

Lecture 2 Introduction to Federated Learning

马汝辉 副教授
计算机科学与工程系
上海交通大学

饮水思源 爱国荣校



1

**Introduction to
Distributed Learning**

2

**Privacy and Security
Issues in Deep Learning**

3

**Strengths of Federated
learning**

4

FedAvg Algorithm



01

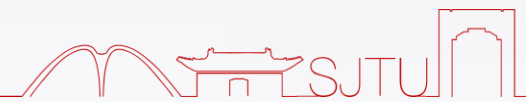
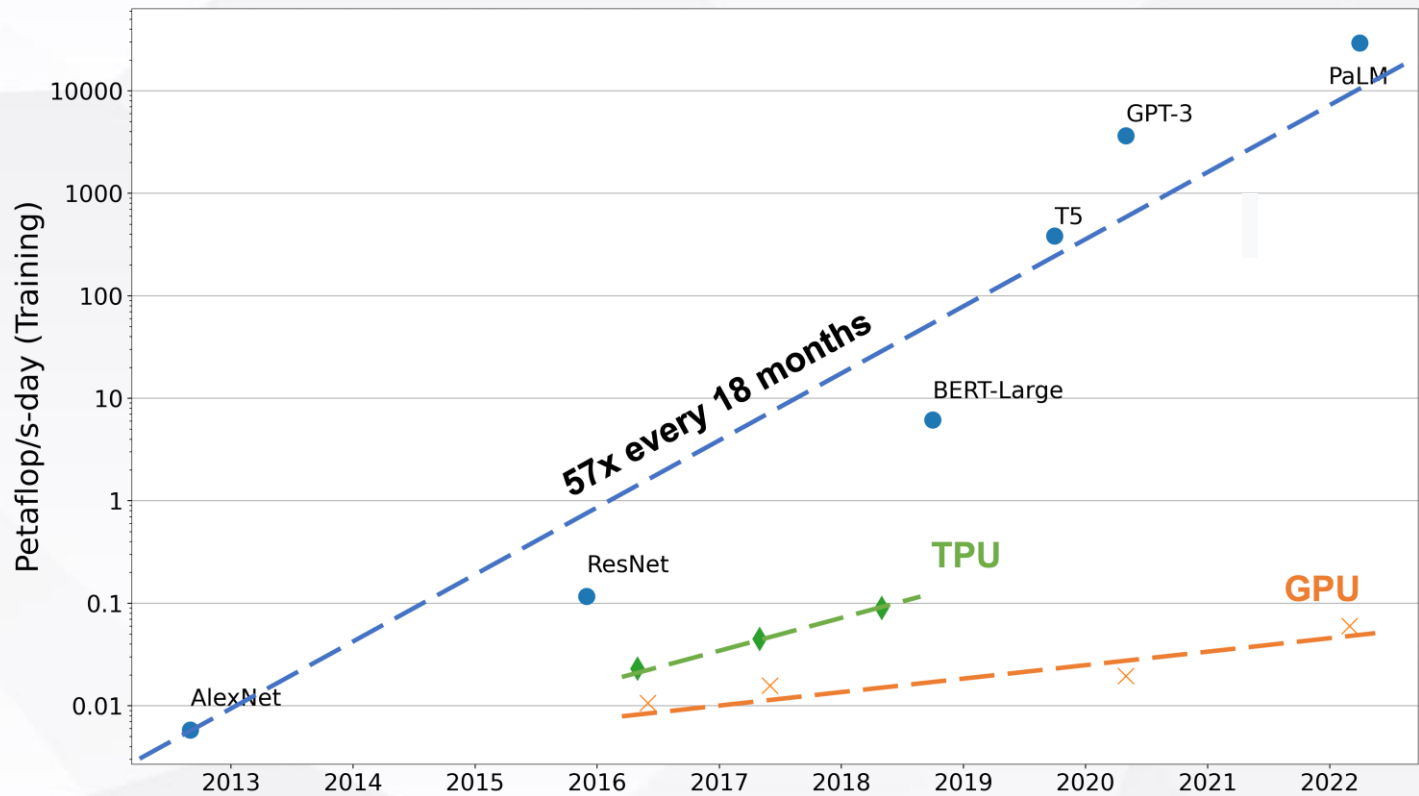
Introduction to Distributed Learning



Motivation



Huge difference in computing power and model size





Model Size

- Parameter 1e9
- Feature Map 2e9
- Gradient 3e9
- Float32 4B
- Total 24GB

Hardware Capacity

- NVIDIA 3070 8G
- NVIDIA 3080 10G
- NVIDIA 3090 24G

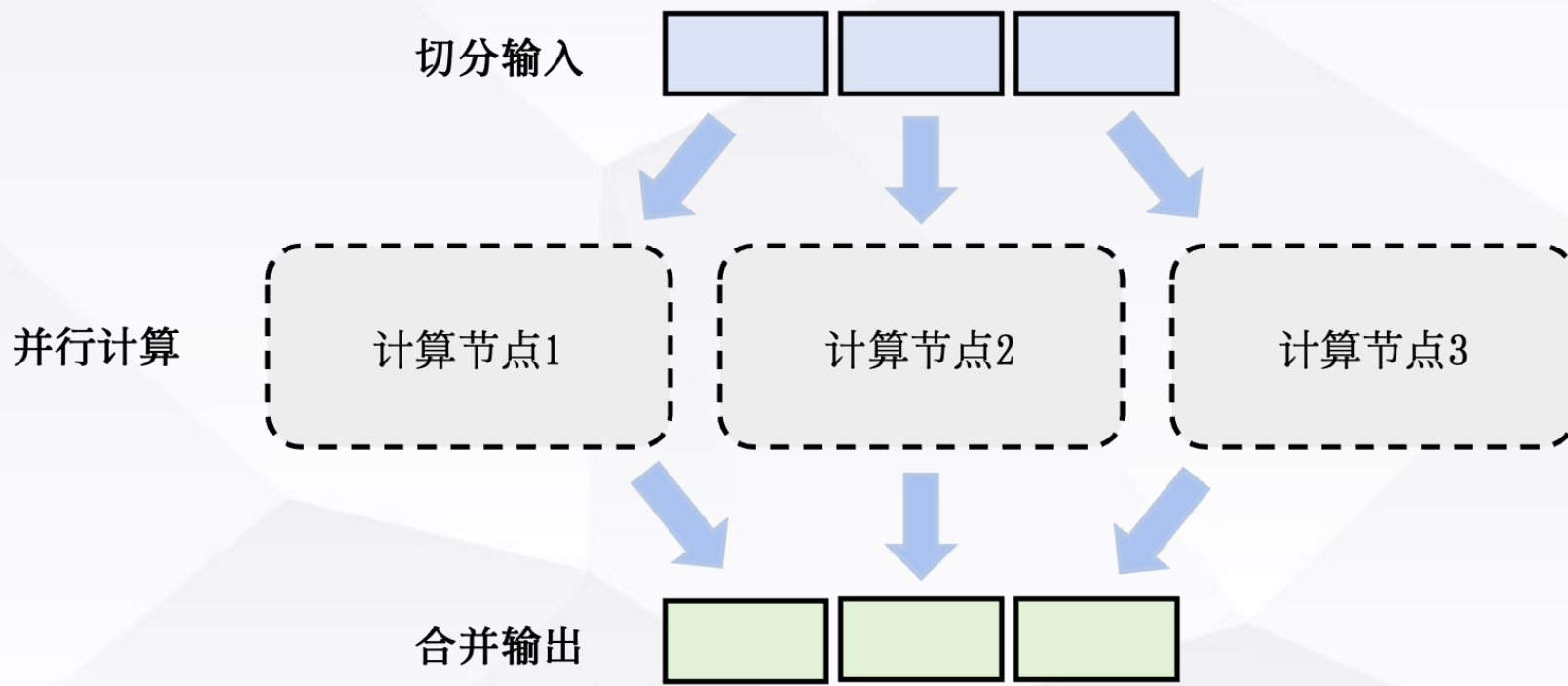
What if there is a bigger model?



