Data-Parallel Deep Learning

Efficient DL, Episode V, 2023

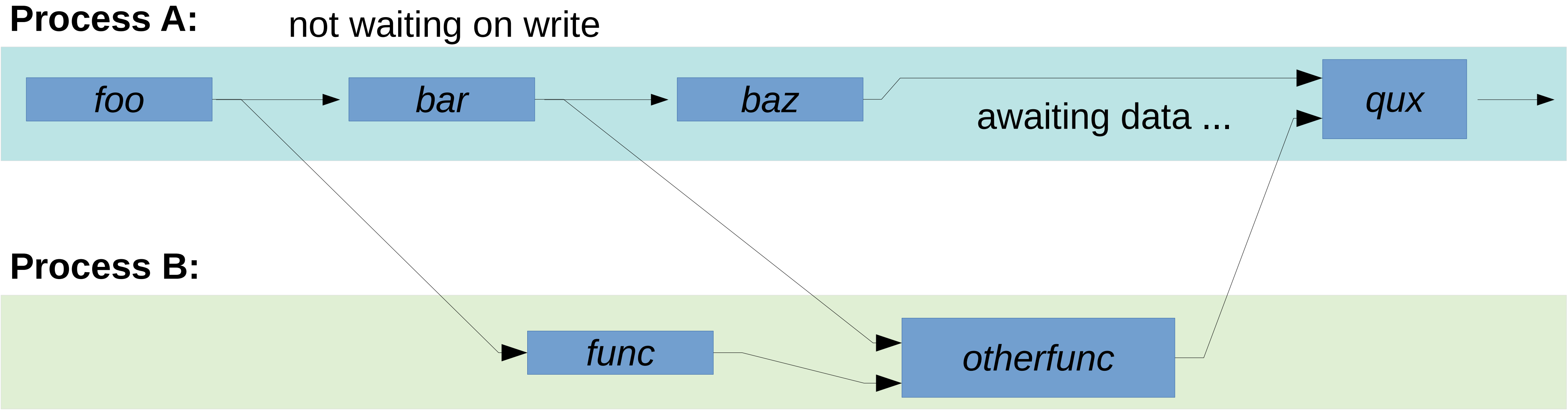
Yandex
Research

LAMBDA

British Hedgehog
Preservation Society

Previously on My Little Pony

MP Rules



Channel (pipe):

- Communication in O(message size)
- **Asynchronous** read/write

Large problems need large models

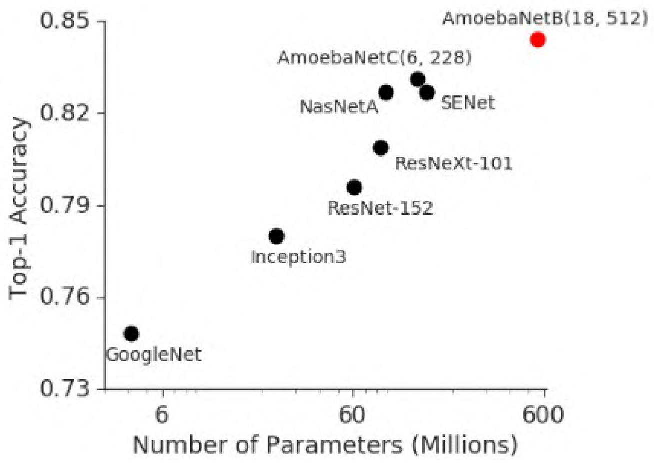
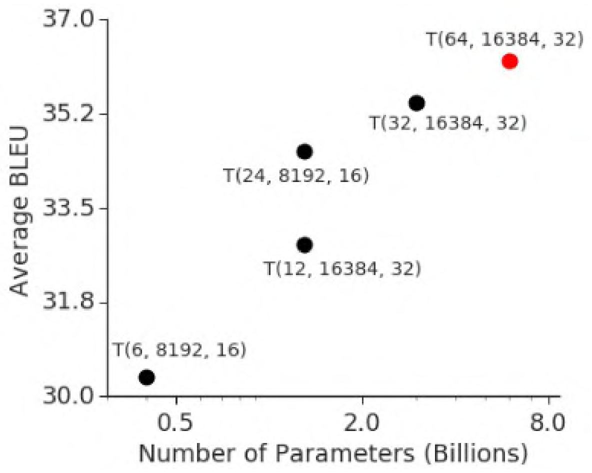


Image Classification
ImageNet



Machine Translation
average over WMT

Source: https://arxiv.org/abs/1811.06965

Machine Learning Supertasks

Image classification – ImageNet, JFT300M

Generative models – ImageNet(biggan), the internet

Language Models – common crawl, BERT / MLM

Machine Translation – multilingual translation

Reinforcement Learning – playstation* & steam :)

* playstation for RL: https://arxiv.org/abs/1912.06101

Meanwhile, exabytes of YouTube videos lay dormant across the web, waiting for someone who can make use of them

Summary: operation parallelism

Data-parallel: ***???***

Model-parallel: ***???***

Summary: operation parallelism

Data-parallel: *one process applies all model on* ***partial data***

best for smaller model, more computations

Model-parallel: *one process applies* ***partial model*** *on all data*

best for larger model, fewer computations

Which one is better..

for word2vec?

In general?

Summary: operation parallelism

Data-parallel: *one process applies all model on **partial data***

best for smaller model, more computations

Model-parallel: *one process applies **partial model** on all data*

best for larger model, fewer computations

~~*Which one is better..*~~
~~*for word2vec?*~~
~~*In general?*~~

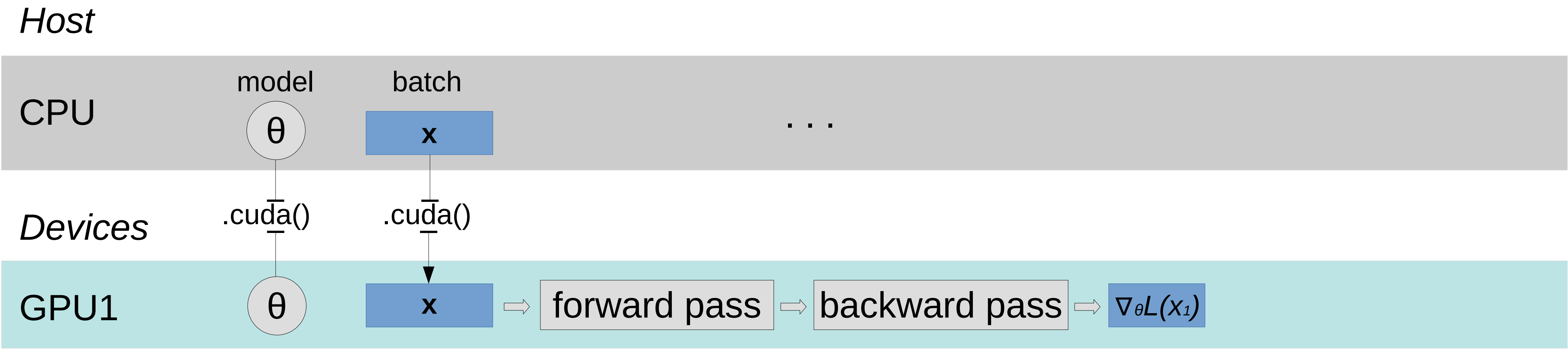
It depends…
- on model size
- on compute

Data-parallel training (naive)

cs.cmu.edu/~muli/file/parameter_server_osdi14.pdf

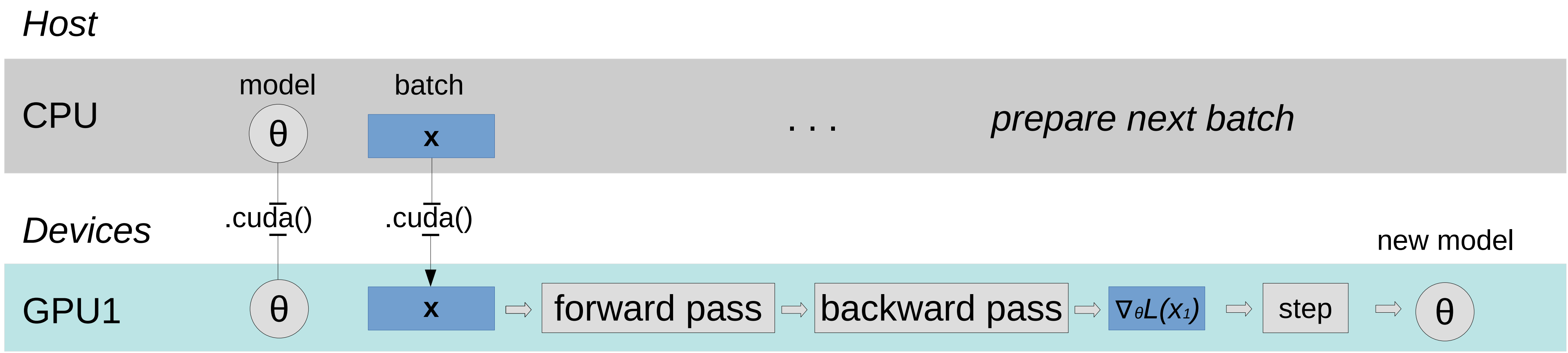
Data-parallel training (naive)

cs.cmu.edu/~muli/file/parameter_server_osdi14.pdf



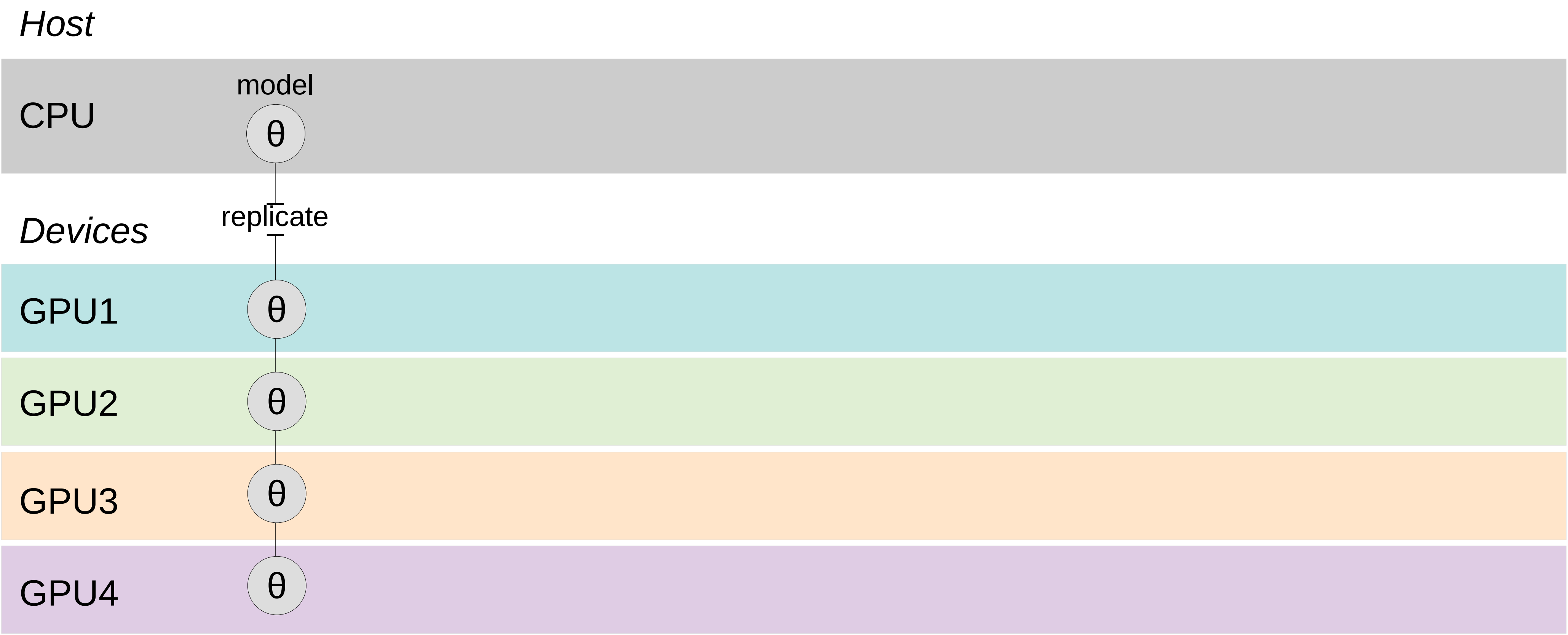
Data-parallel training (naive)

cs.cmu.edu/~muli/file/parameter_server_osdi14.pdf



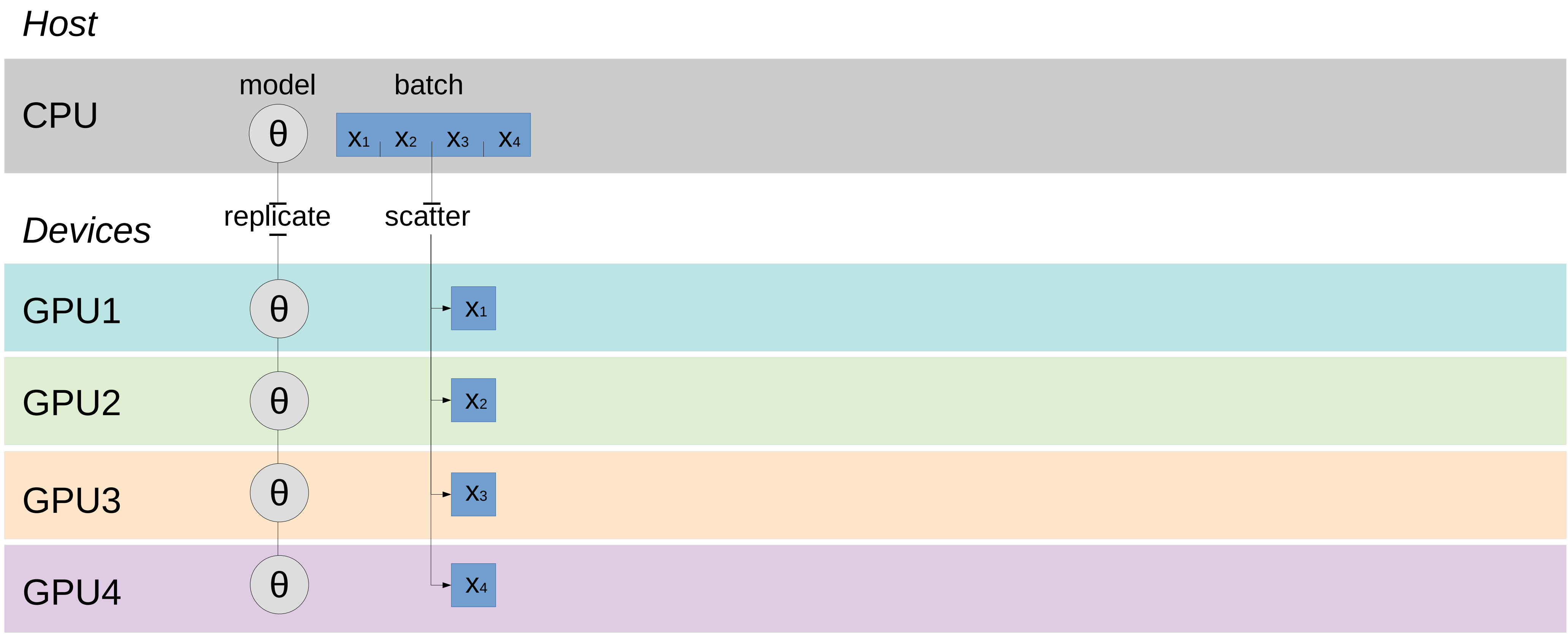
Data-parallel training (naive)

