# Towards *Private* and *Efficient* Cross-Device **Federated Learning**

**PhD Thesis Defense** by **Zhifeng Jiang**

27 May 2024

香港科技大學
THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Advisor: **Wei Wang**

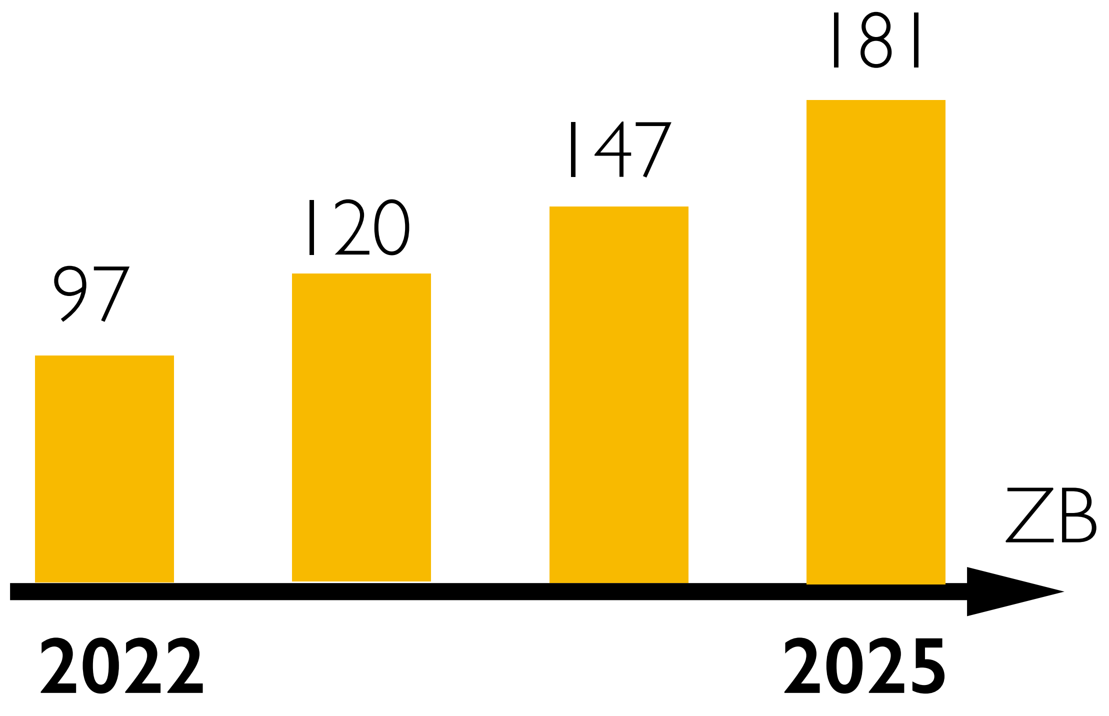
Chairperson: **Yi-Min Lin** (SOSC)

Committee: **Mo Li**, **Shuai Wang**, **Jun Zhang** (ECE), **Cong Wang** (CityU)

# Growth of edge computing

Edge devices generate
massive **data**

**Generated data volume**[1]



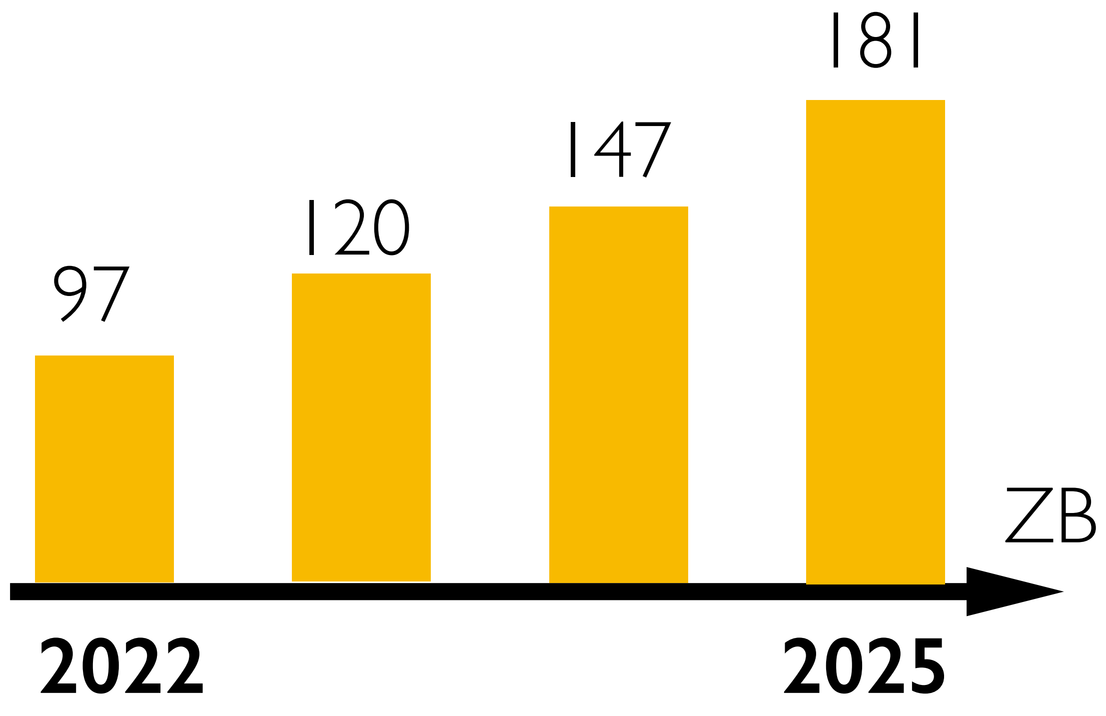

[1]Exploding topics blog, "Amount of Data Created Daily (2024)", 2023

# Growth of edge computing

Edge devices generate massive **data**

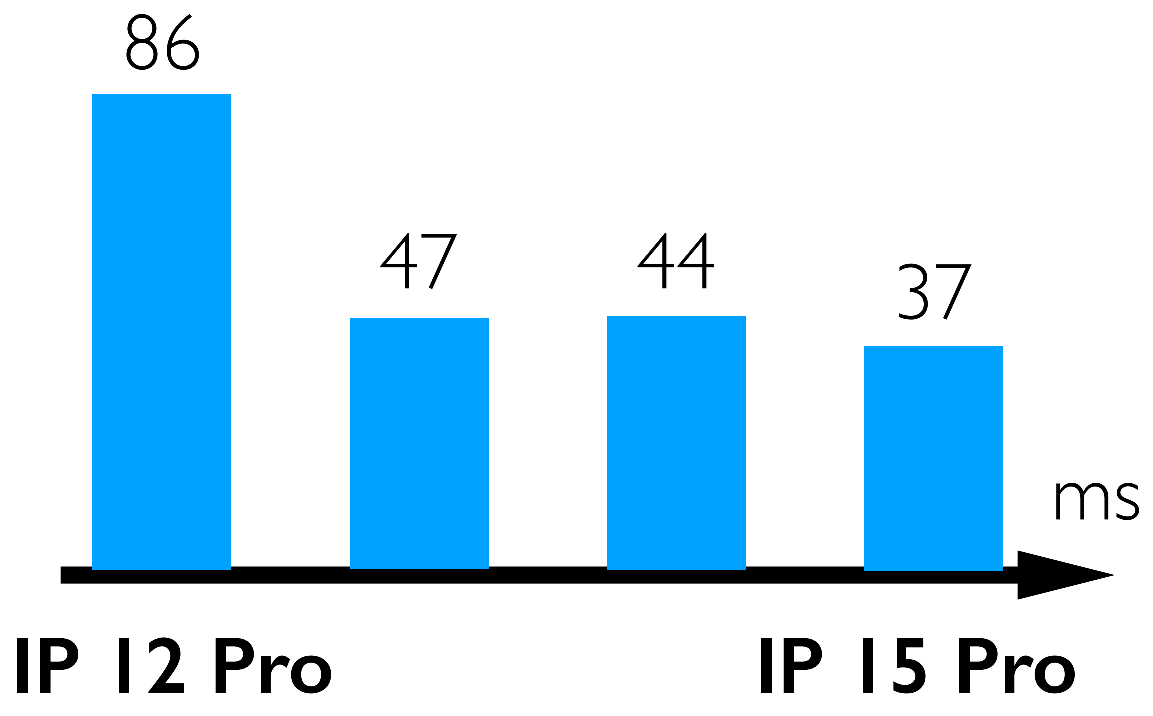
Increasing **resource** on edge devices

**Generated data volume**[1]




+

**Inference time**[2]

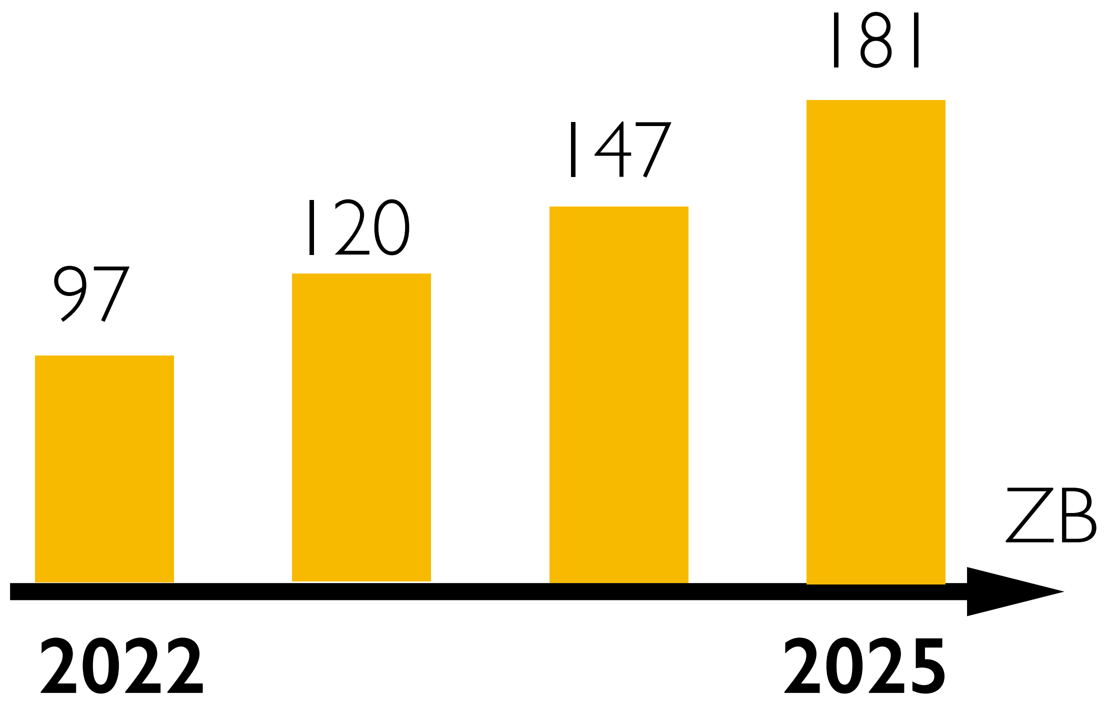



[1]Exploding topics blog, "Amount of Data Created Daily (2024)", 2023
[2]Photoroom blog, "Core ML performance benchmark iPhone 15 (2023)", 2023

# Growth of edge computing

Edge devices generate
massive **data**

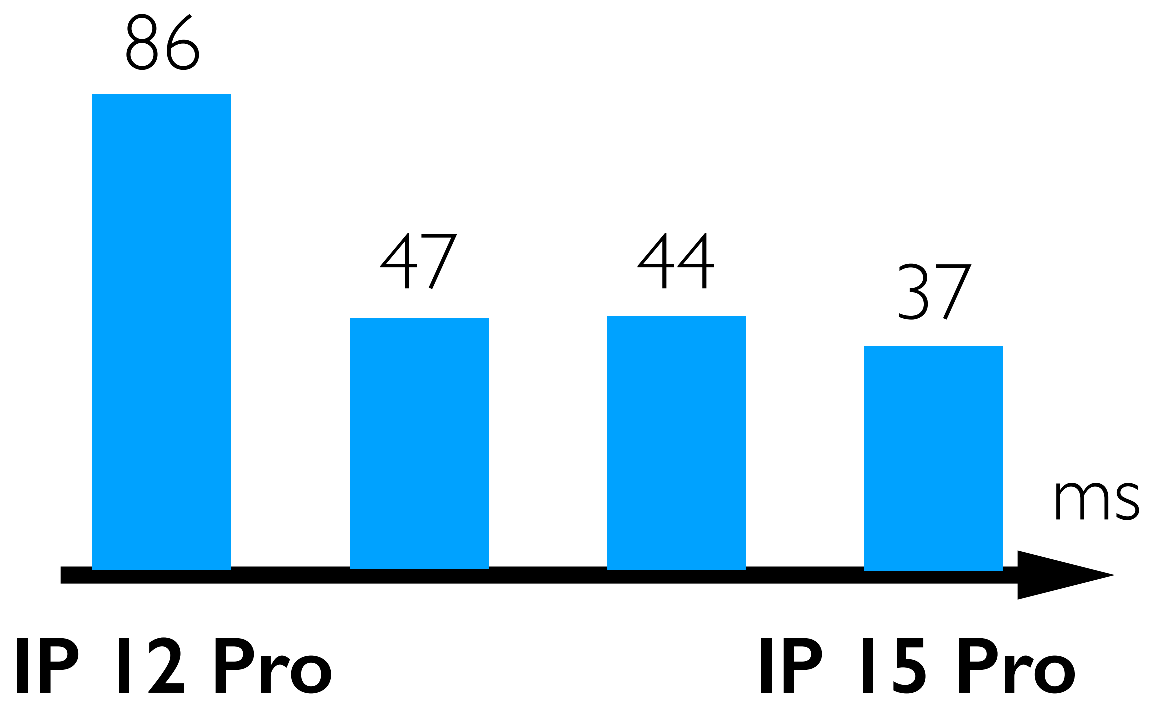
**Generated data volume**[1]



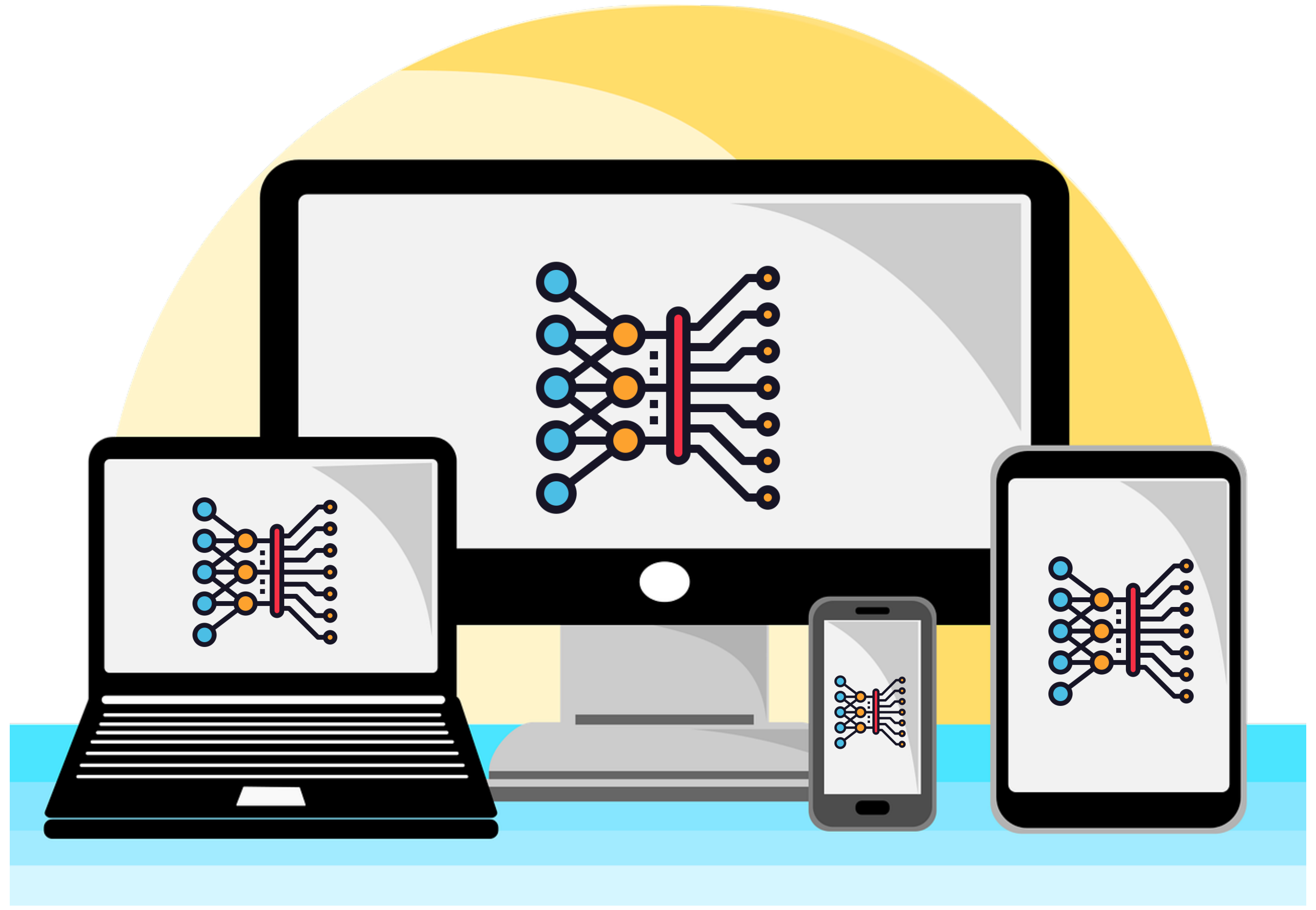

+

Increasing **resource** on
edge devices

**Inference time**[2]




**machine learning**
driven to the edge



[1] Exploding topics blog, "Amount of Data Created Daily (2024)", 2023
[2] Photoroom blog, "Core ML performance benchmark iPhone 15 (2023)", 2023

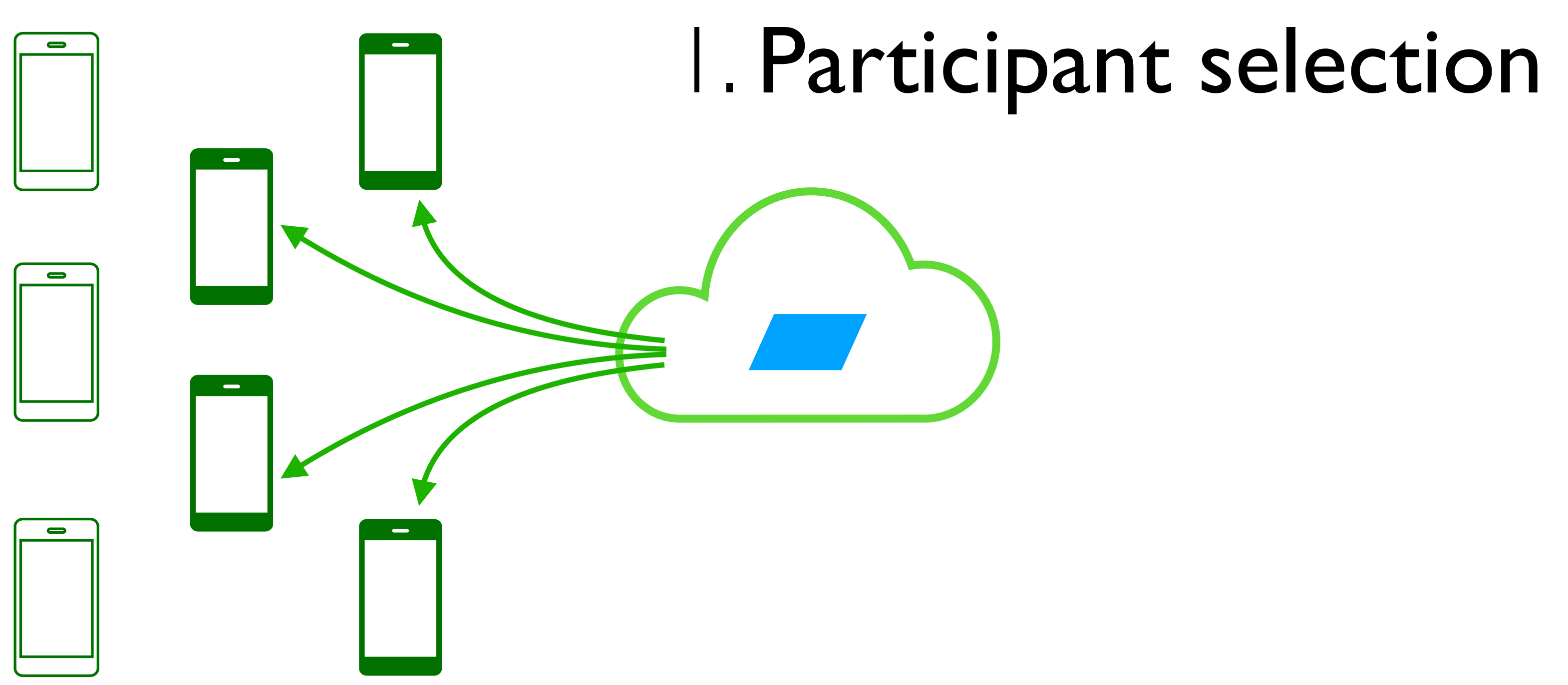
# Private learning on the edge

# Private learning on the edge

| | |
|---|---|
| **Privacy-Enhancing Technique** | Federated Learning[1] |
| **Privacy Guarantee** | Data kept on premises |

[1] McMahan et al. "Communication-Efficient Learning of Deep Networks from Decentralized Data", In AISTATS '17

# Private learning on the edge

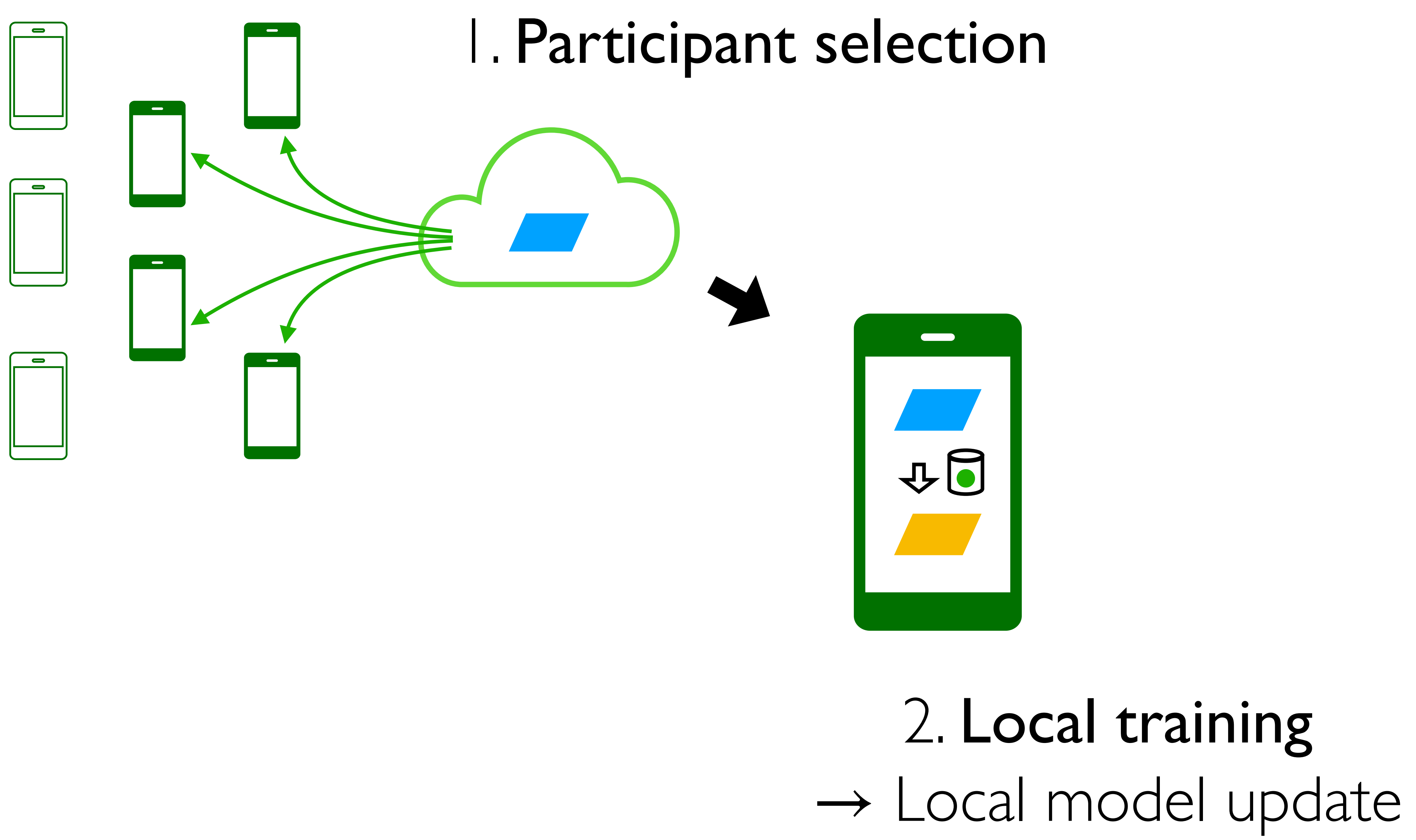



| | |
|---|---|
| **Privacy-Enhancing Technique** | Federated Learning[1] |
| **Privacy Guarantee** | Data kept on premises |

[1] McMahan et al. "Communication-Efficient Learning of Deep Networks from Decentralized Data", In AISTATS '17

# Private learning on the edge

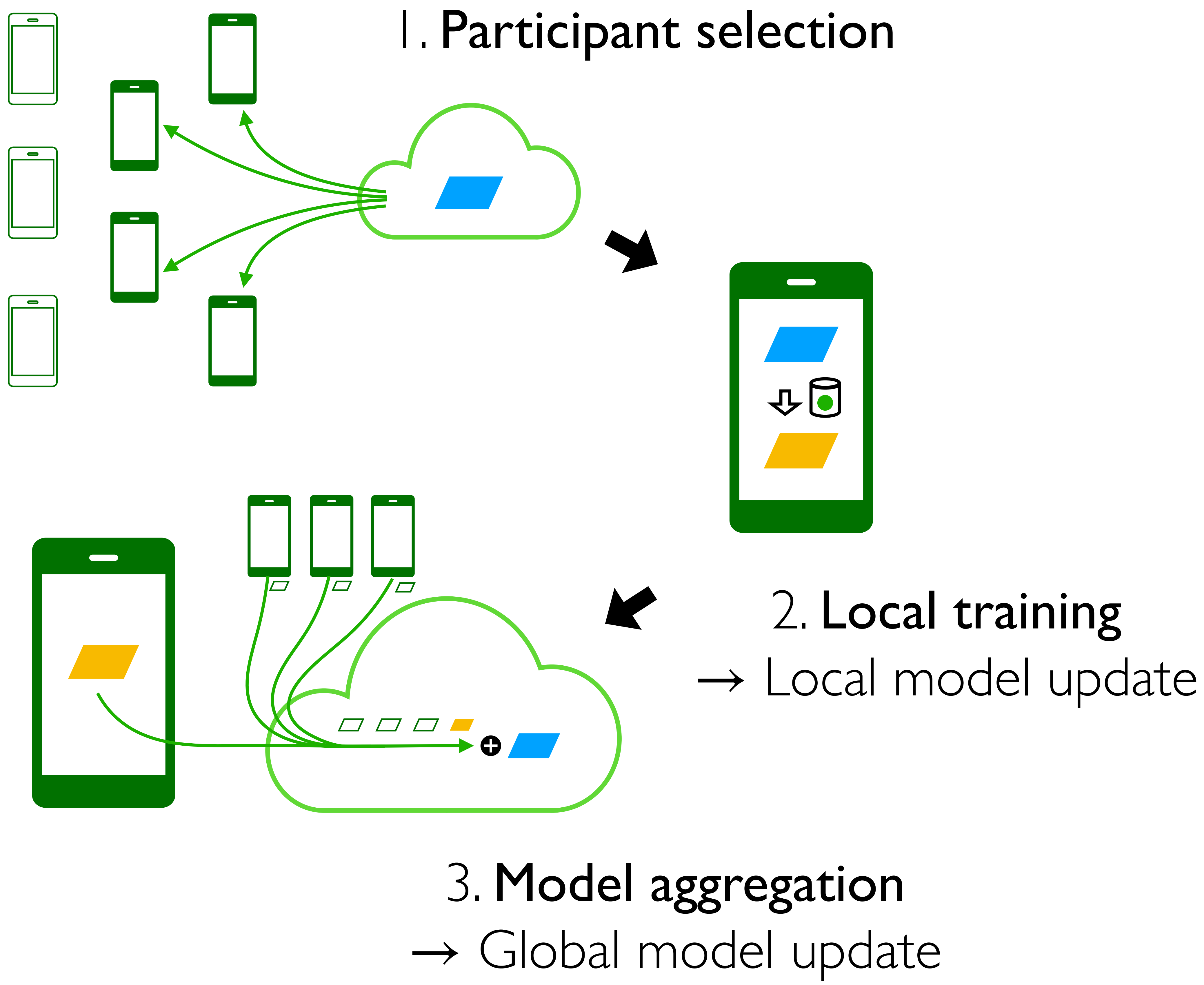
| | |
|---|---|
| **Privacy-Enhancing Technique** | Federated Learning[1] |
| **Privacy Guarantee** | Data kept on premises |




[1] McMahan et al. "Communication-Efficient Learning of Deep Networks from Decentralized Data", In AISTATS '17

# Private learning on the edge

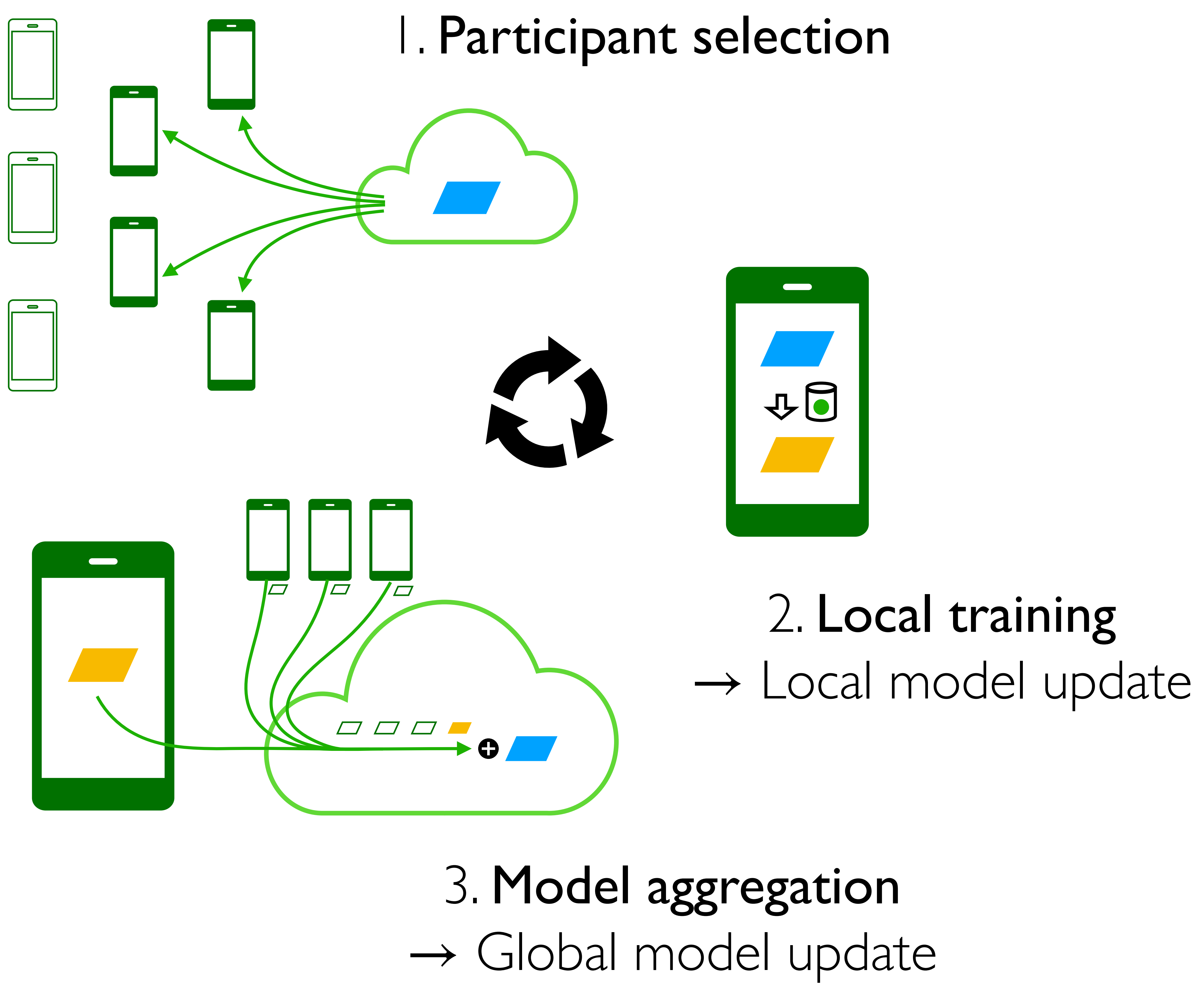
| | |
|---|---|
| **Privacy-Enhancing Technique** | Federated Learning[1] |
| **Privacy Guarantee** | Data kept on premises |




[1] McMahan et al. "Communication-Efficient Learning of Deep Networks from Decentralized Data", In AISTATS '17

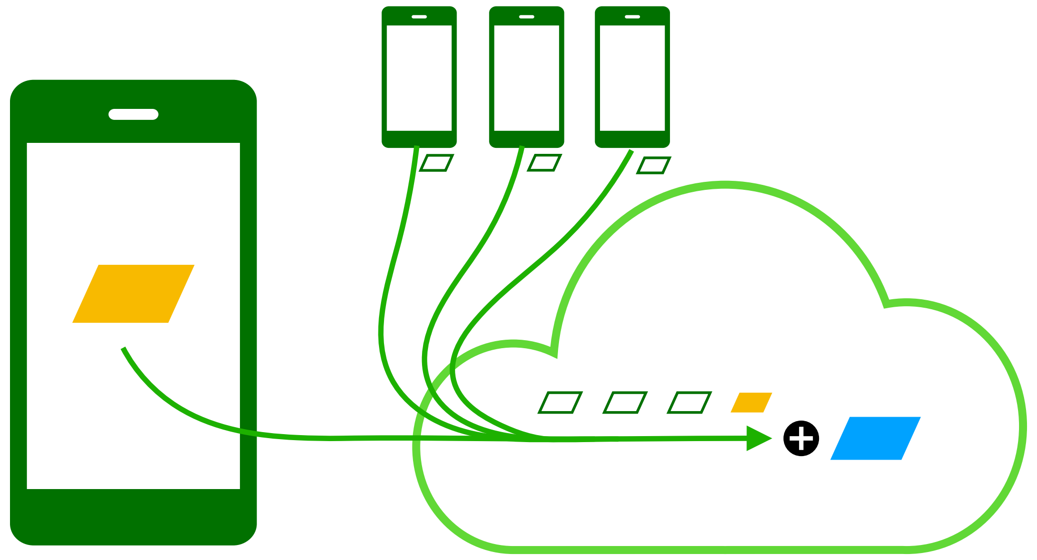
# Private learning on the edge

| | |
|---|---|
| **Privacy-Enhancing Technique** | Federated Learning[1] |
| **Privacy Guarantee** | Data kept on premises |

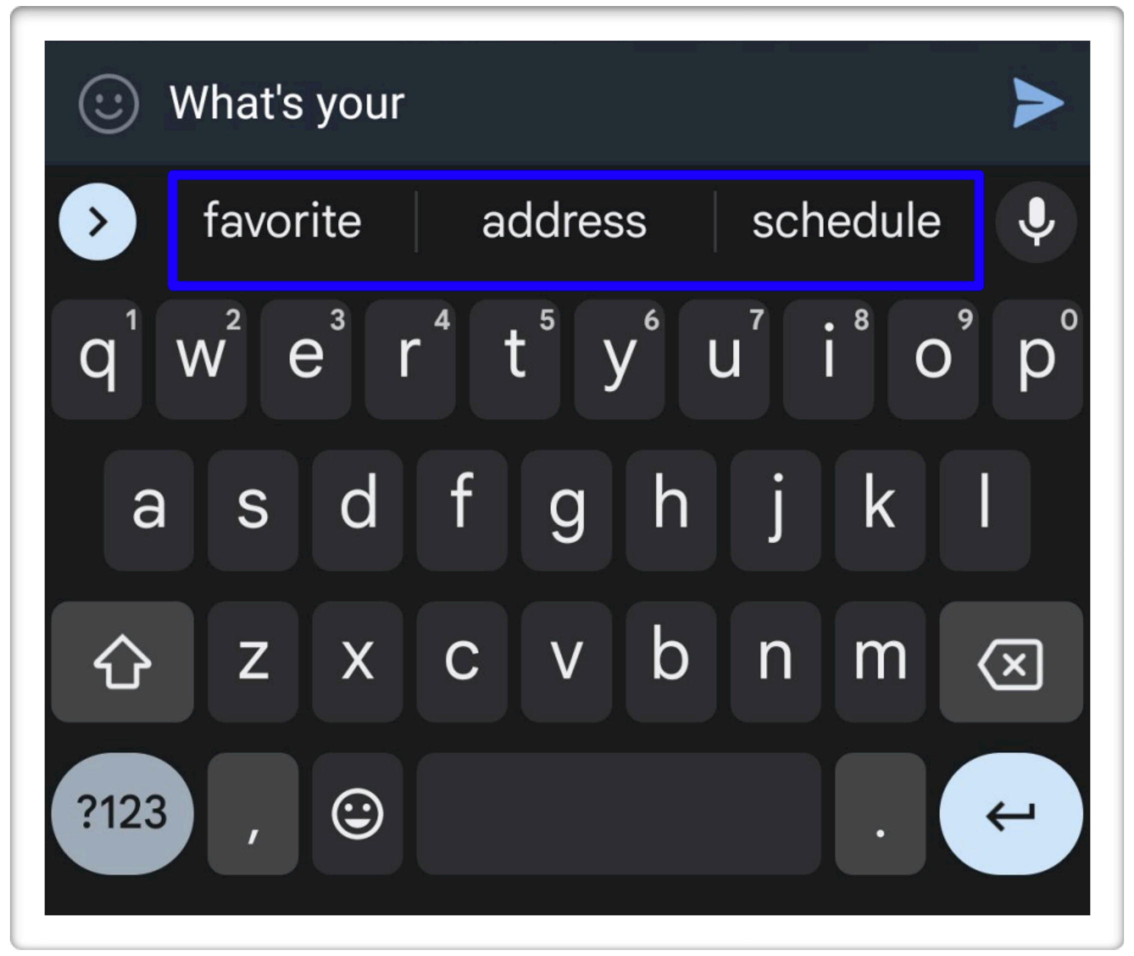



[1] McMahan et al. "Communication-Efficient Learning of Deep Networks from Decentralized Data", In AISTATS '17

# Private learning on the edge



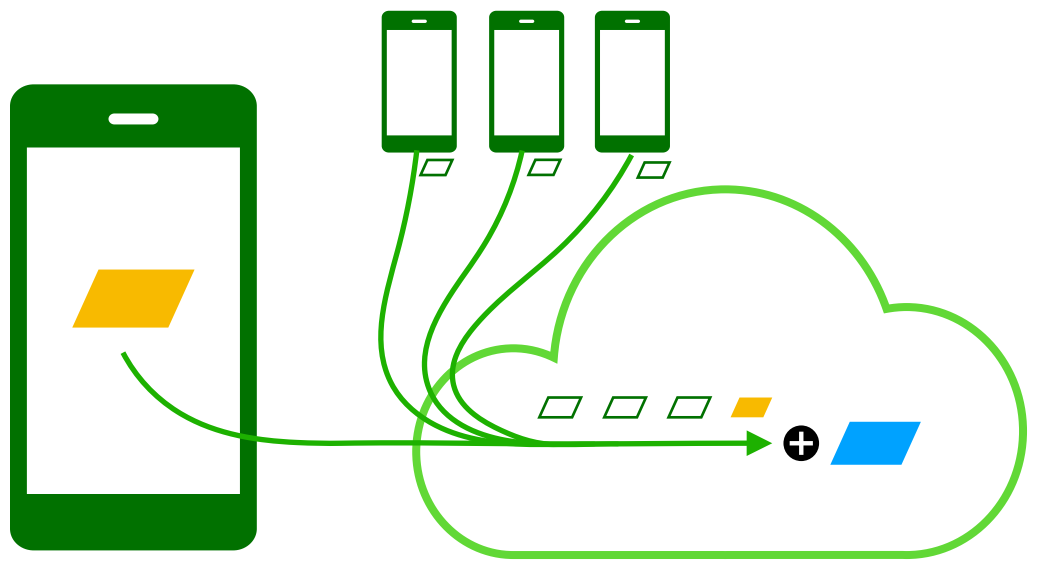
| | |
|---|---|
| **Privacy-Enhancing Technique** | Federated Learning[1] |
| **Privacy Guarantee** | Data kept on premises |



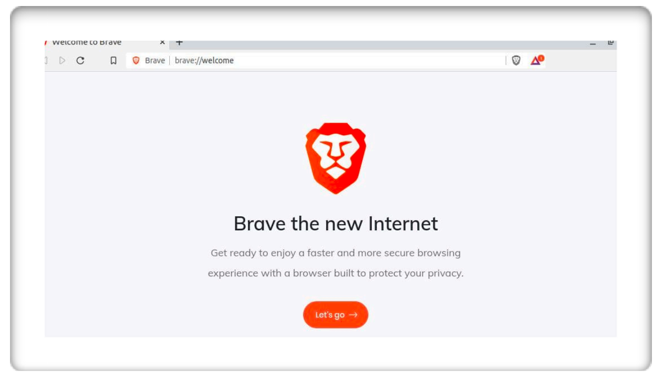

**Real application:**
Google's Keyboard

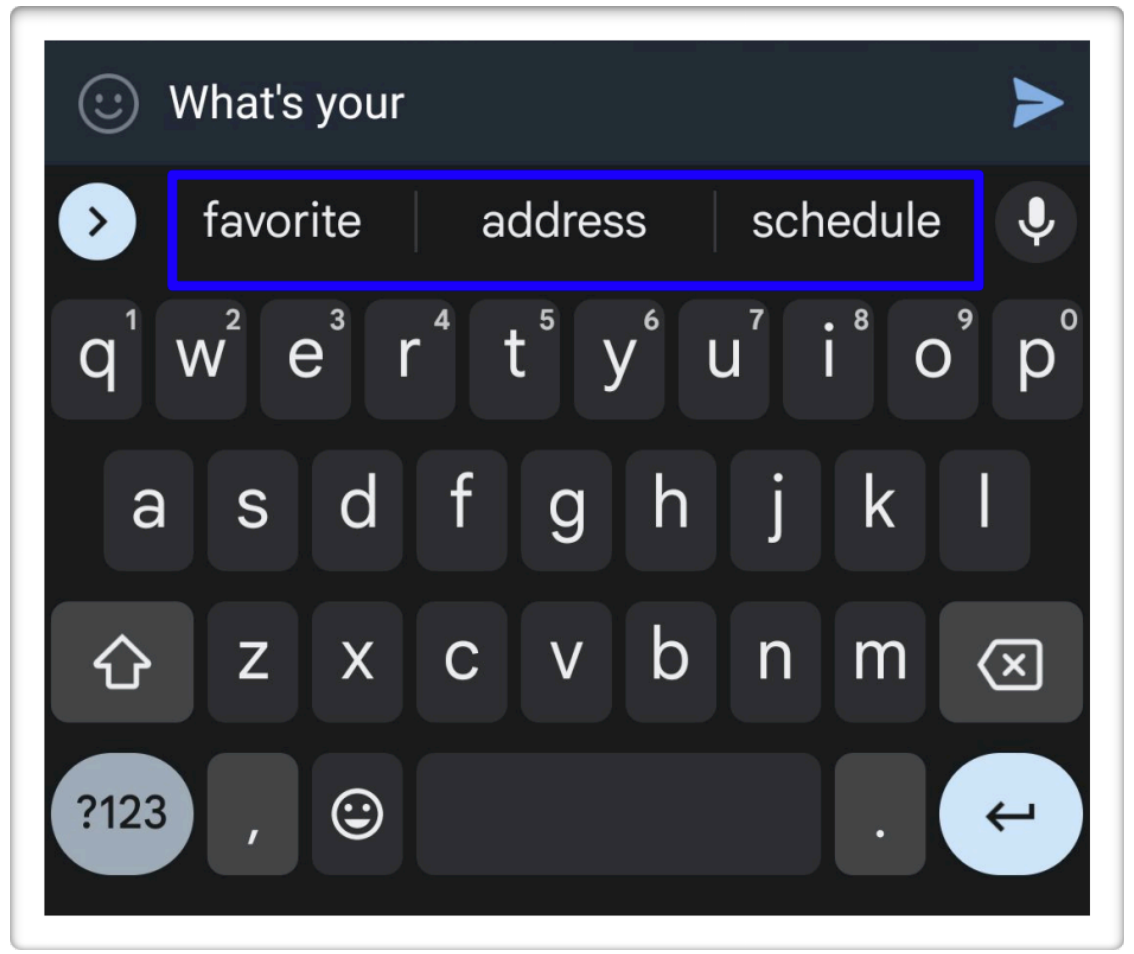
[1] McMahan et al. "Communication-Efficient Learning of Deep Networks from Decentralized Data", In AISTATS '17

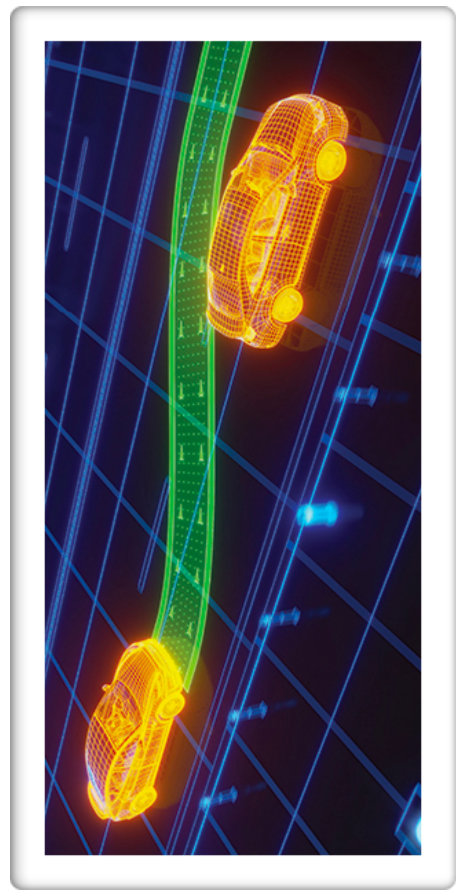
# Private learning on the edge



| | |
|---|---|
| **Privacy-Enhancing Technique** | Federated Learning[1] |
| **Privacy Guarantee** | Data kept on premises |



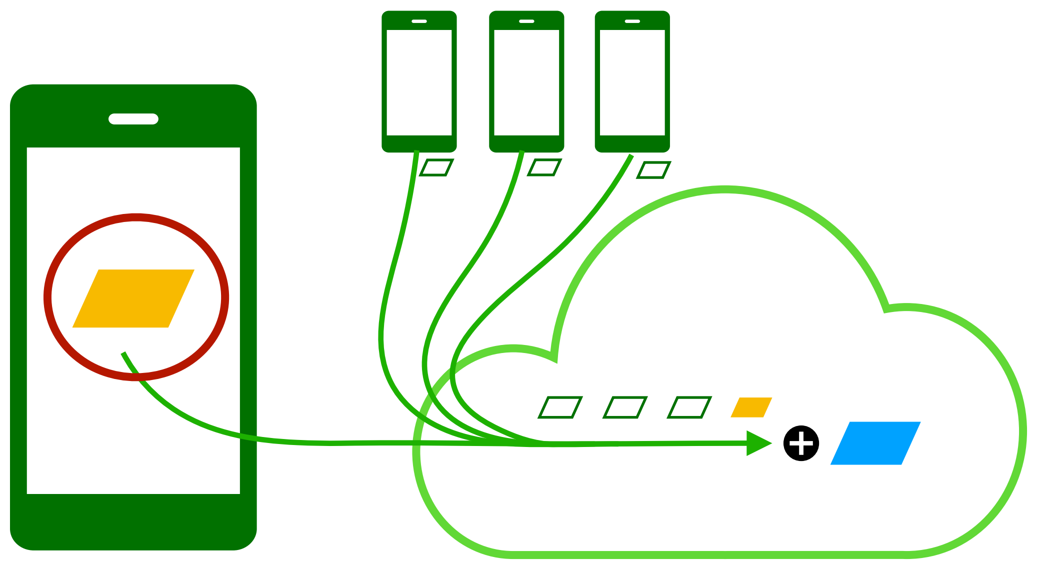



**Real application:**

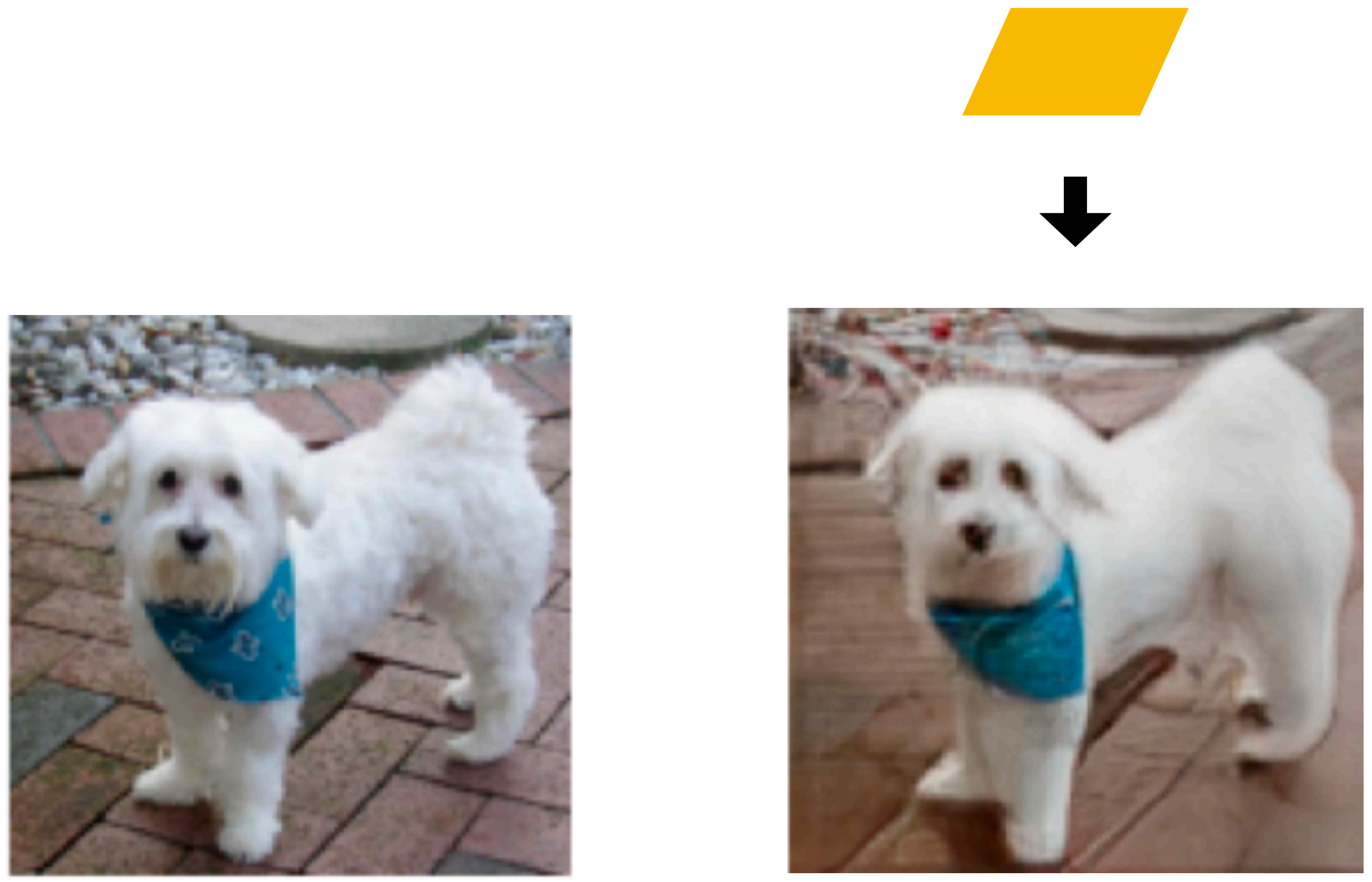
Google's Keyboard, …

[1]McMahan et al. "Communication-Efficient Learning of Deep Networks from Decentralized Data", In AISTATS '17

# Private learning on the edge



| | |
|---|---|
| **Privacy-Enhancing Technique** | Federated Learning[1] |
| **Privacy Guarantee** | Data kept on premises |

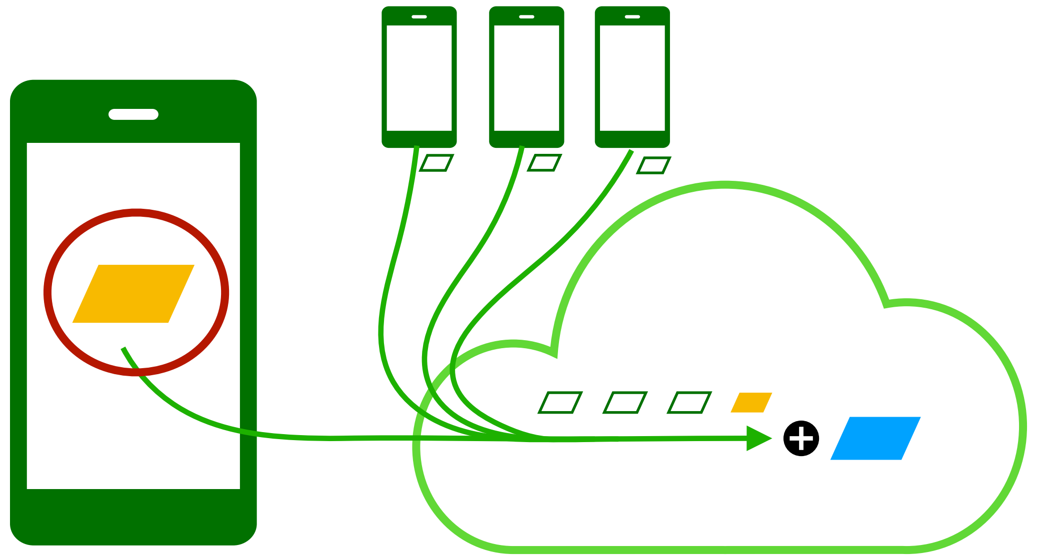


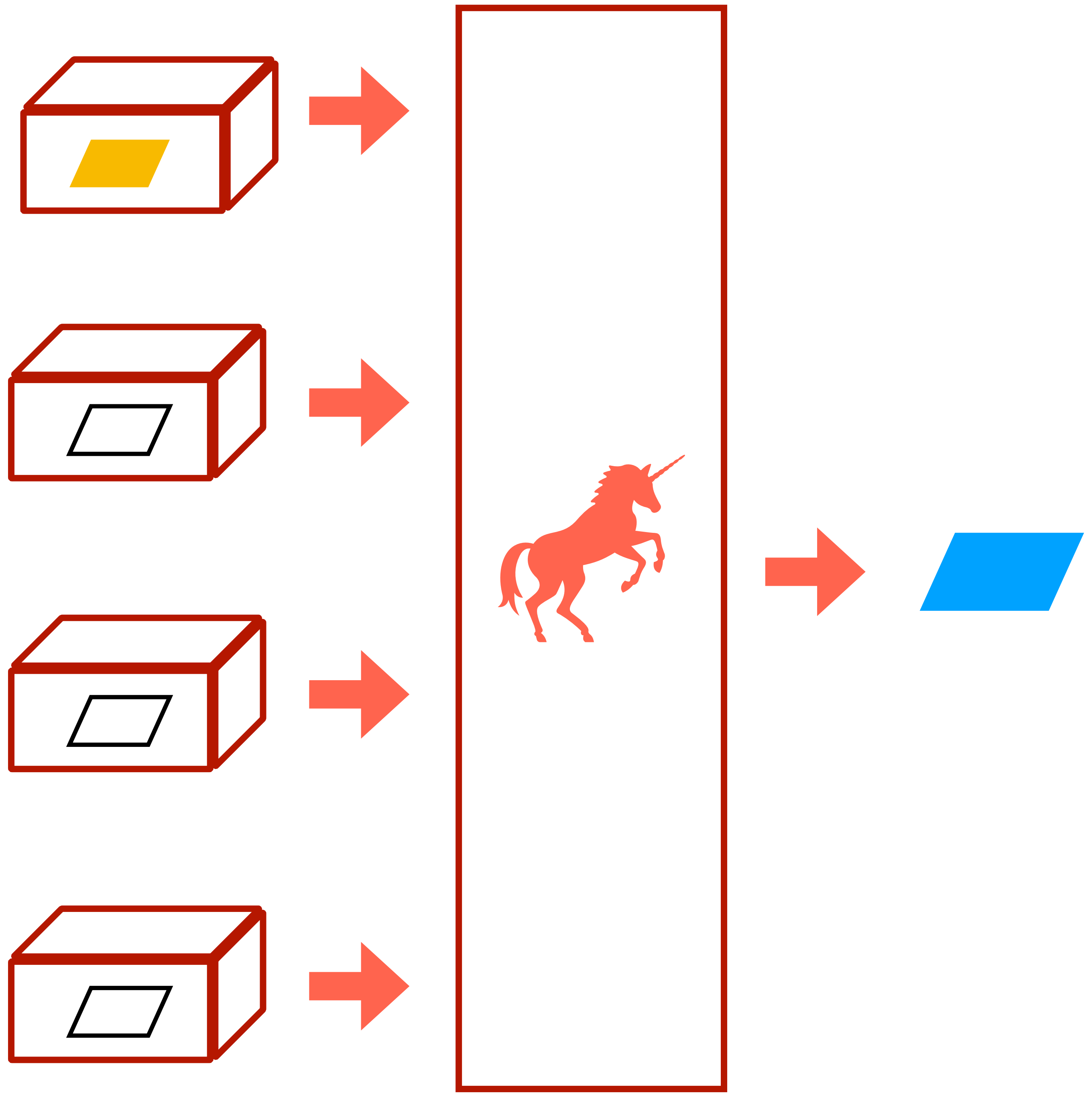
Ground truth | Reconstructed

**Problem**: Data can be reconstructed from **local model updates**[2]

[1]McMahan et al. "Communication-Efficient Learning of Deep Networks from Decentralized Data", In AISTATS '17
[2]Yue et al. "Gradient Obfuscation Gives a False Sense of Security in Federated Learning", In Security '23

# Private learning on the edge

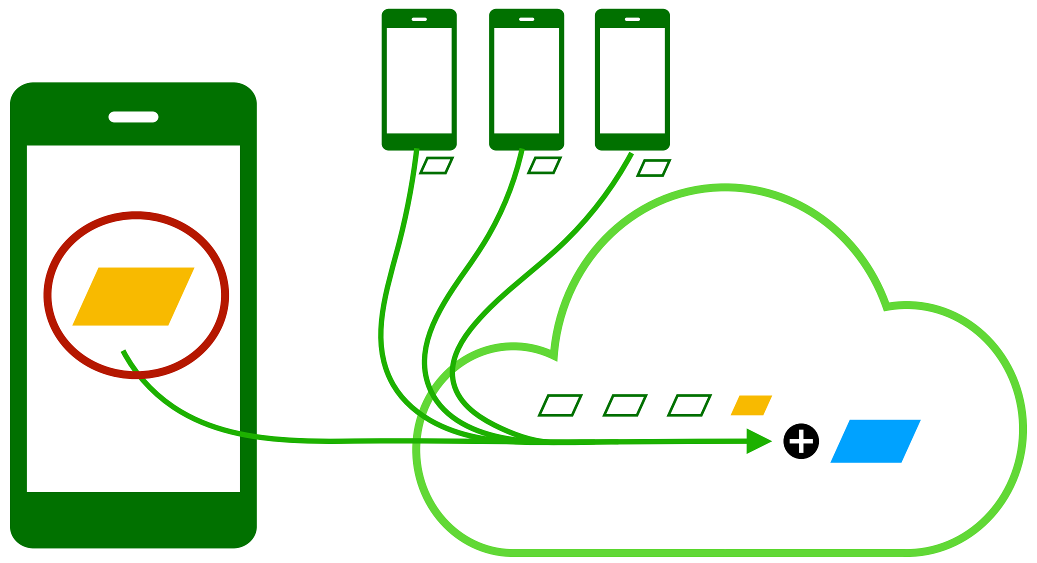


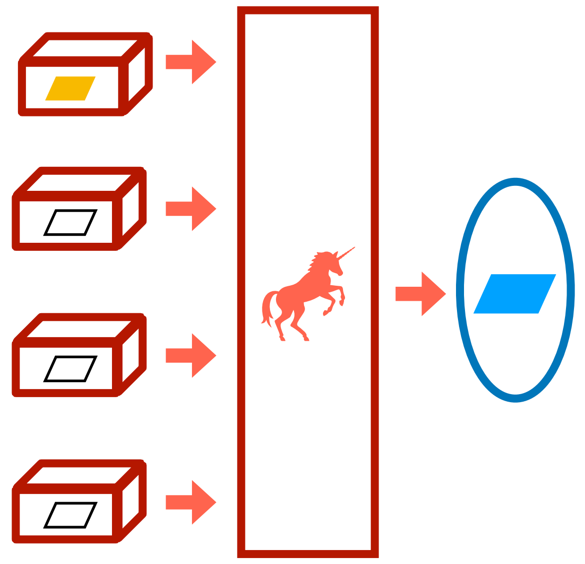


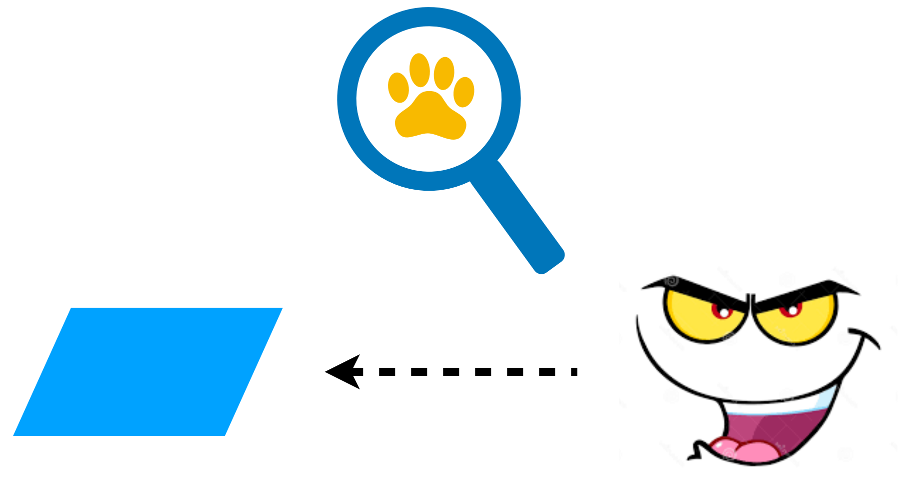
| | | |
|---|---|---|
| **Privacy-Enhancing Technique** | Federated Learning[1] | Secure Aggregation[3,4] |
| **Privacy Guarantee** | Data kept on premises | Local updates unseen |

[1]McMahan et al. "Communication-Efficient Learning of Deep Networks from Decentralized Data", In AISTATS '17
[2]Yue et al. "Gradient Obfuscation Gives a False Sense of Security in Federated Learning", In Security '23
[3]Bonawitz et al. "Practical Secure Aggregation for Privacy-Preserving Machine Learning", In CCS '17
[4]Bell et al. "Secure Single-Server Aggregation with (Poly) Logarithmic Overhead", In CCS '20

# Private learning on the edge







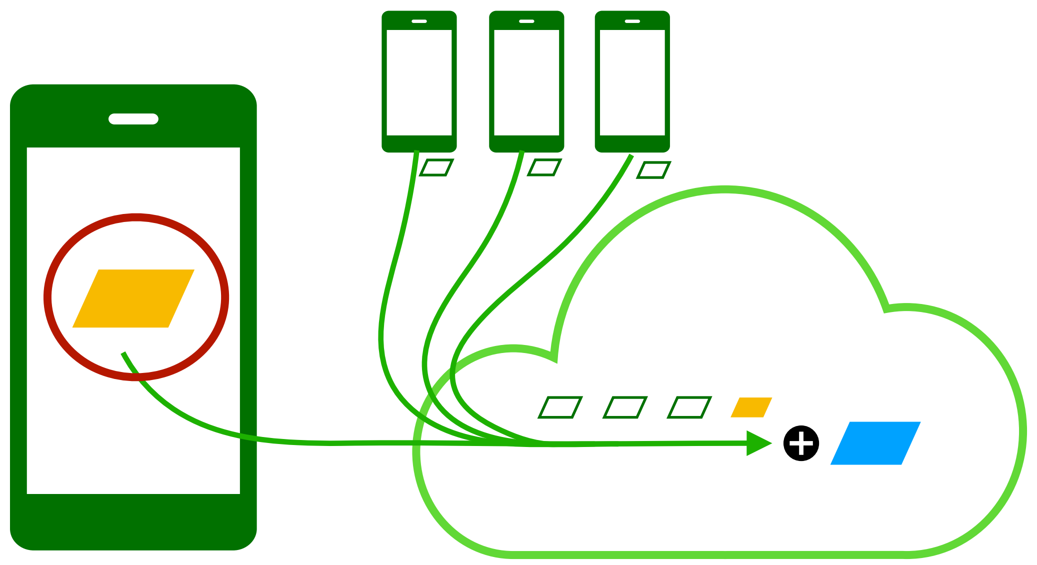
| | | |
|---|---|---|
| **Privacy-Enhancing Technique** | Federated Learning[1] | Secure Aggregation[3,4] |
| **Privacy Guarantee** | Data kept on premises | Local updates unseen |

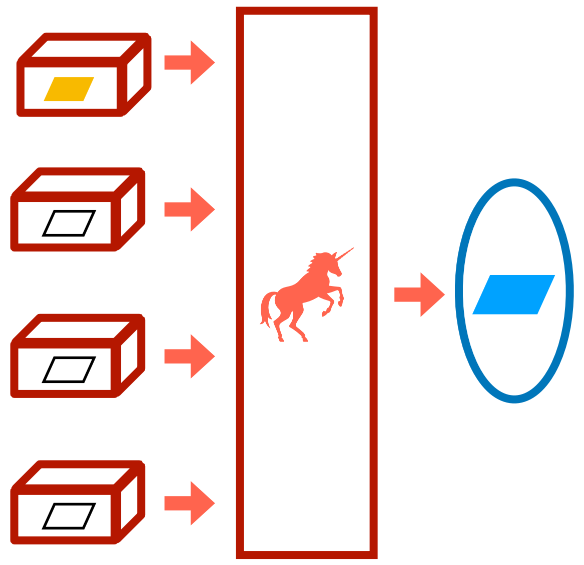
**Problem**: Data still has footprints in **global model update**[5]

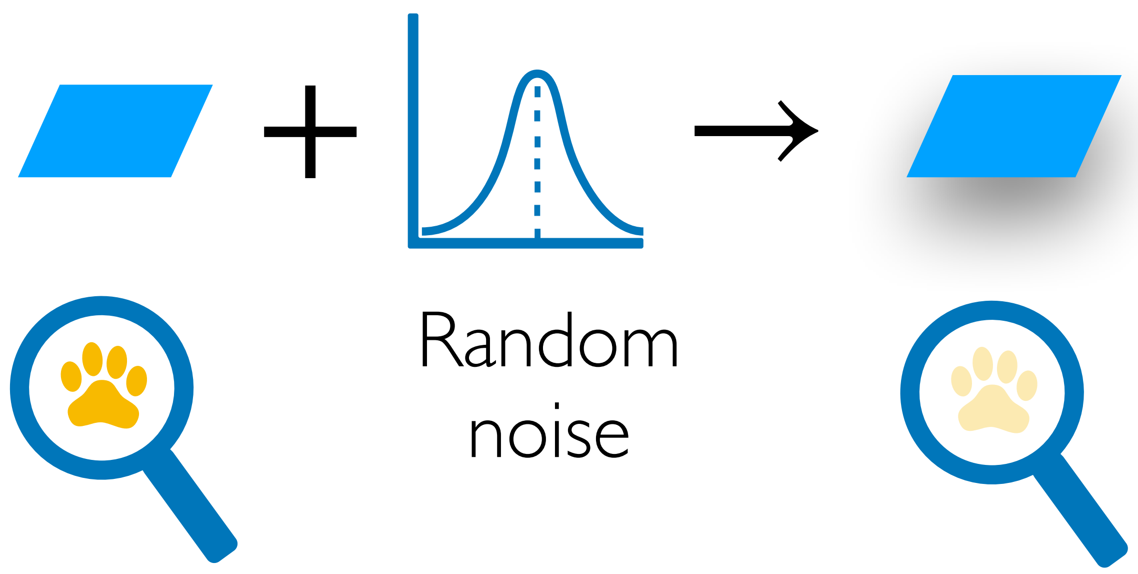
[1]McMahan et al. "Communication-Efficient Learning of Deep Networks from Decentralized Data", In AISTATS '17
[2]Yue et al. "Gradient Obfuscation Gives a False Sense of Security in Federated Learning", In Security '23
[3]Bonawitz et al. "Practical Secure Aggregation for Privacy-Preserving Machine Learning", In CCS '17
[4]Bell et al. "Secure Single-Server Aggregation with (Poly) Logarithmic Overhead", In CCS '20

[5]Nasr et al. "Comprehensive Privacy Analysis of Deep Learning: Passive and Active White-box Inference Attacks against Centralized and Federated Learning", In S&P '19

# Private learning on the edge



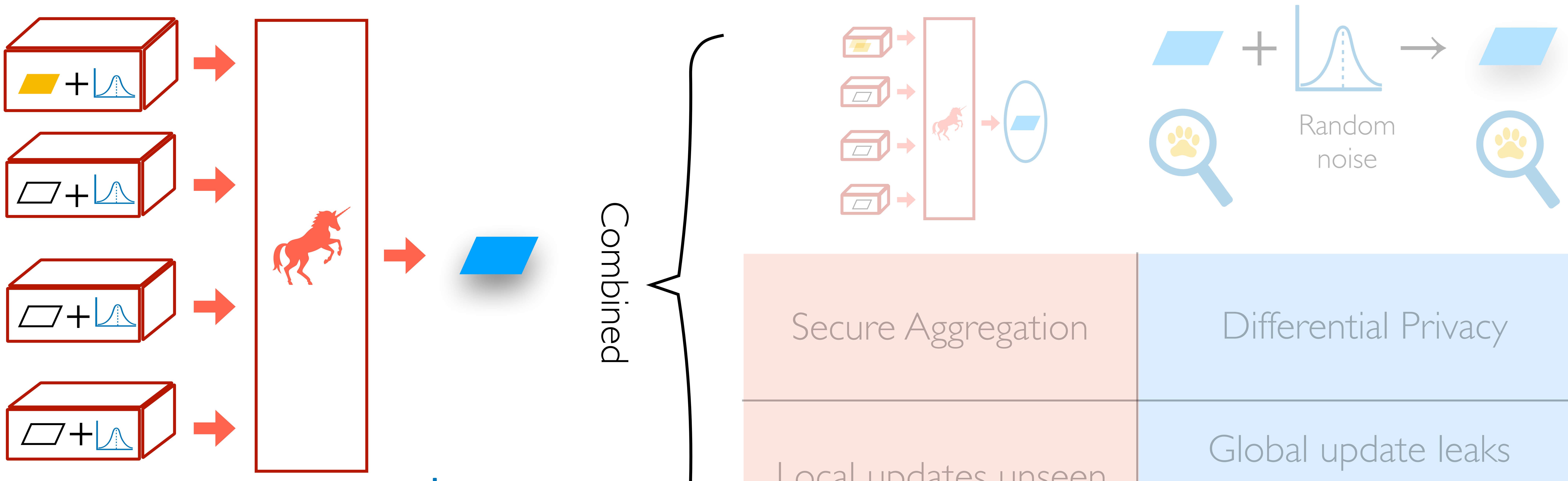





| Privacy-Enhancing Technique | Federated Learning[1] | Secure Aggregation[3,4] | Differential Privacy[6] |
|---|---|---|---|
| Privacy Guarantee | Data kept on premises | Local updates unseen | Global update leaks little about any client |

[1]McMahan et al. "Communication-Efficient Learning of Deep Networks from Decentralized Data", In AISTATS '17
[2]Yue et al. "Gradient Obfuscation Gives a False Sense of Security in Federated Learning", In Security '23
[3]Bonawitz et al. "Practical Secure Aggregation for Privacy-Preserving Machine Learning", In CCS '17
[4]Bell et al. "Secure Single-Server Aggregation with (Poly) Logarithmic Overhead", In CCS '20
[5]Nasr et al. "Comprehensive Privacy Analysis of Deep Learning: Passive and Active White-box Inference Attacks against Centralized and Federated Learning", In S&P '19
[6]Cynthia. "Differential Privacy", 06.