Single Node vs Distributed



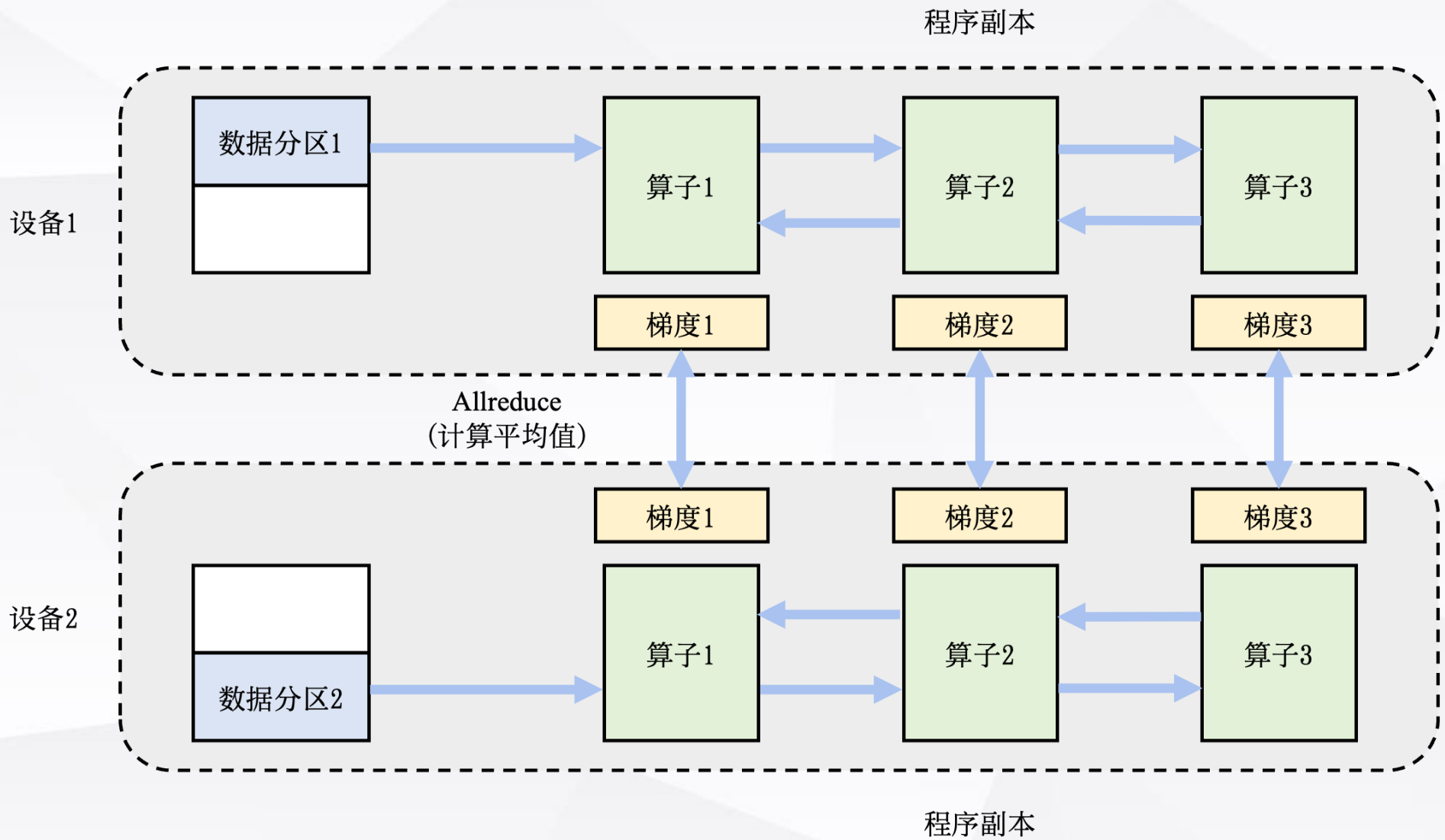
(a) 单节点执行



(b) 分布式多节点执行

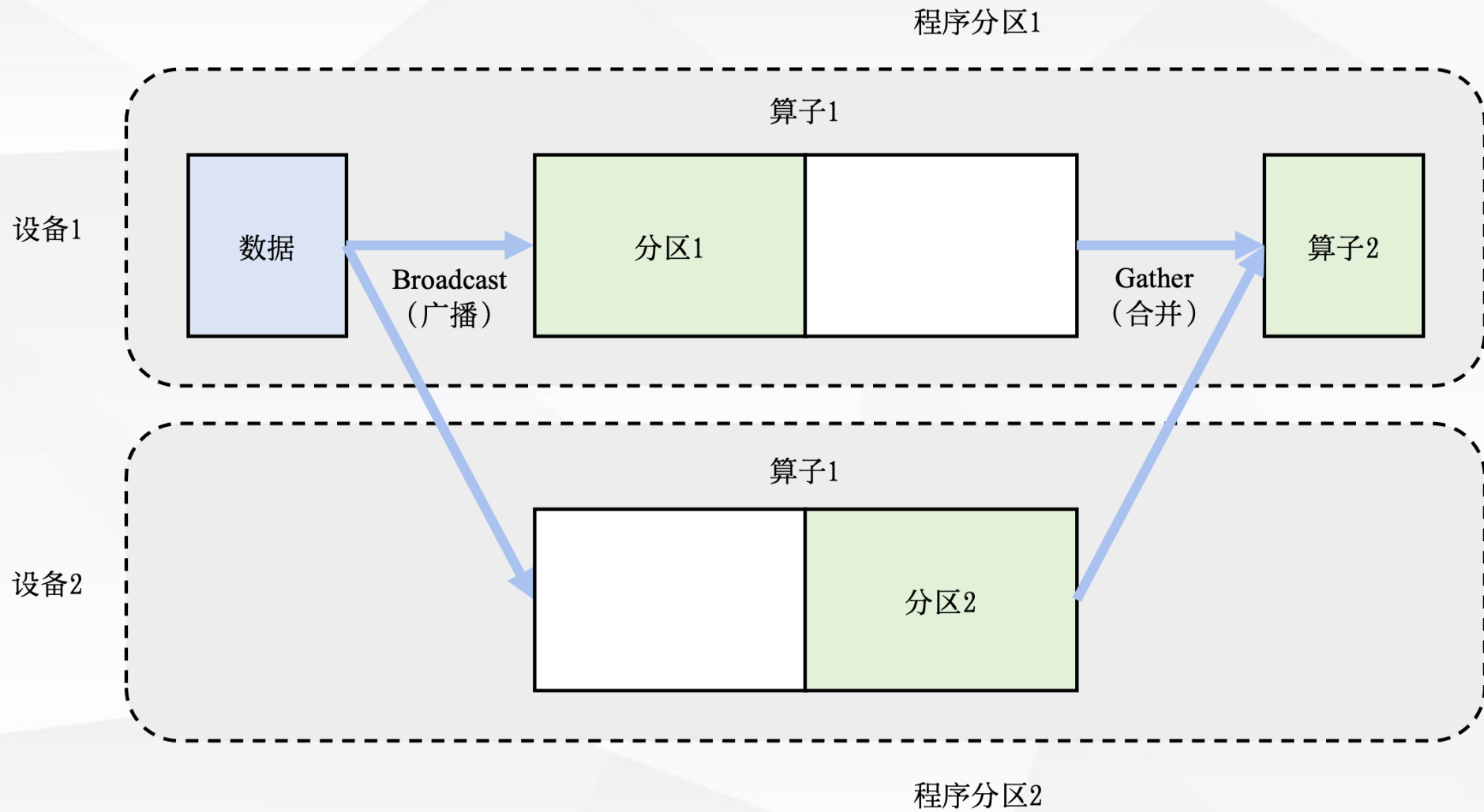


Distributed Methods#1



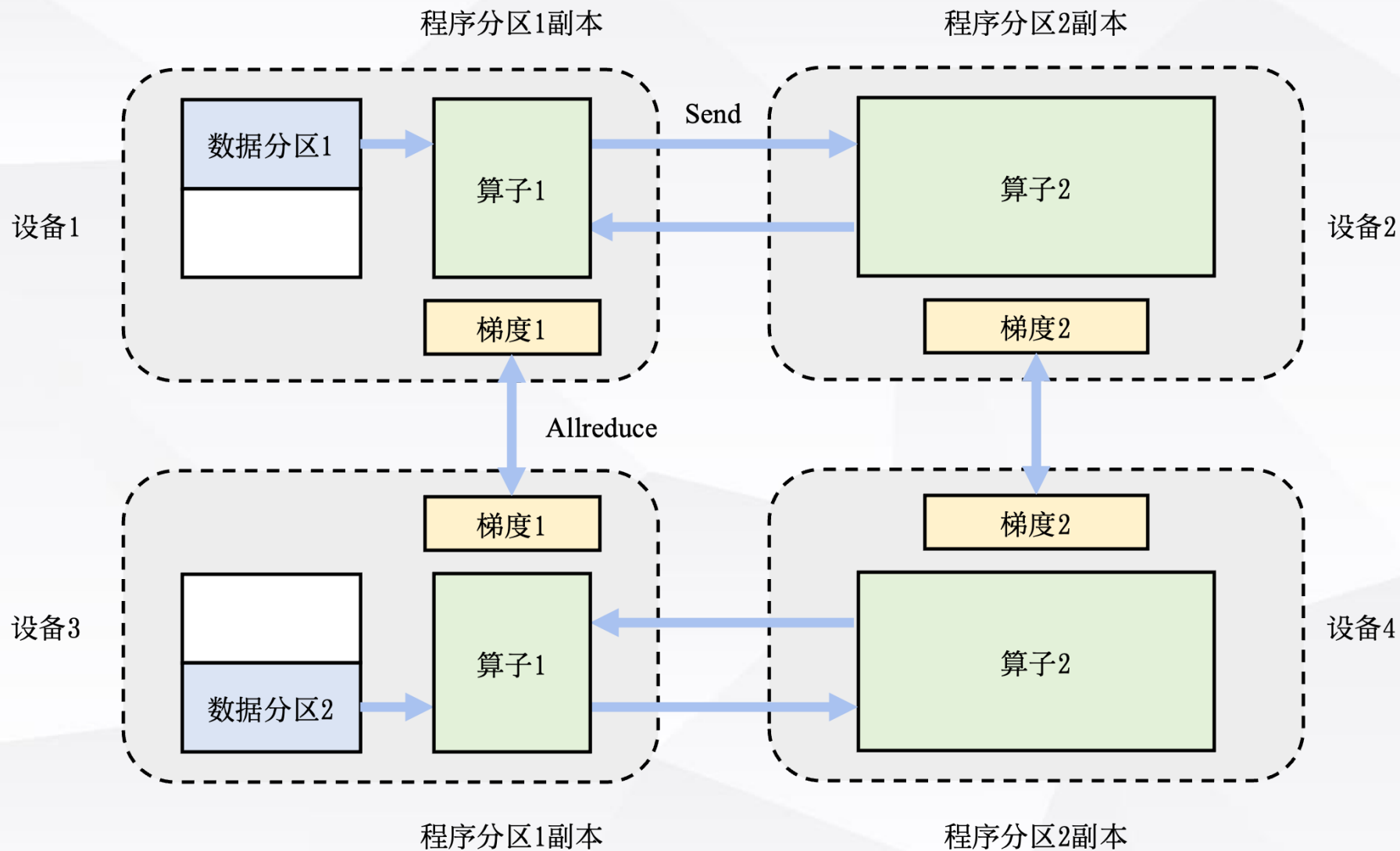


Distributed Methods#2



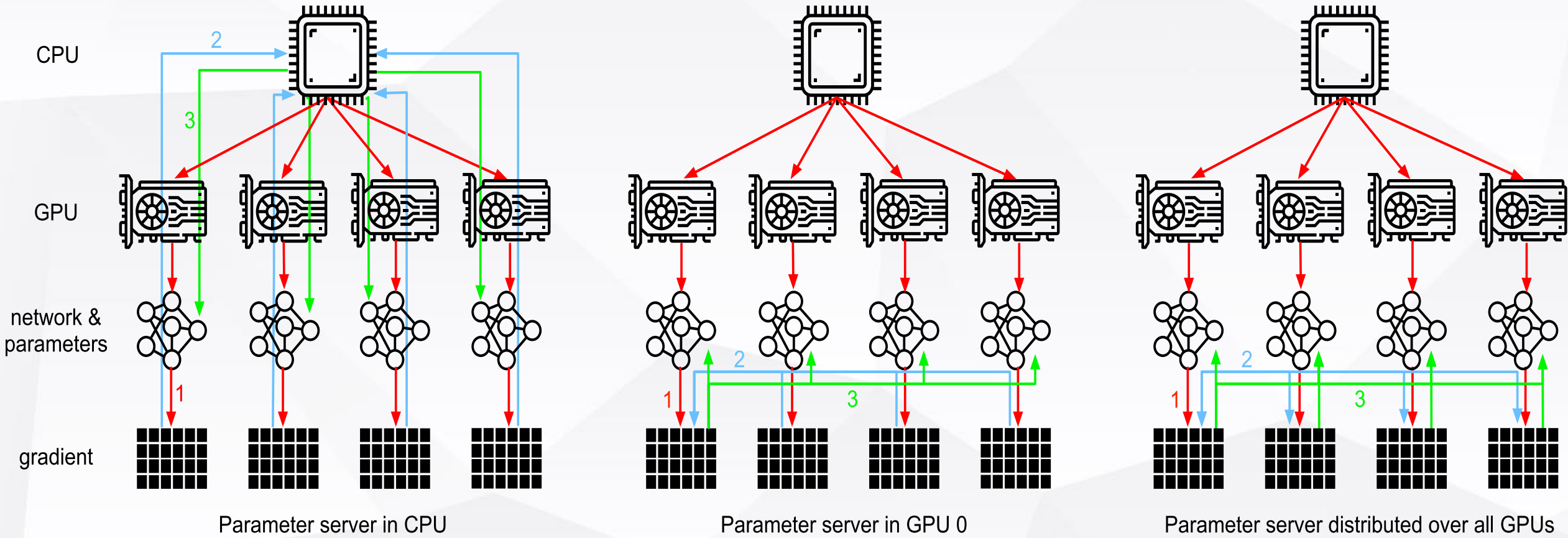


Distributed Methods#3





Problem Of Synchronization





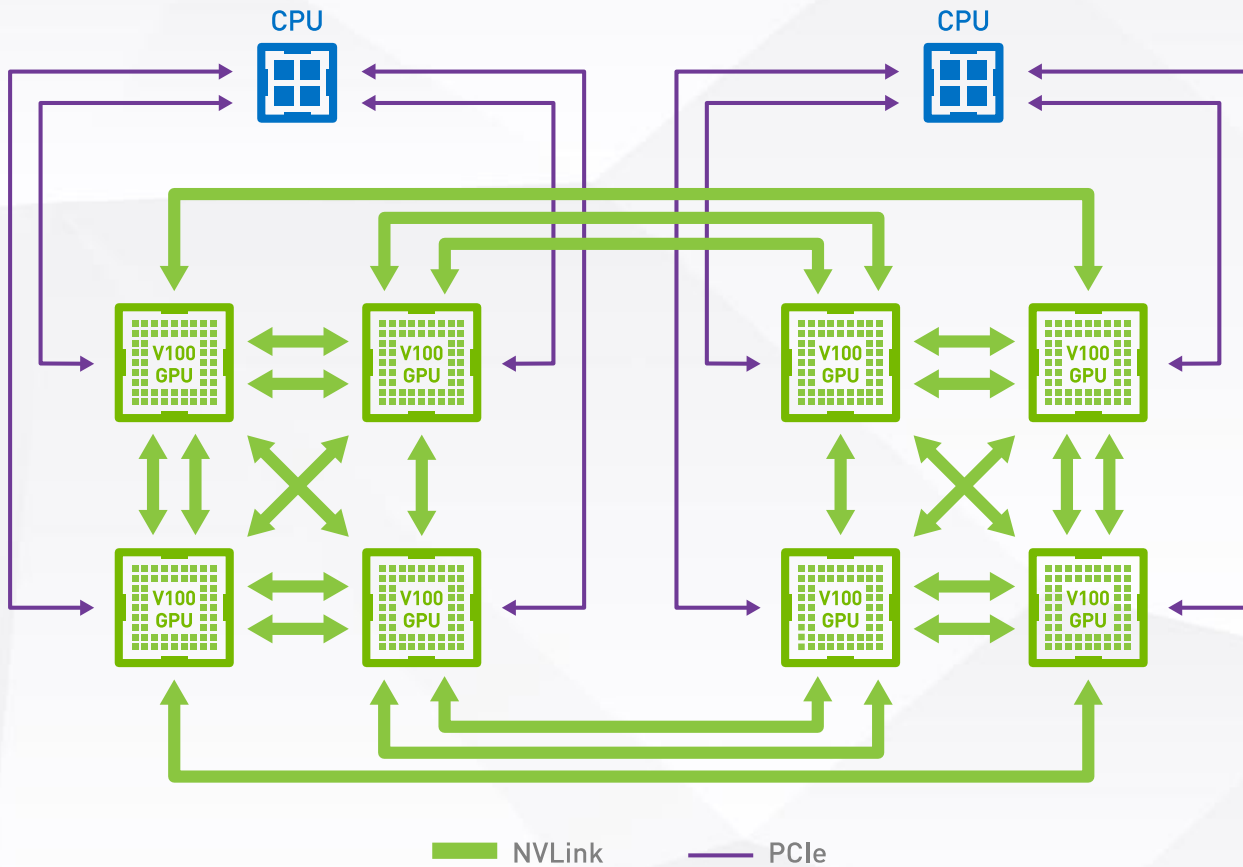
Problem Of Synchronization



- Assume that the gradients are of 160 MB. In this case it takes 30 ms to send the gradients from all 3 remaining GPUs to the fourth one (each transfer takes $10 \text{ ms} = 160 \text{ MB} / 16 \text{ GB/s}$). Adding another 30 ms to transmit the weight vectors back we arrive at a total of 60 ms.
- If we send all data to the CPU we incur a penalty of 40 ms since each of the four GPUs needs to send the data to the CPU, yielding a total of 80 ms.
- we are able to split the gradients into 4 parts of 40 MB each. Now we can aggregate each of the parts on a different GPU simultaneously since the PCIe switch offers a full-bandwidth operation between all links. Instead of 30 ms this takes 7.5 ms, yielding a total of 15 ms for a synchronization operation.



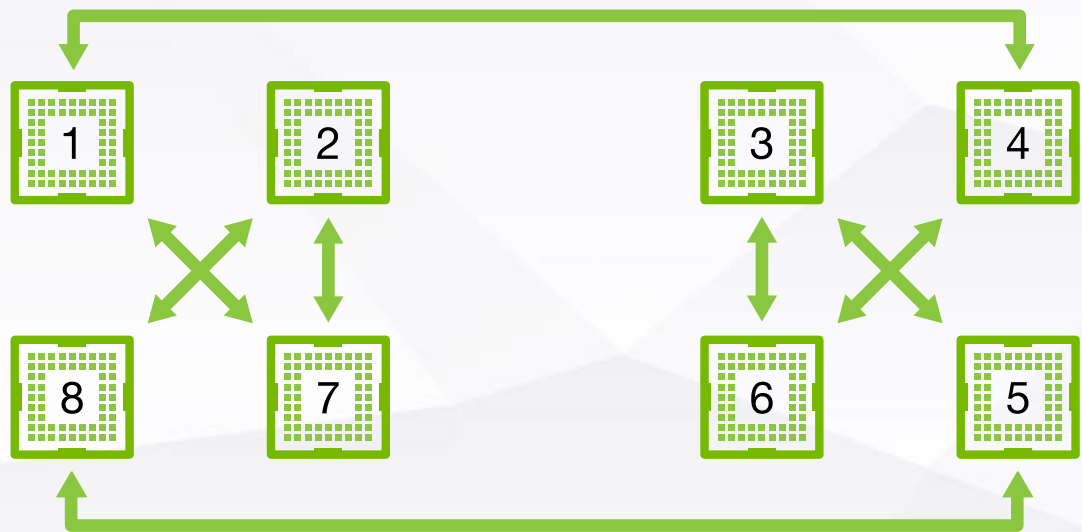
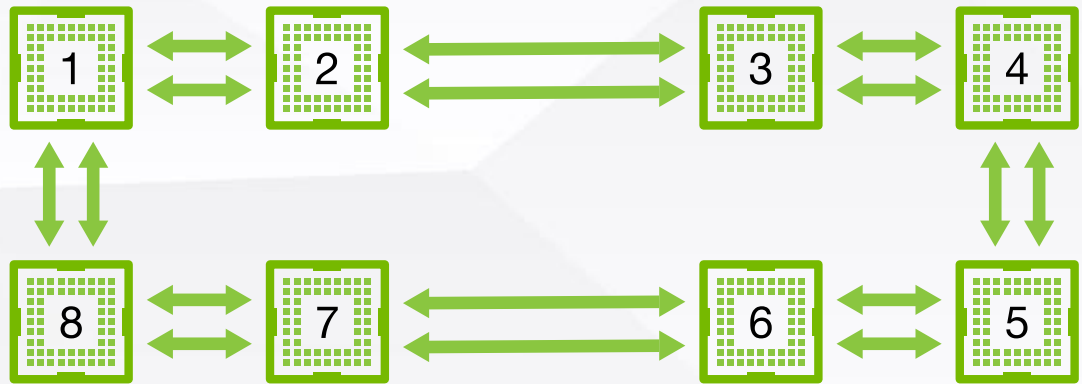
Ring Synchronization



- Each GPU connects to a host CPU via a PCIe link which operates at best at 16 GB/s.
- GPU also has 6 NVLink connections, each of which is capable of transferring 300 Gbit/s bidirectionally.
- This amounts to around 18 GB/s per link. In short, the aggregate NVLink bandwidth is significantly higher than the PCIe bandwidth. The question is how to use it most efficiently.



