https://hao-ai-lab.github.io/dsc291-s24/

DSC 291: ML Systems
Spring 2024

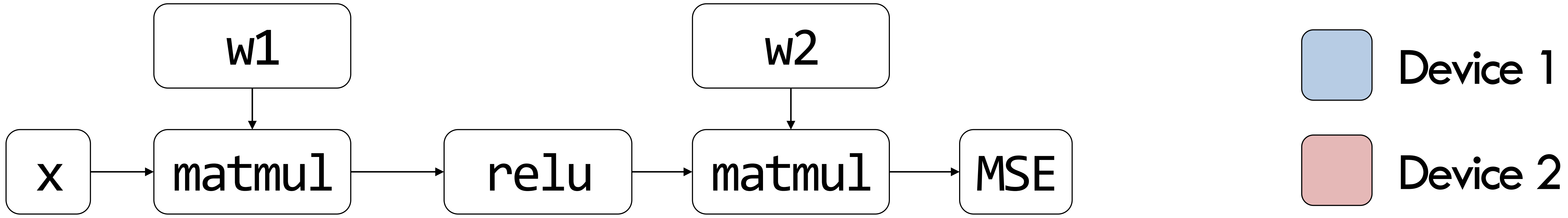
LLMs

Parallelization

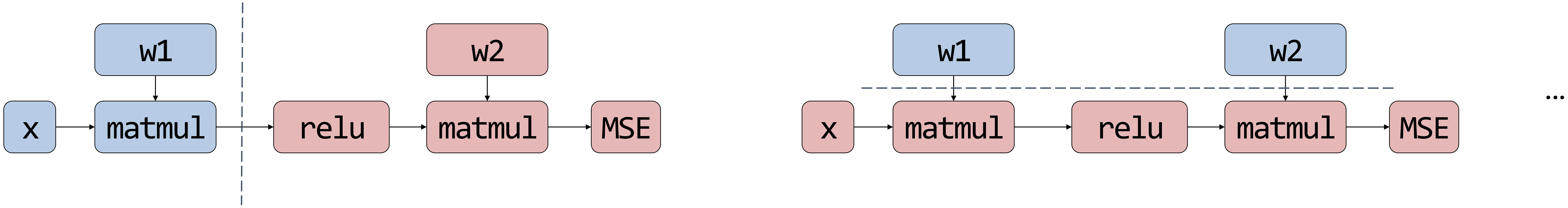
Single-device Optimization

Basics

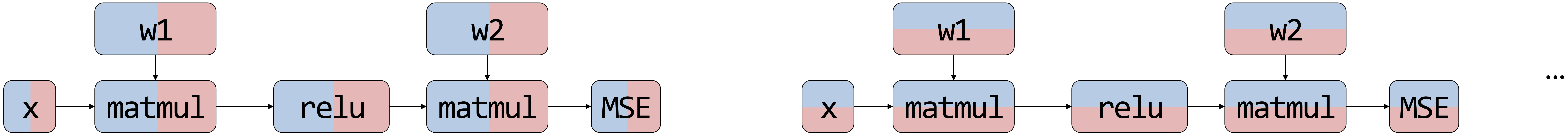
Inter-op and Intra-op Parallelism: Characteristics



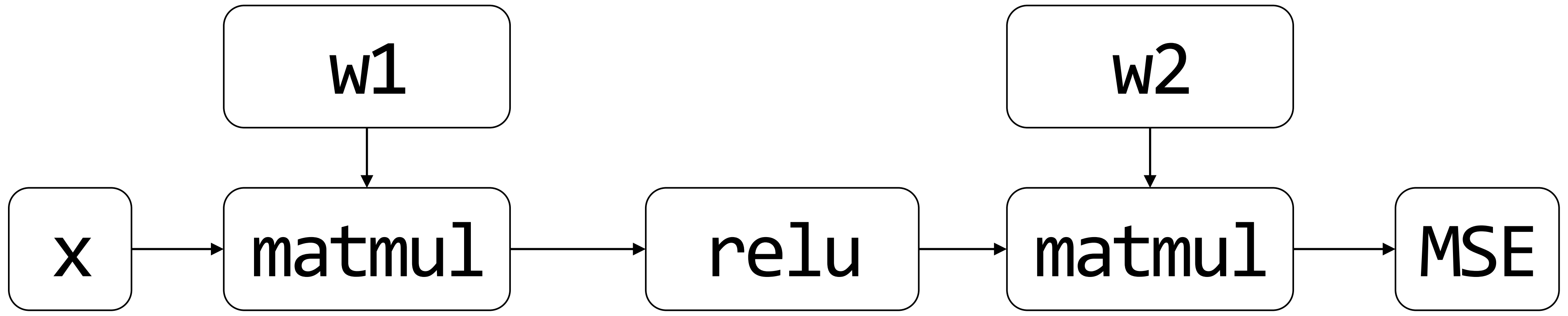
Inter-op parallelism: Requires point-to-point communication but results in device idle



Intra-op parallelism: Devices are busy but requires collective communication



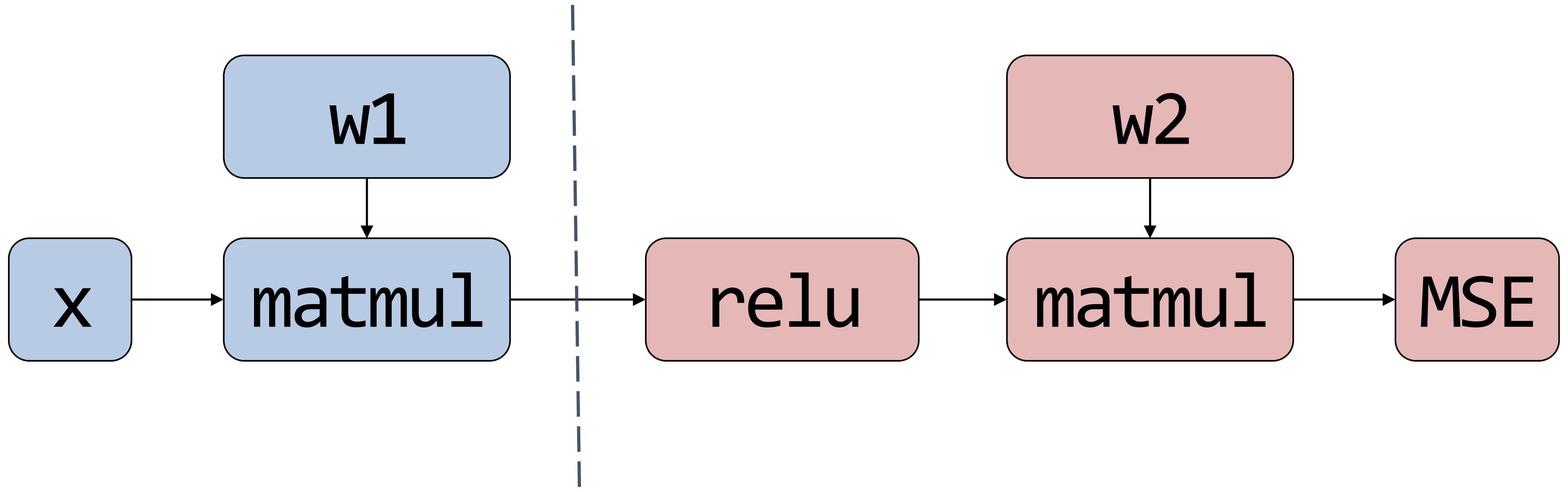
Inter-op and Intra-op Parallelism: Characteristics



