Zero-Knowledge Proofs from Multiparty Computation: Recent Advances

Matthieu Rivain

WRACH 2023

Jun 14, 2023, Roscoff

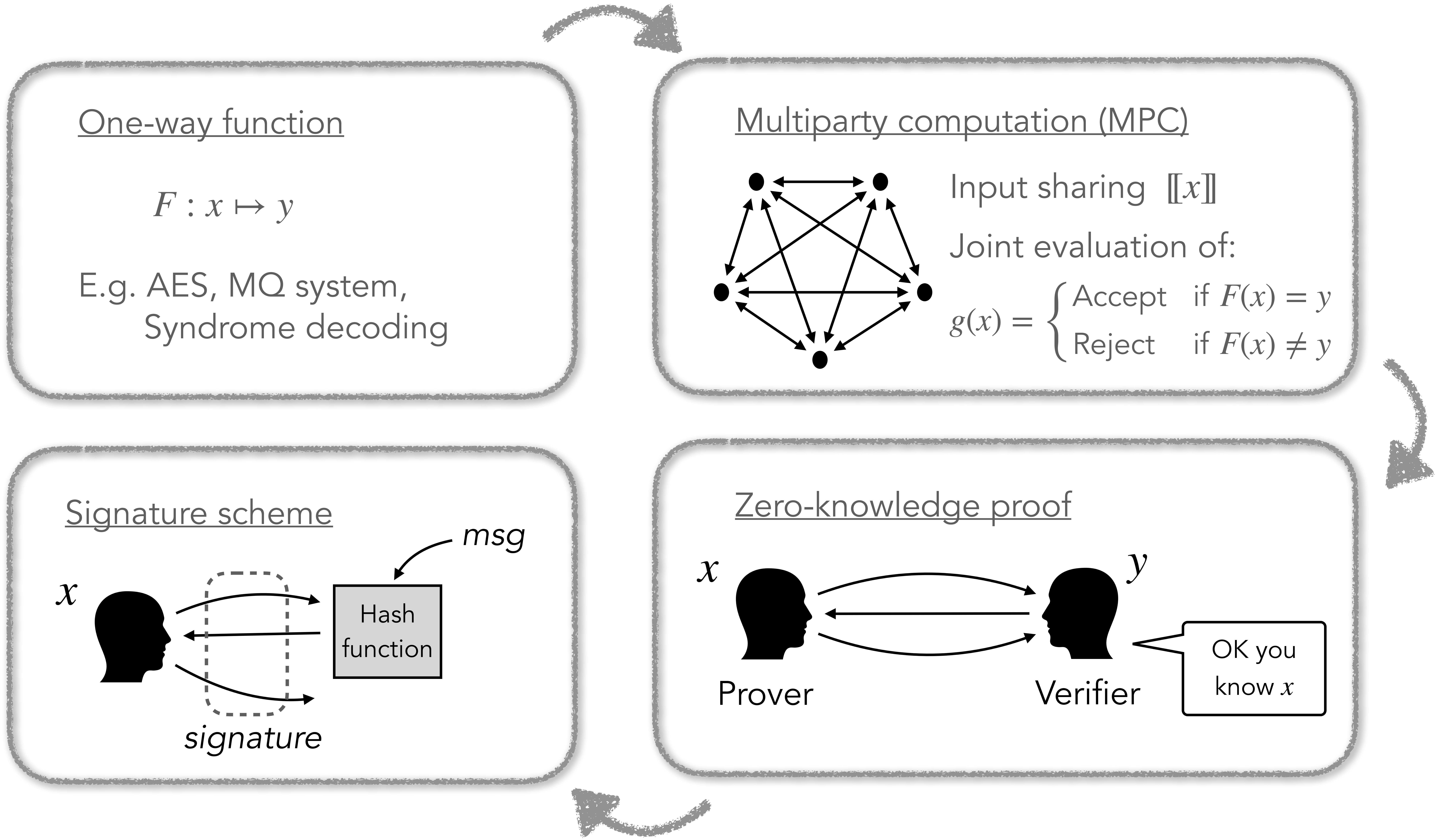
CryptoExperts

WE INNOVATE TO SECURE YOUR BUSINESS

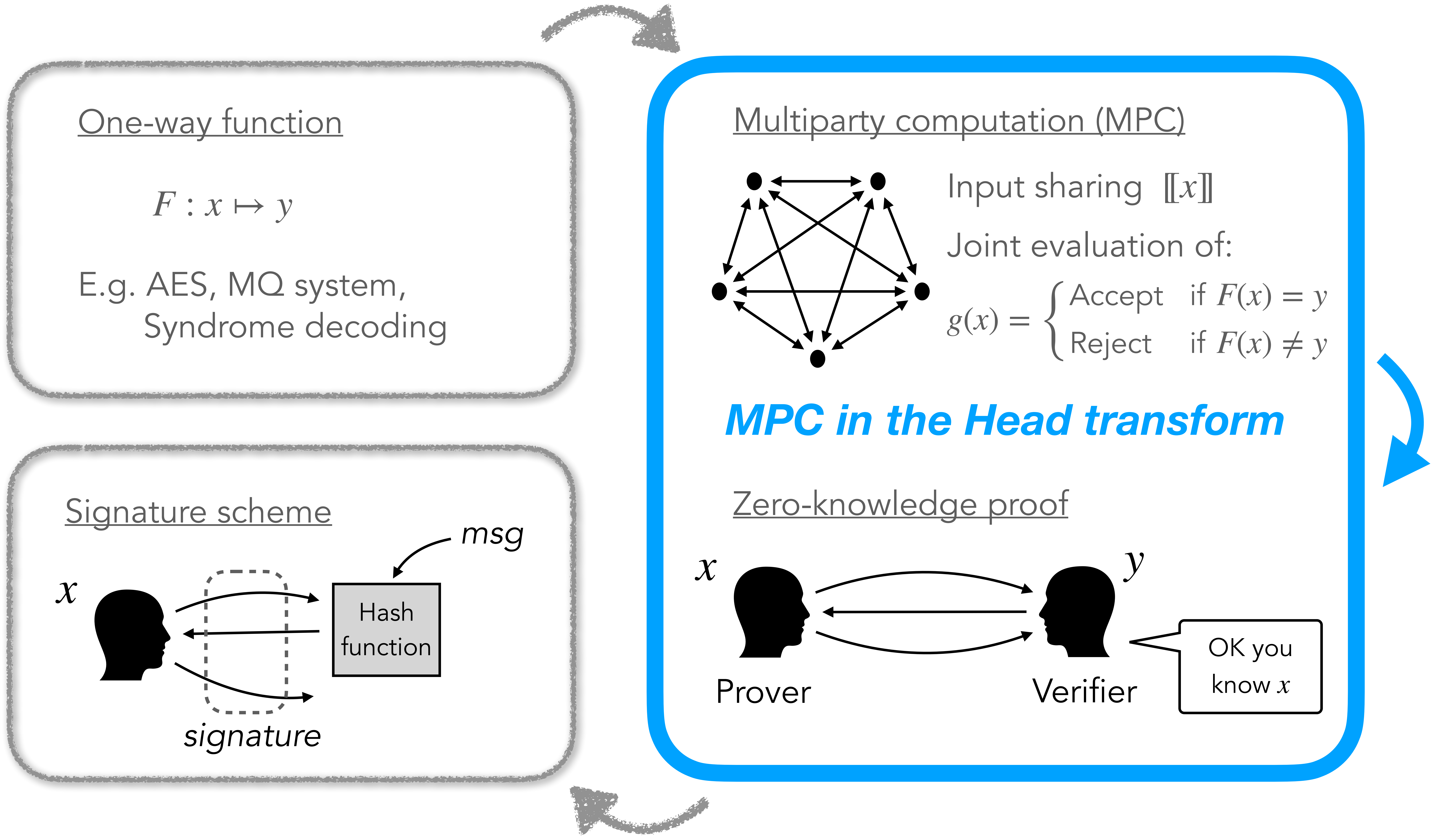
Introduction

MPC in the Head

- **[IKOS07]** Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, Amit Sahai: "Zero-knowledge from secure multiparty computation" (STOC 2007)
- Turn an MPC protocol into a zero knowledge proof of knowledge
- **Generic**: can be apply to any cryptographic problem
- Convenient to build (candidate) **post-quantum signature** schemes
- **Picnic**: submission to NIST (2017)
- Recent NIST call (01/06/2023): 7 MPCitH schemes / 50 submissions



One-way function
F : x ↦ y
E.g. AES, MQ system,
Syndrome decoding
Multiparty computation (MPC)
Input sharing ⟦x⟧
Joint evaluation of:
g(x) = Accept if F(x) = y
Reject if F(x) ≠ y
Zero-knowledge proof
x
y
Prover
Verifier
OK you know x
Signature scheme
x
msg
Hash function
signature

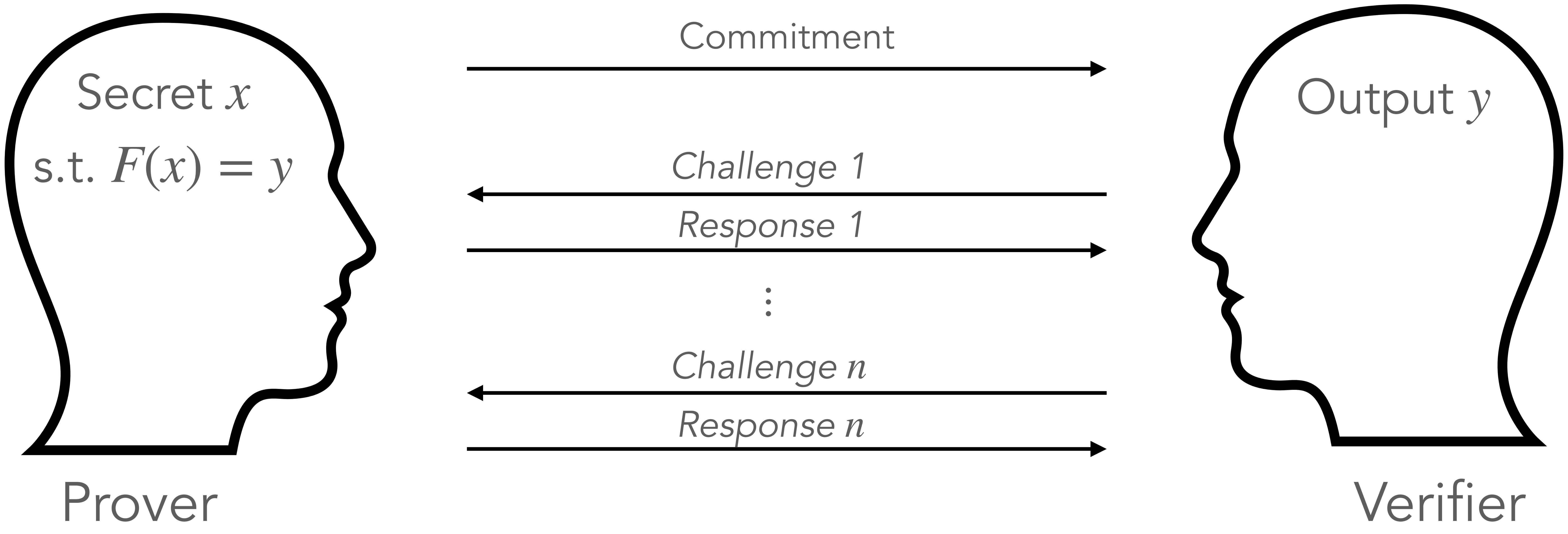

One-way function
$F : x \mapsto y$
E.g. AES, MQ system, Syndrome decoding
Signature scheme
x
msg
Hash function
signature
Multiparty computation (MPC)
Input sharing $[\![x]\!]$
Joint evaluation of:
$g(x) = \begin{cases} \text{Accept} & \text{if } F(x) = y \\ \text{Reject} & \text{if } F(x) \neq y \end{cases}$
MPC in the Head transform
Zero-knowledge proof
x
y
Prover
Verifier
OK you know x

Background: Additive secret sharing

$$[\![x]\!] = ([\![x]\!]_1, \ldots, [\![x]\!]_N) \quad \text{s.t.} \quad x = \sum_{i=1}^{N} [\![x]\!]_i$$

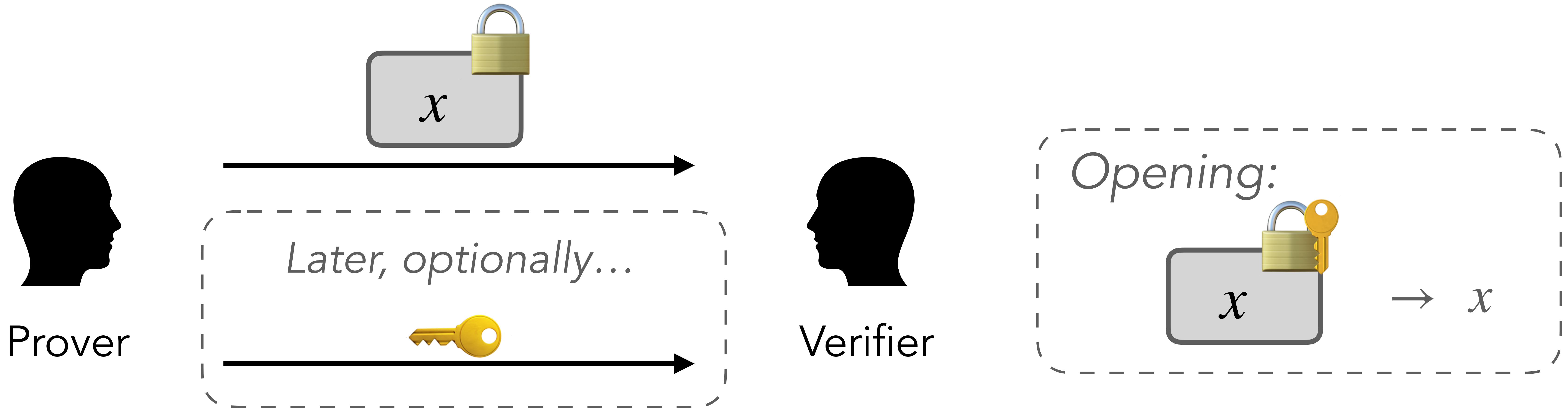
Any set of $N-1$ *shares is random & independent of* x

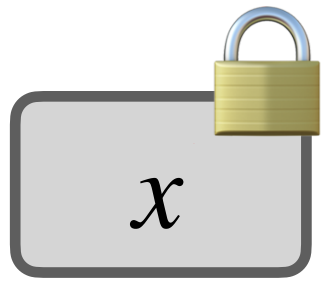
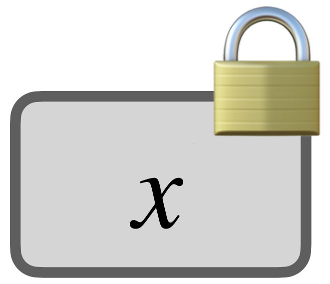
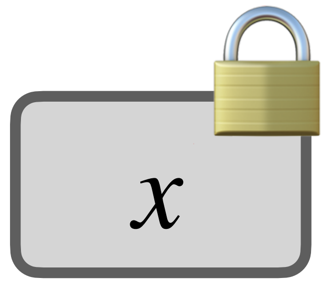
Background: Proof of knowledge



- **Completeness:** Pr[verif ✓ | honest prover] = 1
- **Soundness:** Pr[verif ✓ | malicious prover] $\leq \varepsilon$ (e.g. 2^{-128})
- **Zero-knowledge:** verifier learns nothing on x

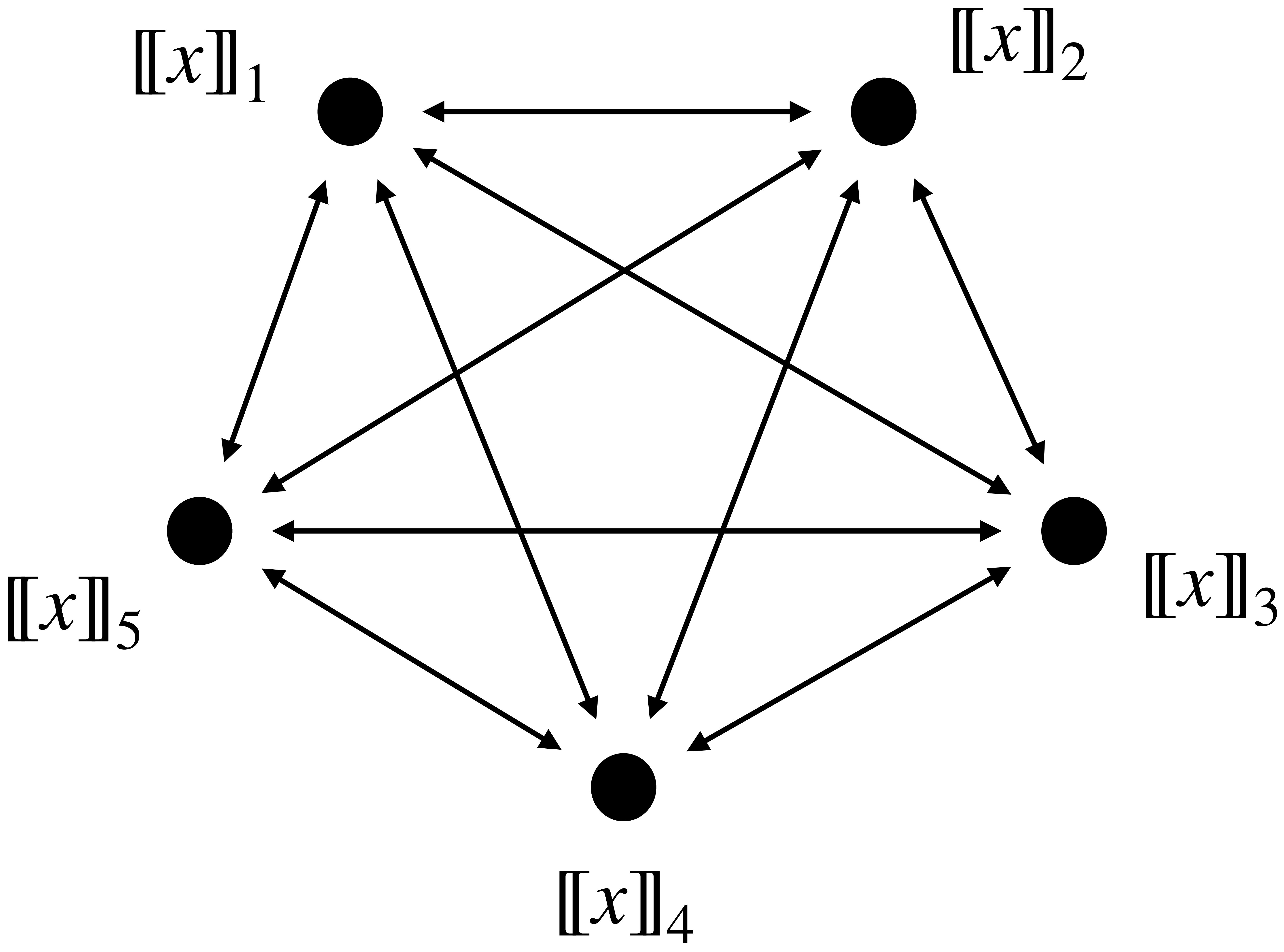
Background: Commitment scheme



- **Binding:** no way  can be opened to $x' \neq x$
- **Hiding:**  does not reveal information about x (without )
- **Hash commitment:**  $:= \text{Hash}(x \mid\mid \rho)$ with $\rho \leftarrow \$$  $:= (x, \rho)$

MPCitH: general principle

MPC model

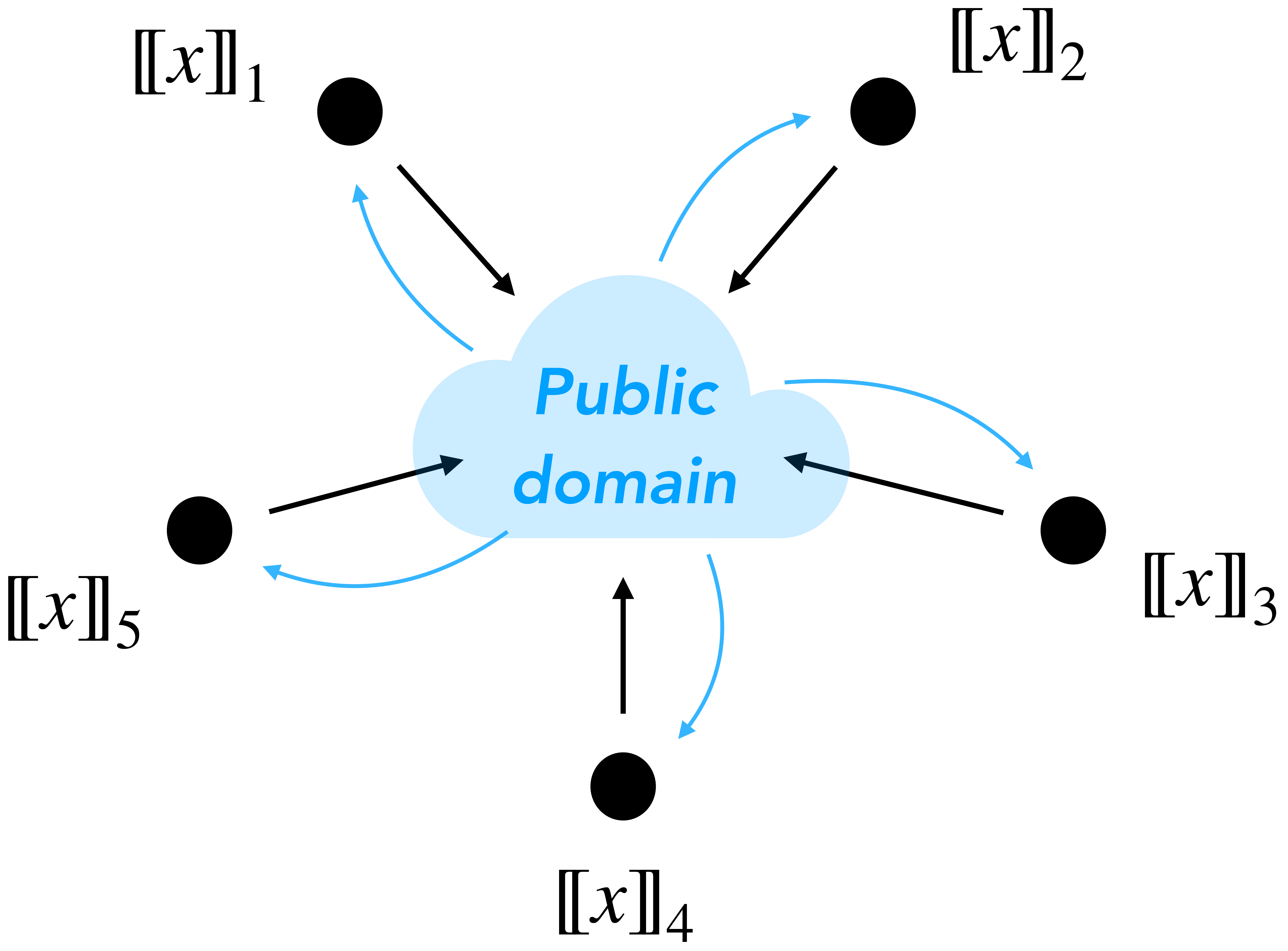


- **Jointly compute**

$$g(x) = \begin{cases} \text{Accept} & \text{if } F(x) = y \\ \text{Reject} & \text{if } F(x) \neq y \end{cases}$$

- $(N-1)$ **private:** the views of any $N-1$ parties provide no information on x
- **Semi-honest model:** assuming that the parties follow the steps of the protocol

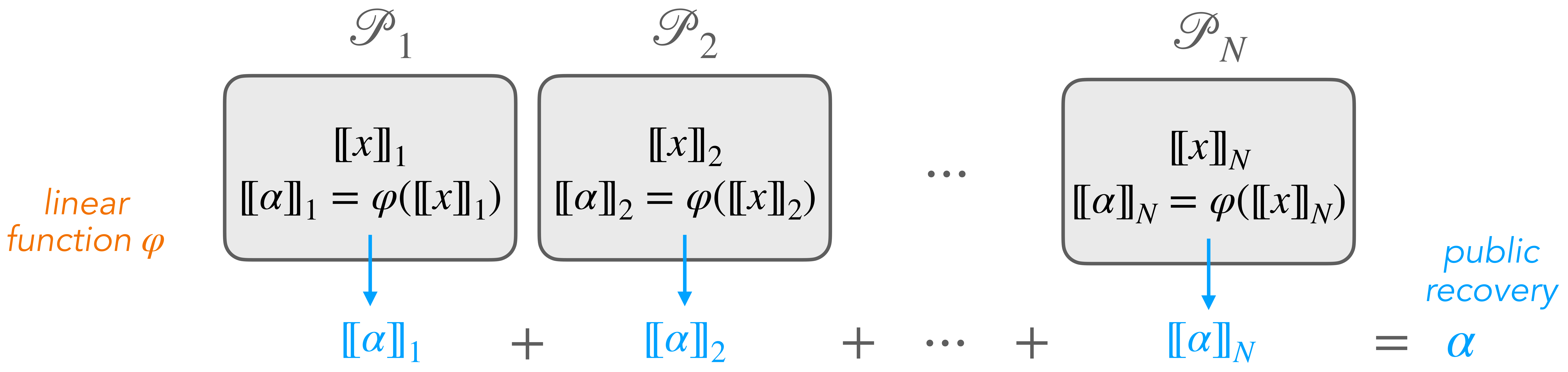
MPC model



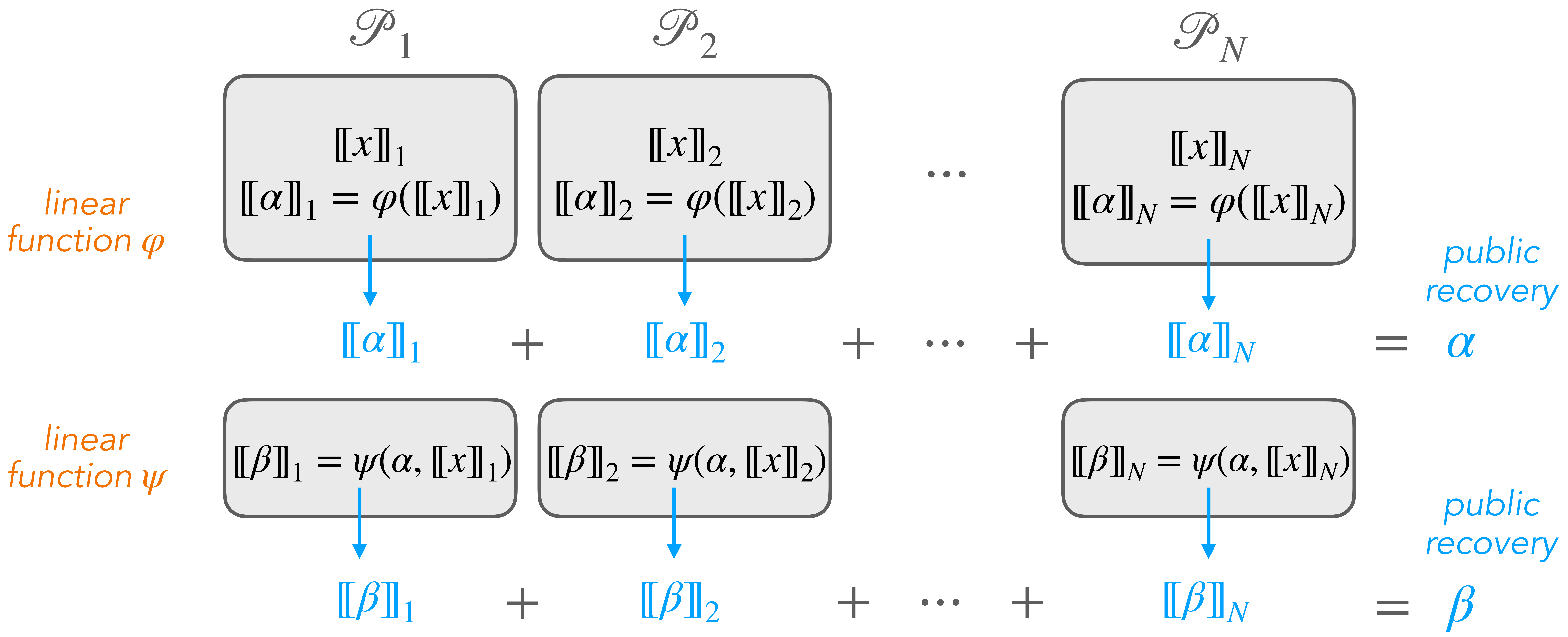
- **Jointly compute**

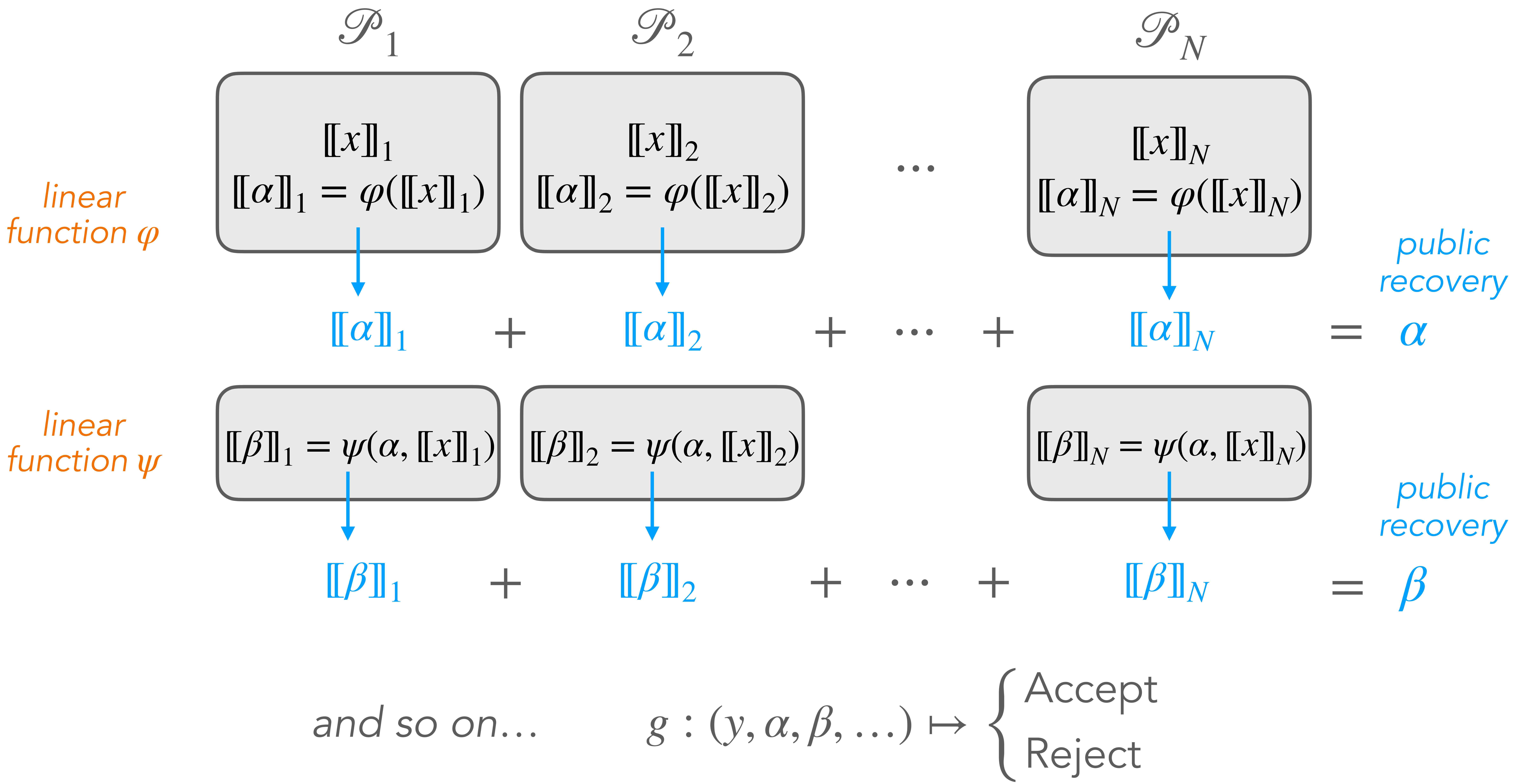
$$g(x) = \begin{cases} \text{Accept} & \text{if } F(x) = y \\ \text{Reject} & \text{if } F(x) \neq y \end{cases}$$

