ACCO:

Accumulate While You Communicate for Communication-Overlapped Sharded LLM Training

Adel Nabli

Dr. Louis Fournier

Dr. Pierre Erbacher

Dr. Louis Serrano

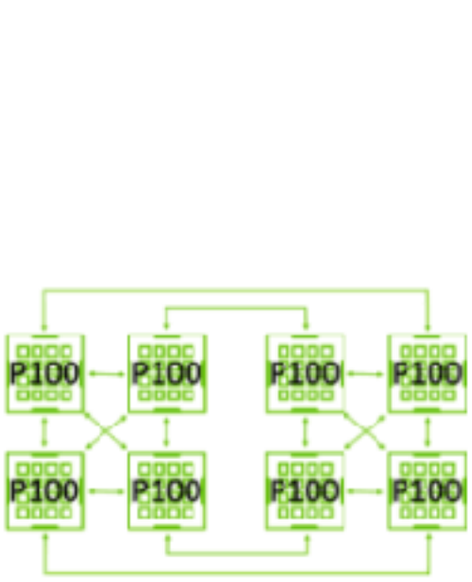
Pr. Eugene Belilovsky

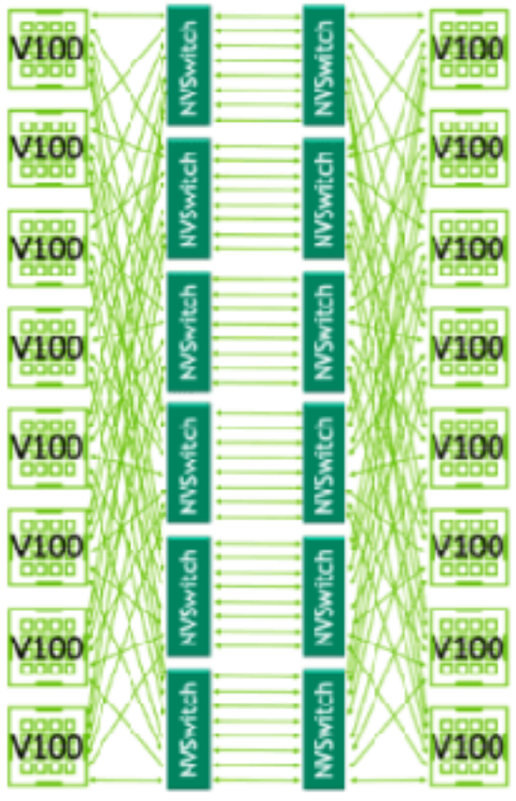
Edouard Oyallon

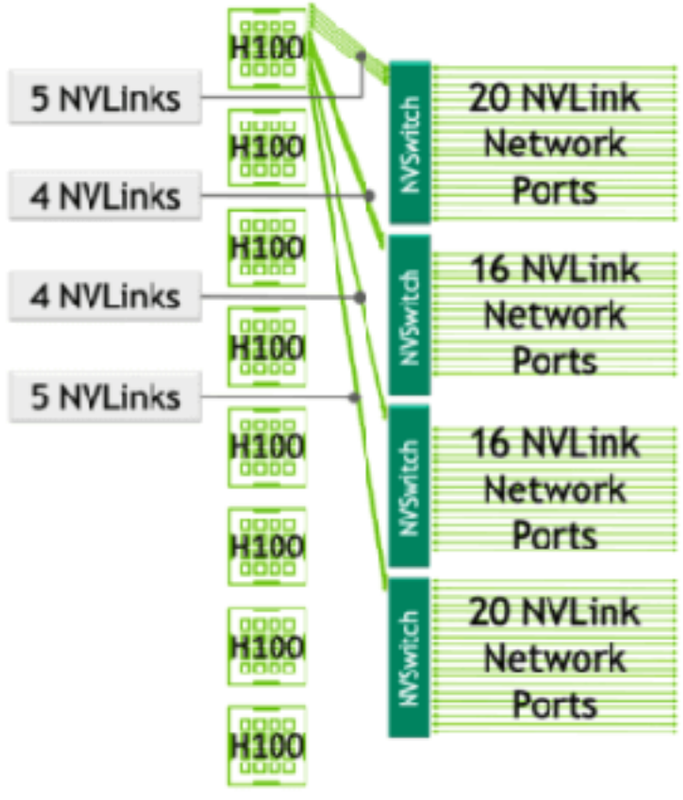
edouard.oyallon@cnrs.fr

Communication Bottlenecks in Modern Clusters

Specialized Hardware for Inter- and Intra-Node Communication.





