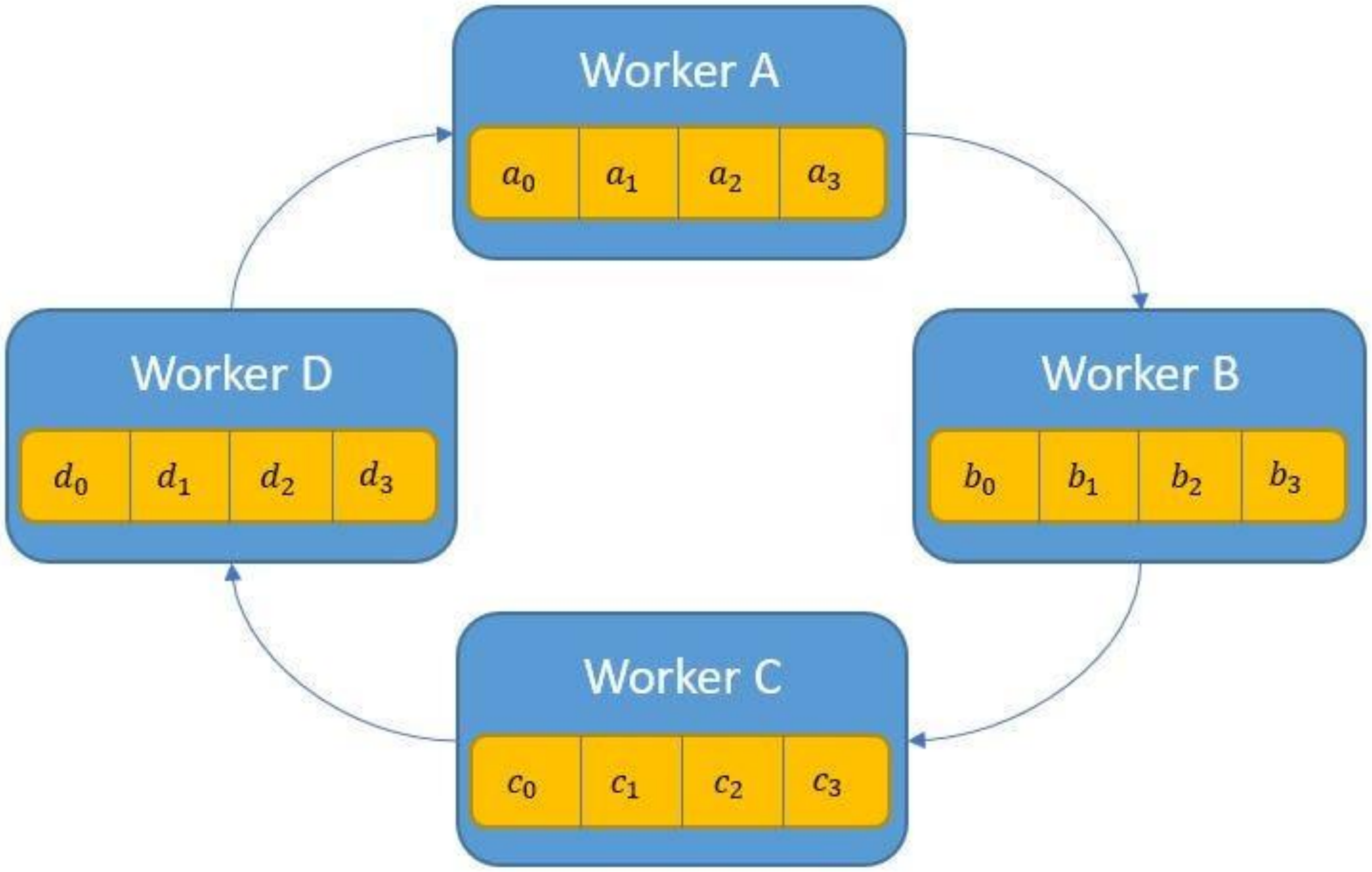
MPI ALLREDUCE

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

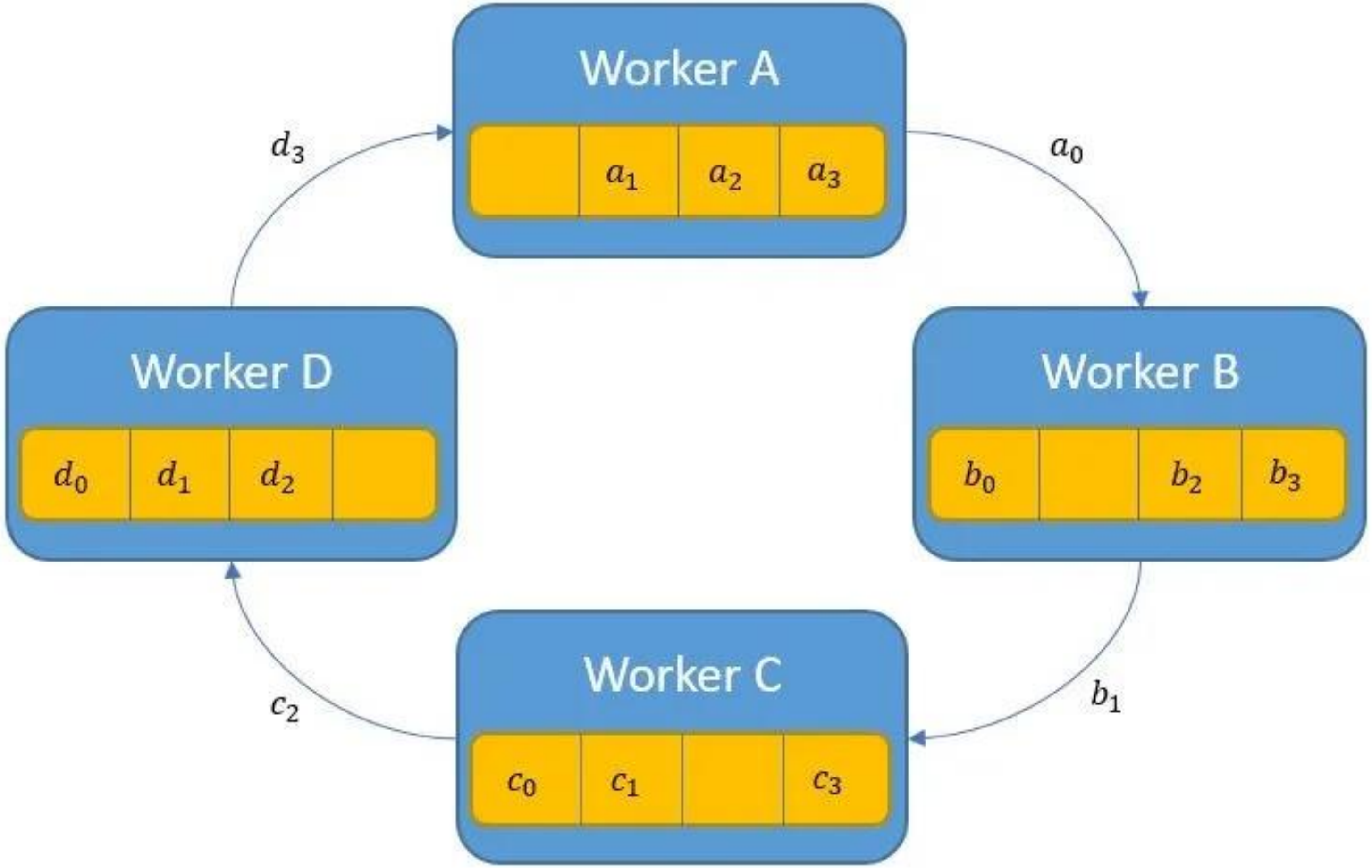
All Reduce – A High-Level Example