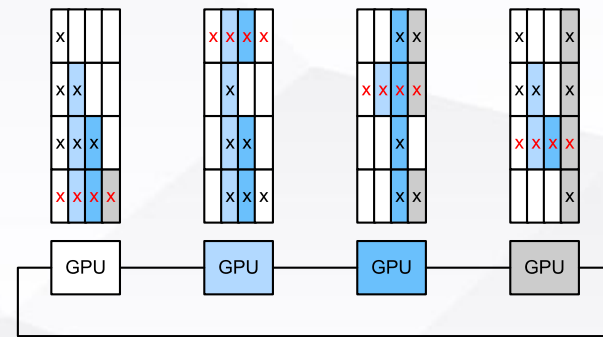
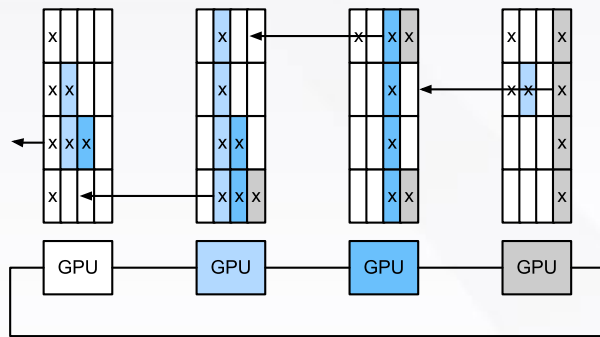
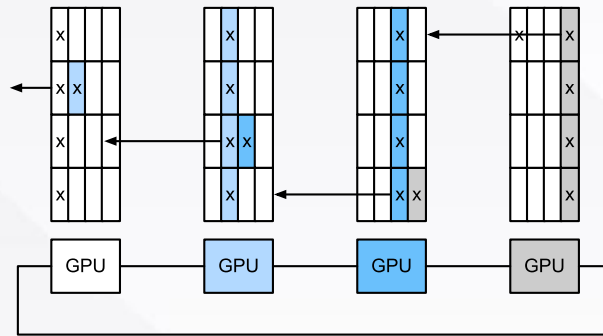
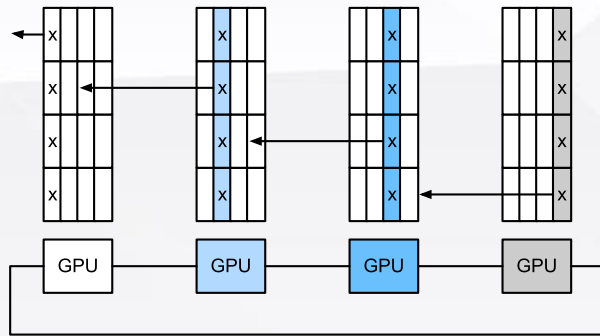
Ring Synchronization



- Network can be decomposed into one ring (1-2-3-4-5-6-7-8-1) with double NVLink bandwidth and into one (1-4-6-3-5-8-2-7-1) with regular bandwidth. Designing an efficient synchronization protocol in this case is nontrivial.



Ring Synchronization

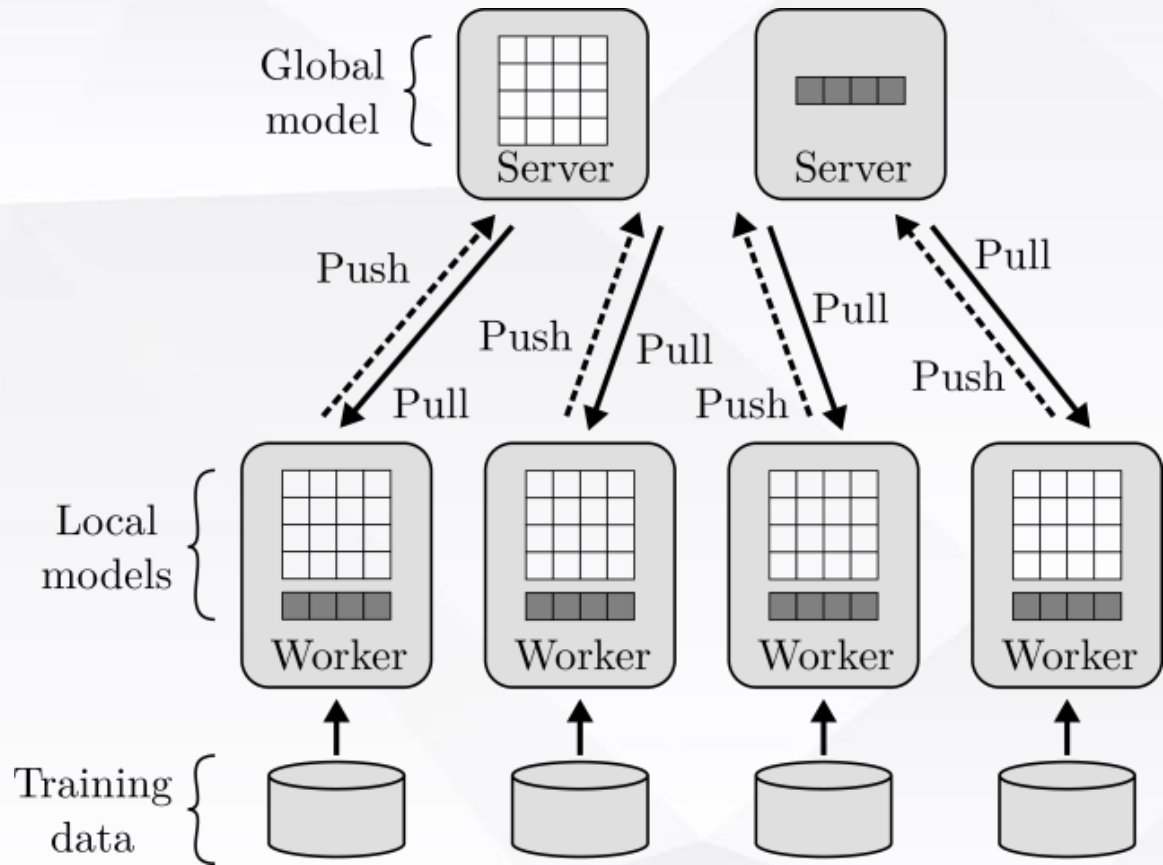


given a ring of n computing nodes we can send gradients from the first to the second node. There it is added to the local gradient and sent on to the third node, and so on. After $n-1$ steps the aggregate gradient can be found in the last-visited node. $O(n)$

broke the gradients into n chunks and started synchronizing chunk i starting at node i . Since each chunk is of size $1/n$ the total time is now $(n-1)/n \approx 1$. $O(1)$



Distributed Methods#4



- A server (or servers) to split data, separate it and aggregate the global model.
- Many workers with separated data to train the separated models.
- Connected by high speed networks



Algorithm 1 Distributed Subgradient Descent

Task Scheduler:

- 1: issue LoadData() to all workers
- 2: **for** iteration $t = 0, \dots, T$ **do**
- 3: issue WORKERITERATE(t) to all workers.
- 4: **end for**

总体的并行训练流程

1. 分发数据到每一个worker节点
2. 每一个worker节点并行执行WORKERITERATE方法，一共迭代T轮，完成模型的并行训练

Worker $r = 1, \dots, m$:

- 1: **function** LOADDATA()
 - 2: load a part of training data $\{y_{i_k}, x_{i_k}\}_{k=1}^{n_r}$
 - 3: pull the working set $w_r^{(0)}$ from servers
- 4: **end function**
- 5: **function** WORKERITERATE(t)
 - 6: gradient $g_r^{(t)} \leftarrow \sum_{k=1}^{n_r} \partial \ell(x_{i_k}, y_{i_k}, w_r^{(t)})$
 - 7: push $g_r^{(t)}$ to servers
 - 8: pull $w_r^{(t+1)}$ from servers
- 9: **end function**

worker的初始化过程

- 1、每个worker载入一部分训练数据
- 2、每个worker将全部初始模型参数 w_0 从servers节点上拉下来

worker节点的迭代计算过程

- 1、仅利用本节点的训练数据计算梯度
- 2、将计算好的梯度 $g_r(t)$ push到servers节点
- 3、将新一轮的模型参数 $w(t+1)$ 拉到本地worker节点

Servers:

- 1: **function** SERVERITERATE(t)
 - 2: aggregate $g^{(t)} \leftarrow \sum_{r=1}^m g_r^{(t)}$
 - 3: $w^{(t+1)} \leftarrow w^{(t)} - \eta (g^{(t)} + \partial \Omega(w^{(t)}))$
- 4: **end function**

server节点的迭代计算过程

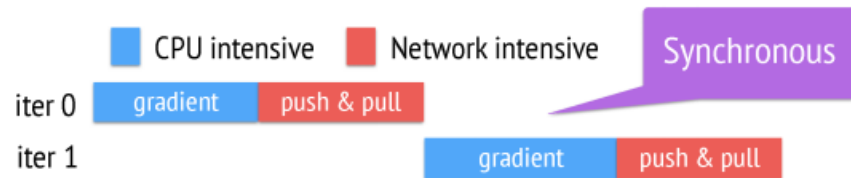
- 1、在收到 m 个worker节点计算的梯度后，汇总梯度形成总梯度
- 2、利用汇总梯度 $g(t)$ ，融合正则化项梯度，计算出新梯度 $w(t+1)$



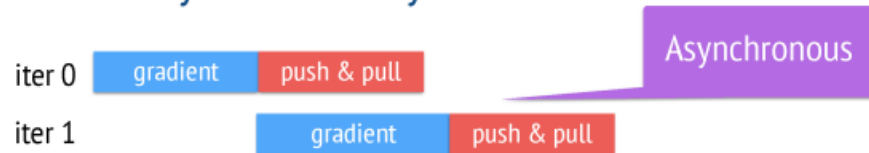
Asynchronous tasks and dependency



“execute-after-finished” dependency



executed asynchronously



- Any push or pull request can be a task, so can be a remote function that is executing. Tasks are generally asynchronous in nature and programs/applications can continue executing after issuing the task.

Bulk Synchronous Execution

Machine



Machine

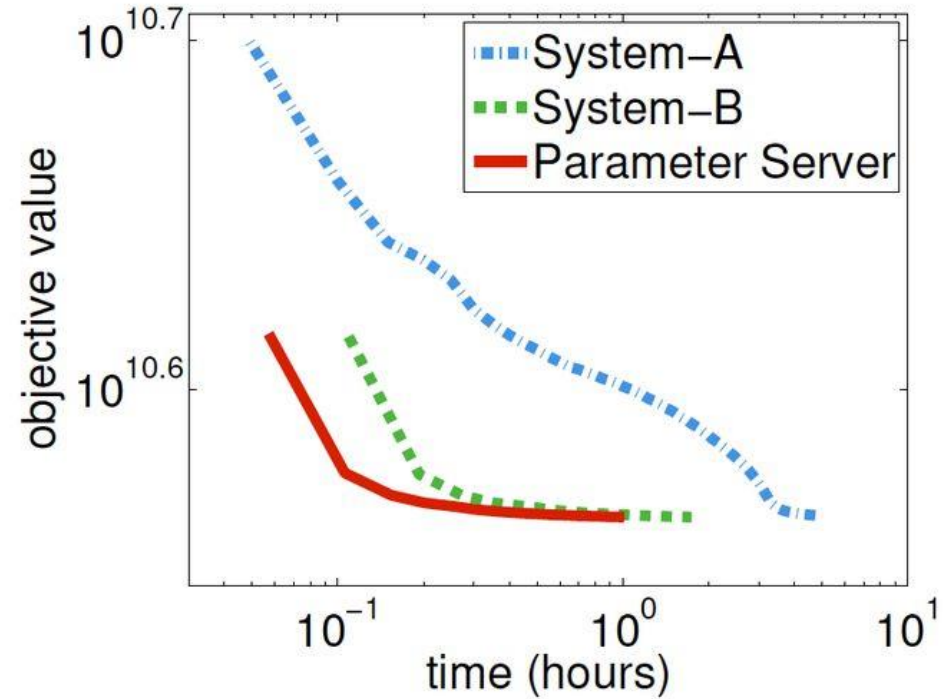
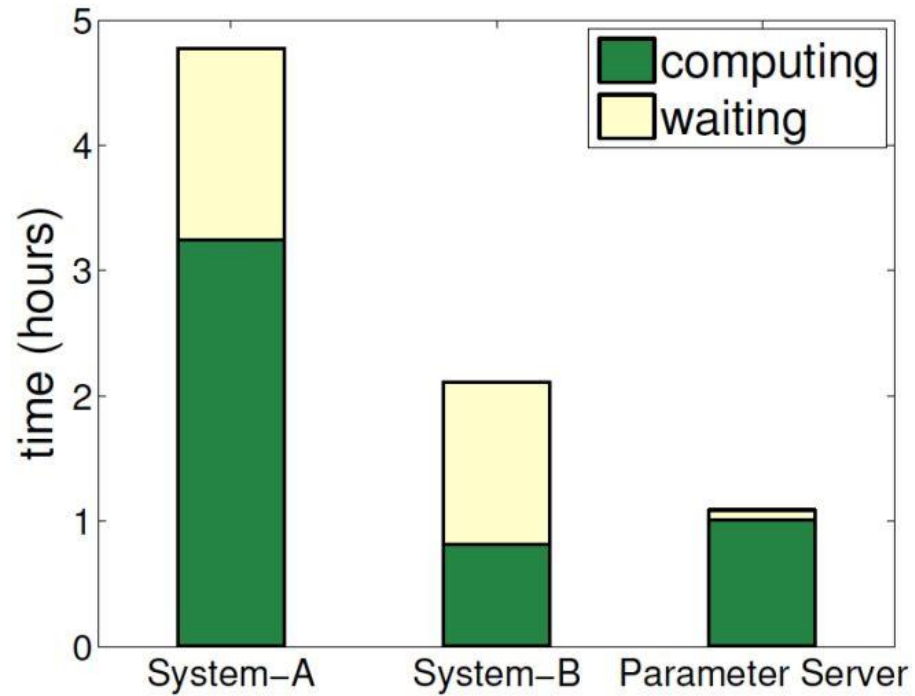