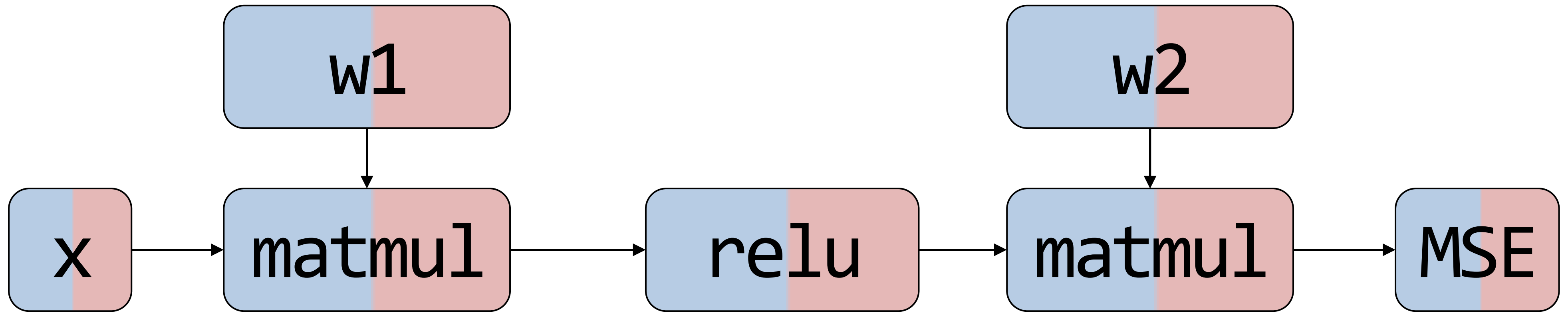
Device 1

Device 2

Inter-op parallelism



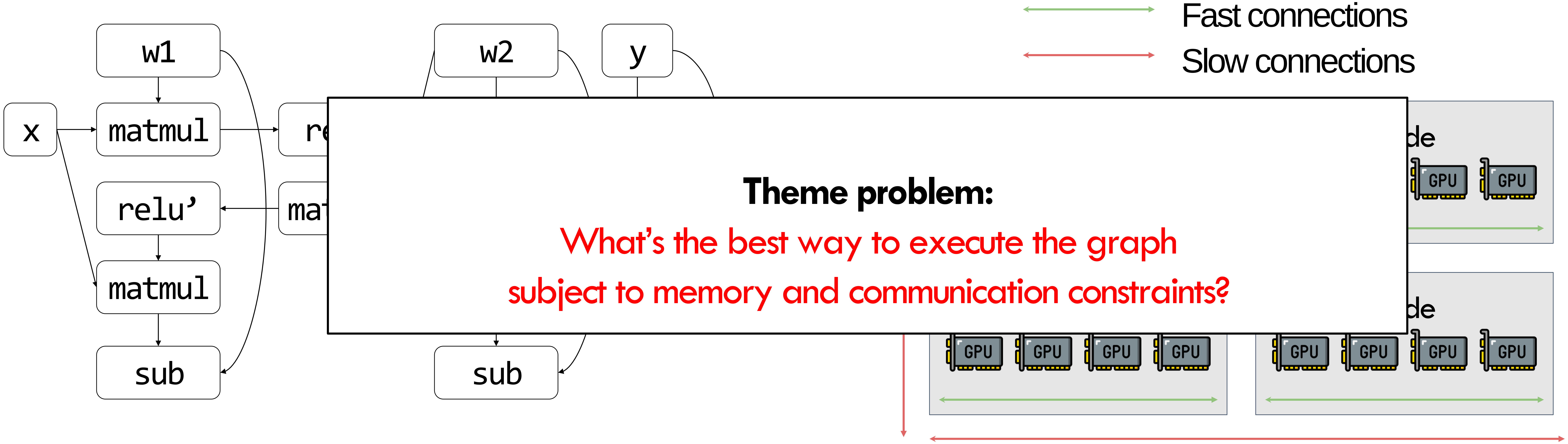
Intra-op parallelism



Trade-off

	Inter-operator Parallelism	**Intra-operator Parallelism**
Communication	Less	More
Device Idle Time	More	Less

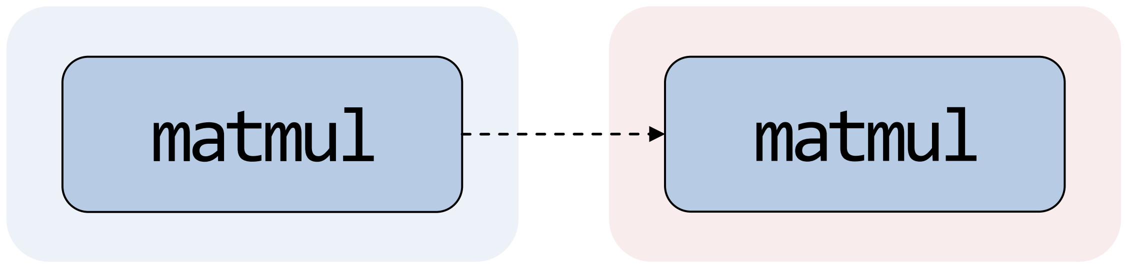
ML Parallelization under New View

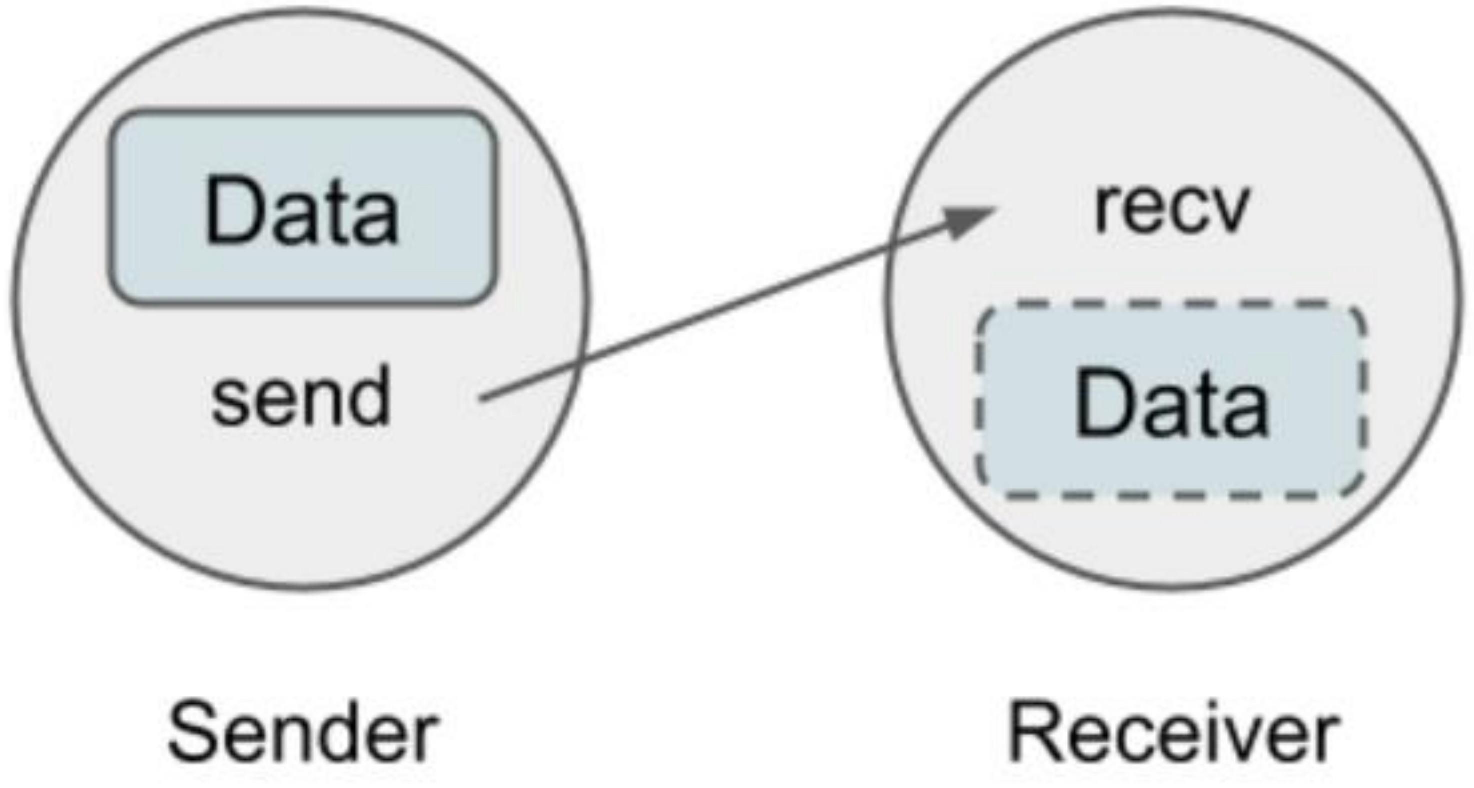


Theme problem:

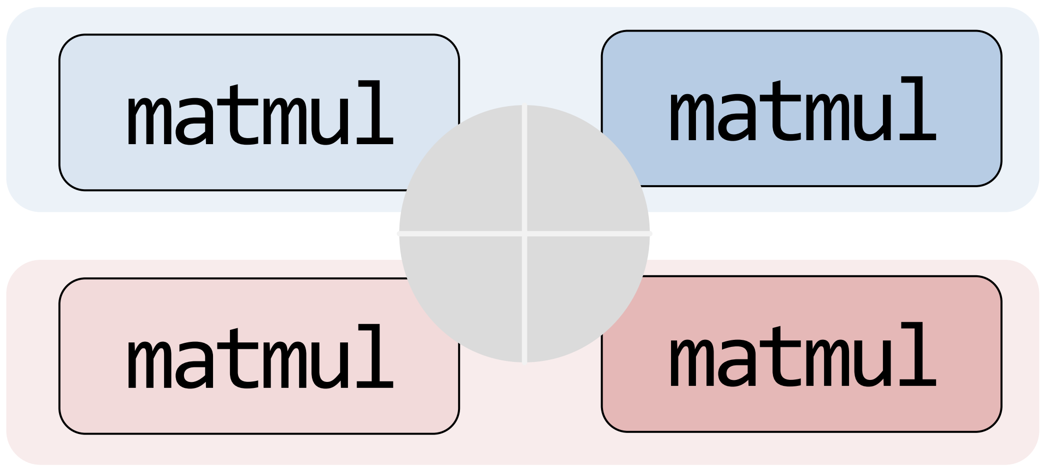
What’s the best way to execute the graph subject to memory and communication constraints?

Terminologies: Point-to-point Communication





Terminologies: Collective Communication




```python
ddp_model = DDP(Model(), device_ids=[rank])
for batch in data_loader:
    loss = train_step(ddp_model, batch)
```

