zkSNARKs in the ROM with Unconditional UC-Security

TL;DR Micali and BCS are UC-secure in the GROM

Giacomo Fenzi @ EPFL

eprint.iacr.org/2024/724

Joint work with Alessandro Chiesa

EPFL

Motivation

zkSNARKs are deployed in the real world

zkSNARKs are deployed in the real world

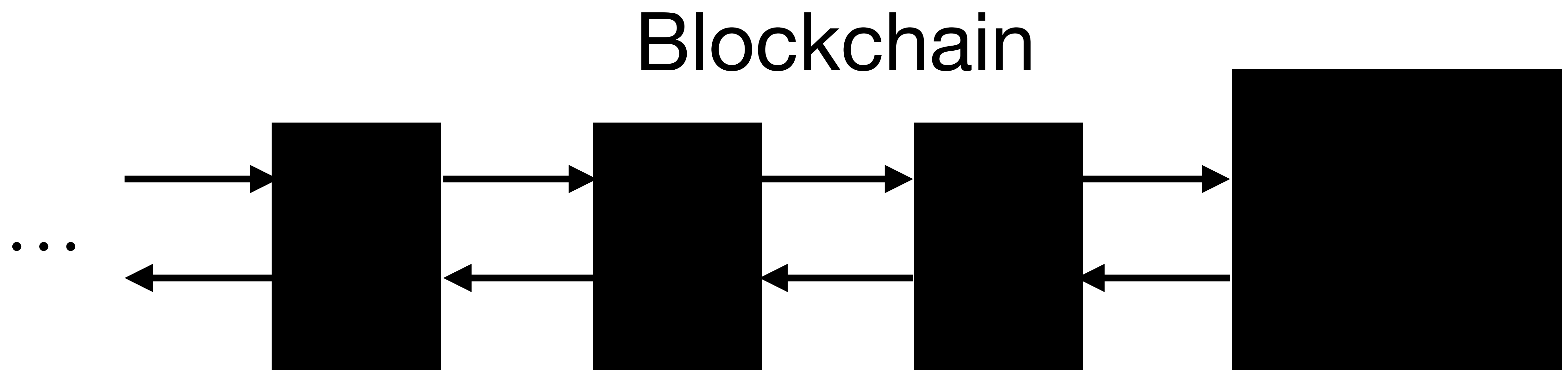
zkSNARKs are ZKPs where verification is **exponentially** faster than execution.

zkSNARKs are deployed in the real world

E.g.: proof based rollups to improve scalability

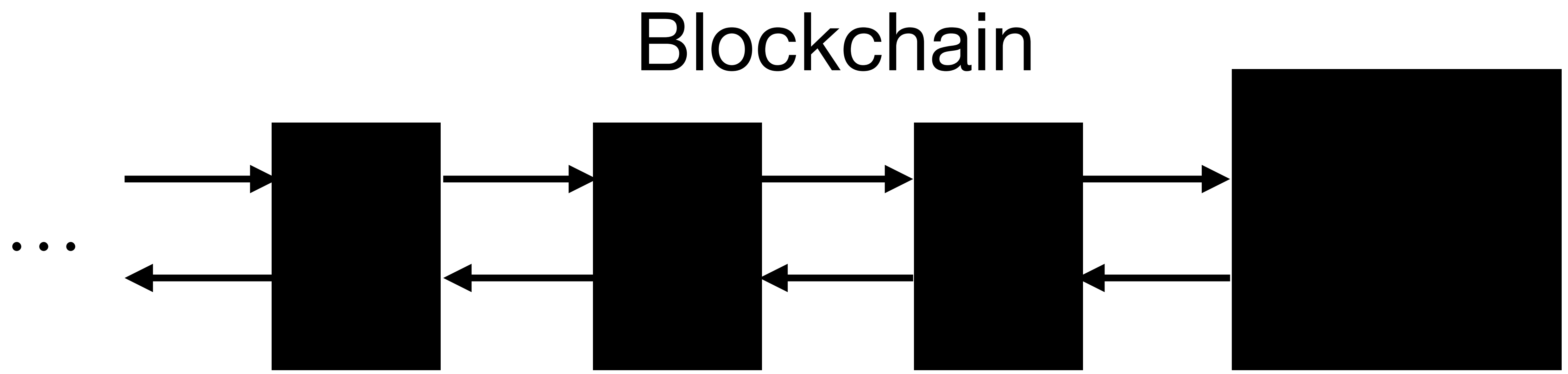
zkSNARKs are deployed in the real world

E.g.: proof based rollups to improve scalability

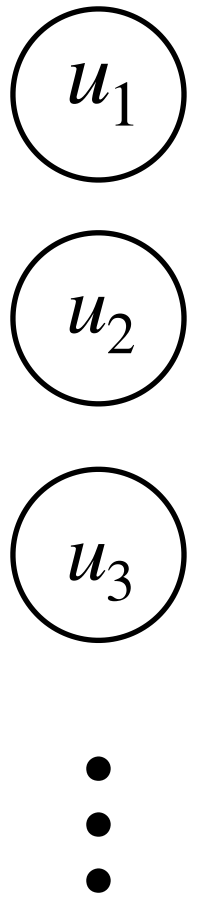


zkSNARKs are deployed in the real world

E.g.: proof based rollups to improve scalability

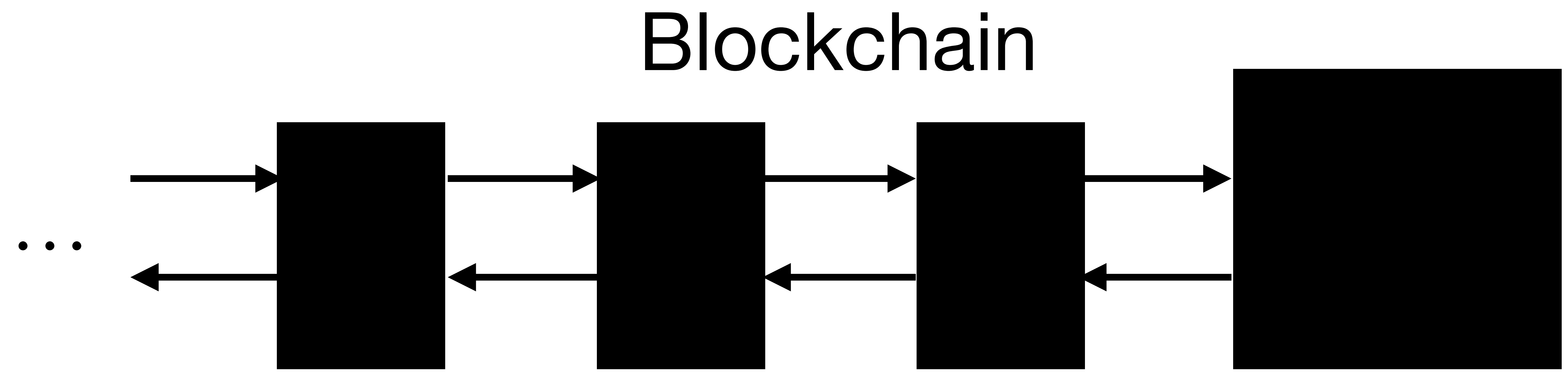


Rollup Users



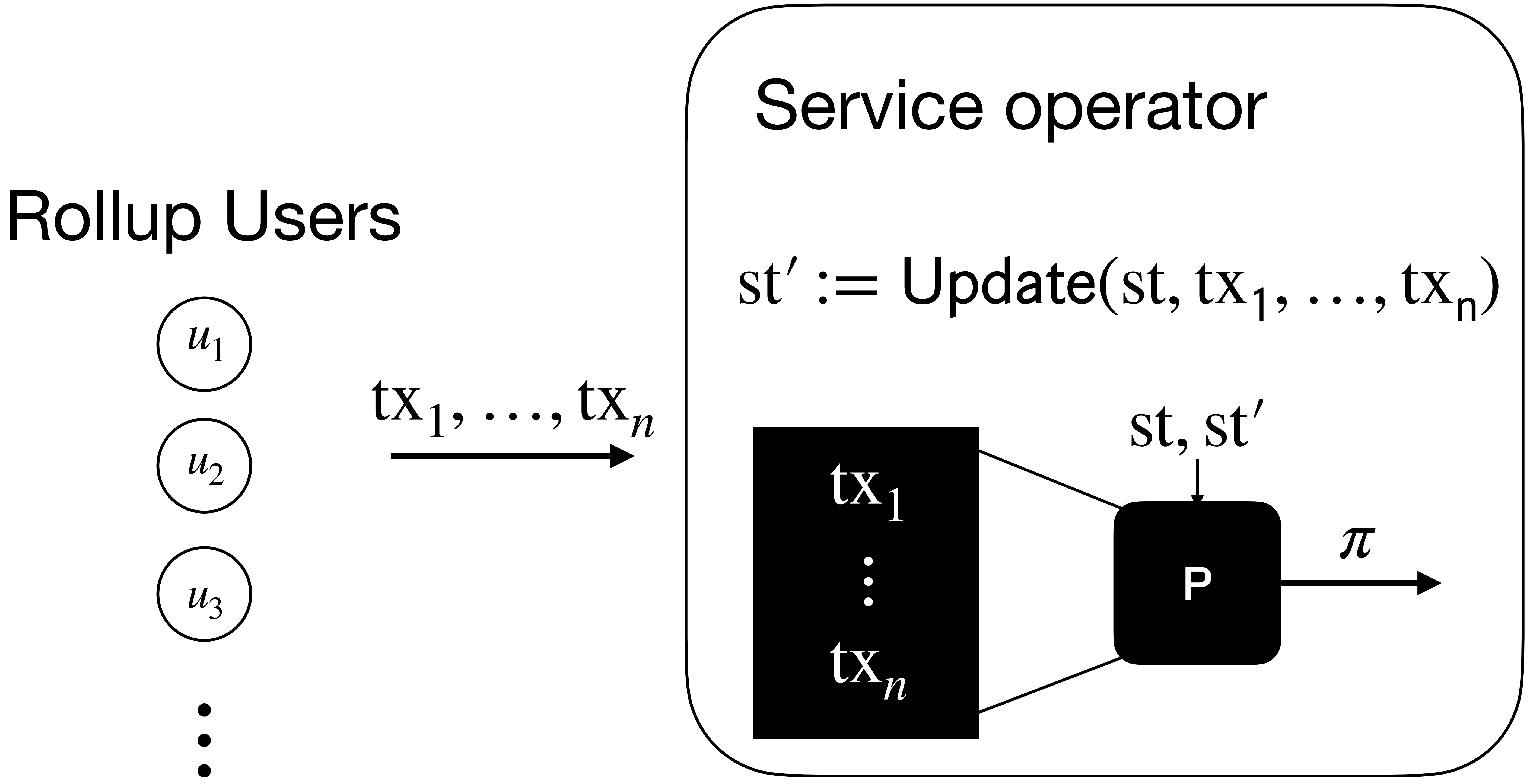
zkSNARKs are deployed in the real world

E.g.: proof based rollups to improve scalability



Blockchain

…



Service operator

Rollup Users

$\mathrm{st}' := \mathsf{Update}(\mathrm{st}, \mathrm{tx}_1, \ldots, \mathrm{tx}_n)$

u_1

u_2

u_3

$\mathrm{tx}_1, \ldots, \mathrm{tx}_n$

tx_1

⋮

tx_n

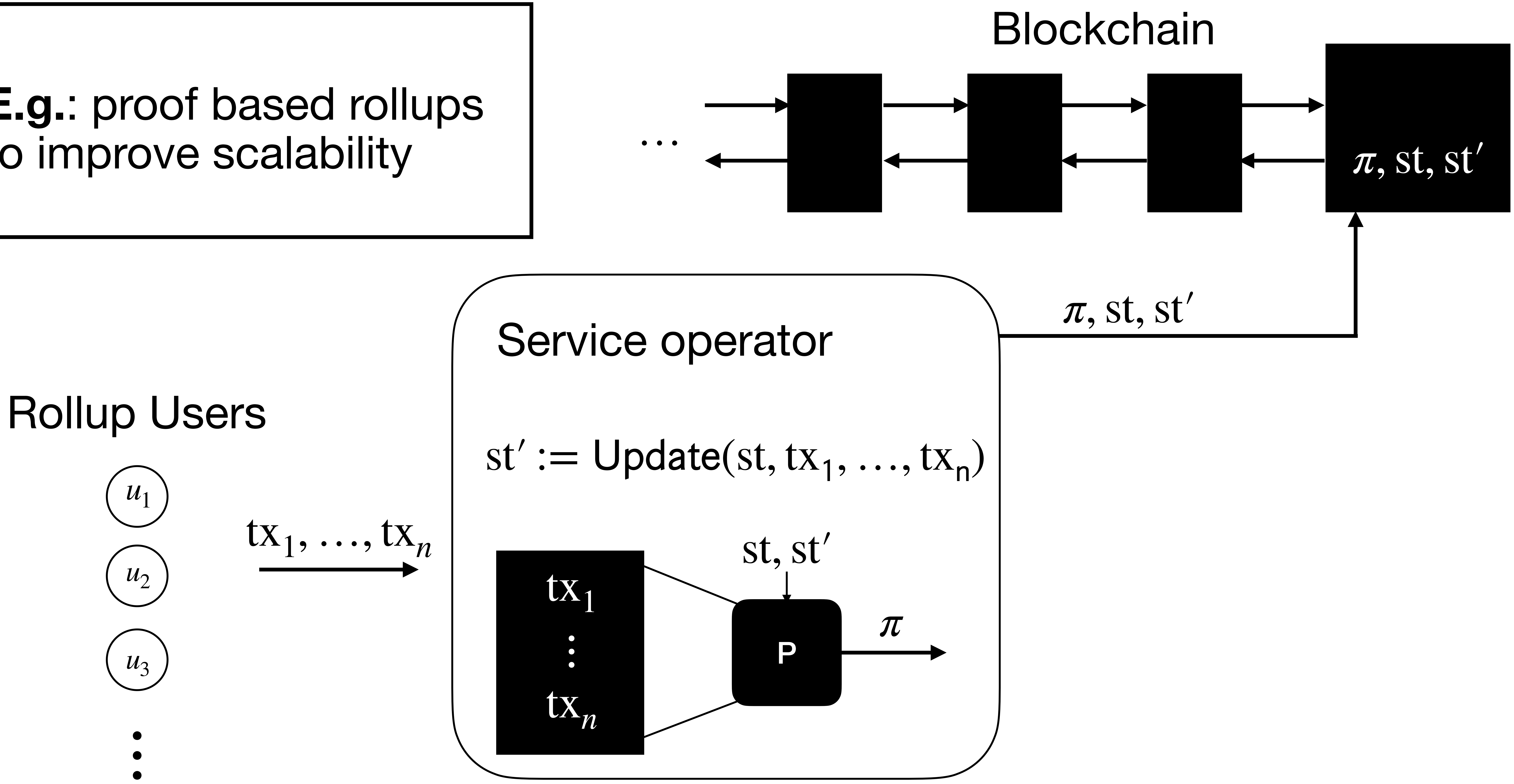
$\mathrm{st}, \mathrm{st}'$

P

π

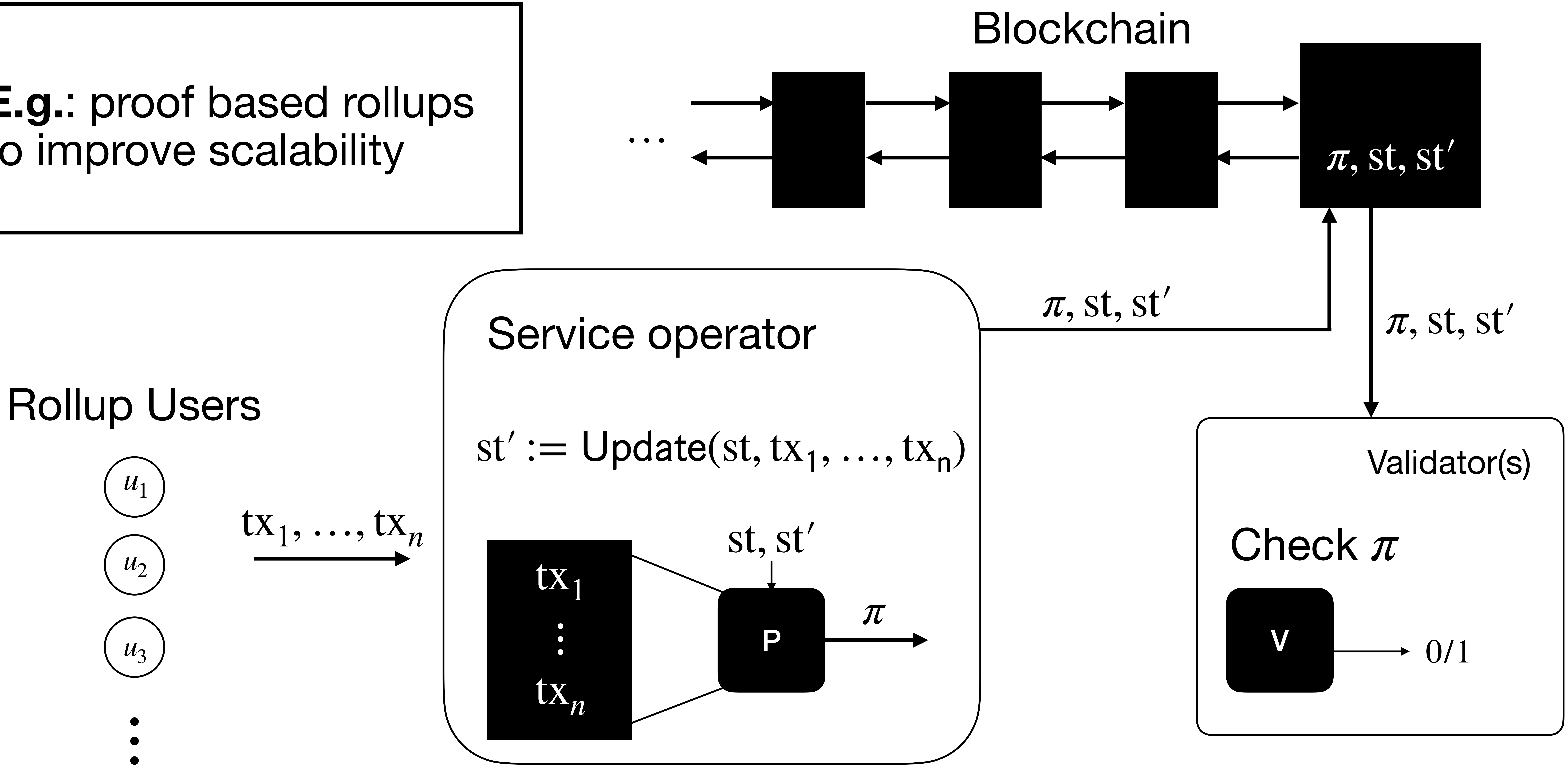
zkSNARKs are deployed in the real world

E.g.: proof based rollups to improve scalability



zkSNARKs are deployed in the real world

E.g.: proof based rollups to improve scalability



Blockchain

…

π, st, st$'$

Service operator

$\mathrm{st}' :=$ Update(st, $\mathrm{tx}_1, \ldots, \mathrm{tx}_\mathrm{n}$)

tx_1 ⋮ tx_n

st, st$'$

P

π

π, st, st$'$

π, st, st$'$

Rollup Users

u_1

u_2

u_3

⋮

$\mathrm{tx}_1, \ldots, \mathrm{tx}_n$

Validator(s)

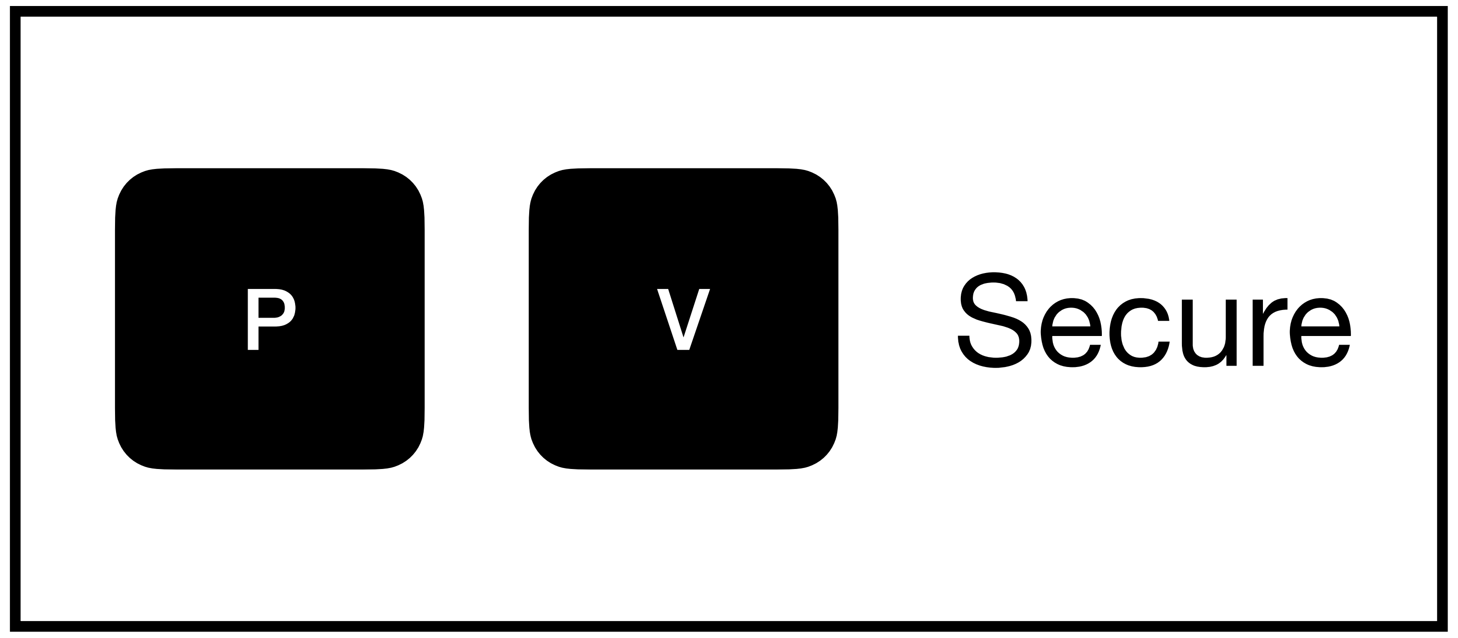
Check π

V

0/1

Goal: Modular Security Analysis

Goal: Modular Security Analysis

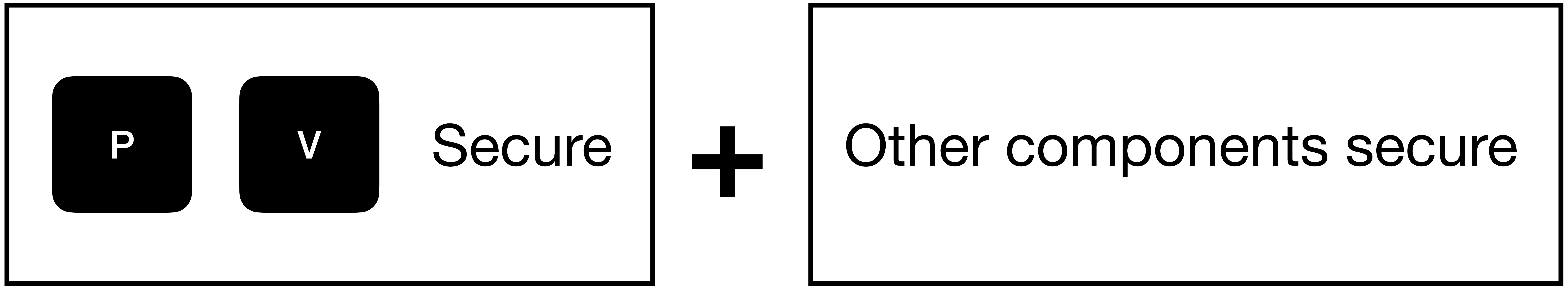


P

V

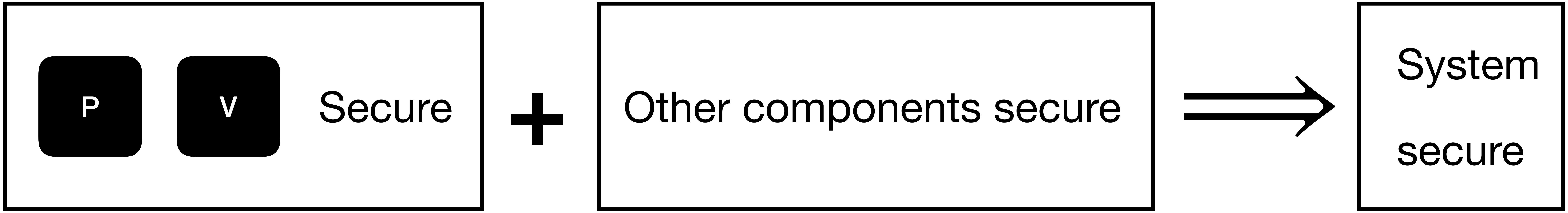
Secure

Goal: Modular Security Analysis



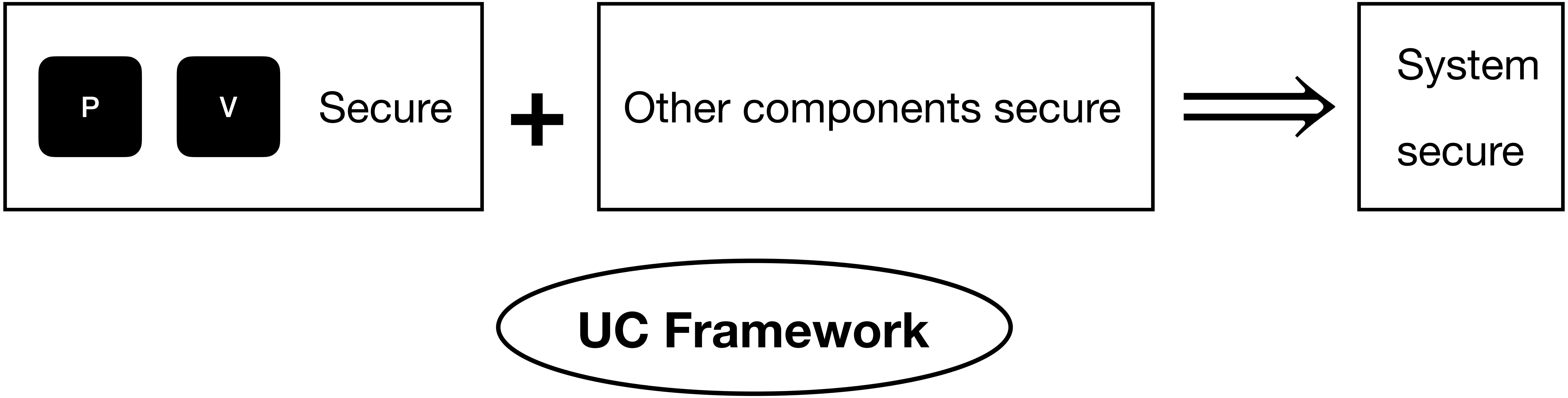
Secure + Other components secure

Goal: Modular Security Analysis



Secure + Other components secure ⟹ System secure

Goal: Modular Security Analysis



P

V

Secure

+

Other components secure

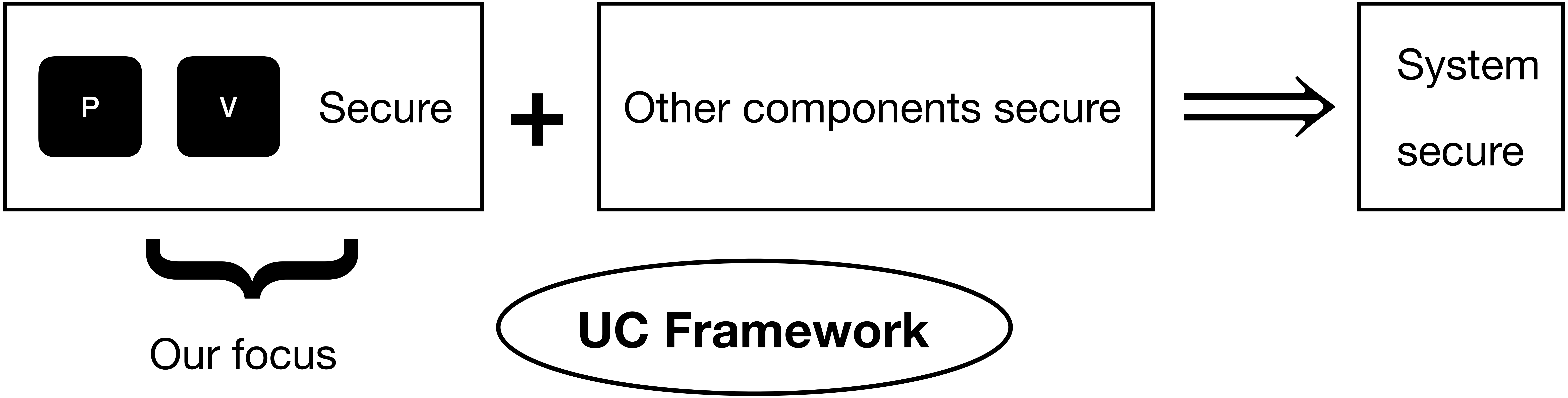
⟹

System

secure

UC Framework

Goal: Modular Security Analysis



P

V

Secure

+

Other components secure

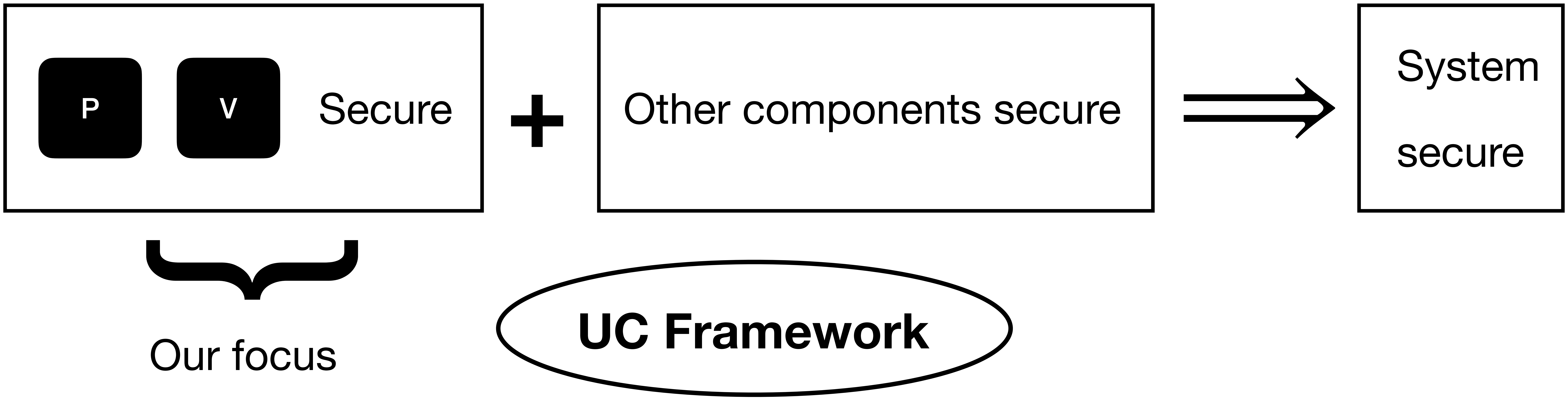
⟹

System secure

Our focus

UC Framework

Goal: Modular Security Analysis



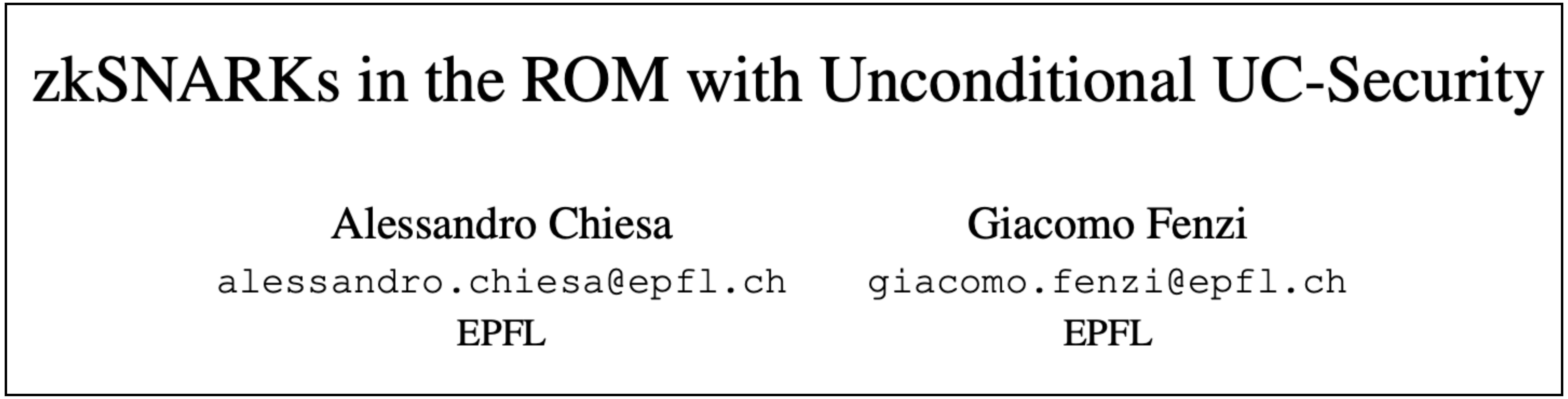
P V Secure + Other components secure ⟹ System secure

Our focus

UC Framework

Which zkSNARKs are UC-secure?

