Pisces:

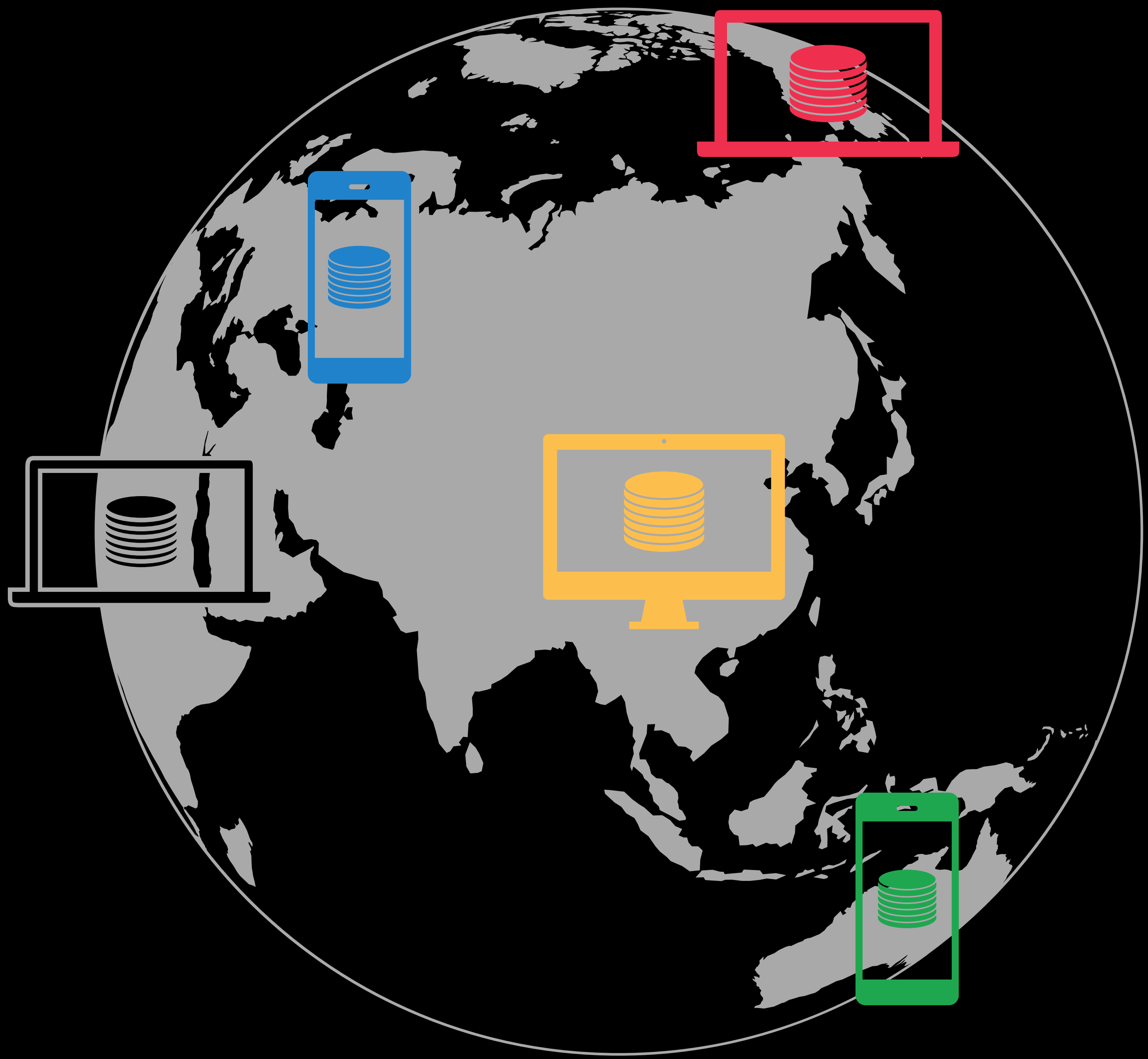
Efficient Federated Learning via Guided Asynchronous Training

