Delayed Gradient Averaging: Tolerate the Communication Latency for Federated Learning

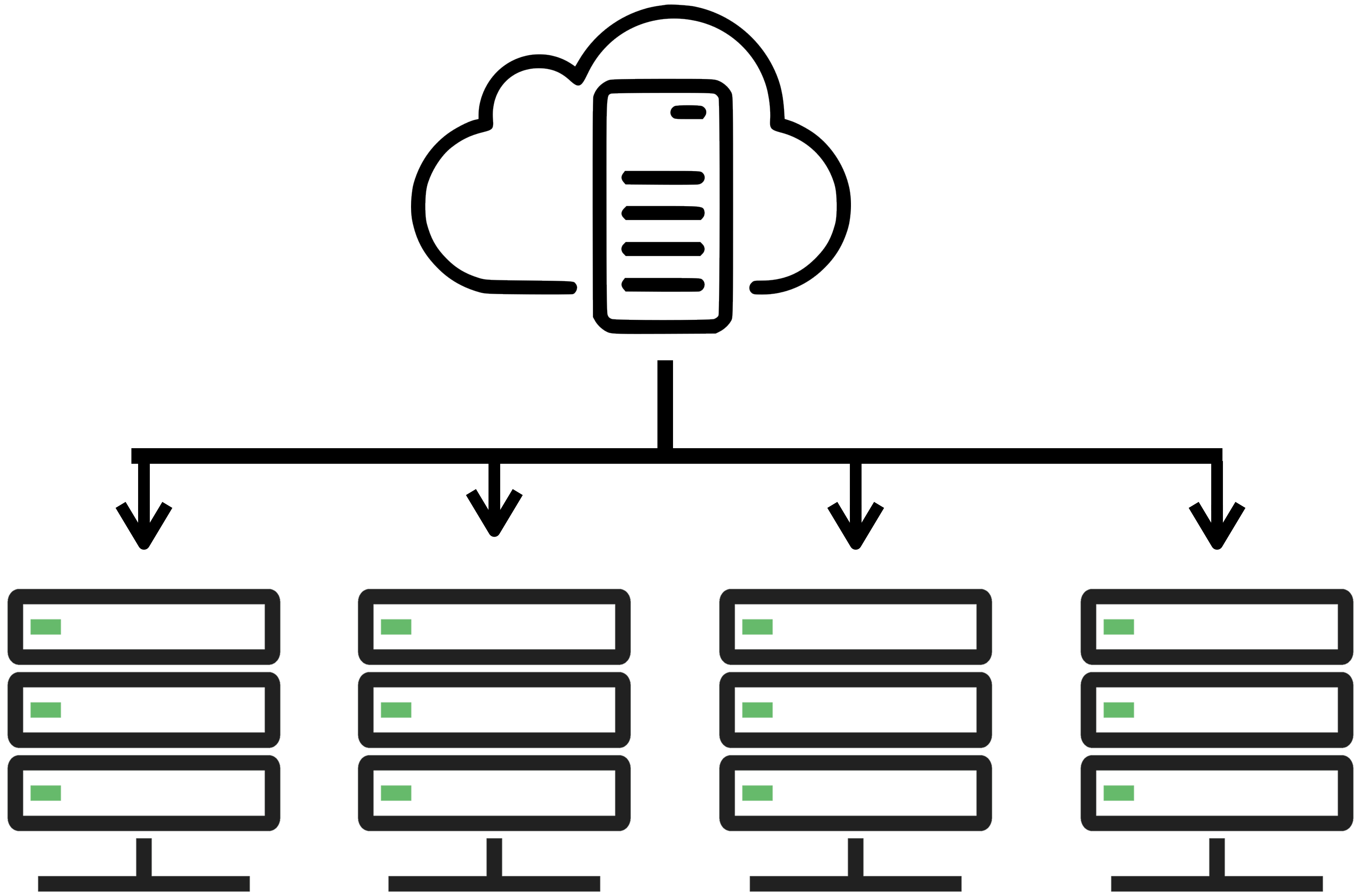
Ligeng Zhu, Hongzhou Lin, Yao Lu*, Yujun Lin, Song Han

Massachusetts Institute of Technology, Google*

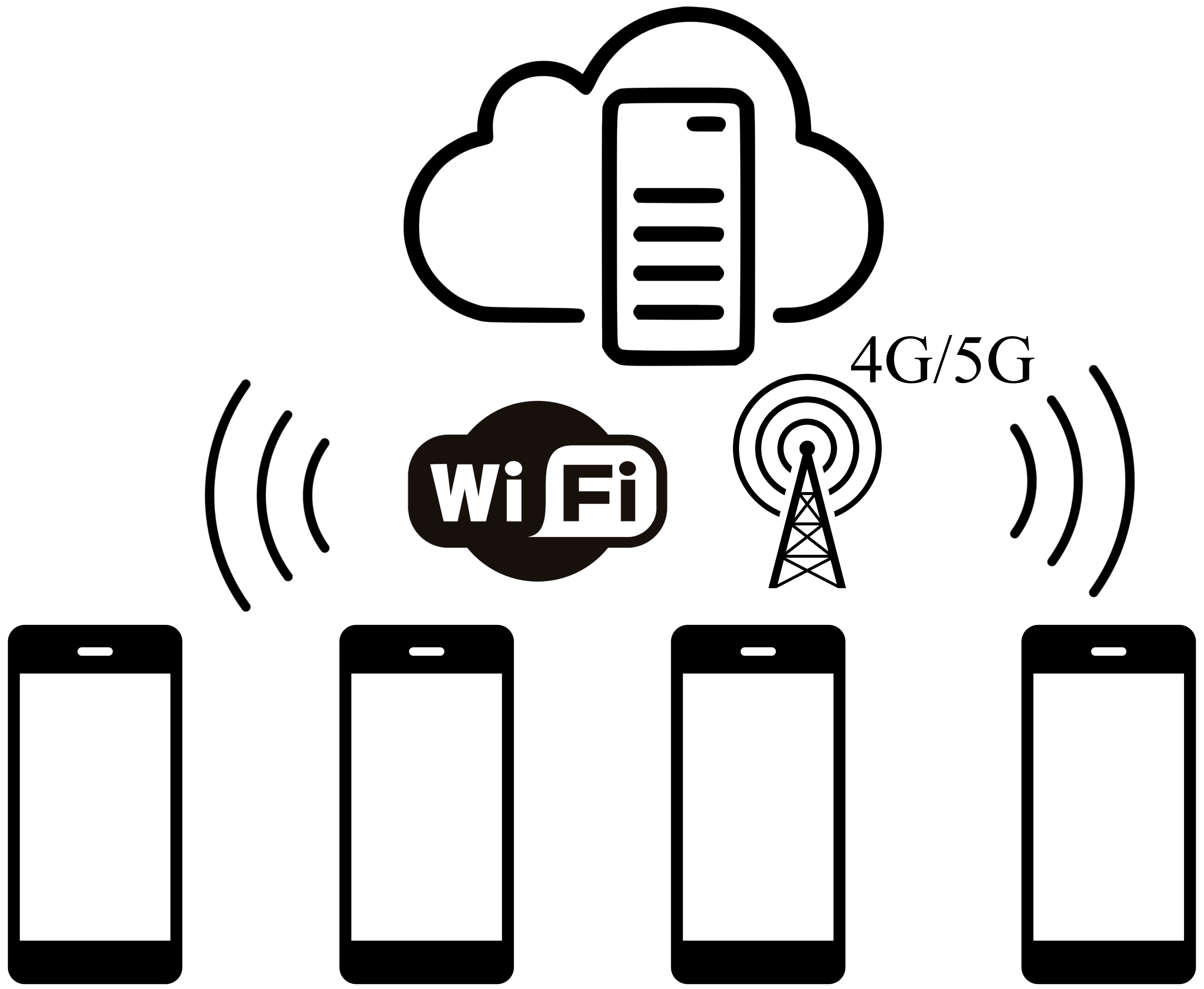
Federated Learning Allows Training without Sharing

- **Security**: Data never leaves devices thus promises security and regularization.
- **Customization**: Models continually adapt to new data from the sensors.

