UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

CS 498: Machine Learning System
Spring 2025

Minjia Zhang

The Grainger College of Engineering

Today

- **Distributed Training**
  - **Problem**
  - **Challenge**
  - **History**
  - **Parameter Server based DP**
  - **Communication terminologies**

- **First paper summary due date: EOD Jan 31 (Friday)**

ML/DL Training Problem Definition Recap

- Given model f, data set $\{xi, y_i\}_{i=1}^{N}$
- Minimize the loss between predicted labels and true labels:

  Min $\frac{1}{N} \sum_{i=1}^{N} loss(f(x_i, y_i))$

- Common loss function
  - Cross-entropy, MSE (mean squared error)

- Common way to solve the minimization problem
  - Adaptive learning rates optimizers (e.g., Adam)
  - Stochastic gradient descent (SGD)

Gradient Descent

- Model f_w is parameterized by weight w
- η > 0 is the learning rate

Backward pass

Forward pass

For t = 1 to T

$\Delta w = \eta \times \frac{1}{N}\sum_{i=1}^{N} \boxed{\nabla} \left(\boxed{loss\left(f_w(x_i, y_i)\right)}\right)$ // compute derivative and update

w -= Δw // apply update

End

Adaptive Learning Rates (Adam)

- Model f_w is parameterized by weight w
- η > 0 is the learning rate

For t = 1 to T

$$\Delta w = \eta \times \frac{1}{N}\sum_{i=1}^{N} \nabla \left(loss\left(f_w(x_i, y_i)\right)\right)$$

$w \mathrel{-}= \boxed{\Delta w}$ // apply update

End

$$\nu_t = \beta_1 * \nu_{t-1} - (1-\beta_1) * g_t$$
$$s_t = \beta_2 * s_{t-1} - (1-\beta_2) * g_t^2$$
$$\Delta\omega_t = -\eta \frac{\nu_t}{\sqrt{s_t + \epsilon}} * g_t$$

g_t : *Gradient at time t along* ω^j

ν_t : *Exponential Average of gradients along* ω_j

s_t : *Exponential Average of squares of gradients along* ω_j

β_1, β_2 : *Hyperparameters*

Adam: A Method for Stochastic Optimization, 2014

Accelerating Gradient Descent

- Model f_w is parameterized by weight w
- η > 0 is the learning rate

Can we accelerate it?

For t = 1 to T

$$\boxed{\Delta w = \eta \times \frac{1}{N} \sum_{i=1}^{N} \nabla \left(loss\left(f_w(x_i, y_i)\right)\right)}$$ // compute derivative and update

w -= Δw // apply update

End

Accelerating Gradient Descent

- Model f_w is parameterized by weight w
- η > 0 is the learning rate

Increase the batch size increases AI but eventually bounded by single GPU compute

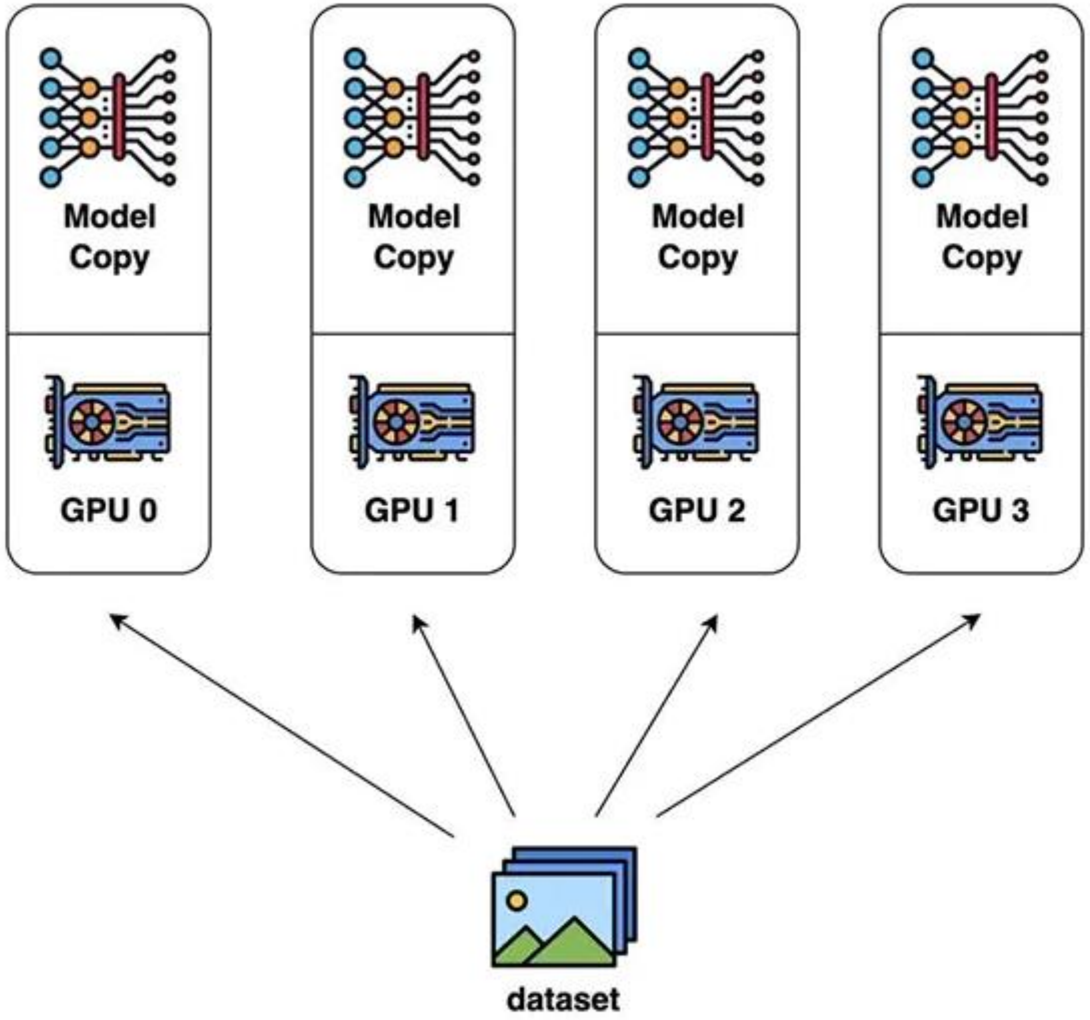
For t = 1 to T

$\boxed{\Delta w = \eta \times \frac{1}{N} \sum_{i=1}^{N} \nabla \left(loss(f_w(x_i, y_i)) \right)}$ // compute derivative and update