# Private learning on the edge




| Secure Aggregation | Differential Privacy |
| --- | --- |
| Local updates unseen | Global update leaks little about any client |

**Practice**[1,2]: = 1/4

Each client adds an **even share** of the target noise to its local model update

[1]Kairouz et al. "The Distributed Discrete Gaussian Mechanism for Federated Learning with Secure Aggregation", In ICML '21
[2]Agarwal. "The Skellam Mechanism for Differentially Private Federated Learning", In NeurIPS '21

# Private learning on the edge

| | | | |
|---|---|---|---|
| **Privacy-Enhancing Technique** | Federated Learning | Secure Aggregation | Differential Privacy |
| **Privacy Guarantee** | Data kept on premises | Local updates unseen | Global update leaks little about any client |

# My Research

| | | | |
|---|---|---|---|
| | Stragglers bottleneck time | Primitives heavy in comp. and comm. | Client dropout yields insufficient noise |
| **Privacy-Enhancing Technique** | Federated Learning | Secure Aggregation | Differential Privacy |
| **Privacy Guarantee** | Data kept on premises | Local updates unseen | Global update leaks little about any client |

# My Research

Can be a dishonest majority ⟷ Only or mostly works with honest participants

| Stragglers bottleneck time | Primitives heavy in comp. and comm. | Client dropout yields insufficient noise |
|---|---|---|

| | | | |
|---|---|---|---|
| **Privacy-Enhancing Technique** | Federated Learning | Secure Aggregation | Differential Privacy |
| **Privacy Guarantee** | Data kept on premises | Local updates unseen | Global update leaks little about any client |

# My Research

**Privacy**
Worst-case defense…

Can be a dishonest majority x ↔ Only or mostly works with honest participants

| | | | |
|---|---|---|---|
| | Stragglers bottleneck time | Primitives heavy in comp. and comm. | Client dropout yields insufficient noise |
| **Privacy-Enhancing Technique** | Federated Learning | Secure Aggregation | Differential Privacy |
| **Privacy Guarantee** | Data kept on premises | Local updates unseen | Global update leaks little about any client |

# My Research

| | | | |
|---|---|---|---|
| **Privacy**<br>Worst-case defense… | Can be a dishonest majority | x ↔ | Only or mostly works with honest participants |
| **Efficiency**<br>Time-to-accuracy… | Stragglers bottleneck time | Primitives heavy in comp. and comm. | Client dropout yields insufficient noise |
| **Privacy-Enhancing Technique** | Federated Learning | Secure Aggregation | Differential Privacy |
| **Privacy Guarantee** | Data kept on premises | Local updates unseen | Global update leaks little about any client |

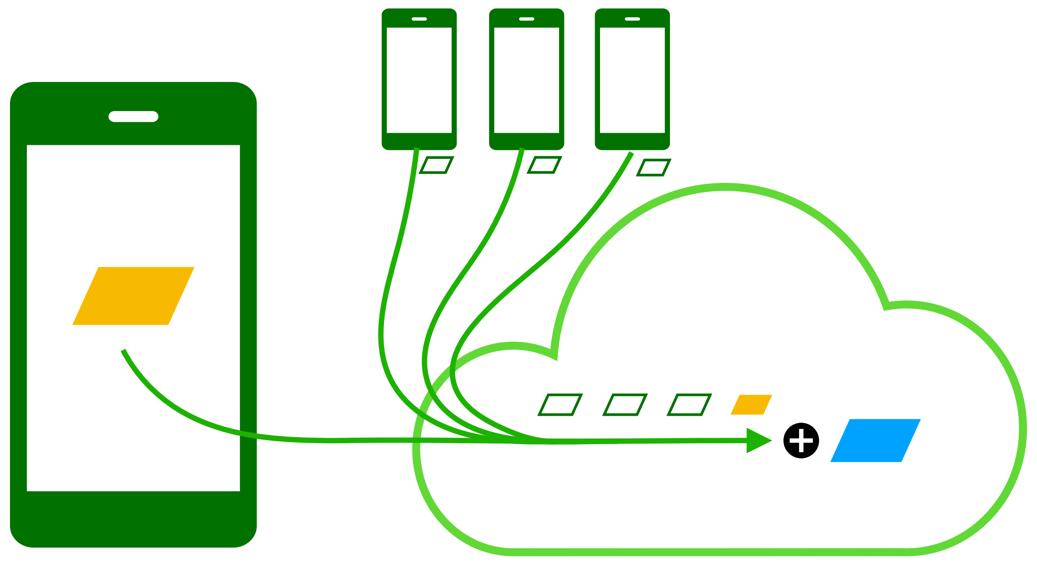
# First work: Pisces[1]

| | | | | |
|---|---|---|---|---|
| **Privacy** Worst-case defense… | Can be a dishonest majority | x ⟷ | Only or mostly works with honest participants | |
| **Efficiency** Time-to-accuracy… | Stragglers bottleneck time | | Primitives heavy in comp. and comm. | Client dropout yields insufficient noise |

| **Privacy-Enhancing Technique** | Federated Learning | Secure Aggregation | Differential Privacy |
|---|---|---|---|
| **Privacy Guarantee** | Data kept on premises | Local updates unseen | Global update leaks little about any client |

[1] Jiang et al. "Pisces: Efficient Federated Learning via Guided Asynchronous Training", In SoCC '22

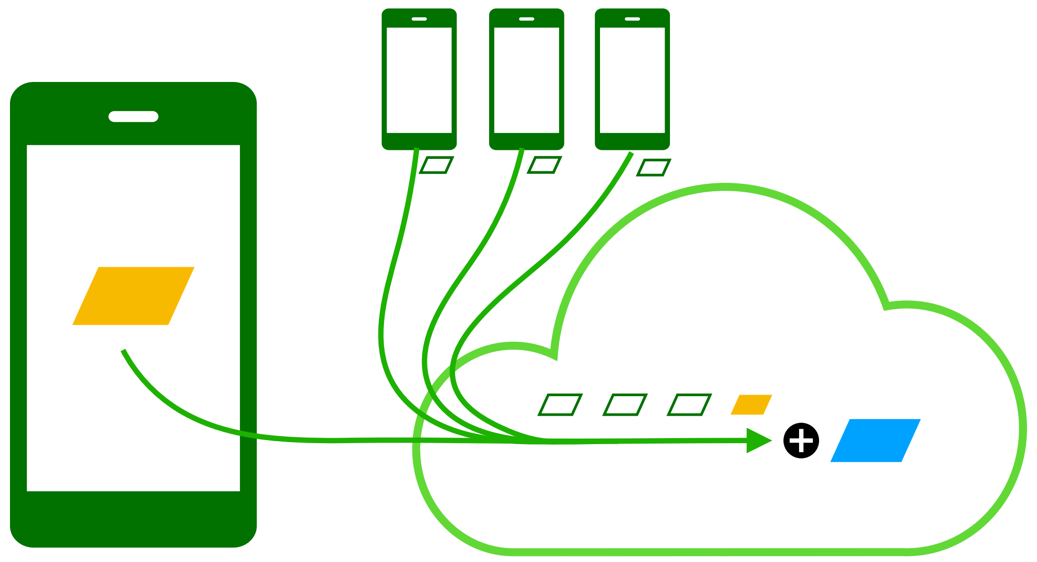
# Need for Pisces



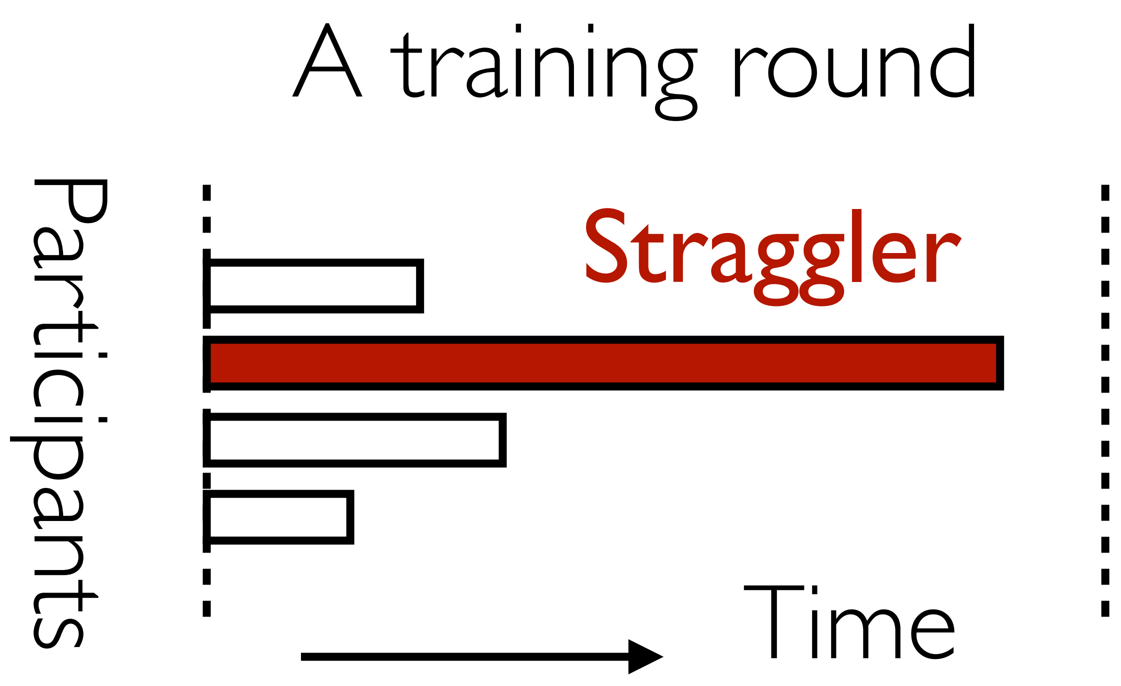
Synchronous

Federated Learning

# Need for Pisces

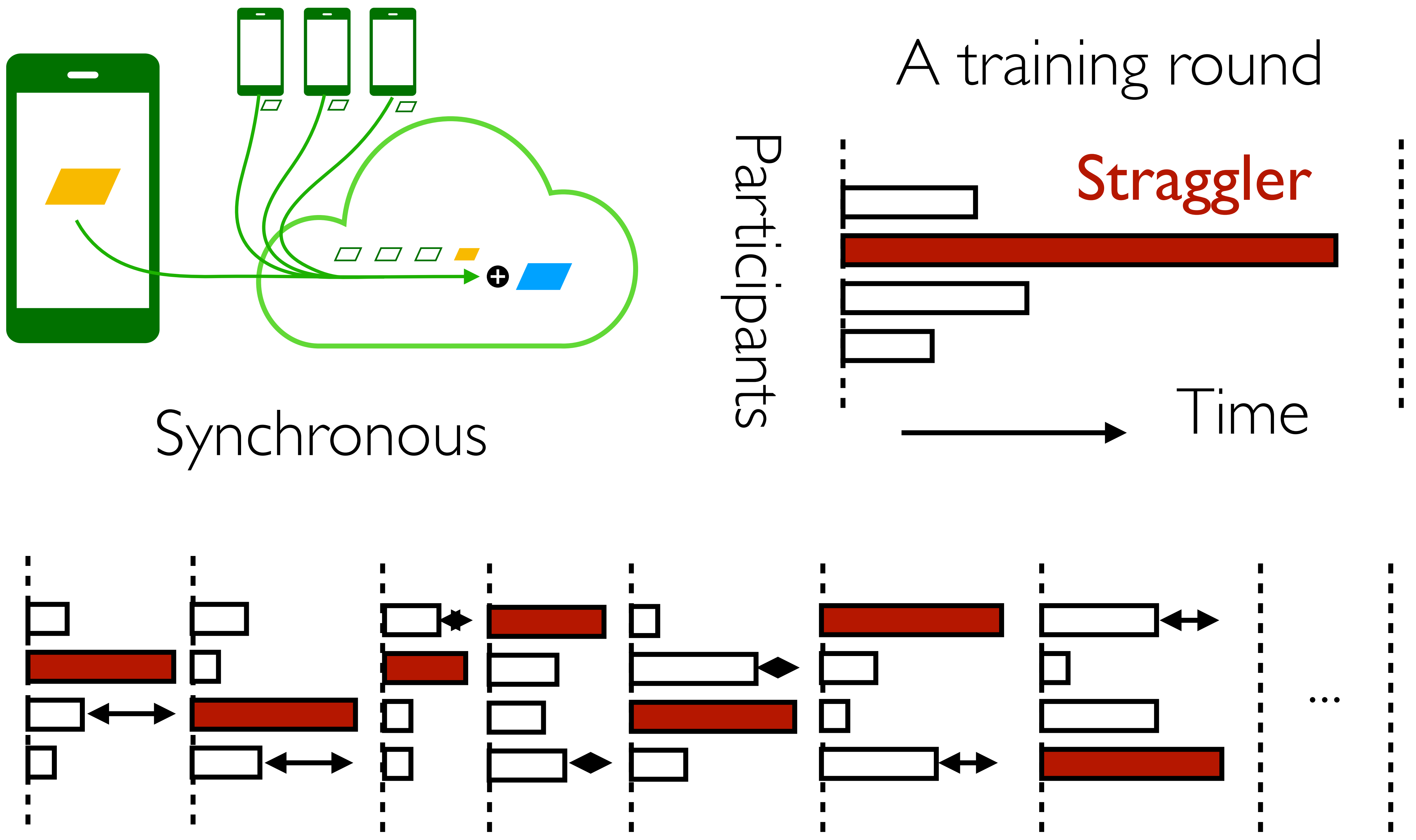


Synchronous



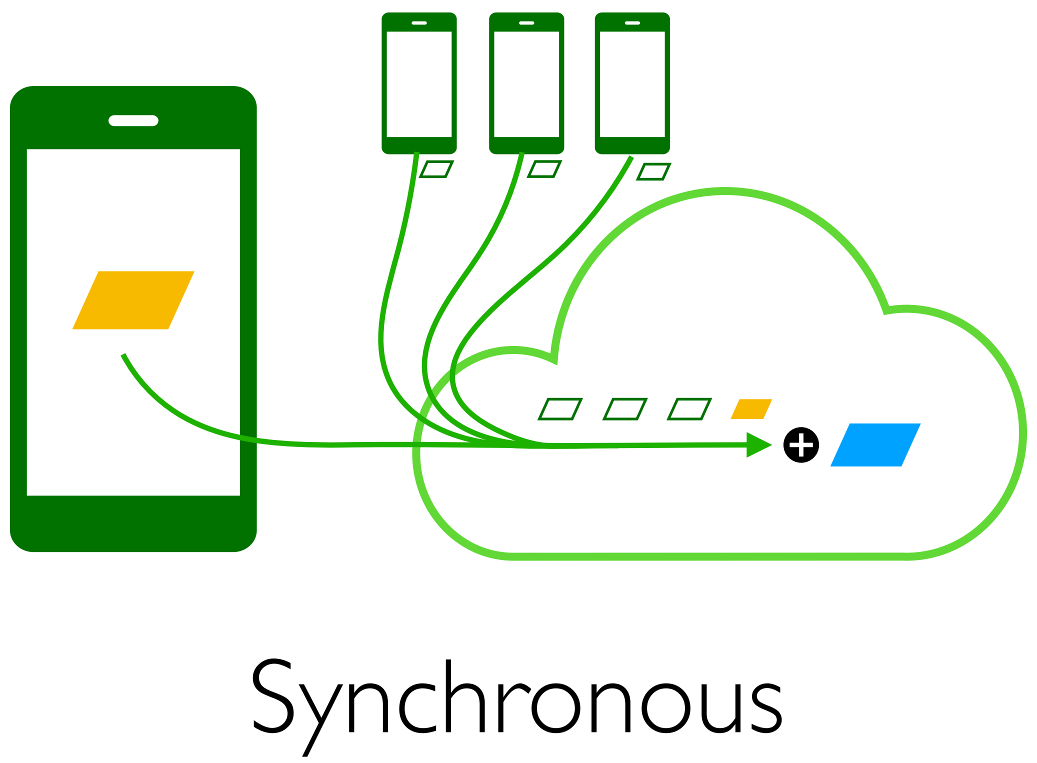

Federated Learning

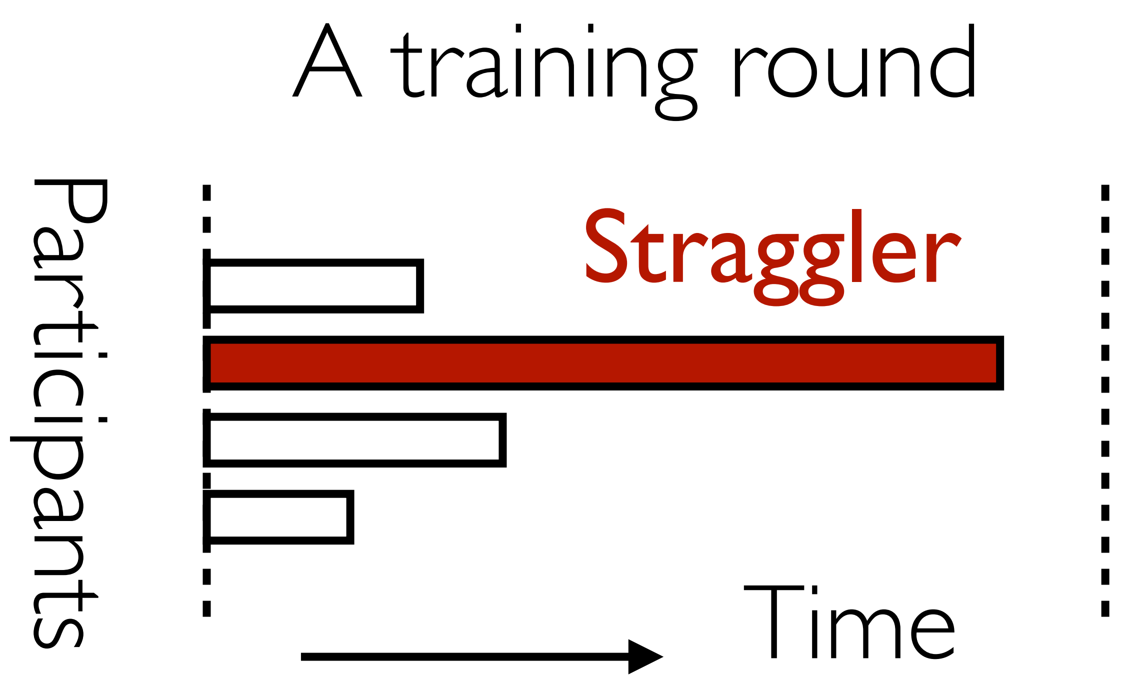
# Need for Pisces

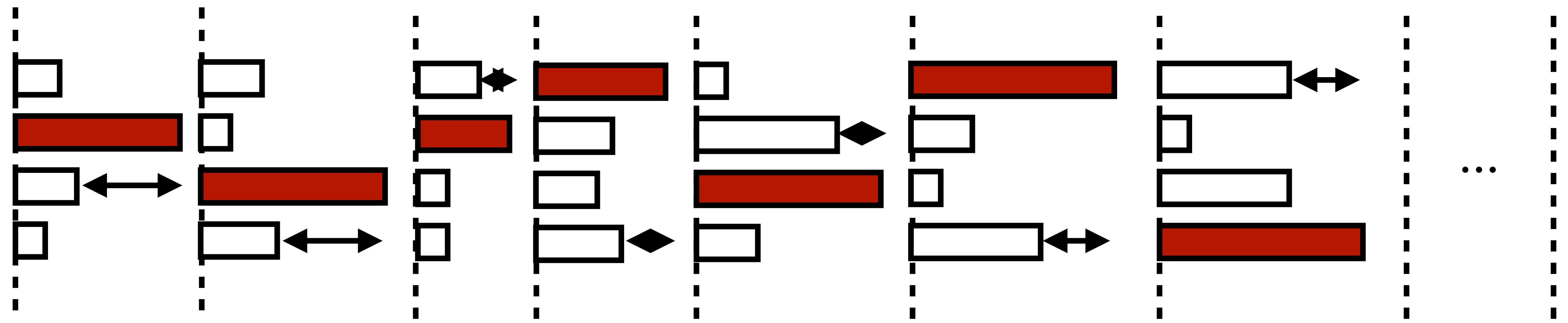



Federated Learning

# Need for Pisces



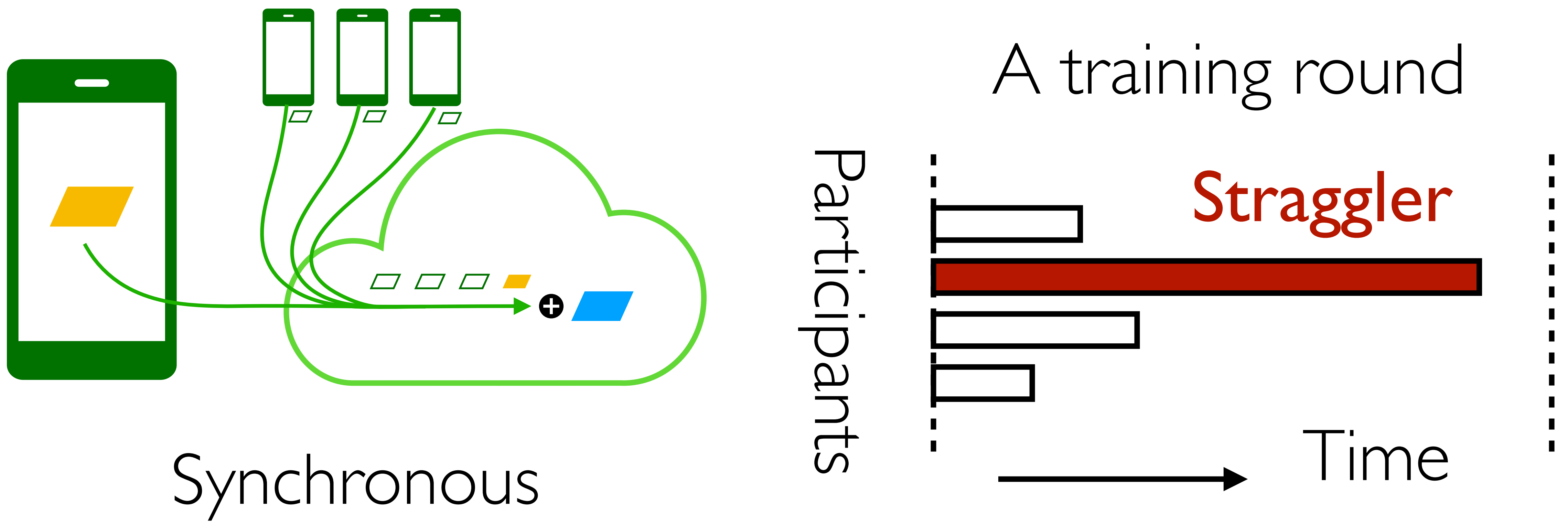



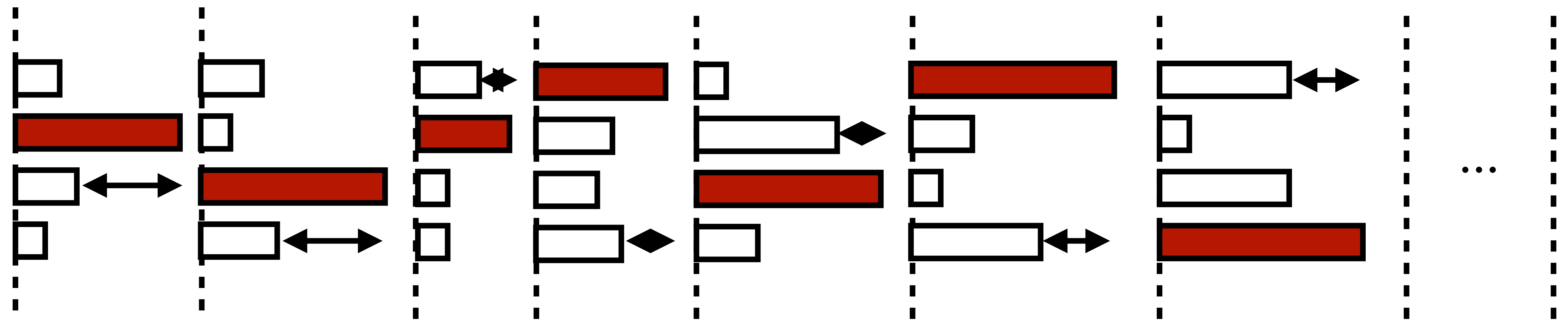




**Idle waiting**: 33.2% to 57.2%

Federated Learning

# Need for Pisces



**Idle waiting**: 33.2% to 57.2%

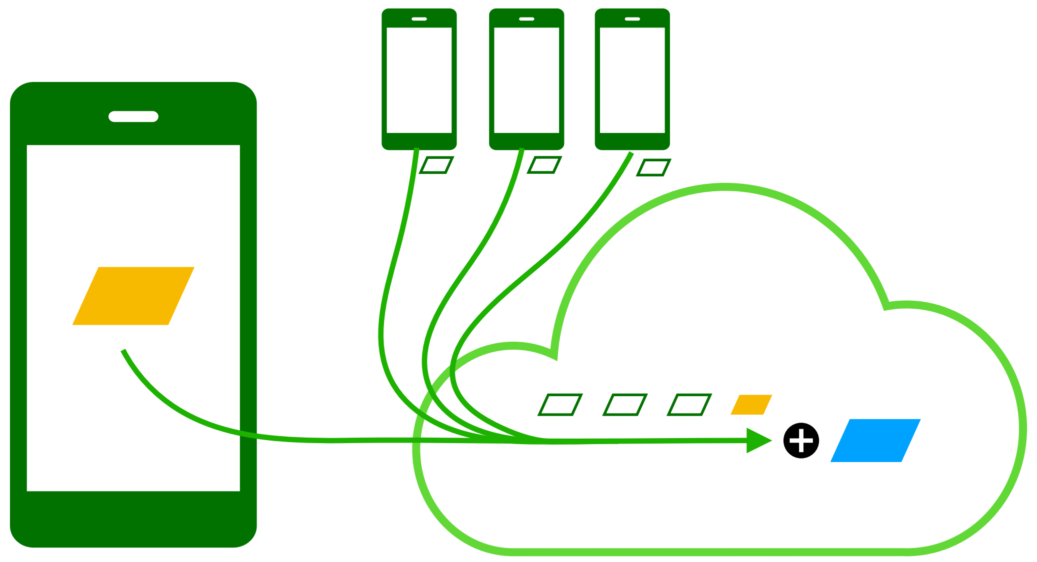
Federated Learning

**Potential approach**:

- Prioritize fast clients in selection

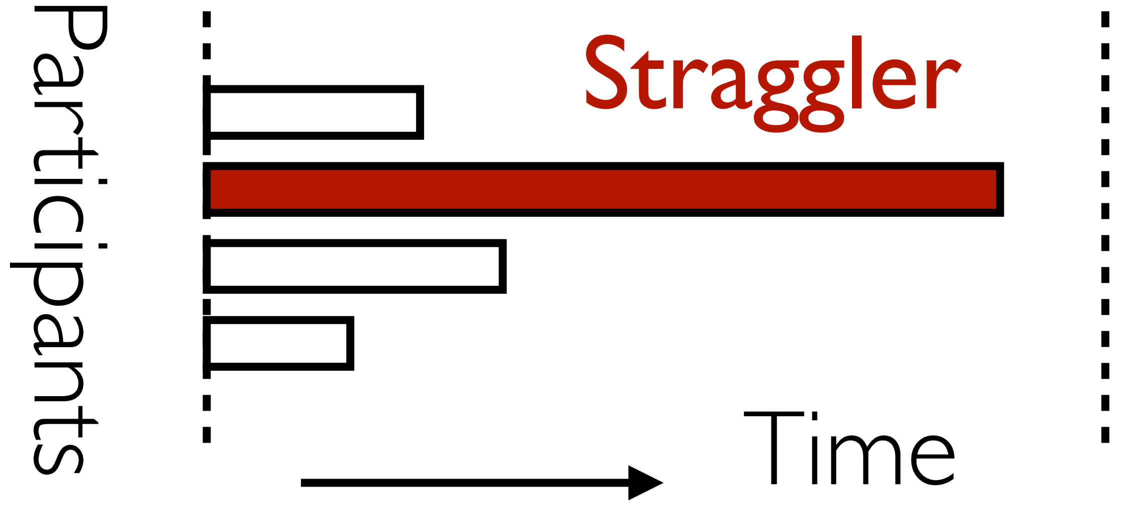
Time-to-accuracy = mean round time × # rounds

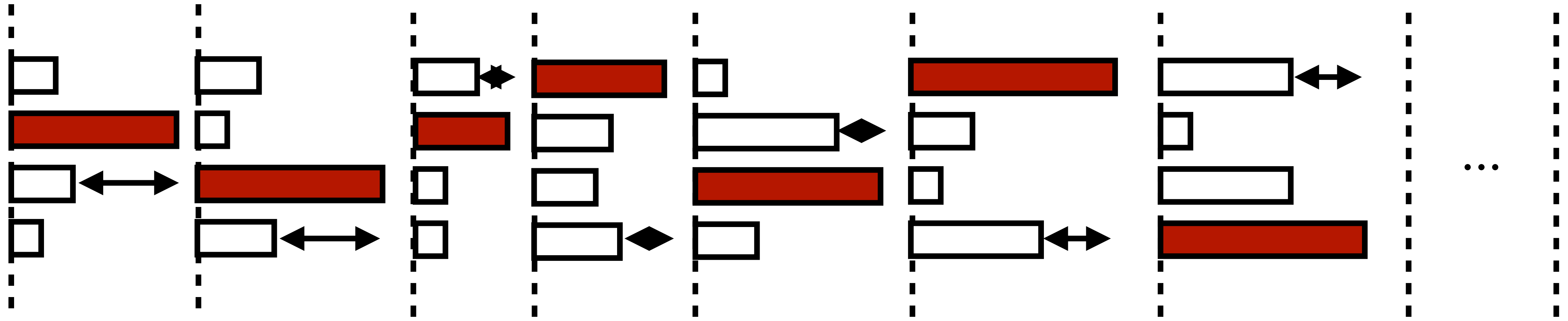
# Need for Pisces



Synchronous

A training round




**Idle waiting**: 33.2% to 57.2%

Federated Learning

**Potential approach**:

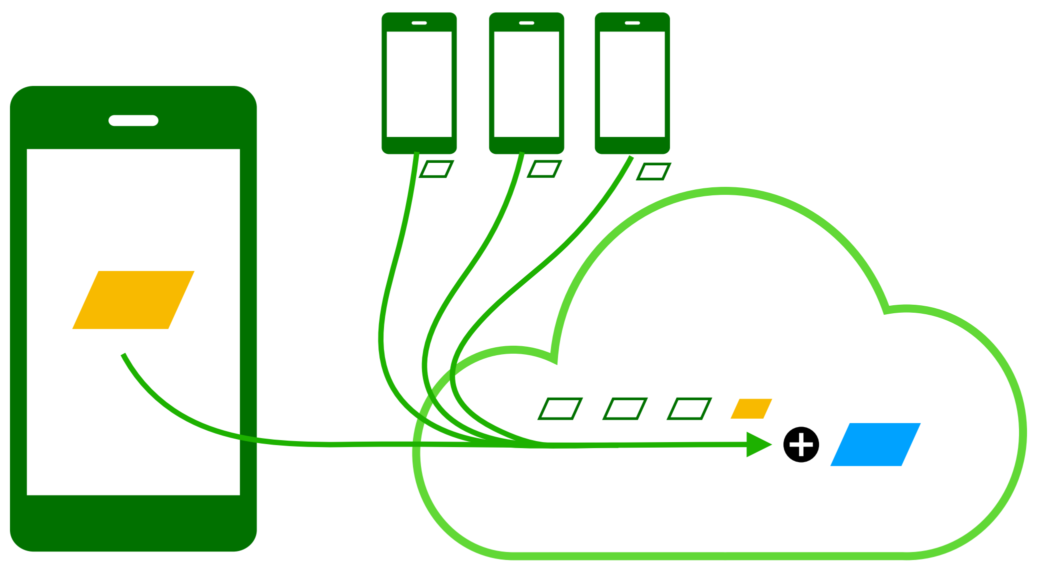
- Prioritize fast clients in selection

Selected clients have bad data quality…

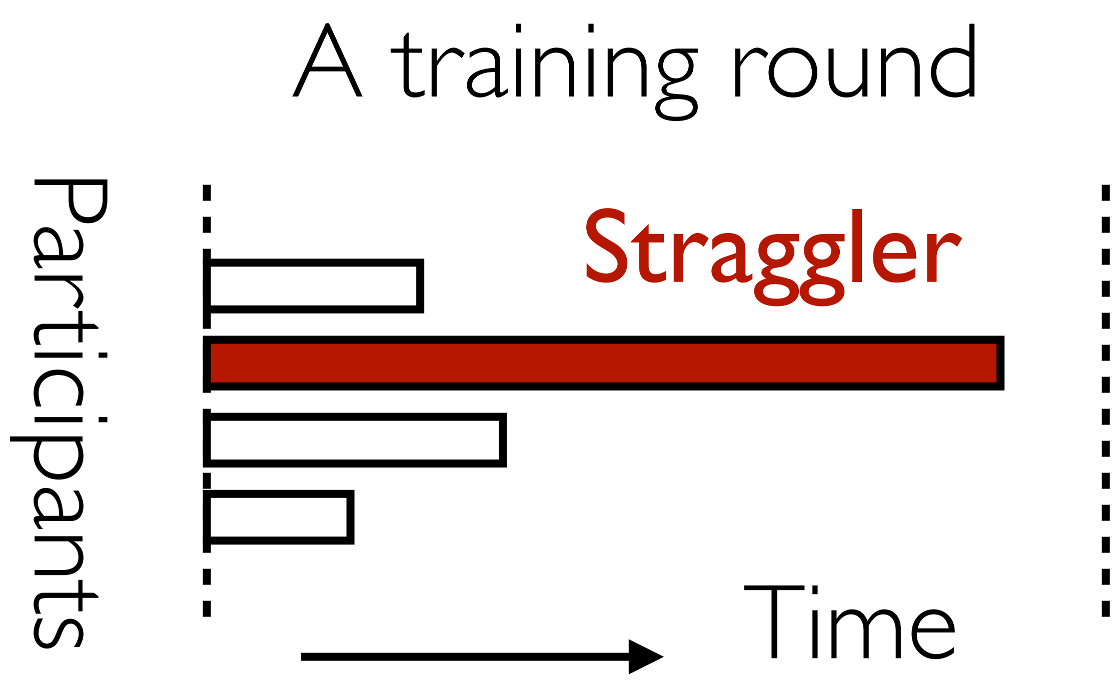
Time-to-accuracy = mean round time × # rounds

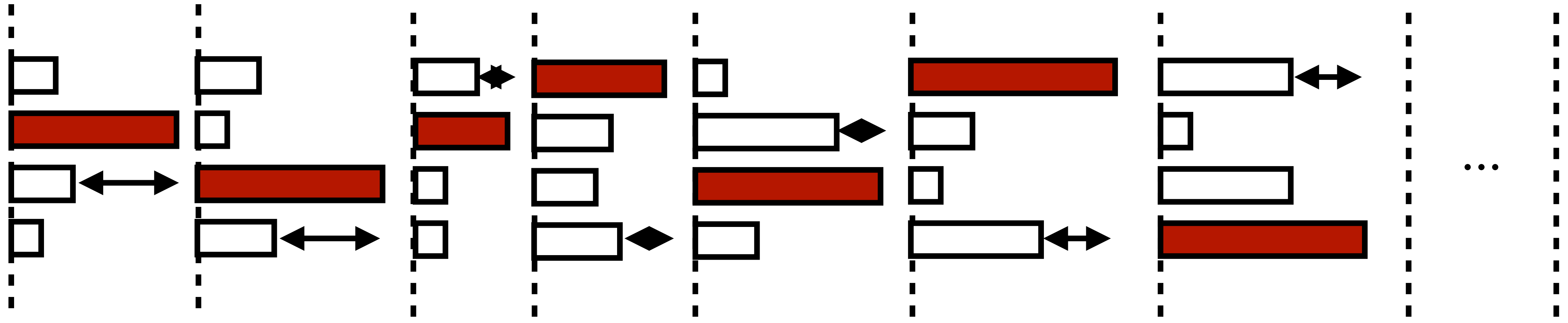
?

# Need for Pisces



Synchronous




**Idle waiting**: 33.2% to 57.2%

Federated Learning

**Potential approach**:

- Prioritize fast clients in selection
- Also consider their data quality

Time-to-accuracy = mean round time × # rounds

# Need for Pisces

**SOTA - Oort**[1]

[1] Lai et al. "Oort: Efficient Federated Learning via Guided Participant Selection", In OSDI '21

# Need for Pisces

**SOTA - Oort**[1]

- Definition of score for $U_i$ client $i$:

$$U_i = \underbrace{\left(\frac{T}{t_i}\right)^{\mathbf{1}(T<t_i)\times\alpha}}_{\text{speed}} \times \underbrace{|B_i|\sqrt{\frac{1}{|B_i|}\sum_{k\in B_i} Loss(k)^2}}_{\text{data quality}}$$

[1]Lai et al. "Oort: Efficient Federated Learning via Guided Participant Selection", In OSDI '21

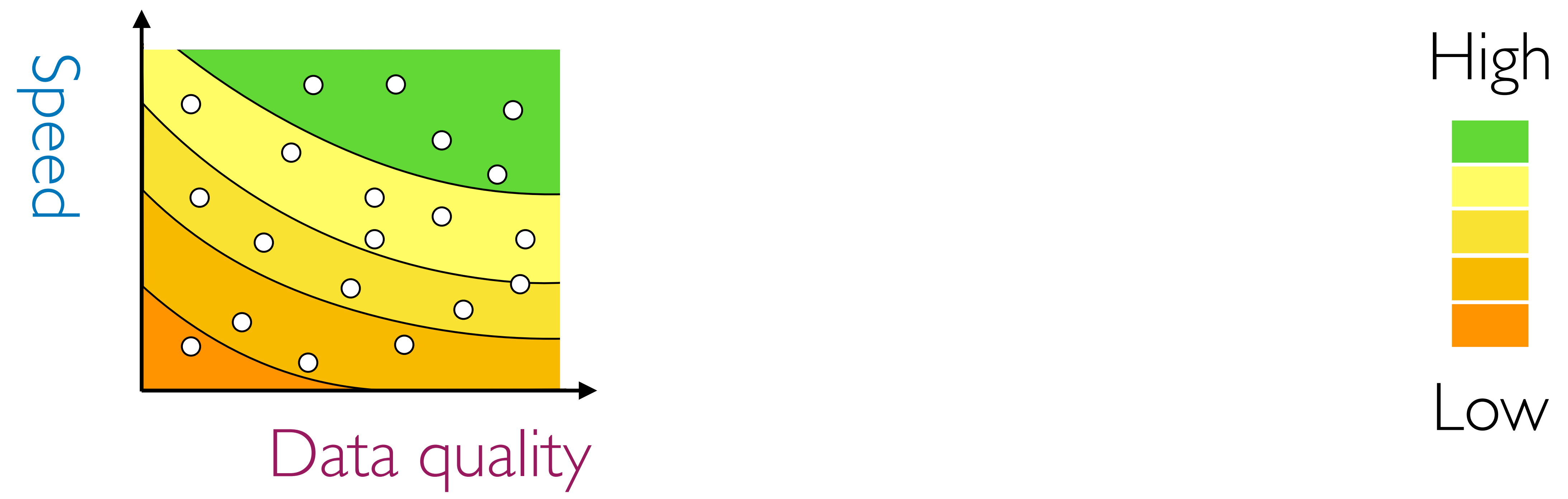
# Need for Pisces

**SOTA - Oort**[1]

- Definition of score for $U_i$ client $i$:

$$U_i = \underbrace{\left(\frac{T}{t_i}\right)^{\mathbf{1}(T<t_i)\times\alpha}}_{\text{speed}} \times \underbrace{|B_i|\sqrt{\frac{1}{|B_i|}\sum_{k\in B_i} Loss(k)^2}}_{\text{data quality}}$$

- Clients with higher score are selected




**Ideal**

→ High speed

& High data quality

[1]Lai et al. "Oort: Efficient Federated Learning via Guided Participant Selection", In OSDI '21

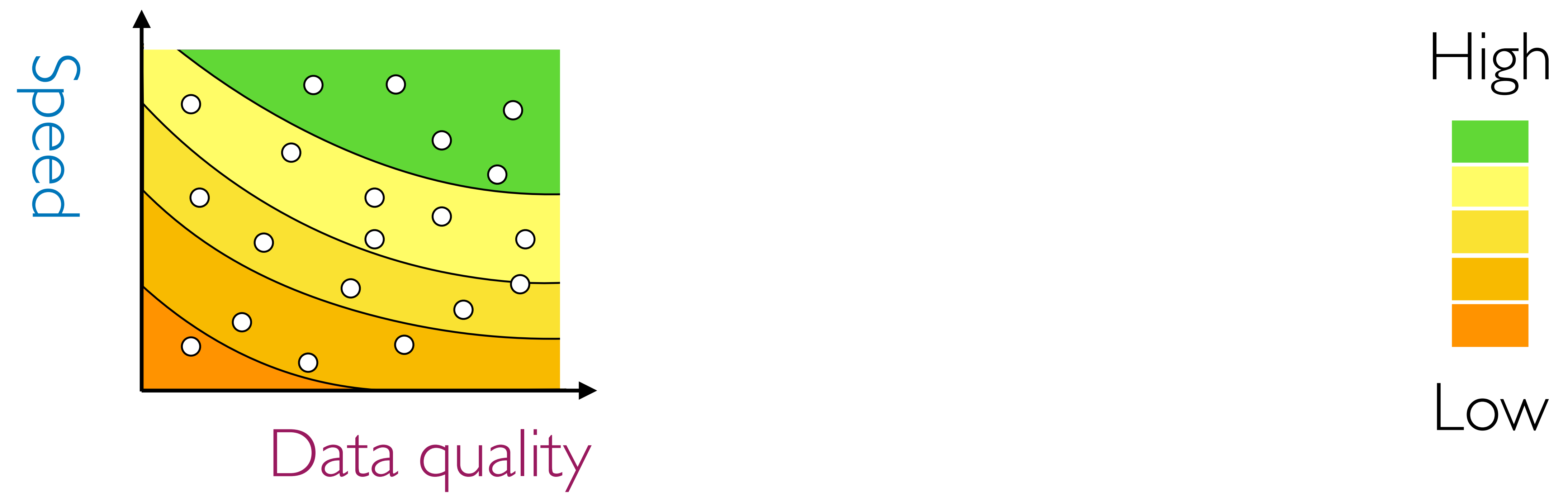
# Need for Pisces

**SOTA - Oort**[1]

- Definition of score for $U_i$ client $i$:

$$U_i = \underbrace{\left(\frac{T}{t_i}\right)^{\mathbb{1}(T<t_i)\times\alpha}}_{\text{speed}} \times \underbrace{|B_i|\sqrt{\frac{1}{|B_i|}\sum_{k\in B_i} Loss(k)^2}}_{\text{data quality}}$$

- Clients with higher score are selected




**Ideal**
→ High speed
& High data quality

$$\textbf{speed} \propto \frac{1}{\textbf{data quality}}$$

[1]Lai et al. "Oort: Efficient Federated Learning via Guided Participant Selection", In OSDI '21

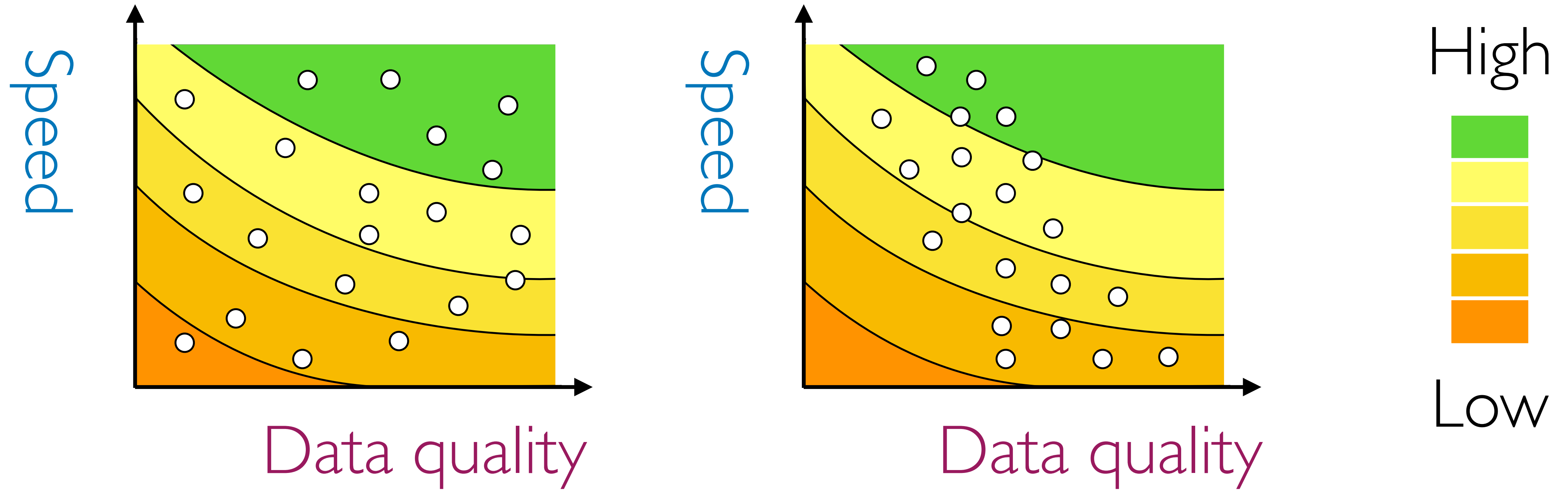
# Need for Pisces

**SOTA - Oort**[1]

- Definition of score for $U_i$ client $i$:

$$U_i = \underbrace{\left(\frac{T}{t_i}\right)^{\mathbb{1}(T<t_i)\times\alpha}}_{\text{speed}} \times \underbrace{|B_i|\sqrt{\frac{1}{|B_i|}\sum_{k\in B_i} Loss(k)^2}}_{\text{data quality}}$$

- Clients with higher score are selected




**Ideal**

→ High speed

& High data quality

$\textbf{speed} \propto \dfrac{1}{\textbf{data quality}}$

→ High speed

& Low data quality

[1]Lai et al. "Oort: Efficient Federated Learning via Guided Participant Selection", In OSDI '21

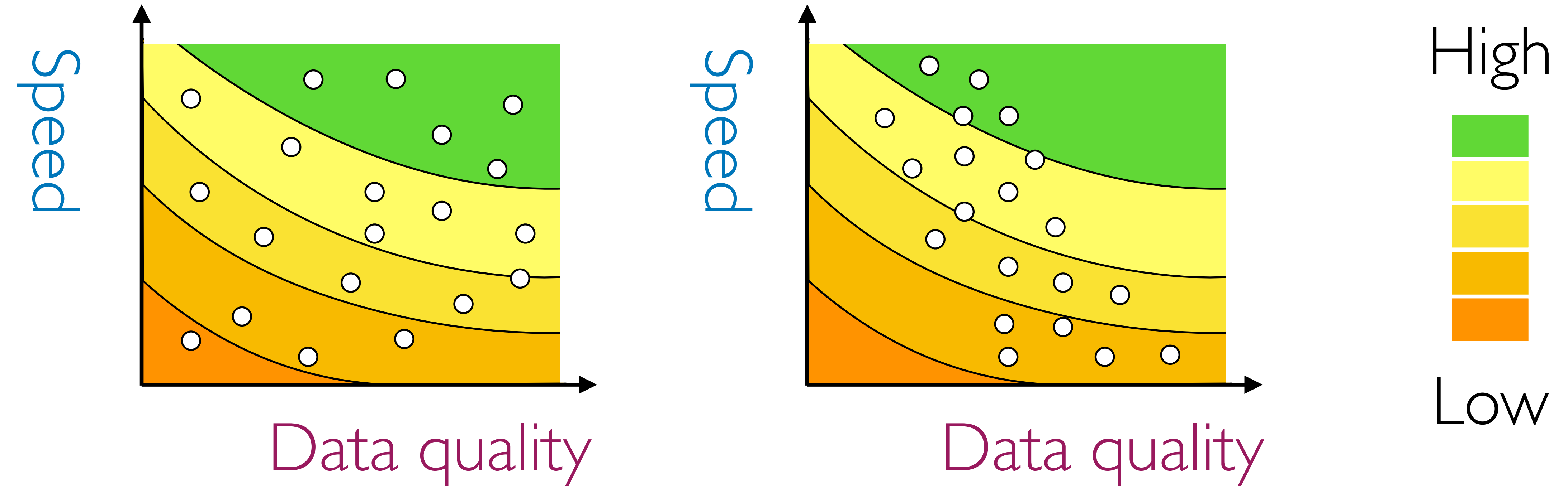
# Need for Pisces

### SOTA - Oort[1]

- Definition of score for $U_i$ client $i$:

$$U_i = \underbrace{\left(\frac{T}{t_i}\right)^{\mathbf{1}(T<t_i)\times\alpha}}_{\text{speed}} \times \underbrace{|B_i|\sqrt{\frac{1}{|B_i|}\sum_{k\in B_i} Loss(k)^2}}_{\text{data quality}}$$

- Clients with higher score are selected




**Ideal**
→ High speed
& High data quality

**speed** $\propto \dfrac{1}{\textbf{data quality}}$
→ High speed
& Low data quality

Oort is **2.7× worse** than random selection

[1] Lai et al. "Oort: Efficient Federated Learning via Guided Participant Selection", In OSDI '21

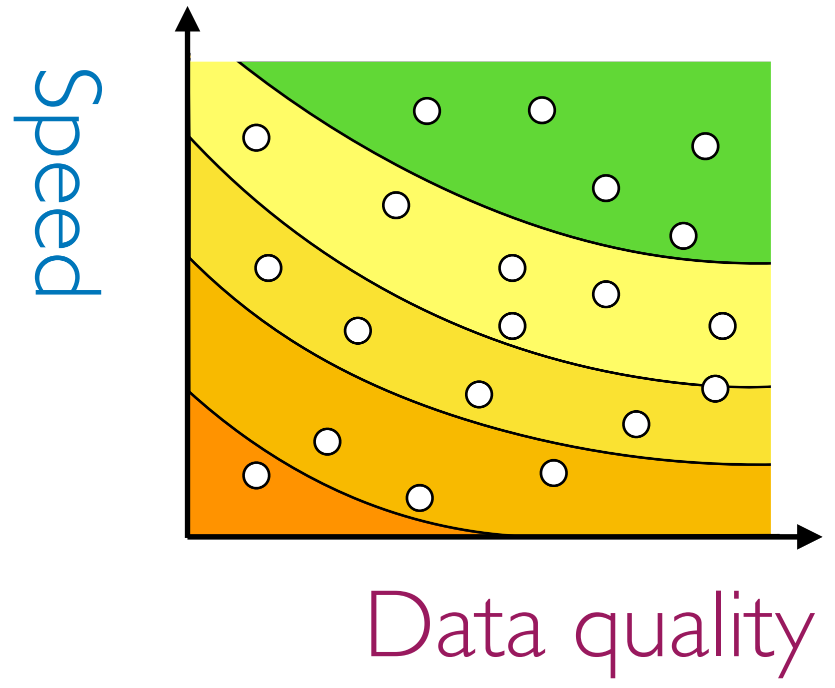
# Need for Pisces

**SOTA - Oort**[1]

- Definition of score for $U_i$ client $i$:

$$U_i = \underbrace{\left(\frac{T}{t_i}\right)^{\mathbb{1}(T<t_i)\times\alpha}}_{\text{speed}} \times \underbrace{|B_i|\sqrt{\frac{1}{|B_i|}\sum_{k\in B_i} Loss(k)^2}}_{\text{data quality}}$$

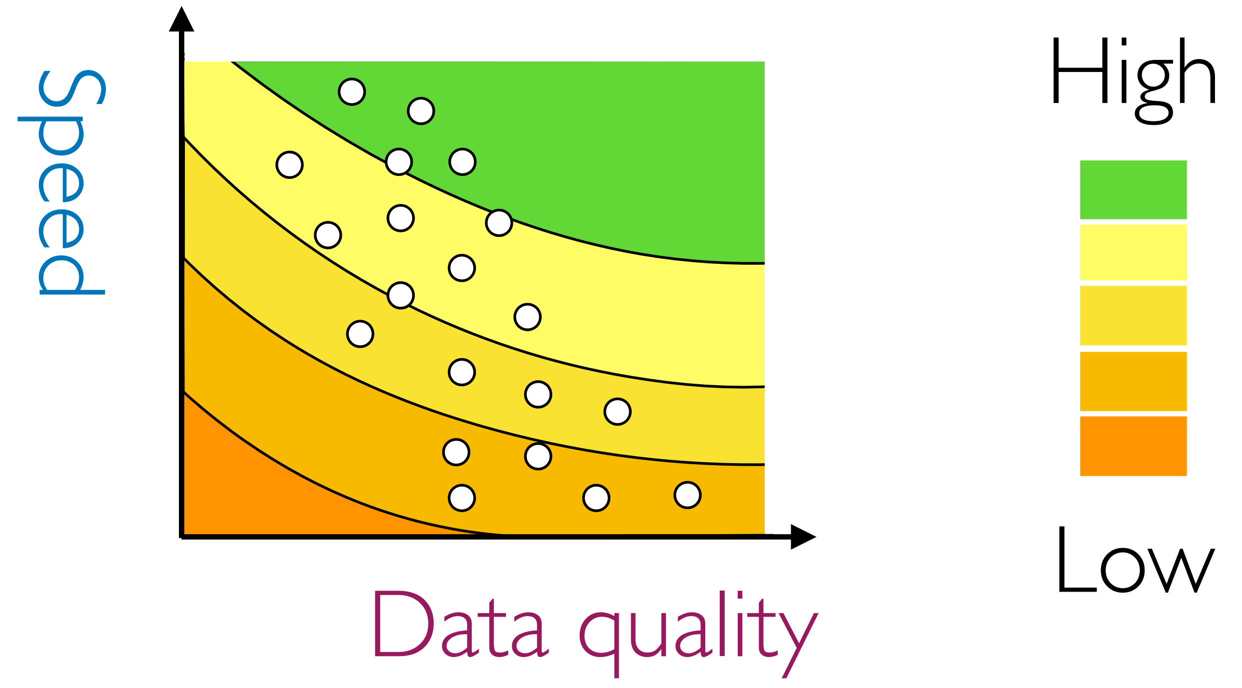
- Clients with higher score are selected




**Ideal**

→ High speed

& High data quality




$$\textbf{speed} \propto \frac{1}{\textbf{data quality}}$$

→ High speed

& Low data quality

Oort is **2.7× worse** than random selection

**Problem**: Navigation between clients' **speed** and **data quality** is **inherently tricky**

[1]Lai et al. "Oort: Efficient Federated Learning via Guided Participant Selection", In OSDI '21

# Pisces - Overview

# Pisces - Overview

**Mitigating** straggler effects for maximum efficiency

# Pisces - Overview

**Mitigating** straggler effects for maximum efficiency

Principled **asynchronous** training: **Side-step** the tricky speed-data tradeoffs with **minimum** side-effects

# Pisces - Overview

**Mitigating** straggler effects for maximum efficiency

Principled **asynchronous** training: **Side-step** the tricky speed-data tradeoffs with **minimum** side-effects

**Theory**

**Provable convergence** for smooth non-convex problems

# Pisces - Overview

**Mitigating** straggler effects for maximum efficiency

Principled **asynchronous** training: **Side-step** the tricky speed-data tradeoffs with **minimum** side-effects

**Theory**

**Provable convergence** for smooth non-convex problems

**Efficiency**

Improvement in **time-to-accuracy**

# Pisces - Overview

**Mitigating** straggler effects for maximum efficiency

Principled **asynchronous** training: **Side-step** the tricky speed-data tradeoffs with **minimum** side-effects

## Theory

**Provable convergence** for smooth non-convex problems

## Efficiency

Improvement in **time-to-accuracy**

## Practicality

Easily Integrated to **production frameworks**

# Problem: Straggler mitigation

**Asynchronous training**:

# Problem: Straggler mitigation

**Asynchronous training**:

- Early **aggregate** available local updates **without waiting** for other running participants

# Problem: Straggler mitigation

**Asynchronous training**:

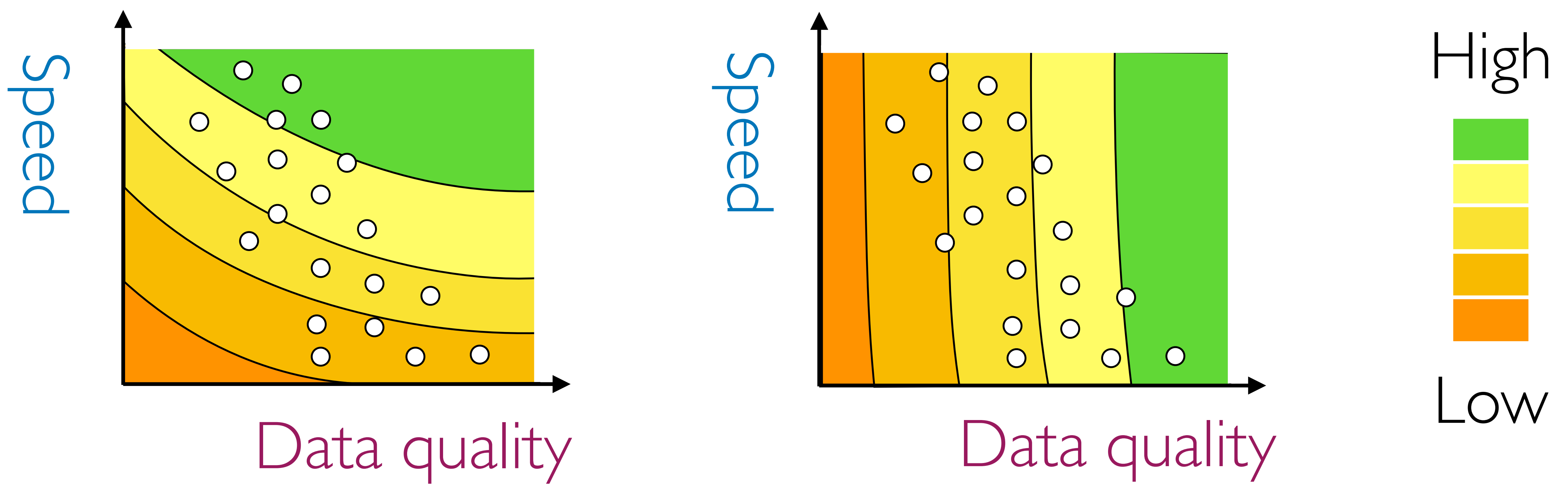
- Early **aggregate** available local updates **without waiting** for other running participants
- **Immediately invokes** available clients

# Problem: Straggler mitigation

**Asynchronous training**:

- Early **aggregate** available local updates **without waiting** for other running participants
- **Immediately invokes** available clients

**Intuition**: side-step the speed-data tradeoff




**Sync** → High speed & Low data quality

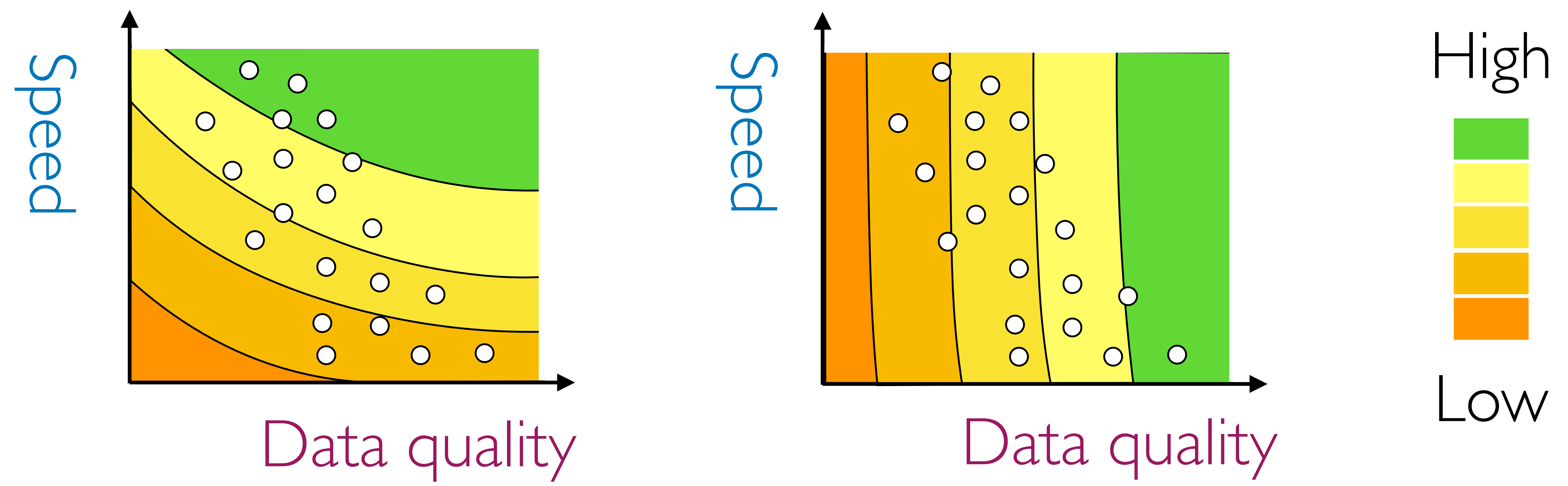
**Async** → High data quality (whatever speed)

# Problem: Straggler mitigation

**Asynchronous training**:

- Early **aggregate** available local updates **without waiting** for other running participants
- **Immediately invokes** available clients

**Intuition**: side-step the speed-data tradeoff




**Sync** → High speed & Low data quality

**Async** → High data quality (whatever speed)

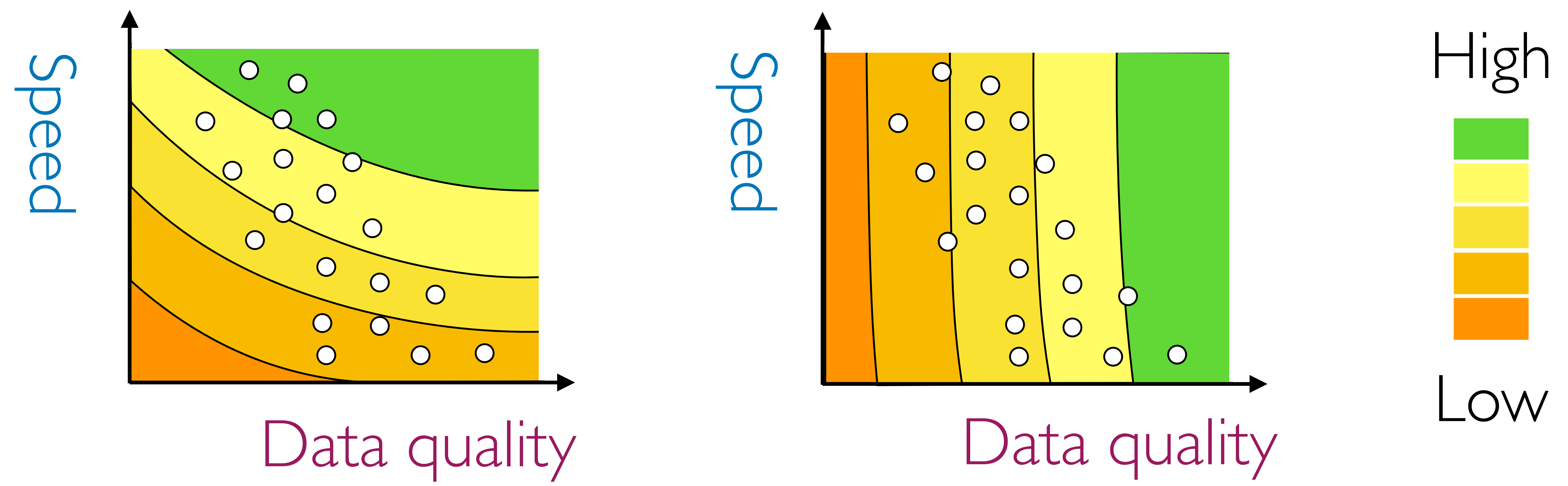
**Problem**: tolerance of slow clients yields **stale** local updates

# Problem: Straggler mitigation

**Asynchronous training**:

- Early **aggregate** available local updates **without waiting** for other running participants
- **Immediately invokes** available clients

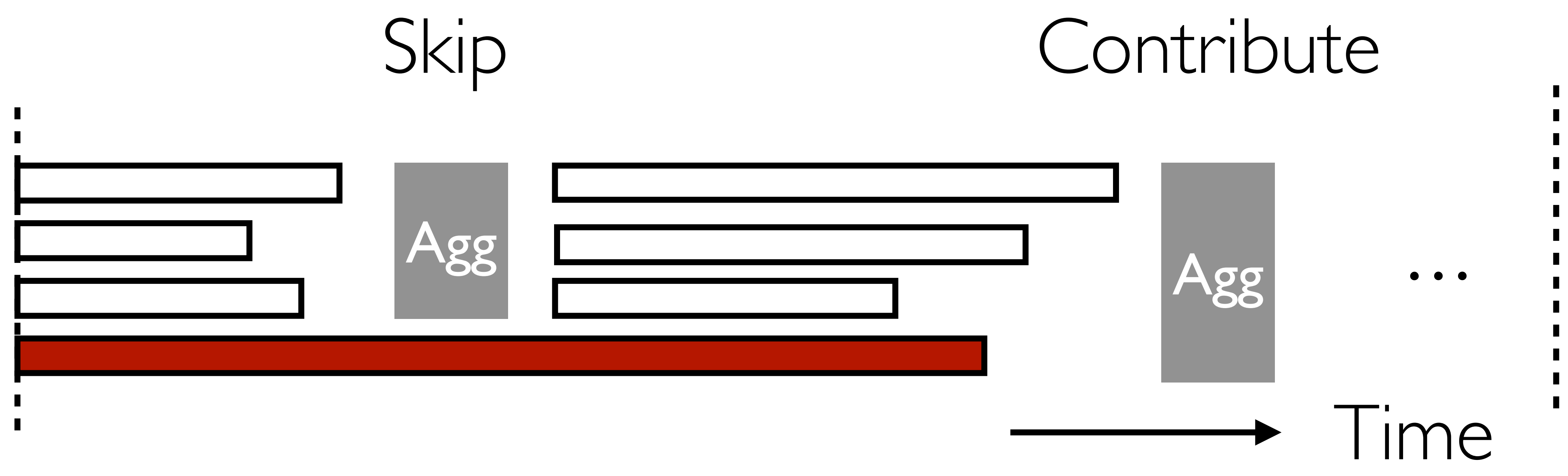
**Intuition**: side-step the speed-data tradeoff




**Sync** → High speed & Low data quality

**Async** → High data quality (whatever speed)

**Problem**: tolerance of slow clients yields **stale** local updates

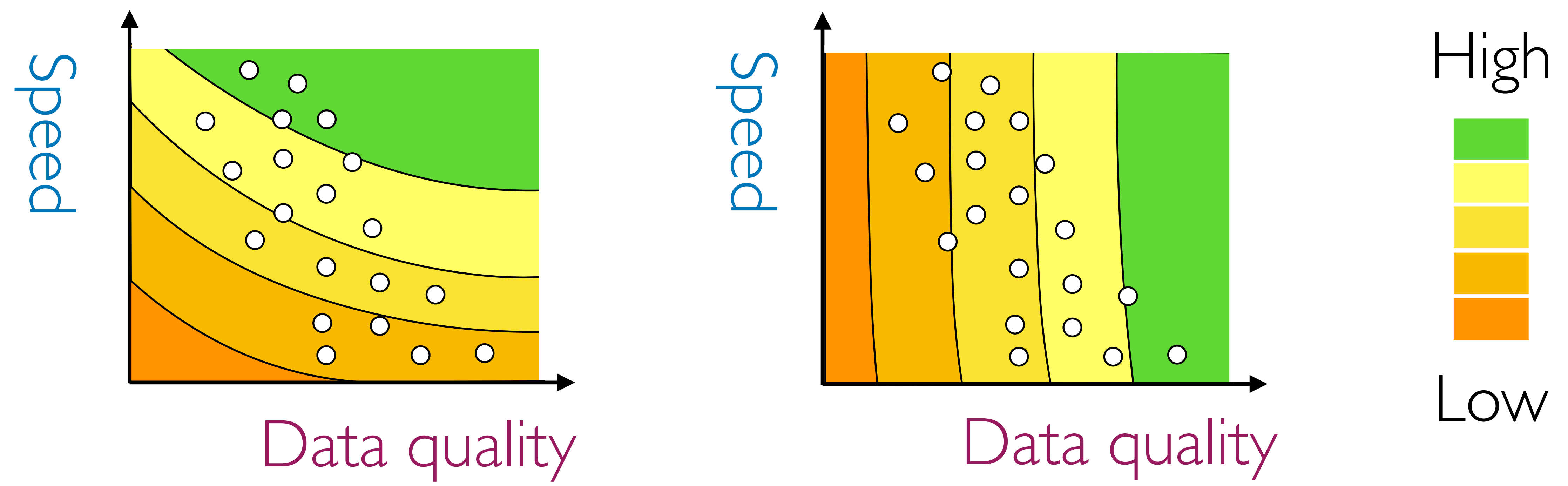


# Problem: Straggler mitigation

**Asynchronous training**:

- Early **aggregate** available local updates **without waiting** for other running participants
- **Immediately invokes** available clients

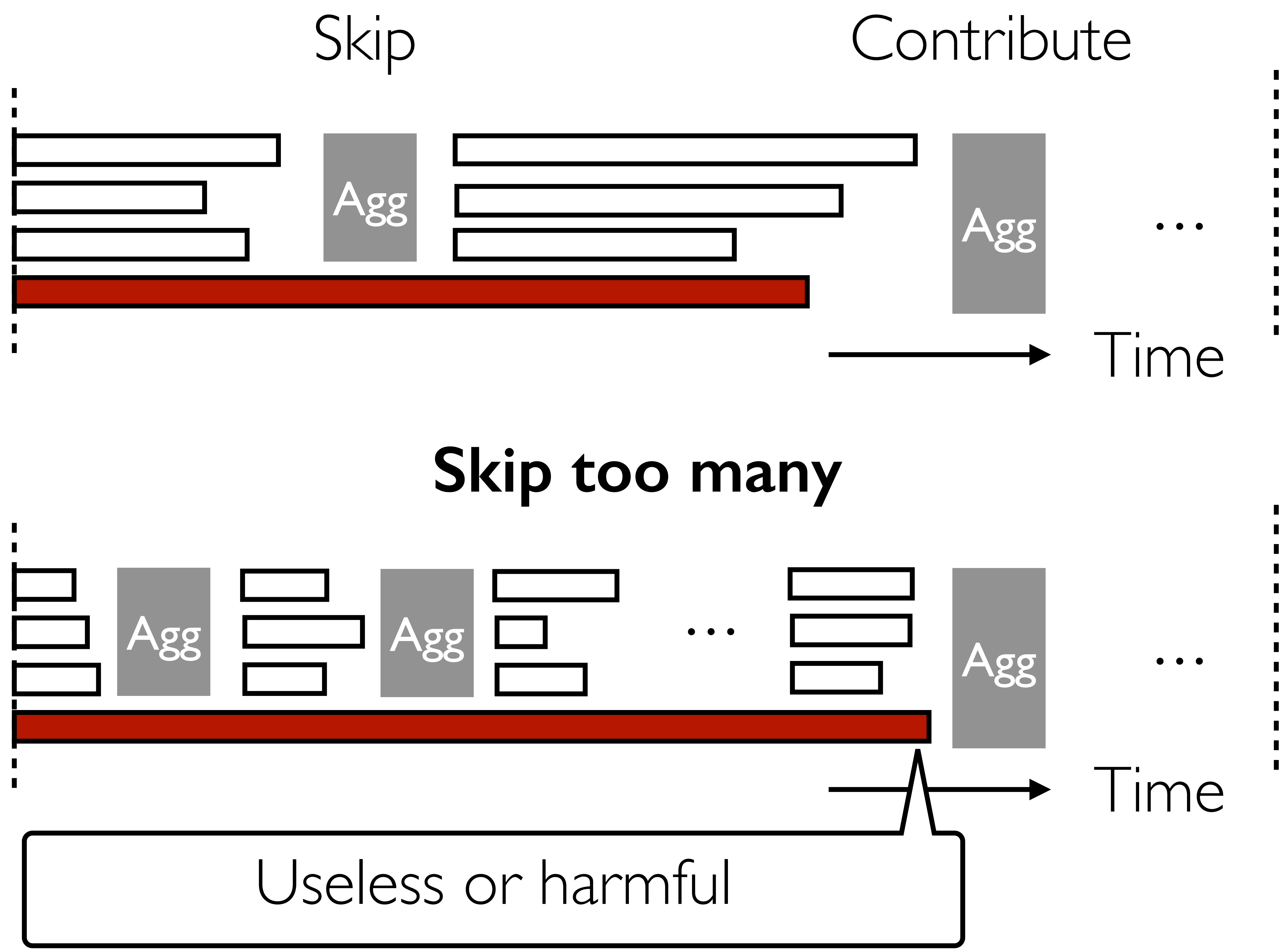
**Intuition**: side-step the speed-data tradeoff




**Sync** → High speed & Low data quality

**Async** → High data quality (whatever speed)

**Problem**: tolerance of slow clients yields **stale** local updates

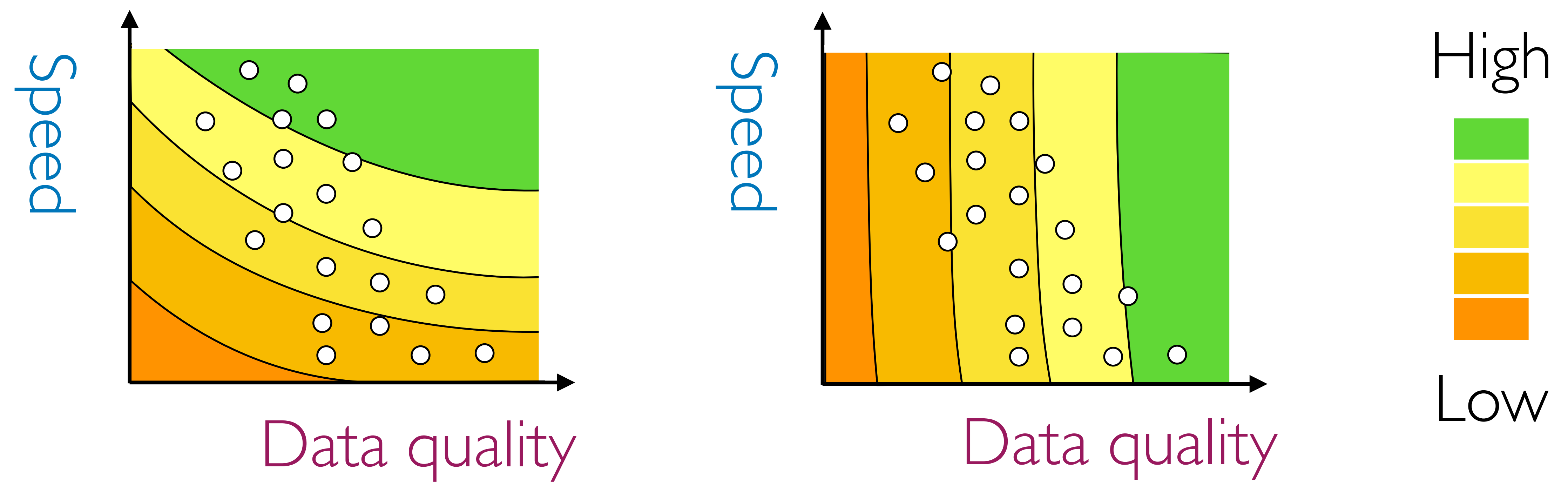


# Problem: Straggler mitigation

**Asynchronous training**:

- Early **aggregate** available local updates **without waiting** for other running participants
- **Immediately invokes** available clients

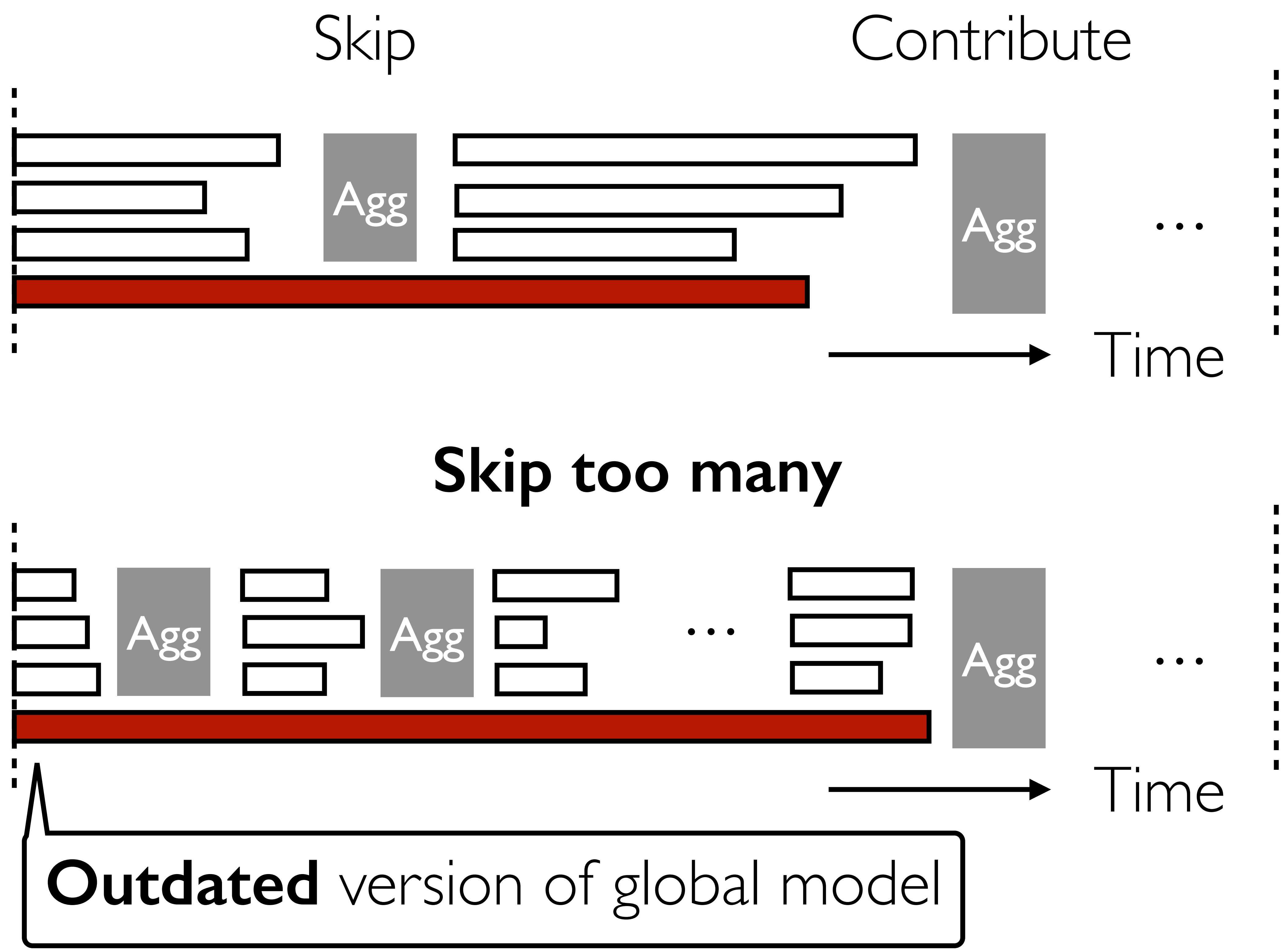
**Intuition**: side-step the speed-data tradeoff




**Sync** → High speed & Low data quality

**Async** → High data quality (whatever speed)

**Problem**: tolerance of slow clients yields **stale** local updates

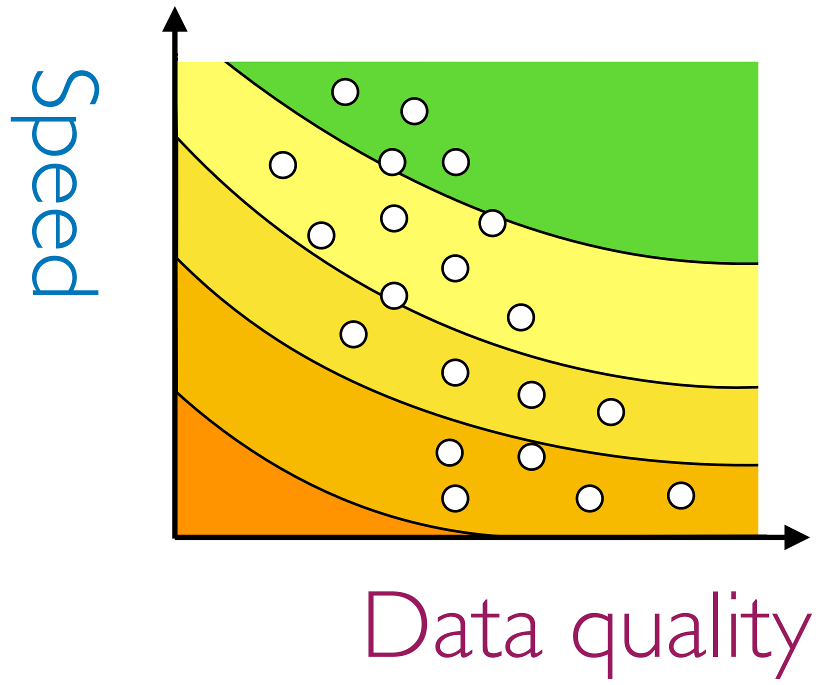


# Problem: Straggler mitigation

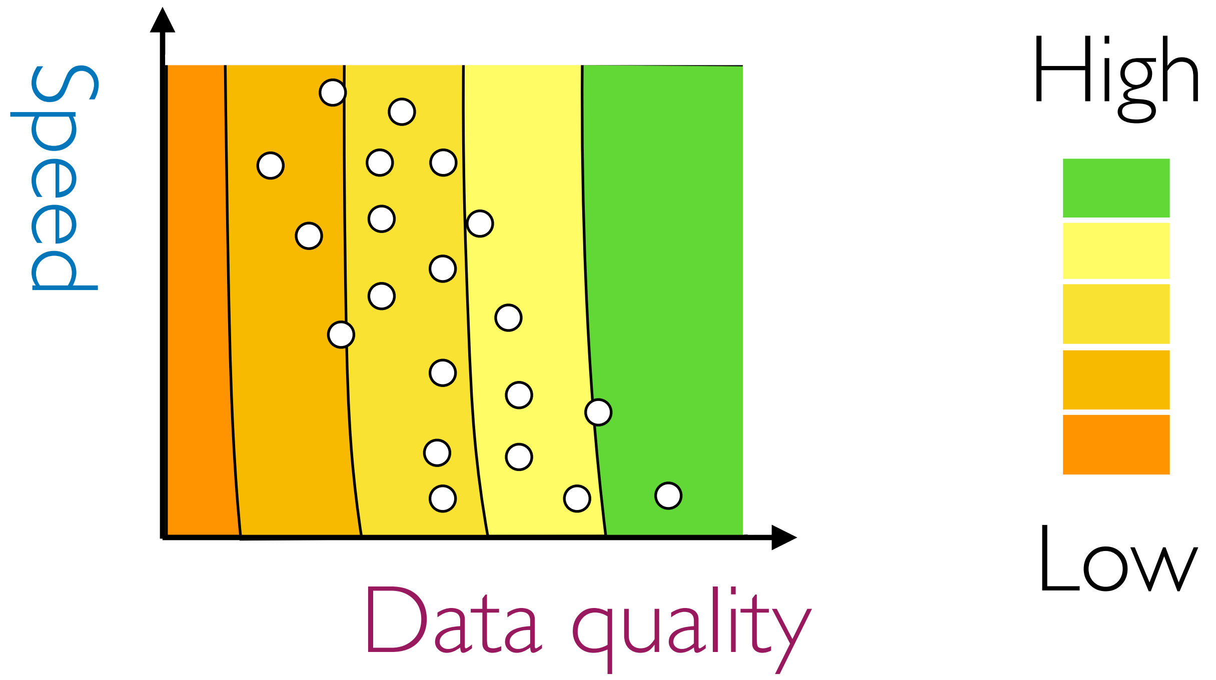
**Asynchronous training**:

- Early **aggregate** available local updates **without waiting** for other running participants
- **Immediately invokes** available clients

**Intuition**: side-step the speed-data tradeoff

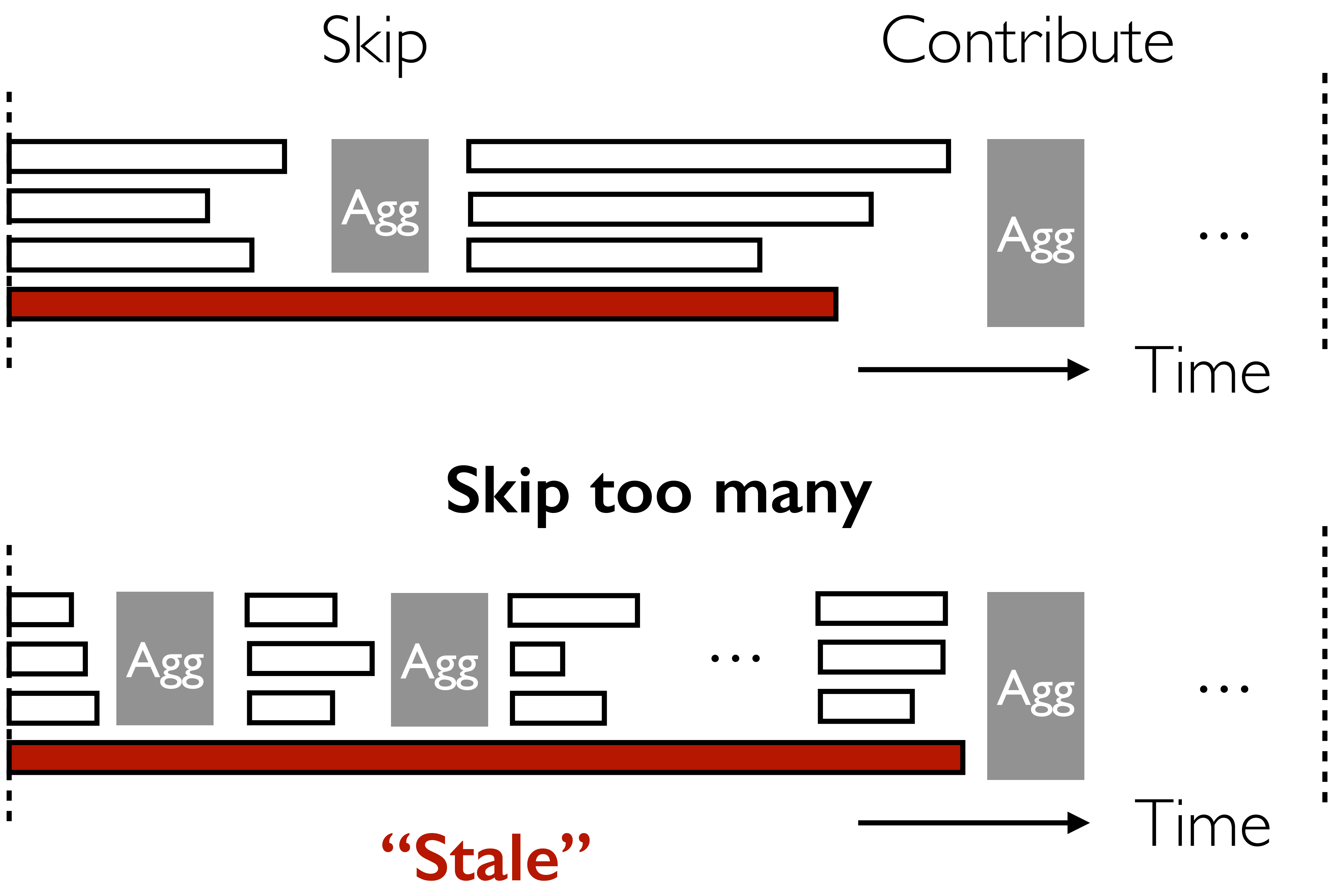



**Sync** → High speed & Low data quality




**Async** → High data quality (whatever speed)

**Problem**: tolerance of slow clients yields **stale** local updates



# Problem: Straggler mitigation

**Potential approach**:

- **Asynchronous** training

# Problem: Straggler mitigation

**Potential approach**:

- **Asynchronous** training
- **Pause aggregation** when someone will exceed the **staleness bound**

