Algebraic Reductions of Knowledge

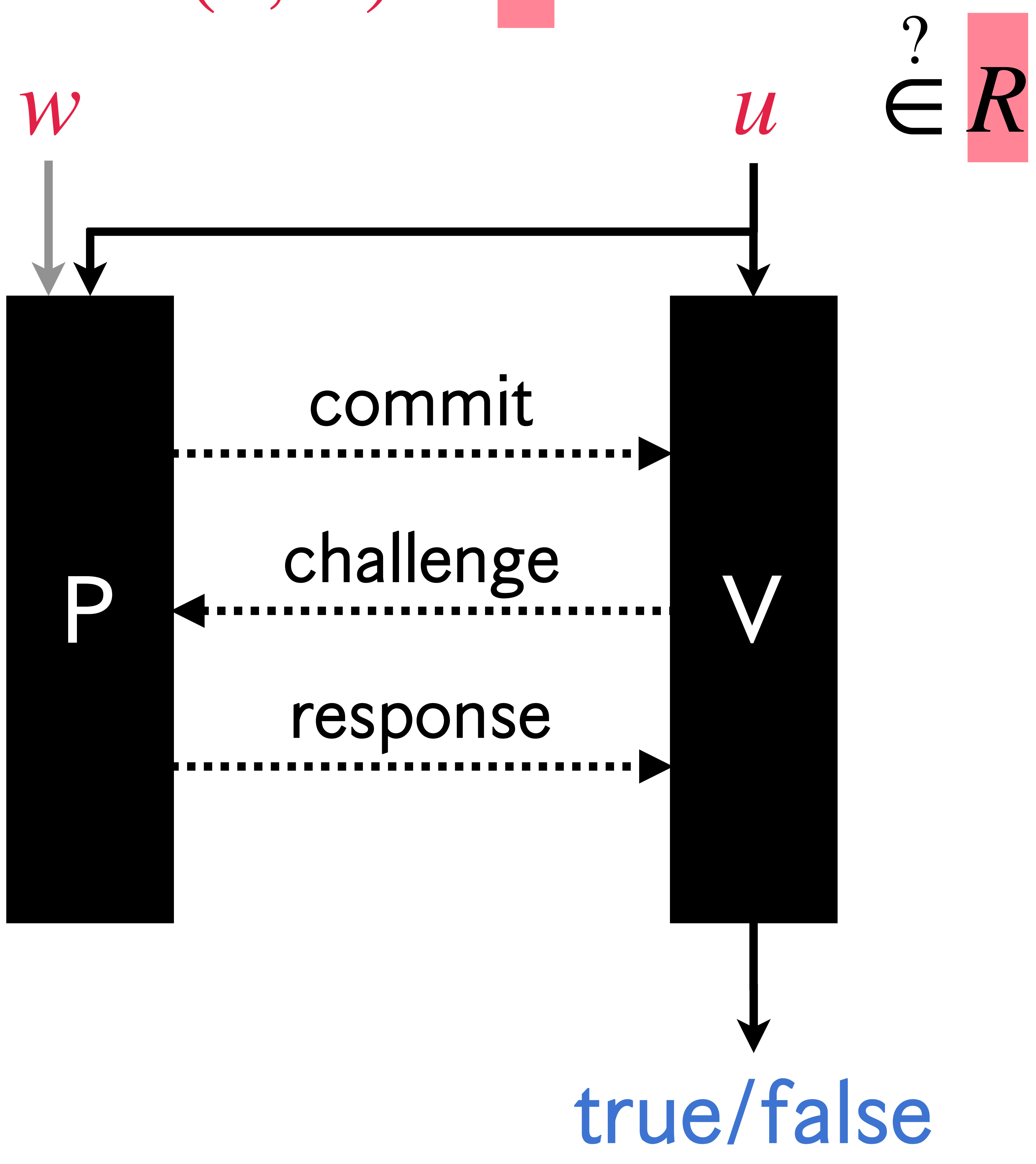
Abhiram Kothapalli [Carnegie Mellon University],
Bryan Parno [Carnegie Mellon University]

akothapalli@cmu.edu

ia.cr/2022/009

Arguments of Knowledge [GMR85]

An argument of knowledge allows a prover to interactively show to a verifier that it *knows* witness w such that $(u, w) \in R$.



A Shift in Perspective

[BDFG21], [RZ21], [ACR21],
[**K**ST22], [BBBPWM18], [BC23],
[BCLMS21], [**K**S23], [CBBZ22],
[BCHO22], [Set20], [Bay13],
[BZ12], [BGH19], [CNRZZ22],
[BCS21], [BMMTV21], [AC20],
[LFKN92], [GKR15], [Lee21],
[Val08], [RZ22], [BCCGP16],
+

Emerging paradigm: The verifier does not fully resolve the prover’s statement, but rather reduces it to a simpler statement to be checked.

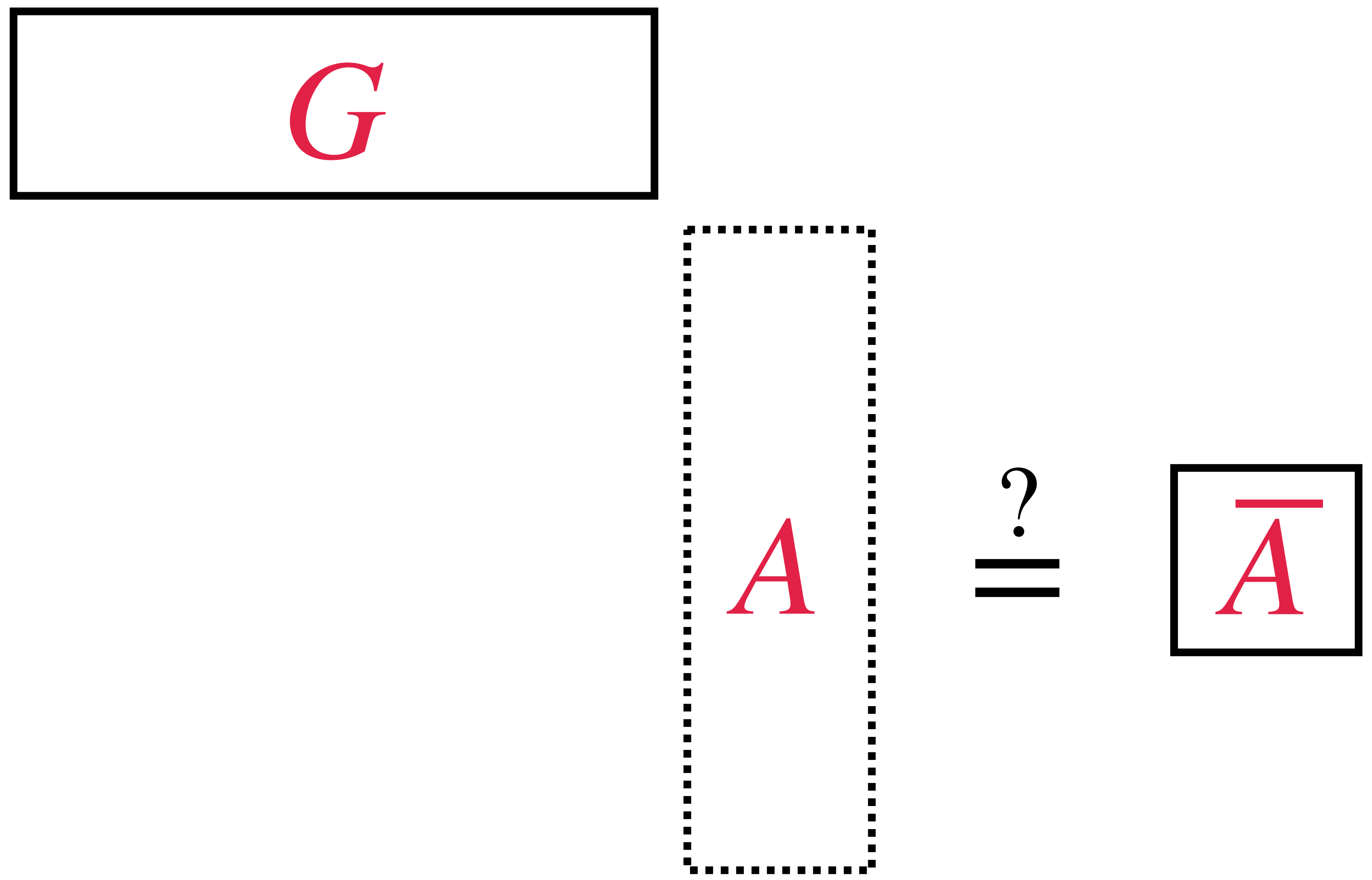
Recursive Inner-Product Argument

"The basic step in our inner product argument is a 2-move reduction to a smaller statement."

- Bootle, Cerulli, Chaidos, Groth, and Petit,

Eurocrypt 2016

Recursive Inner-Product Argument



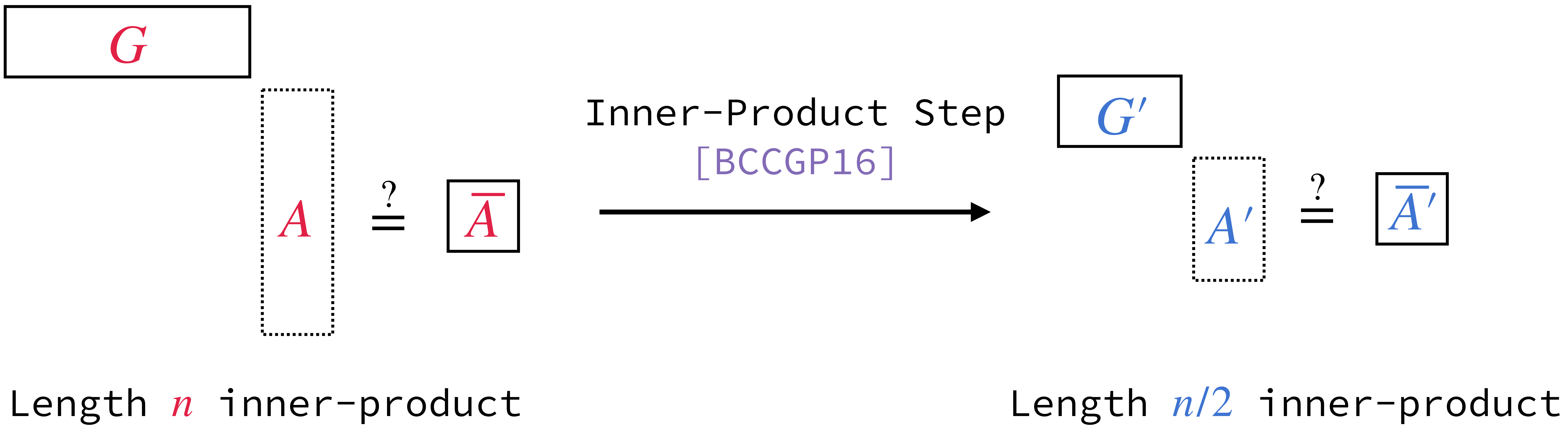
Length n inner-product

"The basic step in our inner product argument is a 2-move reduction to a smaller statement."

- Bootle, Cerulli, Chaidos, Groth, and Petit,

Eurocrypt 2016

Recursive Inner-Product Argument



"The basic step in our inner product argument is a 2-move reduction to a smaller statement."

- Bootle, Cerulli, Chaidos, Groth, and Petit,

Eurocrypt 2016

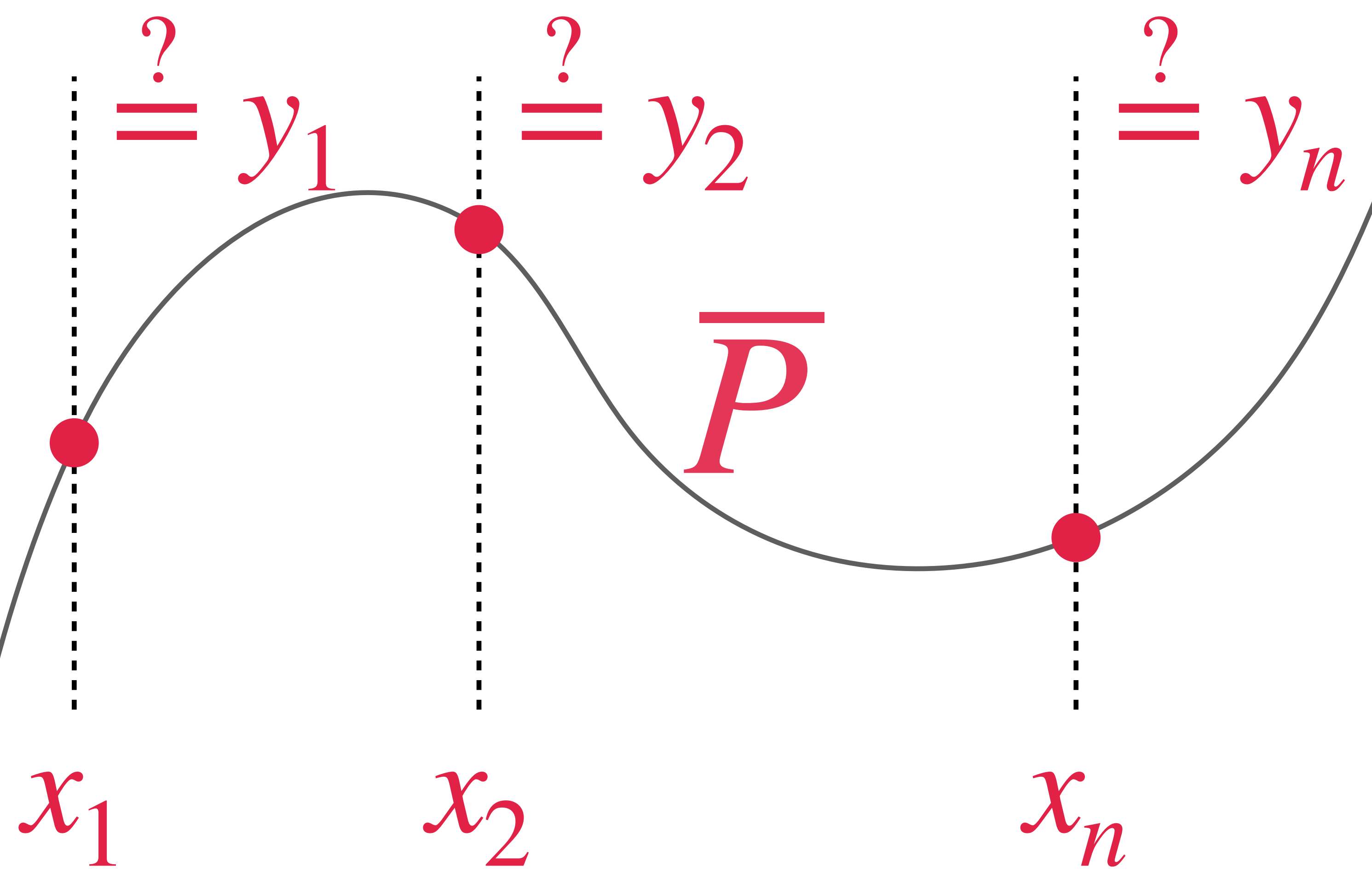
Polynomial Aggregation

"If the prover has a witness for $(\overline{P}, x, y)$, then it must have witnesses for $(\overline{P}, x_1, y_1), \ldots, (\overline{P}, x_n, y_n)$."

- Boneh, Drake, Fisch, and Gabizon,

Crypto 2021

Polynomial Aggregation

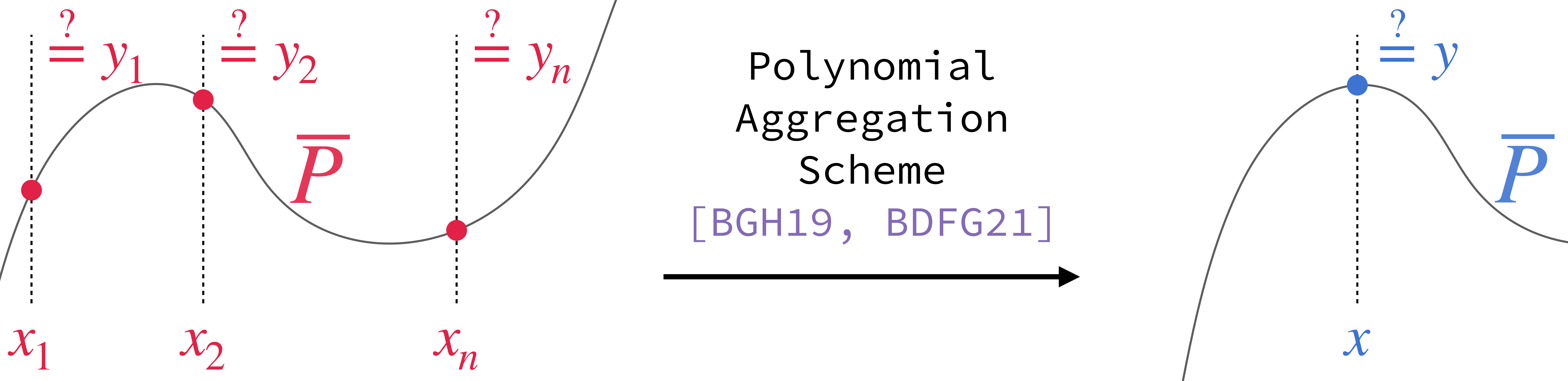


"If the prover has a witness for $(\overline{P}, x, y)$, then it must have witnesses for $(\overline{P}, x_1, y_1), \ldots, (\overline{P}, x_n, y_n)$."

- Boneh, Drake, Fisch, and Gabizon,

Crypto 2021

Polynomial Aggregation



"If the prover has a witness for $(\overline{P}, x, y)$, then it must have witnesses for $(\overline{P}, x_1, y_1), \ldots, (\overline{P}, x_n, y_n)$."

- Boneh, Drake, Fisch, and Gabizon,

Crypto 2021

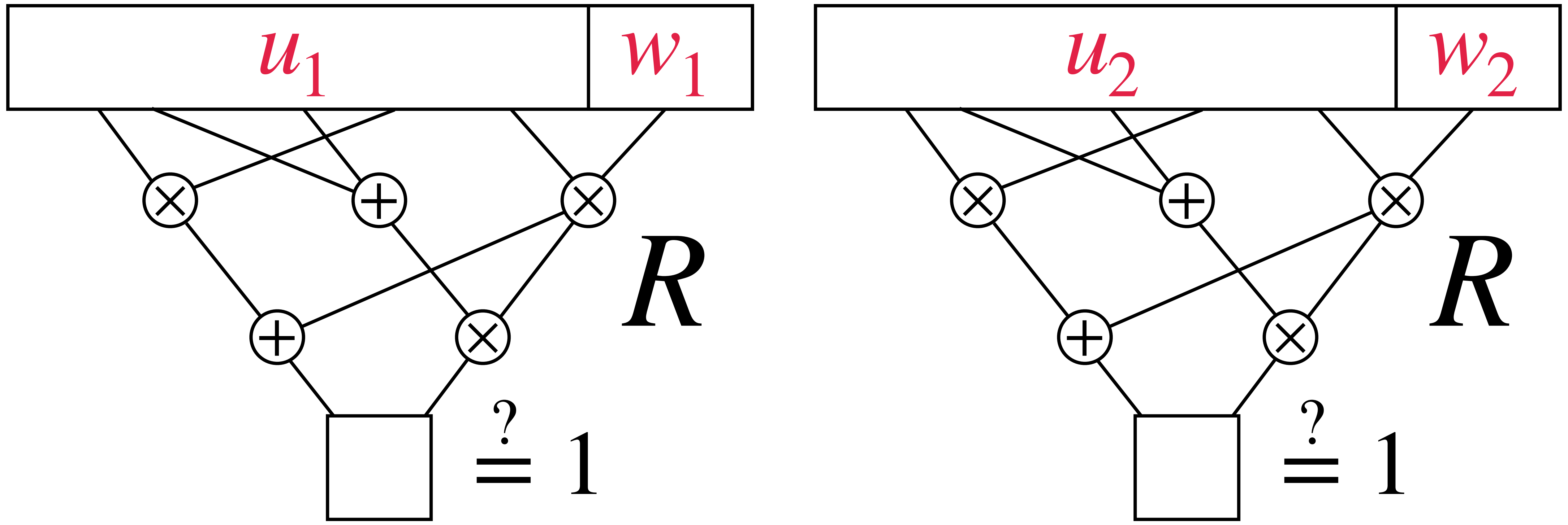
Folding Schemes

"Intuitively, a folding scheme … reduces the task of checking two instances in R to the task of checking a single instance in R."

Joint work with Setty and Tzialla,

Crypto 2022

Folding Schemes

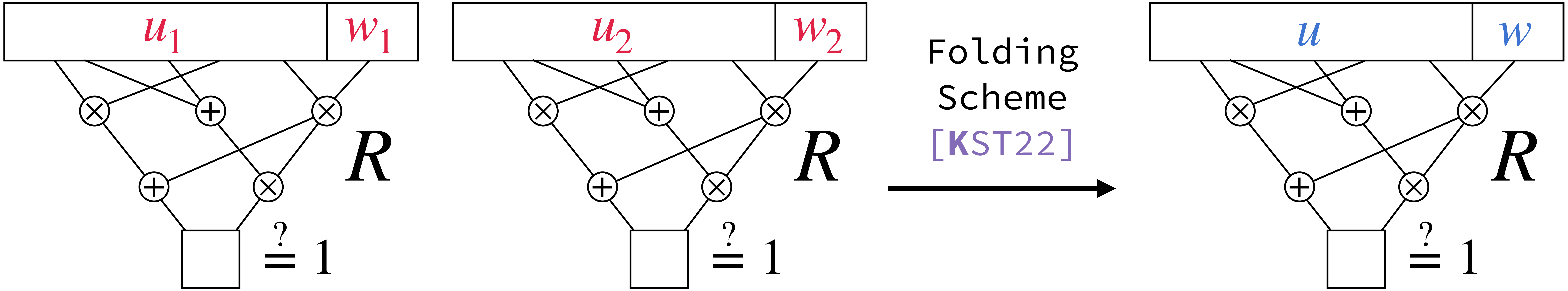


"Intuitively, a folding scheme … reduces the task of checking two instances in R to the task of checking a single instance in R."

Joint work with Setty and Tzialla,

Crypto 2022

Folding Schemes



“Intuitively, a folding scheme … reduces the task of checking two instances in R to the task of checking a single instance in R.”