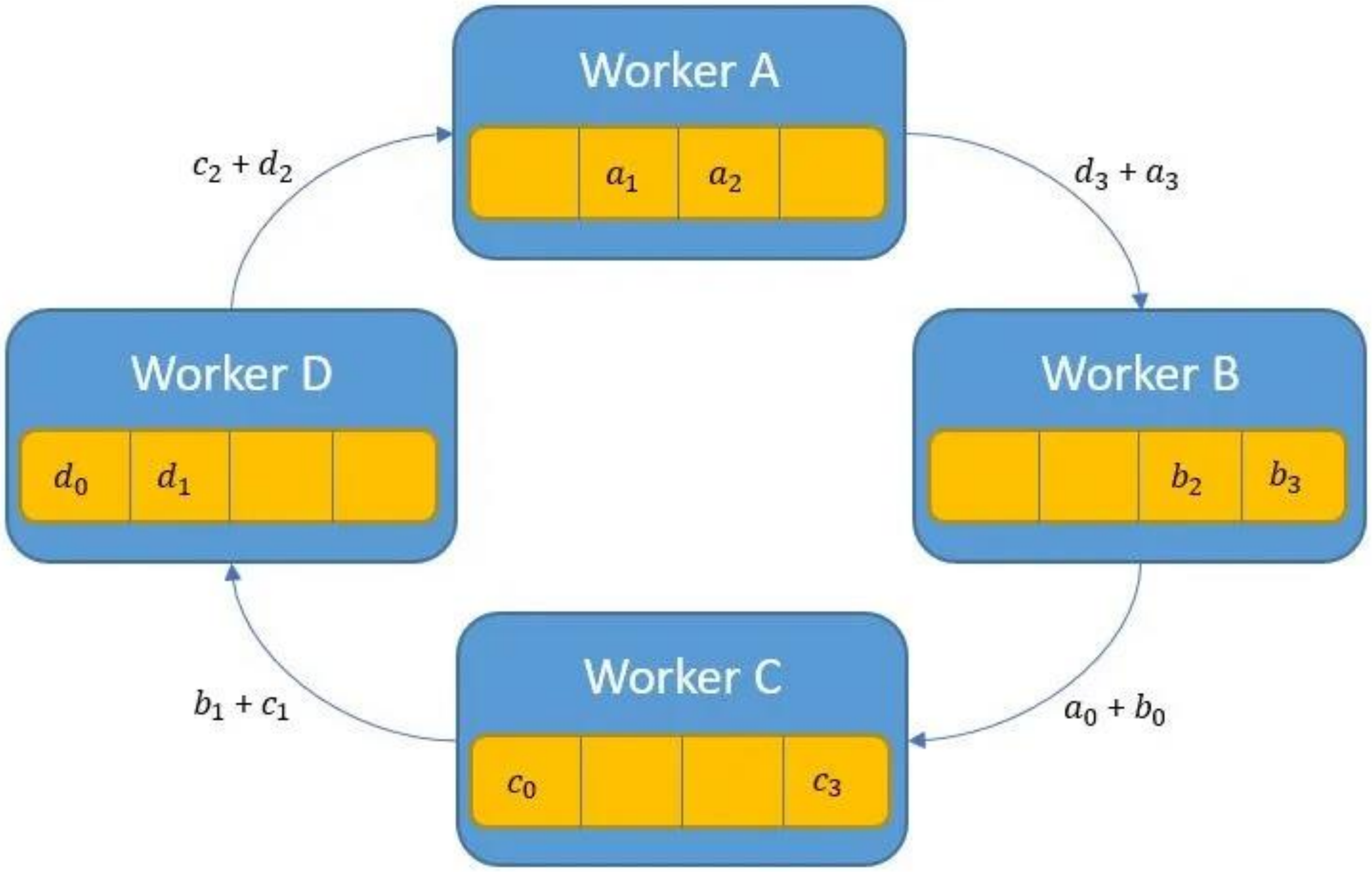
Ring All-Reduce



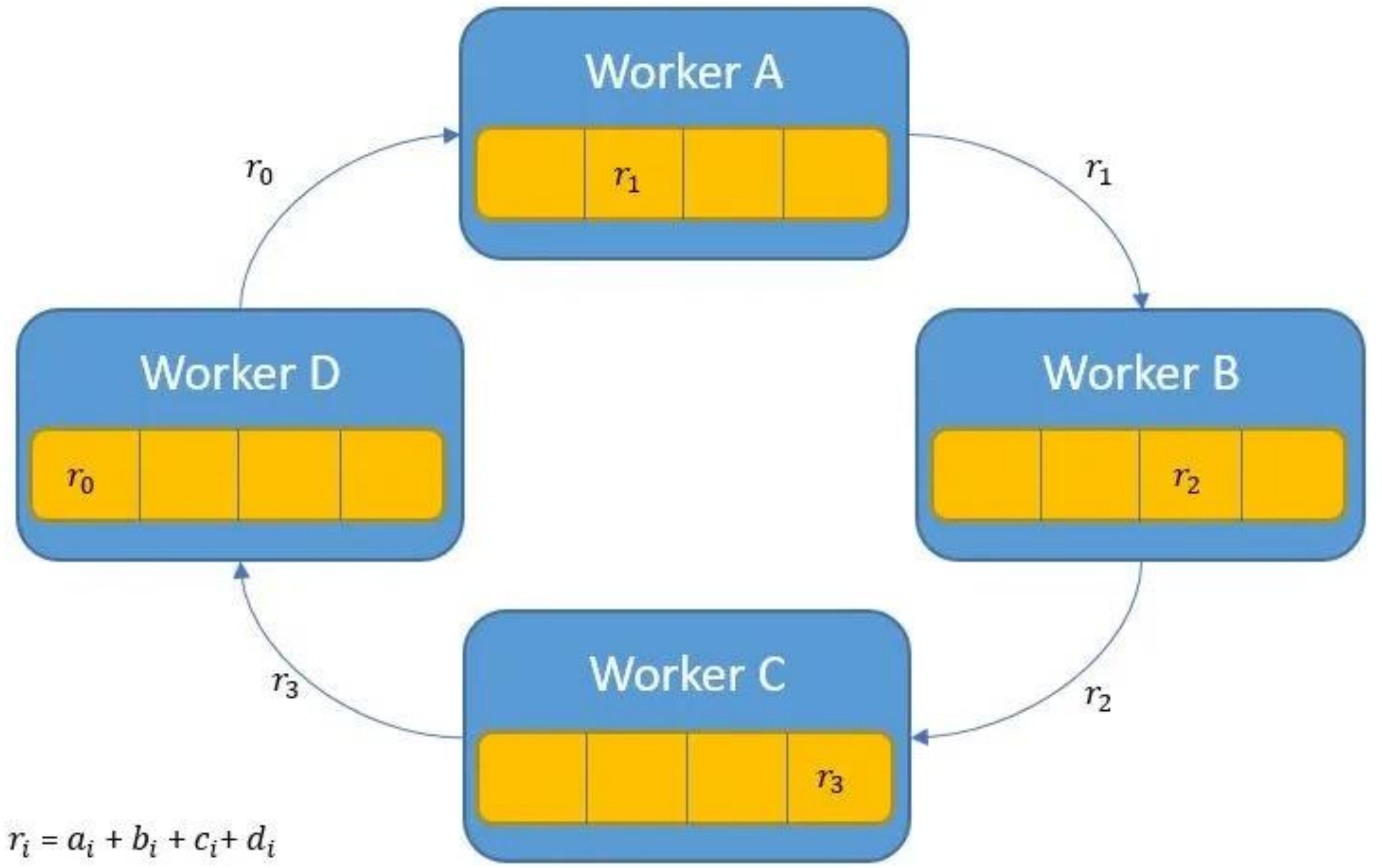
Ring All-Reduce – Step 1



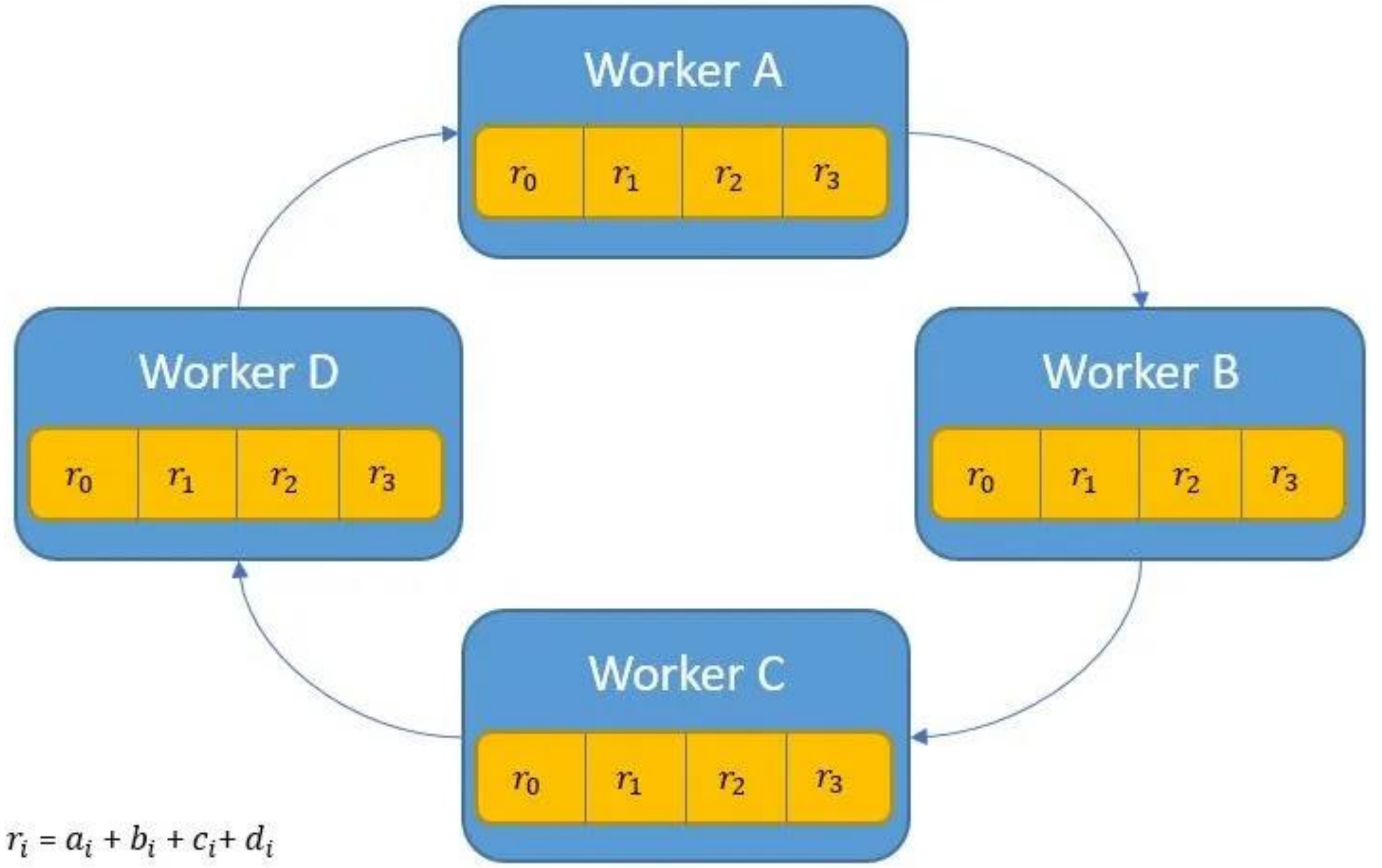
Ring All-Reduce – Step 2



