Efficient NIZKs from LWE via Polynomial Reconstruction and "MPC in the Head"

Riddhi Ghosal
UCLA

Paul Lou
UCLA

Amit Sahai
UCLA

NIZKs for all of NP from LWE [CCH+19, PS19]

Prior to our work, all known NIZK arguments for NP from LWE considered instantiating the Fiat-Shamir paradigm on a ***parallel repetition*** of a public-coin honest-verifier zero-knowledge interactive proof:

$L \in \mathsf{NP}$

NIZKs for all of NP from LWE [CCH+19, PS19]

Prior to our work, all known NIZK arguments for NP from LWE considered instantiating the Fiat-Shamir paradigm on a ***parallel repetition*** of a public-coin honest-verifier zero-knowledge interactive proof:

$L \in \text{NP}$

$x \in L$

NIZKs for all of NP from LWE [CCH+19, PS19]

Prior to our work, all known NIZK arguments for NP from LWE considered instantiating the Fiat-Shamir paradigm on a ***parallel repetition*** of a public-coin honest-verifier zero-knowledge interactive proof:

$L \in \mathrm{NP}$

Karp reduction

$x \in L \Rightarrow x' \in \mathrm{HAM}$

NIZKs for all of NP from LWE [CCH+19, PS19]

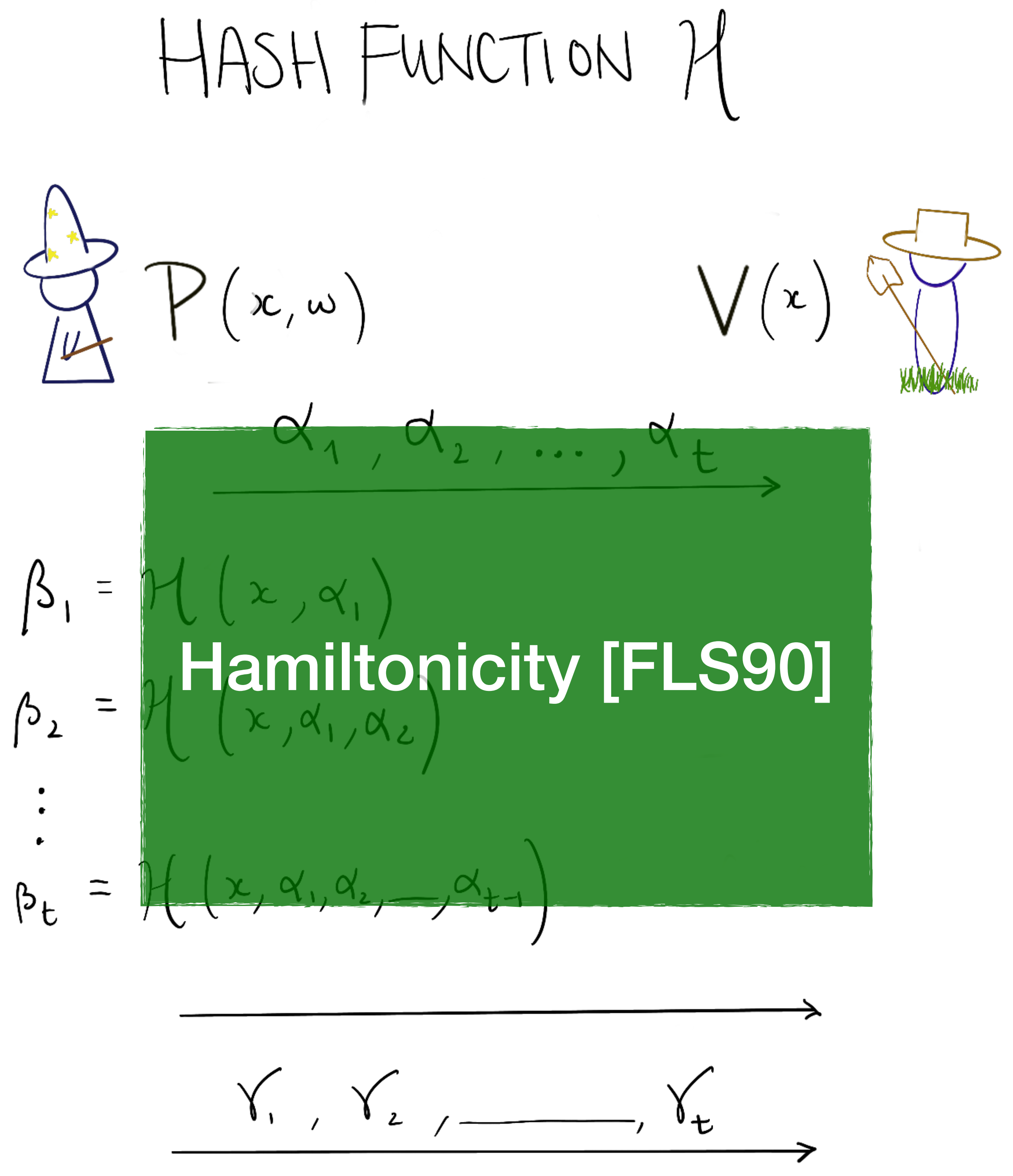
Prior to our work, all known NIZK arguments for NP from LWE considered instantiating the Fiat-Shamir paradigm on a ***parallel repetition*** of a public-coin honest-verifier zero-knowledge interactive proof:

$L \in \mathrm{NP}$

Karp reduction

NIZK Argument in the CRS model

$x \in L$ → $x' \in \mathrm{HAM}$



NIZKs for all of NP from LWE [CCH+19, PS19, HLR21]

Prior to our work, all known NIZK arguments for NP from LWE considered instantiating the Fiat-Shamir paradigm on a ***parallel repetition*** of a public-coin honest-verifier zero-knowledge interactive proof:

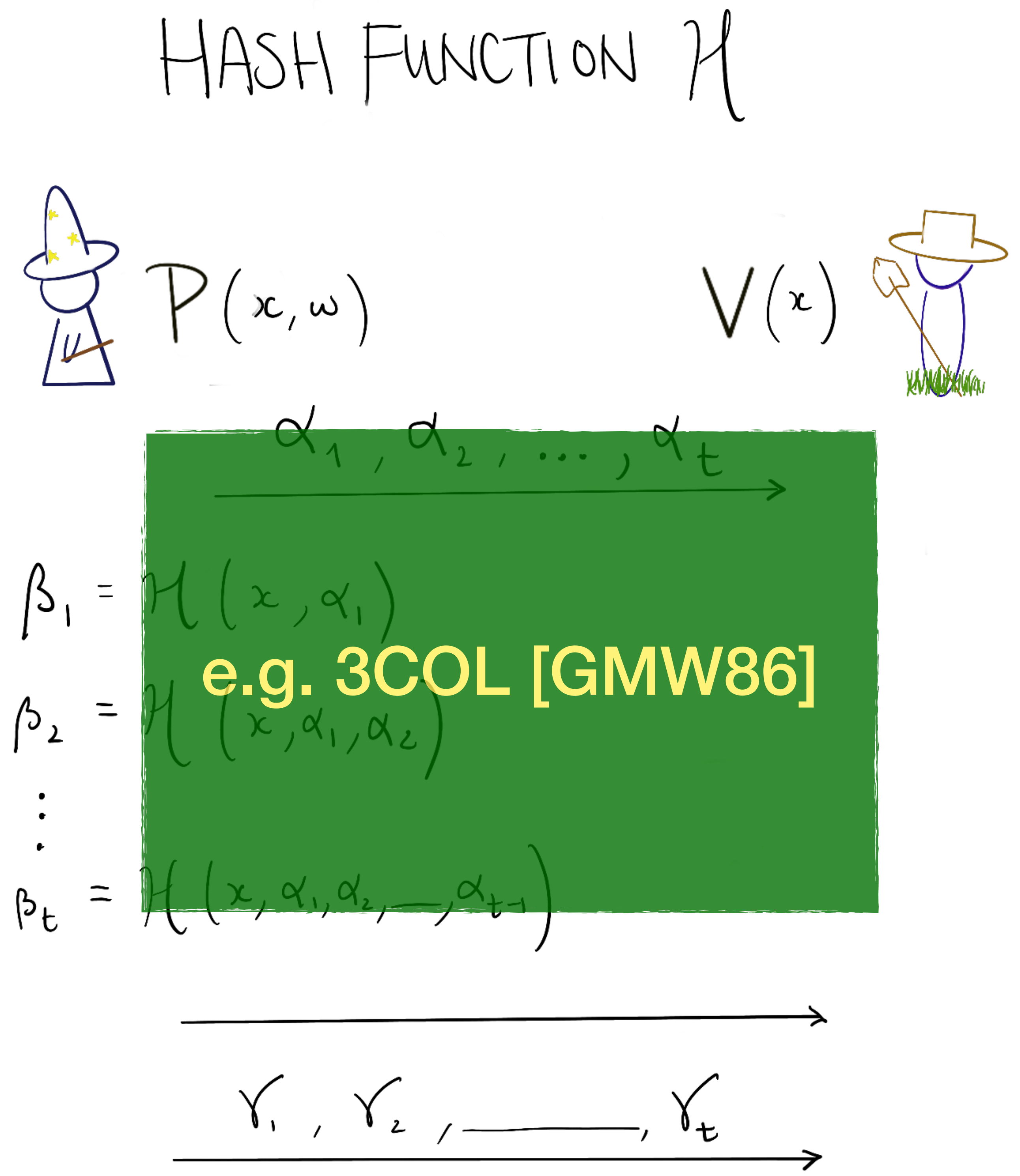
$L \in \mathrm{NP}$

Karp reduction

$x \in L$

$x' \in \mathrm{3COL}$

NIZK Argument for any commit-and-open protocol [HLR21]



NIZKs for all of NP from LWE [CCH+19, PS19, HLR21]

Prior to our work, all known NIZK arguments for NP from LWE considered instantiating the Fiat-Shamir paradigm on a ***parallel repetition*** of a public-coin honest-verifier zero-knowledge interactive proof*:*

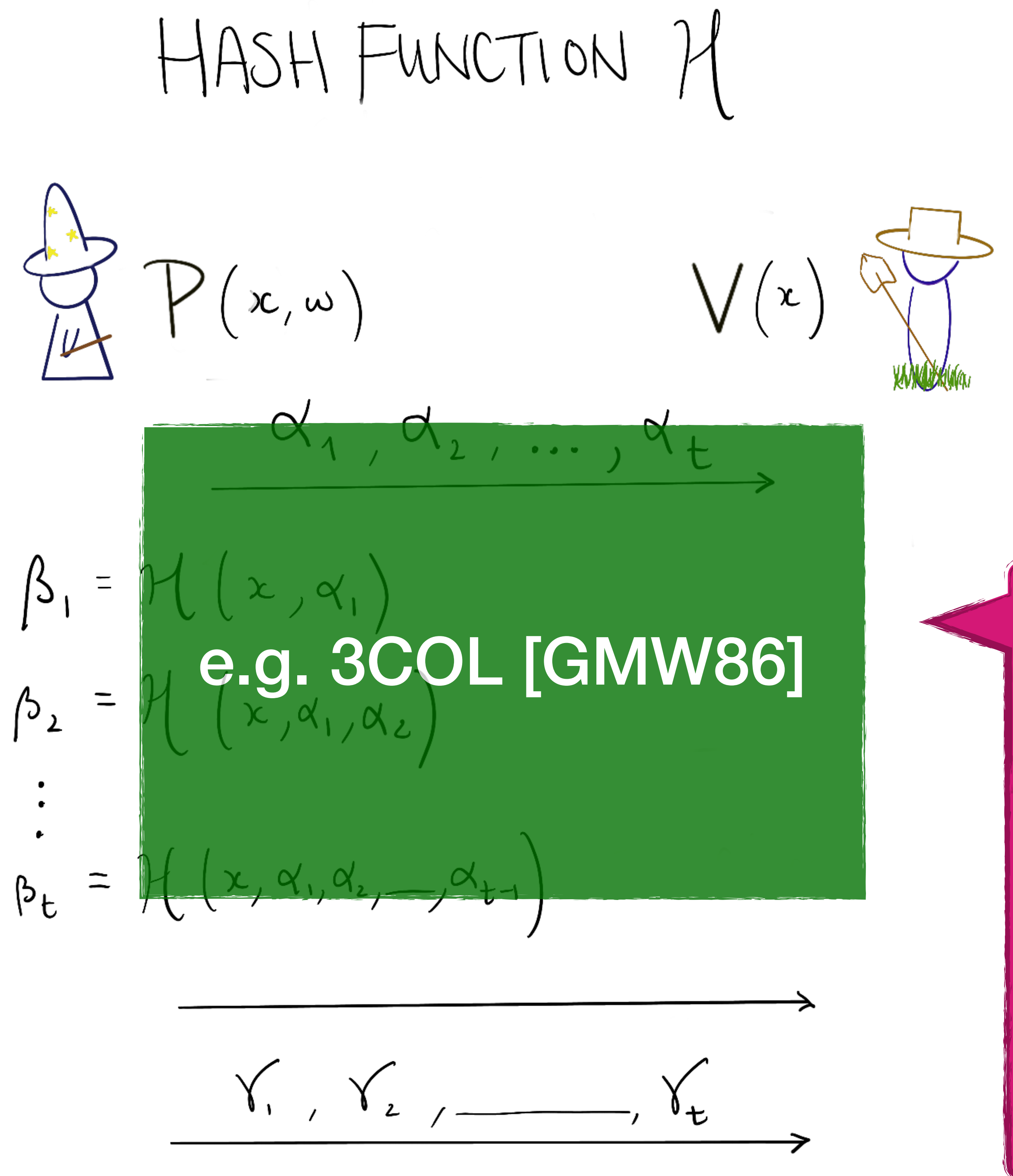
$L \in \mathrm{NP}$

Karp reduction

$x \in L$

NIZK Argument for any commit-and-open protocol [HLR21]

$x' \in \mathrm{3COL}$



Large proof size due to parallel repetition!

NIZKs for all of NP from LWE [CCH+19, PS19, HLR21]

Prior to our work, all known NIZK arguments for NP from LWE considered instantiating the Fiat-Shamir paradigm on a ***parallel repetition*** of a public-coin honest-verifier zero-knowledge interactive proof:

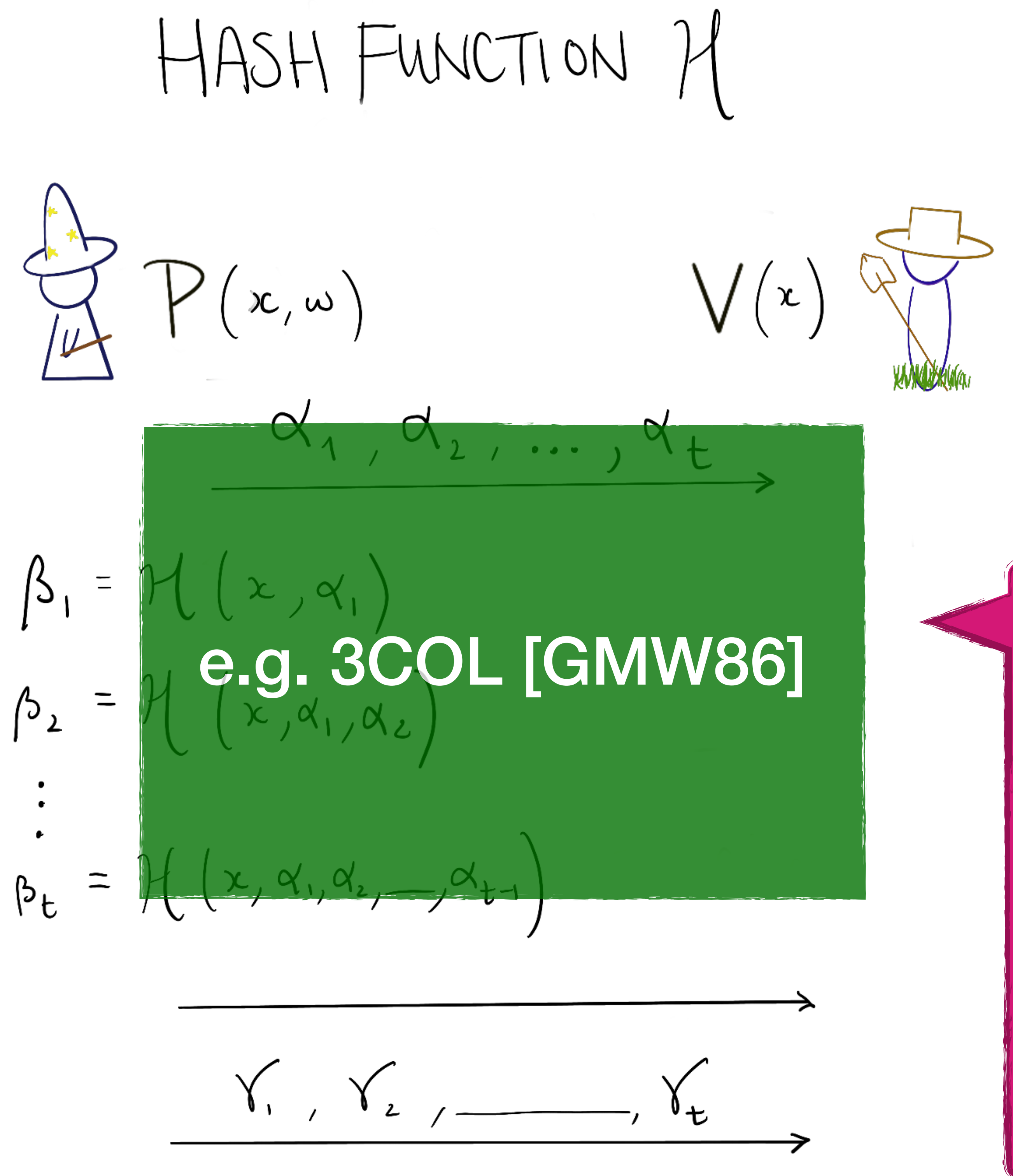
$L \in \mathrm{NP}$

Karp reduction

NIZK Argument for any commit-and-open protocol [HLR21]

$x \in L$ → $x' \in \mathrm{3COL}$

Expensive!

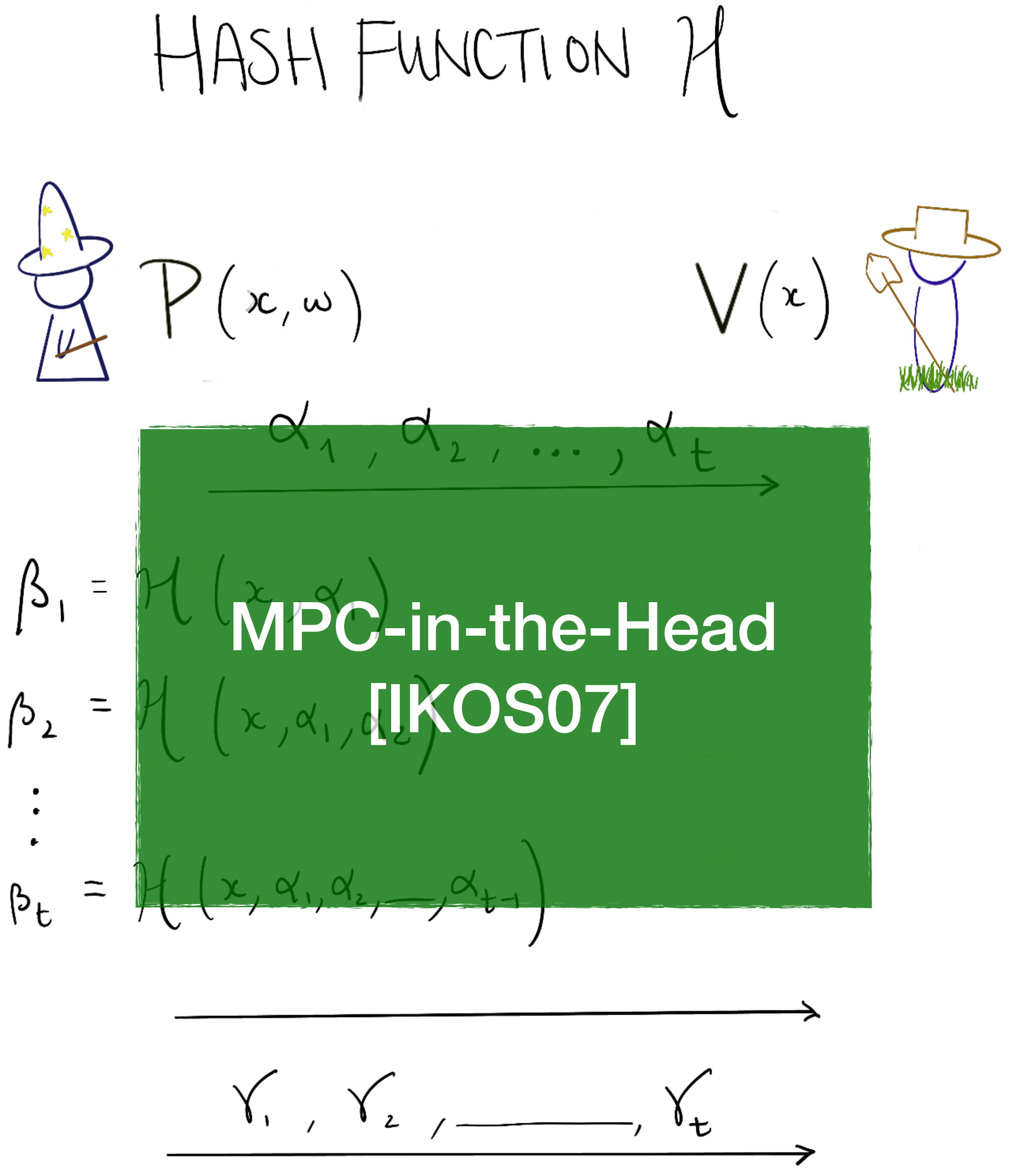


Large proof size due to parallel repetition!

Our Work

We give an *efficient* (smaller proof size) base NIZK construction for NP from LWE *without* parallel repetition and Karp reductions.

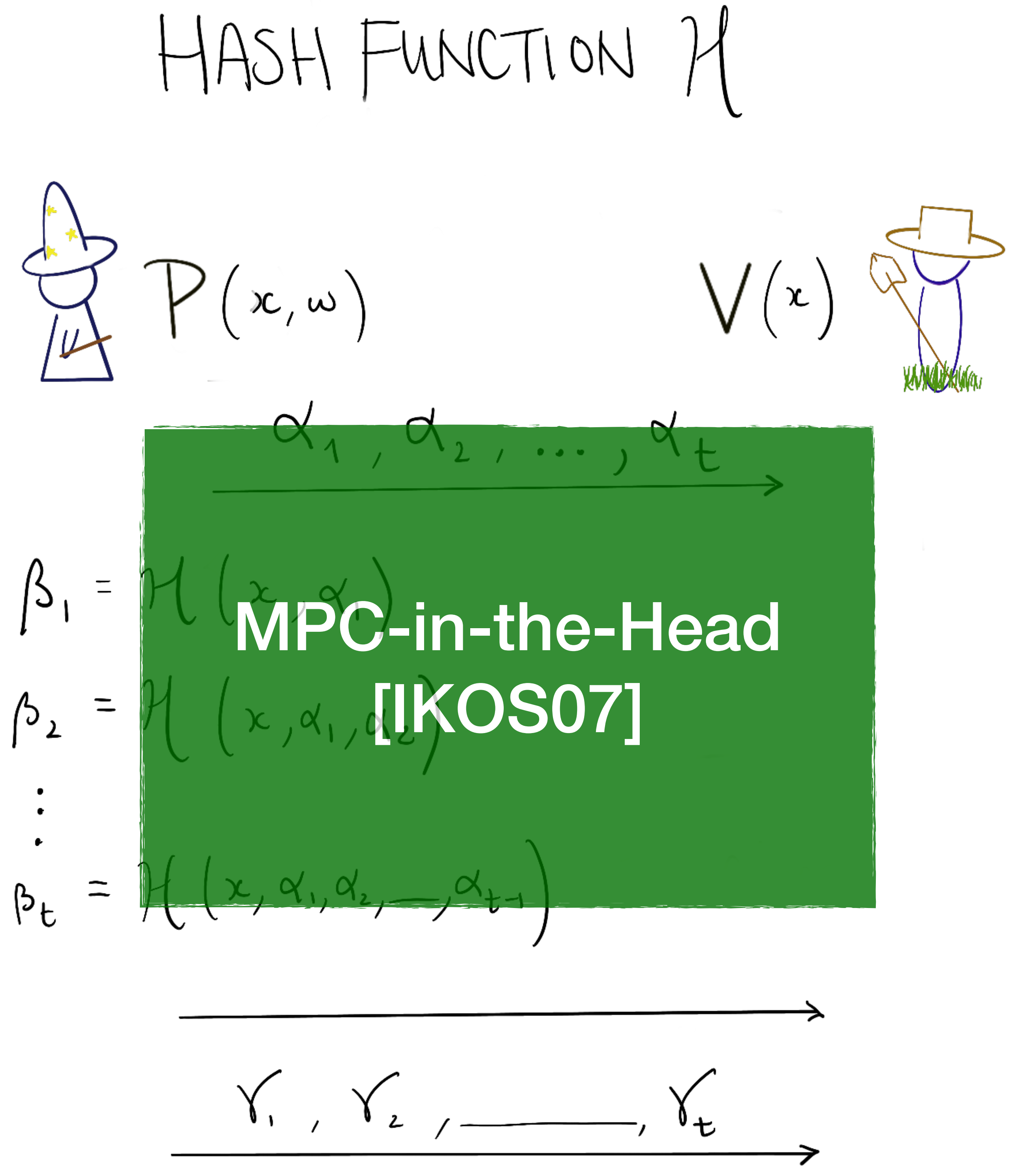
NIZK Argument in the CRS model



Our Work

We give an *efficient* (smaller proof size) base NIZK construction for NP from LWE *without* parallel repetition and Karp reductions.

NIZK Argument in the CRS model



MPC-in-the-Head [IKOS07]

Allows us to translate work on efficient perfectly robust MPC protocols [DIK10, BGJK21, GPS21] to efficient NIZKs from LWE!

Our Work

We give an *efficient* (smaller proof size) base NIZK construction for NP from LWE *without* parallel repetition and Karp reductions.

Main Theorem (informal)

Assuming the hardness of LWE,

Our Work

We give an *efficient* (smaller proof size) base NIZK construction for NP from LWE *without* parallel repetition and Karp reductions.

Main Theorem (informal)

Assuming the hardness of LWE, there exists NIZKs with computational soundness for all of NP whose proof size is $O(|C| + q \cdot \mathrm{depth}(C)) + \mathrm{poly}(k)$ field elements in $\mathbb{F}$, where k is the security parameter, $q = \tilde{O}(k)$, $|\mathbb{F}| \geq 2q$, and C is an arithmetic circuit for the NP verification function.

Our Work

We give an *efficient* (smaller proof size) base NIZK construction for NP from LWE *without* parallel repetition and Karp reductions.

[GGI+15] Can use FHE to bootstrap an underlying NIZK to one with proof size $|w| + \text{poly}(k)$ bits.

Main Theorem (informal)

Assuming the hardness of LWE, there exists NIZKs with computational soundness for all of NP whose proof size is $O(|C| + q \cdot \text{depth}(C)) + \text{poly}(k)$ field elements in $\mathbb{F}$, where k is the security parameter, $q = \tilde{O}(k)$, $|\mathbb{F}| \geq 2q$, and C is an arithmetic circuit for the NP verification function.

