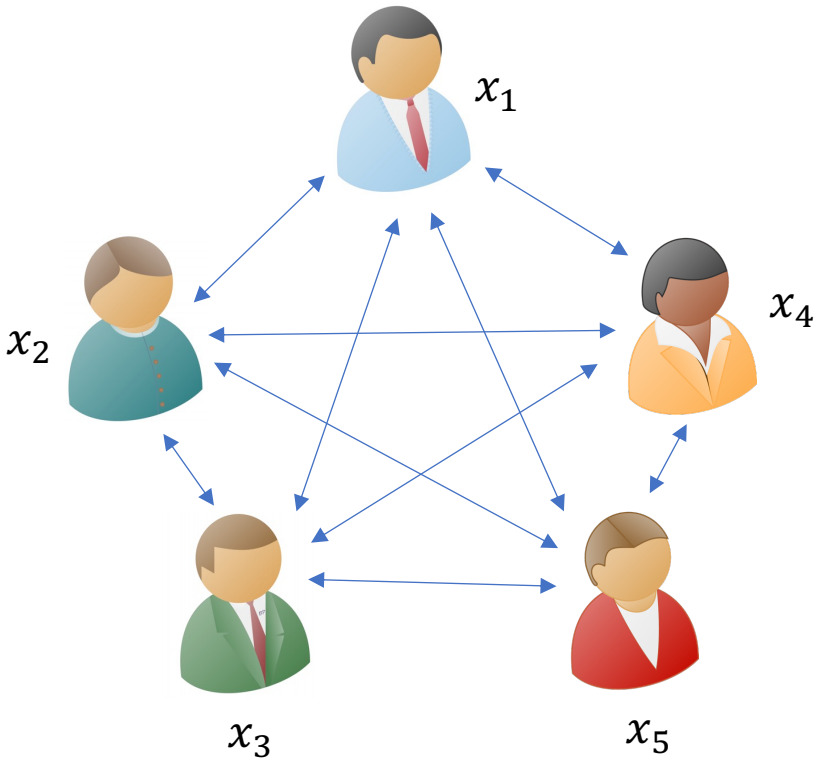
FLUID MPC: Secure Multiparty Computation with Dynamic Participants

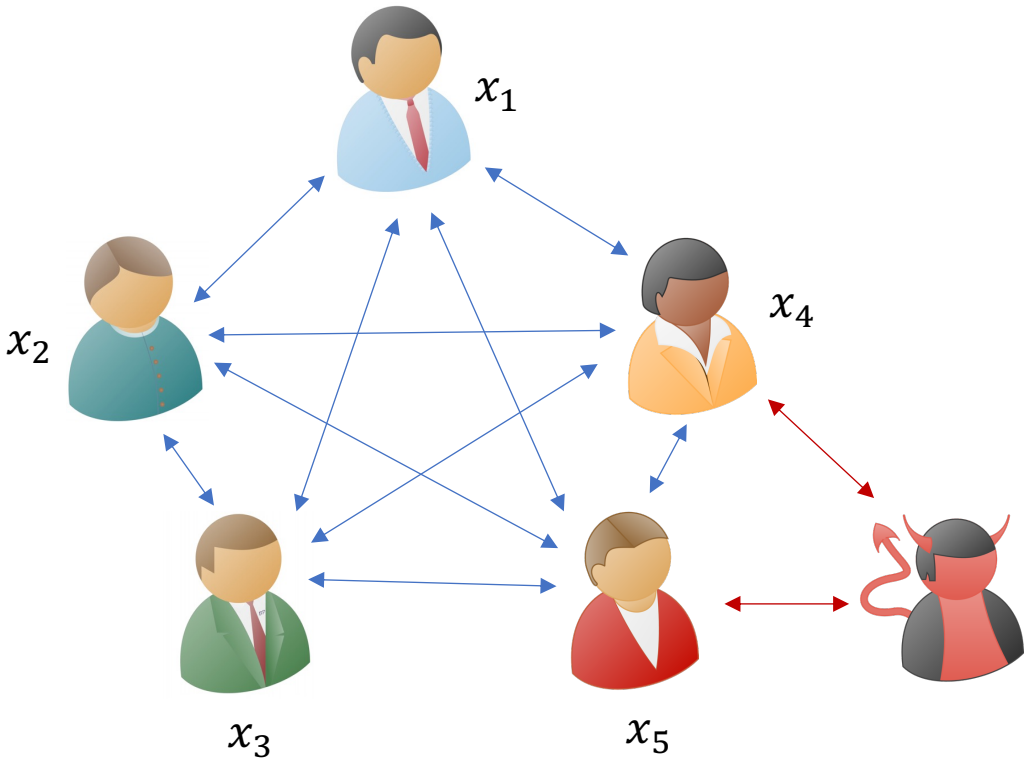
Aarushi Goel

NTT Research

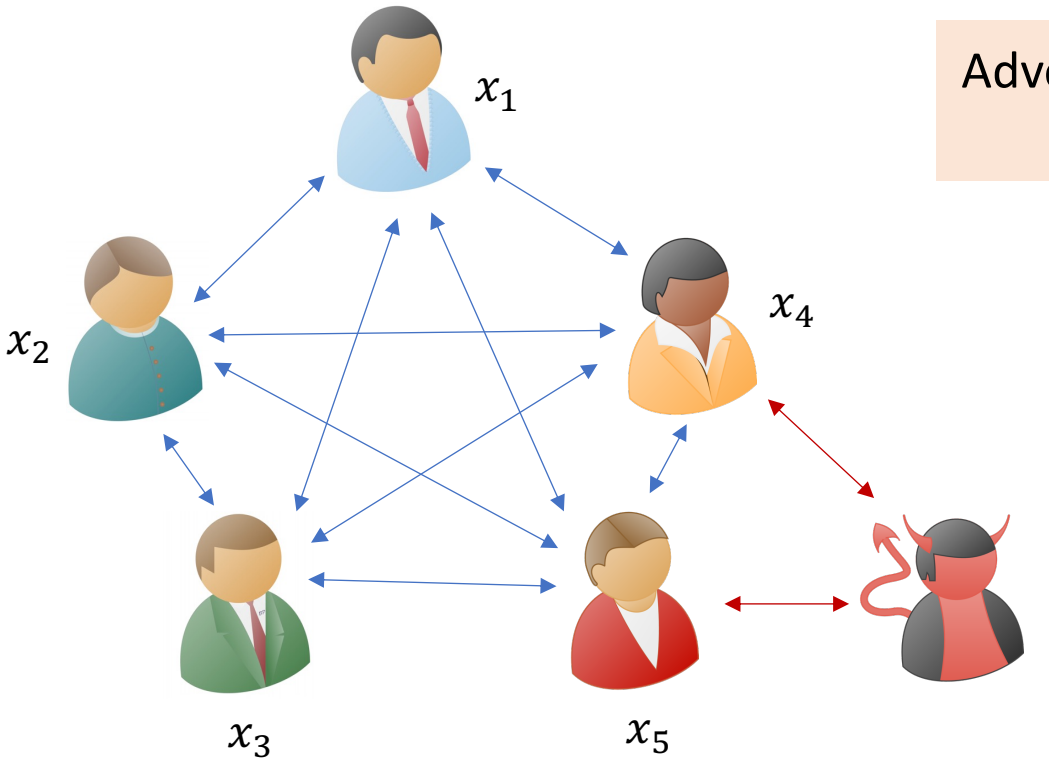
Secure Multiparty Computation



Secure Multiparty Computation



Secure Multiparty Computation



Adversary learns nothing beyond the output of the function, i.e.,
$y = f(x_1, x_2, x_3, x_4, x_5)$

MPC and Emerging Applications

- MPC protocols are becoming increasingly efficient.

MPC and Emerging Applications

- MPC protocols are becoming increasingly efficient.
- Can be used to compute large complex functionalities such as:

MPC and Emerging Applications

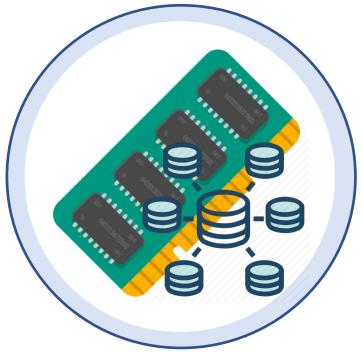
- MPC protocols are becoming increasingly efficient.
- Can be used to compute large complex functionalities such as:

Training machine learning algorithms on massive, distributed datasets.

MPC and Emerging Applications

- MPC protocols are becoming increasingly efficient.
- Can be used to compute large complex functionalities such as:

Training machine learning algorithms on massive, distributed datasets.

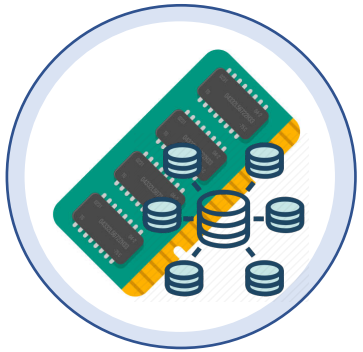


Simulating large RAM programs on distributed datasets

MPC and Emerging Applications

- MPC protocols are becoming increasingly efficient.
- Can be used to compute large complex functionalities such as:

Training machine learning algorithms on massive, distributed datasets.



Simulating large RAM programs on distributed datasets

Issue: Evaluating these functionalities could take up to several hours or even days.

Problem with Static MPC
(Fixed Participants)

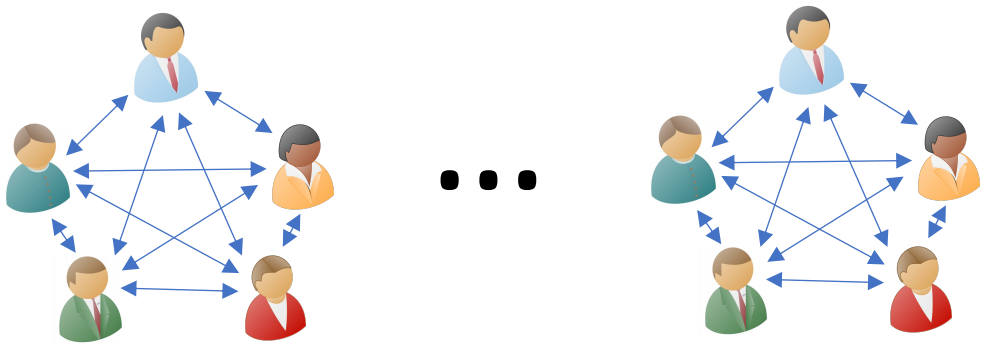
Problem with Static MPC
(Fixed Participants)

Entire Protocol Duration

Problem with Static MPC
(Fixed Participants)

Entire Protocol Duration

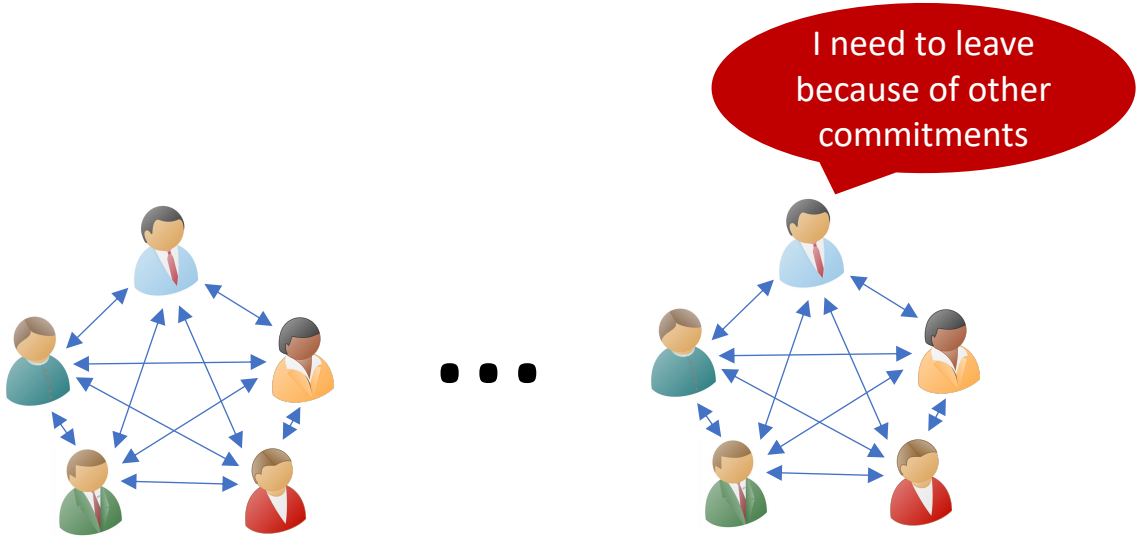
After some time, in the middle of the protocol



Problem with Static MPC (Fixed Participants)

Entire Protocol Duration

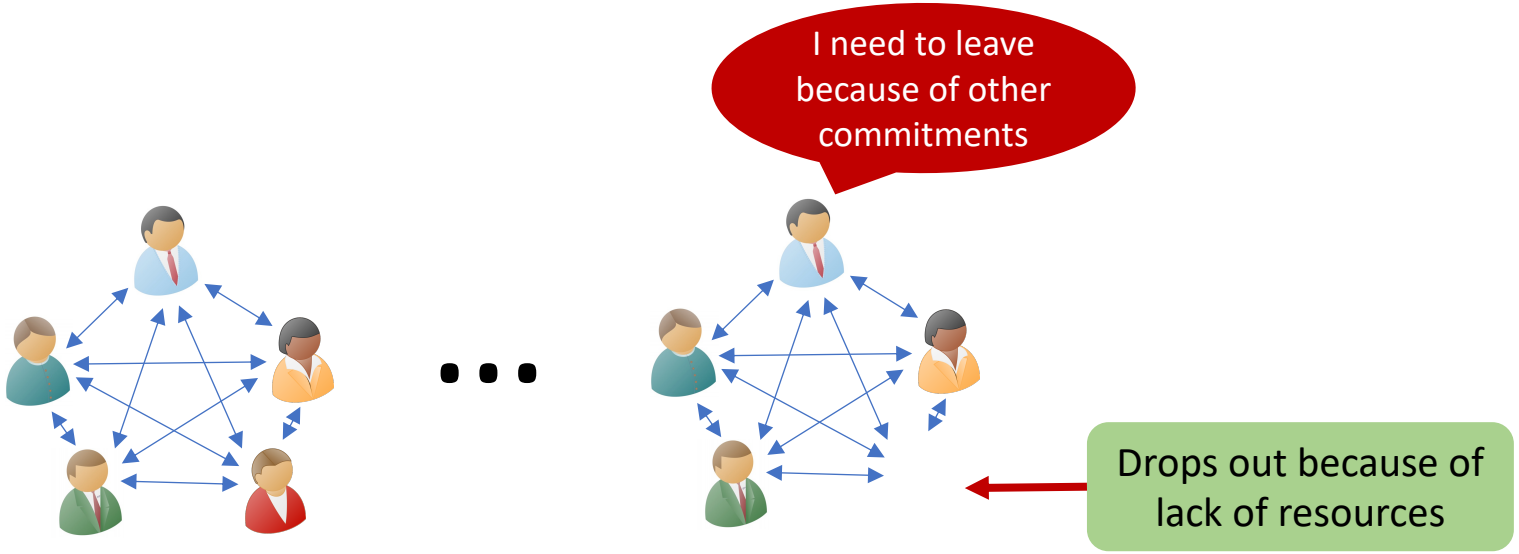
After some time, in the middle of the protocol



Problem with Static MPC (Fixed Participants)

Entire Protocol Duration

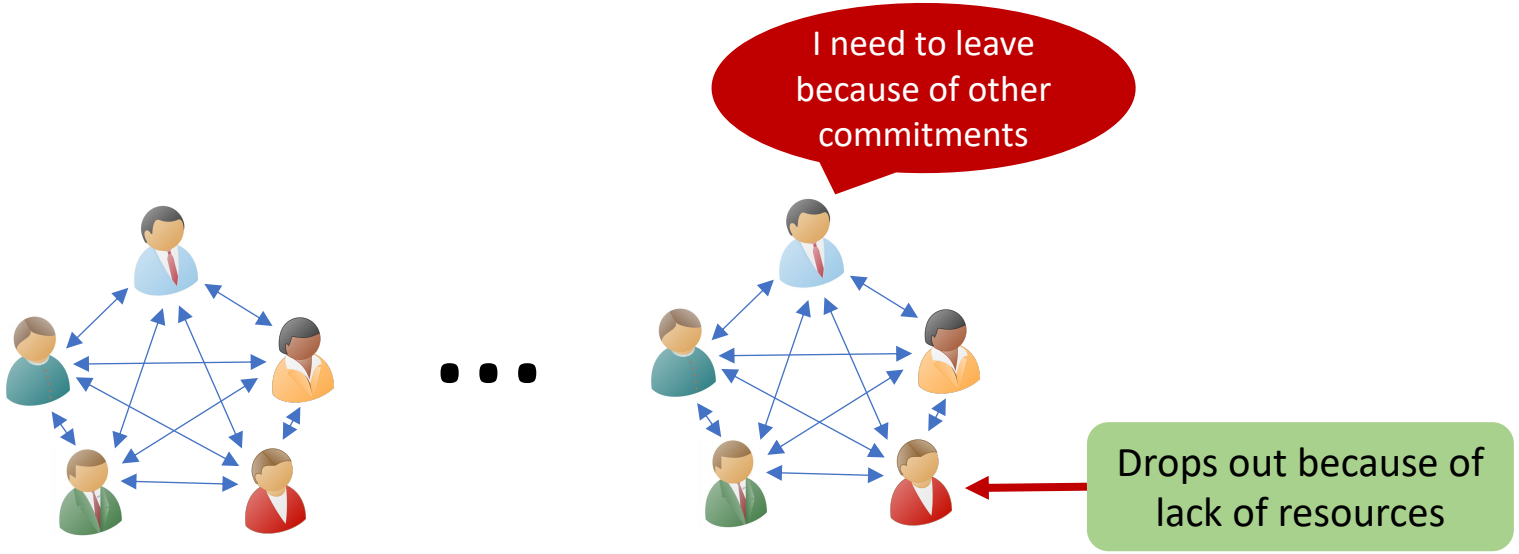
After some time, in the middle of the protocol



Problem with Static MPC
(Fixed Participants)

Entire Protocol Duration

After some time, in the middle of the protocol

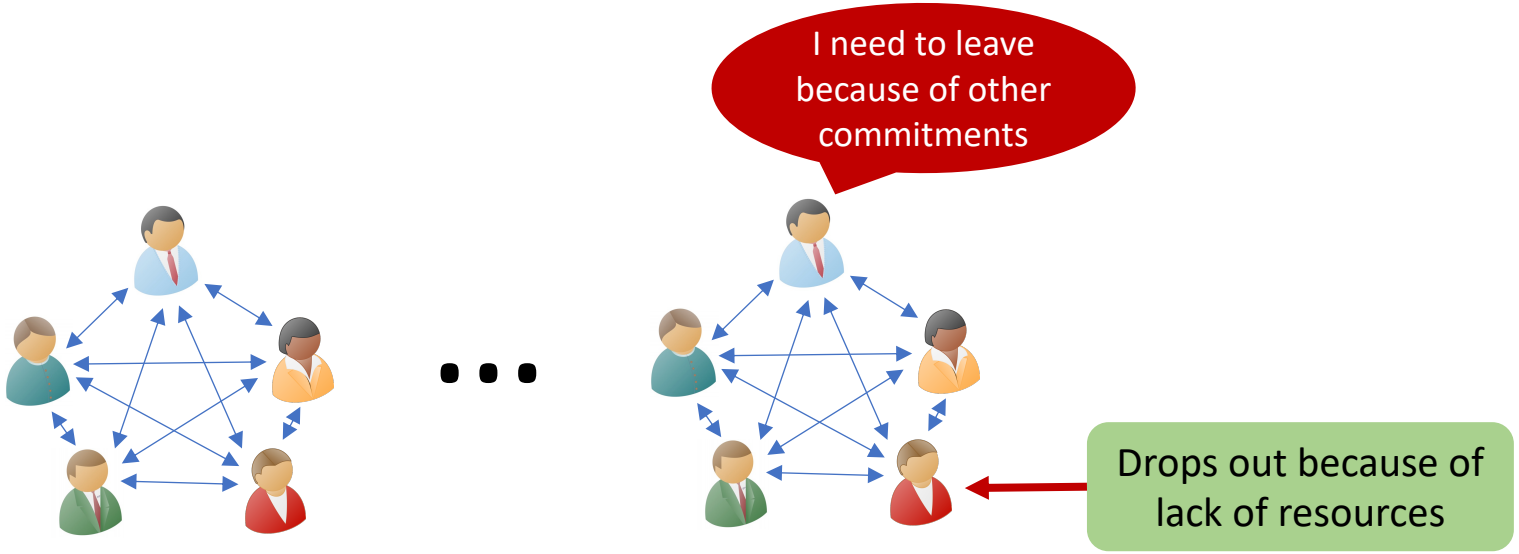


Requiring all participants to stay online throughout the computation is an unrealistic expectation.

Main Question

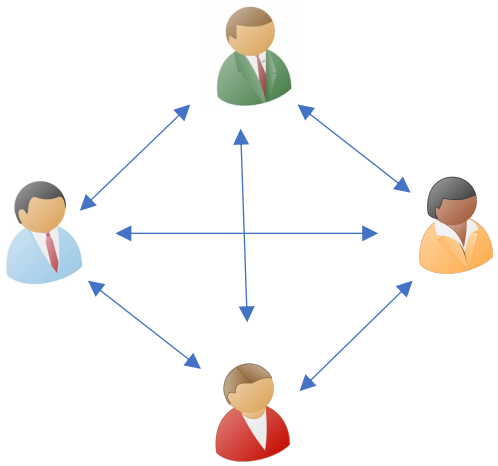
Entire Protocol Duration

After some time, in the middle of the protocol



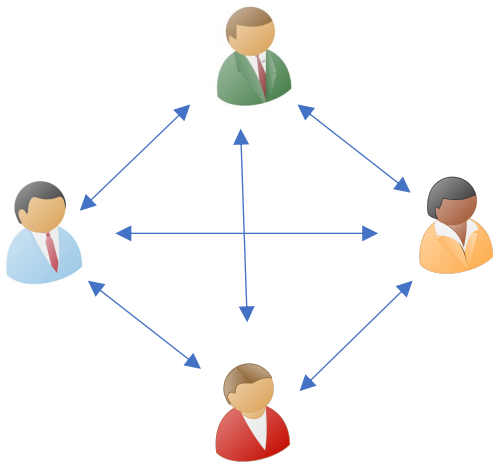
Can we design MPC protocols with Dynamic Participants?

