SNARKs in Relativized Worlds

Thank you for many
of the slides!

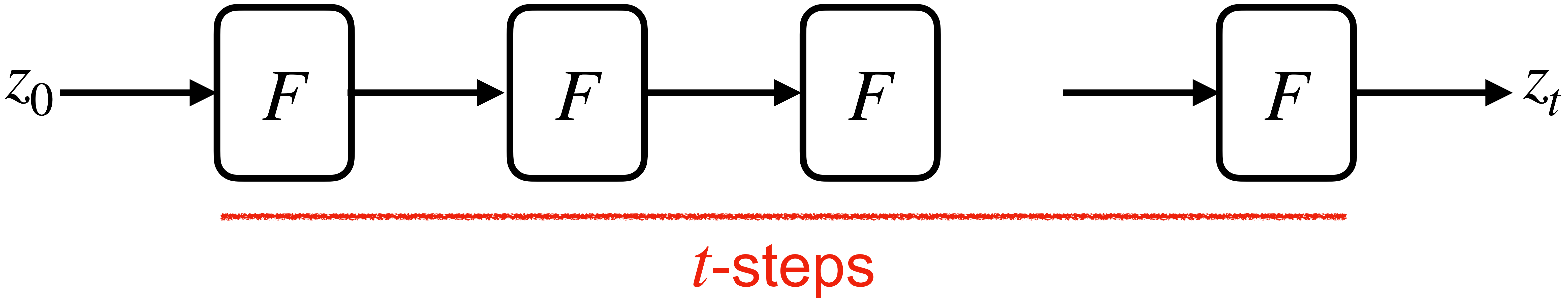
Megan Chen
Joint work with Alessandro Chiesa, Nicholas Spooner
Eurocrypt 2022 (ePrint: 2022/383)

Our setting: “streaming” verification of t-step NP computations

Our setting: “streaming” verification of t-step NP computations

Goal: Prove correctness of a t-step non-deterministic computation:

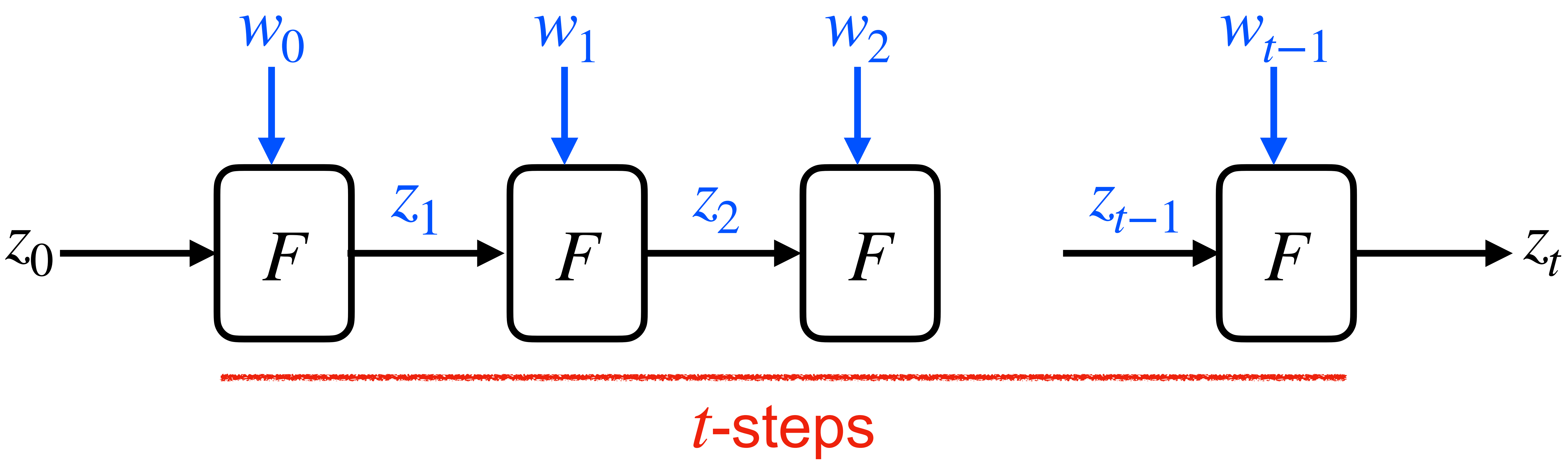
Given $F, z_0, z_t,$



Our setting: "streaming" verification of t-step NP computations

Goal: Prove correctness of a t-step non-deterministic computation:

Given F, z_0, z_t, check that $\exists z_1, \ldots, z_{t-1}, w_0, \ldots, w_{t-1} : \forall i \in [t], F(z_i, w_i) = z_{i+1}$

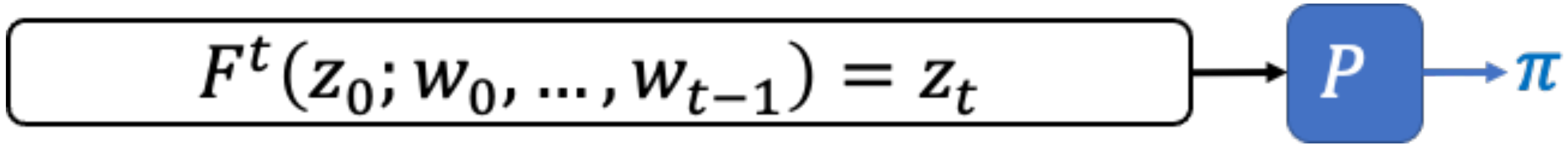


Our setting: "streaming" verification of t-step NP computations

Goal: Prove correctness of a t-step non-deterministic computation:

Given F, z_0, z_t, check that $\exists z_1, \ldots, z_{t-1}, w_0, \ldots, w_{t-1} : \forall i \in [t], F(z_i, w_i) = z_{i+1}$

Option 1: Monolithic proof

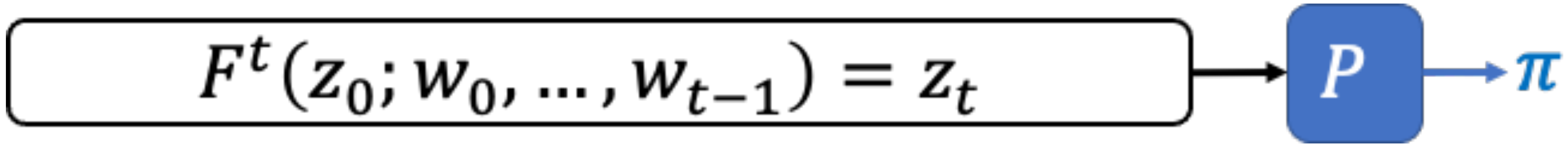


Our setting: "streaming" verification of t-step NP computations

Goal: Prove correctness of a t-step non-deterministic computation:

Given F, z_0, z_t, check that $\exists z_1, \ldots, z_{t-1}, w_0, \ldots, w_{t-1} : \forall i \in [t], F(z_i, w_i) = z_{i+1}$

Option 1: Monolithic proof



Issues:

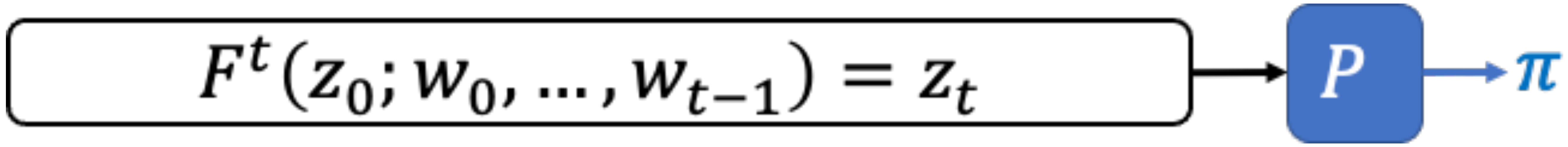
- (Typically) requires prover memory $\Omega(t)$
- Proving $t + 1$ steps requires recomputing entire proof

Our setting: "streaming" verification of t-step NP computations

Goal: Prove correctness of a t-step non-deterministic computation:

Given F, z_0, z_t, check that $\exists z_1, \ldots, z_{t-1}, w_0, \ldots, w_{t-1} : \forall i \in [t], F(z_i, w_i) = z_{i+1}$

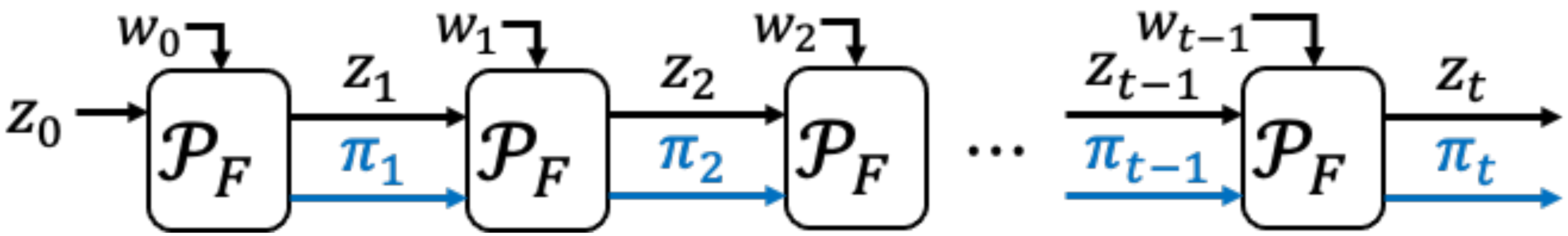
Option 1: Monolithic proof



Issues:

- (Typically) requires prover memory $\Omega(t)$
- Proving $t+1$ steps requires recomputing entire proof

Option 2: **Incrementally verifiable computation (IVC)** [Valiant08]

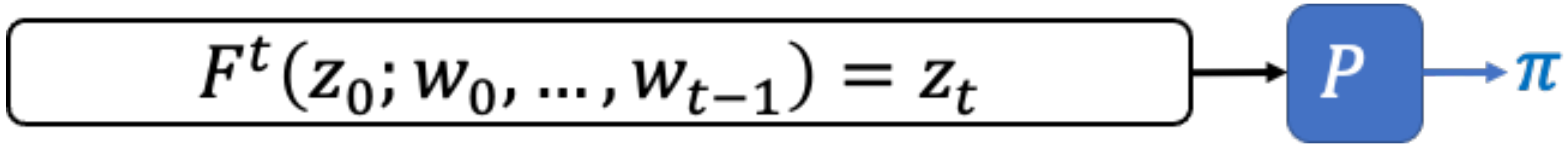


Our setting: “streaming” verification of t-step NP computations

Goal: Prove correctness of a t-step non-deterministic computation:

Given F, z_0, z_t, check that $\exists z_1, \ldots, z_{t-1}, w_0, \ldots, w_{t-1} : \forall i \in [t], F(z_i, w_i) = z_{i+1}$

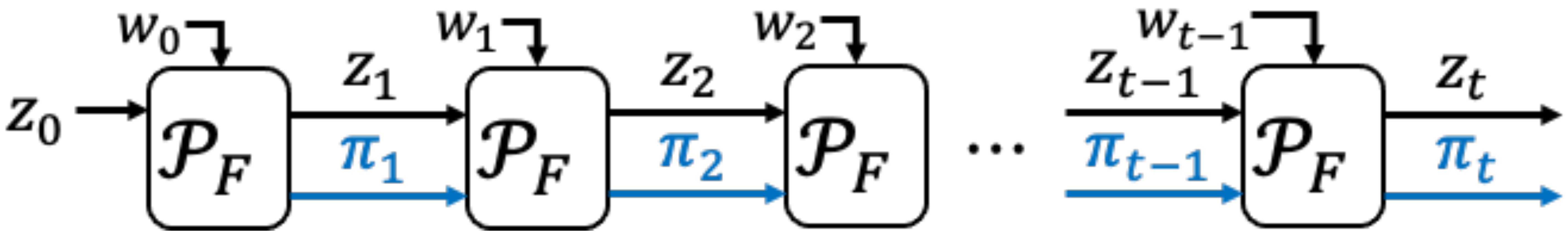
Option 1: Monolithic proof



Issues:

- (Typically) requires prover memory $\Omega(t)$
- Proving $t+1$ steps requires recomputing entire proof

Option 2: **Incrementally verifiable computation (IVC)** [Valiant08]



Proof-carrying data (PCD) [CT10, BCCT13]: generalizes path graph to DAG

Our setting: “streaming” verification of t-step NP computations

Applications include:

- “Succinct” blockchains
- SNARKs with low space complexity
- Verifiable delay functions
- Byzantine agreement
- ZK cluster computing
- Verifiable image editing
- Enforcing language semantics across trust boundaries

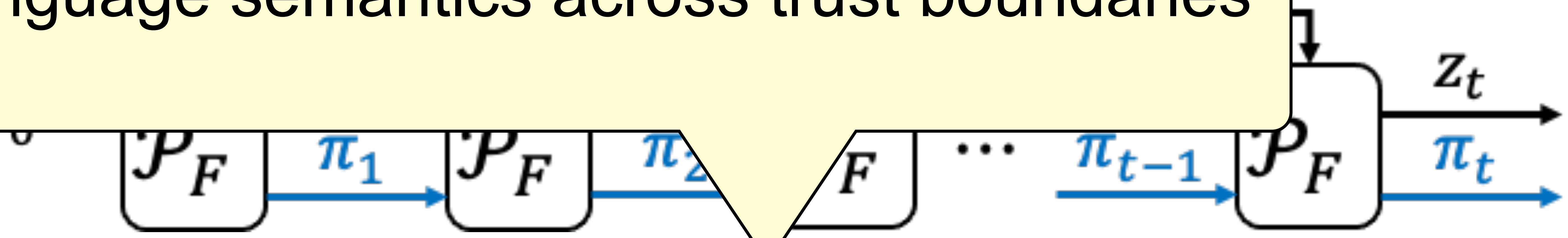
tation:

$[t], F(z_i, w_i) = z_{i+1}$

$\rightarrow \pi$

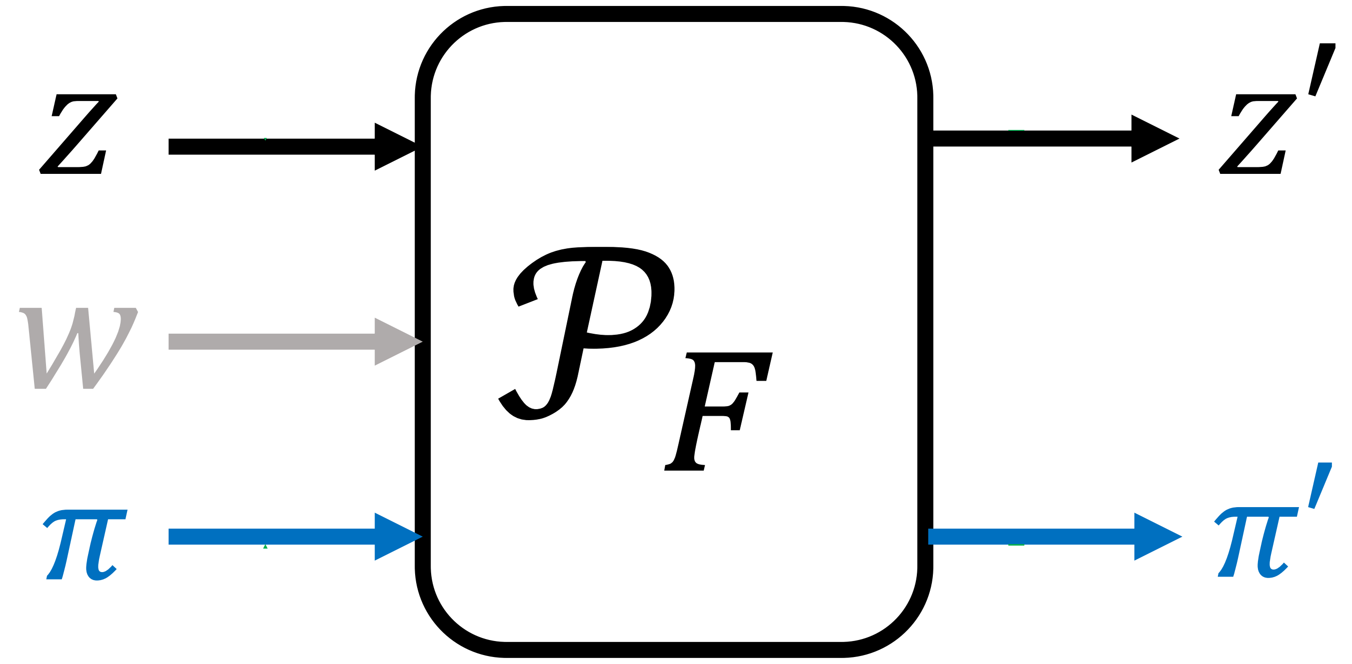
needed to compute F

t08]

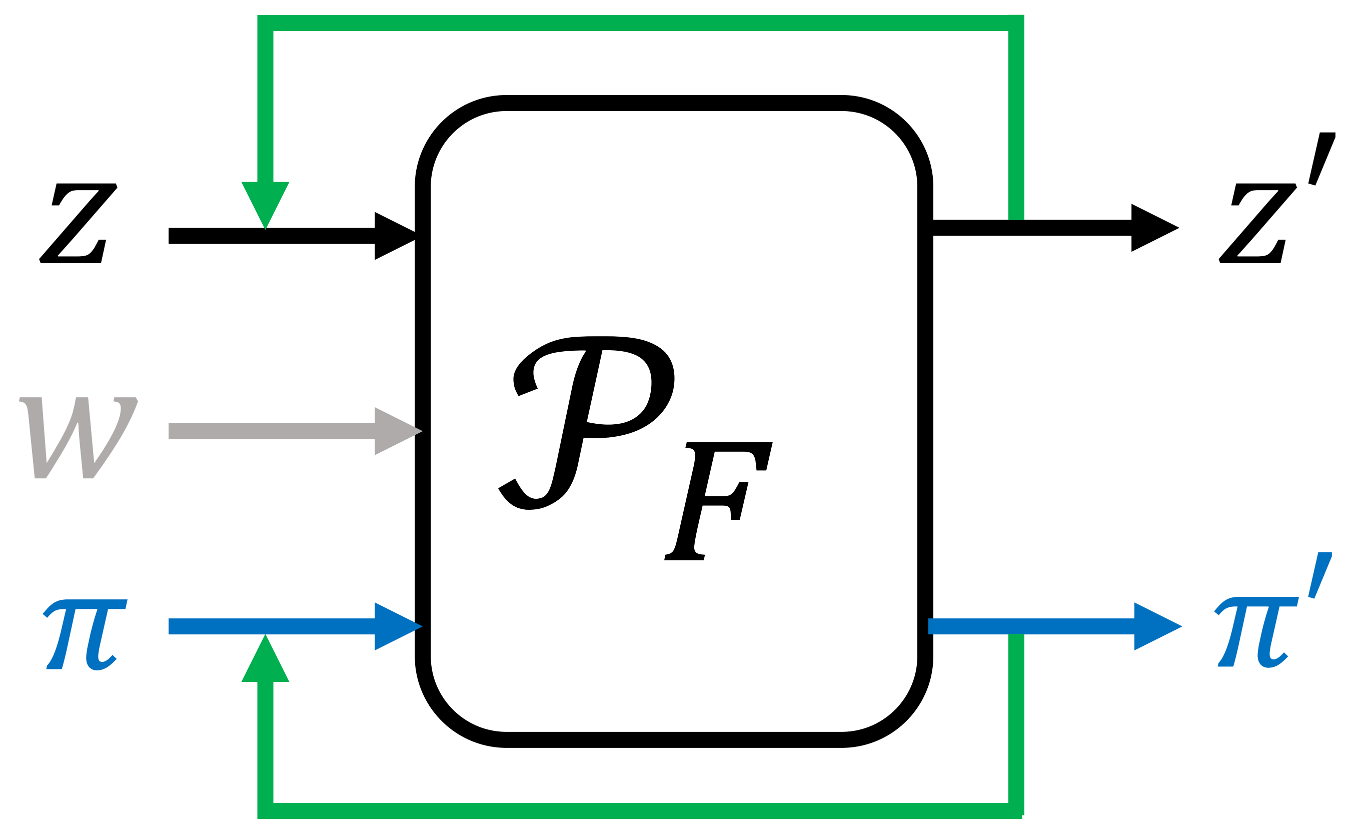


Proof-carrying data (PCD) [CT10, BCCT13]: generalizes path graph to DAG

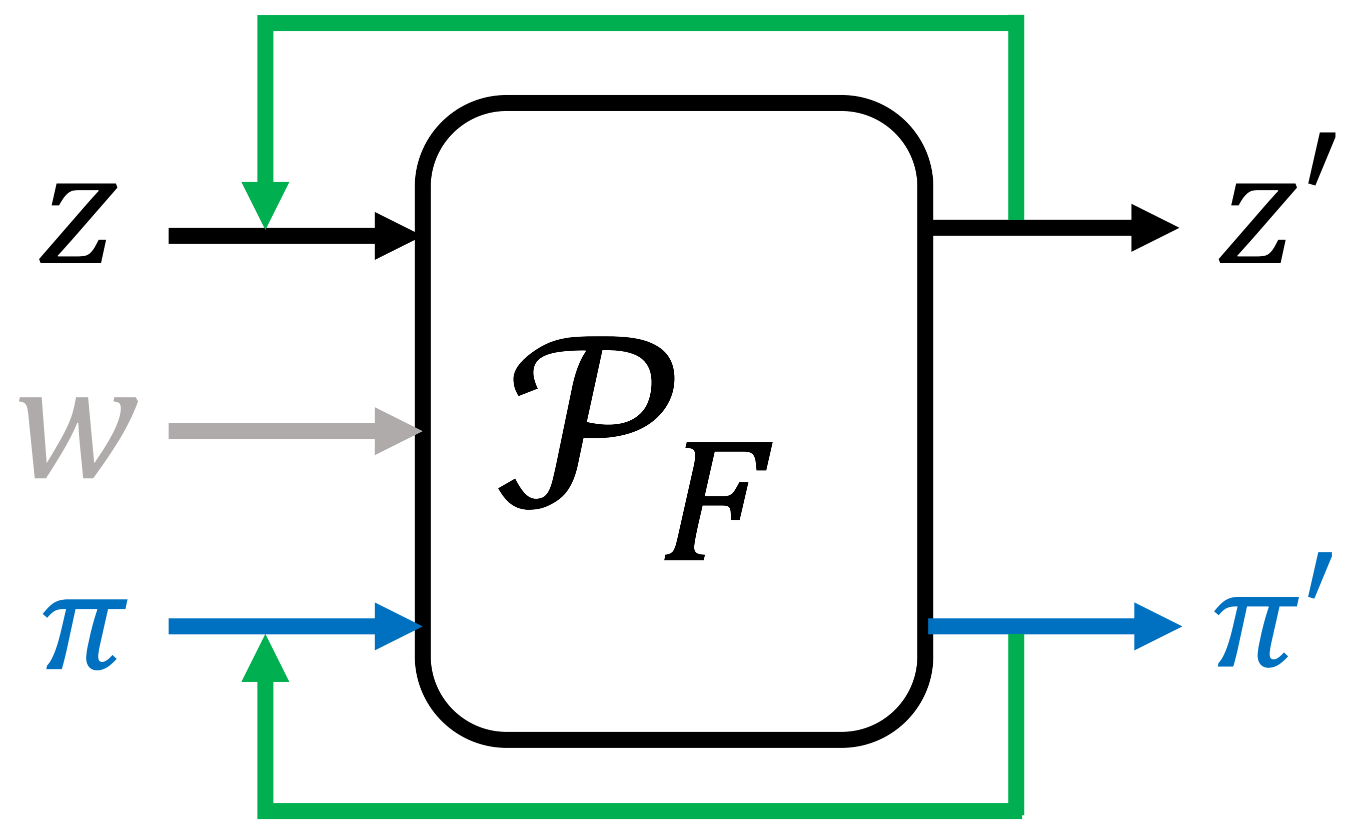
Defining incrementally verifiable computation (IVC)

