Distributed Deep Learning

Max Ryabinin*

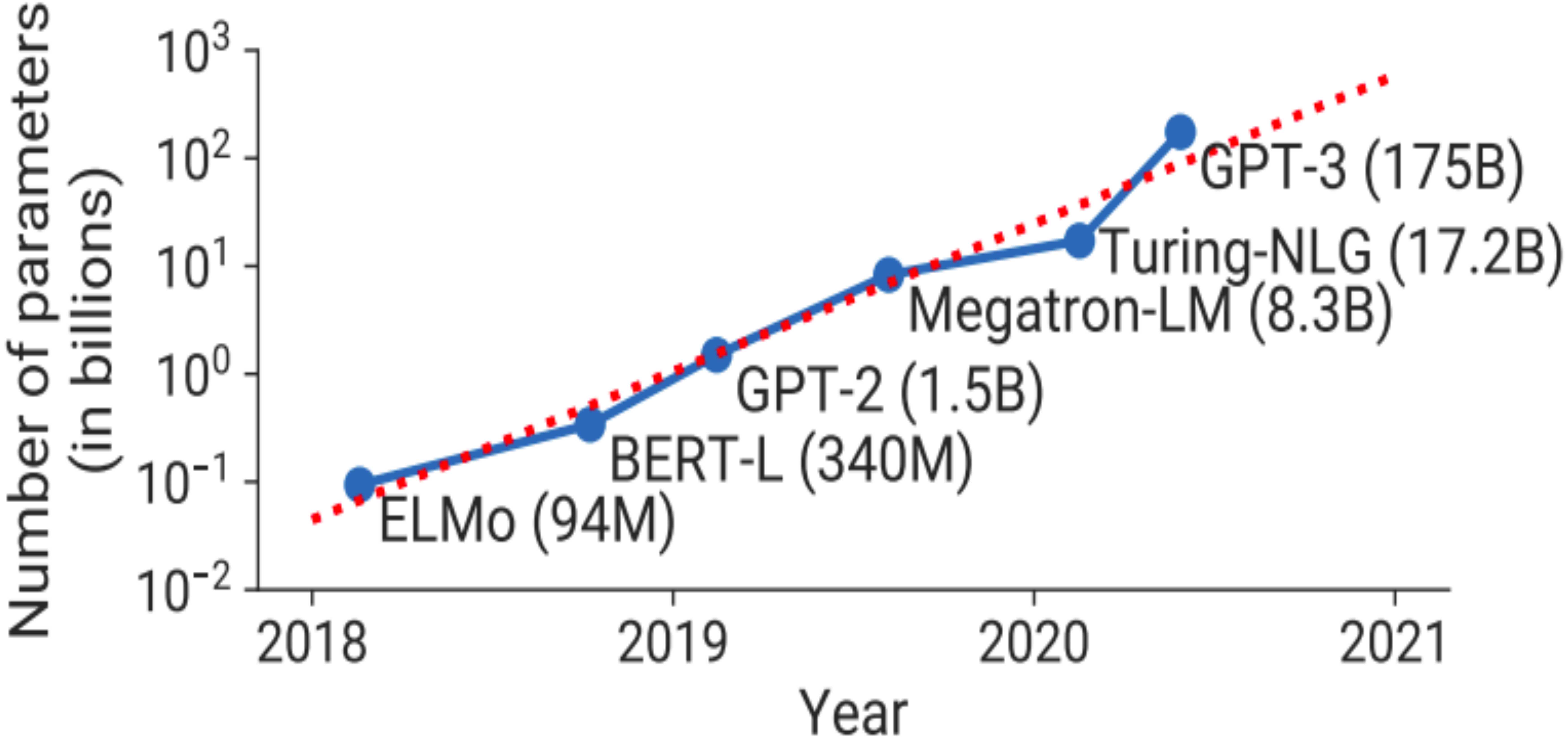
Yandex Research

NATIONAL RESEARCH UNIVERSITY

*Several slides are reused from github.com/yandexdataschool/dlatscale_draft
and @justheuristic's distributed DL overviews

Motivation

arxiv.org/abs/2104.04473



Large problems need large models

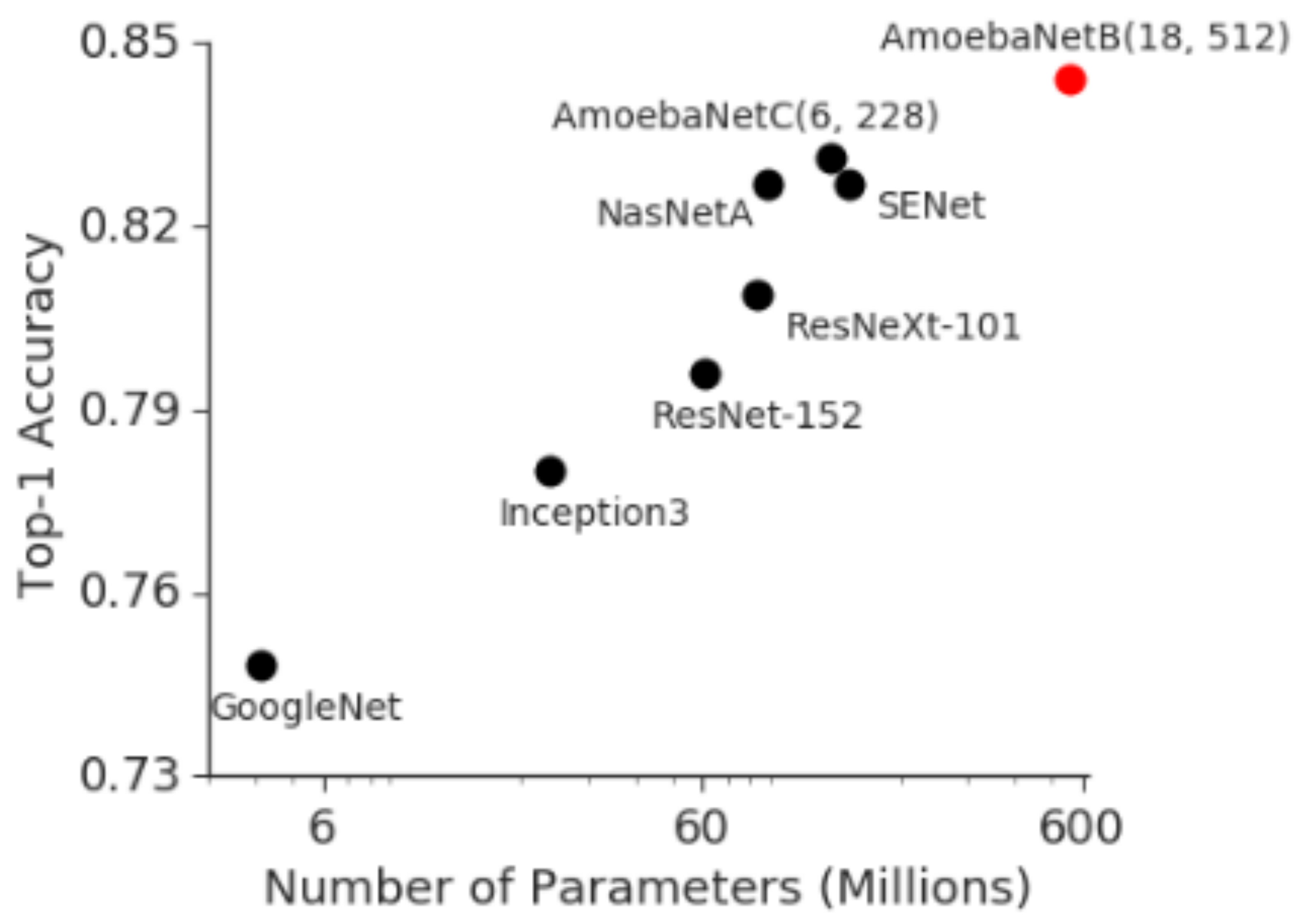
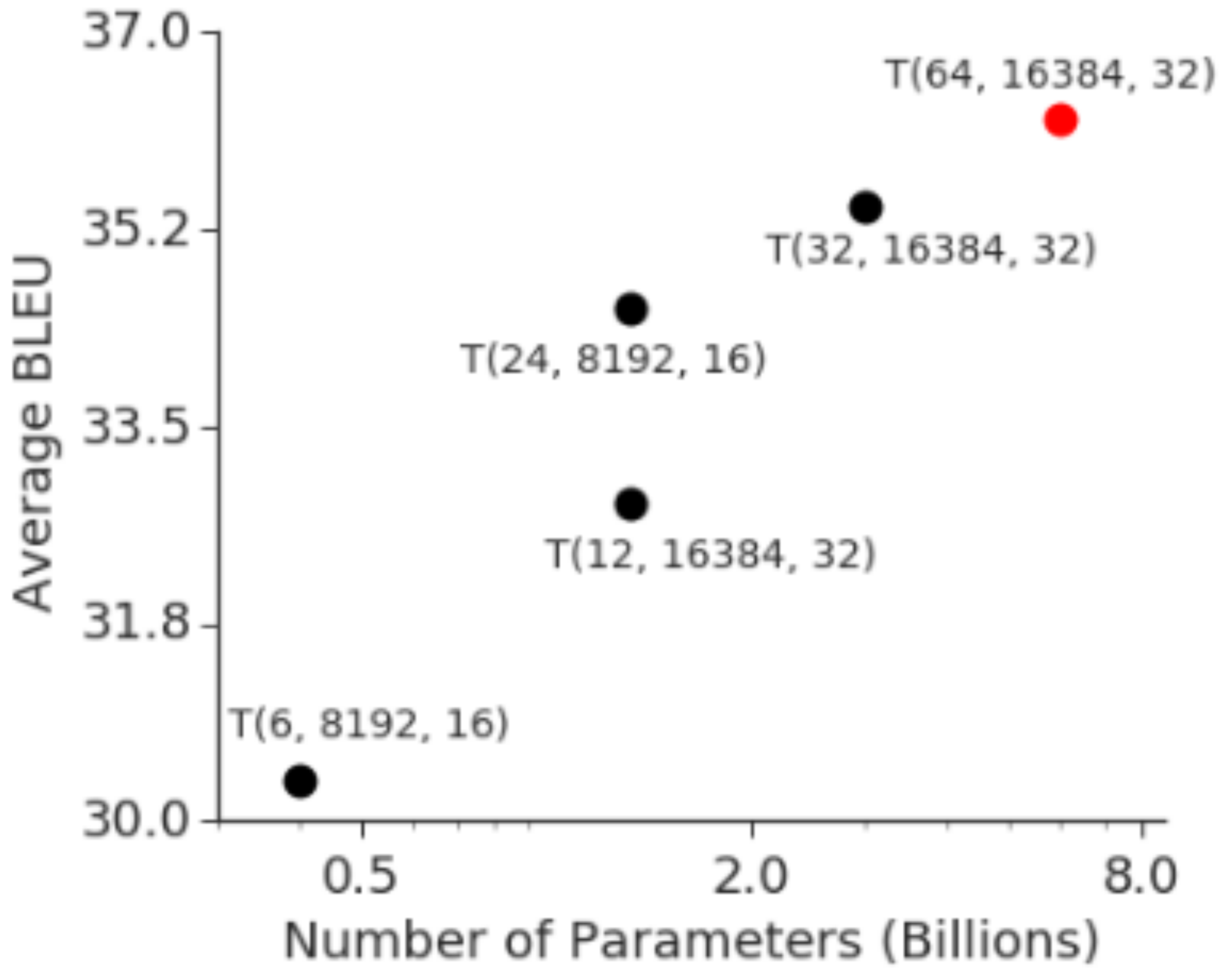


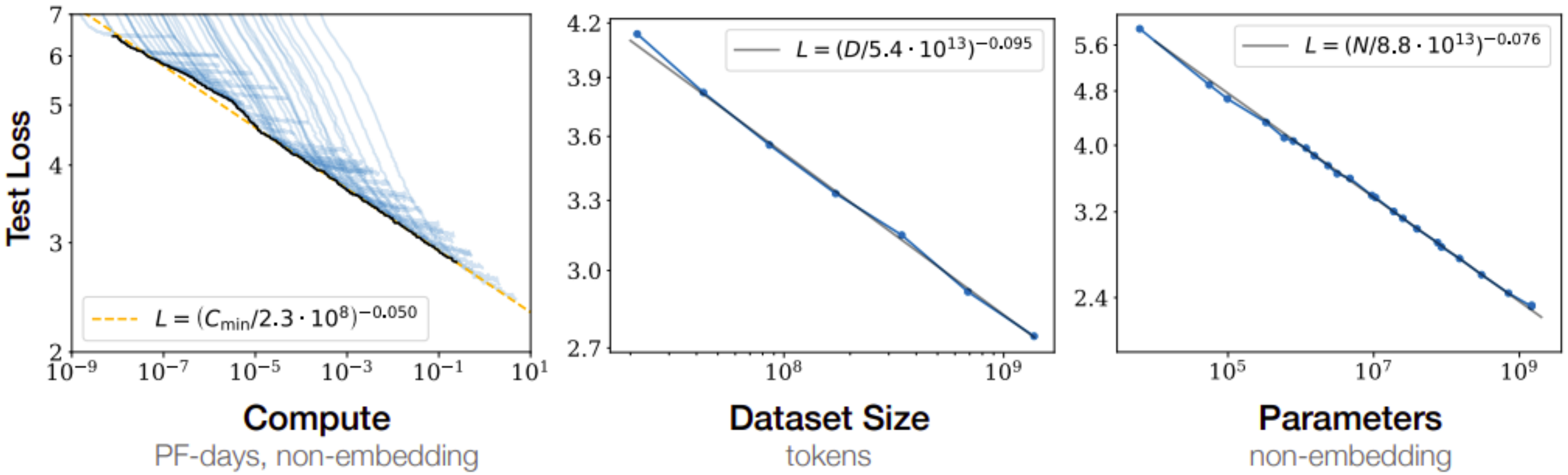
Image classification
ImageNet



Machine translation
average over WMT

Source: arxiv.org/abs/1811.06965

Scaling Laws for Neural Language Models



Source: arxiv.org/abs/2001.08361

Machine Learning supertasks

- Image classification – ImageNet, JFT300M
- Image generation – ImageNet (BigGAN)
- Language models – CommonCrawl, BERT/MLM
- Machine Translation – multilingual translation

Distributed training to the rescue!

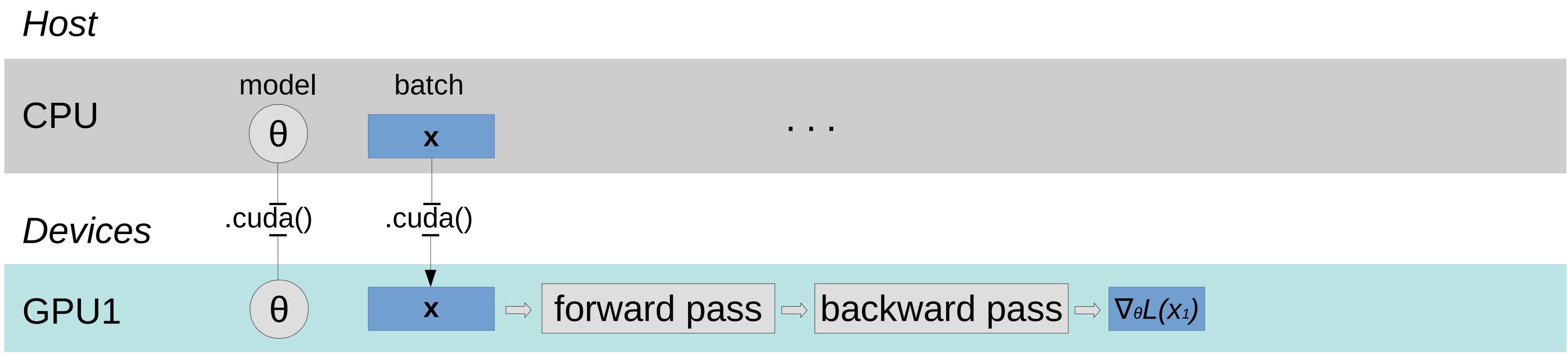
- To gather sufficient computational resources, train on multiple computers
- **Goal of this talk:** a broad overview of practical algorithms in Distributed DL
- Two main groups of methods:
  - Data-parallel training: parallelize SGD over the batch axis
  - Model-parallel training: shard the model, run it on several devices

Data-parallel training (naive)

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

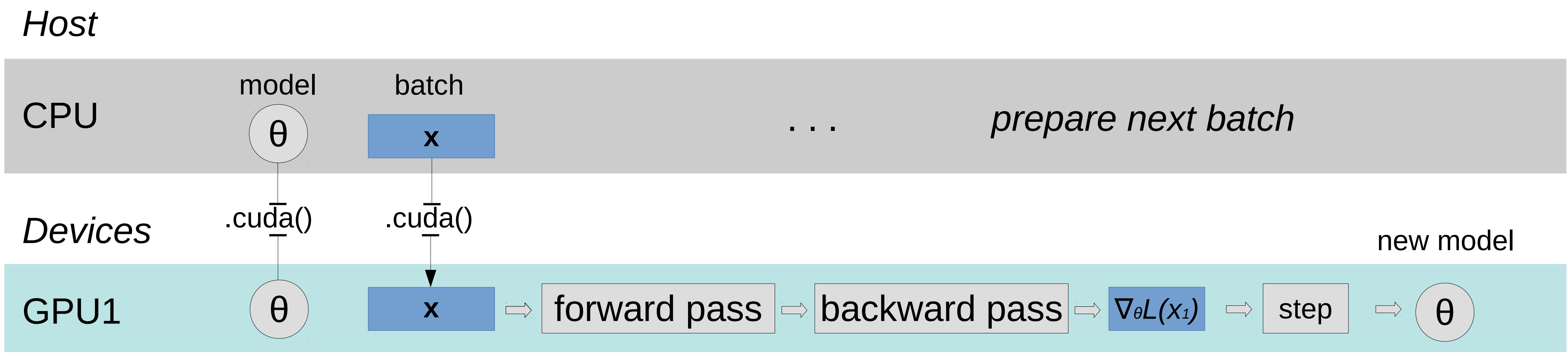
Data-parallel training (naive)

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



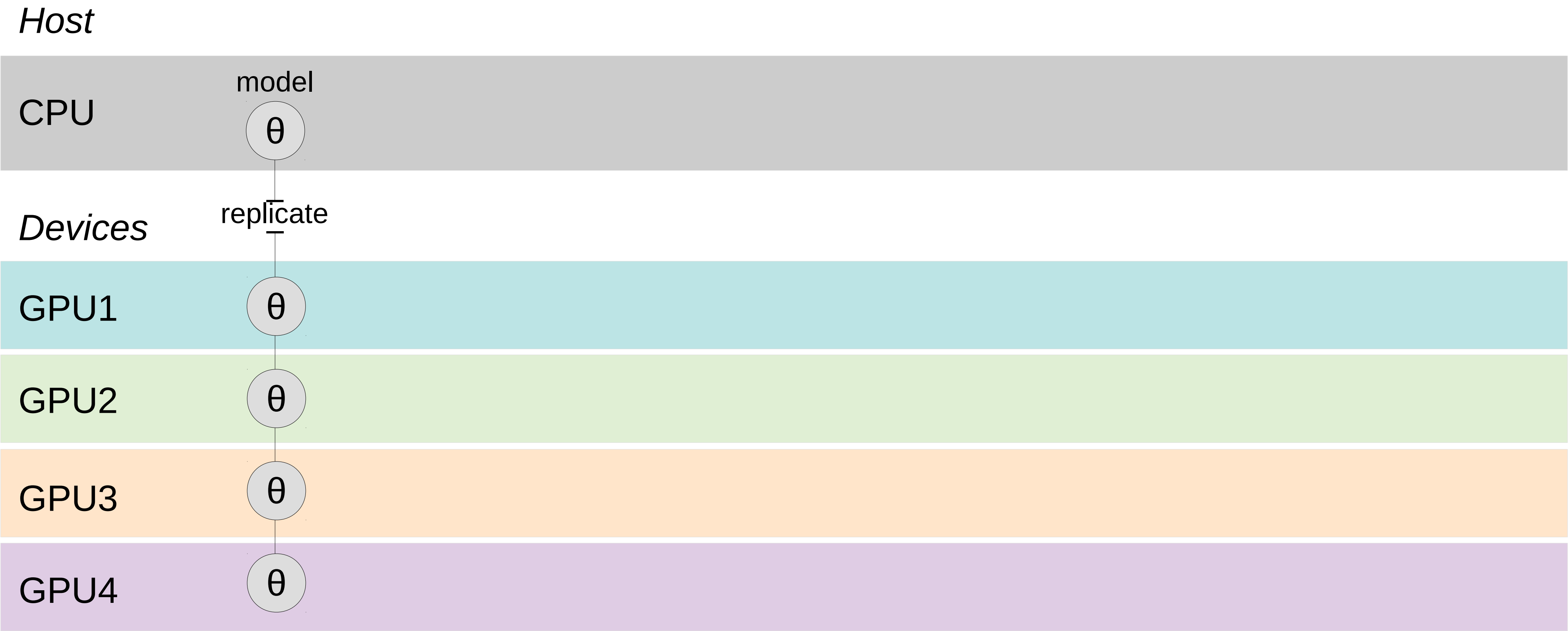
Data-parallel training (naive)

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



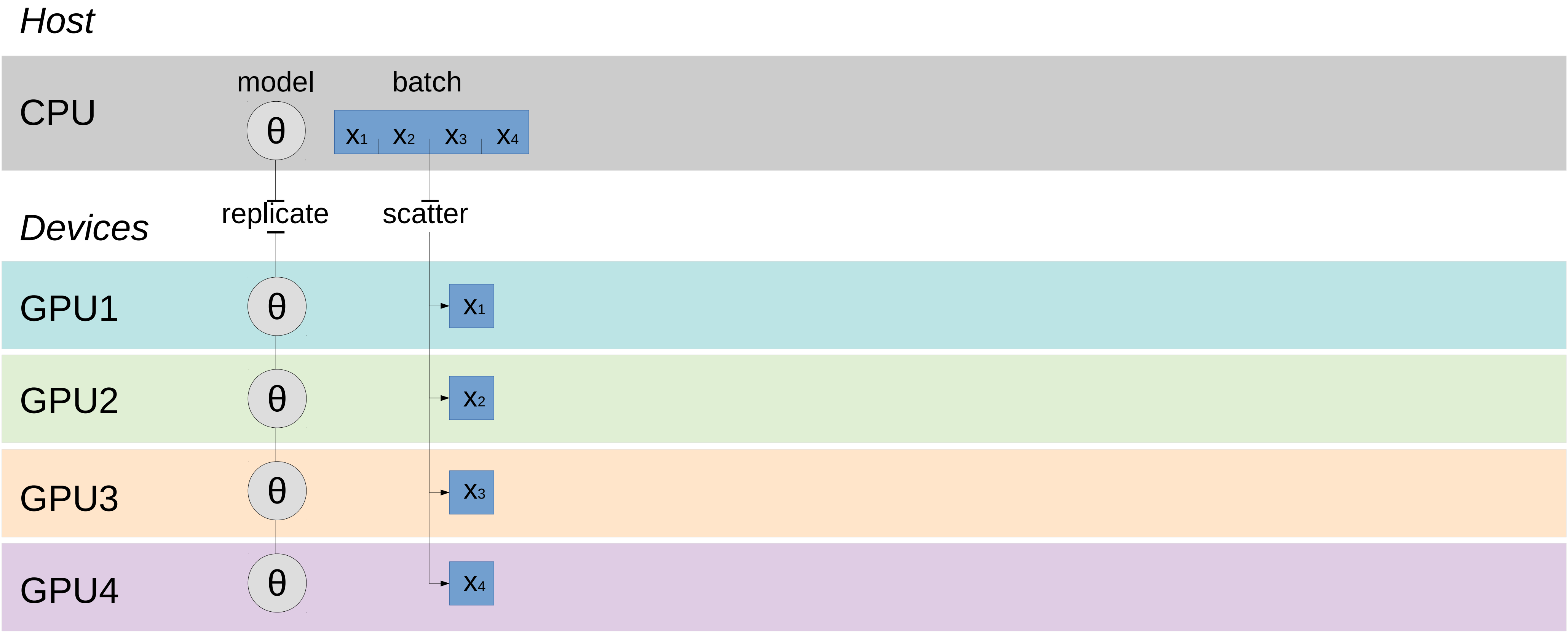
Data-parallel training (naive)

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



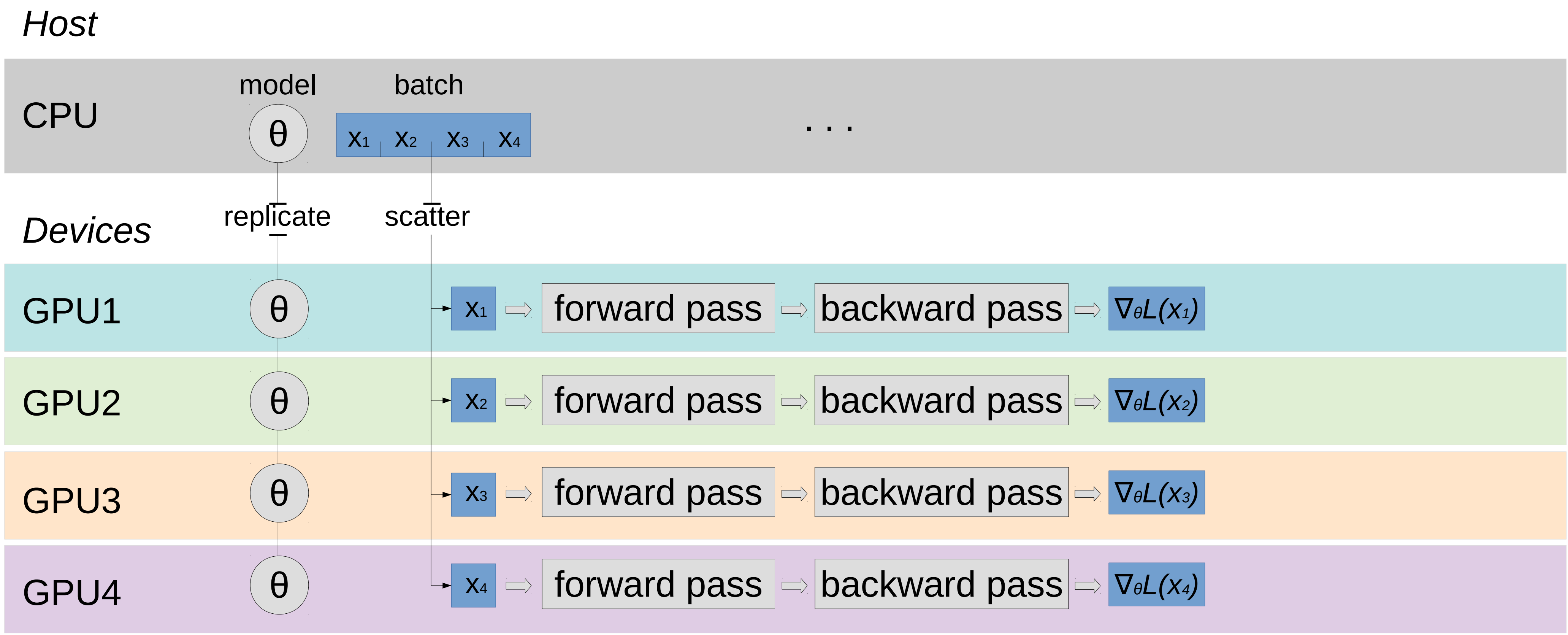
Data-parallel training (naive)

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



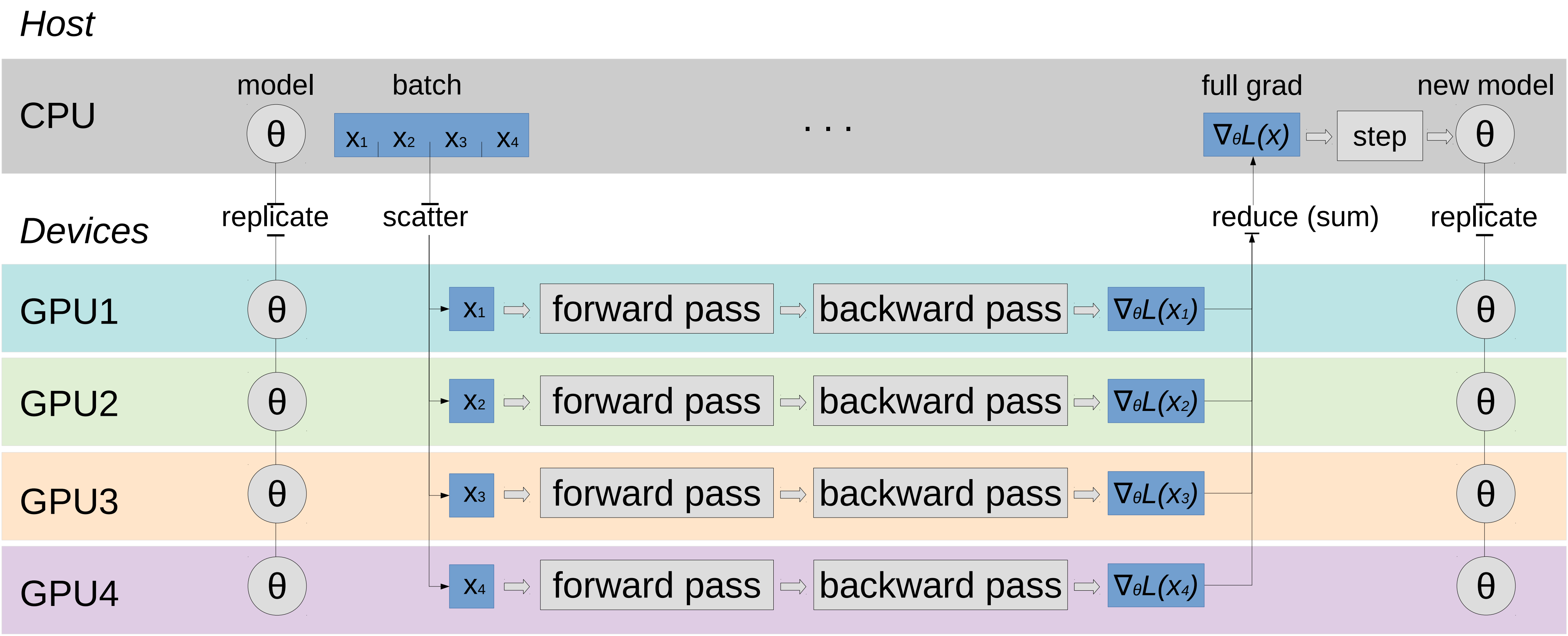
Data-parallel training (naive)

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



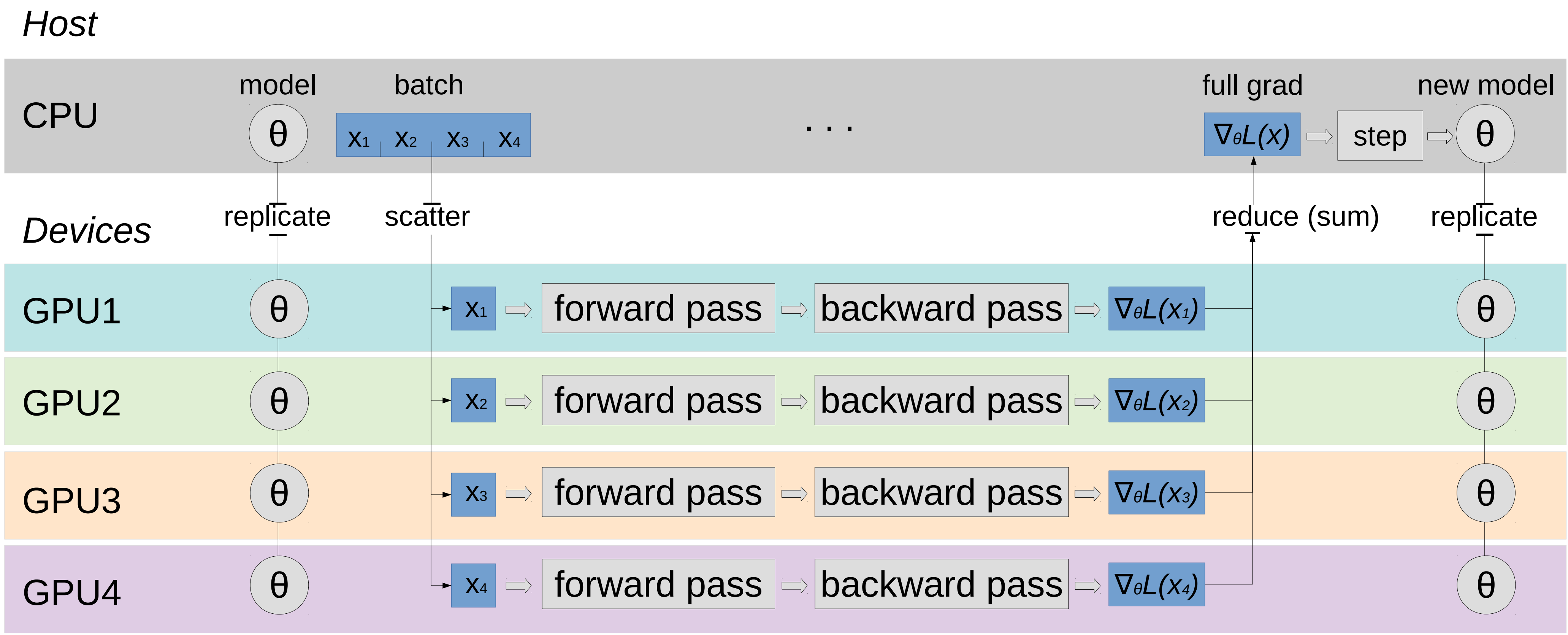
Data-parallel training (naive)

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



Data-parallel training (naive)

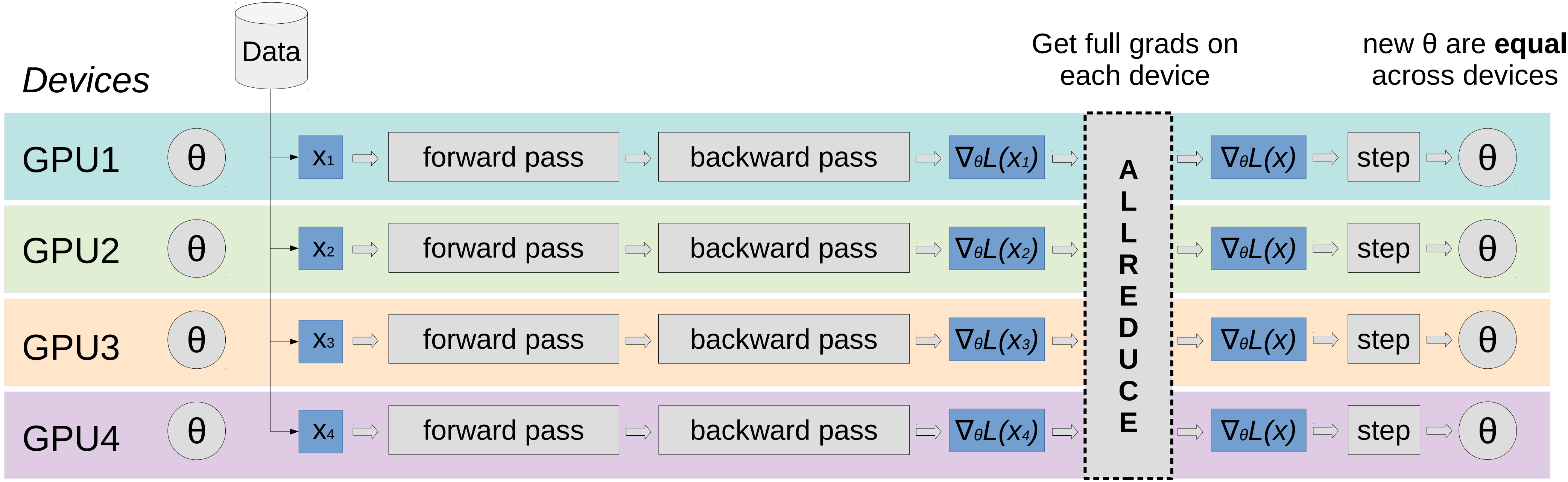
cs.cmu.edu/~muli/file/parameter_server_osdi14.pdf