Overview: Our Technique

- [HLR21]'s coding theoretic approach to instantiating Fiat-Shamir: Block size of list-recoverable error-correcting code determines efficiency.

Overview: Our Technique

- [HLR21]'s coding theoretic approach to instantiating Fiat-Shamir: Block size of list-recoverable error-correcting code determines efficiency.
  - Parvaresh-Vardy code concatenated with a *single* random code achieves block-size of $O(k^{1+\epsilon})$ for any small constant $\epsilon > 0$.

Overview: Our Technique

- [HLR21]'s coding theoretic approach to instantiating Fiat-Shamir: Block size of list-recoverable error-correcting code determines efficiency.
  - Parvaresh-Vardy code concatenated with a *single* random code achieves block-size of $O(k^{1+\epsilon})$ for any small constant $\epsilon > 0$.
  - Can we generically apply this to MPC-in-the-head?

Overview: Our Technique

- [HLR21]'s coding theoretic approach to instantiating Fiat-Shamir: Block size of list-recoverable error-correcting code determines efficiency.
  - Parvaresh-Vardy code concatenated with a *single* random code achieves block-size of $O(k^{1+\epsilon})$ for any small constant $\epsilon > 0$.
  - Can we generically apply this to MPC-in-the-head? Yes, using very specific properties of the Parvaresh-Vardy code!

Overview: Our Technique

- [HLR21]'s coding theoretic approach to instantiating Fiat-Shamir: Block size of list-recoverable error-correcting code determines efficiency.

  - Parvaresh-Vardy code concatenated with a *single* random code achieves block-size of $O(k^{1+\epsilon})$ for any small constant $\epsilon > 0$.

  - Can we generically apply this to MPC-in-the-head? Yes, using very specific properties of the Parvaresh-Vardy code! *But* general list-recovery does not take advantage of the special structure present in the MPC-in-the-head setting.

Overview: Our Technique

- [HLR21]'s coding theoretic approach to instantiating Fiat-Shamir: Block size of list-recoverable error-correcting code determines efficiency.
  - Parvaresh-Vardy code concatenated with a *single* random code achieves block-size of $O(k^{1+\epsilon})$ for any small constant $\epsilon > 0$.
  - Can we generically apply this to MPC-in-the-head? Yes, using very specific properties of the Parvaresh-Vardy code! *But* general list-recovery does not take advantage of the special structure present in the MPC-in-the-head setting.

We show that this yields less efficient proofs.

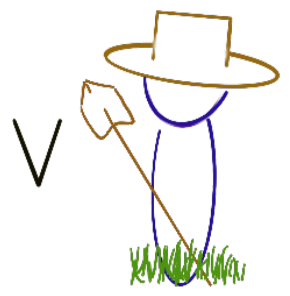
Overview: Our Technique

- [HLR21]'s coding theoretic approach to instantiating Fiat-Shamir: Block size of list-recoverable error-correcting code determines efficiency.
  - Parvaresh-Vardy code concatenated with a *single* random code achieves block-size of $O(k^{1+\epsilon})$ for any small constant $\epsilon > 0$.
  - Can we generically apply this to MPC-in-the-head? Yes, using very specific properties of the Parvaresh-Vardy code! *But* general list-recovery does not take advantage of the special structure present in the MPC-in-the-head setting.
- **Our work:** The bad challenge set structure present in a modification of the [IKOS07] protocol only needs *recurrent* list-recovery.

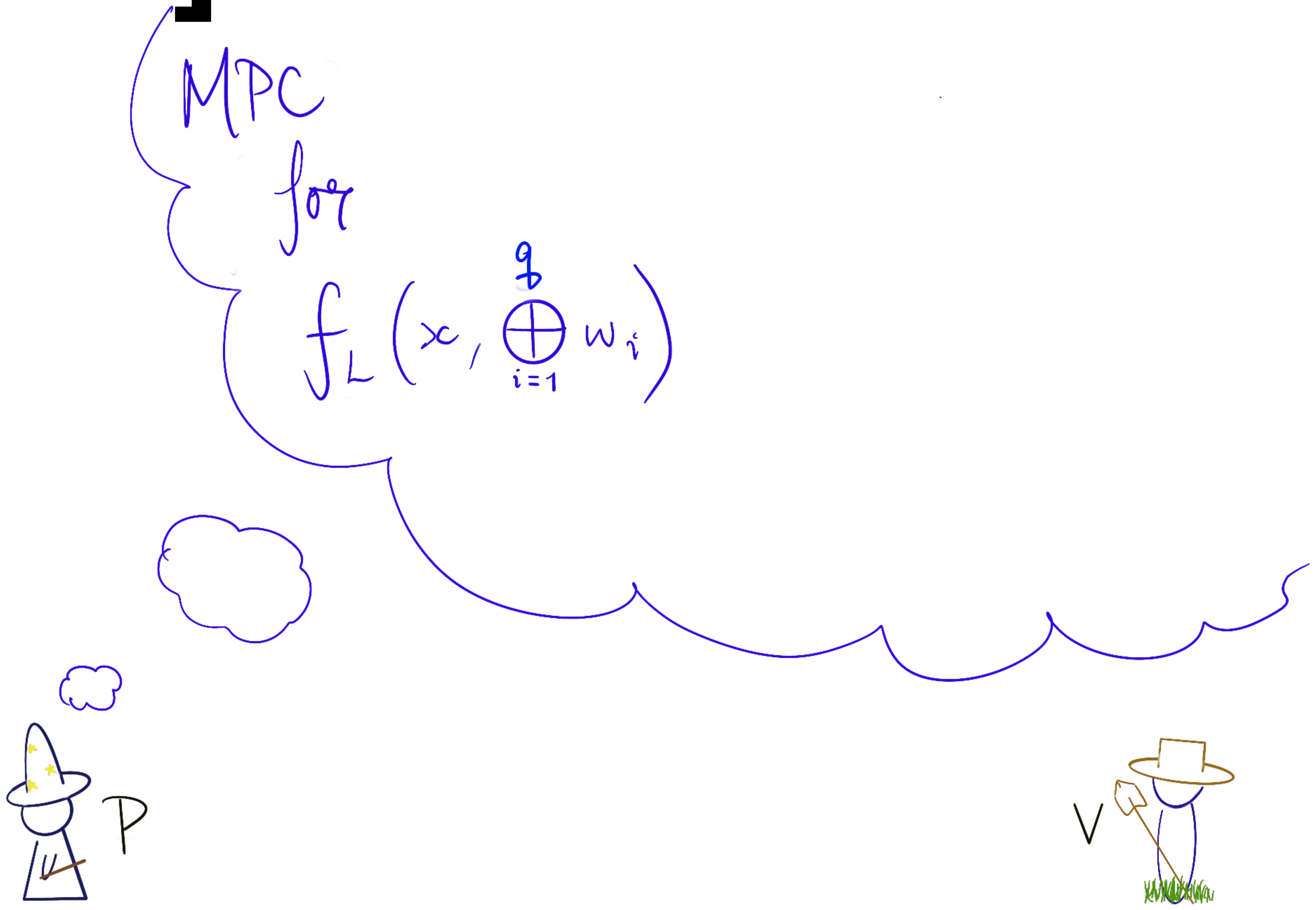
Overview: Our Technique

- [HLR21]'s coding theoretic approach to instantiating Fiat-Shamir: Block size of list-recoverable error-correcting code determines efficiency.
  - Parvaresh-Vardy code concatenated with a *single* random code achieves block-size of $O(k^{1+\epsilon})$ for any small constant $\epsilon > 0$.
  - Can we generically apply this to MPC-in-the-head? Yes, using very specific properties of the Parvaresh-Vardy code! *But* general list-recovery does not take advantage of the special structure present in the MPC-in-the-head setting.
- **Our work:** The bad challenge set structure present in a modification of the [IKOS07] protocol only needs *recurrent* list-recovery. Therefore, we can use *qualitatively simpler* codes (Reed-Solomon codes concatenated with *multiple* random codes) and directly use polynomial reconstruction [Sud97, GS98] to achieve an improved block size of $\tilde{O}(k)$.

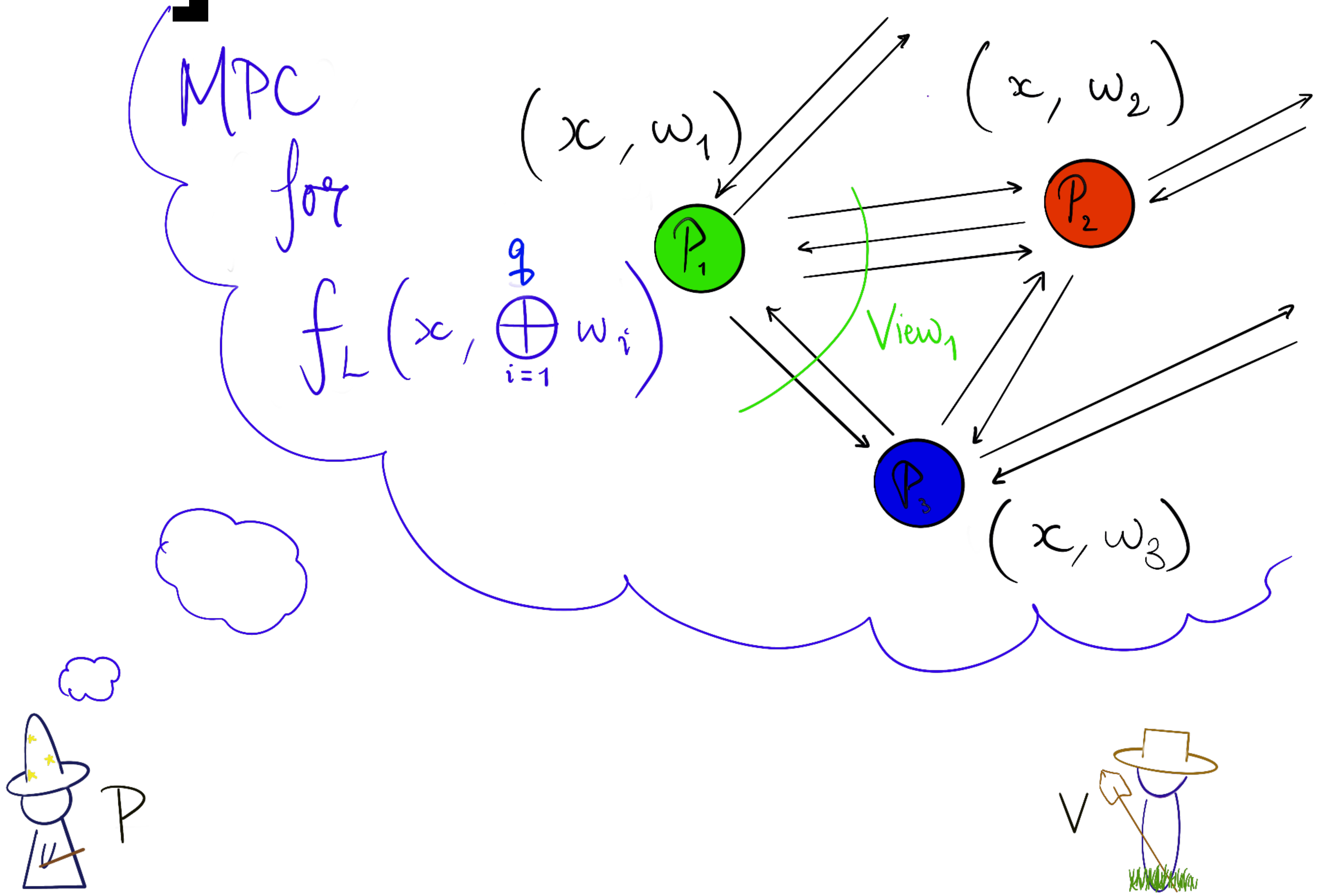
MPC-in-the-Head [IKOS07]



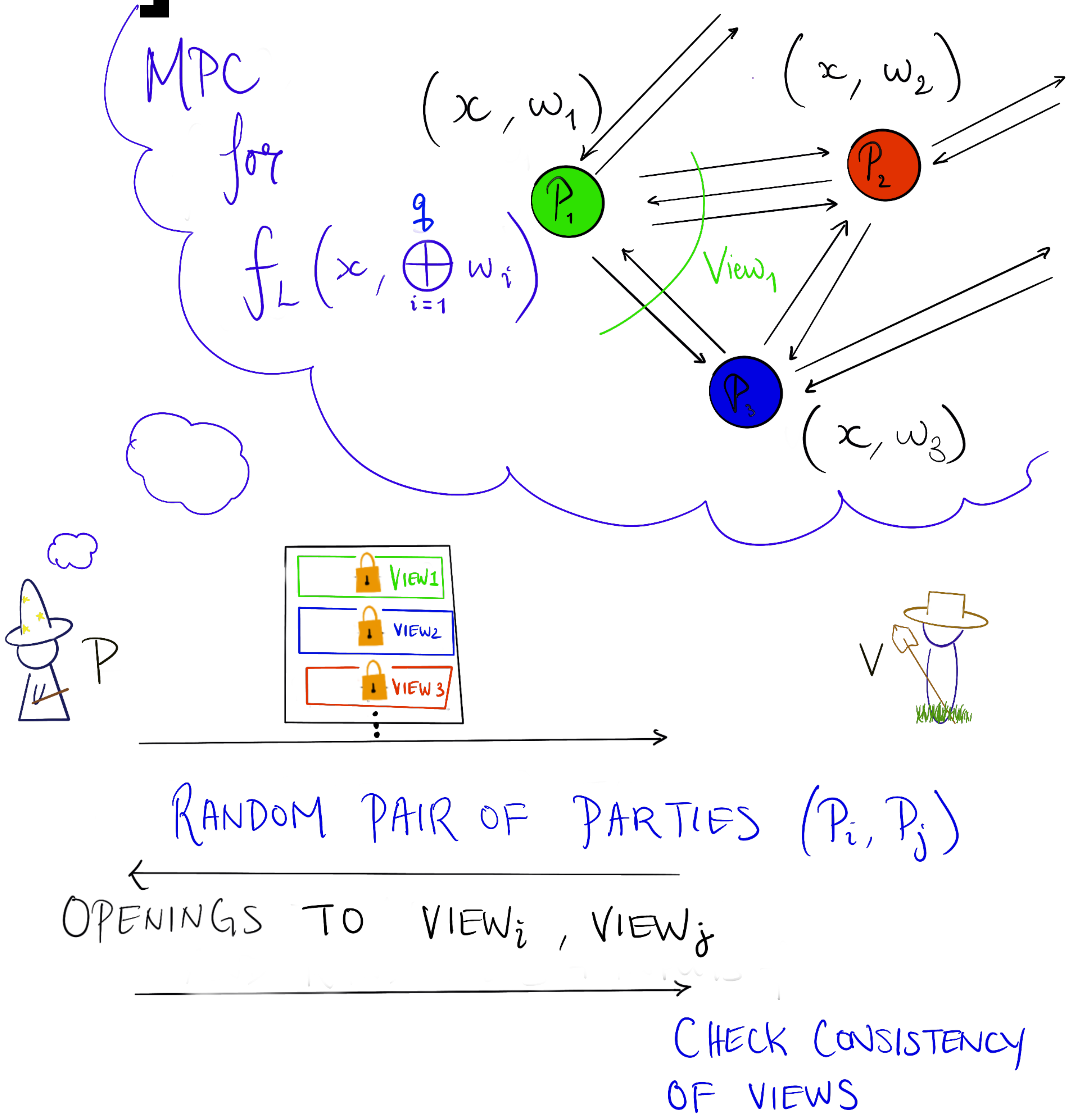
MPC-in-the-Head [IKOS07]



MPC-in-the-Head [IKOS07]



MPC-in-the-Head [IKOS07]



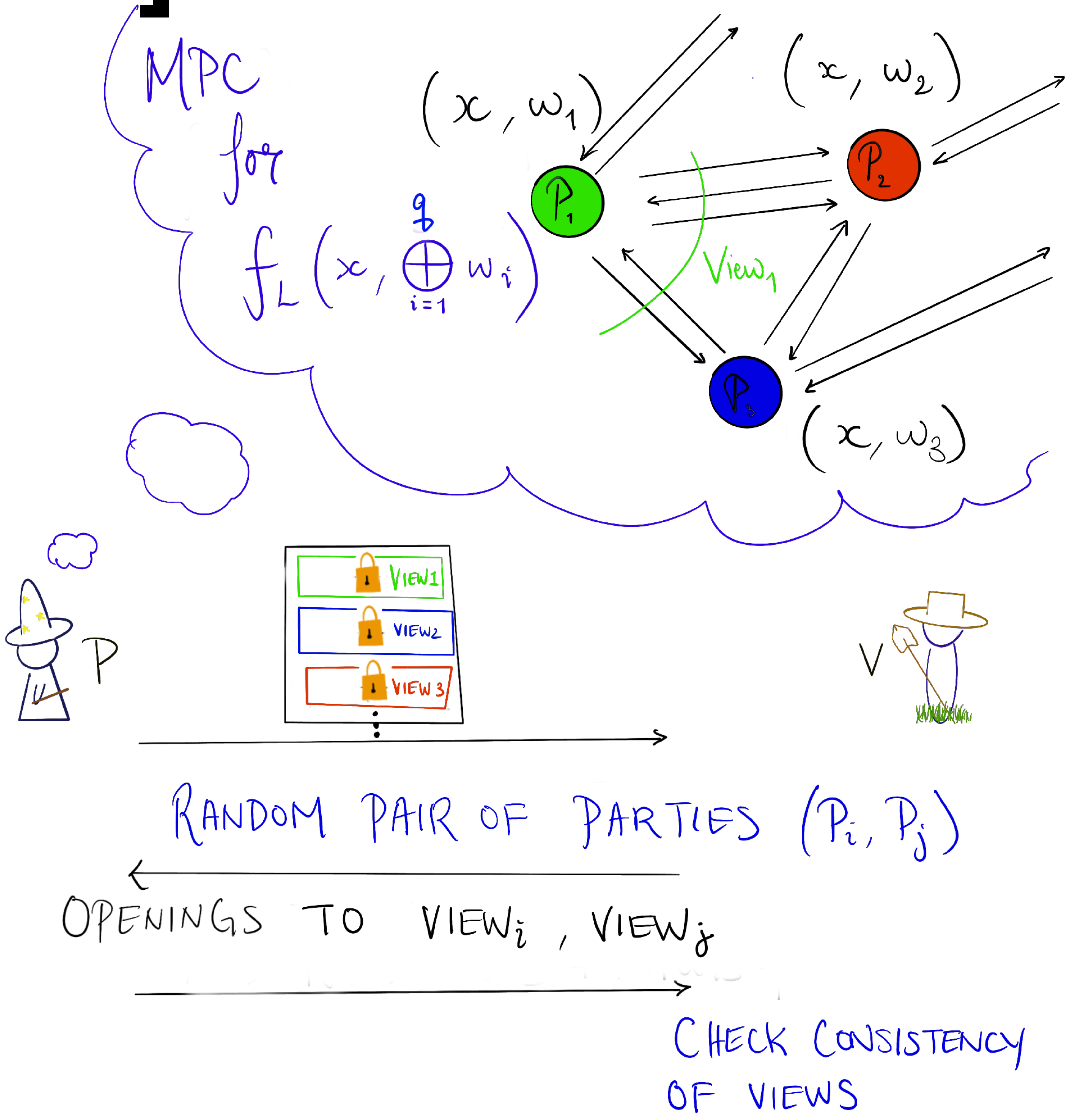
Black-box use of the MPC protocol!

MPC-in-the-Head [IKOS07]

View of $P_1(x, w_1 ; r)$

1. $m_1 \longrightarrow P_2$

2. $m_2 \longleftarrow P_3$

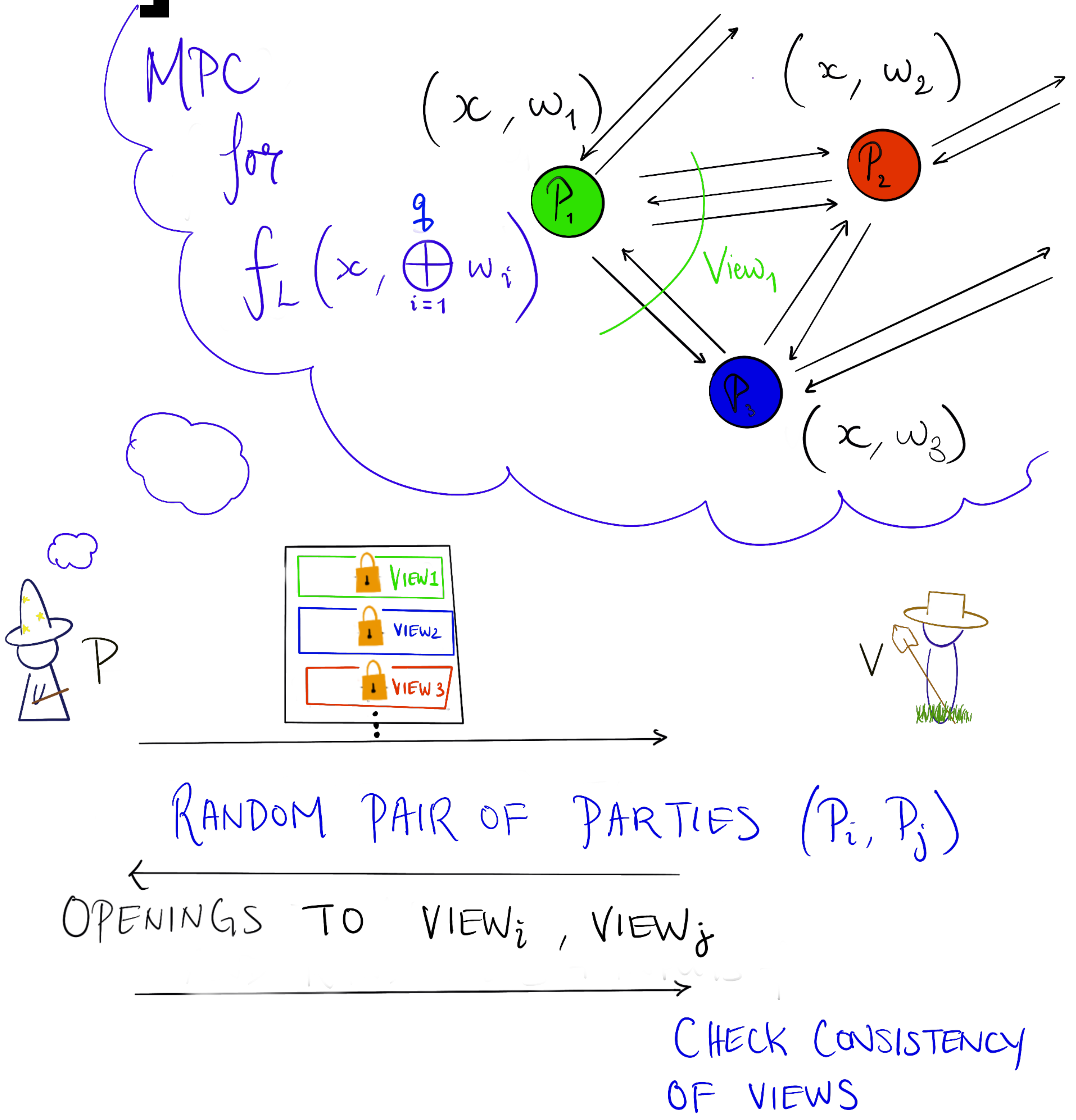


MPC-in-the-Head [IKOS07]

View of $P_1(x, w_1 ; r)$

$$\boxed{\begin{array}{l} 1.\quad m_1 \longrightarrow P_2 \\ \\ 2.\quad m_2 \longleftarrow P_3 \end{array}}$$

$\Downarrow$ $\text{NEXT}(1, x, w_1, r, m_2)$



MPC-in-the-Head [IKOS07]