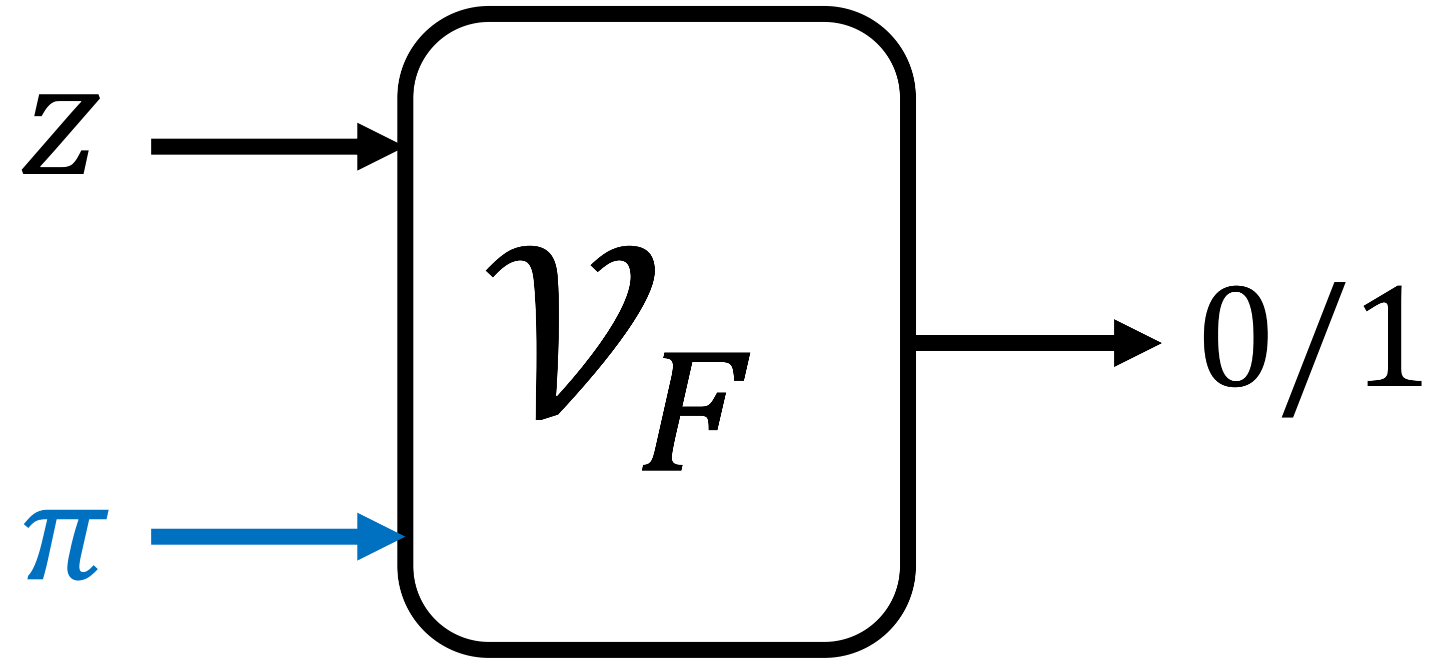


Defining incrementally verifiable computation (IVC)

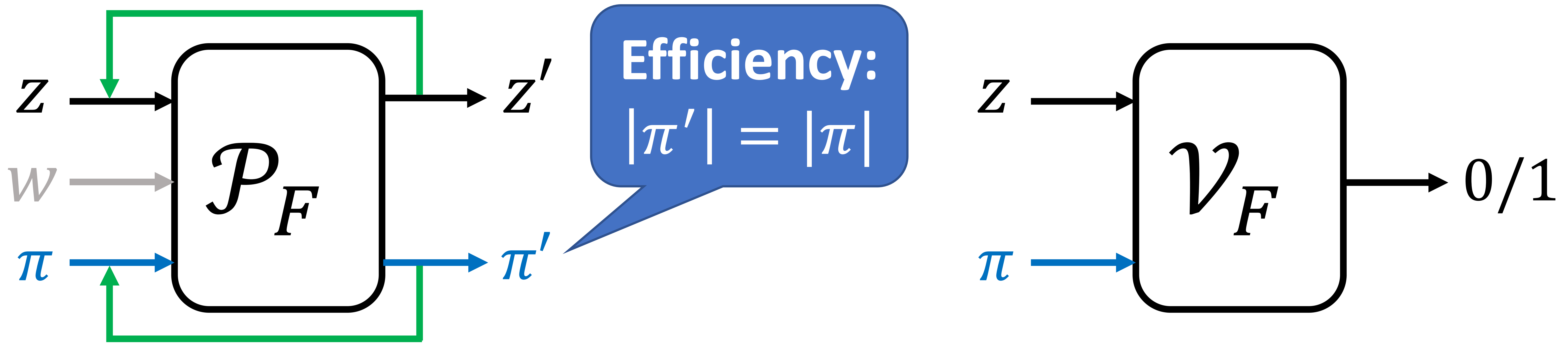


Defining incrementally verifiable computation (IVC)

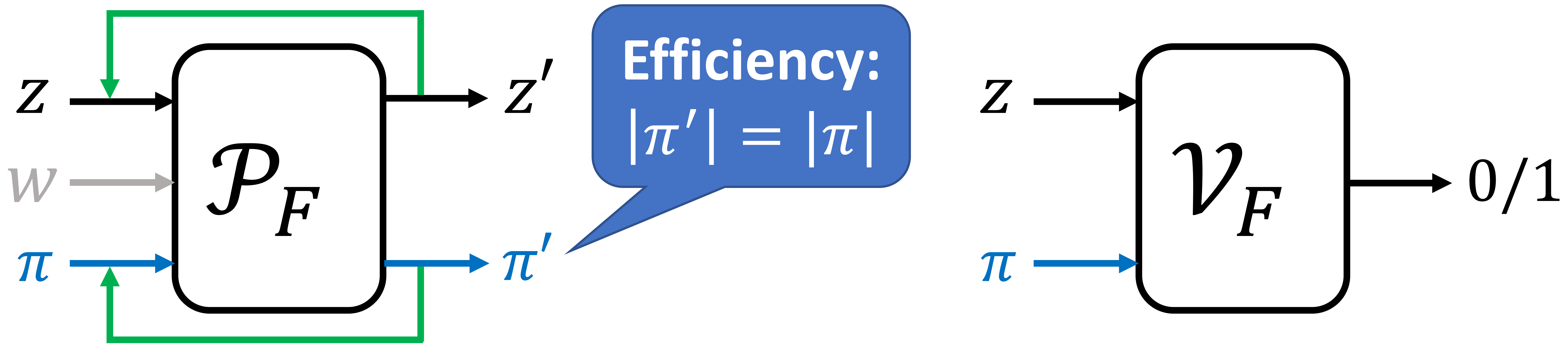




Defining incrementally verifiable computation (IVC)



Defining incrementally verifiable computation (IVC)



How to instantiate IVC?

IVC from SNARK

SNARK = Succinct Non-interactive ARgument of Knowledge

The IVC Prover…

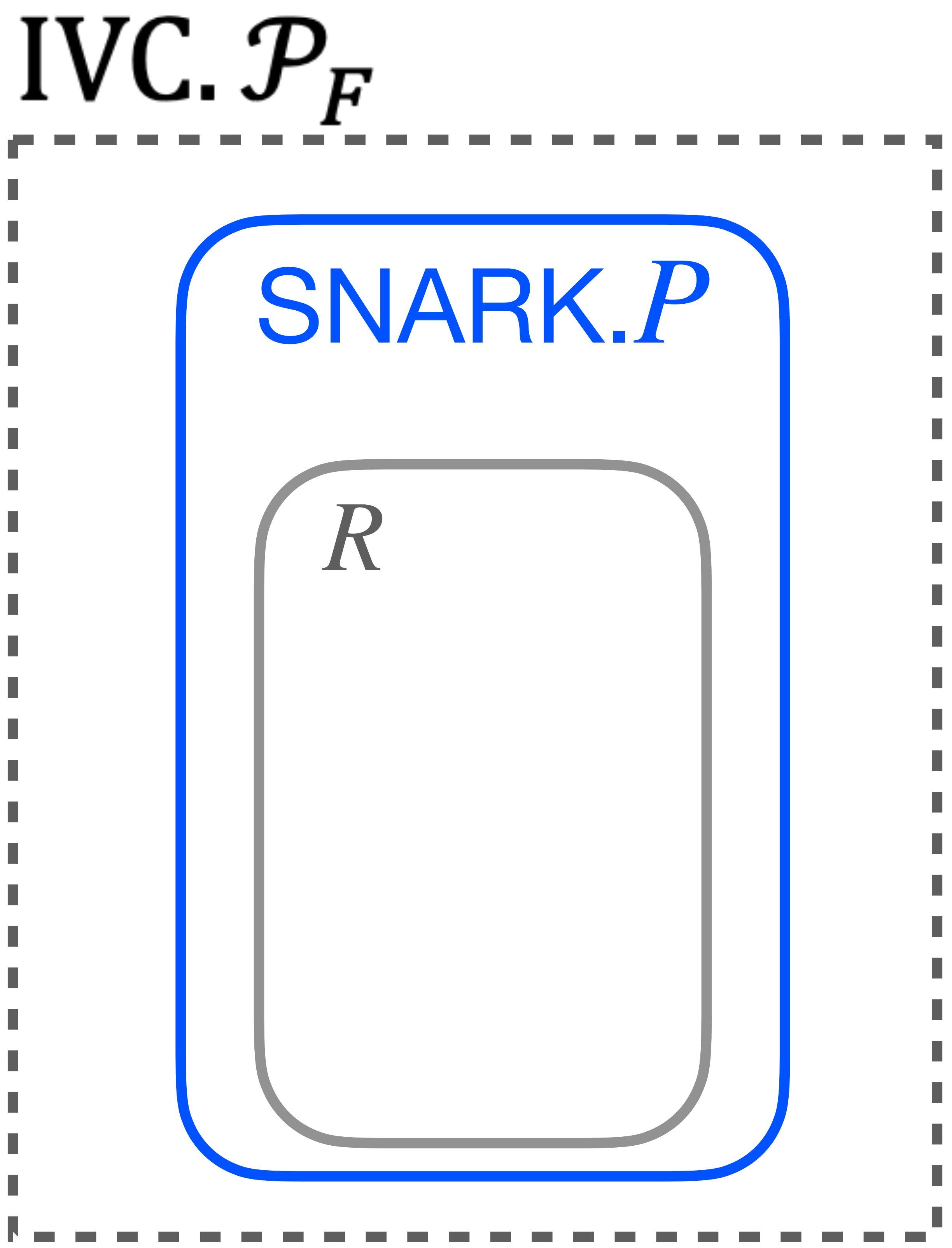
$\text{IVC}.\mathcal{P}_F$

IVC from SNARK

SNARK = Succinct Non-interactive ARgument of Knowledge

The IVC Prover…

Runs a SNARK P, proving the computation R:



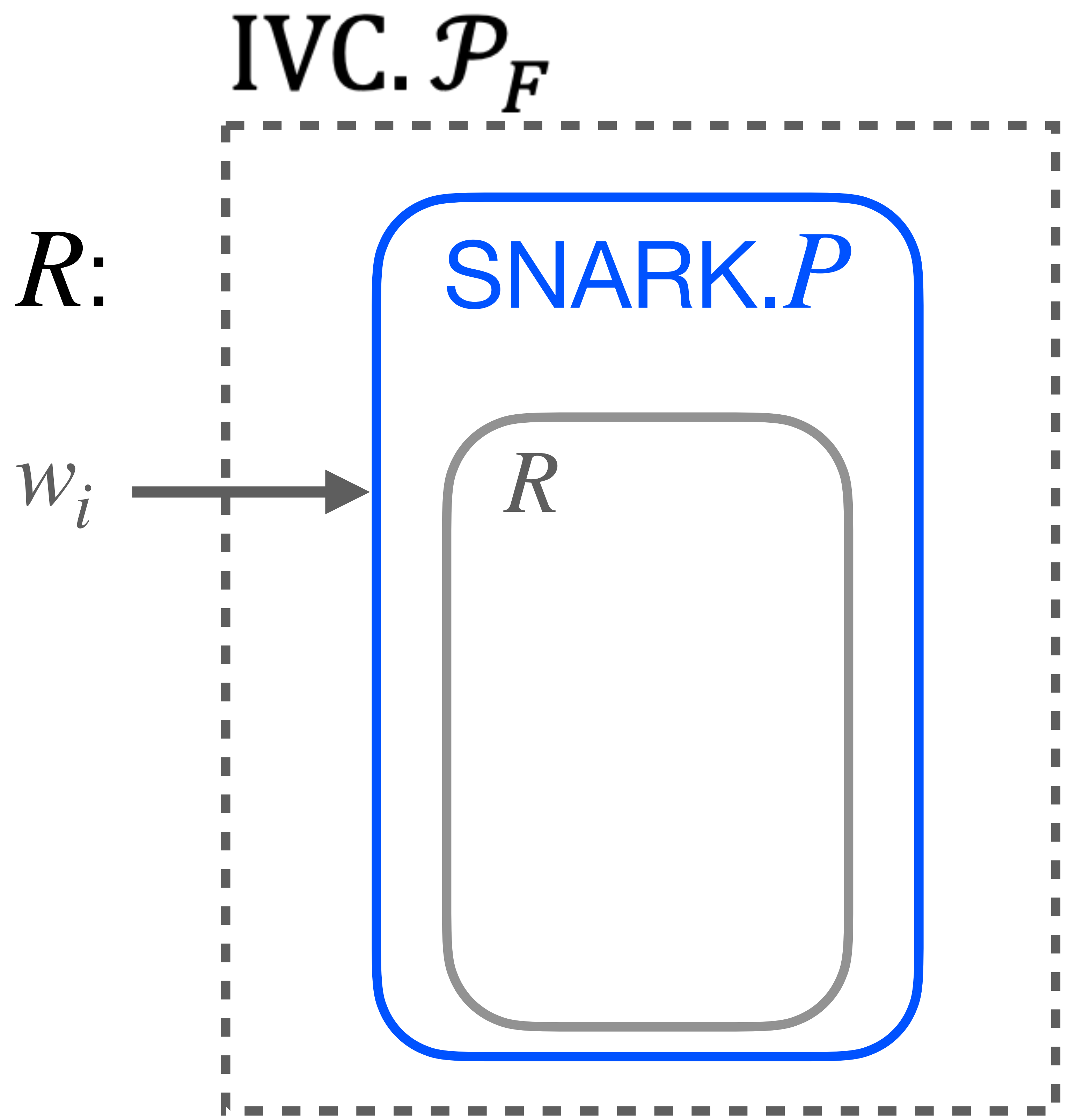
IVC from SNARK

SNARK = Succinct Non-interactive ARgument of Knowledge

The IVC Prover…

Runs a SNARK P, proving the computation R:

For all i, $\exists w_i$ s.t.



IVC from SNARK

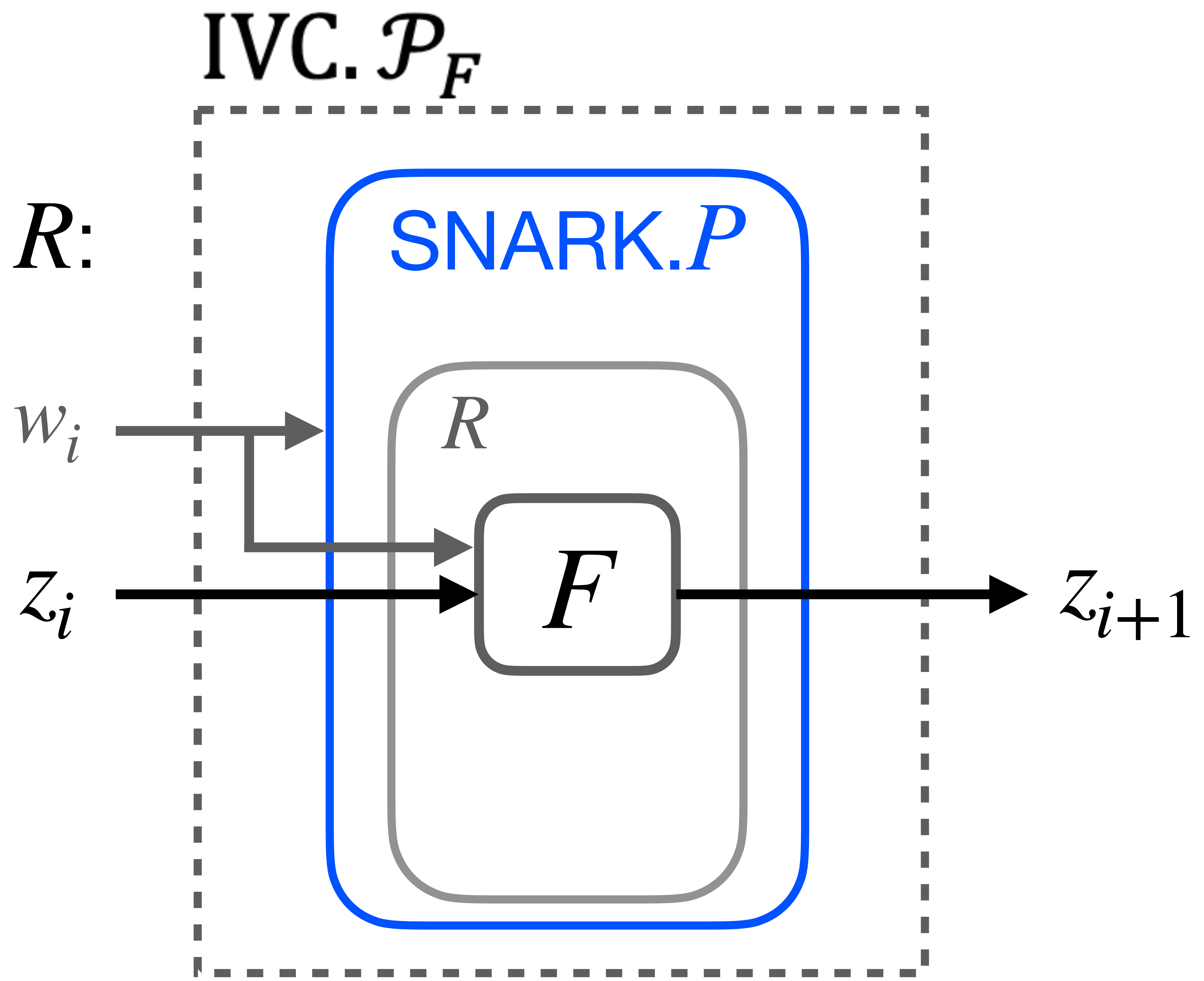
SNARK = Succinct Non-interactive ARgument of Knowledge

The IVC Prover…

Runs a SNARK P, proving the computation R:

For all i, $\exists w_i$ s.t.

- $F(z_i, w_i) = z_{i+1}$ and



IVC from SNARK

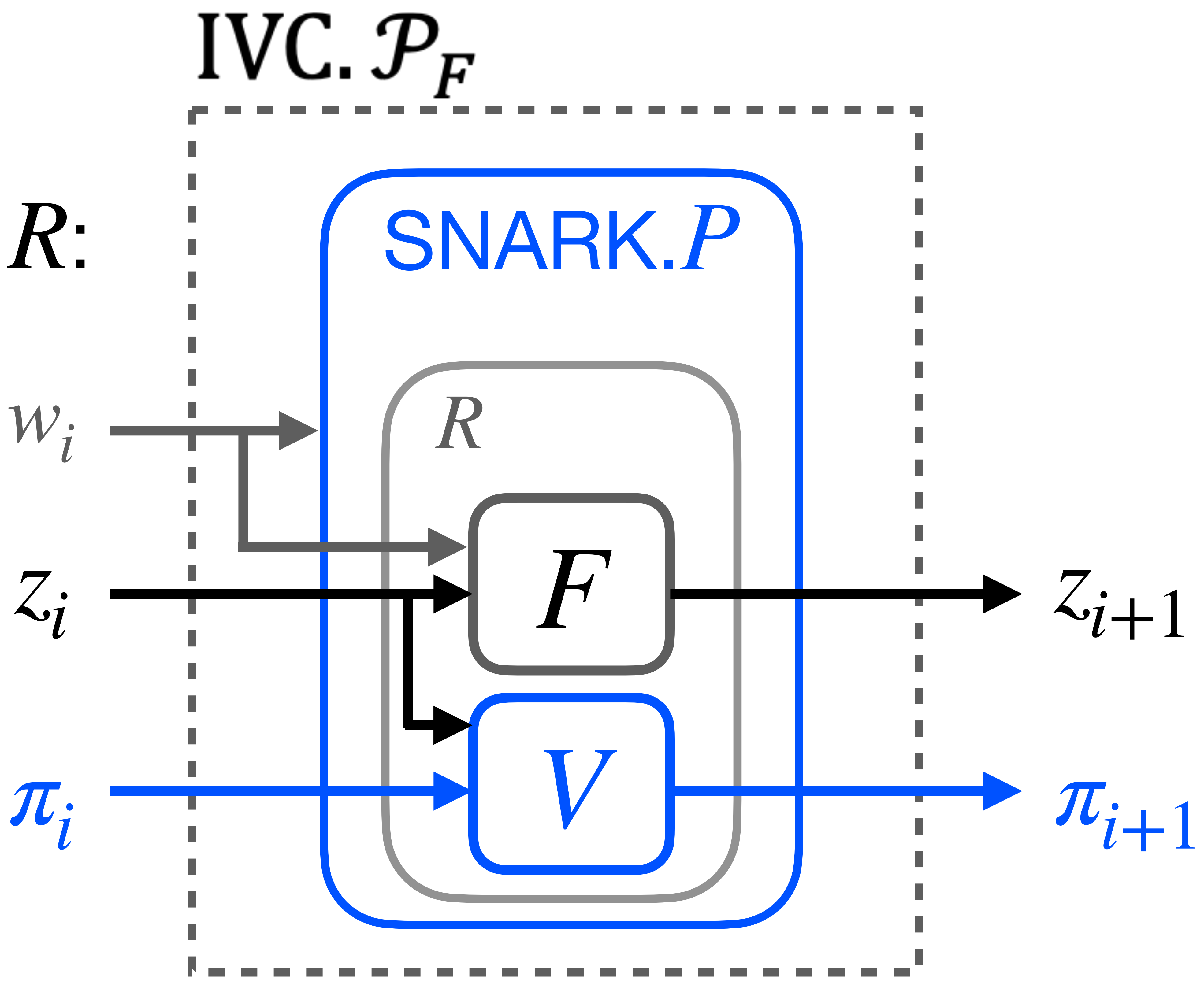
SNARK = Succinct Non-interactive ARgument of Knowledge

The IVC Prover…

Runs a SNARK P, proving the computation R:

For all i, $\exists w_i$ s.t.