Joint work with Setty and Tzialla,

Crypto 2022

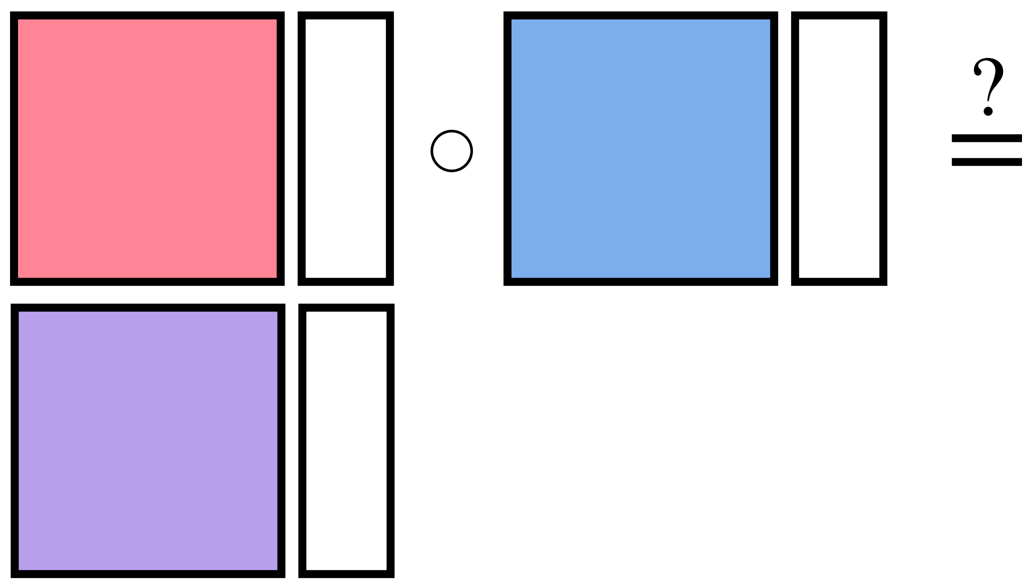
Algebraic Arguments for NP

"We reduce R1CS constraint systems to three algebraic relations"

- Ràfols and Zapico,

Crypto 2021

Algebraic Arguments for NP



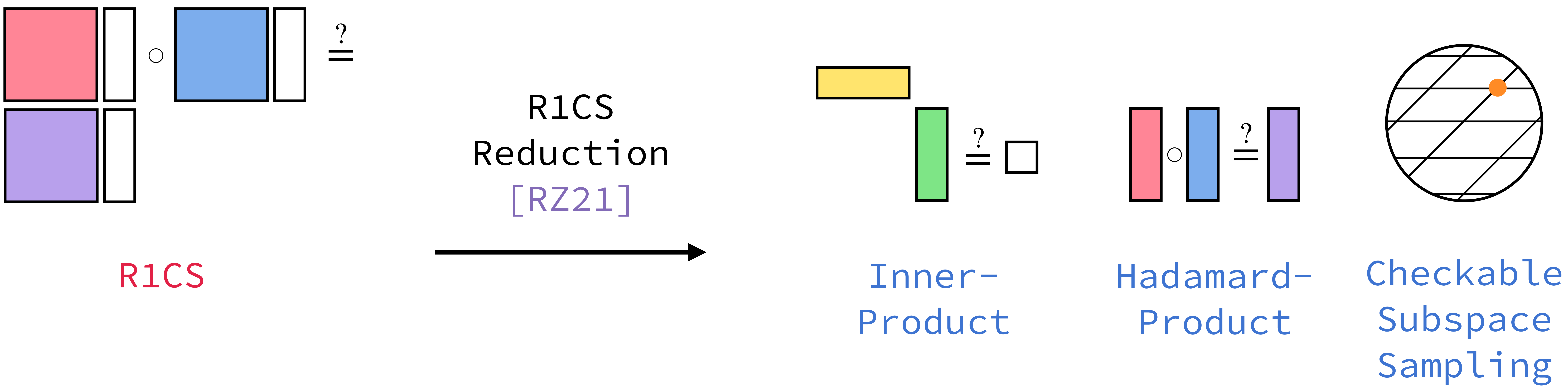
R1CS

"We reduce R1CS constraint systems to three algebraic relations"

- Ràfols and Zapico,

Crypto 2021

Algebraic Arguments for NP



“We reduce R1CS constraint systems to three algebraic relations”

- Ràfols and Zapico,

Crypto 2021

Modern Arguments are Reductions

Split-accumulation schemes reduce the task of checking n instances and accumulators into the task of checking single accumulator. [BCLMS21]

Aggregation schemes for polynomial commitments reduce the task of checking several openings to the task of checking a single opening. [BDFG21]

The ZeroCheck protocol reduces the task of checking that a polynomial vanishes on a set to a Sumcheck. [BTVW14, Set20, CBBZ22]

The tensor-product protocol reduces the task of checking an inner-product with a structured vector to the task of checking several univariate polynomial evaluations. [BCHO22]

The Hadamard-product protocol reduces the task of checking a Hadamard product to the task of checking an inner-product. [Bay13]

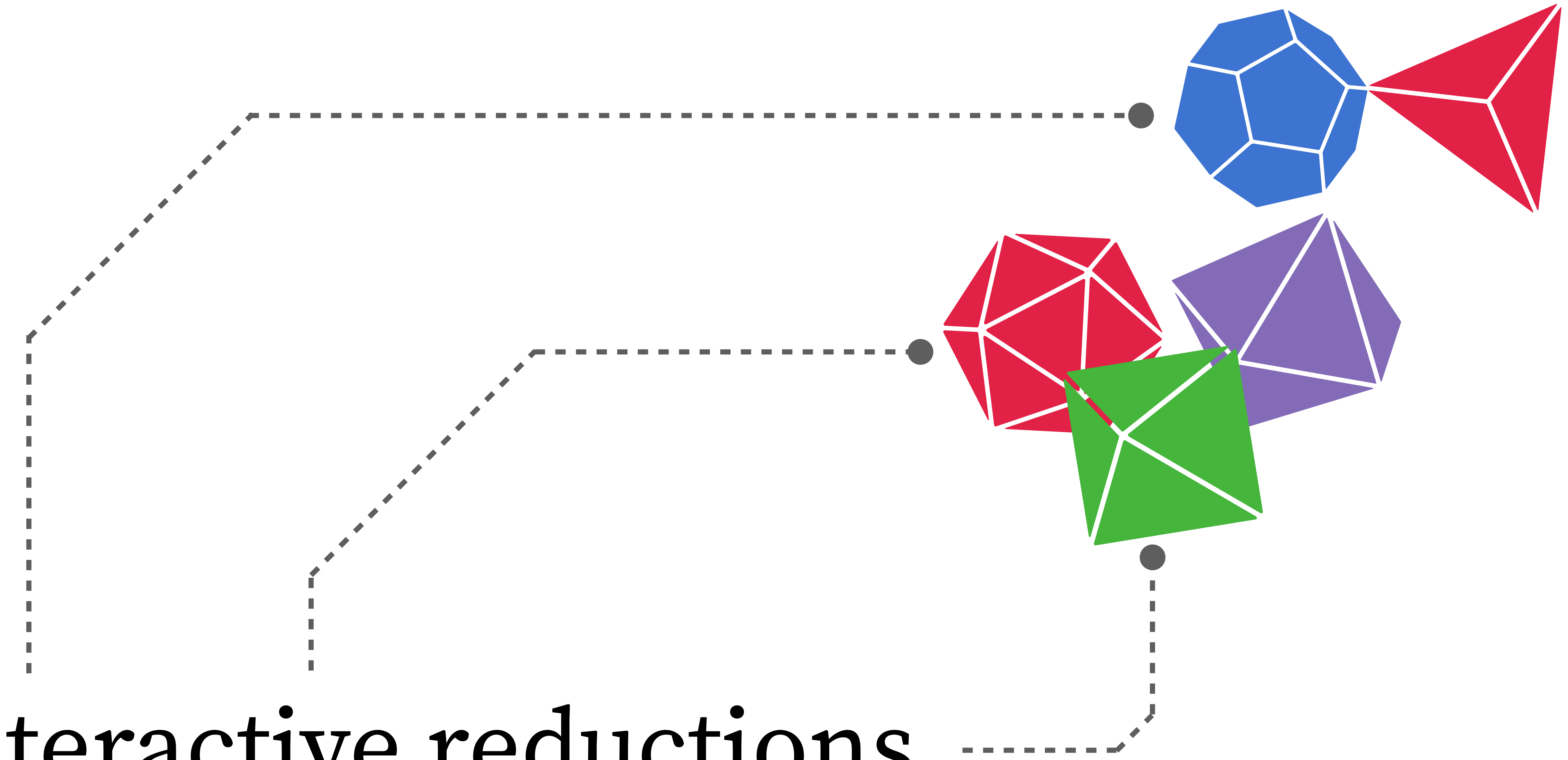
Inner-product arguments reduce the the task of checking the inner-product of size n vectors to checking the inner-product of size $n/2$ vectors. [BCCGP16, BBBPWM18, BMMTV21, Lee21]

Checkable subspace sampling reduces the task of checking matrix evaluations to the task of checking vector evaluations. [RZ21]

Incrementally verifiable computation reduces the task of checking a succinct proof of n applications of function F and a succinct proof of m subsequent applications of F to the task of checking a succinct proof of $n + m$ applications of F. [Val08]

The zero-knowledge HPI argument reduces the task of checking a pre-image of a homomorphism y to the task of checking a pre-image of a randomized homomorphism y'. [BDFG21]

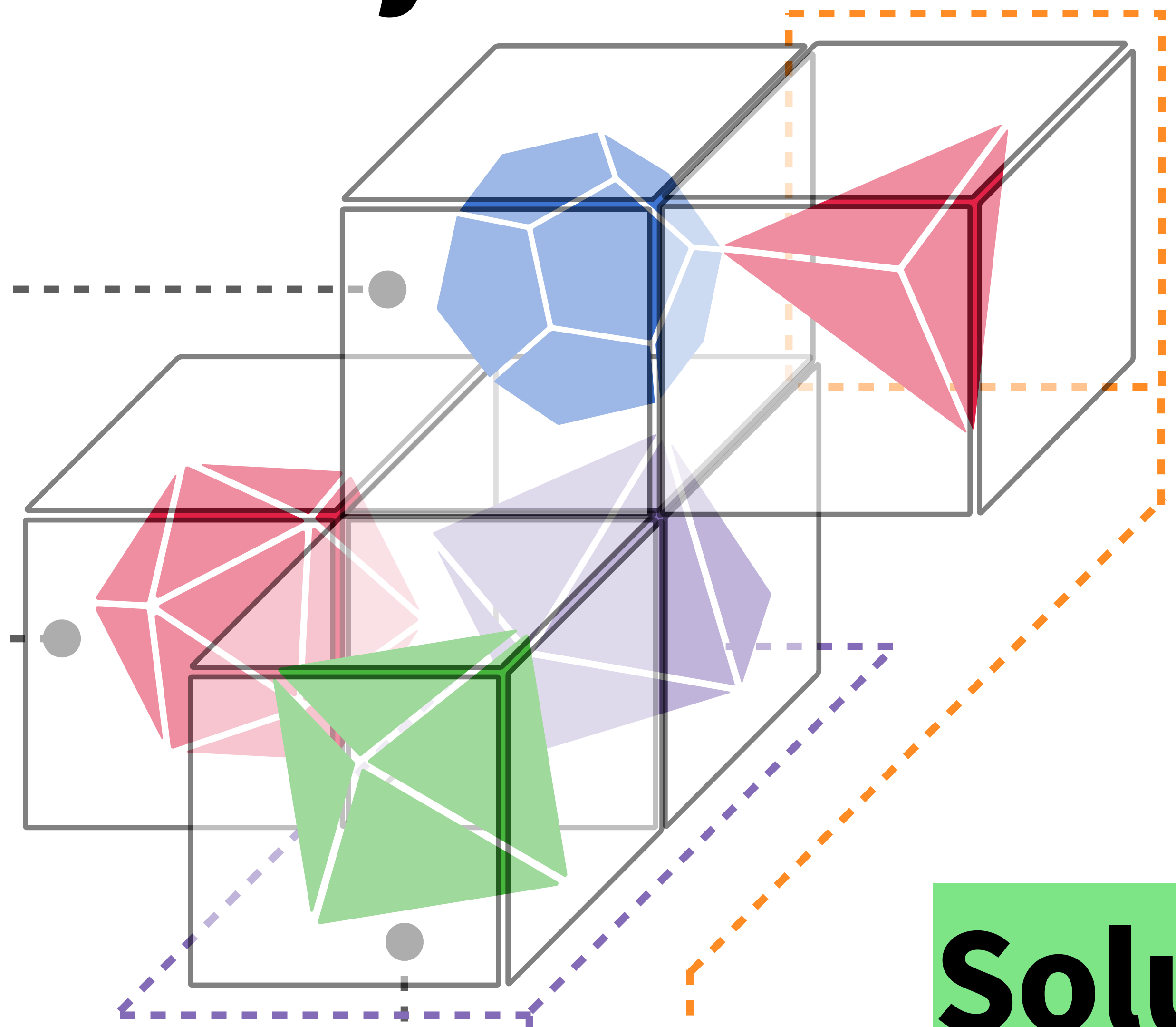
Problem: Need a Unifying Theory



Interactive reductions are universal; their definitions are not

making it difficult to compose compatible techniques hidden under incompatible abstractions.

Problem: Need a Unifying Theory



Interactive reductions are universal; their definitions are not

making it difficult to compose compatible techniques hidden under incompatible abstractions.

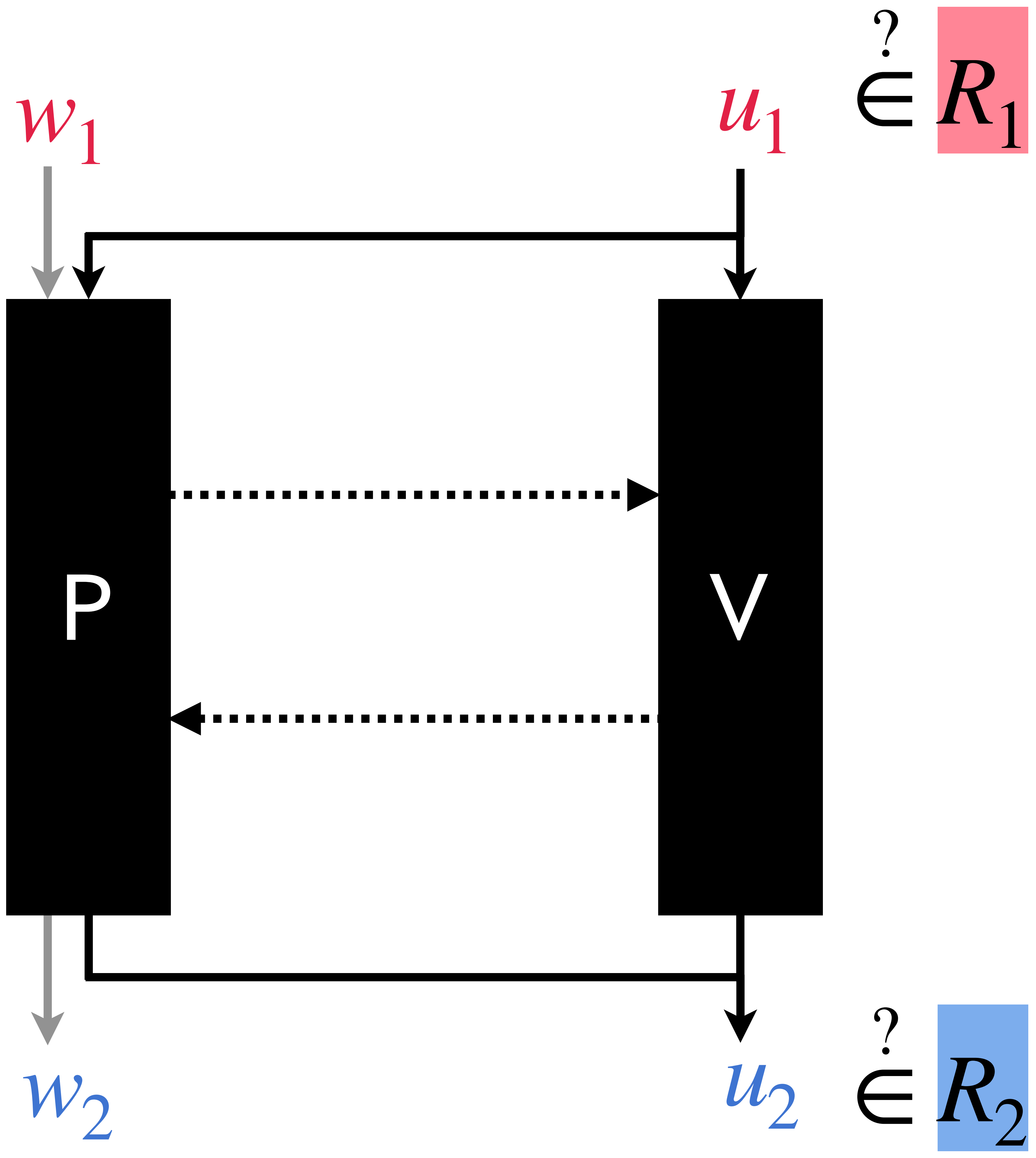
Solution

We formalize **reductions of knowledge** as a common language

which serve as both a unifying abstraction and a compositional framework.

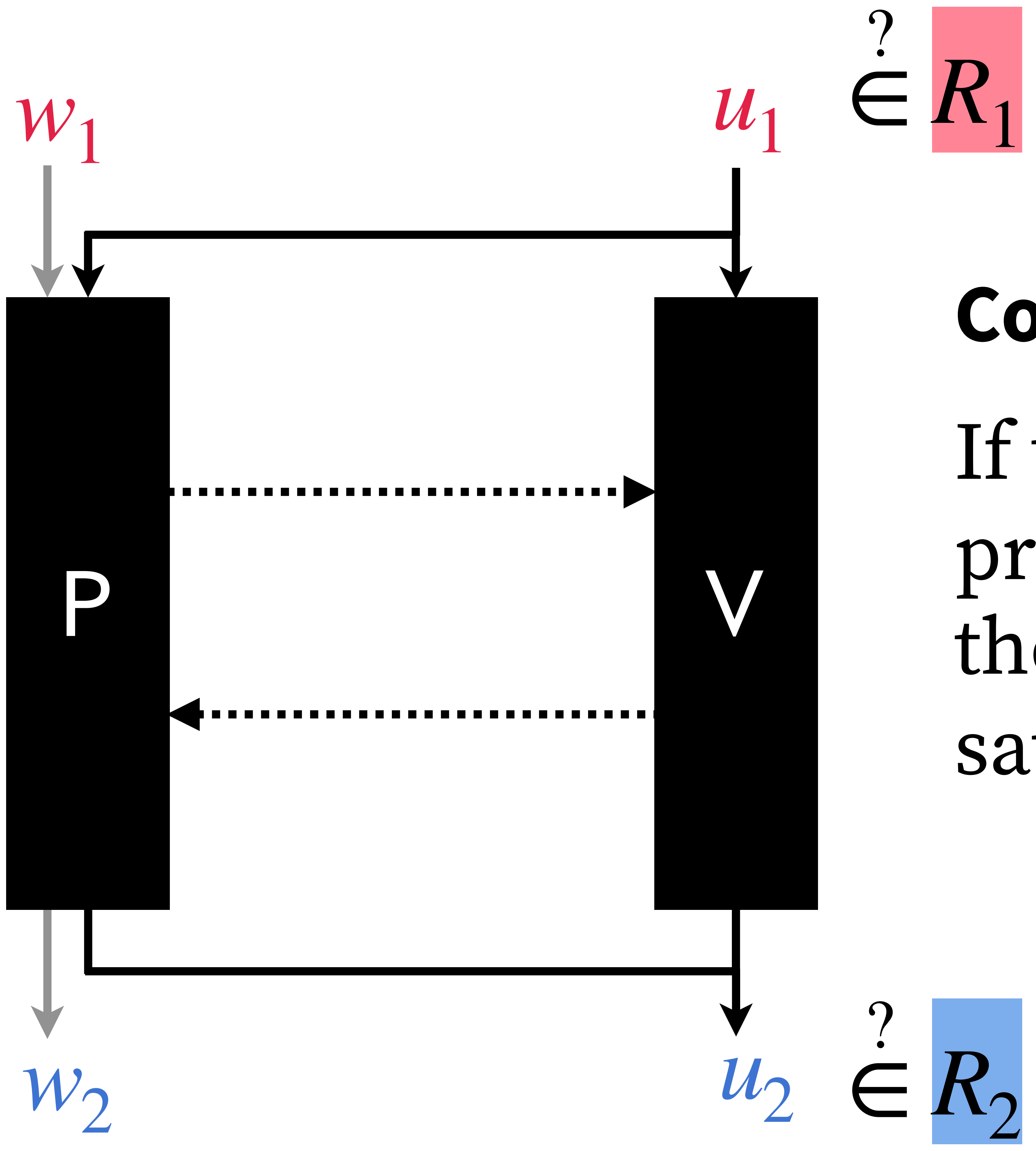
Reductions of Knowledge: A Unifying Language

A reduction of knowledge interactively reduces the claim $(u_1, w_1) \in R_1$ to a claim $(u_2, w_2) \in R_2$



Reductions of Knowledge: A Unifying Language

A reduction of knowledge interactively reduces the claim $(u_1, w_1) \in R_1$ to a claim $(u_2, w_2) \in R_2$