# Problem: Straggler mitigation

**Potential approach**:

- **Asynchronous** training
- **Pause aggregation** when someone will exceed the **staleness bound**

E.g., staleness bound is 2

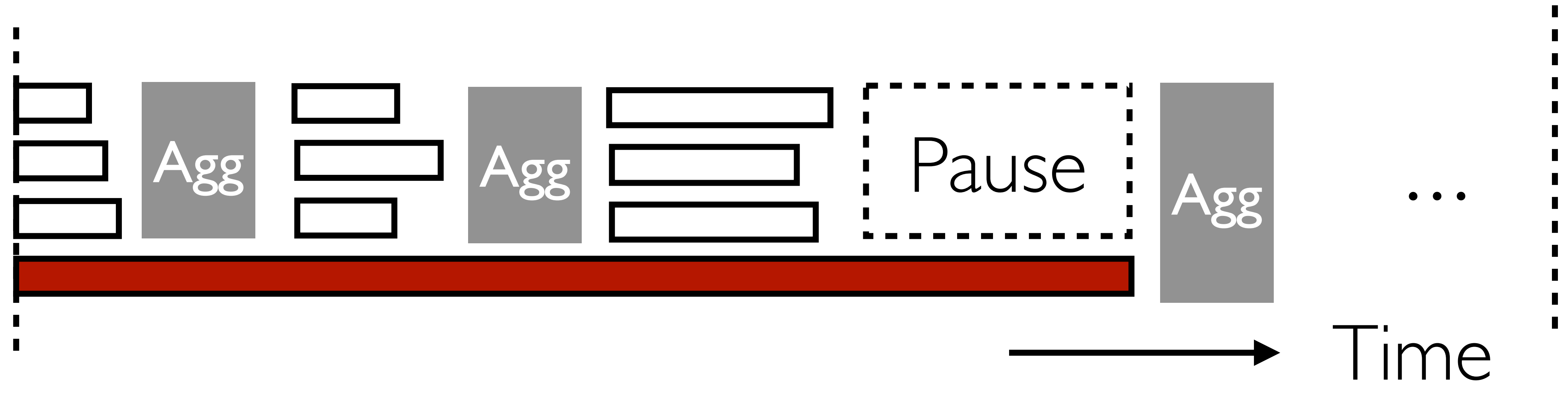
# Problem: Straggler mitigation

**Potential approach**:

- **Asynchronous** training
- **Pause aggregation** when someone will exceed the **staleness bound**

E.g., staleness bound is 2

→No more than 2 aggregations behind



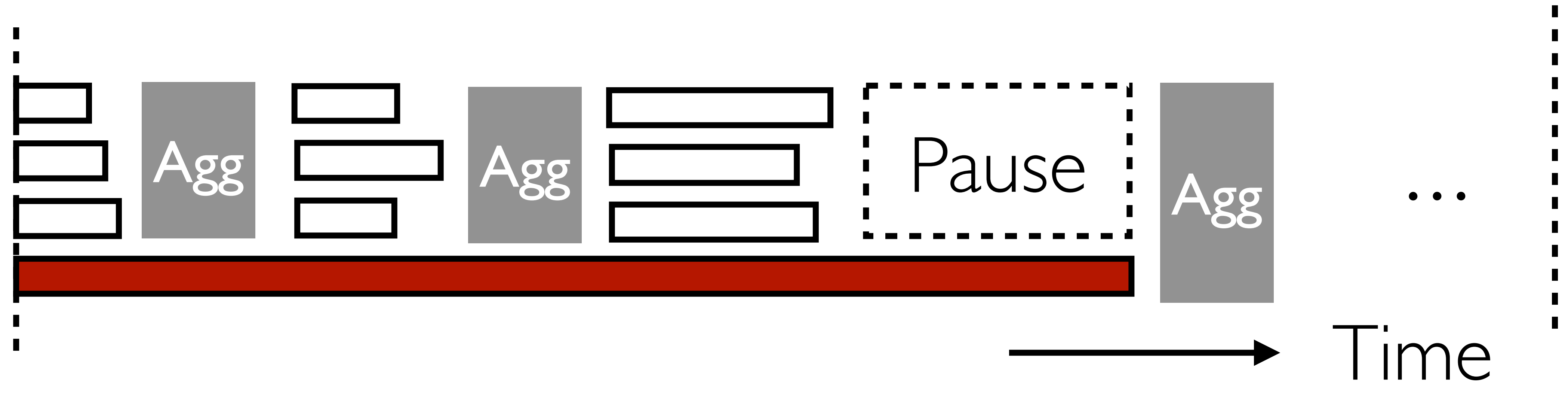
# Problem: Straggler mitigation

**Potential approach**:

- **Asynchronous** training
- **Pause aggregation** when someone will exceed the **staleness bound**

E.g., staleness bound is 2

→No more than 2 aggregations behind




Stale Synchronous Parallel (**SSP**)[1] in traditional ML

[1] Ho et al. "More Effective Distributed ML via a Stale Synchronous Parallel Parameter Server", In NeurIPS '13

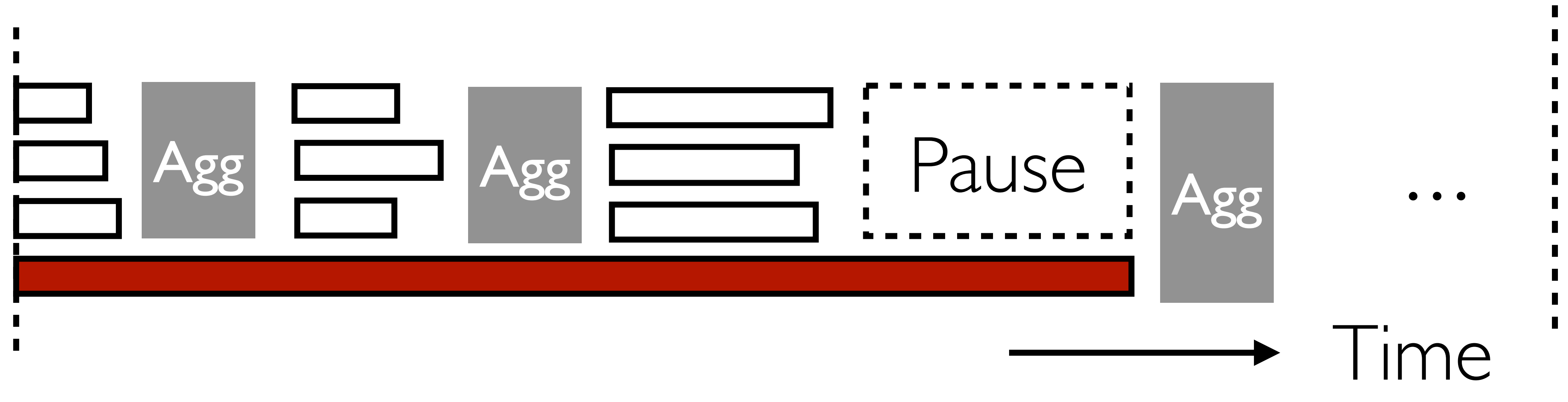
# Problem: Straggler mitigation

**Potential approach**:

- **Asynchronous** training
- **Pause aggregation** when someone will exceed the **staleness bound**

E.g., staleness bound is 2

→No more than 2 aggregations behind




Stale Synchronous Parallel (**SSP**)[1] in traditional ML

**Problem**: Unaware of clients' **speed** and may be **suboptimal** in **client efficiency**

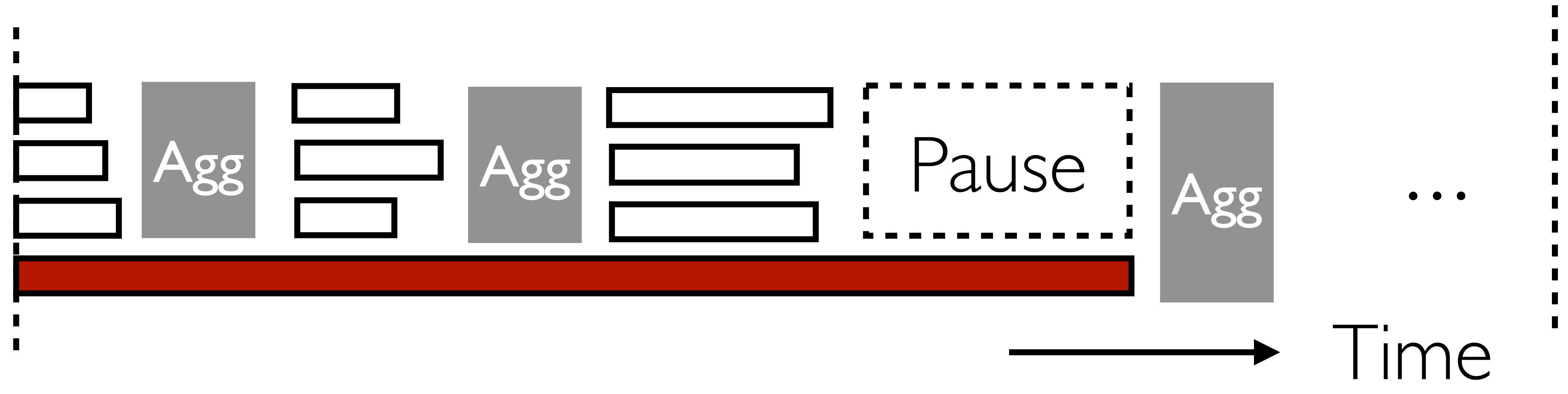
[1] Ho et al. "More Effective Distributed ML via a Stale Synchronous Parallel Parameter Server", In NeurIPS '13

# Problem: Straggler mitigation

**Potential approach**:

- **Asynchronous** training
- **Pause aggregation** when someone will exceed the **staleness bound**

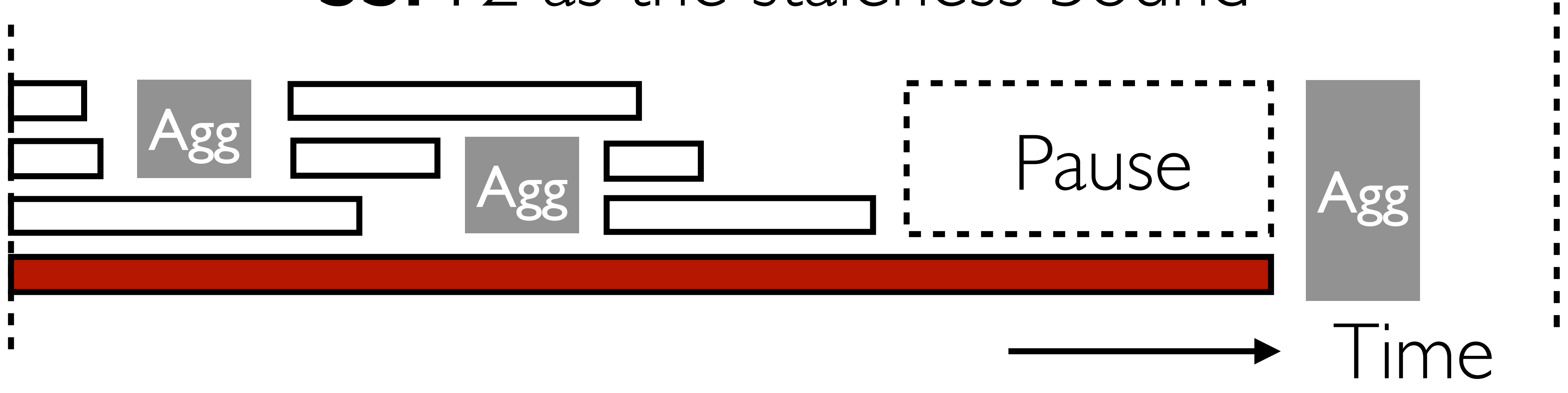
E.g., staleness bound is 2
→No more than 2 aggregations behind




Stale Synchronous Parallel (**SSP**)[1] in traditional ML

**Problem**: Unaware of clients' **speed** and may be **suboptimal** in **client efficiency**

**SSP**: 2 as the staleness bound



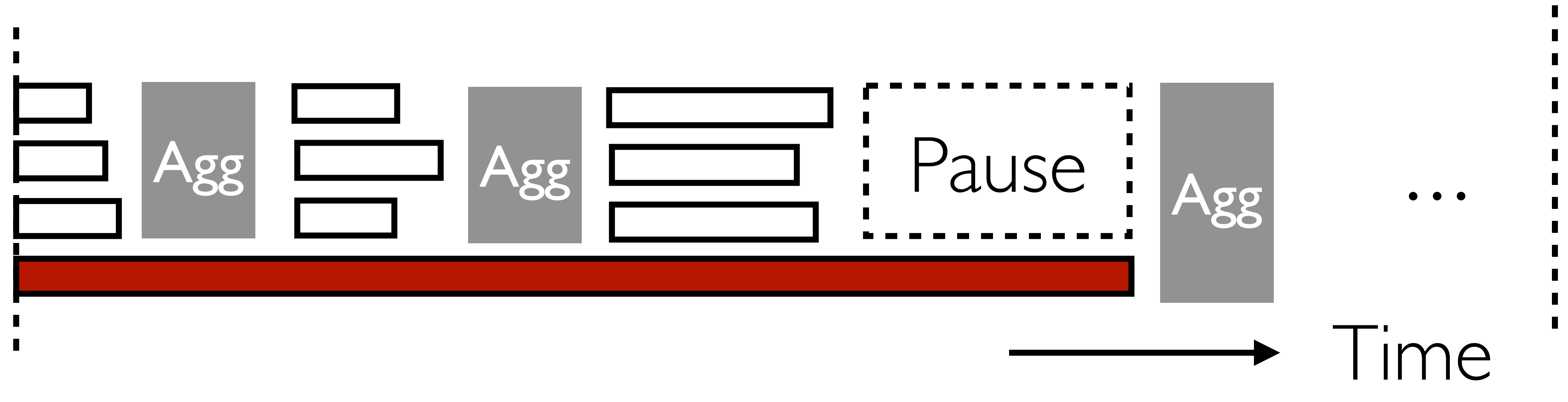

[1] Ho et al. "More Effective Distributed ML via a Stale Synchronous Parallel Parameter Server", In NeurIPS '13

# Problem: Straggler mitigation

**Potential approach**:

- **Asynchronous** training
- **Pause aggregation** when someone will exceed the **staleness bound**

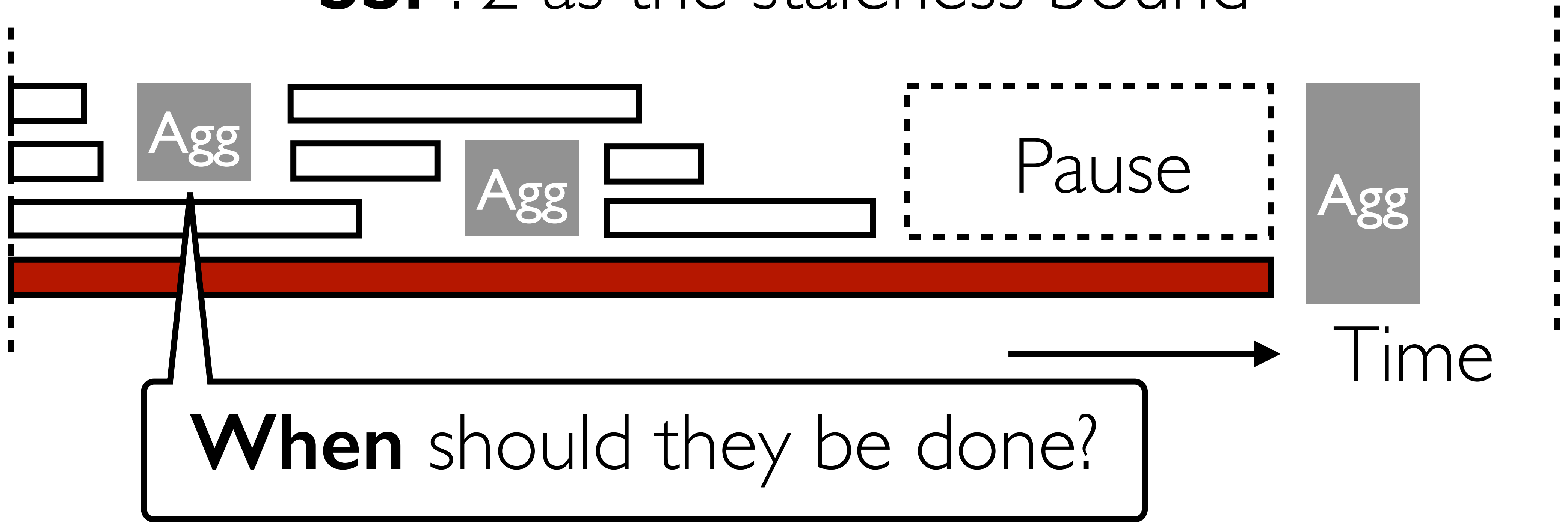
E.g., staleness bound is 2
→No more than 2 aggregations behind




Stale Synchronous Parallel (**SSP**)[1] in traditional ML

**Problem**: Unaware of clients' **speed** and may be **suboptimal** in **client efficiency**

**SSP**: 2 as the staleness bound




[1]Ho et al. "More Effective Distributed ML via a Stale Synchronous Parallel Parameter Server", In NeurIPS '13

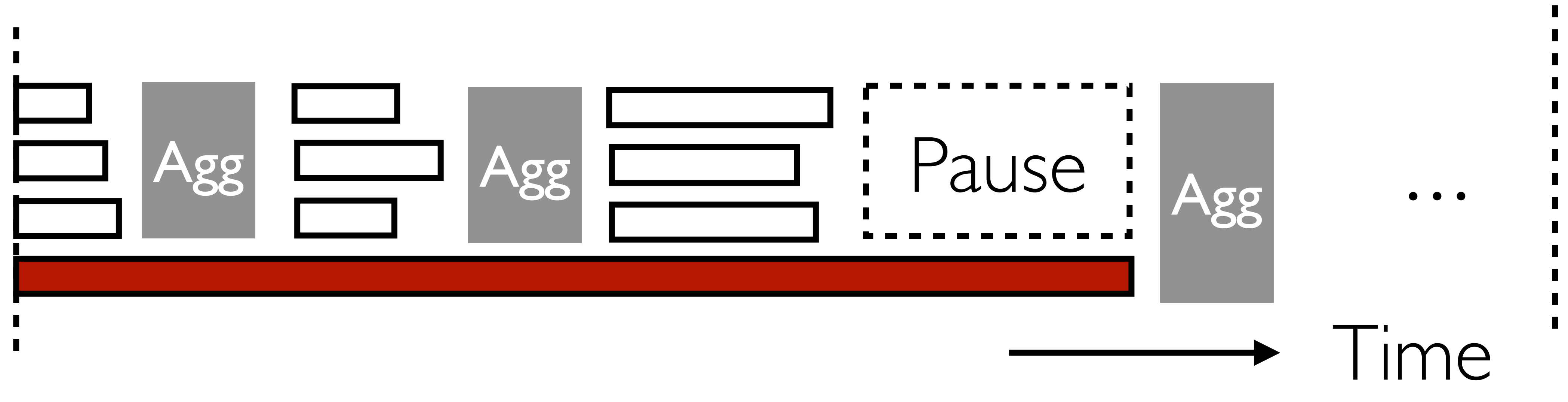
# Problem: Straggler mitigation

**Potential approach**:

- **Asynchronous** training
- **Pause aggregation** when someone will exceed the **staleness bound**

E.g., staleness bound is 2

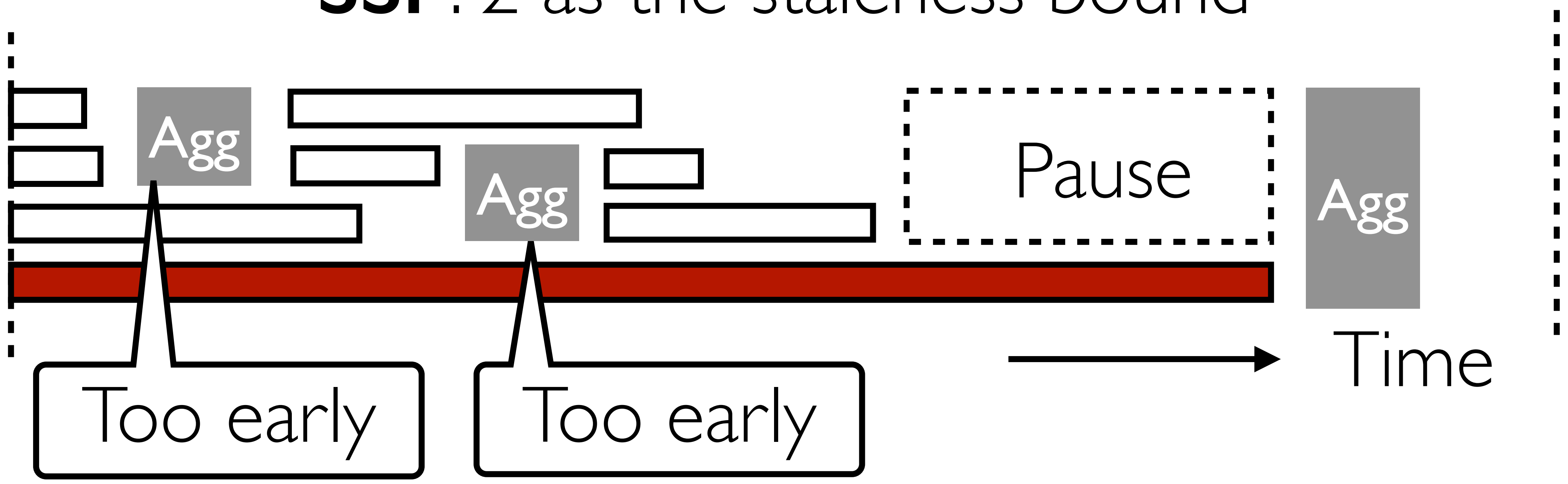
→No more than 2 aggregations behind



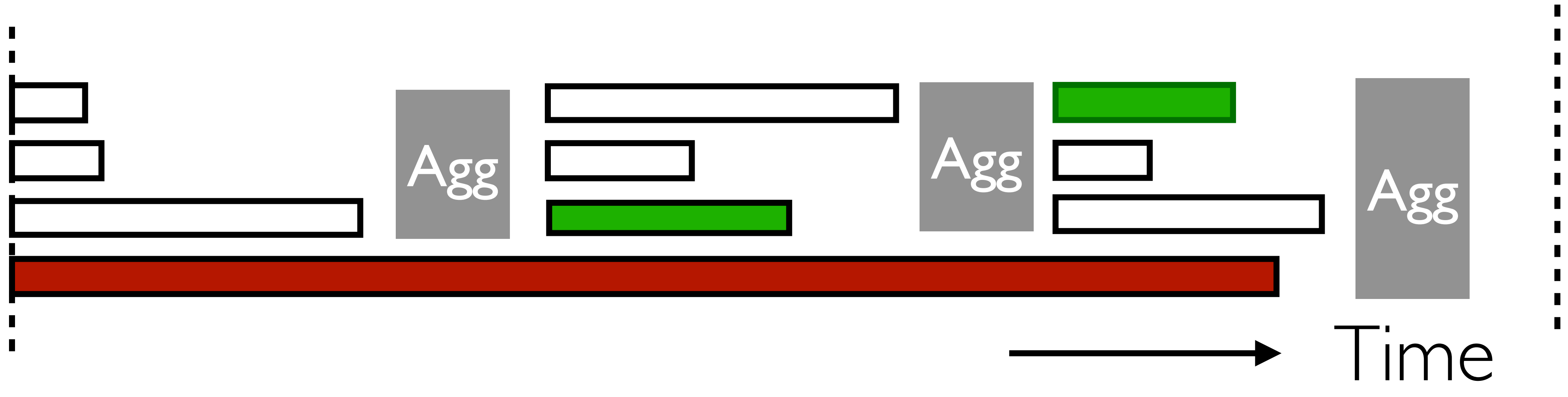

Stale Synchronous Parallel (**SSP**)[1] in traditional ML

**Problem**: Unaware of clients' **speed** and may be **suboptimal** in **client efficiency**

**SSP**: 2 as the staleness bound




**Better case**: **more** contributions




[1]Ho et al. "More Effective Distributed ML via a Stale Synchronous Parallel Parameter Server", In NeurIPS '13

# Problem: Straggler mitigation

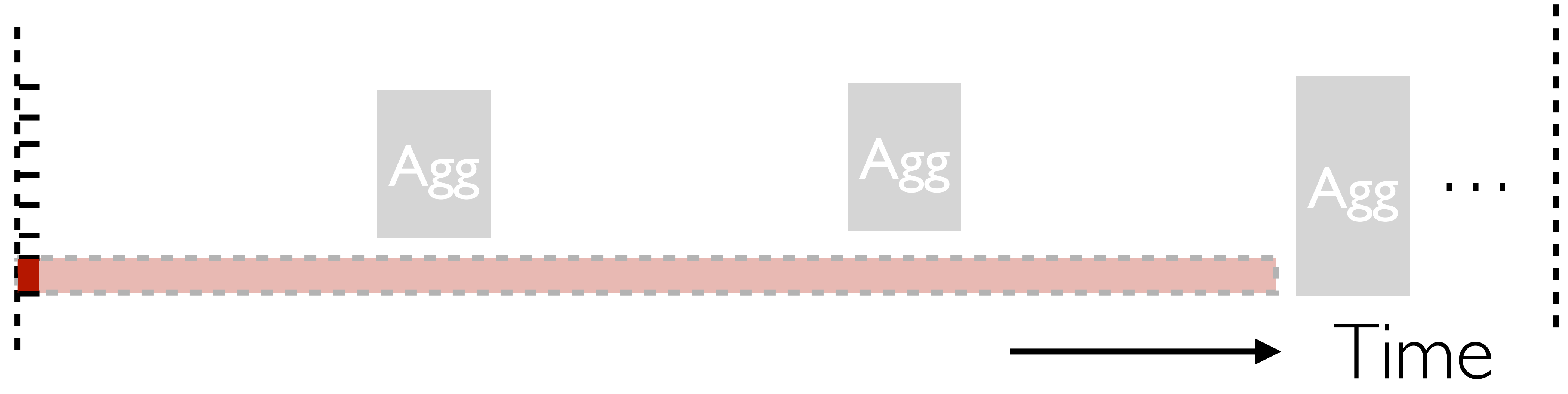
**Solution**: **Speed-aware** aggregation
**pace control** for bounded staleness

# Problem: Straggler mitigation

**Solution**: **Speed-aware** aggregation
**pace control** for bounded staleness

Static point of view

- Interval **evenly distributed**



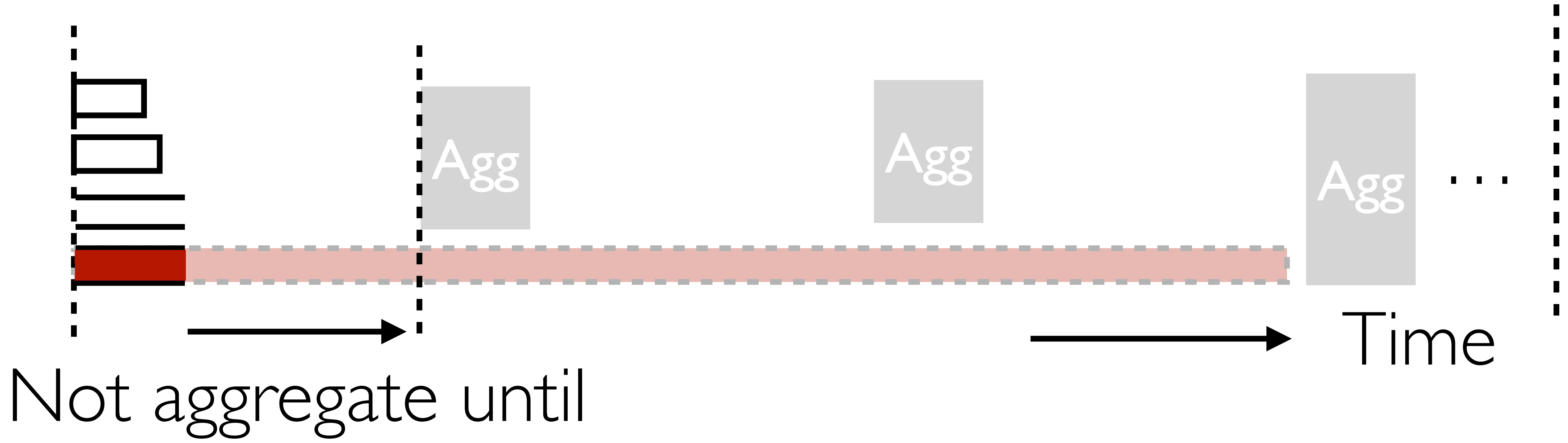

E.g., staleness bound is 2
→No more than 2 aggregation behind

# Problem: Straggler mitigation

**Solution**: **Speed-aware** aggregation
**pace control** for bounded staleness

Static point of view

- Interval **evenly distributed**



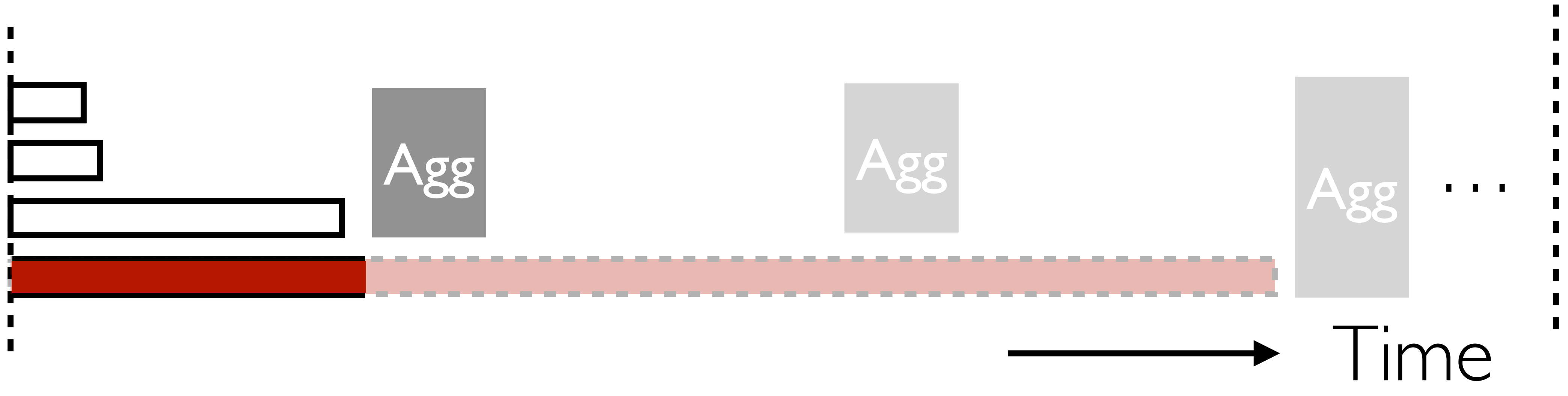

E.g., staleness bound is 2
→No more than 2 aggregation behind

# Problem: Straggler mitigation

**Solution**: **Speed-aware** aggregation
**pace control** for bounded staleness

Static point of view

- Interval **evenly distributed**



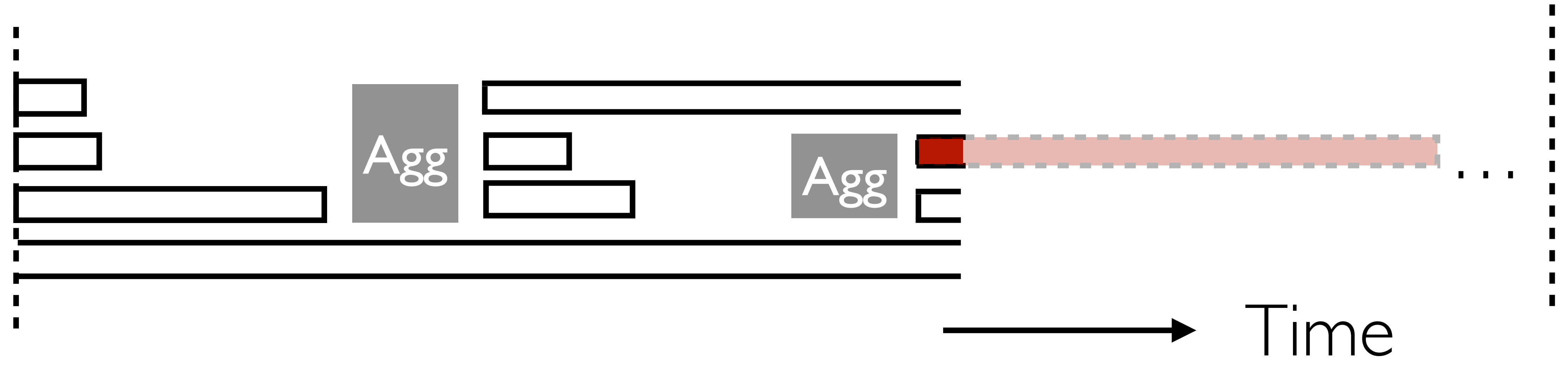

E.g., staleness bound is 2
→No more than 2 aggregation behind

# Problem: Straggler mitigation

**Solution**: **Speed-aware** aggregation
**pace control** for bounded staleness

Static point of view

- Interval **evenly distributed**




Adaptation for dynamics

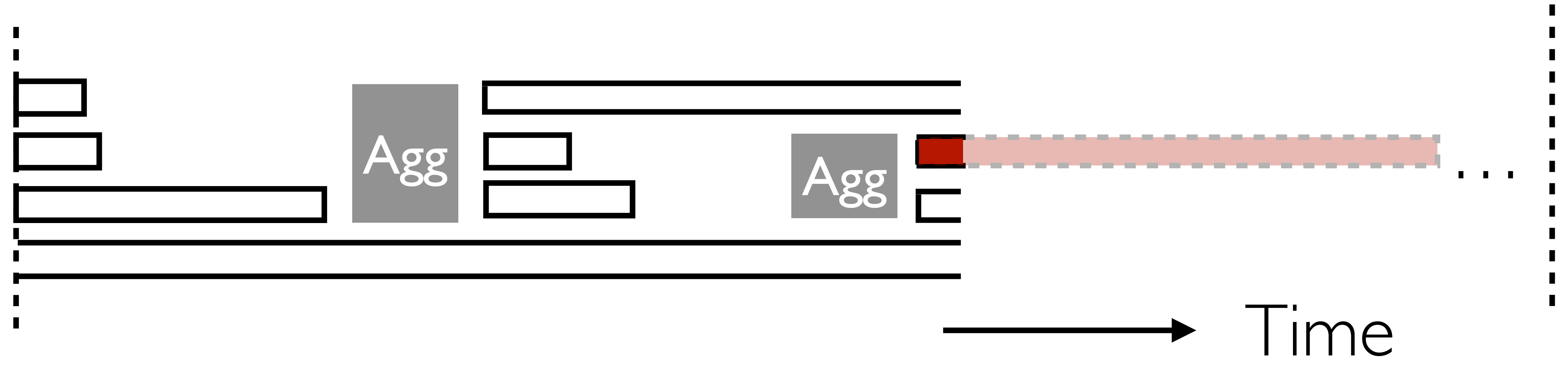
- **Anchored** to the **currently** slowest

# Pisces guarantees convergence

**Solution**: **Speed-aware** aggregation
**pace control** for bounded staleness

Static point of view

- Interval **evenly distributed**




Adaptation for dynamics

- **Anchored** to the **currently** slowest

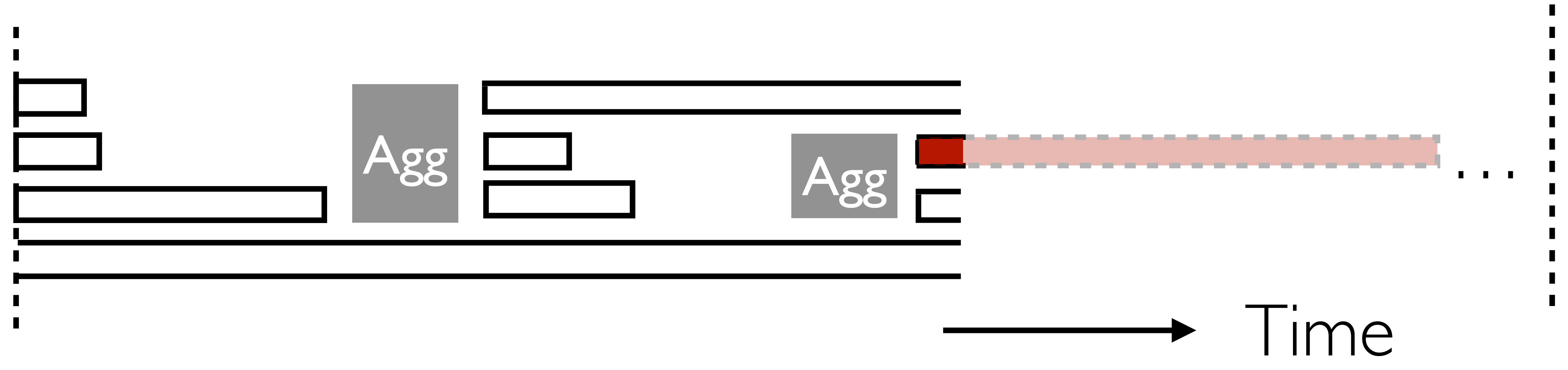
Not only have higher client efficiency
But also achieve **bounded staleness**

# Pisces guarantees convergence

**Solution**: **Speed-aware** aggregation
**pace control** for bounded staleness

Static point of view

- Interval **evenly distributed**




Adaptation for dynamics

- **Anchored** to the **currently** slowest

Not only have higher client efficiency
But also achieve **bounded staleness**

Further guarantee **convergence:**

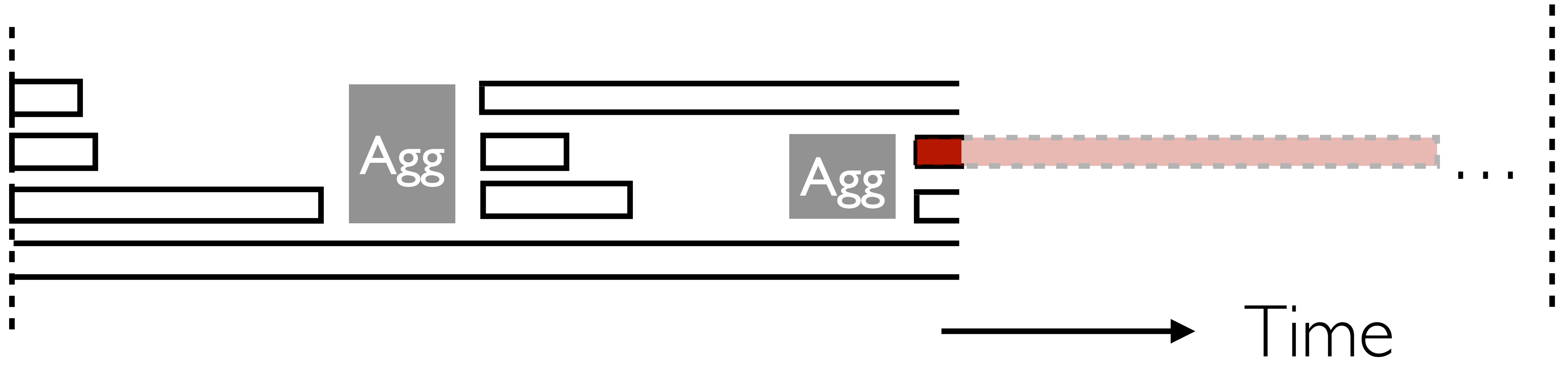
- At a rate **slightly slower than O(1/T)** (T: # rounds)

# Pisces guarantees convergence

**Solution**: **Speed-aware** aggregation **pace control** for bounded staleness

Static point of view

- Interval **evenly distributed**




Adaptation for dynamics

- **Anchored** to the **currently** slowest

Not only have higher client efficiency

But also achieve **bounded staleness**

↓

Further guarantee **convergence:**

- At a rate **slightly slower than O(1/T)** (T: # rounds)

Other designs on efficiency/robustness…

Please find more in the paper :)

# Pisces outperforms in time-to-accuracy

[1] Lai et al. "Oort: Efficient Federated Learning via Guided Participant Selection", In OSDI '21

# Pisces outperforms in time-to-accuracy

**Oort**[1] → State-of-the-art **synchronous** method: navigating the **speed-data tradeoff**

[1]Lai et al. "Oort: Efficient Federated Learning via Guided Participant Selection", In OSDI '21

# Pisces outperforms in time-to-accuracy

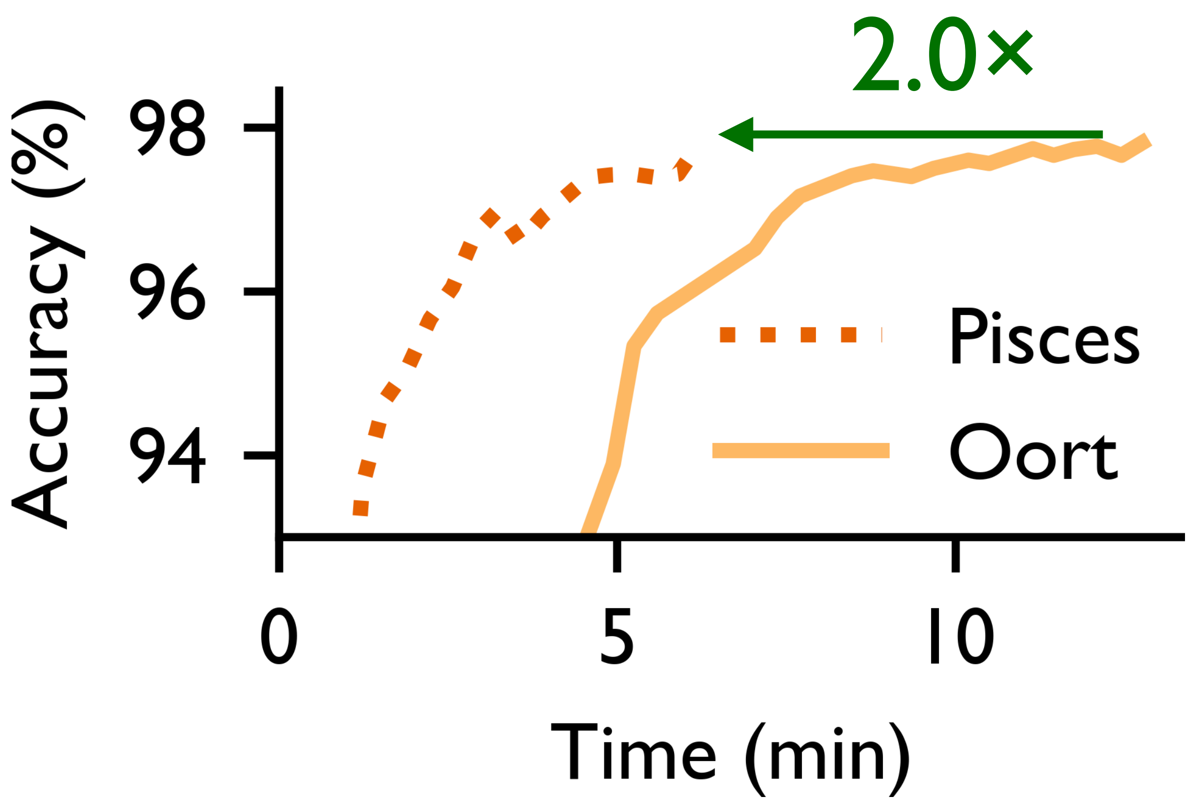
**Oort**[1] → State-of-the-art **synchronous** method: navigating the **speed-data tradeoff**

**MNIST@LeNet5** **FEMNIST@LeNet5** **CIFAR10@ResNet18** **Reddit@Albert**

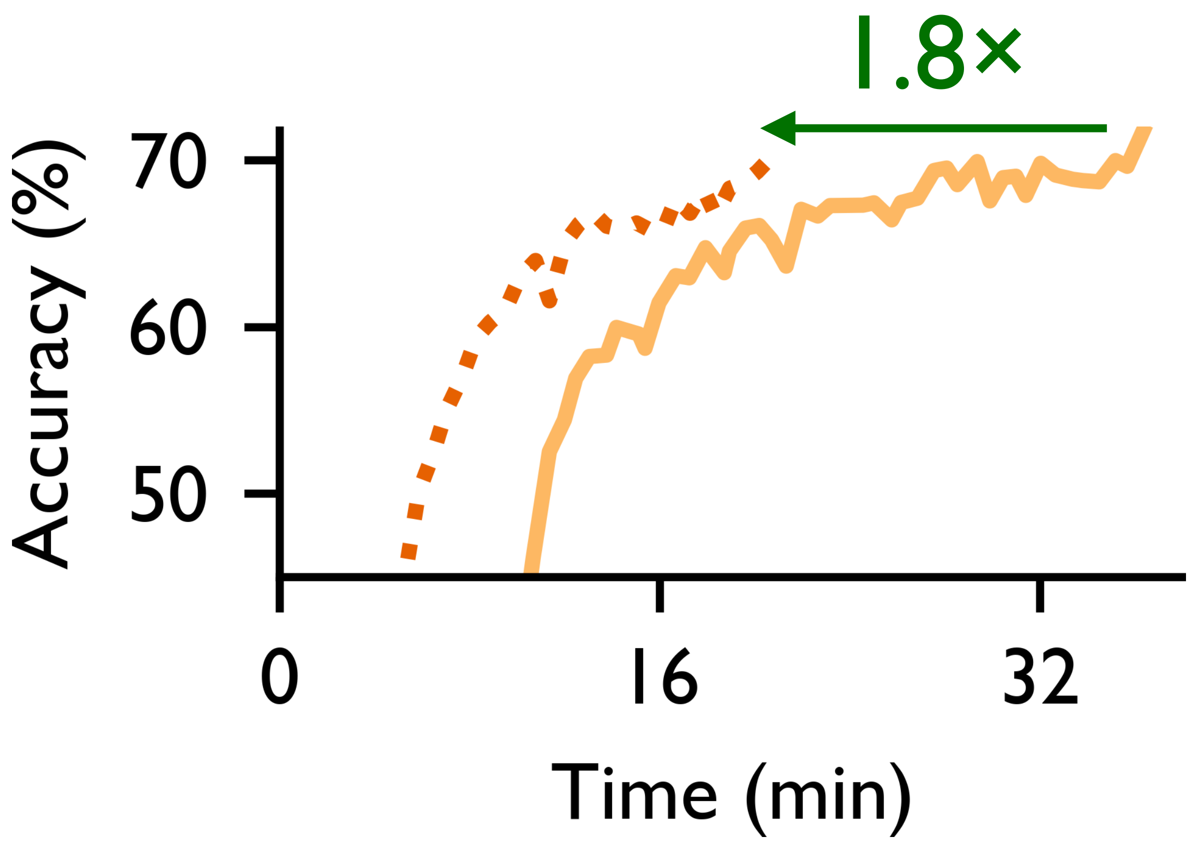
[1] Lai et al. "Oort: Efficient Federated Learning via Guided Participant Selection", In OSDI '21

# Pisces outperforms in time-to-accuracy

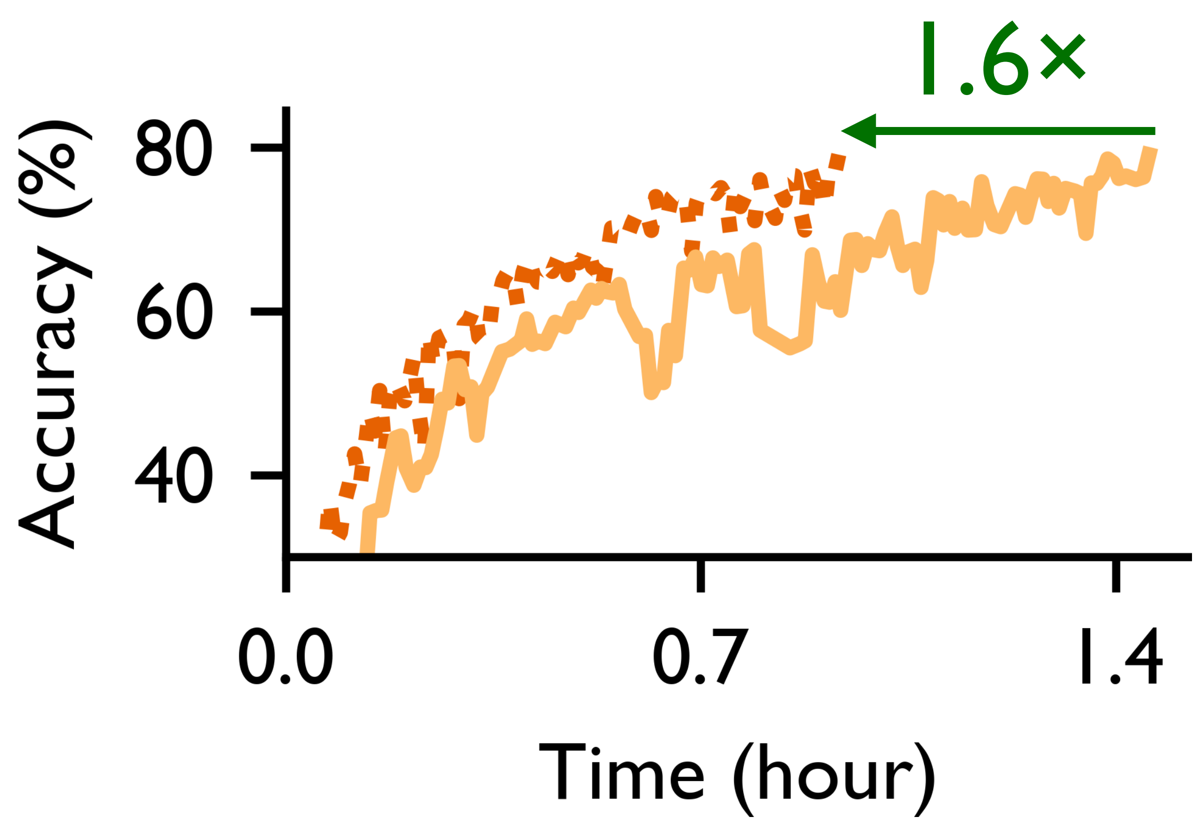
**Oort**[1] → State-of-the-art **synchronous** method: navigating the **speed-data tradeoff**



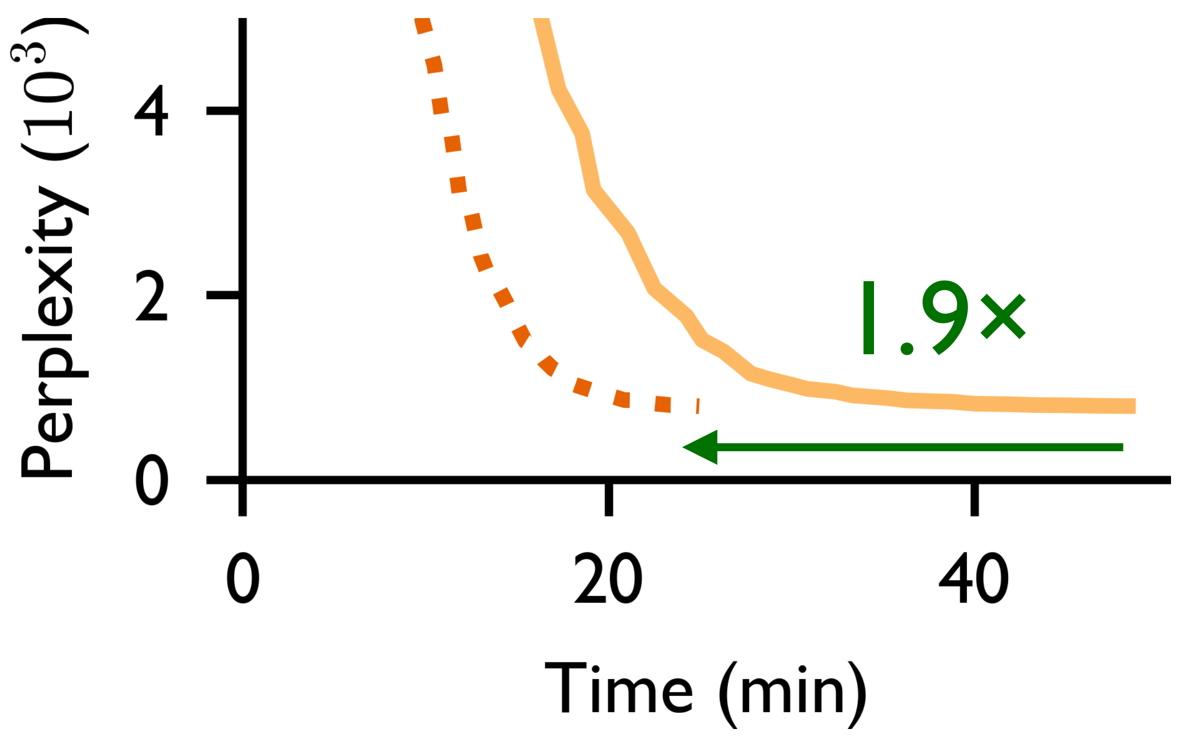

**MNIST@LeNet5**




**FEMNIST@LeNet5**




**CIFAR10@ResNet18**

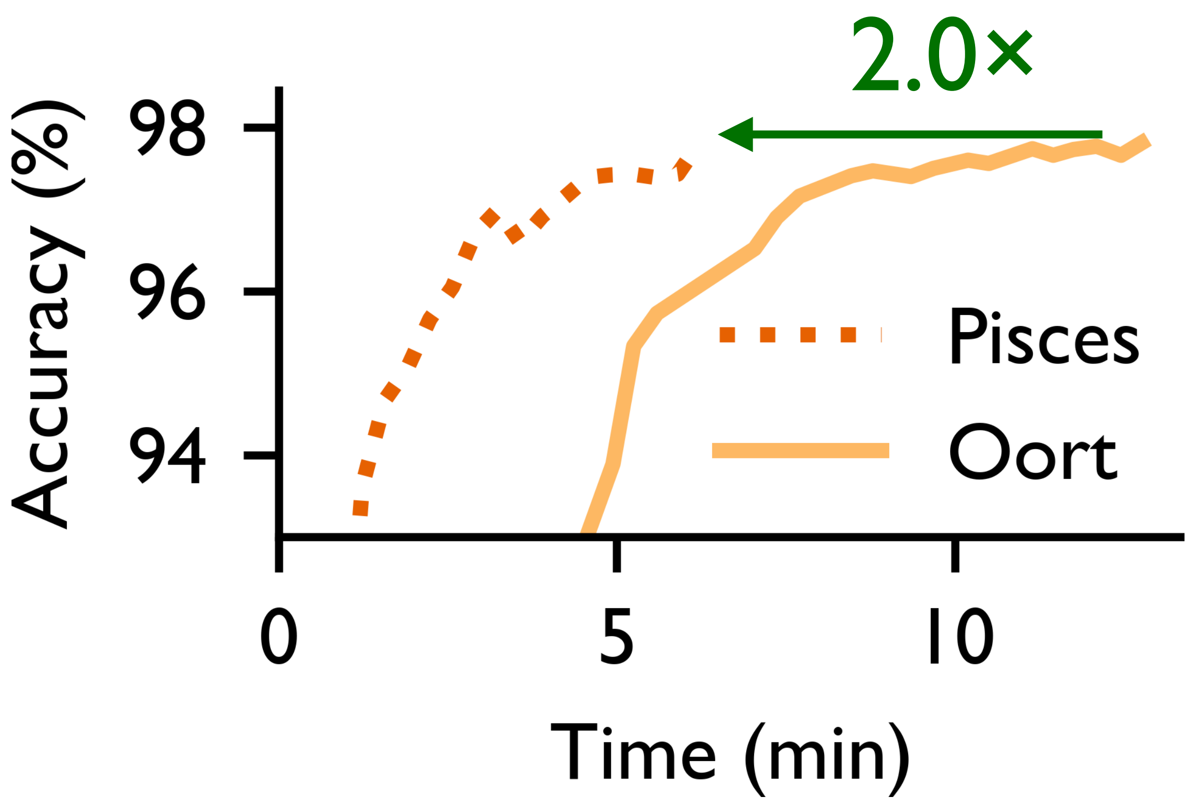



**Reddit@Albert**

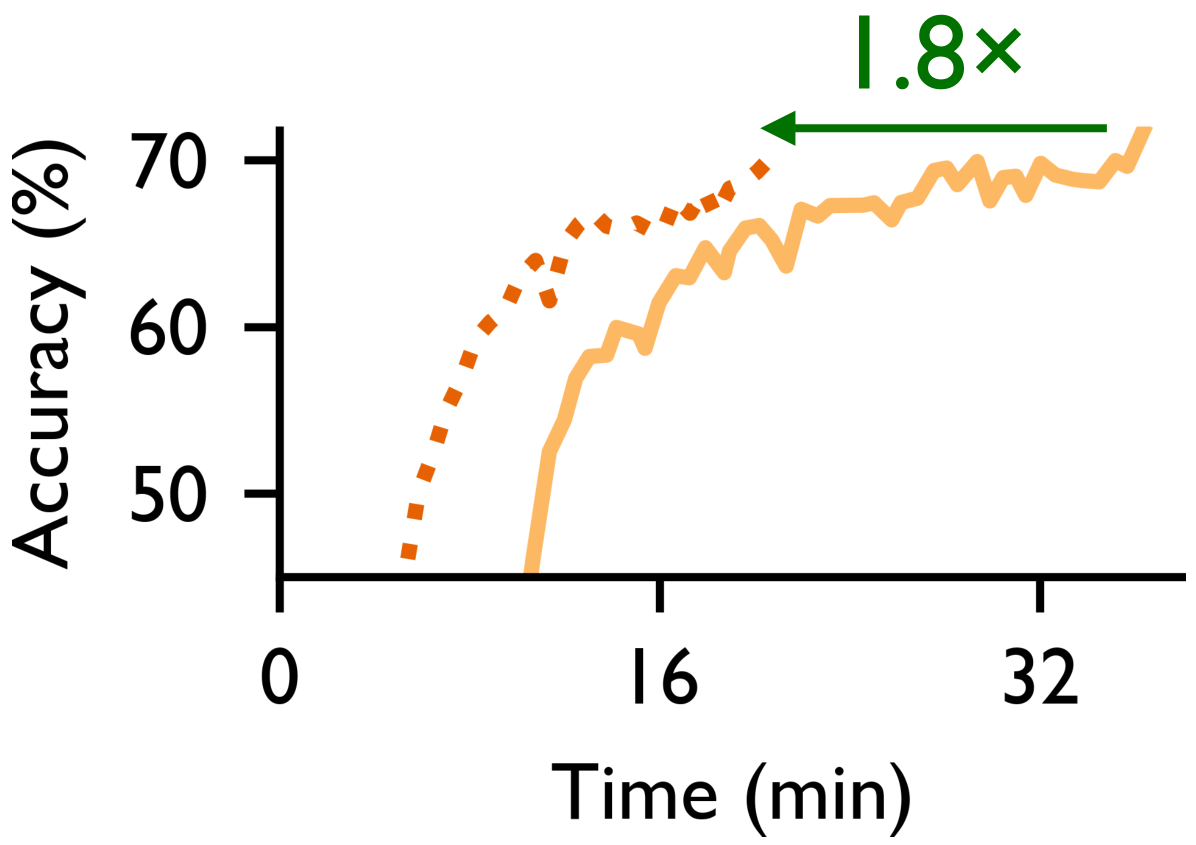
[1] Lai et al. "Oort: Efficient Federated Learning via Guided Participant Selection", In OSDI '21

# Pisces outperforms in time-to-accuracy

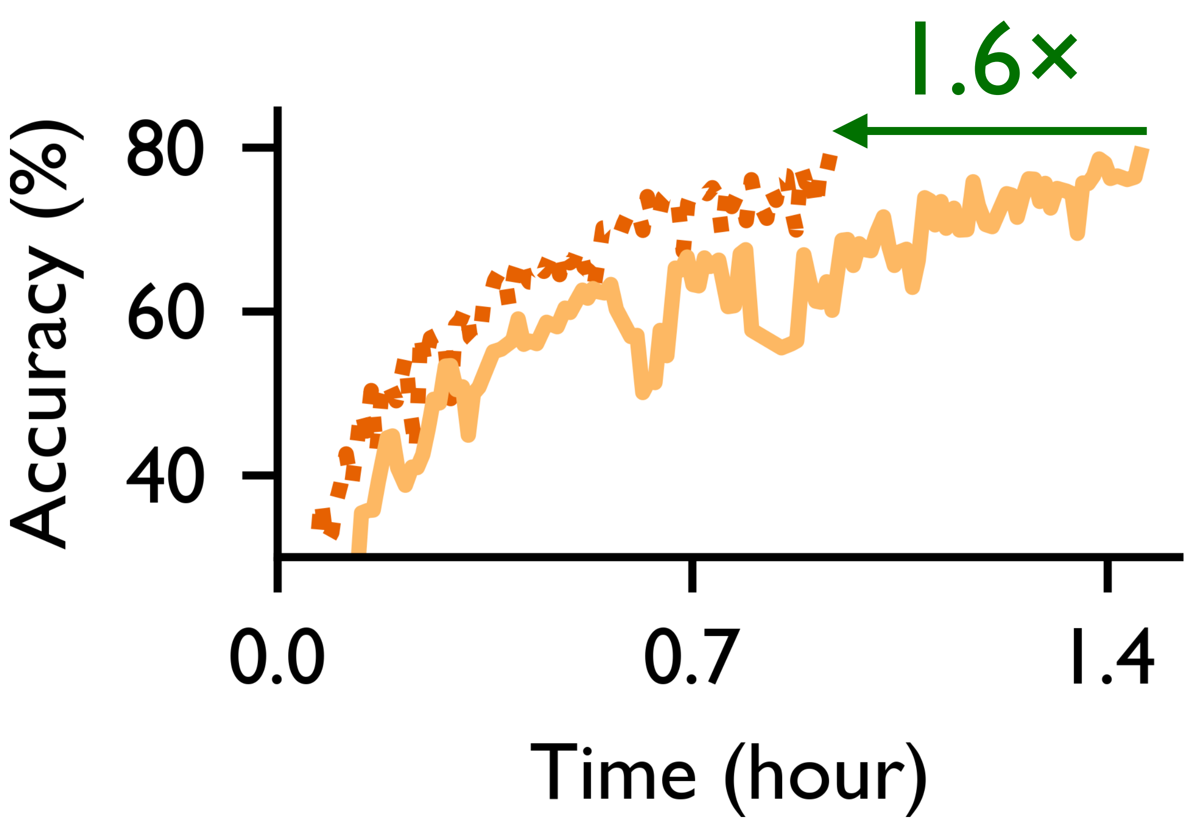
**Oort**[1] → State-of-the-art **synchronous** method: navigating the **speed-data tradeoff**



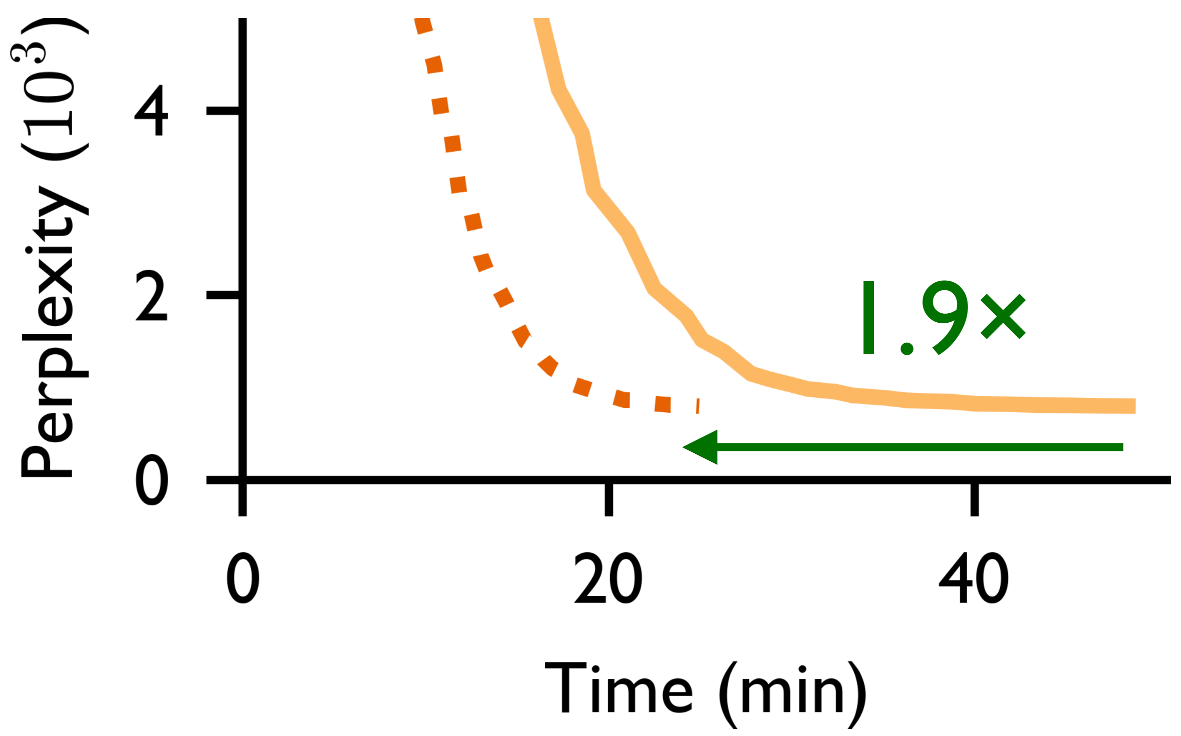

MNIST@LeNet5



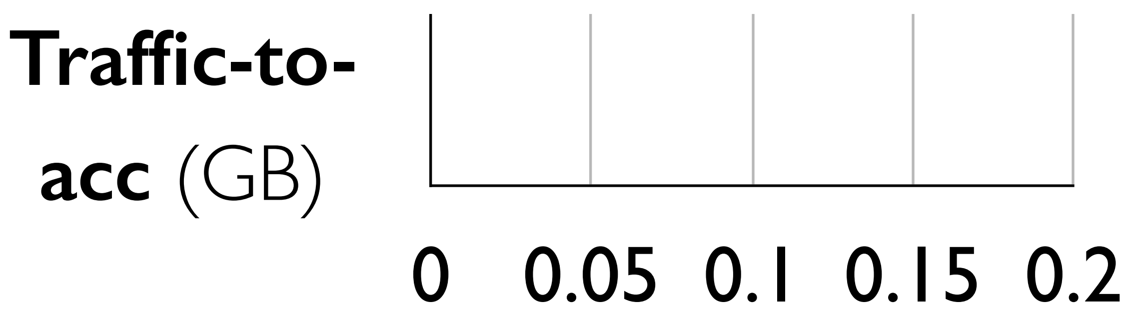

FEMNIST@LeNet5

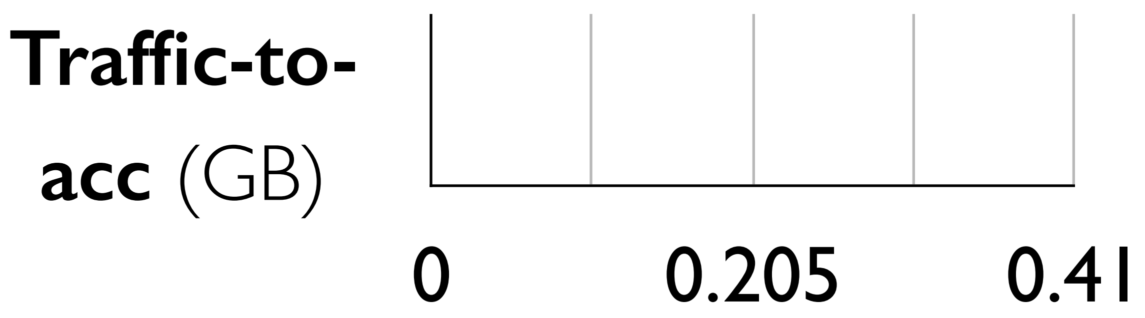



CIFAR10@ResNet18

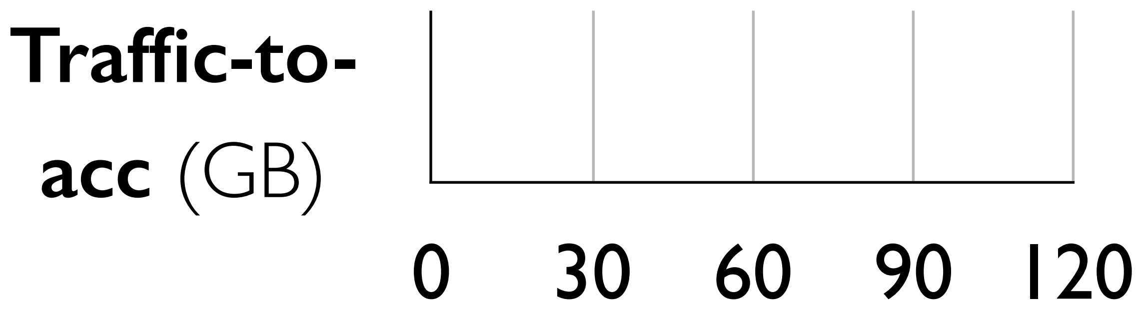



Reddit@Albert








[1] Lai et al. "Oort: Efficient Federated Learning via Guided Participant Selection", In OSDI '21

# Pisces outperforms in time-to-accuracy

**Oort**[1] → State-of-the-art **synchronous** method: navigating the **speed-data tradeoff**


**Pisces accelerates** Oort by up to **2×**
**without** significant **network** cost

[1]Lai et al. "Oort: Efficient Federated Learning via Guided Participant Selection", In OSDI '21

# Pisces: Results summary

**Theory**

**Provable convergence** for smooth non-convex problems based on **bounded staleness**

**Efficiency**

**2.0×** improvement in **time-to-accuracy** with **no network** overhead

**Practicality**

Easily integrated to **production frameworks** like Plato

github.com/SamuelGong/Pisces

# Second work: Dordis[1]

| | | | |
|---|---|---|---|
| **Privacy**<br>Worst-case defense… | Can be a dishonest majority | x ⟷ | Only or mostly works with honest participants |
| **Efficiency**<br>Time-to-accuracy… | Stragglers bottleneck time | Primitives heavy in comp. and comm. | Client dropout yields insufficient noise |

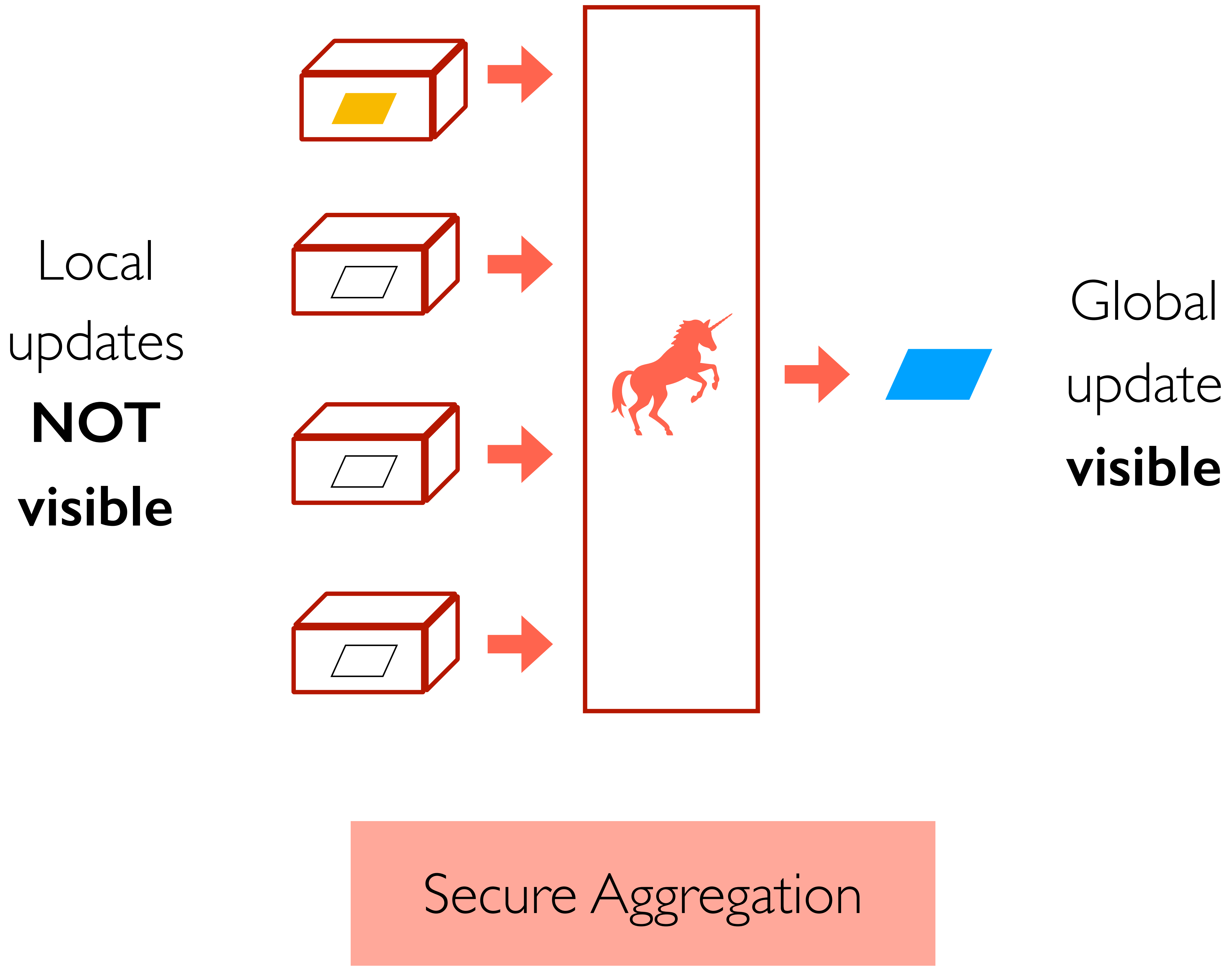
| | | | |
|---|---|---|---|
| **Privacy-Enhancing Technique** | Federated Learning | Secure Aggregation | Differential Privacy |
| **Privacy Guarantee** | Data kept on premises | Local updates unseen | Global update leaks little about any client |

[1] Jiang et al. "Dordis: Efficient Federated Learning with Dropout-Resilient Differential Privacy", In EuroSys '24