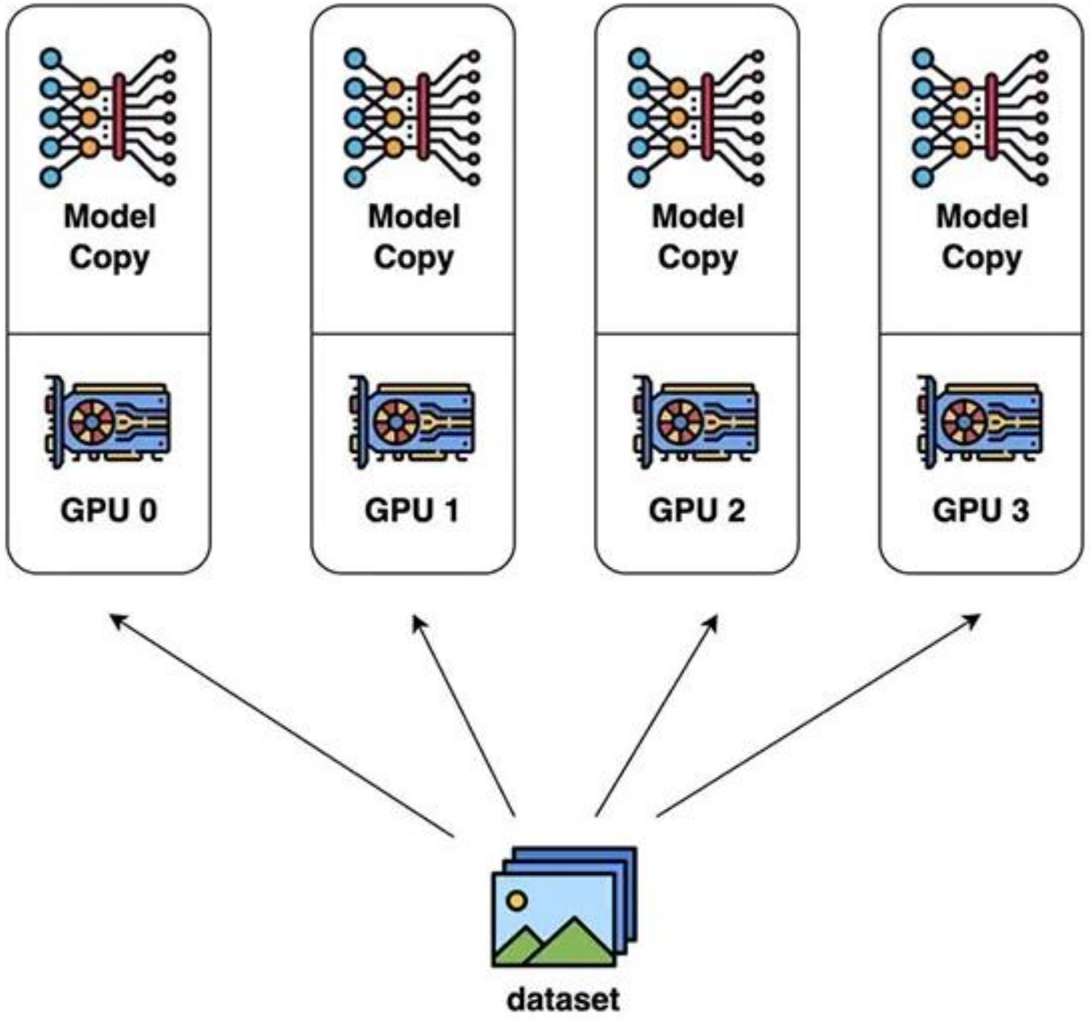
w -= Δw // apply update

End

Data Parallelism (DP)

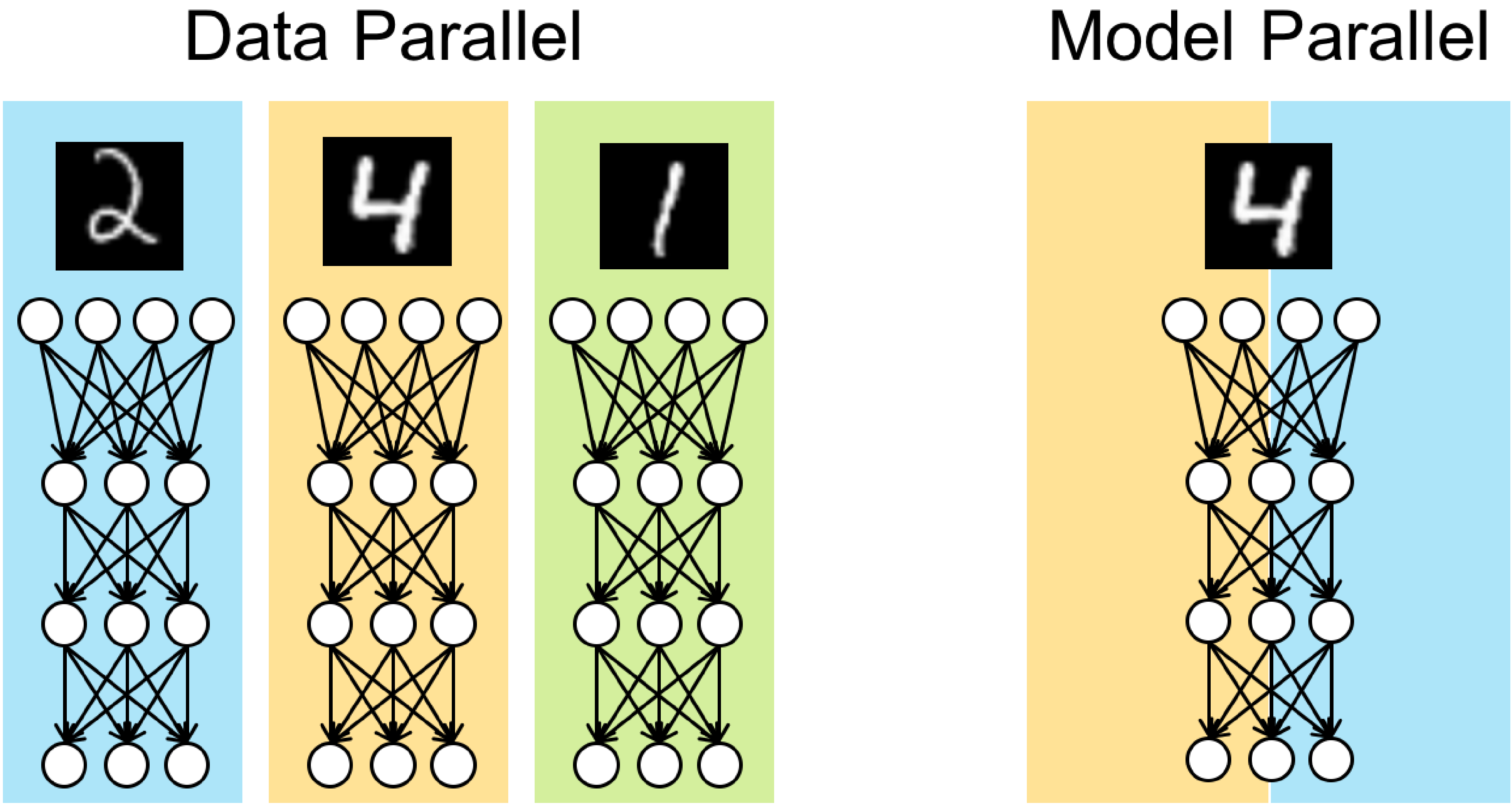


Data Parallelism (DP)



What happens when model does not fit on a single GPU?