Ring All-Reduce – Step 3

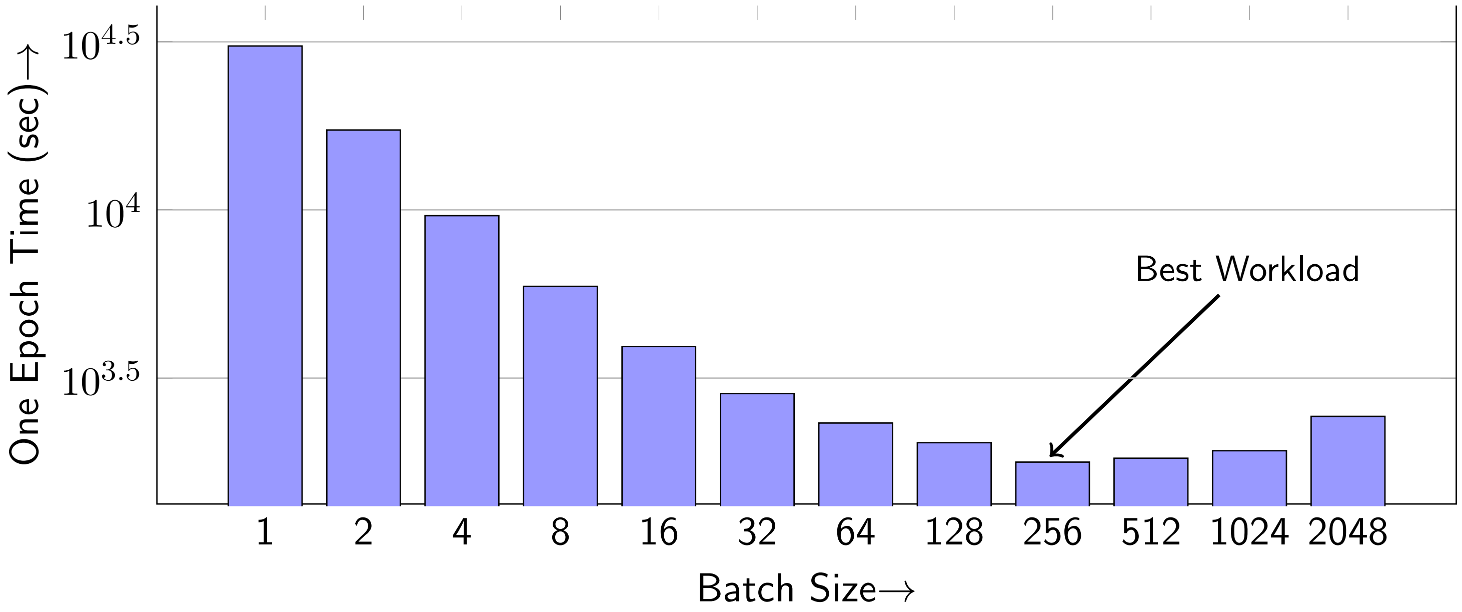


Ring All-Reduce – Step 4



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

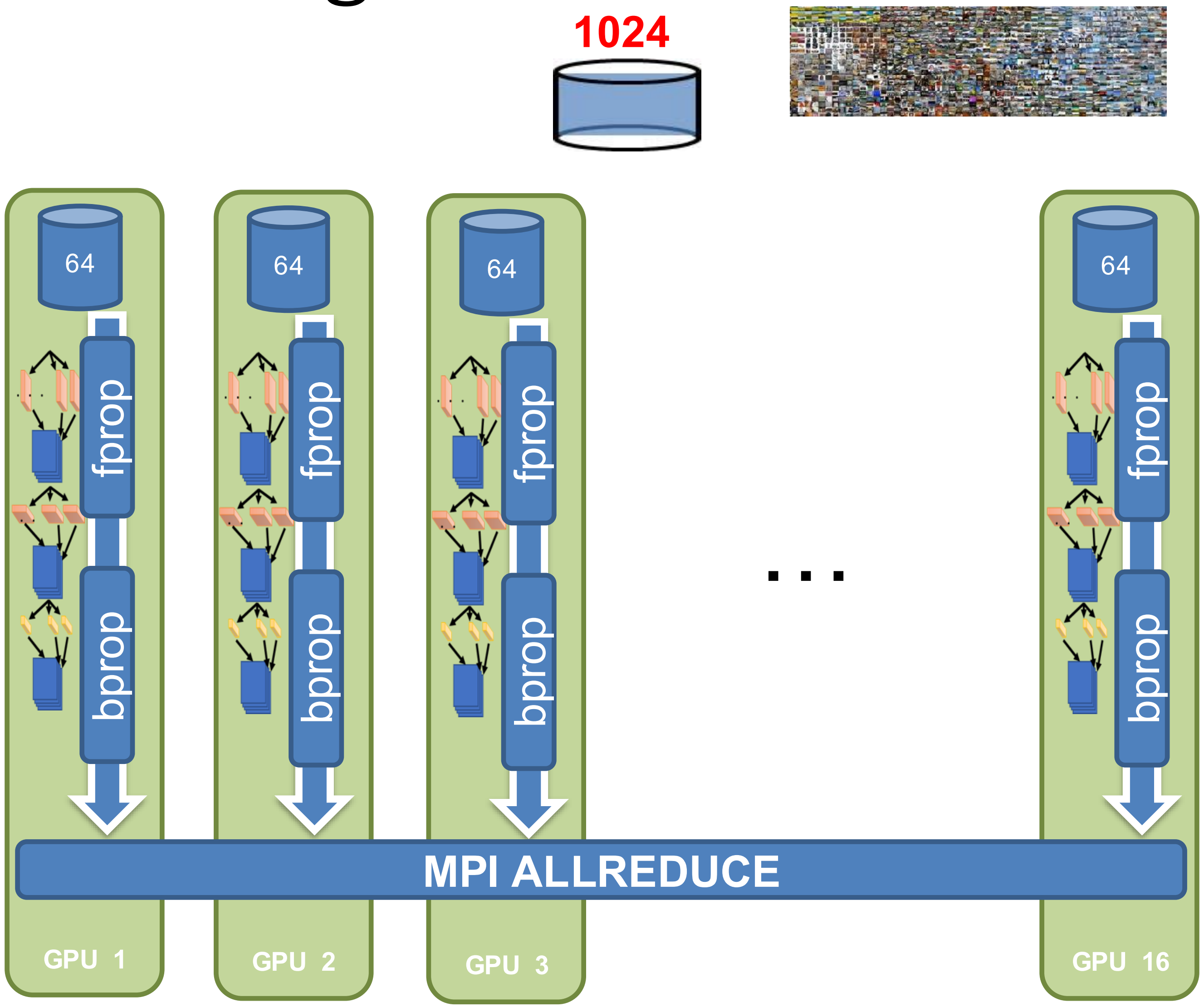


➢ Why?