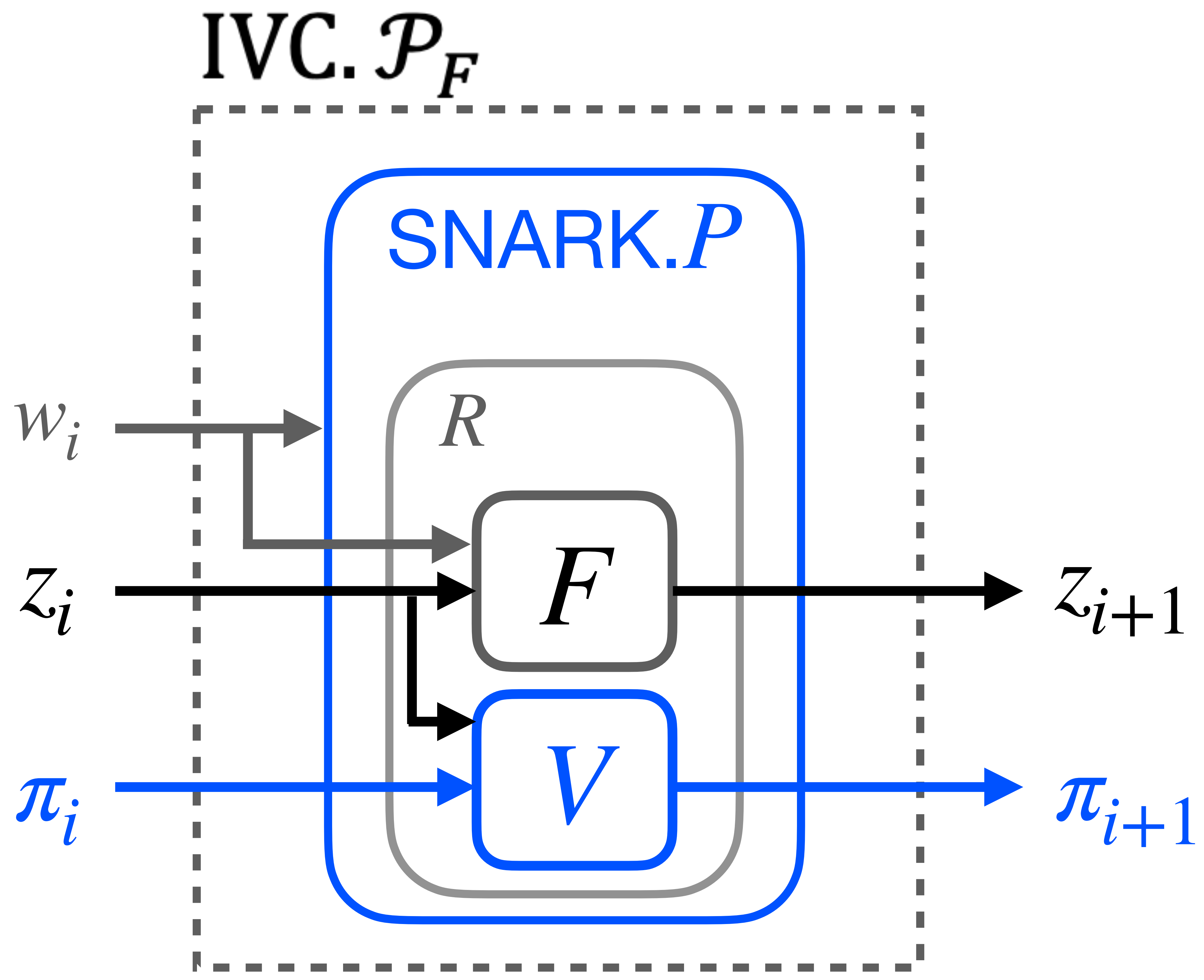
- $F(z_i, w_i) = z_{i+1}$ and
- SNARK $V(z_i, \pi_i) = 1$



Prior works: IVC instantiations from SNARKs

Approach 1: CRS + knowledge (extraction) assumptions

[Groth10; GennaroGPR13; BitanskyCIOP13; Ben-SassonCTV14; BitanskyCCGLRT14; Groth16; GrothKMMM18]



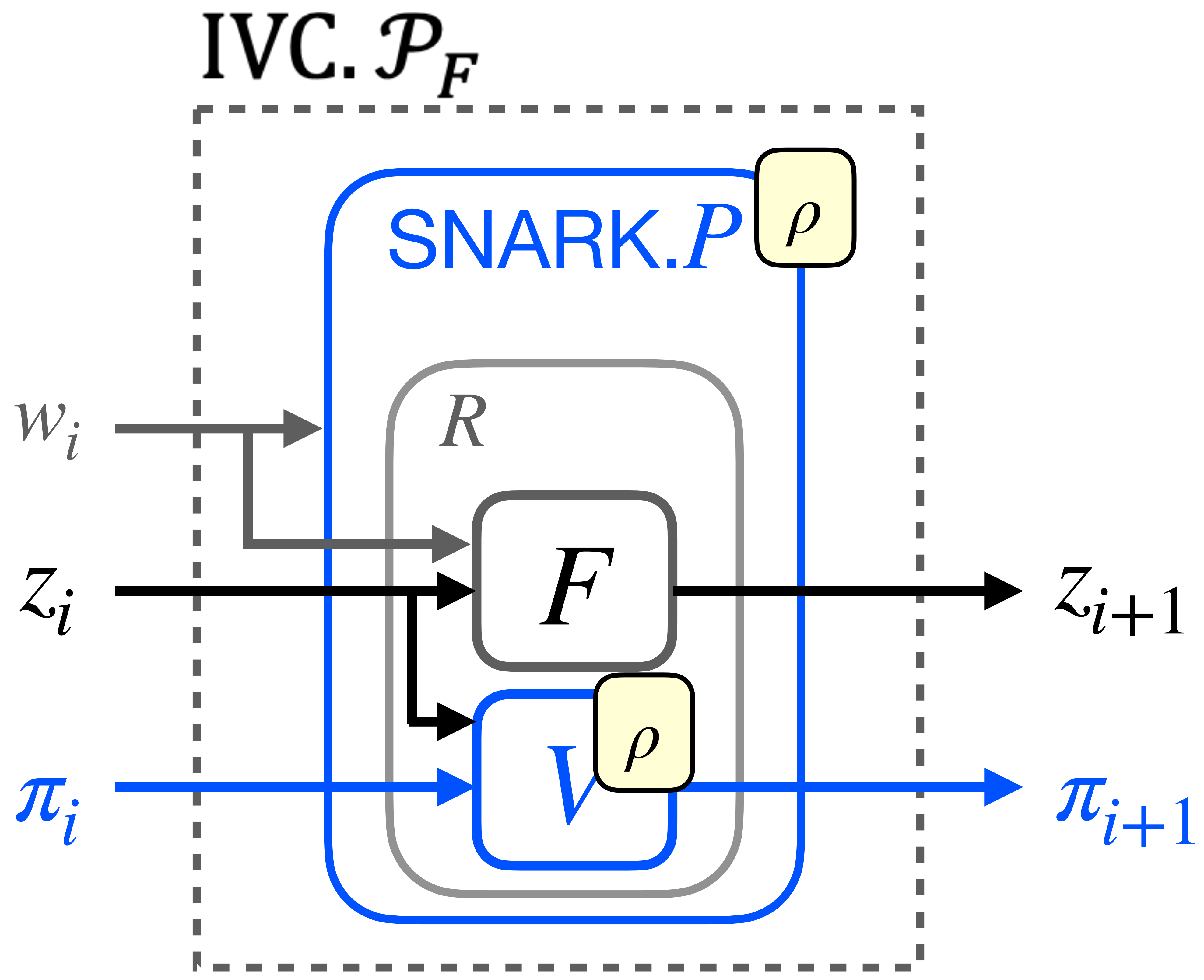
Prior works: IVC instantiations from SNARKs

Approach 1: CRS + knowledge (extraction) assumptions

[Groth10; GennaroGPR13; BitanskyCIOP13; Ben-SassonCTV14; BitanskyCCGLRT14; Groth16; GrothKMMM18]

Approach 2: SNARKs in ROM

[Micali00; Ben-SassonCS16; ChiesaOS20; ChiesaHMMVW20]



Prior works: IVC instantiations from SNARKs

Approach 1: CRS + knowledge (extraction) assumptions

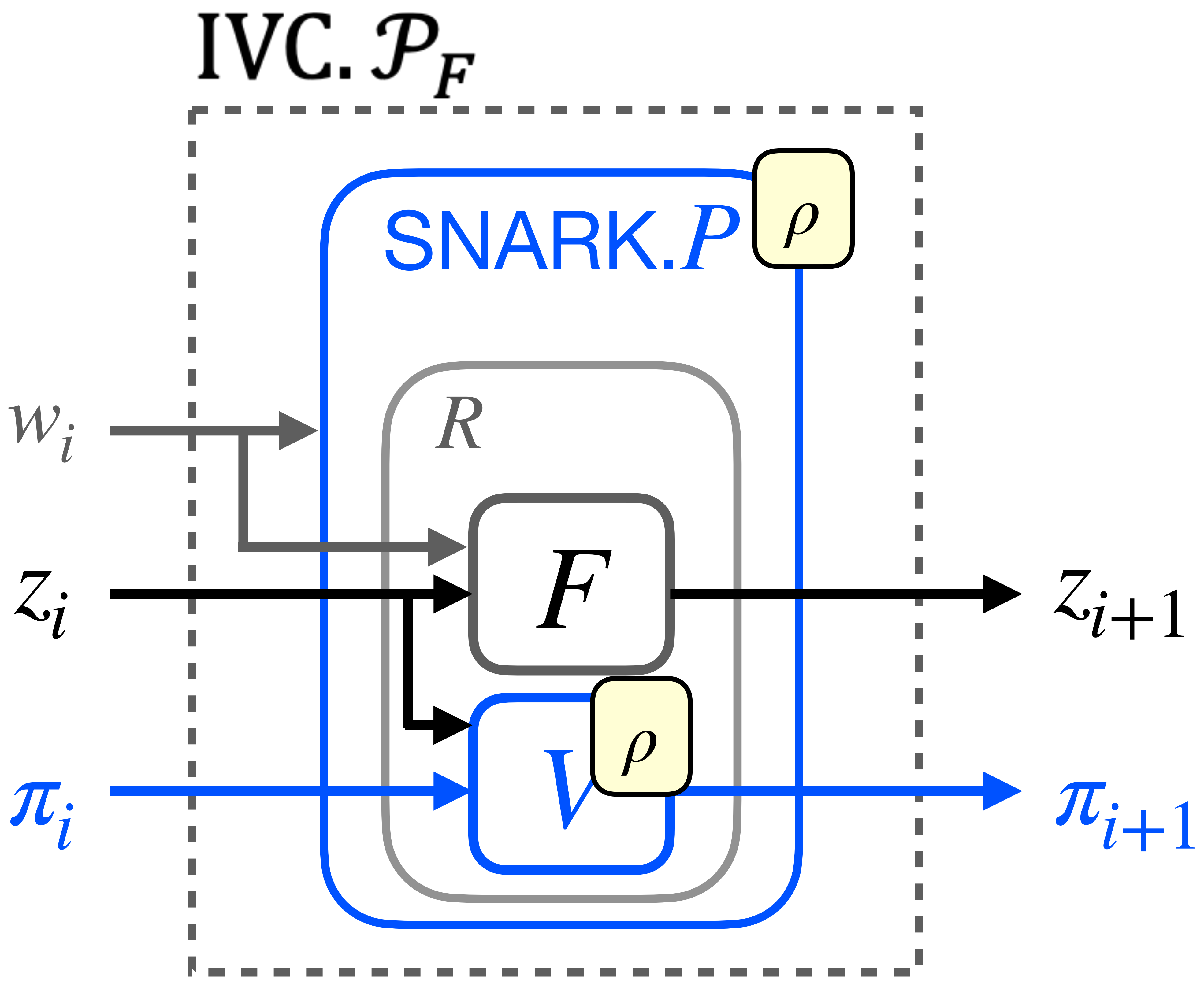
[Groth10; GennaroGPR13; BitanskyCIOP13; Ben-SassonCTV14; BitanskyCCGLRT14; Groth16; GrothKMMM18]

Approach 2: SNARKs in ROM

[Micali00; Ben-SassonCS16; ChiesaOS20; ChiesaHMMVW20]

Benefits:

- Transparent / universal setup
- Efficiency improvements



Prior works: IVC instantiations from SNARKs

Approach 1: CRS + knowledge (extraction) assumptions

[Groth10; GennaroGPR13; BitanskyCIOP13; Ben-SassonCTV14; BitanskyCCGLRT14; Groth16; GrothKMMM18]

Approach 2: SNARKs in ROM

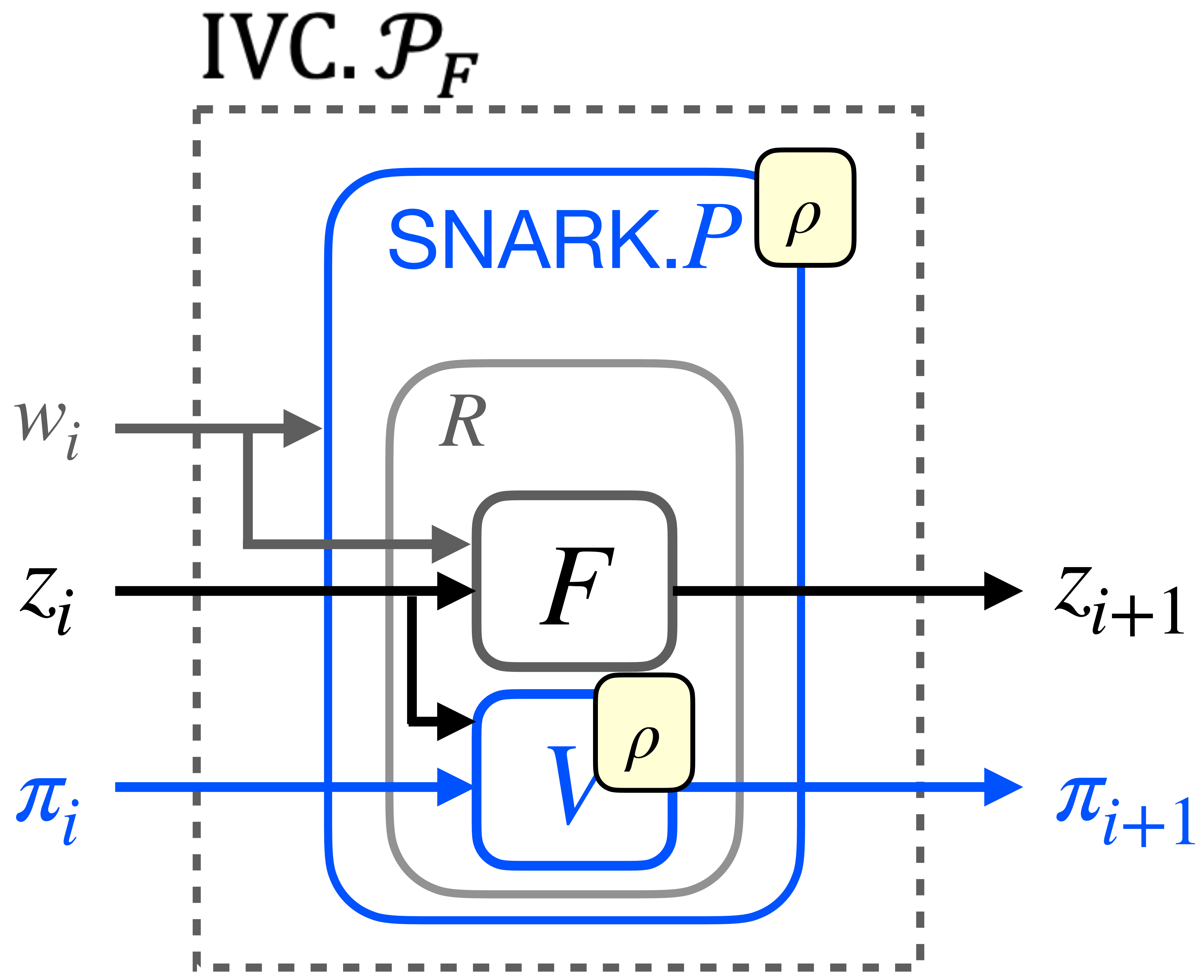
[Micali00; Ben-SassonCS16; ChiesaOS20; ChiesaHMMVW20]

Benefits:

- Transparent / universal setup
- Efficiency improvements

Issues:

- SNARK verifier makes oracle queries, but SNARK is for non-oracle computations.



Prior works: IVC instantiations from SNARKs

Approach 1: CRS + knowledge (extraction) assumptions

[Groth10; GennaroGPR13; BitanskyCIOP13; Ben-SassonCTV14; BitanskyCCGLRT14; Groth16; GrothKMMM18]

Approach 2: SNARKs in ROM

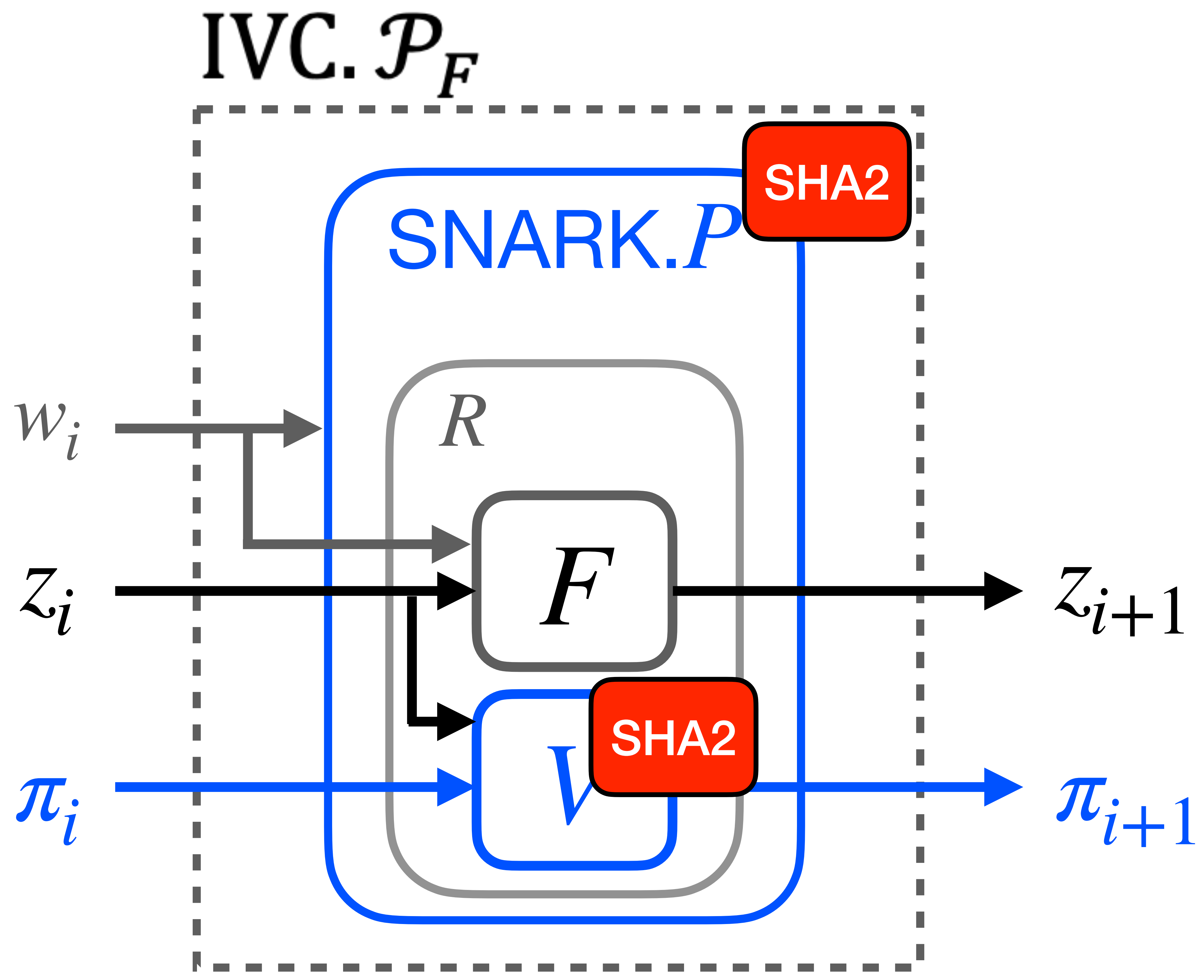
[Micali00; Ben-SassonCS16; ChiesaOS20; ChiesaHMMVW20]