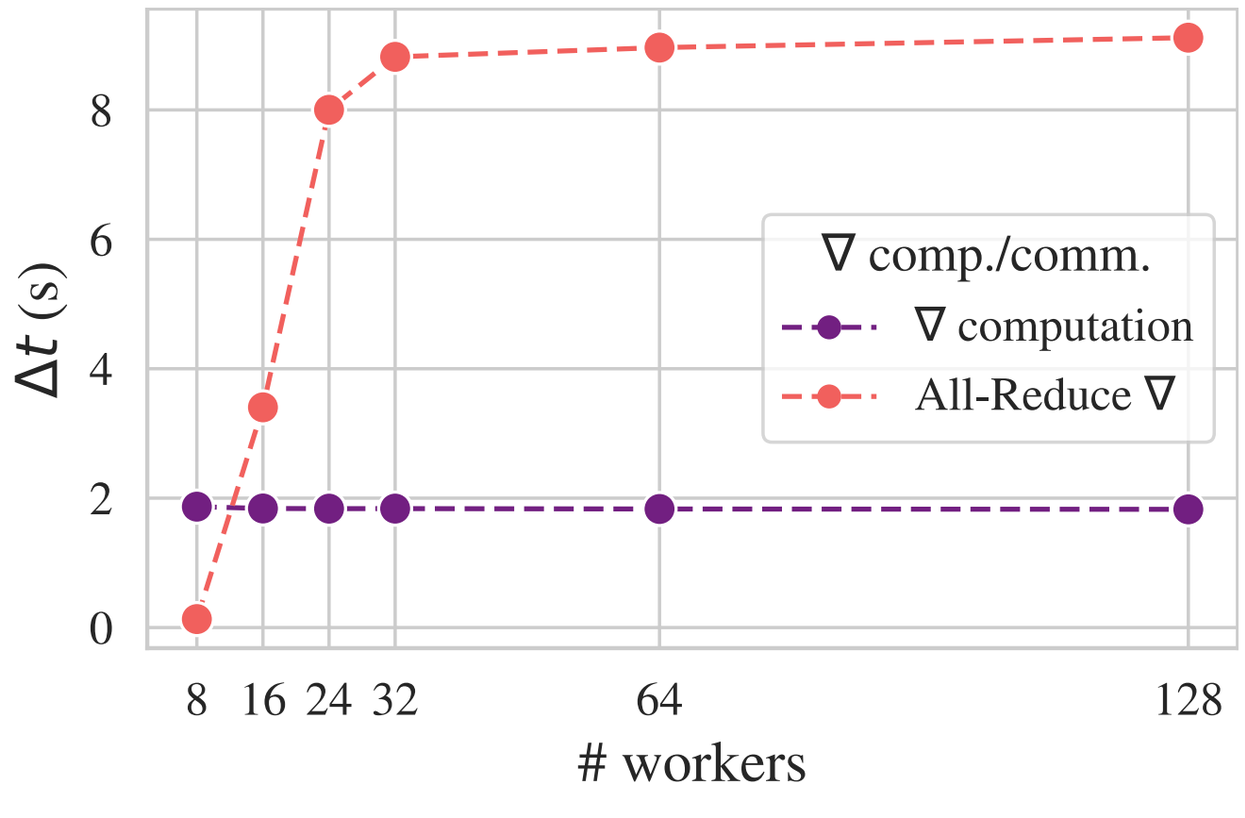


2016	2018	2020	2022
DGX-1 (P100)	DGX-2 (V100)	DGX A100	DGX H100
140GB/s Bisection BW 40GB/s AllReduce BW	2.4TB/s Bisection BW 75GB/s AllReduce BW	2.4TB/s Bisection BW 150GB/s AllReduce BW	3.6TB/s Bisection BW 450GB/s AllReduce BW

Overlapping Communication and Computation: Why It Matters

- Despite specialized hardware, inter- and intra-node communication remains a challenge.

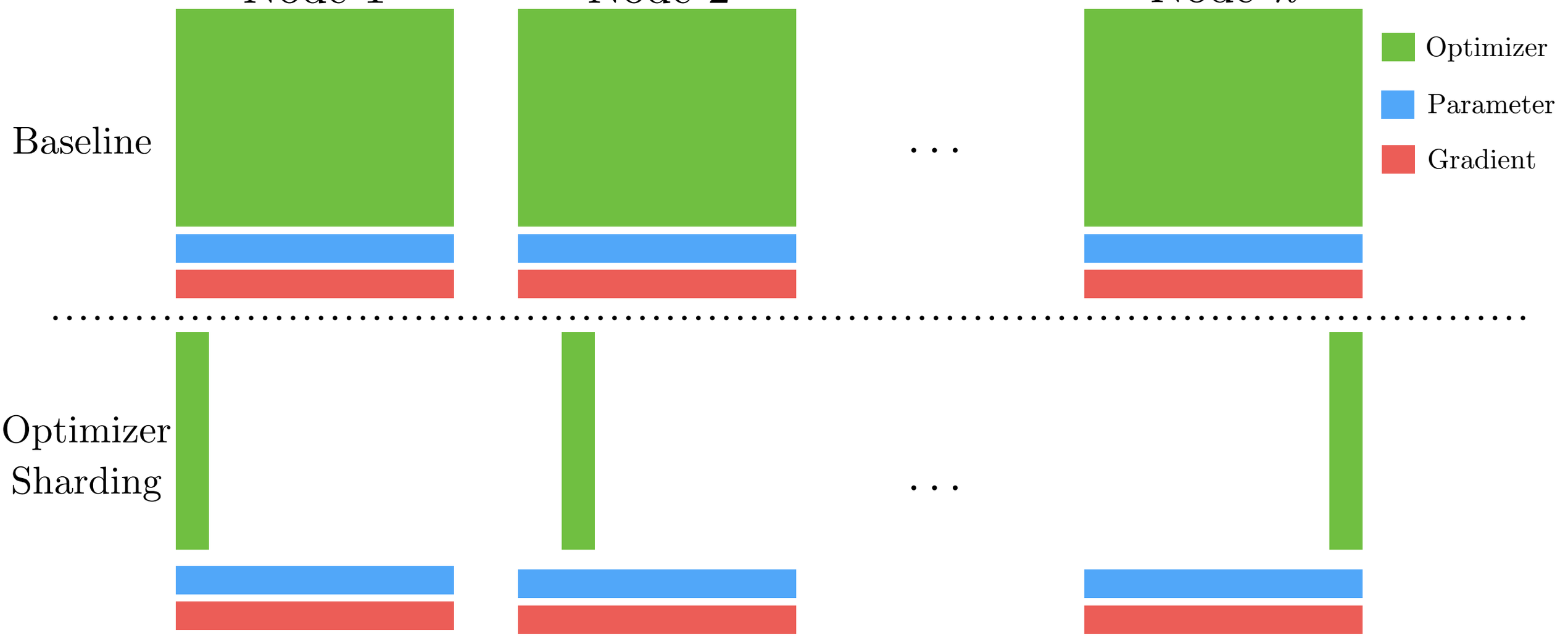
- **Experiment:** JeanZay (the national French AI GPUs-cluster) ; 8× A100 per node; 100 Gb/s inter-node, 300 Gb/s NVLink intra-node; using All-Reduce (no optimizer sharding) on a Llama-7B.



Past one worker, communication dominates.

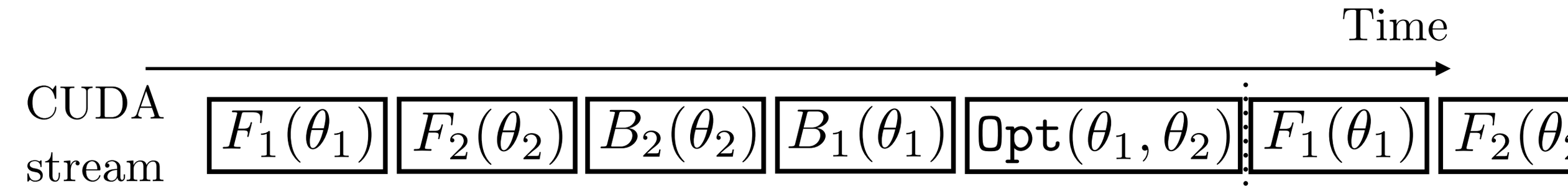
Sharding Strategies

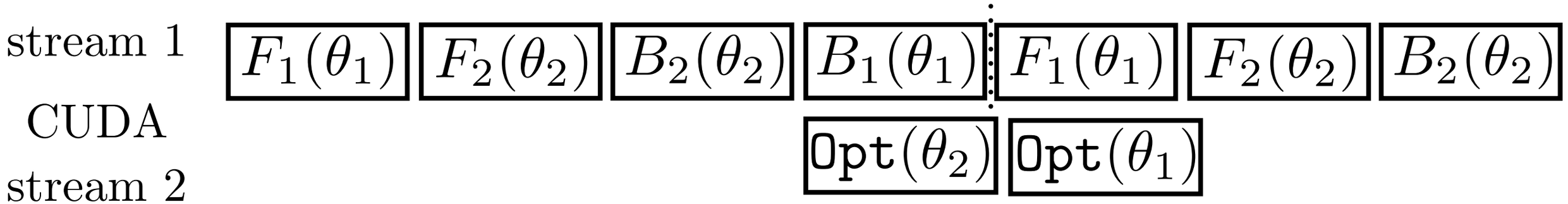
- To reduce memory footprint, optimizer sharding (FSDP/ZeRO) is common— but it **increases communication volume**.
- **With AdamW, optimizer states** (first/second moments in float32, etc.) add **6**$\times$ the parameter memory.
- **Approach:** shard states across workers and **gather/scatter on demand.** Feasible because optimizer updates are **pointwise.**



No Algorithmic Changes: Implementation-Level Comms/Compute Overlap

- Let a network perform a forward pass via F_1, F_2, a backward pass B_1, B_2 and an optimizer step via `Opt`.