View of $P_1(x, w_1\,;\, r)$

1. $m_1 \longrightarrow P_2$

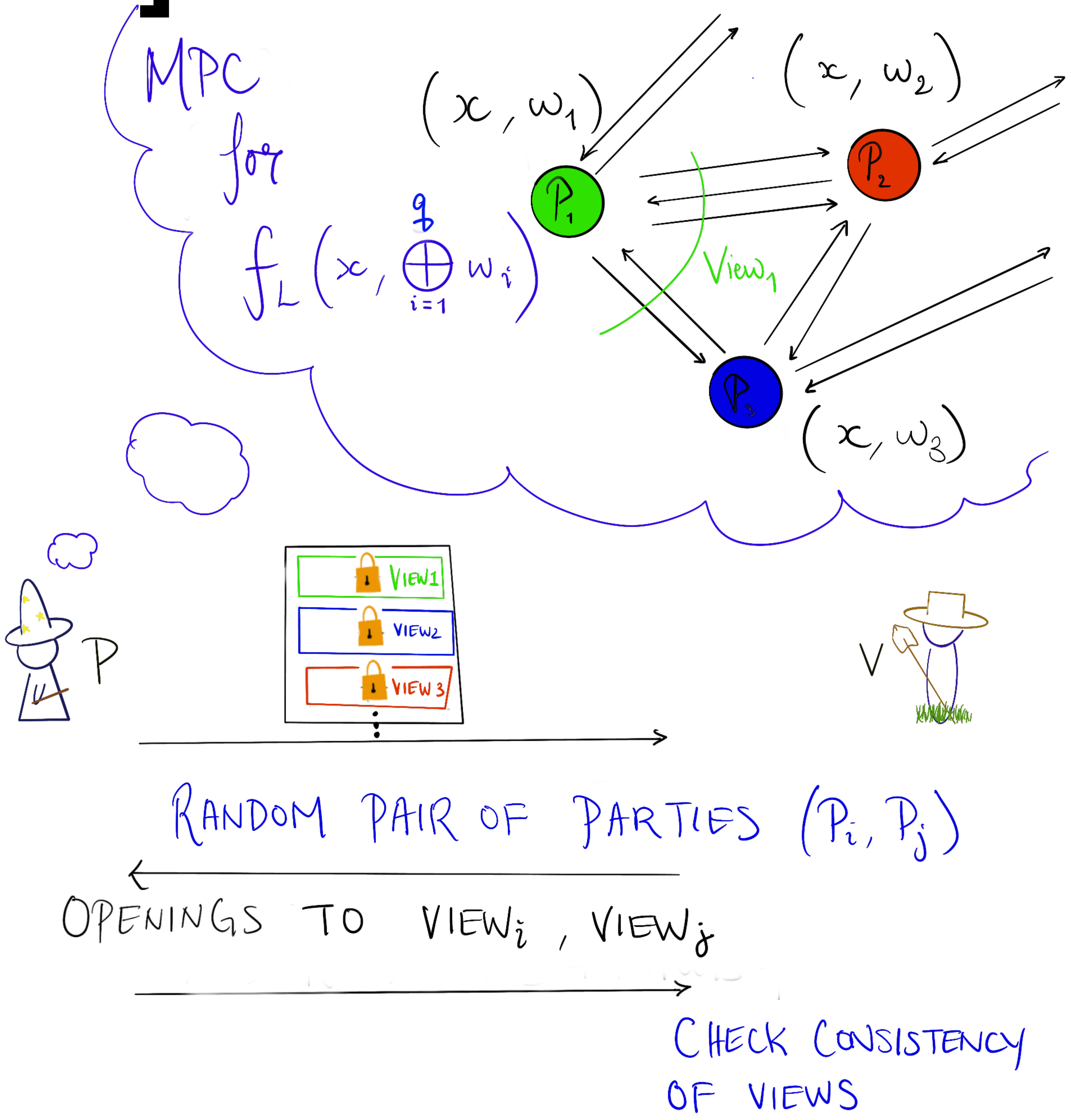
2. $m_2 \longleftarrow P_3$

$\Downarrow$ NEXT$(1, x, w_1, r, m_2)$

Round 3

3. $m_3 \longrightarrow P_2$

$m_4 \longrightarrow P_3$



Our Modification of MPC-in-the-Head

View of $P_1(x, w_1 ; r)$

1. $m_1 \longrightarrow P_2$

2. $m_2 \longleftarrow P_3$

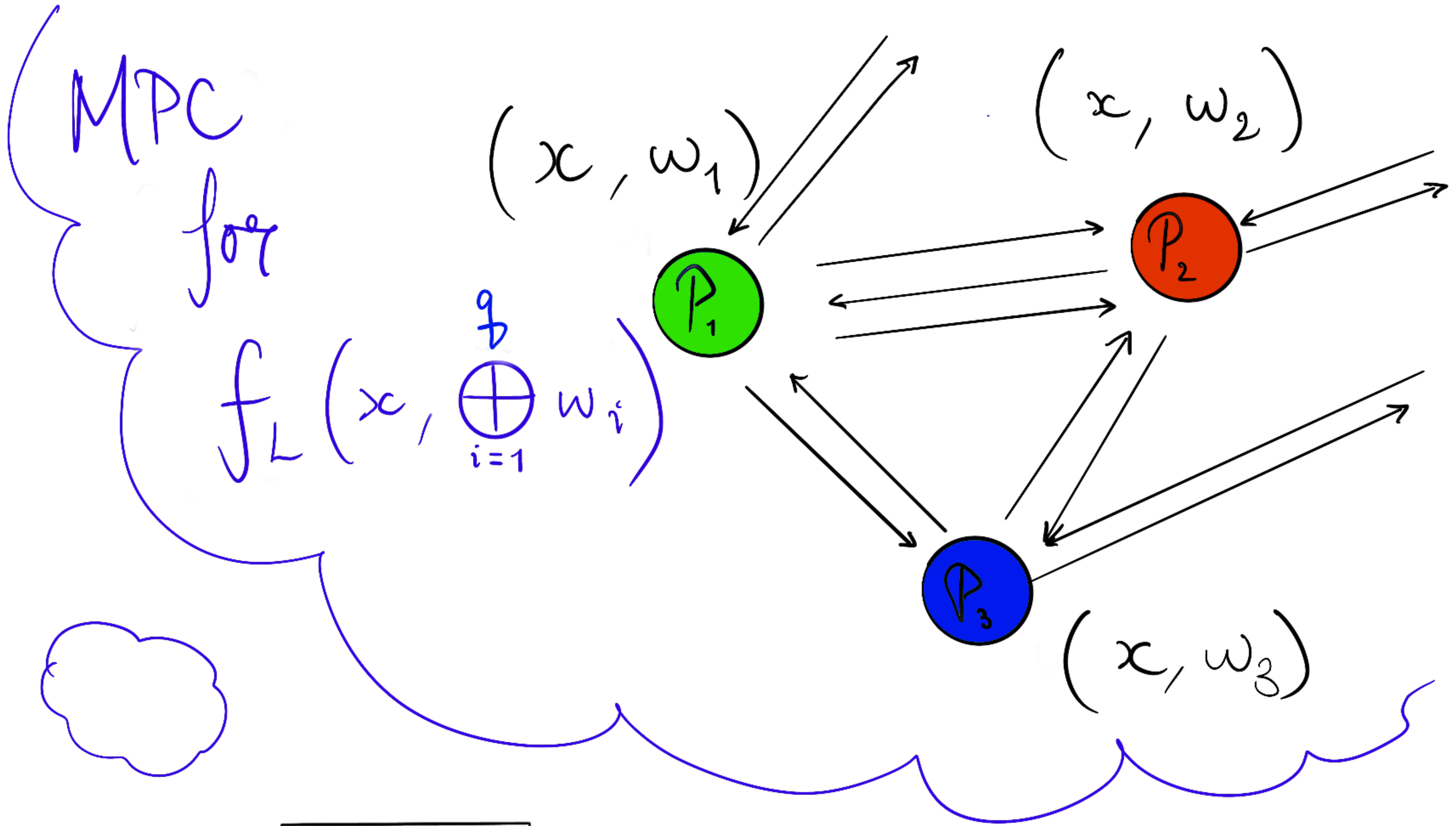
$\Downarrow$ NEXT$(1, x, w_1, r, m_2)$

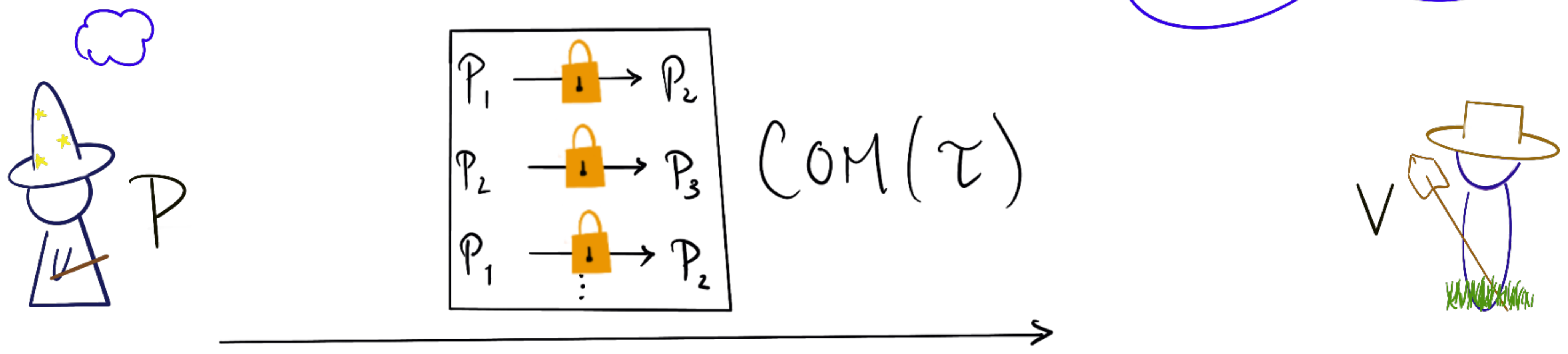
Round 3

3. $m_3 \longrightarrow P_2$

$m_4 \longrightarrow P_3$

Non-black-box use of the MPC protocol!



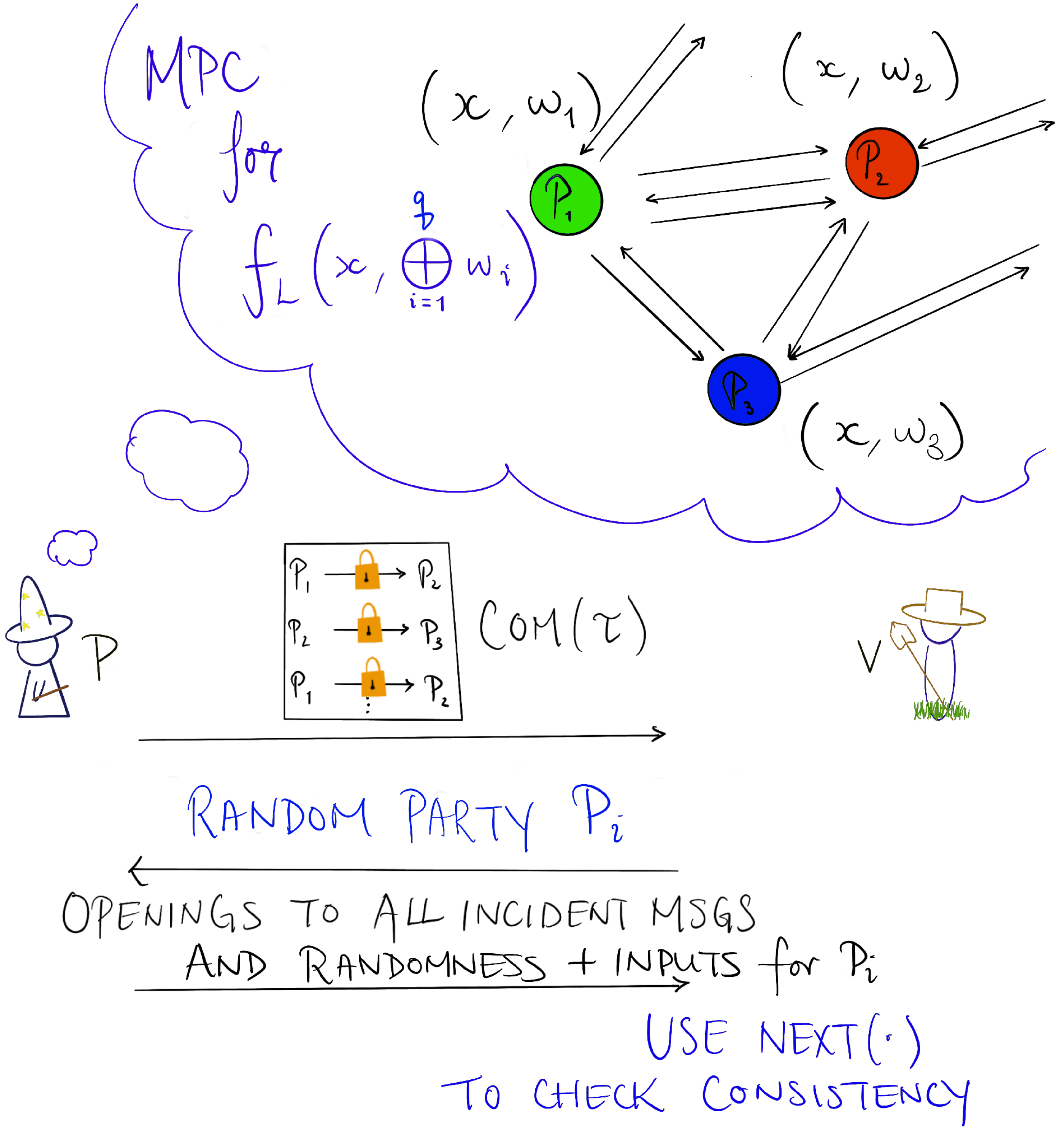


RANDOM PARTY P_i

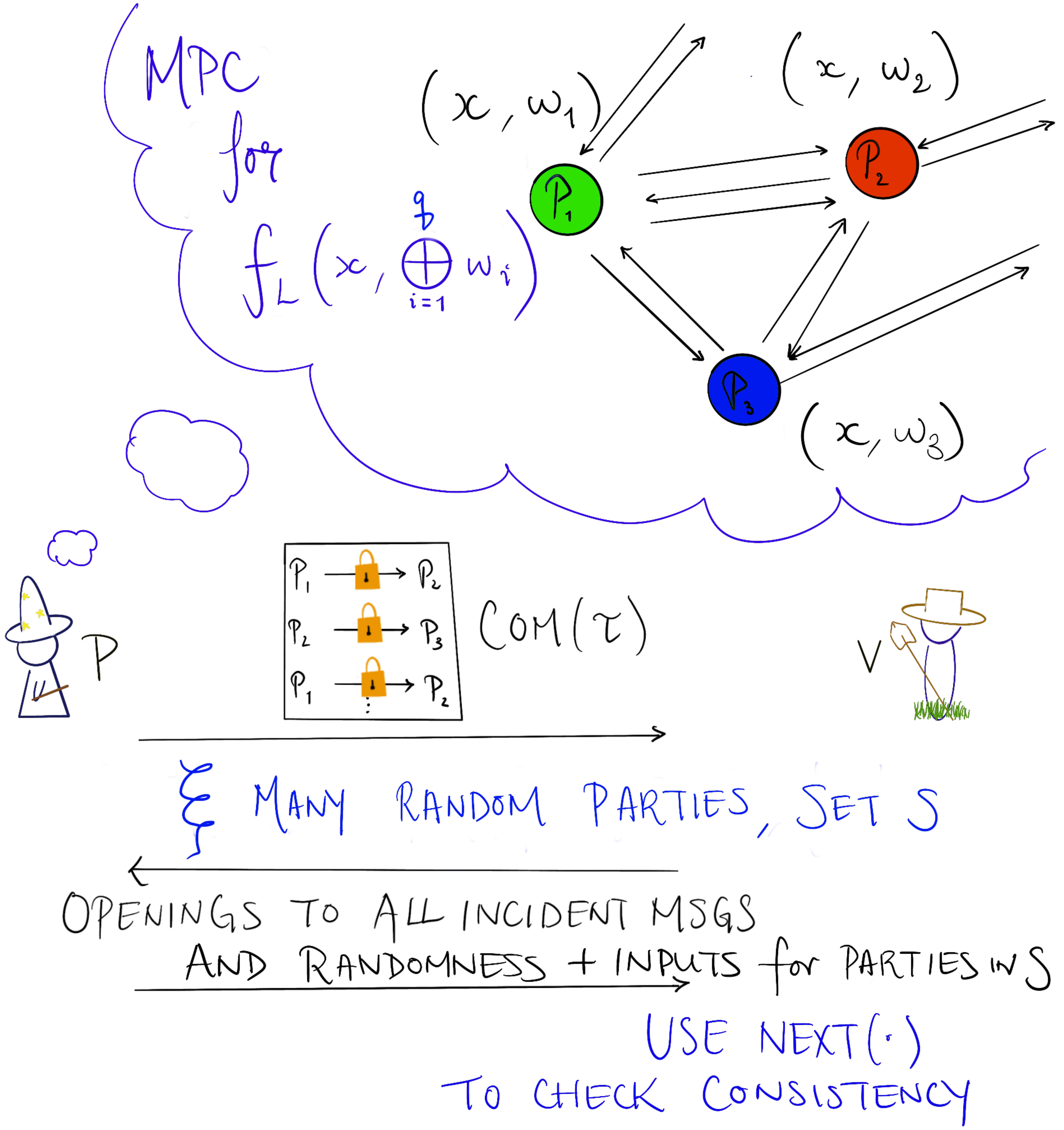
OPENINGS TO ALL INCIDENT MSGS AND RANDOMNESS + INPUTS for P_i

USE NEXT$(\cdot)$ TO CHECK CONSISTENCY

Our Modification of MPC-in-the-Head

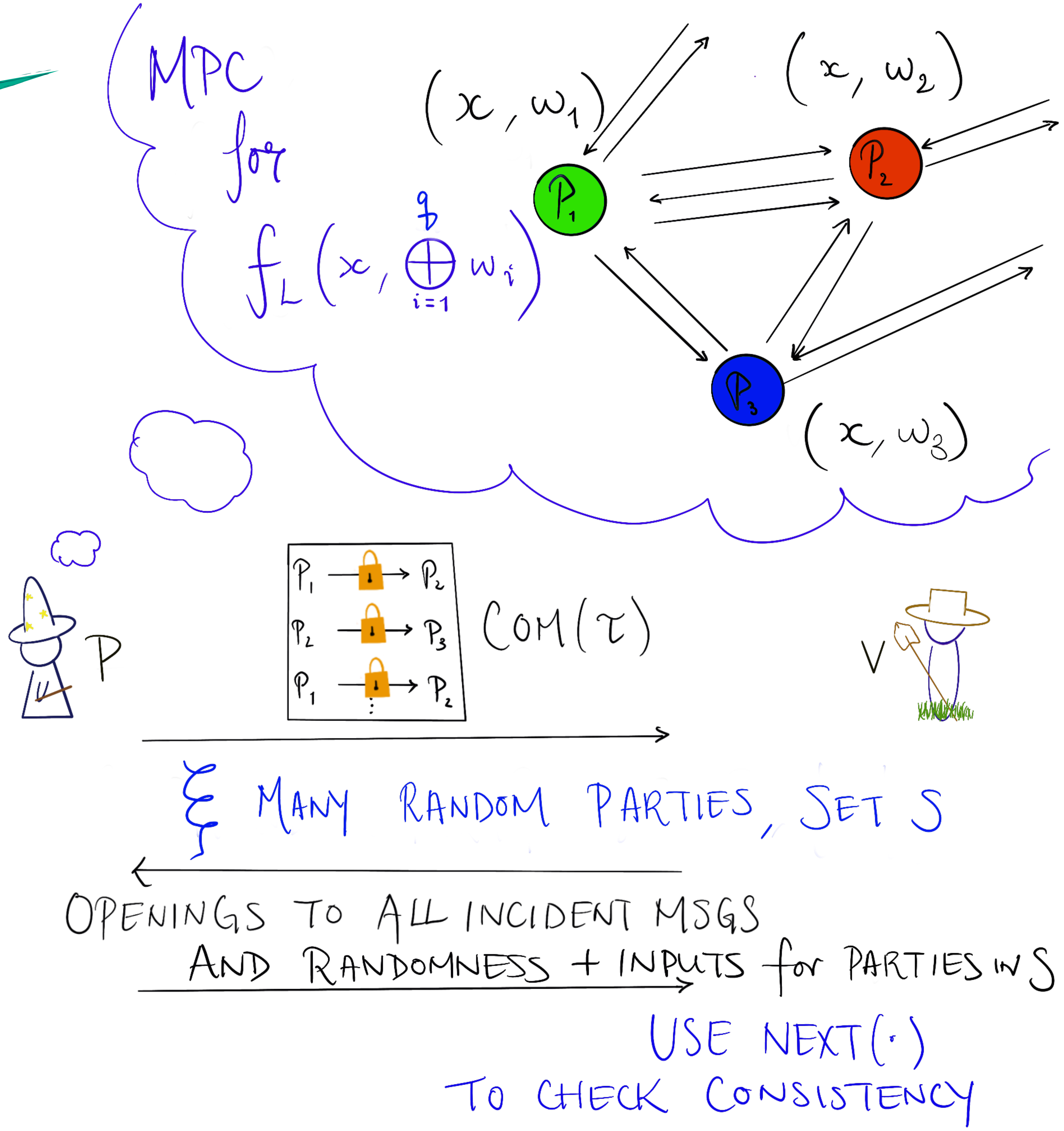


Our Modification of MPC-in-the-Head



Our Modification of MPC-in-the-Head

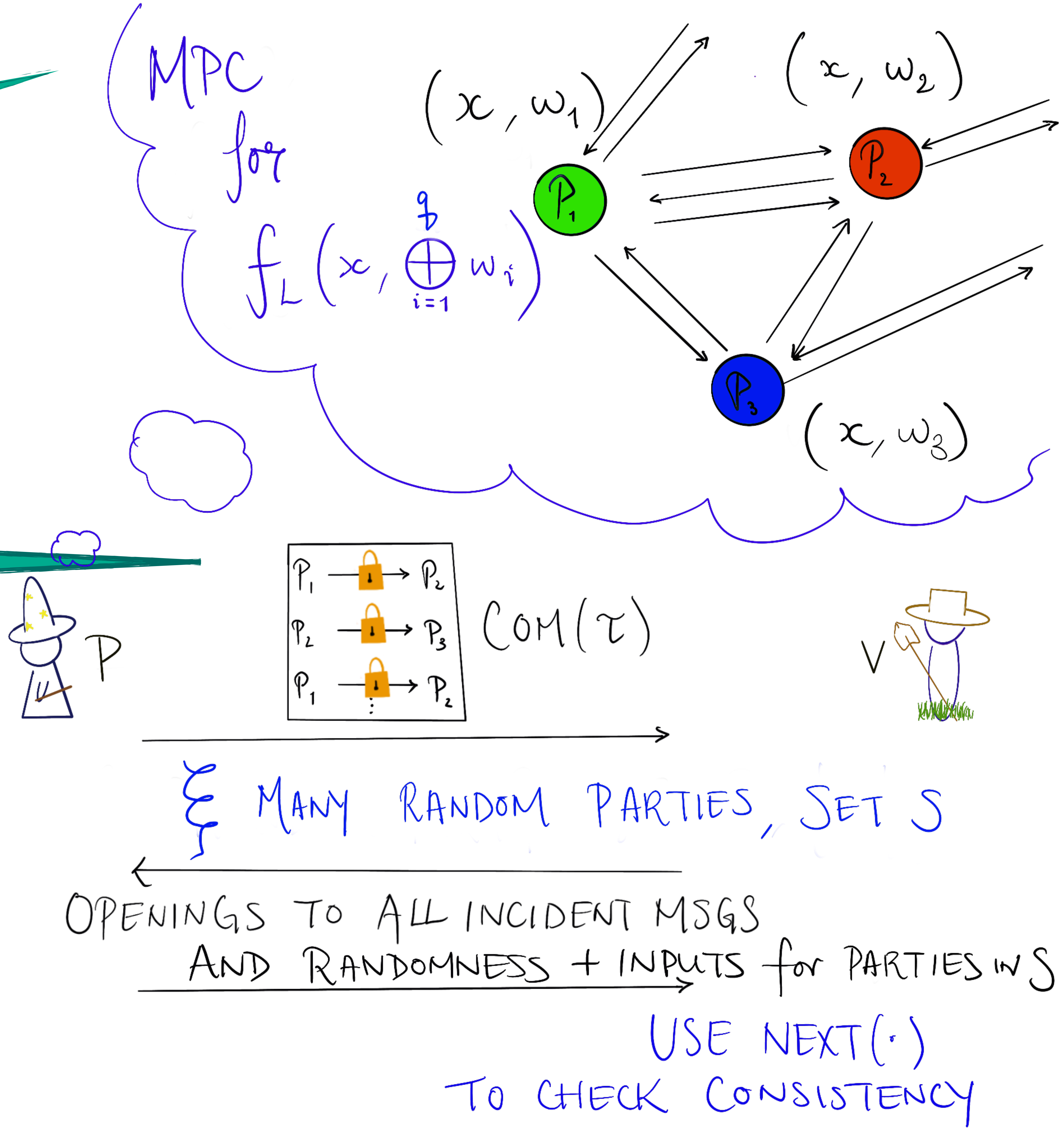
Directly compute NP Verification circuit. Avoids Karp reductions.



Our Modification of MPC-in-the-Head

Directly compute NP Verification circuit. Avoids Karp reductions.

Commit *once* to the transcript τ. Not a parallel repetition!

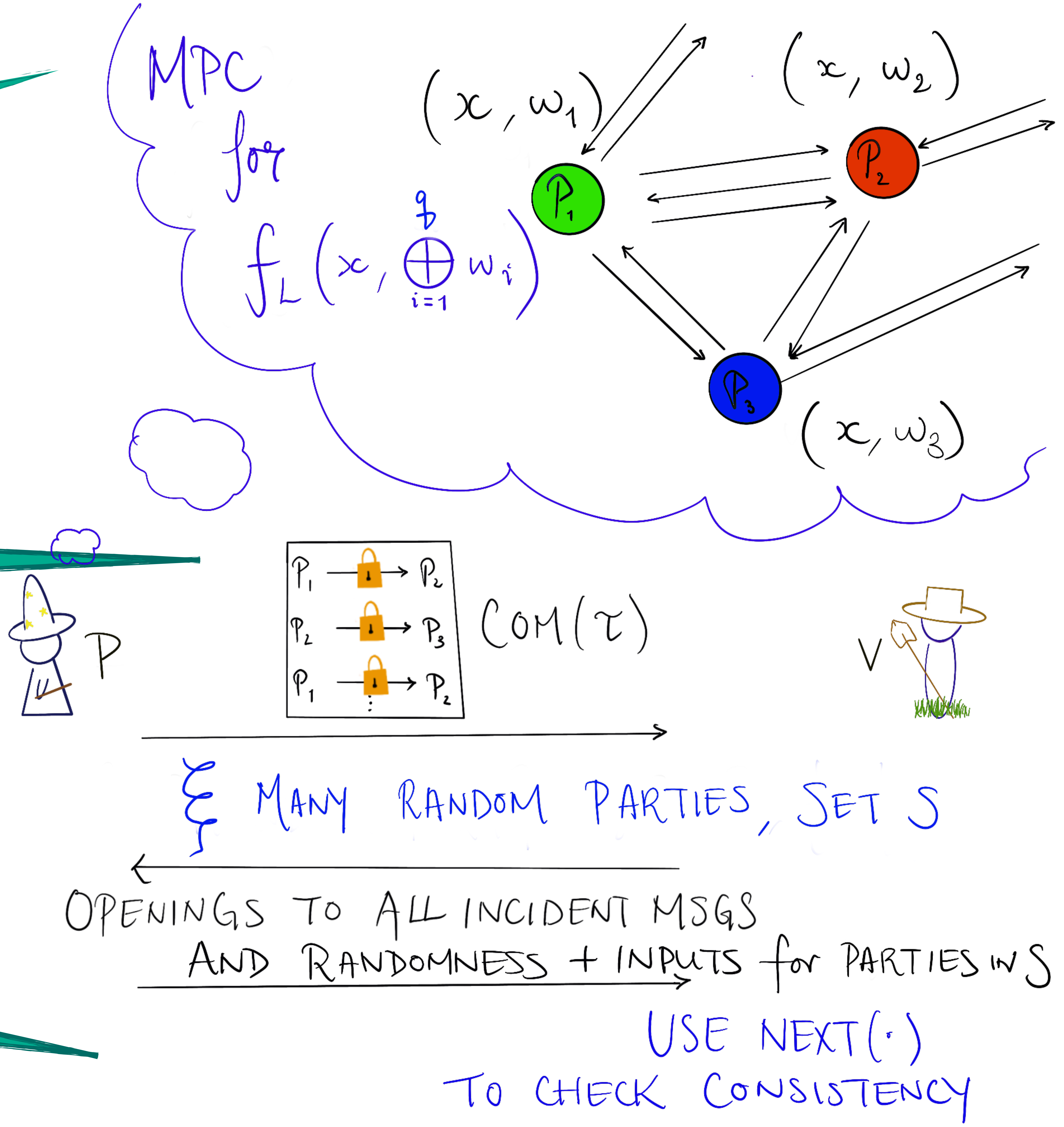


Our Modification of MPC-in-the-Head

Directly compute NP Verification circuit. Avoids Karp reductions.

Commit *once* to the transcript τ. Not a parallel repetition!