- $(N-1)$ **private:** the views of any $N-1$ parties provide no information on x
- **Semi-honest model:** assuming that the parties follow the steps of the protocol
- ***Broadcast model***
  - *Parties locally compute on their shares* $[\![x]\!] \mapsto [\![\alpha]\!]$
  - *Parties broadcast* $[\![\alpha]\!]$ *and recompute* α
  - *Parties start again (now knowing* α*)*



$\mathcal{P}_1$
$\mathcal{P}_2$
$\mathcal{P}_N$
$[\![x]\!]_1$
$[\![\alpha]\!]_1 = \varphi([\![x]\!]_1)$
$[\![x]\!]_2$
$[\![\alpha]\!]_2 = \varphi([\![x]\!]_2)$
$\cdots$
$[\![x]\!]_N$
$[\![\alpha]\!]_N = \varphi([\![x]\!]_N)$
linear function φ
public recovery
$[\![\alpha]\!]_1 + [\![\alpha]\!]_2 + \cdots + [\![\alpha]\!]_N = \alpha$


$\mathscr{P}_1$
$\mathscr{P}_2$
$\mathscr{P}_N$
$[\![x]\!]_1$
$[\![\alpha]\!]_1 = \varphi([\![x]\!]_1)$
$[\![x]\!]_2$
$[\![\alpha]\!]_2 = \varphi([\![x]\!]_2)$
$\cdots$
$[\![x]\!]_N$
$[\![\alpha]\!]_N = \varphi([\![x]\!]_N)$
linear function φ
$[\![\alpha]\!]_1 + [\![\alpha]\!]_2 + \cdots + [\![\alpha]\!]_N = \alpha$
public recovery
linear function ψ
$[\![\beta]\!]_1 = \psi(\alpha, [\![x]\!]_1)$
$[\![\beta]\!]_2 = \psi(\alpha, [\![x]\!]_2)$
$[\![\beta]\!]_N = \psi(\alpha, [\![x]\!]_N)$
$[\![\beta]\!]_1 + [\![\beta]\!]_2 + \cdots + [\![\beta]\!]_N = \beta$
public recovery


$\mathscr{P}_1$
$\mathscr{P}_2$
$\mathscr{P}_N$
linear function φ
$[\![x]\!]_1$
$[\![\alpha]\!]_1 = \varphi([\![x]\!]_1)$
$[\![x]\!]_2$
$[\![\alpha]\!]_2 = \varphi([\![x]\!]_2)$
$\cdots$
$[\![x]\!]_N$
$[\![\alpha]\!]_N = \varphi([\![x]\!]_N)$
public recovery
$[\![\alpha]\!]_1 + [\![\alpha]\!]_2 + \cdots + [\![\alpha]\!]_N = \alpha$
linear function ψ
$[\![\beta]\!]_1 = \psi(\alpha, [\![x]\!]_1)$
$[\![\beta]\!]_2 = \psi(\alpha, [\![x]\!]_2)$
$[\![\beta]\!]_N = \psi(\alpha, [\![x]\!]_N)$
public recovery
$[\![\beta]\!]_1 + [\![\beta]\!]_2 + \cdots + [\![\beta]\!]_N = \beta$
and so on…
$g : (y, \alpha, \beta, \ldots) \mapsto$ Accept / Reject

Example: matrix multiplication $y = Hx$

$\mathscr{P}_1$ | $\mathscr{P}_2$ | $\cdots$ | $\mathscr{P}_N$

$[\![x]\!]_1$, $[\![\alpha]\!]_1 = H \cdot [\![x]\!]_1$

$[\![x]\!]_2$, $[\![\alpha]\!]_2 = H \cdot [\![x]\!]_2$

$\cdots$

$[\![x]\!]_N$, $[\![\alpha]\!]_N = H \cdot [\![x]\!]_N$

mult. by H is linear

public recovery

$$[\![\alpha]\!]_1 + [\![\alpha]\!]_2 + \cdots + [\![\alpha]\!]_N = \alpha$$

$$g(y, \alpha) = \begin{cases} \text{Accept} & \text{if } y = \alpha \\ \text{Reject} & \text{if } y \neq \alpha \end{cases}$$

$$g(y, \alpha) = \text{Accept} \iff Hx = y$$

MPCitH transform

Prover

Verifier

MPCitH transform

① Generate and commit shares

$$[\![x]\!] = ([\![x]\!]_1, \ldots, [\![x]\!]_N)$$

$$\mathrm{Com}^{\rho_1}([\![x]\!]_1)$$

$$\cdots$$

$$\mathrm{Com}^{\rho_N}([\![x]\!]_N)$$

→

Prover

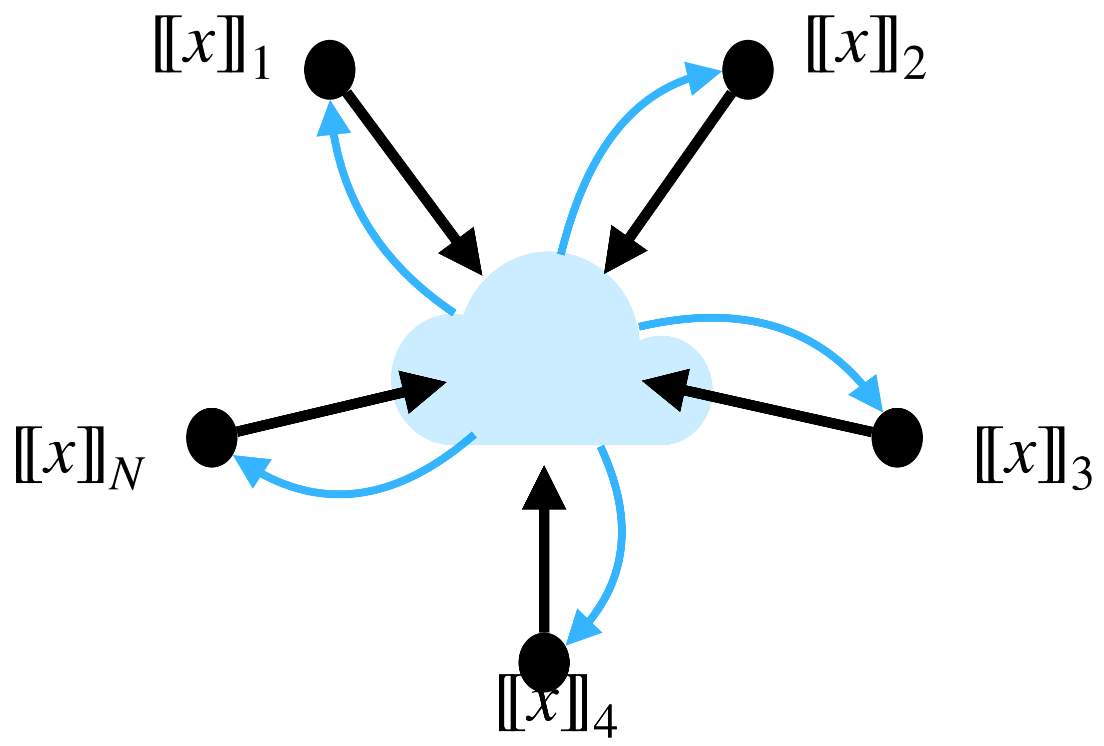
Verifier

MPCitH transform

① Generate and commit shares

$[\![x]\!] = ([\![x]\!]_1, \ldots, [\![x]\!]_N)$

② Run MPC in their head



$\mathrm{Com}^{\rho_1}([\![x]\!]_1)$

…

$\mathrm{Com}^{\rho_N}([\![x]\!]_N)$

→

send broadcast

$[\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N$

→

Prover

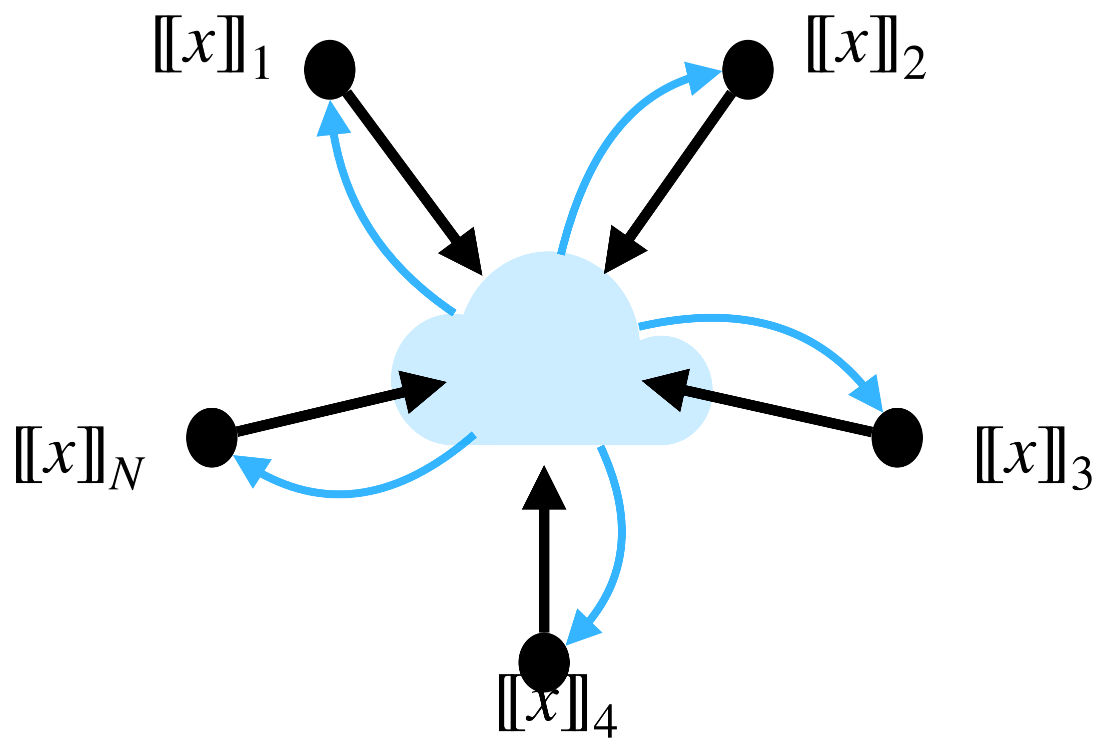
Verifier

MPCitH transform

① Generate and commit shares

$$[\![x]\!] = ([\![x]\!]_1, \ldots, [\![x]\!]_N)$$

② Run MPC in their head



$\mathrm{Com}^{\rho_1}([\![x]\!]_1)$

…

$\mathrm{Com}^{\rho_N}([\![x]\!]_N)$ →

send broadcast

$[\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N$ →

← i^*

③ Chose a random party

$$i^* \xleftarrow{\$} \{1, \ldots, N\}$$

Prover

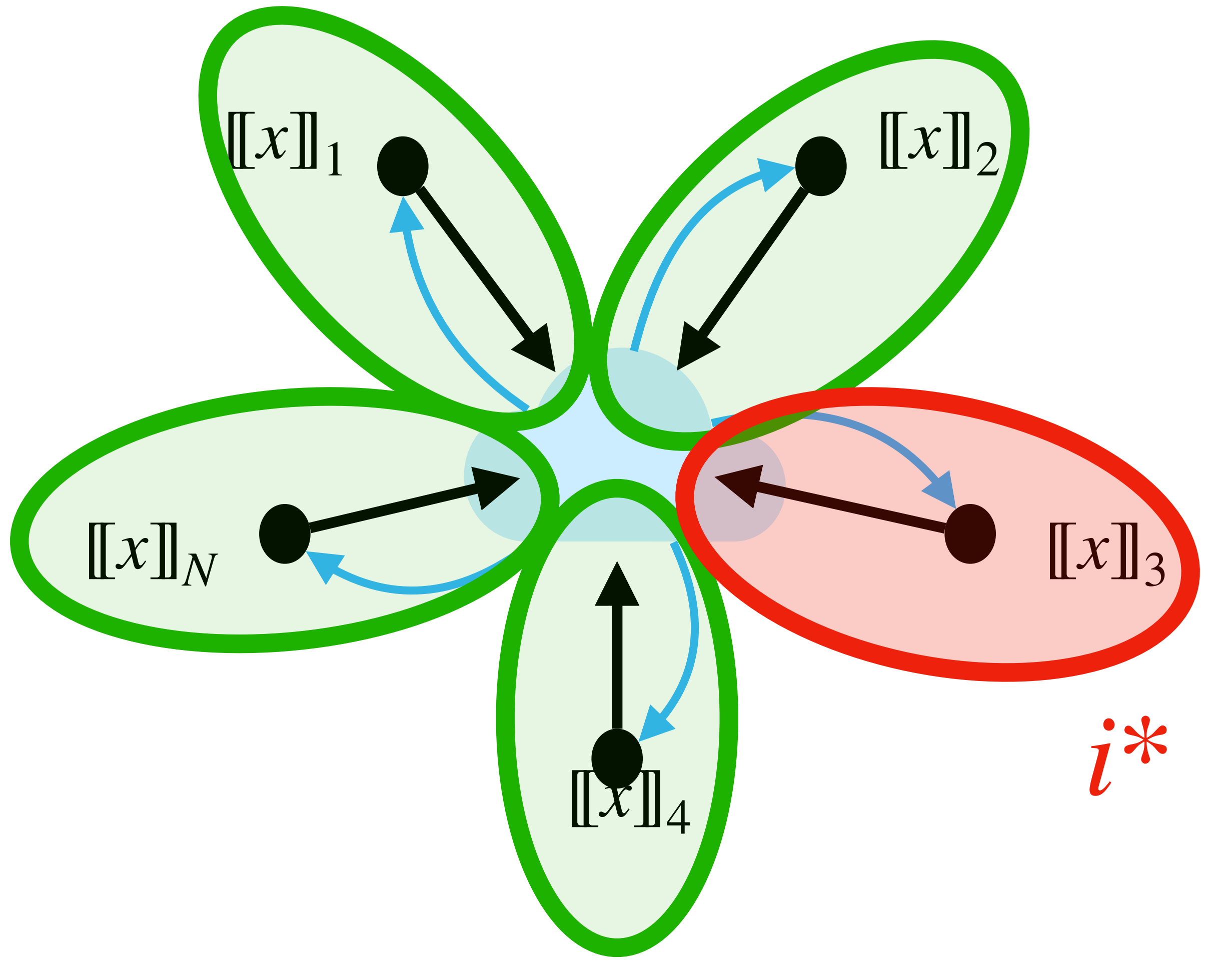
Verifier

MPCitH transform

① Generate and commit shares

$[\![x]\!] = ([\![x]\!]_1, \ldots, [\![x]\!]_N)$

② Run MPC in their head



④ Open parties $\{1,\ldots,N\}\backslash\{i^*\}$

$\mathrm{Com}^{\rho_1}([\![x]\!]_1)$

...

$\mathrm{Com}^{\rho_N}([\![x]\!]_N)$

send broadcast

$[\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N$

i^*

$([\![x]\!]_i, \rho_i)_{i \neq i^*}$

③ Chose a random party

$i^* \xleftarrow{\$} \{1,\ldots,N\}$

Prover

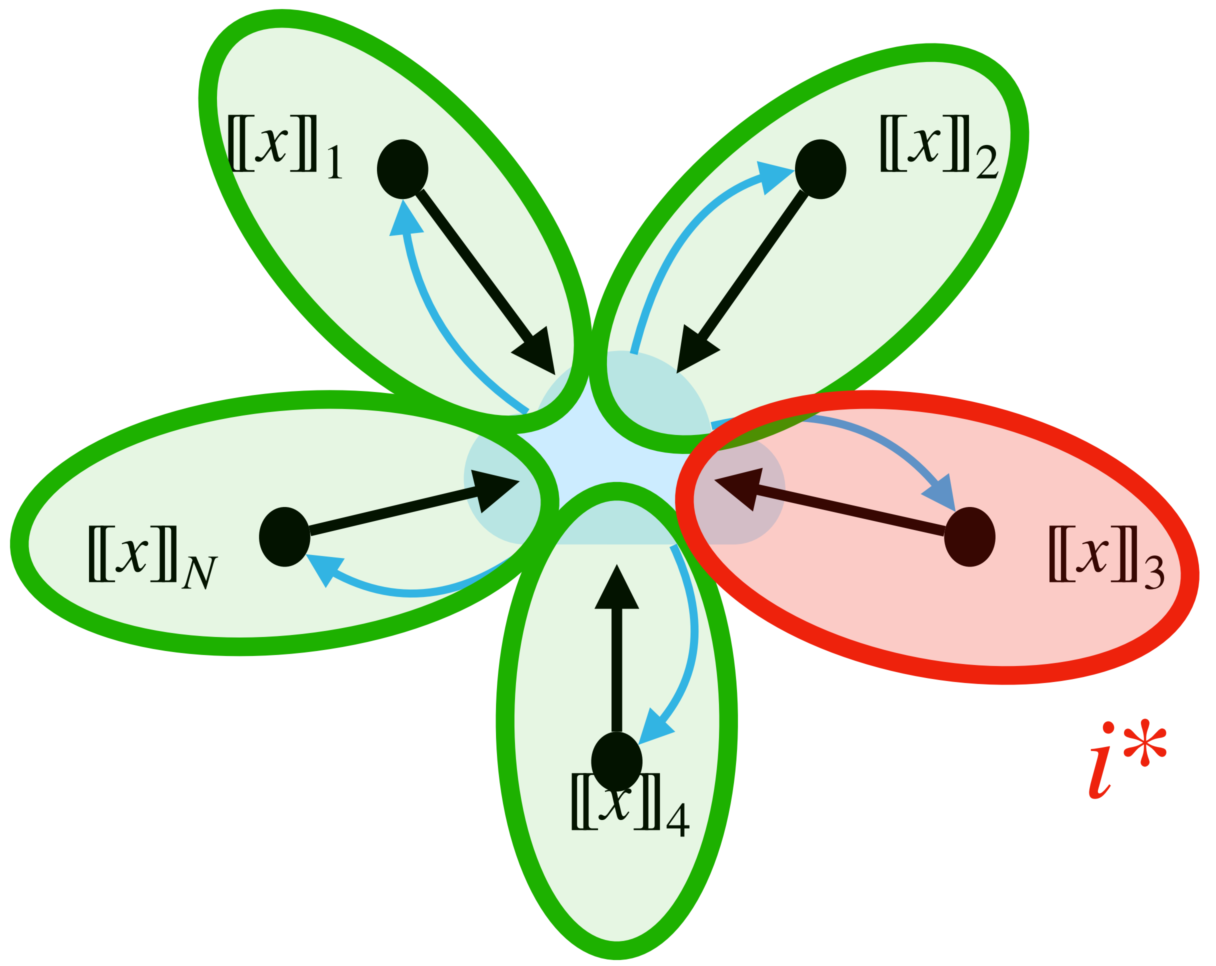
Verifier

MPCitH transform

① Generate and commit shares

$[\![x]\!] = ([\![x]\!]_1, \ldots, [\![x]\!]_N)$

② Run MPC in their head



④ Open parties $\{1,\ldots,N\} \backslash \{i^*\}$

$\mathrm{Com}^{\rho_1}([\![x]\!]_1)$

$\cdots$

$\mathrm{Com}^{\rho_N}([\![x]\!]_N)$ →

send broadcast

$[\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N$ →

← i^*

$([\![x]\!]_i, \rho_i)_{i \neq i^*}$ →

③ Chose a random party

$i^* \xleftarrow{\$} \{1,\ldots,N\}$

⑤ Check $\forall i \neq i^*$

- Commitments $\mathrm{Com}^{\rho_i}([\![x]\!]_i)$
- MPC computation $[\![\alpha]\!]_i = \varphi([\![x]\!]_i)$

Check $g(y, \alpha) =$ Accept

Prover

Verifier

MPCitH transform

- **Zero-knowledge** $\iff$ MPC protocol is $(N-1)$-private

MPCitH transform

- **Zero-knowledge** $\iff$ MPC protocol is $(N-1)$-private

- **Soundness**
  - if $g(y, \alpha) \neq$ Accept → Verifier rejects
  - if $g(y, \alpha) =$ Accept, then
    - either $[\![x]\!]$ = sharing of correct witness $F(x) = y$ → Prover honest
    - or Prover has cheated for at least one party

→ Cheat undetected with proba $\dfrac{1}{N}$

MPCitH transform

- **Zero-knowledge** $\iff$ MPC protocol is $(N-1)$-private
- **Soundness**
  - if $g(y, \alpha) \neq$ Accept → Verifier rejects
  - if $g(y, \alpha) =$ Accept, then
    - either $[\![x]\!]$ = sharing of correct witness $F(x) = y$ → Prover honest
    - or Prover has cheated for at least one party

      → Cheat undetected with proba $\frac{1}{N}$
- **Parallel repetition**

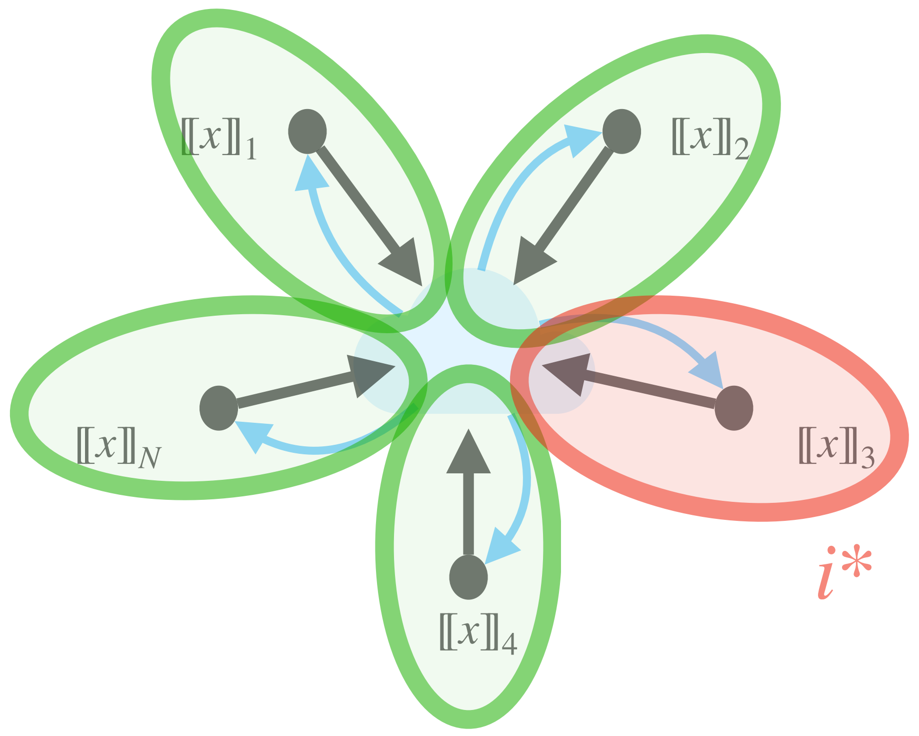
  Protocol repeated τ times in parallel → soundness error $\left(\frac{1}{N}\right)^{\tau}$

Example: matrix multiplication $y = Hx$

$\mathscr{P}_1$	$\mathscr{P}_2$		$\mathscr{P}_N$
$[\![x]\!]_1$ $[\![\alpha]\!]_1 = H \cdot [\![x]\!]_1$	$[\![x]\!]_2$ $[\![\alpha]\!]_2 = H \cdot [\![x]\!]_2$	$\cdots$	$[\![x]\!]_N$ $[\![\alpha]\!]_N = H \cdot [\![x]\!]_N$

$$[\![\alpha]\!]_1 + [\![\alpha]\!]_2 + \cdots + [\![\alpha]\!]_N = \alpha$$

Prover



Prover → Verifier: $\{\mathrm{Com}^{\rho_i}([\![x]\!]_i)\}$

Prover → Verifier: $\{[\![\alpha]\!]_i\}$

Verifier → Prover: i^*

Prover → Verifier: $\{[\![x]\!]_i, \rho_i\}_{i \neq i^*}$

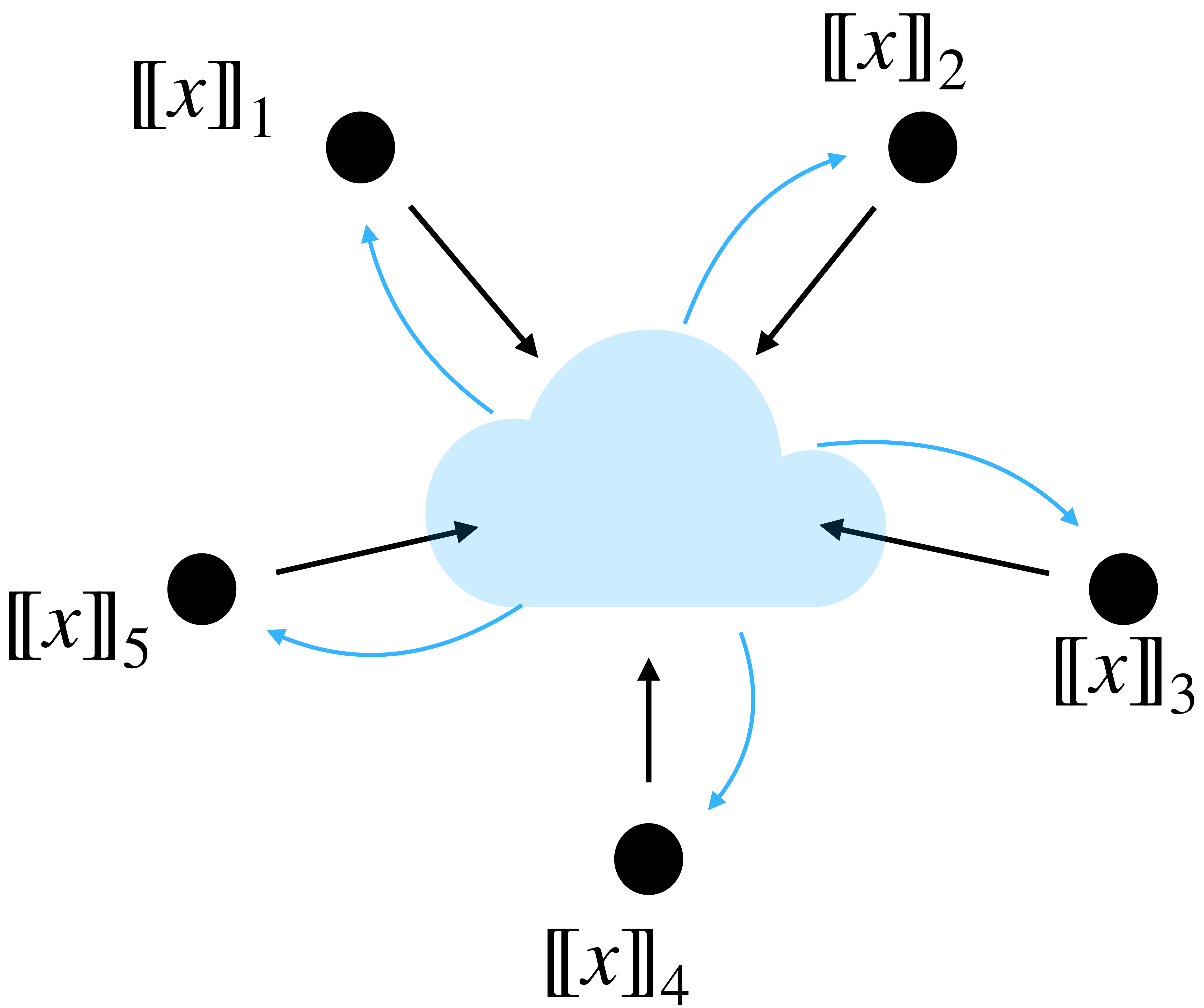
Verifier

Check $\forall i \neq i^*$

- Commitments $\mathrm{Com}^{\rho_i}([\![x]\!]_i)$
- MPC computation $[\![\alpha]\!]_i = H \cdot [\![x]\!]_i$