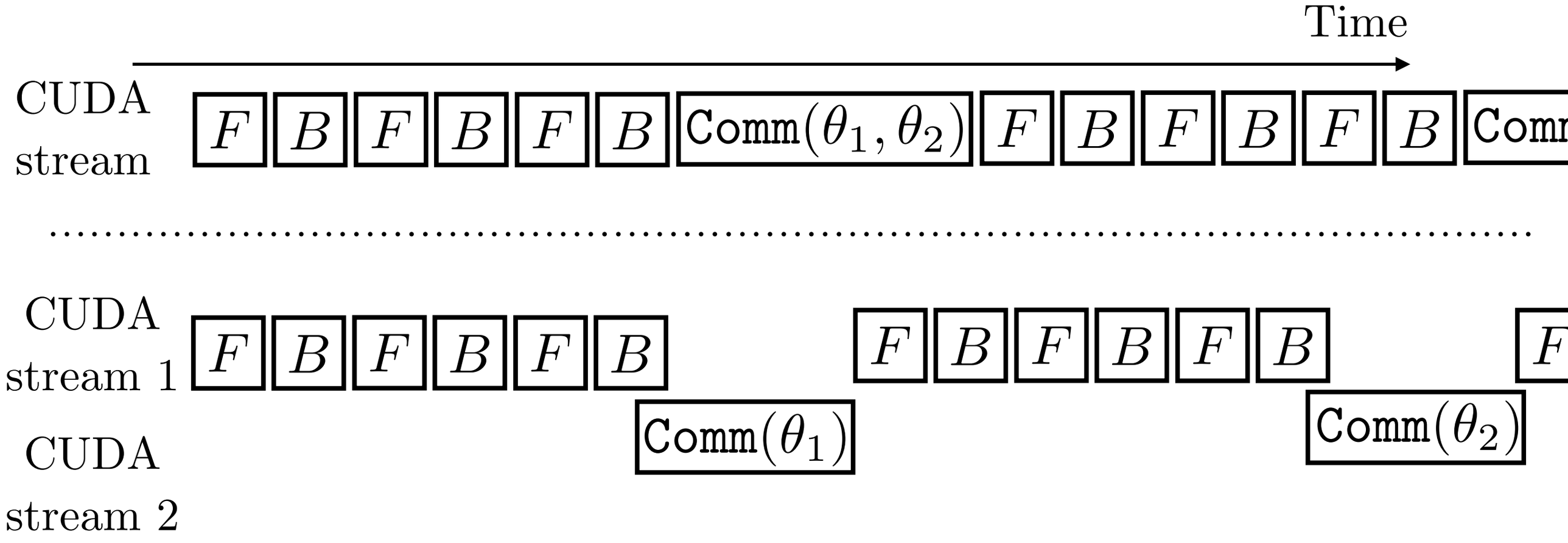




- **Overlap via hooks.** Using backward hooks, a standard technique is to launch communication for a layer's gradients immediately after that layer's backward computation completes, overlapping communication with remaining compute.

Local SGD

- In local SGD, workers run forward/backward locally and periodically all-reduce parameters (overlapping communication with compute at the implementation level)



- A strategy is **"streaming DiLoCo"**: periodically synchronize different subsets of parameters, to reduce local bandwidth, leading to stale updates.

Local SGD is incompatible with Sharding

- **Local SGD requires full state per worker:** Each worker must hold (or rematerialize) the entire model's parameters, gradients, and optimizer state between global syncs.

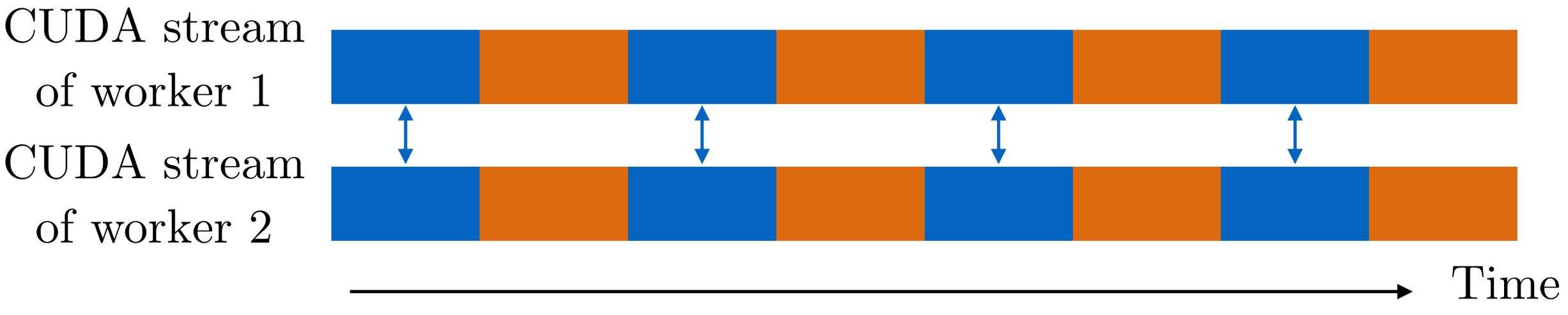
- **Sharding&local SGD:** Partitioning these states across replicas prevents true local SGD steps - unless workers have exceptionally fast interconnects.

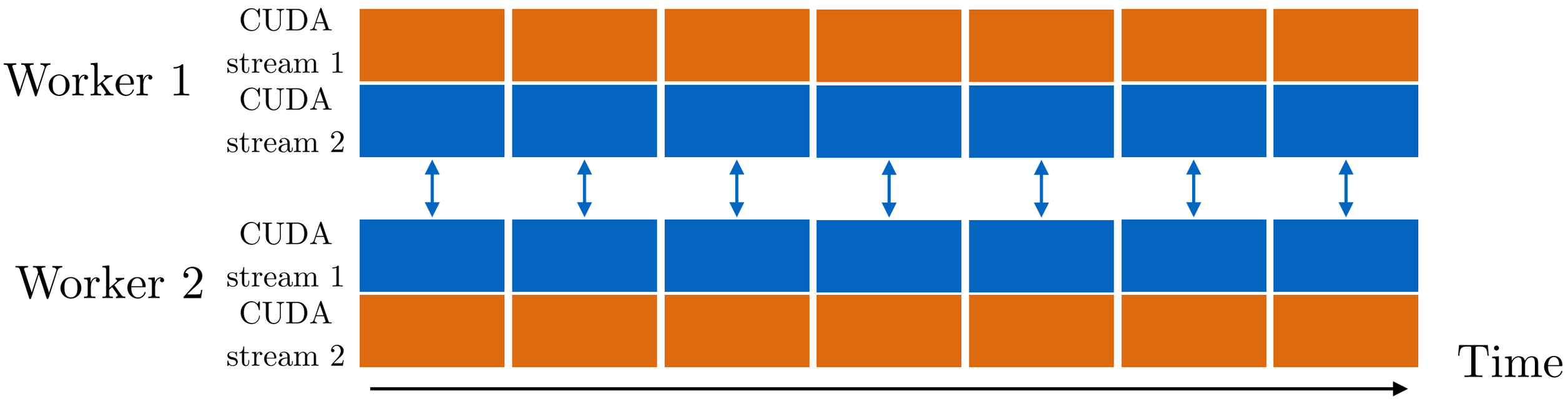
- **Practicality varies by cluster:** This can work on very large, well-connected clusters (e.g., Google TPU pods) but is impractical on typical setups like a single node with 8 well-connected GPUs.