# Need for Dordis - 1

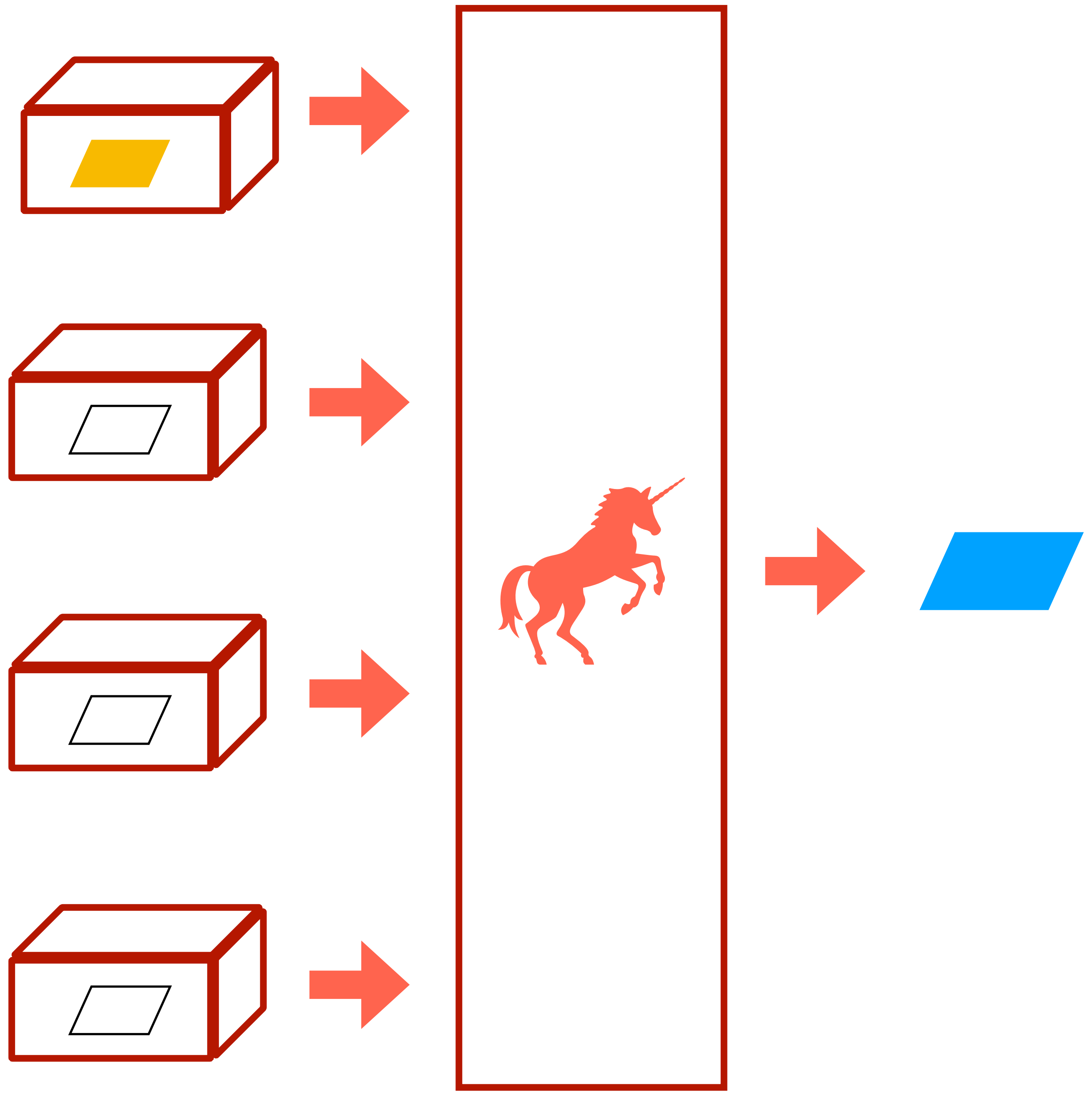
Secure Aggregation

# Need for Dordis - 1



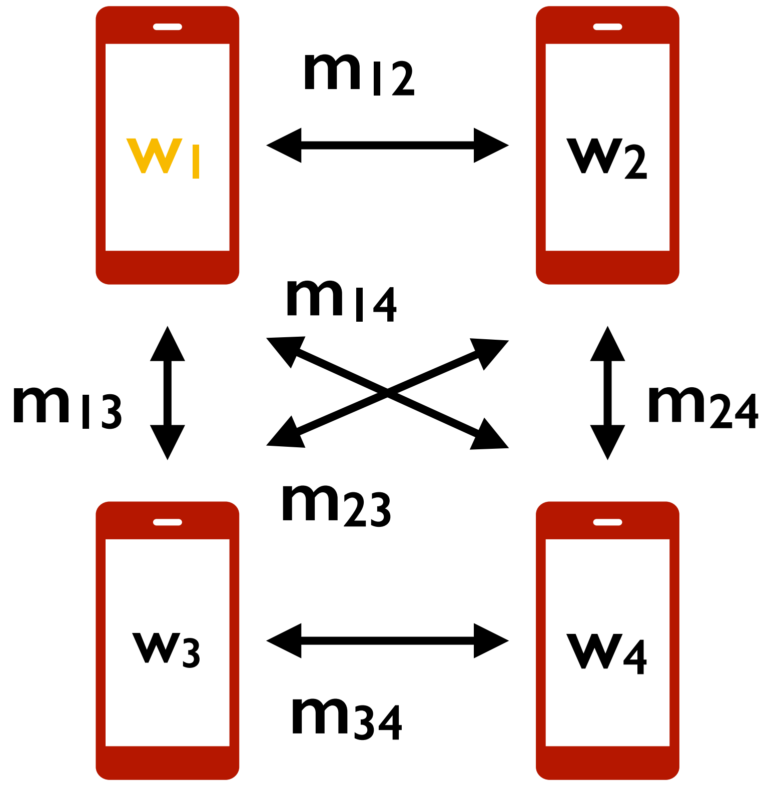
# Need for Dordis - 1

Local
updates
**NOT**
**visible**



Global
update
**visible**

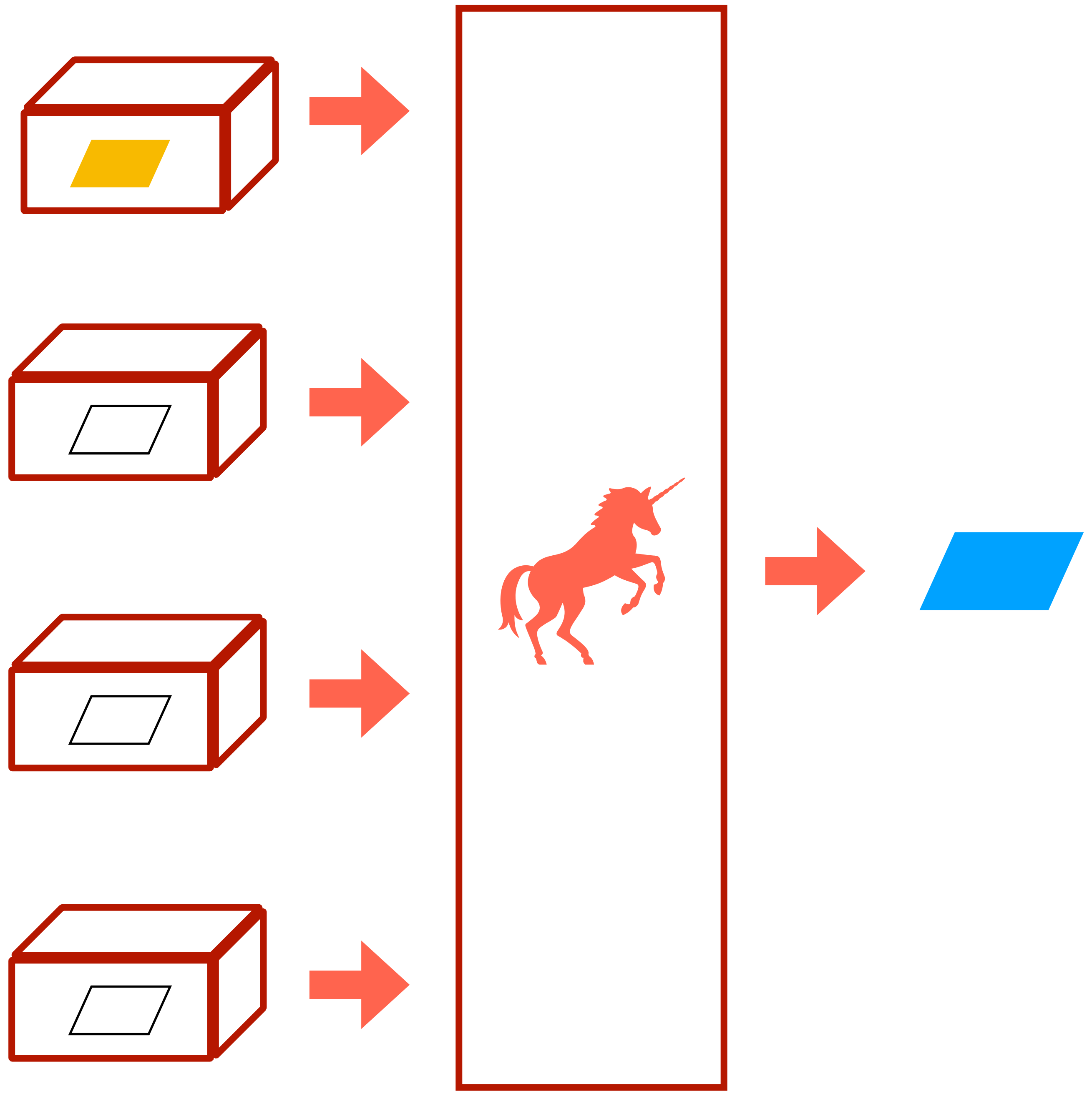
Secure Aggregation




1. **Pairwise** agreement

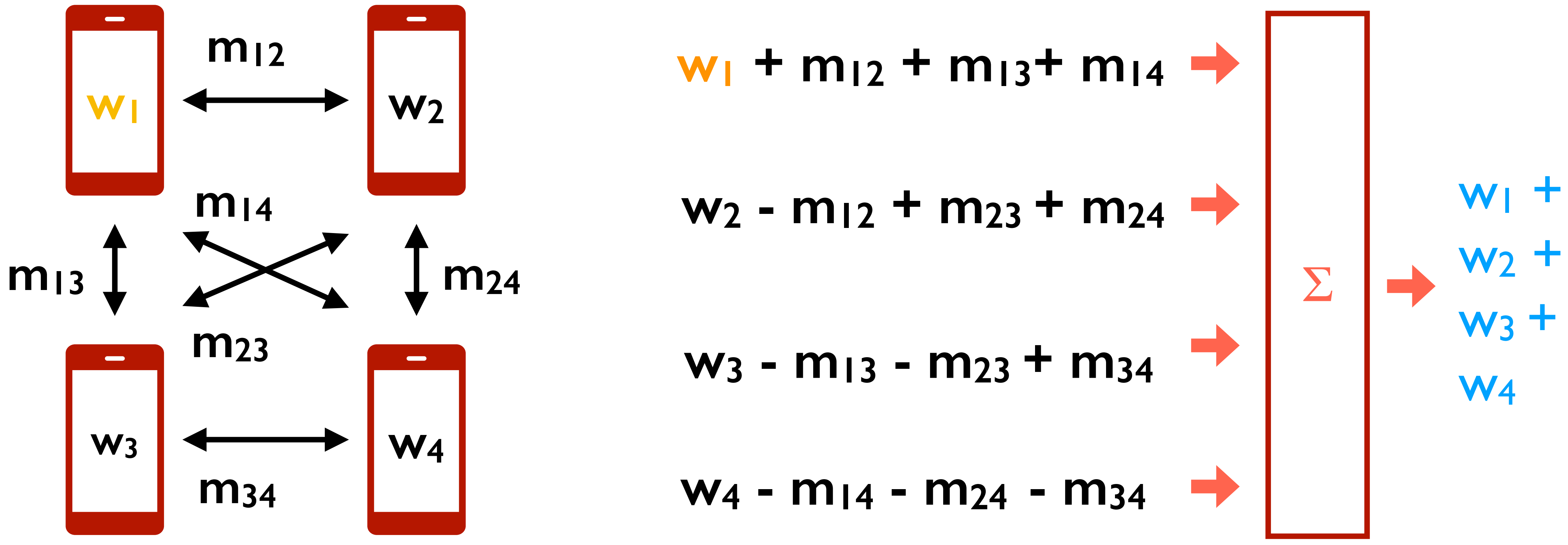
# Need for Dordis - 1

Local updates **NOT visible**



Global update **visible**

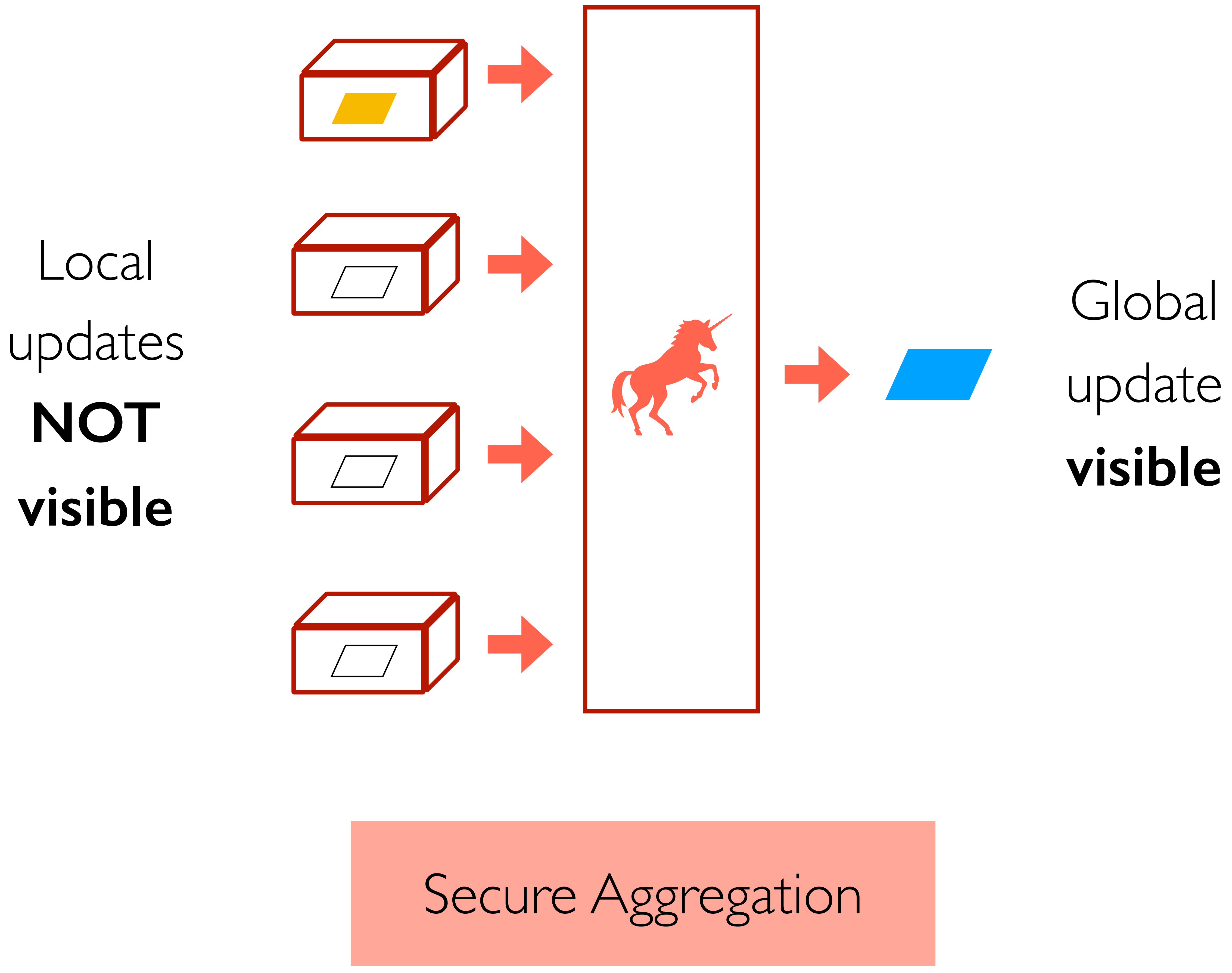
Secure Aggregation

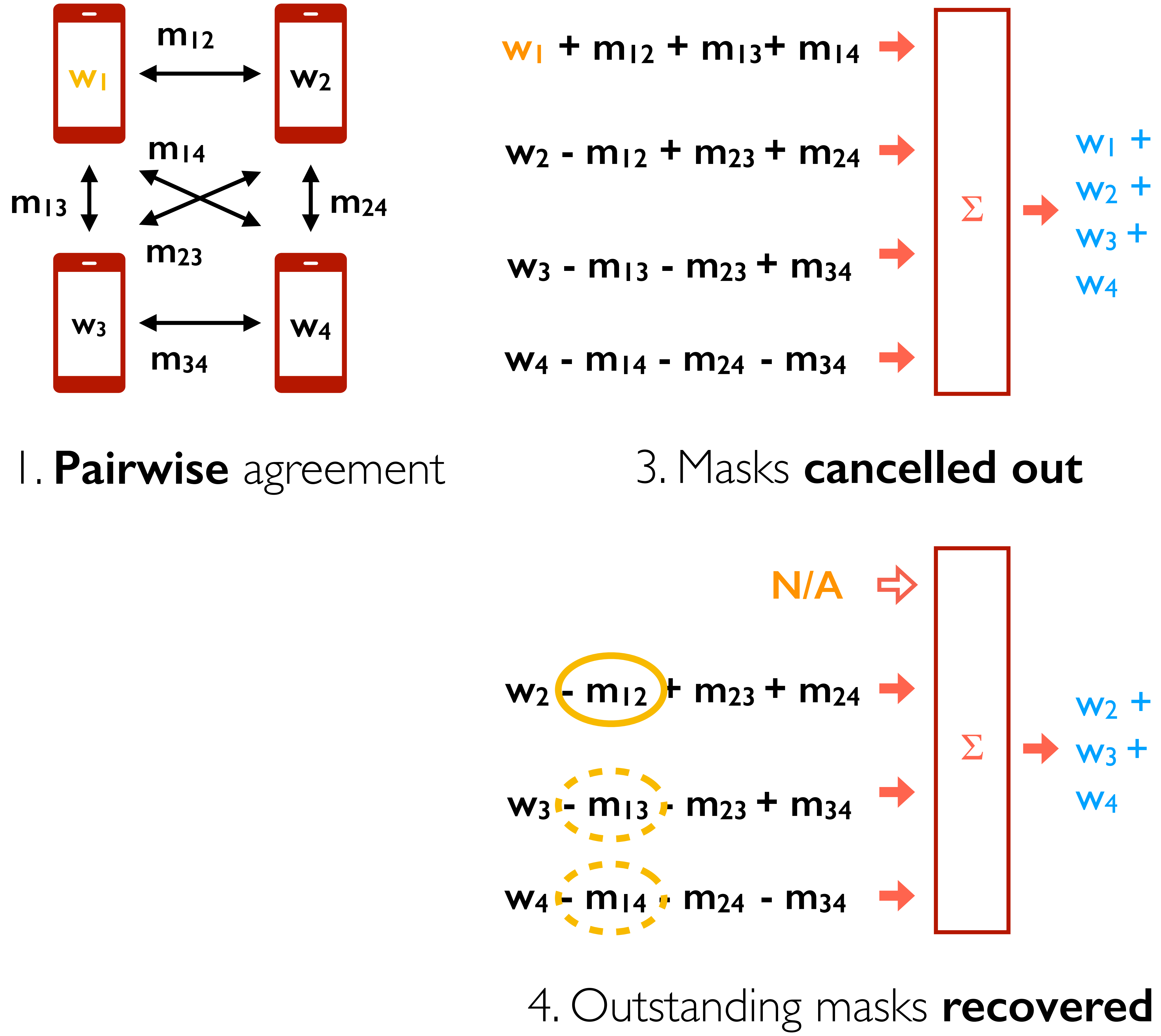



1. **Pairwise** agreement

3. Masks **cancelled out**

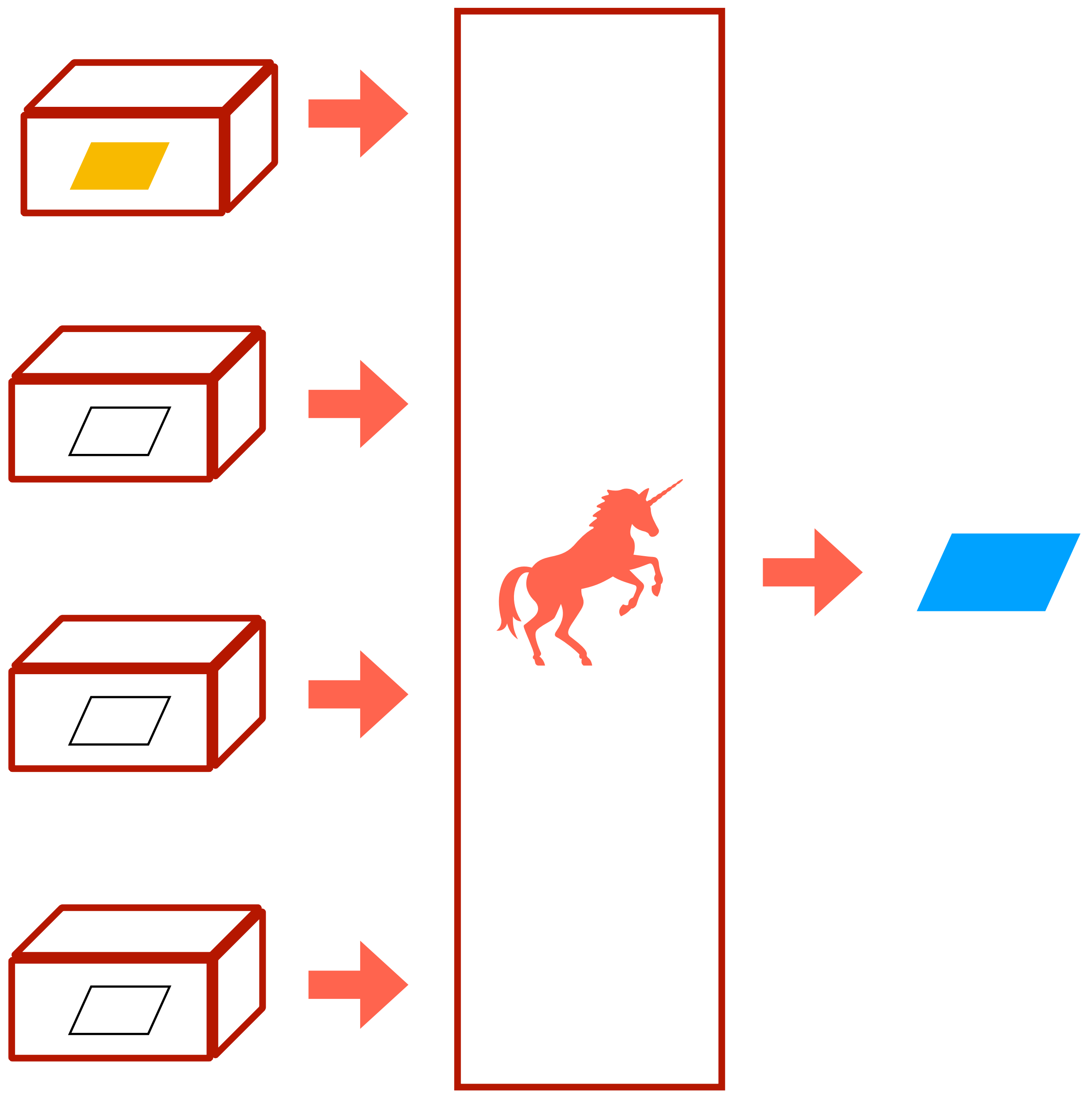
# Need for Dordis - 1



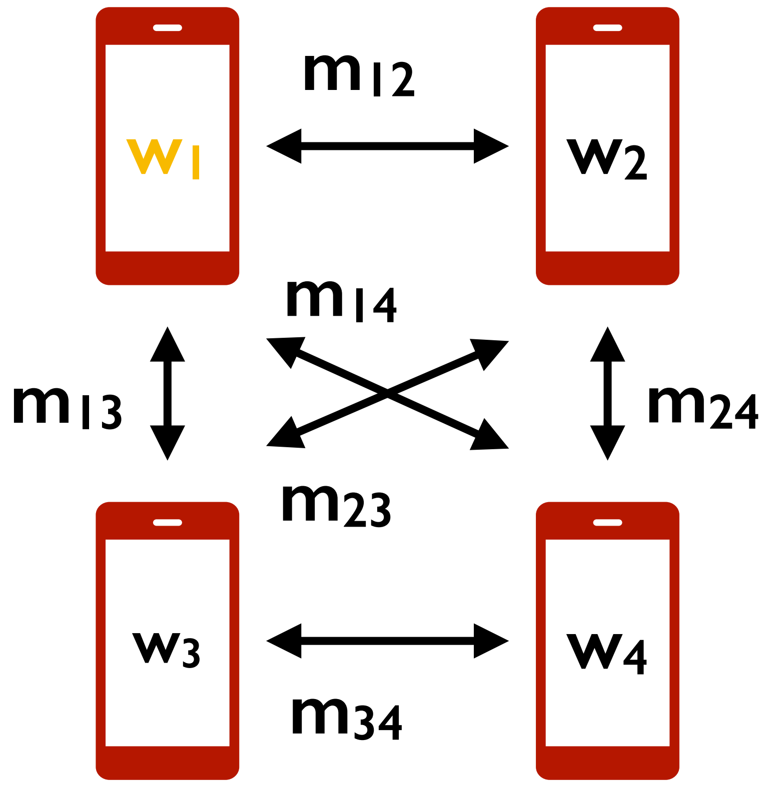
# Need for Dordis - 1

Local updates **NOT visible**

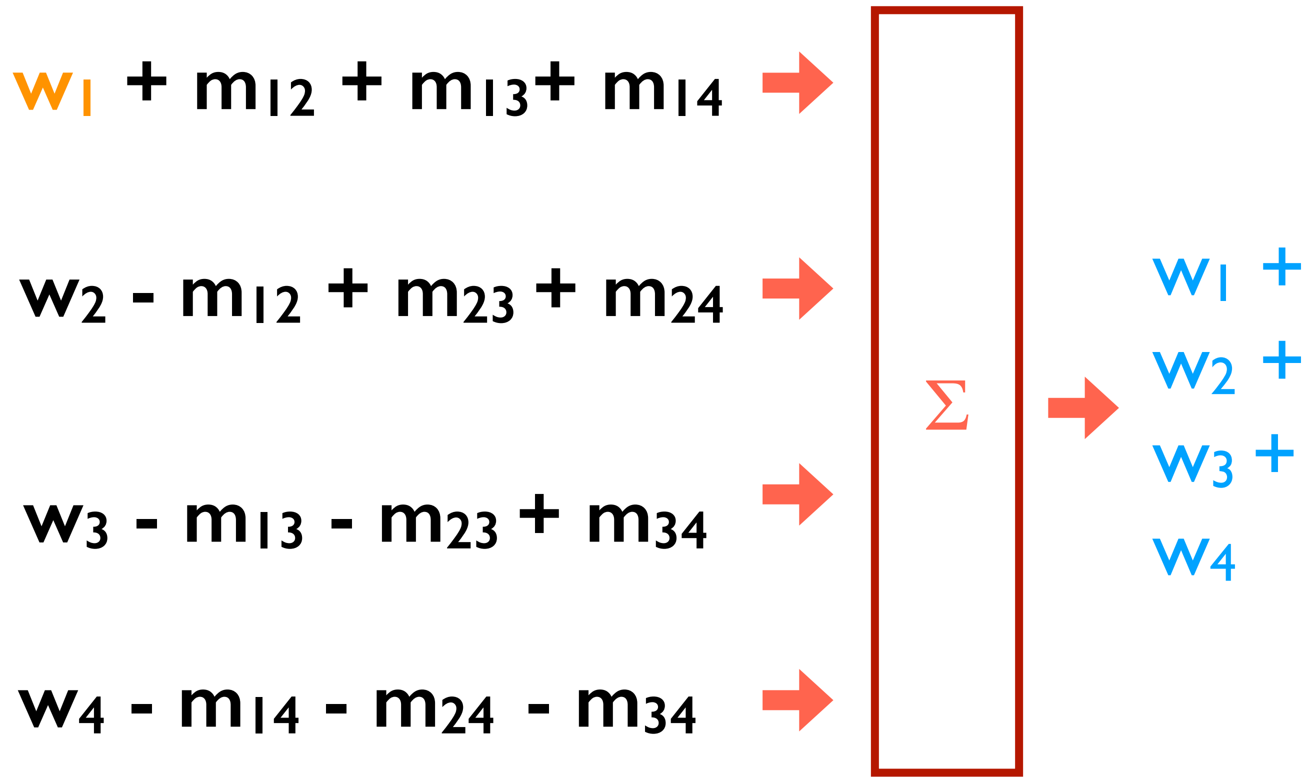


Global update **visible**

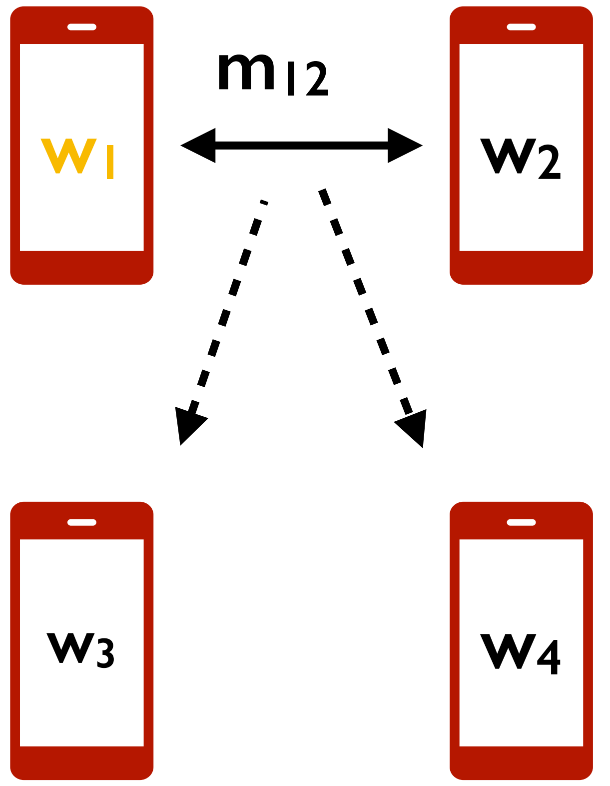
Secure Aggregation



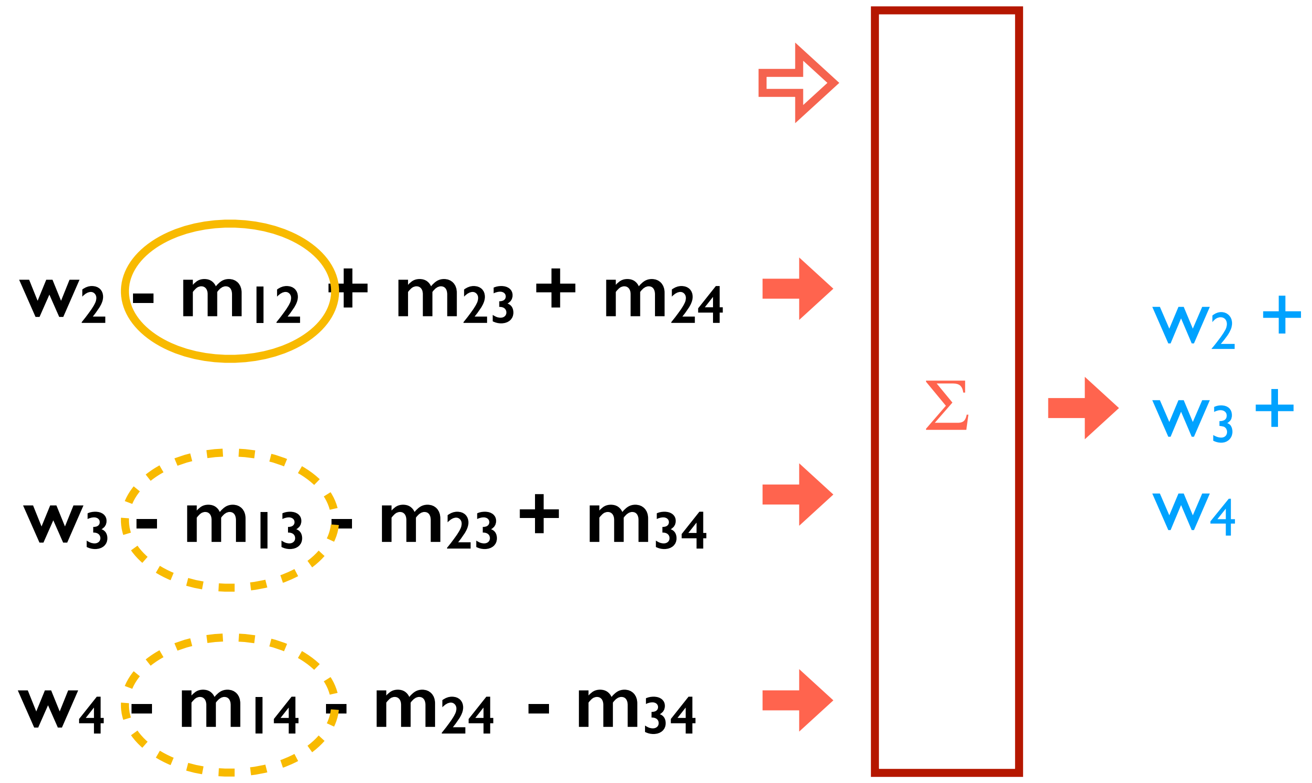

1. **Pairwise** agreement




3. Masks **cancelled out**



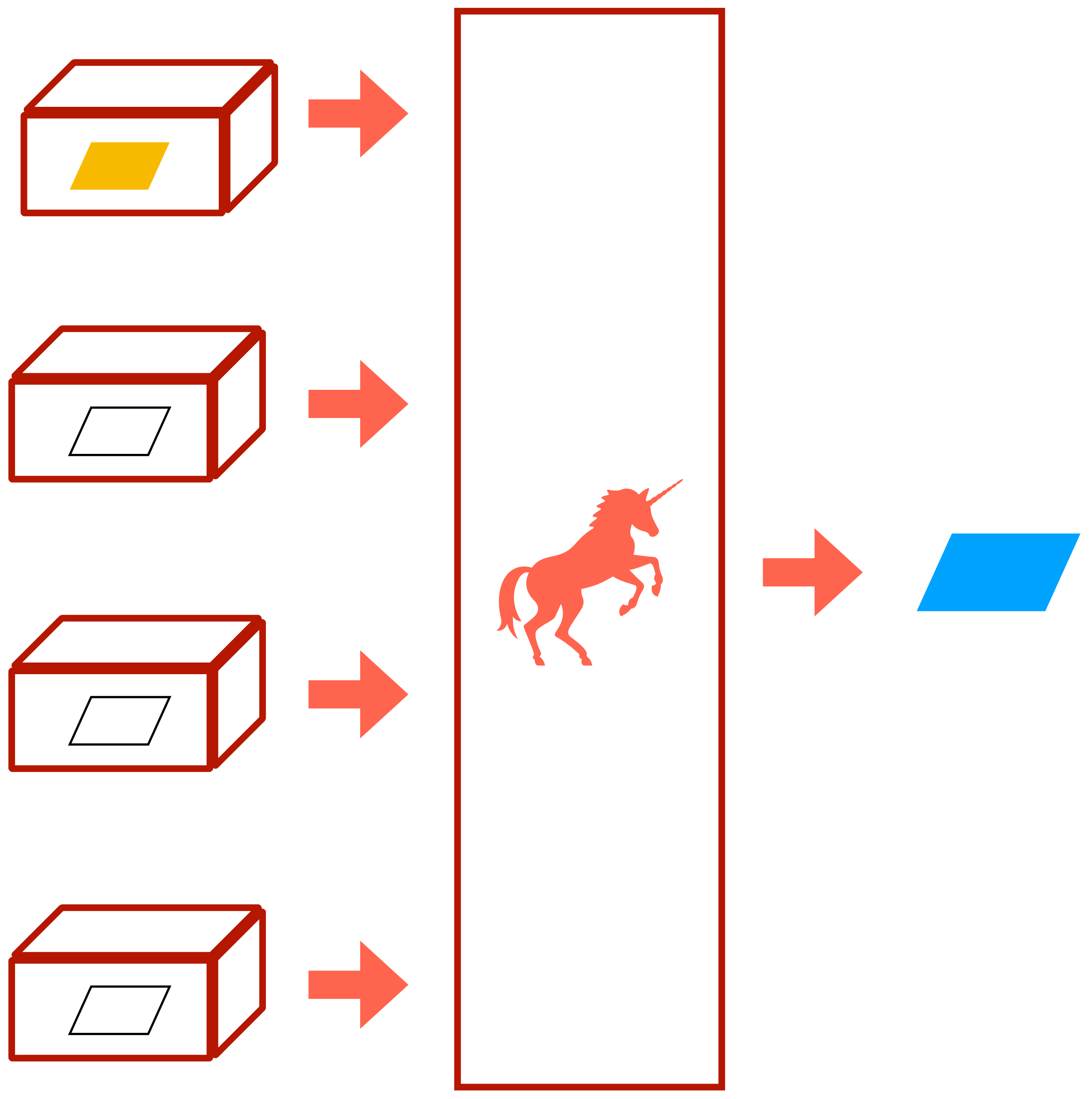

2. Masks **backed up**




4. Outstanding masks **recovered**

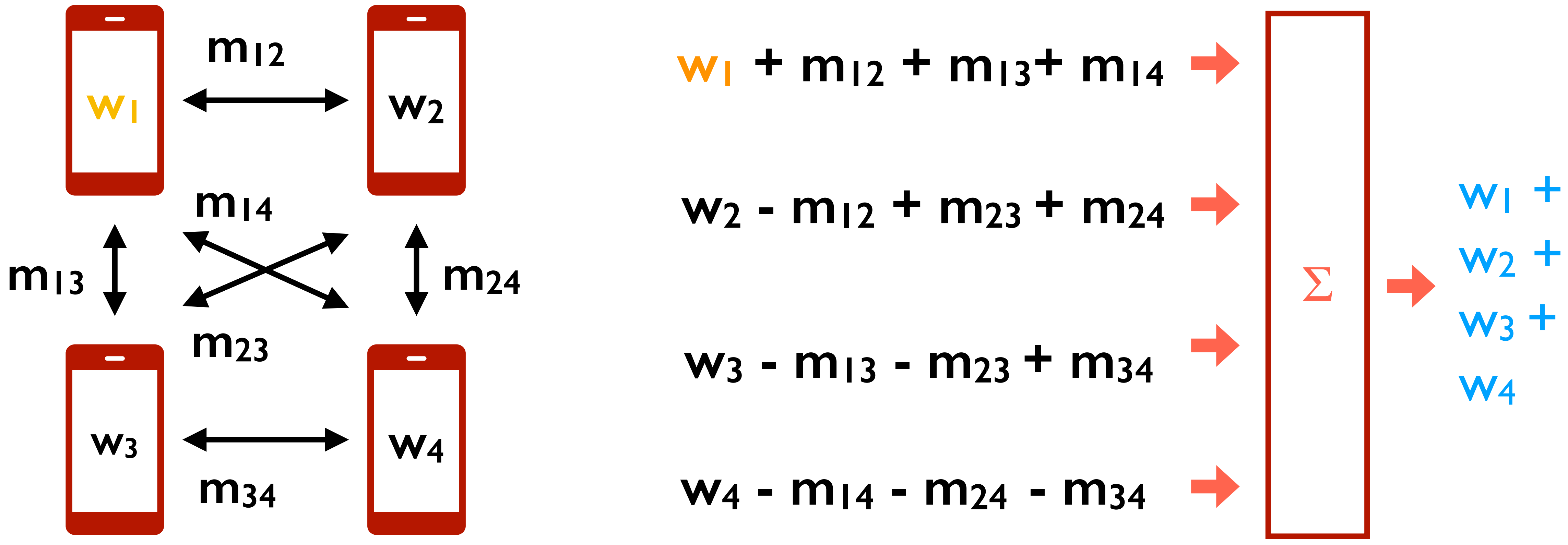
# Need for Dordis - 1

Local updates **NOT visible**

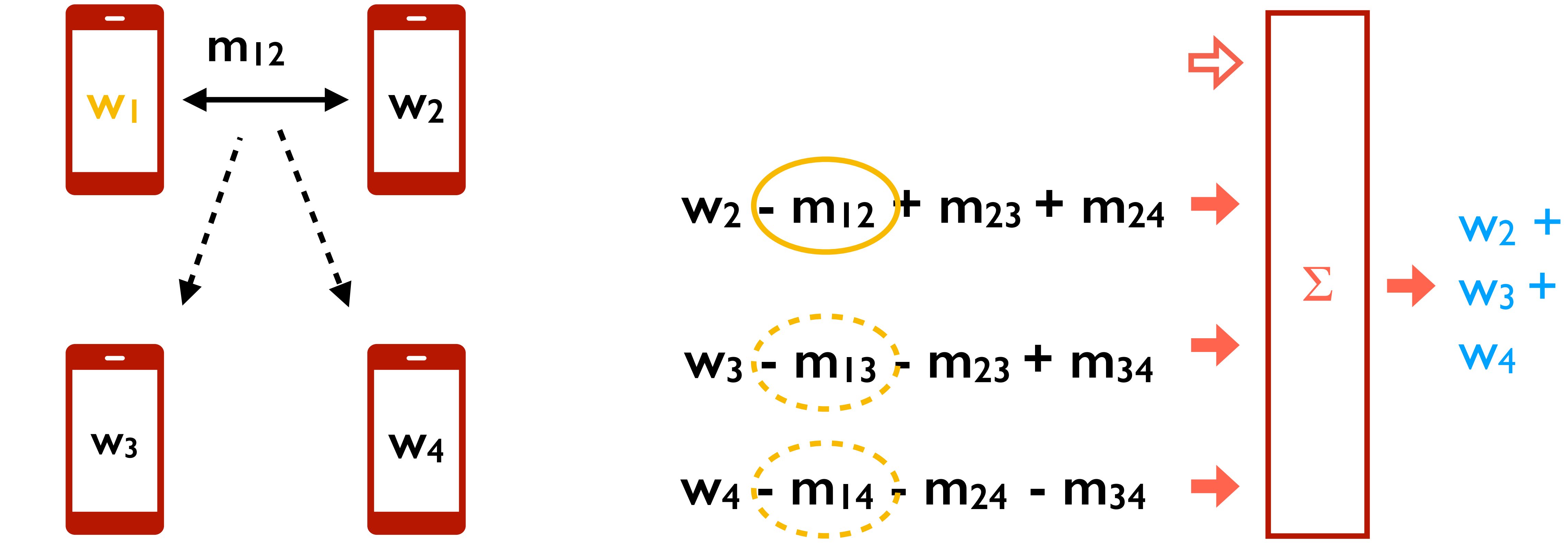


Global update **visible**

Secure Aggregation




Primitive 1: **Pairwise masking** (all-to-all)




Primitive 2: **Secret sharing** (all-to-all)

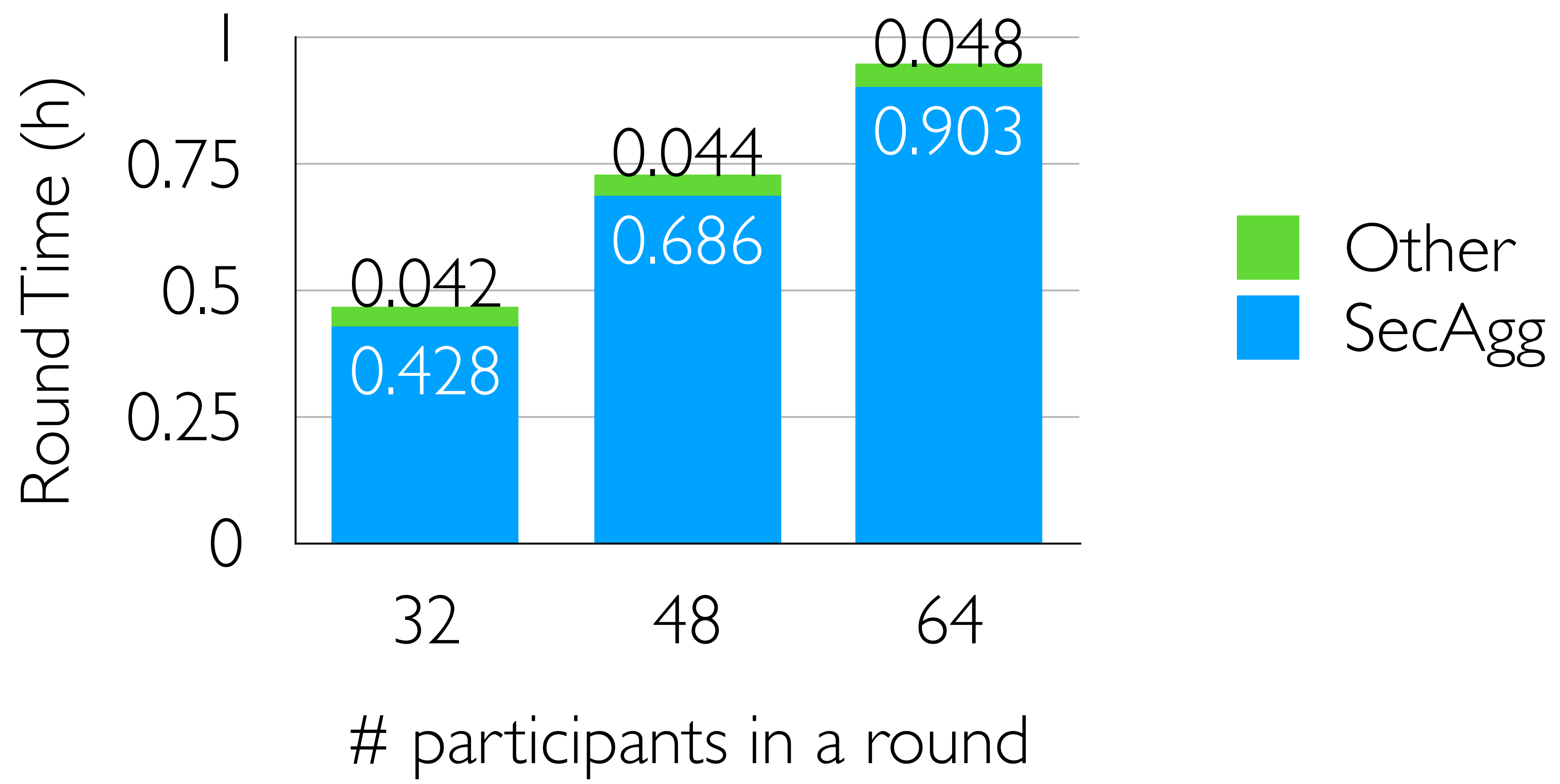
# Need for Dordis - 1

**Problem**: Pairwise masking and secret sharing are necessary but **expensive**

Secure Aggregation

# Need for Dordis - 1

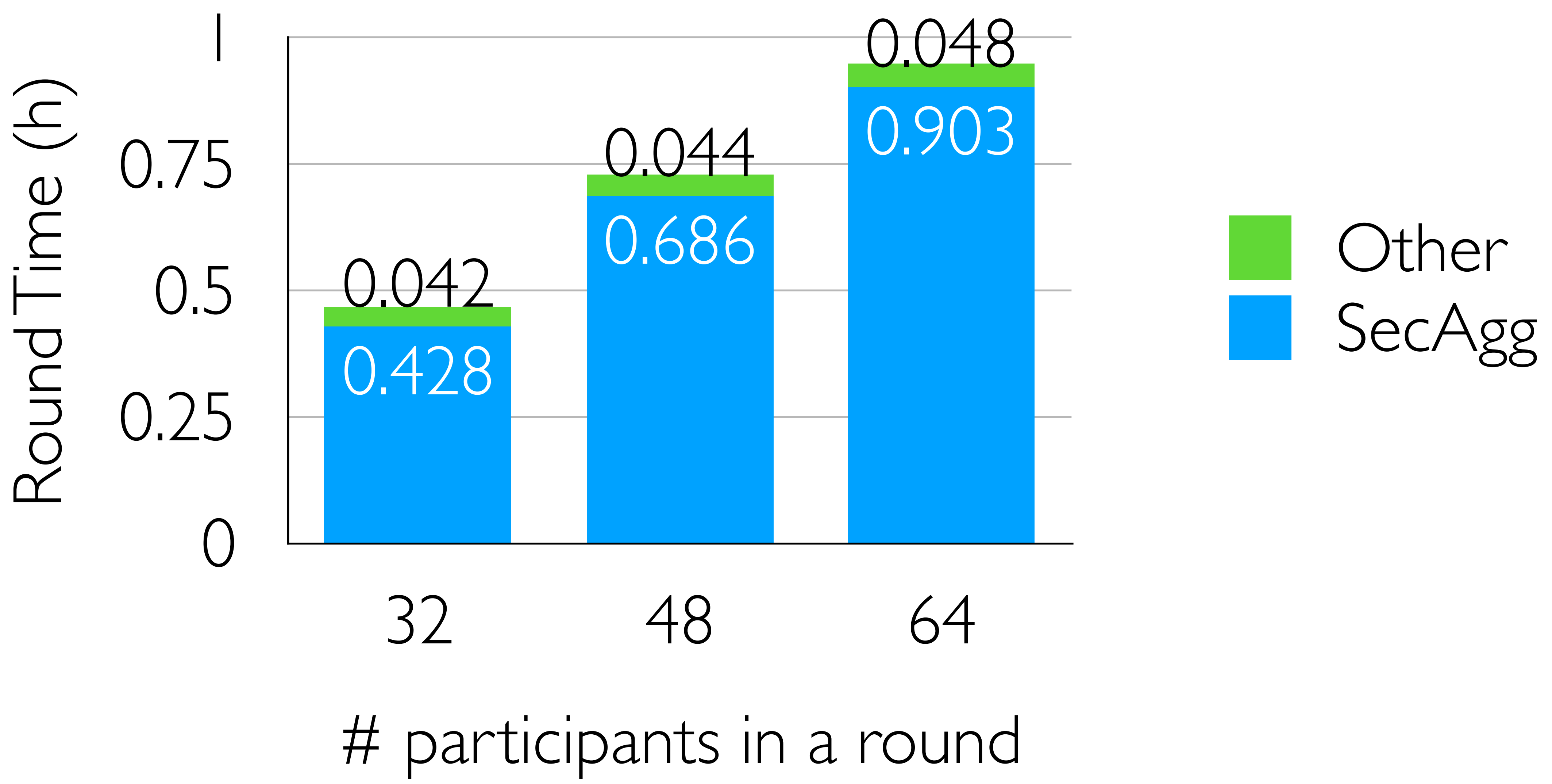
**Problem**: Pairwise masking and secret sharing are necessary but **expensive**




Secure Aggregation

# Need for Dordis - 1

**Problem**: Pairwise masking and secret sharing are necessary but **expensive**




Secure Aggregation

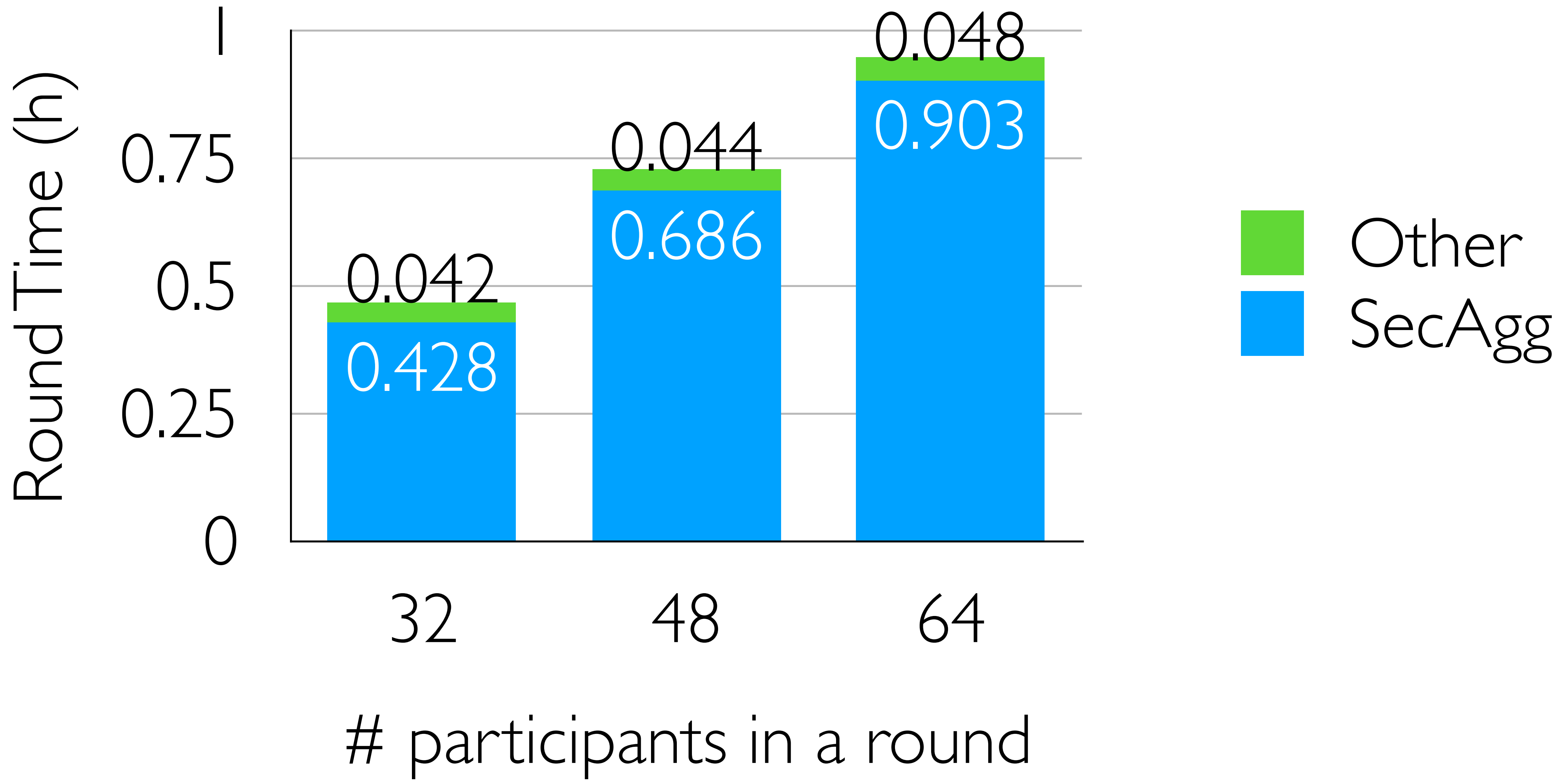
New **algorithms** exist:

- E.g., **SecAgg+**[1]

[1] Bell et al. "Secure Single-Server Aggregation with (Poly) Logarithmic Overhead", In CCS '20

# Need for Dordis - 1

**Problem**: Pairwise masking and secret sharing are necessary but **expensive**




Secure Aggregation

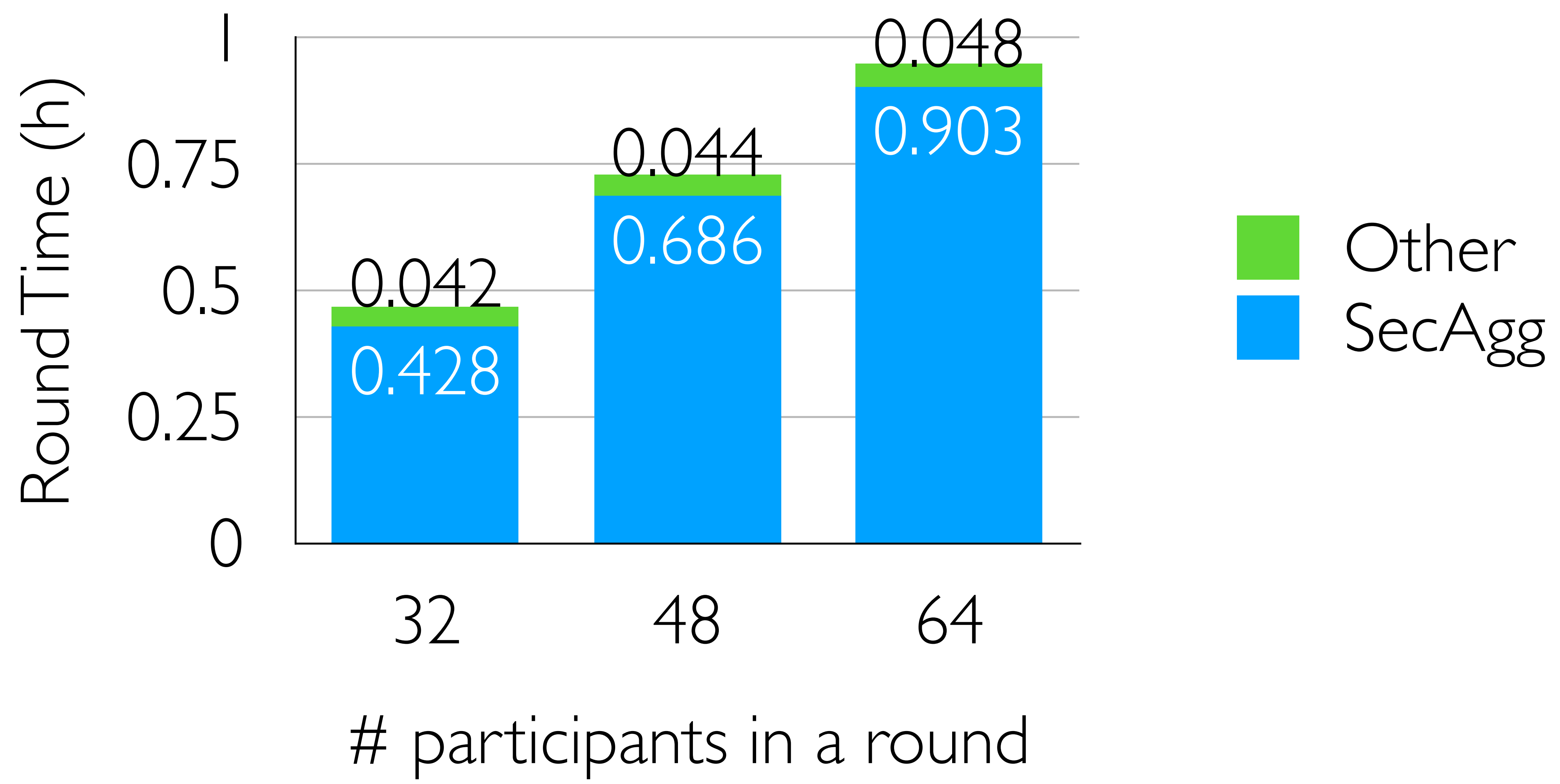
New **algorithms** exist: improve **asymptotically**

- E.g., **SecAgg+**[1], improve the complexity by O(log N)/O(N) (N: # participants in a round)

[1] Bell et al. "Secure Single-Server Aggregation with (Poly) Logarithmic Overhead", In CCS '20

# Need for Dordis - 1

**Problem**: Pairwise masking and secret sharing are necessary but **expensive**




Secure Aggregation

New **algorithms** exist: improve **asymptotically**

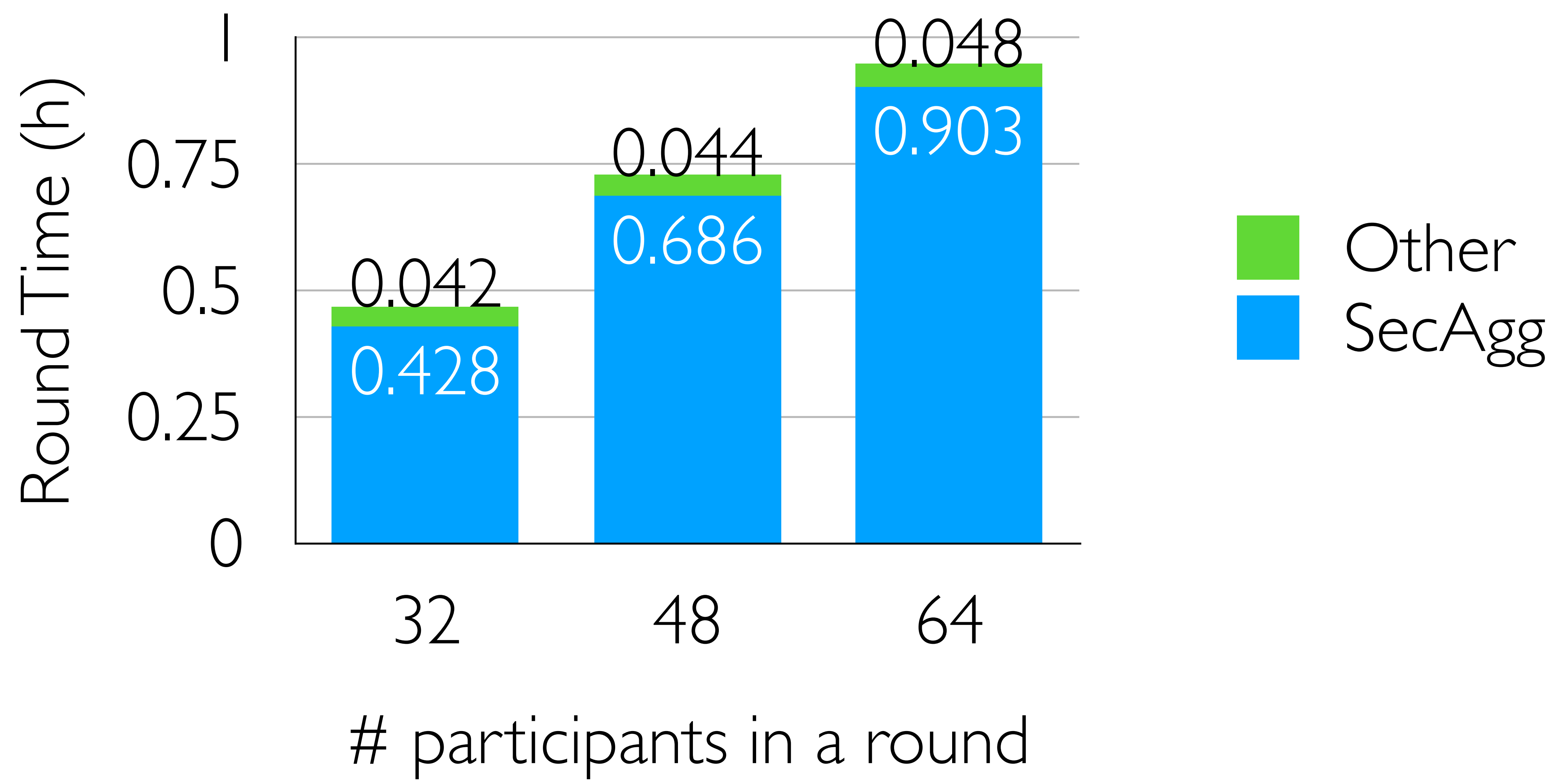
- E.g., **SecAgg+**[1], improve the complexity by O(log N)/O(N) (N: # participants in a round)

NOT so helpful in **FL** where **N = $10^1$-$10^2$**

[1]Bell et al. "Secure Single-Server Aggregation with (Poly) Logarithmic Overhead", In CCS '20

# Need for Dordis - 1

**Problem**: Pairwise masking and secret sharing are necessary but **expensive**

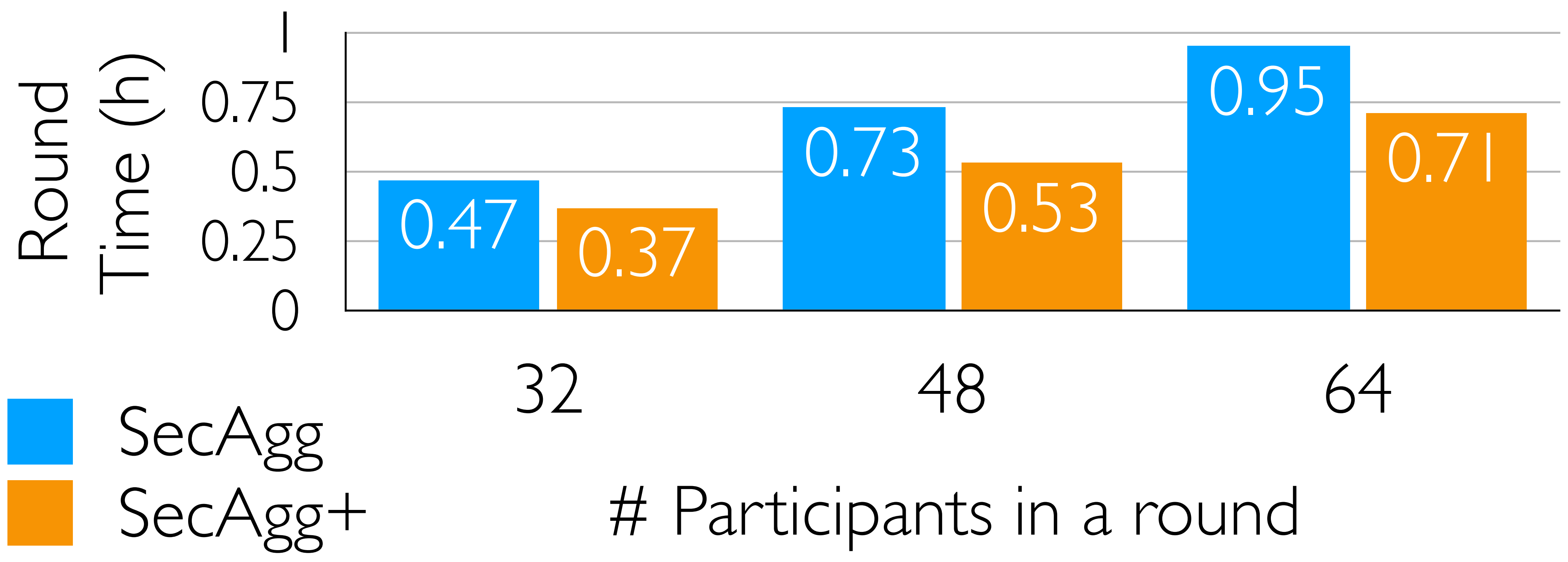



Secure Aggregation

New **algorithms** exist: improve **asymptotically**

- E.g., **SecAgg+**[1], improve the complexity by O(log N)/O(N) (N: # participants in a round)

NOT so helpful in **FL** where **N = $10^1$-$10^2$**

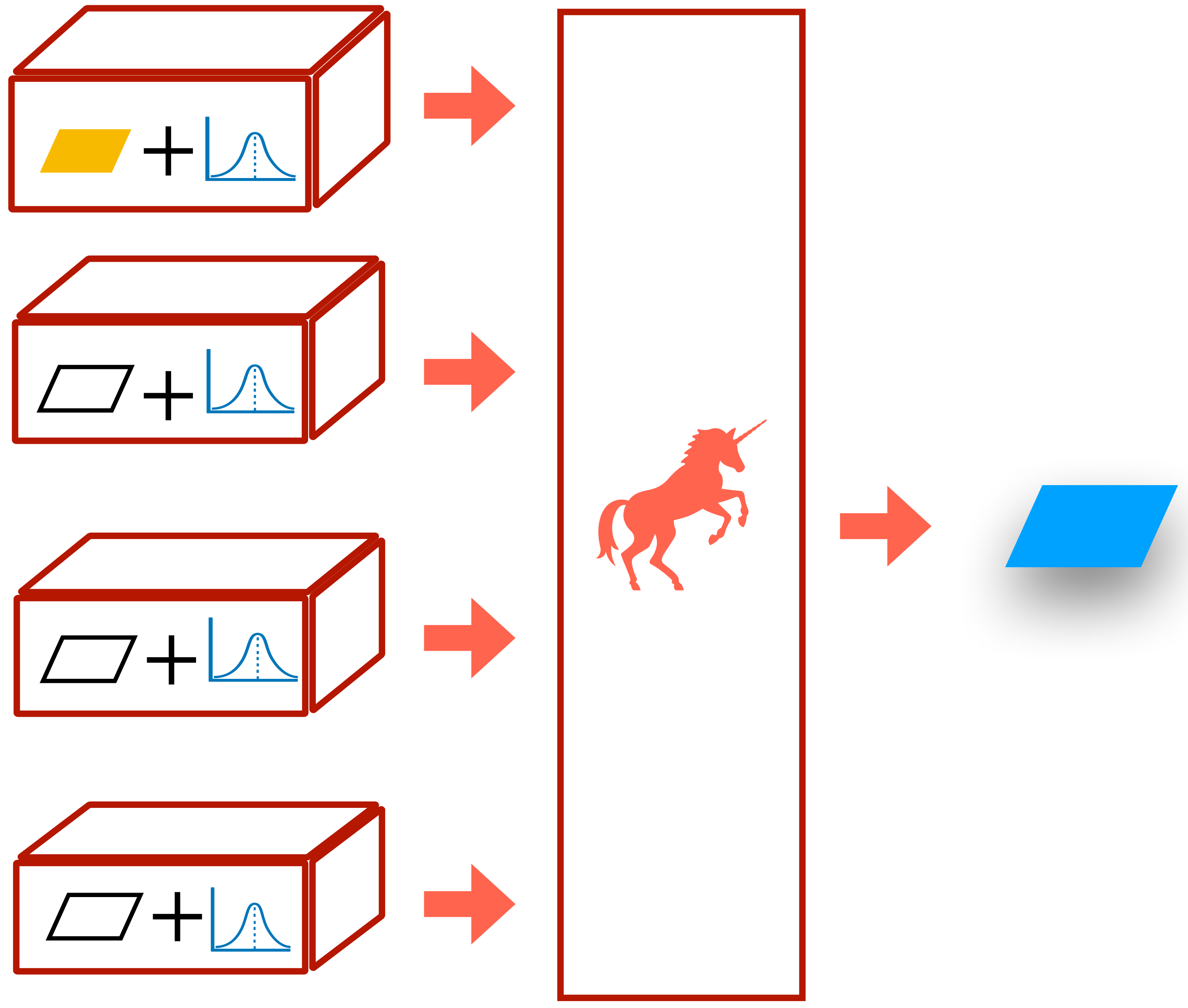



[1] Bell et al. "Secure Single-Server Aggregation with (Poly) Logarithmic Overhead", In CCS '20

# Need for Dordis - 2

Differential Privacy

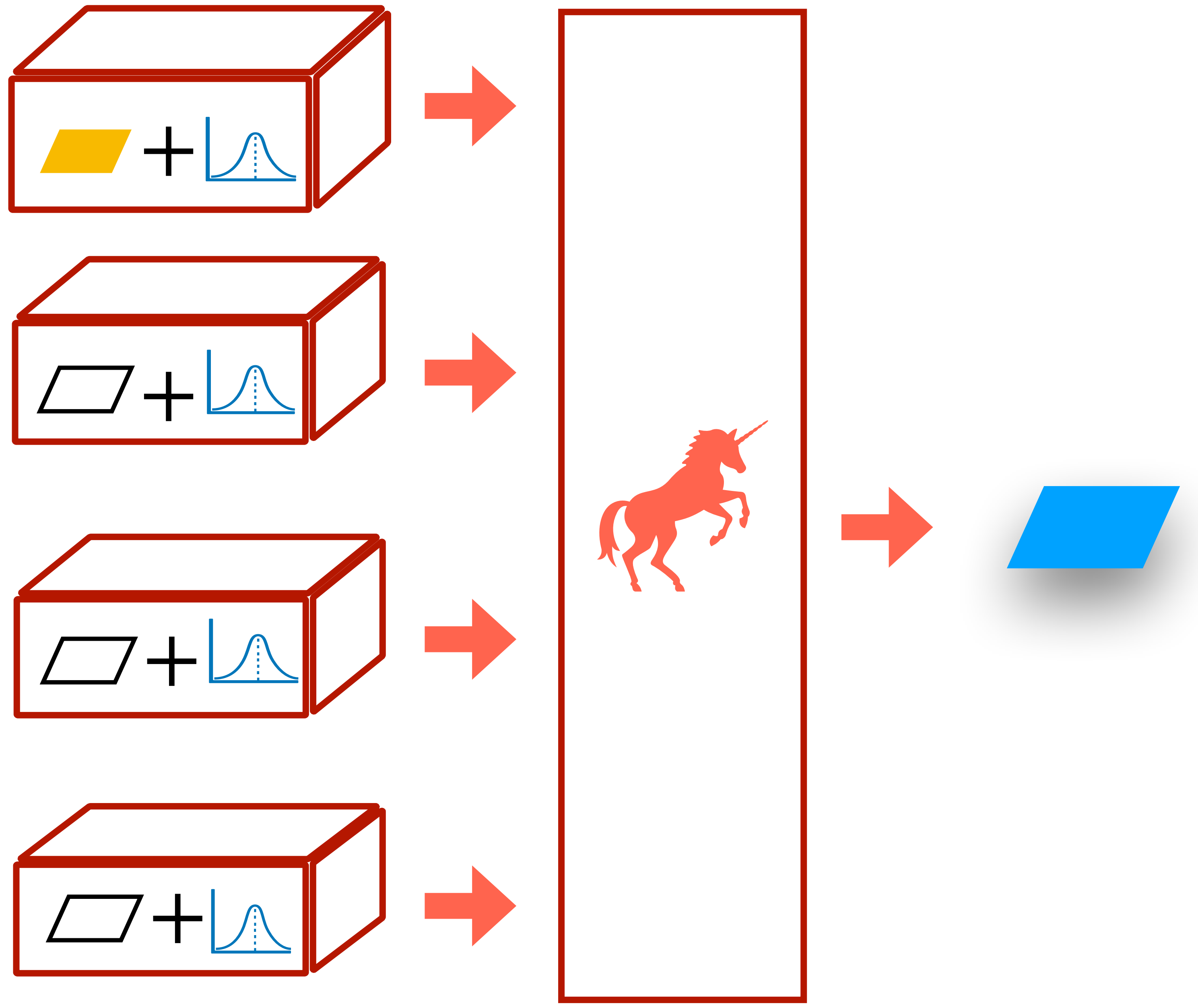
# Need for Dordis - 2



Each client adds an **even share** of the target noise to its local model update

Differential Privacy

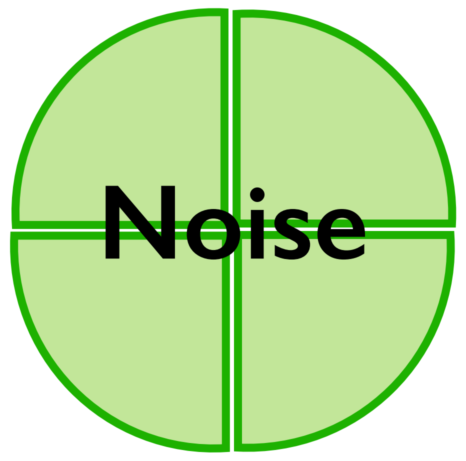
# Need for Dordis - 2



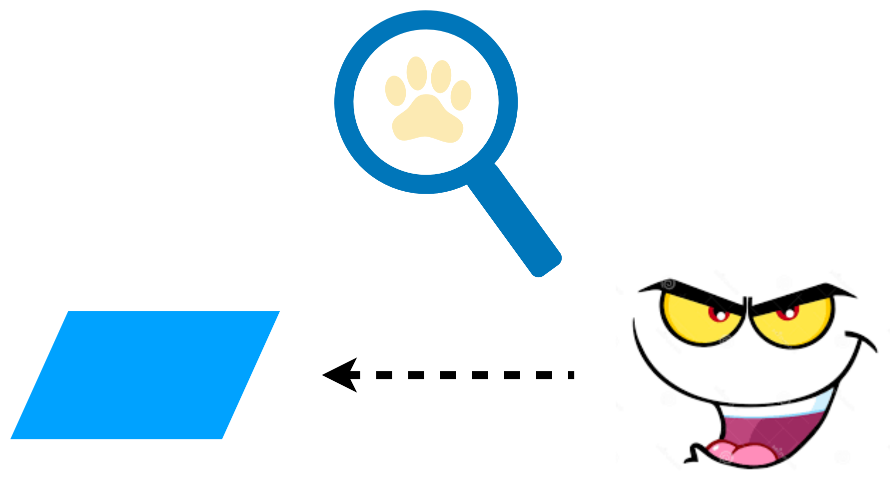
Each client adds an **even share** of the target noise to its local model update

Differential Privacy

**Problem**: **Insufficient** noise at the global update upon client **dropout**

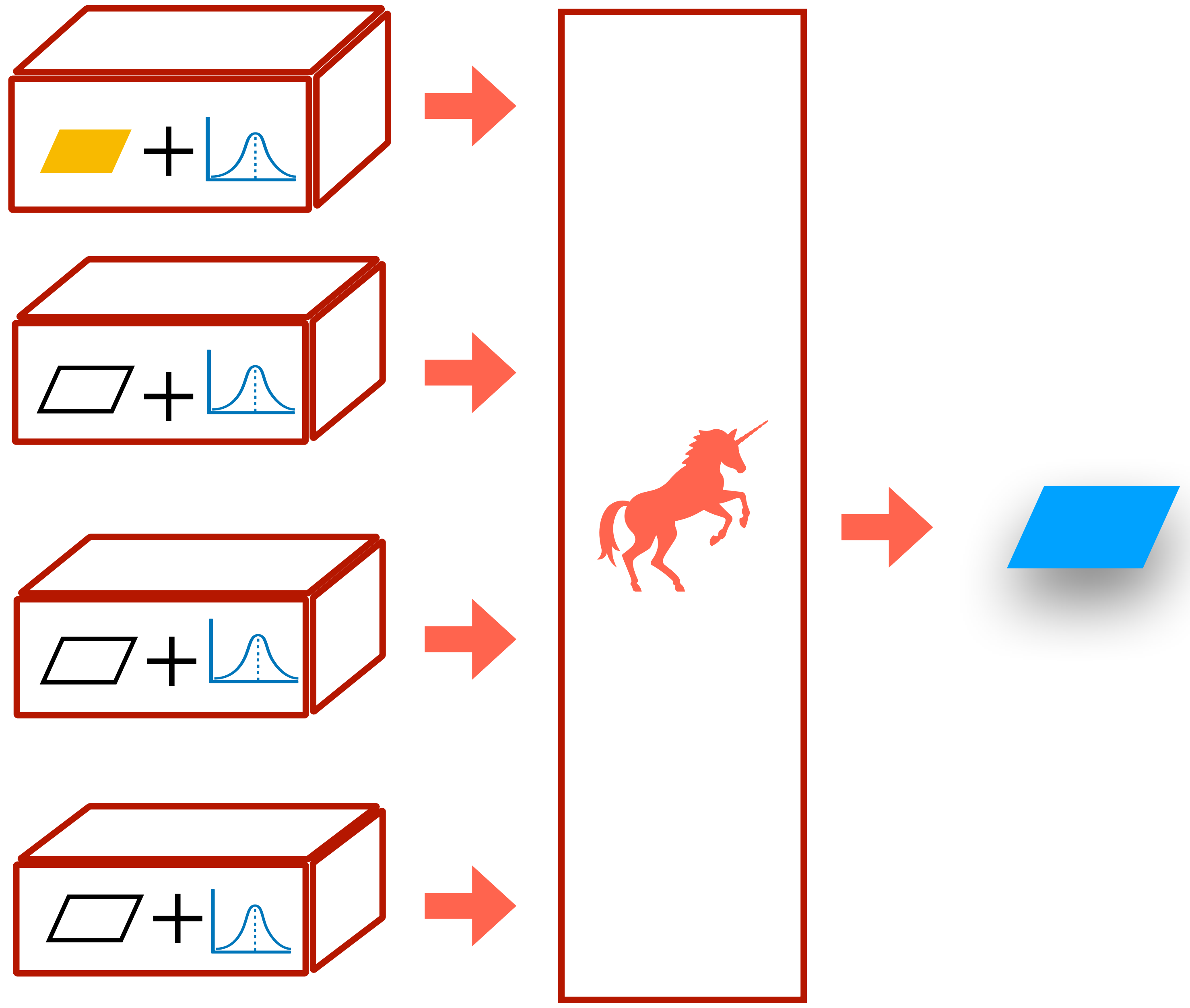



Dropout more severe



Data footprint clearer

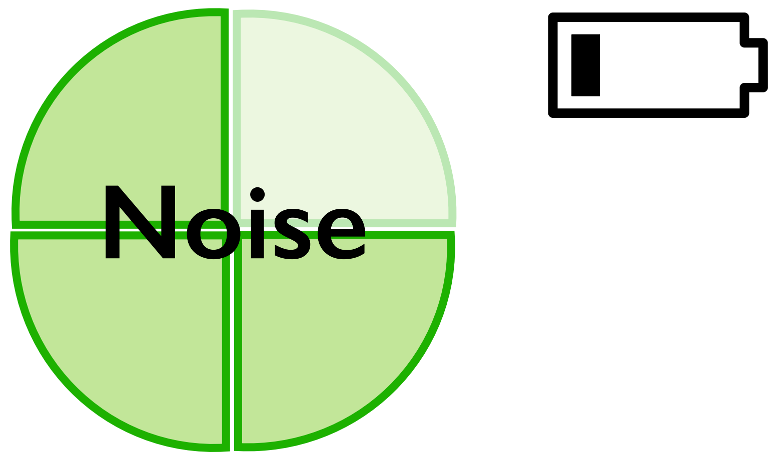
# Need for Dordis - 2



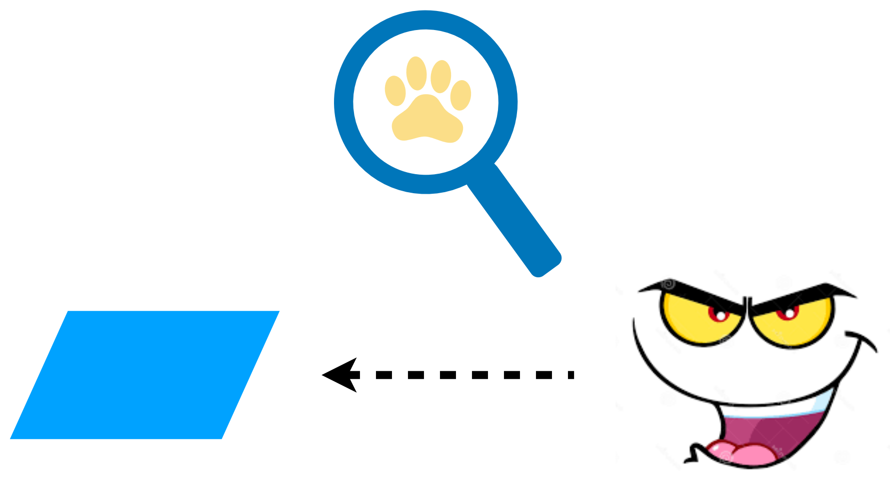
Each client adds an **even share** of the target noise to its local model update