This work

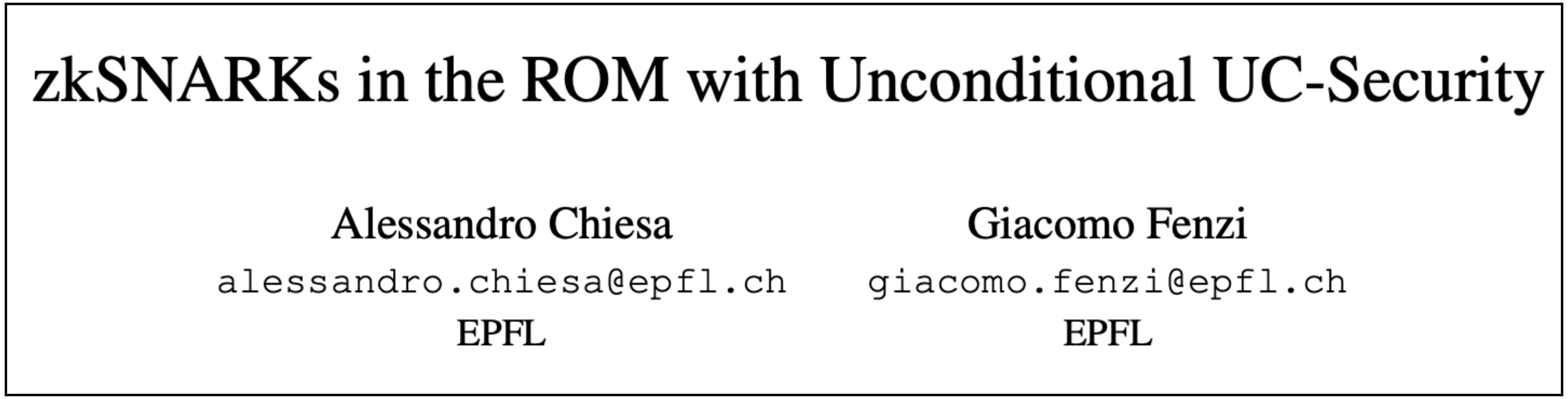


zkSNARKs in the ROM with Unconditional UC-Security

Alessandro Chiesa
alessandro.chiesa@epfl.ch
EPFL

Giacomo Fenzi
giacomo.fenzi@epfl.ch
EPFL

This work



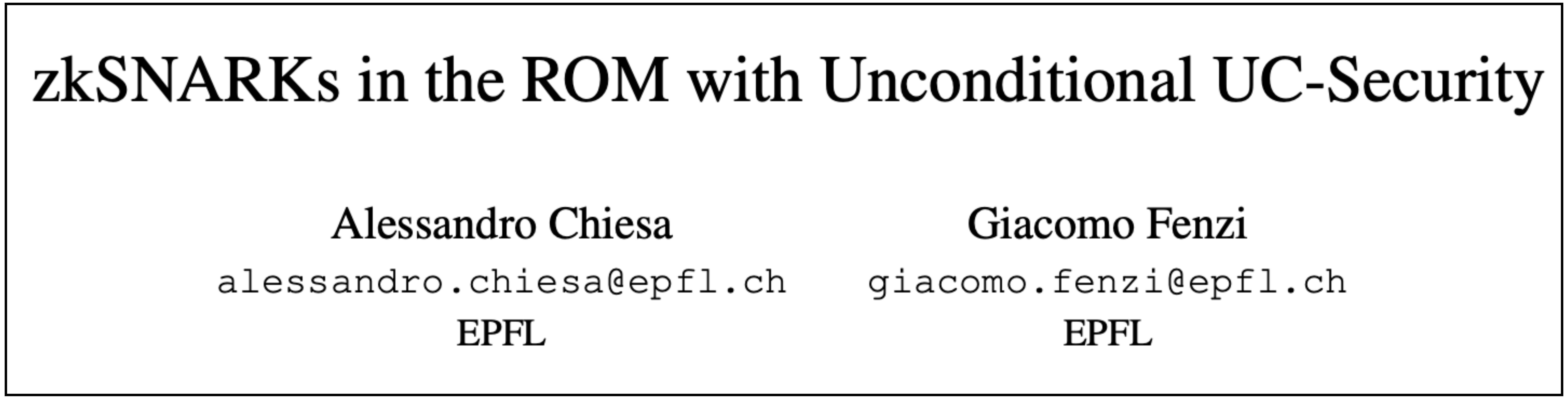
zkSNARKs in the ROM with Unconditional UC-Security

Alessandro Chiesa
alessandro.chiesa@epfl.ch
EPFL

Giacomo Fenzi
giacomo.fenzi@epfl.ch
EPFL

Show **existing** zkSNARKs are UC-secure
(including deployed ones)

This work



zkSNARKs in the ROM with Unconditional UC-Security

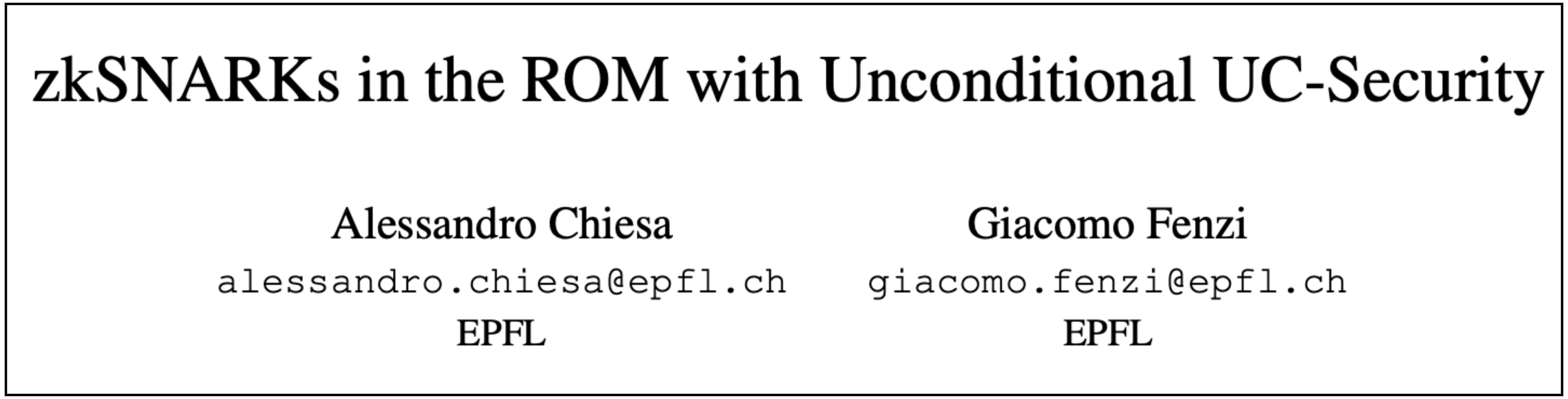
Alessandro Chiesa
alessandro.chiesa@epfl.ch
EPFL

Giacomo Fenzi
giacomo.fenzi@epfl.ch
EPFL

Show **existing** zkSNARKs are UC-secure
(including deployed ones)

Succinct

This work



zkSNARKs in the ROM with Unconditional UC-Security

Alessandro Chiesa
alessandro.chiesa@epfl.ch
EPFL

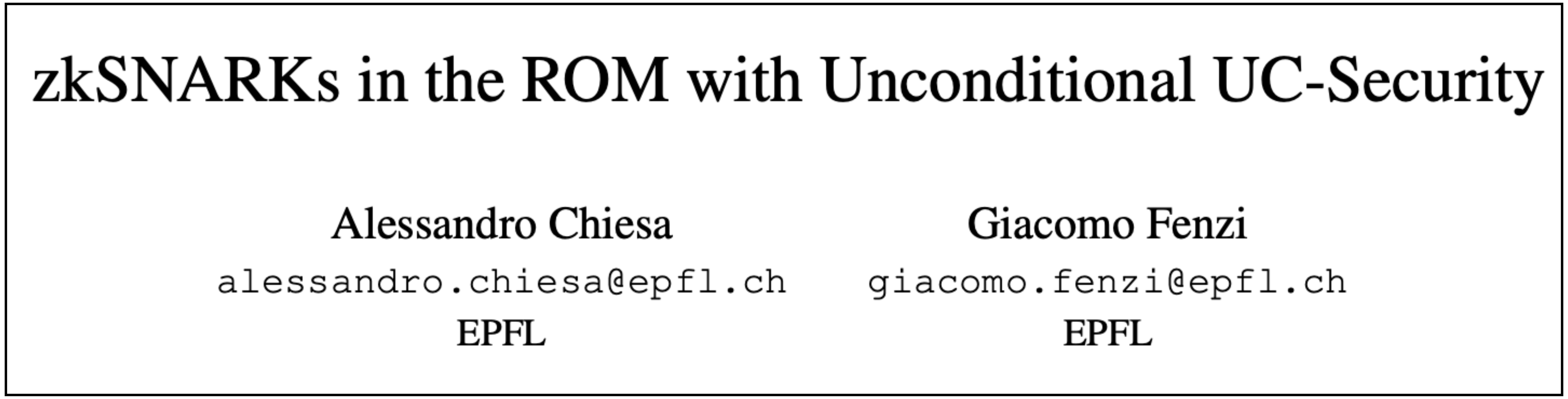
Giacomo Fenzi
giacomo.fenzi@epfl.ch
EPFL

Show **existing** zkSNARKs are UC-secure
(including deployed ones)

Succinct

ROM **only**: transparent, post-quantum,
unconditional security

This work



zkSNARKs in the ROM with Unconditional UC-Security

Alessandro Chiesa
alessandro.chiesa@epfl.ch
EPFL

Giacomo Fenzi
giacomo.fenzi@epfl.ch
EPFL

Show **existing** zkSNARKs are UC-secure (including deployed ones)

Succinct

ROM **only**: transparent, post-quantum, unconditional security

Concrete security bounds: useful for practitioners

Our results

Main Thm.

There exists a zkSNARK that is unconditionally UC-secure in the GROM

Lemma

Let ARG *be a "UC-friendly" argument in the ROM.*

Then, $\Pi[\mathsf{ARG}]$ *is UC-secure in the GROM*

Theorem

The Micali construction is "UC-friendly" in the ROM, provided that the underlying PCP is honest-verifier zero knowledge and knowledge sound.

Corollary

The Micali construction is UC-secure in the GROM, when instantiated as above.

Theorem

The Micali construction is "UC-friendly" in the ROM, provided that the underlying PCP is honest-verifier zero knowledge and knowledge sound.

Same conditions required for KS of Micali in the ROM

Corollary

The Micali construction is UC-secure in the GROM, when instantiated as above.

Theorem

The ***BCS*** *construction is "UC-friendly" in the ROM, provided that the underlying* ***IOP*** *is honest-verifier zero knowledge and* ***(state-restoration)*** *knowledge sound.*

Corollary

The ***BCS*** *construction is UC-secure in the GROM, when instantiated as above.*

Theorem

*The **BCS** construction is "UC-friendly" in the ROM, provided that the underlying **IOP** is honest-verifier zero knowledge and **(state-restoration)** knowledge sound.*

Same conditions required for KS of BCS in the ROM

Corollary

*The **BCS** construction is UC-secure in the GROM, when instantiated as above.*

Techniques

GROM

[CDGLN18]

GROM

[CDGLN18]

Goal: ROM-like interface shared by **all** parties in the security experiment

GROM

[CDGLN18]

Goal: ROM-like interface shared by **all** parties in the security experiment

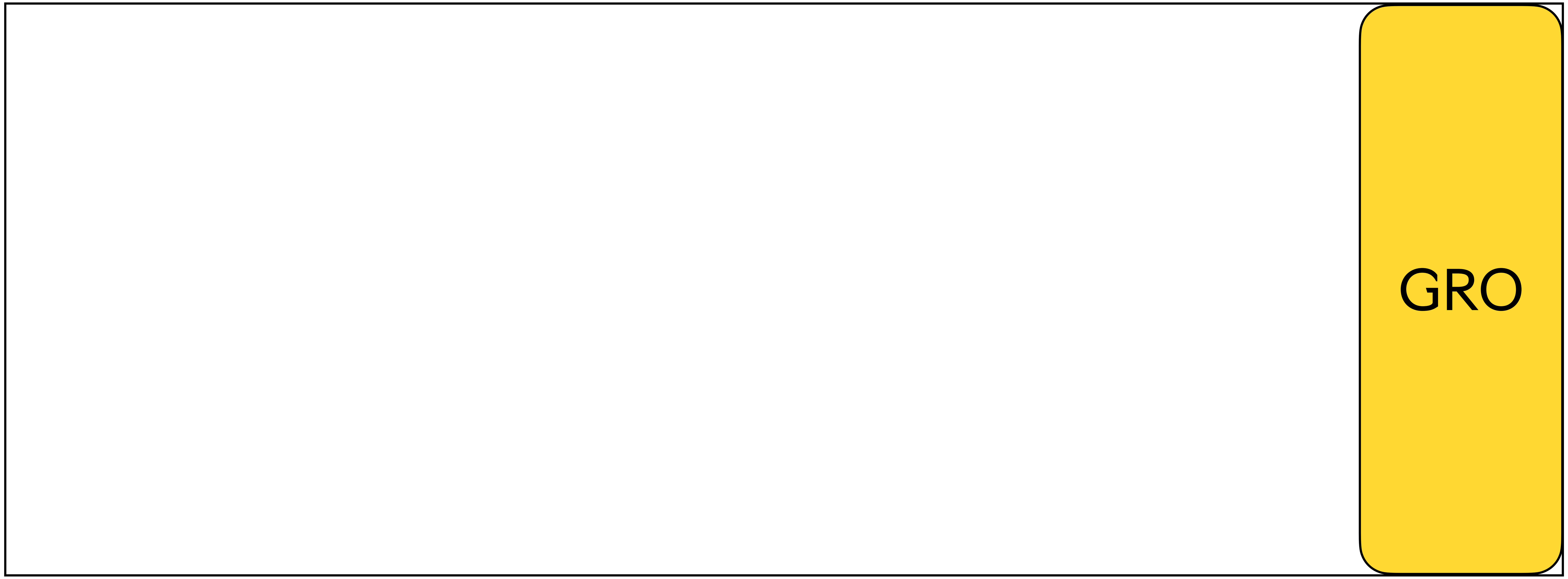
Flavor: **r**estricted **p**rogrammable and **o**bservable **g**lobal **r**andom **o**racle

GROM

[CDGLN18]

Goal: ROM-like interface shared by **all** parties in the security experiment

Flavor: **r**estricted **p**rogrammable and **o**bservable **g**lobal **r**andom **o**racle

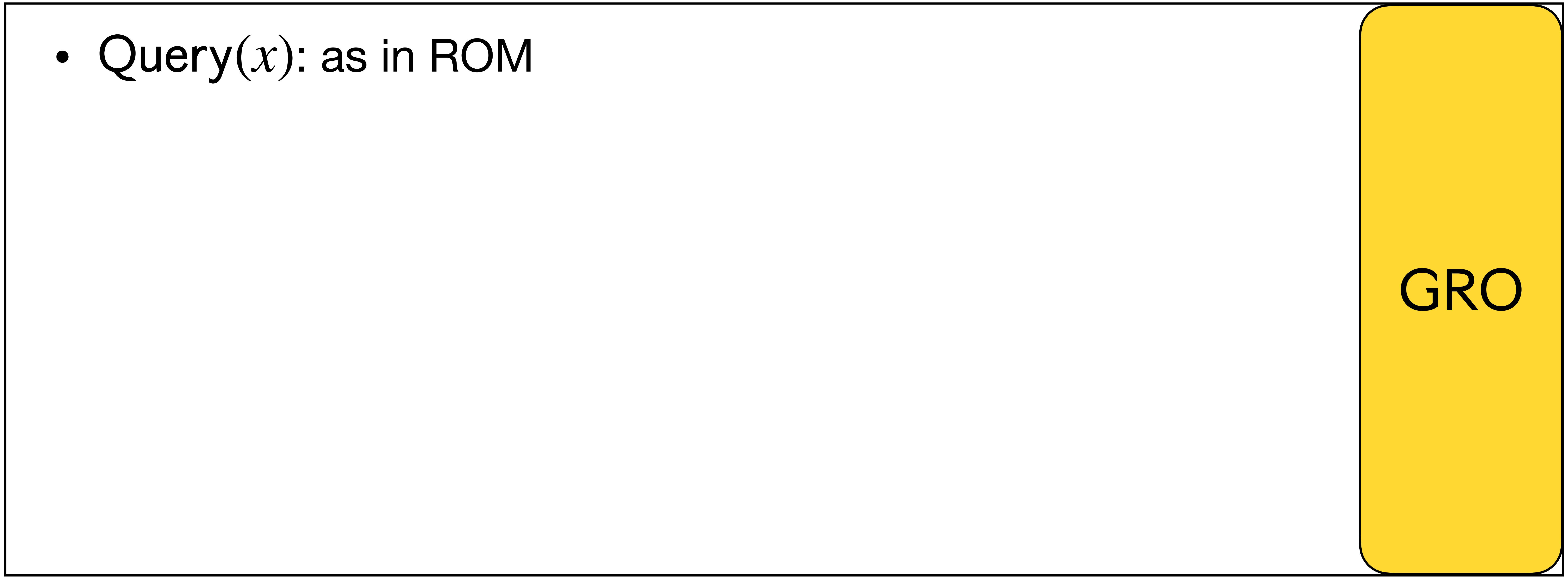


GROM

[CDGLN18]

Goal: ROM-like interface shared by **all** parties in the security experiment

Flavor: **r**estricted **p**rogrammable and **o**bservable **g**lobal **r**andom **o**racle



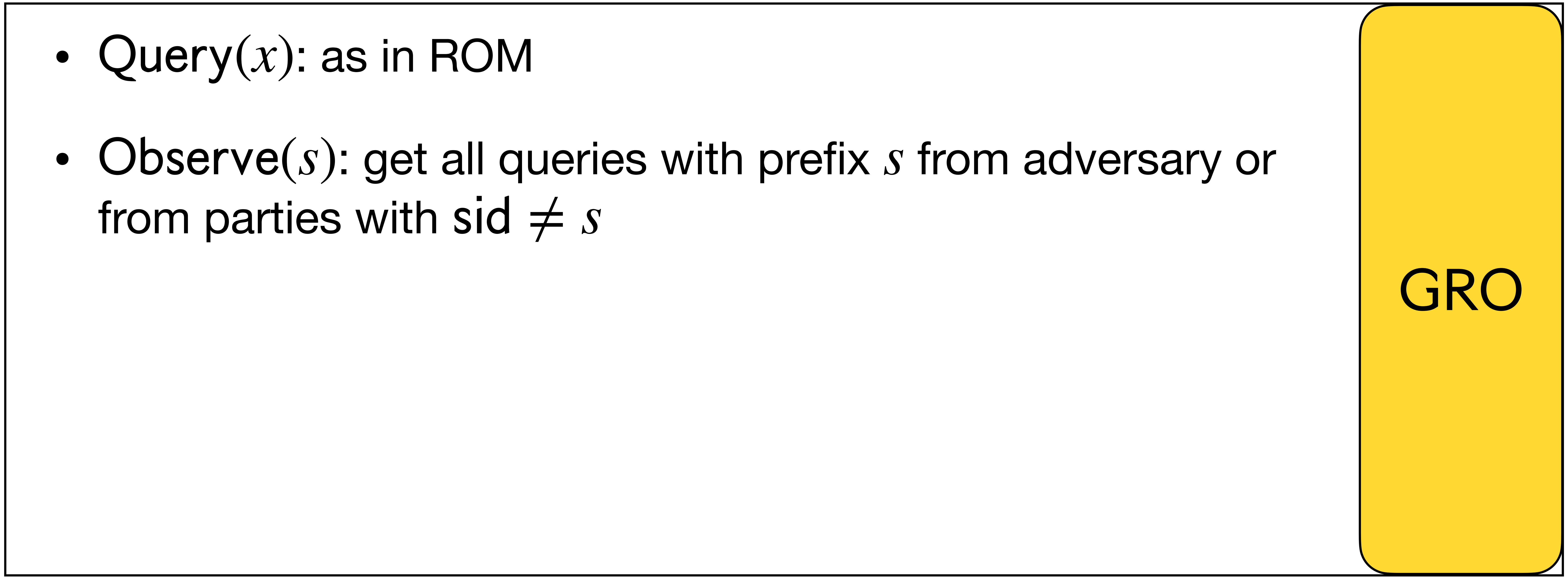
- Query(x): as in ROM

GROM

[CDGLN18]

Goal: ROM-like interface shared by **all** parties in the security experiment

Flavor: **r**estricted **p**rogrammable and **o**bservable **g**lobal **r**andom **o**racle



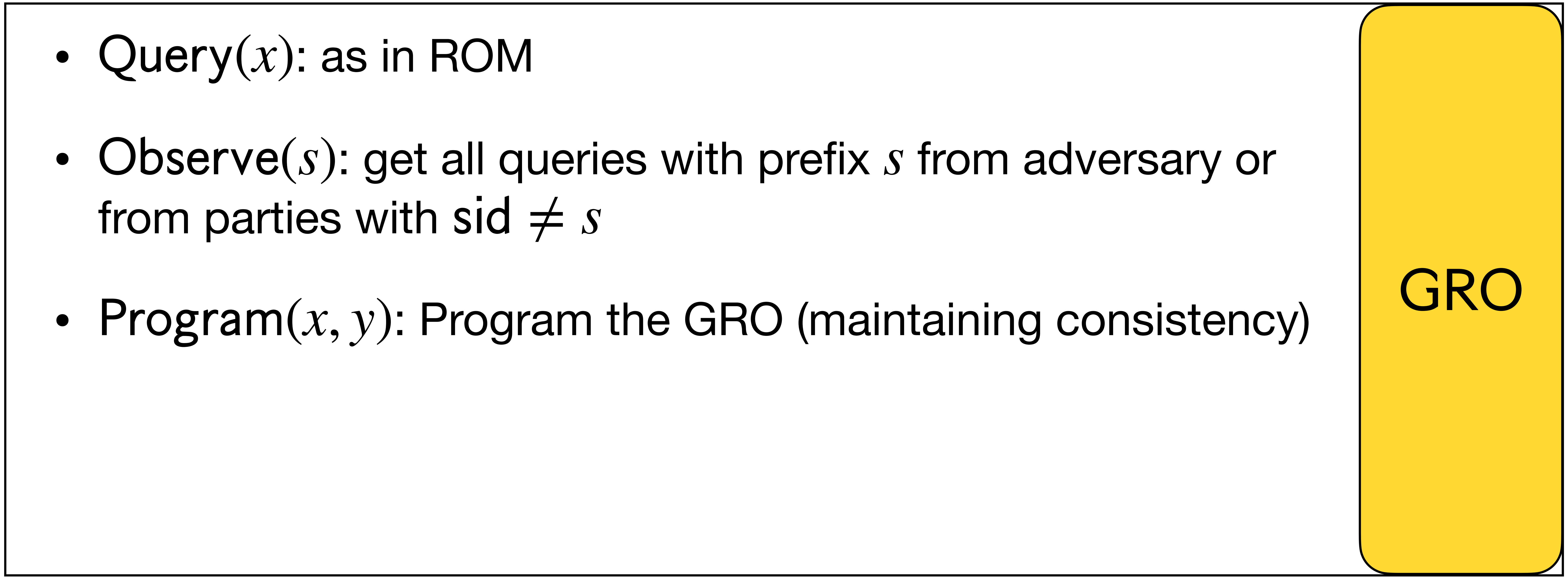
- Query(x): as in ROM
- Observe(s): get all queries with prefix s from adversary or from parties with sid $\neq s$

GROM

[CDGLN18]

Goal: ROM-like interface shared by **all** parties in the security experiment

Flavor: **r**estricted **p**rogrammable and **o**bservable **g**lobal **r**andom **o**racle



- Query(x): as in ROM
- Observe(s): get all queries with prefix s from adversary or from parties with sid $\neq s$
- Program(x, y): Program the GRO (maintaining consistency)

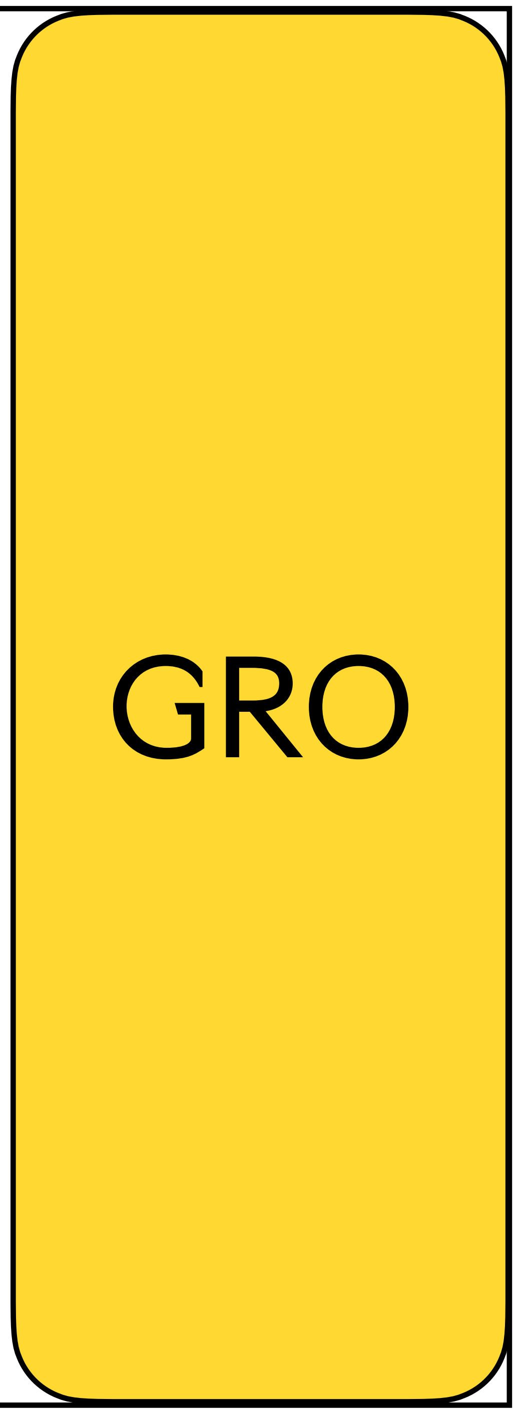
GROM

[CDGLN18]

Goal: ROM-like interface shared by **all** parties in the security experiment

Flavor: **r**estricted **p**rogrammable and **o**bservable **g**lobal **r**andom **o**racle

- $\mathsf{Query}(x)$: as in ROM
- $\mathsf{Observe}(s)$: get all queries with prefix s from adversary or from parties with $\mathsf{sid} \neq s$
- $\mathsf{Program}(x, y)$: Program the GRO (maintaining consistency)
- $\mathsf{IsProgrammed}(x)$: allows parties in session sid to check if a $x = \mathsf{sid} \circ x'$ has been programmed



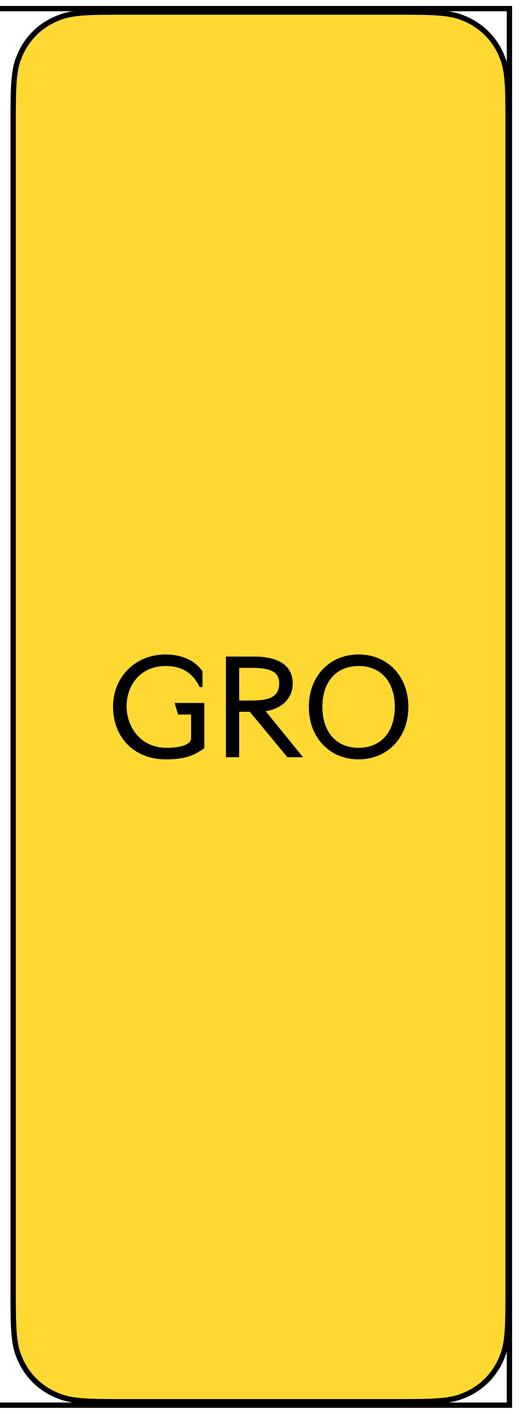
GROM

[CDGLN18]

Goal: ROM-like interface shared by **all** parties in the security experiment

Flavor: **r**estricted **p**rogrammable and **o**bservable **g**lobal **r**andom **o**racle

- $\mathsf{Query}(x)$: as in ROM
- $\mathsf{Observe}(s)$: get all queries with prefix s from adversary or from parties with $\mathsf{sid} \neq s$
- $\mathsf{Program}(x, y)$: Program the GRO (maintaining consistency)
- $\mathsf{IsProgrammed}(x)$: allows parties in session sid to check if a $x = \mathsf{sid} \circ x'$ has been programmed



GROM

[CDGLN18]

Goal: ROM-like interface shared by **all** parties in the security experiment

Flavor: **r**estricted **p**rogrammable and **o**bservable **g**lobal **r**andom **o**racle

- Query(x): as in ROM
- Observe(s): get all queries with prefix s from adversary or from parties with sid $\neq s$
- Program(x, y): Program the GRO (maintaining consistency)
- IsProgrammed(x): allows parties in session sid to check if a $x = \mathsf{sid} \circ x'$ has been programmed

GRO

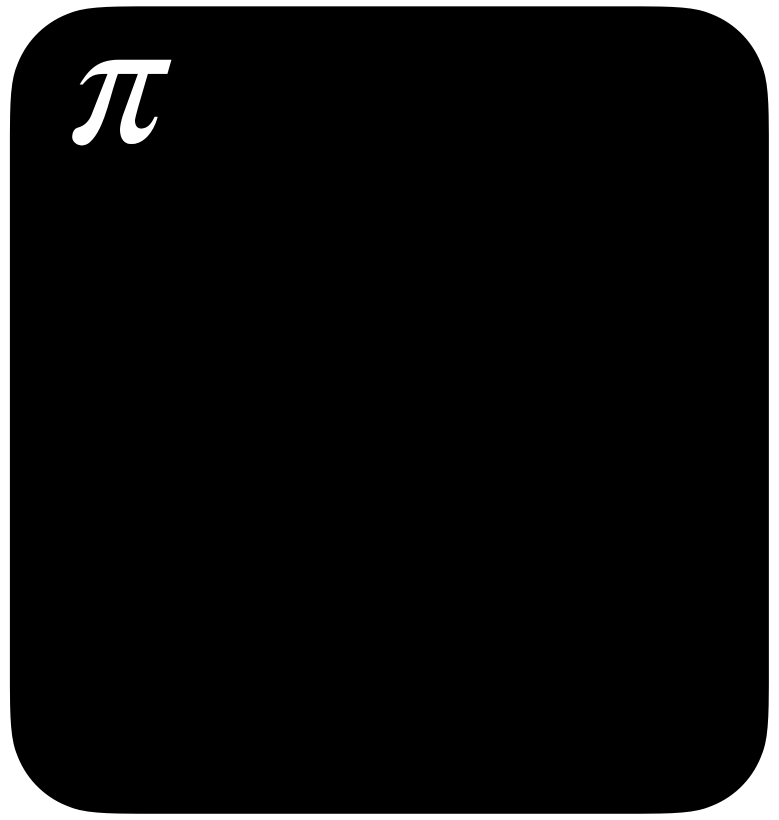
Crucial: Simulator can program points without being detected!