Toward a principled overlapping of Comm. and Comp.:

ACCO

From Sequential to Overlapped Comm. & Comp.





Formal Setting

- **Objective.** Minimize $f : \mathbb{R}^d \to \mathbb{R}$ with access only to an *unbiased* stochastic gradient oracle:

$$\mathbb{E}_\xi[\nabla F(\theta, \xi)] = \nabla f(\theta)$$

- **Workers & heterogeneity**. We have N workers which process N_i samples in parallel:

$$\nabla F_i(\theta, \xi^i) = \frac{1}{N_i} \sum_{k=1}^{N_i} \nabla F(\theta, \xi_k^i) \quad \text{with } \xi^i = (\xi_1^i, ..., \xi_{N_i}^i) \sim \Xi_i$$

- Each worker has two concurrent blocking streams with read/write access to both memory buffers θ and $\tilde{\theta}$.

- This implies that we will study updates of the type:

$$\begin{cases} \theta^{(t+1)} = F_1(\theta^{(t)}, \tilde{\theta}^{(t)}) \\ \tilde{\theta}^{(t+1)} = F_2(\theta^{(t)}, \tilde{\theta}^{(t)}) \end{cases}$$

Baseline 1: All-Reduce GD

- Compute local gradients on each worker; **all-reduce** to average them; apply the `Optimizer` step:

$$\nabla F_i(\theta, \xi) = \frac{1}{N_i} \sum_{k=1}^{N_i} \nabla F(\theta, \xi_k)$$

$$g_i^{(t)} = \nabla F_i(\theta^{(t)}, \xi^{(t)}) \,,$$

$$\theta^{(t+1)} = \texttt{Opt}\left(\theta^{(t)}, \sum_{i=1}^{N} \frac{N_i}{\sum_i N_i} g_i^{(t)}\right)$$

The algorithm uses a synchronous all-reduce (no overlap) with good convergence.

Baseline 2: DPU

- **Delayer Parameter Updates (1-step):** two weight buffers—compute on $t-1$, write to t, then swap; overlap with one-step staleness.

$$g_i^{(t+1)} = \nabla F_i(\tilde{\theta}^{(t)}, \xi^{(t)})$$

$$\tilde{\theta}^{(t+1)} = \theta^{(t)}$$

$$\theta^{(t+1)} = \texttt{Opt}\left(\theta^{(t)}, \sum_{i=1}^{N} \frac{N_i}{\sum_{j=1}^{N} N_j} g_i^{(t)}\right)$$

/!\ However, it degrades convergence.

Baseline 3: WP

- **Weight Prediction (WP):** two buffers; one predicts a subset of parameters *one step ahead*.

$$g_i^{(t+1)} = \nabla F_i(\tilde{\theta}^{(t)}, \xi^{(t)})$$

$$\theta^{(t+1)} = \texttt{Opt}\left(\theta^{(t)}, \sum_{i=1}^{N} \frac{N_i}{\sum_i N_i} g_i^{(t)}\right),$$

$$\tilde{\theta}^{(t+1)} = \texttt{Opt}\left(\theta^{(t+1)}, \sum_{i=1}^{N} \frac{N_i}{\sum_i N_i} g_i^{(t)}\right).$$

/!\ However, it degrades convergence.

Toward ACCO

- **Idea.** ACCO starts from a decoupled formulation and shows that updates $\theta, \tilde{\theta}$ can be decoupled:

$$\inf_{\theta} f(\theta) = \inf_{\theta=\tilde{\theta}} \frac{1}{2}\big(f(\theta) + f(\tilde{\theta})\big).$$

- **Dynamic.** Straightforward derivations yield a simple update rule in the Gradient-Descent setting:

$$\begin{cases} \theta^{(t+1)} = \theta^{(t)} - \frac{\eta}{2}\left(\nabla f\left(\theta^{(t)}\right) + \nabla f\left(\tilde{\theta}^{(t)}\right)\right), \\ \tilde{\theta}^{(t+1)} = \boxed{\theta^{(t)}} - \eta \nabla f\left(\tilde{\theta}^{(t)}\right). \end{cases}$$

- **Consistency.** If $\theta^{(0)} = \tilde{\theta}^{(0)}$, ACCO reduces to standard GD:

$$\theta^{(t)} = \tilde{\theta}^{(t)}.$$

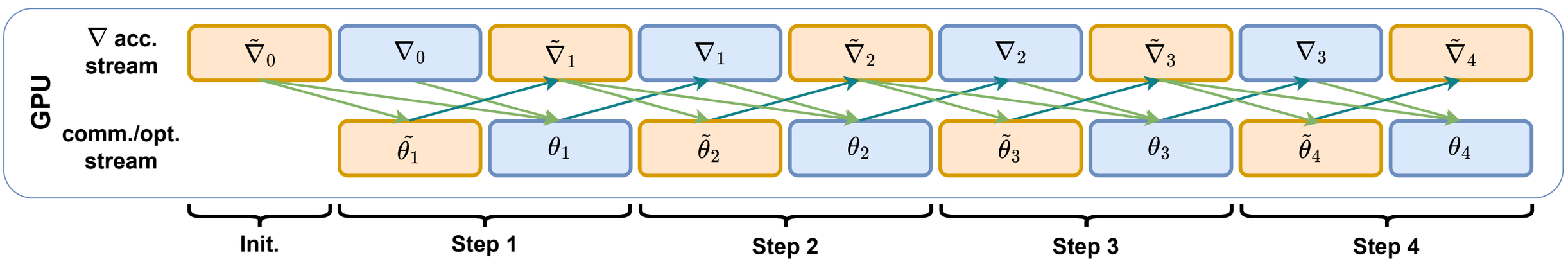
ACCO implementation

$$\begin{cases} \theta^{(t+1)} = \theta^{(t)} - \frac{\eta}{2}\left(\nabla f\left(\theta^{(t)}\right) + \nabla f\left(\tilde{\theta}^{(t)}\right)\right), \\ \tilde{\theta}^{(t+1)} = \theta^{(t)} - \eta \nabla f\left(\tilde{\theta}^{(t)}\right). \end{cases}$$

- We can rewrite ACCO in our setting as:

$$\begin{cases} g_i^{(t)} = \nabla F_i(\theta^{(t)}, \xi^{(t)}), \\ \tilde{\theta}^{(t+1)} = \texttt{Opt}\left(\theta^{(t)}, \sum_{i=1}^N \frac{N_i}{\sum_j N_j} \tilde{g}_i^{(t)}\right), \end{cases} \quad (2) \quad \begin{cases} \tilde{g}_i^{(t+1)} = \nabla F_i(\tilde{\theta}^{(t+1)}, \tilde{\xi}^{(t)}), \\ \theta^{(t+1)} = \texttt{Opt}\left(\theta^{(t)}, \sum_{i=1}^N \frac{N_i}{2\sum_j N_j} (g_i^{(t)} + \tilde{g}_i^{(t)})\right) \end{cases}$$

- Observe the effective parallelisation:



Convergence for GD and SGD.