Zhifeng Jiang, Wei Wang, Baochun Li, Bo Li

THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

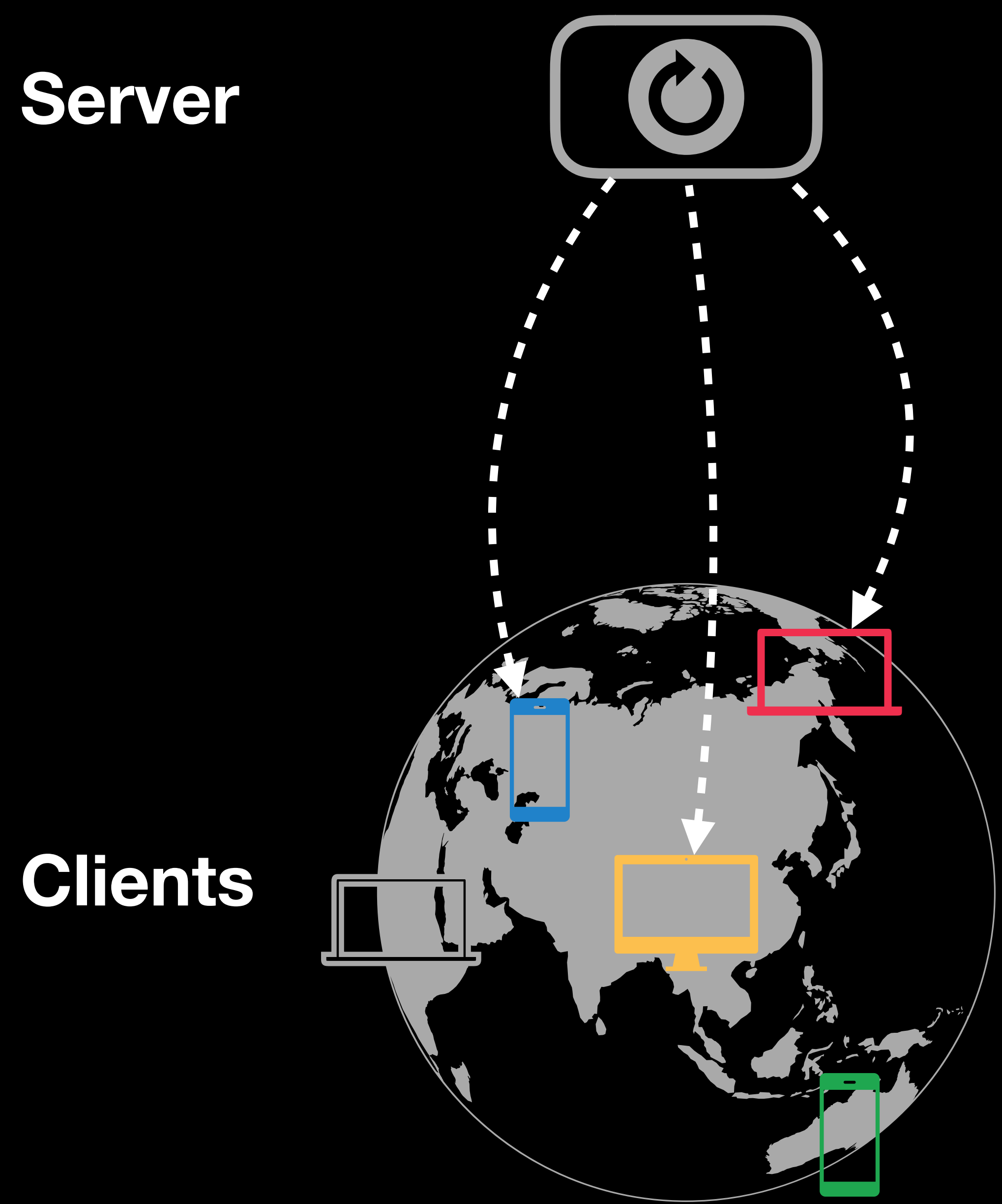
UNIVERSITY OF TORONTO

Federated Learning (FL)



Enable distributed clients to train a global model without revealing their data.

Federated Learning (FL)

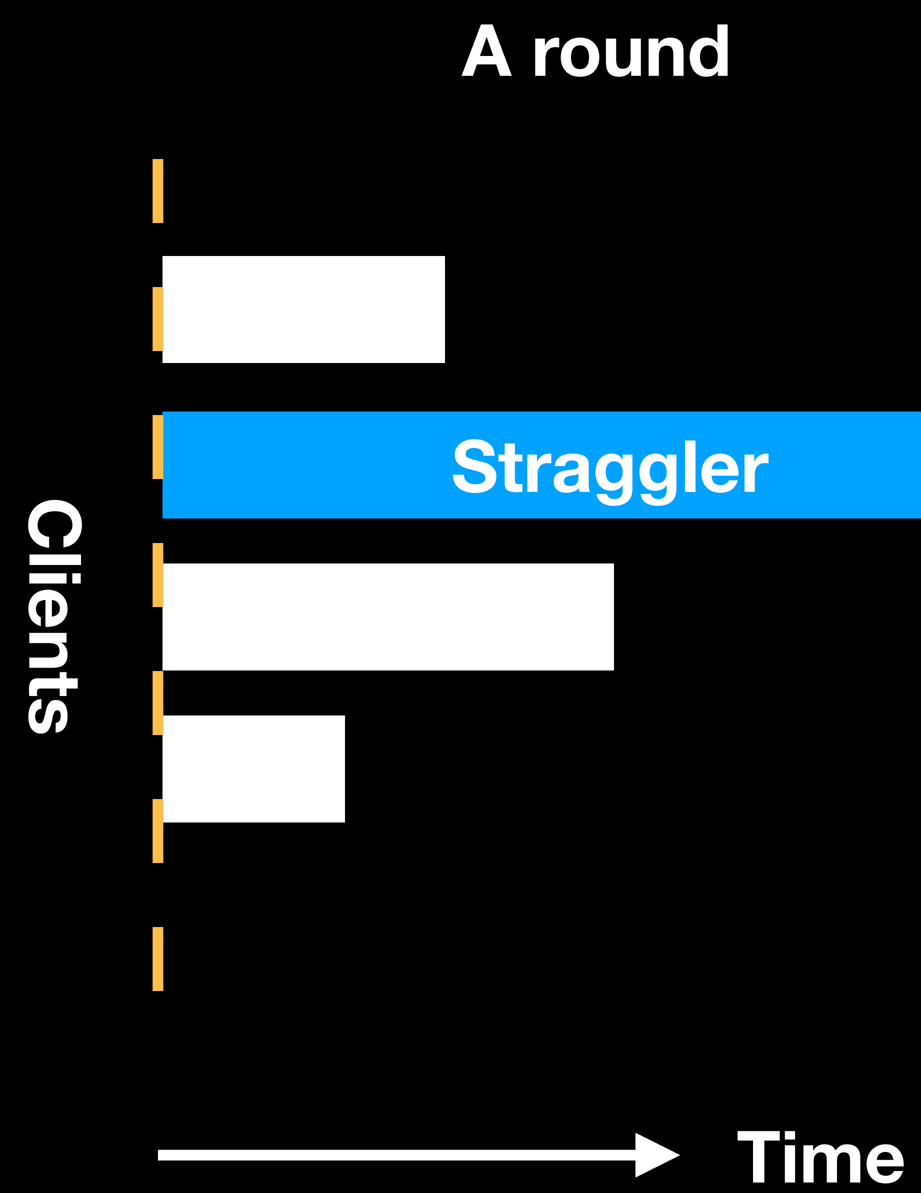


For each round

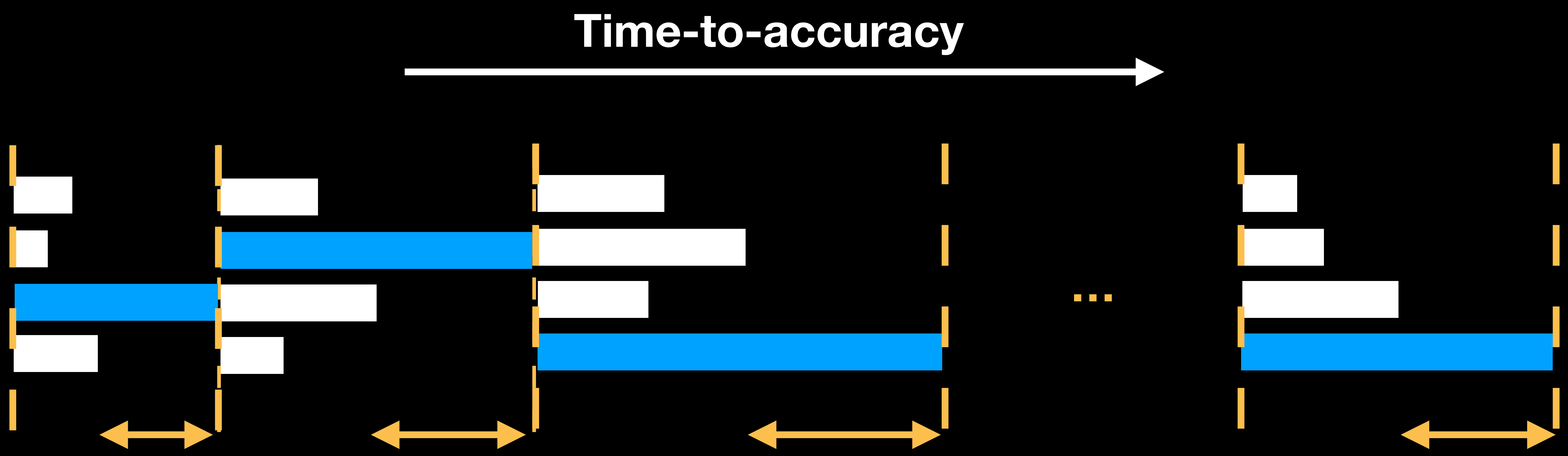
1. Participant selection
2. Local training
3. Model aggregation