Difference between Distributed Training and Federated Learning



Connected through wired ethernet or infinity band
Bandwidth as high as **100Gb/s**, Latency as low as **1us**

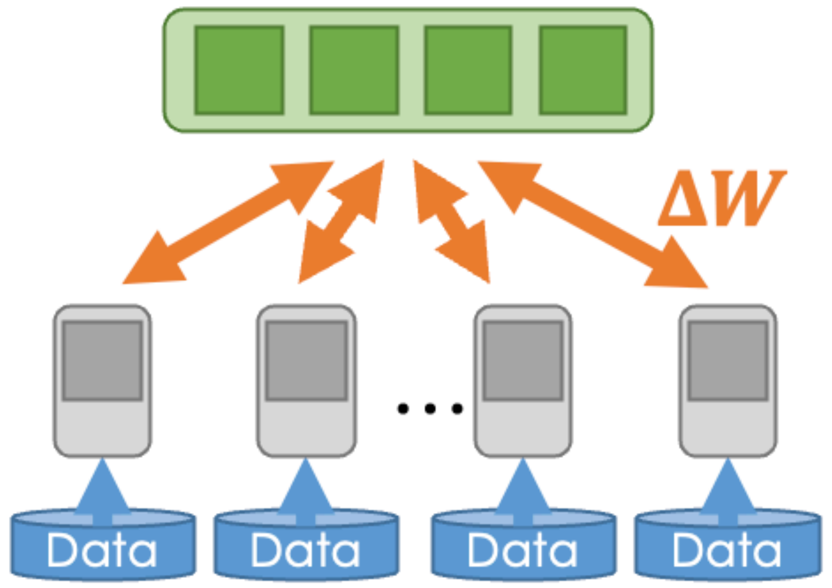


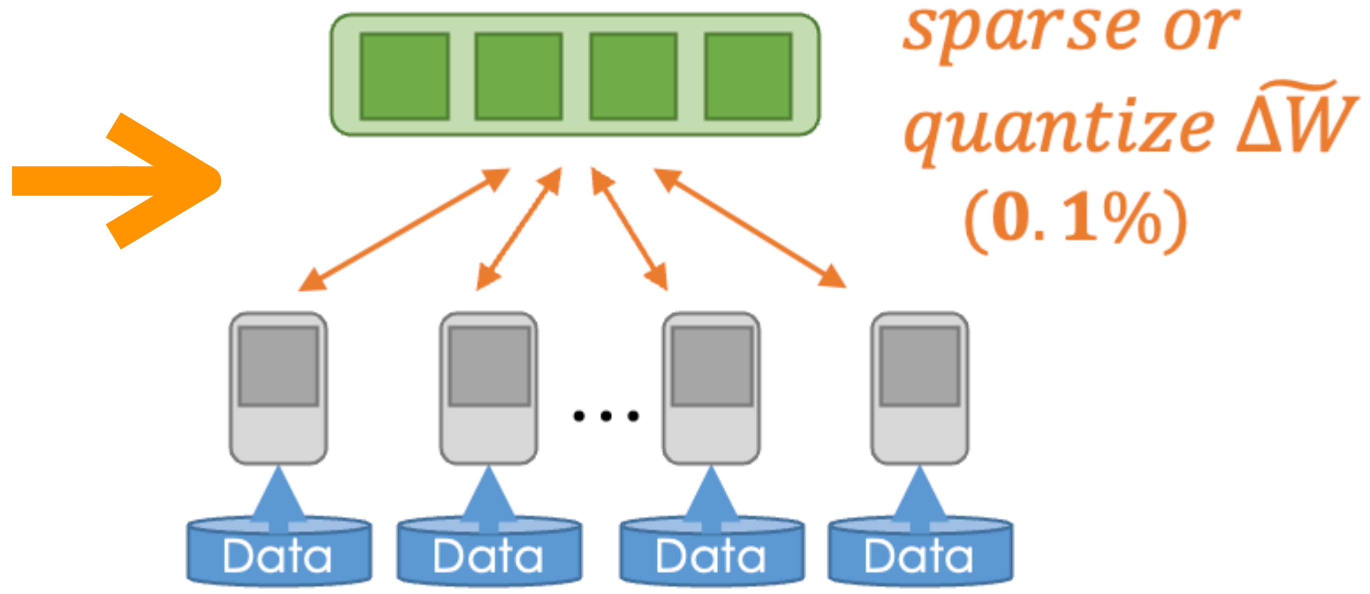
Connected through WiFi or Cellular network
Bandwidth up to **1Gb/s**, Latency **~200ms**.

There is **huge gap between** the network connection of conventional distributed training and federated learning

Network Bottleneck in Federated Learning

- Bandwidth can be always improved by
  - Hardware upgrade
  - Gradient compression[1] and quantization[2]

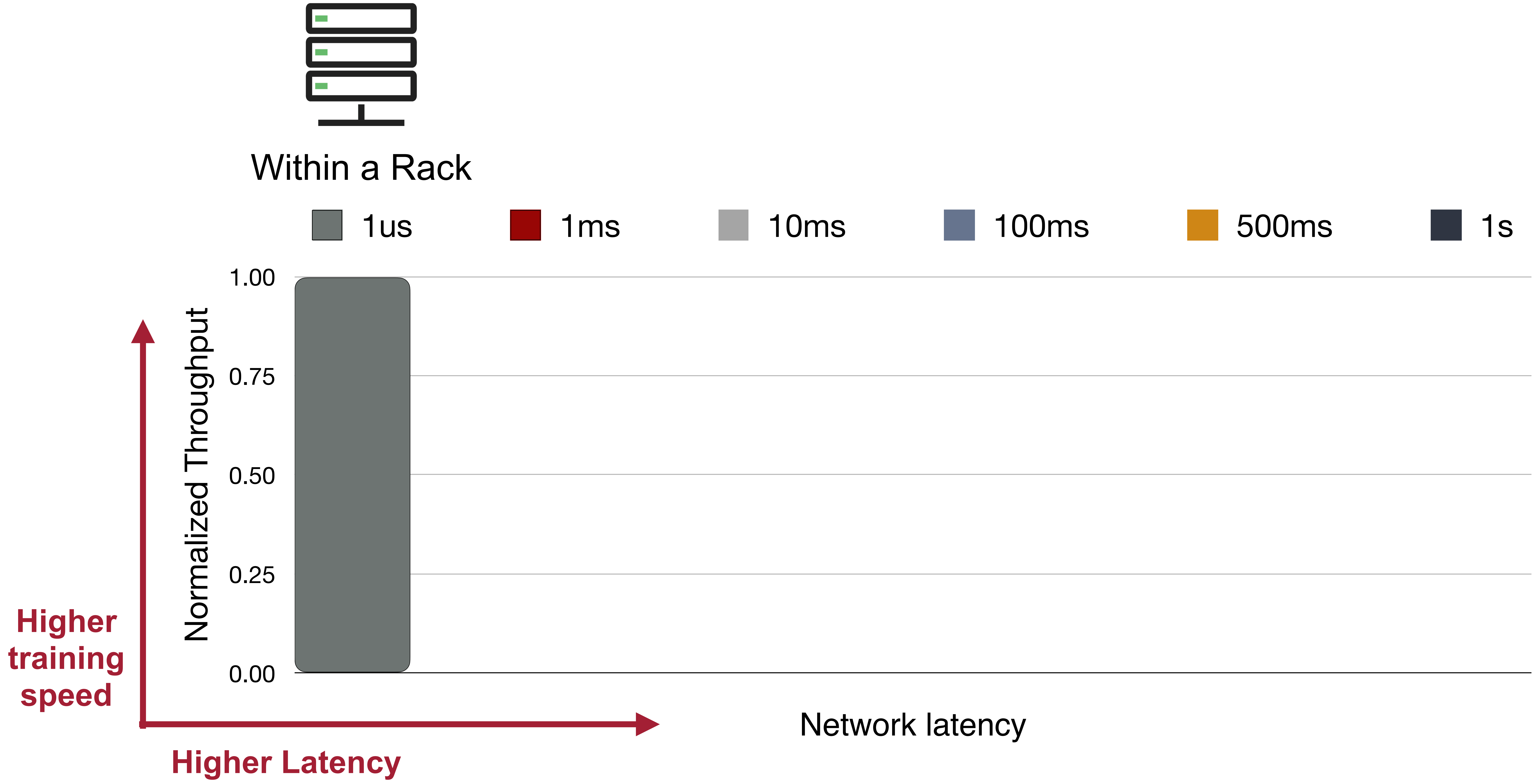




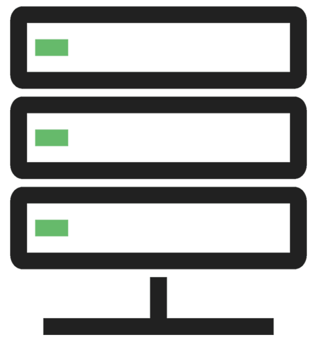
- Latency **is hard to improve** because
  - Physical limits: Shanghai to Boston, even considering the speed of light, still takes 162ms.
  - Signal congestion: Urban office and home creates a lot of signal contention.