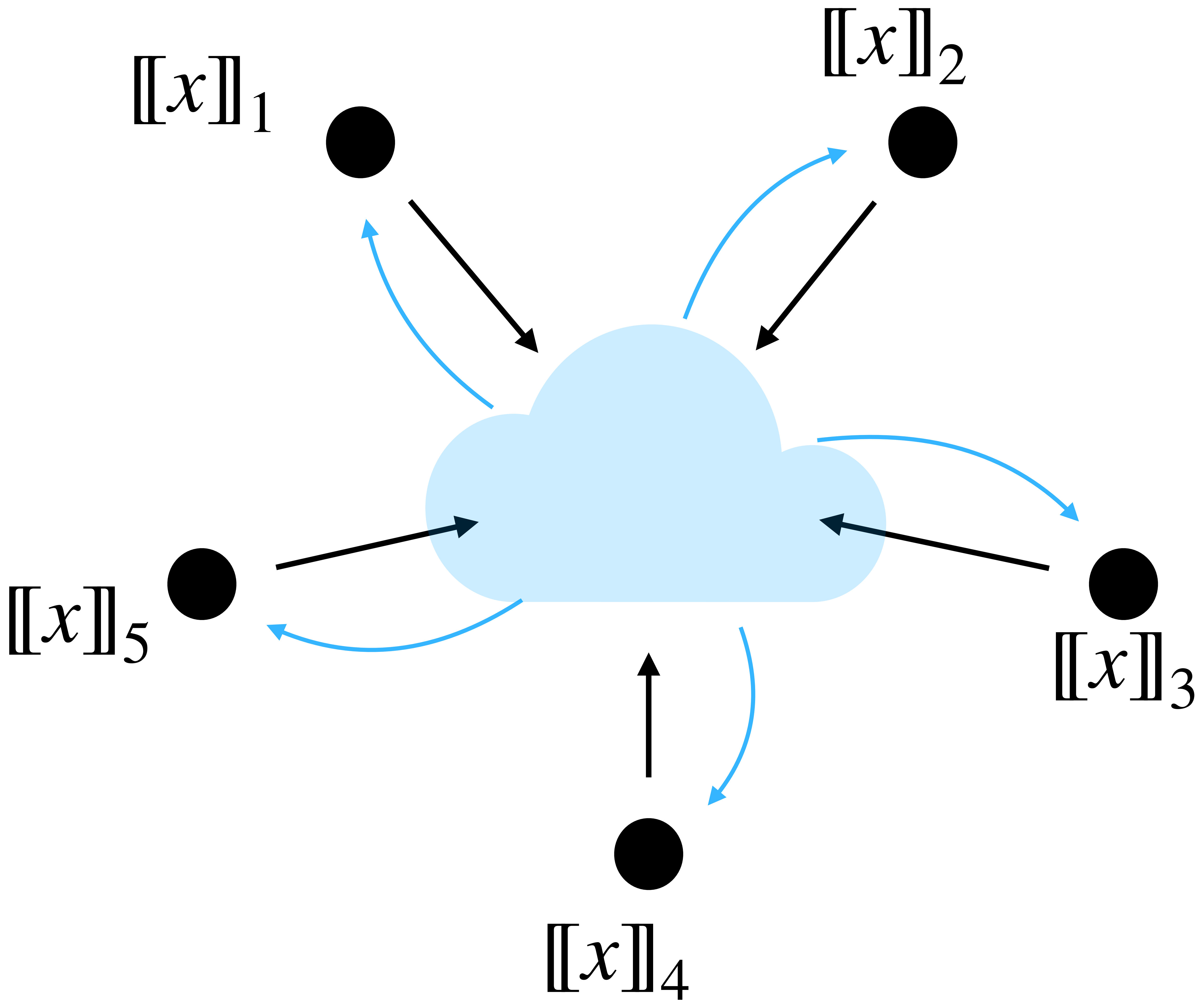
Check $\alpha := \Sigma_i [\![\alpha]\!]_i = y$

Complete MPC model

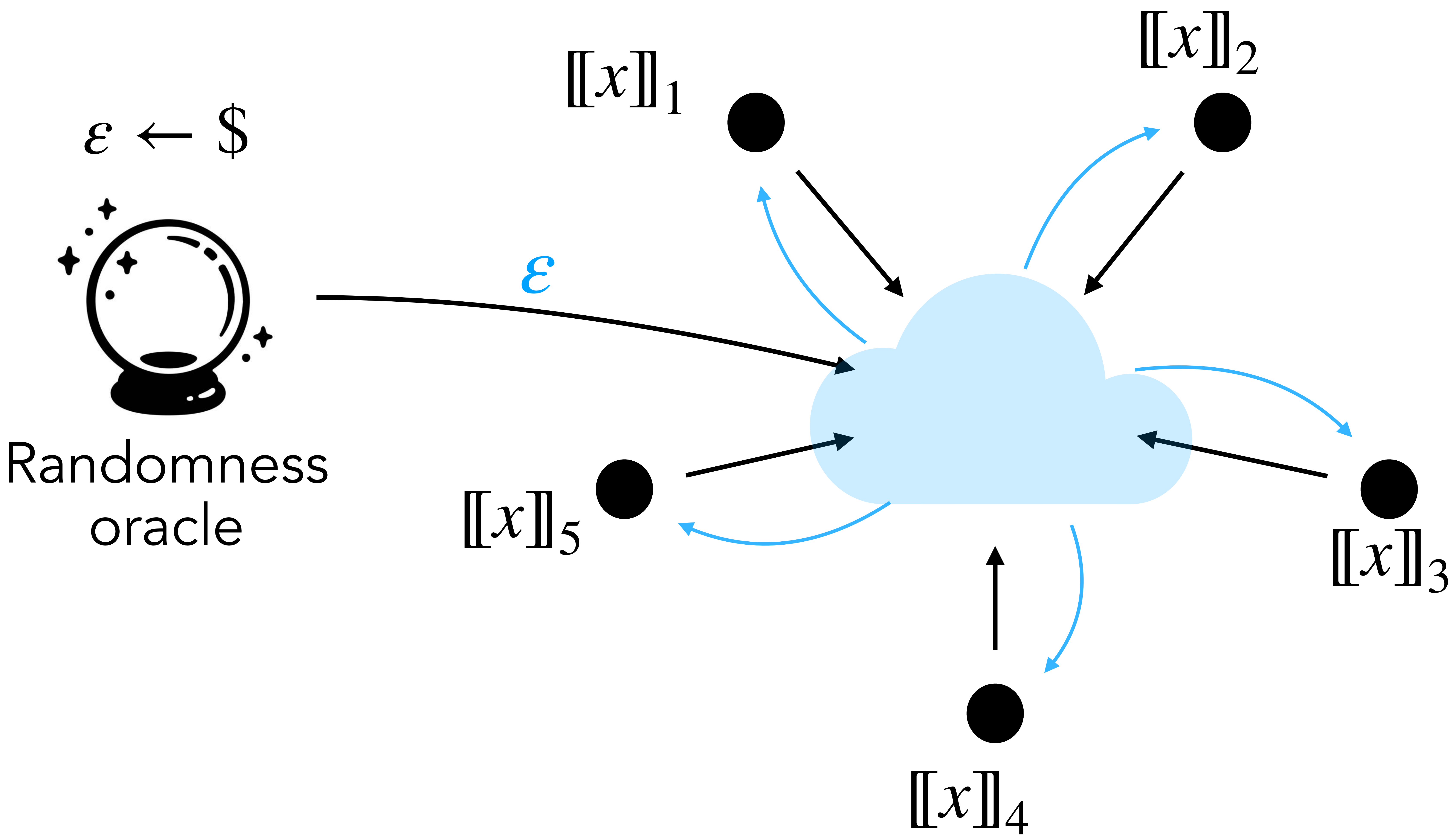


Complete MPC model

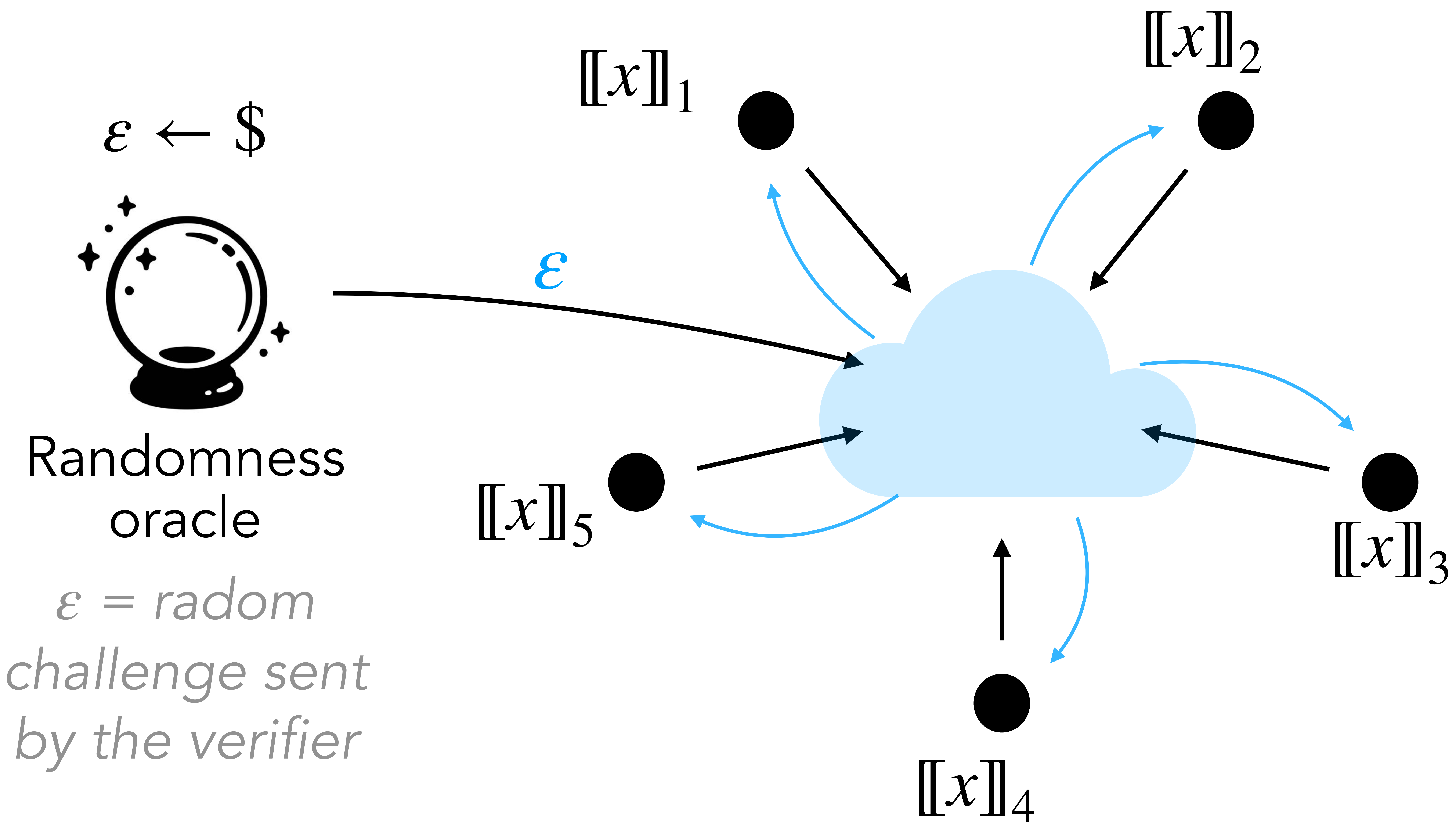
Randomness oracle



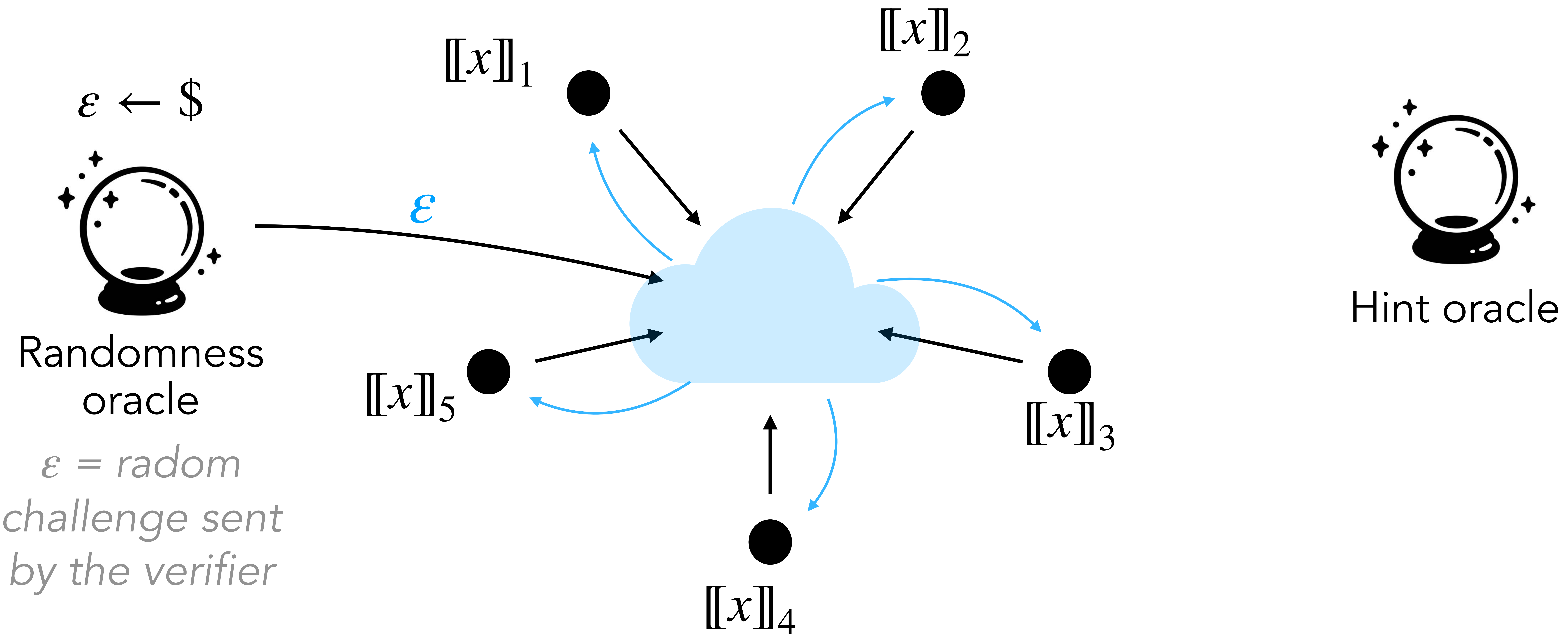
Complete MPC model



Complete MPC model

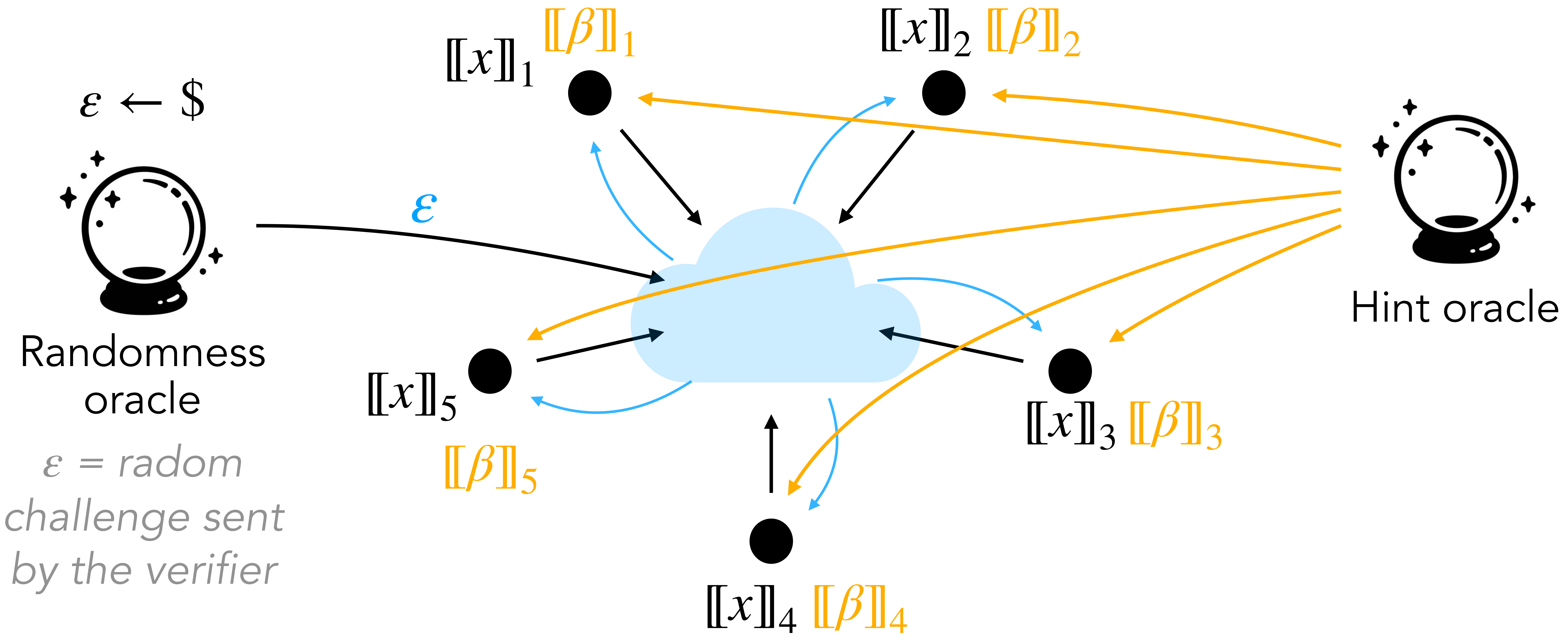


Complete MPC model

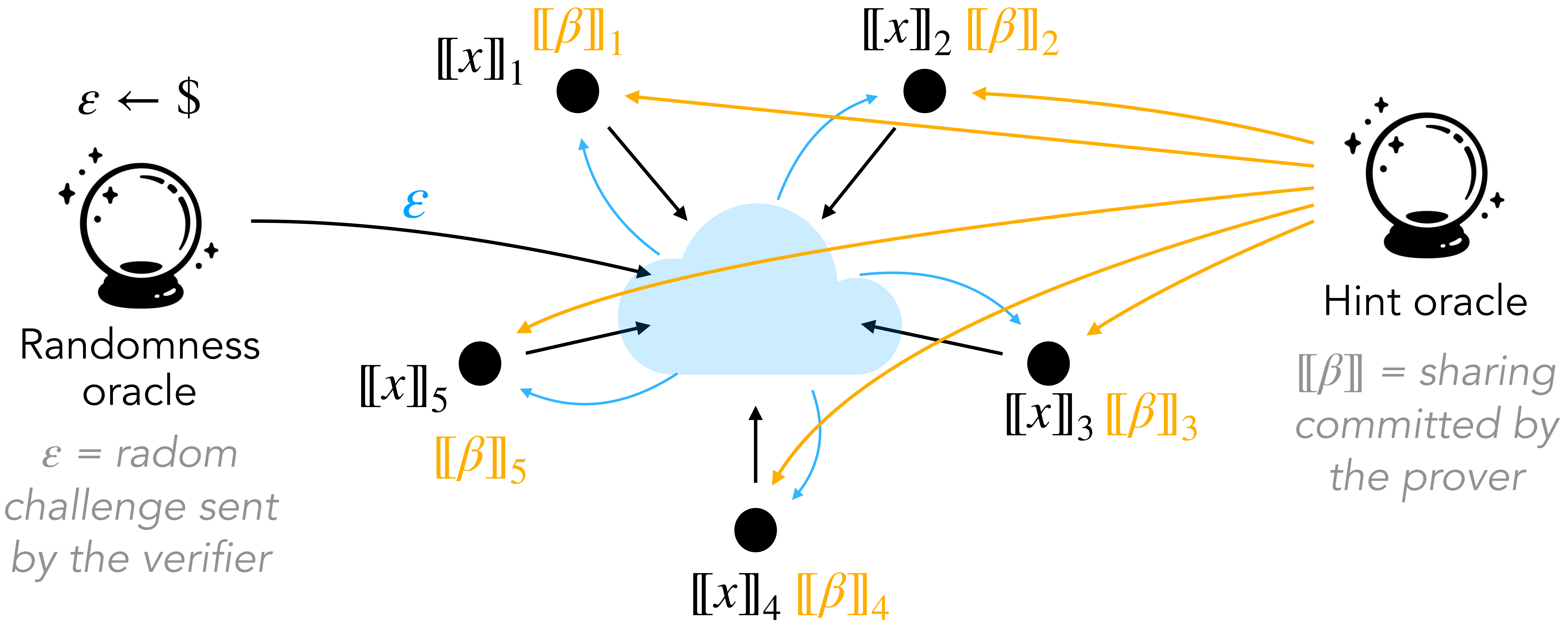


ε = radom challenge sent by the verifier

Complete MPC model



Complete MPC model



Example: [BN20] check product $xy = z$

$\mathscr{P}_1$

$[\![x]\!]_1, [\![y]\!]_1, [\![z]\!]_1$

$\cdots$

$\mathscr{P}_N$

$[\![x]\!]_N, [\![y]\!]_N, [\![z]\!]_N$

Example: [BN20] check product $xy = z$

$\mathscr{P}_1$

$[\![x]\!]_1, [\![y]\!]_1, [\![z]\!]_1$

$[\![a]\!]_1, [\![b]\!]_1, [\![c]\!]_1$

$\cdots$

$\mathscr{P}_N$

$[\![x]\!]_N, [\![y]\!]_N, [\![z]\!]_N$

$[\![a]\!]_N, [\![b]\!]_N, [\![c]\!]_N$

← **hint** $ab = c$

Example: [BN20] check product $xy = z$

$\mathscr{P}_1$

$[\![x]\!]_1, [\![y]\!]_1, [\![z]\!]_1$

$[\![a]\!]_1, [\![b]\!]_1, [\![c]\!]_1$

ε

$\cdots$

$\mathscr{P}_N$

$[\![x]\!]_N, [\![y]\!]_N, [\![z]\!]_N$

$[\![a]\!]_N, [\![b]\!]_N, [\![c]\!]_N$

ε

← **hint** $ab = c$

← **random** ε

Example: [BN20] check product $xy = z$

$\mathscr{P}_1$

$[\![x]\!]_1, [\![y]\!]_1, [\![z]\!]_1$

$[\![a]\!]_1, [\![b]\!]_1, [\![c]\!]_1$

ε

$[\![\alpha]\!]_1 = \varepsilon[\![x]\!]_1 + [\![a]\!]_1$

$[\![\beta]\!]_1 = [\![y]\!]_1 + [\![b]\!]_1$

↓

$[\![\alpha]\!]_1 \; [\![\beta]\!]_1$

…

$\mathscr{P}_N$

$[\![x]\!]_N, [\![y]\!]_N, [\![z]\!]_N$

$[\![a]\!]_N, [\![b]\!]_N, [\![c]\!]_N$

ε

$[\![\alpha]\!]_N = \varepsilon[\![x]\!]_N + [\![a]\!]_N$

$[\![\beta]\!]_N = [\![y]\!]_N + [\![b]\!]_N$

↓

$[\![\alpha]\!]_N \; [\![\beta]\!]_N$

← **hint** $ab = c$

← **random** ε

$\alpha = \epsilon x + a$

$\beta = y + b$

Example: [BN20] check product $xy = z$

$\mathscr{P}_1$

$[\![x]\!]_1, [\![y]\!]_1, [\![z]\!]_1$

$[\![a]\!]_1, [\![b]\!]_1, [\![c]\!]_1$

ε

$[\![\alpha]\!]_1 = \varepsilon[\![x]\!]_1 + [\![a]\!]_1$

$[\![\beta]\!]_1 = [\![y]\!]_1 + [\![b]\!]_1$

$[\![\alpha]\!]_1$ $[\![\beta]\!]_1$

$[\![v]\!]_1 = \epsilon[\![z]\!]_1 - [\![c]\!]_1 + \alpha[\![b]\!]_1 + \beta[\![a]\!]_1 - \alpha\beta$

$[\![v]\!]_1$

...