Benefits:

- Transparent / universal setup
- Efficiency improvements

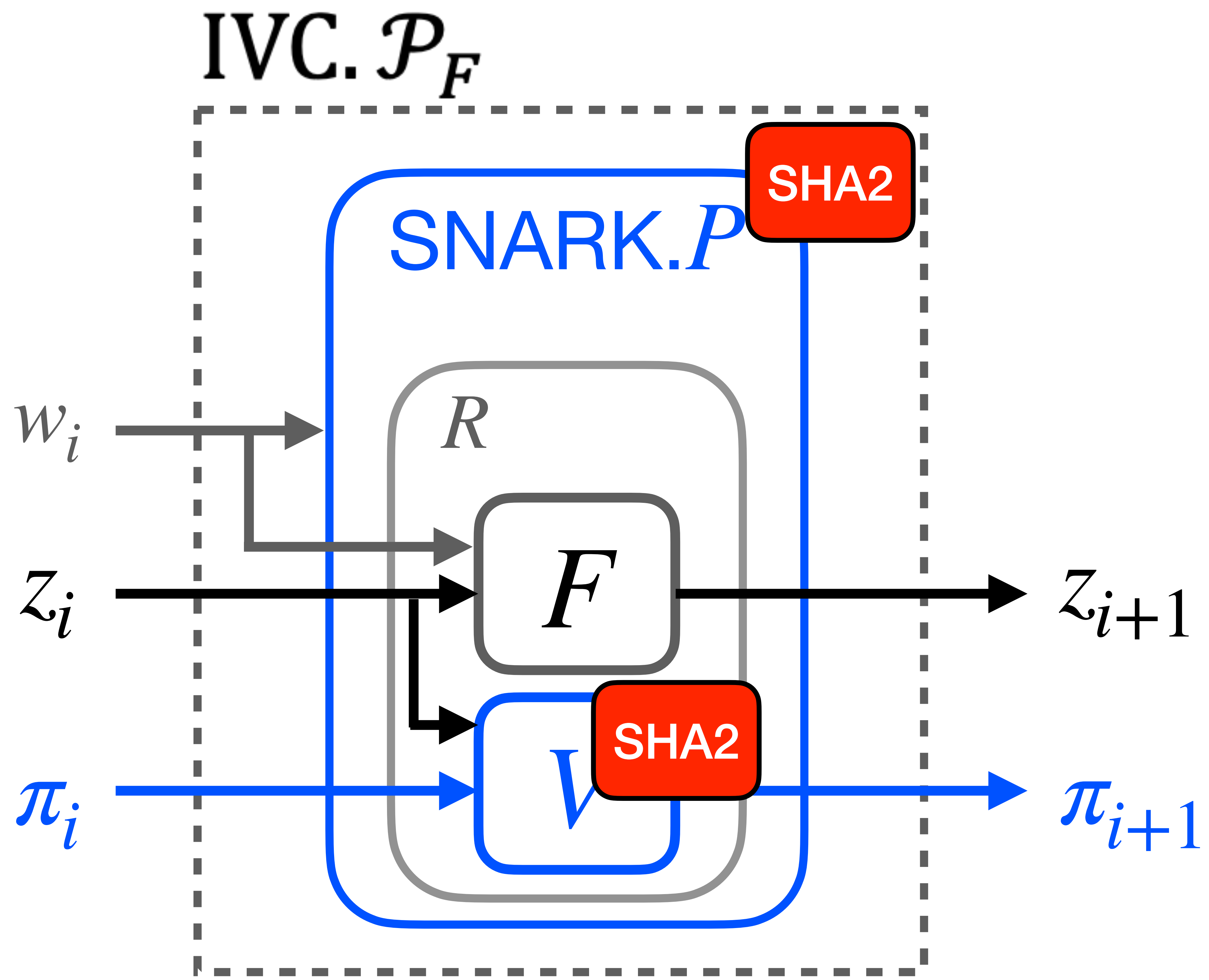
Issues:

- SNARK verifier makes oracle queries, but SNARK is for non-oracle computations.
- [ChiesaOS20; …] Heuristically instantiate ρ



Issues with heuristic RO instantiation

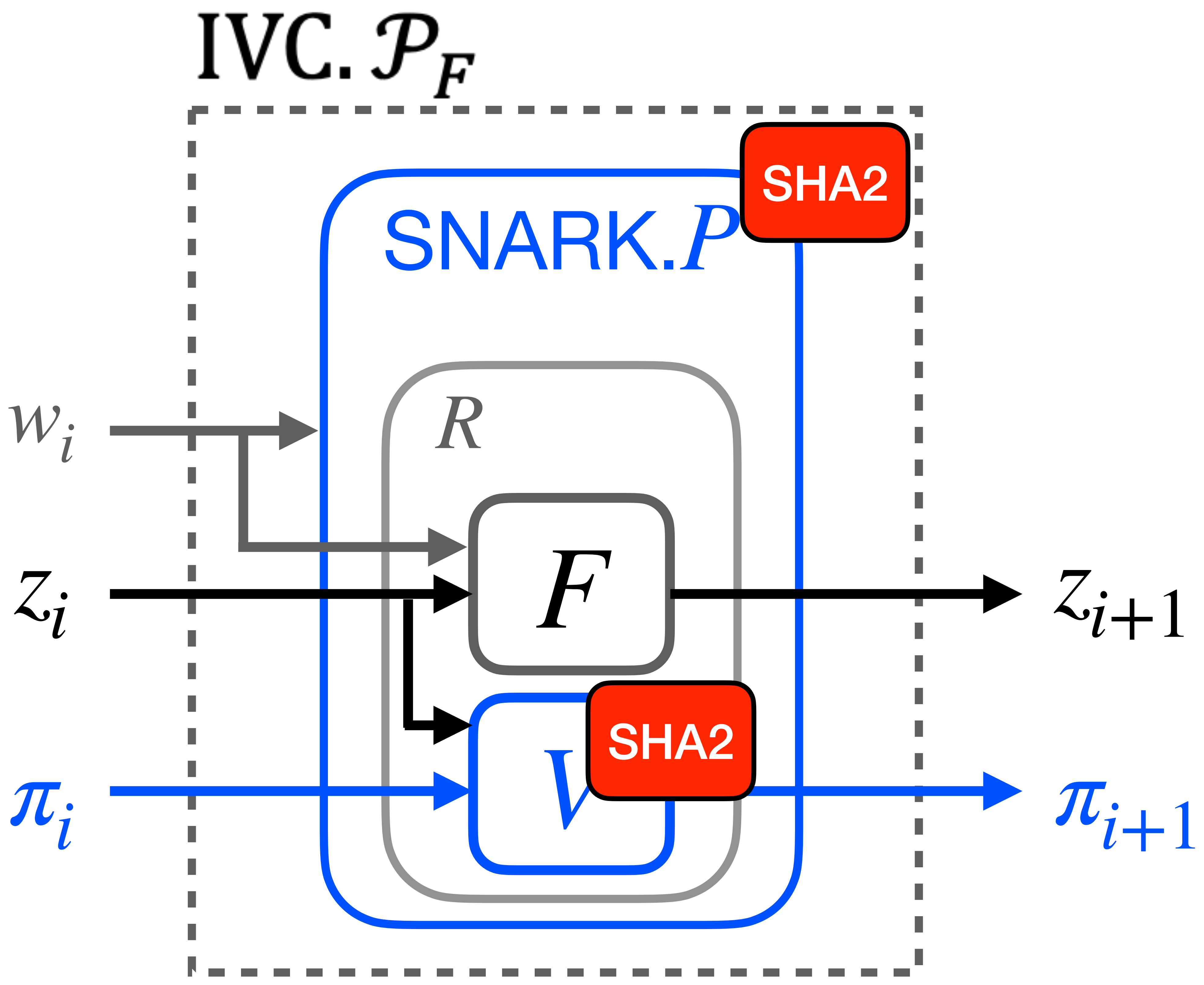
Theoretical:



Issues with heuristic RO instantiation

Theoretical:

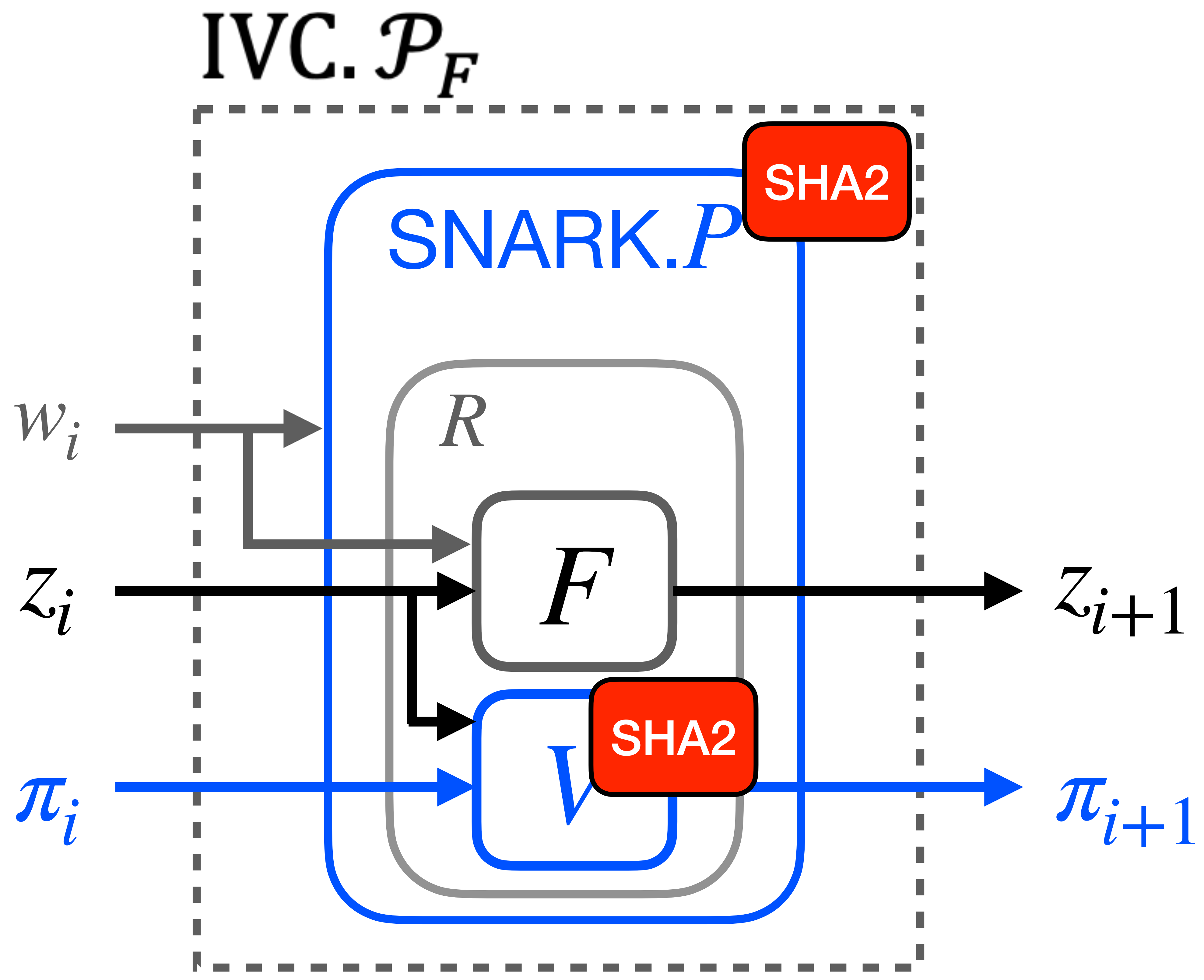
- Requires non-blackbox use of oracle; this breaks the RO abstraction.



Issues with heuristic RO instantiation

Theoretical:

- Requires non-blackbox use of oracle; this breaks the RO abstraction.
- Security flaws may be in the heuristic step [GoldwasserK03; CanettiGH04].

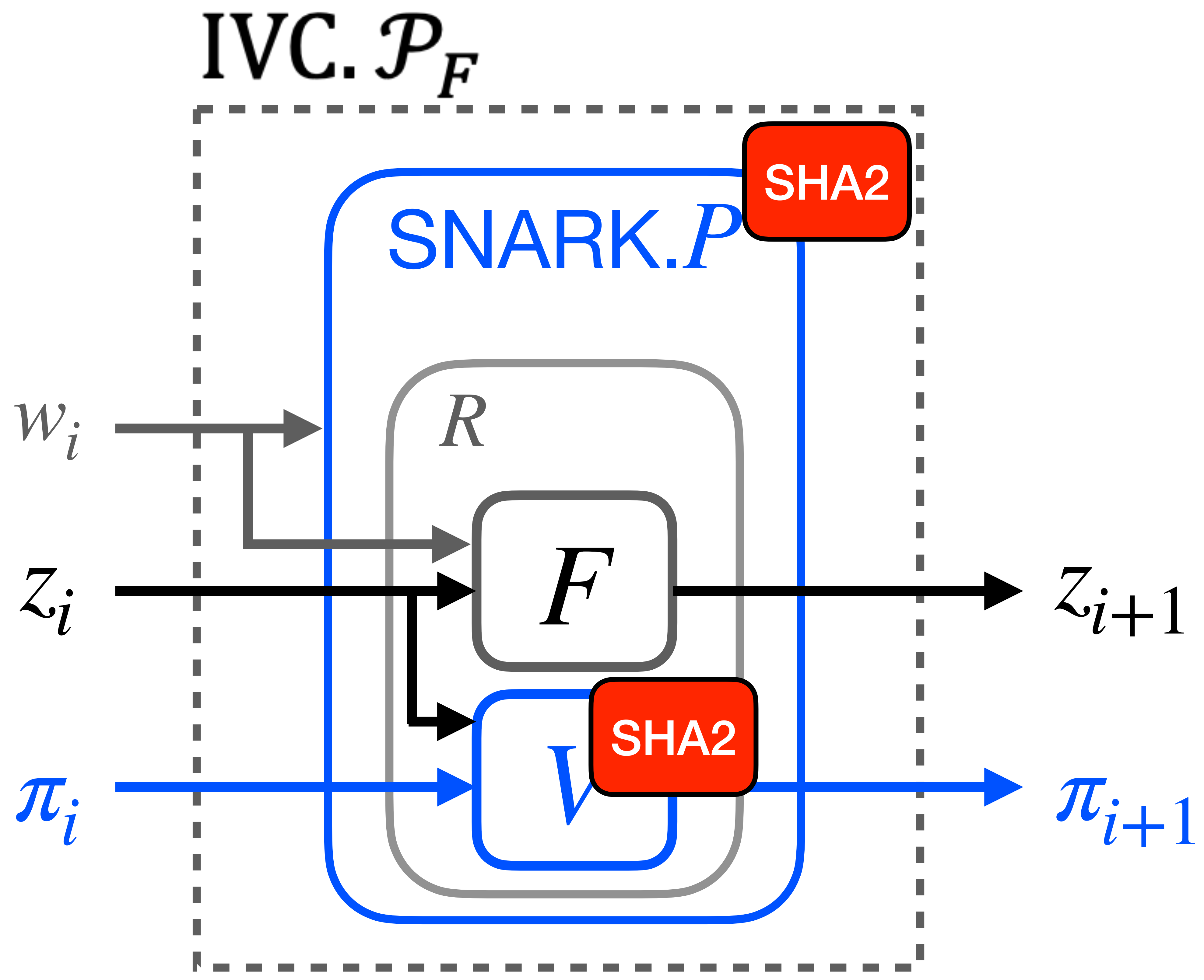


Issues with heuristic RO instantiation

Theoretical:

- Requires non-blackbox use of oracle; this breaks the RO abstraction.
- Security flaws may be in the heuristic step [GoldwasserK03; CanettiGH04].

Practical:



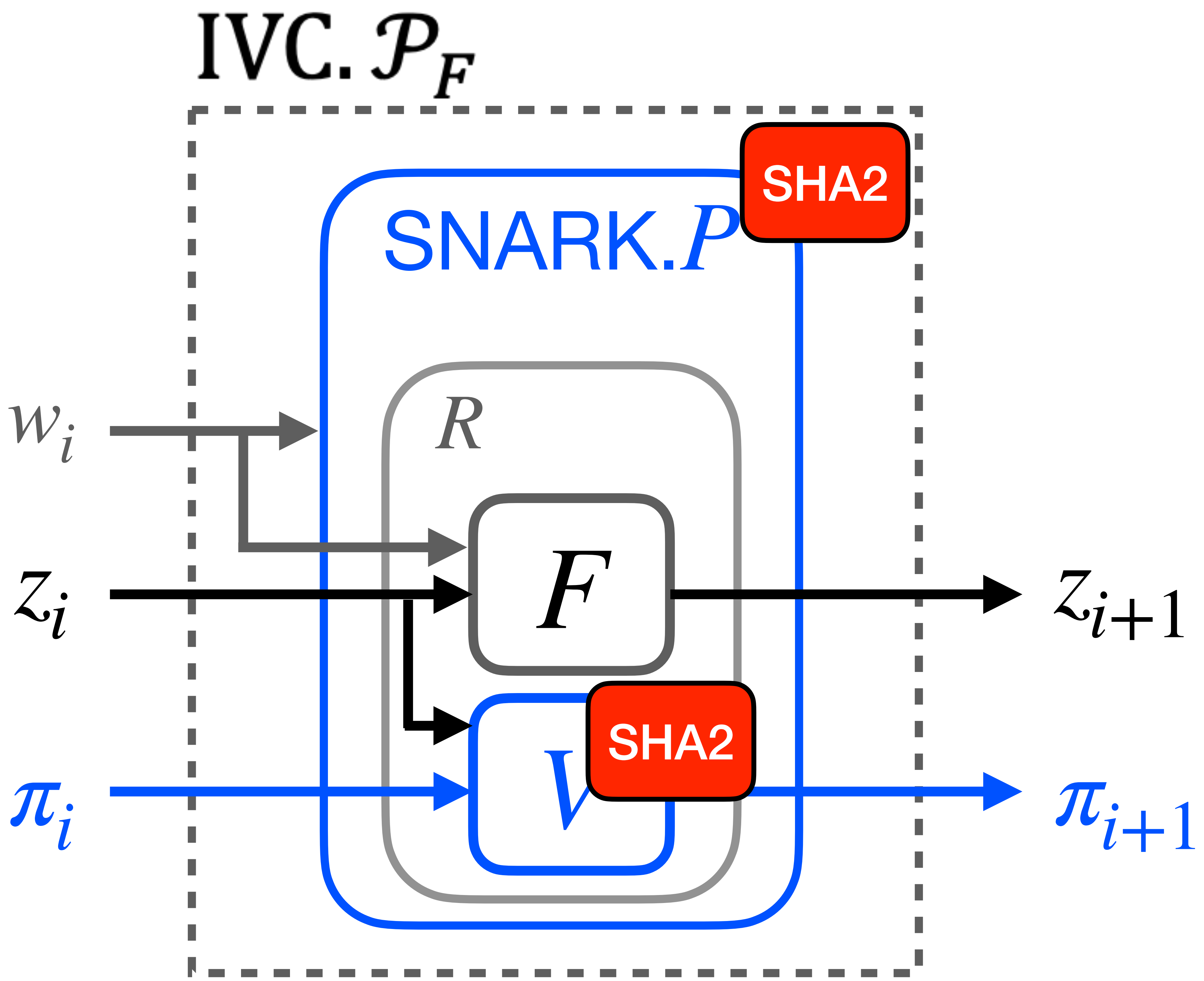
Issues with heuristic RO instantiation

Theoretical:

- Requires non-blackbox use of oracle; this breaks the RO abstraction.
- Security flaws may be in the heuristic step [GoldwasserK03; CanettiGH04].

Practical:

- No flexibility: Oracle must be instantiated as a circuit: can't use MPC, hardware token.



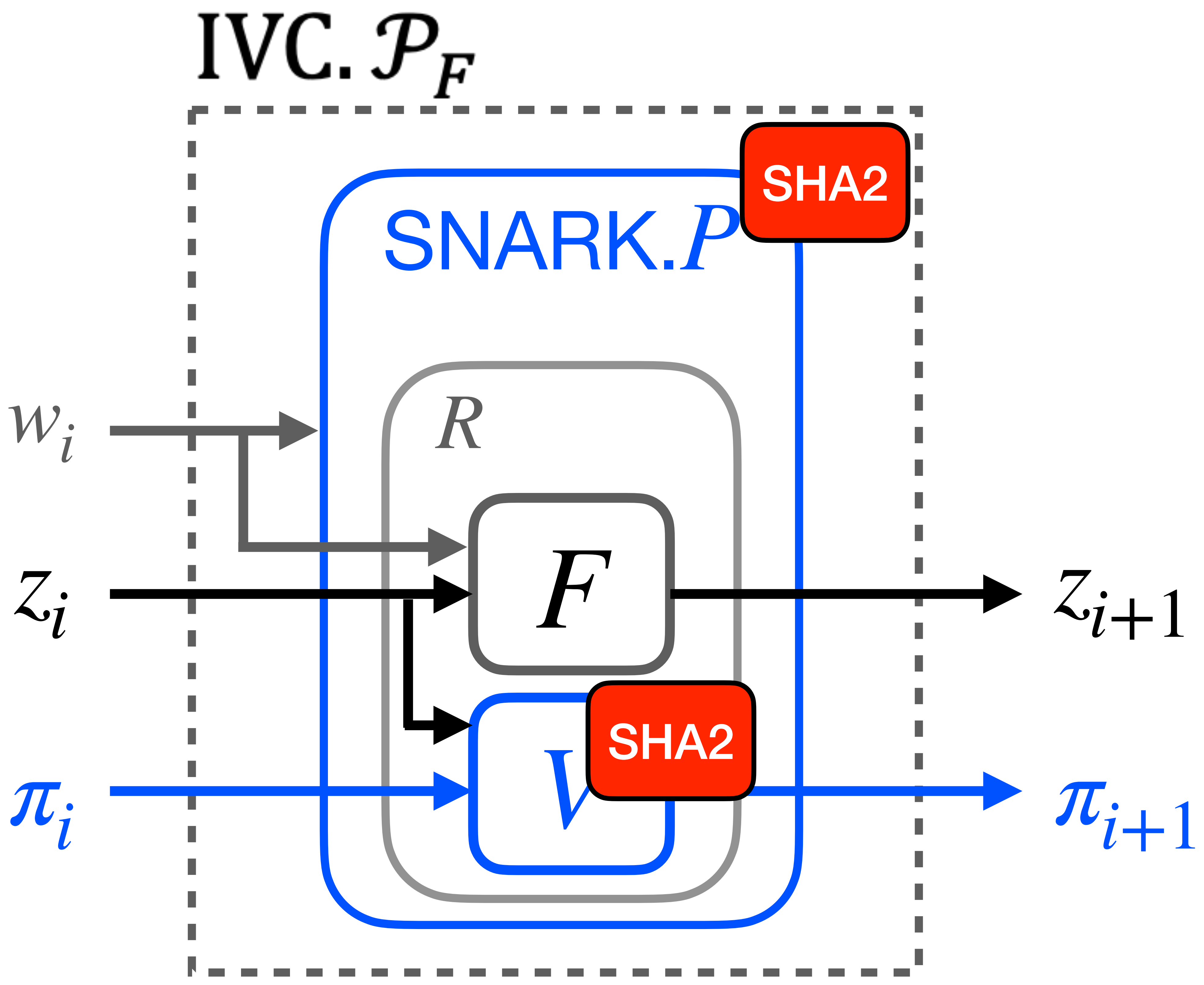
Issues with heuristic RO instantiation

Theoretical:

- Requires non-blackbox use of oracle; this breaks the RO abstraction.
- Security flaws may be in the heuristic step [GoldwasserK03; CanettiGH04].

