Private aggregation at scale with lightweight distributed trust

Yiping Ma
UC Berkeley

Trust in Decentralized Systems, Simons Institute, 2026

Motivation: Private aggregation for federated learning (FL)

Federated learning

Average of $\begin{bmatrix}1.2\\0.1\\7.0\\\dots\\4.3\\2.9\\1.1\end{bmatrix}, \dots, \begin{bmatrix}1.1\\0.2\\7.9\\\dots\\3.0\\2.2\\1.5\end{bmatrix}$

$\begin{bmatrix}1.2\\0.1\\7.0\\\dots\\4.3\\2.9\\1.1\end{bmatrix}$ ← Train(local data)

Motivation: Private aggregation for federated learning (FL)

Private aggregation

Server learns only $\sum \mathbf{x}_i$

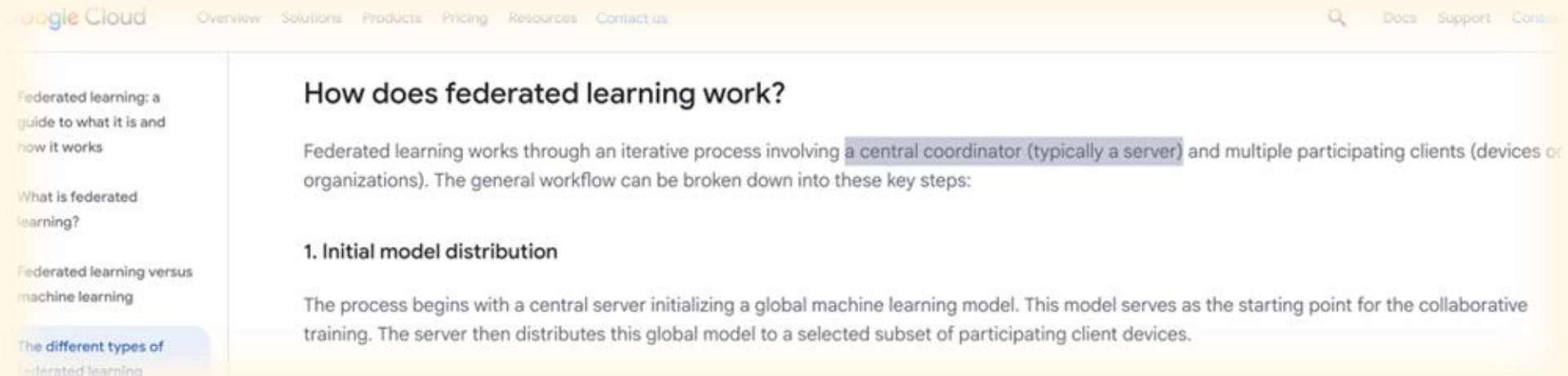
Many instances of aggregation!

$\mathbf{x}_1$ $\mathbf{x}_2$ ...

Client i holds input $\mathbf{x}_i$

Problem: FL setting has stringent system constraints

- **A single untrusted server**



Problem: FL setting has stringent system constraints

- **A single untrusted server**

  → Restrict the private solution to work with only one server!

Problem: FL setting has stringent system constraints

- **A single untrusted server**

  → Restrict the private solution to work with only one server!

- **Many mobile clients**
  - Weak computational power
  - Unstable in execution

Problem: FL setting has stringent system constraints

- **A single untrusted server**

  → Restrict the private solution to work with only one server!

- **Many mobile clients**
  - Weak computational power → Client part must be lightweight
  - Unstable in execution

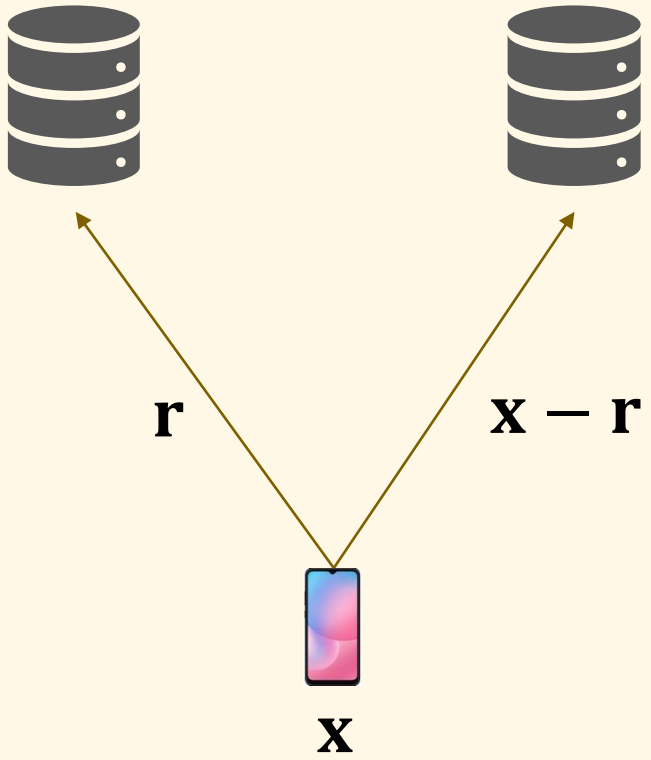
Problem: FL setting has stringent system constraints

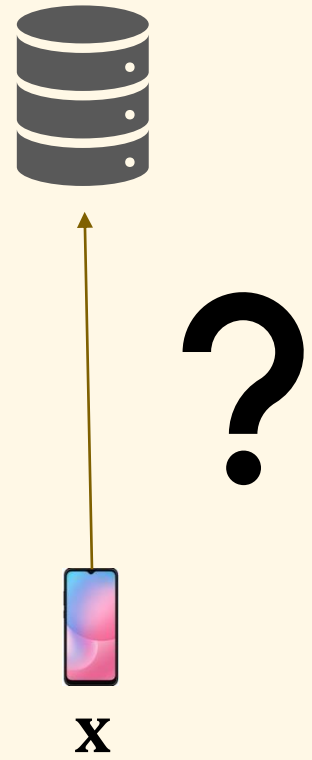
- **A single untrusted server**

  → Restrict the private solution to work with only one server!

- **Many mobile clients**
  - Weak computational power → Client part must be lightweight
  - Unstable in execution → Protocol needs to handle failures

Problem: Classical multi-server solutions do not work



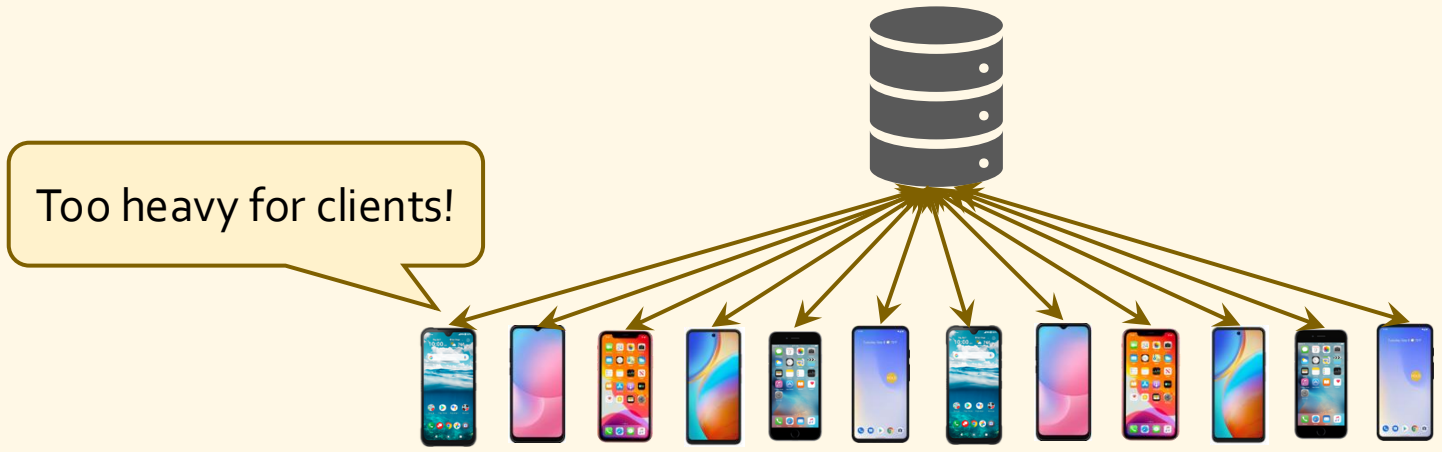


Classical idea: Distribute trust among clients

- **A naïve solution**: A generic MPC among clients
  - Clients have point-to-point channels
  - Require each client to talk to every other client