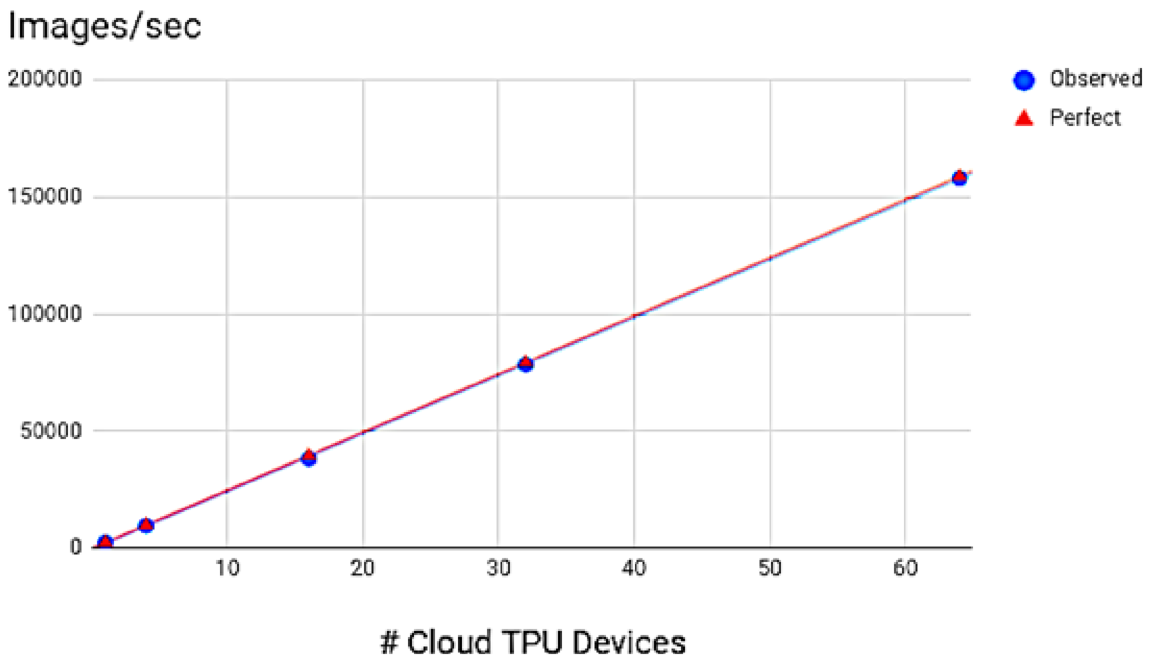
One epoch training time of AlexNet computed on an Intel KNL system

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 …



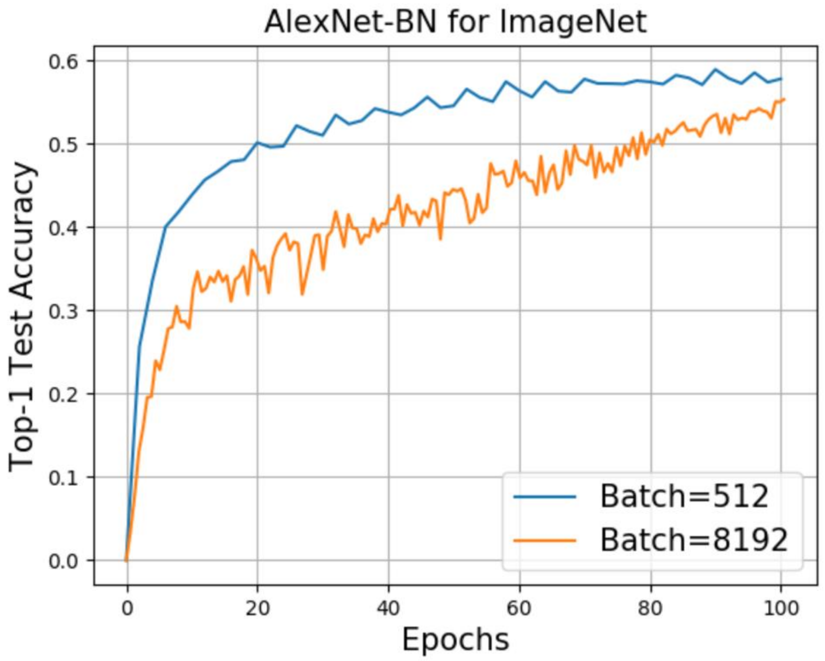
$$\left(\frac{\text{“Learning”}}{\text{Second}}\right) = \left(\frac{\text{“Learning”}}{\text{Record}}\right) \times \left(\frac{\text{Record}}{\text{Second}}\right)$$

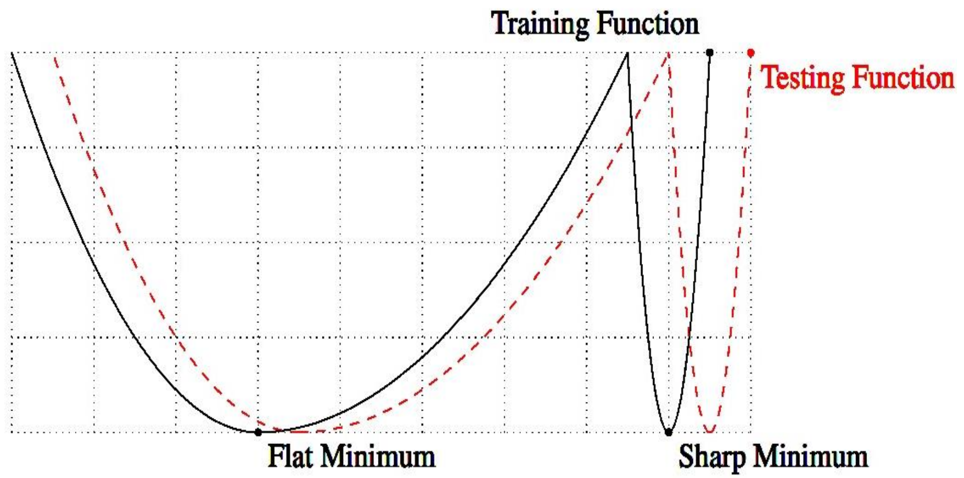
Convergence
Machine Learning
Property

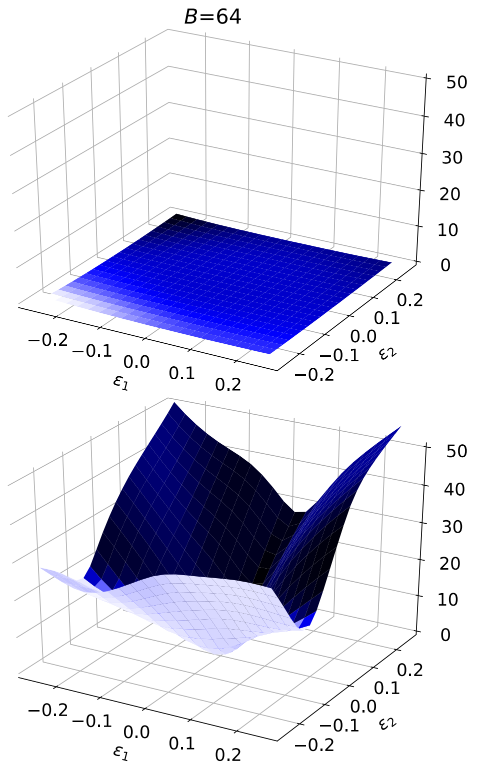
Throughput
System
Property

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"

