WHIR 

Proximity testing for Reed–Solomon+

Gal Arnon

WEIZMANN INSTITUTE OF SCIENCE

Giacomo Fenzi

EPFL

Alessandro Chiesa

EPFL

Eylon Yogev

ePrint 2024/1586

Motivation

SNARKs

Succinct Non-interactive Arguments of Knowledge

SNARKs

Succinct Non-interactive Arguments of Knowledge

Want to show “knowledge” of w s.t. $(x, w) \in R$

SNARKs

Succinct Non-interactive Arguments of Knowledge

Want to show “knowledge” of w s.t. $(x, w) \in R$

e.g. $R := \{(x, w) : \text{SHA3}(w) = x\}$

SNARKs

Succinct Non-interactive Arguments of Knowledge

Want to show “knowledge” of w s.t. $(x, w) \in R$

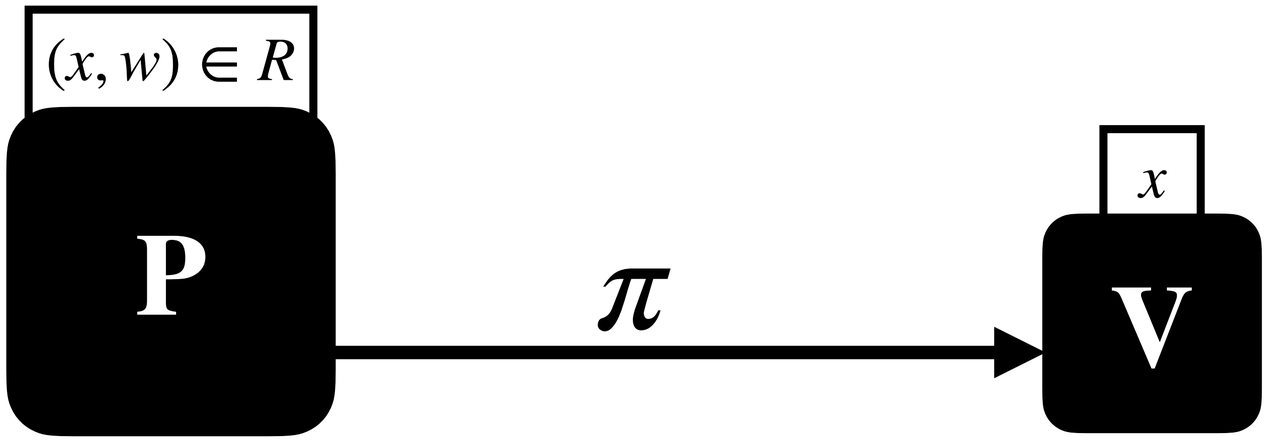
e.g. $R := \{(x, w) : \text{SHA3}(w) = x\}$

SNARKs

Succinct Non-interactive Arguments of Knowledge

Want to show “knowledge” of w s.t. $(x, w) \in R$

e.g. $R := \{(x, w) : \text{SHA3}(w) = x\}$

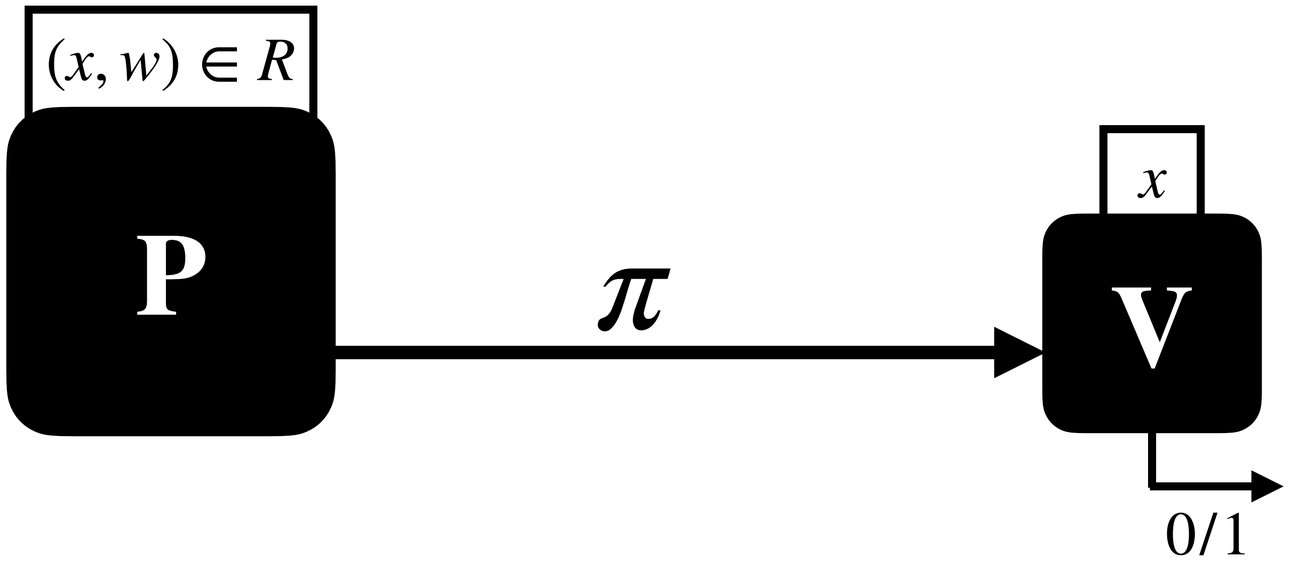


SNARKs

Succinct Non-interactive Arguments of Knowledge

Want to show “knowledge” of w s.t. $(x, w) \in R$

e.g. $R := \{(x, w) : \text{SHA3}(w) = x\}$

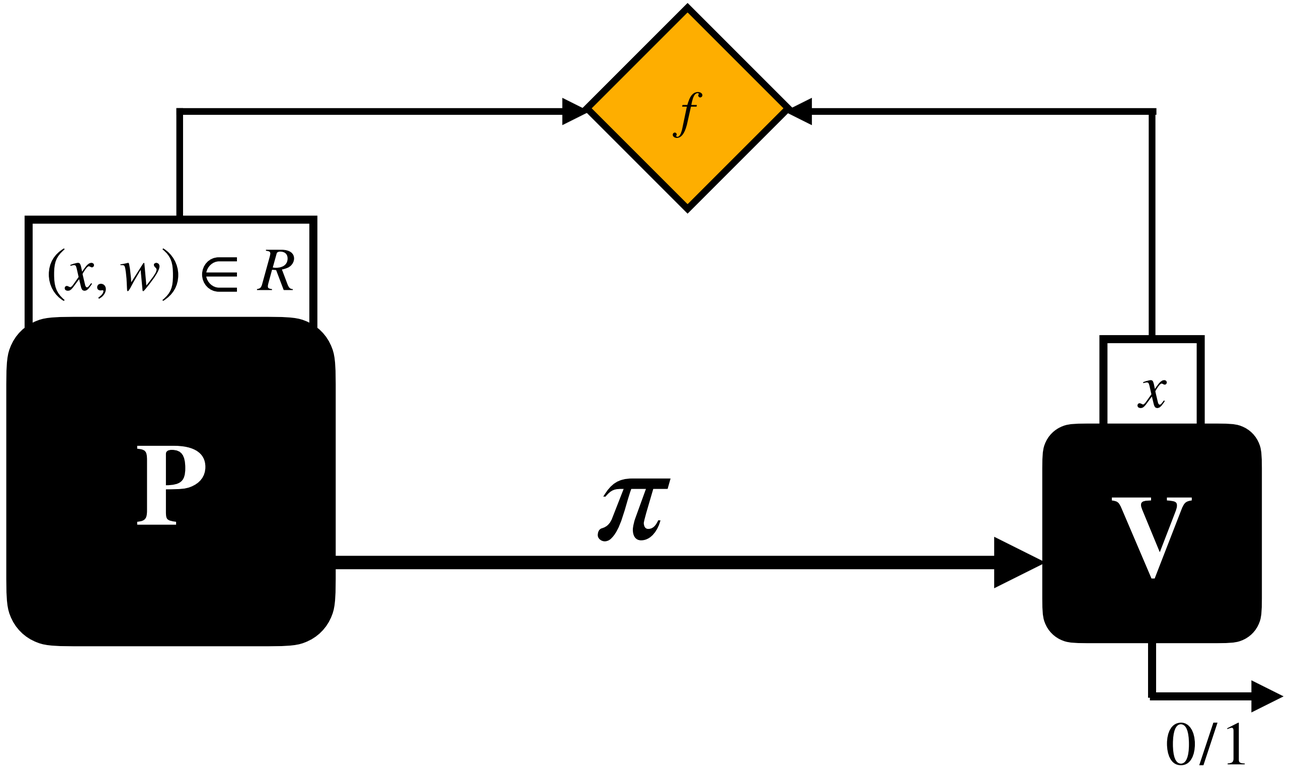


SNARKs

Succinct Non-interactive Arguments of Knowledge

Want to show "knowledge" of w s.t. $(x, w) \in R$

e.g. $R := \{(x, w) : \text{SHA3}(w) = x\}$



Today: we focus on

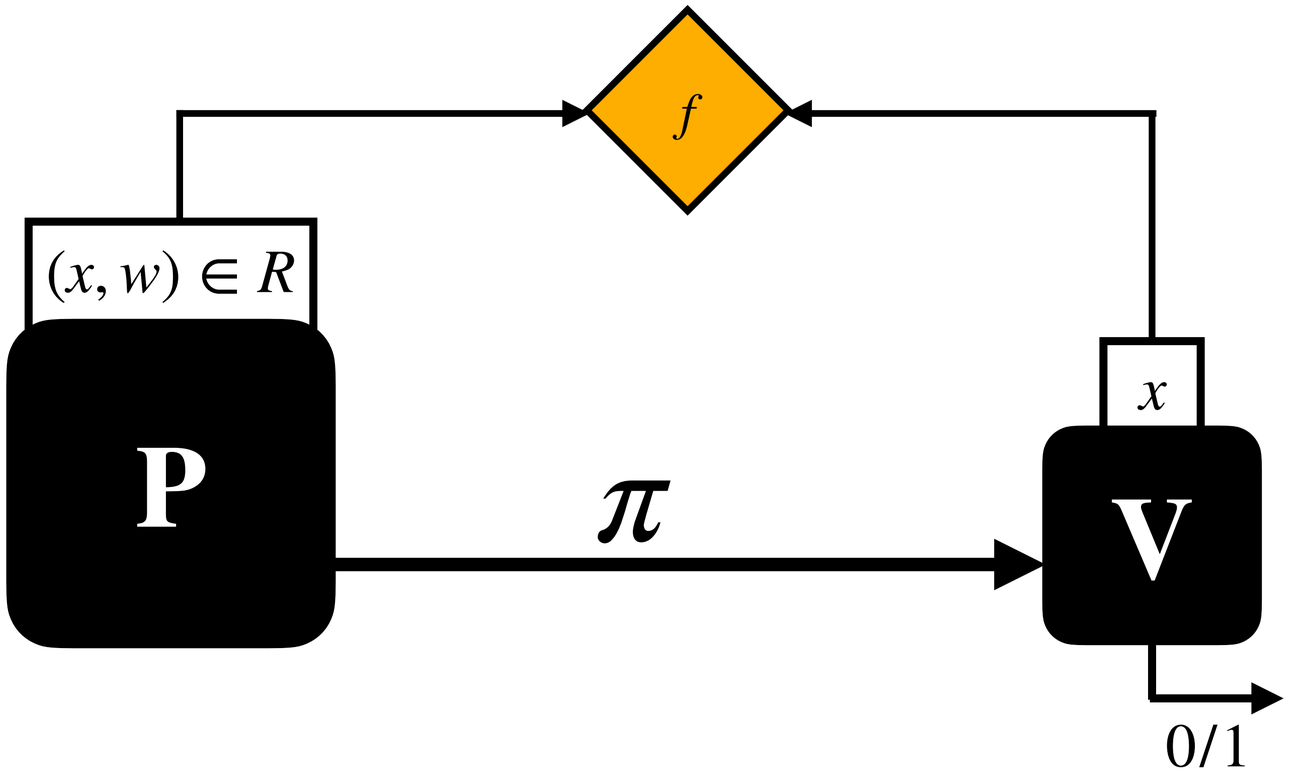
hash-based SNARKs

SNARKs

Succinct Non-interactive Arguments of Knowledge

Want to show "knowledge" of w s.t. $(x, w) \in R$

e.g. $R := \{(x, w) : \text{SHA3}(w) = x\}$



Today: we focus on

hash-based SNARKs

i.e. sound in the **pure** random oracle model.

Hash-based SNARKs

In practice

Hash-based SNARKs

In practice

Instantiating random oracle gives amazing SNARKs:

Hash-based SNARKs

In practice

Instantiating random oracle gives amazing SNARKs:

- Transparent setup (choice of hash)

Hash-based SNARKs

In practice

Instantiating random oracle gives amazing SNARKs:

- Transparent setup (choice of hash)
- Highly efficient implementations (no public-key crypto)

Hash-based SNARKs

In practice

Instantiating random oracle gives amazing SNARKs:

- Transparent setup (choice of hash)
- Highly efficient implementations (no public-key crypto)
- Plausibly post-quantum secure (secure in QROM)

Hash-based SNARKs

In practice

Instantiating random oracle gives amazing SNARKs:

- Transparent setup (choice of hash)
- Highly efficient implementations (no public-key crypto)
- Plausibly post-quantum secure (secure in QROM)

Used to secure billions of dollars in real-world blockchains:

Hash-based SNARKs

In practice

Instantiating random oracle gives amazing SNARKs:

- Transparent setup (choice of hash)
- Highly efficient implementations (no public-key crypto)
- Plausibly post-quantum secure (secure in QROM)

Used to secure billions of dollars in real-world blockchains:

STARKWARE

polygon

Irreducible

LAMBDA CLASS

zkSync

dYdX

Matter Labs

RISC ZERO

Succinct

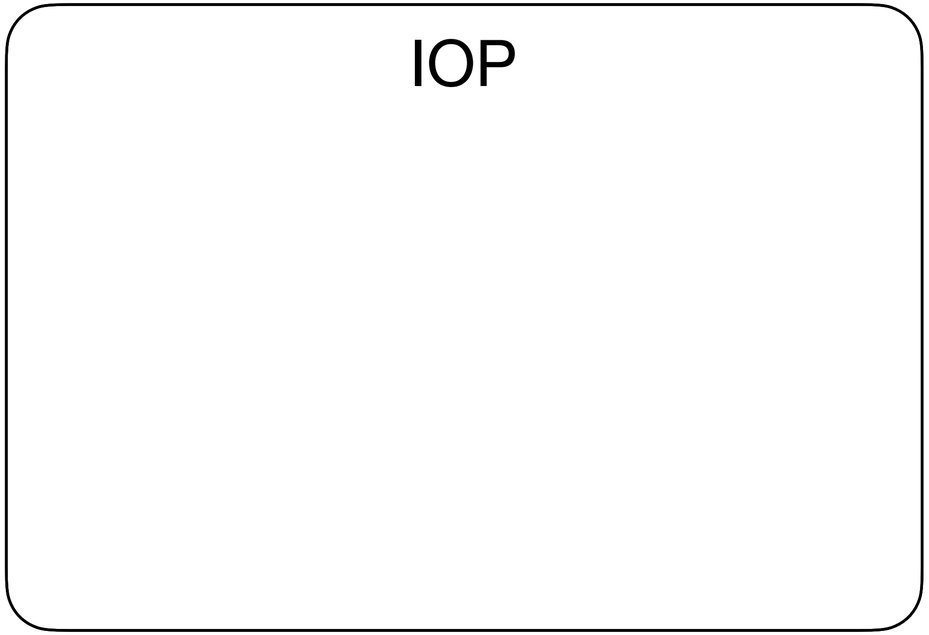
And many more…

Constructing SNARKs

[BCS16] Construction

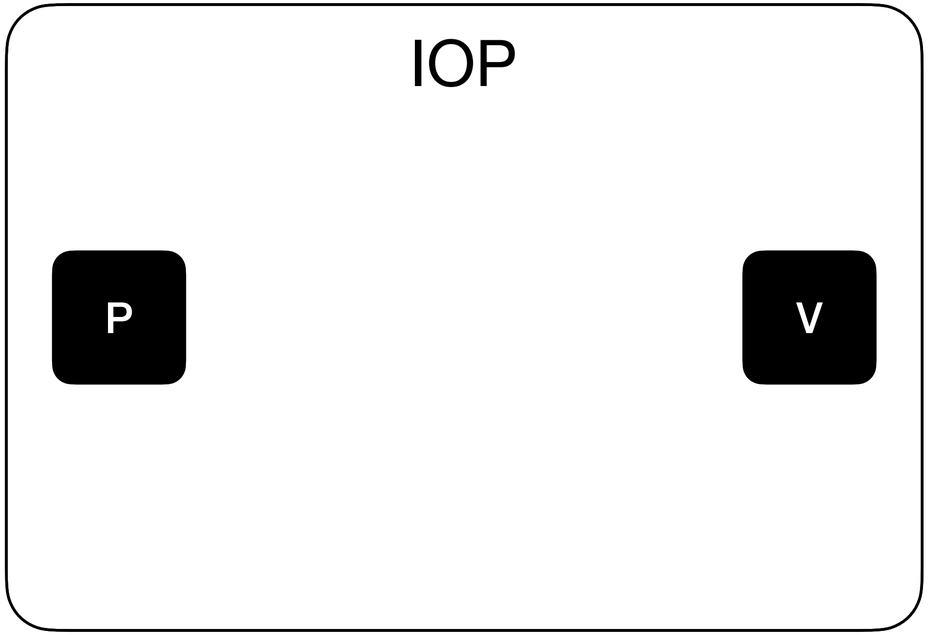
Constructing SNARKs

[BCS16] Construction



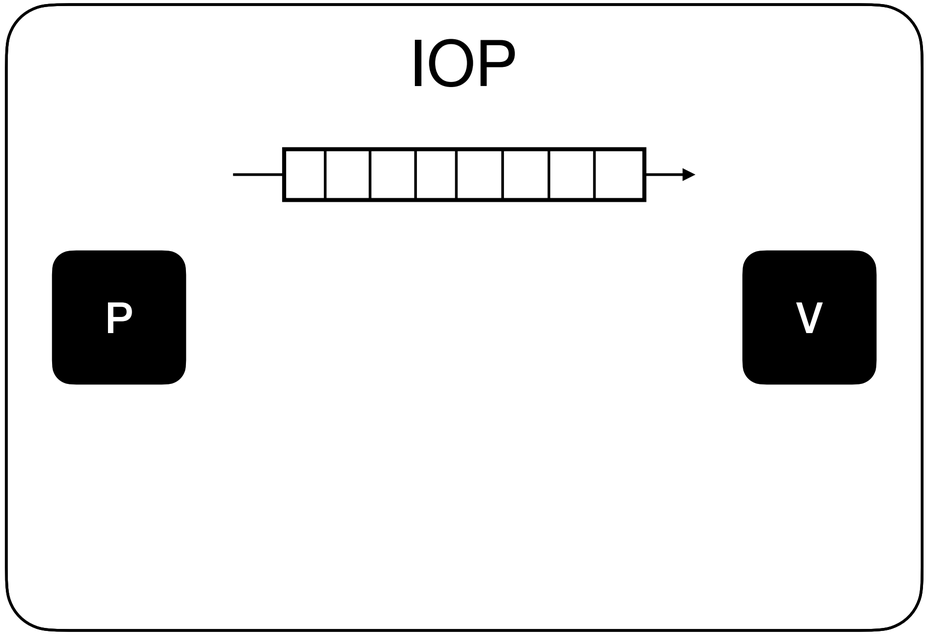
Constructing SNARKs

[BCS16] Construction



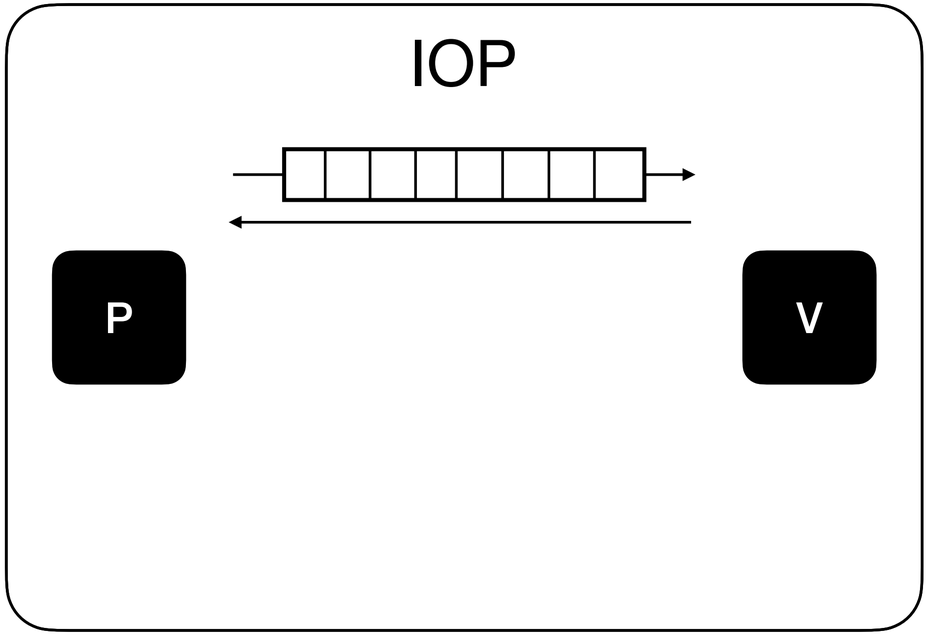
Constructing SNARKs

[BCS16] Construction



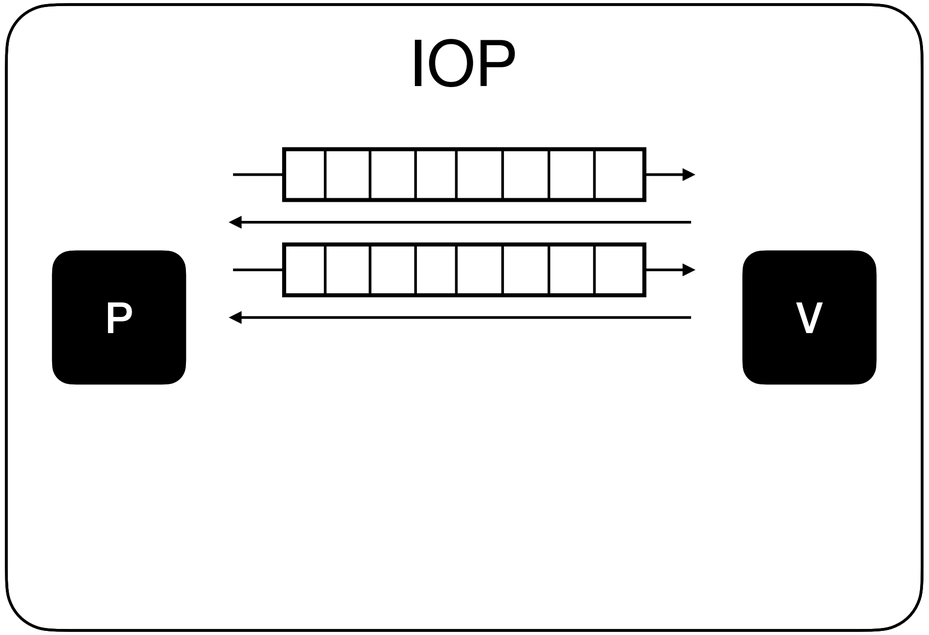
Constructing SNARKs

[BCS16] Construction



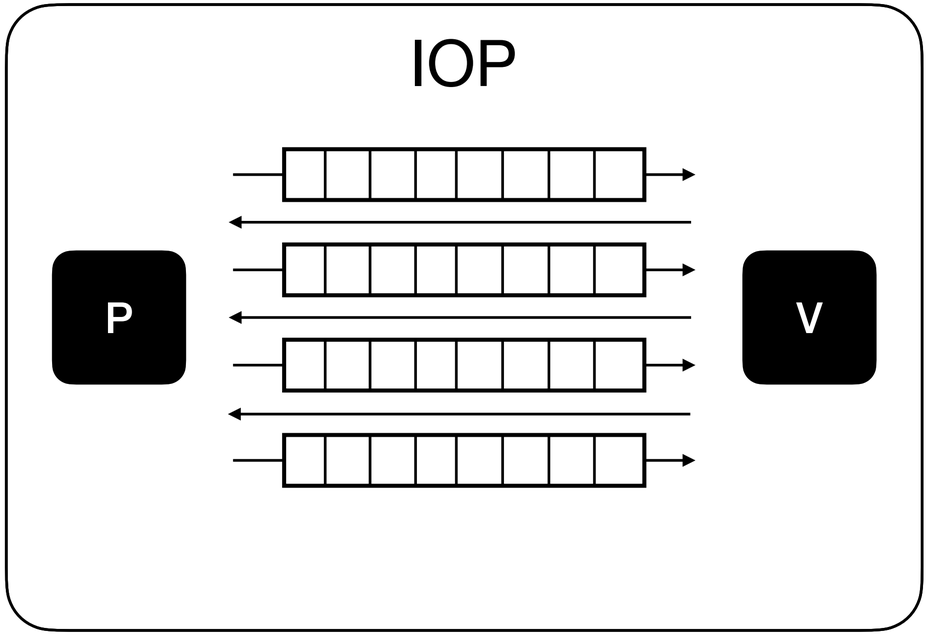
Constructing SNARKs

[BCS16] Construction



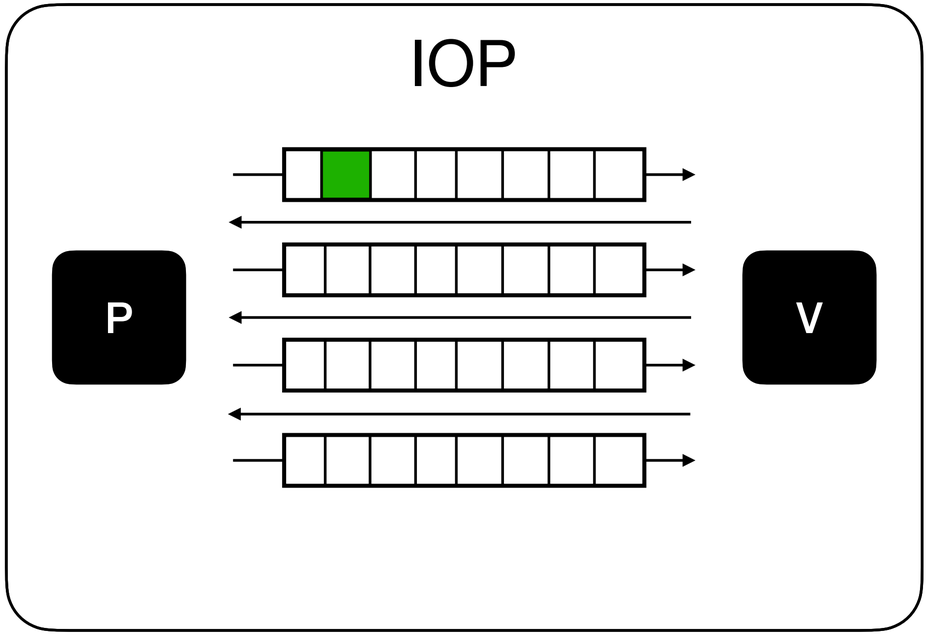
Constructing SNARKs

[BCS16] Construction



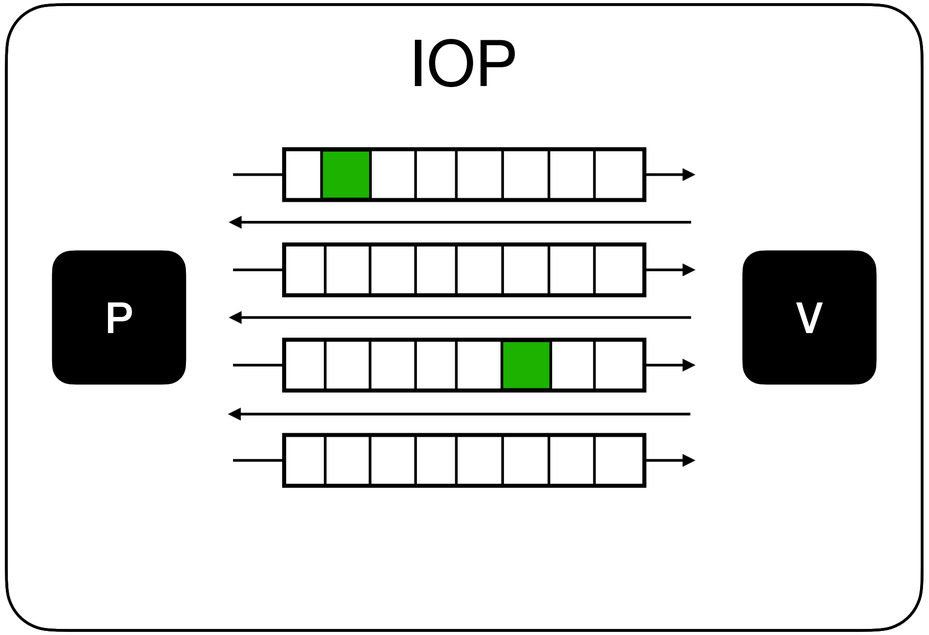
Constructing SNARKs

[BCS16] Construction



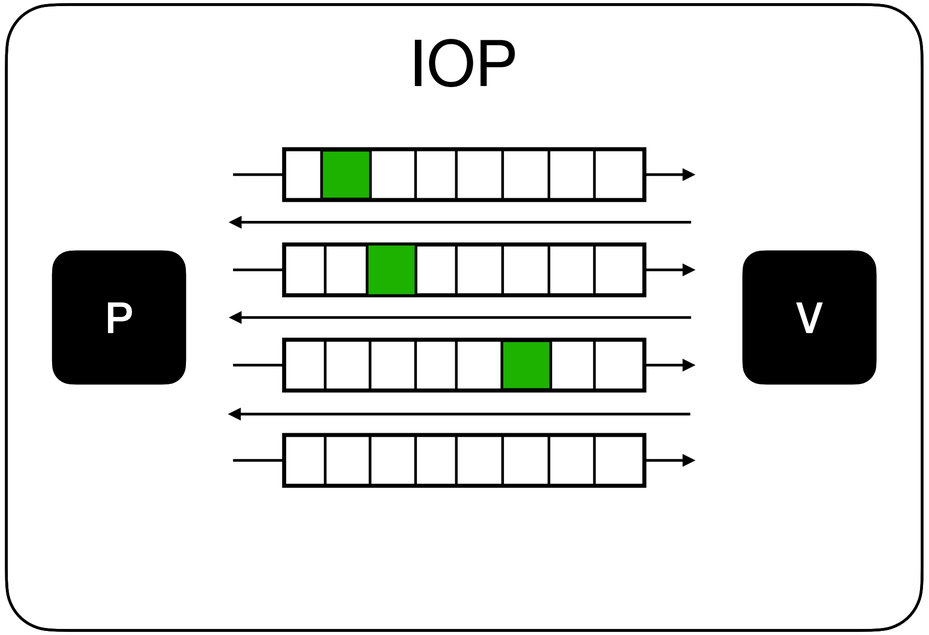
Constructing SNARKs

[BCS16] Construction



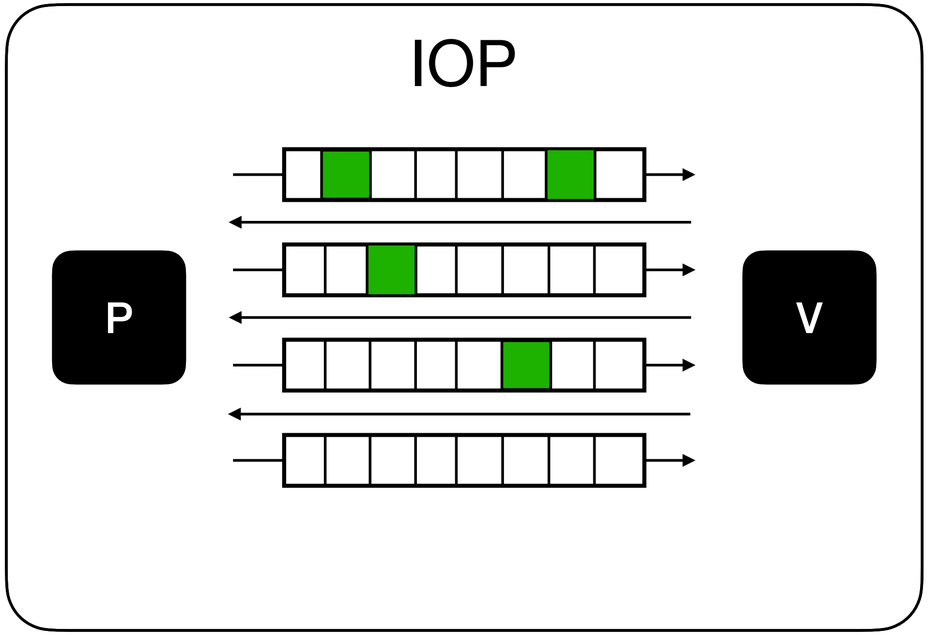
Constructing SNARKs

[BCS16] Construction



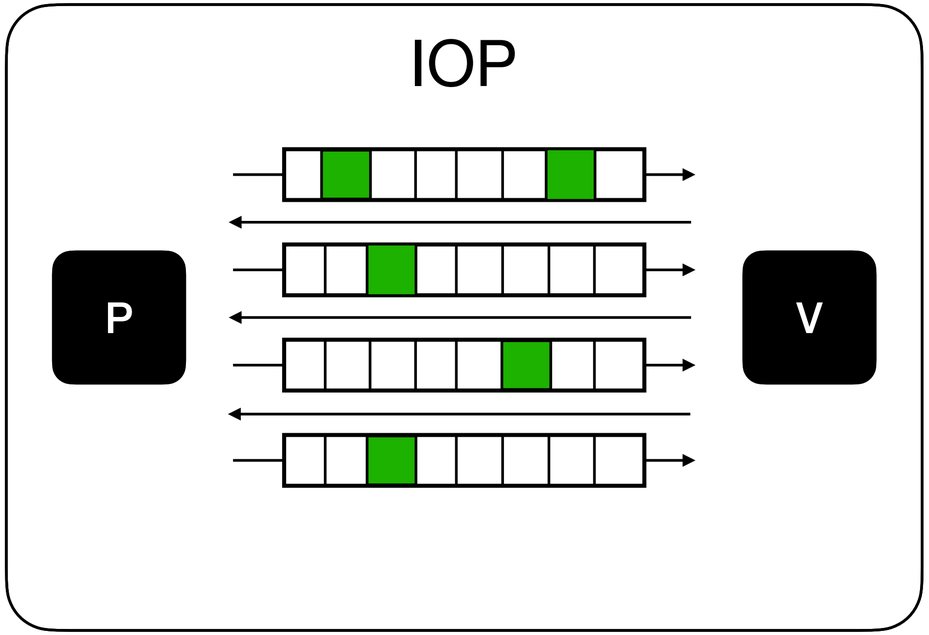
Constructing SNARKs

[BCS16] Construction



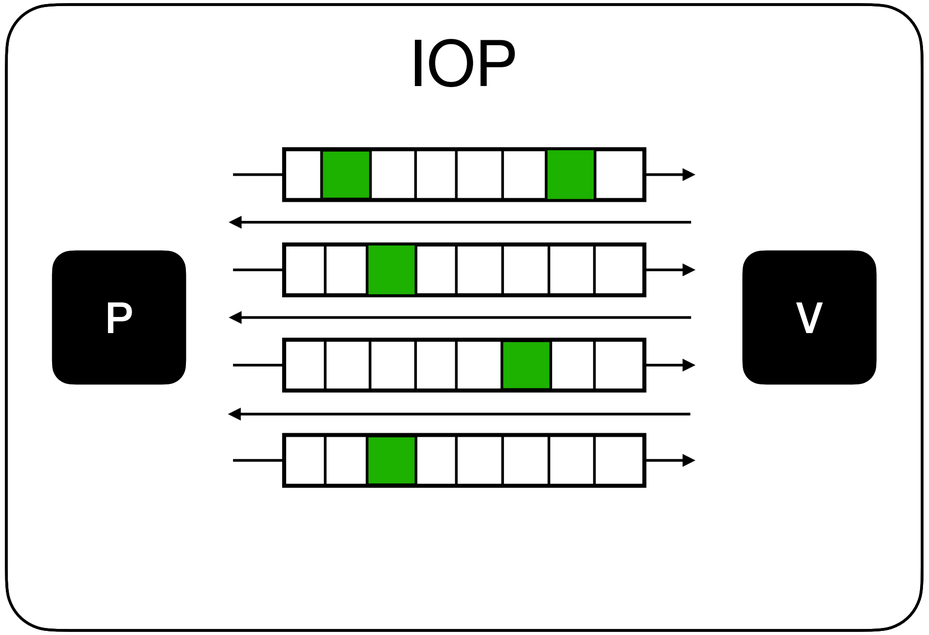
Constructing SNARKs

[BCS16] Construction



Constructing SNARKs

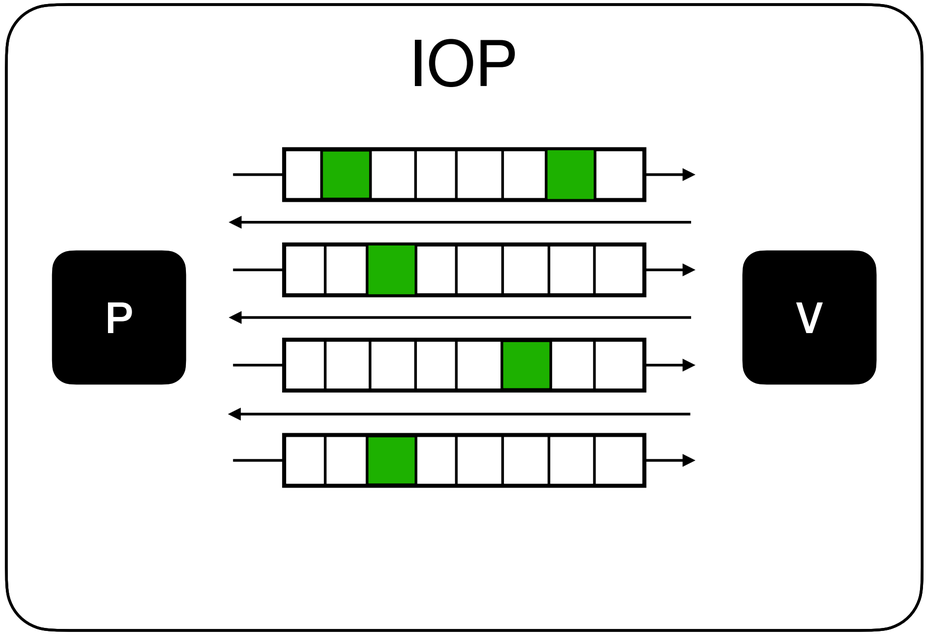
[BCS16] Construction



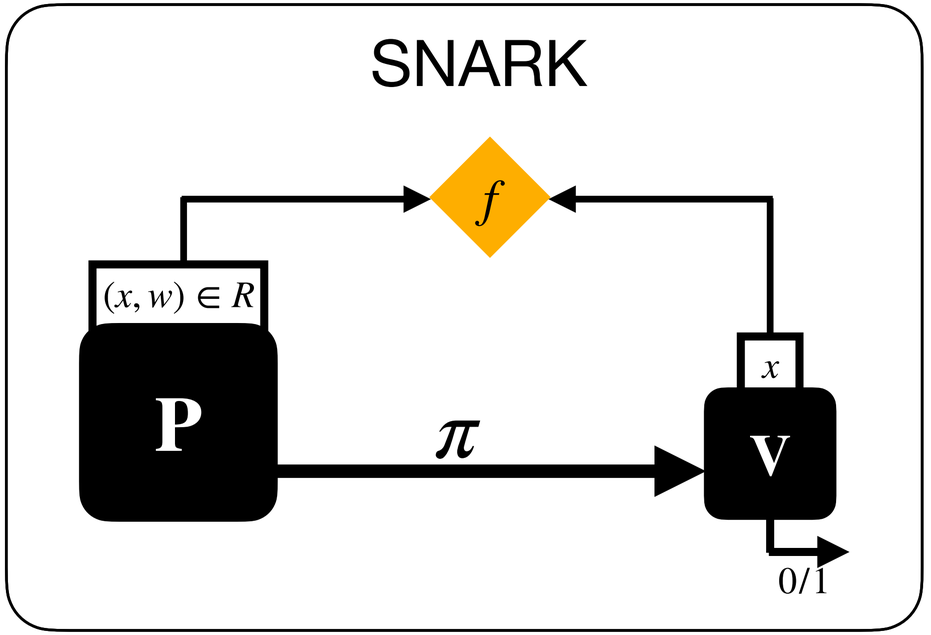
BCS construction:
Merkle Trees + FS

Constructing SNARKs

[BCS16] Construction

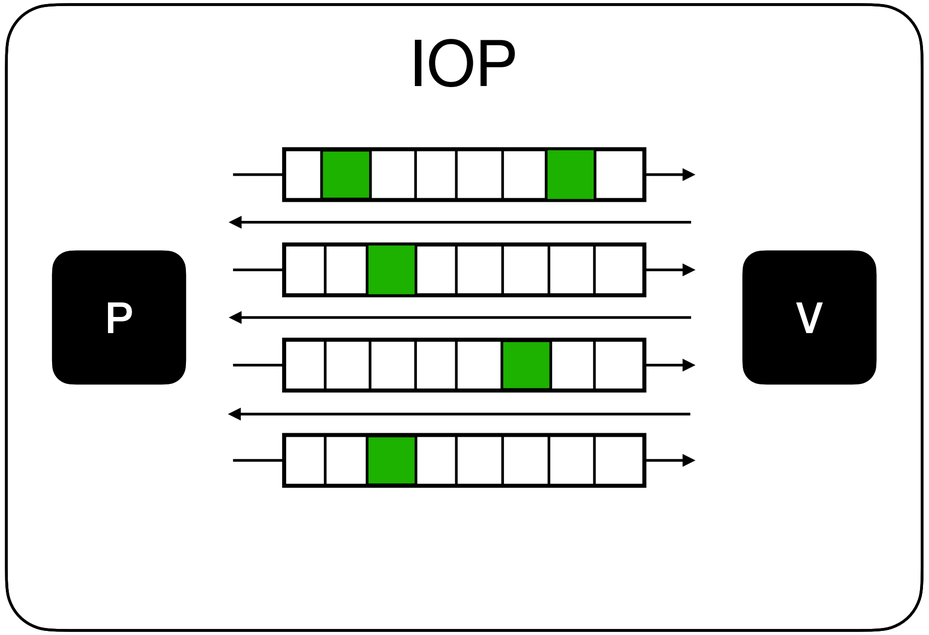


BCS construction:
Merkle Trees + FS

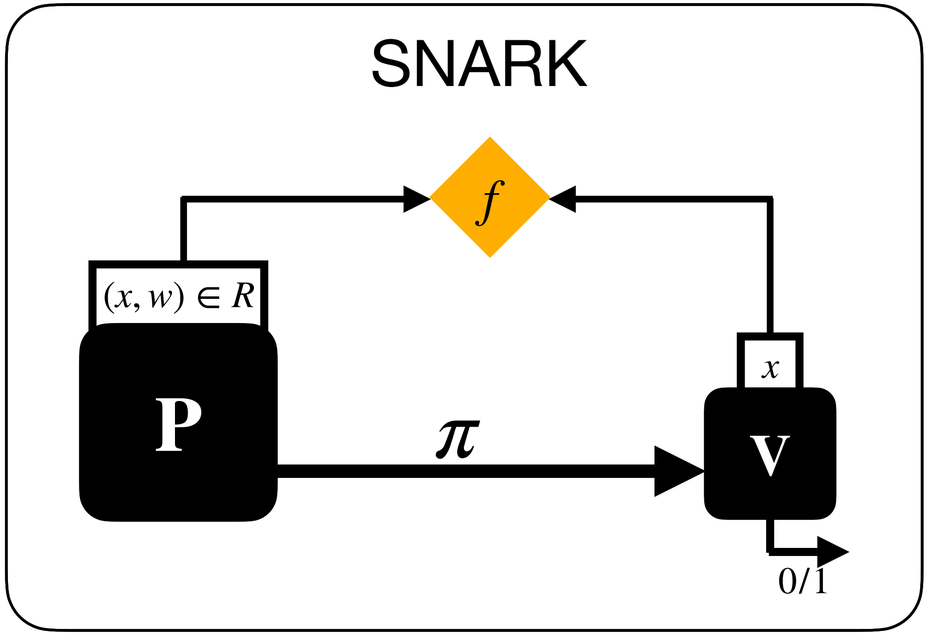


Constructing SNARKs

[BCS16] Construction



BCS construction:
Merkle Trees + FS

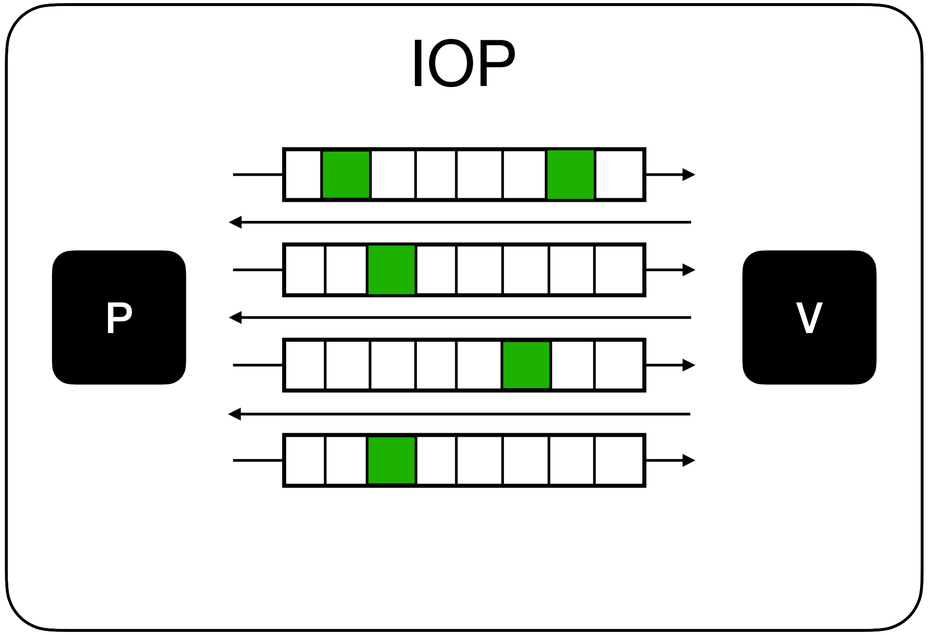


Proof length l $\approx O(n)$

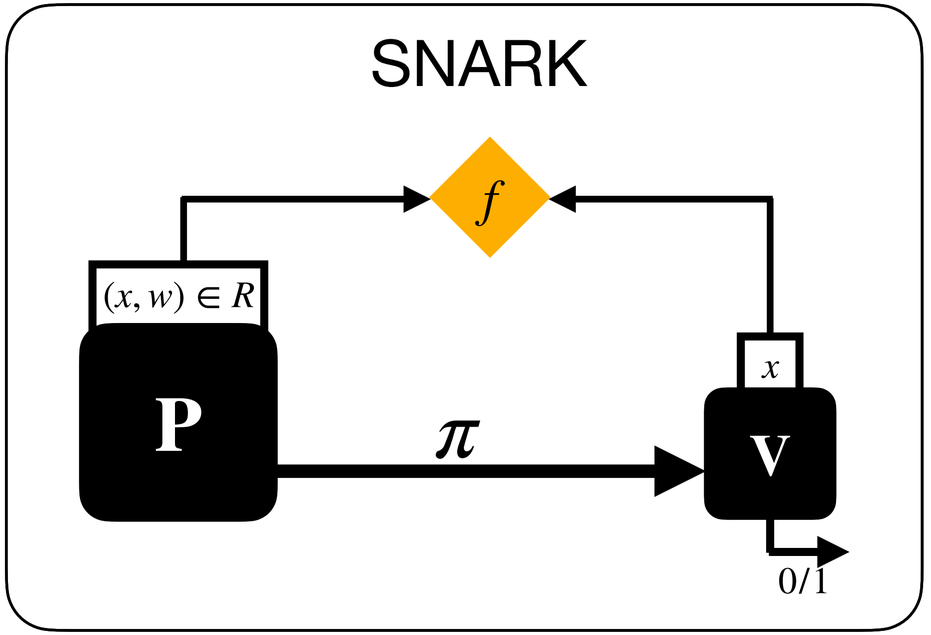
Queries q $\approx O(\log n)$

Constructing SNARKs

[BCS16] Construction



BCS construction:
Merkle Trees + FS

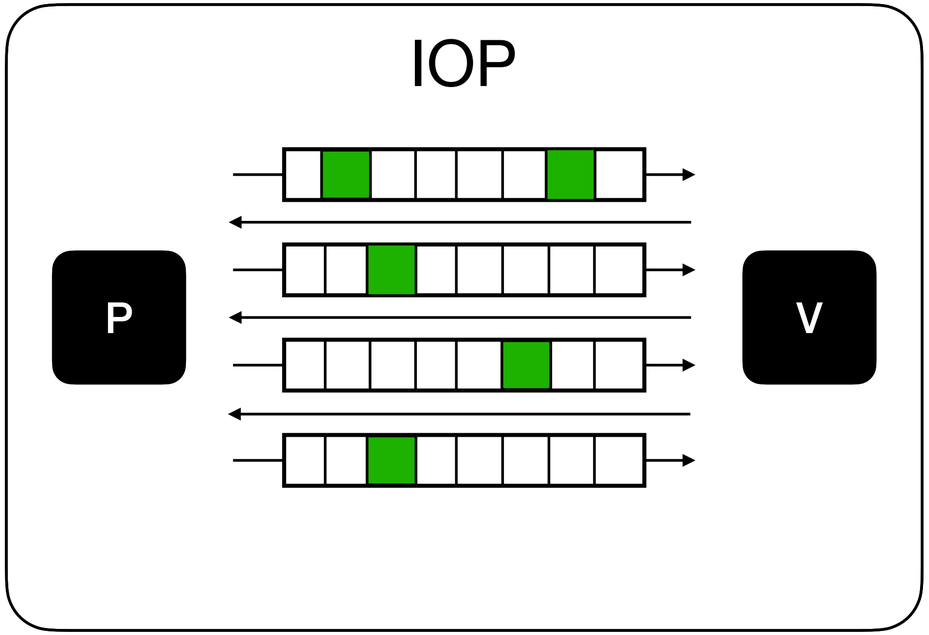


Proof length l $\approx O(n)$ — Large, think 2^{24}

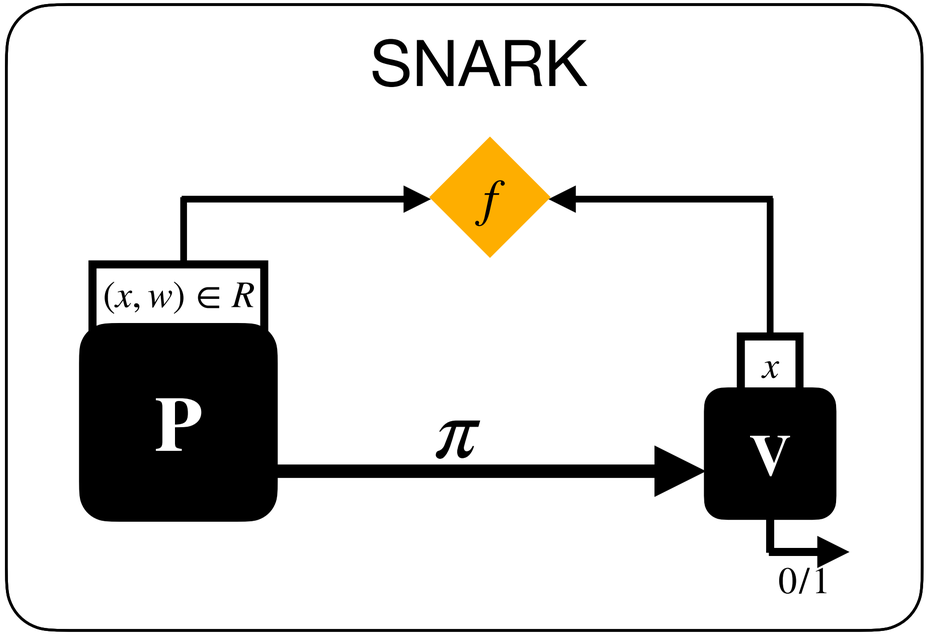
Queries q $\approx O(\log n)$ — Small, think ~400

Constructing SNARKs

[BCS16] Construction



BCS construction:
Merkle Trees + FS



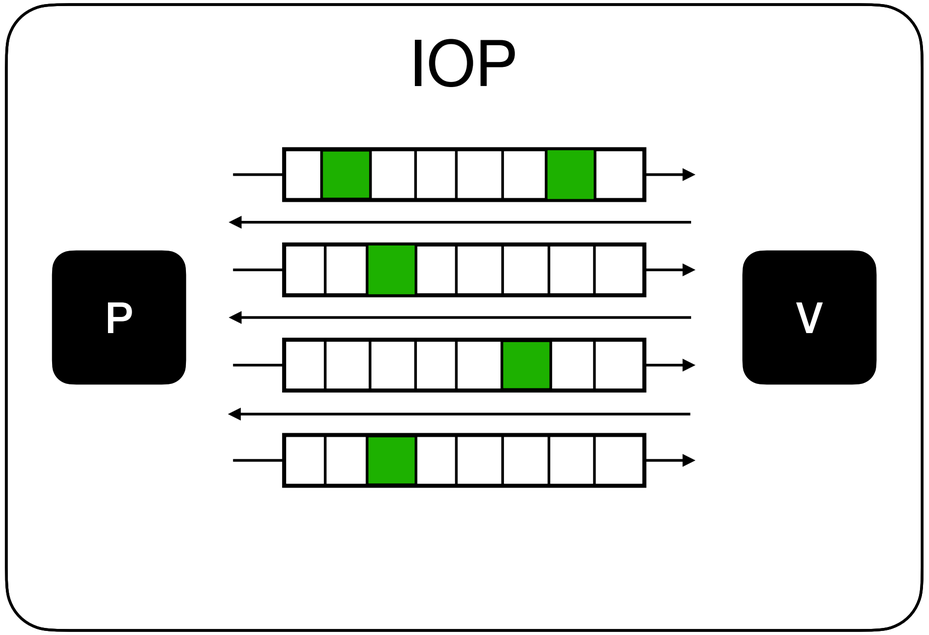
Proof length $\mathsf{l} \approx O(n)$ — Large, think 2^{24}

Queries $\mathsf{q} \approx O(\log n)$ — Small, think ~400

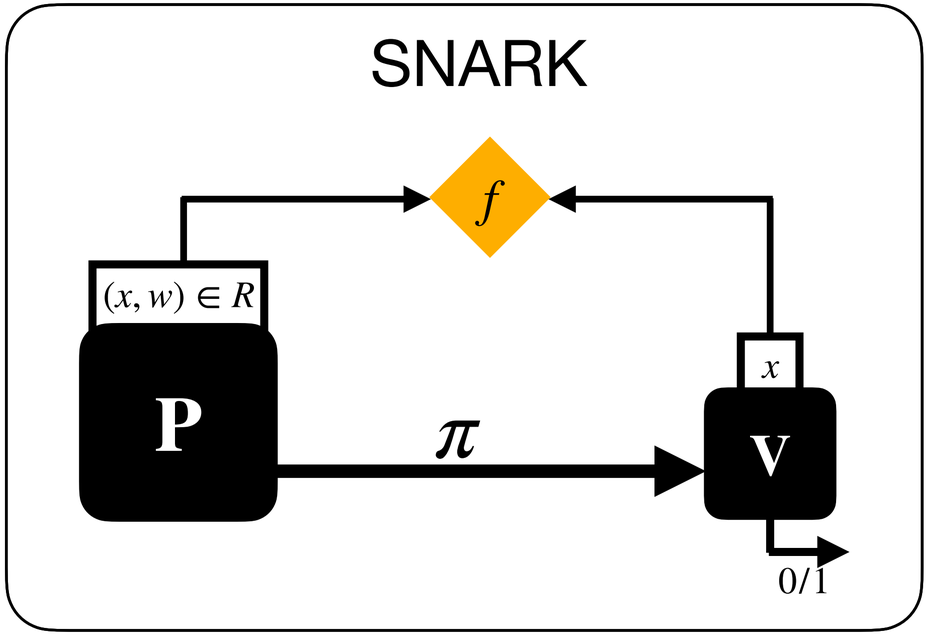
Argument size $O(\lambda \cdot \mathsf{q} \cdot \log \mathsf{l})$

Constructing SNARKs

[BCS16] Construction



BCS construction:
Merkle Trees + FS



Proof length $\mathsf{l} \approx O(n)$ — Large, think 2^{24}

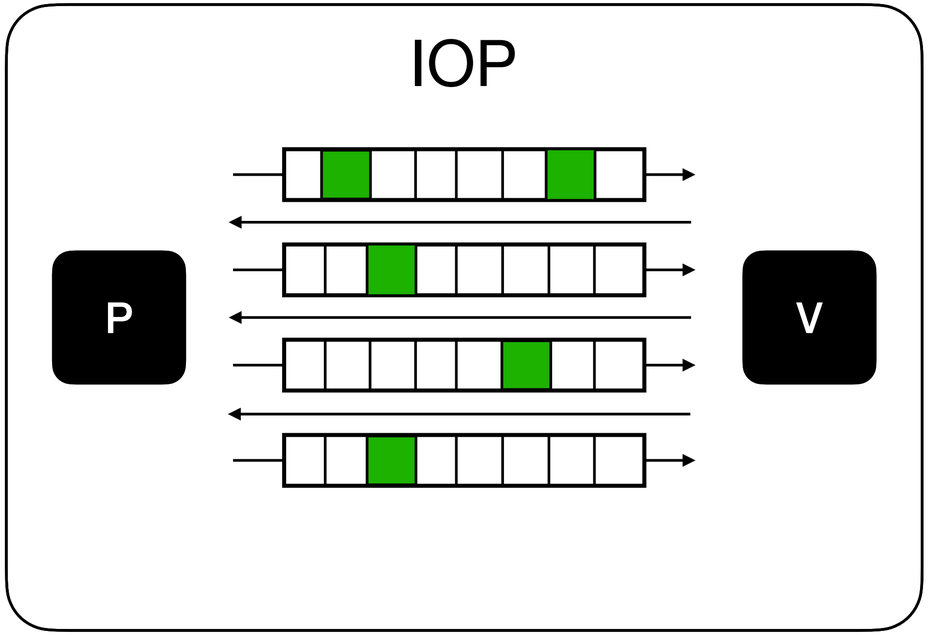
Queries $\mathsf{q} \approx O(\log n)$ — Small, think ~400

Argument size $O(\lambda \cdot \mathsf{q} \cdot \log \mathsf{l})$ — Small, tens of KiB

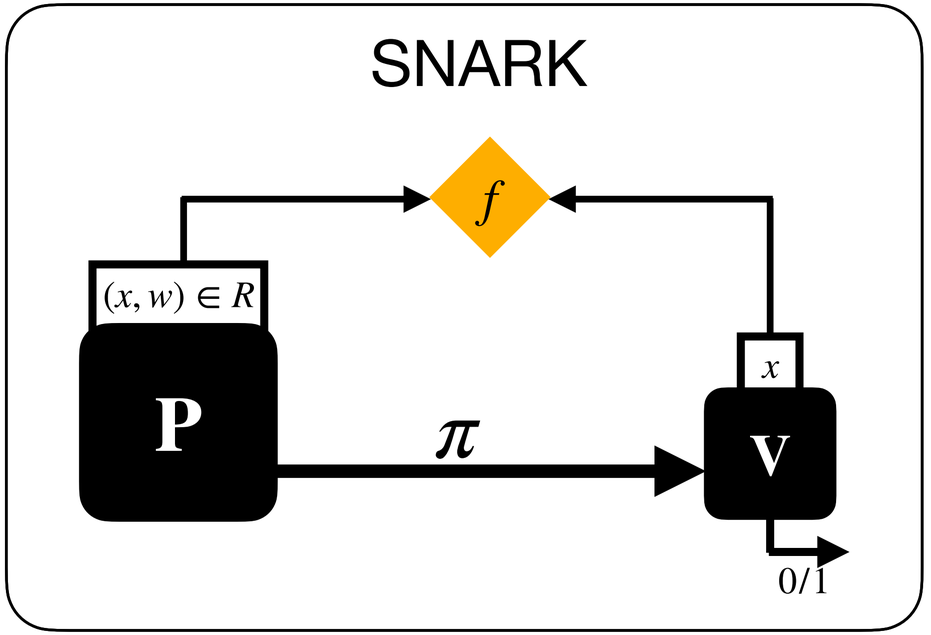
Constructing SNARKs

[BCS16] Construction

In this talk, we focus on the IOP!



BCS construction:
Merkle Trees + FS



Proof length $l \approx O(n)$ — Large, think 2^{24}

Queries $q \approx O(\log n)$ — Small, think ~400

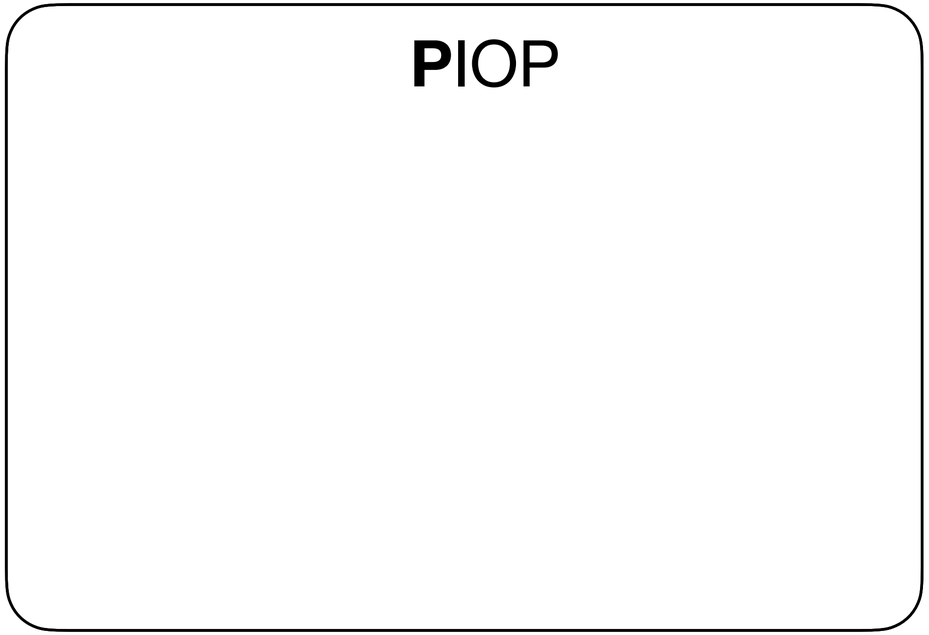
Argument size $O(\lambda \cdot q \cdot \log l)$ — Small, tens of KiB

Constructing IOPs

Traditionally

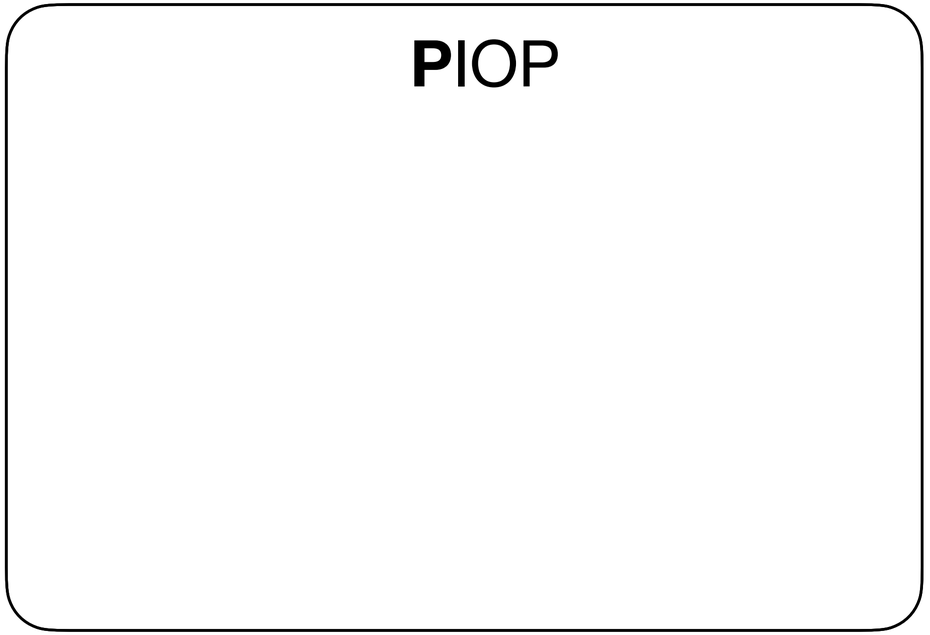
Constructing IOPs

Traditionally



Constructing IOPs

Traditionally

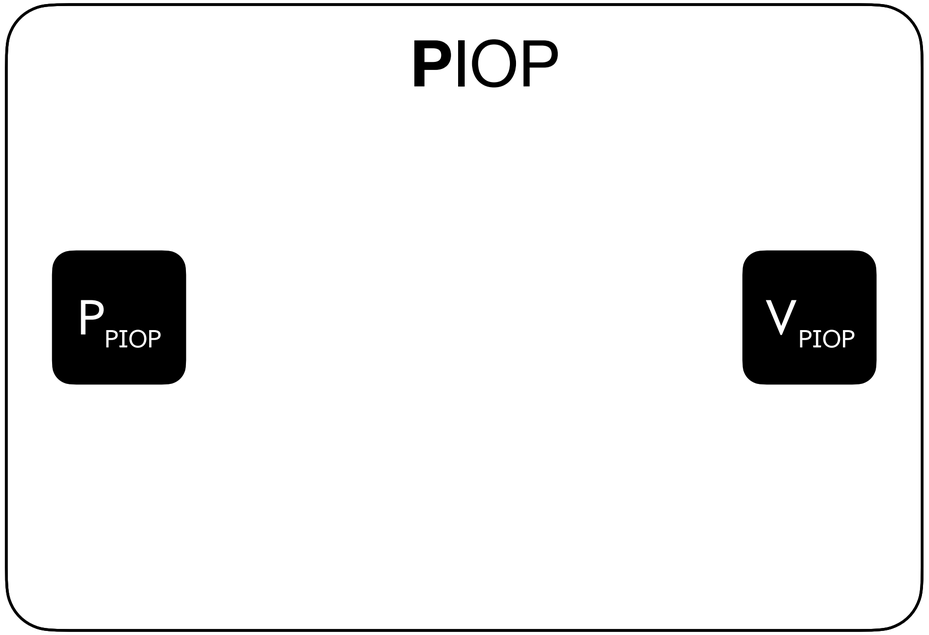


Just like IOPs, but prover is forced to send polynomials $\mathbb{F}^{<d}[X]$.

E.g. Aurora, STARK PIOP etc.

Constructing IOPs

Traditionally

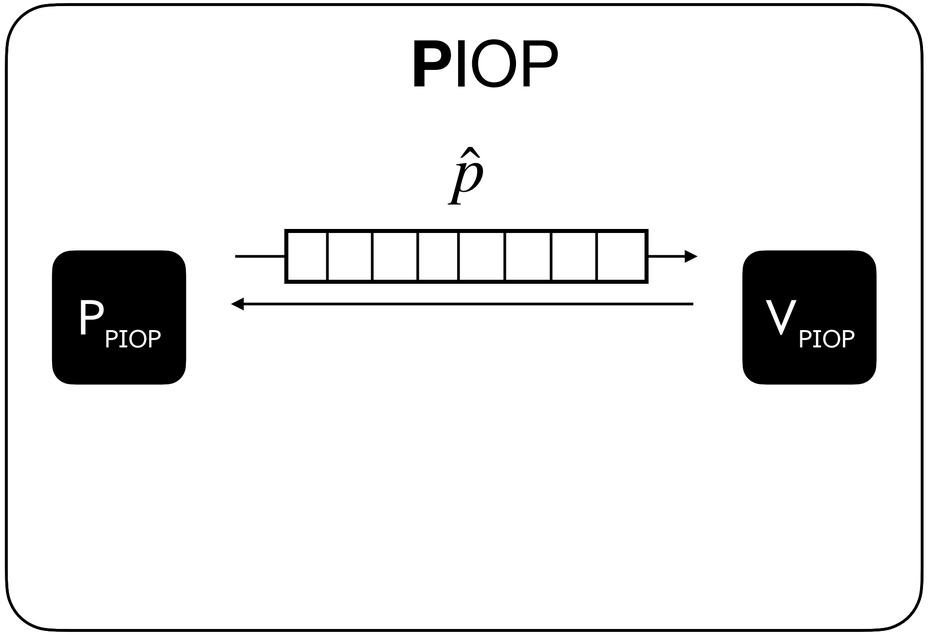


Just like IOPs, but prover is forced to send polynomials $\mathbb{F}^{<d}[X]$.