Practical:

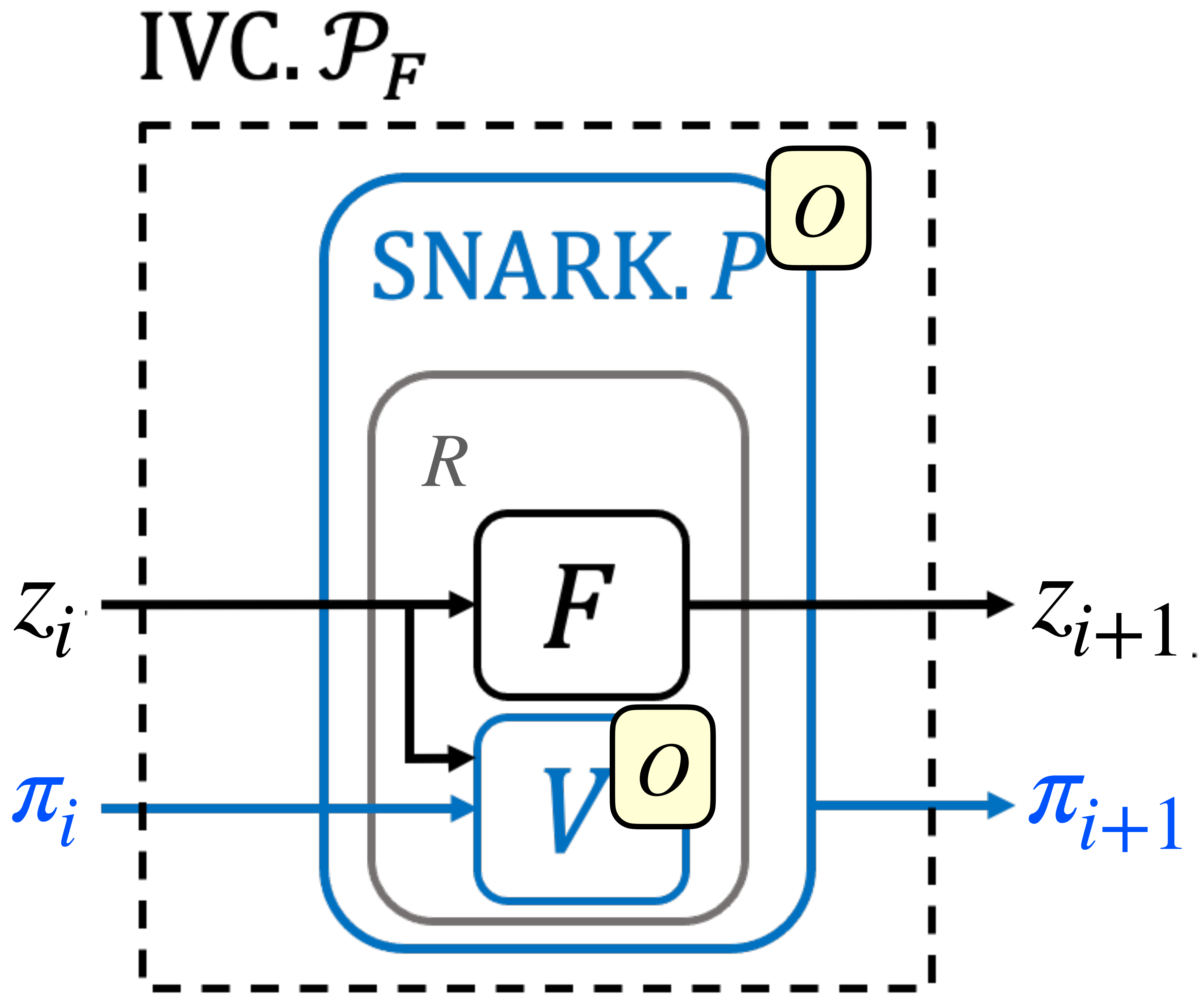
- No flexibility: Oracle must be instantiated as a circuit: can't use MPC, hardware token.
- Inefficient: SNARKs about SHA2, BLAKE are expensive!



Research question

Is there an oracle model O such that

1. there are SNARKs in the O model; and
2. the SNARK can prove statements about O?

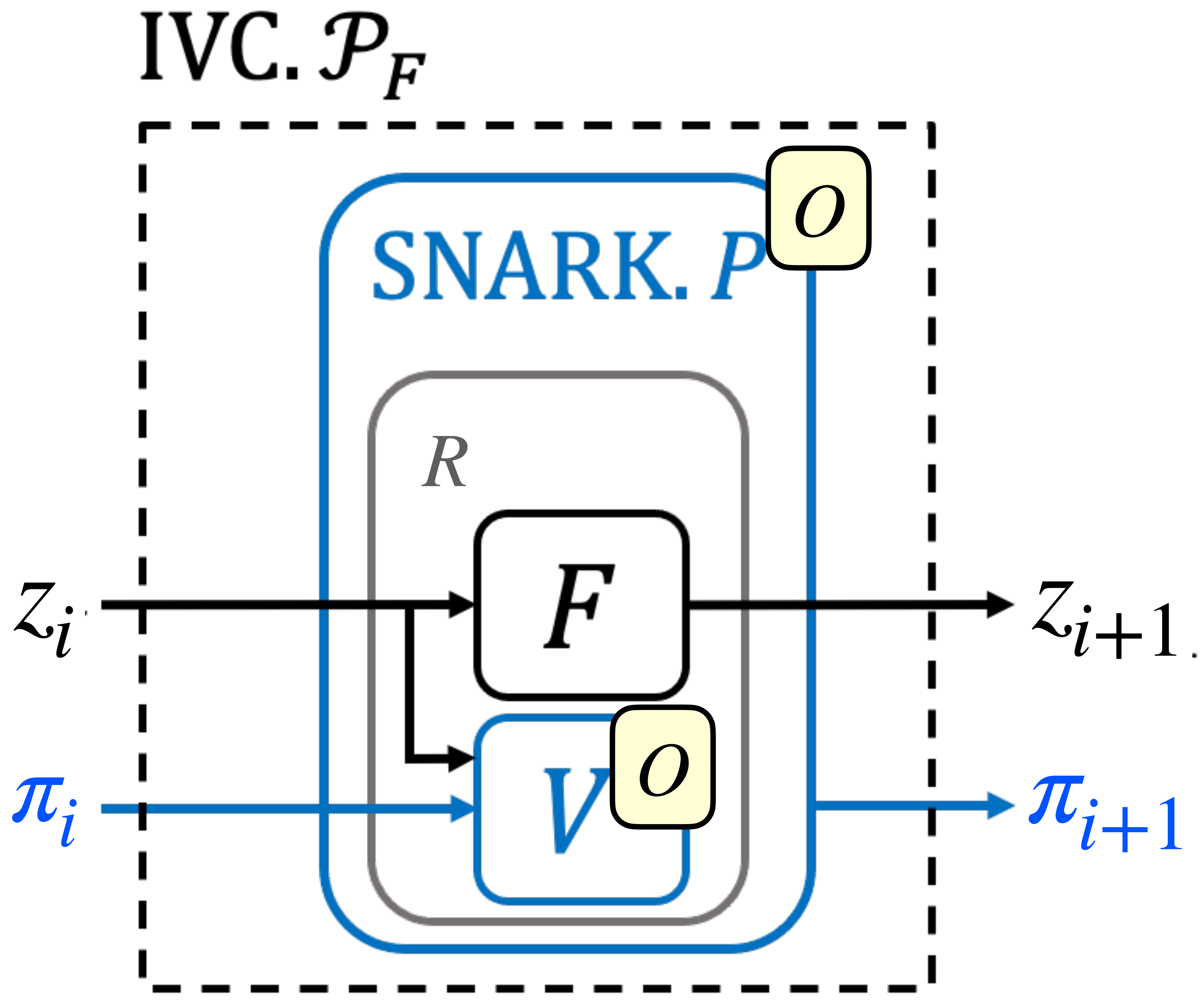


Research question

Is there an oracle model O such that

1. there are SNARKs in the O model; and
2. the SNARK can prove statements about O?

Having O means we can build IVC.



Research question

Is there an oracle model O such that

1. there are SNARKs in the O model; and
2. the SNARK can prove statements about O?

Impossible when O is the random oracle!

Our results

Define: low-degree random oracle (LDRO)

Our results

Define: low-degree random oracle (LDRO)

Correctness of

NP$^{\hat{\rho}}$ computation

SNARK in
LDROM for
LDROM
computations

Our results

Define: low-degree random oracle (LDRO)

Correctness of $\text{NP}^{\hat{\rho}}$ computation = Correctness of NP computation + Succinct verification of M's $\hat{\rho}$ queries

SNARK in LDROM for LDROM computations

Our results

Define: low-degree random oracle (LDRO)

Correctness of $NP^{\hat{\rho}}$ computation = Correctness of NP computation + Succinct verification of M's $\hat{\rho}$ queries

SNARK in LDROM for LDROM computations = SNARK in LDROM for non-oracle computations +

Our results

Define: low-degree random oracle (LDRO)

Correctness of $NP^{\hat{\rho}}$ computation = Correctness of NP computation + Succinct verification of M's $\hat{\rho}$ queries

SNARK in LDROM for LDROM computations = SNARK in LDROM for non-oracle computations + NI query reduction for LDROM queries

Uses ideas from [KalaiRaz08].

Our results

Define: low-degree random oracle (LDRO)

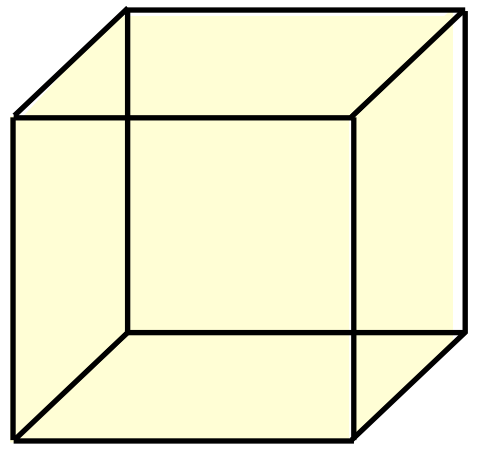
Correctness of $\mathrm{NP}^{\hat{\rho}}$ computation = Correctness of NP computation + Succinct verification of M's $\hat{\rho}$ queries

SNARK in LDROM for LDROM computations = **SNARK in LDROM for non-oracle computations** + **NI query reduction for LDROM queries**

Uses ideas from [KalaiRaz08].

Random oracle

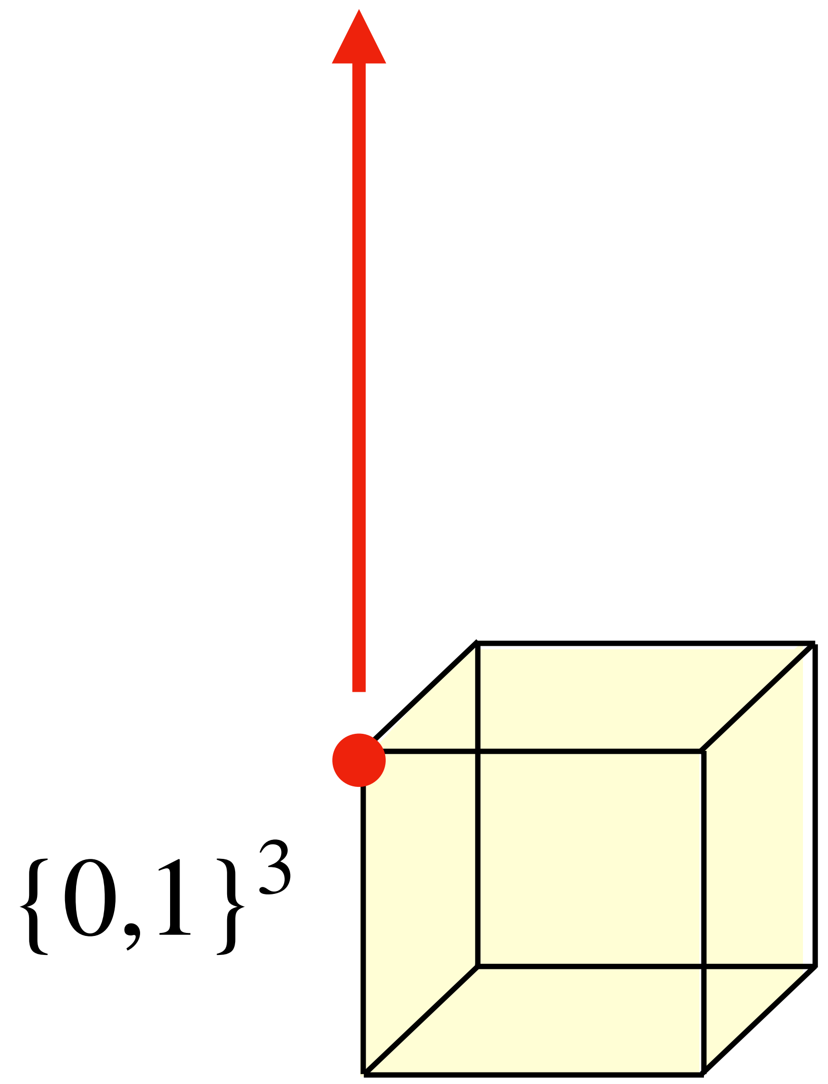
$\{0,1\}^3$



$\mathsf{RO} : \{0,1\}^m \to \mathbb{F}$

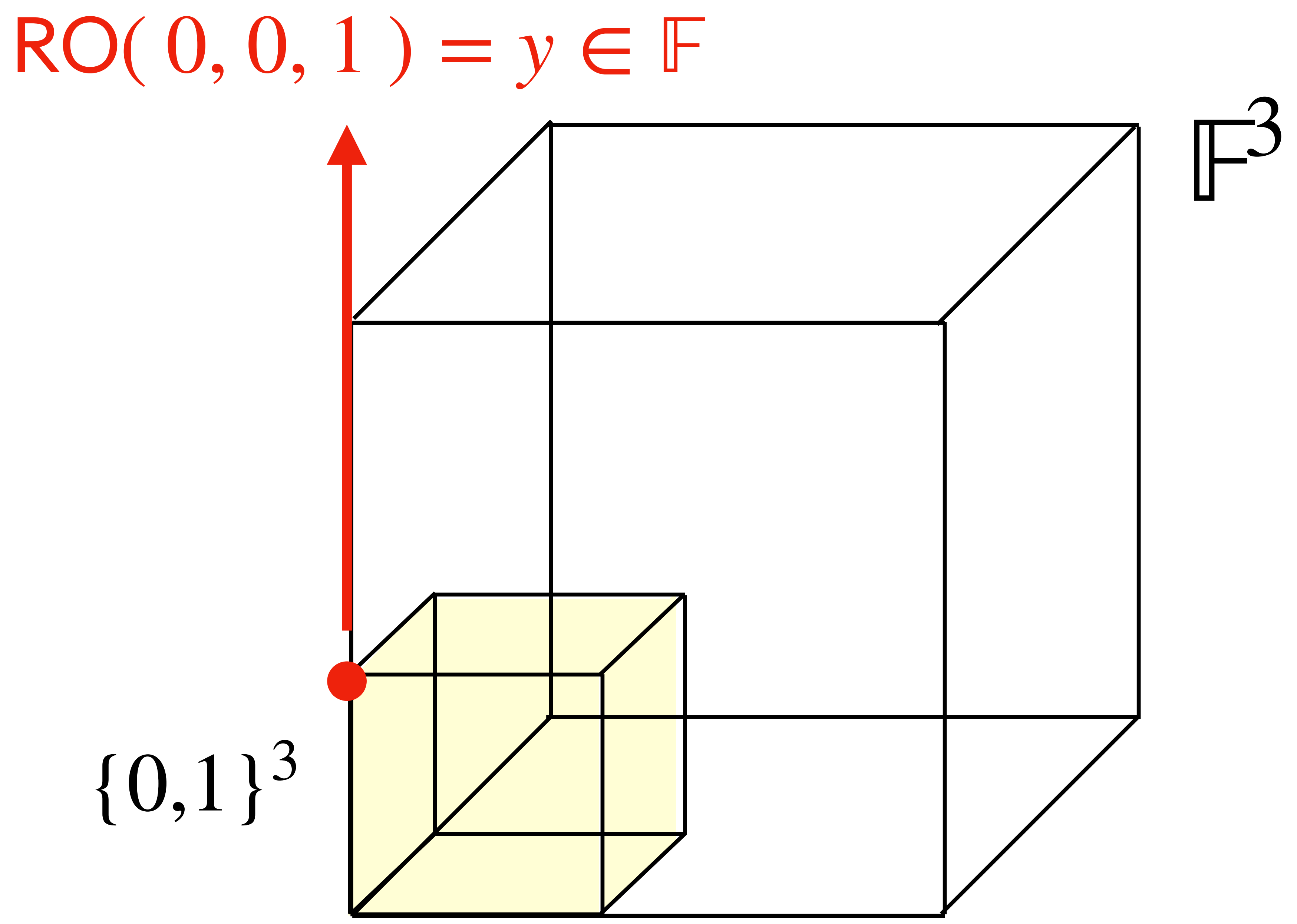
Random oracle

$\text{RO}(\,0, 0, 1\,) = y \in \mathbb{F}$



$\text{RO} : \{0,1\}^m \rightarrow \mathbb{F}$

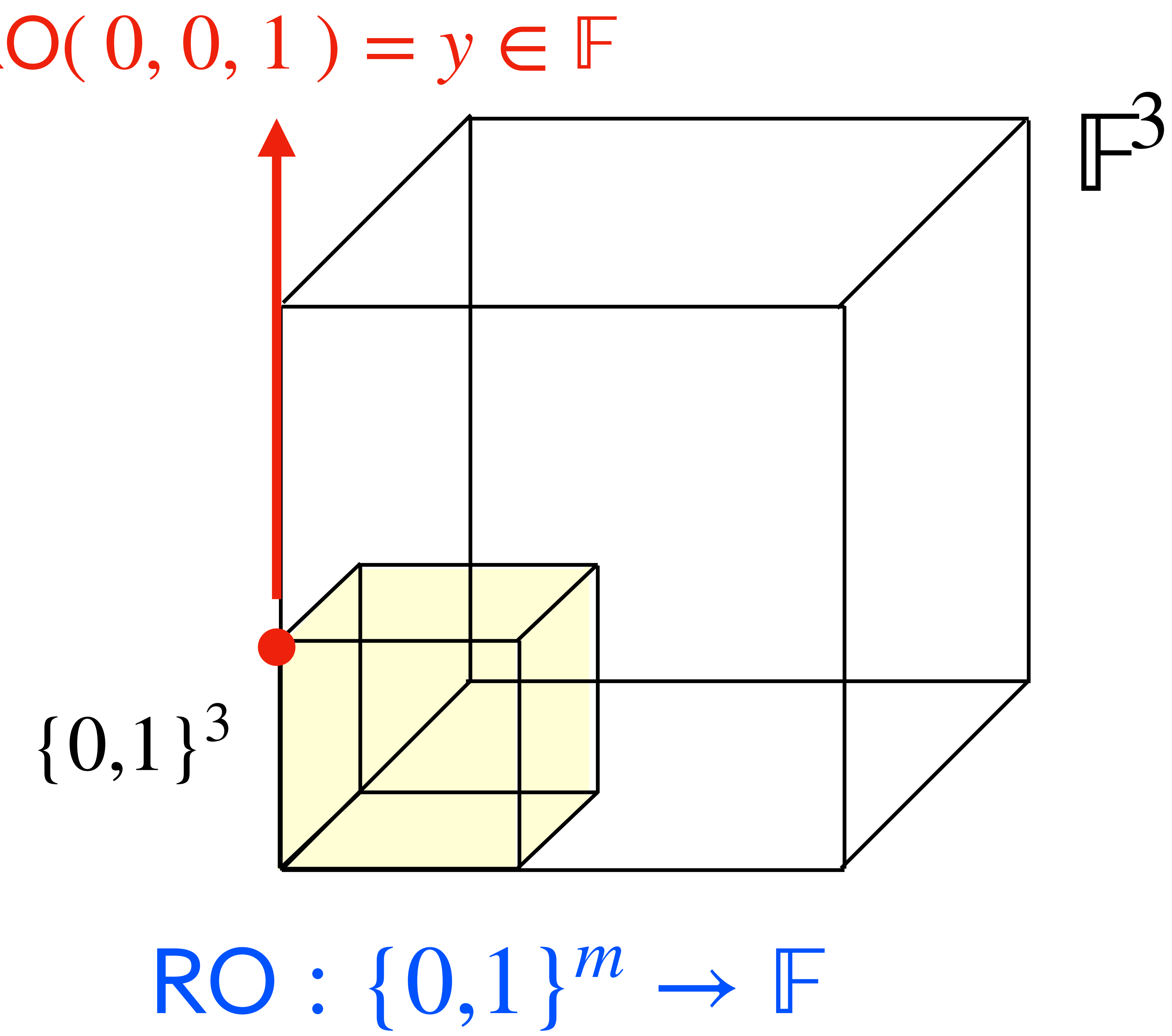
Low-degree random oracle ($\hat{\rho}$)