- The proof relies on introducing the Lyapunov function:

$$V(\theta, \tilde{\theta}) \triangleq f(\theta) - f(\theta^*) + \eta L(f(\tilde{\theta}) - f(\theta^*)) + L\|\theta - \tilde{\theta}\|^2 \geq 0$$

- **Theorem**: If f is L-smooth, and if $0 < \eta \leq \frac{1}{2L}, T > 0$, then:

$$\frac{1}{T}\sum_{t=0}^{T-1}\left(\|\nabla f(\theta_t)\|^2 + \|\nabla f(\tilde{\theta}_t)\|^2\right) \leq \frac{8}{\eta T}\left(f(\theta_0) + f(\tilde{\theta}_0) - 2f(\theta^*) + L\|\theta_0 - \tilde{\theta}_0\|^2\right)$$

- Furthermore, if $\begin{cases} \theta^{(t+1)} = \theta^{(t)} - \frac{\eta}{2}(g_t + \tilde{g}_t), \\ \tilde{\theta}^{(t+1)} = \theta^{(t)} - \eta\,\tilde{g}_t, \end{cases}$ so that $\{g_t, \tilde{g}_t\}$

are conditionally independent with variance σ^2, and if

$$\mathbb{E}[g_t \mid \theta^{(t)}] = \nabla f(\theta^{(t)}), \mathbb{E}[\tilde{g}_t \mid \tilde{\theta}^{(t)}] = \nabla f(\tilde{\theta}^{(t)}),$$

then:

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\Big[\|\nabla f(\theta_t)\|^2 + \|\nabla f(\tilde{\theta}_t)\|^2\Big] \leq \frac{16}{\eta T}\big(f(\theta_0) - f(\theta^*)\big) + 8\,\sigma^2 L\eta.$$

Numerical analysis of ACCO

Caution: Local SGD Can Bite

K: memory multiplier of the optimizer

N: numbers of workers

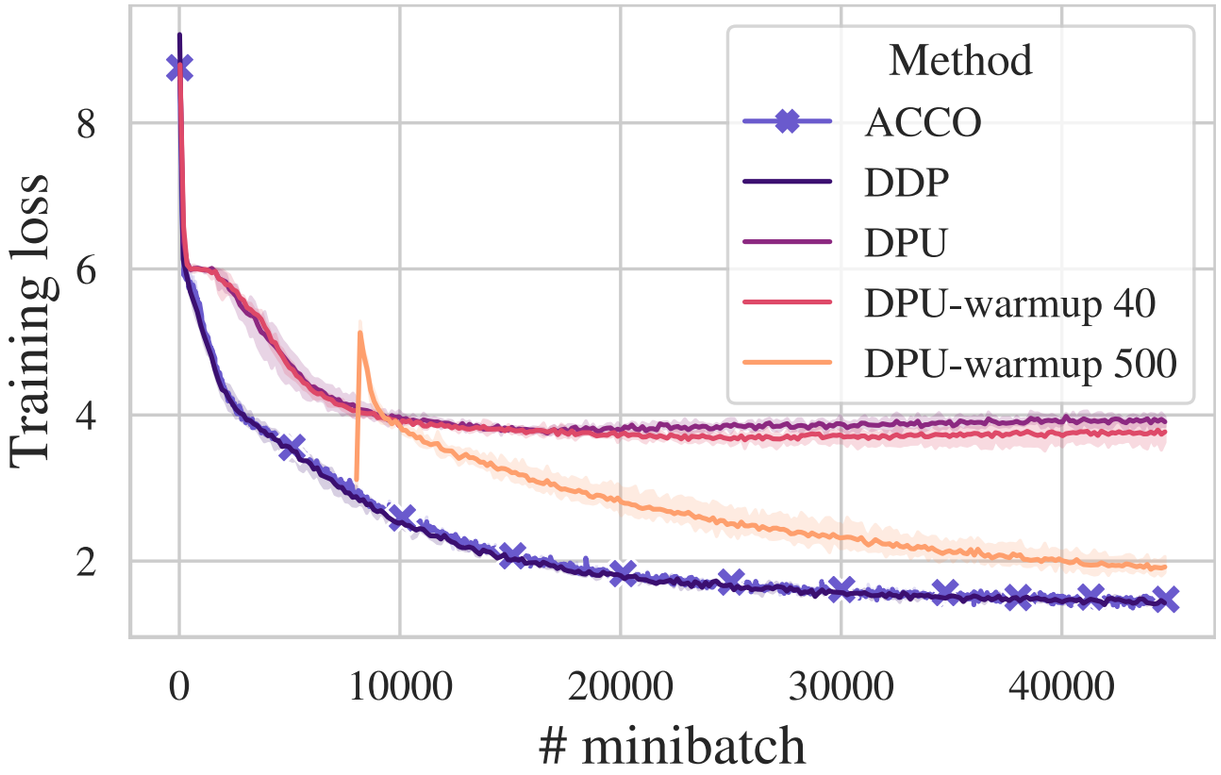
Ψ: numbers of parameters

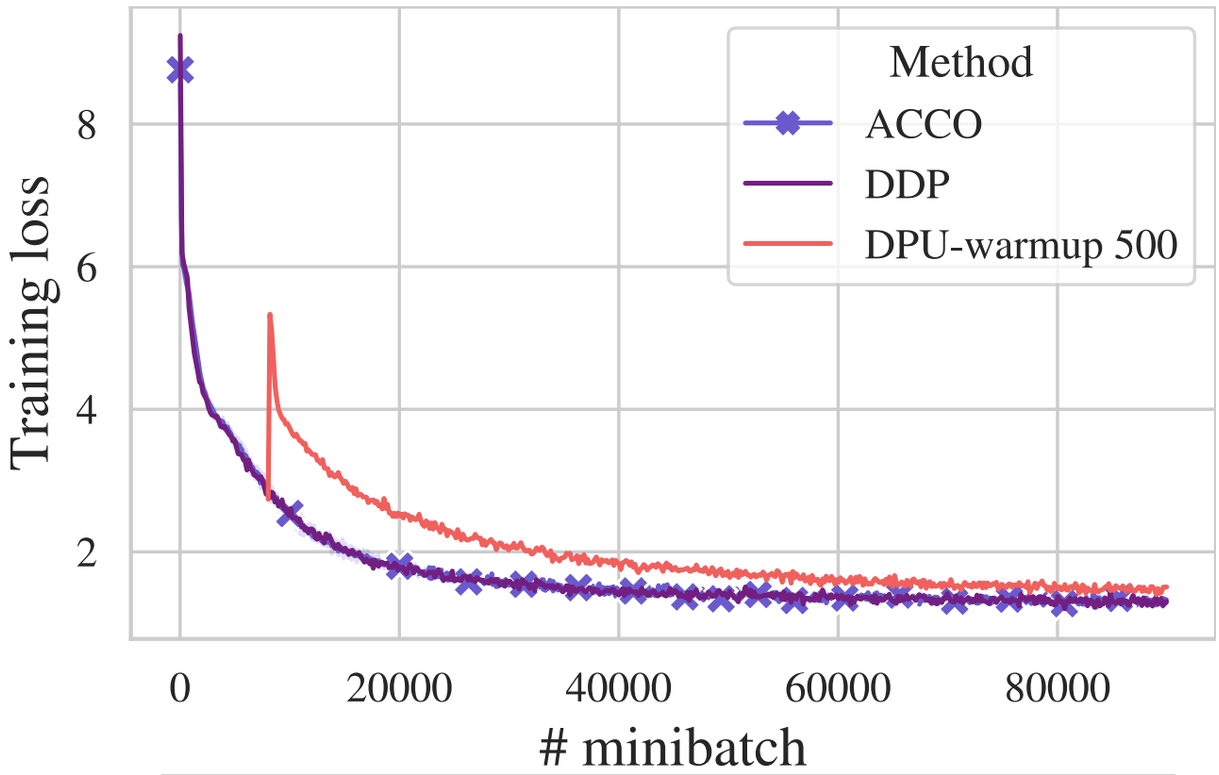
We must communicate state to perform optimizer steps — this undermines the gains from sharing settings.