E.g. Aurora, STARK PIOP etc.

Constructing IOPs

Traditionally

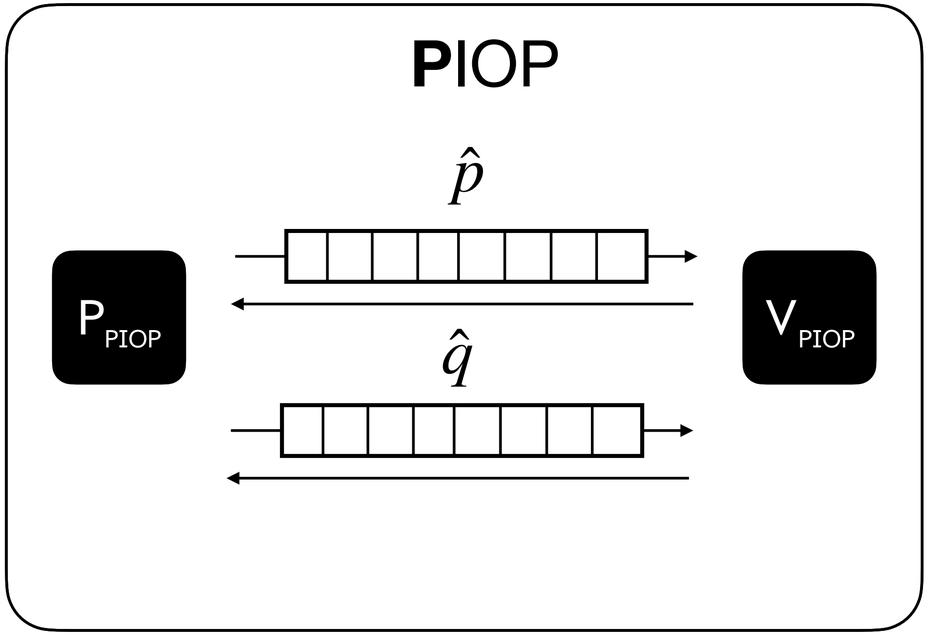


Just like IOPs, but prover is forced to send polynomials $\mathbb{F}^{<d}[X]$.

E.g. Aurora, STARK PIOP etc.

Constructing IOPs

Traditionally

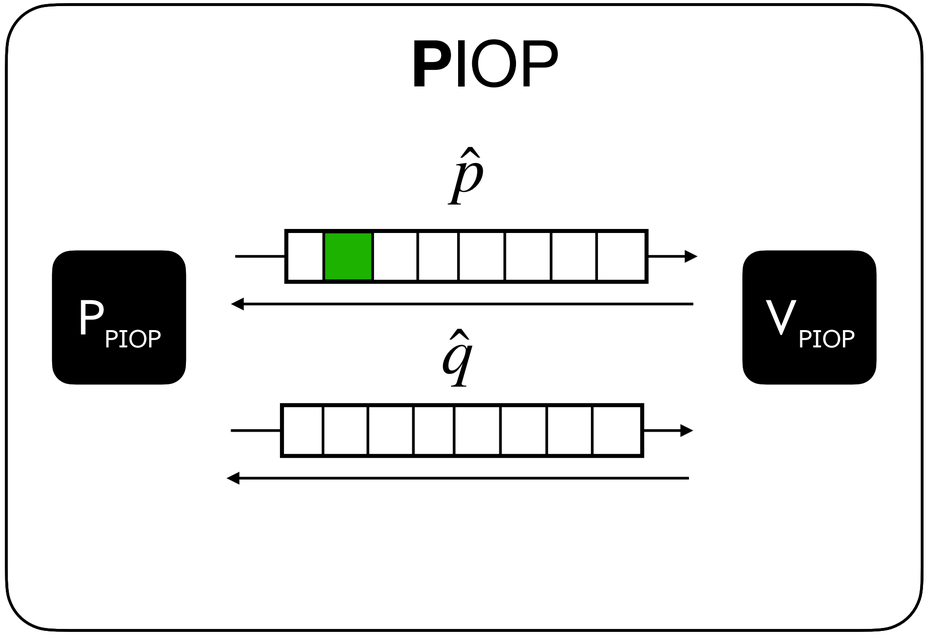


Just like IOPs, but prover is forced to send polynomials $\mathbb{F}^{<d}[X]$.

E.g. Aurora, STARK PIOP etc.

Constructing IOPs

Traditionally

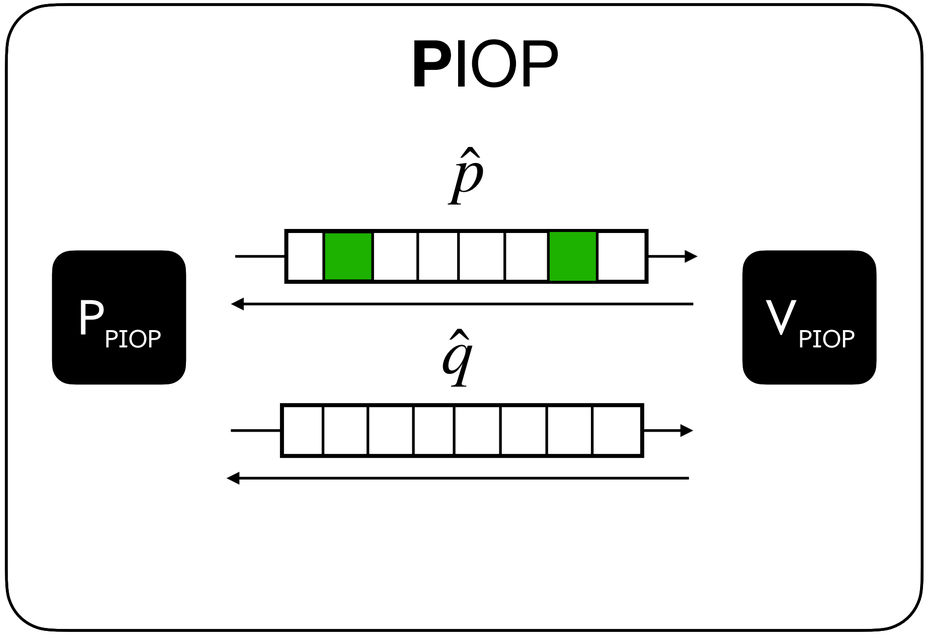


Just like IOPs, but prover is forced to send polynomials $\mathbb{F}^{<d}[X]$.

E.g. Aurora, STARK PIOP etc.

Constructing IOPs

Traditionally

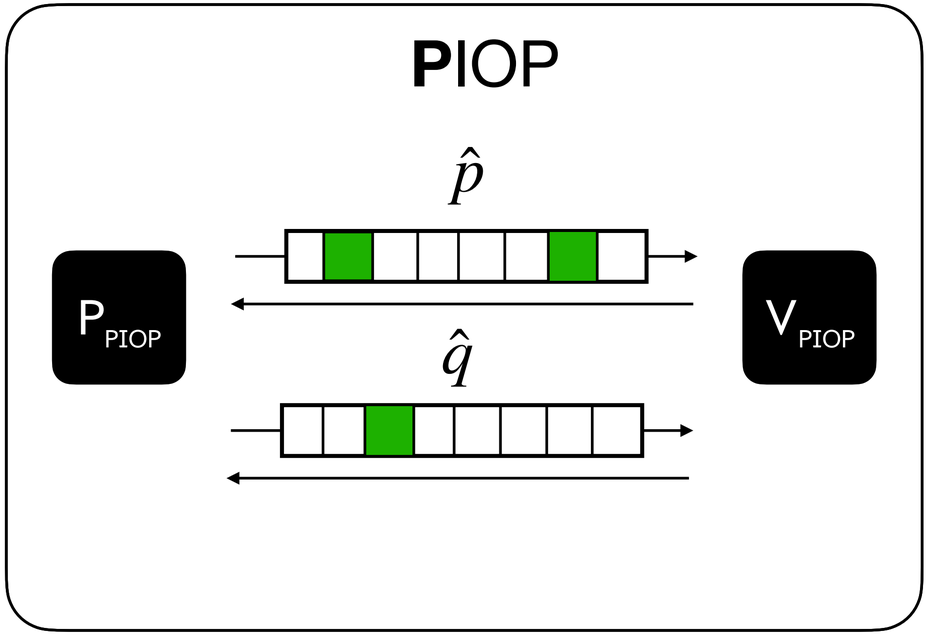


Just like IOPs, but prover is forced to send polynomials $\mathbb{F}^{<d}[X]$.

E.g. Aurora, STARK PIOP etc.

Constructing IOPs

Traditionally

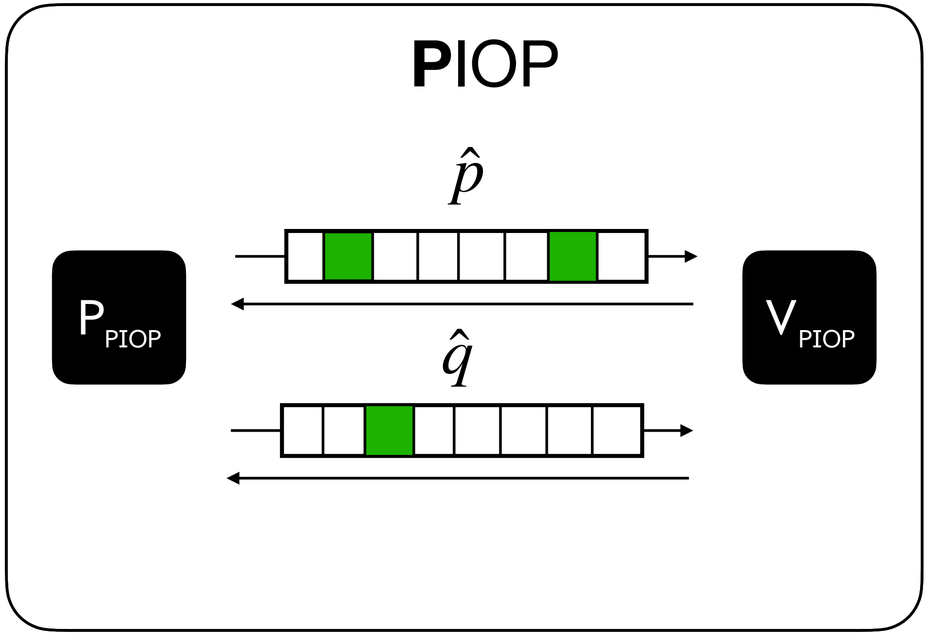


Just like IOPs, but prover is forced to send polynomials $\mathbb{F}^{<d}[X]$.

E.g. Aurora, STARK PIOP etc.

Constructing IOPs

Traditionally



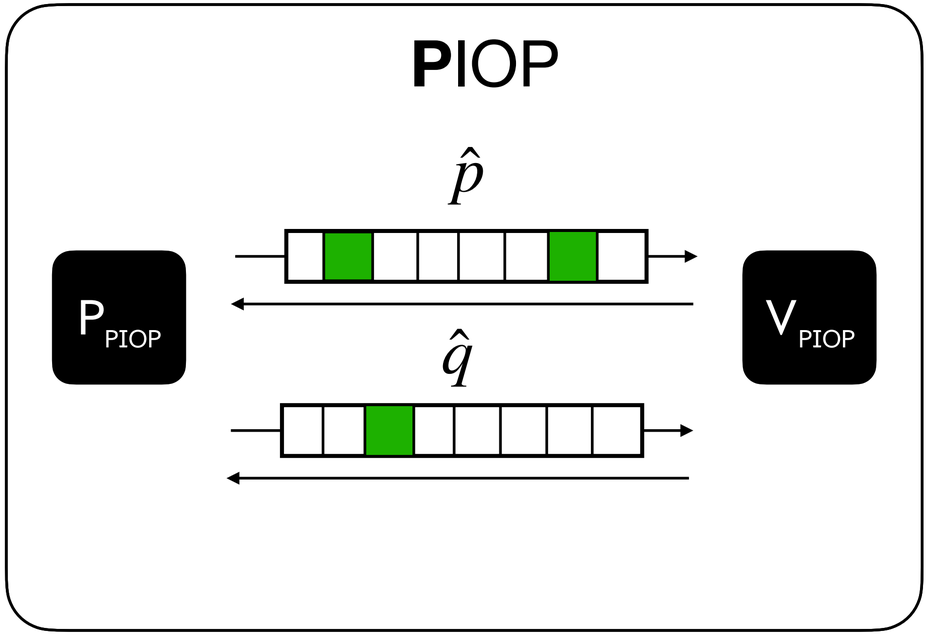
Just like IOPs, but prover is forced to send polynomials $\mathbb{F}^{<d}[X]$.

E.g. Aurora, STARK PIOP etc.

Strategy: use Reed-Solomon codes as “redundant” encoding. Use a proximity test to check claims on encoded oracles.

Constructing IOPs

Traditionally



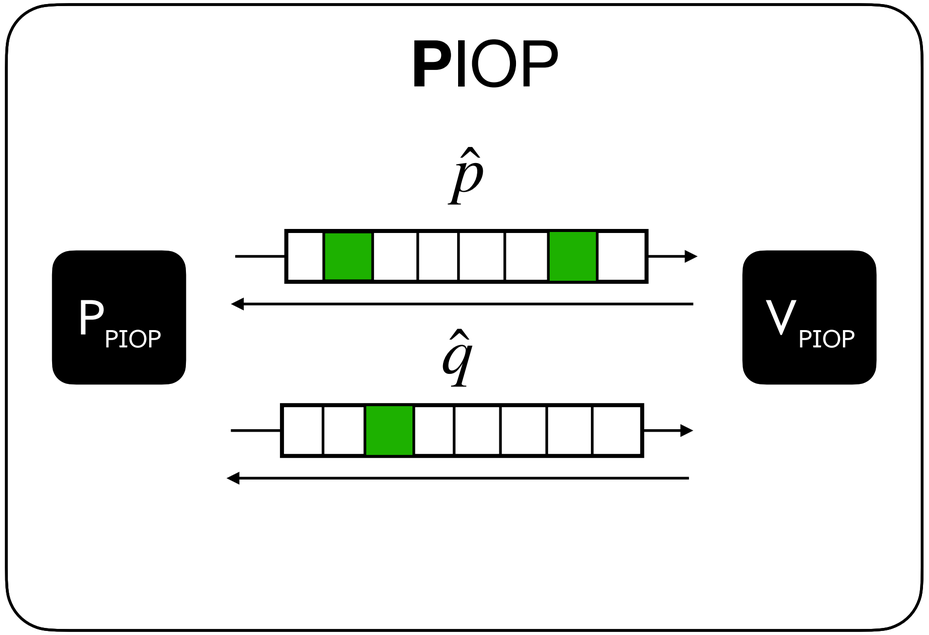
Just like IOPs, but prover is forced to send polynomials $\mathbb{F}^{<d}[X]$.

E.g. Aurora, STARK PIOP etc.

Strategy: use Reed-Solomon codes as “redundant” encoding. Use a proximity test to check claims on encoded oracles.

Constructing IOPs

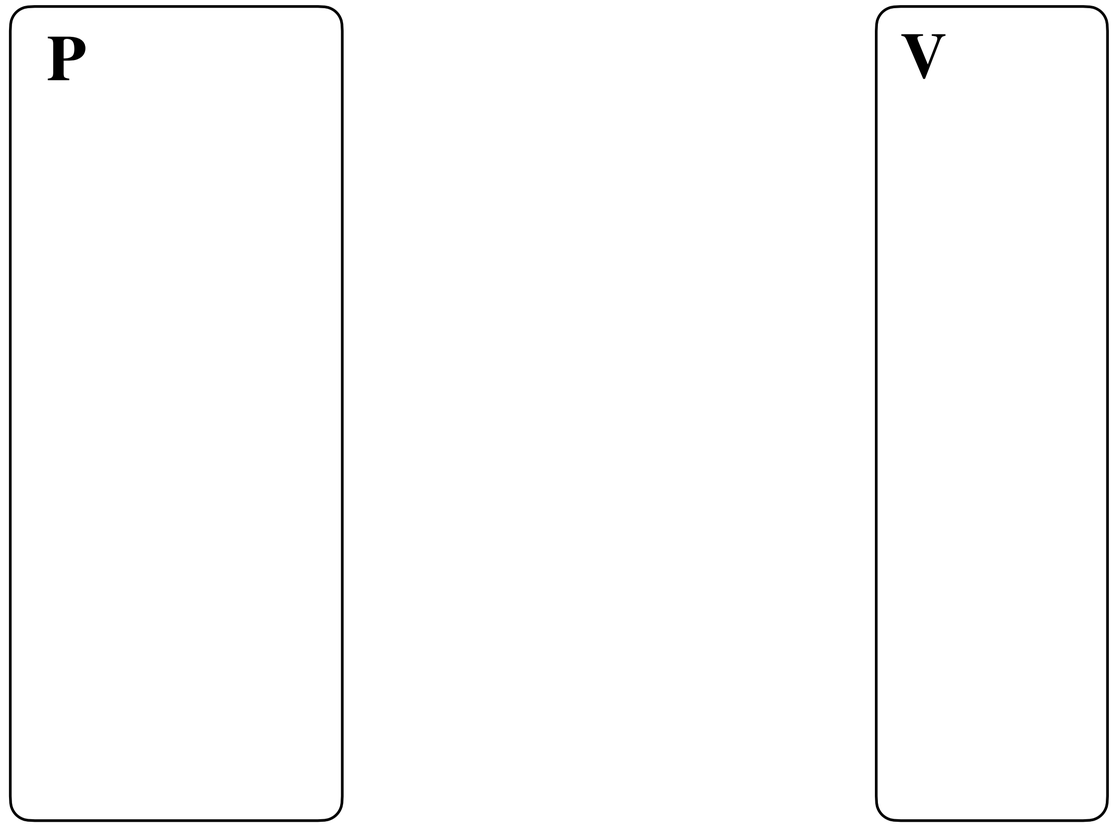
Traditionally



Just like IOPs, but prover is forced to send polynomials $\mathbb{F}^{<d}[X]$.

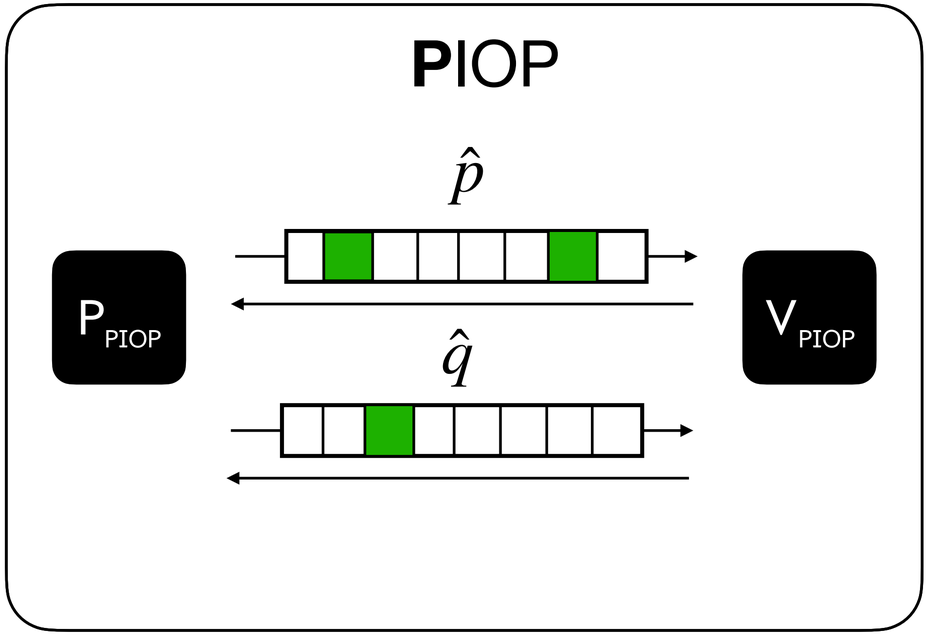
E.g. Aurora, STARK PIOP etc.

Strategy: use Reed-Solomon codes as "redundant" encoding. Use a proximity test to check claims on encoded oracles.



Constructing IOPs

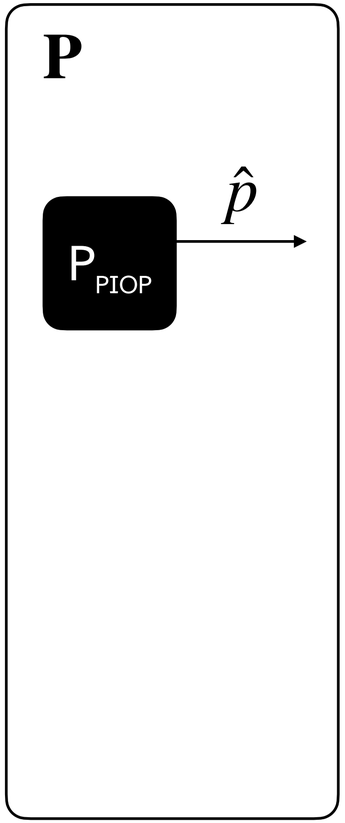
Traditionally

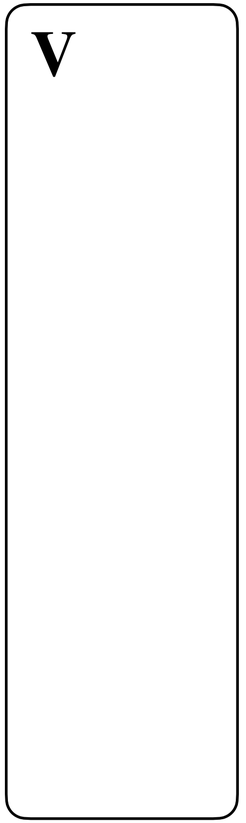


Just like IOPs, but prover is forced to send polynomials $\mathbb{F}^{<d}[X]$.

E.g. Aurora, STARK PIOP etc.

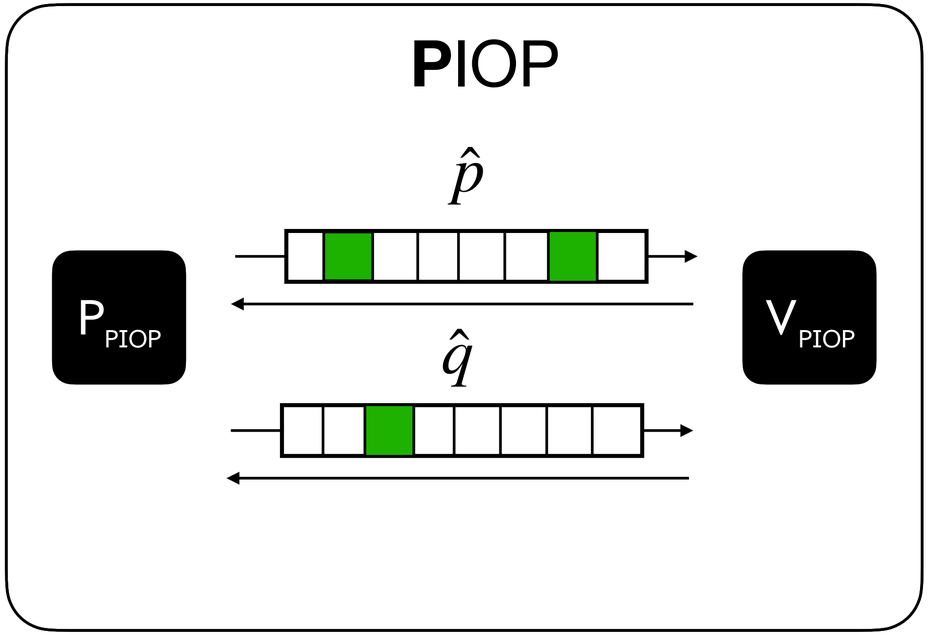
Strategy: use Reed-Solomon codes as "redundant" encoding. Use a proximity test to check claims on encoded oracles.





Constructing IOPs

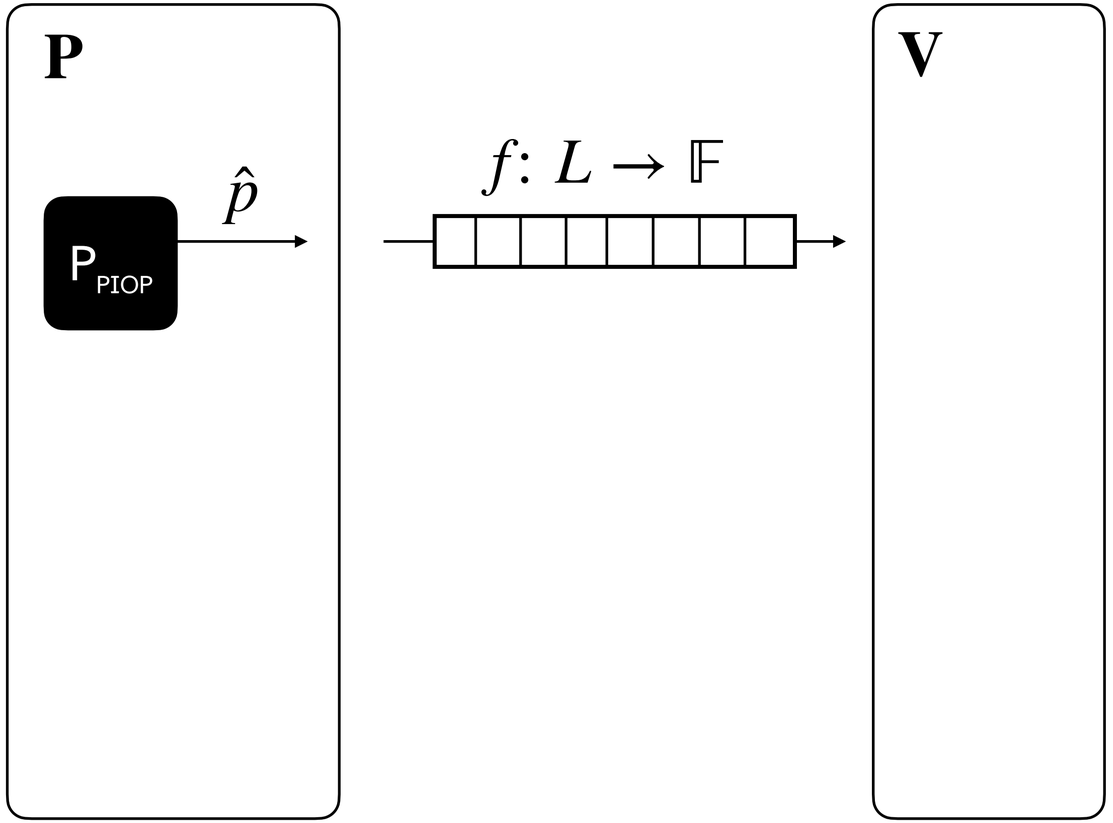
Traditionally



Just like IOPs, but prover is forced to send polynomials $\mathbb{F}^{<d}[X]$.

E.g. Aurora, STARK PIOP etc.

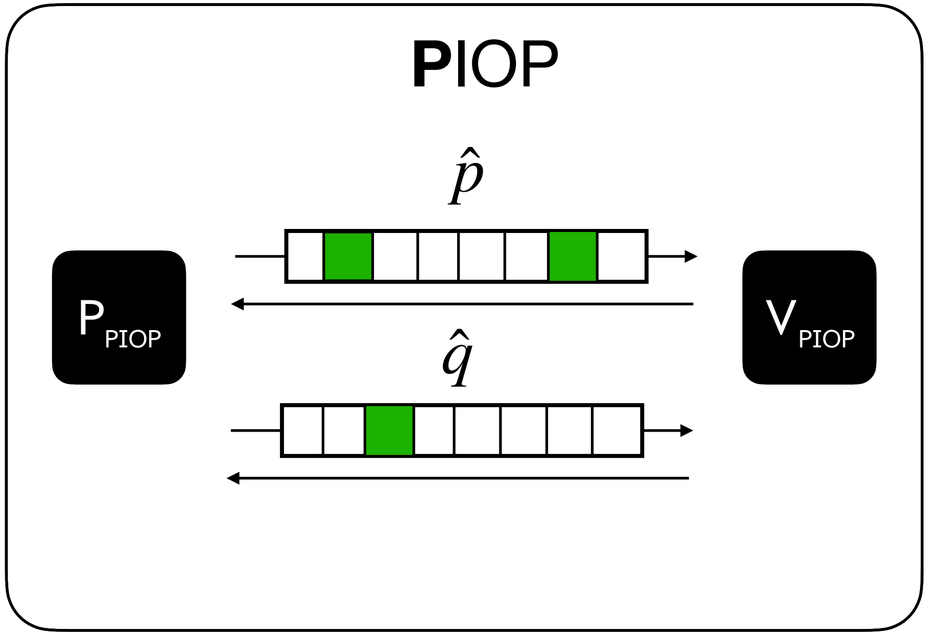
Strategy: use Reed-Solomon codes as "redundant" encoding. Use a proximity test to check claims on encoded oracles.

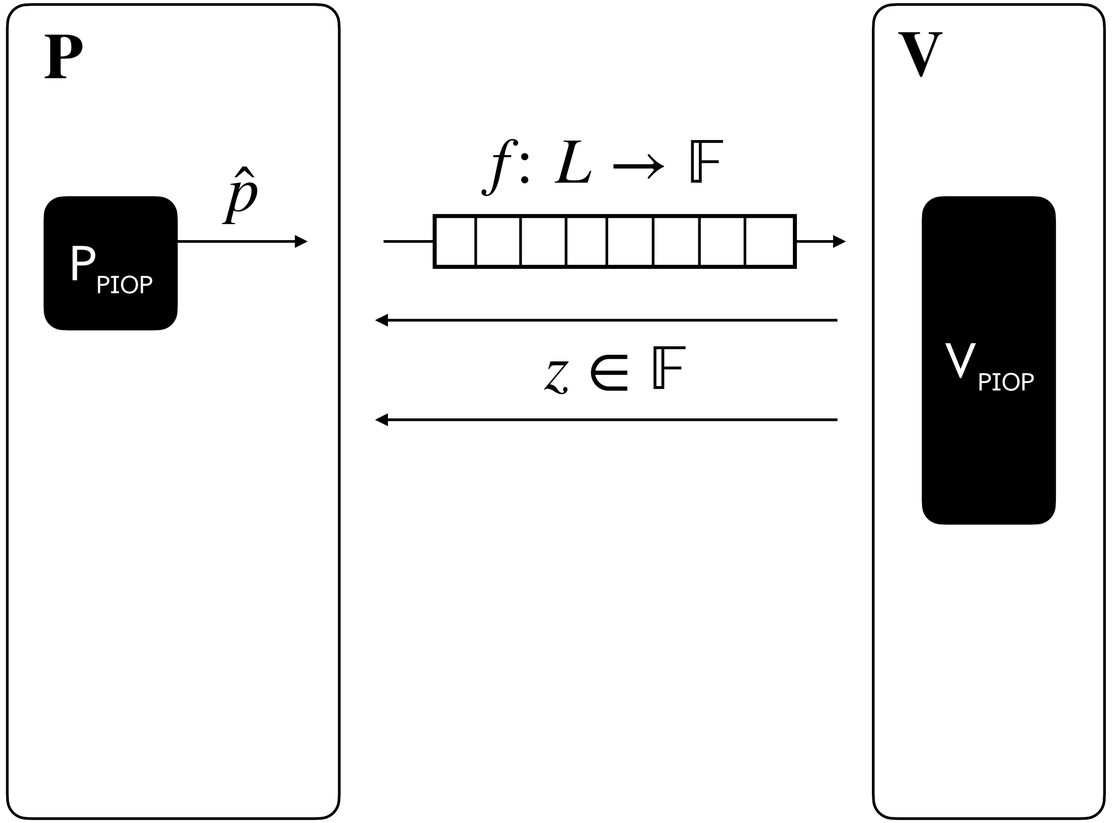


Constructing IOPs

Traditionally

Strategy: use Reed-Solomon codes as "redundant" encoding. Use a proximity test to check claims on encoded oracles.





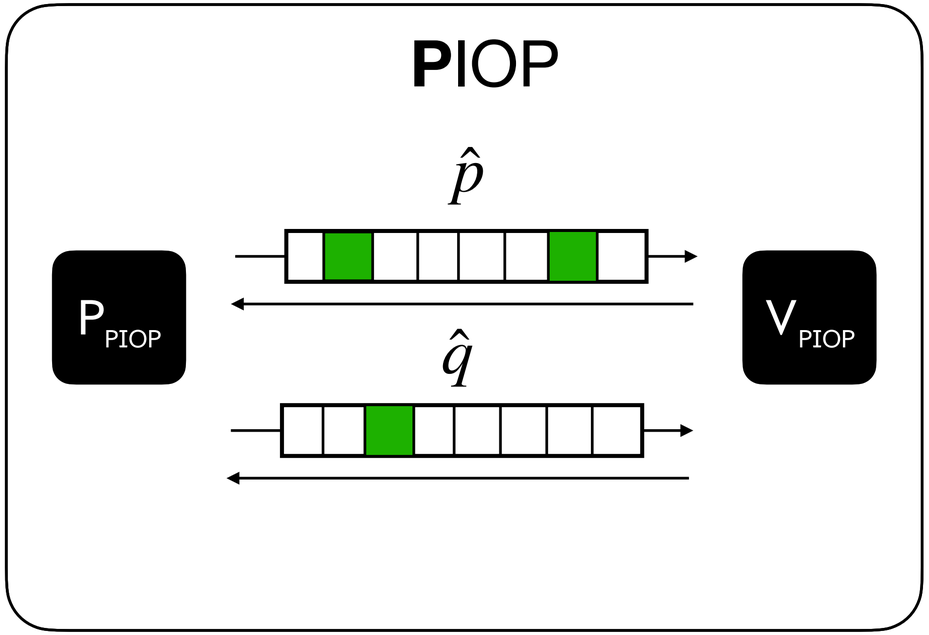
Just like IOPs, but prover is forced to send polynomials $\mathbb{F}^{<d}[X]$.

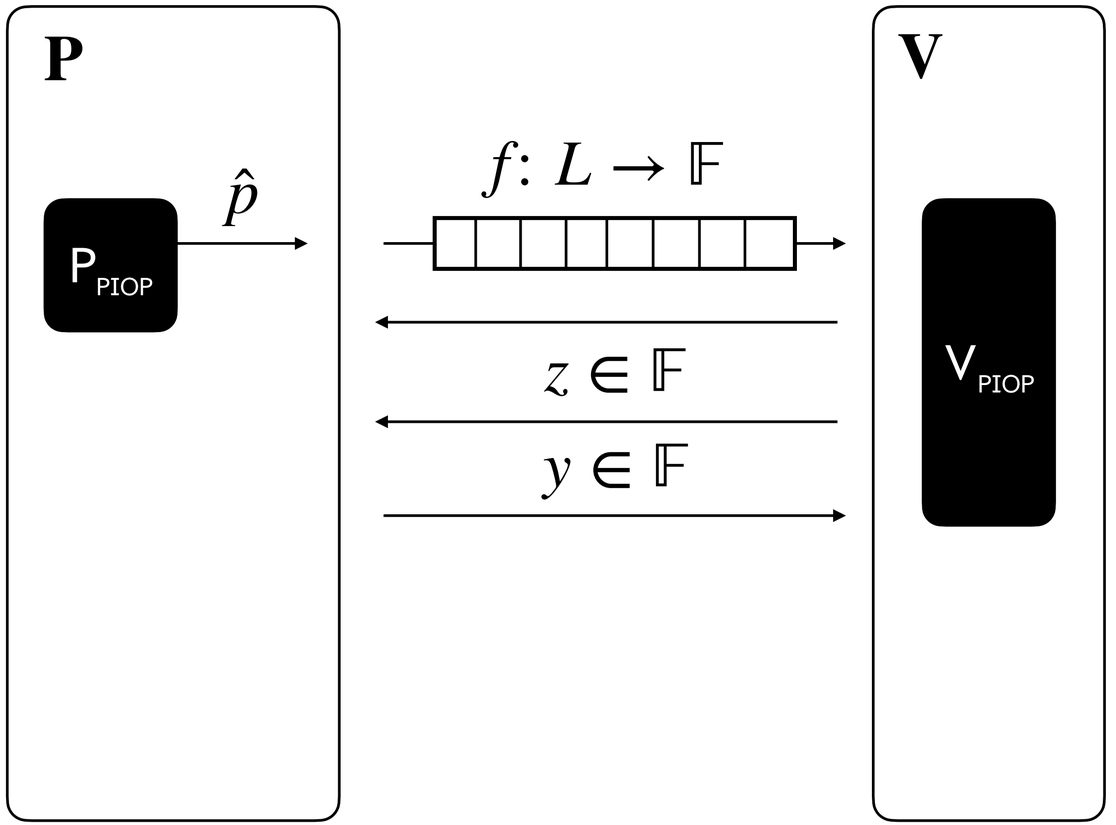
E.g. Aurora, STARK PIOP etc.

Constructing IOPs

Traditionally

Strategy: use Reed-Solomon codes as "redundant" encoding. Use a proximity test to check claims on encoded oracles.



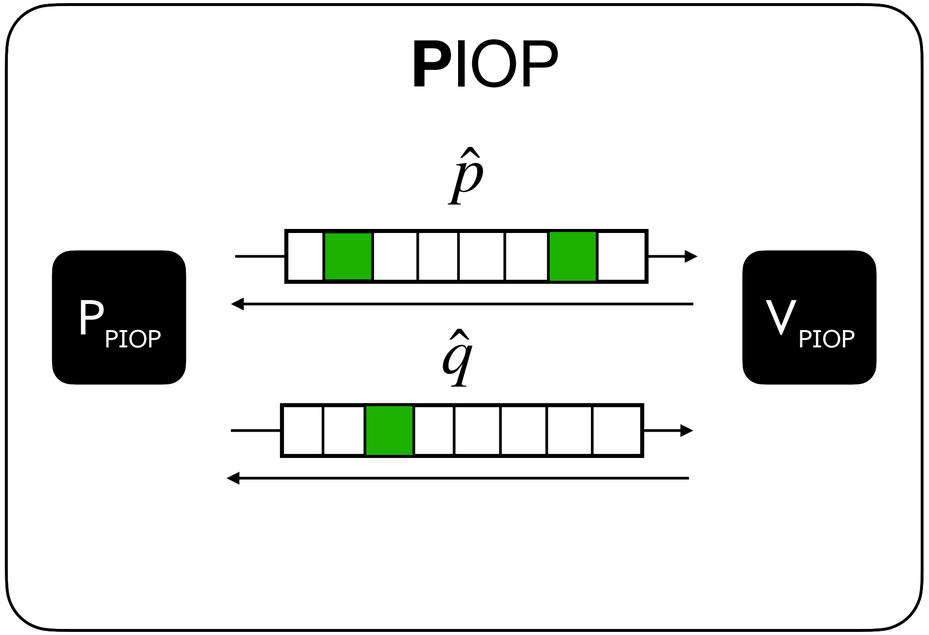


Just like IOPs, but prover is forced to send polynomials $\mathbb{F}^{<d}[X]$.

E.g. Aurora, STARK PIOP etc.

Constructing IOPs

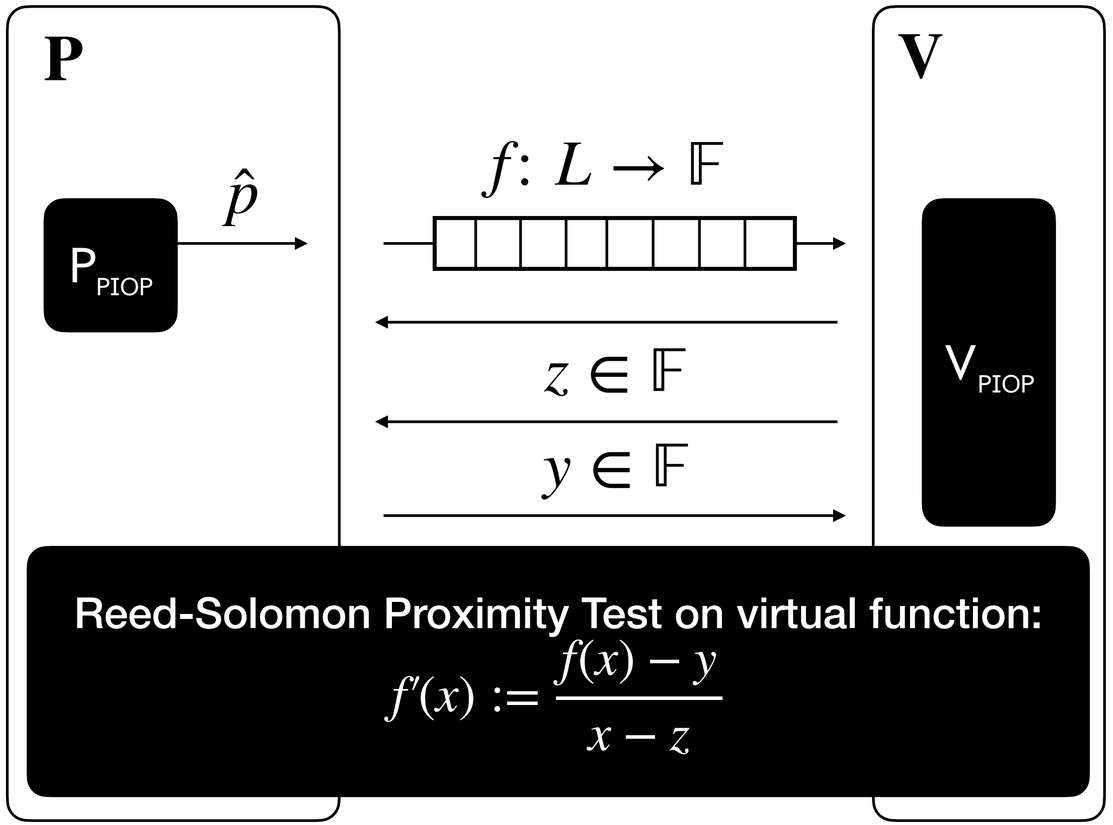
Traditionally



Just like IOPs, but prover is forced to send polynomials $\mathbb{F}^{<d}[X]$.

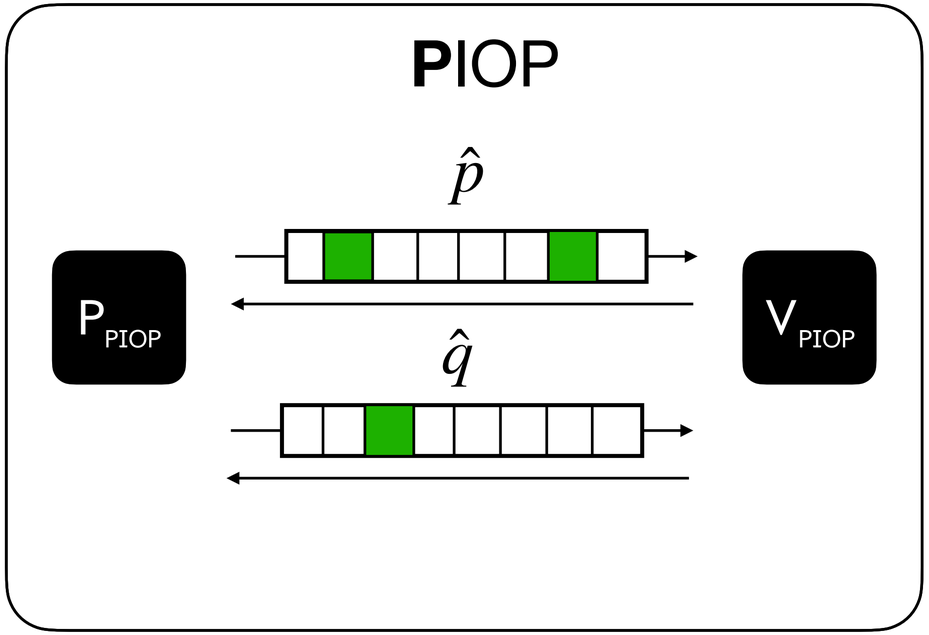
E.g. Aurora, STARK PIOP etc.

Strategy: use Reed-Solomon codes as "redundant" encoding. Use a proximity test to check claims on encoded oracles.



Constructing IOPs

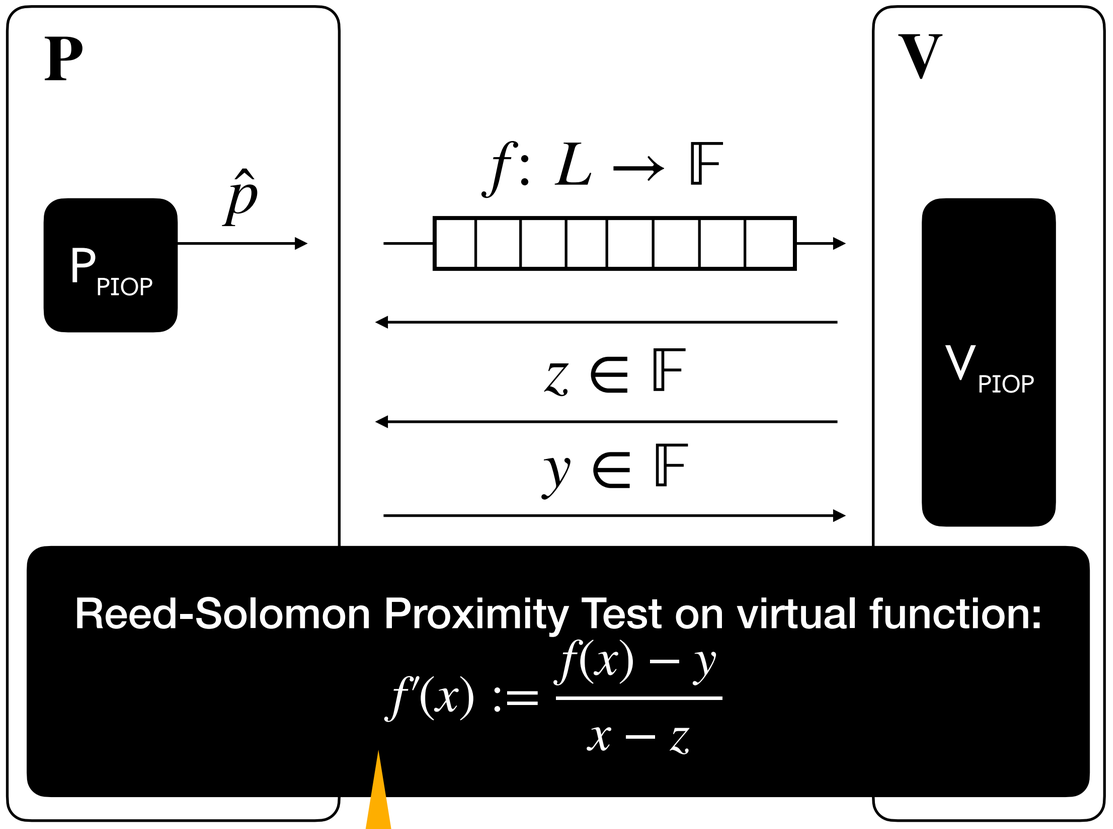
Traditionally



Just like IOPs, but prover is forced to send polynomials $\mathbb{F}^{<d}[X]$.

E.g. Aurora, STARK PIOP etc.

Strategy: use Reed-Solomon codes as "redundant" encoding. Use a proximity test to check claims on encoded oracles.



The IOP inherits asymptotics almost entirely from the proximity test

IOP of Proximity to RS codes

IOP of Proximity to RS codes

$\mathsf{RS}[n, m, \rho] :=$

IOP of Proximity to RS codes

Convenience

$$\mathrm{RS}[n, m, \rho] := \left\{ \begin{array}{l} \text{Evaluations of polynomials of degree } < 2^m \\ \text{on a domain } L \subseteq \mathbb{F} \text{ of size } n.\ \rho := \dfrac{2^m}{n} \end{array} \right\}$$

IOP of Proximity to RS codes

Convenience

$$\mathrm{RS}[n, m, \rho] := \left\{ \begin{array}{l} \text{Evaluations of polynomials of degree } < 2^m \\ \text{on a domain } L \subseteq \mathbb{F} \text{ of size } n.\ \rho := \dfrac{2^m}{n} \end{array} \right\}$$

Rate of the code

IOP of Proximity to RS codes

Convenience

$$\mathrm{RS}[n, m, \rho] := \left\{ \begin{array}{l} \text{Evaluations of polynomials of degree } < 2^m \\ \text{on a domain } L \subseteq \mathbb{F} \text{ of size } n.\ \rho := \dfrac{2^m}{n} \end{array} \right\}$$

Rate of the code

IOPP for RS

IOP of Proximity to RS codes

Convenience

$$\mathrm{RS}[n, m, \rho] := \left\{ \begin{array}{l} \text{Evaluations of polynomials of degree } < 2^m \\ \text{on a domain } L \subseteq \mathbb{F} \text{ of size } n.\ \rho := \dfrac{2^m}{n} \end{array} \right\}$$

Rate of the code

IOPP for RS

P

V

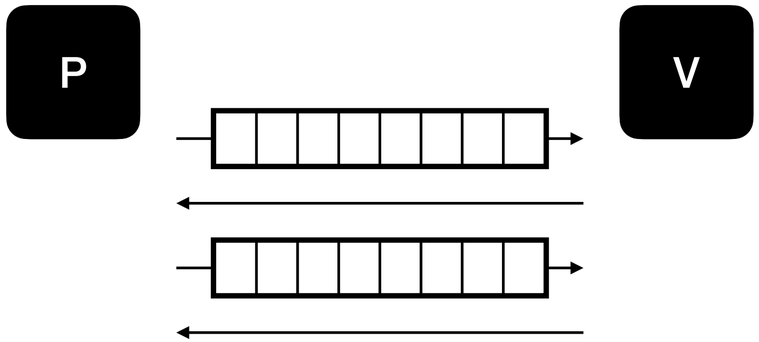
IOP of Proximity to RS codes

Convenience

$$\mathrm{RS}[n, m, \rho] := \left\{ \begin{array}{l} \text{Evaluations of polynomials of degree } < 2^m \\ \text{on a domain } L \subseteq \mathbb{F} \text{ of size } n.\ \rho := \dfrac{2^m}{n} \end{array} \right\}$$

Rate of the code

IOPP for RS



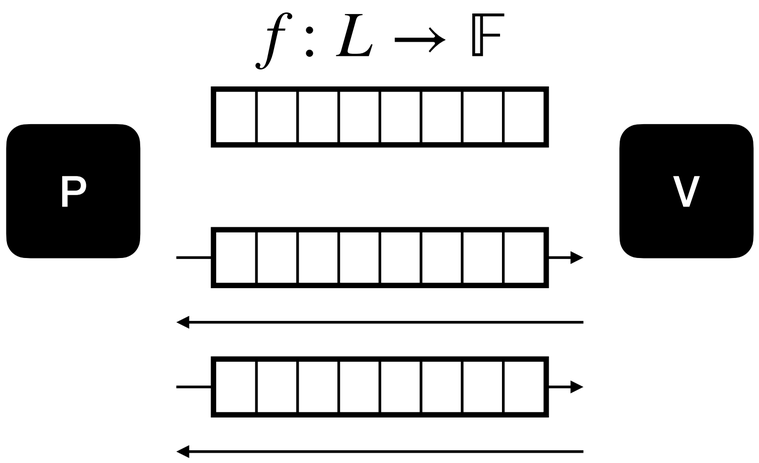
IOP of Proximity to RS codes

Convenience

$$\mathrm{RS}[n, m, \rho] := \left\{ \begin{array}{l} \text{Evaluations of polynomials of degree } < 2^m \\ \text{on a domain } L \subseteq \mathbb{F} \text{ of size } n.\ \rho := \dfrac{2^m}{n} \end{array} \right\}$$

Rate of the code

IOPP for RS



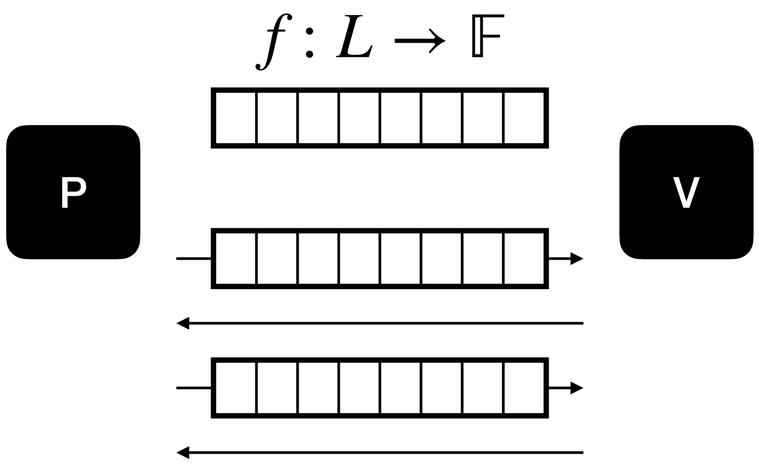
IOP of Proximity to RS codes

Convenience

$$\mathrm{RS}[n, m, \rho] := \left\{ \text{Evaluations of polynomials of degree } < 2^m \text{ on a domain } L \subseteq \mathbb{F} \text{ of size } n.\ \rho := \frac{2^m}{n} \right\}$$

Rate of the code

IOPP for RS



- If $f \in \mathrm{RS}[n, m, \rho]$, $\mathbf{V}$ accepts.
- If f is δ-far from $\mathrm{RS}[n, m, \rho]$, $\mathbf{V}$ accepts w.p. $\varepsilon_{\mathrm{RBR}} \leq 2^{-\lambda}$

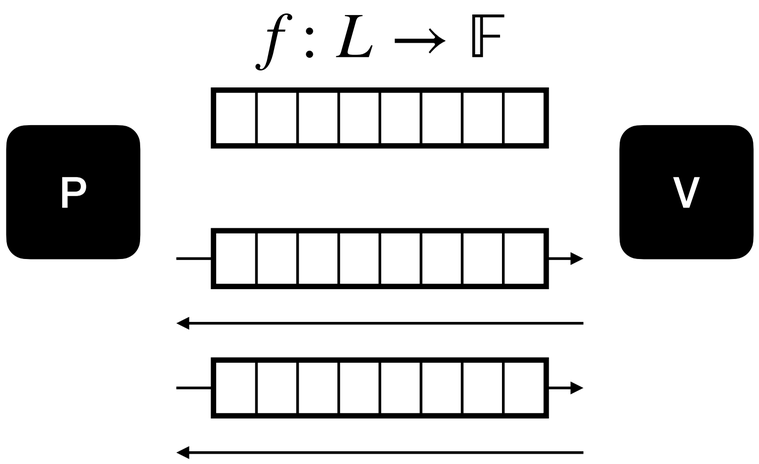
IOP of Proximity to RS codes

Convenience

$$\mathrm{RS}[n, m, \rho] := \left\{ \text{Evaluations of polynomials of degree } < 2^m \text{ on a domain } L \subseteq \mathbb{F} \text{ of size } n.\ \rho := \frac{2^m}{n} \right\}$$

Rate of the code

IOPP for RS



- If $f \in \mathrm{RS}[n, m, \rho]$, $\mathbf{V}$ accepts.
- If f is δ-far from $\mathrm{RS}[n, m, \rho]$, $\mathbf{V}$ accepts w.p. $\varepsilon_{\mathrm{RBR}} \leq 2^{-\lambda}$

Goal: minimize queries to f and other proof oracles.

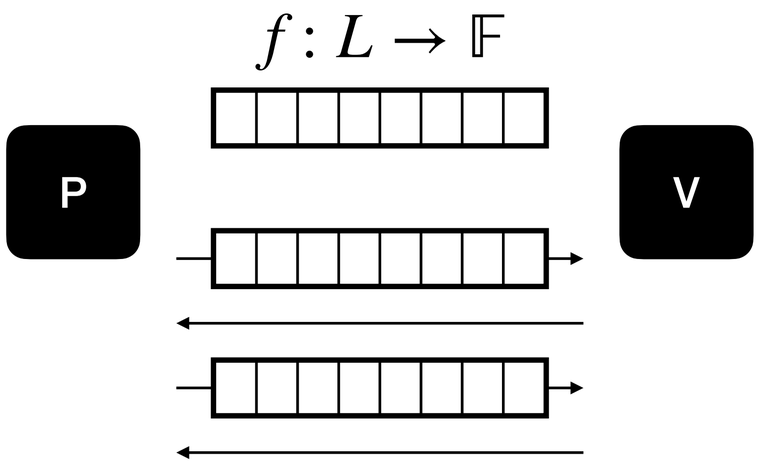
IOP of Proximity to RS codes

Convenience

$$\mathrm{RS}[n, m, \rho] := \left\{ \text{Evaluations of polynomials of degree } < 2^m \text{ on a domain } L \subseteq \mathbb{F} \text{ of size } n.\ \rho := \frac{2^m}{n} \right\}$$

Rate of the code

IOPP for RS



- If $f \in \mathrm{RS}[n, m, \rho]$, $\mathbf{V}$ accepts.
- If f is δ-far from $\mathrm{RS}[n, m, \rho]$, $\mathbf{V}$ accepts w.p. $\varepsilon_{\mathrm{RBR}} \leq 2^{-\lambda}$

Round by round, required by BCS transform.

Goal: minimize queries to f and other proof oracles.

Constrained RS tests

Constrained RS tests

What we are running:

Reed-Solomon Proximity Test on virtual function:

$$f'(x) := \frac{f(x) - y}{x - z}$$

Constrained RS tests

What we are running:

Reed-Solomon Proximity Test on virtual function:

$$f'(x) := \frac{f(x) - y}{x - z}$$

What we really want to show:

I have a polynomial $\hat{f}$ and a commitment to (an encoding of it) f such that

$$\hat{f}(z) = y$$

Constrained RS tests

What we are running:

Reed-Solomon Proximity Test on virtual function:

$$f'(x) := \frac{f(x) - y}{x - z}$$

What we really want to show:

I have a polynomial $\hat{f}$ and a commitment to (an encoding of it) f such that

$$\hat{f}(z) = y$$

Break it down as:

Test for **constrained** encoding

Constrained RS tests

What we are running:

Reed-Solomon Proximity Test on virtual function:

$$f'(x) := \frac{f(x) - y}{x - z}$$

What we really want to show:

I have a polynomial $\hat{f}$ and a commitment to (an encoding of it) f such that

$$\hat{f}(z) = y$$

Break it down as:

Test for **constrained** encoding

Quotient $f'(x) := \frac{f(x) - y}{x - z}$

Constrained RS tests

What we are running:

Reed-Solomon Proximity Test on virtual function:

$$f'(x) := \frac{f(x) - y}{x - z}$$

What we really want to show:

I have a polynomial $\hat{f}$ and a commitment to (an encoding of it) f such that

$$\hat{f}(z) = y$$

Break it down as:

Test for **constrained** encoding

Quotient $f'(x) := \frac{f(x) - y}{x - z}$

Embeds the constraint $\hat{f}(z) = y$ into f'

Constrained RS tests

What we are running:

Reed-Solomon Proximity Test on virtual function:

$$f'(x) := \frac{f(x) - y}{x - z}$$

What we really want to show:

I have a polynomial $\hat{f}$ and a commitment to (an encoding of it) f such that

$$\hat{f}(z) = y$$

Break it down as:

Test for **constrained** encoding

Quotient $f'(x) := \frac{f(x) - y}{x - z}$ + Reed—Solomon proximity test for f'

Embeds the constraint $\hat{f}(z) = y$ into f'

Constrained RS tests

What we are running:

Reed-Solomon Proximity Test on virtual function:

$$f'(x) := \frac{f(x) - y}{x - z}$$

What we really want to show:

I have a polynomial $\hat{f}$ and a commitment to (an encoding of it) f such that

$$\hat{f}(z) = y$$

Break it down as:

Test for **constrained** encoding

Quotient $f'(x) := \frac{f(x) - y}{x - z}$ + Reed—Solomon proximity test for f'

Embeds the constraint $\hat{f}(z) = y$ into f'

Can the proximity test **directly** enforce the constraint?

Constrained RS tests

What we are running:

Reed-Solomon Proximity Test on virtual function:

$$f'(x) := \frac{f(x) - y}{x - z}$$

What we really want to show:

I have a polynomial $\hat{f}$ and a commitment to (an encoding of it) f such that

$$\hat{f}(z) = y$$

Break it down as:

Test for **constrained** encoding

Quotient $f'(x) := \frac{f(x) - y}{x - z}$

+

Reed—Solomon proximity test for f'

Embeds the constraint $\hat{f}(z) = y$ into f'

Can the proximity test **directly** enforce the constraint?

Yes! IOPP for **constrained codes**

Constrained RS codes

Constrained RS codes

$$\mathrm{RS}[n, m, \rho] := \left\{ \begin{array}{l} \text{Evaluations of univariate} \\ \hat{f} \in \mathbb{F}^{<2^m}[X] \text{ on } L \end{array} \right\}$$

Constrained RS codes

$$\mathrm{RS}[n, m, \rho] := \left\{ \begin{array}{l} \text{Evaluations of univariate} \\ \hat{f} \in \mathbb{F}^{<2^m}[X] \text{ on } L \end{array} \right\}$$

Rewrite RS codes to be about multilinear polynomials:

$\mathrm{coeff}(\hat{p}) = \mathrm{coeff}(\hat{q})$

implies that

$\hat{p}(z) = \hat{q}(z, z^2, \ldots, z^{2^{m-1}})$

Constrained RS codes

$$\mathrm{RS}[n, m, \rho] := \left\{ \begin{array}{l} \text{Evaluations of univariate} \\ \hat{f} \in \mathbb{F}^{<2^m}[X] \text{ on } L \end{array} \right\}$$

$$= \left\{ \begin{array}{l} \text{Evaluations of multilinear} \\ \hat{f} \in \mathbb{F}^{\leq 1}[X_1, \ldots, X_m] \text{ on } L \end{array} \right\}$$

Rewrite RS codes to be about multilinear polynomials:

$\mathrm{coeff}(\hat{p}) = \mathrm{coeff}(\hat{q})$

implies that

$\hat{p}(z) = \hat{q}(z, z^2, \ldots, z^{2^{m-1}})$

Constrained RS codes

$$\mathrm{RS}[n, m, \rho] := \left\{ \begin{array}{l} \text{Evaluations of univariate} \\ \hat{f} \in \mathbb{F}^{<2^m}[X] \text{ on } L \end{array} \right\}$$

$$= \left\{ \begin{array}{l} \text{Evaluations of multilinear} \\ \hat{f} \in \mathbb{F}^{\leq 1}[X_1, \ldots, X_m] \text{ on } L \end{array} \right\}$$

Rewrite RS codes to be about multilinear polynomials:

$\mathrm{coeff}(\hat{p}) = \mathrm{coeff}(\hat{q})$

implies that

$\hat{p}(z) = \hat{q}(z, z^2, \ldots, z^{2^{m-1}})$

$$\mathrm{CRS}[n, m, \rho, \hat{w}, \sigma] :=$$

Constrained RS codes

$$\mathrm{RS}[n, m, \rho] := \left\{ \begin{array}{l} \text{Evaluations of univariate} \\ \hat{f} \in \mathbb{F}^{<2^m}[X] \text{ on } L \end{array} \right\}$$

$$= \left\{ \begin{array}{l} \text{Evaluations of multilinear} \\ \hat{f} \in \mathbb{F}^{\leq 1}[X_1, \ldots, X_m] \text{ on } L \end{array} \right\}$$

Rewrite RS codes to be about multilinear polynomials:

$\mathrm{coeff}(\hat{p}) = \mathrm{coeff}(\hat{q})$ implies that $\hat{p}(z) = \hat{q}(z, z^2, \ldots, z^{2^{m-1}})$

Constraint polynomial

$$\mathrm{CRS}[n, m, \rho, \hat{w}, \sigma] :=$$

Constrained RS codes

$$\mathrm{RS}[n, m, \rho] := \left\{ \begin{array}{l} \text{Evaluations of univariate} \\ \hat{f} \in \mathbb{F}^{<2^m}[X] \text{ on } L \end{array} \right\}$$

$$= \left\{ \begin{array}{l} \text{Evaluations of multilinear} \\ \hat{f} \in \mathbb{F}^{\leq 1}[X_1, \ldots, X_m] \text{ on } L \end{array} \right\}$$

Rewrite RS codes to be about multilinear polynomials:

$\mathrm{coeff}(\hat{p}) = \mathrm{coeff}(\hat{q})$ implies that $\hat{p}(z) = \hat{q}(z, z^2, \ldots, z^{2^{m-1}})$

Constraint polynomial

Value of constraint

$$\mathrm{CRS}[n, m, \rho, \hat{w}, \sigma] :=$$

Constrained RS codes

$$\mathrm{RS}[n, m, \rho] := \left\{ \begin{array}{l} \text{Evaluations of univariate} \\ \hat{f} \in \mathbb{F}^{<2^m}[X] \text{ on } L \end{array} \right\}$$

$$= \left\{ \begin{array}{l} \text{Evaluations of multilinear} \\ \hat{f} \in \mathbb{F}^{\leq 1}[X_1, \ldots, X_m] \text{ on } L \end{array} \right\}$$

Rewrite RS codes to be about multilinear polynomials:

$\mathrm{coeff}(\hat{p}) = \mathrm{coeff}(\hat{q})$ implies that $\hat{p}(z) = \hat{q}(z, z^2, \ldots, z^{2^{m-1}})$

Constraint polynomial

Value of constraint

$$\mathrm{CRS}[n, m, \rho, \hat{w}, \sigma] := \left\{ f \in \mathrm{RS}[n, m, \rho] \ : \sum_{b \in \{0,1\}^m} \hat{w}(\hat{f}(b), b) = \sigma \right\}$$

Constrained RS codes

$$\mathrm{RS}[n, m, \rho] := \left\{ \begin{array}{l} \text{Evaluations of univariate} \\ \hat{f} \in \mathbb{F}^{<2^m}[X] \text{ on } L \end{array} \right\}$$

$$= \left\{ \begin{array}{l} \text{Evaluations of multilinear} \\ \hat{f} \in \mathbb{F}^{\leq 1}[X_1, \ldots, X_m] \text{ on } L \end{array} \right\}$$

Rewrite RS codes to be about multilinear polynomials:

$\mathrm{coeff}(\hat{p}) = \mathrm{coeff}(\hat{q})$ implies that $\hat{p}(z) = \hat{q}(z, z^2, \ldots, z^{2^{m-1}})$

Constraint polynomial

Value of constraint

$$\mathrm{CRS}[n, m, \rho, \hat{w}, \sigma] := \left\{ f \in \mathrm{RS}[n, m, \rho] \ : \ \sum_{b \in \{0,1\}^m} \hat{w}(\hat{f}(b), b) = \sigma \right\}$$

$\mathrm{RS}[n, m, \rho] = \mathrm{CRS}[n, m, \rho, 0, 0]$

Constrained RS codes

$$\mathsf{RS}[n, m, \rho] := \left\{ \begin{array}{l} \text{Evaluations of univariate} \\ \hat{f} \in \mathbb{F}^{<2^m}[X] \text{ on } L \end{array} \right\}$$

$$= \left\{ \begin{array}{l} \text{Evaluations of multilinear} \\ \hat{f} \in \mathbb{F}^{\leq 1}[X_1, \ldots, X_m] \text{ on } L \end{array} \right\}$$

Rewrite RS codes to be about multilinear polynomials:

$\text{coeff}(\hat{p}) = \text{coeff}(\hat{q})$ implies that $\hat{p}(z) = \hat{q}(z, z^2, \ldots, z^{2^{m-1}})$

Constraint polynomial

Value of constraint

$$\mathsf{CRS}[n, m, \rho, \hat{w}, \sigma] := \left\{ f \in \mathsf{RS}[n, m, \rho] \ : \ \sum_{b \in \{0,1\}^m} \hat{w}(\hat{f}(b), b) = \sigma \right\}$$

If $\hat{w} = Z \cdot \text{eq}(\mathbf{X}, \mathbf{r})$ we recover multilinear polynomial evaluation

$\mathsf{RS}[n, m, \rho] = \mathsf{CRS}[n, m, \rho, 0, 0]$

Constrained RS codes

$$\mathsf{RS}[n, m, \rho] := \left\{ \begin{array}{l} \text{Evaluations of univariate} \\ \hat{f} \in \mathbb{F}^{<2^m}[X] \text{ on } L \end{array} \right\}$$

$$= \left\{ \begin{array}{l} \text{Evaluations of multilinear} \\ \hat{f} \in \mathbb{F}^{\leq 1}[X_1, \ldots, X_m] \text{ on } L \end{array} \right\}$$

Rewrite RS codes to be about multilinear polynomials:

$\text{coeff}(\hat{p}) = \text{coeff}(\hat{q})$ implies that $\hat{p}(z) = \hat{q}(z, z^2, \ldots, z^{2^{m-1}})$

Constraint polynomial

Value of constraint

$$\mathsf{CRS}[n, m, \rho, \hat{w}, \sigma] := \left\{ f \in \mathsf{RS}[n, m, \rho] \ : \ \sum_{b \in \{0,1\}^m} \hat{w}(\hat{f}(b), b) = \sigma \right\}$$

If $\hat{w} = Z \cdot \text{eq}(\mathbf{X}, \mathbf{r})$ we recover multilinear polynomial evaluation

$\mathsf{RS}[n, m, \rho] = \mathsf{CRS}[n, m, \rho, 0, 0]$

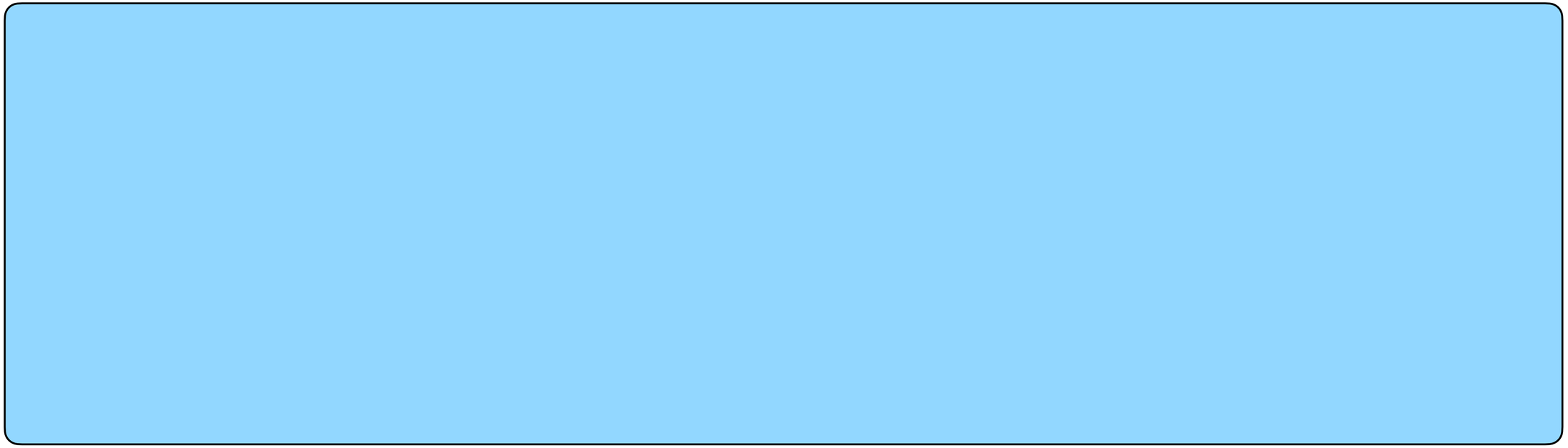
We test **proximity** to CRS!