Implicit allreduce here

all-reduce

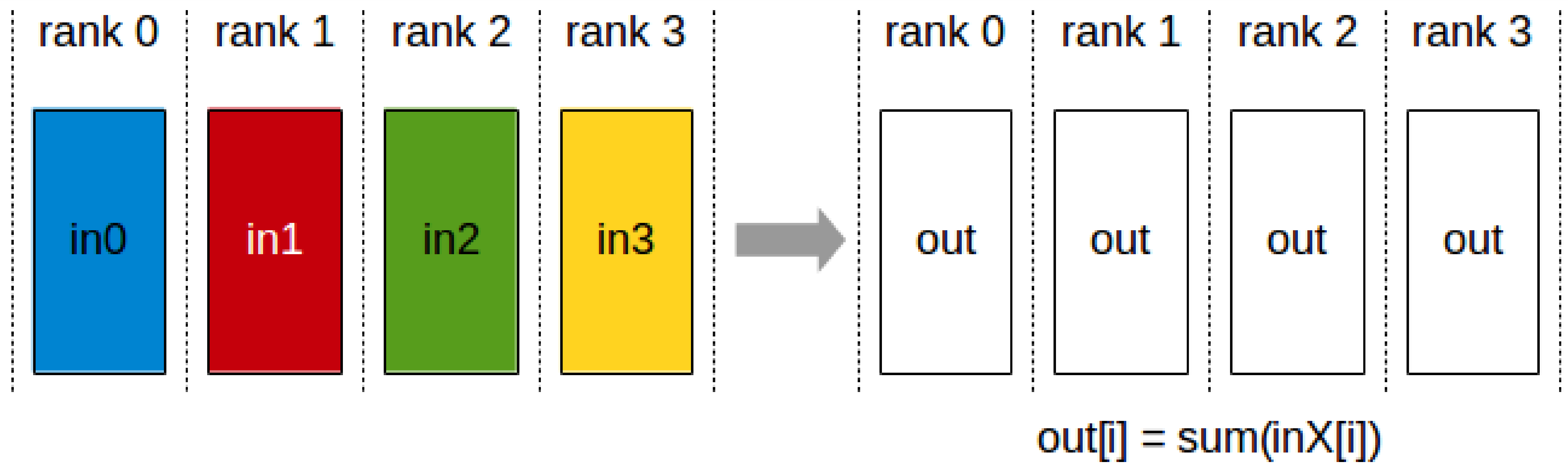
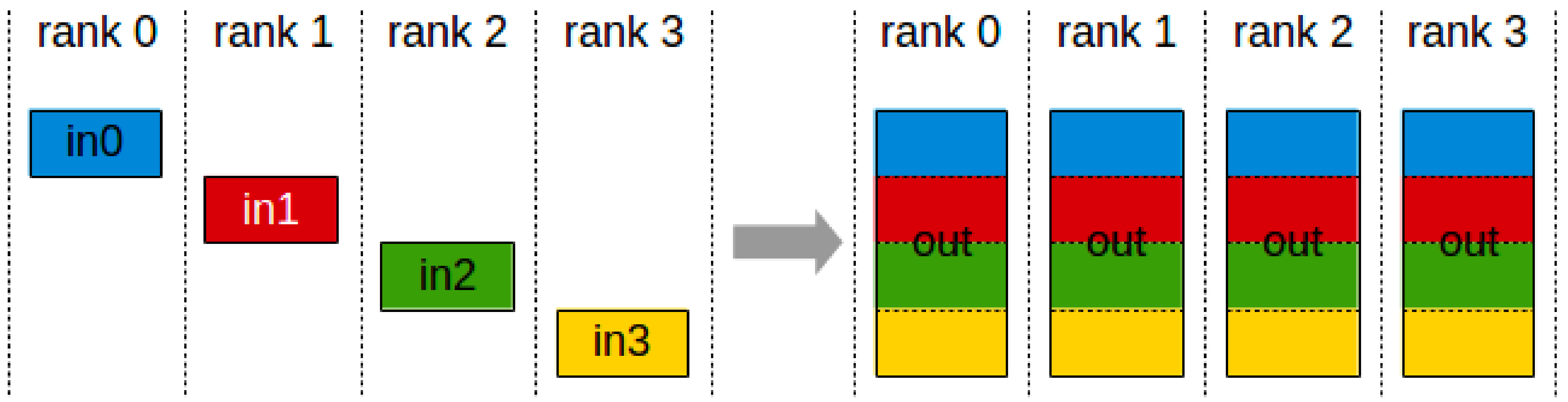


Figure from NCCL documentation

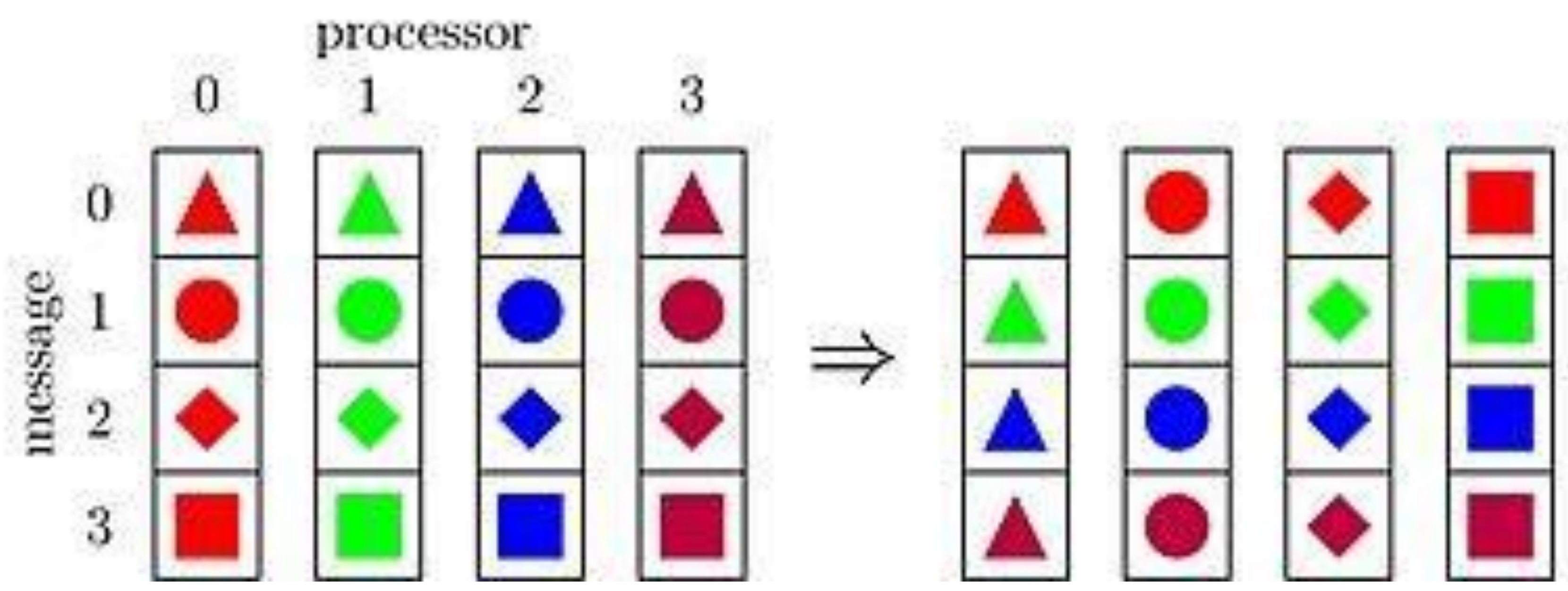
Terminologies: Collective Communication

all-gather

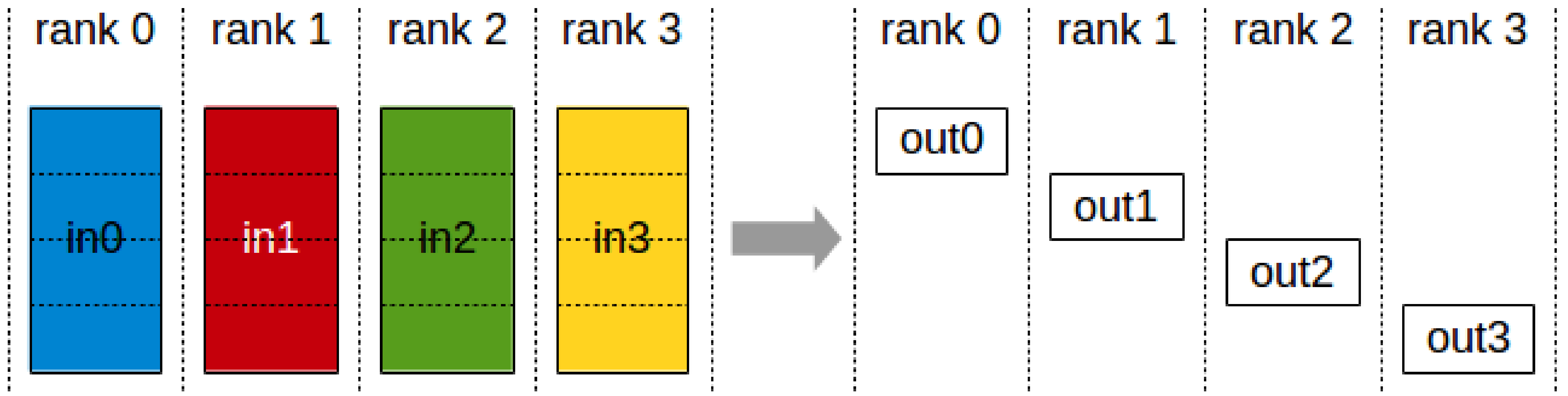


out[Y*count+i] = inY[i]

all-to-all



Reduce-scatter



outY[i] = sum(inX[Y*count+i])

Figures from NCCL documentation

Some Basics

- Collective is much more expensive than P2P
  - Collective can be assembled using many P2P
- Collective is high optimized throughout the past 20 years
  - Look for “X”CCL libraries
    - NCCL, MCCL,
- Collective is not fault-tolerant
- Collective: Minimal spanning Tree vs. Ring
  - MST: latency ++
  - Ring: bandwidth utilization ++

Some Transformations

Reduce-scatter

$$(p-1)\alpha+\frac{p-1}{p}n(\beta+\gamma)$$

Scatter

$$\log(p)\alpha+\frac{p-1}{p}n\beta$$

Gather

$$\log(p)\alpha+\frac{p-1}{p}n\beta$$

Allgather

$$(p-1)\alpha+\frac{p-1}{p}n\beta$$

Reduce(-to-one)