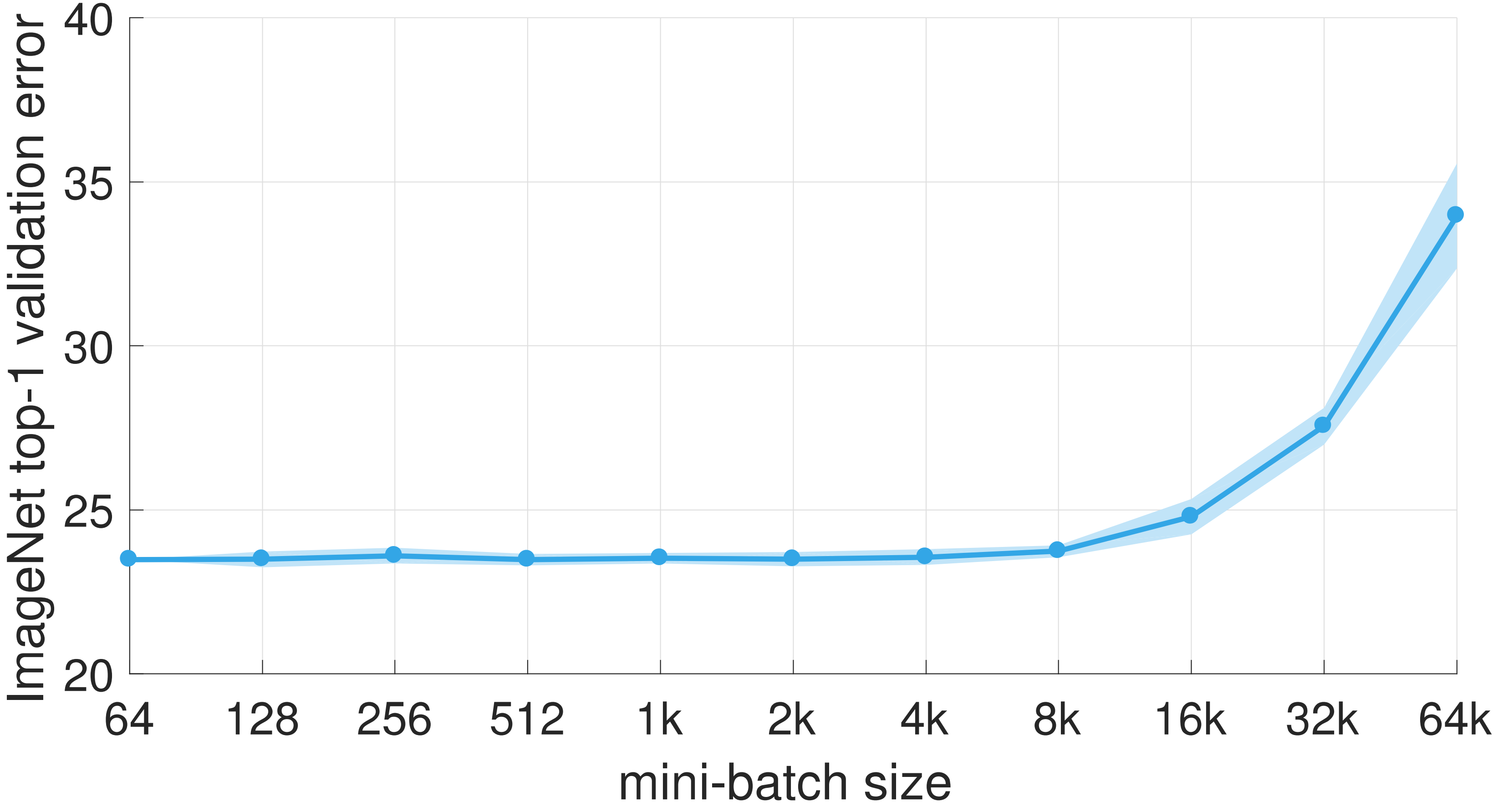






Keskar et al., On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima, ICLR'16.
Z. Yao, A. Gholami, Q. Lei, K. Keutzer, M. Mahoney. Hessian-based Analysis of Large Batch Training and Robustness to Adversaries, NeurIPS'18.
Ginsburg, Boris, Igor Gitman, and Yang You. "Large Batch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

Generalization Gap Problem



Larger batch sizes harm generalization performance.

Goyal, Priya, et al. "Accurate, large minibatch SGD: Training imagenet in 1 hour." arXiv preprint arXiv:1706.02677 (2017).

Data Parallelism Summary

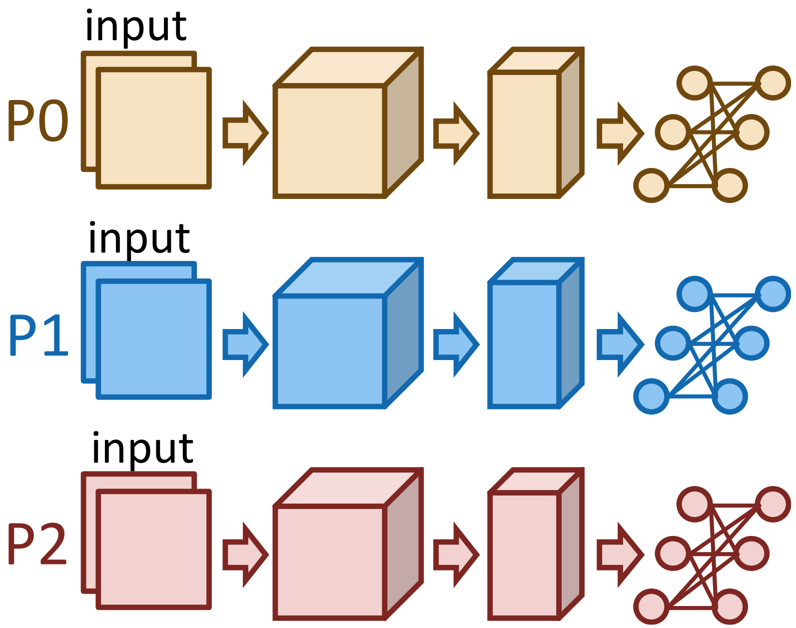
- An efficient parallel training method where the comm time is independent of processors with ring allreduce
- Very easy to implement. Only requires allreduce operation before updating parameters
- Very challenging to scale. Using large batch training is not an option as it hurts generalization performance.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