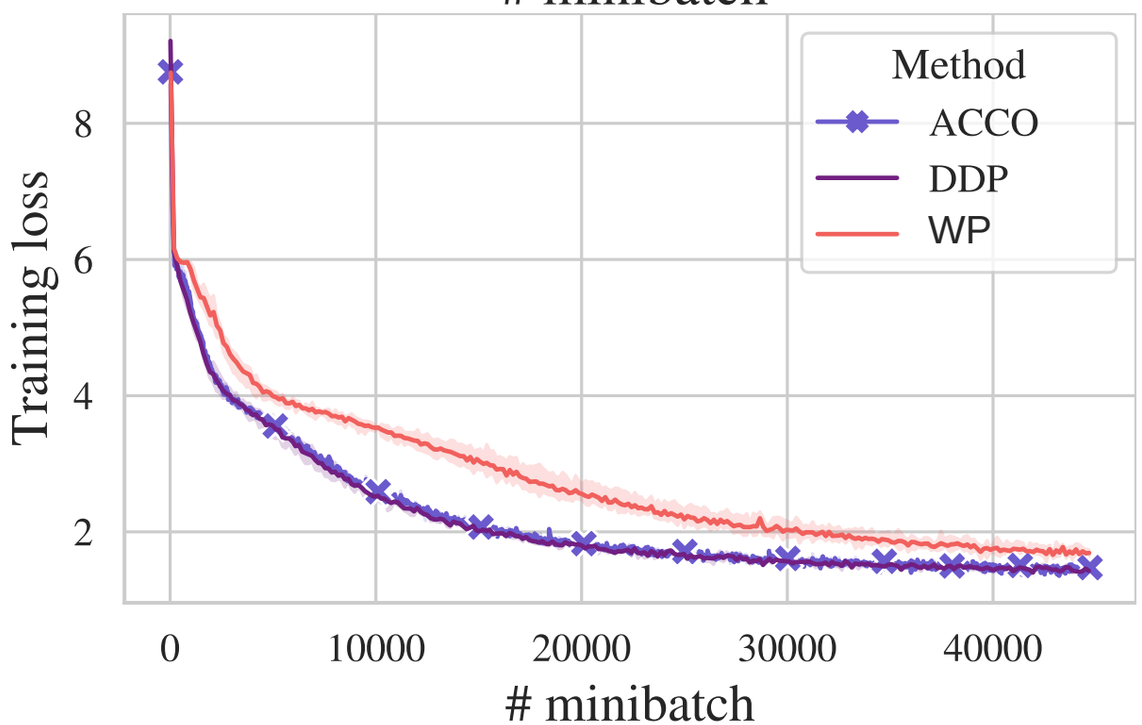
Method	**Overlap comm/comp**	**Hetero. hardware**	**No outer loop**	**Convergence Rates**	**Memory per replicas** $(K, N, \Psi)=(12, 64, 7.5\text{B})$
DDP [31]	✗	✗	✓	✓	$(2+2+K)\Psi = 120$ GB
ZeRO-1 [55]	✗	✗	✓	✓	$(2+2+\frac{K}{N})\Psi = 31$ GB
SlowMo [75]	∼	✗	✗	∼	$(2+2+2\times2+K)\Psi = 150$ GB
DiLoCo [14]	✓	✗	✗	?	$(2+2+2\times2+K)\Psi = 150$ GB
CO2 [69]	✓	✗	✗	✓	$(2+2+4\times2+K)\Psi = 180$ GB
DPU [60]	✓	✗	✓	✗	$(2+2+2+\frac{K}{N})\Psi = 46$ GB
WP [8]	✓	✗	✓	✗	$(2+2+2+\frac{K}{N})\Psi = 46$ GB
ACCO (Ours)	✓	✓	✓	✓	$(2+2+2+\frac{K}{N})\Psi = 46$ GB

Does it work? Yes!

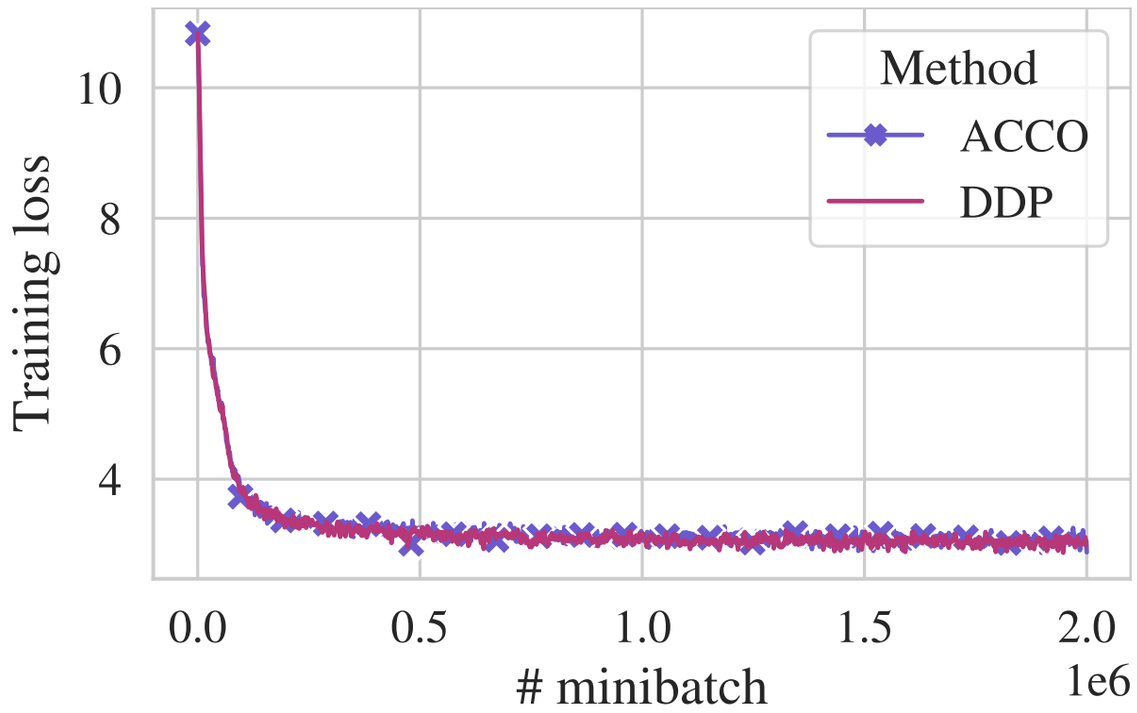
- TinyStories + 36M-parameter GPT-Neo-based transformer with 32 workers on 5B tokens:

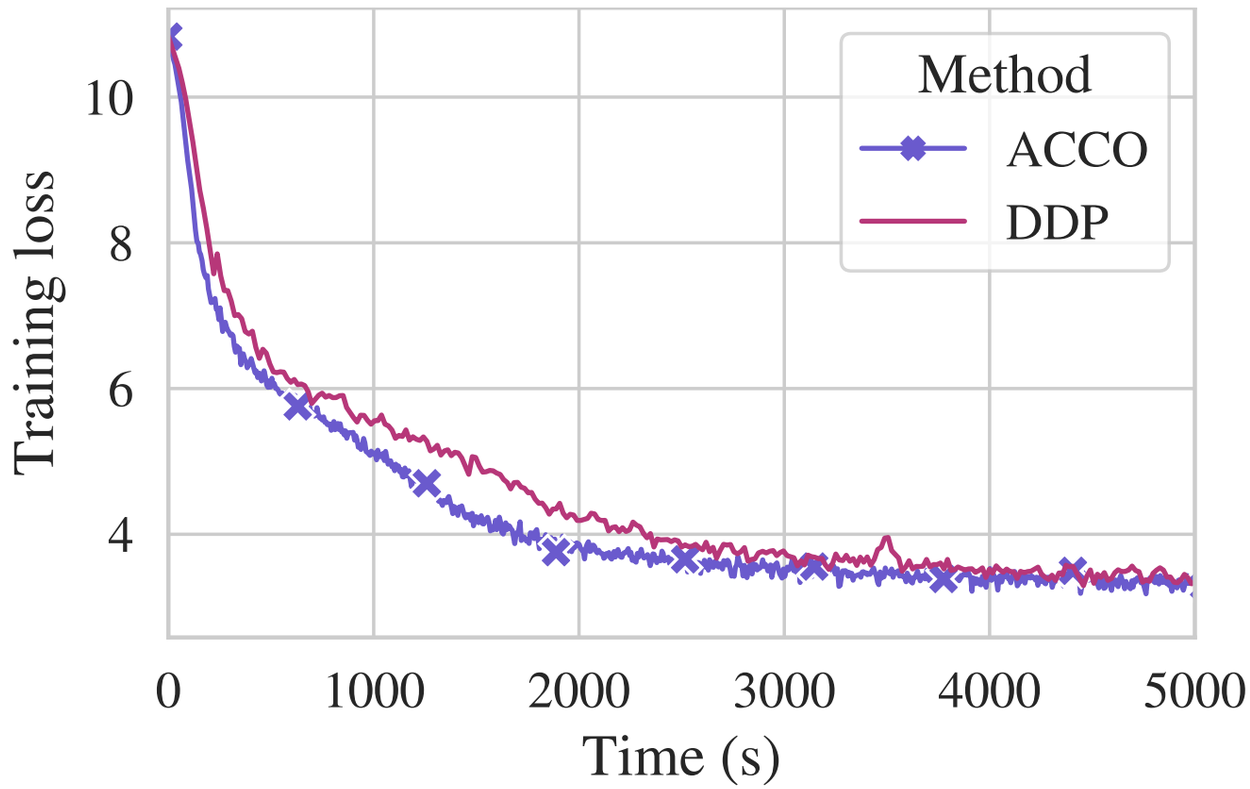






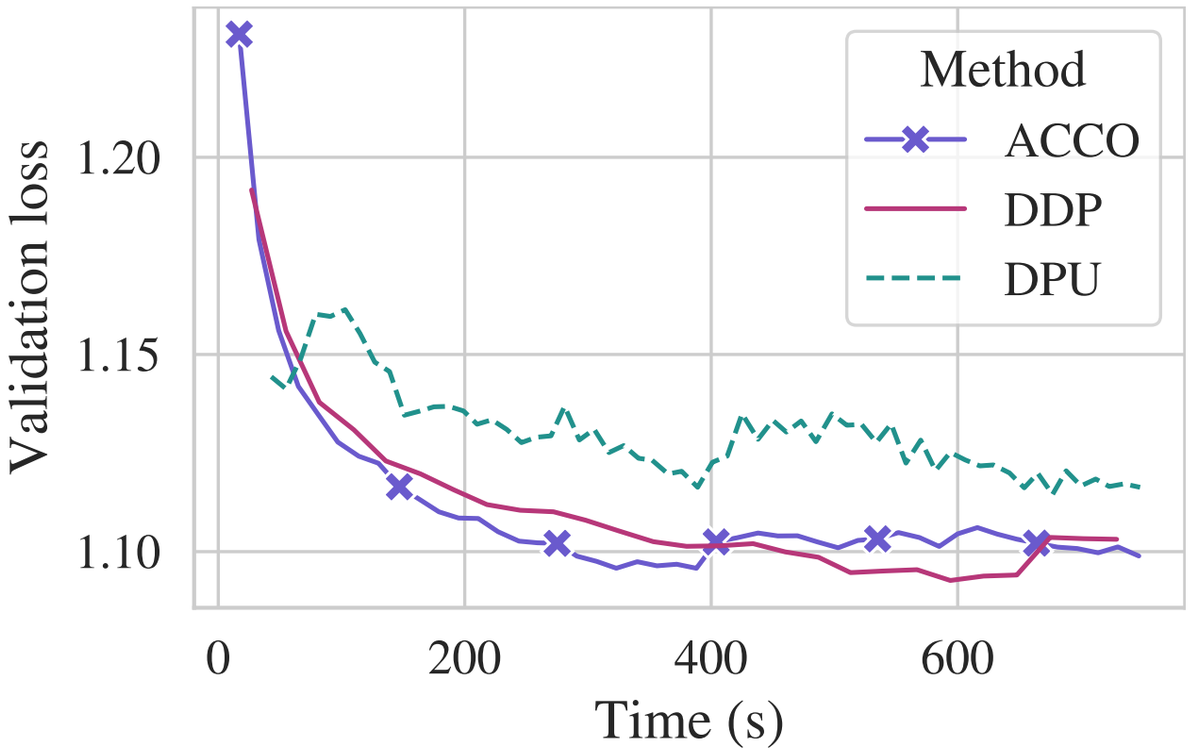
Timing?