Machine



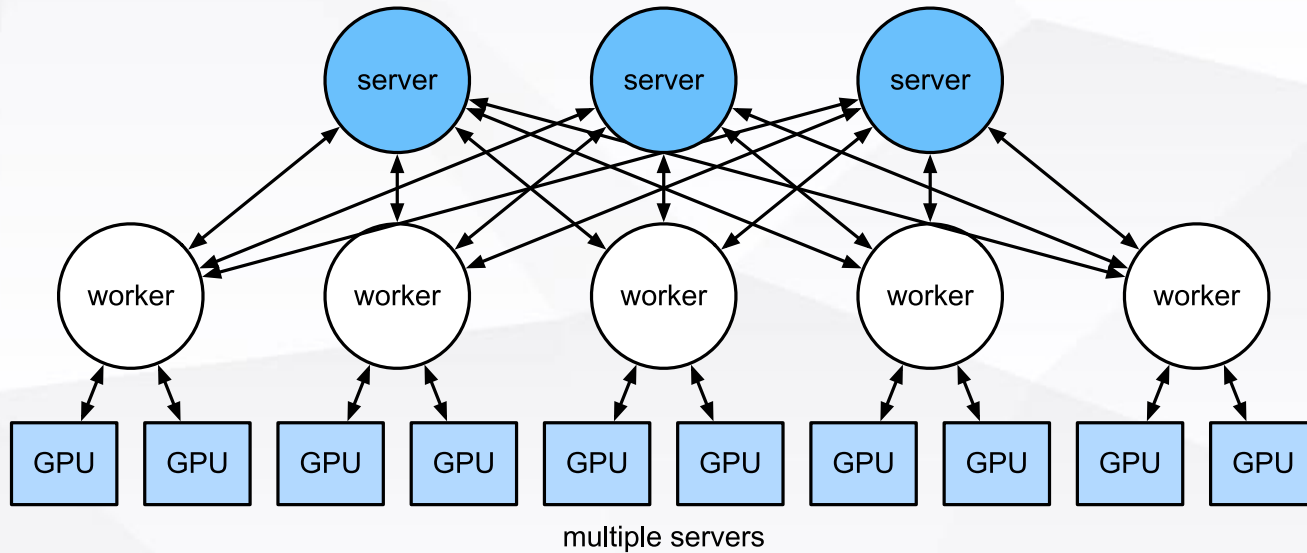
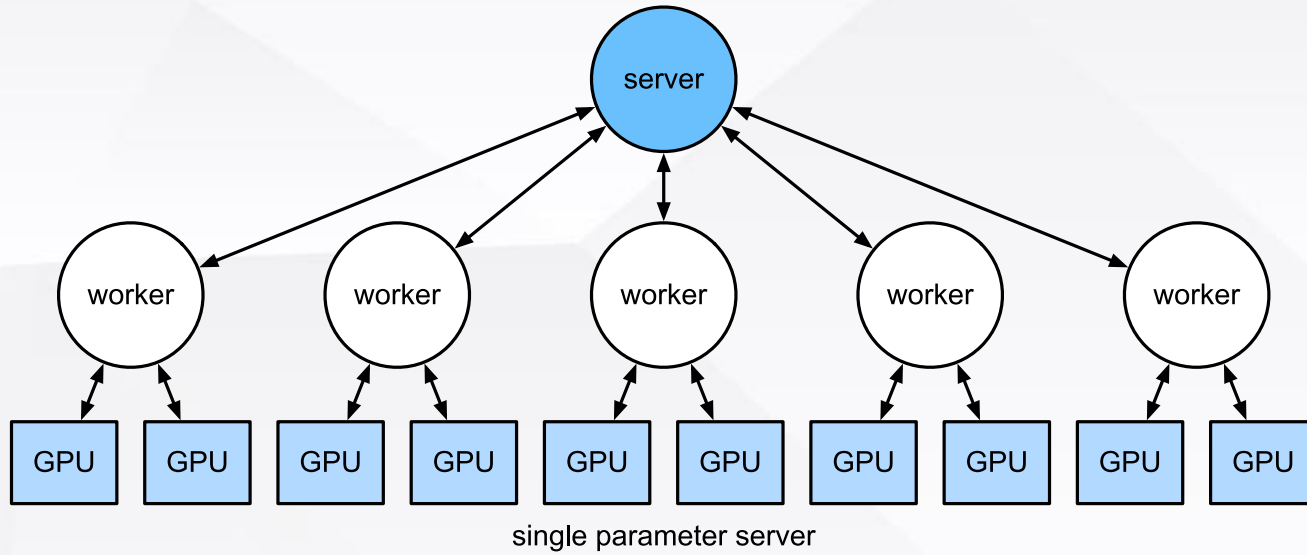


PS vs Bulk Synchronous system





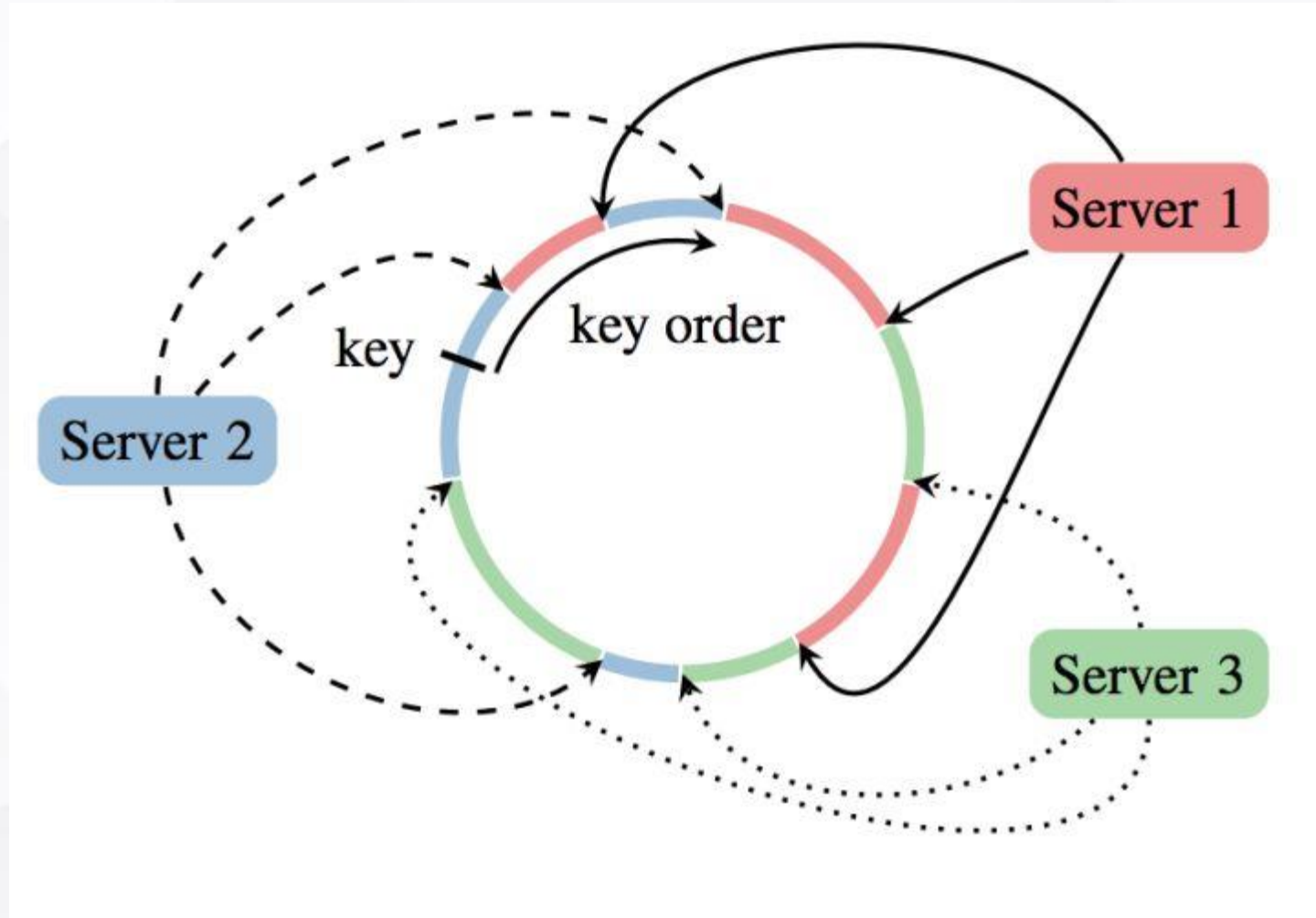
Coordinate of Multi-PS



- a single parameter server is a bottleneck since its bandwidth is finite.
- multiple parameter servers store parts of the parameters with aggregate bandwidth.



Consistent Hash





02

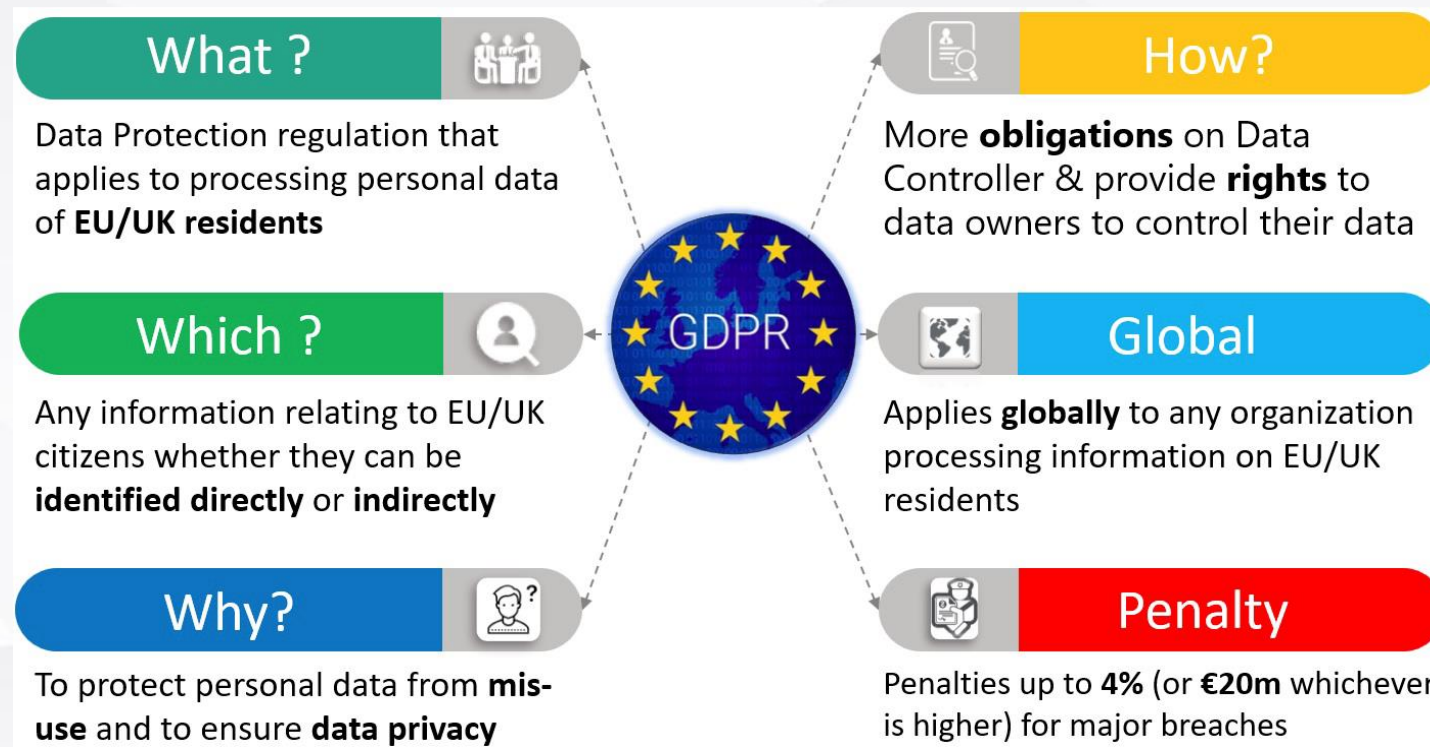
Privacy and Security Issues in Deep Learning

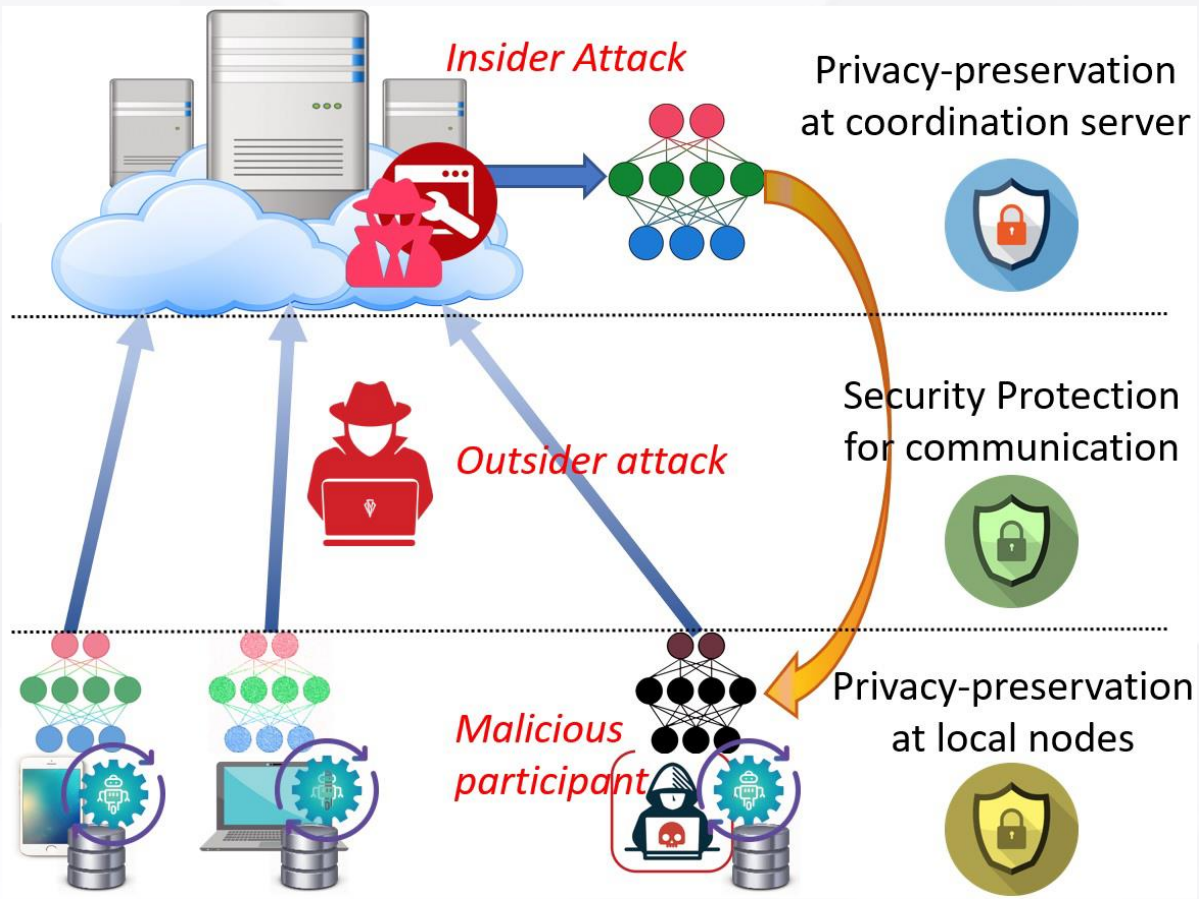


Privacy Principles for Learning and Analytics



- The challenges are not only due to transporting high-volume, high-velocity, high-veracity, and heterogeneous data across organisations but also the data protection regulations and restrictions such as the EU General Data Protection Regulation (GDPR)





- ① prevent insiders at the server from conducting inference attacks
- ② prevent Byzantine participants from conducting model poisoning



Privacy inference attacks

- Existing work suggested that adversaries can infer different levels of sensitive information from the updated gradients

Eavesdropping attacks

- The adversaries located in the communication channel between central server and local workers can launch eavesdropping attacks. The adversaries can steal or tamper some meaningful information, such as model weights or gradients, in each communication.