Our results

WHIR

$k > 1$ is a folding parameter

A constrained Reed-Solomon proximity test



WHIR 

$k > 1$ is a folding parameter

A <u>constrained</u> Reed-Solomon proximity test

Rounds: $O(m)$

Alphabet: $\mathbb{F}^{2^k}$

Proof length: $O(n/2^k)$

Verifier time: $O\left(q_{\text{WHIR}} \cdot (2^k + m)\right)$

WHIR

$k > 1$ is a folding parameter

A constrained Reed-Solomon proximity test

Rounds: $O(m)$

Alphabet: $\mathbb{F}^{2^k}$

Proof length: $O(n/2^k)$

Verifier time: $O\left(q_{\text{WHIR}} \cdot (2^k + m)\right)$

Query complexity:

$q_{\text{WHIR}} =$

WHIR

$k > 1$ is a folding parameter

A <u>constrained</u> Reed-Solomon proximity test

Rounds: $O(m)$

Alphabet: $\mathbb{F}^{2^k}$

Proof length: $O(n/2^k)$

Verifier time: $O\left(q_{\text{WHIR}} \cdot (2^k + m)\right)$

Query complexity:

$q_{\text{WHIR}} =$

$\lambda \gg m$

WHIR

$k > 1$ is a folding parameter

A constrained Reed-Solomon proximity test

Rounds: $O(m)$

Alphabet: $\mathbb{F}^{2^k}$

Proof length: $O(n/2^k)$

Verifier time: $O\left(q_{\text{WHIR}} \cdot (2^k + m)\right)$

Query complexity:

$$q_{\text{WHIR}} = O\left(\frac{\lambda}{k} \cdot \log m\right) =$$

$\lambda \gg m$

WHIR

$k > 1$ is a folding parameter

A constrained Reed-Solomon proximity test

Rounds: $O(m)$

Alphabet: $\mathbb{F}^{2^k}$

Proof length: $O(n/2^k)$

Verifier time: $O\left(q_{\text{WHIR}} \cdot (2^k + m)\right)$

Query complexity:

$$q_{\text{WHIR}} = O\left(\frac{\lambda}{k} \cdot \log m\right) =$$

$k \approx \log m$

$\lambda \gg m$

WHIR

$k > 1$ is a folding parameter

A constrained Reed-Solomon proximity test

Rounds: $O(m)$

Alphabet: $\mathbb{F}^{2^k}$

Proof length: $O(n/2^k)$

Verifier time: $O\left(q_{\text{WHIR}} \cdot (2^k + m)\right)$

Query complexity:

$$q_{\text{WHIR}} = O\left(\frac{\lambda}{k} \cdot \log m\right) = O(\lambda)$$

$k \approx \log m$

$\lambda \gg m$

WHIR 🌪️

A <u>constrained</u> Reed-Solomon proximity test

$k > 1$ is a folding parameter

Rounds: $O(m)$

Alphabet: $\mathbb{F}^{2^k}$

Proof length: $O(n/2^k)$

Verifier time: $O\left(q_{\text{WHIR}} \cdot (2^k + m)\right)$

Query complexity:

$$q_{\text{WHIR}} = O\left(\frac{\lambda}{k} \cdot \log m\right) = O(\lambda)$$

$k \approx \log m$

$\lambda \gg m$

Σ-IOP + CRS IOPP (WHIR 🌪️) = IOP

WHIR

$k > 1$ is a folding parameter

A constrained Reed-Solomon proximity test

Rounds: $O(m)$

Alphabet: $\mathbb{F}^{2^k}$

Proof length: $O(n/2^k)$

Verifier time: $O\left(q_{\text{WHIR}} \cdot (2^k + m)\right)$

Query complexity:

$$q_{\text{WHIR}} = O\left(\frac{\lambda}{k} \cdot \log m\right) = O(\lambda)$$

$k \approx \log m$

$\lambda \gg m$

Σ-IOP + CRS IOPP (WHIR) = IOP

High soundness analogue of RS PIOP compiler (w/o quotients)

WHIR

$k > 1$ is a folding parameter

A constrained Reed-Solomon proximity test

Rounds: $O(m)$

Alphabet: $\mathbb{F}^{2^k}$

Proof length: $O(n/2^k)$

Verifier time: $O\left(q_{\text{WHIR}} \cdot (2^k + m)\right)$

Query complexity:

$$q_{\text{WHIR}} = O\left(\frac{\lambda}{k} \cdot \log m\right) = O(\lambda)$$

$k \approx \log m$

$\lambda \gg m$

Σ-IOP + CRS IOPP (WHIR) = IOP

High soundness analogue of RS PIOP compiler (w/o quotients)

Implementation available at WizardOfMenlo/whir

Comparison with prior work

	Queries	Verifier Time	Alphabet
BaseFold	$q_{\text{BF}} = O(\lambda \cdot m)$	$O(q_{\text{BF}})$	$\mathbb{F}^2$
FRI	$q_{\text{FRI}} = O\left(\frac{\lambda}{k} \cdot m\right)$	$O(q_{\text{FRI}} \cdot 2^k)$	$\mathbb{F}^{2^k}$
STIR	$q_{\text{STIR}} = O\left(\frac{\lambda}{k} \cdot \log m\right)$	$O(q_{\text{STIR}} \cdot 2^k + \lambda^2 \cdot 2^k)$	$\mathbb{F}^{2^k}$
WHIR	$q_{\text{WHIR}} = O\left(\frac{\lambda}{k} \cdot \log m\right)$	$O(q_{\text{WHIR}} \cdot (2^k + m))$	$\mathbb{F}^{2^k}$

Comparison with prior work

	Queries	Verifier Time	Alphabet
BaseFold	$q_{\text{BF}} = O(\lambda \cdot m)$	$O(q_{\text{BF}})$	$\mathbb{F}^2$
FRI	$q_{\text{FRI}} = O\left(\frac{\lambda}{k} \cdot m\right)$	$O(q_{\text{FRI}} \cdot 2^k)$	$\mathbb{F}^{2^k}$
STIR	$q_{\text{STIR}} = O\left(\frac{\lambda}{k} \cdot \log m\right)$	$O(q_{\text{STIR}} \cdot 2^k + \lambda^2 \cdot 2^k)$	$\mathbb{F}^{2^k}$
WHIR	$q_{\text{WHIR}} = O\left(\frac{\lambda}{k} \cdot \log m\right)$	$O(q_{\text{WHIR}} \cdot (2^k + m))$	$\mathbb{F}^{2^k}$

When $k \approx \log m$
$q_{\text{WHIR}} = O(\lambda)$
OPTIMAL

When $k \approx \log m$
$O(q_{\text{WHIR}} \cdot |\Sigma|)$
OPTIMAL

Comparison with prior work

	Queries	Verifier Time	Alphabet
BaseFold	$q_{\text{BF}} = O(\lambda \cdot m)$	$O(q_{\text{BF}})$	$\mathbb{F}^2$
FRI	$q_{\text{FRI}} = O\left(\frac{\lambda}{k} \cdot m\right)$	$O(q_{\text{FRI}} \cdot 2^k)$	$\mathbb{F}^{2^k}$
STIR	$q_{\text{STIR}} = O\left(\frac{\lambda}{k} \cdot \log m\right)$	$O(q_{\text{STIR}} \cdot 2^k + \lambda^2 \cdot 2^k)$	$\mathbb{F}^{2^k}$
WHIR	$q_{\text{WHIR}} = O\left(\frac{\lambda}{k} \cdot \log m\right)$	$O(q_{\text{WHIR}} \cdot (2^k + m))$	$\mathbb{F}^{2^k}$

When $k \approx \log m$
$q_{\text{WHIR}} = O(\lambda)$
OPTIMAL

When $k \approx \log m$
$O(q_{\text{WHIR}} \cdot |\Sigma|)$
OPTIMAL

When $k \approx \log m$
$\Sigma = \mathbb{F}^m$
Improving is an open question

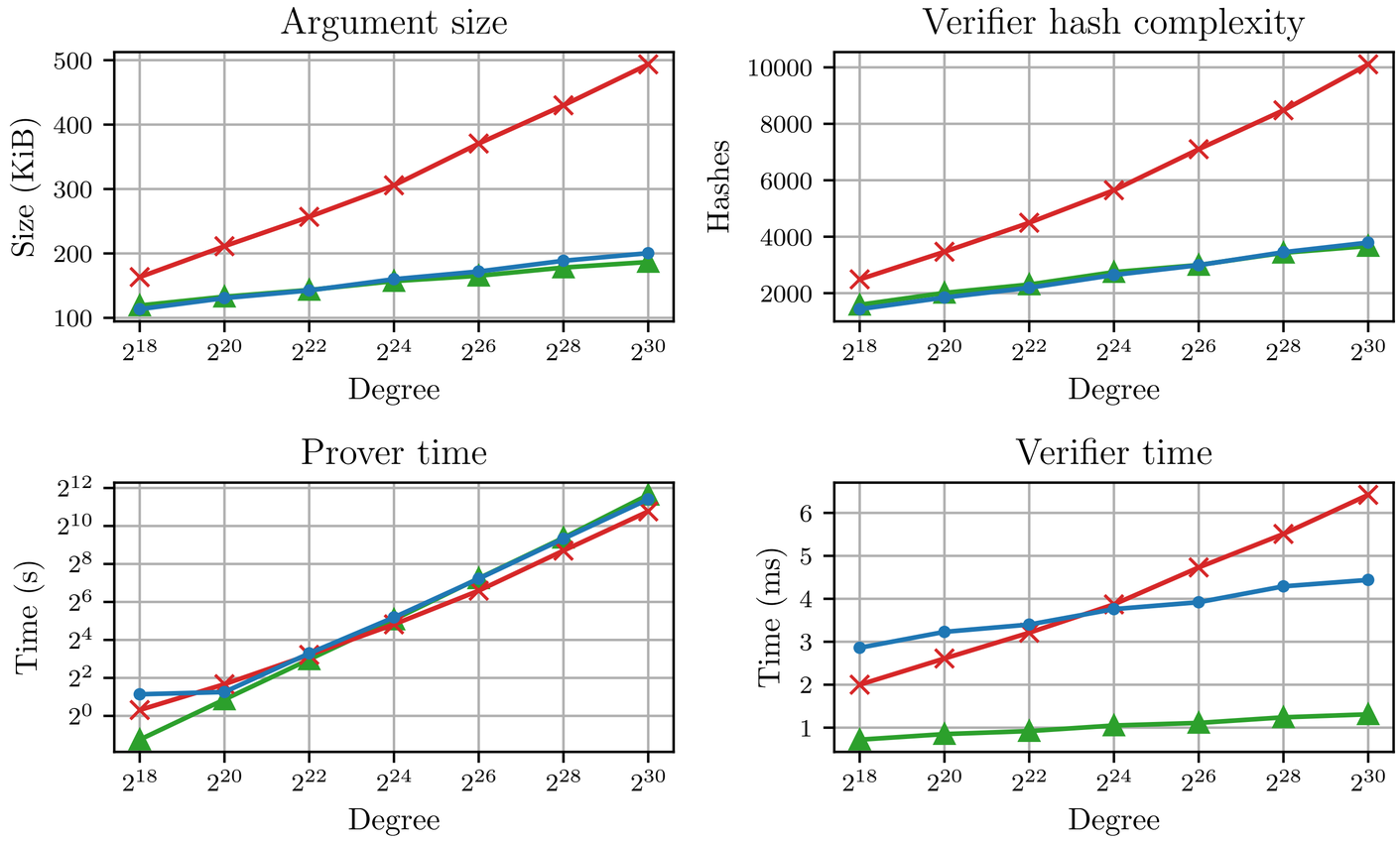
Comparison with FRI and STIR

Note: prover time graph is now outdated due to new optimizations discovered

128-bits security level.

$\lambda = 106 + 22$ bits of PoW + “list-decoding” assumptions.

Rate of the code $\rho = 1/2$



FRI, STIR, WHIR

Comparison with FRI and STIR

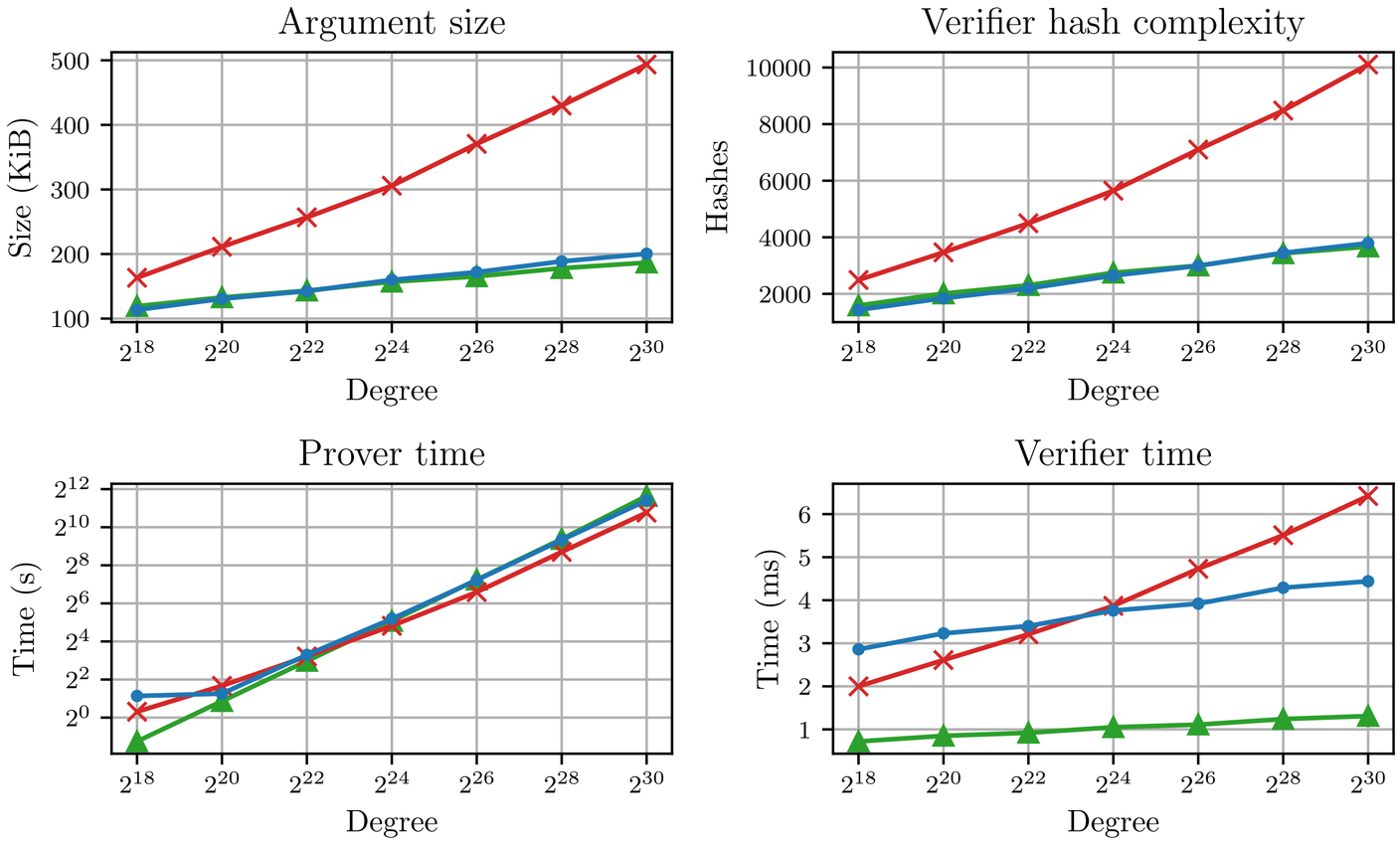
Note: prover time graph is now outdated due to new optimizations discovered

128-bits security level.

$\lambda = 106 + 22$ bits of PoW + “list-decoding” assumptions.

2^24 coeffs rate = 1/4	FRI	WHIR
Size (KiB)	177	110
Verifier time	2.4ms	700µs

Rate of the code $\rho = 1/2$



FRI, STIR, WHIR

Comparison with FRI and STIR

Note: prover time graph is now outdated due to new optimizations discovered

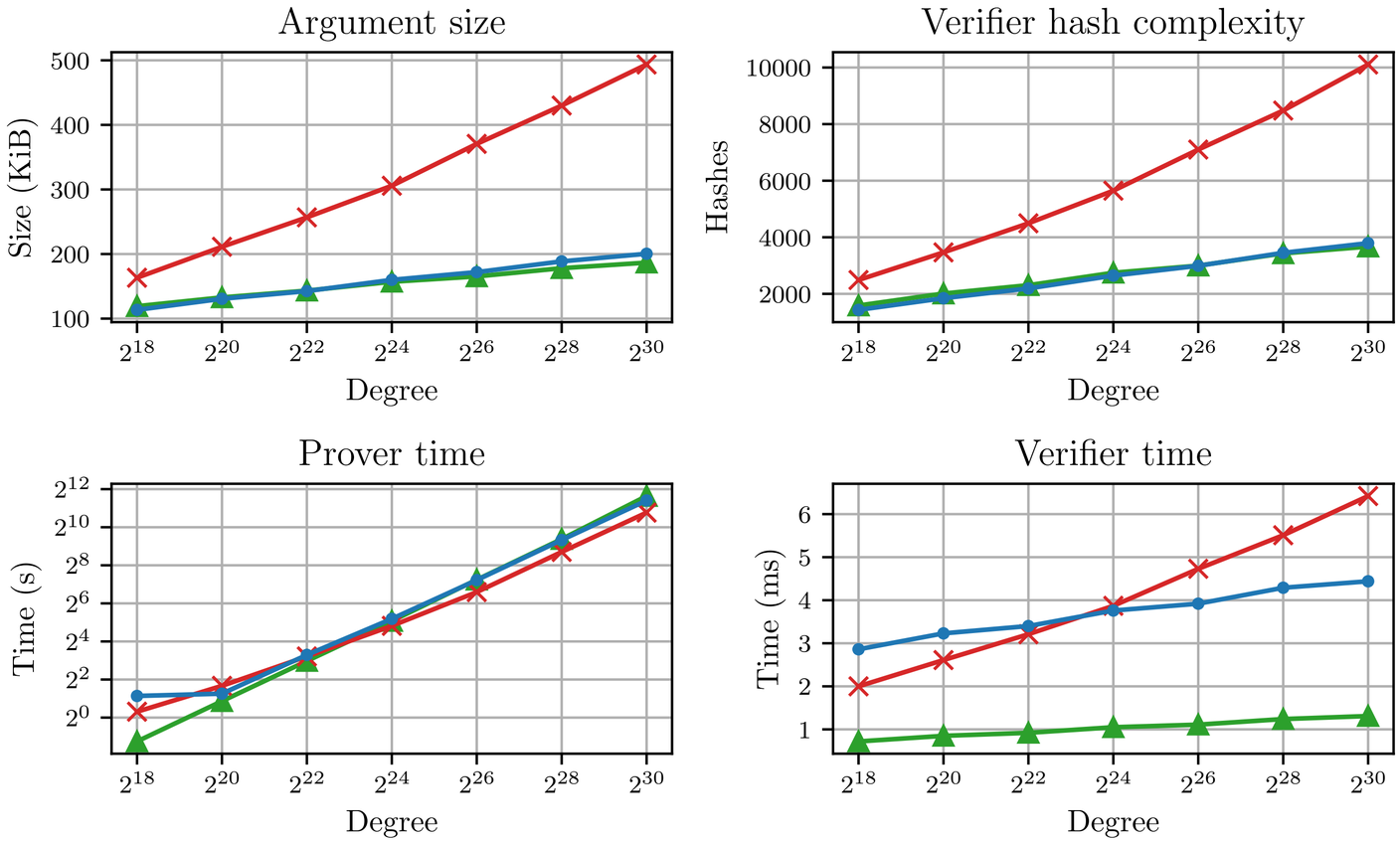
128-bits security level.

$\lambda = 106 + 22$ bits of PoW + “list-decoding” assumptions.

2^24 coeffs rate = 1/4	FRI	WHIR
Size (KiB)	177	110
Verifier time	2.4ms	700µs

2^30 coeffs rate = 1/2	FRI	WHIR
Size (KiB)	494	187
Verifier time	4.4ms	1.3ms

Rate of the code $\rho = 1/2$



FRI, STIR, WHIR

Super fast verifier

Super fast verifier

- The WHIR verifier typically runs in a few hundred **MICRO**-seconds.

Super fast verifier

- The WHIR verifier typically runs in a few hundred **MICRO**-seconds.
- Other verifiers require several **MILLI**-seconds (and more).

Super fast verifier

- The WHIR verifier typically runs in a few hundred **MICRO**-seconds.
- Other verifiers require several **MILLI**-seconds (and more).
- While maintaining **state-of-the-art** prover time & argument size

Super fast verifier

- The WHIR verifier typically runs in a few hundred **MICRO**-seconds.
- Other verifiers require several **MILLI**-seconds (and more).
- While maintaining **state-of-the-art** prover time & argument size

Verifier time (ms)	**Brakedown**	**Ligero**	**Greyhound**	**Hyrax**	**PST**	**KZG**	**WHIR[1/2]**	**WHIR[1/16]**
$\lambda = 100$	3500	733	-	100	7.81	2.42	0.61	0.29
$\lambda = 128$	3680	750	130	151	9.92	3.66	1.4	0.6

Super fast verifier

- The WHIR verifier typically runs in a few hundred **MICRO**-seconds.
- Other verifiers require several **MILLI**-seconds (and more).
- While maintaining **state-of-the-art** prover time & argument size

Verifier time (ms)	**Brakedown**	**Ligero**	**Greyhound**	**Hyrax**	**PST**	**KZG**	**WHIR[1/2]**	**WHIR[1/16]**
$\lambda = 100$	3500	733	-	100	7.81	2.42	0.61	0.29
$\lambda = 128$	3680	750	130	151	9.92	3.66	1.4	0.6

WHIR[ρ] denotes WHIR with rate ρ

Super fast verifier

- The WHIR verifier typically runs in a few hundred **MICRO**-seconds.
- Other verifiers require several **MILLI**-seconds (and more).
- While maintaining **state-of-the-art** prover time & argument size

Schemes with **trusted** setup using pairings!

Verifier time (ms)	**Brakedown**	**Ligero**	**Greyhound**	**Hyrax**	**PST**	**KZG**	**WHIR[1/2]**	**WHIR[1/16]**
$\lambda = 100$	3500	733	-	100	7.81	2.42	0.61	0.29
$\lambda = 128$	3680	750	130	151	9.92	3.66	1.4	0.6

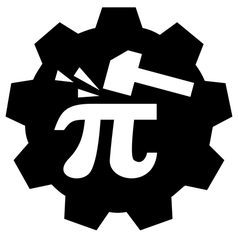
WHIR[ρ] denotes WHIR with rate ρ

Practical adoption

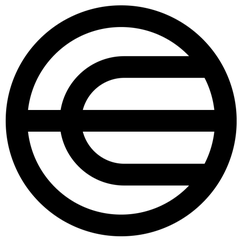
Practical adoption

Implementation available
@ WizardOfMenlo/whir

Practical adoption

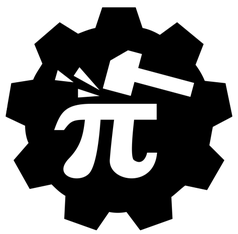


Implementation available
@ WizardOfMenlo/whir

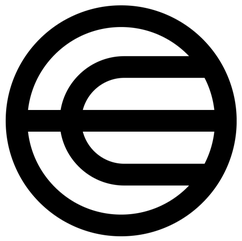


World client-side prover
@ worldfnd/ProveKit

Practical adoption



Implementation available
@ WizardOfMenlo/whir



World client-side prover
@ worldfnd/ProveKit

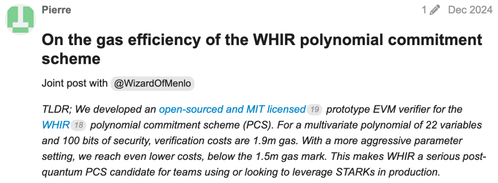
To be deployed to
26M+ users!

Practical adoption

Implementation available
@ WizardOfMenlo/whir

World client-side prover
@ worldfnd/ProveKit

To be deployed to
26M+ users!



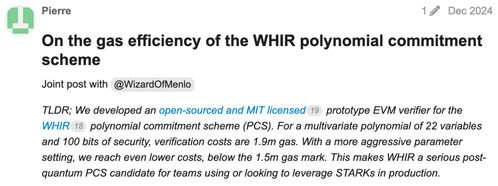
Solidity verifier implementation
@ privacy-scaling-explorations/sol-whir

Practical adoption

Implementation available
@ WizardOfMenlo/whir

World client-side prover
@ worldfnd/ProveKit

To be deployed to 26M+ users!

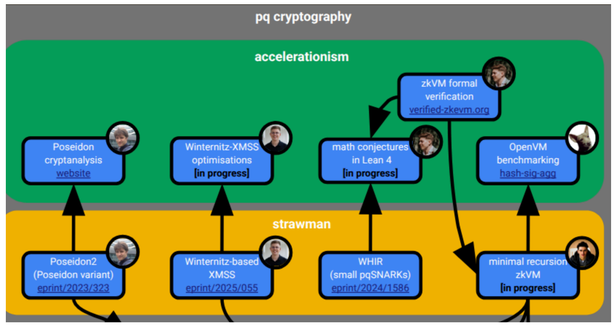


~1M gas for verification, competitive with pre-quantum alternatives

Solidity verifier implementation
@ privacy-scaling-explorations/sol-whir

Practical adoption

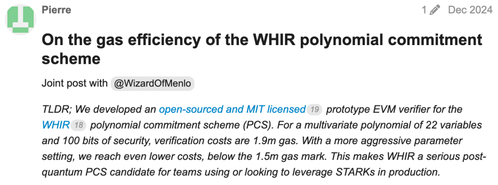
Ethereum pq transition



Implementation available
@ WizardOfMenlo/whir

World client-side prover
@ worldfnd/ProveKit

To be deployed to 26M+ users!



Pierre 1 Dec 2024

On the gas efficiency of the WHIR polynomial commitment scheme

Joint post with @WizardOfMenlo

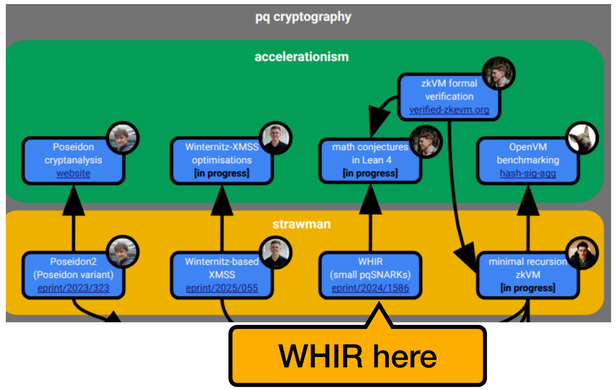
TLDR; We developed an open-sourced and MIT licensed 19 prototype EVM verifier for the WHIR 18 polynomial commitment scheme (PCS). For a multivariate polynomial of 22 variables and 100 bits of security, verification costs are 1.9m gas. With a more aggressive parameter setting, we reach even lower costs, below the 1.5m gas mark. This makes WHIR a serious post-quantum PCS candidate for teams using or looking to leverage STARKs in production.

~1M gas for verification, competitive with pre-quantum alternatives

Solidity verifier implementation
@ privacy-scaling-explorations/sol-whir

Practical adoption

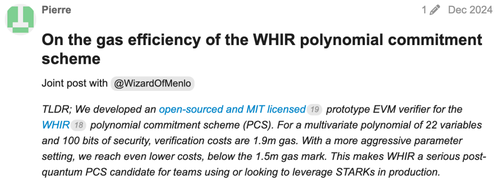
Ethereum pq transition



Implementation available
@ WizardOfMenlo/whir

World client-side prover
@ worldfnd/ProveKit

To be deployed to 26M+ users!



Pierre 1 Dec 2024

On the gas efficiency of the WHIR polynomial commitment scheme

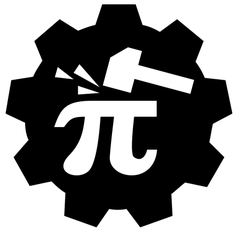
Joint post with @WizardOfMenlo

TLDR; We developed an open-sourced and MIT licensed 19 prototype EVM verifier for the WHIR 18 polynomial commitment scheme (PCS). For a multivariate polynomial of 22 variables and 100 bits of security, verification costs are 1.9m gas. With a more aggressive parameter setting, we reach even lower costs, below the 1.5m gas mark. This makes WHIR a serious post-quantum PCS candidate for teams using or looking to leverage STARKs in production.

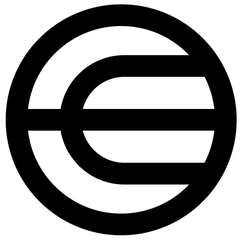
~1M gas for verification, competitive with pre-quantum alternatives

Solidity verifier implementation
@ privacy-scaling-explorations/sol-whir

Practical adoption



Implementation available
@ WizardOfMenlo/whir



World client-side prover
@ worldfnd/ProveKit

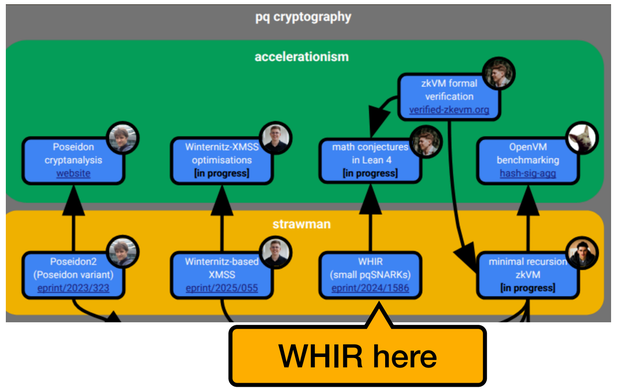
To be deployed to 26M+ users!

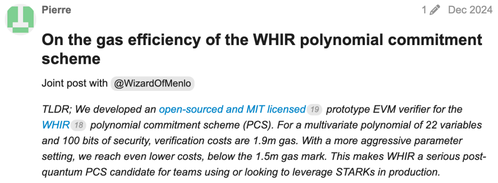
Plonky3

POWERED BY polygon

Plonky3 implementation
@ tcoratger/whir-p3

Ethereum pq transition





Pierre 1 Dec 2024

On the gas efficiency of the WHIR polynomial commitment scheme

Joint post with @WizardOfMenlo

TLDR; We developed an open-sourced and MIT licensed 19 prototype EVM verifier for the WHIR 18 polynomial commitment scheme (PCS). For a multivariate polynomial of 22 variables and 100 bits of security, verification costs are 1.9m gas. With a more aggressive parameter setting, we reach even lower costs, below the 1.5m gas mark. This makes WHIR a serious post-quantum PCS candidate for teams using or looking to leverage STARKs in production.

~1M gas for verification, competitive with pre-quantum alternatives

Solidity verifier implementation
@ privacy-scaling-explorations/sol-whir

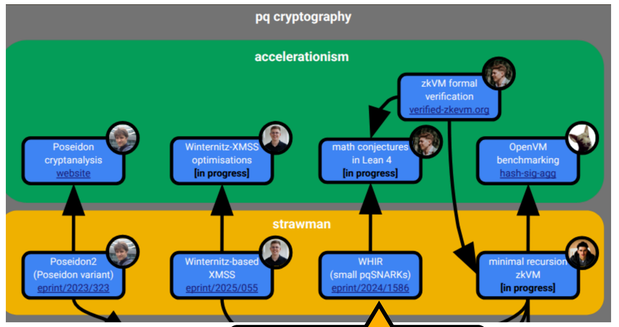
Practical adoption

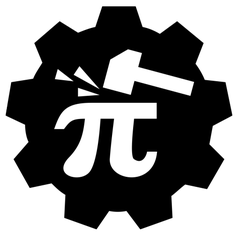
Plonky3

POWERED BY polygon

Plonky3 implementation
@ tcoratger/whir-p3

Ethereum pq transition





Implementation available
@ WizardOfMenlo/whir

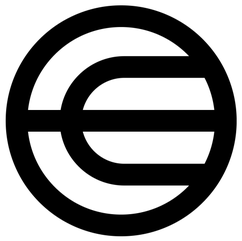
Whirlaway

A hash-based SNARK with lightweight proofs, powered by the Whir Polynomial Commitment Scheme.

Specifications

- **Arithmetization**: AIR (Algebraic Intermediate Representation) with preprocessed columns
- **Security level**: 128 bits (without conjectures), presumably post-quantum (hash-based protocol)
- **Ingredients**: WHIR + Ring-Switching + Sumcheck + Univariate Skip

Hash-based SNARK
based on WHIR
@ TomWambsgans/Whirlaway



World client-side prover
@ worldfnd/ProveKit

To be deployed to
26M+ users!

Pierre 1 Dec 2024

On the gas efficiency of the WHIR polynomial commitment scheme

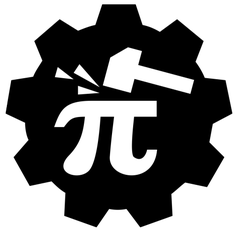
Joint post with @WizardOfMenlo

TLDR; We developed an open-sourced and MIT licensed [19] prototype EVM verifier for the WHIR [18] polynomial commitment scheme (PCS). For a multivariate polynomial of 22 variables and 100 bits of security, verification costs are 1.9m gas. With a more aggressive parameter setting, we reach even lower costs, below the 1.5m gas mark. This makes WHIR a serious post-quantum PCS candidate for teams using or looking to leverage STARKs in production.

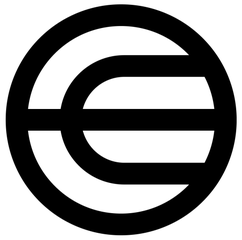
~1M gas for verification,
competitive with pre-quantum
alternatives

Solidity verifier implementation
@ privacy-scaling-explorations/sol-whir

Practical adoption



Implementation available
@ WizardOfMenlo/whir



World client-side prover
@ worldfnd/ProveKit

To be deployed to 26M+ users!

Plonky3

POWERED BY polygon

Plonky3 implementation
@ tcoratger/whir-p3

Whirlaway

A hash-based SNARK with lightweight proofs, powered by the Whir Polynomial Commitment Scheme.

Specifications

- **Arithmetization**: AIR (Algebraic Intermediate Representation) with preprocessed columns
- **Security level**: 128 bits (without conjectures), presumably post-quantum (hash-based protocol)
- **Ingredients**: WHIR + Ring-Switching + Sumcheck + Univariate Skip

Hash-based SNARK
based on WHIR
@ TomWambsgans/Whirlaway

with CUDA backend!

Pierre 1 Dec 2024

On the gas efficiency of the WHIR polynomial commitment scheme

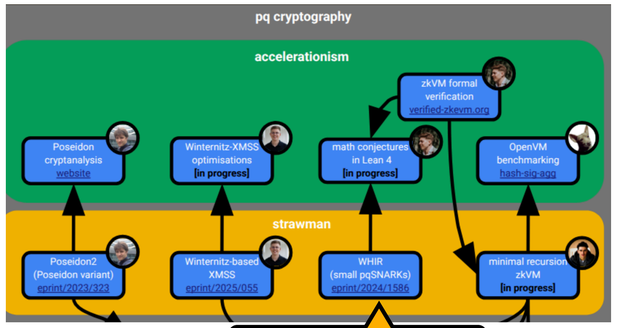
Joint post with @WizardOfMenlo

TLDR; We developed an open-sourced and MIT licensed 19 prototype EVM verifier for the WHIR 18 polynomial commitment scheme (PCS). For a multivariate polynomial of 22 variables and 100 bits of security, verification costs are 1.9m gas. With a more aggressive parameter setting, we reach even lower costs, below the 1.5m gas mark. This makes WHIR a serious post-quantum PCS candidate for teams using or looking to leverage STARKs in production.

~1M gas for verification, competitive with pre-quantum alternatives

Solidity verifier implementation
@ privacy-scaling-explorations/sol-whir

Ethereum pq transition



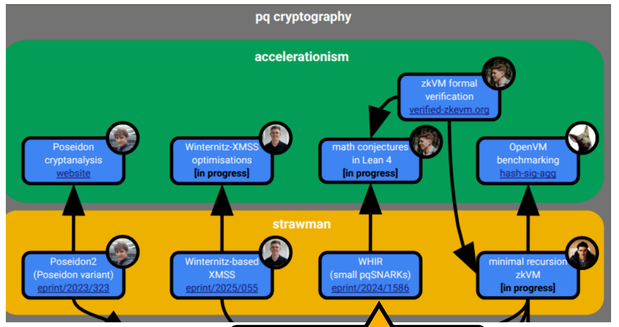
Practical adoption

Plonky3

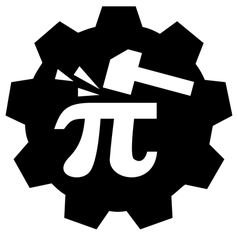
POWERED BY polygon

Plonky3 implementation
@ tcoratger/whir-p3

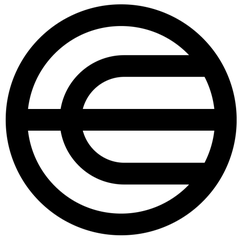
Ethereum pq transition



WHIR here



Implementation available
@ WizardOfMenlo/whir



World client-side prover
@ worldfnd/ProveKit

To be deployed to 26M+ users!

Whirlaway

A hash-based SNARK with lightweight proofs, powered by the Whir Polynomial Commitment Scheme.

Specifications

- **Arithmetization**: AIR (Algebraic Intermediate Representation) with preprocessed columns
- **Security level**: 128 bits (without conjectures), presumably post-quantum (hash-based protocol)
- **Ingredients**: WHIR + Ring-Switching + Sumcheck + Univariate Skip

Hash-based SNARK
based on WHIR
@ TomWambsgans/Whirlaway

with CUDA backend!

Pierre — 1 Dec 2024

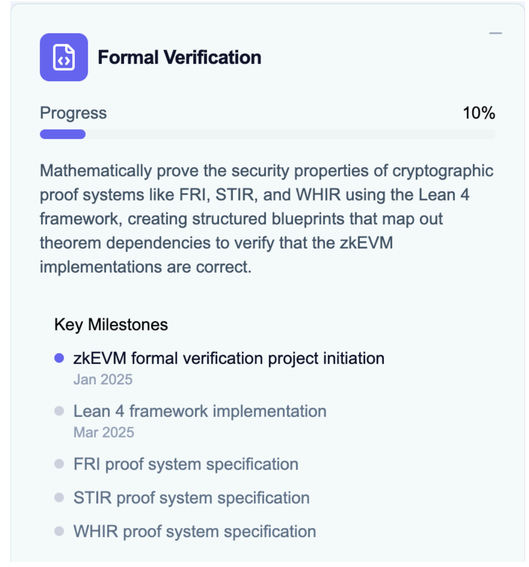
On the gas efficiency of the WHIR polynomial commitment scheme

Joint post with @WizardOfMenlo

TLDR; We developed an open-sourced and MIT licensed 19 prototype EVM verifier for the WHIR 18 polynomial commitment scheme (PCS). For a multivariate polynomial of 22 variables and 100 bits of security, verification costs are 1.9m gas. With a more aggressive parameter setting, we reach even lower costs, below the 1.5m gas mark. This makes WHIR a serious post-quantum PCS candidate for teams using or looking to leverage STARKs in production.

~1M gas for verification, competitive with pre-quantum alternatives

Solidity verifier implementation
@ privacy-scaling-explorations/sol-whir



Formal verification effort

Conclusion

Summary

Summary

WHIR 🌪️: a new IOPP for CRS codes.

Summary

WHIR 🌪️: a new IOPP for CRS codes.

Query complexity:

$$O\left(\frac{\lambda}{k} \cdot \log m\right)$$

Verifier complexity:

$$O(q_{\text{WHIR}} \cdot (2^k + m))$$

Summary

WHIR 🌪️: a new IOPP for CRS codes.

Query complexity:

$$O\left(\frac{\lambda}{k} \cdot \log m\right)$$

Verifier complexity:

$$O(q_{\text{WHIR}} \cdot (2^k + m))$$

- **State-of-the-art** prover time, argument size and hash complexity

Summary

WHIR 🌪️: a new IOPP for CRS codes.

Query complexity:

$$O\left(\frac{\lambda}{k} \cdot \log m\right)$$

Verifier complexity:

$$O(q_{\text{WHIR}} \cdot (2^k + m))$$

- **State-of-the-art** prover time, argument size and hash complexity
- **Fastest** verification of any PCS (including trusted setups!)

Summary

WHIR 🌪️: a new IOPP for CRS codes.

Query complexity:

$$O\left(\frac{\lambda}{k} \cdot \log m\right)$$

Verifier complexity:

$$O(q_{\text{WHIR}} \cdot (2^k + m))$$

- **State-of-the-art** prover time, argument size and hash complexity
- **Fastest** verification of any PCS (including trusted setups!)
- **Rapid** practical adoption

Summary

WHIR 🌪️: a new IOPP for CRS codes.

Query complexity:

$$O\left(\frac{\lambda}{k} \cdot \log m\right)$$

Verifier complexity:

$$O(q_{\text{WHIR}} \cdot (2^k + m))$$

- **State-of-the-art** prover time, argument size and hash complexity
- **Fastest** verification of any PCS (including trusted setups!)
- **Rapid** practical adoption
- Enables high-soundness compilation for **Σ-IOP**

Σ-IOP + CRS IOPP (WHIR 🌪️) = IOP

Summary

WHIR 🌪️: a new IOPP for CRS codes.

Query complexity:

$$O\left(\frac{\lambda}{k} \cdot \log m\right)$$

Verifier complexity:

$$O(q_{\text{WHIR}} \cdot (2^k + m))$$

- **State-of-the-art** prover time, argument size and hash complexity
- **Fastest** verification of any PCS (including trusted setups!)
- **Rapid** practical adoption
- Enables high-soundness compilation for **Σ-IOP**

Σ-IOP + CRS IOPP (WHIR 🌪️) = IOP

Open question:
Can argument size be improved at the **same prover cost?**

Extra slides

Techniques

FRI & STIR Folding

Reduce RS$[n, m, \rho]$ **to** RS$[n/2^k, m - k, \rho]$

(Think $k = 4$)

FRI & STIR Folding

Reduce $\mathrm{RS}[n, m, \rho]$ **to** $\mathrm{RS}[n/2^k, m - k, \rho]$

(Think $k = 4$)

Unchanged!

FRI & STIR Folding

Reduce RS$[n, m, \rho]$ **to** RS$[n/2^k, m - k, \rho]$

(Think $k = 4$)

Unchanged!

$$f : L \to \mathbb{F}$$

P

V

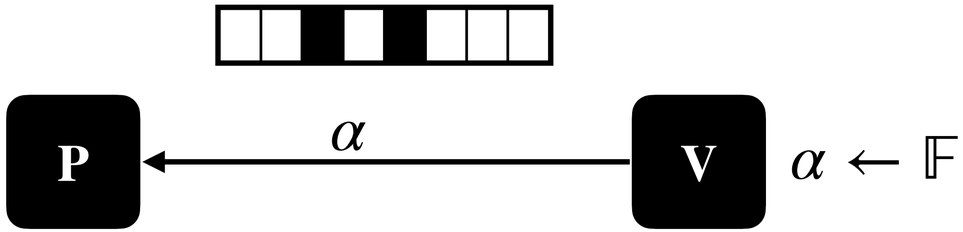
FRI & STIR Folding

Reduce RS$[n, m, \rho]$ **to** RS$[n/2^k, m - k, \rho]$

(Think $k = 4$)

Unchanged!

$f : L \to \mathbb{F}$

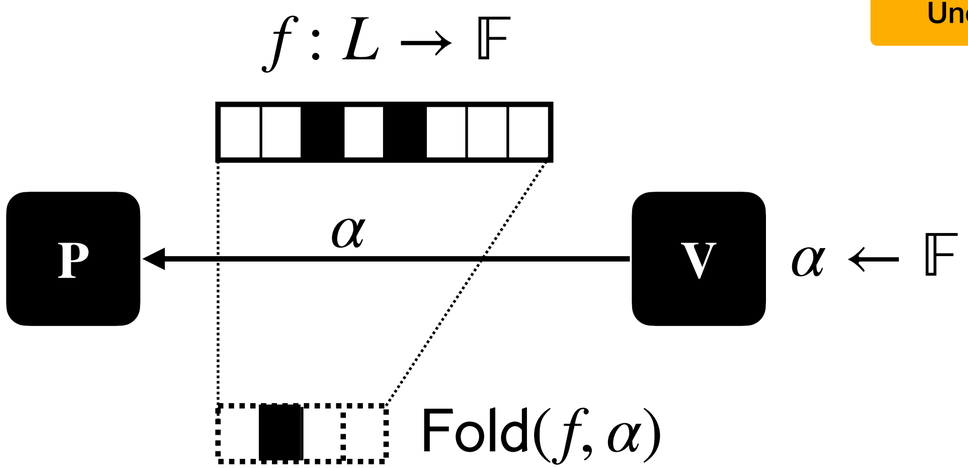


FRI & STIR Folding

Reduce RS$[n, m, \rho]$ **to** RS$[n/2^k, m - k, \rho]$

(Think $k = 4$)

Unchanged!

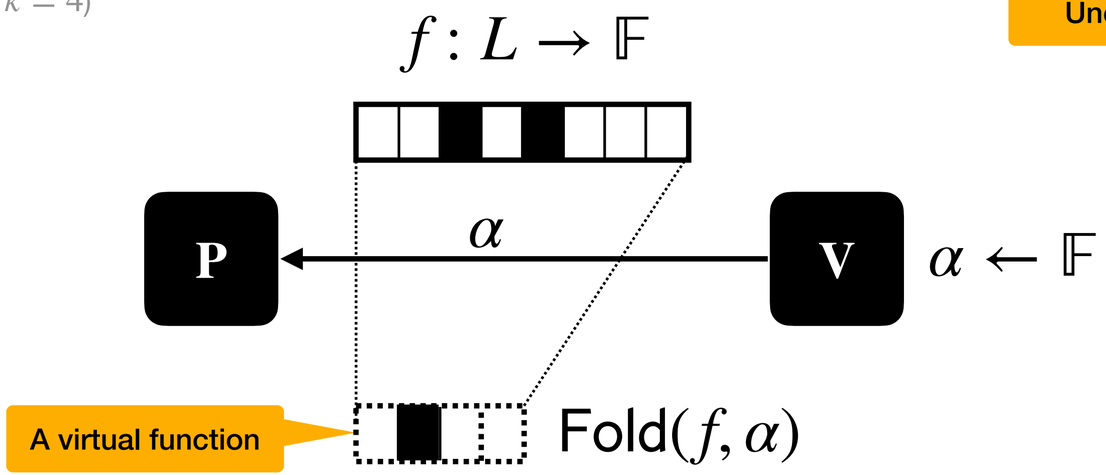


FRI & STIR Folding

Reduce RS$[n, m, \rho]$ **to** RS$[n/2^k, m - k, \rho]$

(Think $k = 4$)

Unchanged!

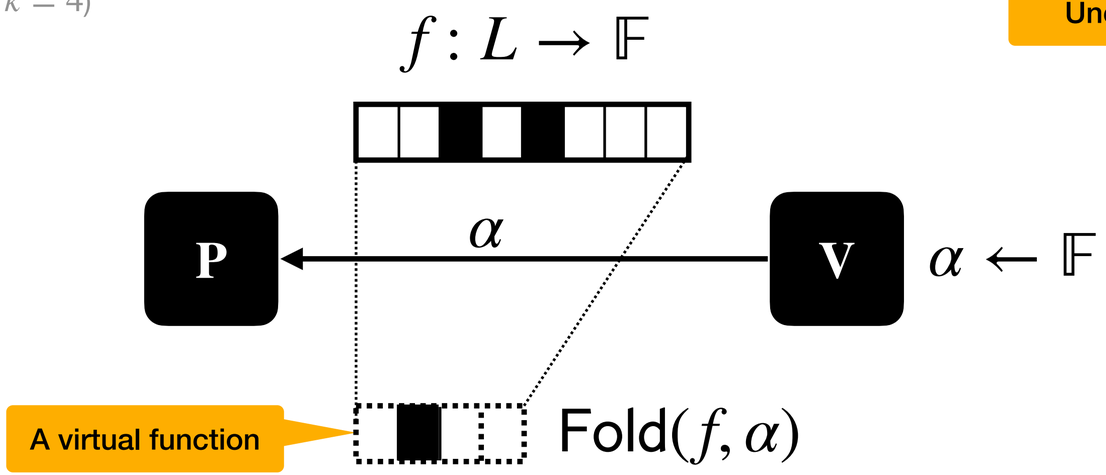


FRI & STIR Folding

Reduce $\mathrm{RS}[n, m, \rho]$ **to** $\mathrm{RS}[n/2^k, m-k, \rho]$

(Think $k = 4$)

Unchanged!



How? Inspiration from FFTs, for $k = 1$:

$$\mathrm{Fold}(f, \alpha) := f_{\text{odd}} + \alpha \cdot f_{\text{even}}$$

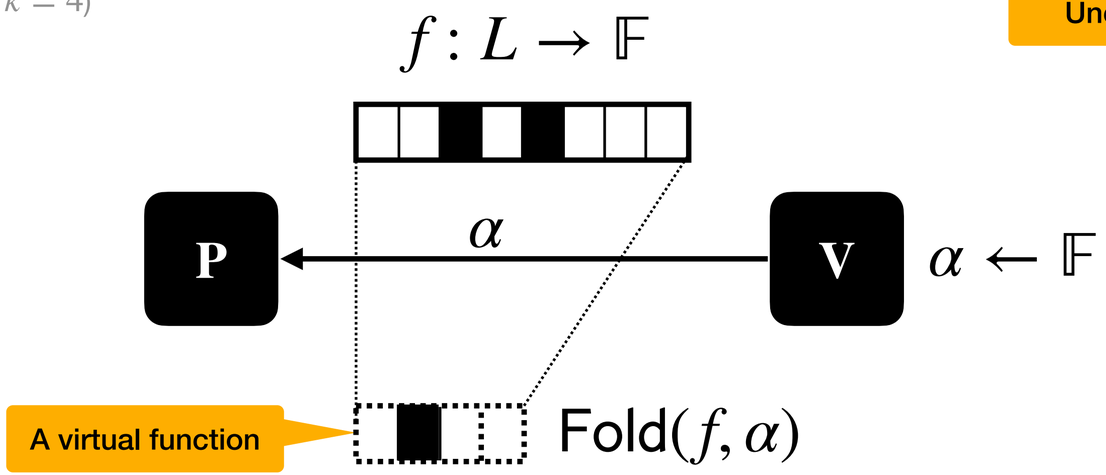
Can extend to every k that is a power of two.

FRI & STIR Folding

Reduce $\mathrm{RS}[n, m, \rho]$ **to** $\mathrm{RS}[n/2^k, m-k, \rho]$

(Think $k = 4$)

Unchanged!



How? Inspiration from FFTs, for $k = 1$:

$$\mathrm{Fold}(f, \alpha) := f_{\text{odd}} + \alpha \cdot f_{\text{even}}$$

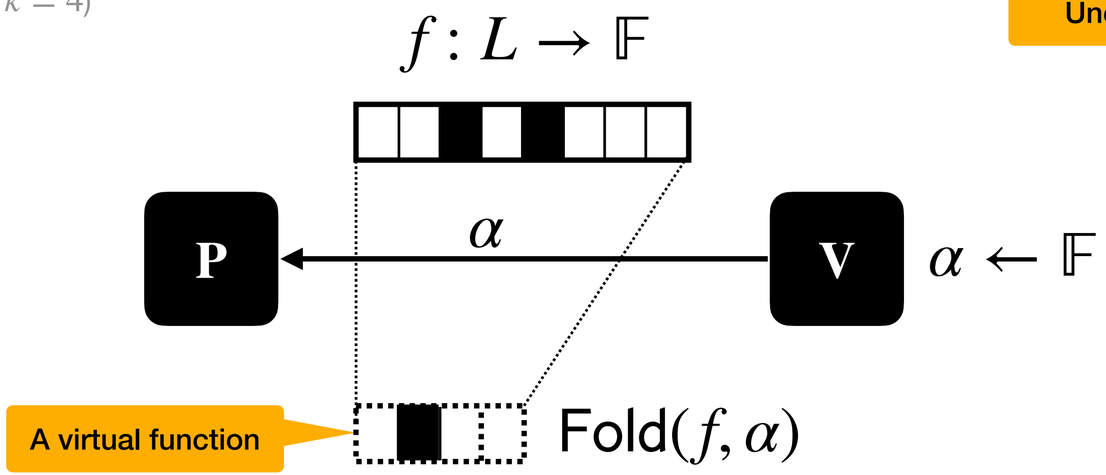
Can extend to every k that is a power of two.

Properties:

FRI & STIR Folding

Reduce $\mathrm{RS}[n, m, \rho]$ **to** $\mathrm{RS}[n/2^k, m-k, \rho]$

(Think $k = 4$)



Unchanged!

How? Inspiration from FFTs, for $k = 1$:

$$\mathrm{Fold}(f, \alpha) := f_{\text{odd}} + \alpha \cdot f_{\text{even}}$$

Can extend to every k that is a power of two.

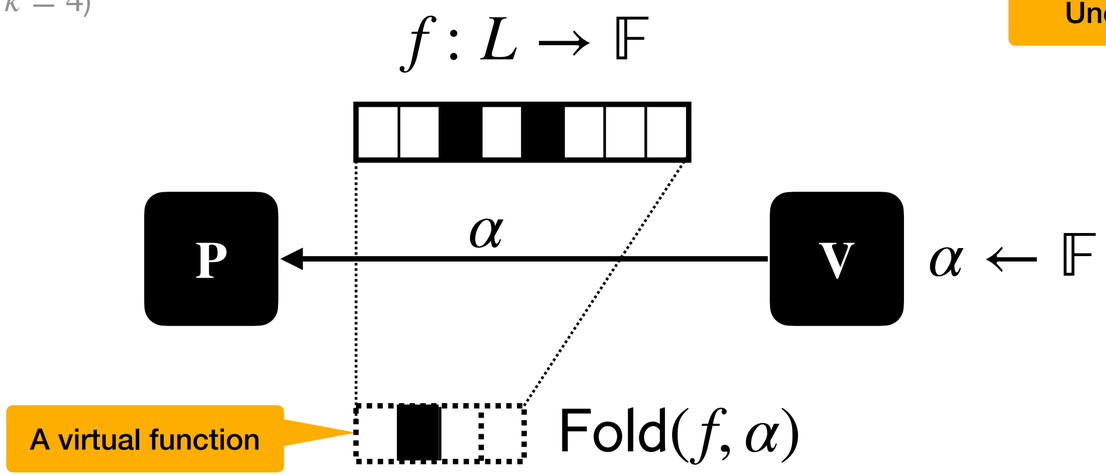
Properties:

Local: compute $\mathrm{Fold}(f, \alpha)(z)$ at any point $z \in L^{2^k}$ with 2^k queries to f.

FRI & STIR Folding

Reduce $\mathrm{RS}[n, m, \rho]$ **to** $\mathrm{RS}[n/2^k, m-k, \rho]$

(Think $k = 4$)



Unchanged!

How? Inspiration from FFTs, for $k = 1$:

$$\mathrm{Fold}(f, \alpha) := f_{\text{odd}} + \alpha \cdot f_{\text{even}}$$

Can extend to every k that is a power of two.

Properties:

Local: compute $\mathrm{Fold}(f, \alpha)(z)$ at any point $z \in L^{2^k}$ with 2^k queries to f.

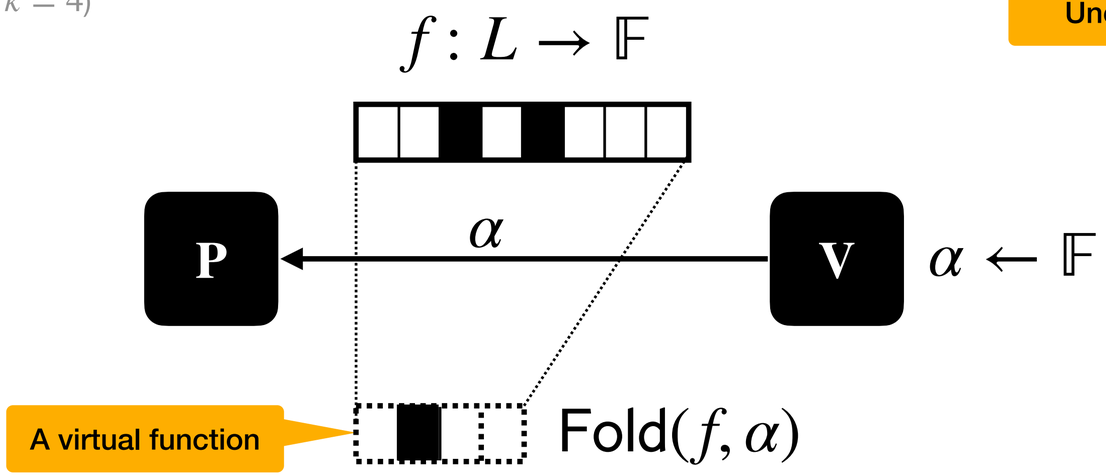
$$\delta \in \left(0, 1 - \sqrt{\rho}\right)$$

Distance preservation: if f is δ-far from $\mathrm{RS}[n, m, \rho]$, then w.h.p. $\mathrm{Fold}(f, \alpha)$ remains also δ-far from $\mathrm{RS}[n/2^k, m-k, \rho]$

FRI & STIR Folding

Reduce $\mathrm{RS}[n, m, \rho]$ **to** $\mathrm{RS}[n/2^k, m-k, \rho]$

(Think $k = 4$)



Unchanged!

How? Inspiration from FFTs, for $k = 1$:

$$\mathrm{Fold}(f, \alpha) := f_{\text{odd}} + \alpha \cdot f_{\text{even}}$$

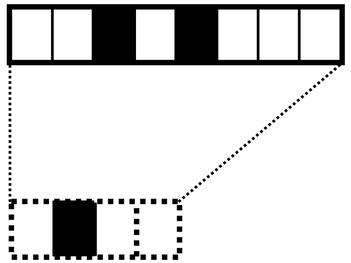
Can extend to every k that is a power of two.

Properties:

Local: compute $\mathrm{Fold}(f, \alpha)(z)$ at any point $z \in L^{2^k}$ with 2^k queries to f.

$$\delta \in \left(0, 1 - \sqrt{\rho}\right)$$

Distance preservation: if f is δ-far from $\mathrm{RS}[n, m, \rho]$, then w.h.p. $\mathrm{Fold}(f, \alpha)$ remains also δ-far from $\mathrm{RS}[n/2^k, m-k, \rho]$

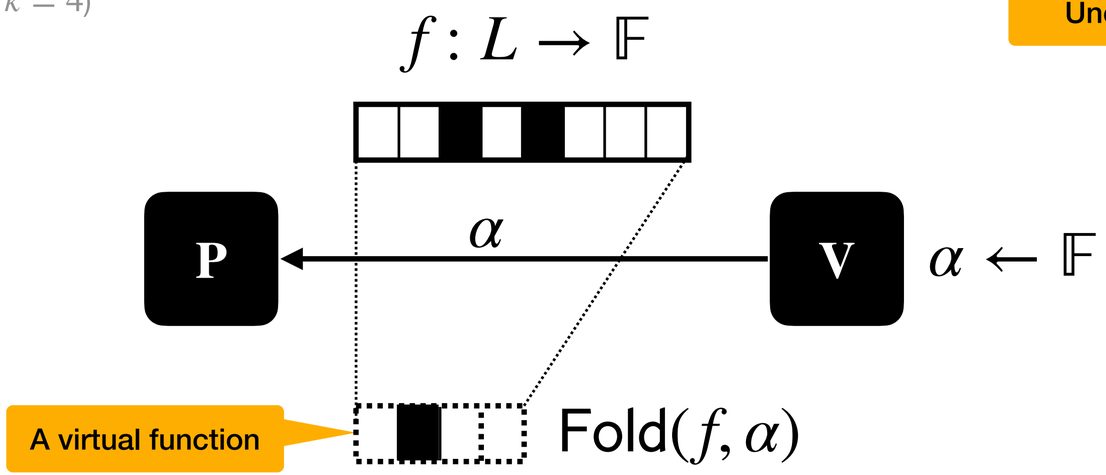


Unless w.p. $\approx \frac{\text{poly}(n, 2^m)}{|\mathbb{F}|}$, the fraction of "corrupted" entries does not decrease.

FRI & STIR Folding

Reduce $\mathrm{RS}[n, m, \rho]$ **to** $\mathrm{RS}[n/2^k, m-k, \rho]$

(Think $k = 4$)



Unchanged!

How? Inspiration from FFTs, for $k = 1$:

$$\mathrm{Fold}(f, \alpha) := f_{\text{odd}} + \alpha \cdot f_{\text{even}}$$

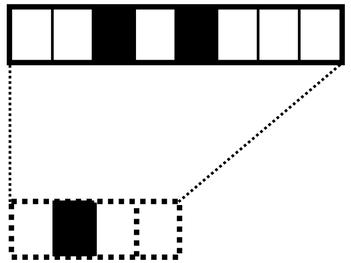
Can extend to every k that is a power of two.

Properties:

Local: compute $\mathrm{Fold}(f, \alpha)(z)$ at any point $z \in L^{2^k}$ with 2^k queries to f.

$$\delta \in \left(0, 1 - \sqrt{\rho}\right)$$

Distance preservation: if f is δ-far from $\mathrm{RS}[n, m, \rho]$, then w.h.p. $\mathrm{Fold}(f, \alpha)$ remains also δ-far from $\mathrm{RS}[n/2^k, m-k, \rho]$



Unless w.p. $\approx \frac{\mathrm{poly}(n, 2^m)}{|\mathbb{F}|}$, the fraction of "corrupted" entries does not decrease.