All-Reduce data parallel

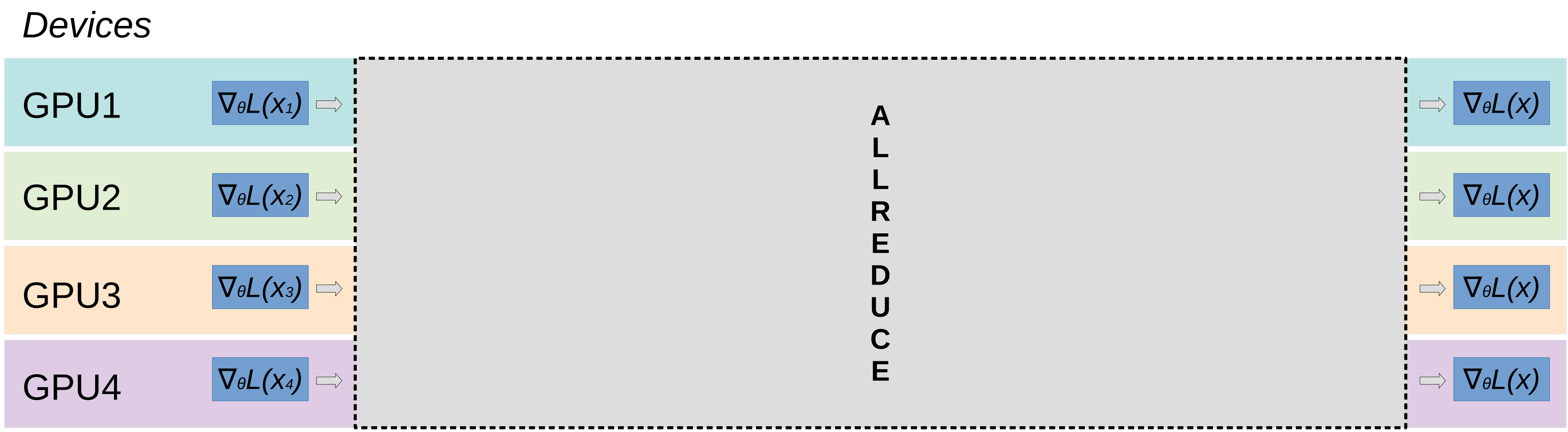
arxiv.org/abs/1706.02677

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



Faster allreduce

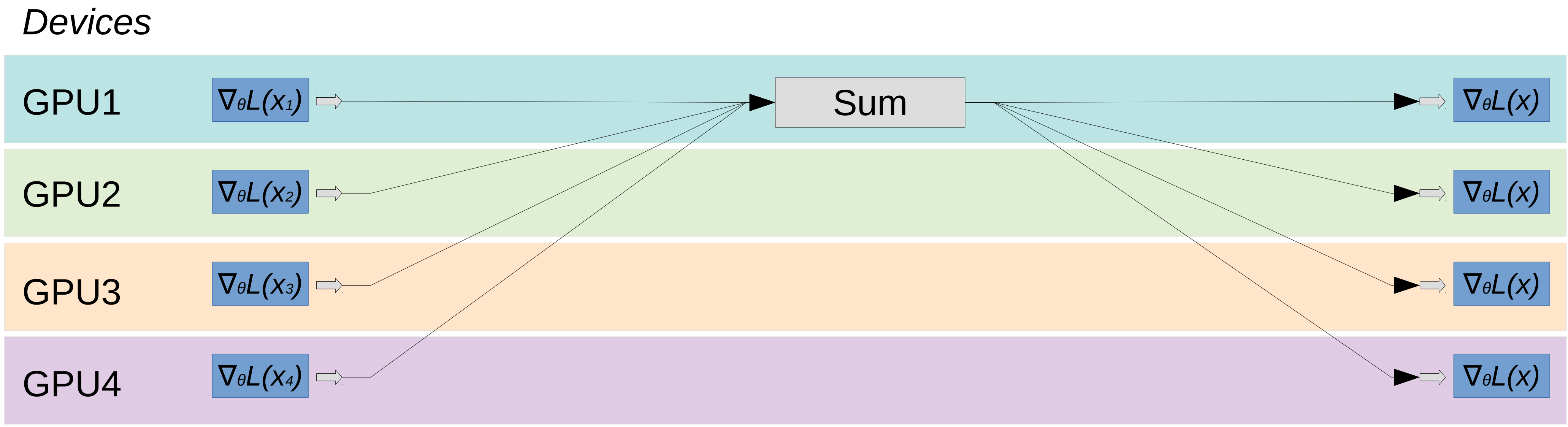
Input: each device has its its own vector
Output: each device gets a sum of all vectors



Faster allreduce

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

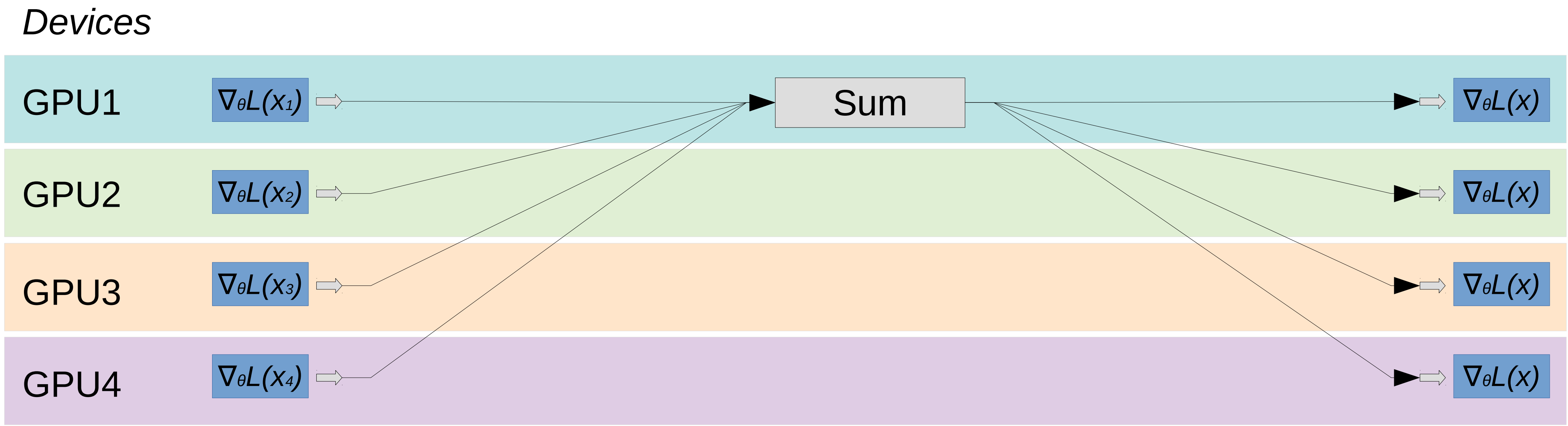
Naive implementation



Faster allreduce

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

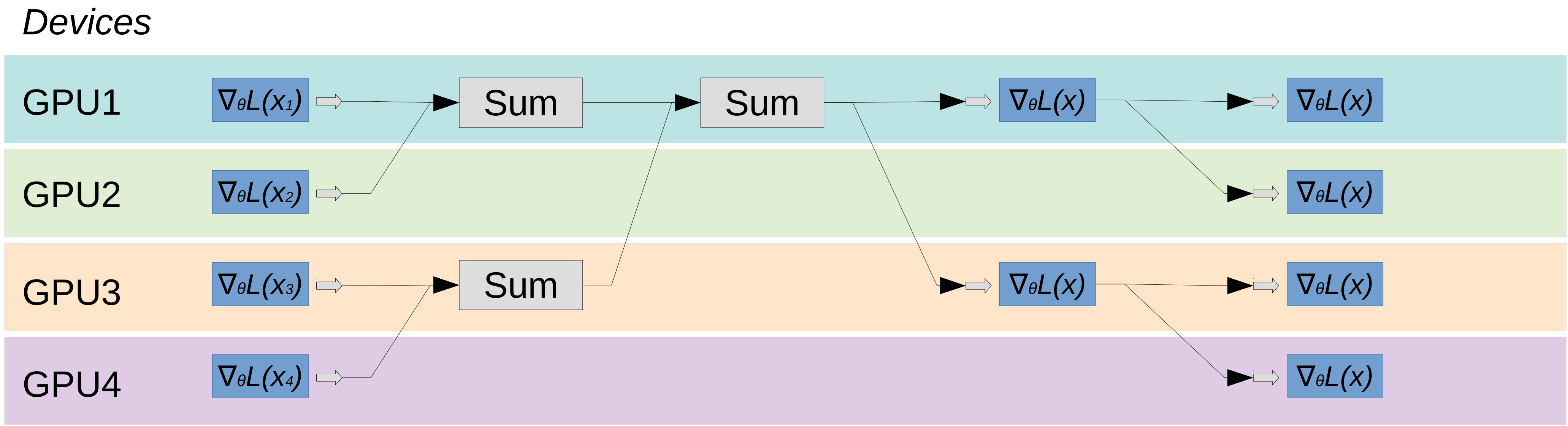
Q: Can we do better?



Faster allreduce

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

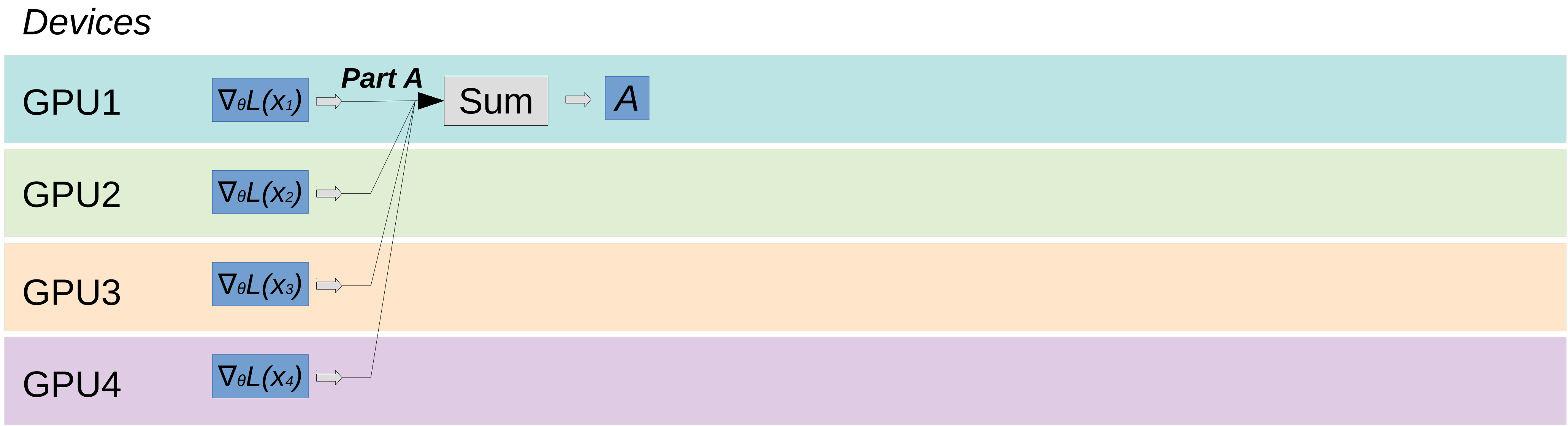
Tree-allreduce



Faster allreduce

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

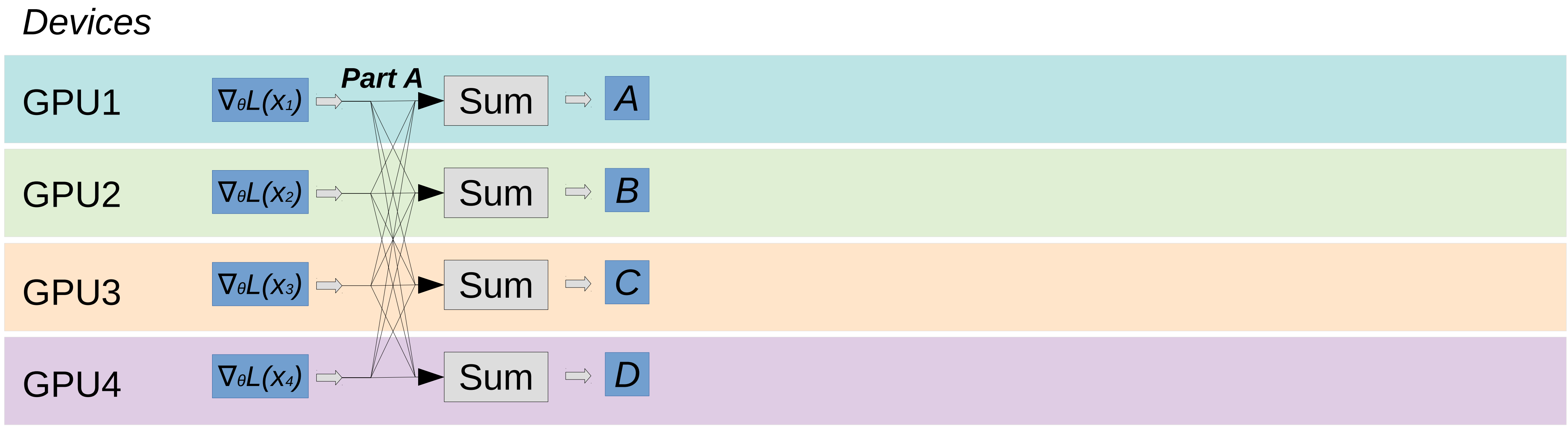
Butterfly-allreduce – split data into chunks (ABCD)



Faster allreduce

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

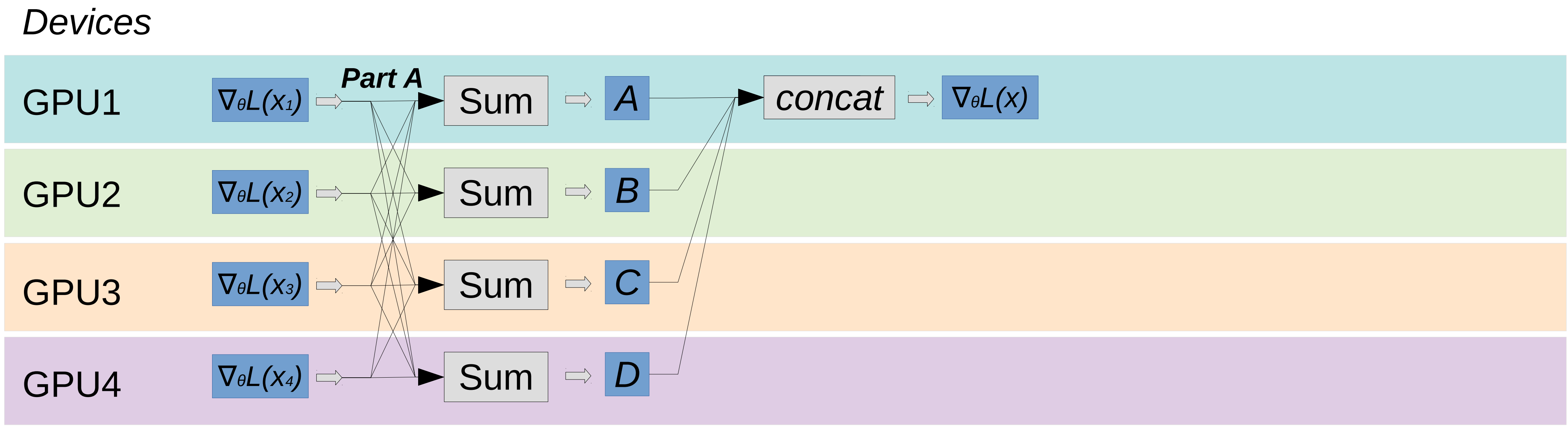
Butterfly-allreduce – split data into chunks (ABCD)



Faster allreduce

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

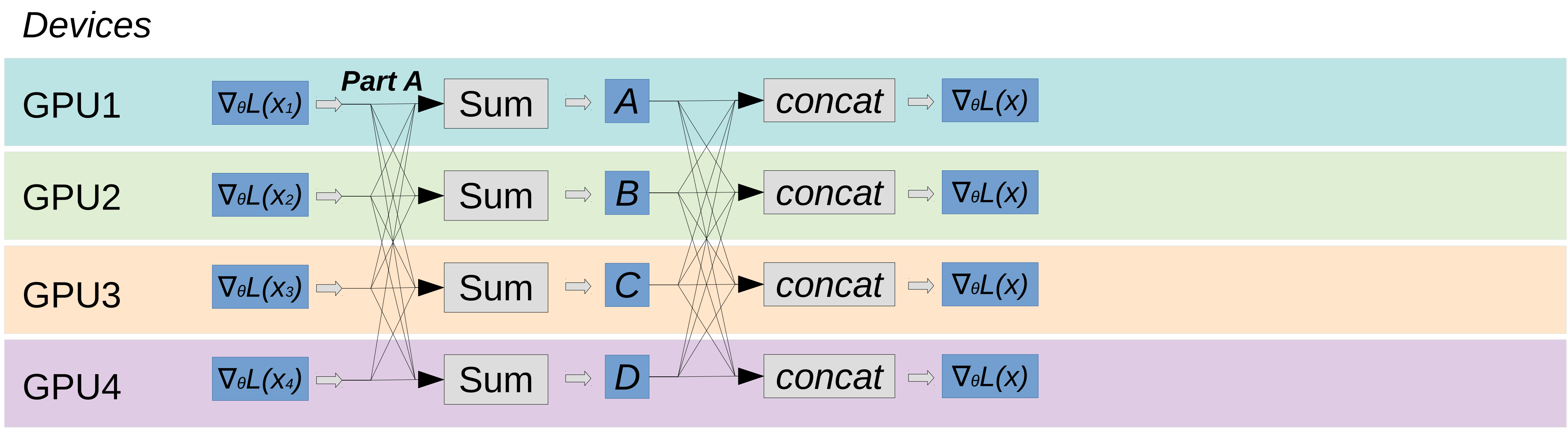
Butterfly-allreduce – split data into chunks (ABCD)



Faster allreduce

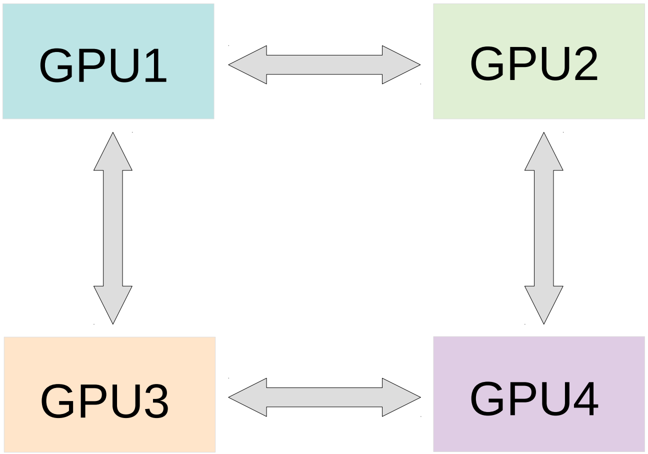
Input: each device has its its own vector
Output: each device gets a sum of all vectors

Ring-allreduce – split data into chunks (ABCD)



Ring allreduce

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



Ring topology

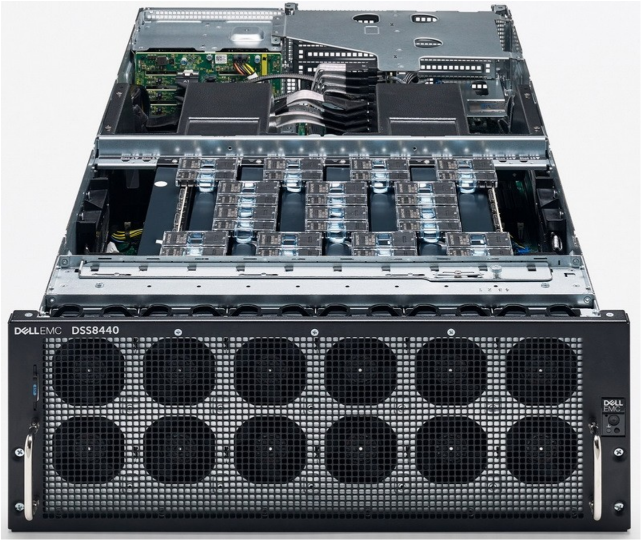


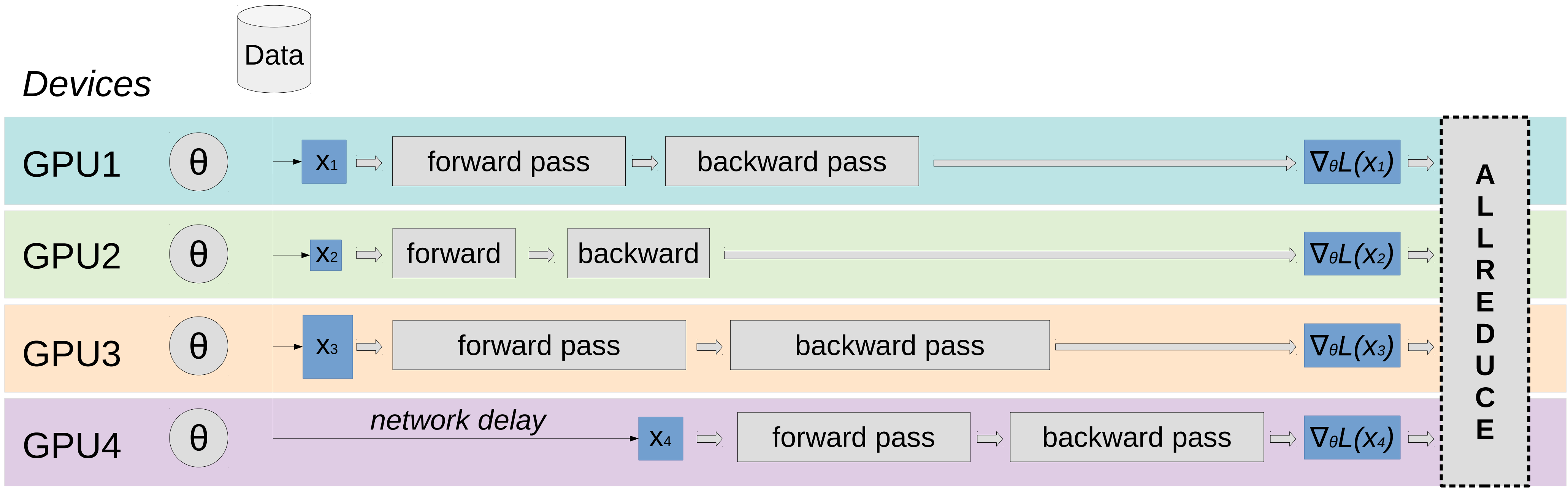
Image: graphcore ipu server

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

All-Reduce data parallel VS reality

arxiv.org/abs/1706.02677

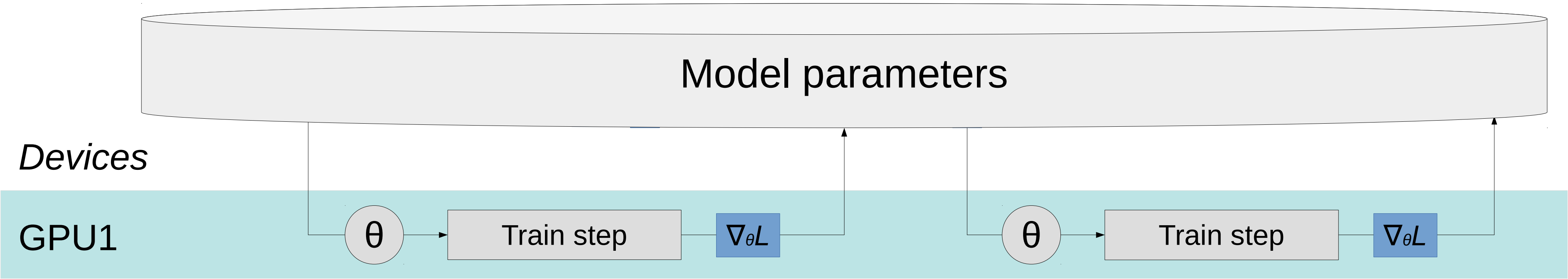
Each gpu has different processing time & delays
Q: can we improve device utilization?



Asynchronous data-parallel training

HOGWILD! arxiv.org/abs/1106.5730

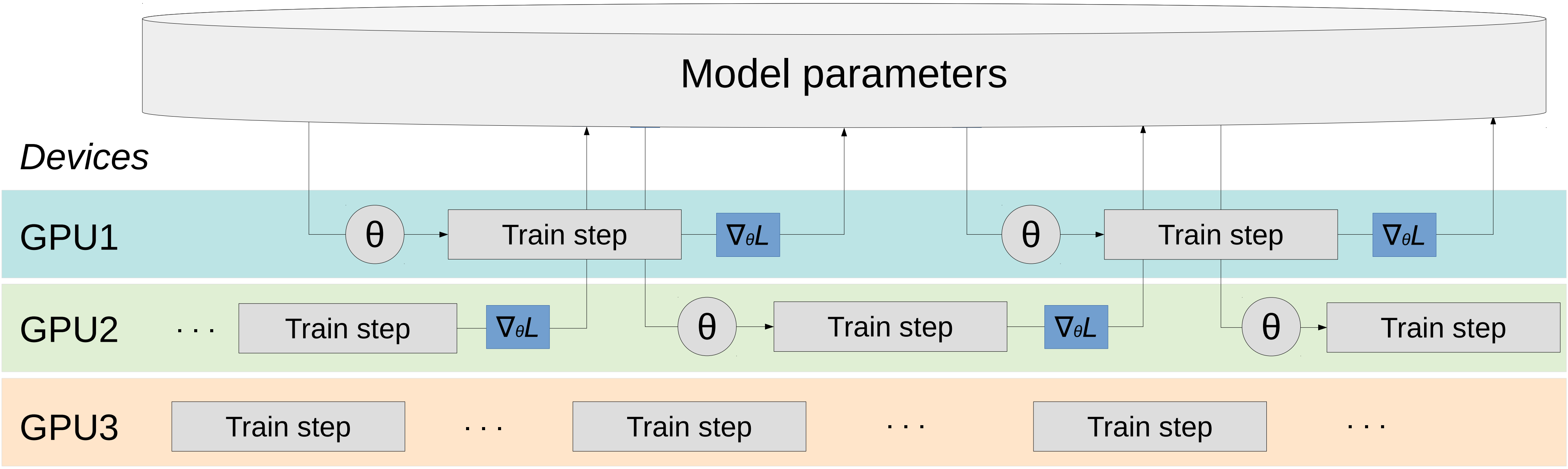
Idea: remove synchronization step alltogether, use parameter server



Asynchronous data-parallel training

HOGWILD! arxiv.org/abs/1106.5730

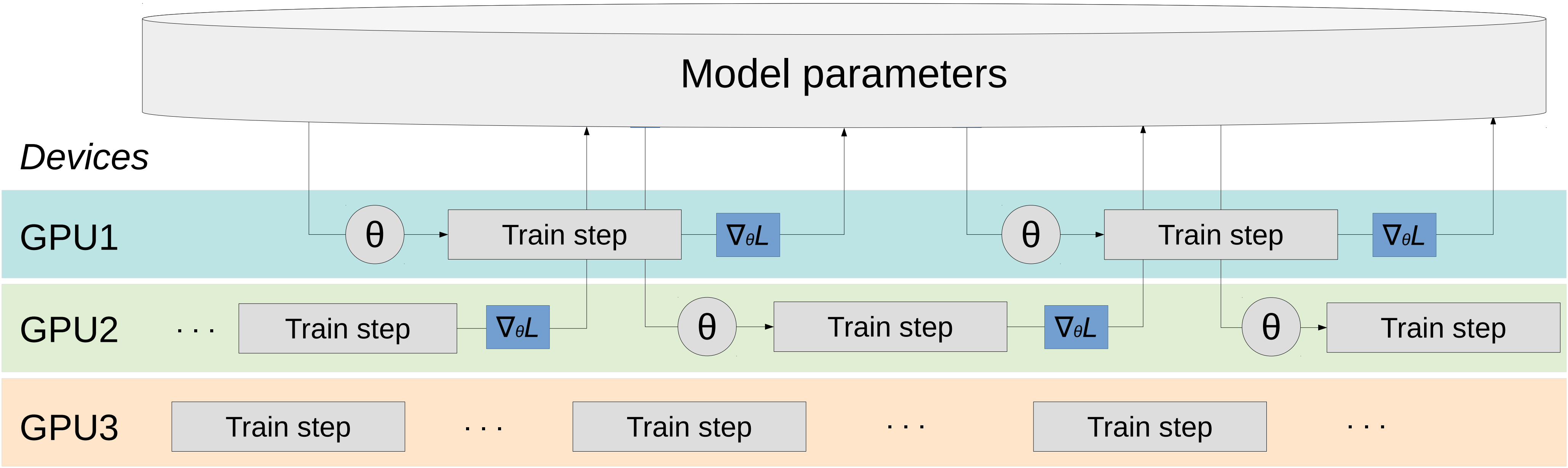
Idea: remove synchronization step alltogether, use parameter server



Asynchronous data-parallel training

HOGWILD! arxiv.org/abs/1106.5730

Idea: remove synchronization step alltogether, use parameter server

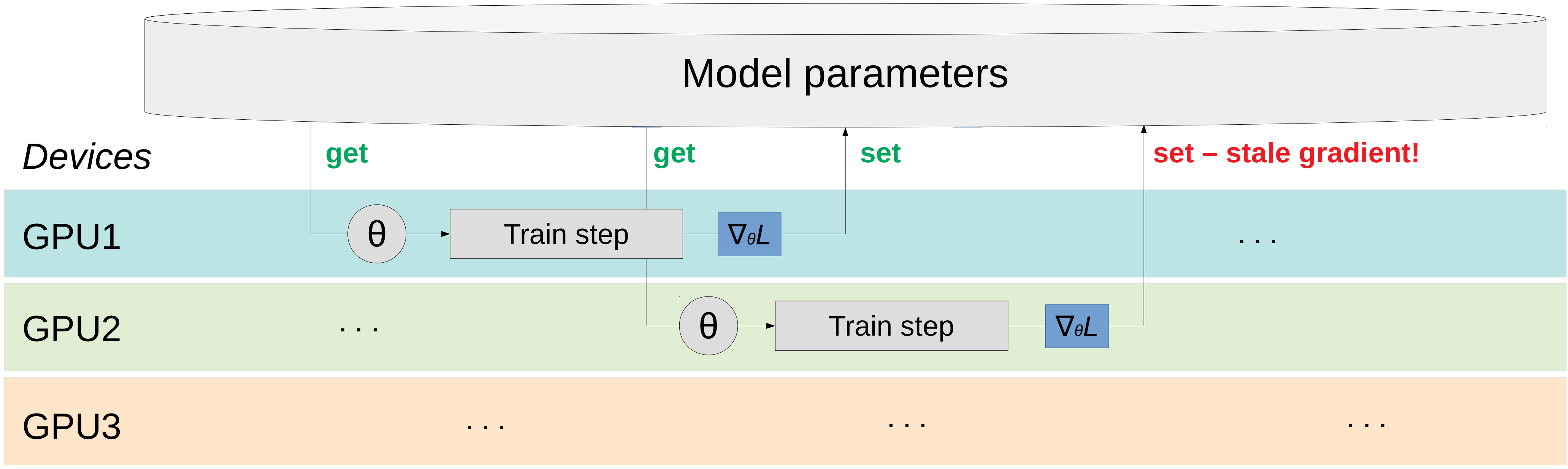


Q: have we lost anything by going asynchronous?