$\mathscr{P}_N$

$[\![x]\!]_N, [\![y]\!]_N, [\![z]\!]_N$

$[\![a]\!]_N, [\![b]\!]_N, [\![c]\!]_N$

ε

$[\![\alpha]\!]_N = \varepsilon[\![x]\!]_N + [\![a]\!]_N$

$[\![\beta]\!]_N = [\![y]\!]_N + [\![b]\!]_N$

$[\![\alpha]\!]_N$ $[\![\beta]\!]_N$

$[\![v]\!]_N = \epsilon[\![z]\!]_N - [\![c]\!]_N + \alpha[\![b]\!]_N + \beta[\![a]\!]_N - \alpha\beta$

$[\![v]\!]_N$

← *hint* $ab = c$

← *random* ε

$\alpha = \epsilon x + a$

$\beta = y + b$

v

Example: [BN20] check product $xy = z$

$\mathscr{P}_1$

$[\![x]\!]_1, [\![y]\!]_1, [\![z]\!]_1$

$[\![a]\!]_1, [\![b]\!]_1, [\![c]\!]_1$

ε

$[\![\alpha]\!]_1 = \varepsilon[\![x]\!]_1 + [\![a]\!]_1$

$[\![\beta]\!]_1 = [\![y]\!]_1 + [\![b]\!]_1$

$[\![\alpha]\!]_1 \; [\![\beta]\!]_1$

$[\![v]\!]_1 = \epsilon[\![z]\!]_1 - [\![c]\!]_1 + \alpha[\![b]\!]_1 + \beta[\![a]\!]_1 - \alpha\beta$

$[\![v]\!]_1$

…

$\mathscr{P}_N$

$[\![x]\!]_N, [\![y]\!]_N, [\![z]\!]_N$

$[\![a]\!]_N, [\![b]\!]_N, [\![c]\!]_N$ ← hint $ab = c$

ε ← random ε

$[\![\alpha]\!]_N = \varepsilon[\![x]\!]_N + [\![a]\!]_N$

$[\![\beta]\!]_N = [\![y]\!]_N + [\![b]\!]_N$

$[\![\alpha]\!]_N \; [\![\beta]\!]_N$

$\alpha = \epsilon x + a$

$\beta = y + b$

$[\![v]\!]_N = \epsilon[\![z]\!]_N - [\![c]\!]_N + \alpha[\![b]\!]_N + \beta[\![a]\!]_N - \alpha\beta$

$[\![v]\!]_N$

v

$$g(v) = \begin{cases} \text{Accept} & \text{if } v = 0 \\ \text{Reject} & \text{if } v \neq 0 \end{cases}$$

Example: [BN20] check product $xy = z$

$\mathscr{P}_1$

$$[\![x]\!]_1, [\![y]\!]_1, [\![z]\!]_1$$
$$[\![a]\!]_1, [\![b]\!]_1, [\![c]\!]_1$$
$$\varepsilon$$
$$[\![\alpha]\!]_1 = \varepsilon[\![x]\!]_1 + [\![a]\!]_1$$
$$[\![\beta]\!]_1 = [\![y]\!]_1 + [\![b]\!]_1$$

$$[\![\alpha]\!]_1 \ [\![\beta]\!]_1$$

$$[\![v]\!]_1 = \epsilon[\![z]\!]_1 - [\![c]\!]_1 + \alpha[\![b]\!]_1 + \beta[\![a]\!]_1 - \alpha\beta$$

$$[\![v]\!]_1$$

…

$\mathscr{P}_N$

$$[\![x]\!]_N, [\![y]\!]_N, [\![z]\!]_N$$
$$[\![a]\!]_N, [\![b]\!]_N, [\![c]\!]_N$$
$$\varepsilon$$
$$[\![\alpha]\!]_N = \varepsilon[\![x]\!]_N + [\![a]\!]_N$$
$$[\![\beta]\!]_N = [\![y]\!]_N + [\![b]\!]_N$$

$$[\![\alpha]\!]_N \ [\![\beta]\!]_N$$

$$[\![v]\!]_N = \epsilon[\![z]\!]_N - [\![c]\!]_N + \alpha[\![b]\!]_N + \beta[\![a]\!]_N - \alpha\beta$$

$$[\![v]\!]_N$$

← **hint** $ab = c$

← **random** ε

$$\alpha = \epsilon x + a$$
$$\beta = y + b$$

v

$$g(v) = \begin{cases} \text{Accept} & \text{if } v = 0 \\ \text{Reject} & \text{if } v \neq 0 \end{cases}$$

If $xy = z$ and $ab = c$, then $v = 0$

If $xy \neq z$ or $ab \neq c$, then $\Pr[v = 0] = 1/|\mathbb{F}|$

Example: [BN20] check product $xy = z$

$\mathscr{P}_1$

$[\![x]\!]_1, [\![y]\!]_1, [\![z]\!]_1$

$[\![a]\!]_1, [\![b]\!]_1, [\![c]\!]_1$

ε

$[\![\alpha]\!]_1 = \varepsilon[\![x]\!]_1 + [\![a]\!]_1$

$[\![\beta]\!]_1 = [\![y]\!]_1 + [\![b]\!]_1$

↓ $[\![\alpha]\!]_1 \; [\![\beta]\!]_1$

$[\![v]\!]_1 = \epsilon[\![z]\!]_1 - [\![c]\!]_1 + \alpha[\![b]\!]_1 + \beta[\![a]\!]_1 - \alpha\beta$

↓ $[\![v]\!]_1$

…

$\mathscr{P}_N$

$[\![x]\!]_N, [\![y]\!]_N, [\![z]\!]_N$

$[\![a]\!]_N, [\![b]\!]_N, [\![c]\!]_N$ ← hint $ab = c$

ε ← random ε

$[\![\alpha]\!]_N = \varepsilon[\![x]\!]_N + [\![a]\!]_N$

$[\![\beta]\!]_N = [\![y]\!]_N + [\![b]\!]_N$

↓ $[\![\alpha]\!]_N \; [\![\beta]\!]_N$

$\alpha = \epsilon x + a$

$\beta = y + b$

$[\![v]\!]_N = \epsilon[\![z]\!]_N - [\![c]\!]_N + \alpha[\![b]\!]_N + \beta[\![a]\!]_N - \alpha\beta$

↓ $[\![v]\!]_N$

$$g(v) = \begin{cases} \text{Accept} & \text{if } v = 0 \\ \text{Reject} & \text{if } v \neq 0 \end{cases}$$

If $xy = z$ and $ab = c$, then $v = 0$

If $xy \neq z$ or $ab \neq c$, then $\Pr[v = 0] = 1/|\mathbb{F}|$

false positive probability

Verifying arbitrary circuits

- Previous slide reference:

  [BN20] Baum, Nof. "Concretely-Efficient Zero-Knowledge Arguments for Arithmetic Circuits and Their Application to Lattice-Based Cryptography" (PKC 2020)

- Product-check protocol ⇒ protocol for checking any arithmetic circuit $C(x) = y$
- Principle:
  - Let $\{c_i = a_i \cdot b_i\}$ all the multiplications in C
  - Extended witness: $w = x \parallel (c_1, \ldots, c_m)$
  - Compute $[\![y]\!]$ = linear function of $[\![w]\!]$ → check $[\![y]\!]$ = sharing of y
  - $[\![a_i]\!], [\![b_i]\!], [\![c_i]\!]$ = linear functions of $[\![w]\!]$ → product check on $[\![a_i]\!], [\![b_i]\!], [\![c_i]\!]$

MPCitH: optimisations

Optimising communication (sig. size)

- **Signature = transcript P → V**
  - $\{\mathrm{Com}^{\rho_i}([\![x]\!]_i)\}$ → N commitments
  - $[\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N$ → N MPC broadcasts
  - $\{[\![x]\!]_i, \rho_i\}_{i \neq i^*}$ → $N - 1$ input shares + random tapes

Optimising communication (sig. size)

- **Signature = transcript P → V**
  - $\{\mathrm{Com}^{\rho_i}([\![x]\!]_i)\}$ → N commitments
  - $[\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N$ → N MPC broadcasts
  - $\{[\![x]\!]_i, \rho_i\}_{i \neq i^*}$ → $N - 1$ input shares + random tapes
- First optimisation: **hashing**
  - $[\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N$ → $h = \mathrm{Hash}([\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N),\ \ \alpha = \Sigma_i [\![\alpha]\!]_i$
  - Verification
    - $[\![\alpha]\!]_i = \varphi([\![x]\!]_i) \quad \forall i \neq i^*$
    - $[\![\alpha]\!]_{i^*} = \alpha - \Sigma_{i \neq i^*} [\![\alpha]\!]_i$
    - Check $\mathrm{Hash}([\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N) = h$

Optimising communication (sig. size)

- **Signature = transcript P → V**
  - $\{\mathrm{Com}^{\rho_i}([\![x]\!]_i)\}$ → N commitments
  - $[\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N$ → ~~N MPC broadcasts~~ → hash (+1 MPC broadcast)
  - $\{[\![x]\!]_i, \rho_i\}_{i \neq i^*}$ → $N - 1$ input shares + random tapes
- First optimisation: **hashing**
  - $[\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N$ → $h = \mathrm{Hash}([\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N)$, $\alpha = \Sigma_i [\![\alpha]\!]_i$
  - Verification
    - $[\![\alpha]\!]_i = \varphi([\![x]\!]_i) \quad \forall i \neq i^*$
    - $[\![\alpha]\!]_{i^*} = \alpha - \Sigma_{i \neq i^*} [\![\alpha]\!]_i$
    - Check $\mathrm{Hash}([\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N) = h$

Optimising communication (sig. size)

- **Signature = transcript P → V**
  - $\{\mathrm{Com}^{\rho_i}([\![x]\!]_i)\}$ → ~~N commitments~~ → hash +1 commitment
  - $[\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N$ → ~~N MPC broadcasts~~ → hash (+1 MPC broadcast)
  - $\{[\![x]\!]_i, \rho_i\}_{i \neq i^*}$ → $N - 1$ input shares + random tapes
- First optimisation: **hashing**
  - $[\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N$ → $h = \mathrm{Hash}([\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N)$, $\alpha = \Sigma_i [\![\alpha]\!]_i$
  - Verification
    - $[\![\alpha]\!]_i = \varphi([\![x]\!]_i) \quad \forall i \neq i^*$
    - $[\![\alpha]\!]_{i^*} = \alpha - \Sigma_{i \neq i^*} [\![\alpha]\!]_i$
    - Check $\mathrm{Hash}([\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N) = h$
- Also works with commitments

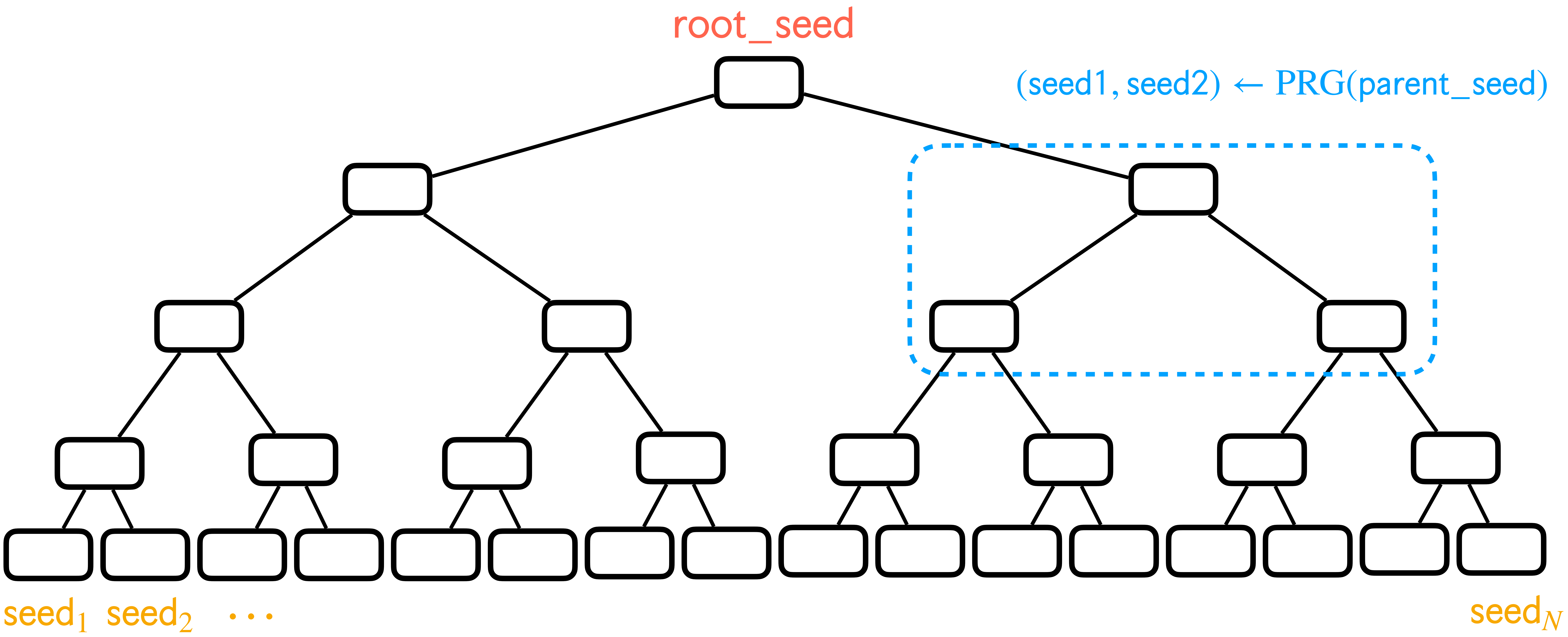
Optimising communication (sig. size)

- **Signature = transcript P → V**
  - $\{\mathrm{Com}^{\rho_i}([\![x]\!]_i)\}$ → ~~N commitments~~ → hash +1 commitment
  - $[\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N$ → ~~N MPC broadcasts~~ → hash (+1 MPC broadcast)
  - $\{[\![x]\!]_i, \rho_i\}_{i \neq i^*}$ → $N-1$ input shares + random tapes **main cost**
- First optimisation: **hashing**
  - $[\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N$ → $h = \mathrm{Hash}([\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N)$, $\alpha = \Sigma_i [\![\alpha]\!]_i$
  - Verification
    - $[\![\alpha]\!]_i = \varphi([\![x]\!]_i) \quad \forall i \neq i^*$
    - $[\![\alpha]\!]_{i^*} = \alpha - \Sigma_{i \neq i^*} [\![\alpha]\!]_i$
    - Check $\mathrm{Hash}([\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N) = h$
- Also works with commitments

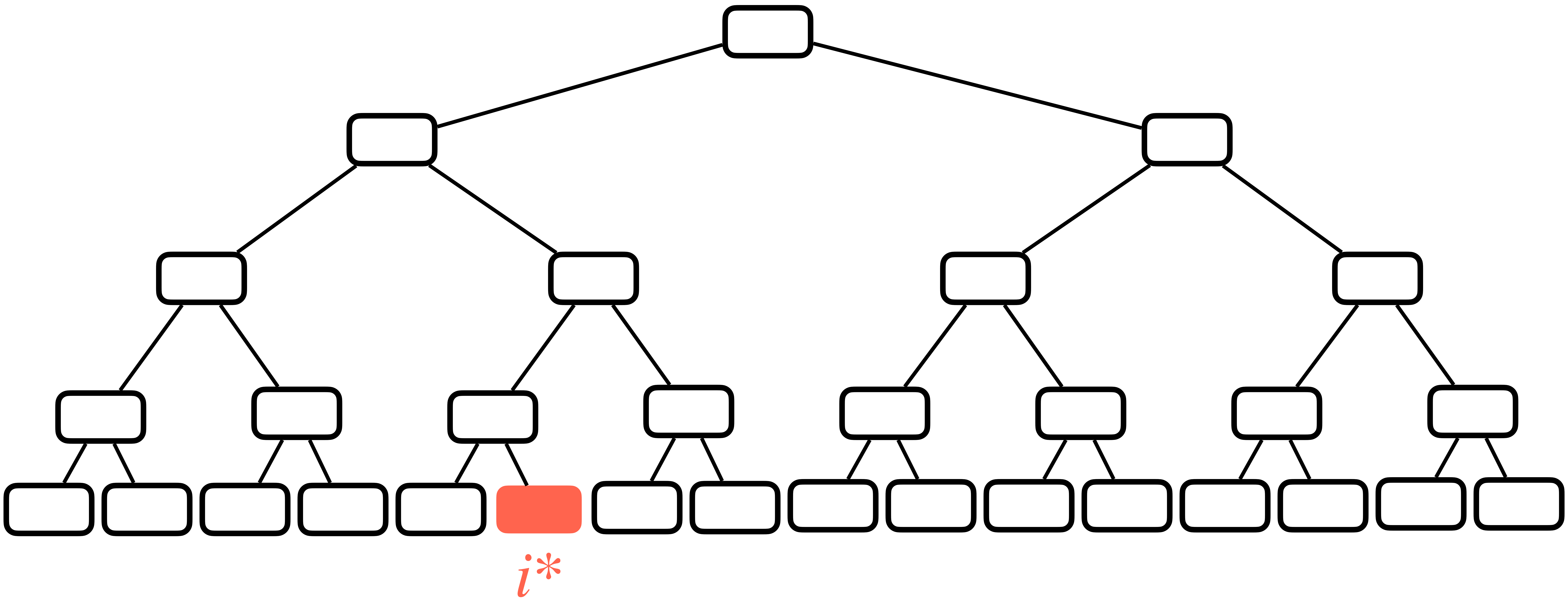
Second optimisation: seed trees

- **[KKW18]** Katz, Kolesnikov, Wang: "Improved Non-Interactive Zero Knowledge with Applications to Post-Quantum Signatures" (CCS 2018)
- Pseudorandom generation from seed
  - $([\![x]\!]_i, \rho_i) \leftarrow \mathrm{PRG}(\mathsf{seed}_i)$
  - $[\![x]\!]_N = x - \Sigma_{i=1}^{N} [\![x]\!]_i$
- Seeds $\{\mathsf{seed}_i\}$ generated from a common "root seed"
- Goal: revealing $\{\mathsf{seed}_i\}_{i \neq i*}$ with less than $(N-1) \cdot \lambda$ bits

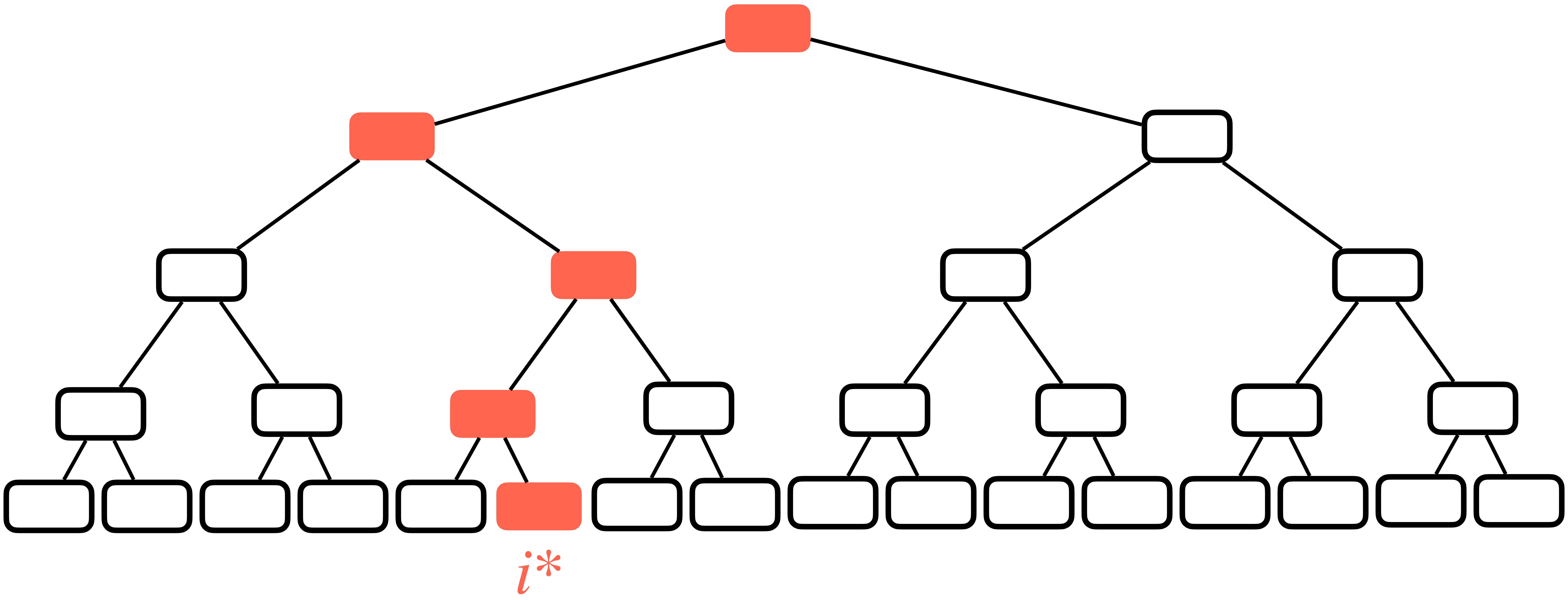
Second optimisation: seed trees



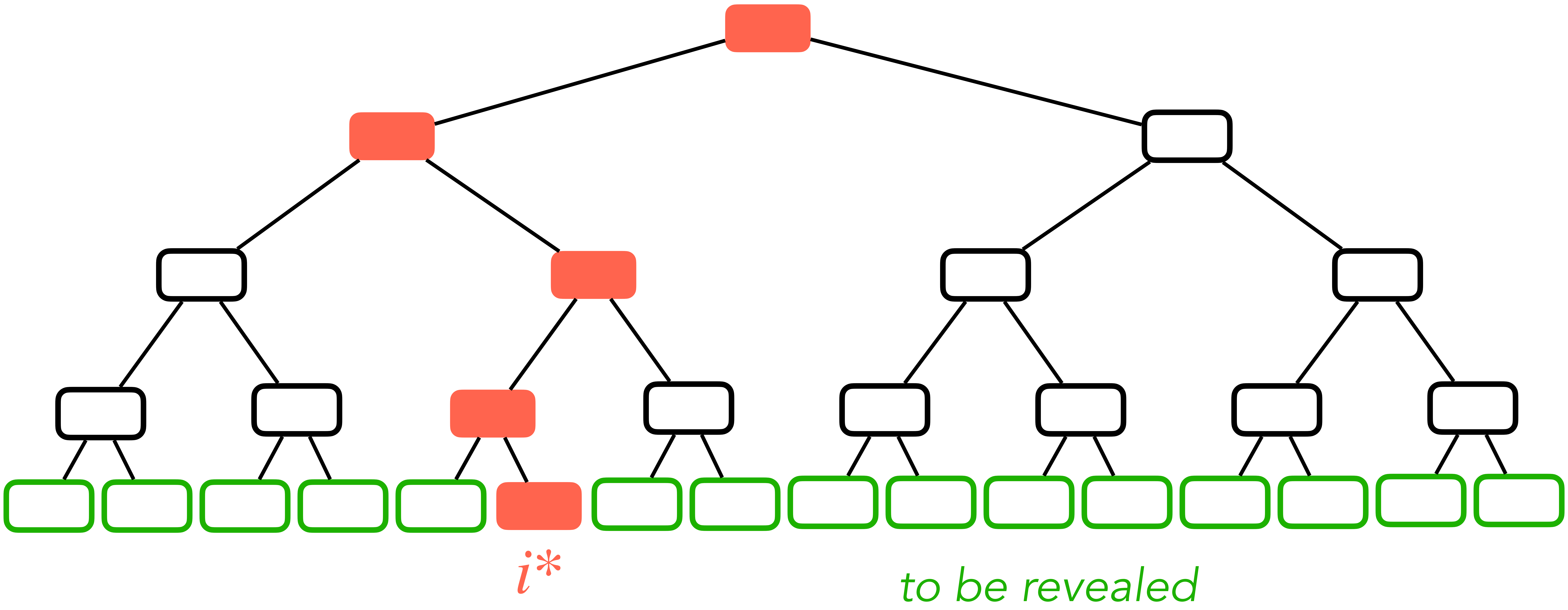
Second optimisation: seed trees



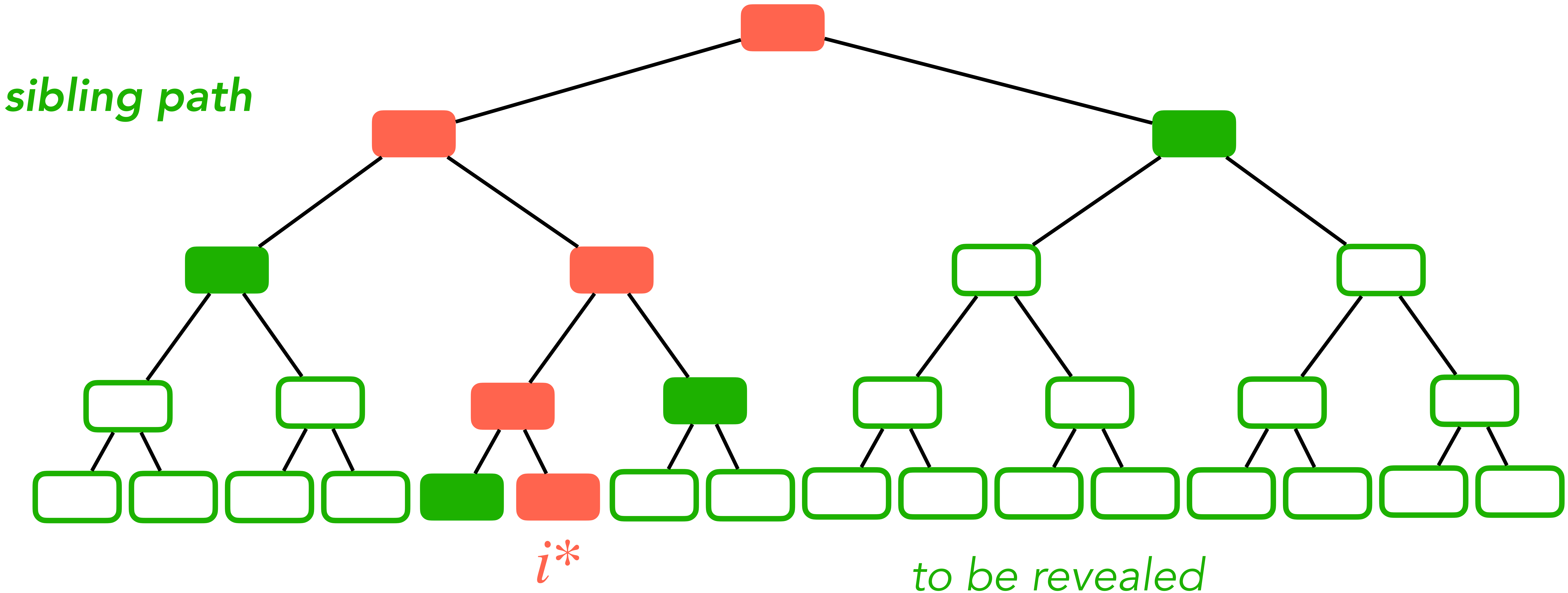
Second optimisation: seed trees



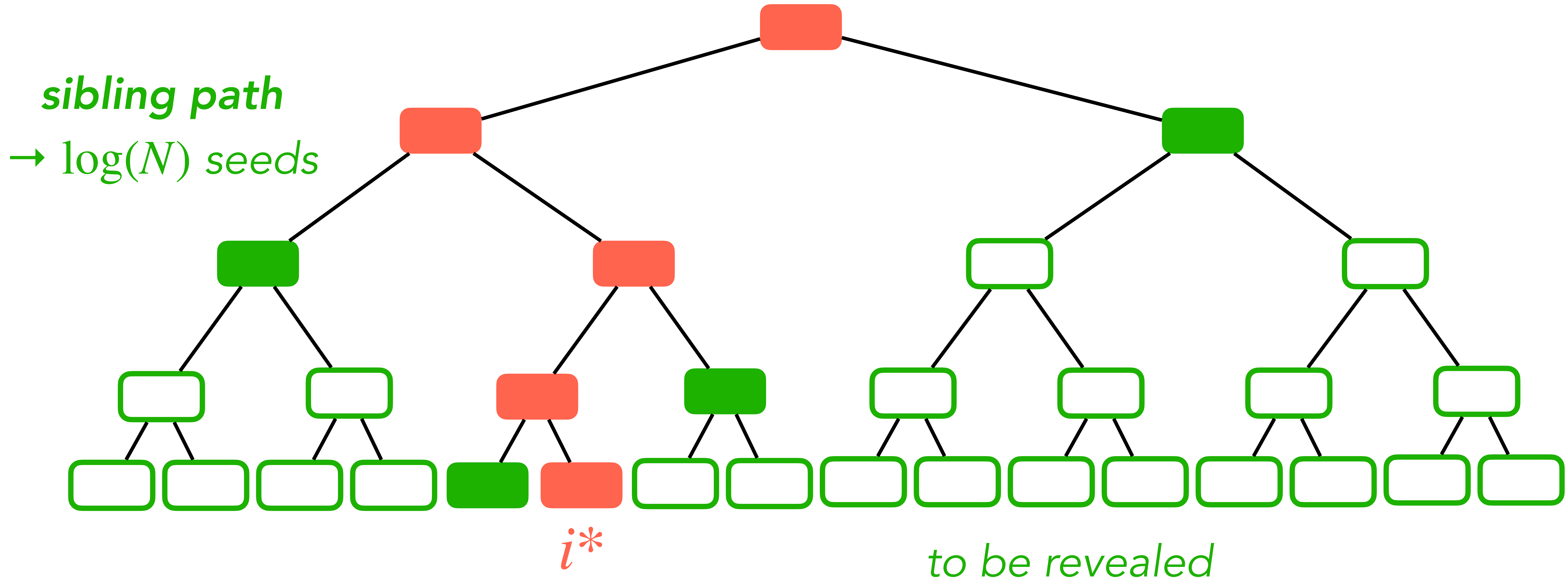
Second optimisation: seed trees



Second optimisation: seed trees



Second optimisation: seed trees



Second optimisation: seed trees

- **Signature = transcript P → V**
  - $\{\mathrm{Com}^{\rho_i}([\![x]\!]_i)\}$ → ~~N commitments~~ → hash +1 commitment
  - $[\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N$ → ~~N MPC broadcasts~~ → hash (+1 MPC broadcast)
  - $\{[\![x]\!]_i, \rho_i\}_{i \neq i^*}$ → ~~$N-1$ input shares + random tapes~~ → $\log(N)$ seeds + $[\![x]\!]_N$ if $i^* \neq N$
- Verification
  - Sibling path → $\{\mathsf{seed}_i\}_{i \neq i^*}$
  - seed_i → $([\![x]\!]_i, \rho_i)$ $\forall\, i \neq i^*$
  - …

Optimising computation: hypercube technique

- **[AGHHJY23]** Aguilar Melchor, Gama, Howe, Hülsing, Joseph, Yue. "The Return of the SDitH" (EUROCRYPT 2023)
- High-level principle
  - Apply MPC computation to sums of shares

$$\Sigma_{i \in I} [\![x_i]\!] \xrightarrow{\varphi} \Sigma_{i \in I} [\![\alpha_i]\!]$$

  - Only $\log N + 1$ such party computations necessary for the prover
  - Only $\log N$ for the verifier
- See Nicolas' talk at EC: https://youtu.be/z6nE4fOWvZA (49:33)