MPC with Dynamic Participants



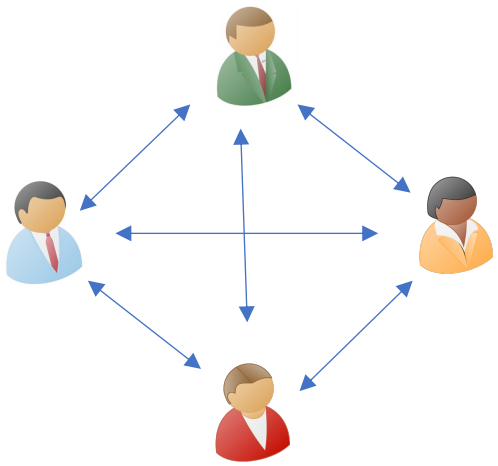
A group of parties start the computation

MPC with Dynamic Participants



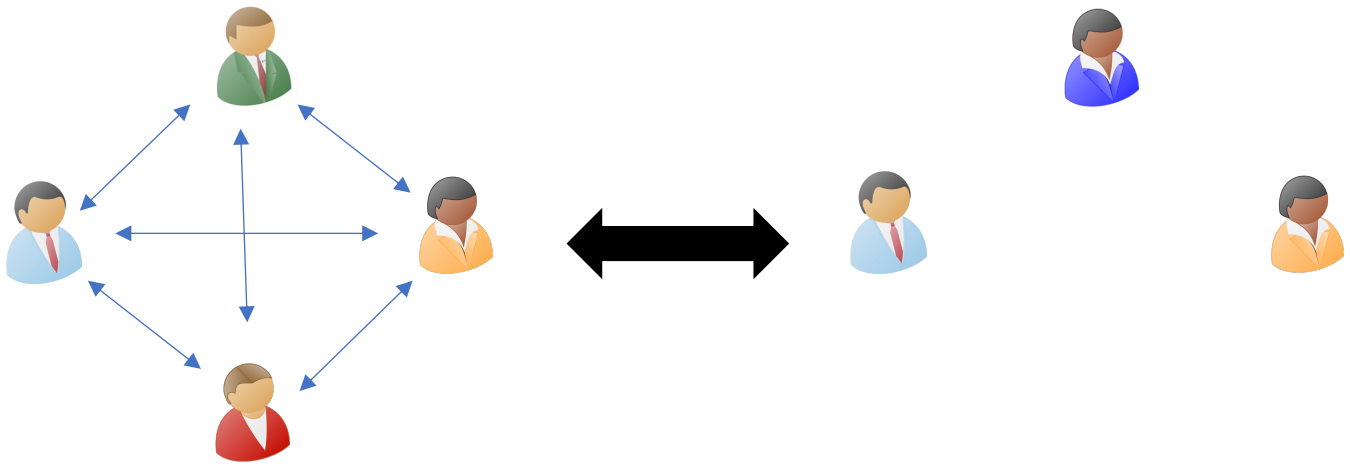
After some time two parties have to leave

MPC with Dynamic Participants



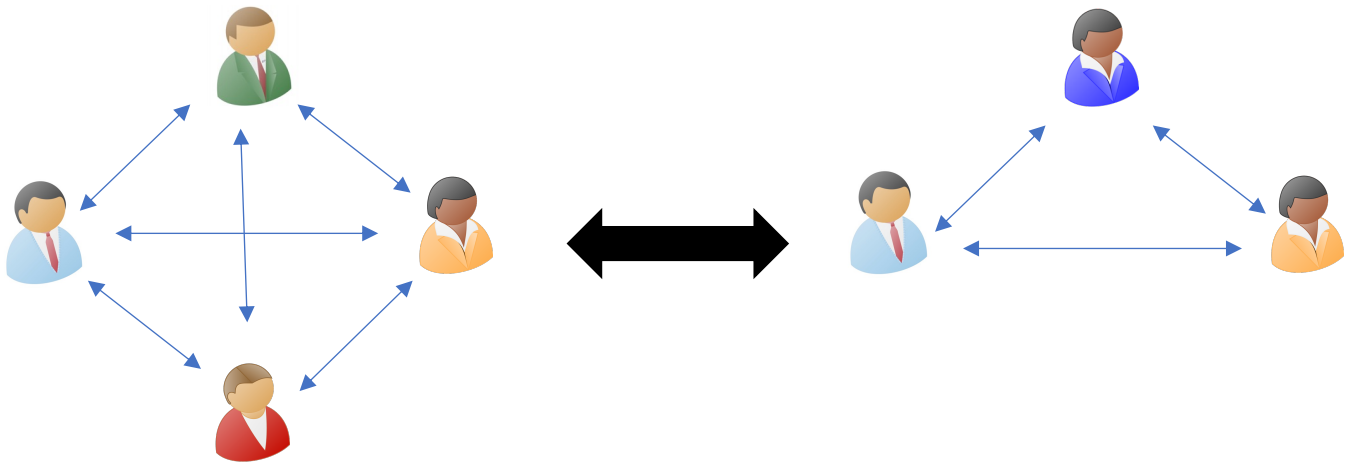
And a new party wants to join the computation

MPC with Dynamic Participants



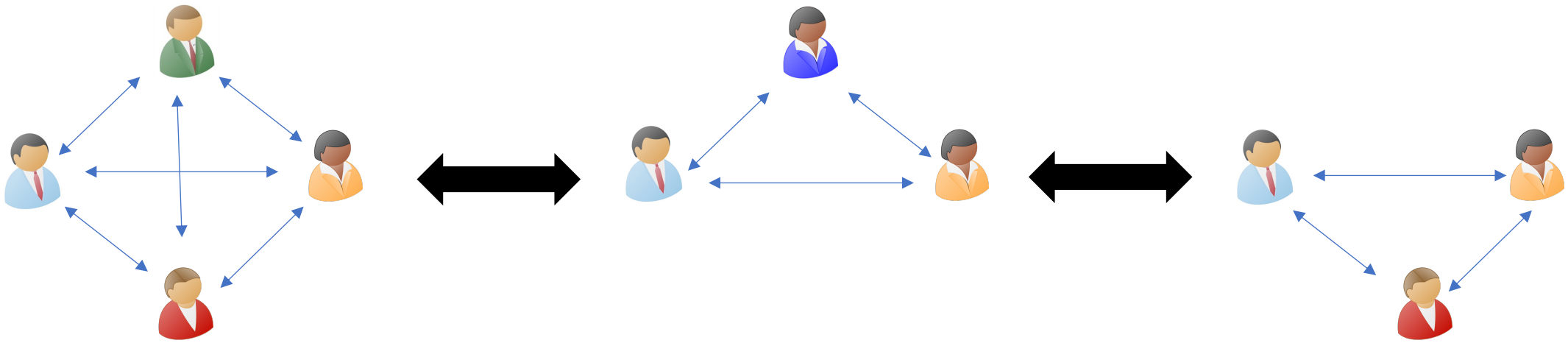
The previous group of parties securely distributes information about the computation so far, to the new group

MPC with Dynamic Participants

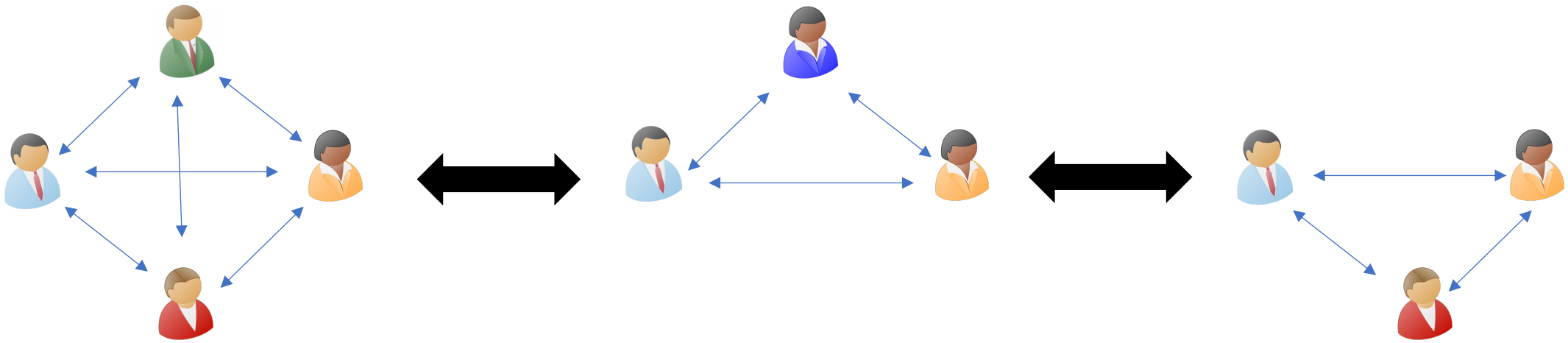


Given this information, the new group
continues with the rest of the computation

MPC with Dynamic Participants

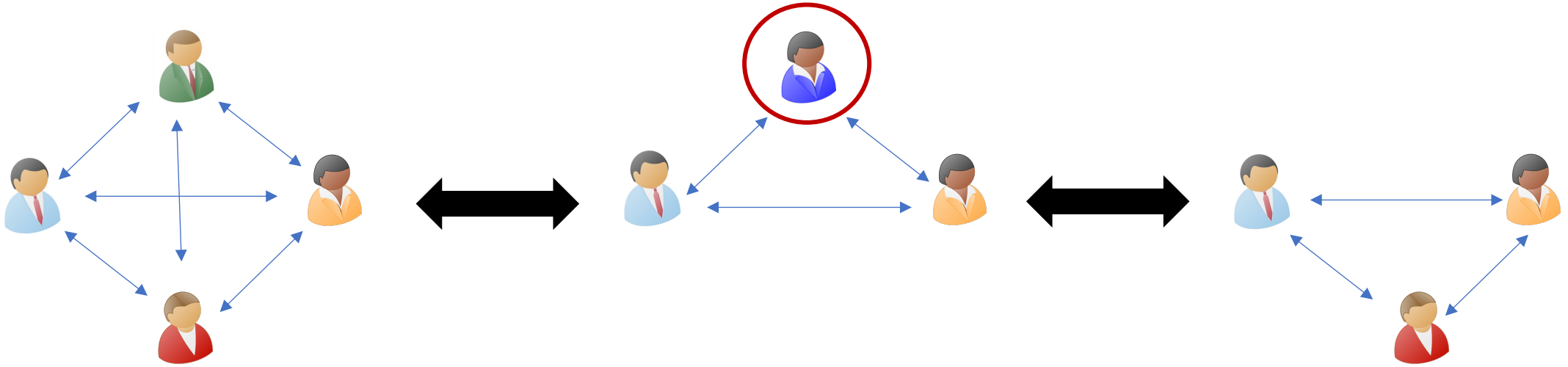


MPC with Dynamic Participants



This reduces the burden of computation on individual parties

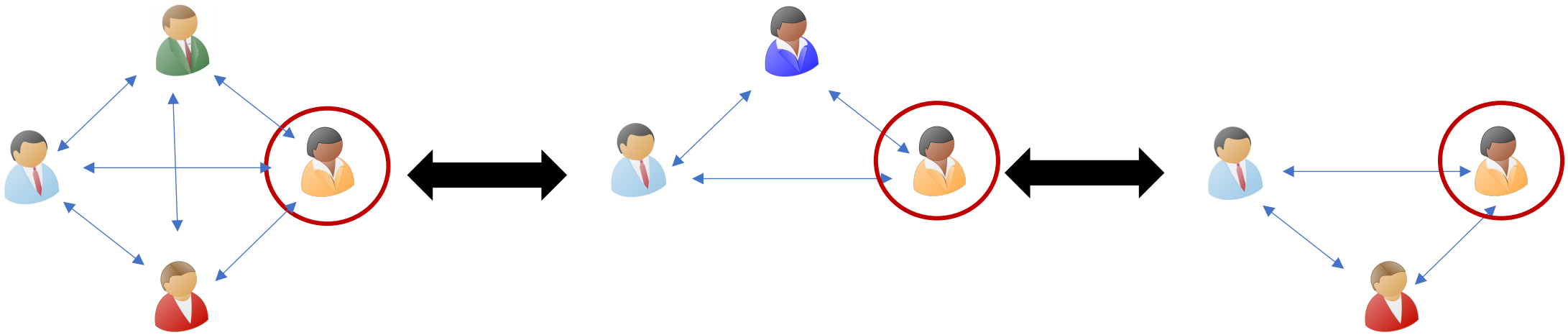
MPC with Dynamic Participants



This reduces the burden of computation on individual parties

Parties with low computational resources can also participate for a small time

MPC with Dynamic Participants

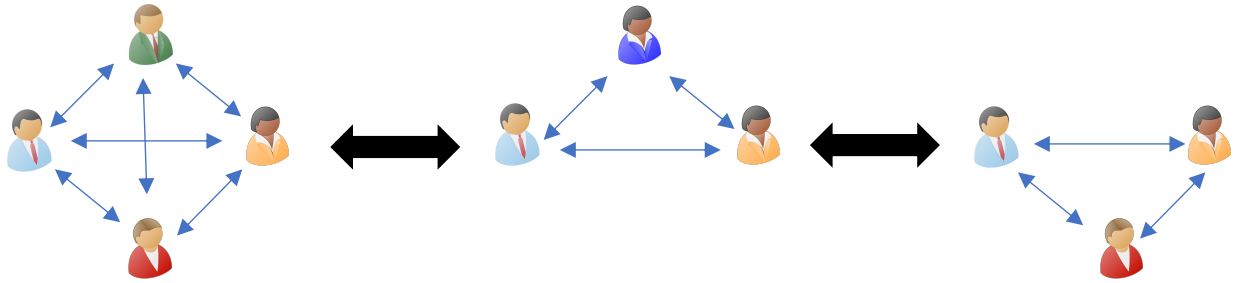


This reduces the burden of computation on individual parties

Parties with low computational resources can also participate for a small time

While parties with more time and computational resources can help with the computation for a longer time

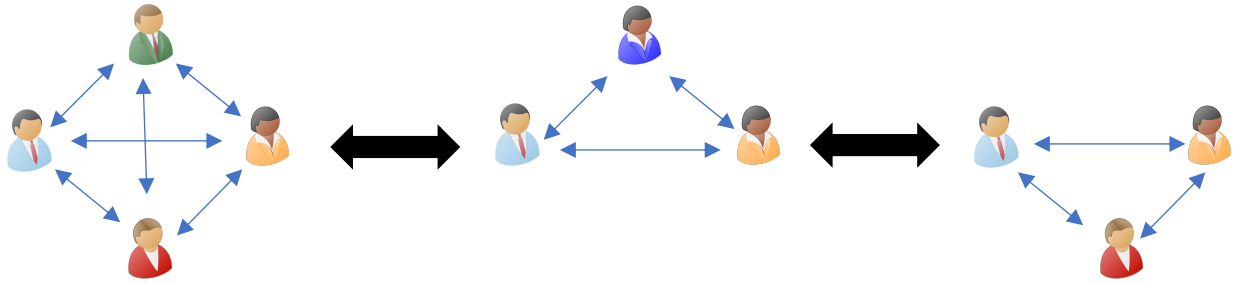
MPC as a Service



MPC with Dynamic Participants

- Allows Participants to join and leave at will
- Reduces burden of computation on individual participants

MPC as a Service



MPC with Dynamic Participants

- Allows Participants to join and leave at will
- Reduces burden of computation on individual participants

Tor

bitcoin

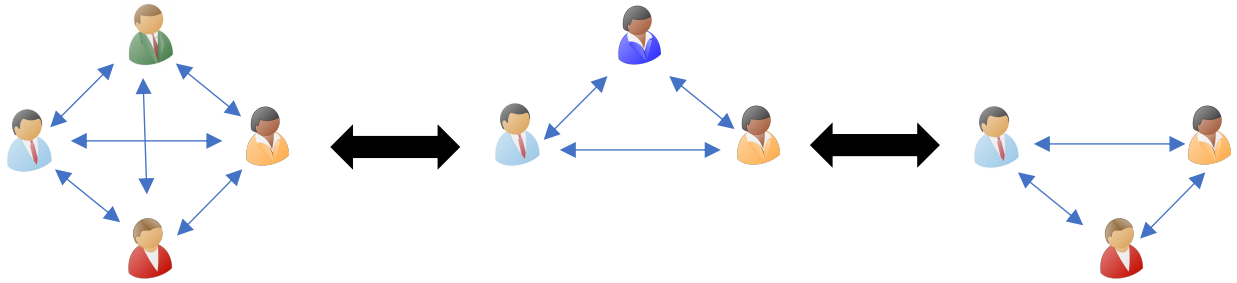
ETHERIUM

Dynamic Peer-to-peer networks.

- Powered by volunteer nodes- that can come and go as they wish.
- Very Successful!

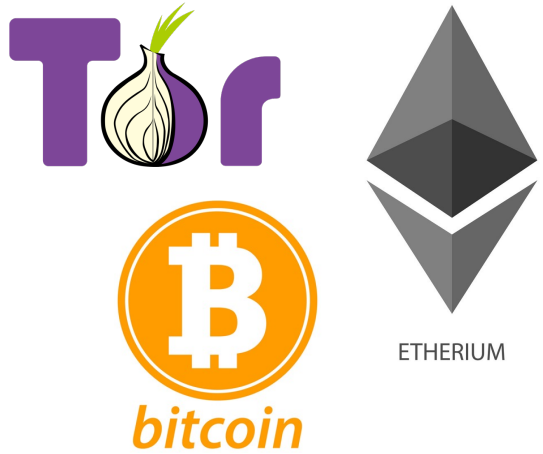
Compatible with each other

MPC as a Service



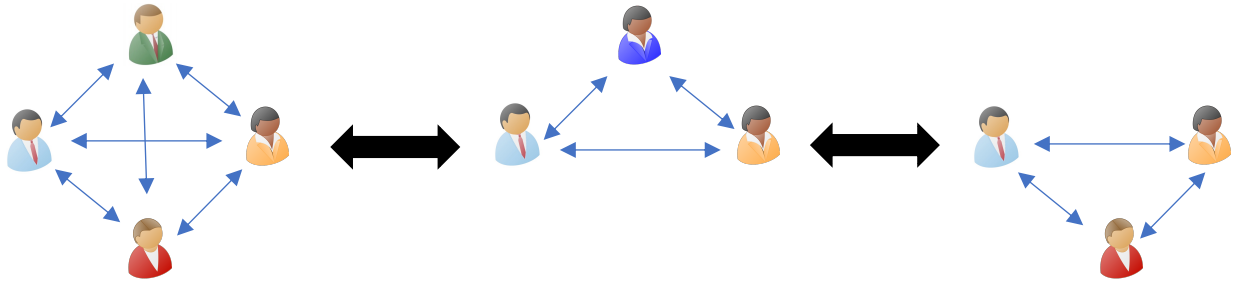
MPC with Dynamic Participants

+



Dynamic Peer-to-peer networks.

MPC as a Service



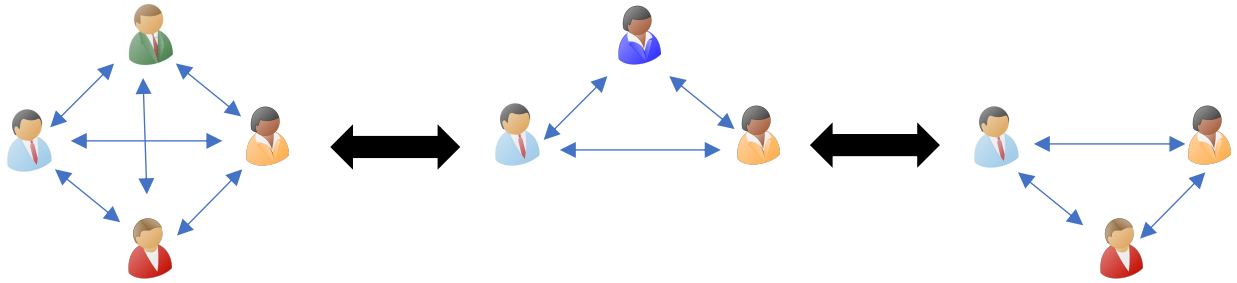
MPC with Dynamic Participants

+

Dynamic Peer-to-peer networks.

Volunteer networks capable of private computation.

MPC as a Service



MPC with Dynamic Participants

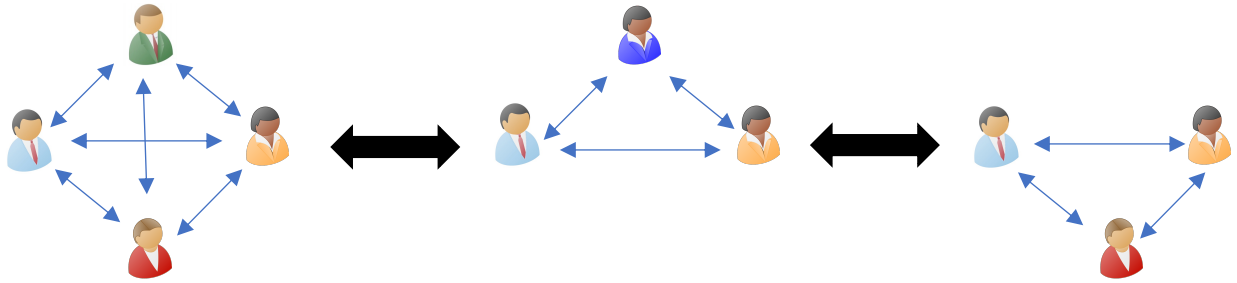
+

Dynamic Peer-to-peer networks.

Volunteer networks capable of private computation.

MPC-as-a-service framework - anyone can volunteer to participate irrespective of their computational power or availability.

MPC as a Service



MPC with Dynamic Participants

+

Dynamic Peer-to-peer networks.

Volunteer networks capable of private computation.

MPC-as-a-service framework - anyone can volunteer to participate irrespective of their computational power or availability.

Clients can delegate computations to such services.

Player Replaceability

- Byzantine Agreement [Mic17, CM19] : After every round, the current set of players can be replaced by new ones.

- Blockchains [GHMVZ17]: This idea is used in the design of Algorand.
  - Helps mitigate targeted attacks on chosen participants after their identity is revealed.

- YOSO [GHKMNRY21]: Extended the above notion to MPC.

Other Related Work

- Proactive MPC [OY91]
  - Static participants
  - Mobile adversaries

- Secret Sharing with dynamic participants [GKMPS20, BGGHKLRR20]
  - Computational setting
  - Guaranteed output delivery

Contributions

[Choudhuri-G-Green-Jain-Kaptchuk 21]

[Deligios-G-Liu-Zhang 23]

Fluid MPC: A formal model for MPC with dynamic participants [CGGJK21]

Semi-honest and maliciously secure Fluid MPC protocols

[CGGJK21]	Perfectly secure semi-honest
	Statistically secure with abort
[DGL23]	Perfectly secure with guaranteed output delivery (positive/negative)

Modeling Dynamic Computation

Modeling Dynamic Computation

- Client-server model
- Clients delegate computation to volunteer servers

Modeling Dynamic Computation

- Client-server model
- Clients delegate computation to volunteer servers

Input Stage

Clients pre-process their inputs and hand them to the servers

Modeling Dynamic Computation

- Client-server model
- Clients delegate computation to volunteer servers

Input Stage	Execution Stage
Clients pre-process their inputs and hand them to the servers	Dynamic servers participate to compute the function

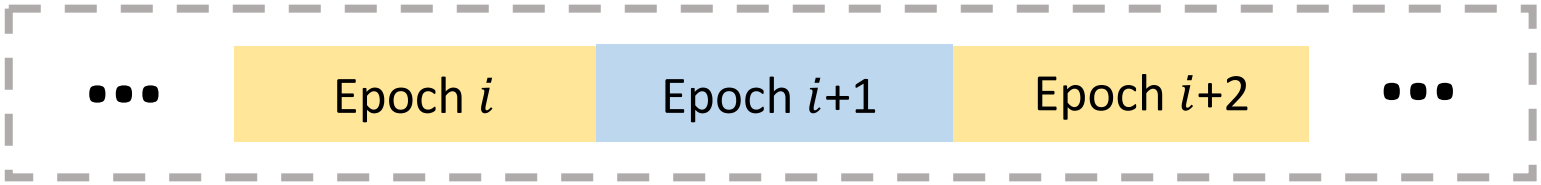
Modeling Dynamic Computation

- Client-server model
- Clients delegate computation to volunteer servers

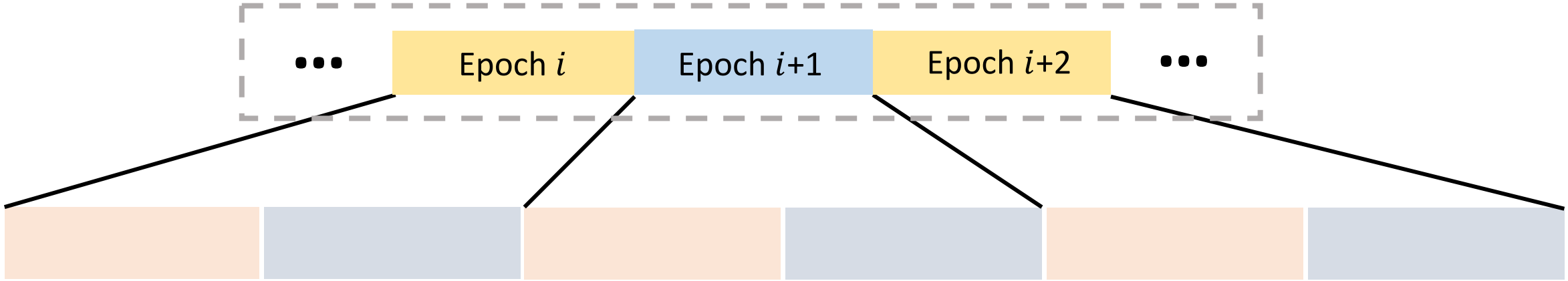
Input Stage	Execution Stage	Output Stage
Clients pre-process their inputs and hand them to the servers	Dynamic servers participate to compute the function	Clients reconstruct the output of the function