Data Parallelism (DP) vs. Model Parallelism



Distributed DL History

2012

Reflections of DL parallelization in early DL papers

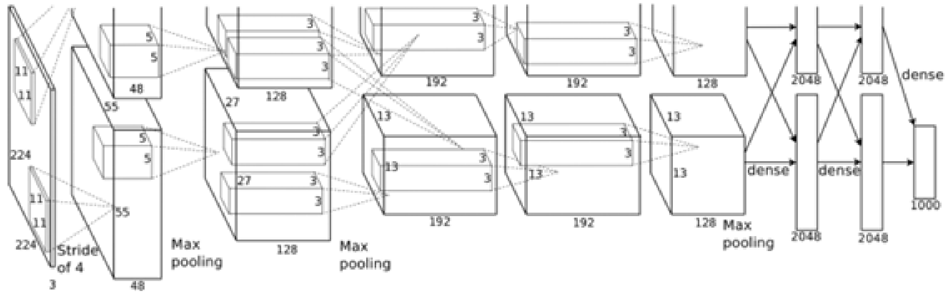


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

Figure from AlexNet [Krizhevsky et al., NeurIPS 2012], [Krizhevsky et al., preprint, 2014]

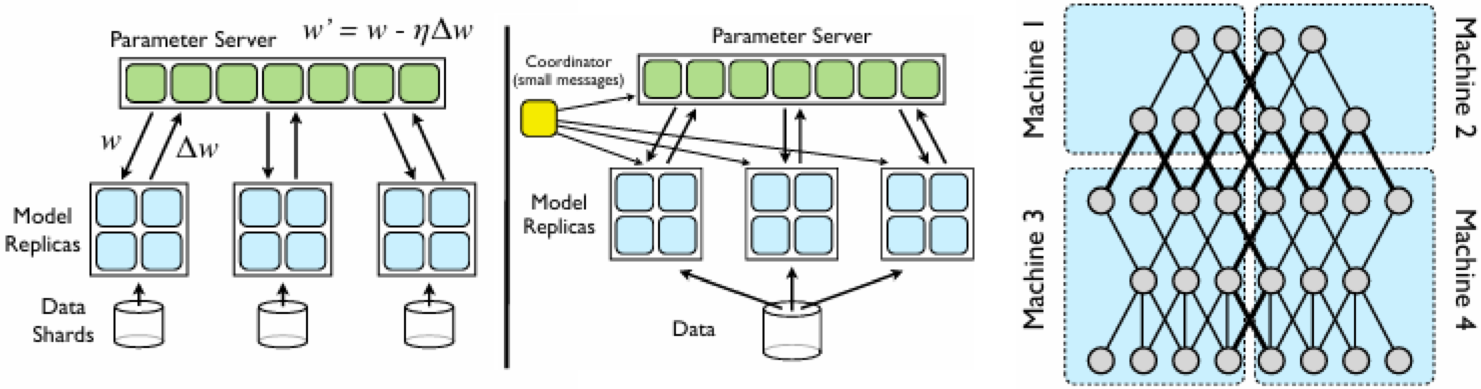


Figure from DistBelief [Dean et al., NeurIPS 2012]

Data Parallelism with Parameter Server

2012

2016

Focus: Data parallelism with **Parameter Server**

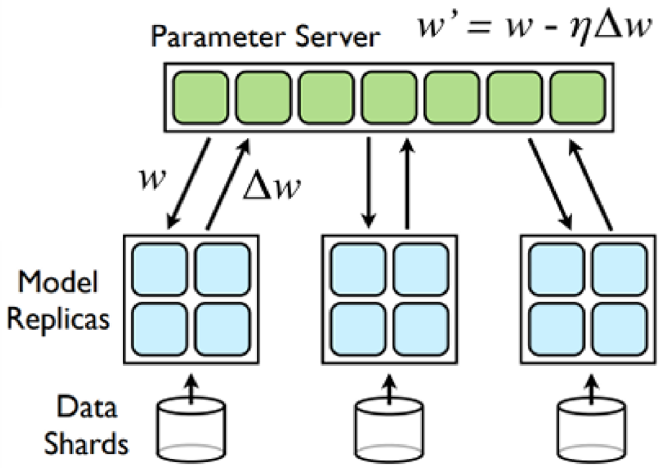


Figure from DistBelief
[Dean et al., NeurIPS 2012]

Various implementations of parameter servers

- DistBelief [Dean et al., NeurIPS 2012]
- Parameter server [Li et al., NeurIPS 2012], [Li et al., OSDI 2014]
- Bosen [Wei et al., SoCC 2015]
- GeePS [Cui et al., Eurosys 2016], Poseidon [Zhang et al., ATC 2017]

Data Parallelism with Parameter Server (Async)

2012

2016

Focus: Data parallelism with **Parameter Server**

Asynchrony: update every N iters instead of 1

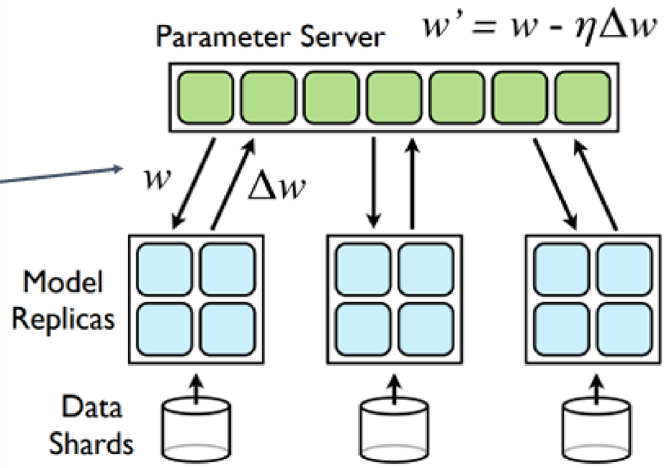


Figure from DistBelief [Dean et al., NeurIPS 2012]

Various implementations of parameter servers

- DistBelief [Dean et al., NeurIPS 2012]
- Parameter server [Li et al., NeurIPS 2012], [Li et al., OSDI 2014]
- Bosen [Wei et al., SoCC 2015]
- GeePS [Cui et al., Eurosys 2016], Poseidon [Zhang et al., ATC 2017]