Allreduce

Broadcast

Recap

Reduce-scatter

$$(p-1)\alpha+\frac{p-1}{p}n(\beta+\gamma)$$

Scatter

$$\log(p)\alpha+\frac{p-1}{p}n\beta$$

Gather

$$\log(p)\alpha+\frac{p-1}{p}n\beta$$

Allgather

$$(p-1)\alpha+\frac{p-1}{p}n\beta$$

Reduce(-to-one)

$$(p-1+\log(p))\alpha+\frac{p-1}{p}n(2\beta+\gamma)$$

Allreduce

Broadcast

Recap

Reduce-scatter

$$(p-1)\alpha+\frac{p-1}{p}n(\beta+\gamma)$$

Scatter

$$log(p)\alpha+\frac{p-1}{p}n\beta$$

Gather

$$log(p)\alpha+\frac{p-1}{p}n\beta$$

Allgather

$$(p-1)\alpha+\frac{p-1}{p}n\beta$$

Reduce(-to-one)

$$(p-1+log(p))\alpha+\frac{p-1}{p}n(2\beta+\gamma)$$

Allreduce

$$2(p-1)\alpha+\frac{p-1}{p}n(2\beta+\gamma)$$

Broadcast

$$(log(p)+p-1)\alpha+2\frac{p-1}{p}n\beta$$

Recap

Reduce-scatter

$$(p-1)\alpha+\frac{p-1}{p}n(\beta+\gamma)$$

Scatter

$$log(p)\alpha+\frac{p-1}{p}n\beta$$

Gather

$$log(p)\alpha+\frac{p-1}{p}n\beta$$

Allgather

$$(p-1)\alpha+\frac{p-1}{p}n\beta$$

Reduce(-to-one)

$$(p-1+log(p))\alpha+\frac{p-1}{p}n(2\beta+\gamma)$$

Allreduce

$$2(p-1)\alpha+\frac{p-1}{p}n(2\beta+\gamma)$$

Broadcast

Recap

Reduce-scatter

$(p-1)\alpha+\frac{p-1}{p}n(\beta+\gamma)$

Scatter

$log(p)\alpha+\frac{p-1}{p}n\beta$

Gather

$log(p)\alpha+\frac{p-1}{p}n\beta$

Allgather

$(p-1)\alpha+\frac{p-1}{p}n\beta$

Reduce(-to-one)

$(p-1+log(p))\alpha+\frac{p-1}{p}n(2\beta+\gamma)$

Allreduce

$2(p-1)\alpha+\frac{p-1}{p}n(2\beta+\gamma)$

Broadcast

$(log(p)+p-1)\alpha+2\frac{p-1}{p}n\beta$

Where We Are

- Motivation
- History
- Parallelism Overview
- **Data parallelism**
- Model parallelism
  - Inter and intra-op parallelism
- Auto-parallelization

How to Distribute this Equation?

Gradient / backward computation

$$\theta^{(t)} = \theta^{(t-1)} + \boxed{\varepsilon \cdot \nabla_{\mathcal{L}}(\theta^{(t-1)}, D^{(t)})}$$

objective

data

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

How to perform this sum?

Two Solutions

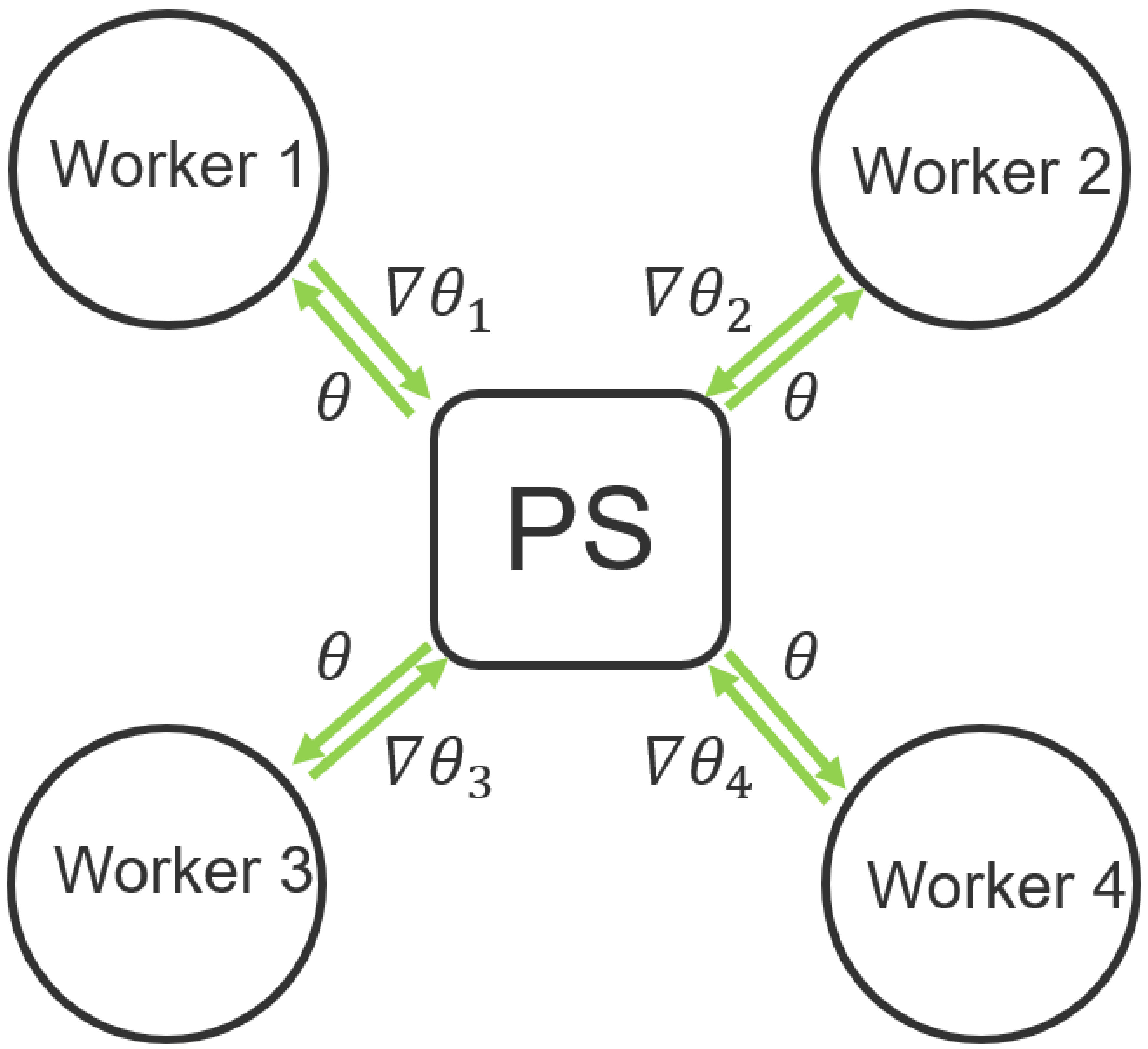
- Parameter Server
- AllReduce
- Key assumption:
  - The model can fit into an (GPU) worker memoy hence we can create many replica

Parameter Server Assumption

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

- Very heavy communication per iteration
- Compute : communication = 1:10 in the era of 2012

Parameter Server Naturally emerges

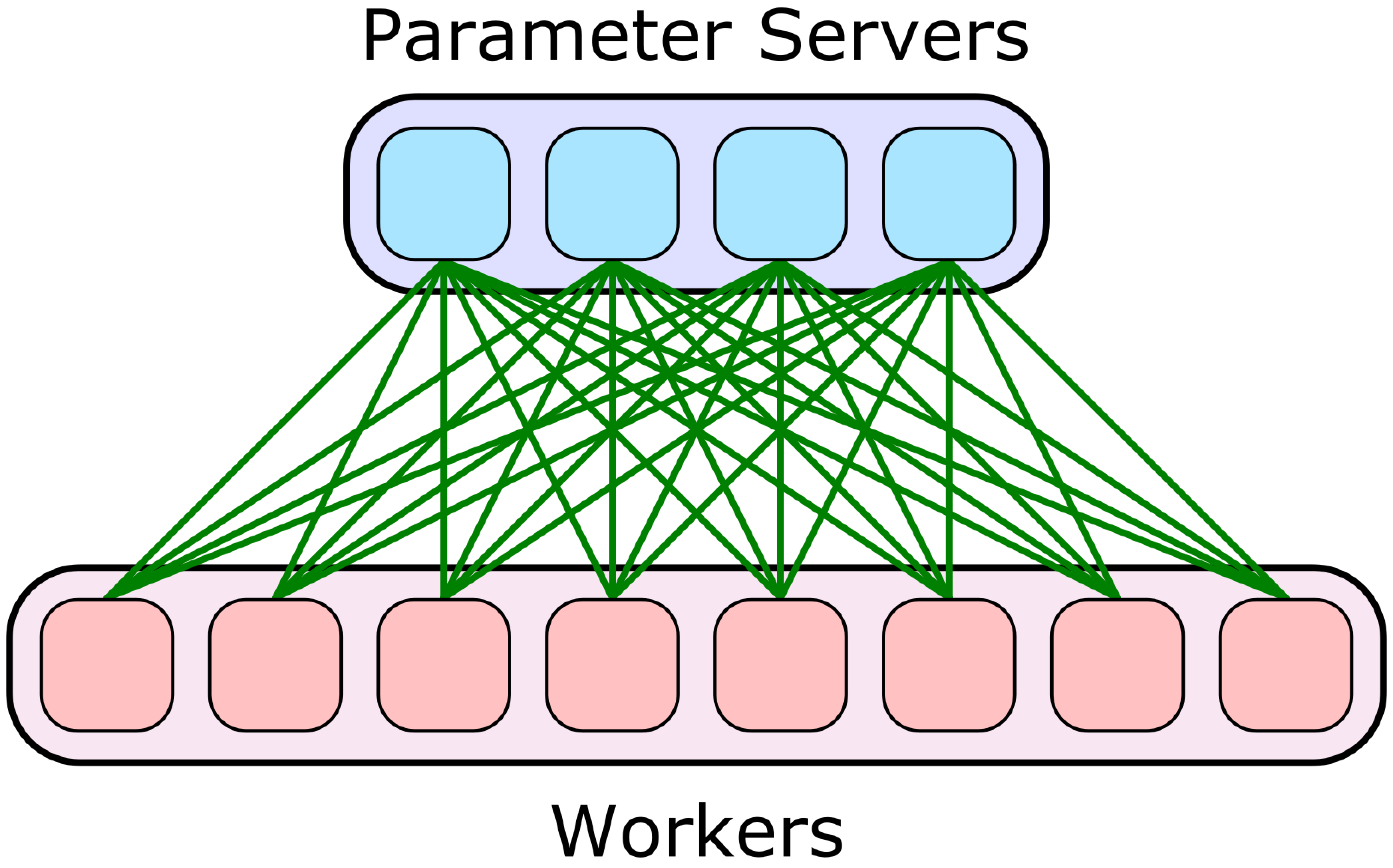


How to Implement Parameter Server?