Modeling Execution Stage

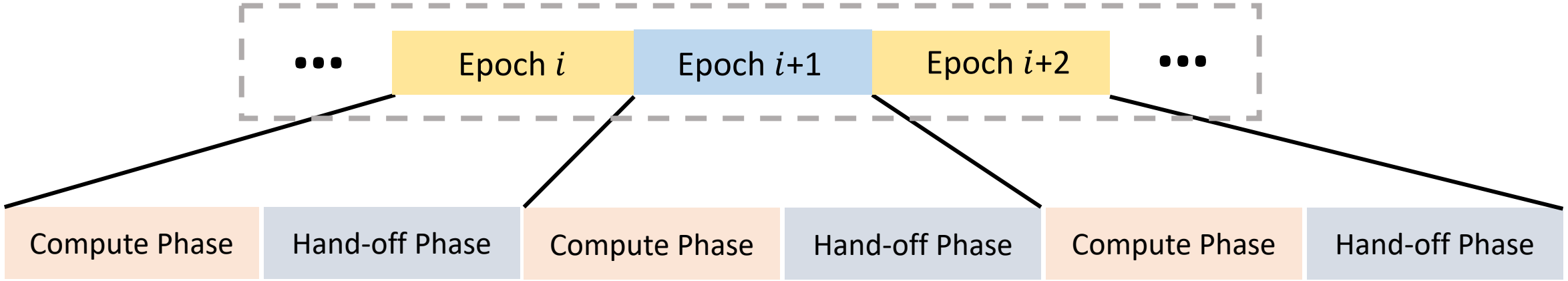
Modeling Execution Stage



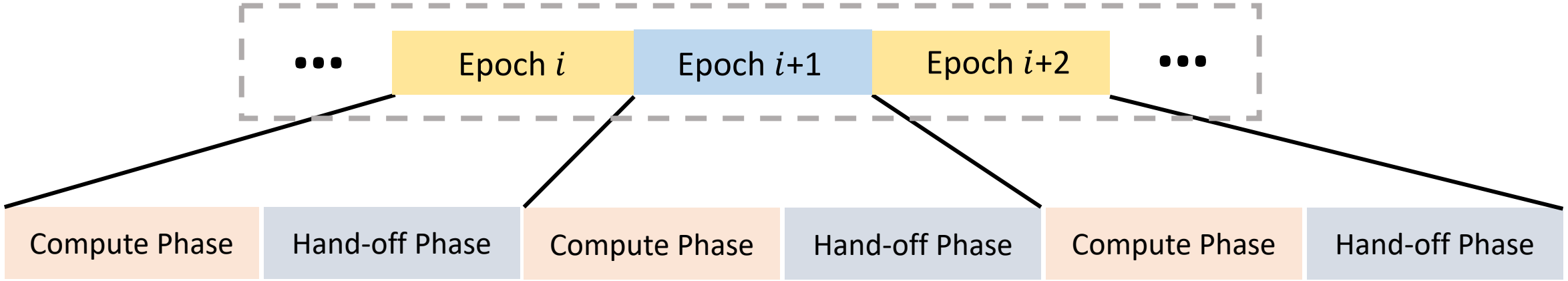
Modeling Execution Stage



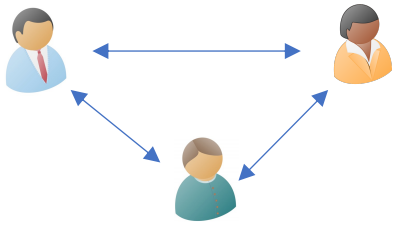
Modeling Execution Stage



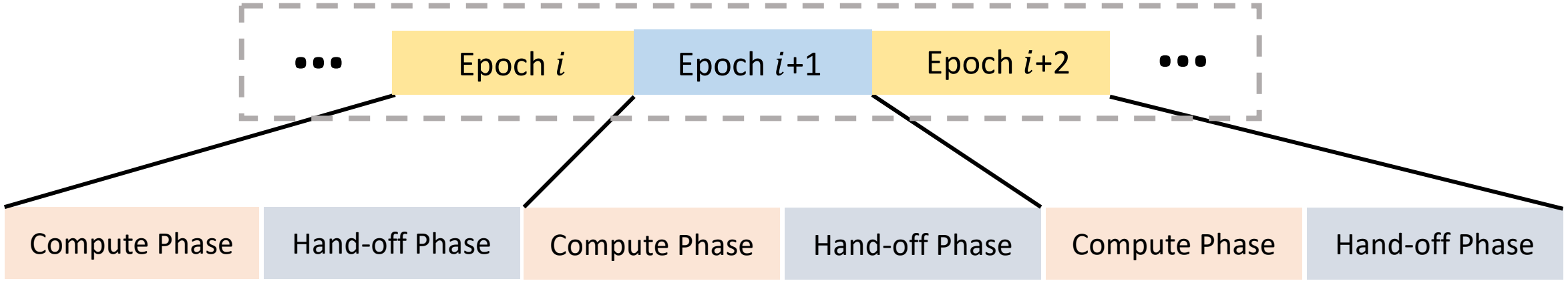
Modeling Execution Stage



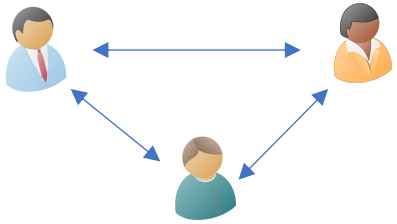
Committee S^i



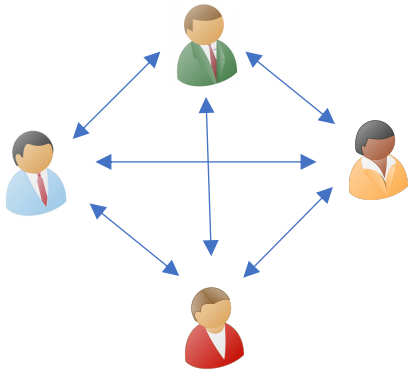
Modeling Execution Stage



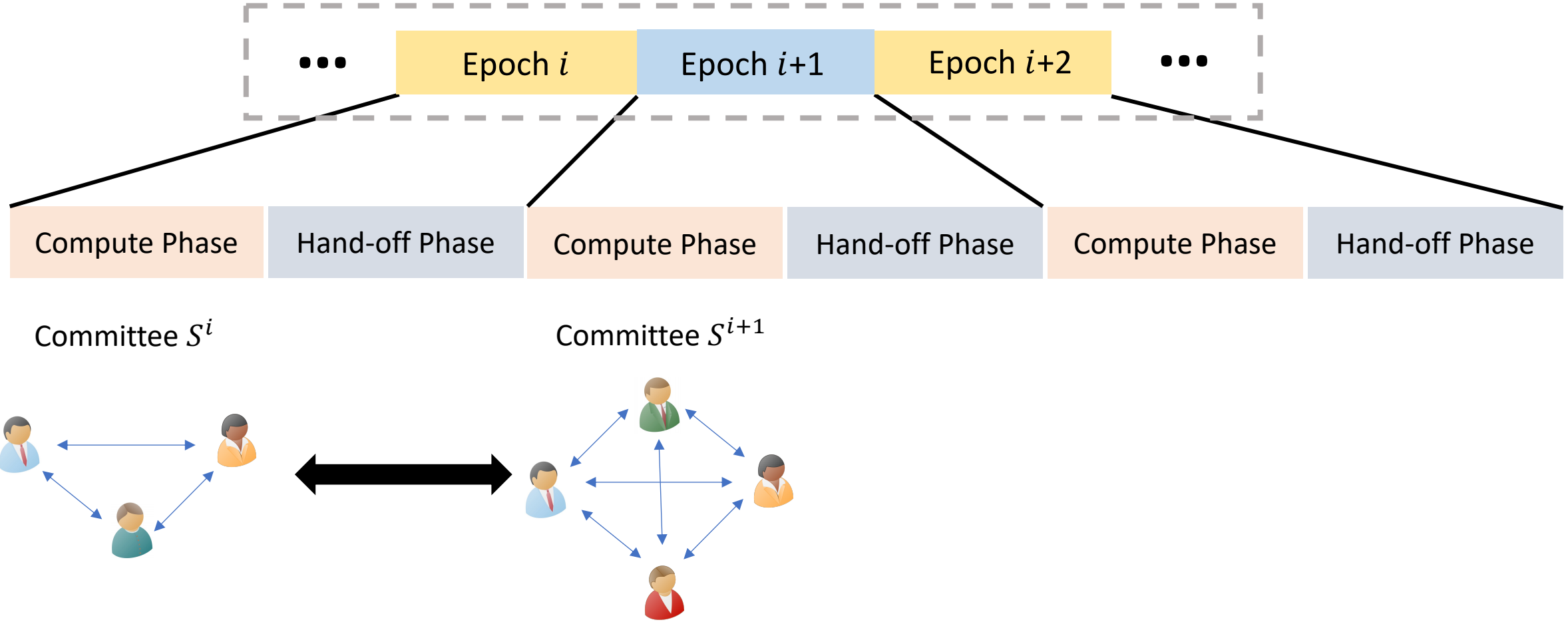
Committee S^i



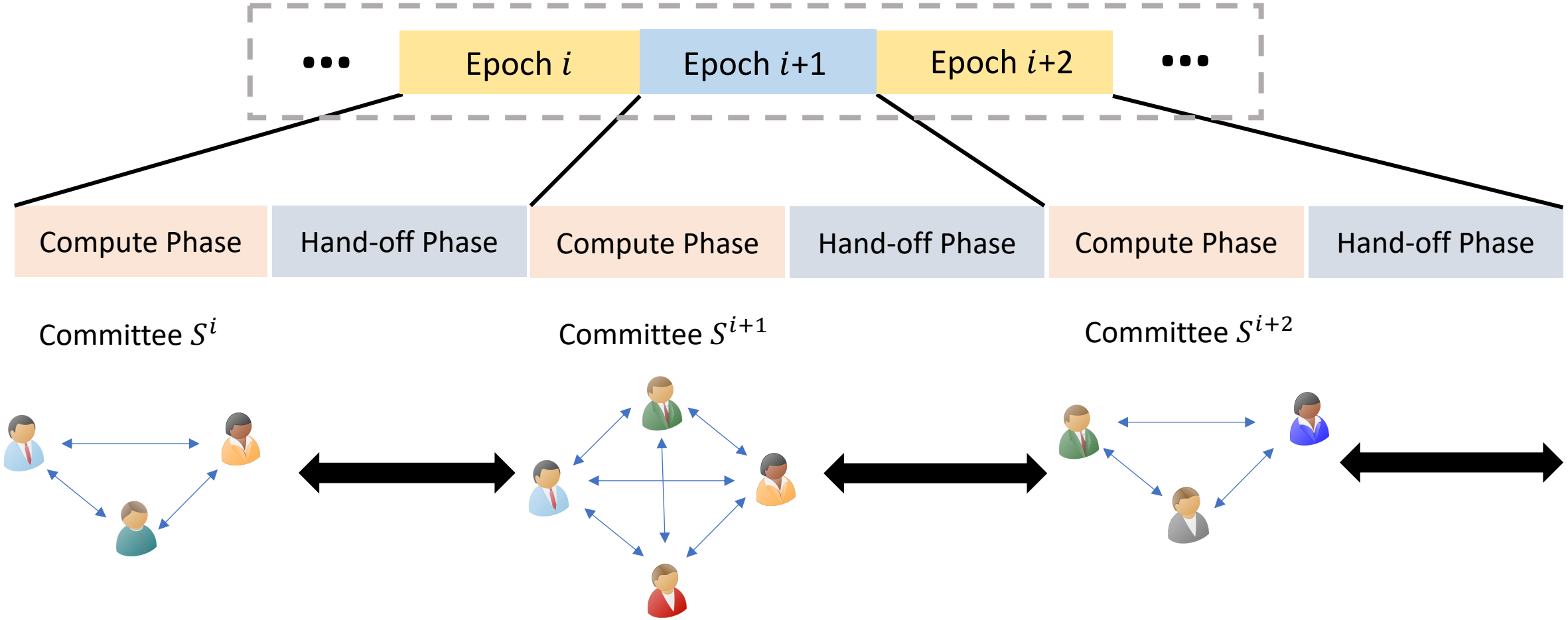
Committee S^{i+1}



Modeling Execution Stage

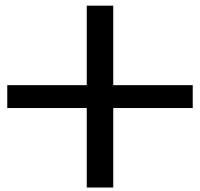


Modeling Execution Stage



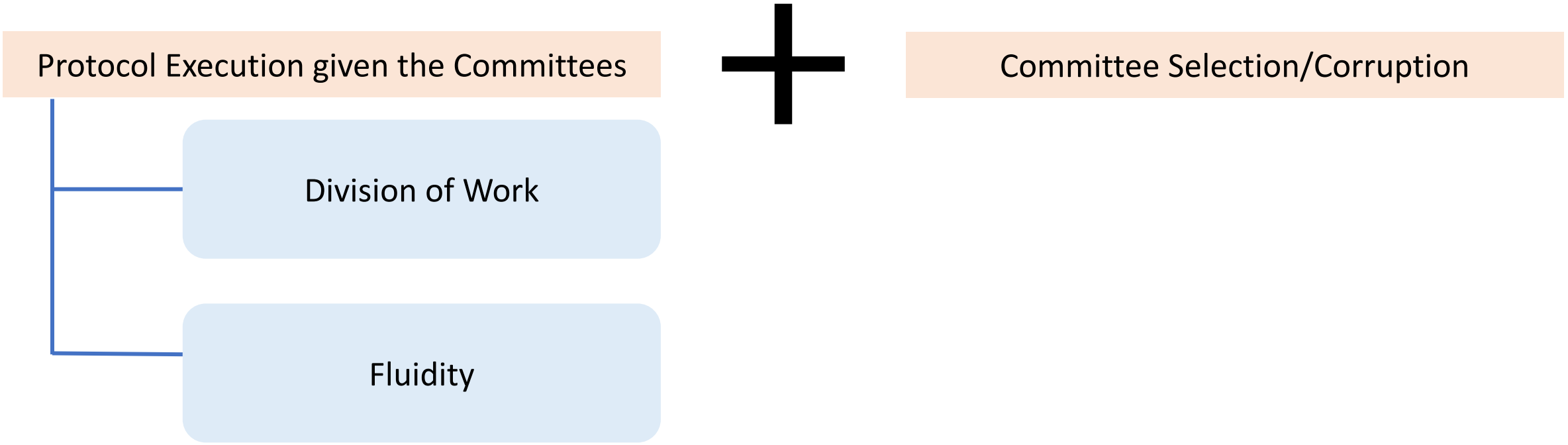
Fluid MPC Protocol

Protocol Execution given the Committees



Committee Selection/Corruption

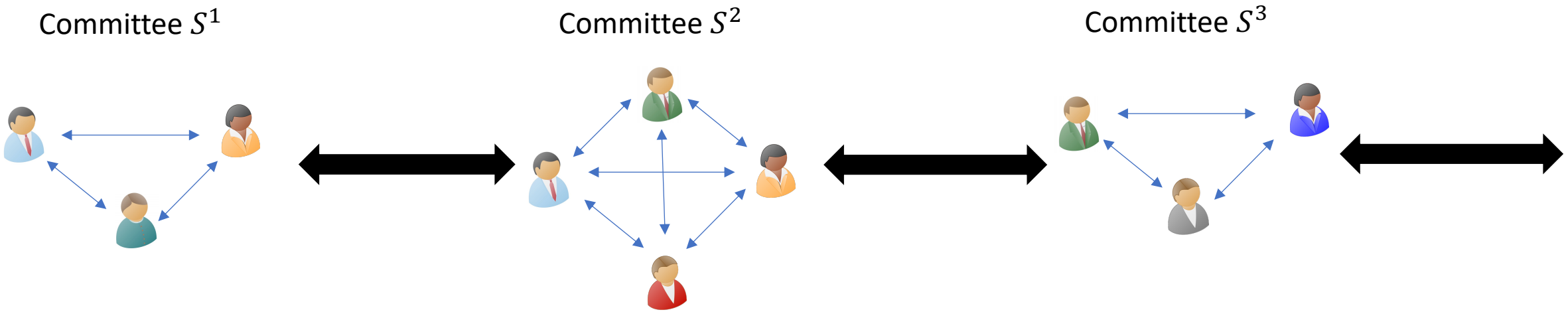
Fluid MPC Protocol



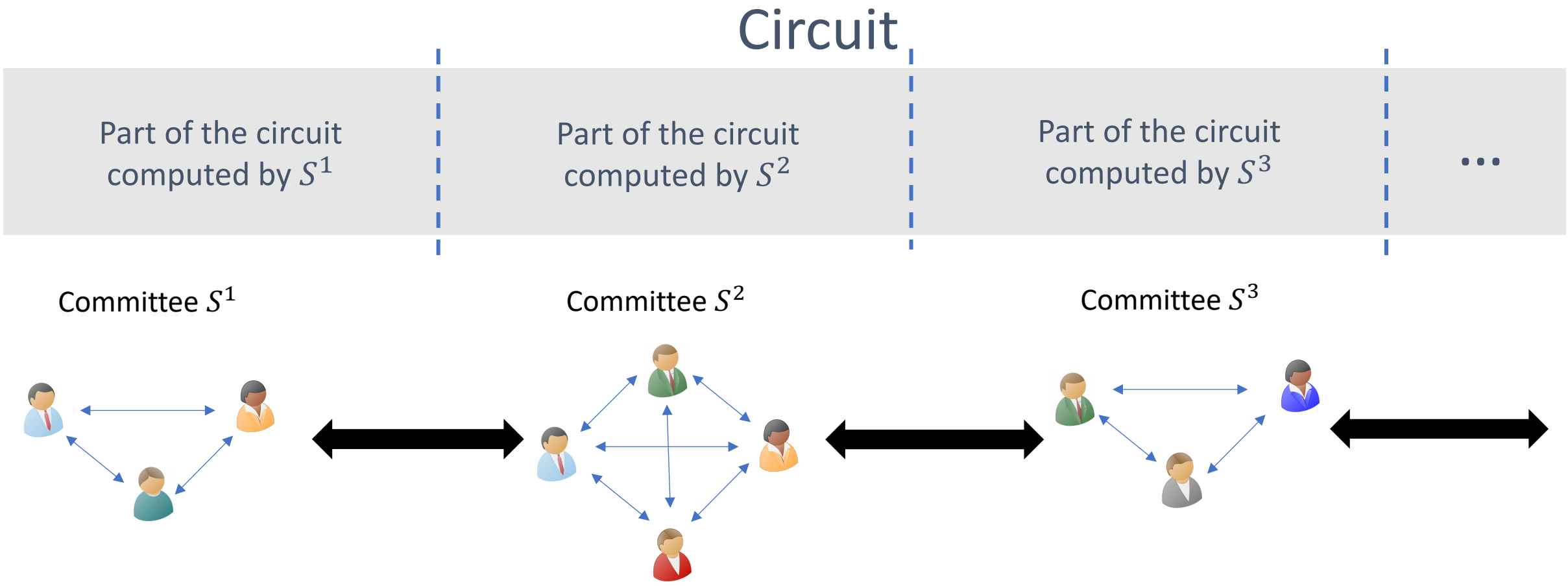
Division of Work

Division of Work

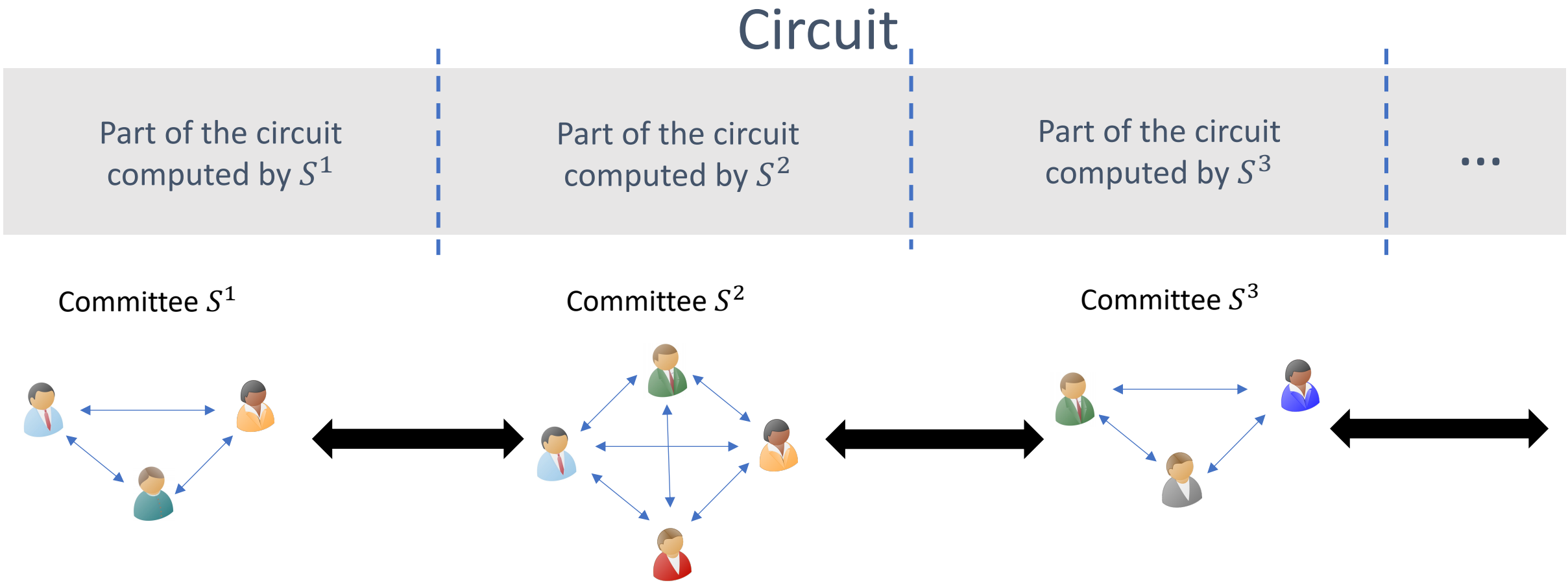
Circuit



Division of Work

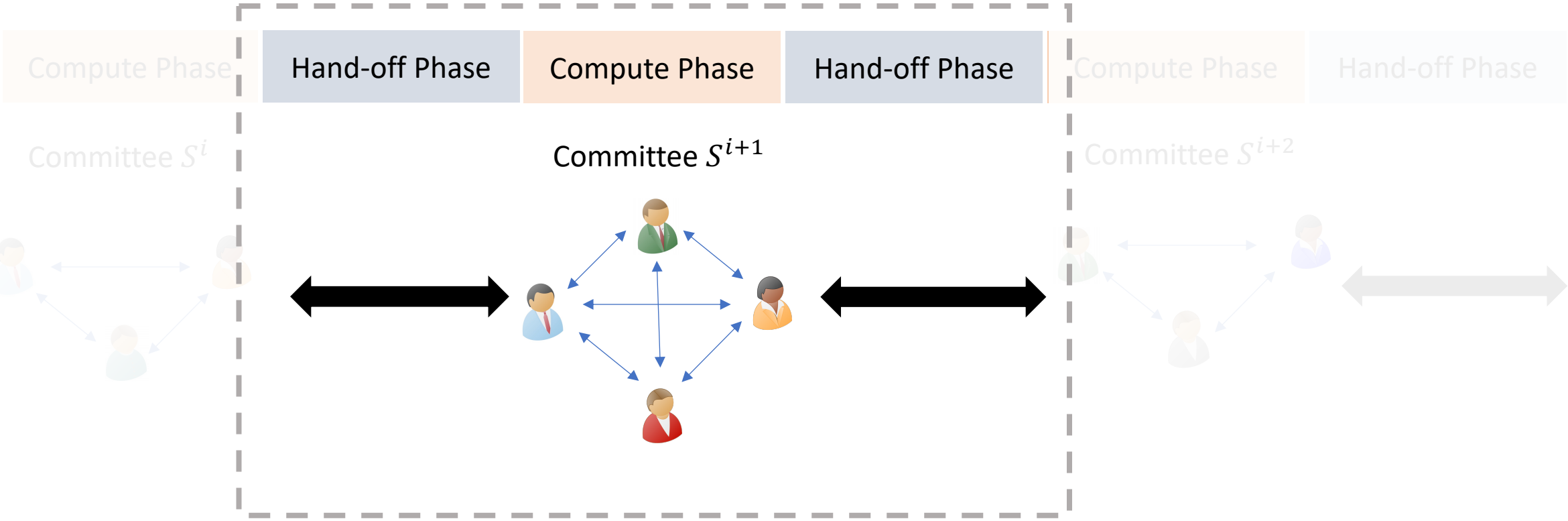


Division of Work



Per-committee work independent of the depth of the circuit

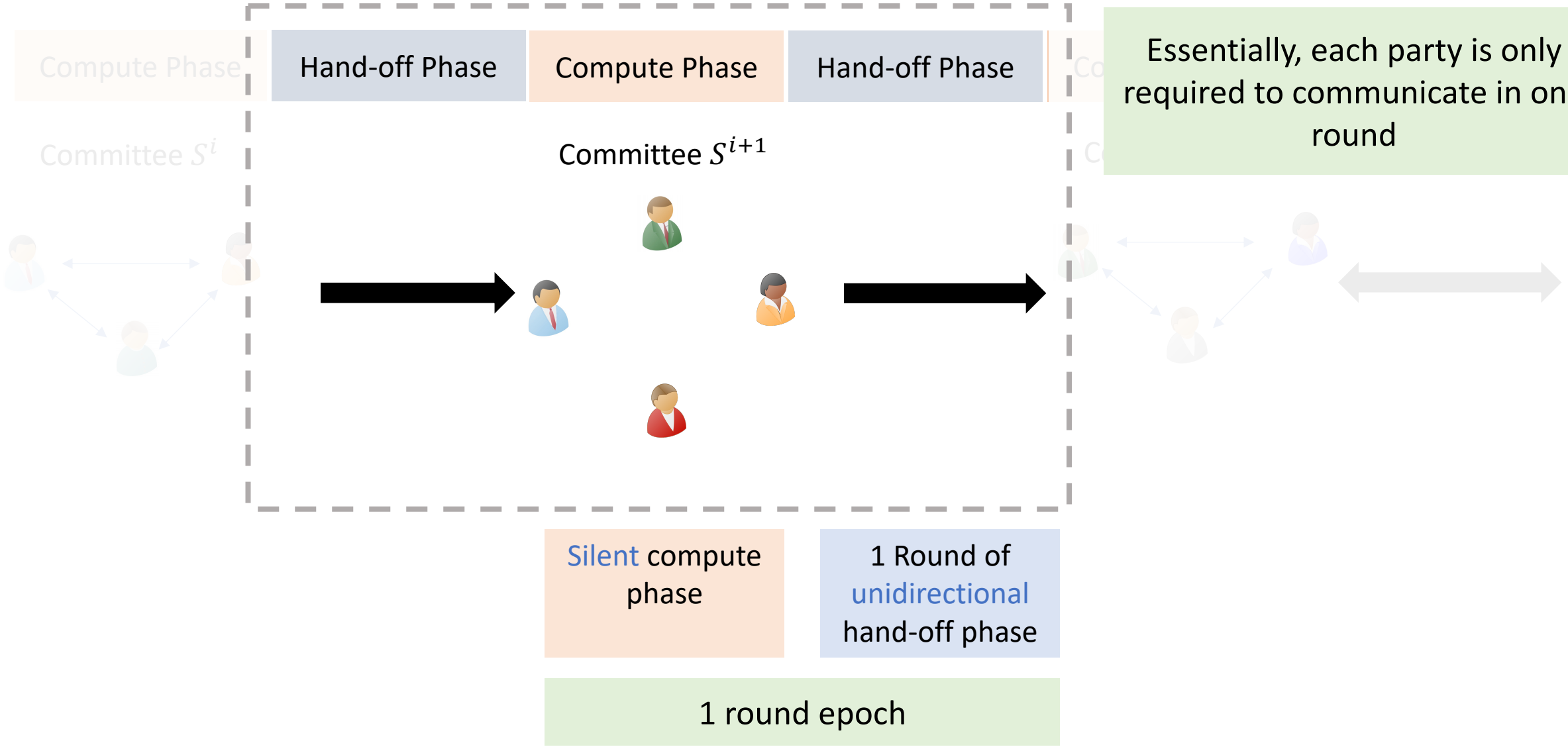
Fluidity



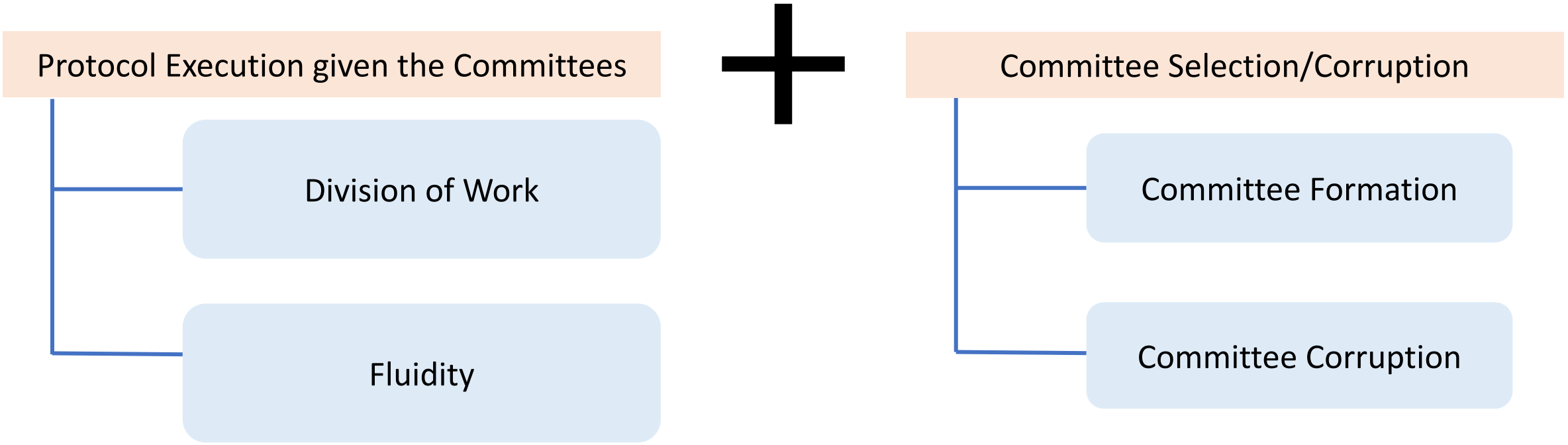
Fluidity is the minimum commitment a server needs to make for participating in the protocol.

Measured by the number of rounds in an epoch

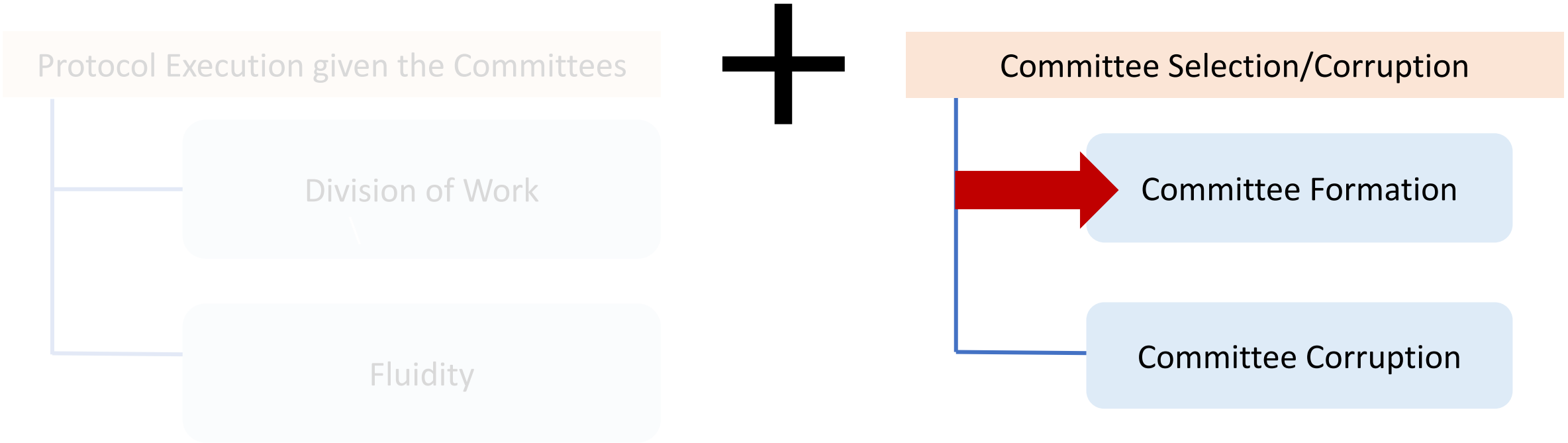
Maximal Fluidity



Fluid MPC Protocol

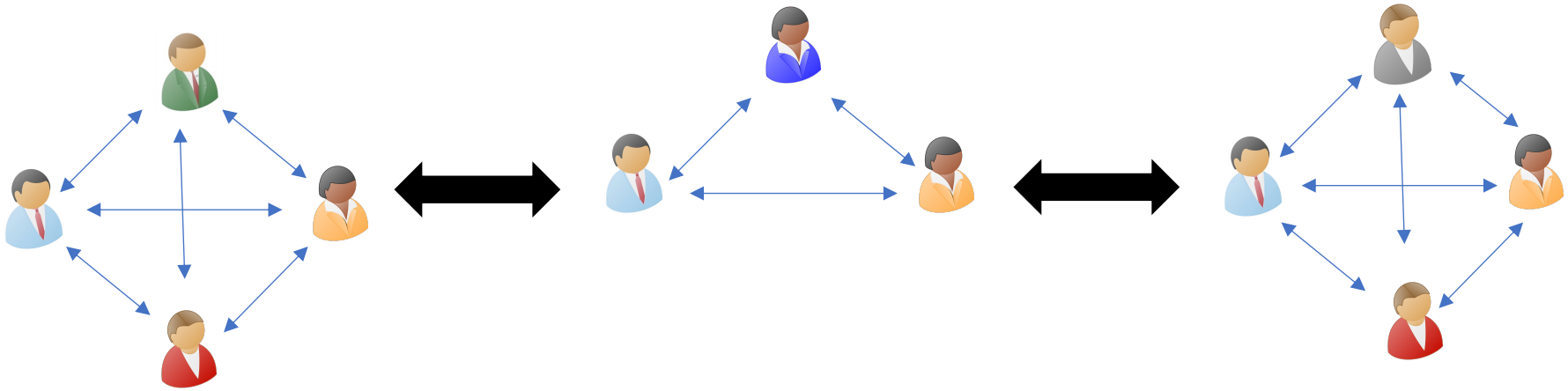


Fluid MPC Protocol