Data Parallelism with AllReduce

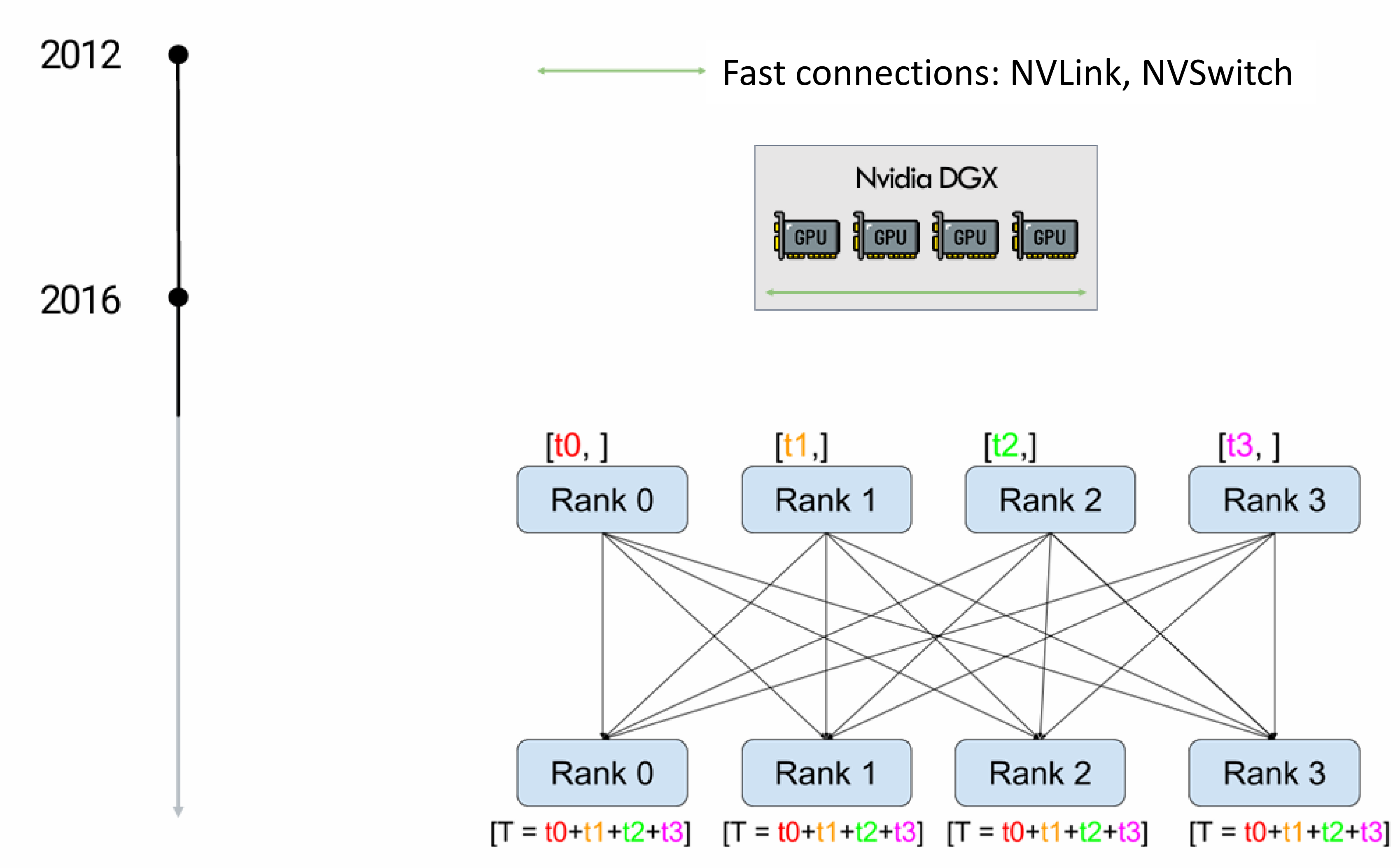


Figure from PyTorch Tutorials

Distributed Data Parallel

2012

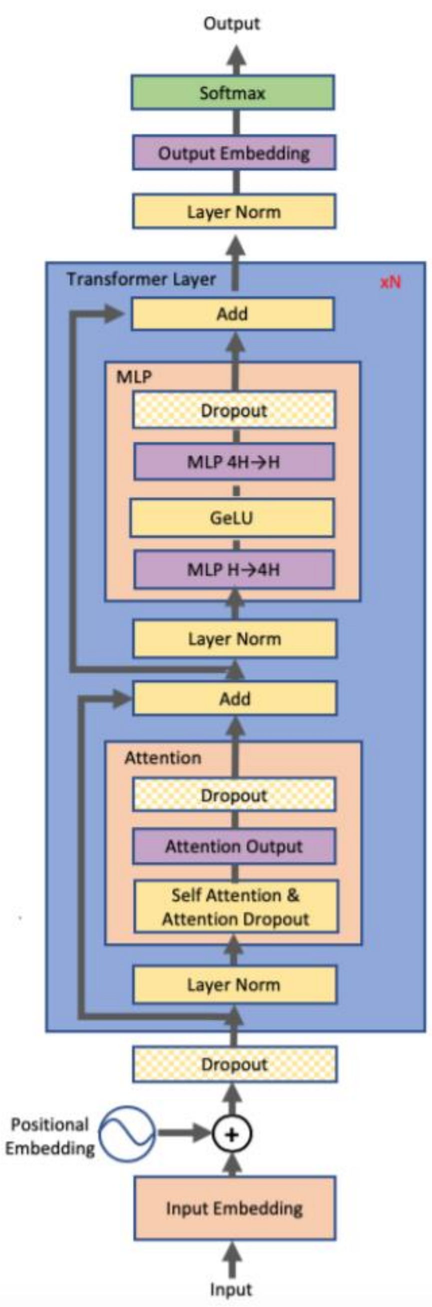
2016

```
import torch.nn.parallel as dist
from torch.nn.parallel import DistributedDataParallel as DDP

dist.init_process_group("nccl", rank=rank, world_size=world_size)
ddp_model = DDP(Model(), device_ids=[rank])

for batch in data_loader:
    loss = train_step(ddp_model, batch)
```

Sergeev et al., "Horovod: fast and easy distributed deep learning in TensorFlow". *Preprint 2018.*
Li et al., "PyTorch Distributed: Experiences on Accelerating Data Parallel Training". VLDB 2020.

Large Model Training Challenges

	Bert-Large	GPT-2	Turing 17.2 NLG	GPT-3
Parameters	0.32B	1.5B	17.2B	175B
Layers	24	48	78	96
Hidden Dimension	1024	1600	4256	12288
Relative Computation	1x	4.7x	54x	547x
Memory Footprint	5.12GB	24GB	275GB	2800GB