cs.cmu.edu/~muli/file/parameter_server_osdi14.pdf



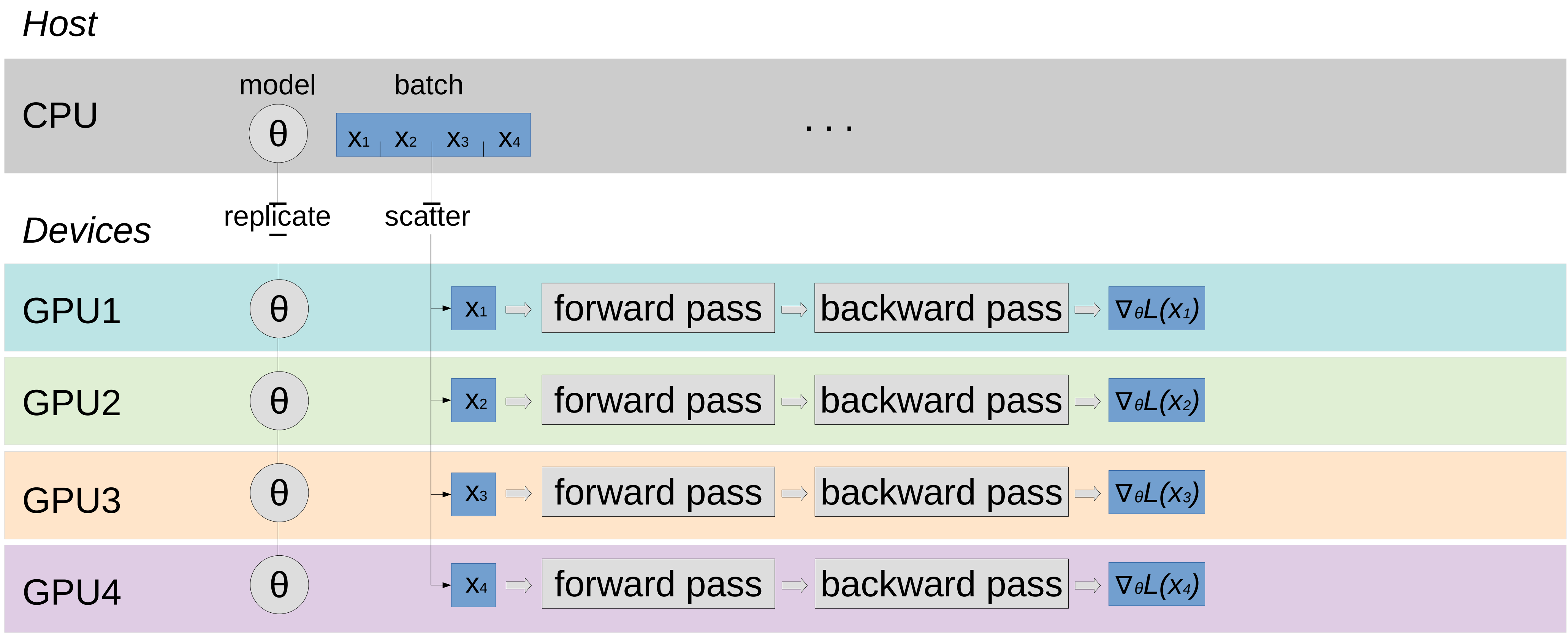
Data-parallel training (naive)

cs.cmu.edu/~muli/file/parameter_server_osdi14.pdf



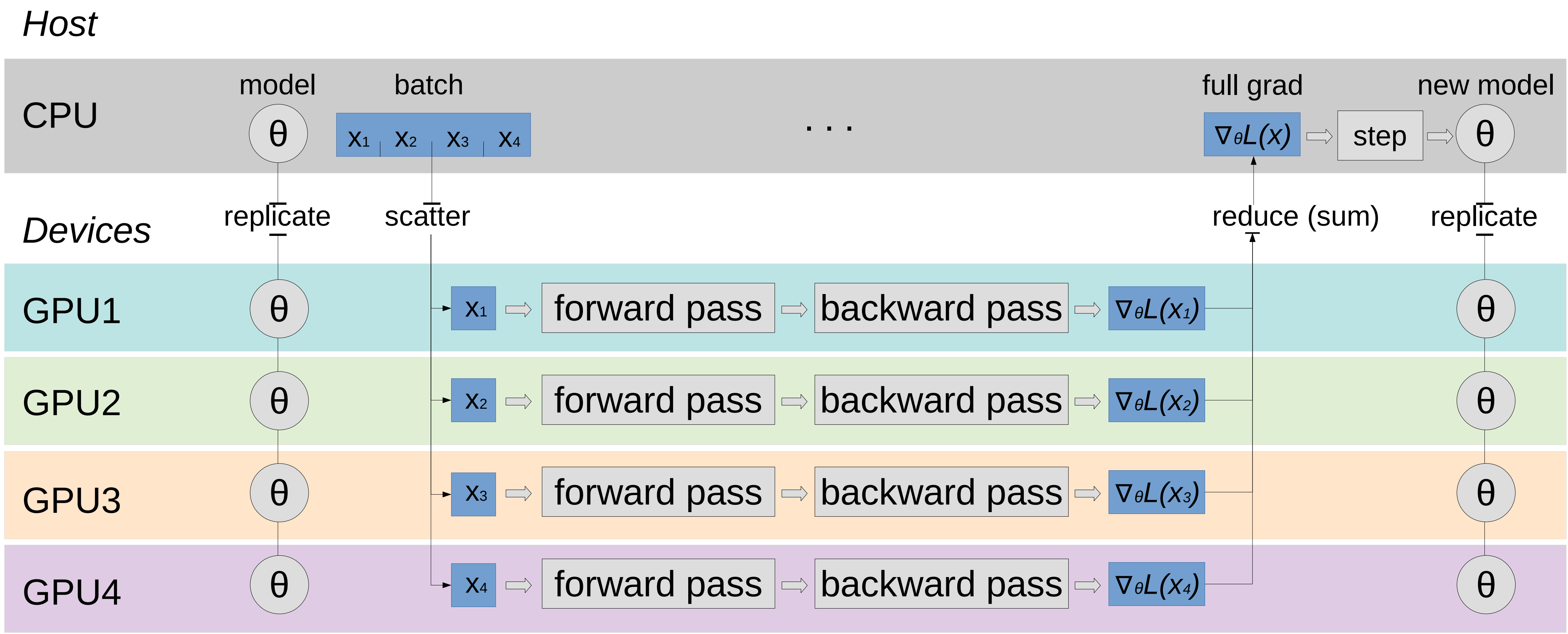
Data-parallel training (naive)

cs.cmu.edu/~muli/file/parameter_server_osdi14.pdf



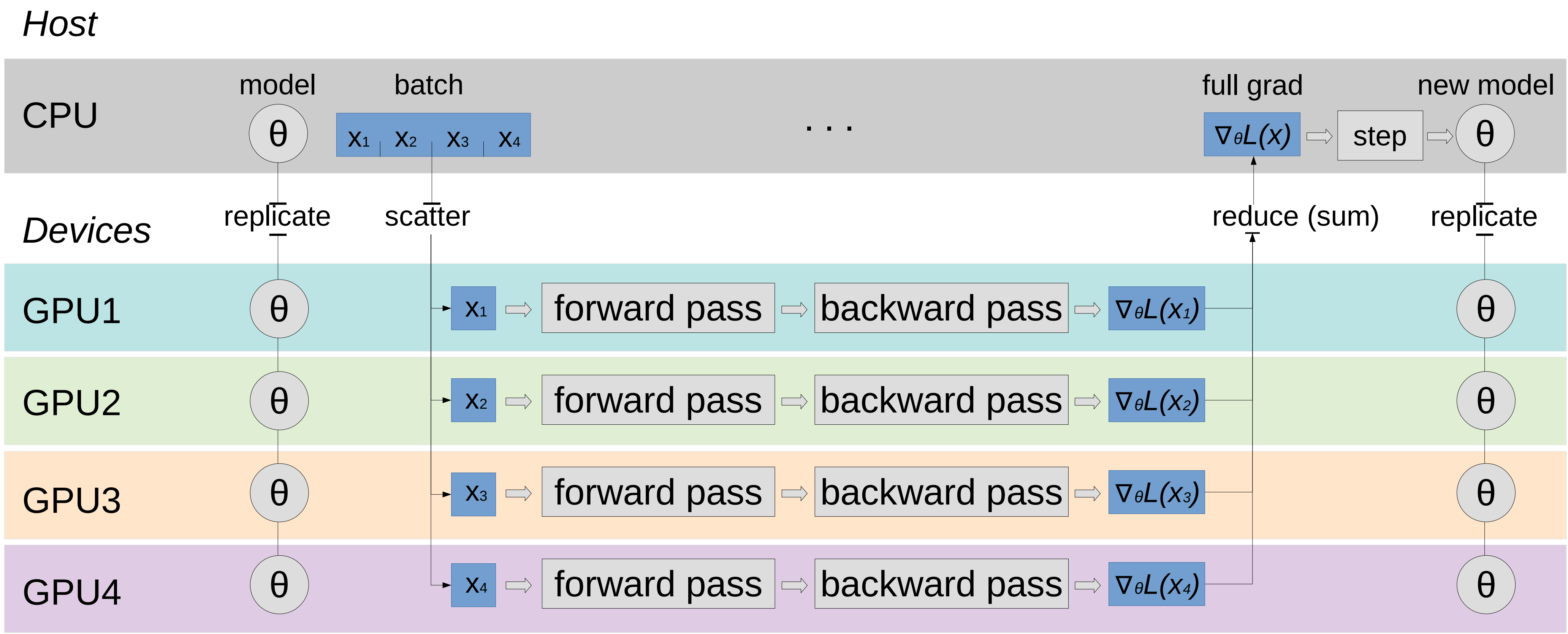
Data-parallel training (naive)

cs.cmu.edu/~muli/file/parameter_server_osdi14.pdf



Data-parallel training (naive)

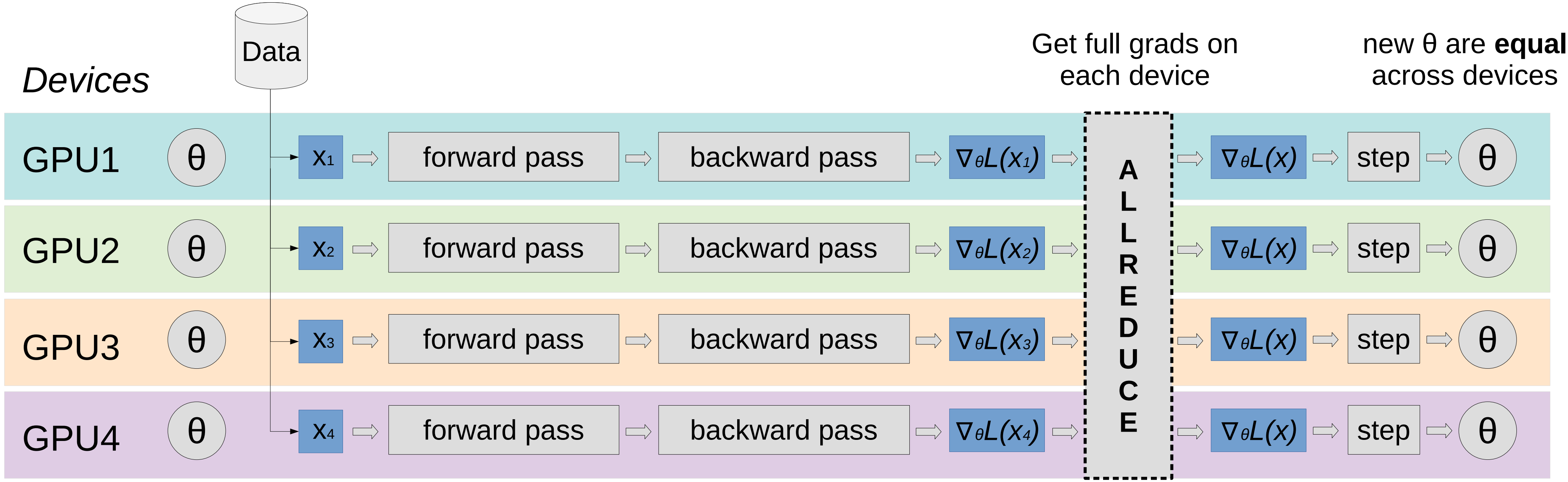
cs.cmu.edu/~muli/file/parameter_server_osdi14.pdf



All-Reduce data parallel

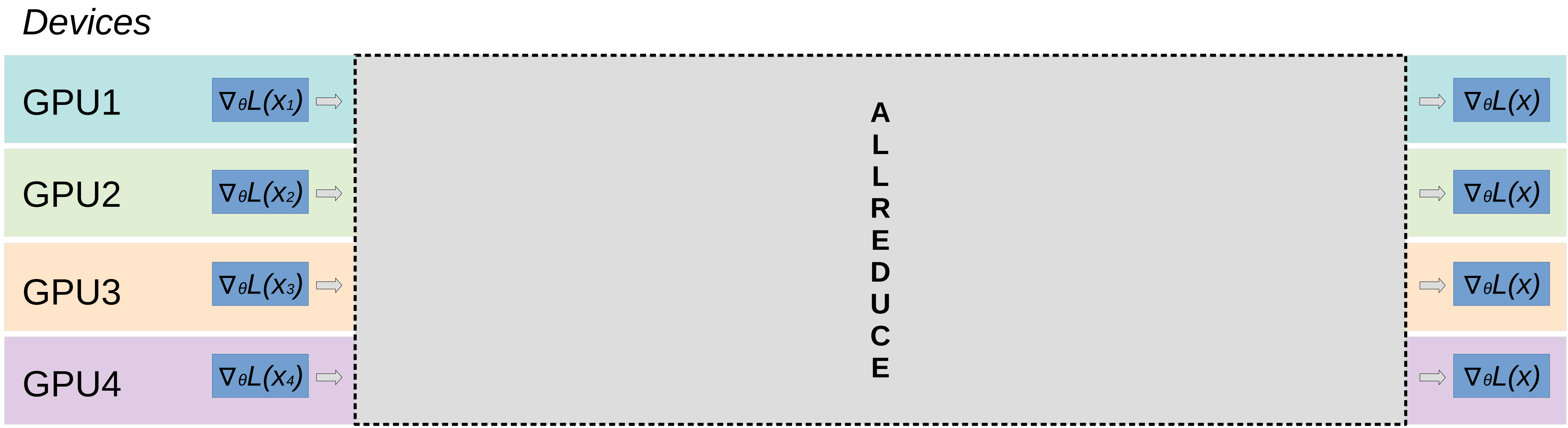
arxiv.org/abs/1706.02677

Idea: get rid of the host, each gpu runs its own computation
Q: why will weights be equal after such step?