Synchronous

The Straggler Problem in Sync FL



The Straggler Problem in Sync FL



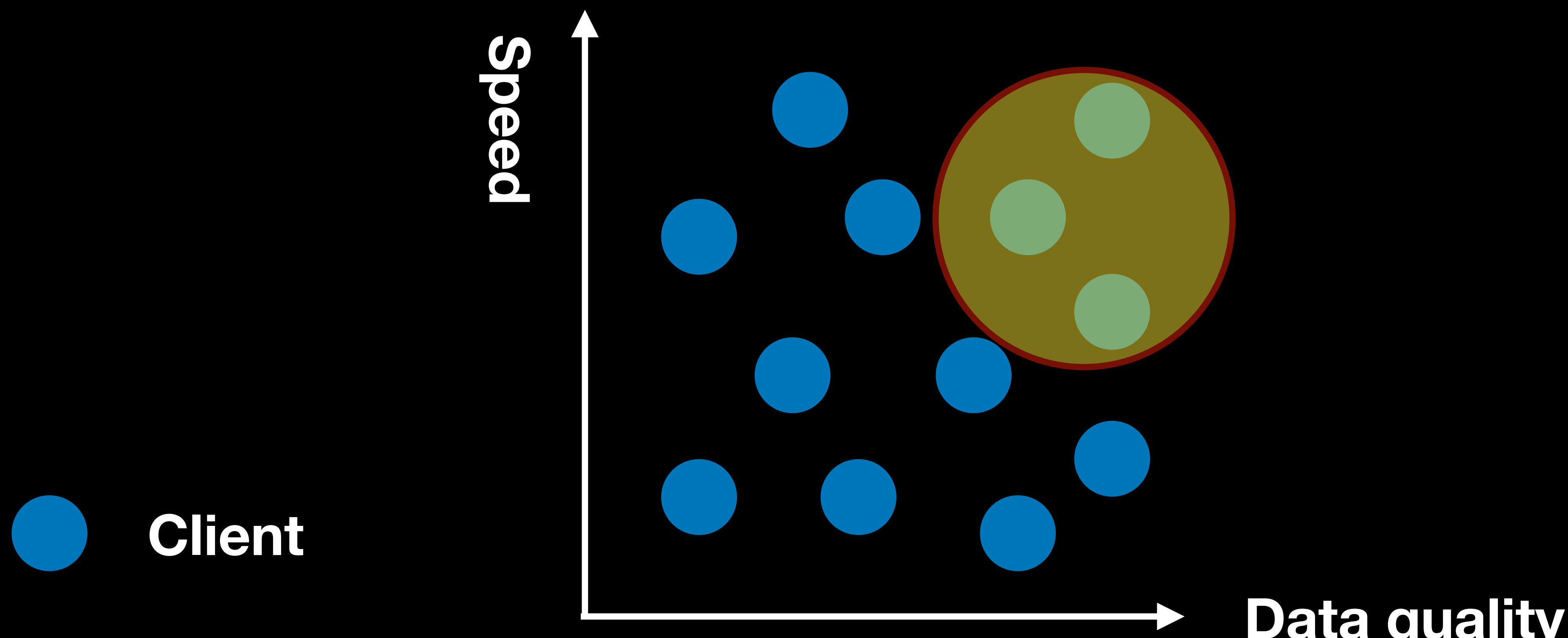
Idle waiting: 33.2% to 57.2%

Fix Sync FL by Participant Selection

Existing work: Reconcile the demands for

Speed & Data quality

Client utility in Oort[1] = Speed × Data quality

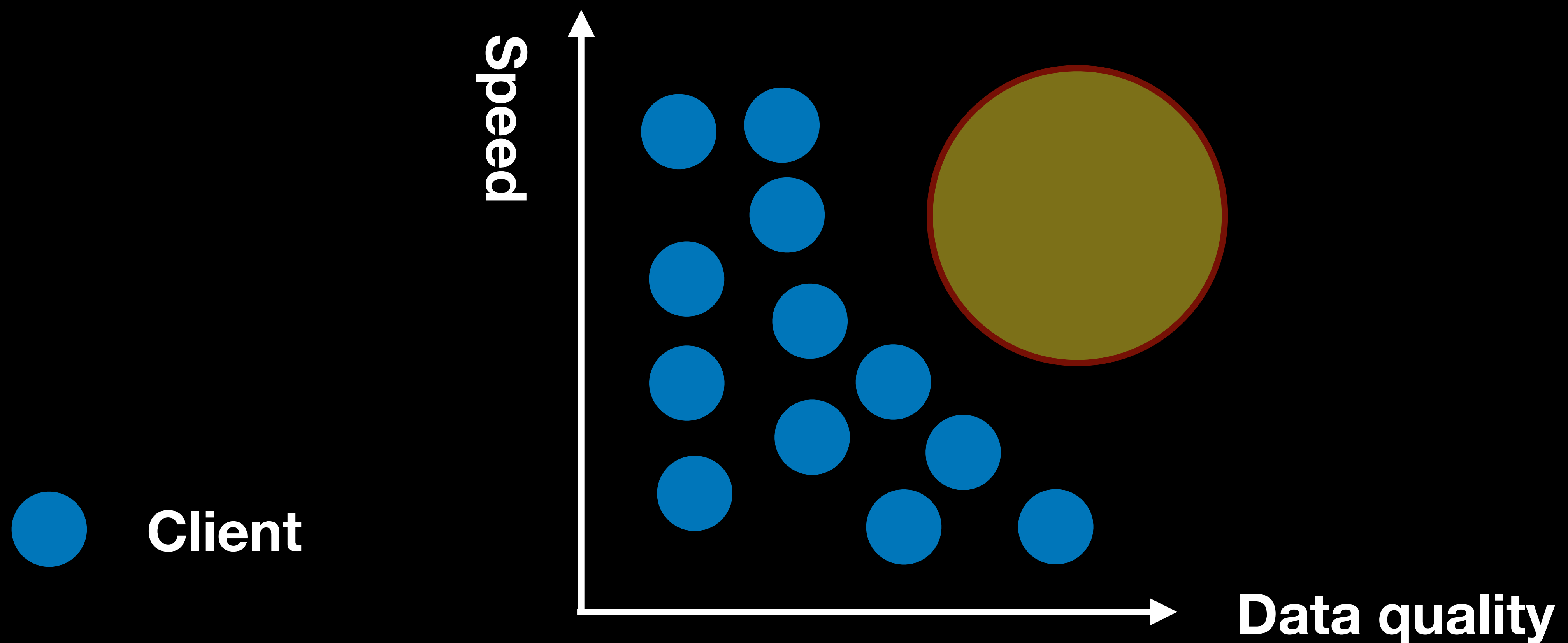


[1] Efficient federated learning via guided participant selection, OSDI'21

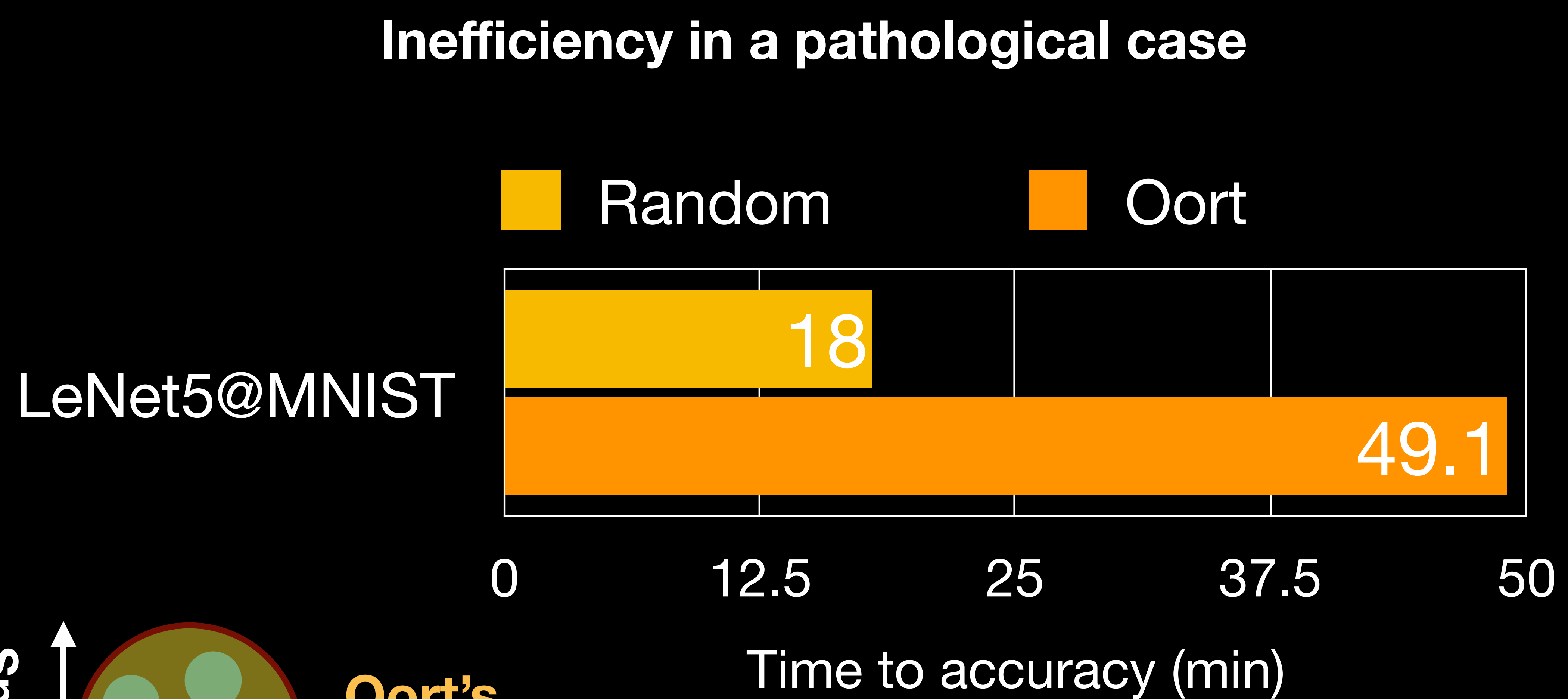
Fix Sync FL by Participant Selection