UC with Budgets

Plain UC only models adversaries that are **computationally** bounded using import

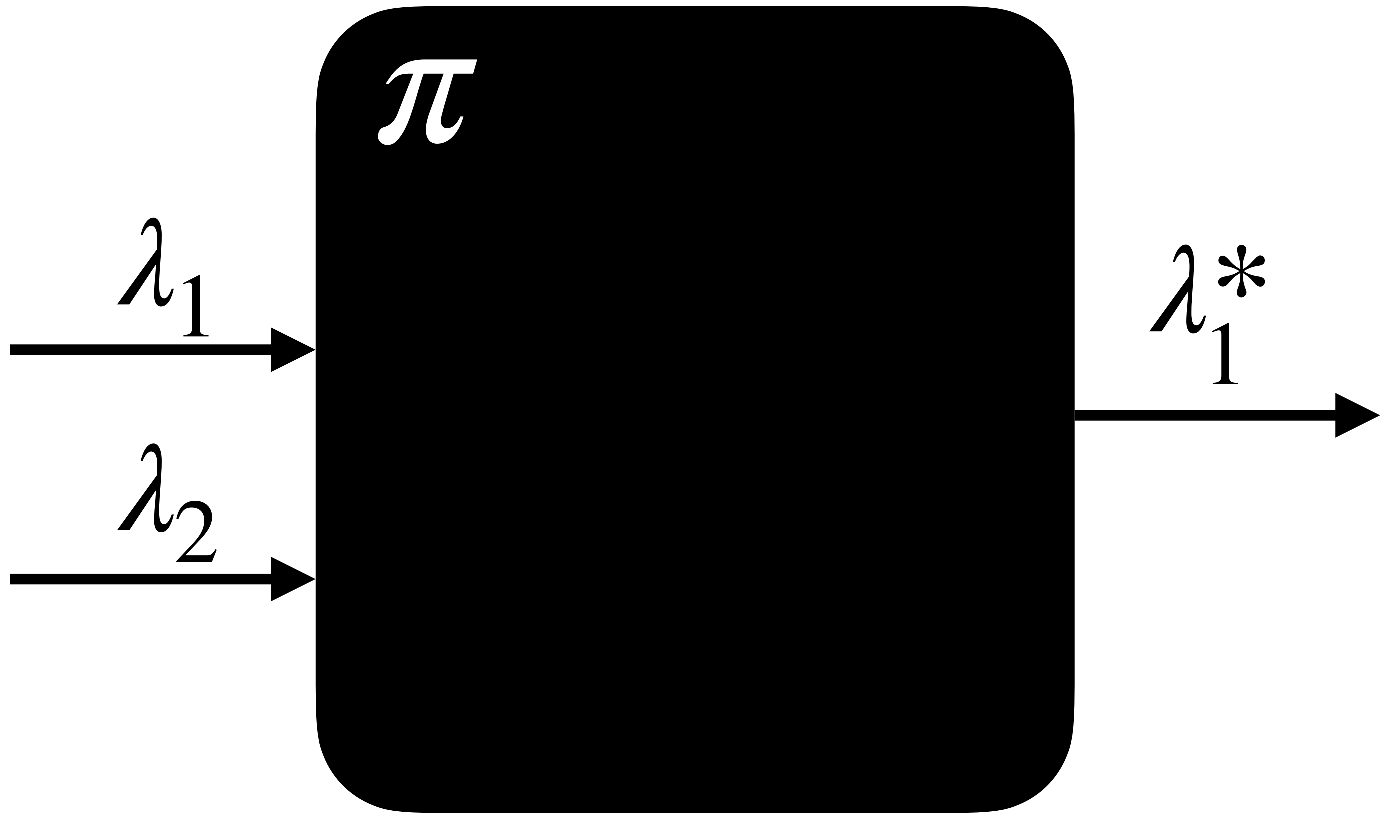
UC with Budgets

Plain UC only models adversaries that are **computationally** bounded using import



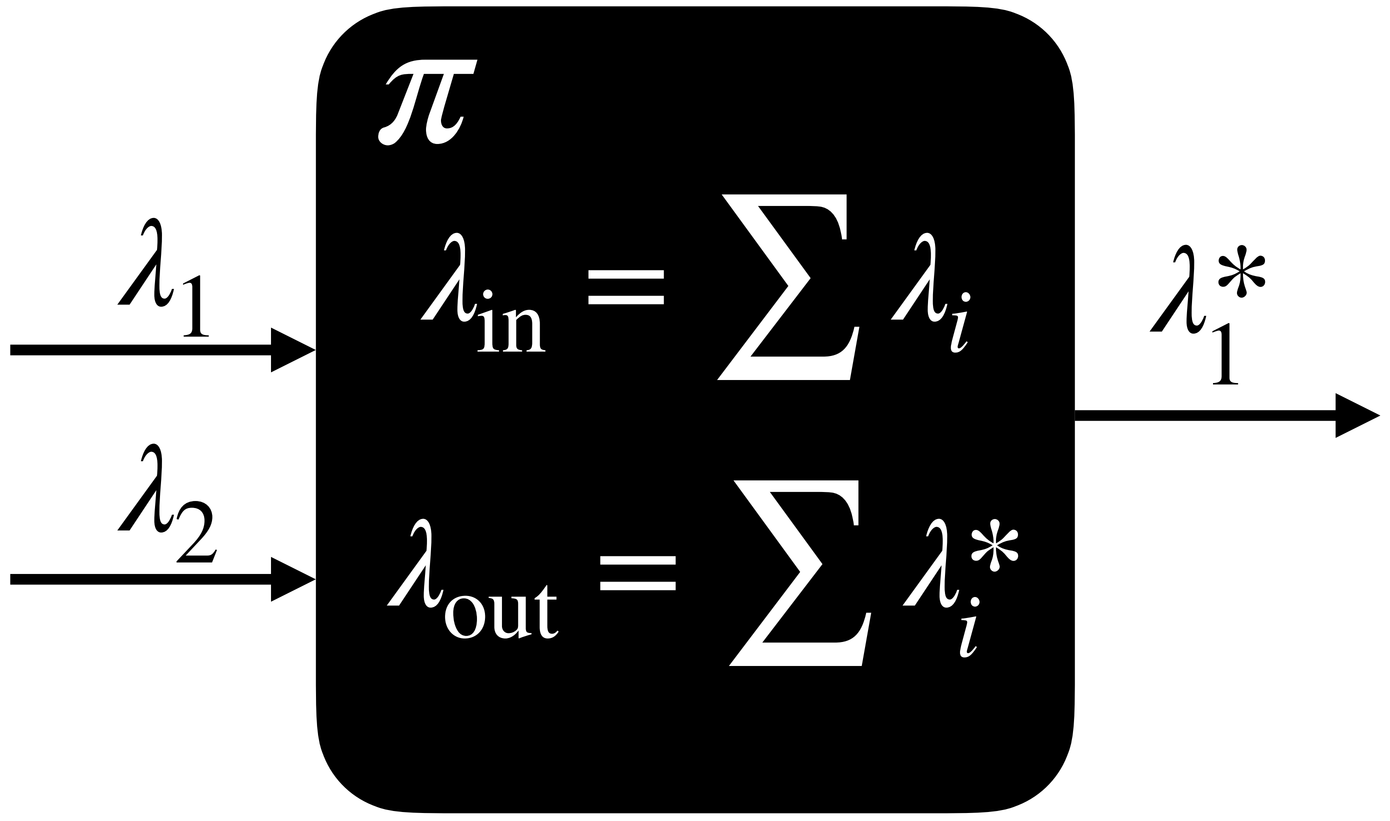
UC with Budgets

Plain UC only models adversaries that are **computationally** bounded using import



UC with Budgets

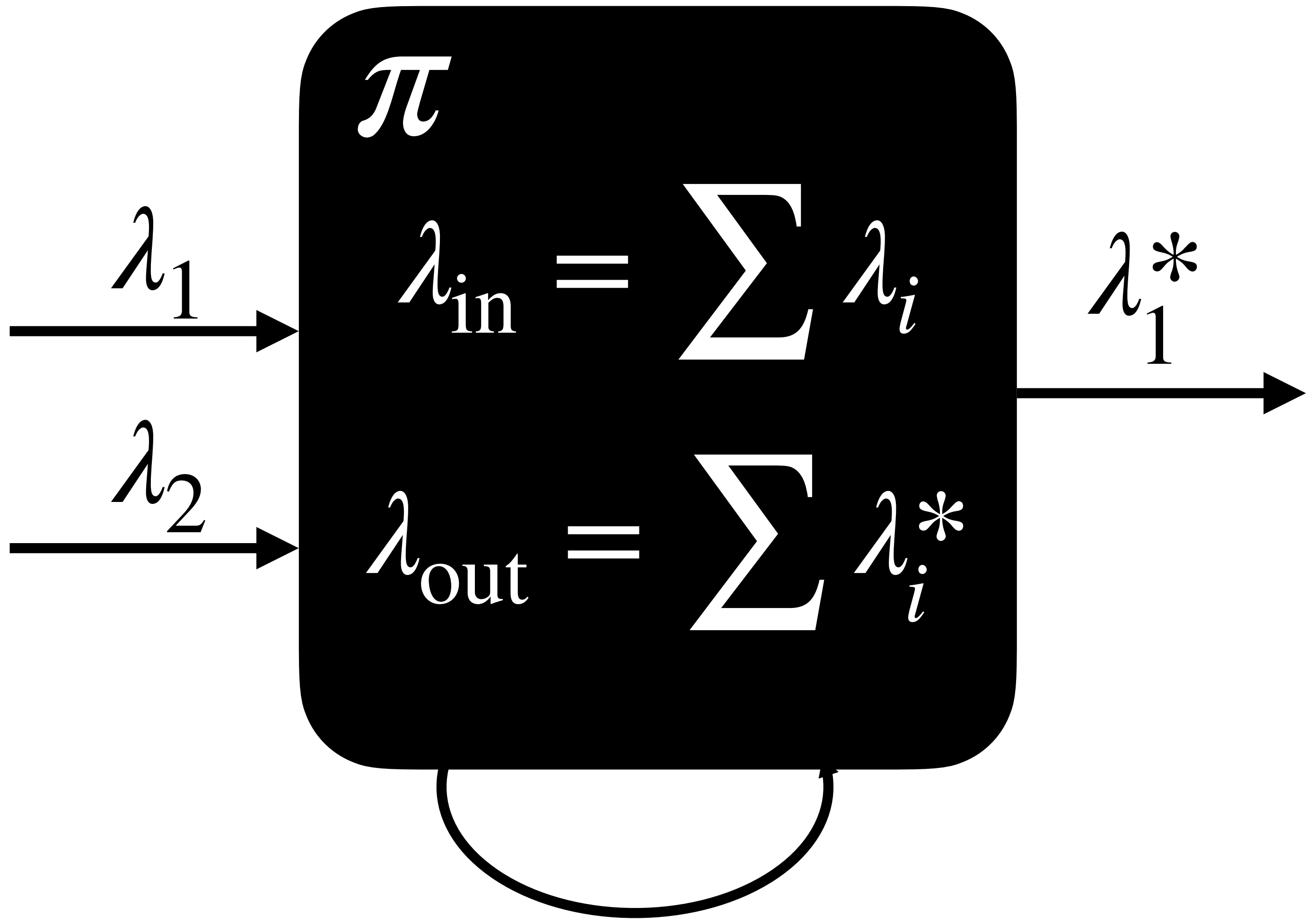
Plain UC only models adversaries that are **computationally** bounded using import



π
λ_1
λ_2
$\lambda_{\text{in}} = \sum \lambda_i$
$\lambda_{\text{out}} = \sum \lambda_i^*$
λ_1^*

UC with Budgets

Plain UC only models adversaries that are **computationally** bounded using import



π

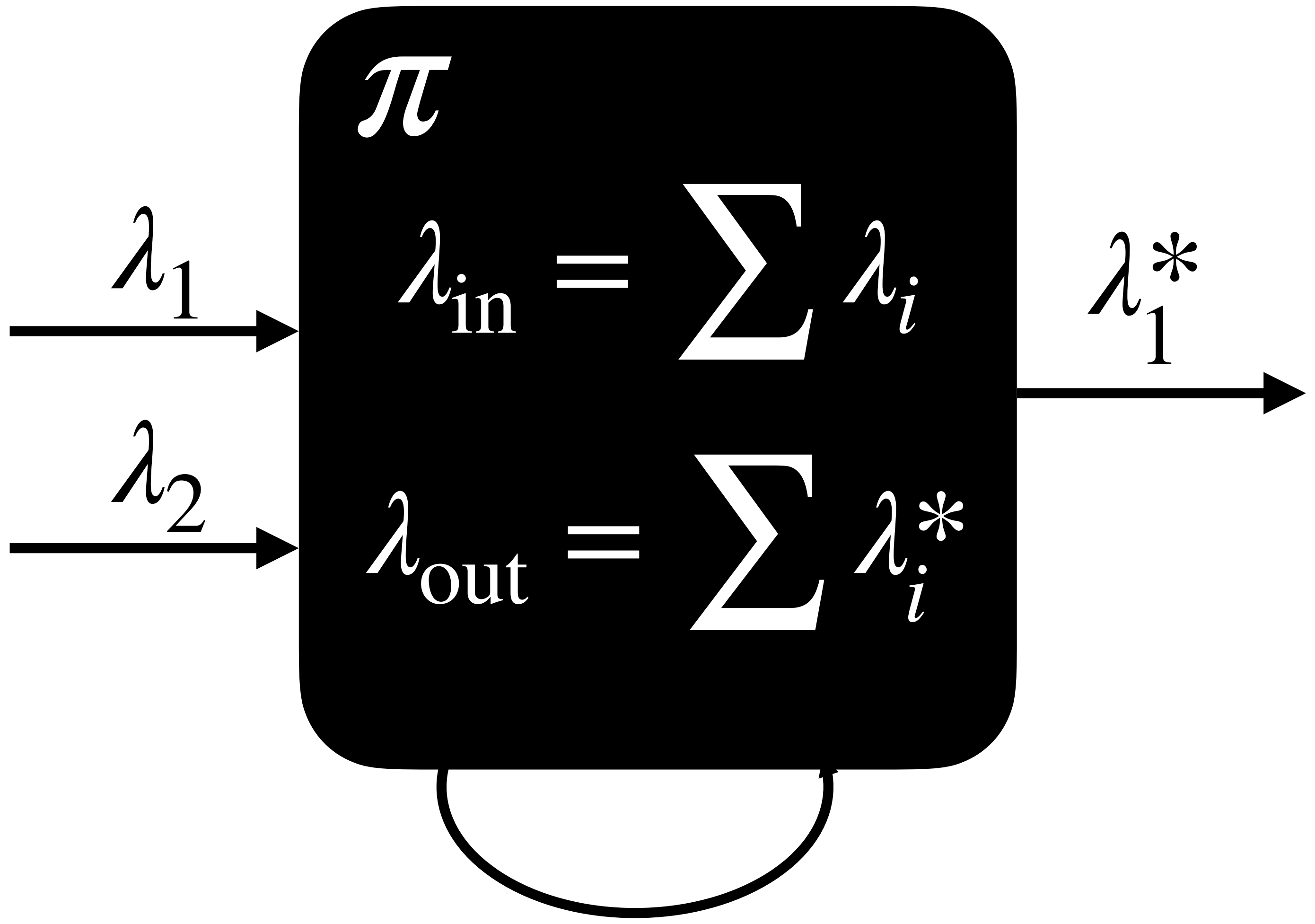
$\lambda_{\text{in}} = \sum \lambda_i$

$\lambda_{\text{out}} = \sum \lambda_i^*$

time$(\pi) \leq p(\lambda_{\text{in}} - \lambda_{\text{out}})$

UC with Budgets

Plain UC only models adversaries that are **computationally** bounded using import

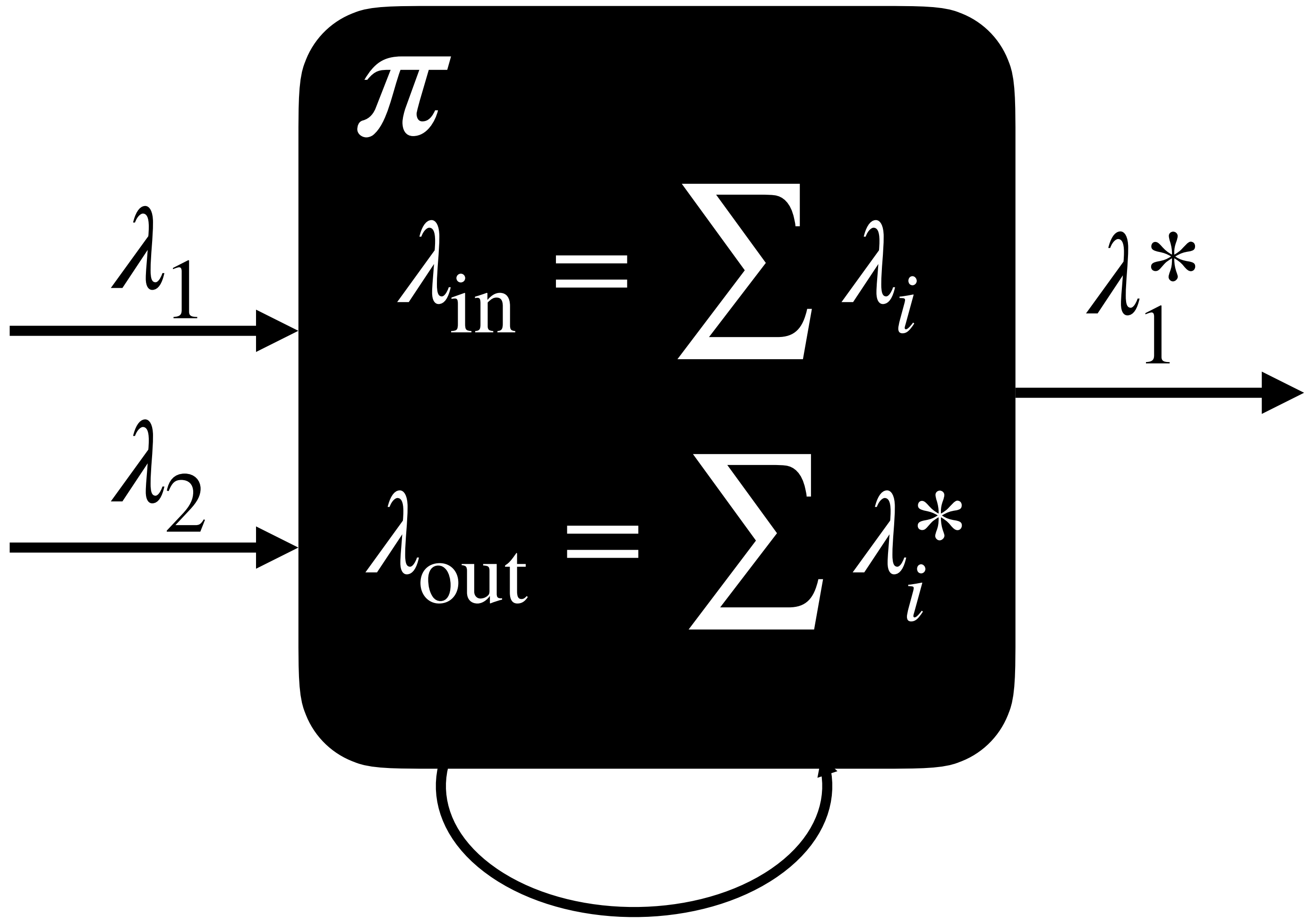

π
λ_1
λ_2
$\lambda_{\text{in}} = \sum \lambda_i$
$\lambda_{\text{out}} = \sum \lambda_i^*$
λ_1^*

time$(\pi) \leq p(\lambda_{\text{in}} - \lambda_{\text{out}})$

We consider adversaries that are **resource** bounded and computationally **unbounded.** We model this introducing budgets

UC with Budgets

Plain UC only models adversaries that are **computationally** bounded using import

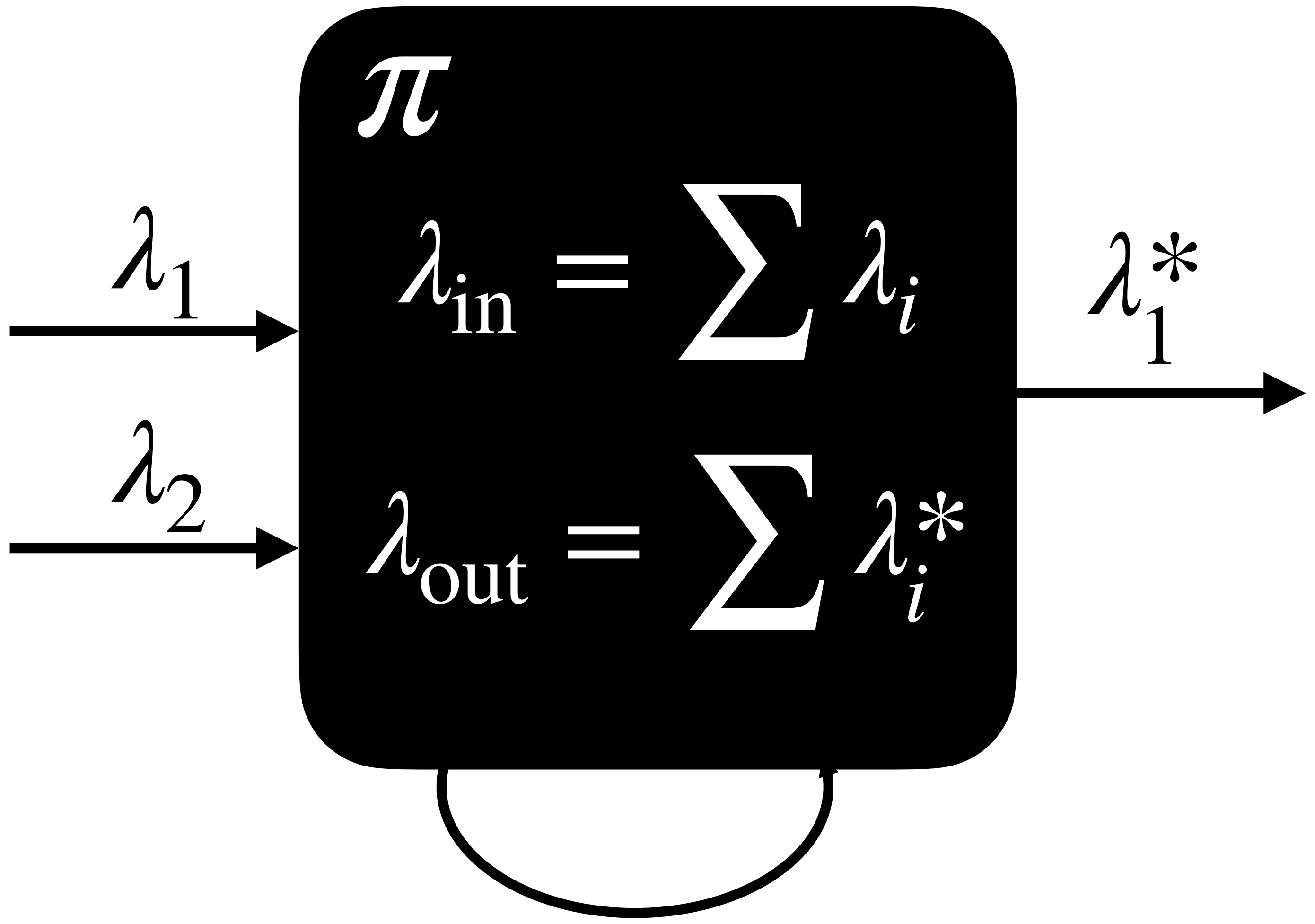

π
λ_1
λ_2
$\lambda_{\text{in}} = \sum \lambda_i$
$\lambda_{\text{out}} = \sum \lambda_i^*$
λ_1^*

time$(\pi) \leq p(\lambda_{\text{in}} - \lambda_{\text{out}})$

We consider adversaries that are **resource** bounded and computationally **unbounded.** We model this introducing budgets

UC with Budgets

Plain UC only models adversaries that are **computationally** bounded using import

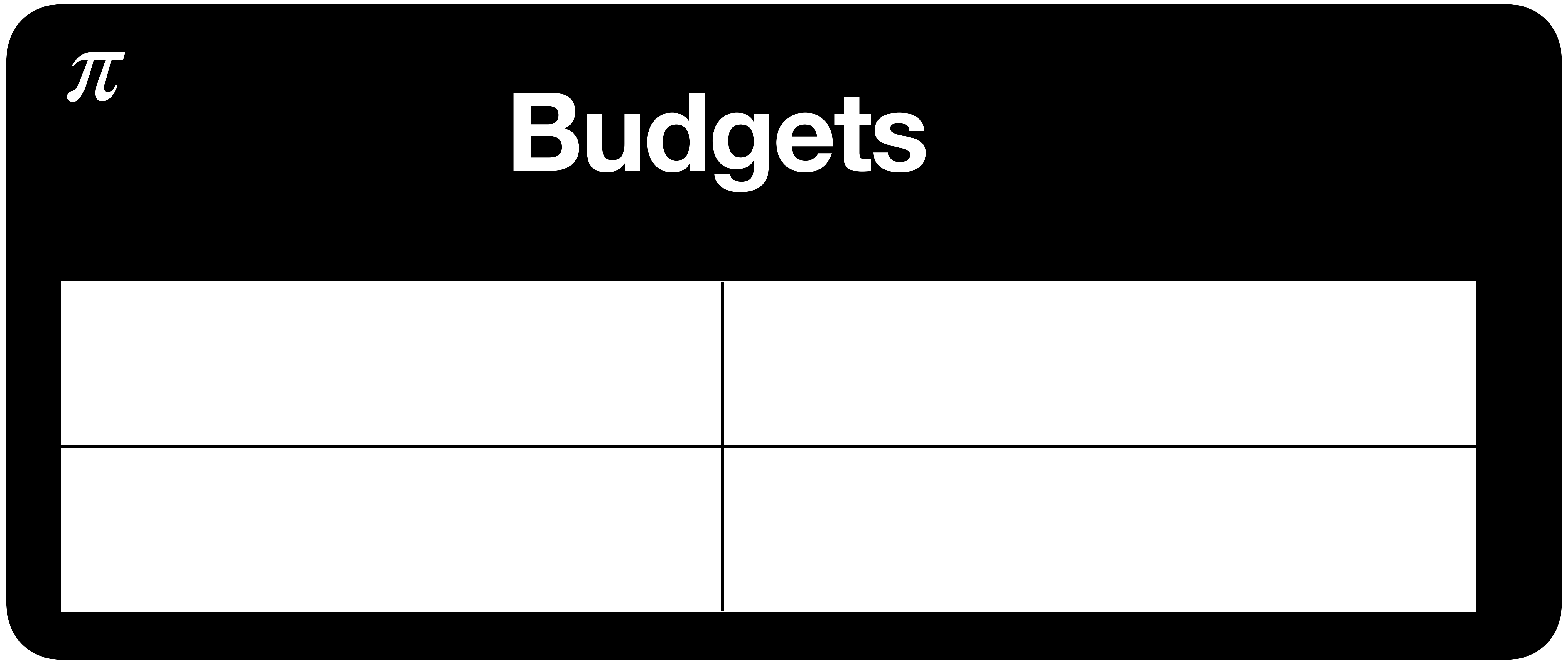


$$\lambda_{\text{in}} = \sum \lambda_i$$

$$\lambda_{\text{out}} = \sum \lambda_i^*$$

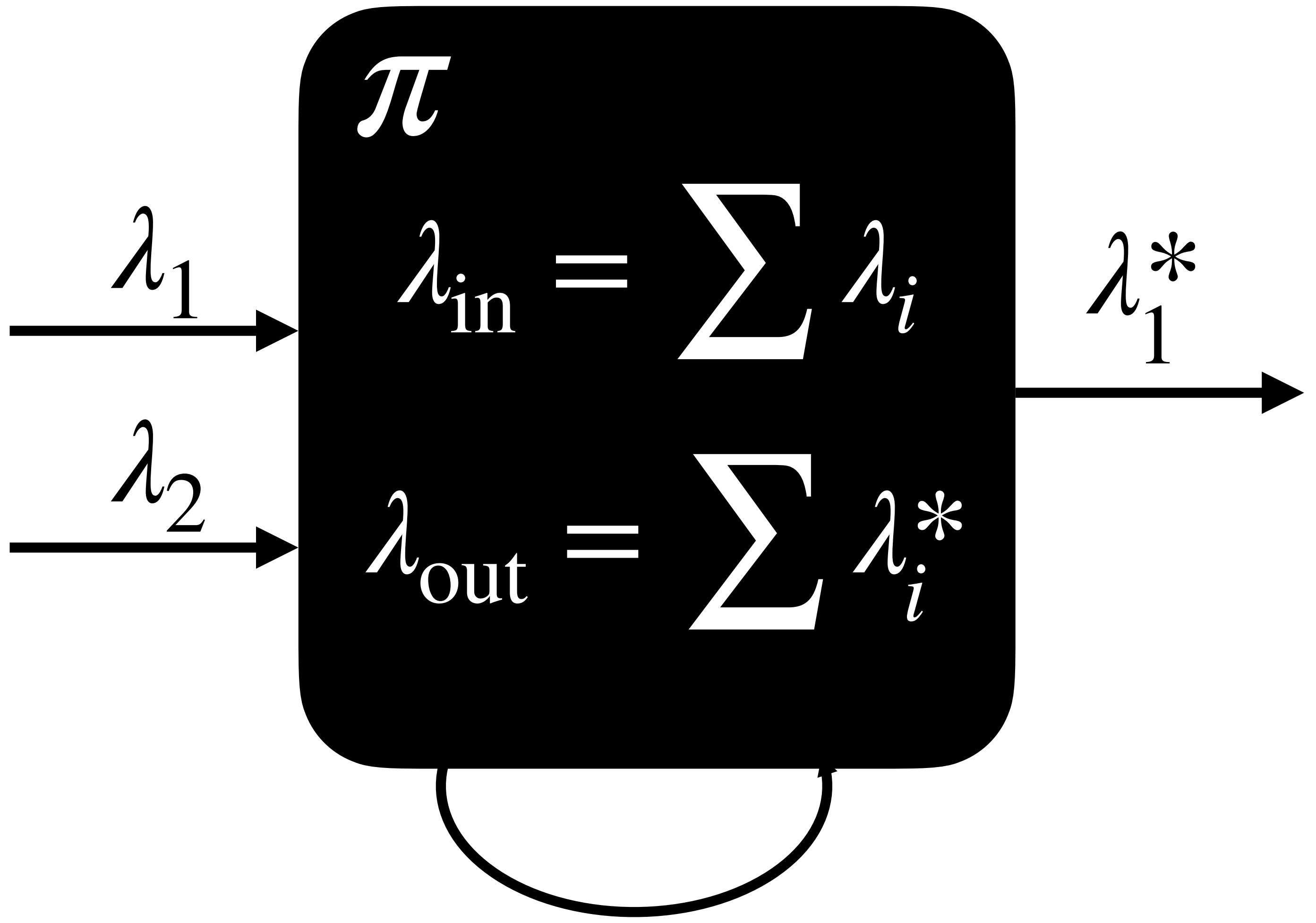
time(π) $\leq p(\lambda_{\text{in}} - \lambda_{\text{out}})$

We consider adversaries that are **resource** bounded and computationally **unbounded.** We model this introducing budgets



UC with Budgets

Plain UC only models adversaries that are **computationally** bounded using import


π
λ_1
λ_2
$\lambda_{\text{in}} = \sum \lambda_i$
$\lambda_{\text{out}} = \sum \lambda_i^*$
λ_1^*

$$\text{time}(\pi) \leq p(\lambda_{\text{in}} - \lambda_{\text{out}})$$

We consider adversaries that are **resource** bounded and computationally **unbounded.** We model this introducing budgets

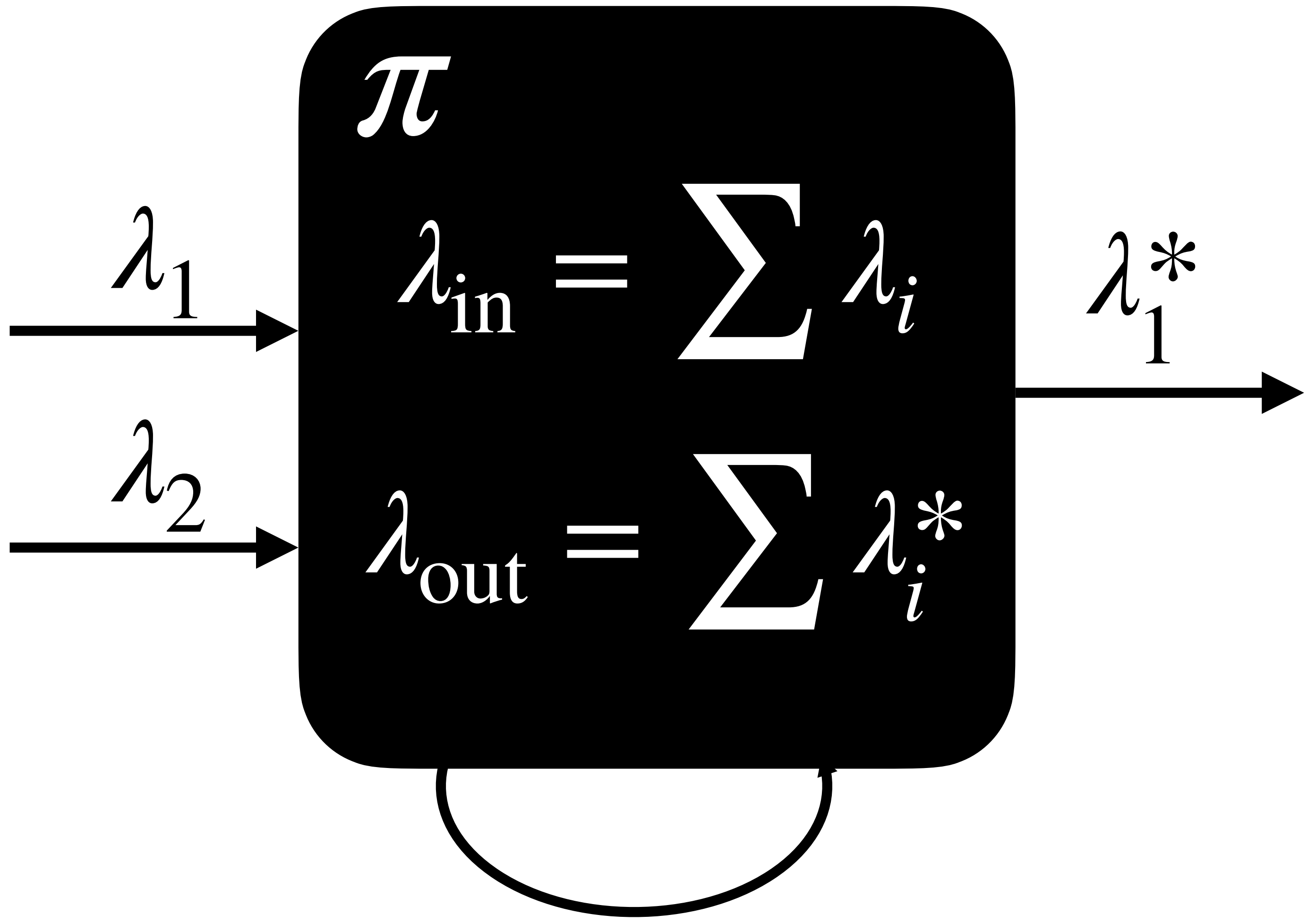
π

Budgets

t_q query	

UC with Budgets

Plain UC only models adversaries that are **computationally** bounded using import


π
λ_1
λ_2
$\lambda_{\text{in}} = \sum \lambda_i$
$\lambda_{\text{out}} = \sum \lambda_i^*$
λ_1^*

$\text{time}(\pi) \leq p(\lambda_{\text{in}} - \lambda_{\text{out}})$

We consider adversaries that are **resource** bounded and computationally **unbounded.** We model this introducing budgets