Poisoning attacks

- Poisoning attacks on machine learning models have been widely studied. These attacks occur in the training phase against FL. On the one hand, adversaries can impair the performance of the final global model on untargeted tasks. On the other hand, adversaries can inject a backdoor into the final global model.



03

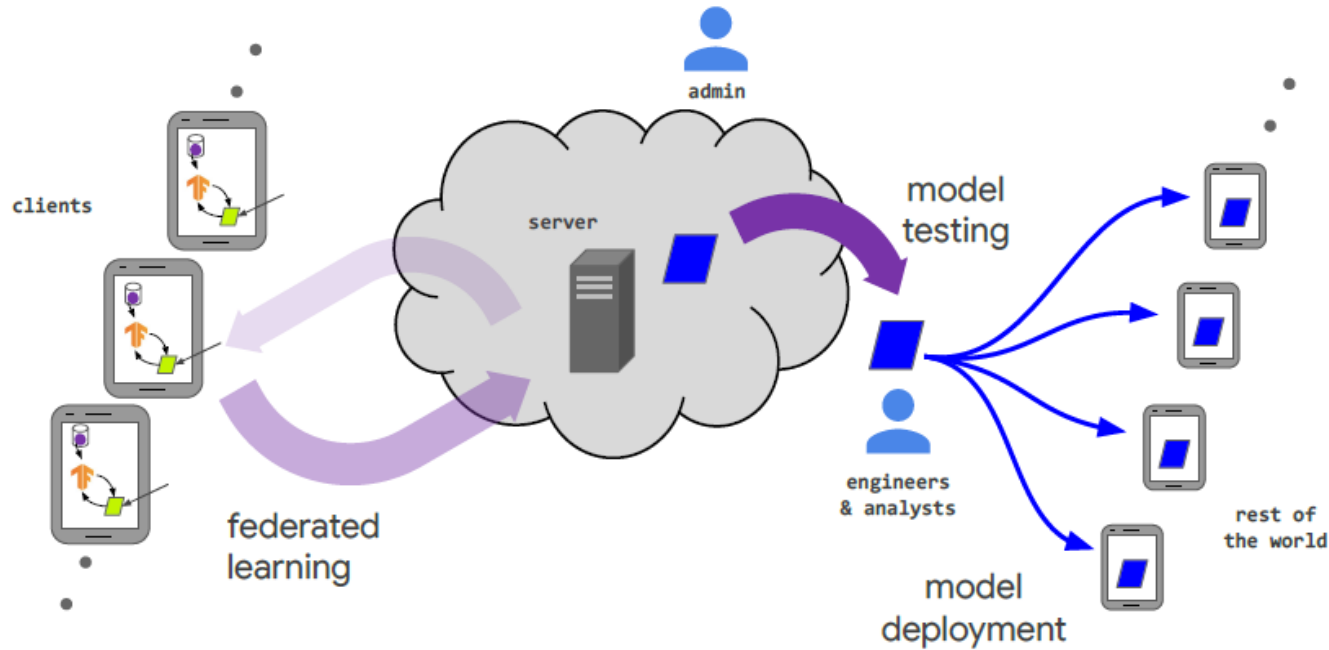
Strength of Federated Learning



What is Federated Learning?



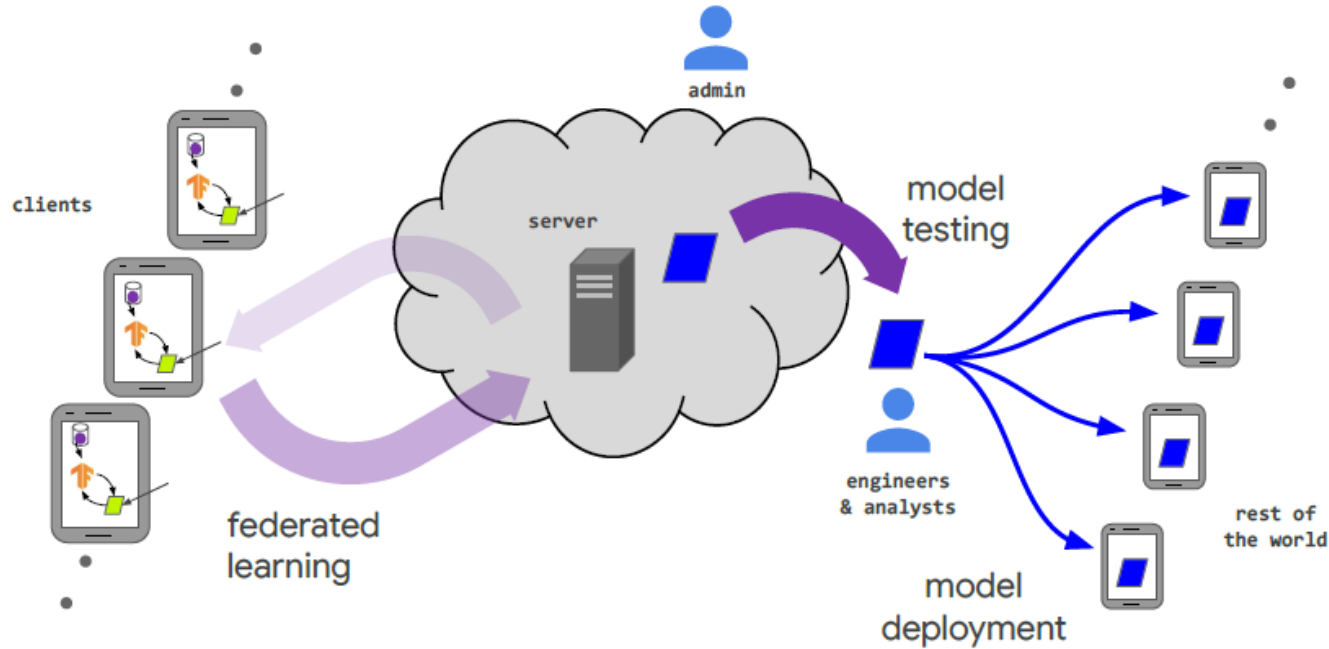
- The term federated learning was introduced in 2016 by McMahan et al:
 - “We term our approach Federated Learning, since the learning task is solved by a loose federation of participating devices (which we refer to as clients) which are coordinated by a central server.”
- A broader definition
 - Federated learning is a machine learning setting where multiple entities (clients) collaborate in solving a machine learning problem, under the coordination of a central server or service provider. Each client’s raw data is stored locally and not exchanged or transferred; instead, focused updates intended for immediate aggregation are used to achieve the learning objective



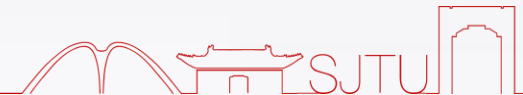
- ❶ **Problem identification:** The model engineer identifies a problem to be solved with FL.
- ❷ **Client instrumentation:** If needed, the clients (e.g. an app running on mobile phones) are instrumented to store locally (with limits on time and quantity) the necessary training data.



Life Cycle of Federated Learning#2

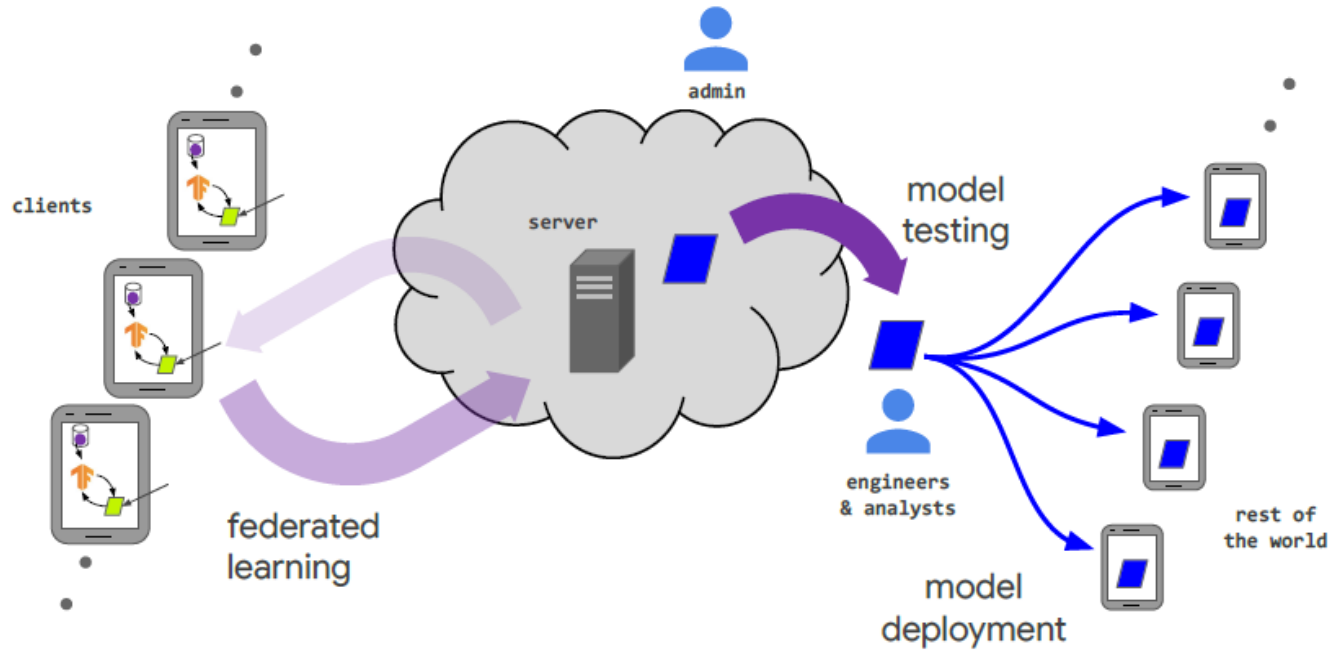


- ❶ **Simulation prototyping** (optional): The model engineer may prototype model architectures and test learning hyperparameters in an FL simulation using a proxy dataset.
- ❷ **Federated model training**: Multiple federated training tasks are started to train different variations of the model, or use different optimization hyperparameters.





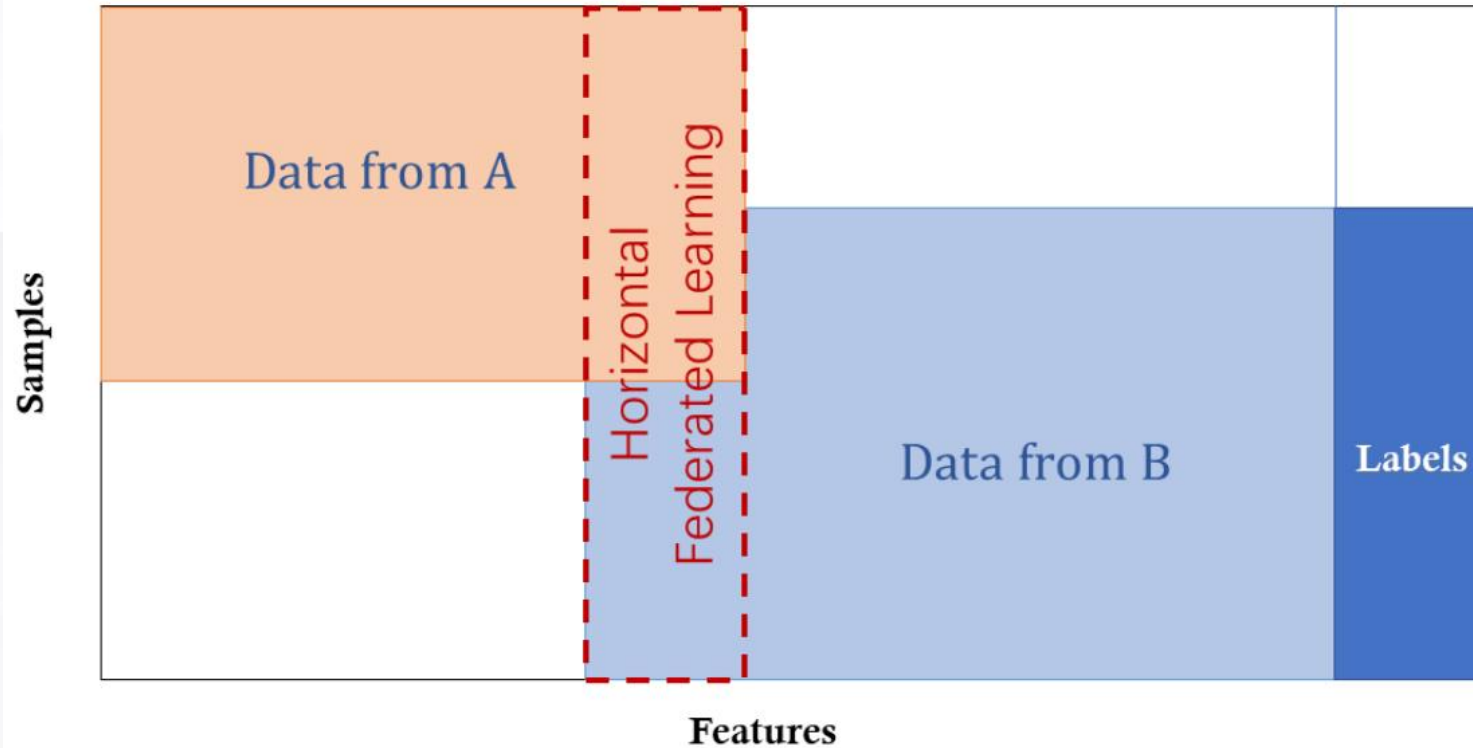
Life Cycle of Federated Learning#3



- ❶ (Federated) **model evaluation**: After the tasks have trained sufficiently (typically a few days), the models are analyzed and good candidates selected. Analysis may include metrics computed on standard datasets in the datacenter, or federated evaluation wherein the models are pushed to held-out clients for evaluation on local client data.