Proximity Gaps for Reed–Solomon Codes

Eli Ben-Sasson* Dan Carmon* Yuval Ishai† Swastik Kopparty ‡
Shubhangi Saraf§

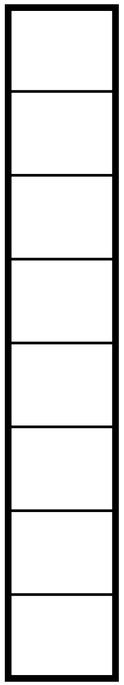
July 3, 2021

Mutual correlated agreement

Test a random linear combination

Mutual correlated agreement

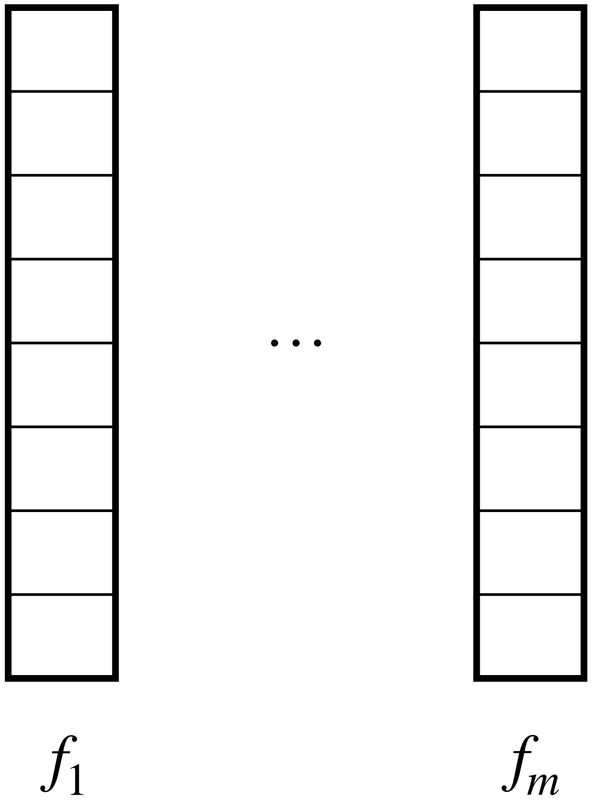
Test a random linear combination



f_1

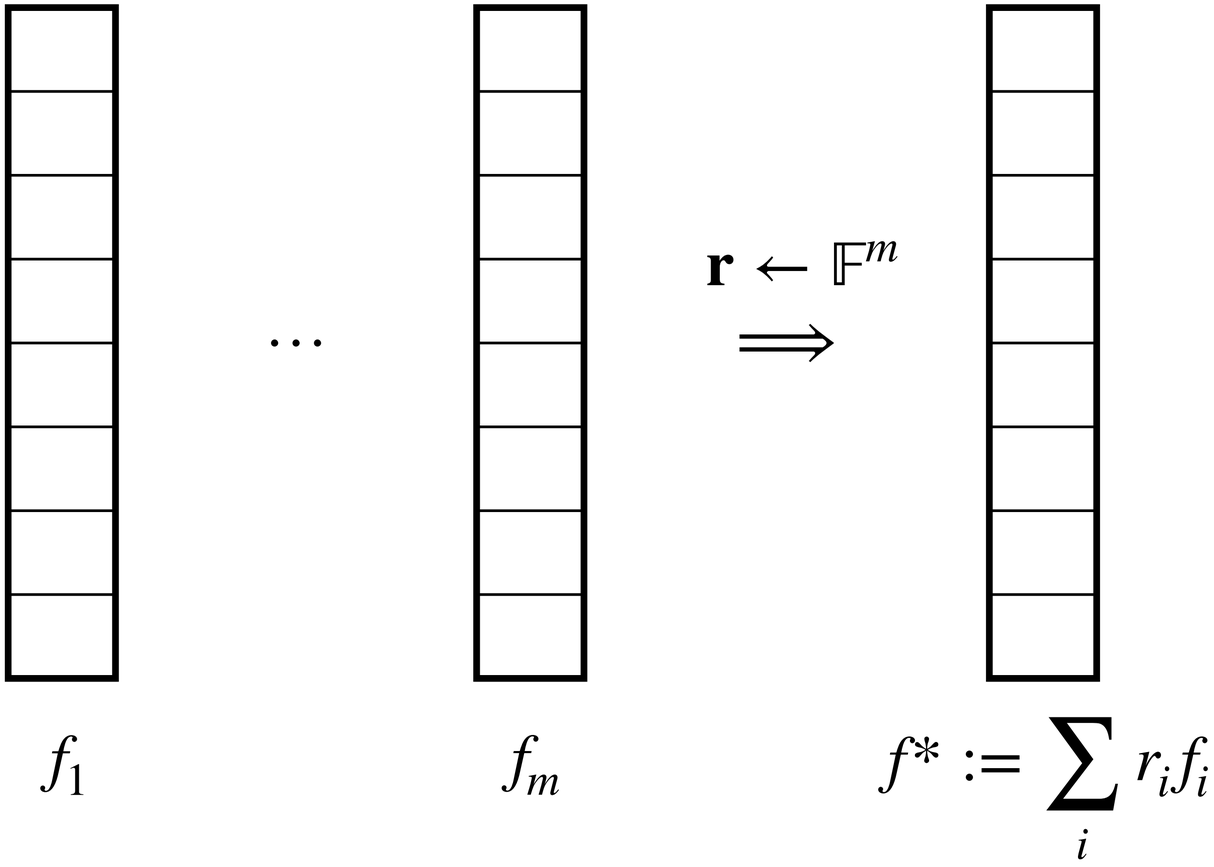
Mutual correlated agreement

Test a random linear combination



Mutual correlated agreement

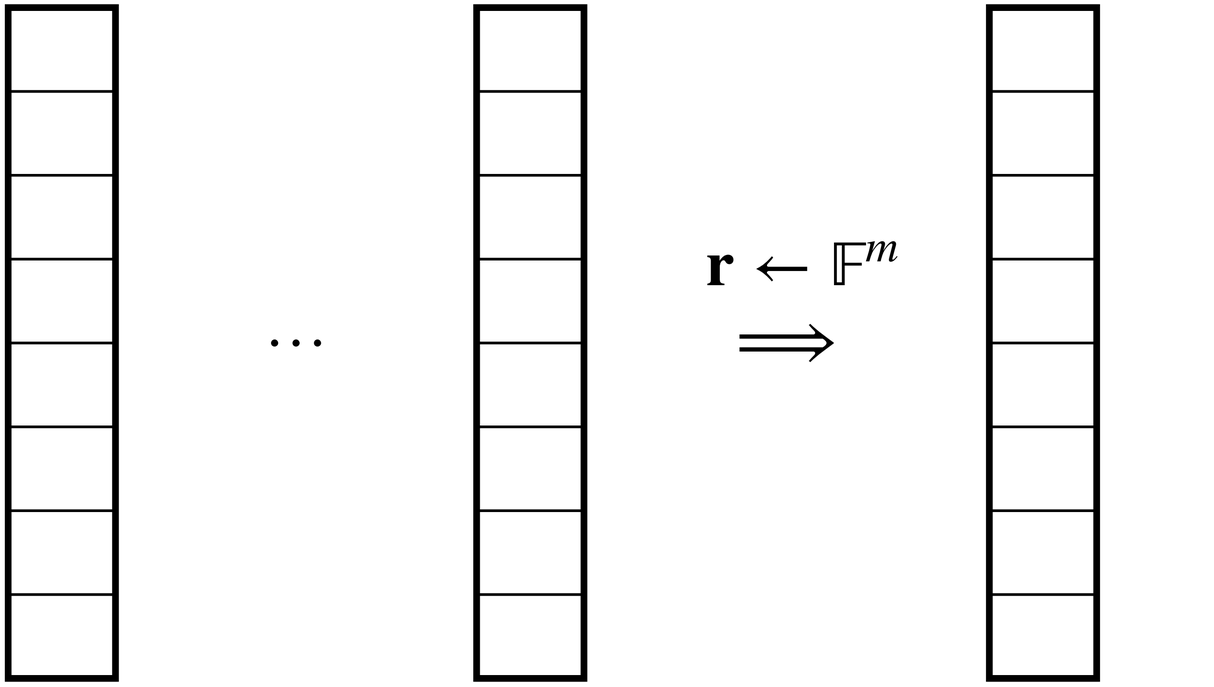
Test a random linear combination



Mutual correlated agreement

Test a random linear combination

if w.h.p. $\Delta(f^*, \mathscr{C}) \leq \delta$:

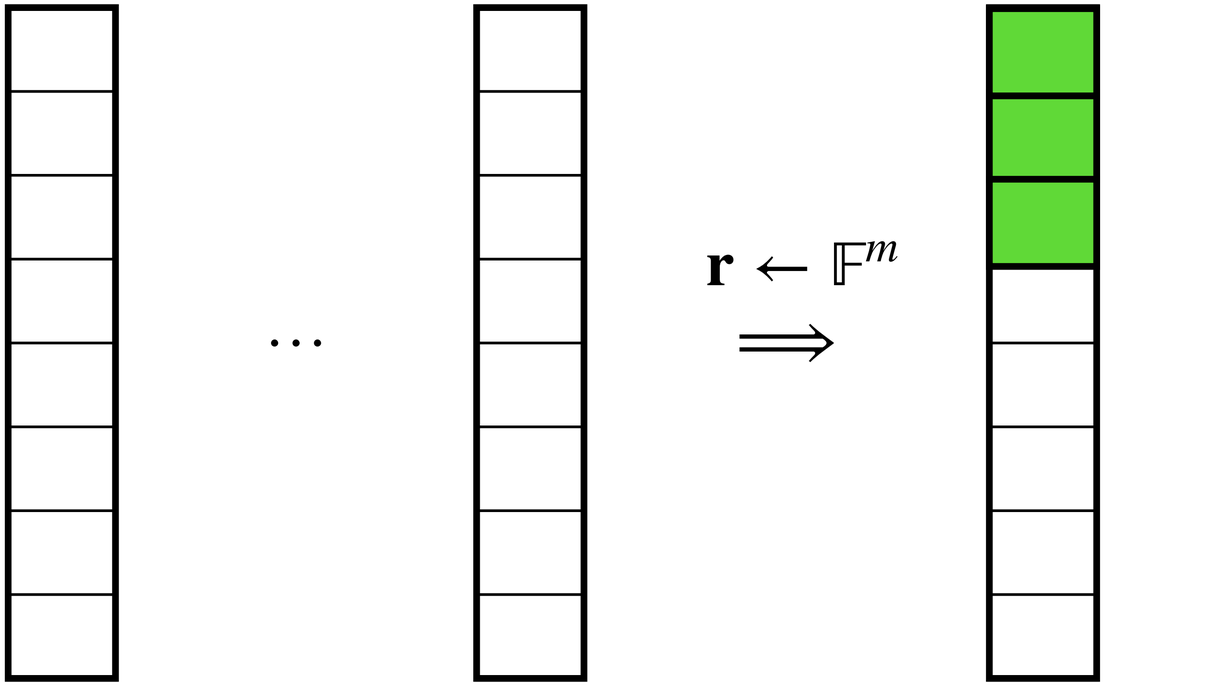


f_1 f_m $f^* := \sum_i r_i f_i$

Mutual correlated agreement

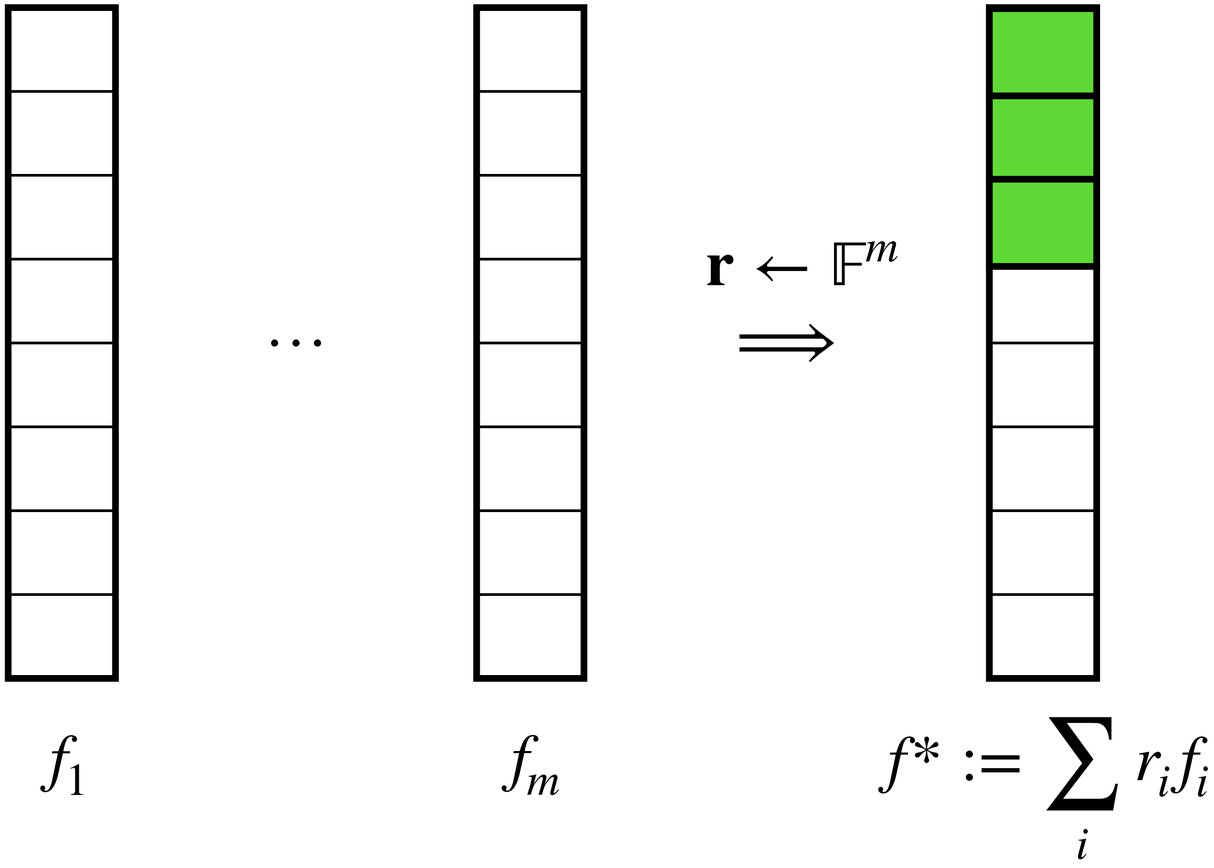
Test a random linear combination

if w.h.p. $\Delta(f^*, \mathscr{C}) \leq \delta$:



Mutual correlated agreement

Test a random linear combination

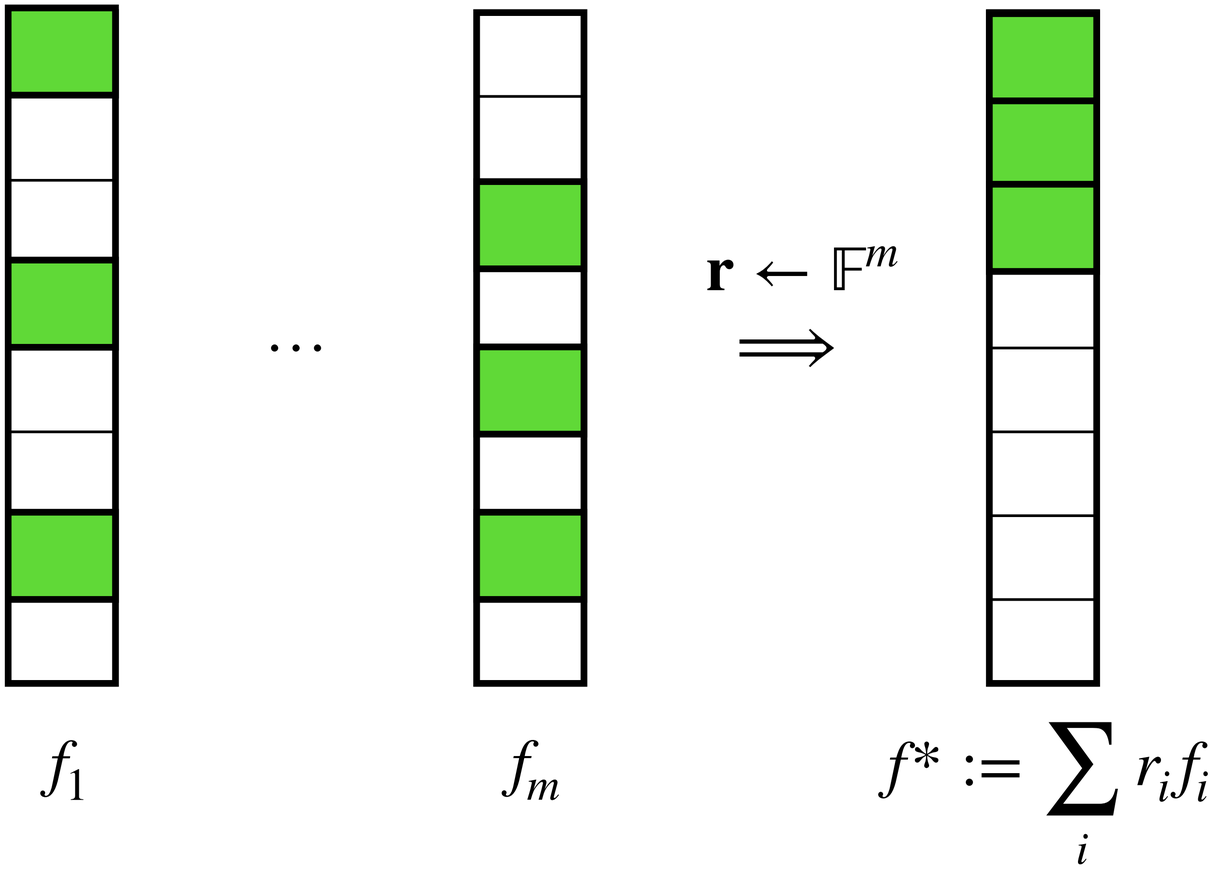


if w.h.p. $\Delta(f^*, \mathscr{C}) \leq \delta$:

Agreement: then $\Delta(f_i, \mathscr{C}) \leq \delta$.

Mutual correlated agreement

Test a random linear combination

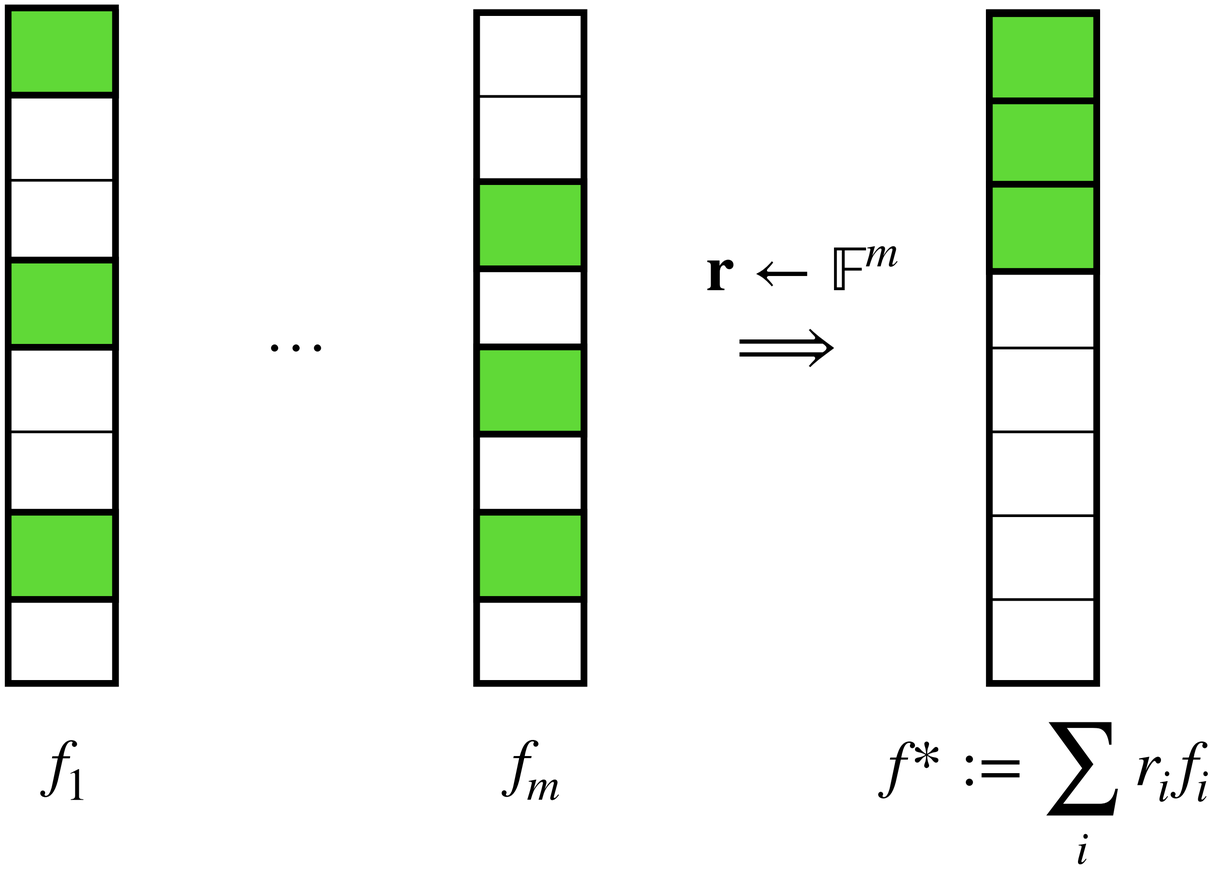


if w.h.p. $\Delta(f^*, \mathscr{C}) \leq \delta$:

Agreement: then $\Delta(f_i, \mathscr{C}) \leq \delta$.

Mutual correlated agreement

Test a random linear combination



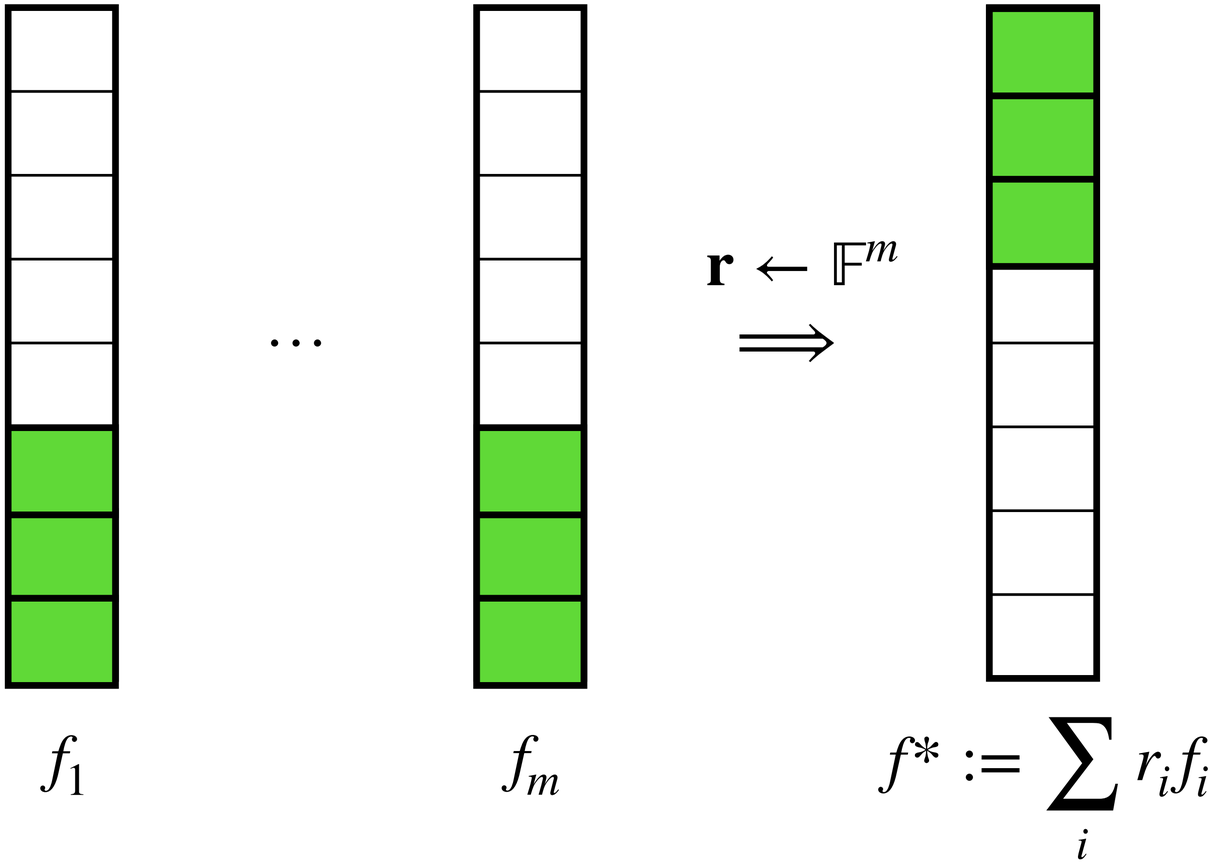
if w.h.p. $\Delta(f^*, \mathscr{C}) \leq \delta$:

Agreement: then $\Delta(f_i, \mathscr{C}) \leq \delta$.

Correlated agreement: then $f_1, \ldots, f_m$ agree with $\mathscr{C}$ on the same “stripe”

Mutual correlated agreement

Test a random linear combination



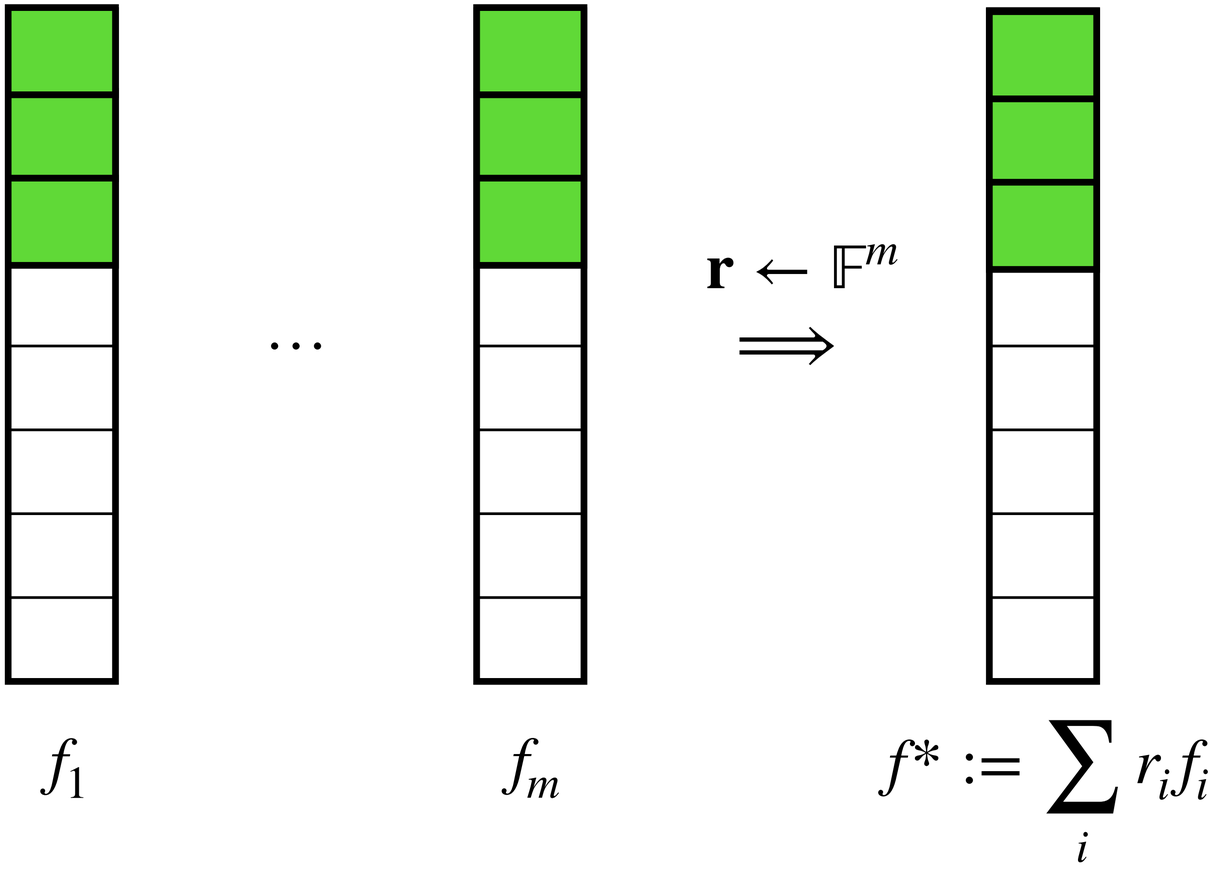
if w.h.p. $\Delta(f^*, \mathscr{C}) \leq \delta$:

Agreement: then $\Delta(f_i, \mathscr{C}) \leq \delta$.

Correlated agreement: then $f_1, \ldots, f_m$ agree with $\mathscr{C}$ on the same “stripe”

Mutual correlated agreement

Test a random linear combination



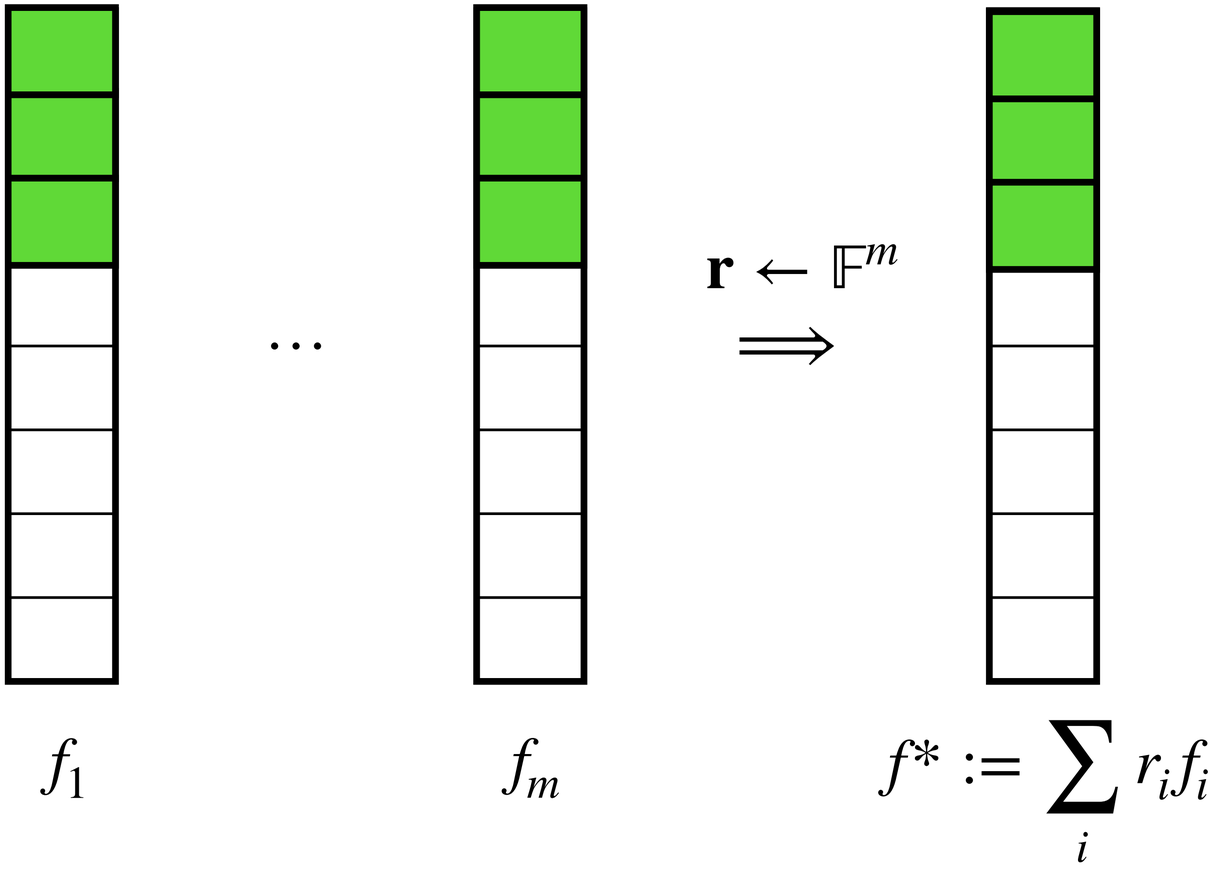
if w.h.p. $\Delta(f^*, \mathscr{C}) \leq \delta$:

Agreement: then $\Delta(f_i, \mathscr{C}) \leq \delta$.

Correlated agreement: then $f_1, \ldots, f_m$ agree with $\mathscr{C}$ on the same “stripe”

Mutual correlated agreement

Test a random linear combination



if w.h.p. $\Delta(f^*, \mathscr{C}) \leq \delta$:

Agreement: then $\Delta(f_i, \mathscr{C}) \leq \delta$.

Correlated agreement: then $f_1, \ldots, f_m$ agree with $\mathscr{C}$ on the same "stripe"

Mutual correlated agreement: the stripe in which $f_1, \ldots, f_m$ agree with $\mathscr{C}$ is the same on which f^* does:

"*No new correlated domains appear*"

Folding and lists commute

Implied by mutual correlated agreement

$\Lambda(\mathscr{C}, f, \delta)$ is the list of codewords of $\mathscr{C}$ that are δ-close to f

Folding and lists commute

Implied by mutual correlated agreement

$\Lambda(\mathscr{C}, f, \delta)$ is the list of codewords of $\mathscr{C}$ that are δ-close to f

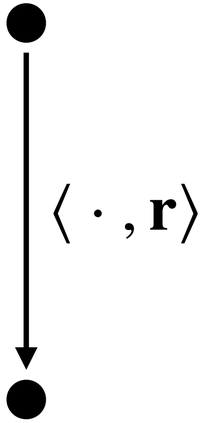
$f_1, \ldots, f_m \colon L \to \mathbb{F}$

Folding and lists commute

Implied by mutual correlated agreement

$\Lambda(\mathscr{C}, f, \delta)$ is the list of codewords of $\mathscr{C}$ that are δ-close to f

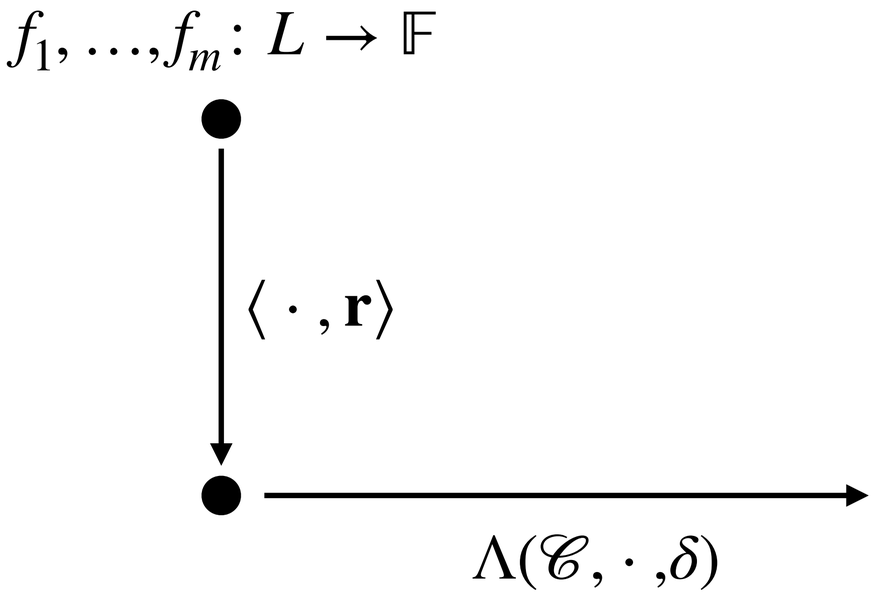
$f_1, \ldots, f_m \colon L \to \mathbb{F}$



Folding and lists commute

Implied by mutual correlated agreement

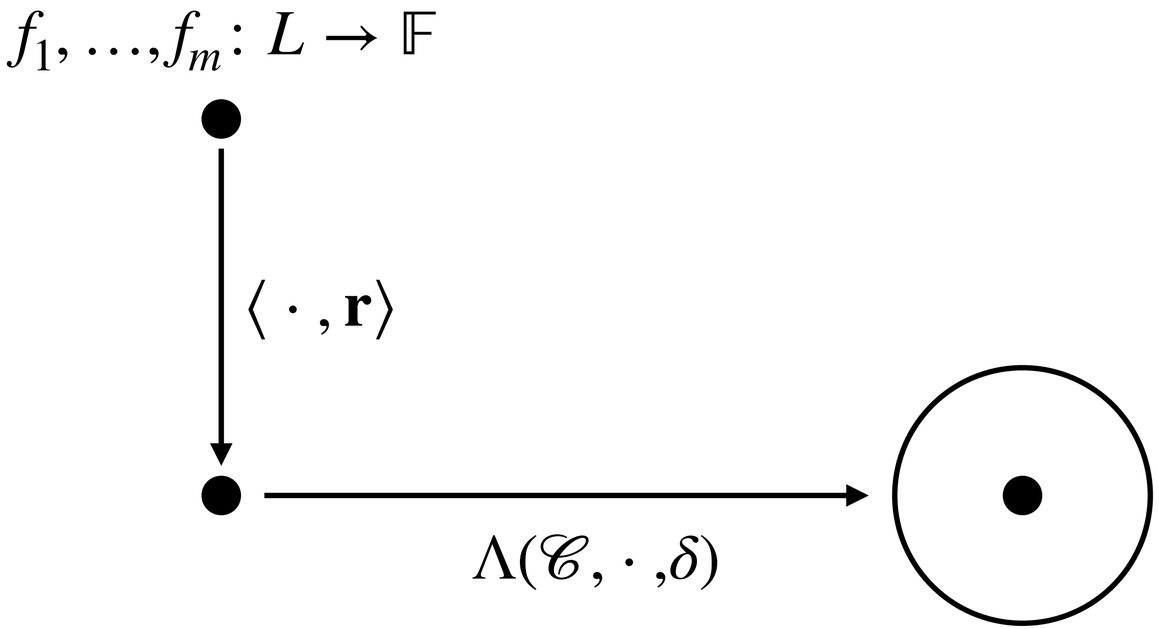
$\Lambda(\mathscr{C}, f, \delta)$ is the list of codewords of $\mathscr{C}$ that are δ-close to f



Folding and lists commute

Implied by mutual correlated agreement

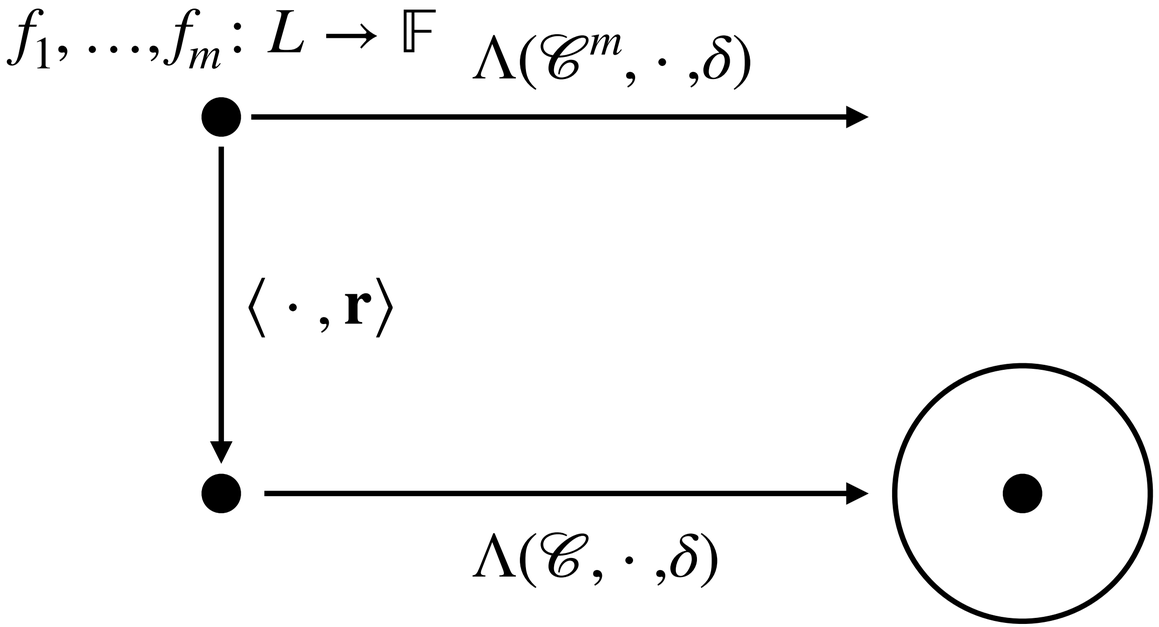
$\Lambda(\mathscr{C}, f, \delta)$ is the list of codewords of $\mathscr{C}$ that are δ-close to f



Folding and lists commute

Implied by mutual correlated agreement

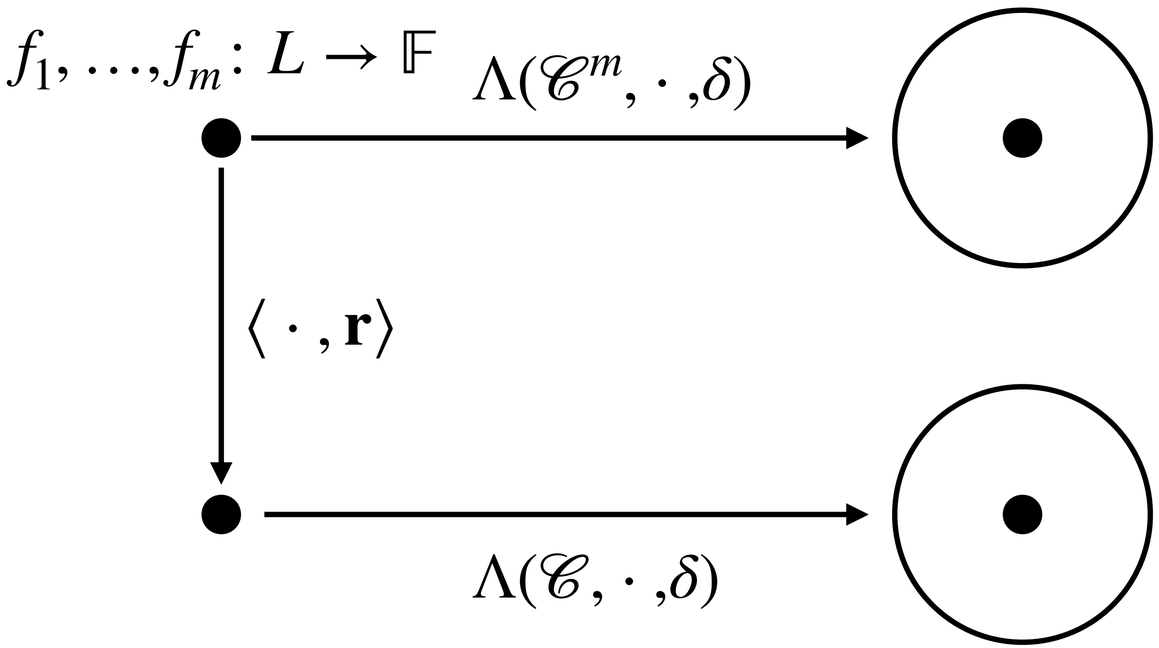
$\Lambda(\mathscr{C}, f, \delta)$ is the list of codewords of $\mathscr{C}$ that are δ-close to f



Folding and lists commute

Implied by mutual correlated agreement

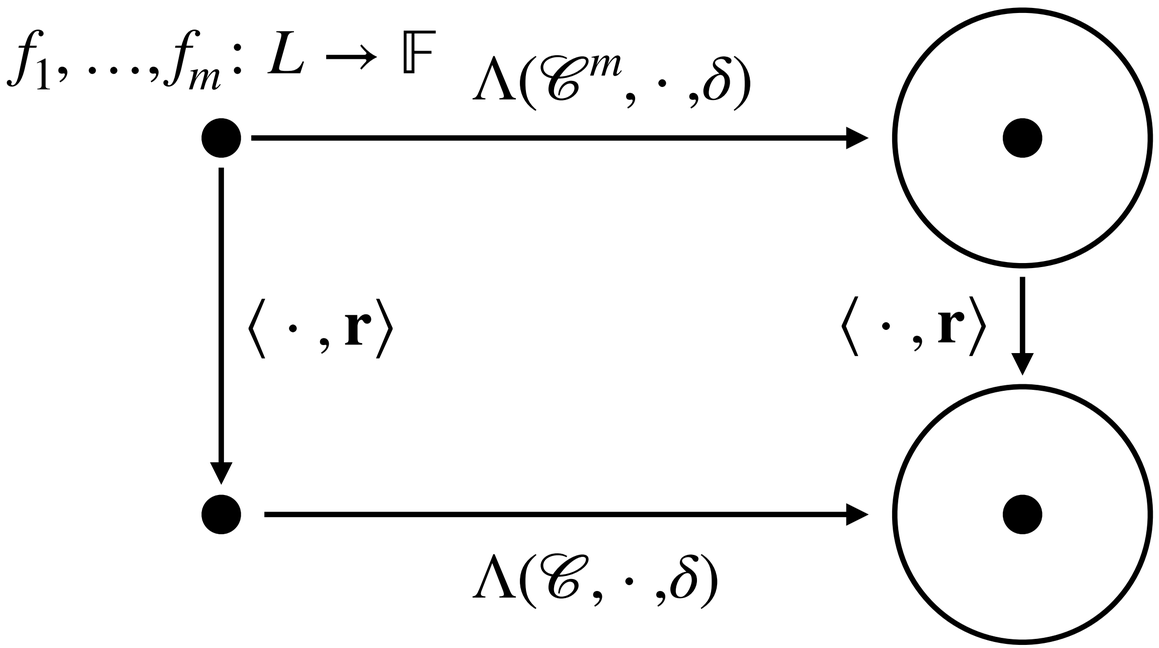
$\Lambda(\mathscr{C}, f, \delta)$ is the list of codewords of $\mathscr{C}$ that are δ-close to f



Folding and lists commute

Implied by mutual correlated agreement

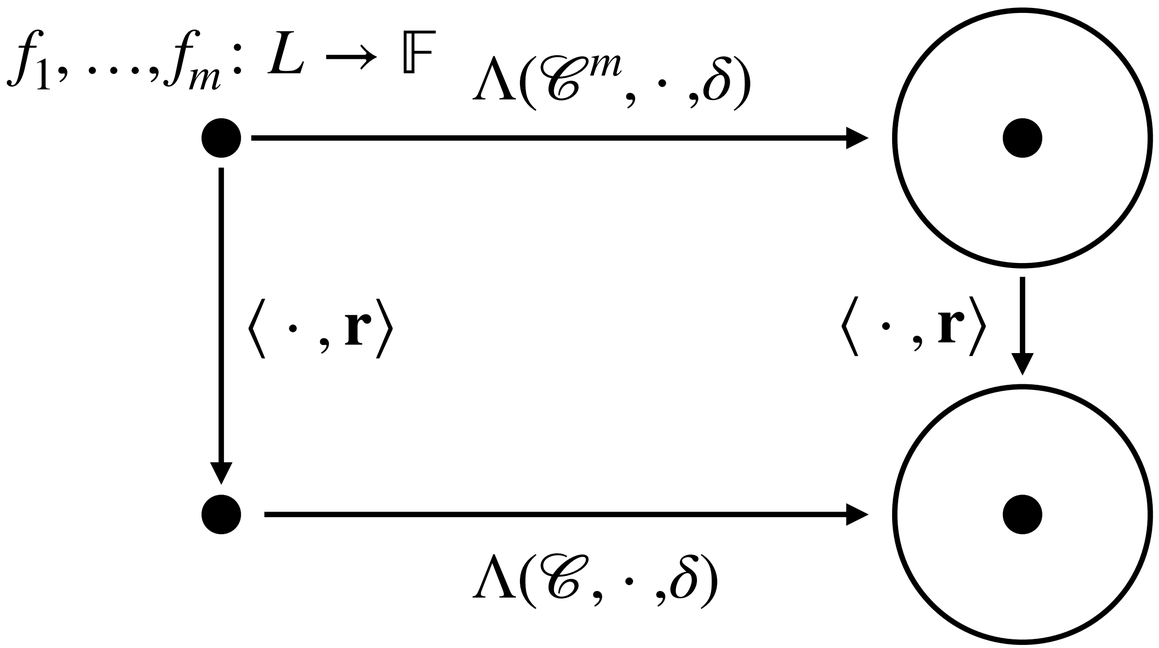
$\Lambda(\mathscr{C}, f, \delta)$ is the list of codewords of $\mathscr{C}$ that are δ-close to f



Folding and lists commute

Implied by mutual correlated agreement

$\Lambda(\mathscr{C}, f, \delta)$ is the list of codewords of $\mathscr{C}$ that are δ-close to f

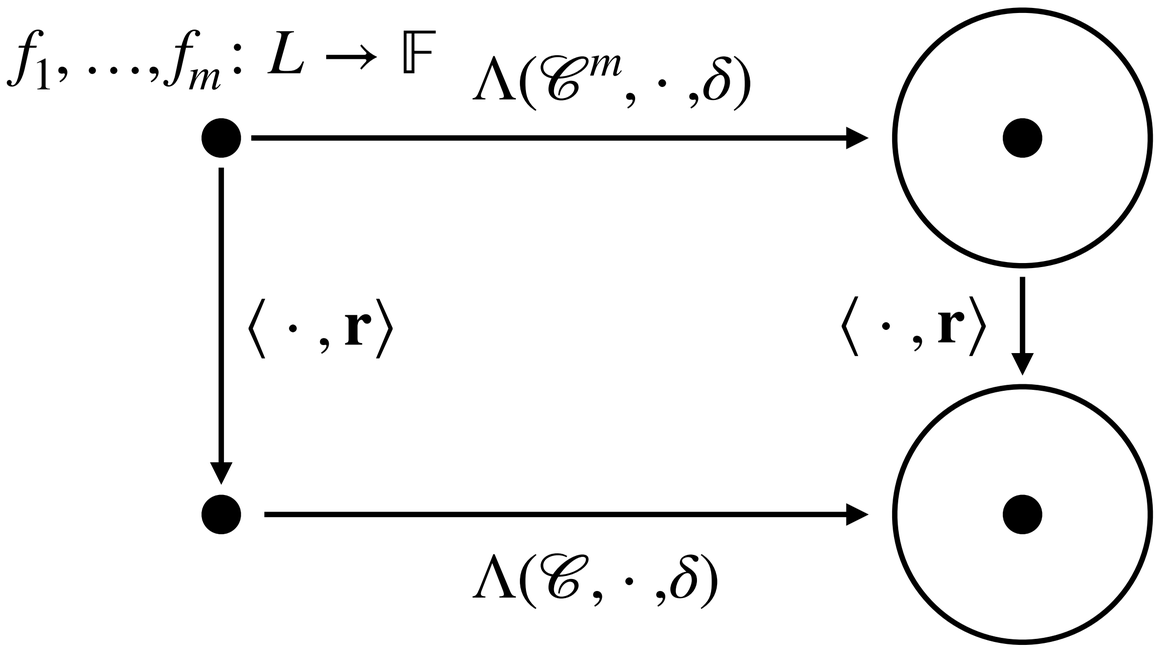


Stronger than what is required for STIR's soundness

Folding and lists commute

Implied by mutual correlated agreement

$\Lambda(\mathscr{C}, f, \delta)$ is the list of codewords of $\mathscr{C}$ that are δ-close to f



Stronger than what is required for STIR's soundness

Lemma

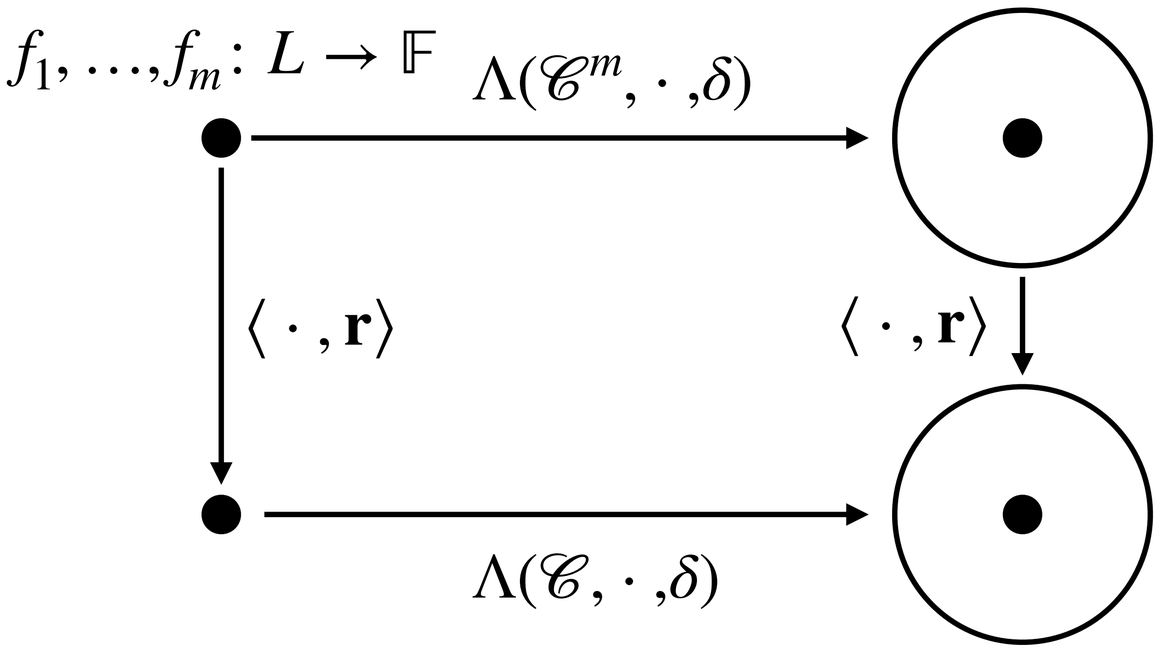
w.h.p. over $\mathbf{r}$:

$$\Lambda(\mathscr{C}, \langle \mathbf{f}, \mathbf{r} \rangle, \delta) = \left\{ \langle \mathbf{u}, \mathbf{r} \rangle : \mathbf{u} \in \Lambda(\mathscr{C}^m, \mathbf{f}, \delta) \right\}$$

Folding and lists commute

Implied by mutual correlated agreement

$\Lambda(\mathscr{C}, f, \delta)$ is the list of codewords of $\mathscr{C}$ that are δ-close to f



Stronger than what is required for STIR's soundness

Lemma

w.h.p. over $\mathbf{r}$:

$$\Lambda(\mathscr{C}, \langle \mathbf{f}, \mathbf{r} \rangle, \delta) = \left\{ \langle \mathbf{u}, \mathbf{r} \rangle : \mathbf{u} \in \Lambda(\mathscr{C}^m, \mathbf{f}, \delta) \right\}$$

Lemma

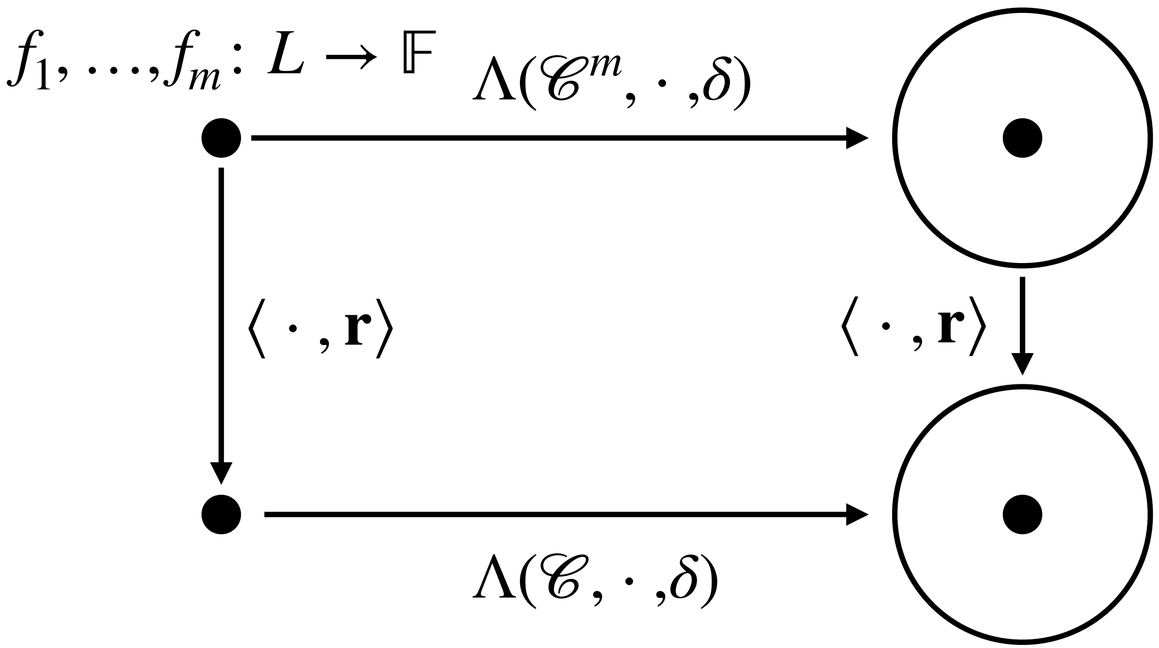
w.h.p. over $\mathbf{r}$:

$$\Lambda(\mathscr{C}, \mathsf{Fold}(f, \alpha), \delta) = \left\{ \mathsf{Fold}(u, \alpha) : u \in \Lambda(\mathscr{C}, f, \delta) \right\}$$

Folding and lists commute

Implied by mutual correlated agreement

$\Lambda(\mathscr{C}, f, \delta)$ is the list of codewords of $\mathscr{C}$ that are δ-close to f



Stronger than what is required for STIR's soundness

Lemma

w.h.p. over $\mathbf{r}$:

$$\Lambda(\mathscr{C}, \langle \mathbf{f}, \mathbf{r} \rangle, \delta) = \left\{ \langle \mathbf{u}, \mathbf{r} \rangle : \mathbf{u} \in \Lambda(\mathscr{C}^m, \mathbf{f}, \delta) \right\}$$

Lemma

w.h.p. over $\mathbf{r}$:

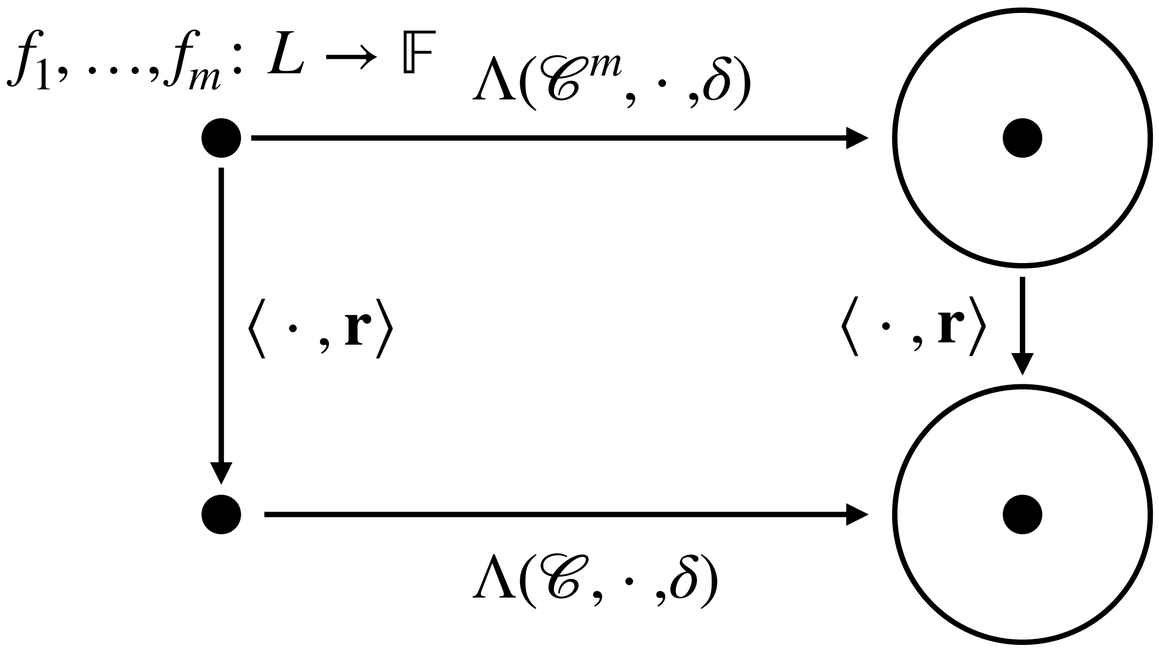
$$\Lambda(\mathscr{C}, \mathsf{Fold}(f, \alpha), \delta) = \left\{ \mathsf{Fold}(u, \alpha) : u \in \Lambda(\mathscr{C}, f, \delta) \right\}$$

Alternatively, each term in the l.h.s can be "explained" by terms in the r.h.s.

Folding and lists commute

Implied by mutual correlated agreement

$\Lambda(\mathscr{C}, f, \delta)$ is the list of codewords of $\mathscr{C}$ that are δ-close to f



Stronger than what is required for STIR's soundness

Lemma

w.h.p. over $\mathbf{r}$:

$$\Lambda(\mathscr{C}, \langle \mathbf{f}, \mathbf{r} \rangle, \delta) = \left\{ \langle \mathbf{u}, \mathbf{r} \rangle : \mathbf{u} \in \Lambda(\mathscr{C}^m, \mathbf{f}, \delta) \right\}$$

Lemma

w.h.p. over $\mathbf{r}$:

$$\Lambda(\mathscr{C}, \mathsf{Fold}(f, \alpha), \delta) = \left\{ \mathsf{Fold}(u, \alpha) : u \in \Lambda(\mathscr{C}, f, \delta) \right\}$$

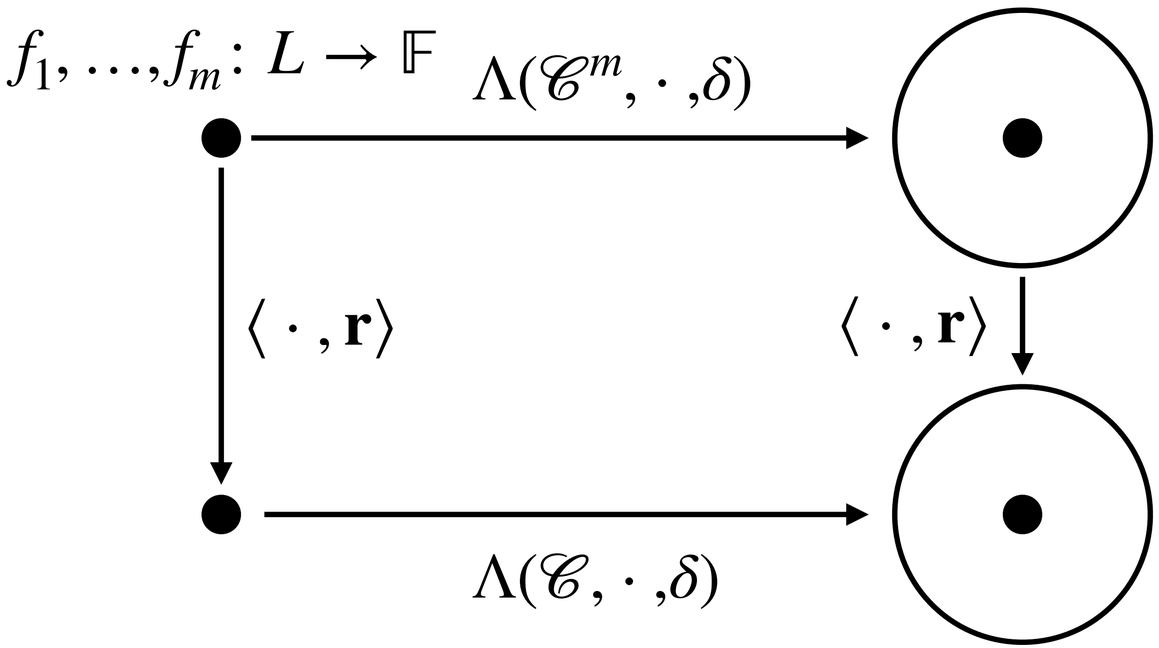
Alternatively, each term in the l.h.s can be "explained" by terms in the r.h.s.

We show correlated agreement implies mutual correlated agreement in *unique decoding.*

Folding and lists commute

Implied by mutual correlated agreement

$\Lambda(\mathscr{C}, f, \delta)$ is the list of codewords of $\mathscr{C}$ that are δ-close to f



Stronger than what is required for STIR's soundness

Lemma

w.h.p. over $\mathbf{r}$:

$$\Lambda(\mathscr{C}, \langle \mathbf{f}, \mathbf{r} \rangle, \delta) = \left\{ \langle \mathbf{u}, \mathbf{r} \rangle : \mathbf{u} \in \Lambda(\mathscr{C}^m, \mathbf{f}, \delta) \right\}$$

Lemma

w.h.p. over $\mathbf{r}$:

$$\Lambda(\mathscr{C}, \mathsf{Fold}(f, \alpha), \delta) = \left\{ \mathsf{Fold}(u, \alpha) : u \in \Lambda(\mathscr{C}, f, \delta) \right\}$$

Recent results show that mutual correlated agreement holds up to 1.5 Johnson for general linear codes!