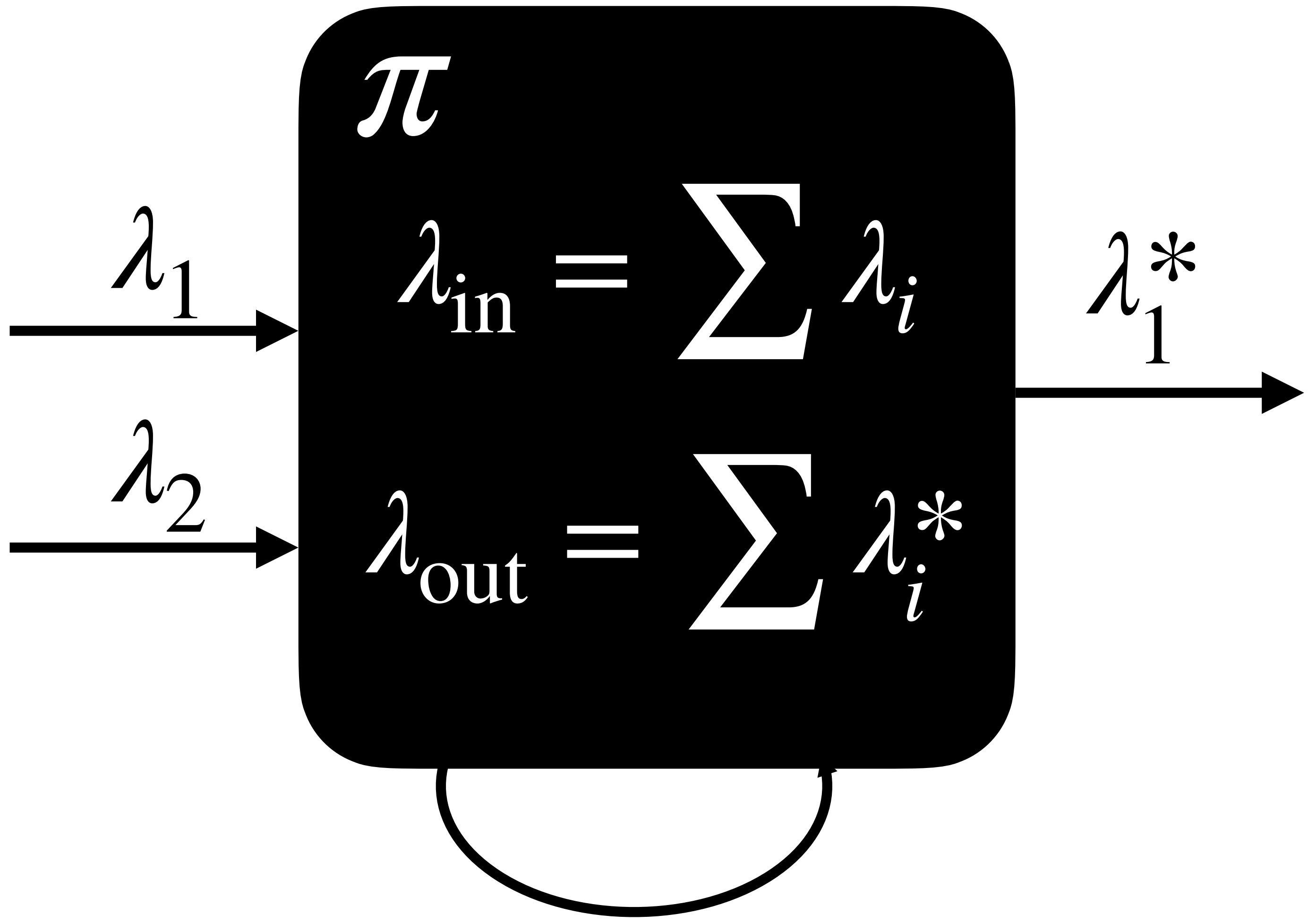
π

Budgets

t_{q} query	
t_{p} programming	

UC with Budgets

Plain UC only models adversaries that are **computationally** bounded using import



π
λ_1
λ_2
$\lambda_{\text{in}} = \sum \lambda_i$
$\lambda_{\text{out}} = \sum \lambda_i^*$
λ_1^*

$\text{time}(\pi) \leq p(\lambda_{\text{in}} - \lambda_{\text{out}})$

We consider adversaries that are **resource** bounded and computationally **unbounded.** We model this introducing budgets

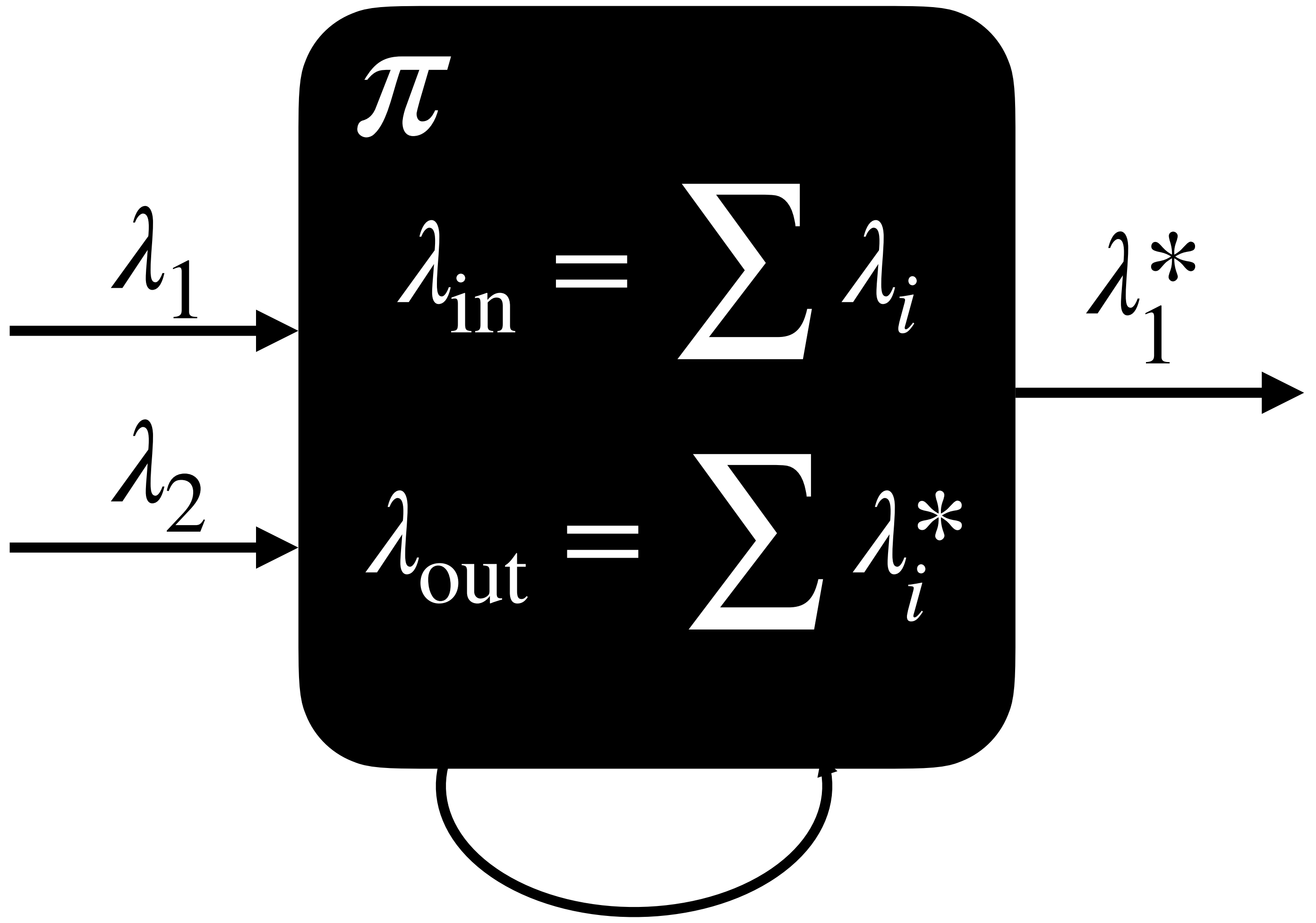
π

Budgets

t_{q} query	ℓ_{p} proving
t_{p} programming	

UC with Budgets

Plain UC only models adversaries that are **computationally** bounded using import


π
λ_1
λ_2
$\lambda_{\text{in}} = \sum \lambda_i$
$\lambda_{\text{out}} = \sum \lambda_i^*$
λ_1^*

$\text{time}(\pi) \leq p(\lambda_{\text{in}} - \lambda_{\text{out}})$

We consider adversaries that are **resource** bounded and computationally **unbounded.** We model this introducing budgets

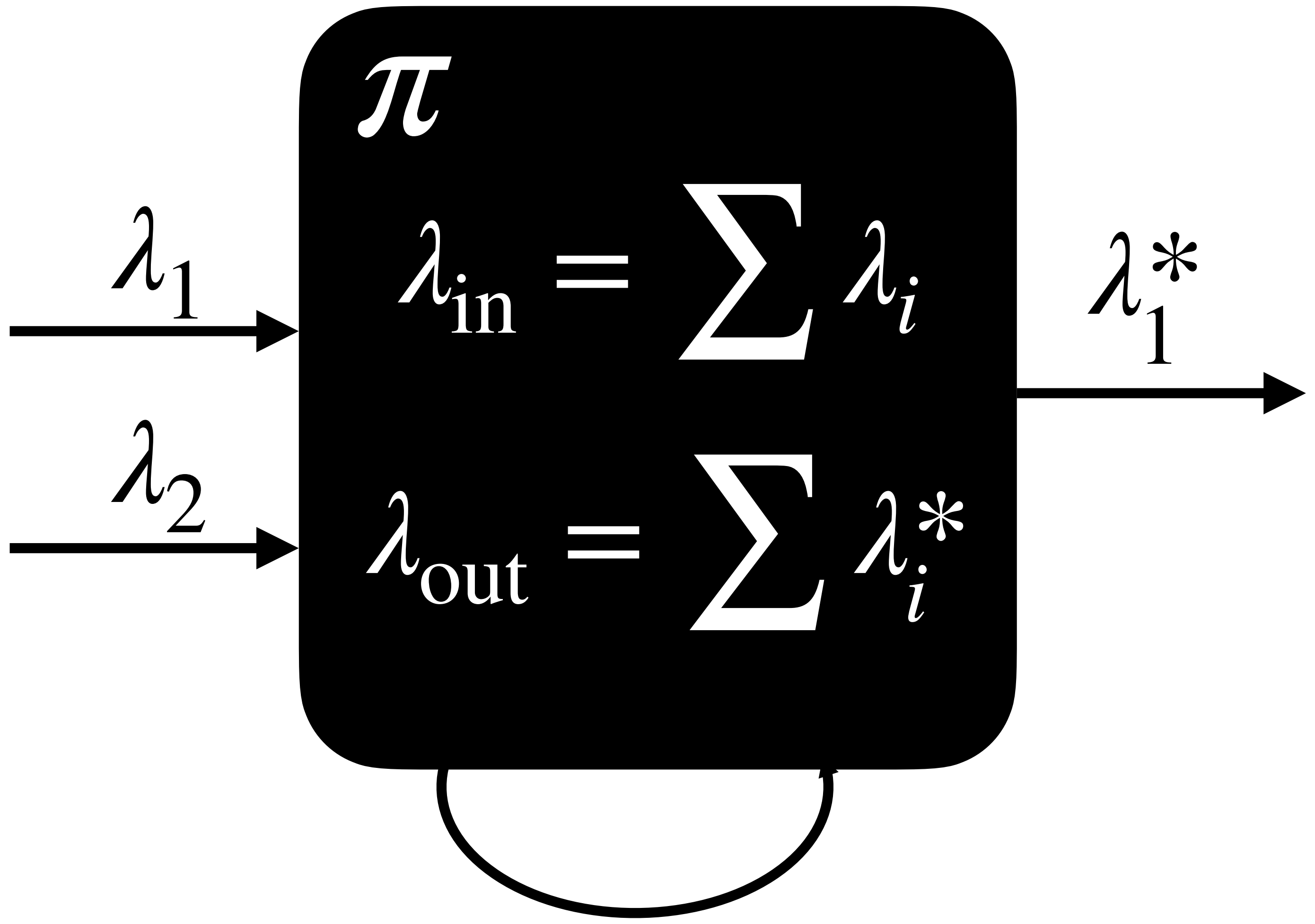
π

Budgets

t_{q} query	ℓ_{p} proving
t_{p} programming	ℓ_{v} verification

UC with Budgets

Plain UC only models adversaries that are **computationally** bounded using import



π
λ_1
λ_2
$\lambda_{\text{in}} = \sum \lambda_i$
$\lambda_{\text{out}} = \sum \lambda_i^*$
λ_1^*

time$(\pi) \leq p(\lambda_{\text{in}} - \lambda_{\text{out}})$

We consider adversaries that are **resource** bounded and computationally **unbounded**. We model this introducing budgets

π

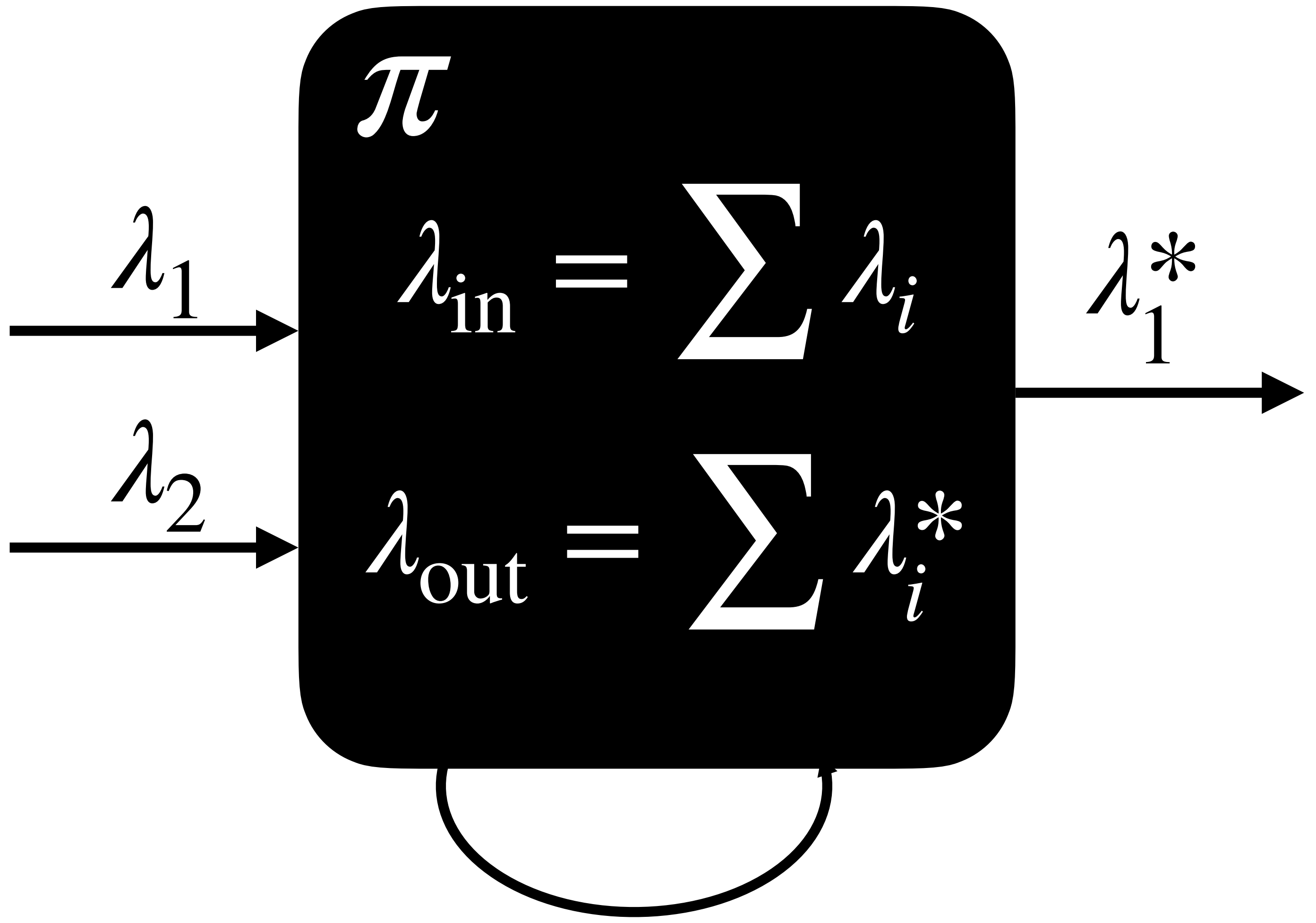
Budgets

t_{q} query	ℓ_{p} proving
t_{p} programming	ℓ_{v} verification

Budget can then be spent on:

UC with Budgets

Plain UC only models adversaries that are **computationally** bounded using import


π
λ_1
λ_2
$\lambda_{\text{in}} = \sum \lambda_i$
$\lambda_{\text{out}} = \sum \lambda_i^*$
λ_1^*

$\text{time}(\pi) \leq p(\lambda_{\text{in}} - \lambda_{\text{out}})$

We consider adversaries that are **resource** bounded and computationally **unbounded**. We model this introducing budgets

π

Budgets

t_{q} query	ℓ_{p} proving
t_{p} programming	ℓ_{v} verification

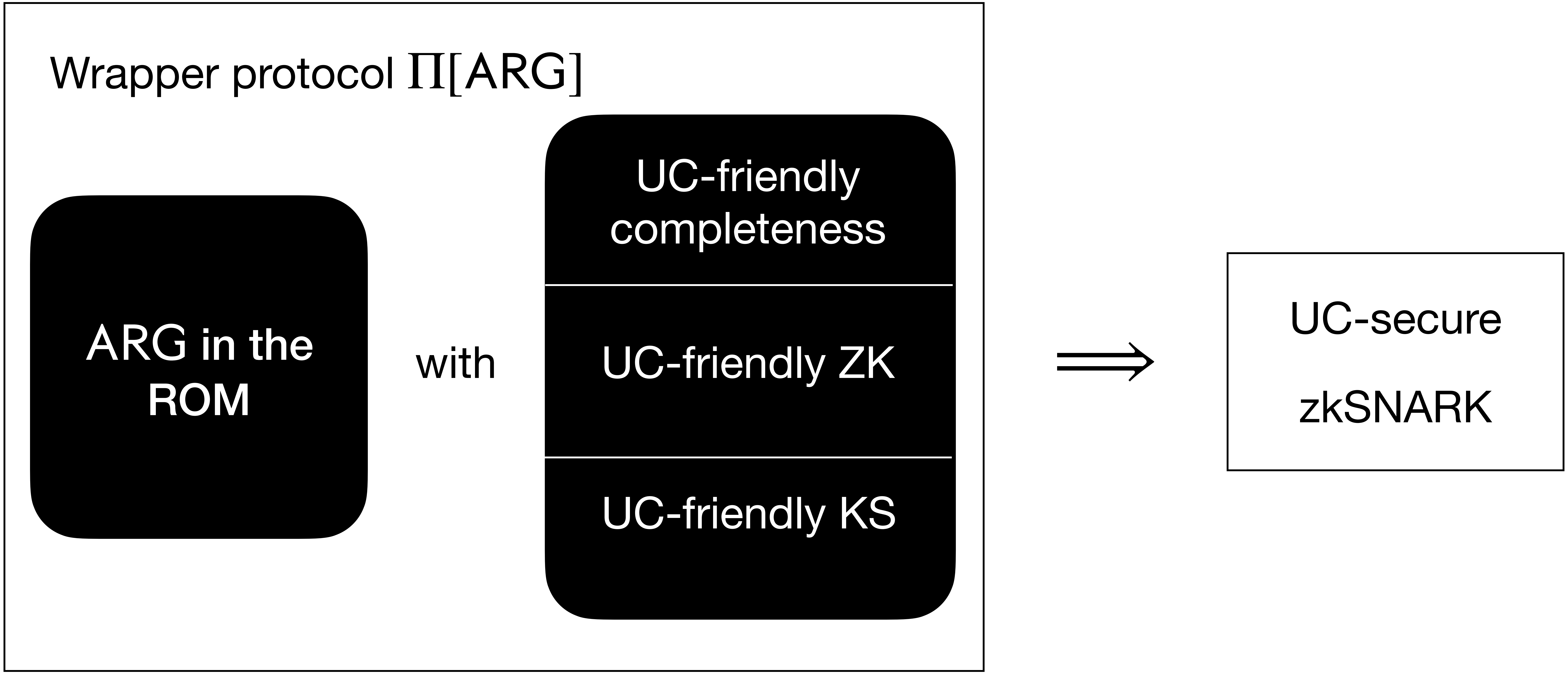
Budget can then be spent on:

GRO

Prove

Verify

Our main lemma



Wrapper protocol $\Pi[\mathrm{ARG}]$

ARG in the ROM

with

UC-friendly completeness

UC-friendly ZK

UC-friendly KS

⟹

UC-secure

zkSNARK

UC-friendly $\Longrightarrow$ UC-secure

UC-friendly $\Longrightarrow$ UC-secure

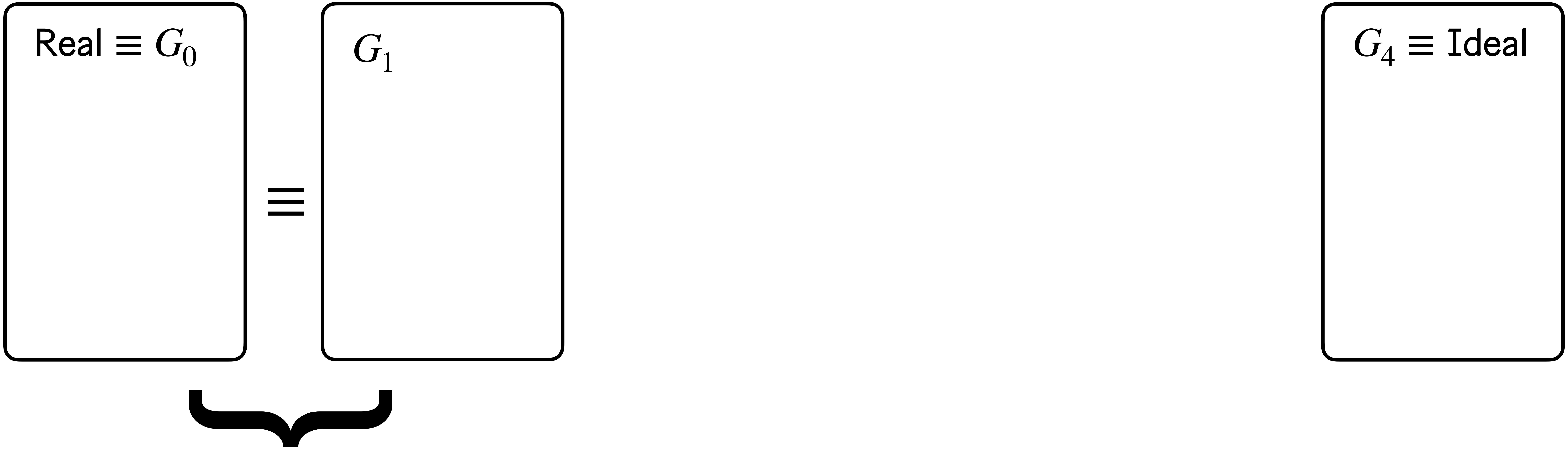
Real $\equiv G_0$

UC-friendly $\implies$ UC-secure

Real $\equiv G_0$

$G_4 \equiv$ Ideal

UC-friendly $\Longrightarrow$ UC-secure



$\mathsf{Real} \equiv G_0$

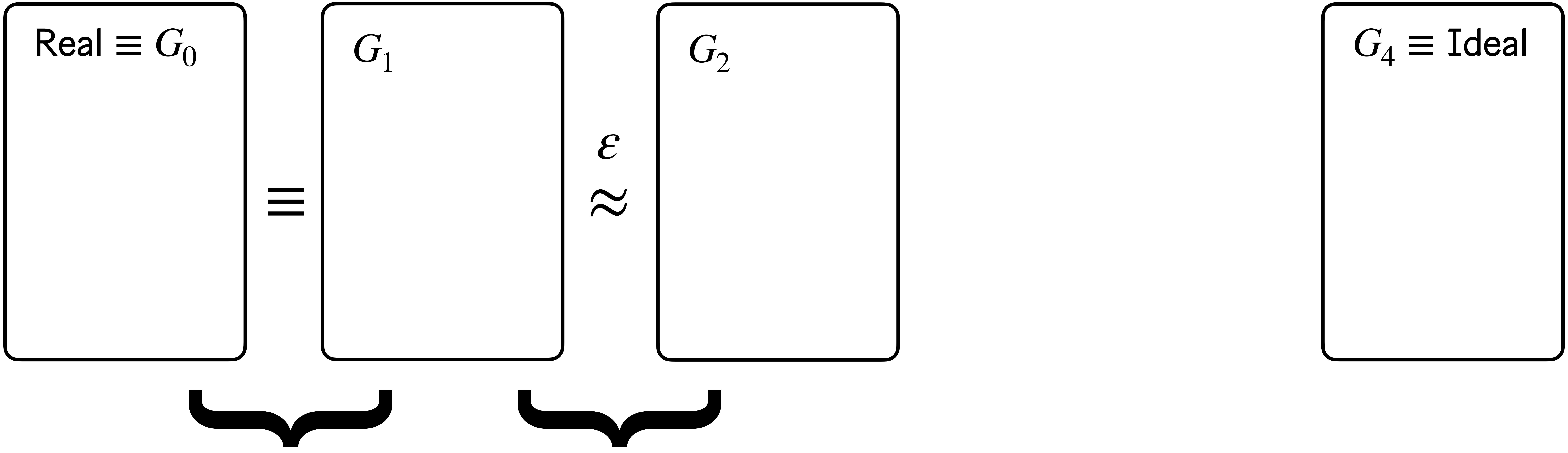
$\equiv$

G_1

$G_4 \equiv \mathtt{Ideal}$

Simulator can program undetectably

UC-friendly $\implies$ UC-secure



$\mathsf{Real} \equiv G_0$

$\equiv$

G_1

$\overset{\varepsilon}{\approx}$

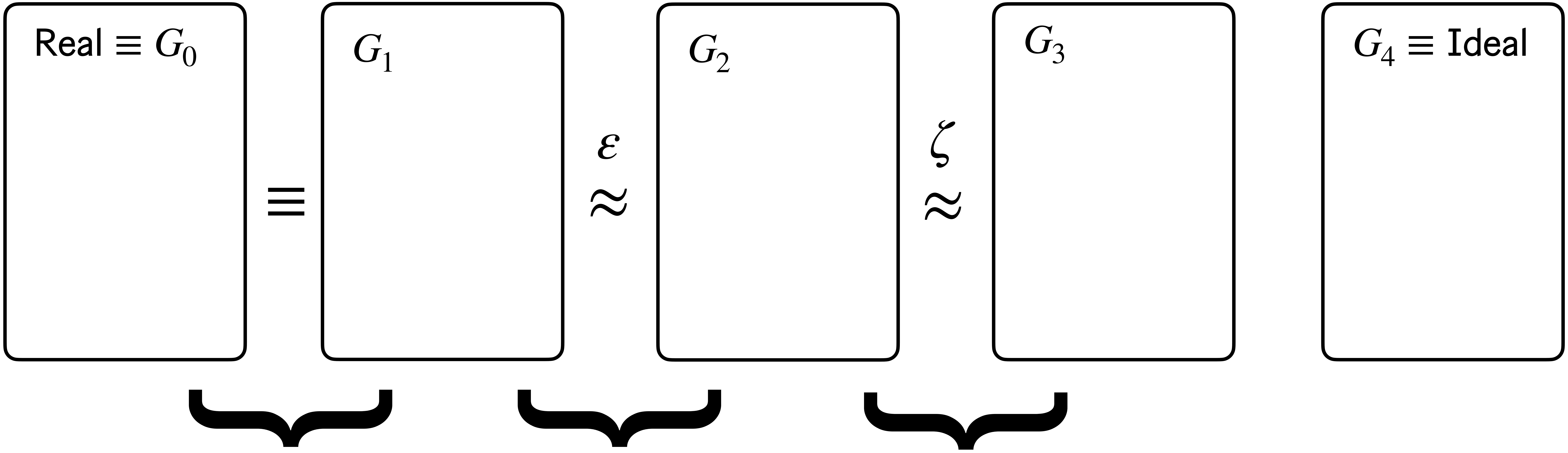
G_2

$G_4 \equiv \mathsf{Ideal}$

Simulator can program undetectably

UC-friendly completeness

UC-friendly $\Longrightarrow$ UC-secure



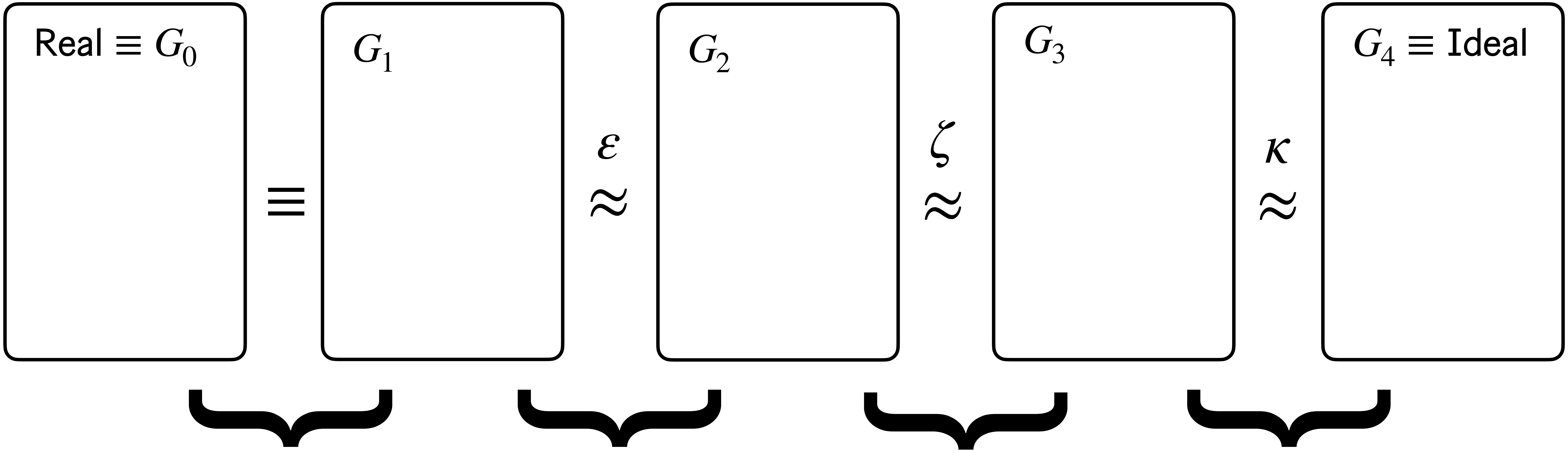
$\text{Real} \equiv G_0$ $\equiv$ G_1 $\overset{\varepsilon}{\approx}$ G_2 $\overset{\zeta}{\approx}$ G_3 $\quad$ $G_4 \equiv \texttt{Ideal}$

Simulator can program undetectably

UC-friendly completeness

UC-friendly ZK

UC-friendly $\implies$ UC-secure



$\text{Real} \equiv G_0$ $\equiv$ G_1 $\overset{\varepsilon}{\approx}$ G_2 $\overset{\zeta}{\approx}$ G_3 $\overset{\kappa}{\approx}$ $G_4 \equiv \text{Ideal}$

Simulator can program undetectably

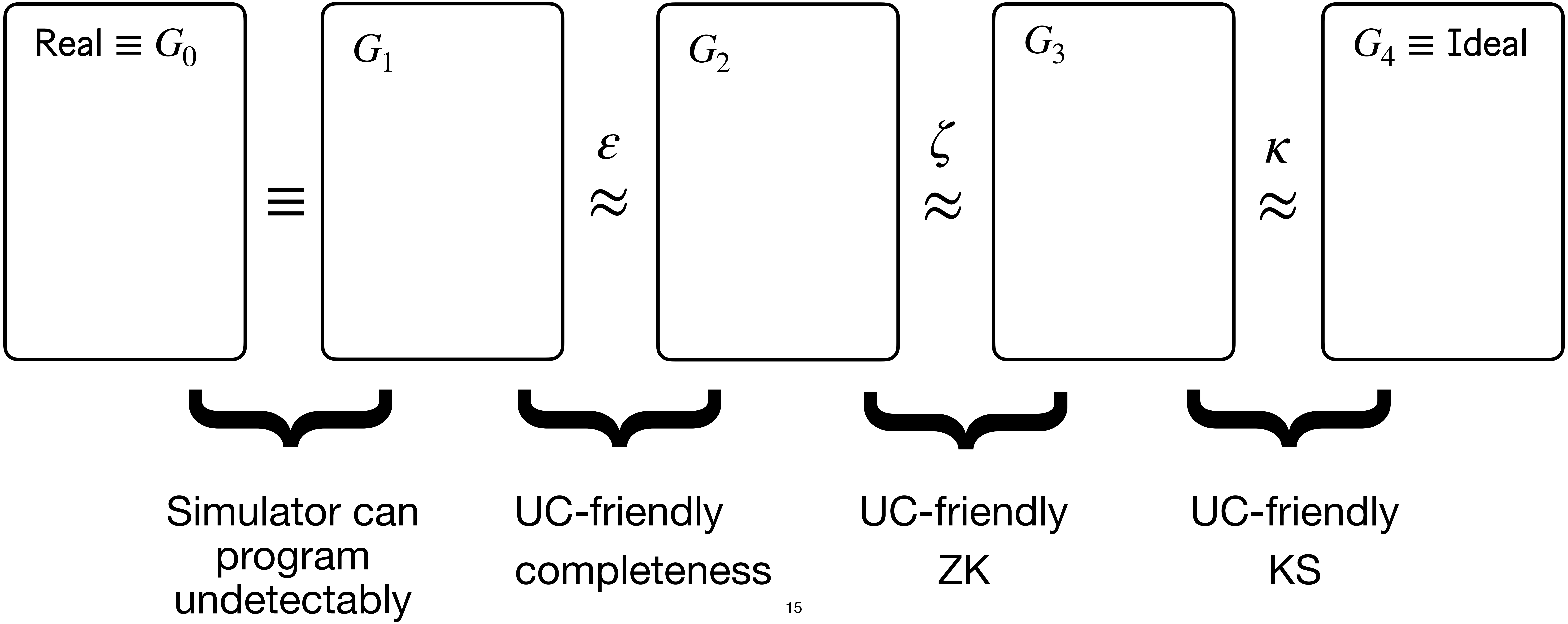
UC-friendly completeness

UC-friendly ZK

UC-friendly KS

UC-friendly $\Longrightarrow$ UC-secure

UC-friendly properties exactly defined for these game hops



Real $\equiv G_0$ $\equiv$ G_1 $\overset{\varepsilon}{\approx}$ G_2 $\overset{\zeta}{\approx}$ G_3 $\overset{\kappa}{\approx}$ $G_4 \equiv$ Ideal

Simulator can program undetectably

UC-friendly completeness

UC-friendly ZK

UC-friendly KS

UC-friendly completeness

Adversary should not be able to make honestly generated proofs fail to verify.

UC-friendly completeness

Adversary should not be able to make honestly generated proofs fail to verify.