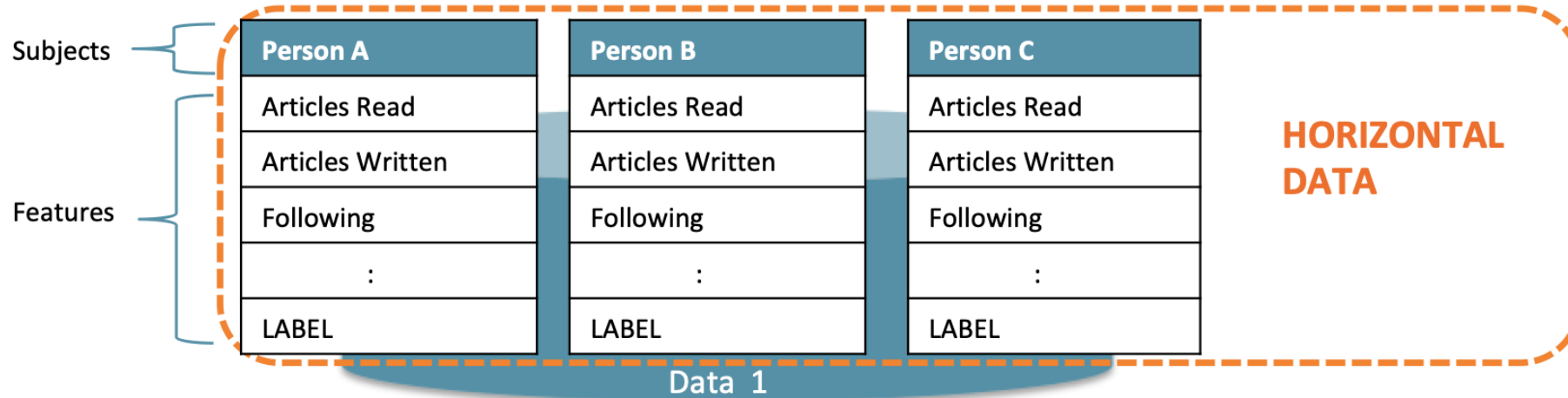
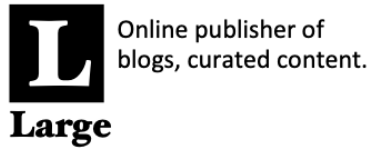
Classification



“Horizontal federated learning, or sample-based federated learning, is introduced in the scenarios that data sets share the same feature space but different in sample.”



Horizontal data example

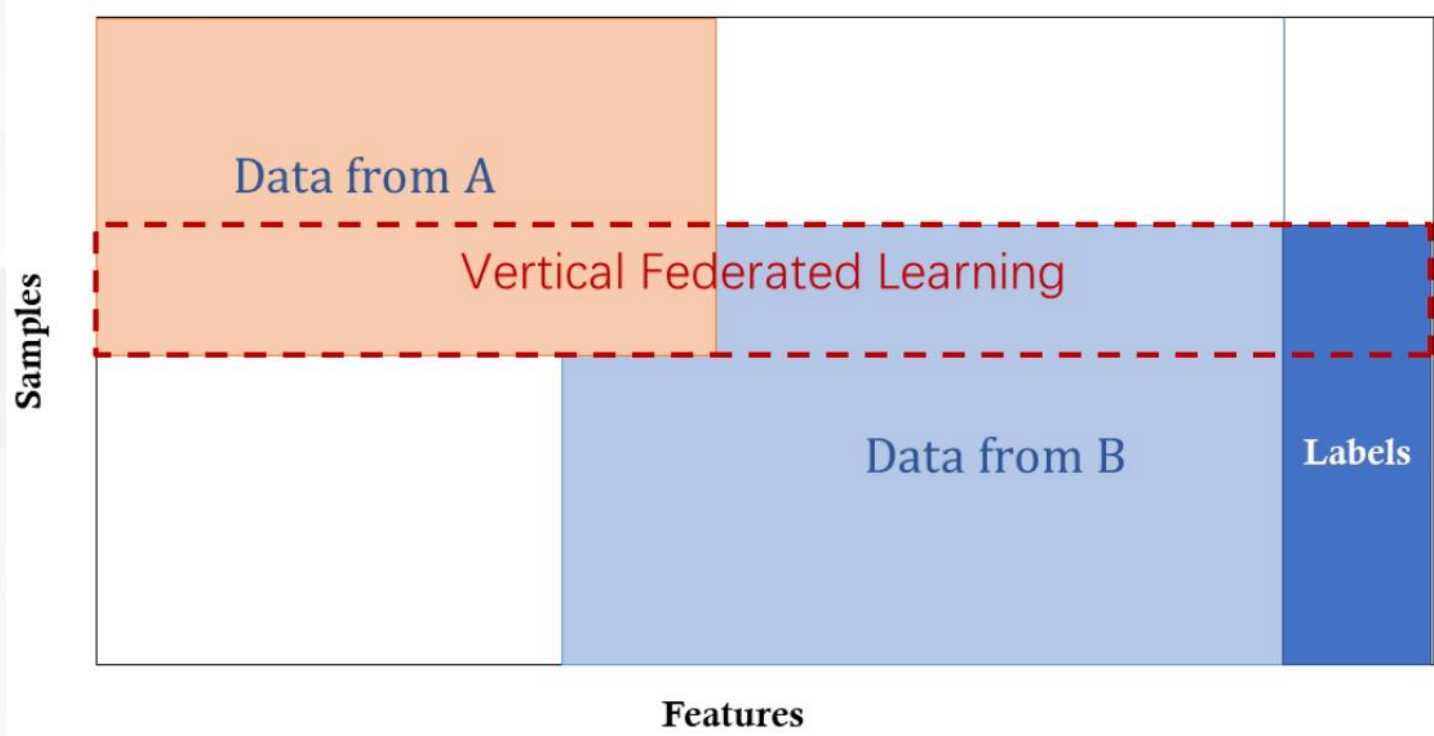


With horizontal data, rows of data are available with a consistent set of features. This is exactly the type of data you'd feed into a supervised machine learning task.





Classification



“Vertical federated learning or feature-based federated learning ... is applicable to the cases that two data sets share the same sample ID space but differ in feature space.”

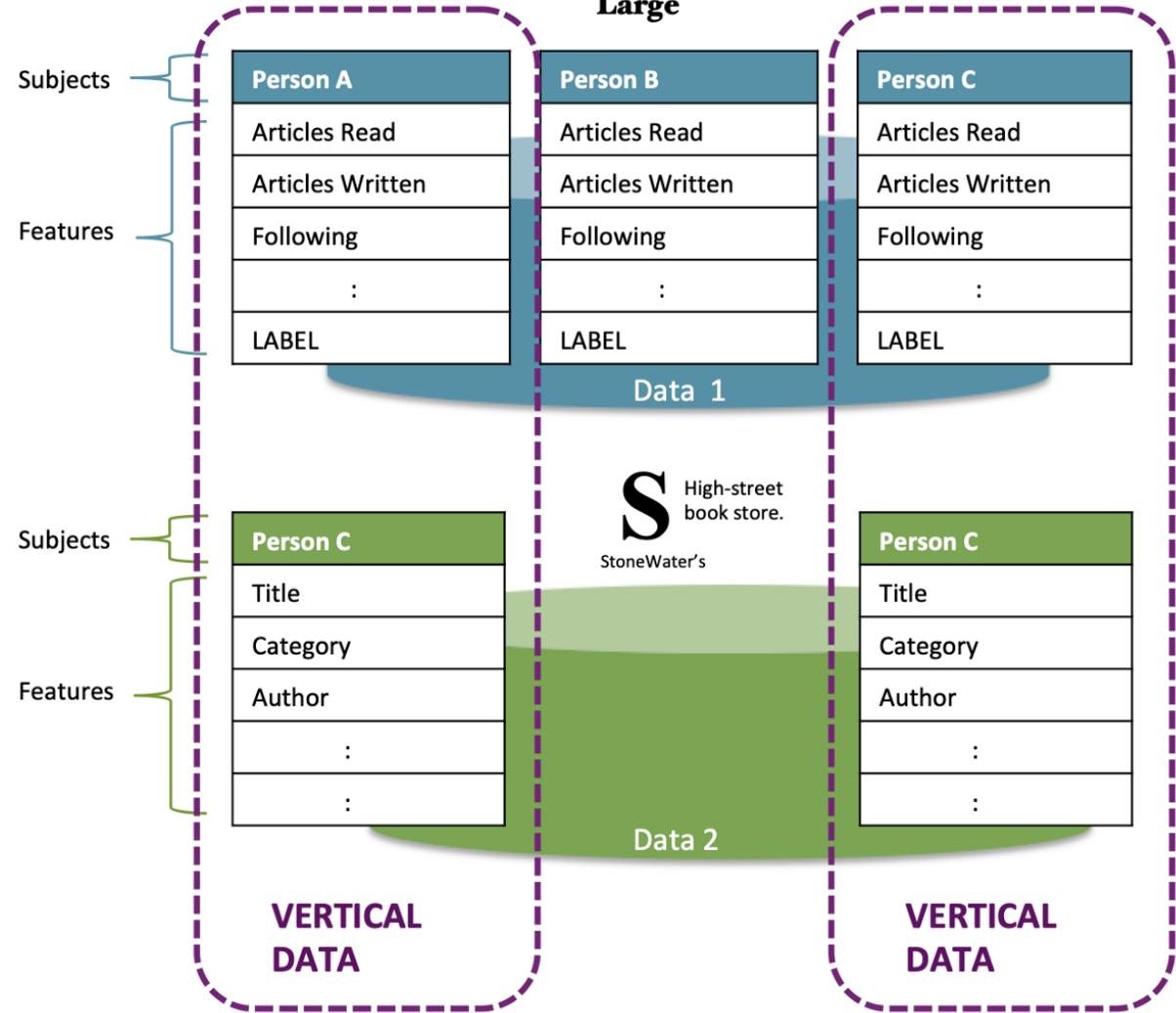



Vertical data example



L Online publisher of blogs, curated content.
Large

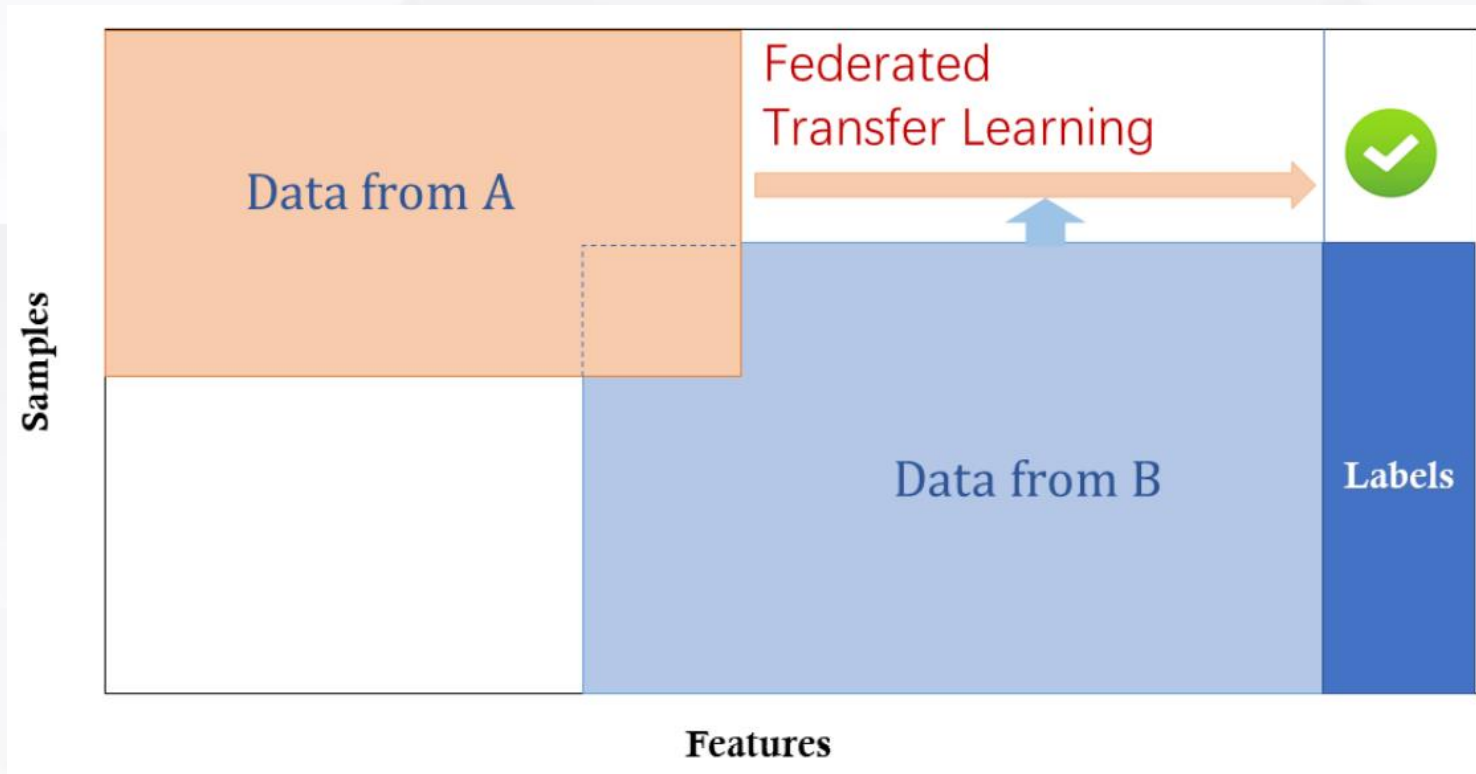
S High-street book store.
 StoneWater's



 fictitious high street book retailer StoneWater's have some of the same customers as online blogging curator Large (also totally fictitious) and capture different features such as book 'Title', 'Category', 'Author' from each purchase.