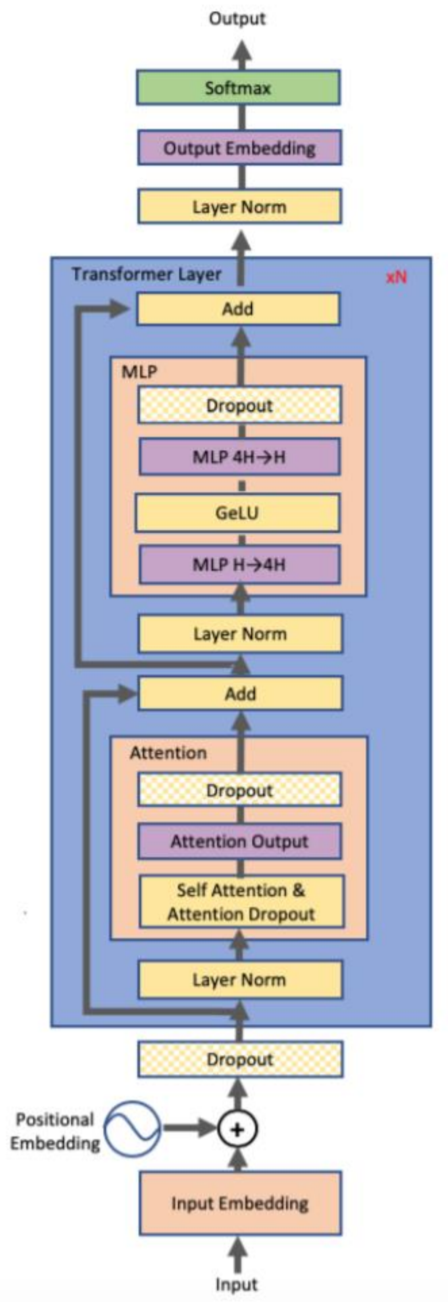


Large Model Training Challenges

	Bert-Large	GPT-2	Turing 17.2 NLG	GPT-3
Parameters	0.32B	1.5B	17.2B	175B
Layers	24	48	78	96
Hidden Dimension	1024	1600	4256	12288
Relative Computation	1x	4.7x	**54x**	**547x**
Memory Footprint	5.12GB	24GB	**275GB**	**2800GB**

Out of Memory

NVIDIA V100 GPU memory capacity: 16G/32G
NVIDIA A100 GPU memory capacity: 40G/80G



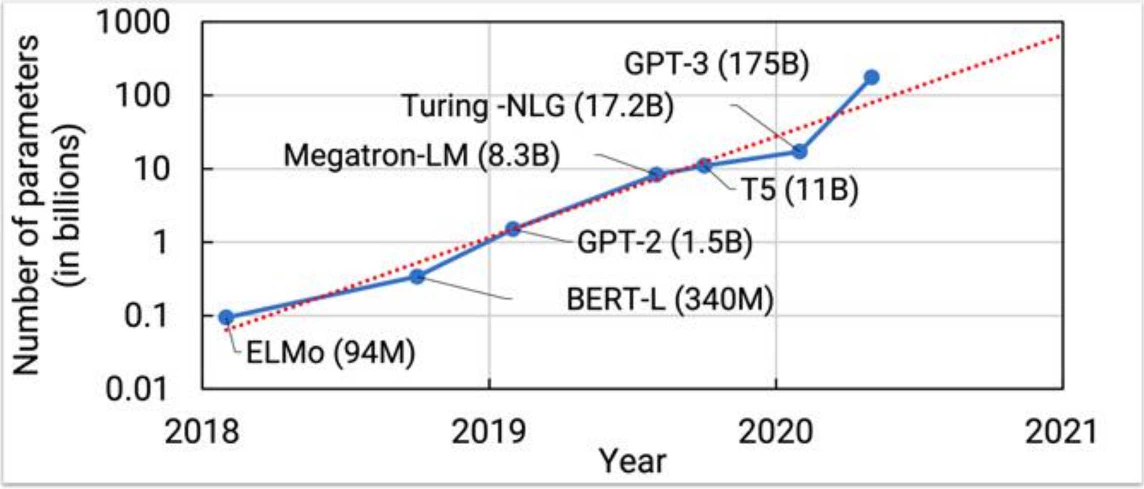
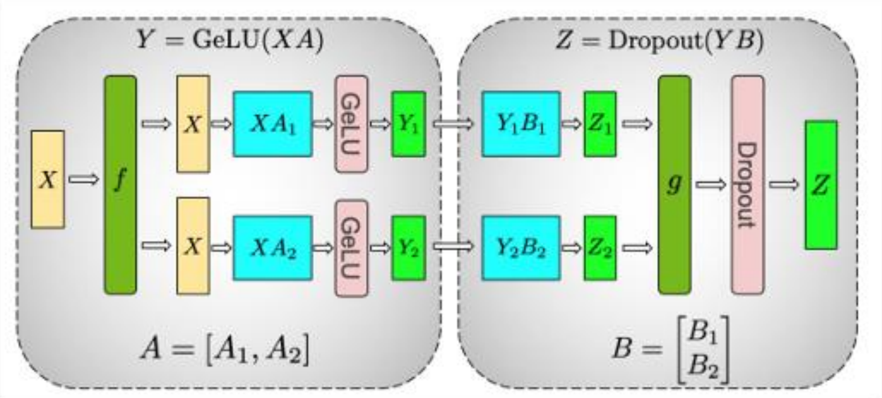
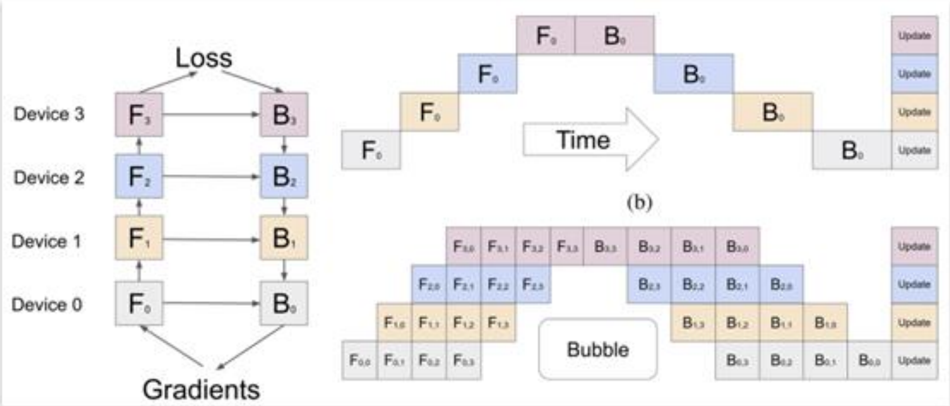