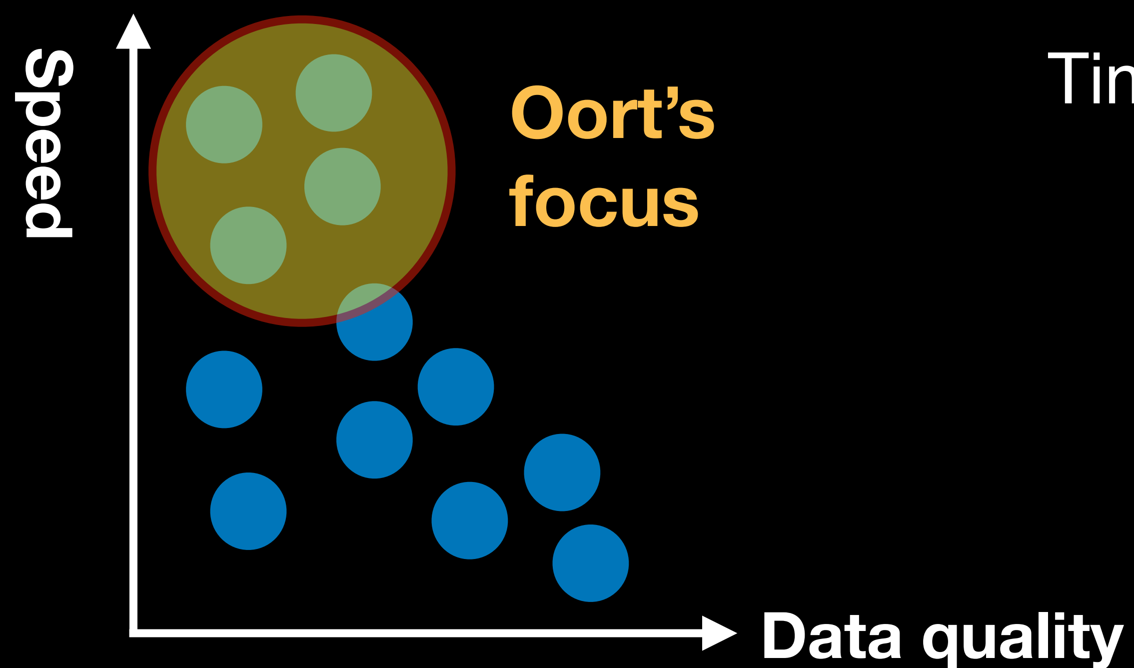
Inefficiency in a pathological case

Speed and data quality are inversely correlated



Fix Sync FL by Participant Selection



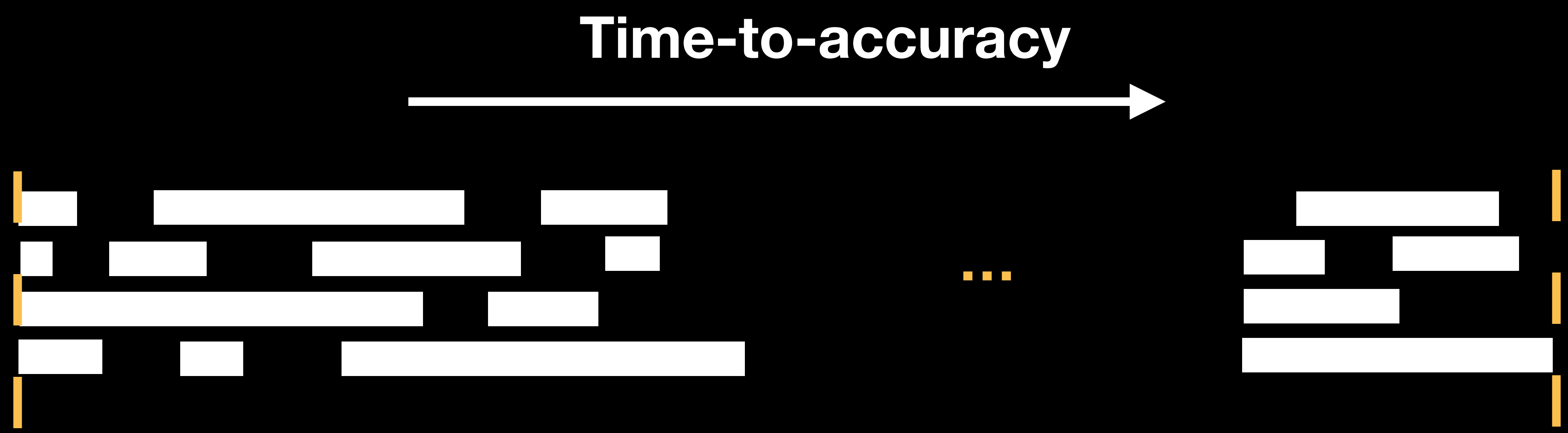


Fix Sync FL by Participant Selection

Inefficiency in a pathological case

Intrinsically hard to navigate in synchronous FL

Call for Async FL

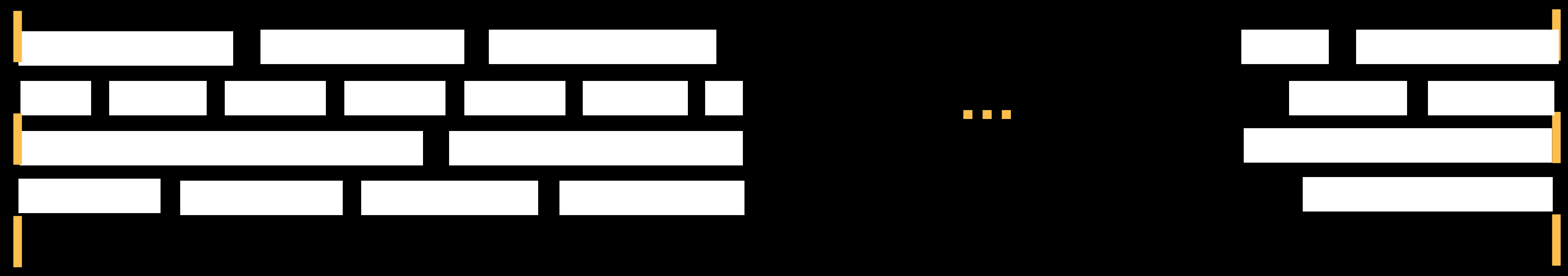


Freedom in
- Participant selection
- Model aggregation

More tolerance for slow clients → Less tension between data quality and speeds