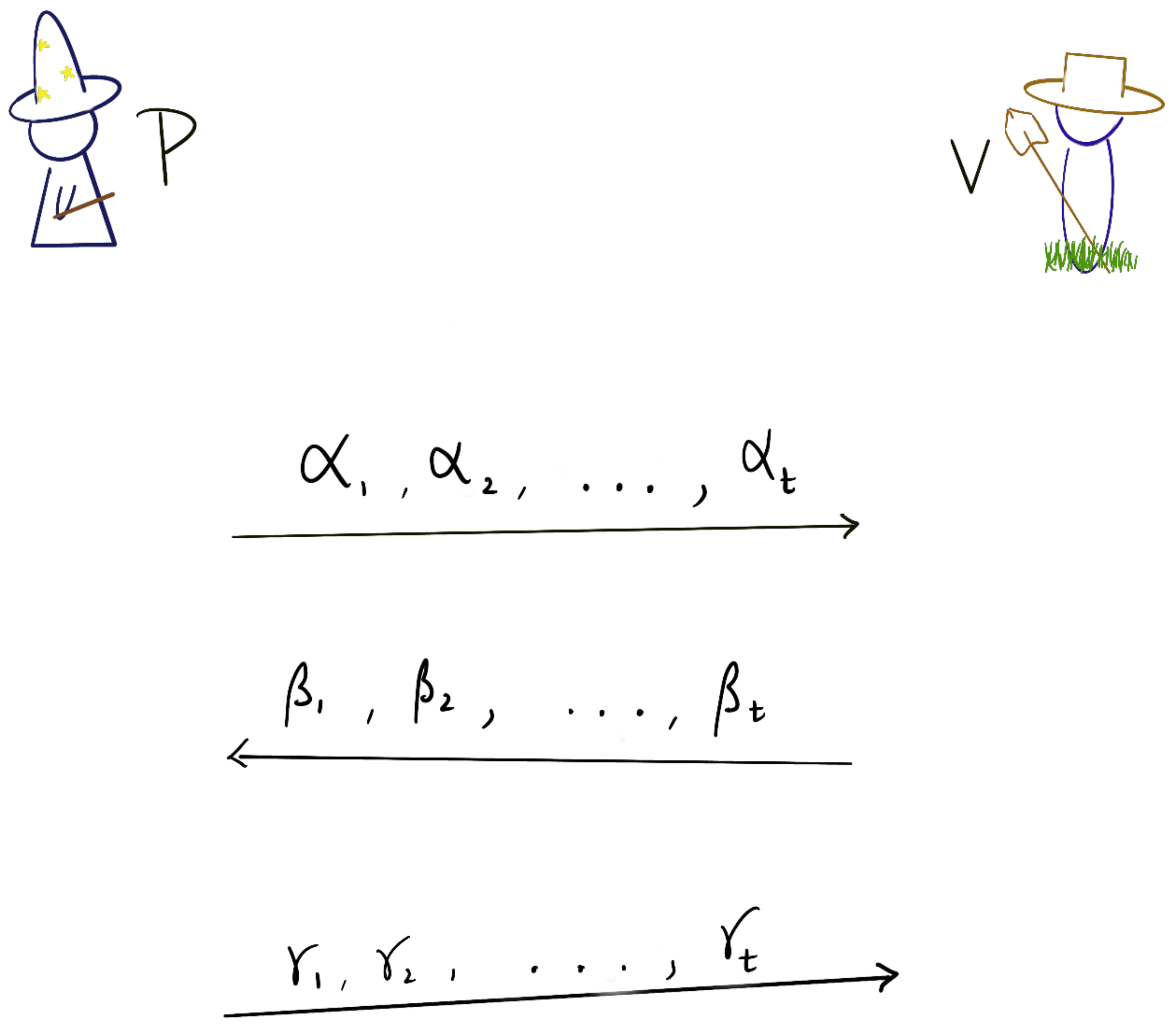
Each party's view is now *independently* verifiable!



A Coding-Theoretic Instantiation of Fiat-Shamir following [HLR21]

Amplifying Soundness via Parallel Repetition

Prior to our work, all known NIZK arguments for NP from LWE considered instantiating the Fiat-Shamir paradigm on a ***parallel repetition*** of a public-coin honest-verifier zero-knowledge interactive proof*:*

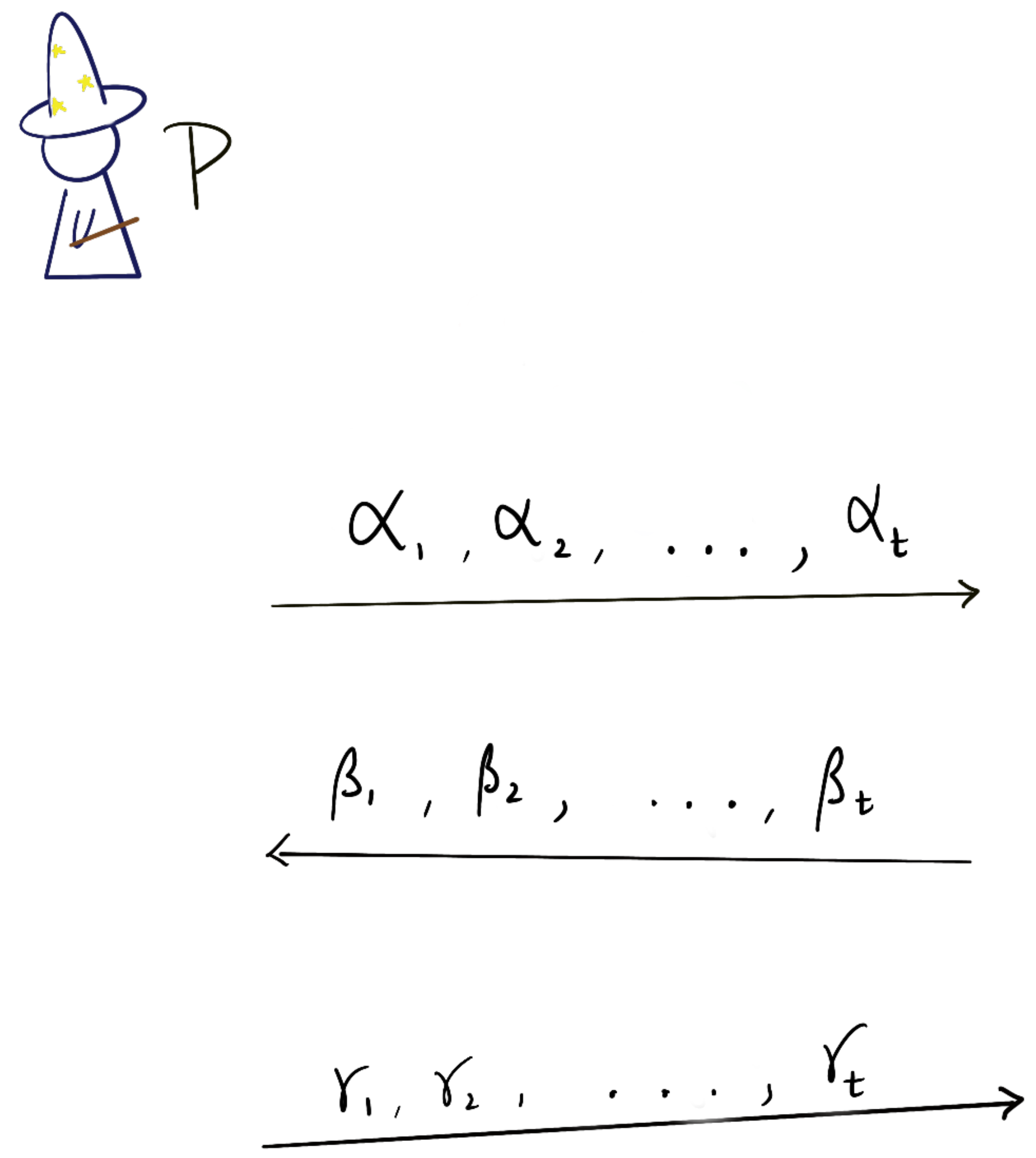


Consider an interactive proof for some NP language L that satisfies:

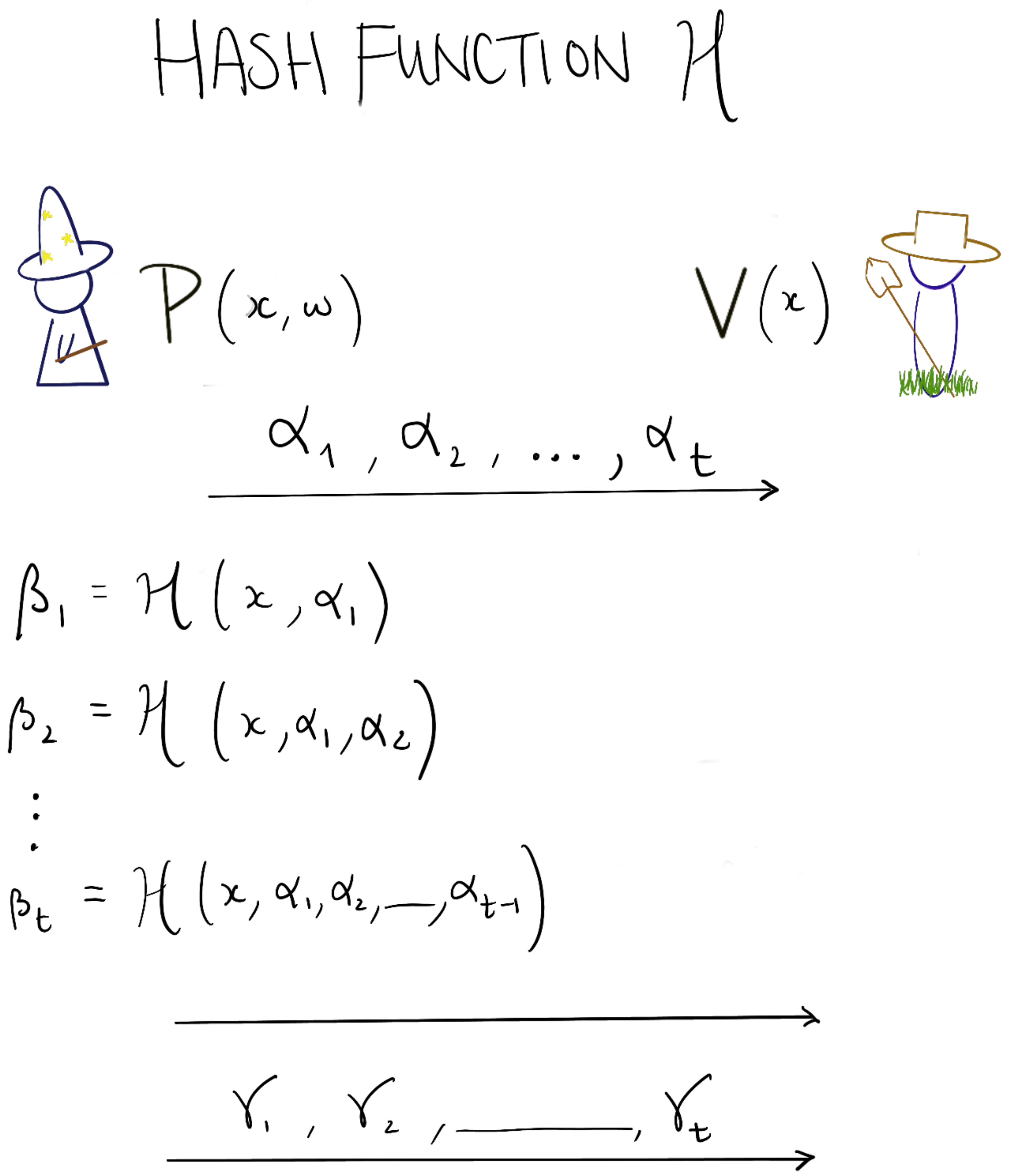
- Completeness
- $negl$-soundness against unbounded provers (statistical soundness)
- Honest-verifier zero-knowledge (HVZK)
- Public coin

Fiat-Shamir Paradigm [FS87]

Prior to our work, all known NIZK arguments for NP from LWE considered instantiating the Fiat-Shamir paradigm on a ***parallel repetition*** of a public-coin honest-verifier zero-knowledge interactive proof:

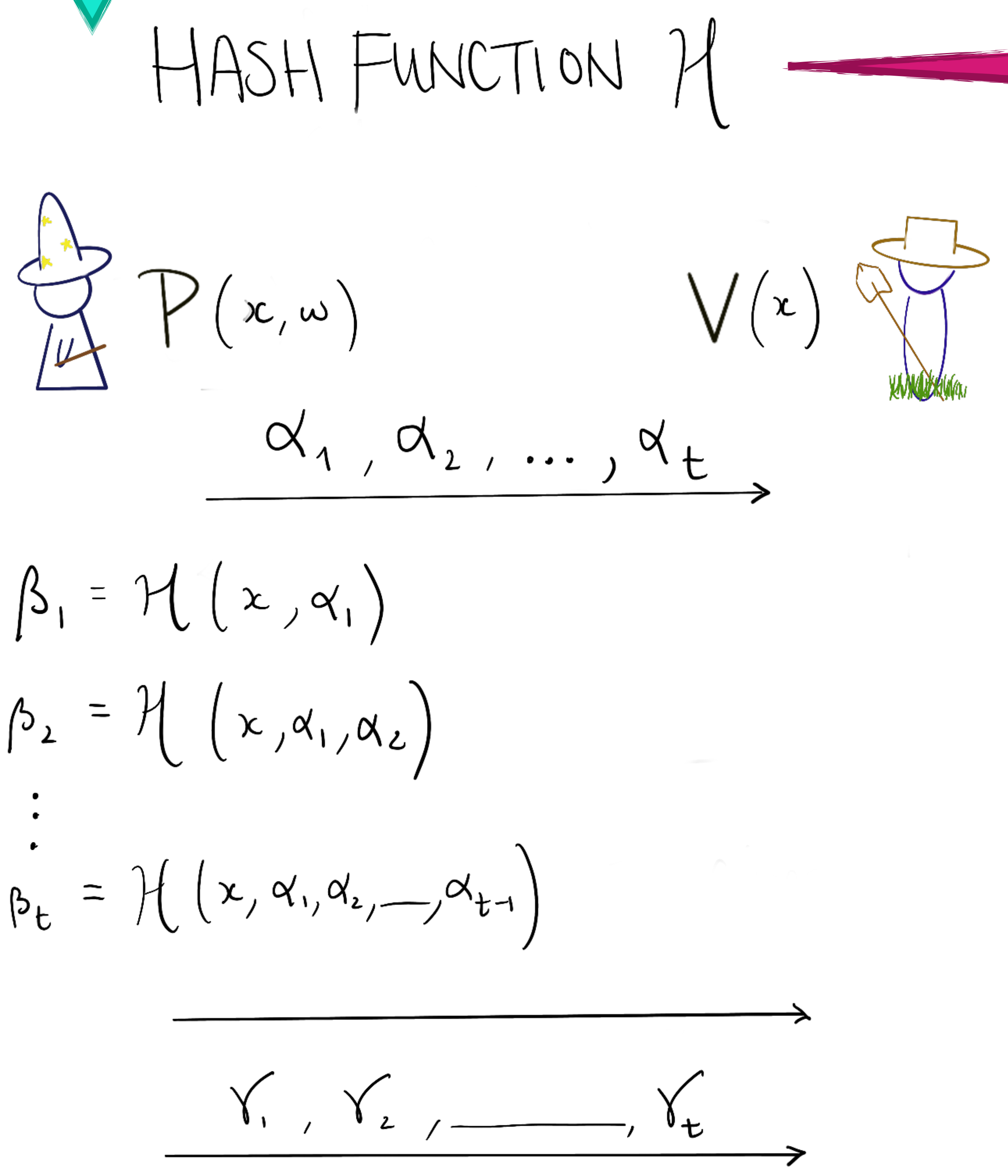


Fiat-Shamir Paradigm [FS87]



Correlation Intractability [CGH04]

Prior to our work, all known NIZK arguments for NP from LWE considered instantiating the Fiat-Shamir paradigm on a ***parallel repetition*** of a public-coin honest-verifier zero-knowledge interactive proof:



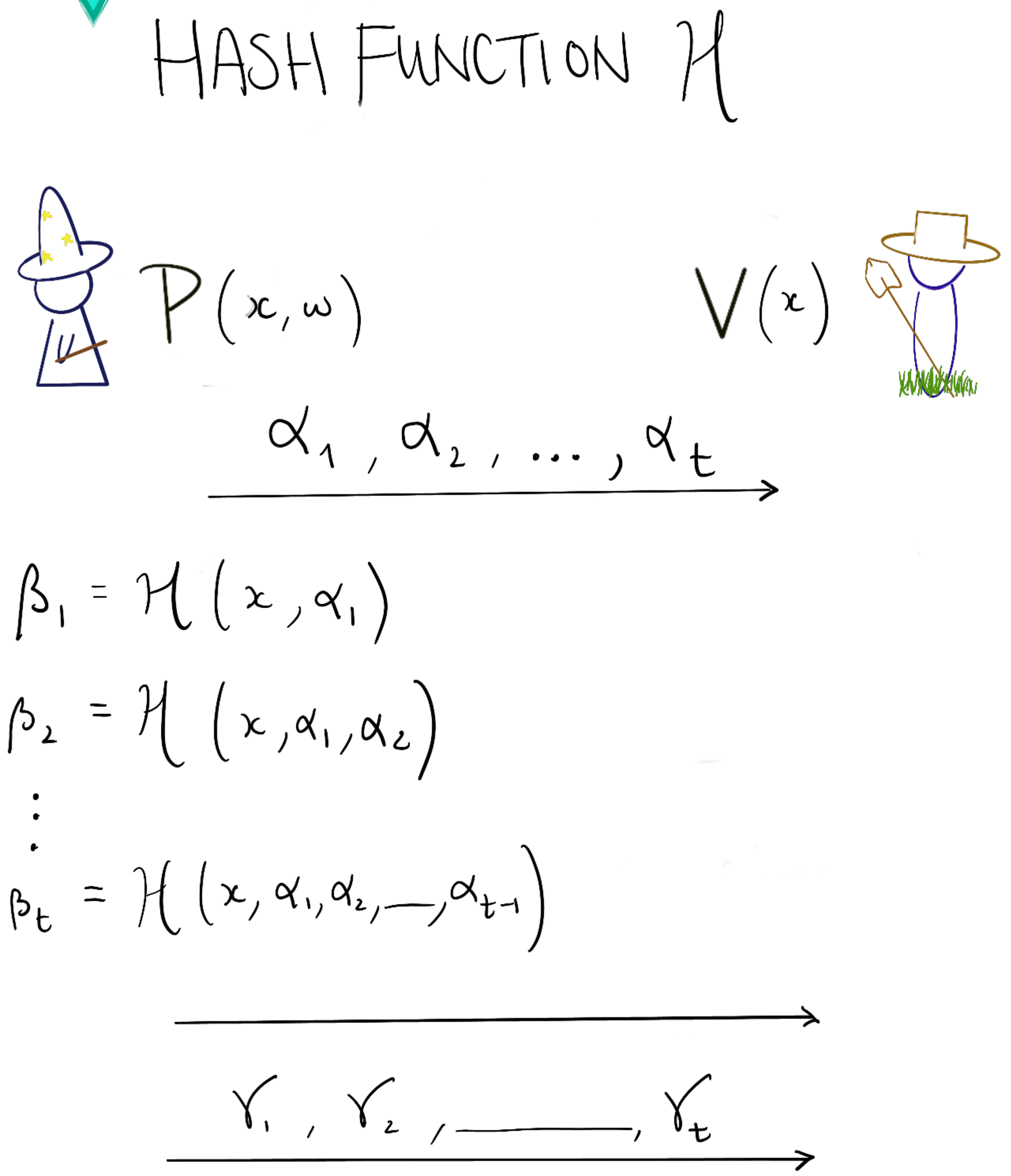
Soundness is preserved if H is sampled from a correlation intractable hash family for an appropriate relation R.

[CGH04] **Def'n:** A hash family $\mathscr{H}$ is *correlation intractable* (CI) for a sparse relation R if for all PPT $\mathscr{A}$

$$Pr_{\substack{h \leftarrow \mathscr{H} \\ x \leftarrow \mathscr{A}(h)}} \left[(x, h(x)) \in R\right] = \text{negl}$$

Correlation Intractability [CGH04]

Prior to our work, all known NIZK arguments for NP from LWE considered instantiating the Fiat-Shamir paradigm on a ***parallel repetition*** of a public-coin honest-verifier zero-knowledge interactive proof:



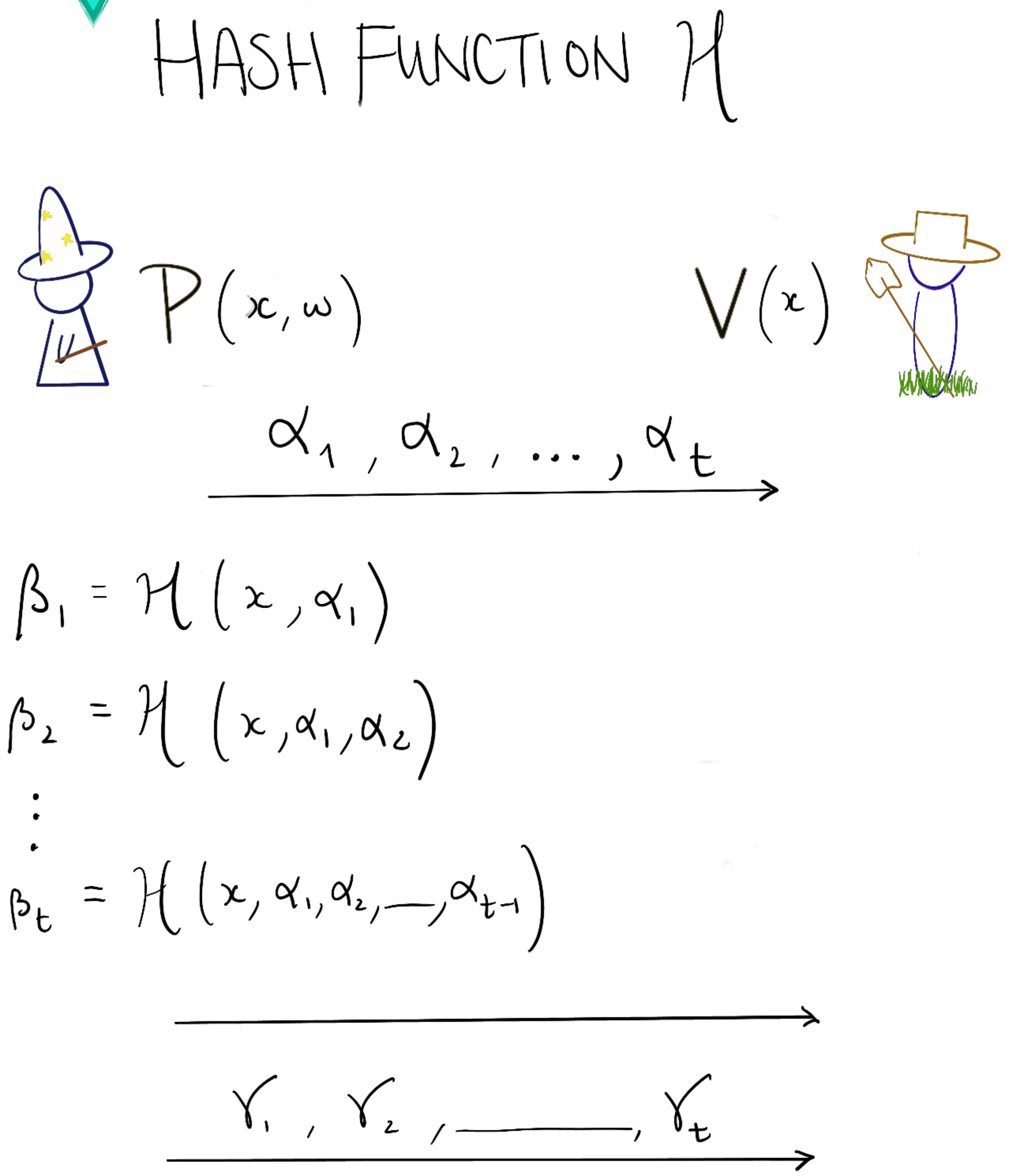
What relation do we consider?

[CGH04] **Def'n:** A hash family $\mathscr{H}$ is *correlation intractable* (CI) for a sparse relation R if for all PPT $\mathscr{A}$

$$\Pr_{\substack{h \leftarrow \mathscr{H} \\ x \leftarrow \mathscr{A}(h)}} \left[(x, h(x)) \in R \right] = \mathrm{negl}$$

Correlation Intractability [CGH04]

Prior to our work, all known NIZK arguments for NP from LWE considered instantiating the Fiat-Shamir paradigm on a ***parallel repetition*** of a public-coin honest-verifier zero-knowledge interactive proof:



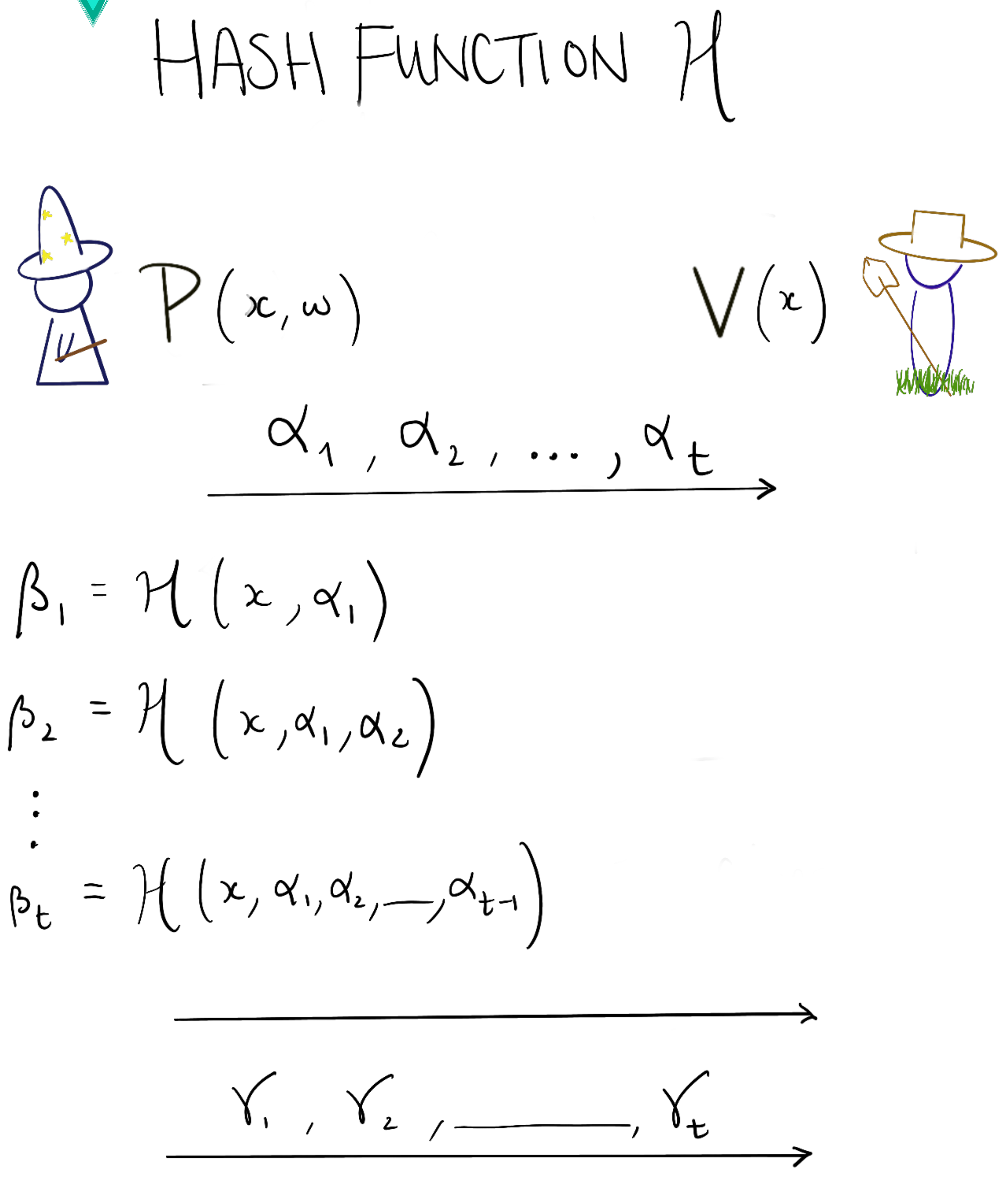
What relation do we consider?