$\boxed{\forall \mathscr{A}}$

UC-friendly completeness

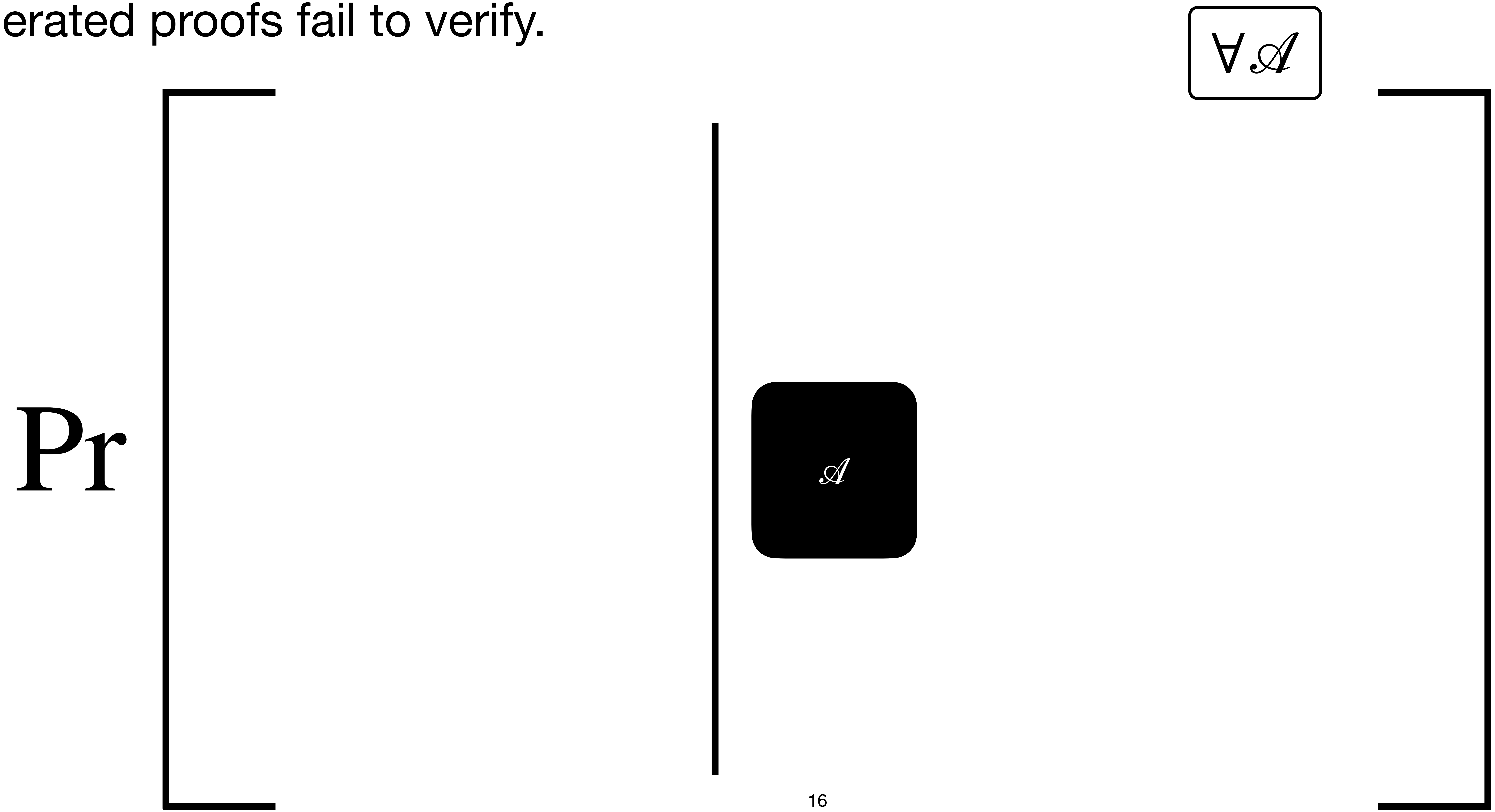
Adversary should not be able to make honestly generated proofs fail to verify.

$\boxed{\forall \mathscr{A}}$

$$\Pr\left[\qquad \middle| \qquad \right]$$

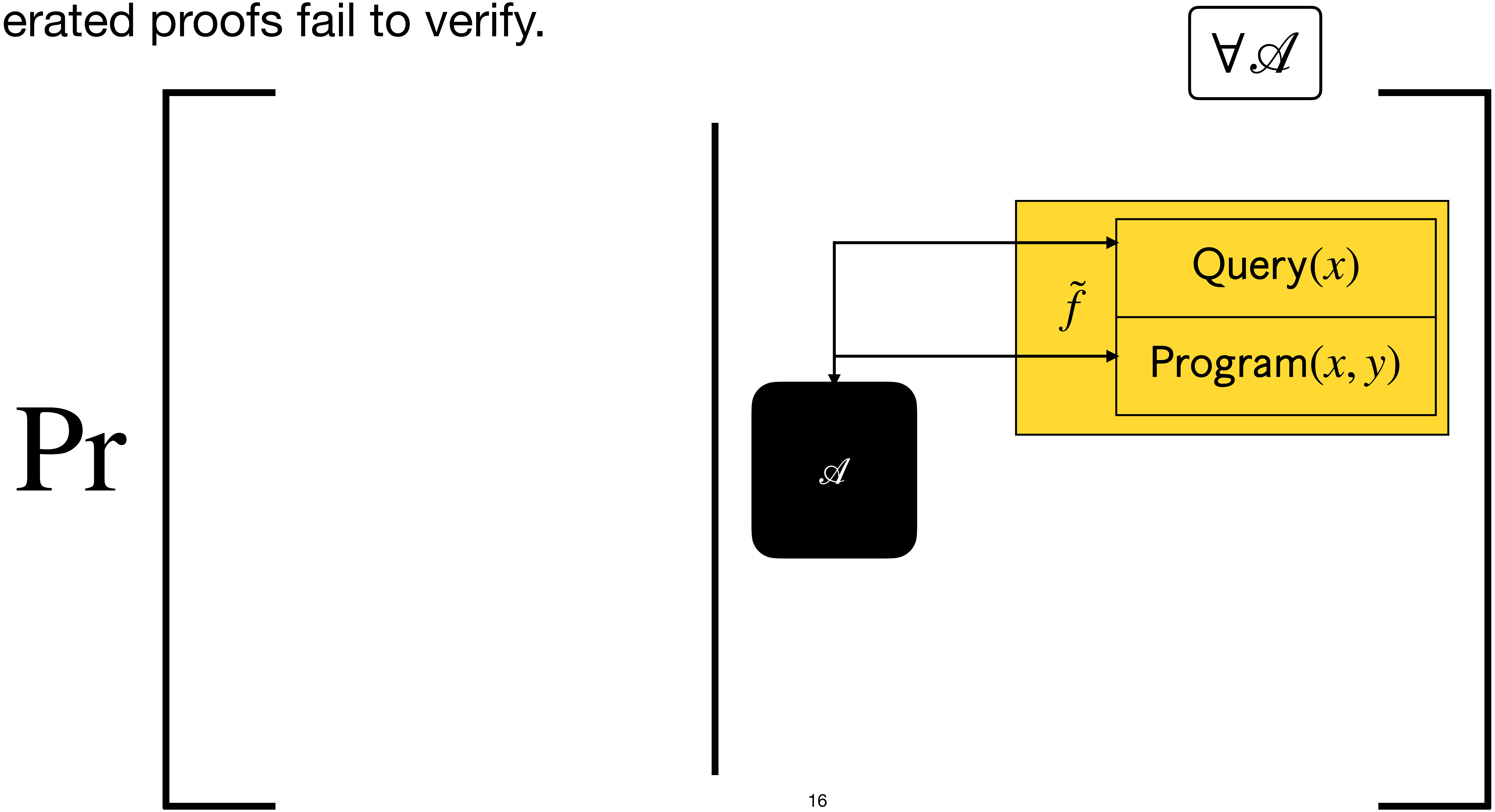
UC-friendly completeness

Adversary should not be able to make honestly generated proofs fail to verify.



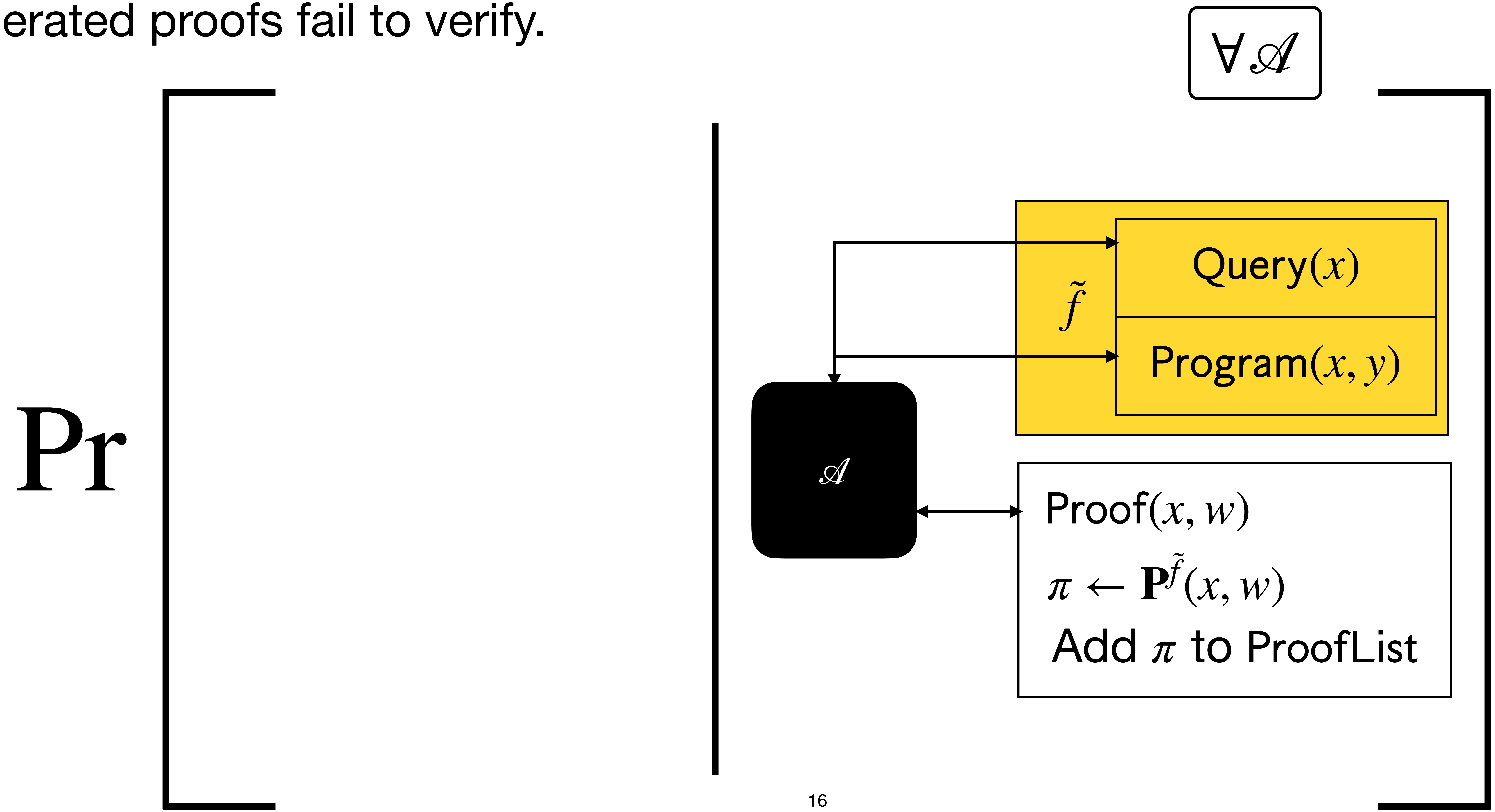
UC-friendly completeness

Adversary should not be able to make honestly generated proofs fail to verify.



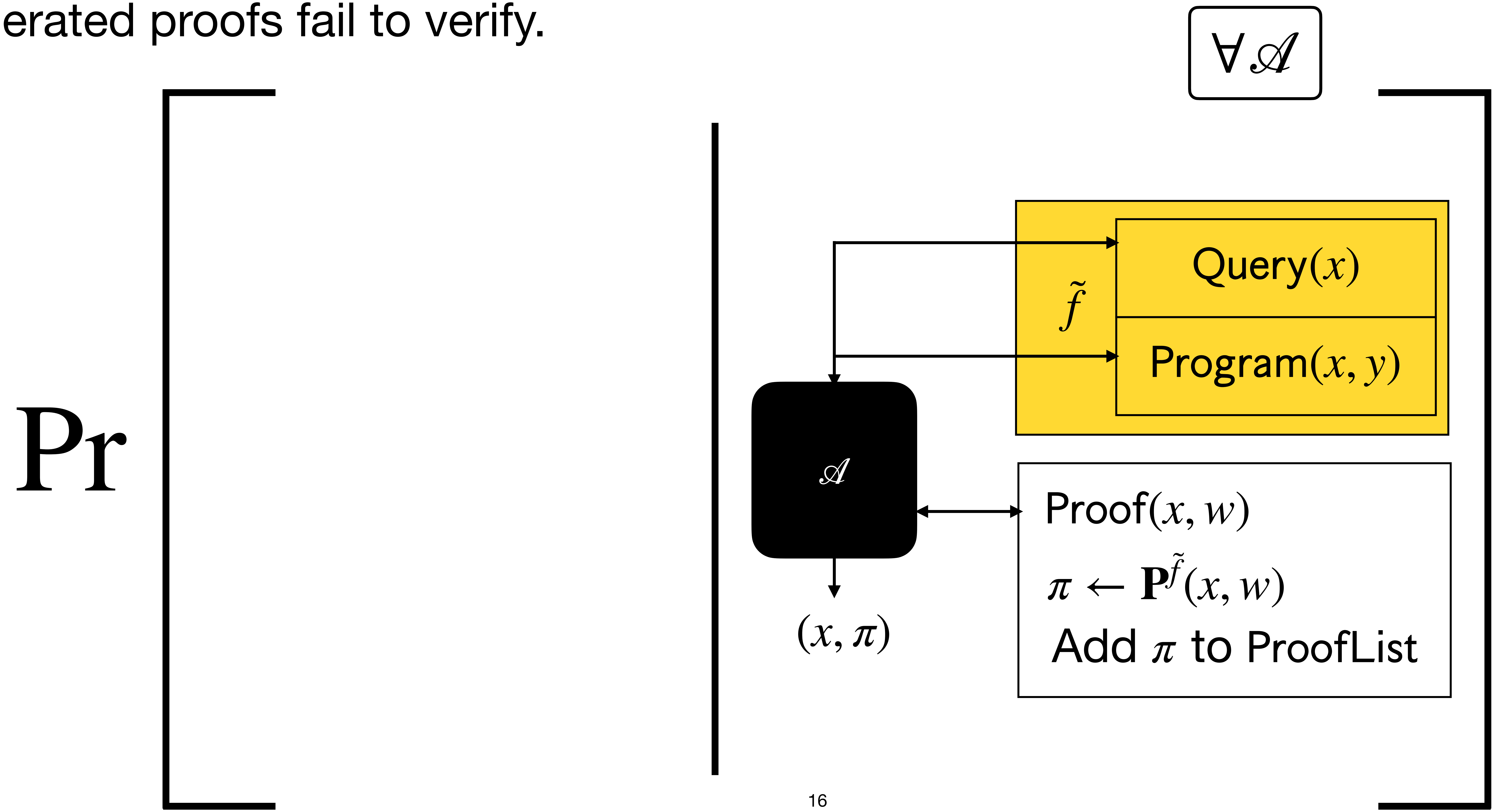
UC-friendly completeness

Adversary should not be able to make honestly generated proofs fail to verify.



UC-friendly completeness

Adversary should not be able to make honestly generated proofs fail to verify.



$\forall \mathscr{A}$

Pr

$\tilde{f}$

Query(x)

Program(x, y)

$\mathscr{A}$

(x, π)

Proof(x, w)

$\pi \leftarrow \mathbf{P}^{\tilde{f}}(x, w)$

Add π to ProofList

UC-friendly completeness

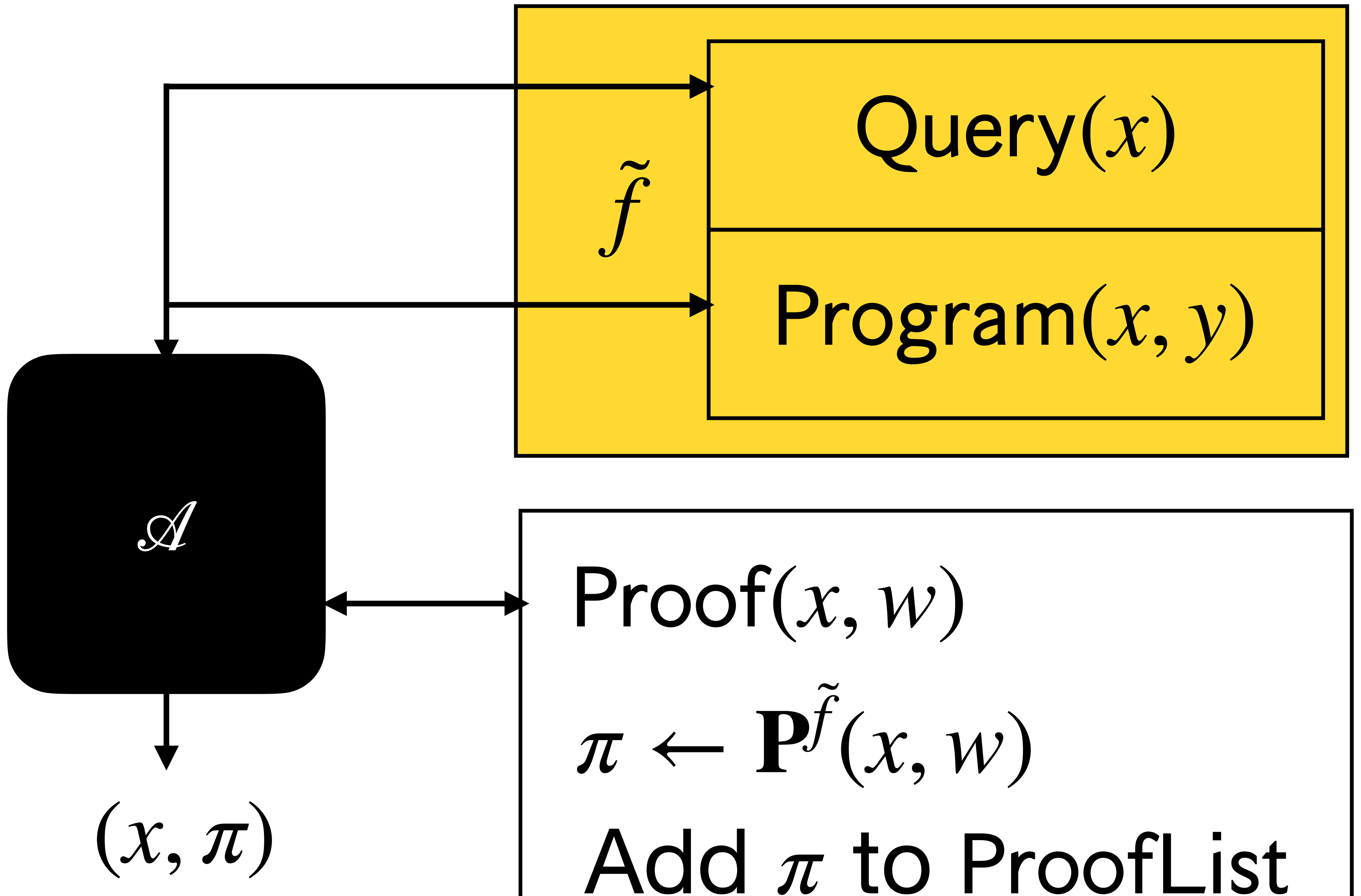
Adversary should not be able to make honestly generated proofs fail to verify.

$\forall \mathscr{A}$

$\Pr$ [$(x, \pi) \in \mathsf{ProofList}$

and

|



Proof(x, w)

$\pi \leftarrow \mathbf{P}^{\tilde{f}}(x, w)$

Add π to ProofList

]

UC-friendly completeness

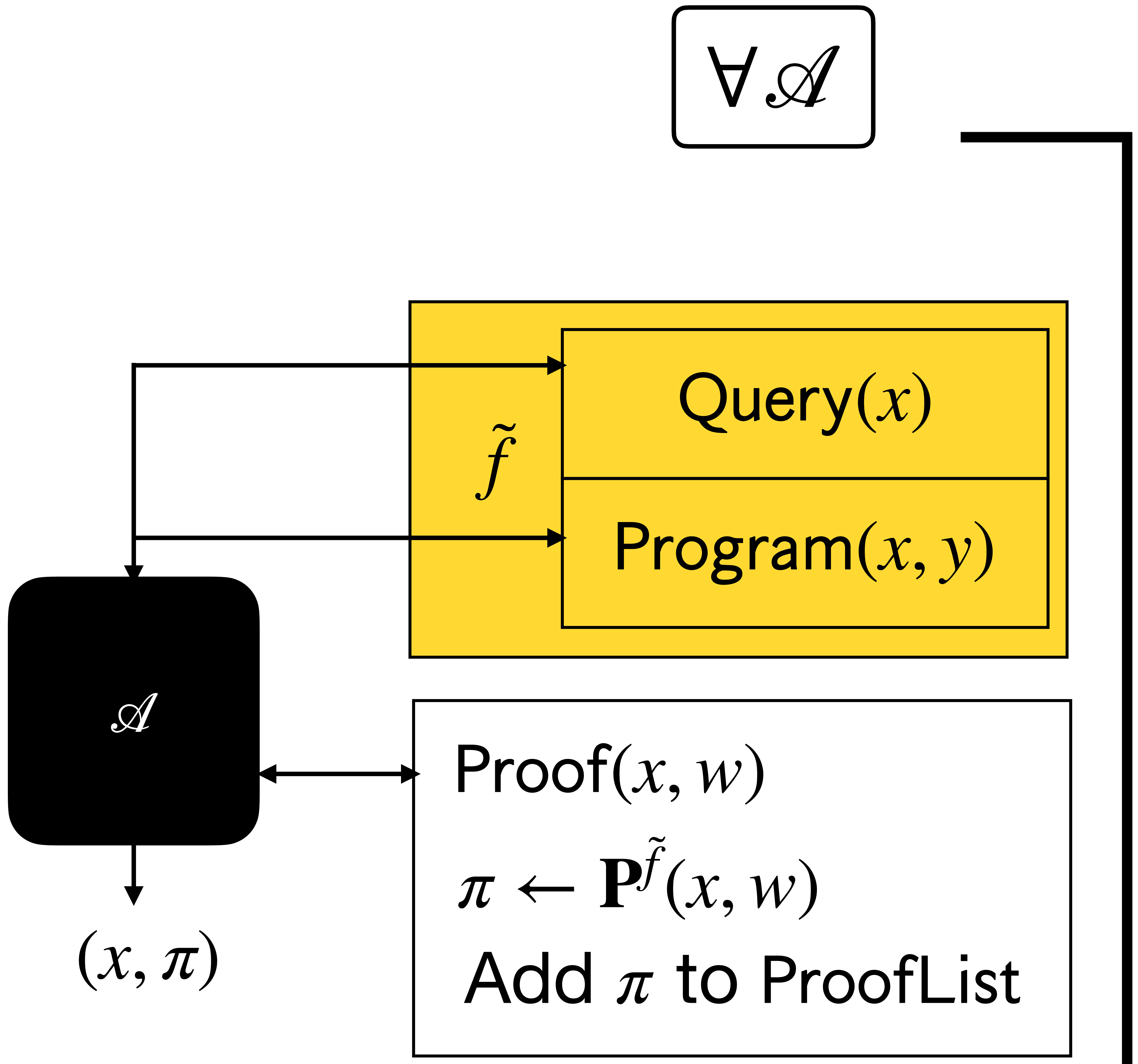
Adversary should not be able to make honestly generated proofs fail to verify.

$\Pr$

$(x, \pi) \in \text{ProofList}$

and

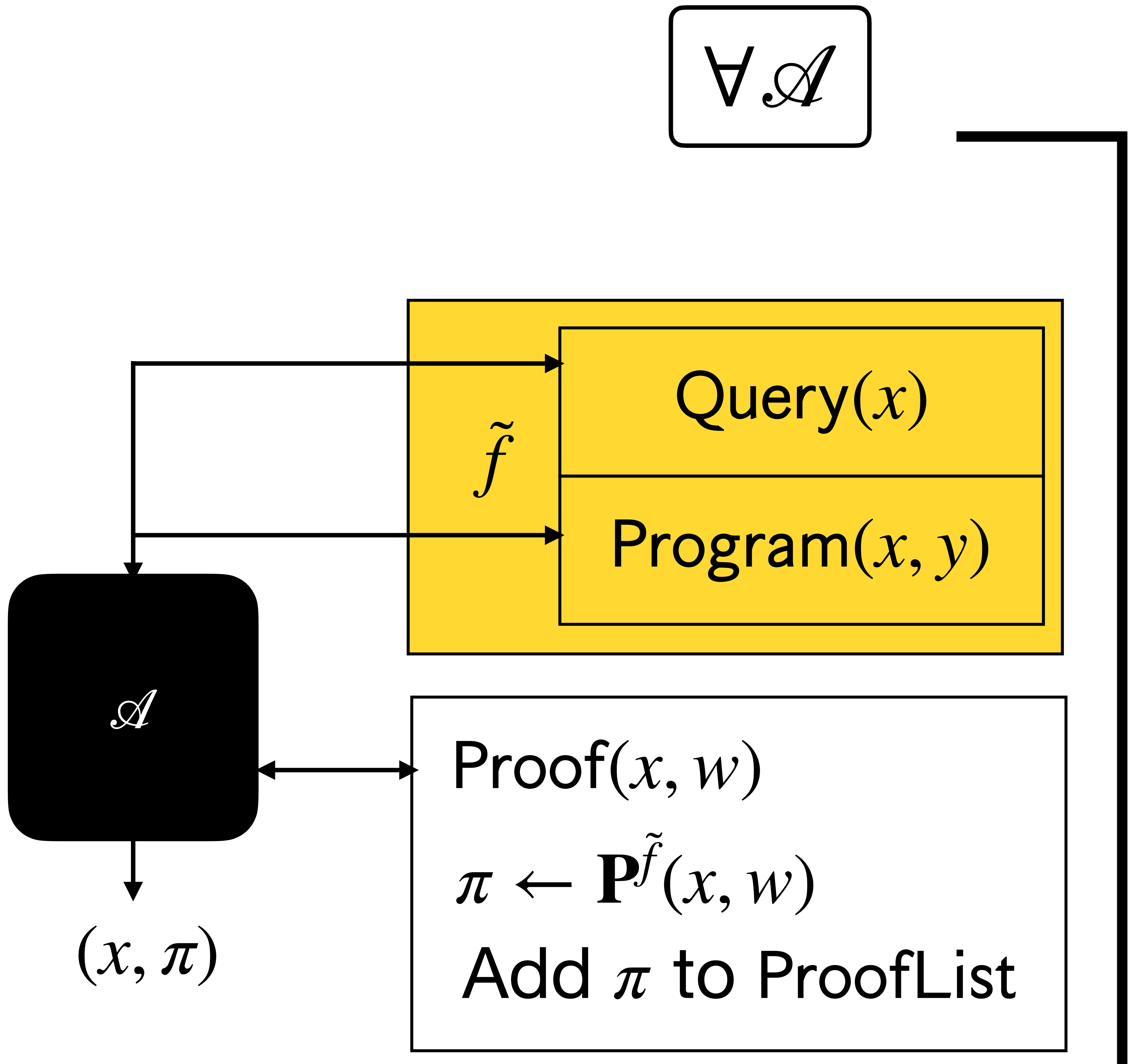
$\mathbf{V}^{\tilde{f}}(x, \pi) \neq 1$



UC-friendly completeness

Adversary should not be able to make honestly generated proofs fail to verify.

$$\Pr\left[\begin{array}{c} (x, \pi) \in \text{ProofList} \\ \text{and} \\ \left(\begin{array}{c} \boxed{\mathbf{V}^{\tilde{f}}(x, \pi) \neq 1} \\ \text{or} \\ \mathbf{V} \text{ queries} \\ \text{programmed points} \end{array} \right) \end{array} \;\middle|\; \right]$$



$\forall \mathscr{A}$

$\tilde{f}$

Query(x)

Program(x, y)

$\mathscr{A}$

(x, π)

Proof(x, w)

$\pi \leftarrow \mathbf{P}^{\tilde{f}}(x, w)$

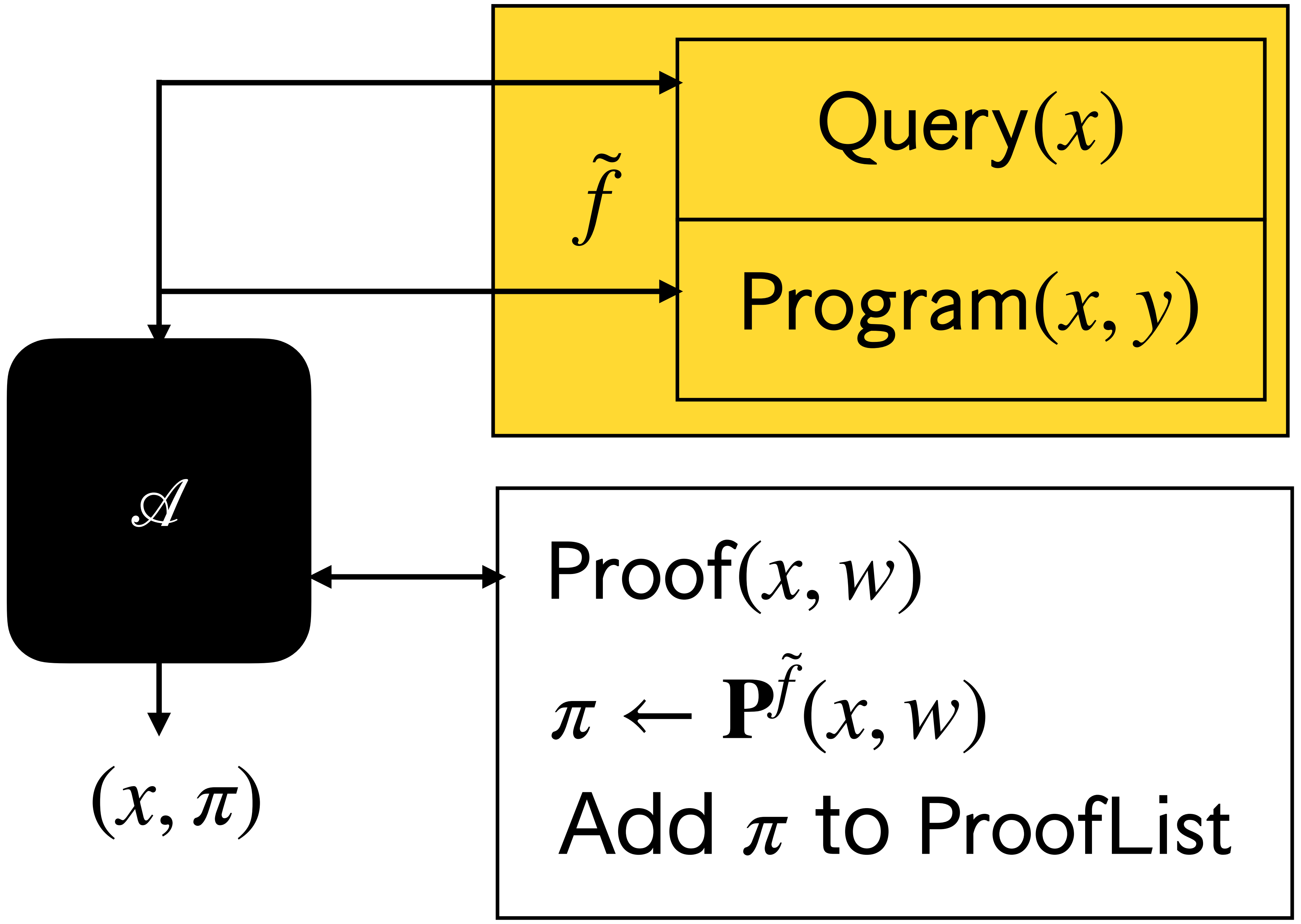
Add π to ProofList

UC-friendly completeness

Adversary should not be able to make honestly generated proofs fail to verify.

$\forall \mathscr{A}$

$$\Pr\left[\begin{array}{c}(x,\pi)\in\text{ProofList}\\ \text{and}\\ \left(\begin{array}{c}\boxed{\mathbf{V}^{\tilde{f}}(x,\pi)\neq 1}\\ \text{or}\\ \mathbf{V}\text{ queries}\\ \text{programmed points}\end{array}\right)\end{array}\ \middle|\ \quad\right] \le \varepsilon$$

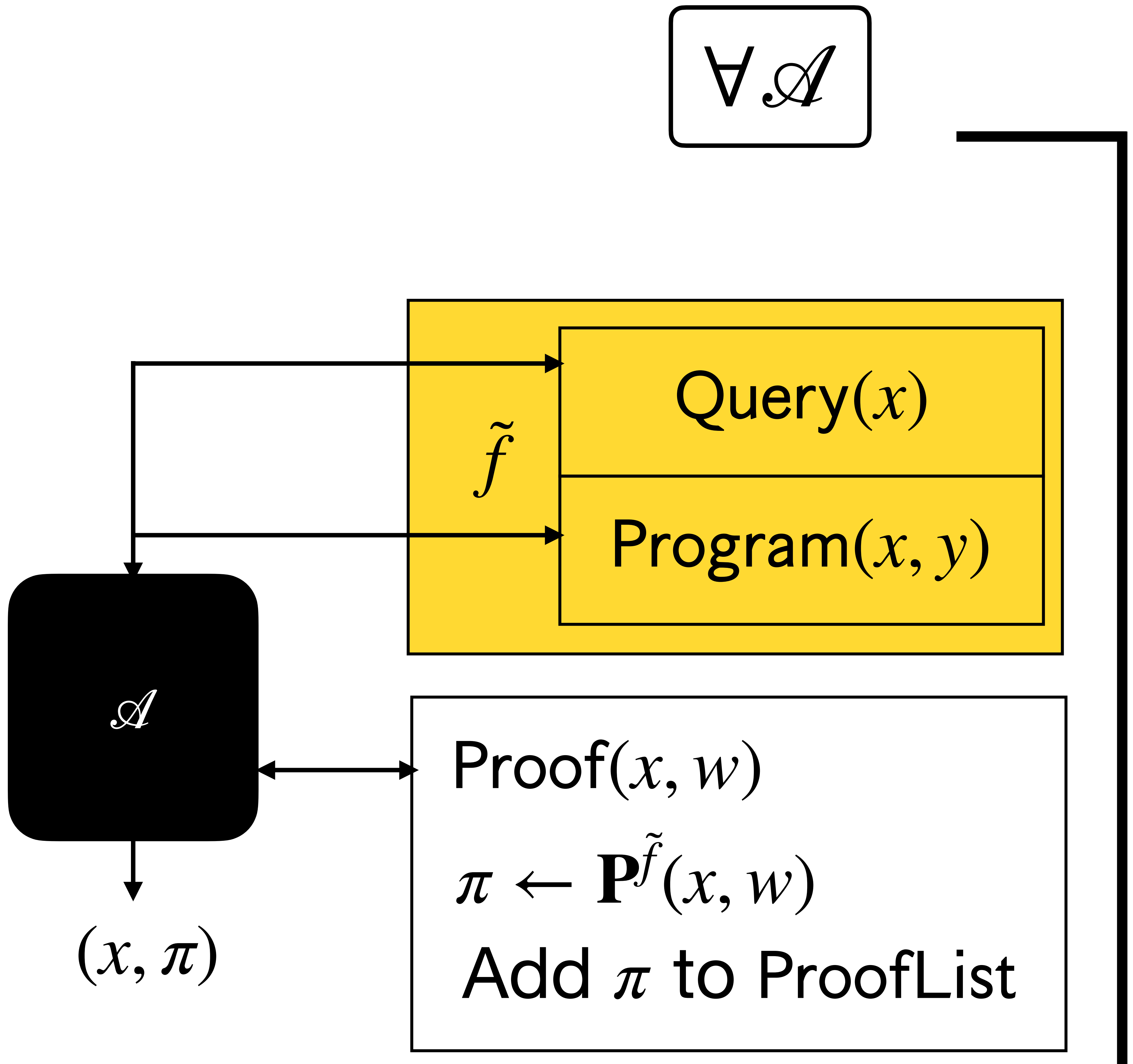


UC-friendly completeness

Micali has UC-friendly completeness because queries to GROM are **unpredictable!**

Adversary should not be able to make honestly generated proofs fail to verify.

$$\Pr\left[\left(\begin{array}{c} (x, \pi) \in \text{ProofList} \\ \text{and} \\ \boxed{\mathbf{V}^{\tilde{f}}(x, \pi) \neq 1} \\ \text{or} \\ \mathbf{V} \text{ queries} \\ \text{programmed points} \end{array} \right) \middle| \quad \right] \leq \varepsilon$$



$\forall \mathscr{A}$

$\tilde{f}$

Query(x)

Program(x, y)

$\mathscr{A}$

(x, π)

Proof(x, w)

$\pi \leftarrow \mathbf{P}^{\tilde{f}}(x, w)$

Add π to ProofList

UC-friendly ZK

Adversary should not be able to distinguish real and simulated proofs, even with access to a programming oracle.