- Key considerations:
  - Server: Communication bottleneck
  - Fault tolerance
  - Programming Model
  - Handling GPUs

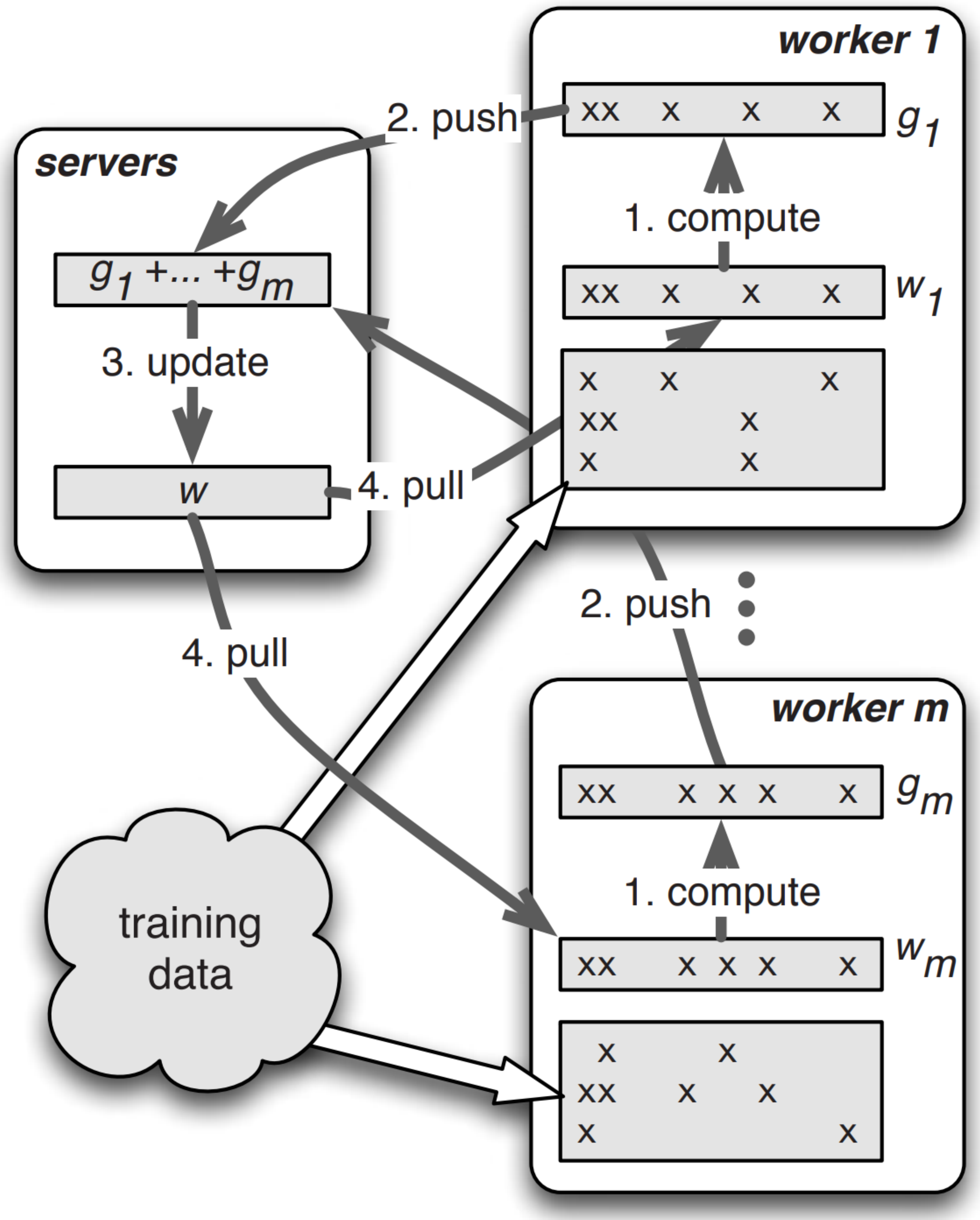
Parameter Server Implementation

- Sharded parameter server: sharded KV stores
  - Avoid communication bottleneck
  - Redundancy across different PS shards



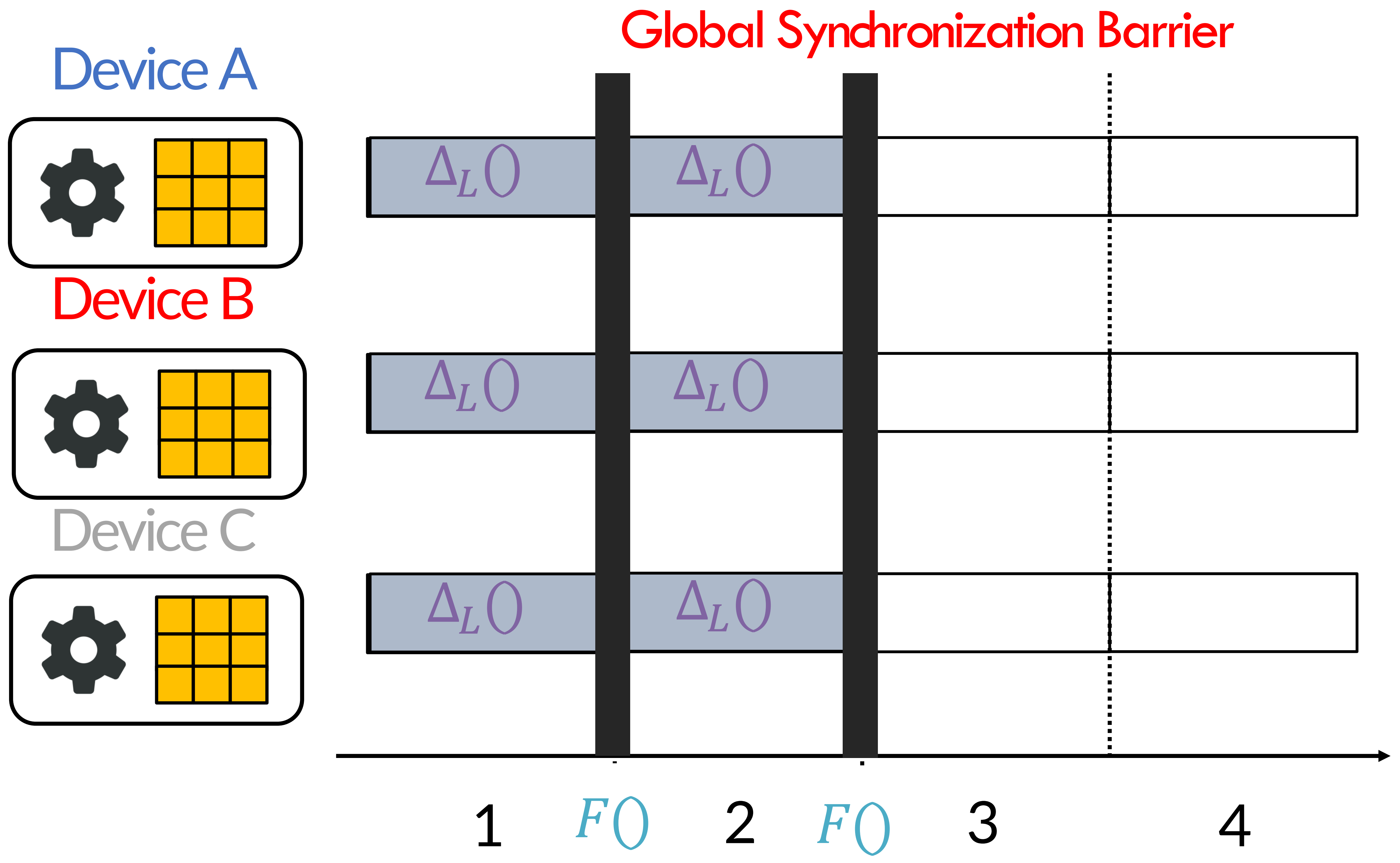
Programming Model

- Client:
  - Push()
  - Pull()
  - Compute()
- Server:
  - Update()
- Very similar to the spirit of Map Reduce
- A lot of flexibility for users to customize
  - Recall Mapreduce vs. Spark