Naively for a statement $x \notin L$:

$$R_x = \left\{ \left((\alpha_1, \dots, \alpha_t), (\beta_1, \dots, \beta_t)\right) : \exists (\gamma_1 \dots, \gamma_t) \text{ s.t. } V(x, \vec{\alpha}, \vec{\beta}, \vec{\gamma}) = 1 \right\}$$

Correlation Intractability [CGH04]

Prior to our work, all known NIZK arguments for NP from LWE considered instantiating the Fiat-Shamir paradigm on a ***parallel repetition*** of a public-coin honest-verifier zero-knowledge interactive proof:



What relation do we consider?

Naively for a statement $x \notin L$:

$$R_x = \left\{ \left((\alpha_1, \ldots, \alpha_t), (\beta_1, \ldots, \beta_t) \right) : \exists (\gamma_1 \ldots, \gamma_t) \text{ s.t. } V(x, \vec{\alpha}, \vec{\beta}, \vec{\gamma}) = 1 \right\}$$

[CCH+19] “*Bad Challenges*” (there’s some response that fools V into accepting)

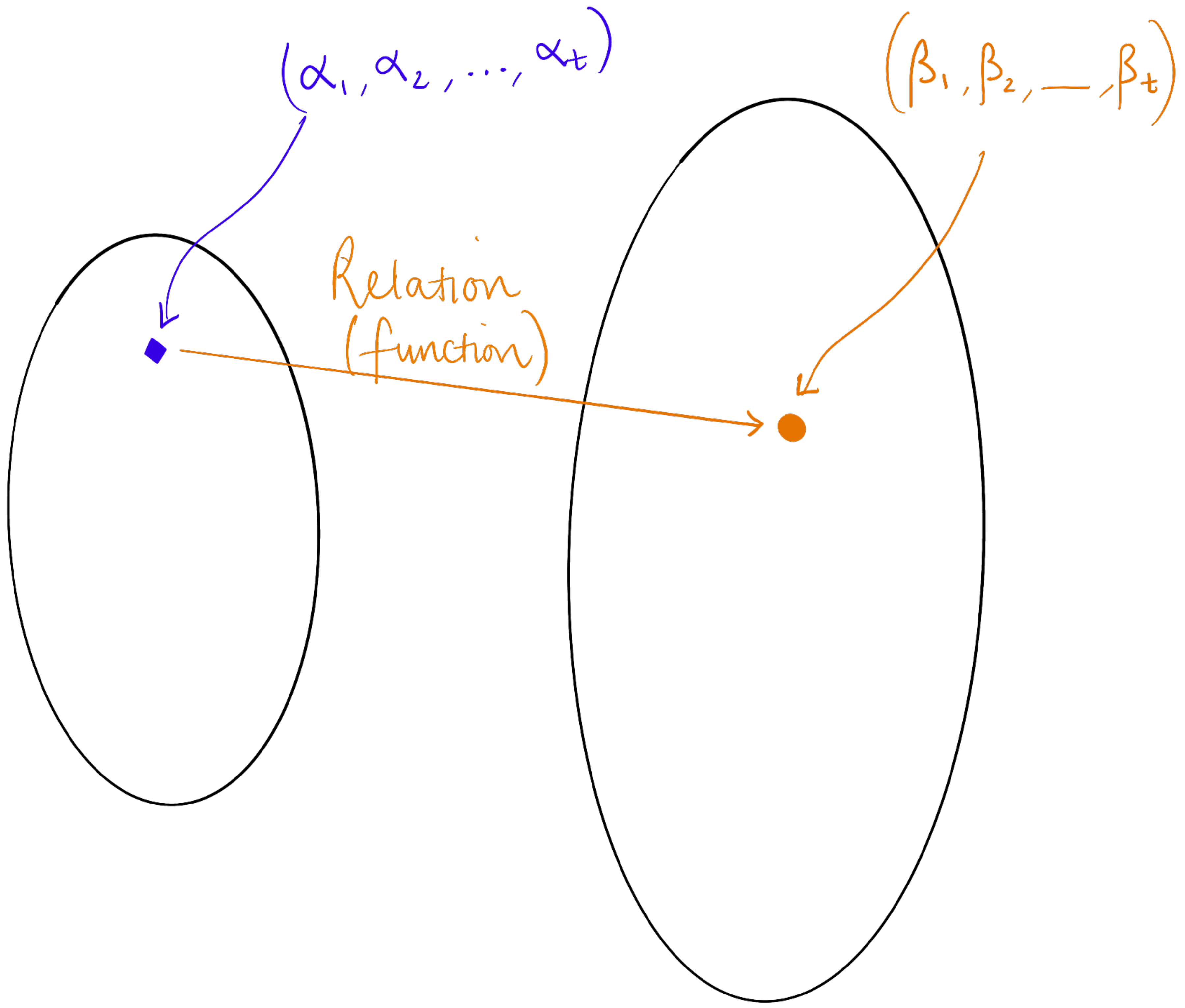
Fiat-Shamir from Coding Theory [HLR21]

Parallel repetition gives a bad challenge set with a nice combinatorial structure.

For a statement $x \notin L$:

$$R_x = \left\{ \left((\alpha_1, \ldots, \alpha_t), (\beta_1, \ldots, \beta_t) \right) : \exists (\gamma_1 \ldots, \gamma_t) \text{ s.t. } V(x, \overrightarrow{\alpha}, \vec{\beta}, \vec{\gamma}) = 1 \right\}$$

[PS19] addresses the case of functions.



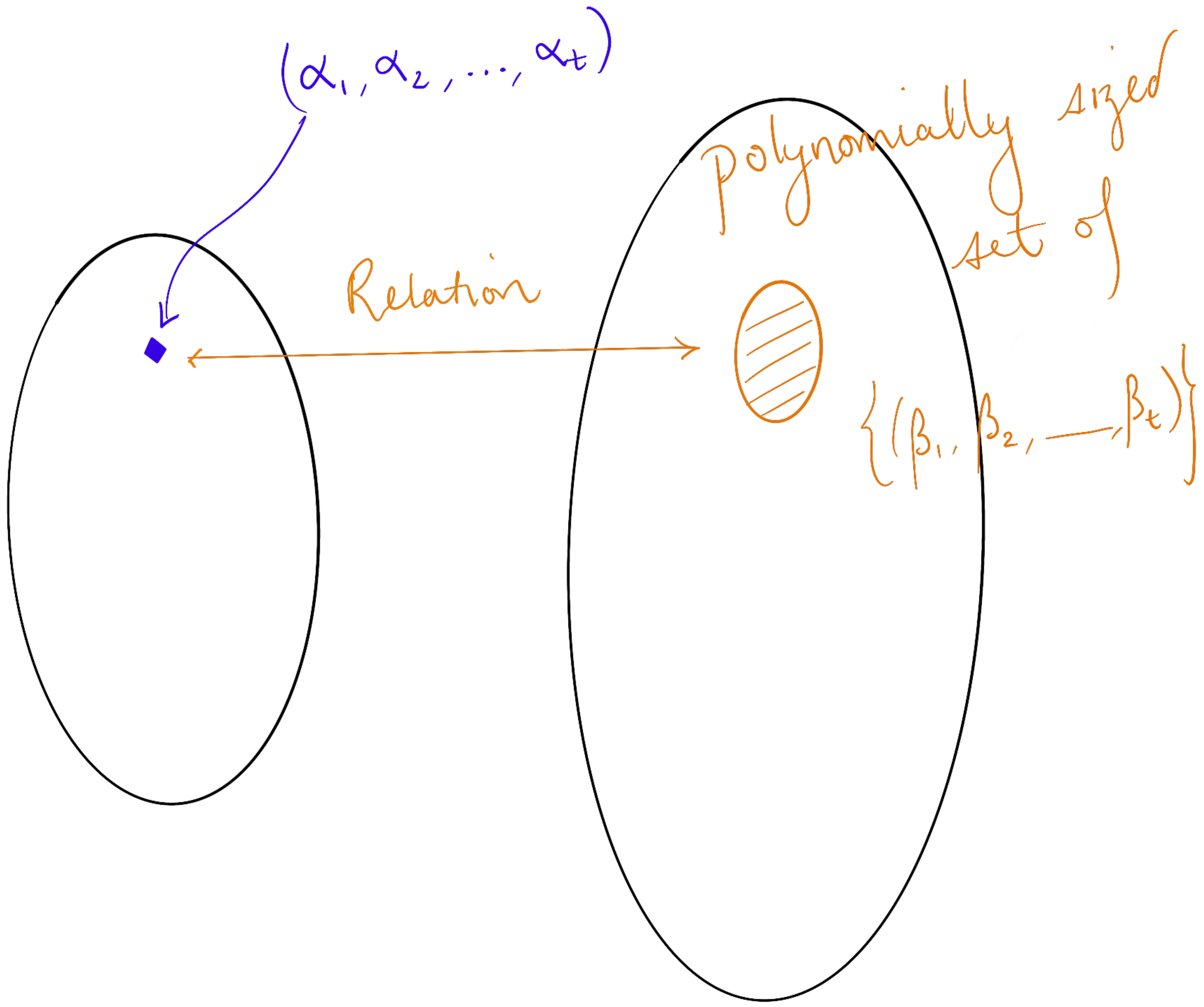
Fiat-Shamir from Coding Theory [HLR21]

Parallel repetition gives a bad challenge set with a nice combinatorial structure.

For a statement $x \notin L$:

$$R_x = \left\{ \left((\alpha_1, \ldots, \alpha_t), (\beta_1, \ldots, \beta_t)\right) : \exists(\gamma_1 \ldots, \gamma_t) \text{ s.t. } V(x, \vec{\alpha}, \vec{\beta}, \vec{\gamma}) = 1 \right\}$$

By a guessing reduction, [CCH+19, PS19] also addresses the case of polynomially many bad challenges.



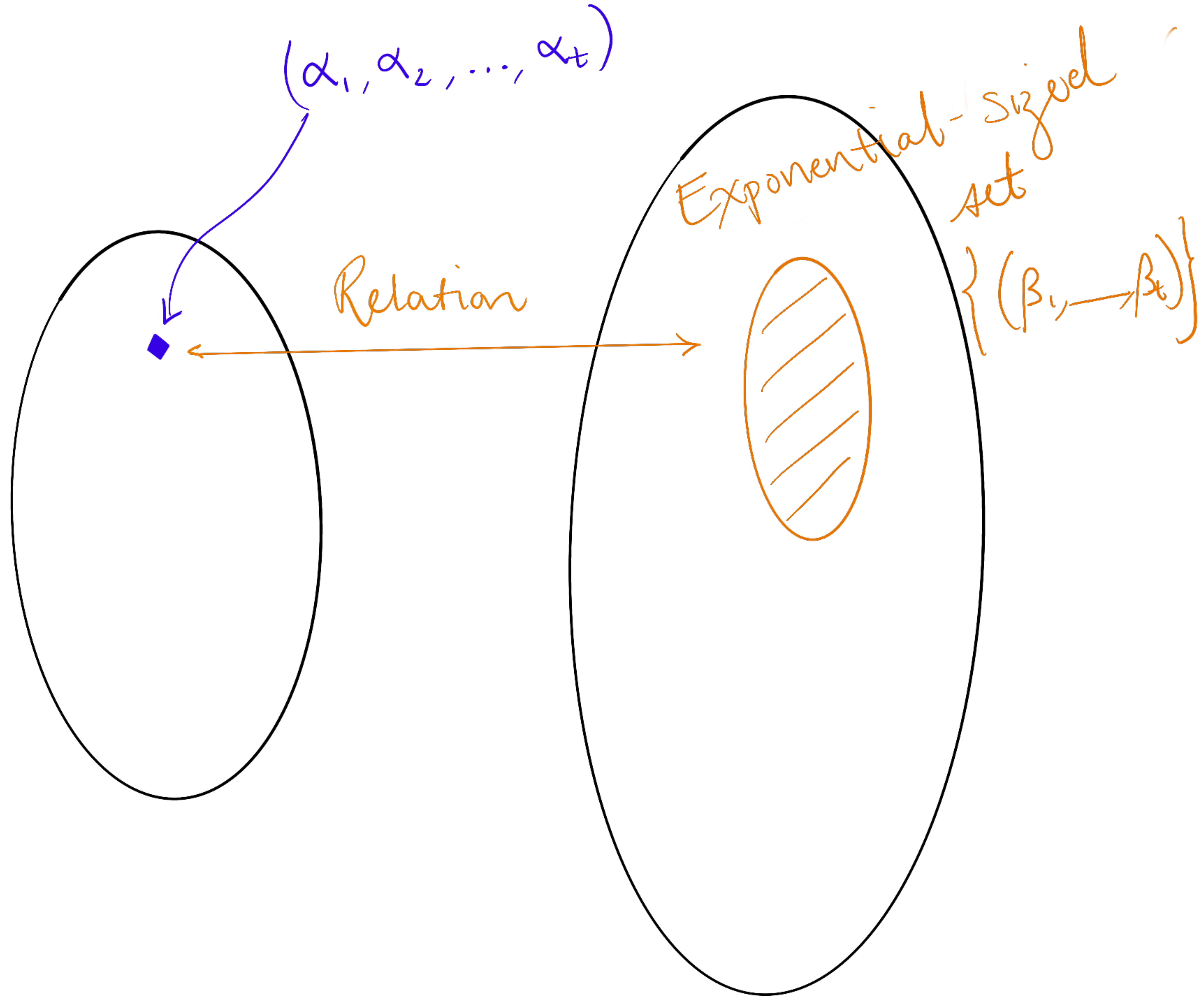
Fiat-Shamir from Coding Theory [HLR21]

Parallel repetition gives a bad challenge set with a nice combinatorial structure.

For a statement $x \notin L$:

$$R_x = \left\{ \left((\alpha_1, \ldots, \alpha_t), (\beta_1, \ldots, \beta_t) \right) : \exists (\gamma_1 \ldots, \gamma_t) \text{ s.t. } V(x, \vec{\alpha}, \vec{\beta}, \vec{\gamma}) = 1 \right\}$$

Too many bad challenges for the techniques of [CCH+19, PS19].



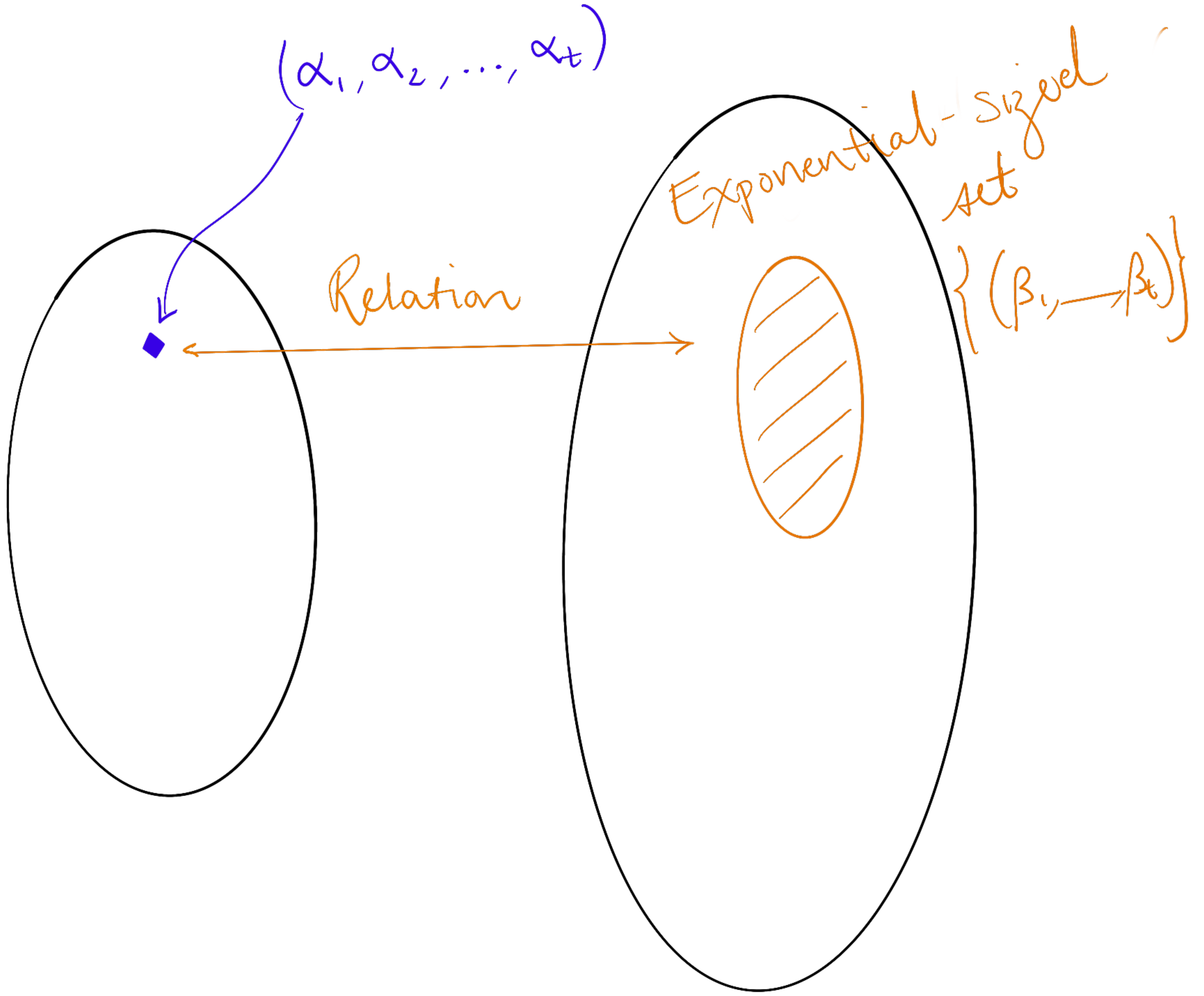
Fiat-Shamir from Coding Theory [HLR21]

Parallel repetition gives a bad challenge set with a nice combinatorial structure.

For a statement $x \notin L$:

$$R_x = \left\{ \big((\alpha_1, \ldots, \alpha_t), (\beta_1, \ldots, \beta_t)\big) : \exists(\gamma_1 \ldots, \gamma_t) \text{ s.t. } V(x, \vec{\alpha}, \vec{\beta}, \vec{\gamma}) = 1 \right\}$$

[HLR21] Use the product structure!



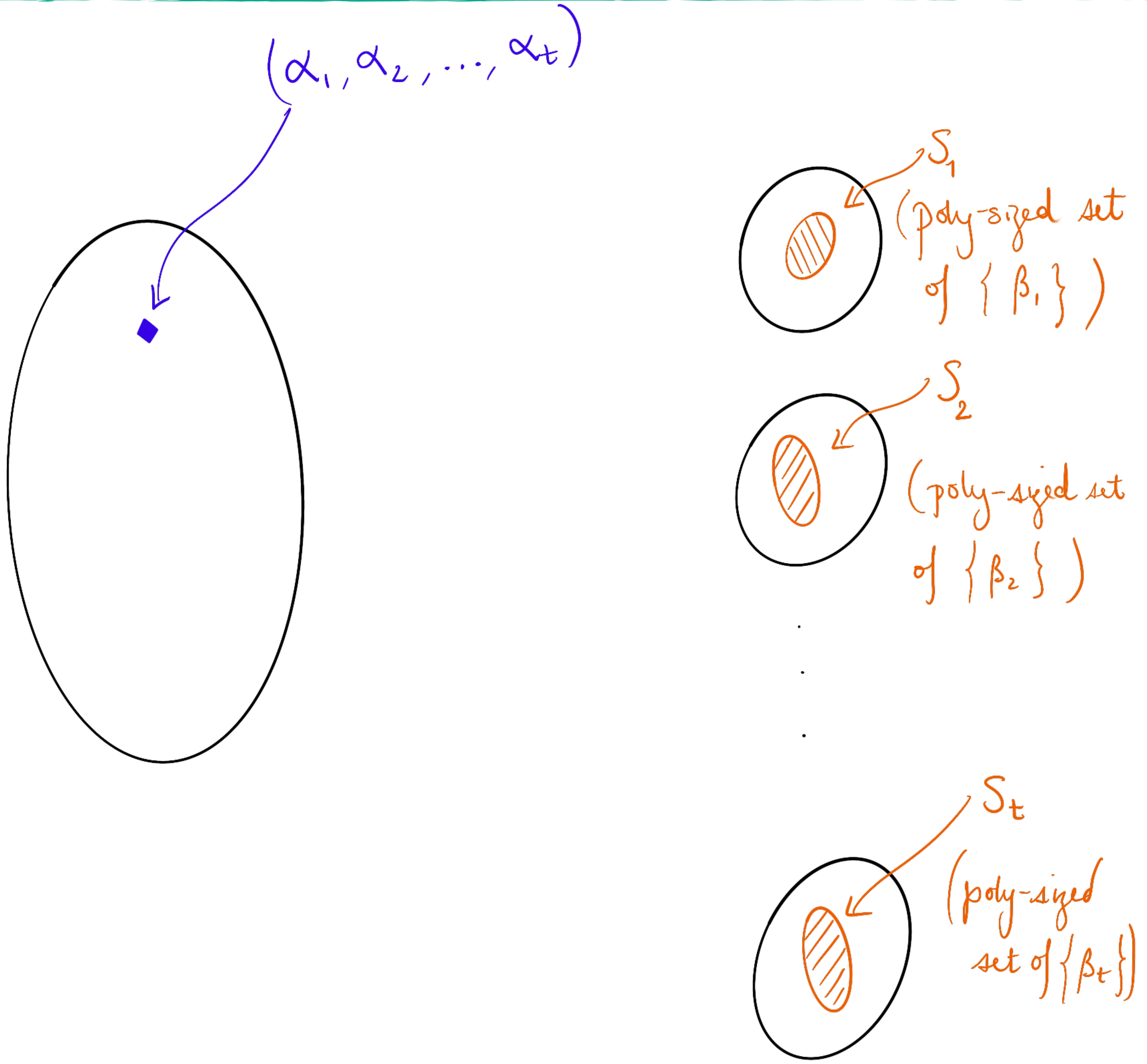
Fiat-Shamir from Coding Theory [HLR21]

Parallel repetition gives a bad challenge set with a nice combinatorial structure.

For a statement $x \notin L$:

$$R_x = \left\{ \left((\alpha_1, \ldots, \alpha_t), (\beta_1, \ldots, \beta_t)\right) : \exists (\gamma_1 \ldots, \gamma_t) \text{ s.t. } V(x, \vec{\alpha}, \vec{\beta}, \vec{\gamma}) = 1 \right\}$$

[HLR21] Use the product structure!



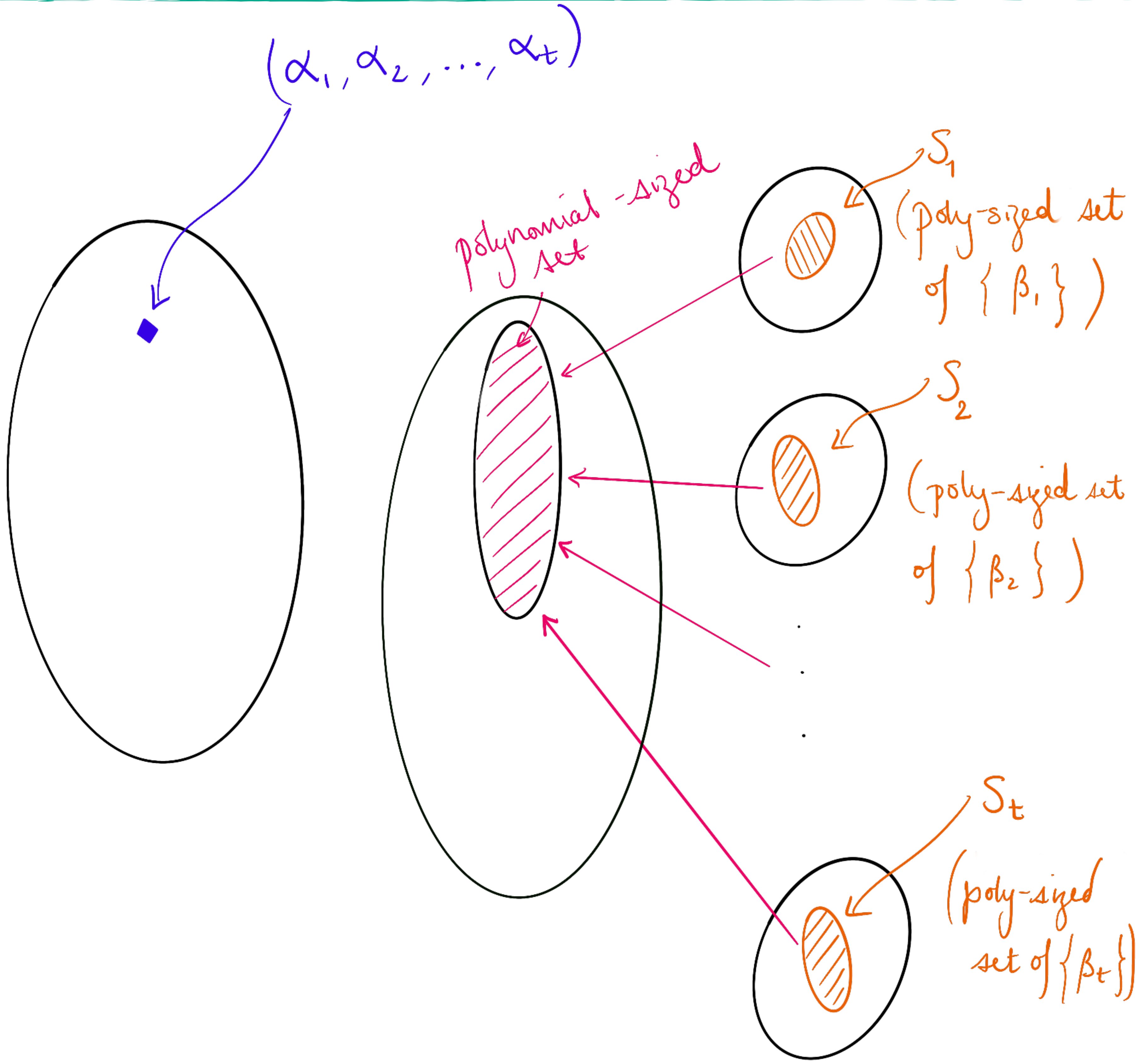
Fiat-Shamir from Coding Theory [HLR21]

Parallel repetition gives a bad challenge set with a nice combinatorial structure.

For a statement $x \notin L$:

$$R_x = \left\{ \left((\alpha_1, \ldots, \alpha_t), (\beta_1, \ldots, \beta_t) \right) : \exists (\gamma_1 \ldots, \gamma_t) \text{ s.t. } V(x, \vec{\alpha}, \vec{\beta}, \vec{\gamma}) = 1 \right\}$$

[HLR21] Use the product structure!