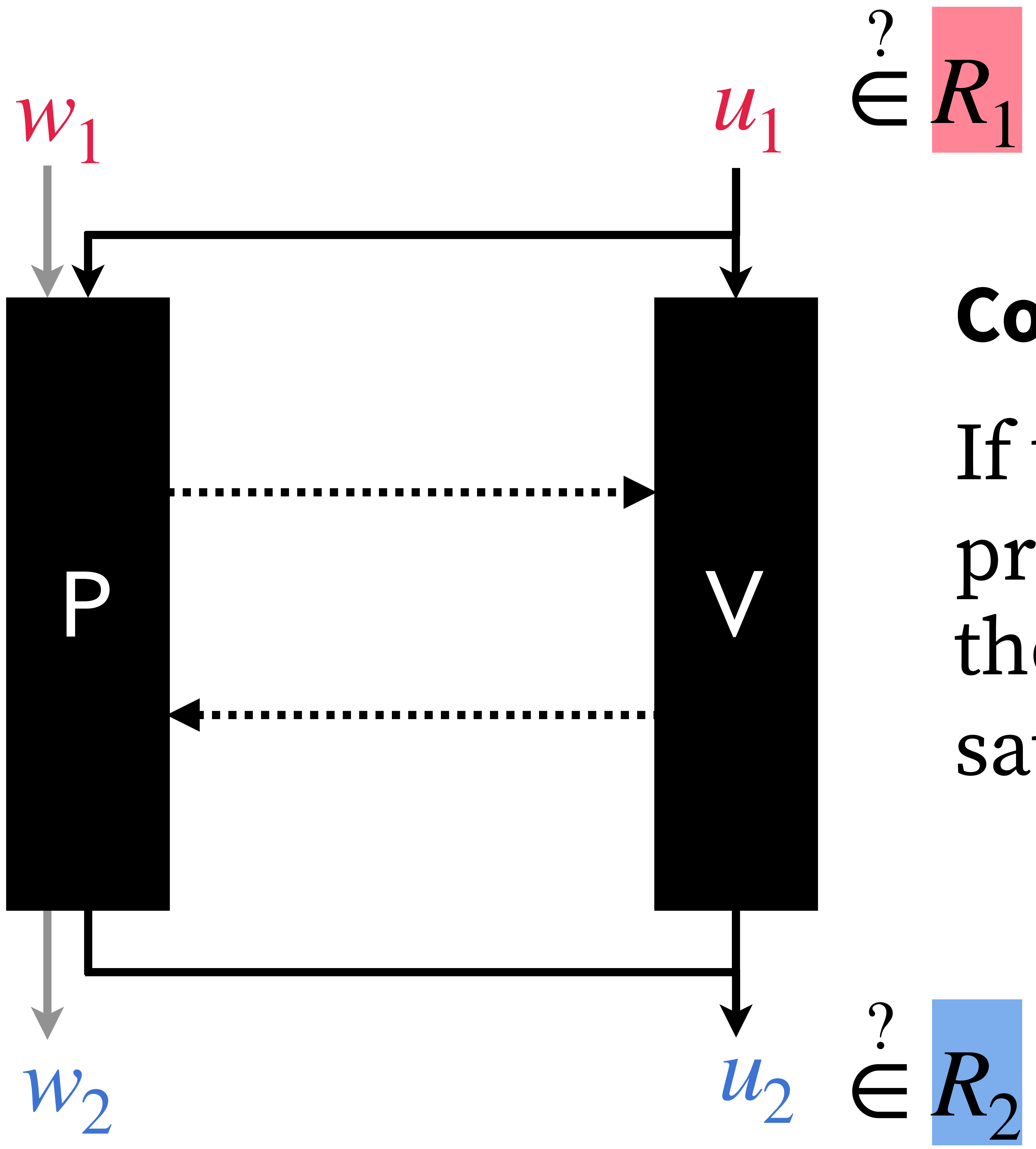


Completeness

If the prover is provided satisfying w_1 then it must output a satisfying w_2

Reductions of Knowledge: A Unifying Language

A reduction of knowledge interactively reduces the claim $(u_1, w_1) \in R_1$ to a claim $(u_2, w_2) \in R_2$



Knowledge Soundness

If prover outputs satisfying w_2 then it must almost certainly *know* a satisfying w_1

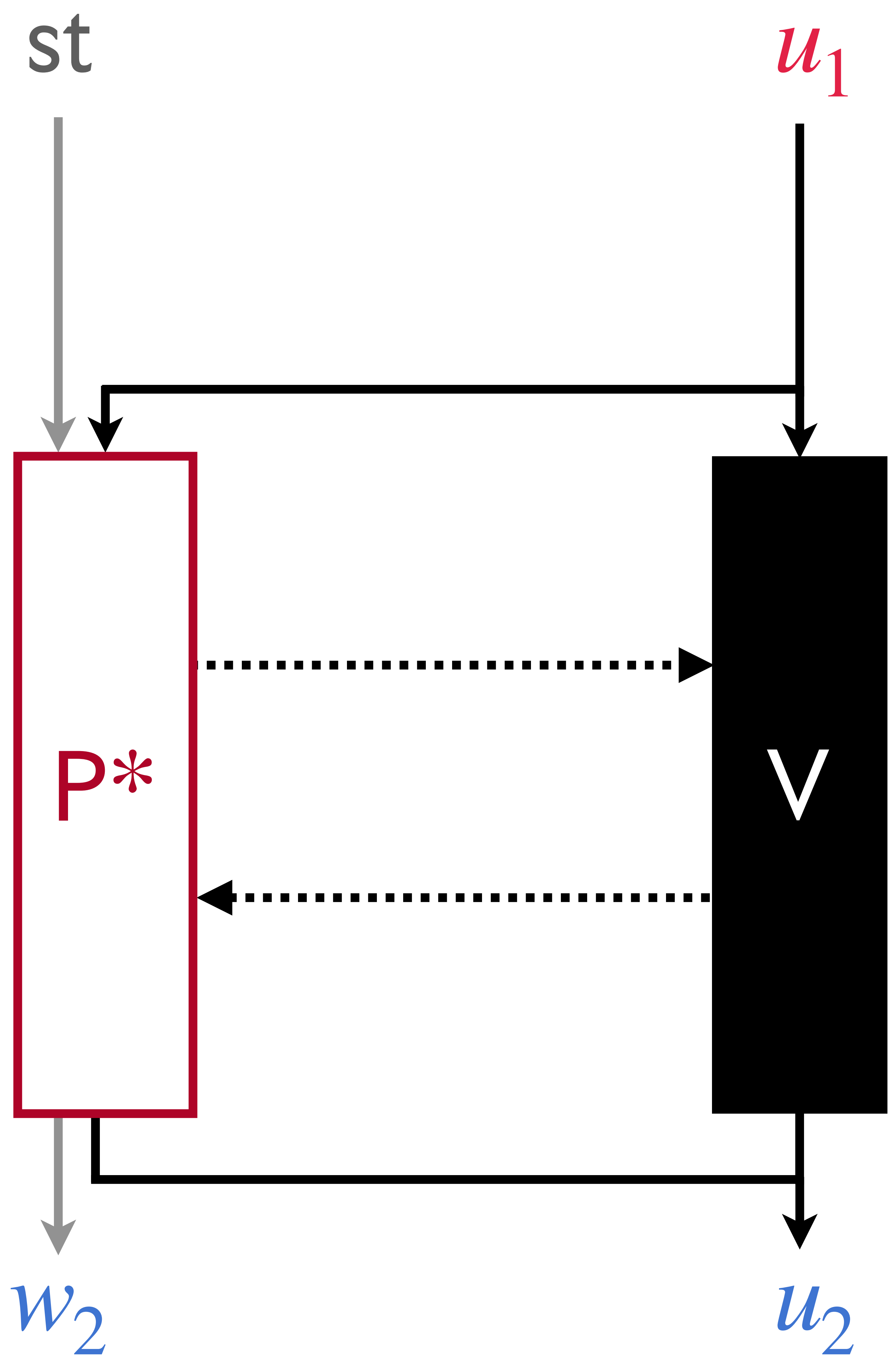
Completeness

If the prover is provided satisfying w_1 then it must output a satisfying w_2

Knowledge Soundness

Consider P^* s.t. for (u_1, st)

$$\Pr[\langle P^*, V\rangle(u_1, \mathrm{st}) \in R_2] = \varepsilon$$



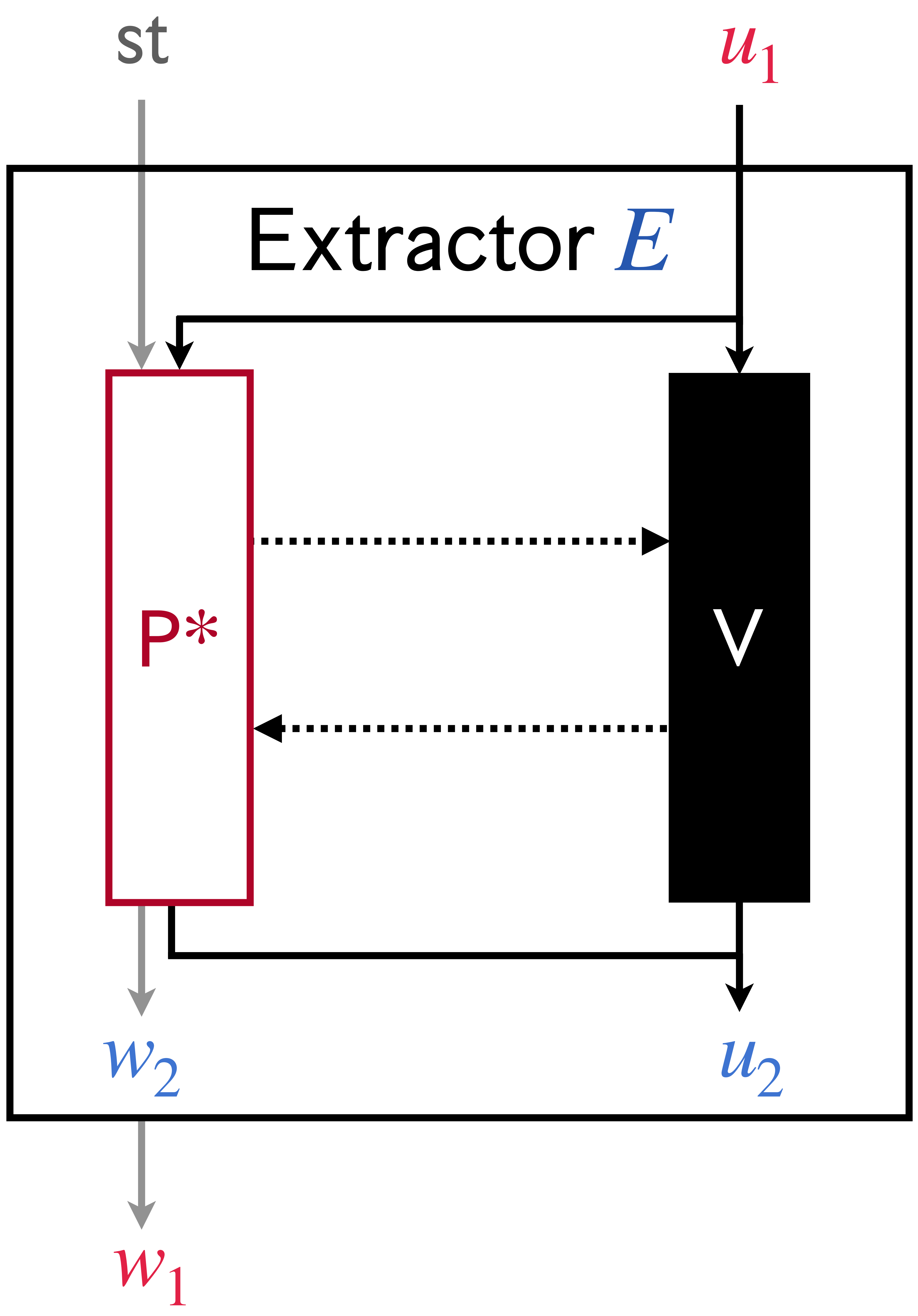
Knowledge Soundness

Consider P^* s.t. for (u_1, st)

$$\Pr[\langle P^*, V\rangle(u_1, \mathsf{st}) \in R_2] = \varepsilon$$

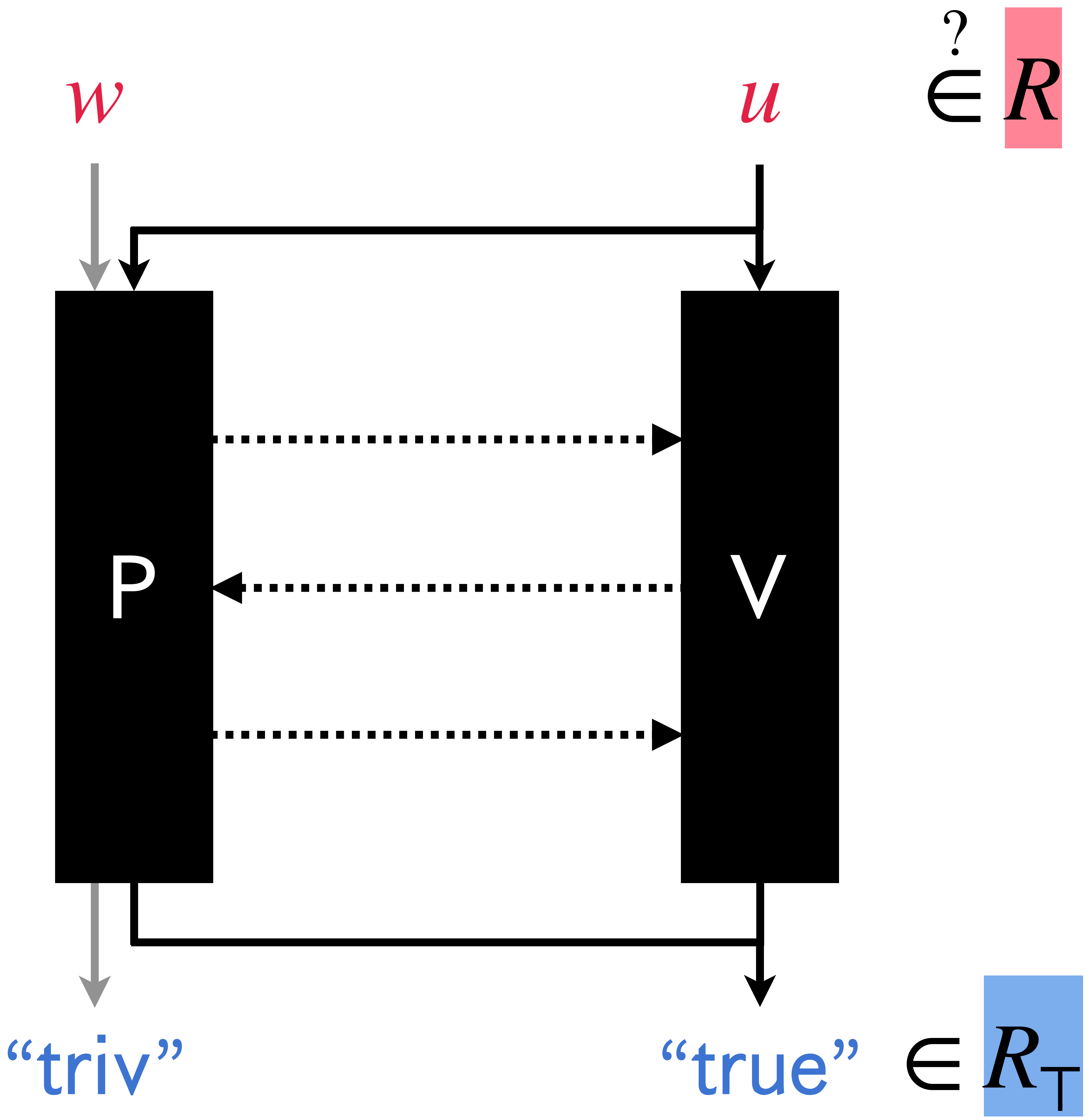
Then there exists an extractor E s.t.

$$\Pr[(u_1, E(u_1, \mathsf{st})) \in R_1] \approx \varepsilon$$



Reconciling Reductions with Arguments

An **argument of knowledge** is a reduction of knowledge from R to $R_\top = \{(\text{"true"}, \text{"triv"})\}$.



First Example: Inner-Product Reduction [BCCGP16]

Define the Inner-Product Relation as

$$R_{\mathsf{IP}}(n) = \left\{ \left((G, \overline{A}), A \right) \in \left((\mathbb{G}^n, \mathbb{G}), \mathbb{F}^n \right) \middle| \left\langle G, A \right\rangle = \overline{A} \right\}$$

First Example: Inner-Product Reduction [BCCGP16]

Define the Inner-Product Relation as

Characterized by length n

$$R_{\mathrm{IP}}(n) = \left\{ \left((G, \bar{A}), A \right) \in \left((\mathbb{G}^n, \mathbb{G}), \mathbb{F}^n \right) \middle| \langle G, A \rangle = \bar{A} \right\}$$

First Example: Inner-Product Reduction [BCCGP16]

Define the Inner-Product Relation as

Characterized by length n

$$R_{\text{IP}}(n) = \left\{ \left((G, \bar{A}), A \right) \in \left((\mathbb{G}^n, \mathbb{G}), \mathbb{F}^n \right) \middle| \langle G, A \rangle = \bar{A} \right\}$$

Statement

First Example: Inner-Product Reduction [BCCGP16]

Define the Inner-Product Relation as

Characterized by length n

Witness

$$R_{\mathsf{IP}}(n) = \left\{ \left((G, \bar{A}),\ A \right) \in \left((\mathbb{G}^n, \mathbb{G}), \mathbb{F}^n \right) \middle| \langle G, A \rangle = \bar{A} \right\}$$

Statement

First Example: Inner-Product Reduction [BCCGP16]

Define the Inner-Product Relation as

Characterized by length n

Witness

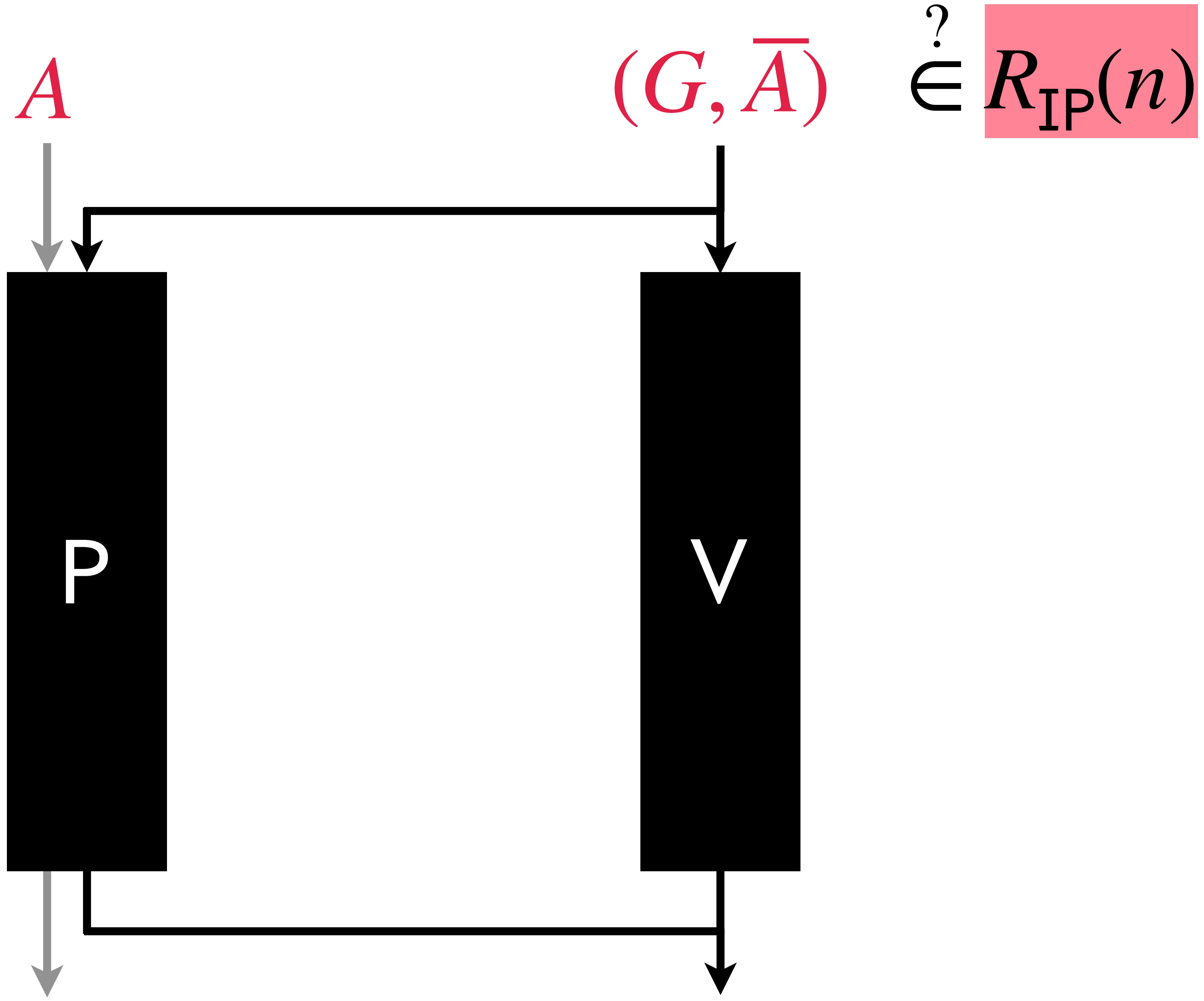
$$R_{\mathsf{IP}}(n) = \left\{ \left((G, \bar{A}),\ A \right) \in \left((\mathbb{G}^n, \mathbb{G}), \mathbb{F}^n \right) \,\middle|\, \langle G, A \rangle = \bar{A} \right\}$$

Statement

Satisfying Condition

First Example: Inner-Product Reduction [BCCGP16]

There exists a reduction of knowledge from $R_{\mathsf{IP}}(n)$ to $R_{\mathsf{IP}}(n/2)$.