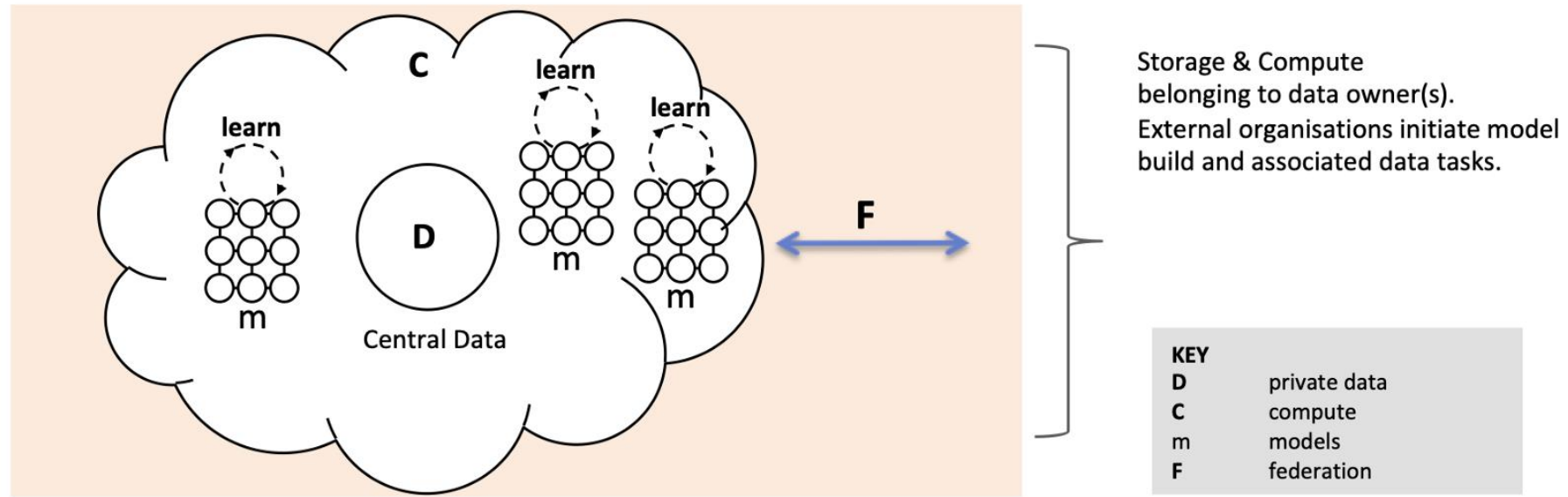
Classification



“Transfer federated learning is applicable to the cases that two data sets share some same sample ID space and feature space.”



Data Centric Federated Learning



- ⊗ This is a newer, emerging type of Federated Learning, and in some ways may be outgrowing the Federated term, having a more peer-to-peer feel.
- ⊗ An owner, or in future—owners, of private data can provide access for external organisations to build models on their data without sharing that data.

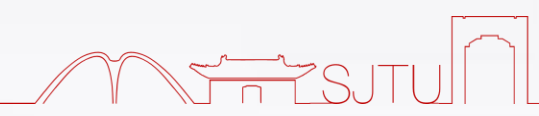


Comparison



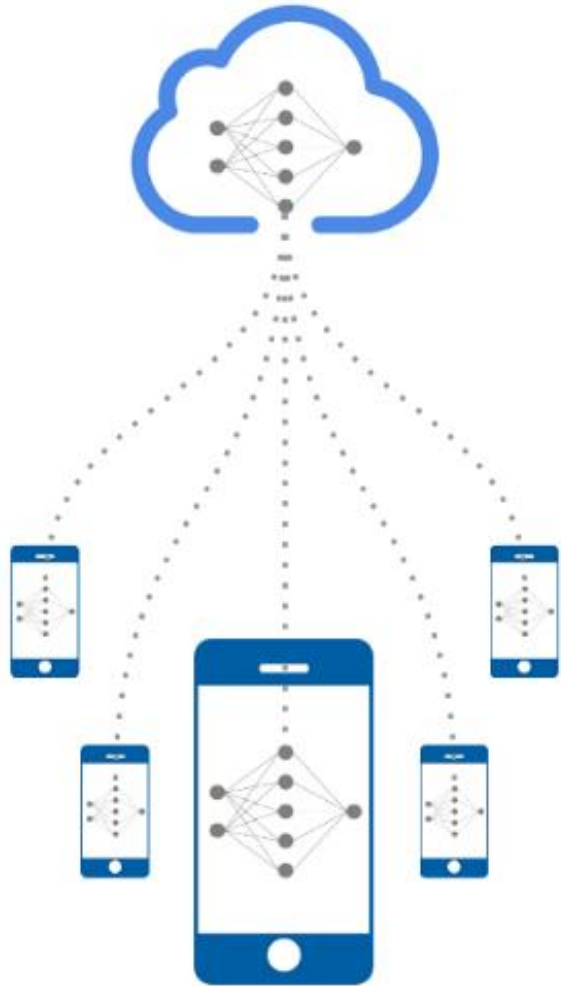
TABLE 1: TYPICAL FL SETTINGS AND OF TRADITIONAL DISTRIBUTED LEARNING

	DATACENTER DISTRIBUTED LEARNING	CROSS-SILO FEDERATED LEARNING	CROSS-DEVICE FEDERATED LEARNING
Setting	Training a model on a large but “flat” dataset. Clients are compute nodes in a single cluster or datacenter.	Training a model on siloed data. Clients are different organizations (e.g., medical or financial) or datacenters in different geographical regions.	The clients are a very large number of mobile or IoT devices.
Data distribution	Data is centrally stored, so it can be shuffled and balanced across clients. Any client can read any part of the dataset.	Data is generated locally and remains decentralized. Each client stores its own data and cannot read the data of other clients. Data is not independently or identically distributed.	
Orchestration	Centrally orchestrated.	A central orchestration server/service organizes the training, but never sees raw data.	
Distribution scale	Typically 1 - 1000 clients.	Typically 2 - 100 clients.	Up to 10^{10} clients.
Client properties	Clients are reliable and almost always available to participate in computations. Clients may be directly addressed, and can maintain state across computation rounds.		Clients are often unavailable and can only be accessed by random sampling from available devices. For large populations a single client will typically only participate once in a given computation.





Strengths



Hyper-Personalized



Low Cloud Infra Overheads



Minimum Latencies



Privacy Preserving



Strengths



- ① FL enables devices like mobile phones to collaboratively learn a shared prediction model while keeping the training data on the device instead of requiring the data to be uploaded and stored on a central server.
- ① Moves model training to the edge, namely devices such as smartphones, tablets, IoT, or even “organizations” like hospitals that are required to operate under strict privacy constraints.
- ① Makes real-time prediction possible, since prediction happens on the device itself.
- ① Since the models reside on the device, the prediction process works even when there is no internet connectivity.
- ① FL reduces the amount of hardware infrastructure required.



Various technologies for Privacy

Technologies	Characteristic
Differential Privacy (local, central, shuffled, aggregated, and hybrid models)	A quantification of how much information could be learned about an individual from the output of an analysis on a dataset that includes the user. Algorithms with differential privacy necessarily incorporate some amount of randomness or noise, which can be tuned to mask the influence of the user on the output.
Secure Multi-Party Computation	Two or more participants collaborate to simulate, through cryptography, a fully trusted third party who can: <ul style="list-style-type: none">• Compute a function of inputs provided by all the participants;• Reveal the computed value to a chosen subset of the participants, with no party learning anything further.



Various technologies for Privacy

Technologies	Characteristic
Homomorphic Encryption	Enables a party to compute functions of data to which they do not have plain-text access, by allowing mathematical operations to be performed on ciphertexts without decrypting them. Arbitrarily complicated functions of the data can be computed this way (“Fully Homomorphic Encryption”) though at greater computational cost.
Trusted Execution Environments (secure enclaves)	<ul style="list-style-type: none">• Confidentiality: The state of the code’s execution remains secret, unless the code explicitly publishes a message;• Integrity: The code’s execution cannot be affected, except by the code explicitly receiving an input;• Measurement/Attestation: The TEE can prove to a remote party what code (binary) is executing and what its starting state was, defining the initial conditions for confidentiality and integrity.



04

FedAvg Algorithm



Key Properties



Non-IID

- The training data on a given client is typically based on the usage of the mobile device by a particular user, and hence any particular user's local dataset will not be representative of the population distribution.

Unbalanced

- Similarly, some users will make much heavier use of the service or app than others, leading to varying amounts of local training data.

Massively distributed

- We expect the number of clients participating in an optimization to be much larger than the average number of examples per client.

Limited communication

- Mobile devices are frequently offline or on slow or expensive connections.





Objective Function



$$\min_{w \in \mathbb{R}^d} f(w) \quad \text{where} \quad f(w) \stackrel{\text{def}}{=} \frac{1}{n} \sum_{i=1}^n f_i(w)$$

⊗ $f_i(w) = l(x_i; y_i; w)$, that is, the loss of the prediction on example $(x_i; y_i)$ made with model parameters w .

⊗ We assume there are K clients over which the data is partitioned, with \mathcal{P}_k the set of indexes of data points on client k . Thus, we can re-write the objective

$$f(w) = \sum_{k=1}^K \frac{n_k}{n} F_k(w) \quad \text{where} \quad F_k(w) = \frac{1}{n_k} \sum_{i \in \mathcal{P}_k} f_i(w)$$



From SGD to FedAvg



$$F_k(w) = \frac{1}{n_k} \sum_{i \in \mathcal{P}_k} f_i(w)$$

$$g_k = \nabla F_k(w_t)$$

$$w_{t+1} \leftarrow w_t - \eta \sum_{k=1}^K \frac{n_k}{n} g_k$$

$$\forall k, w_{t+1}^k \leftarrow w_t - \eta g_k$$

$$w_{t+1} \leftarrow \sum_{k=1}^K \frac{n_k}{n} w_{t+1}^k$$

Algorithm 1 FederatedAveraging. The K clients are indexed by k ; B is the local minibatch size, E is the number of local epochs, and η is the learning rate.

Server executes:

initialize w_0

for each round $t = 1, 2, \dots$ **do**

$m \leftarrow \max(C \cdot K, 1)$

$S_t \leftarrow$ (random set of m clients)

for each client $k \in S_t$ **in parallel do**

$w_{t+1}^k \leftarrow$ ClientUpdate(k, w_t)

$w_{t+1} \leftarrow \sum_{k=1}^K \frac{n_k}{n} w_{t+1}^k$

ClientUpdate(k, w): // Run on client k

$\mathcal{B} \leftarrow$ (split \mathcal{P}_k into batches of size B)

for each local epoch i from 1 to E **do**

for batch $b \in \mathcal{B}$ **do**

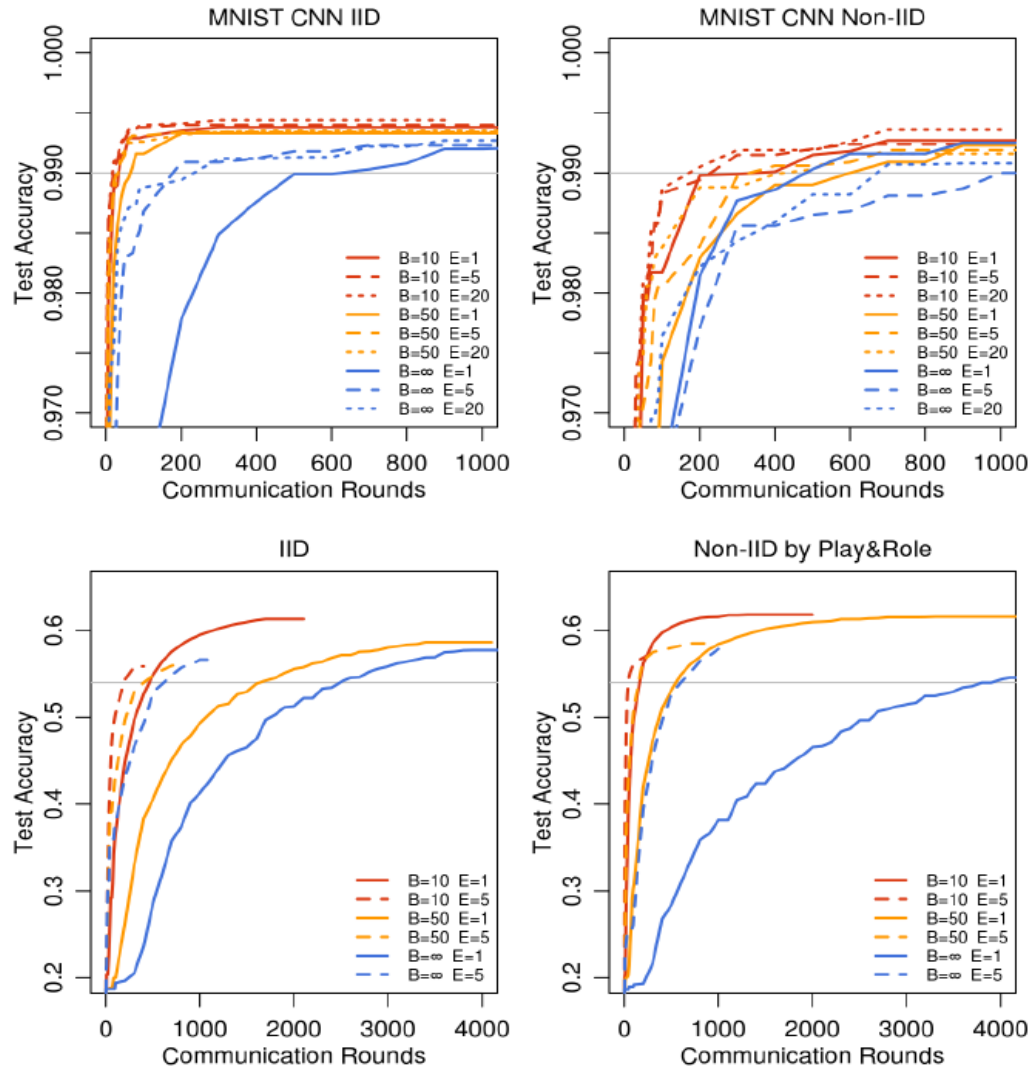
$w \leftarrow w - \eta \nabla \ell(w; b)$