We show correlated agreement implies mutual correlated agreement in *unique decoding.*

WHIR Folding

Reduce CRS$[n, m, \rho, \hat{w}, \sigma]$ **to** CRS$[n/2, m-1, \rho, \hat{w}_\alpha, \sigma_\alpha]$

WHIR Folding

Reduce CRS$[n, m, \rho, \hat{w}, \sigma]$ **to** CRS$[n/2, m-1, \rho, \hat{w}_\alpha, \sigma_\alpha]$

Interleave sumcheck with FRI folding, similar to BaseFold, Hyperplonk, Gemini

WHIR Folding

Interleave sumcheck with FRI folding, similar to BaseFold, Hyperplonk, Gemini

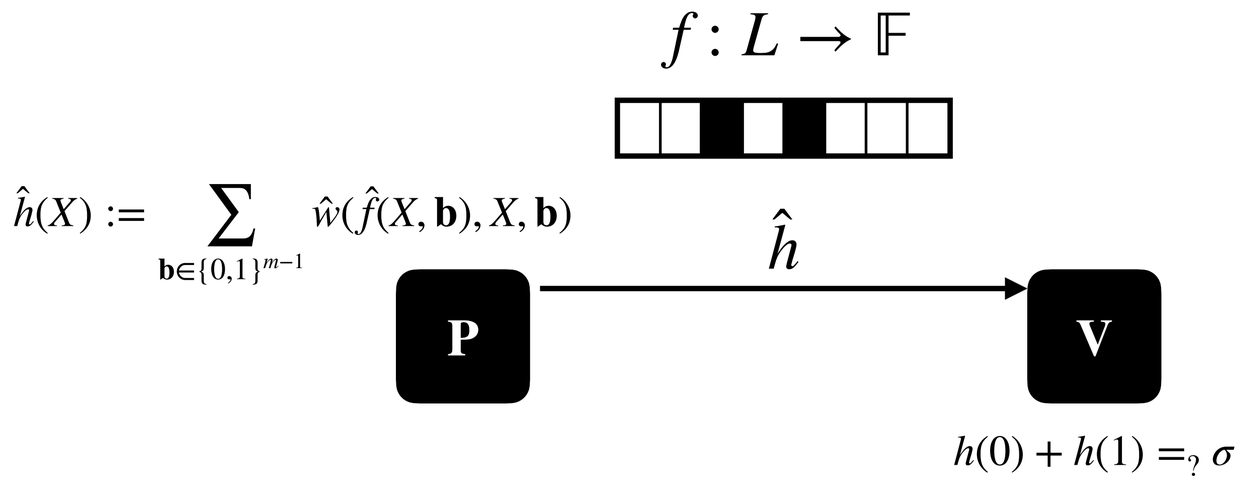
Reduce CRS$[n, m, \rho, \hat{w}, \sigma]$ **to** CRS$[n/2, m-1, \rho, \hat{w}_\alpha, \sigma_\alpha]$

$$f : L \to \mathbb{F}$$

WHIR Folding

Interleave sumcheck with FRI folding, similar to BaseFold, Hyperplonk, Gemini

Reduce CRS$[n, m, \rho, \hat{w}, \sigma]$ **to** CRS$[n/2, m-1, \rho, \hat{w}_\alpha, \sigma_\alpha]$



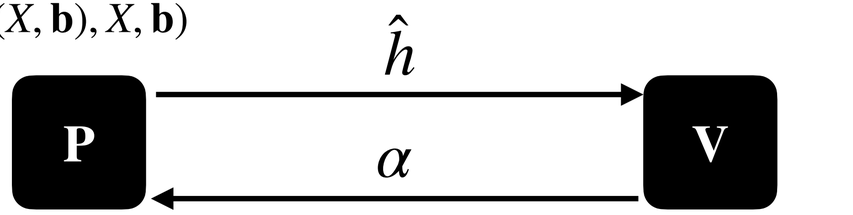
WHIR Folding

Interleave sumcheck with FRI folding, similar to BaseFold, Hyperplonk, Gemini

Reduce CRS$[n, m, \rho, \hat{w}, \sigma]$ **to** CRS$[n/2, m-1, \rho, \hat{w}_\alpha, \sigma_\alpha]$

$$f : L \to \mathbb{F}$$

$$\hat{h}(X) := \sum_{\mathbf{b} \in \{0,1\}^{m-1}} \hat{w}(\hat{f}(X, \mathbf{b}), X, \mathbf{b})$$

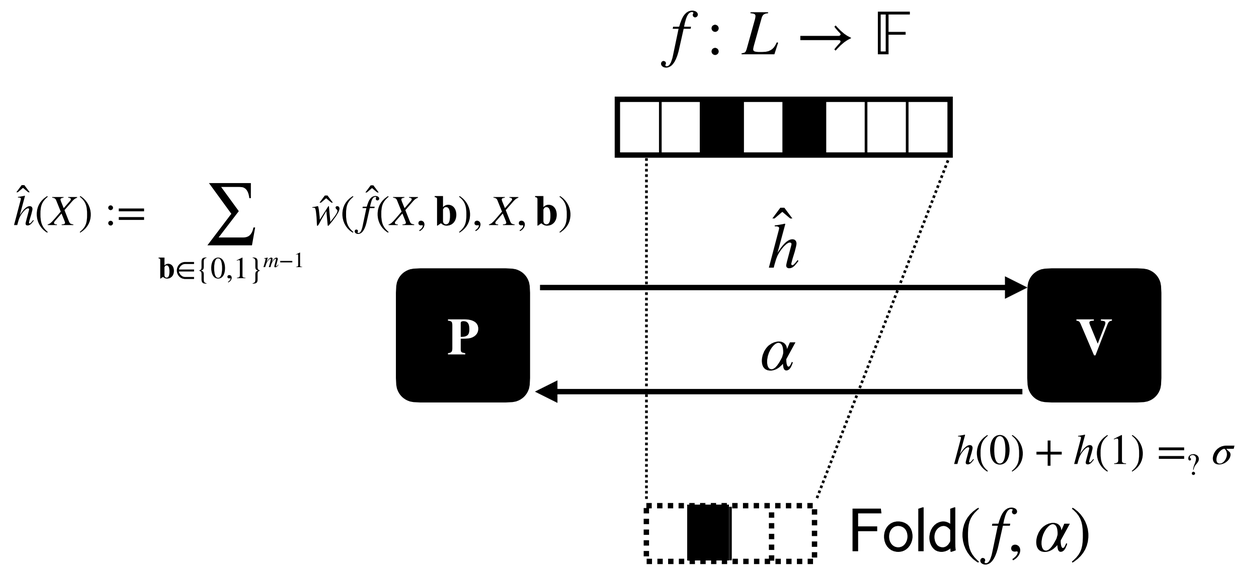


$$h(0) + h(1) =_? \sigma$$

WHIR Folding

Interleave sumcheck with FRI folding, similar to BaseFold, Hyperplonk, Gemini

Reduce CRS$[n, m, \rho, \hat{w}, \sigma]$ **to** CRS$[n/2, m-1, \rho, \hat{w}_\alpha, \sigma_\alpha]$

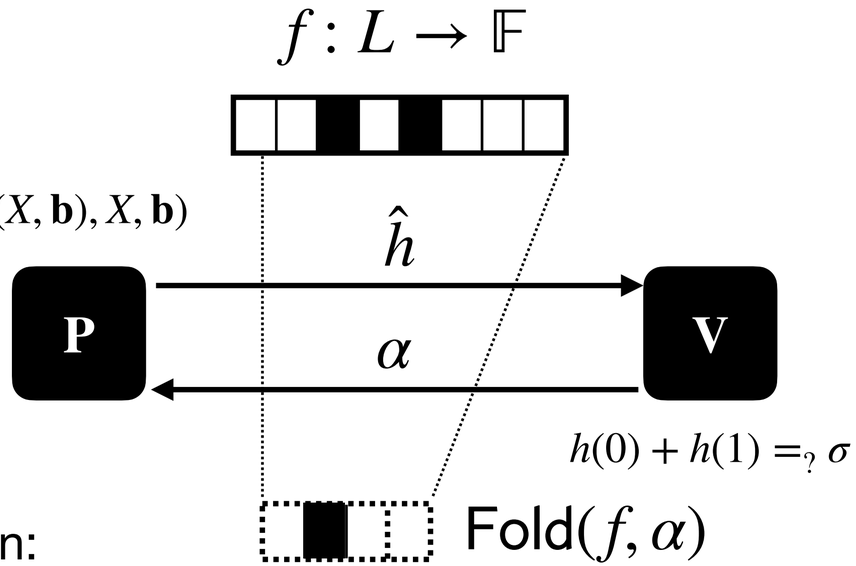


WHIR Folding

Interleave sumcheck with FRI folding, similar to BaseFold, Hyperplonk, Gemini

Reduce CRS$[n, m, \rho, \hat{w}, \sigma]$ **to** CRS$[n/2, m-1, \rho, \hat{w}_\alpha, \sigma_\alpha]$

$$\hat{h}(X) := \sum_{\mathbf{b} \in \{0,1\}^{m-1}} \hat{w}(\hat{f}(X, \mathbf{b}), X, \mathbf{b})$$

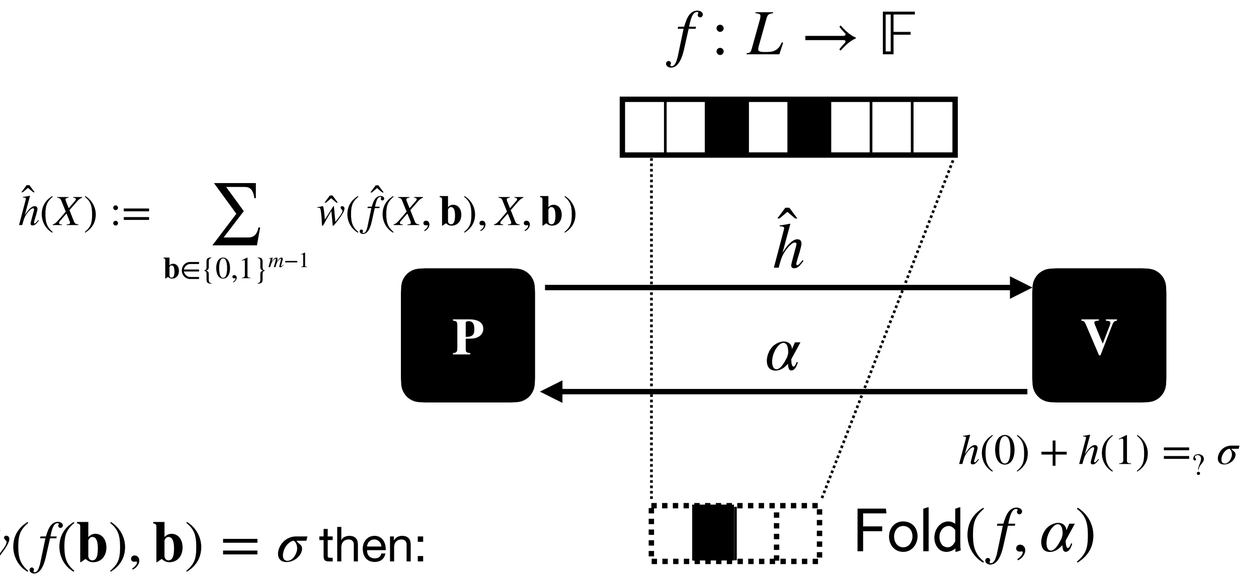


Completeness: $\sum_{\mathbf{b}} \hat{w}(f(\mathbf{b}), \mathbf{b}) = \sigma$ then:

WHIR Folding

Interleave sumcheck with FRI folding, similar to BaseFold, Hyperplonk, Gemini

Reduce CRS$[n, m, \rho, \hat{w}, \sigma]$ **to** CRS$[n/2, m-1, \rho, \hat{w}_\alpha, \sigma_\alpha]$



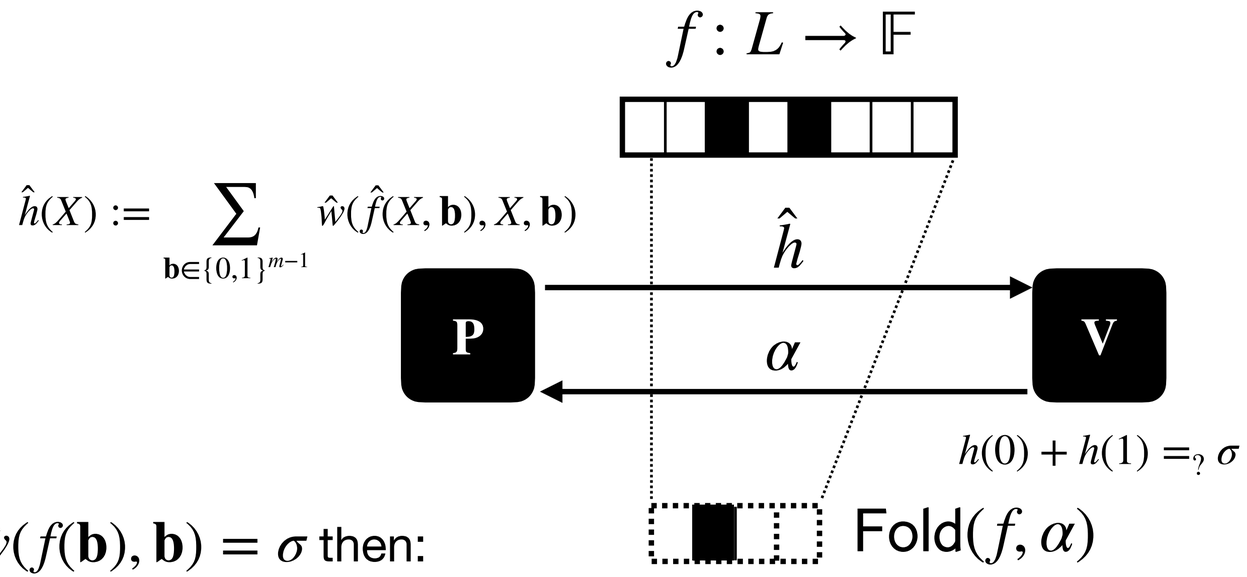
Completeness: $\sum_{\mathbf{b}} \hat{w}(f(\mathbf{b}), \mathbf{b}) = \sigma$ then:

- $h(0) + h(1) = \sigma$,

WHIR Folding

Interleave sumcheck with FRI folding, similar to BaseFold, Hyperplonk, Gemini

Reduce CRS$[n, m, \rho, \hat{w}, \sigma]$ **to** CRS$[n/2, m-1, \rho, \hat{w}_\alpha, \sigma_\alpha]$



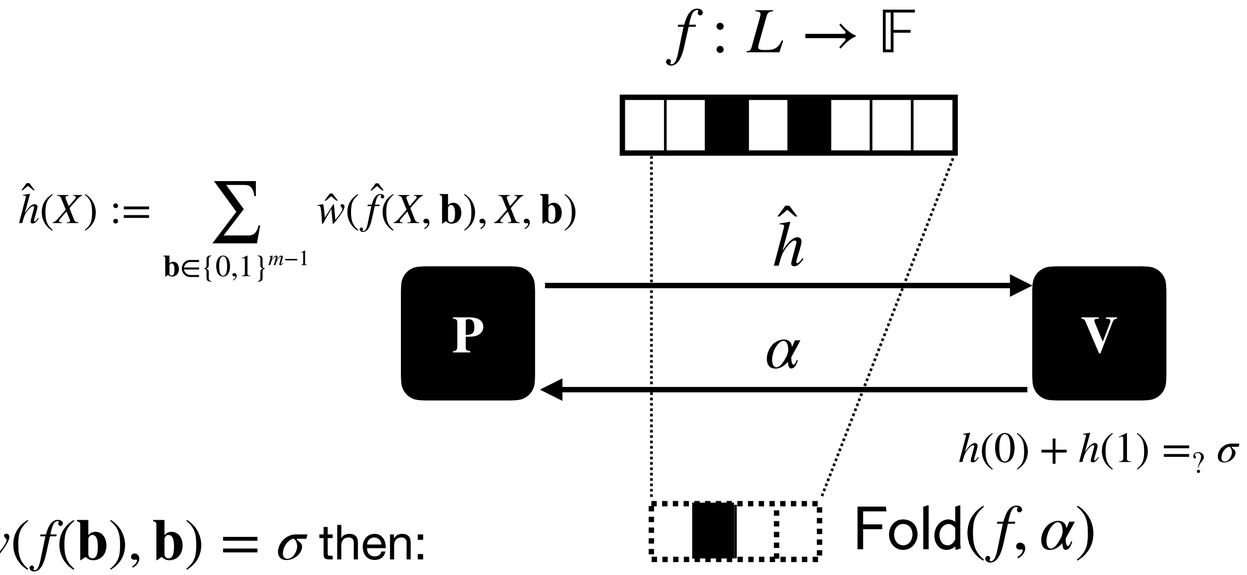
Completeness: $\sum_{\mathbf{b}} \hat{w}(f(\mathbf{b}), \mathbf{b}) = \sigma$ then:

- $h(0) + h(1) = \sigma$,
- $\sum_{\mathbf{b}} \hat{w}(f(\alpha, \mathbf{b}), \alpha, \mathbf{b}) = \hat{h}(\alpha)$

WHIR Folding

Interleave sumcheck with FRI folding, similar to BaseFold, Hyperplonk, Gemini

Reduce CRS$[n, m, \rho, \hat{w}, \sigma]$ **to** CRS$[n/2, m-1, \rho, \hat{w}_\alpha, \sigma_\alpha]$



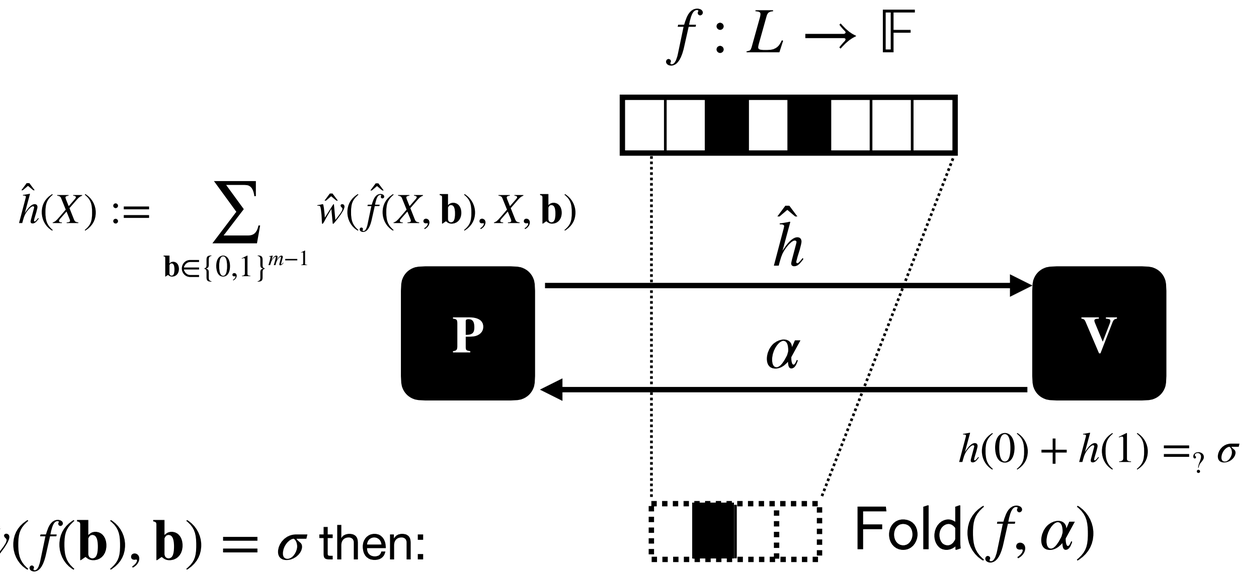
Completeness: $\sum_{\mathbf{b}} \hat{w}(f(\mathbf{b}), \mathbf{b}) = \sigma$ then:

- $h(0) + h(1) = \sigma,$
- $\sum_{\mathbf{b}} \hat{w}(f(\alpha, \mathbf{b}), \alpha, \mathbf{b}) = \hat{h}(\alpha)$
- $\widehat{\text{Fold}(f, \alpha)} = \hat{f}(\alpha, \cdot)$

WHIR Folding

Interleave sumcheck with FRI folding, similar to BaseFold, Hyperplonk, Gemini

Reduce CRS$[n, m, \rho, \hat{w}, \sigma]$ **to** CRS$[n/2, m-1, \rho, \hat{w}_\alpha, \sigma_\alpha]$



Completeness: $\sum_{\mathbf{b}} \hat{w}(f(\mathbf{b}), \mathbf{b}) = \sigma$ then:

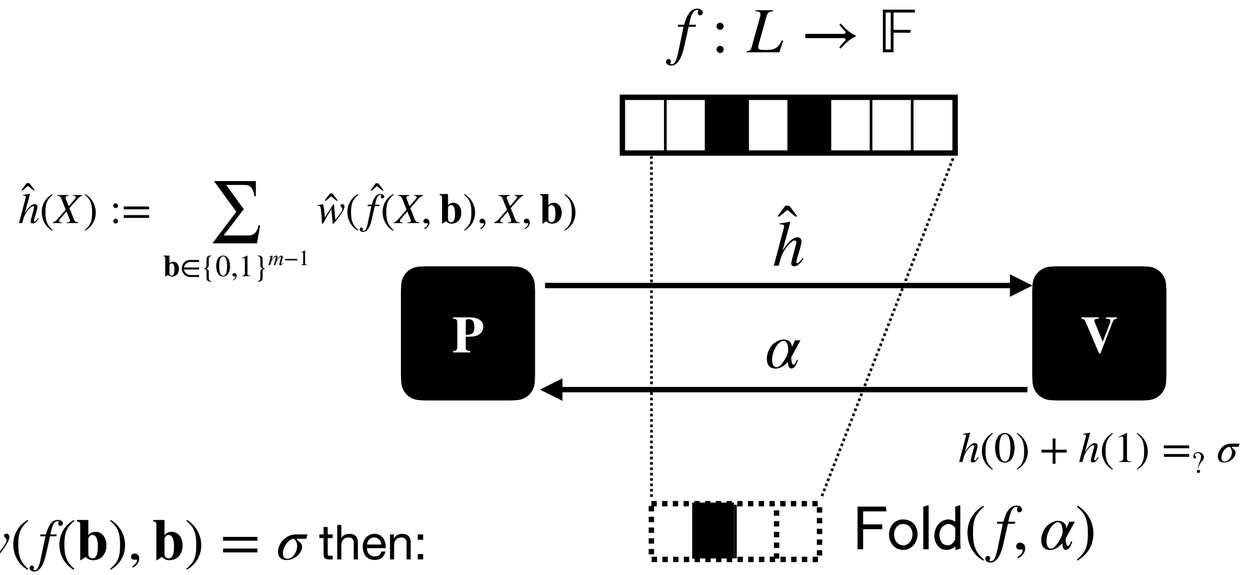
- $h(0) + h(1) = \sigma$,
- $\sum_{\mathbf{b}} \hat{w}(f(\alpha, \mathbf{b}), \alpha, \mathbf{b}) = \hat{h}(\alpha)$
- $\widehat{\text{Fold}(f, \alpha)} = \hat{f}(\alpha, \cdot)$

Soundness: by mutual correlated agreement, w.h.p. if $\Delta(f, \text{CRS}[n, m, \rho, \hat{w}, \sigma]) > \delta$ then $\Delta(\text{Fold}(f, \alpha), \text{CRS}[n/2, m-1, \rho, \hat{w}_\alpha, \hat{h}(\alpha)]) > \delta$

WHIR Folding

Interleave sumcheck with FRI folding, similar to BaseFold, Hyperplonk, Gemini

Reduce $\mathrm{CRS}[n, m, \rho, \hat{w}, \sigma]$ **to** $\mathrm{CRS}[n/2, m-1, \rho, \hat{w}_\alpha, \sigma_\alpha]$



Completeness: $\sum_{\mathbf{b}} \hat{w}(f(\mathbf{b}), \mathbf{b}) = \sigma$ then:

- $h(0) + h(1) = \sigma$,
- $\sum_{\mathbf{b}} \hat{w}(f(\alpha, \mathbf{b}), \alpha, \mathbf{b}) = \hat{h}(\alpha)$
- $\widehat{\mathrm{Fold}(f, \alpha)} = \hat{f}(\alpha, \cdot)$

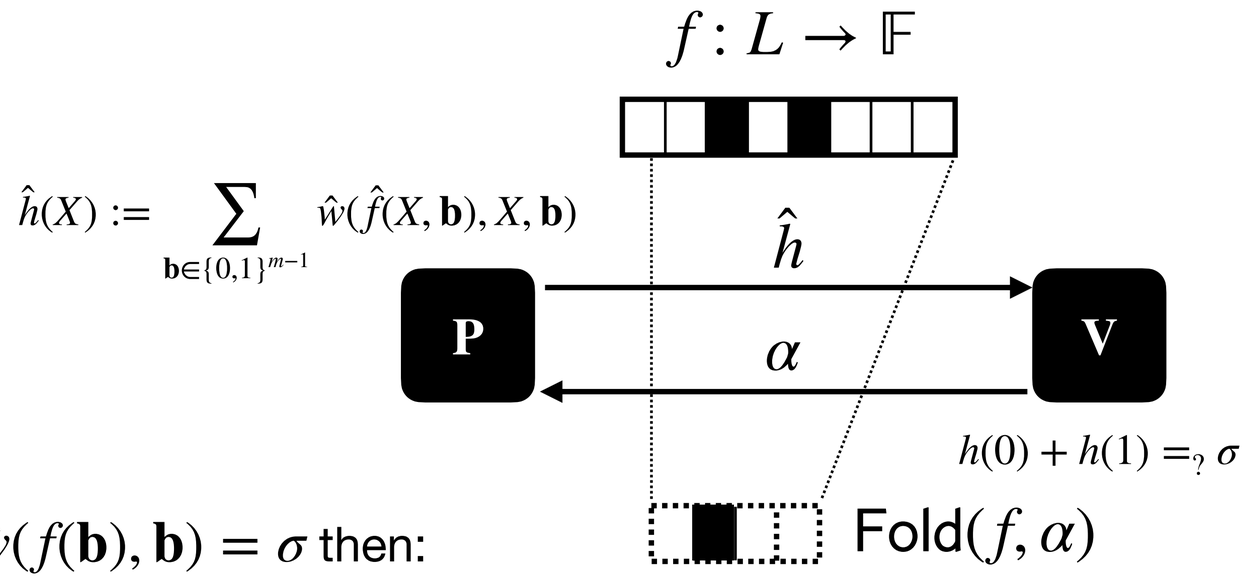
Soundness: by mutual correlated agreement, w.h.p. if $\Delta(f, \mathrm{CRS}[n, m, \rho, \hat{w}, \sigma]) > \delta$ then $\Delta(\mathrm{Fold}(f, \alpha), \mathrm{CRS}[n/2, m-1, \rho, \hat{w}_\alpha, \hat{h}(\alpha)]) > \delta$

$\hat{w}_\alpha(Z, \mathbf{X}) = \hat{w}(Z, \alpha, \mathbf{X})$

WHIR Folding

Interleave sumcheck with FRI folding, similar to BaseFold, Hyperplonk, Gemini

Reduce CRS$[n, m, \rho, \hat{w}, \sigma]$ **to** CRS$[n/2, m-1, \rho, \hat{w}_\alpha, \sigma_\alpha]$



Completeness: $\sum_{\mathbf{b}} \hat{w}(f(\mathbf{b}), \mathbf{b}) = \sigma$ then:

- $h(0) + h(1) = \sigma$,
- $\sum_{\mathbf{b}} \hat{w}(f(\alpha, \mathbf{b}), \alpha, \mathbf{b}) = \hat{h}(\alpha)$
- $\widehat{\text{Fold}(f, \alpha)} = \hat{f}(\alpha, \cdot)$

Soundness: by mutual correlated agreement, w.h.p. if $\Delta(f, \text{CRS}[n, m, \rho, \hat{w}, \sigma]) > \delta$ then $\Delta(\text{Fold}(f, \alpha), \text{CRS}[n/2, m-1, \rho, \hat{w}_\alpha, \hat{h}(\alpha)]) > \delta$

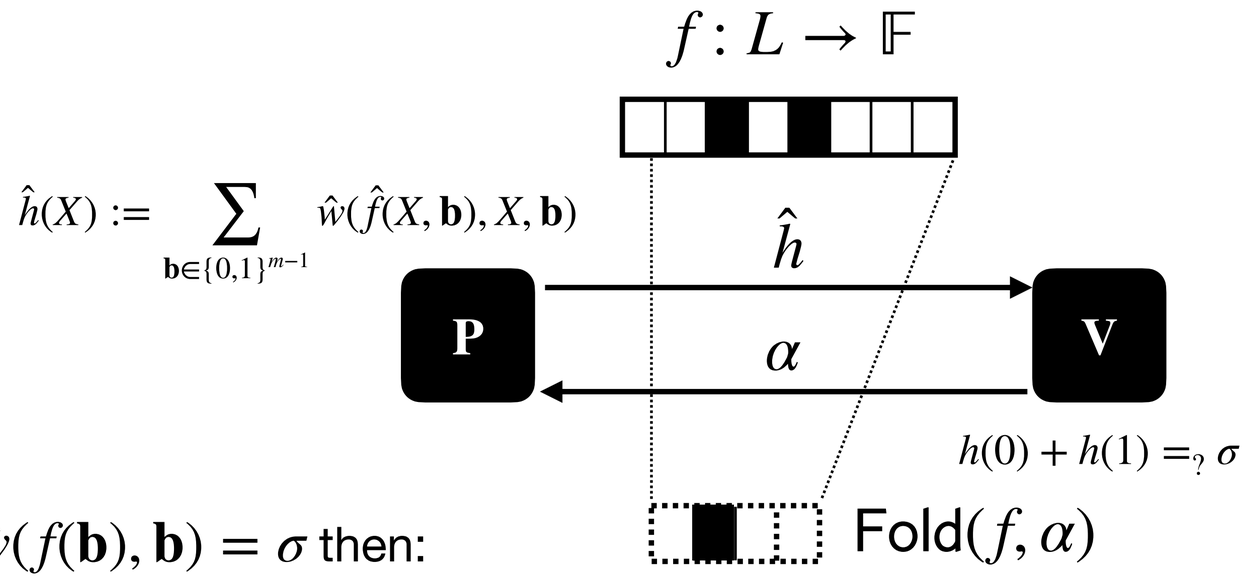
$\hat{w}_\alpha(Z, \mathbf{X}) = \hat{w}(Z, \alpha, \mathbf{X})$

Unchanged!

WHIR Folding

Interleave sumcheck with FRI folding, similar to BaseFold, Hyperplonk, Gemini

Reduce CRS$[n, m, \rho, \hat{w}, \sigma]$ **to** CRS$[n/2, m-1, \rho, \hat{w}_\alpha, \sigma_\alpha]$



In the full protocol, we fold by 2-by-2 k times. Can also fold by 2^k at a time (nice for first round!)

Completeness: $\sum_{\mathbf{b}} \hat{w}(f(\mathbf{b}), \mathbf{b}) = \sigma$ then:

- $h(0) + h(1) = \sigma$,
- $\sum_{\mathbf{b}} \hat{w}(f(\alpha, \mathbf{b}), \alpha, \mathbf{b}) = \hat{h}(\alpha)$
- $\widehat{\text{Fold}(f, \alpha)} = \hat{f}(\alpha, \cdot)$

Soundness: by mutual correlated agreement, w.h.p. if $\Delta(f, \text{CRS}[n, m, \rho, \hat{w}, \sigma]) > \delta$ then $\Delta(\text{Fold}(f, \alpha), \text{CRS}[n/2, m-1, \rho, \hat{w}_\alpha, \hat{h}(\alpha)]) > \delta$

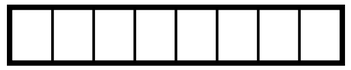
$\hat{w}_\alpha(Z, \mathbf{X}) = \hat{w}(Z, \alpha, \mathbf{X})$

Unchanged!

WHIR iteration

WHIR iteration

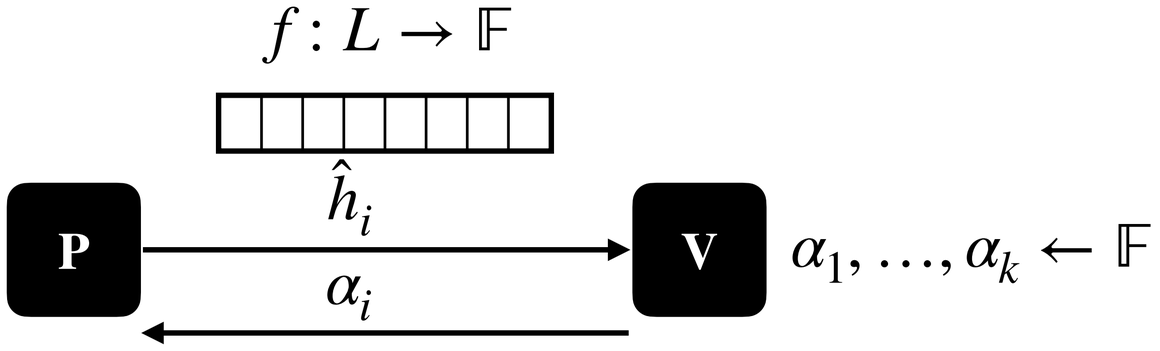
$f : L \to \mathbb{F}$



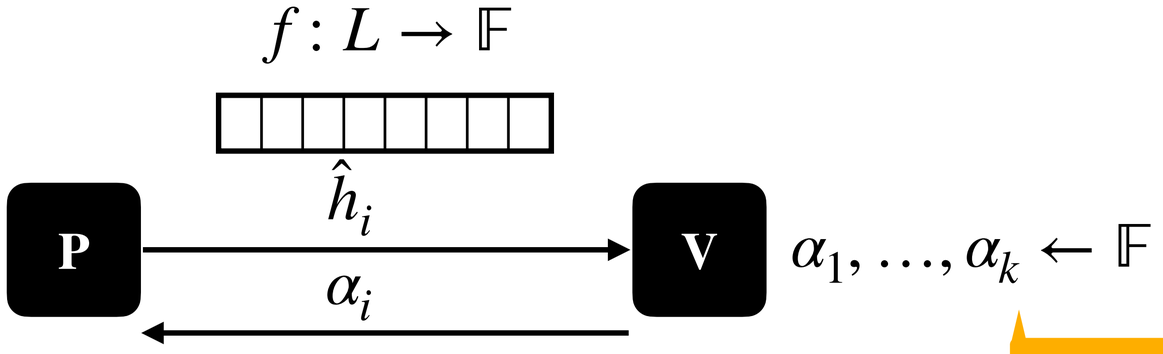
P

V

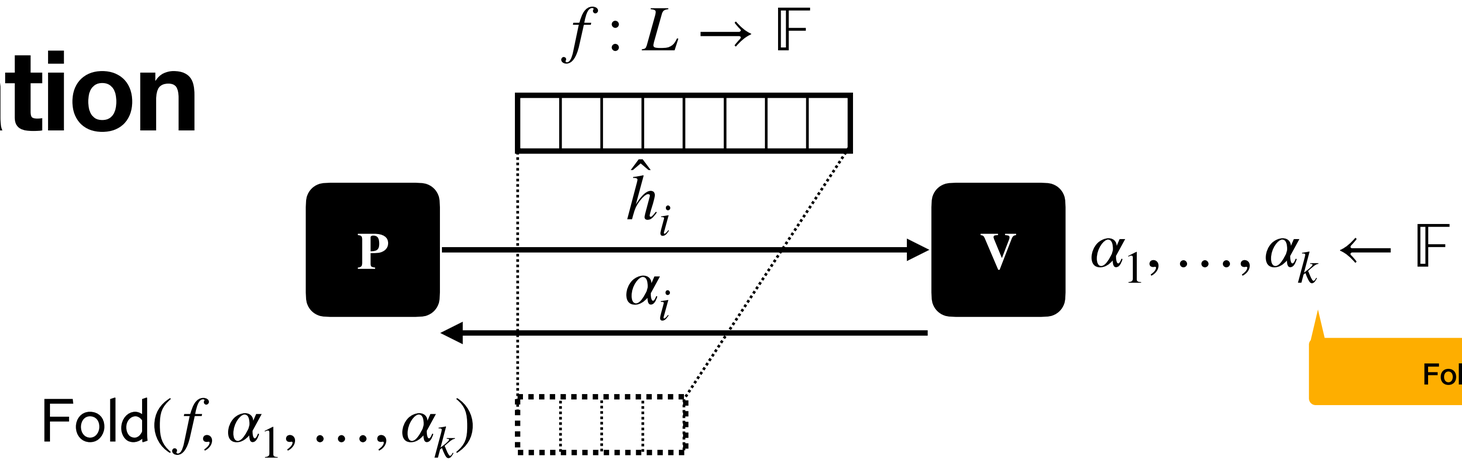
WHIR iteration



WHIR iteration

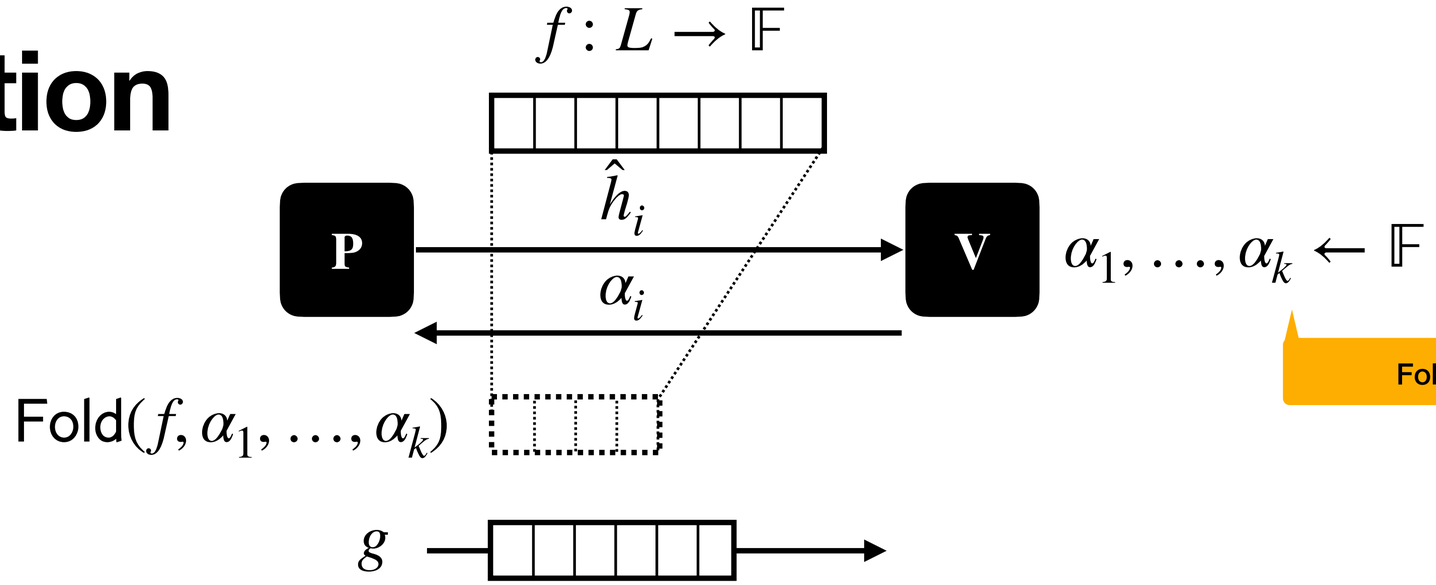


WHIR iteration



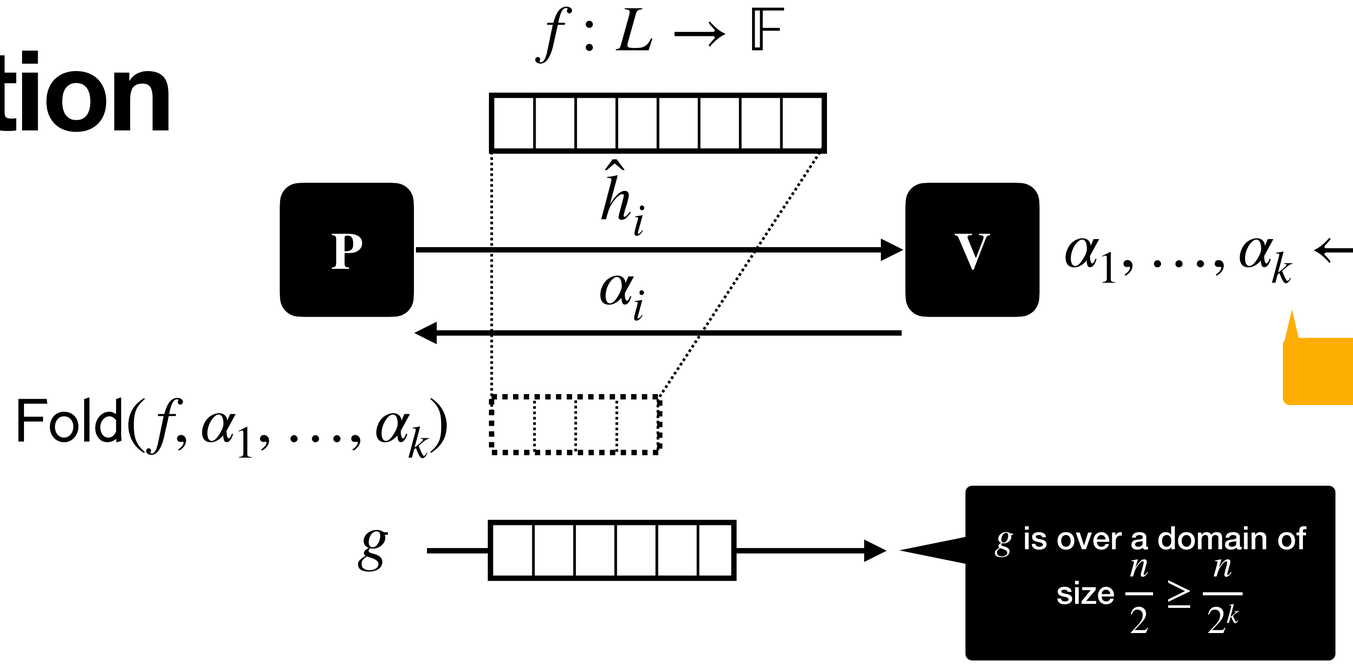
Folding k times by 2

WHIR iteration

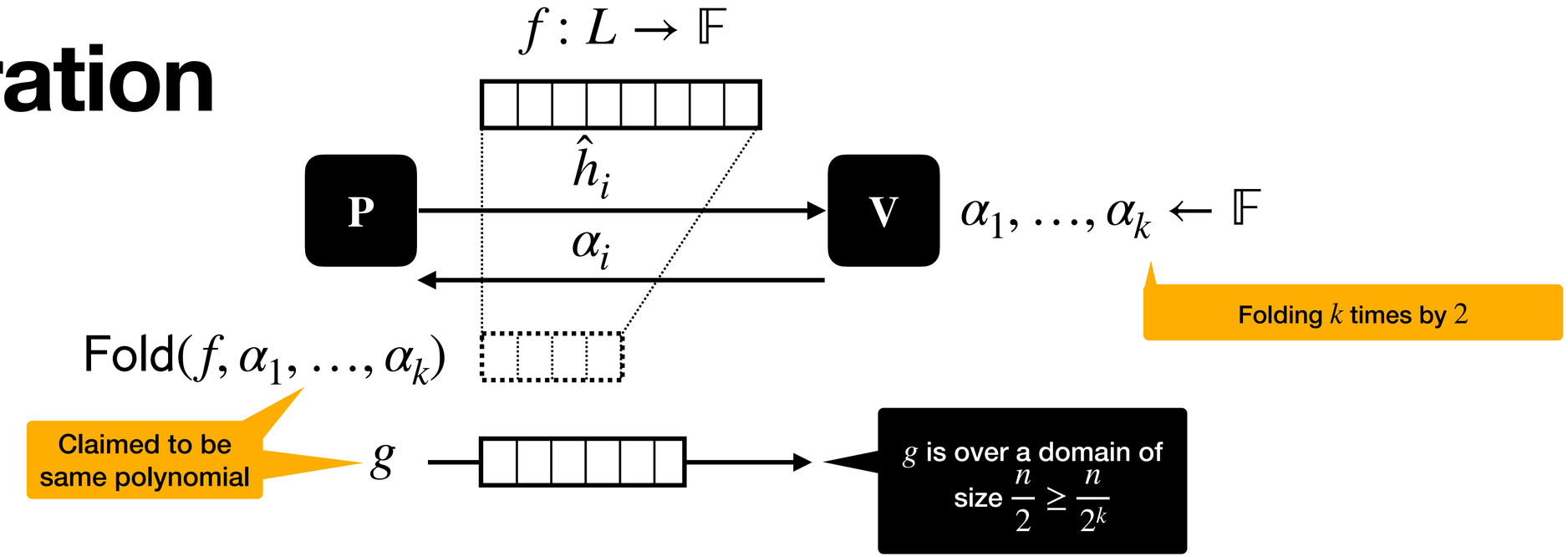


Folding k times by 2

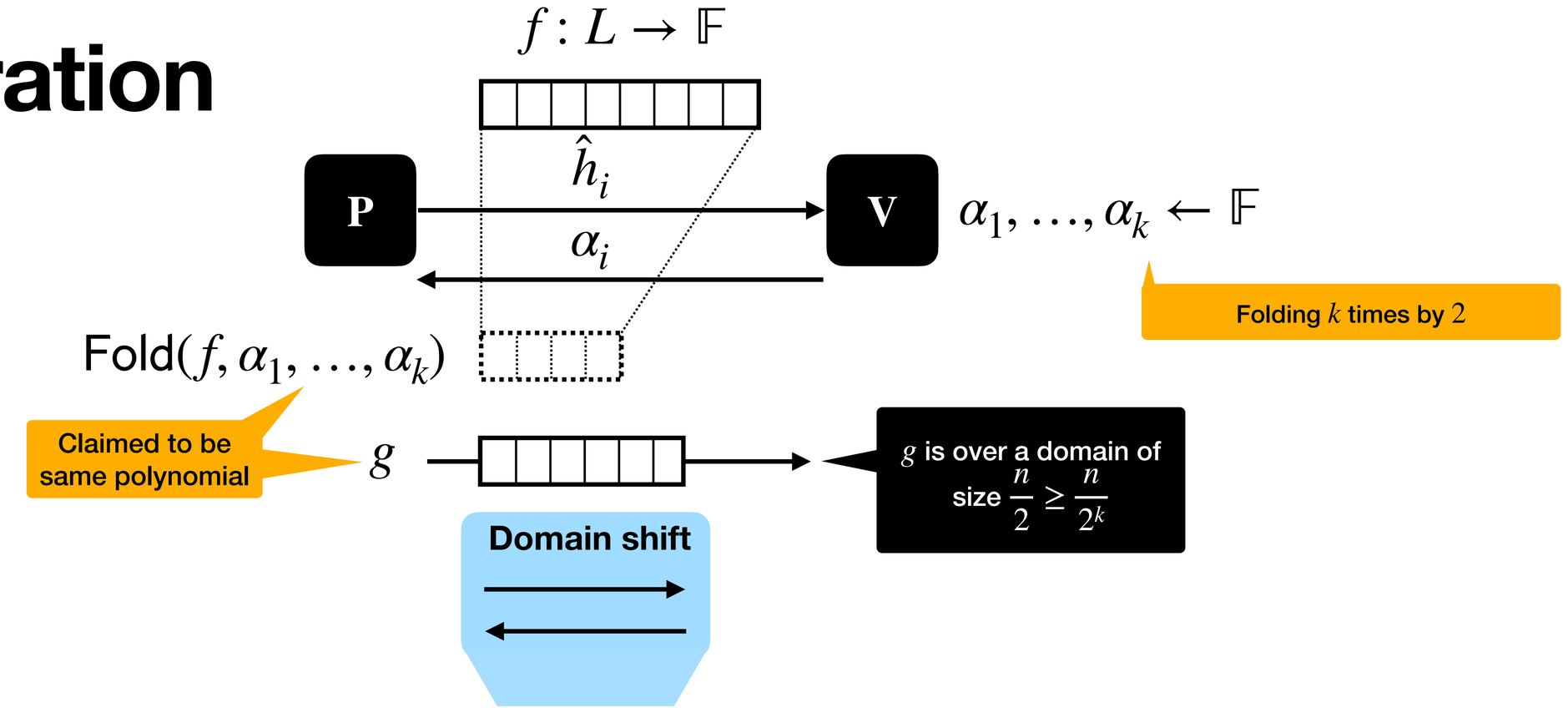
WHIR iteration



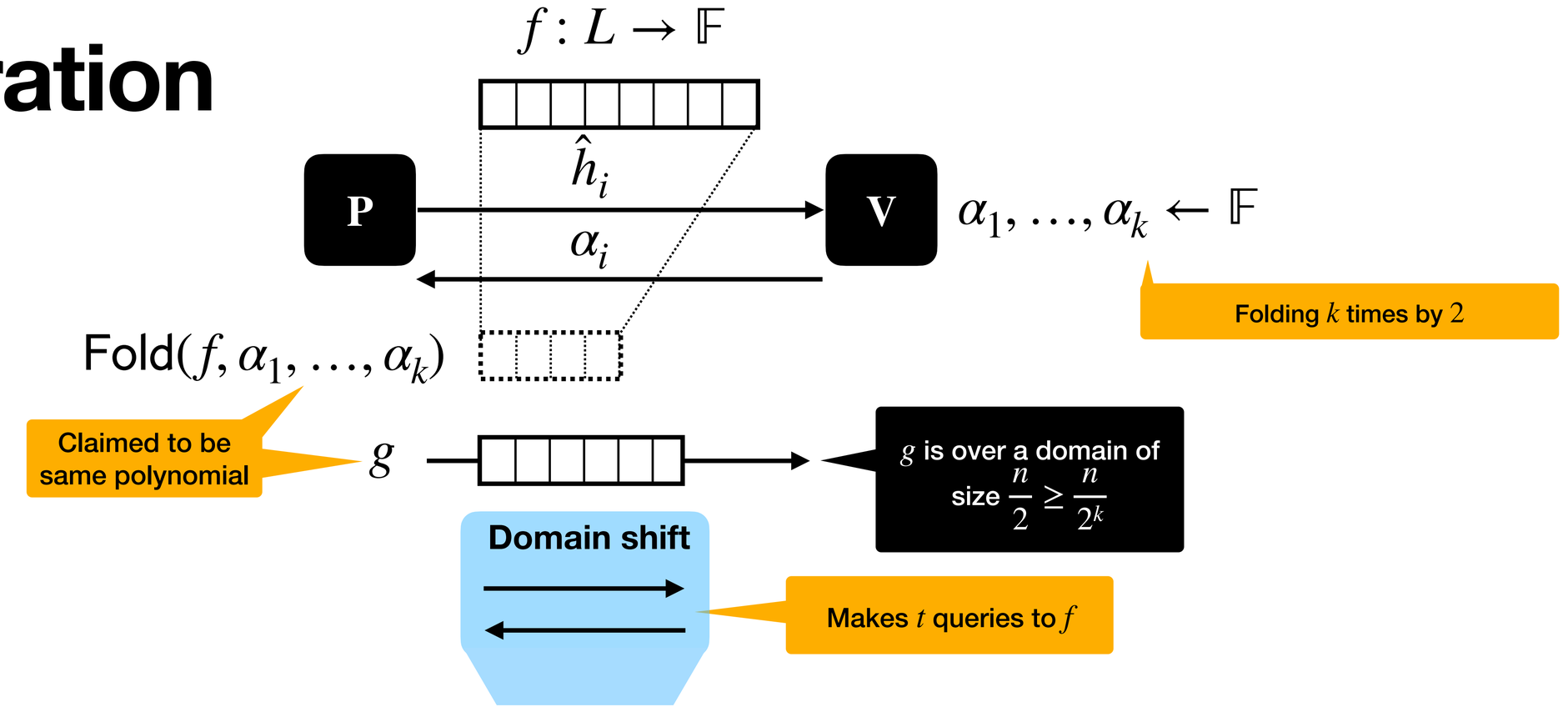
WHIR iteration



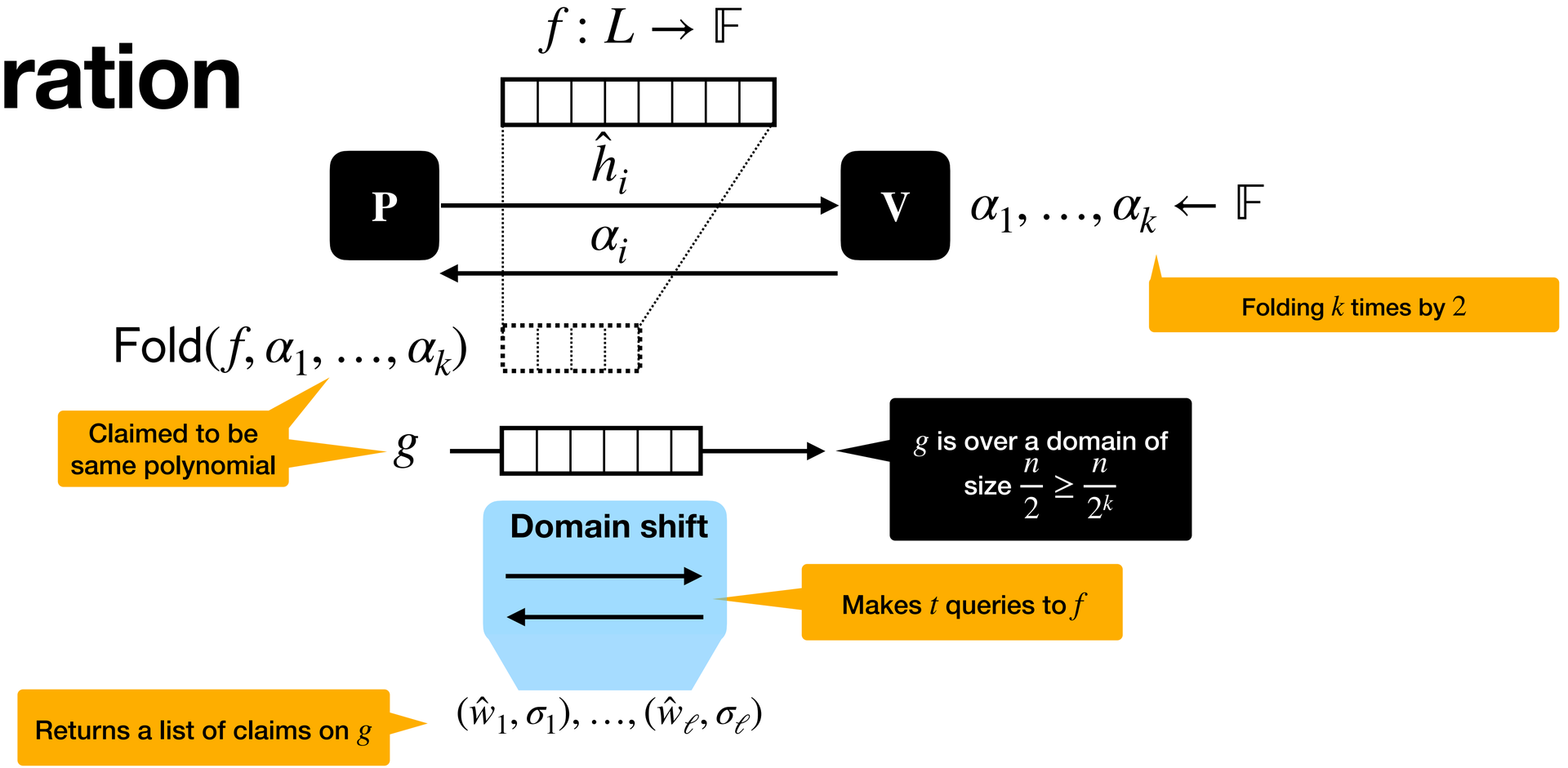
WHIR iteration



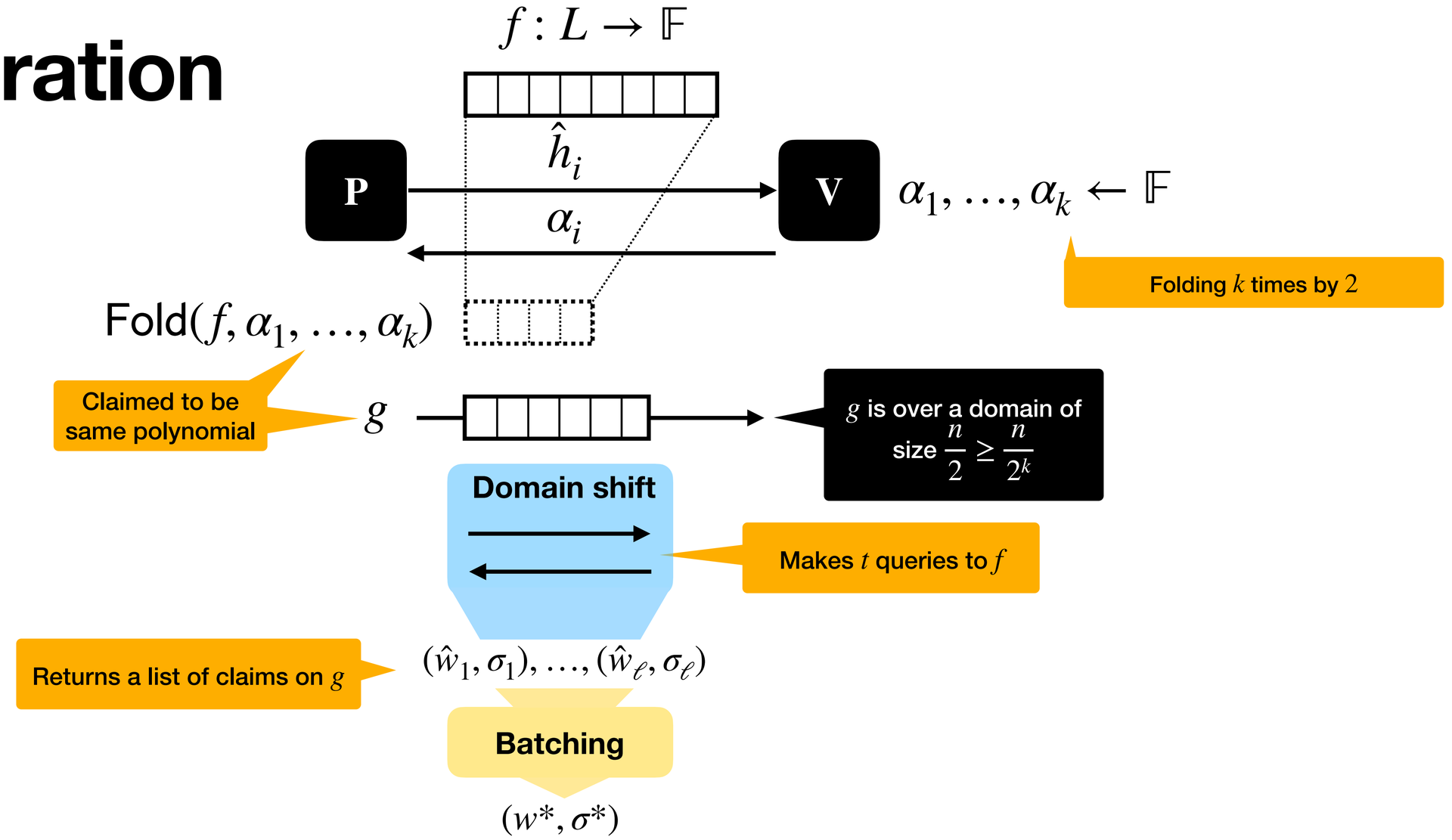
WHIR iteration



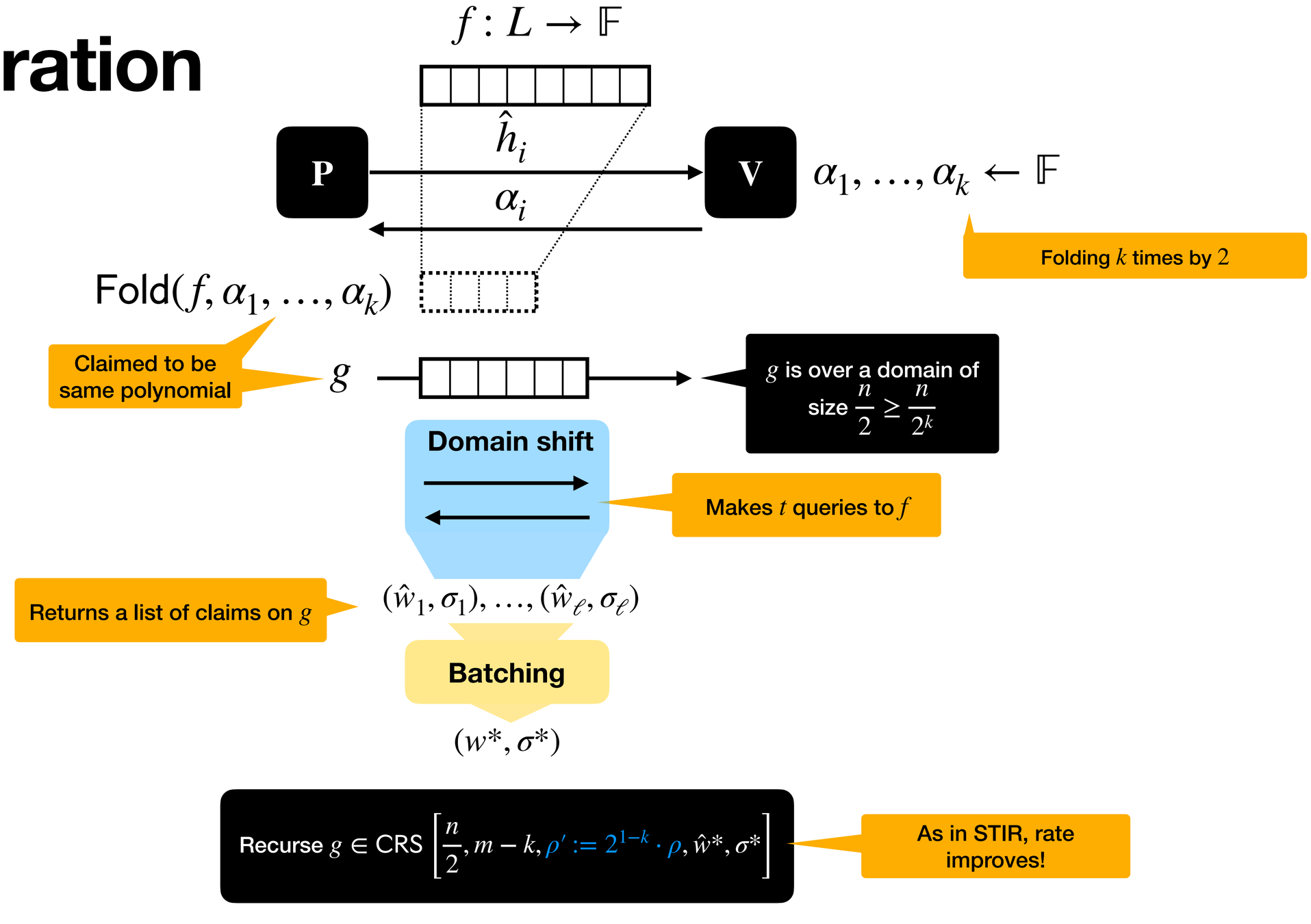
WHIR iteration



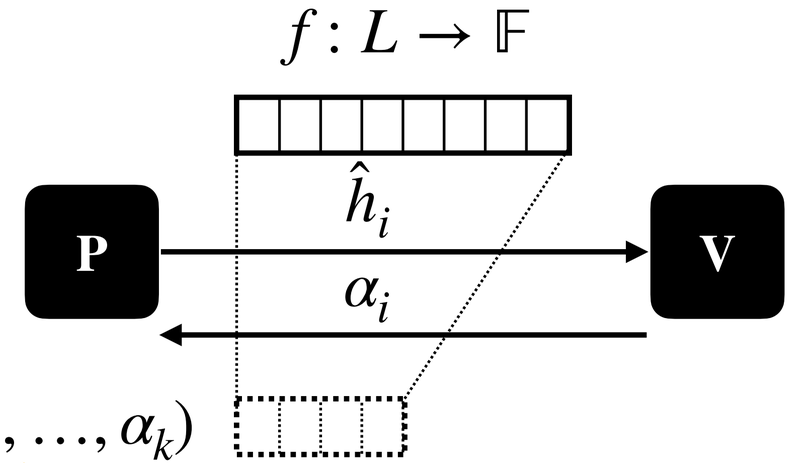
WHIR iteration



WHIR iteration



WHIR iteration



$\alpha_1, \ldots, \alpha_k \leftarrow \mathbb{F}$

Folding k times by 2

$\text{Fold}(f, \alpha_1, \ldots, \alpha_k)$

Claimed to be same polynomial

g

g is over a domain of size $\frac{n}{2} \geq \frac{n}{2^k}$

Domain shift

Makes t queries to f

Returns a list of claims on g

$(\hat{w}_1, \sigma_1), \ldots, (\hat{w}_\ell, \sigma_\ell)$

Batching

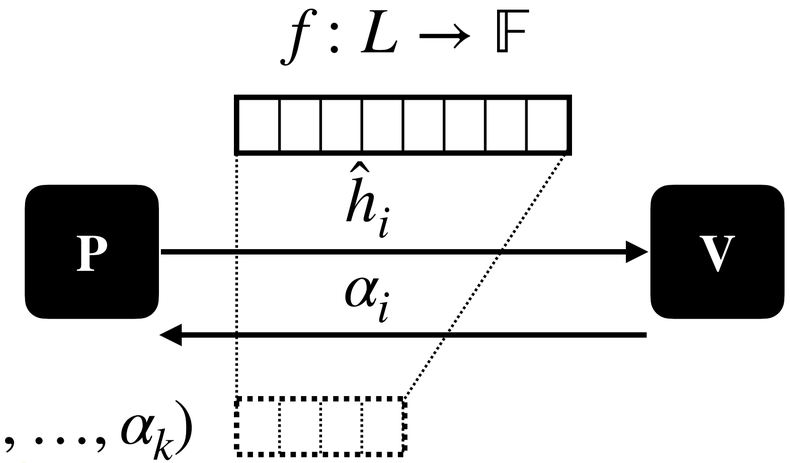
(w^*, σ^*)

Recurse $g \in \text{CRS}\left[\frac{n}{2}, m-k, \rho' := 2^{1-k} \cdot \rho, \hat{w}^*, \sigma^*\right]$

As in STIR, rate improves!

Similar structure to STIR! Multilinear structure **forbids** using quotients: we need new ideas to domain shift!

WHIR iteration



Folding k times by 2

Claimed to be same polynomial

g

g is over a domain of size $\frac{n}{2} \geq \frac{n}{2^k}$

Domain shift

Makes t queries to f

Returns a list of claims on g

$(\hat{w}_1, \sigma_1), \ldots, (\hat{w}_\ell, \sigma_\ell)$

Batching

Not shown

(w^*, σ^*)

Recurse $g \in \mathrm{CRS}\left[\frac{n}{2}, m-k, \rho' := 2^{1-k} \cdot \rho, \hat{w}^*, \sigma^*\right]$

As in STIR, rate improves!

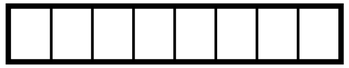
Similar structure to STIR! Multilinear structure **forbids** using quotients: we need new ideas to domain shift!

Domain shifting

Domain shifting

Claim on f: $(\hat{w}, \sigma)$

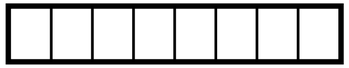
$$f : L \to \mathbb{F}$$



Domain shifting

Claim on f: $(\hat{w}, \sigma)$

$f : L \to \mathbb{F}$



$g : L^* \to \mathbb{F}$

Domain shifting

Claim on f: $(\hat{w}, \sigma)$

$f : L \to \mathbb{F}$

$g : L^* \to \mathbb{F}$

Output claims on g:
$(\hat{w}_1, \sigma_1), \ldots, (\hat{w}_\ell, \sigma_\ell)$

Domain shifting

Claim on f: $(\hat{w}, \sigma)$

$f : L \to \mathbb{F}$

$g : L^* \to \mathbb{F}$

Output claims on g:
$(\hat{w}_1, \sigma_1), \ldots, (\hat{w}_\ell, \sigma_\ell)$

f and g claimed to be evaluations of same polynomial. Want to output **claims** on g.

Goal: If f is $\left(1 - \sqrt{\rho}\right)$-far from CRS$[|L|, m, \rho, \hat{w}, \sigma]$, w.h.p. g is $\left(1 - \sqrt{\rho'}\right)$-far from CRS$[|L^*|, m, \rho', \hat{w}_i, \sigma_i]$ for at least one $i \in [\ell]$

Domain shifting

Claim on f: $(\hat{w}, \sigma)$

$f : L \to \mathbb{F}$

$g : L^* \to \mathbb{F}$

Output claims on g:
$(\hat{w}_1, \sigma_1), \ldots, (\hat{w}_\ell, \sigma_\ell)$

f and g claimed to be evaluations of same polynomial. Want to output **claims** on g.

Goal: If f is $\left(1 - \sqrt{\rho}\right)$-far from CRS$[|L|, m, \rho, \hat{w}, \sigma]$, w.h.p. g is $\left(1 - \sqrt{\rho'}\right)$-far from CRS$[|L^*|, m, \rho', \hat{w}_i, \sigma_i]$ for at least one $i \in [\ell]$

Assume there is unique polynomial $\hat{p}$ that is $\left(1 - \sqrt{\rho'}\right)$-close to g.

OOD subprotocol (next)

Domain shifting

Claim on f: $(\hat{w}, \sigma)$

$f : L \to \mathbb{F}$

$g : L^* \to \mathbb{F}$

Output claims on g:
$(\hat{w}_1, \sigma_1), \ldots, (\hat{w}_\ell, \sigma_\ell)$

f and g claimed to be evaluations of same polynomial. Want to output **claims** on g.

Goal: If f is $\left(1 - \sqrt{\rho}\right)$-far from CRS$[|L|, m, \rho, \hat{w}, \sigma]$, w.h.p. g is $\left(1 - \sqrt{\rho'}\right)$-far from CRS$[|L^*|, m, \rho', \hat{w}_i, \sigma_i]$ for at least one $i \in [\ell]$

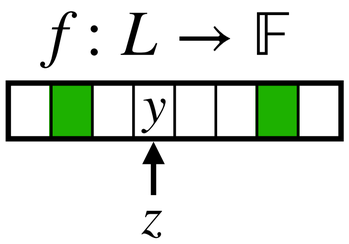
Assume there is unique polynomial $\hat{p}$ that is $\left(1 - \sqrt{\rho'}\right)$-close to g.

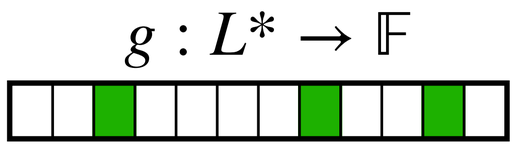
OOD subprotocol (next)

Then, if $\hat{p}$ satisfies the $(\hat{w}, \sigma)$-constraint f must be be $\left(1 - \sqrt{\rho}\right)$-far from it.

Domain shifting

Claim on f: $(\hat{w}, \sigma)$





Output claims on g:
$(\hat{w}_1, \sigma_1), \ldots, (\hat{w}_\ell, \sigma_\ell)$

f and g claimed to be evaluations of same polynomial. Want to output **claims** on g.

Goal: If f is $\left(1 - \sqrt{\rho}\right)$-far from CRS$[|L|, m, \rho, \hat{w}, \sigma]$, w.h.p. g is $\left(1 - \sqrt{\rho'}\right)$-far from CRS$[|L^*|, m, \rho', \hat{w}_i, \sigma_i]$ for at least one $i \in [\ell]$

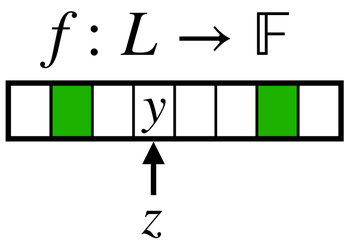
Assume there is unique polynomial $\hat{p}$ that is $\left(1 - \sqrt{\rho'}\right)$-close to g.

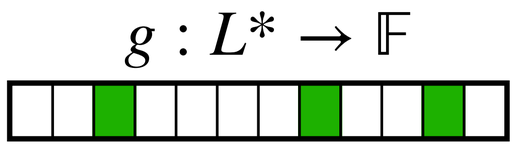
OOD subprotocol (next)

Then, if $\hat{p}$ satisfies the $(\hat{w}, \sigma)$-constraint f must be be $\left(1 - \sqrt{\rho}\right)$-far from it.

Domain shifting

Claim on f: $(\hat{w}, \sigma)$





Output claims on g:
$(\hat{w}_1, \sigma_1), \ldots, (\hat{w}_\ell, \sigma_\ell)$

f and g claimed to be evaluations of same polynomial. Want to output **claims** on g.

Goal: If f is $\left(1 - \sqrt{\rho}\right)$-far from CRS$[|L|, m, \rho, \hat{w}, \sigma]$, w.h.p. g is $\left(1 - \sqrt{\rho'}\right)$-far from CRS$[|L^*|, m, \rho', \hat{w}_i, \sigma_i]$ for at least one $i \in [\ell]$

Assume there is unique polynomial $\hat{p}$ that is $\left(1 - \sqrt{\rho'}\right)$-close to g.

OOD subprotocol (next)

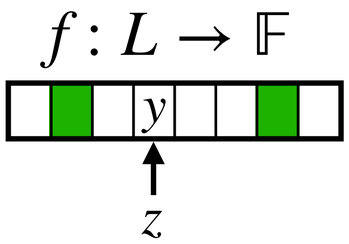
Then, if $\hat{p}$ satisfies the $(\hat{w}, \sigma)$-constraint f must be be $\left(1 - \sqrt{\rho}\right)$-far from it.

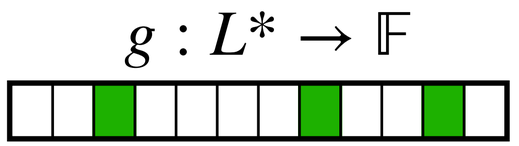
New constraints: (i) original constraint $(\hat{w}, \sigma)$ (ii) $\hat{p}(z) = y$ for some random point z.

Just an evaluation constraint which we know how to handle!

Domain shifting

Claim on f: $(\hat{w}, \sigma)$





Output claims on g:
$(\hat{w}_1, \sigma_1), \ldots, (\hat{w}_\ell, \sigma_\ell)$

f and g claimed to be evaluations of same polynomial. Want to output **claims** on g.

Goal: If f is $\left(1 - \sqrt{\rho}\right)$-far from CRS$[|L|, m, \rho, \hat{w}, \sigma]$, w.h.p. g is $\left(1 - \sqrt{\rho'}\right)$-far from CRS$[|L^*|, m, \rho', \hat{w}_i, \sigma_i]$ for at least one $i \in [\ell]$

Assume there is unique polynomial $\hat{p}$ that is $\left(1 - \sqrt{\rho'}\right)$-close to g.

OOD subprotocol (next)

Then, if $\hat{p}$ satisfies the $(\hat{w}, \sigma)$-constraint f must be be $\left(1 - \sqrt{\rho}\right)$-far from it.

New constraints: (i) original constraint $(\hat{w}, \sigma)$ (ii) $\hat{p}(z) = y$ for some random point z.

Just an evaluation constraint which we know how to handle!

So, except with probability $\sqrt{\rho}$, g is $\left(1 - \sqrt{\rho'}\right)$-far from CRS$[|L^*|, m, \rho', (\hat{w}_1, \sigma_1), \ldots, (\hat{w}_\ell, \sigma_\ell)]$.