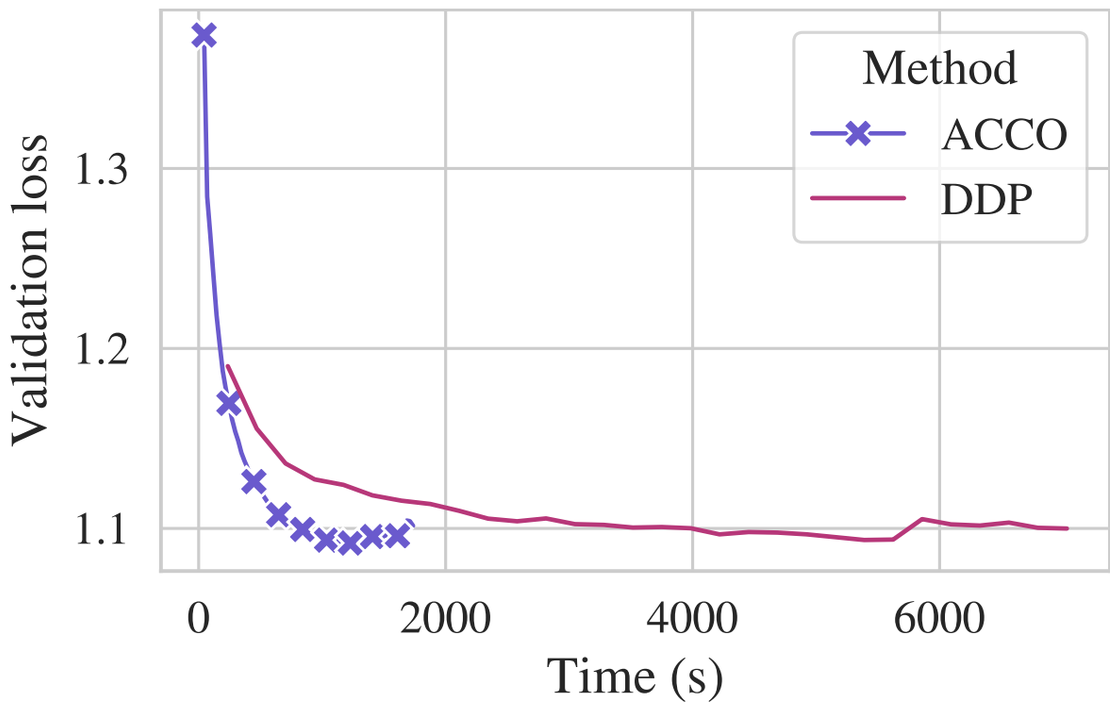

Training loss
10
8
6
4
0.0
0.5
1.0
1.5
2.0
1e6
minibatch
Method
ACCO
DDP


Training loss
10
8
6
4
0
1000
2000
3000
4000
5000
Time (s)
Method
ACCO
DDP

Fine tuning

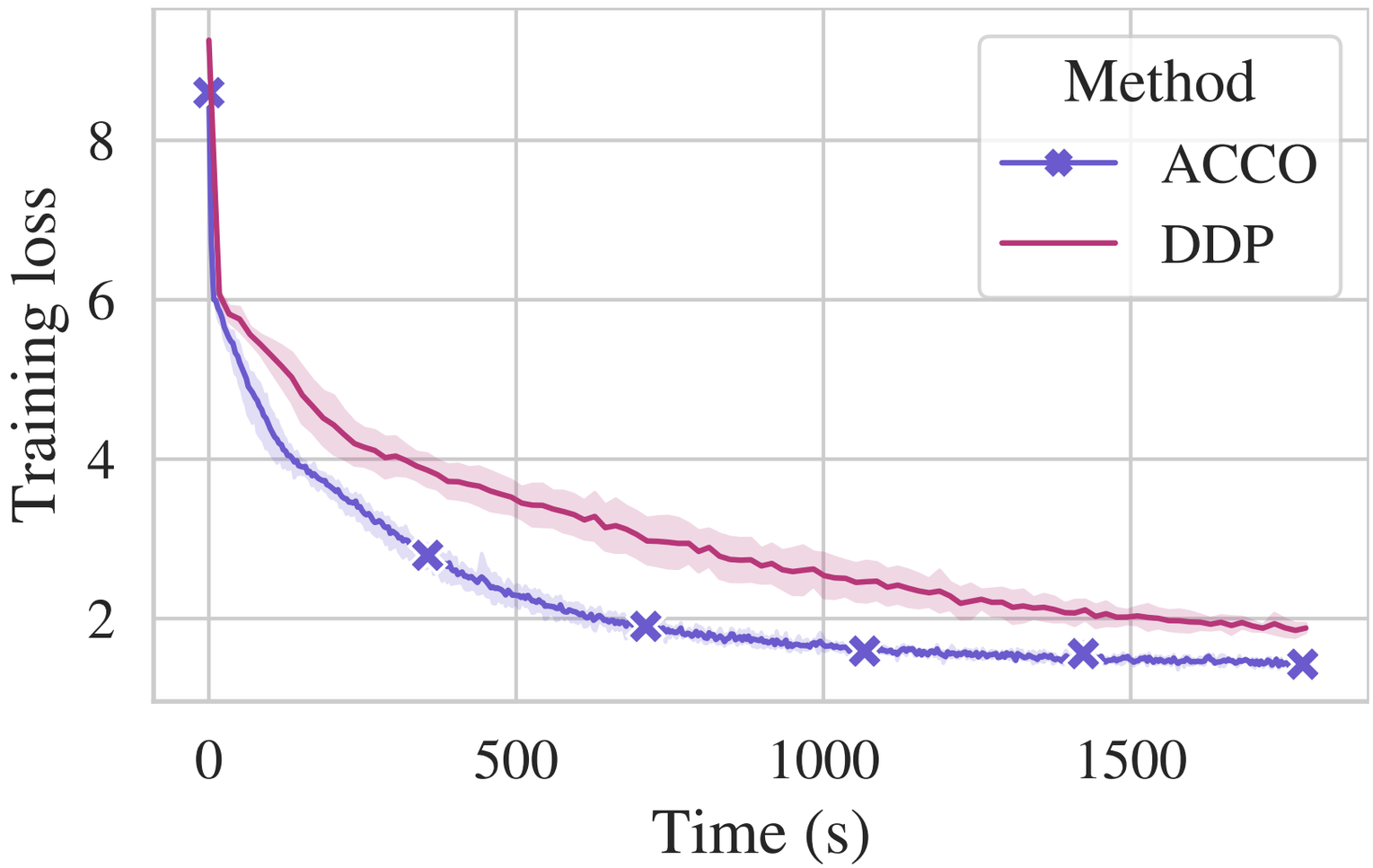
- On the Alpaca dataset, for a GPT-Neo 2.7B models, retrained on Piles: (mini-batches have heterogeneous sizes: more sync times for DDPs)





Robustness to straggler workers:

- TinyStories with 1 worker x4 slower:



Conclusion

- Overlapping can be neatly done by using a **novel** dynamic.

- Looking for efficient pipelining for LLMs? Check PETRA! (*spotlight* at ICLR2025)

- More papers on distributed optimization: https://edouardoyallon.github.io/

Thank you!