RO : $\{0,1\}^m \to \mathbb{F}$

Low-degree random oracle ($\hat{\rho}$)

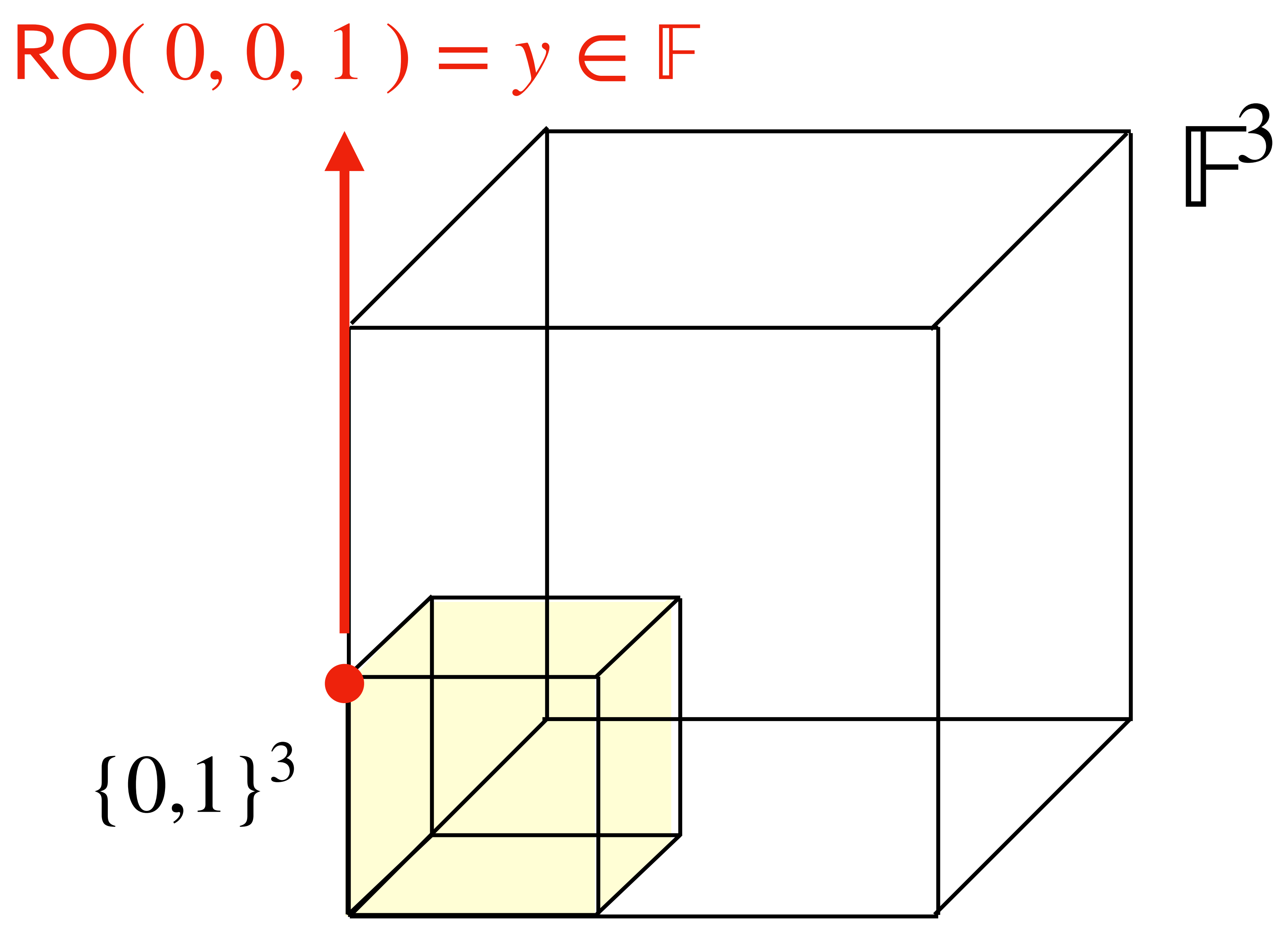
m-variate polynomials over $\mathbb{F}$,
individual degree $\leq d$,
evaluated over $\mathbb{F}^m$



Random $\hat{\rho} \in \mathbb{F}^{\leq d}[X_1, \ldots, X_m]$ s.t.:

Low-degree random oracle ($\hat{\rho}$)

m-variate polynomials over $\mathbb{F}$,
individual degree $\leq d$,
evaluated over $\mathbb{F}^m$

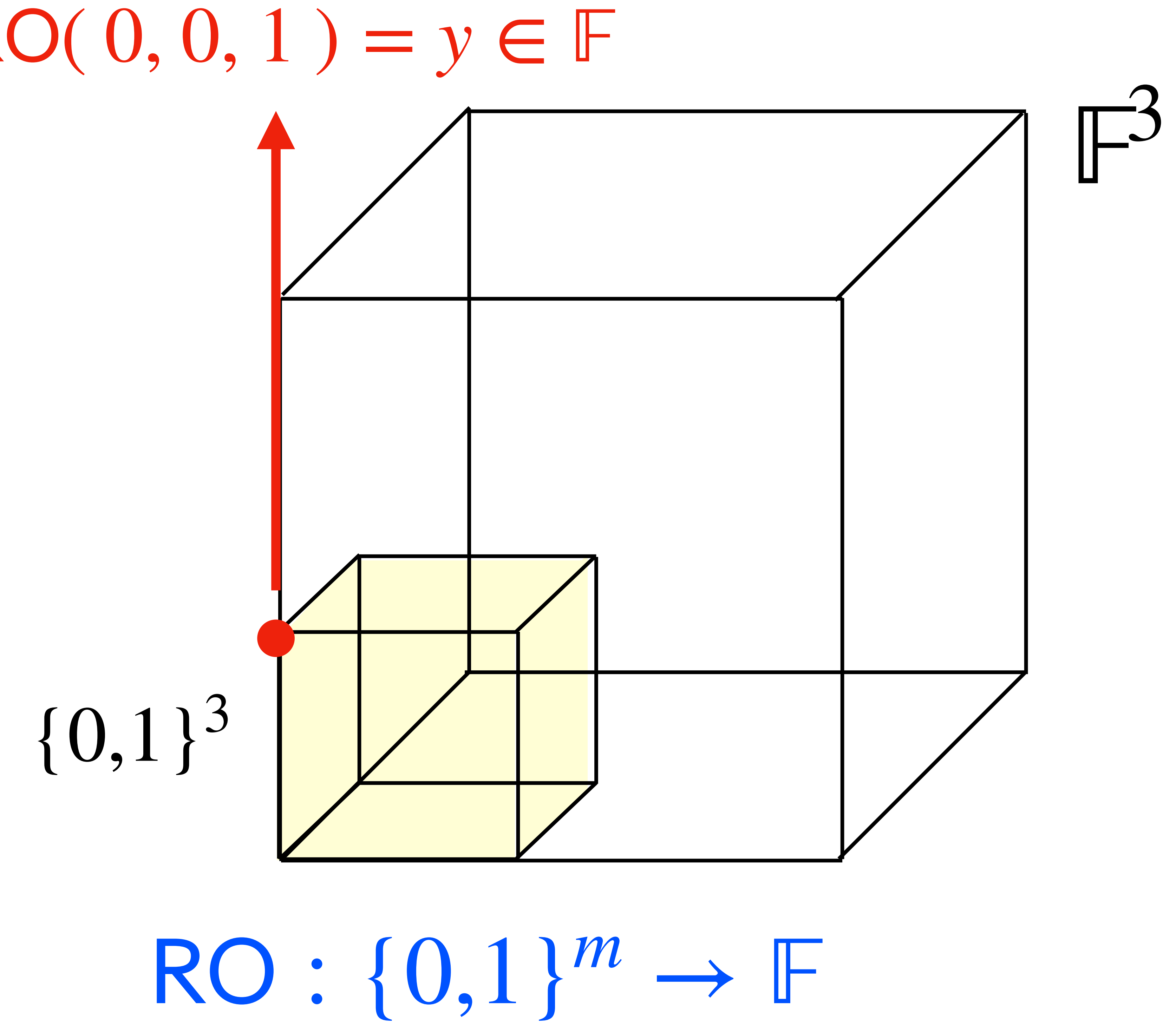


RO : $\{0,1\}^m \to \mathbb{F}$

Random $\hat{\rho} \in \mathbb{F}^{\leq d}[X_1, \ldots, X_m]$ s.t.:

- Points in Boolean hypercube agrees with random oracle

Low-degree random oracle ($\hat{\rho}$)

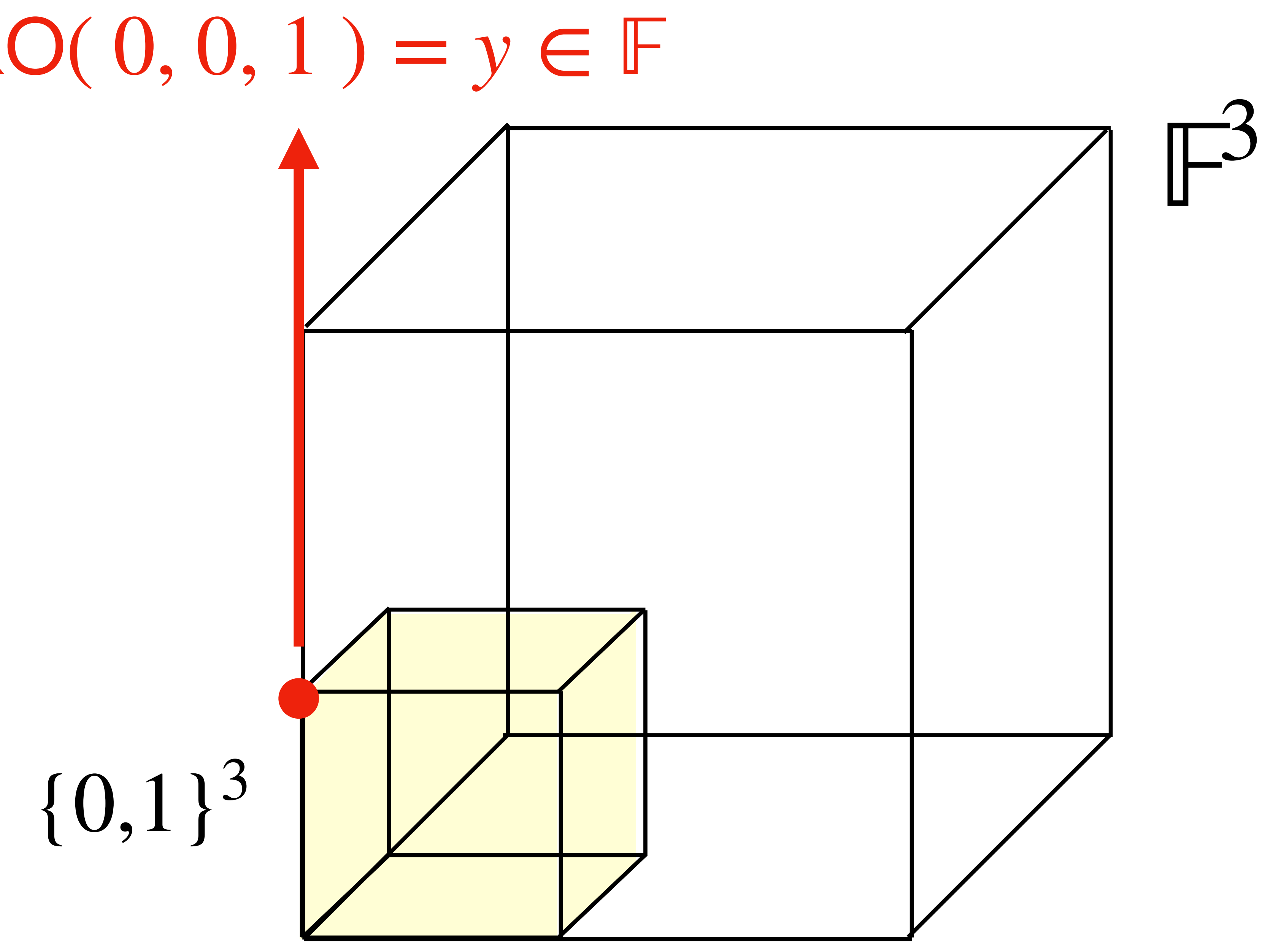


m-variate polynomials over $\mathbb{F}$,
individual degree $\leq d$,
evaluated over $\mathbb{F}^m$

Random $\hat{\rho} \in \mathbb{F}^{\leq d}[X_1, \ldots, X_m]$ s.t.:

- Points in Boolean hypercube agrees with random oracle
- Is low degree (e.g. $d = O(1)$).

Low-degree random oracle ($\hat{\rho}$)



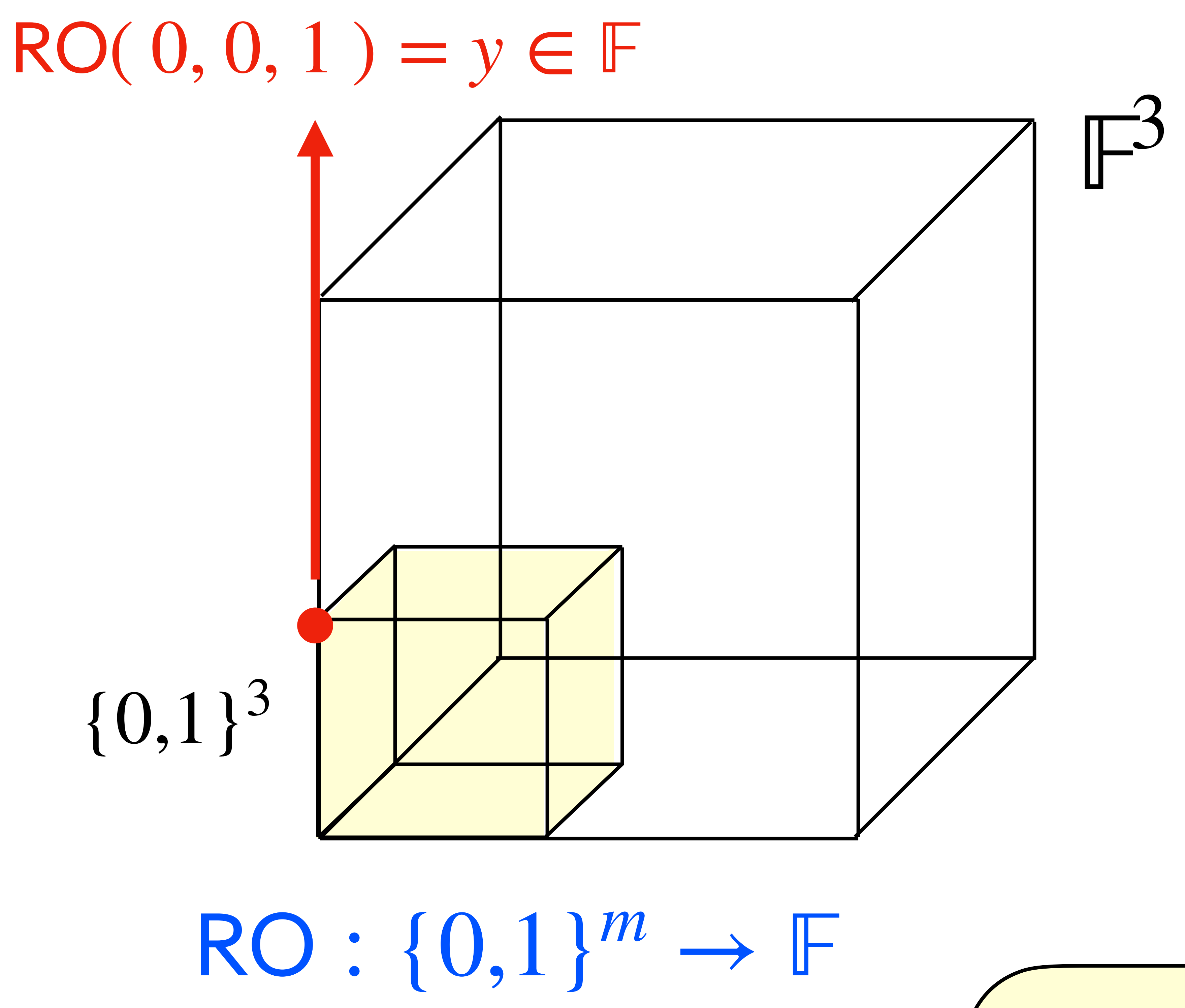
RO : $\{0,1\}^m \to \mathbb{F}$

m-variate polynomials over $\mathbb{F}$, individual degree $\leq d$, evaluated over $\mathbb{F}^m$

Random $\hat{\rho} \in \mathbb{F}^{\leq d}[X_1, \ldots, X_m]$ s.t.:

- Points in Boolean hypercube agrees with random oracle
- Is low degree (e.g. $d = O(1)$).
- Can query ANY point in $\mathbb{F}^m$

Low-degree random oracle ($\hat{\rho}$)



m-variate polynomials over $\mathbb{F}$, individual degree $\leq d$, evaluated over $\mathbb{F}^m$

Random $\hat{\rho} \in \mathbb{F}^{\leq d}[X_1, \ldots, X_m]$ s.t.:

- Points in Boolean hypercube agrees with random oracle
- Is low degree (e.g. $d = O(1)$).
- Can query ANY point in $\mathbb{F}^m$

Is $\hat{\rho}$ **simulatable** (can do lazily sampling) and **programmable**?

Lazy sampling of LDRO

Lemma: There is perfect, stateful simulation of LDROs.

Lazy sampling of LDRO

Lemma: There is perfect, stateful simulation of LDROs.

Lazy sampling strategy:

Given a query x, check if $y = \hat{\rho}(x)$ is already determined.

Lazy sampling of LDRO

Lemma: There is perfect, stateful simulation of LDROs.

Lazy sampling strategy:

Given a query x, check if $y = \hat{\rho}(x)$ is already determined.

Succinct constraint detection algorithm exists for low-degree polynomials [Ben-SassonCFGRS17]

Lazy sampling of LDRO

Lemma: There is perfect, stateful simulation of LDROs.

Lazy sampling strategy:

Given a query x, check if $y = \hat{\rho}(x)$ is already determined.

- If yes, use determined y.

Succinct constraint detection algorithm exists for low-degree polynomials [Ben-SassonCFGRS17]

Lazy sampling of LDRO

Lemma: There is perfect, stateful simulation of LDROs.

Lazy sampling strategy:

Given a query x, check if $y = \hat{\rho}(x)$ is already determined.

- If yes, use determined y.
- If no, sample $y \leftarrow_R \mathbb{F}$.

Succinct constraint detection algorithm exists for low-degree polynomials [Ben-SassonCFGRS17]

How to (heuristically) instantiate the LDRO?

How to (heuristically) instantiate the LDRO?

1. Since LDRO has stateful simulation, use trusted party or MPC protocol.

How to (heuristically) instantiate the LDRO?

1. Since LDRO has stateful simulation, use trusted party or MPC protocol.

2. Obfuscate (or embed in hardware token)

$$P(x_1, \ldots, x_m) = \sum_{\overrightarrow{a} \in [d]^m} F(\overrightarrow{a}) \cdot x_1^{a_1} \cdots x_m^{a_m}$$

where F is the structured PRF by [BenabbasGennaroVahlis11].

How to (heuristically) instantiate the LDRO?

1. Since LDRO has stateful simulation, use trusted party or MPC protocol.

2. Obfuscate (or embed in hardware token)

$$P(x_1, \ldots, x_m) = \sum_{\overrightarrow{a} \in [d]^m} F(\overrightarrow{a}) \cdot x_1^{a_1} \cdots x_m^{a_m}$$

   where F is the structured PRF by [BenabbasGennaroVahlis11].

3. **Future work:** arithmetize an existing “strong” hash function?

How do we efficiently verify LDRO queries?

How do we efficiently verify LDRO queries?

Step 1: Recall [KalaiRaz08]'s *interactive* query reduction protocol

How do we efficiently verify LDRO queries?

Step 1: Recall [KalaiRaz08]'s *interactive* query reduction protocol

Step 2: Make [KalaiRaz08] SNARK-friendly

What is query reduction?

Goal: verify polynomial queries

$$\{(x_1, y_1), \ldots, (x_n, y_n)\} \in \mathbb{F}^m \times \mathbb{F}$$

What is query reduction?