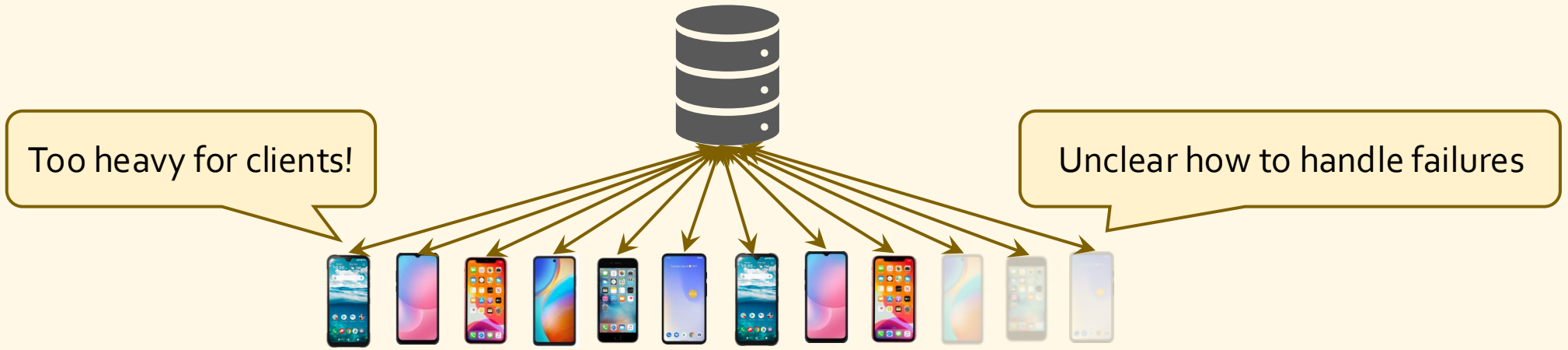
Classical idea: Distribute trust among clients

- **A naïve solution**: A generic MPC among clients
  - ~~Clients have point-to-point channels~~ Client communicate through the server
  - Require each client to talk to every other client



Classical idea: Distribute trust among clients

- **A naïve solution**: A generic MPC among clients
  - ~~Clients have point-to-point channels~~ Client communicate through the server
  - Require each client to talk to every other client



Prior work has greatly improved client cost

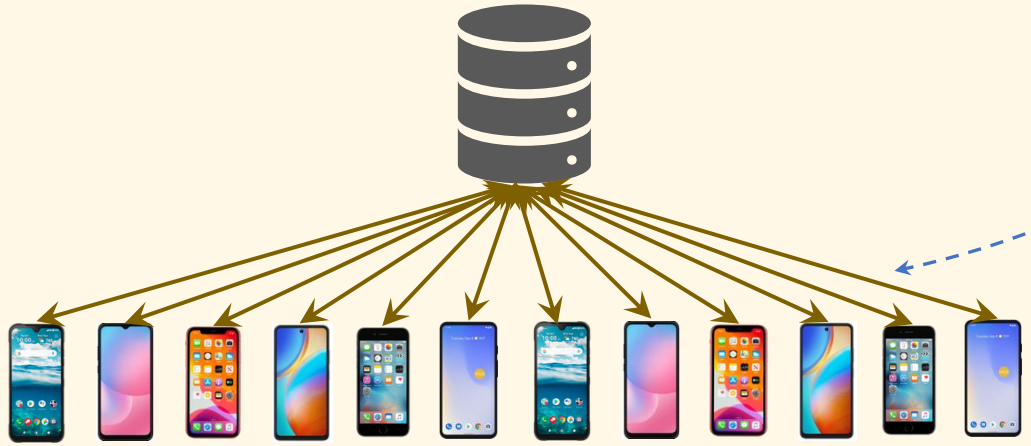
Parameters: input length ℓ, number of clients n.

	Handle failures?	Client comp.	Client comm.	Rounds
Naïve MPC	✗	$O(\ell \cdot n)$	$O(\ell \cdot n)$	2
Bonawitz et al. 2017	✓	$O(\ell \cdot n)$	$O(\ell + n)$	6

Prior work has greatly improved client cost

Parameters: input length ℓ, number of clients n.

	Handle failures?	Client comp.	Client comm.	Rounds
Naïve MPC	✗	$O(\ell \cdot n)$	$O(\ell \cdot n)$	2
Bonawitz et al. 2017	✓	$O(\ell \cdot n)$	$O(\ell + n)$	6
Bell et al. 2020	✓	$O(\ell + \text{polylog } n)$	$O(\ell + \text{polylog } n)$	6



Problem: End-to-end performance is bottlenecked by rounds

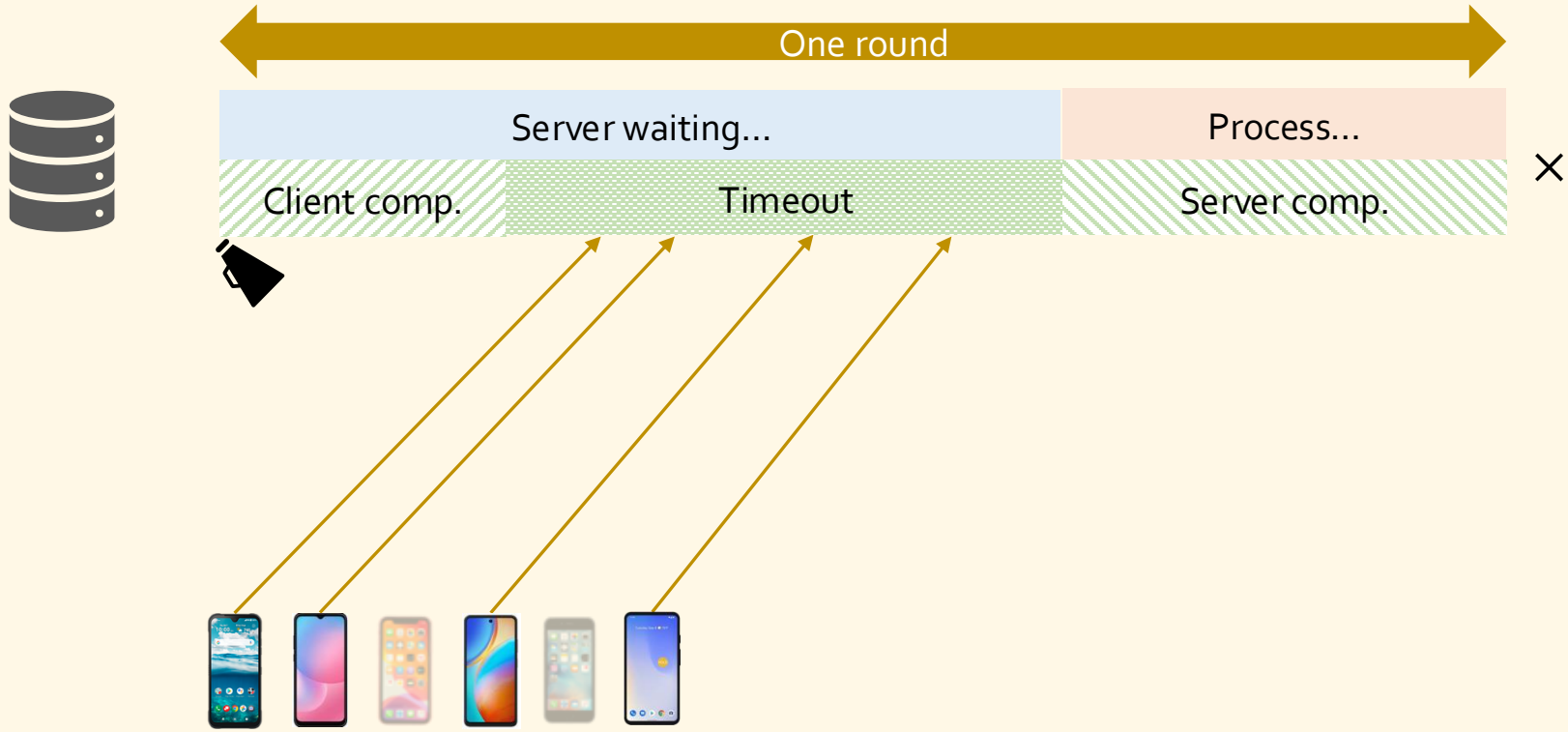
- Not only the number of rounds
- But each round the server needs to interact with all the clients

- **The key issue**: their design requires the server to wait for sufficiently many client responses to proceed

Problem: Bottlenecked by waiting on clients

Decreasing the timeout?

- Fewer clients included in the sum
- Can break the failure tolerance guarantee!