Differential Privacy

**Problem**: **Insufficient** noise at the global update upon client **dropout**

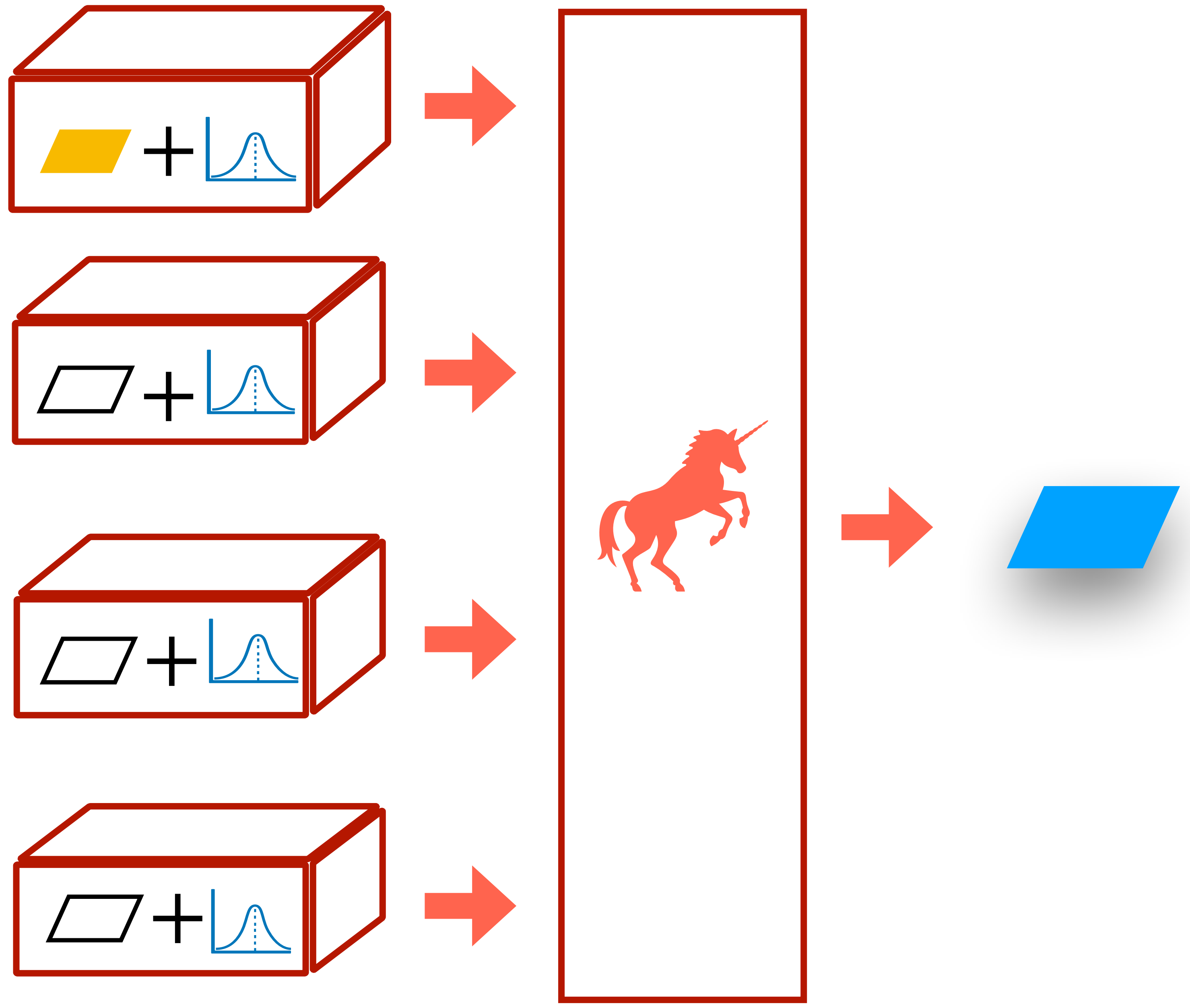



Dropout more severe



Data footprint clearer

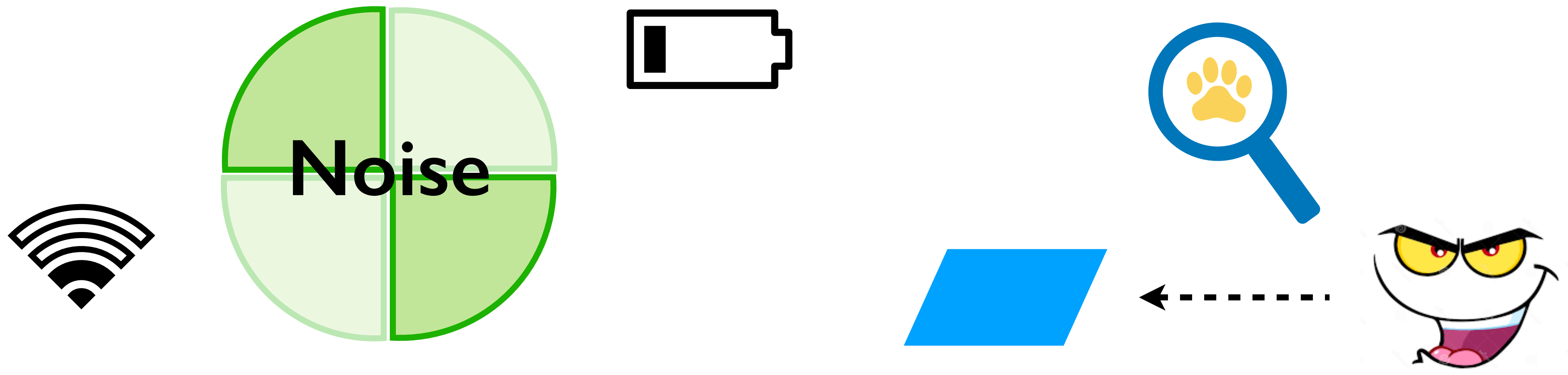
# Need for Dordis - 2



Each client adds an **even share** of the target noise to its local model update

Differential Privacy

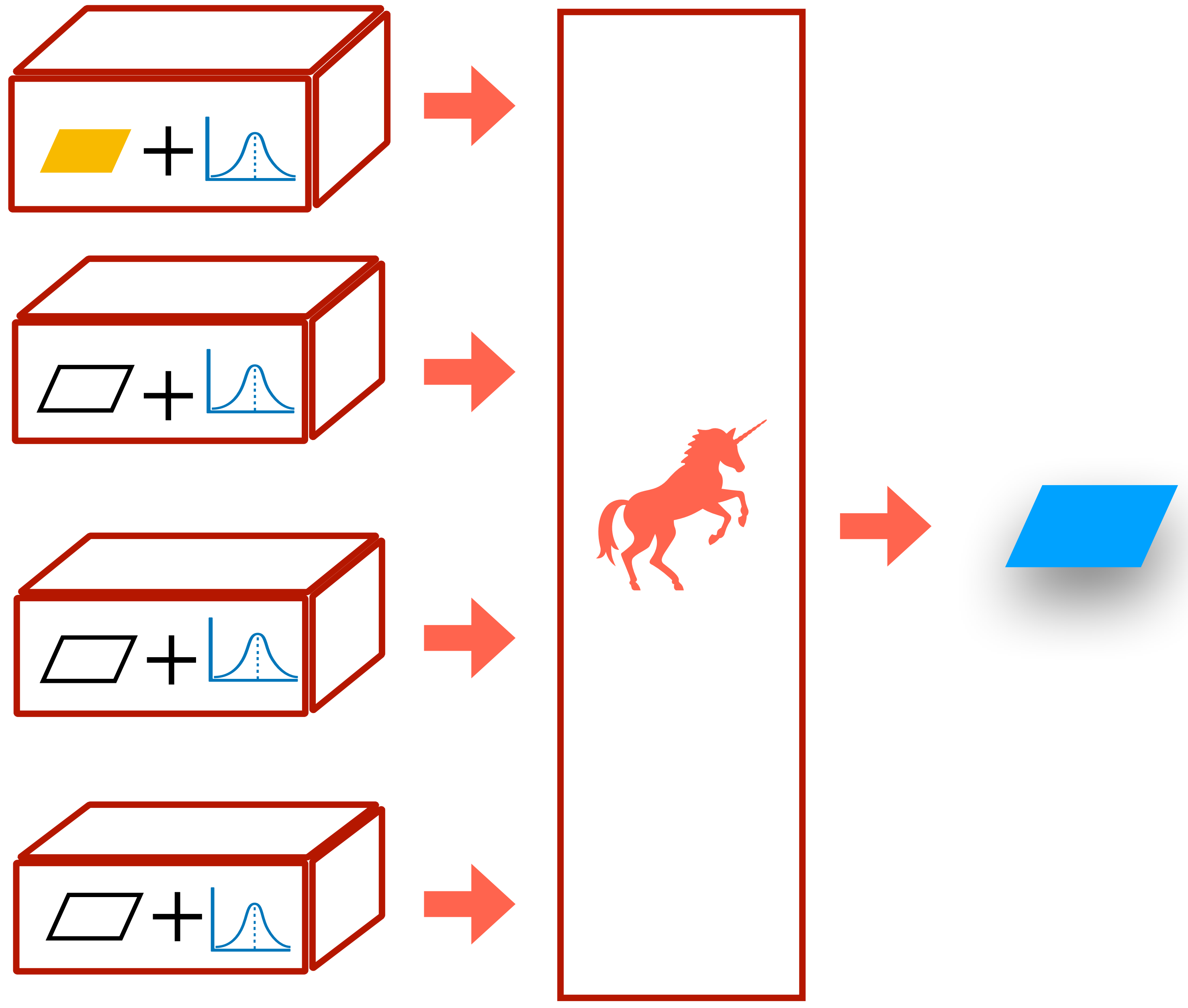
**Problem**: **Insufficient** noise at the global update upon client **dropout**




Dropout more severe

Data footprint clearer

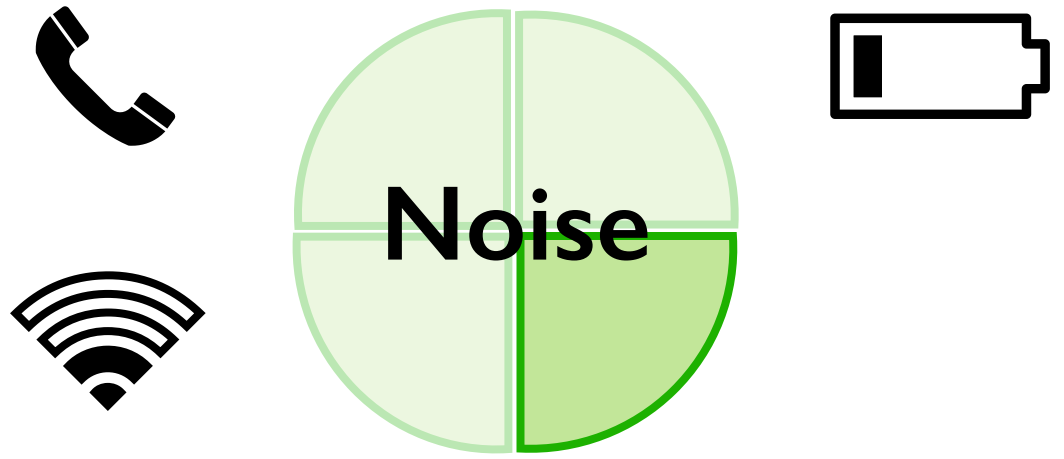
# Need for Dordis - 2



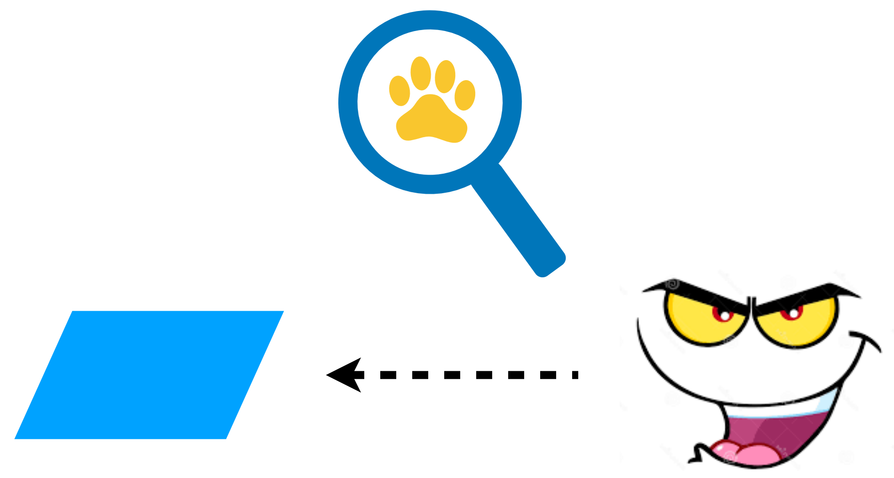
Each client adds an **even share** of the target noise to its local model update

Differential Privacy

**Problem**: **Insufficient** noise at the global update upon client **dropout**

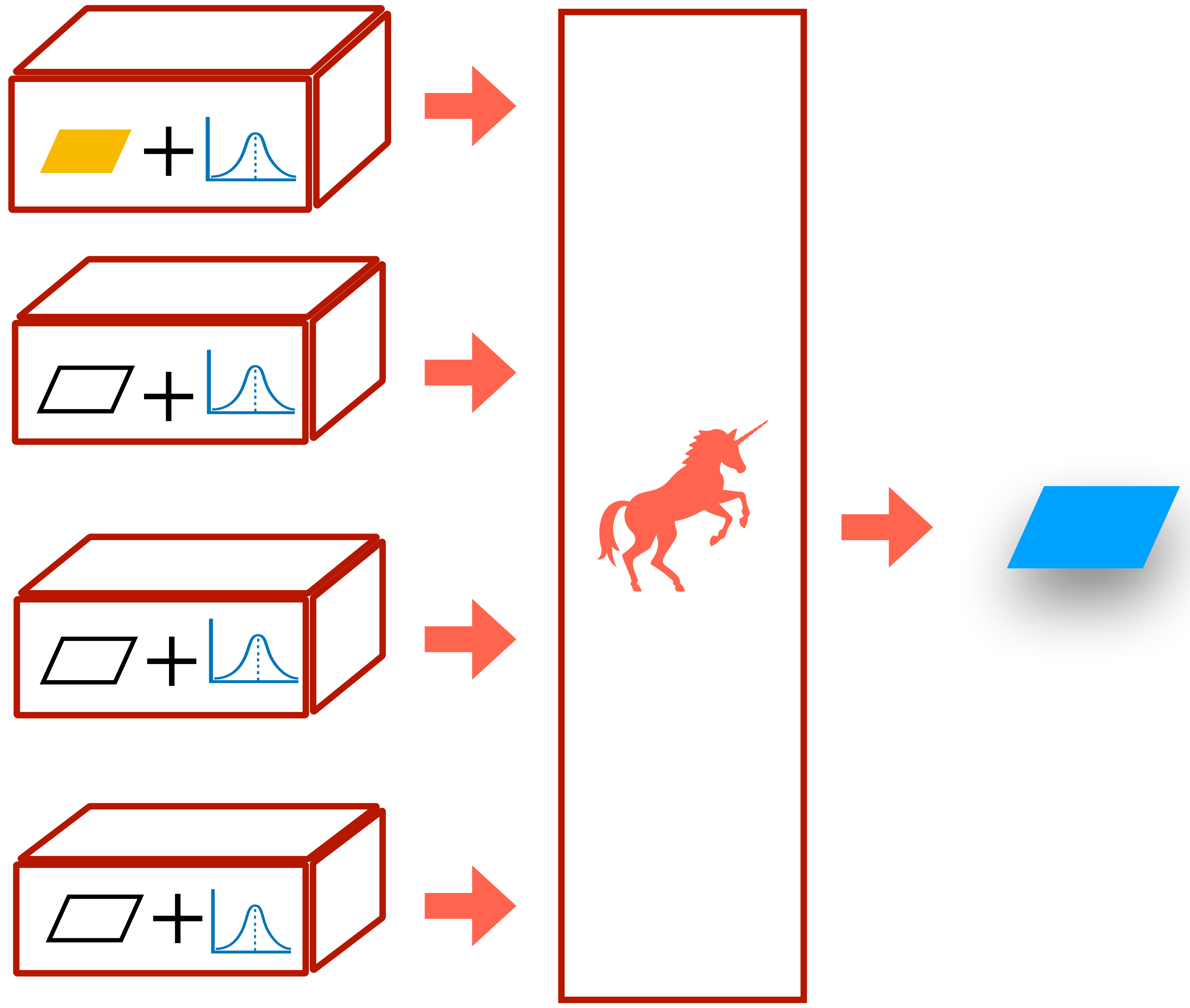



Dropout more severe



Data footprint clearer

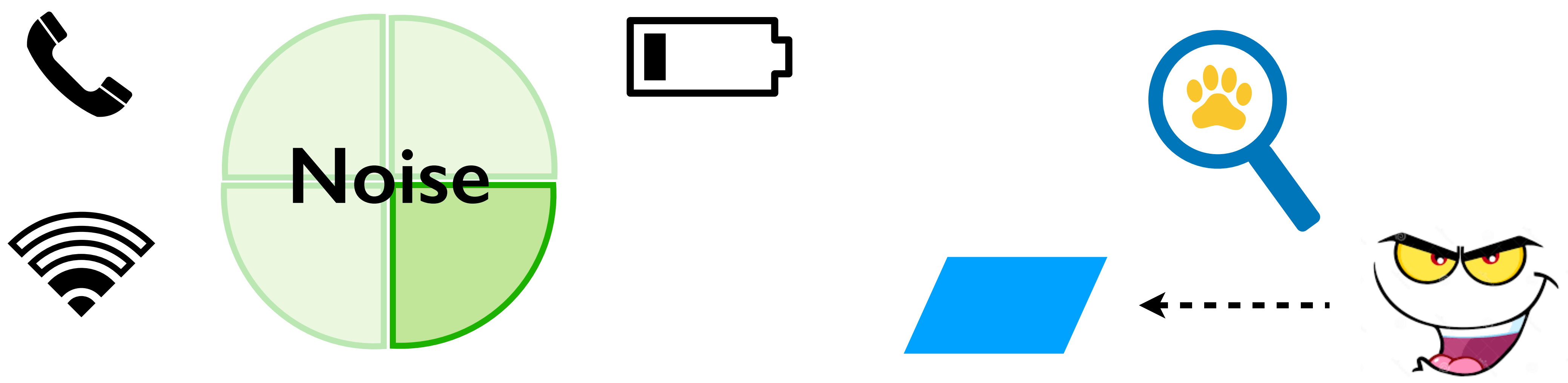
# Need for Dordis - 2



Each client adds an **even share** of the target noise to its local model update

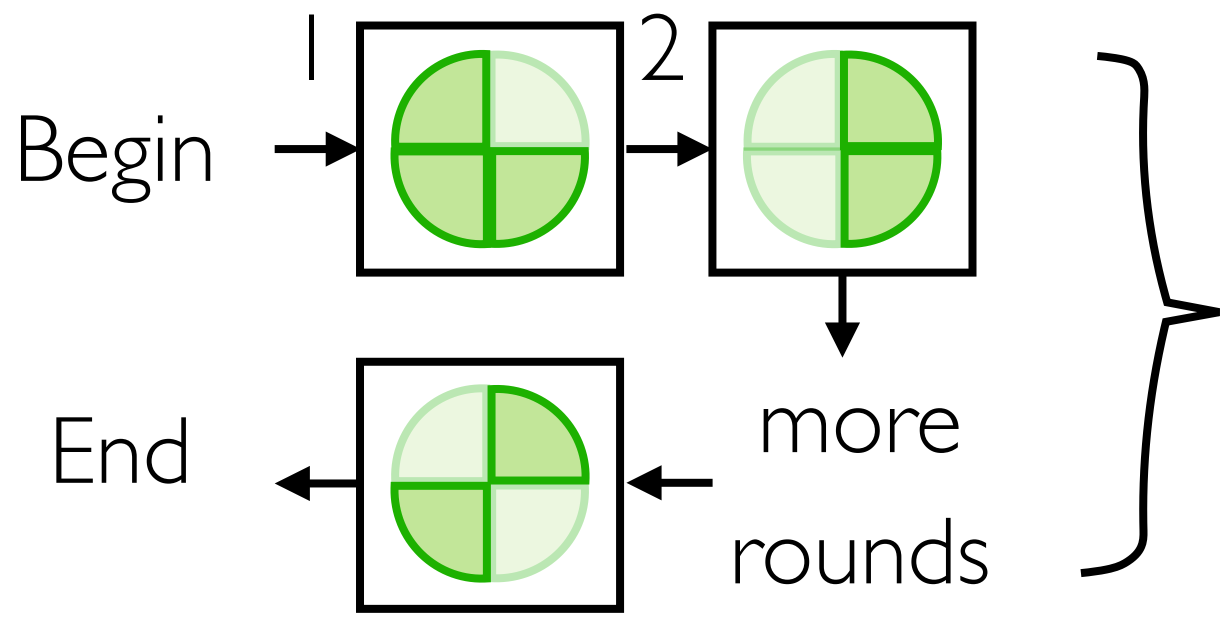
Differential Privacy

**Problem**: **Insufficient** noise at the global update upon client **dropout**




Dropout more severe

Data footprint clearer




Even worse when **accumulated** across rounds

# Dordis - Overview

# Dordis - Overview

Goal 1: **Efficient** secure aggregation

# Dordis - Overview

Goal 1: **Efficient** secure aggregation

**System-level** optimization

# Dordis - Overview

Goal 1: **Efficient** secure aggregation

**System-level** optimization:

FL-specific **pipeline parallelism**

**Efficiency**

**Substantial speedup** for

**general** workloads

# Dordis - Overview

Goal 1: **Efficient** secure aggregation

Goal 2: **Dropout-resilient** DP

**System-level** optimization:

FL-specific **pipeline parallelism**

**Efficiency**

**Substantial speedup** for

**general** workloads

# Dordis - Overview

Goal 1: **Efficient** secure aggregation

Goal 2: **Dropout-resilient** DP

**System-level** optimization:
FL-specific **pipeline parallelism**

Precise **noise enforcement**:
add-then-remove

**Efficiency**

**Resilience**

**Substantial speedup** for
**general** workloads

**Privacy preserved**
**regardless** of client dropout

# Dordis - Overview

Goal 1: **Efficient** secure aggregation

Goal 2: **Dropout-resilient** DP

**System-level** optimization:
FL-specific **pipeline parallelism**

Precise **noise enforcement**:
add-then-remove

**Efficiency**

**Integration**

**Resilience**

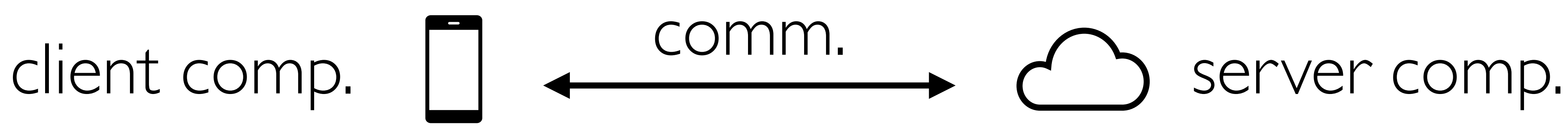
**Substantial speedup** for **general** workloads

**Seamlessly** packed in one **comprehensive** system

**Privacy preserved regardless** of client dropout

# Problem 1: Performance Bottleneck

**System opt.**: Utilize existing resources



# Problem 1: Performance Bottleneck

**System opt.**: Utilize existing resources

client comp. ⟷ comm. ⟷ server comp.

| Step | Operation | Resource |
|---|---|---|
| 1 | Clients encode updates | **client comp.** |
| 2 | Clients generate security keys | **client comp.** |
| 3 | Clients establish shared secrets | **client comp.** |
| 4 | Clients mask encoded updates | **client comp.** |
| 5 | Clients upload masked updates | **comm.** |
| 6 | Server deals with dropout | **server comp.** |
| 7 | Server computes the sum | **server comp.** |
| 8 | Server updates global model | **server comp.** |
| 9 | Server dispatches global model | **comm.** |
| 10 | Clients decode global model | **client comp.** |
| 11 | Clients use global model | **client comp.** |

# Problem 1: Performance Bottleneck

**System opt.**: Utilize existing resources

client comp. ⟵ comm. ⟶ server comp.

| Step | Operation | Resource | Stage |
|---|---|---|---|
| 1 | Clients encode updates | client comp. | 1 |
| 2 | Clients generate security keys | | |
| 3 | Clients establish shared secrets | | |
| 4 | Clients mask encoded updates | | |
| 5 | Clients upload masked updates | comm. | 2 |
| 6 | Server deals with dropout | server comp. | 3 |
| 7 | Server computes the sum | | |
| 8 | Server updates global model | | |
| 9 | Server dispatches global model | comm. | 4 |
| 10 | Clients decode global model | client comp. | 5 |
| 11 | Clients use global model | | |

# Problem 1: Performance Bottleneck

**System opt.**: Utilize existing resources

client comp. ⟷ comm. ⟷ server comp.

| Step | Operation | Resource | Stage |
|---|---|---|---|
| 1 | Clients encode updates | client comp. | 1 |
| 2 | Clients generate security keys | | |
| 3 | Clients establish shared secrets | | |
| 4 | Clients mask encoded updates | | |
| 5 | Clients upload masked updates | comm. | 2 |
| 6 | Server deals with dropout | server comp. | 3 |
| 7 | Server computes the sum | | |
| 8 | Server updates global model | | |
| 9 | Server dispatches global model | comm. | 4 |
| 10 | Clients decode global model | client comp. | 5 |
| 11 | Clients use global model | | |

**Potential approach**:

- Pipeline parallelism

# Problem 1: Performance Bottleneck

**System opt.**: Utilize existing resources

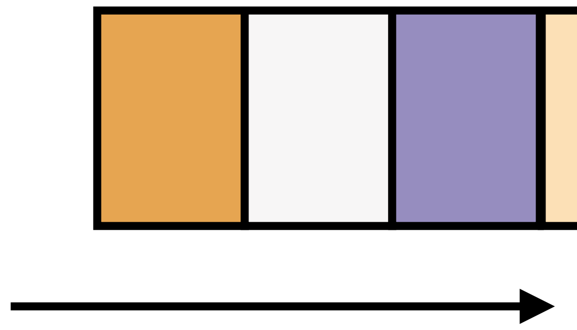
client comp. ↔ comm. ↔ server comp.

| Step | Operation | Resource | Stage |
|---|---|---|---|
| 1 | Clients encode updates | client comp. | 1 |
| 2 | Clients generate security keys | | |
| 3 | Clients establish shared secrets | | |
| 4 | Clients mask encoded updates | | |
| 5 | Clients upload masked updates | comm. | 2 |
| 6 | Server deals with dropout | server comp. | 3 |
| 7 | Server computes the sum | | |
| 8 | Server updates global model | | |
| 9 | Server dispatches global model | comm. | 4 |
| 10 | Clients decode global model | client comp. | 5 |
| 11 | Clients use global model | | |

**Potential approach**:

- Pipeline parallelism

Diff. stages,
**diff** resources



Workflow

**Traditional ML**: Free data movement

# Problem 1: Performance Bottleneck

**System opt.**: Utilize existing resources

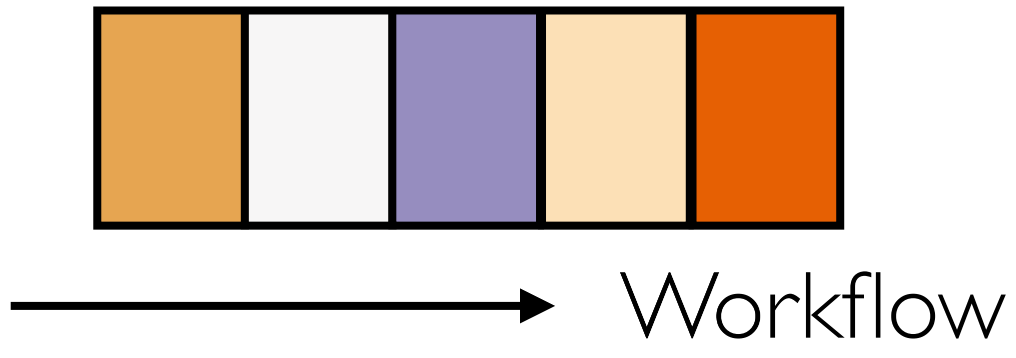
client comp. ←comm.→ server comp.

| Step | Operation | Resource | Stage |
|---|---|---|---|
| 1 | Clients encode updates | client comp. | 1 |
| 2 | Clients generate security keys | | |
| 3 | Clients establish shared secrets | | |
| 4 | Clients mask encoded updates | | |
| 5 | Clients upload masked updates | comm. | 2 |
| 6 | Server deals with dropout | server comp. | 3 |
| 7 | Server computes the sum | | |
| 8 | Server updates global model | | |
| 9 | Server dispatches global model | comm. | 4 |
| 10 | Clients decode global model | client comp. | 5 |
| 11 | Clients use global model | | |

**Potential approach**:

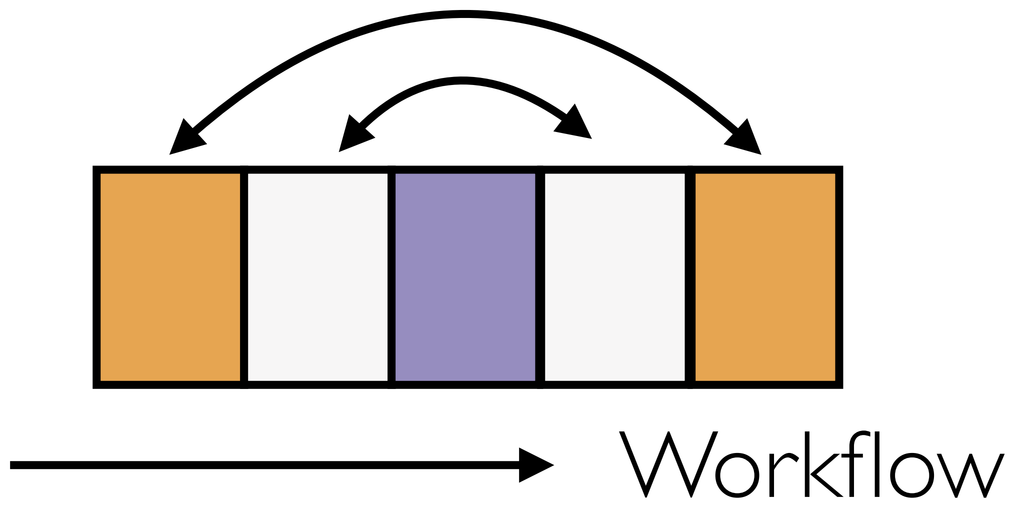
- Pipeline parallelism

Diff. stages, **diff** resources




**Traditional ML**: Free data movement

Diff. stages, **same** resource




**FL**: Data movement restricted due to privacy

**Challenge**: New **constraints** in optimizing pipeline parallelism

# Problem 1: Performance Bottleneck

**Solution:** pipeline parallelism tailored for **FL**

# Problem 1: Performance Bottleneck

**Solution:** pipeline parallelism tailored for **FL**

1. **Task partitioning**: enable parallelism

# Problem 1: Performance Bottleneck

**Solution:** pipeline parallelism tailored for **FL**

1. **Task partitioning**: enable parallelism

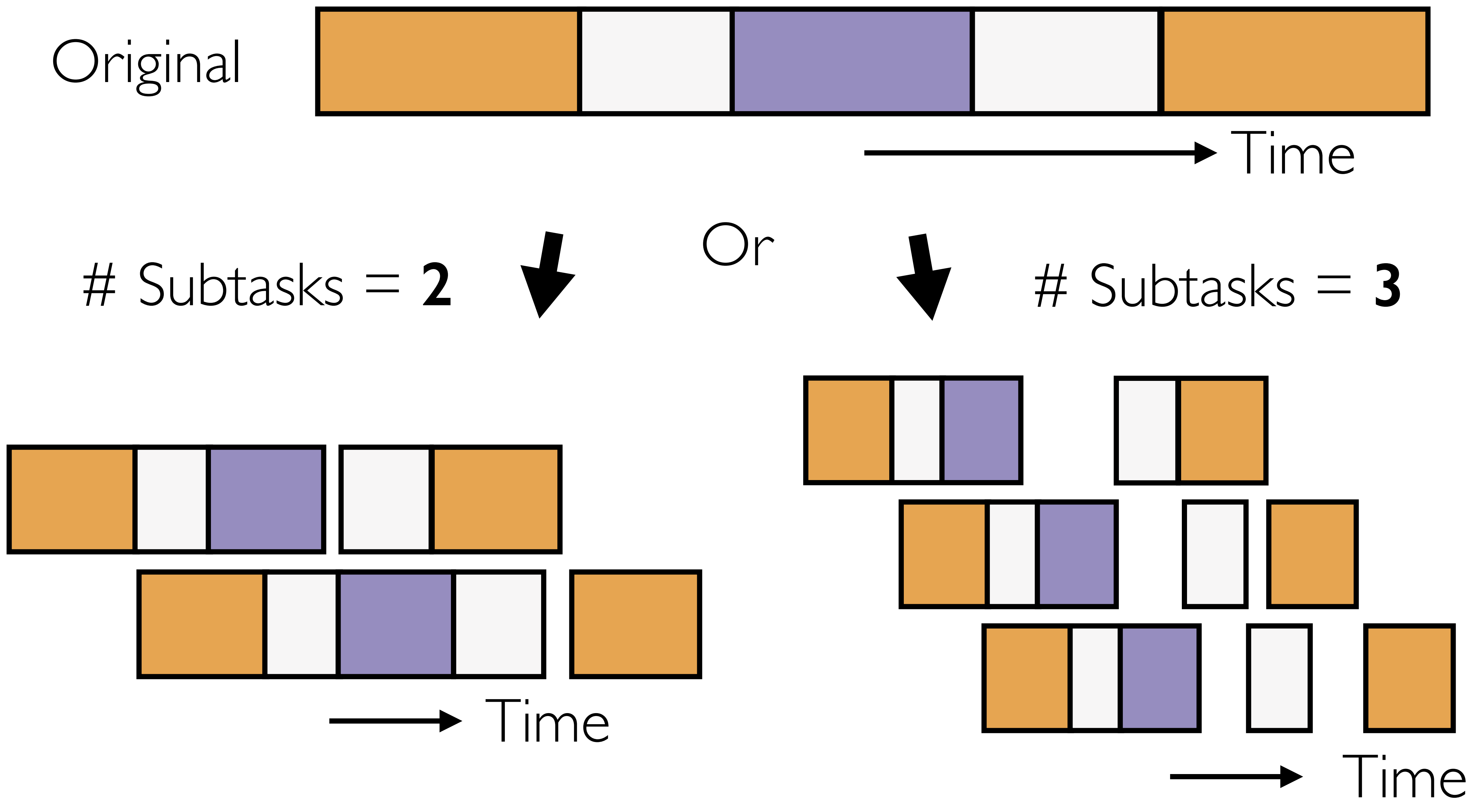
- # Subtasks: decision variable to optimize

# Problem 1: Performance Bottleneck

**Solution:** pipeline parallelism tailored for **FL**

1. **Task partitioning**: enable parallelism

- # Subtasks: decision variable to optimize



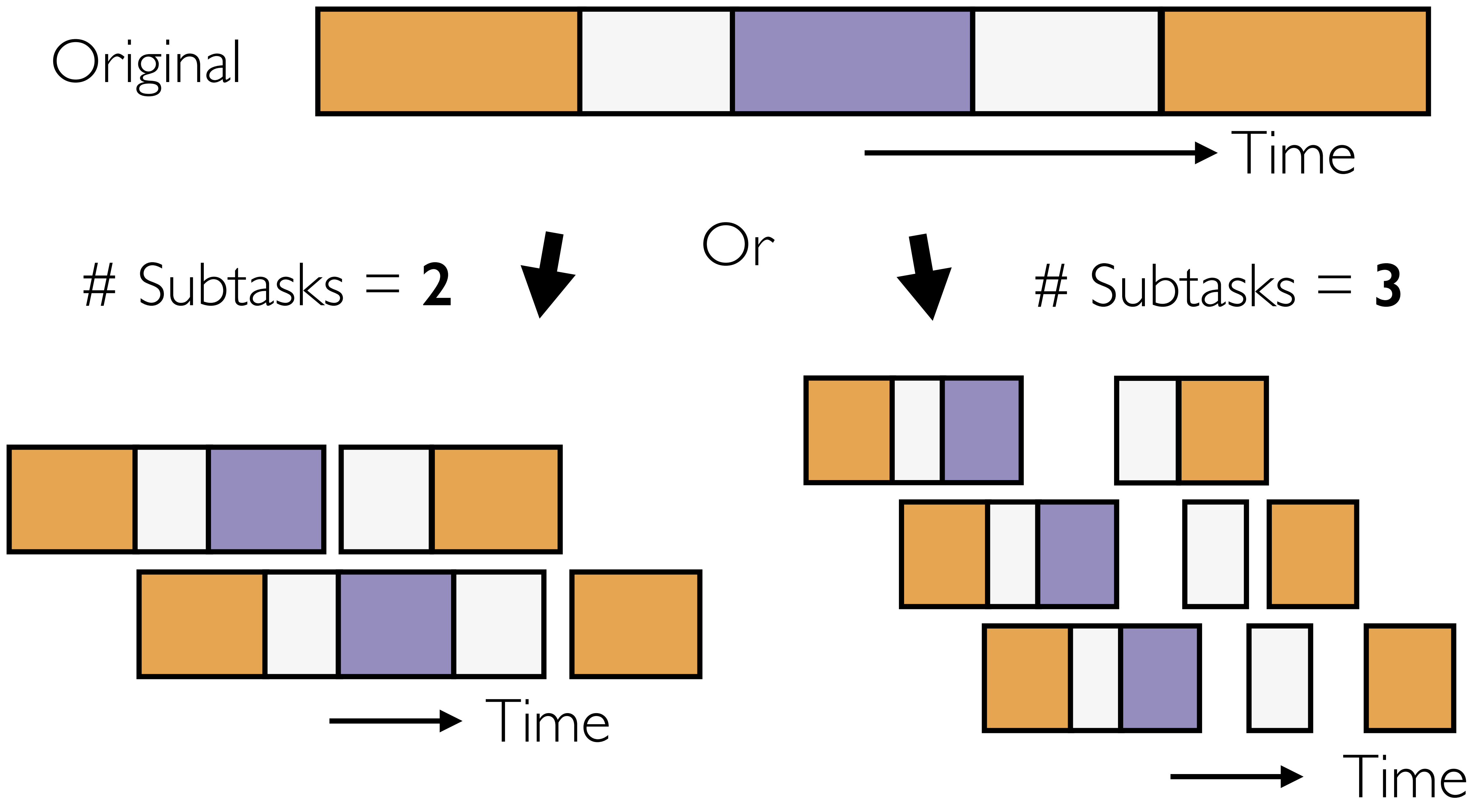

**Different time**

# Problem 1: Performance Bottleneck

**Solution:** pipeline parallelism tailored for **FL**

1. **Task partitioning**: enable parallelism

- # Subtasks: decision variable to optimize




**Different time**

2. **Constrained optimization**

$$m^* = \arg\min_{m \in N_+} f_{a,m}$$ Optimal # subtasks

$$s.t. \quad f_{s,c} = b_{s,c} + l_s$$

$$b_{s,c} = \max\{o_{s,c}, r_{s,c}\}$$

$$o_{s,c} = \begin{cases} 0, & \text{if } s = 0, \\ f_{s-1,c} & \end{cases}$$

**The FL constraint**

$$r_{s,c} = \begin{cases} 0, & \text{if } s = 0 \text{ and } c = 0, \\ f_{q,m} \text{ or } \perp, & \text{if } s \neq 0 \text{ and } c = 0, \\ f_{s,c-1}, & \text{otherwise} \end{cases}$$

Please find more in the paper :)

# Dordis generally boosts performance

**FEMNIST @ResNet18**

More participants

**CIFAR-10 @ResNet18**

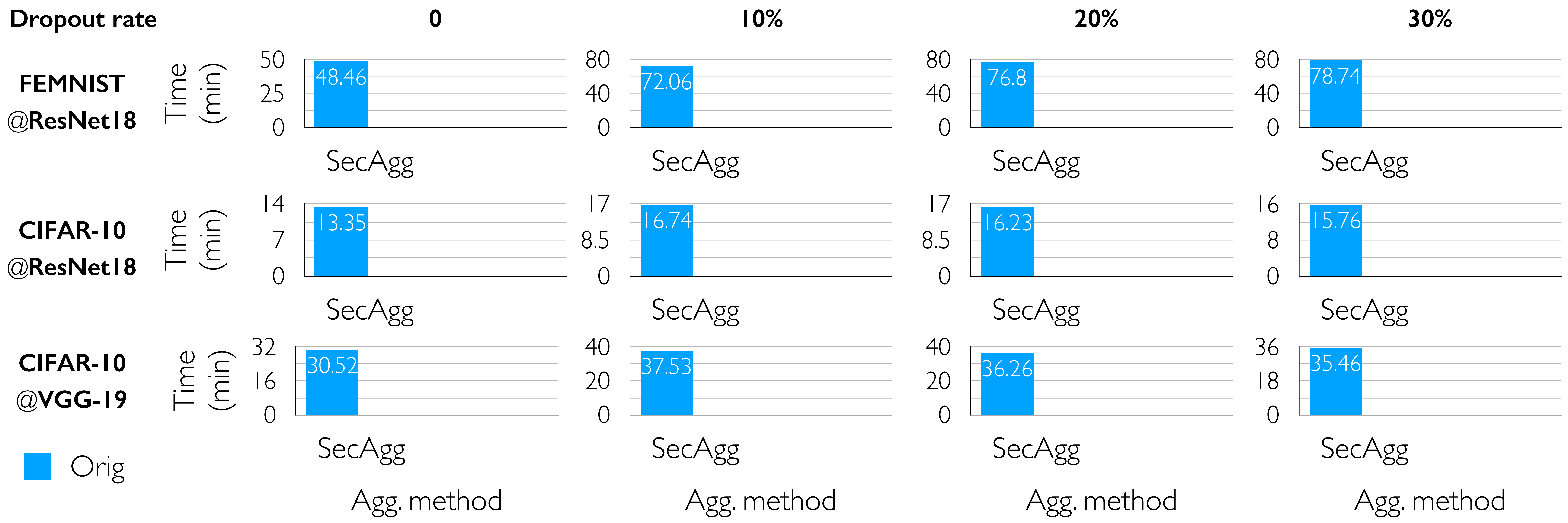
Larger model

**CIFAR-10 @VGG-19**

# Dordis generally boosts performance

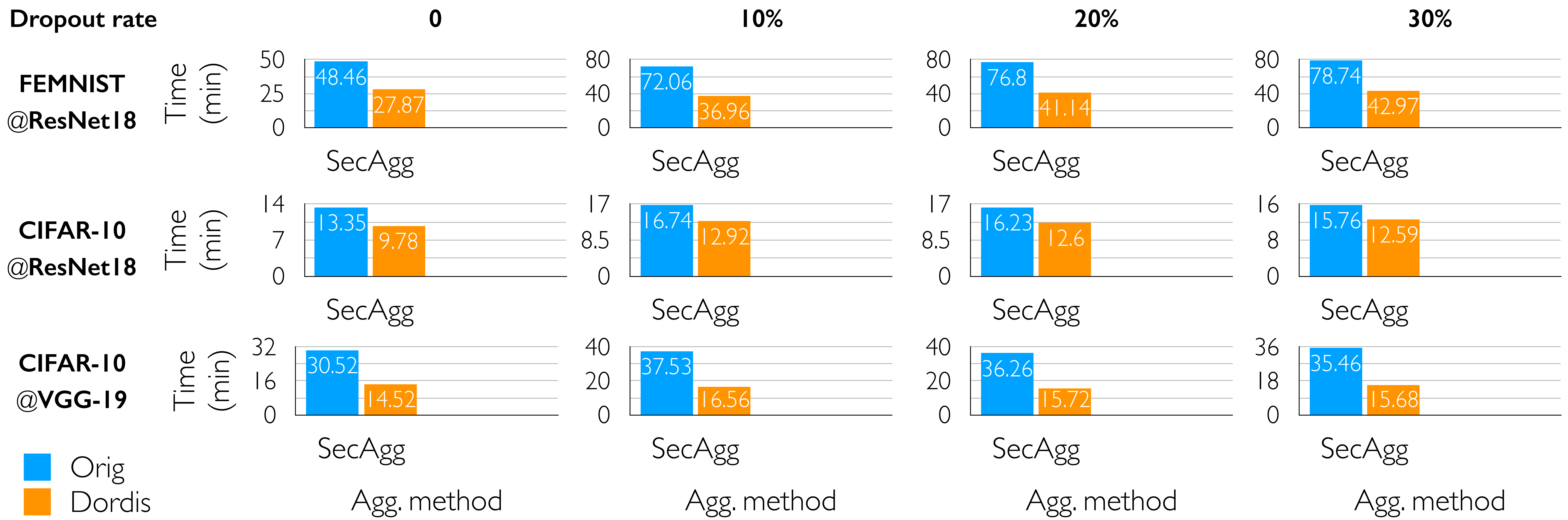
| Dropout rate | 0 | 10% | 20% | 30% |
|---|---|---|---|---|
| FEMNIST @ResNet18 | | | | |
| CIFAR-10 @ResNet18 | | | | |
| CIFAR-10 @VGG-19 | | | | |

# Dordis generally boosts performance




**Orig** → Plain sequential execution

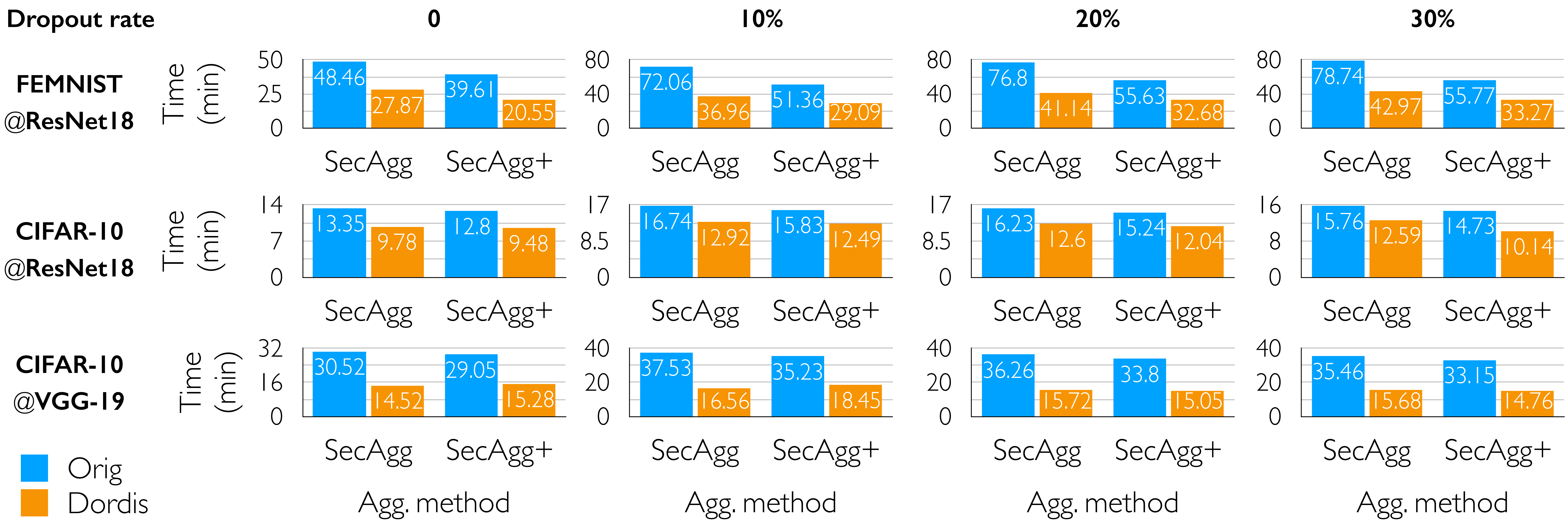
# Dordis generally boosts performance




**Orig** → Plain sequential execution

**Dordis accelerates** by up to **2.4×** across different **participant** scale, **model** size, **dropout** situations

# Dordis generally boosts performance




**Orig** → Plain sequential execution

**Dordis accelerates** by up to **2.4×** across different **participant** scale, **model** size, **dropout** situations, and **aggregation** methods

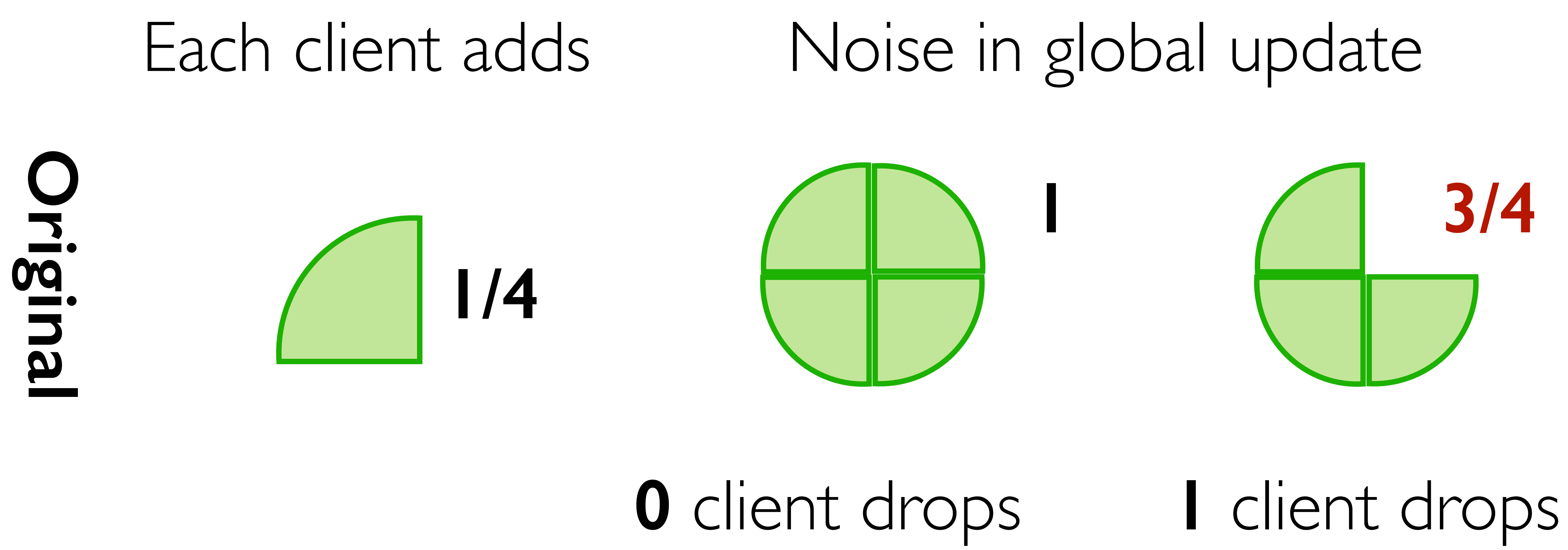
# Problem 2: Noise Deficiency

**Intuition** - Data privacy

# Problem 2: Noise Deficiency

**Intuition** - Data privacy

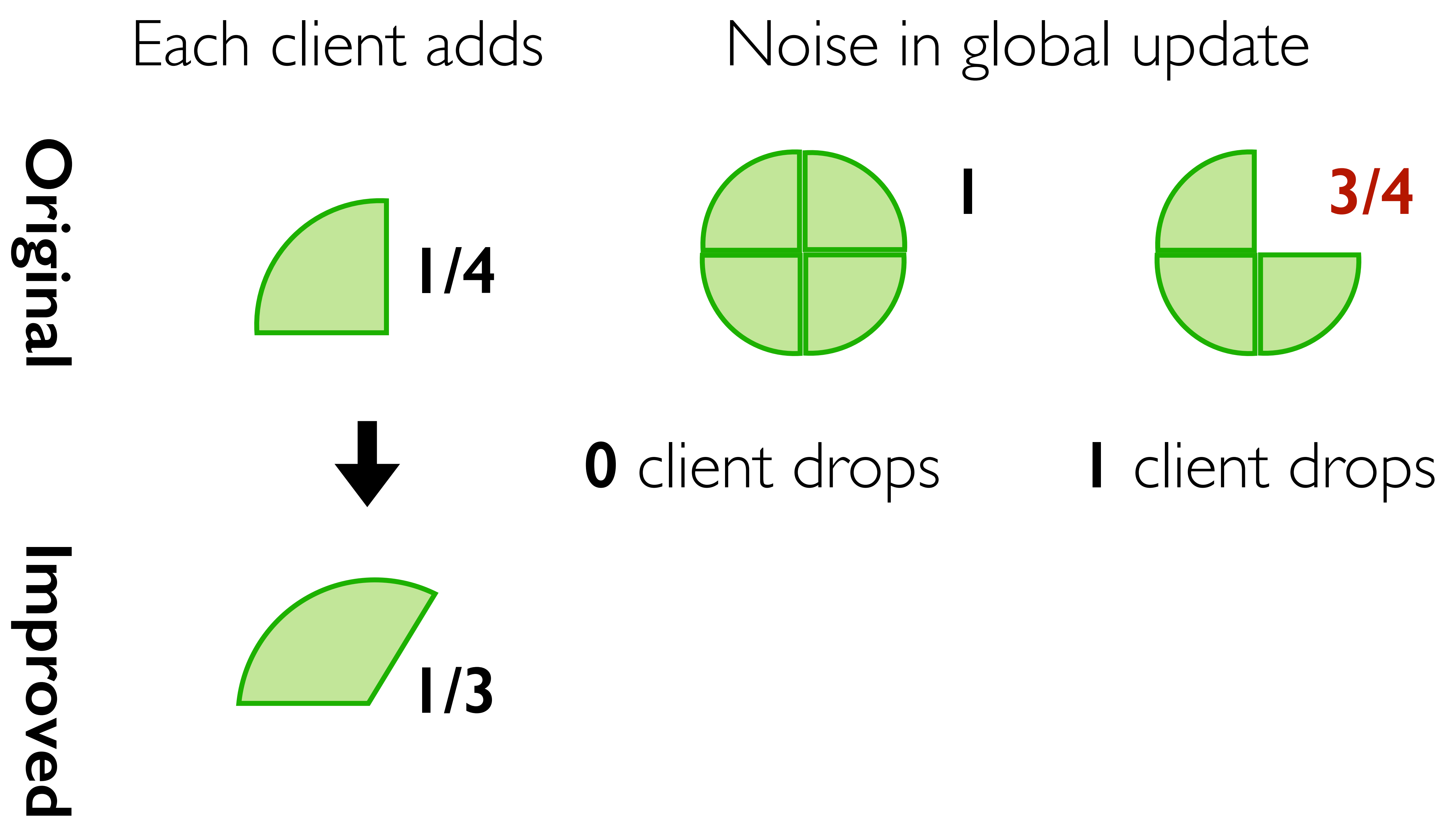
- Noise should **never** be **insufficient**



# Problem 2: Noise Deficiency

**Intuition** - Data privacy

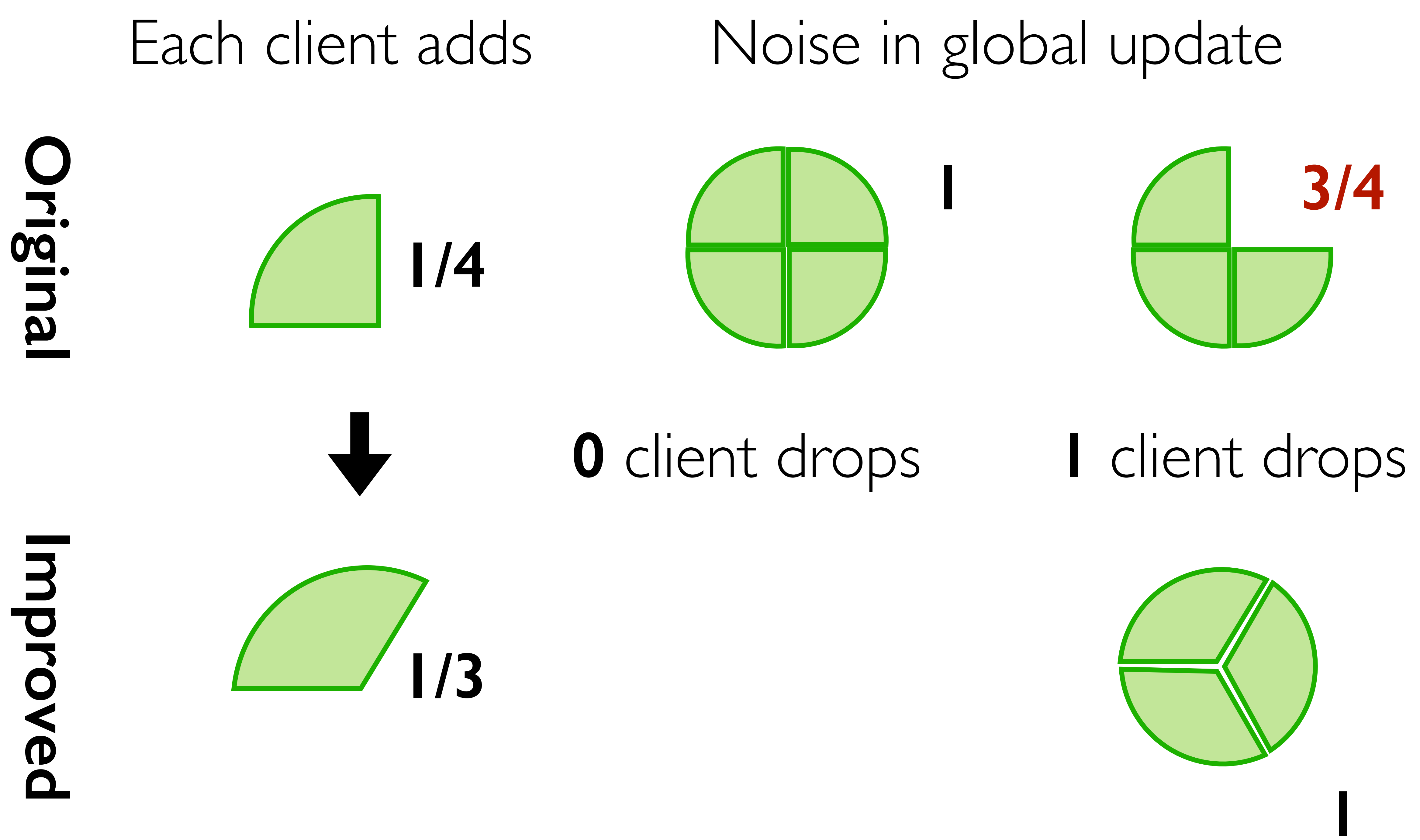
- Noise should **never** be **insufficient** → Proactively **add more** noise than needed



# Problem 2: Noise Deficiency

**Intuition** - Data privacy

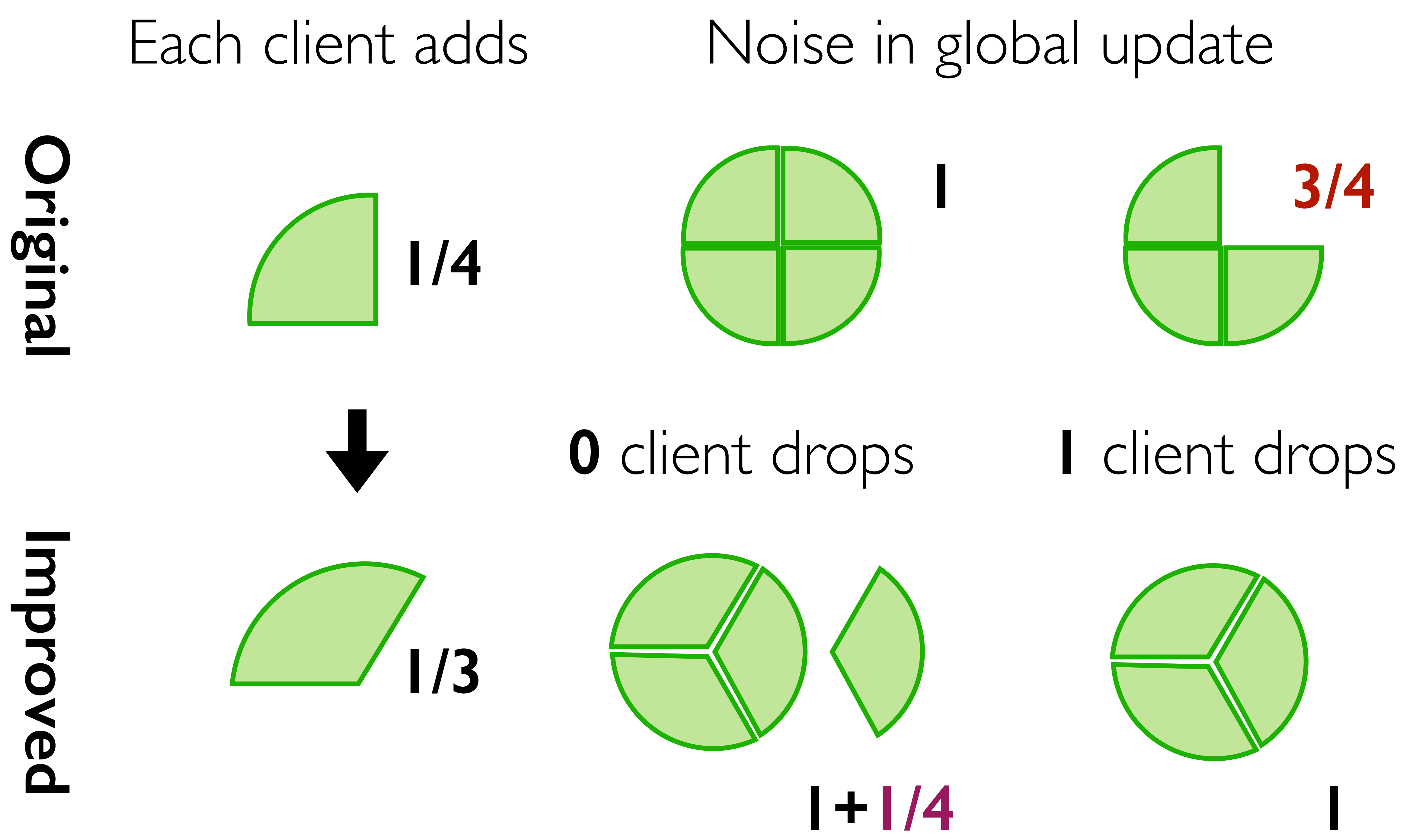
- Noise should **never** be **insufficient** → Proactively **add more** noise than needed



# Problem 2: Noise Deficiency

**Intuition** - Data privacy

- Noise should **never** be **insufficient** → Proactively **add more** noise than needed



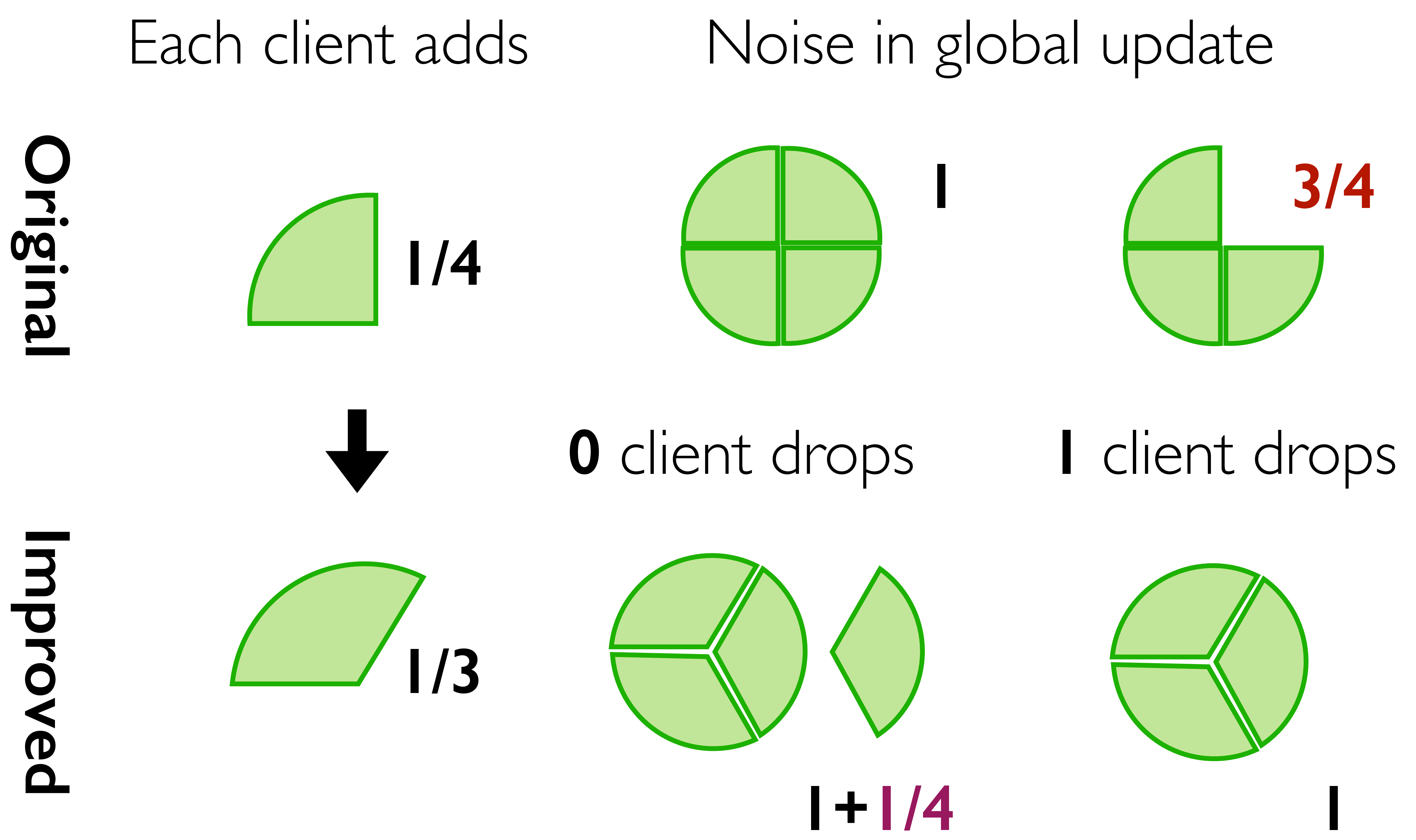

**Intuition** - Model Utility

- The **less** noise the **better**→**remove redundant** noise when dropout is settled

# Problem 2: Noise Deficiency

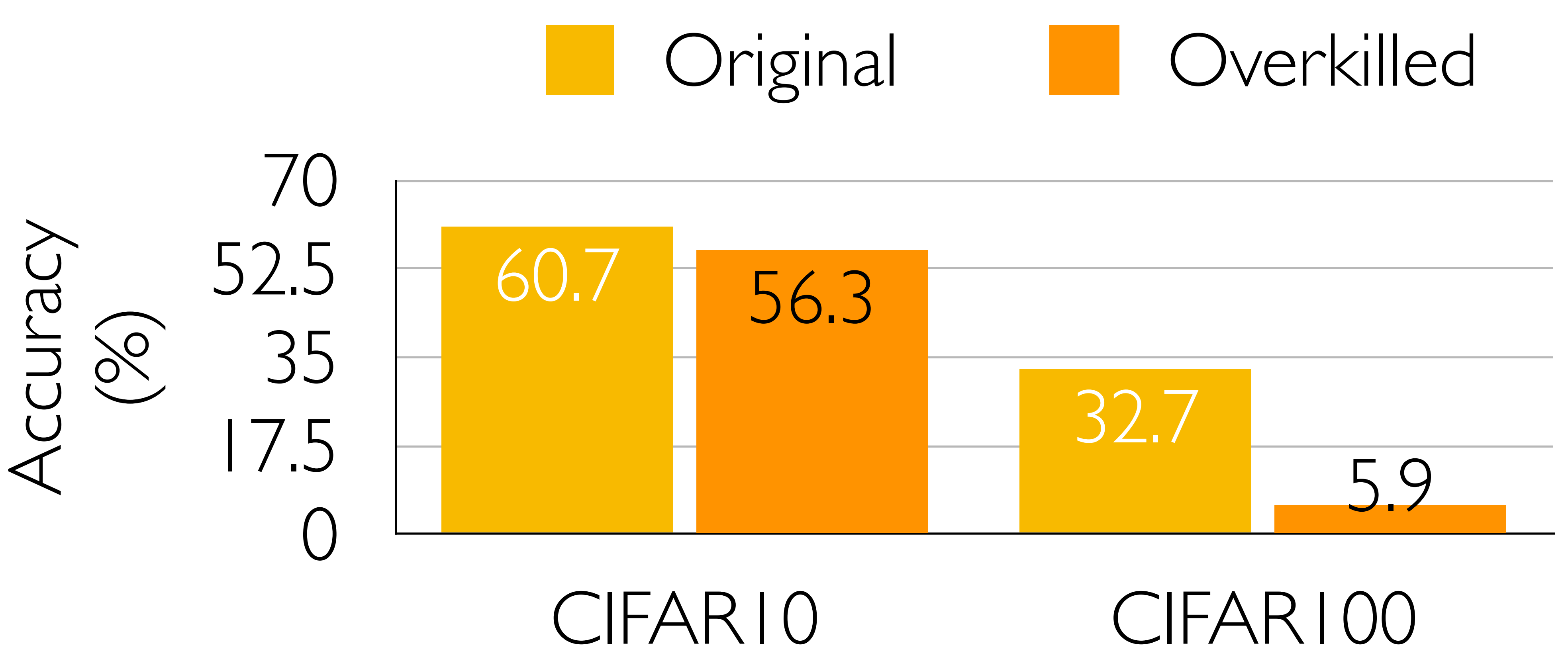
**Intuition** - Data privacy

- Noise should **never** be **insufficient** → Proactively **add more** noise than needed




**Intuition** - Model Utility

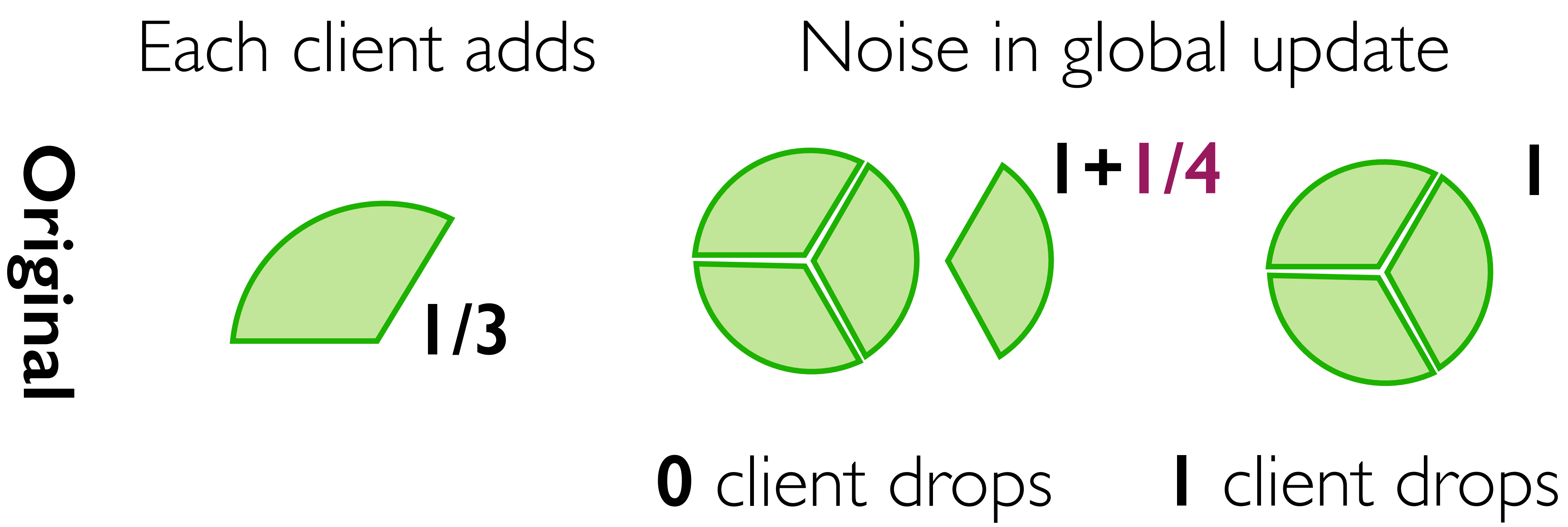
- The **less** noise the **better**→**remove redundant** noise when dropout is settled



# Problem 2: Noise Deficiency

**Potential approach**

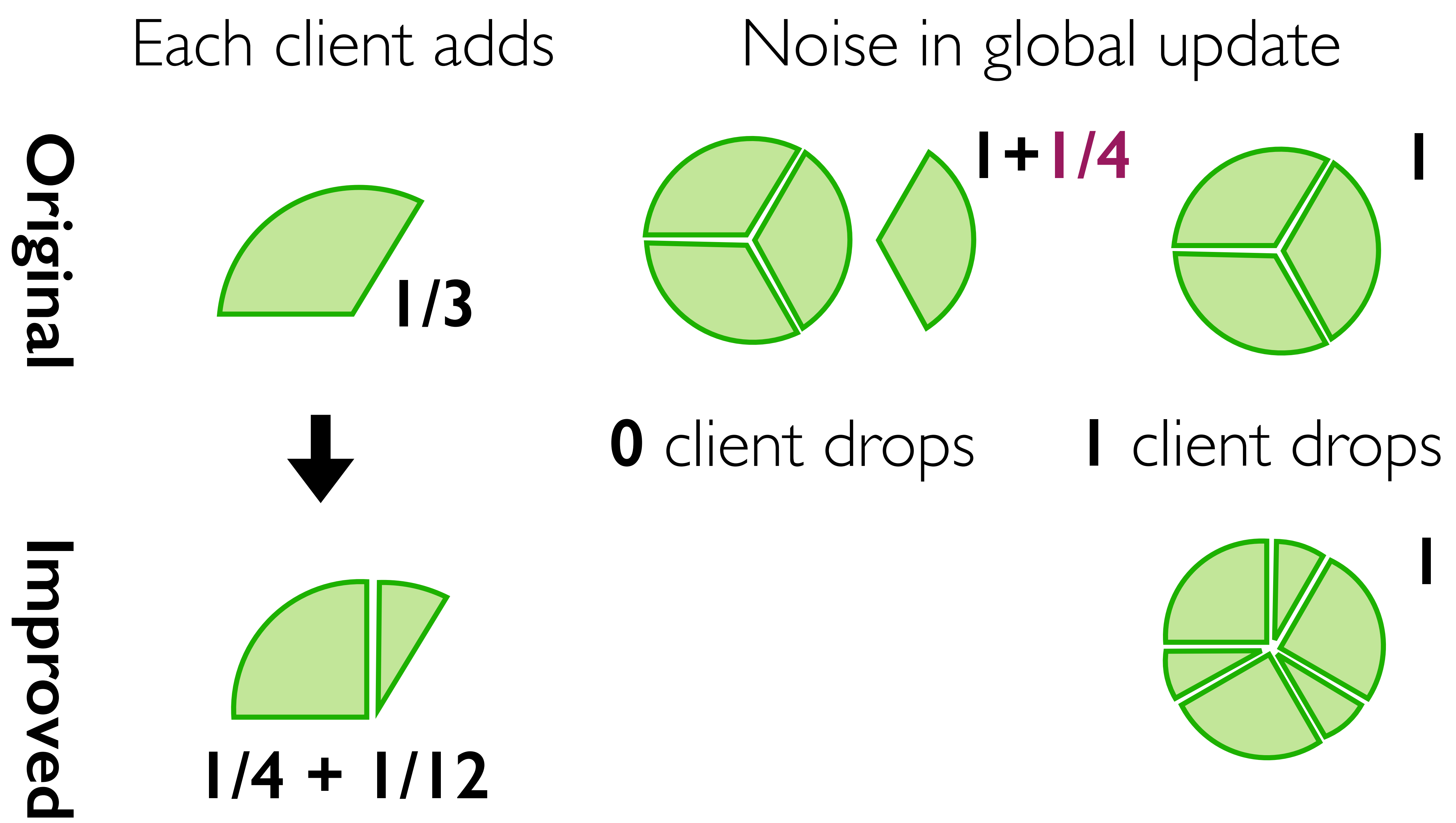
- Noise **decomposition** during addition



# Problem 2: Noise Deficiency

**Potential approach**

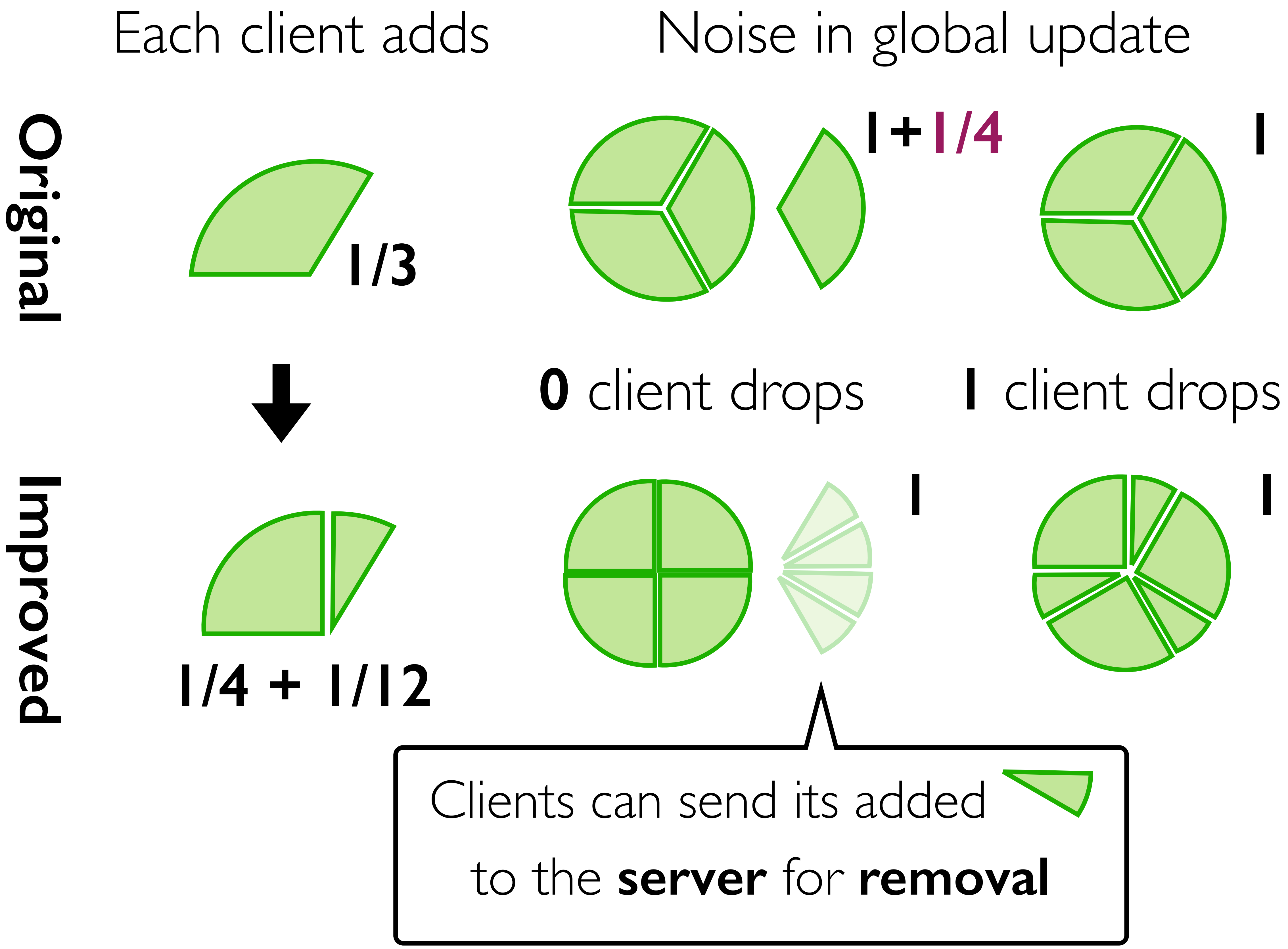
- Noise **decomposition** during addition



# Problem 2: Noise Deficiency

**Potential approach**

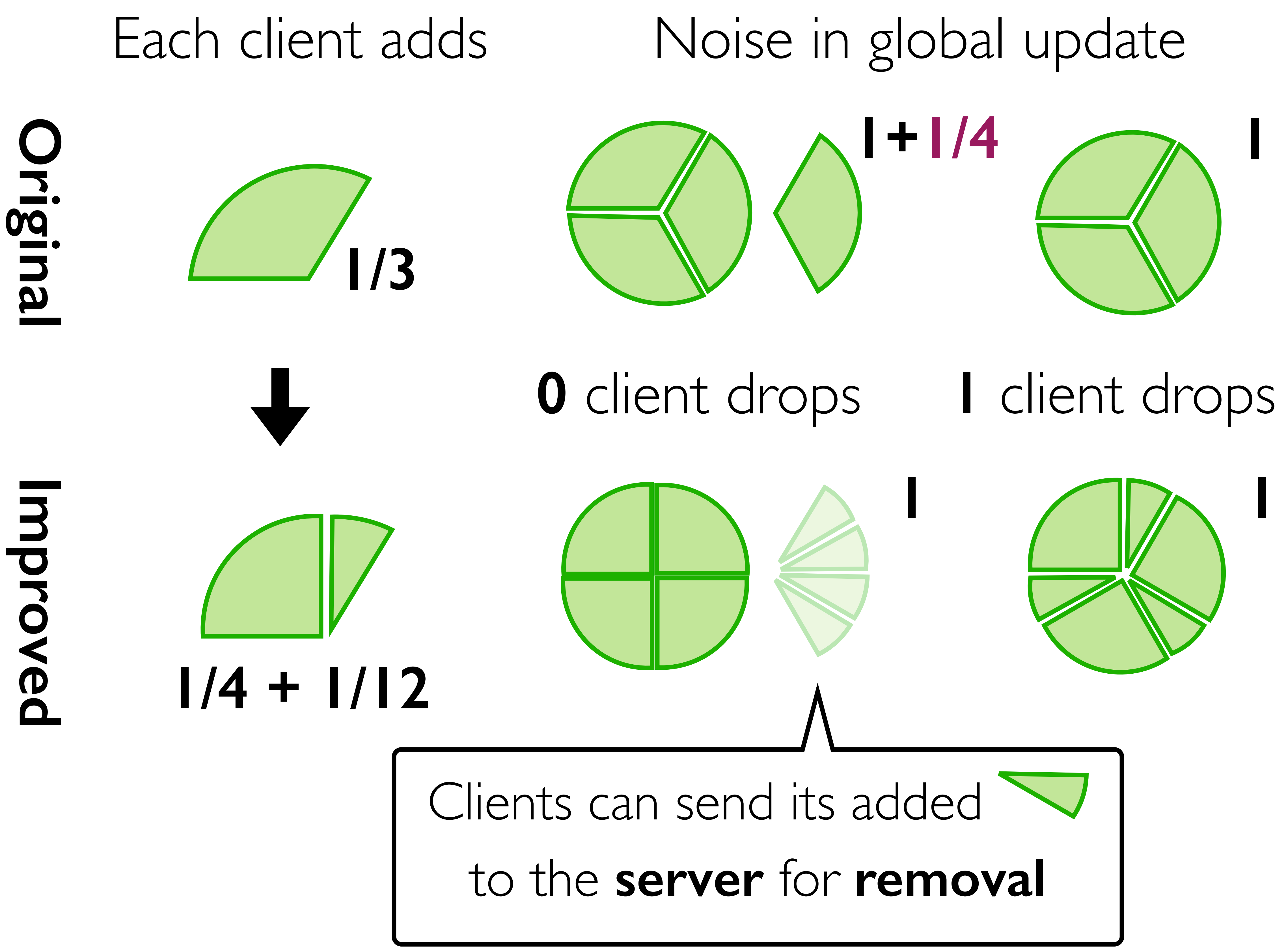
- Noise **decomposition** during addition