UC-friendly ZK

Adversary should not be able to distinguish real and simulated proofs, even with access to a programming oracle.

$\exists \mathbf{S}, \forall \mathscr{A}$

s.t

UC-friendly ZK

Adversary should not be able to distinguish real and simulated proofs, even with access to a programming oracle.

$\exists \mathbf{S}, \forall \mathscr{A}$

s.t

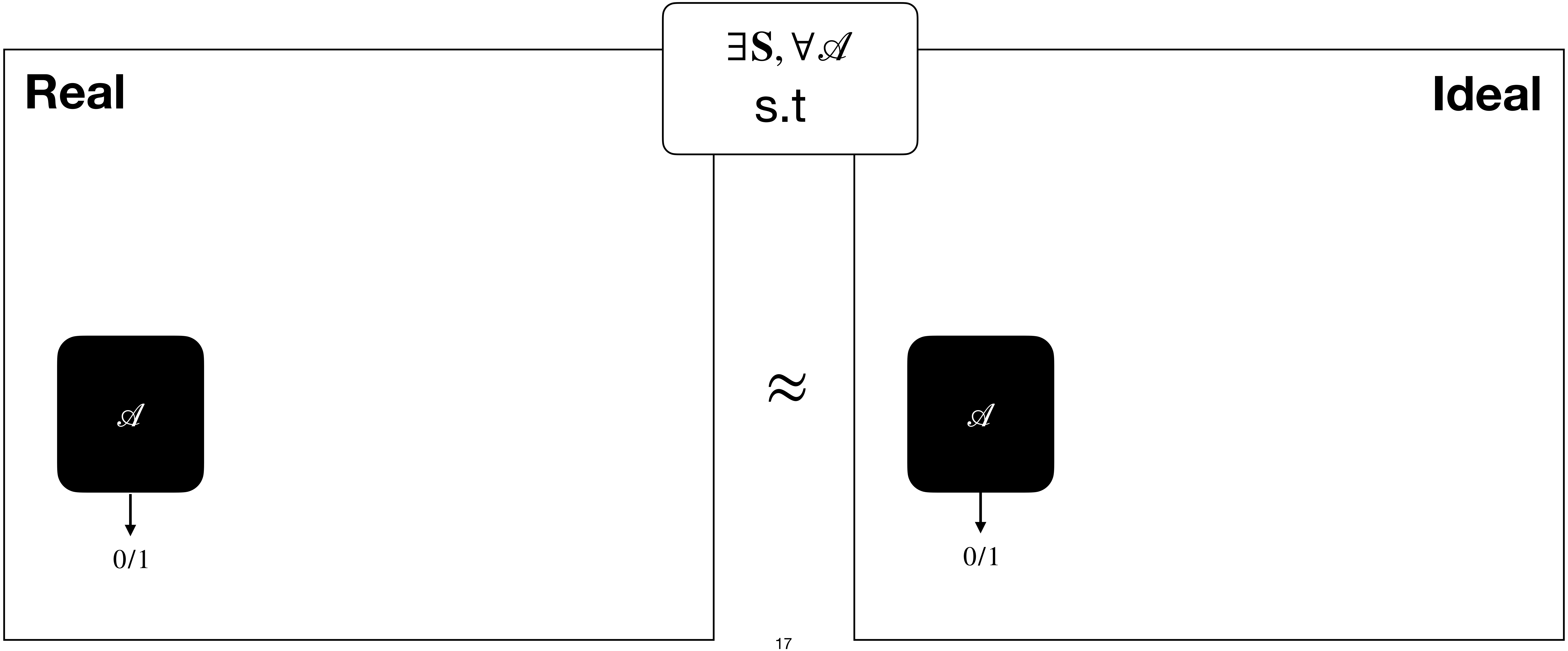
Real

Ideal

$\approx$

UC-friendly ZK

Adversary should not be able to distinguish real and simulated proofs, even with access to a programming oracle.



$\exists \mathbf{S}, \forall \mathscr{A}$

s.t

Real

Ideal

$\approx$

$\mathscr{A}$

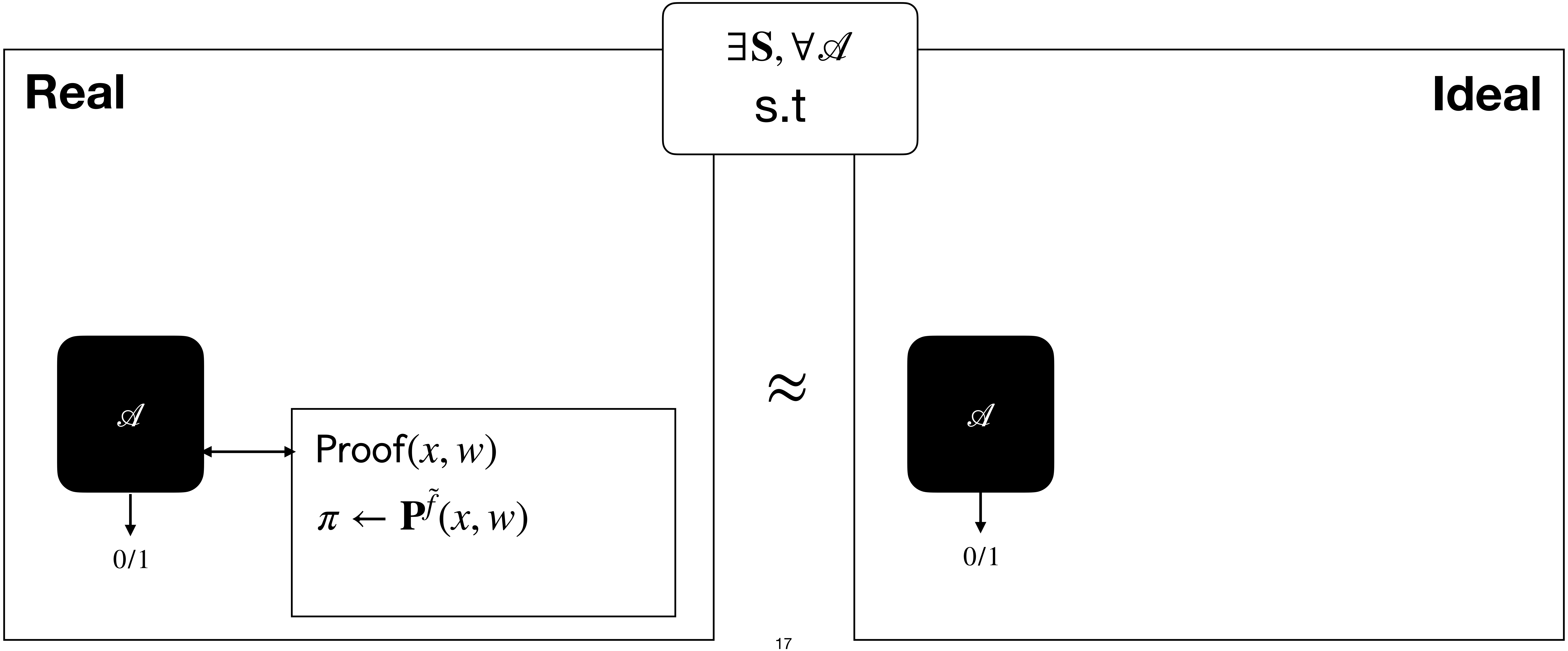
0/1

$\mathscr{A}$

0/1

UC-friendly ZK

Adversary should not be able to distinguish real and simulated proofs, even with access to a programming oracle.



$\exists \mathbf{S}, \forall \mathscr{A}$

s.t

Real

$\mathscr{A}$

0/1

Proof(x, w)

$\pi \leftarrow \mathbf{P}^{\tilde{f}}(x, w)$

$\approx$

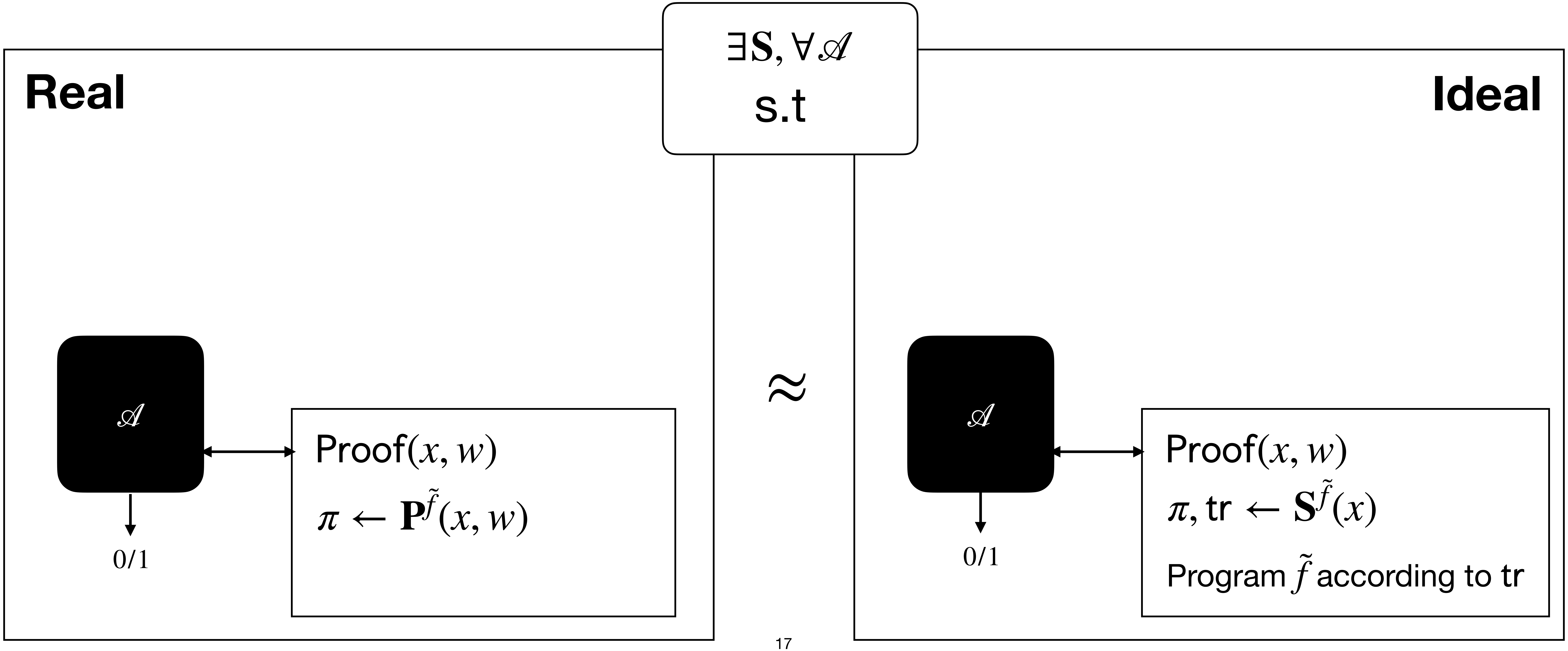
Ideal

$\mathscr{A}$

0/1

UC-friendly ZK

Adversary should not be able to distinguish real and simulated proofs, even with access to a programming oracle.



$\exists \mathbf{S}, \forall \mathscr{A}$

s.t

Real

$\mathscr{A}$

$0/1$

Proof(x, w)

$\pi \leftarrow \mathbf{P}^{\tilde{f}}(x, w)$

$\approx$

Ideal

$\mathscr{A}$

$0/1$

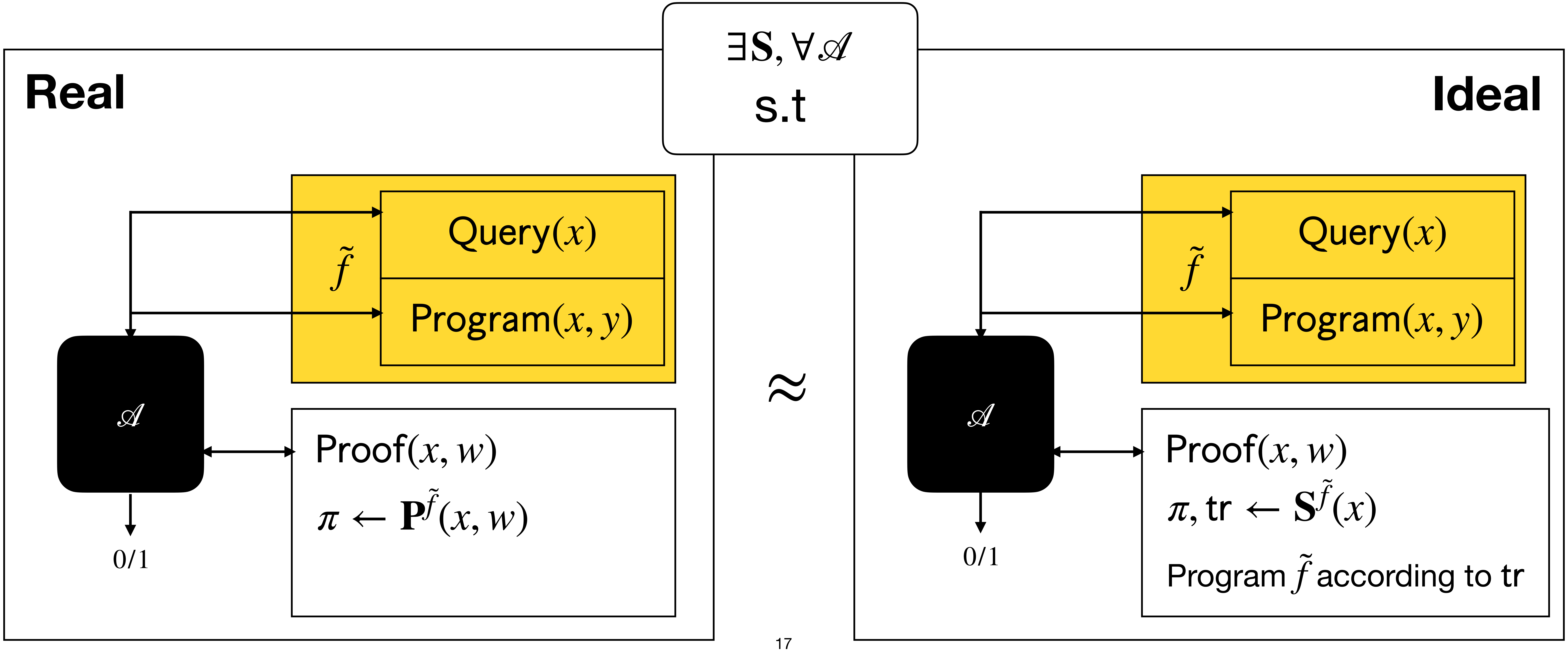
Proof(x, w)

$\pi,$ tr $\leftarrow \mathbf{S}^{\tilde{f}}(x)$

Program $\tilde{f}$ according to tr

UC-friendly ZK

Adversary should not be able to distinguish real and simulated proofs, even with access to a programming oracle.



$\exists \mathbf{S}, \forall \mathscr{A}$

s.t

Real

$\tilde{f}$

Query(x)

Program(x, y)

$\mathscr{A}$

$0/1$

Proof(x, w)

$\pi \leftarrow \mathbf{P}^{\tilde{f}}(x, w)$

$\approx$

Ideal

$\tilde{f}$

Query(x)

Program(x, y)

$\mathscr{A}$

$0/1$

Proof(x, w)

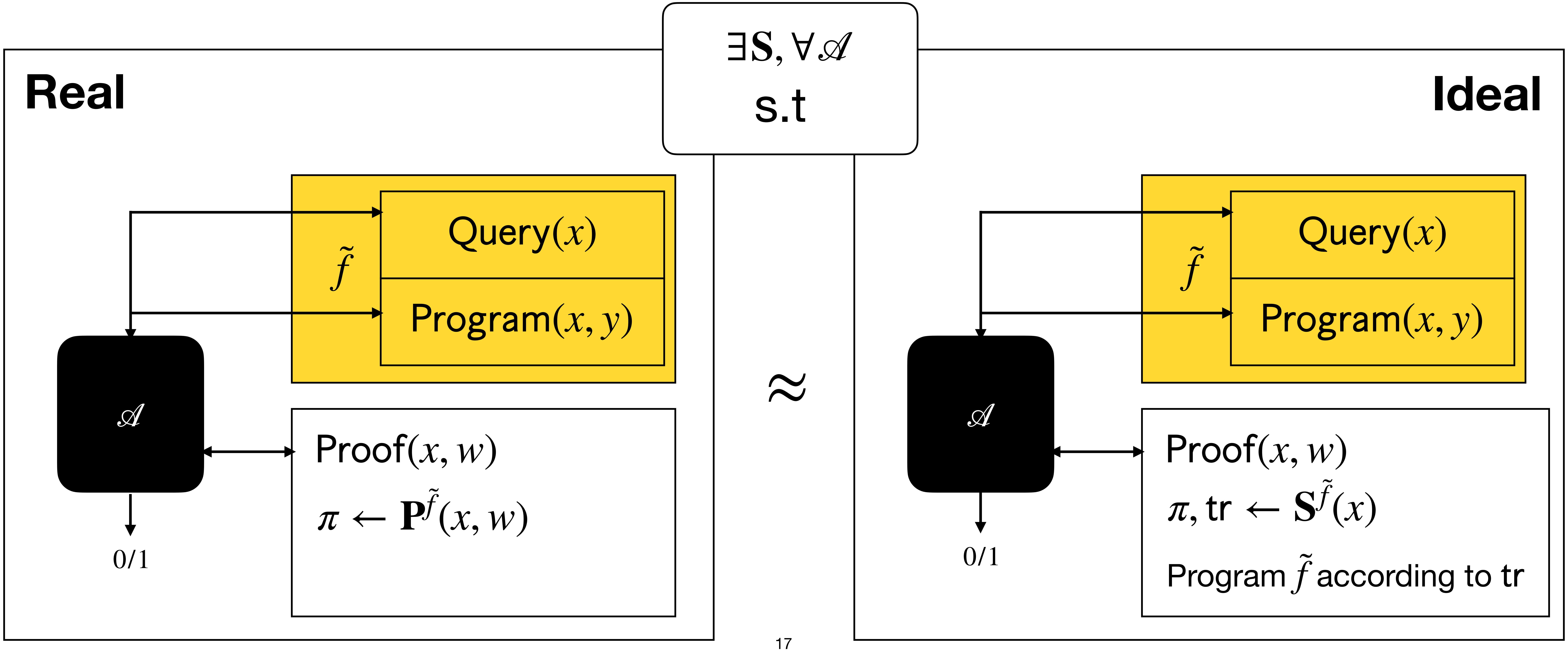
π, tr $\leftarrow \mathbf{S}^{\tilde{f}}(x)$

Program $\tilde{f}$ according to tr

UC-friendly ZK

Adversary should not be able to distinguish real and si even with access to a programming oracle.

Micali has UC-friendly zero knowledge, more involved but follows closely zero-knowledge of Micali in the ROM



$\exists \mathbf{S}, \forall \mathscr{A}$

s.t

Real

$\tilde{f}$

Query(x)

Program(x, y)

$\mathscr{A}$

$0/1$

Proof(x, w)

$\pi \leftarrow \mathbf{P}^{\tilde{f}}(x, w)$

$\approx$

Ideal

$\tilde{f}$

Query(x)

Program(x, y)

$\mathscr{A}$

$0/1$

Proof(x, w)

π, tr $\leftarrow \mathbf{S}^{\tilde{f}}(x)$

Program $\tilde{f}$ according to tr

UC-friendly knowledge soundness

Adversary should not be able to generate fresh proofs
that the extractor cannot extract a witness from

UC-friendly knowledge soundness

Adversary should not be able to generate fresh proofs that the extractor cannot extract a witness from

$\boxed{\exists \mathscr{E} \, \forall \mathscr{A}}$

UC-friendly knowledge soundness

Adversary should not be able to generate fresh proofs that the extractor cannot extract a witness from

$$\boxed{\exists \mathscr{E}\ \forall \mathscr{A}}$$

$$\Pr\left[\quad \middle| \quad\right]$$

UC-friendly knowledge soundness

Adversary should not be able to generate fresh proofs that the extractor cannot extract a witness from

$\boxed{\exists \mathscr{E}\ \forall \mathscr{A}}$

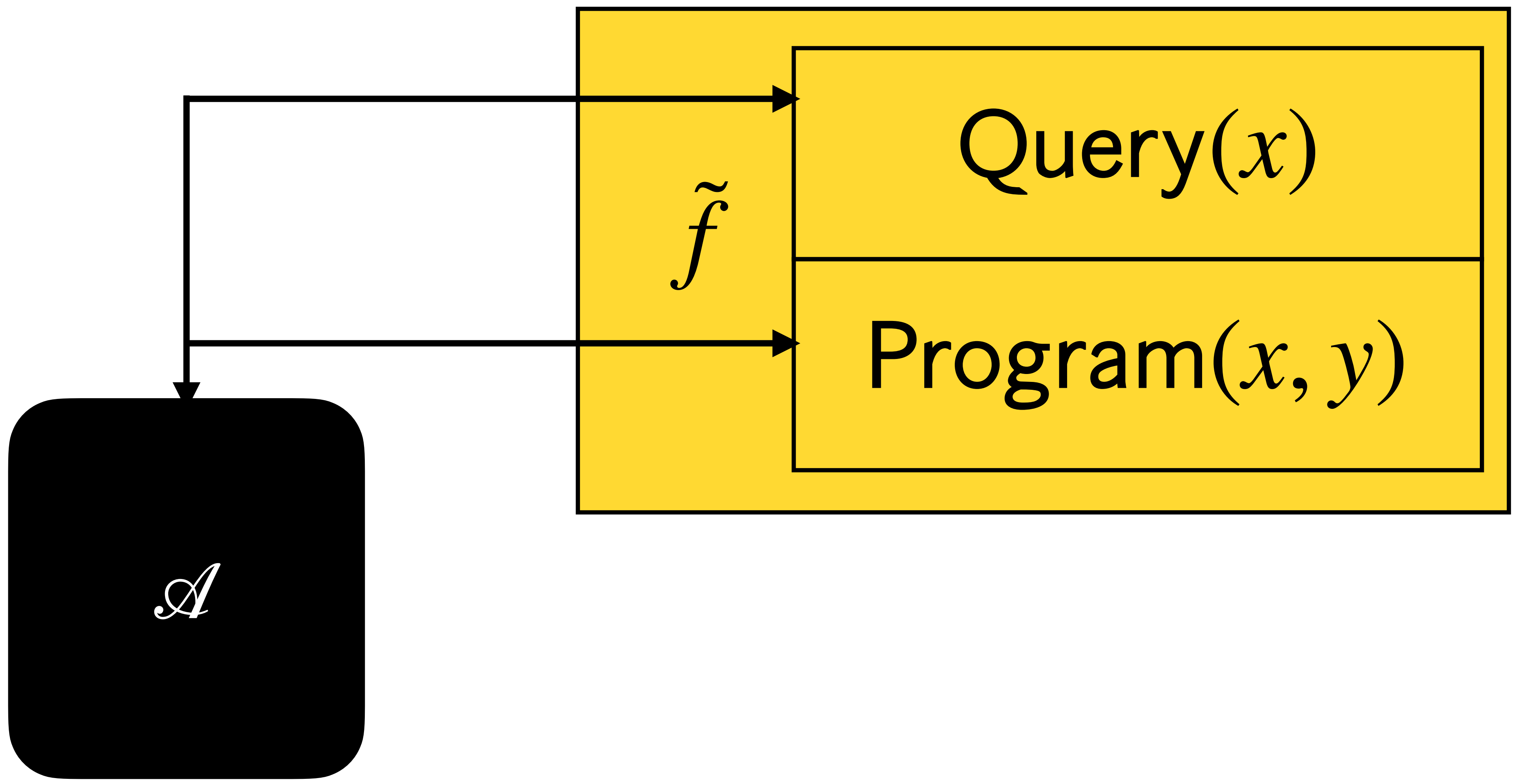
$$\Pr\left[\qquad \middle| \qquad \boxed{\mathscr{A}} \qquad \right]$$

UC-friendly knowledge soundness

Adversary should not be able to generate fresh proofs that the extractor cannot extract a witness from

$\exists \mathscr{E} \forall \mathscr{A}$

$\Pr \Big[\quad \Big| \quad \Big]$

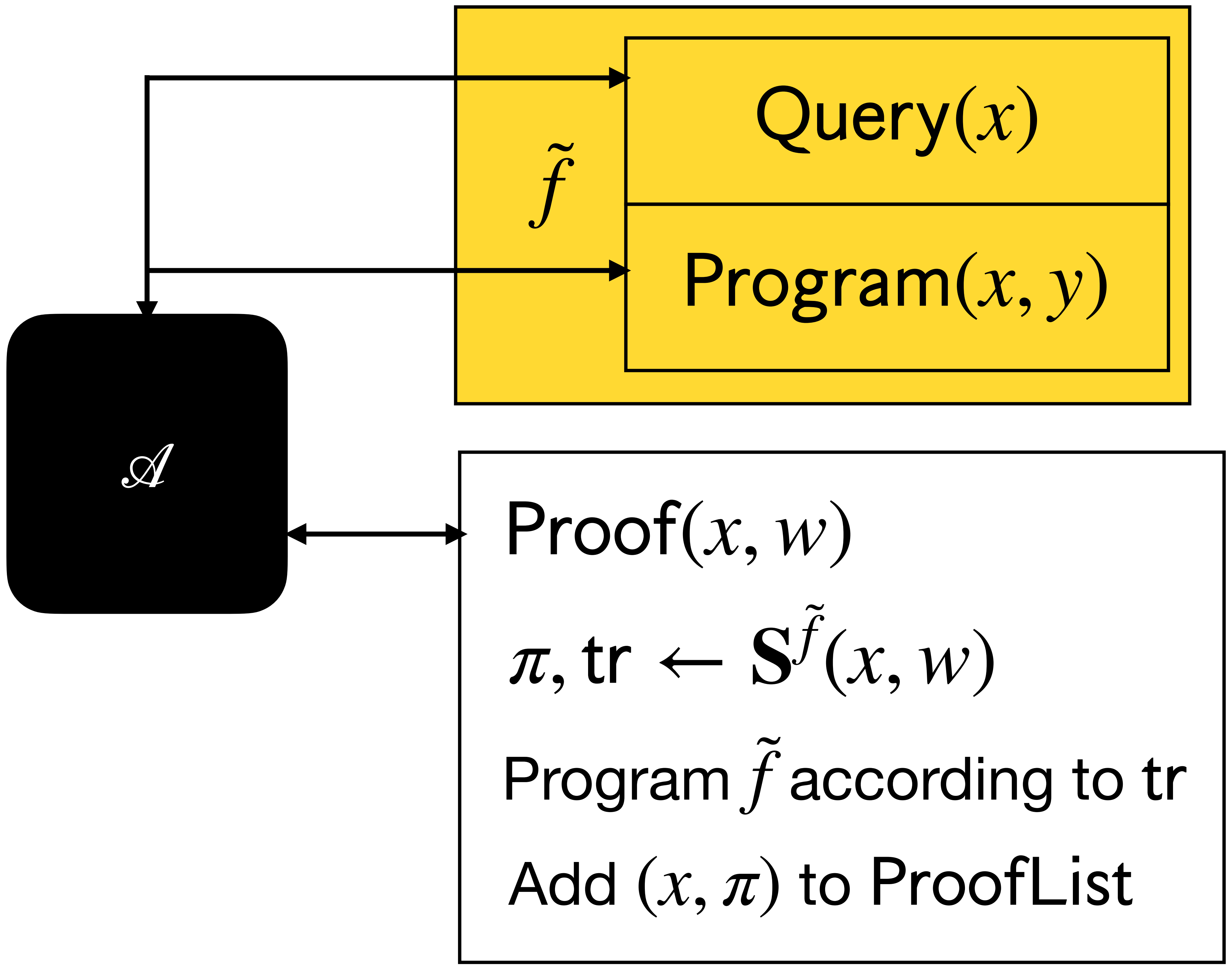


UC-friendly knowledge soundness

Adversary should not be able to generate fresh proofs that the extractor cannot extract a witness from

$\exists \mathscr{E} \forall \mathscr{A}$

$\Pr\Big[$



$\tilde{f}$

Query(x)

Program(x, y)

$\mathscr{A}$

Proof(x, w)

π, tr $\leftarrow \mathbf{S}^{\tilde{f}}(x, w)$

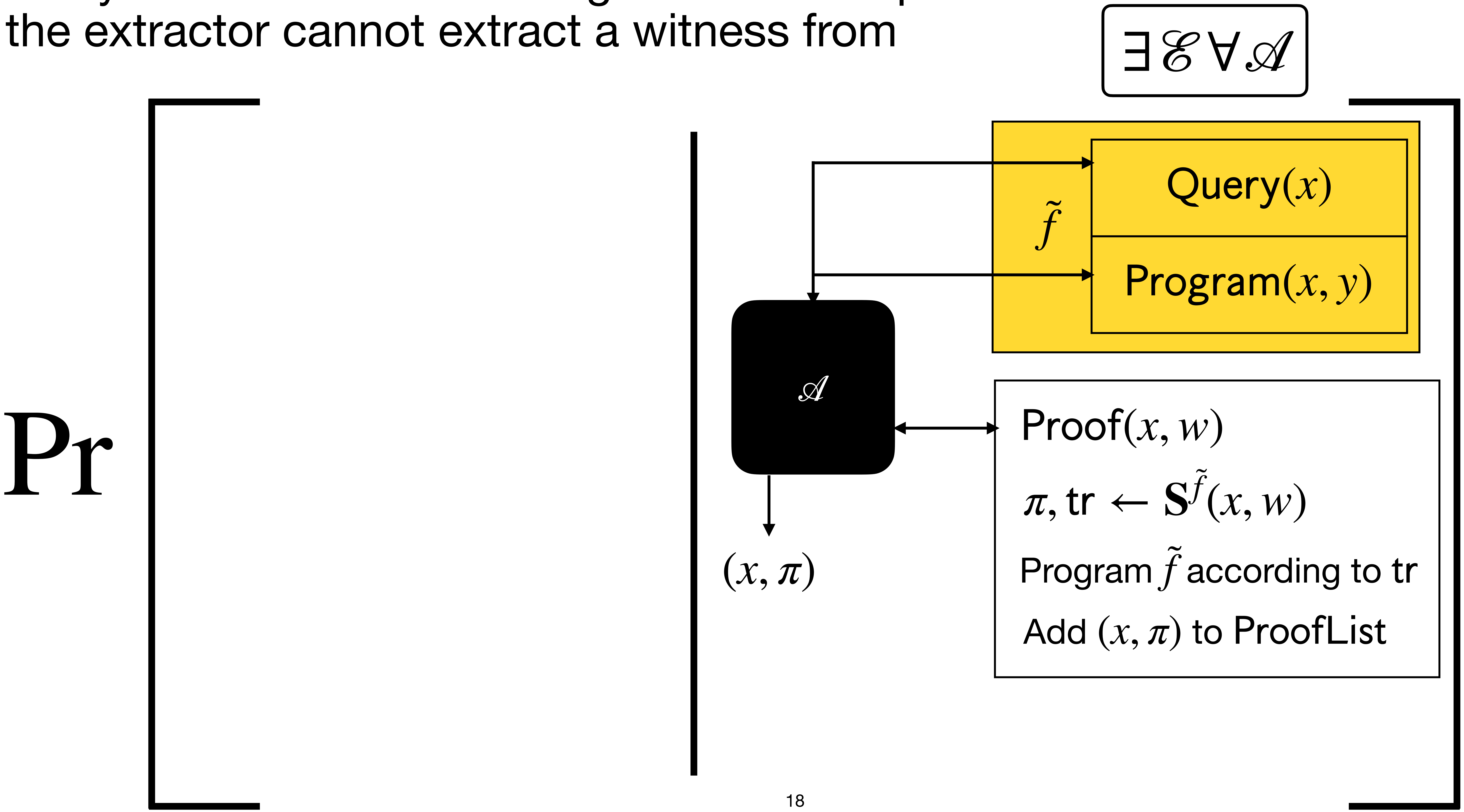
Program $\tilde{f}$ according to tr

Add (x, π) to ProofList

$\Big]$

UC-friendly knowledge soundness

Adversary should not be able to generate fresh proofs that the extractor cannot extract a witness from