Faster allreduce

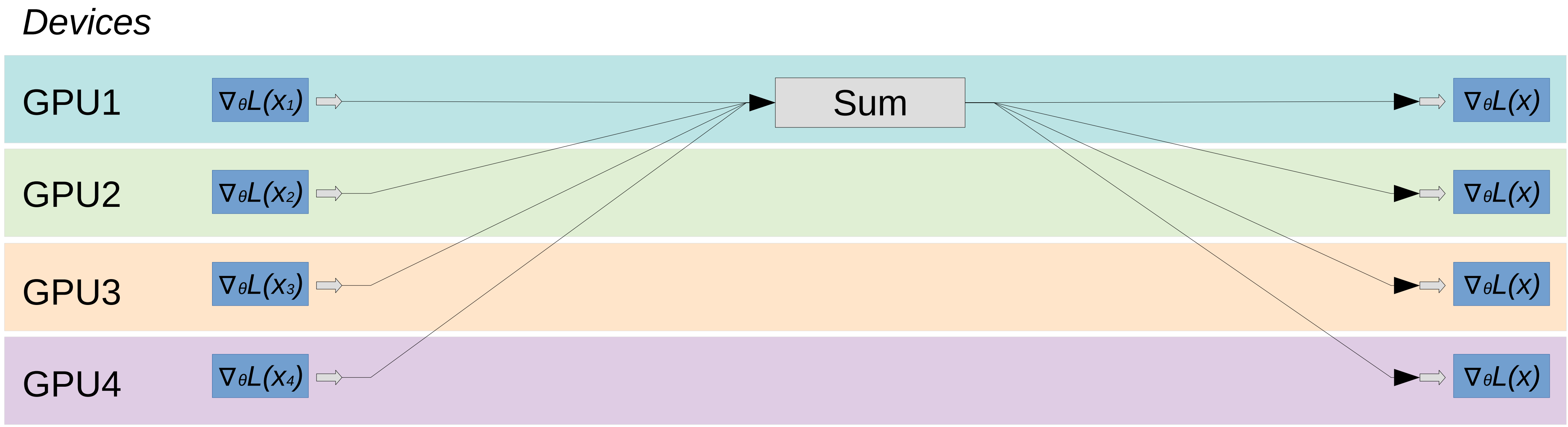
Input: each device has its its own vector
Output: each device gets a sum of all vectors



Faster allreduce

Input: each device has its its own vector
Output: each device gets a sum of all vectors

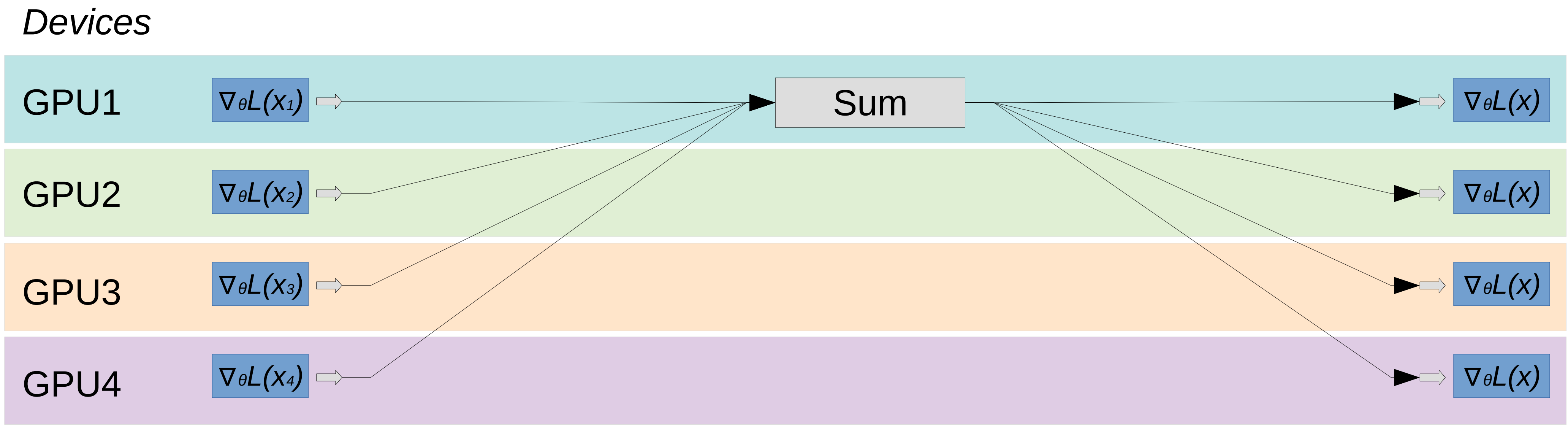
Naive implementation



Faster allreduce

Input: each device has its its own vector
Output: each device gets a sum of all vectors

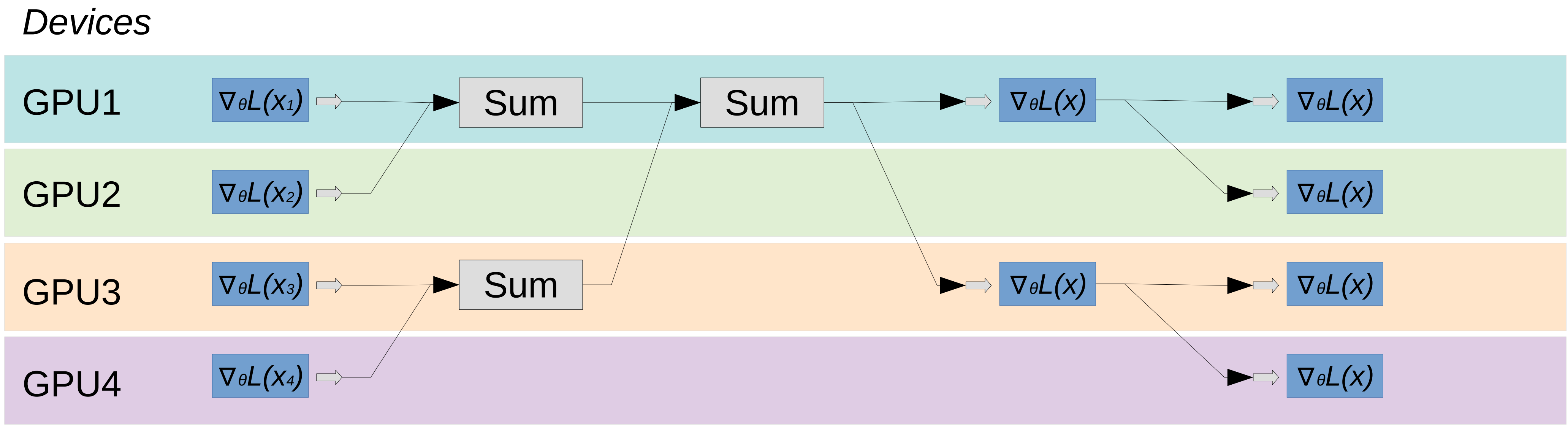
Q: Can we do better?



Faster allreduce

Input: each device has its its own vector
Output: each device gets a sum of all vectors

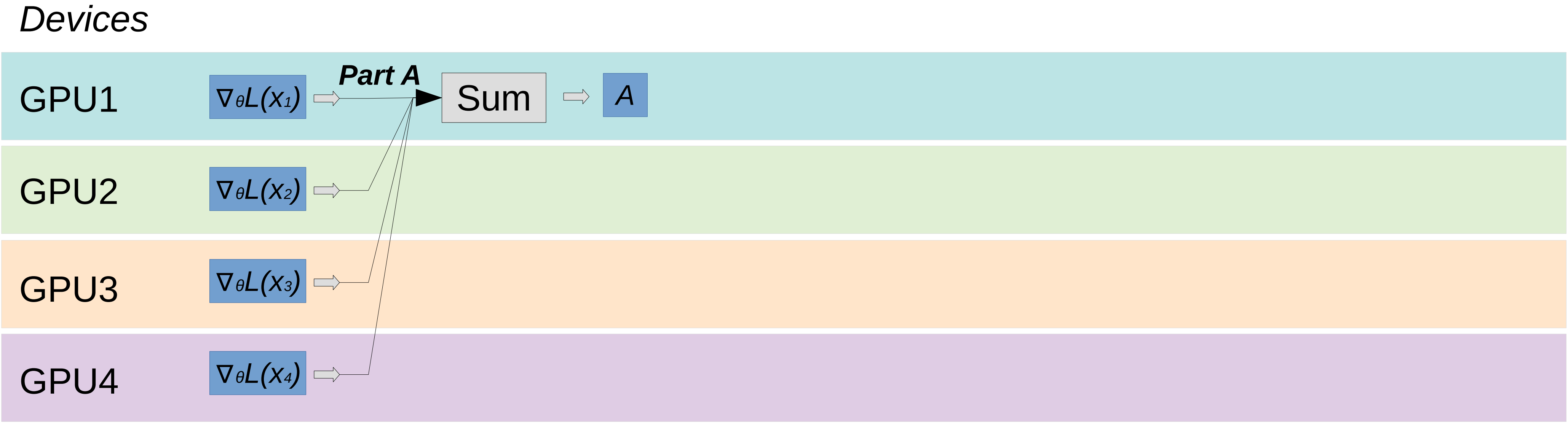
Tree-allreduce



Faster allreduce

Input: each device has its its own vector
Output: each device gets a sum of all vectors

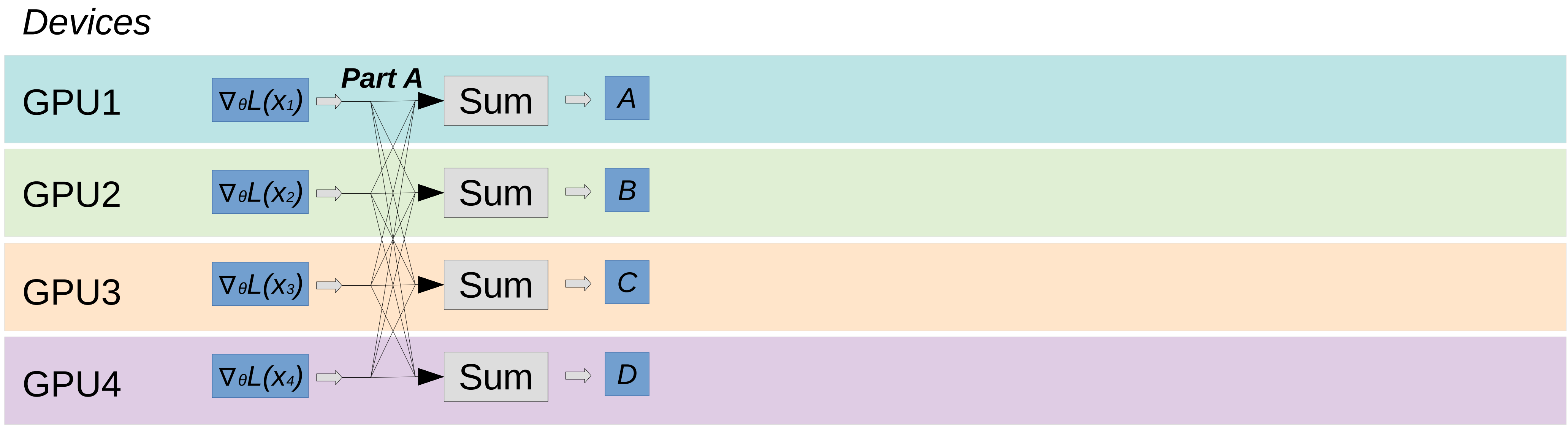
Butterfly-allreduce – split data into chunks (ABCD)



Faster allreduce

Input: each device has its its own vector
Output: each device gets a sum of all vectors