Figure from Nvidia

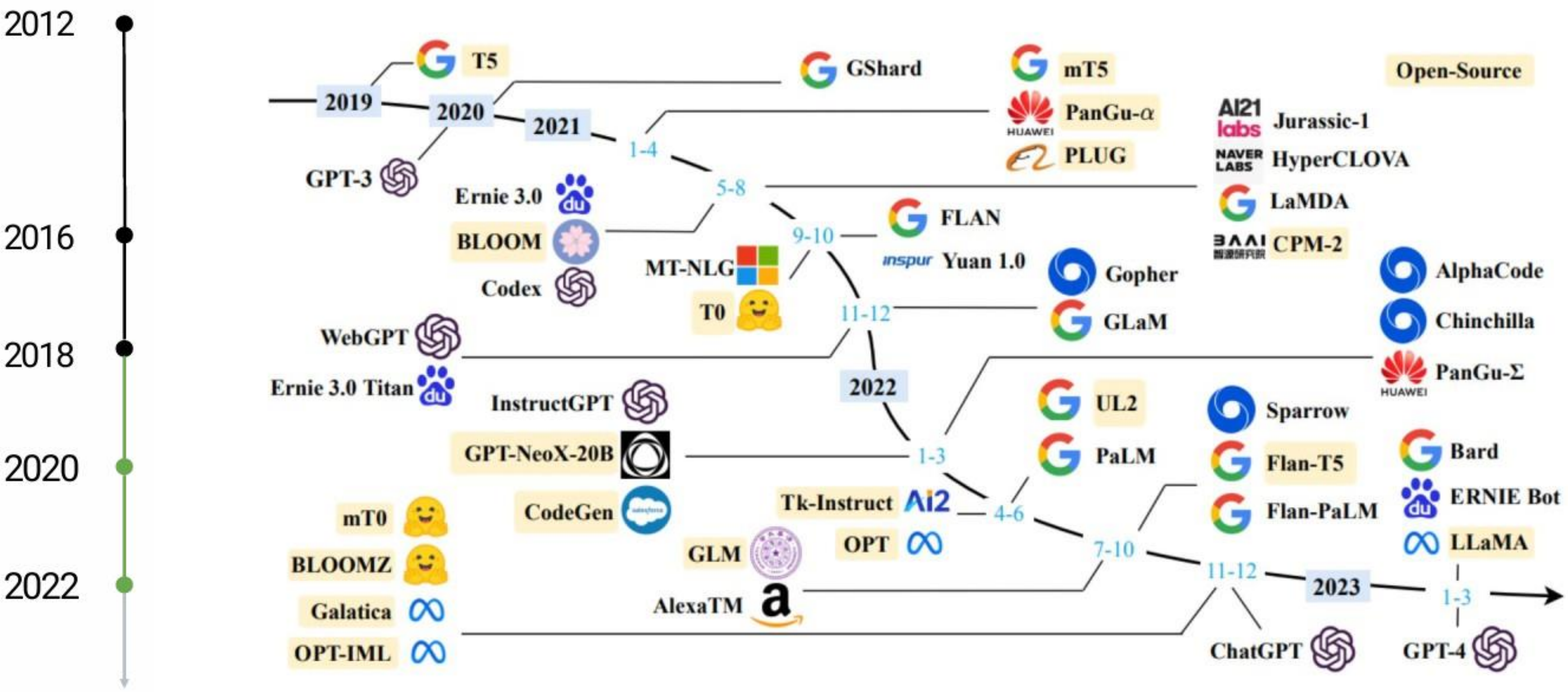


Matmul partitioning
[Shoeybi et al., ICML 2020]



Pipeline parallelism
[Huang et al., NeurIPS 2019]

Big Model Era



Today

- **Distributed Training**
  - Problem
  - Challenge
  - History
  - **Parameter Server based DP**

Accelerating Gradient Descent

- Model f_w is parameterized by weight w
- η > 0 is the learning rate

Increase the batch size increases AI but eventually bounded by single GPU compute

For t = 1 to T

$$\boxed{\Delta w = \eta \times \frac{1}{N} \sum_{i=1}^{N} \nabla \left(loss\left(f_w(x_i, y_i)\right)\right)}$$ // compute derivative and update

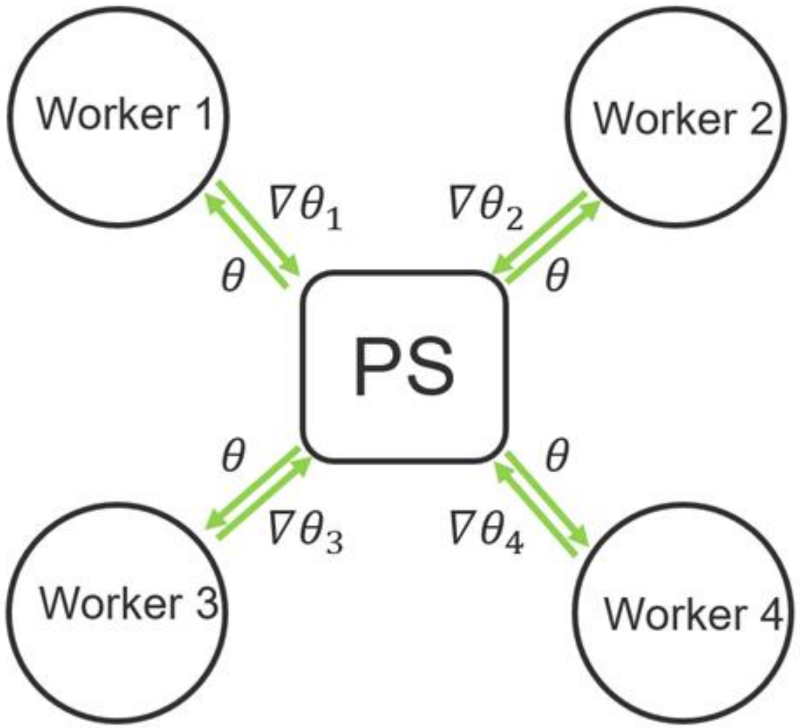
w -= Δw // apply update

End

Two Solutions

- Parameter Server
- AllReduce
- Key assumption:
  - The model can fit into a (GPU) worker memory so we can create many model replica

Parameter Server Naturally Emerges

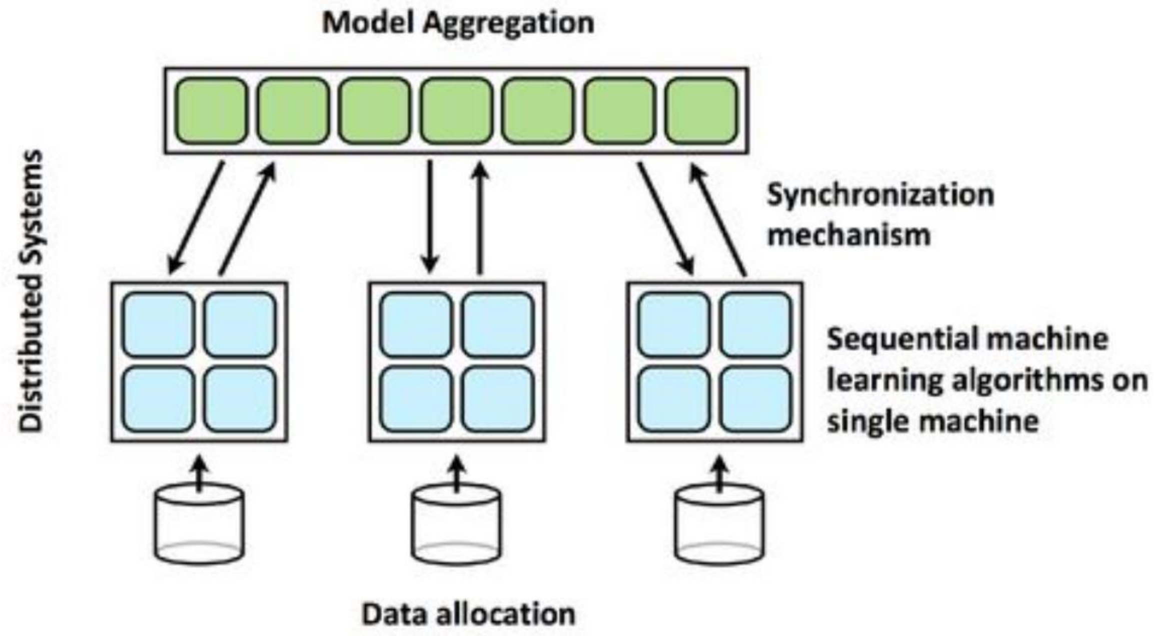


Assumptions (similar as MapReduce):
- Very heavy communication per iteration
- Compute : communication = 1:10 in the era of 2012
- Model is small enough to hold on single GPU