Challenges in Async FL

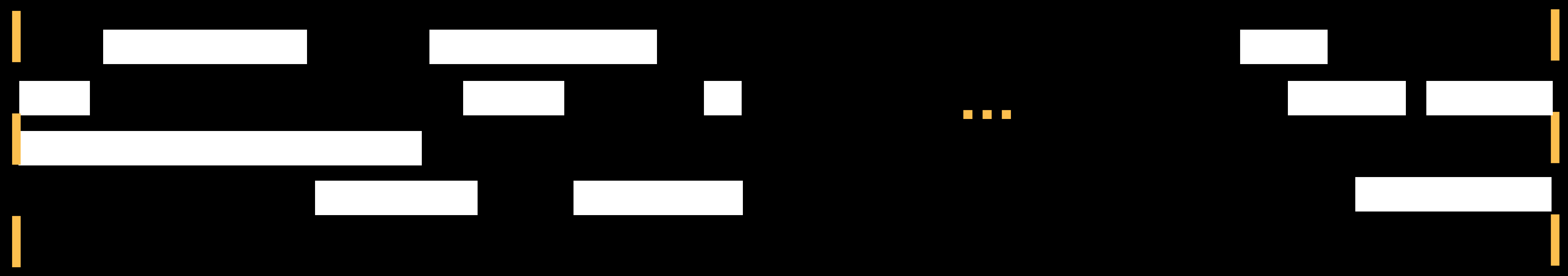
- Freedom in selection allows all clients to run concurrently
- High concurrency yields marginal gain



Challenges in Async FL

#1: How should concurrency quotas be fully utilized?

- Freedom in selection allows all clients to run concurrently
- High concurrency yields marginal gain



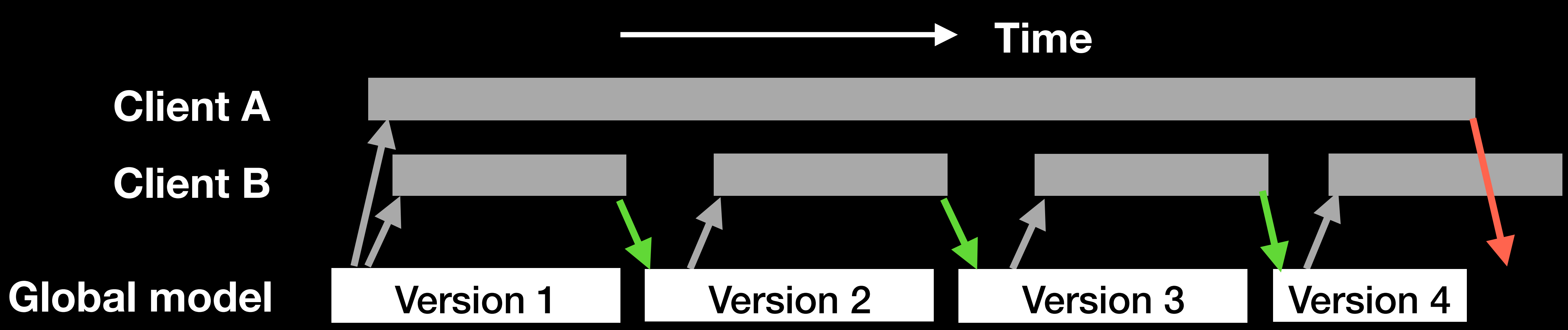
Resource Efficiency ✓

Challenges in Async FL

#1: How should concurrency quotas be fully utilized?