Asynchronous data-parallel training

HOGWILD! arxiv.org/abs/1106.5730

Idea: remove synchronization step alltogether, use parameter server

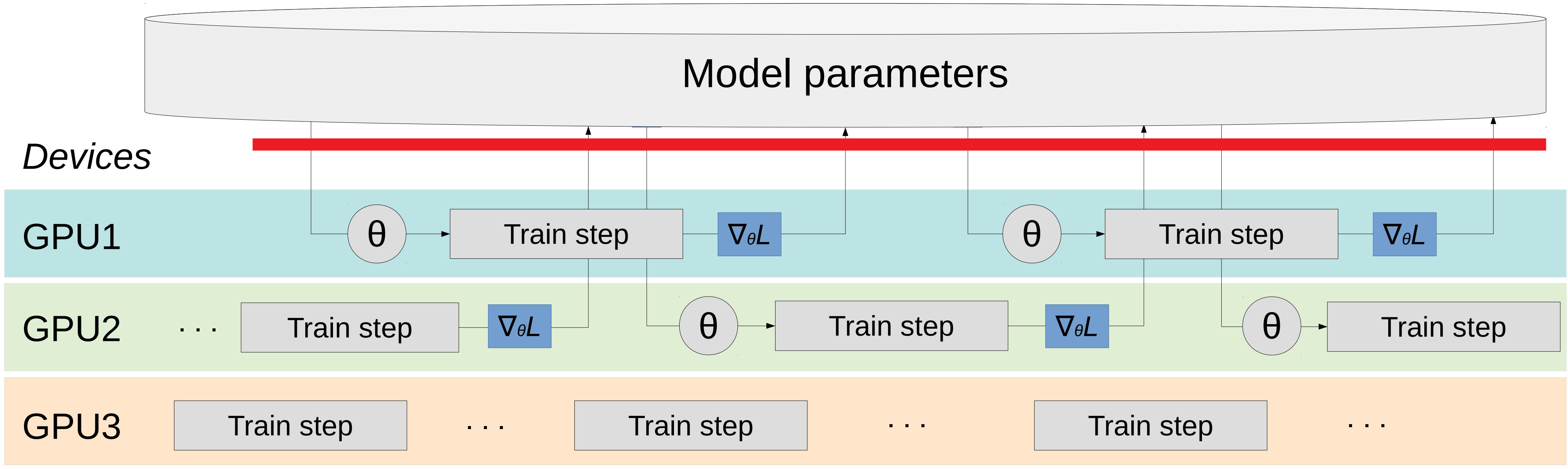


Correction for staleness: arxiv.org/abs/1511.05950 & many others

Recap: Parameter Server

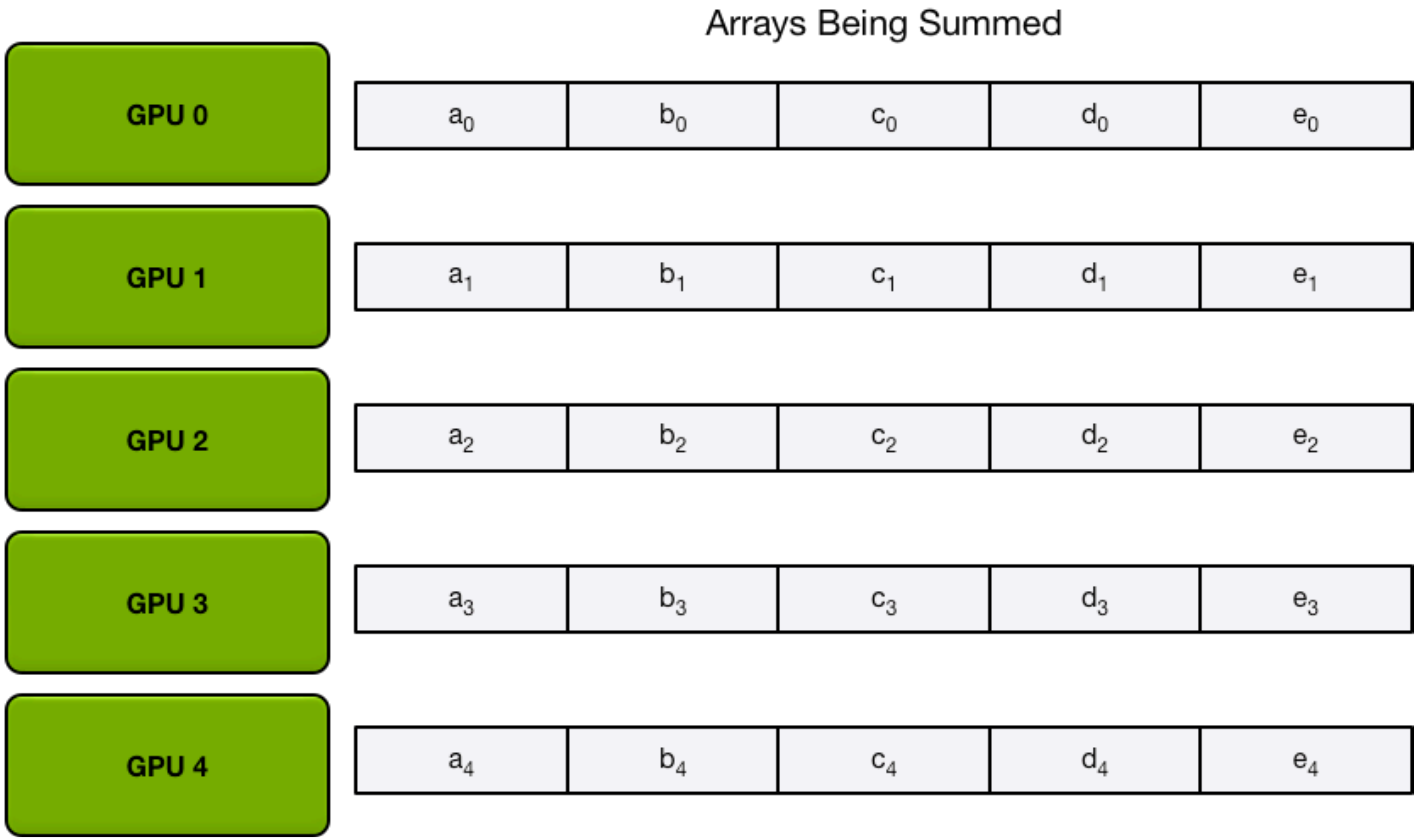
HOGWILD! arxiv.org/abs/1106.5730

Idea: remove synchronization step alltogether, use parameter server



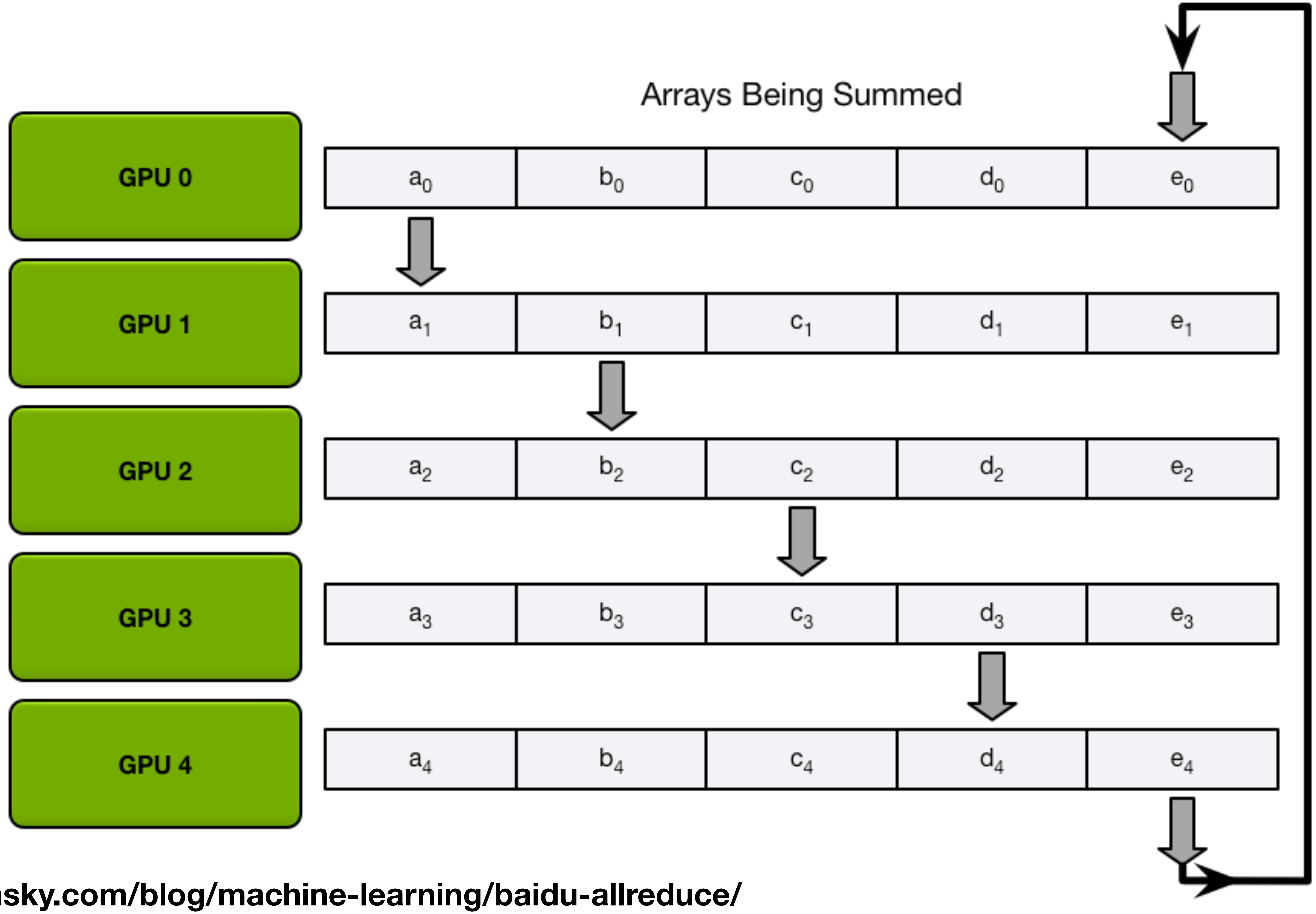
Problem: parameter servers need to ingest tons of data over training

Ring All-Reduce



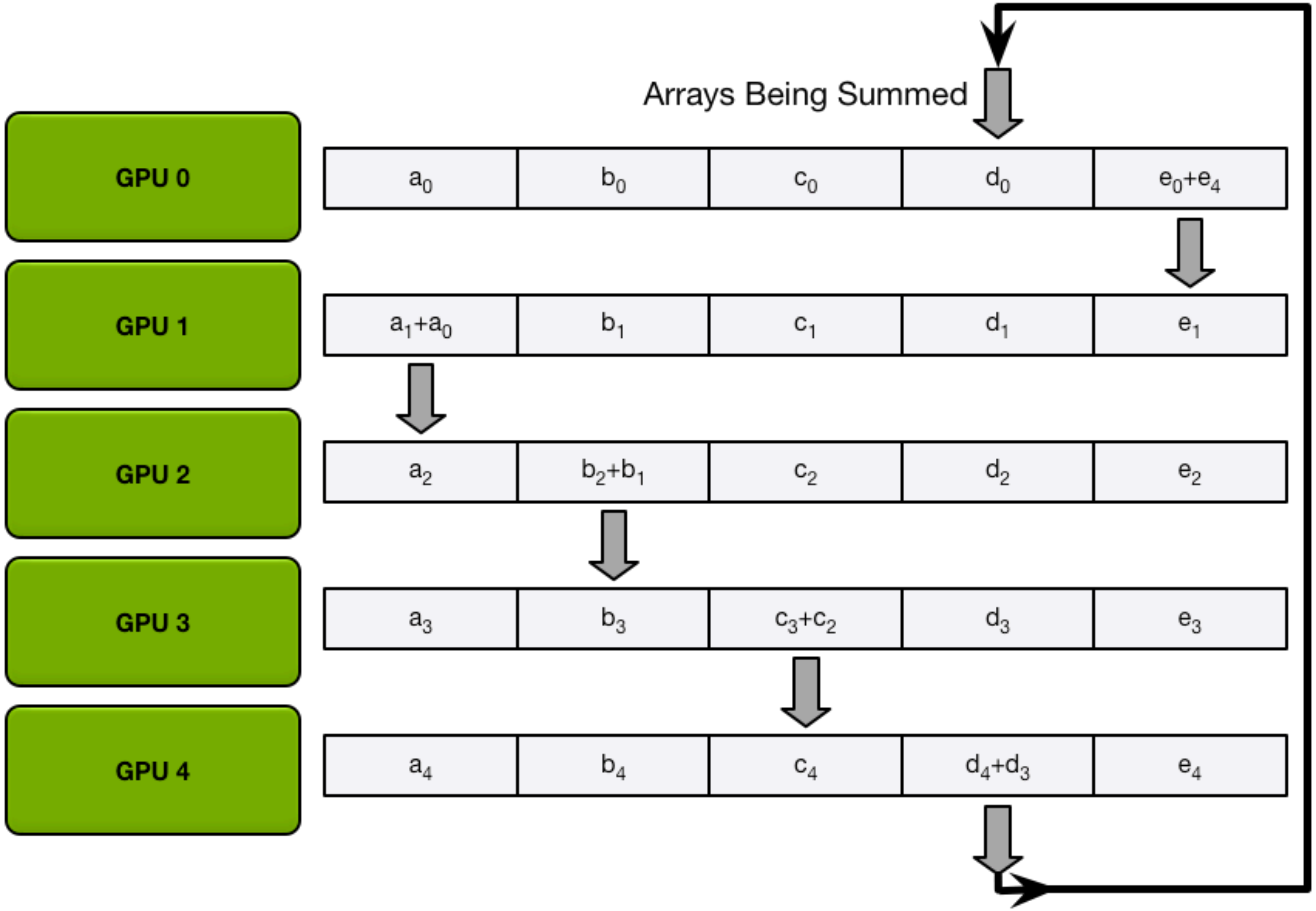
src: andrew.gibiansky.com/blog/machine-learning/baidu-allreduce/

Ring All-Reduce



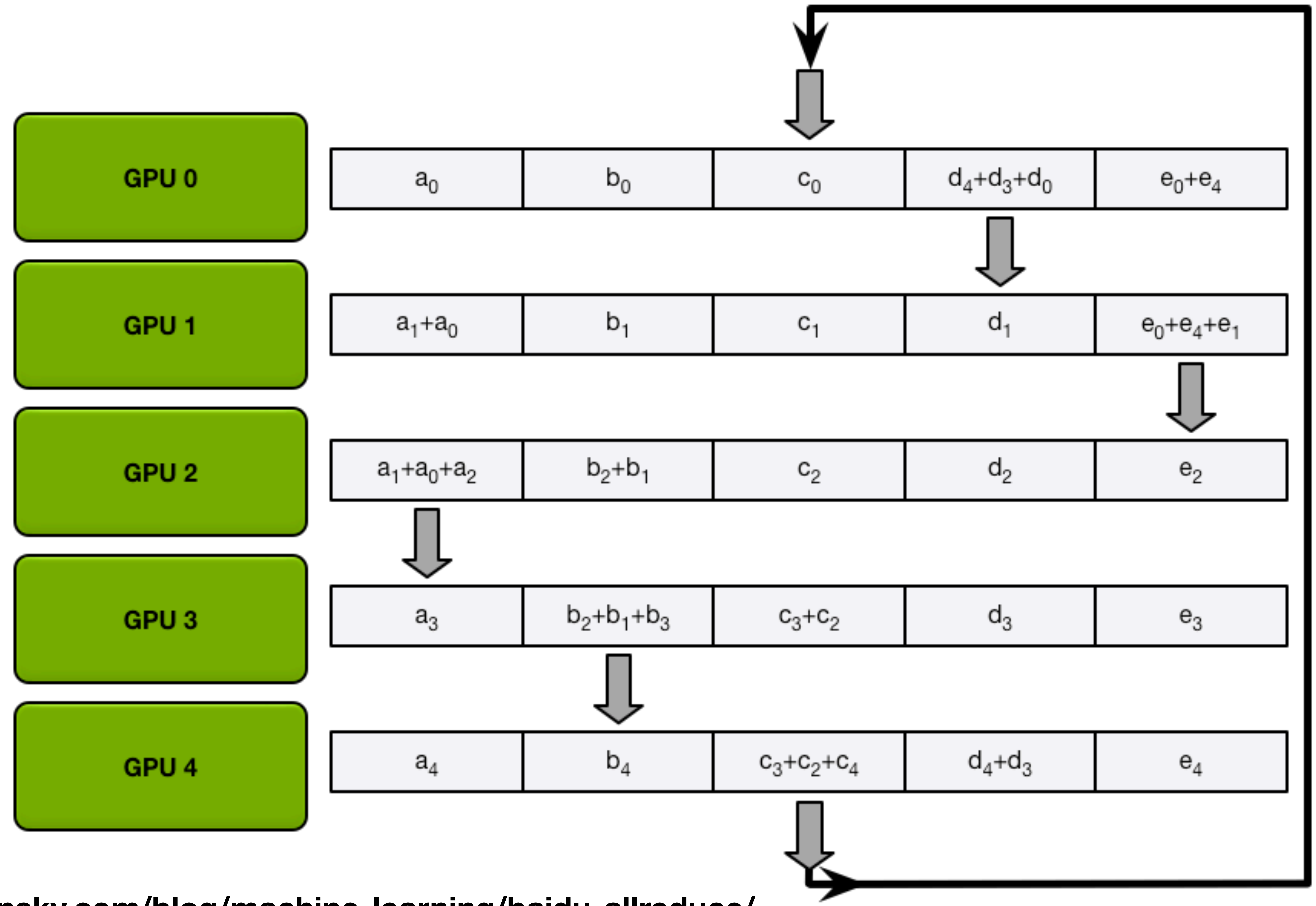
src: andrew.gibiansky.com/blog/machine-learning/baidu-allreduce/

Ring All-Reduce



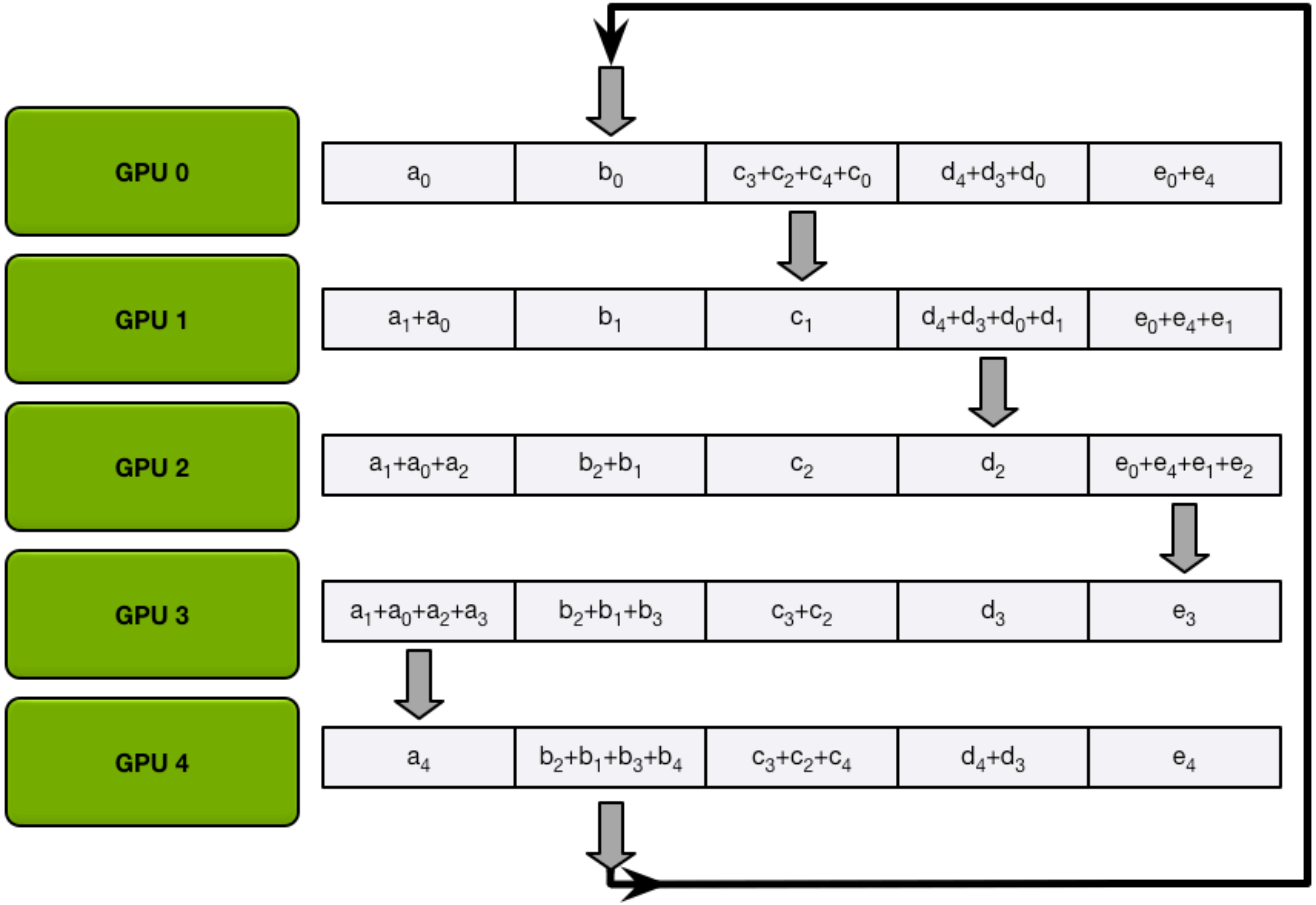
src: andrew.gibiansky.com/blog/machine-learning/baidu-allreduce/

Ring All-Reduce



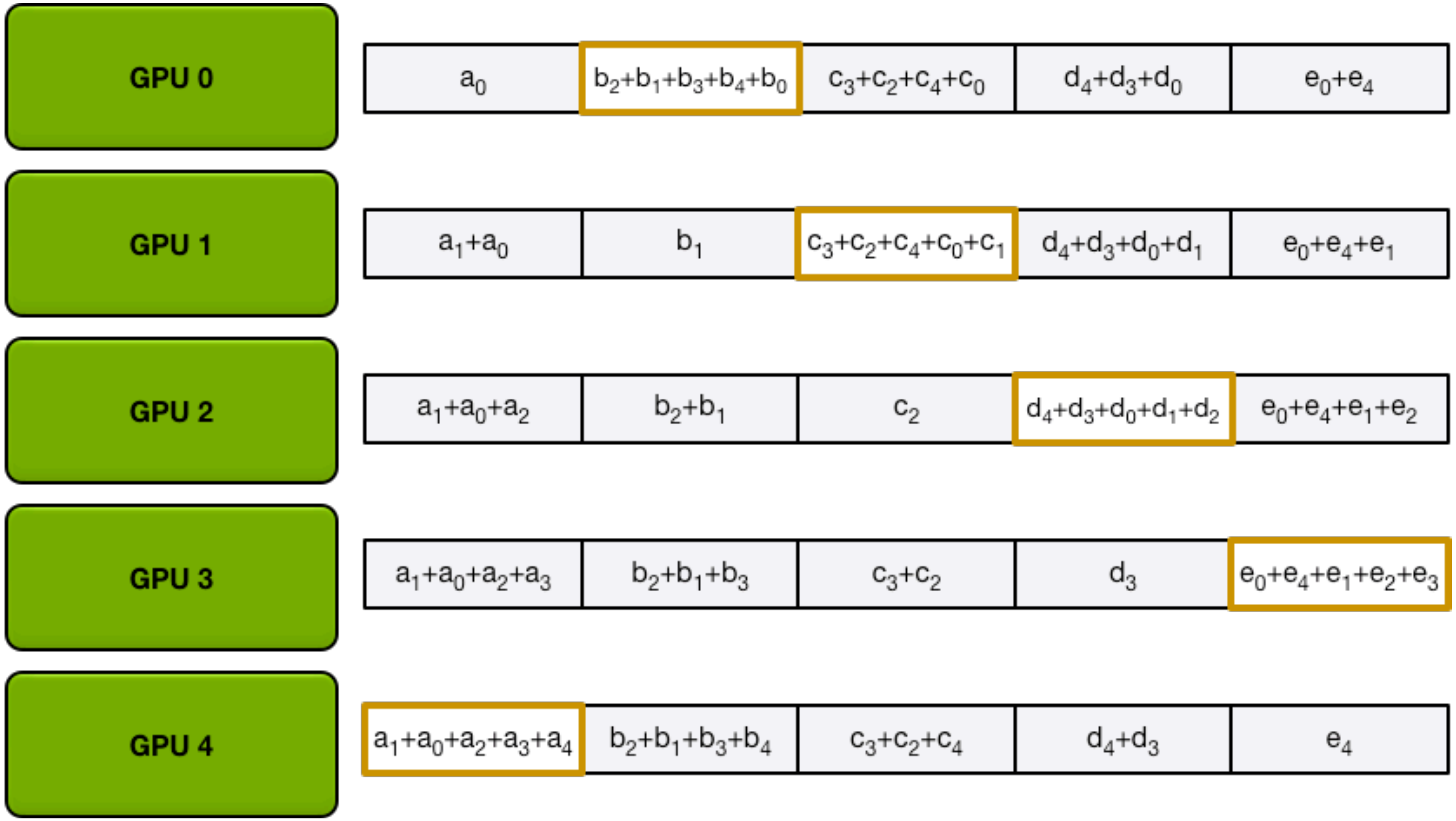
src: andrew.gibiansky.com/blog/machine-learning/baidu-allreduce/

Ring All-Reduce



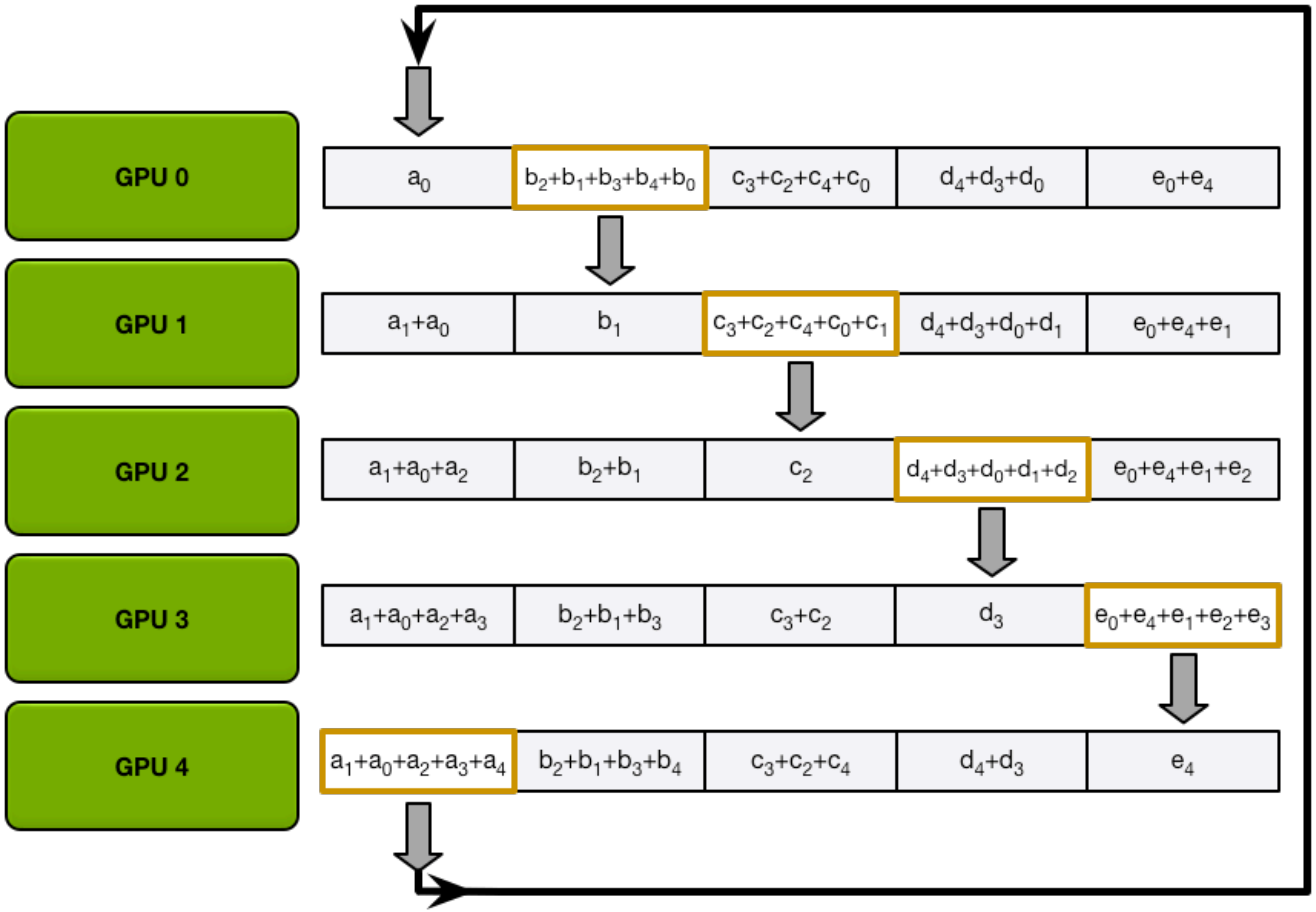
src: andrew.gibiansky.com/blog/machine-learning/baidu-allreduce/

Ring All-Reduce



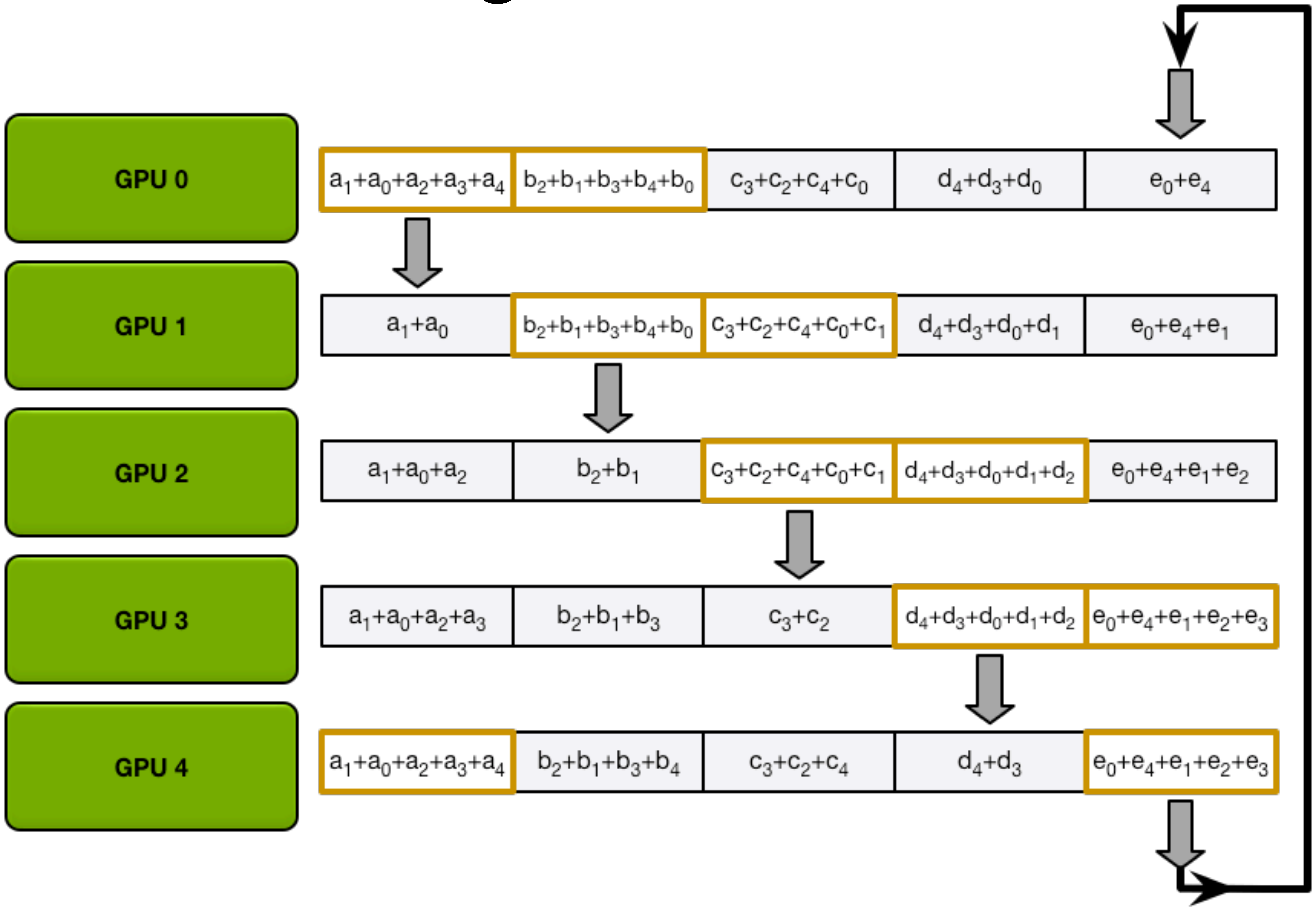
src: andrew.gibiansky.com/blog/machine-learning/baidu-allreduce/

Ring All-Reduce



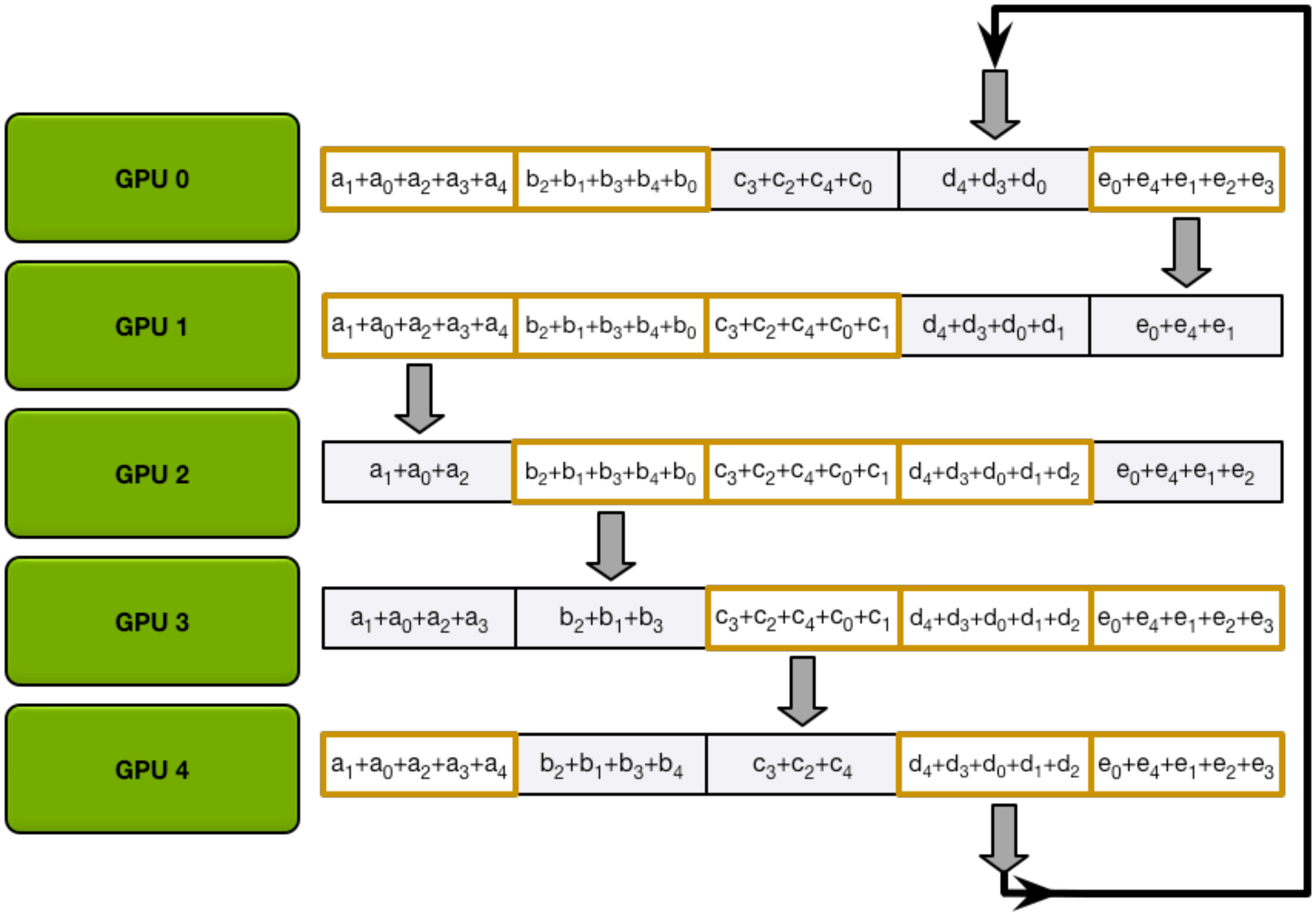
src: andrew.gibiansky.com/blog/machine-learning/baidu-allreduce/