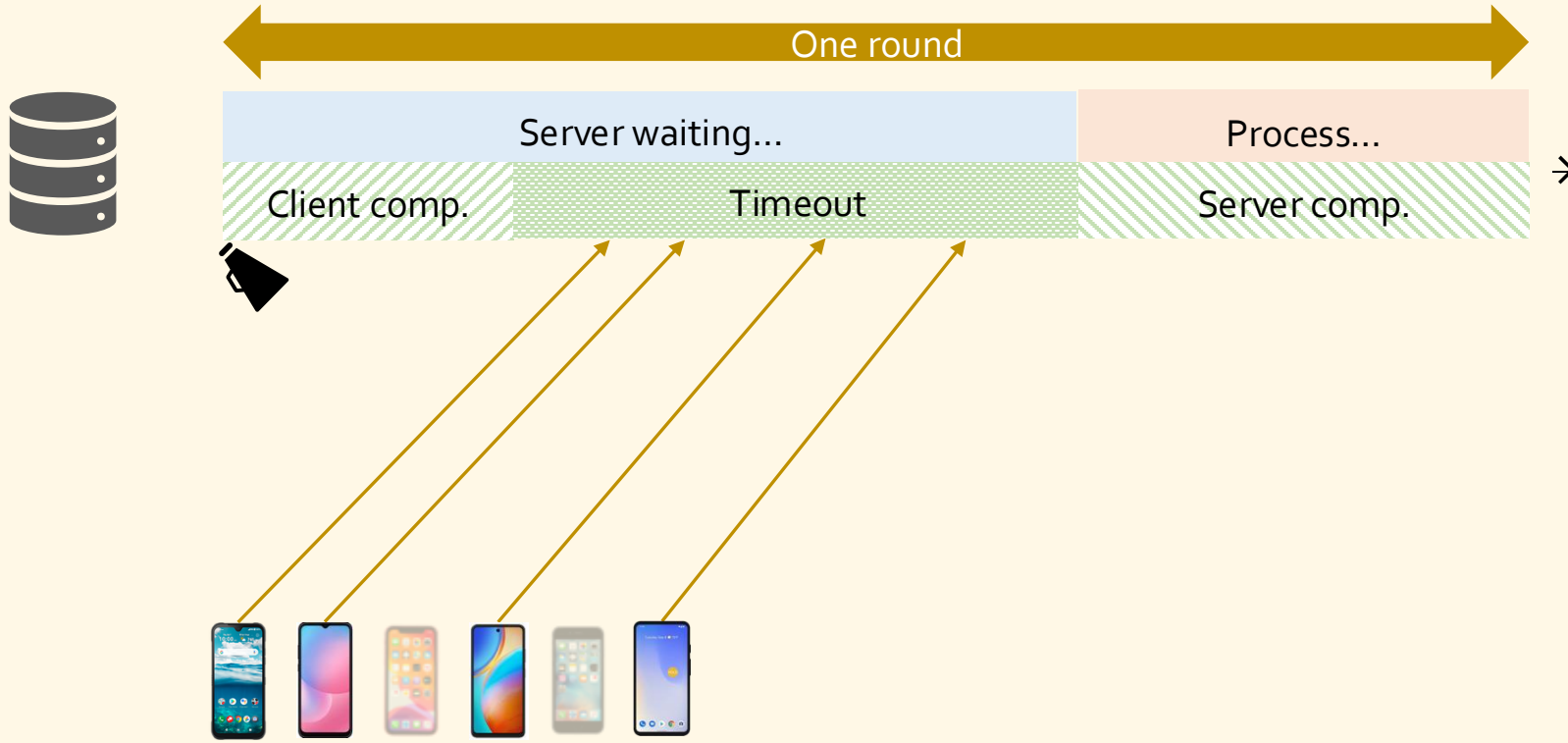


× the number of rounds

Observation

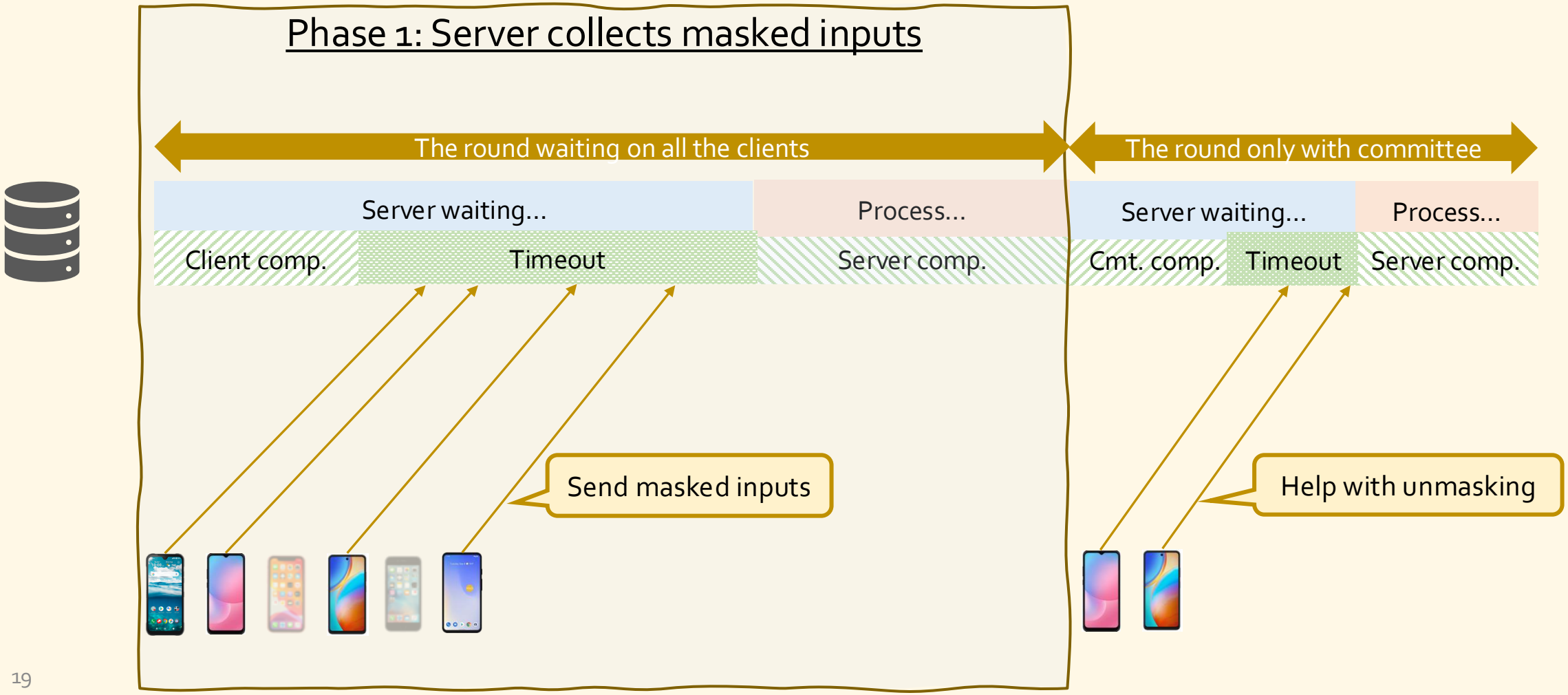
- Essentially only **one round** needs to wait on **all** the clients
- A round that asks all the clients to submit their inputs (but in masked form)
- After that, unmasking doesn't fundamentally need all-client involvement