- Freedom in selection solicits high concurrency
- High concurrency yields marginal gain

- Freedom in aggregation solicits stale local updates



Challenges in Async FL

#1: How should concurrency quotas be fully utilized?
- Freedom in selection solicits high concurrency
- High concurrency yields marginal gain

#2: How should stale local updates be avoided?
- Freedom in aggregation solicits stale local updates
- Local updates with high staleness harms the convergence

Participant Selection

Prefer clients with high

Aggregate training loss[1]

Vulnerable to corrupted clients in async FL

Challenges to tackle

Concurrency quota utilization

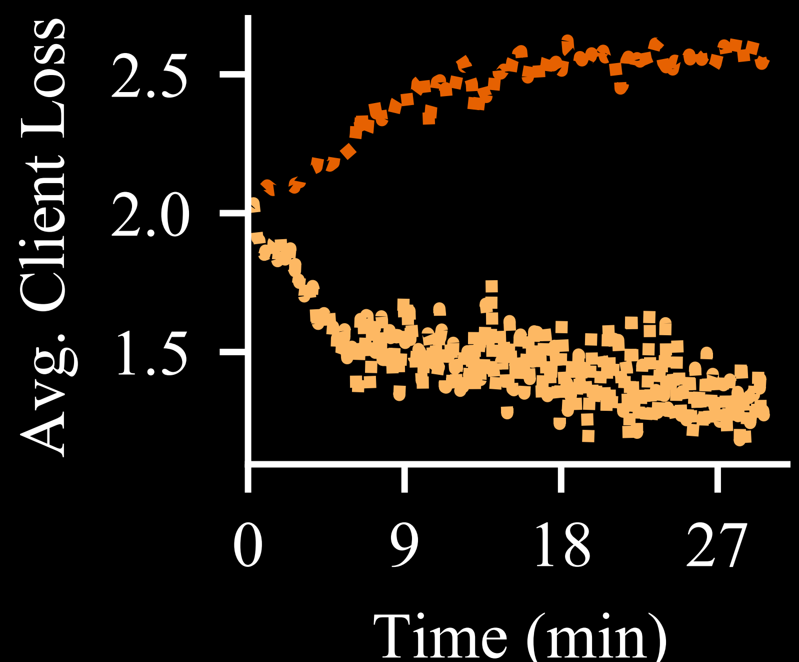
#1: Useful clients tends to have large gradients/losses

Intuitions

Participant Selection

Prefer clients with high

Aggregate training loss[1]



Detect corrupted clients by clustering

Challenges to tackle

Concurrency quota utilization

Robustness against outliers

Intuitions

#1: Useful clients tends to have large gradients/losses

#2: Clients with corrupted data have outlier losses

Participant Selection

Prefer clients with high

$$\frac{\text{Aggregate training loss}^{[1]}}{\text{Tendency for being stale}}$$

Detect corrupted clients by clustering

Challenges to tackle

- Concurrency quota utilization
- Stale update avoidance
- Robustness against outliers

Intuitions

#1: Useful clients tends to have large gradients/losses

#2: Clients with corrupted data have outlier losses

#3: Reduce stale computation in the first place

Participant Selection

Prefer clients with high

$$\frac{\text{Aggregate training loss}^{[1]}}{\text{Tendency for being stale}}$$

Use the moving average to estimate

Detect corrupted clients by clustering

Challenges to tackle

- Concurrency quota utilization
- Stale update avoidance
- Robustness against outliers

Intuitions

- #1: Useful clients tends to have large gradients/losses
- #2: Clients with corrupted data have outlier losses
- #3: Reduce stale computation in the first place
- #4: Clients' staleness evolves steadily over time

Aggregation Pace Control

Convergence guarantee

↑

Goal: bound the staleness for all clients during the entire training

↑

Subproblem: bound the staleness growth for all clients in a certain time period

Challenges to tackle

Stale Update Avoidance

Intuitions

#1: Bounded staleness guarantees progress in each aggregation

Aggregation Pace Control

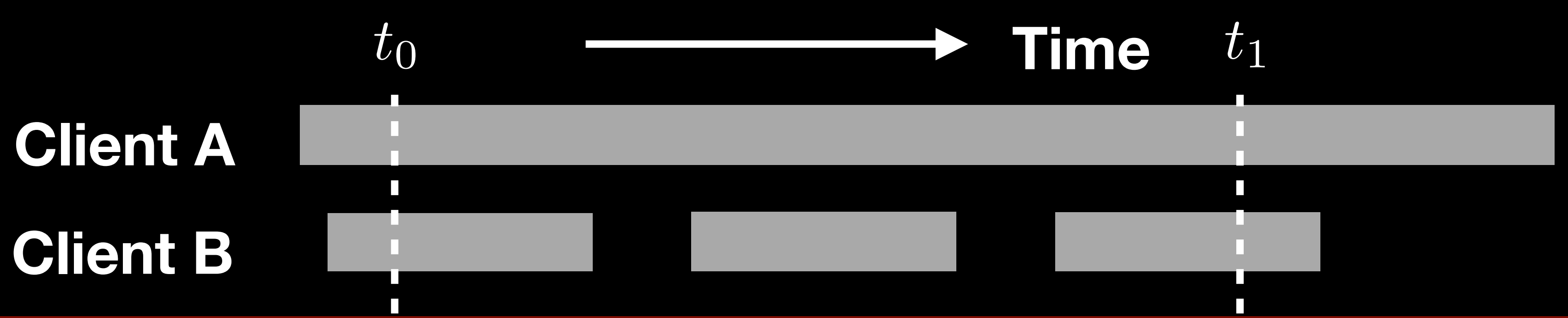
Convergence guarantee

↑

Goal: bound the **staleness** for all clients during **the entire training**

↑

Subproblem: bound the **staleness growth** for all clients in **a certain time period**



Challenges to tackle

Stale Update Avoidance

Intuitions

#1: Bounded staleness guarantees progress in each aggregation

#2: Bounding the staleness growth for a client in a time period guarantees the same for other faster clients

Aggregation Pace Control

Convergence guarantee

↑

Goal: bound the staleness for all clients during the entire training

↑

Subproblem: bound the staleness growth for all clients in a certain time period

↑

Our algorithm: adjust the aggregation pace to the currently running slowest client's speed

Challenges to tackle

Stale Update Avoidance

Intuitions

#1: Bounded staleness guarantees progress in each aggregation

#2: Bounding the staleness growth for a client in a time period guarantees the same for other faster clients