[1] Deep Gradient Compression: Reducing the Communication Bandwidth for Distributed Training
[2] 1-Bit Stochastic Gradient Descent and Application to Data-Parallel Distributed Training of Speech DNNs

High Latency Slows Federated Learning

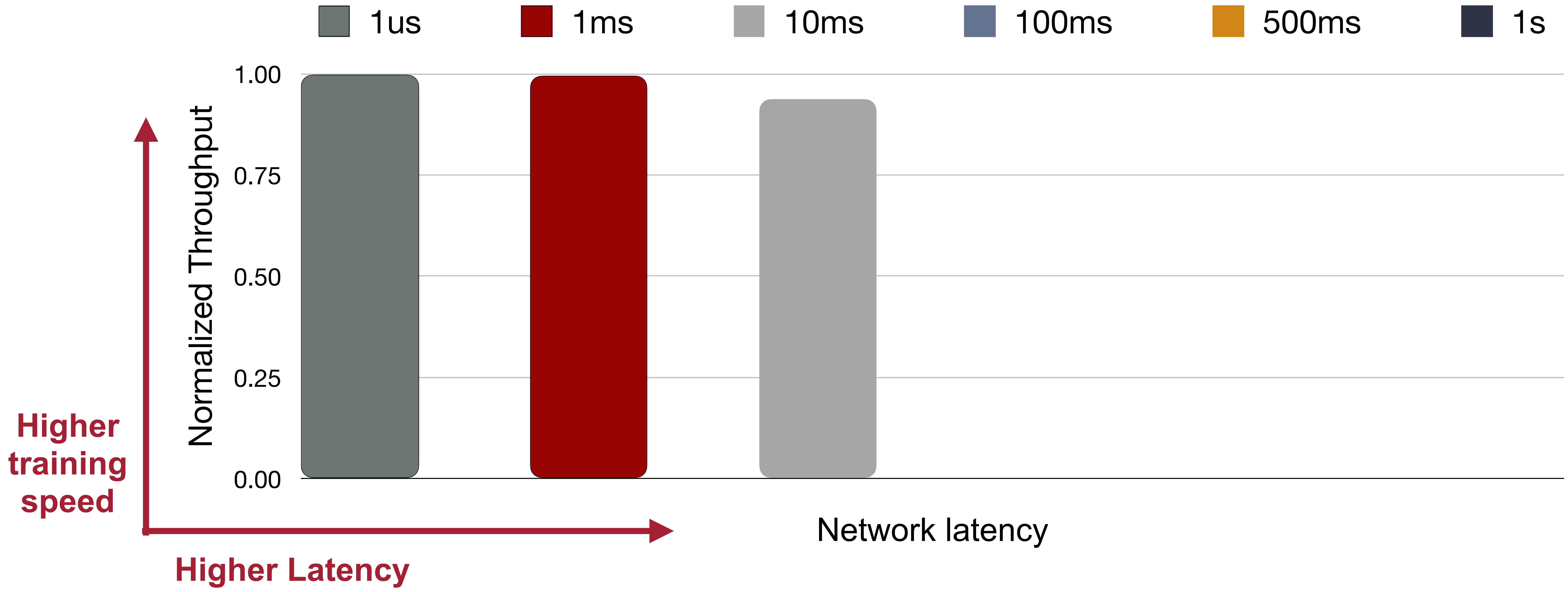


High Latency Slows Federated Learning



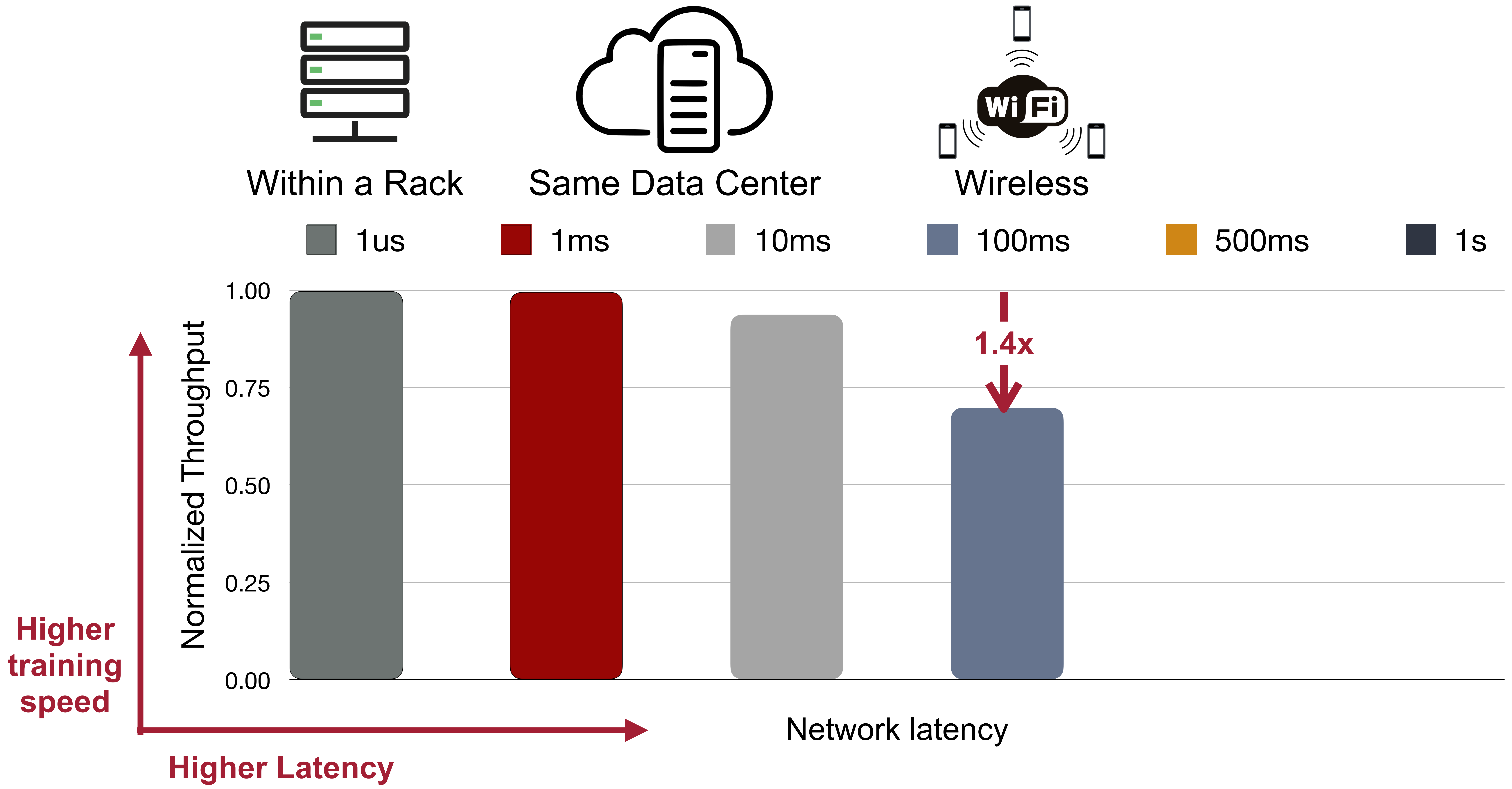
Within a Rack

Same Data Center



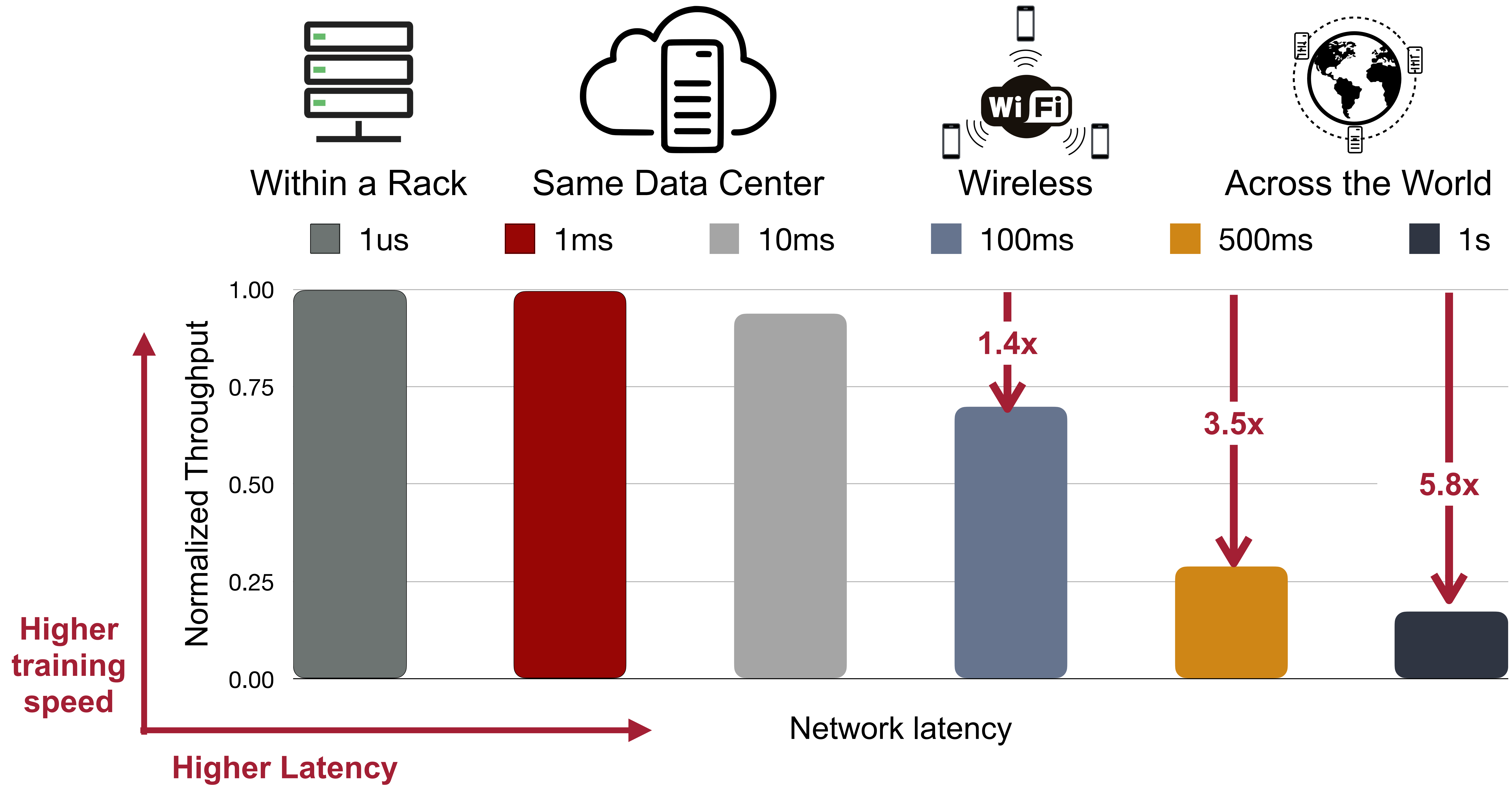
In cluster network latency does not affect training