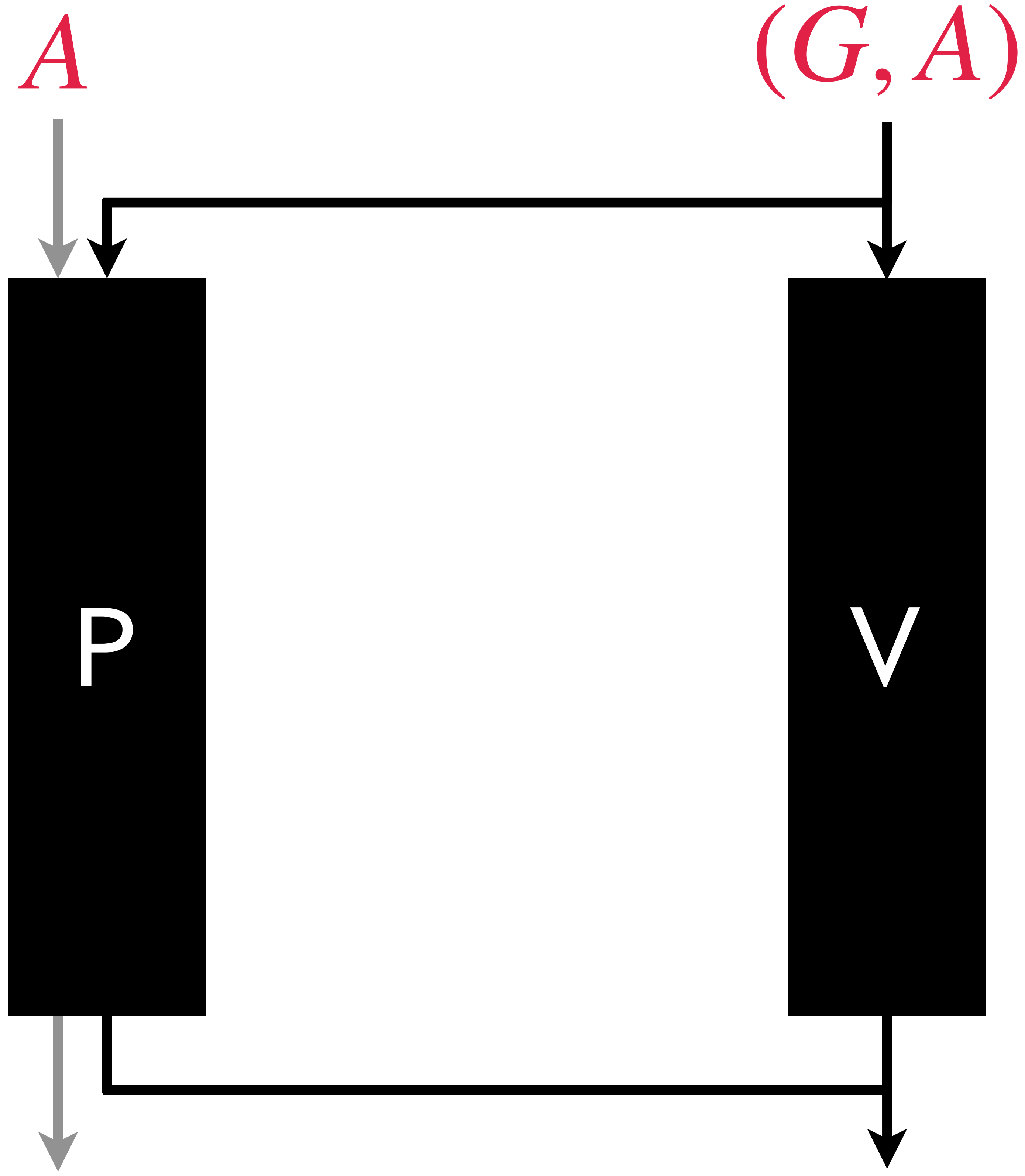


First Example: Inner-Product Reduction [BCCGP16]

There exists a reduction of knowledge from $R_{\mathtt{IP}}(n)$ to $R_{\mathtt{IP}}(n/2)$.

$(G, \overline{A}) \stackrel{?}{\in} R_{\mathtt{IP}}(n)$

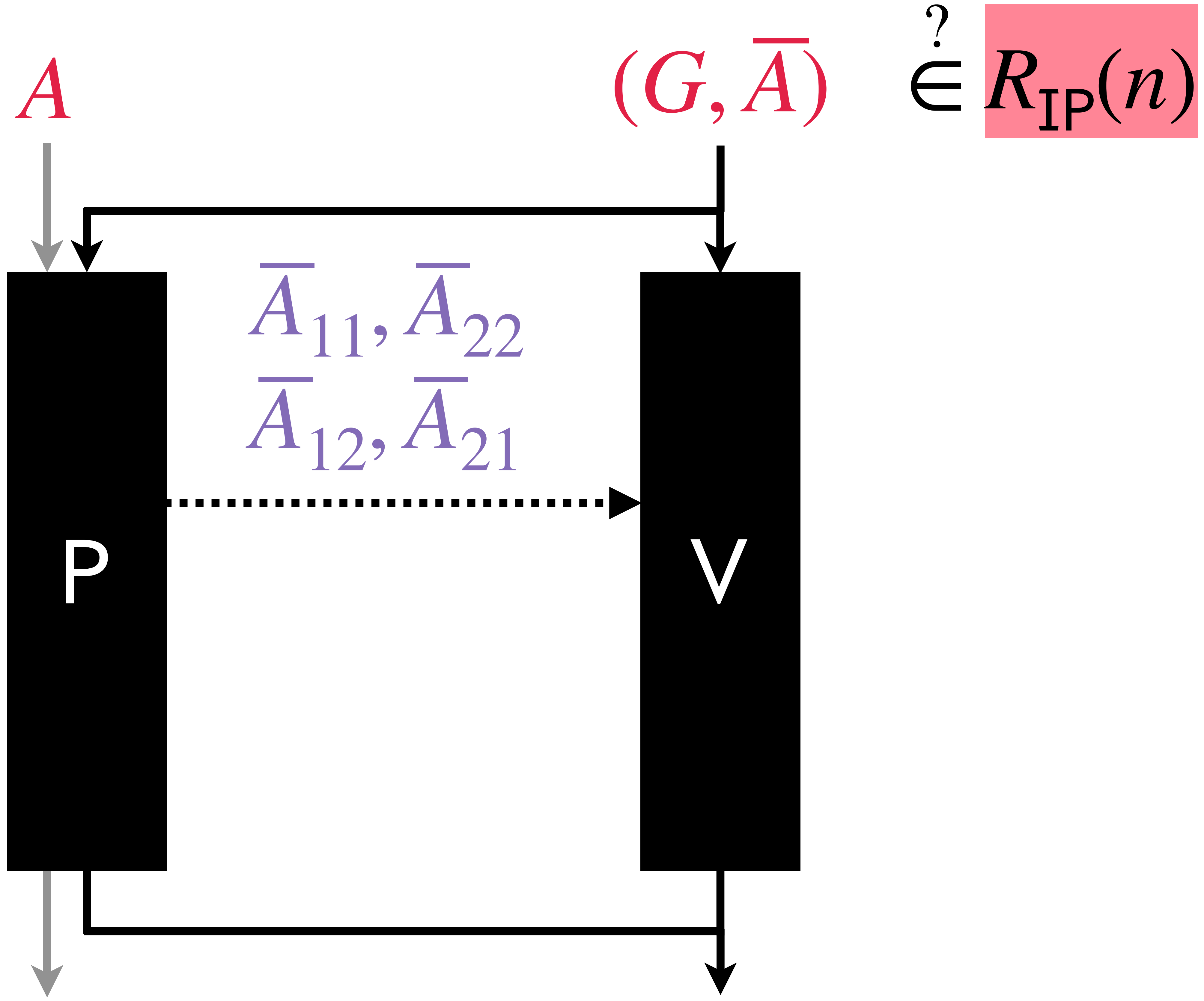
$\overline{A}_{ij} \leftarrow G_i(A_j)$ for $i, j \in \{1,2\}$



First Example: Inner-Product Reduction [BCCGP16]

There exists a reduction of knowledge from $R_{\mathtt{IP}}(n)$ to $R_{\mathtt{IP}}(n/2)$.

$\bar{A}_{ij} \leftarrow G_i(A_j)$ for $i, j \in \{1,2\}$

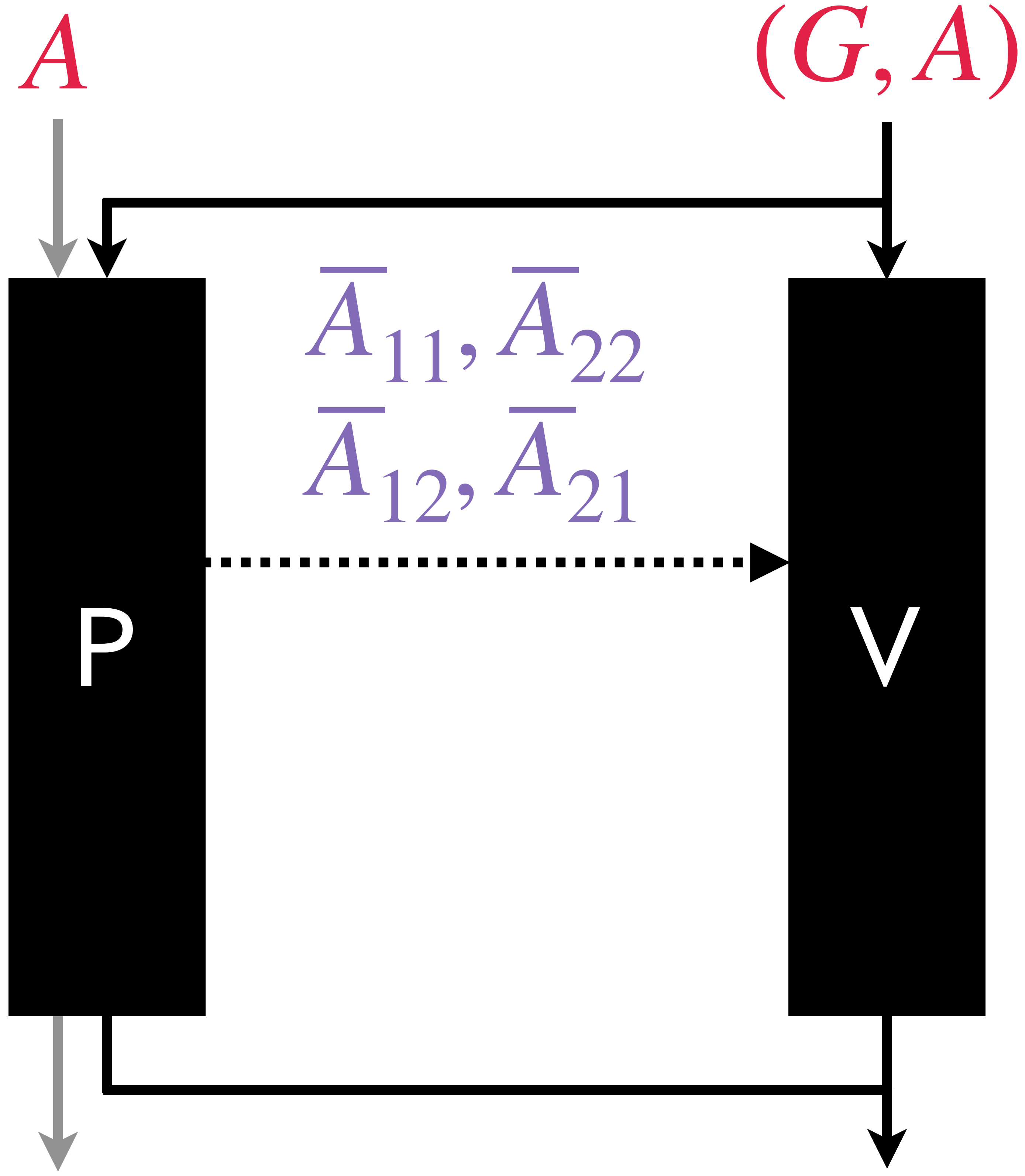


First Example: Inner-Product Reduction [BCCGP16]

There exists a reduction of knowledge from $R_{\mathrm{IP}}(n)$ to $R_{\mathrm{IP}}(n/2)$.

$(G, \bar{A}) \stackrel{?}{\in} R_{\mathrm{IP}}(n)$

$\bar{A}_{ij} \leftarrow G_i(A_j)$ for $i, j \in \{1,2\}$



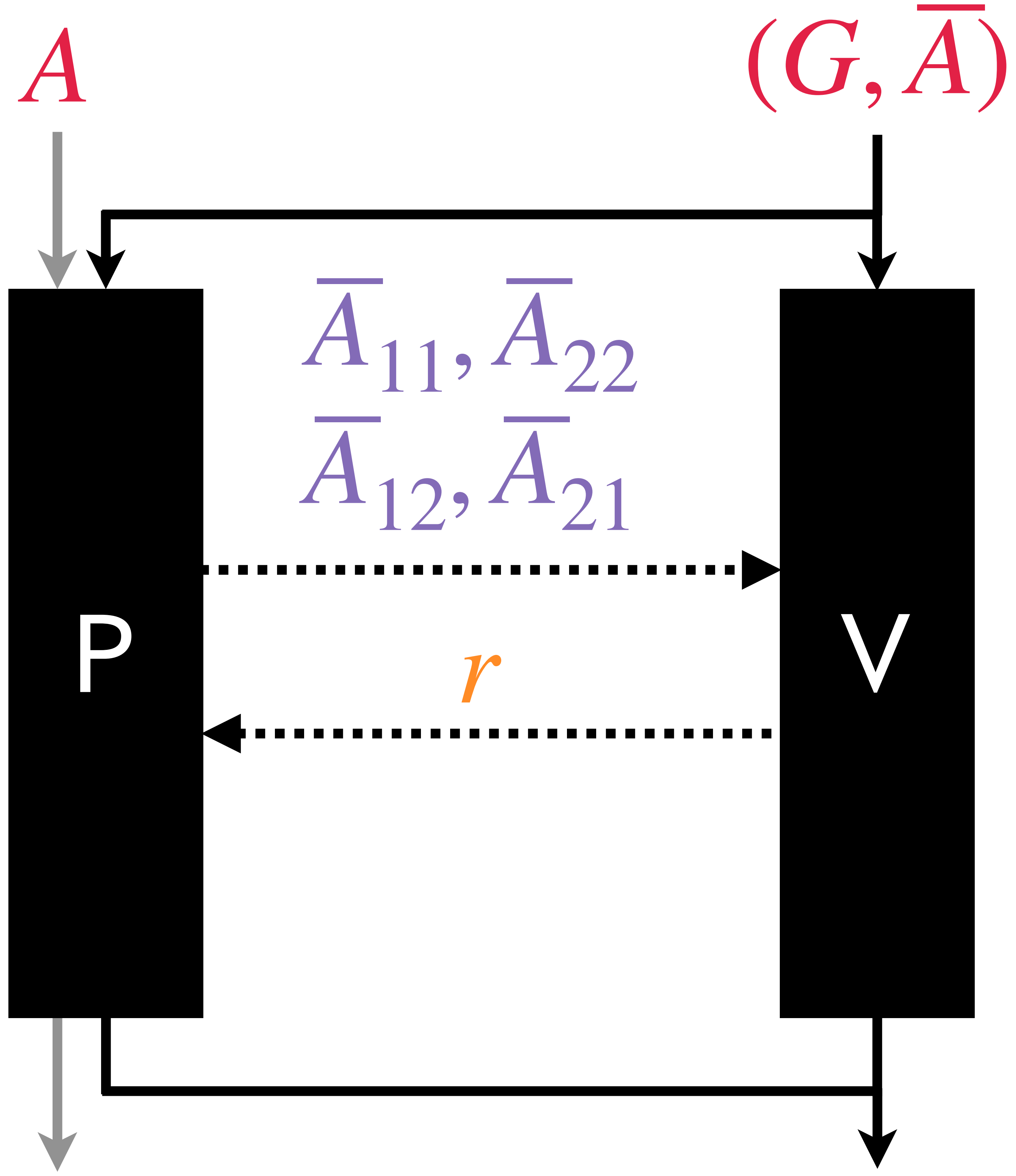
$r \xleftarrow{\$} \mathbb{F}$

First Example: Inner-Product Reduction [BCCGP16]

There exists a reduction of knowledge from $R_{\mathsf{IP}}(n)$ to $R_{\mathsf{IP}}(n/2)$.

$\overset{?}{\in} R_{\mathsf{IP}}(n)$

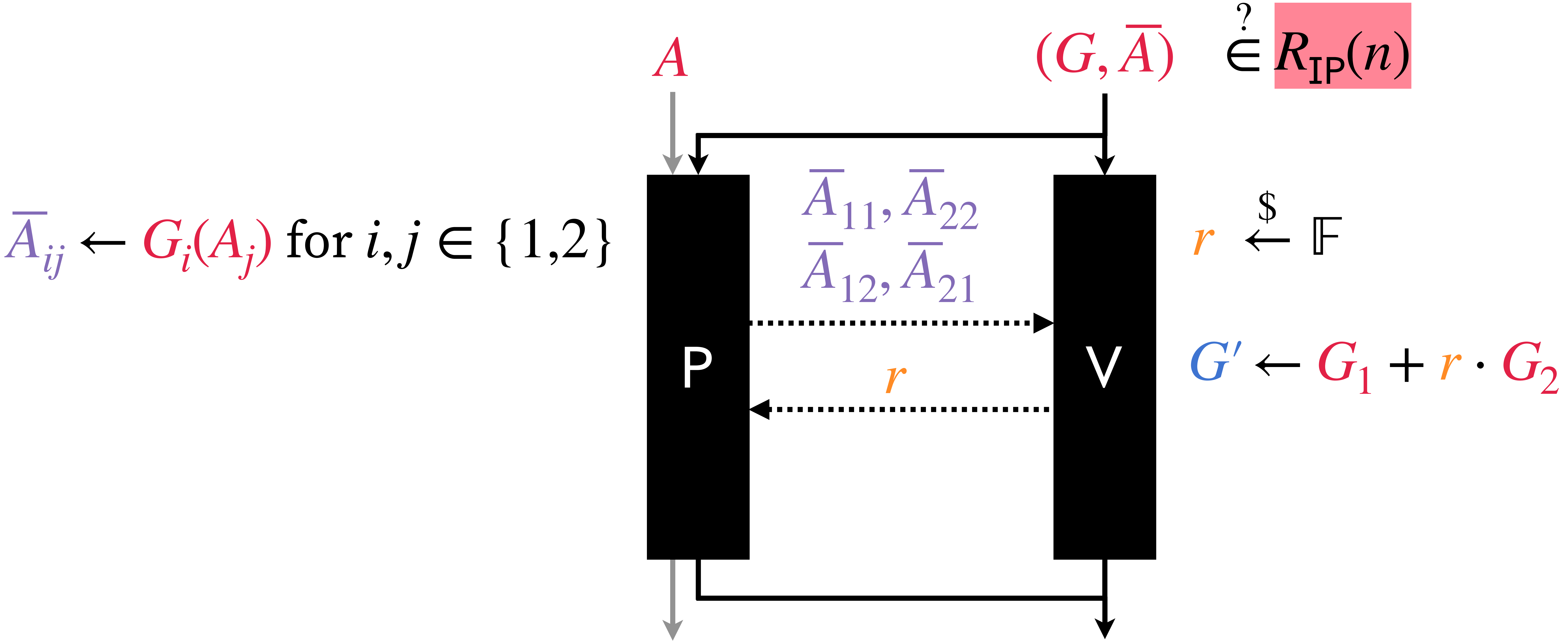
$\bar{A}_{ij} \leftarrow G_i(A_j)$ for $i, j \in \{1,2\}$



$r \overset{\$}{\leftarrow} \mathbb{F}$

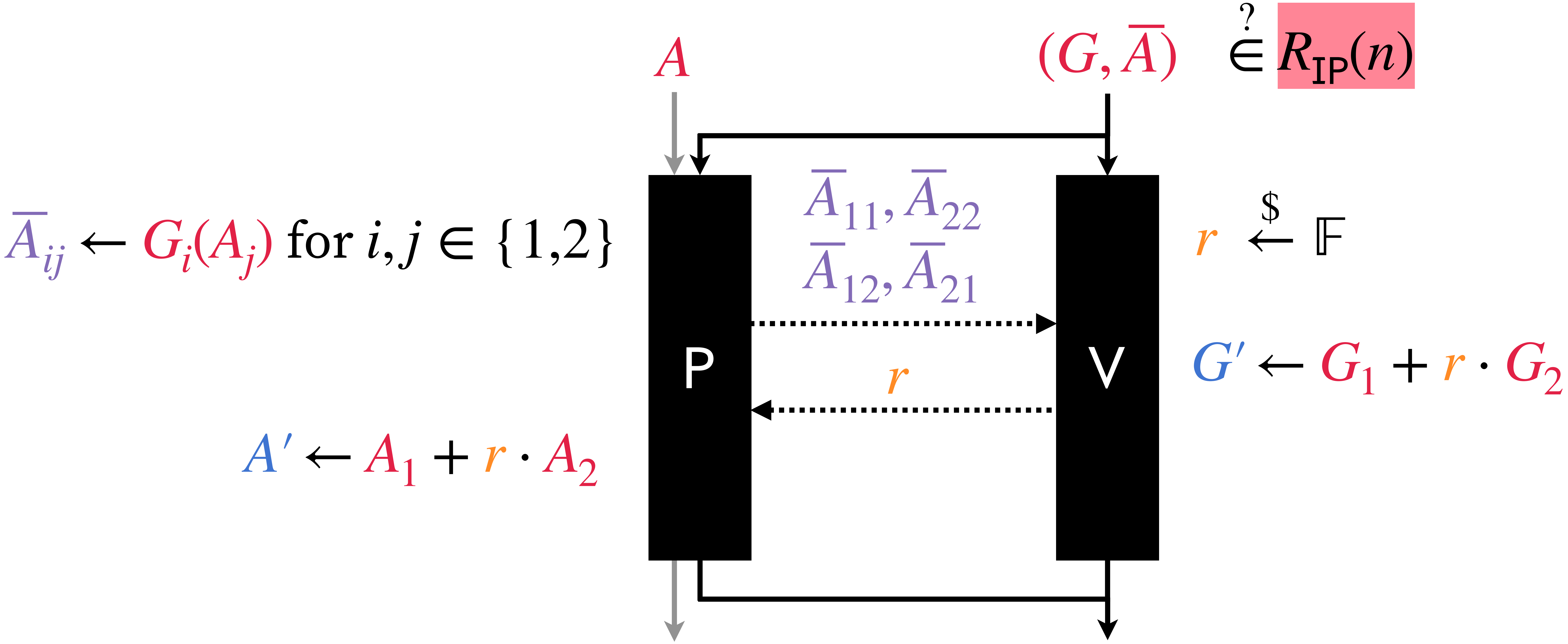
First Example: Inner-Product Reduction [BCCGP16]

There exists a reduction of knowledge from $R_{\mathrm{IP}}(n)$ to $R_{\mathrm{IP}}(n/2)$.