Our approach: A new aggregation paradigm

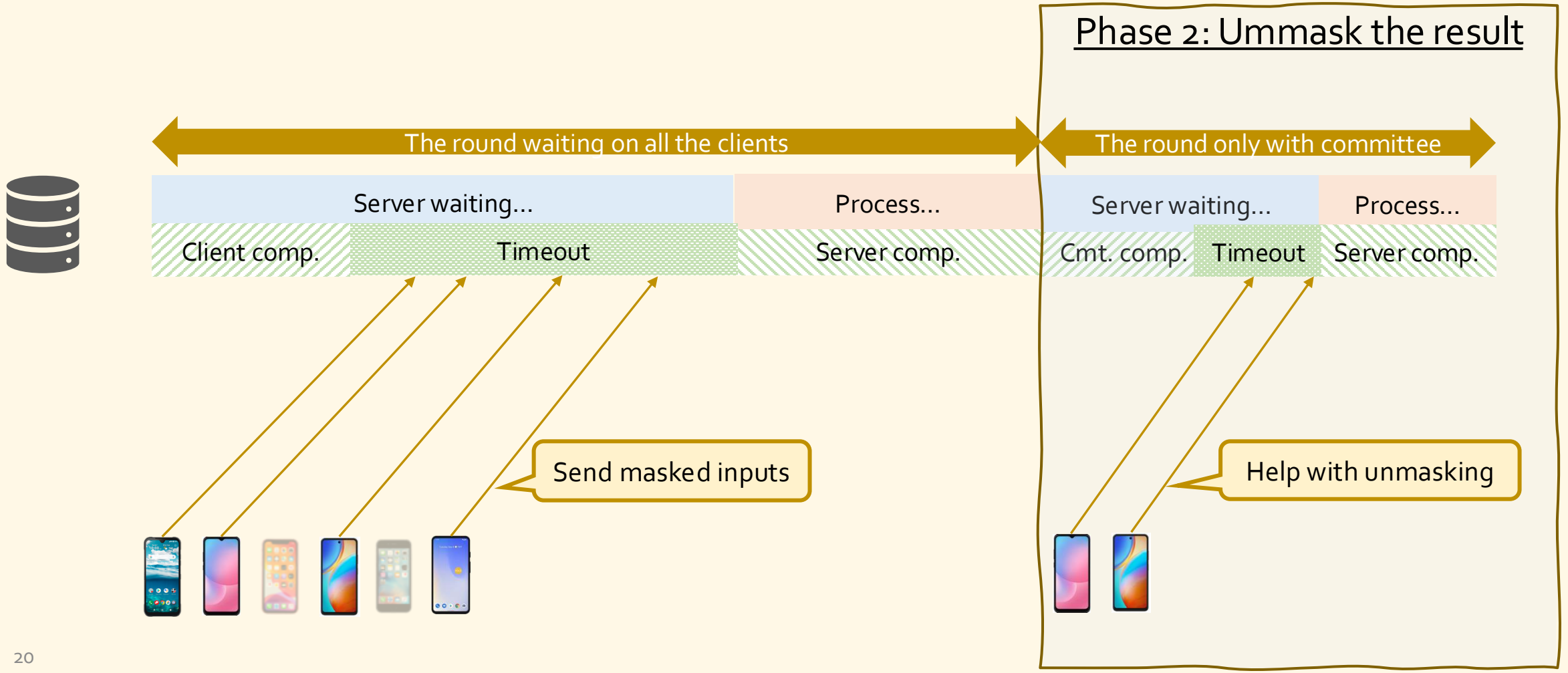


~~× the number of rounds~~

Our approach: A new aggregation paradigm



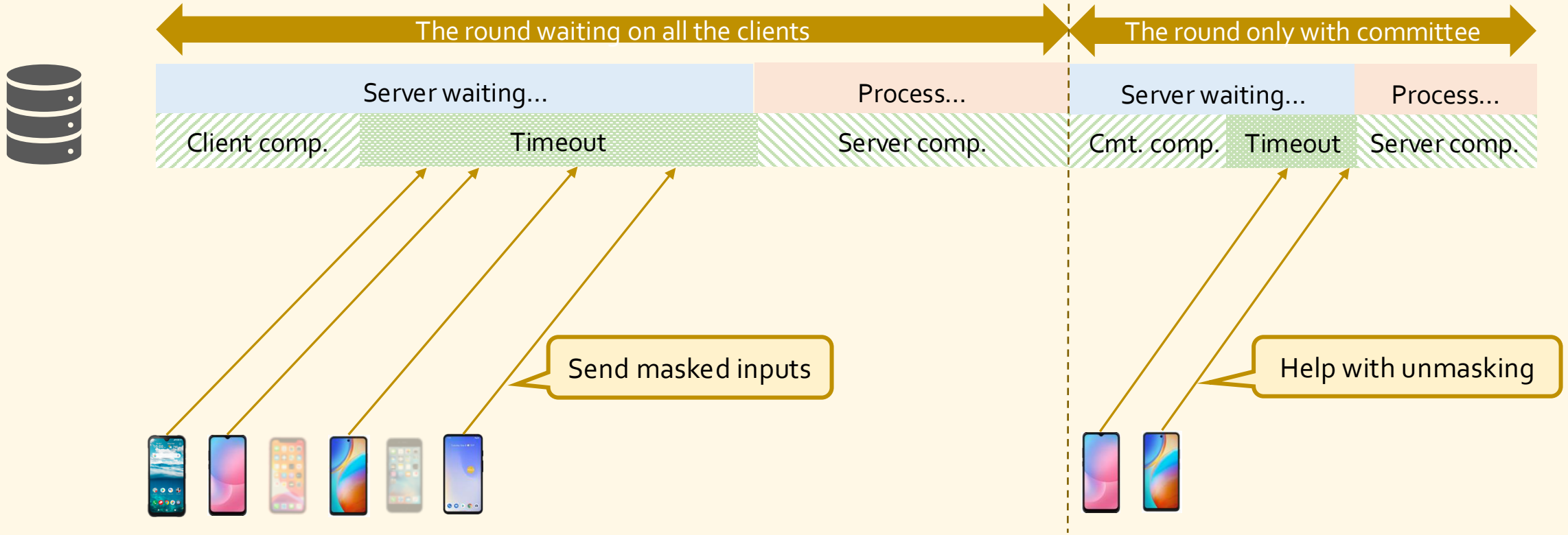
Our approach: A new aggregation paradigm



Our approach: A new aggregation paradigm

Decreasing the timeout?

- Not on the first round
- But on the rest of the rounds



Flamingo 2023

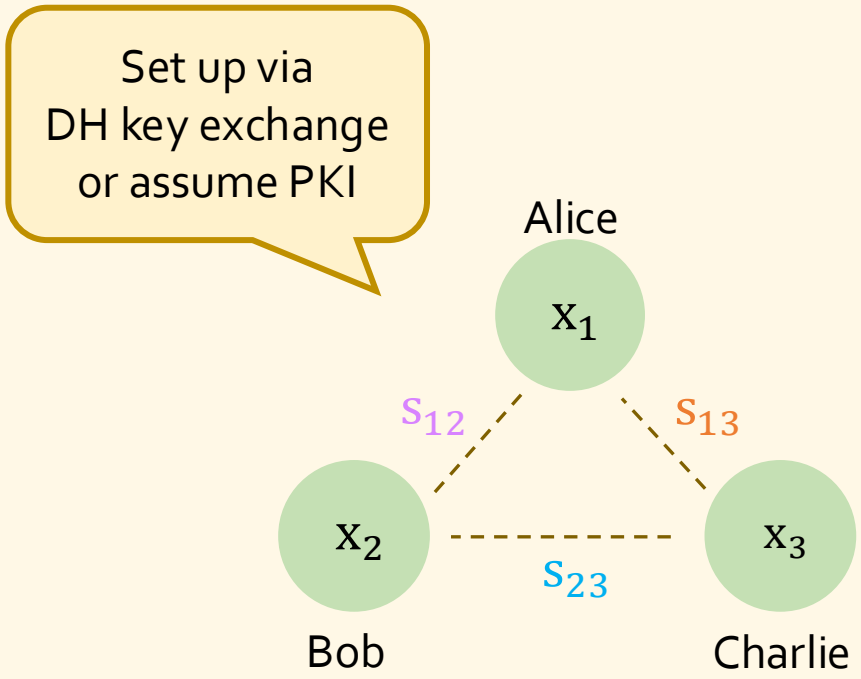
Adversary	Handle failures?	Regular client work	Committee work	Rounds
Semi-honest	✓	$O(\ell + \text{polylog } n)$	$O(n)$	1 + 1
Malicious	✓	$O(\ell + \text{polylog } n)$	$O(n)$	1 + 2

Parameters: input length ℓ, number of clients n.

Flamingo 2023: Yiping Ma, Jess Woods, Sebastian Angel, Antigoni Polychroniadou, and Tal Rabin

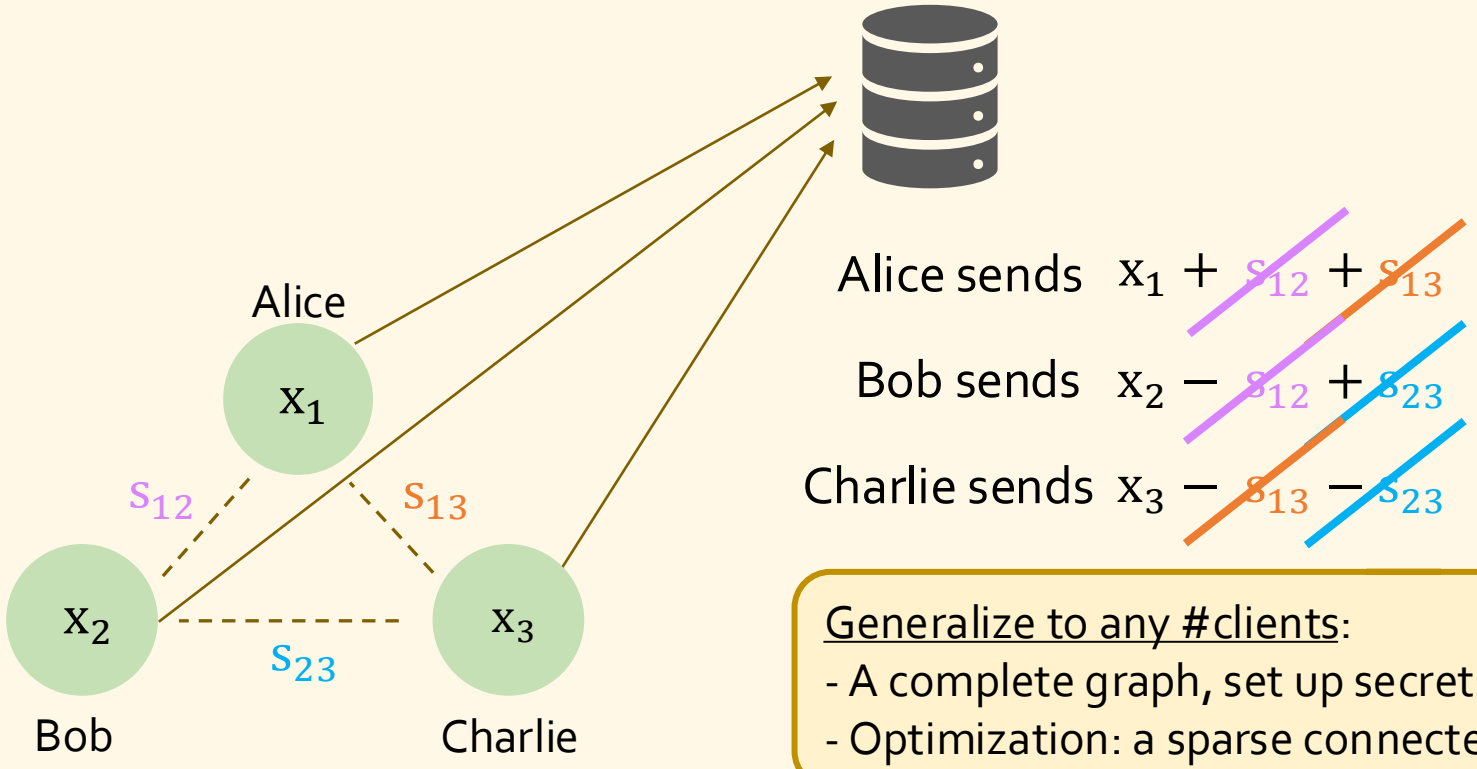
Starting point: pairwise masking [Chaum88]