Pipeline Parallelism

Really a form of model parallelism

Parallel and distributed training

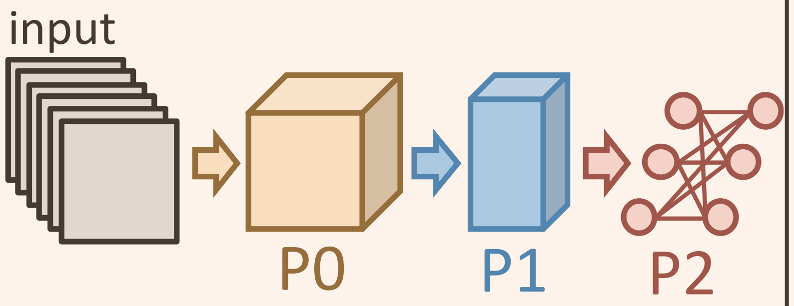
Data parallelism



Pros:
a. Easy to realize

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

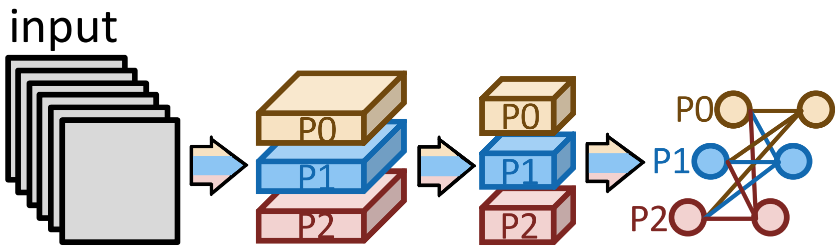
Pipeline parallelism



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

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

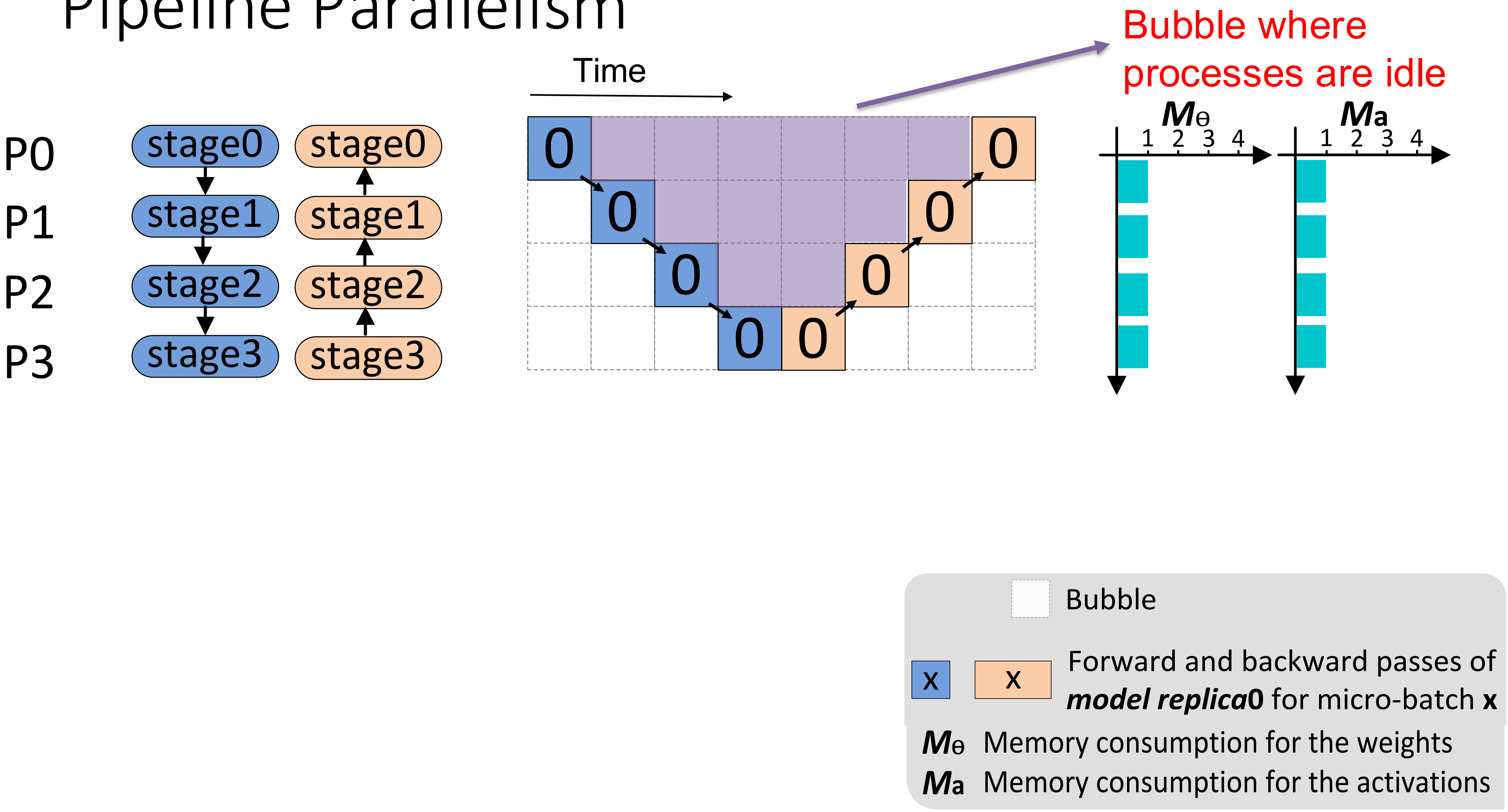
Model parallelism



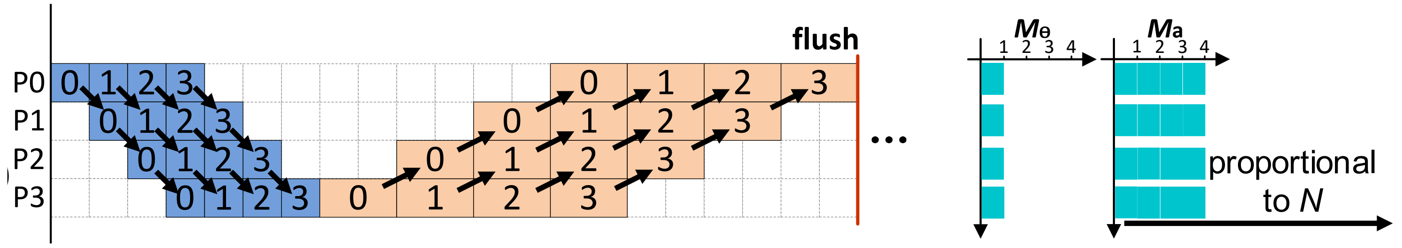
Pros:
a. Make large model training feasible

Cons:
b. Communication for each operator (or each layer)

Pipeline Parallelism



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

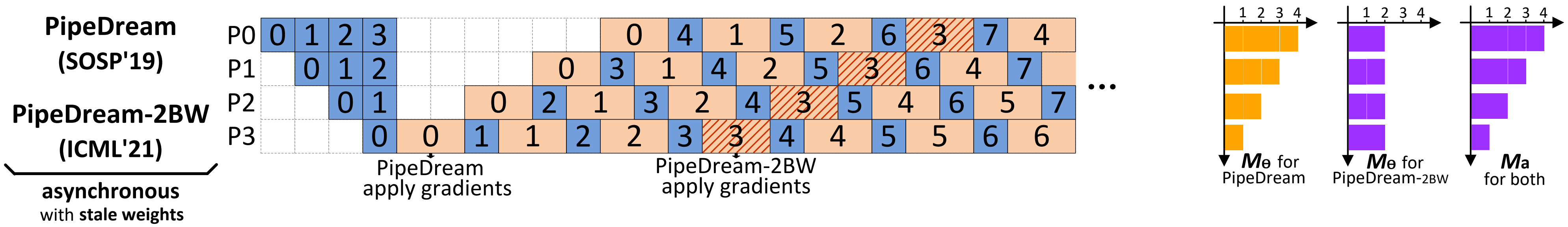
Bubble

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

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

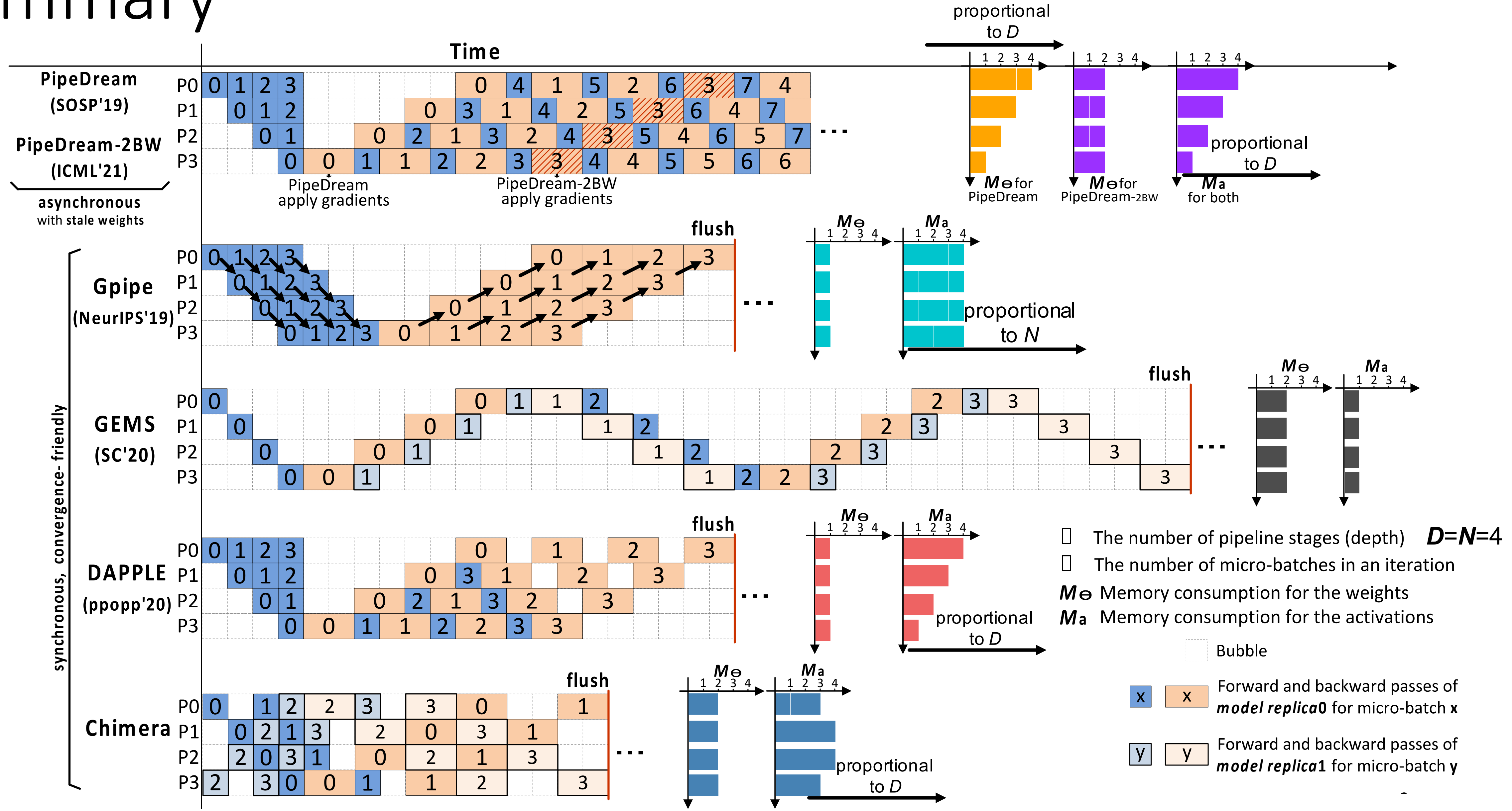
***M*a** Memory consumption for the activations

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.