# Problem 2: Noise Deficiency

## Potential approach

- Noise **decomposition** during addition

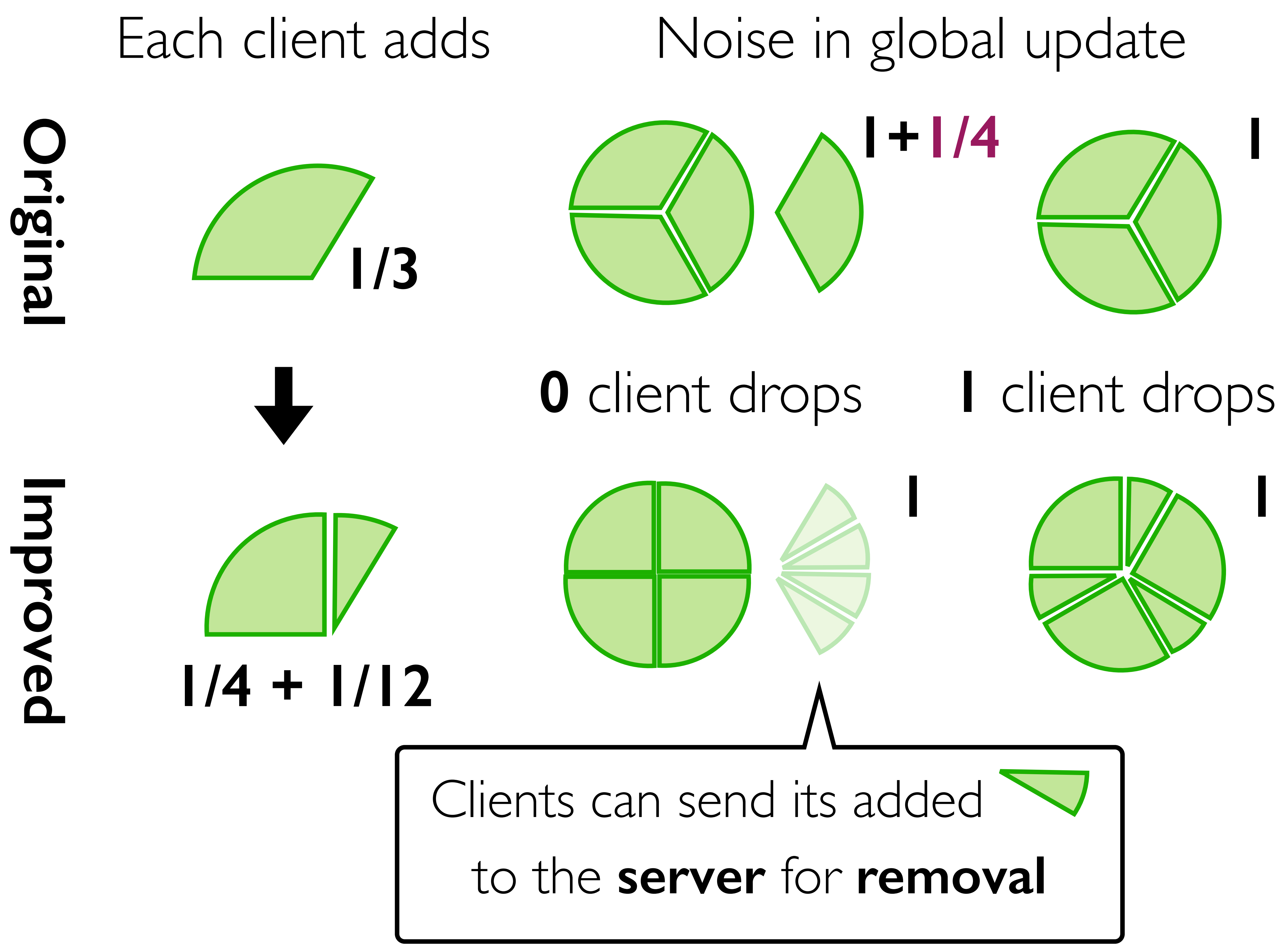



**Solution**: **Generalized** design for noise decomposition

# Problem 2: Noise Deficiency

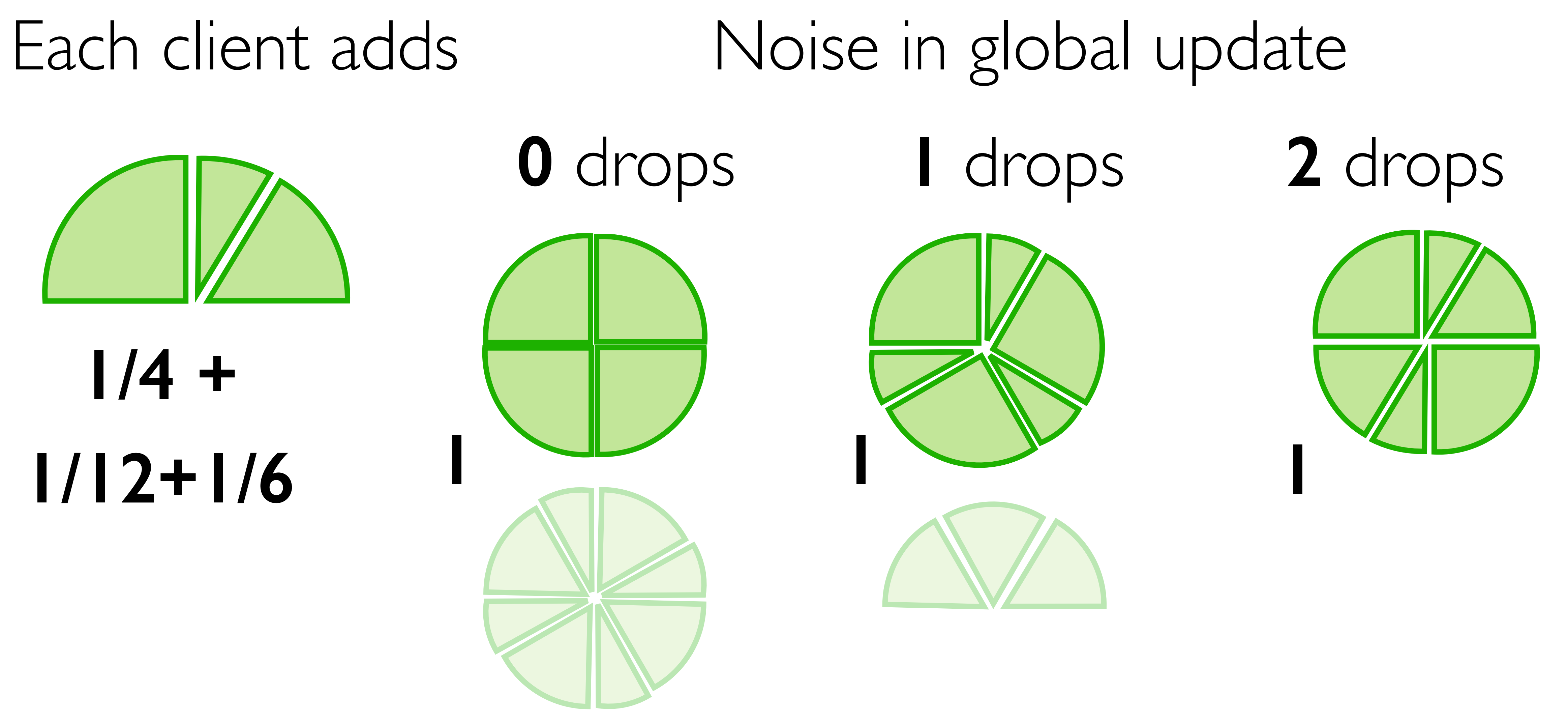
**Potential approach**

- Noise **decomposition** during addition




**Solution**: **Generalized** design for noise decomposition

E.g., 4 clients again, but tolerate up to **2** dropped clients



# Problem 2: Noise Deficiency

## Closed-form method

- **Noise addition**: Decompose Client $i$'s added noise $n_i \sim \chi\left(\frac{\sigma_*^2}{|S| - t}\right)$ into $t+1$ components: $n_i = \sum_{k=0}^{t} n_{i,k}$, $n_{i,0} \sim \chi\left(\frac{\sigma_*^2}{|S|}\right)$, and $n_{i,k} \sim \chi\left(\frac{\sigma_*^2}{(|S| - k + 1)(|S| - k)}\right)$ $(k \in [t])$
- **Noise removal**: when $|D|$ clients drop out, the noise components $n_{i,k}$ contributed by the surviving clients $i \in S \backslash D$ with the index $k > |D|$ becomes excessive and is removed by the server

# Dordis enforces the target noise

## Closed-form method

- **Noise addition**: Decompose Client $i$'s added noise $n_i \sim \chi\left(\frac{\sigma_*^2}{|S|-t}\right)$ into $t+1$ components: $n_i = \sum_{k=0}^{t} n_{i,k}$, $n_{i,0} \sim \chi\left(\frac{\sigma_*^2}{|S|}\right)$, and $n_{i,k} \sim \chi\left(\frac{\sigma_*^2}{(|S|-k+1)(|S|-k)}\right)$ $(k \in [t])$
- **Noise removal**: when $|D|$ clients drop out, the noise components $n_{i,k}$ contributed by the surviving clients $i \in S \backslash D$ with the index $k > |D|$ becomes excessive and is removed by the server

**Guarantee**: **Dordis enforces the target noise** when all are semi-honest

# Dordis enforces the target noise

## Closed-form method

- **Noise addition**: Decompose Client $i$'s added noise $n_i \sim \chi\left(\frac{\sigma_*^2}{|S|-t}\right)$ into $t+1$ components: $n_i = \sum_{k=0}^{t} n_{i,k}$, $n_{i,0} \sim \chi\left(\frac{\sigma_*^2}{|S|}\right)$, and $n_{i,k} \sim \chi\left(\frac{\sigma_*^2}{(|S|-k+1)(|S|-k)}\right)$ $(k \in [t])$
- **Noise removal**: when $|D|$ clients drop out, the noise components $n_{i,k}$ contributed by the surviving clients $i \in S \backslash D$ with the index $k > |D|$ becomes excessive and is removed by the server

**Guarantee**: **Dordis** enforces the target noise when all are **semi-honest**, or when even the server is **malicious**

Please find more in the paper :)

# Dordis enforces the target noise

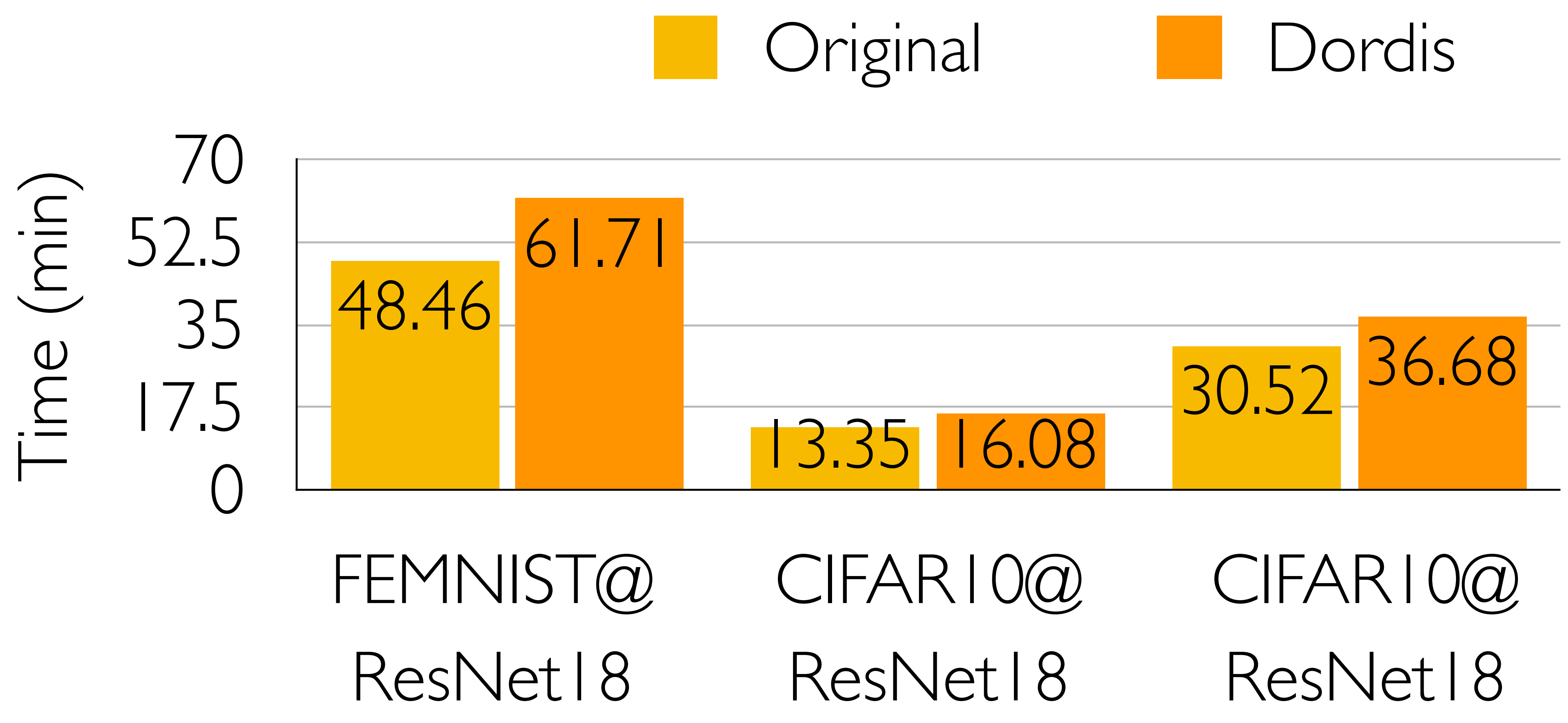
## Closed-form method

- **Noise addition**: Decompose Client $i$'s added noise $n_i \sim \chi\left(\frac{\sigma_*^2}{|S|-t}\right)$ into $t+1$ components: $n_i = \sum_{k=0}^{t} n_{i,k}$, $n_{i,0} \sim \chi\left(\frac{\sigma_*^2}{|S|}\right)$, and $n_{i,k} \sim \chi\left(\frac{\sigma_*^2}{(|S|-k+1)(|S|-k)}\right)$ $(k \in [t])$
- **Noise removal**: when $|D|$ clients drop out, the noise components $n_{i,k}$ contributed by the surviving clients $i \in S \backslash D$ with the index $k > |D|$ becomes excessive and is removed by the server

**Dordis runtime** overhead **≤34%**

**Guarantee**: **Dordis** enforces the target noise when all are **semi-honest**, or when even the server is **malicious**

Please find more in the paper :)



# Dordis: Results summary

**Efficiency**

**Substantial speedup** up to **2.4×** for **general** workloads

**Integration**

**Seamlessly** packed in one **comprehensive** system

**Resilience**

**Privacy preserved** with target noise **precisely** enforced **regardless** of client dropout

github.com/SamuelGong/Dordis

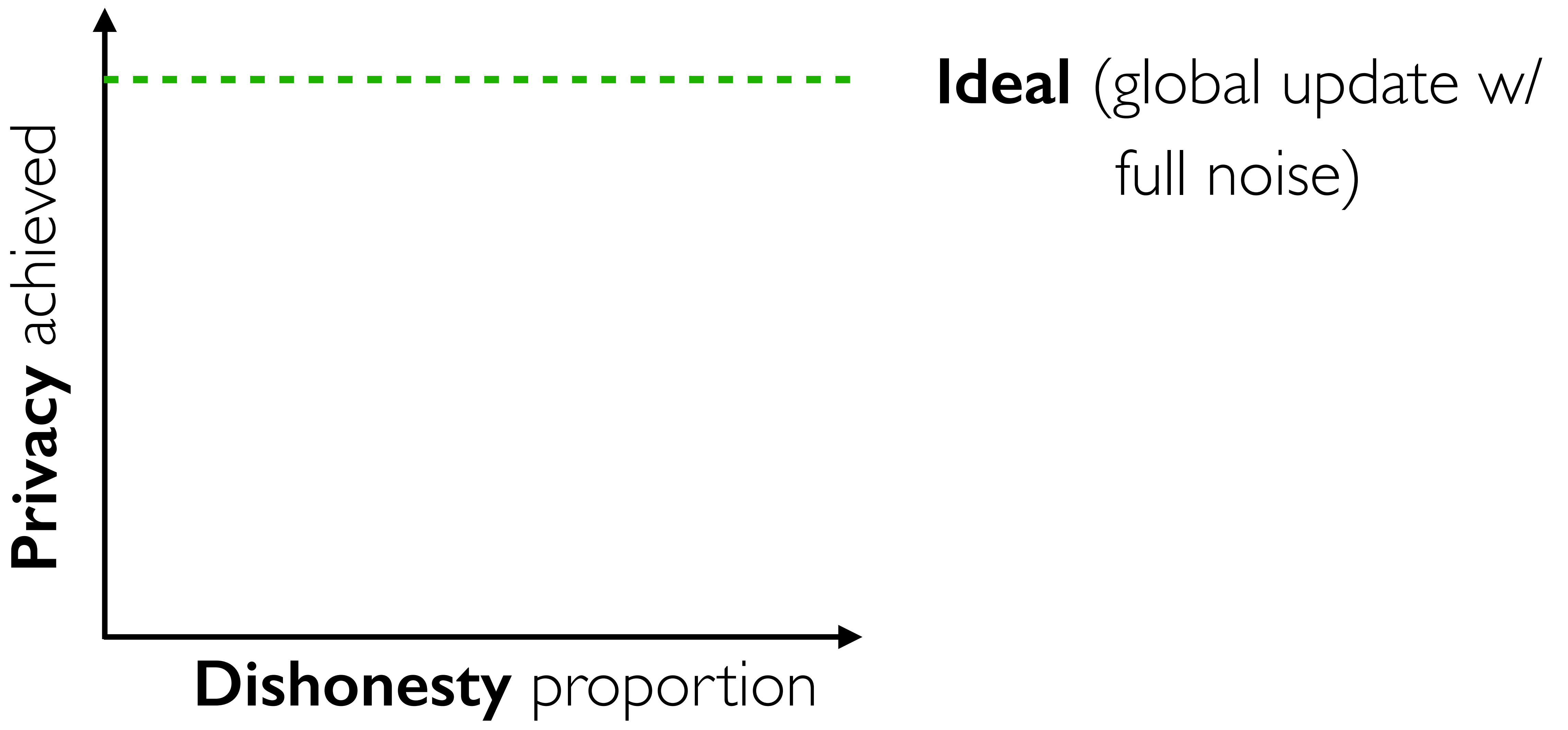
# Third work: Lotto [1]

| | | | |
|---|---|---|---|
| **Privacy** Worst-case defense… | Can be a dishonest majority | x ⟷ | Only or mostly works with honest participants |
| **Efficiency** Time-to-accuracy… | Stragglers bottleneck time | Primitives heavy in comp. and comm. | Client dropout yields insufficient noise |

| **Privacy-Enhancing Technique** | Federated Learning | Secure Aggregation | Differential Privacy |
|---|---|---|---|
| **Privacy Guarantee** | Data kept on premises | Local updates unseen | Global update leaks little about any client |

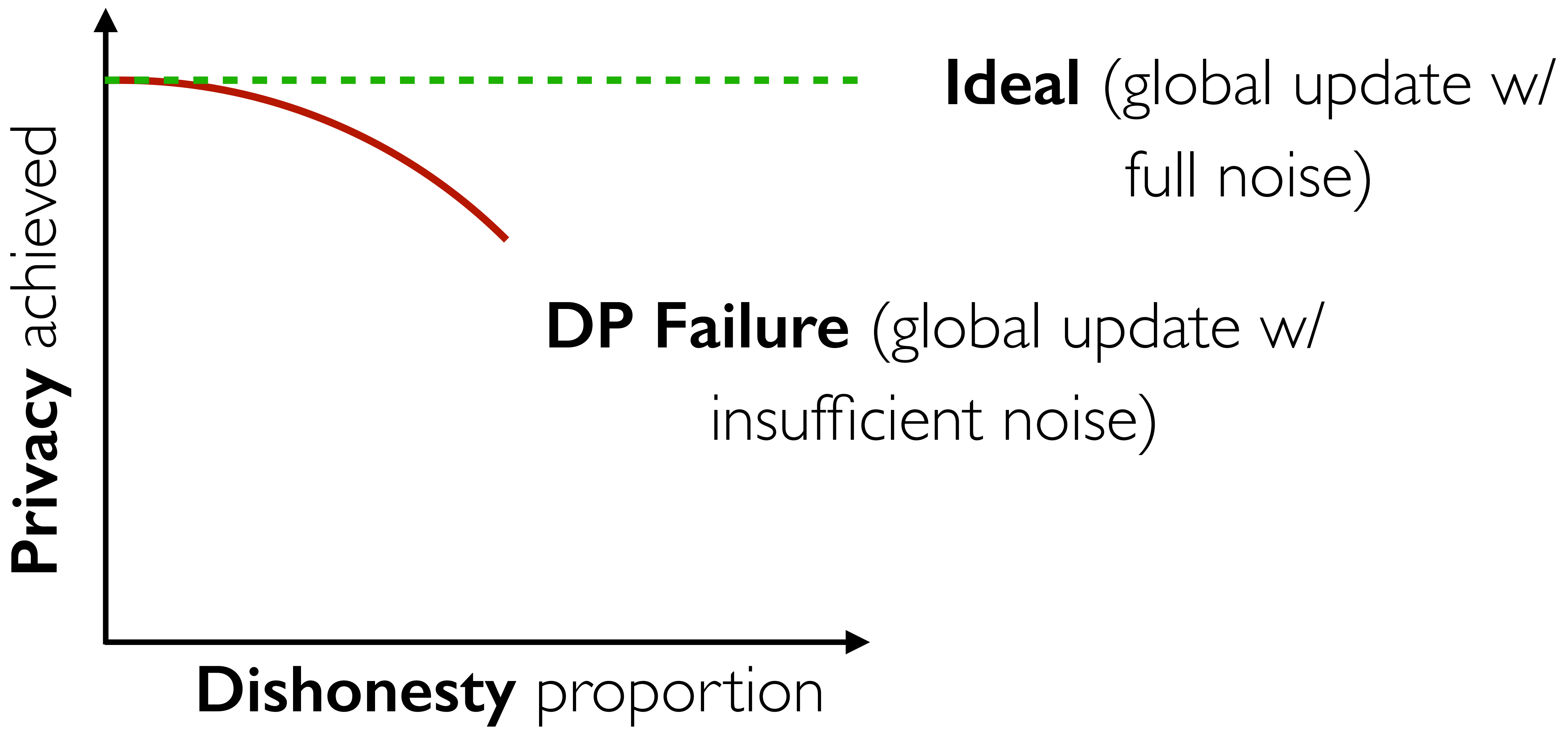
[1] Jiang et al. "Lotto: Secure Participant Selection against Adversarial Servers in Federated Learning", In Security '24

# Need for Lotto



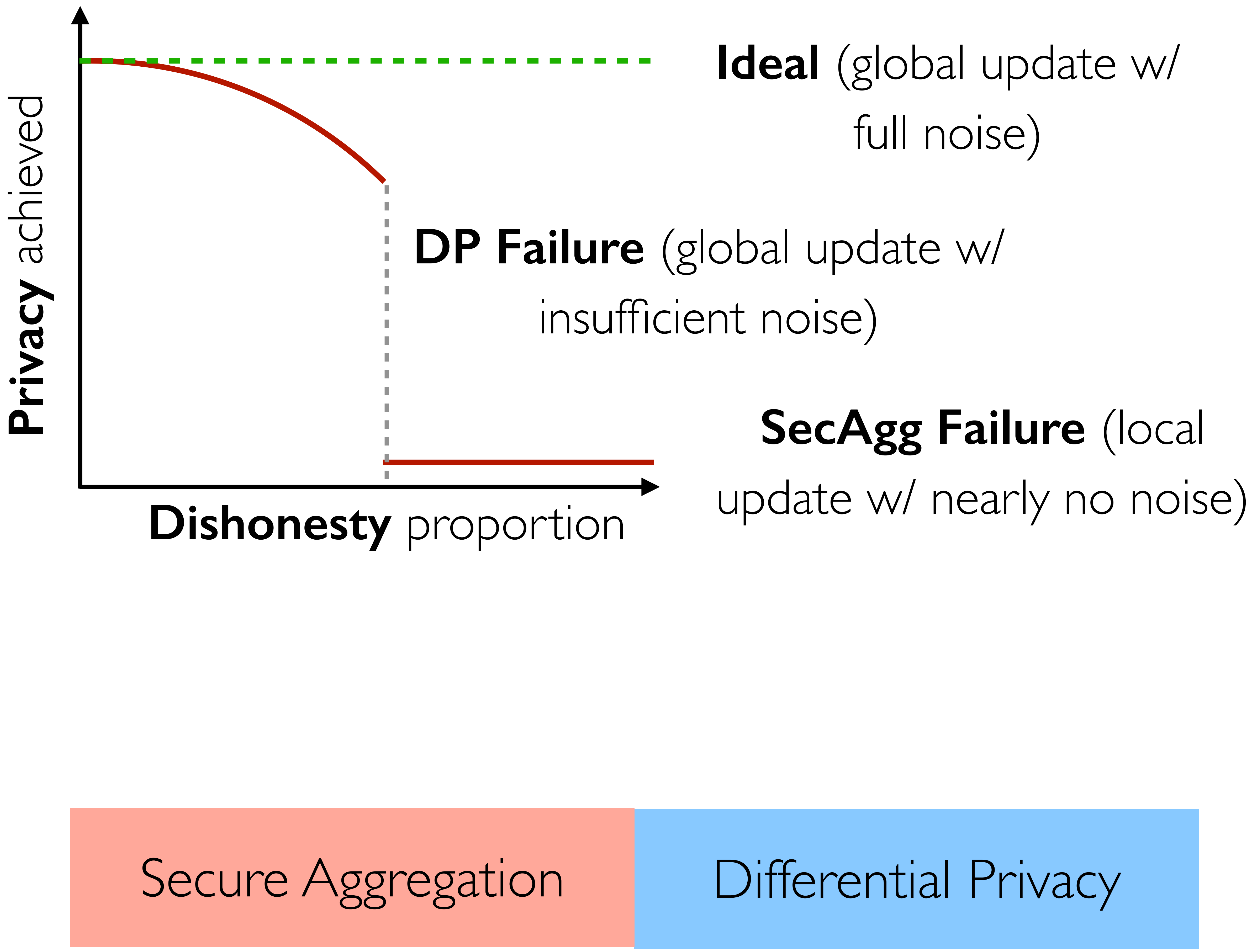

Secure Aggregation | Differential Privacy

# Need for Lotto

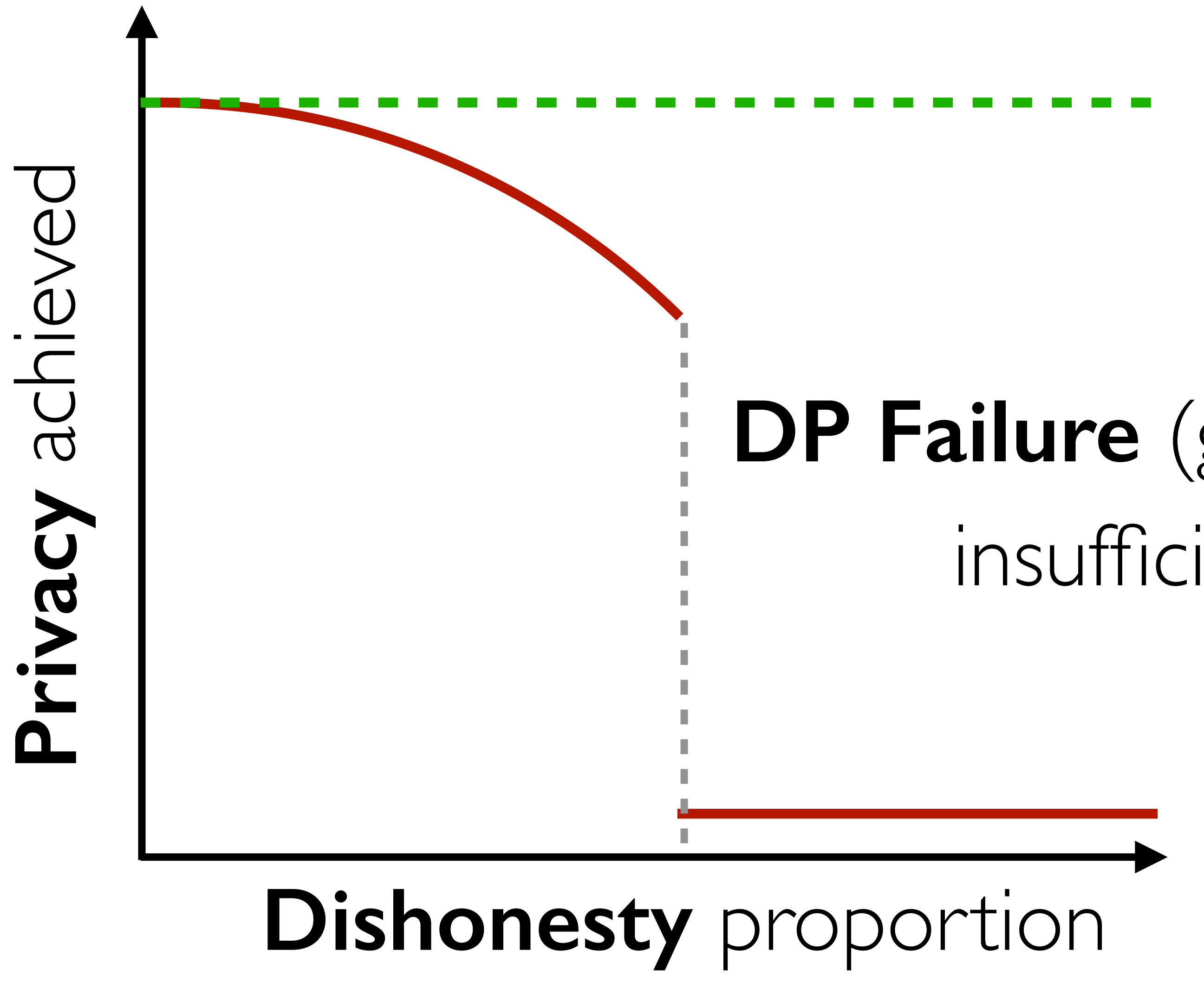



Secure Aggregation | Differential Privacy

# Need for Lotto



# Need for Lotto




**Ideal** (global update w/ full noise)

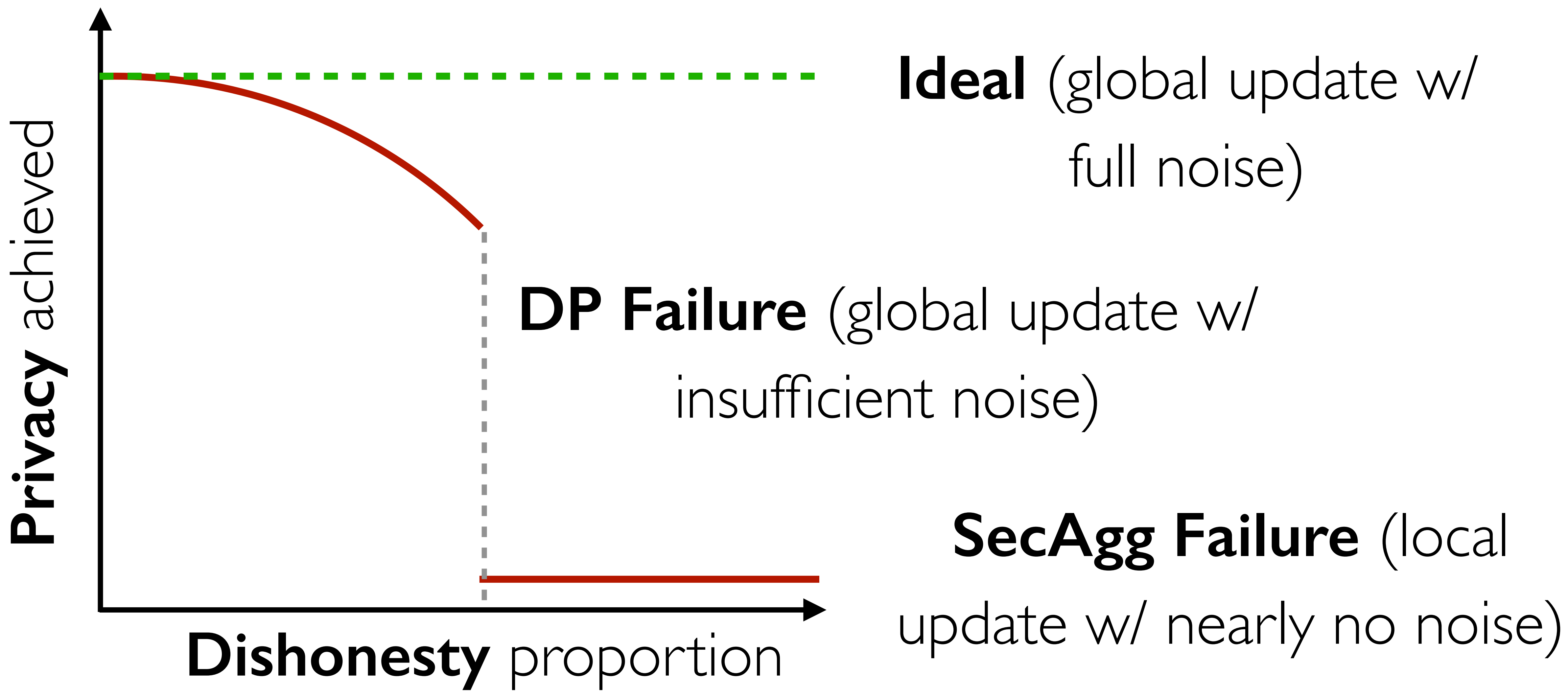
**DP Failure** (global update w/ insufficient noise)

**SecAgg Failure** (local update w/ nearly no noise)

**Assumption**: honest participants

| Secure Aggregation | Differential Privacy |
|---|---|

# Need for Lotto

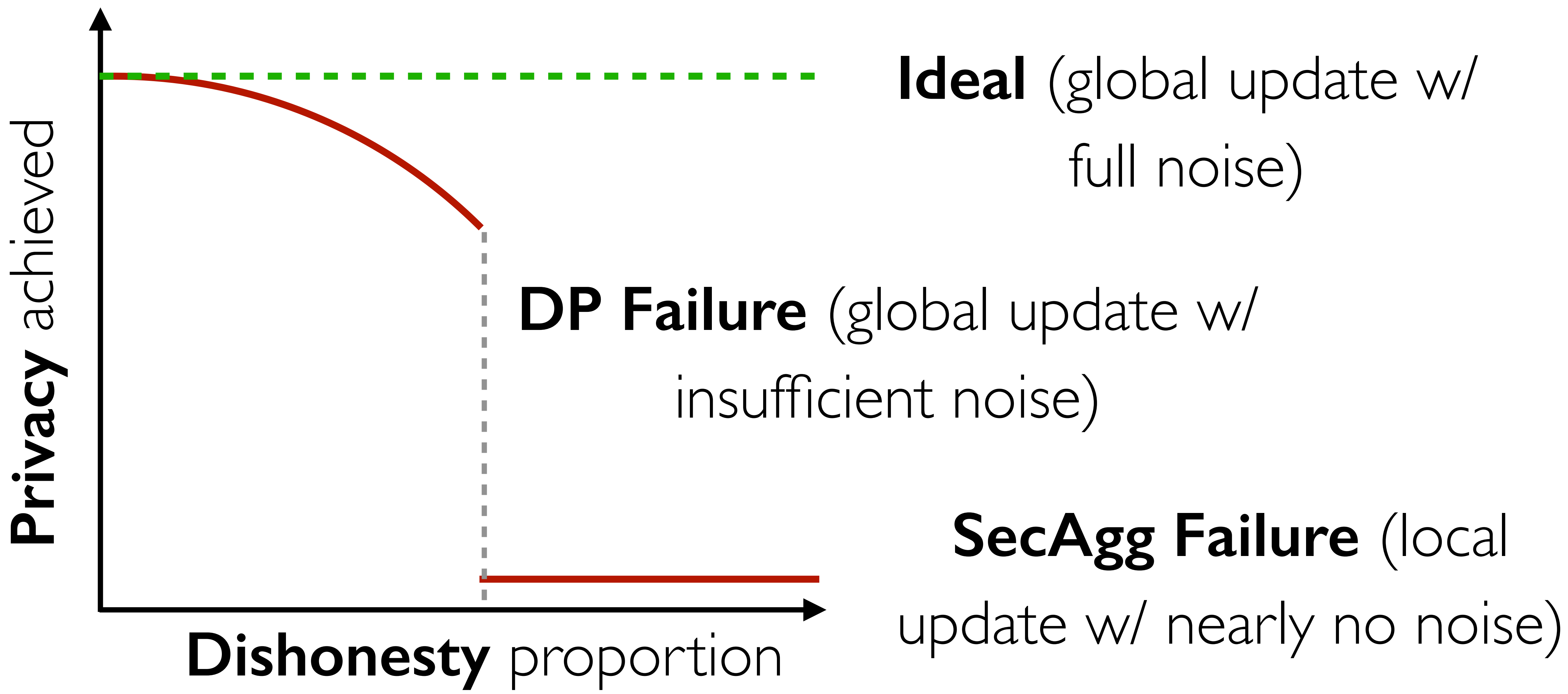



**Assumption**: honest participants

Secure Aggregation | Differential Privacy

Federated Learning

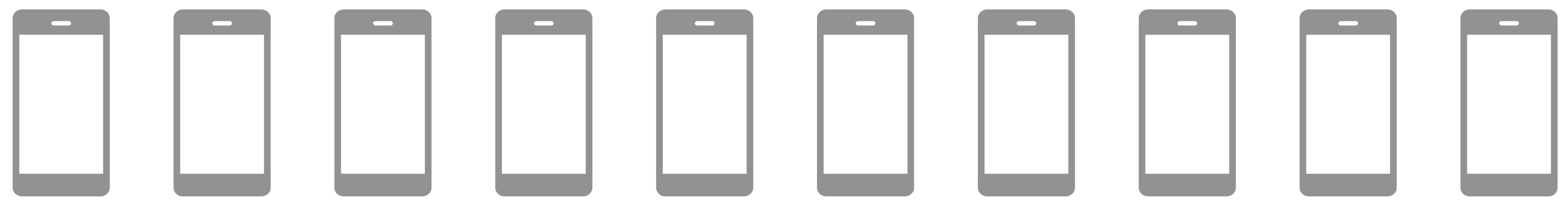
# Need for Lotto




**Assumption**: honest participants

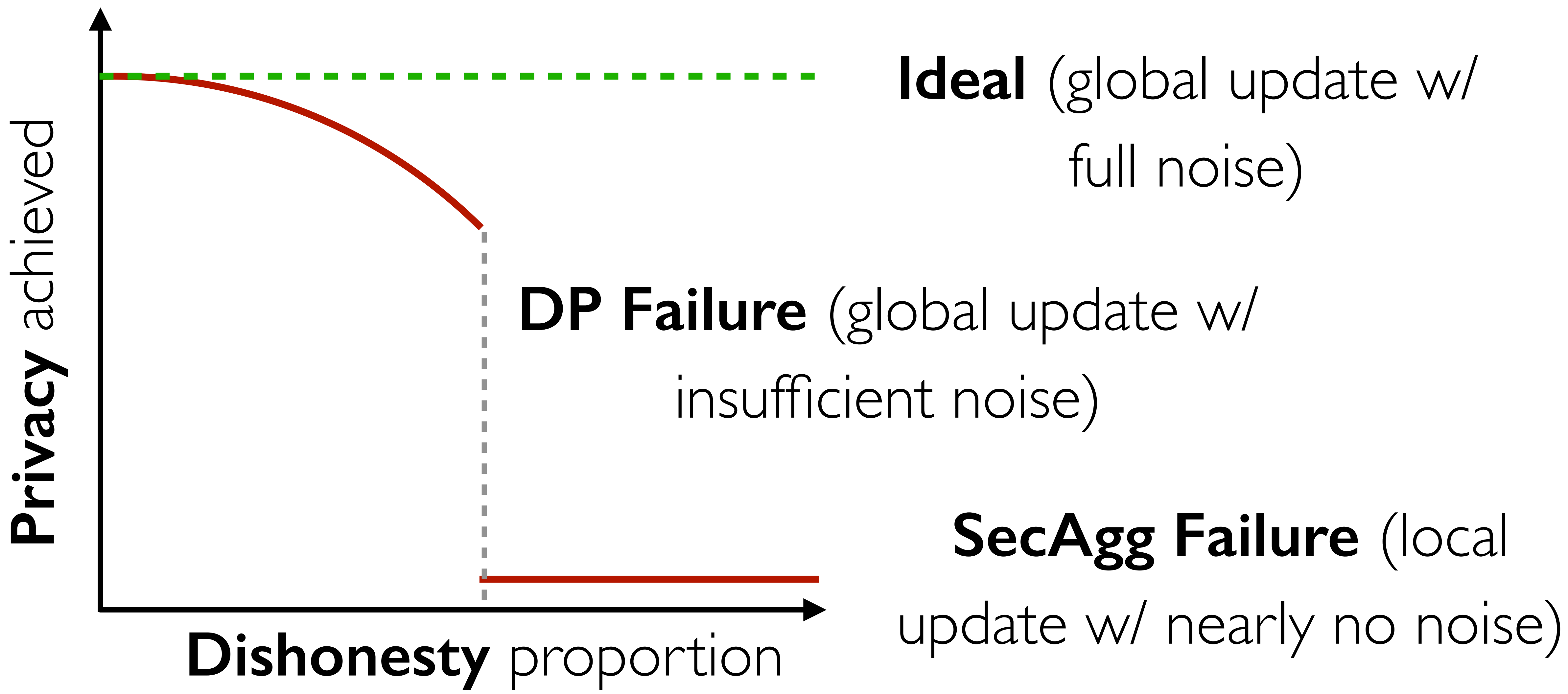
Secure Aggregation | Differential Privacy

**Population** ($10^4$ - $10^8$)



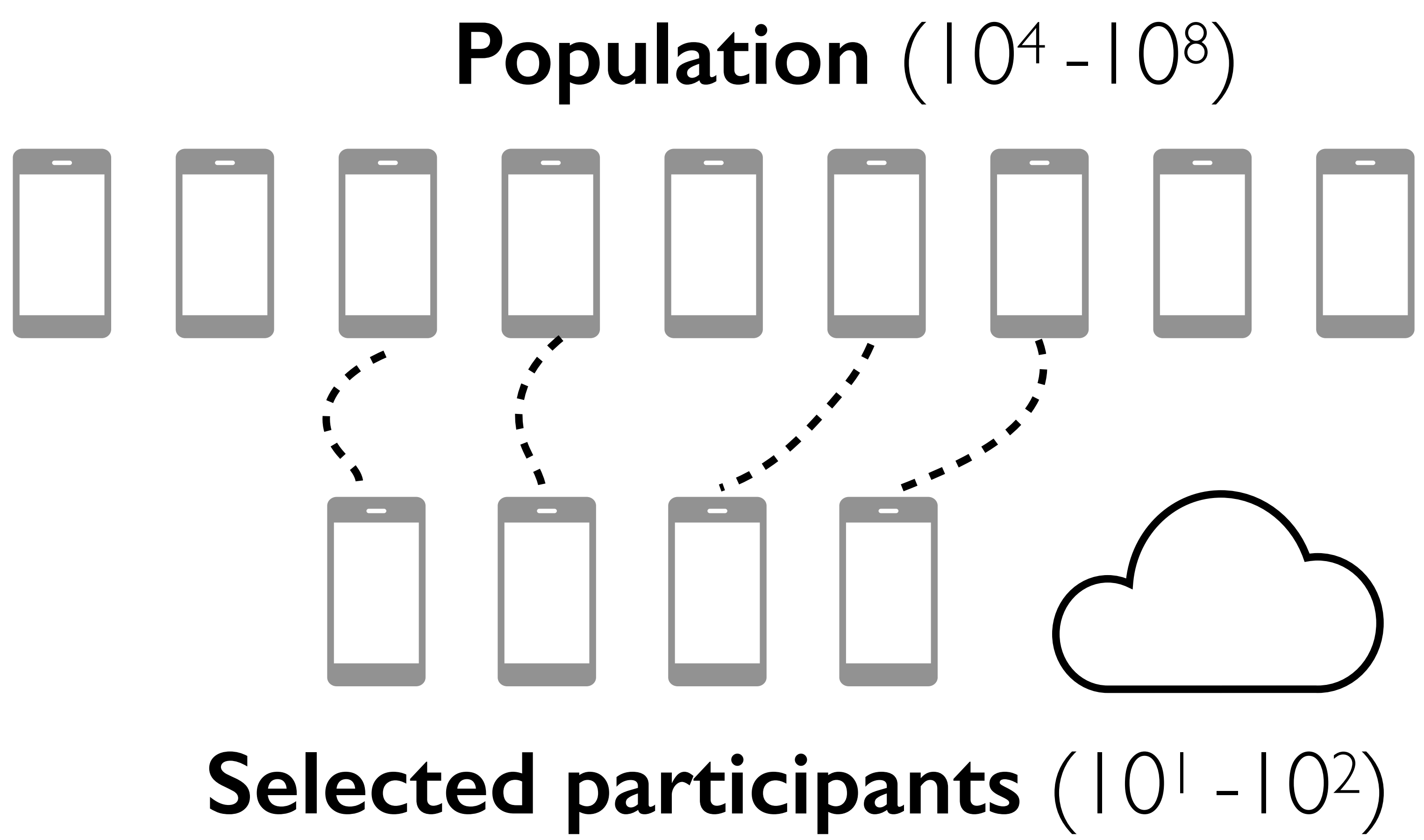
Federated Learning

# Need for Lotto



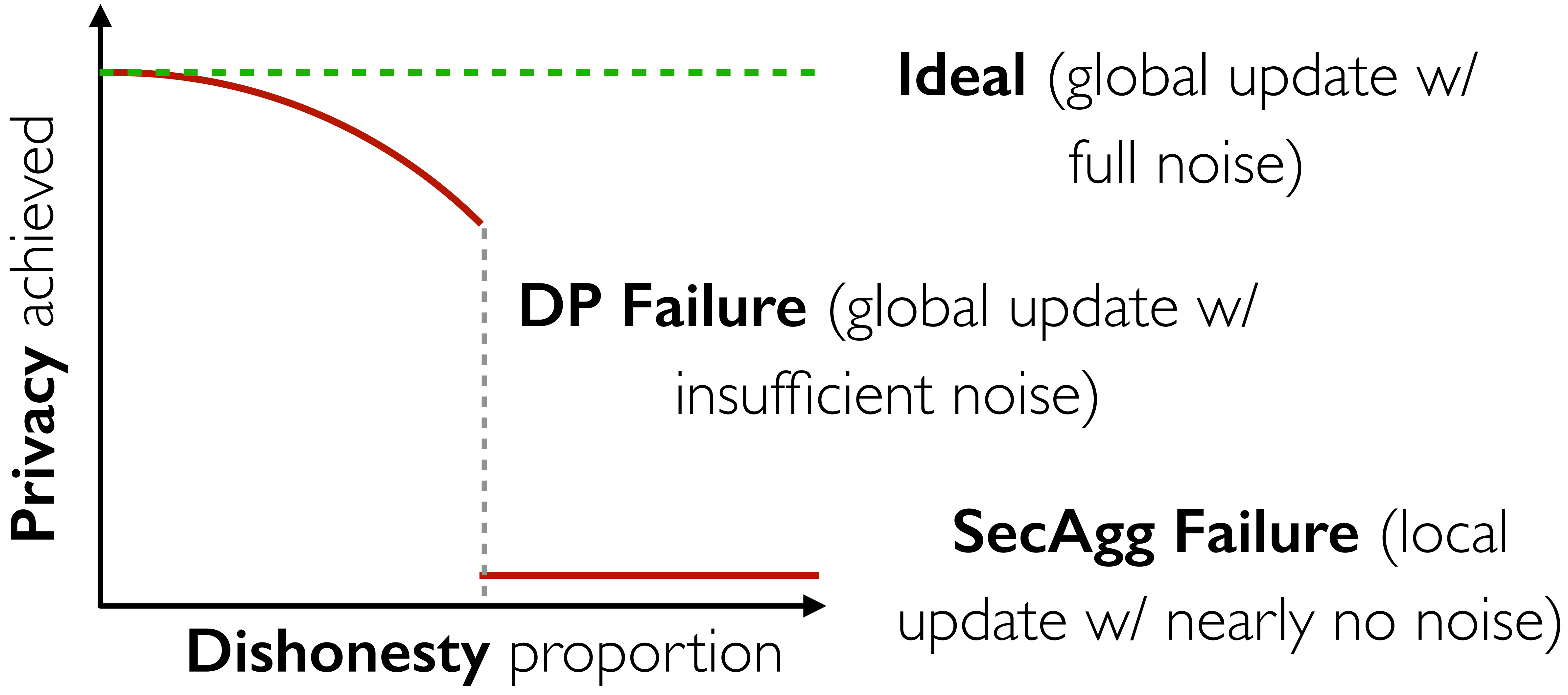

**Assumption**: honest participants

Secure Aggregation | Differential Privacy




Federated Learning

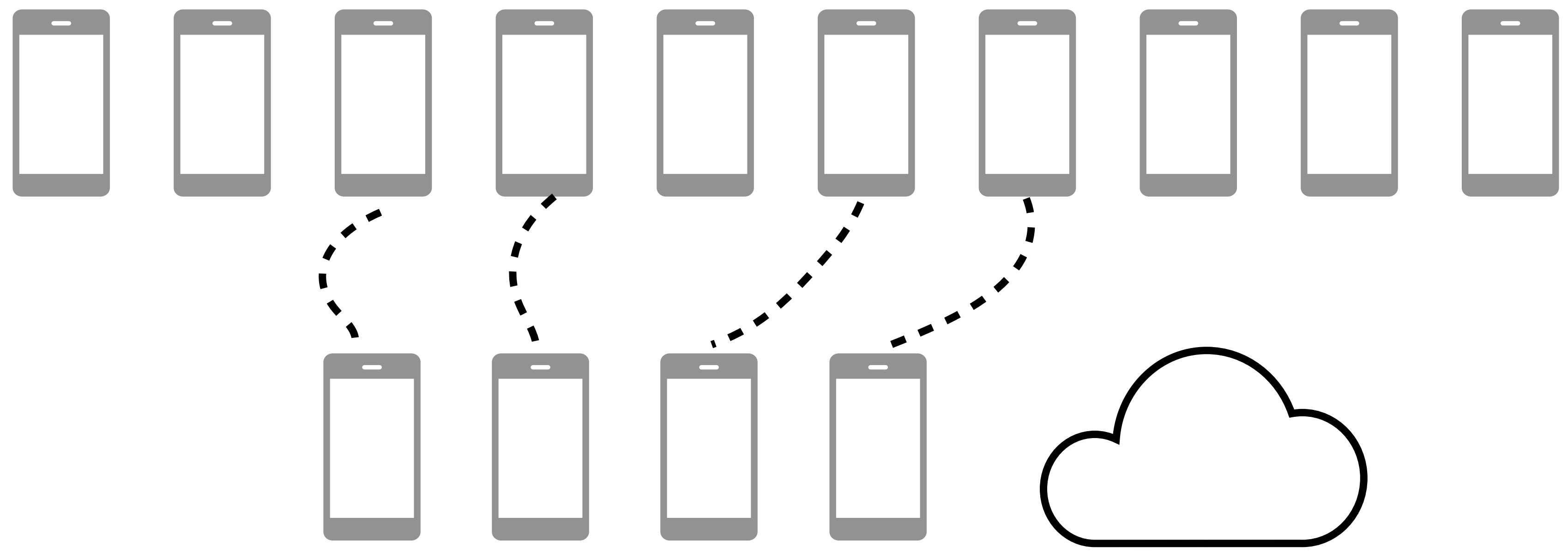
# Need for Lotto




**Assumption**: honest participants

Secure Aggregation | Differential Privacy

**Population** ($10^4$ - $10^8$)

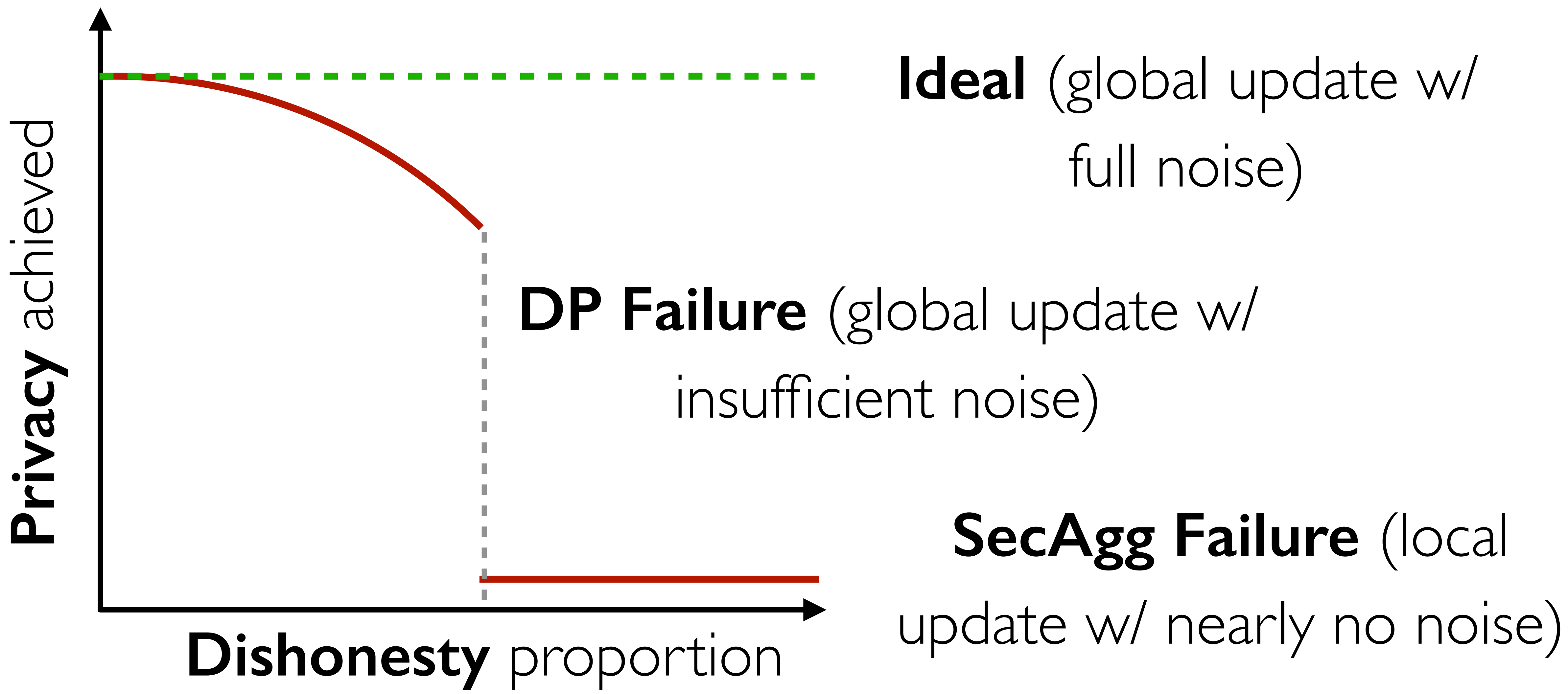


**Selected participants** ($10^1$ - $10^2$)

- **Random**: uniform chance

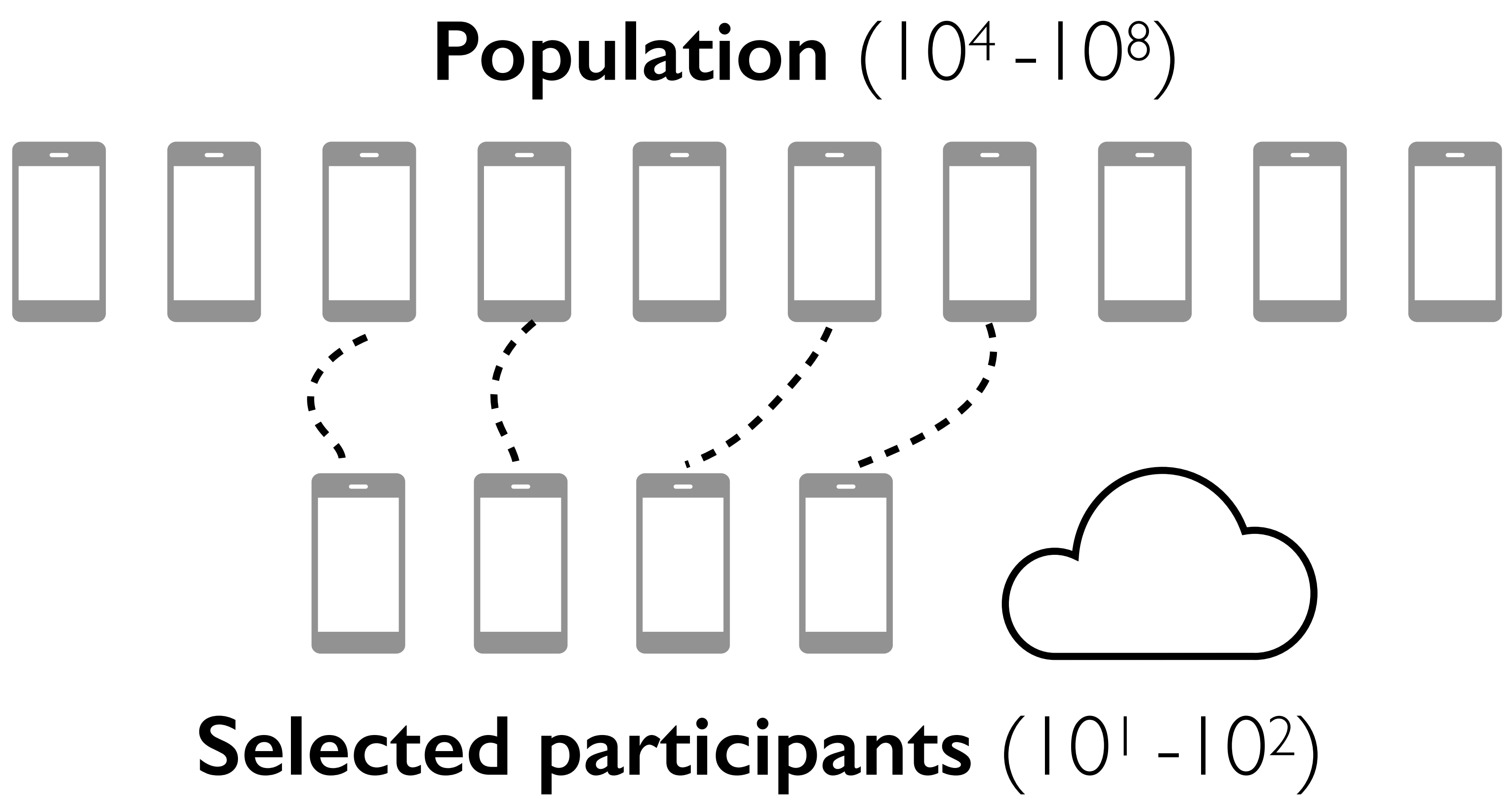
Federated Learning

# Need for Lotto




**Assumption**: honest participants

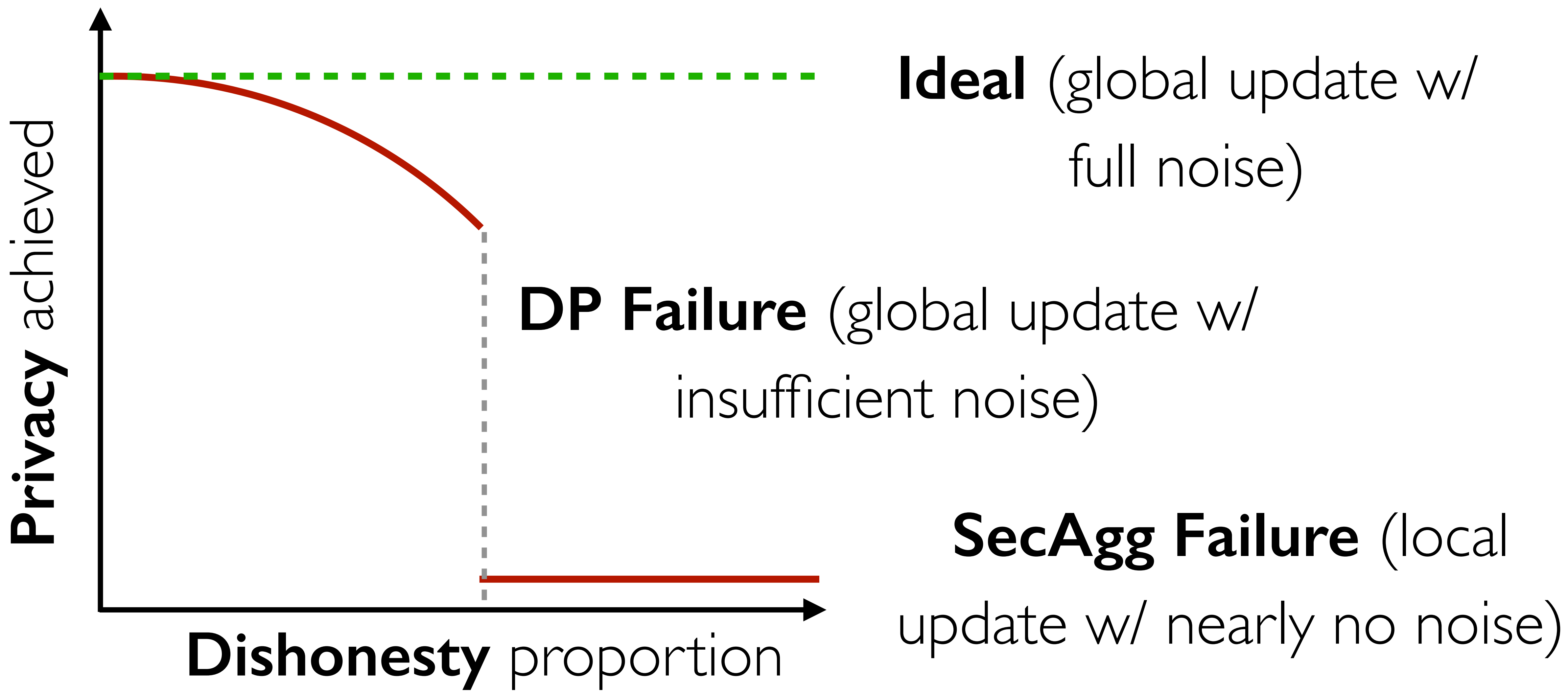
Secure Aggregation | Differential Privacy




- **Random**: uniform chance
- **Informed**: "best-performing" clients are preferred (e.g., high speed and/or rich data)

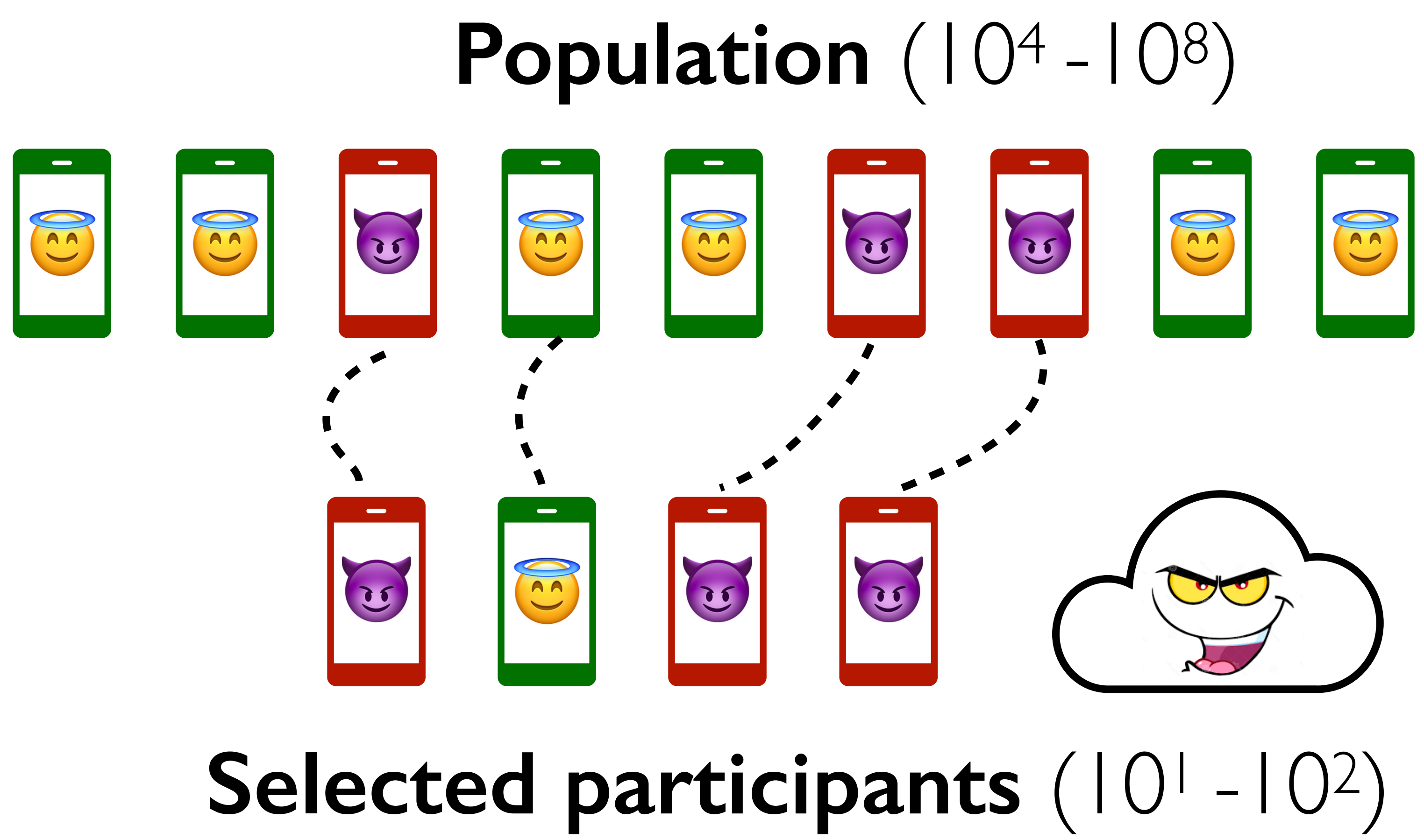
Federated Learning

# Need for Lotto




**Assumption**: honest participants

Secure Aggregation | Differential Privacy




**Problem**: participant selection can be **manipulated** by the **malicious** server

Federated Learning

# Lotto - Overview

# Lotto - Overview

**No peer-to-peer network**: all traffic relayed by the server

# Lotto - Overview

**No peer-to-peer network**: all traffic relayed by the server

Threat model: **malicious server colluding** with some clients, and a public key infrastructure (**PKI**)

# Lotto - Overview

**No peer-to-peer network**: all traffic relayed by the server

Threat model: **malicious server colluding** with some clients, and a public key infrastructure (**PKI**)

**Functionality**

Support both **random** and **informed** selection

# Lotto - Overview

**No peer-to-peer network**: all traffic relayed by the server

Threat model: **malicious server colluding** with some clients, and a public key infrastructure (**PKI**)

**Functionality**

Support both **random** and **informed** selection

**Security**

**Theoretical guarantee** of preventing manipulation

# Lotto - Overview

**No peer-to-peer network**: all traffic relayed by the server

Threat model: **malicious server colluding** with some clients, and a public key infrastructure (**PKI**)

**Functionality**

Support both **random** and **informed** selection

**Security**

**Theoretical guarantee** of preventing manipulation

**Efficiency**

Mild **runtime overhead** with no **network cost**

# Problem: Random selection

# Problem: Random selection

Current round: 2

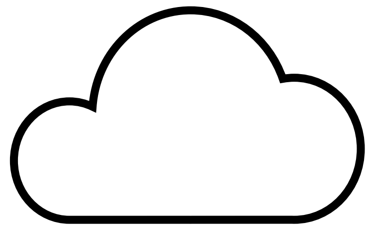
| | Randomness |
|---|---|
| #1 | $\mathbf{RF}_{pk1}(2) = 9$ |
| #2 | $\mathbf{RF}_{pk2}(2) = 1$ |
| #3 | $\mathbf{RF}_{pk3}(2) = 7$ |
| … | … |

Public keys

Selection criteria: <3

# Problem: Random selection

Current round: 2

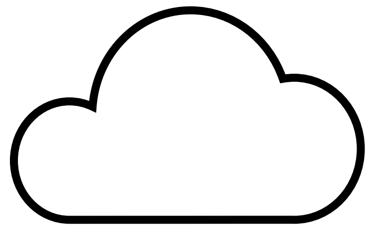


| | Randomness | Select |
|---|---|---|
| #1 | $\mathbf{RF}_{pk1}(2) = 9$ | No |
| #2 | $\mathbf{RF}_{pk2}(2) = 1$ | Yes |
| #3 | $\mathbf{RF}_{pk3}(2) = 7$ | No |
| ... | ... | ... |

Selection criteria: <3

# Problem: Random selection

Current round: 2



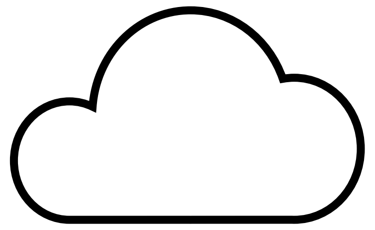
| | Randomness | Select | Randomness | Select |
|---|---|---|---|---|
| #1 | $\mathbf{RF}_{pk1}(2) = 9$ | No | | Yes |
| #2 | $\mathbf{RF}_{pk2}(2) = 1$ | Yes | Does NOT matter. | No |
| #3 | $\mathbf{RF}_{pk3}(2) = 7$ | No | | No |
| … | … | … | | … |

Selection criteria: <3

For dishonest majority

# Problem: Random selection

Current round: 2



| | Randomness | Select | Randomness | Select |
|---|---|---|---|---|
| #1 | $\mathbf{RF}_{pk1}(2) = 9$ | No | | Yes |
| #2 | $\mathbf{RF}_{pk2}(2) = 1$ | Yes | Does NOT matter. | No |
| #3 | $\mathbf{RF}_{pk3}(2) = 7$ | No | | No |
| … | … | … | | … |

Selection criteria: <3

For dishonest majority

**Potential approach**:

- Outcome verification

# Problem: Random selection

Current round: 2

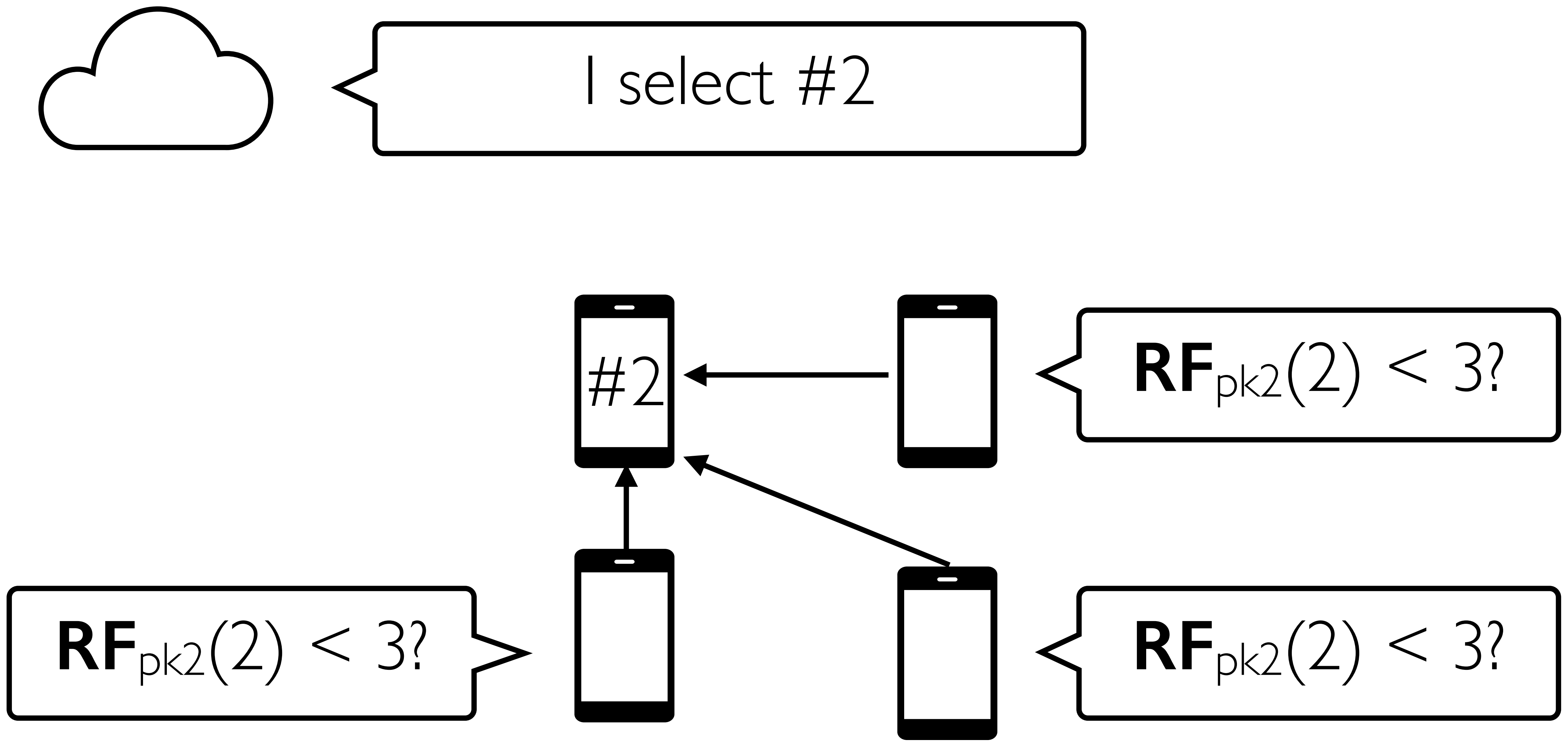
| | Randomness | Select | Randomness | Select |
|---|---|---|---|---|
| #1 | $\mathbf{RF}_{pk1}(2) = 9$ | No | | Yes |
| #2 | $\mathbf{RF}_{pk2}(2) = 1$ | Yes | Does NOT matter. | No |
| #3 | $\mathbf{RF}_{pk3}(2) = 7$ | No | | No |
| … | … | … | | … |

Selection criteria: <3

For dishonest majority

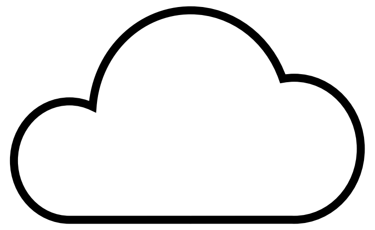
**Potential approach**:

- Outcome verification



# Problem: Random selection

Current round: 2



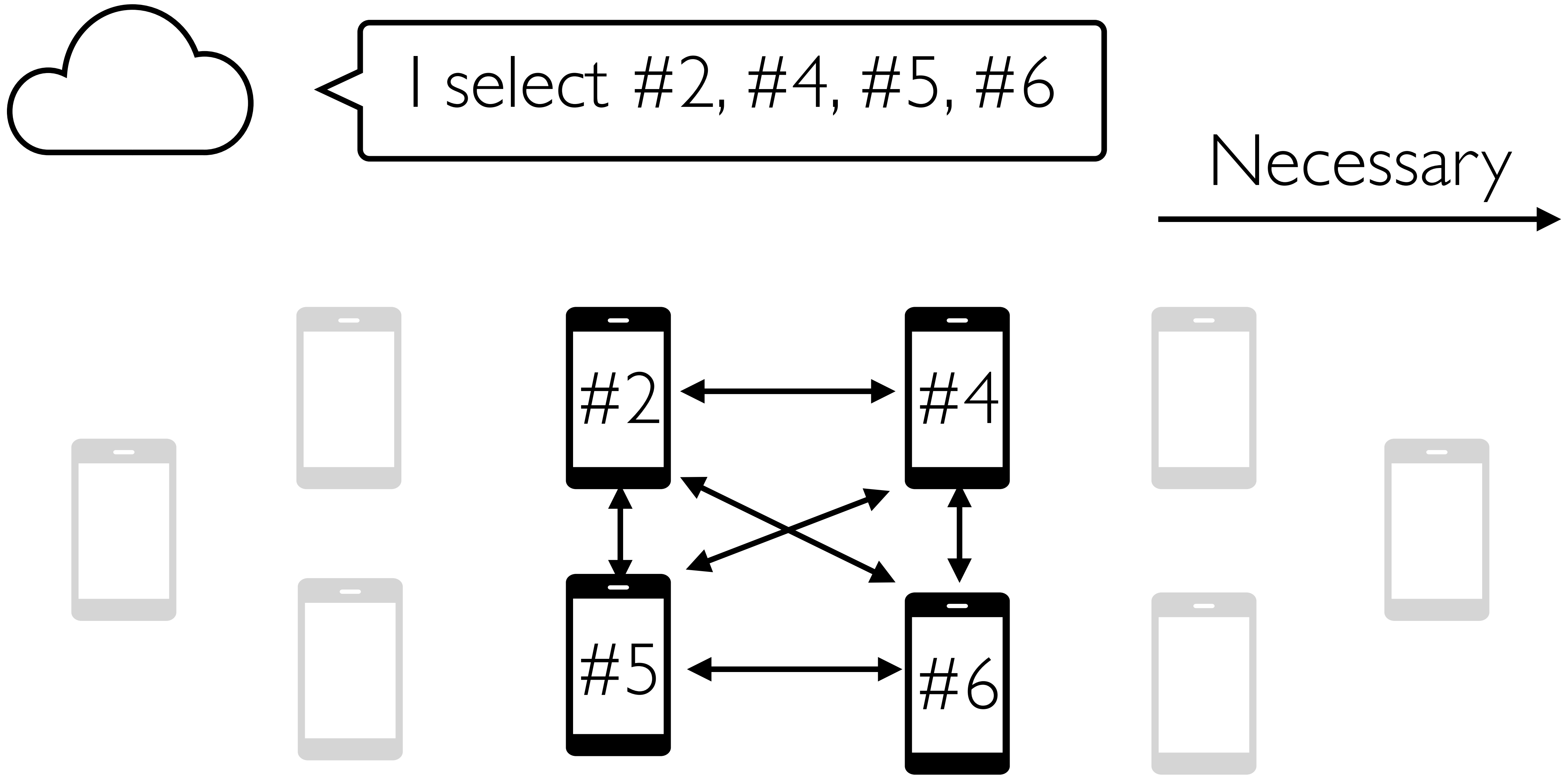
| | Randomness | Select | Randomness | Select |
|---|---|---|---|---|
| #1 | $\mathbf{RF}_{pk1}(2) = 9$ | No | | Yes |
| #2 | $\mathbf{RF}_{pk2}(2) = 1$ | Yes | Does NOT matter. | No |
| #3 | $\mathbf{RF}_{pk3}(2) = 7$ | No | | No |
| … | … | … | | … |

Selection criteria: <3

For dishonest majority

**Potential approach**:

- Outcome verification
- Only within participants ($10^1$ - $10^2$)



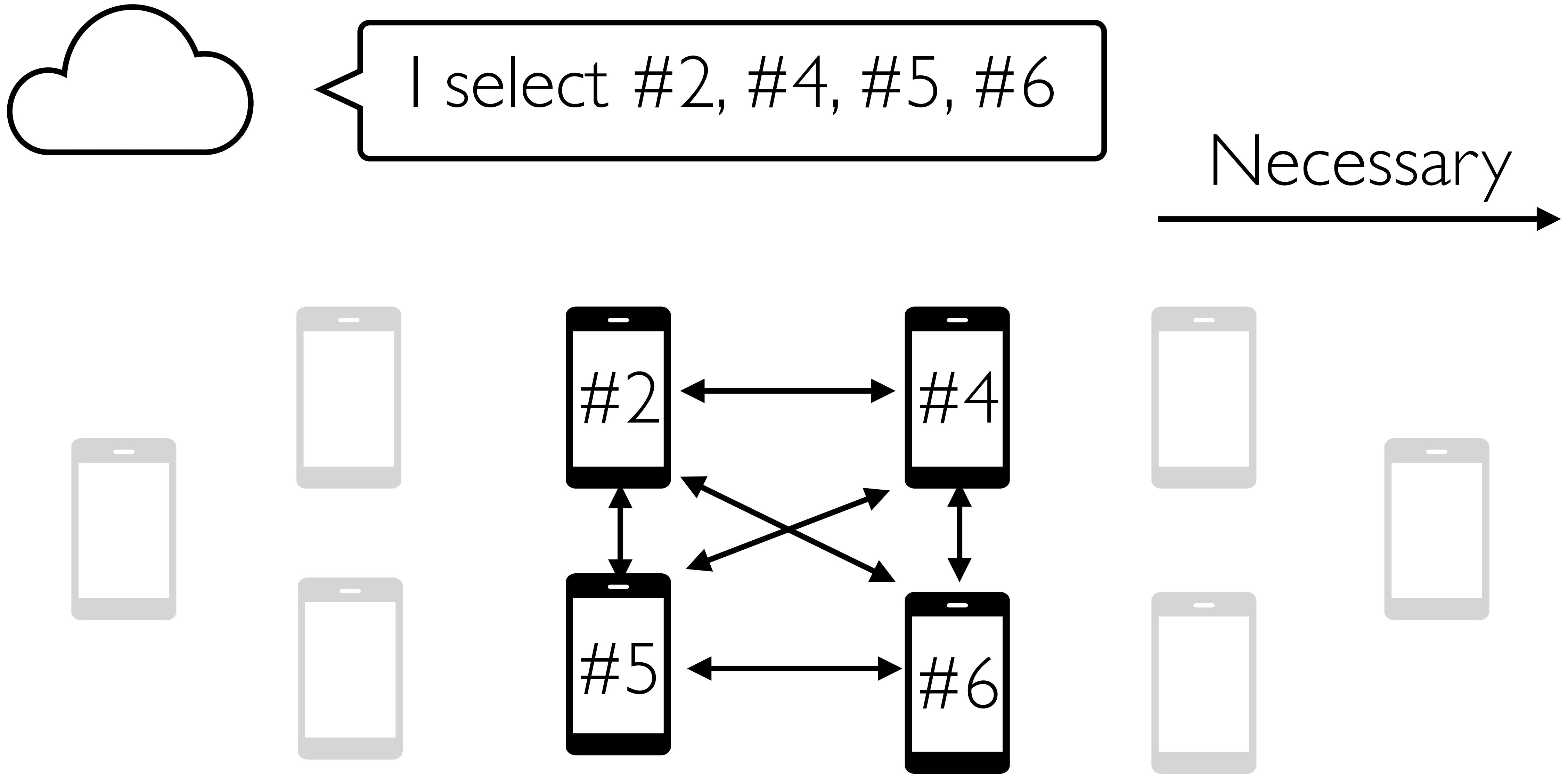
# Problem: Random selection

**What is achieved**:

Each participant
    sees a list of peers

**Potential approach**:

- Outcome verification
- Only within participants ($10^1$ - $10^2$)



# Problem: Random selection

**What is achieved**:

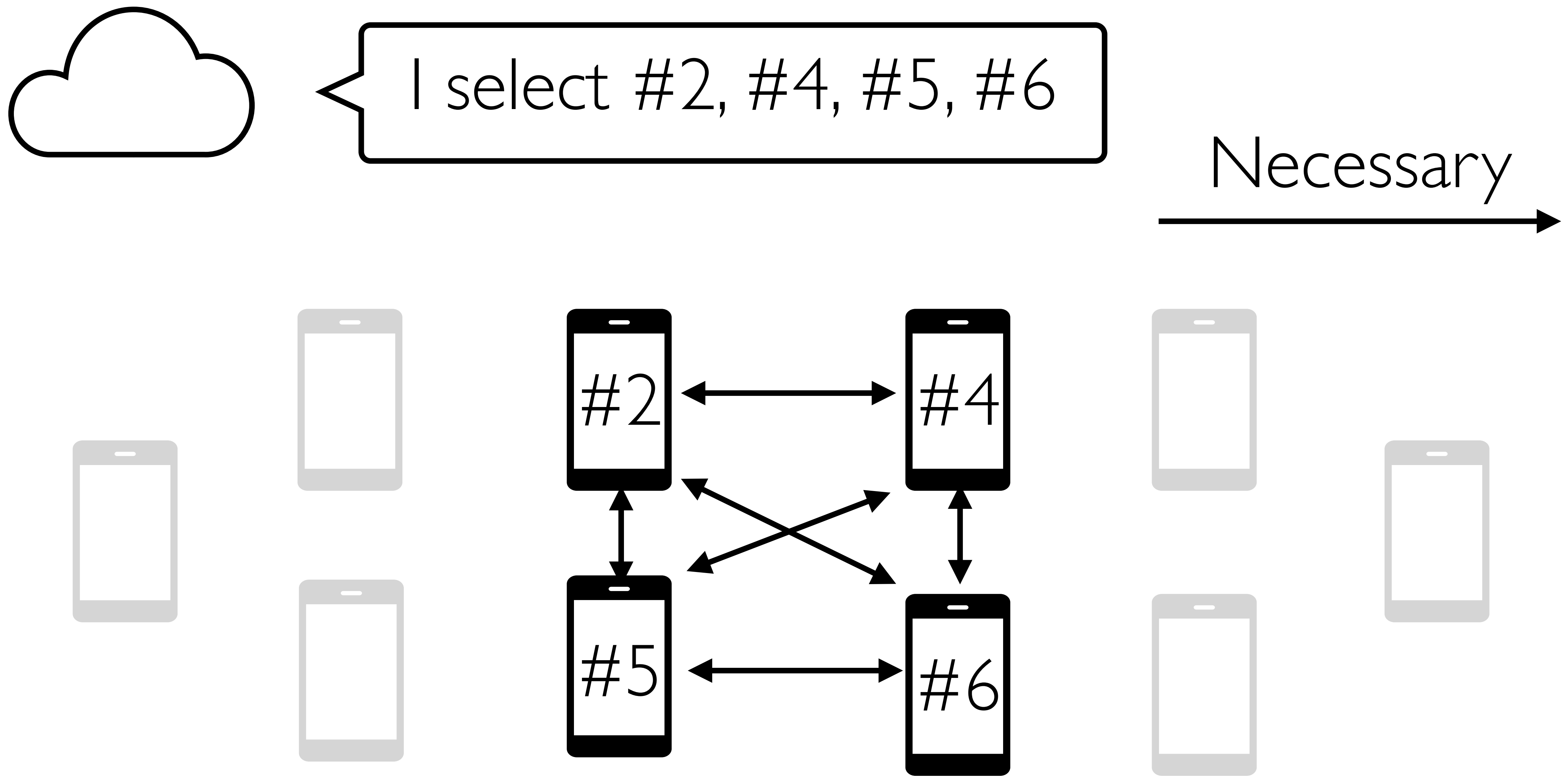
Each participant

sees a list of peers who

presents only by chance.