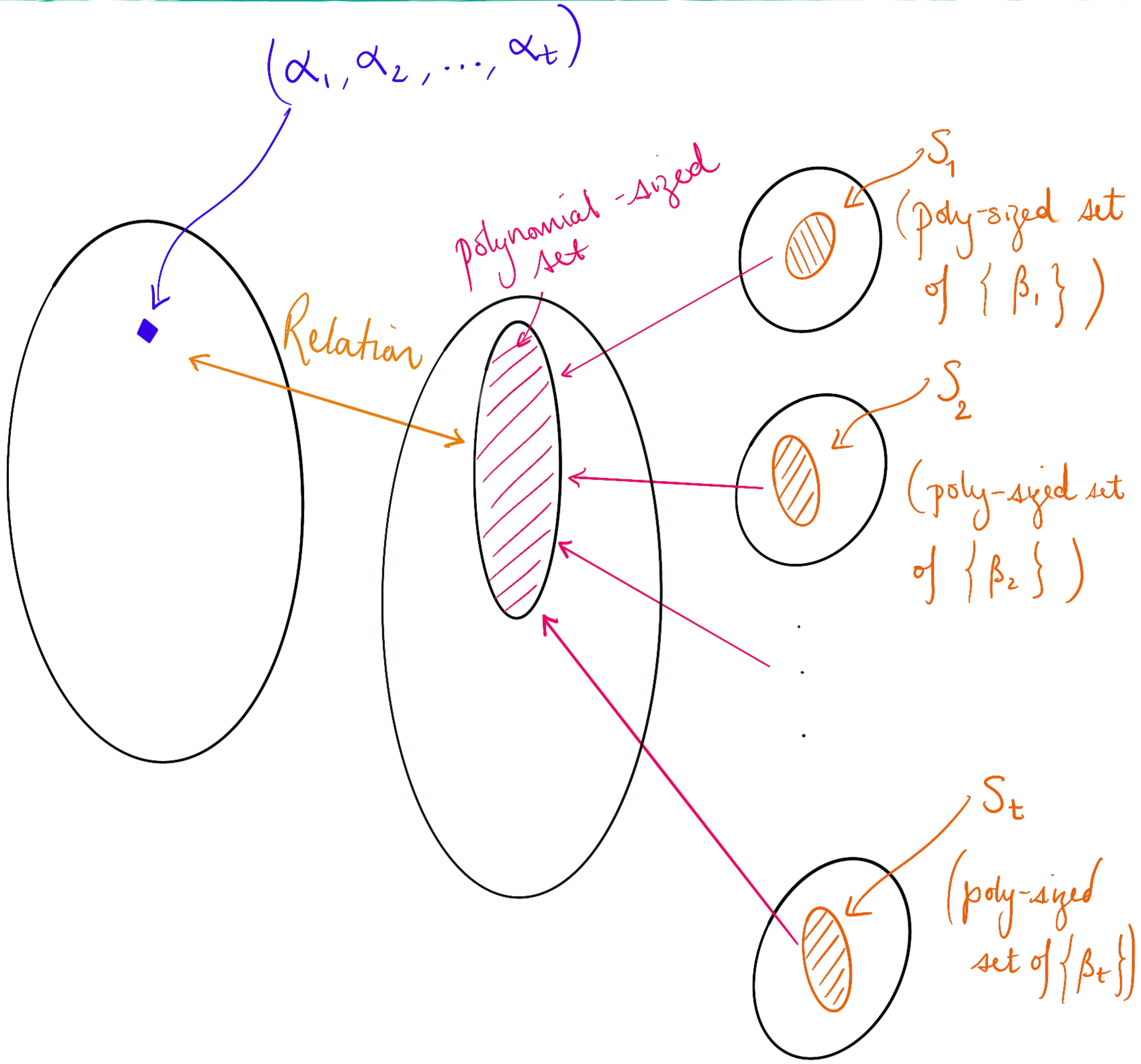
Fiat-Shamir from Coding Theory [HLR21]

Parallel repetition gives a bad challenge set with a nice combinatorial structure.

For a statement $x \notin L$:

$$R_x = \left\{ \left((\alpha_1, \ldots, \alpha_t), (\beta_1, \ldots, \beta_t) \right) : \exists (\gamma_1 \ldots, \gamma_t) \text{ s.t. } V(x, \vec{\alpha}, \vec{\beta}, \vec{\gamma}) = 1 \right\}$$

[HLR21] Use the product structure!



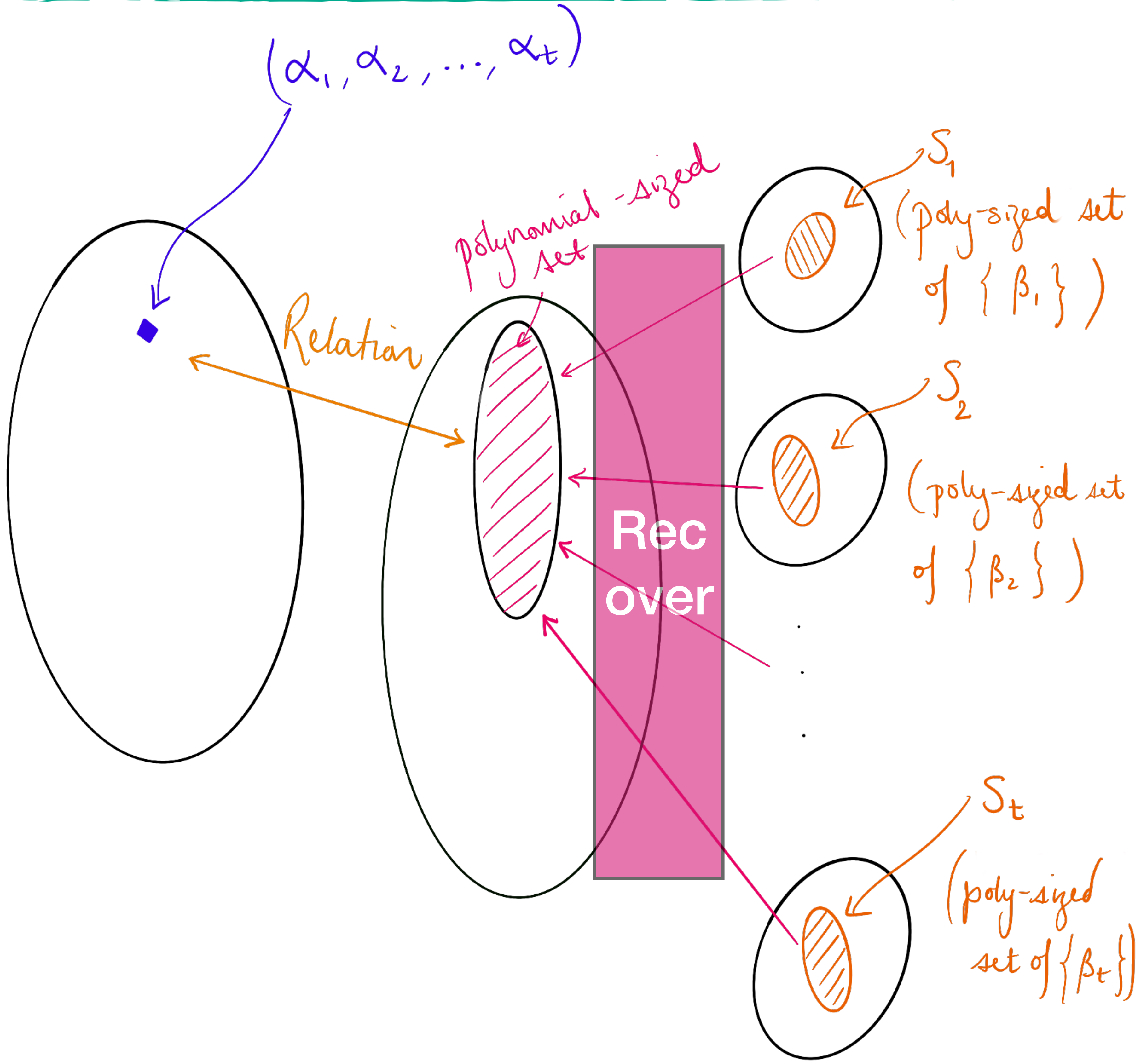
Fiat-Shamir from Coding Theory [HLR21]

Parallel repetition gives a bad challenge set with a nice combinatorial structure.

For a statement $x \notin L$:

$$R_x = \left\{ \left((\alpha_1, \ldots, \alpha_t), (\beta_1, \ldots, \beta_t)\right) : \exists(\gamma_1 \ldots, \gamma_t) \text{ s.t. } V(x, \overline{\alpha}, \vec{\beta}, \vec{\gamma}) = 1 \right\}$$

[HLR21] This is exactly list recovery!
Use a list-recoverable code!



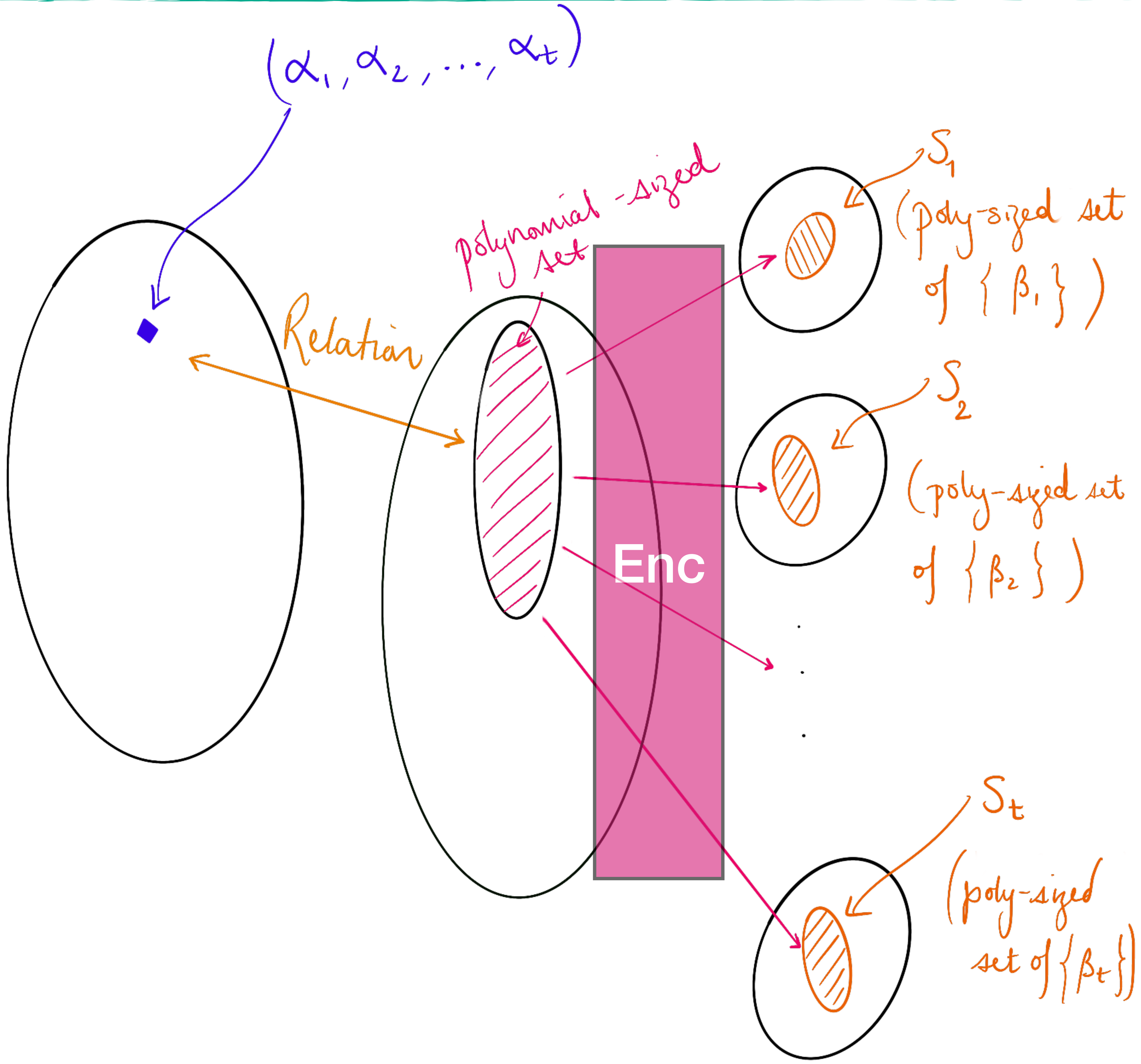
Fiat-Shamir from Coding Theory [HLR21]

Parallel repetition gives a bad challenge set with a nice combinatorial structure.

For a statement $x \notin L$:

$$R_x = \left\{ \big((\alpha_1, \ldots, \alpha_t), r\big) : (\text{Encode}(r))_i \in S_i \right\}$$

[HLR21] This is exactly list recovery!
Use a list-recoverable code!



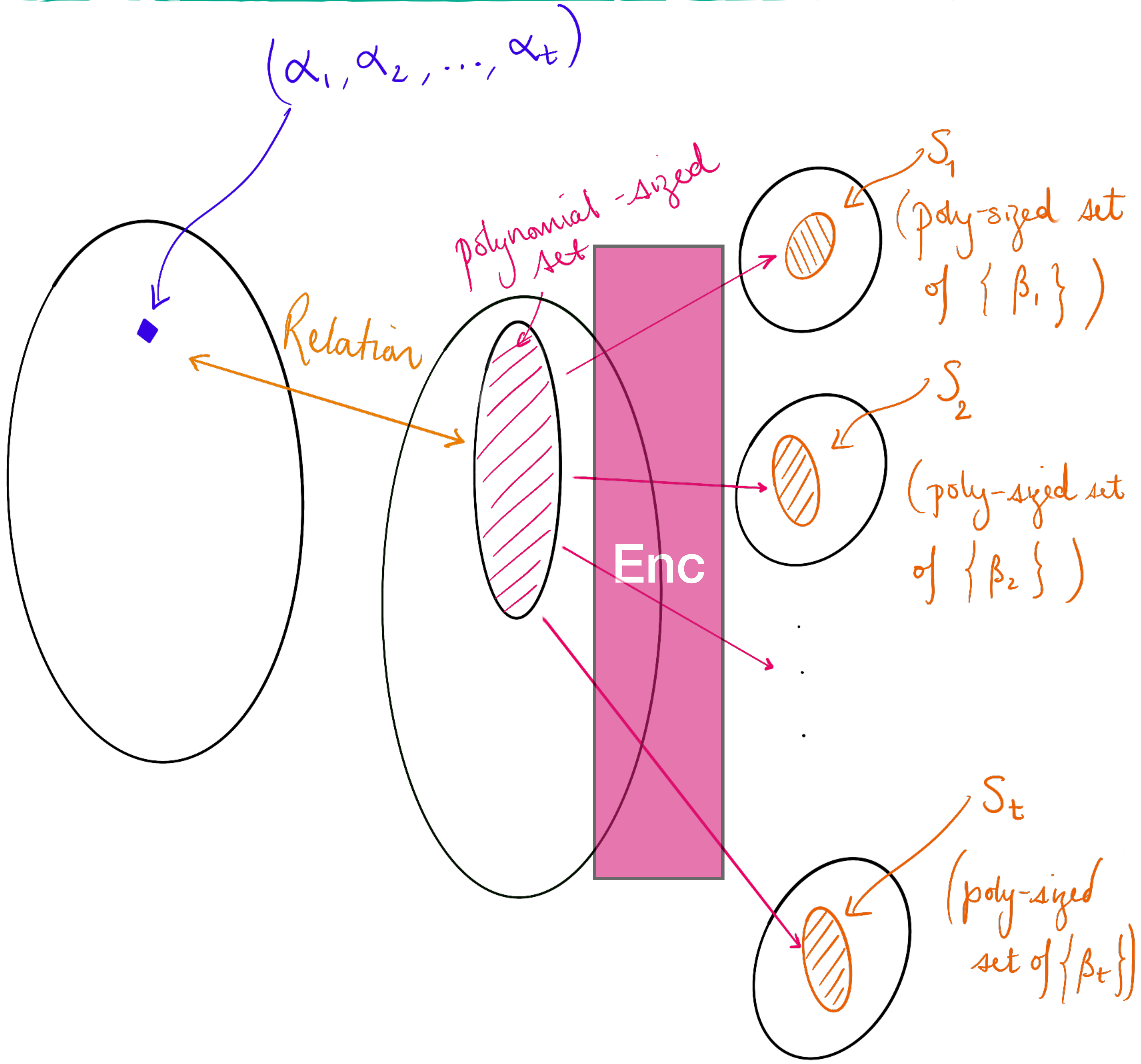
Fiat-Shamir from Coding Theory [HLR21]

Parallel repetition gives a bad challenge set with a nice combinatorial structure.

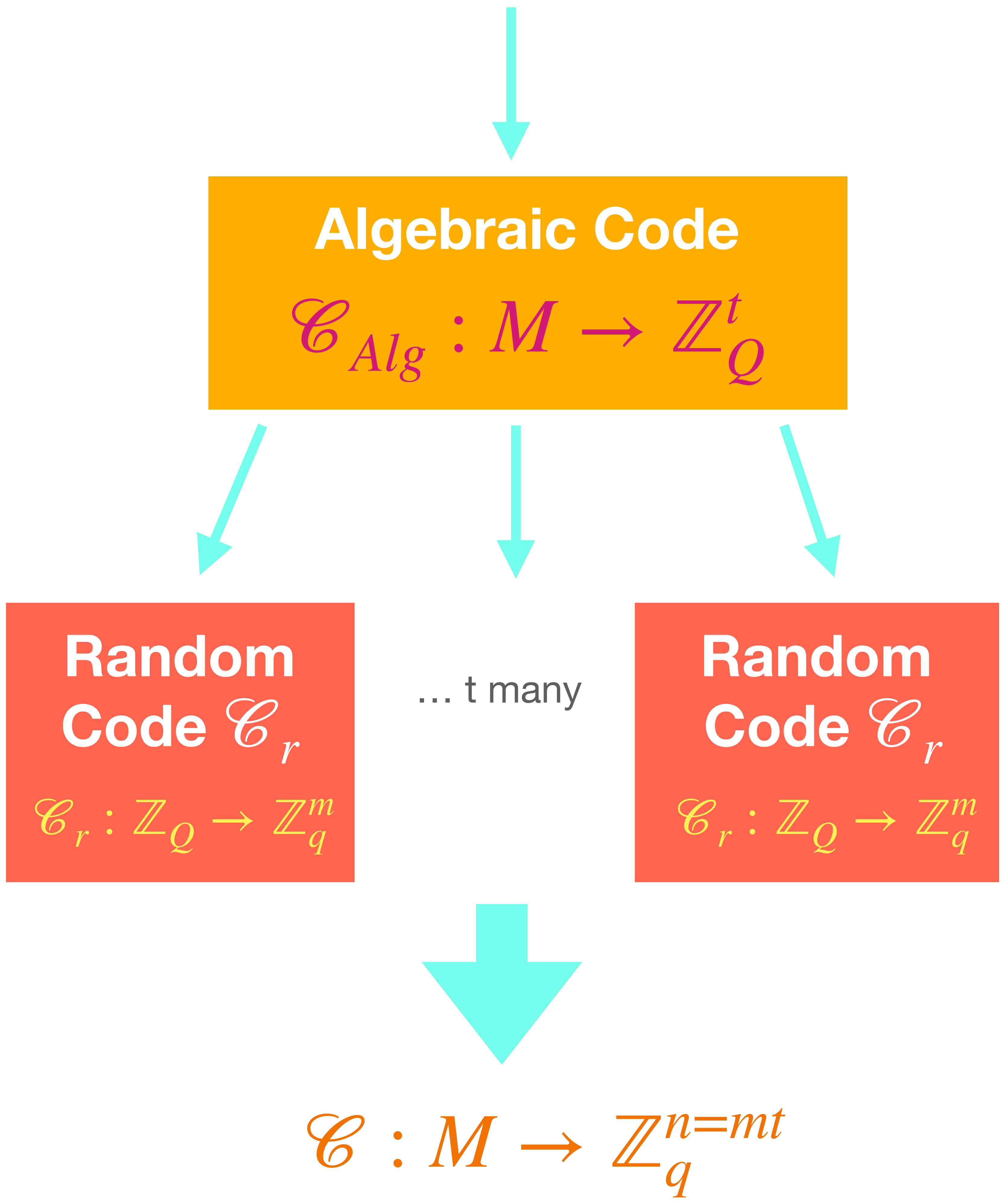
For a statement $x \notin L$:

$$R_x = \left\{ \left((\alpha_1, \ldots, \alpha_t), r\right) : (\mathsf{Encode}(r))_i \in S_i \right\}$$

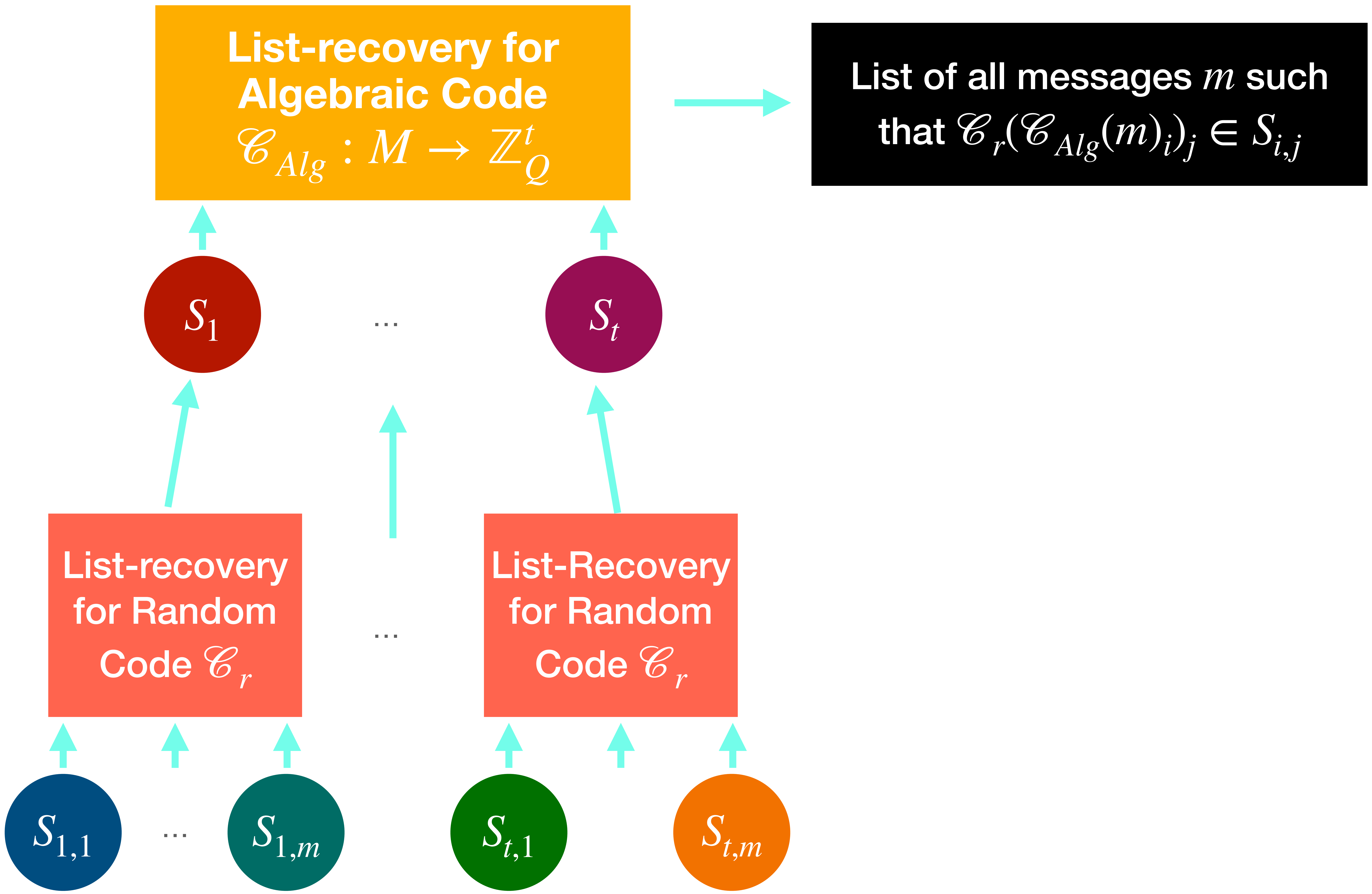
[HLR21] Use Parvaresh-Vardy code concatenated with a single random code.



Code Contenation



List-Recovery for Concatenated Codes



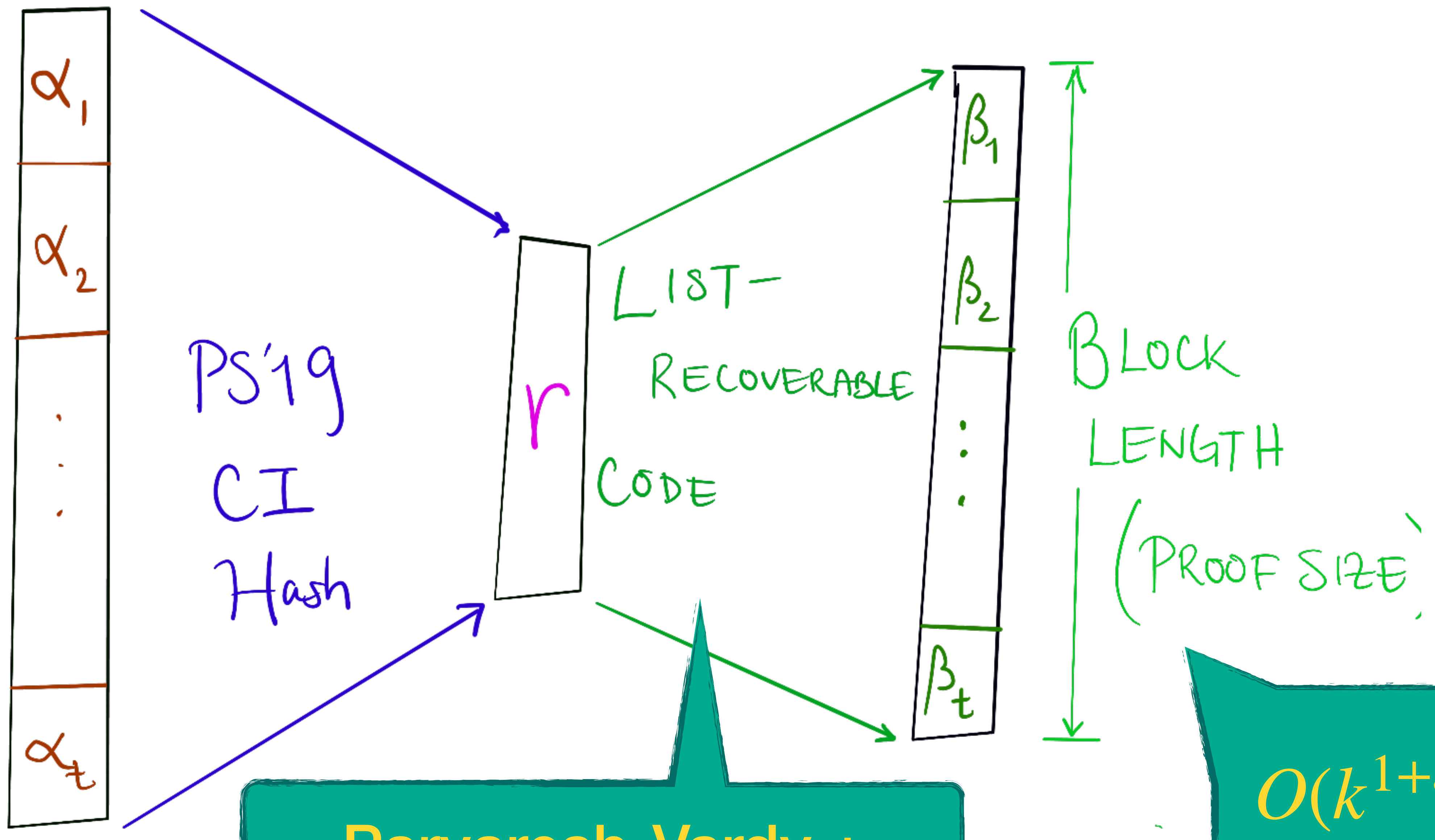
Fiat-Shamir from Coding Theory [HLR21]

Parallel repetition gives a bad challenge set with a nice combinatorial structure.