Ring All-Reduce



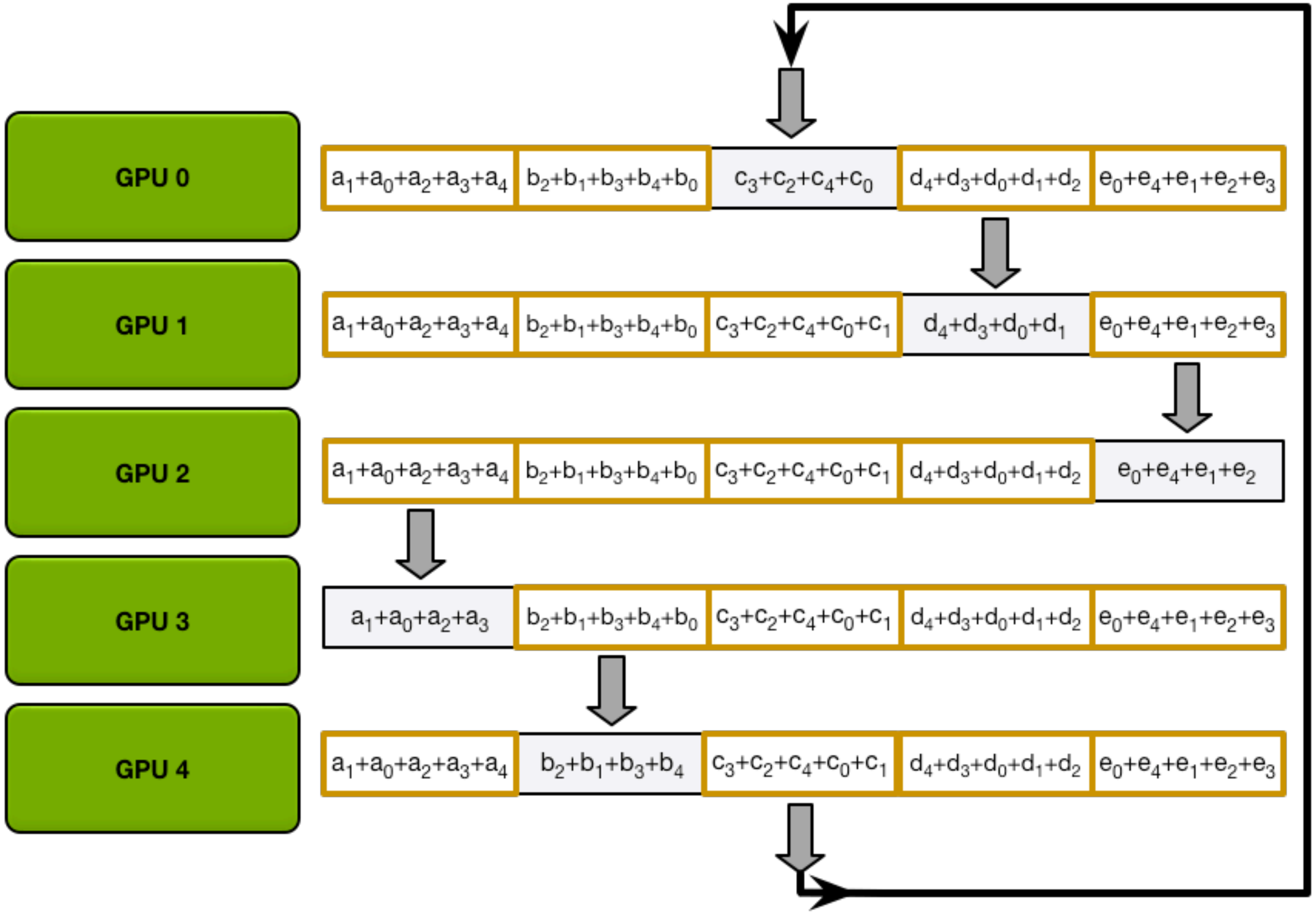
src: andrew.gibiansky.com/blog/machine-learning/baidu-allreduce/

Ring All-Reduce



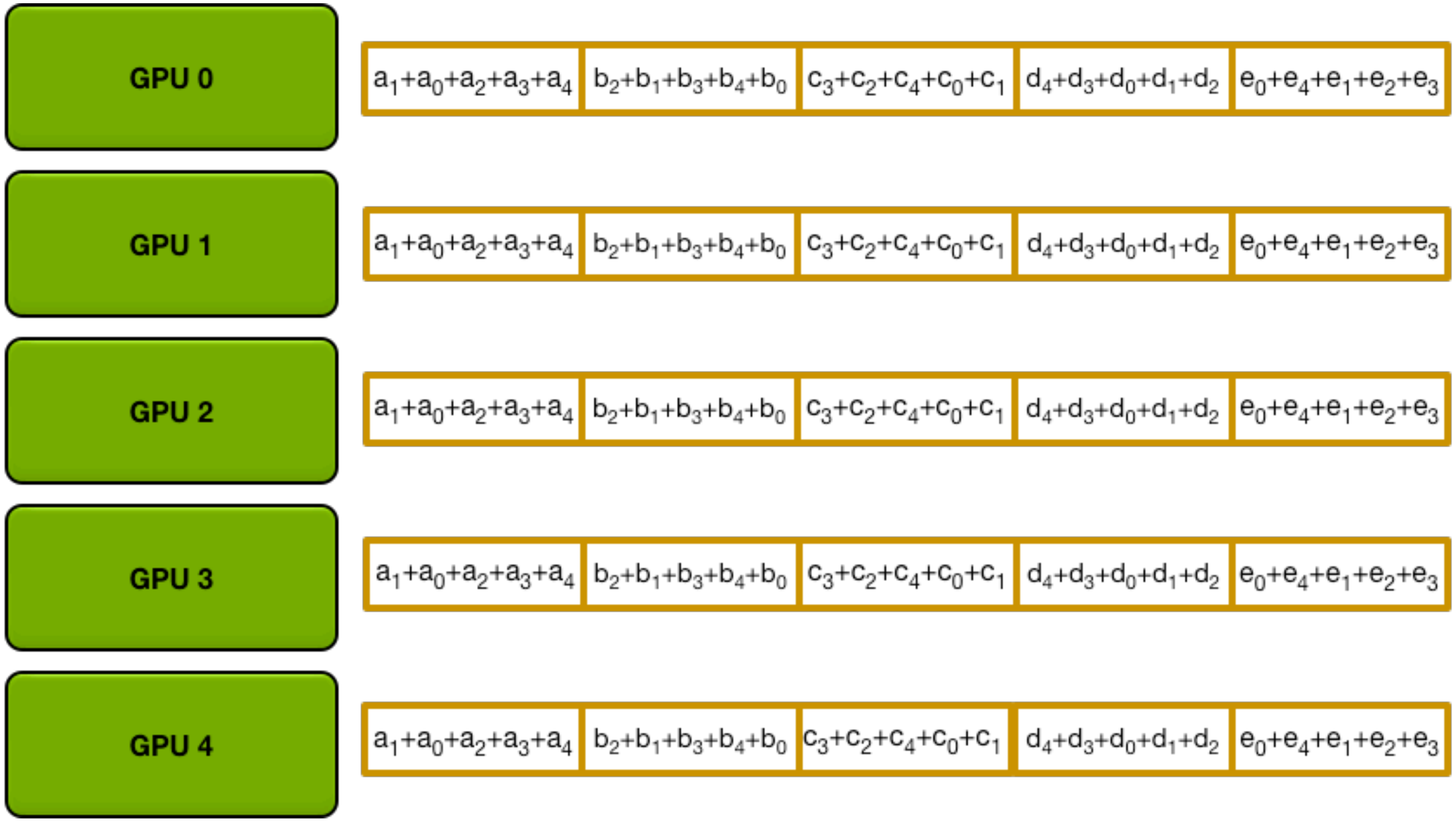
src: andrew.gibiansky.com/blog/machine-learning/baidu-allreduce/

Ring All-Reduce



src: andrew.gibiansky.com/blog/machine-learning/baidu-allreduce/

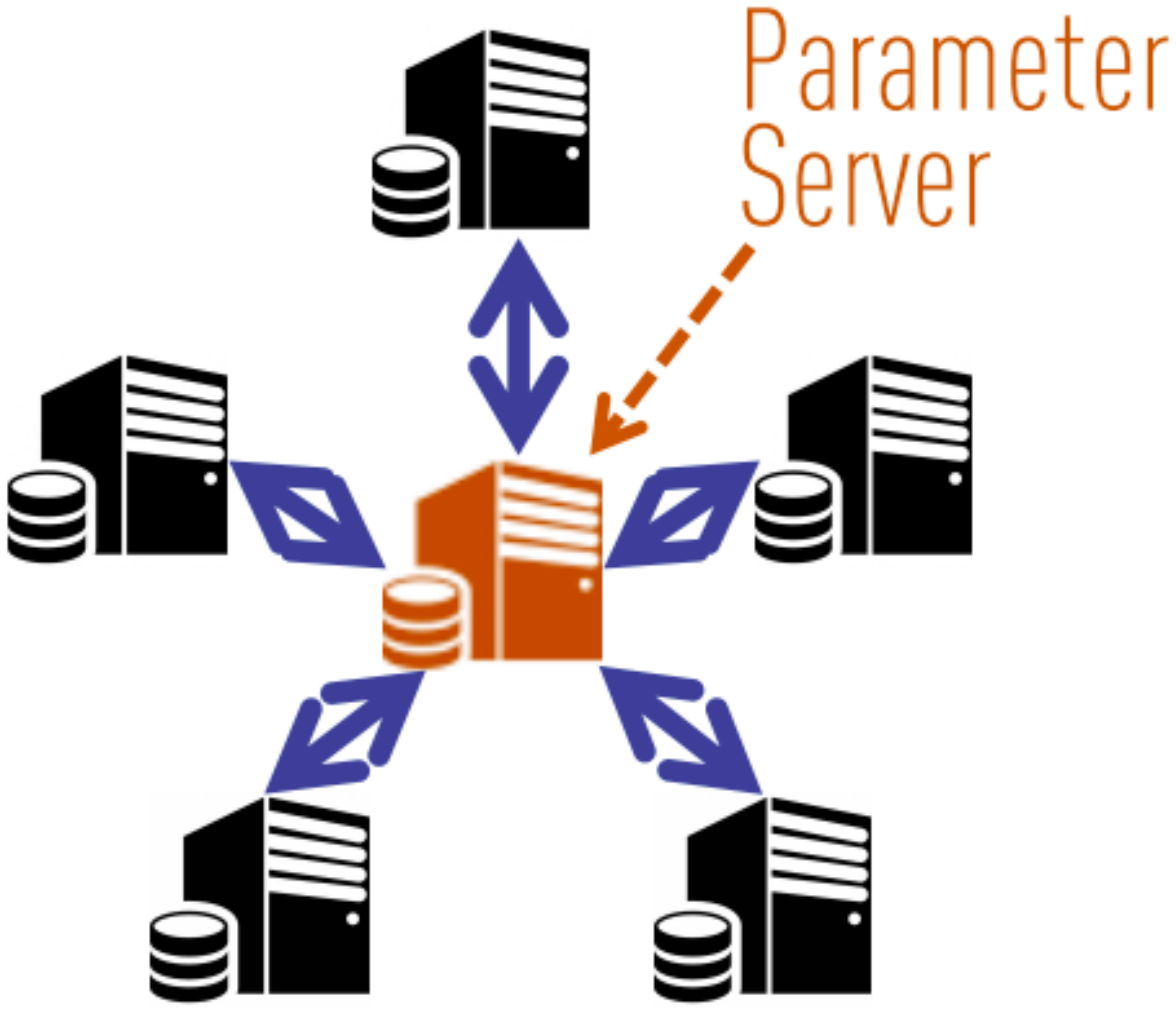
Ring All-Reduce



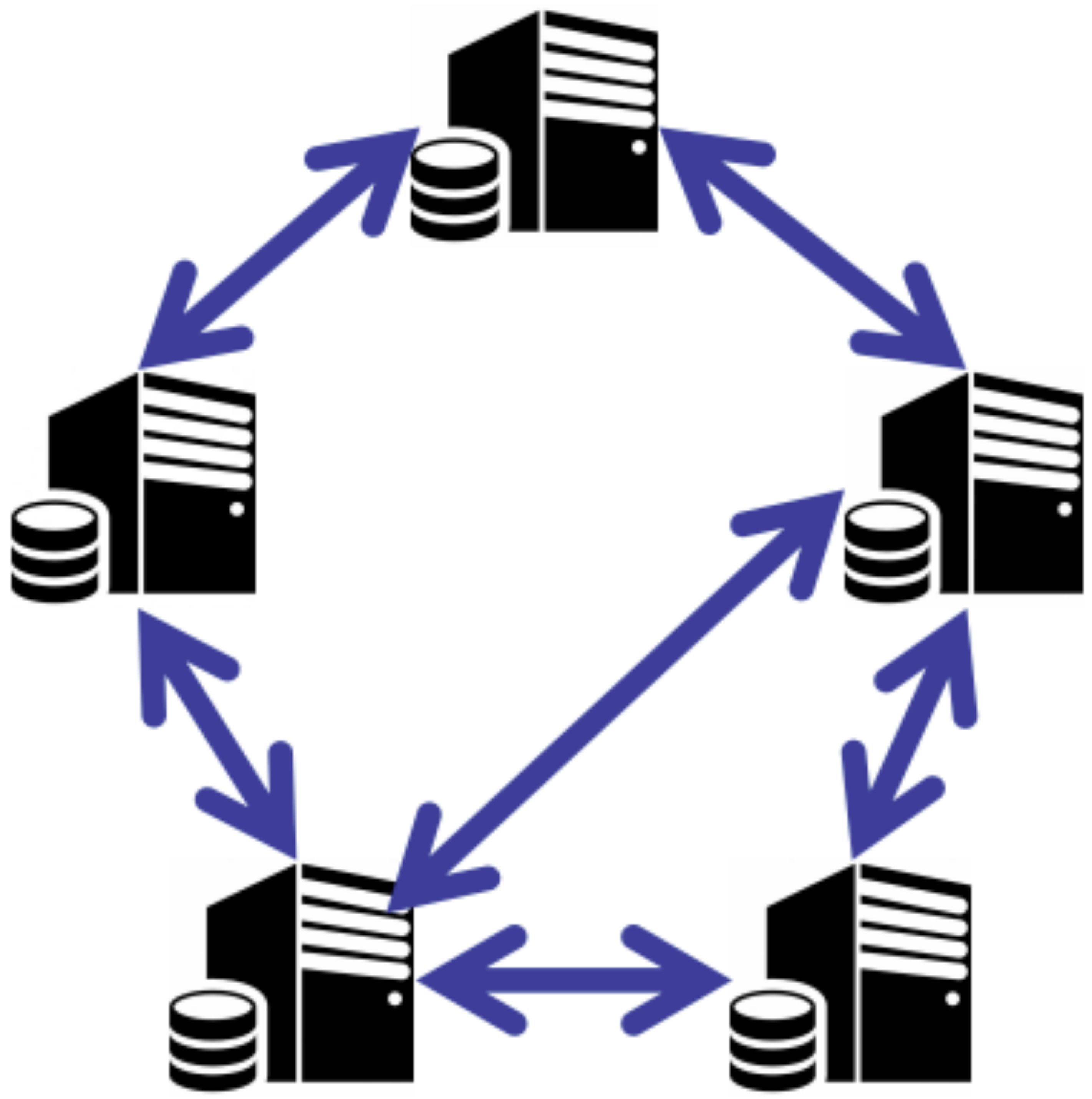
src: andrew.gibiansky.com/blog/machine-learning/baidu-allreduce/

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

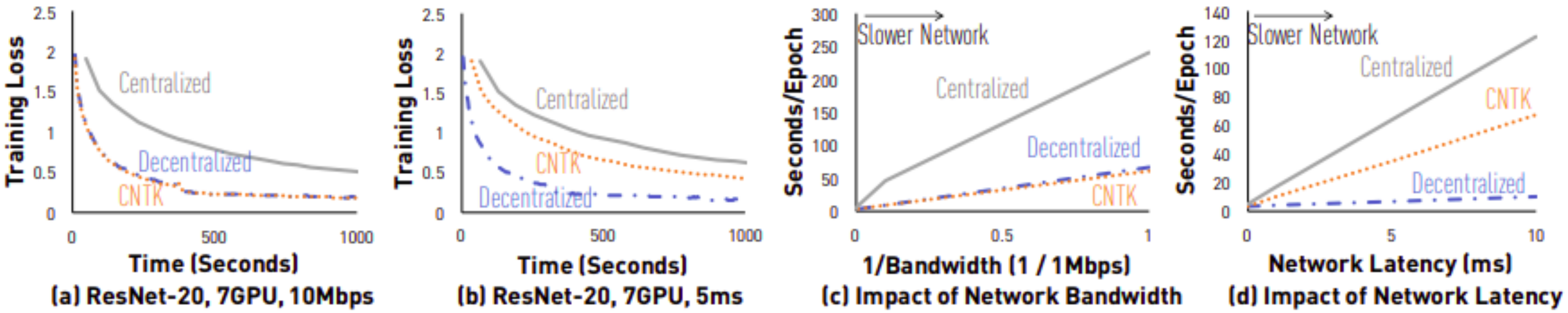


Figure 2: Comparison between D-PSGD and two centralized implementations (7 and 10 GPUs).

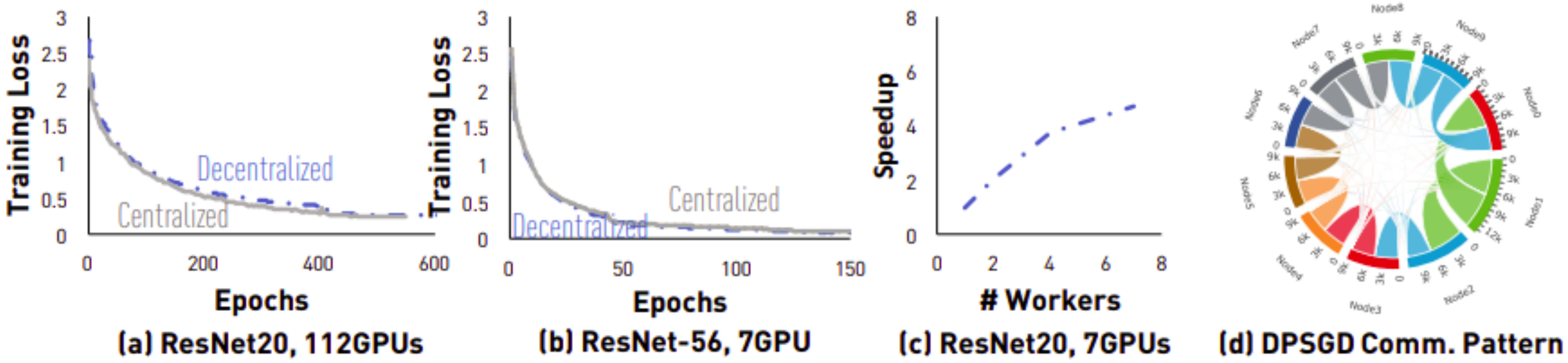
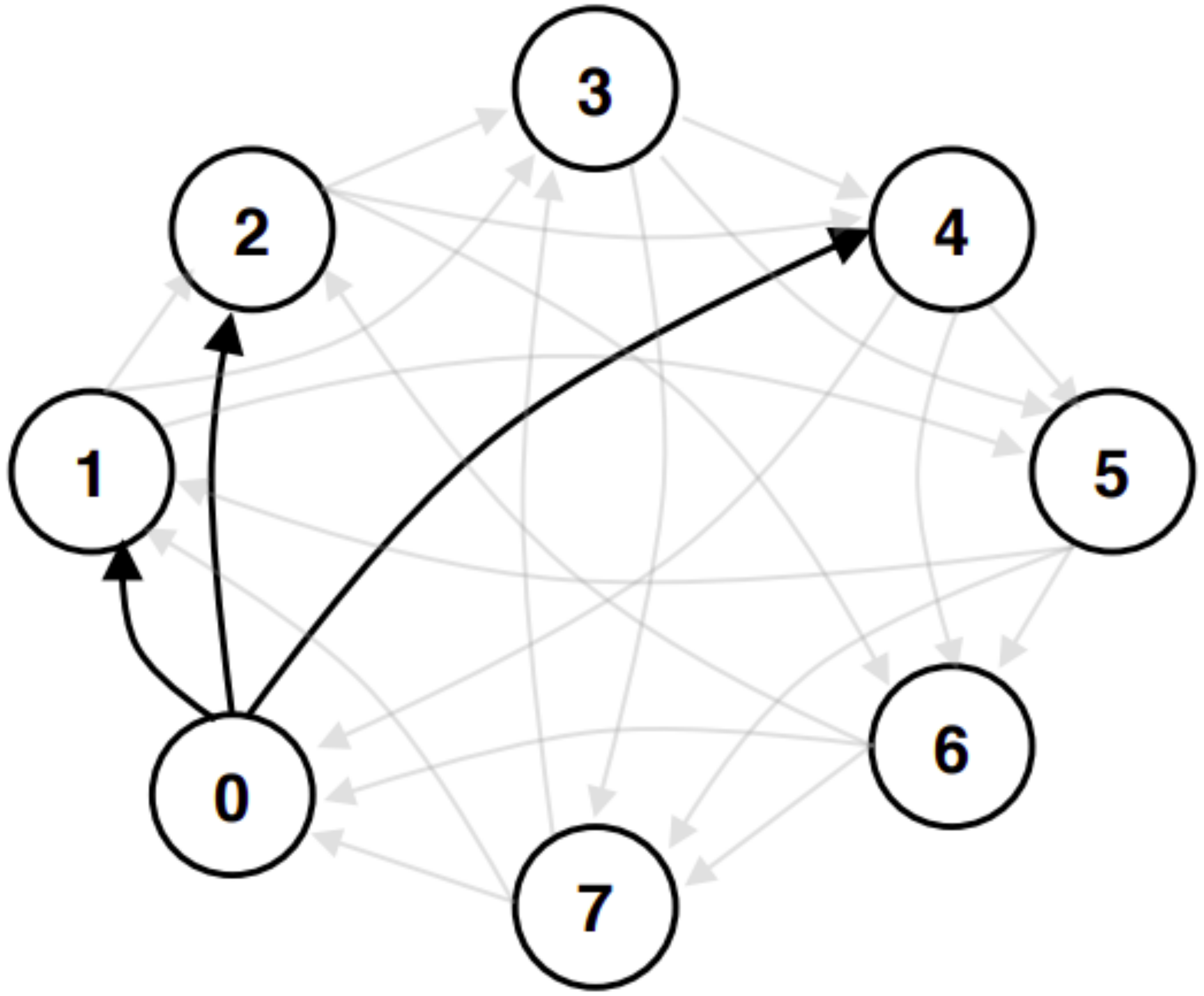


Figure 3: (a) Convergence Rate; (b) D-PSGD Speedup; (c) D-PSGD Communication Patterns.

Stochastic Gradient Push

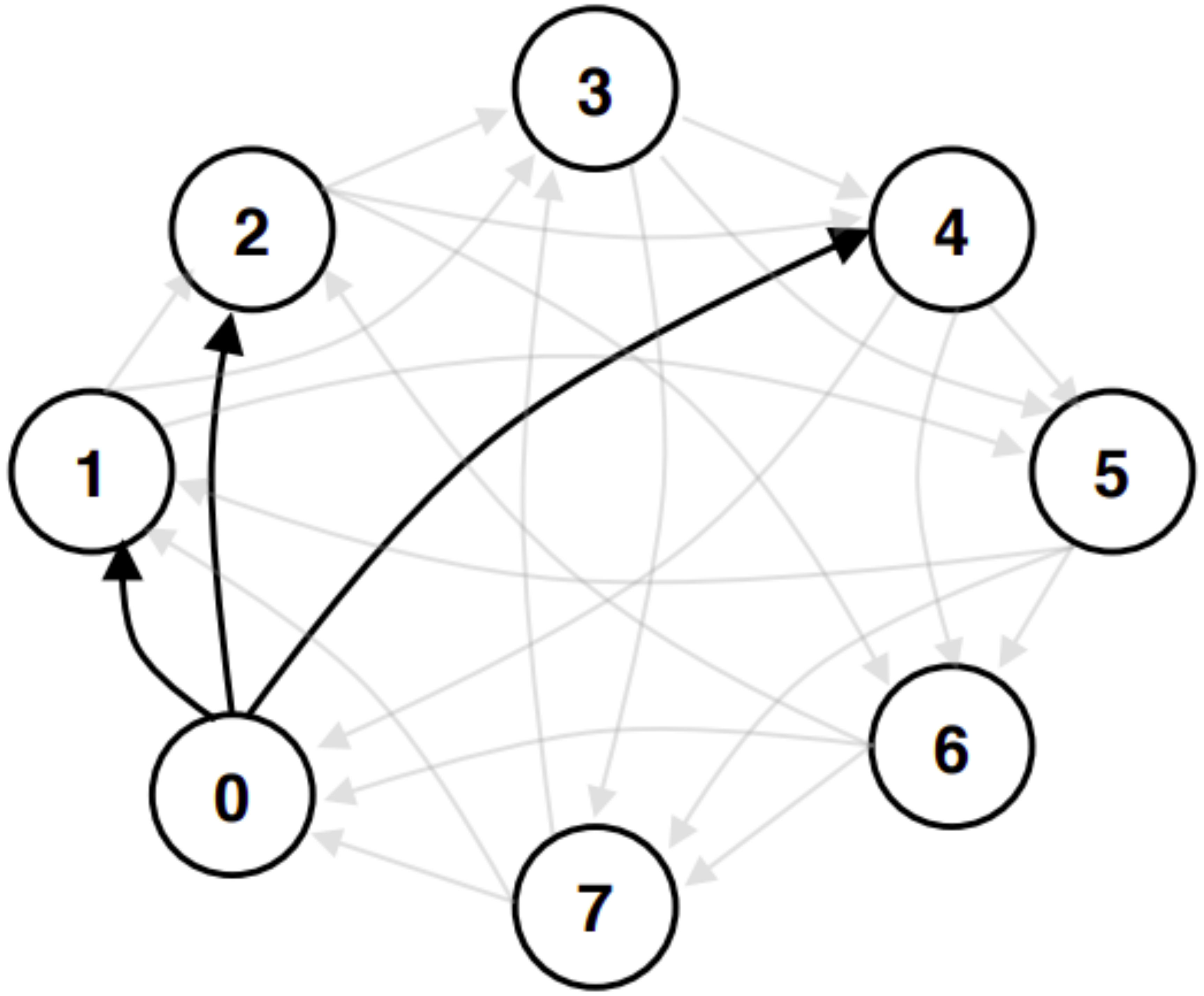
Source: https://arxiv.org/abs/1811.10792



(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**

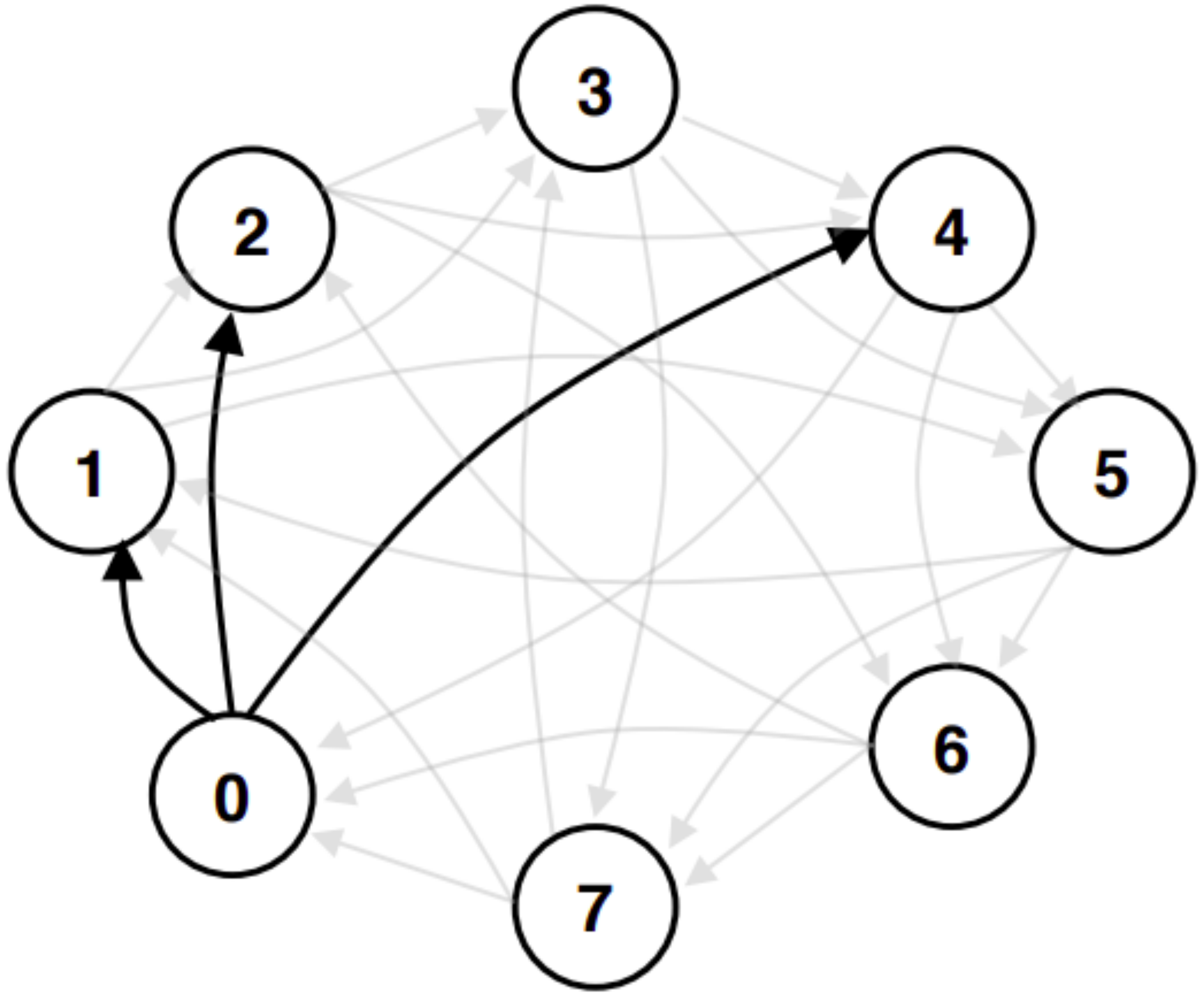
2: Sample new mini-batch $\xi_i^{(k)} \sim \mathcal{D}_i$ from local distribution

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 $\boldsymbol{z}_i^{(k)}$: $\nabla \boldsymbol{F}_i(\boldsymbol{z}_i^{(k)}; \xi_i^{(k)})$

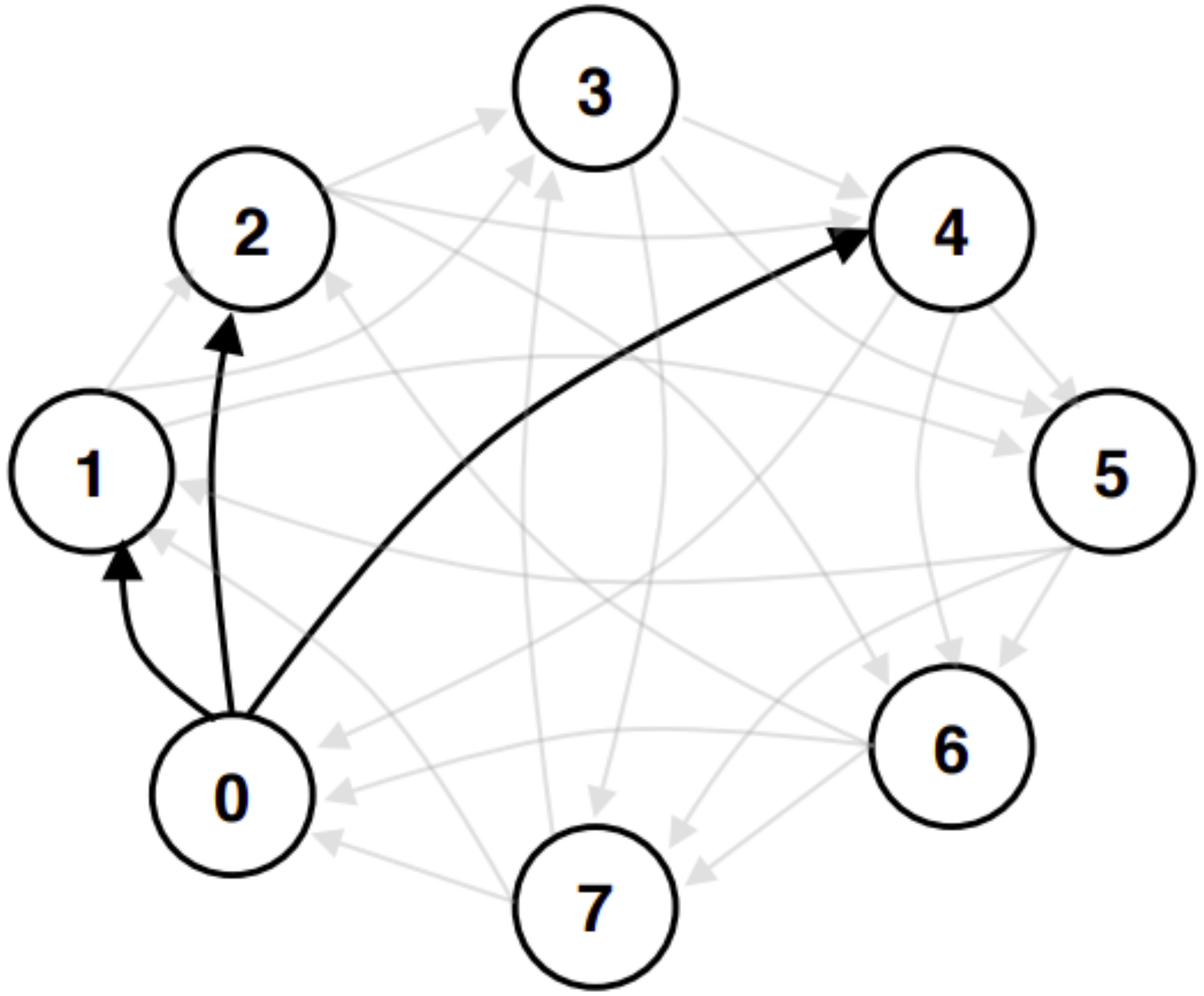
4: $\boldsymbol{x}_i^{(k+\frac{1}{2})} = \boldsymbol{x}_i^{(k)} - \gamma \nabla \boldsymbol{F}_i(\boldsymbol{z}_i^{(k)}; \xi_i^{(k)})$

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 $i \in \{1, 2, \ldots, n\}$

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 $\boldsymbol{z}_i^{(k)}$: $\nabla \boldsymbol{F}_i(\boldsymbol{z}_i^{(k)}; \xi_i^{(k)})$