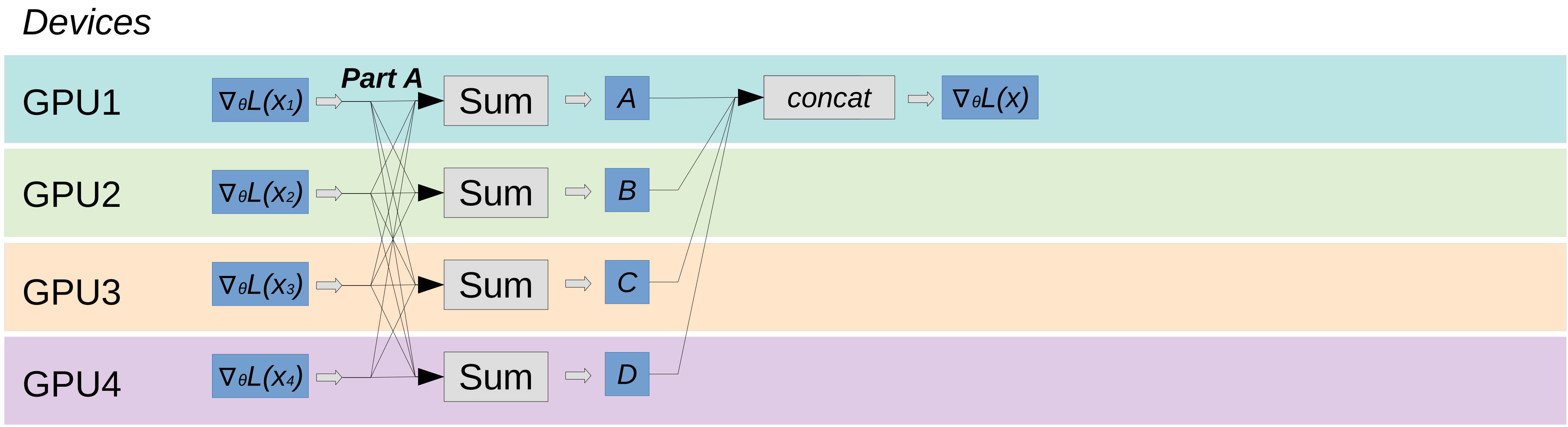
Butterfly-allreduce – split data into chunks (ABCD)



Faster allreduce

Input: each device has its its own vector
Output: each device gets a sum of all vectors

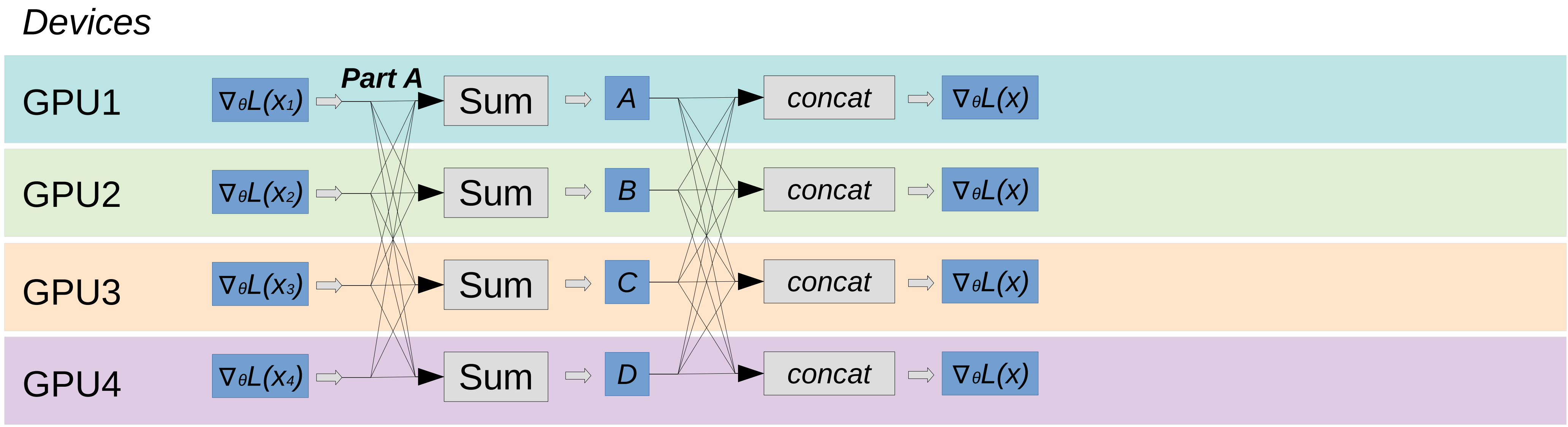
Butterfly-allreduce – split data into chunks (ABCD)



Faster allreduce

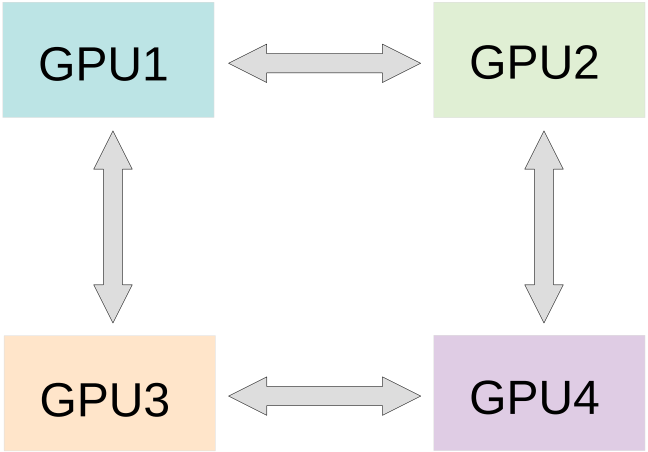
Input: each device has its its own vector
Output: each device gets a sum of all vectors

Ring-allreduce – split data into chunks (ABCD)



Ring allreduce

Bonus quest: you can only send data between **adjacent** gpus



Ring topology

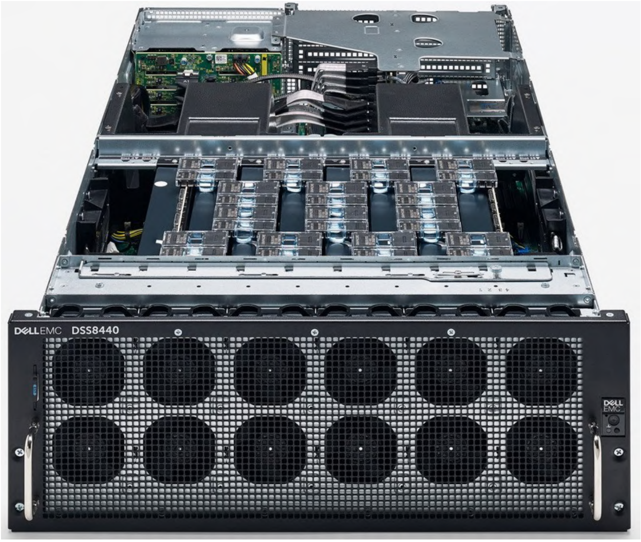


Image: graphcore IPU server

Answer & more: tinyurl.com/ring-allreduce-blog

Ring allreduce

Bonus quest: you can only send data between **adjacent** gpus

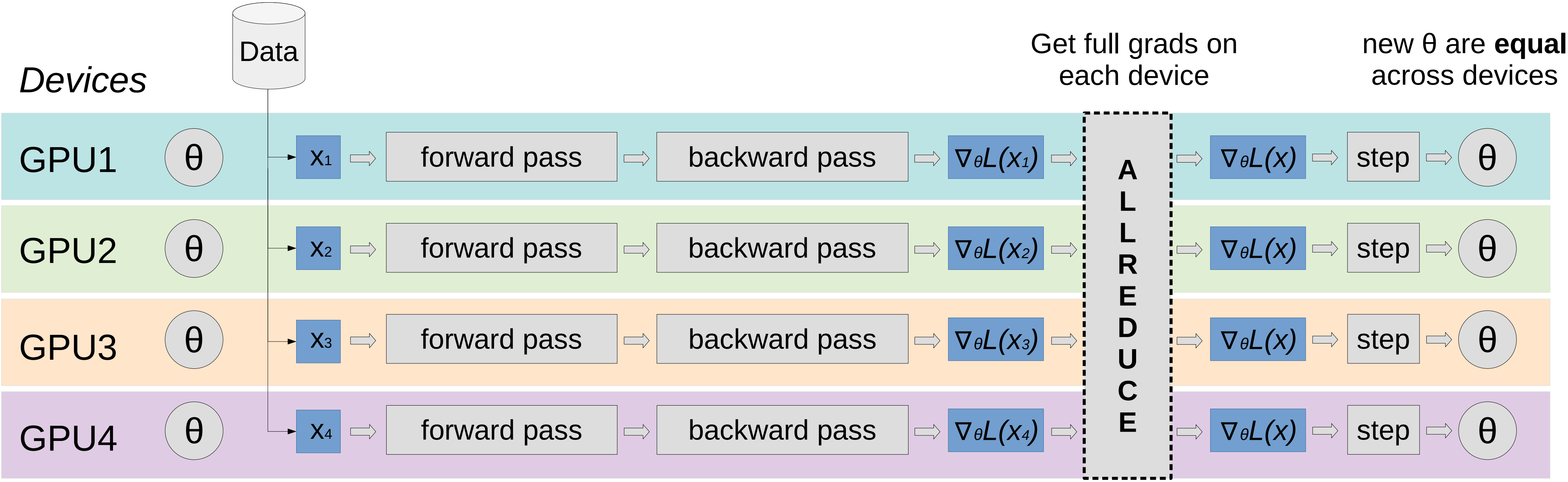
[Time to use the whiteboard]

Answer & more: tinyurl.com/ring-allreduce-blog

All-Reduce data parallel

arxiv.org/abs/1706.02677

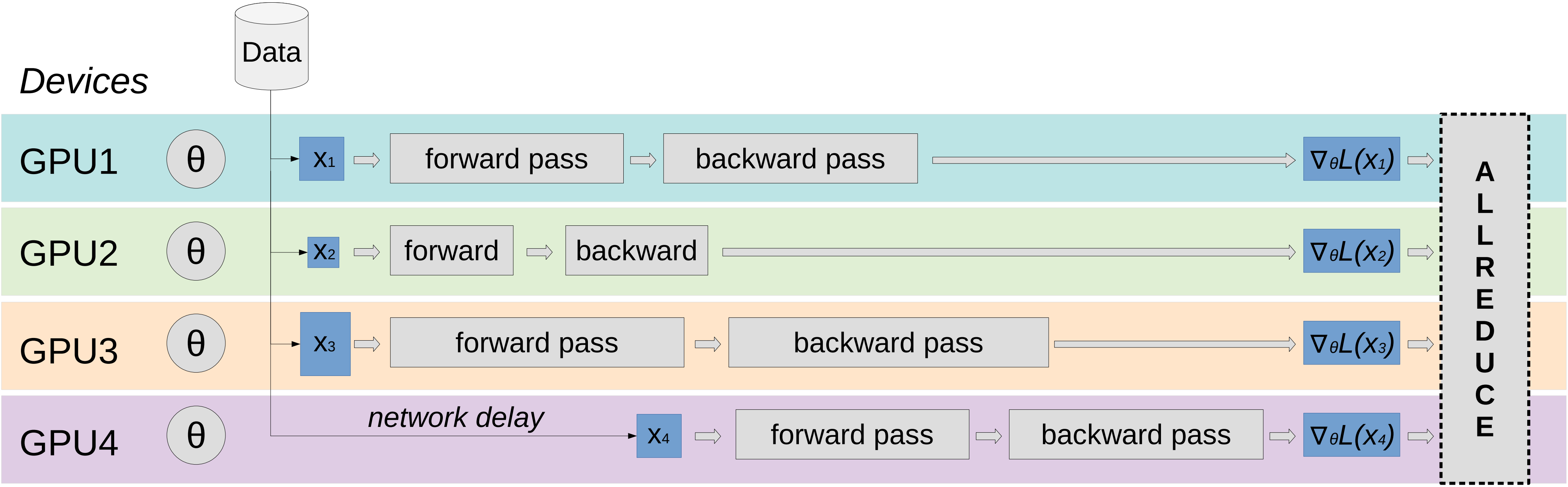
Idea: get rid of the host, each gpu runs its own computation
Q: why will weights be equal after such step?



All-Reduce data parallel VS reality

arxiv.org/abs/1706.02677

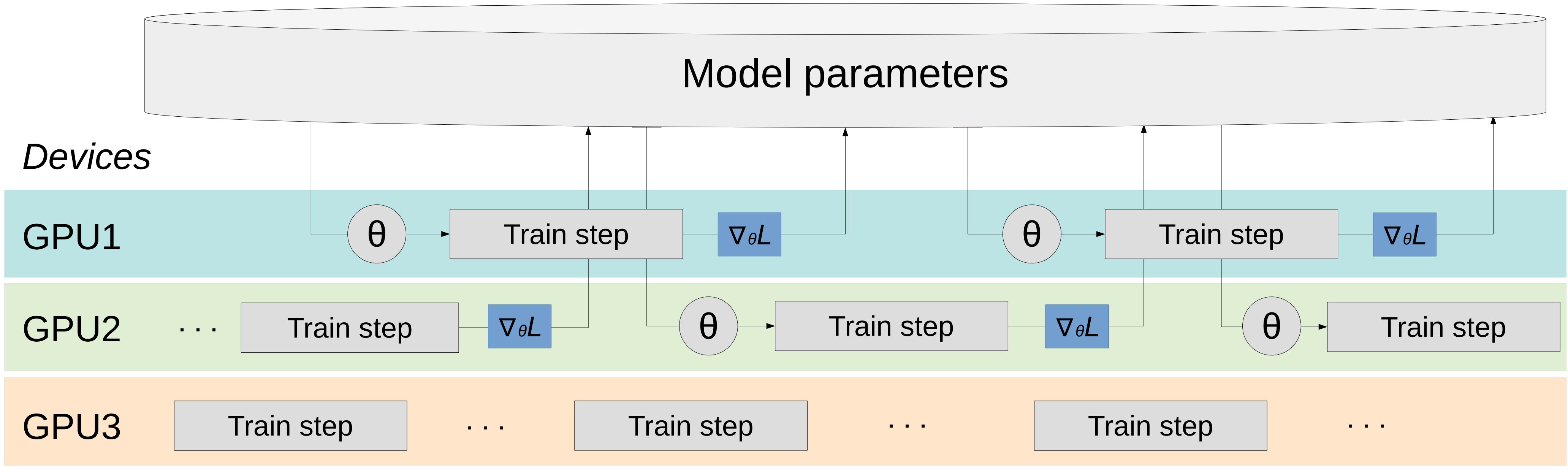
Each gpu has different processing time & delays
Q: can we improve device utilization?