For a statement $x \notin L$:

$$R_x = \left\{ \left((\alpha_1, \ldots, \alpha_t), (\beta_1, \ldots, \beta_t) \right) : \exists (\gamma_1 \ldots, \gamma_t) \text{ s.t. } V(x, \vec{\alpha}, \vec{\beta}, \vec{\gamma}) = 1 \right\}$$

[HLR21] This is a CI hash for the desired relation.



Parvaresh-Vardy + Single Random Code

$O(k^{1+\varepsilon})$ for $\varepsilon > 0$

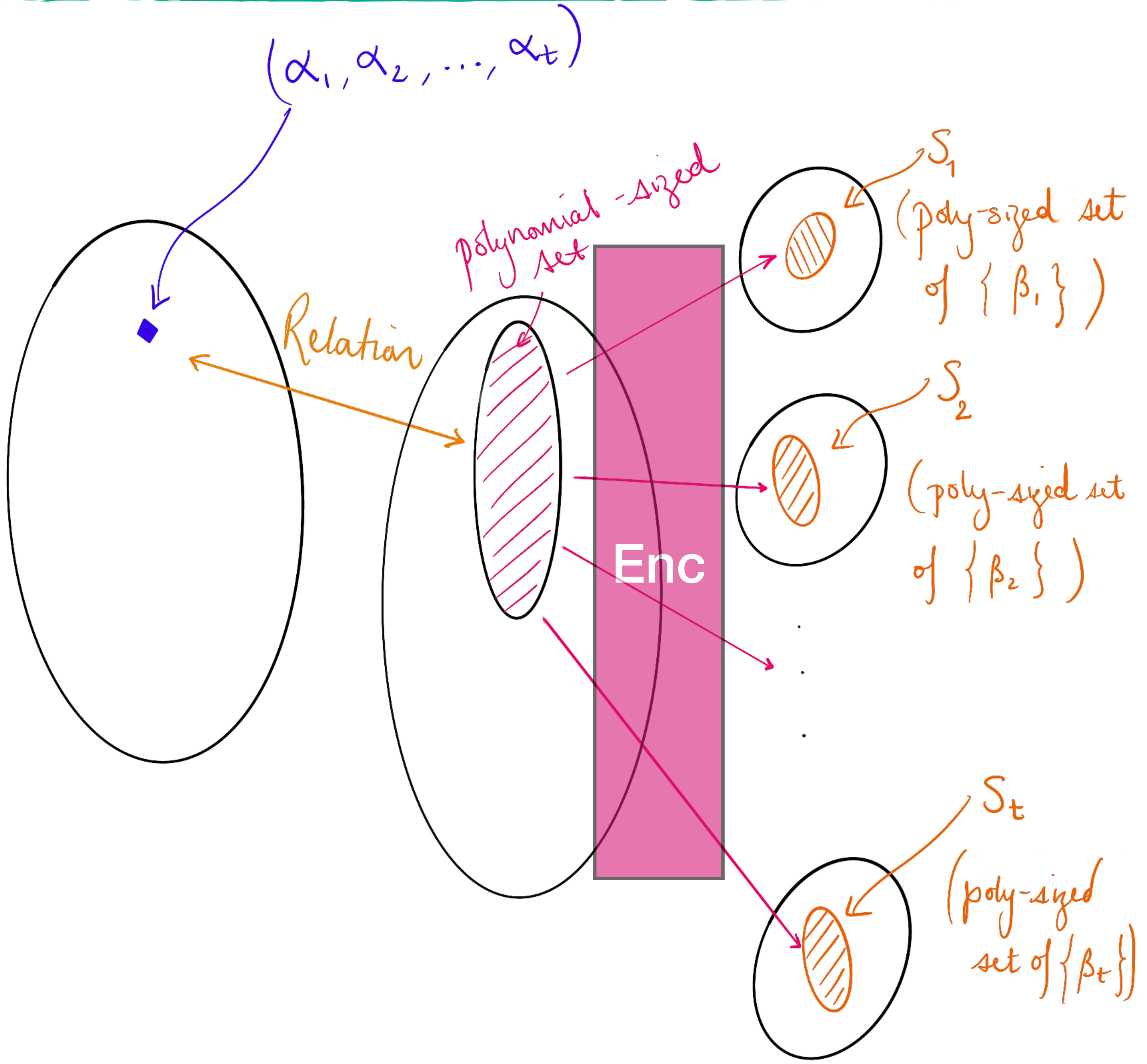
Fiat-Shamir from Coding Theory [HLR21]

Parallel repetition gives a bad challenge set with a nice combinatorial structure.

For a statement $x \notin L$:

$$R_x = \left\{ \left((\alpha_1, \ldots, \alpha_t), r\right) : (\text{Encode}(r))_i \in S_i \right\}$$

General list-recovery addresses product sets $S_1 \times S_2 \times \cdots \times S_t$ where each S_i may differ.



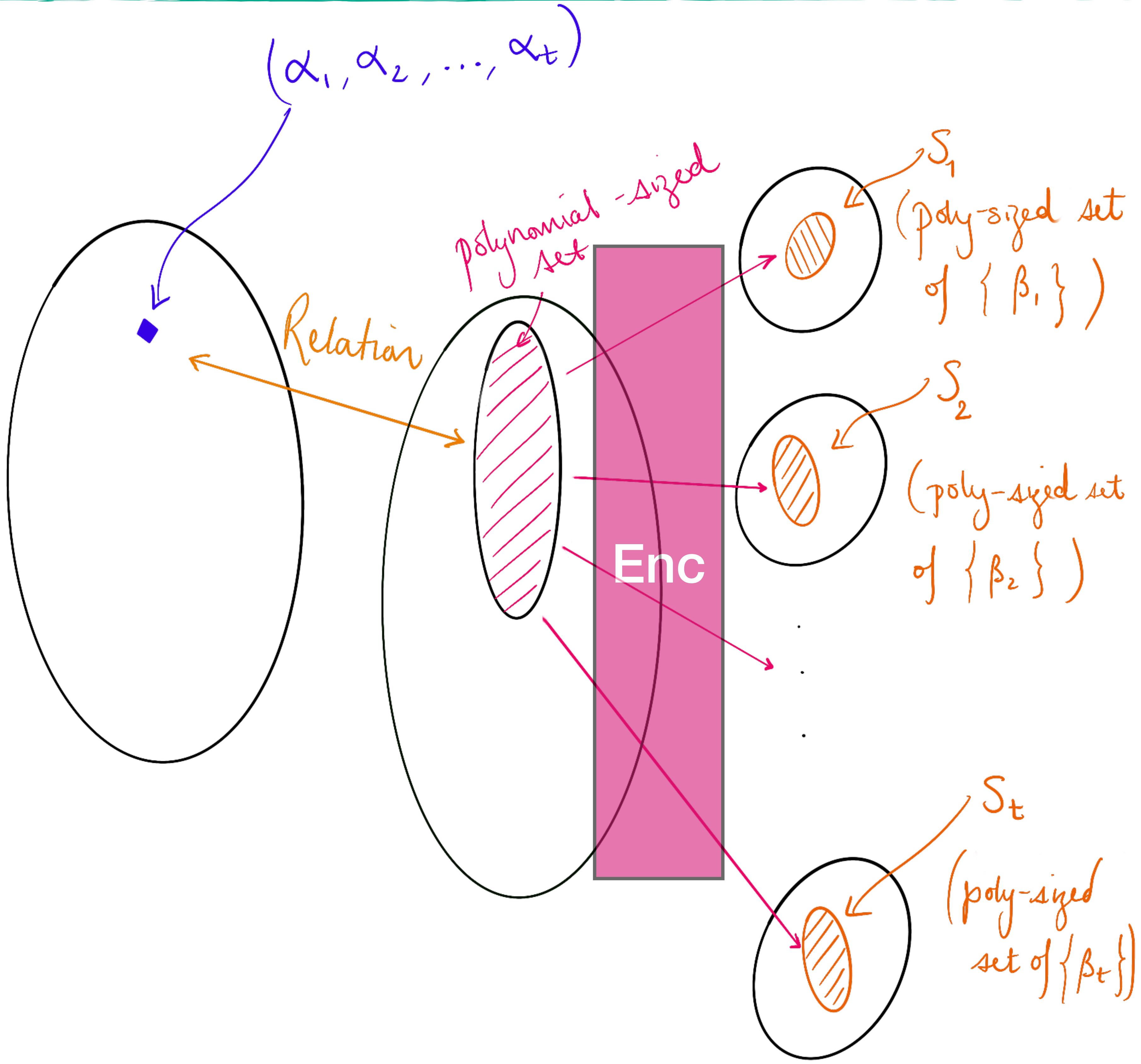
Fiat-Shamir from Coding Theory [HLR21]

Parallel repetition gives a bad challenge set with a nice combinatorial structure.

For a statement $x \notin L$:

$$R_x = \left\{ \big((\alpha_1, \ldots, \alpha_t), r\big) : (\mathsf{Encode}(r))_i \in S_i \right\}$$

Is general list-recoverability necessary for the setting of MPC-in-the-Head?



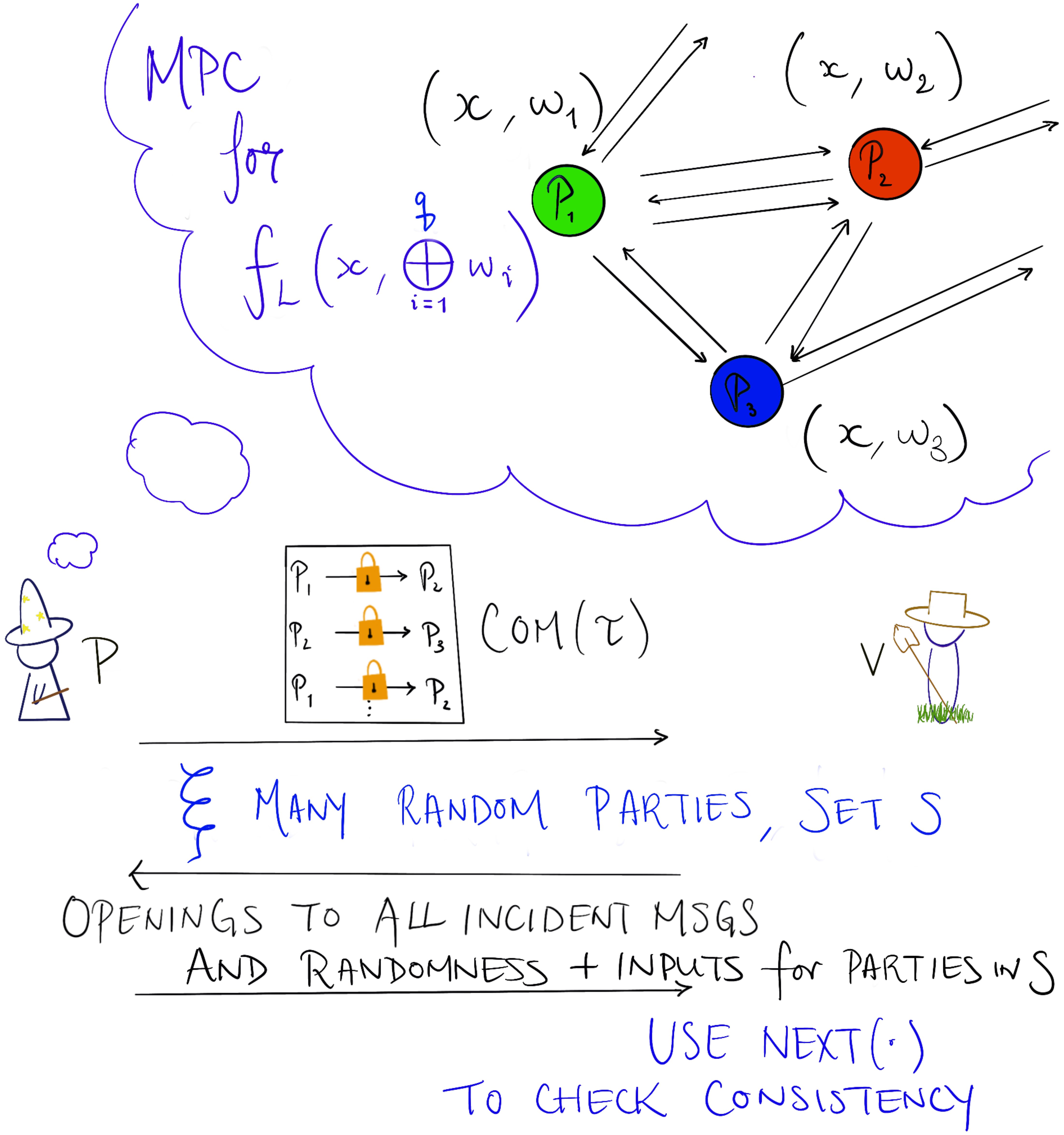
Bad Challenge Structure of MPC-in-the-Head

Bad Challenge Set:

$$S_{Com(\tau)} \times \cdots \times S_{Com(\tau)}$$

$$S_{Com(\tau)} = \{ i : \text{View}_i \text{ consistent } \} \subset \mathbb{Z}_q$$

For our MPC-in-the-head protocol, we have a product sets $S \times S \times \cdots \times S$ for a single set S, a much simpler structure.



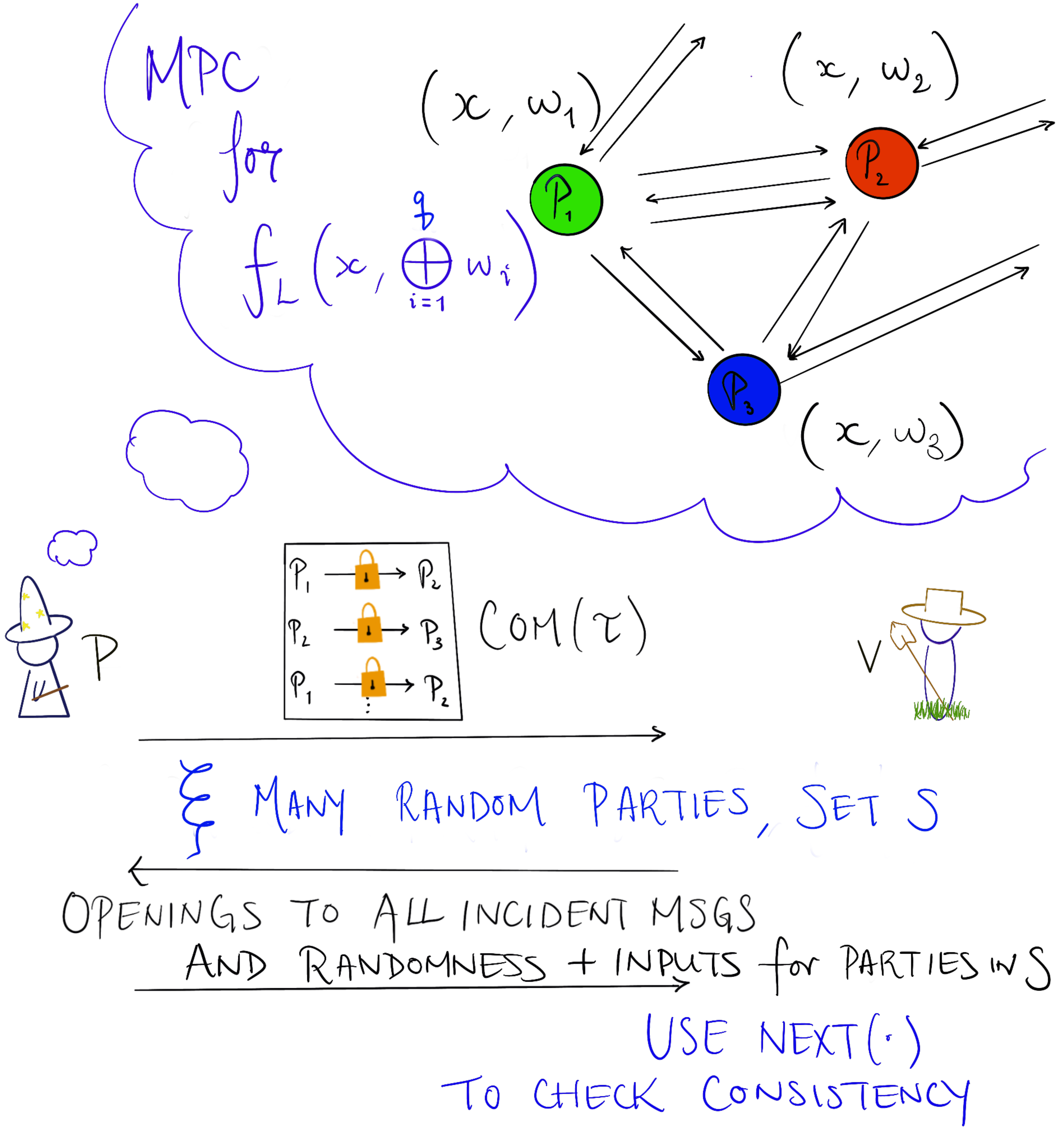
Bad Challenge Structure of MPC-in-the-Head

Bad Challenge Set:

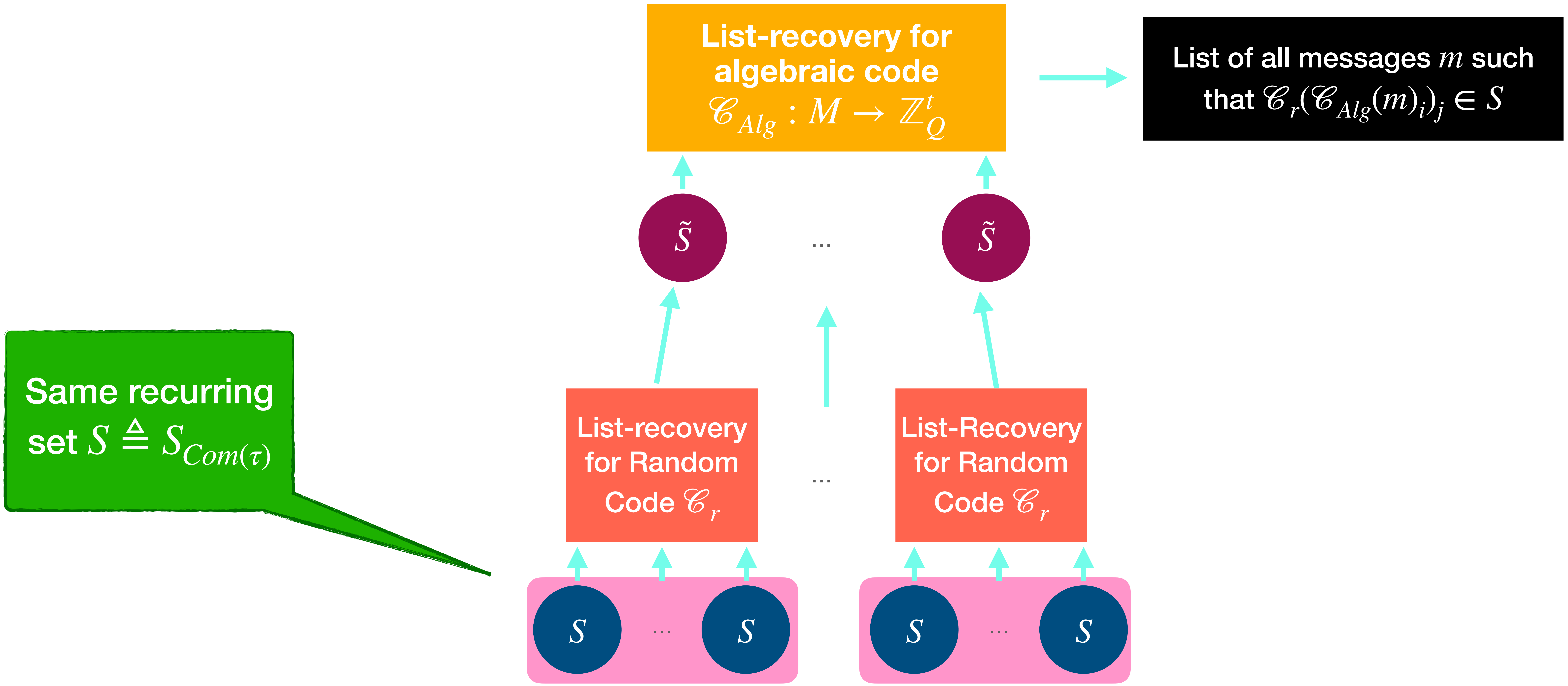
$$S_{Com(\tau)} \times \cdots \times S_{Com(\tau)}$$

$$S_{Com(\tau)} = \{ i : \text{View}_i \text{ consistent} \} \subset \mathbb{Z}_q$$

Does this simpler bad challenge structure allow the usage of a derandomization technique both *simpler* and *more efficient* than general list-recoverability?

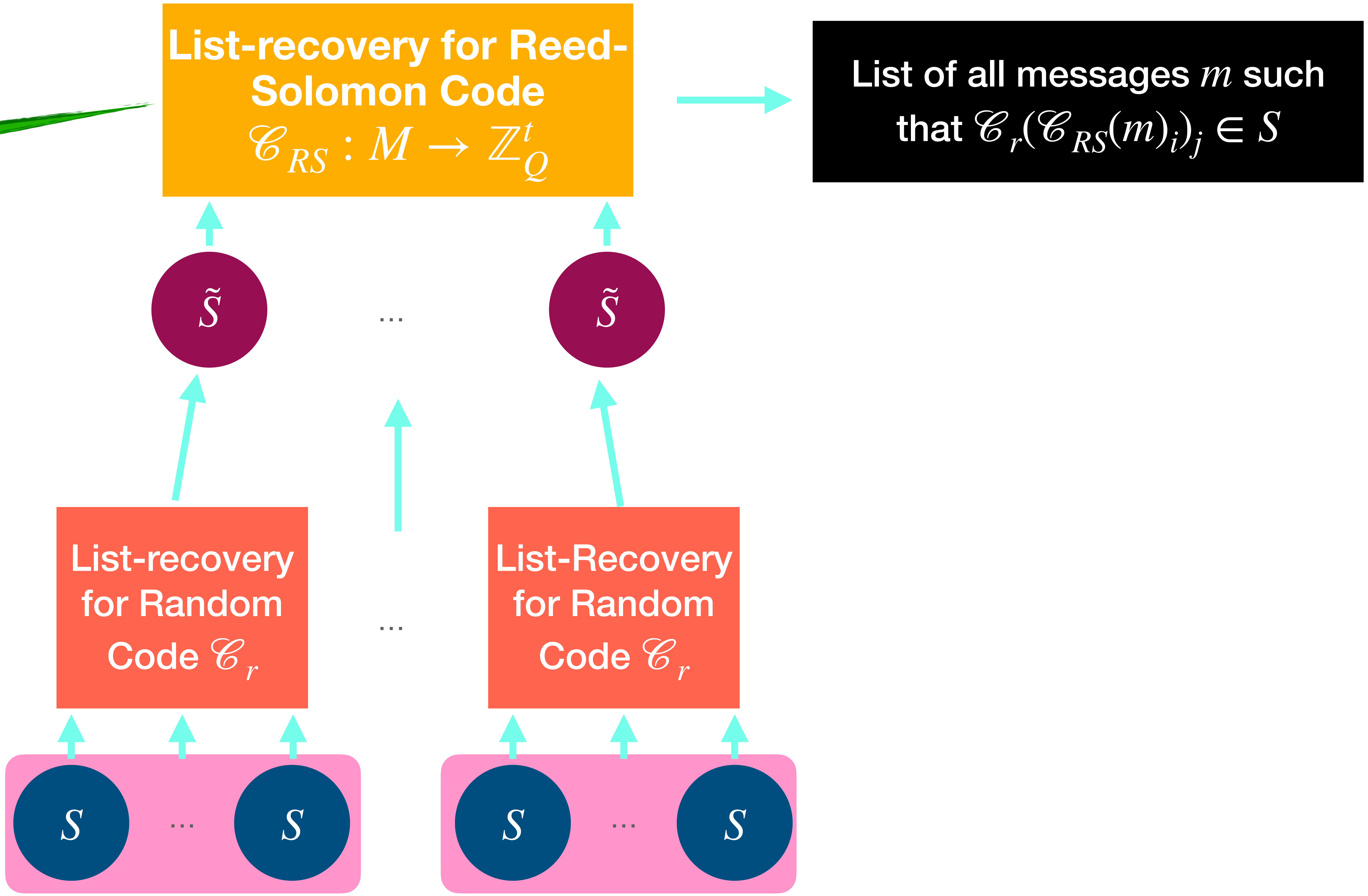


Recurrent List-Recovery



Recurrent List-Recovery

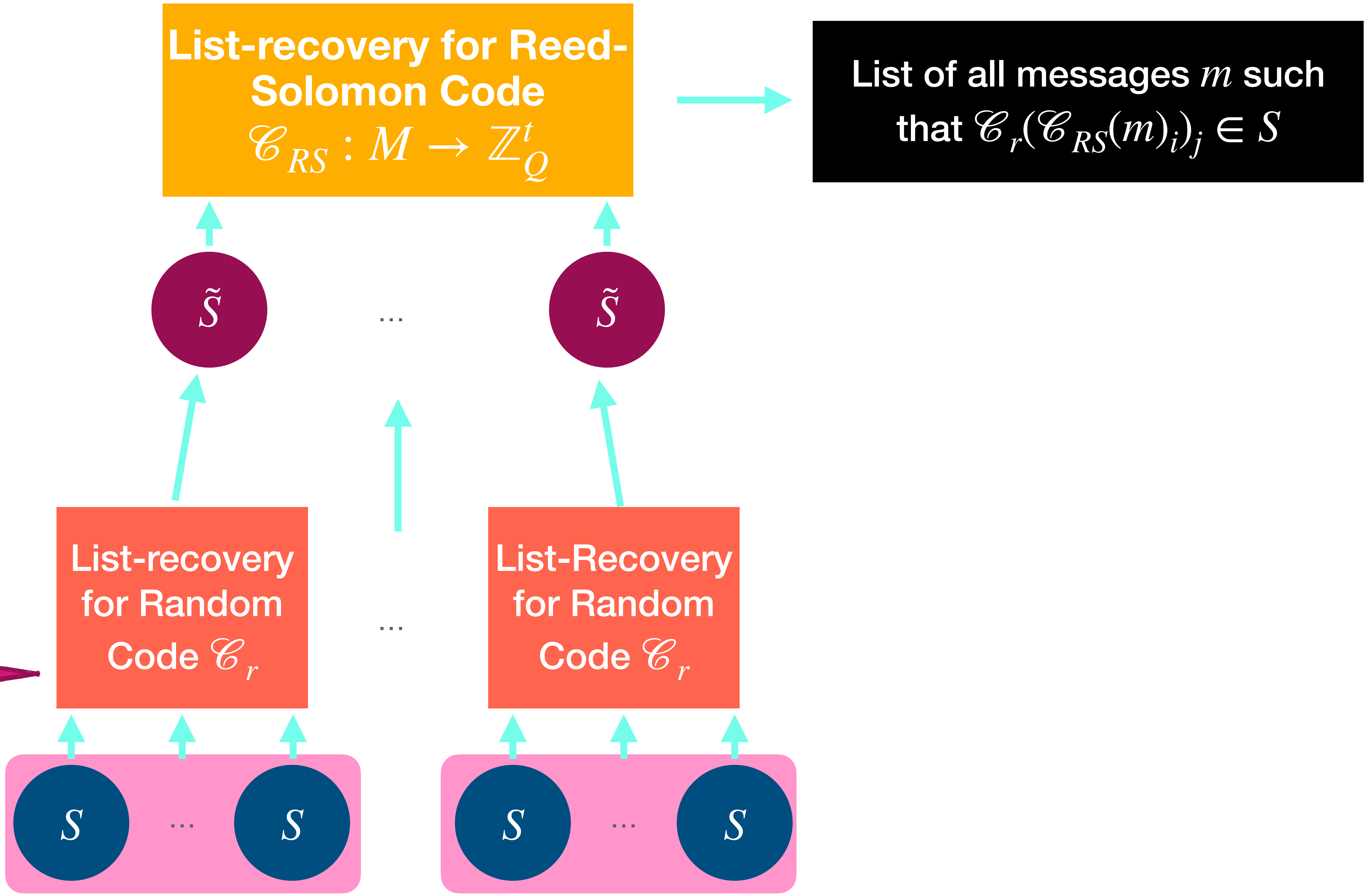
Let's try to use a simple algebraic code to instantiate recurrent list-recovery!