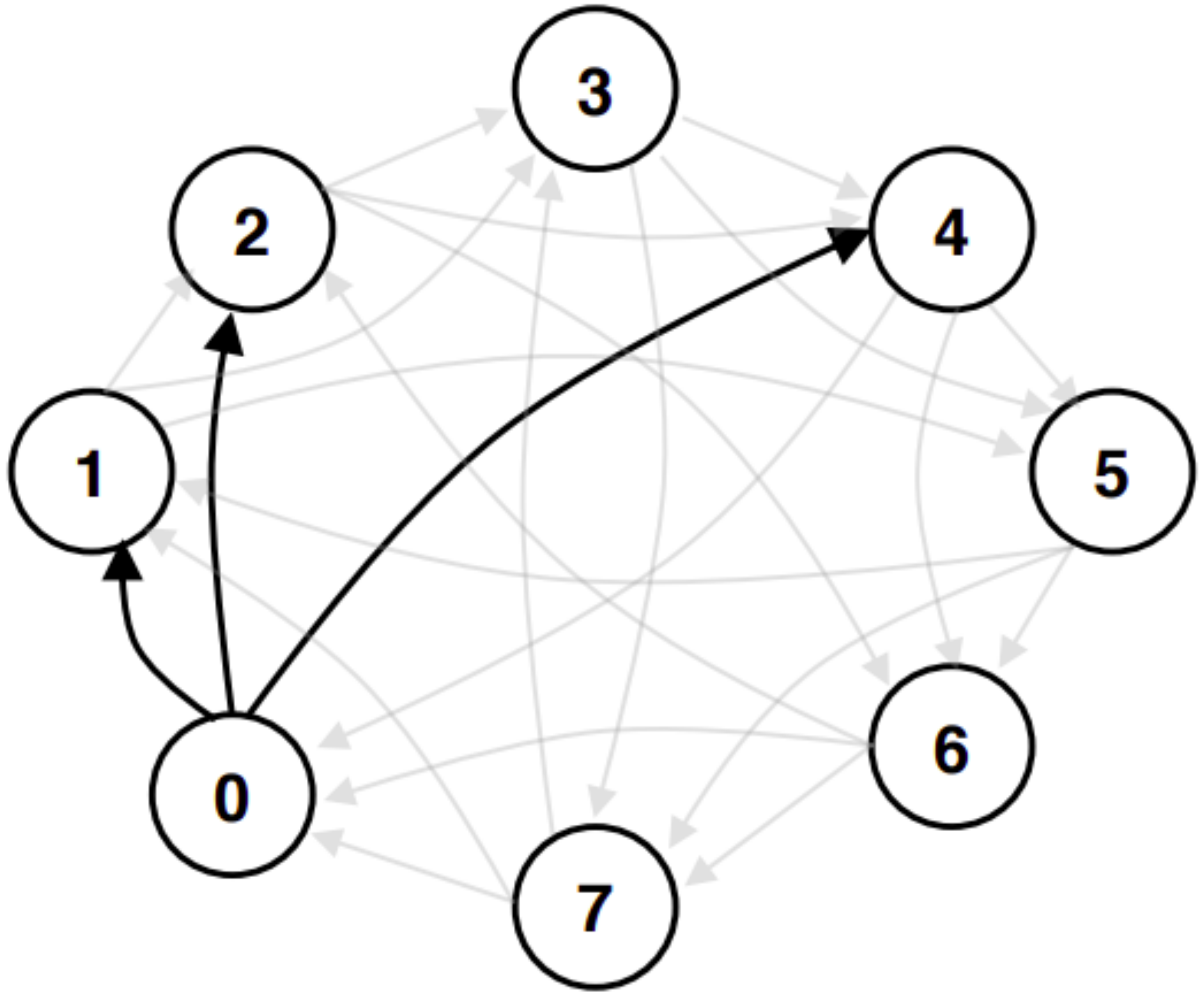
$\boldsymbol{x}_i^{(k+\frac{1}{2})} = \boldsymbol{x}_i^{(k)} - \gamma \nabla \boldsymbol{F}_i(\boldsymbol{z}_i^{(k)}; \xi_i^{(k)})$

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

<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

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

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

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

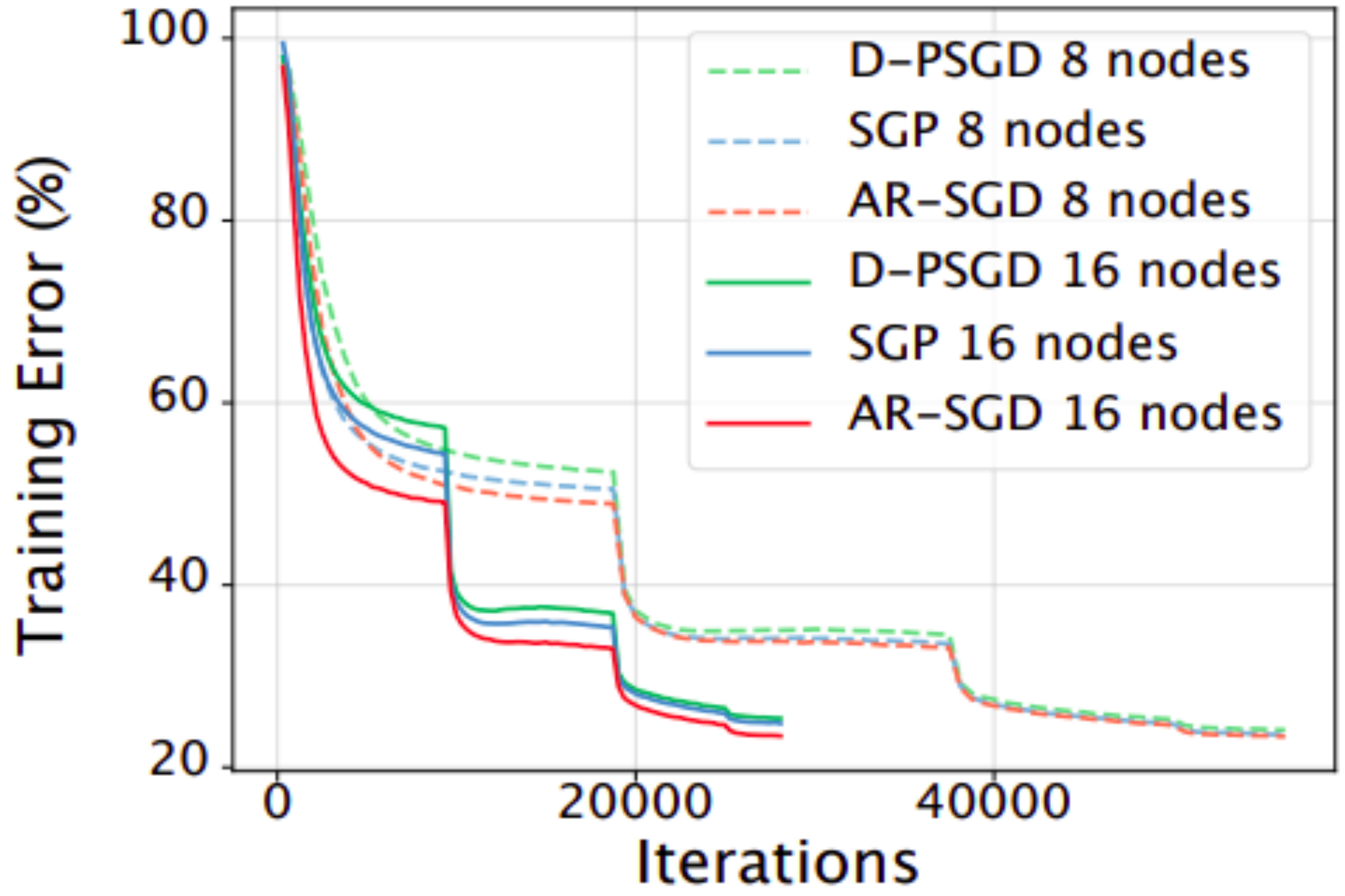
end for

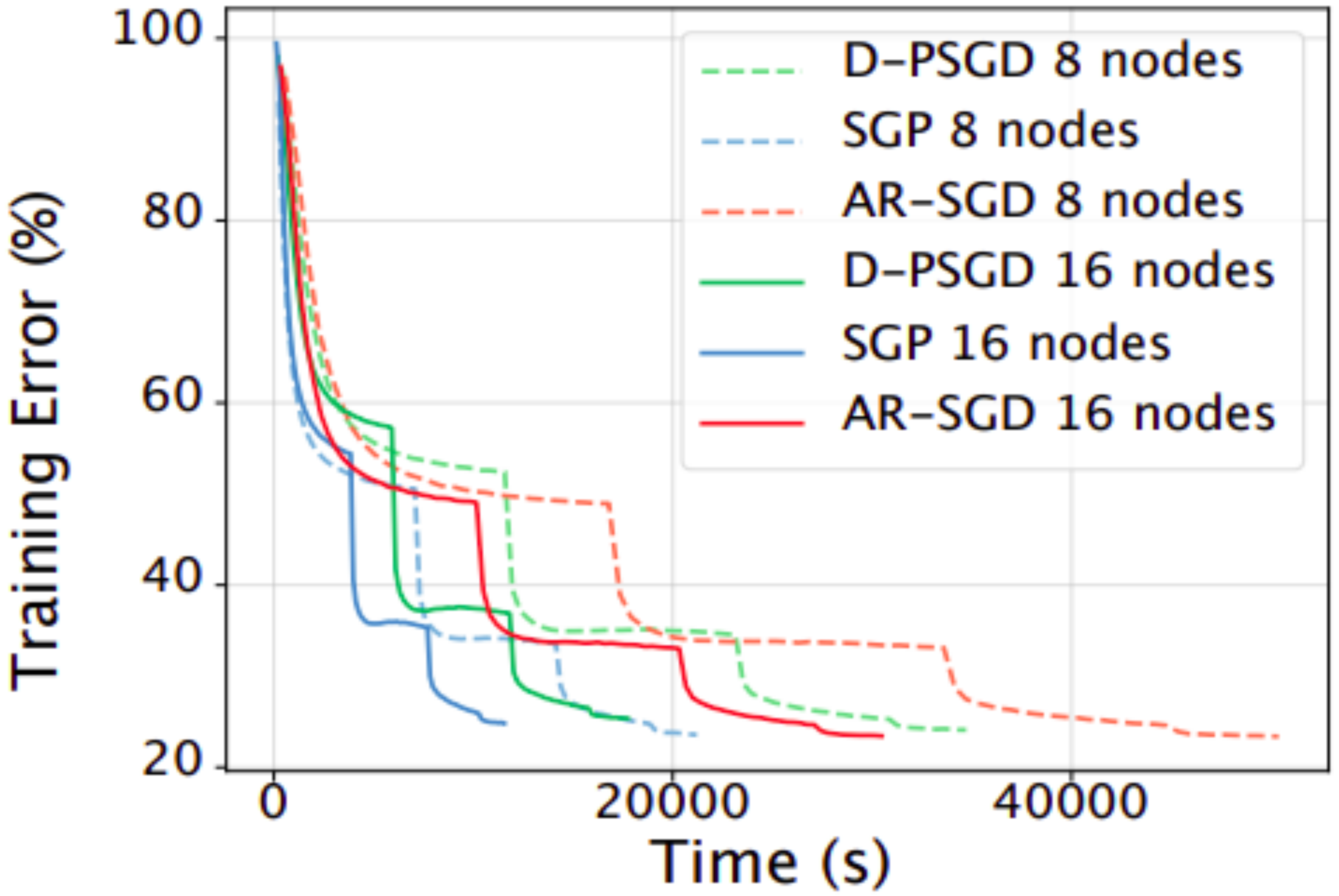
weighted average

Stochastic Gradient Push

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

SGP vs ImageNet (ResNet50 + SGD w/ momentum)

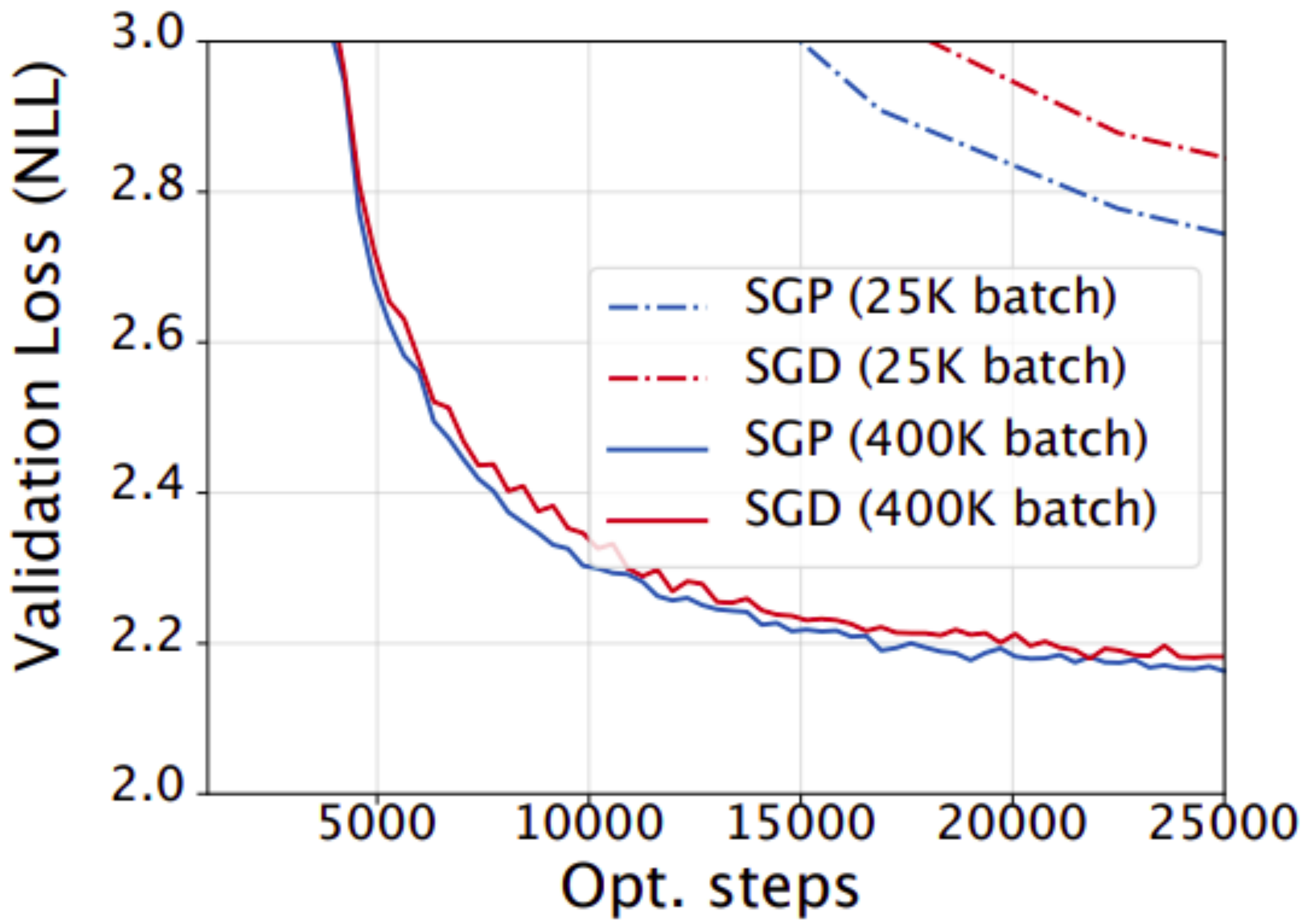


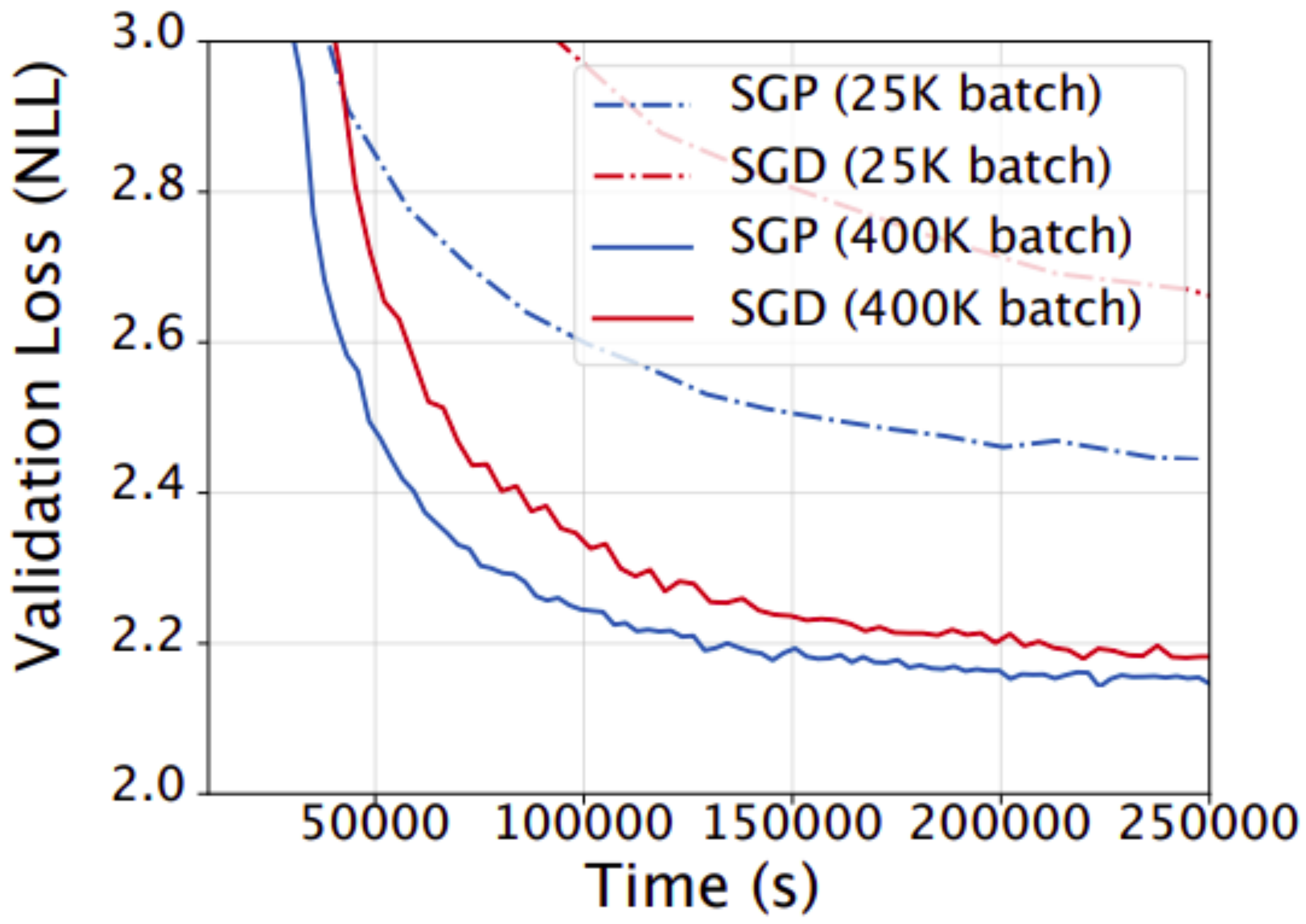


Stochastic Gradient Push

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

SGP vs WMT English-German (Transformer, Adam)



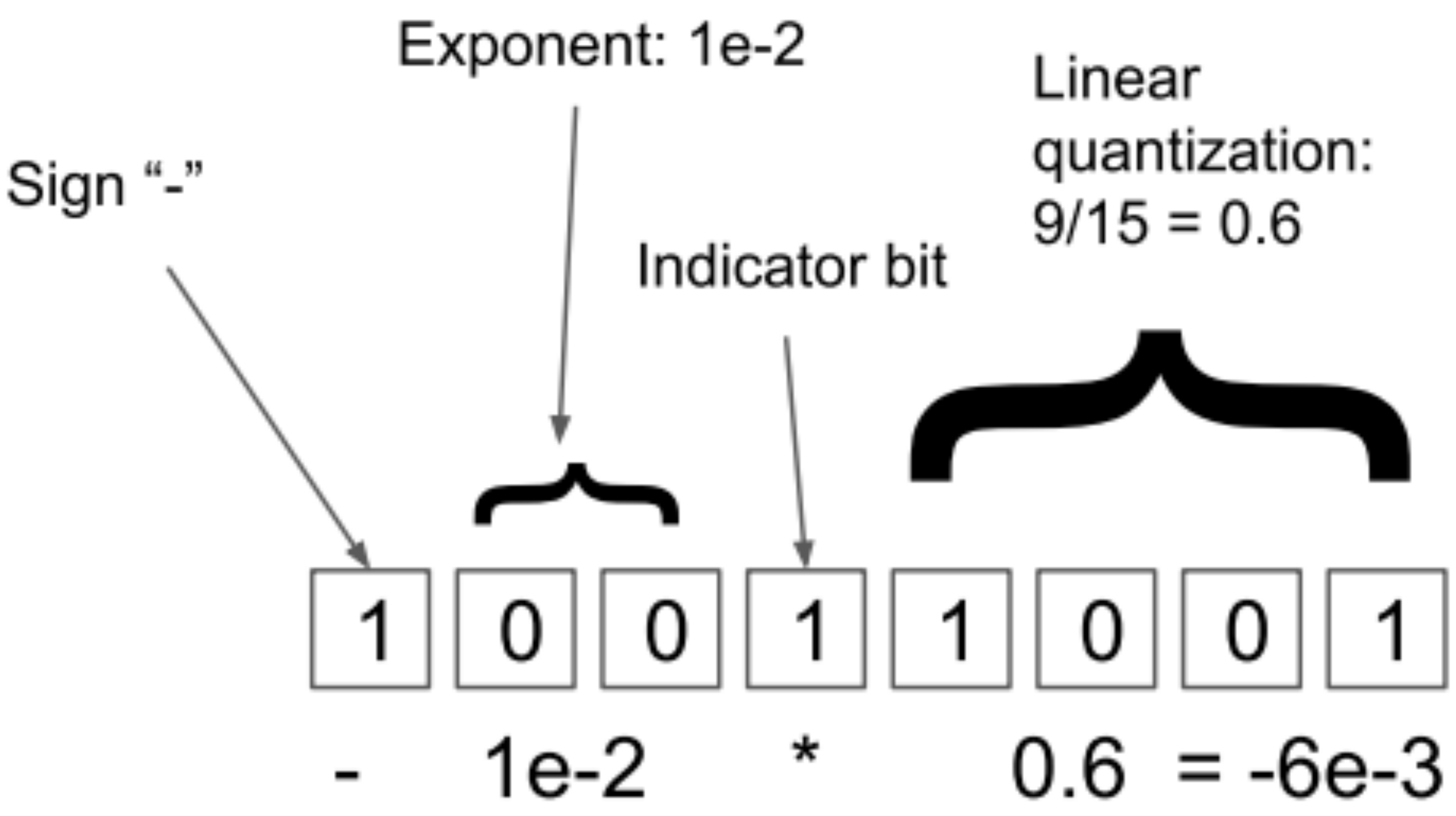


Communication Efficiency

Your thoughts?

Quantization

https://arxiv.org/abs/1511.04561



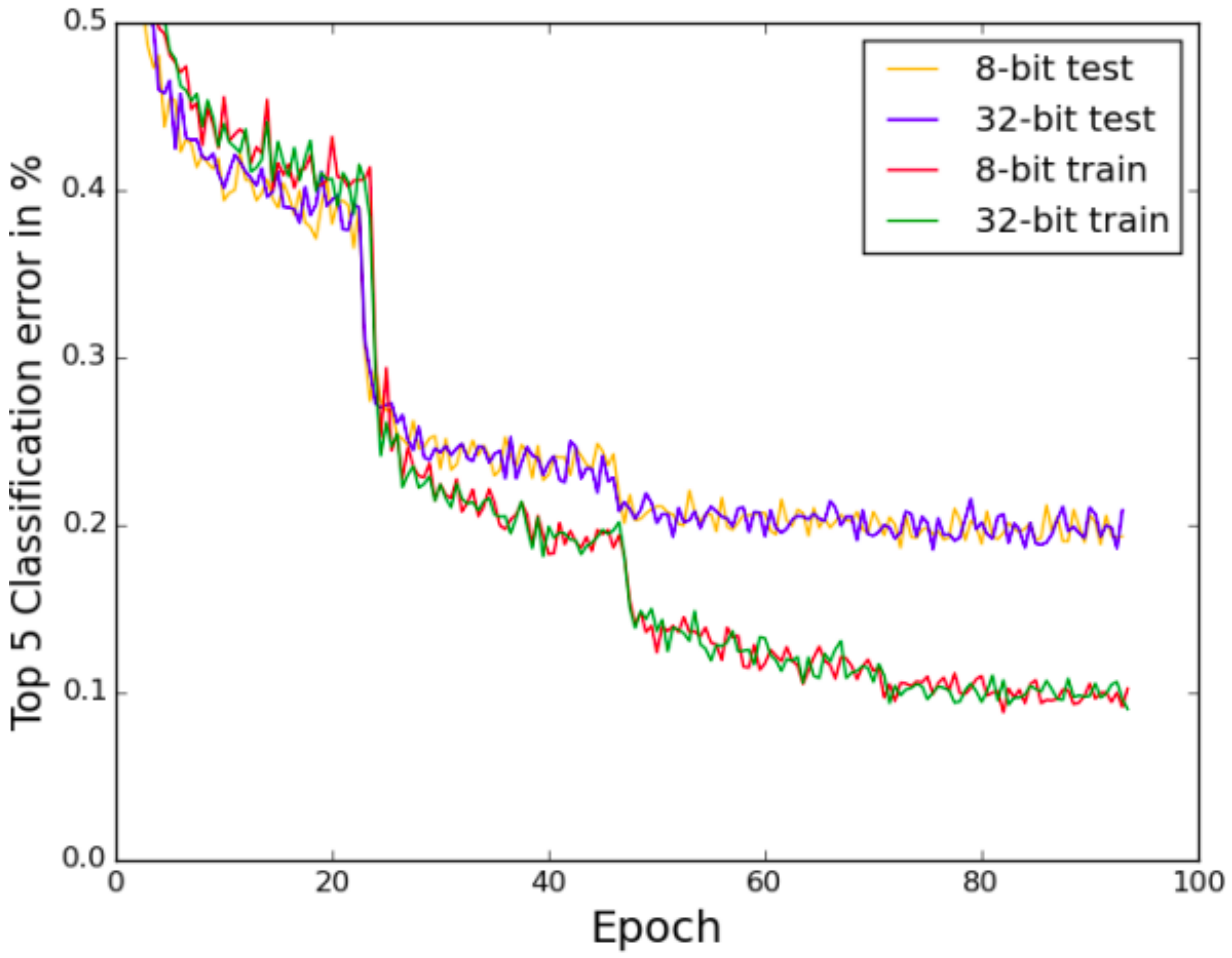
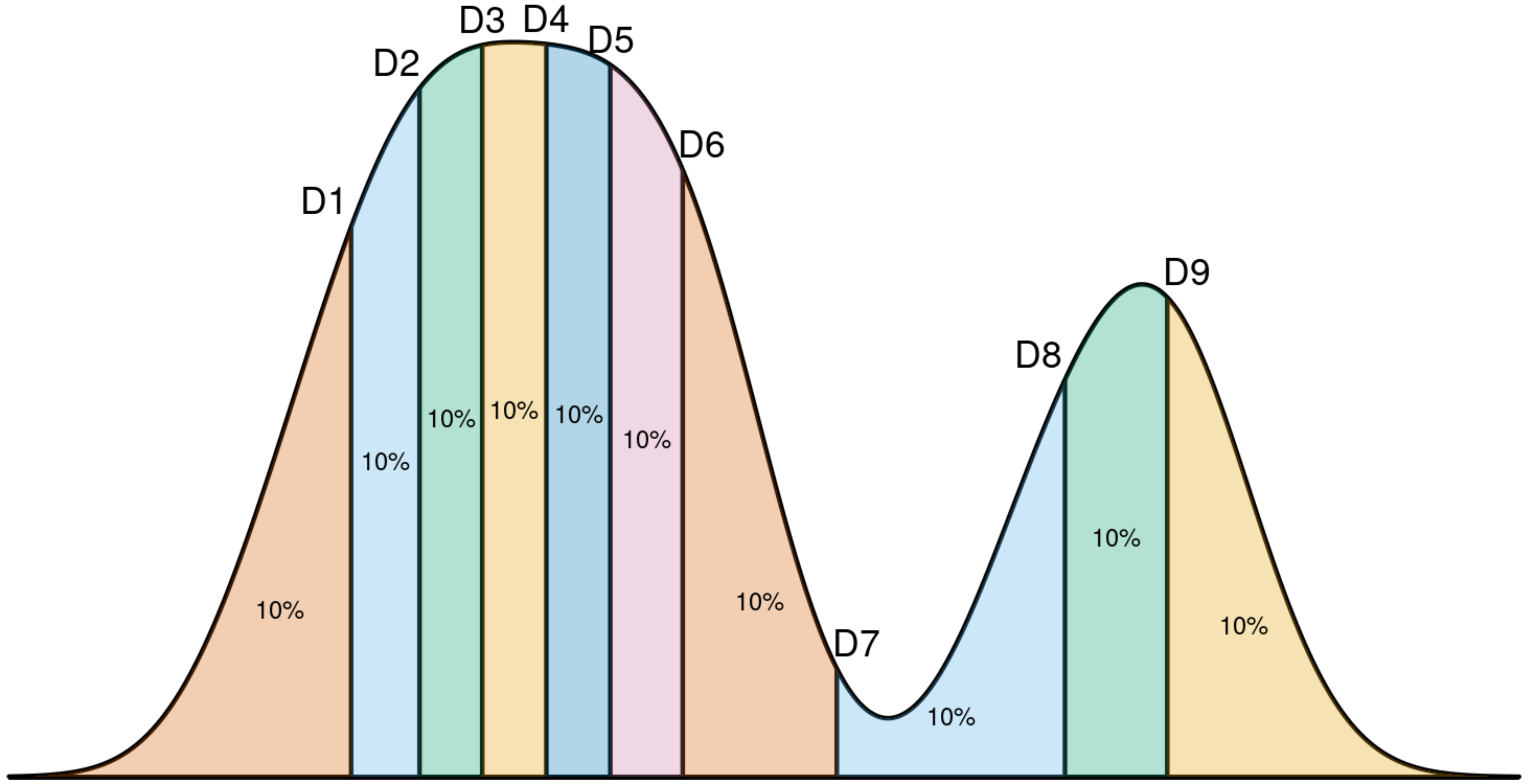
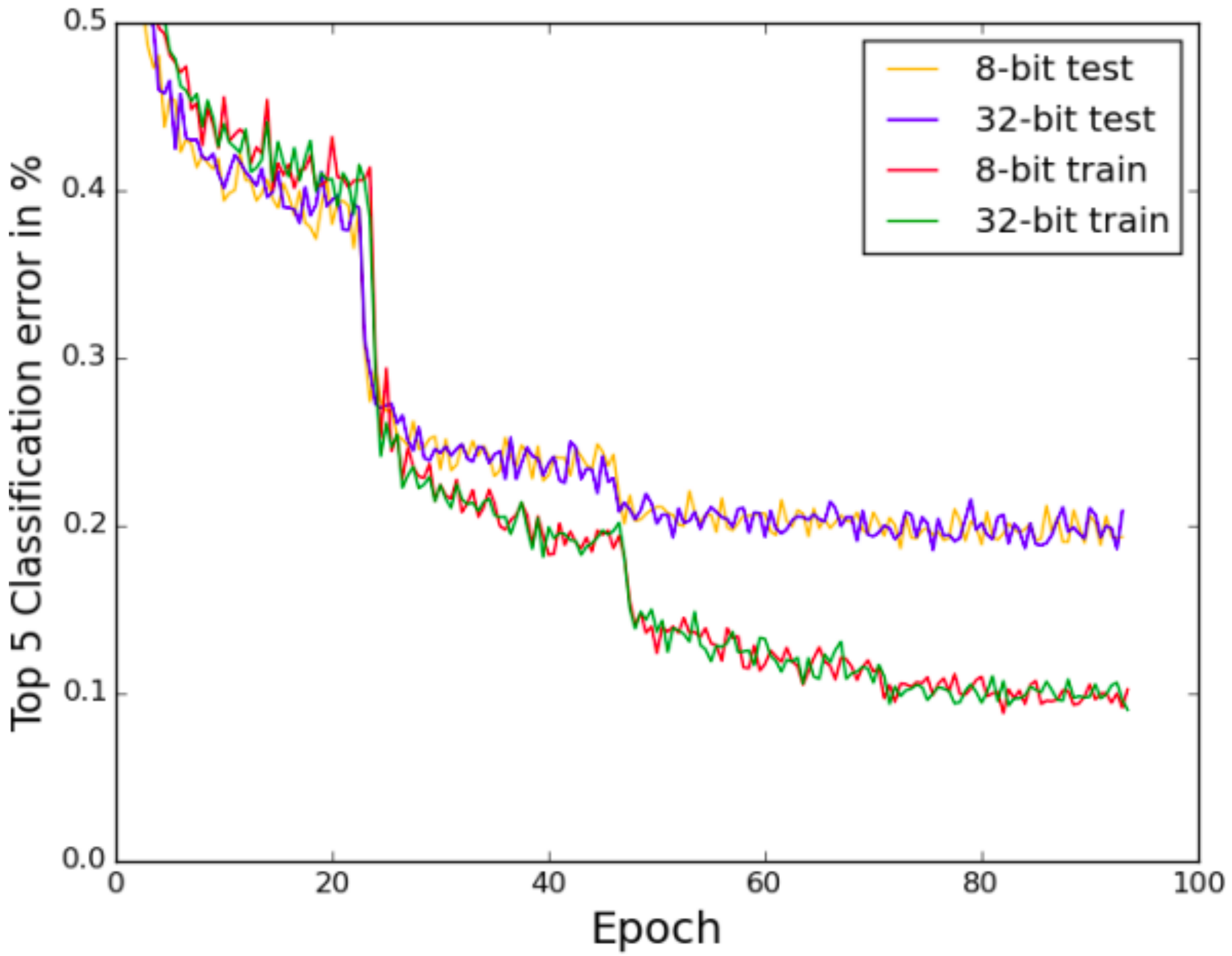


Image source: https://arxiv.org/pdf/2110.02861.pdf

Quantization

https://arxiv.org/abs/1511.04561





Biased Compression

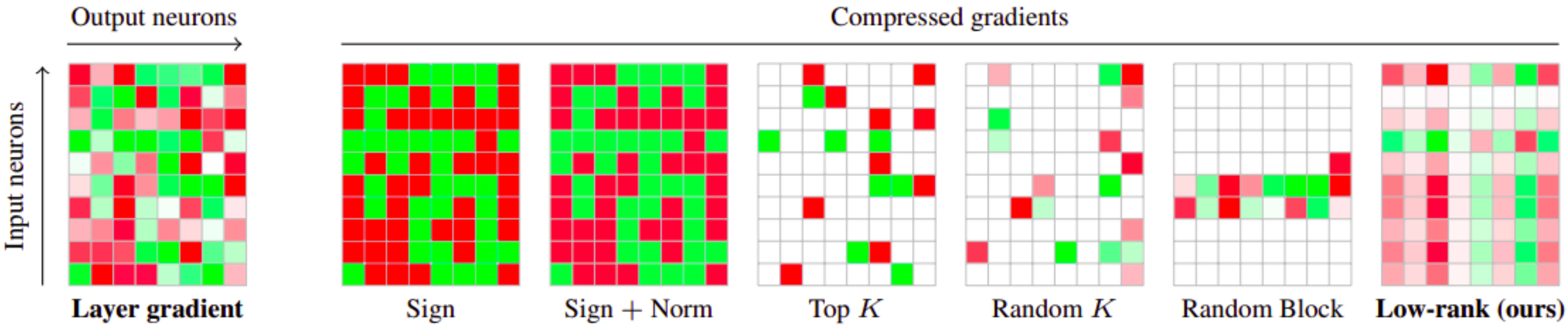
https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/IS140694.pdf