Step 1. Set up secrets



Starting point: pairwise masking [Chaum88]

Step 2. Send masked inputs



Alice sends $x_1 + s_{12} + s_{13}$

Bob sends $x_2 - s_{12} + s_{23}$

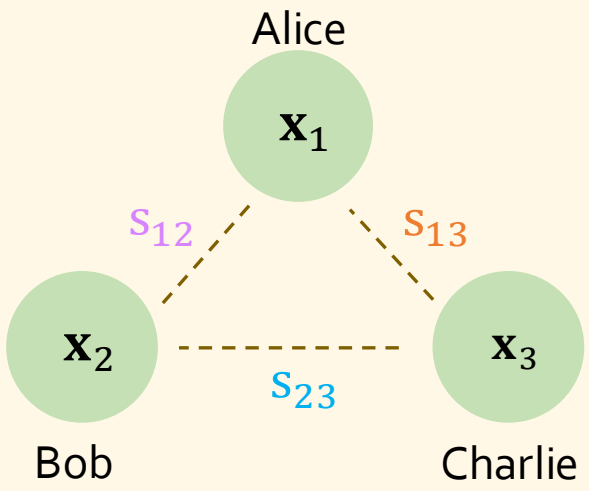
Charlie sends $x_3 - s_{13} - s_{23}$

Generalize to any #clients:
- A complete graph, set up secrets on edges
- Optimization: a sparse connected graph [BBGLR20]

Flamingo 2023: Yiping Ma, Jess Woods, Sebastian Angel, Antigoni Polychroniadou, and Tal Rabin

Starting point: pairwise masking [Chaum88]

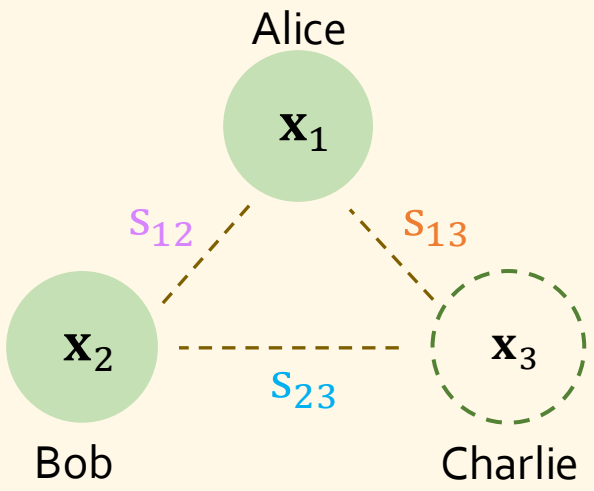
Step 2. Send masked inputs



$$\mathbf{y}_1 = \mathbf{x}_1 + \mathrm{PRG}(s_{12}) + \mathrm{PRG}(s_{13})$$

Problem: Pairwise masking cannot handle failures

Step 2. Send masked inputs



Each input is still hidden

Reveal the secrets?

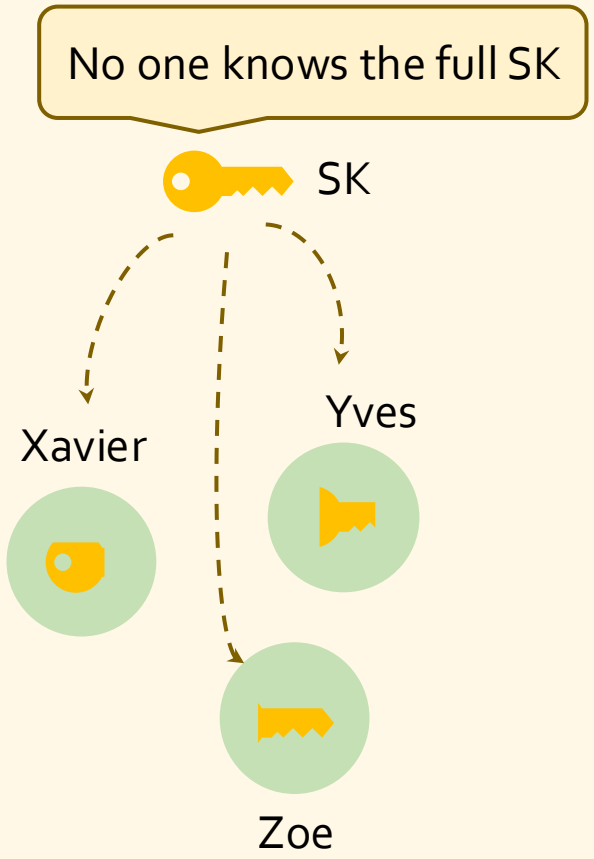
$$\mathbf{y}_1 = \mathbf{x}_1 + \mathrm{PRG}(s_{12}) + \mathrm{PRG}(s_{13})$$

$$\mathbf{y}_2 = \mathbf{x}_2 - \mathrm{PRG}(s_{12}) + \mathrm{PRG}(s_{23})$$

Flamingo 2023: Yiping Ma, Jess Woods, Sebastian Angel, Antigoni Polychroniadou, and Tal Rabin

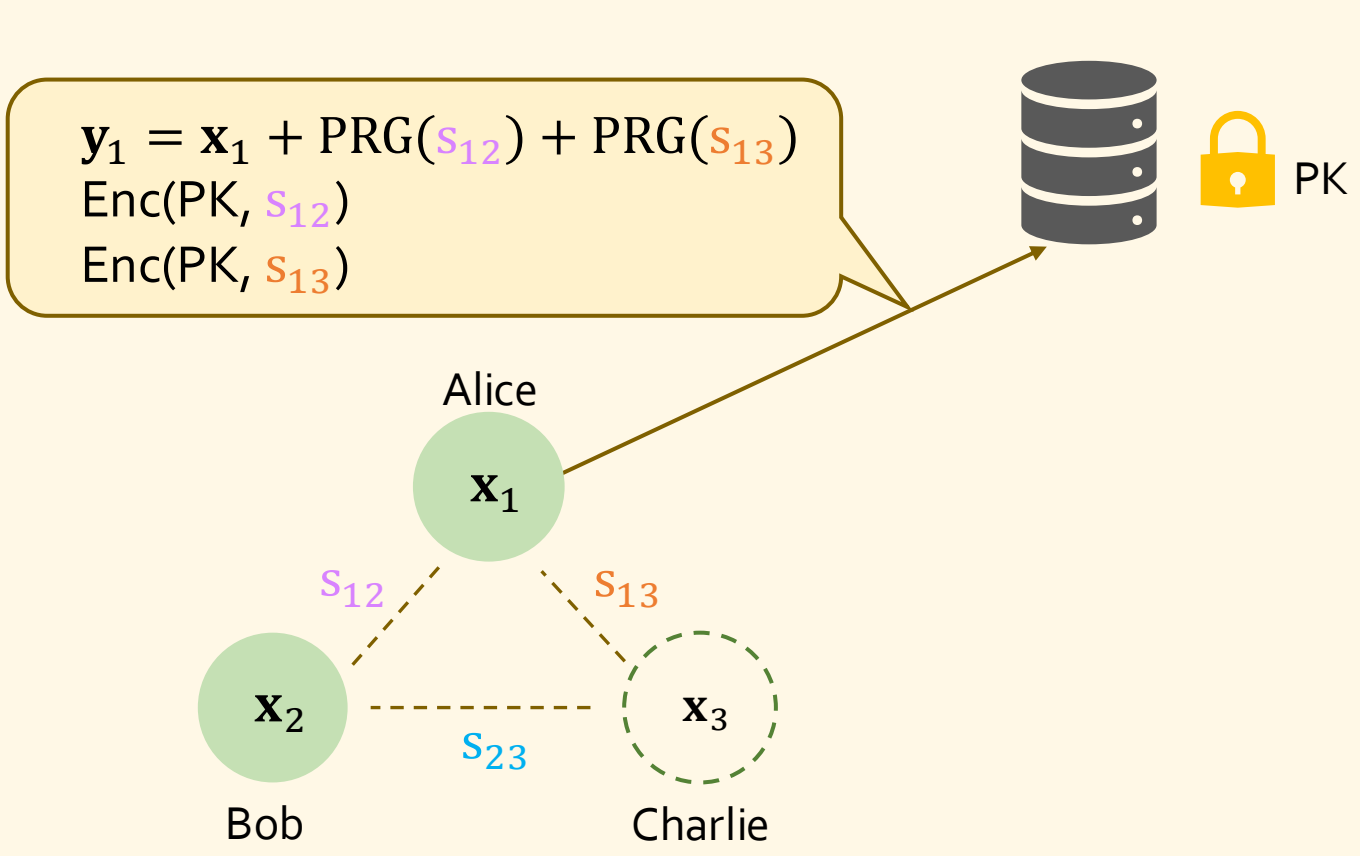
Idea: Reveal offline-online pair via the committee