First Example: Inner-Product Reduction [BCCGP16]

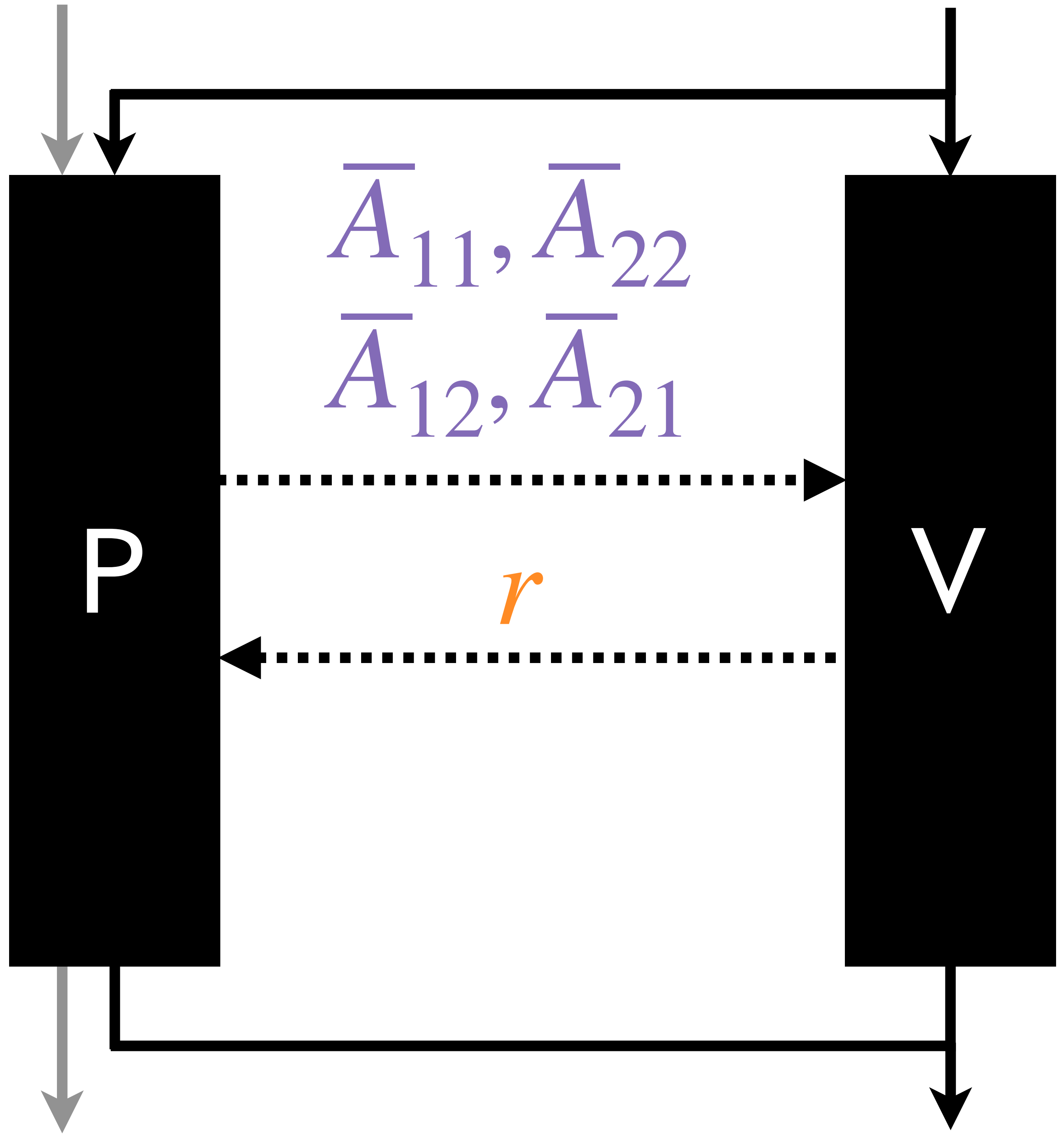
There exists a reduction of knowledge from $R_{\mathtt{IP}}(n)$ to $R_{\mathtt{IP}}(n/2)$.



First Example: Inner-Product Reduction [BCCGP16]

There exists a reduction of knowledge from $R_{\mathrm{IP}}(n)$ to $R_{\mathrm{IP}}(n/2)$.

A $\qquad$ $(G, \overline{A}) \stackrel{?}{\in} R_{\mathrm{IP}}(n)$



$\overline{A}_{ij} \leftarrow G_i(A_j)$ for $i, j \in \{1,2\}$

$r \xleftarrow{\$} \mathbb{F}$

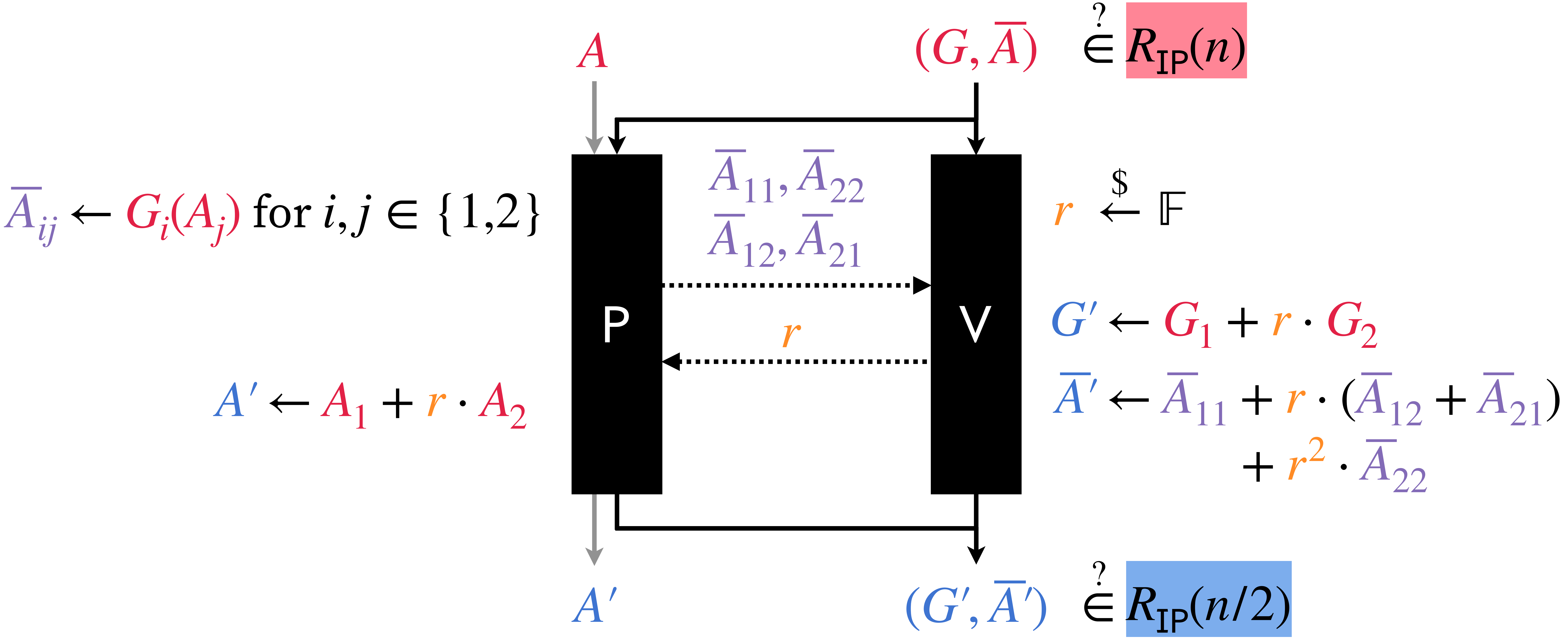
$G' \leftarrow G_1 + r \cdot G_2$

$A' \leftarrow A_1 + r \cdot A_2$

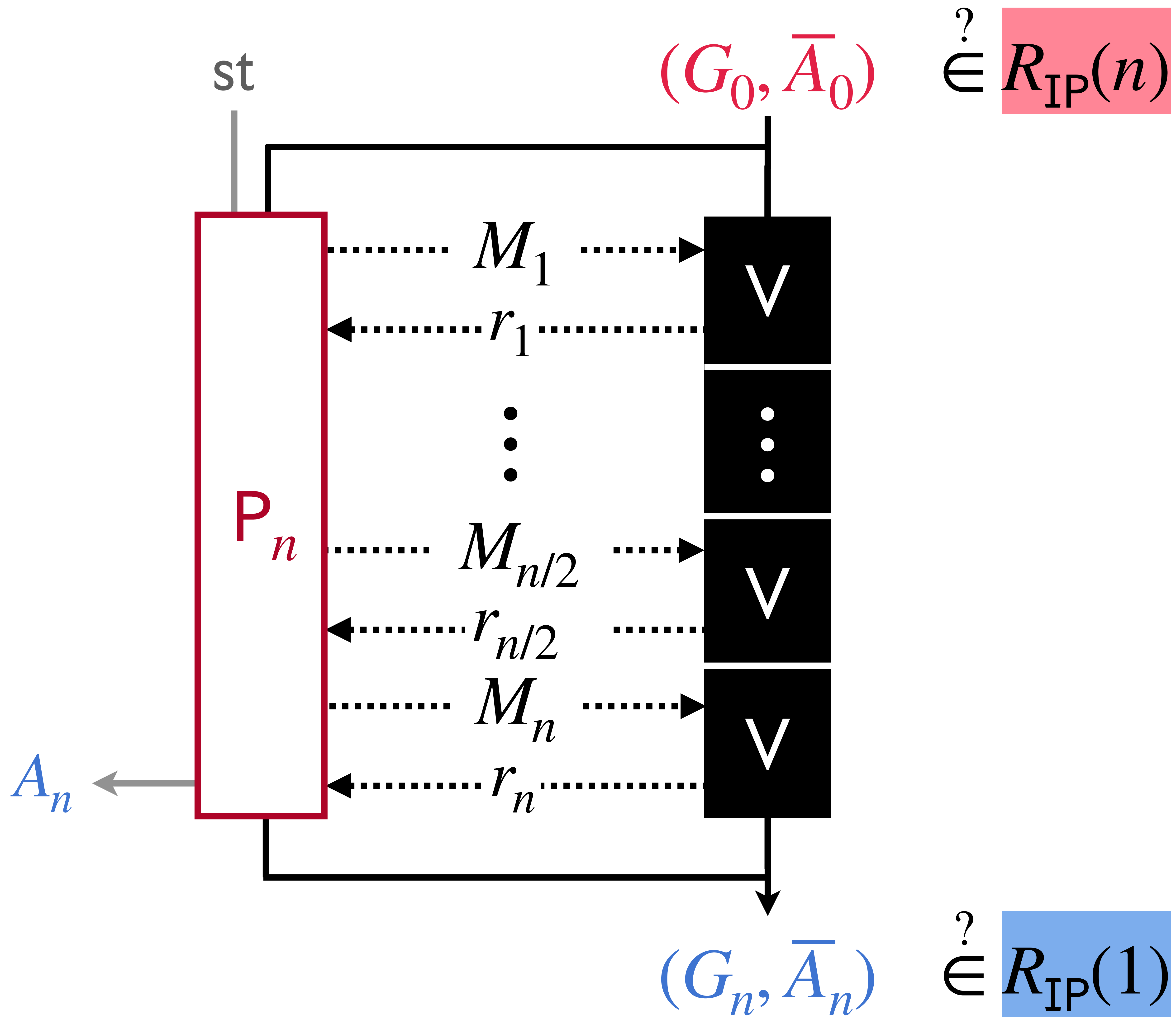
$$\overline{A}' \leftarrow \overline{A}_{11} + r \cdot (\overline{A}_{12} + \overline{A}_{21}) + r^2 \cdot \overline{A}_{22}$$

First Example: Inner-Product Reduction [BCCGP16]

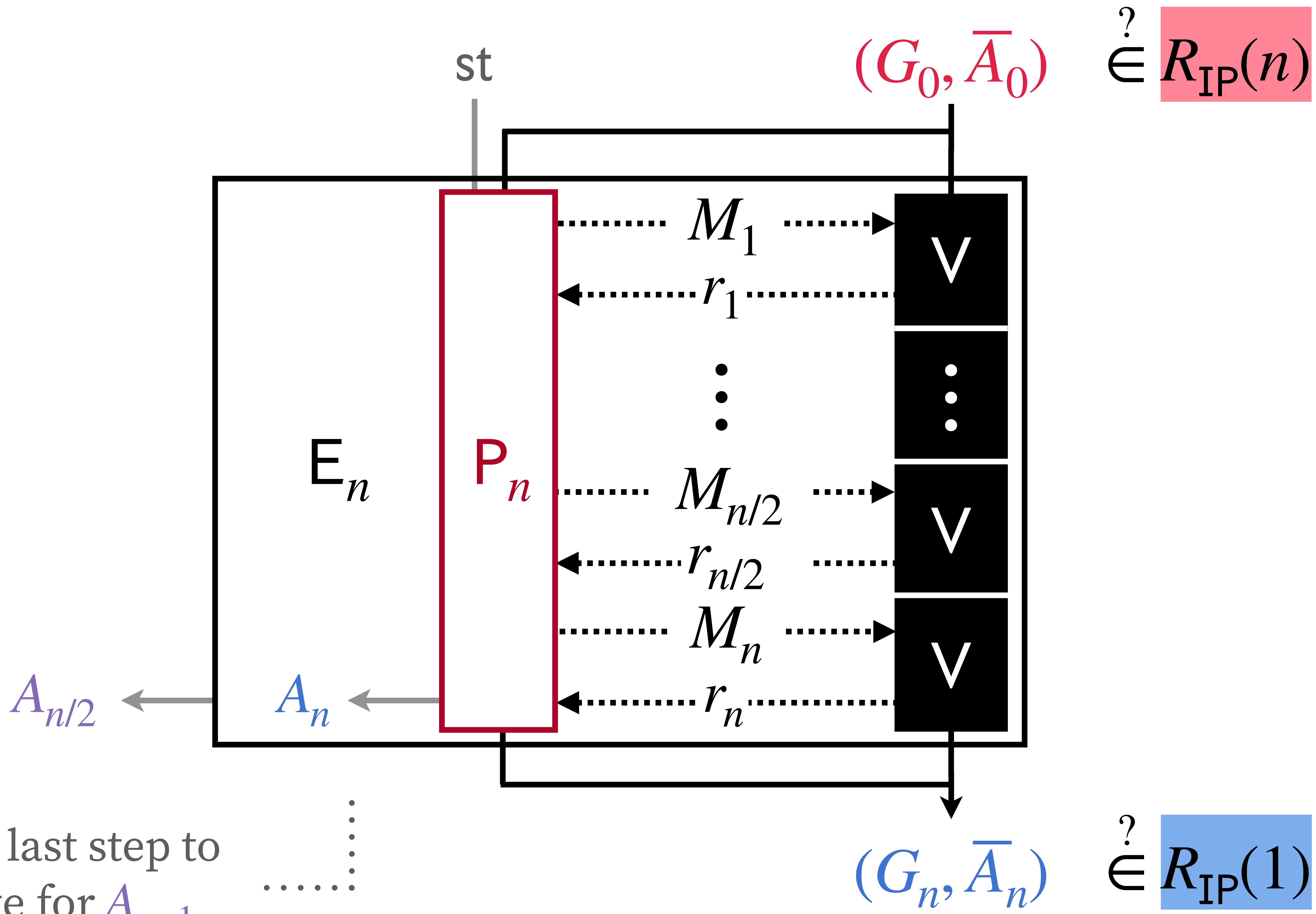
There exists a reduction of knowledge from $R_{\mathtt{IP}}(n)$ to $R_{\mathtt{IP}}(n/2)$.



Problem: Simple Construction, Complex Proof

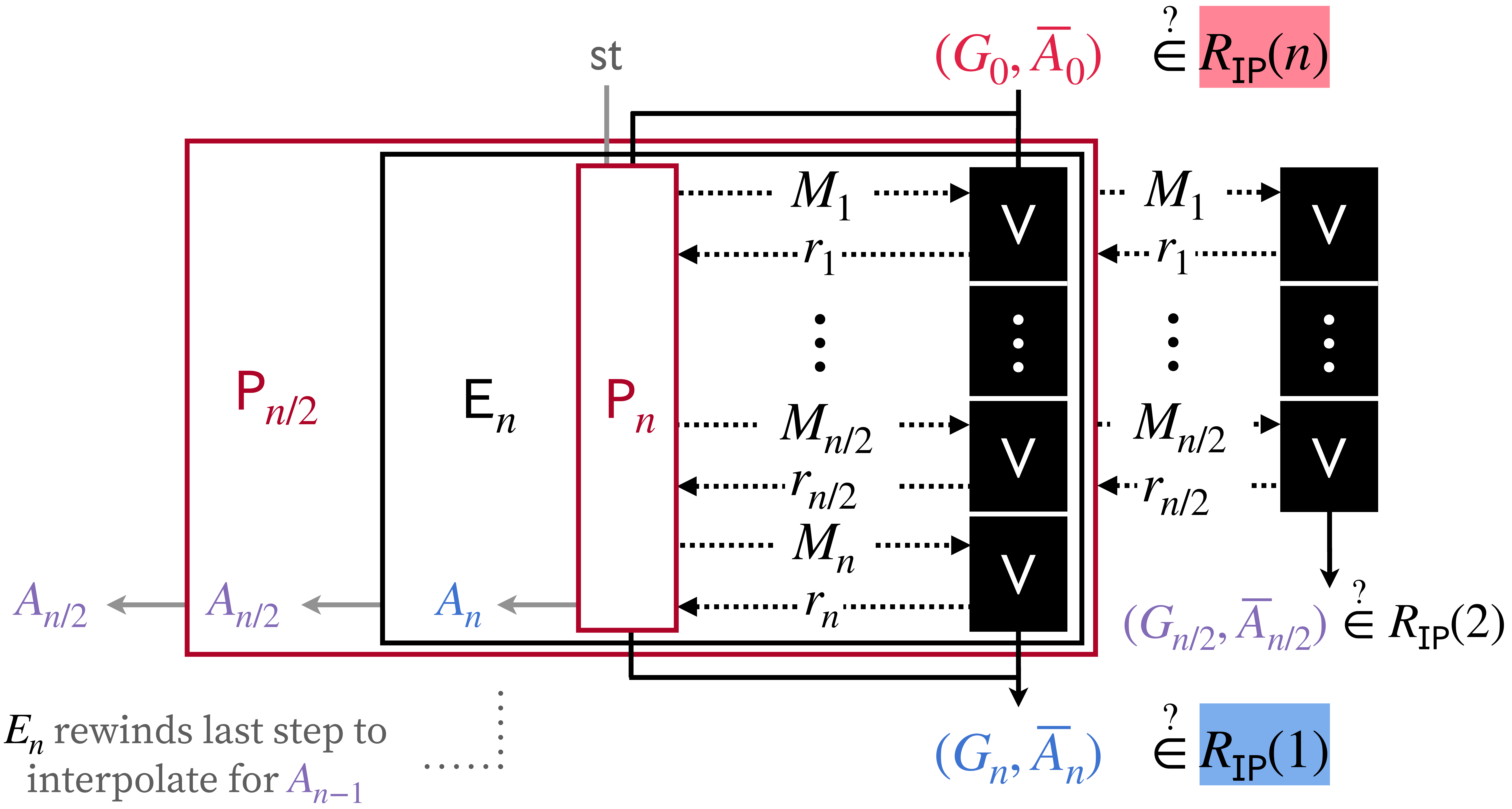


Problem: Simple Construction, Complex Proof

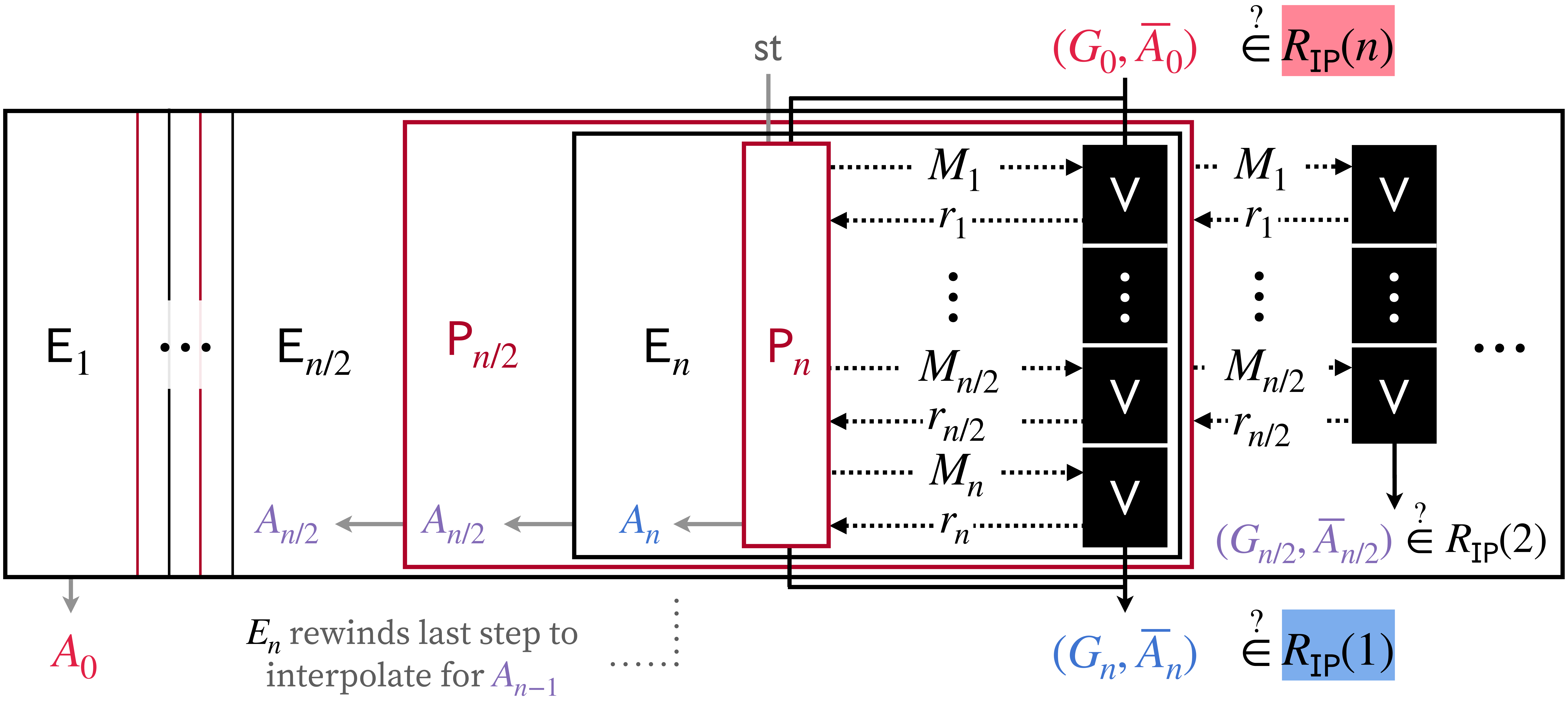


E_n rewinds last step to interpolate for A_{n-1}

Problem: Simple Construction, Complex Proof



Problem: Simple Construction, Complex Proof



Solution: Sequential Composition Theorem

We prove that reductions are sequentially composable.