Recap: Parameter Server

HOGWILD! arxiv.org/abs/1106.5730

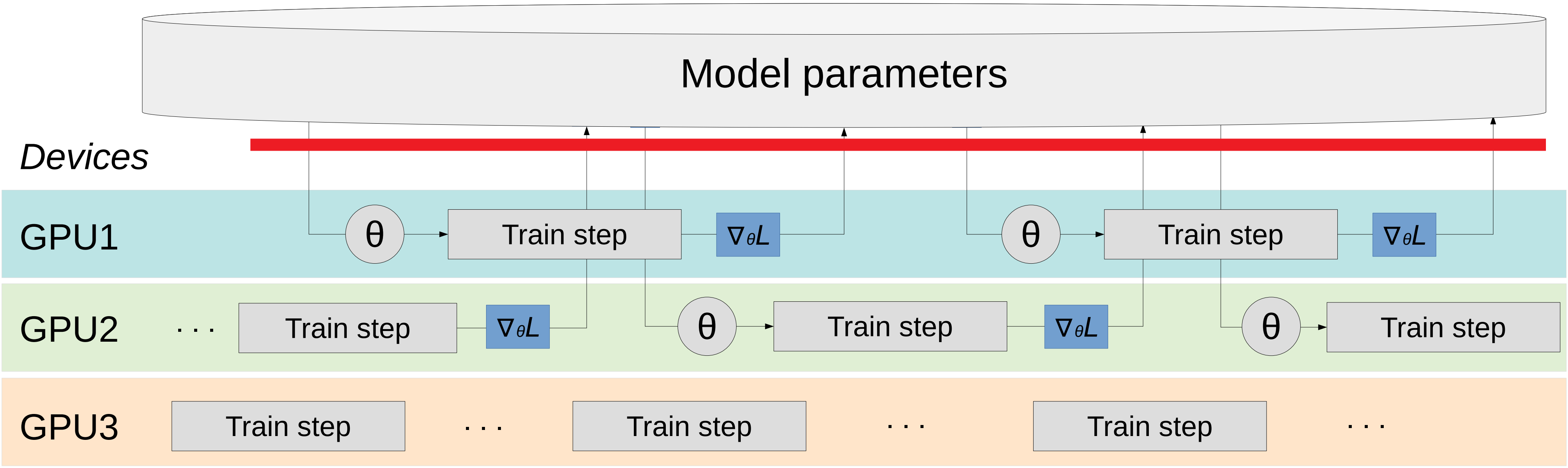
Idea: remove synchronization step alltogether, use parameter server



Recap: Parameter Server

HOGWILD! arxiv.org/abs/1106.5730

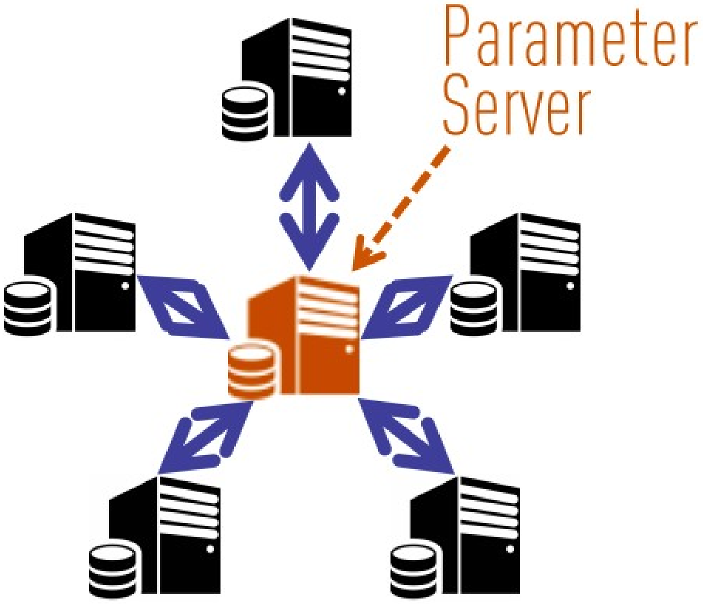
Idea: remove synchronization step alltogether, use parameter server



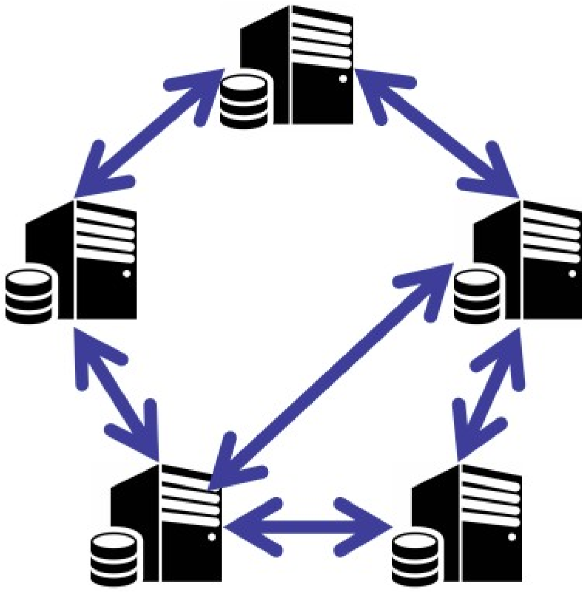
Problem: parameter servers need to ingest tons of data over training

Decentralized Training with Gossip

Gossip (communication): https://tinyurl.com/boyd-gossip-2006
Gossip outperforms All-Reduce: https://tinyurl.com/can-dsgd-outperform



(a) Centralized Topology



(b) Decentralized Topology

Decentralized Training with Gossip

Source: https://tinyurl.com/can-dsgd-outperform

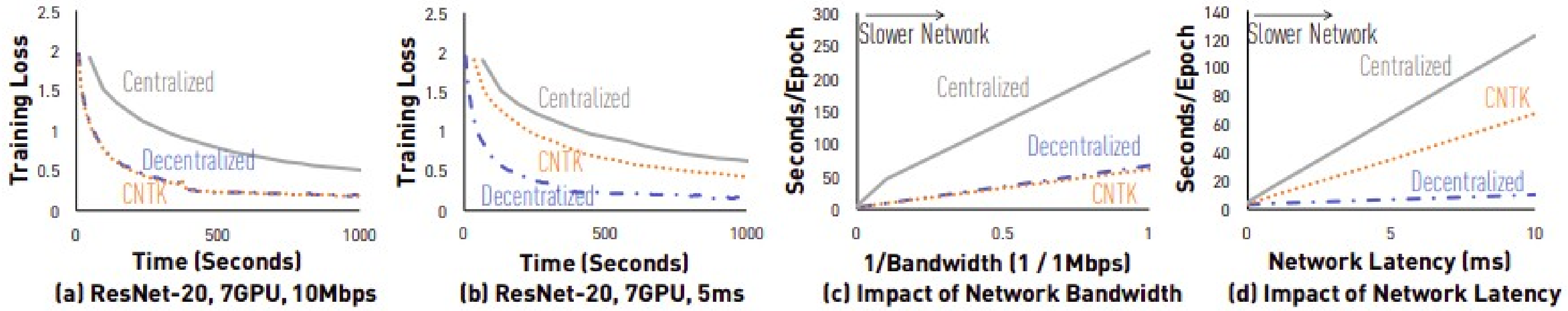


Figure 2: Comparison between D-PSGD and two centralized implementations (7 and 10 GPUs).

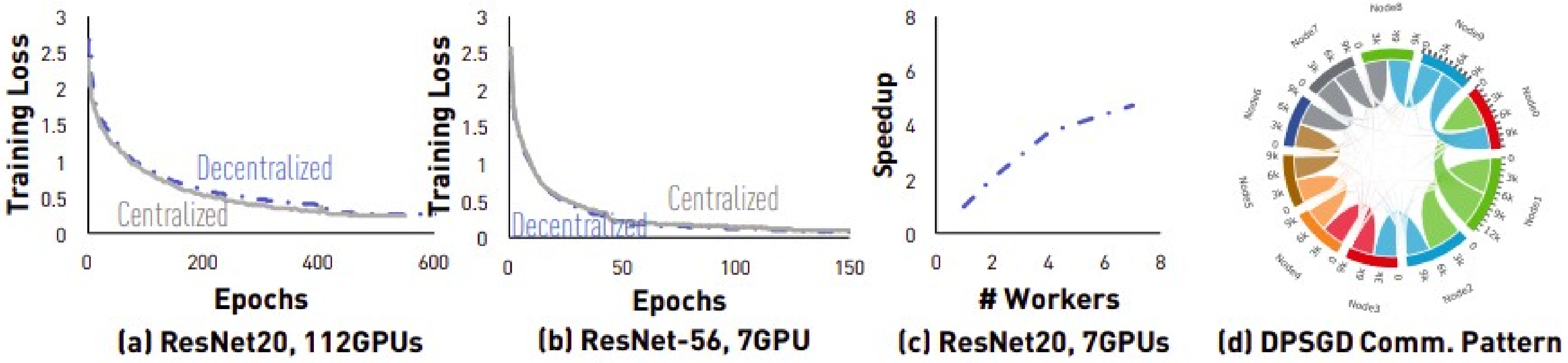
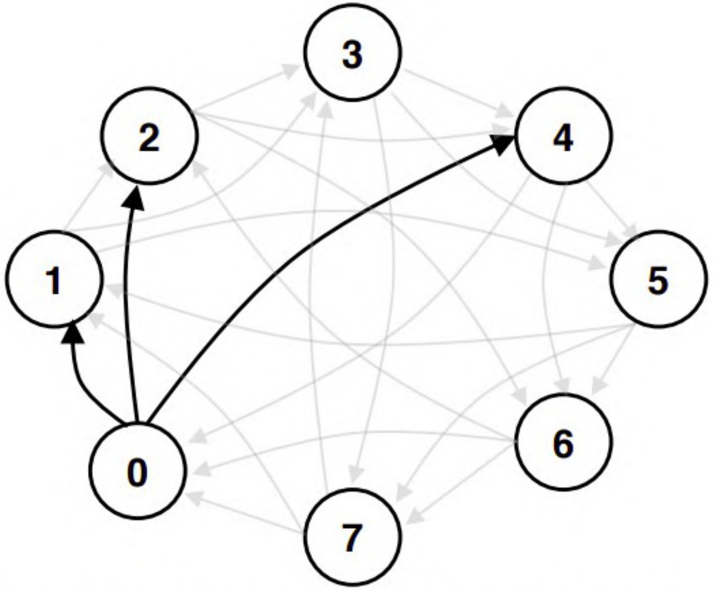


Figure 3: (a) Convergence Rate; (b) D-PSGD Speedup; (c) D-PSGD Communication Patterns.

Stochastic Gradient Push

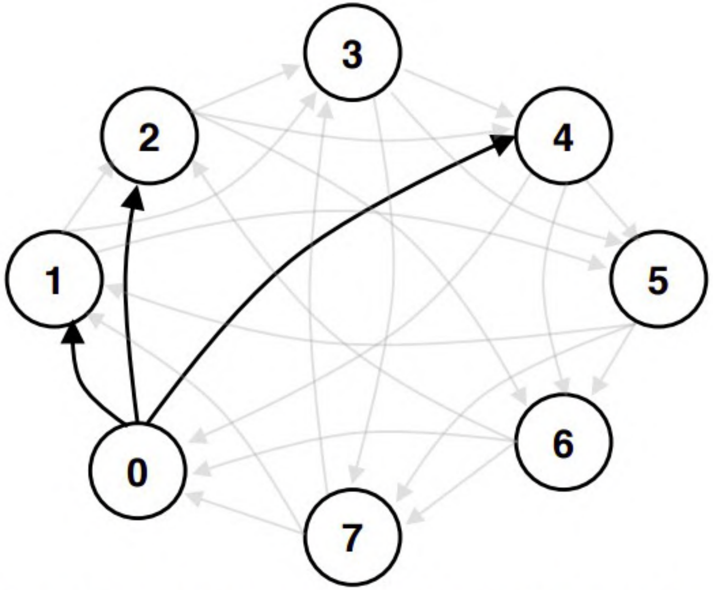
Source: https://arxiv.org/abs/1811.10792



(a) Directed Exponential Graph highlighting node 0's out-neighbours

Stochastic Gradient Push

Source: https://arxiv.org/abs/1811.10792



(a) Directed Exponential Graph highlighting node 0's out-neighbours

Algorithm 1 Stochastic Gradient Push (SGP)

Require: Initialize $\gamma > 0$, $\boldsymbol{x}_i^{(0)} = \boldsymbol{z}_i^{(0)} \in \mathbb{R}^d$ and $w_i^{(0)} = 1$ for all nodes $i \in \{1, 2, \ldots, n\}$

1: **for** $k = 0, 1, 2, \cdots, K$, at node i, **do**

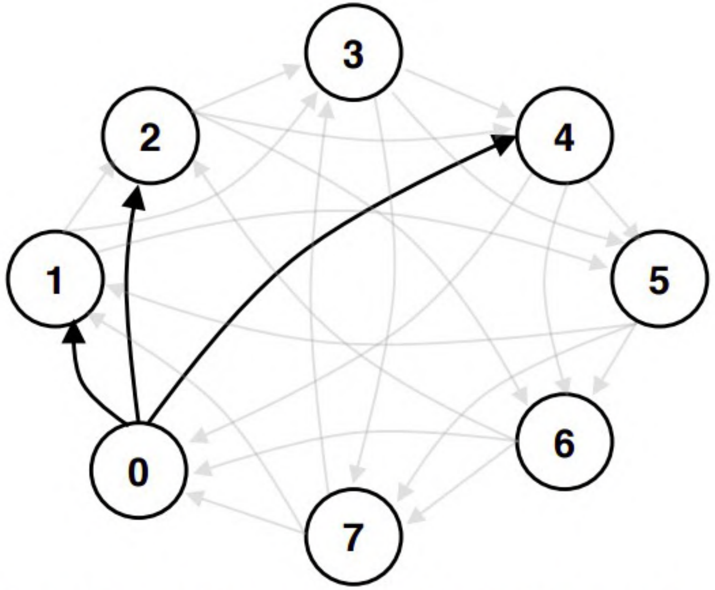
2: Sample new mini-batch $\xi_i^{(k)} \sim \mathcal{D}_i$ from local distribution

3: Compute mini-batch gradient at $\boldsymbol{z}_i^{(k)}$: $\nabla \boldsymbol{F}_i(\boldsymbol{z}_i^{(k)}; \xi_i^{(k)})$

<to be continued>

Stochastic Gradient Push

Source: https://arxiv.org/abs/1811.10792



(a) Directed Exponential Graph highlighting node 0's out-neighbours

---

Algorithm 1 Stochastic Gradient Push (SGP)