**Potential approach**:

- Outcome verification
- Only within participants ($10^1$ - $10^2$)



# Problem: Random selection

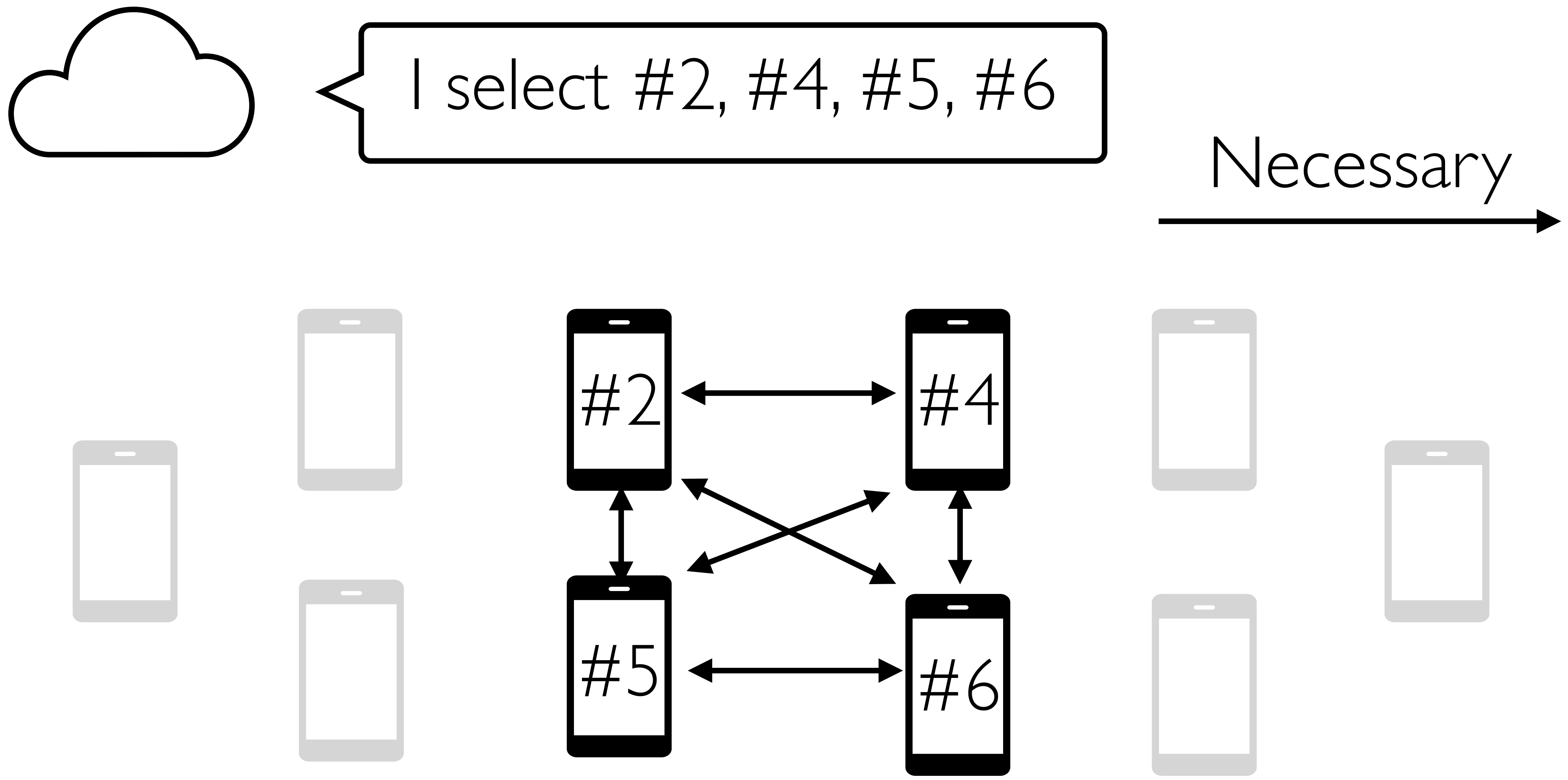
**What is achieved**:

Each participant
sees a list of peers who
presents only by chance.

E.g., $\frac{\text{Selection criteria: } \mathbf{<3}}{\text{Output range: } \mathbf{[0,\ 10)}} = 3/10$

**Potential approach**:

- Outcome verification
- Only within participants ($10^1$ - $10^2$)



# Problem: Random selection

**What is achieved**:

Each participant

sees a list of peers who

presents only by chance.

E.g., $\frac{\text{Selection criteria: } \mathbf{<3}}{\text{Output range: } \mathbf{[0,\ 10)}} = 3/10$

# Problem: Random selection

**What is achieved**:

Each participant

sees a list of peers who

presents only by chance.

What happens to the absent?

# Problem: Random selection

**What is achieved**:

Each participant

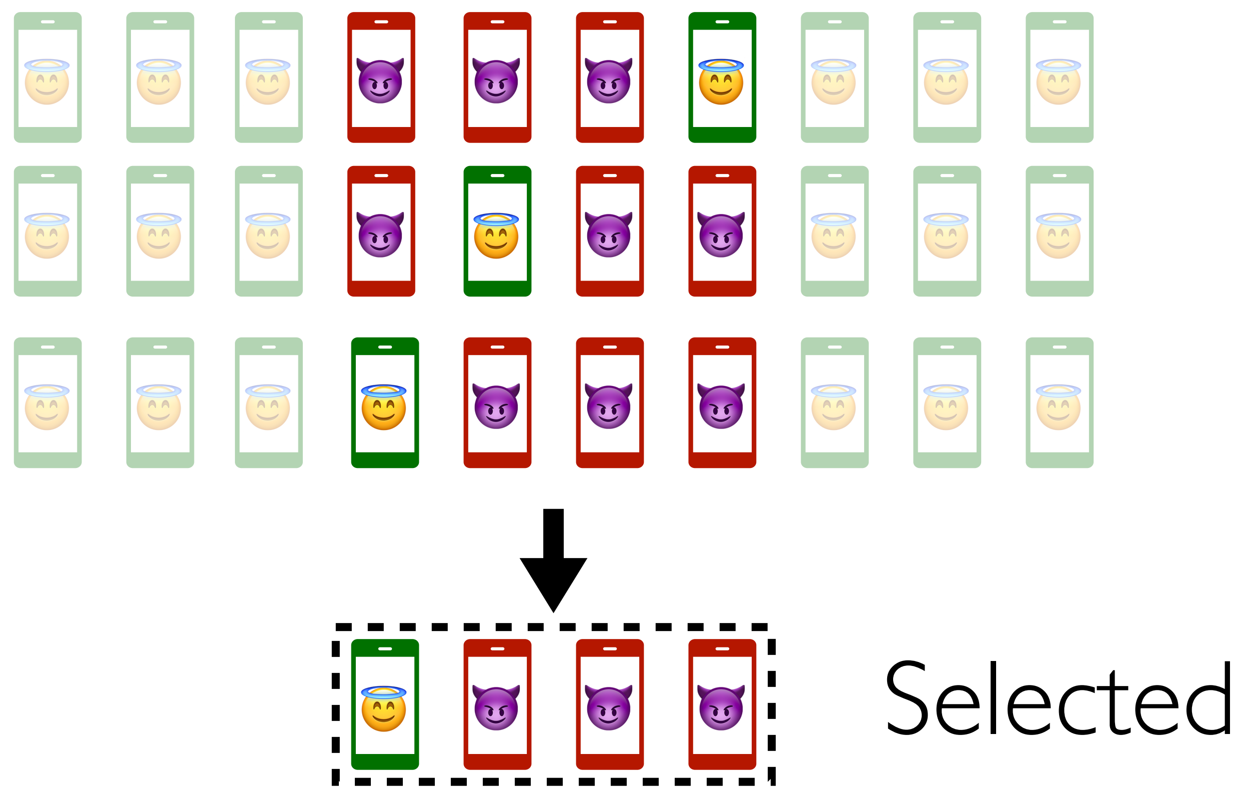
sees a list of peers who

presents only by chance.

What happens to the absent?

**Problem**: The server may arbitrarily **ignore honest** clients

Ignore **before** selection



# Problem: Random selection

**What is achieved**:

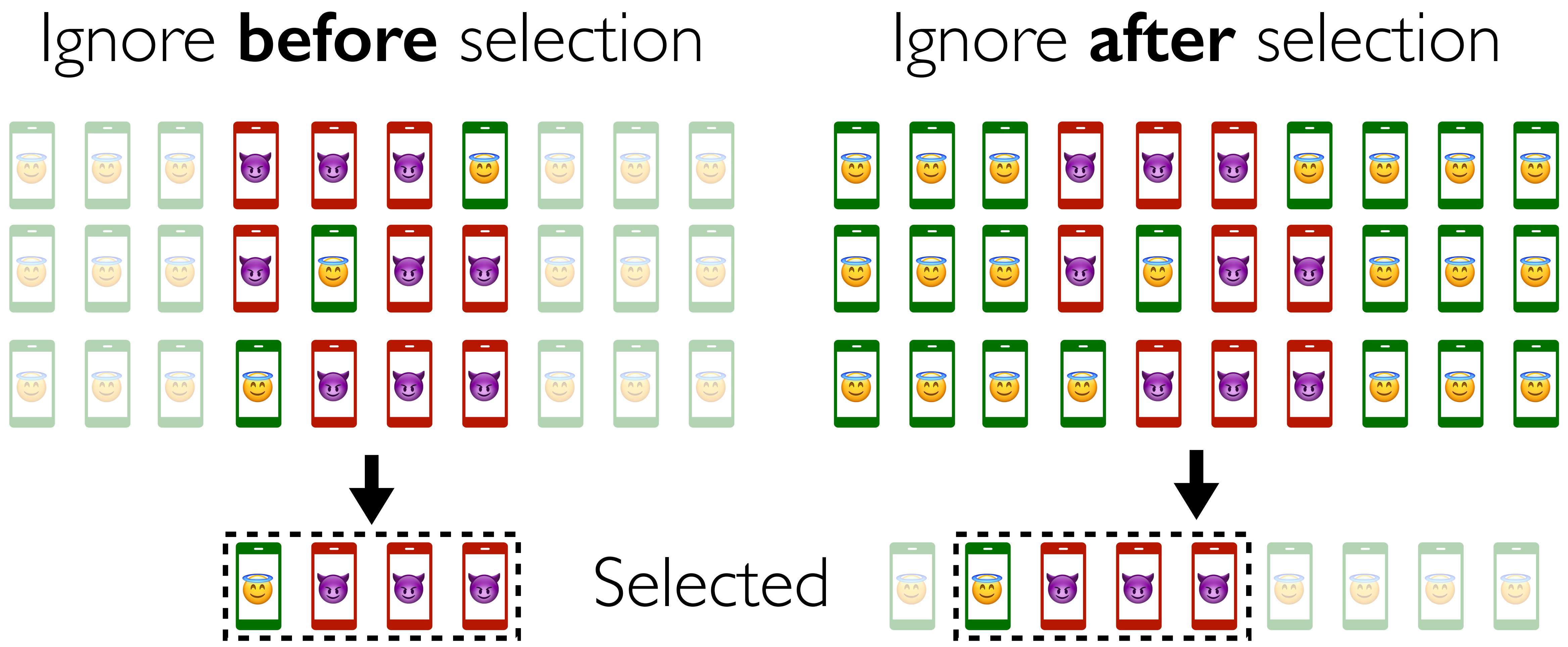
Each participant

sees a list of peers who

presents only by chance.

What happens to the absent?

**Problem**: The server may arbitrarily **ignore honest** clients



# Problem: Random selection

**What is achieved**:

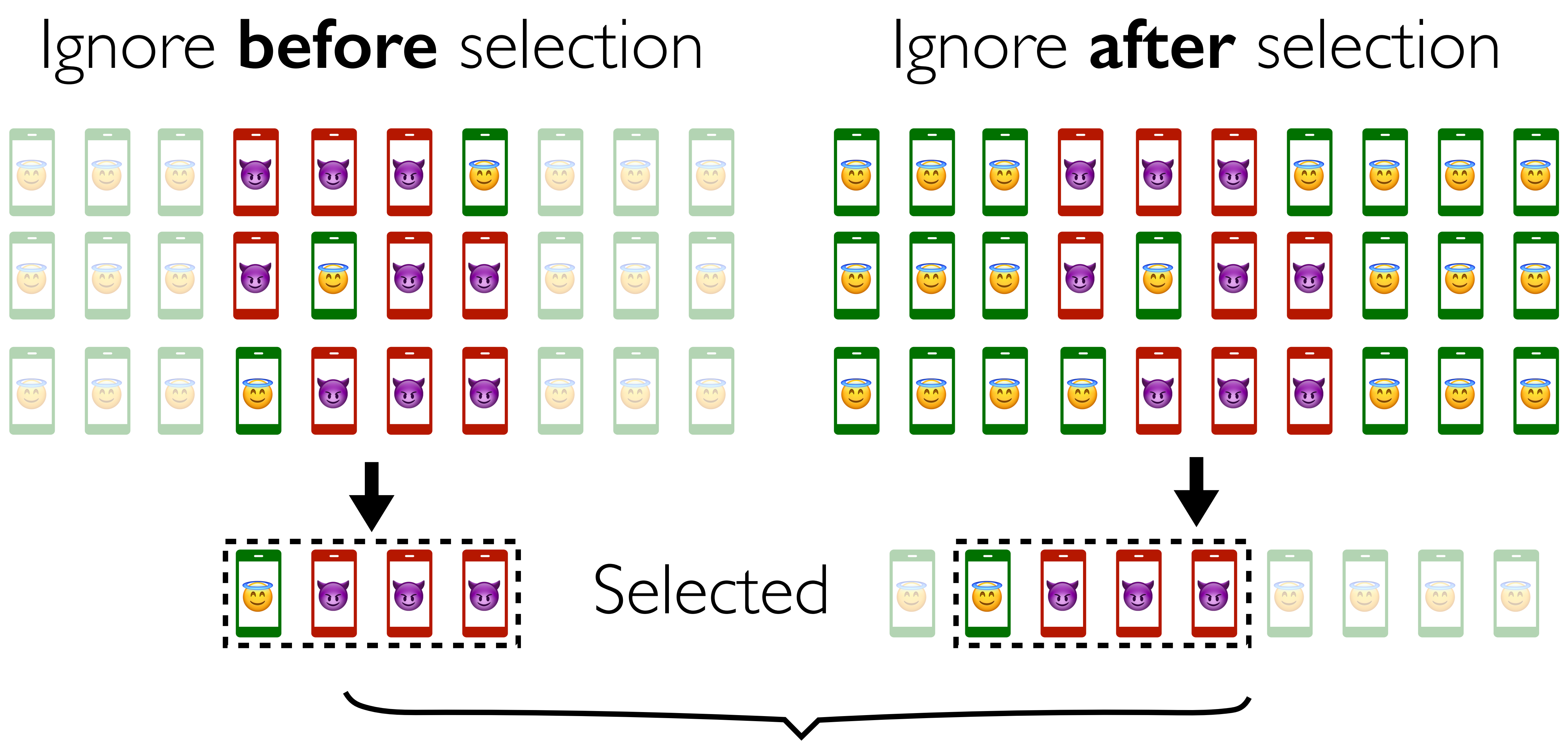
Each participant

sees a list of peers who

presents only by chance.

What happens to the absent?

**Problem**: The server may arbitrarily **ignore honest** clients




**Unbounded** advantage in growing dishonesty

# Problem: Random selection

**What is achieved**:

Each participant

sees a list of peers who

presents only by chance.

What happens to the absent?

**Solution**: Enforce a **large enough list** and a **small enough chance**.

# Problem: Random selection

**What is achieved**:

Each participant
 sees a list of peers who
 presents only by chance.

What happens to the absent?

**Solution**: Enforce a **large enough list** and a **small enough chance**.

Example

- **len(list)**: ≥ 200
- **Chance**: ≤ 0.1%

# Problem: Random selection

**What is achieved**:

Each participant

sees a list of peers who

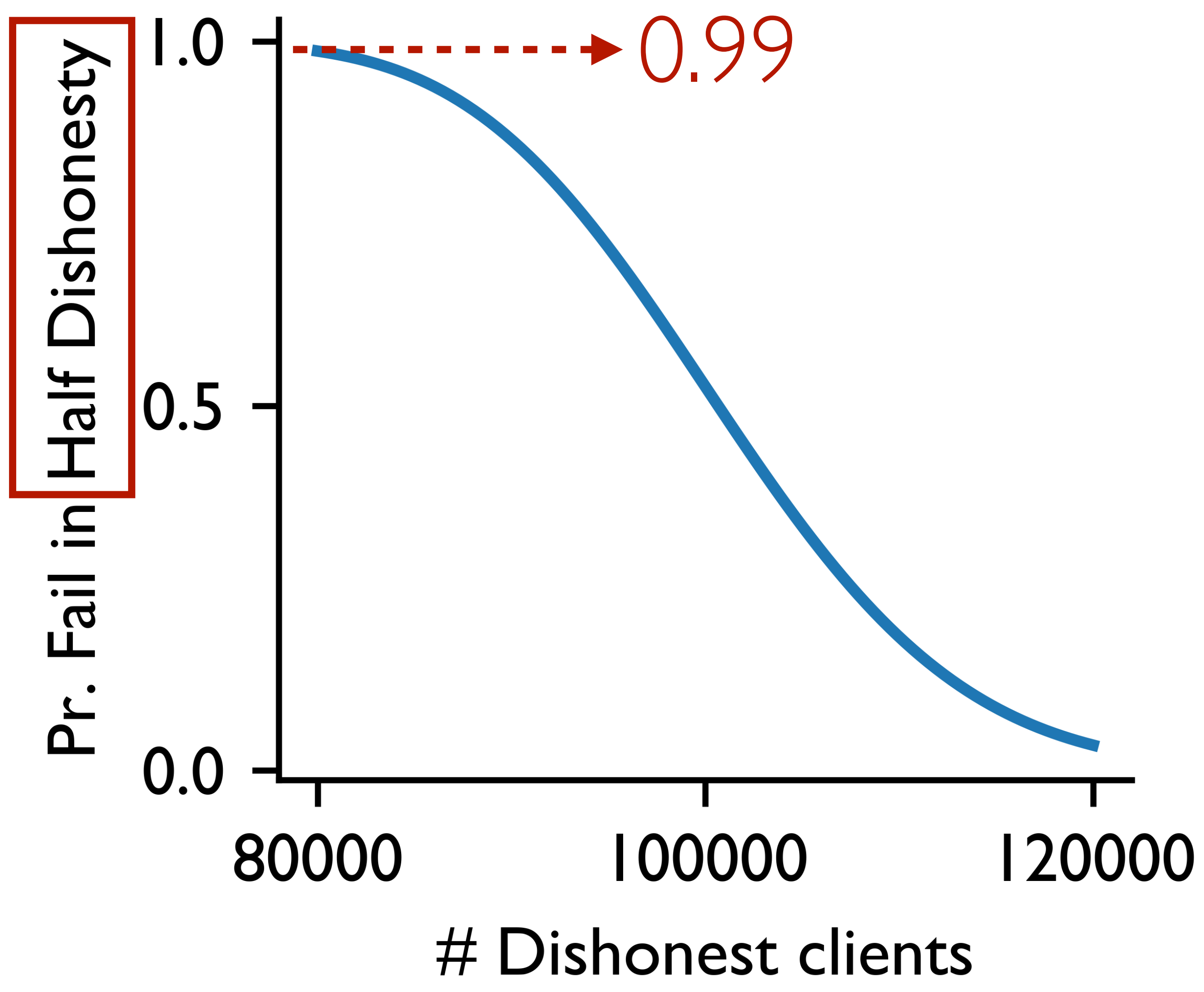
presents only by chance.

What happens to the absent?

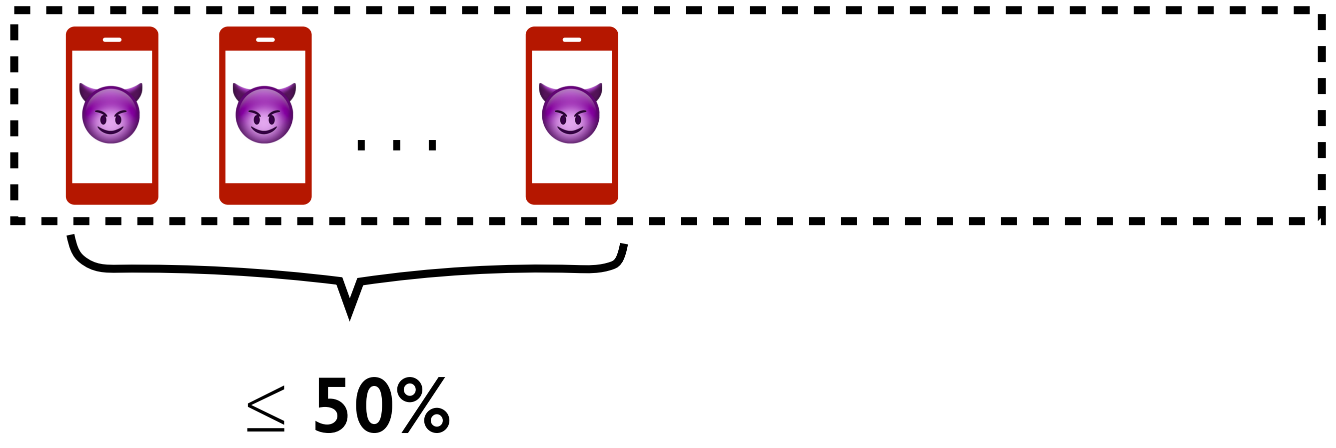
**Solution**: Enforce a **large enough list** and a **small enough chance**.

Example

- **len(list)**: ≥ 200
- **Chance**: ≤ 0.1%




Selected



# Problem: Random selection

**What is achieved**:

Each participant

sees a list of peers who

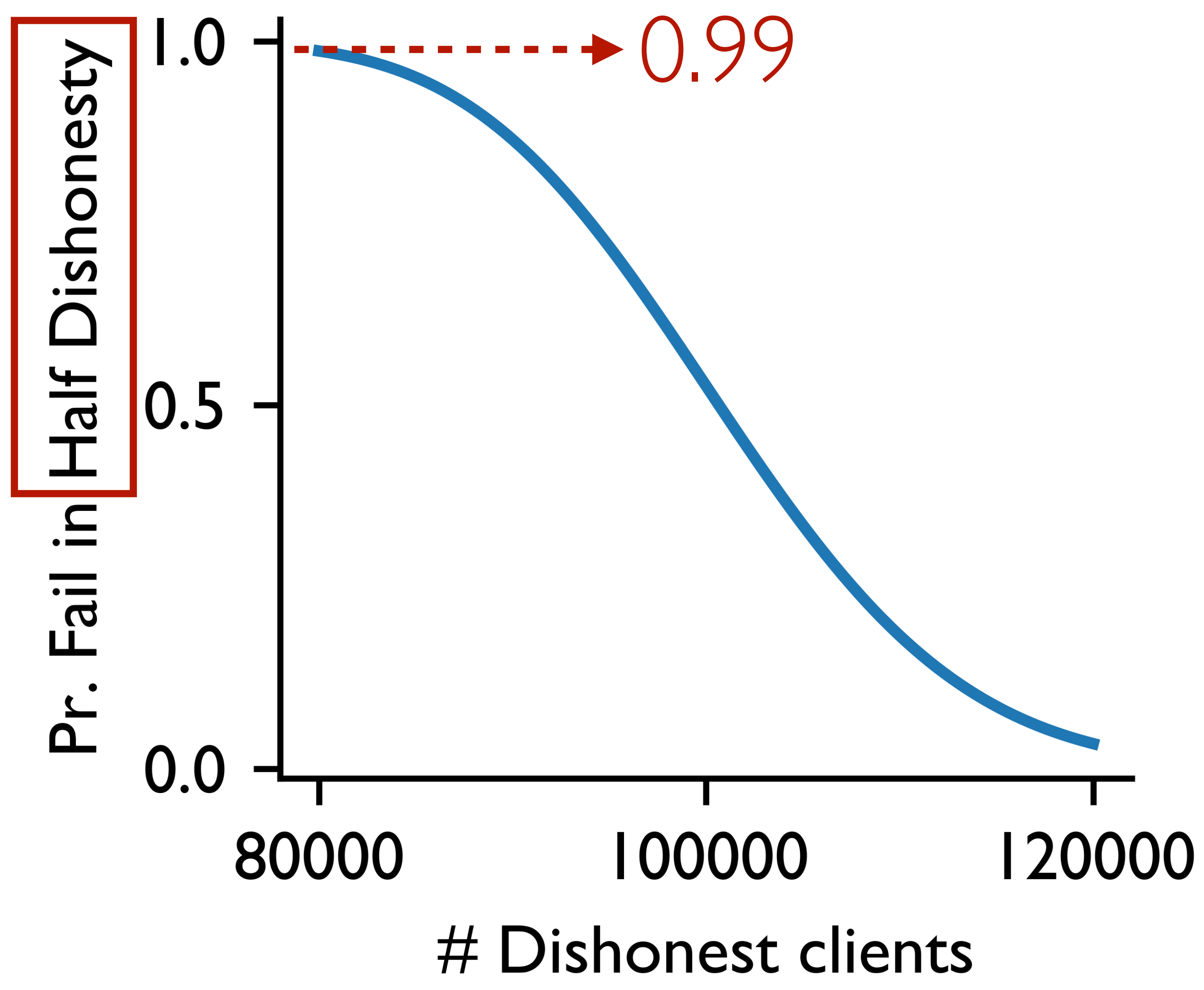
presents only by chance.

The absent will not get arbitrarily ignored

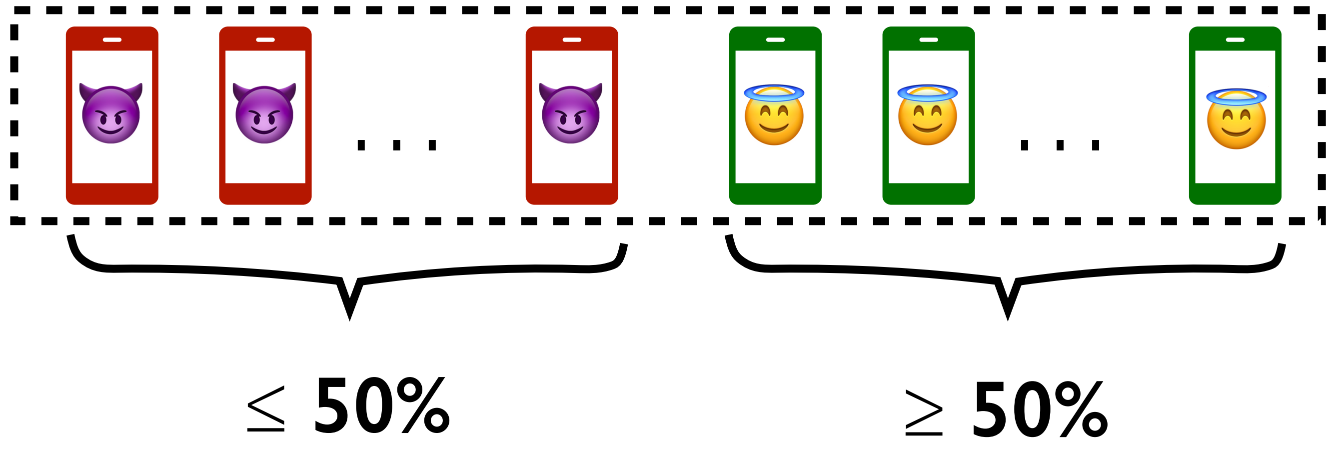
**Solution**: Enforce a **large enough list** and a **small enough chance**.

Example

- **len(list)**: ≥ 200
- **Chance**: ≤ 0.1%




Selected



# Problem: Random selection

**What is achieved**:

Each participant

sees a list of peers who

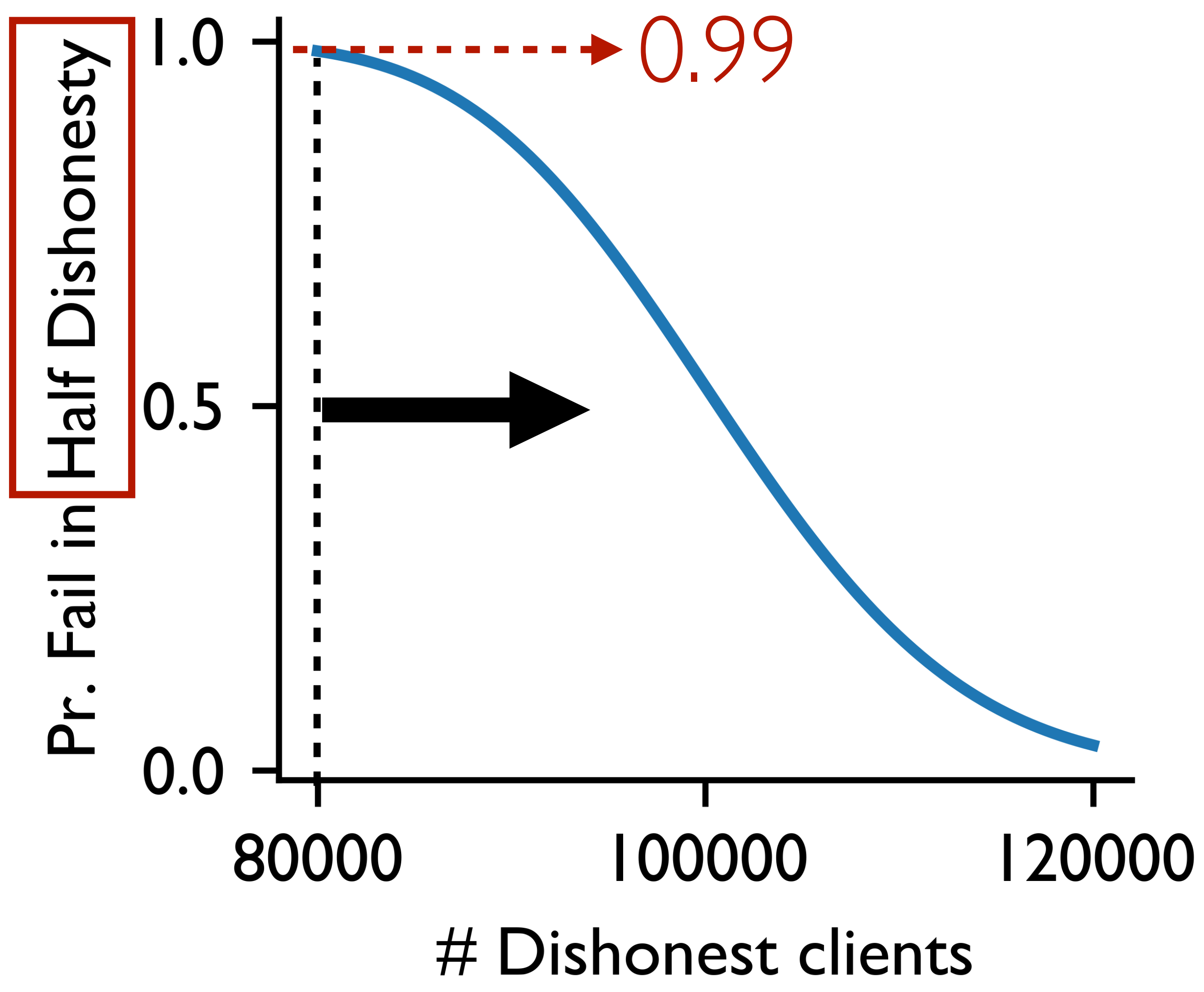
presents only by chance.

The absent will not get arbitrarily ignored

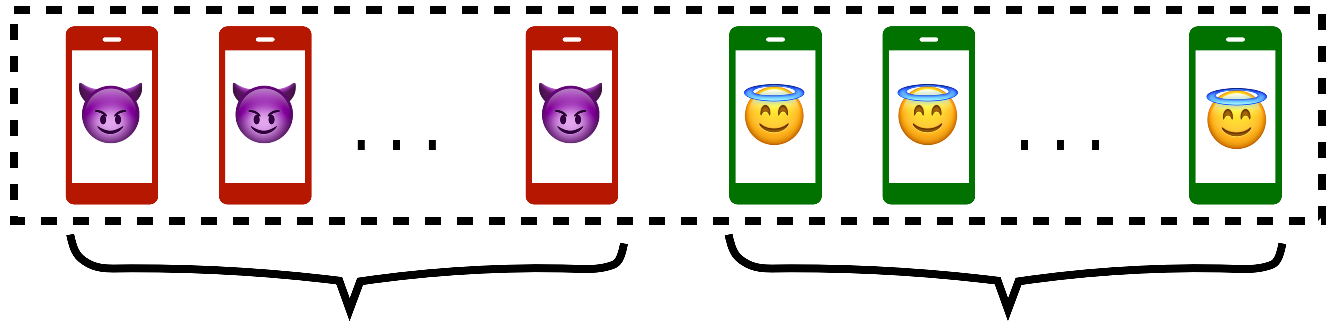
**Solution**: Enforce a **large enough list** and a **small enough chance**.

Example

- **len(list)**: ≥ 200
- **Chance**: ≤ 0.1%




Selected




≤ **50%**

≥ **50%**

# Problem: Random selection

**What is achieved**:

Each participant

sees a list of peers who

presents only by chance.

Predictable

to server?

The absent will not get

arbitrarily ignored

Public Round index

**Examples**: #2 will be selected as $\mathbf{RF}_{pk2}(2) = 1 < 3$.

Public Public keys

# Problem: Random selection

**What is achieved**:

Each participant

sees a list of peers who

presents only by chance.

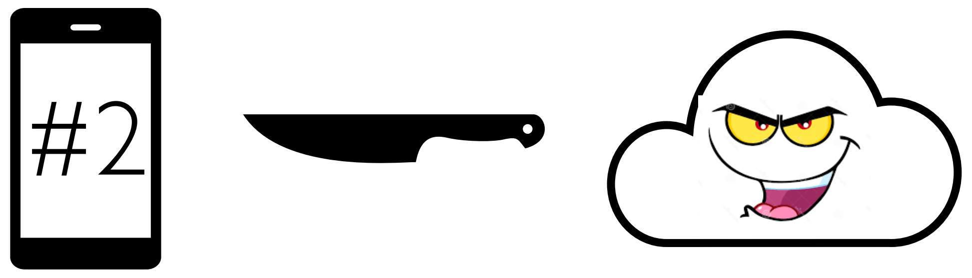
Predictable to server?

The absent will not get arbitrarily ignored

**Problem**: Attack surfaces **enlarged**!

**Examples**: #2 will be selected as $\mathbf{RF}_{pk2}(2) = 1 < 3$.

It's honest, so the server may grow its advantage by




Focused hacking

# Problem: Random selection

**What is achieved**:

Each participant

sees a list of peers who

presents only by chance.

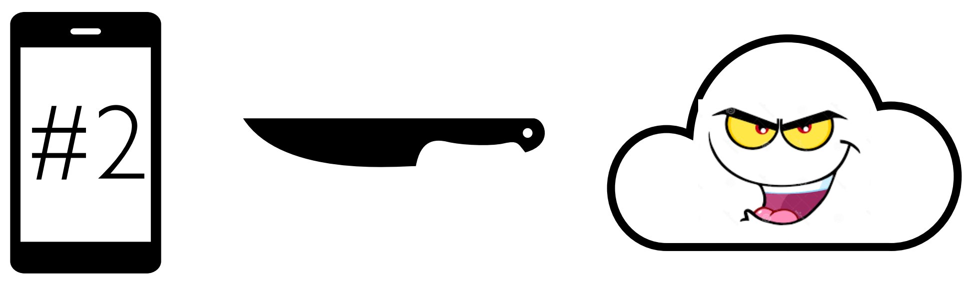
Predictable to server?

The absent will not get arbitrarily ignored

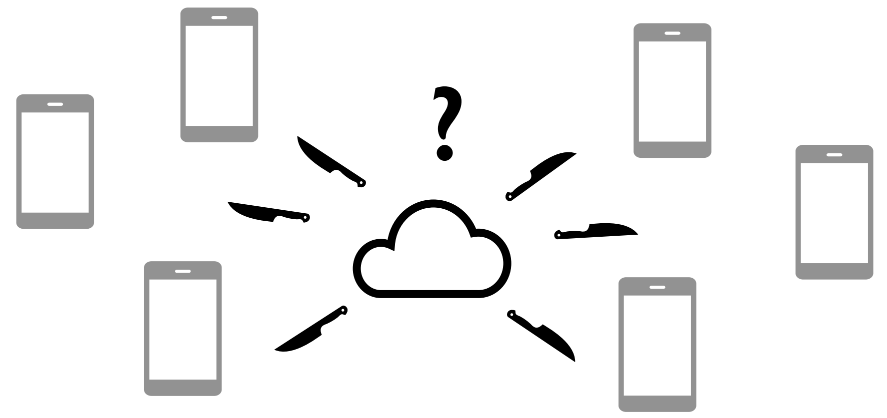
**Problem**: Attack surfaces **enlarged**!

**Examples**: #2 will be selected as $\mathbf{RF}_{pk2}(2) = 1 < 3$.

It's honest, so the server may grow its advantage by




vs




Focused hacking

Random compromise

# Problem: Random selection

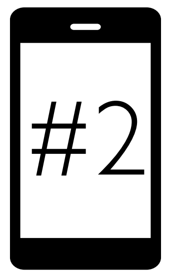
**What is achieved**:

Each participant sees a list of peers who presents only by chance.

Predictable to server?

The absent will not get arbitrarily ignored

**Solution**: **Self-sampling** with verifiable random functions (**VRFs**)[1,2].




Evaluation: $\mathbf{VRF.eval}_{sk2}(2) = (\ 1,\quad )\ (\ \text{output},\ ,\qquad )$

Secret key

[1]Micali et al. "Verifiable random functions", In FOCS '99
[2]Dodis et al. "A verifiable random function with short proofs and keys", In PKC '05

# Problem: Random selection

**What is achieved**:

Each participant

sees a list of peers who

presents only by chance.

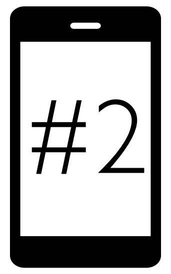
Predictable

to server?

The absent will not get

arbitrarily ignored

**Solution**: **Self-sampling** with verifiable random functions (**VRFs**)[1,2].




Evaluation: $\textbf{VRF.eval}_{sk2}(2) = (\ 1, \boldsymbol{\pi_2}\ )$ ( output, **proof** )

[1]Micali et al. "Verifiable random functions", In FOCS '99

[2]Dodis et al. "A verifiable random function with short proofs and keys", In PKC '05

# Problem: Random selection

**What is achieved**:

Each participant

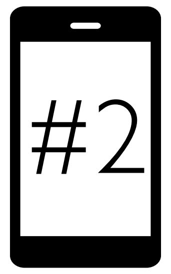
sees a list of peers who

presents only by chance.

Predictable to server?

The absent will not get arbitrarily ignored

**Solution**: **Self-sampling** with verifiable random functions (**VRFs**)[1,2].




Evaluation: $\mathbf{VRF.eval}_{sk2}(2) = (\ 1, \boldsymbol{\pi_2}\ )$ ( output, **proof** )

Verification: $\mathbf{VRF.ver}_{pk2}(\ 2, 1, \boldsymbol{\pi_2}\ ) = \text{True}$

Public key

[1]Micali et al. "Verifiable random functions", In FOCS '99

[2]Dodis et al. "A verifiable random function with short proofs and keys", In PKC '05

# Problem: Random selection

**What is achieved**:

Each participant

sees a list of peers who

presents only by chance.

Unpredictable

to server

The absent will not get

arbitrarily ignored

**Solution**: **Self-sampling** with verifiable random functions (**VRFs**)[1,2].

I self-sample

with $(1, \pi_2)$

#2

Evaluation: $\mathbf{VRF.eval}_{sk2}(2) = (\ 1, \boldsymbol{\pi_2}\ )$ ( output, **proof** )

Verification: $\mathbf{VRF.ver}_{pk2}(\ 2, 1, \boldsymbol{\pi_2}\ ) = \text{True}$

[1] Micali et al. "Verifiable random functions", In FOCS '99

[2] Dodis et al. "A verifiable random function with short proofs and keys", In PKC '05

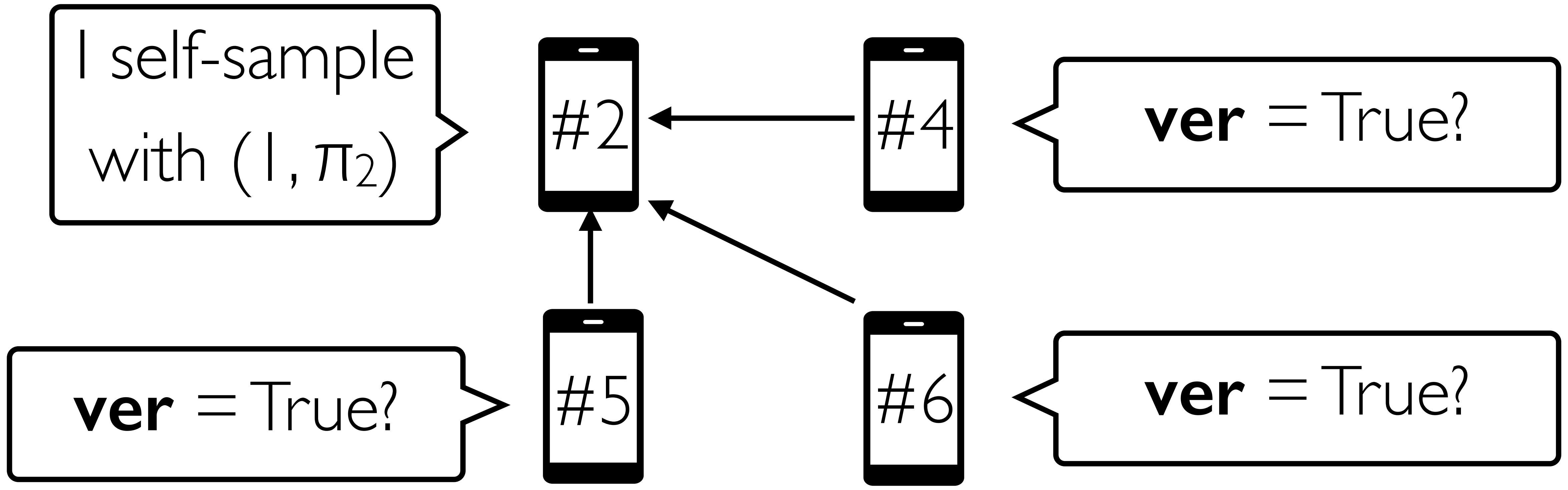
# Problem: Random selection

**Solution**: **Self-sampling** with verifiable random functions (**VRFs**)[1,2].

**What is achieved**:
Each participant
sees a list of peers who
presents only by chance.

Unpredictable to server

The absent will not get arbitrarily ignored




Evaluation: $\textbf{VRF.eval}_{sk2}(2) = (\ 1, \boldsymbol{\pi_2}\ )$ ( output, **proof** )

Verification: $\textbf{VRF.ver}_{pk2}(\ 2, 1, \boldsymbol{\pi_2}\ ) = \text{True}$

[1]Micali et al. "Verifiable random functions", In FOCS '99
[2]Dodis et al. "A verifiable random function with short proofs and keys", In PKC '05

# Problem: Random selection

Actual participants
throughout the training?

**What is achieved**:
Each participant
sees a list of peers who
presents only by chance.

Unpredictable
to server

The absent will not get
arbitrarily ignored

# Problem: Random selection

Actual participants
throughout the training?

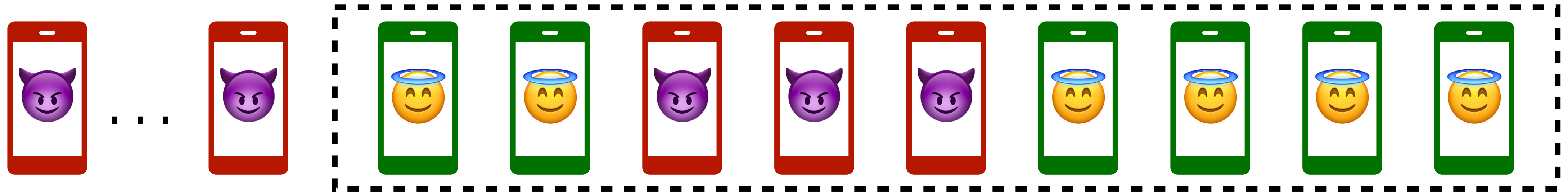
**What is achieved**:
Each participant
sees a list of peers who
presents only by chance.

Unpredictable
to server

The absent will not get
arbitrarily ignored

**Problem**: The server may **not follow**.

Involve **non-selected dishonest** ones



# Problem: Random selection

Actual participants
throughout the training?

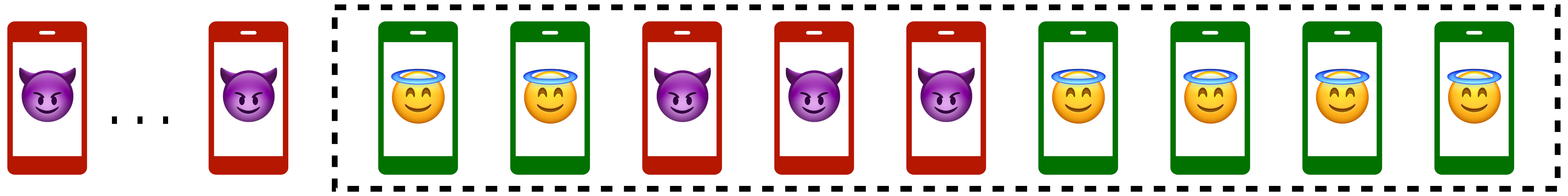
**What is achieved**:
Each participant
sees a list of peers who
presents only by chance.

Unpredictable
to server

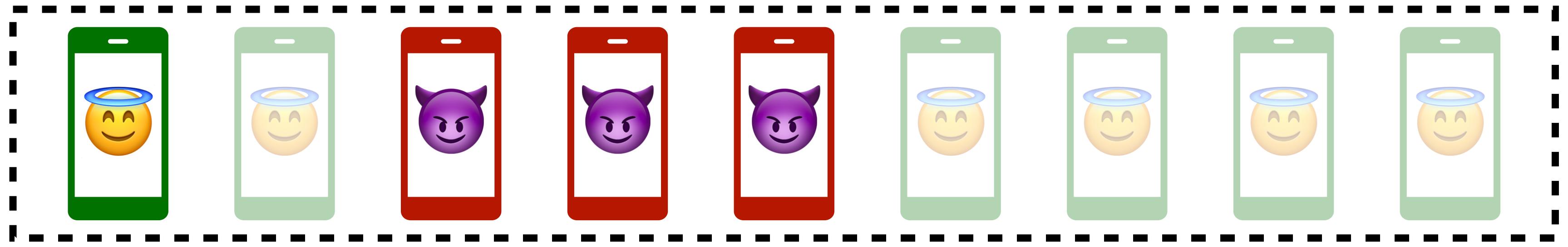
The absent will not get
arbitrarily ignored

**Problem**: The server may **not follow**.

Involve **non-selected dishonest** ones



Disregard **selected honest** ones



# Problem: Random selection

Actual participants
throughout the training?

**What is achieved**:
Each participant
sees a list of peers who
presents only by chance.

Unpredictable
to server

The absent will not get
arbitrarily ignored

**Solution**: Utilize existing **secure semantics** of secure aggregation[1]

[1]Thus also of distributed DP (other privacy-enhancing techniques may not have this feature and this is left for future work).

# Problem: Random selection

Actual participants
throughout the training?

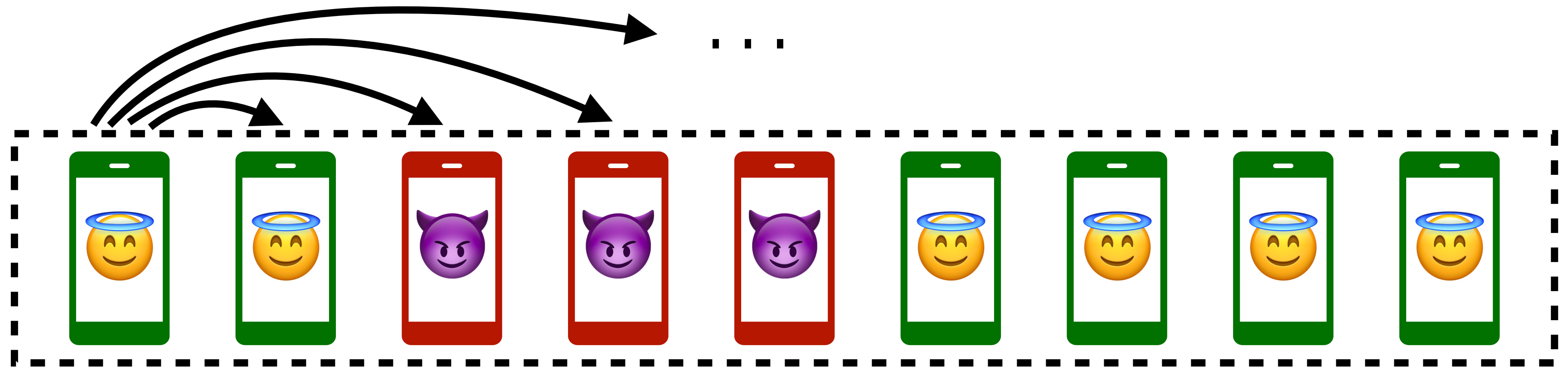
**What is achieved**:
Each participant
sees a list of peers who
presents only by chance.

Unpredictable
to server

The absent will not get
arbitrarily ignored

**Solution**: Utilize existing **secure semantics** of secure aggregation[1]

- **Commitment**: necessary info shared only once




[1]Thus also of distributed DP (other privacy-enhancing techniques may not have this feature and this is left for future work).

# Problem: Random selection

Actual participants
throughout the training?

**What is achieved**:
Each participant
sees a list of peers who
presents only by chance.

Unpredictable
to server

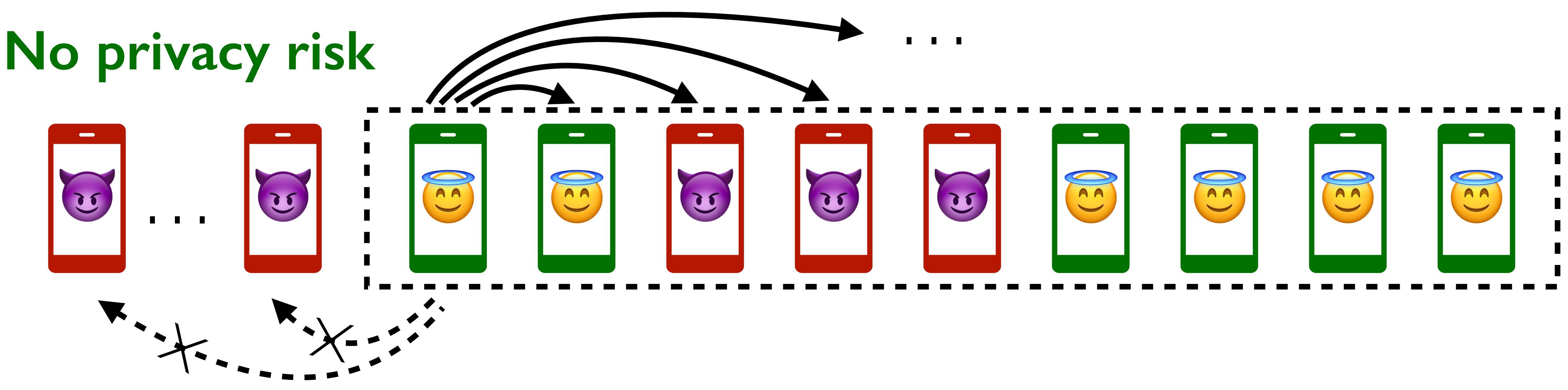
The absent will not get
arbitrarily ignored

[1]Thus also of distributed DP (other privacy-enhancing techniques may not have this feature and this is left for future work).

**Solution**: Utilize existing **secure semantics** of secure aggregation[1]

- **Commitment**: necessary info shared only once

No privacy risk



# Problem: Random selection

Actual participants throughout the training?

**What is achieved**:

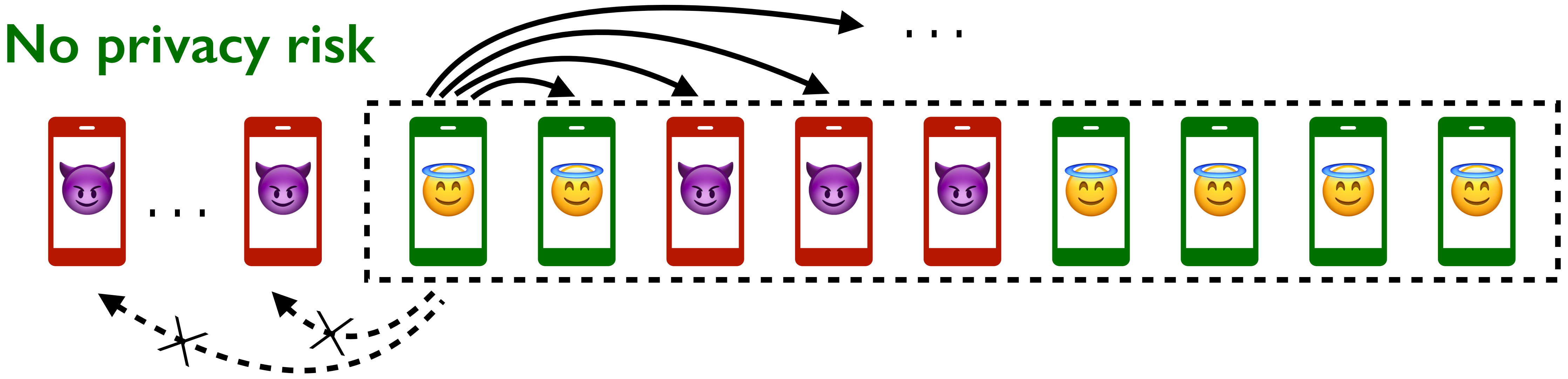
Each participant sees a list of peers who presents only by chance.

Unpredictable to server

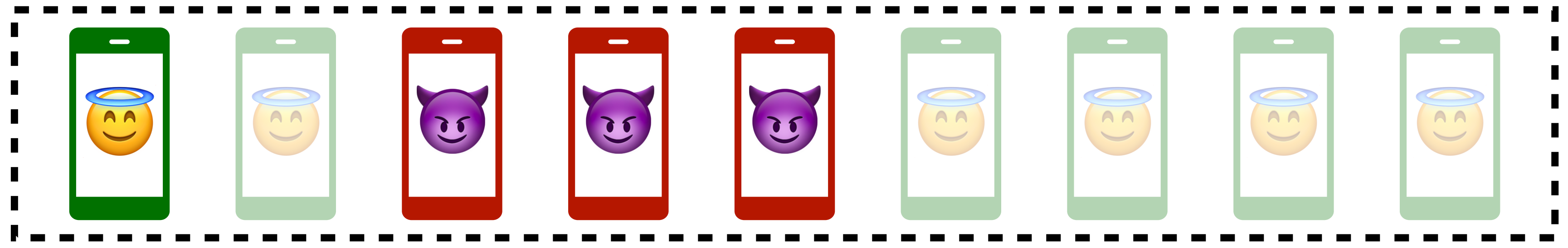
The absent will not get arbitrarily ignored

**Solution**: Utilize existing **secure semantics** of secure aggregation[1]

- **Commitment**: necessary info shared only once




- **Consistency check**: to know remaining participants



[1]Thus also of distributed DP (other privacy-enhancing techniques may not have this feature and this is left for future work).

# Problem: Random selection

Actual participants throughout the training

**What is achieved**:

Each participant

sees a list of peers who

presents only by chance.

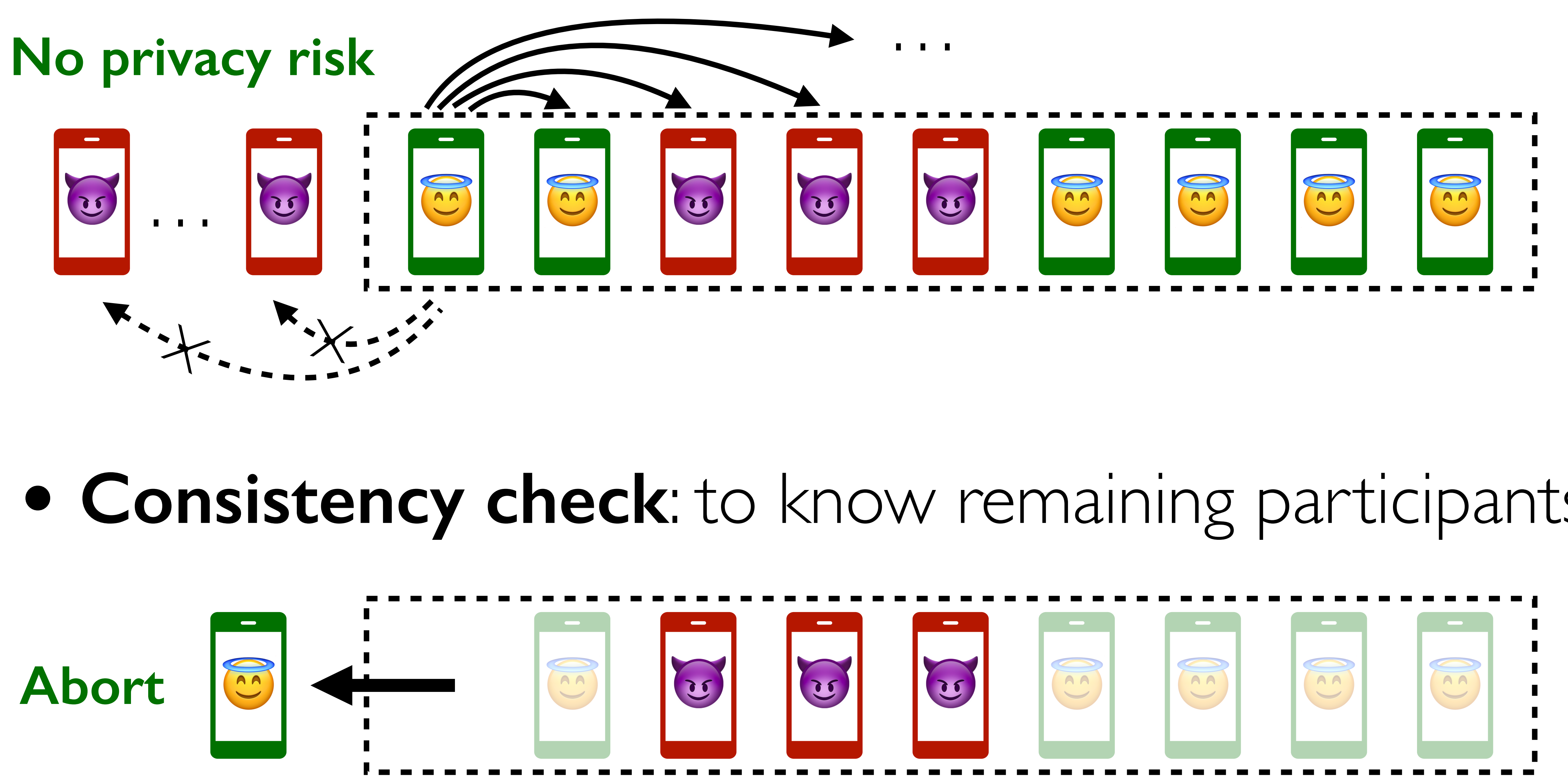
Unpredictable to server

The absent will not get arbitrarily ignored

**Solution**: Utilize existing **secure semantics** of secure aggregation[1]

- **Commitment**: necessary info shared only once

- **Consistency check**: to know remaining participants




[1] Thus also of distributed DP (other privacy-enhancing techniques may not have this feature and this is left for future work).

# Problem: Random selection

Actual participants throughout the training

**What is achieved**:

Each participant sees a list of peers who presents only by chance.

Unpredictable to server

The absent will not get arbitrarily ignored

Minor issues:

- **Fixed sample size**: over-selection
- **Consistent round index**: uniqueness check

…

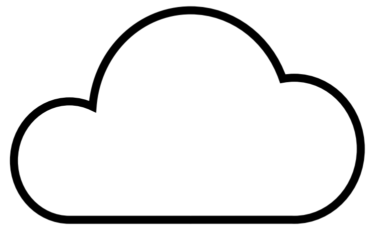
Please find more in the paper :)

[1] Thus also of distributed DP (other privacy-enhancing techniques may not have this feature and this is left for future work).

# Problem: Informed selection

# Problem: Informed selection

**Example**



| | (Est.) latency |
|---|---|
| #1 | 1.2s |
| #2 | 2.7s |
| #3 | 1.6s |
| … | … |

# Problem: Informed selection

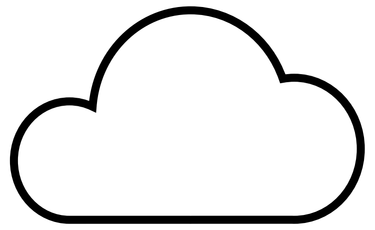
**Example**

| | (Est.) latency | Select |
|---|---|---|
| #1 | 1.2s | Yes |
| #2 | 2.7s | No |
| #3 | 1.6s | Yes |
| … | … | … |

Selection criteria: the fastest For dishonest majority

# Problem: Informed selection

**Example**



| | (Est.) latency | Select | (Est.) latency | Select |
|---|---|---|---|---|
| #1 | 1.2s | Yes | | Yes |
| #2 | 2.7s | No | Does NOT matter. | No |
| #3 | 1.6s | Yes | | No |
| … | … | … | | … |

Selection criteria: the fastest For dishonest majority

# Problem: Informed selection

**Example**

| | (Est.) latency | Select | (Est.) latency | Select |
|---|---|---|---|---|
| #1 | 1.2s | Yes | | Yes |
| #2 | 2.7s | No | Does NOT matter. | No |
| #3 | 1.6s | Yes | | No |
| ... | ... | ... | | ... |

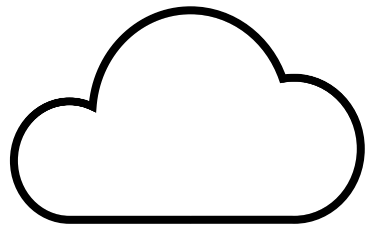
Selection criteria: the fastest

For dishonest majority

**Major Challenge**: Client metrics are **hard to verify** by honest clients

# Problem: Informed selection

**Example**



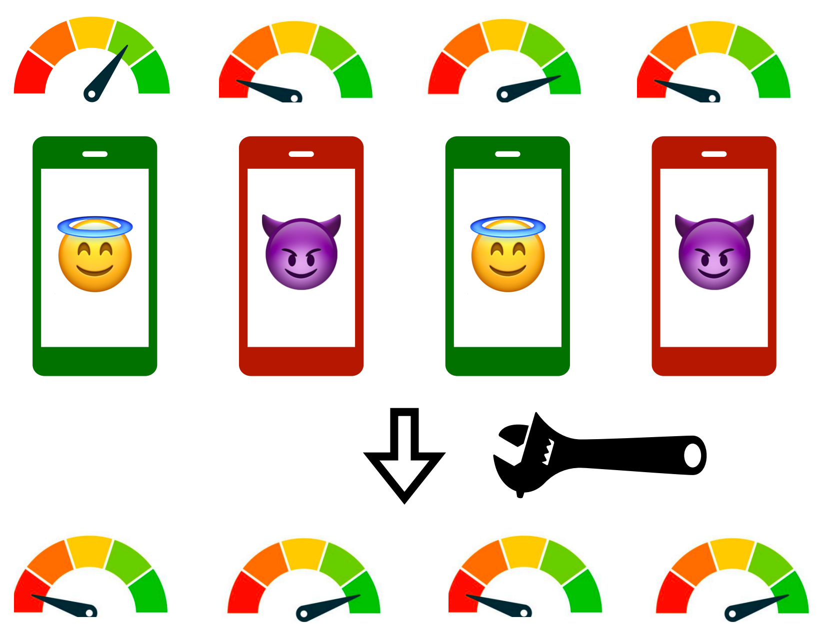
| | (Est.) latency | Select | (Est.) latency | Select |
|---|---|---|---|---|
| #1 | 1.2s | Yes | | Yes |
| #2 | 2.7s | No | Does NOT matter. | No |
| #3 | 1.6s | Yes | | No |
| … | … | … | | … |

Selection criteria: the fastest

For dishonest majority

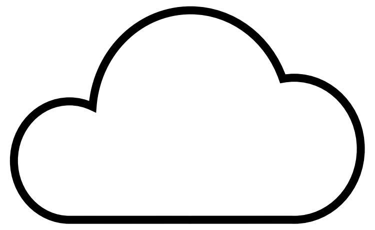
**Major Challenge**: Client metrics are **hard to verify** by honest clients

Metrics are fake



# Problem: Informed selection

**Example**



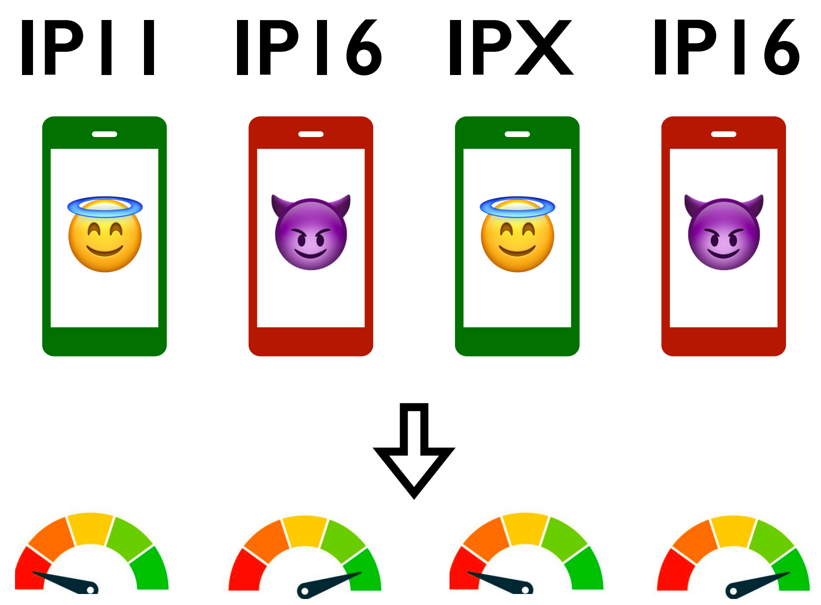
| | (Est.) latency | Select | (Est.) latency | Select |
|---|---|---|---|---|
| #1 | 1.2s | Yes | | Yes |
| #2 | 2.7s | No | Does NOT matter. | No |
| #3 | 1.6s | Yes | | No |
| ... | ... | ... | | ... |

Selection criteria: the fastest For dishonest majority

**Major Challenge**: Client metrics are **hard to verify** by honest clients

Metrics are fake

Metrics are true, but…



# Problem: Informed selection

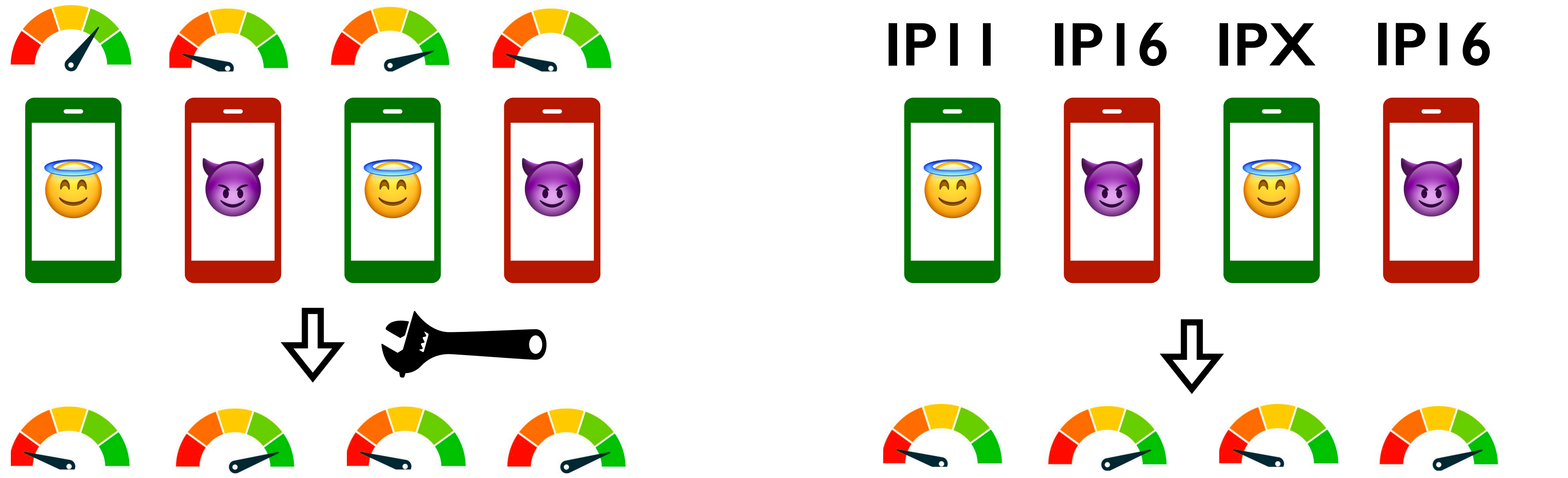
**Example**

| | (Est.) latency | Select | (Est.) latency | Select |
|---|---|---|---|---|
| #1 | 1.2s | Yes | Does NOT matter. | Yes |
| #2 | 2.7s | No | | No |
| #3 | 1.6s | Yes | | No |
| … | … | … | | … |

Selection criteria: the fastest    For dishonest majority

**Major Challenge**: Client metrics are **hard to verify** by honest clients

Metrics are fake    Metrics are true, but…



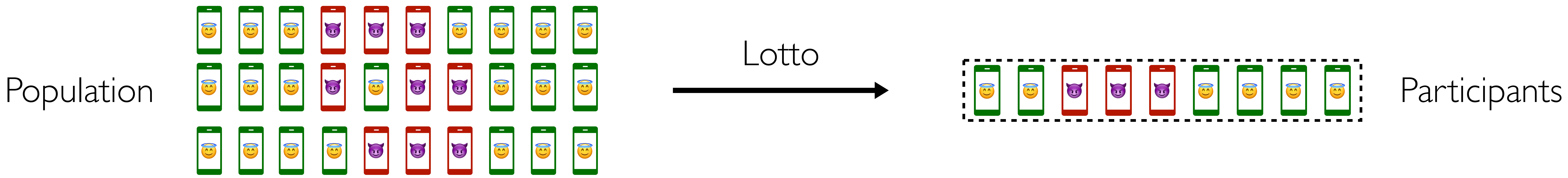

**Solution**: **Approximate** inform selection by **random** selection

Please find more in the paper :)

# Lotto prevents arbitrary manipulation

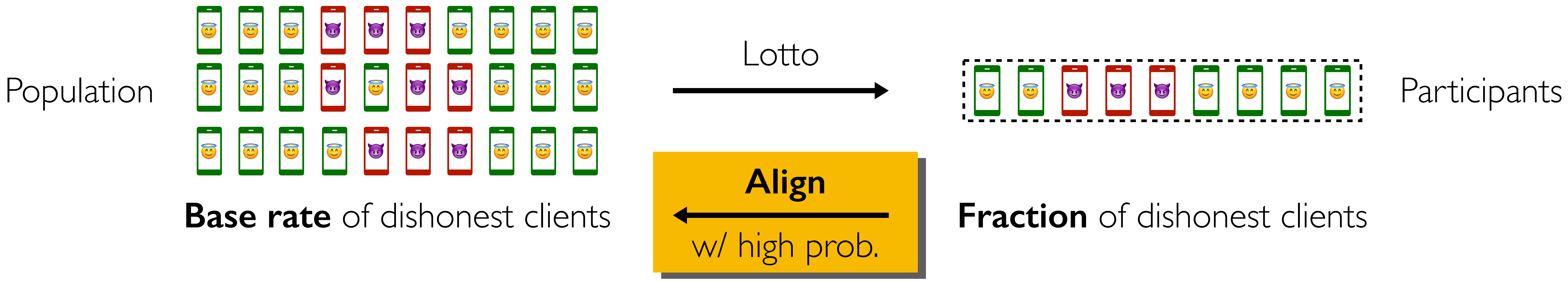
# Lotto prevents arbitrary manipulation

What can be **proven**:



# Lotto prevents arbitrary manipulation

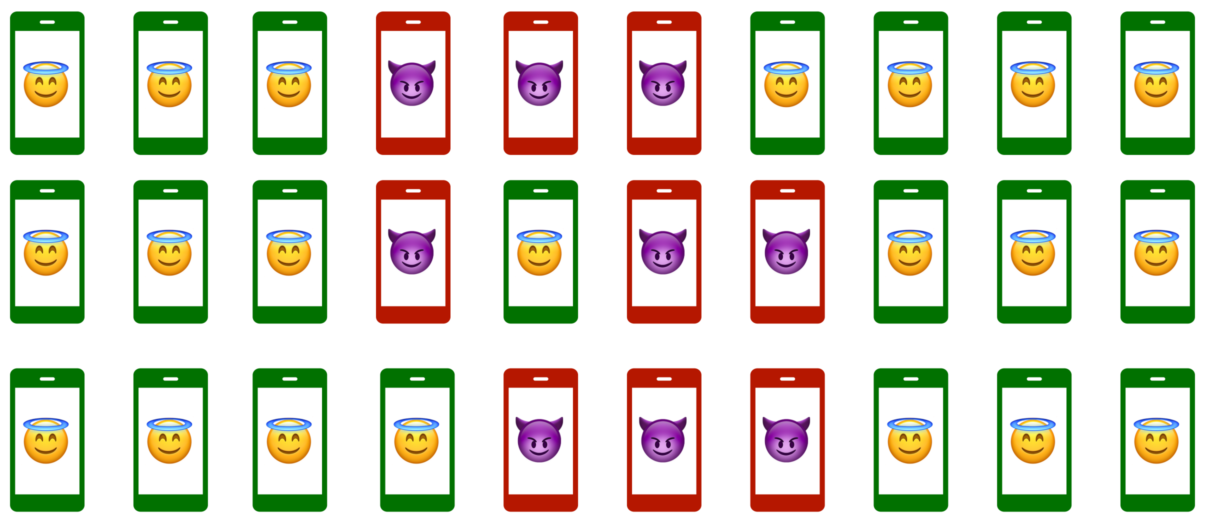
What can be **proven**:



# Lotto prevents arbitrary manipulation

What can be **proven**:

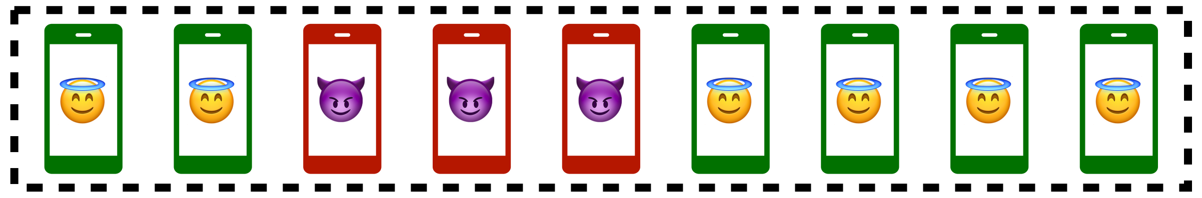
Population



**Base rate** of dishonest clients

Lotto →

**Align** ← w/ high prob.



Participants

**Fractions** of dishonest clients

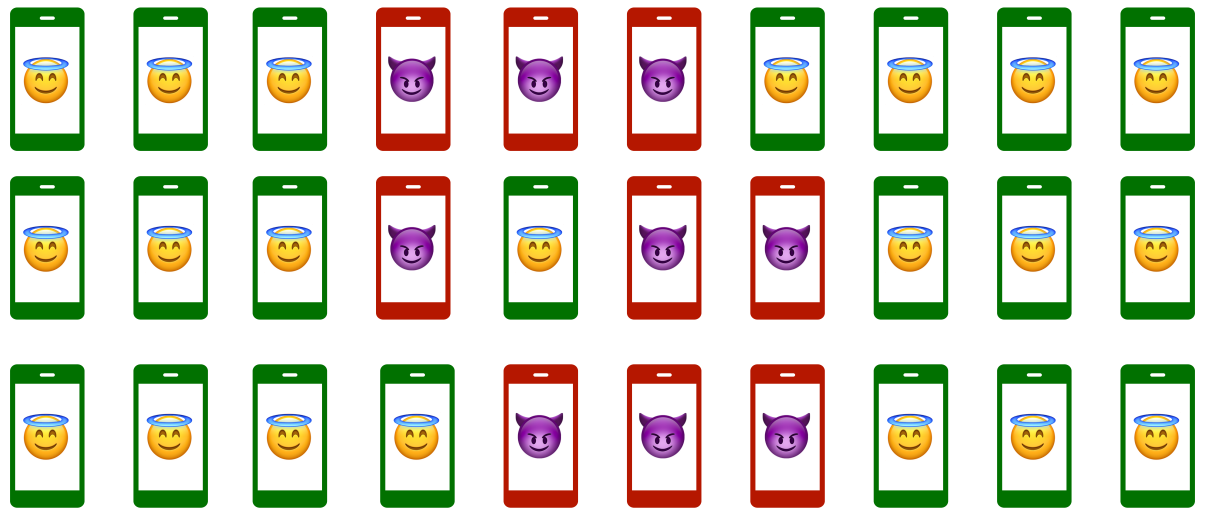
Example

- **Population**: 200,000
- **Dishonesty base rate**: 0.005

# Lotto prevents arbitrary manipulation

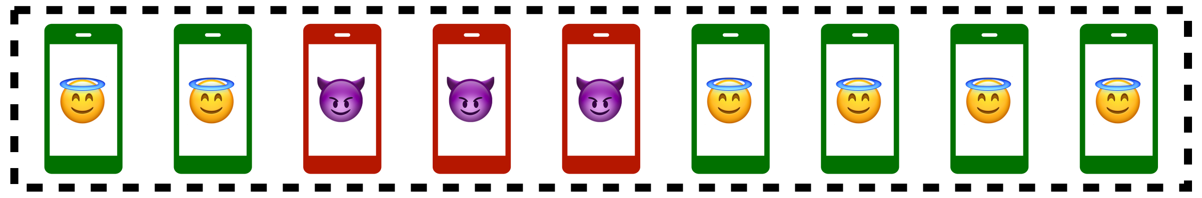
What can be **proven**:

Population



Lotto

Participants



**Align**

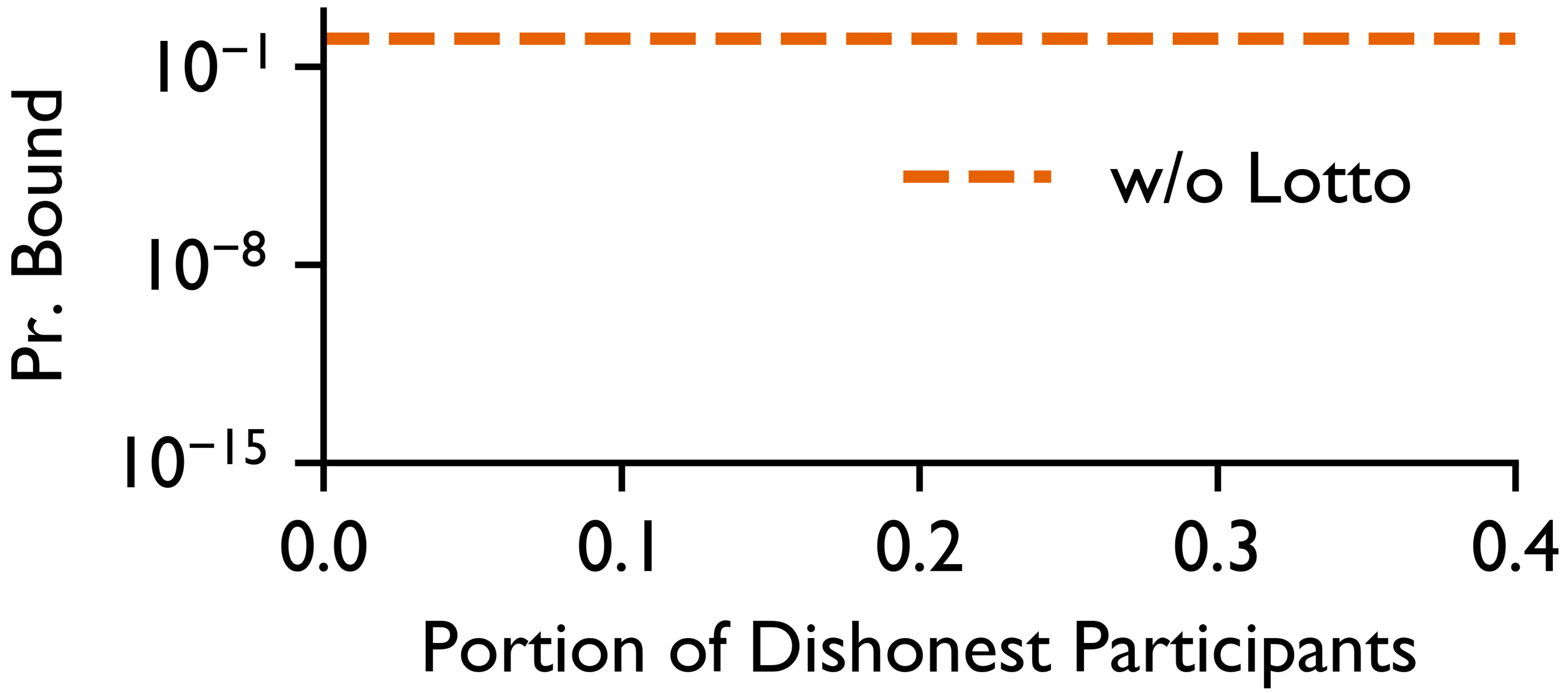
w/ high prob.