Solution: Sequential Composition Theorem

We prove that reductions are sequentially composable.

$$\Pi_1 : R_1 \to R_2$$

Solution: Sequential Composition Theorem

We prove that reductions are sequentially composable.

$\Pi_1 : R_1 \to R_2$

$\Pi_2 : R_2 \to R_3$

Solution: Sequential Composition Theorem

We prove that reductions are sequentially composable.

$$\Pi_1 : R_1 \to R_2 \quad \circ \quad \Pi_2 : R_2 \to R_3$$

Solution: Sequential Composition Theorem

We prove that reductions are sequentially composable.

$$\Pi_1 : R_1 \to R_2 \quad \circ \quad \Pi_2 : R_2 \to R_3$$

$$\Pi_2 \circ \Pi_1 : R_1 \to R_3$$

Inner-Product Argument with a Simple Proof

Simpler soundness proof: Invoke sequential composition.

Inner-Product Argument with a Simple Proof

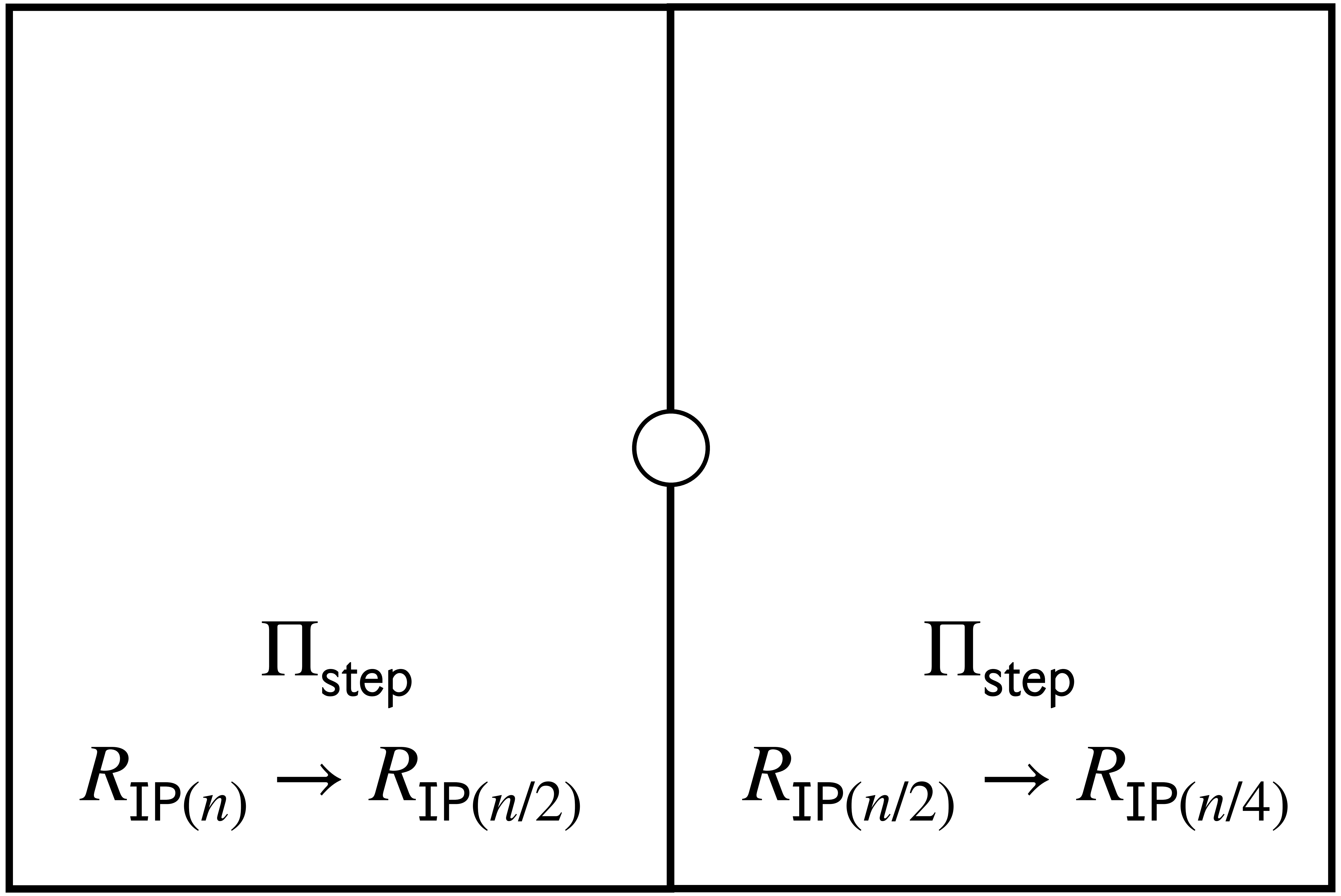
Simpler soundness proof: Invoke sequential composition.

$$\Pi_{\mathsf{step}}$$

$$R_{\mathsf{IP}(n)} \to R_{\mathsf{IP}(n/2)}$$

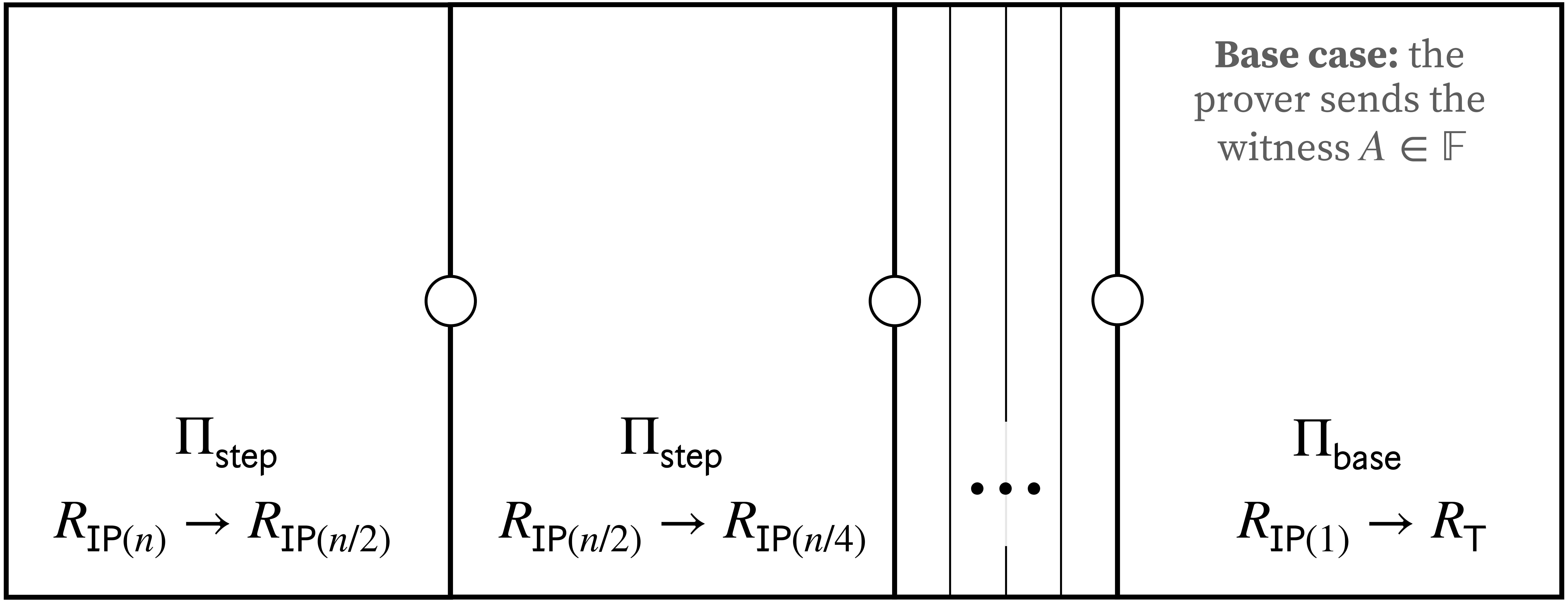
Inner-Product Argument with a Simple Proof

Simpler soundness proof: Invoke sequential composition.



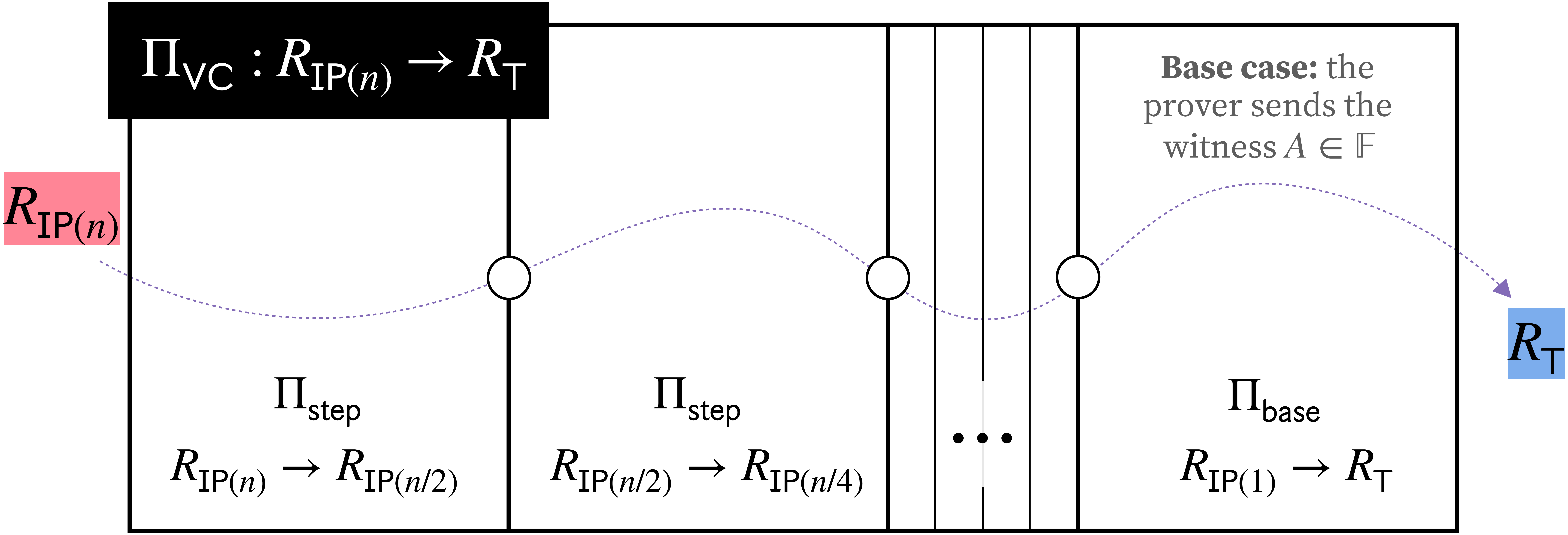
Inner-Product Argument with a Simple Proof

Simpler soundness proof: Invoke sequential composition.



Inner-Product Argument with a Simple Proof

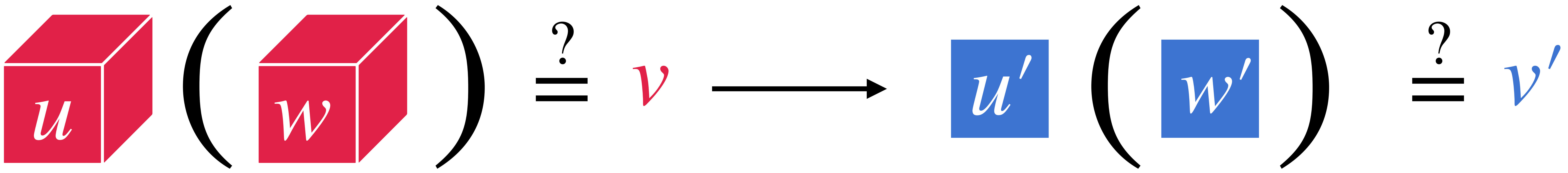
Simpler soundness proof: Invoke sequential composition.



Our Generalization: Tensor Reduction of Knowledge

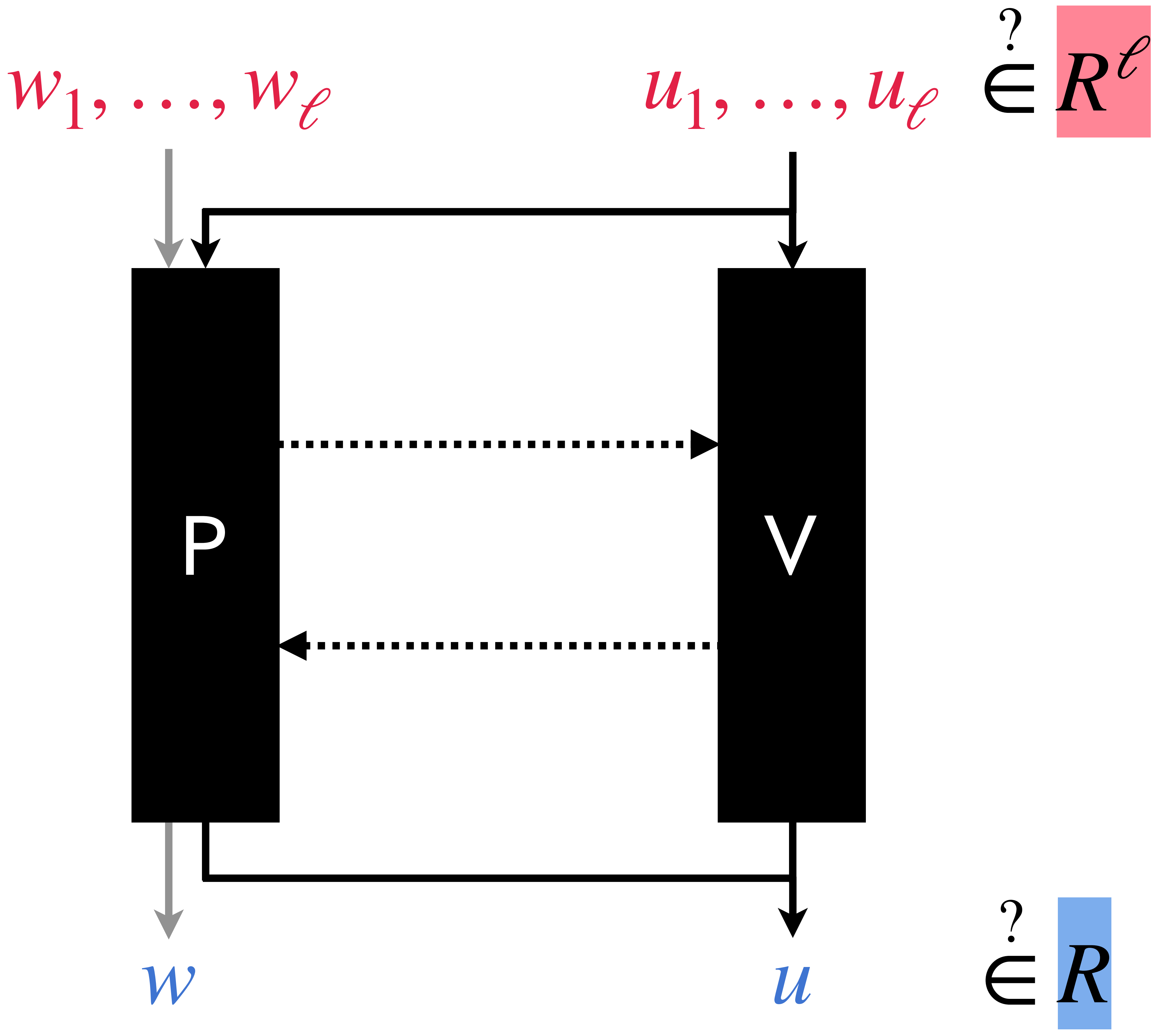
This generalizes techniques in [BCCGP16], [BBBPWM18], [BCS21], [BMMTV21], [AC20], and [ACR21]

Theorem. There exists a reduction of knowledge that reduces the task of checking knowledge of w such that $u(w) = v$ for $u \in \hom(W^n, V)$ to the task of checking knowledge of w' such that $u'(w') = v'$ for $u' \in \hom(W, V)$.



Second Example: Folding Schemes

An ℓ-**folding scheme** is a reduction of knowledge from $R^\ell = R \times \cdots \times R$ to R.



Problem: Simple Construction, Complex Proof [RZ22]

Consider a 2-folding scheme $\Pi_2 : R^2 \rightarrow R$

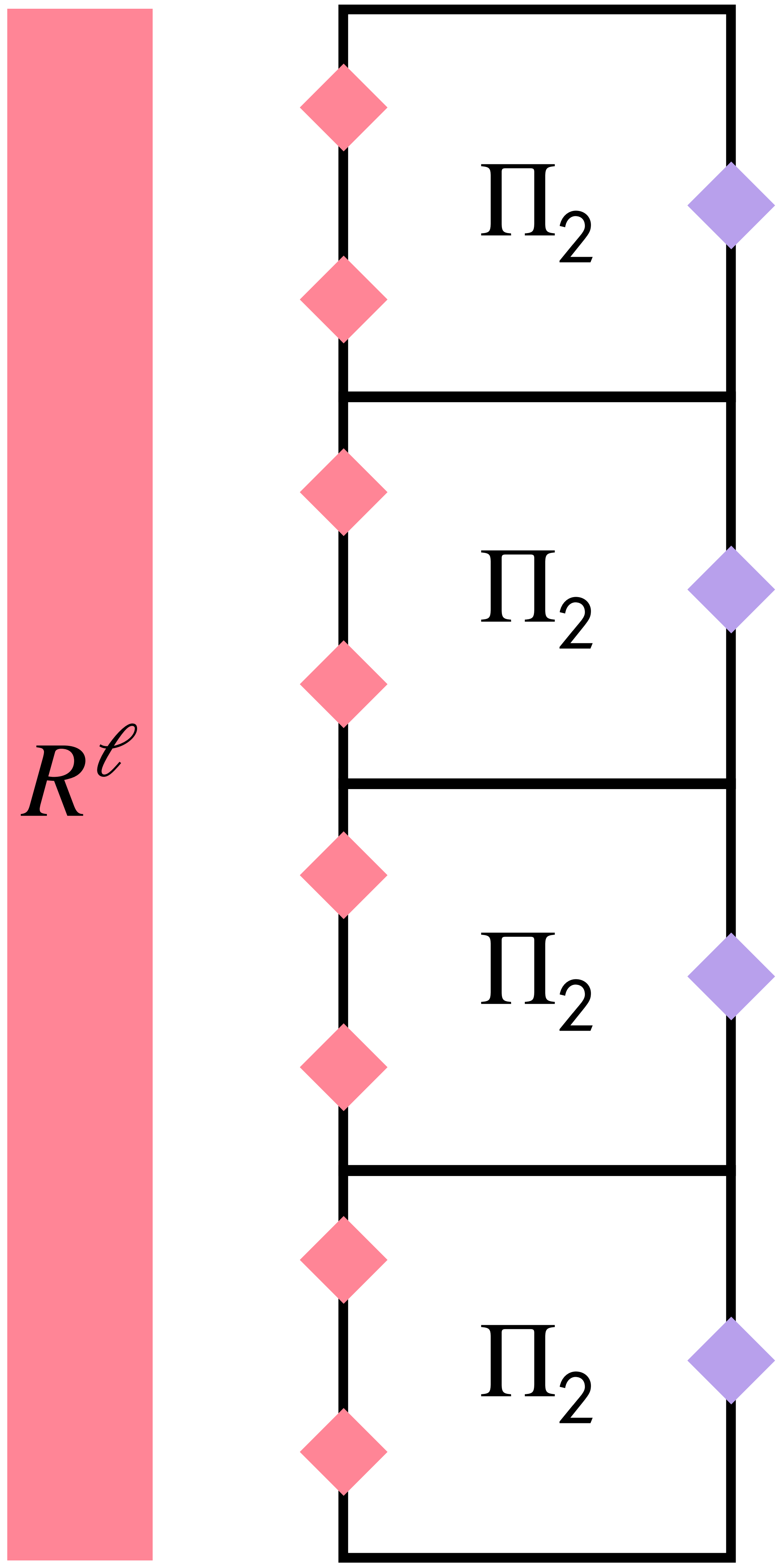
Problem: Simple Construction, Complex Proof [RZ22]

Consider a 2-folding scheme $\Pi_2 : R^2 \to R$

R^ℓ

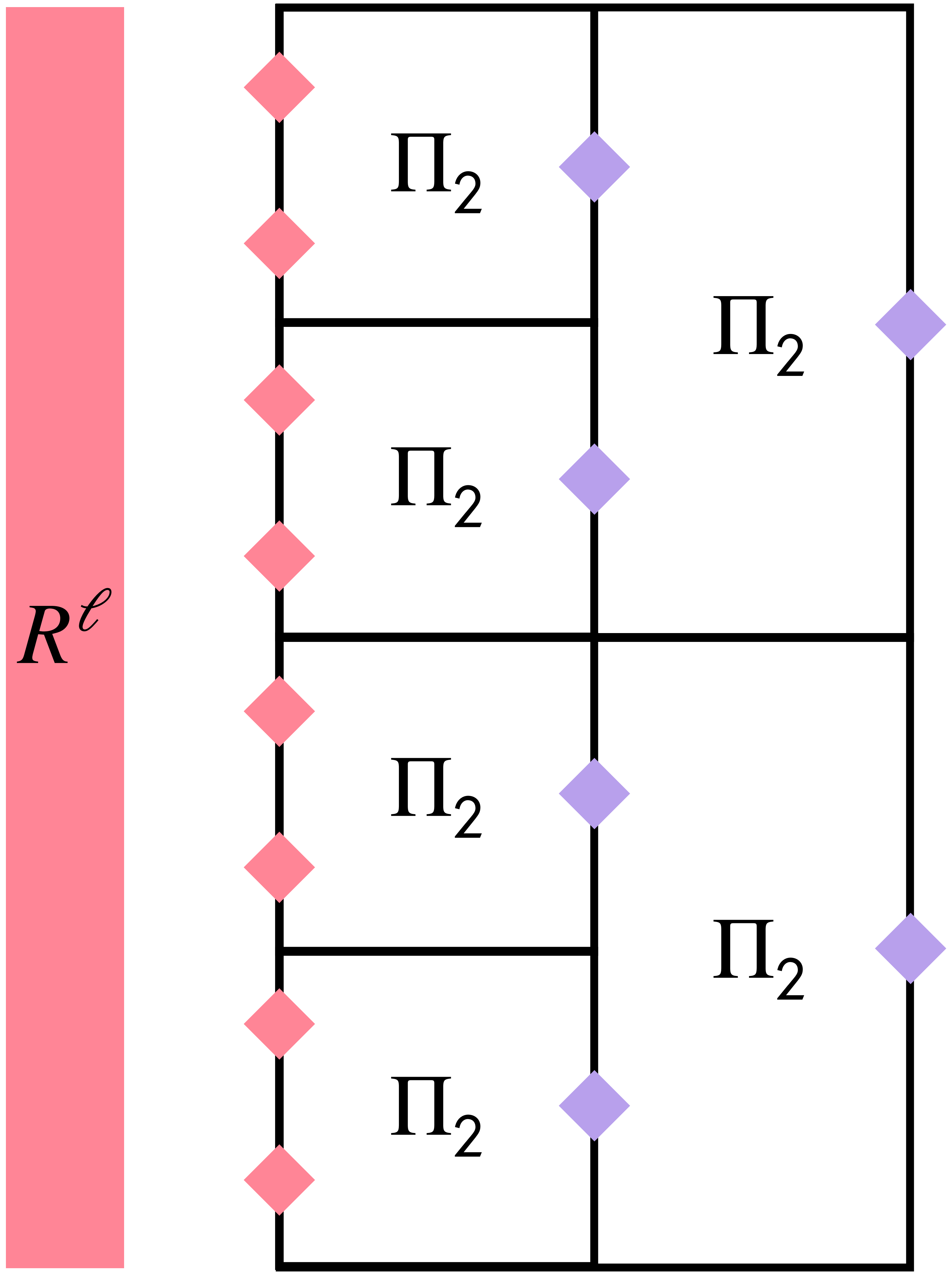
Problem: Simple Construction, Complex Proof [RZ22]