MPCitH with threshold LSSS

Background: Shamir's secret sharing

- Sharing $[\![x]\!] = ([\![x]\!]_1, \ldots, [\![x]\!]_N)$ such that
  - Let $(r_1, \ldots, r_\ell) \leftarrow \$$
  - Let P the polynomial of coefficients $(x, r_1, \ldots, r_\ell)$

$$\begin{cases} [\![x]\!]_1 = P(f_1) \\ \quad \vdots \\ [\![x]\!]_N = P(f_N) \end{cases}$$

with $f_1, \ldots, f_N \in \mathbb{F}$ distinct field elements

Background: Shamir's secret sharing

- Sharing $[\![x]\!] = ([\![x]\!]_1, \ldots, [\![x]\!]_N)$ such that
  - Let $(r_1, \ldots, r_\ell) \leftarrow \$$
  - Let P the polynomial of coefficients $(x, r_1, \ldots, r_\ell)$

$$\begin{cases} [\![x]\!]_1 = P(f_1) \\ \vdots \\ [\![x]\!]_N = P(f_N) \end{cases}$$

with $f_1, \ldots, f_N \in \mathbb{F}$ distinct field elements

- $(\ell + 1, N)$-threshold linear secret sharing scheme (LSSS)

Background: Shamir's secret sharing

- Sharing $[\![x]\!] = ([\![x]\!]_1, \ldots, [\![x]\!]_N)$ such that
  - Let $(r_1, \ldots, r_\ell) \leftarrow \$$
  - Let P the polynomial of coefficients $(x, r_1, \ldots, r_\ell)$

$$\begin{cases} [\![x]\!]_1 = P(f_1) \\ \quad\vdots \\ [\![x]\!]_N = P(f_N) \end{cases} \quad \text{with } f_1, \ldots, f_N \in \mathbb{F} \text{ distinct field elements}$$

- $(\ell + 1, N)$-threshold linear secret sharing scheme (LSSS)
  - Linearity: $[\![x]\!] + [\![y]\!] = [\![x + y]\!]$

Background: Shamir's secret sharing

- Sharing $[\![x]\!] = ([\![x]\!]_1, \ldots, [\![x]\!]_N)$ such that
  - Let $(r_1, \ldots, r_\ell) \leftarrow \$$
  - Let P the polynomial of coefficients $(x, r_1, \ldots, r_\ell)$

$$\begin{cases} [\![x]\!]_1 = P(f_1) \\ \quad\vdots \\ [\![x]\!]_N = P(f_N) \end{cases} \quad \text{with } f_1, \ldots, f_N \in \mathbb{F} \text{ distinct field elements}$$

- $(\ell + 1, N)$-**threshold** linear secret sharing scheme (LSSS)
  - Linearity: $[\![x]\!] + [\![y]\!] = [\![x + y]\!]$
  - Any set of ℓ shares is random and independent of x
  - Any set of $\ell + 1$ shares → coefficients $(x, r_1, \ldots, r_\ell)$ → all the shares

Background: Shamir's secret sharing

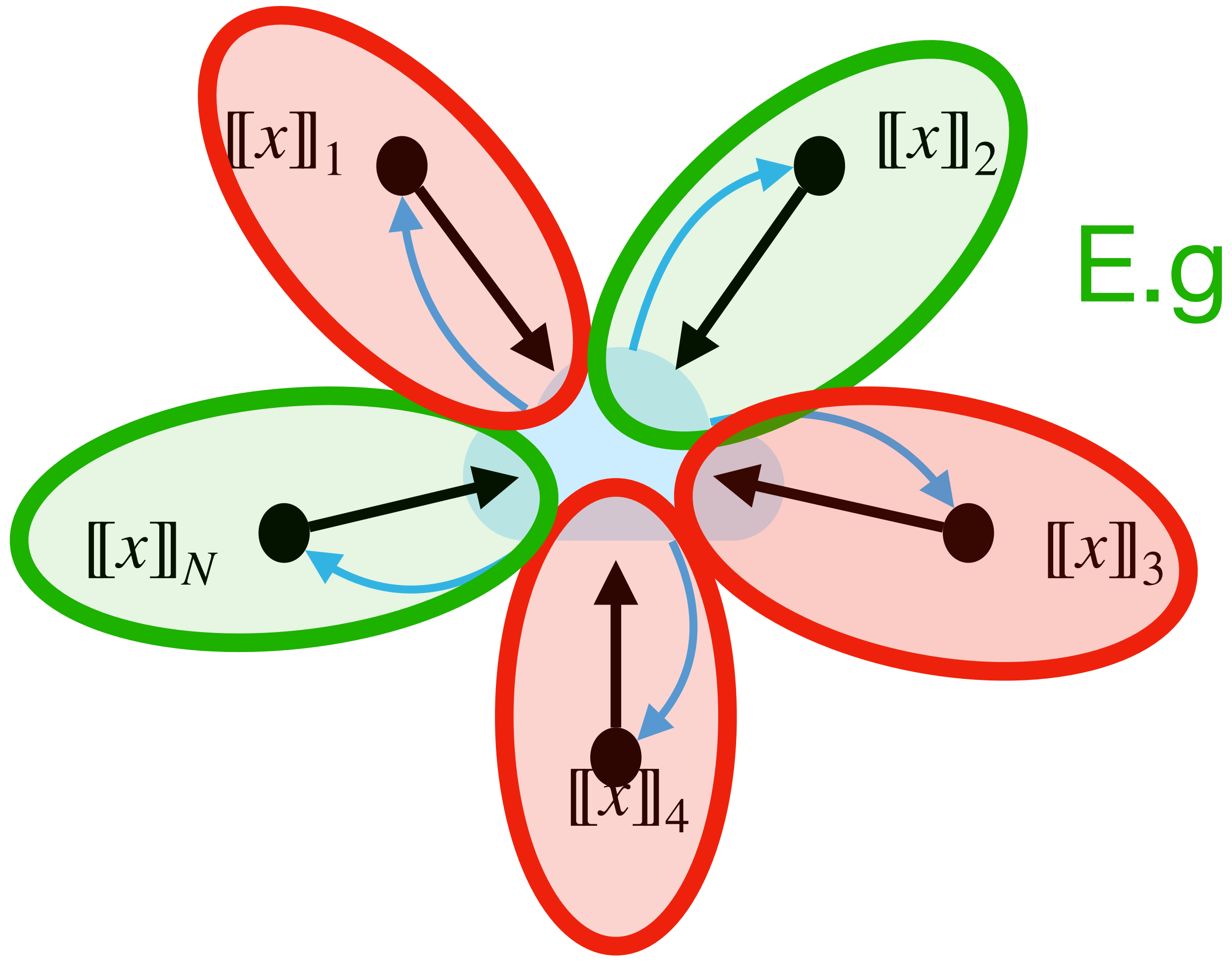
- Sharing $[\![x]\!] = ([\![x]\!]_1, \ldots, [\![x]\!]_N)$ such that
  - Let $(r_1, \ldots, r_\ell) \leftarrow \$$
  - Let P the polynomial of coefficients $(x, r_1, \ldots, r_\ell)$

$$\begin{cases} [\![x]\!]_1 = P(f_1) \\ \quad\vdots \\ [\![x]\!]_N = P(f_N) \end{cases} \quad \text{with } f_1, \ldots, f_N \in \mathbb{F} \text{ distinct field elements}$$

- $(\ell + 1, N)$-**threshold** linear secret sharing scheme (LSSS)
  - Linearity: $[\![x]\!] + [\![y]\!] = [\![x + y]\!]$
  - Any set of ℓ shares is random and independent of x
  - Any set of $\ell + 1$ shares → coefficients $(x, r_1, \ldots, r_\ell)$ → all the shares
  - $[\![x]\!] = ([\![x]\!]_1, \ldots, [\![x]\!]_N)$ is a **Reed-Solomon codeword** of $(x, r_1, \ldots, r_\ell)$

MPCitH with threshold LSSS

- **[FR22]** Feneuil, Rivain. "Threshold Linear Secret Sharing to the Rescue of MPC-in-the-Head" (ePrint 2022)
- ZK property ⇒ only open ℓ parties
  - Verifier challenges a set $I \subseteq \{1, \ldots, N\}$ s.t. $|I| = \ell$
  - Prover opens $\{[\![x]\!]_i, \rho_i\}_{i \in I}$



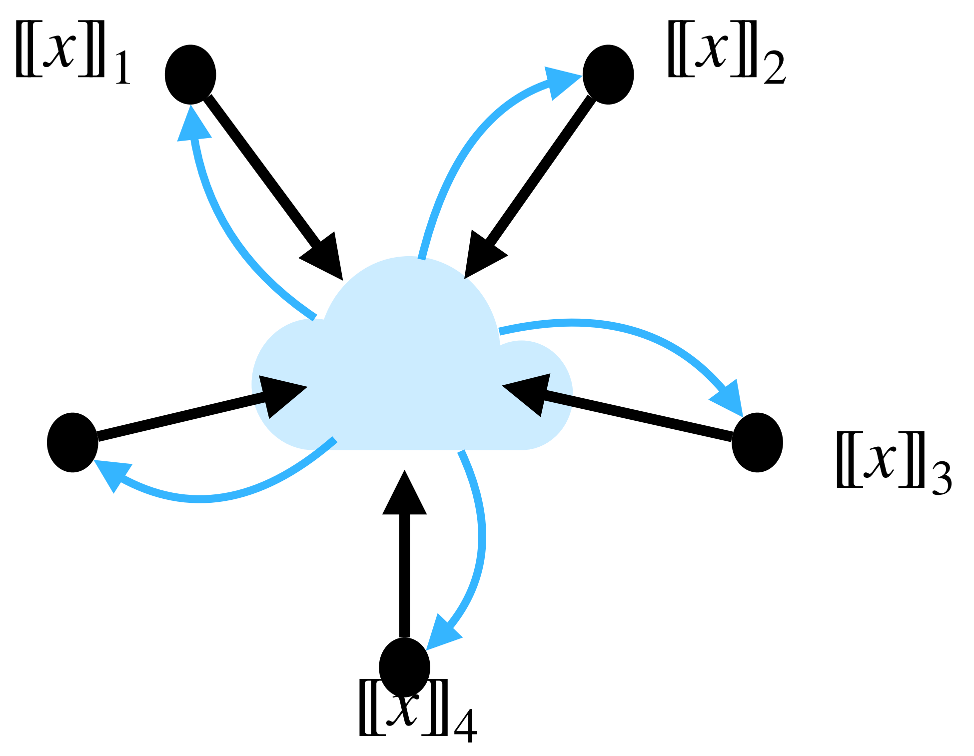
E.g. $\ell = 2$

MPCitH transform with threshold LSSS

① Generate and commit shares

$[\![x]\!] = ([\![x]\!]_1, \ldots, [\![x]\!]_N)$

② Run MPC in their head



④ Open parties in I

$\text{Com}^{\rho_1}([\![x]\!]_1)$

$\cdots$

$\text{Com}^{\rho_N}([\![x]\!]_N)$ →

send broadcast

$[\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N$ →

← I

$([\![x]\!]_i, \rho_i)_{i \in I}$ →

③ Chose random set of parties

$I \subseteq \{1,\ldots,N\}$, s.t. $|I| = \ell$

⑤ Check $\forall i \in I$

- Commitments $\text{Com}^{\rho_i}([\![x]\!]_i)$
- MPC computation $[\![\alpha]\!]_i = \varphi([\![x]\!]_i)$

Check $g(y, \alpha) = \text{Accept}$

Prover

Verifier

MPCitH transform with threshold LSSS

① Generate and commit shares

$[\![x]\!] = ([\![x]\!]_1, \ldots, [\![x]\!]_N)$

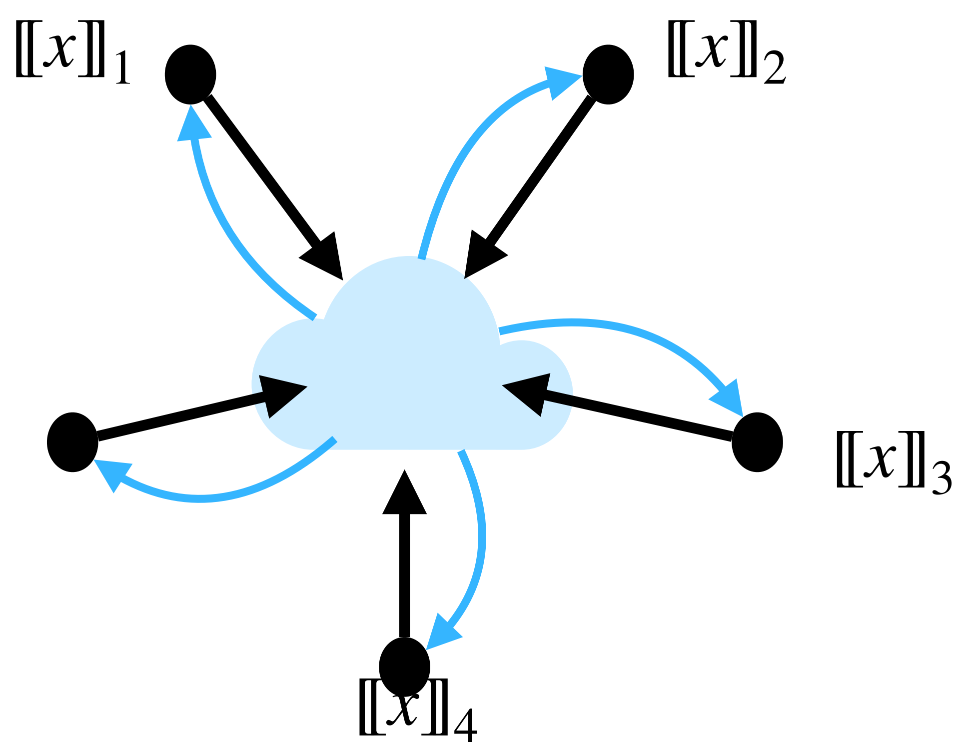
$\mathrm{Com}^{\rho_1}([\![x]\!]_1)$

$\cdots$

$\mathrm{Com}^{\rho_N}([\![x]\!]_N)$

Threshold LSSS ⇒ cannot generate shares from seeds

② Run MPC in their head



send broadcast

$[\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N$

③ Chose random set of parties

$I \subseteq \{1,\ldots,N\}$, s.t. $|I| = \ell$

I

④ Open parties in I

$([\![x]\!]_i, \rho_i)_{i \in I}$

⑤ Check $\forall i \in I$

- Commitments $\mathrm{Com}^{\rho_i}([\![x]\!]_i)$
- MPC computation $[\![\alpha]\!]_i = \varphi([\![x]\!]_i)$

Check $g(y, \alpha) =$ Accept

Prover

Verifier

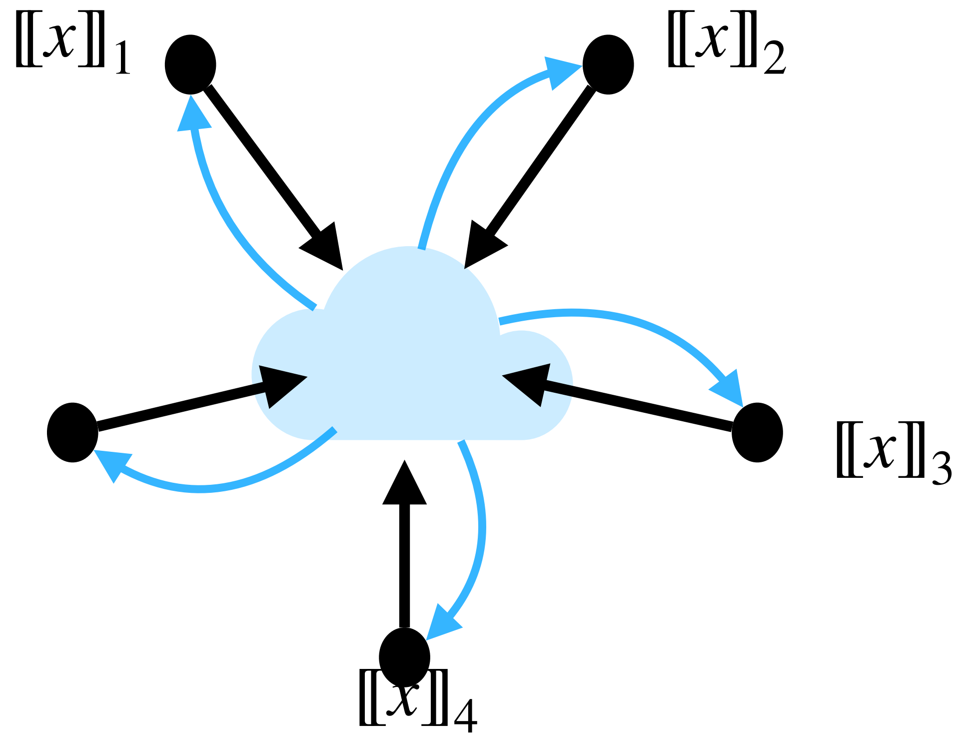
MPCitH transform with threshold LSSS

① Generate and commit shares
$[\![x]\!] = ([\![x]\!]_1, \ldots, [\![x]\!]_N)$

$\mathrm{Com}^{\rho_1}([\![x]\!]_1)$
$\cdots$
$\mathrm{Com}^{\rho_N}([\![x]\!]_N)$

Threshold LSSS $\Rightarrow$ cannot generate shares from seeds

② Run MPC in their head



send broadcast
$[\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N$

$[\![\alpha]\!]$ is an RS codeword $\Rightarrow \ell + 1$ shares fully determine the sharing

③ Chose random set of parties
$I \subseteq \{1, \ldots, N\}$, s.t. $|I| = \ell$

I

④ Open parties in I

$([\![x]\!]_i, \rho_i)_{i \in I}$

⑤ Check $\forall i \in I$
- Commitments $\mathrm{Com}^{\rho_i}([\![x]\!]_i)$
- MPC computation $[\![\alpha]\!]_i = \varphi([\![x]\!]_i)$

Check $g(y, \alpha) =$ Accept

Prover

Verifier

MPCitH transform with threshold LSSS

① Generate and commit shares

$[\![x]\!] = ([\![x]\!]_1, \ldots, [\![x]\!]_N)$

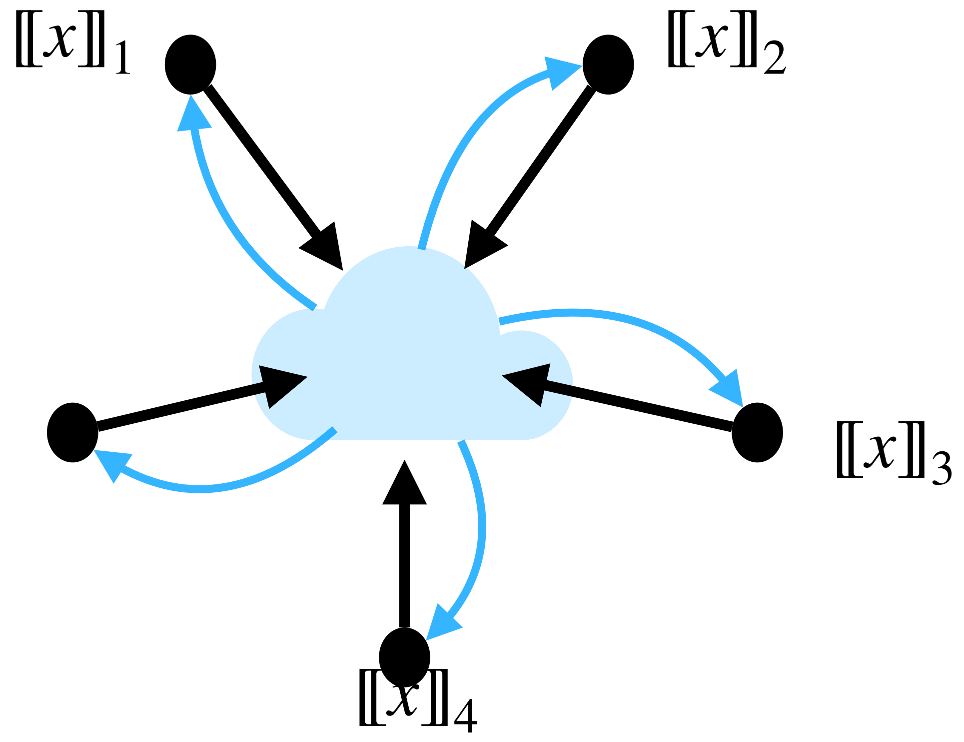
$\mathrm{Com}^{\rho_1}([\![x]\!]_1)$

$\cdots$

$\mathrm{Com}^{\rho_N}([\![x]\!]_N)$

Threshold LSSS ⇒ cannot generate shares from seeds

② Run MPC in their head



send broadcast $[\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N$

$[\![\alpha]\!]$ is an RS codeword ⇒ $\ell + 1$ shares fully determine the sharing

③ Chose random set of parties $I \subseteq \{1, \ldots, N\}$, s.t. $|I| = \ell$

I

④ Open parties in I

$([\![x]\!]_i, \rho_i)_{i \in I}$

⑤ Check $\forall i \in I$

- Commitments $\mathrm{Com}^{\rho_i}([\![x]\!]_i)$
- MPC computation $[\![\alpha]\!]_i = \varphi([\![x]\!]_i)$

Check $g(y, \alpha) =$ Accept

⇒ only $\ell + 1$ party computations required

Prover

Verifier

MPCitH transform with threshold LSSS

① Generate and commit shares

$[\![x]\!] = ([\![x]\!]_1, \ldots, [\![x]\!]_N)$

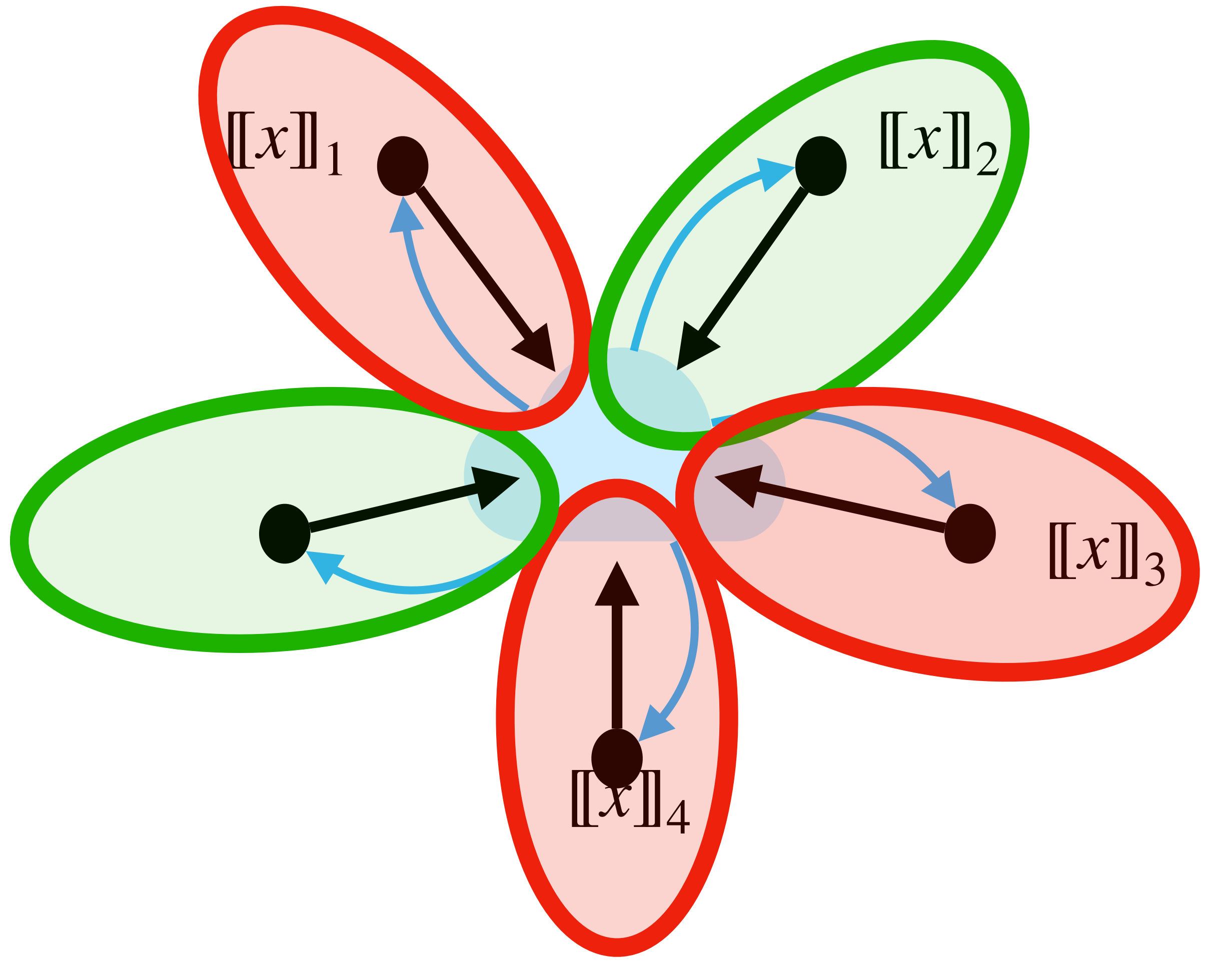
$\mathrm{Com}^{\rho_1}([\![x]\!]_1)$

$\cdots$

$\mathrm{Com}^{\rho_N}([\![x]\!]_N)$

Threshold LSSS $\Rightarrow$ cannot generate shares from seeds

② Run MPC in their head



send broadcast

$[\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N$

$[\![\alpha]\!]$ is an RS codeword $\Rightarrow \ell + 1$ shares fully determine the sharing

③ Chose random set of parties $I \subseteq \{1,\ldots,N\}$, s.t. $|I| = \ell$

I

④ Open parties in I

$([\![x]\!]_i, \rho_i)_{i \in I}$

⑤ Check $\forall i \in I$

- Commitments $\mathrm{Com}^{\rho_i}([\![x]\!]_i)$
- MPC computation $[\![\alpha]\!]_i = \varphi([\![x]\!]_i)$

Check $g(y, \alpha) = $ Accept

Prover

ℓ parties opened instead of $N - 1$

$\Rightarrow$ only $\ell + 1$ party computations required

Verifier

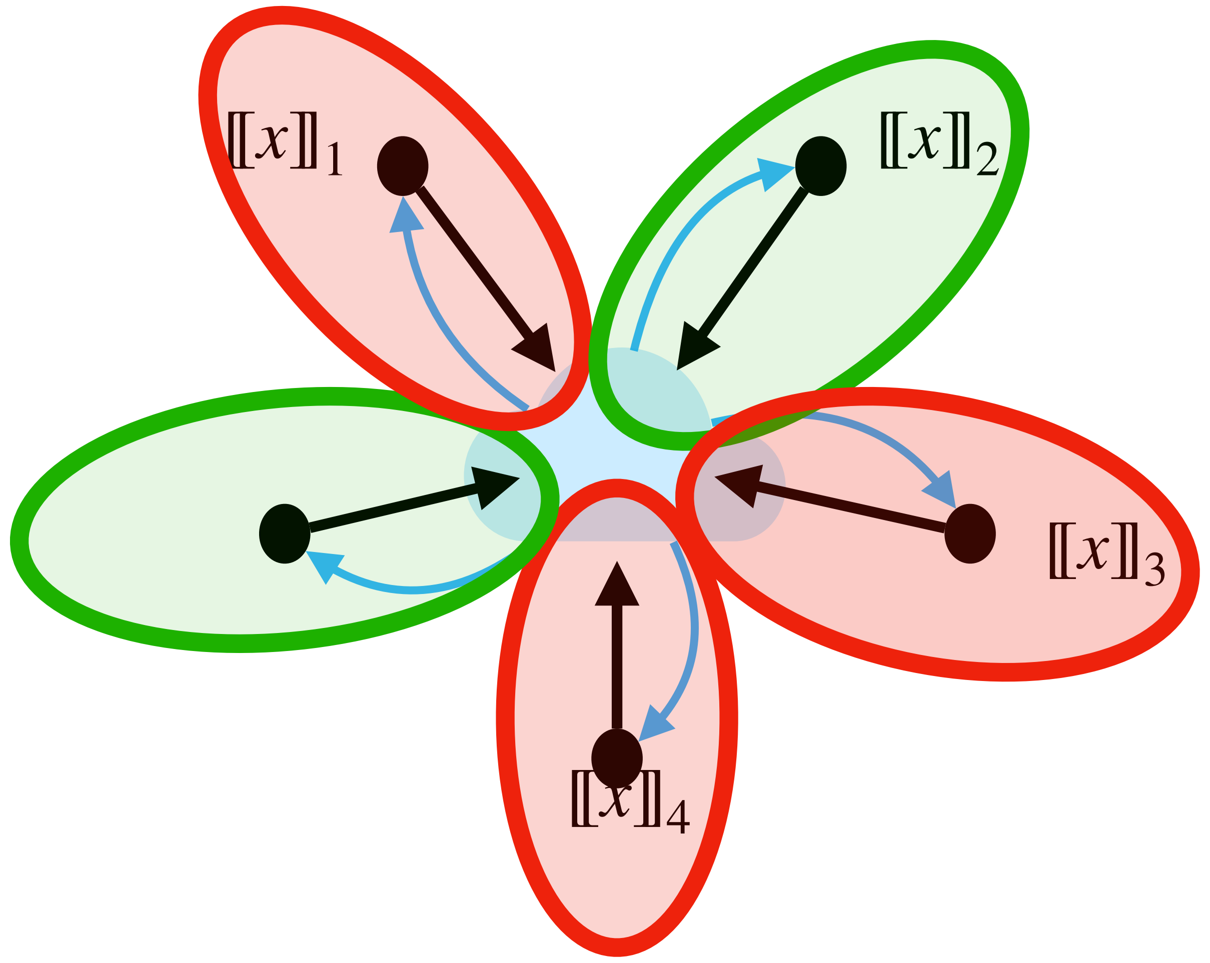
MPCitH transform with threshold LSSS

① Generate and commit shares
$[\![x]\!] = ([\![x]\!]_1, \ldots, [\![x]\!]_N)$

$\mathrm{Com}^{\rho_1}([\![x]\!]_1)$
…
$\mathrm{Com}^{\rho_N}([\![x]\!]_N)$

Threshold LSSS ⇒ cannot generate shares from seeds

② Run MPC in their head



send broadcast
$[\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N$

$[\![\alpha]\!]$ is an RS codeword ⇒ $\ell + 1$ shares fully determine the sharing