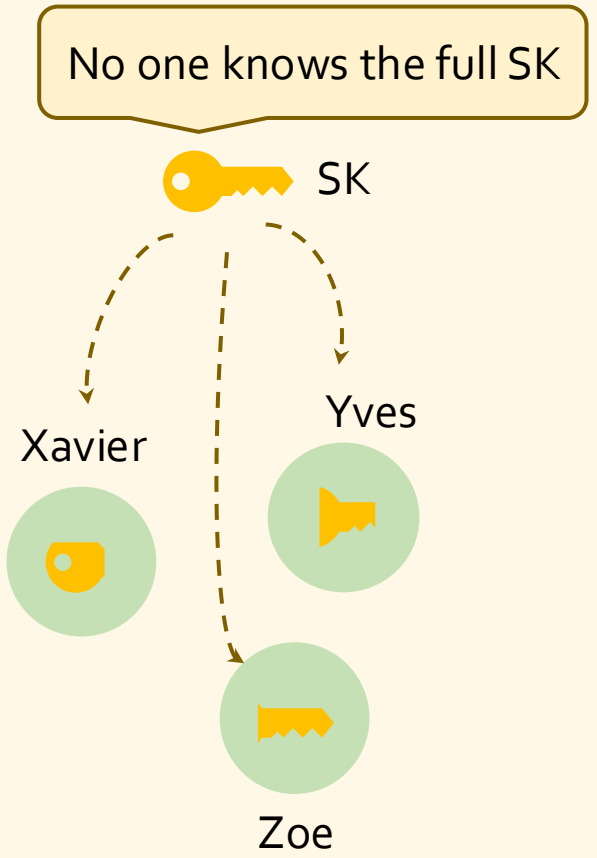
Tool: threshold decryption



Flamingo 2023: Yiping Ma, Jess Woods, Sebastian Angel, Antigoni Polychroniadou, and Tal Rabin

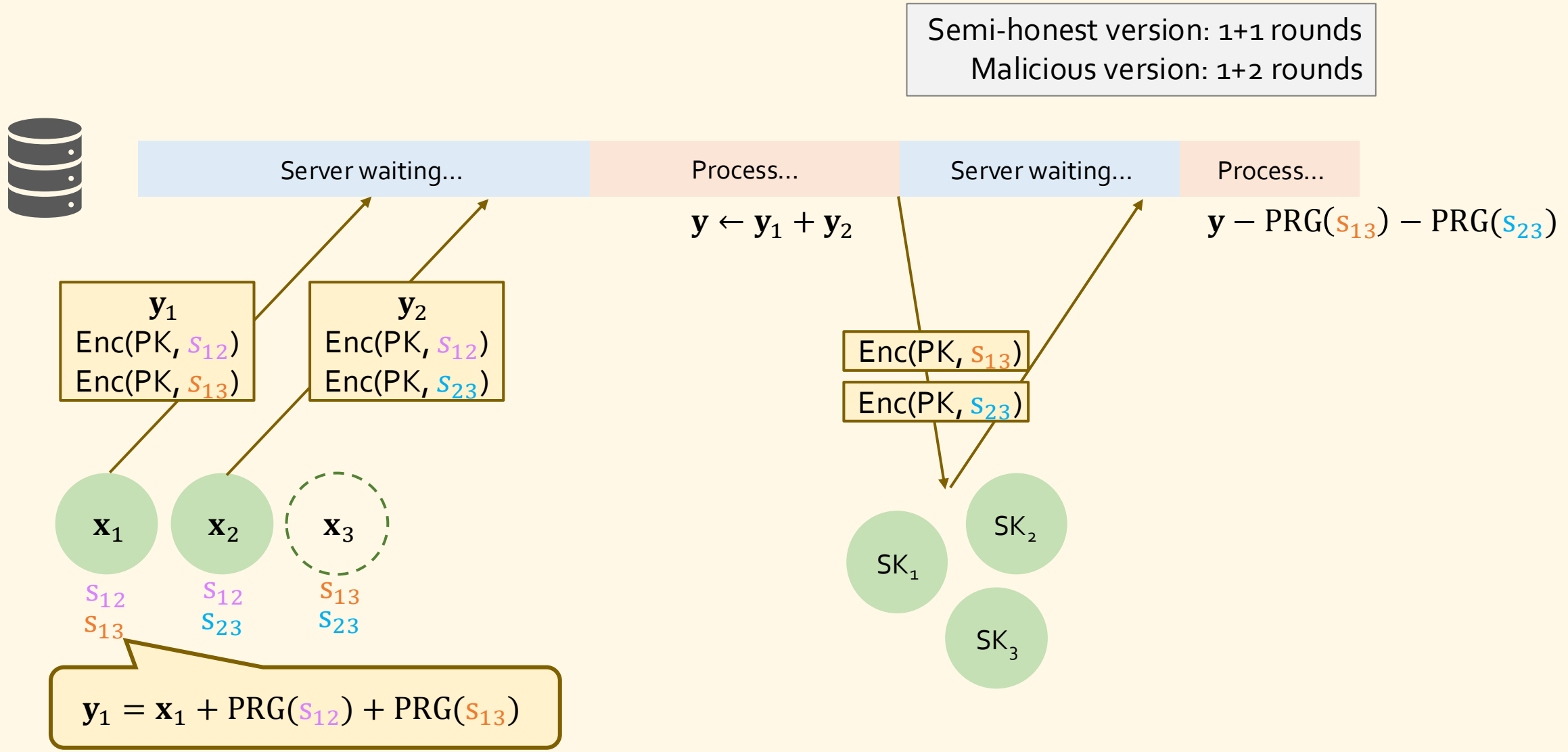
Idea: Reveal offline-online pair via the committee





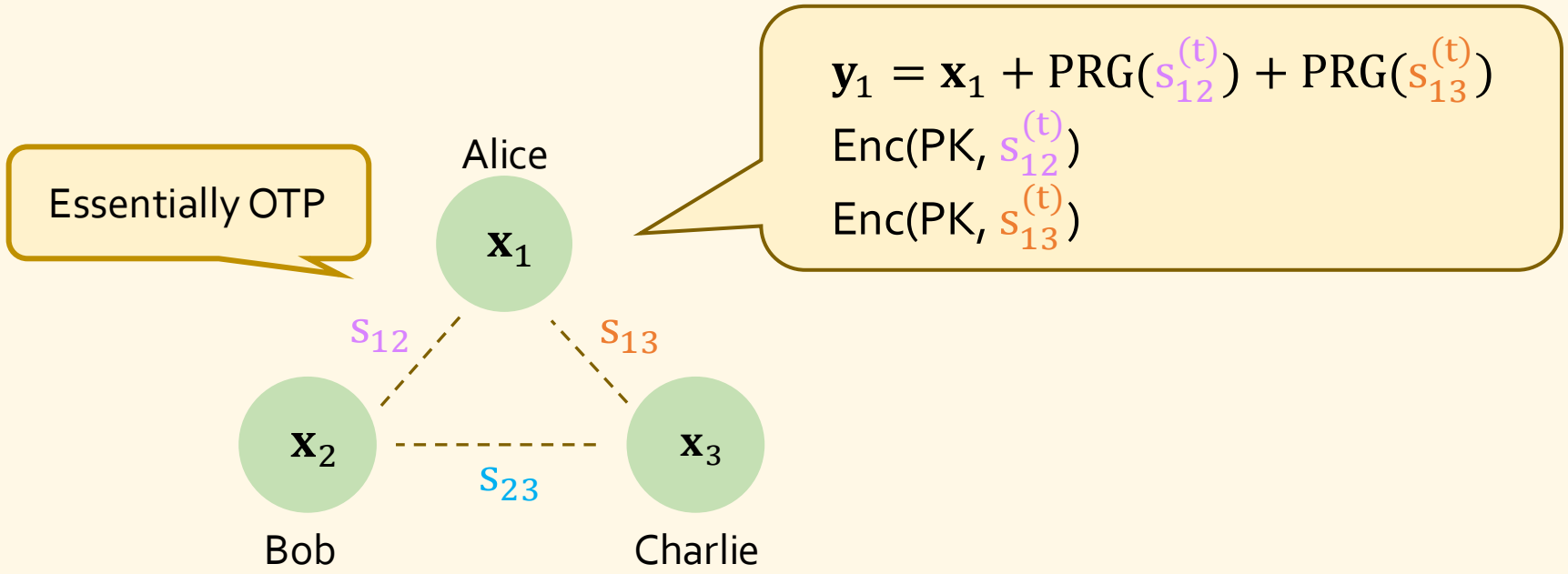
Flamingo 2023: Yiping Ma, Jess Woods, Sebastian Angel, Antigoni Polychroniadou, and Tal Rabin