Can amplify to $\sqrt{\rho}^t$

Out Of Domain

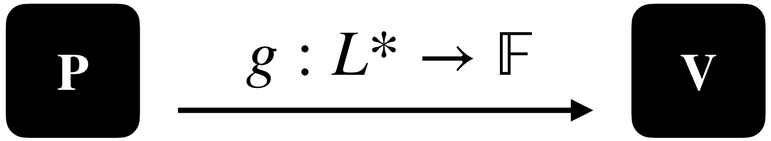
Subprotocol to force unique

Out Of Domain

Subprotocol to force unique

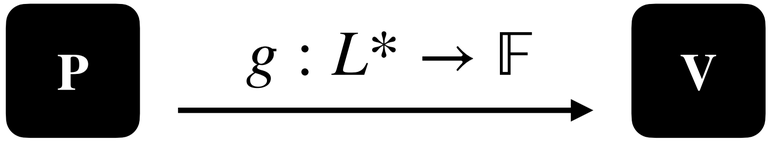
Out Of Domain

Subprotocol to force unique



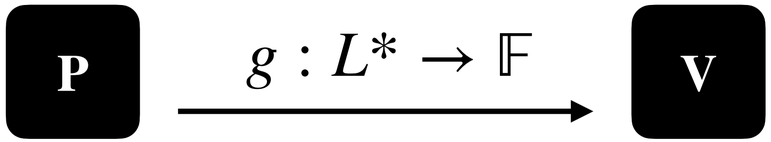
Out Of Domain

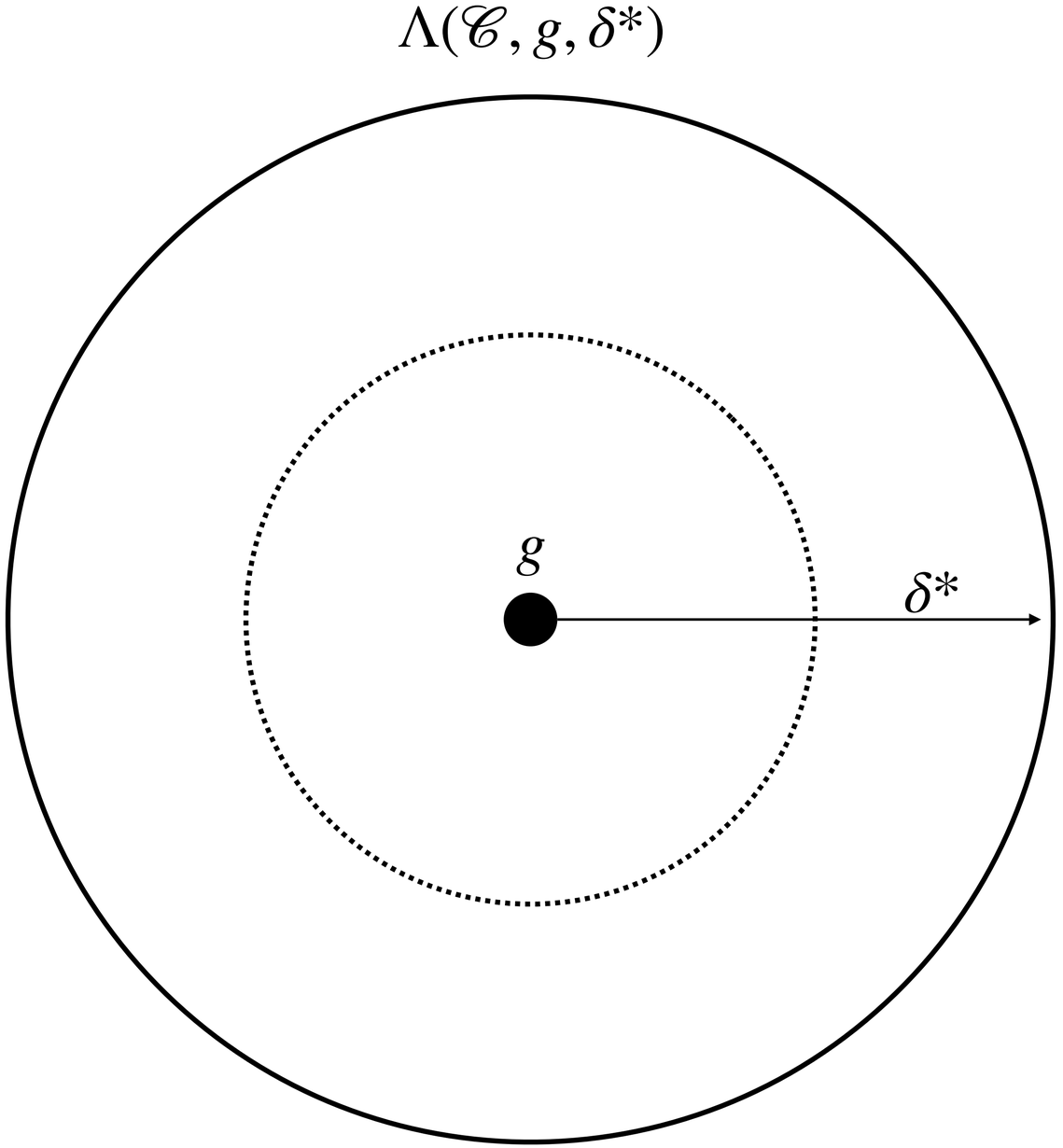
Subprotocol to force unique



Out Of Domain

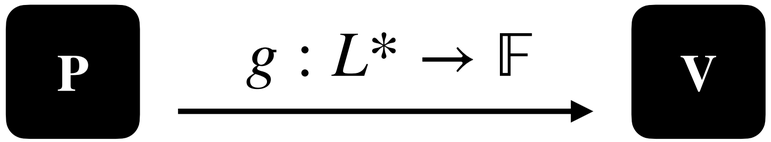
Subprotocol to force unique

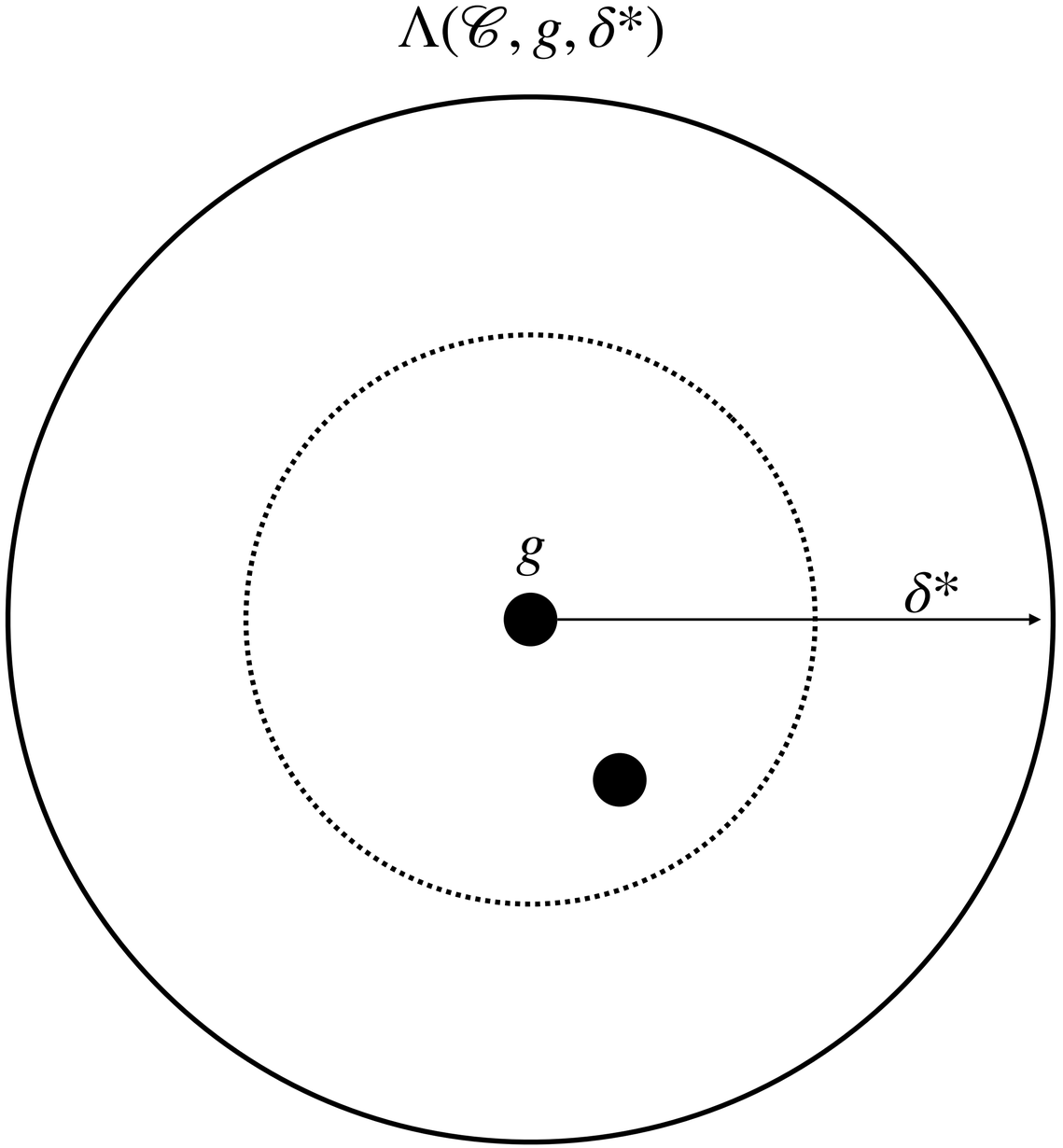




Out Of Domain

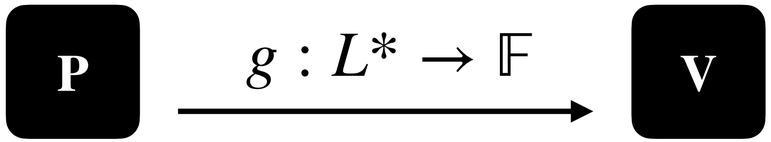
Subprotocol to force unique

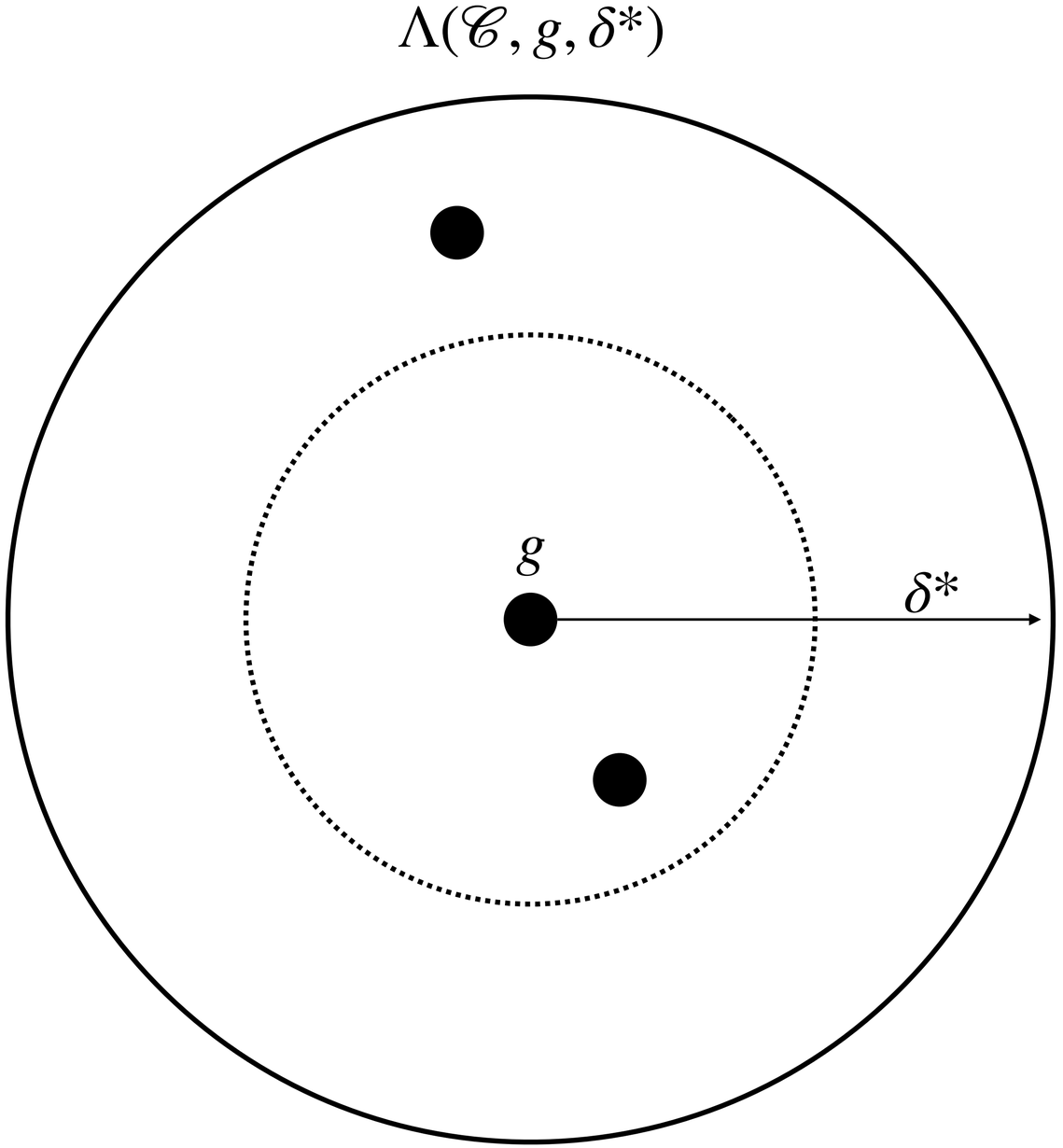




Out Of Domain

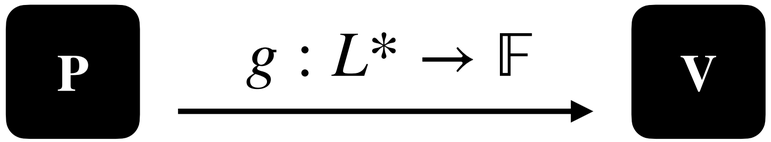
Subprotocol to force unique

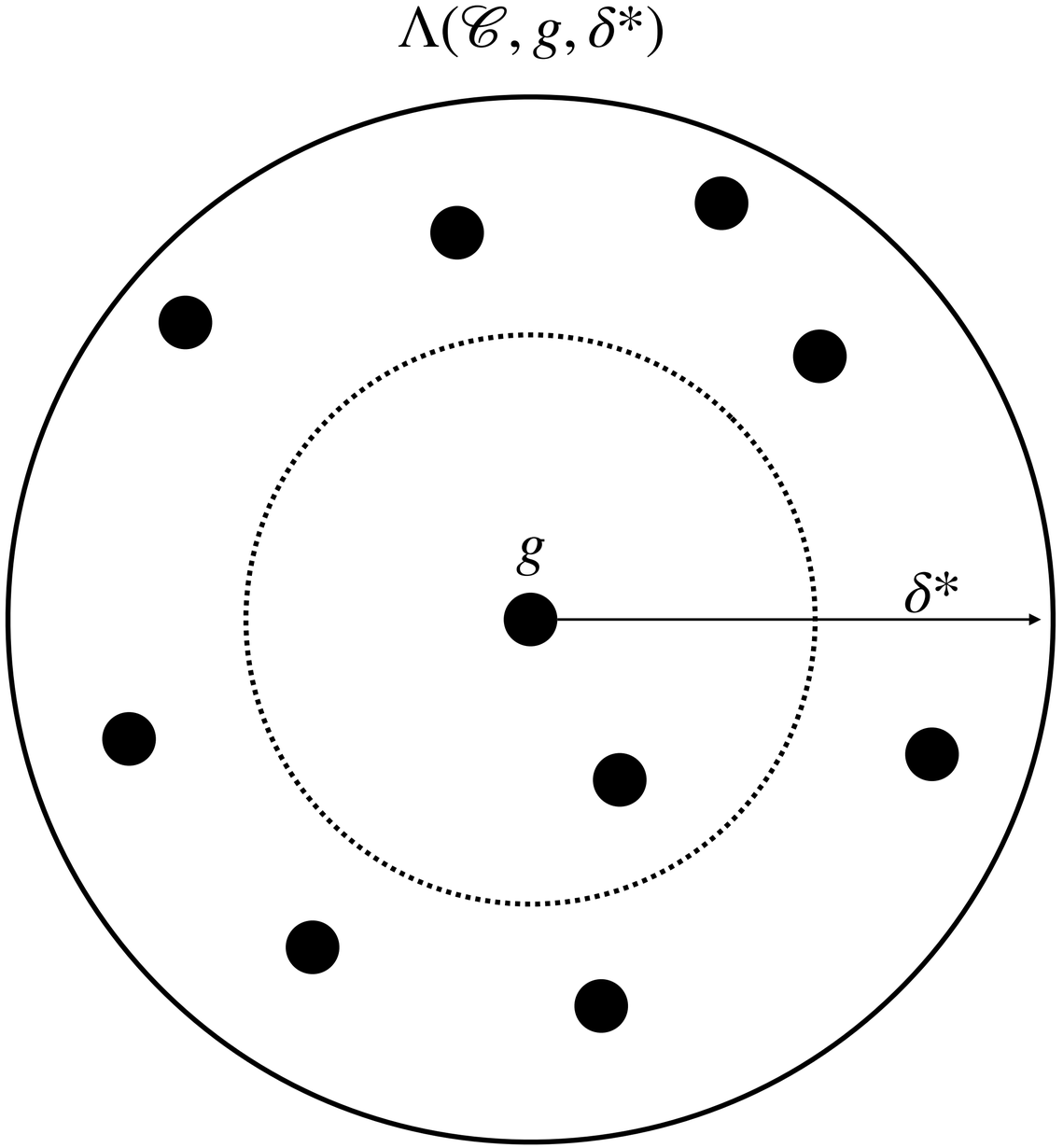




Out Of Domain

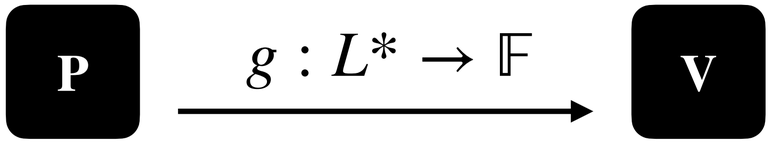
Subprotocol to force unique

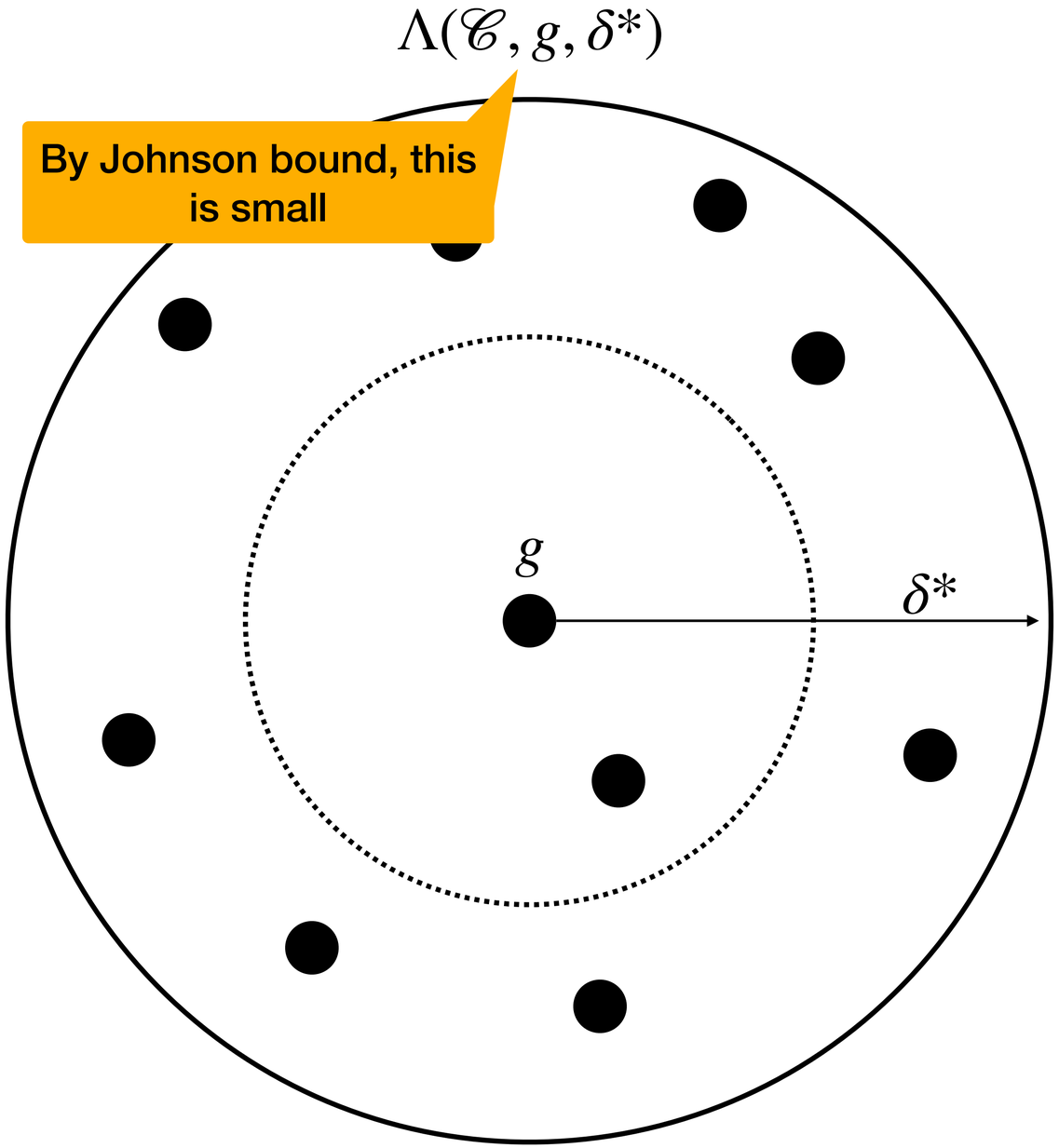




Out Of Domain

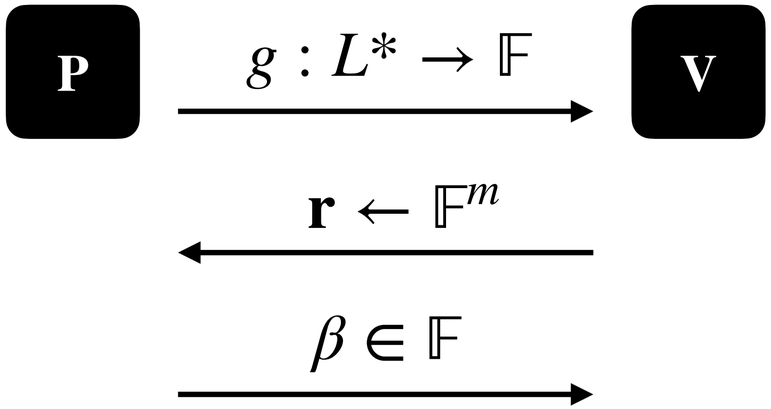
Subprotocol to force unique

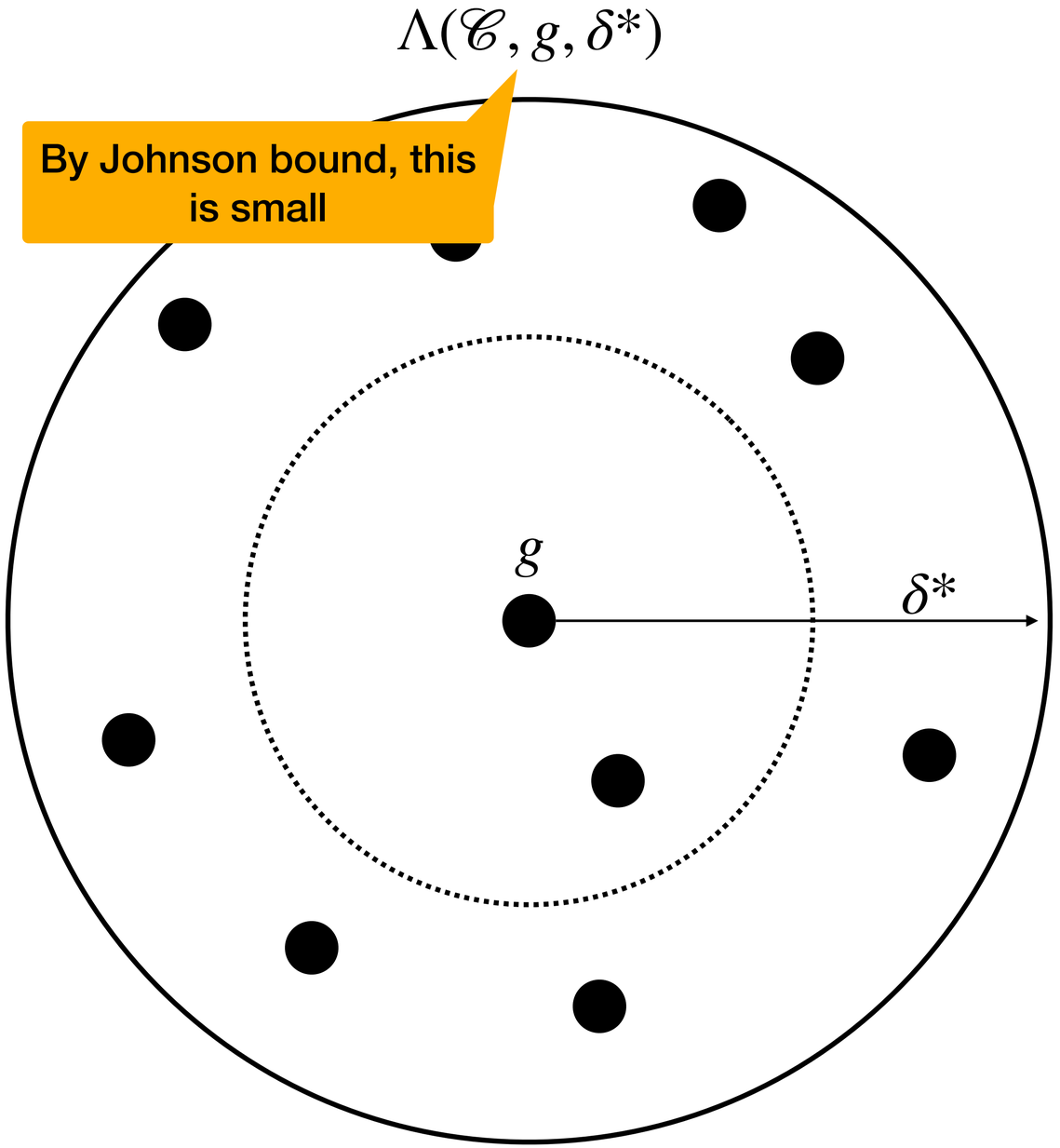




Out Of Domain

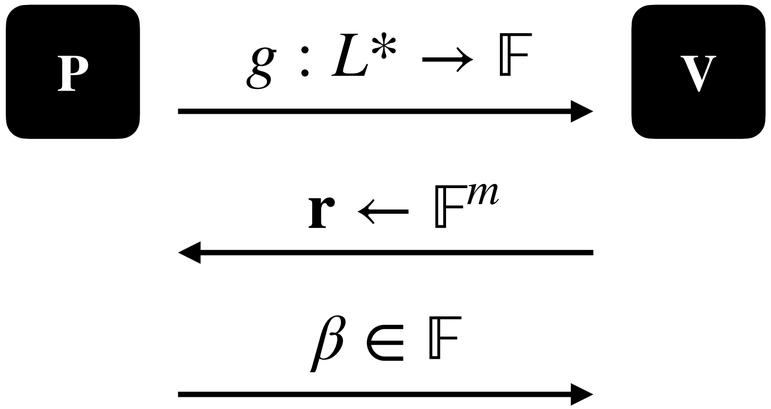
Subprotocol to force unique



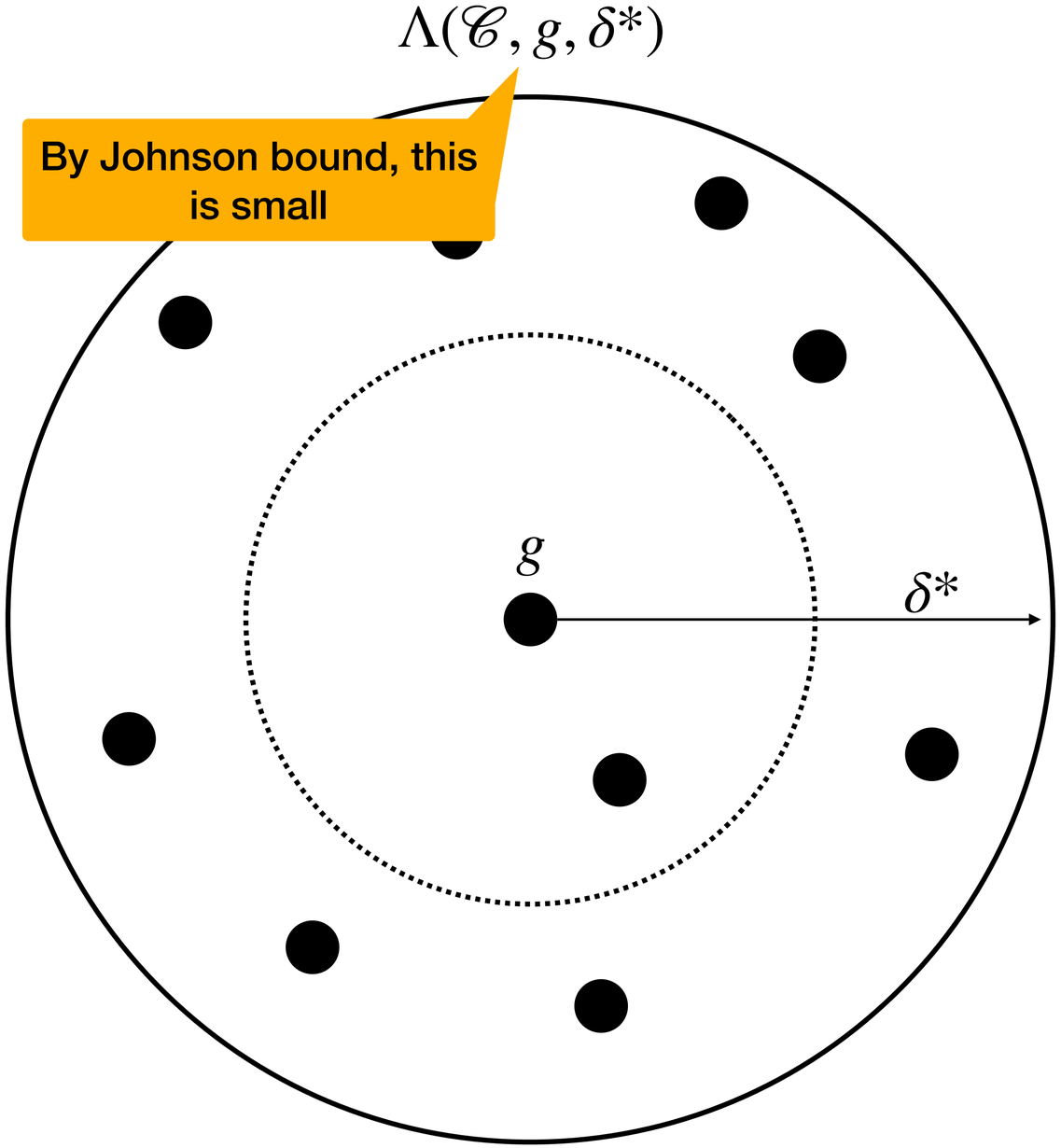


Out Of Domain

Subprotocol to force unique

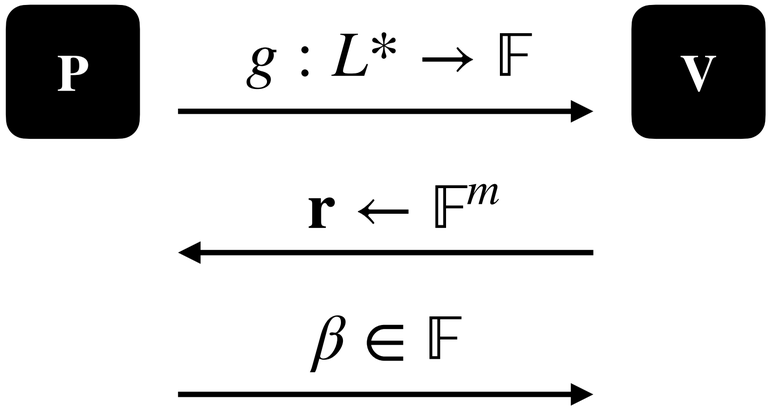


- By SZ lemma w.h.p.
  no pair $\hat{u}, \hat{v}$ with $\hat{u}(\mathbf{r}) = \hat{v}(\mathbf{r})$
- Prover "chooses" which codeword $\hat{u}$ it "commits" to

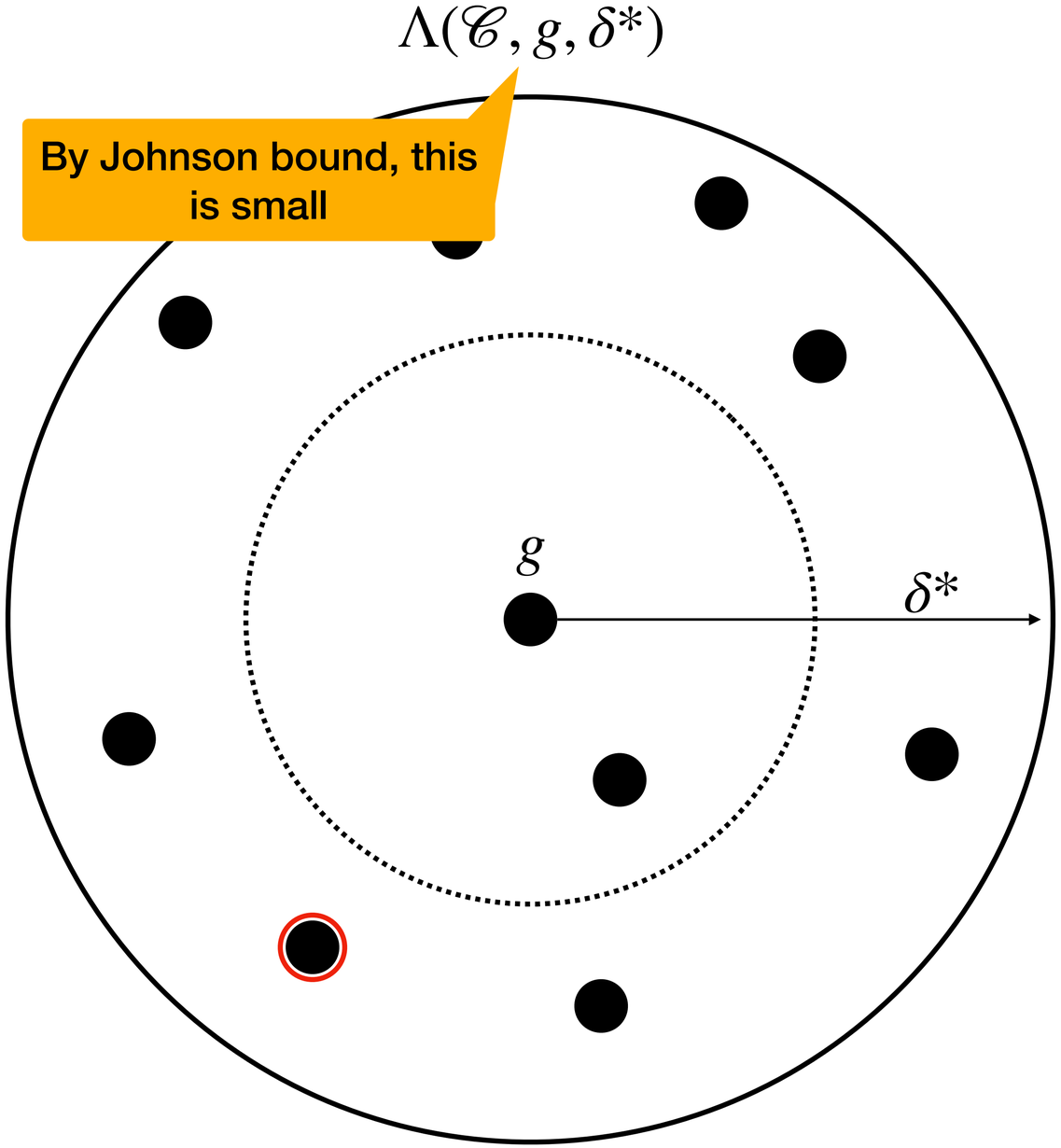


Out Of Domain

Subprotocol to force unique

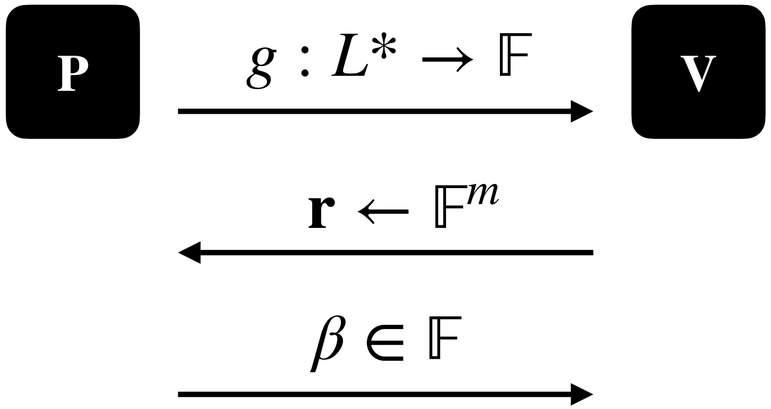


- By SZ lemma w.h.p.
  no pair $\hat{u}, \hat{v}$ with $\hat{u}(\mathbf{r}) = \hat{v}(\mathbf{r})$
- Prover "chooses" which codeword $\hat{u}$ it "commits" to

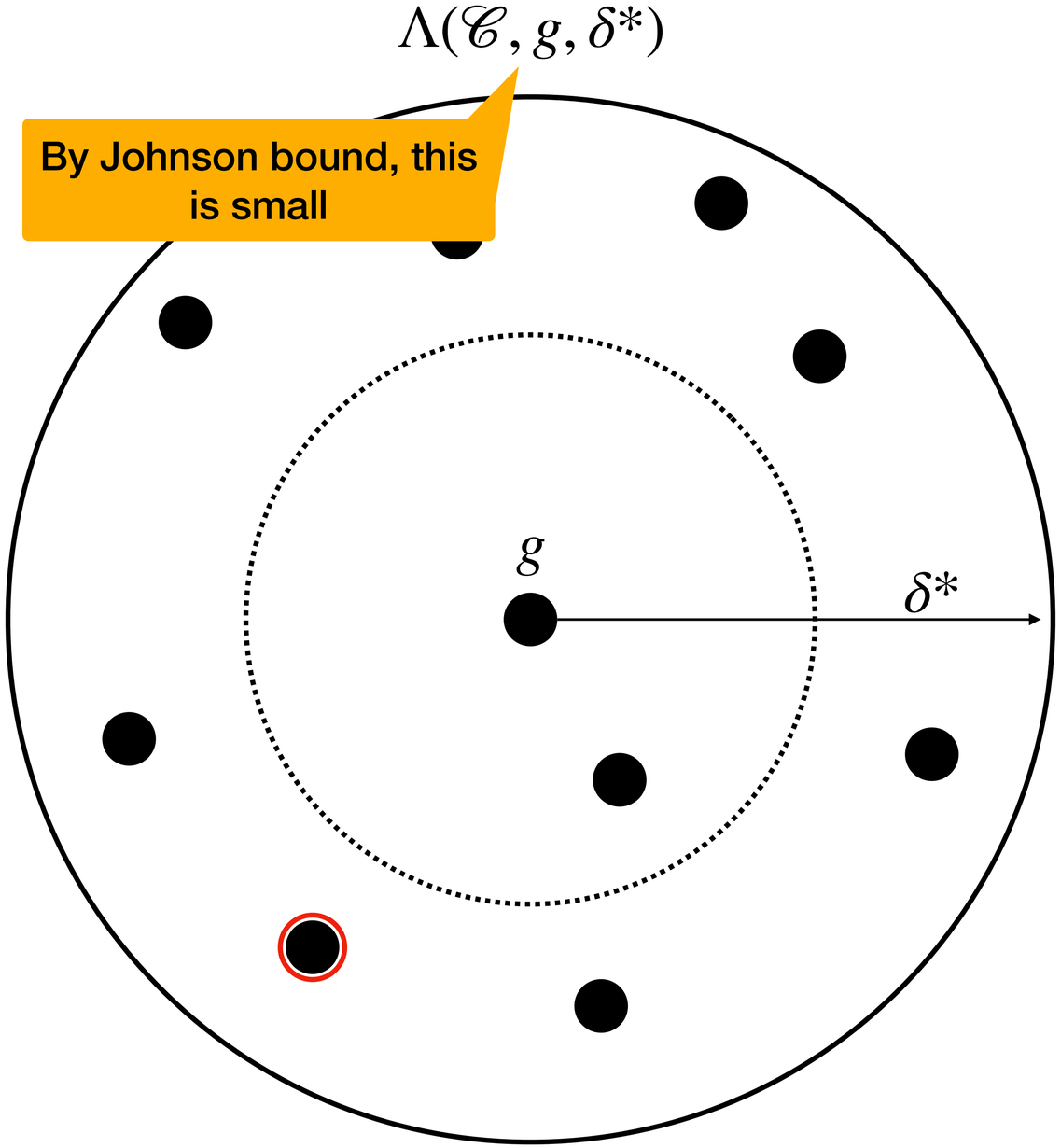


Out Of Domain

Subprotocol to force unique



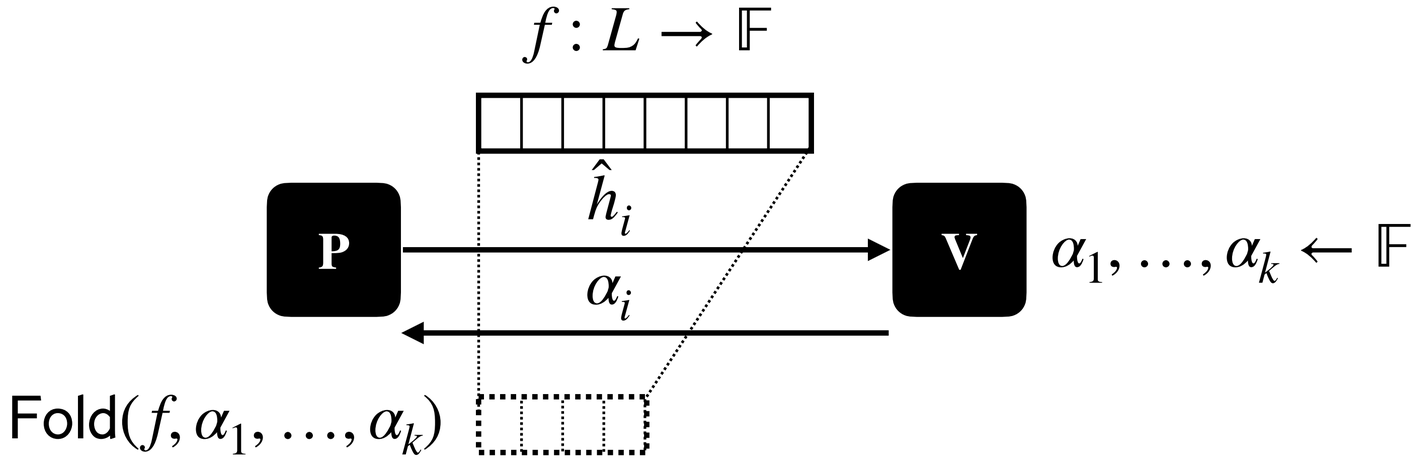
- By SZ lemma w.h.p.
  no pair $\hat{u}, \hat{v}$ with $\hat{u}(\mathbf{r}) = \hat{v}(\mathbf{r})$
- Prover "chooses" which codeword $\hat{u}$ it "commits" to



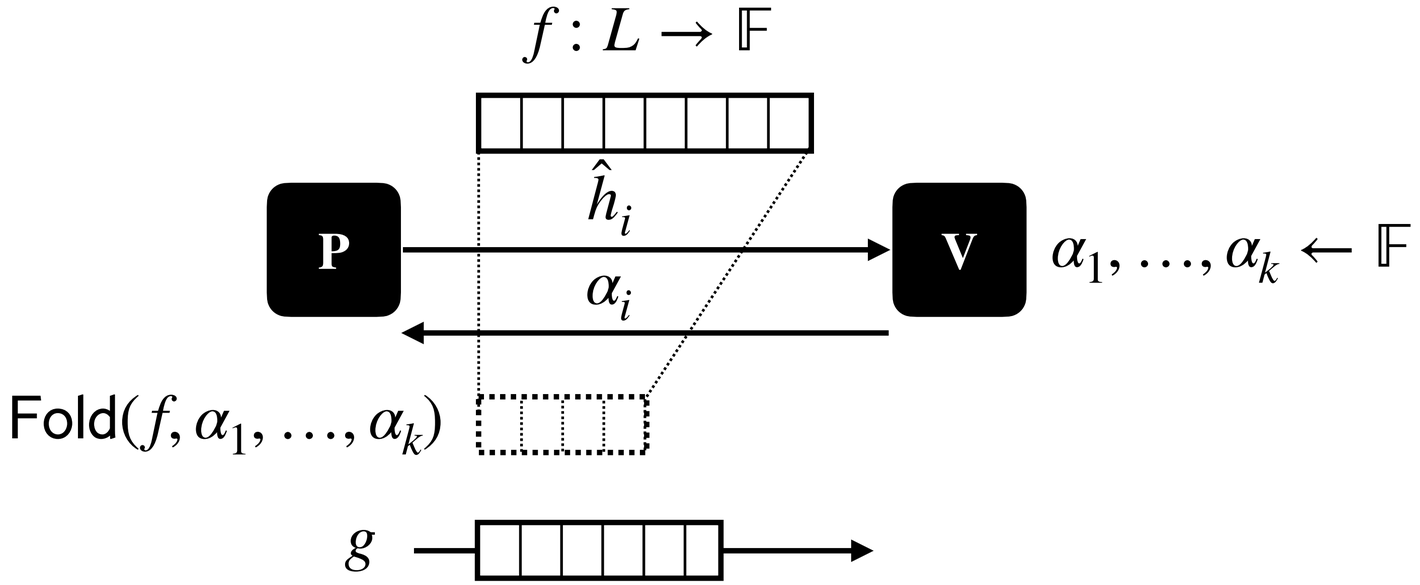
Add to list of constraints to **enforce**!

WHIR 

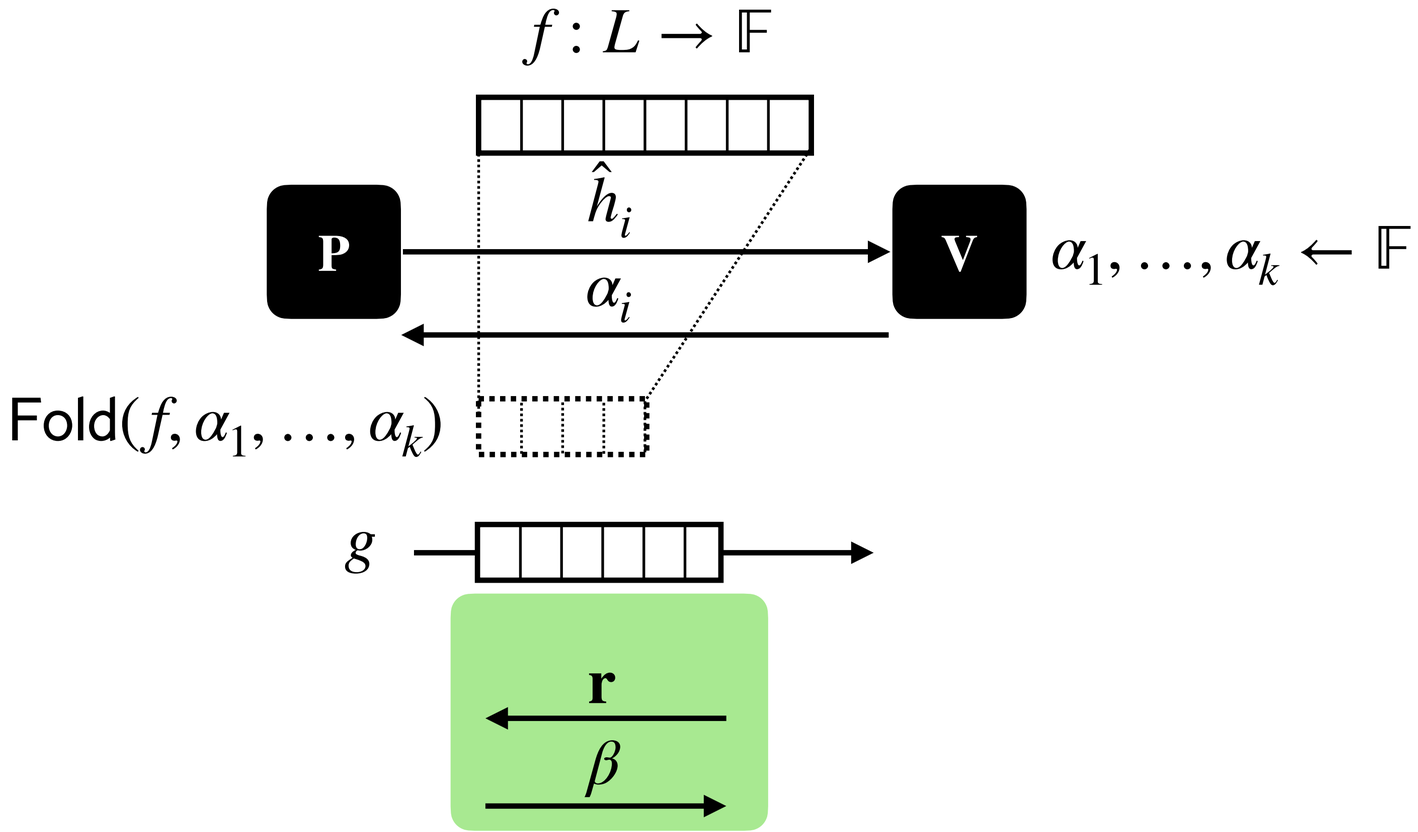
WHIR


$f : L \to \mathbb{F}$
$\hat{h}_i$
P
V
α_i
$\alpha_1, \ldots, \alpha_k \leftarrow \mathbb{F}$
Fold$(f, \alpha_1, \ldots, \alpha_k)$

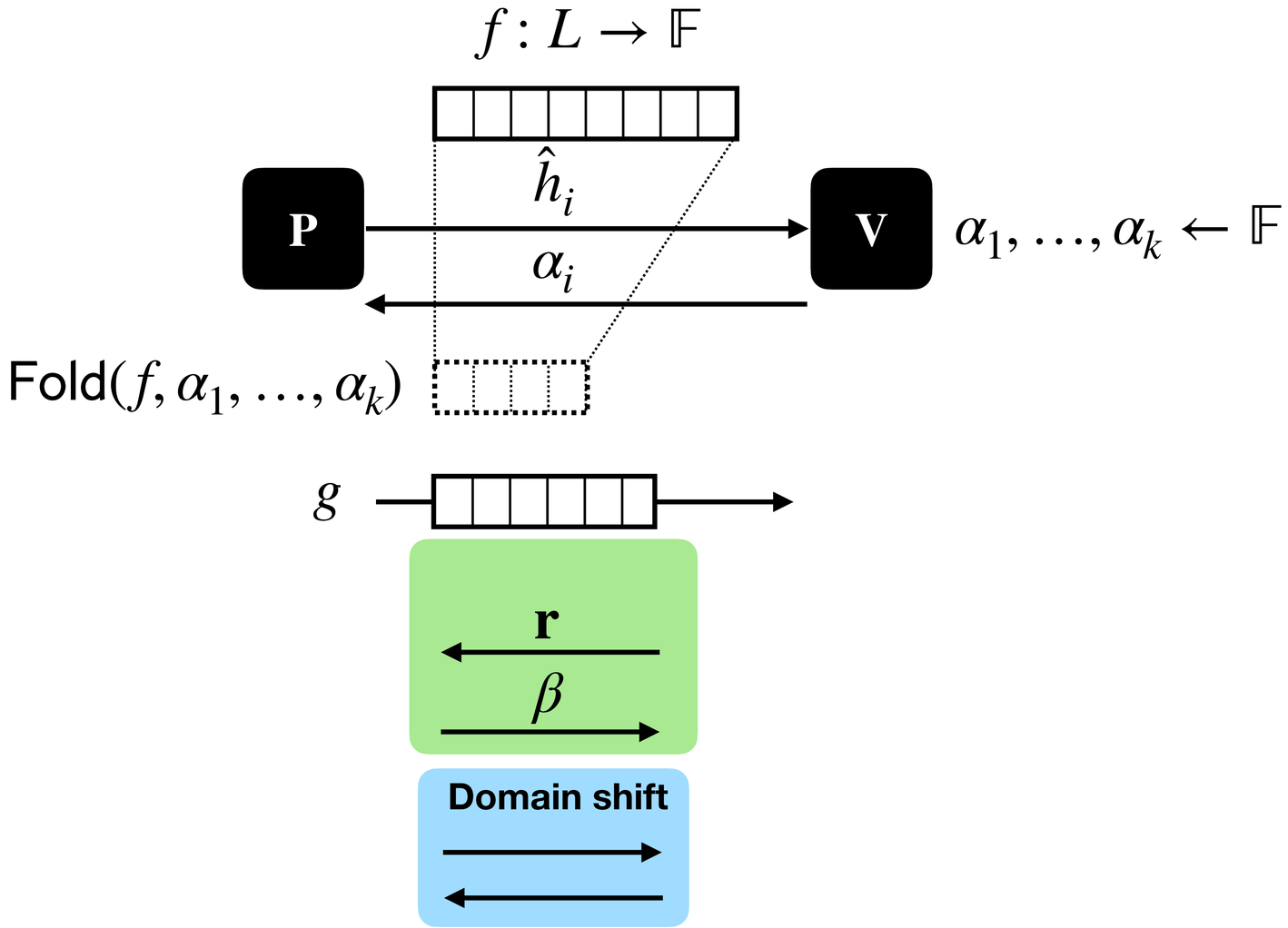
WHIR


$f : L \to \mathbb{F}$
$\hat{h}_i$
P
V
α_i
$\alpha_1, \ldots, \alpha_k \leftarrow \mathbb{F}$
Fold$(f, \alpha_1, \ldots, \alpha_k)$
g

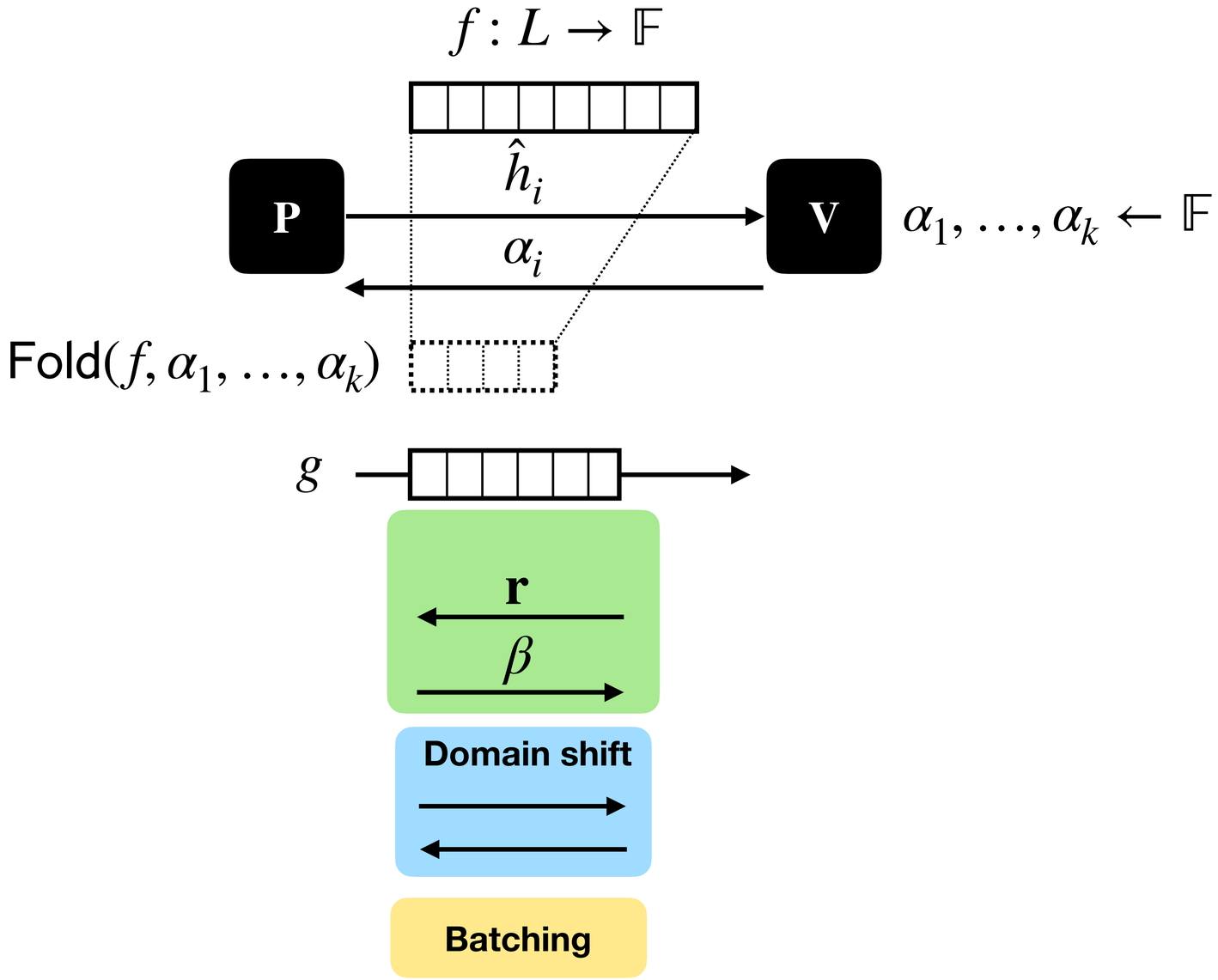
WHIR


$f : L \to \mathbb{F}$
$\hat{h}_i$
P
V
$\alpha_1, \ldots, \alpha_k \leftarrow \mathbb{F}$
α_i
Fold$(f, \alpha_1, \ldots, \alpha_k)$
g
r
β

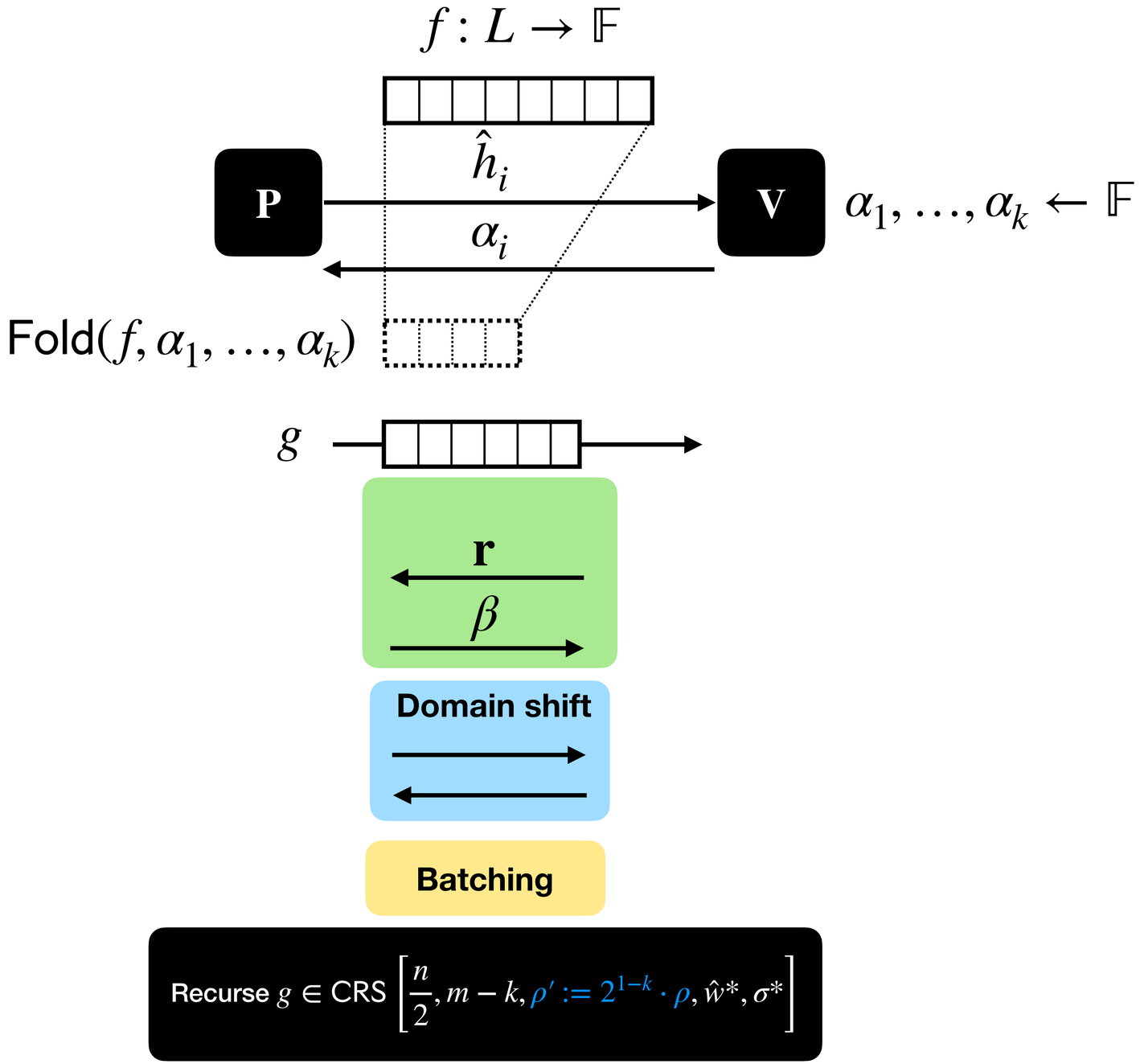
WHIR


$f : L \to \mathbb{F}$
$\hat{h}_i$
P
V
α_i
$\alpha_1, \ldots, \alpha_k \leftarrow \mathbb{F}$
$\text{Fold}(f, \alpha_1, \ldots, \alpha_k)$
g
r
β
Domain shift

WHIR


$f : L \to \mathbb{F}$
$\hat{h}_i$
P
V
α_i
$\alpha_1, \ldots, \alpha_k \leftarrow \mathbb{F}$
$\text{Fold}(f, \alpha_1, \ldots, \alpha_k)$
g
r
β
Domain shift
Batching

WHIR

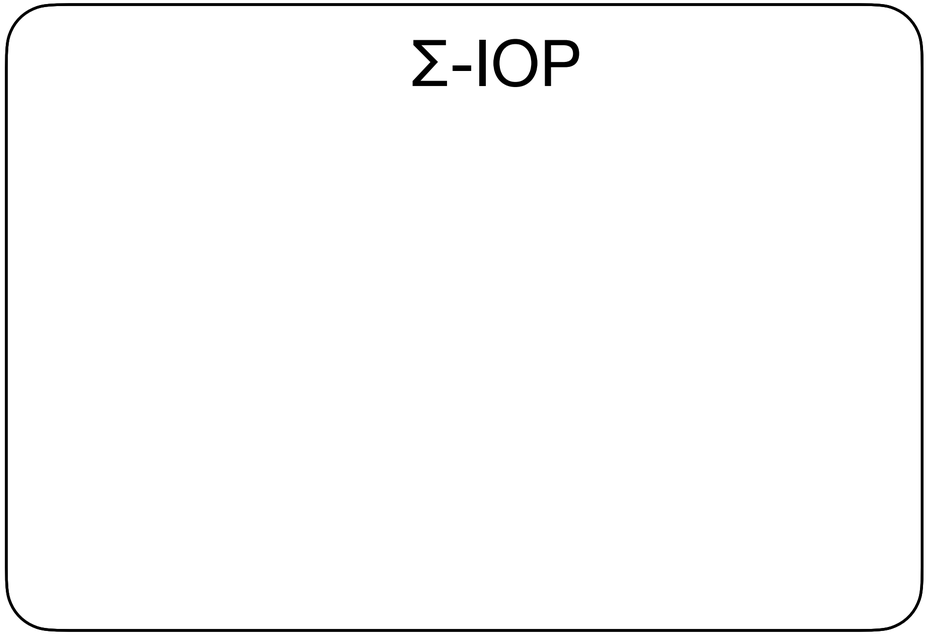

$f : L \to \mathbb{F}$
$\hat{h}_i$
P
V
α_i
$\alpha_1, \ldots, \alpha_k \leftarrow \mathbb{F}$
$\text{Fold}(f, \alpha_1, \ldots, \alpha_k)$
g
r
β
Domain shift
Batching
Recurse $g \in \text{CRS}$ $\left[\frac{n}{2}, m-k, \rho' := 2^{1-k} \cdot \rho, \hat{w}*, \sigma*\right]$

Application: Σ-IOP

High soundness compilation using constrained codes

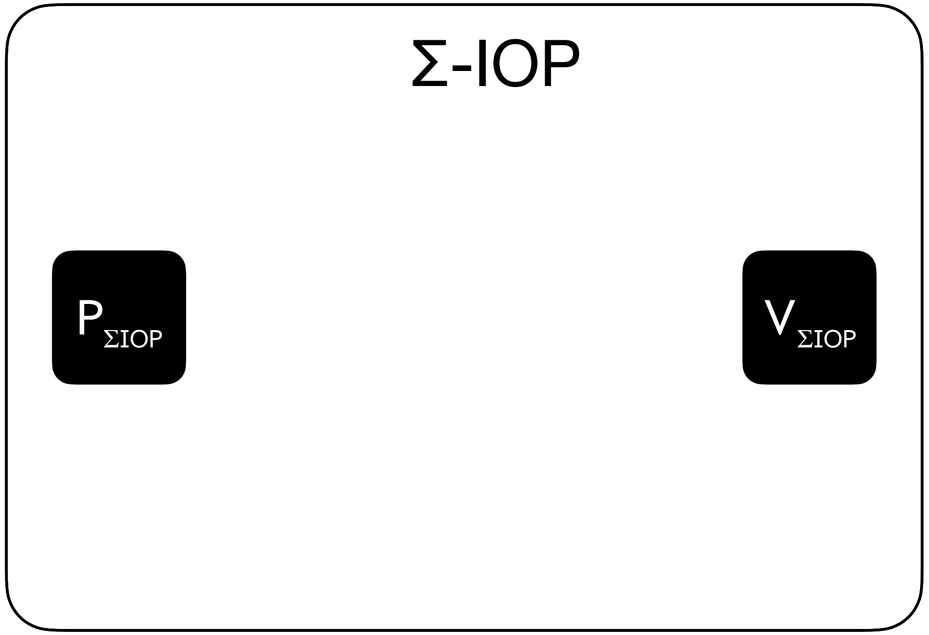
Application: Σ-IOP

High soundness compilation using constrained codes



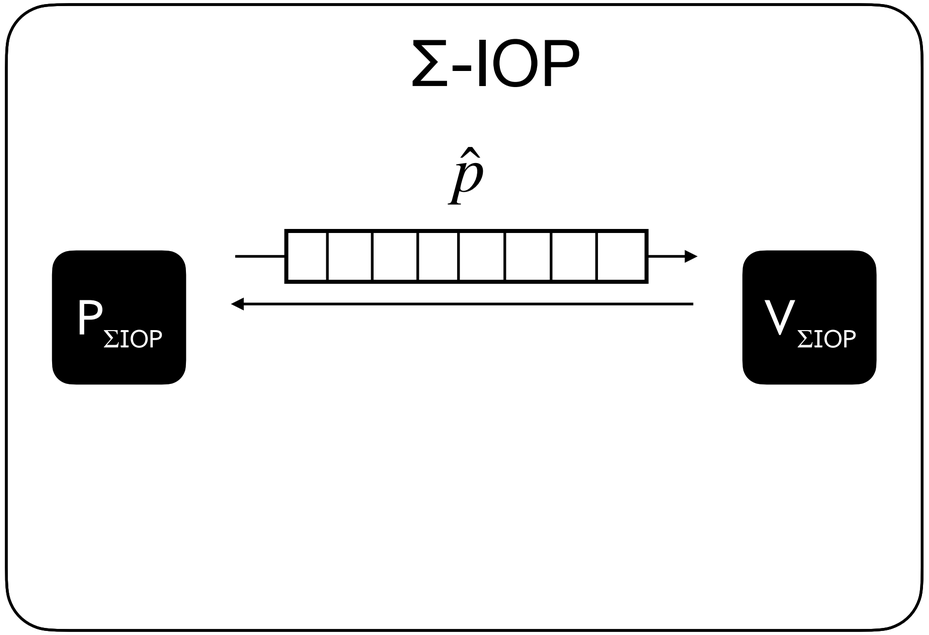
Application: Σ-IOP

High soundness compilation using constrained codes



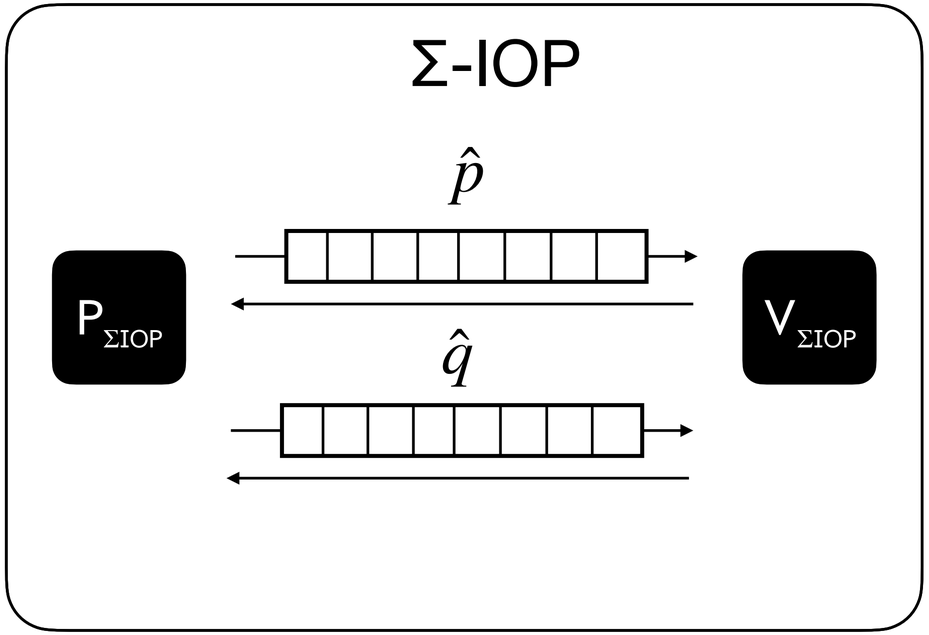
Application: Σ-IOP

High soundness compilation using constrained codes



Application: Σ-IOP

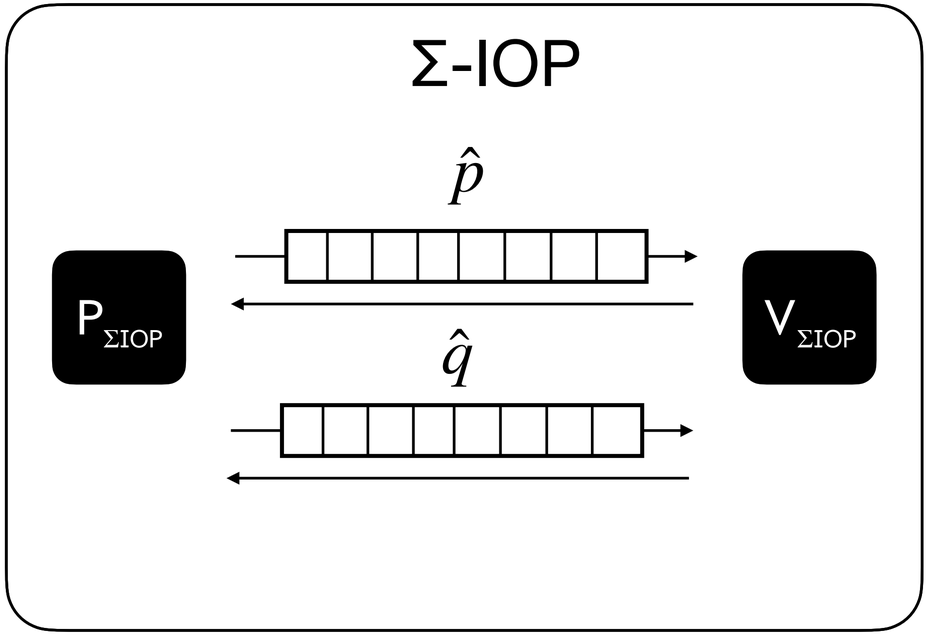
High soundness compilation using constrained codes



Application: Σ-IOP

High soundness compilation using constrained codes

Generalizes univariate and multilinear PIOPs at no extra cost!