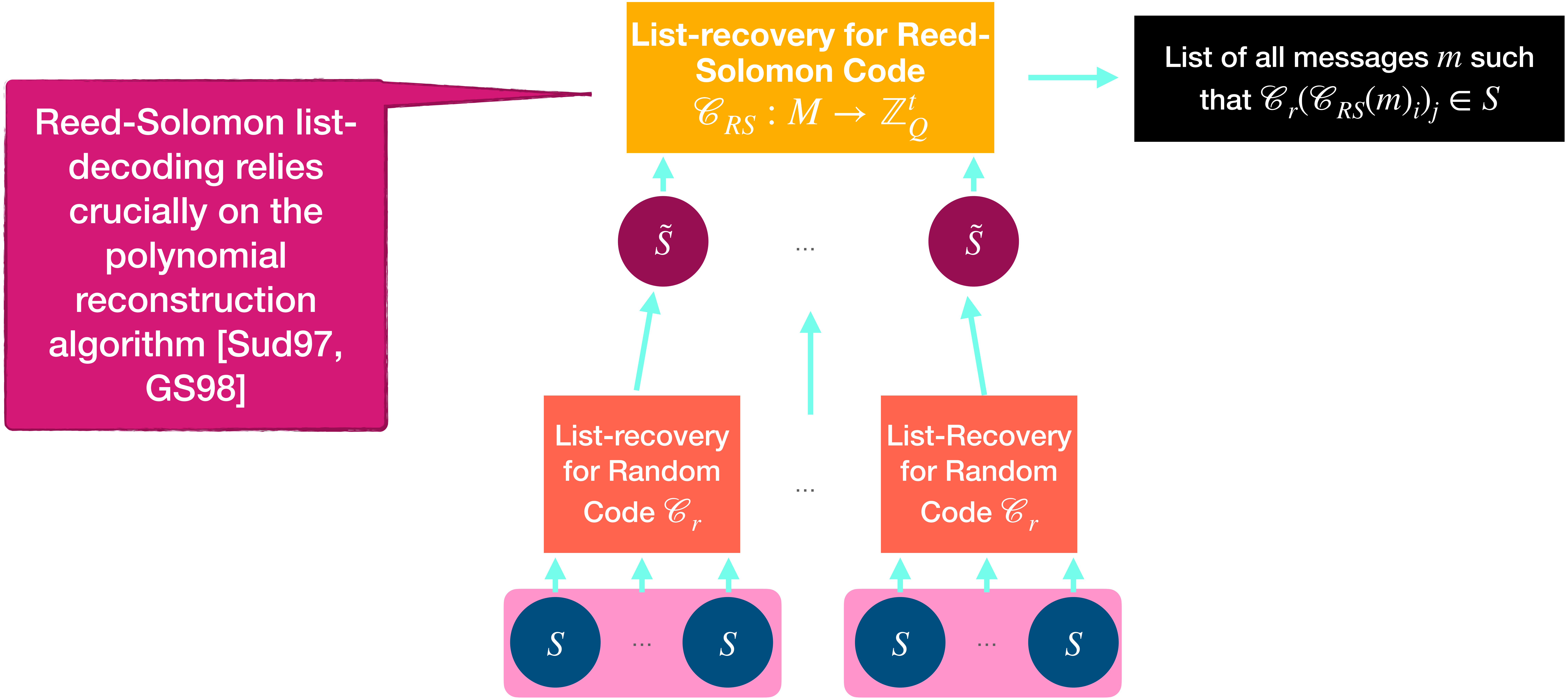
Recurrent List-Recovery

List-recovery for a *single* random code $\mathscr{C}_r$ may result in an output set $\tilde{S}$ that is too large for RS list-recovery!

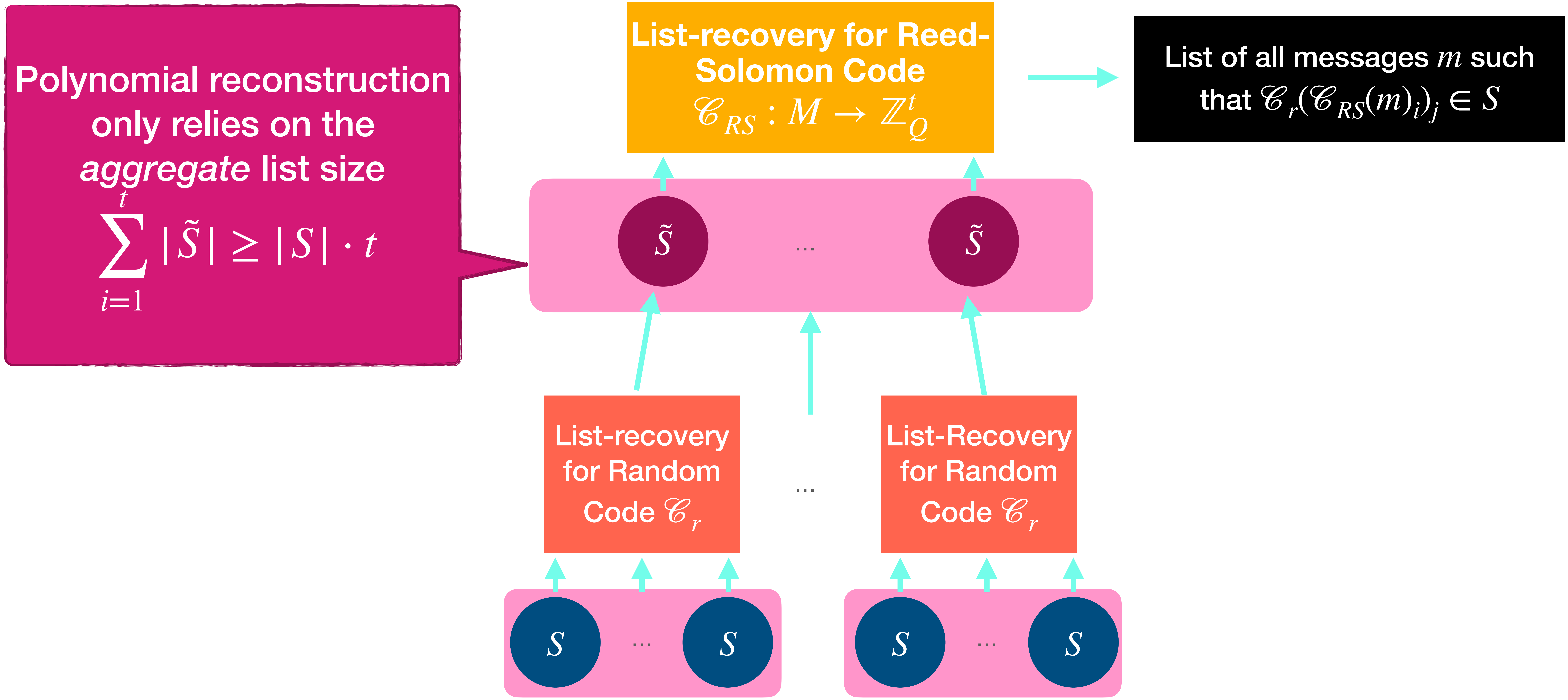
For a fixed random code, this happens with non-negligible probability over Prover's choice of S.



Recurrent List-Recovery

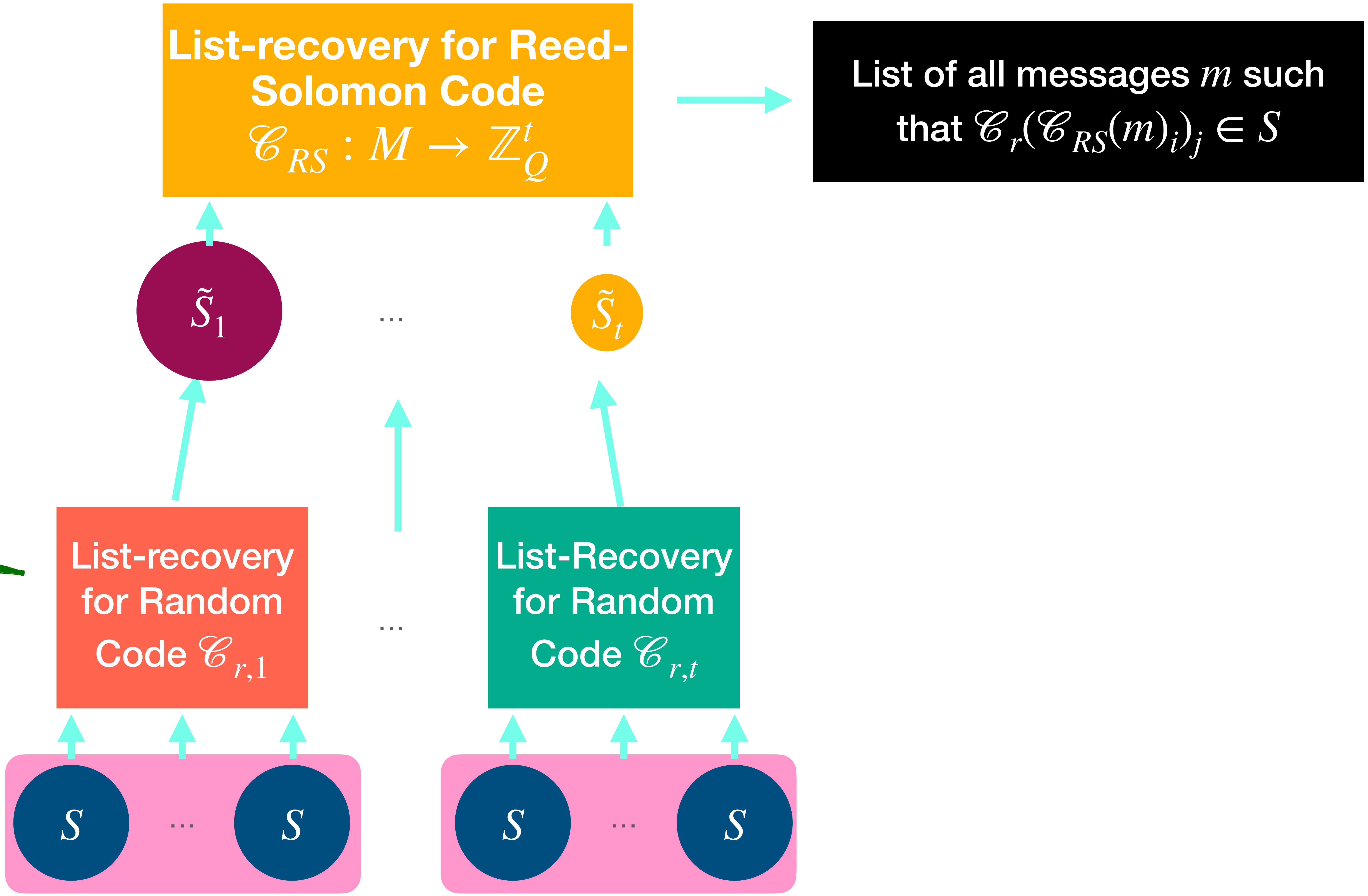


Recurrent List-Recovery



Aggregate Size Analysis

If we use *multiple* random codes, then while some output sets may be large, others may be small.

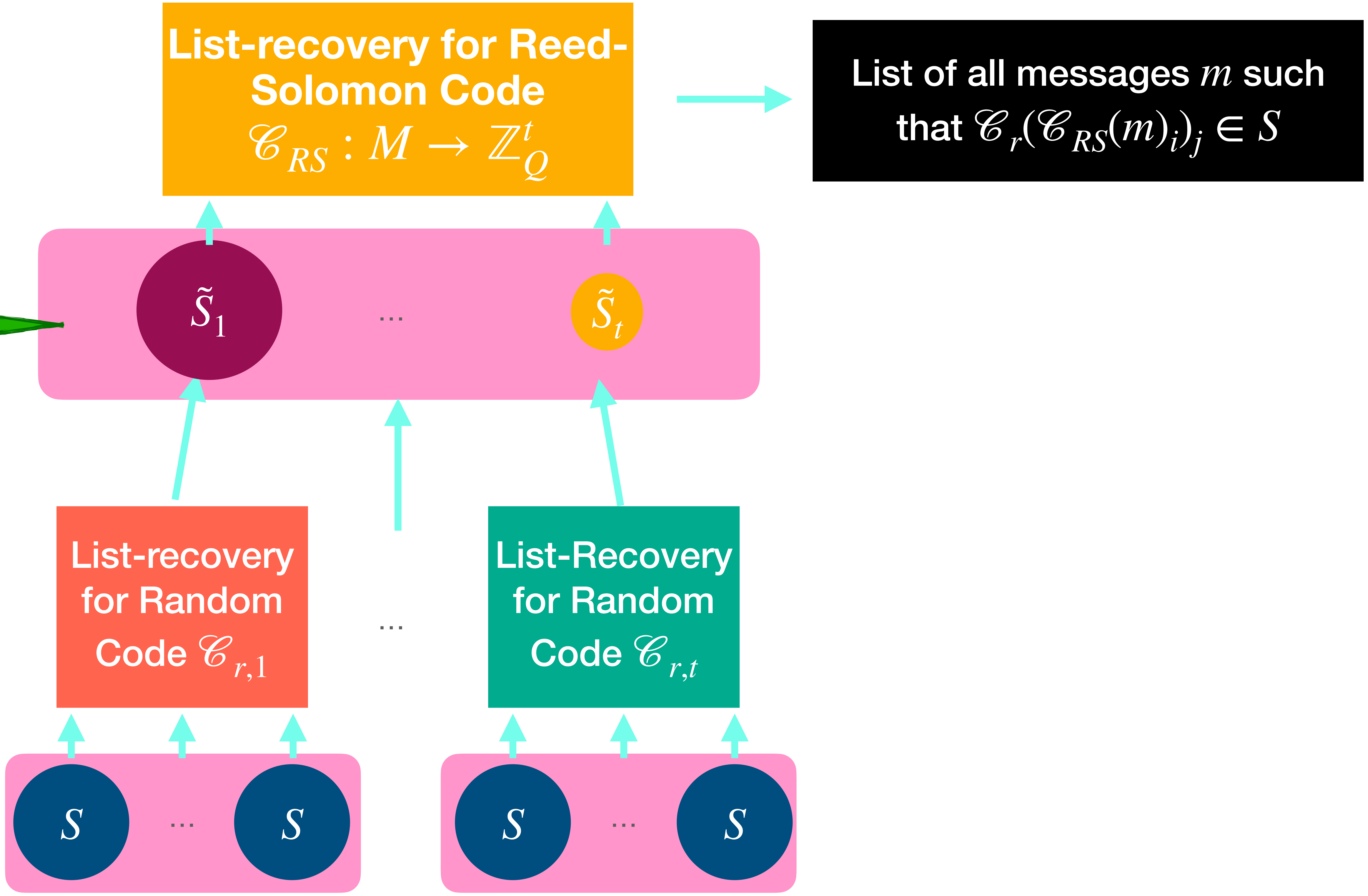


Aggregate Size Analysis

For $|S| = \alpha \cdot q$ for $\alpha \in (0,1)$, $q = \tilde{O}(k)$ we achieve

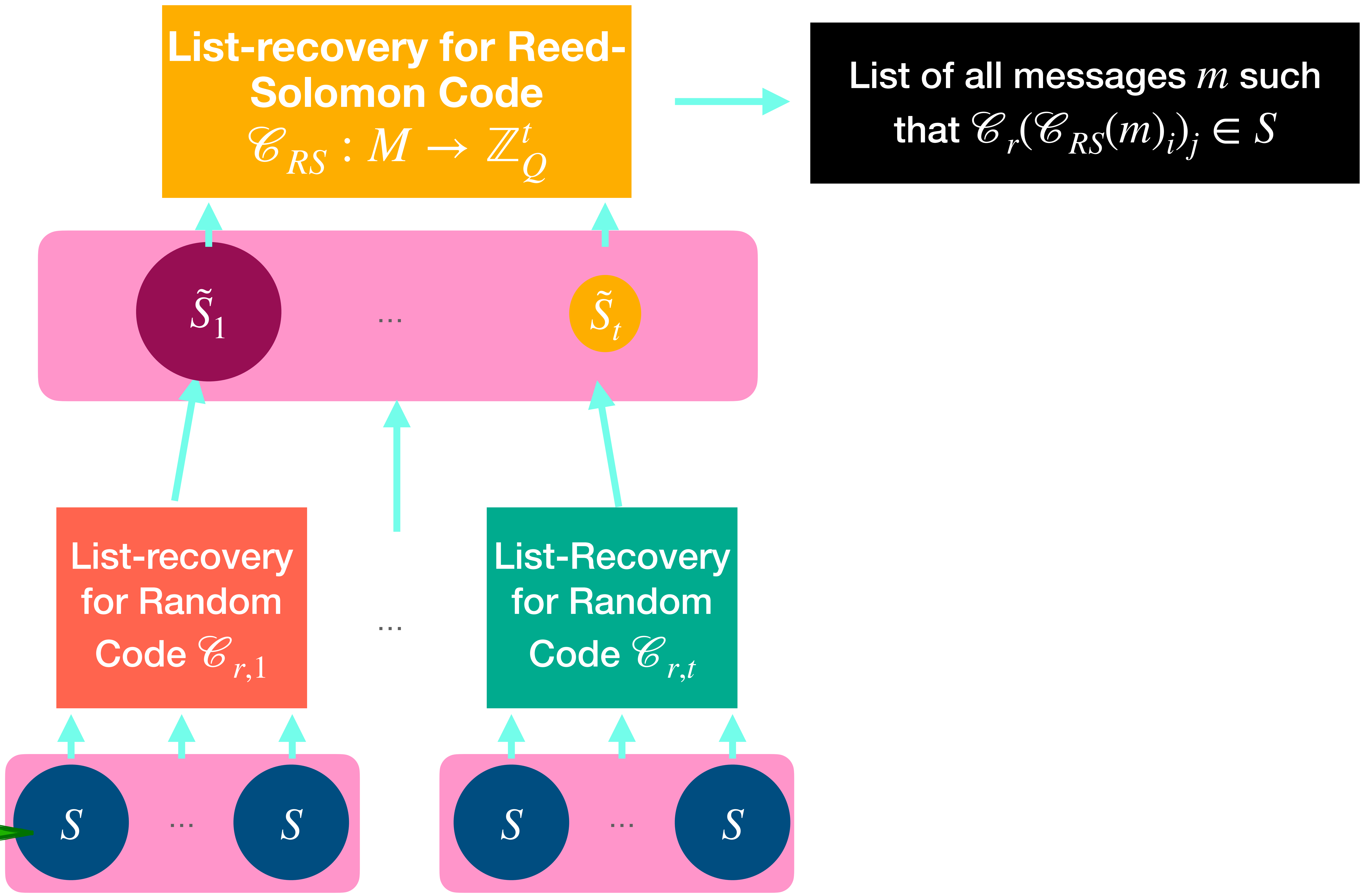
$$\sum |\tilde{S}_i| \leq \tilde{O}\left(|S| \right)$$

with all but negligible probability.



Aggregate Size Analysis

Polynomial reconstruction succeeds for every choice of the set S (of the appropriate size) with all but negligible probability.



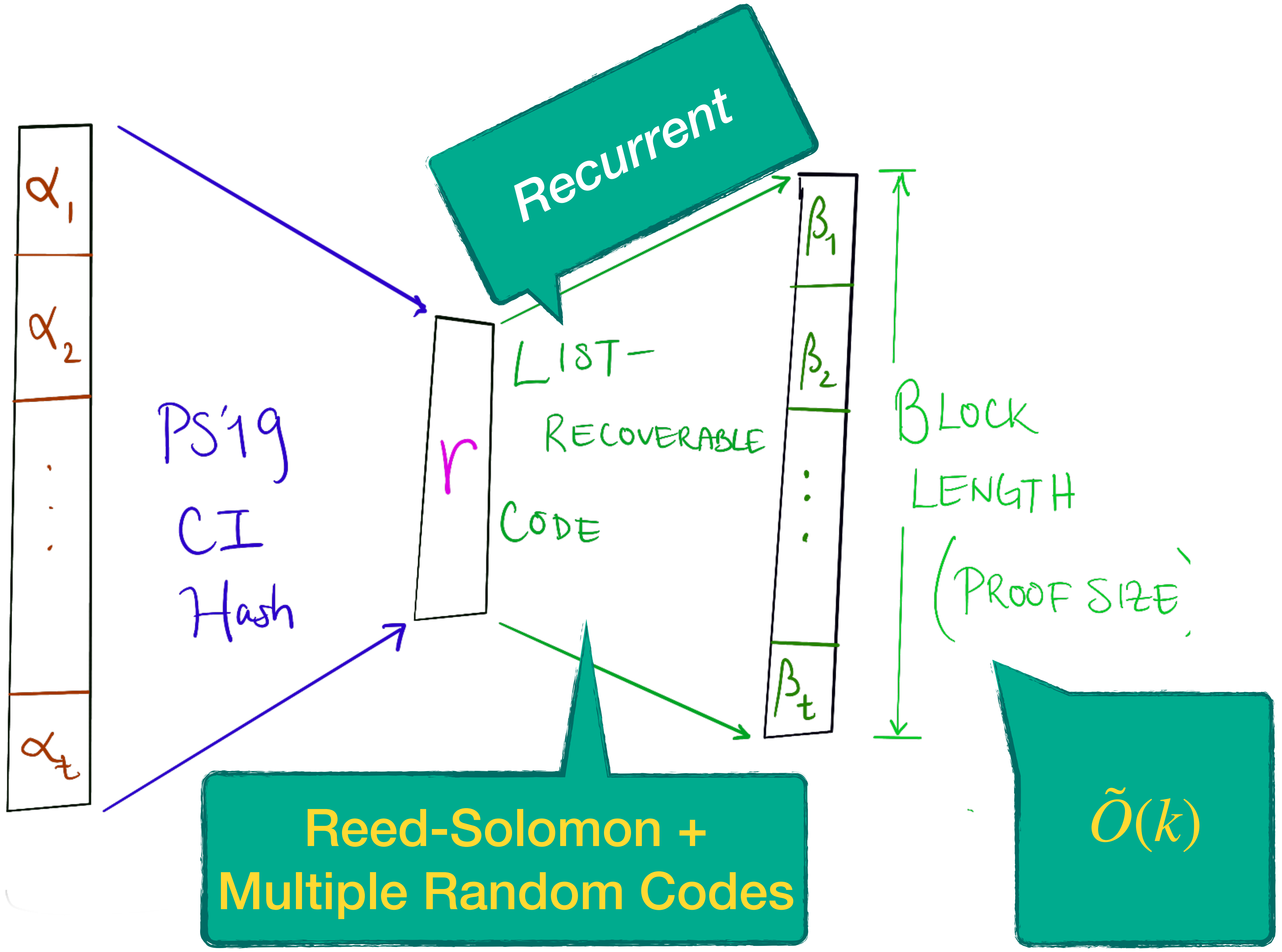
Summary:

We modify the MPC-in-the-head protocol [IKOS07] so that it has a bad challenge set amenable to ***recurrent list-recovery***. We instantiate the code with a Reed-Solomon code concatenated with multiple random codes, and use aggregate size analysis to obtain a quasi-linear block length!

For a statement $x \notin L$:

$$R_x = \left\{ \left((\alpha_1, \ldots, \alpha_t), (\beta_1, \ldots, \beta_t) \right) : \exists (\gamma_1 \ldots, \gamma_t) \text{ s.t. } V(x, \vec{\alpha}, \vec{\beta}, \vec{\gamma}) = 1 \right\}$$

This is still a CI hash for the desired relation.



Thank you!

Appendix

Reed-Solomon Codes + Polynomial Reconstruction

Def [RS60]: A Reed-Solomon code $\mathscr{C}_\lambda \colon \mathbb{Z}_Q^{k+1} \to \mathbb{Z}_Q^t$ is parameterized by a base field size $Q = Q(\lambda)$, a degree $k = k(\lambda)$, a block length $t = t(\lambda)$, and a set of values $A_\lambda = \{\alpha_1, \ldots, \alpha_t\}$. $\mathscr{C}_\lambda$ takes as input a polynomial p of degree k over $\mathbb{Z}_Q$, represented by its $k+1$ coefficients, and outputs the vector of evaluations $\big(p(\alpha_1), \ldots, p(\alpha_t)\big)$ of p on each of the points α_i.

Reed-Solomon Codes + Polynomial Reconstruction

Def [RS60]: A Reed-Solomon code $\mathscr{C}_\lambda : \mathbb{Z}_Q^{k+1} \to \mathbb{Z}_Q^t$ is parameterized by a base field size $Q = Q(\lambda)$, a degree $k = k(\lambda)$, a block length $t = t(\lambda)$, and a set of values $A_\lambda = \{\alpha_1, \ldots, \alpha_t\}$. $\mathscr{C}_\lambda$ takes as input a polynomial p of degree k over $\mathbb{Z}_Q$, represented by its $k+1$ coefficients, and outputs the vector of evaluations $\big(p(\alpha_1), \ldots, p(\alpha_t)\big)$ of p on each of the points α_i.

Polynomial Reconstruction:

- **INPUT:** Integers k_p, n_p. Distinct pairs $\{(\alpha_i, y_i)\}_{i \in [n_p]}$, where $\alpha_i, y_i \in \mathbb{Z}_Q$.
- **OUTPUT:** A list of all polynomials $p(X) \in \mathbb{Z}_Q[X]$ of degree at most k_p, which satisfy $p(\alpha_i) = y_i, \forall\ i \in [n_p]$.