Committees: When are they formed?

Epoch	Committee
1	
2	
3	



Input Stage

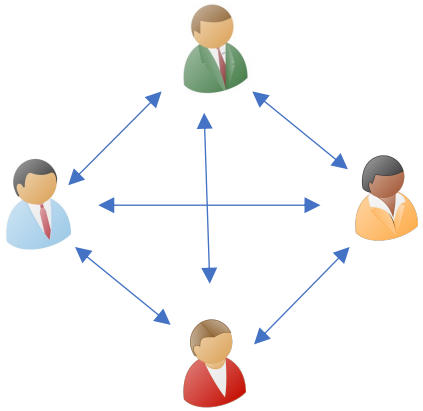
Execution Stage

Static Committee Formation: Committee for each epoch is known at the start of the protocol.

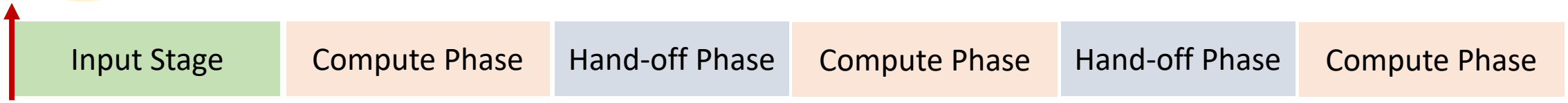
Too Restrictive!

Committees: When are they formed?

Committee S^1

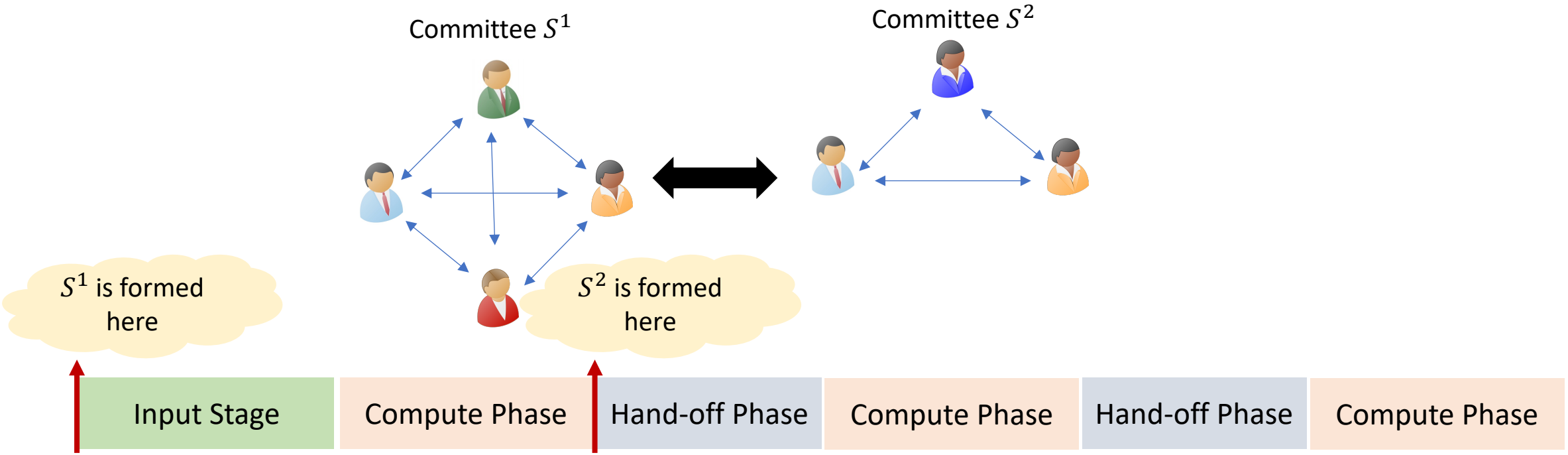


S^1 is formed here



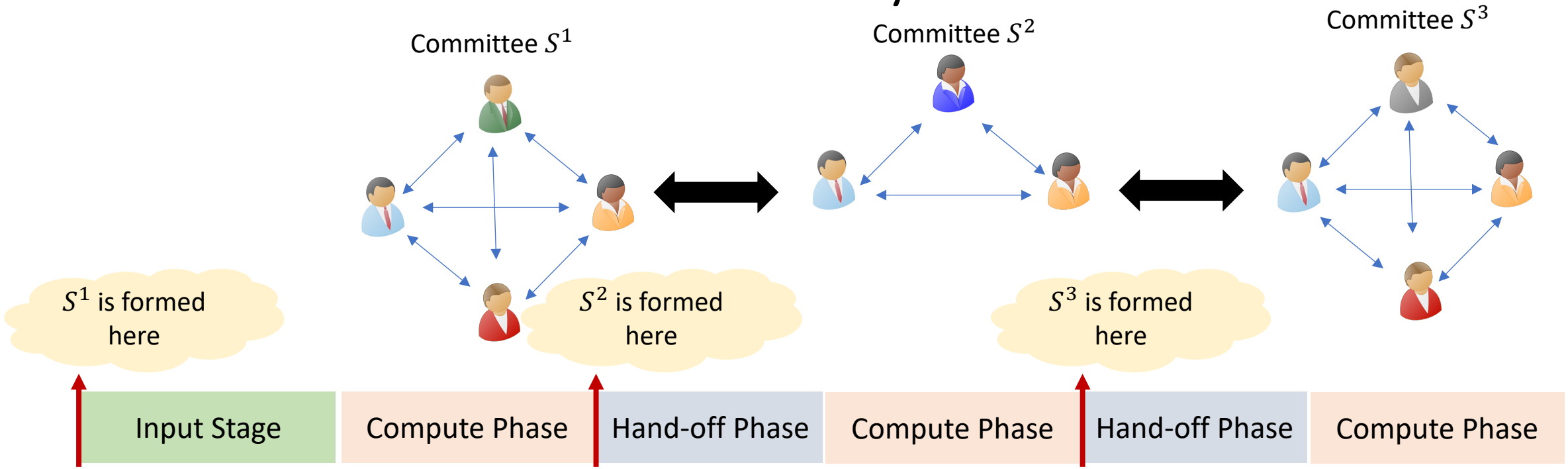
On-the-fly Committee Formation: Committee for each epoch is known at the start of the hand-off phase of the previous epoch.

Committees: When are they formed?



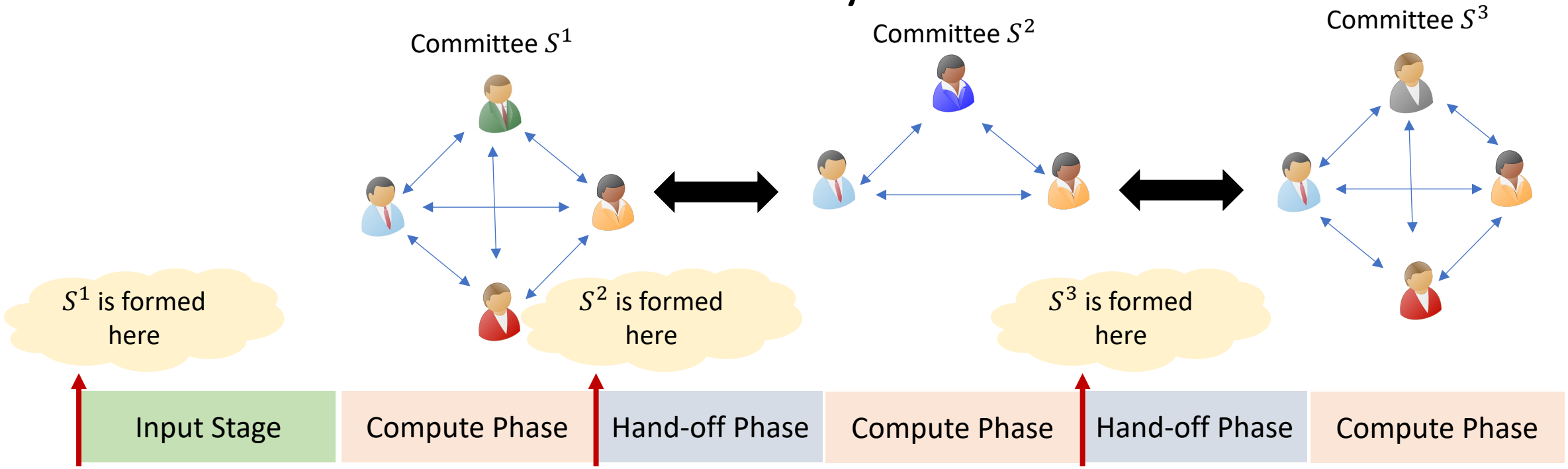
On-the-fly Committee Formation: Committee for each epoch is known at the start of the hand-off phase of the previous epoch.

Committees: When are they formed?



On-the-fly Committee Formation: Committee for each epoch is known at the start of the hand-off phase of the previous epoch.

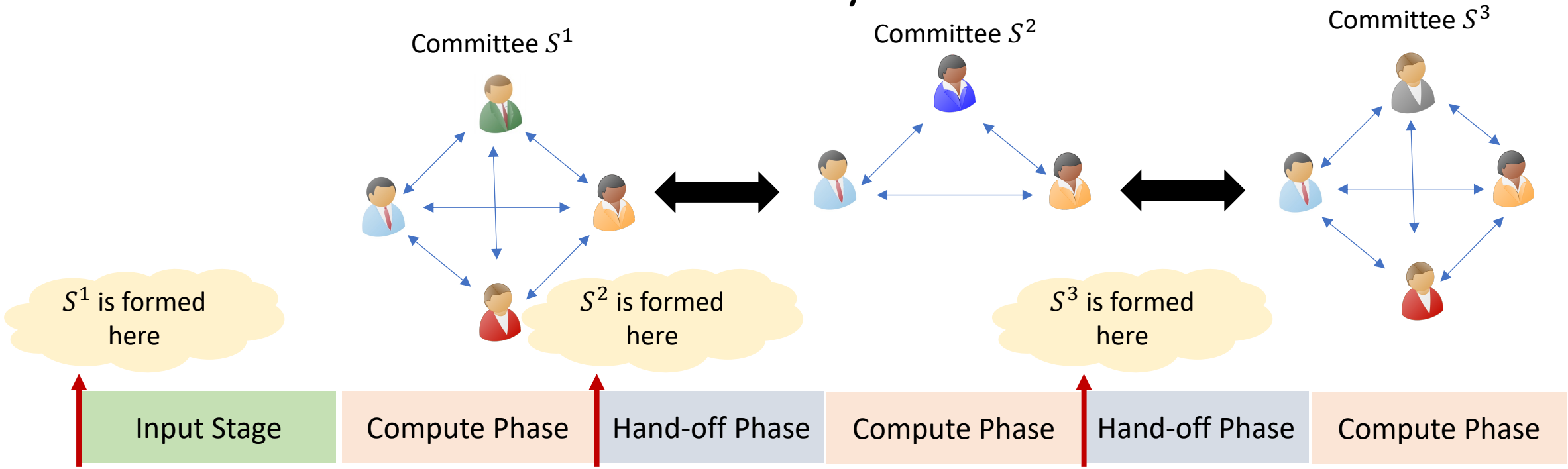
Committees: How are they formed?



On-the-fly Committee Formation:

Volunteer: Anyone who volunteers can join the computation (Corruption threshold is difficult to enforce)

Committees: How are they formed?



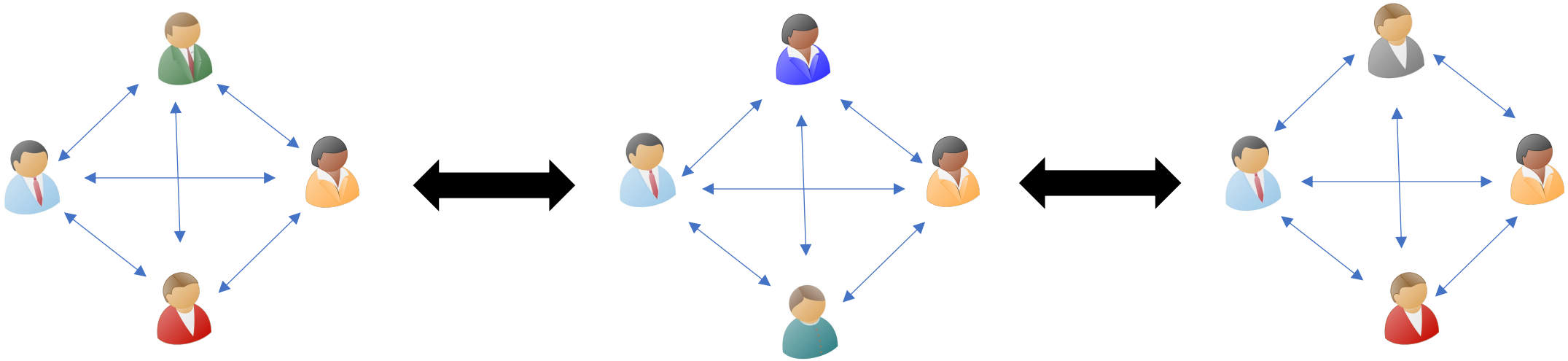
On-the-fly Committee Formation:

Volunteer: Anyone who volunteers can join the computation (Corruption threshold is difficult to enforce)

Elected: Anyone can nominate themself and an election process decides which nominees will participate (e.g., [BGGHKLRR20, GHMNY20] uses proof-of-stake blockchains)