③ Chose random set of parties
$I \subseteq \{1,\ldots,N\}$, s.t. $|I| = \ell$

I

④ Open parties in I

$([\![x]\!]_i, \rho_i)_{i \in I}$

⑤ Check $\forall i \in I$

- Commitments $\mathrm{Com}^{\rho_i}([\![x]\!]_i)$
- MPC computation $[\![\alpha]\!]_i = \varphi([\![x]\!]_i)$

Check $g(y, \alpha) =$ Accept

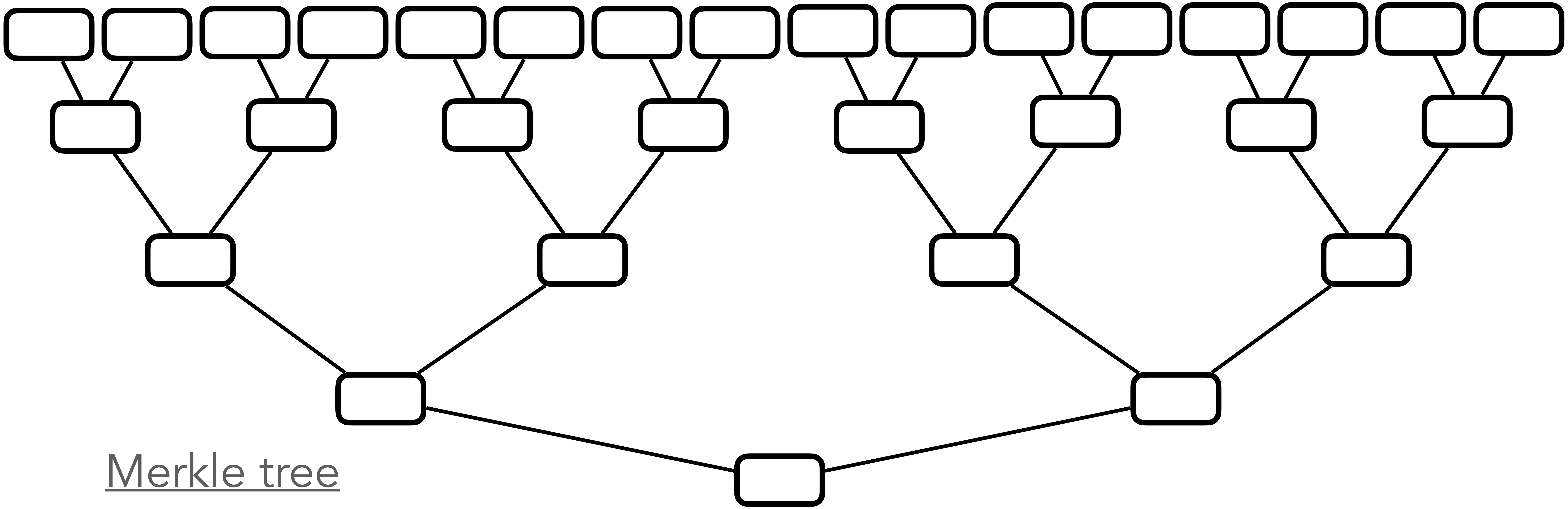
<u>Prover</u>

ℓ parties opened instead of $N - 1$

⇒ only $\ell + 1$ party computations required

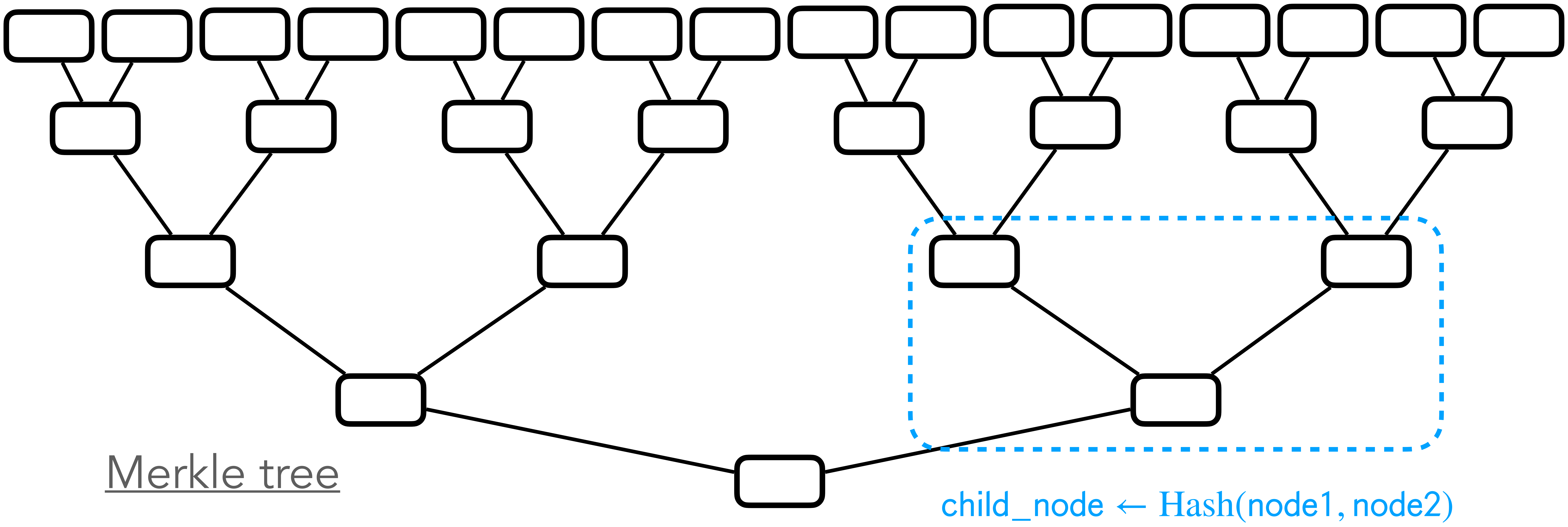
only ℓ party computations required

Sharing and commitments

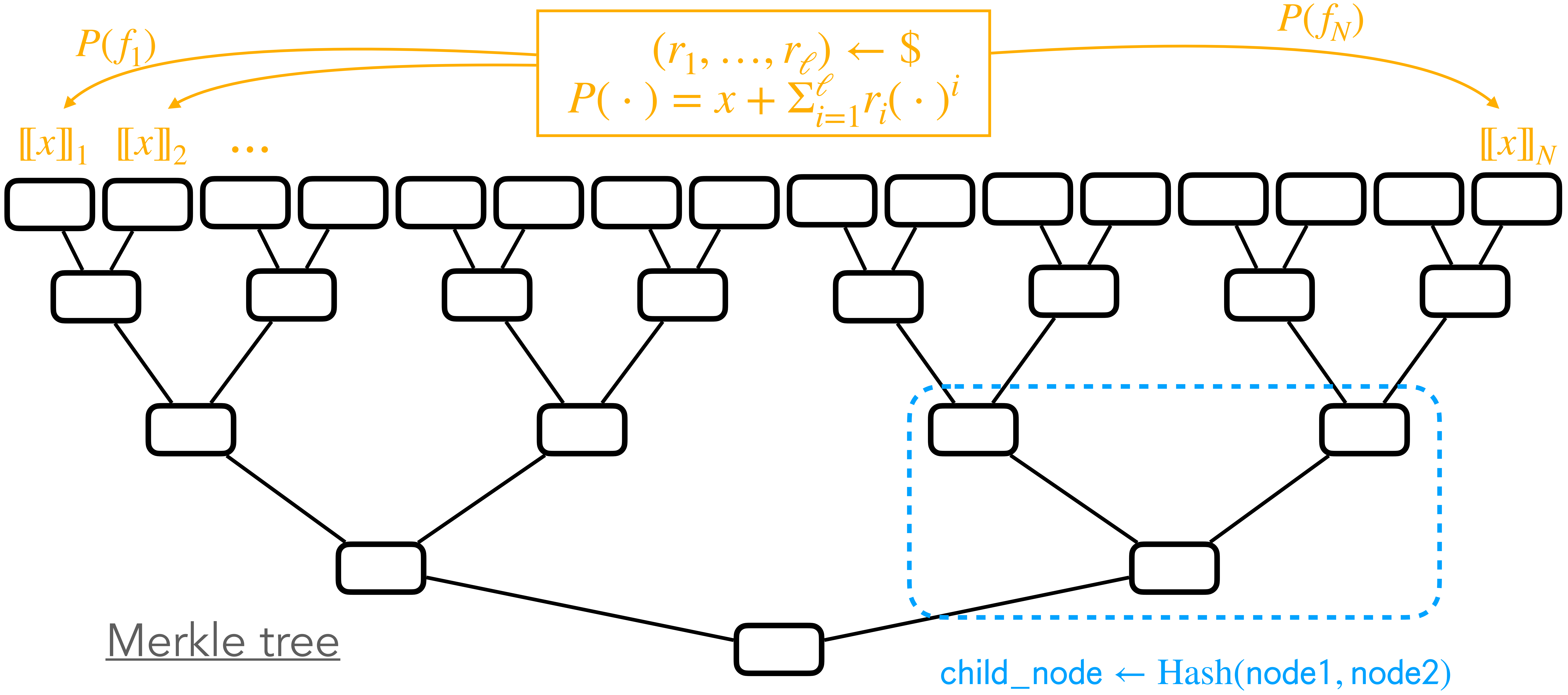


Merkle tree

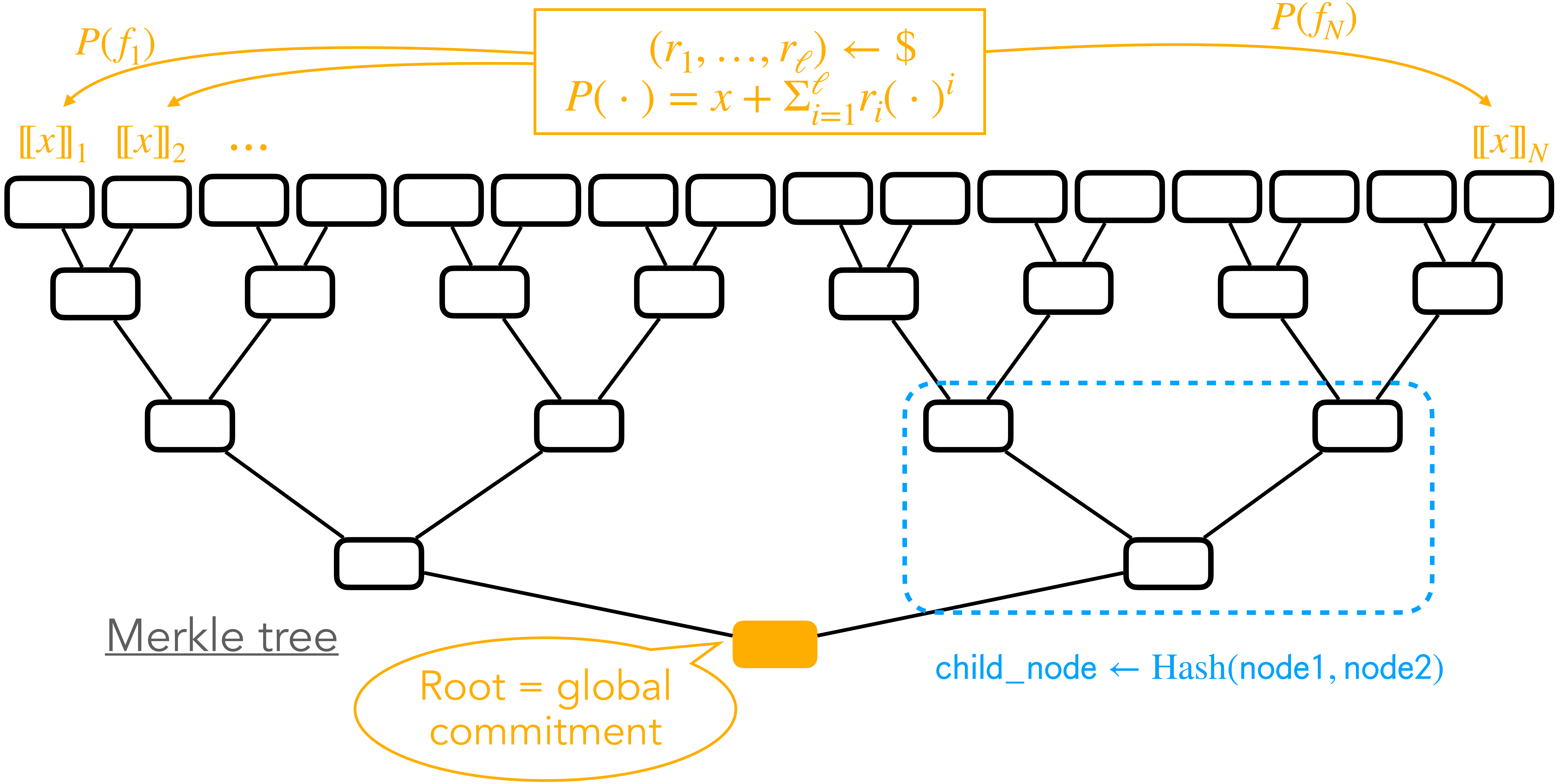
Sharing and commitments



Sharing and commitments

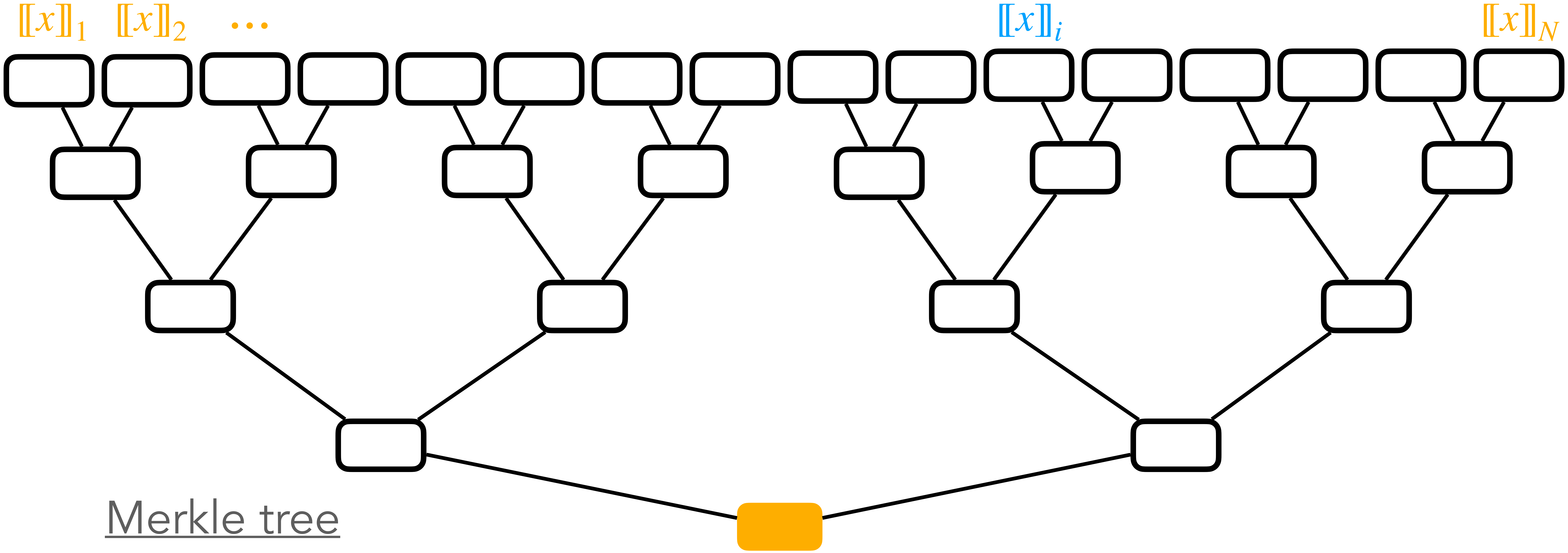


Sharing and commitments



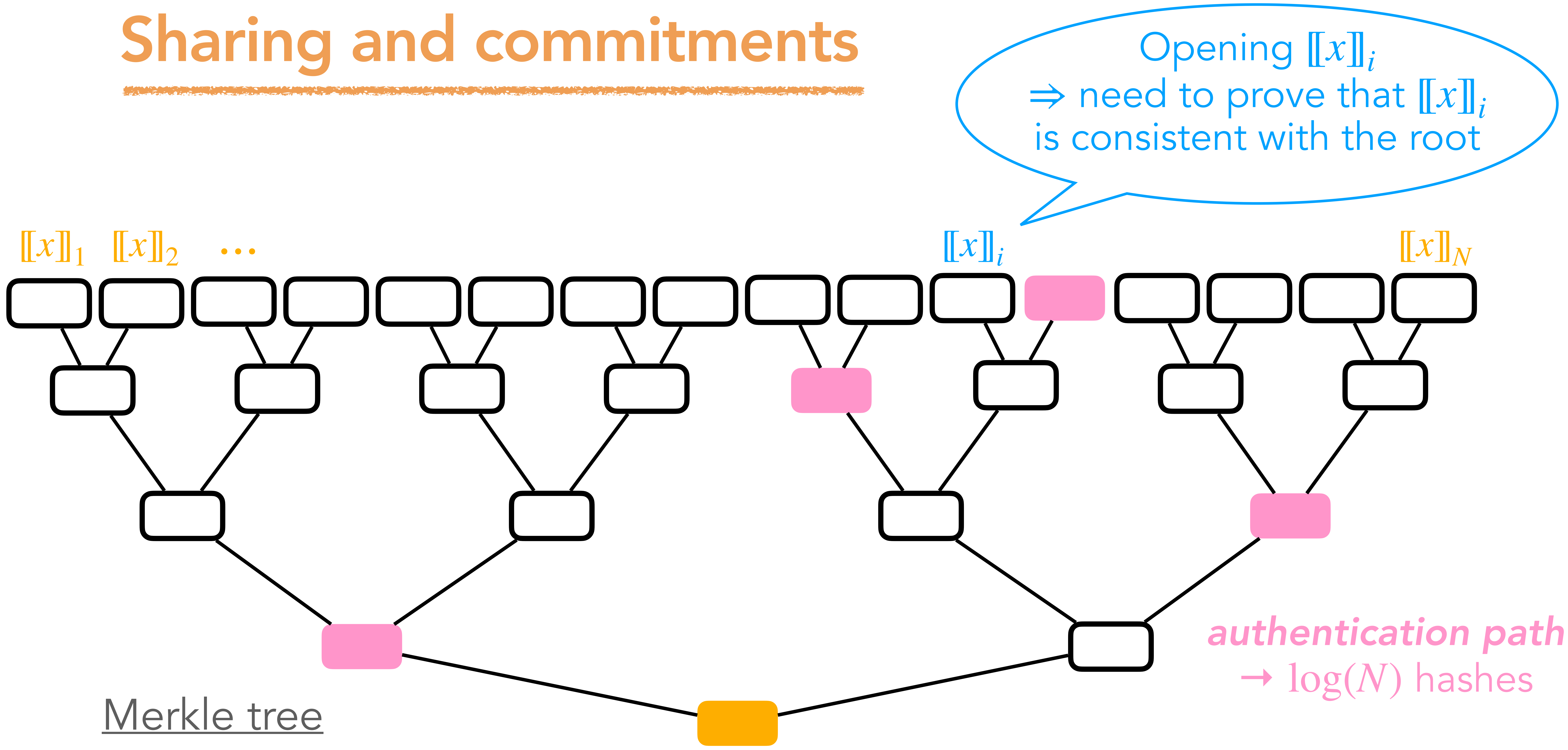
Sharing and commitments

Opening $[\![x]\!]_i$
$\Rightarrow$ need to prove that $[\![x]\!]_i$ is consistent with the root