High Latency Slows Federated Learning



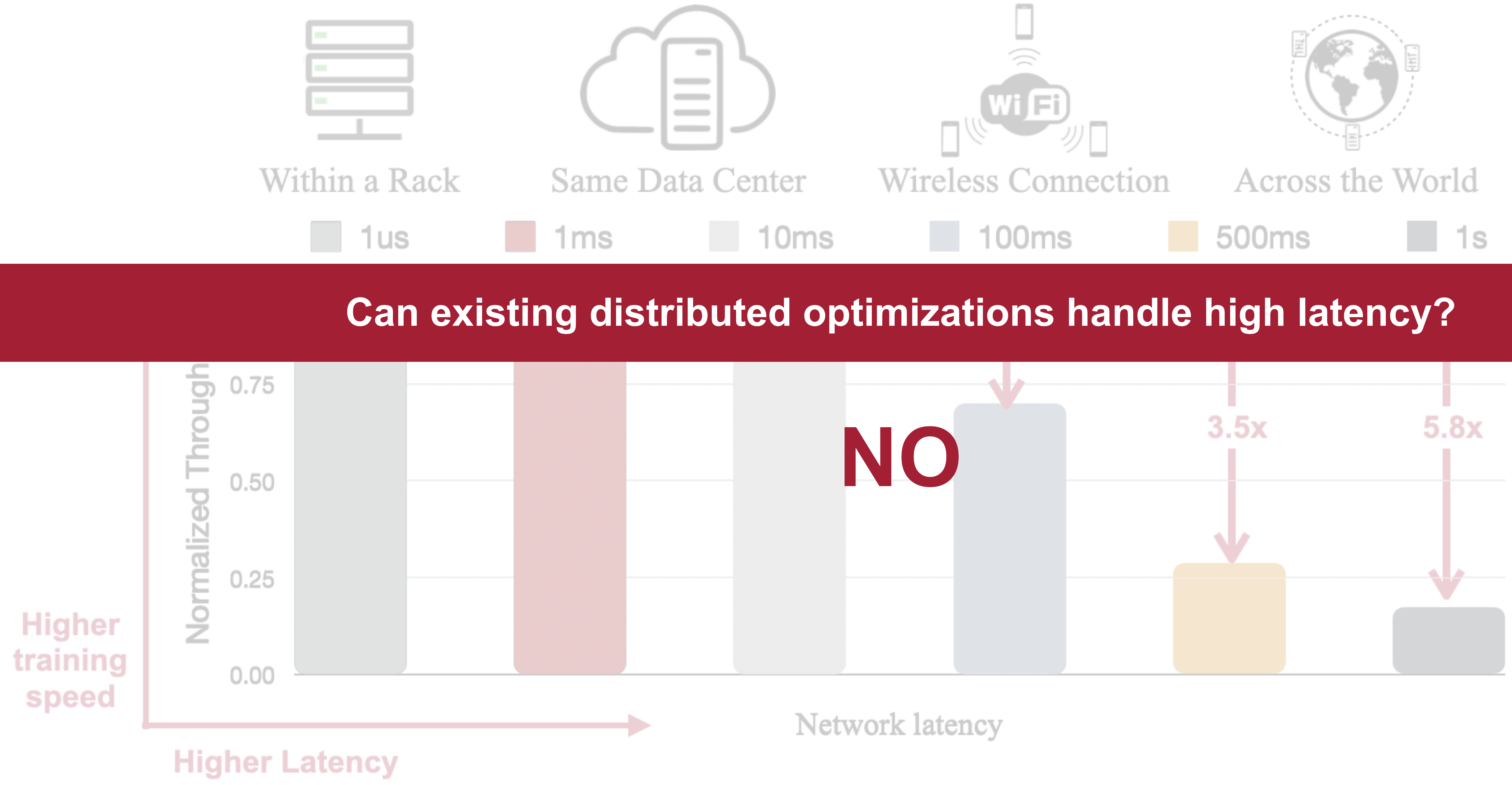
In home wireless connection slows the training by certain margin.

High Latency Slows Federated Learning



Long-distance connection slows the training by a large margin.

High Latency Slows Federated Learning



Can existing distributed optimizations handle high latency?

NO

Conventional Algorithms Suffer from High Latency

Distributed Synchronous SGD

1. Sample and calculate $\nabla w_{(i,j)}$
2. Send $\nabla w_{(i,j)}$ to other nodes
3. Recv $\nabla w_{(i,j)}$ from other nodes
4. $\overline{\nabla w_{(i)}} = \frac{1}{J} \sum_{j=1}^{J} \nabla w_{(i,j)}$
5. $w_{(i,j)} = w_{(i,j)} - \eta \overline{\nabla w_{(i)}}$

Local updates and communication are performed sequentially.
Worker **<u>has to wait the transmission finish</u>** before next step.

Computation Communication

i: iteration, j: work index, x: training data, w: model weights

Conventional Algorithms Suffer from High Latency

Federated Averaging [McMahan 16]

1. Sample and calculate $\nabla w_{(i,j)}$
2. If i mod K:
    1. Send $\nabla w_{(i,j)}$ to other nodes
    2. Recv $\nabla w_{(i,j)}$ from other nodes
    3. $G_i = \frac{1}{J}\sum_{j=1}^{J} \nabla w_{(i,j)}$
3. Else
    1. $G_i = \nabla w_{(i,j)}$
4. $w_{(i,j)} = w_{(i,j)} - \eta G_i$

Increase K (K=2 in the example) can **amortize the effect**, but the training still **slows when latency is high**.

Computation   Communication

i: iteration, j: work index, x: training data, w: model weights

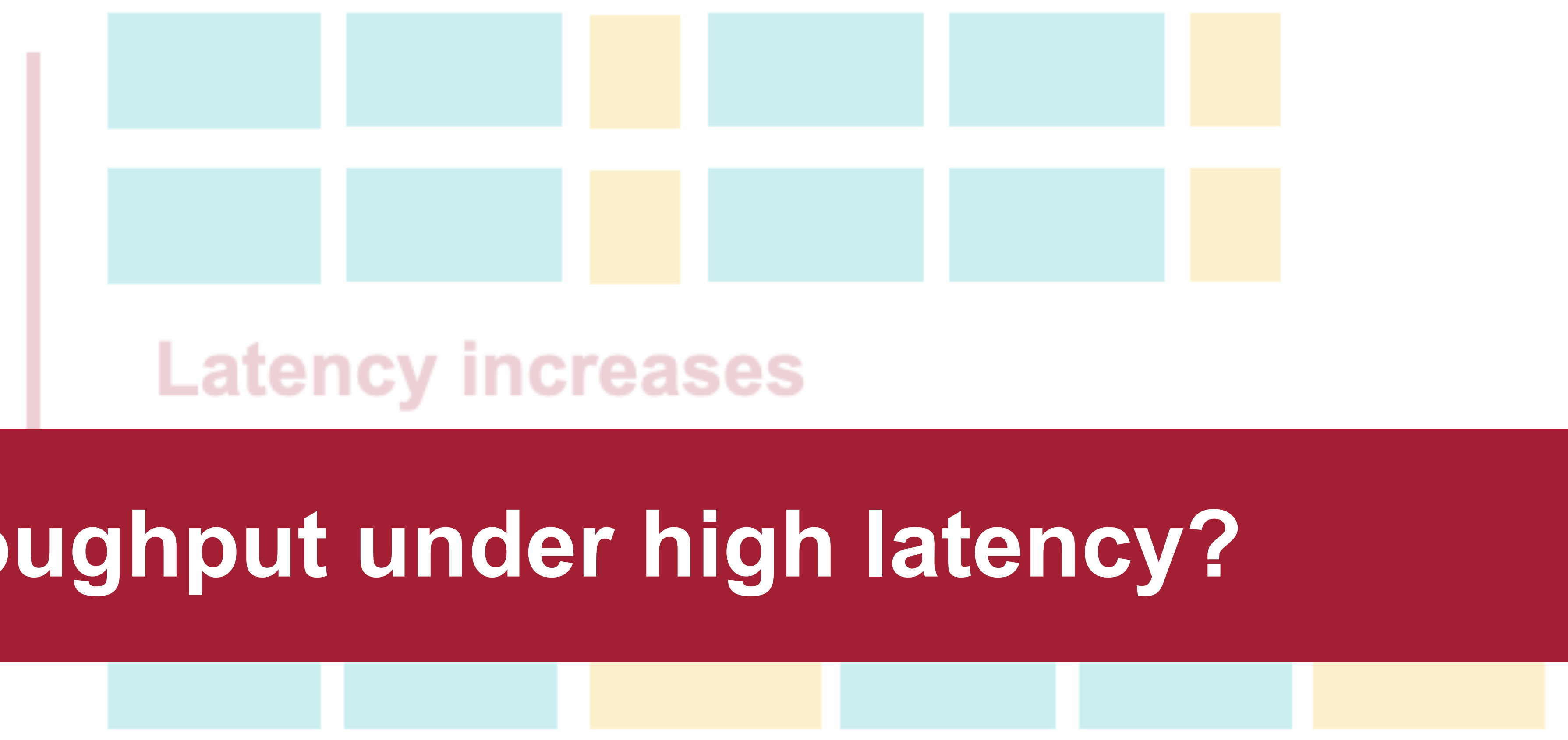
Conventional Algorithms Suffer from High Latency