Algorithm 2 Distributed Error-feedback SGD with Momentum

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 $w = 1, \dots, W$ **do**
  initialize memory $\mathbf{e}_w \leftarrow \mathbf{0} \in \mathbb{R}^d$
  for each iterate $t = 0, \dots$ **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
    $\mathcal{C}(\Delta_w) \leftarrow \text{COMPRESS}(\Delta_w)$
    $\mathbf{e}_w \leftarrow \Delta_w - \text{DECOMPRESS}(\mathcal{C}(\Delta_w))$ ▷ Memorize local errors
    $\mathcal{C}(\Delta) \leftarrow \text{AGGREGATE}(\mathcal{C}(\Delta_1), \dots, \mathcal{C}(\Delta_W))$ ▷ Exchange gradients
    $\Delta' \leftarrow \text{DECOMPRESS}(\mathcal{C}(\Delta))$ ▷ 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

Biased Compression

https://arxiv.org/abs/1905.13727



Biased Compression

https://arxiv.org/abs/1905.13727

Algorithm 1 Rank-r POWERSGD compression

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

Moshpit SGD: Communication-Efficient Decentralized Training on Heterogeneous Unreliable Devices

Max Ryabinin*
Yandex, Russia
HSE University, Russia

Eduard Gorbunov*
MIPT, Russia
HSE University, Russia
Yandex, Russia

Vsevolod Plokhotnyuk
Yandex, Russia
HSE University, Russia

Gennady Pekhimenko
University of Toronto, Canada
Vector Institute, Canada

Moshpit SGD: Communication-Efficient Decentralized Training on Heterogeneous Unreliable Devices

Max Ryabinin*
Yandex, Russia
HSE University, Russia

Eduard Gorbunov*
MIPT, Russia
HSE University, Russia
Yandex, Russia

Vsevolod Plokhotnyuk
Yandex, Russia
HSE University, Russia

Gennady Pekhimenko
University of Toronto, Canada
Vector Institute, Canada

- We propose a new algorithm for decentralized AllReduce-like averaging

Moshpit SGD: Communication-Efficient Decentralized Training on Heterogeneous Unreliable Devices

Max Ryabinin*
Yandex, Russia
HSE University, Russia

Eduard Gorbunov*
MIPT, Russia
HSE University, Russia
Yandex, Russia

Vsevolod Plokhotnyuk
Yandex, Russia
HSE University, Russia

Gennady Pekhimenko
University of Toronto, Canada
Vector Institute, Canada

- We propose a new algorithm for decentralized AllReduce-like averaging
- Main idea: average in smaller non-overlapping groups

Moshpit SGD: Communication-Efficient Decentralized Training on Heterogeneous Unreliable Devices

Max Ryabinin*
Yandex, Russia
HSE University, Russia

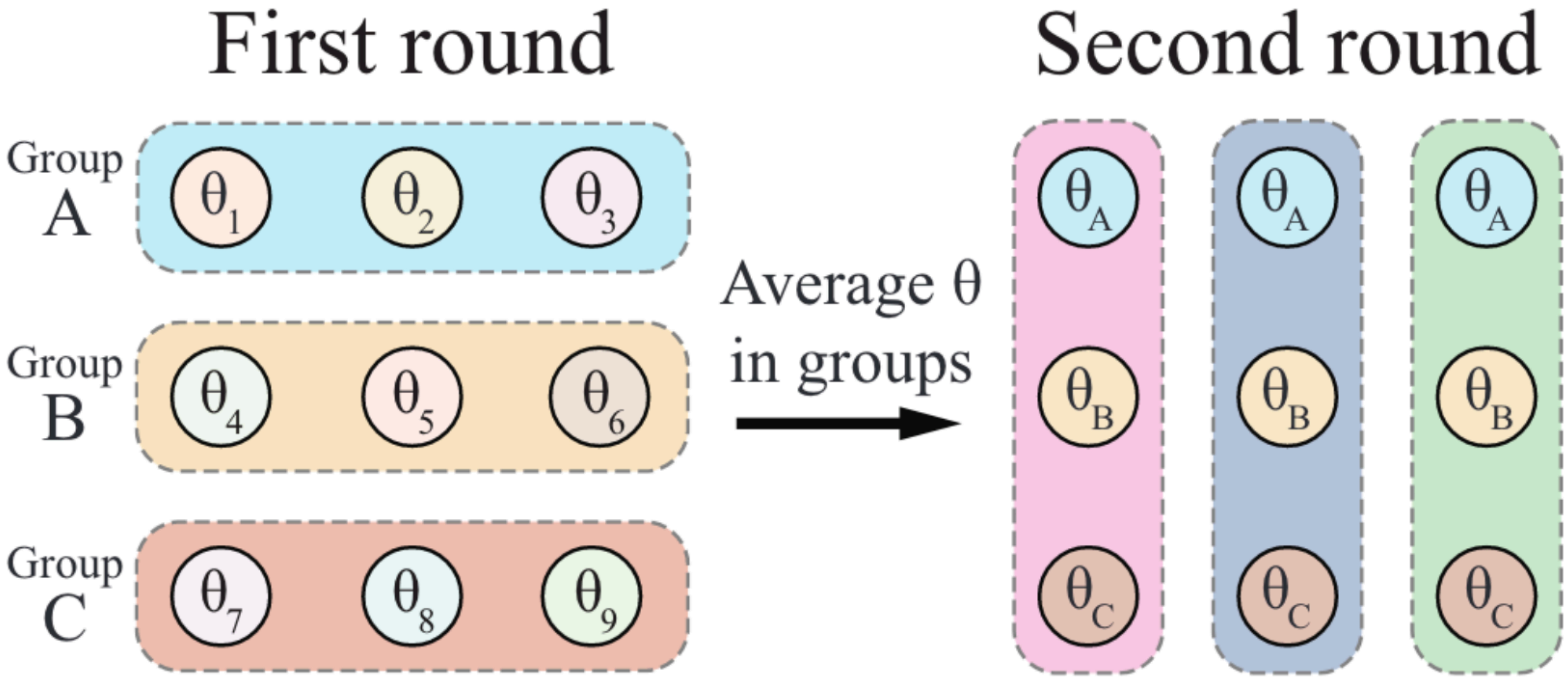
Eduard Gorbunov*
MIPT, Russia
HSE University, Russia
Yandex, Russia

Vsevolod Plokhotnyuk
Yandex, Russia
HSE University, Russia

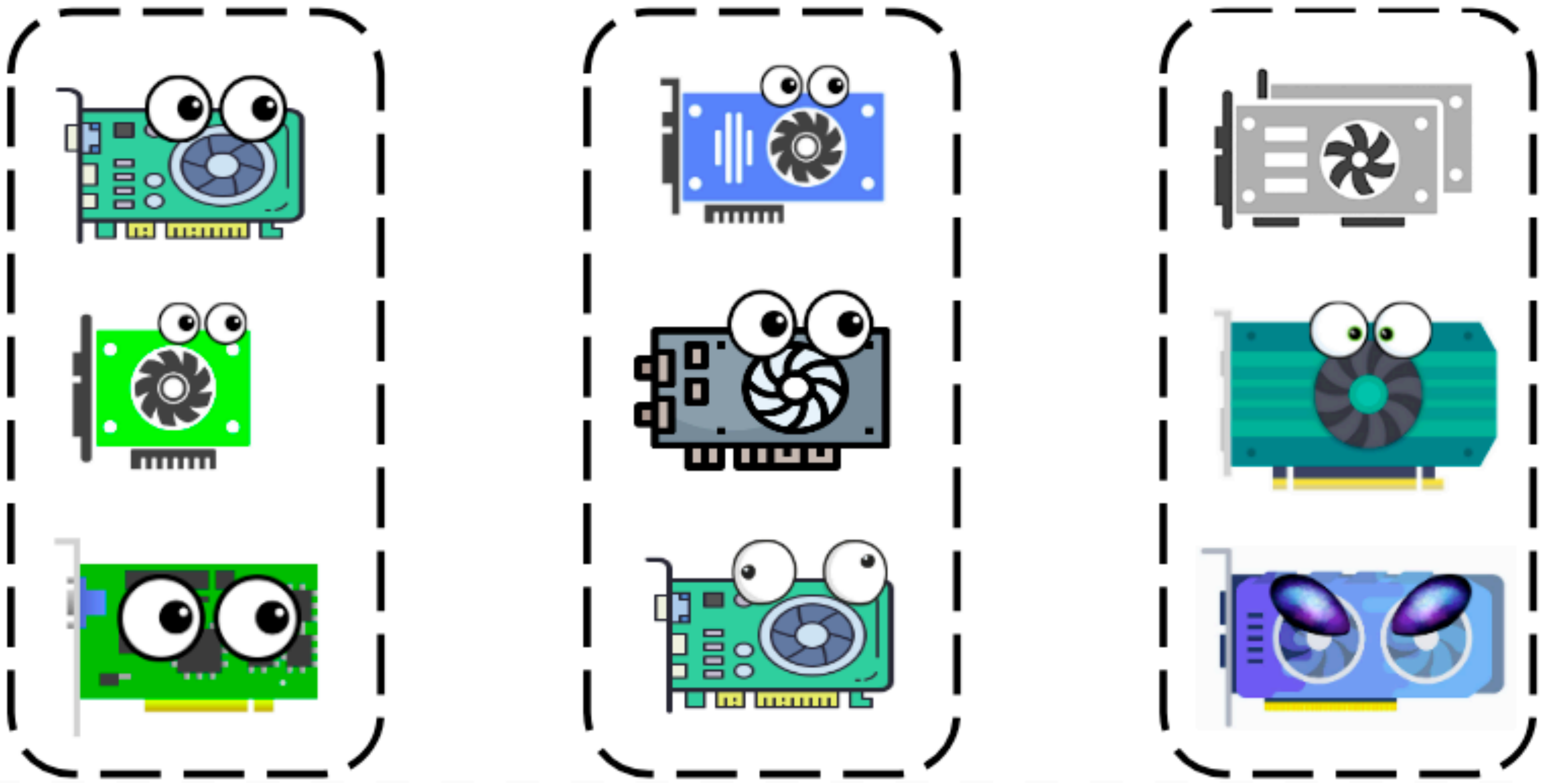
Gennady Pekhimenko
University of Toronto, Canada
Vector Institute, Canada

- We propose a new algorithm for decentralized AllReduce-like averaging
- Main idea: average in smaller non-overlapping groups
- Communication-efficient and fault-tolerant method

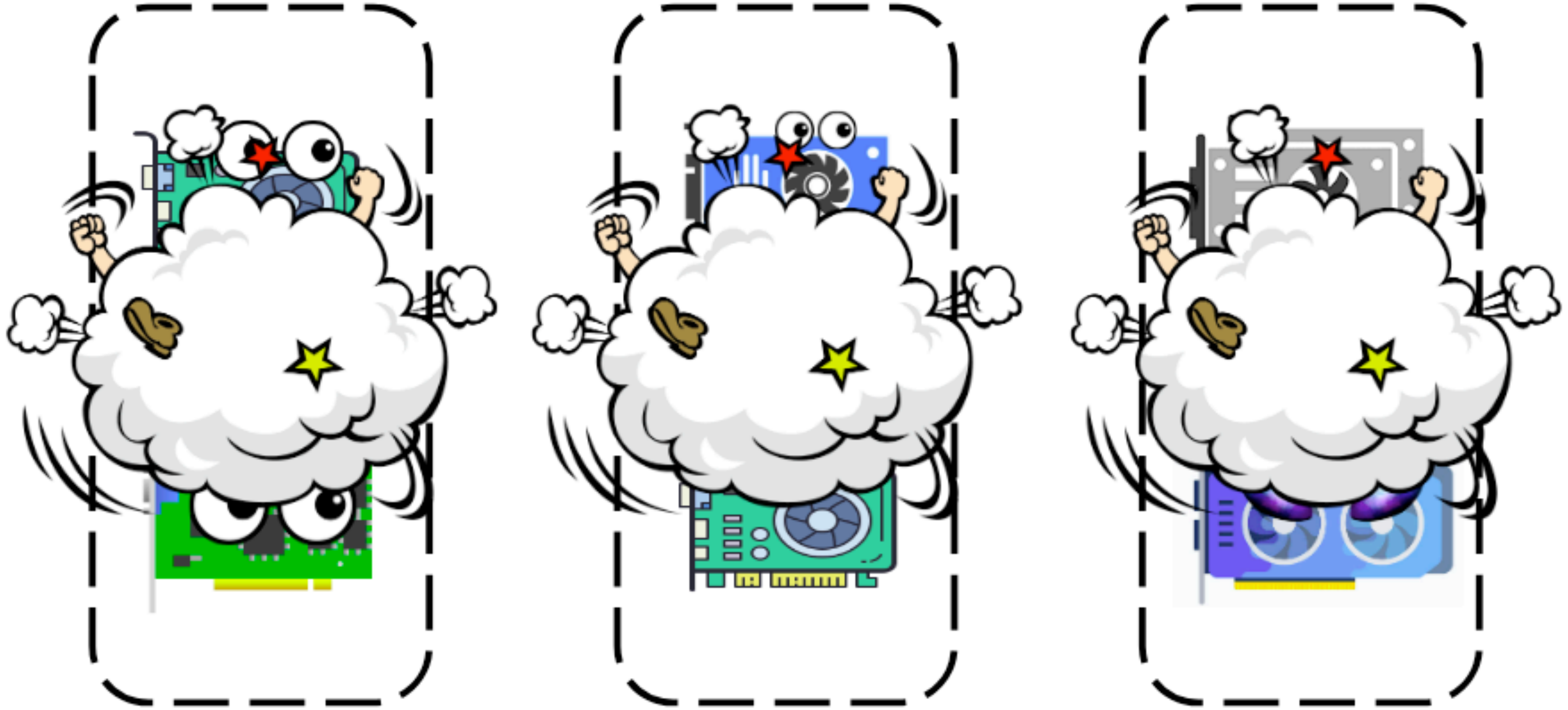
Moshpit Averaging: core idea



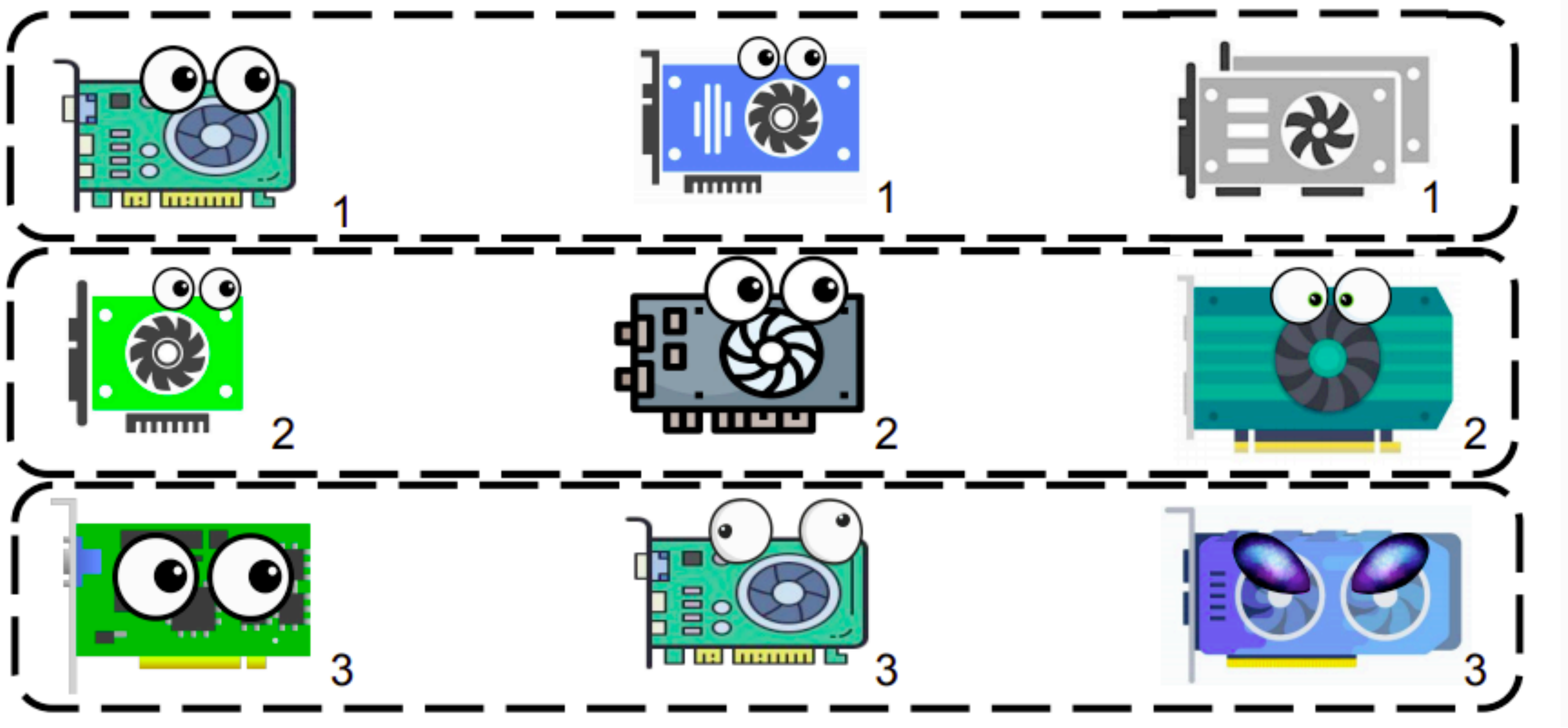
Moshpit All-Reduce: core idea



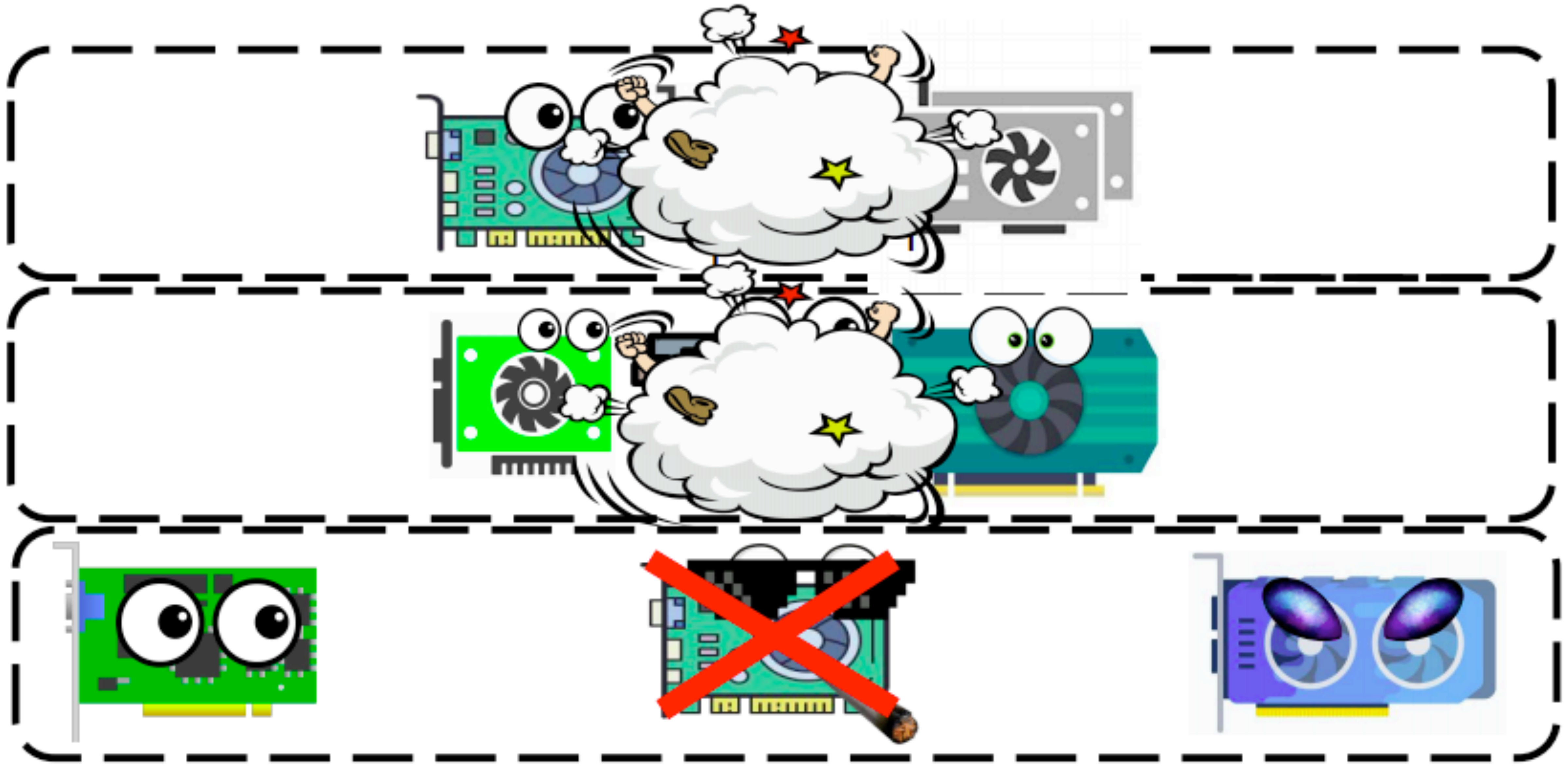
Moshpit All-Reduce: core idea



Moshpit All-Reduce: core idea



Moshpit All-Reduce: core idea



Experiments

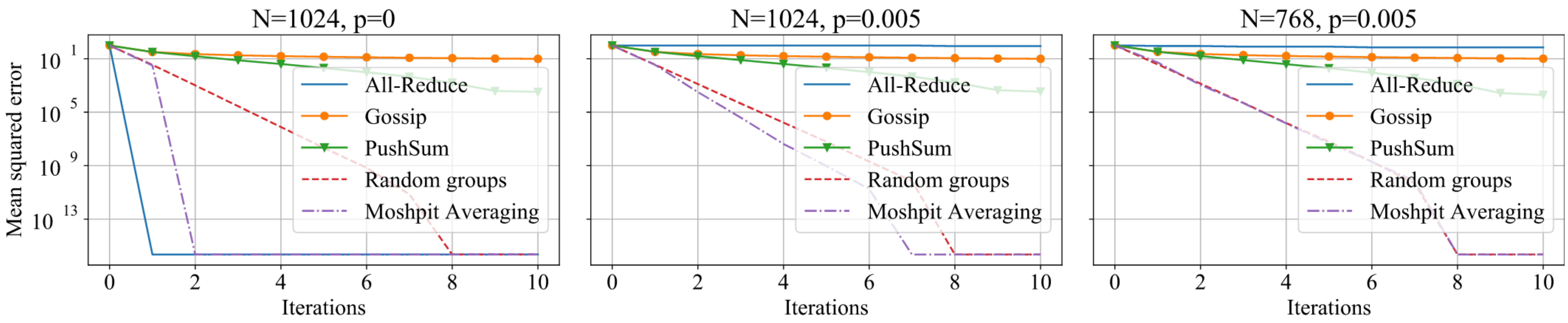
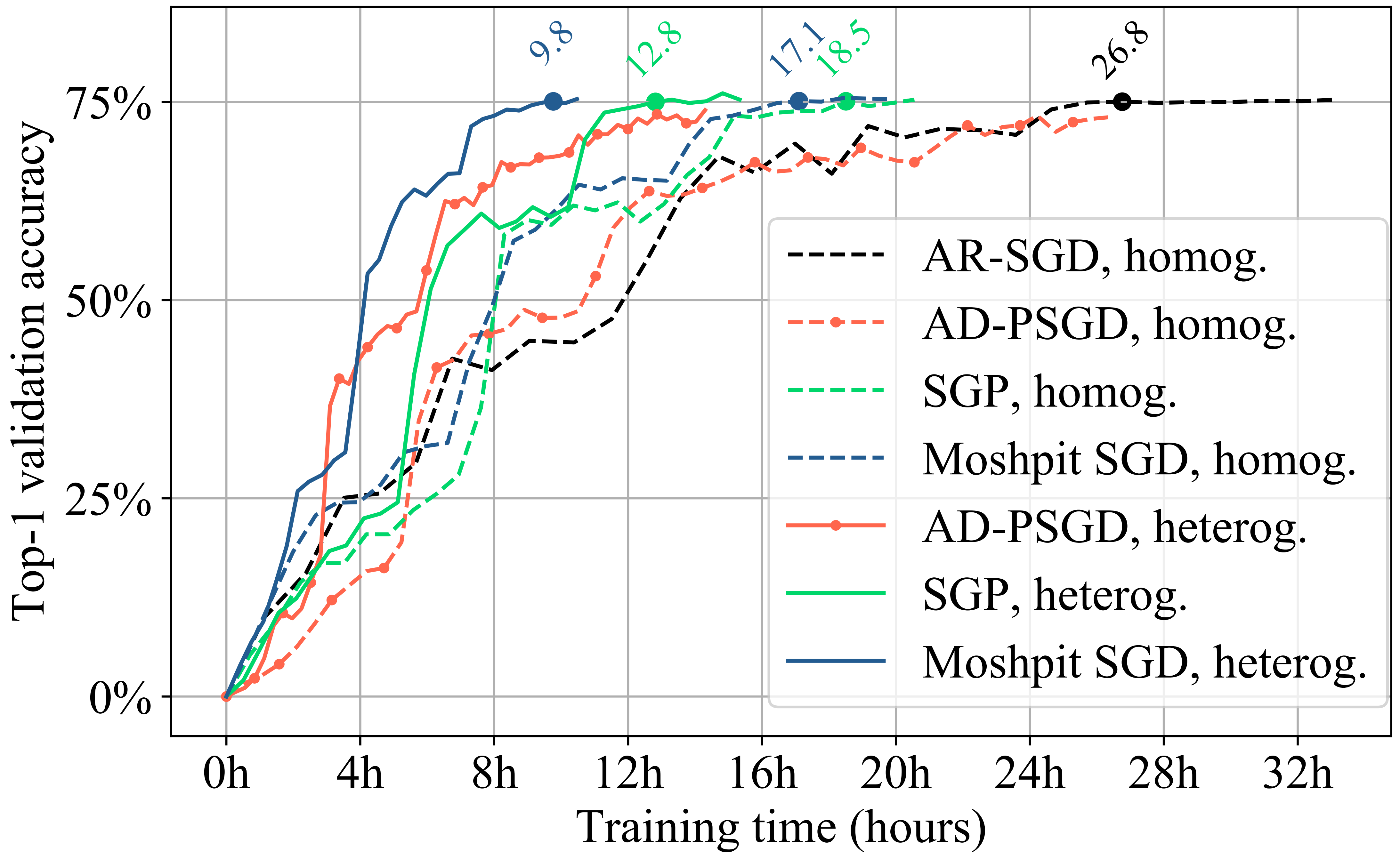
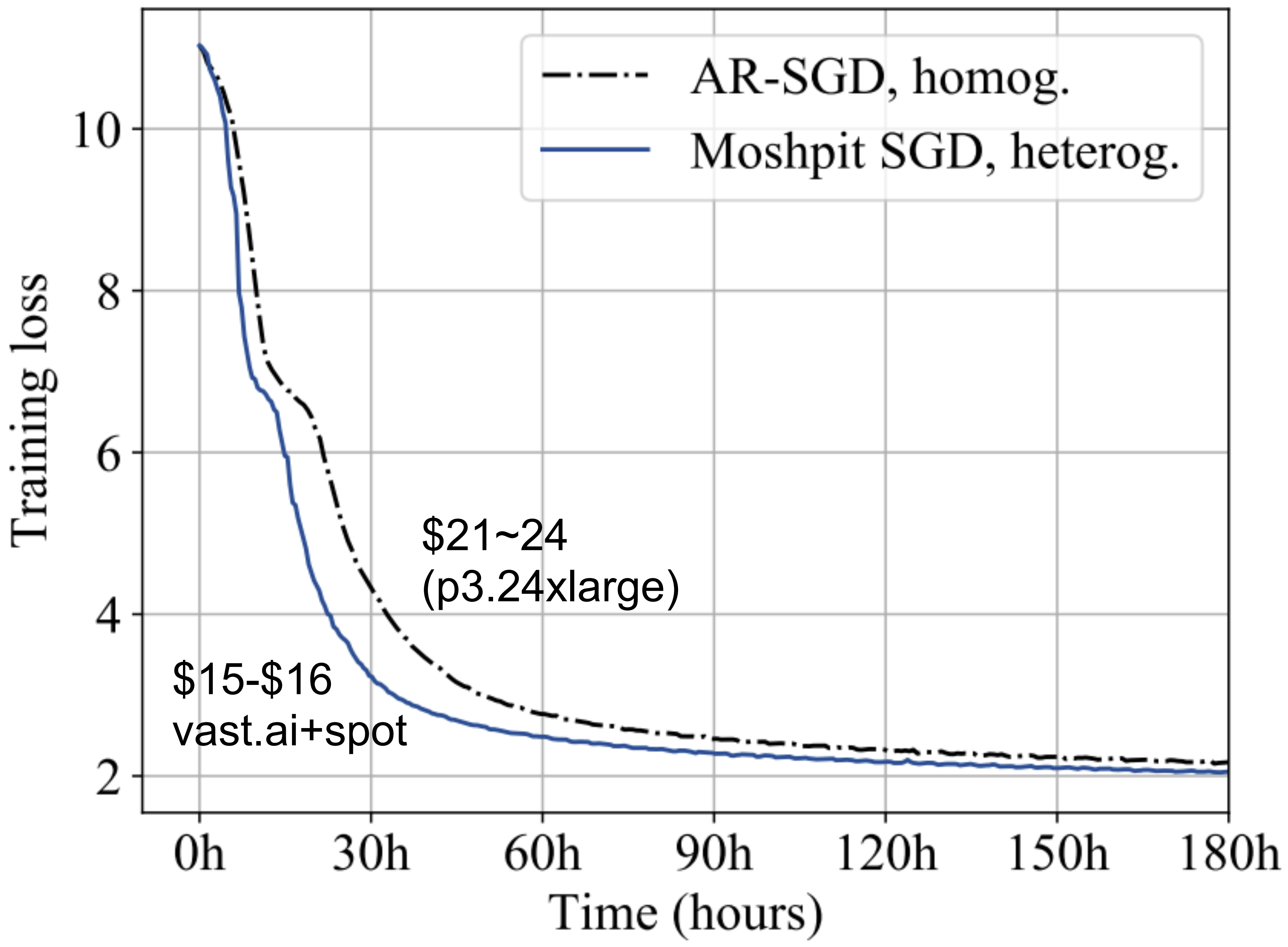


Figure 3: Convergence of averaging algorithms in different configurations.

Experiments





Analysis TL;DR

- The averaging converges exponentially quickly

Theorem 3.2. *Consider a modification of Moshpit All-Reduce that works as follows: at each iteration $k \geq 1$, 1) peers are randomly split in r disjoint groups of sizes $M_1^k, \ldots, M_r^k$ in such a way that $\sum_{i=1}^{r} M_i^k = N$ and $M_i^k \geq 1$ for all $i = 1, \ldots, r$ and 2) peers from each group compute their group average via All-Reduce. Let $\theta_1, \ldots, \theta_N$ be the input vectors of this procedure and $\theta_1^T, \ldots, \theta_N^T$ be the outputs after T iterations. Also, let $\overline{\theta} = \frac{1}{N}\sum_{i=1}^{N}\theta_i$ Then,*

$$\mathbb{E}\left[\frac{1}{N}\sum_{i=1}^{N}\|\theta_i^T - \overline{\theta}\|^2\right] = \left(\frac{r-1}{N} + \frac{r}{N^2}\right)^T \frac{1}{N}\sum_{i=1}^{N}\|\theta_i - \overline{\theta}\|^2. \quad (5)$$

Analysis TL;DR

- The averaging converges exponentially quickly

Theorem 3.2. *Consider a modification of Moshpit All-Reduce that works as follows: at each iteration $k \geq 1$, 1) peers are randomly split in r disjoint groups of sizes $M_1^k, \ldots, M_r^k$ in such a way that $\sum_{i=1}^{r} M_i^k = N$ and $M_i^k \geq 1$ for all $i = 1, \ldots, r$ and 2) peers from each group compute their group average via All-Reduce. Let $\theta_1, \ldots, \theta_N$ be the input vectors of this procedure and $\theta_1^T, \ldots, \theta_N^T$ be the outputs after T iterations. Also, let $\overline{\theta} = \frac{1}{N}\sum_{i=1}^{N} \theta_i$ Then,*

$$\mathbb{E}\left[\frac{1}{N}\sum_{i=1}^{N}\|\theta_i^T - \overline{\theta}\|^2\right] = \left(\frac{r-1}{N} + \frac{r}{N^2}\right)^T \frac{1}{N}\sum_{i=1}^{N}\|\theta_i - \overline{\theta}\|^2. \tag{5}$$

- For Moshpit SGD — equivalent results to Local SGD

Theorem 3.4 (Non-convex case). *Let $f_1 = \ldots = f_N = f$, function f be L-smooth and bounded from below by f_*, and Assumptions 3.1 and 3.2 hold with $\Delta_{pv}^k = \delta_{pv,1}\gamma\mathbb{E}[\|\nabla f(\theta^k)\|^2] + L\gamma^2\delta_{pv,2}^2$, $\delta_{pv,1} \in [0, 1/2)$, $\delta_{pv,2} \geq 0$. Then there exists such choice of γ that $\mathbb{E}\left[\|\nabla f(\theta_{rand}^K)\|^2\right] \leq \varepsilon^2$ after K iterations of Moshpit SGD, where K equals*

$$\mathcal{O}\left(\frac{L\Delta_0}{(1-2\delta_{pv,1})^2\varepsilon^2}\left[1 + \tau\sqrt{1-2\delta_{pv,1}} + \frac{\delta_{pv,2}^2 + \sigma^2/N_{\min}}{\varepsilon^2} + \frac{\sqrt{(1-2\delta_{pv,1})(\delta_{aq}^2 + (\tau-1)\sigma^2)}}{\varepsilon}\right]\right),$$

$\Delta_0 = f(\theta^0) - f(\theta^*)$ and θ_{rand}^K is chosen uniformly from $\{\theta^0, \theta^1, \ldots, \theta^{K-1}\}$ defined in As. 3.2.