Flamingo 2023: Workflow



Flamingo 2023: Yiping Ma, Jess Woods, Sebastian Angel, Antigoni Polychroniadou, and Tal Rabin

Reusing secrets from setup



Iteration t: $s_{12}^{(t)} = \mathrm{PRF}(s_{12}, t)$

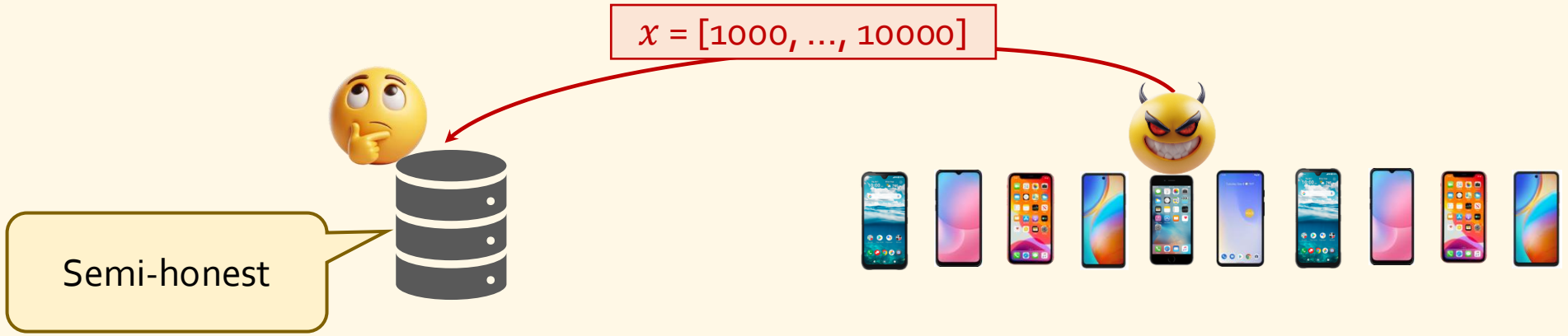
Flamingo 2023: Yiping Ma, Jess Woods, Sebastian Angel, Antigoni Polychroniadou, and Tal Rabin

Performance: Comparison for wait time

#Clients/Iteration	Flamingo Wait	BBGLR Wait	Wait Speedup
128	5.77s	43.66s	**7.6**×
256	6.11s	41.39s	**6.8**×
512	5.84s	48.10s	**8.2**×

Flamingo 2023: Yiping Ma, Jess Woods, Sebastian Angel, Antigoni Polychroniadou, and Tal Rabin

Armadillo 2025



Robustness: even when some clients misbehave, the server can always learn $\sum \mathbf{x}_i$ for those valid $\mathbf{x}_i$'s.

This work: L_2, L_∞ norms

A generic recipe: Proof of correct execution

- Protocol execution in round 1:
  - Client local computation
  - Send message to server
- Computation: A sequence of + and × on private and public inputs
- Throw in a zero-knowledge proof (ZKP) for that sequence
- ZKP is expensive, so we want the sequence to be "simple"

Armadillo 2025: Yiping Ma, Yue Guo, Harish Karthikeyan, Antigoni Polychroniadou

Our approach: Co-design aggregation and proof

- A re-design of aggregation: very simple arithmetic computation
- Each client can write proof statement as an inner-product relation

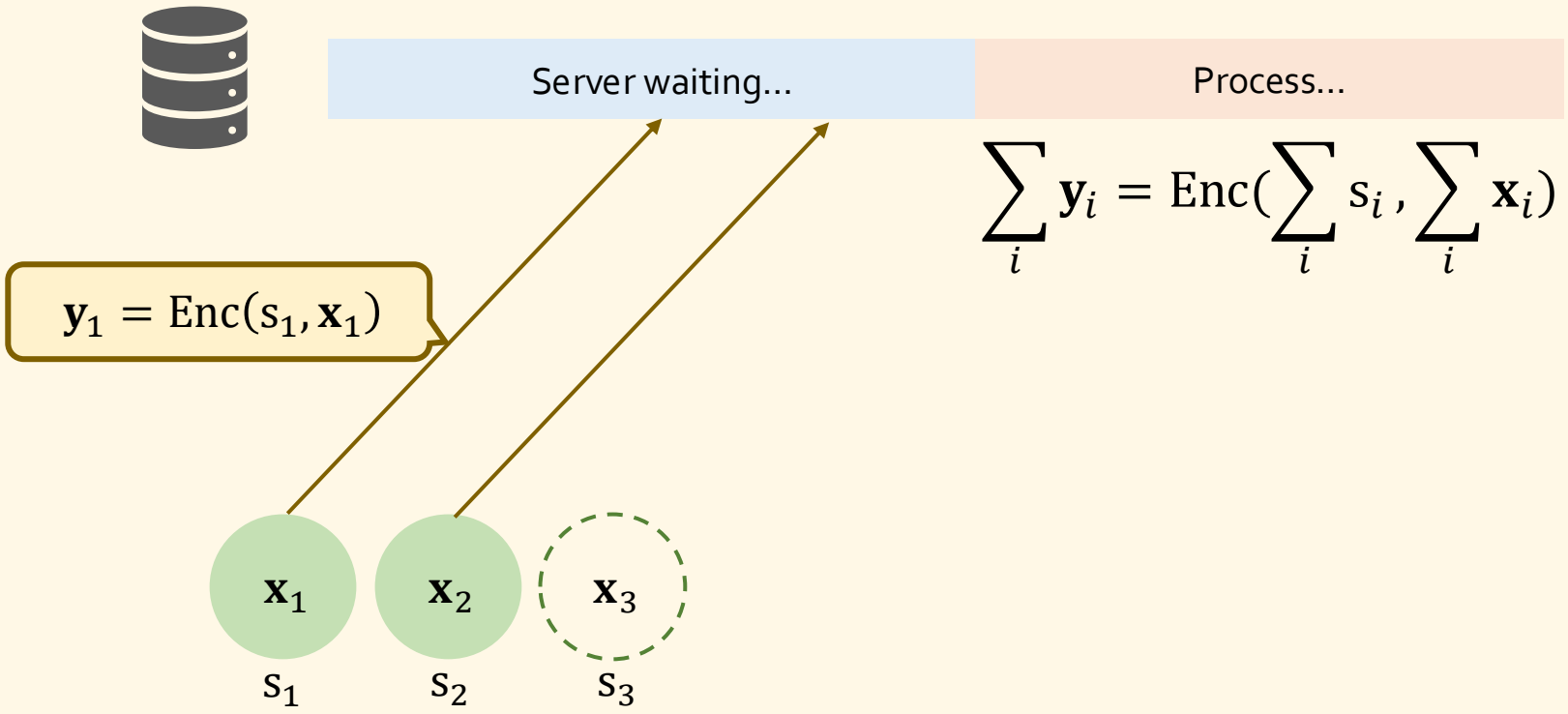
Our approach: Co-design aggregation and proof

- A re-design of aggregation: very simple arithmetic computation
- Tool: Key-and-message-homomorphic encryption

$$\mathrm{Enc}(\mathbf{s}_1, \mathbf{x}_1) + \mathrm{Enc}(\mathbf{s}_2, \mathbf{x}_2) = \mathrm{Enc}(\mathbf{s}_1 + \mathbf{s}_2, \mathbf{x}_1 + \mathbf{x}_2)$$