Require: Initialize $\gamma > 0$, $\boldsymbol{x}_i^{(0)} = \boldsymbol{z}_i^{(0)} \in \mathbb{R}^d$ and $w_i^{(0)} = 1$ for all nodes $i \in \{1, 2, \ldots, n\}$

1: **for** $k = 0, 1, 2, \cdots, K$, at node i, **do**

2: Sample new mini-batch $\xi_i^{(k)} \sim \mathcal{D}_i$ from local distribution

3: Compute mini-batch gradient at $\boldsymbol{z}_i^{(k)}$: $\nabla \boldsymbol{F}_i(\boldsymbol{z}_i^{(k)}; \xi_i^{(k)})$

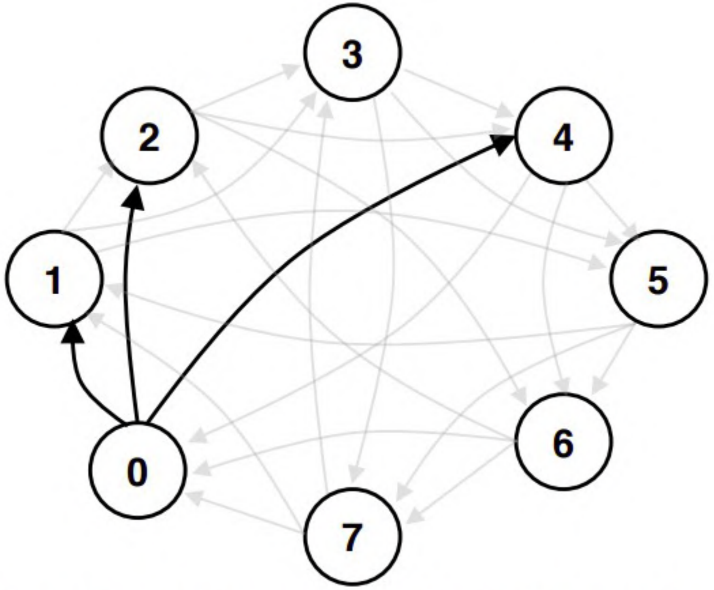
4: $\boldsymbol{x}_i^{(k+\frac{1}{2})} = \boldsymbol{x}_i^{(k)} - \gamma \nabla \boldsymbol{F}_i(\boldsymbol{z}_i^{(k)}; \xi_i^{(k)})$

normal GD step

<to be continued>

Stochastic Gradient Push

Source: https://arxiv.org/abs/1811.10792



(a) Directed Exponential Graph highlighting node 0's out-neighbours

---

Algorithm 1 Stochastic Gradient Push (SGP)

Require: Initialize $\gamma > 0$, $\boldsymbol{x}_i^{(0)} = \boldsymbol{z}_i^{(0)} \in \mathbb{R}^d$ and $w_i^{(0)} = 1$ for all nodes $i \in \{1, 2, \ldots, n\}$

1: **for** $k = 0, 1, 2, \cdots, K$, at node i, **do**

2: Sample new mini-batch $\xi_i^{(k)} \sim \mathcal{D}_i$ from local distribution

3: Compute mini-batch gradient at $\boldsymbol{z}_i^{(k)}$: $\nabla \boldsymbol{F}_i(\boldsymbol{z}_i^{(k)}; \xi_i^{(k)})$

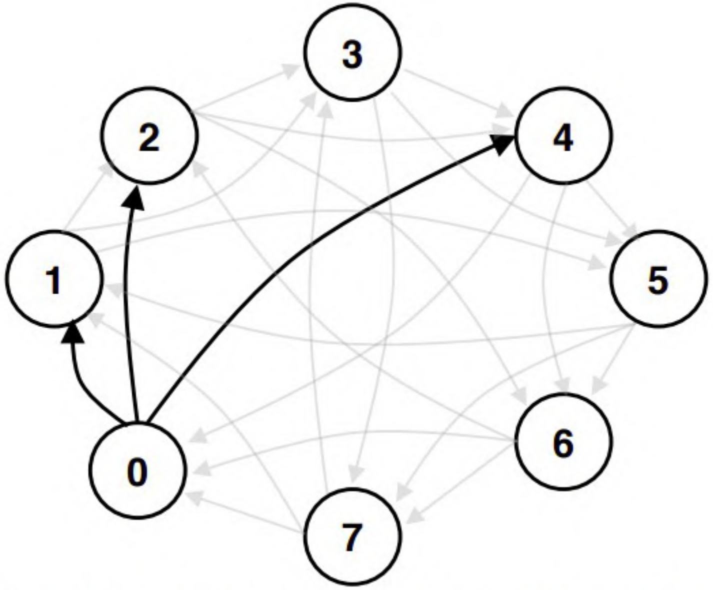
4: $\boldsymbol{x}_i^{(k+\frac{1}{2})} = \boldsymbol{x}_i^{(k)} - \gamma \nabla \boldsymbol{F}_i(\boldsymbol{z}_i^{(k)}; \xi_i^{(k)})$

5: Send $\left(p_{j,i}^{(k)} \boldsymbol{x}_i^{(k+\frac{1}{2})}, p_{j,i}^{(k)} w_i^{(k)}\right)$ to out-neighbors;
receive $\left(p_{i,j}^{(k)} \boldsymbol{x}_j^{(k+\frac{1}{2})}, p_{i,j}^{(k)} w_j^{(k)}\right)$ from in-neighbors

<to be continued>

Stochastic Gradient Push

Source: https://arxiv.org/abs/1811.10792



(a) Directed Exponential Graph highlighting node 0's out-neighbours

Algorithm 1 Stochastic Gradient Push (SGP)

Require: Initialize $\gamma > 0$, $\boldsymbol{x}_i^{(0)} = \boldsymbol{z}_i^{(0)} \in \mathbb{R}^d$ and $w_i^{(0)} = 1$ for all nodes $i \in \{1, 2, \ldots, n\}$

for $k = 0, 1, 2, \cdots, K$, at node i, **do**

  Sample new mini-batch $\xi_i^{(k)} \sim \mathcal{D}_i$ from local distribution

  Compute mini-batch gradient at $\boldsymbol{z}_i^{(k)}$: $\nabla \boldsymbol{F}_i(\boldsymbol{z}_i^{(k)}; \xi_i^{(k)})$

  $\boldsymbol{x}_i^{(k+\frac{1}{2})} = \boldsymbol{x}_i^{(k)} - \gamma \nabla \boldsymbol{F}_i(\boldsymbol{z}_i^{(k)}; \xi_i^{(k)})$

  Send $\left(p_{j,i}^{(k)} \boldsymbol{x}_i^{(k+\frac{1}{2})}, p_{j,i}^{(k)} w_i^{(k)}\right)$ to out-neighbors;

  receive $\left(p_{i,j}^{(k)} \boldsymbol{x}_j^{(k+\frac{1}{2})}, p_{i,j}^{(k)} w_j^{(k)}\right)$ from in-neighbors

  $\boldsymbol{x}_i^{(k+1)} = \sum_j p_{i,j}^{(k)} \boldsymbol{x}_j^{(k+\frac{1}{2})}$

  $w_i^{(k+1)} = \sum_j p_{i,j}^{(k)} w_j^{(k)}$

  $\boldsymbol{z}_i^{(k+1)} = \boldsymbol{x}_i^{(k+1)} / w_i^{(k+1)}$

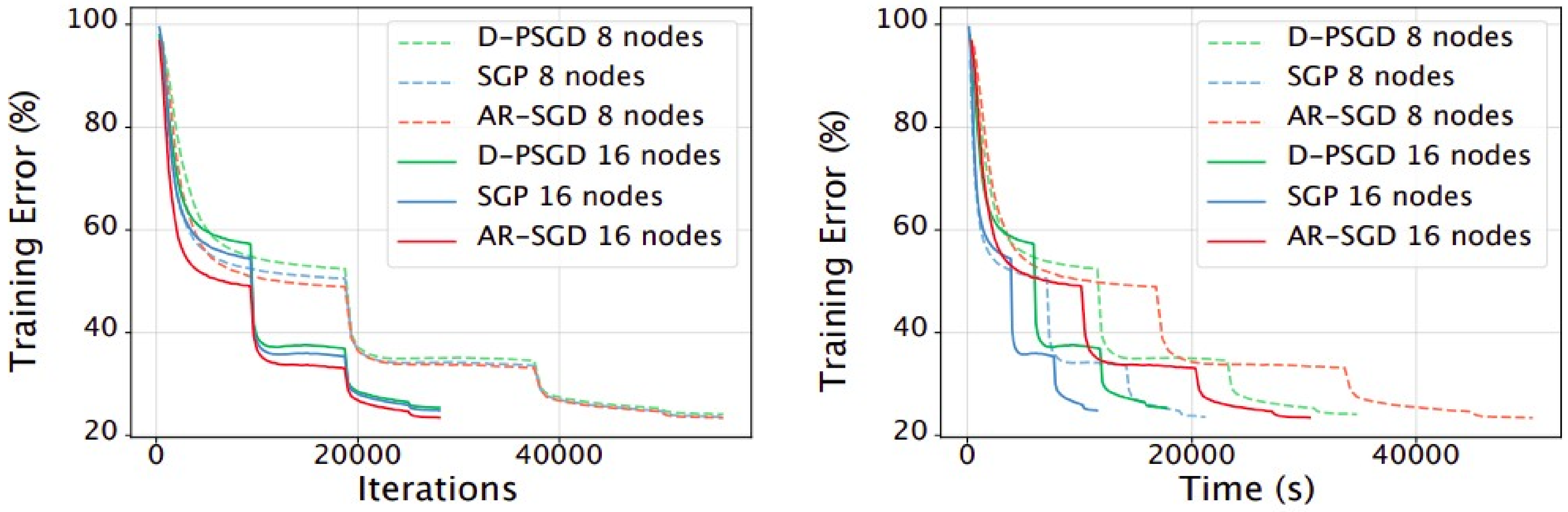
end for

weighted
average

Stochastic Gradient Push

Source: https://arxiv.org/abs/1811.10792

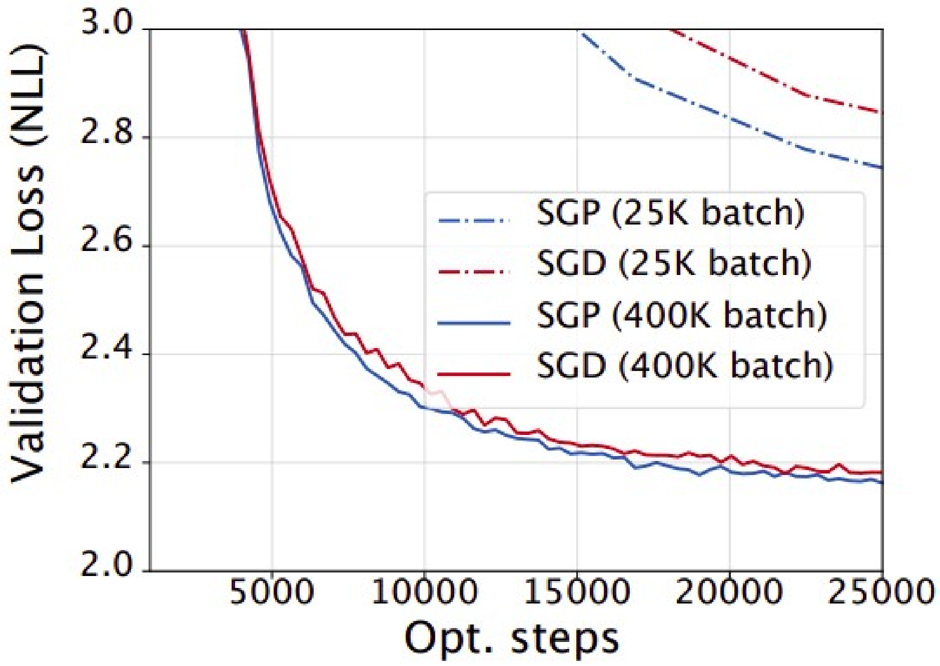
SGP vs ImageNet (ResNet50 + SGD w/ momentum)

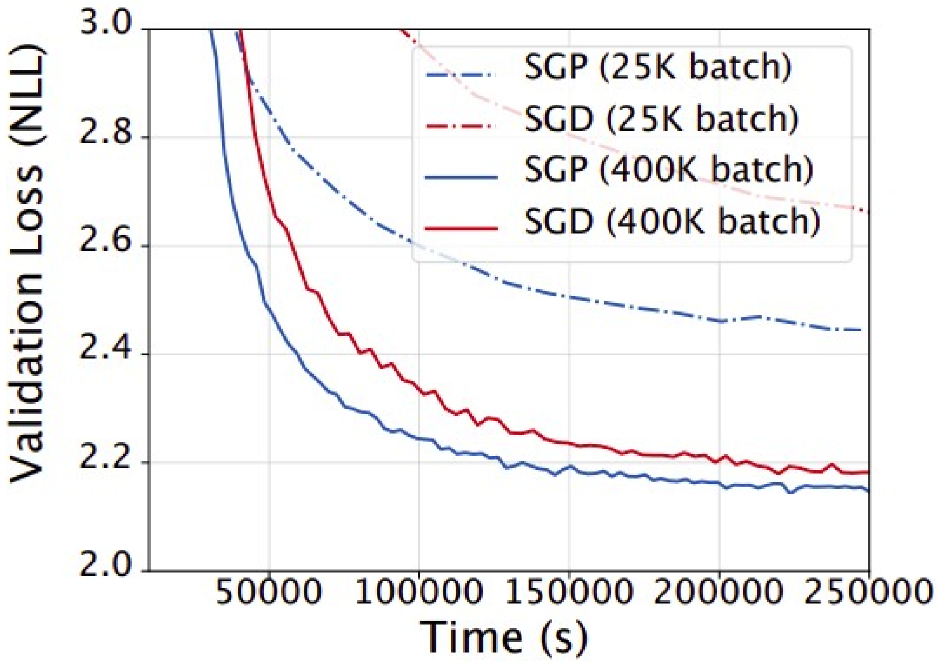


Stochastic Gradient Push

Source: https://arxiv.org/abs/1811.10792

SGP vs WMT English-German (Transformer, Adam)





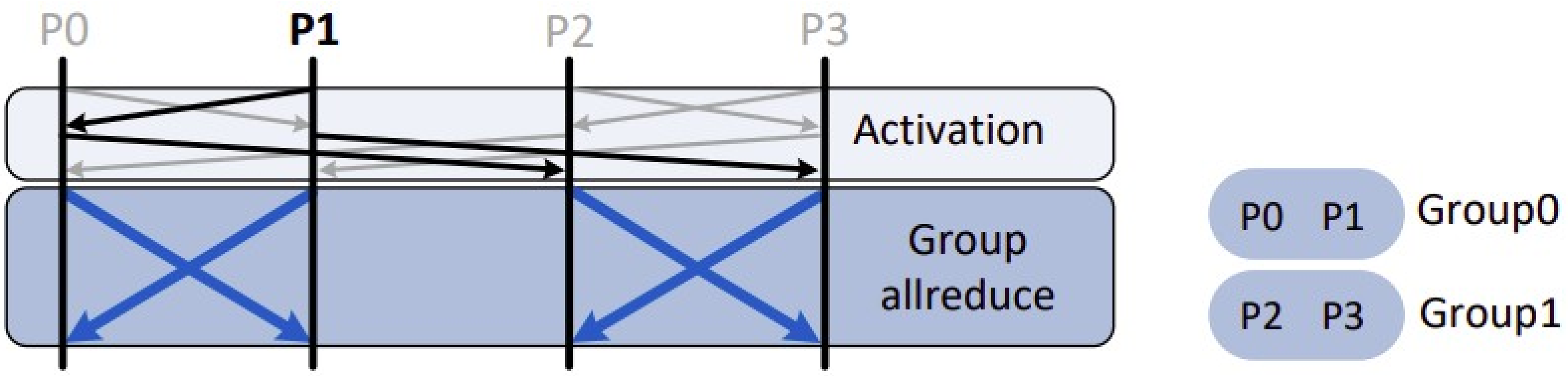
Gossip vs All-Reduce

Your thoughts?