return w to server





Test Result

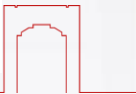
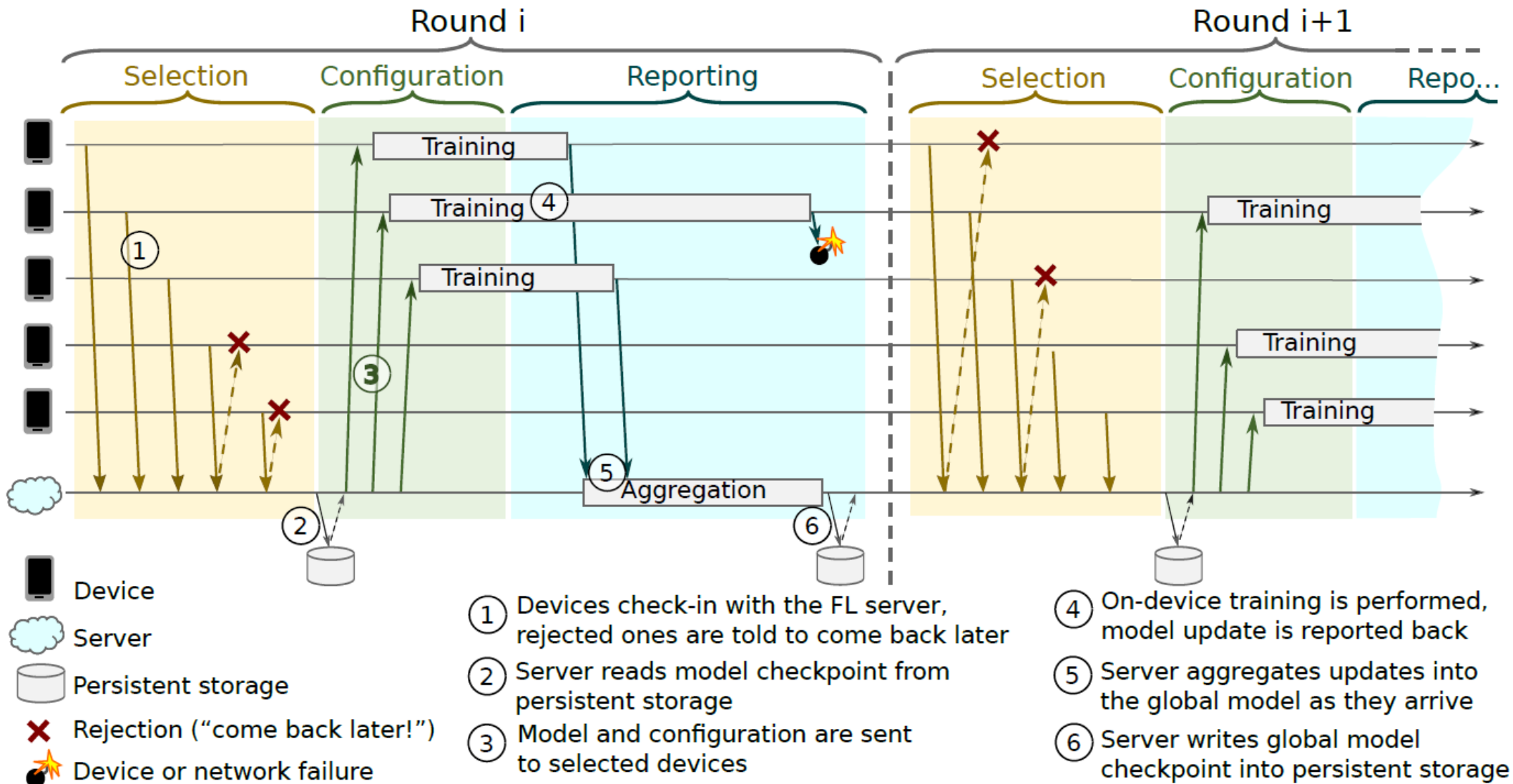


- Test set accuracy vs. communication rounds for the MNIST CNN (IID and then pathological non-IID) and Shakespeare LSTM (IID and then by Play&Role) with

- $C = 0.1$ and optimized η .

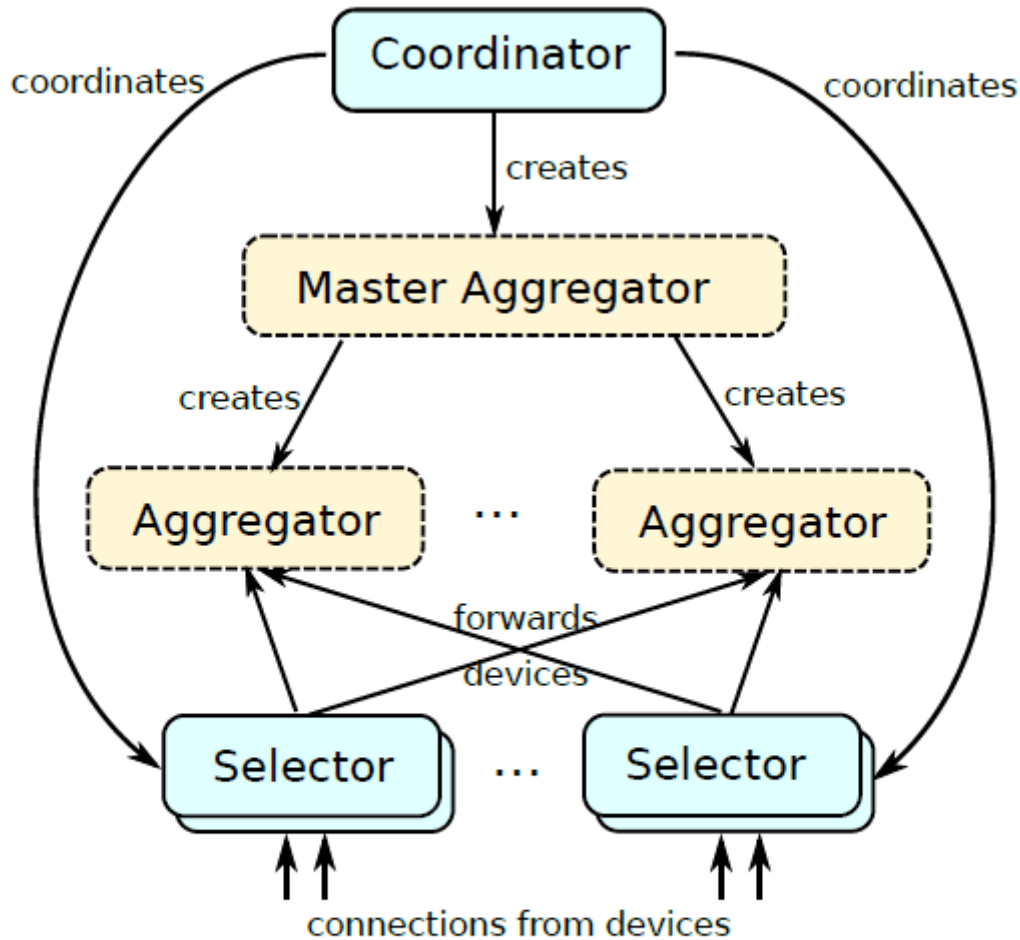




Federated Learning Protocol





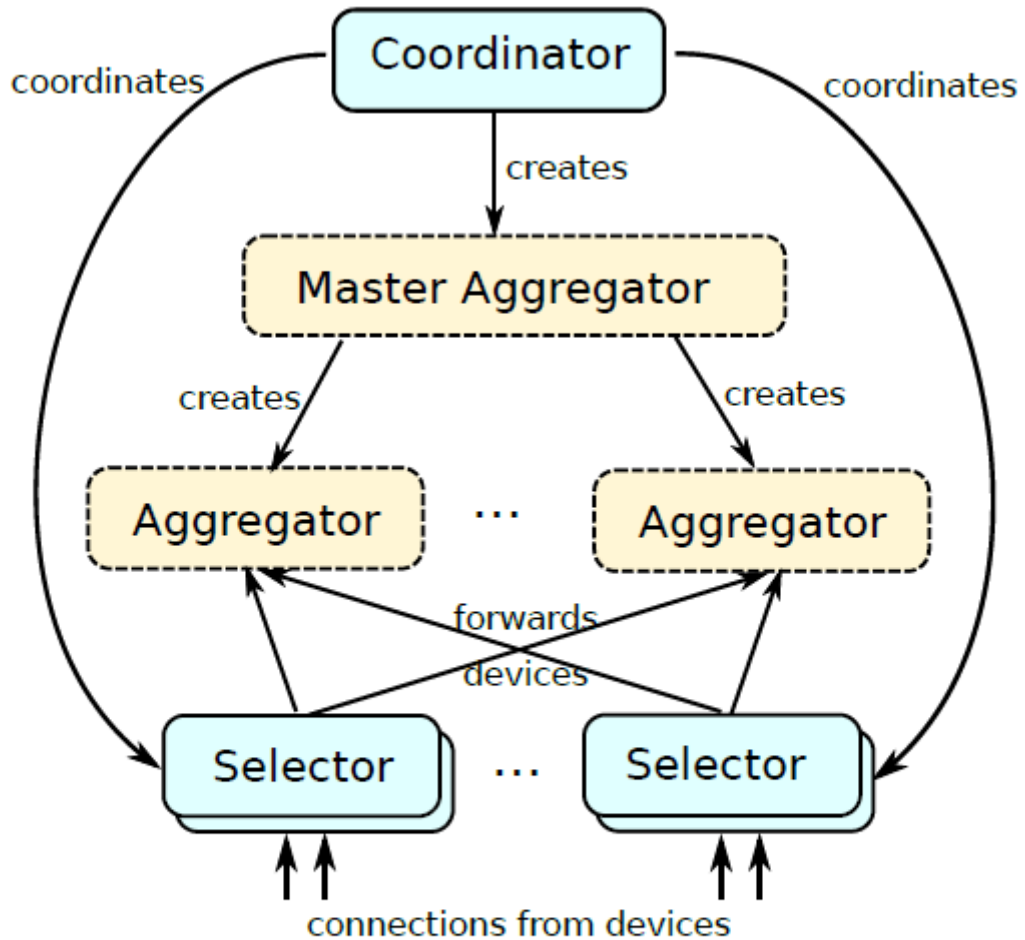
Actor Model of FL





-  Persistent (long-lived) actor
-  Ephemeral (short-lived) actor

Coordinators

- The top-level actors which enable global synchronization and advancing rounds in lockstep.
- There are multiple Coordinators, and each one is responsible for an FL population of devices.

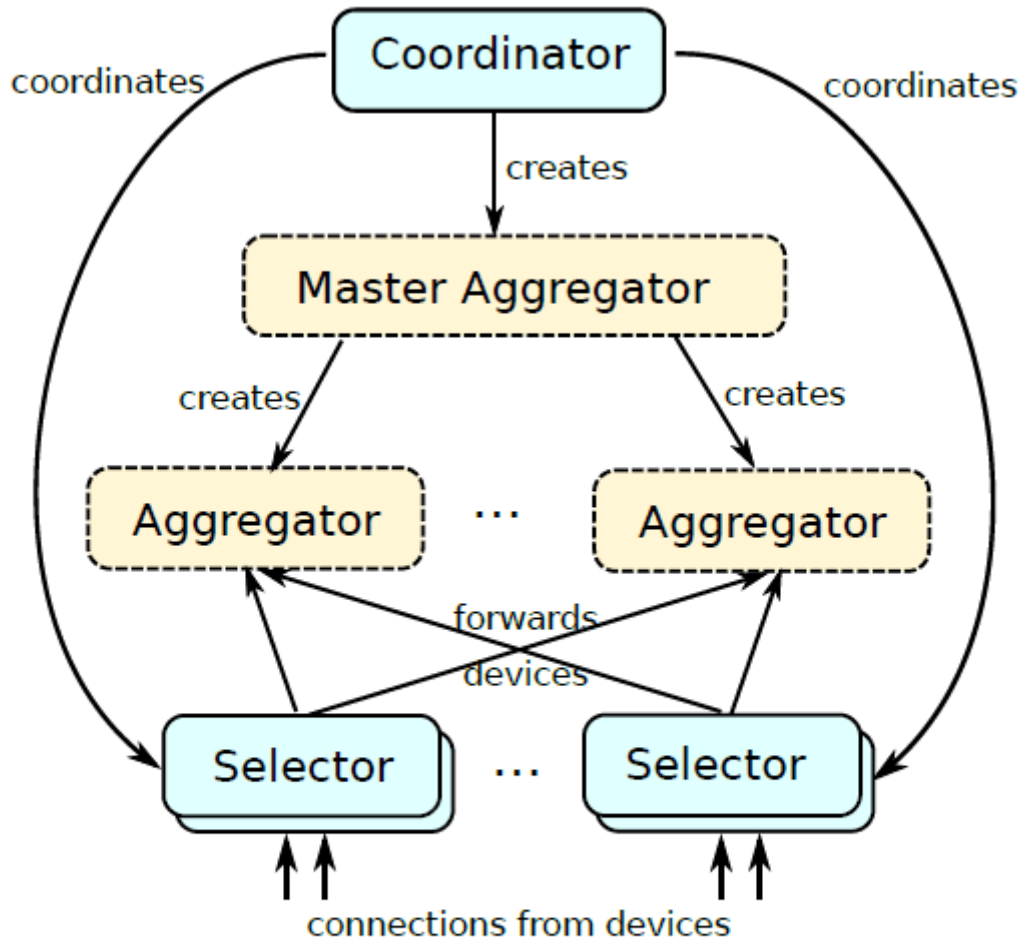


 Persistent (long-lived) actor

 Ephemeral (short-lived) actor

Selectors

- Responsible for accepting and forwarding deviceconnections.
- They periodically receive information from the Coordinator about how many devices are needed for each FL population, which they use to make local decisions about whether or not to accept each device.



□ Persistent (long-lived) actor
□ Ephemeral (short-lived) actor

Master Aggregators

- Manage the rounds of each FL task.
- In order to scale with the number of devices and update size, they make dynamic decisions to spawn one or more Aggregators to which work is delegated.



上海交通大學

SHANGHAI JIAO TONG UNIVERSITY

谢谢!

饮水思源 爱国荣校