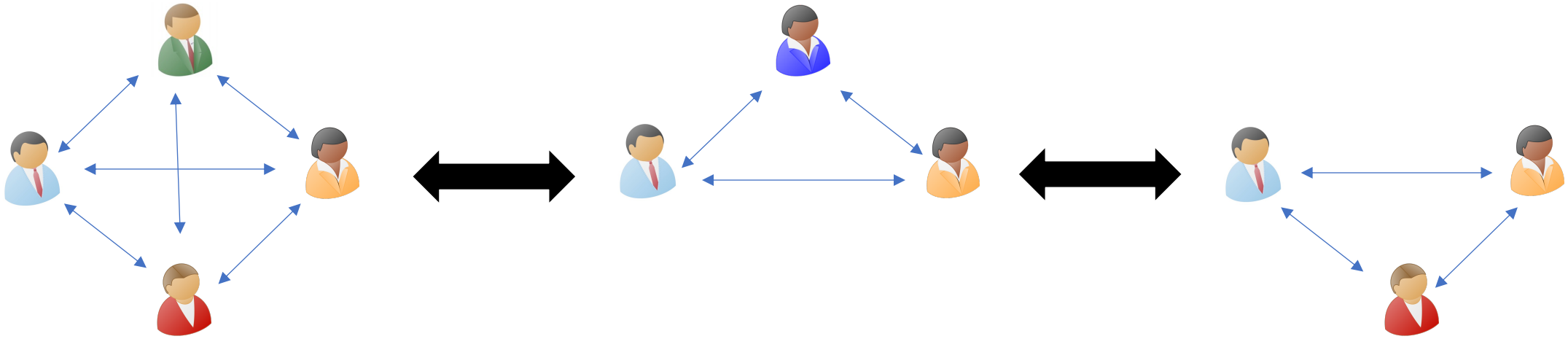
Committees: Size

Fixed-Sized Committees



Committees: Size

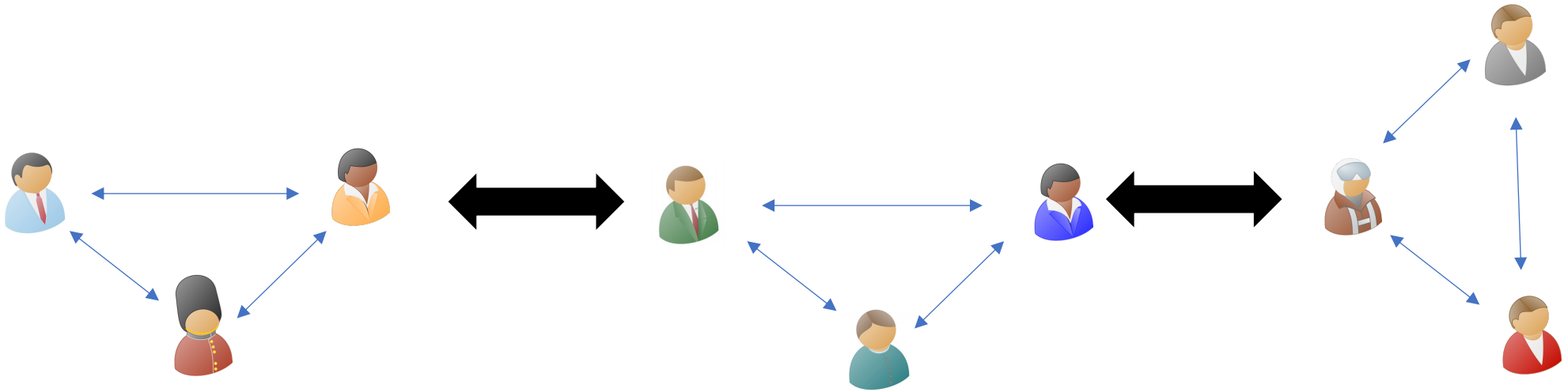
Variable-Sized Committees



Committees: Overlap

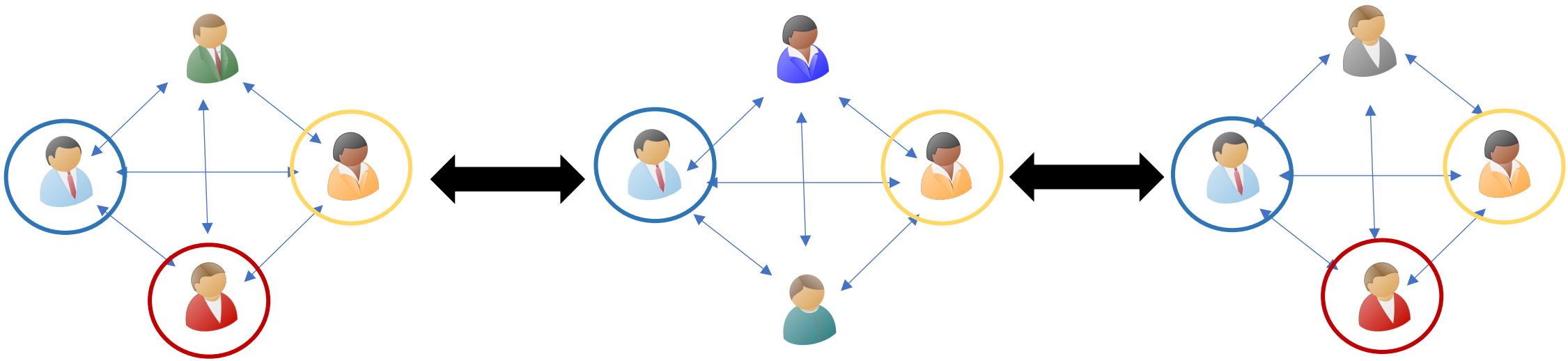
Committees: Overlap

Restricted or No overlap of parties across committees

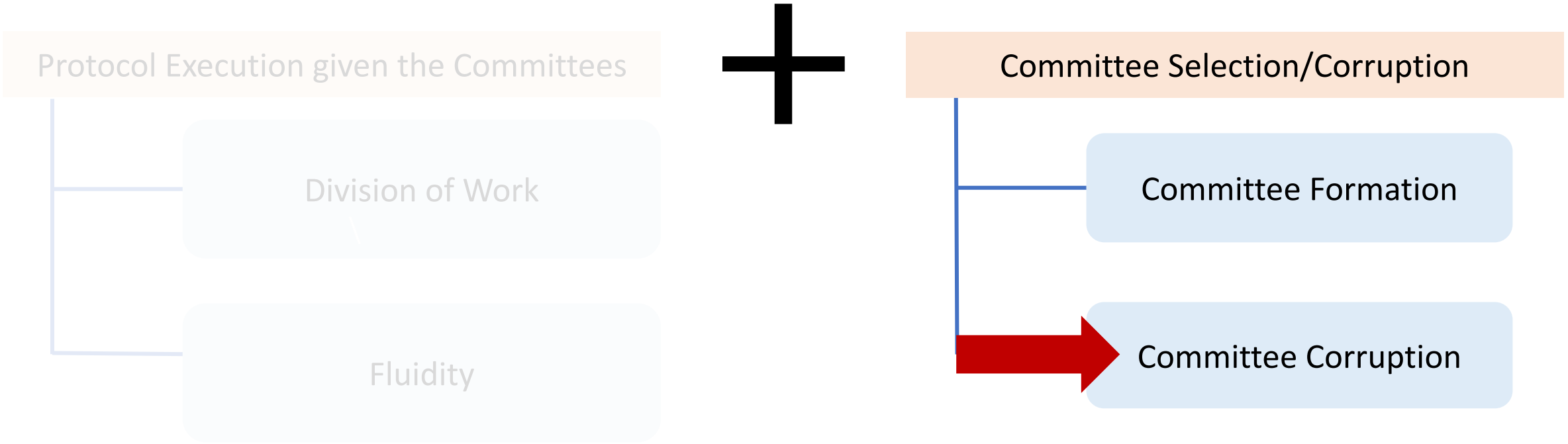


Committees: Overlap

Unrestricted overlap of parties across committees

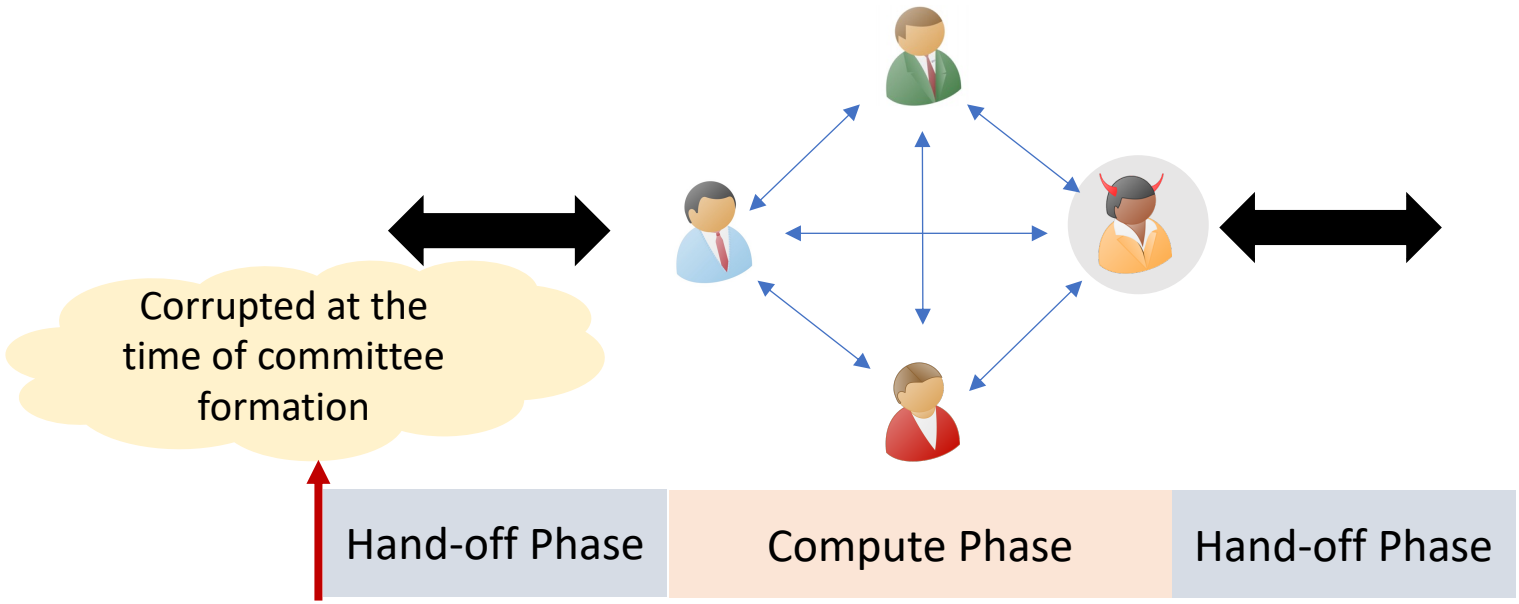


Fluid MPC Protocol



Committee Corruption

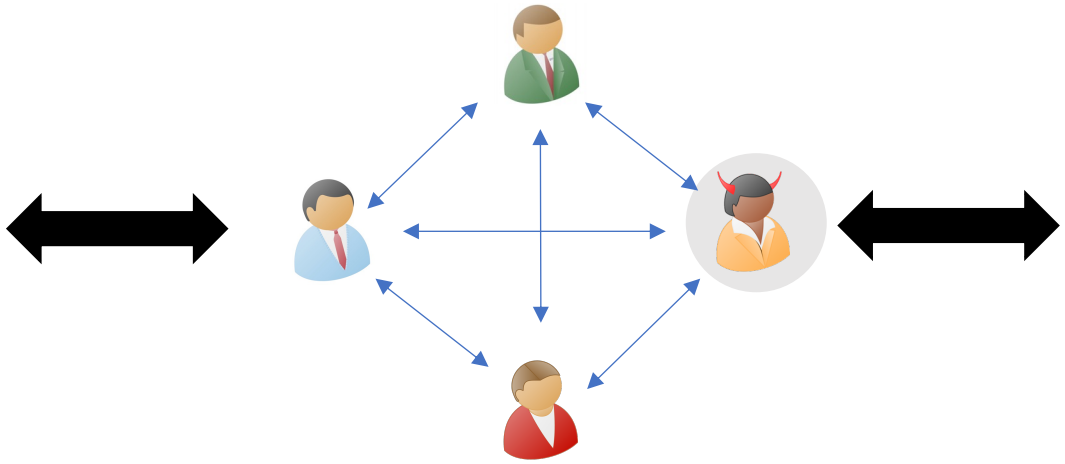
When can a server be corrupted?



Static Corruption

Committee Corruption

When can a server be corrupted?

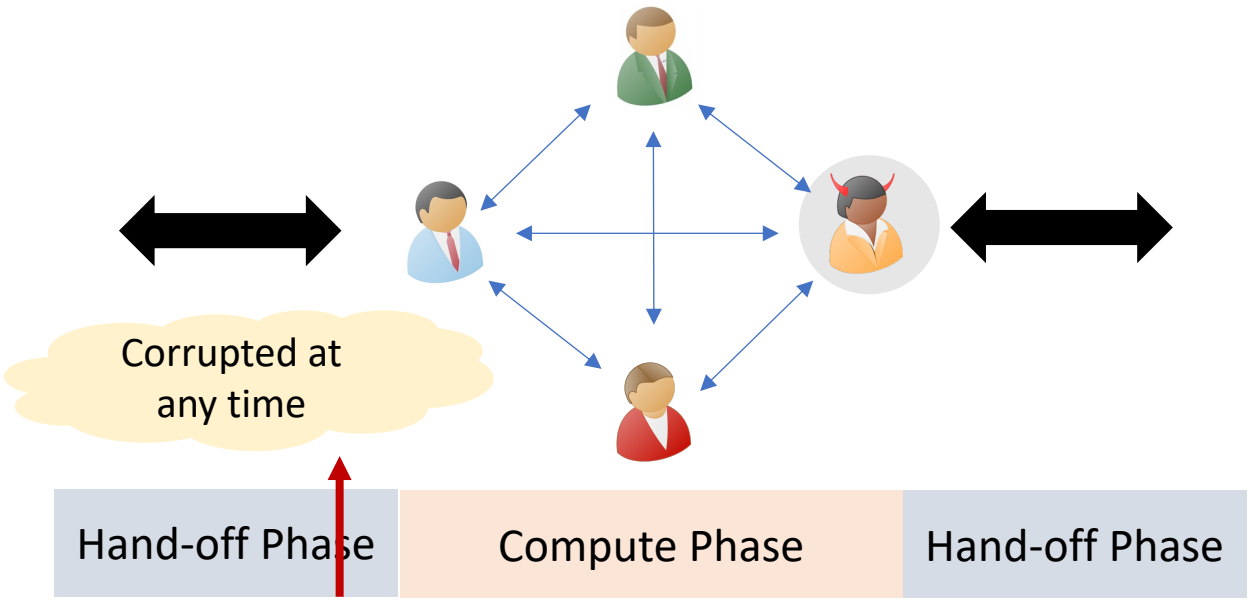


Hand-off Phase | Compute Phase | Hand-off Phase

Adaptive Corruption

Committee Corruption

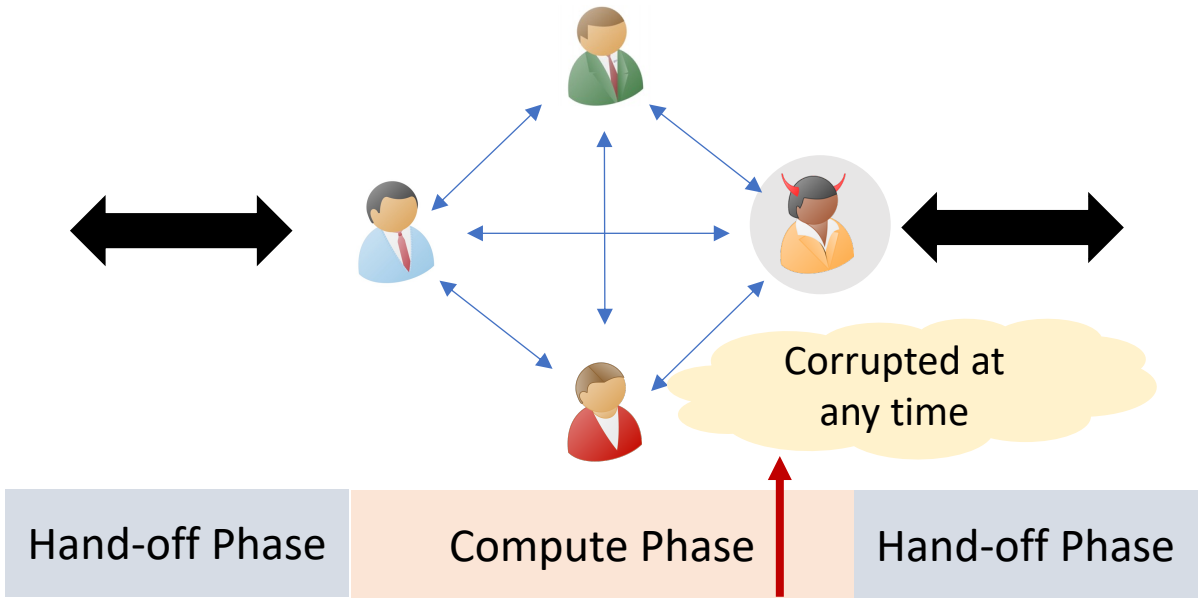
When can a server be corrupted?



Adaptive Corruption

Committee Corruption

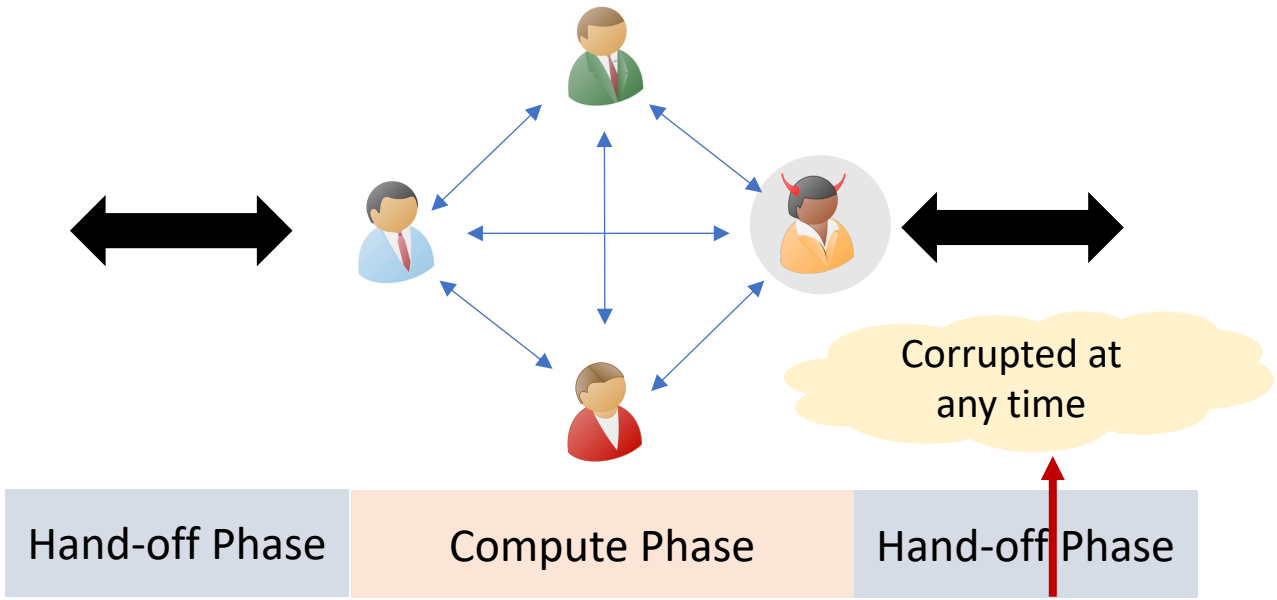
When can a server be corrupted?



Adaptive Corruption

Committee Corruption

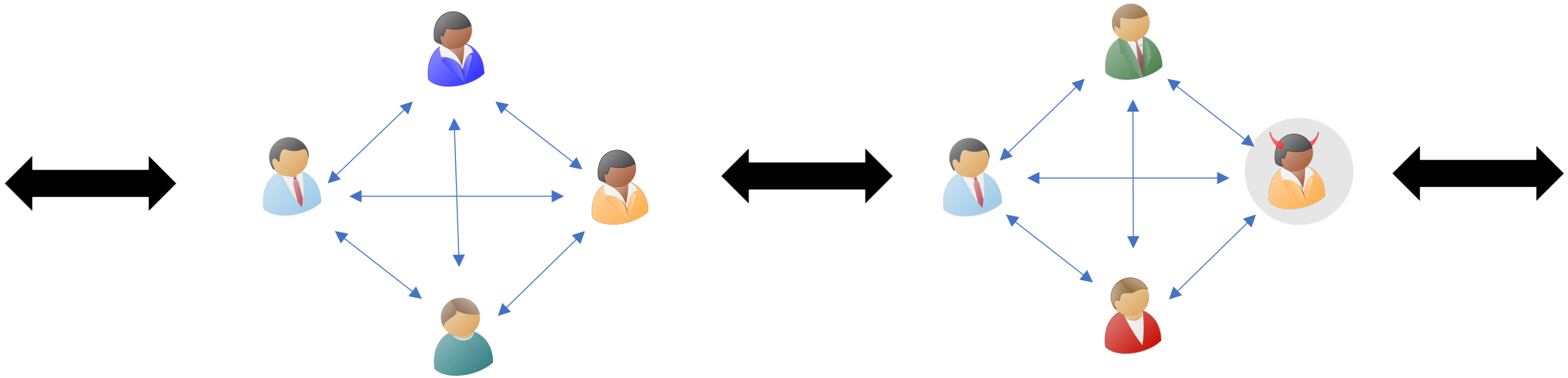
When can a server be corrupted?



Adaptive Corruption

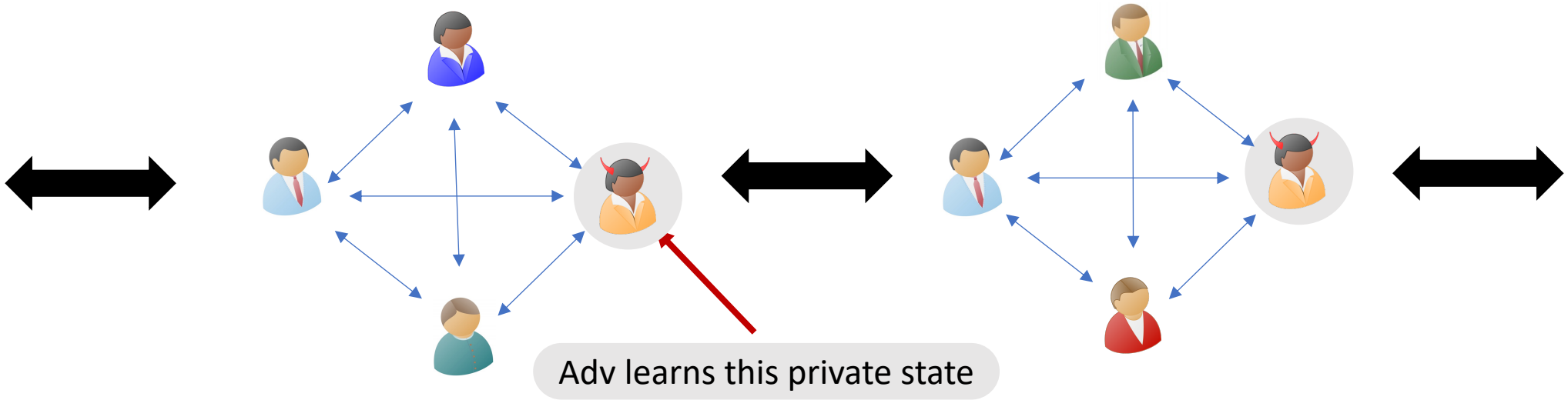
Effect of Committee Corruption on Prior Epochs

What effect does corrupting a server have on the prior epochs where it participated?



Effect of Committee Corruption on Prior Epochs

What effect does corrupting a server have on the prior epochs where it participated?



If there is overlap across committees, a server can only be corrupted if it does not violate the corruption threshold of prior epochs.