Aggregation Pace Control

Convergence guarantee

Goal: bound the staleness for all clients during the entire training

Subproblem: bound the staleness growth for all clients in a certain time period

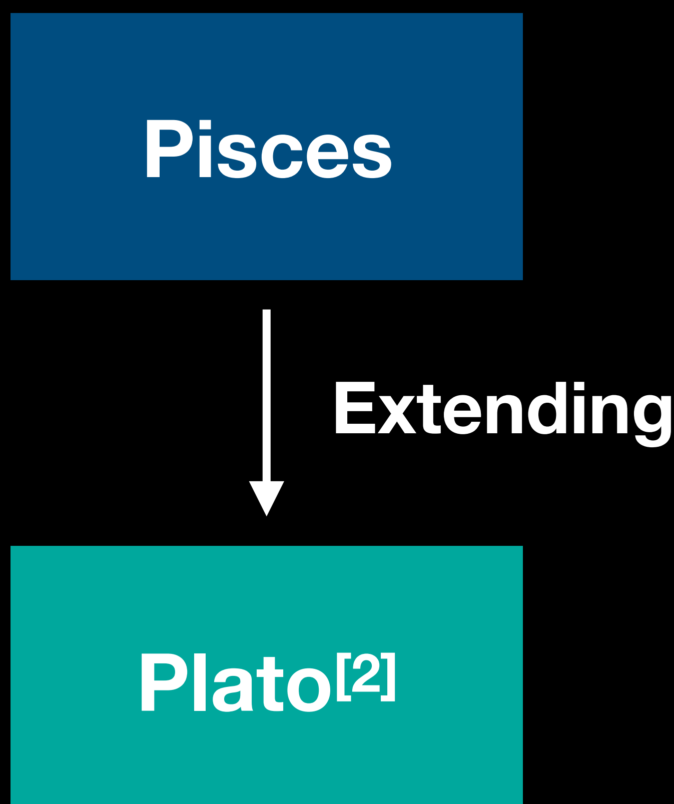
Our algorithm: adjust the aggregation pace to the currently running slowest client's speed