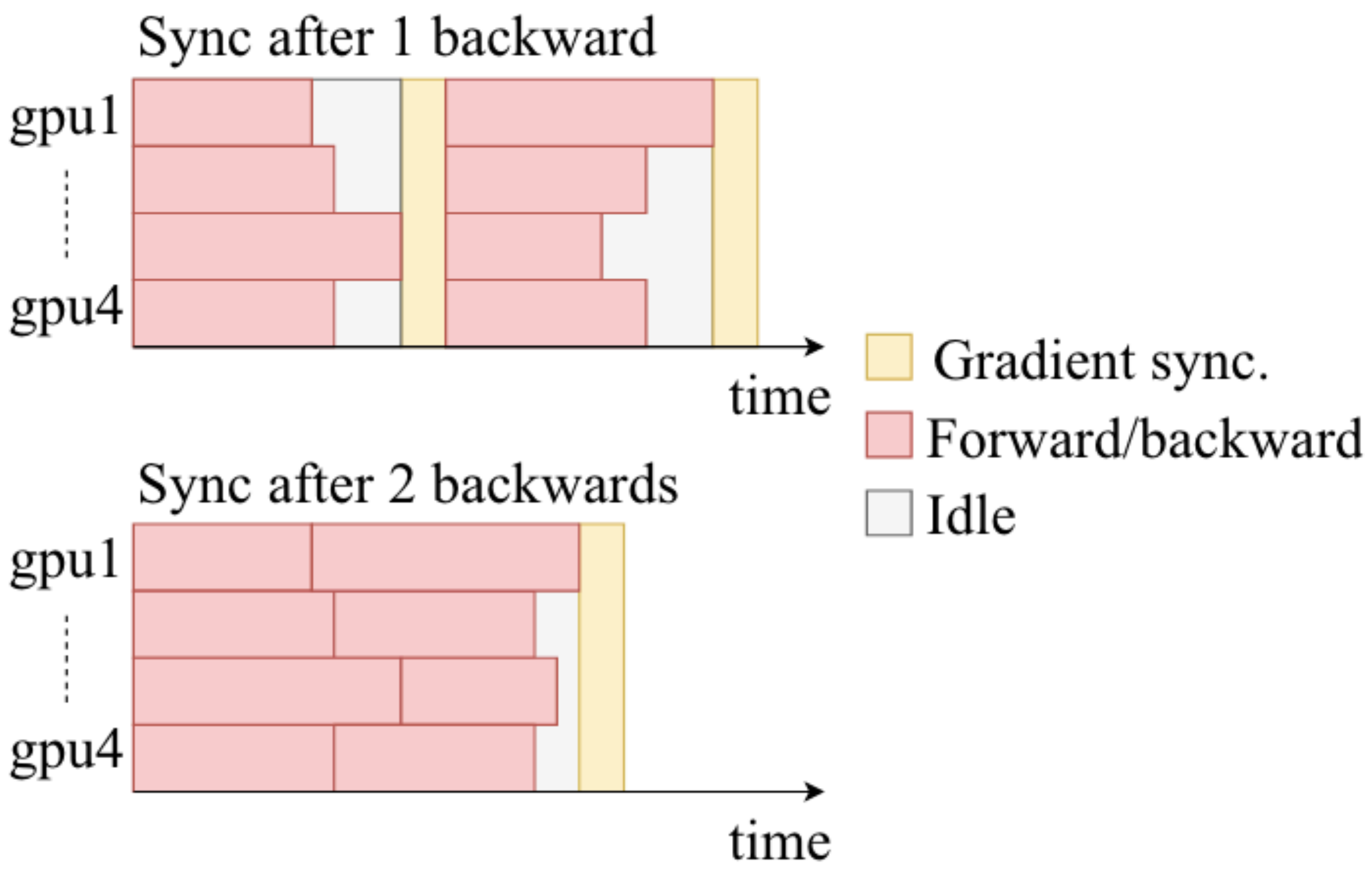
Again, if $\delta_{pv,1} \leq 1/3$, $N_{\min} = \Omega(N)$, $\delta_{pv,2}^2 = \mathcal{O}(\sigma^2/N_{\min})$, and $\delta_{aq}^2 = \mathcal{O}((\tau-1)\sigma)$, then the above theorem recovers the state-of-the-art results in the non-convex case for Local-SGD [64, 63].

</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

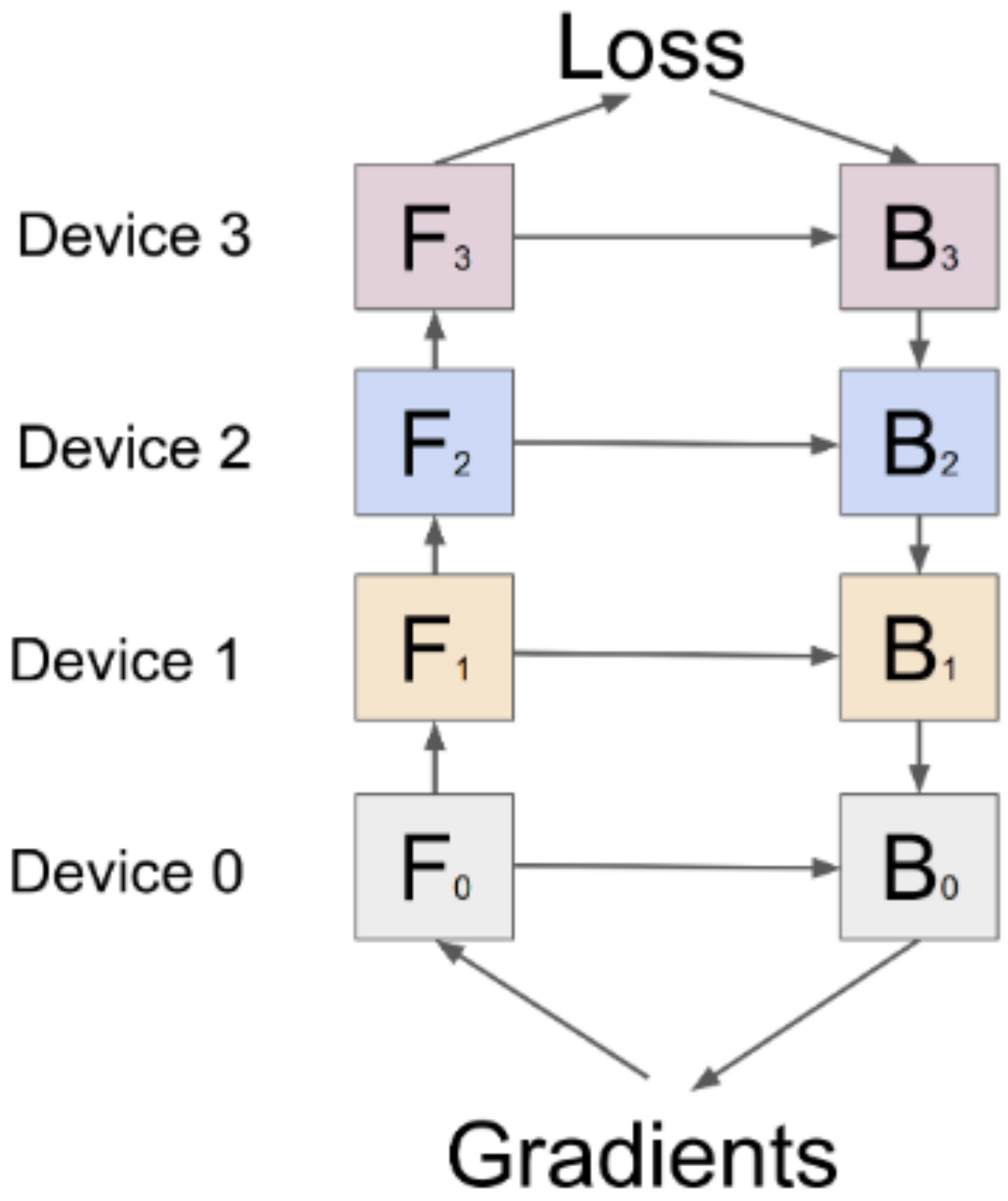
Practical considerations:

- Gradient accumulation helps balance the load (see arxiv.org/abs/1806.00187)
- Communication can be overlapped with computation



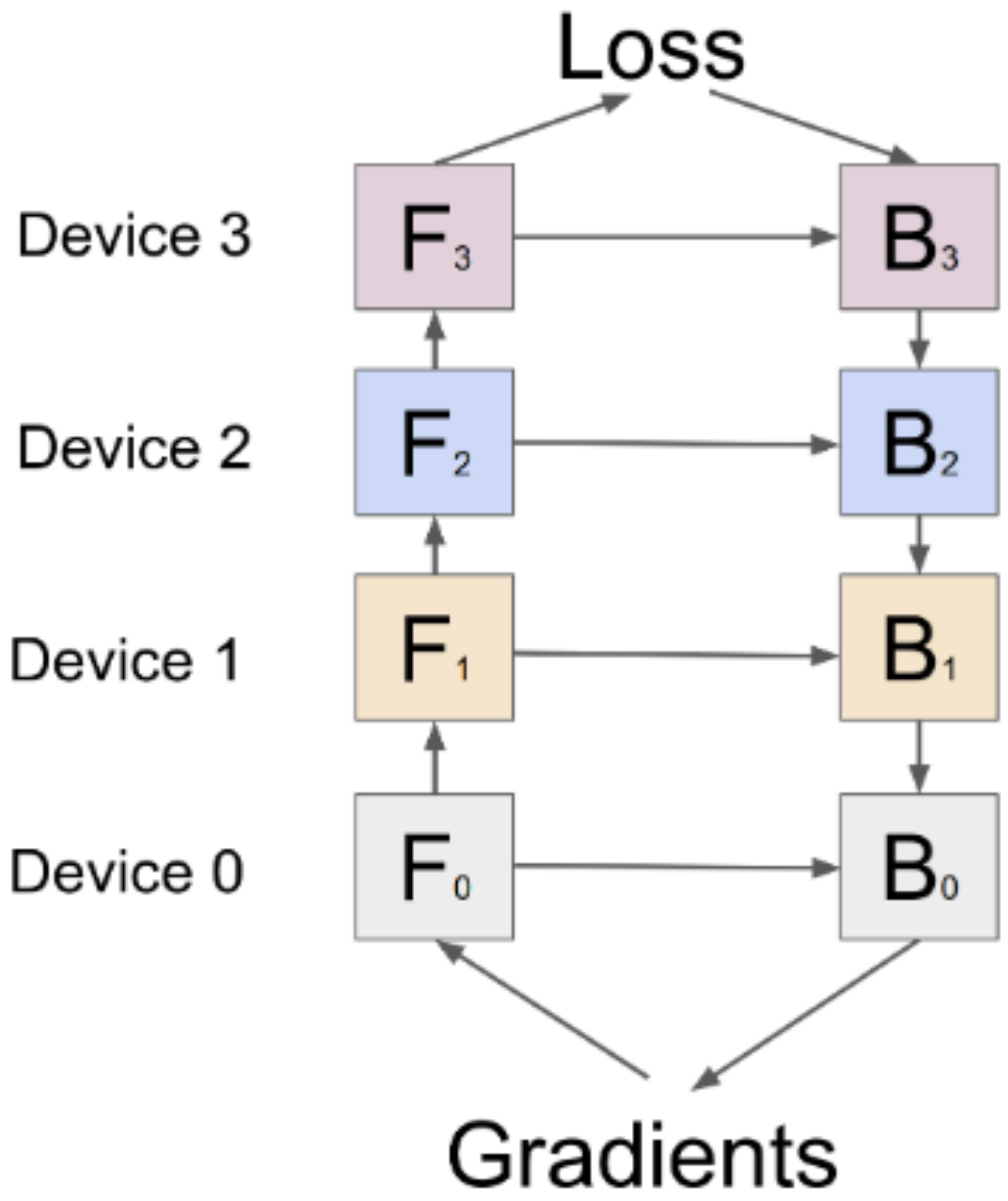
Model-parallel training

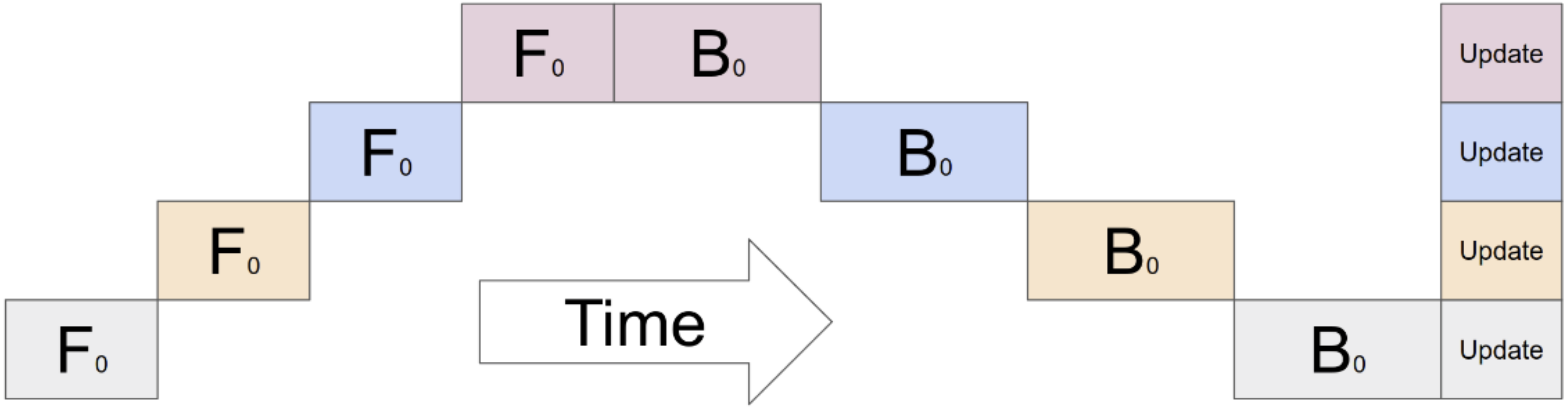
Q: What if a model is larger than GPU?



Model-parallel training

Q: What if a model is larger than GPU?



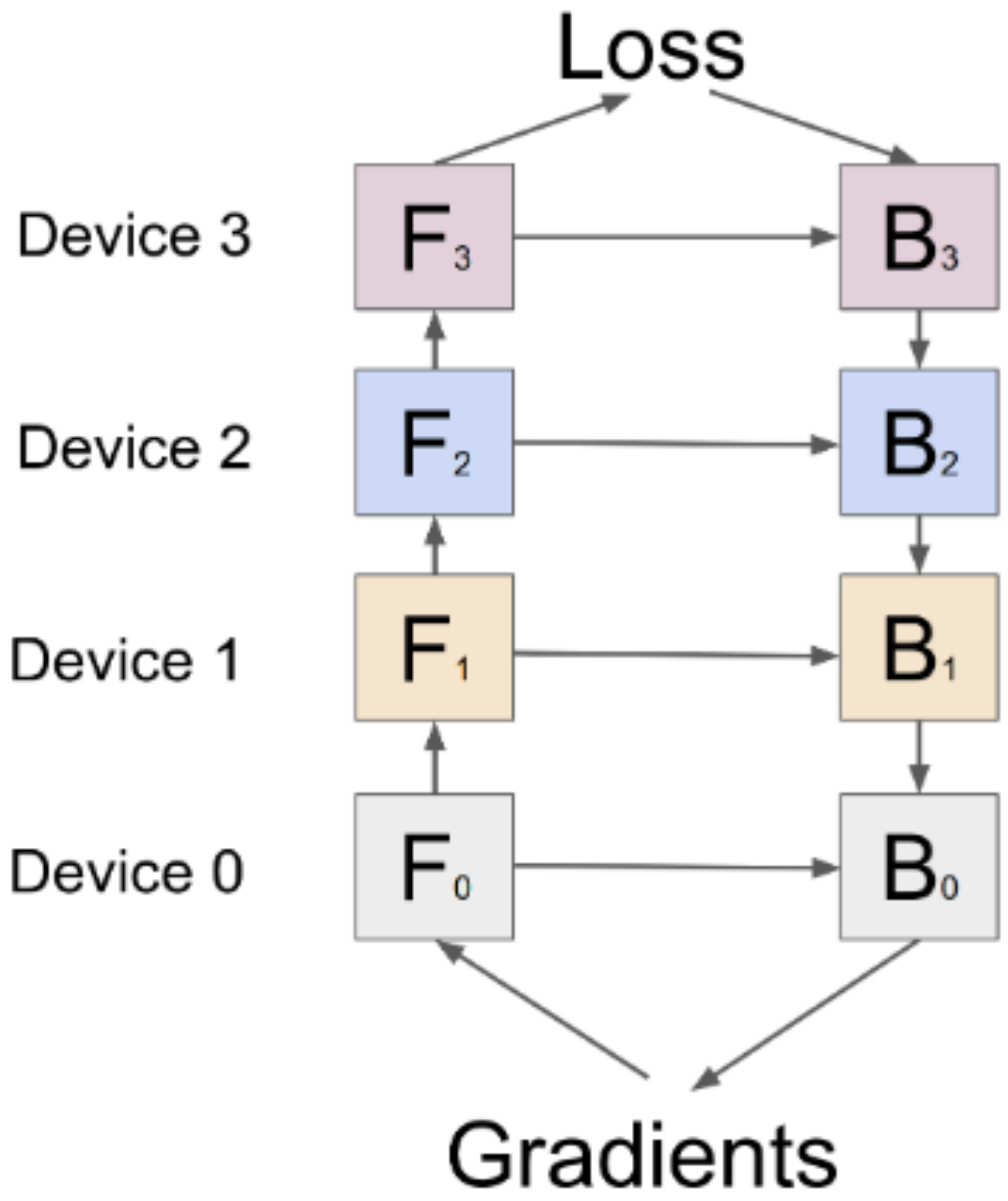


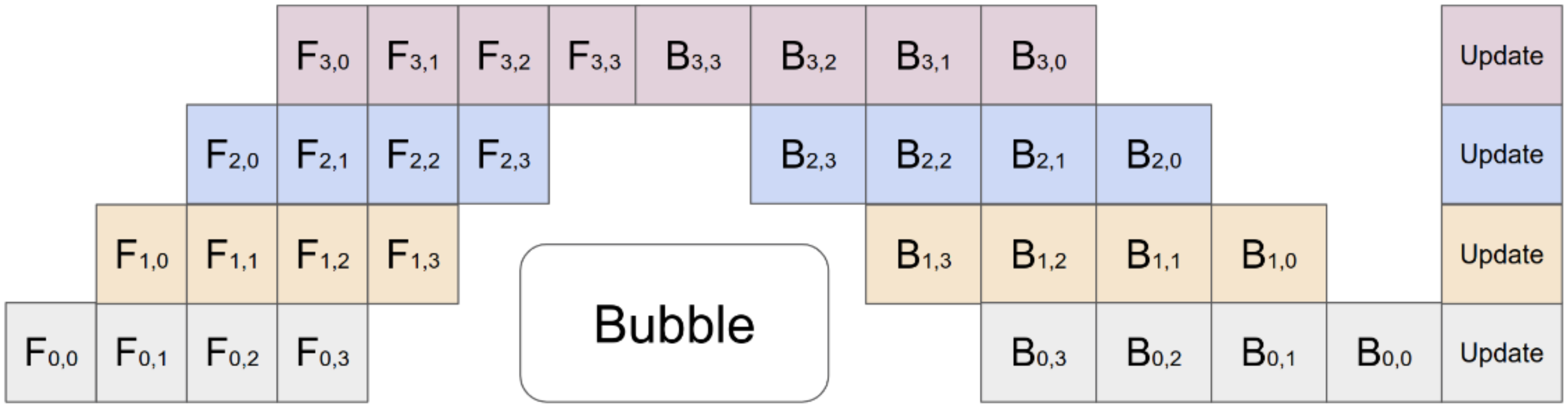
model size: O(N)
throughput: O(1)

Q: Can we go faster?

Pipelining

Idea: split data into micro-batches and form a pipeline (right)



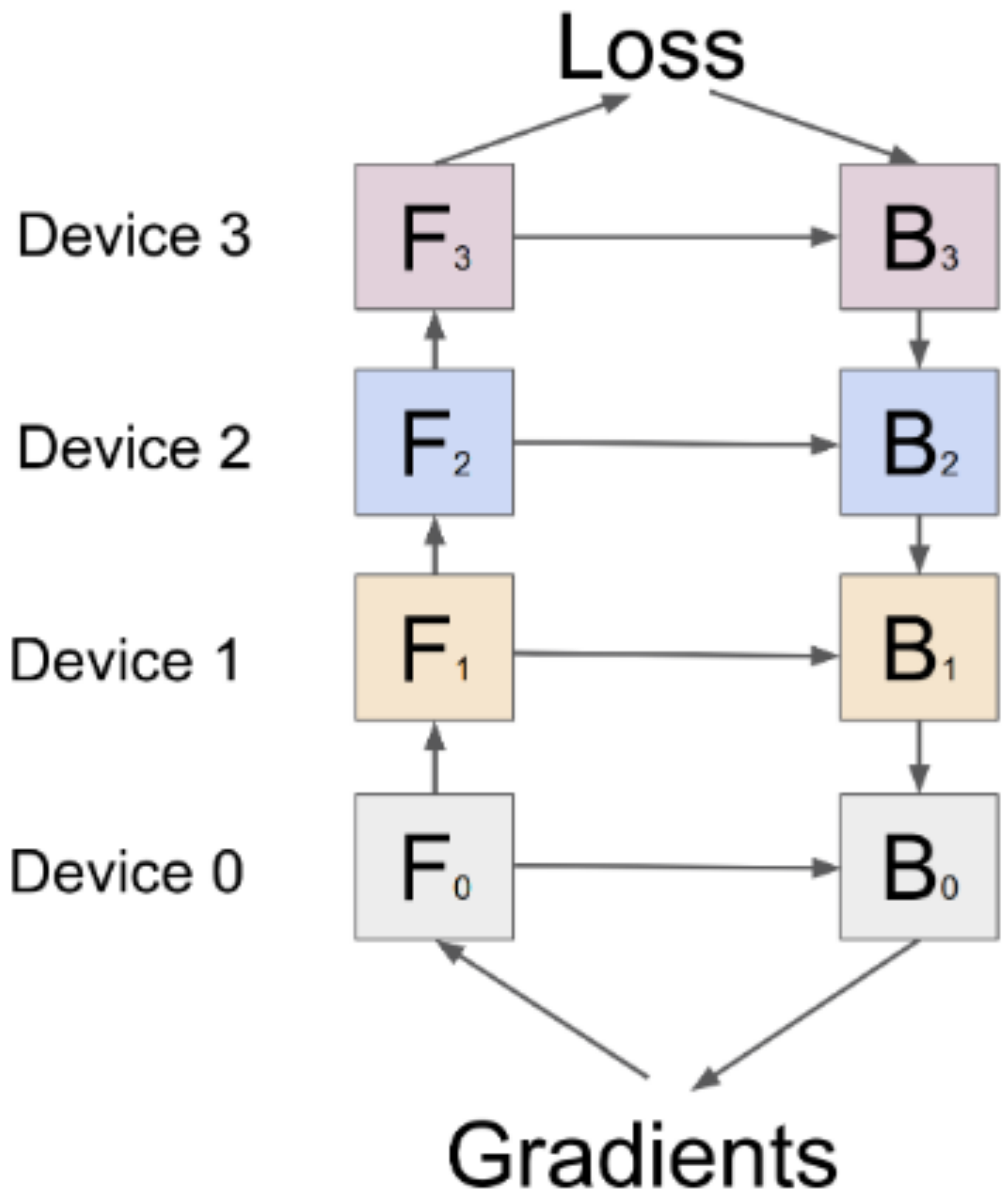


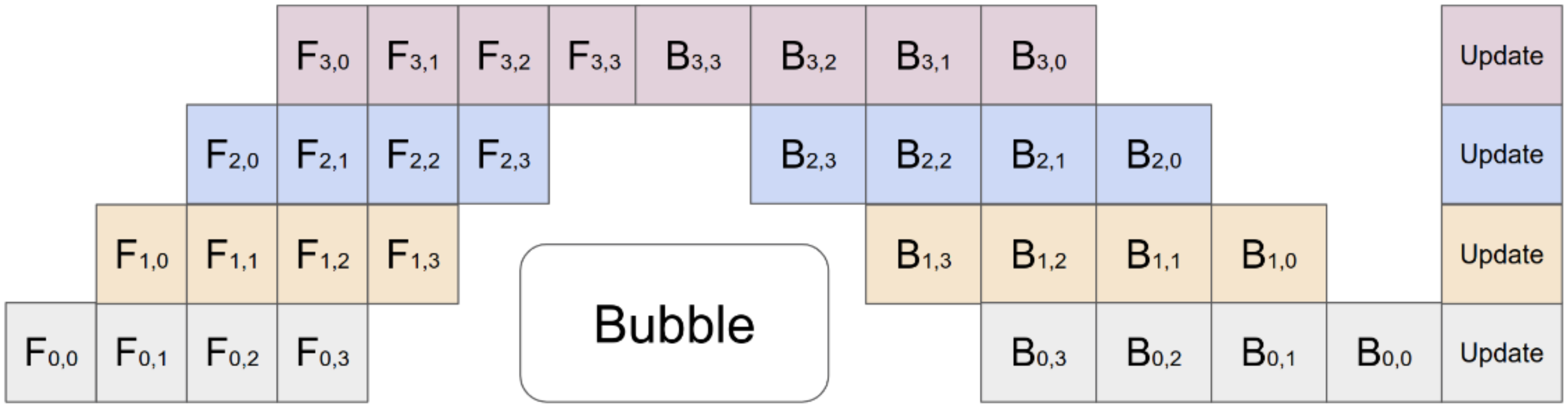
model size: O(n)
throughput: O(n) – with caveats

GPipe: arxiv.org/abs/1811.06965 – good starting point, *not* the first paper

Pipelining

Idea: split data into micro-batches and form a pipeline (right)





model size: O(n)
throughput: O(n) – with caveats

Q: Even faster?

GPipe: arxiv.org/abs/1811.06965 – good starting point, *not* the first paper

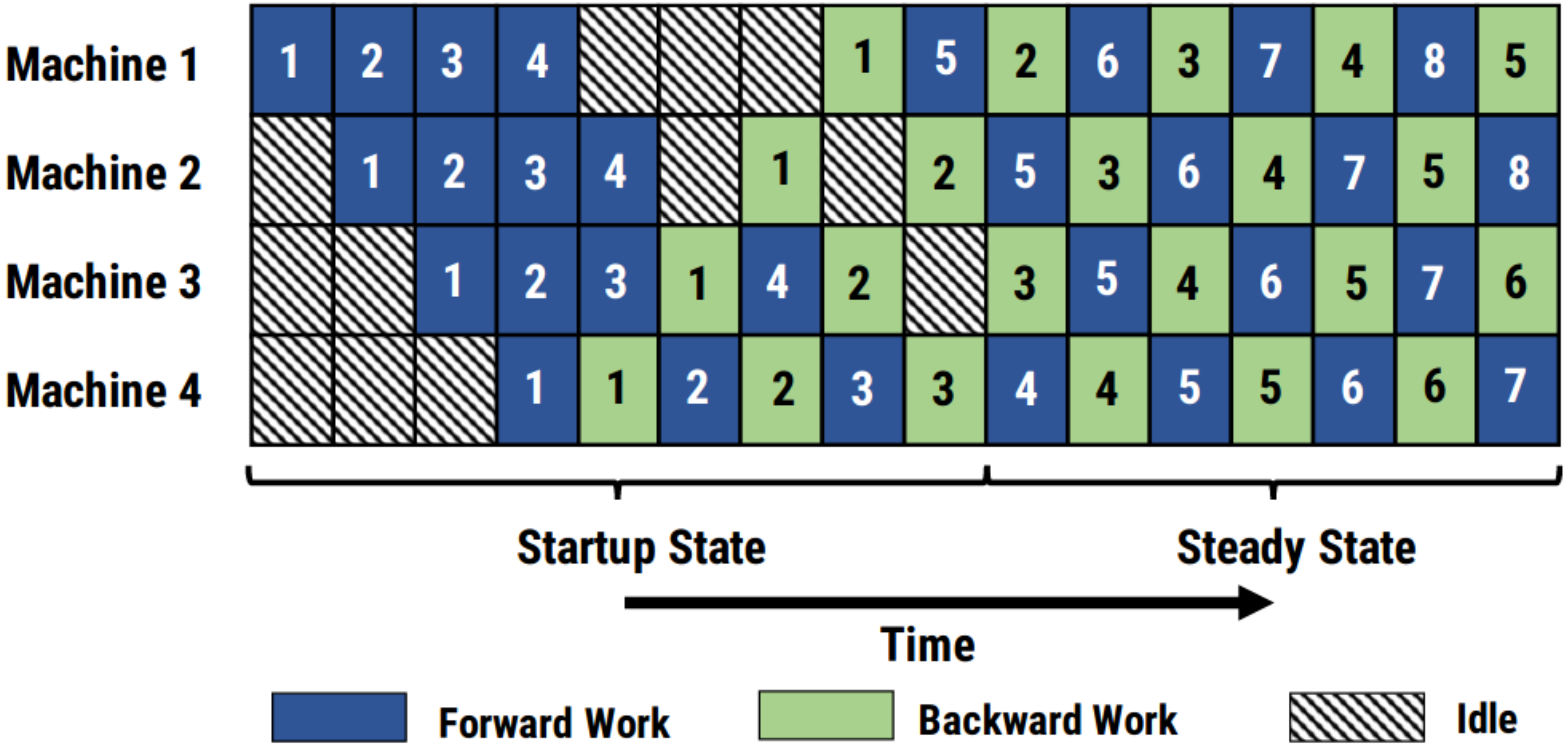
Pipeline-parallel training

PipeDream: arxiv.org/abs/1806.03377

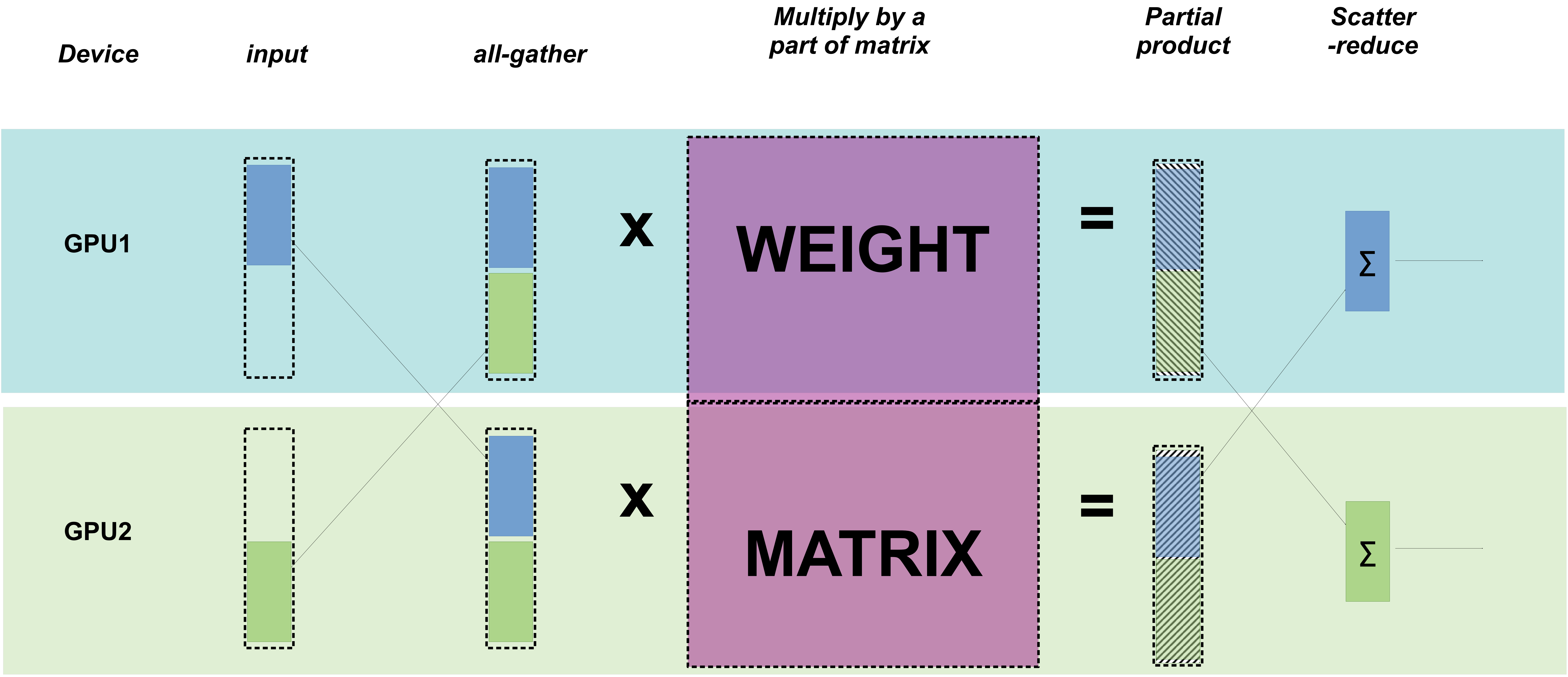
Idea: apply gradients with every microbatch for maximum throughput

Also neat:

- Automatically partition layers to GPUs via dynamic programming
- Store k past weight versions to reduce gradient staleness
- Aims at high latency



Tensor-parallel training



Q: find AllReduce op here

Q: find AllReduce op here

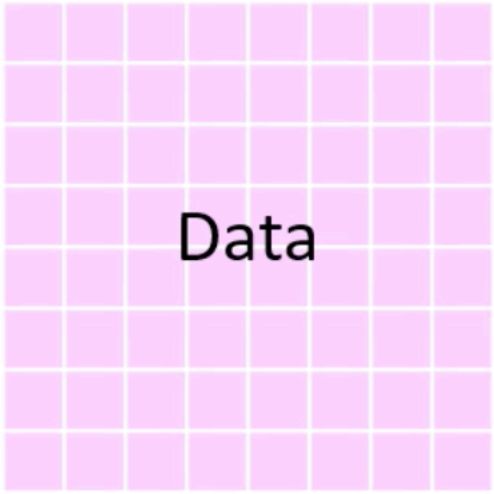
</Model-parallel>

+ model larger than GPU
+ faster for small
* typical size: 2-8 gpus
- model partitioning is tricky
tensor parallelism is easier, but requires ultra low latency
- latency is critical, go buy nvlink
except for PipeDream
- often combined with gradient checkpointing

Tensor parallelism: M*esh TensorFlow* (*arxiv.org/abs/1811.02084*)