Federated Averaging [McMahan 16]

1. $\nabla w_{(i,j)} = \dfrac{\partial F(x_{(i,j)}, y_{(i,j)}; w)}{\partial w}$
2. If i mod K:

    3. $G_i = \dfrac{1}{J} \sum_{j=1}^{J} \nabla w_{(i,j)}$

3. Else
    1. $G_i = \nabla w_{(i,j)}$
4. $w_{(i,j)} = w_{(i,j)} - \eta G_i$



Increase K can amortize the effect, but still, the training suffers from high latency.

Computation

Communication

How to improve training throughput under high latency?

Pipeline computation and communication!

Delayed Gradient Averaging

Delay Gradient Averaging [Ours]

1. Sample and calculate $\nabla w_{(i,j)}$
2. If i mod K == 0
    1. $\boxed{\text{Send fresh } \nabla w_{(i,j)} \text{ to other nodes}}$
3. If i mod K == D
    1. $\boxed{\text{Recv stale } \nabla w_{(i-D,j)} \text{ from other nodes}}$
    2. $\overline{\nabla w_{(i-D)}} = \frac{1}{J}\sum_{j=1}^{J} \nabla w_{(i-D,j)}$
4. $\boxed{w_{(i,j)} = w_{(i,j)} - \eta(\nabla w_{(i,j)} - \nabla w_{(i-D,j)} + \overline{\nabla w_{(i-D)}})}$

1. Delay the averaging to a later iteration.

2. Correction term to compensate the accuracy.

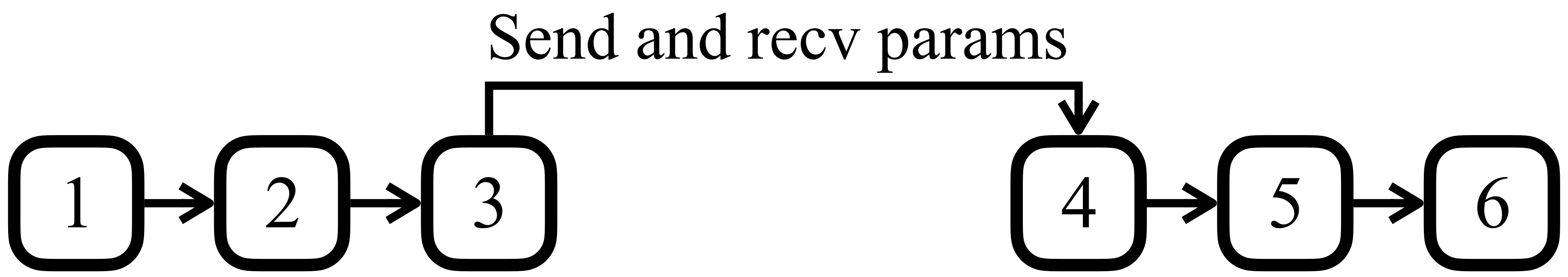
i: iteration, j: work index, x: training data, w: model weights

Delayed Gradient Averaging

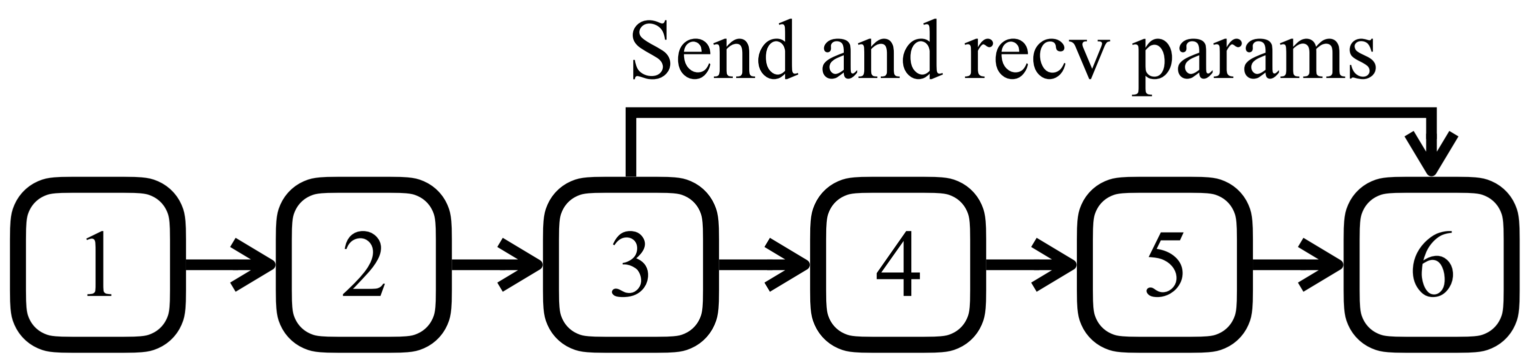
Delay Gradient Averaging [Ours]

1. Sample and calculate $\nabla w_{(i,j)}$
2. If i mod K == 0
   1. Send fresh $\nabla w_{(i,j)}$ to other nodes

   Delay D steps
3. If i mod K == D
   1. Recv stale $\nabla w_{(i-D,j)}$ from other nodes
   2. $\overline{\nabla w_{(i-D)}} = \frac{1}{J}\sum_{j=1}^{J} \nabla w_{(i-D,j)}$
4. $w_{(i,j)} = w_{(i,j)} - \eta(\nabla w_{(i,j)} - \nabla w_{(i-D,j)} + \overline{\nabla w_{(i-D)}})$



W/o delay: all the local machines are blocked to wait for the synchronization to finish