Gossip + All-Reduce

Source: arxiv.org/abs/2005.00124

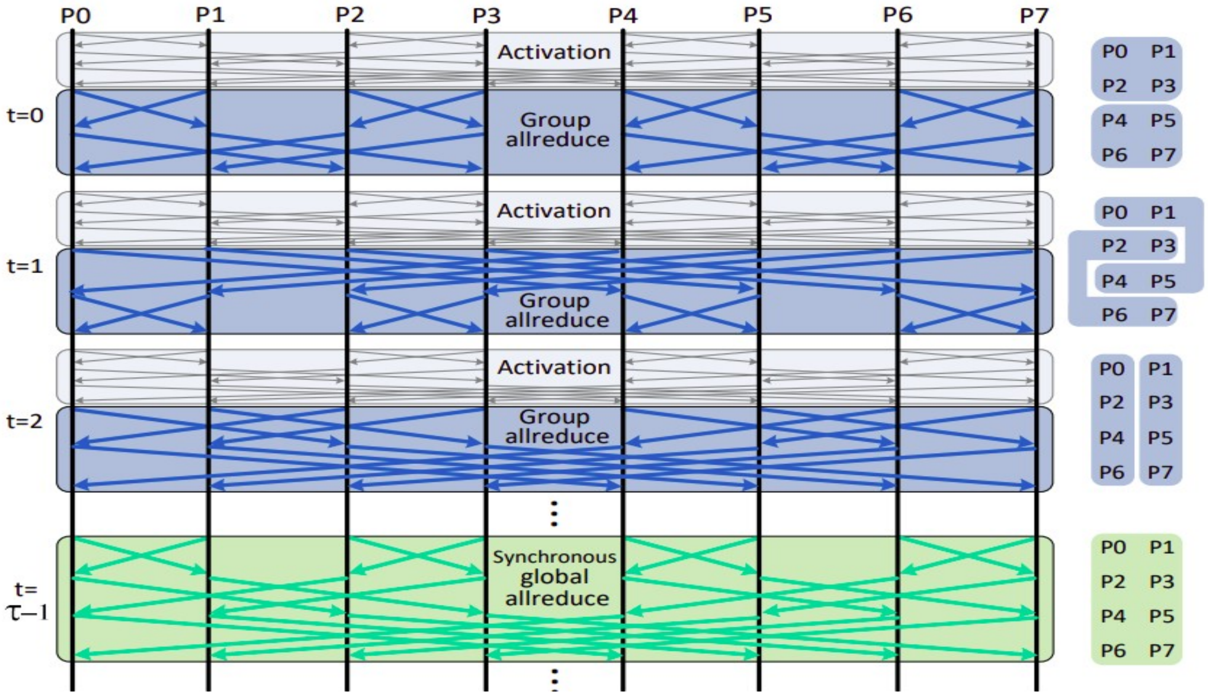
Core idea: run all-reduce in independent groups
You only have to synchronize for your small group
Swap groupmates between iterations



Gossip + All-Reduce

Source: arxiv.org/abs/2005.00124

Core idea: run all-reduce in independent groups
You only have to synchronize for your small group
Swap groupmates between iterations

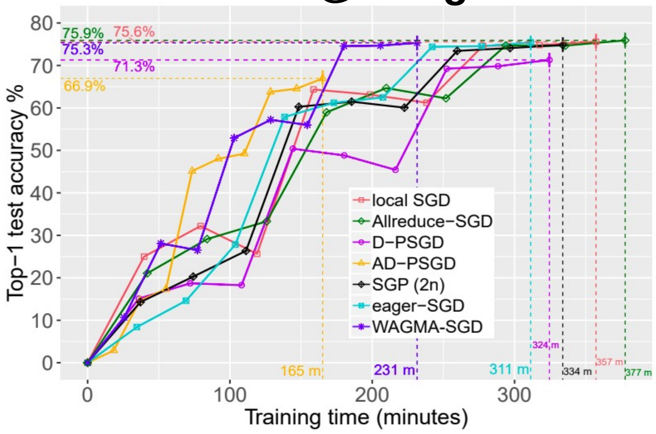


Gossip + All-Reduce

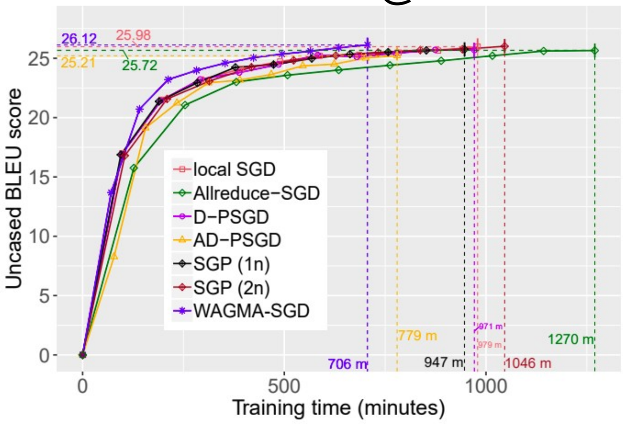
Source: arxiv.org/abs/2005.00124

Experiment setup: up to 1024 GPU,
Natural (or emulated) network latency

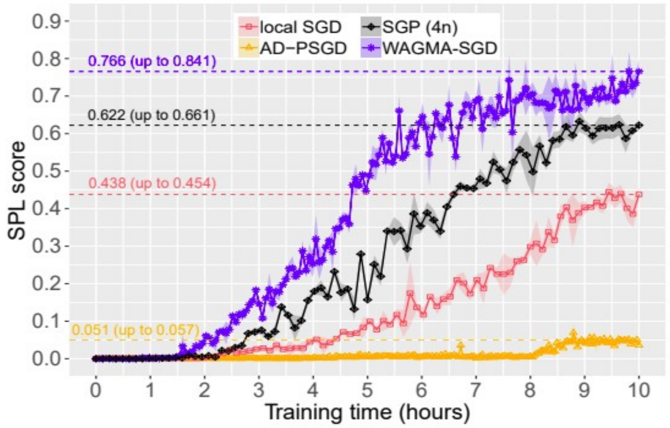
Image Classification
ResNet50 @ ImageNet



Machine Translation
Transformer @ WMT17



Reinforcement Learning
DDPO on Habitat



</Data-parallel>

+ easy to implement
+ can scale to 100s of gpus
+ can be fault-tolerant
- model must fit in 1 gpu
- large batches aren't always good for generalization

- 2-4 GPUs & no time – naive data parallel tinyurl.com/torch-data-parallel
- 4+ GPUs or multiple hosts – distributed (allreduce) github.com/horovod/horovod
  - Intro to pytorch distributed: tinyurl.com/distributed-dp **or in 15 minutes!**
- Somewhat faulty GPU/network: synchronous data parallel + drop stragglers
- Very faulty or uneven resources: asynchronous data parallel (more later)
- Efficient training with large batches: LAMB https://arxiv.org/abs/1904.00962
- Dynamically adding or removing resources: https://tinyurl.com/torch-elastic

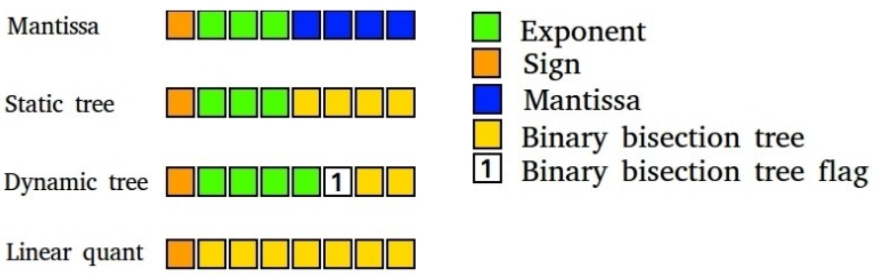
Q: what if sending tensors during AllReduce takes too long?

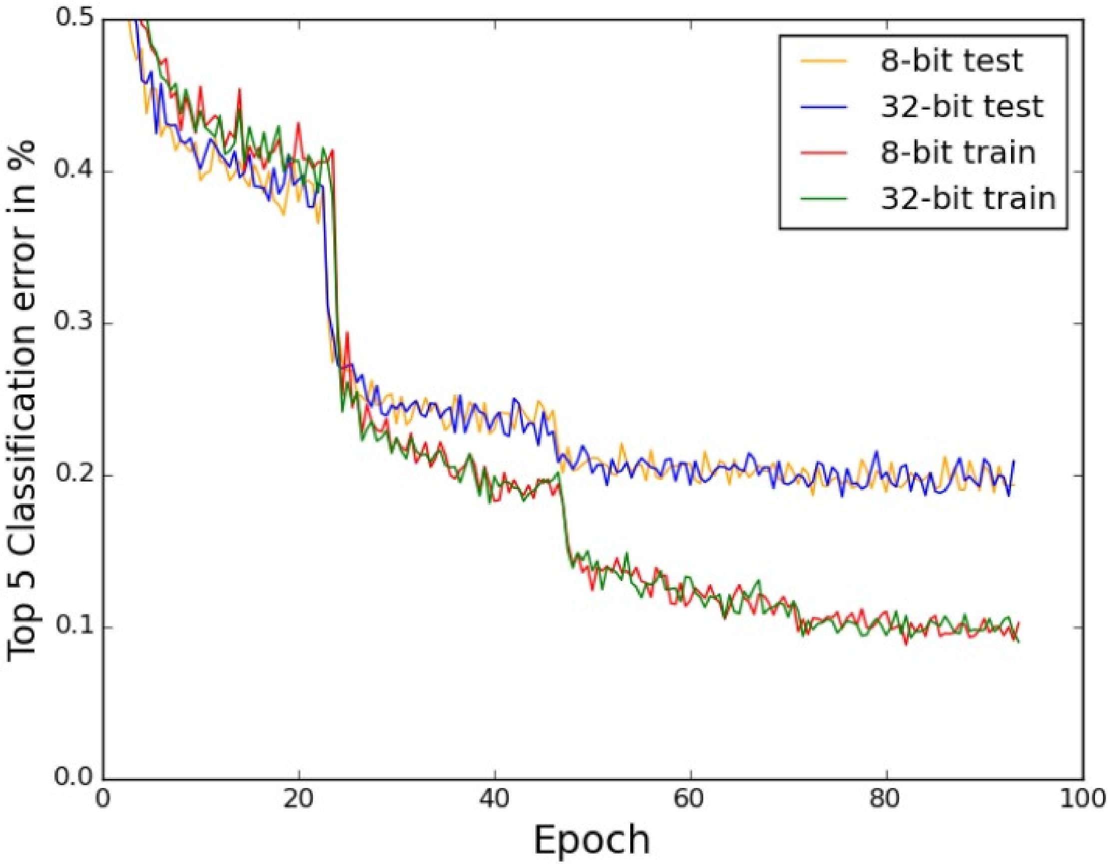
Quantized communication

https://arxiv.org/abs/1511.04561

TL;DR
- send data in 8-bit
- all computations in 32-bit
- choose best data format

PROFIT: same quality as float16





Can we compress further?
without losing quality

Error Feedback + PowerSGD

https://arxiv.org/abs/1901.09847 - EF theory
https://arxiv.org/abs/1905.13727 - PowerSGD