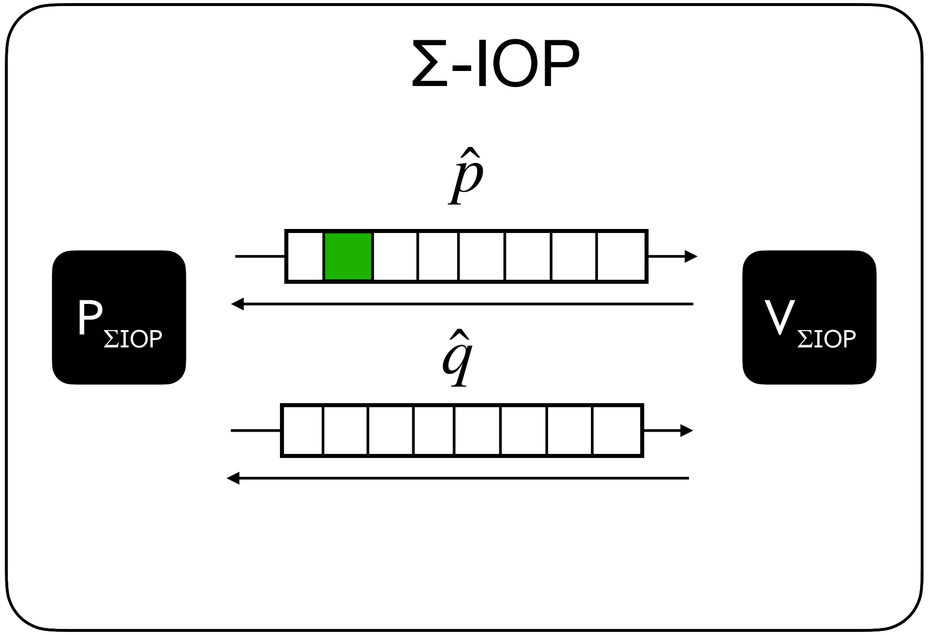
Verifier can ask **sumcheck queries**

i.e. send $\hat{w}$ and receive $\sum_{\mathbf{b}} \hat{w}(\hat{f}(\mathbf{b}), \mathbf{b})$

Generalizes univariate and multilinear PIOPs at no extra cost!

Application: Σ-IOP

High soundness compilation using constrained codes



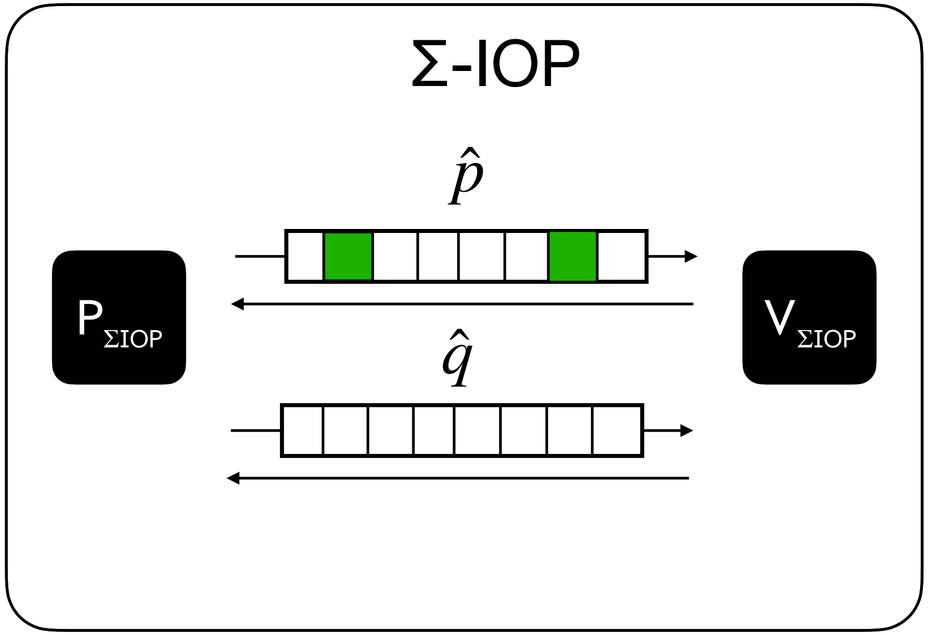
Verifier can ask **sumcheck queries**

i.e. send $\hat{w}$ and receive $\sum_{\mathbf{b}} \hat{w}(\hat{f}(\mathbf{b}), \mathbf{b})$

Generalizes univariate and multilinear PIOPs at no extra cost!

Application: Σ-IOP

High soundness compilation using constrained codes



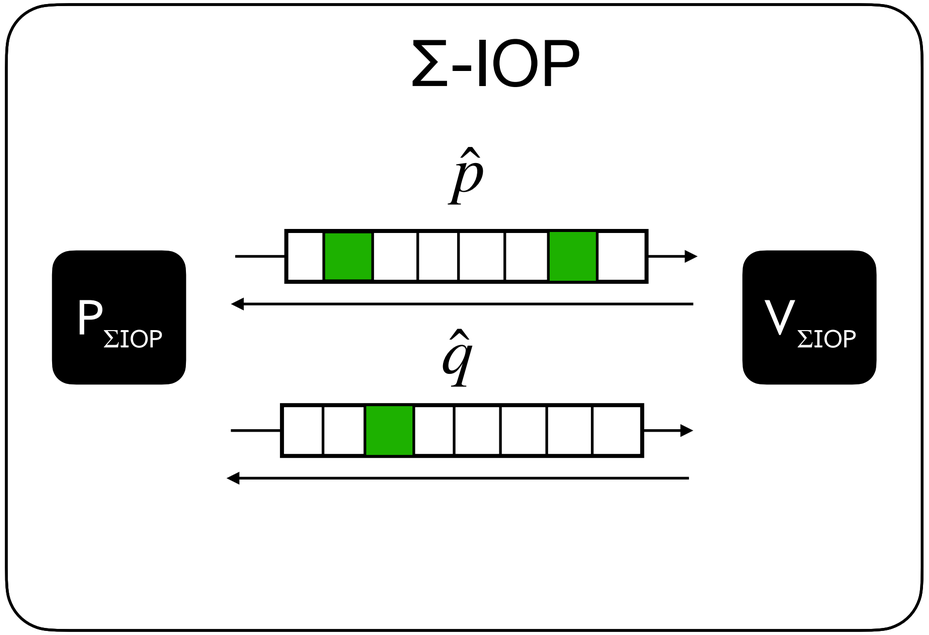
Verifier can ask **sumcheck queries**

i.e. send $\hat{w}$ and receive $\sum_{\mathbf{b}} \hat{w}(\hat{f}(\mathbf{b}), \mathbf{b})$

Generalizes univariate and multilinear PIOPs at no extra cost!

Application: Σ-IOP

High soundness compilation using constrained codes



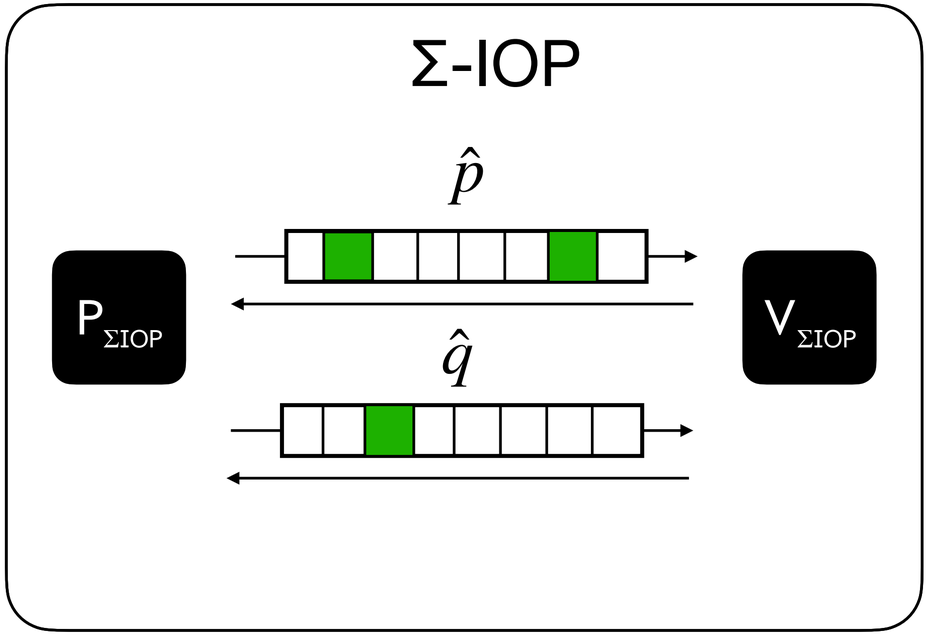
Verifier can ask **sumcheck queries**

i.e. send $\hat{w}$ and receive $\sum_{\mathbf{b}} \hat{w}(\hat{f}(\mathbf{b}), \mathbf{b})$

Application: Σ-IOP

High soundness compilation using constrained codes

Generalizes univariate and multilinear PIOPs at no extra cost!



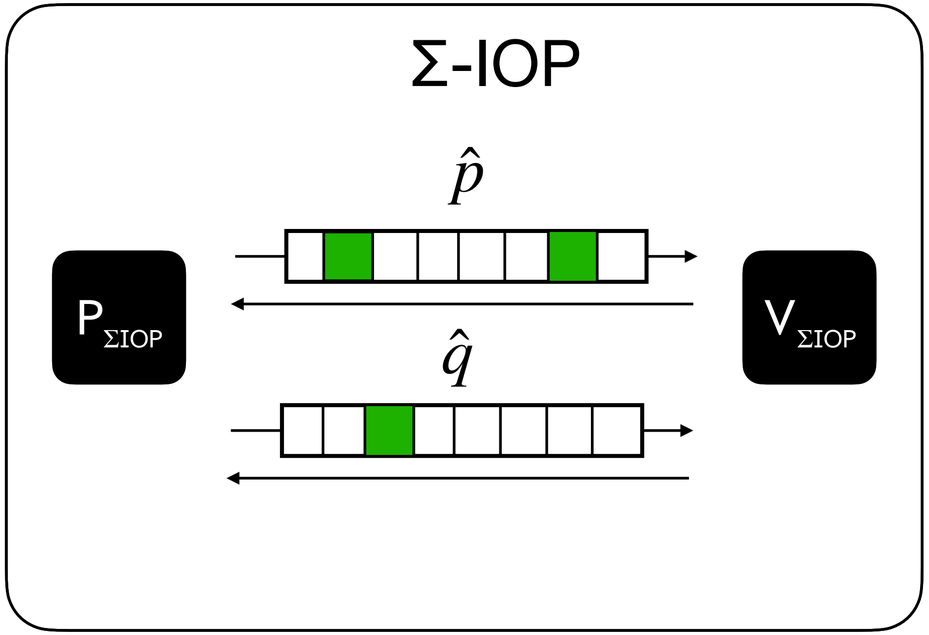
Verifier can ask **sumcheck queries**

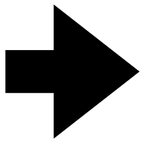
i.e. send $\hat{w}$ and receive $\sum_{\mathbf{b}} \hat{w}(\hat{f}(\mathbf{b}), \mathbf{b})$

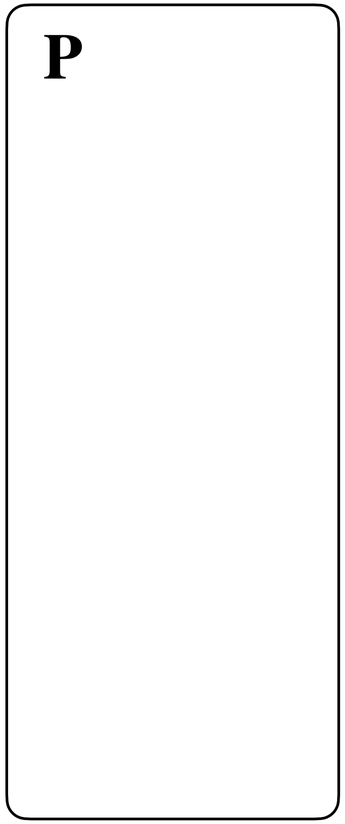
Application: Σ-IOP

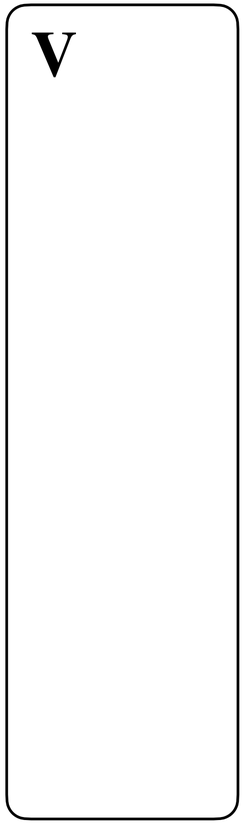
High soundness compilation using constrained codes

Generalizes univariate and multilinear PIOPs at no extra cost!









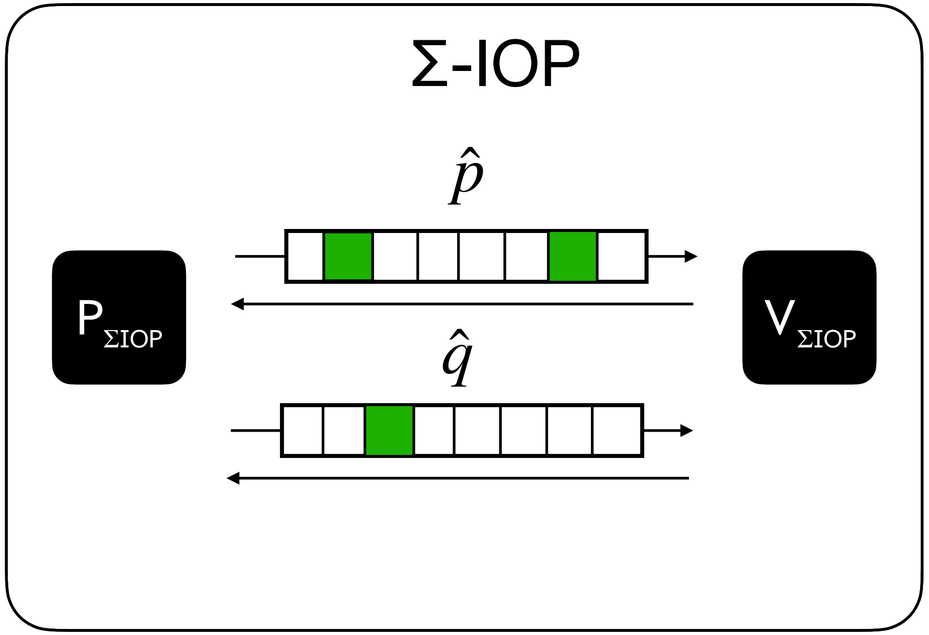
Verifier can ask **sumcheck queries**

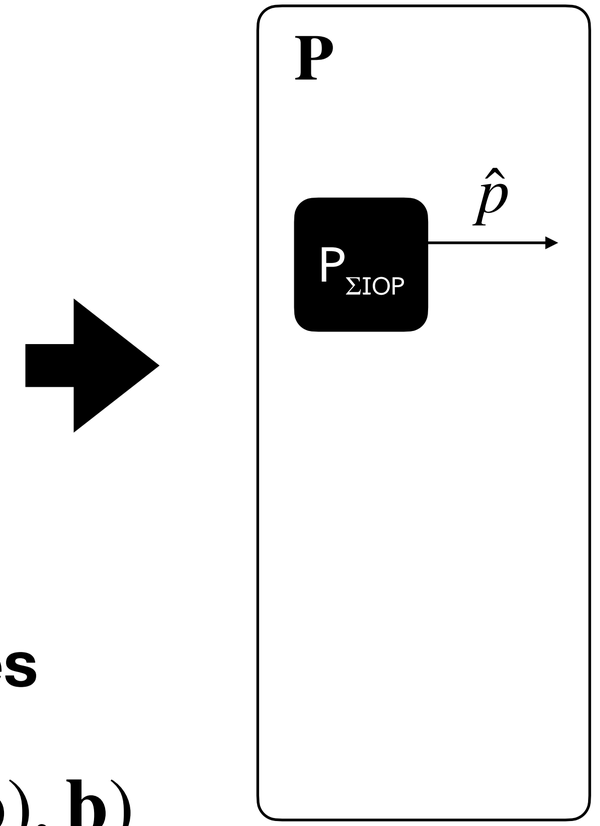
i.e. send $\hat{w}$ and receive $\sum_{\mathbf{b}} \hat{w}(\hat{f}(\mathbf{b}), \mathbf{b})$

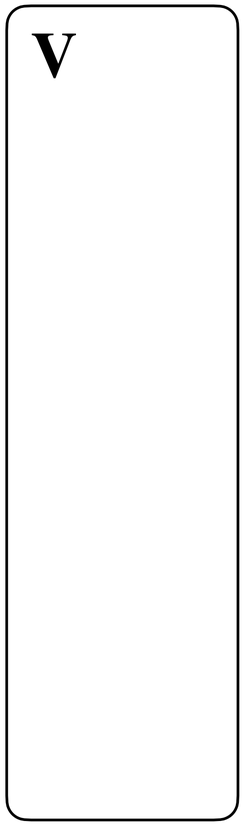
Application: Σ-IOP

High soundness compilation using constrained codes

Generalizes univariate and multilinear PIOPs at no extra cost!







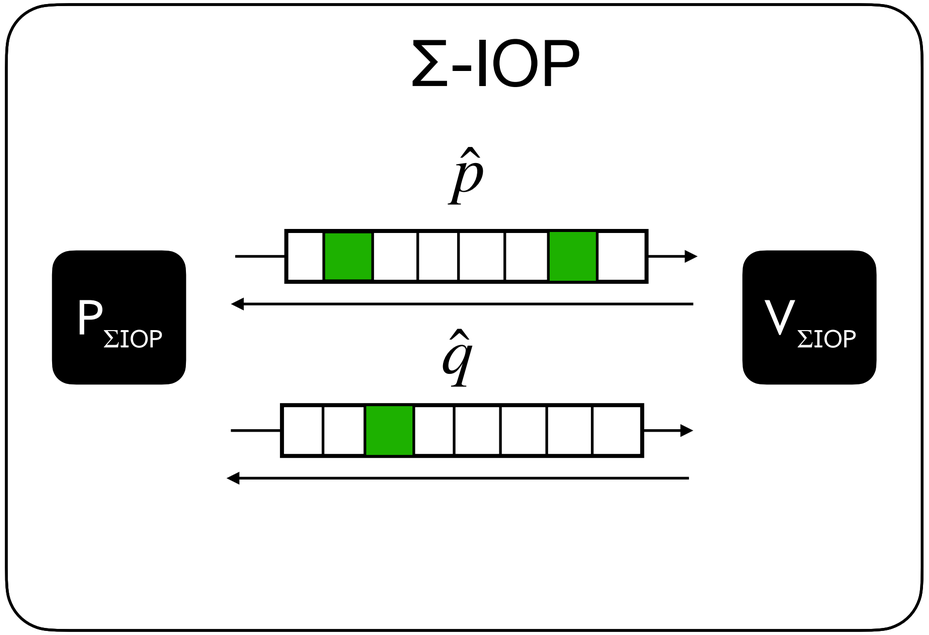
Verifier can ask **sumcheck queries**

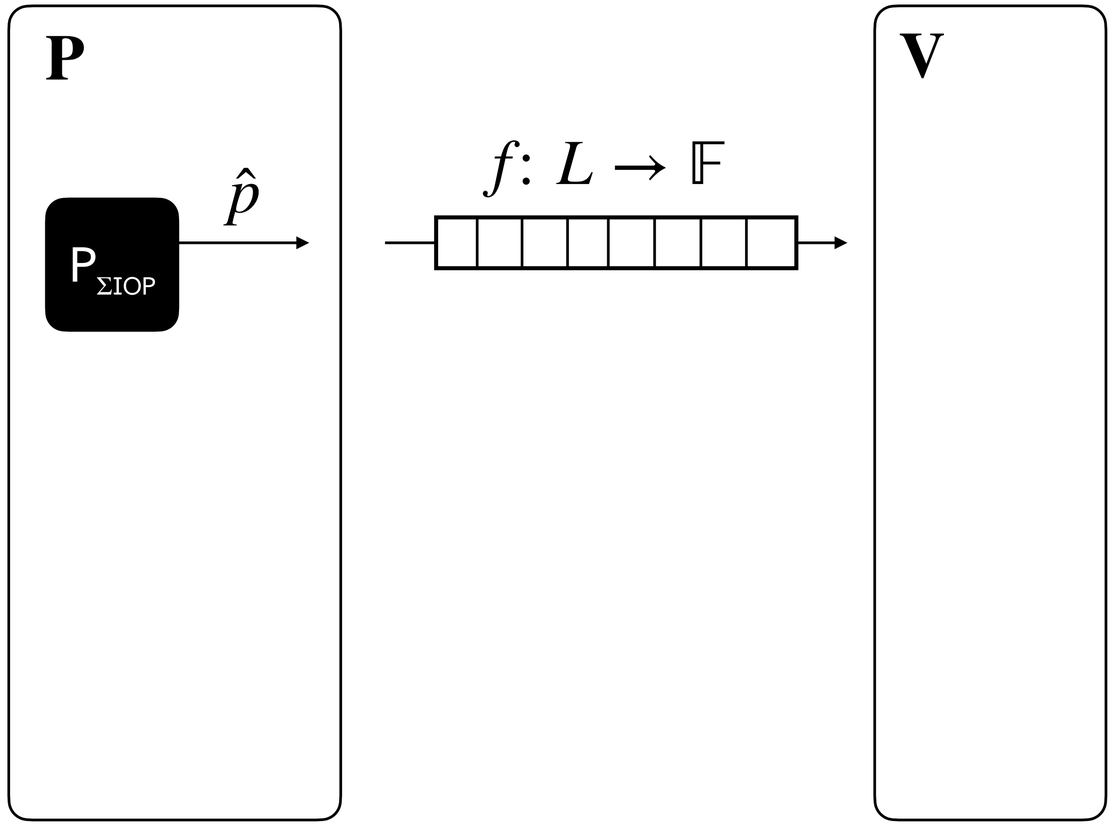
i.e. send $\hat{w}$ and receive $\sum_{\mathbf{b}} \hat{w}(\hat{f}(\mathbf{b}), \mathbf{b})$

Application: Σ-IOP

High soundness compilation using constrained codes

Generalizes univariate and multilinear PIOPs at no extra cost!





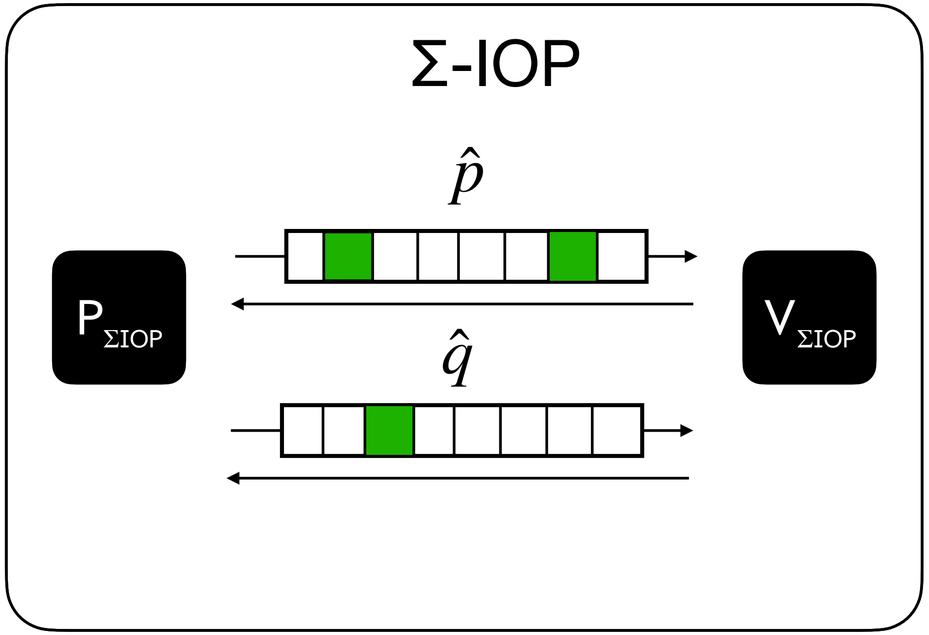
Verifier can ask **sumcheck queries**

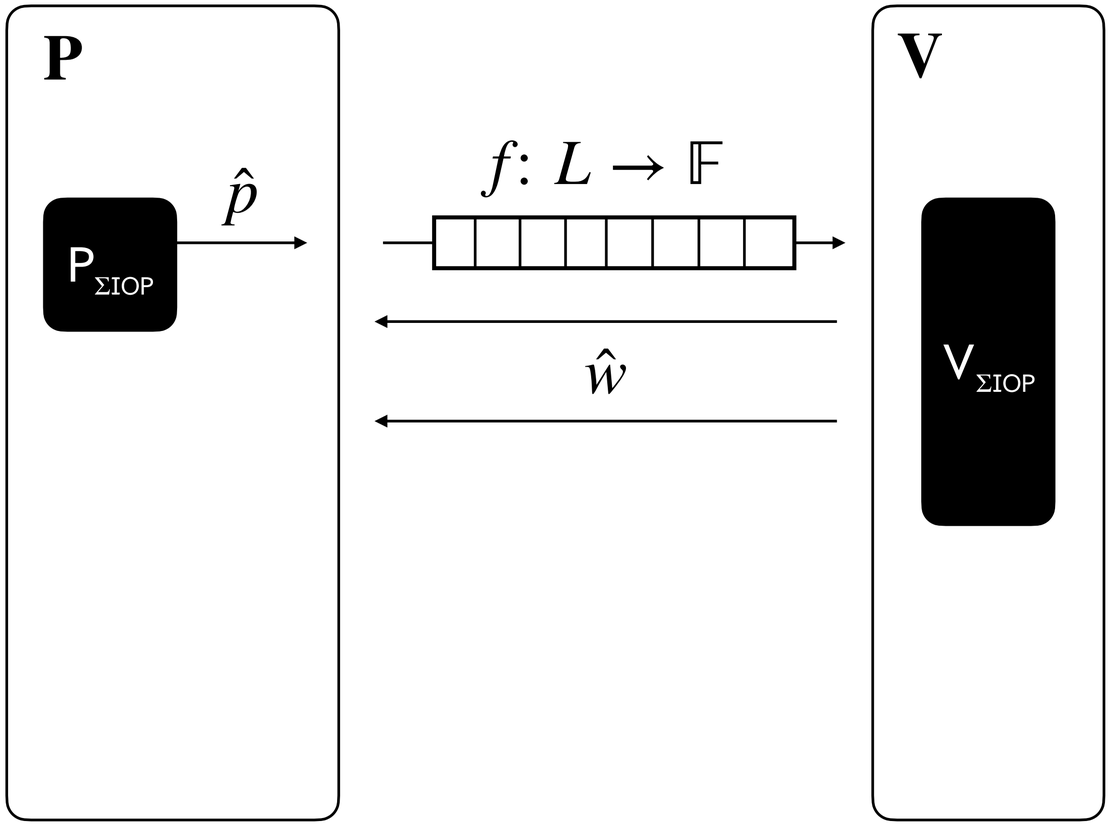
i.e. send $\hat{w}$ and receive $\sum_{\mathbf{b}} \hat{w}(\hat{f}(\mathbf{b}), \mathbf{b})$

Application: Σ-IOP

High soundness compilation using constrained codes

Generalizes univariate and multilinear PIOPs at no extra cost!





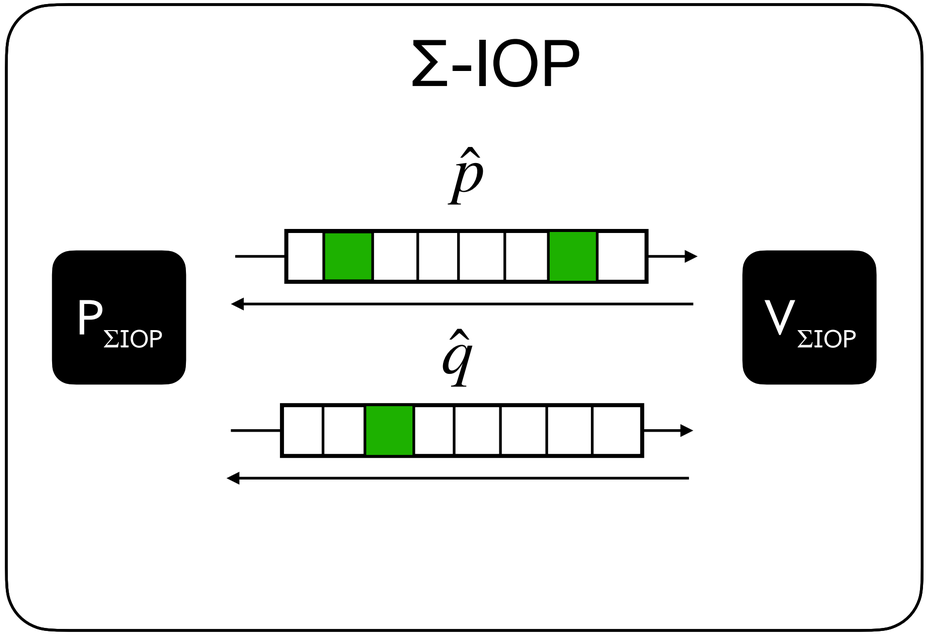
Verifier can ask **sumcheck queries**

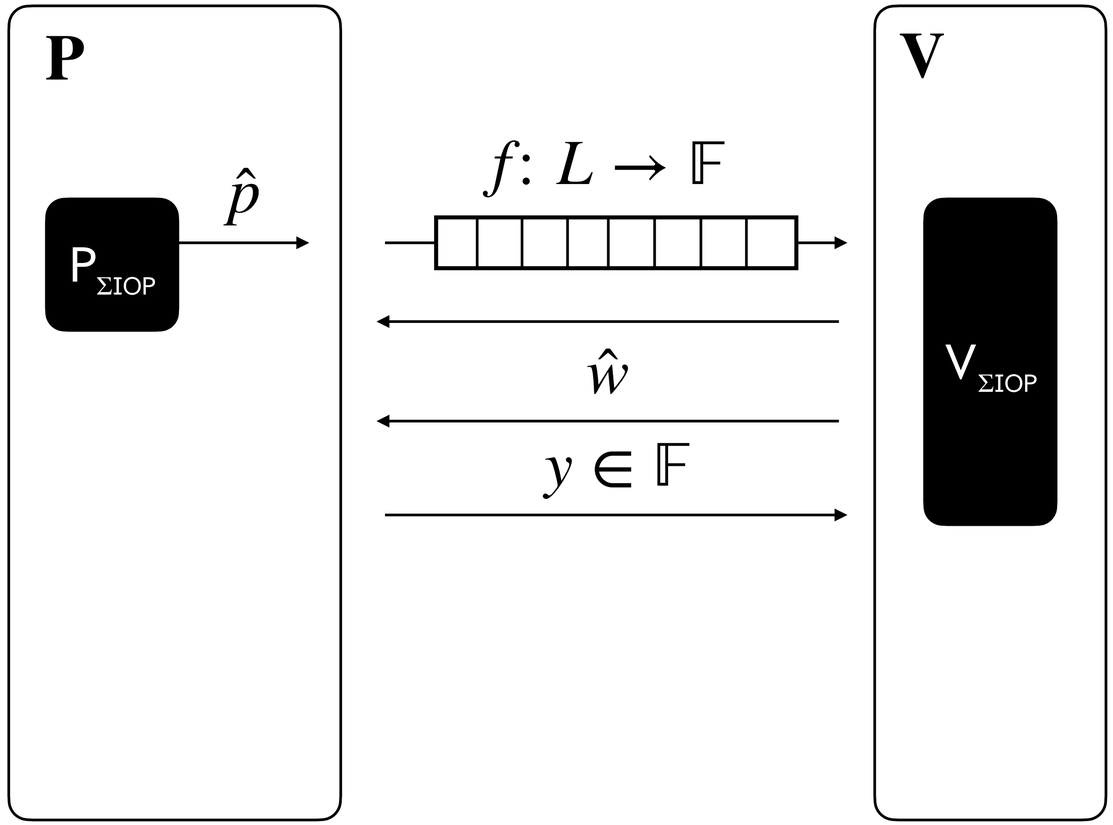
i.e. send $\hat{w}$ and receive $\sum_{\mathbf{b}} \hat{w}(\hat{f}(\mathbf{b}), \mathbf{b})$

Application: Σ-IOP

High soundness compilation using constrained codes

Generalizes univariate and multilinear PIOPs at no extra cost!





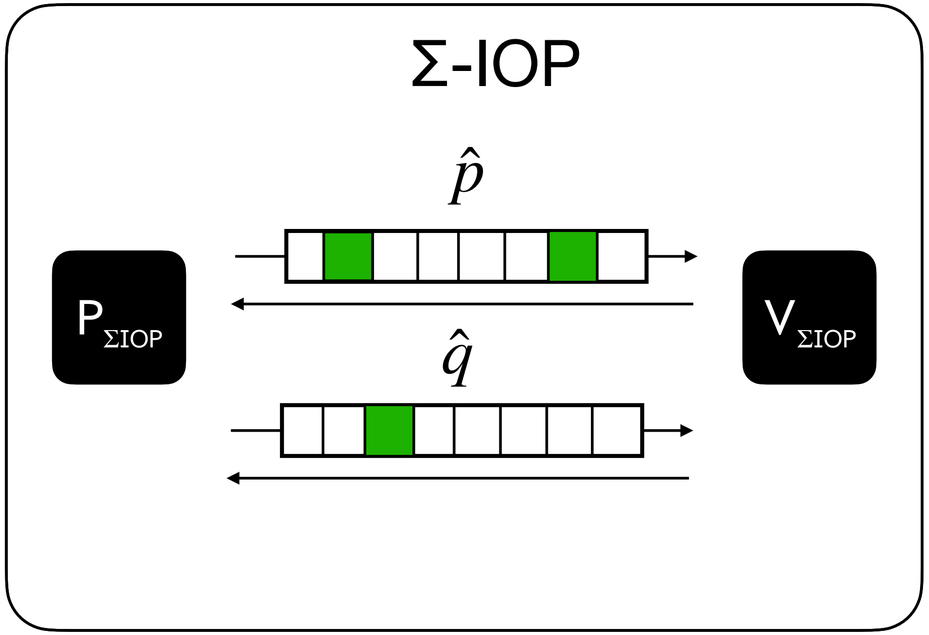
Verifier can ask **sumcheck queries**

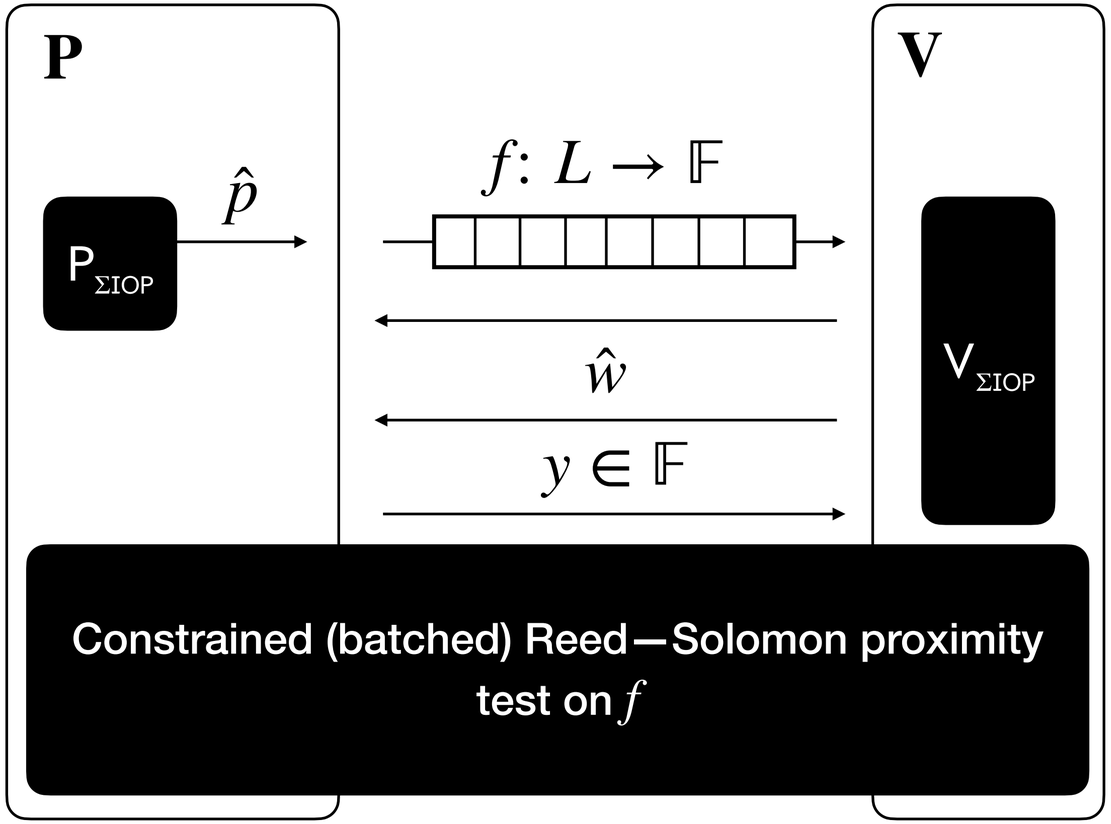
i.e. send $\hat{w}$ and receive $\sum_{\mathbf{b}} \hat{w}(\hat{f}(\mathbf{b}), \mathbf{b})$

Application: Σ-IOP

High soundness compilation using constrained codes

Generalizes univariate and multilinear PIOPs at no extra cost!





Verifier can ask **sumcheck queries**

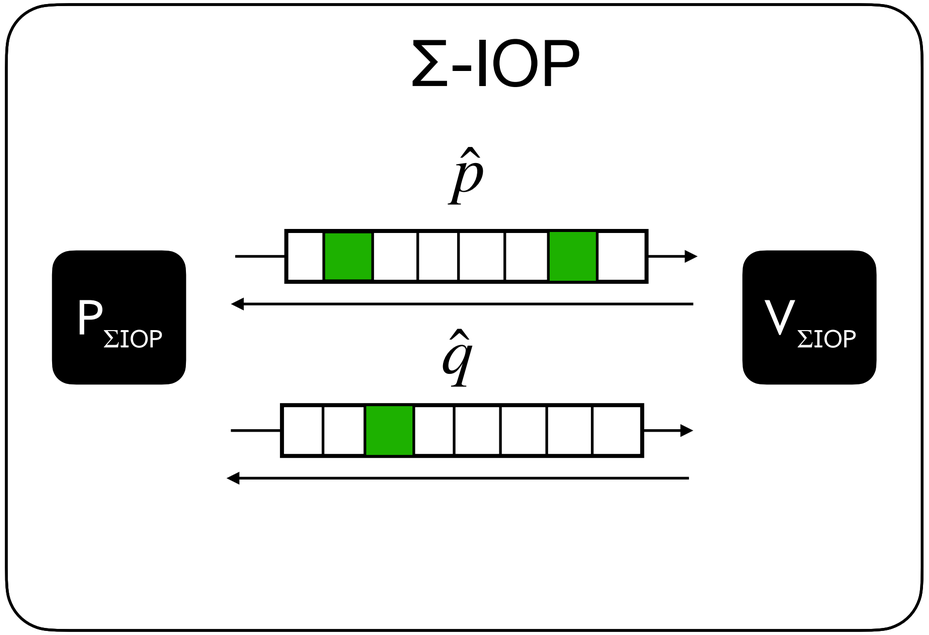
i.e. send $\hat{w}$ and receive $\sum_{\mathbf{b}} \hat{w}(\hat{f}(\mathbf{b}), \mathbf{b})$

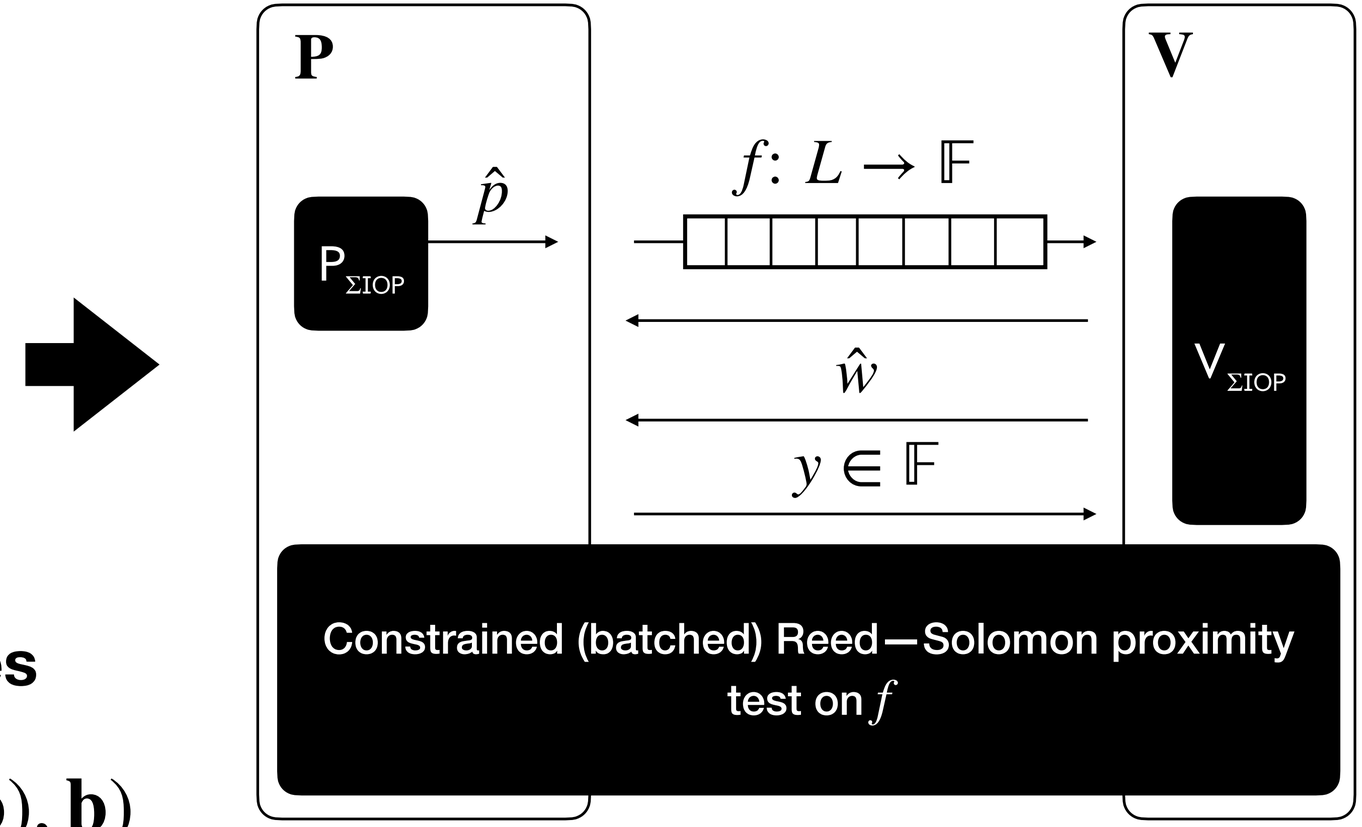
Application: Σ-IOP

High soundness compilation using constrained codes

Generalizes univariate and multilinear PIOPs at no extra cost!

Q: Can we use this to do more **efficient** arithmetizations?



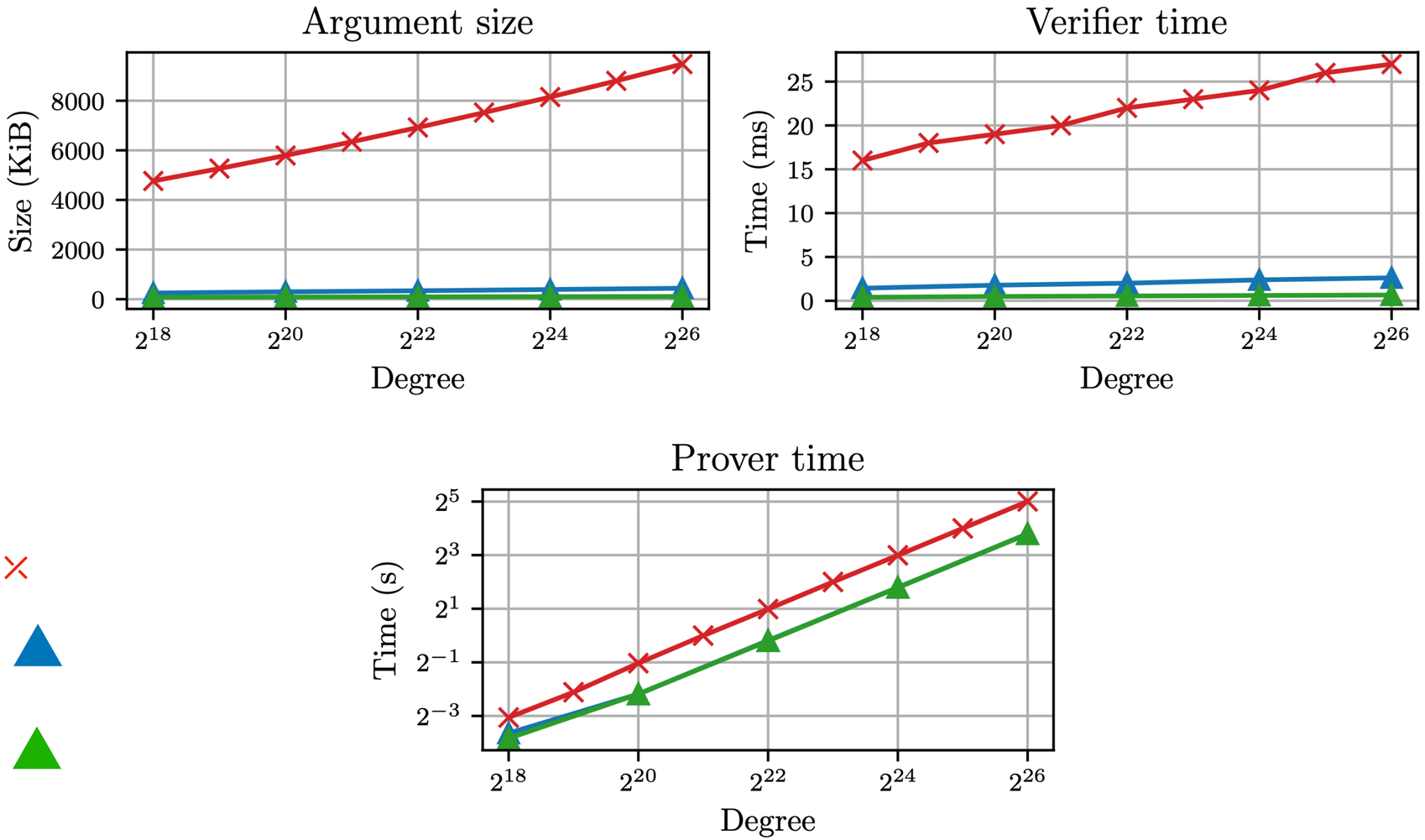


Verifier can ask **sumcheck queries**

i.e. send $\hat{w}$ and receive $\sum_{\mathbf{b}} \hat{w}(\hat{f}(\mathbf{b}), \mathbf{b})$

Extra slides

Comparison with BaseFold

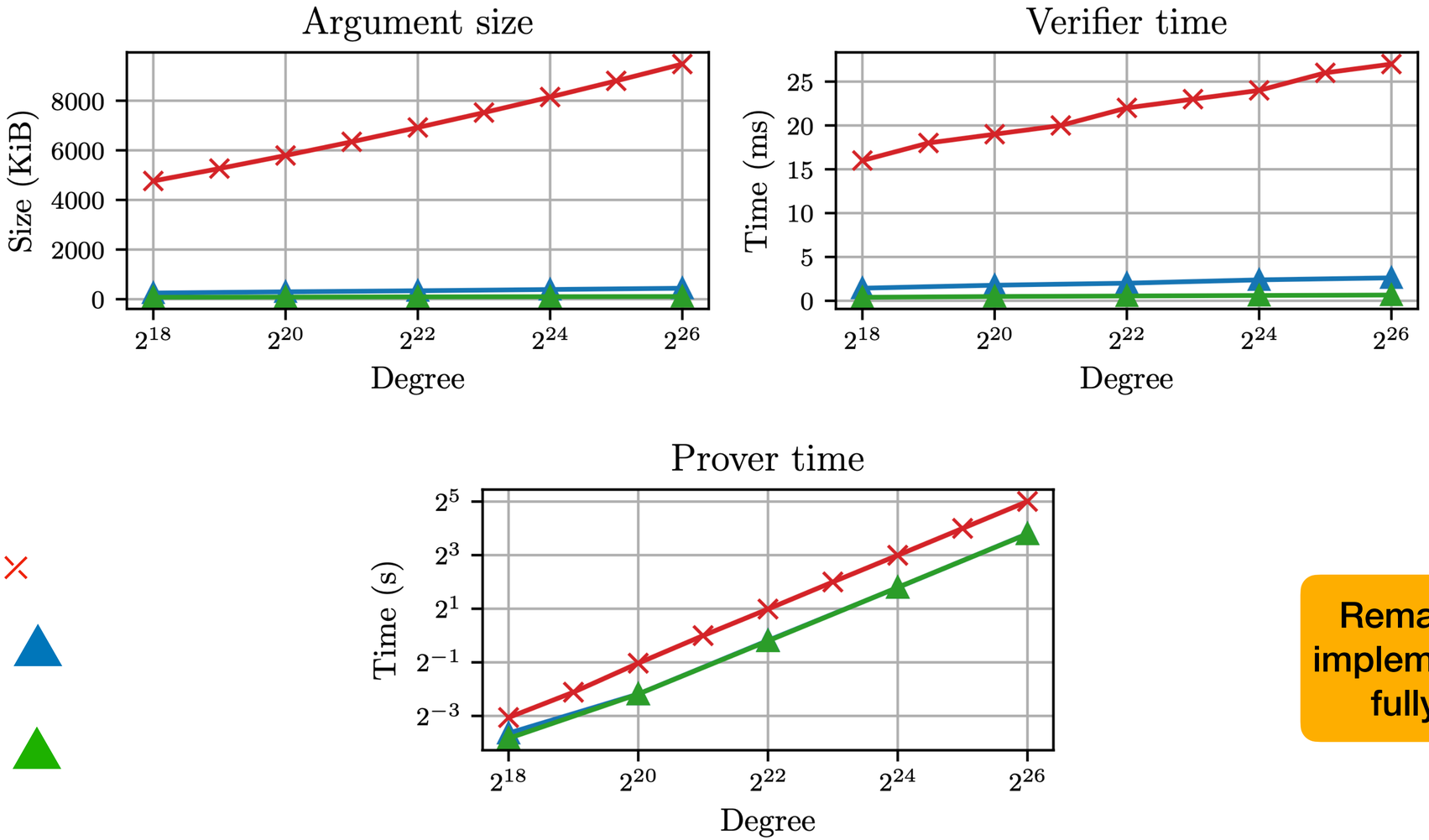


BaseFold: ×

WHIR-UD: ▲

WHIR-CB: ▲

Comparison with BaseFold



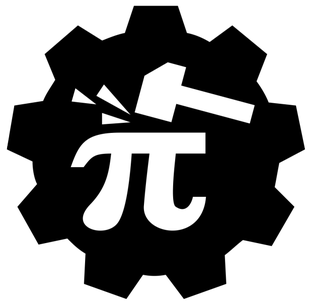
BaseFold: ×

WHIR-UD: ▲

WHIR-CB: ▲

Remark: BaseFold implementation is not fully optimised

Implementation

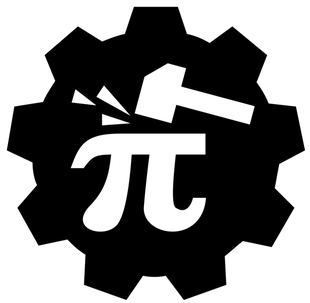


```
=========================================
Whir (PCS) 🌪
Field: Goldilocks2 and MT: Blake3
Number of variables: 20, folding factor: 4
Security level: 100 bits using ConjectureList security and 19 bits of PoW
initial_folding_pow_bits: 0
Num_queries: 41, rate: 2^-2, pow_bits: 18, ood_samples: 2, folding_pow: 0
Num_queries: 17, rate: 2^-5, pow_bits: 15, ood_samples: 2, folding_pow: 2
Num_queries: 11, rate: 2^-8, pow_bits: 12, ood_samples: 2, folding_pow: 4
Num_queries: 8, rate: 2^-11, pow_bits: 12, ood_samples: 2, folding_pow: 6
final_queries: 6, final_rate: 2^-14, final_pow_bits: 16, final_folding_pow_bits: 0
-----------------------------------
Round by round soundness analysis:
-----------------------------------
167.0 bits -- OOD commitment
102.0 bits -- (x4) prox gaps: 103.0, sumcheck: 102.0, pow: 0.0
171.0 bits -- OOD sample
100.0 bits -- query error: 82.0, combination: 94.6, pow: 18.0
100.0 bits -- (x4) prox gaps: 101.0, sumcheck: 100.0, pow: 0.0
175.0 bits -- OOD sample
100.0 bits -- query error: 85.0, combination: 93.8, pow: 15.0
100.0 bits -- (x4) prox gaps: 99.0, sumcheck: 98.0, pow: 2.0
179.0 bits -- OOD sample
100.0 bits -- query error: 88.0, combination: 92.3, pow: 12.0
100.0 bits -- (x4) prox gaps: 97.0, sumcheck: 96.0, pow: 4.0
183.0 bits -- OOD sample
100.0 bits -- query error: 88.0, combination: 90.7, pow: 12.0
100.0 bits -- (x4) prox gaps: 95.0, sumcheck: 94.0, pow: 6.0
100.0 bits -- query error: 84.0, pow: 16.0

Prover time: 356.9ms
Proof size: 58.7 KiB
Verifier time: 342.8µs
Average hashes: 1.1k
```

Implementation

- Rust 🦀 implementation, available at WizardOfMenlo/whir

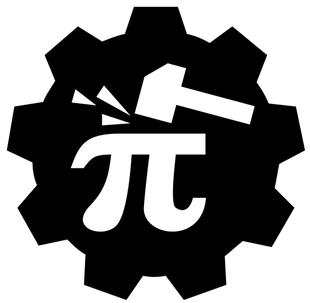


```
=========================================
Whir (PCS) 🌪
Field: Goldilocks2 and MT: Blake3
Number of variables: 20, folding factor: 4
Security level: 100 bits using ConjectureList security and 19 bits of PoW
initial_folding_pow_bits: 0
Num_queries: 41, rate: 2^-2, pow_bits: 18, ood_samples: 2, folding_pow: 0
Num_queries: 17, rate: 2^-5, pow_bits: 15, ood_samples: 2, folding_pow: 2
Num_queries: 11, rate: 2^-8, pow_bits: 12, ood_samples: 2, folding_pow: 4
Num_queries: 8, rate: 2^-11, pow_bits: 12, ood_samples: 2, folding_pow: 6
final_queries: 6, final_rate: 2^-14, final_pow_bits: 16, final_folding_pow_bits: 0
-----------------------------------
Round by round soundness analysis:
-----------------------------------
167.0 bits -- OOD commitment
102.0 bits -- (x4) prox gaps: 103.0, sumcheck: 102.0, pow: 0.0
171.0 bits -- OOD sample
100.0 bits -- query error: 82.0, combination: 94.6, pow: 18.0
100.0 bits -- (x4) prox gaps: 101.0, sumcheck: 100.0, pow: 0.0
175.0 bits -- OOD sample
100.0 bits -- query error: 85.0, combination: 93.8, pow: 15.0
100.0 bits -- (x4) prox gaps: 99.0, sumcheck: 98.0, pow: 2.0
179.0 bits -- OOD sample
100.0 bits -- query error: 88.0, combination: 92.3, pow: 12.0
100.0 bits -- (x4) prox gaps: 97.0, sumcheck: 96.0, pow: 4.0
183.0 bits -- OOD sample
100.0 bits -- query error: 88.0, combination: 90.7, pow: 12.0
100.0 bits -- (x4) prox gaps: 95.0, sumcheck: 94.0, pow: 6.0
100.0 bits -- query error: 84.0, pow: 16.0

Prover time: 356.9ms
Proof size: 58.7 KiB
Verifier time: 342.8µs
Average hashes: 1.1k
```

Implementation

- Rust 🦀 implementation, available at WizardOfMenlo/whir
- Arkworks as backend, (extension of) Goldilocks for benchmarks

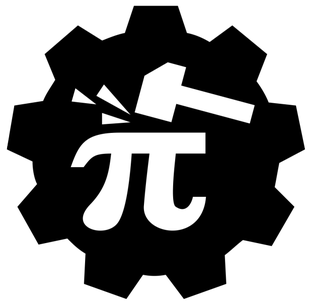


```
=========================================
Whir (PCS) 🌪
Field: Goldilocks2 and MT: Blake3
Number of variables: 20, folding factor: 4
Security level: 100 bits using ConjectureList security and 19 bits of PoW
initial_folding_pow_bits: 0
Num_queries: 41, rate: 2^-2, pow_bits: 18, ood_samples: 2, folding_pow: 0
Num_queries: 17, rate: 2^-5, pow_bits: 15, ood_samples: 2, folding_pow: 2
Num_queries: 11, rate: 2^-8, pow_bits: 12, ood_samples: 2, folding_pow: 4
Num_queries: 8, rate: 2^-11, pow_bits: 12, ood_samples: 2, folding_pow: 6
final_queries: 6, final_rate: 2^-14, final_pow_bits: 16, final_folding_pow_bits: 0
-----------------------------------
Round by round soundness analysis:
-----------------------------------
167.0 bits -- OOD commitment
102.0 bits -- (x4) prox gaps: 103.0, sumcheck: 102.0, pow: 0.0
171.0 bits -- OOD sample
100.0 bits -- query error: 82.0, combination: 94.6, pow: 18.0
100.0 bits -- (x4) prox gaps: 101.0, sumcheck: 100.0, pow: 0.0
175.0 bits -- OOD sample
100.0 bits -- query error: 85.0, combination: 93.8, pow: 15.0
100.0 bits -- (x4) prox gaps: 99.0, sumcheck: 98.0, pow: 2.0
179.0 bits -- OOD sample
100.0 bits -- query error: 88.0, combination: 92.3, pow: 12.0
100.0 bits -- (x4) prox gaps: 97.0, sumcheck: 96.0, pow: 4.0
183.0 bits -- OOD sample
100.0 bits -- query error: 88.0, combination: 90.7, pow: 12.0
100.0 bits -- (x4) prox gaps: 95.0, sumcheck: 94.0, pow: 6.0
100.0 bits -- query error: 84.0, pow: 16.0

Prover time: 356.9ms
Proof size: 58.7 KiB
Verifier time: 342.8µs
Average hashes: 1.1k
```

Implementation

- Rust 🦀 implementation, available at WizardOfMenlo/whir
- Arkworks as backend, (extension of) Goldilocks for benchmarks
  - Huge thanks to Remco Bloemen!!!

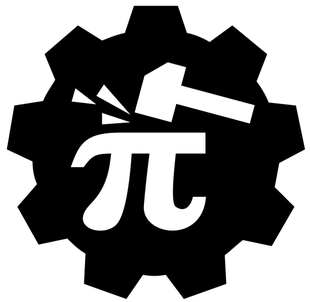


```
=========================================
Whir (PCS) 🌪
Field: Goldilocks2 and MT: Blake3
Number of variables: 20, folding factor: 4
Security level: 100 bits using ConjectureList security and 19 bits of PoW
initial_folding_pow_bits: 0
Num_queries: 41, rate: 2^-2, pow_bits: 18, ood_samples: 2, folding_pow: 0
Num_queries: 17, rate: 2^-5, pow_bits: 15, ood_samples: 2, folding_pow: 2
Num_queries: 11, rate: 2^-8, pow_bits: 12, ood_samples: 2, folding_pow: 4
Num_queries: 8, rate: 2^-11, pow_bits: 12, ood_samples: 2, folding_pow: 6
final_queries: 6, final_rate: 2^-14, final_pow_bits: 16, final_folding_pow_bits: 0
-----------------------------------
Round by round soundness analysis:
-----------------------------------
167.0 bits -- OOD commitment
102.0 bits -- (x4) prox gaps: 103.0, sumcheck: 102.0, pow: 0.0
171.0 bits -- OOD sample
100.0 bits -- query error: 82.0, combination: 94.6, pow: 18.0
100.0 bits -- (x4) prox gaps: 101.0, sumcheck: 100.0, pow: 0.0
175.0 bits -- OOD sample
100.0 bits -- query error: 85.0, combination: 93.8, pow: 15.0
100.0 bits -- (x4) prox gaps: 99.0, sumcheck: 98.0, pow: 2.0
179.0 bits -- OOD sample
100.0 bits -- query error: 88.0, combination: 92.3, pow: 12.0
100.0 bits -- (x4) prox gaps: 97.0, sumcheck: 96.0, pow: 4.0
183.0 bits -- OOD sample
100.0 bits -- query error: 88.0, combination: 90.7, pow: 12.0
100.0 bits -- (x4) prox gaps: 95.0, sumcheck: 94.0, pow: 6.0
100.0 bits -- query error: 84.0, pow: 16.0

Prover time: 356.9ms
Proof size: 58.7 KiB
Verifier time: 342.8µs
Average hashes: 1.1k
```

Implementation

- Rust 🦀 implementation, available at WizardOfMenlo/whir
- Arkworks as backend, (extension of) Goldilocks for benchmarks
  - Huge thanks to Remco Bloemen!!!
- We compared to FRI, STIR and BaseFold.



```
=========================================
Whir (PCS) 🌪
Field: Goldilocks2 and MT: Blake3
Number of variables: 20, folding factor: 4
Security level: 100 bits using ConjectureList security and 19 bits of PoW
initial_folding_pow_bits: 0
Num_queries: 41, rate: 2^-2, pow_bits: 18, ood_samples: 2, folding_pow: 0
Num_queries: 17, rate: 2^-5, pow_bits: 15, ood_samples: 2, folding_pow: 2
Num_queries: 11, rate: 2^-8, pow_bits: 12, ood_samples: 2, folding_pow: 4
Num_queries: 8, rate: 2^-11, pow_bits: 12, ood_samples: 2, folding_pow: 6
final_queries: 6, final_rate: 2^-14, final_pow_bits: 16, final_folding_pow_bits: 0
-----------------------------------
Round by round soundness analysis:
-----------------------------------
167.0 bits -- OOD commitment
102.0 bits -- (x4) prox gaps: 103.0, sumcheck: 102.0, pow: 0.0
171.0 bits -- OOD sample
100.0 bits -- query error: 82.0, combination: 94.6, pow: 18.0
100.0 bits -- (x4) prox gaps: 101.0, sumcheck: 100.0, pow: 0.0
175.0 bits -- OOD sample
100.0 bits -- query error: 85.0, combination: 93.8, pow: 15.0
100.0 bits -- (x4) prox gaps: 99.0, sumcheck: 98.0, pow: 2.0
179.0 bits -- OOD sample
100.0 bits -- query error: 88.0, combination: 92.3, pow: 12.0
100.0 bits -- (x4) prox gaps: 97.0, sumcheck: 96.0, pow: 4.0
183.0 bits -- OOD sample
100.0 bits -- query error: 88.0, combination: 90.7, pow: 12.0
100.0 bits -- (x4) prox gaps: 95.0, sumcheck: 94.0, pow: 6.0
100.0 bits -- query error: 84.0, pow: 16.0

Prover time: 356.9ms
Proof size: 58.7 KiB
Verifier time: 342.8µs
Average hashes: 1.1k
```

Comparison to STIR and FRI

Comparison to STIR and FRI

FRI: $O\left(\frac{\lambda}{k} \cdot m\right)$

STIR & WHIR $O\left(\frac{\lambda}{k} \cdot \log m\right)$

Comparison to STIR and FRI

FRI: $O\left(\frac{\lambda}{k} \cdot m\right)$

STIR & **WHIR** $O\left(\frac{\lambda}{k} \cdot \log m\right)$

- **Drop-in** replacement of FRI and STIR (when used for CRS$[\mathbb{F}, m, \rho, 0, 0]$)

Comparison to STIR and FRI

FRI: $O\left(\frac{\lambda}{k} \cdot m\right)$

STIR & **WHIR** $O\left(\frac{\lambda}{k} \cdot \log m\right)$

- **Drop-in** replacement of FRI and STIR (when used for CRS$[\mathbb{F}, m, \rho, 0, 0]$)
- **Same** benefits as STIR over FRI, and **faster** prover time.