Consider a 2-folding scheme $\Pi_2 : R^2 \to R$



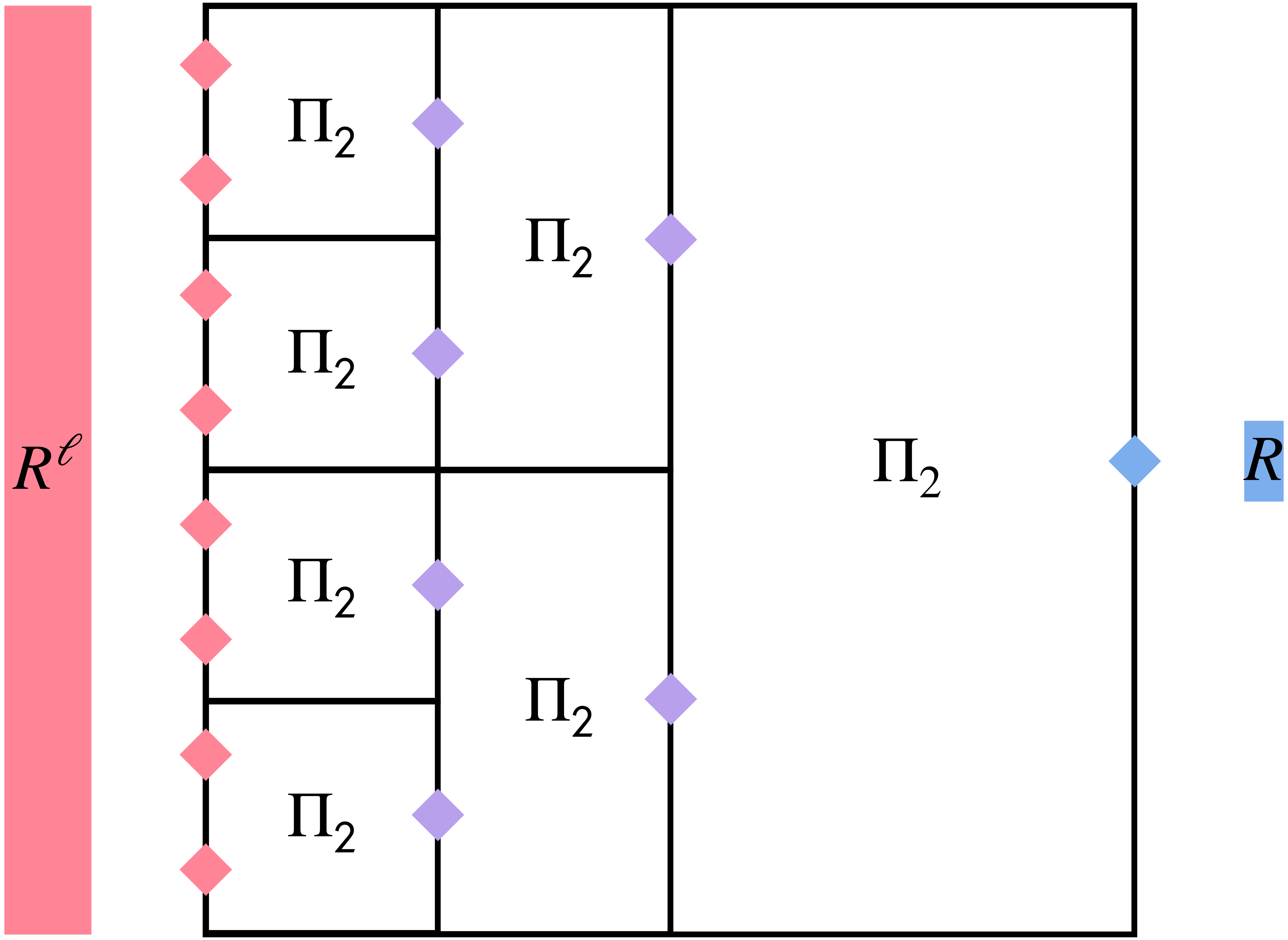
Problem: Simple Construction, Complex Proof [RZ22]

Consider a 2-folding scheme $\Pi_2 : R^2 \to R$

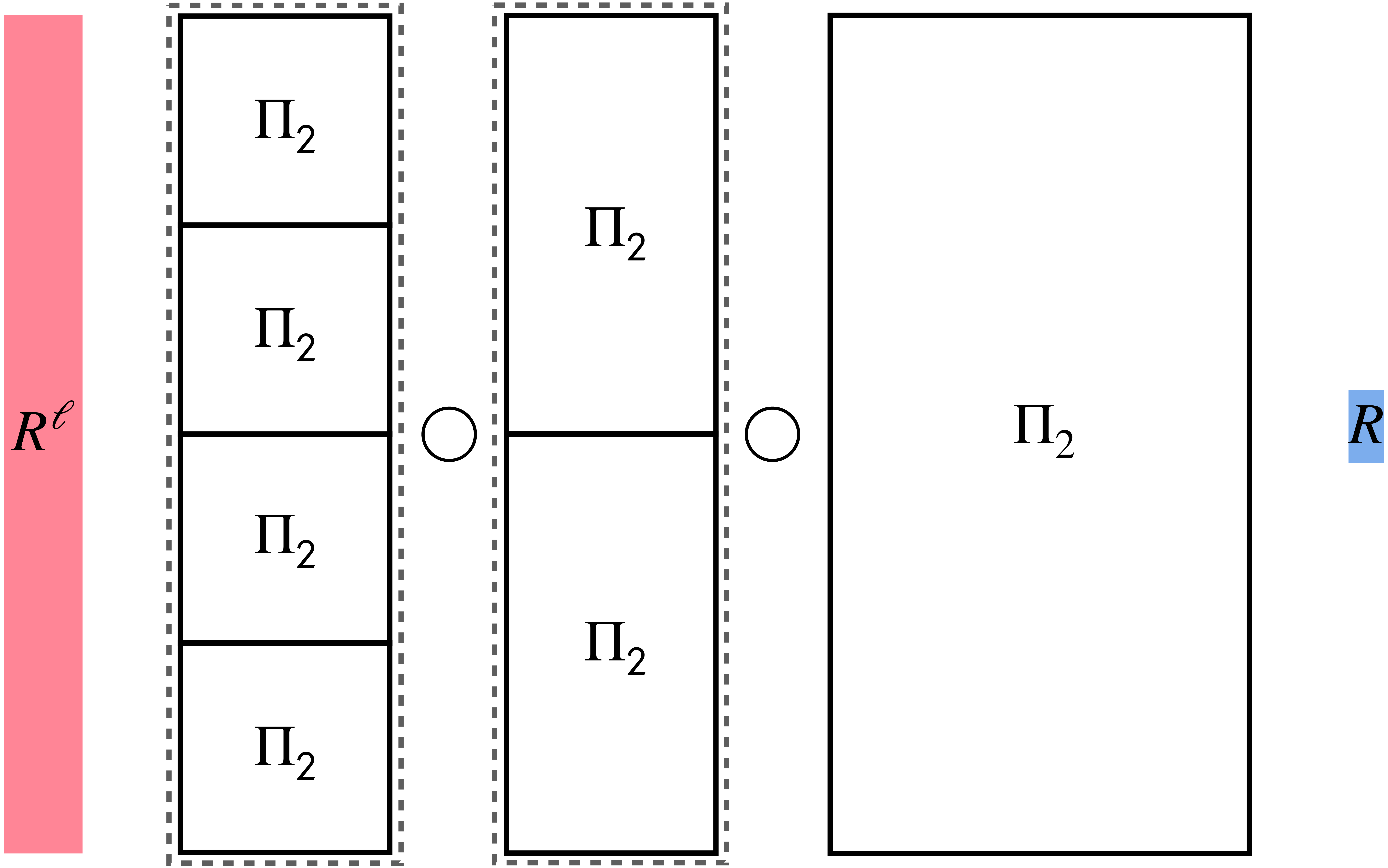


Problem: Simple Construction, Complex Proof [RZ22]

Consider a 2-folding scheme $\Pi_2 : R^2 \to R$



Problem: Simple Construction, Complex Proof [RZ22]



Solution: Parallel Composition Theorem

We prove that reductions can be composed in parallel.

Solution: Parallel Composition Theorem

We prove that reductions can be composed in parallel.

$$\Pi_1 : R_1 \rightarrow R_2$$

Solution: Parallel Composition Theorem

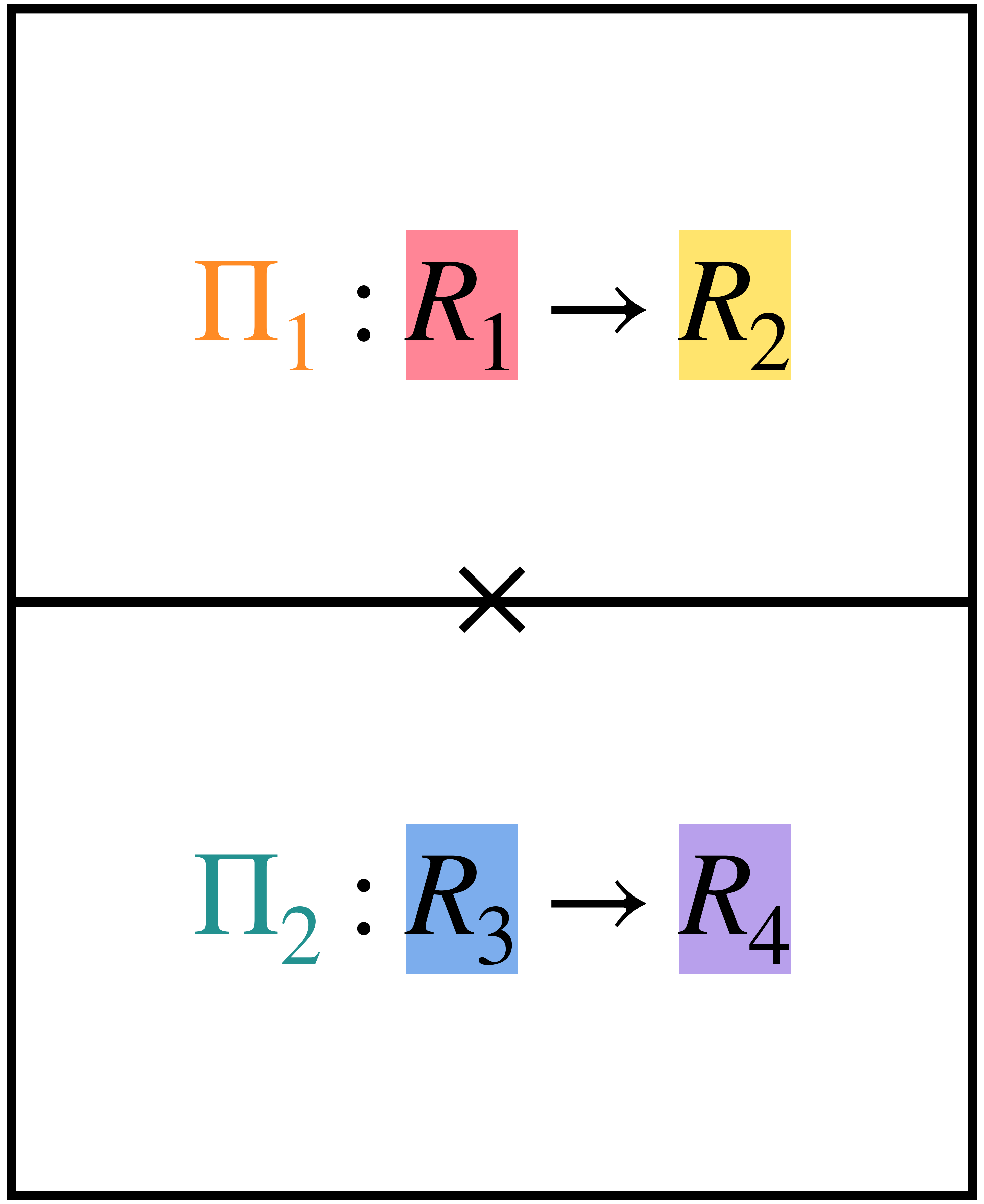
We prove that reductions can be composed in parallel.

$$\Pi_1 : R_1 \to R_2$$

$$\Pi_2 : R_3 \to R_4$$

Solution: Parallel Composition Theorem

We prove that reductions can be composed in parallel.



Solution: Parallel Composition Theorem

We prove that reductions can be composed in parallel.

$$\Pi_1 : R_1 \to R_2$$

$$\times$$

$$\Pi_2 : R_3 \to R_4$$

$$\Pi_1 \times \Pi_2 : R_1 \times R_3 \to R_2 \times R_4$$

Solution: Parallel Composition Theorem

We prove that reductions can be composed in parallel.

$$\Pi_1 : R_1 \to R_2$$

$$\times$$

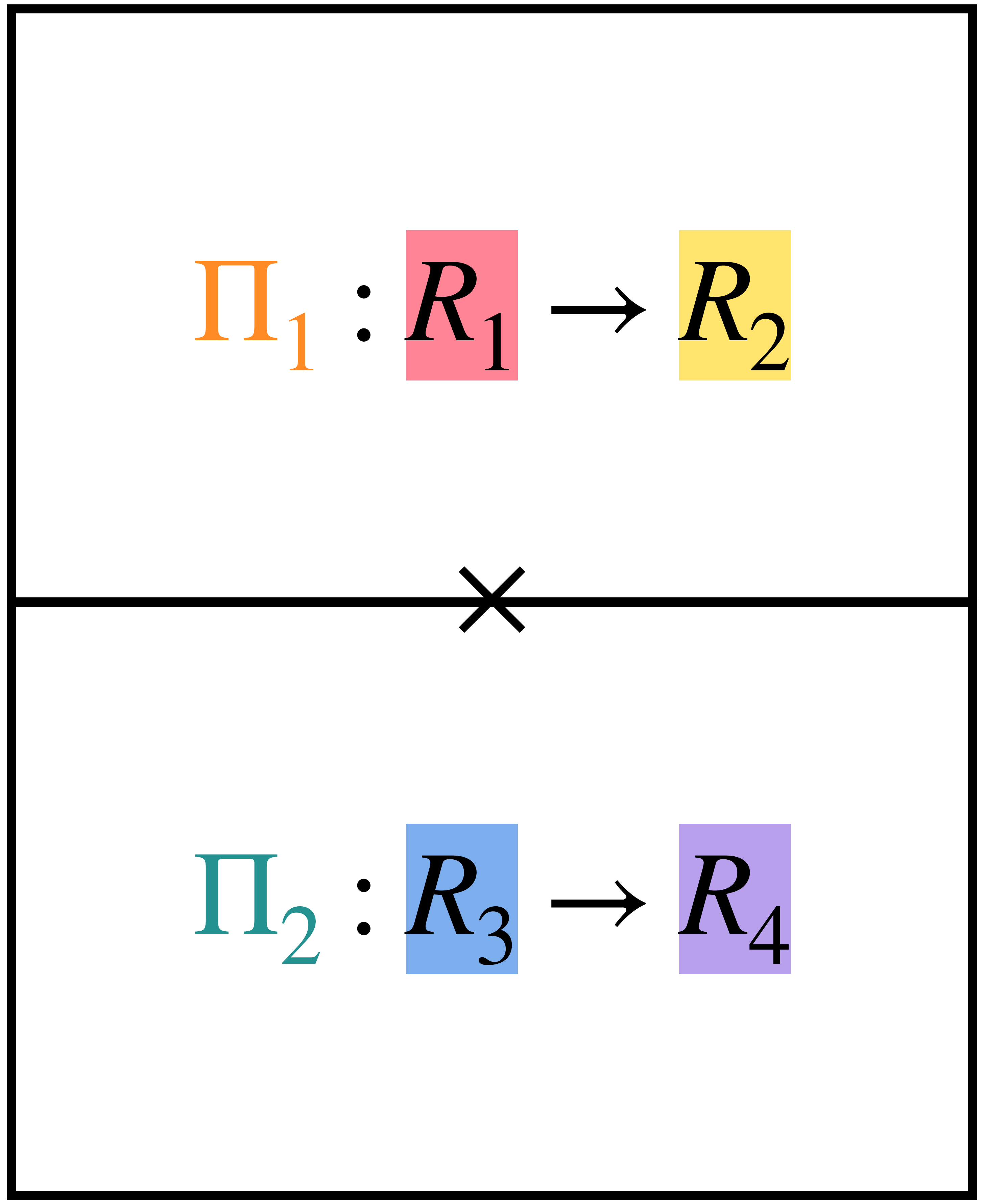
$$\Pi_2 : R_3 \to R_4$$

$$\Pi_1 \times \Pi_2 : R_1 \times R_3 \to R_2 \times R_4$$

$$\left\{ (u_1, u_3), (w_1, w_3) \;\middle|\; \begin{array}{l} (u_1, w_1) \in R_1 \\ (u_3, w_3) \in R_3 \end{array} \right\}$$

Solution: Parallel Composition Theorem

We prove that reductions can be composed in parallel.



$$\left\{ (u_2, u_4), (w_2, w_4) \;\middle|\; \begin{array}{l} (u_2, w_2) \in R_2 \\ (u_4, w_4) \in R_4 \end{array} \right\}$$

$$\Pi_1 \times \Pi_2 : R_1 \times R_3 \to R_2 \times R_4$$

$$\left\{ (u_1, u_3), (w_1, w_3) \;\middle|\; \begin{array}{l} (u_1, w_1) \in R_1 \\ (u_3, w_3) \in R_3 \end{array} \right\}$$

Tree Folding Scheme with a Simple Proof

Given $\Pi_2 : R^2 \to R$, then $\Pi_\ell = \Pi_2 \circ (\Pi_{\ell/2} \times \Pi_{\ell/2}) : R^\ell \to R$.