Fluid MPC Protocols

Perfectly secure semi-honest [CGGJK21]

Statistically secure with abort [CGGJK21]

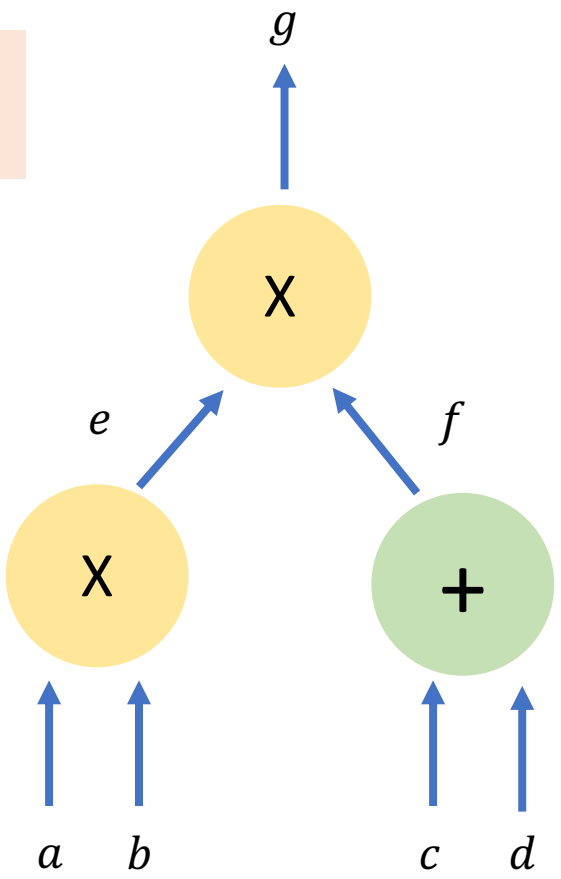
Perfectly secure with guaranteed output delivery [DGL23]

Fluid MPC Protocol (Semi-Honest)

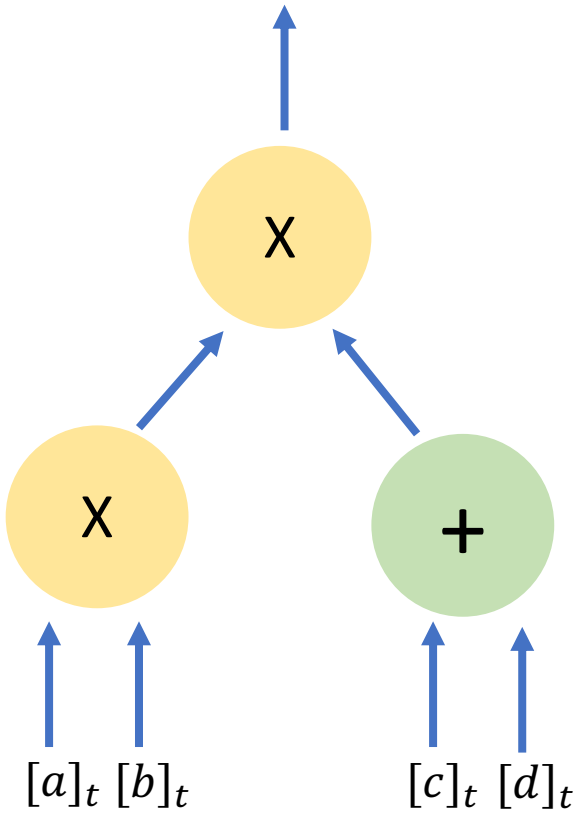
Semi-Honest BGW [GRR98] can be adapted to obtain a maximally Fluid semi-honest MPC

Semi-honest BGW [GRR98]

Gate-by-Gate evaluation on secret shared inputs



Semi-honest BGW [GRR98]



Input sharing: t-out-of-n shares of inputs

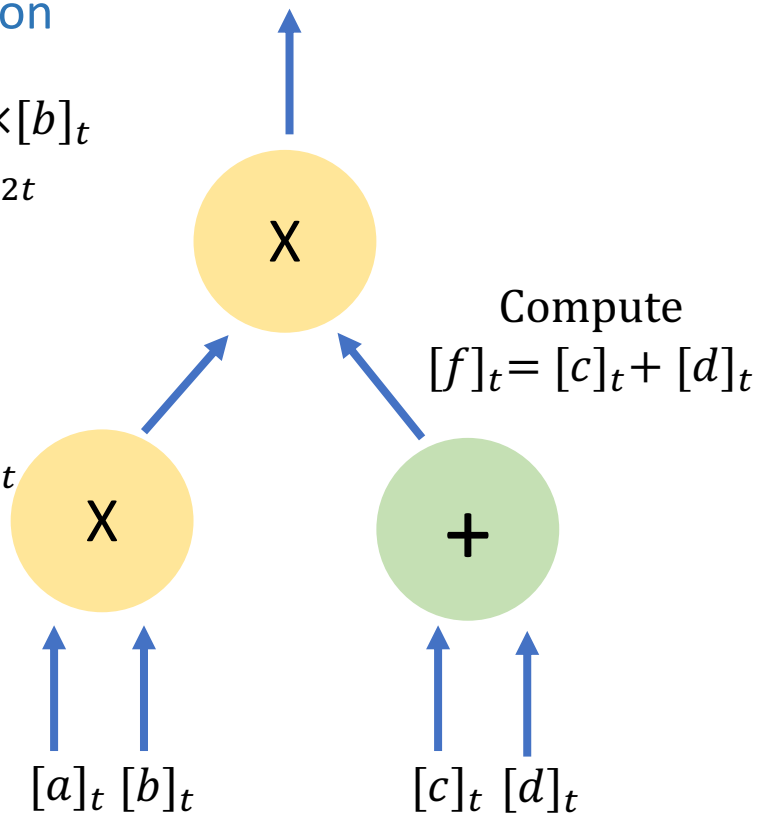
Semi-honest BGW [GRR98]

Gate-by-Gate Evaluation

Compute $[e]_{2t} = [a]_t \times [b]_t$
$[[e]_{2t}]_t \leftarrow [e]_{2t}$

Exchange $[[e]_{2t}]_t$
(Shares of Shares)

Compute $[e]_t \leftarrow [[e]_{2t}]_t$



Input sharing: t-out-of-n shares of inputs

Semi-honest BGW [GRR98]

Output Reconstruction

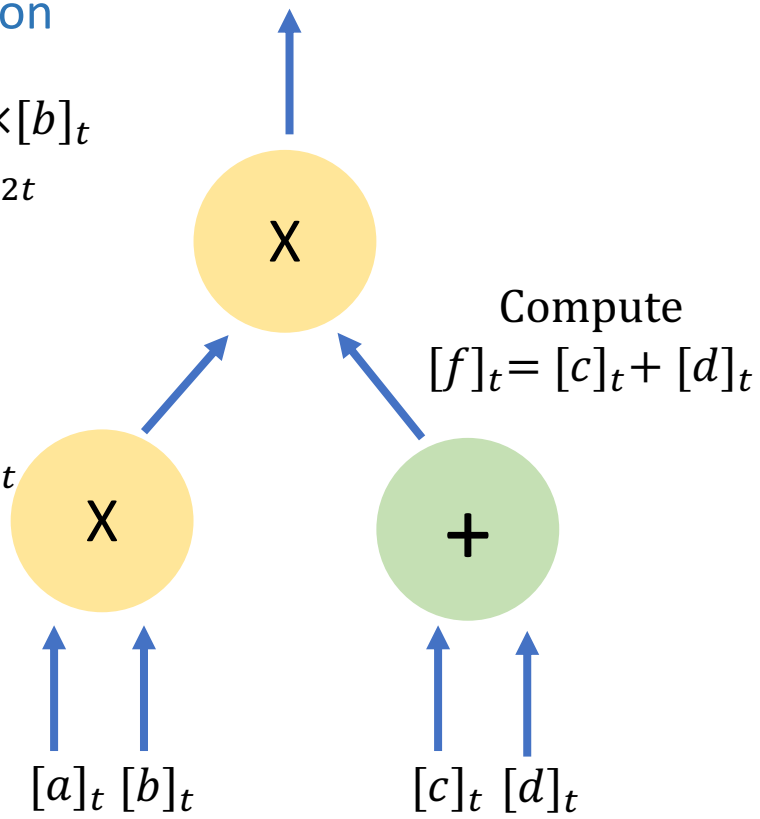
$g \leftarrow [g]_t$

Gate-by-Gate Evaluation

Compute $[e]_{2t} = [a]_t \times [b]_t$
$[[e]_{2t}]_t \leftarrow [e]_{2t}$

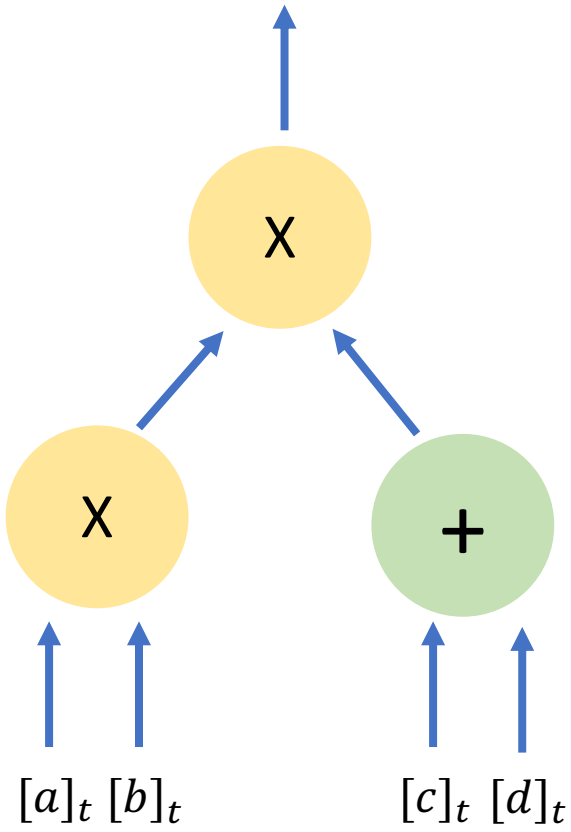
Exchange $[[e]_{2t}]_t$
(Shares of Shares)

Compute $[e]_t \leftarrow [[e]_{2t}]_t$



Input sharing: t-out-of-n shares of inputs

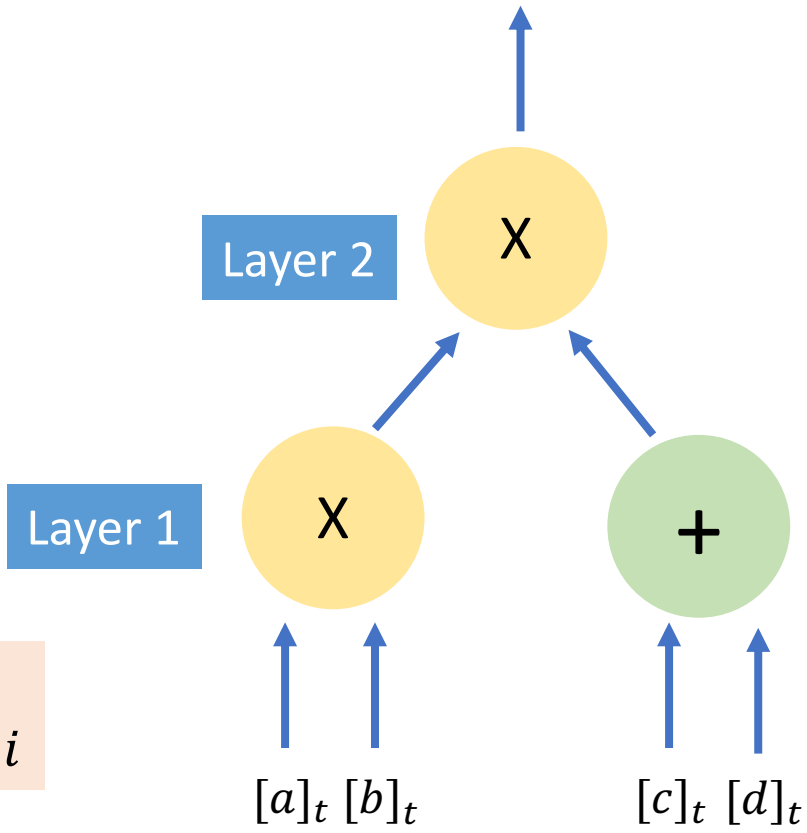
Semi-honest Fluid-BGW



Input Phase: Clients send t-out-of-n shares of inputs to the first committee

Semi-honest Fluid-BGW

Execution Stage



Layer-wise computations
Committee i computes layer i

Input Phase: Clients send t-out-of-n shares of inputs to the first committee

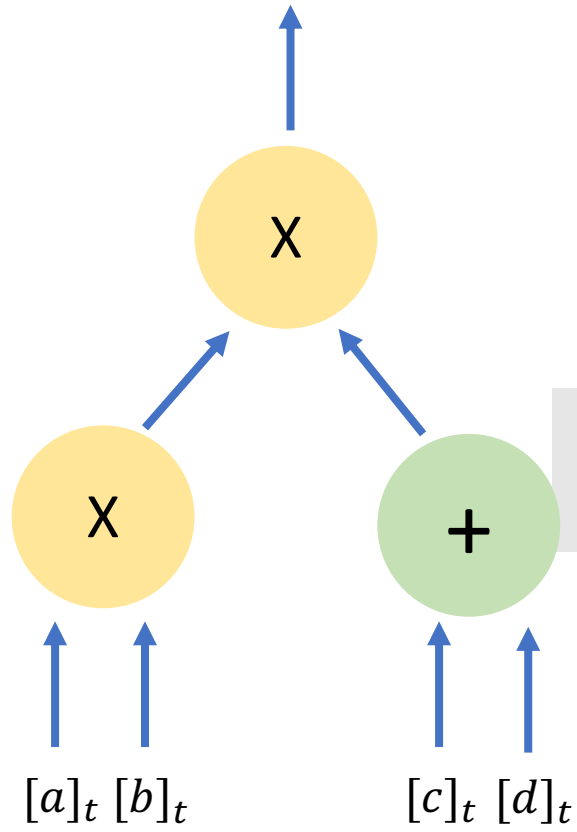
Semi-honest Fluid-BGW

Execution Stage

Computation Phase : $[e]_t \leftarrow [[e]_{2t}]_t$
of Epoch 2

Handoff Phase $[[e]_{2t}]_t$

Computation Phase : $[e]_{2t} = [a]_t \times [b]_t$
of Epoch 1 $[[e]_{2t}]_t \leftarrow [e]_{2t}$



Computation Phase : $[f]_t \leftarrow [[f]_t]_t$
of Epoch 2

Handoff Phase $[[f]_t]_t$

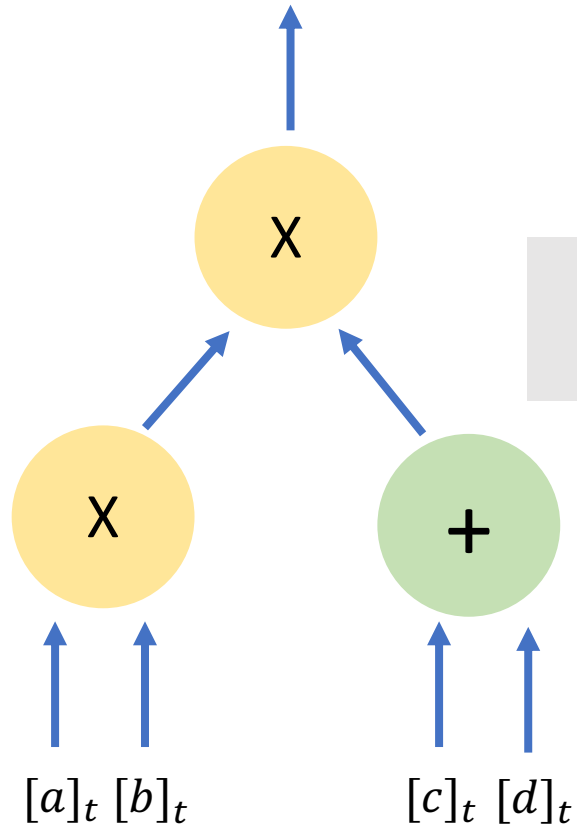
Computation Phase : $[f]_t = [c]_t + [d]_t$
of Epoch 1 $[[f]_t]_t \leftarrow [f]_t$

Input Phase: Clients send t-out-of-n shares of inputs to the first committee

Semi-honest Fluid-BGW

Execution Stage

Computation Phase : of Epoch 2	$[e]_t \leftarrow [[e]_{2t}]_t$
Handoff Phase	$[[e]_{2t}]_t$
Computation Phase : of Epoch 1	$[e]_{2t} = [a]_t \times [b]_t$ $[[e]_{2t}]_t \leftarrow [e]_{2t}$



Computation Phase : of Epoch 2	$[f]_t \leftarrow [[f]_t]_t$
Handoff Phase	$[[f]_t]_t$
Computation Phase : of Epoch 1	$[f]_t = [c]_t + [d]_t$ $[[f]_t]_t \leftarrow [f]_t$

Input Phase: Clients send t-out-of-n shares of inputs to the first committee

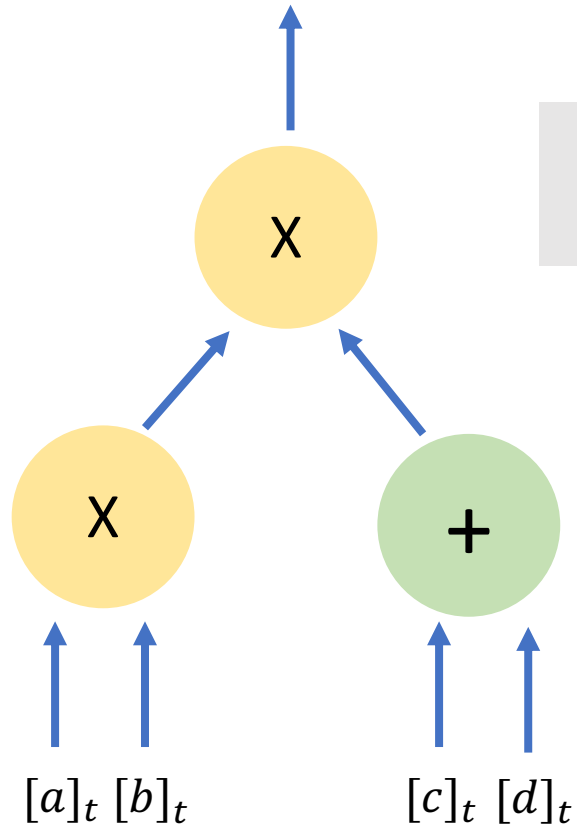
Semi-honest Fluid-BGW

Execution Stage

Computation Phase : $[e]_t \leftarrow [[e]_{2t}]_t$
of Epoch 2

Handoff Phase $[[e]_{2t}]_t$

Computation Phase : $[e]_{2t} = [a]_t \times [b]_t$
of Epoch 1 $[[e]_{2t}]_t \leftarrow [e]_{2t}$



Computation Phase : $[f]_t \leftarrow [[f]_t]_t$
of Epoch 2

Handoff Phase $[[f]_t]_t$

Computation Phase : $[f]_t = [c]_t + [d]_t$
of Epoch 1 $[[f]_t]_t \leftarrow [f]_t$

Input Phase: Clients send t-out-of-n shares of inputs to the first committee

Semi-honest Fluid-BGW

Output Phase

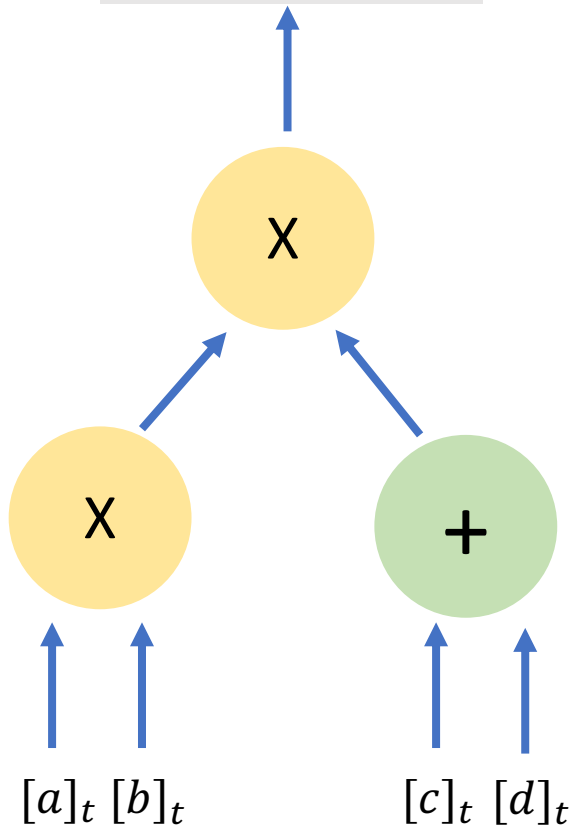
$g \leftarrow [g]_t$

Execution Stage

Computation Phase : $[e]_t \leftarrow [[e]_{2t}]_t$
of Epoch 2

Handoff Phase $[[e]_{2t}]_t$

Computation Phase : $[e]_{2t} = [a]_t \times [b]_t$
of Epoch 1 $[[e]_{2t}]_t \leftarrow [e]_{2t}$



Computation Phase : $[f]_t \leftarrow [[f]_t]_t$
of Epoch 2

Handoff Phase $[[f]_t]_t$

Computation Phase : $[f]_t = [c]_t + [d]_t$
of Epoch 1 $[[f]_t]_t \leftarrow [f]_t$

Input Phase: Clients send t-out-of-n shares of inputs to the first committee

Semi-honest Fluid-BGW

- ☑ Division of Work
- ☑ Maximal Fluidity

Output Phase

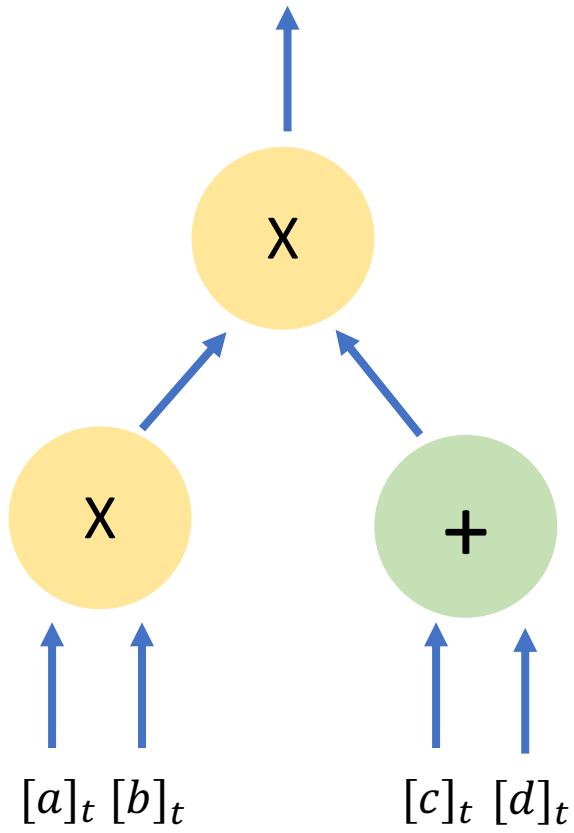
$g \leftarrow [g]_t$

Execution Stage

Computation Phase : $[e]_t \leftarrow [[e]_{2t}]_t$
of Epoch 2

Handoff Phase $[[e]_{2t}]_t$

Computation Phase : $[e]_{2t} = [a]_t \times [b]_t$
of Epoch 1 $[[e]_{2t}]_t \leftarrow [e]_{2t}$



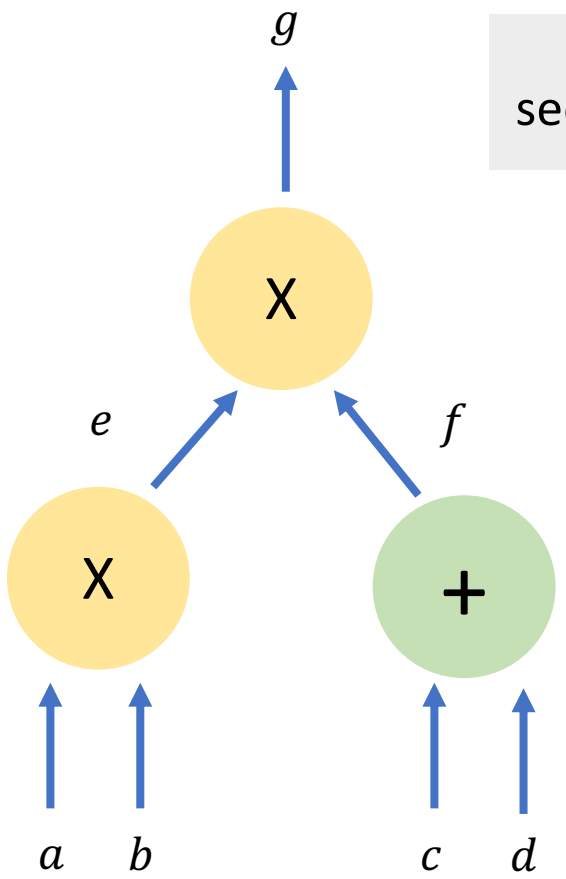
Computation Phase : $[f]_t \leftarrow [[f]_t]_t$
of Epoch 2

Handoff Phase $[[f]_t]_t$

Computation Phase : $[f]_t = [c]_t + [d]_t$
of Epoch 1 $[[f]_t]_t \leftarrow [f]_t$

Input Phase: Clients send t-out-of-n shares of inputs to the first committee

Additive Attack Paradigm [GIPST14]



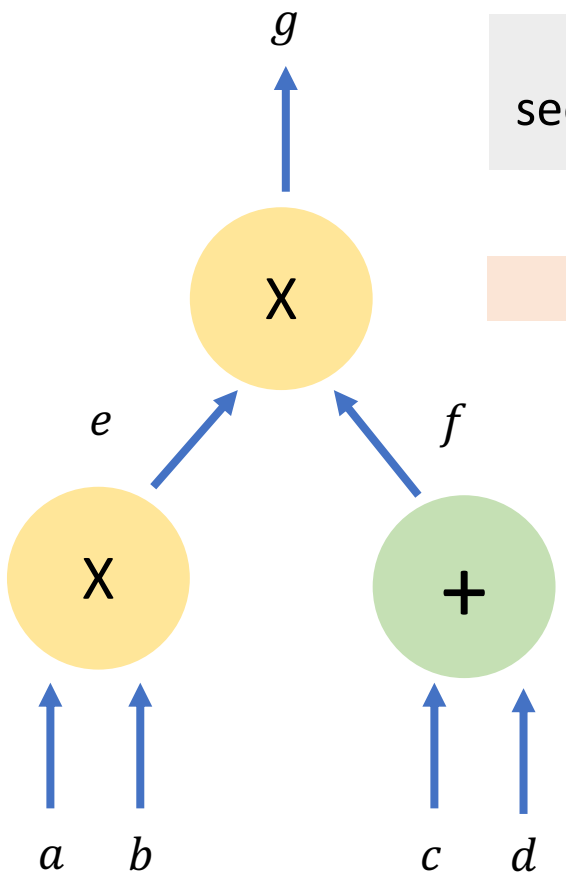
Most secret sharing based semi-honest protocols are secure against malicious adversaries up to additive attacks:

Fluid MPC Protocol (Security with Abort)

A compiler that that transforms “certain” semi-honest Fluid MPC protocols into maliciously secure protocols:

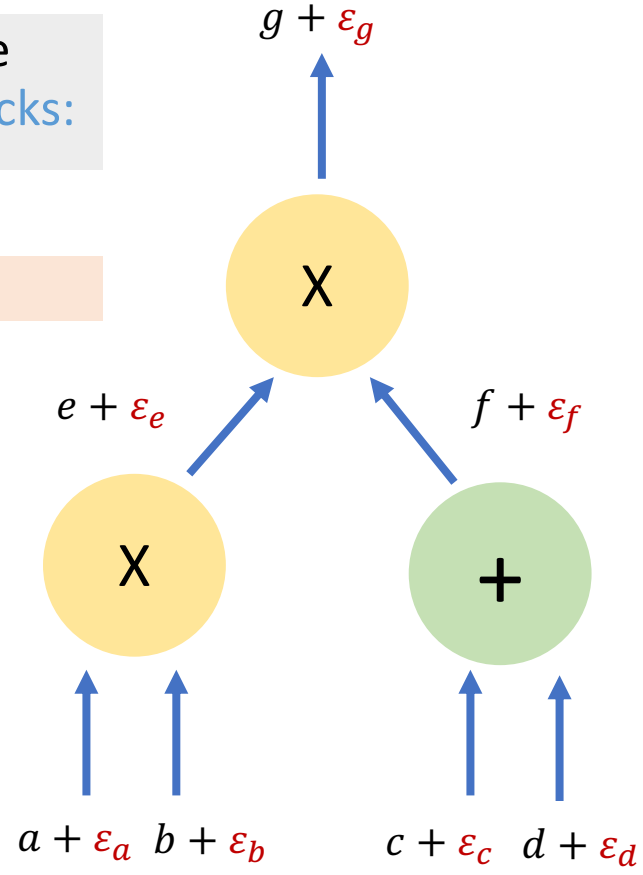
- security with abort
- $4 \times$ communication complexity
- Preserves fluidity

Additive Attack Paradigm [GIPST14]



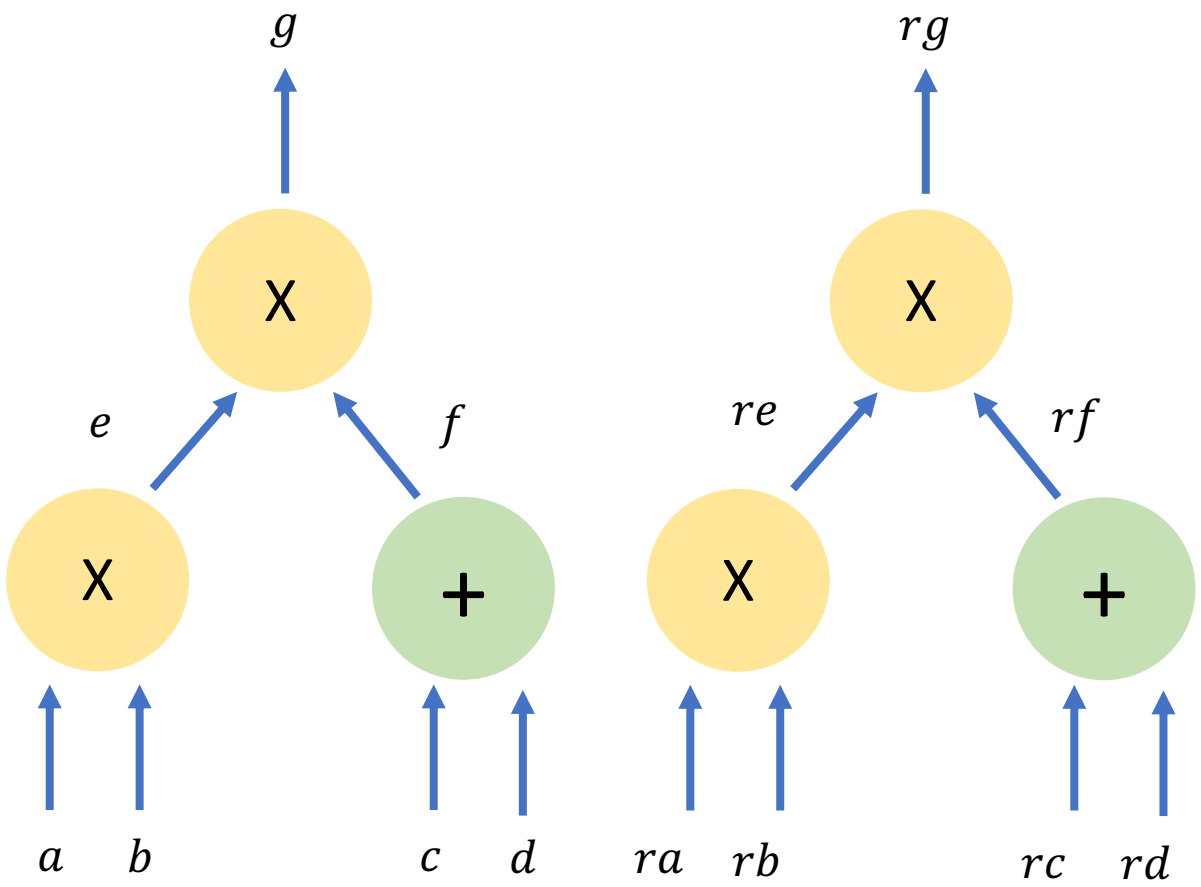
Most secret sharing based semi-honest protocols are secure against malicious adversaries up to additive attacks:

ε values are independent of the actual wire values



Efficient Maliciously Secure Protocols [DPSZ12,CGHIKLN18]

Modern efficient maliciously secure protocols rely on this additive attack paradigm.



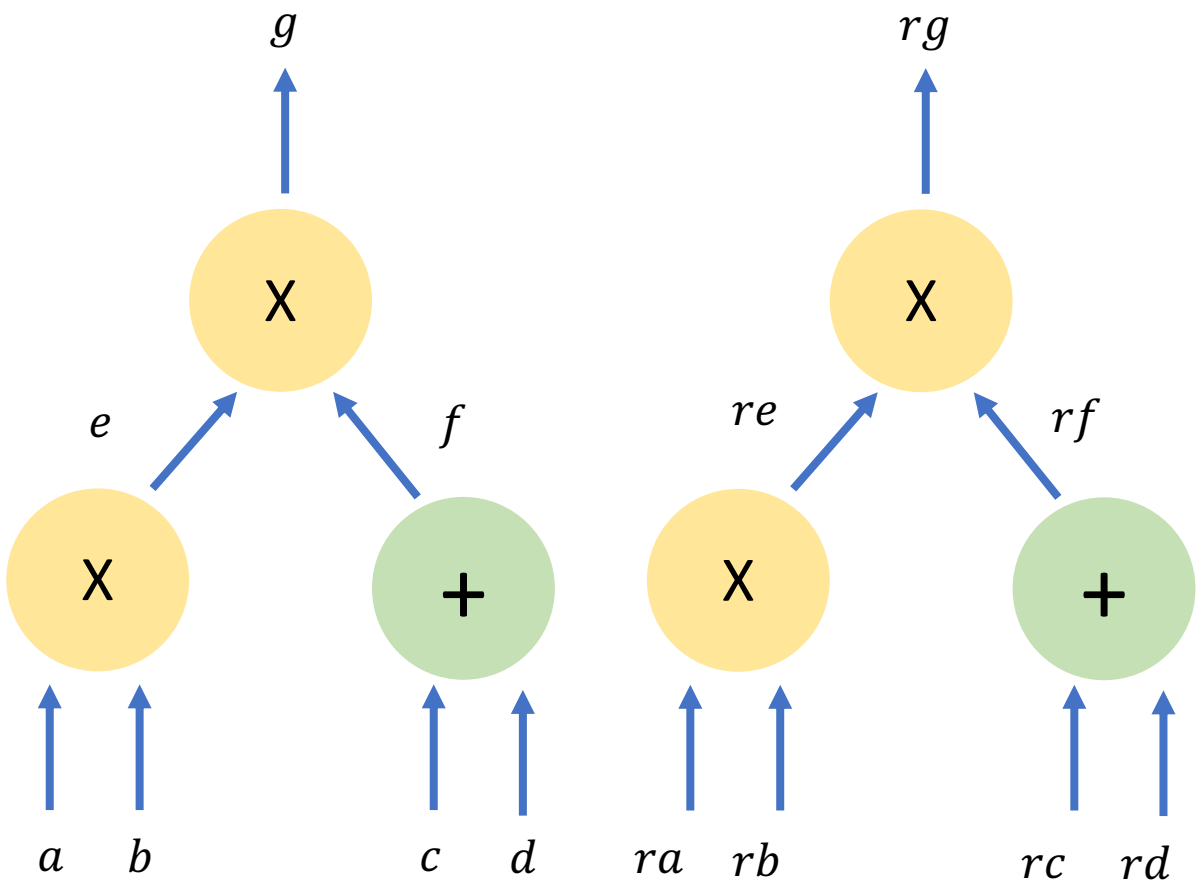
Semi-honest execution

Semi-honest execution

Dual execution: On actual inputs and randomized inputs.

Efficient Maliciously Secure Protocols [DPSZ12,CGHIKLN18]

Modern efficient maliciously secure protocols rely on this additive attack paradigm.



Semi-honest execution

Semi-honest execution

Dual execution: On actual inputs and randomized inputs.

Check for correctness by comparing a random linear combination of all the intermediate values at the end.

$$r(\alpha_1 a + \alpha_2 b + \cdots + \alpha_7 g)$$
$$=?=$$
$$\alpha_1 ra + \alpha_2 rb + \cdots + \alpha_7 rg$$

Maliciously secure Fluid MPC

Additive Attack Paradigm?

Semi-honest Fluid BGW  Maliciously secure Fluid MPC

Maliciously secure Fluid MPC

Additive Attack Paradigm?

Semi-honest Fluid BGW  Maliciously secure Fluid MPC

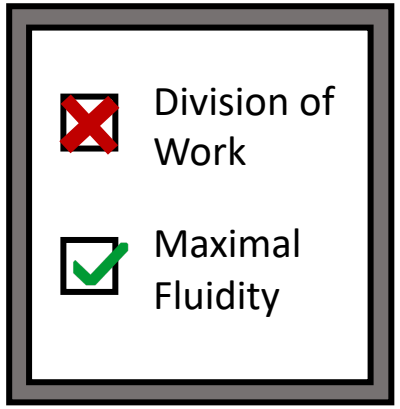
We Show: Additive Attack Paradigm extends to the Fluid MPC setting

Maliciously secure Fluid MPC

Can we use known techniques in the additive attack paradigm?

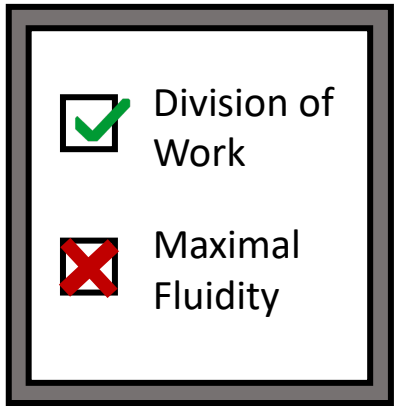
If the linear combination is computed at the end

All intermediate values must be passed along till the end of the protocol.

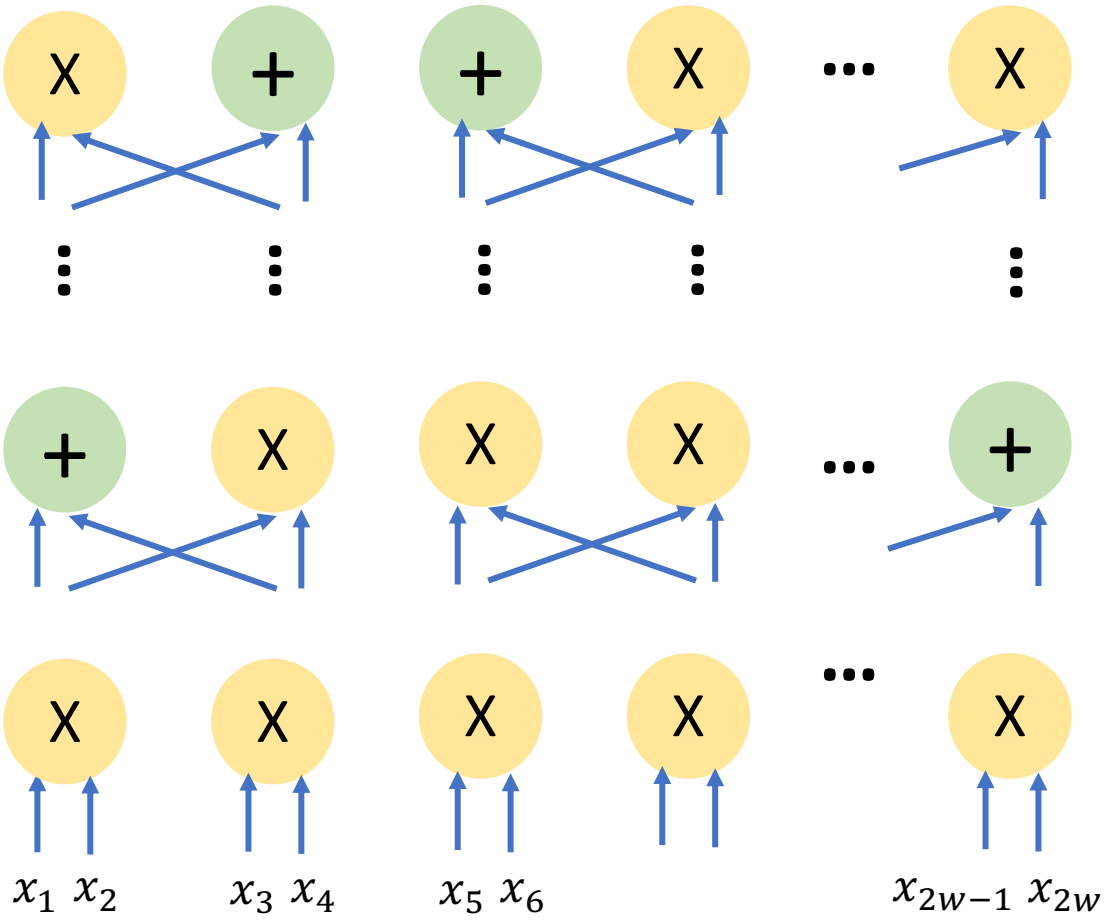


If the linear combination is computed incrementally layer-by-layer

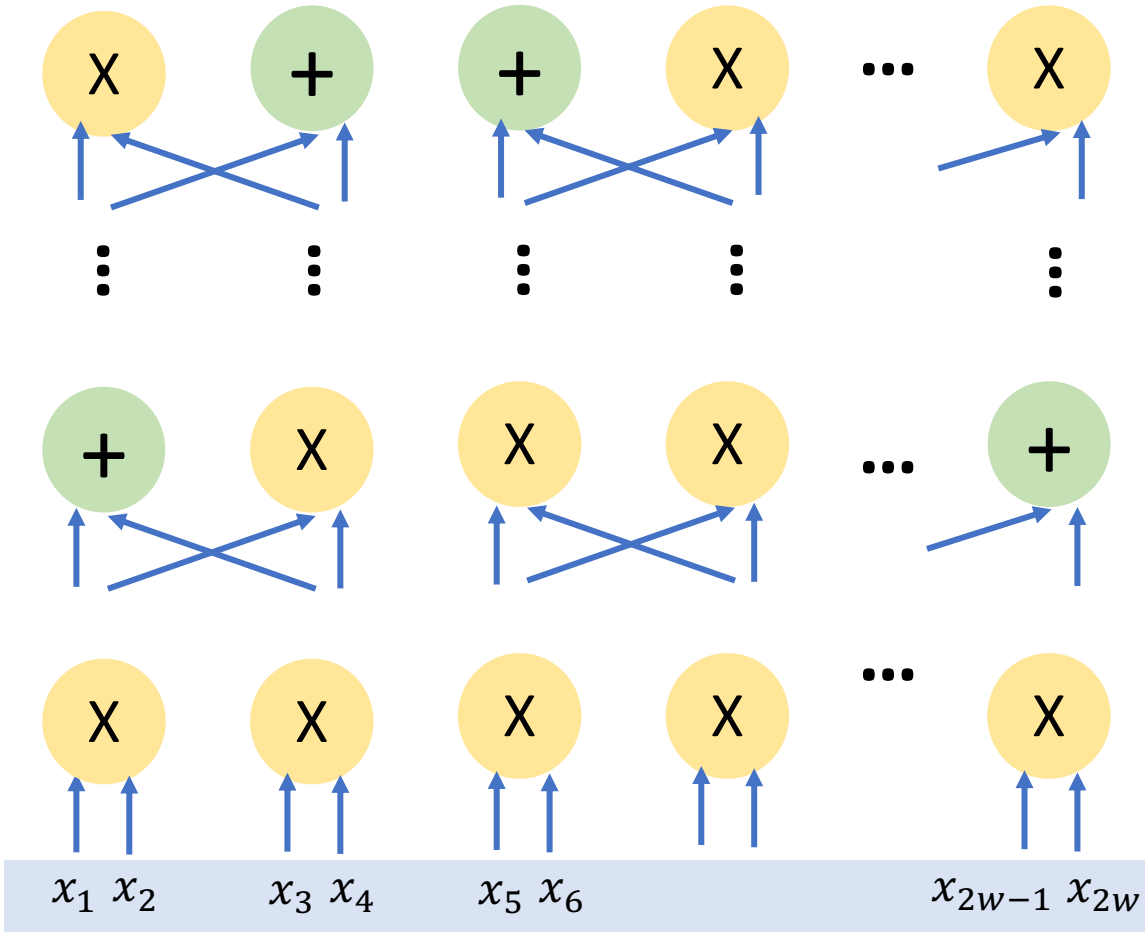
Random α values used in the linear combination will have to be generated on the fly, which may take many rounds.