**Base rate** of dishonest clients

**Fractions** of dishonest clients

Example

- **Population**: 200,000
- **Dishonesty base rate**: 0.005
- **Target participants**: 200



# Lotto prevents arbitrary manipulation

What can be **proven**:

Population


Participants

**Base rate** of dishonest clients

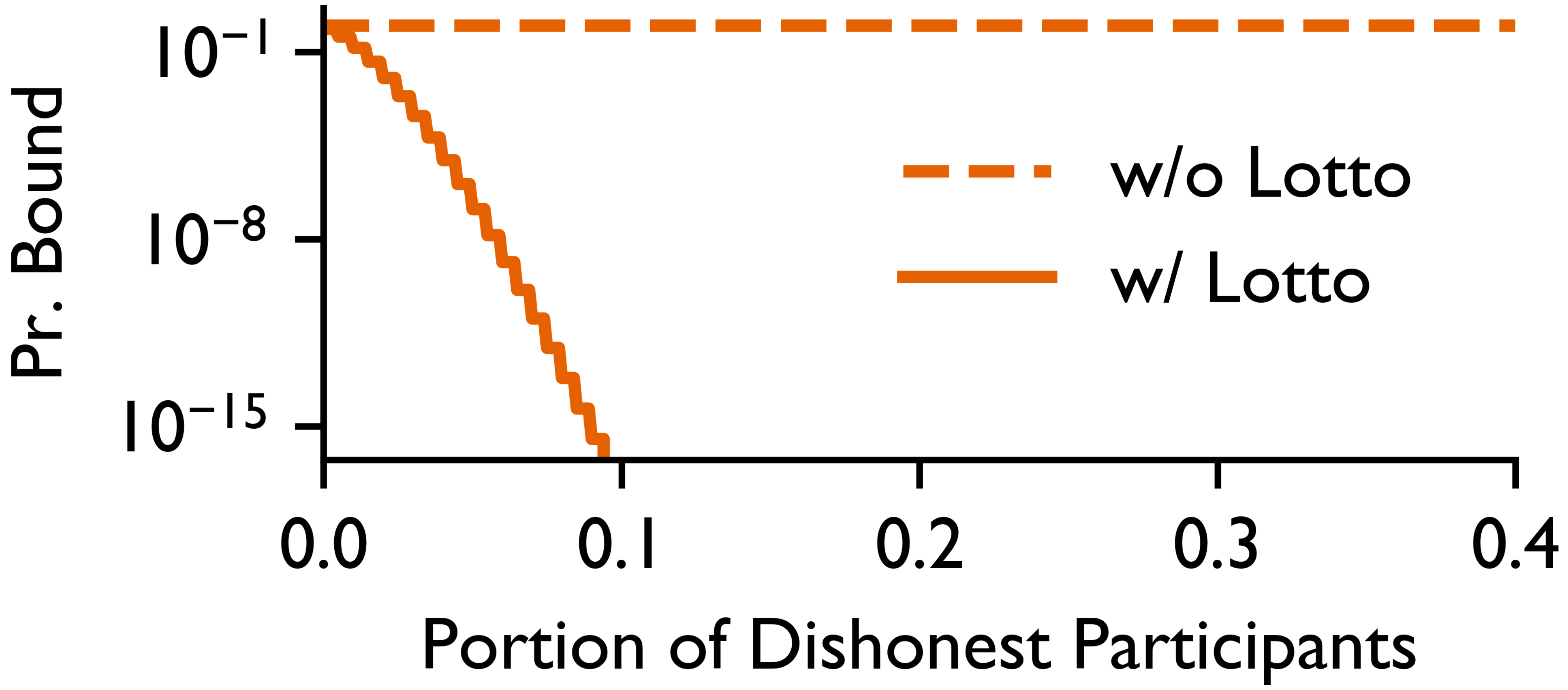
**Align**

w/ high prob.

**Fractions** of dishonest clients

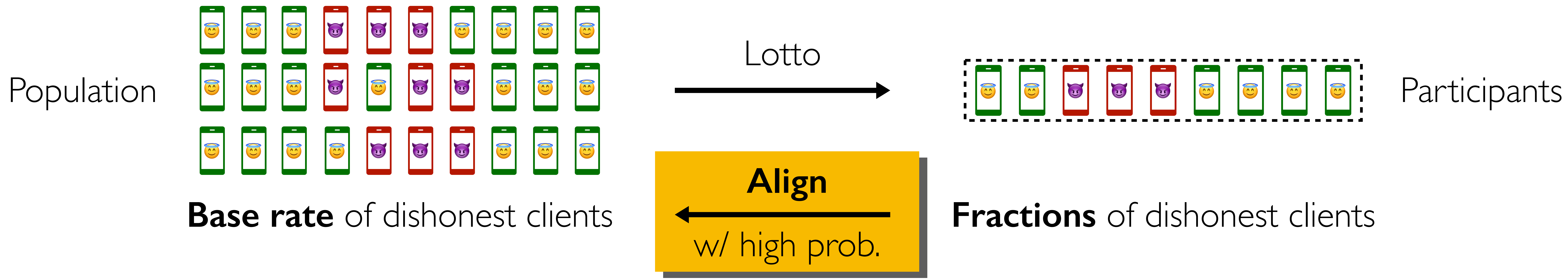
Example

- **Population**: 200,000
- **Dishonesty base rate**: 0.005
- **Target participants**: 200



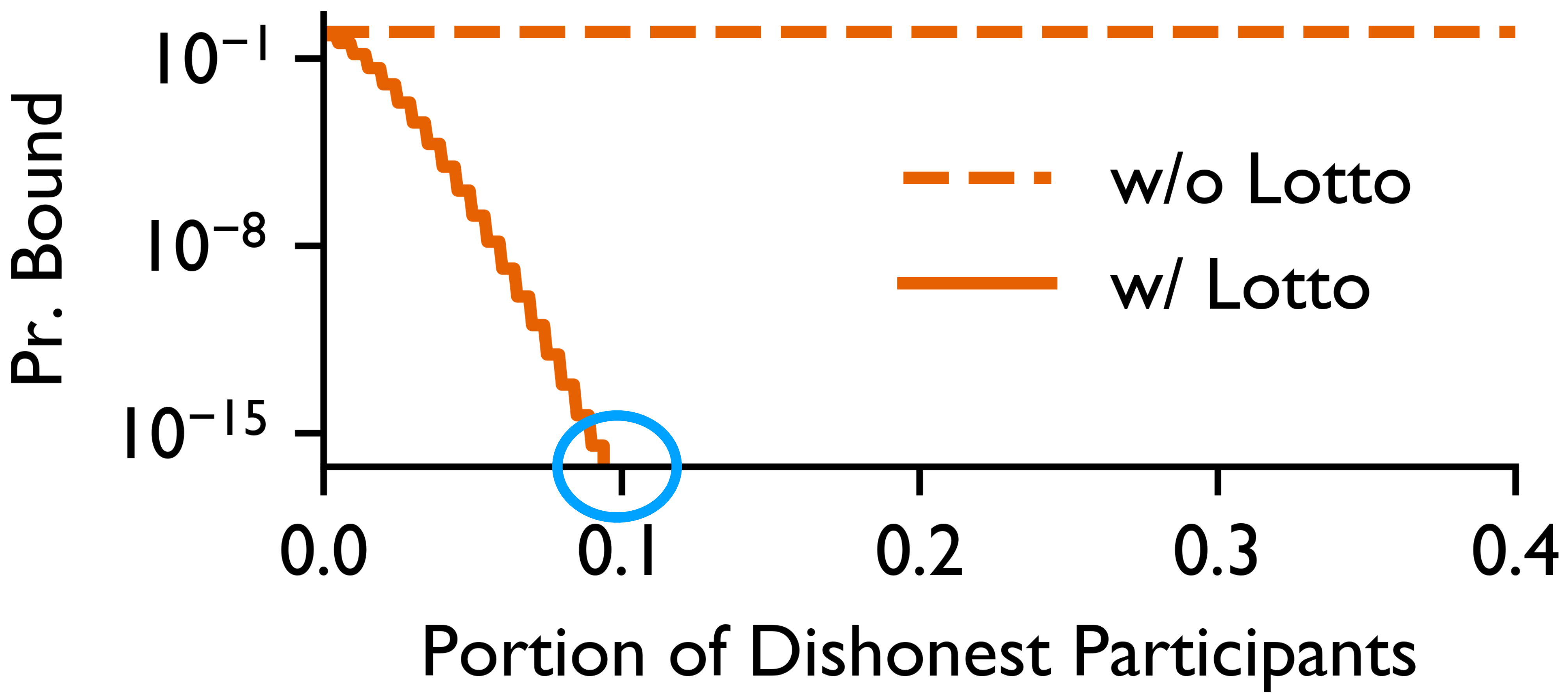
# Lotto prevents arbitrary manipulation

What can be **proven**:




Example

- **Population**: 200,000
- **Dishonesty base rate**: 0.005
- **Target participants**: 200



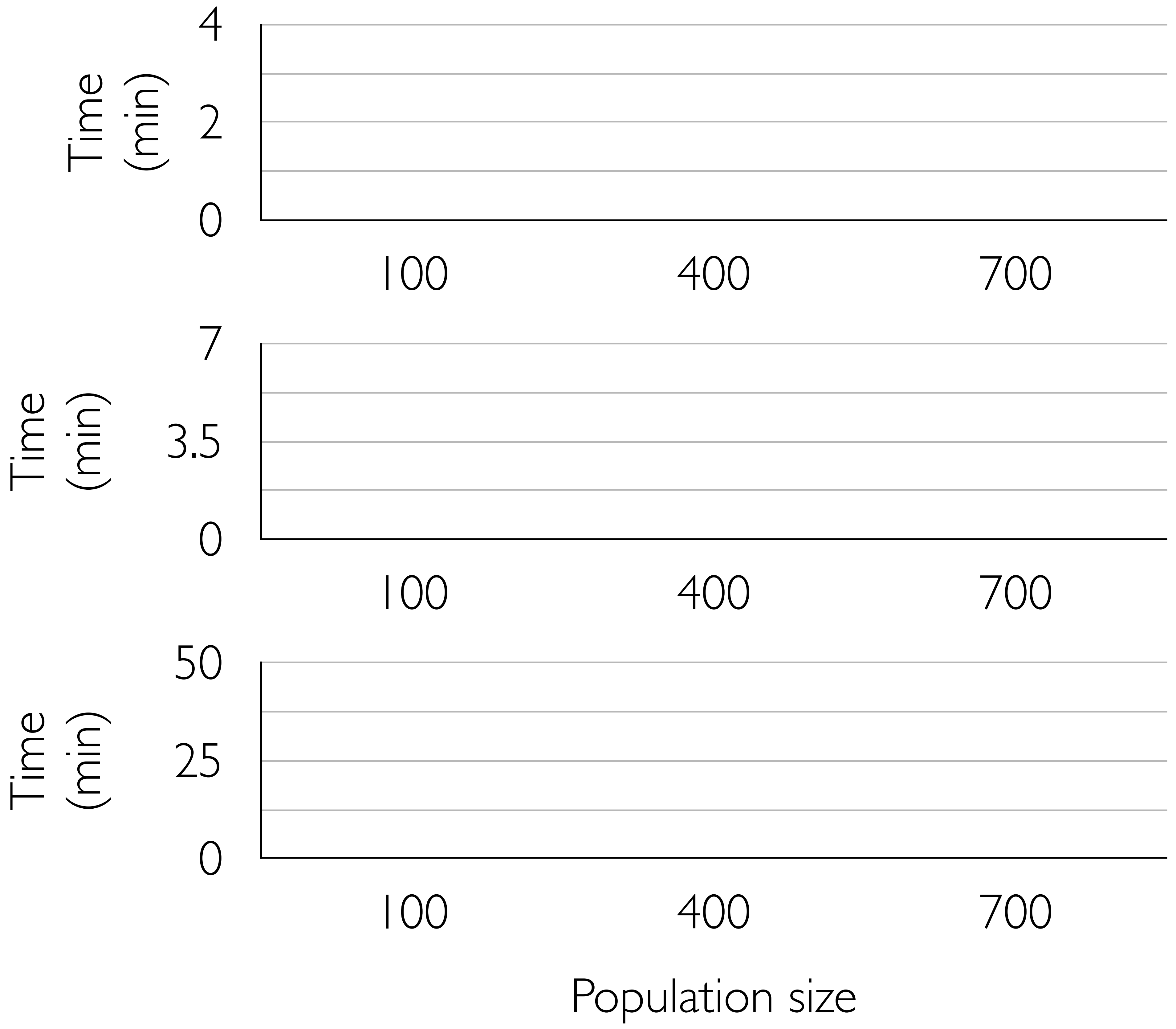
# Lotto induces no or mild overhead

# Lotto induces no or mild overhead

**FEMNIST**
**@CNN**

**OpenImage**
**@MobileNet**

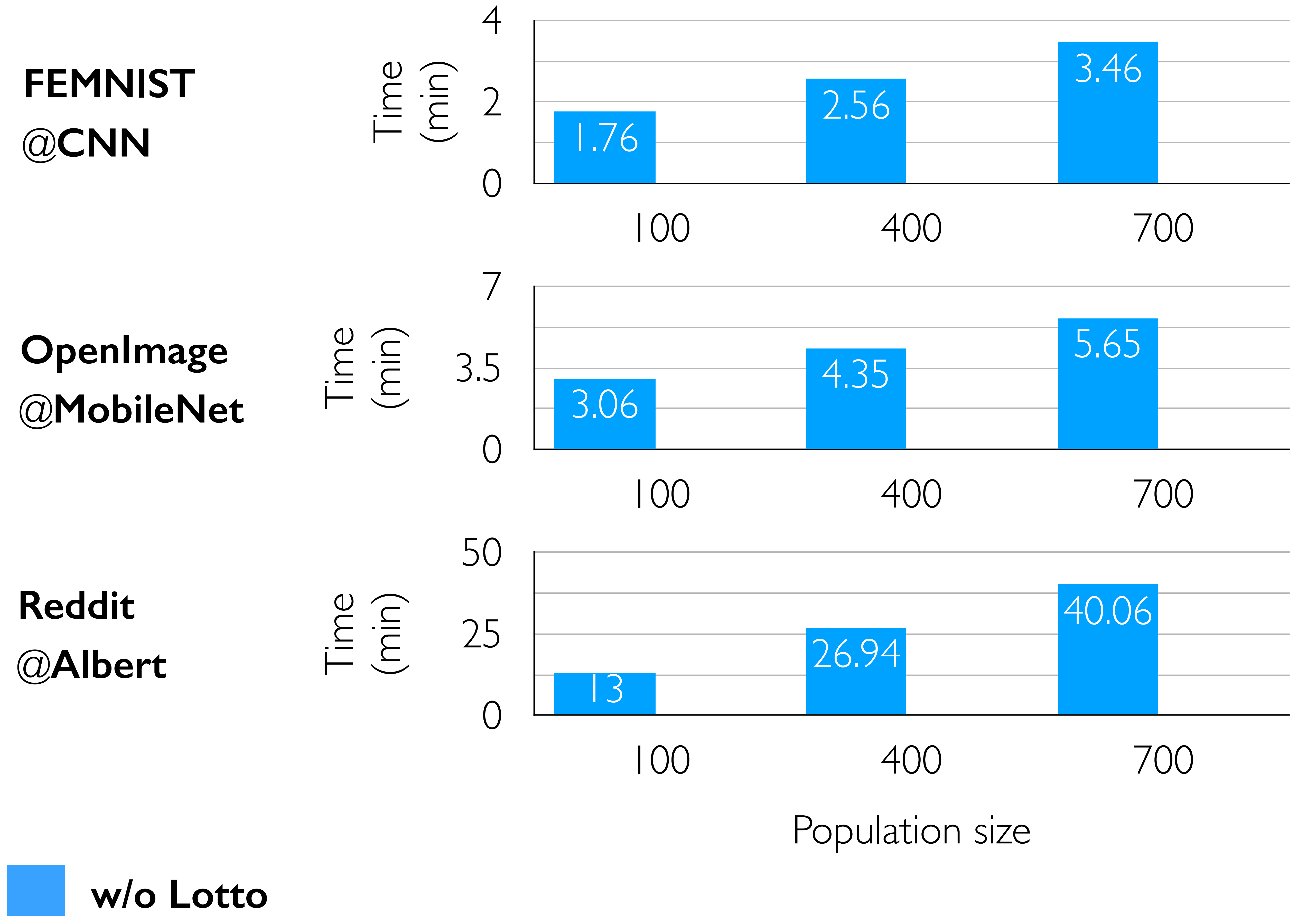
**Reddit**
**@Albert**





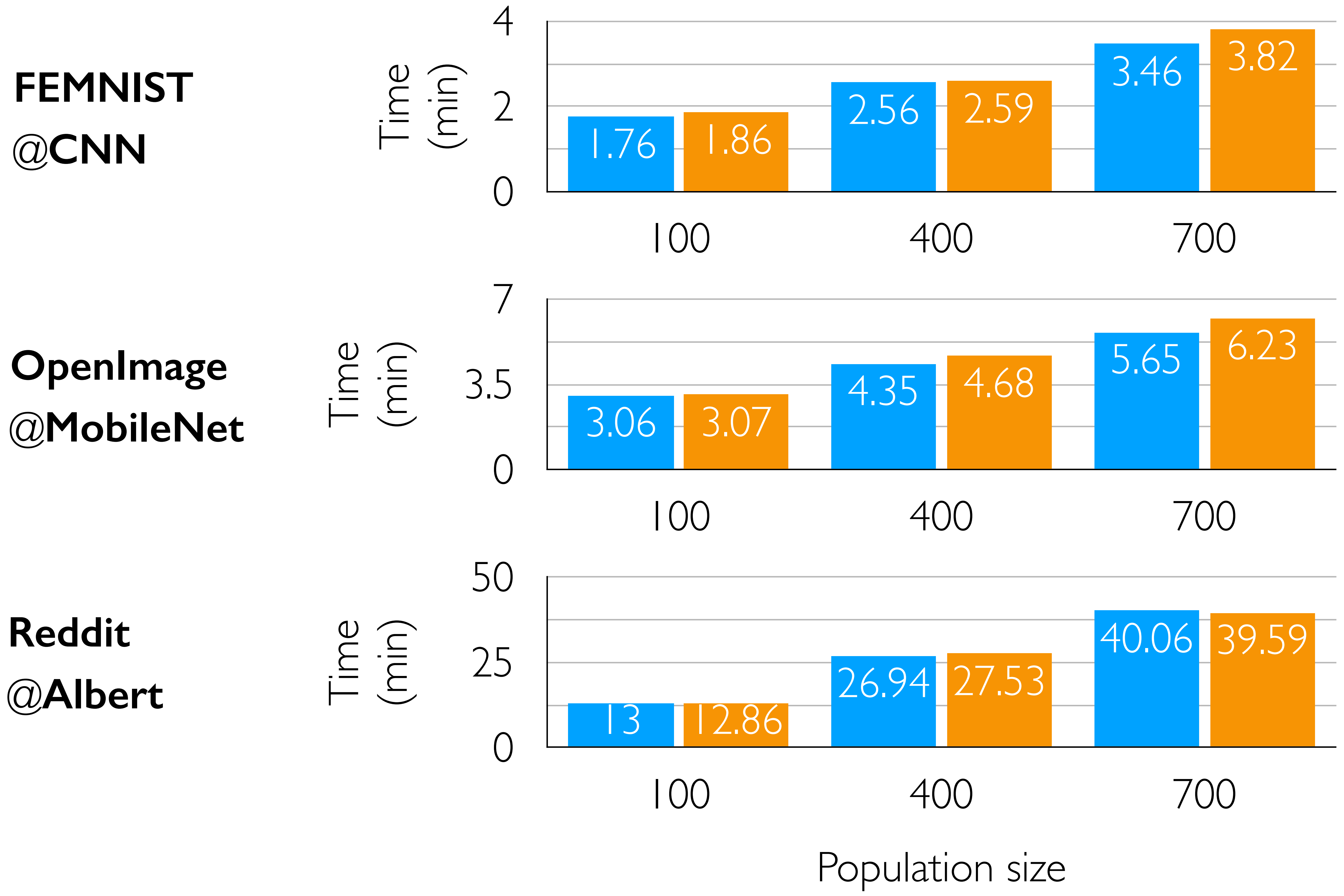

[1]Random selection as an example. See results for informed selection in the paper.

# Lotto induces no or mild overhead




[1]Random selection as an example. See results for informed selection in the paper.

# Lotto induces no or mild overhead

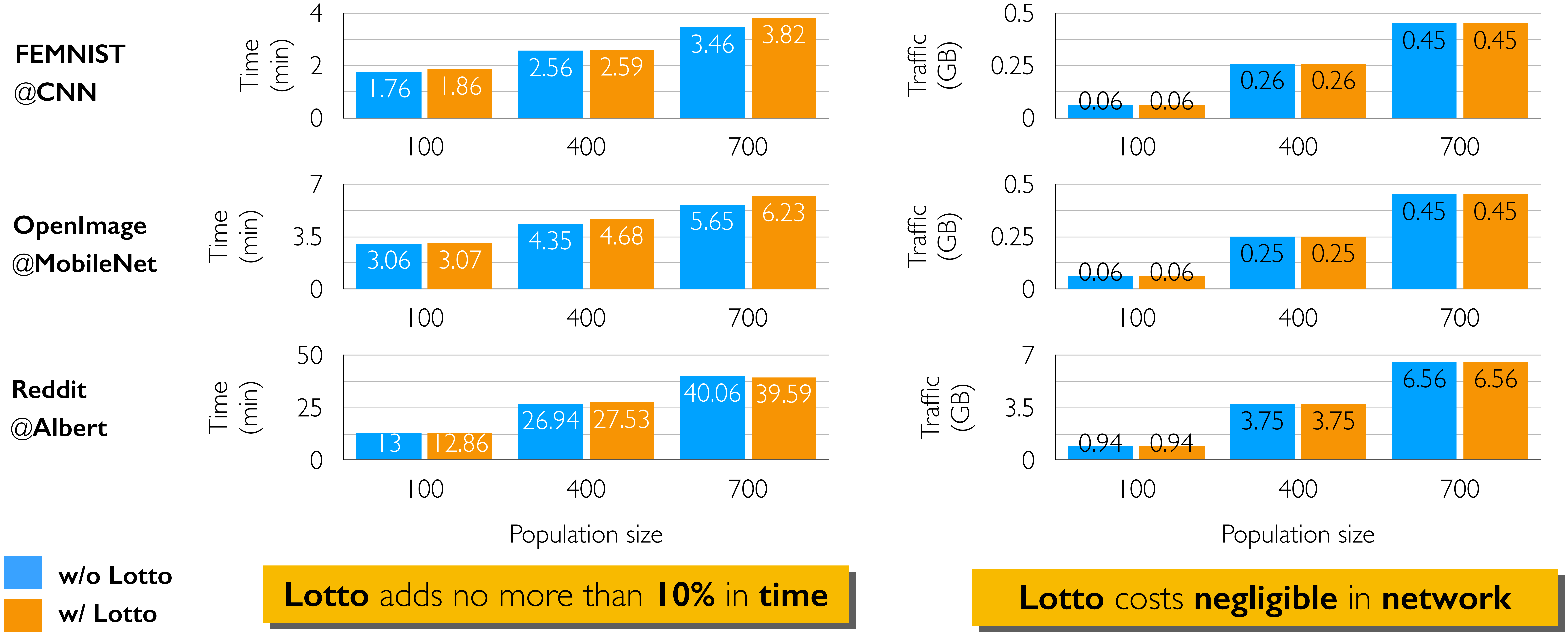



w/o Lotto
w/ Lotto

**Lotto** adds no more than **10%** in **time**

[1]Random selection as an example. See results for informed selection in the paper.

# Lotto induces no or mild overhead




**Lotto** adds no more than **10%** in **time**

**Lotto** costs **negligible** in **network**

[1]Random selection as an example. See results for informed selection in the paper.

# Lotto functions as insecure selectors

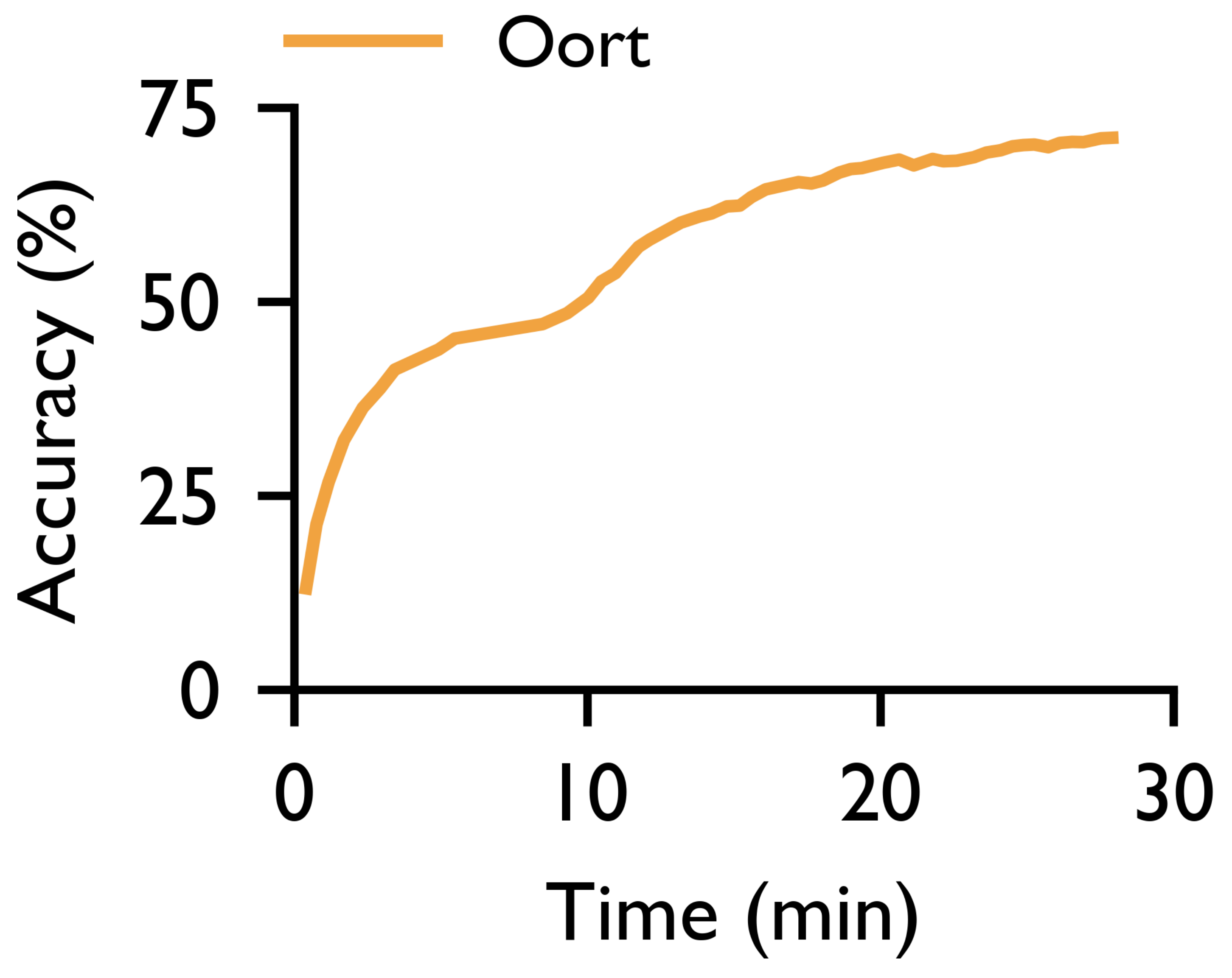
# Lotto functions as insecure selectors

**Oort**[1] → State-of-the-art **informed** selector: optimized for **time-to-accuracy** of training

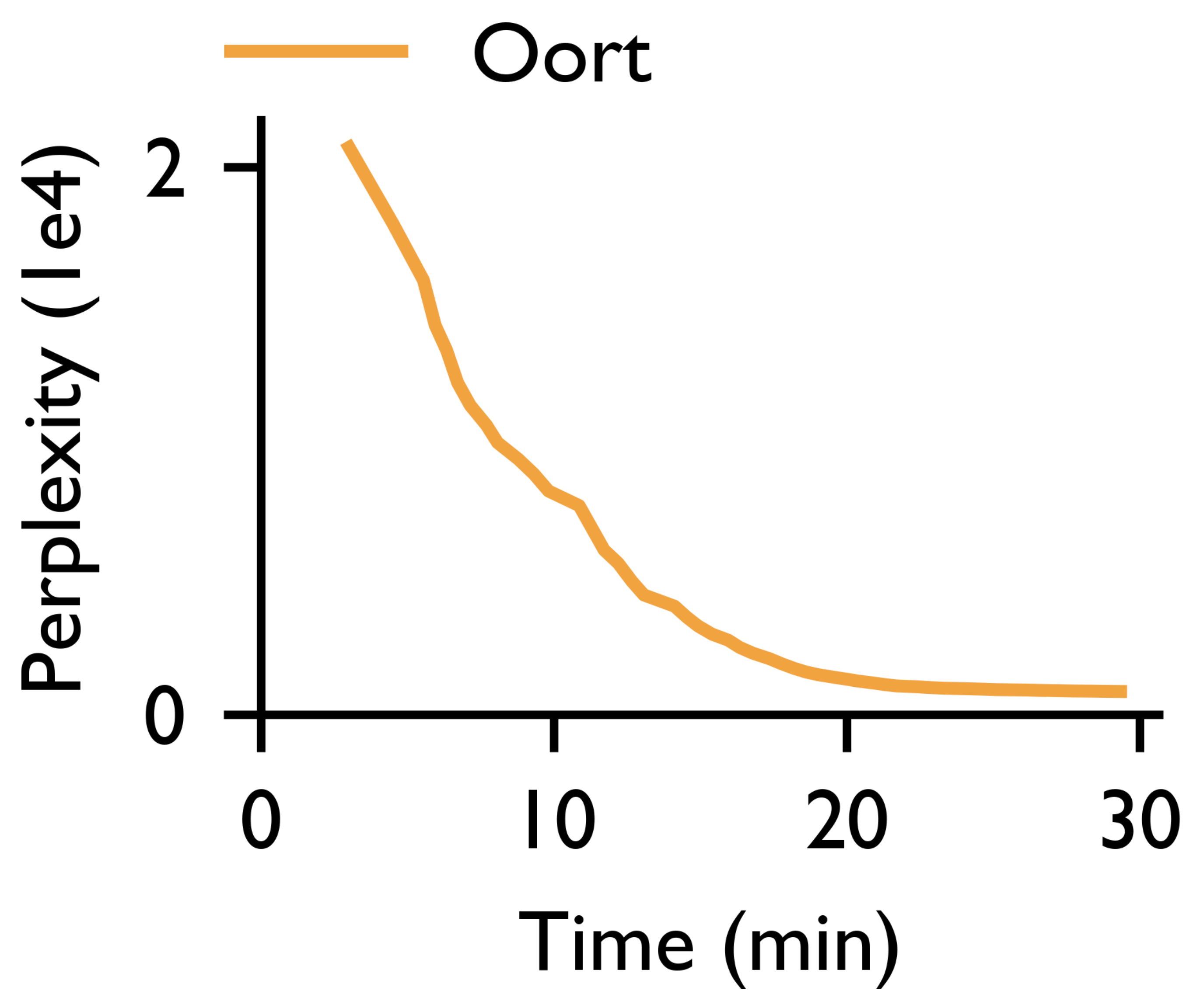
[1] Lai et al. "Oort: Efficient Federated Learning via Guided Participant Selection", In OSDI '21

# Lotto functions as insecure selectors

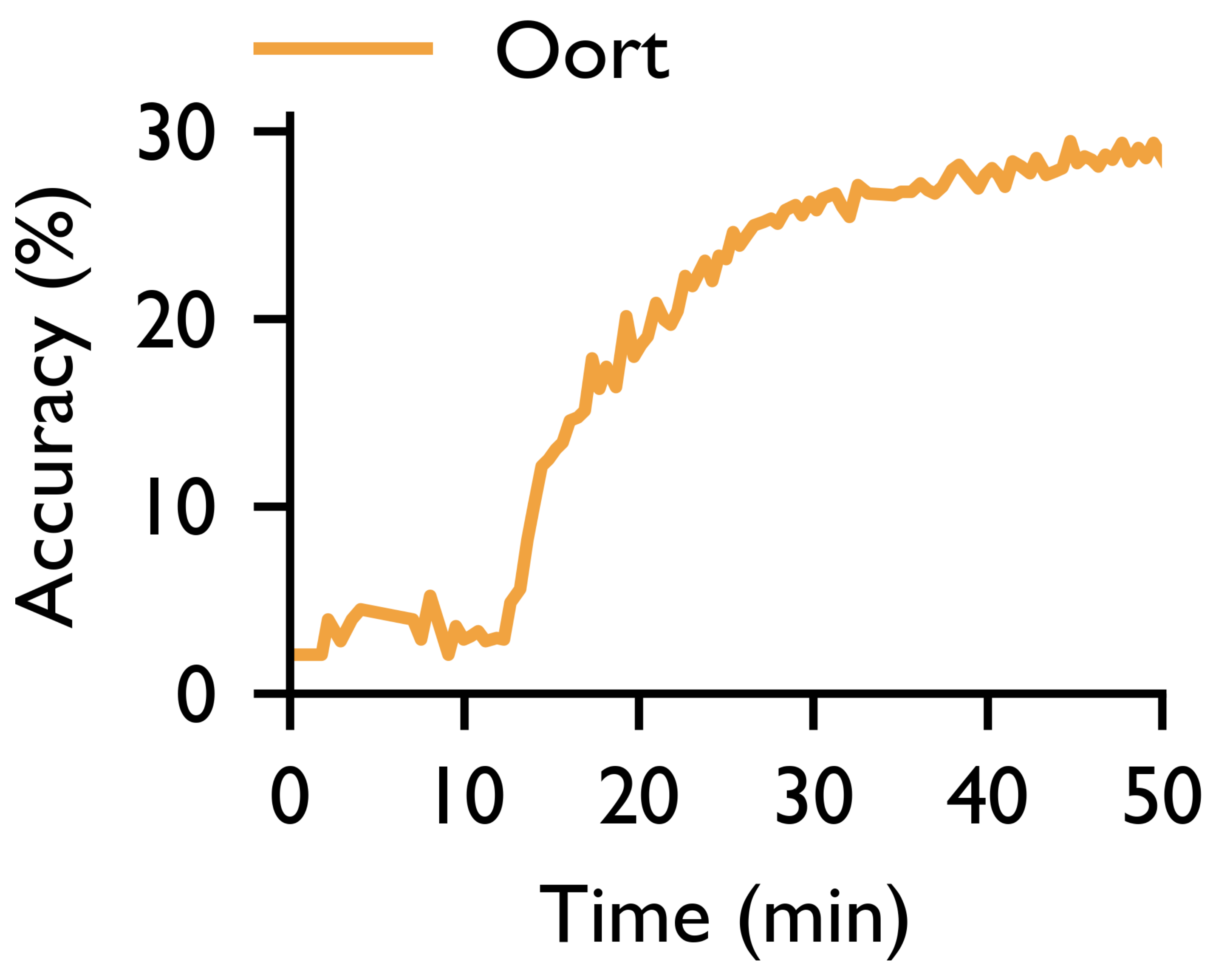
**Oort**[1] → State-of-the-art **informed** selector: optimized for **time-to-accuracy** of training




**FEMNIST@CNN**




**OpenImage@MobileNet**

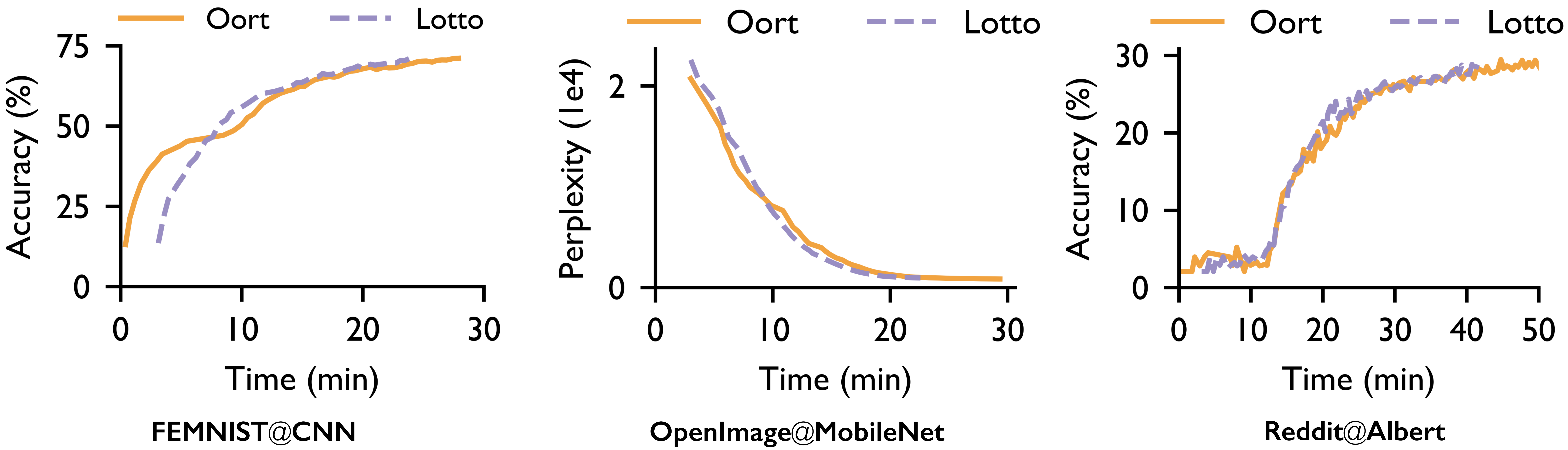



**Reddit@Albert**

[1] Lai et al. "Oort: Efficient Federated Learning via Guided Participant Selection", In OSDI '21

# Lotto functions as insecure selectors

**Oort**[1] → State-of-the-art **informed** selector: optimized for **time-to-accuracy** of training




**Lotto** well approximate Oort with **no cost in time-to-accuracy** performance

[1]Lai et al. "Oort: Efficient Federated Learning via Guided Participant Selection", In OSDI '21

# Lotto: Results summary

**Functionality**

Support both **random (exact)** and **informed (well approximated)** selection

**Security**

**Theoretical guarantee (tight probability bound)** of preventing manipulation

**Efficiency**

Mild **runtime overhead (≤10%)** with no **network cost (<1%)**

github.com/SamuelGong/Lotto

# Summary: My efforts

**Privacy**
Worst-case defense…

**Lotto**

**Efficiency**
Time-to-accuracy…

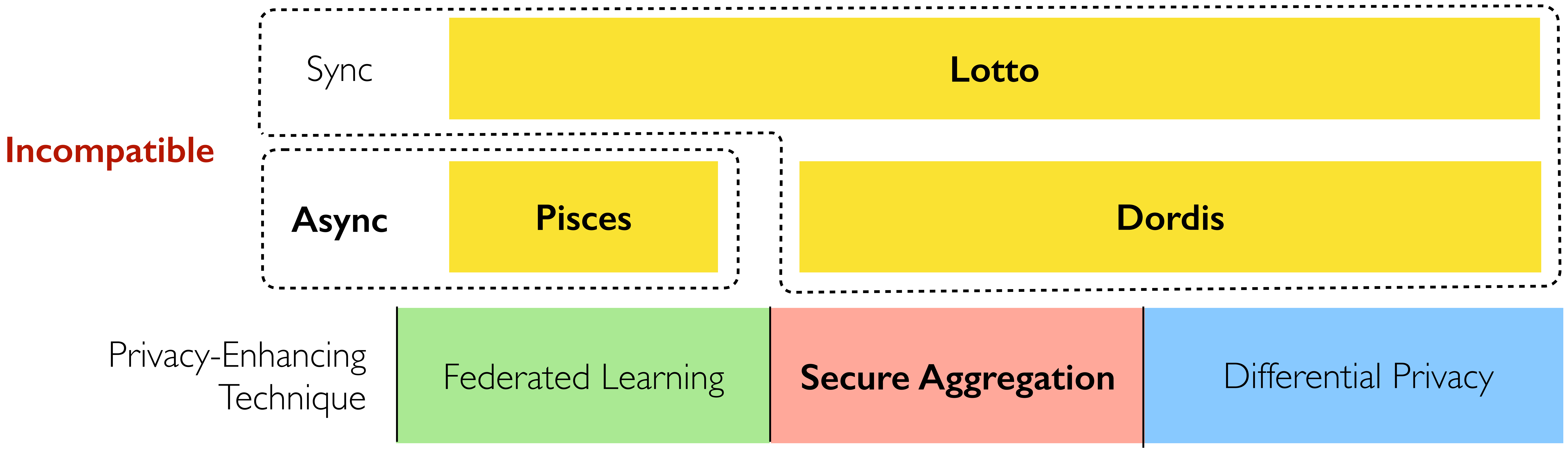
**Pisces**

**Dordis**

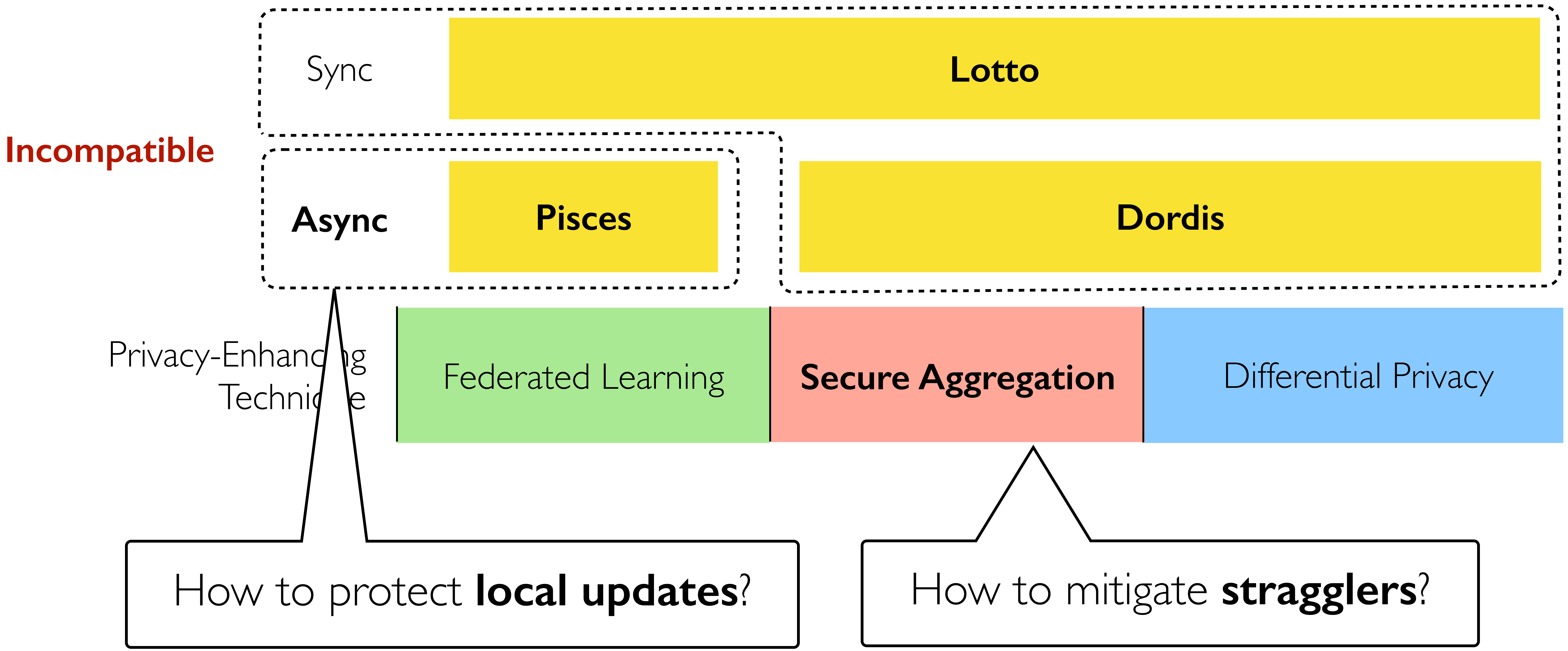
| | | | |
|---|---|---|---|
| **Privacy-Enhancing Technique** | Federated Learning | Secure Aggregation | Differential Privacy |
| **Privacy Guarantee** | Data kept on premises | Local updates unseen | Global update leaks little about any client |

# Future work

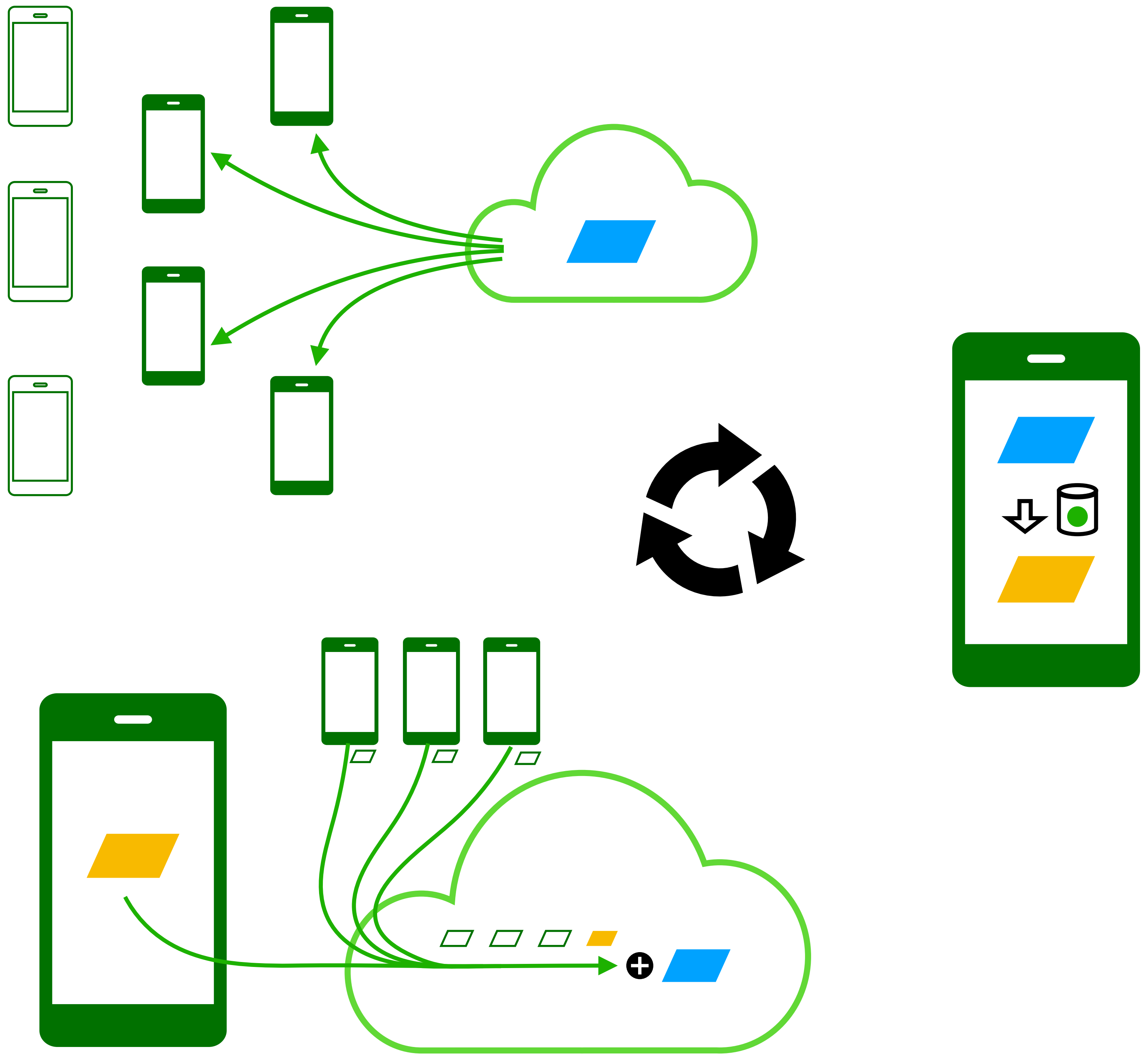
# 1. Better private learning on the edge



# 1. Better private learning on the edge

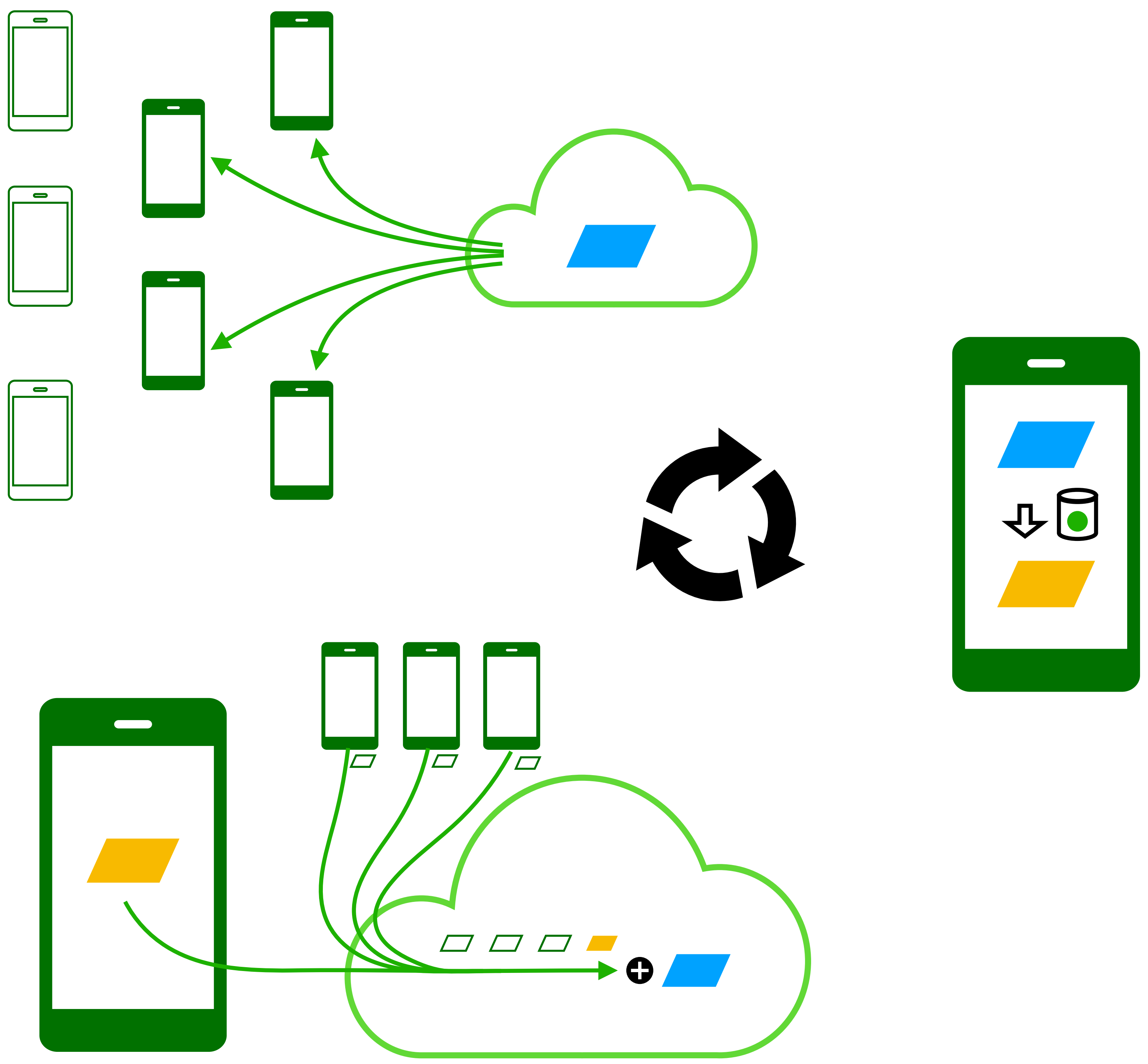


# 2. Private unlearning on the edge

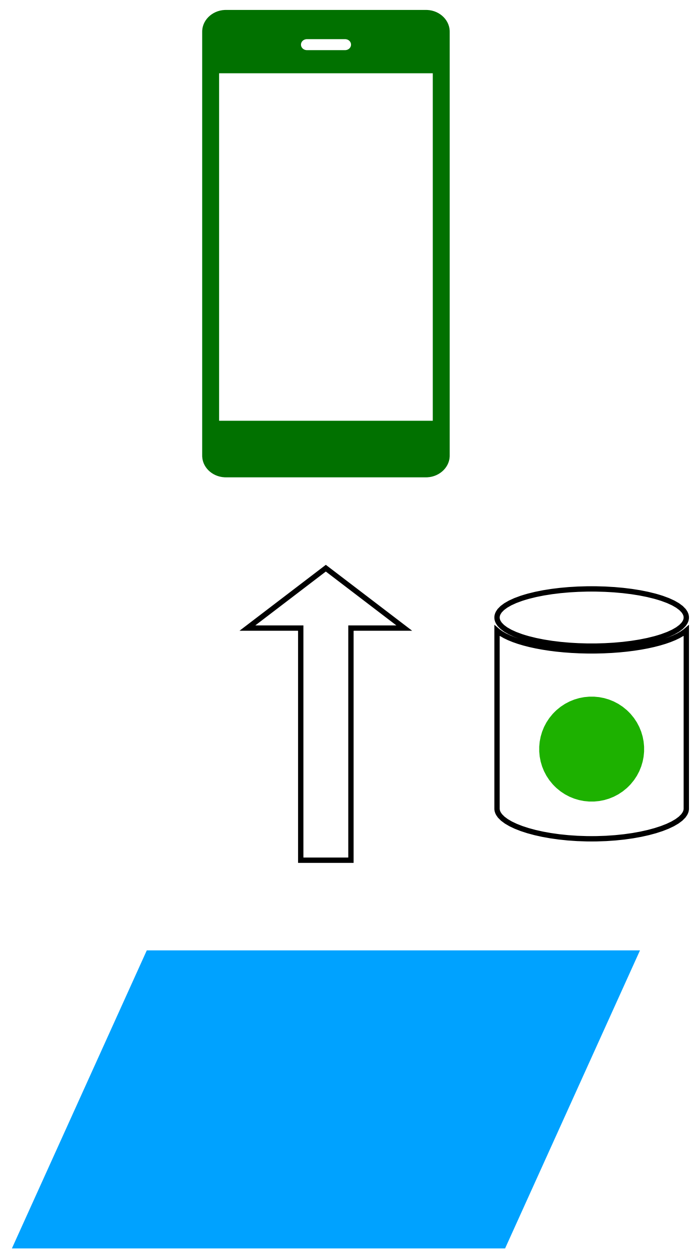


Learning

# 2. Private unlearning on the edge

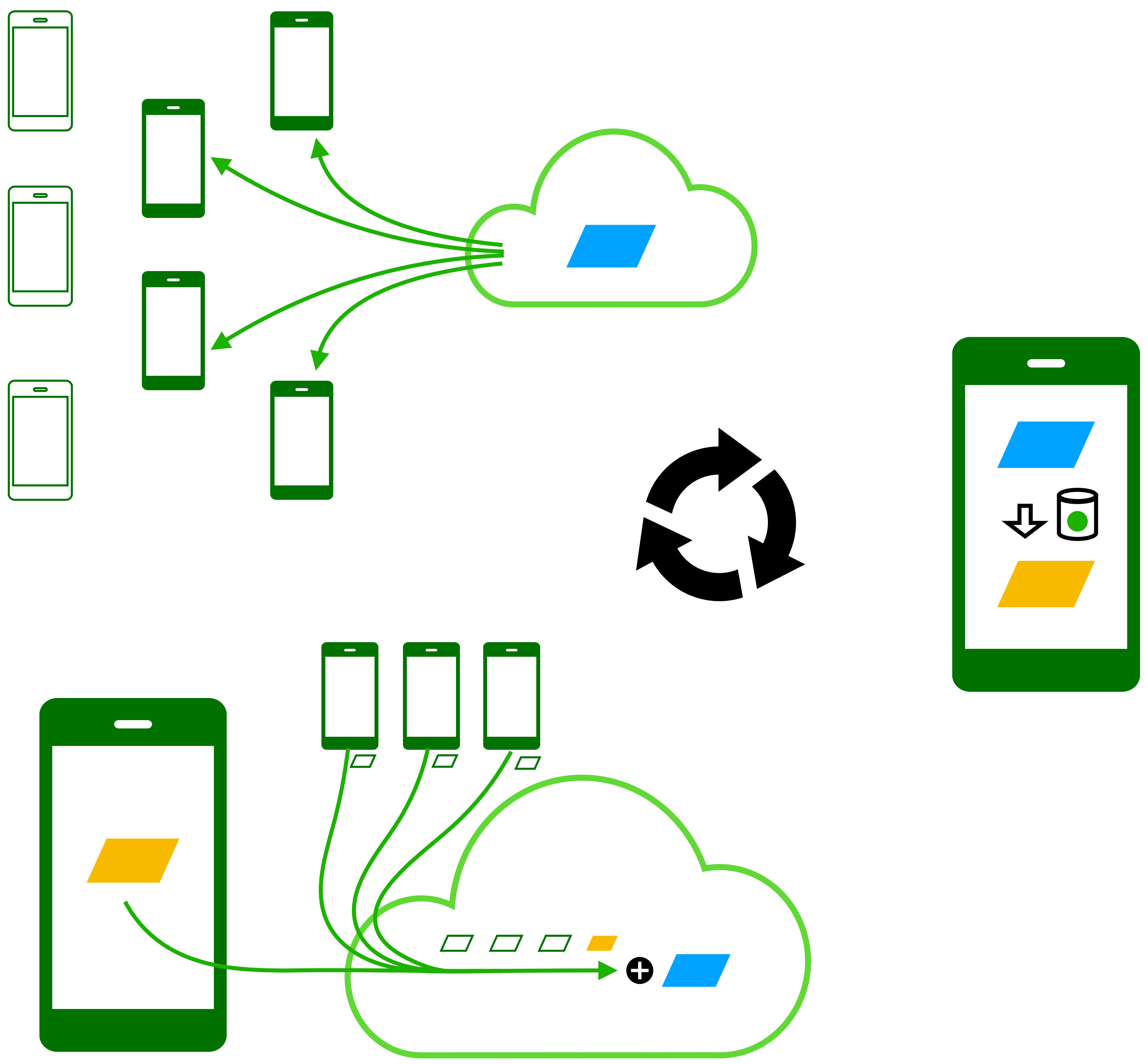


Learning

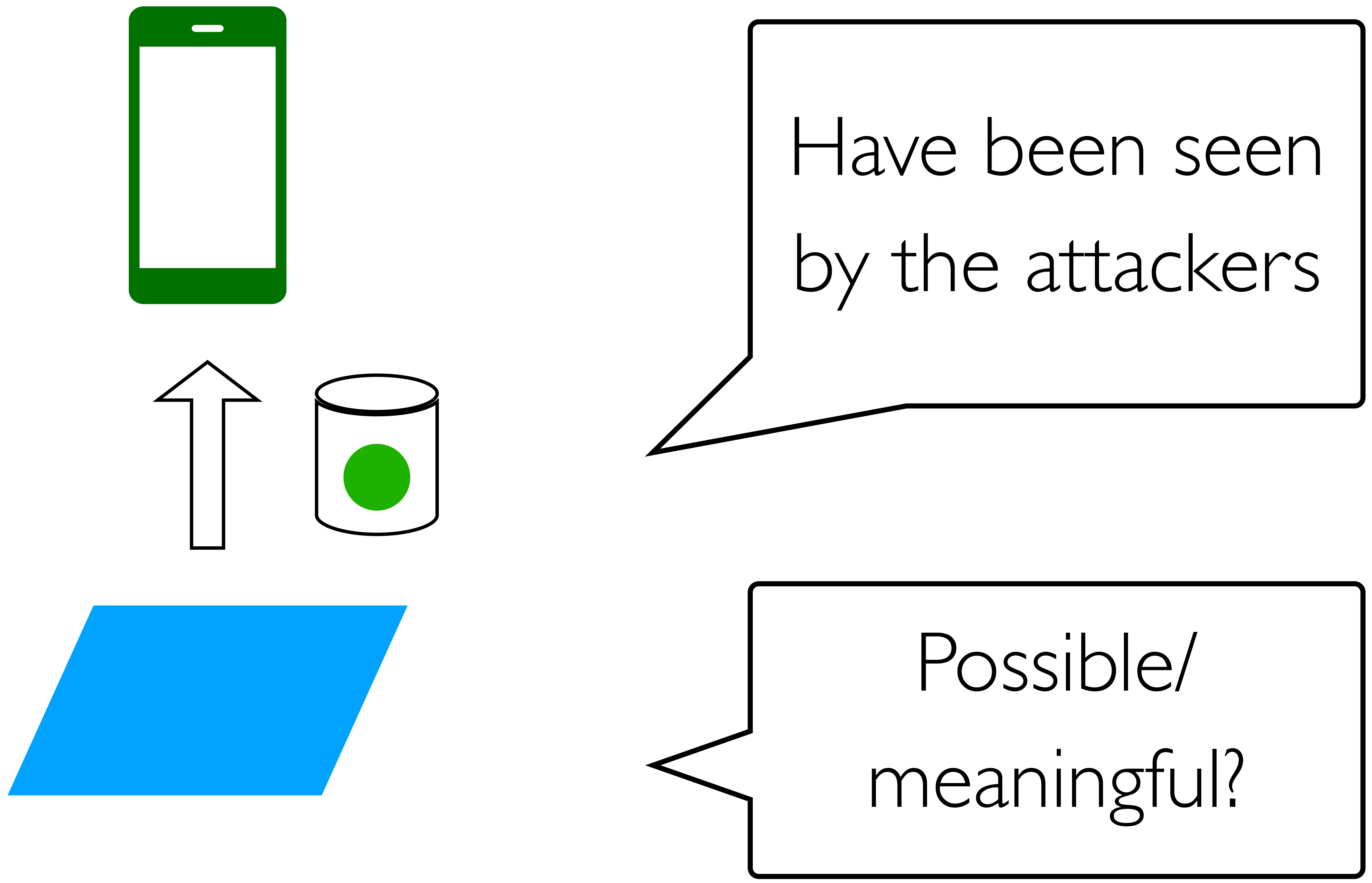


**Unlearning**

# 2. Private unlearning on the edge

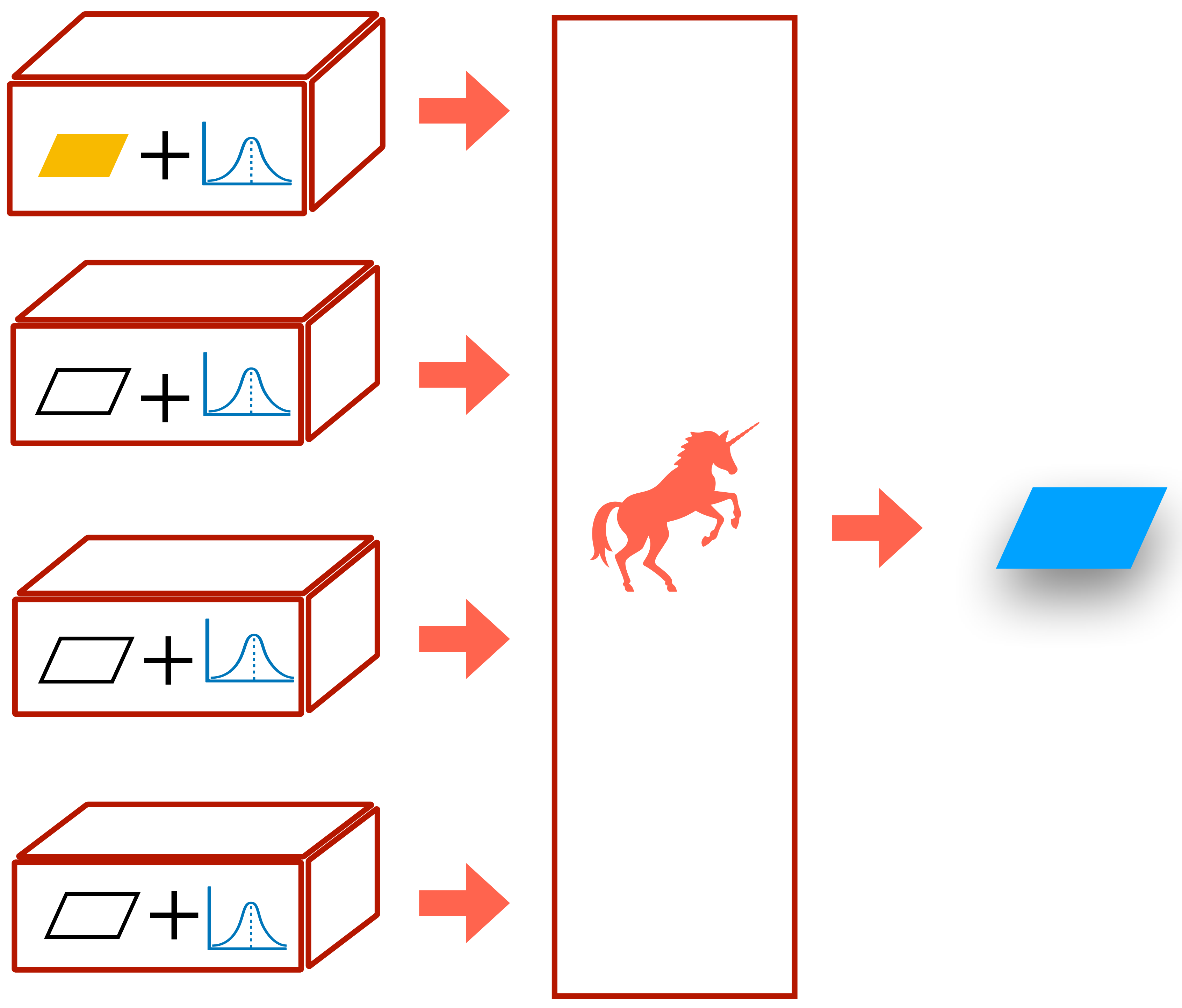


Learning



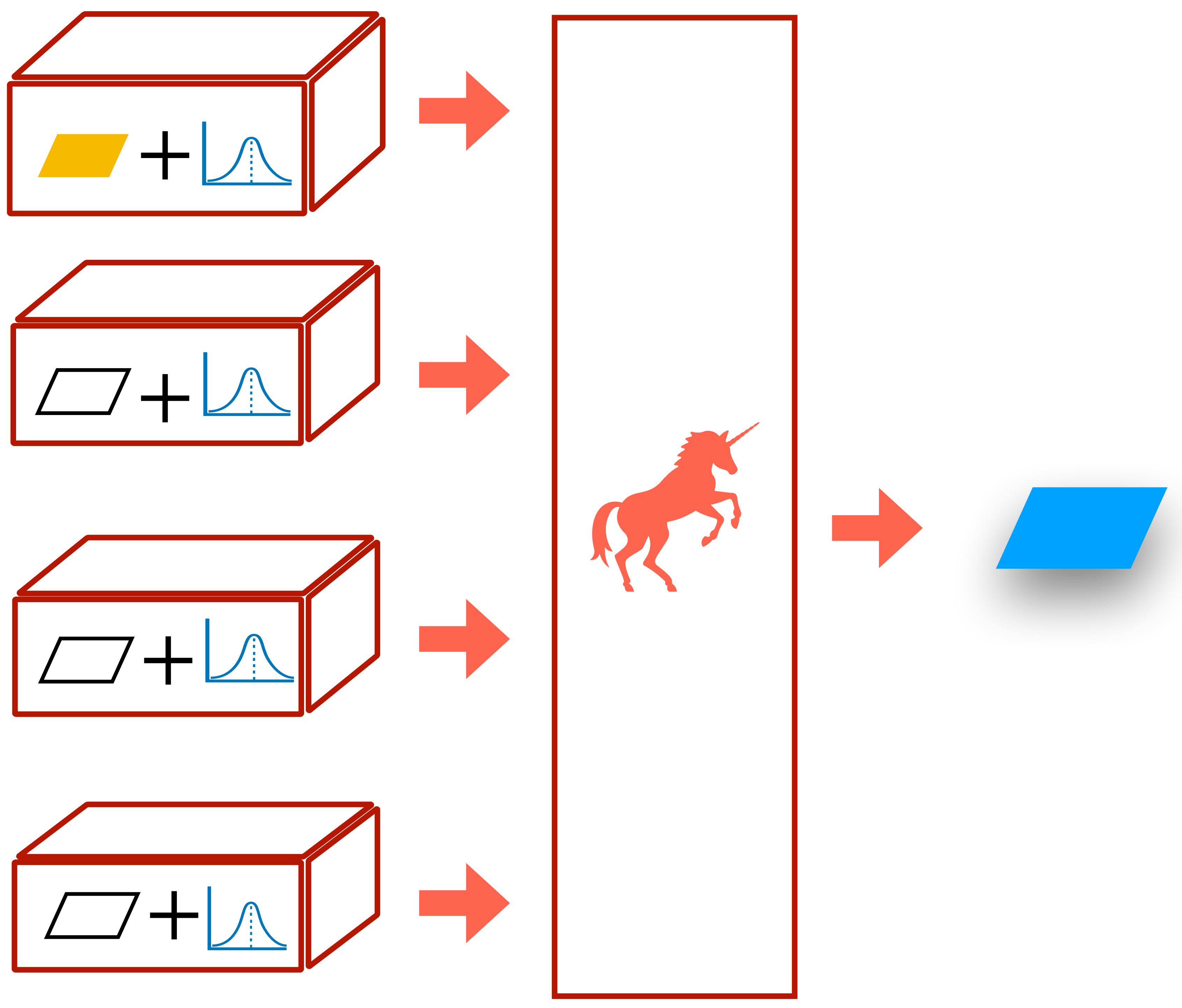

**Unlearning**

# 3. Security: Beyond privacy

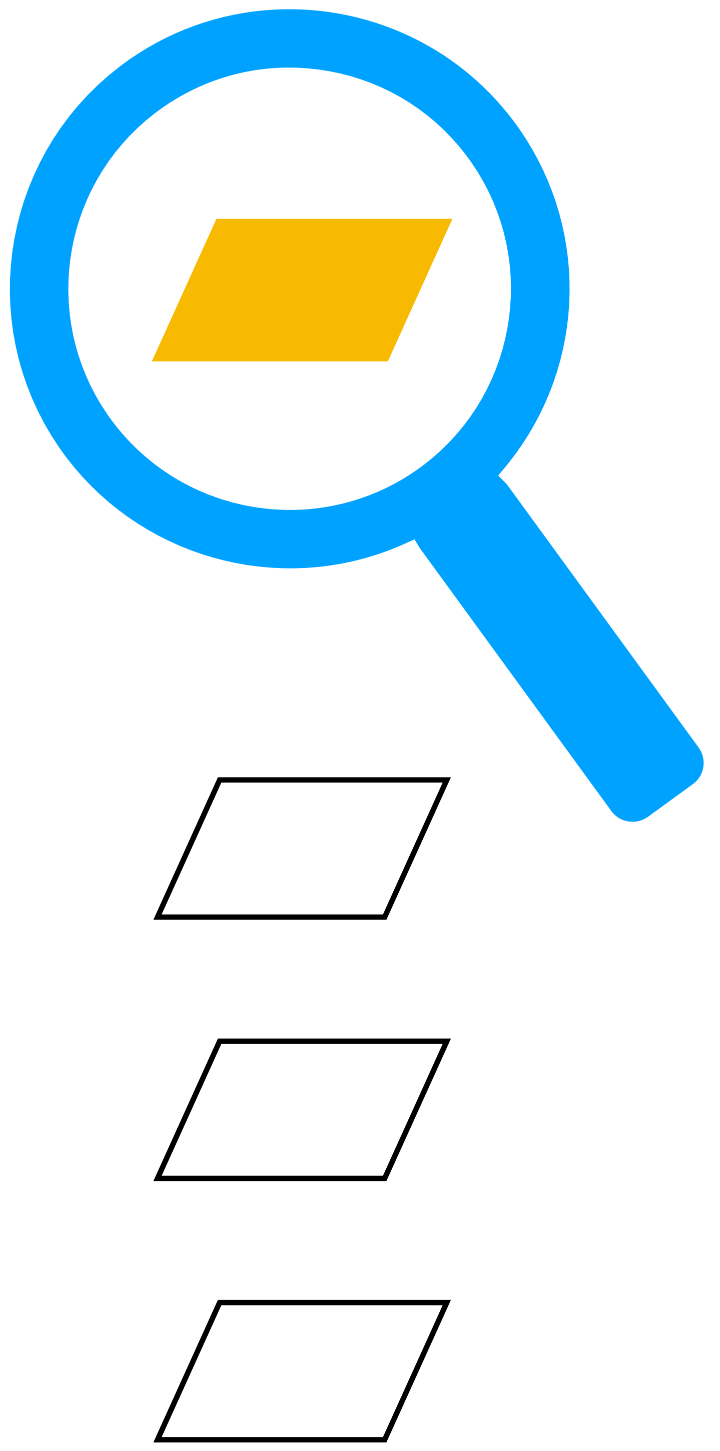


Data privacy

# 3. Security: Beyond privacy



Data privacy



Model security

# List of Publications

1. ☆ Lotto: Secure Participant Selection against Adversarial Servers in Federated Learning. **[Security 2024]**
   - Zhifeng Jiang, Peng Ye, Shiqi He, Wei Wang, Ruichuan Chen, Bo Li
2. ☆ Dordis: Efficient Federated Learning with Dropout-Resilient Differential Privacy. **[EuroSys 2024]**
   - Zhifeng Jiang, Wei Wang, Ruichuan Chen
3. ☆ Pisces: Efficient Federated Learning via Guided Asynchronous Training. **[SoCC 2022]**
   - Zhifeng Jiang, Wei Wang, Baochun Li, Bo Li
4. Towards Efficient Synchronous Federated Training: A Survey on System Optimization Strategies. **[IEEE Trans. Big Data 2022]**
   - Zhifeng Jiang, Wei Wang, Bo Li, Qiang Yang

The publications **covered by this thesis** is marked with ☆

5. Gillis: Serving Large Neural Networks in Serverless Functions with Automatic Model Partitioning. **[ICDCS 2021]**
   - Minchen Yu, Zhifeng Jiang, Hok Chun Ng, Wei Wang, Ruichuan Chen, Bo Li
6. Feature Reconstruction Attacks and Countermeasures of DNN Training in Vertical Federated Learning. **[IEEE TDSC 2024, Pending Major Revision]**
   - Peng Ye, Zhifeng Jiang, Wei Wang, Bo Li, Baochun Li
7. FedCA: Efficient Federated Learning with Client Autonomy. **[In Submission]**
   - Na Lv, Zhi Shen, Chen Chen, Zhifeng Jiang, Jiayi Zhang, Quan Chen, Minyi Guo
8. FLASHE: Additively Symmetric Homomorphic Encryption for Cross-Silo Federated Learning. **[arXiv 2021]**
   - Zhifeng Jiang, Wei Wang, Yang Liu