$\exists \mathscr{E} \forall \mathscr{A}$

$\Pr$

$\tilde{f}$

Query(x)

Program(x, y)

$\mathscr{A}$

(x, π)

Proof(x, w)

π, tr $\leftarrow \mathbf{S}^{\tilde{f}}(x, w)$

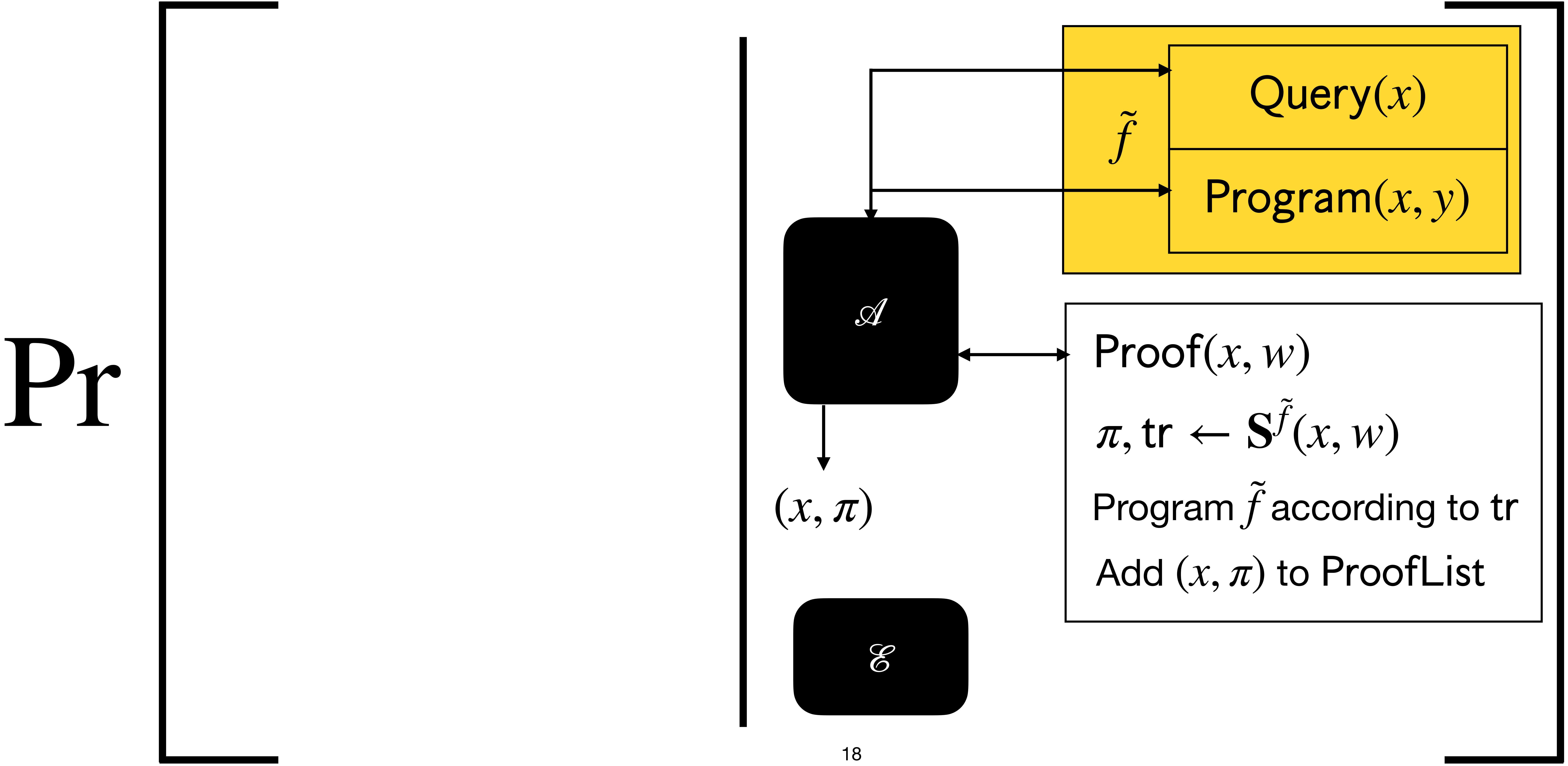
Program $\tilde{f}$ according to tr

Add (x, π) to ProofList

UC-friendly knowledge soundness

Adversary should not be able to generate fresh proofs that the extractor cannot extract a witness from

$\exists \mathscr{E} \forall \mathscr{A}$



Proof(x, w)

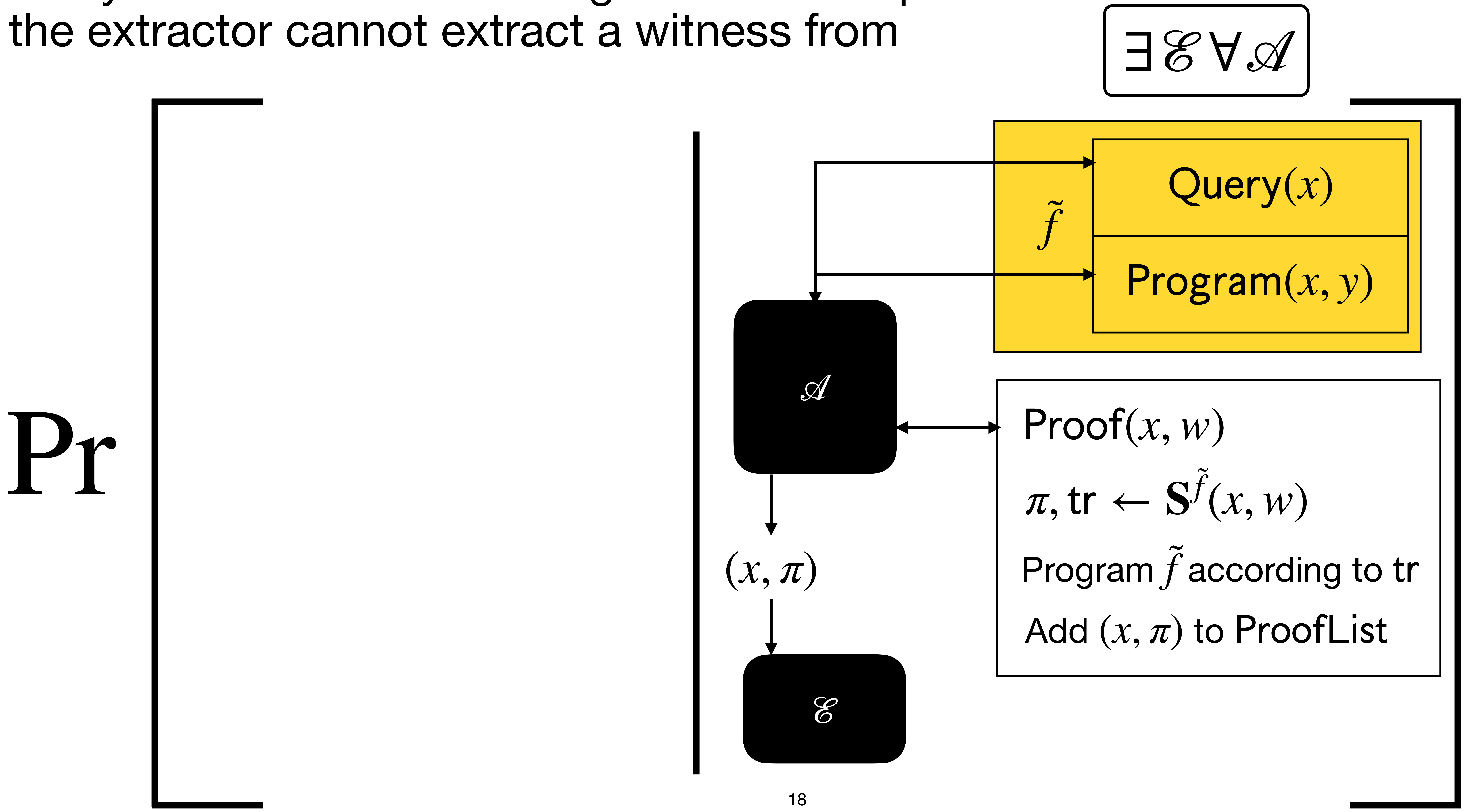
π, tr $\leftarrow \mathbf{S}^{\tilde{f}}(x, w)$

Program $\tilde{f}$ according to tr

Add (x, π) to ProofList

UC-friendly knowledge soundness

Adversary should not be able to generate fresh proofs that the extractor cannot extract a witness from



$\exists \mathscr{E} \forall \mathscr{A}$

$\Pr$

$\tilde{f}$

Query(x)

Program(x, y)

$\mathscr{A}$

(x, π)

$\mathscr{E}$

Proof(x, w)

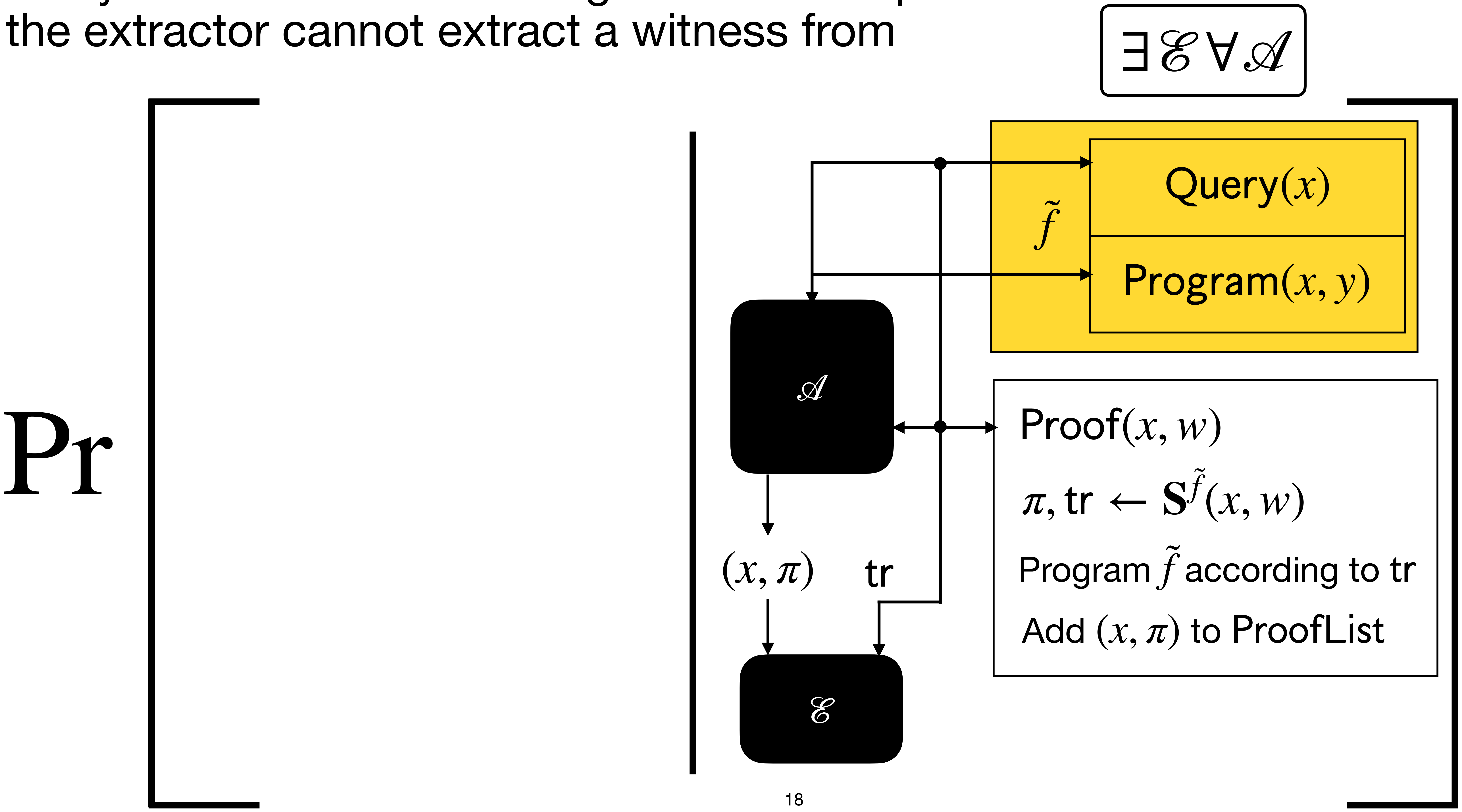
π, tr $\leftarrow \mathbf{S}^{\tilde{f}}(x, w)$

Program $\tilde{f}$ according to tr

Add (x, π) to ProofList

UC-friendly knowledge soundness

Adversary should not be able to generate fresh proofs that the extractor cannot extract a witness from



Proof(x, w)

π, tr $\leftarrow \mathbf{S}^{\tilde{f}}(x, w)$

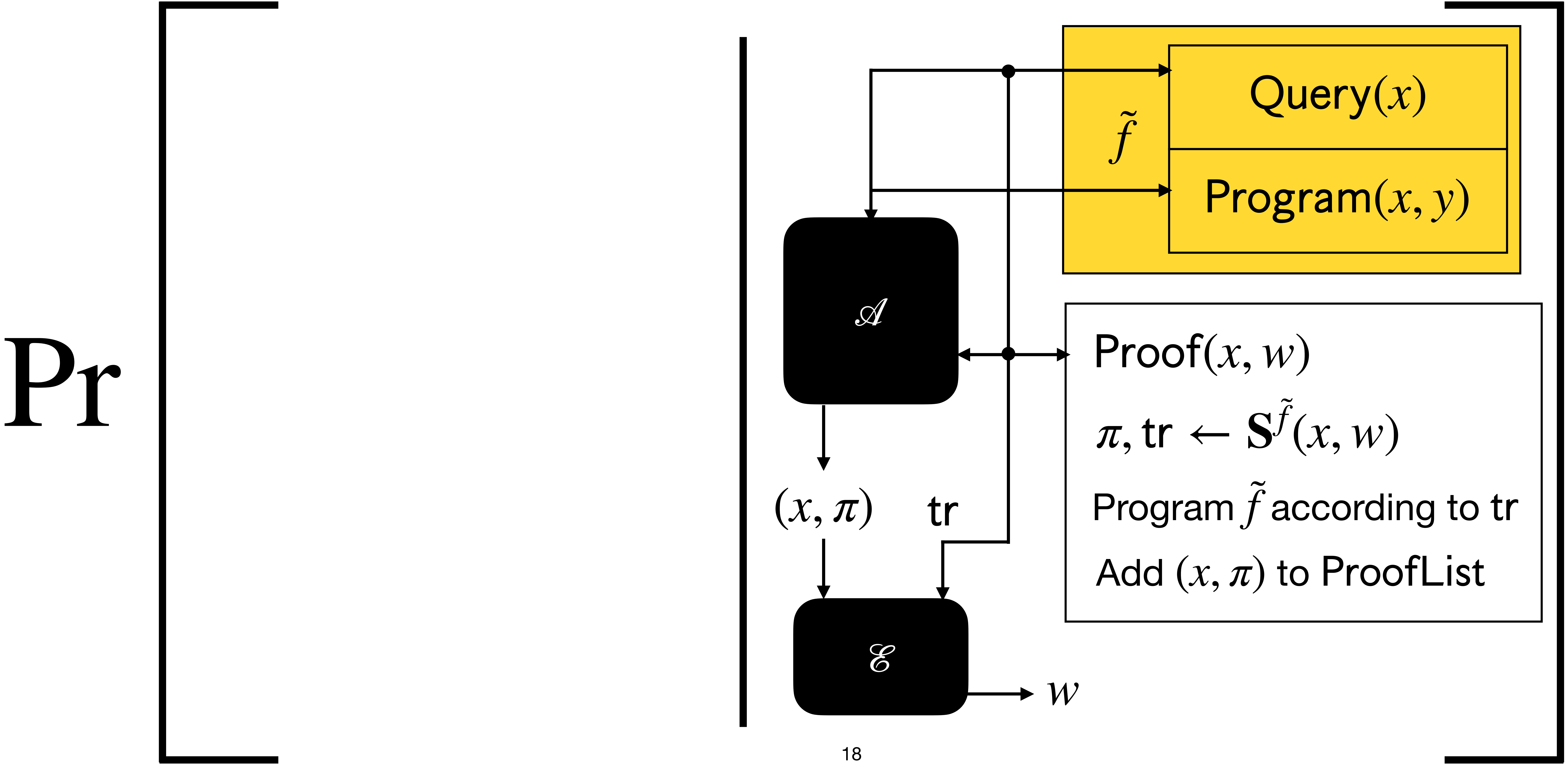
Program $\tilde{f}$ according to tr

Add (x, π) to ProofList

UC-friendly knowledge soundness

Adversary should not be able to generate fresh proofs that the extractor cannot extract a witness from

$\exists \mathscr{E} \forall \mathscr{A}$



Proof(x, w)

π, tr $\leftarrow \mathbf{S}^{\tilde{f}}(x, w)$

Program $\tilde{f}$ according to tr

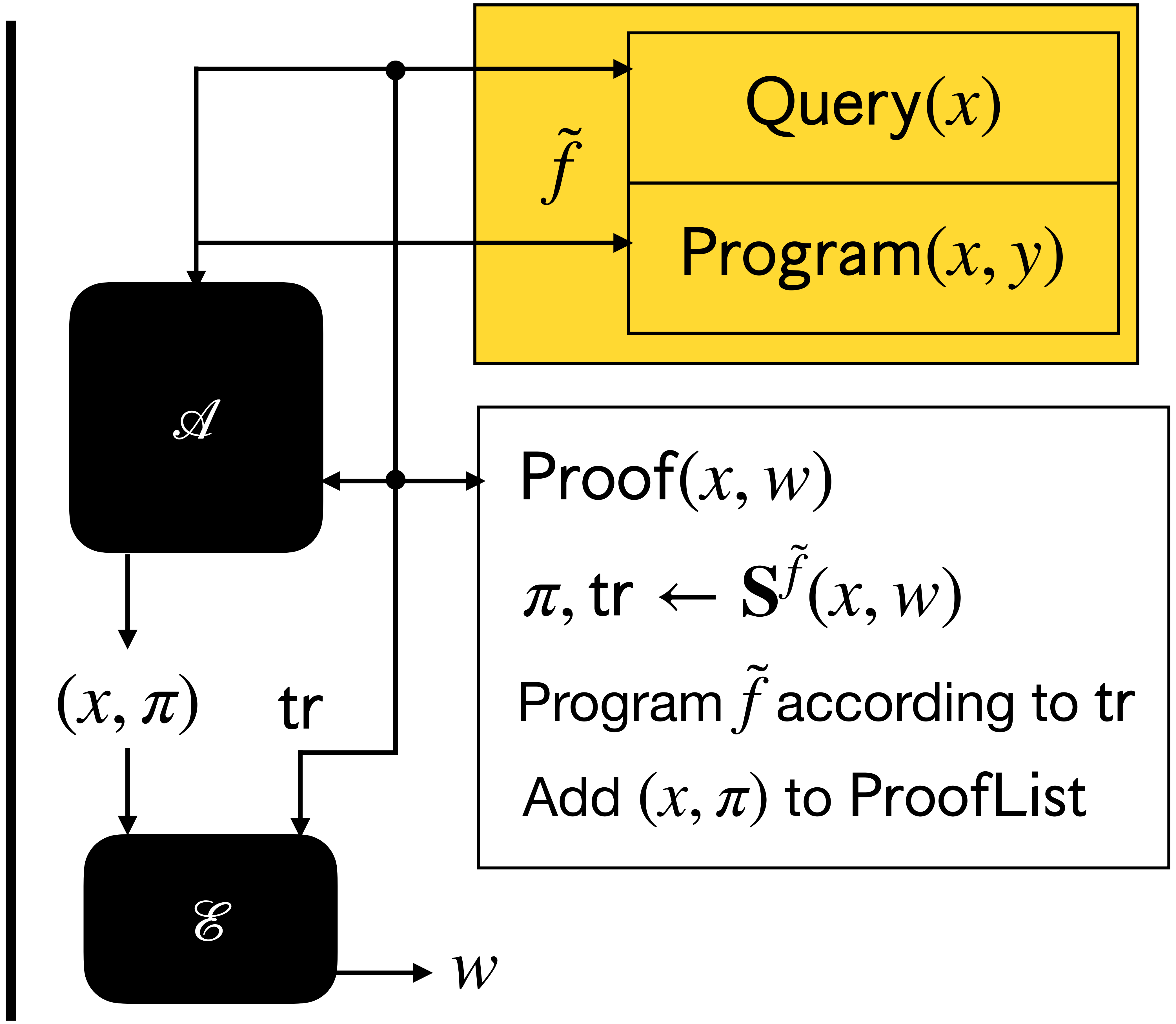
Add (x, π) to ProofList

UC-friendly knowledge soundness

Adversary should not be able to generate fresh proofs that the extractor cannot extract a witness from

$\exists \mathscr{E} \forall \mathscr{A}$

$$\Pr\left[\ \mathbf{V}^{\tilde{f}}(x, \pi) = 1 \ \middle|\ \right]$$



Proof(x, w)

π, tr $\leftarrow \mathbf{S}^{\tilde{f}}(x, w)$

Program $\tilde{f}$ according to tr

Add (x, π) to ProofList

UC-friendly knowledge soundness

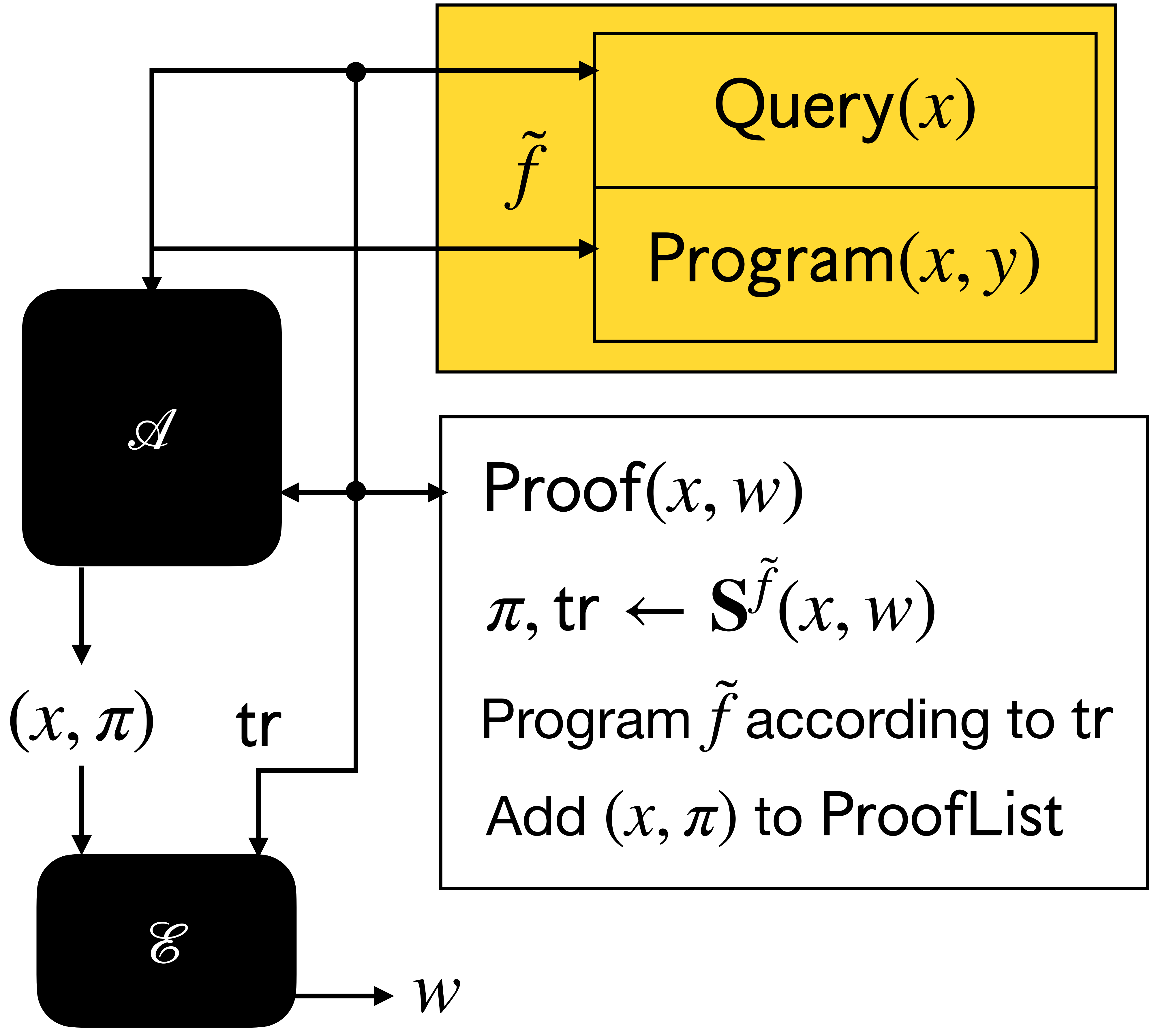
Adversary should not be able to generate fresh proofs that the extractor cannot extract a witness from

$\exists \mathscr{E} \forall \mathscr{A}$

$\Pr$

$\mathbf{V}^{\tilde{f}}(x, \pi) = 1$

V does not query programmed points



Proof(x, w)

π, tr $\leftarrow \mathbf{S}^{\tilde{f}}(x, w)$

Program $\tilde{f}$ according to tr

Add (x, π) to ProofList

UC-friendly knowledge soundness

Adversary should not be able to generate fresh proofs that the extractor cannot extract a witness from

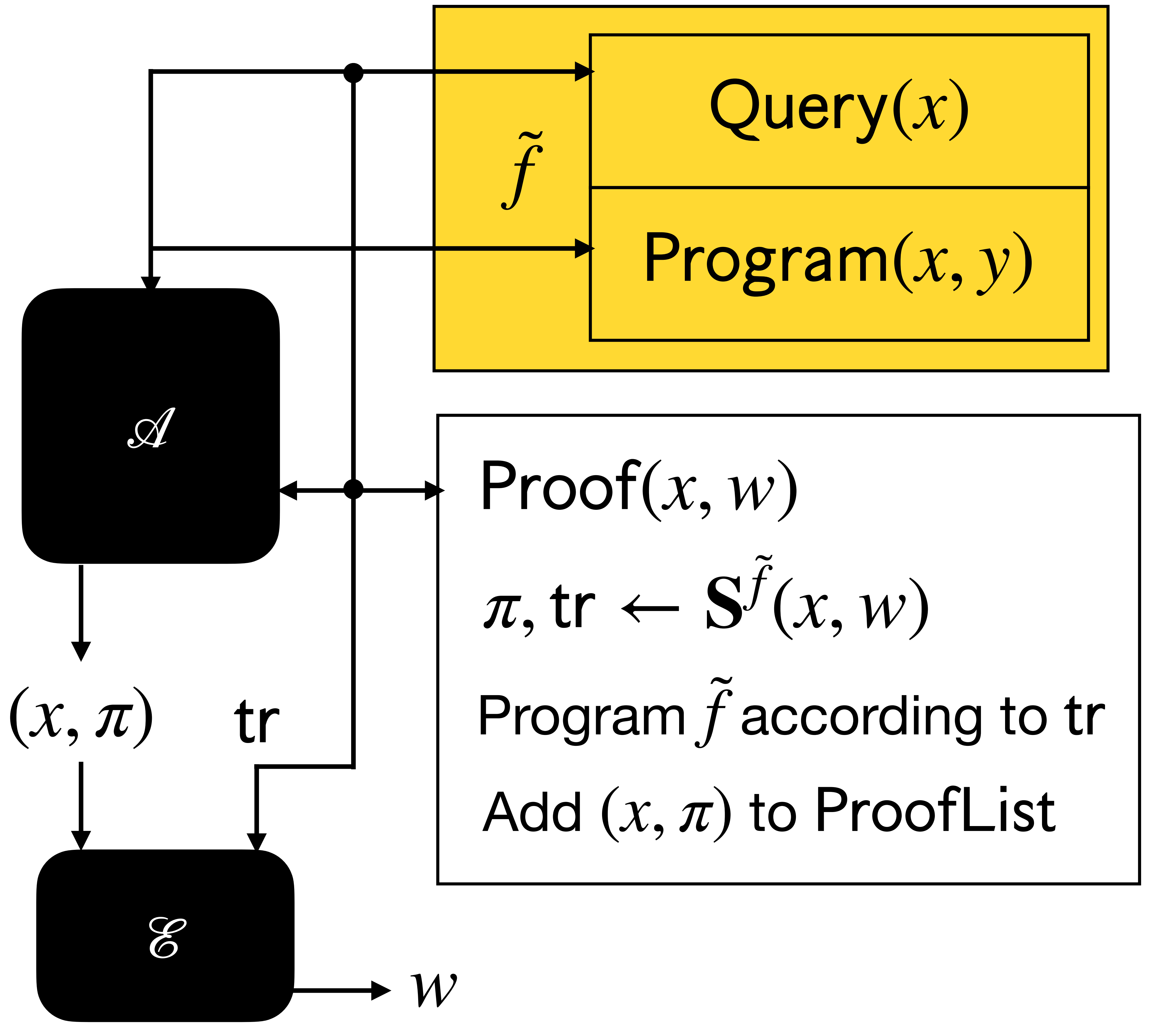
$\exists \mathscr{E} \forall \mathscr{A}$

Pr

$\mathbf{V}^{\tilde{f}}(x, \pi) = 1$

$\mathbf{V}$ does not query programmed points

$(x, \pi) \notin$ ProofList



Proof(x, w)

π, tr $\leftarrow \mathbf{S}^{\tilde{f}}(x, w)$

Program $\tilde{f}$ according to tr

Add (x, π) to ProofList

UC-friendly knowledge soundness

Adversary should not be able to generate fresh proofs that the extractor cannot extract a witness from

$\exists \mathcal{E} \forall \mathcal{A}$

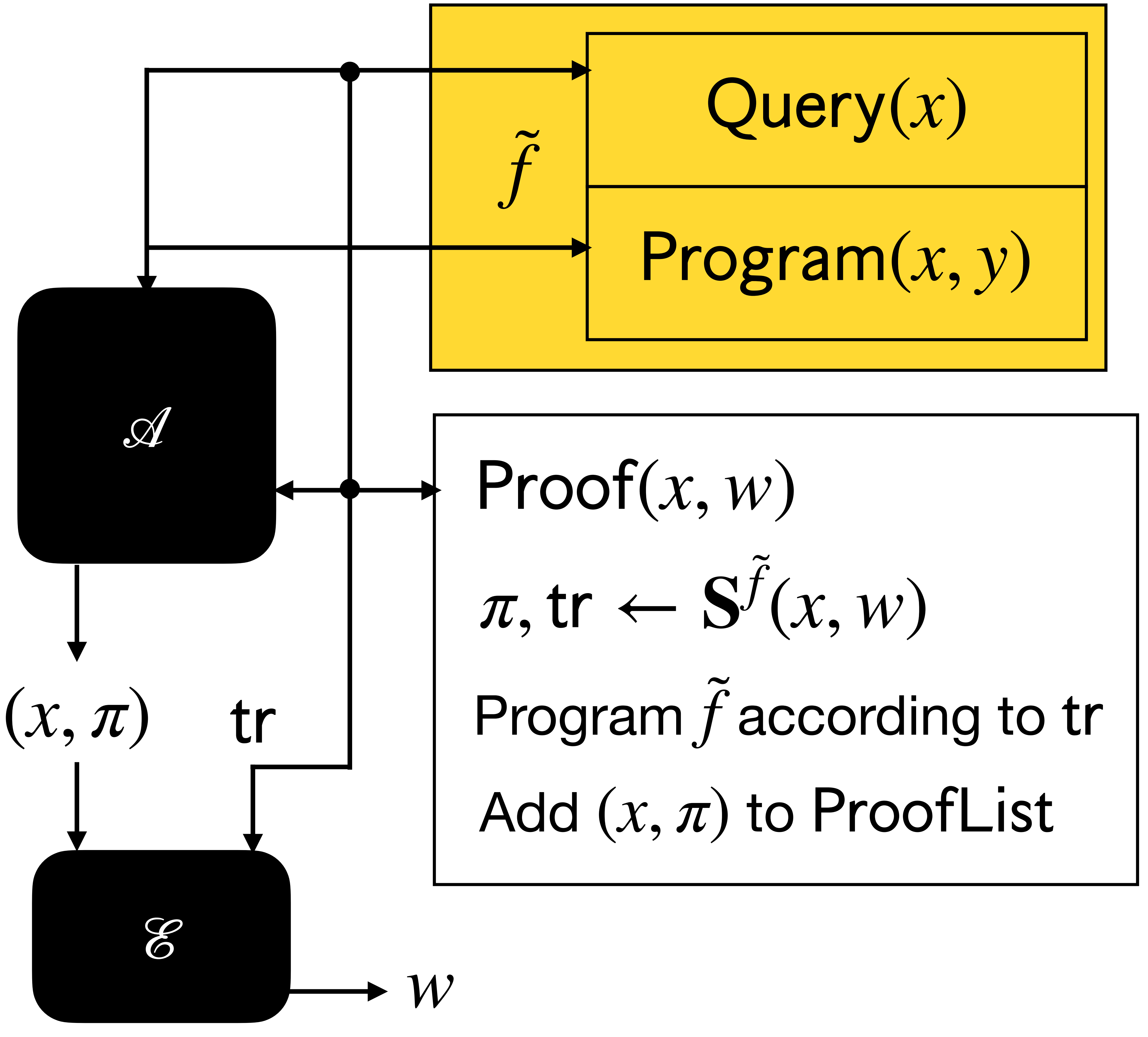
Pr

$\mathbf{V}^{\tilde{f}}(x, \pi) = 1$

V does not query programmed points

$(x, \pi) \notin$ ProofList

$(x, w) \notin R$



Proof(x, w)