Maliciously Secure Fluid MPC: Our Idea

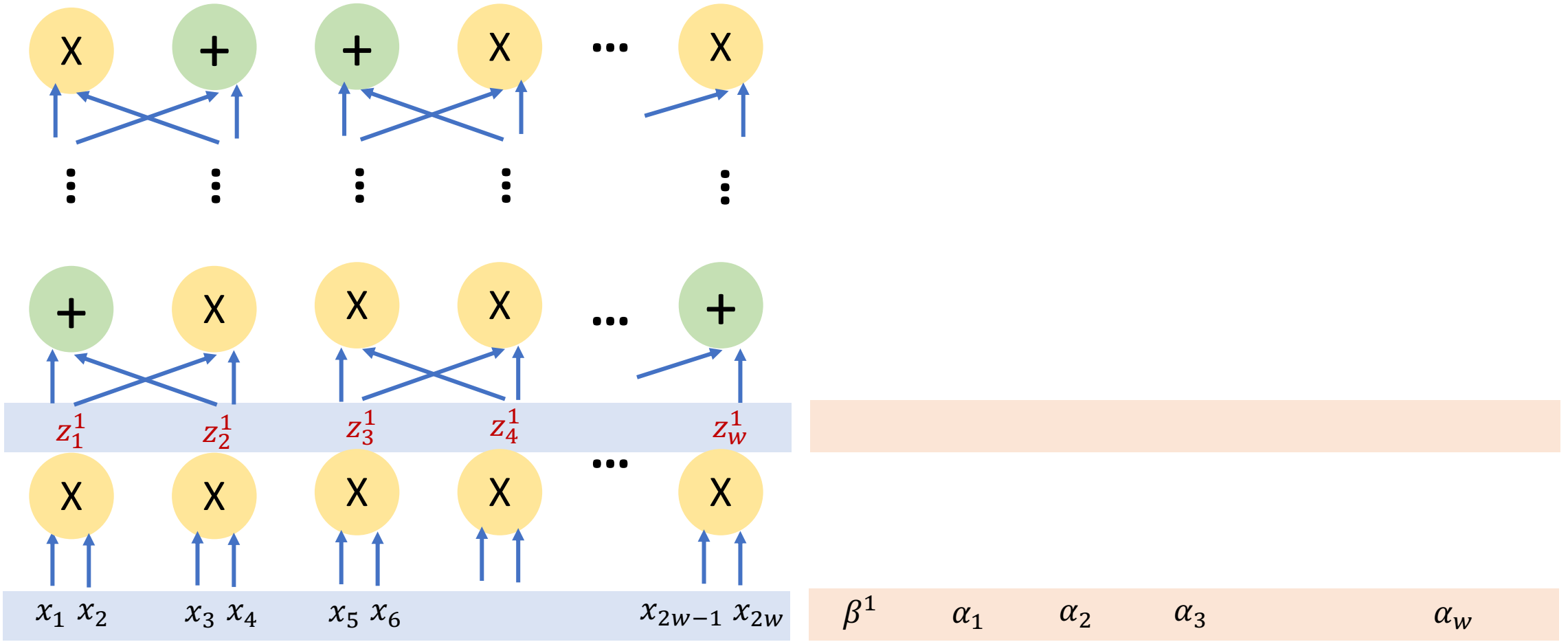


Maliciously Secure Fluid MPC: Our Idea

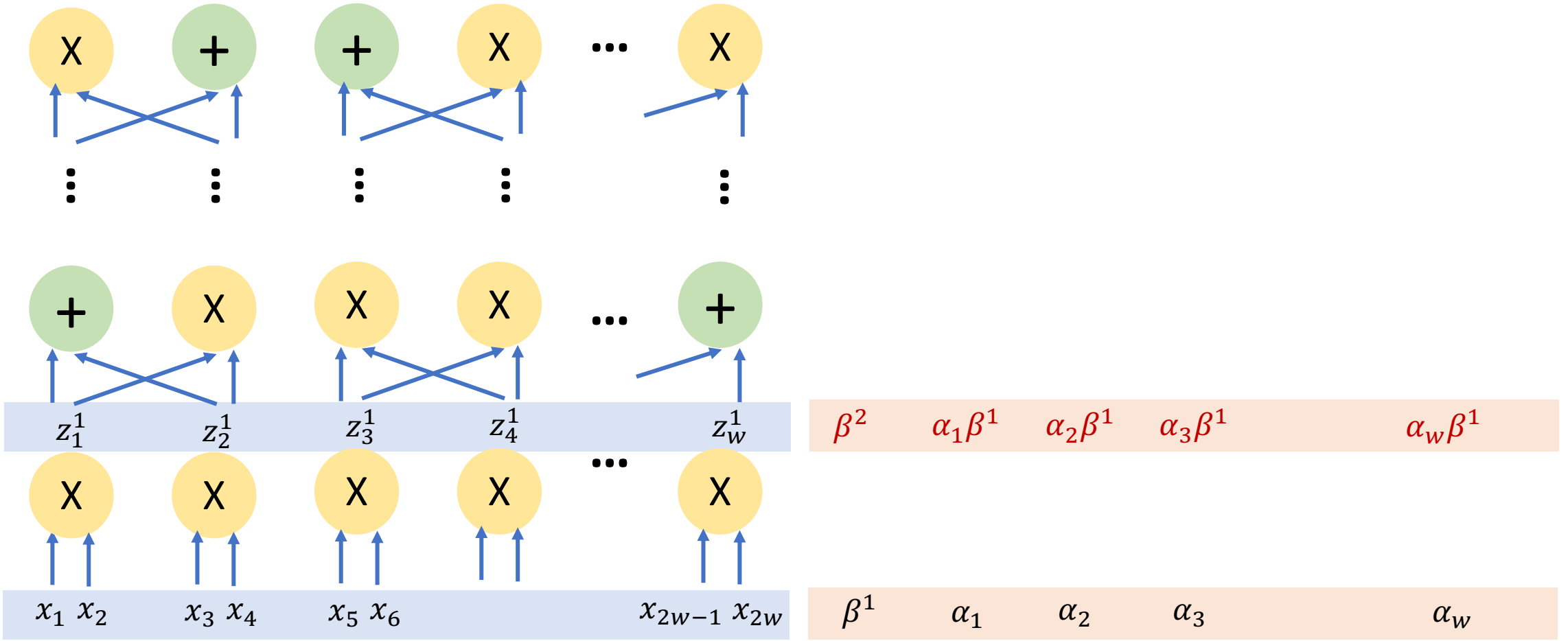


β^1 α_1 α_2 α_3 α_w

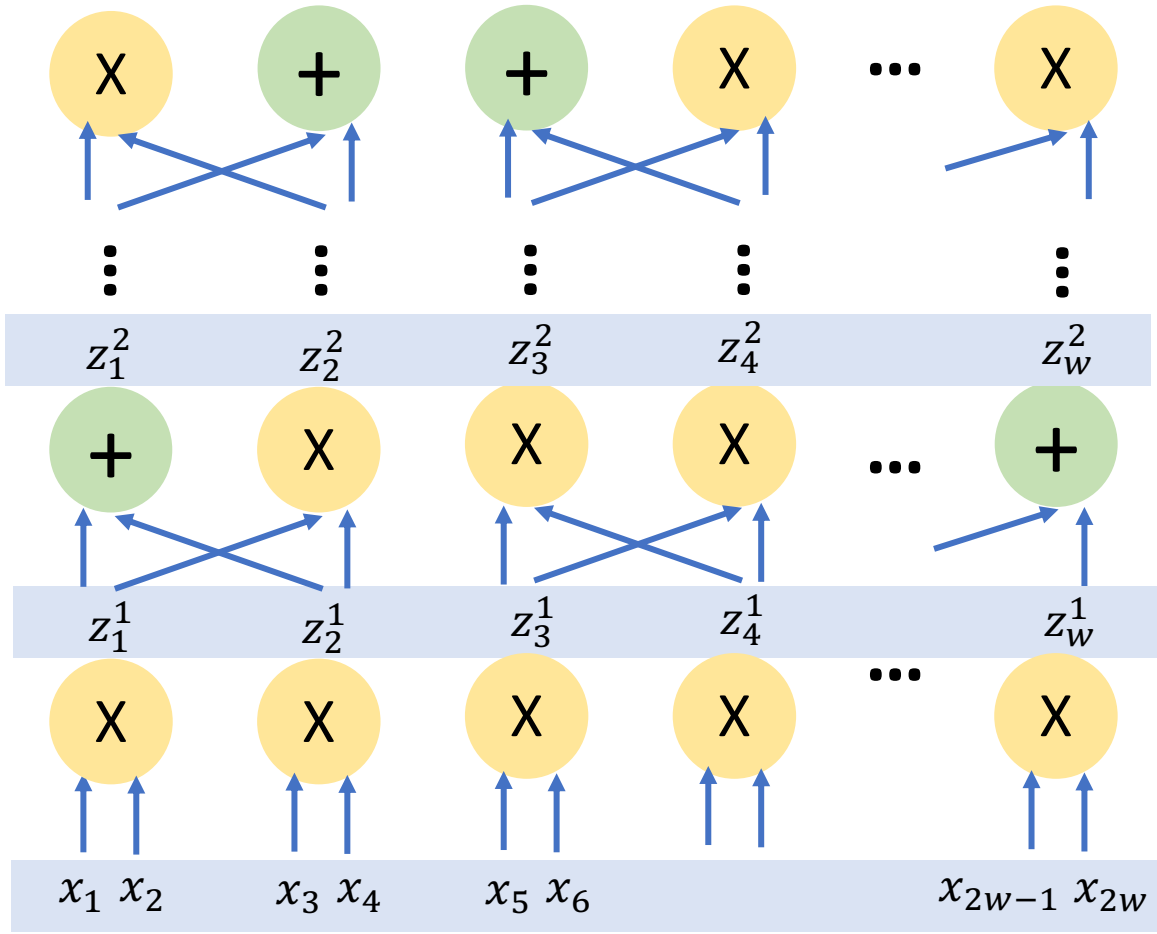
Maliciously Secure Fluid MPC: Our Idea



Maliciously Secure Fluid MPC: Our Idea



Maliciously Secure Fluid MPC: Our Idea

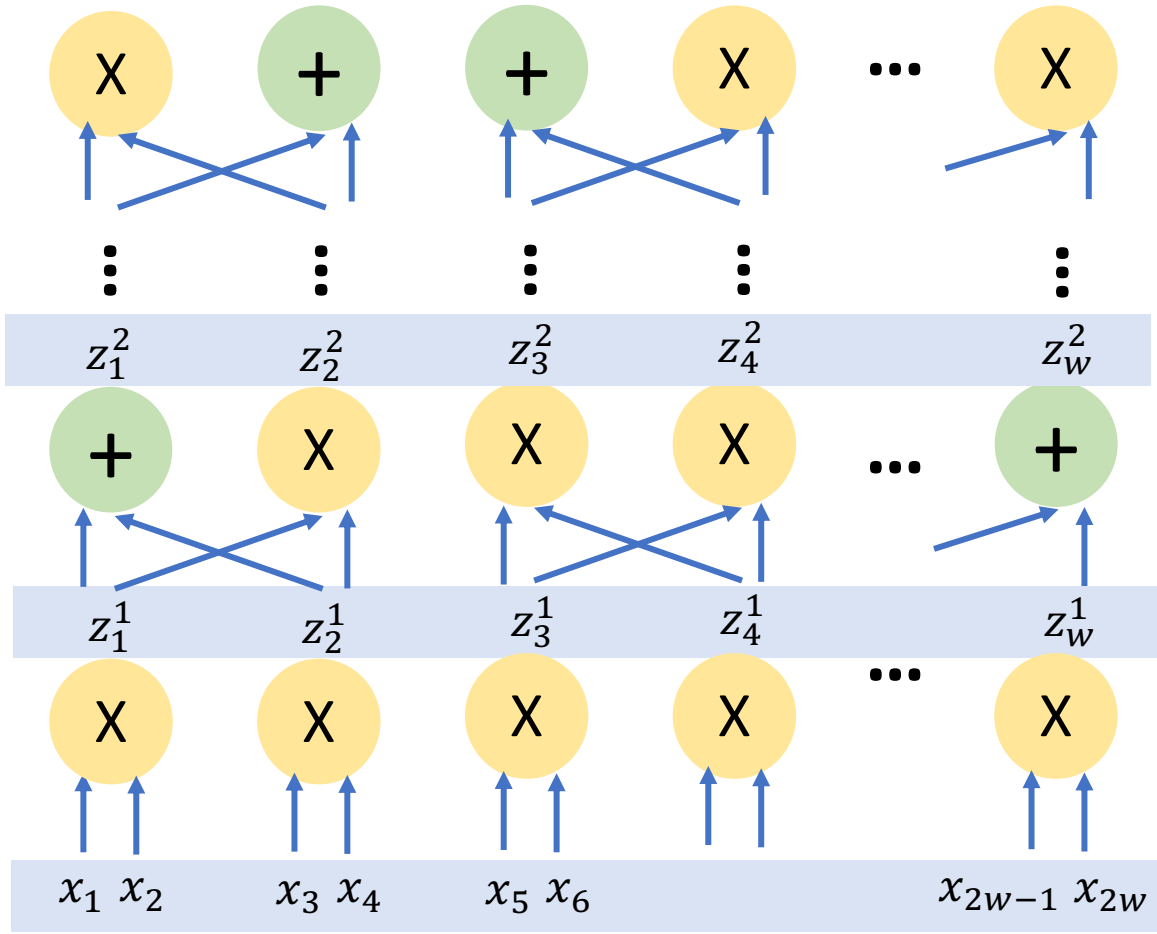


β^3 $\alpha_1\beta^2$ $\alpha_2\beta^2$ $\alpha_3\beta^2$ $\alpha_w\beta^2$

β^2 $\alpha_1\beta^1$ $\alpha_2\beta^1$ $\alpha_3\beta^1$ $\alpha_w\beta^1$

β^1 α_1 α_2 α_3 α_w

Maliciously Secure Fluid MPC: Our Idea



β^3 $\alpha_1\beta^2$ $\alpha_2\beta^2$ $\alpha_3\beta^2$ $\alpha_w\beta^2$

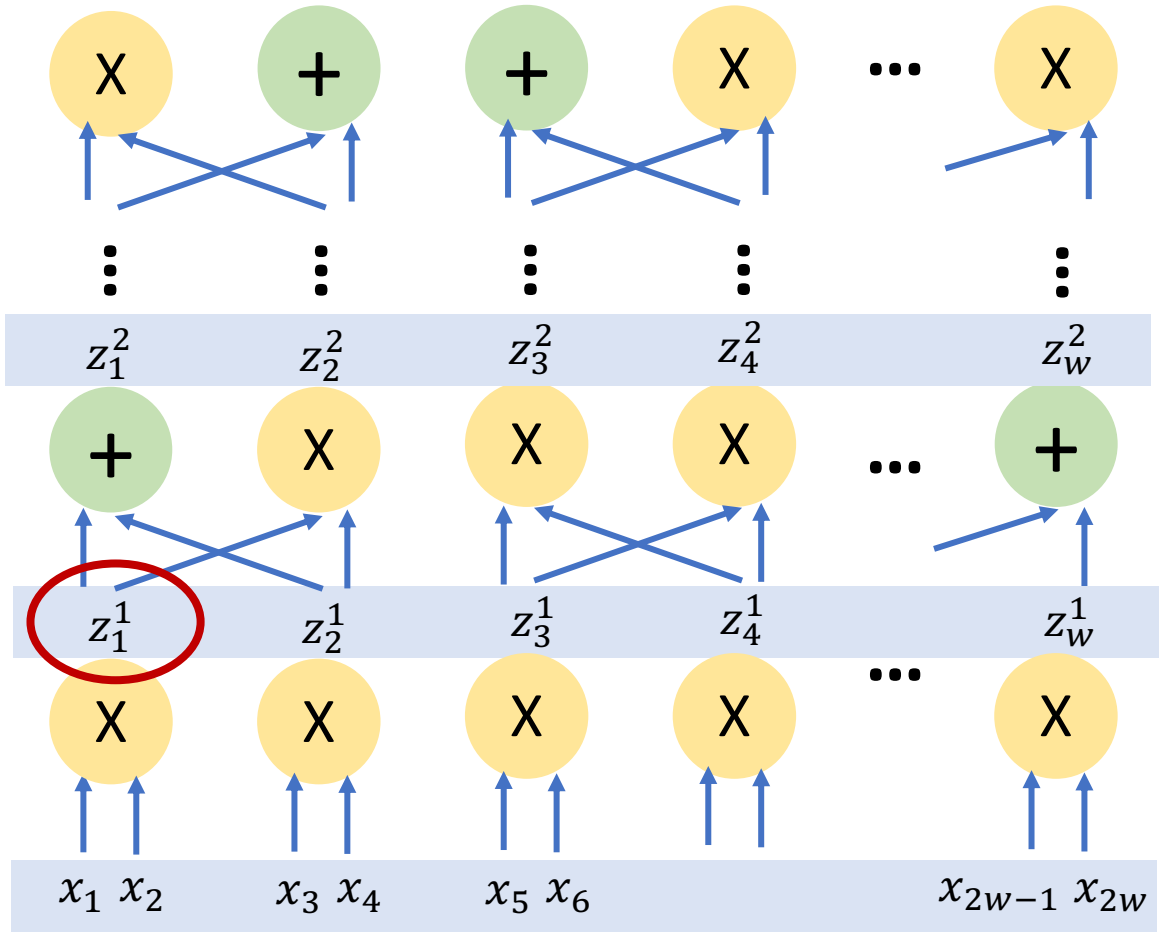
$$u_1 = u_0 + \alpha_1\beta^1 z_1^1 + \alpha_2\beta^1 z_2^1 + \cdots + \alpha_w\beta^1 z_w^1$$

β^2 $\alpha_1\beta^1$ $\alpha_2\beta^1$ $\alpha_3\beta^1$ $\alpha_w\beta^1$

$$u_0 = 0$$

β^1 α_1 α_2 α_3 α_w

Maliciously Secure Fluid MPC: Our Idea



β^3 $\alpha_1\beta^2$ $\alpha_2\beta^2$ $\alpha_3\beta^2$ $\alpha_w\beta^2$

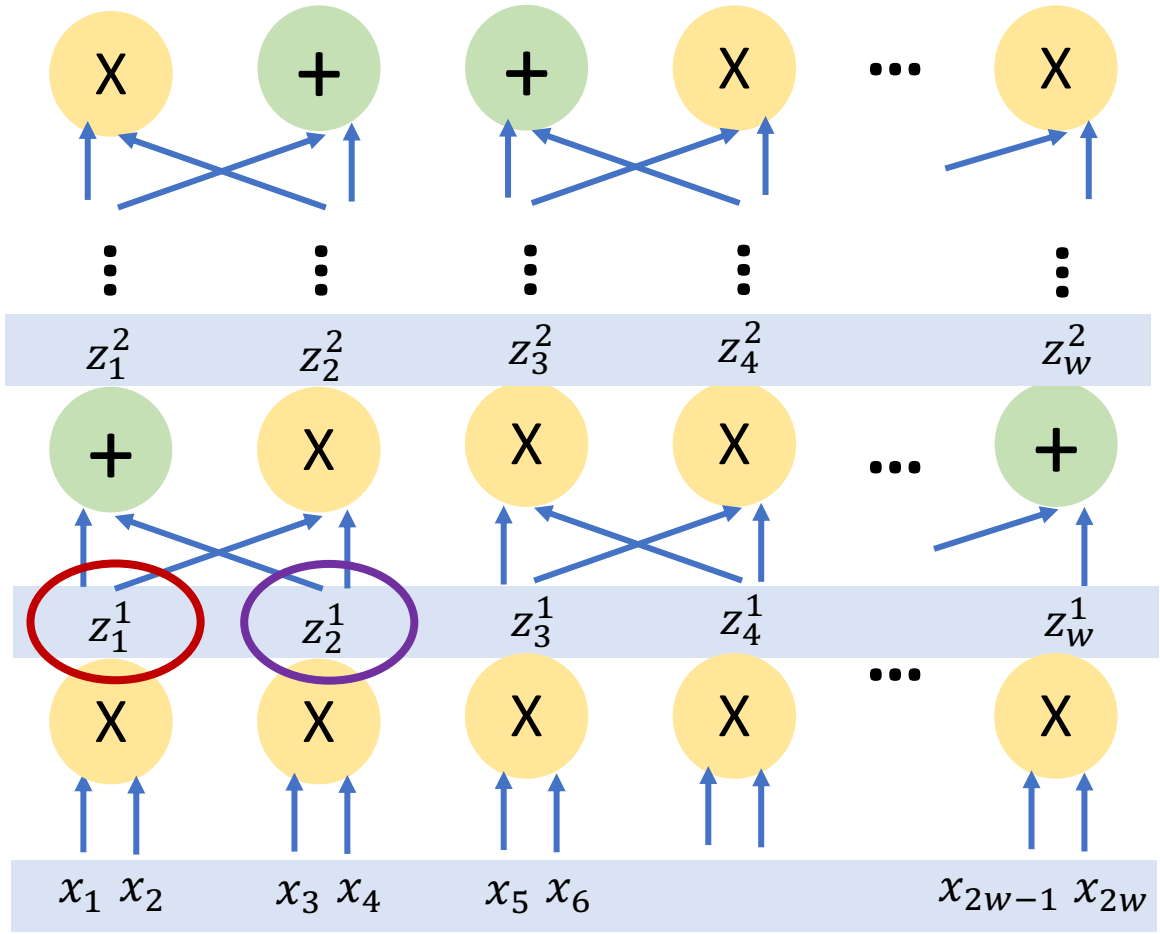
$$u_1 = u_0 + \alpha_1\beta^1 z_1^1 + \alpha_2\beta^1 z_2^1 + \cdots + \alpha_w\beta^1 z_w^1$$

β^2 $\alpha_1\beta^1$ $\alpha_2\beta^1$ $\alpha_3\beta^1$ $\alpha_w\beta^1$

$$u_0 = 0$$

β^1 α_1 α_2 α_3 α_w

Maliciously Secure Fluid MPC: Our Idea



β^3 $\alpha_1\beta^2$ $\alpha_2\beta^2$ $\alpha_3\beta^2$ $\alpha_w\beta^2$

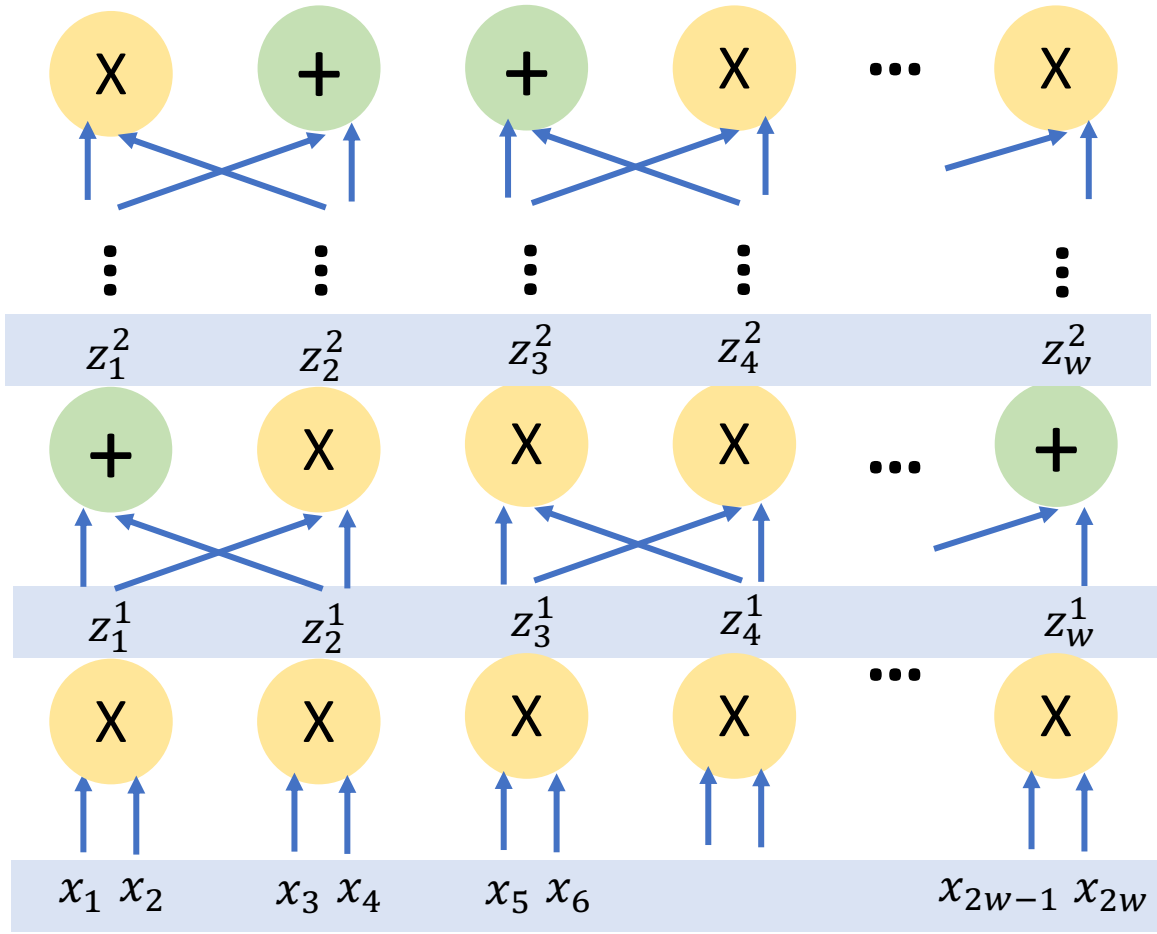
$$u_1 = u_0 + \alpha_1\beta^1 z_1^1 + \alpha_2\beta^1 z_2^1 + \cdots + \alpha_w\beta^1 z_w^1$$

β^2 $\alpha_1\beta^1$ $\alpha_2\beta^1$ $\alpha_3\beta^1$ $\alpha_w\beta^1$

$$u_0 = 0$$

β^1 α_1 α_2 α_3 α_w

Maliciously Secure Fluid MPC: Our Idea



β^3 $\alpha_1\beta^2$ $\alpha_2\beta^2$ $\alpha_3\beta^2$ $\alpha_w\beta^2$

Epoch 2

$$u_1 = u_0 + \alpha_1\beta^1 z_1^1 + \alpha_2\beta^1 z_2^1 + \cdots + \alpha_w\beta^1 z_w^1$$

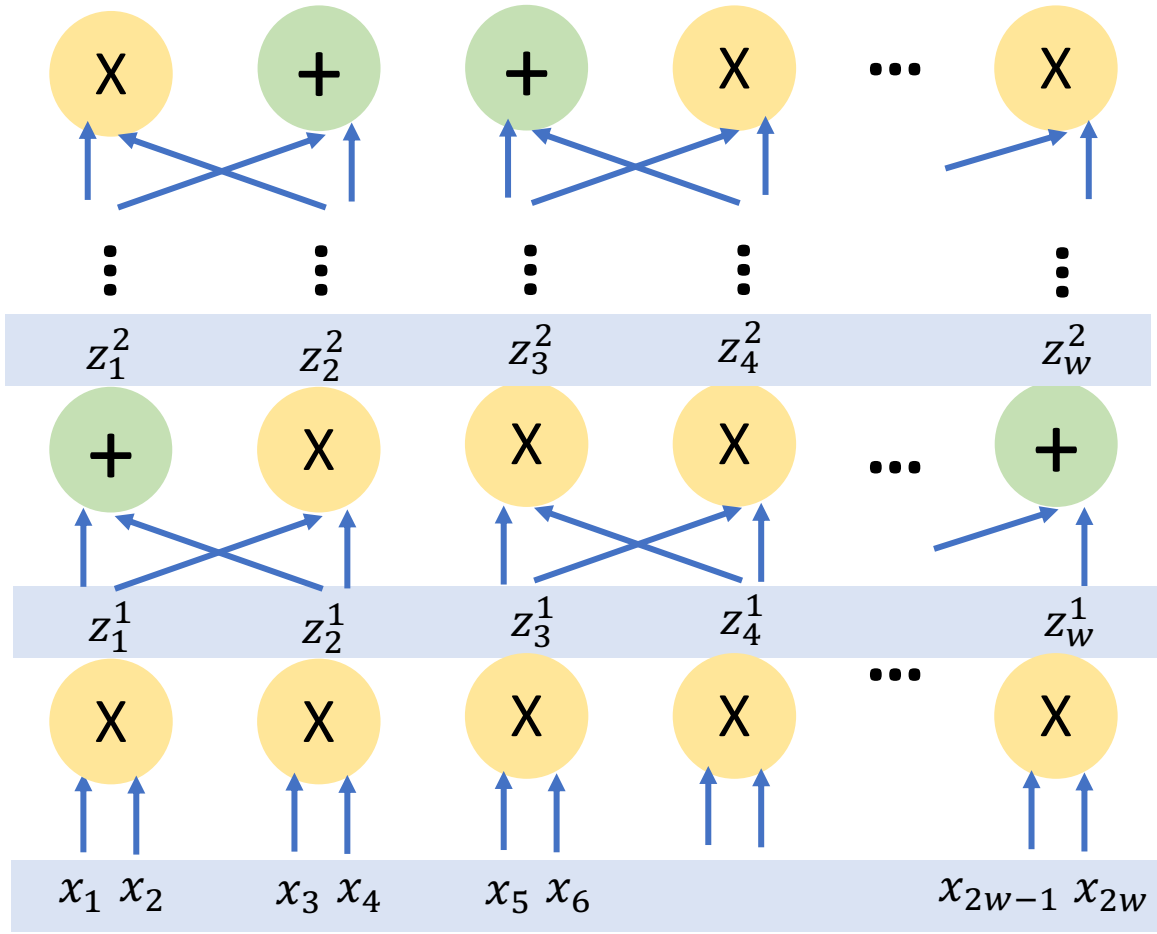
β^2 $\alpha_1\beta^1$ $\alpha_2\beta^1$ $\alpha_3\beta^1$ $\alpha_w\beta^1$

Epoch 1

$$u_0 = 0$$

β^1 α_1 α_2 α_3 α_w

Maliciously Secure Fluid MPC: Our Idea



- ☑ Division of Work
- ☑ Maximal Fluidity

β^3 $\alpha_1\beta^2$ $\alpha_2\beta^2$ $\alpha_3\beta^2$ $\alpha_w\beta^2$

Epoch 2

$$u_1 = u_0 + \alpha_1\beta^1 z_1^1 + \alpha_2\beta^1 z_2^1 + \cdots + \alpha_w\beta^1 z_w^1$$

β^2 $\alpha_1\beta^1$ $\alpha_2\beta^1$ $\alpha_3\beta^1$ $\alpha_w\beta^1$

Epoch 1

$$u_0 = 0$$

β^1 α_1 α_2 α_3 α_w

Fluid MPC Protocol (Guaranteed Output Delivery) [DGL23]

1. Perfectly secure maximally fluid protocol for $t < n/3$ corruptions in each committee.
2. Information-theoretic, maximally-fluid MPC with guaranteed output delivery is impossible with $t > n/3$ corruptions in each committee.
3. A computationally secure maximally fluid protocol for $t < n/2$ corruptions in each committee.

Challenges and Main Idea

- Player elimination doesn’t work
- Regular secret sharing is not enough, need verifiable secret sharing
- VSS with maximal fluidity:
  - 1 round VSS is impossible [PCRR09]
  - Known VSS protocols are not stateless

Summary [CGGJK21, DGL23]

Fluid MPC: A formal model for MPC with dynamic participants.

Construct semi-honest and malicious Fluid MPC protocols that have maximum fluidity.

Thank You!