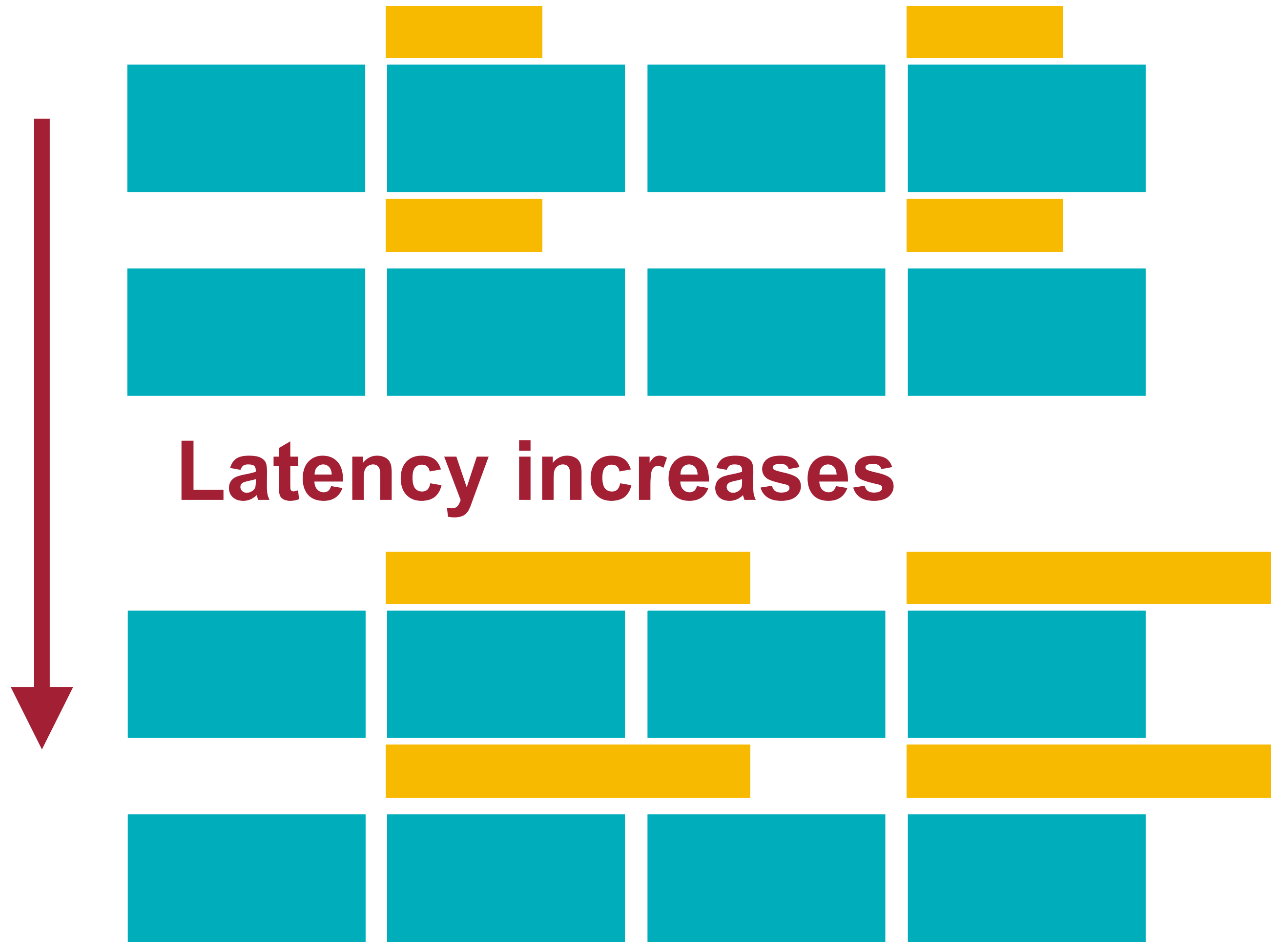


With delay: Worker keep performing local updates while the parameters are in transmission.

i: iteration, j: work index, x: training data, w: model weights

Delayed Gradient Averaging

Communication is covered by computation.



Delay Gradient Averaging [Ours]

1. Sample and calculate $\nabla w_{(i,j)}$
2. If i mod K == 0
   1. Send fresh $\nabla w_{(i,j)}$ to other nodes
3. If i mod K == D

   Delay D steps
   1. Recv stale $\nabla w_{(i-D,j)}$ from other nodes
   2. $\overline{\nabla w_{(i-D)}} = \frac{1}{J}\sum_{j=1}^{J} \nabla w_{(i-D,j)}$
4. $w_{(i,j)} = w_{(i,j)} - \eta(\nabla w_{(i,j)} - \nabla w_{(i-D,j)} + \overline{\nabla w_{(i-D)}})$

As long as the transmission finishes within $D \times T_{\text{computation}}$ the training will not be blocked.

Computation Communication

i: iteration, j: work index, x: training data, w: model weights

The Design of Correction Term

Current local gradients

Stale global gradients

$$w_{(i,j)} = w_{(i,j)} - \eta(\nabla w_{(i,j)} - \nabla w_{(i-D,j)} + \overline{\nabla w_{(i-D)}})$$

Stale local gradients

Consider the **3rd iteration** with $D = 2$

$$w_{(3,j)} = w_{(1,j)} - \eta(\boxed{\nabla w_{(1,j)} + \nabla w_{(2,j)} + \nabla w_{(3,j)}})$$

Local gradients

The Design of Correction Term

$$w_{(i,j)} = w_{(i,j)} - \eta(\overbrace{\nabla w_{(i,j)}}^{\text{Current local gradients}} \underbrace{\boxed{- \nabla w_{(i-D,j)}}}_{\text{Stale local gradients}} + \overbrace{\overline{\nabla w_{(i-D)}}}^{\text{Stale global gradients}})$$

Consider the <u>**3rd iteration**</u> with $D = 2$

$$w_{(3,j)} = w_{(1,j)} - \eta(\cancel{\nabla w_{(1,j)}} + \nabla w_{(2,j)} + \nabla w_{(3,j)})$$

The Design of Correction Term

Current local gradients

Stale global gradients

$$w_{(i,j)} = w_{(i,j)} - \eta(\nabla w_{(i,j)} - \nabla w_{(i-D,j)} \boxed{+ \overline{\nabla w_{(i-D)}}})$$

Stale local gradients

Consider the **<u>3rd iteration</u>** with $D = 2$

$$w_{(3,j)} = w_{(1,j)} - \eta(\underbrace{\cancel{\nabla w_{(1,j)}}}_{\overline{\nabla \mathbf{w}_{(1)}}} + \nabla w_{(2,j)} + \nabla w_{(3,j)})$$

The Design of Correction Term

$$w_{(i,j)} = w_{(i,j)} - \eta(\underbrace{\nabla w_{(i,j)}}_{\text{Current local gradients}} - \underbrace{\nabla w_{(i-D,j)}}_{\text{Stale local gradients}} + \underbrace{\overline{\nabla w_{(i-D)}}}_{\text{Stale global gradients}})$$

Consider the **3rd iteration** with $D = 2$

$$w_{(3,j)} = w_{(1,j)} - \eta(\overline{\nabla w_{(1)}} + \nabla w_{(2,j)} + \nabla w_{(3,j)})$$

Replacing oldest local gradients with global averaged ones!

The Design of Correction Term

$$w_{(i,j)} = w_{(i,j)} - \eta(\underbrace{\nabla w_{(i,j)}}_{\text{Current local gradients}} \boxed{- \underbrace{\nabla w_{(i-D,j)}}_{\text{Stale local gradients}}} + \underbrace{\overline{\nabla w_{(i-D)}}}_{\text{Stale global gradients}})$$

Consider the **4th iteration** with $D = 2$

$$w_{(3,j)} = w_{(1,j)} - \eta(\overline{\nabla w_{(1)}} + \nabla w_{(2,j)} + \nabla w_{(3,j)})$$

$$w_{(4,j)} = w_{(1,j)} - \eta(\overline{\nabla w_{(1)}} + \cancel{\nabla w_{(2,j)}} + \nabla w_{(3,j)} + \nabla w_{(4,j)})$$

The Design of Correction Term

$$w_{(i,j)} = w_{(i,j)} - \eta(\underbrace{\nabla w_{(i,j)}}_{\text{Current local gradients}} - \underbrace{\nabla w_{(i-D,j)}}_{\text{Stale local gradients}} + \boxed{\underbrace{\overline{\nabla w_{(i-D)}}}_{\text{Stale global gradients}}})$$

Consider the **4th iteration** with $D = 2$

$$w_{(3,j)} = w_{(1,j)} - \eta(\overline{\nabla w_{(1)}} + \nabla w_{(2,j)} + \nabla w_{(3,j)})$$

$$w_{(4,j)} = w_{(1,j)} - \eta(\overline{\nabla w_{(1)}} + \underbrace{\cancel{\nabla w_{(2,j)}}}_{\overline{\nabla \mathbf{w}_{(2)}}} + \nabla w_{(3,j)} + \nabla w_{(4,j)})$$

The Design of Correction Term

$$w_{(i,j)} = w_{(i,j)} - \eta(\underbrace{\nabla w_{(i,j)}}_{\text{Current local gradients}} - \underbrace{\nabla w_{(i-D,j)}}_{\text{Stale local gradients}} + \underbrace{\overline{\nabla w_{(i-D)}}}_{\text{Stale global gradients}})$$

Consider the **4th iteration** with $D = 2$

$$w_{(3,j)} = w_{(1,j)} - \eta(\overline{\nabla w_{(1)}} + \nabla w_{(2,j)} + \nabla w_{(3,j)})$$

$$w_{(4,j)} = w_{(1,j)} - \eta(\overline{\nabla w_{(1)}} + \overline{\nabla w_{(2)}} + \nabla w_{(3,j)} + \nabla w_{(4,j)})$$

The Design of Correction Term

Current local gradients / Stale global gradients

$$w_{(i,j)} = w_{(i,j)} - \eta(\nabla w_{(i,j)} - \nabla w_{(i-D,j)} + \overline{\nabla w_{(i-D)}})$$

Stale local gradients

$$w_{(3,j)} = w_{(1,j)} - \eta(\overline{\nabla w_{(1)}} + \nabla w_{(2,j)} + \nabla w_{(3,j)})$$

$$w_{(4,j)} = w_{(1,j)} - \eta(\overline{\nabla w_{(1)}} + \overline{\nabla w_{(2)}} + \nabla w_{(3,j)} + \nabla w_{(4,j)})$$

$$w_{(i,j)} = w_{(1,j)} - \eta(\overline{\nabla w_{(1)}} + \ldots + \overline{\nabla w_{(i-D,j)}} + \boxed{\nabla w_{(i-D+1,j)} + \ldots + \nabla w_{(i,j)}})$$

Only **most recent *D* updates** are local gradients.

The Design of Correction Term

Our DGA:

$$w_{(i,j)} = w_{(1,j)} - \eta(\overline{\nabla w_{(1)}} + \ldots + \overline{\nabla w_{(i-D,j)}} + \boxed{\nabla w_{(i-D+1,j)} + \ldots + \nabla w_{(i,j)})}$$

The divergence is bounded.

Vanilla Distributed SGD:

$$w_{(i,j)} = w_{(1,j)} - \eta(\overline{\nabla w_{(1)}} + \ldots + \overline{\nabla w_{(i-D,j)}} + \boxed{\overline{\nabla w_{(i-D+1)}} + \ldots + \overline{\nabla w_{(i)}})}$$

Usual training consists of >10k iterations, such divergence is small.