Goal: verify polynomial queries

$$\{(x_1, y_1), \ldots, (x_n, y_n)\} \in \mathbb{F}^m \times \mathbb{F}$$

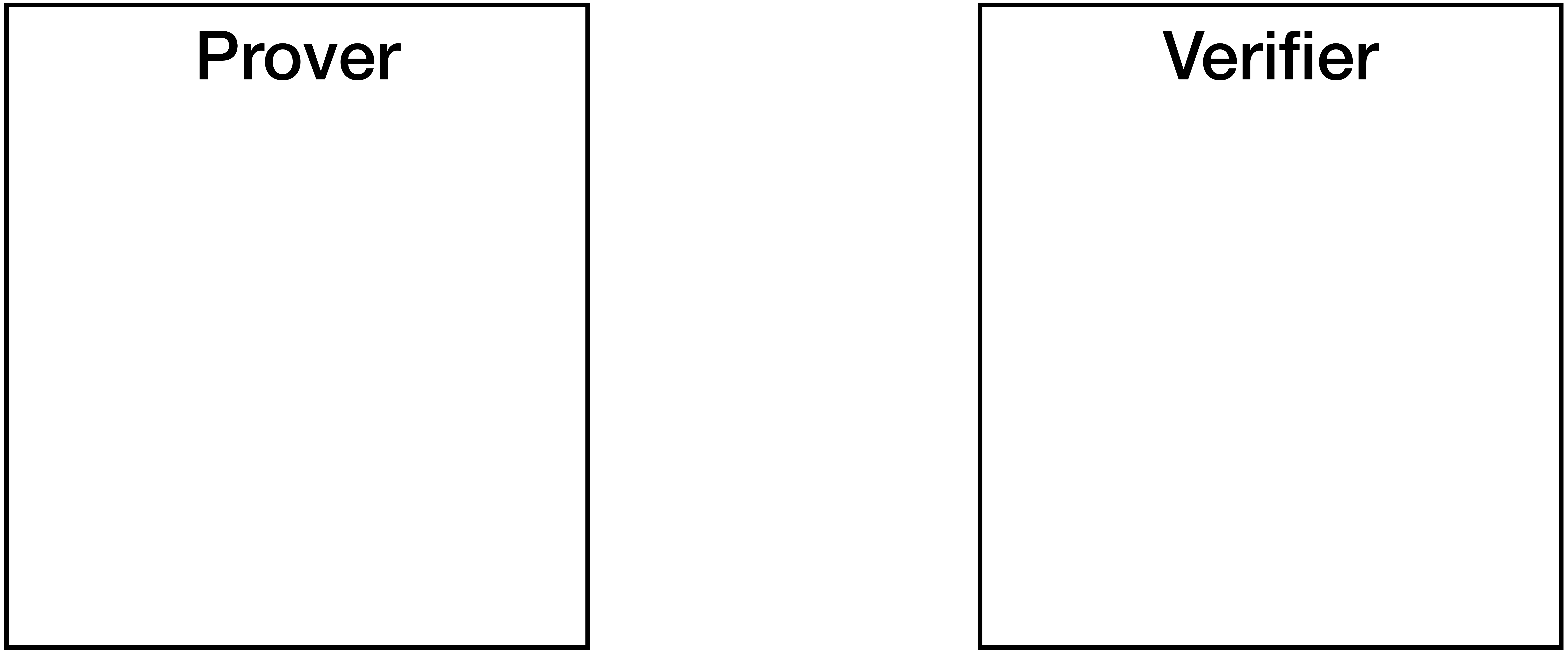
Idea: Verifier has help from a prover

- [KalaiRaz08] gives an IP for this task
- Only requires 1 query to $\hat{\rho}$

[KalaiRaz08] Interactive query reduction protocol

Input: $\{(x_1, y_1), \ldots, (x_n, y_n)\} \in \mathbb{F}^m \times \mathbb{F}$

Goal: Check $\hat{\rho}(x_i) = y_i \; \forall i \in [n]$ with 1 verifier query

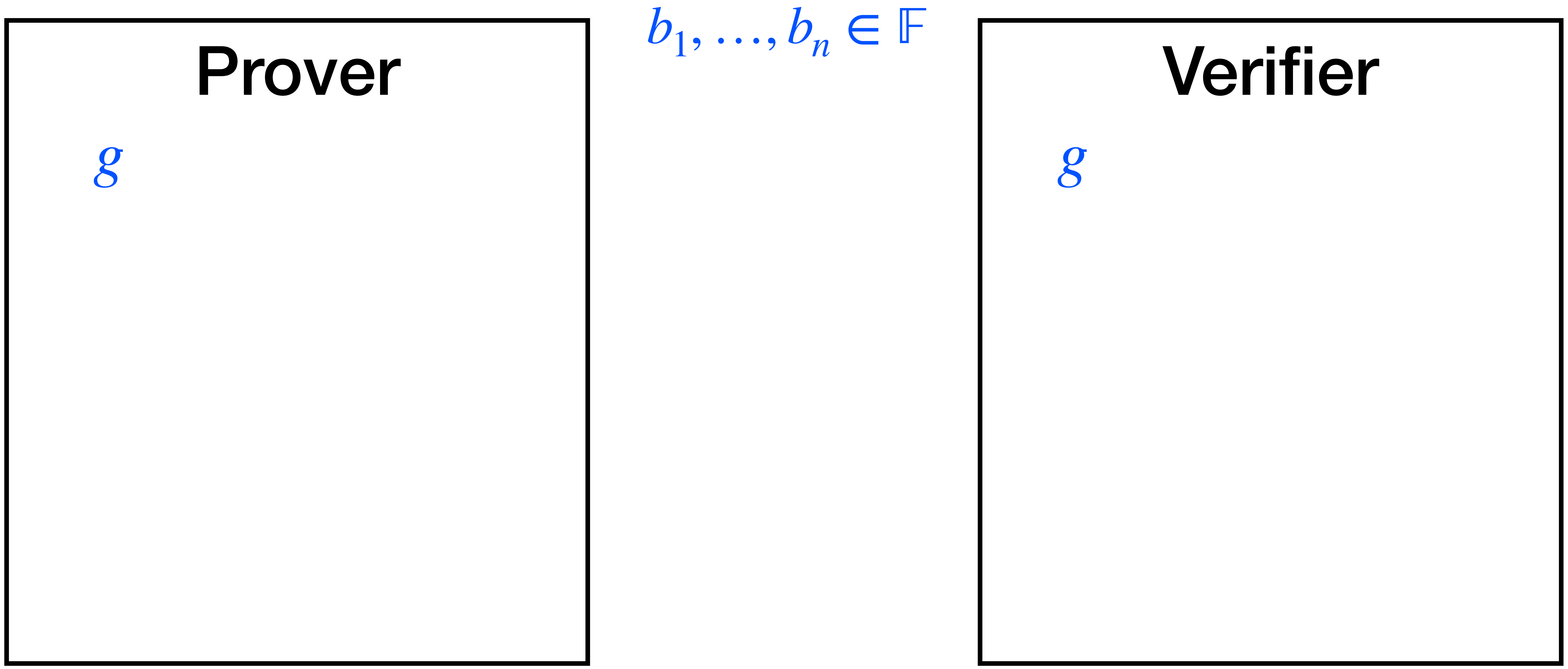


[KalaiRaz08] Interactive query reduction protocol

Input: $\{(x_1, y_1), \ldots, (x_n, y_n)\} \in \mathbb{F}^m \times \mathbb{F}$

Goal: Check $\hat{\rho}(x_i) = y_i \ \forall i \in [n]$ with 1 verifier query

Compute a curve g s.t. $g(b_i) = x_i \ \forall i \in [n]$.

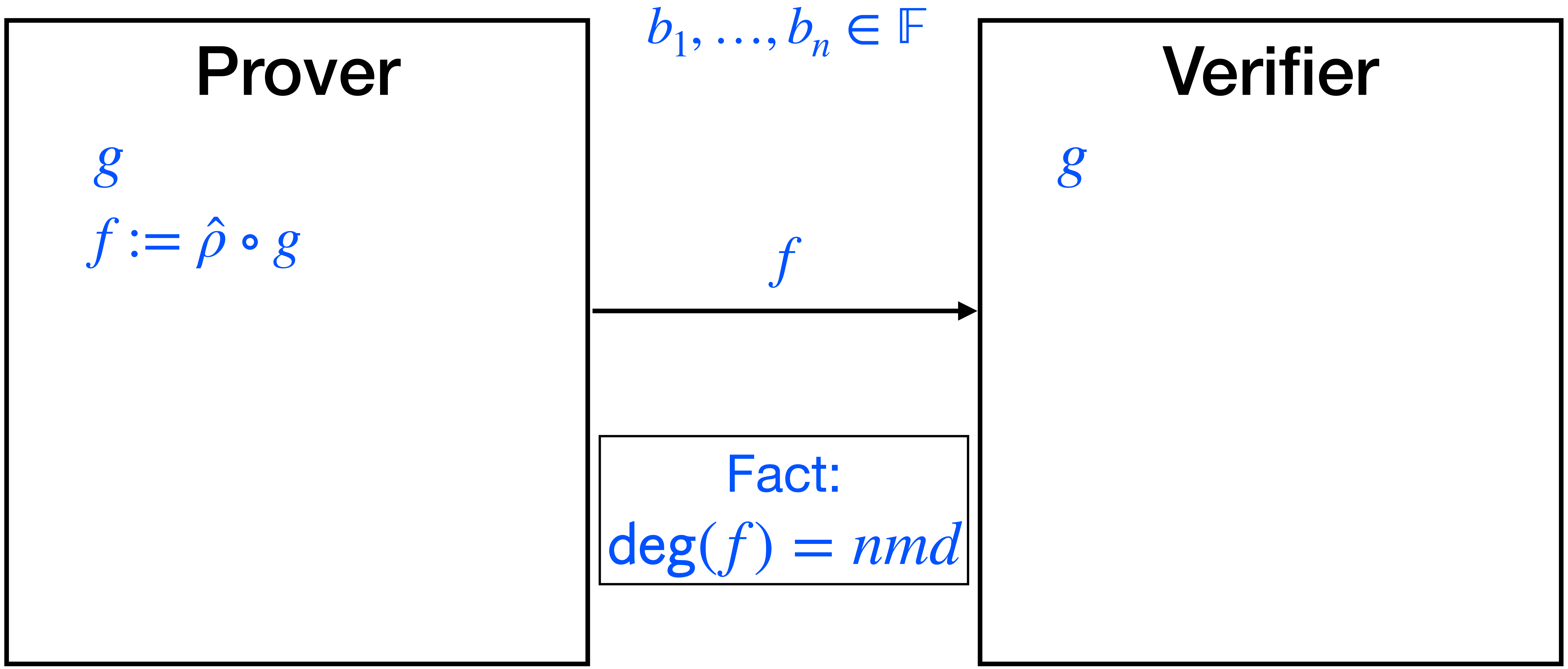


[KalaiRaz08] Interactive query reduction protocol

Input: $\{(x_1, y_1), \ldots, (x_n, y_n)\} \in \mathbb{F}^m \times \mathbb{F}$

Goal: Check $\hat{\rho}(x_i) = y_i \; \forall i \in [n]$ with 1 verifier query

Compute a curve g s.t. $g(b_i) = x_i \; \forall i \in [n]$.

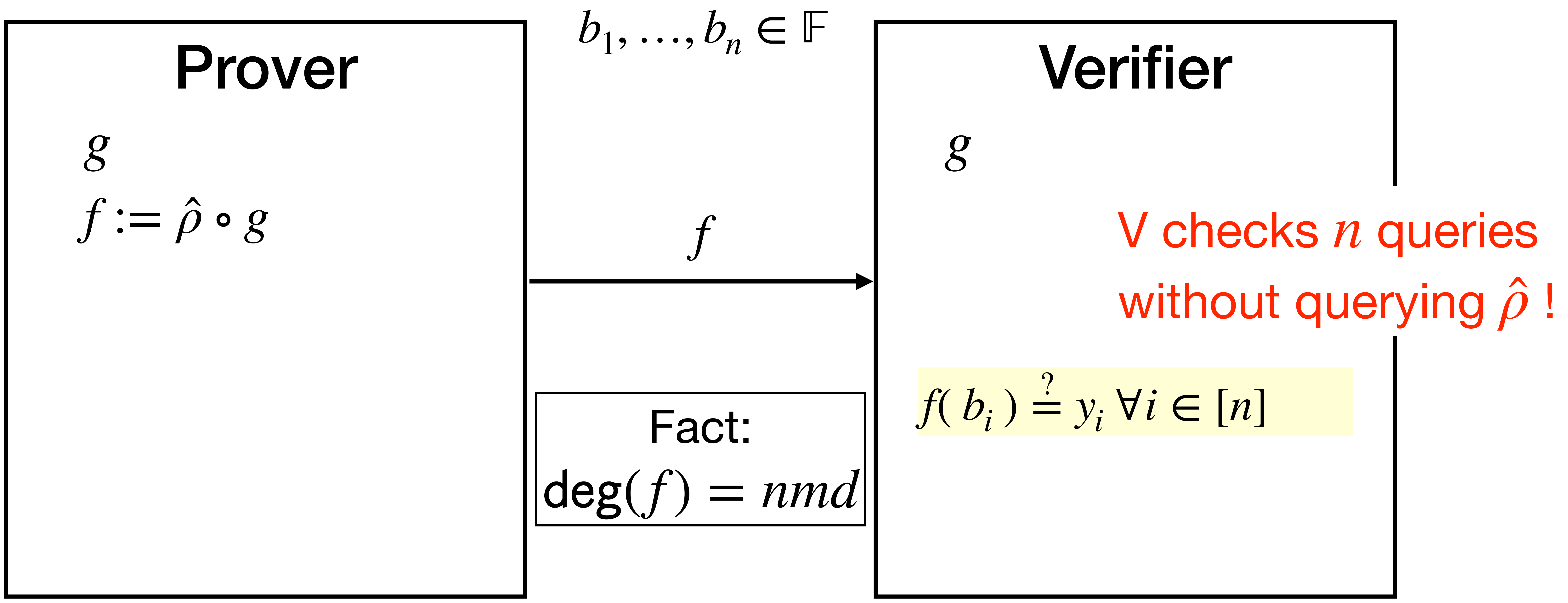


[KalaiRaz08] Interactive query reduction protocol

Input: $\{(x_1, y_1), \ldots, (x_n, y_n)\} \in \mathbb{F}^m \times \mathbb{F}$

Goal: Check $\hat{\rho}(x_i) = y_i \ \forall i \in [n]$ with 1 verifier query

Compute a curve g s.t. $g(b_i) = x_i \ \forall i \in [n]$.

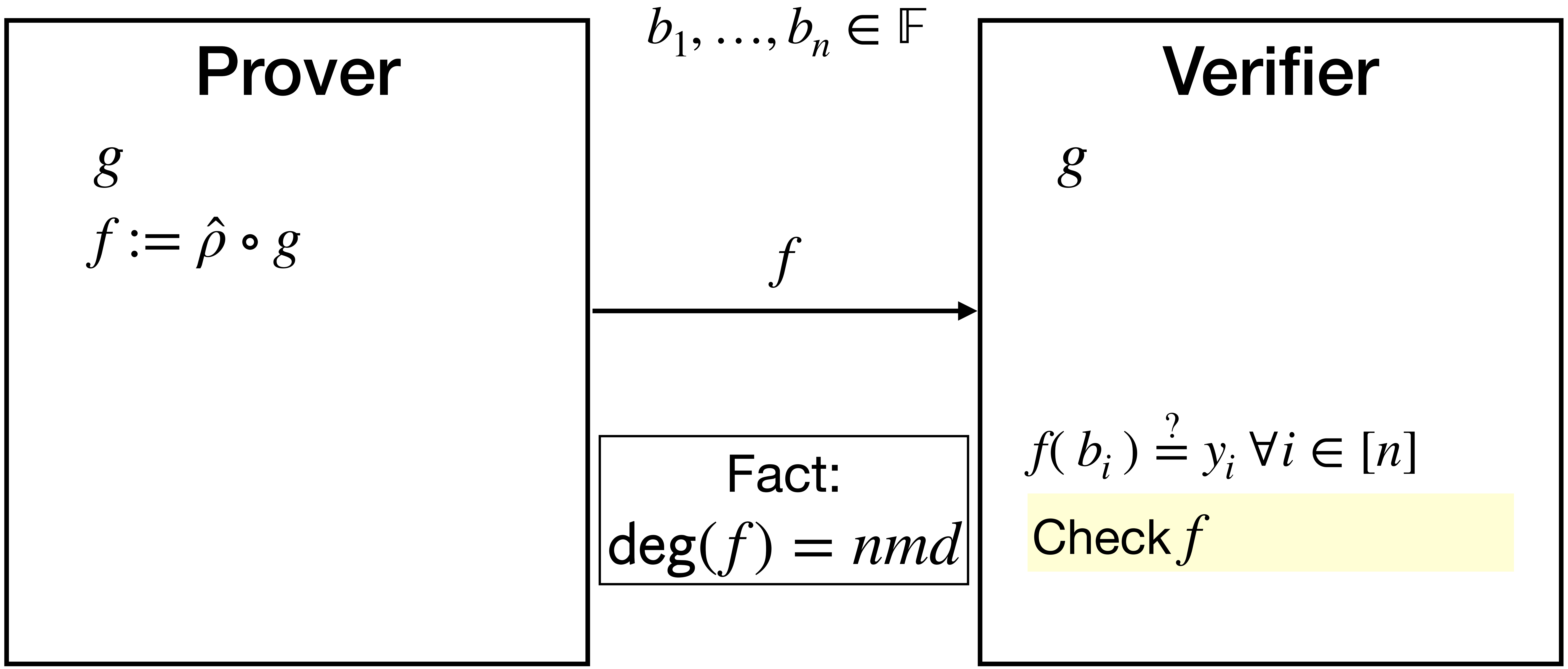


[KalaiRaz08] Interactive query reduction protocol

Input: $\{(x_1, y_1), \ldots, (x_n, y_n)\} \in \mathbb{F}^m \times \mathbb{F}$

Goal: Check $\hat{\rho}(x_i) = y_i \; \forall i \in [n]$ with 1 verifier query

Compute a curve g s.t. $g(b_i) = x_i \; \forall i \in [n]$.

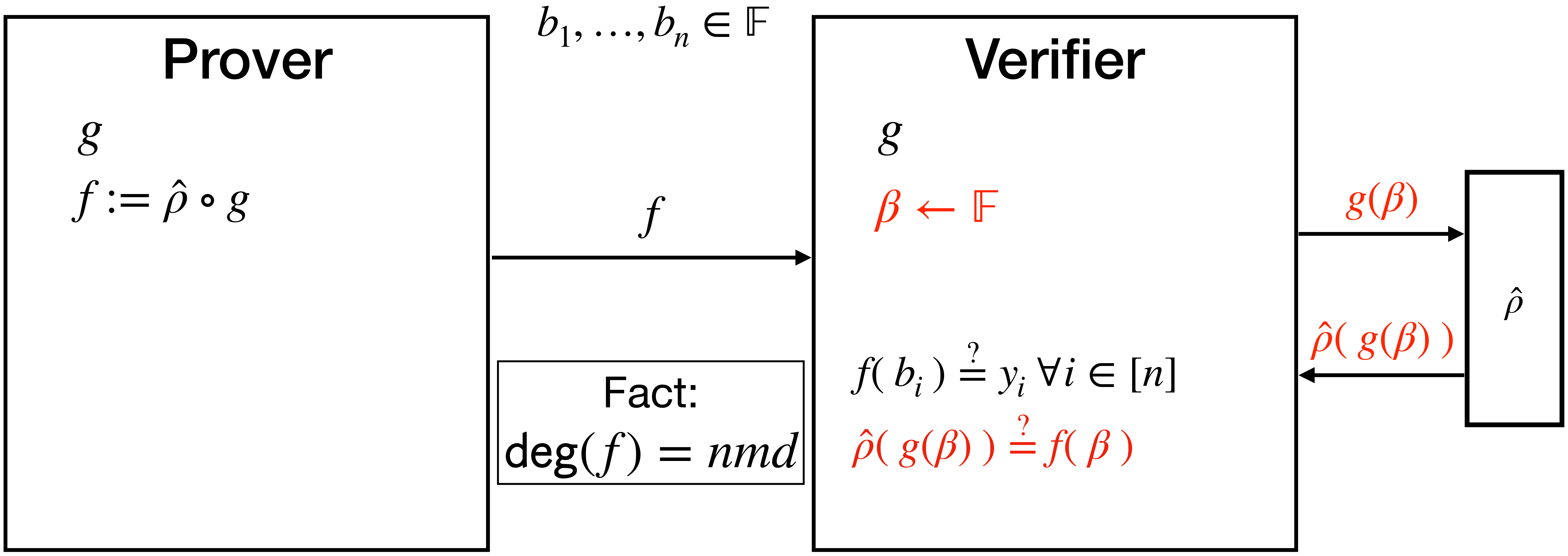


[KalaiRaz08] Interactive query reduction protocol

Input: $\{(x_1, y_1), \ldots, (x_n, y_n)\} \in \mathbb{F}^m \times \mathbb{F}$

Goal: Check $\hat{\rho}(x_i) = y_i \ \forall i \in [n]$ with 1 verifier query

Compute a curve g s.t. $g(b_i) = x_i \ \forall i \in [n]$.

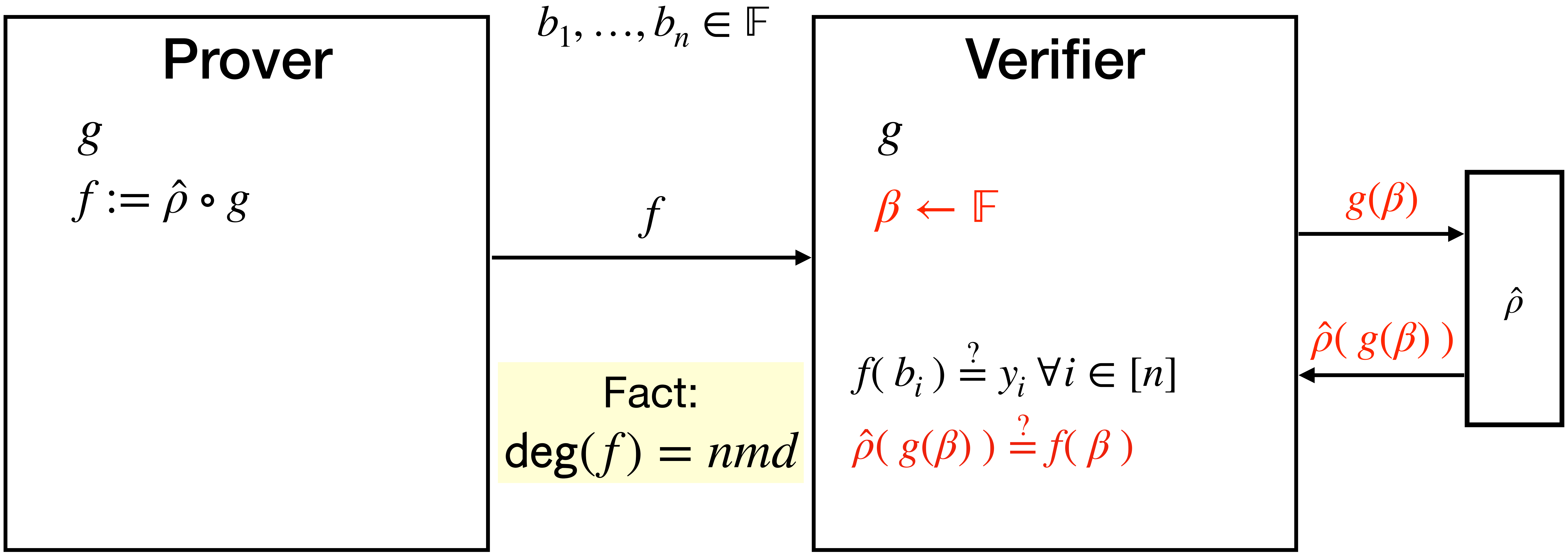


[KalaiRaz08] Interactive query reduction protocol

Input: $\{(x_1, y_1), \ldots, (x_n, y_n)\} \in \mathbb{F}^m \times \mathbb{F}$

Goal: Check $\hat{\rho}(x_i) = y_i \ \forall i \in [n]$ with 1 verifier query

- **Soundness:** $\frac{nmd}{|\mathbb{F}|}$
- **Communication:** $O(nmd)$

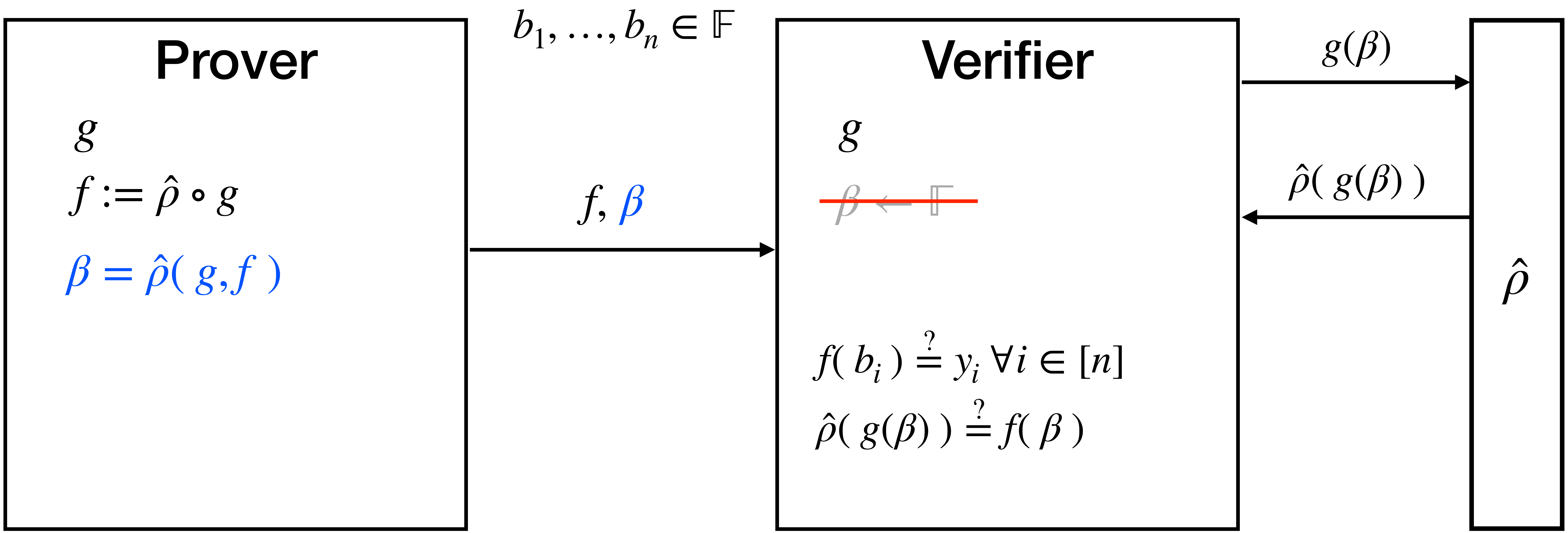


SNARK-friendly [KalaiRaz08]: de-randomize V

Problem: Verifier is randomized (samples β)

Fix: “Fiat-Shamir” transform

Compute a curve g s.t. $g(b_i) = x_i \ \forall i \in [n]$.