How to Implement Parameter Server

- Key considerations:
  - Server-Client: Communication bottleneck
  - Fault tolerance
  - Programming model
  - Consistency

Parameter Server Implementation

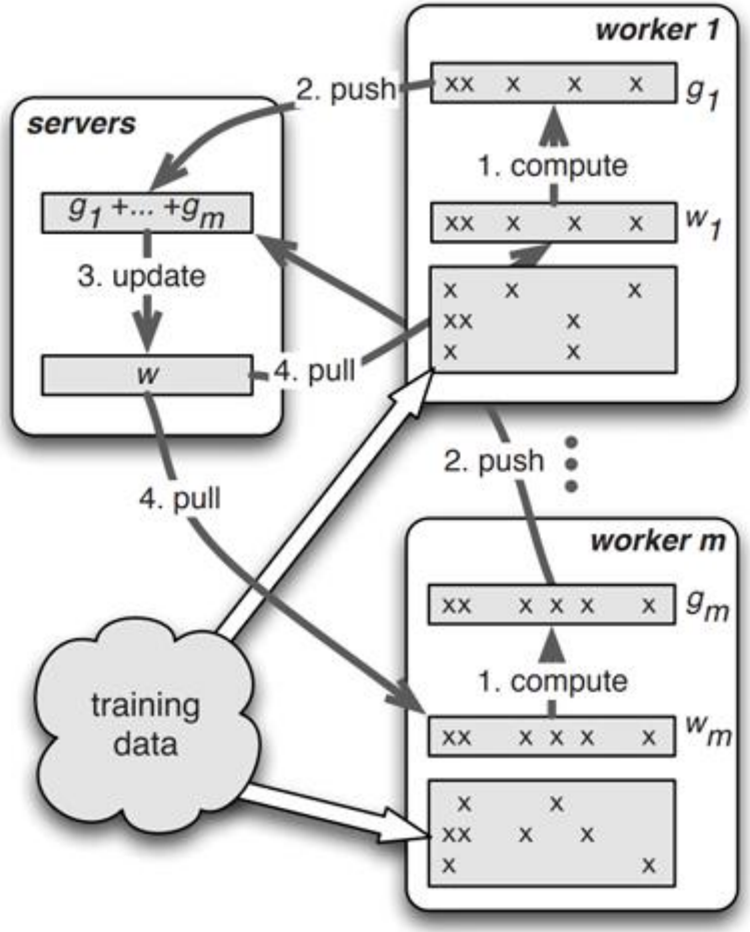


1. Partition the training data
2. Parallel training on different machines
3. Synchronize the local updates
4. Refresh local model with new parameters, then go to 2

`Scaling Distributed Machine Learning with the Parameter Server, 2014`

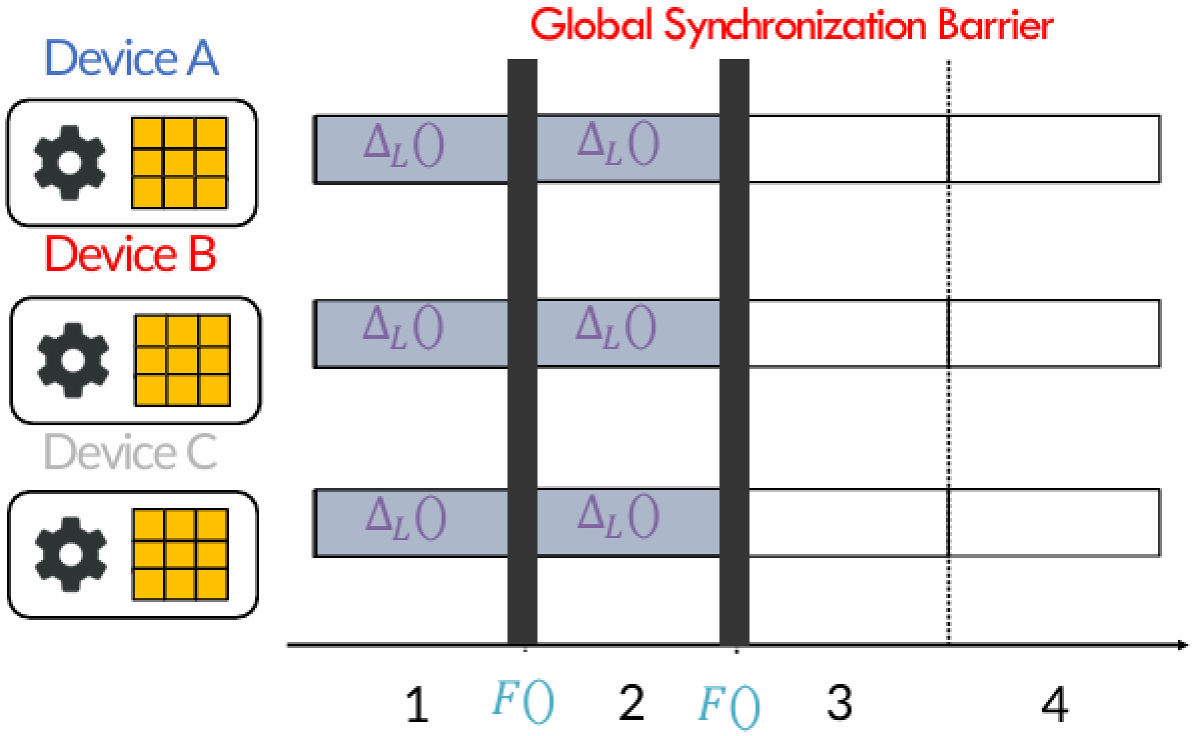
Programming Model

- Client:
  - Pull()
  - Compute()
  - Push()
- Server:
  - Update()
- Very similar to the spirit of MapReduce
- Flexible and can be used in various ML algorithms, not just DL