Tree Folding Scheme with a Simple Proof

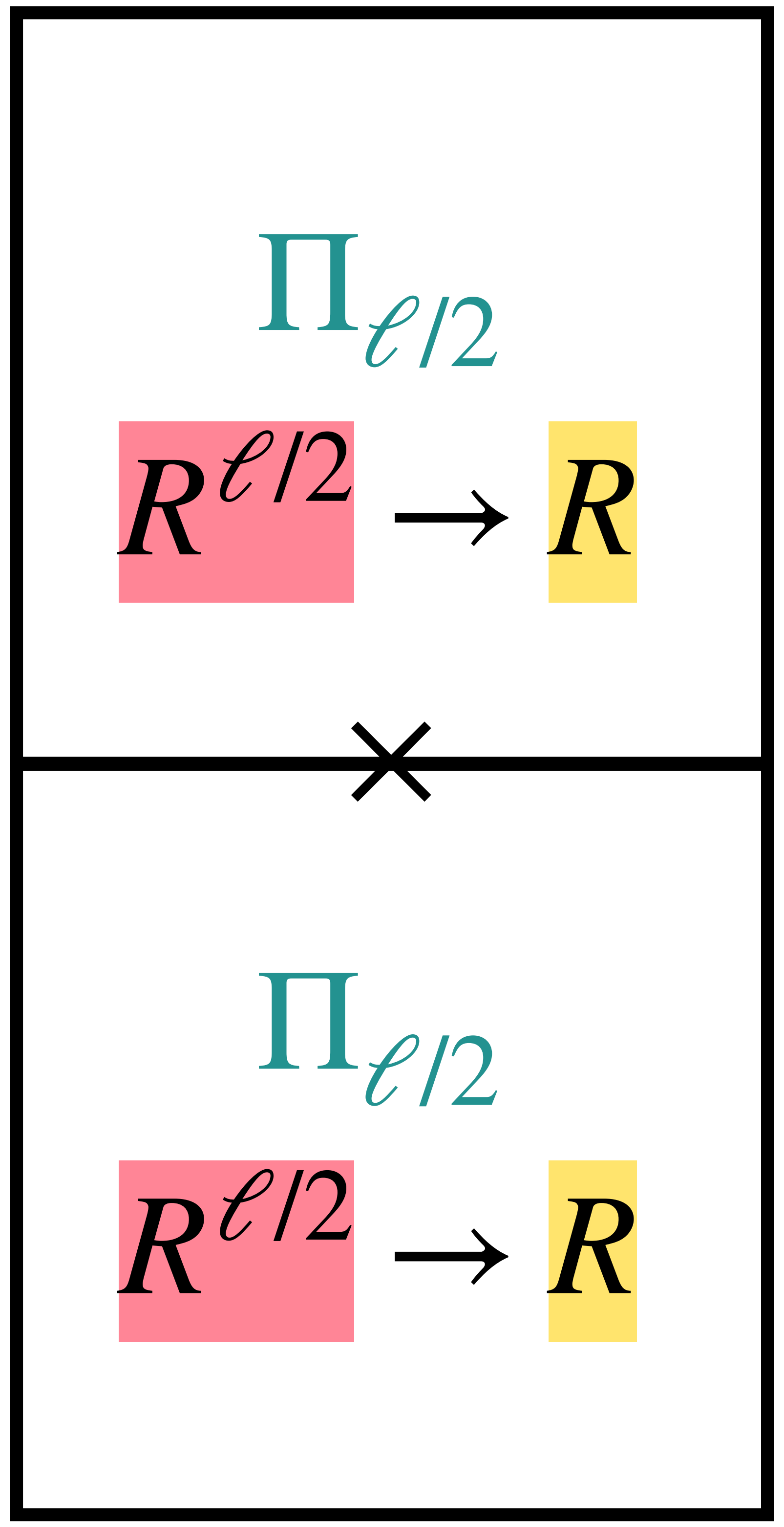
Given $\Pi_2 : R^2 \to R$, then $\Pi_\ell = \Pi_2 \circ (\Pi_{\ell/2} \times \Pi_{\ell/2}) : R^\ell \to R$.

$$\Pi_{\ell/2}$$

$$R^{\ell/2} \to R$$

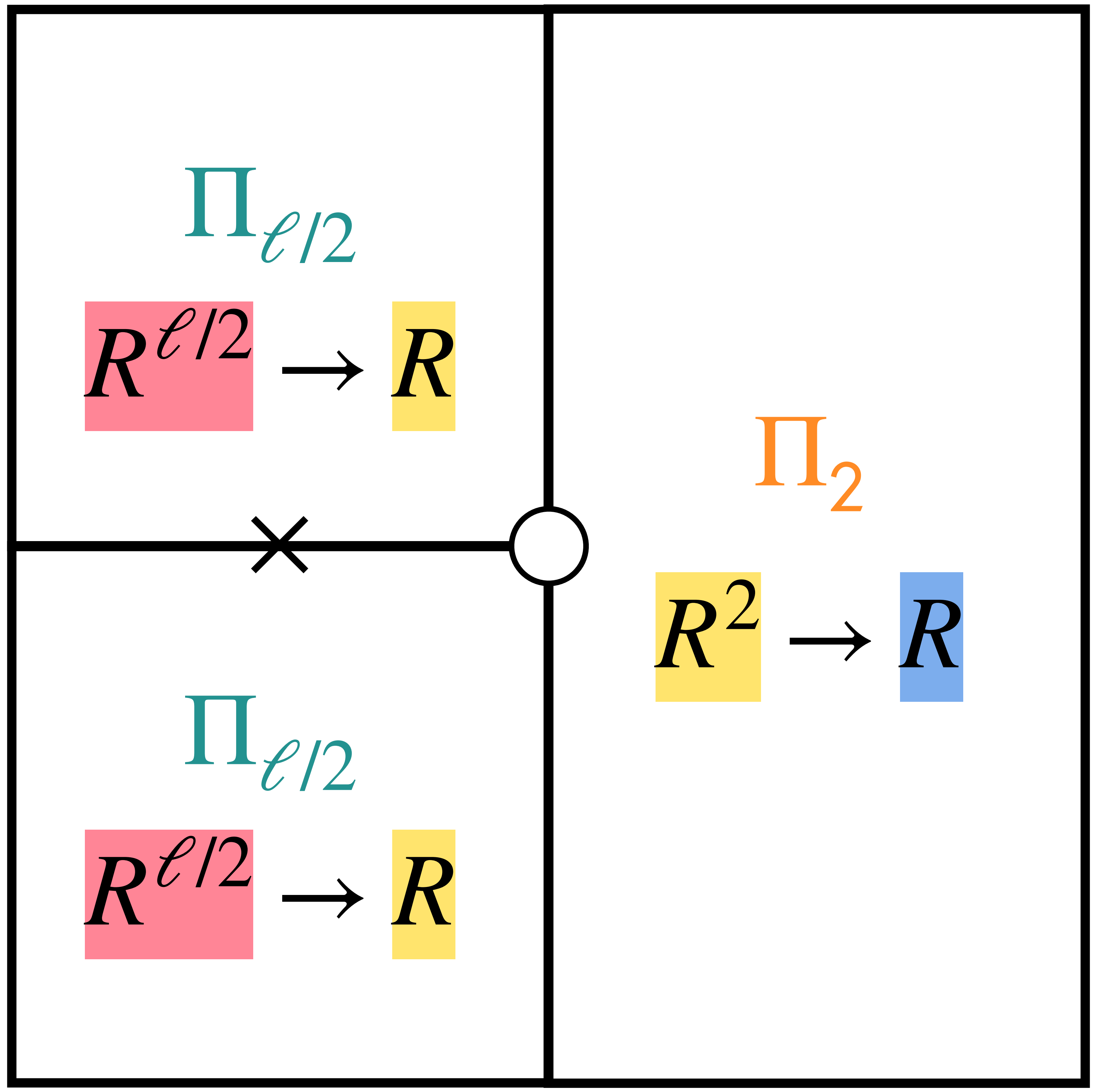
Tree Folding Scheme with a Simple Proof

Given $\Pi_2 : R^2 \to R$, then $\Pi_\ell = \Pi_2 \circ (\Pi_{\ell/2} \times \Pi_{\ell/2}) : R^\ell \to R$.



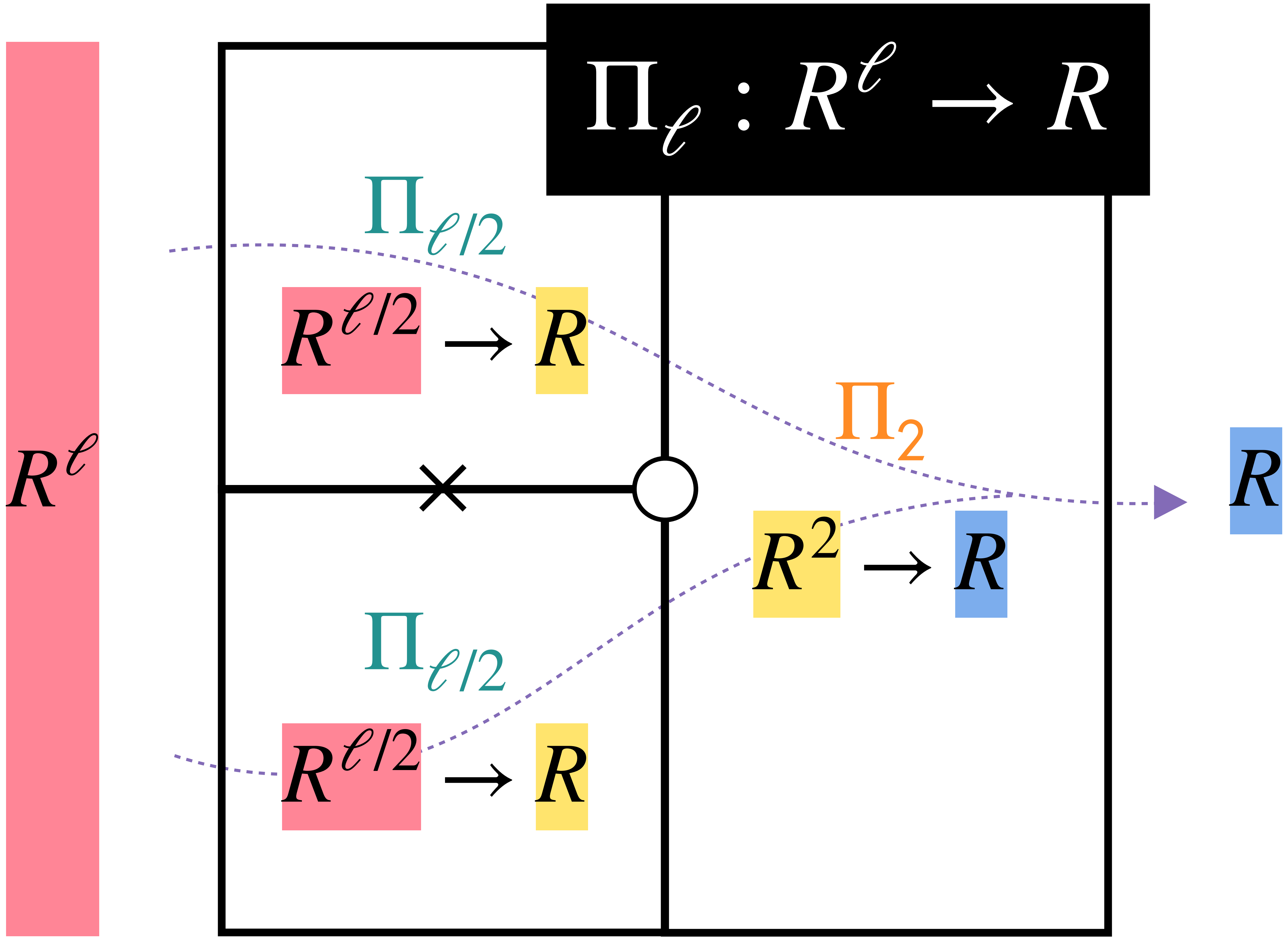
Tree Folding Scheme with a Simple Proof

Given $\Pi_2 : R^2 \to R$, then $\Pi_\ell = \Pi_2 \circ (\Pi_{\ell/2} \times \Pi_{\ell/2}) : R^\ell \to R$.

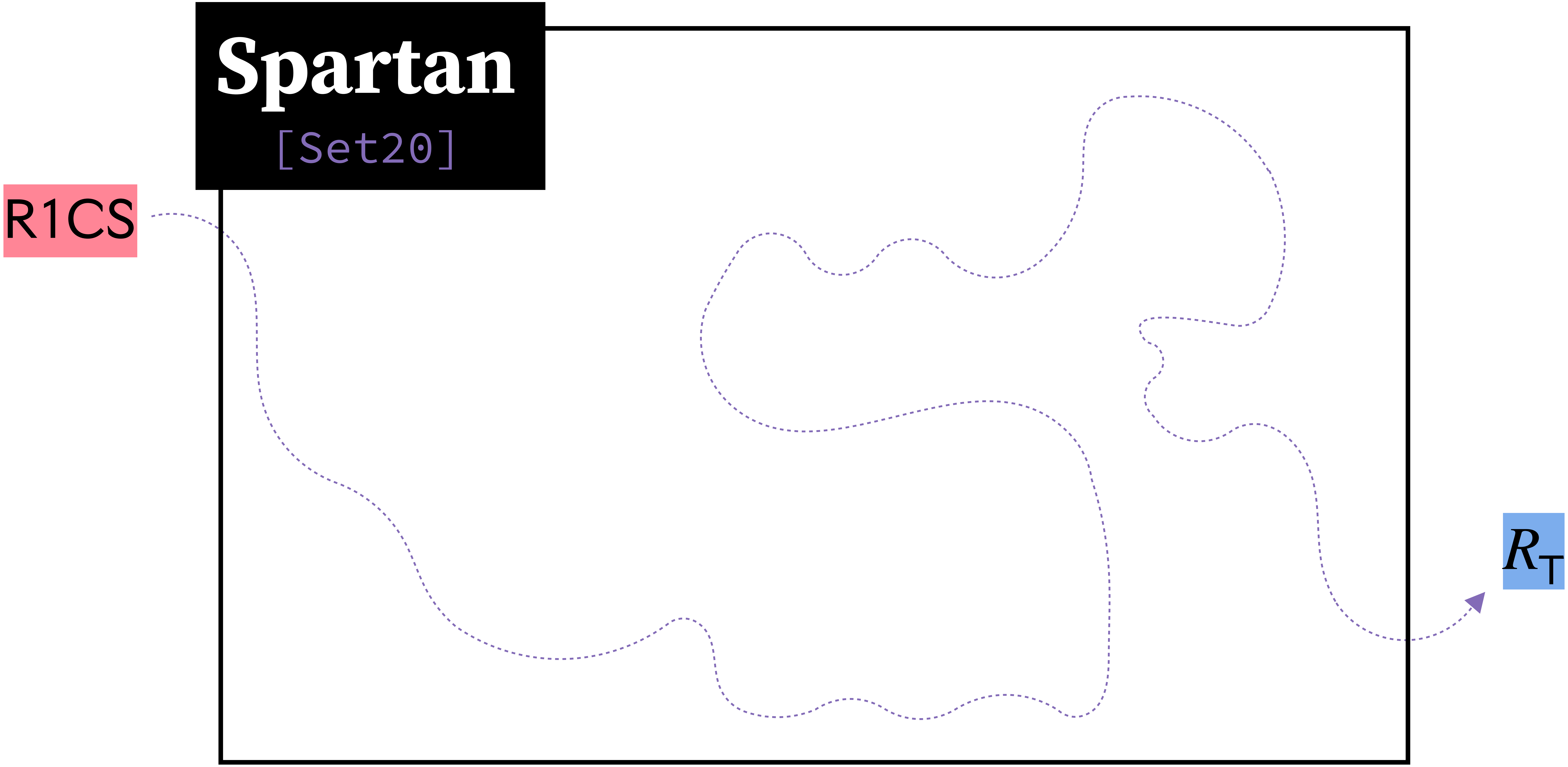


Tree Folding Scheme with a Simple Proof

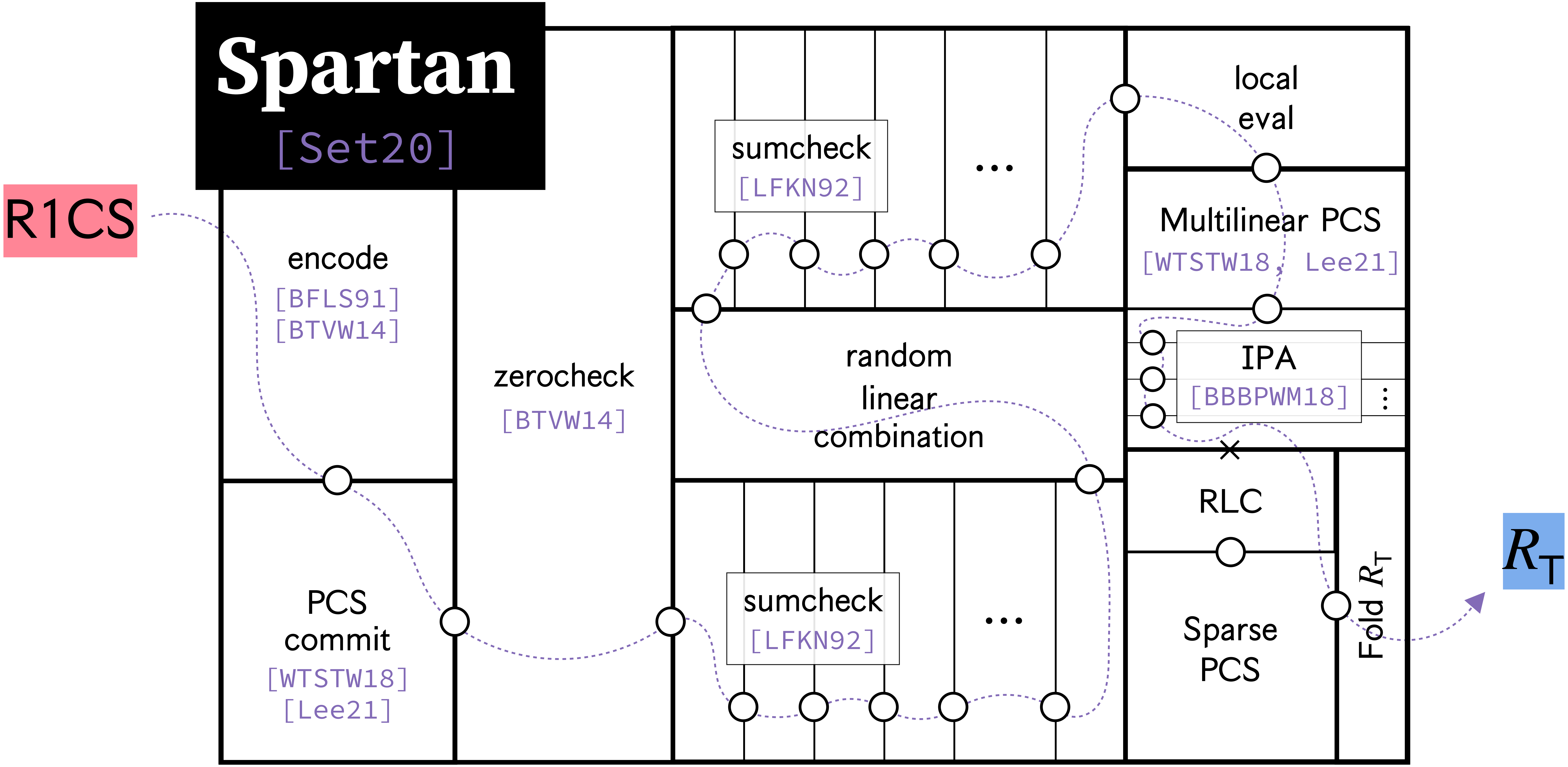
Given $\Pi_2 : R^2 \to R$, then $\Pi_\ell = \Pi_2 \circ (\Pi_{\ell/2} \times \Pi_{\ell/2}) : R^\ell \to R$.

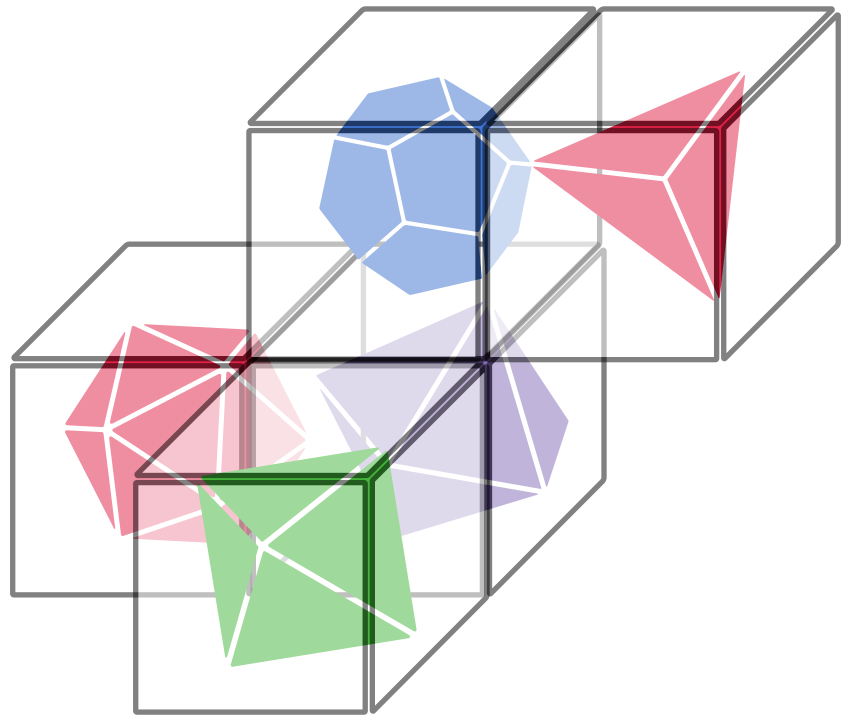


Taming the Complexity of Modern Arguments



Taming the Complexity of Modern Arguments





Reductions of knowledge serve as both a unifying abstraction and a compositional framework.

akothapalli@cmu.edu

References

[BCLMS21] Bünz, Chiesa, Lin, Mishra, Spooner. Proof Carrying Data without Succinct Arguments.

[BDFG21] Boneh, Drake, Fisch, Gabizon. Halo Infinite: Recursive zkSNARKs from any Additive Polynomial Commitment Scheme.

[BCCGP16] Bootle, Cerulli, Chaidos, Groth, Petit. Efficient Zero-Knowledge Arguments for Arithmetic Circuits in the Discrete Log Setting.

[RZ21] Ràfols and Zapico. An Algebraic Framework for Universal and Updatable SNARKs.

[KST22] Kothapalli, Setty, Tzialla. Nova: Recursive Zero-Knowledge Arguments from Folding Schemes.

[CNRZZ22] Campanelli, Nitulescu, Rafols, Zacharakis, Zapico. Linear-map vector commitments and their practical applications.

[LFKN92] Lund, Fortnow, Karloff, Nisan. Algebraic methods for interactive proof systems.

[BBBPWM18] Bünz, Bootle, Boneh, Poelstra, Wuille, and Maxwell. Bulletproofs: Short Proofs for Confidential Transactions and More.

[BMMTV21] Bünz, Maller, Mishra, Tyagi, and Vesely. Proofs for inner pairing products and applications.

[WTSTW18] Wahby, Tzialla, Shelat, Thaler, and Walfish. Doubly-efficient zkSNARKs without trusted setup.

[BTVW14] Blumberg, Thaler, Vu, and Walfish. Verifiable computation using multiple provers.

[GMR85] Goldwasser, Micali, and Rackoff. The knowledge complexity of interactive proof systems.

[Lee21] Lee. Dory: Efficient, Transparent arguments for Generalised Inner Products and Polynomial Commitments.

[BZ12] Bayer, and Groth. Efficient zero-knowledge argument for correctness of a shuffle.

[CBBZ22] Chen, Bünz, Boneh, Zhang. HyperPlonk: Plonk with Linear-Time Prover and High-Degree Custom Gates.

[Set20] Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup.

[BCHO22] Gemini: Bootle, Chiesa, Hu, Orrù. Elastic SNARKs for Diverse Environments.

[Bay13] Bayer. Practical Zero-Knowledge Protocols Based on the Discrete Logarithm Assumption.

[BCS21] Bootle, Chiesa, and Sotiraki. Sumcheck Arguments and their Applications.

[RZ22] Ràfols, and Zacharakis. Folding Schemes with Selective Verification.

[KS23] Kothapalli, and Setty. HyperNova: Recursive arguments for customizable constraint systems.

[Val08] Valiant. Incrementally Verifiable Computation or Proofs of Knowledge Imply Time/Space Efficiency.

[BGH19] Bowe, Grigg, Hopwood. Recursive proof composition without a trusted setup.

[AC20] Attema and Cramer. Compressed-protocol theory and practical application to plug & play secure algorithmics.

[ACR21] Attema, Cramer, and Rambaud. Compressed Sigma protocols for bilinear group arithmetic circuits and application to logarithmic transparent threshold signatures.

[GKR15] Goldwasser, Tauman Kalai, and Rothblum. Delegating computation: interactive proofs for muggles

[BFLS91] Babai, Fortnow, Levin, and Szegedy. Checking computations in polylogarithmic time.

[BC23] Bünz, and Chen. ProtoStar: Generic Efficient Accumulation/Folding for Special Sound Protocols