Summary



Time
PipeDream
(SOSP'19)
PipeDream-2BW
(ICML'21)
asynchronous
with stale weights
PipeDream
apply gradients
PipeDream-2BW
apply gradients
proportional
to D
Mθ for
PipeDream
Mθ for
PipeDream-2BW
Ma
for both
synchronous, convergence- friendly
Gpipe
(NeurIPS'19)
flush
proportional
to N
GEMS
(SC'20)
DAPPLE
(ppopp'20)
Chimera
The number of pipeline stages (depth)
The number of micro-batches in an iteration
D=N=4
Mθ Memory consumption for the weights
Ma Memory consumption for the activations
Bubble
Forward and backward passes of
model replica0 for micro-batch x
Forward and backward passes of
model replica1 for micro-batch y

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

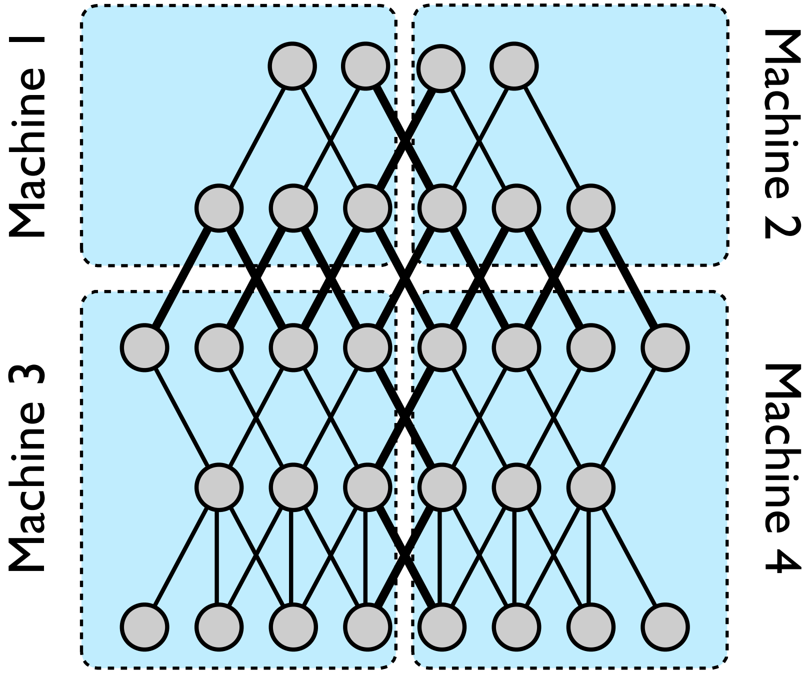
Model Parallelism

AKA Operator Parallelism

Model Parallelism

Divide the model across machines and replicate the data.

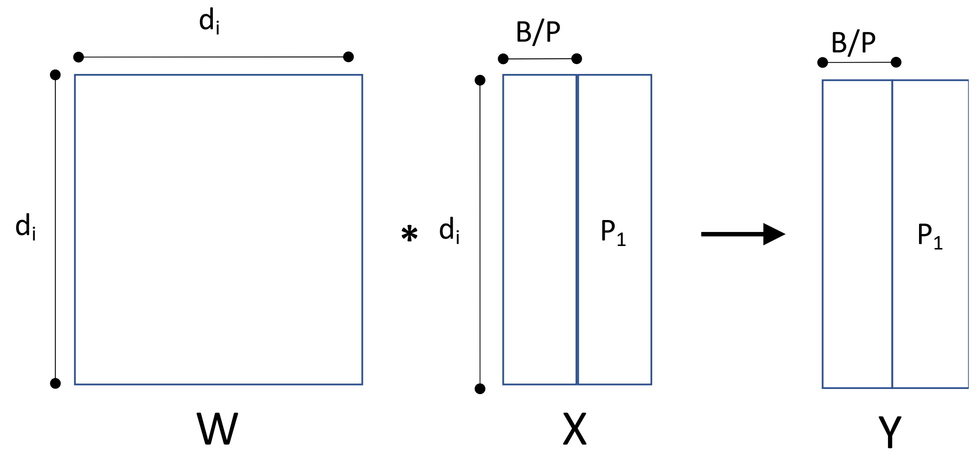
- ➢Supports large models and activations
- ➢Requires communication within single evaluation
- ➢How to best divide a model?
  - ➢Split individual layers
    - ➢which dimension?
      - ➢Weights or spatial → depends on operation
  - ➢Split across layers
    - ➢Only one set of layers active a time → poor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



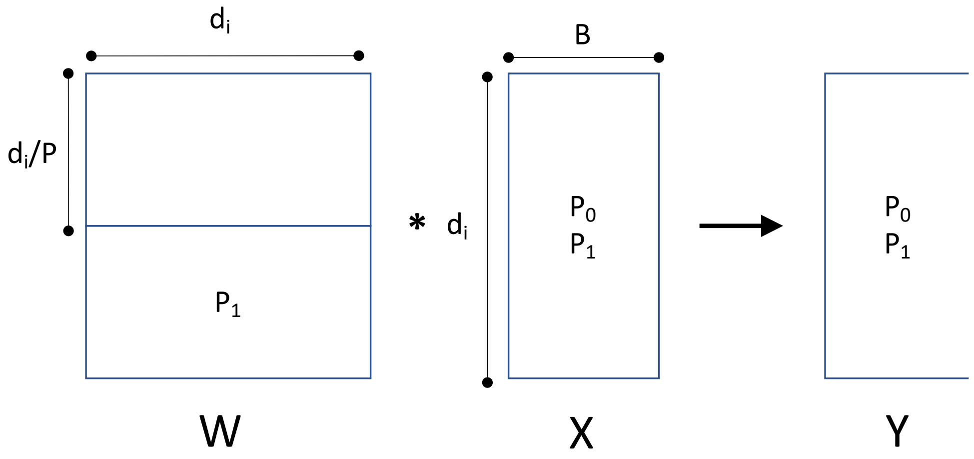
Model Parallelism: Weights

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

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

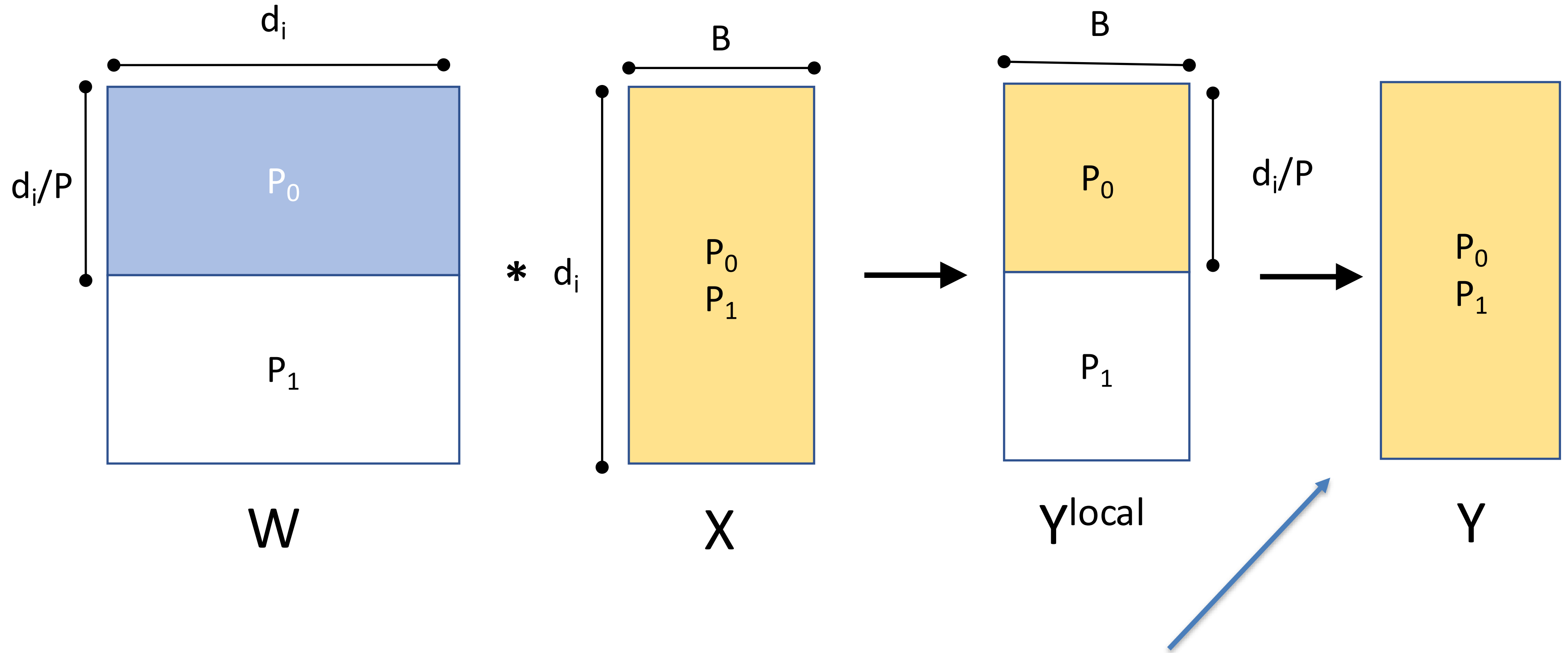


Model Parallelism: Partition weights across different Processes (W dimension)