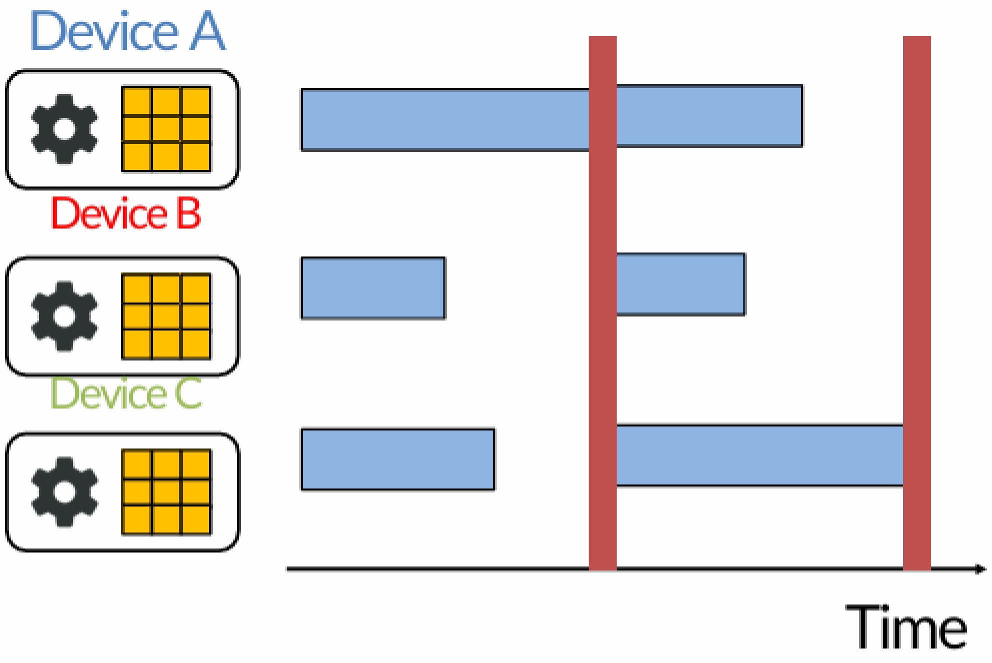
Bulk Synchronous Parallel: Consistency

$$\theta^{(t+1)} = \theta^{(t)} + \varepsilon \sum_{p=1}^{P} \nabla_{\mathcal{L}}(\theta^{(t)}, D_p^{(t)})$$



Stragglers

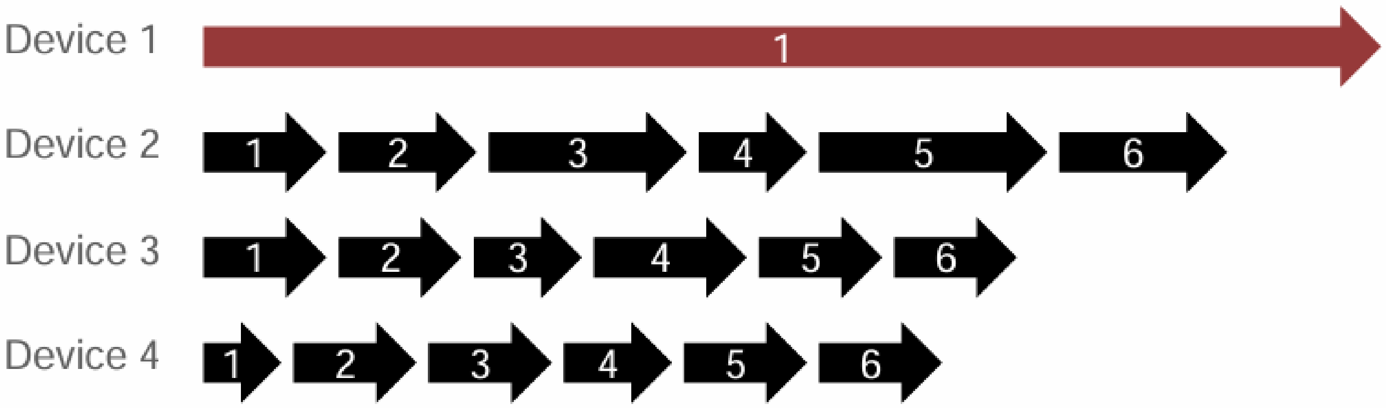
- **BSP suffers from stragglers**
  - Slow devices (stragglers) force all devices to wait
  - More devices → higher chance of having a straggler



Asynchronous Communication (No Consistency)

- **Asynchronous (Async):** removes all communication barriers
  - Maximizes computing time
  - Transient stragglers will cause messages to be extremely stale
    - Ex: Device 2 is at $t = 6$, but Device 1 has only sent message for $t = 1$

- **Some Async software:** messages can be applied while computing $F()$, $\Delta_L()$
  - Unpredictable behavior, can hurt statistical efficiency!

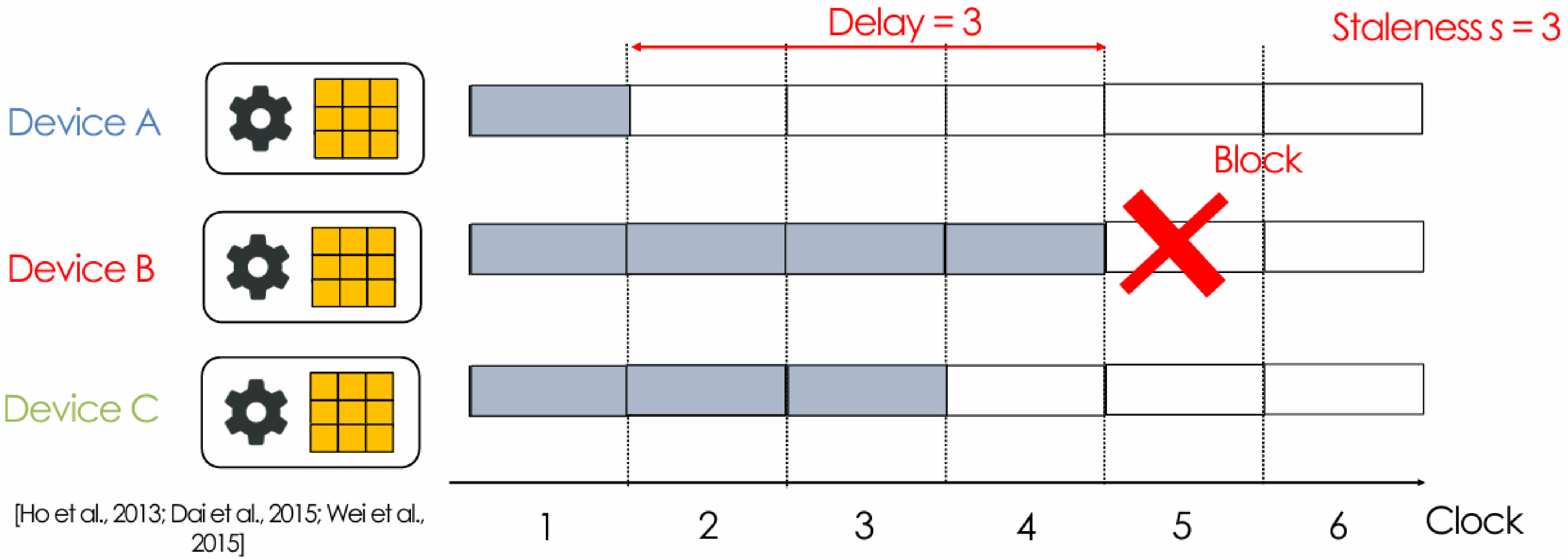


Bounded Consistency

Bounded consistency models: Middle ground between BSP and fully-asynchronous (no-barrier)

e.g. Stale Synchronous Parallel (SSP): Devices allowed to iterate at different speeds

- Fastest & slowest device must not drift > s iterations apart (in this example, $s = 3$)
  - s is the maximum staleness



[Ho et al., 2013; Dai et al., 2015; Wei et al., 2015]

Impacts of Consistency/Staleness: Unbounded Staleness