DGA Guarantees the Convergence

- Assumption 1: the loss function $F(w; x, y)$ is **Lipchitz smooth**

$$\nabla f_j(x) - \nabla f_j(y)\,||\leq L\,||x-y||\,. \quad \forall x, y \in \mathbb{R}^d$$

- Assumption 2: **Bounded gradients and variances**

$$\mathbb{E}_{\zeta_j}||\nabla F_j(w;\zeta_i)||^2 \leq G^2, \forall w, \forall j,\ \mathbb{E}_{\zeta_j}||\nabla F_j(w;\zeta_j) - \nabla f_j(w)||^2 \leq \sigma^2, \forall w, \forall j\,.$$

The convergence rate of DGA is $O(\frac{\Delta + \sigma^2}{\sqrt{JN}} + \frac{Jd^2}{N})$ (details in paper)

When $D < O(N^{\frac{1}{4}} J^{-\frac{3}{4}})$, **DGA converges as fast as original SGD** which is $O(\frac{\Delta + \sigma^2}{\sqrt{JN}})$.

(N:iterations, J:number of machines)

DGA Improves the Accuracy

	Paritions	FedAvg(k=5)		FedAvg(k=10)		FedAvg(k=20)		DGA(K=5,D=20)	
CIFAR	I.I.D	88.7	1.0x	88.5	1.51x	88.1	2.05x	88.6	3.16x
	Non-I.I.D	48.2		47.2		43.9		48.0	

DGA shows **negligible accuracy drop**.

DGA Improves the Accuracy

	Paritions	FedAvg(k=5)		FedAvg(k=10)		FedAvg(k=20)		DGA(K=5,D=20)	
CIFAR	I.I.D	88.7	1.0x	88.5	1.51x	88.1	2.05x	88.6	3.16x
	Non-I.I.D	48.2		47.2		43.9		48.0	

DGA shows **much better accuracy** on non I.I.D partitions.

DGA Improves the Accuracy

	Paritions	FedAvg(k=5)		FedAvg(k=10)		FedAvg(k=20)		DGA(K=5,D=20)	
CIFAR	I.I.D	88.7	1.0x	88.5	1.51x	88.1	2.05x	88.6	3.16x
	Non-I.I.D	48.2		47.2		43.9		48.0	

While producing higher accuracy, DGA also demonstrates **faster training speed** as it fully covers communication with computation.

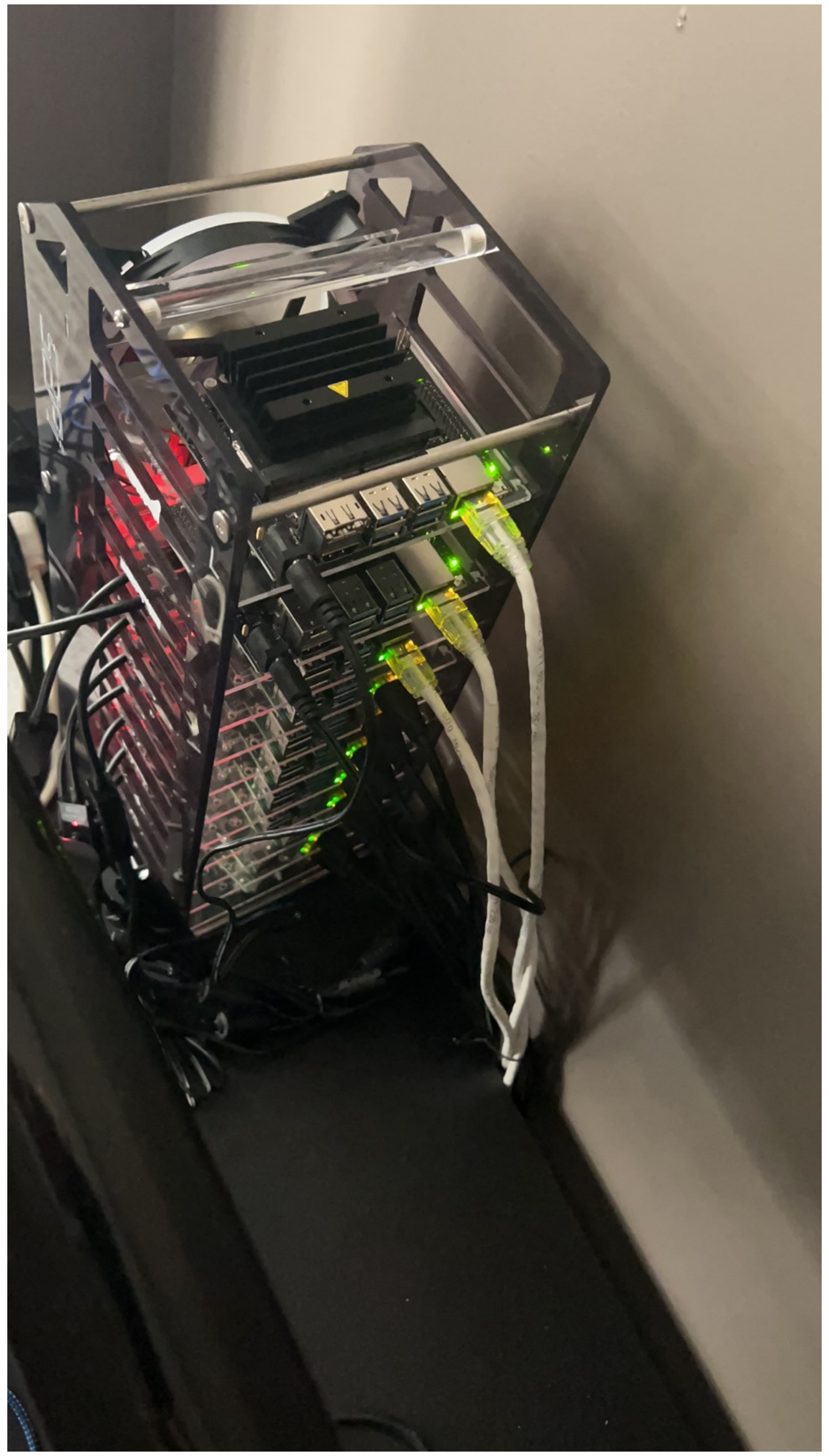
DGA Improves the Accuracy

	Paritions	FedAvg(k=5)		FedAvg(k=10)		FedAvg(k=20)		DGA(K=5,D=20)	
CIFAR	I.I.D	88.7	1.0x	88.5	1.51x	88.1	2.05x	88.6	3.16x
	Non-I.I.D	48.2		47.2		43.9		48.0	
ImageNet	I.I.D	76.6	1.0x	76.5	1.43x	76.2	1.81x	76.4	2.55x
	Non-I.I.D	55.4		52.5		48.6		54.9	

DGA Improves the Accuracy

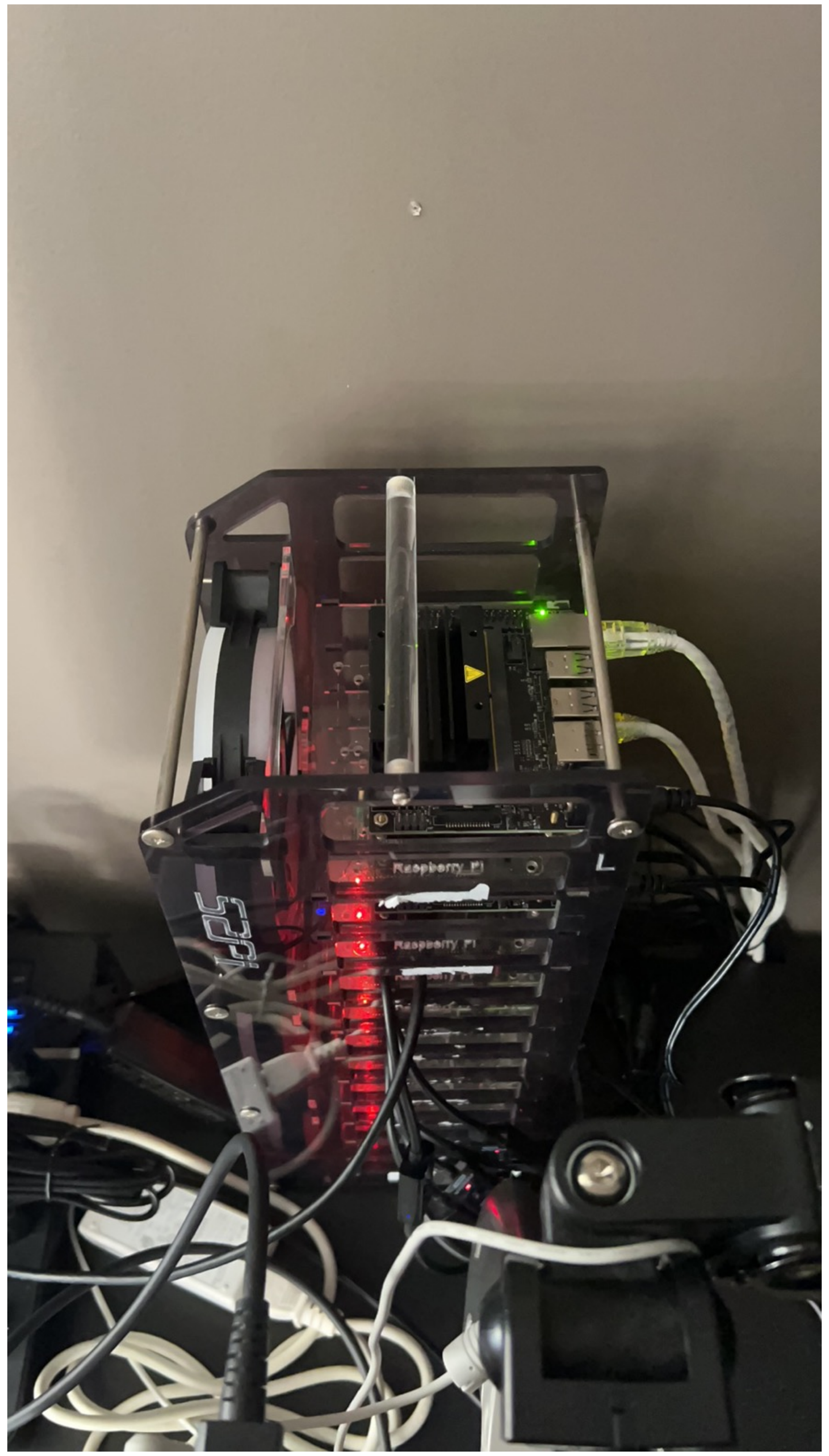
	Paritions	FedAvg(k=5)		FedAvg(k=10)		FedAvg(k=20)		DGA(K=5,D=20)	
CIFAR	I.I.D	88.7	1.0x	88.5	1.51x	88.1	2.05x	88.6	3.16x
	Non-I.I.D	48.2		47.2		43.9		48.0	
ImageNet	I.I.D	76.6	1.0x	76.5	1.43x	76.2	1.81x	76.4	2.55x
	Non-I.I.D	55.4		52.5		48.6		54.9	
Shakespeare	I.I.D	47.6	1.0x	47.3	1.66x	47.3	2.51x	47.1	4.07x
	Non-I.I.D	36.9		34.3		30.1		36.3	