Comparison to STIR and FRI

FRI: $O\left(\frac{\lambda}{k} \cdot m\right)$

STIR & WHIR $O\left(\frac{\lambda}{k} \cdot \log m\right)$

- **Drop-in** replacement of FRI and STIR (when used for CRS$[\mathbb{F}, m, \rho, 0, 0]$)
- **Same** benefits as STIR over FRI, and **faster** prover time.
- **Additionally, richer proximity tests means that:**

Comparison to STIR and FRI

FRI: $O\left(\frac{\lambda}{k} \cdot m\right)$

STIR & WHIR $O\left(\frac{\lambda}{k} \cdot \log m\right)$

- **Drop-in** replacement of FRI and STIR (when used for CRS$[\mathbb{F}, m, \rho, 0, 0]$)
- **Same** benefits as STIR over FRI, and **faster** prover time.
- **Additionally, richer proximity tests means that:**
  - Can be used as a **multilinear** PCS (instead of BaseFold, FRI-Binius, etc)

Comparison to STIR and FRI

FRI: $O\left(\frac{\lambda}{k} \cdot m\right)$

STIR & WHIR $O\left(\frac{\lambda}{k} \cdot \log m\right)$

- **Drop-in** replacement of FRI and STIR (when used for $\mathsf{CRS}[\mathbb{F}, m, \rho, 0, 0]$)
- **Same** benefits as STIR over FRI, and **faster** prover time.
- **Additionally, richer proximity tests means that:**
  - Can be used as a **multilinear** PCS (instead of BaseFold, FRI-Binius, etc)
  - Additionally, **bivariate** PCS (and anything in between)

Comparison to STIR and FRI

FRI: $O\left(\frac{\lambda}{k} \cdot m\right)$

STIR & WHIR $O\left(\frac{\lambda}{k} \cdot \log m\right)$

- **Drop-in** replacement of FRI and STIR (when used for $\mathsf{CRS}[\mathbb{F}, m, \rho, 0, 0]$)
- **Same** benefits as STIR over FRI, and **faster** prover time.
- **Additionally, richer proximity tests means that:**
  - Can be used as a **multilinear** PCS (instead of BaseFold, FRI-Binius, etc)
  - Additionally, **bivariate** PCS (and anything in between)
  - Can be used in compiler for Σ-IOP

Comparison to STIR and FRI

FRI: $O\left(\frac{\lambda}{k} \cdot m\right)$

STIR & WHIR $O\left(\frac{\lambda}{k} \cdot \log m\right)$

- **Drop-in** replacement of FRI and STIR (when used for $\mathsf{CRS}[\mathbb{F}, m, \rho, 0, 0]$)
- **Same** benefits as STIR over FRI, and **faster** prover time.
- **Additionally, richer proximity tests means that:**
  - Can be used as a **multilinear** PCS (instead of BaseFold, FRI-Binius, etc)
  - Additionally, **bivariate** PCS (and anything in between)
  - Can be used in compiler for Σ-IOP
- **Further,** super-fast verification (next)

Batching

Pick your favourite sumcheck batching

Batching

Pick your favourite sumcheck batching

$g : L \to \mathbb{F}$

Sumcheck claims on g:
$(\hat{w}_1, \sigma_1), \ldots, (\hat{w}_\ell, \sigma_\ell)$

Batching

Pick your favourite sumcheck batching

$g : L \to \mathbb{F}$

Sumcheck claims on g:

$(\hat{w}_1, \sigma_1), \ldots, (\hat{w}_\ell, \sigma_\ell)$

Batching

Batching

Pick your favourite sumcheck batching

$g : L \to \mathbb{F}$

Sumcheck claims on g:
$(\hat{w}_1, \sigma_1), \ldots, (\hat{w}_\ell, \sigma_\ell)$

Batching

$g : L \to \mathbb{F}$

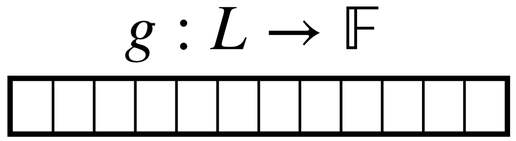
Sumcheck claim on g: $(\hat{w}^*, \sigma^*)$

Batching

Pick your favourite sumcheck batching

Sumcheck claims on g:
$(\hat{w}_1, \sigma_1), \ldots, (\hat{w}_\ell, \sigma_\ell)$

Batching

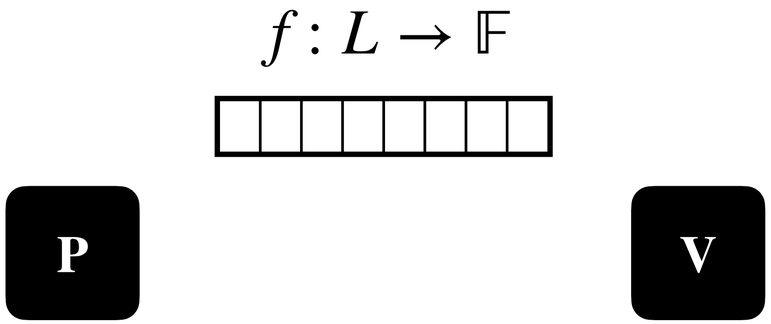


Sumcheck claim on g: $(\hat{w}^*, \sigma^*)$

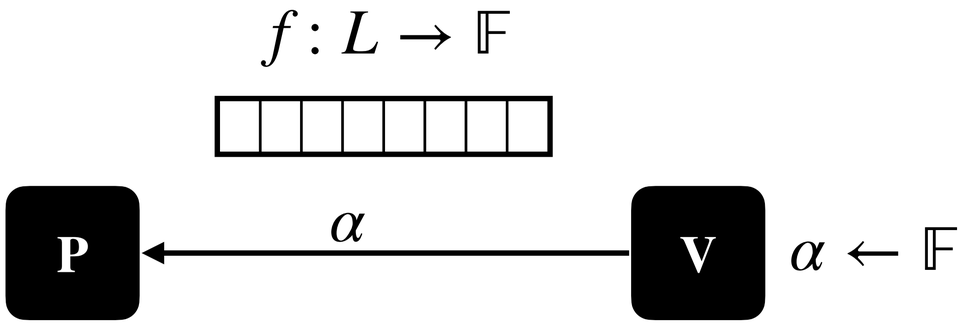
Many ways this can be done: **we chose random linear combination.**

Review: FRI iteration

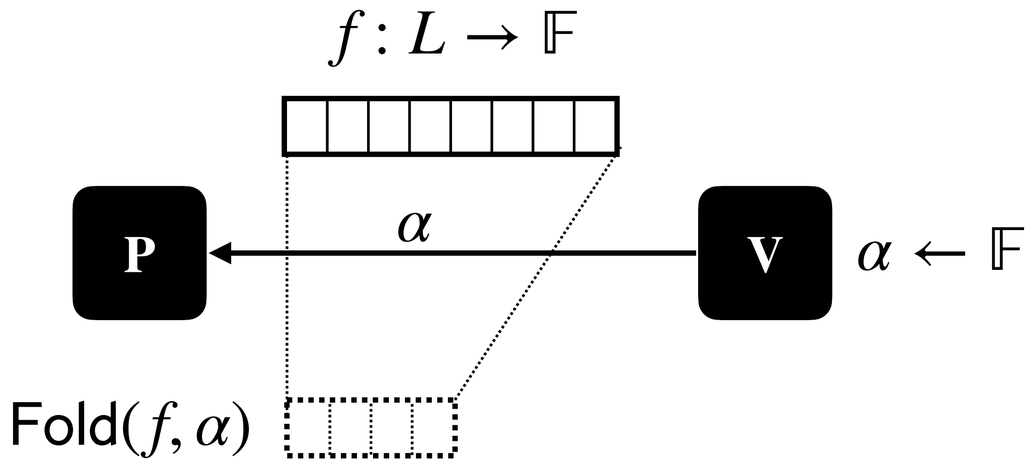
Review: FRI iteration



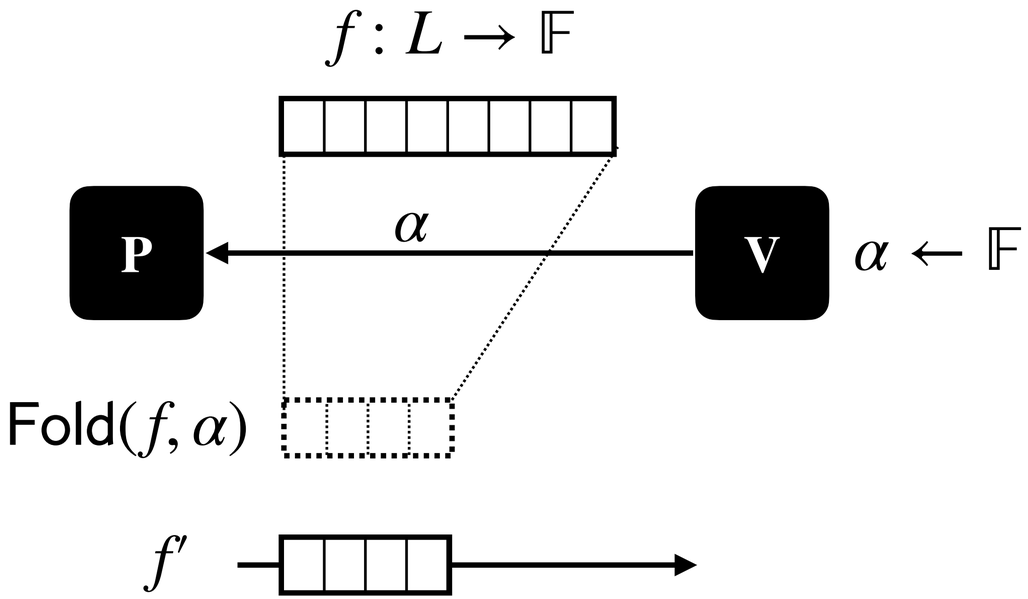
Review: FRI iteration



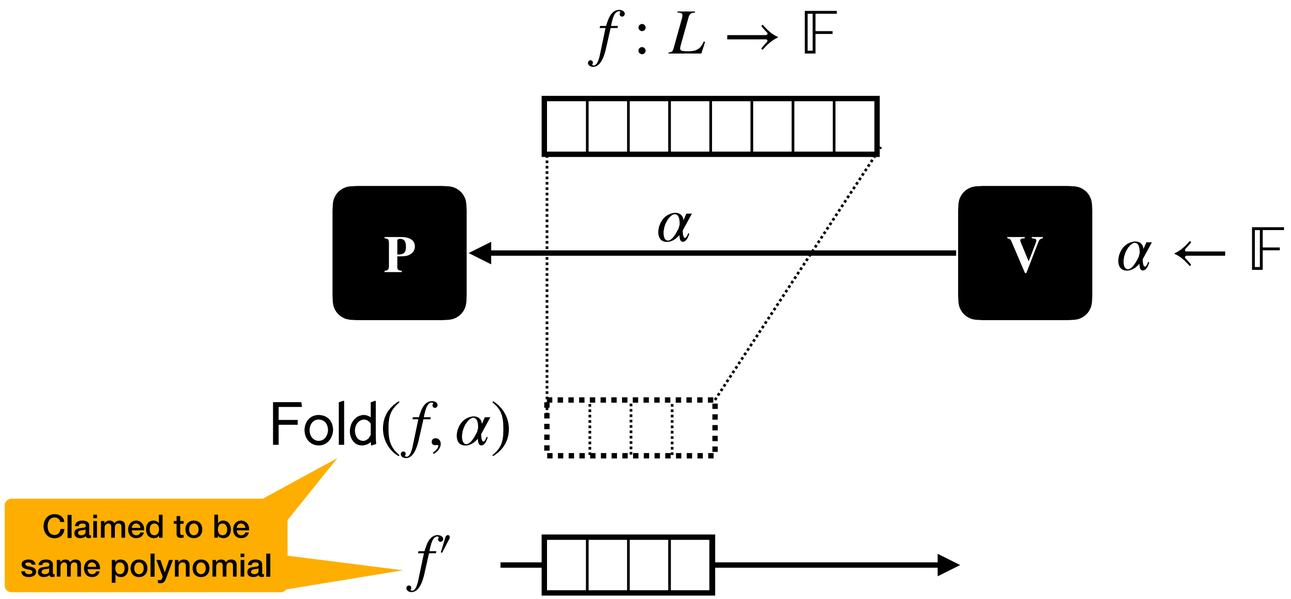
Review: FRI iteration



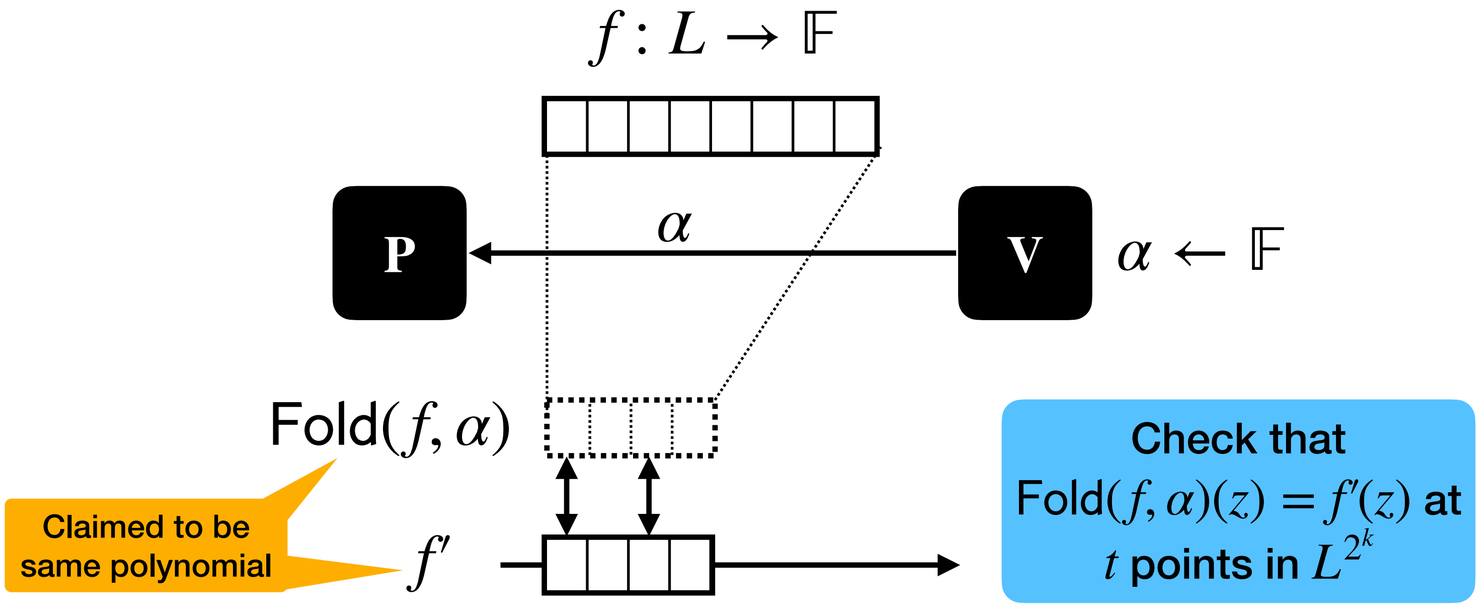
Review: FRI iteration



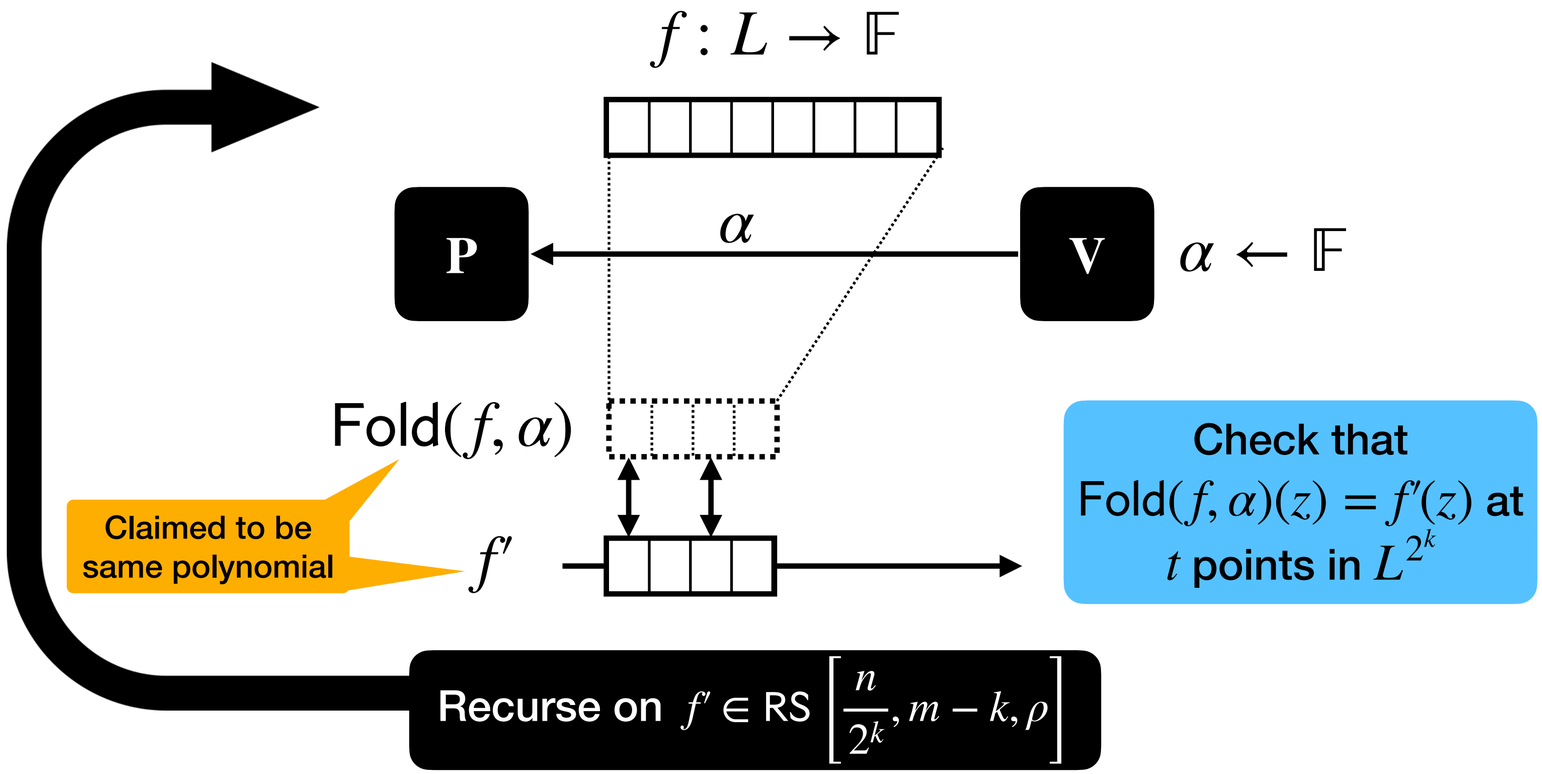
Review: FRI iteration



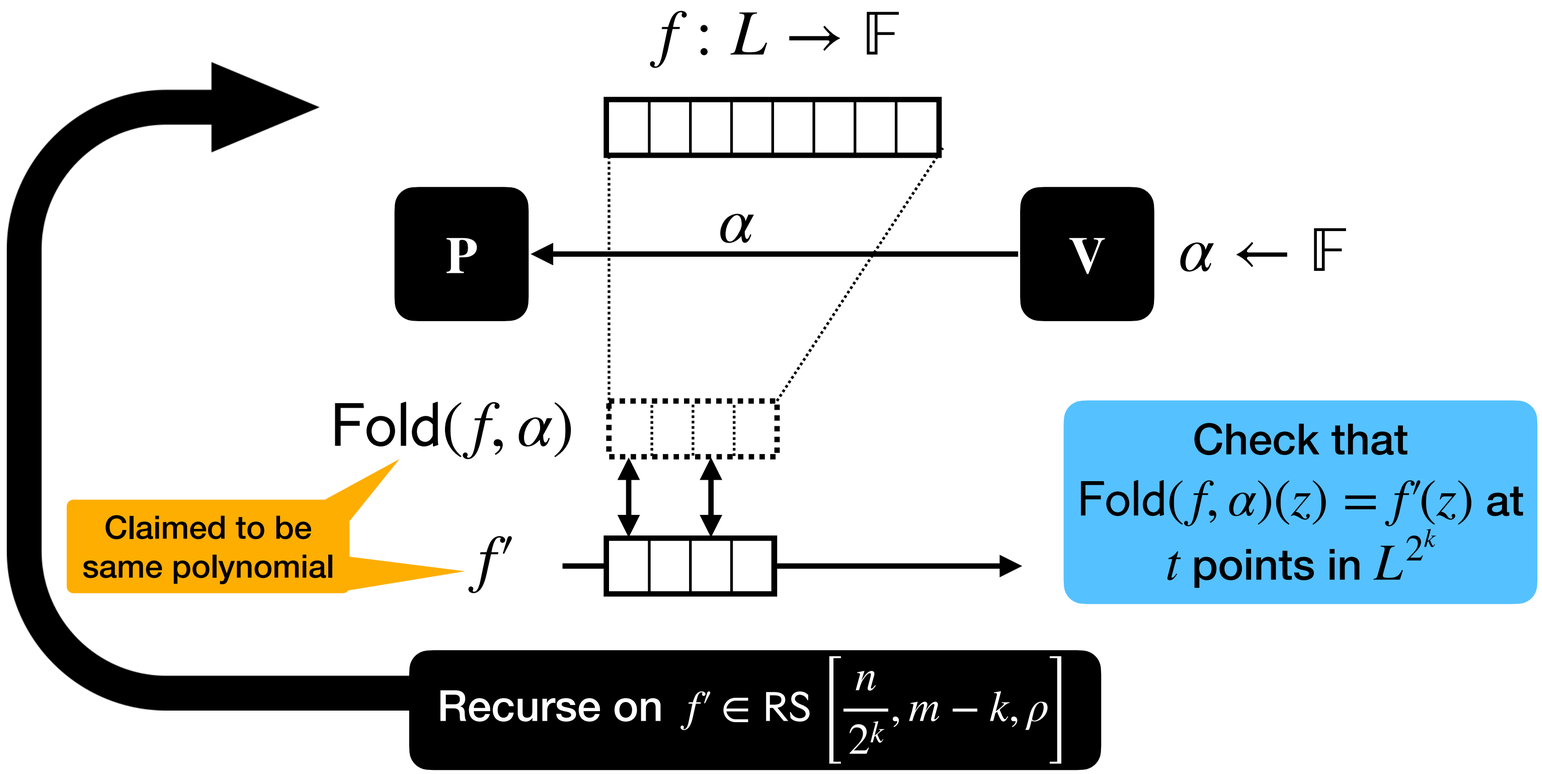
Review: FRI iteration



Review: FRI iteration

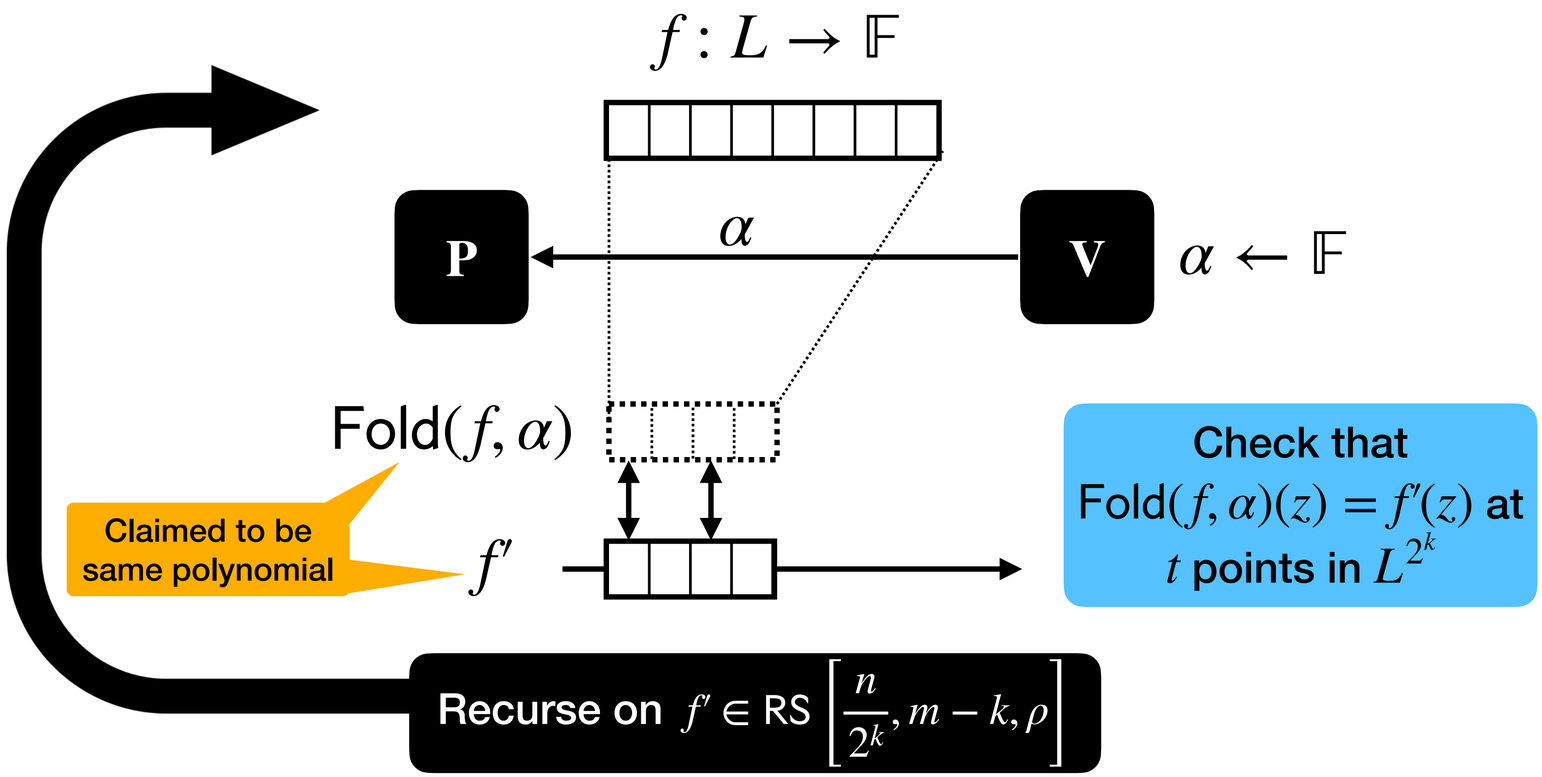


Review: FRI iteration



Disclaimer: in full FRI consistency checks are correlated between rounds.

Review: FRI iteration



Soundness:

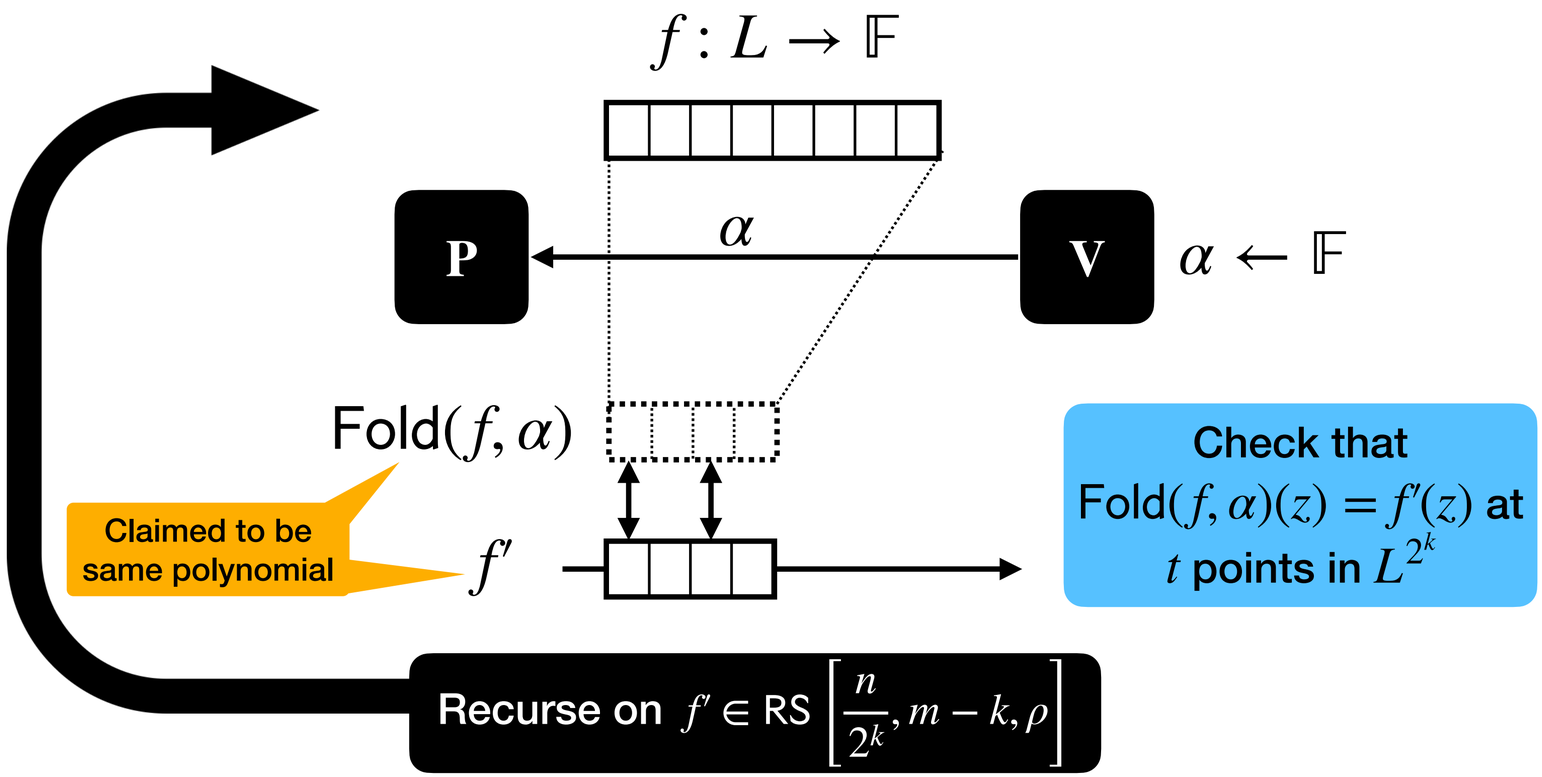
Suppose that $f' \in \mathrm{RS}[n/2^k, m-k, \rho]$.

If f is δ-far from $\mathrm{RS}[n, m, \rho]$,

Fold(f, α) must be δ-far from $\mathrm{RS}[n/2^k, m-k, \rho]$

Disclaimer: in full FRI consistency checks are correlated between rounds.

Review: FRI iteration



Disclaimer: in full FRI consistency checks are correlated between rounds.

Soundness:

Suppose that $f' \in \mathrm{RS}[n/2^k, m-k, \rho]$.

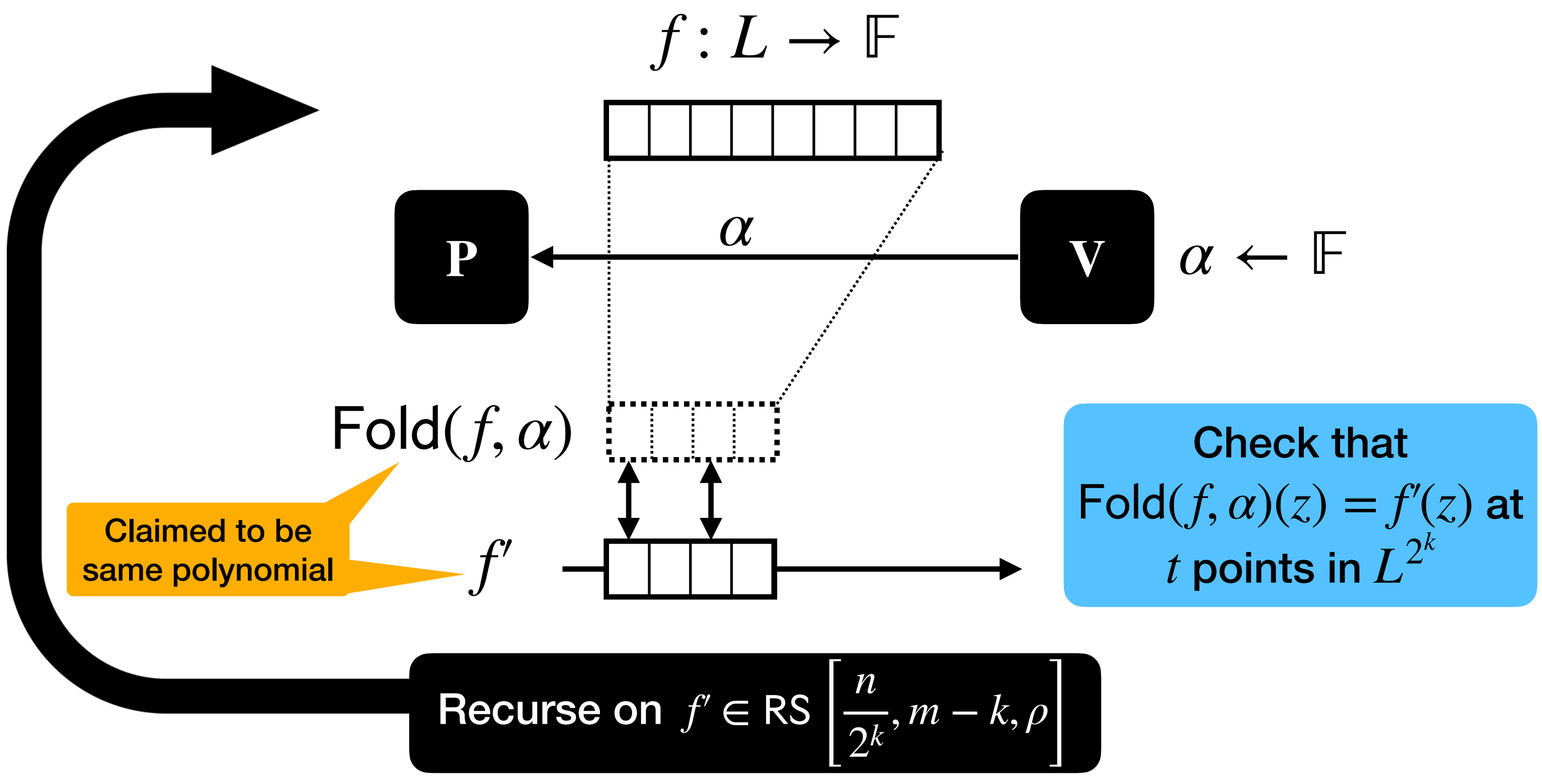
If f is δ-far from $\mathrm{RS}[n, m, \rho]$,

$\mathrm{Fold}(f, \alpha)$ must be δ-far from $\mathrm{RS}[n/2^k, m-k, \rho]$

Then, f' and $\mathrm{Fold}(f, \alpha)$ differ on a δ-fraction.

Soundness error is $(1-\delta)^t$

Review: FRI iteration



Disclaimer: in full FRI consistency checks are correlated between rounds.

Soundness:

Suppose that $f' \in \mathrm{RS}[n/2^k, m-k, \rho]$.

If f is δ-far from $\mathrm{RS}[n, m, \rho]$,

Fold(f, α) must be δ-far from $\mathrm{RS}[n/2^k, m-k, \rho]$

Then, f' and Fold(f, α) differ on a δ-fraction.

Soundness error is $(1-\delta)^t$

To get soundness error $\varepsilon_{\mathrm{RBR}} \leq 2^{-\lambda}$:

set $\delta := 1 - \sqrt{\rho}$ and $t := \dfrac{\lambda}{-\log\sqrt{\rho}}$

List-RLC lemma and List-Fold

Implied by mutual correlated agreement

$\Lambda(\mathscr{C}, f, \delta)$ is the list of codewords of $\mathscr{C}$ that are δ-close to f

List-RLC lemma and List-Fold

Implied by mutual correlated agreement

$\Lambda(\mathscr{C}, f, \delta)$ is the list of codewords of $\mathscr{C}$ that are δ-close to f

$f_1, \ldots, f_m \colon L \to \mathbb{F}$

List-RLC lemma and List-Fold

Implied by mutual correlated agreement

$\Lambda(\mathscr{C}, f, \delta)$ is the list of codewords of $\mathscr{C}$ that are δ-close to f

$f_1, \ldots, f_m \colon L \to \mathbb{F}$

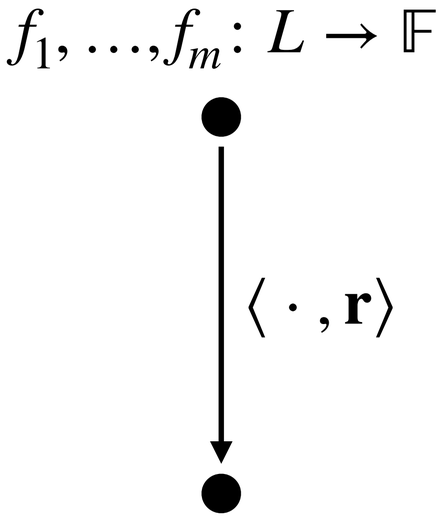
- Taking lists and (random) combinations **commute** (if mutual correlated agreement holds).

List-RLC lemma and List-Fold

Implied by mutual correlated agreement

$\Lambda(\mathscr{C}, f, \delta)$ is the list of codewords of $\mathscr{C}$ that are δ-close to f

- Taking lists and (random) combinations **commute** (if mutual correlated agreement holds).

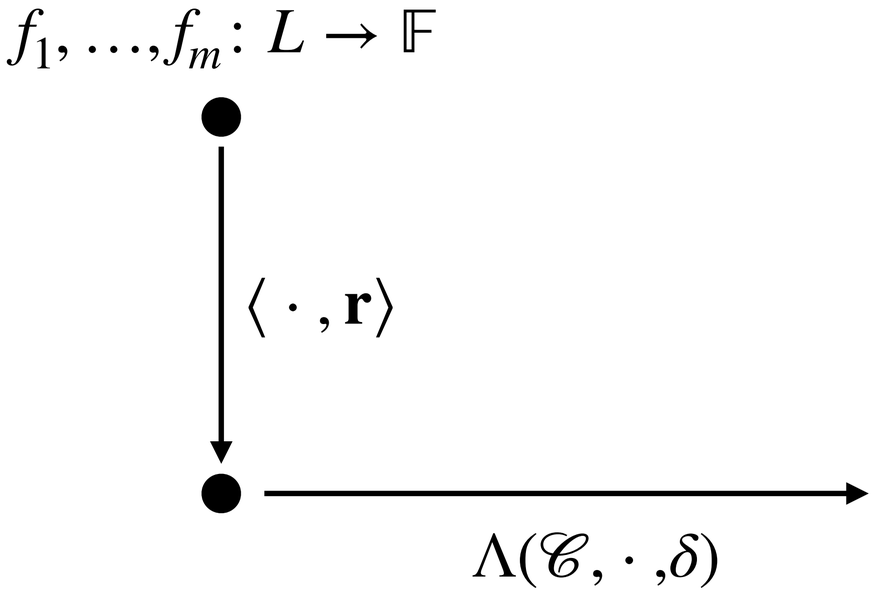


List-RLC lemma and List-Fold

Implied by mutual correlated agreement

$\Lambda(\mathscr{C}, f, \delta)$ is the list of codewords of $\mathscr{C}$ that are δ-close to f

- Taking lists and (random) combinations **commute** (if mutual correlated agreement holds).

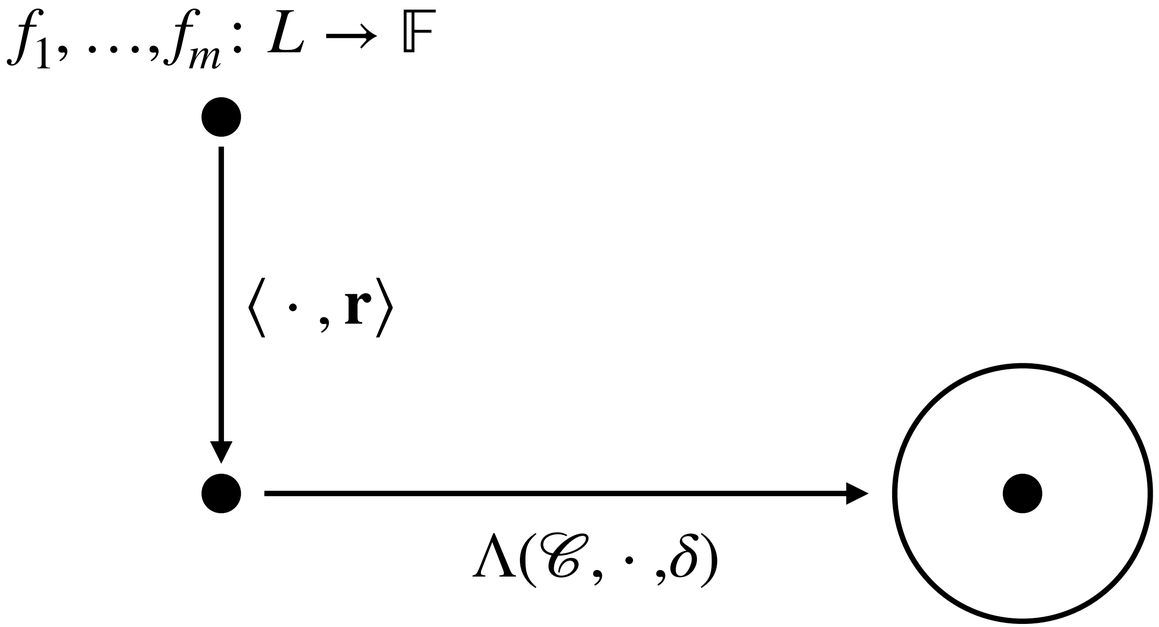


List-RLC lemma and List-Fold

Implied by mutual correlated agreement

$\Lambda(\mathscr{C}, f, \delta)$ is the list of codewords of $\mathscr{C}$ that are δ-close to f

- Taking lists and (random) combinations **commute** (if mutual correlated agreement holds).

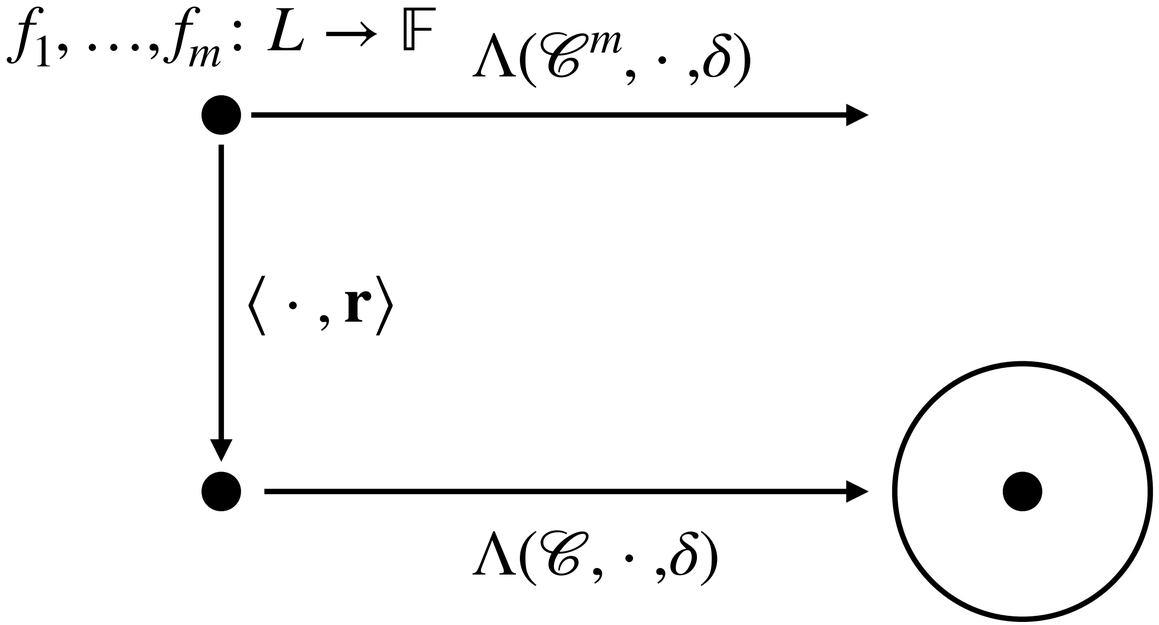


List-RLC lemma and List-Fold

Implied by mutual correlated agreement

$\Lambda(\mathscr{C}, f, \delta)$ is the list of codewords of $\mathscr{C}$ that are δ-close to f

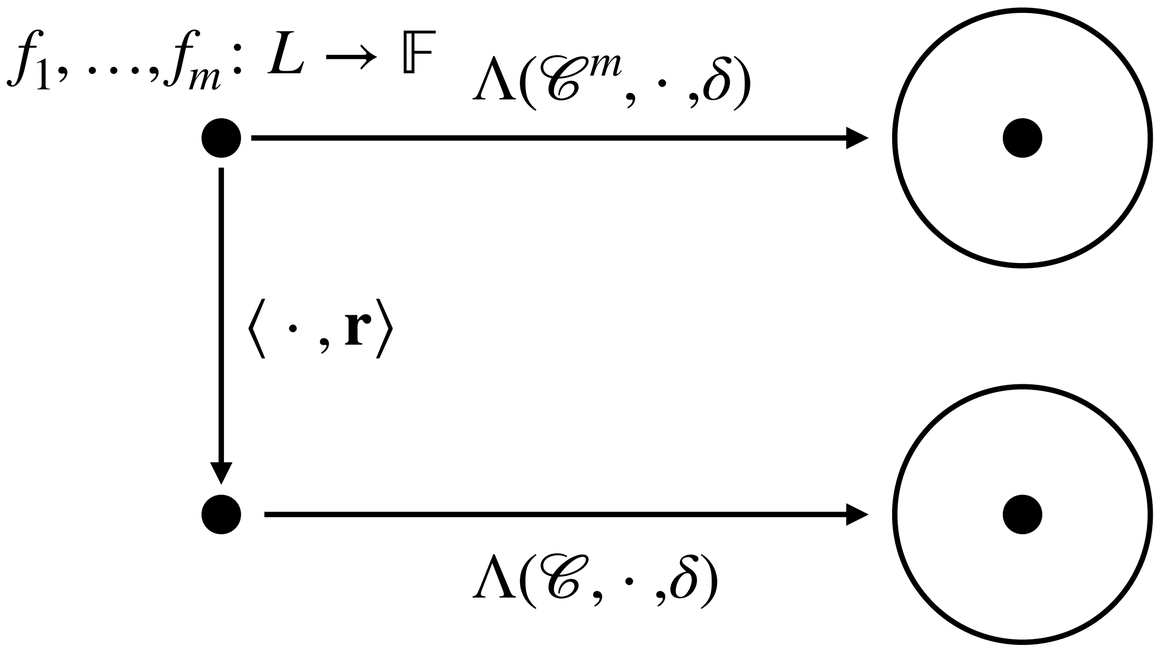
- Taking lists and (random) combinations **commute** (if mutual correlated agreement holds).



List-RLC lemma and List-Fold

Implied by mutual correlated agreement

$\Lambda(\mathscr{C}, f, \delta)$ is the list of codewords of $\mathscr{C}$ that are δ-close to f

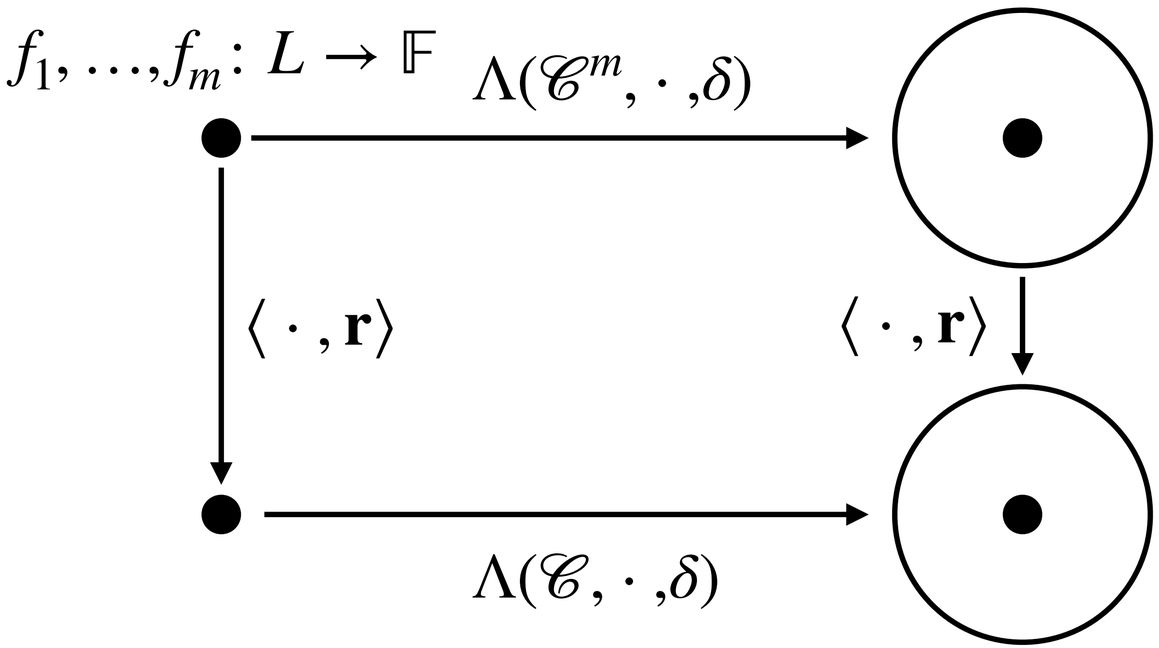


- Taking lists and (random) combinations **commute** (if mutual correlated agreement holds).

List-RLC lemma and List-Fold

Implied by mutual correlated agreement

$\Lambda(\mathscr{C}, f, \delta)$ is the list of codewords of $\mathscr{C}$ that are δ-close to f

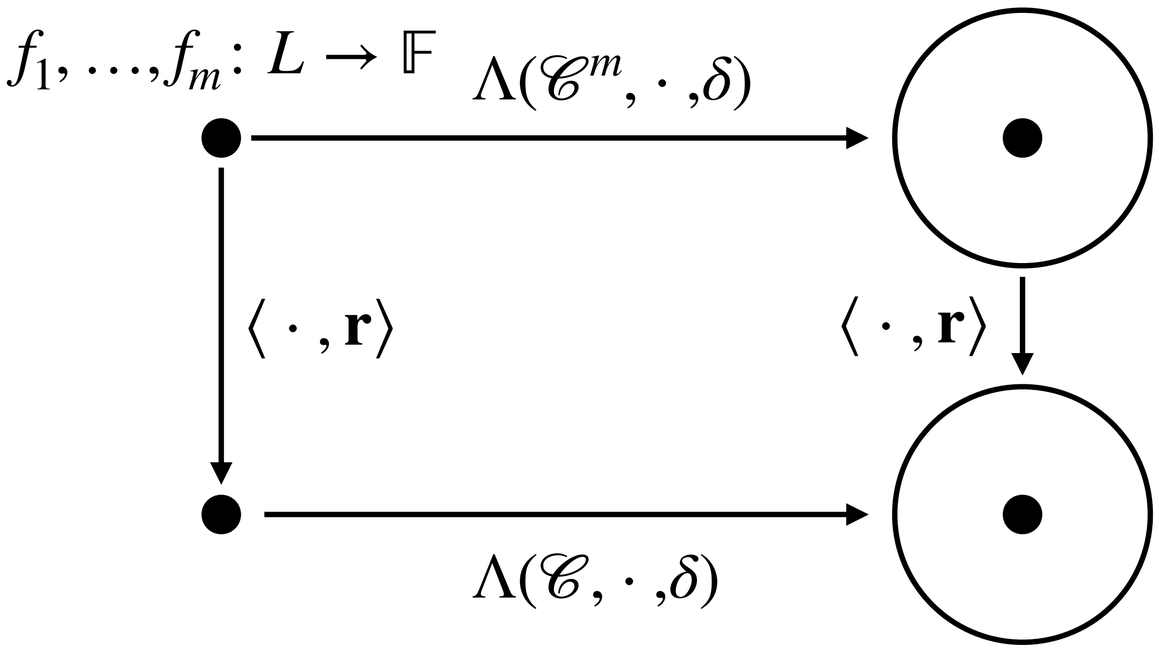


- Taking lists and (random) combinations **commute** (if mutual correlated agreement holds).

List-RLC lemma and List-Fold

Implied by mutual correlated agreement

$\Lambda(\mathscr{C}, f, \delta)$ is the list of codewords of $\mathscr{C}$ that are δ-close to f

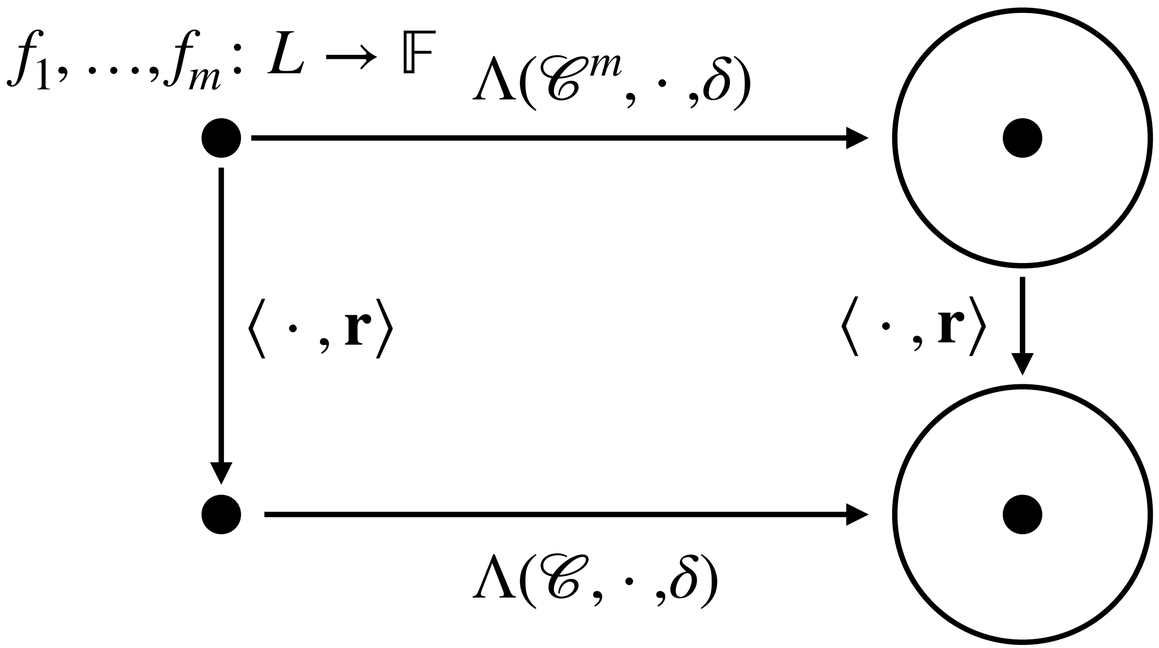


- Taking lists and (random) combinations **commute** (if mutual correlated agreement holds).
- Random linear combination version: w.h.p. over $\mathbf{r}$: $\Lambda(\mathscr{C}, \langle \mathbf{f}, \mathbf{r} \rangle, \delta) = \left\{ \langle \mathbf{u}, \mathbf{r} \rangle : \mathbf{u} \in \Lambda(\mathscr{C}^m, \mathbf{f}, \delta) \right\}$

List-RLC lemma and List-Fold

Implied by mutual correlated agreement

$\Lambda(\mathscr{C}, f, \delta)$ is the list of codewords of $\mathscr{C}$ that are δ-close to f

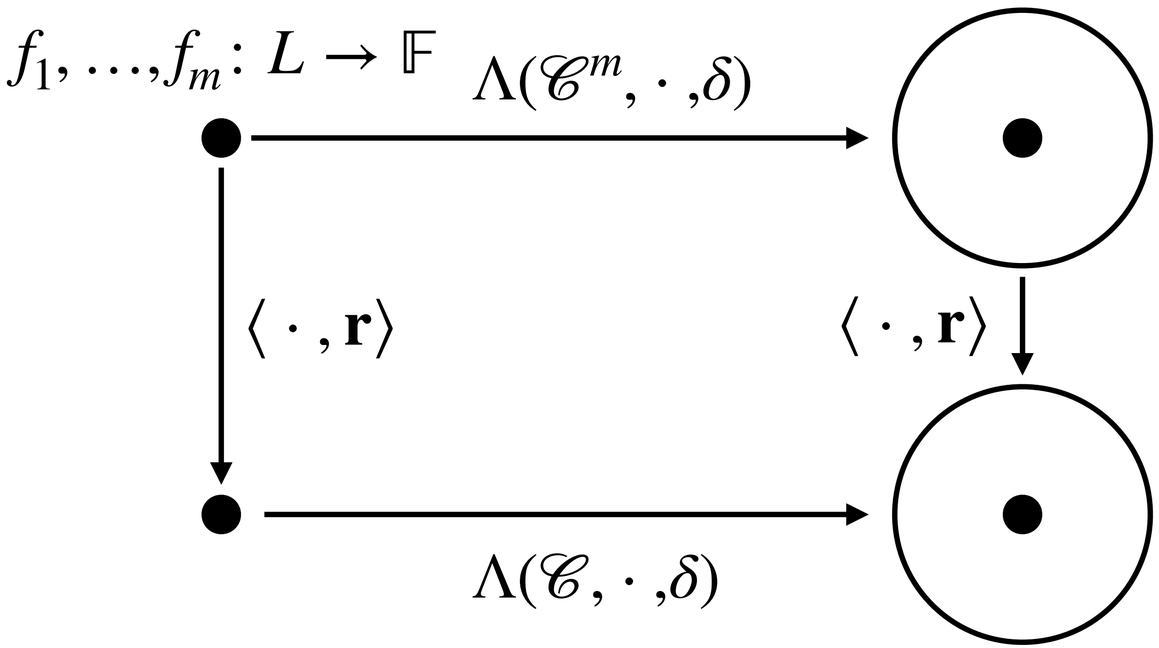


- Taking lists and (random) combinations **commute** (if mutual correlated agreement holds).
- Random linear combination version: w.h.p. over $\mathbf{r}$:
  $\Lambda(\mathscr{C}, \langle \mathbf{f}, \mathbf{r} \rangle, \delta) = \left\{ \langle \mathbf{u}, \mathbf{r} \rangle : \mathbf{u} \in \Lambda(\mathscr{C}^m, \mathbf{f}, \delta) \right\}$
- Folding version: w.h.p. over α:
  $\Lambda(\mathscr{C}, \mathsf{Fold}(f, \alpha), \delta) = \left\{ \mathsf{Fold}(u, \alpha) : u \in \Lambda(\mathscr{C}, f, \delta) \right\}$

List-RLC lemma and List-Fold

Implied by mutual correlated agreement

$\Lambda(\mathscr{C}, f, \delta)$ is the list of codewords of $\mathscr{C}$ that are δ-close to f

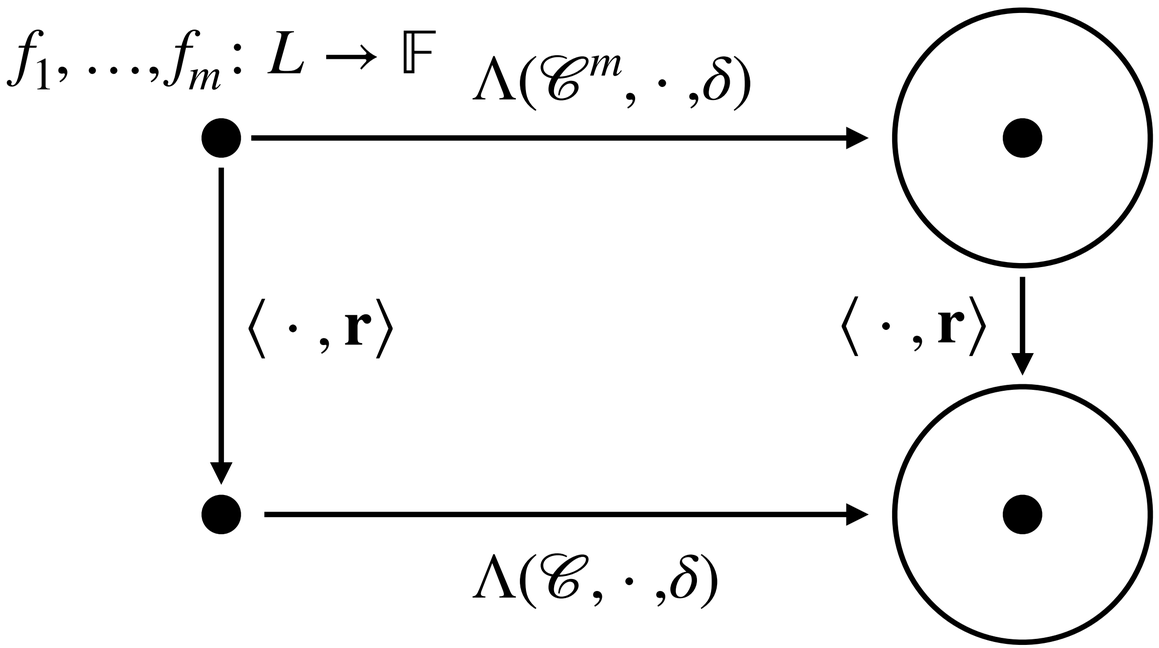


- Taking lists and (random) combinations **commute** (if mutual correlated agreement holds).
- Random linear combination version: w.h.p. over $\mathbf{r}$:
  $\Lambda(\mathscr{C}, \langle \mathbf{f}, \mathbf{r} \rangle, \delta) = \left\{ \langle \mathbf{u}, \mathbf{r} \rangle : \mathbf{u} \in \Lambda(\mathscr{C}^m, \mathbf{f}, \delta) \right\}$
- Folding version: w.h.p. over α:
  $\Lambda(\mathscr{C}, \mathsf{Fold}(f, \alpha), \delta) = \left\{ \mathsf{Fold}(u, \alpha) : u \in \Lambda(\mathscr{C}, f, \delta) \right\}$
- Alternatively, each term in the l.h.s can be “explained" by terms in the r.h.s.

List-RLC lemma and List-Fold

Implied by mutual correlated agreement

$\Lambda(\mathscr{C}, f, \delta)$ is the list of codewords of $\mathscr{C}$ that are δ-close to f

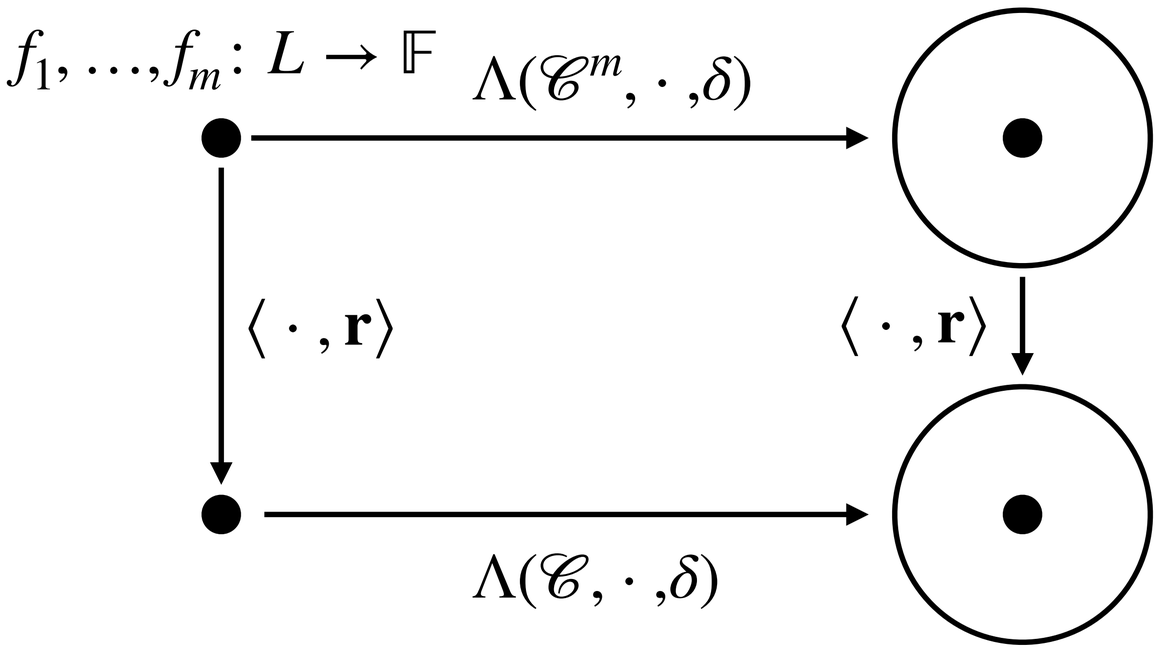


- Taking lists and (random) combinations **commute** (if mutual correlated agreement holds).
- Random linear combination version: w.h.p. over $\mathbf{r}$:
  $\Lambda(\mathscr{C}, \langle \mathbf{f}, \mathbf{r} \rangle, \delta) = \left\{ \langle \mathbf{u}, \mathbf{r} \rangle : \mathbf{u} \in \Lambda(\mathscr{C}^m, \mathbf{f}, \delta) \right\}$
- Folding version: w.h.p. over α:
  $\Lambda(\mathscr{C}, \mathsf{Fold}(f, \alpha), \delta) = \left\{ \mathsf{Fold}(u, \alpha) : u \in \Lambda(\mathscr{C}, f, \delta) \right\}$
- Alternatively, each term in the l.h.s can be “explained" by terms in the r.h.s.
- We show correlated agreement implies mutual correlated agreement in *unique decoding.*

List-RLC lemma and List-Fold

Implied by mutual correlated agreement

$\Lambda(\mathscr{C}, f, \delta)$ is the list of codewords of $\mathscr{C}$ that are δ-close to f



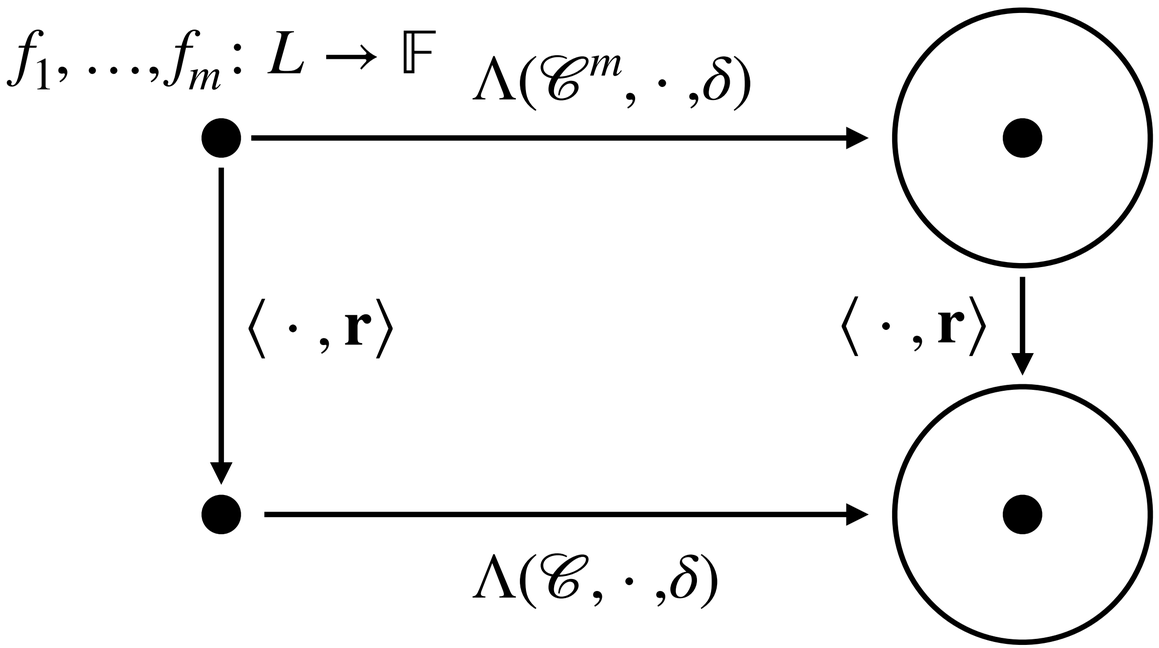
Stronger than what is required for STIR's soundness

- Taking lists and (random) combinations **commute** (if mutual correlated agreement holds).
- Random linear combination version: w.h.p. over $\mathbf{r}$:
  $\Lambda(\mathscr{C}, \langle \mathbf{f}, \mathbf{r} \rangle, \delta) = \left\{ \langle \mathbf{u}, \mathbf{r} \rangle : \mathbf{u} \in \Lambda(\mathscr{C}^m, \mathbf{f}, \delta) \right\}$
- Folding version: w.h.p. over α:
  $\Lambda(\mathscr{C}, \mathsf{Fold}(f, \alpha), \delta) = \left\{ \mathsf{Fold}(u, \alpha) : u \in \Lambda(\mathscr{C}, f, \delta) \right\}$
- Alternatively, each term in the l.h.s can be "explained" by terms in the r.h.s.
- We show correlated agreement implies mutual correlated agreement in *unique decoding.*

List-RLC lemma and List-Fold

Implied by mutual correlated agreement

$\Lambda(\mathscr{C}, f, \delta)$ is the list of codewords of $\mathscr{C}$ that are δ-close to f



Stronger than what is required for STIR's soundness

- Taking lists and (random) combinations **commute** (if mutual correlated agreement holds).
- Random linear combination version: w.h.p. over $\mathbf{r}$:
  $\Lambda(\mathscr{C}, \langle \mathbf{f}, \mathbf{r} \rangle, \delta) = \left\{ \langle \mathbf{u}, \mathbf{r} \rangle : \mathbf{u} \in \Lambda(\mathscr{C}^m, \mathbf{f}, \delta) \right\}$
- Folding version: w.h.p. over α:
  $\Lambda(\mathscr{C}, \mathsf{Fold}(f, \alpha), \delta) = \left\{ \mathsf{Fold}(u, \alpha) : u \in \Lambda(\mathscr{C}, f, \delta) \right\}$
- Alternatively, each term in the l.h.s can be "explained" by terms in the r.h.s.
- We show correlated agreement implies mutual correlated agreement in *unique decoding.*

Recent results show it holds up to 1.5 Johnson for general linear codes!