DeepSpeed

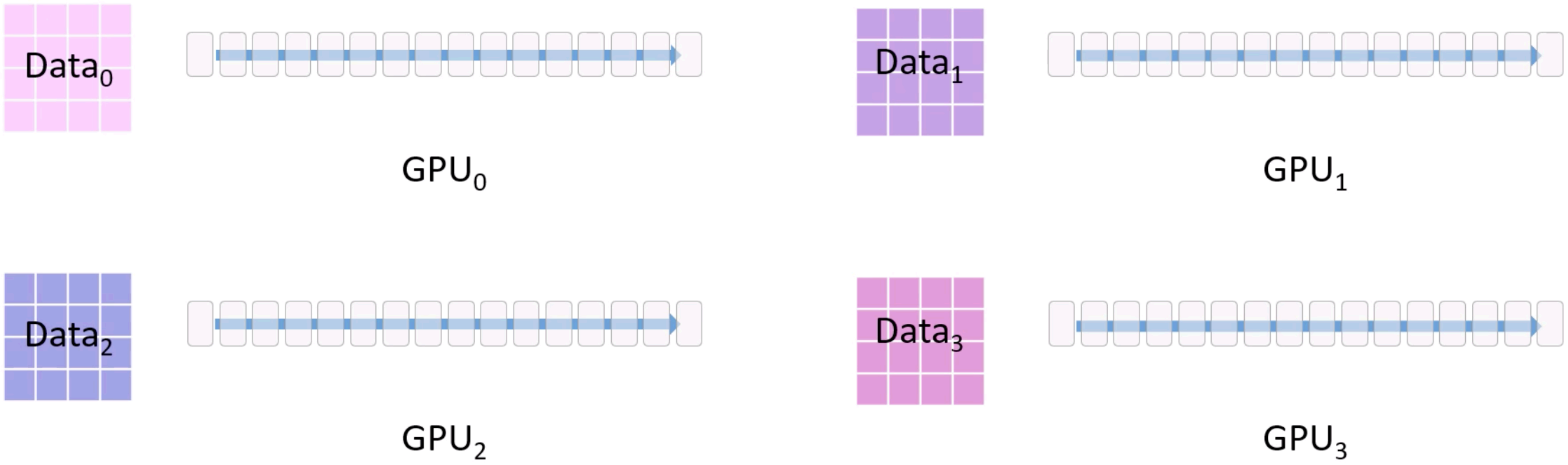
Source: <u>microsoft</u>



Here is a big training dataset

DeepSpeed

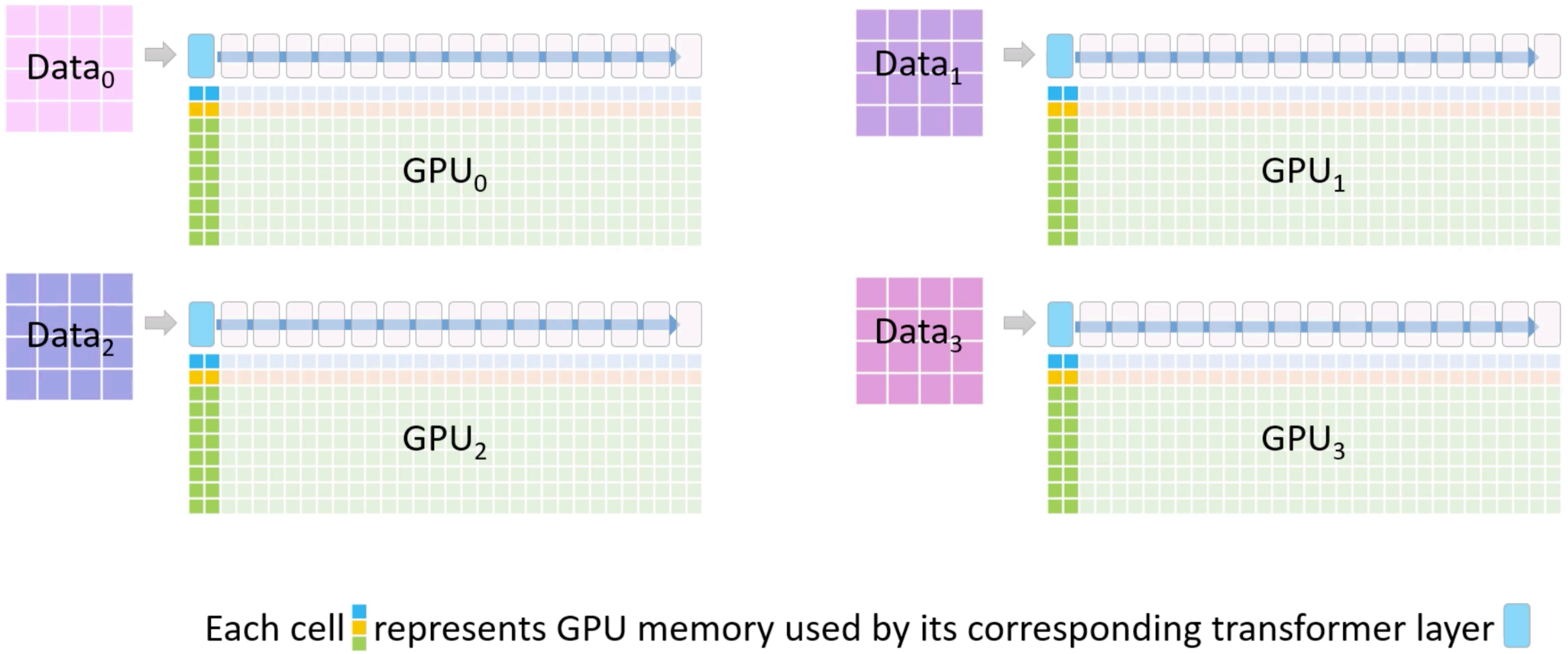
Source: microsoft



We will use 4-way data parallelism and ZeRO P_{os+g+p} memory optimization
Each GPU will optimize the same model on different data

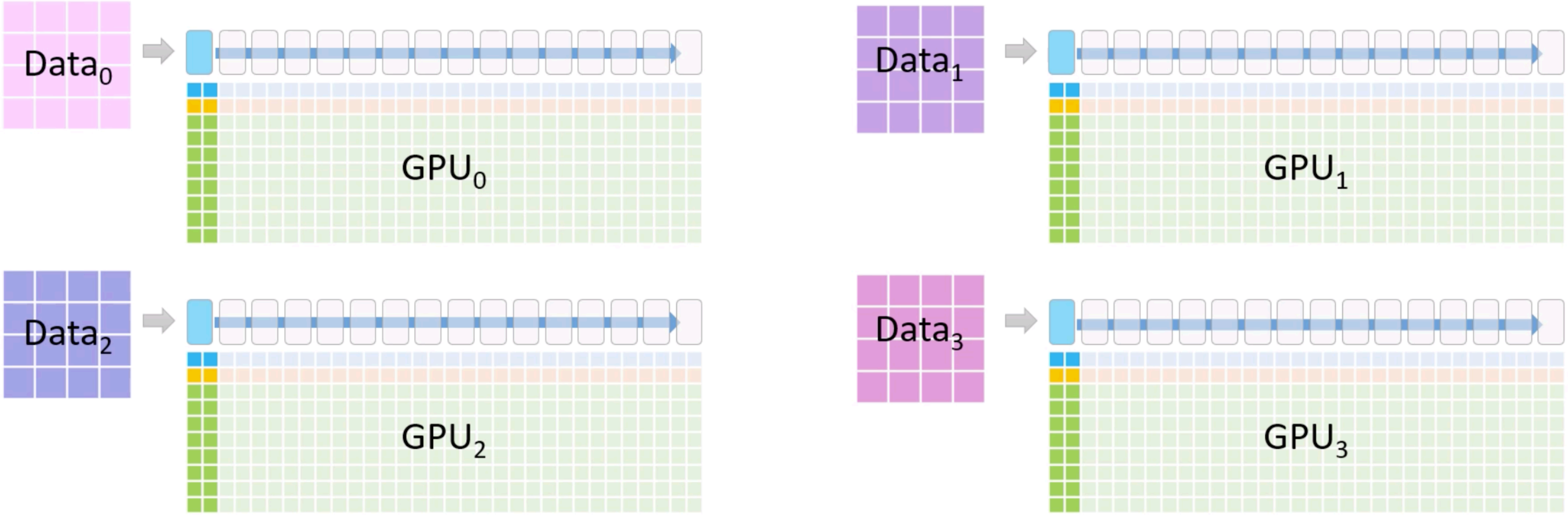
DeepSpeed

Source: microsoft



DeepSpeed

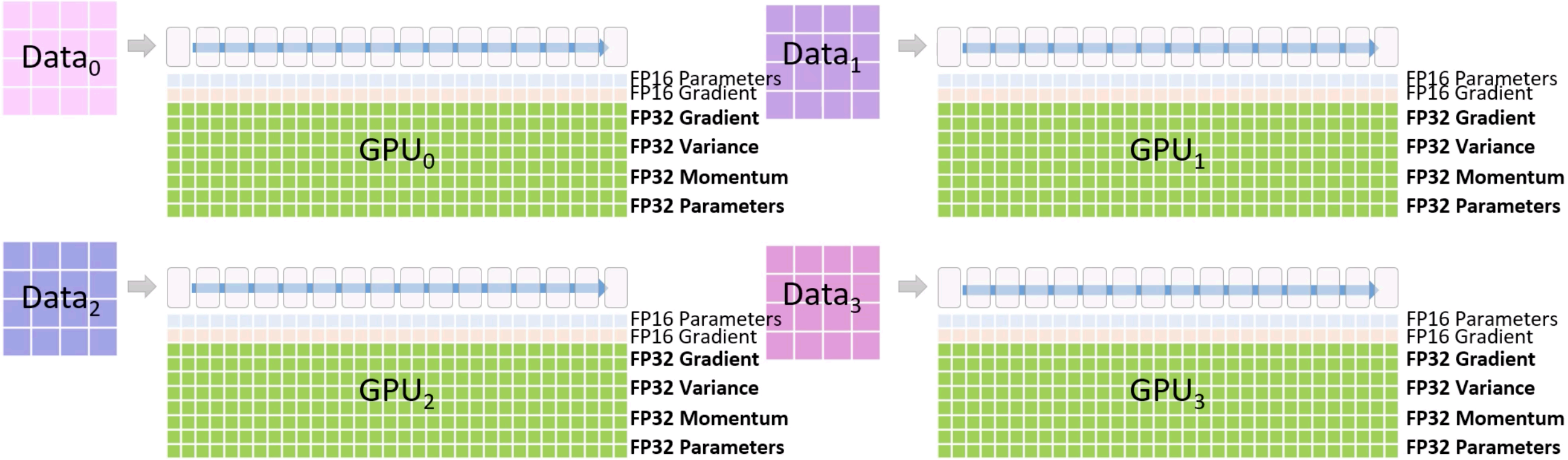
Source: microsoft



Each cell represents GPU memory used by its corresponding transformer layer

DeepSpeed

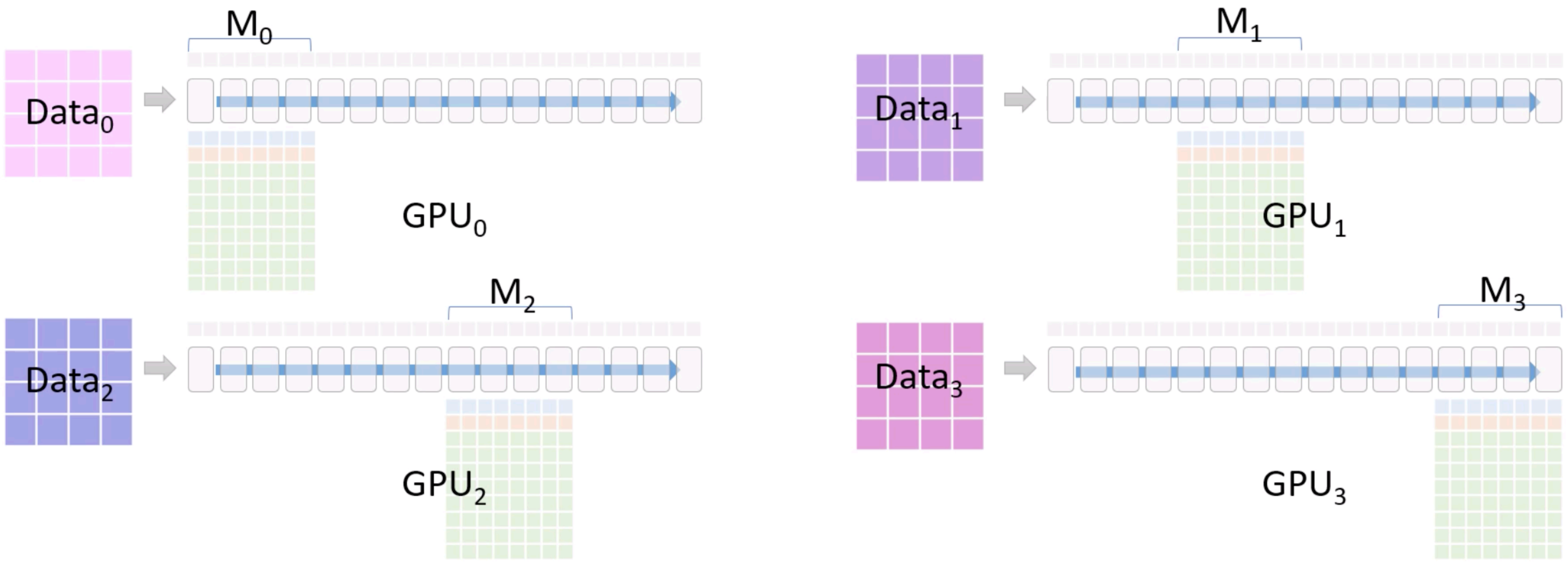
Source: microsoft



The last (massive) block of memory is used by the Optimizer.
This is not used until after the fp16 gradients are computed

DeepSpeed

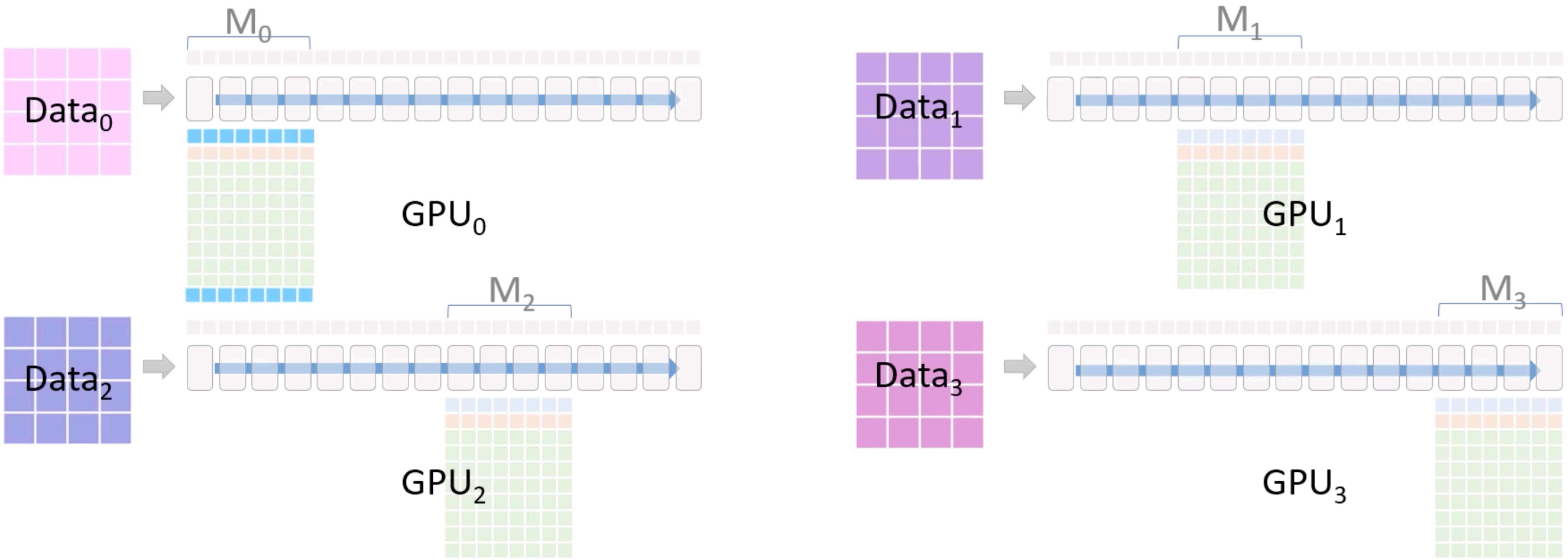
Source: microsoft



Each GPU is responsible for 1 piece of the end model
ZeRO P_{os+g+p} and Gradient accumulation are used with the 4-way data parallelism

DeepSpeed

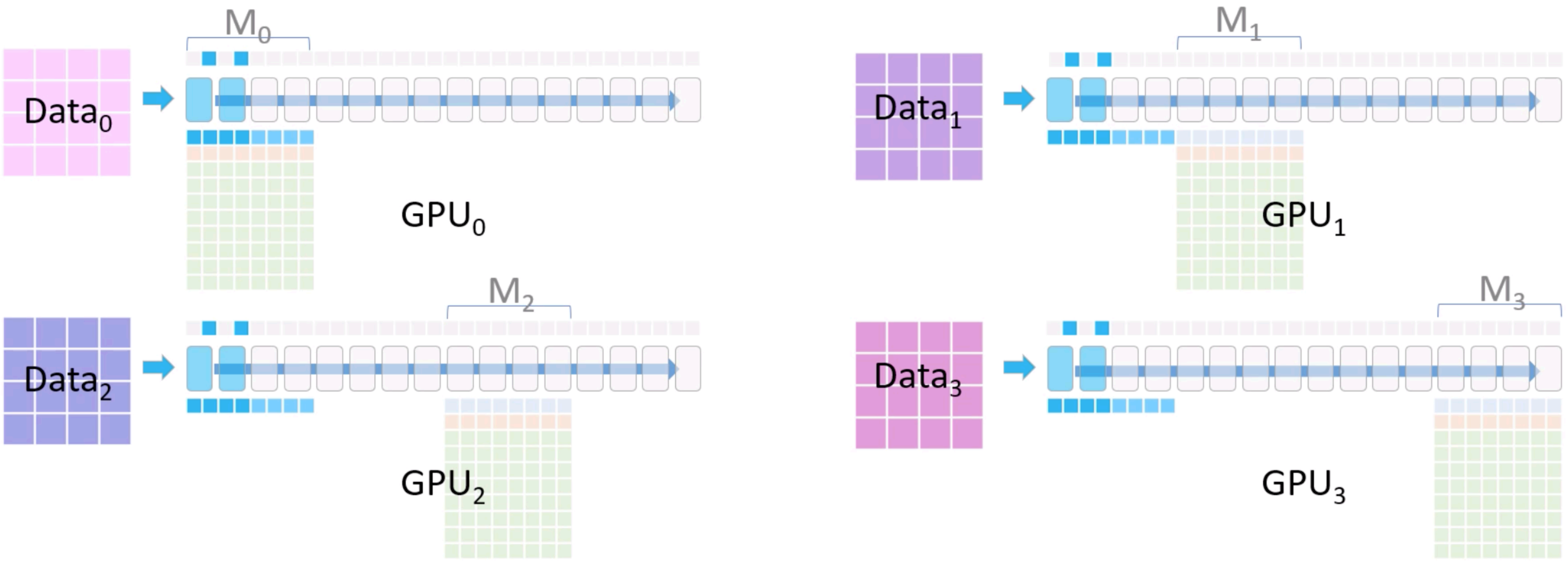
Source: microsoft



Only GPU_0 initially has the model parameters for M_0.
It broadcasts them to $GPU_{1,2,3}$

DeepSpeed

Source: microsoft



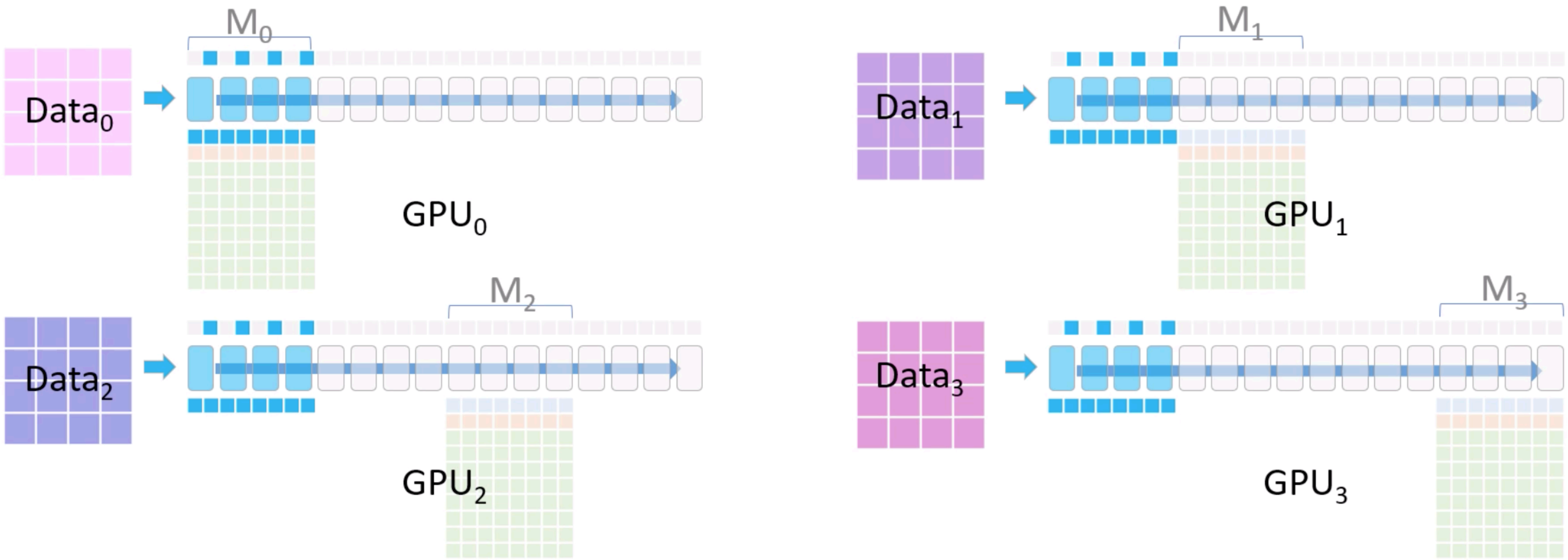
Run the forward pass

Each GPU runs on M_0's parameters using its own data

Only part of each layer's activations are retained

DeepSpeed

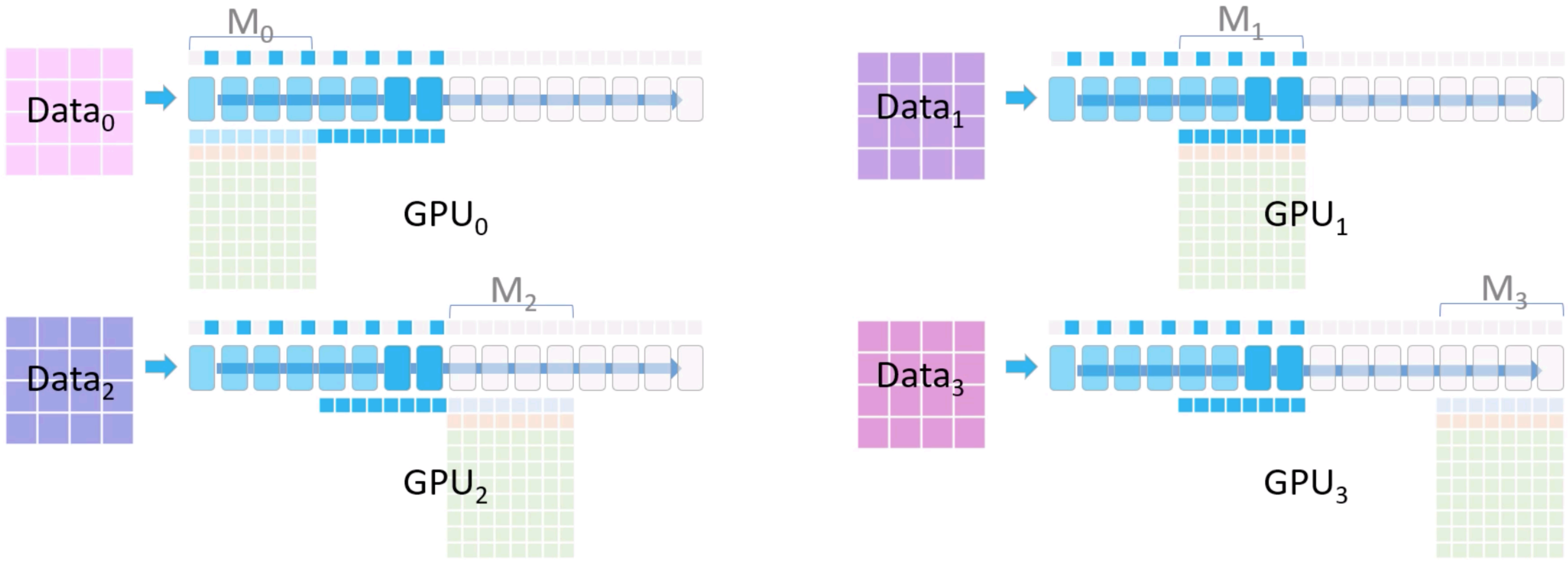
Source: microsoft



Once M_0 is complete, $GPU_{1,2,3}$ can delete the parameters for M_0

DeepSpeed

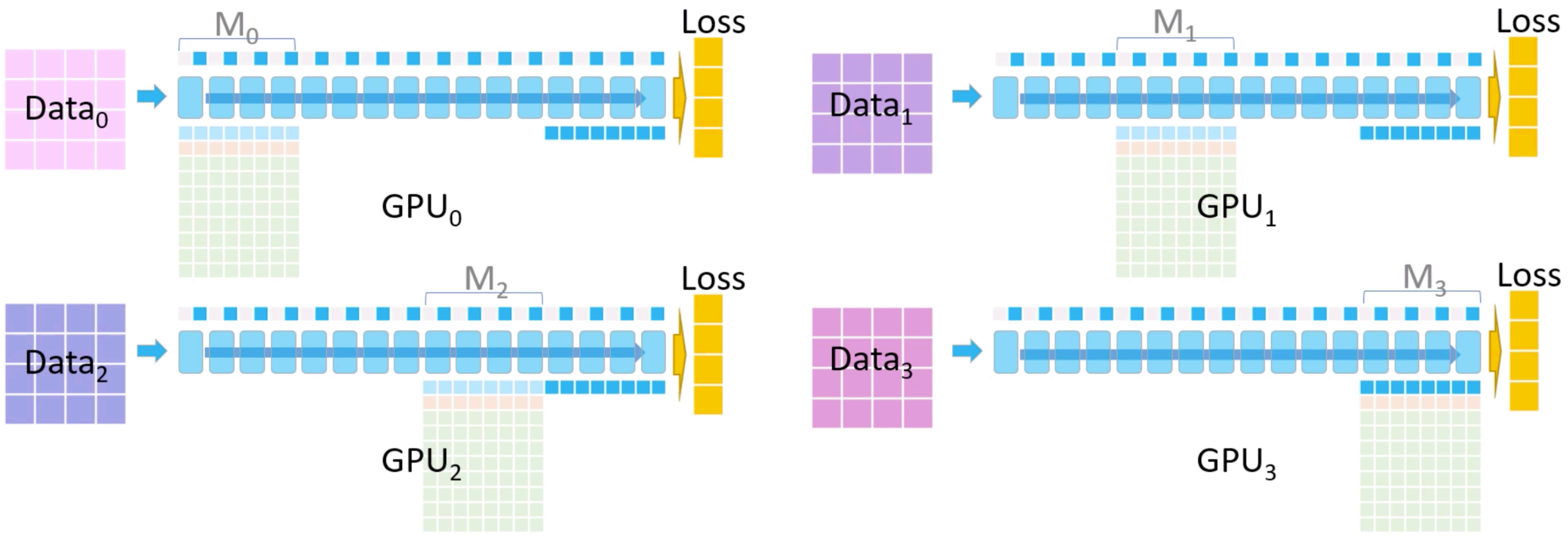
Source: <u>microsoft</u>



The forward pass continues across all GPUs on M_1

DeepSpeed

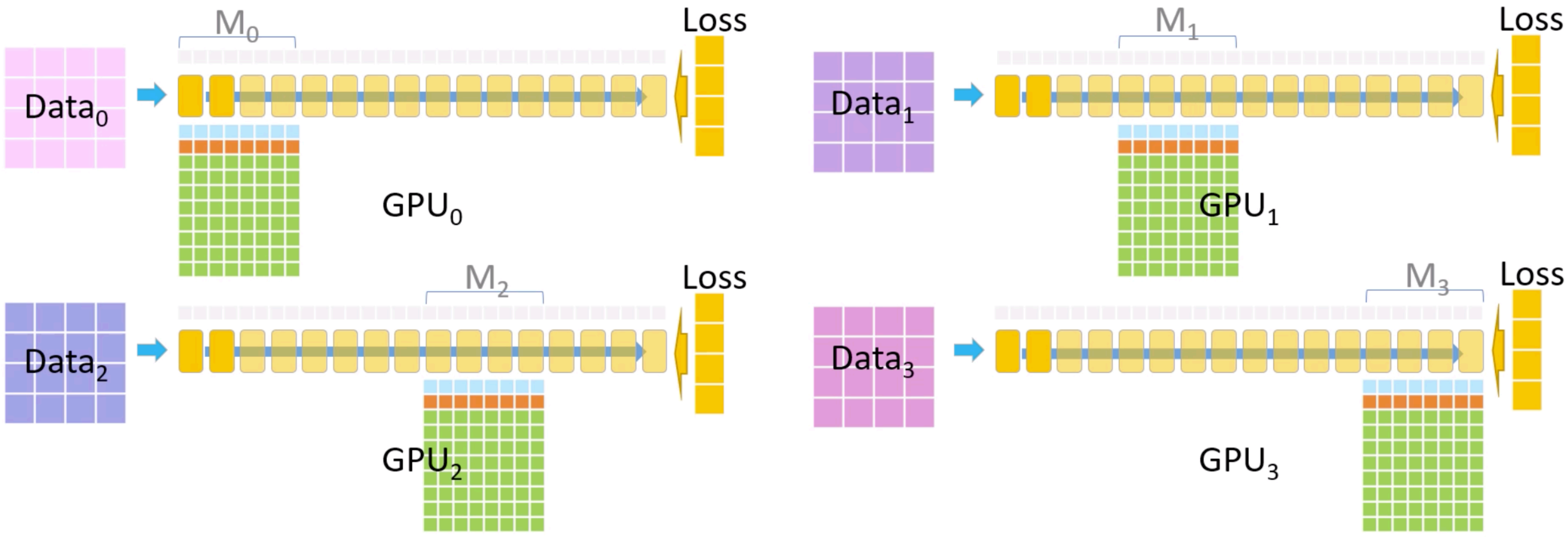
Source: microsoft



The forward pass is complete.
The loss is computed on each GPU for its respective dataset

DeepSpeed

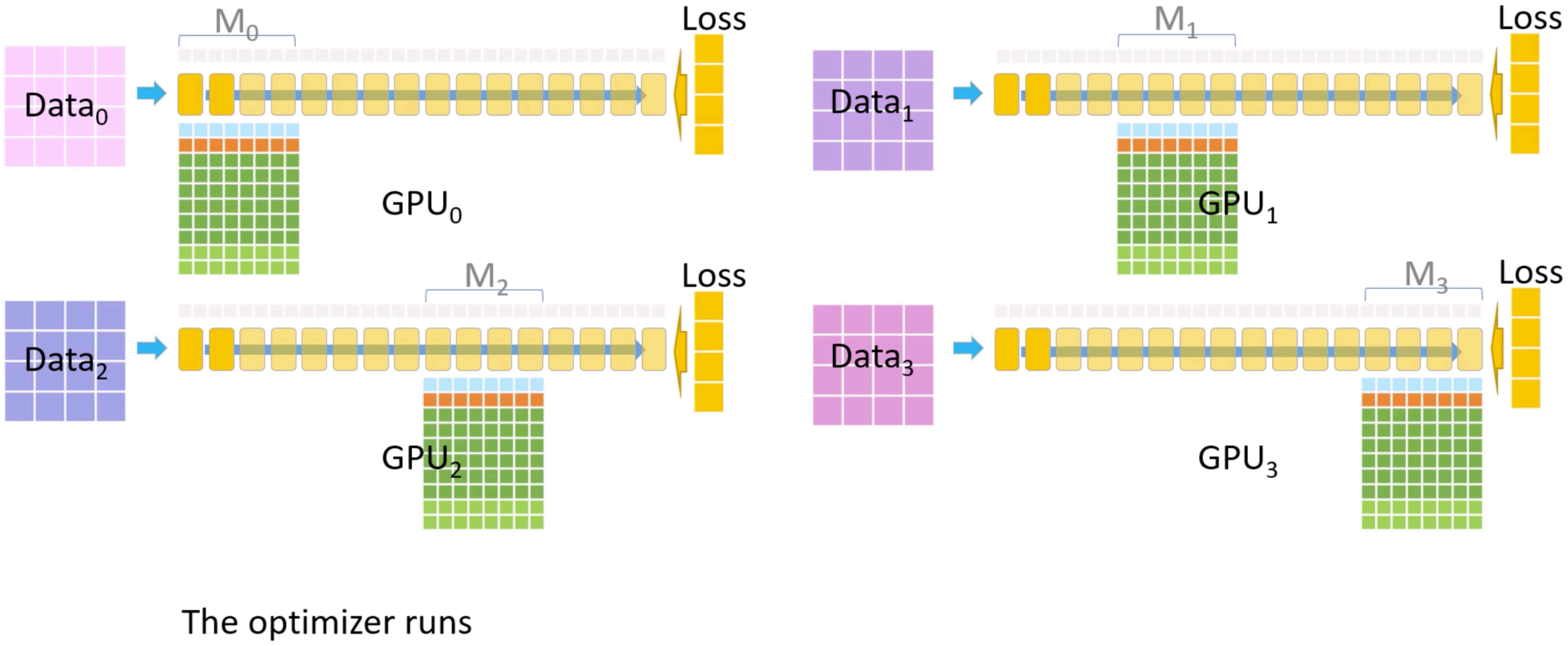
Source: microsoft



The Optimization step begins in parallel on each GPU

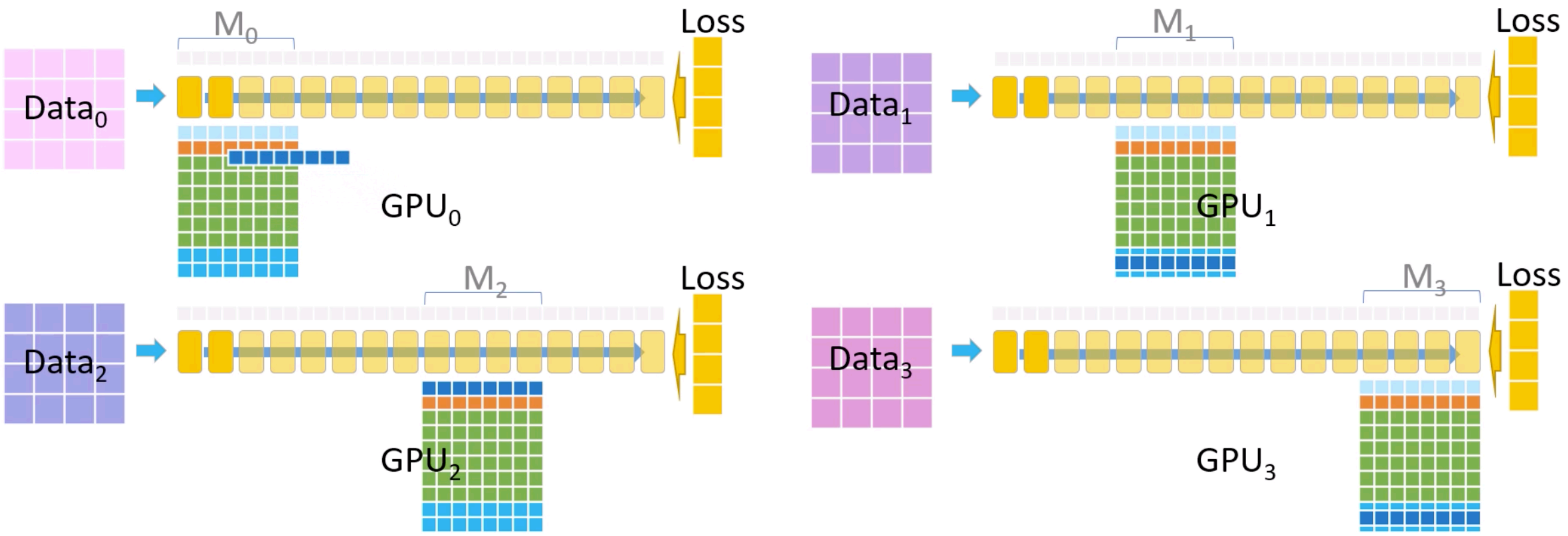
DeepSpeed

Source: microsoft



DeepSpeed

Source: microsoft



The fp16 weights become the model parameters for the next iteration
Training iteration complete!

Conclusions

- Distributed algorithms are highly important in large scale DL
- Key problems are network-related (latency and bandwidth)
- Efficient algorithms allow scaling to hundreds and thousands of GPUs
- The field is growing rapidly!