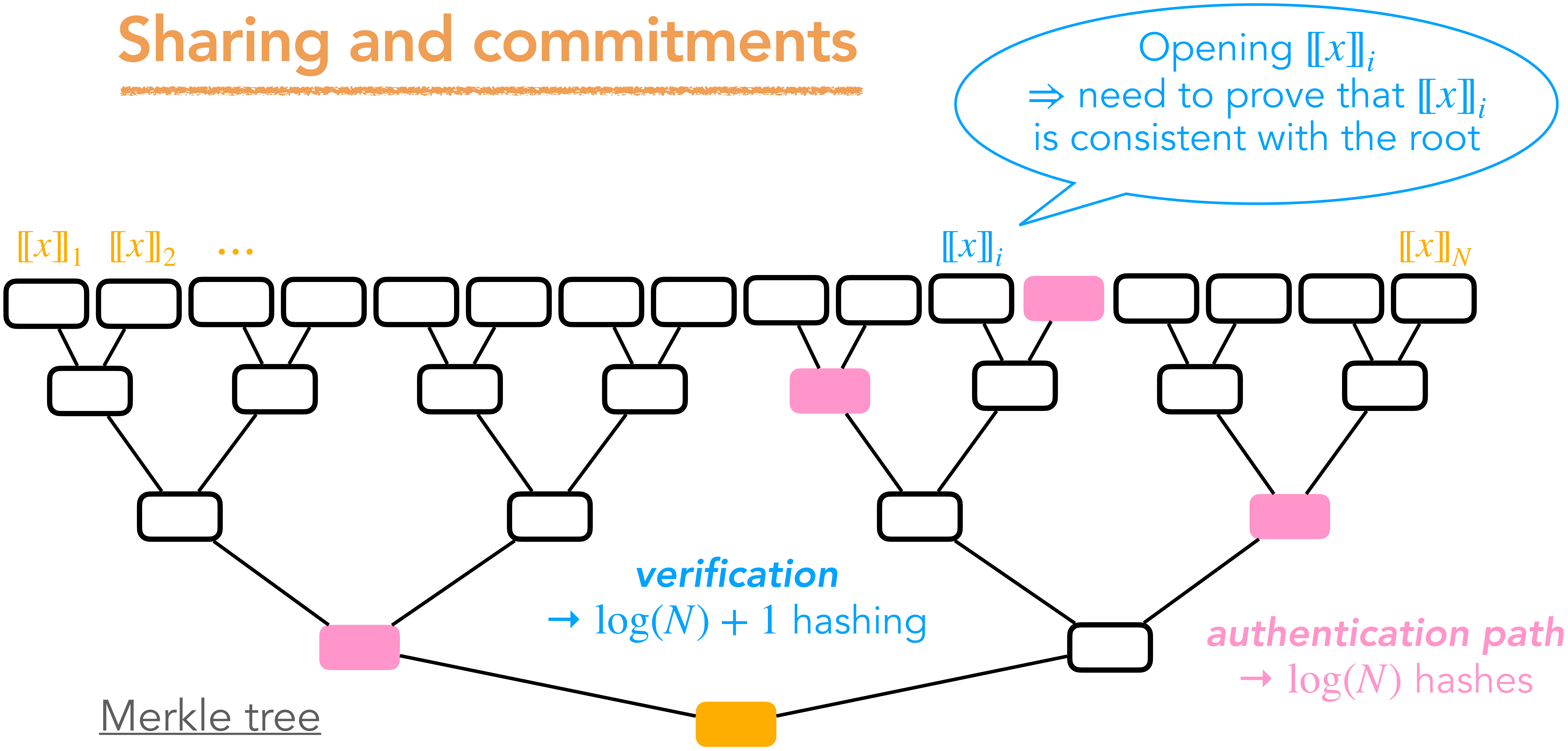


Merkle tree

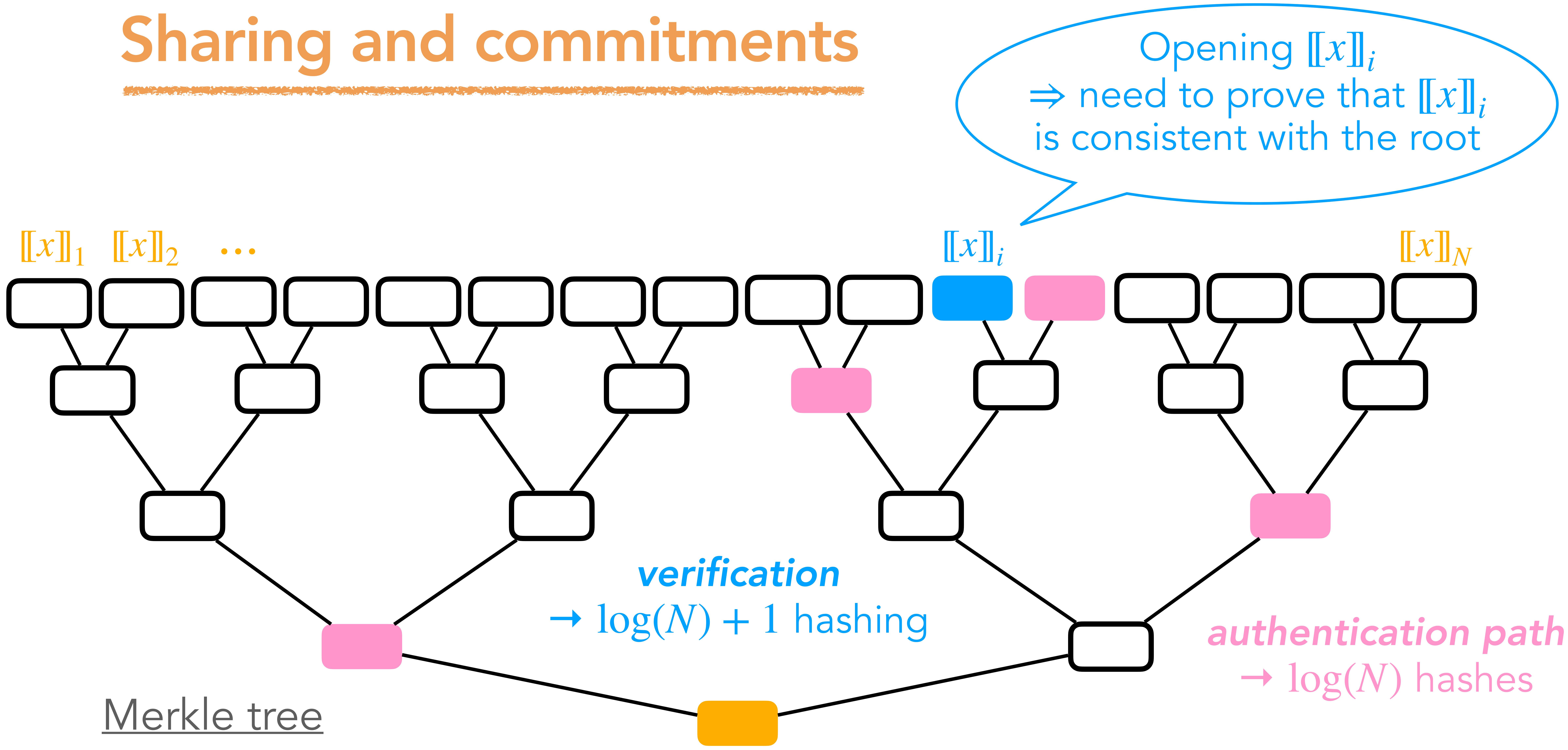
Sharing and commitments



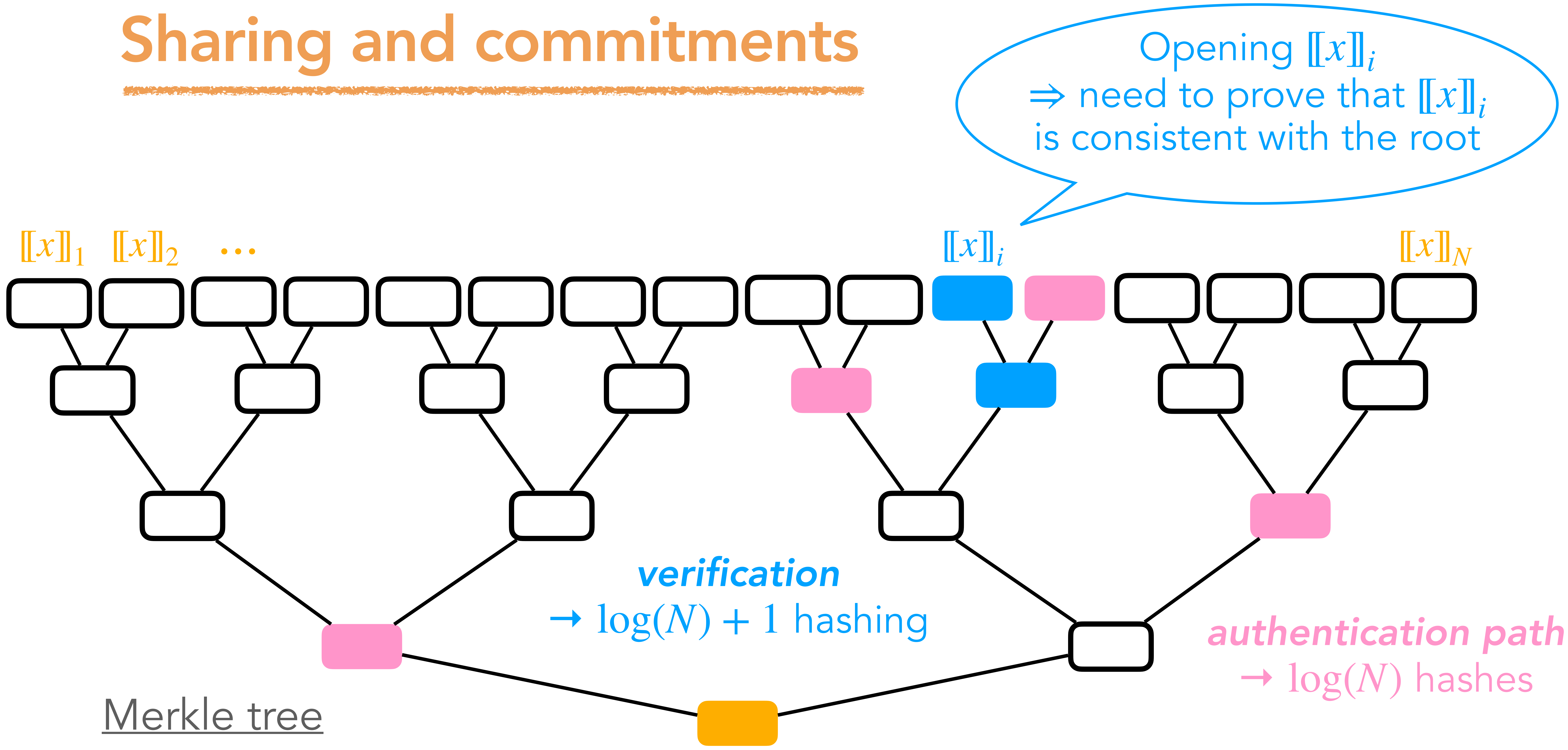
Sharing and commitments



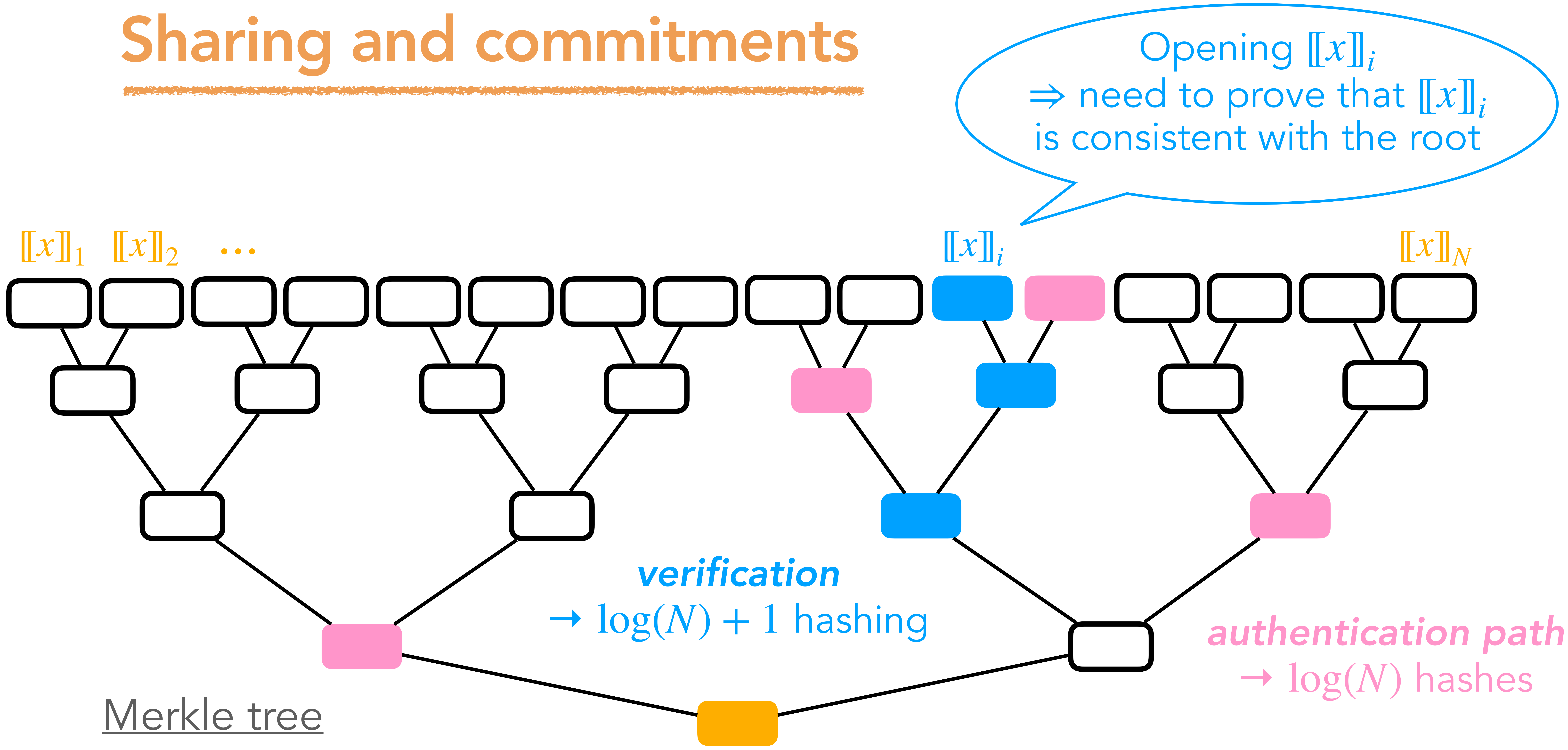
Sharing and commitments



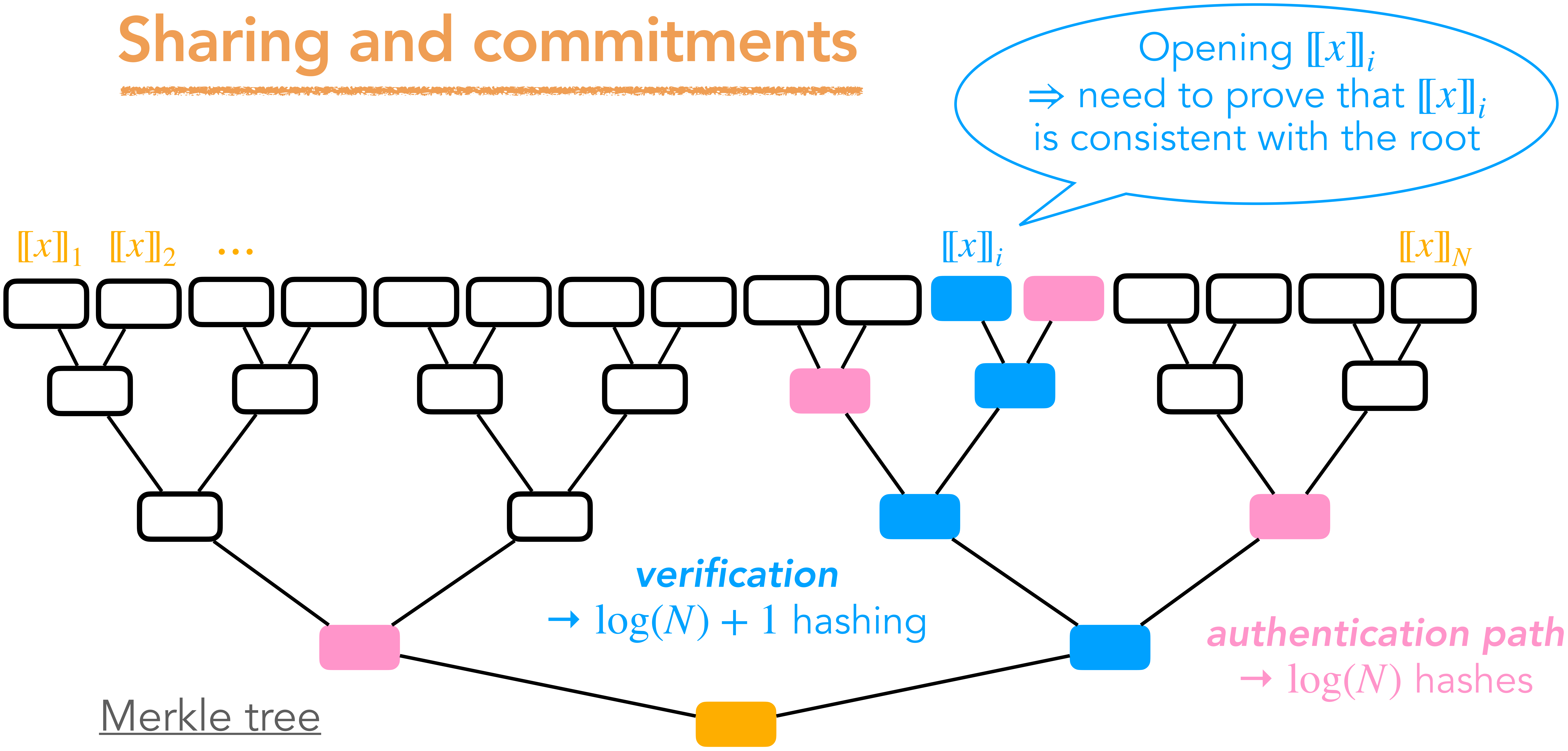
Sharing and commitments



Sharing and commitments



Sharing and commitments

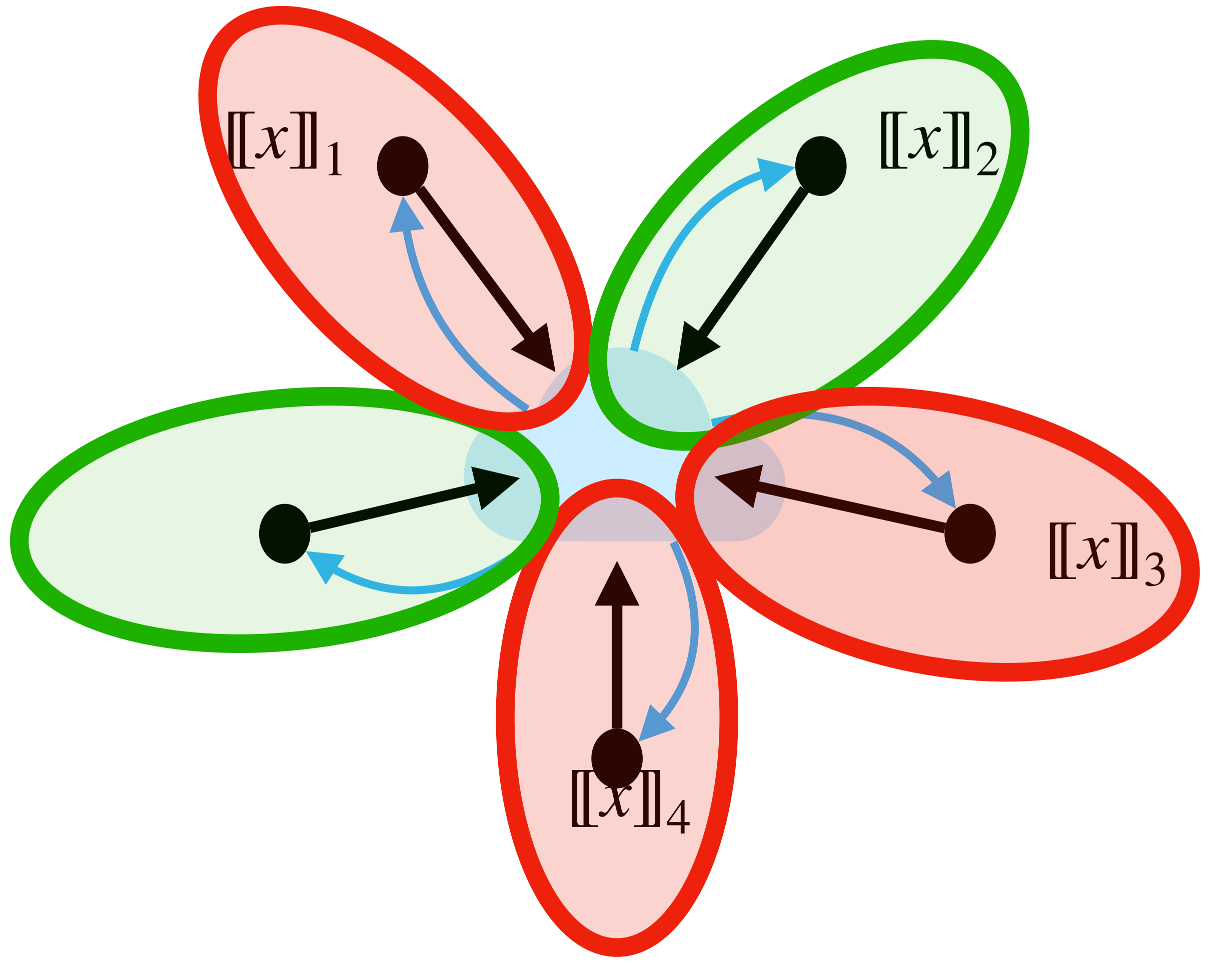


MPCitH transform with threshold LSSS

① Generate and commit shares

$[\![x]\!] = ([\![x]\!]_1, \ldots, [\![x]\!]_N)$

② Run MPC in their head



④ Open parties in I

Merkle root →

send broadcast
$[\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N$ →

← I

$\{[\![x]\!]_i, \text{auth}_i\}_{i \in I}$ →

authentication path

③ Chose random set of parties
$I \subseteq \{1, \ldots, N\}$, s.t. $|I| = \ell$

⑤ Check $\forall i \in I$

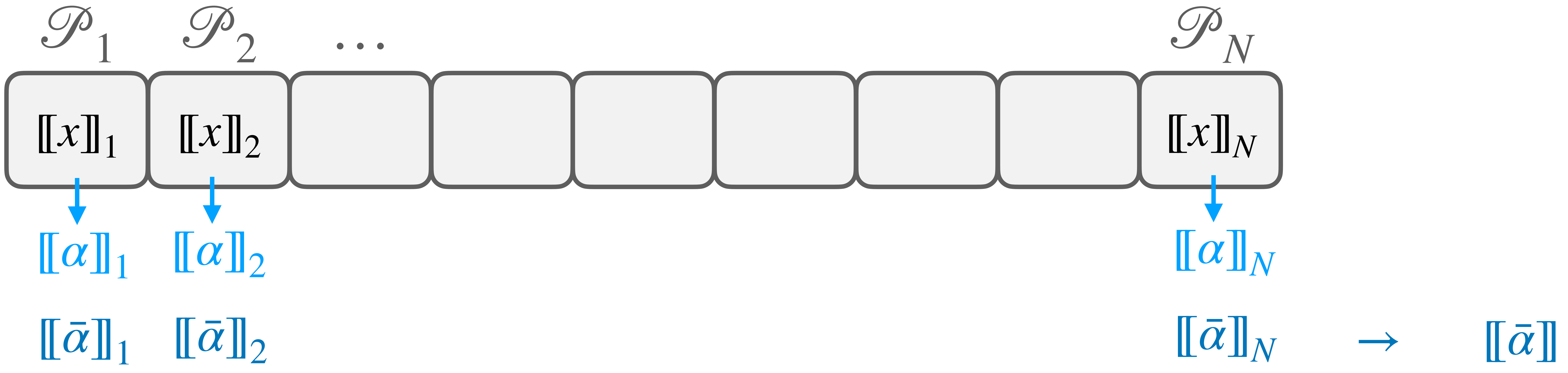
- Commitments $\mathrm{Com}^{\rho_i}([\![x]\!]_i)$
- MPC computation $[\![\alpha]\!]_i = \varphi([\![x]\!]_i)$

Check $g(y, \alpha) =$ Accept

Prover

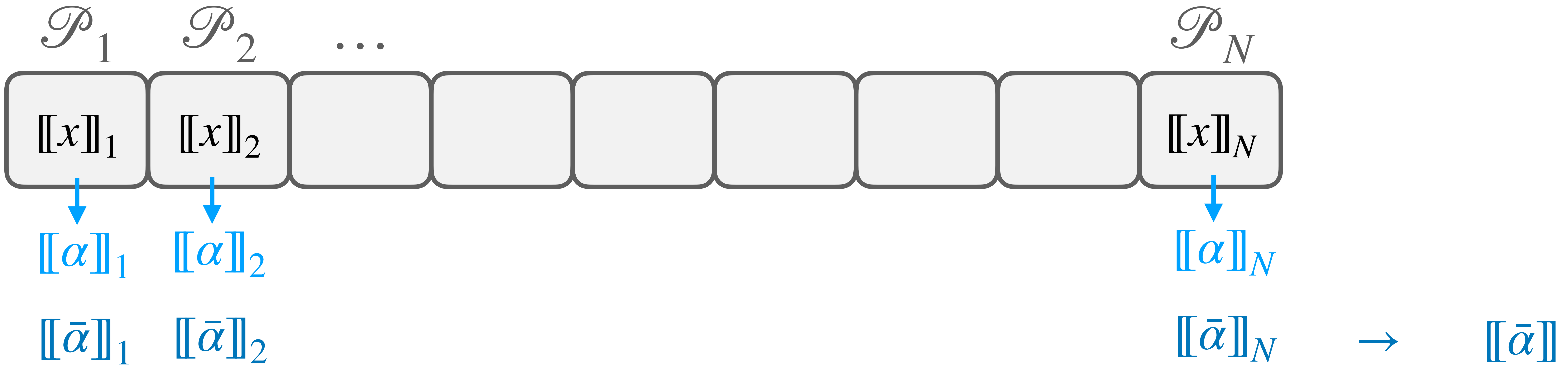
Verifier

Soundness



sharing sent to the verifier s.t.
$g(y, \bar{\alpha}) = \text{Accept}$

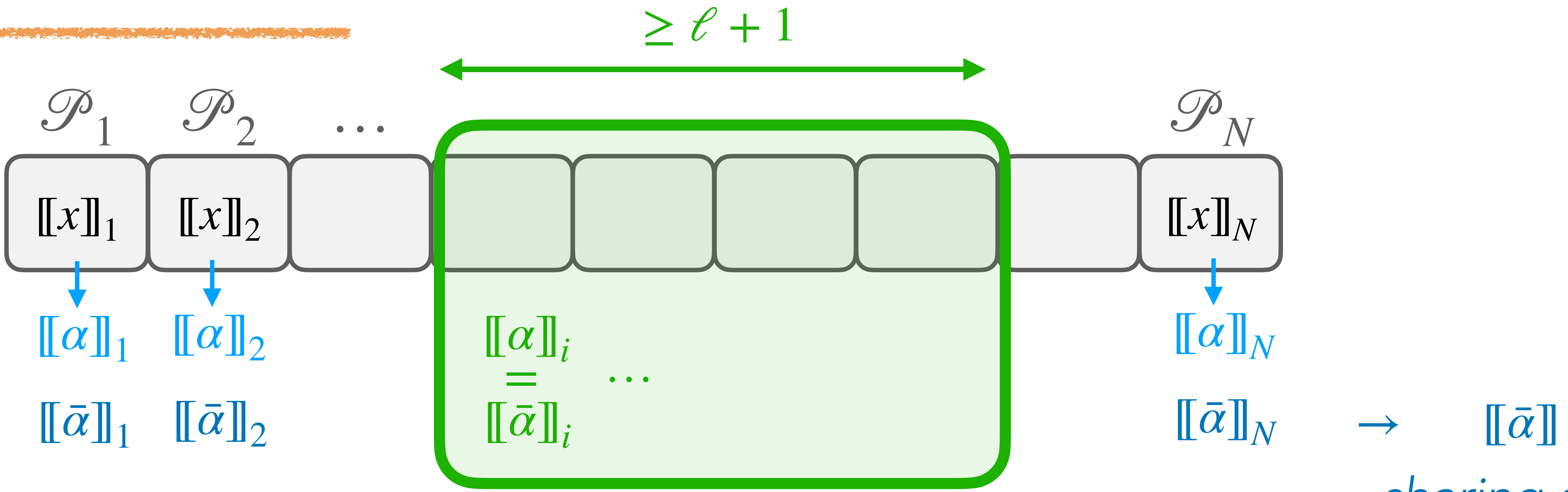
Soundness



sharing sent to the verifier s.t. $g(y, \bar{\alpha}) = \text{Accept}$

- $\mathscr{P}_i$ is "honest" if $[\![\alpha]\!]_i = [\![\bar{\alpha}]\!]_i$

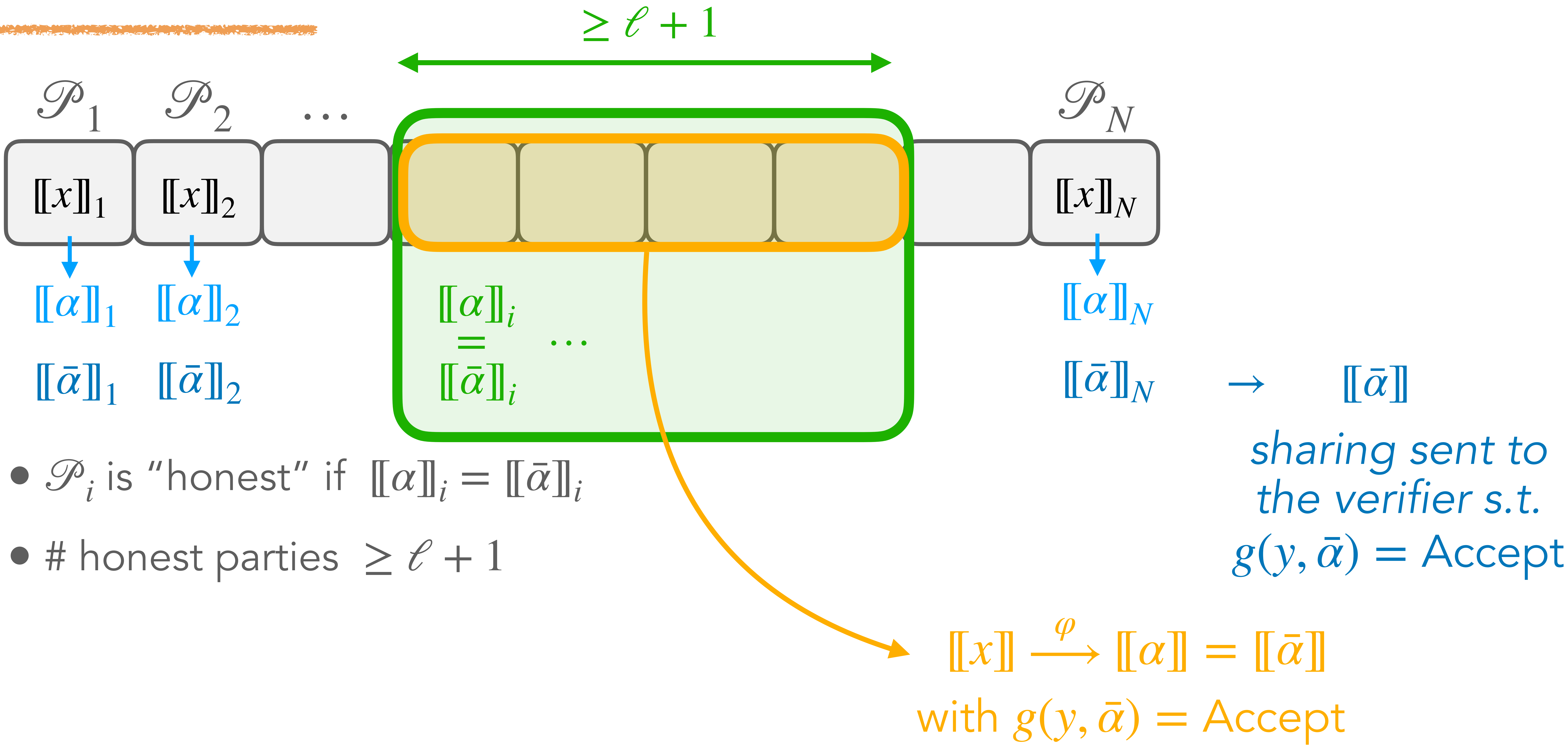
Soundness



- $\mathcal{P}_i$ is "honest" if $[\![\alpha]\!]_i = [\![\bar{\alpha}]\!]_i$
- # honest parties $\geq \ell + 1$

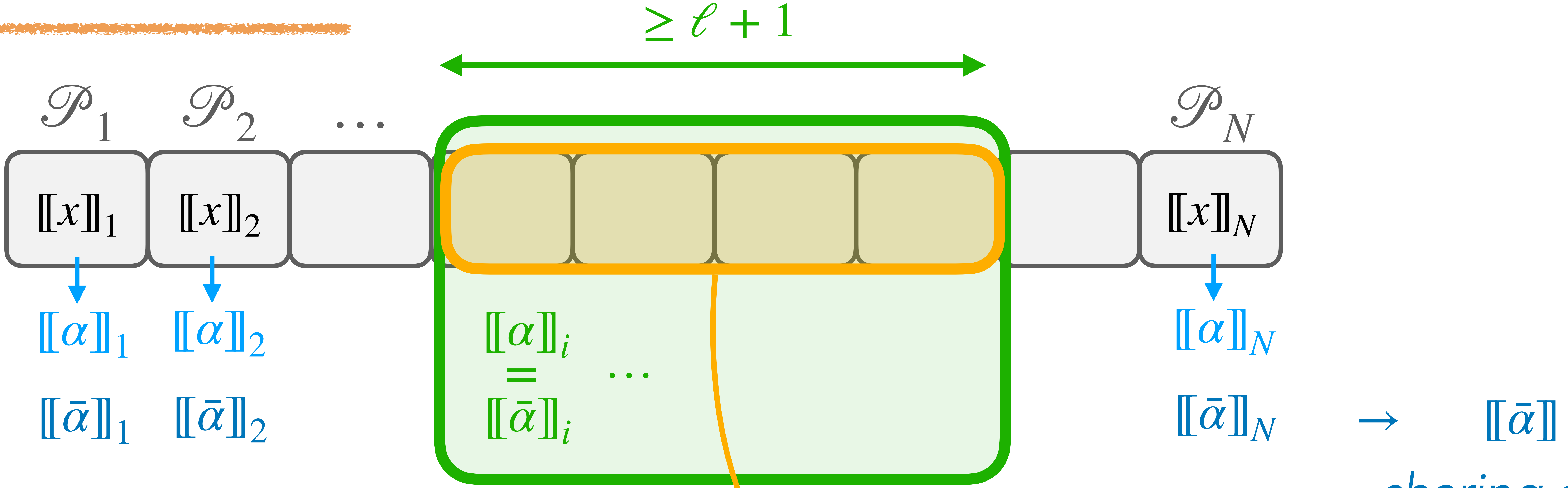
sharing sent to the verifier s.t. $g(y, \bar{\alpha}) =$ Accept

Soundness



- $\mathscr{P}_i$ is "honest" if $[\![\alpha]\!]_i = [\![\bar{\alpha}]\!]_i$
- # honest parties $\geq \ell + 1$