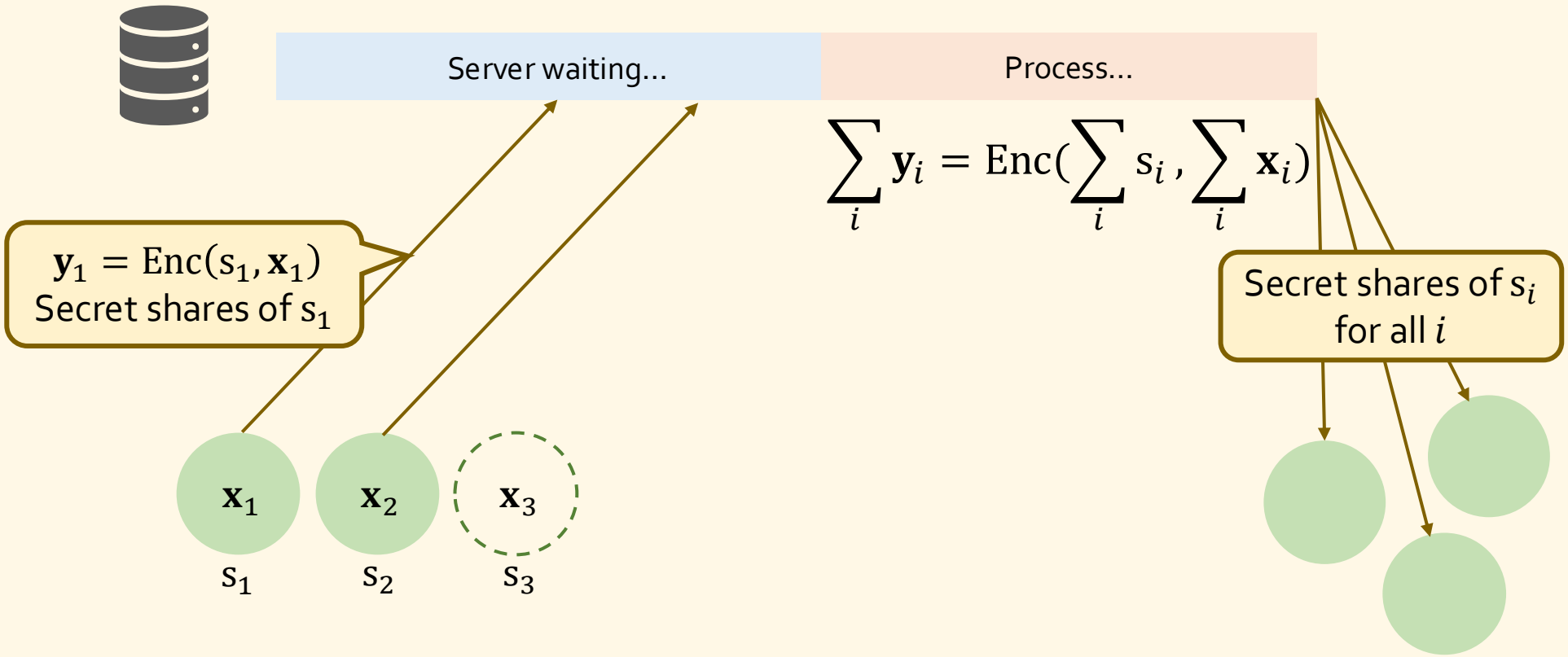
Armadillo 2025: Yiping Ma, Yue Guo, Harish Karthikeyan, Antigoni Polychroniadou

Our protocol: Two-layer aggregation



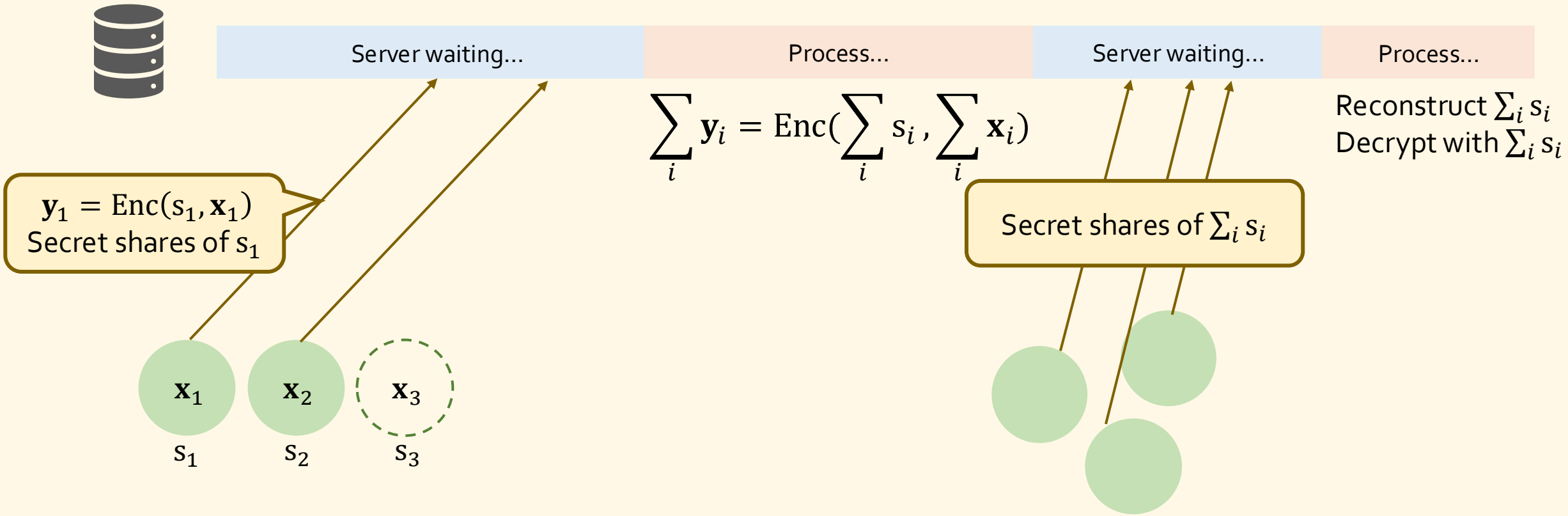
Armadillo 2025: Yiping Ma, Yue Guo, Harish Karthikeyan, Antigoni Polychroniadou

Our protocol: Two-layer aggregation



Armadillo 2025: Yiping Ma, Yue Guo, Harish Karthikeyan, Antigoni Polychroniadou

Our protocol: Two-layer aggregation



Armadillo 2025: Yiping Ma, Yue Guo, Harish Karthikeyan, Antigoni Polychroniadou

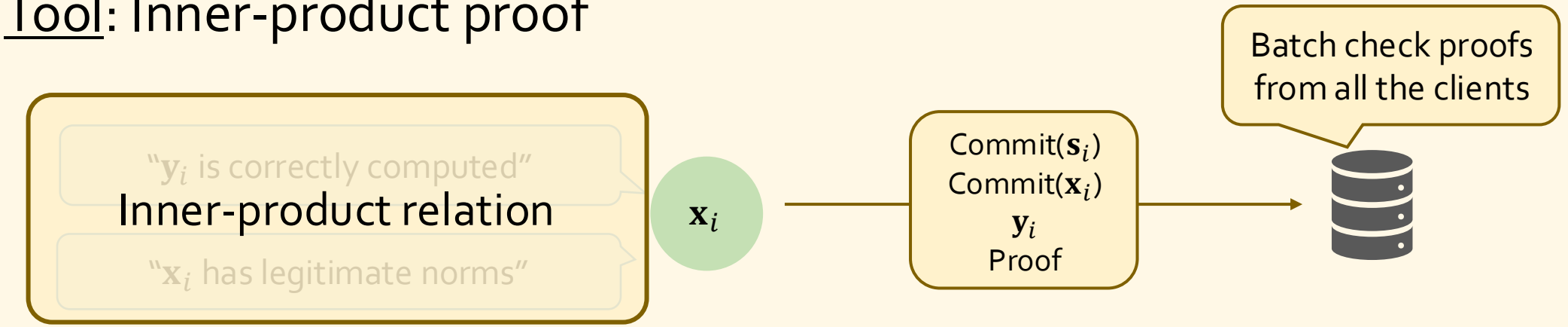
Our approach: Co-design aggregation and proof

- A re-design of aggregation: very simple arithmetic computation
- Each client can write proof statement as an inner-product relation
- Tool: Inner-product proof

"Given public value c, Commit(**a**) and Commit(**b**), prove that <**a**, **b**>=c"

Our approach: Co-design aggregation and proof

- A re-design of aggregation: very simple arithmetic computation
- Each client can write proof statement as an inner-product relation
- Tool: Inner-product proof



Armadillo 2025: Yiping Ma, Yue Guo, Harish Karthikeyan, Antigoni Polychroniadou

The progress made so far

The progress made so far

	Dropouts	Robustness	Client comp.		Client comm.		Rounds
Naïve MPC	✓	X	$O(\ell \cdot n)$		$O(\ell \cdot n)$		2
Flamingo 2023	✓	X	$O(\ell + \text{polylog } n)$	$O(n)$	$O(\ell + \text{polylog } n)$	$O(n)$	1 + 2
Armadillo 2025	✓	✓	Same asymptotics as Flamingo				

And many more under the committee model:
Lerna, OPA, Willow, WillowFold, …

The progress made so far

	Dropouts	Robustness	Client comp.		Client comm.		Rounds
Naïve MPC	✓	X	$O(\ell \cdot n)$		$O(\ell \cdot n)$		2
E							
							$g\ n)$
Flamingo 2023	✓	X	$O(\ell + \text{polylog}\ n)$	$O(n)$	$O(\ell + \text{polylog}\ n)$	$O(n)$	1 + 2
Armadil		✓	Same asymptotics as Flamingo				
Lighthouse 2026	✓	X	$O(\ell + \text{polylog}\ n)$	$O(1)$	$O(\ell + \text{polylog}\ n)$	$O(1)$	1 + 2

A new result in batch threshold decryption

And many more under the committee model:
Lerna, OPA, Willow, WillowFold, …

Open questions

- Research direction 1: How small can the committee cost be?
- Research direction 2: What if the adversary is adaptive?
- Research direction 3: Verification for inputs, how "sufficient" can norms be?
- Research direction 4: Richer functionality class
  - GMPC by Bar Alon, Moni Naor, Eran Omri, Uri Stemmer

Thank you!