Challenges to tackle

Stale Update Avoidance

Flexibility to environments

Evaluation

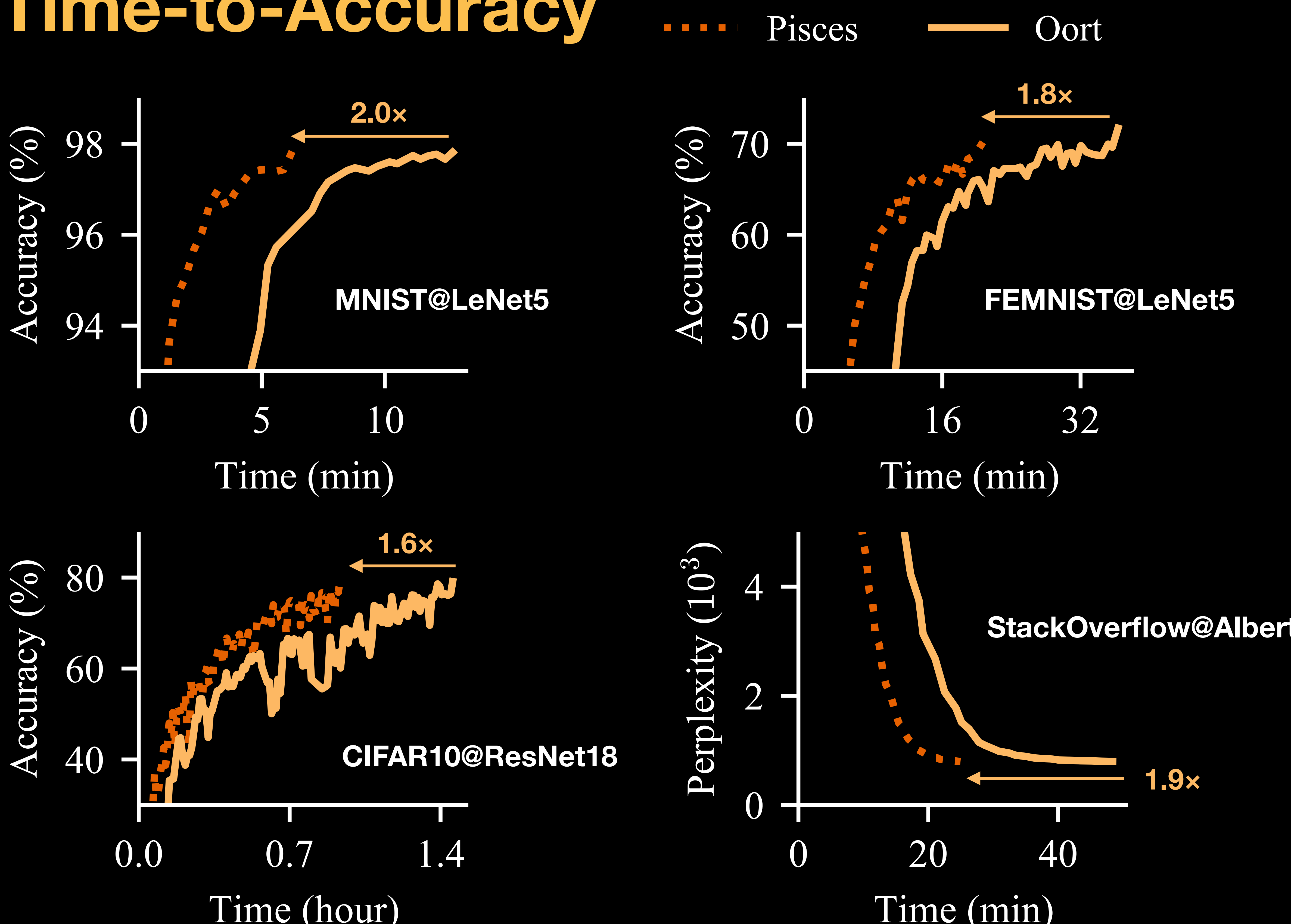


Setup

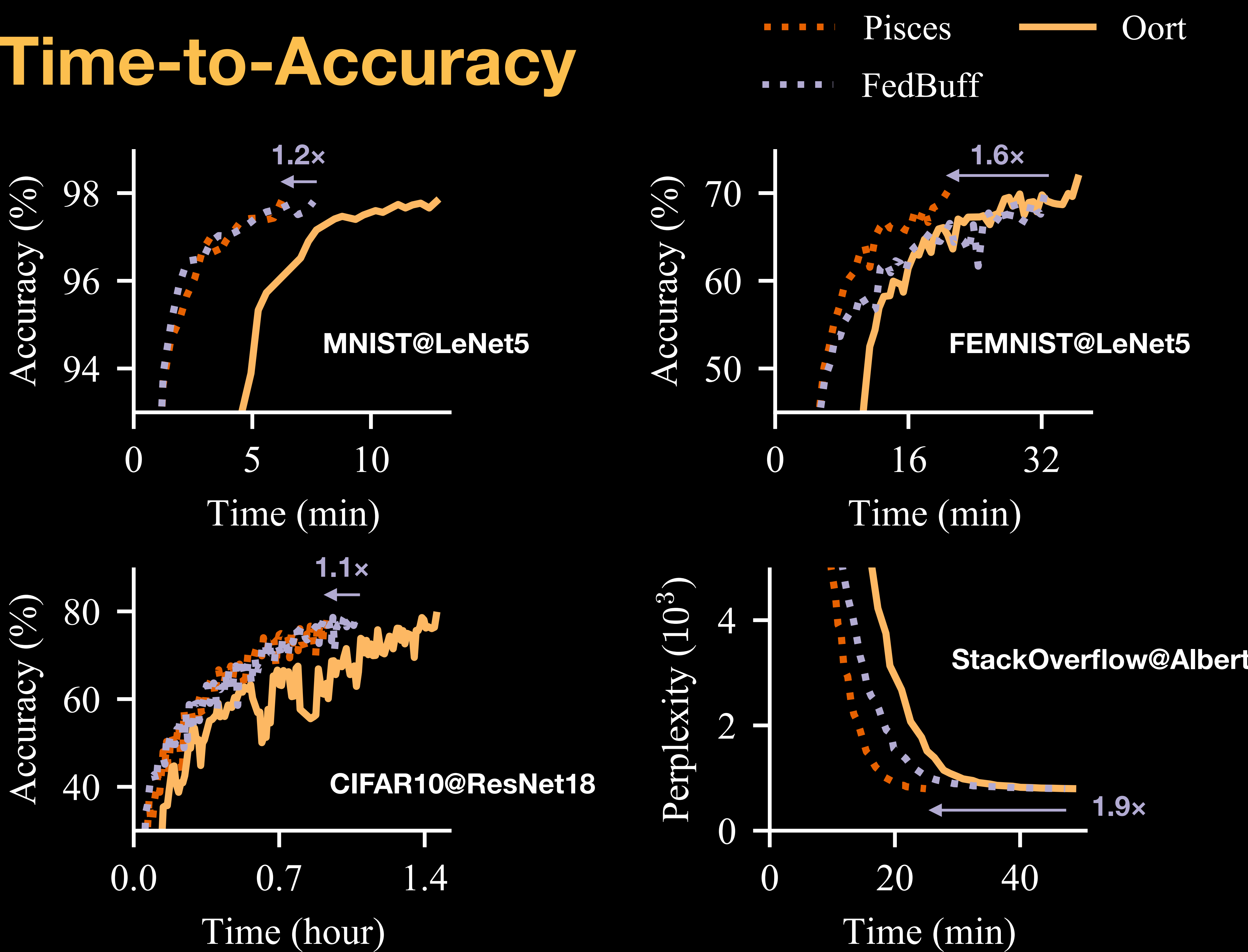
- Testbed w/ 200 clients
  - Concurrency limit is 10%
- Heterogeneity
  - System: Zipf's distribution
  - Data: Realistic or synthetic
- Baselines
  - Oort: SOTA Sync FL
  - FedBuff[3]: SOTA Async FL

[2] Plato GitHub repo: https://github.com/TL-System/plato
[3] Federated learning with buffered asynchronous aggregation, AISTATS'22

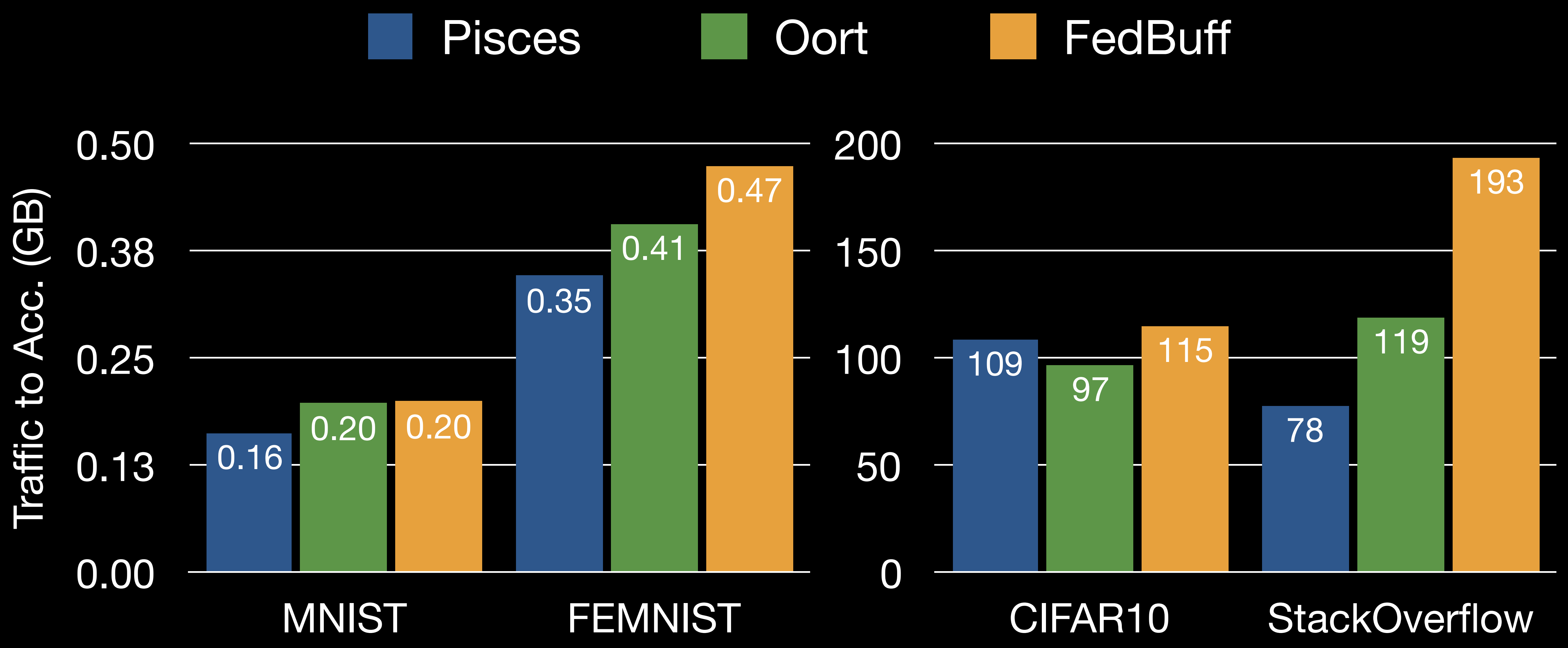
Time-to-Accuracy



Time-to-Accuracy



Traffic-to-Accuracy



Pisces

https://github.com/SamuelGong/Pisces

An async FL framework for

- Efficiency
- Robustness
- Flexibility