Let’s discuss the communication details, step by step

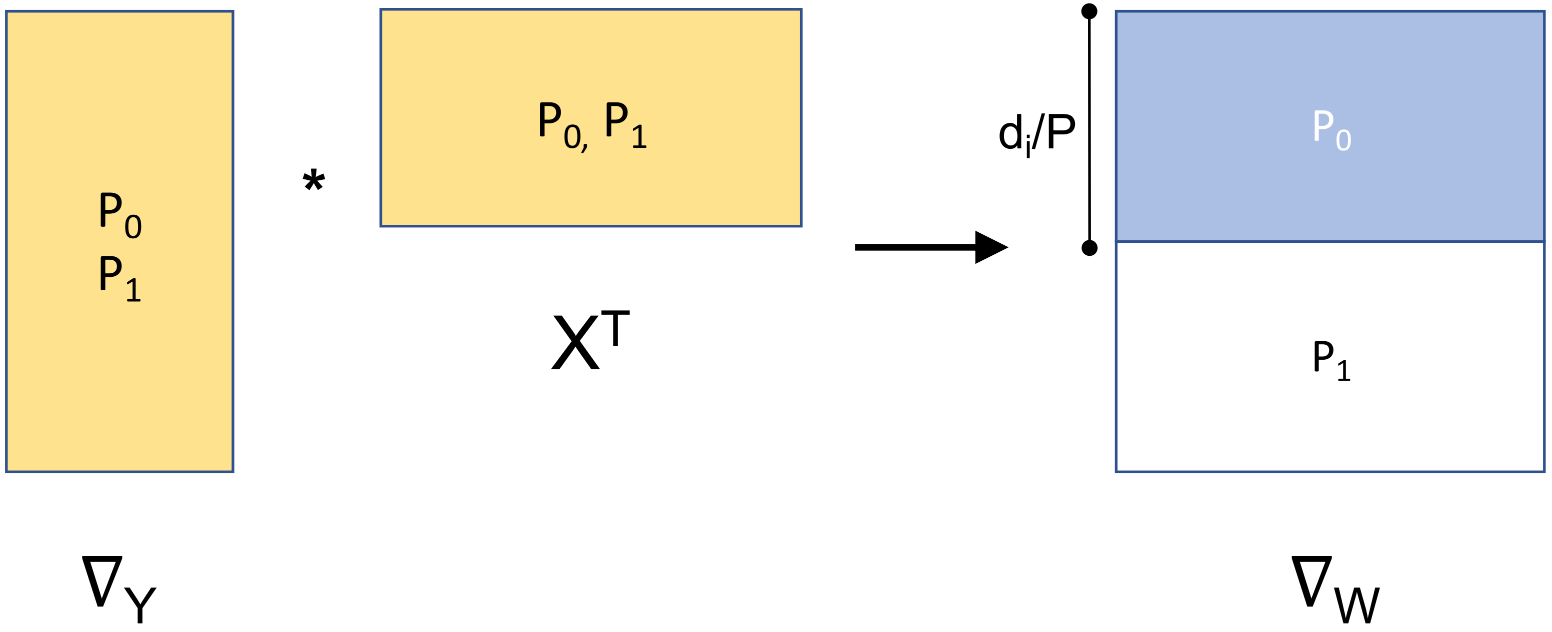
Model Parallelism: Forward Pass



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

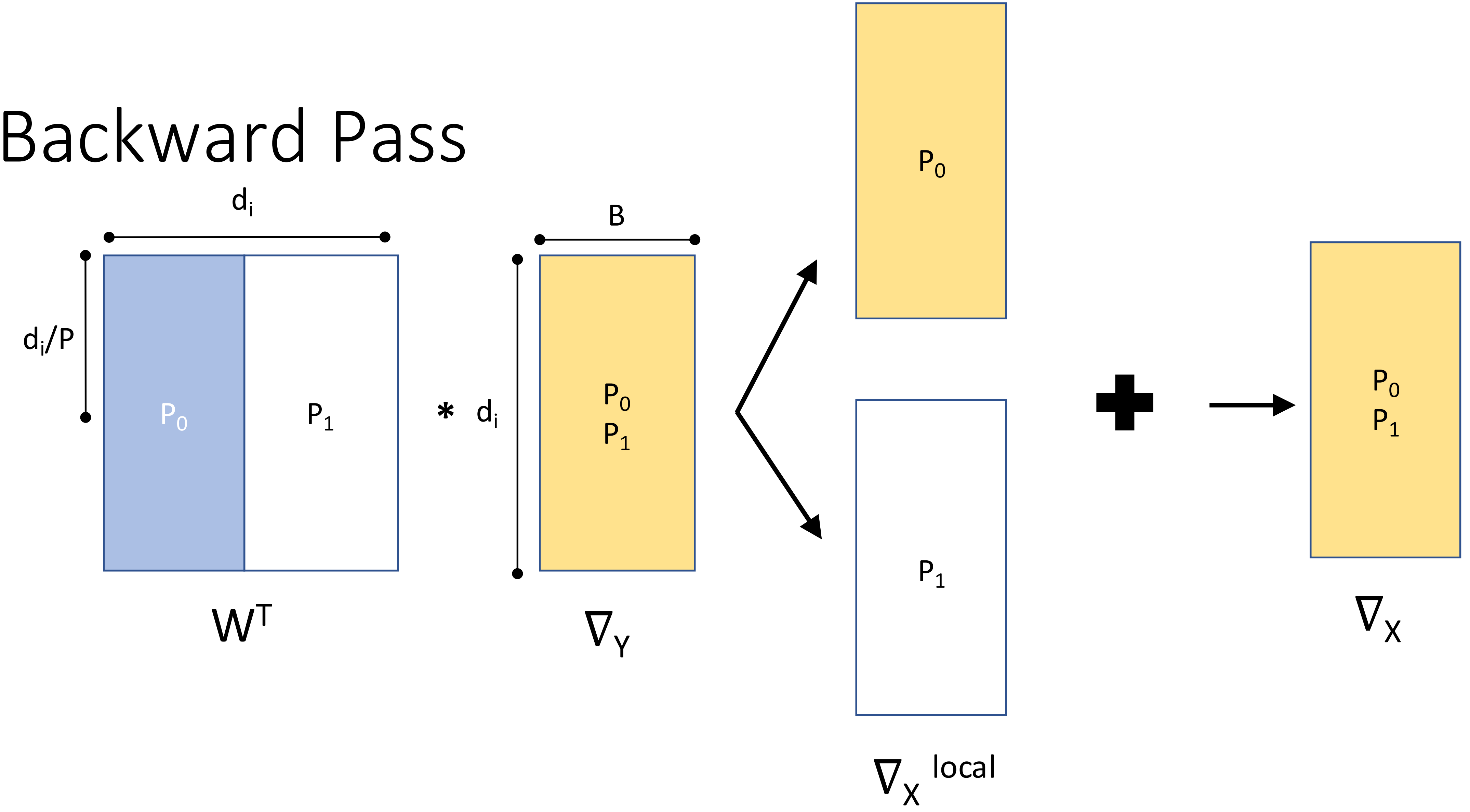
$$\sum_{i=1}^{L}\left(\beta(P-1)\frac{Bd_i}{P}\right)$$

Model Parallelism: Backward Pass



No communication needed as every processor only needs the gradient of its own parameters

Backward Pass



- Aggregating input gradient requires an allreduce operation

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

Communication 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

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

All Gather | 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 comm on d_i
- It is useful for layers with very large weights d_i >> 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

- More optimal comm time 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 parallel
- Processes are never idle