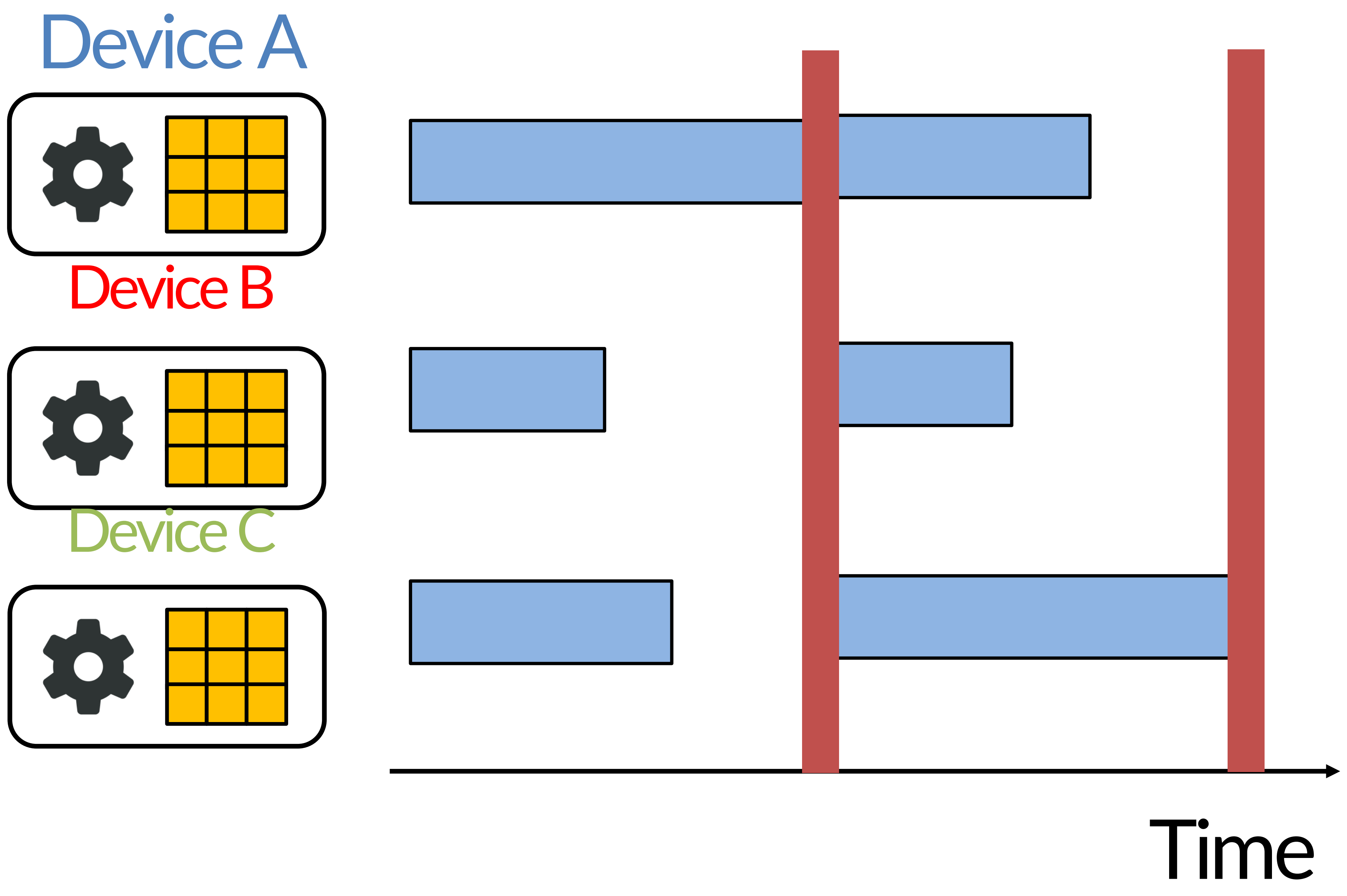
Consistency

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



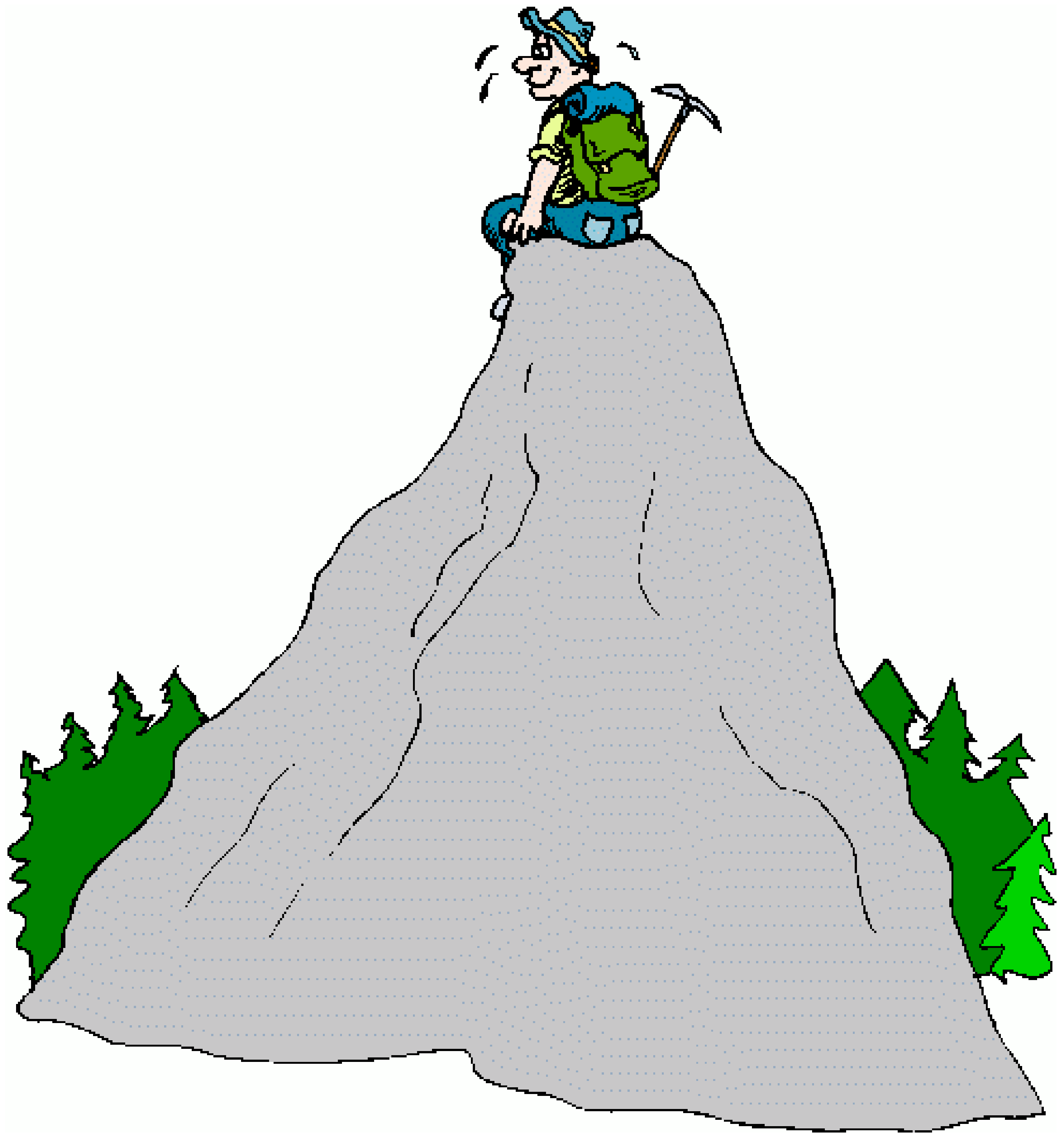
BSP's Weakness: Stragglers

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

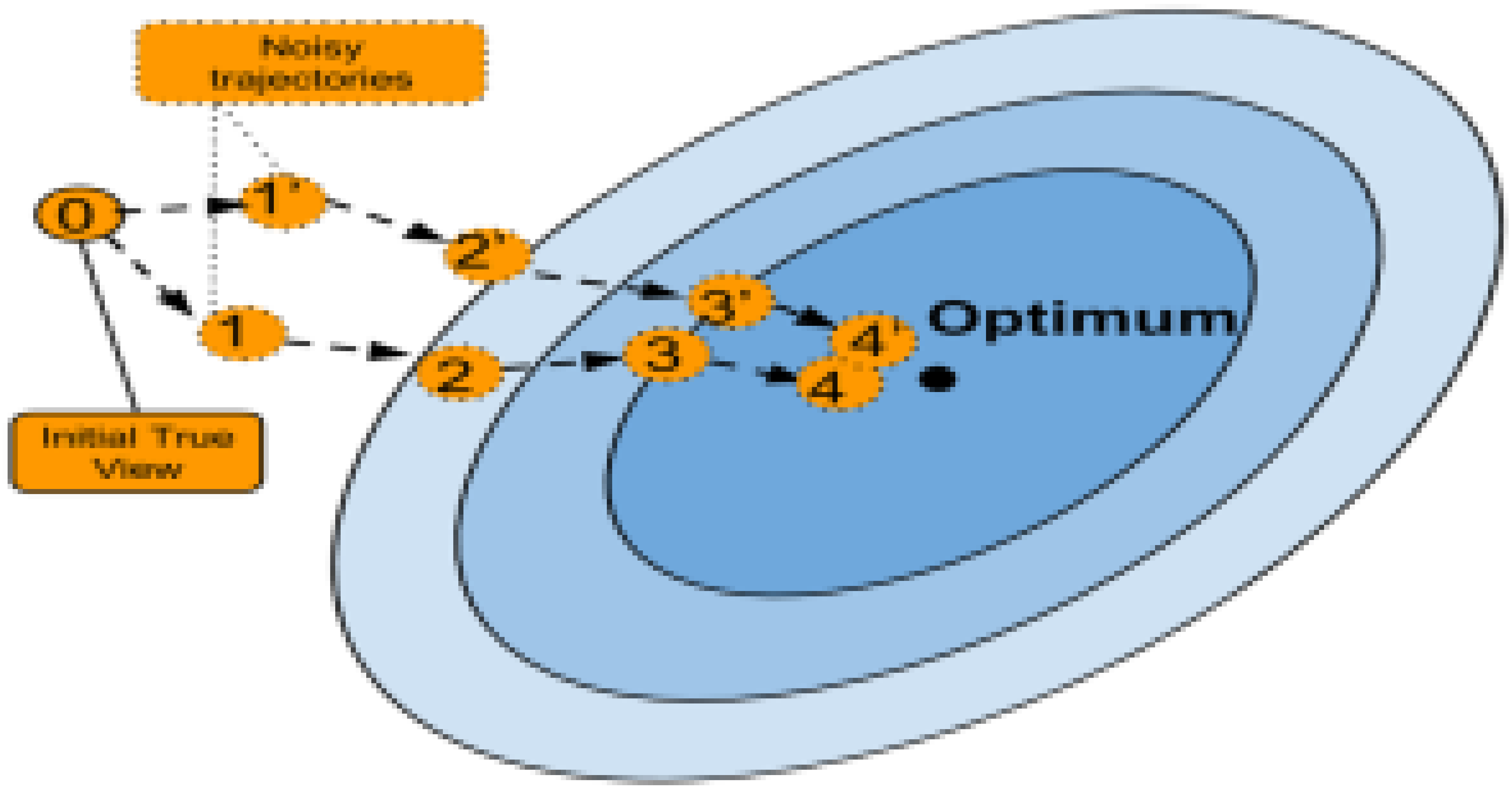


An interesting property of Gradient Descent (ascent)