Soundness



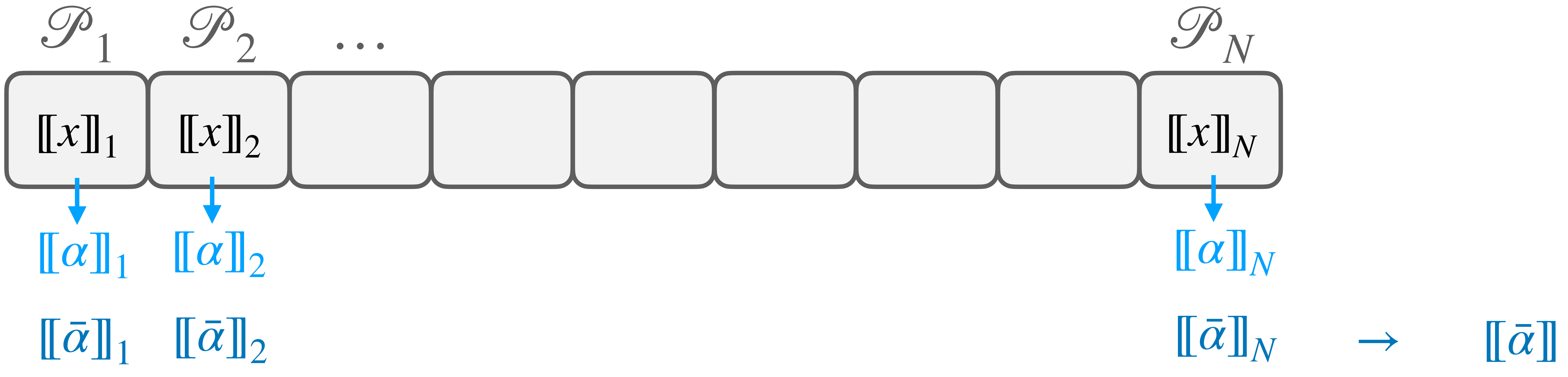
- $\mathscr{P}_i$ is "honest" if $[\![\alpha]\!]_i = [\![\bar{\alpha}]\!]_i$
- # honest parties $\geq \ell + 1$

sharing sent to the verifier s.t.
$g(y, \bar{\alpha}) =$ Accept

$[\![x]\!] \xrightarrow{\varphi} [\![\alpha]\!] = [\![\bar{\alpha}]\!]$
with $g(y, \bar{\alpha}) =$ Accept

$\Leftrightarrow$ $[\![x]\!]$ encodes a genuine x

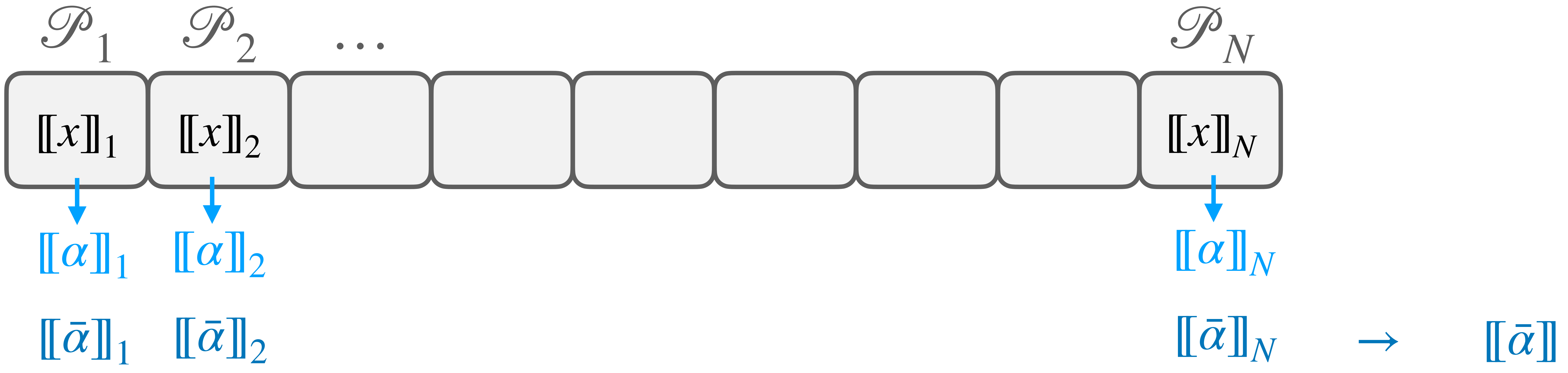
Soundness



sharing sent to the verifier s.t.
$g(y, \bar{\alpha}) =$ Accept

- $\mathscr{P}_i$ is "honest" if $[\![\alpha]\!]_i = [\![\bar{\alpha}]\!]_i$
- # honest parties $\geq \ell + 1 \Rightarrow$ honest prover

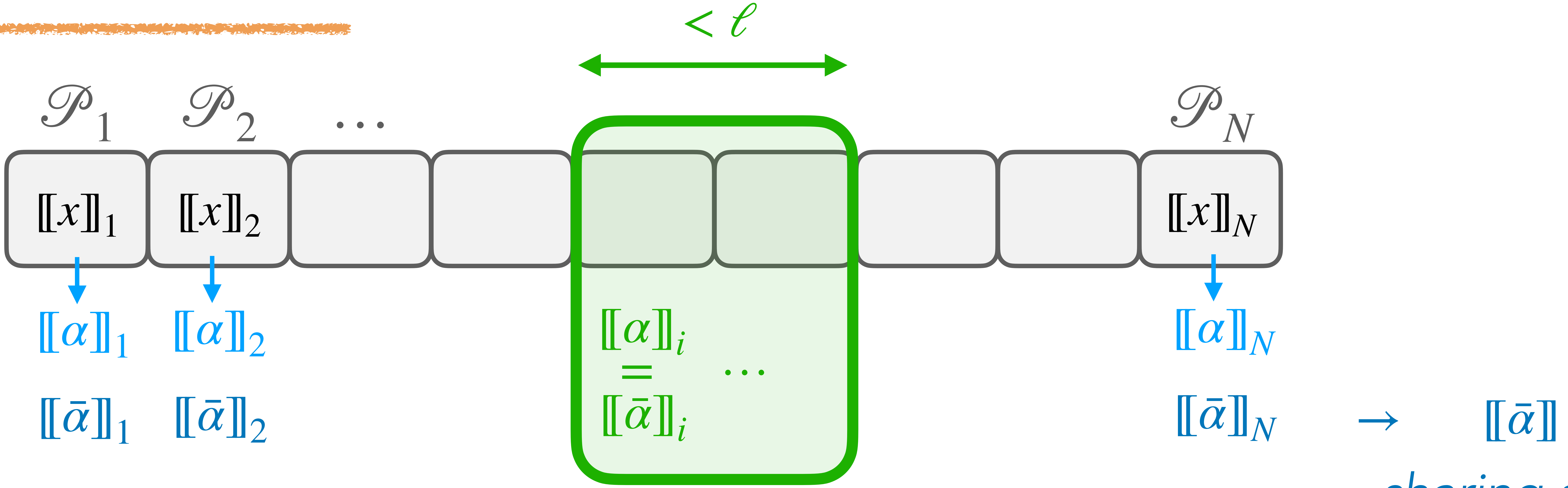
Soundness



sharing sent to the verifier s.t. $g(y, \bar{\alpha}) =$ Accept

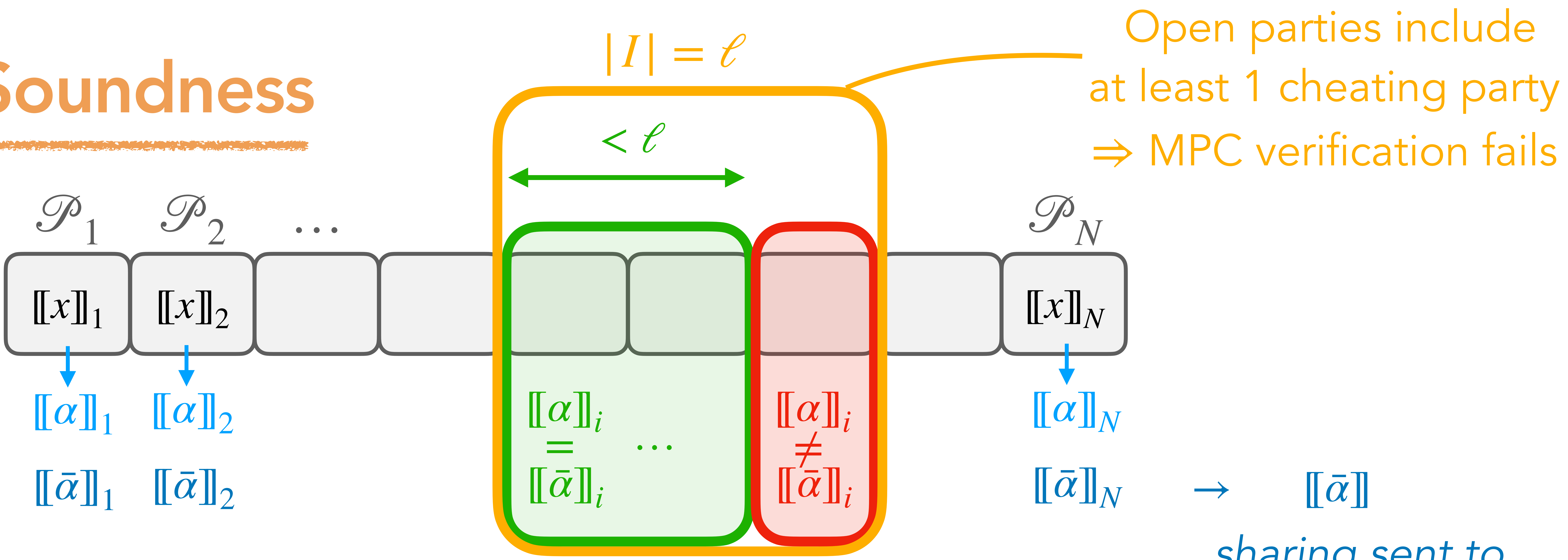
- $\mathscr{P}_i$ is "honest" if $[\![\alpha]\!]_i = [\![\bar{\alpha}]\!]_i$
- # honest parties $\geq \ell + 1$ $\Rightarrow$ honest prover
- Malicious prover $\Rightarrow$ # honest parties $\leq \ell$

Soundness



- $\mathcal{P}_i$ is "honest" if $[\![\alpha]\!]_i = [\![\bar{\alpha}]\!]_i$
- # honest parties $\geq \ell + 1 \Rightarrow$ honest prover
- Malicious prover $\Rightarrow$ # honest parties $\leq \ell$
  - # honest parties $< \ell$

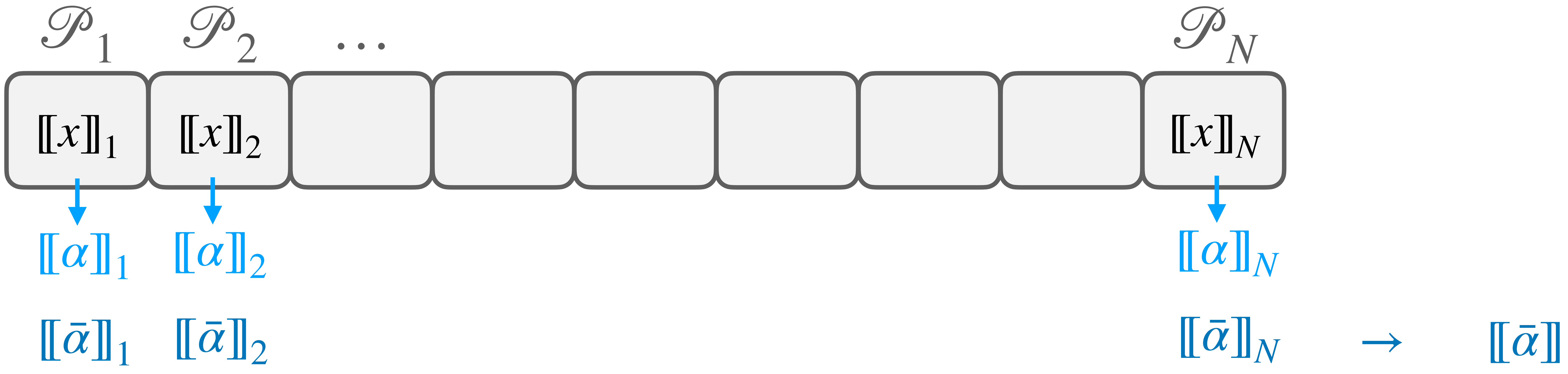
Soundness



sharing sent to the verifier s.t. $g(y, \bar{\alpha})$ = Accept

- $\mathcal{P}_i$ is "honest" if $[\![\alpha]\!]_i = [\![\bar{\alpha}]\!]_i$
- # honest parties $\geq \ell + 1$ ⇒ honest prover
- Malicious prover ⇒ # honest parties $\leq \ell$
  - # honest parties $< \ell$

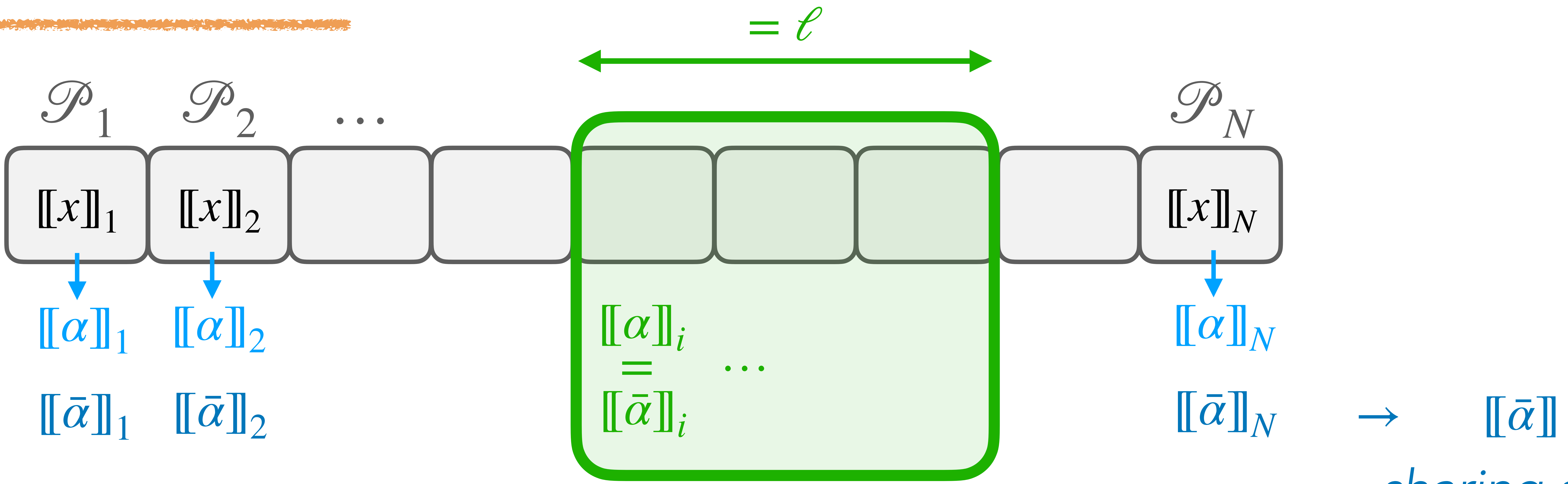
Soundness



sharing sent to the verifier s.t. $g(y, \bar{\alpha}) =$ Accept

- $\mathscr{P}_i$ is "honest" if $[\![\alpha]\!]_i = [\![\bar{\alpha}]\!]_i$
- # honest parties $\geq \ell + 1$ $\Rightarrow$ honest prover
- Malicious prover $\Rightarrow$ # honest parties $\leq \ell$
  - # honest parties $< \ell$ $\Rightarrow$ cheat always detected

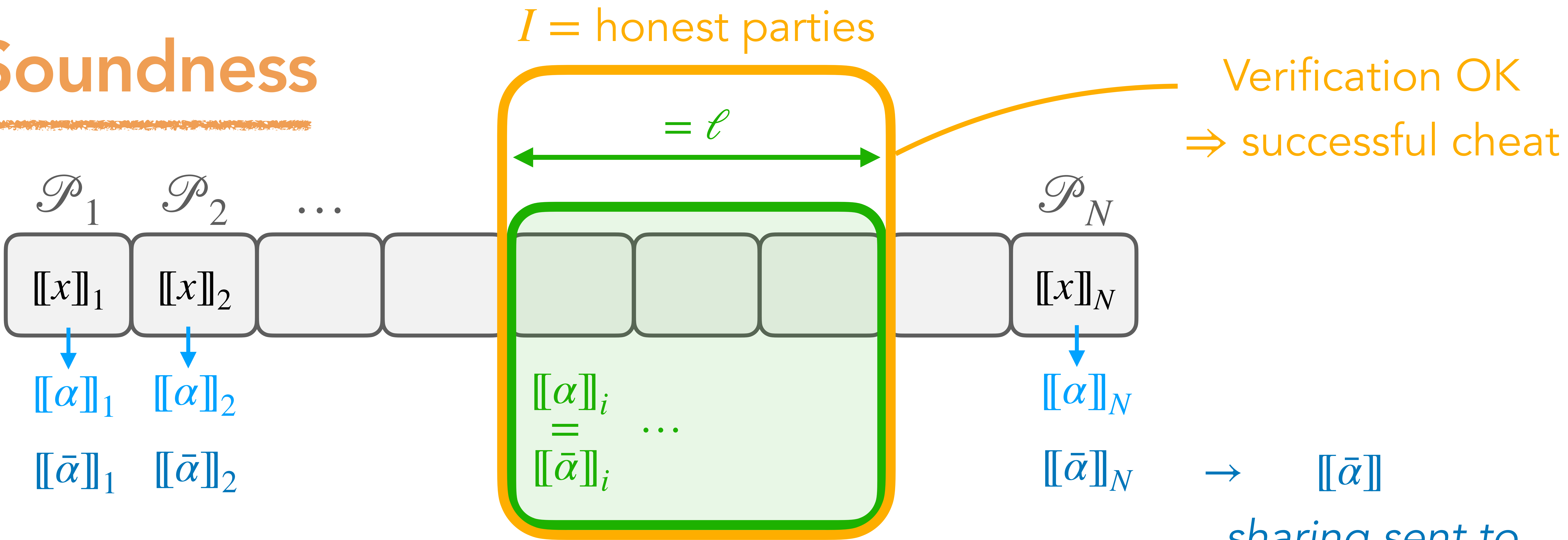
Soundness



sharing sent to the verifier s.t. $g(y, \bar{\alpha}) =$ Accept

- $\mathcal{P}_i$ is "honest" if $[\![\alpha]\!]_i = [\![\bar{\alpha}]\!]_i$
- # honest parties $\geq \ell + 1 \Rightarrow$ honest prover
- Malicious prover $\Rightarrow$ # honest parties $\leq \ell$
  - # honest parties $< \ell \Rightarrow$ cheat always detected
  - # honest parties $= \ell$

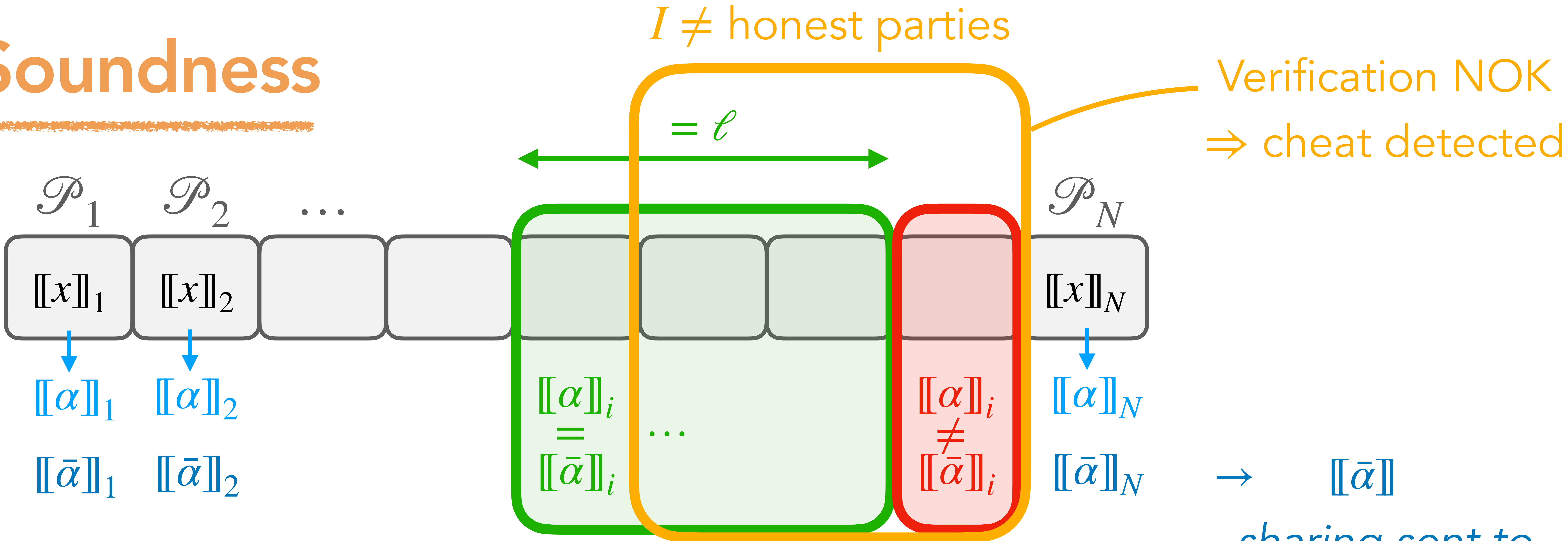
Soundness



sharing sent to the verifier s.t. $g(y, \bar{\alpha})$ = Accept

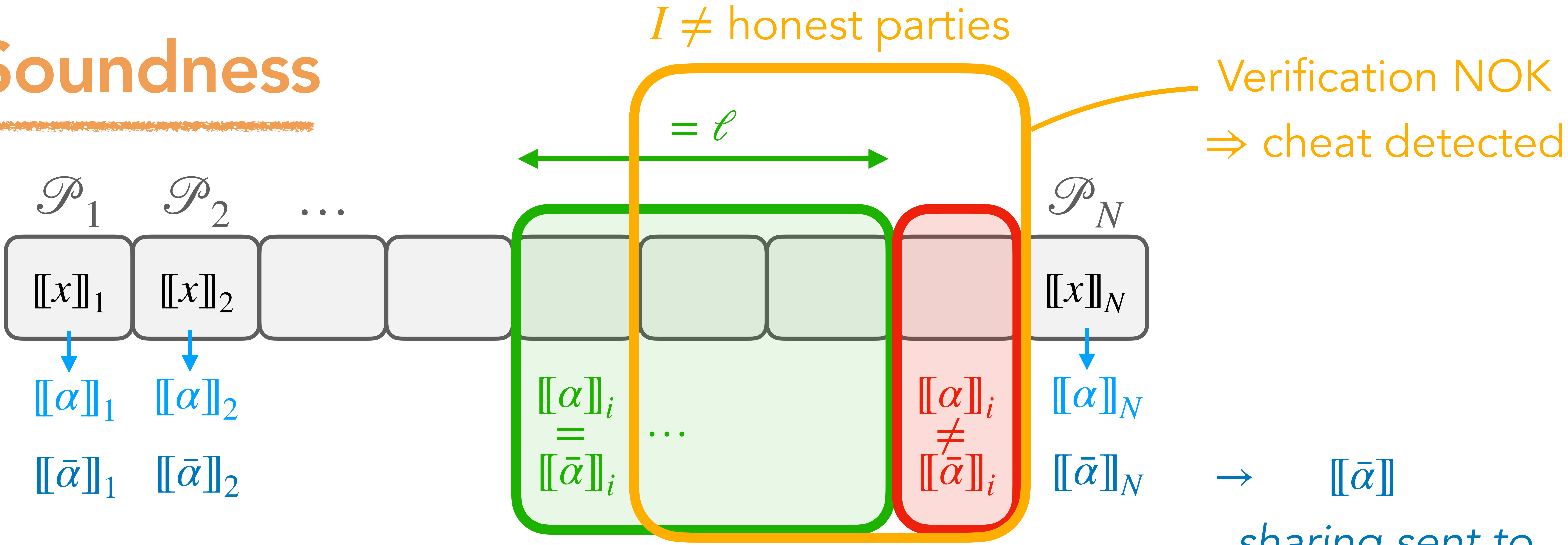
- $\mathscr{P}_i$ is "honest" if $[\![\alpha]\!]_i = [\![\bar{\alpha}]\!]_i$
- # honest parties $\geq \ell + 1$ ⇒ honest prover
- Malicious prover ⇒ # honest parties $\leq \ell$
  - # honest parties $< \ell$ ⇒ cheat always detected
  - # honest parties $= \ell$

Soundness



- $\mathcal{P}_i$ is "honest" if $[\![\alpha]\!]_i = [\![\bar\alpha]\!]_i$
- # honest parties $\geq \ell + 1$ $\Rightarrow$ honest prover
- Malicious prover $\Rightarrow$ # honest parties $\leq \ell$
  - # honest parties $< \ell$ $\Rightarrow$ cheat always detected
  - # honest parties $= \ell$

Soundness

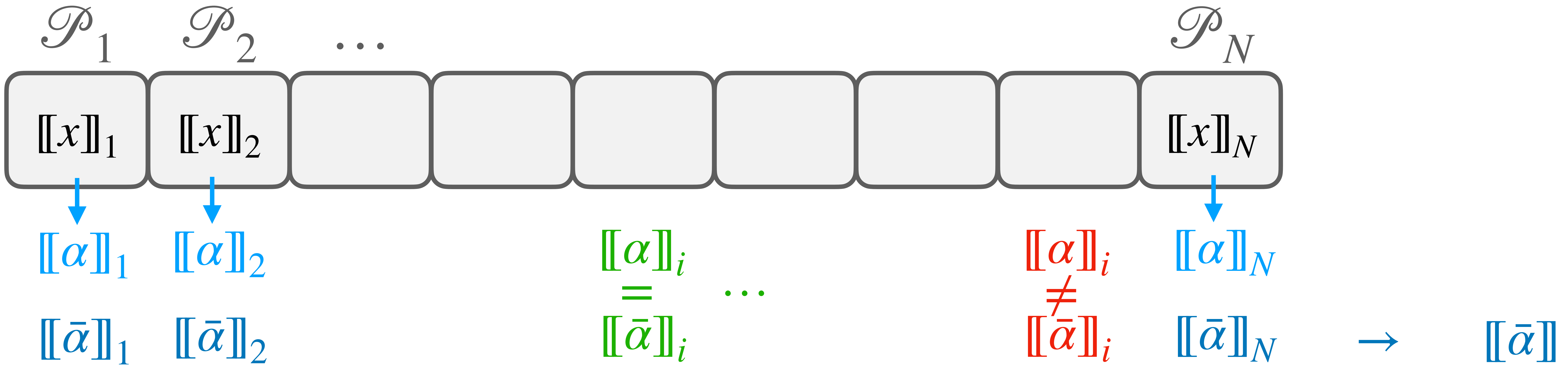


sharing sent to the verifier s.t. $g(y, \bar\alpha) =$ Accept

- $\mathcal{P}_i$ is "honest" if $[\![\alpha]\!]_i = [\![\bar\alpha]\!]_i$
- # honest parties $\geq \ell + 1$ $\Rightarrow$ honest prover
- Malicious prover $\Rightarrow$ # honest parties $\leq \ell$
  - # honest parties $< \ell$ $\Rightarrow$ cheat always detected
  - # honest parties $= \ell$

💡 Cheat successful iff $I =$ honest parties

Soundness



sharing sent to the verifier s.t. $g(y, \bar{\alpha})$ = Accept

- $\mathscr{P}_i$ is "honest" if $[\![\alpha]\!]_i = [\![\bar{\alpha}]\!]_i$
- # honest parties $\geq \ell + 1$ $\Rightarrow$ honest prover
- Malicious prover $\Rightarrow$ # honest parties $\leq \ell$
  - # honest parties $< \ell$ $\Rightarrow$ cheat always detected
  - # honest parties $= \ell$ $\Rightarrow$ soundness error $\dfrac{1}{\binom{N}{\ell}}$

💡 Cheat successful iff I = honest parties

Soundness

- We implicitly assumed that the MPC protocol has no false positive
- False positive probability $p \neq 0$ → more complex analysis **[FR22]**
- Soundness error

$$\frac{1}{\binom{N}{\ell}} + p\,\frac{\ell(N-\ell)}{\ell+1}$$

- Fiat-Shamir transform: p should be small for efficient application

Comparison

	Additive sharing + seed trees + hypercube	**Threshold LSSS** with $\ell = 1$
Soundness error	$\frac{1}{N} + p\left(1 - \frac{1}{N}\right)$	$\frac{1}{N} + p\left(\frac{N-1}{2}\right)$
Prover # party computations	$\log N + 1$	2
Verifier # party computations	$\log N$	1
Size of seed / Merkle tree	$\lambda(\log N)$	$2\lambda(\log N)^{*}$

$*$ *might be more for MPC protocols with many rounds of oracle queries*

Comparison

	Additive sharing + seed trees + hypercube	**Threshold LSSS** with $\ell = 1$
For signatures with $\lambda = 128,\ N = 256,\ \tau = 16$		
Prover # party computations	144	32
Verifier # party computations	128	16
Size of seed / Merkle tree	2KB	4KB

Conclusion

- MPC in the Head is great!
- Efficient and short ZK proofs for small circuits / one-way functions
  - Typical application: PQ signatures
  - (For larger computation, ZK-SNARK are better)
- Two interesting options (trade-off)
  - Additive sharing (with seed trees and hypercube)
  - Threshold sharing
- Other type of sharing: sharing over the integers / MPCitH with rejection

  [FMRV22] Feneuil, Maire, Rivain, Vergnaud. "Zero-Knowledge Protocols for the Subset Sum Problem from MPC-in-the-Head with Rejection" (ASIACRYPT 2022)