TL;DR - use extreme compression, e.g. 1-bit or top-5% gradients

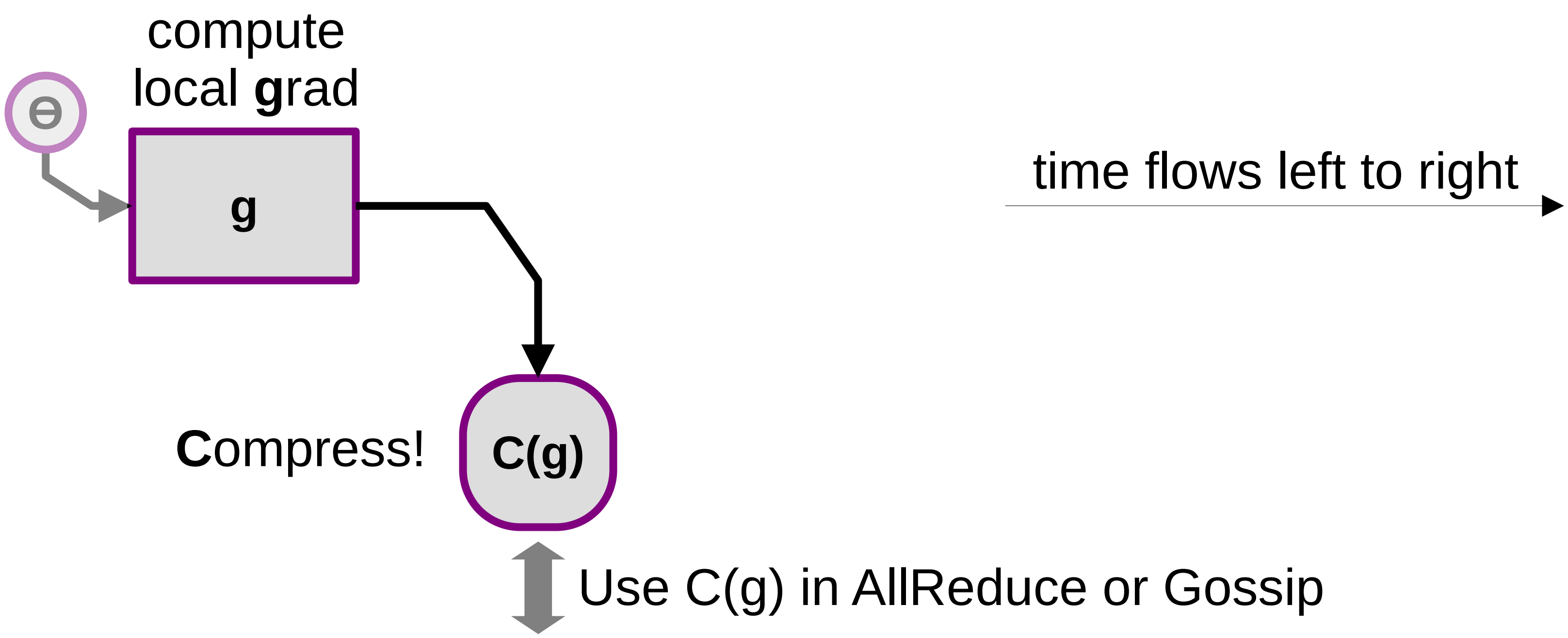
- if you lose something in compression, **reuse it on the next step**

Error Feedback + PowerSGD

https://arxiv.org/abs/1901.09847 - EF theory
https://arxiv.org/abs/1905.13727 - PowerSGD

TL;DR - use extreme compression, e.g. 1-bit or top-5% gradients

- if you lose something in compression, **reuse it on the next step**

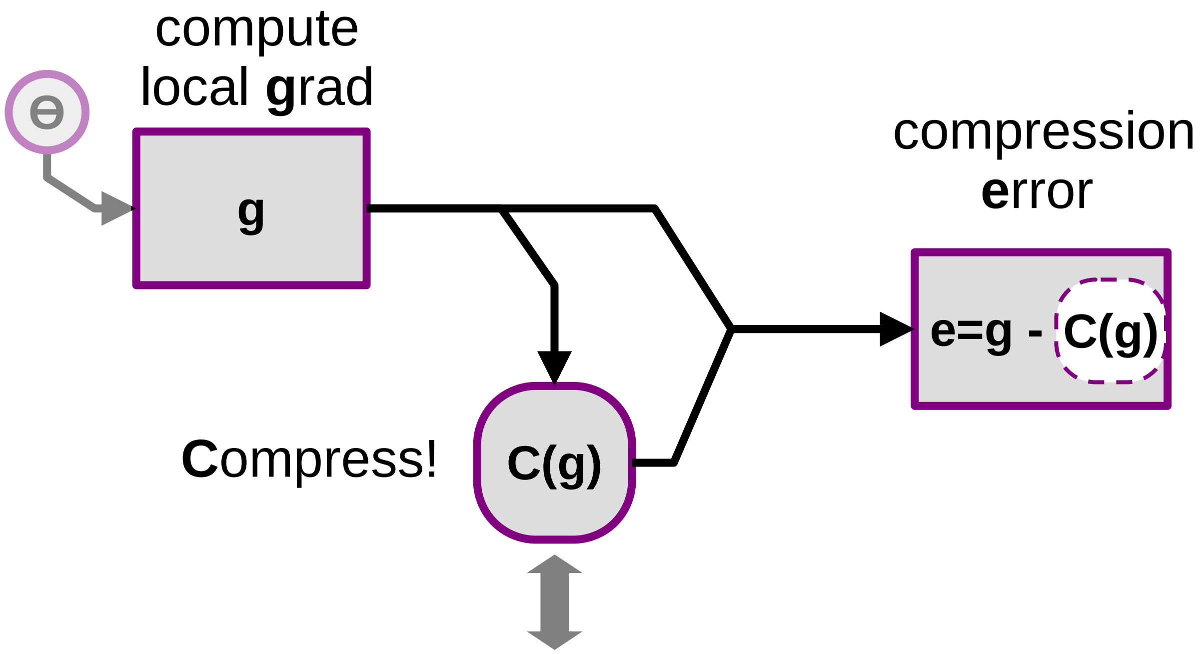


Error Feedback + PowerSGD

https://arxiv.org/abs/1901.09847 - EF theory
https://arxiv.org/abs/1905.13727 - PowerSGD

TL;DR - use extreme compression, e.g. 1-bit or top-5% gradients

- if you lose something in compression, **reuse it on the next step**

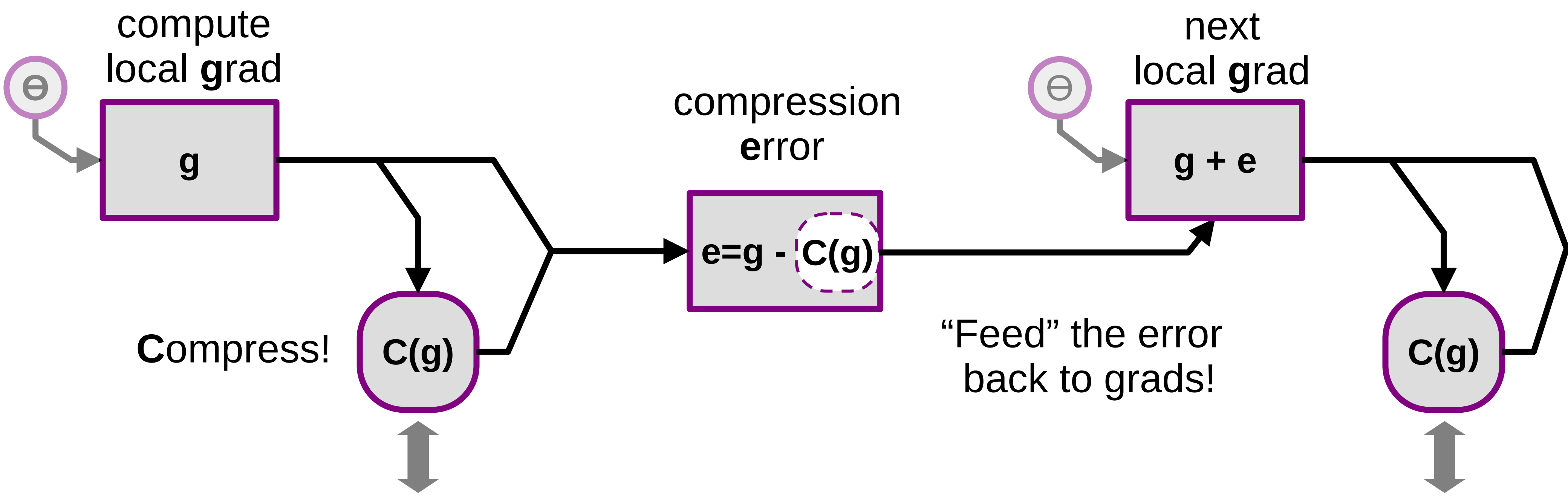


Error Feedback + PowerSGD

https://arxiv.org/abs/1901.09847 - EF theory
https://arxiv.org/abs/1905.13727 - PowerSGD

TL;DR - use extreme compression, e.g. 1-bit or top-5% gradients

- if you lose something in compression, **reuse it on the next step**

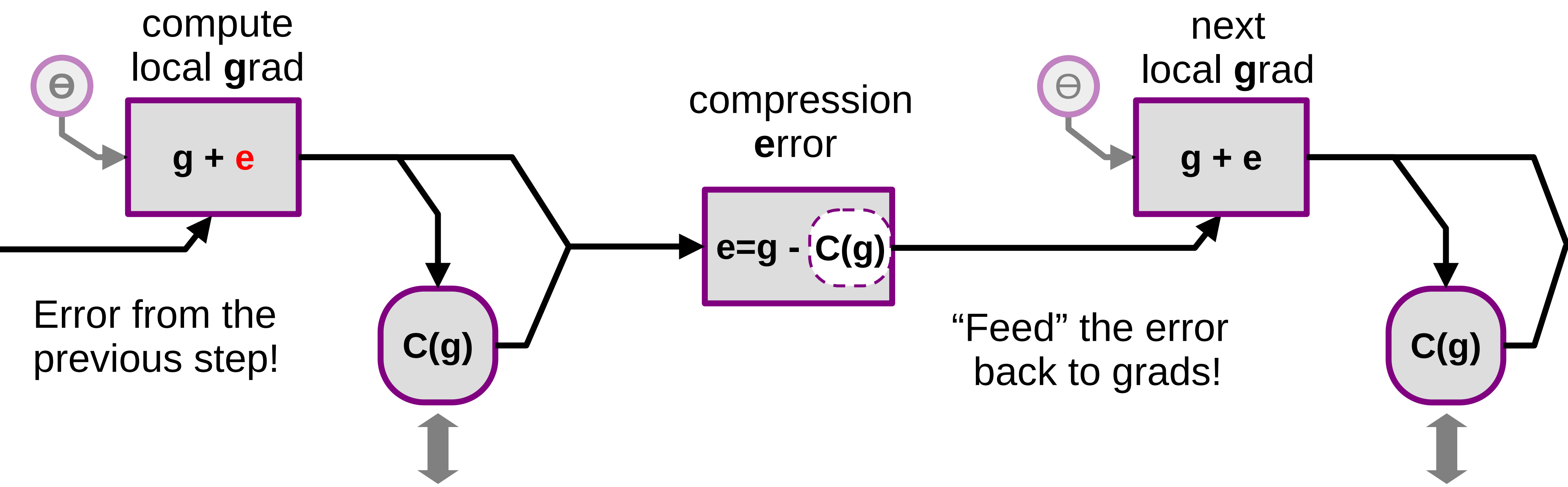


Error Feedback + PowerSGD

https://arxiv.org/abs/1901.09847 - EF theory
https://arxiv.org/abs/1905.13727 - PowerSGD

TL;DR - use extreme compression, e.g. 1-bit or top-5% gradients

- if you lose something in compression, **reuse it on the next step**



Error Feedback + PowerSGD

https://arxiv.org/abs/1901.09847 - EF theory
https://arxiv.org/abs/1905.13727 - PowerSGD

hyperparameters: learning rate γ, momentum parameter λ
initialize model parameters $\mathbf{x} \in \mathbb{R}^d$, momentum $\mathbf{m} \leftarrow \mathbf{0} \in \mathbb{R}^d$, replicated across workers
at each worker $w = 1, \ldots, W$ **do**
    initialize memory $\mathbf{e}_w \leftarrow \mathbf{0} \in \mathbb{R}^d$
    for each iterate $t = 0, \ldots$ **do**
        Compute a stochastic gradient $\mathbf{g}_w \in \mathbb{R}^d$.
        $\Delta_w \leftarrow \mathbf{g}_w + \mathbf{e}_w$ ▷ Incorporate error-feedback into update
        $\mathcal{C}(\Delta_w) \leftarrow \textsc{compress}(\Delta_w)$
        $\mathbf{e}_w \leftarrow \Delta_w - \textsc{decompress}(\mathcal{C}(\Delta_w))$ ▷ Memorize local errors
        $\mathcal{C}(\Delta) \leftarrow \textsc{aggregate}(\mathcal{C}(\Delta_1), \ldots, \mathcal{C}(\Delta_W))$ ▷ Exchange gradients
        $\Delta' \leftarrow \textsc{decompress}(\mathcal{C}(\Delta))$ ▷ Reconstruct an update $\in \mathbb{R}^d$
        $\mathbf{m} \leftarrow \lambda \mathbf{m} + \Delta'$
        $\mathbf{x} \leftarrow \mathbf{x} - \gamma\,(\Delta' + \mathbf{m})$
    end for
end at

PowerSGD: low-rank approx grads + Error Feedback

https://arxiv.org/abs/1901.09847 - EF theory
https://arxiv.org/abs/1905.13727 - PowerSGD

The update vector Δ_w is treated as a list of tensors corresponding to individual model parameters. Vector-shaped parameters (biases) are aggregated uncompressed. Other parameters are reshaped into matrices. The functions below operate on such matrices independently. For each matrix $M \in \mathbb{R}^{n \times m}$, a corresponding $Q \in \mathbb{R}^{m \times r}$ is initialized from an i.i.d. standard normal distribution.

function COMPRESS+AGGREGATE(update matrix $M \in \mathbb{R}^{n \times m}$, previous $Q \in \mathbb{R}^{m \times r}$)

  $P \leftarrow MQ$

  $P \leftarrow$ ALL REDUCE MEAN(P) ▷ Now, $P = \frac{1}{W}(M_1 + \ldots + M_W)Q$

  $\hat{P} \leftarrow$ ORTHOGONALIZE(P) ▷ Orthonormal columns

  $Q \leftarrow M^\top \hat{P}$

  $Q \leftarrow$ ALL REDUCE MEAN(Q) ▷ Now, $Q = \frac{1}{W}(M_1 + \ldots + M_W)^\top \hat{P}$

  return the compressed representation $(\hat{P}, Q)$.

end function

function DECOMPRESS$(\hat{P} \in \mathbb{R}^{n \times r}, Q \in \mathbb{R}^{m \times r})$

  return $\hat{P} Q^\top$

end function

Read More: gradient compression

https://arxiv.org/abs/1901.09847 - EF theory

https://arxiv.org/abs/1905.13727 - PowerSGD

https://arxiv.org/abs/2106.05203 - better EF’21

https://arxiv.org/abs/2110.03294 - more EF’21

```
import torch.distributed.algorithms.ddp_comm_hooks.powerSGD_hook as powerSGD

ddp_model = nn.parallel.DistributedDataParallel(
    module=model,
    device_ids=[rank],
)

state = PowerSGD.PowerSGDState(
  process_group=process_group,
  matrix_approximation_rank=1,
  start_powerSGD_iter=1_000,
)
ddp_model.register_comm_hook(state, PowerSGD.powerSGD_hook)
```


"That's all Folks!"
Isberg©