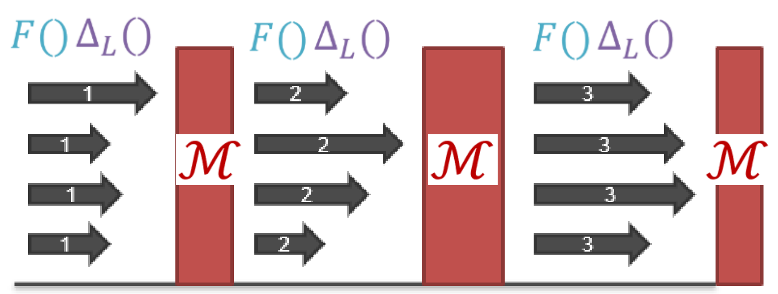
$$\theta^{(t+1)} = \theta^{(t)} + \varepsilon \sum_{p=1}^{P} \nabla_{\mathcal{L}}(\theta^{(t)}, D_p^{(t)})$$



Machine Learning is Error-tolerant (under certain conditions)



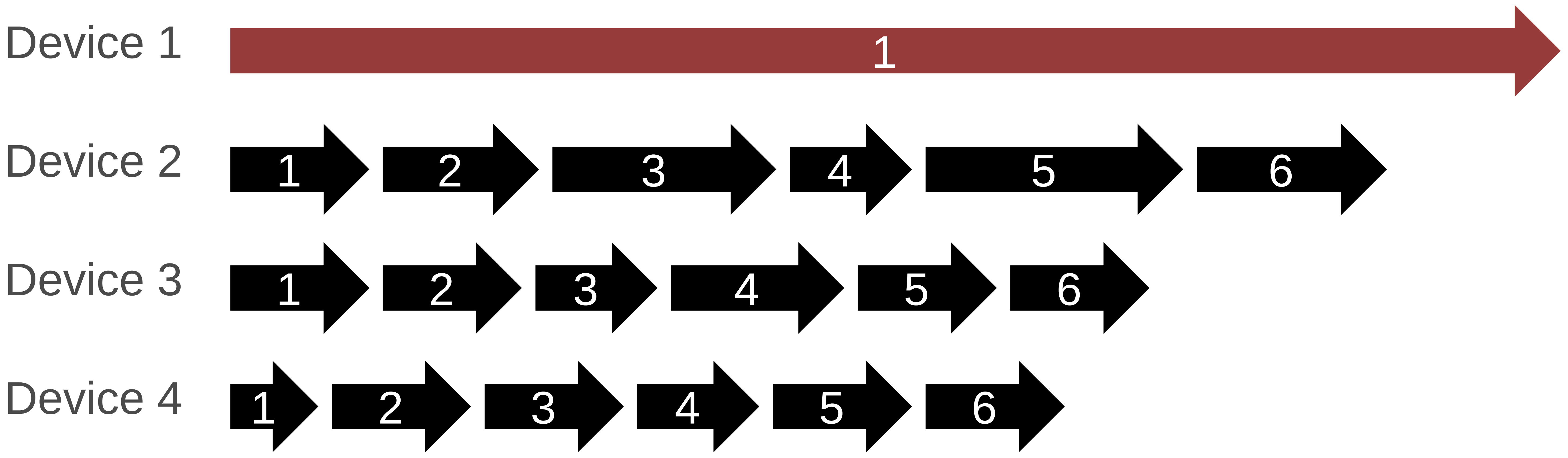
Bridging/Consistency Model



Background: 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:** <u>messages can be applied while computing</u> $F()$, $\Delta_L()$
  - Unpredictable behavior, can hurt statistical efficiency!

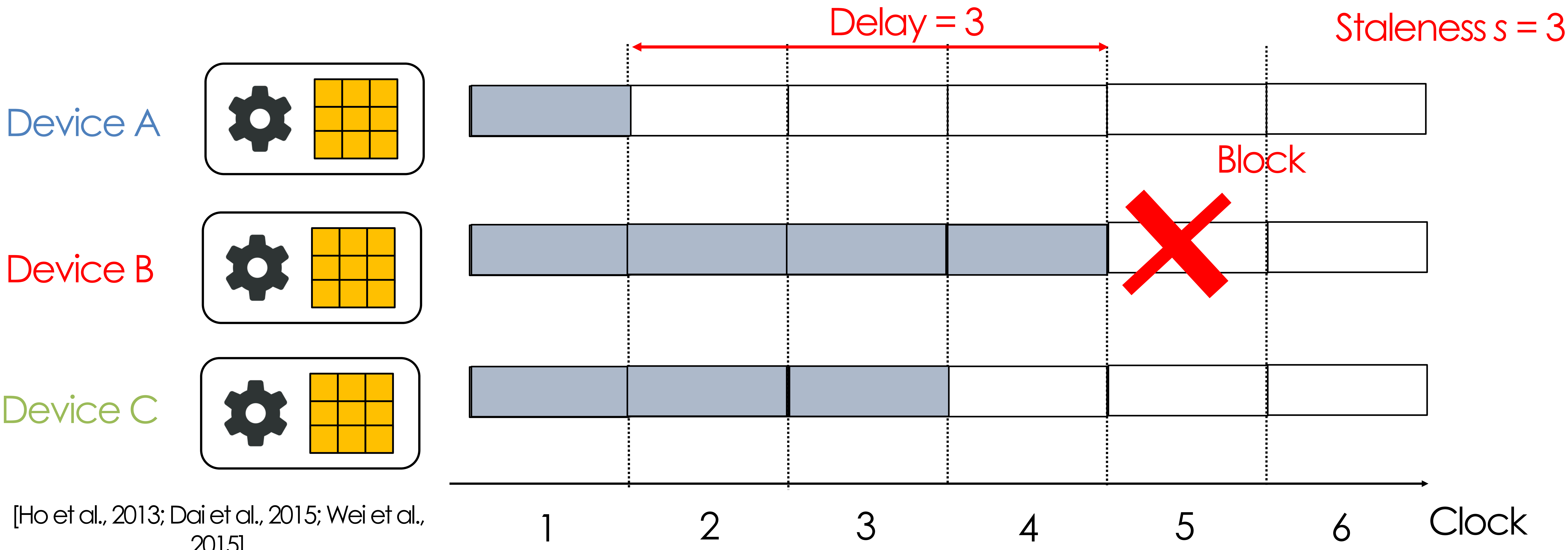


Background: Bounded Consistency

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

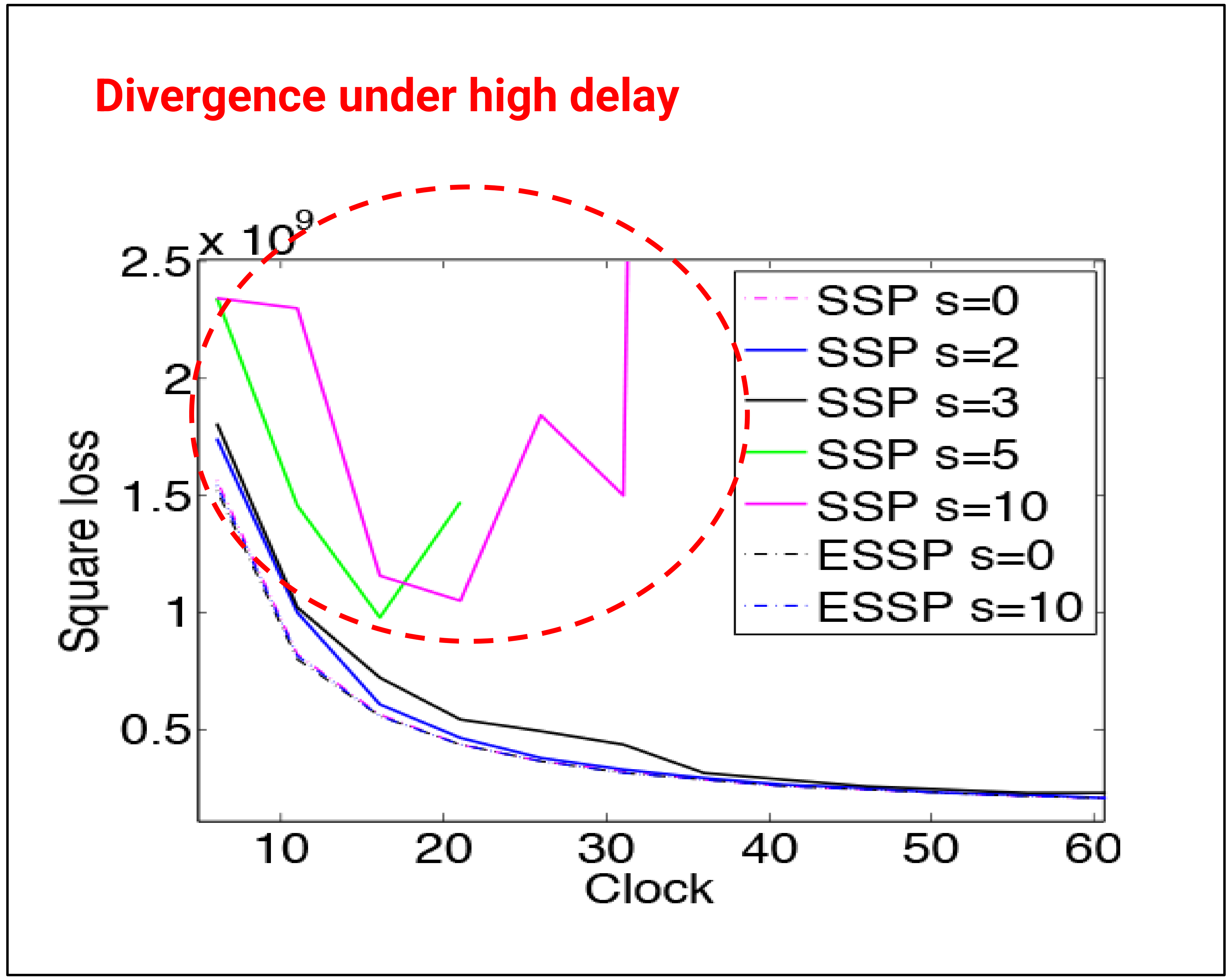
e.g. Stale Synchronous Parallel (SSP): Devices allowed to iterate at different speeds