Real-world Benchmark

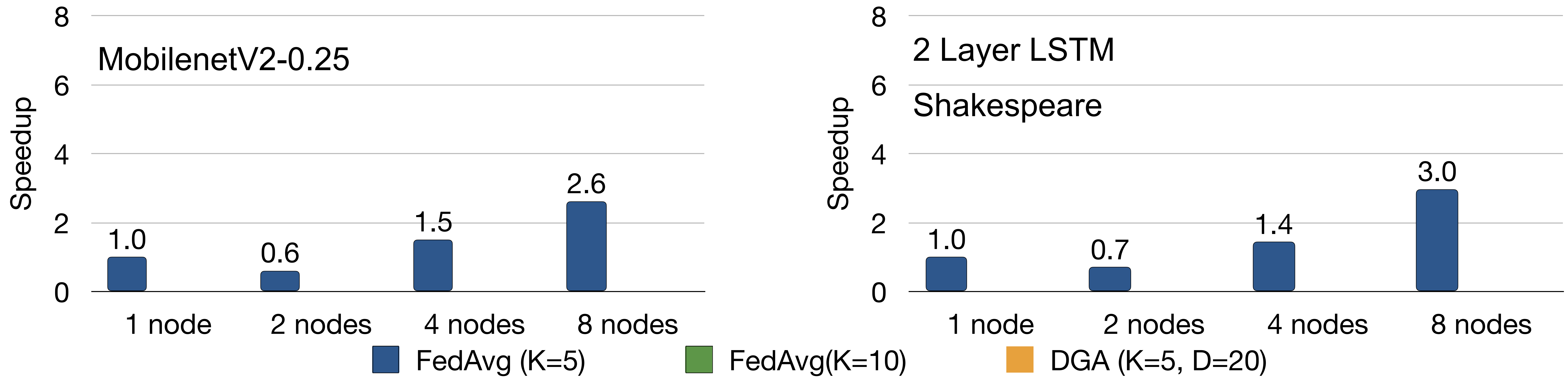


We build a raspberry pi cluster to simulate real-world federated learning scenarios.

- Device: 8 x Raspberry Pi 4B+ Models
- Device OS: Debian 10
- Router: Netgear R6300v2
- Router OS: OpenWRT

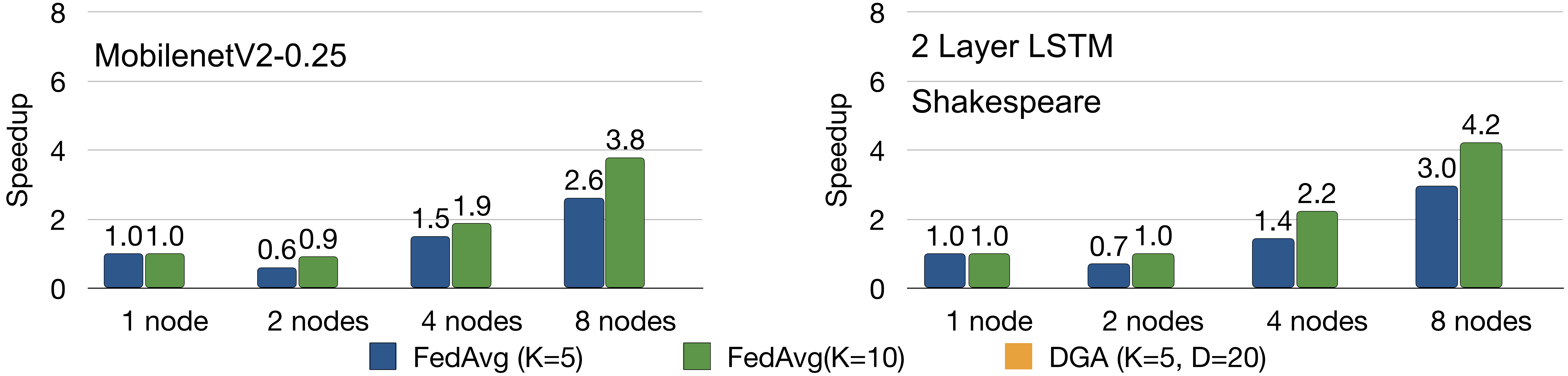


Benchmark on Raspberry Pi Farms



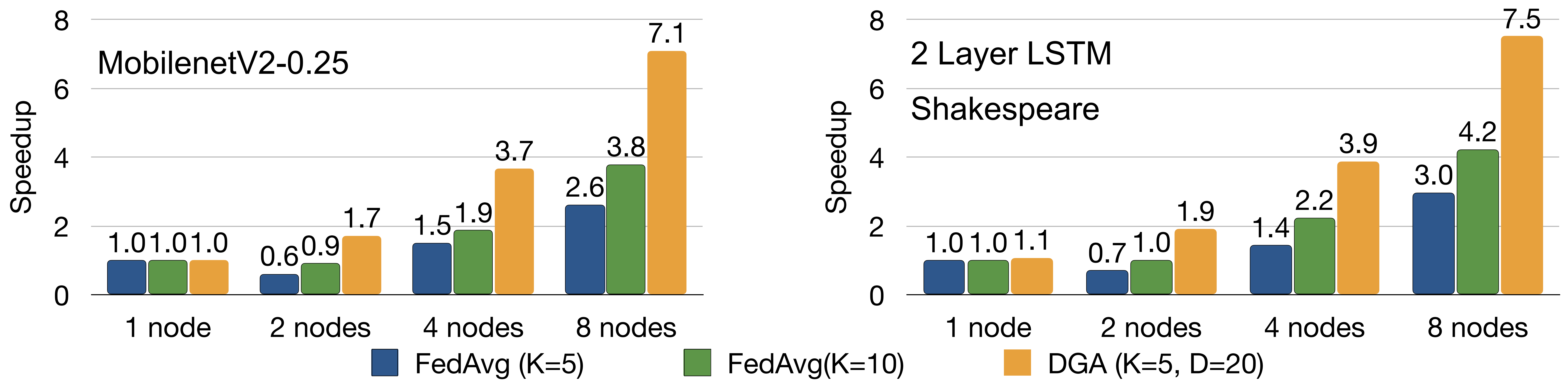
When scaling the training to two devices, the normalized throughput is only 0.6, which is even slower than single device.

Benchmark on Raspberry Pi Farms



Even we set a larger value of K, the scalability is still less than 0.5 and not comparable with training throughput based on in-cluster networks.

Benchmark on Raspberry Pi Farms



Our proposed DGA demonstrates ideal scalability under high-latency network. The speedup on eight-device is about 7.1, which close to what conventional algorithms achieved inside a data center.

Thanks for listening!

We design **D**elayed **G**radient **A**veraging (DGA) that

- Delays averaging to a later iteration to tolerate high network latency
- New update formula to compensate the accuracy

We evaluate the algorithm's

- Convergence and accuracy both theoretically and empirically.
- Training throughput under a real-world pi-cluster.