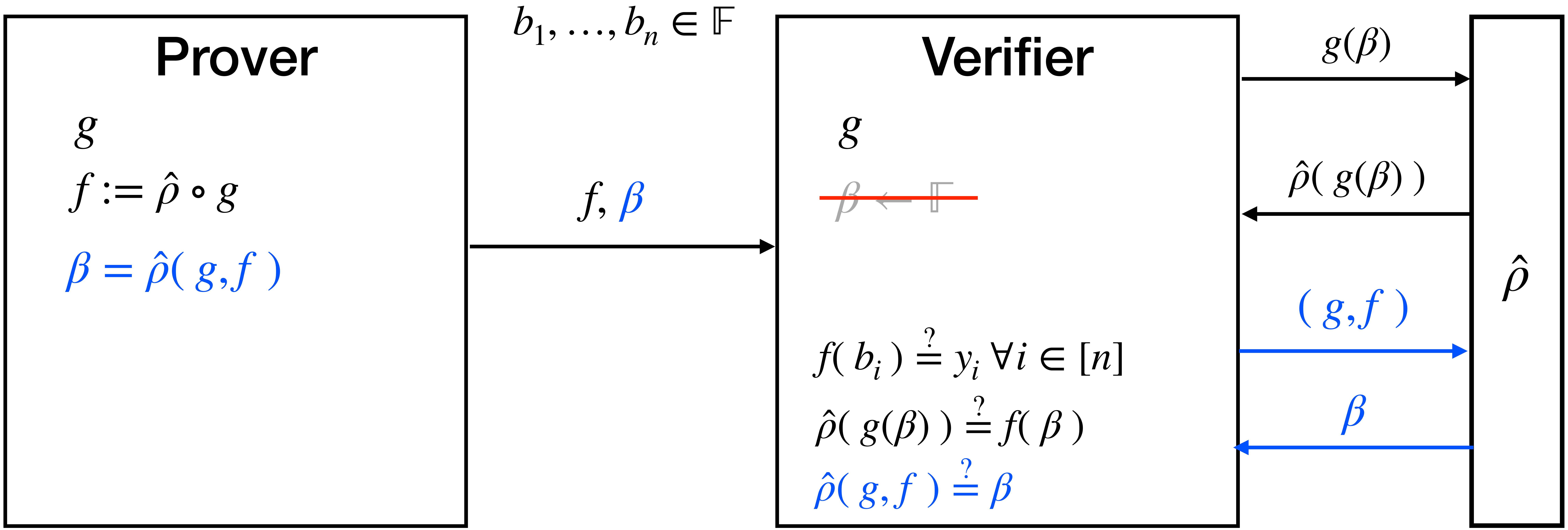


SNARK-friendly [KalaiRaz08]: de-randomize V

Problem: Verifier is randomized (samples β)

Fix: "Fiat-Shamir" transform

Compute a curve g s.t. $g(b_i) = x_i \ \forall i \in [n]$.



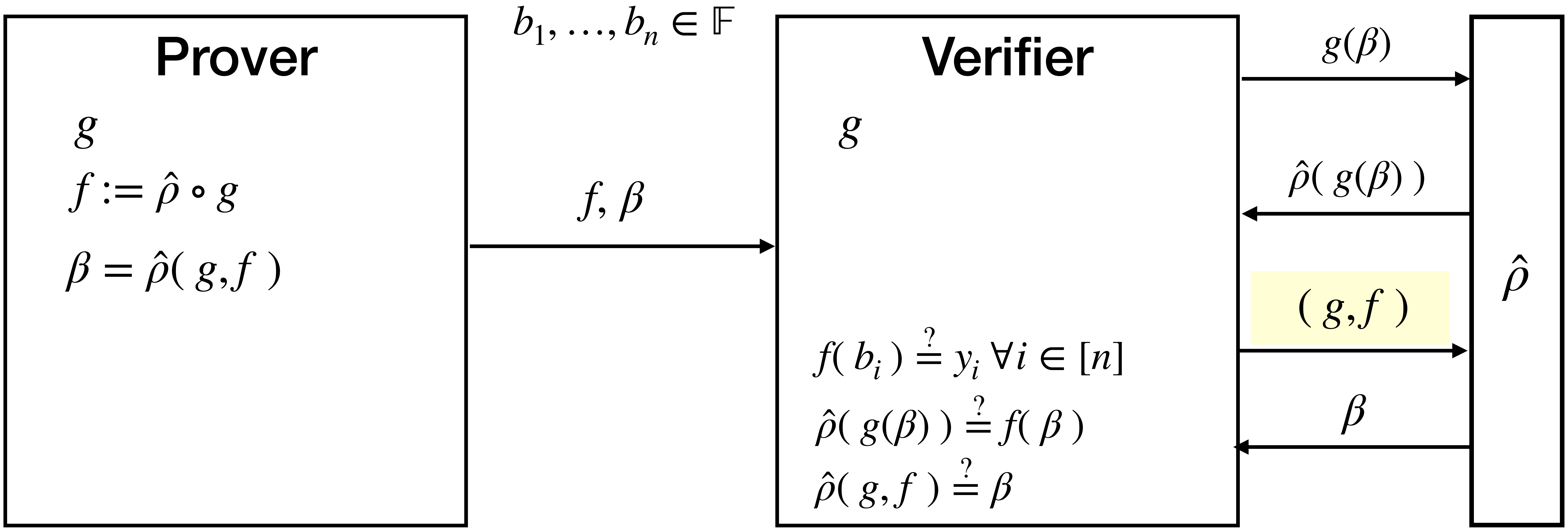
SNARK-friendly [KalaiRaz08]: succinct oracle queries

Problem: $|f|$ linear in the number of queries.

Fix: Use a hash function / compressing commitment

g is specified by $x_1, \ldots, x_n$

Compute a curve g s.t. $g(b_i) = x_i \; \forall i \in [n]$.



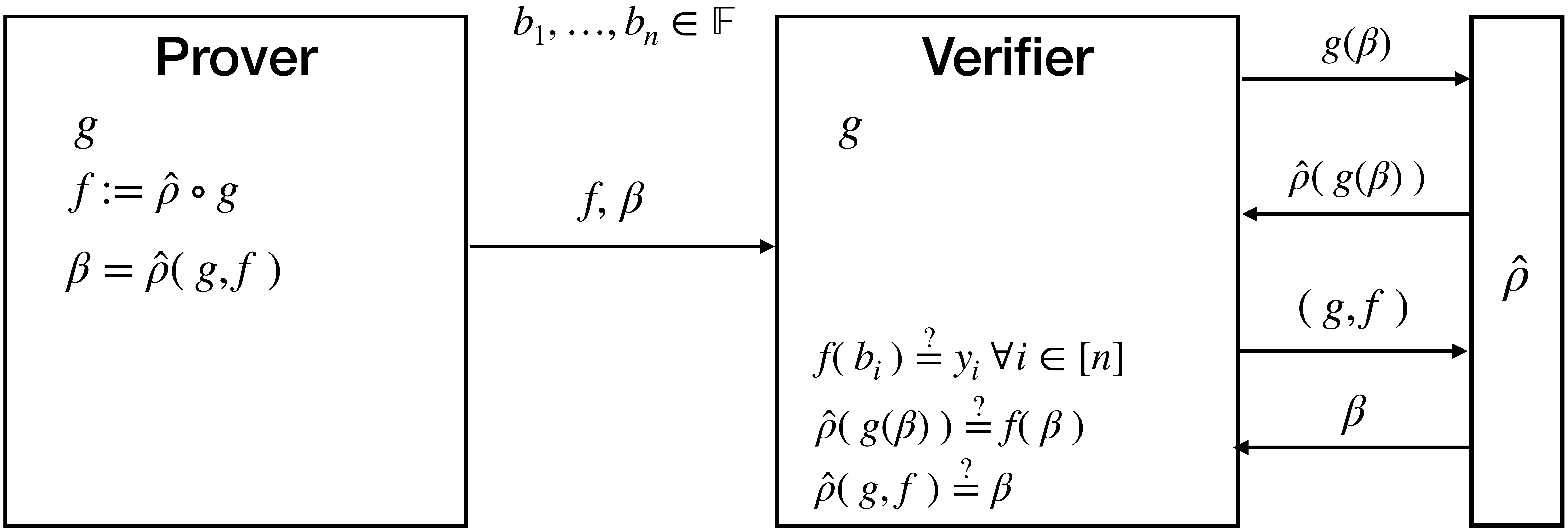
SNARK-friendly [KalaiRaz08]: succinct oracle queries

Problem: $|f|$ linear in the number of queries.

Fix: Use a hash function / compressing commitment

g is specified by $x_1, \ldots, x_n$

Compute a curve g s.t. $g(b_i) = x_i \; \forall i \in [n]$.



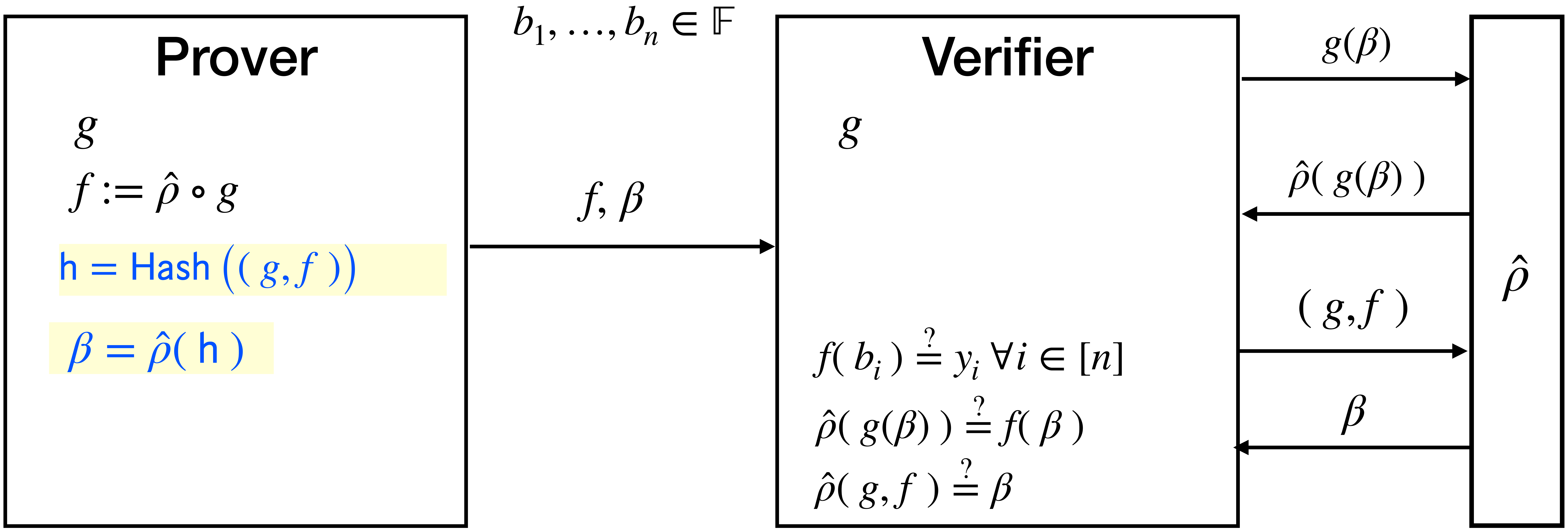
SNARK-friendly [KalaiRaz08]: succinct oracle queries

Problem: $|f|$ linear in the number of queries.

Fix: Use a hash function / compressing commitment

g is specified by $x_1, \ldots, x_n$

Compute a curve g s.t. $g(b_i) = x_i \; \forall i \in [n]$.



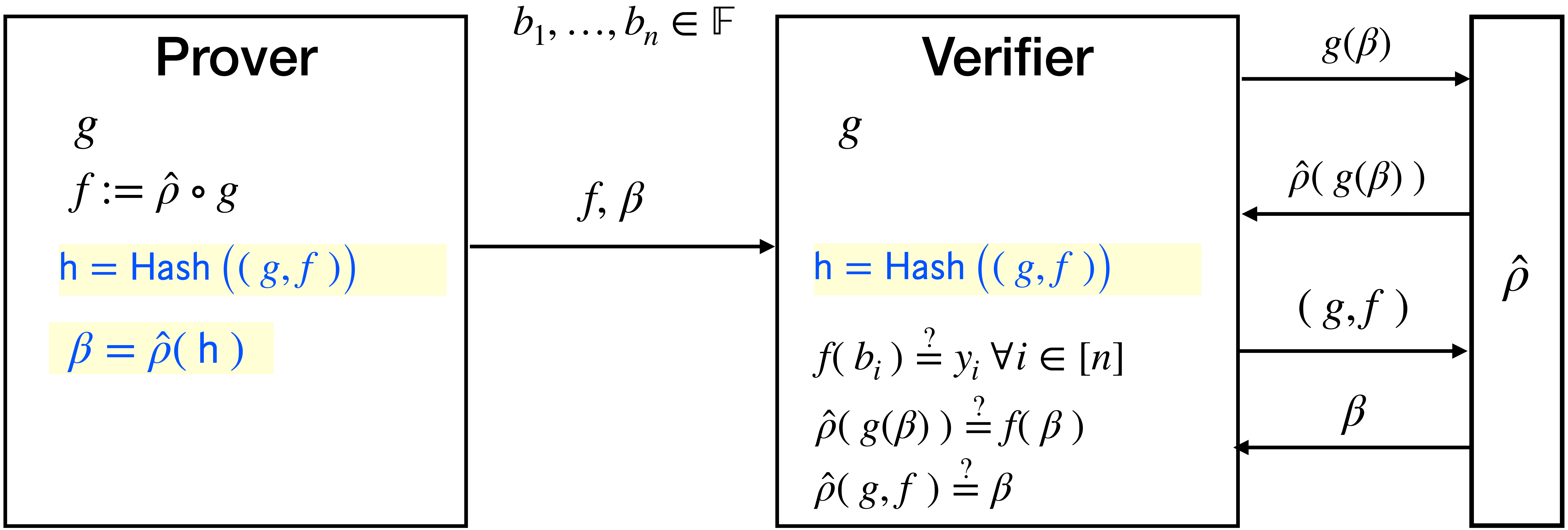
SNARK-friendly [KalaiRaz08]: succinct oracle queries

Problem: $|f|$ linear in the number of queries.

Fix: Use a hash function / compressing commitment

g is specified by $x_1, \ldots, x_n$

Compute a curve g s.t. $g(b_i) = x_i \; \forall i \in [n]$.



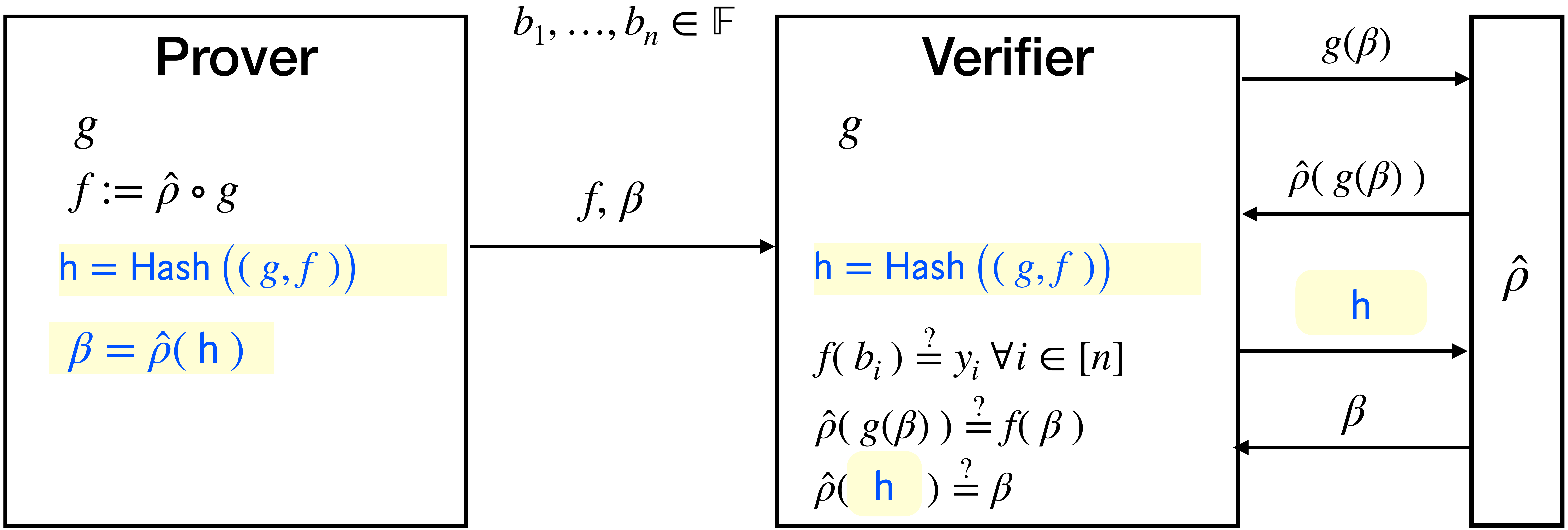
SNARK-friendly [KalaiRaz08]: succinct oracle queries

Problem: $|f|$ linear in the number of queries.

Fix: Use a hash function / compressing commitment

g is specified by $x_1, \ldots, x_n$

Compute a curve g s.t. $g(b_i) = x_i \ \forall i \in [n]$.



Review

We created a scheme that checks LDROM queries efficiently and non-interactively.

Soundness for NI query reduction?

- Bad event:

  Adversary outputs $f \not\equiv \hat{\rho} \circ g$

  s.t. $\beta = \hat{\rho}(\,\mathsf{Hash}(f, g)\,)$ and $f(\,\beta\,) = (\hat{\rho} \circ g)(\,\beta\,)$.

- Proof: uses a new LDROM forking lemma

Other results

Define: low-degree random oracle (LDRO)

Correctness of $NP^{\hat{\rho}}$ computation = Correctness of NP computation + Succinct verification of M's $\hat{\rho}$ queries

SNARK in LDROM for LDROM computations = **SNARK in LDROM for non-oracle computations** + **NI query reduction for LDROM queries**

Uses ideas from [KalaiRaz08].

Other results

Define: low-degree random oracle (LDRO)

Correctness of $NP^{\hat{\rho}}$ computation = Correctness of NP computation + Succinct verification of M's $\hat{\rho}$ queries

SNARK in LDROM for LDROM computations = **SNARK in LDROM for non-oracle computations** + **NI query reduction for LDROM queries**

[Micali00] SNARK using LDROM.

Soundness: proved with LDRO forking lemma

Uses ideas from [KalaiRaz08].

Other results

Define: low-degree random oracle (LDRO)

Correctness of $NP^{\hat{\rho}}$ computation = Correctness of NP computation + Succinct verification of M's $\hat{\rho}$ queries

ZK

ZK

SNARK in LDROM for LDROM computations = **SNARK in LDROM for non-oracle computations** + **NI query reduction for LDROM queries**

[Micali00] SNARK using LDROM.

Soundness: proved with LDRO forking lemma

Uses ideas from [KalaiRaz08].

Other results

Define: low-degree random oracle (LDRO)

Correctness of $NP^{\hat{\rho}}$ computation = Correctness of NP computation + Succinct verification of M's $\hat{\rho}$ queries

ZK

SNARK in LDROM for LDROM computations

=

ZK

SNARK in LDROM for non-oracle computations

[Micali00] SNARK using LDROM.

Soundness: proved with LDRO forking lemma

+

ZK

NI query reduction for LDROM queries

Uses ideas from [KalaiRaz08].

Thanks!

https://ia.cr/2022/383