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

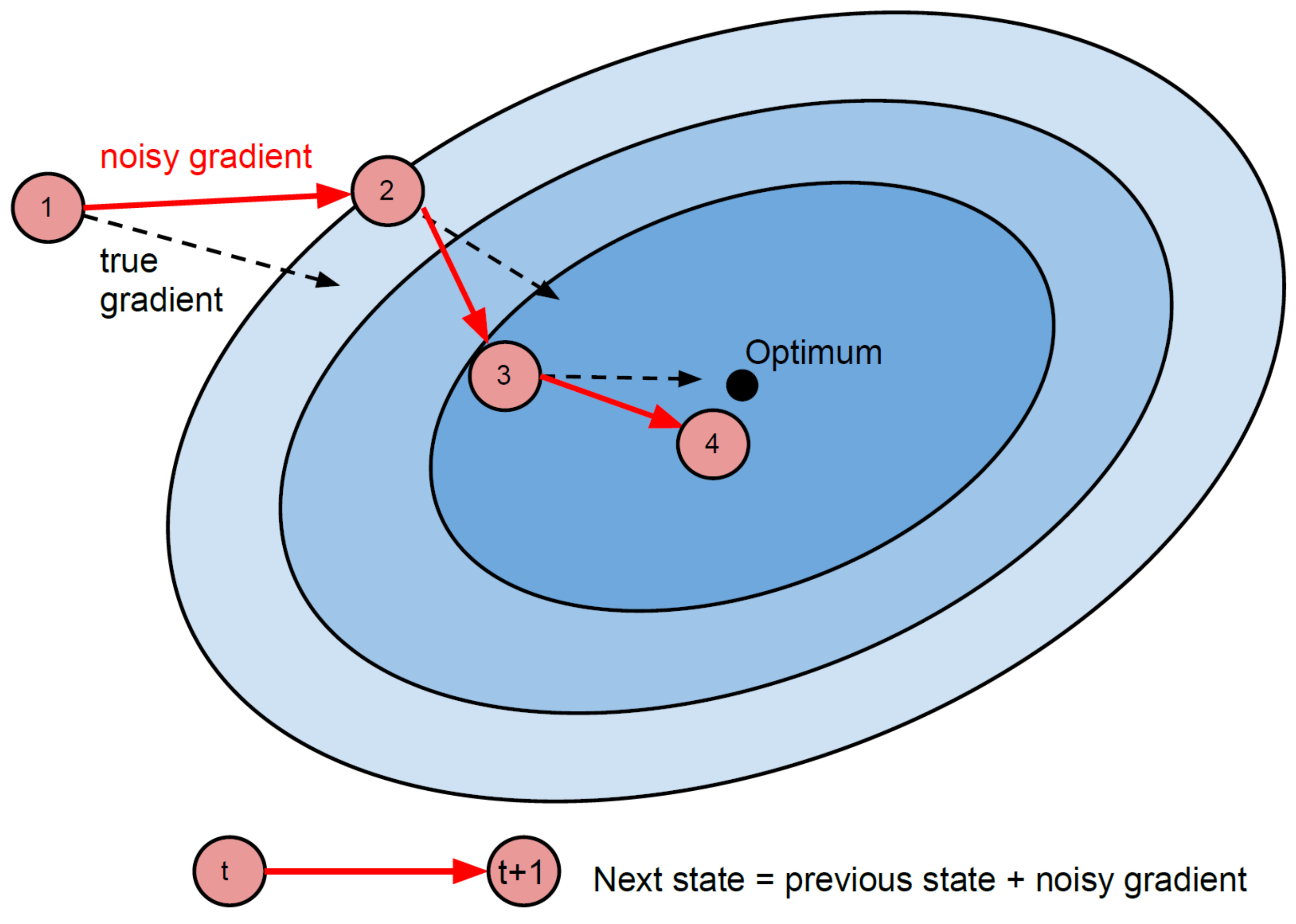


[Ho et al., 2013; Dai et al., 2015; Wei et al., 2015]

Impacts of Consistency/Staleness: Unbounded Staleness



Theory: SSP Expectation Bound

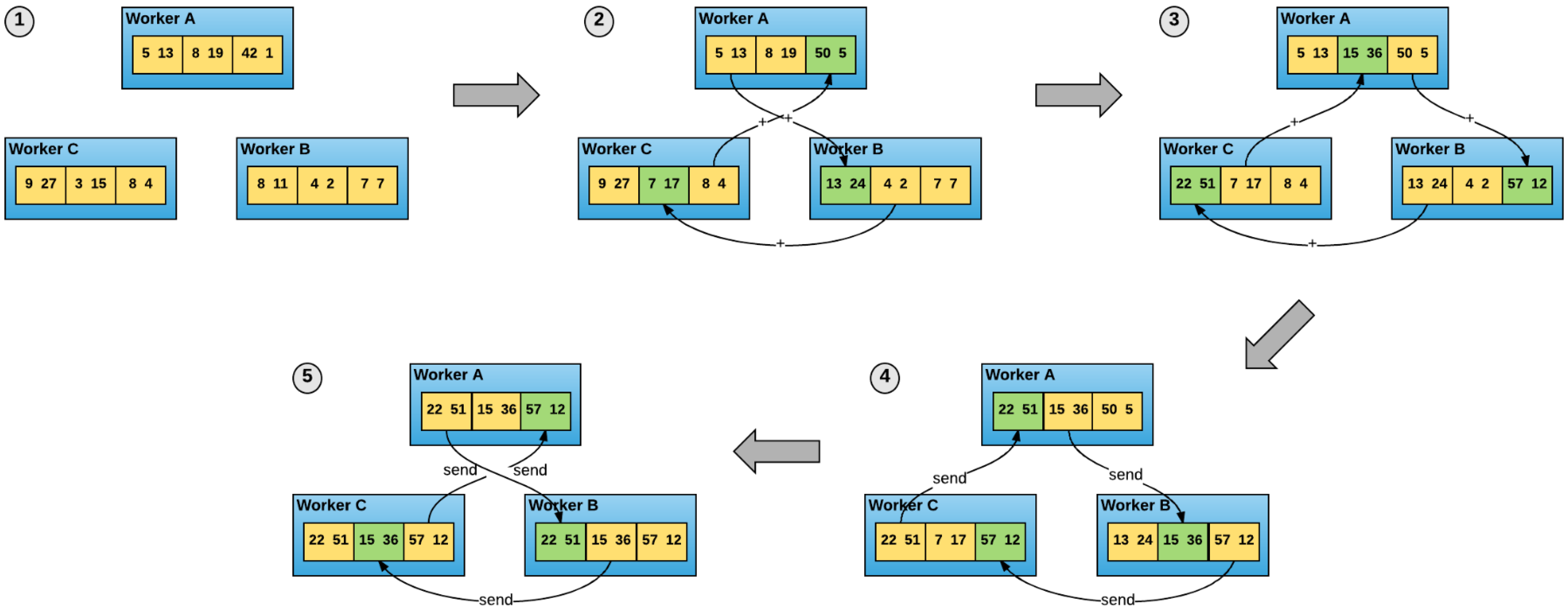


$$R[\mathbf{X}] := \underbrace{\left[\frac{1}{T}\sum_{t=1}^{T} f_t(\tilde{\mathbf{x}}_t)\right] - f(\mathbf{x}^*)}_{\text{Difference between SSP estimate and true optimum}} \leq 4FL\sqrt{\frac{2(s+1)P}{T}}$$

Summary: Parameter Server

- Why did it emerge?
- Why did it become irrelevant?

AllReduce



1
Worker A
5 13
8 19
42 1
Worker C
9 27
3 15
8 4
Worker B
8 11
4 2
7 7
2
Worker A
5 13
8 19
50 5
Worker C
9 27
7 17
8 4
Worker B
13 24
4 2
7 7
3
Worker A
5 13
15 36
50 5
Worker C
22 51
7 17
8 4
Worker B
13 24
4 2
57 12
4
Worker A
22 51
15 36
50 5
send
send
Worker C
22 51
7 17
57 12
Worker B
13 24
15 36
57 12
send
5
Worker A
22 51
15 36
57 12
send
send
Worker C
22 51
15 36
57 12
Worker B
22 51
15 36
57 12
send

Allreduce

- Initially implemented in Horovod
- Being Optimized by nvidia (hw/sw cooptimizaiton)
- Being adopted in PyTorch DDP

Where We Are

- Motivation
- History
- Parallelism Overview
- Data parallelism
- **Model parallelism**
  - Inter and intra-op parallelism
- Auto-parallelization