π, tr $\leftarrow \mathbf{S}^{\tilde{f}}(x, w)$

Program $\tilde{f}$ according to tr

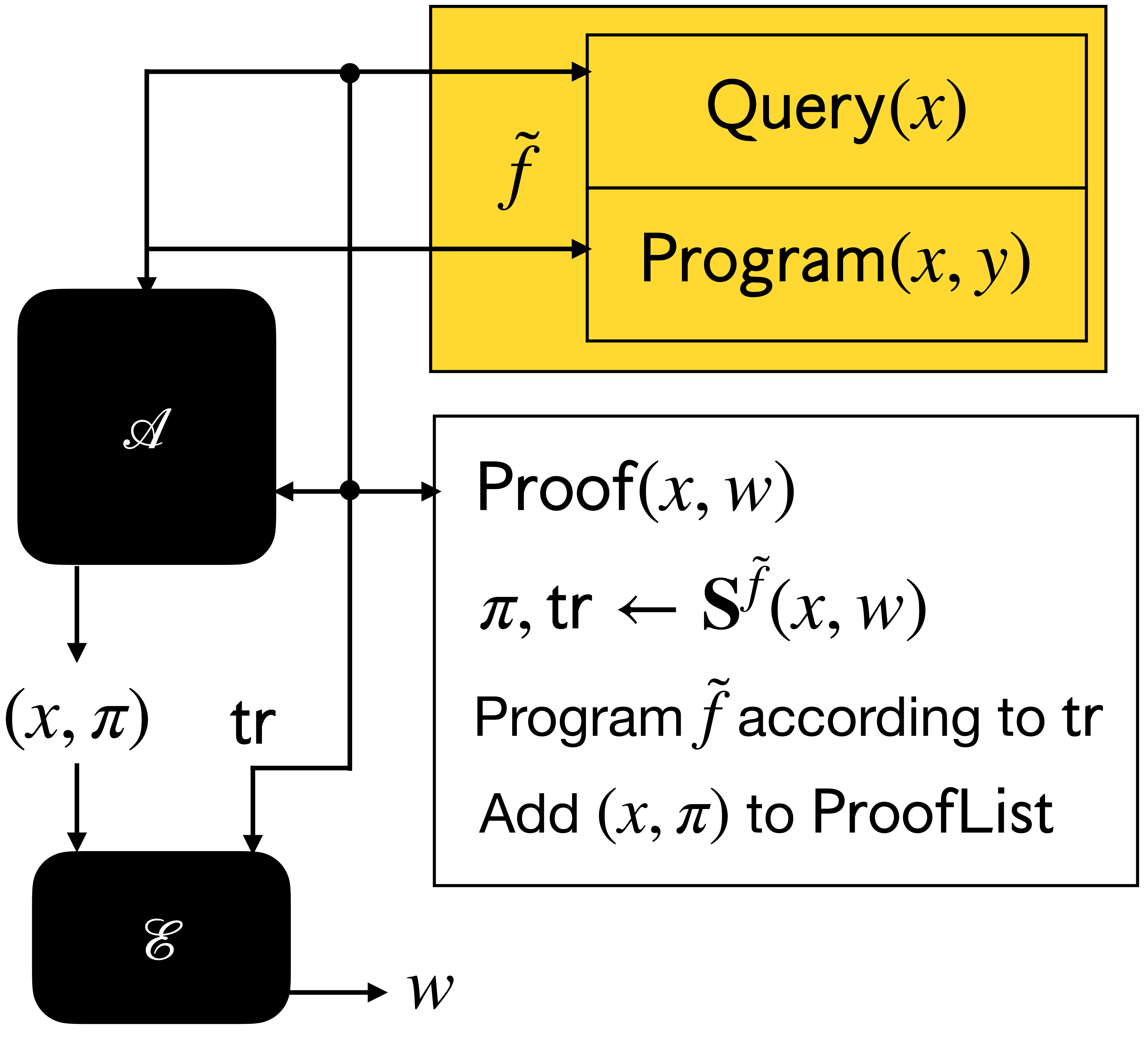
Add (x, π) to ProofList

UC-friendly knowledge soundness

Adversary should not be able to generate fresh proofs that the extractor cannot extract a witness from

$\boxed{\exists \mathscr{E} \forall \mathscr{A}}$

$$\Pr\left[\begin{array}{c|c} \boxed{\mathbf{V}^{\tilde{f}}(x,\pi)=1} & \\ \mathbf{V} \text{ does not query programmed points} & \\ (x,\pi) \notin \text{ProofList} & \\ (x,w) \notin R & \end{array}\right] \leq \kappa$$



Proof(x,w)

π, tr $\leftarrow \mathbf{S}^{\tilde{f}}(x,w)$

Program $\tilde{f}$ according to tr

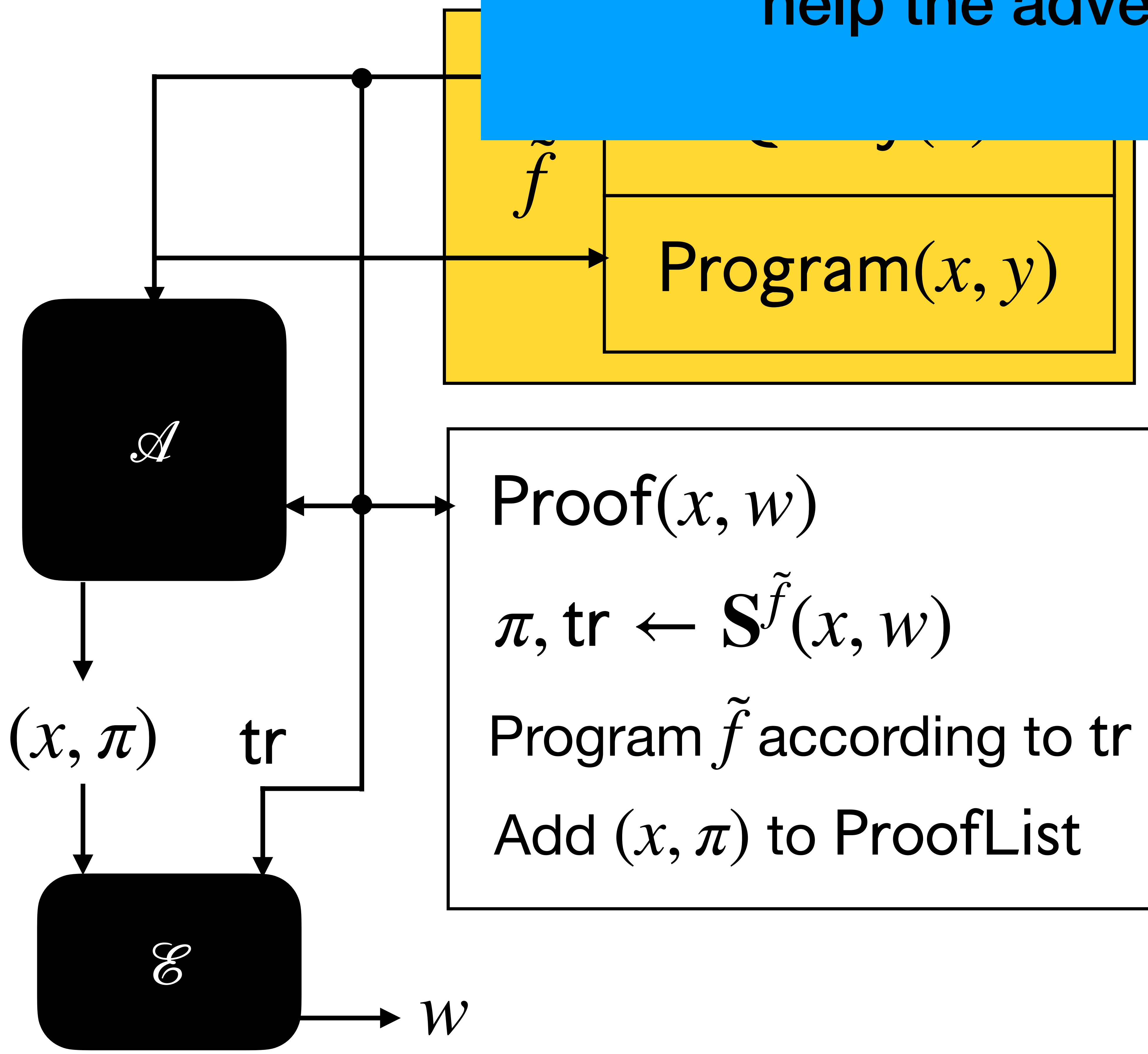
Add (x,π) to ProofList

UC-friendly knowledge sound

Similar to simulation-extractability: in Micali it holds because Merkle trees have strong extraction properties in the ROM, and programming does not help the adversary

Adversary should not be able to generate fresh proofs that the extractor cannot extract a witness from

$$\Pr\left[\begin{array}{c} \mathbf{V}^{\tilde f}(x,\pi)=1 \\ \mathbf{V}\text{ does not query programmed points} \\ (x,\pi)\notin \text{ProofList} \\ (x,w)\notin R \end{array}\right|$$



Proof(x,w)

π, tr $\leftarrow \mathbf{S}^{\tilde f}(x,w)$

Program $\tilde f$ according to tr

Add (x,π) to ProofList

$$\Big] \le \kappa$$

Conclusion

Recap:

What we talked about

Recap:

What we talked about

- UC with budgets

Recap:

What we talked about

- UC with budgets
- UC-friendly security properties imply UC-security

Recap:

What we talked about

- UC with budgets
- UC-friendly security properties imply UC-security
    - UC-friendly completeness

Recap:

What we talked about

- UC with budgets
- UC-friendly security properties imply UC-security
    - UC-friendly completeness
    - UC-friendly zero knowledge

Recap:

What we talked about

- UC with budgets
- UC-friendly security properties imply UC-security
    - UC-friendly completeness
    - UC-friendly zero knowledge
    - UC-friendly knowledge soundness

There is more!

What we did not talk about

There is more!

What we did not talk about

- Concrete security bounds

There is more!

What we did not talk about

- Concrete security bounds
- UC-security of Micali & BCS (leads to UC-security of **deployed** zkSNARKs)

There is more!

What we did not talk about

- Concrete security bounds
- UC-security of Micali & BCS (leads to UC-security of **deployed** zkSNARKs)
- UC-friendly properties are **necessary**

There is more!

What we did not talk about

- Concrete security bounds
- UC-security of Micali & BCS (leads to UC-security of **deployed** zkSNARKs)
- UC-friendly properties are **necessary**
- We can handle adaptive corruptions with **strong** UC-friendly properties

There is more!

What we did not talk about

- Concrete security bounds
- UC-security of Micali & BCS (leads to UC-security of **deployed** zkSNARKs)
- UC-friendly properties are **necessary**
- We can handle adaptive corruptions with **strong** UC-friendly properties
- Merkle trees have (strong) UC-friendly hiding

There is more!

What we did not talk about

- Concrete security bounds
- UC-security of Micali & BCS (leads to UC-security of **deployed** zkSNARKs)
- UC-friendly properties are **necessary**
- We can handle adaptive corruptions with **strong** UC-friendly properties
- Merkle trees have (strong) UC-friendly hiding
- Merkle trees have (strong) UC-friendly extraction

There is more!

What we did not talk about

- Concrete security bounds
- UC-security of Micali & BCS (leads to UC-security of **deployed** zkSNARKs)
- UC-friendly properties are **necessary**
- We can handle adaptive corruptions with **strong** UC-friendly properties
- Merkle trees have (strong) UC-friendly hiding
- Merkle trees have (strong) UC-friendly extraction
- **Open question:** extend the result to IOPs without straightline KS

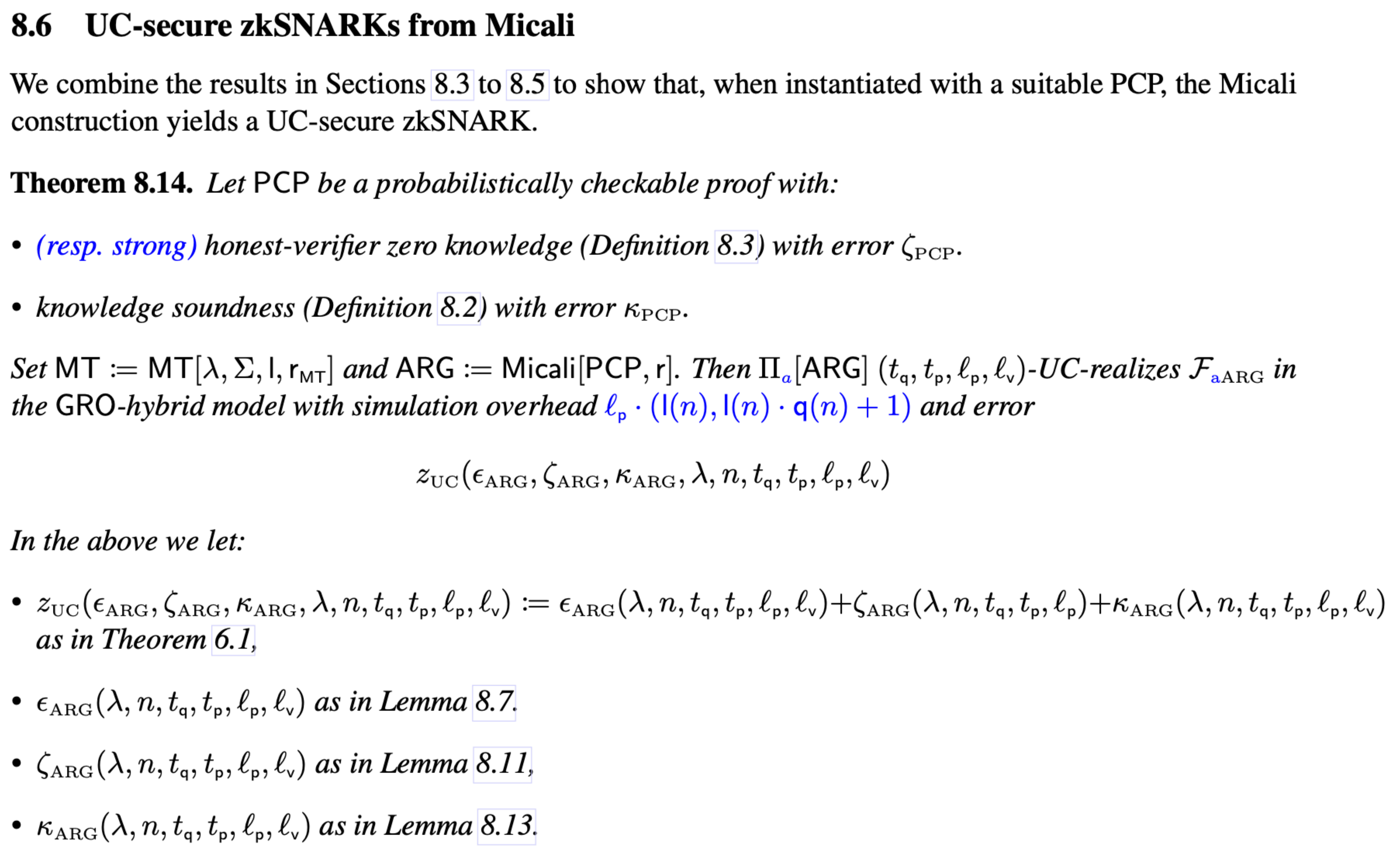
Conclusion

Micali

&

BCS

These zkSNARKs are UC-secure in the GROM



8.6 UC-secure zkSNARKs from Micali

We combine the results in Sections 8.3 to 8.5 to show that, when instantiated with a suitable PCP, the Micali construction yields a UC-secure zkSNARK.

Theorem 8.14. *Let* PCP *be a probabilistically checkable proof with:*

- *(resp. strong) honest-verifier zero knowledge (Definition 8.3) with error* ζ_{PCP}.
- *knowledge soundness (Definition 8.2) with error* κ_{PCP}.

Set $\mathsf{MT} := \mathsf{MT}[\lambda, \Sigma, \mathsf{l}, \mathsf{r}_{\mathsf{MT}}]$ *and* $\mathsf{ARG} := \mathsf{Micali}[\mathsf{PCP}, \mathsf{r}]$. *Then* $\Pi_{a}[\mathsf{ARG}]$ $(t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}})$*-UC-realizes* $\mathcal{F}_{\mathrm{aARG}}$ *in the* GRO*-hybrid model with simulation overhead* $\ell_{\mathsf{p}} \cdot (\mathsf{l}(n), \mathsf{l}(n) \cdot \mathsf{q}(n) + 1)$ *and error*

$$z_{\mathrm{UC}}(\epsilon_{\mathrm{ARG}}, \zeta_{\mathrm{ARG}}, \kappa_{\mathrm{ARG}}, \lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}})$$

In the above we let:

- $z_{\mathrm{UC}}(\epsilon_{\mathrm{ARG}}, \zeta_{\mathrm{ARG}}, \kappa_{\mathrm{ARG}}, \lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}}) := \epsilon_{\mathrm{ARG}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}}) + \zeta_{\mathrm{ARG}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}) + \kappa_{\mathrm{ARG}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}})$ *as in Theorem 6.1,*
- $\epsilon_{\mathrm{ARG}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}})$ *as in Lemma 8.7.*
- $\zeta_{\mathrm{ARG}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}})$ *as in Lemma 8.11,*
- $\kappa_{\mathrm{ARG}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}})$ *as in Lemma 8.13.*

Concrete security bounds!

Thank you!

Extra slides

Modelling shared functionalities [BCHTZ22]

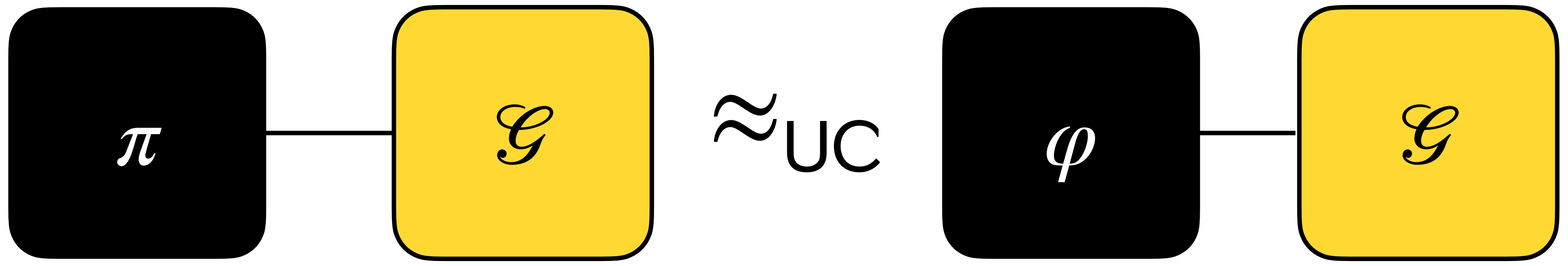
Modelling shared functionalities [BCHTZ22]

Plain UC security **not enough** for **shared** setups

Modelling shared functionalities [BCHTZ22]

Plain UC security **not enough** for **shared** setups

Plain UC:

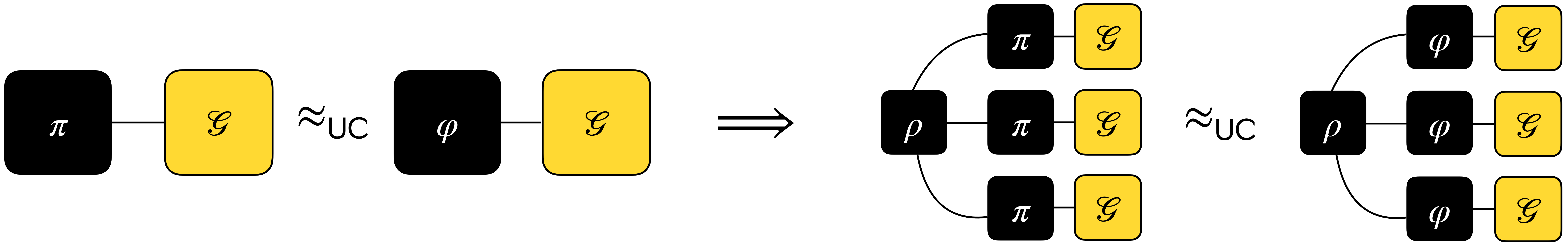


Modelling shared functionalities

[BCHTZ22]

Plain UC security **not enough** for **shared** setups

Plain UC:

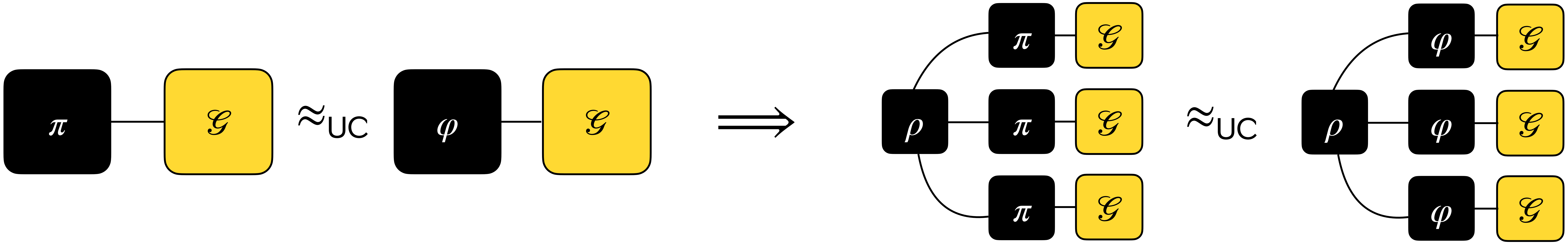


Modelling shared functionalities

[BCHTZ22]

Plain UC security **not enough** for **shared** setups

Plain UC:



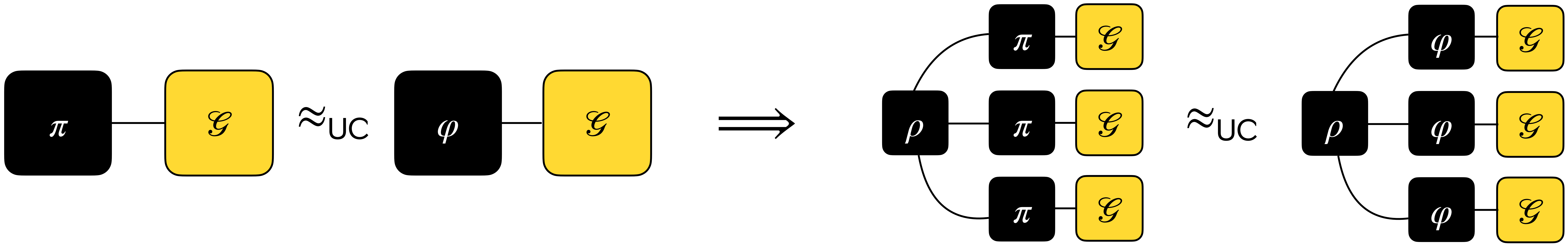
Solution: UC with Global Subroutines!

Modelling shared functionalities

[BCHTZ22]

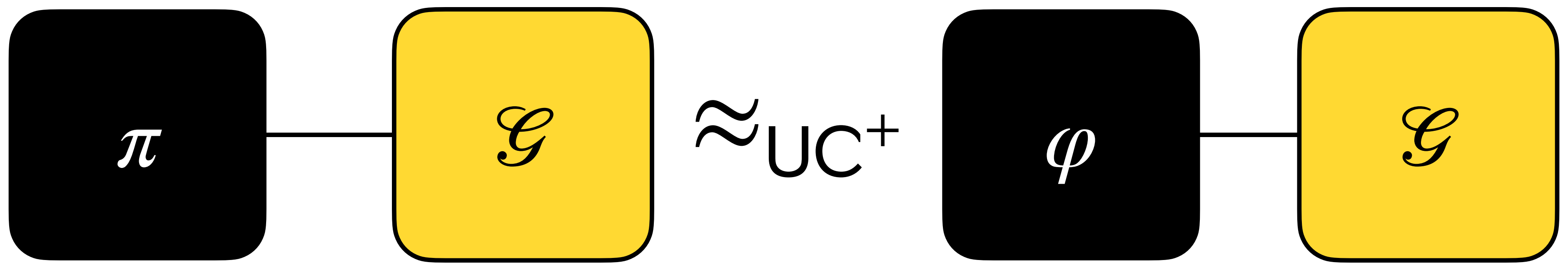
Plain UC security **not enough** for **shared** setups

Plain UC:



Solution: UC with Global Subroutines!

UCGS:

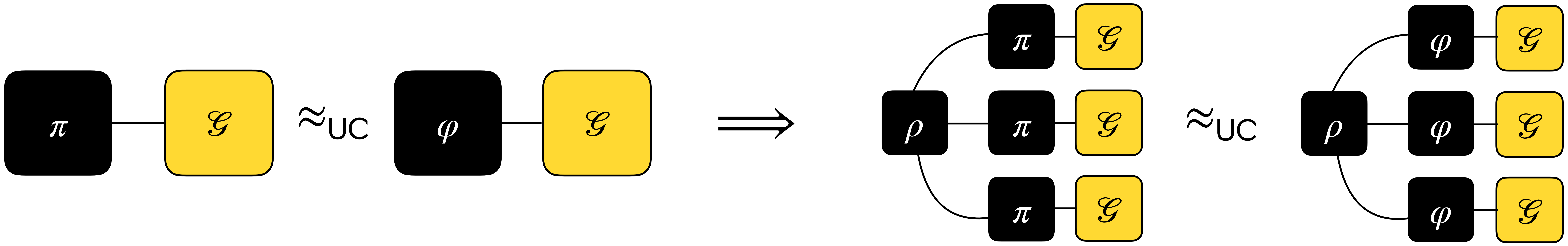


Modelling shared functionalities

[BCHTZ22]

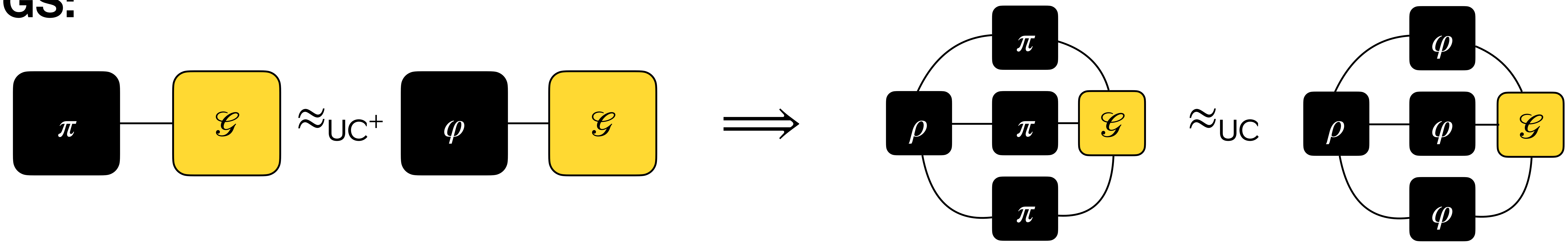
Plain UC security **not enough** for **shared** setups

Plain UC:



Solution: UC with Global Subroutines!

UCGS:



Micali has UC-friendly ZK

Micali has UC-friendly ZK

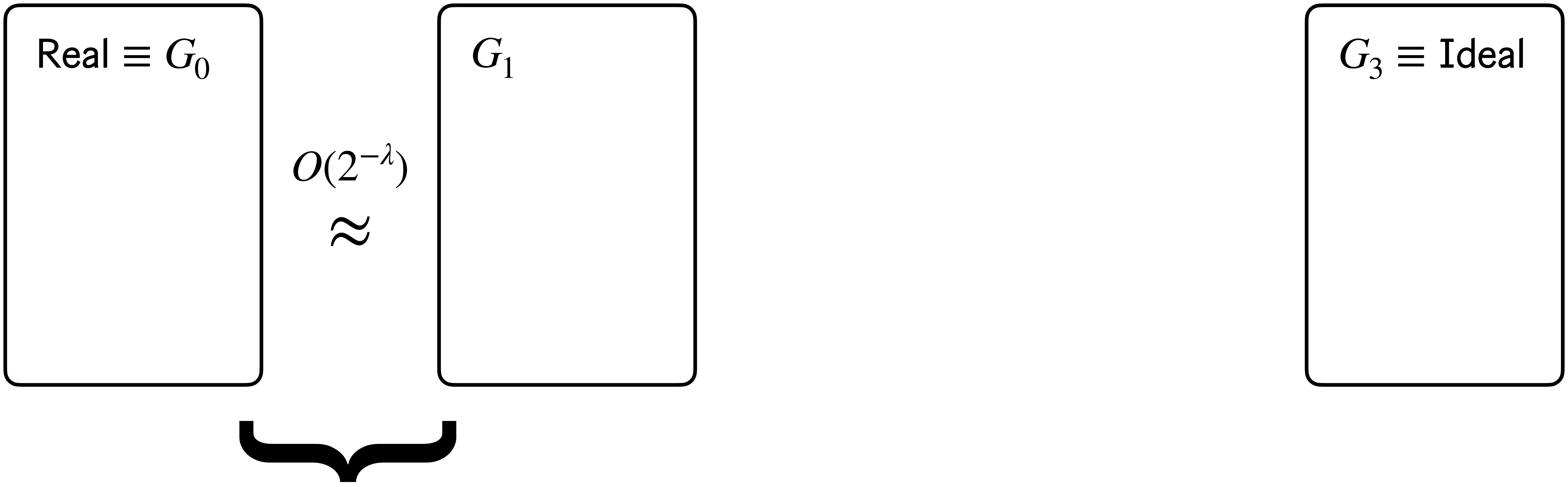
Real $\equiv G_0$

Micali has UC-friendly ZK

Real $\equiv G_0$

$G_3 \equiv$ Ideal

Micali has UC-friendly ZK



Real $\equiv G_0$

$O(2^{-\lambda})$

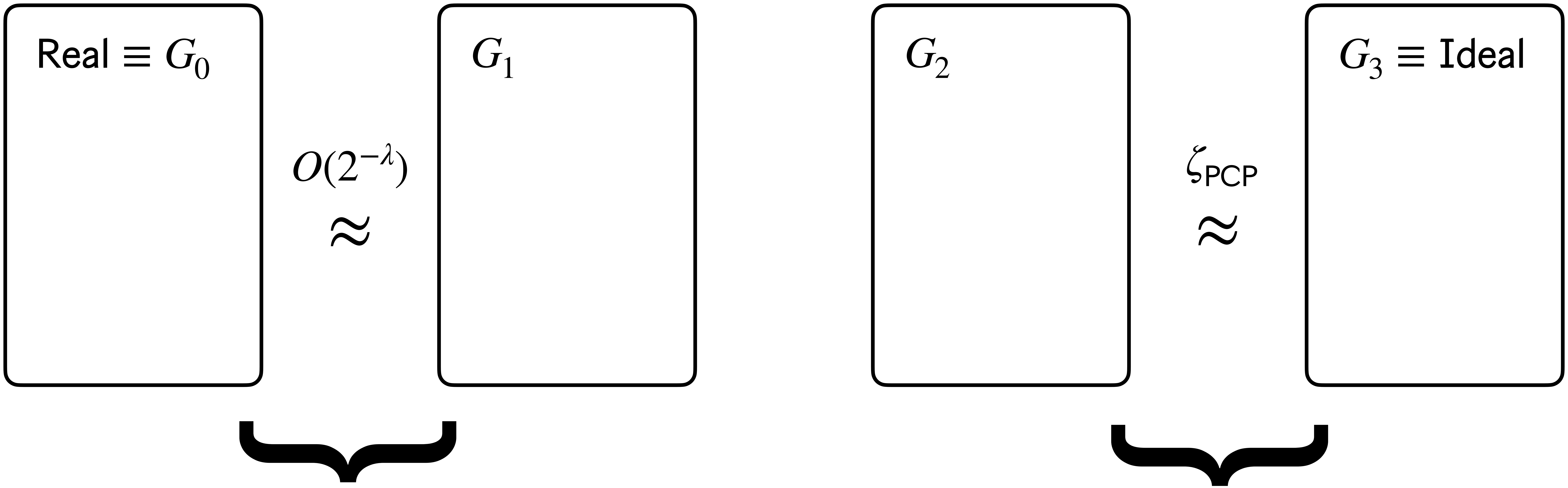
$\approx$

G_1

$G_3 \equiv$ Ideal

FS input hard
to predict

Micali has UC-friendly ZK



$\mathsf{Real} \equiv G_0$

G_1

G_2

$G_3 \equiv \mathsf{Ideal}$

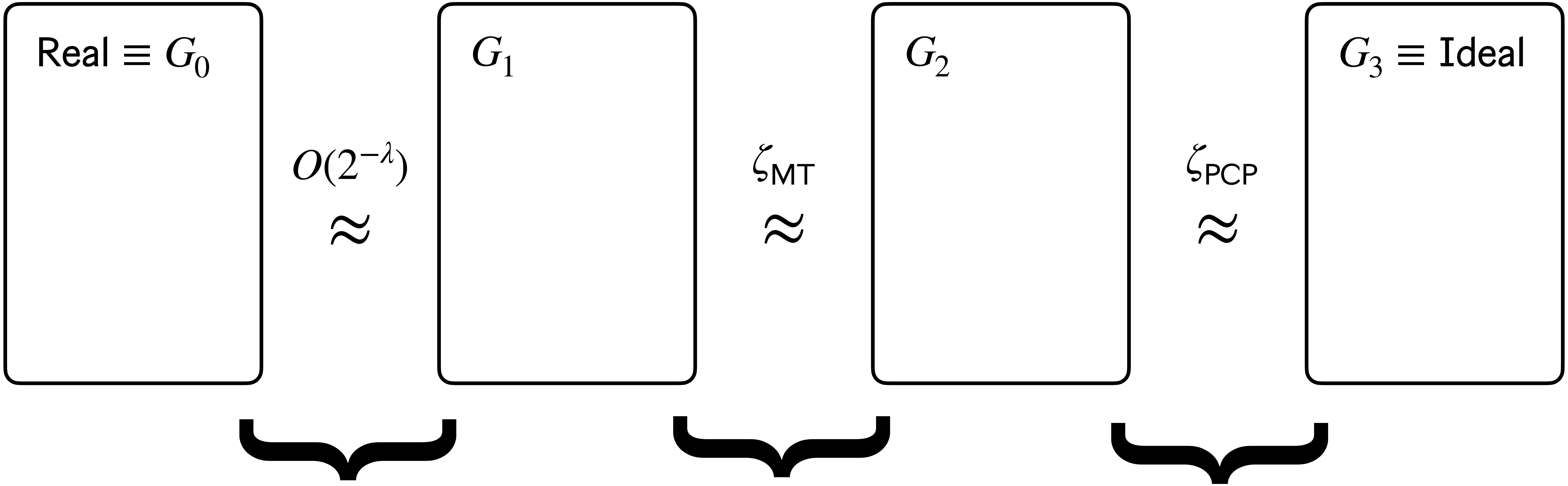
$\overset{O(2^{-\lambda})}{\approx}$

$\overset{\zeta_{\mathsf{PCP}}}{\approx}$

FS input hard
to predict

PCP honest-verifier
ZK

Micali has UC-friendly ZK



Real $\equiv G_0$

$O(2^{-\lambda})$ $\approx$

G_1

ζ_{MT} $\approx$

G_2

ζ_{PCP} $\approx$

$G_3 \equiv$ Ideal

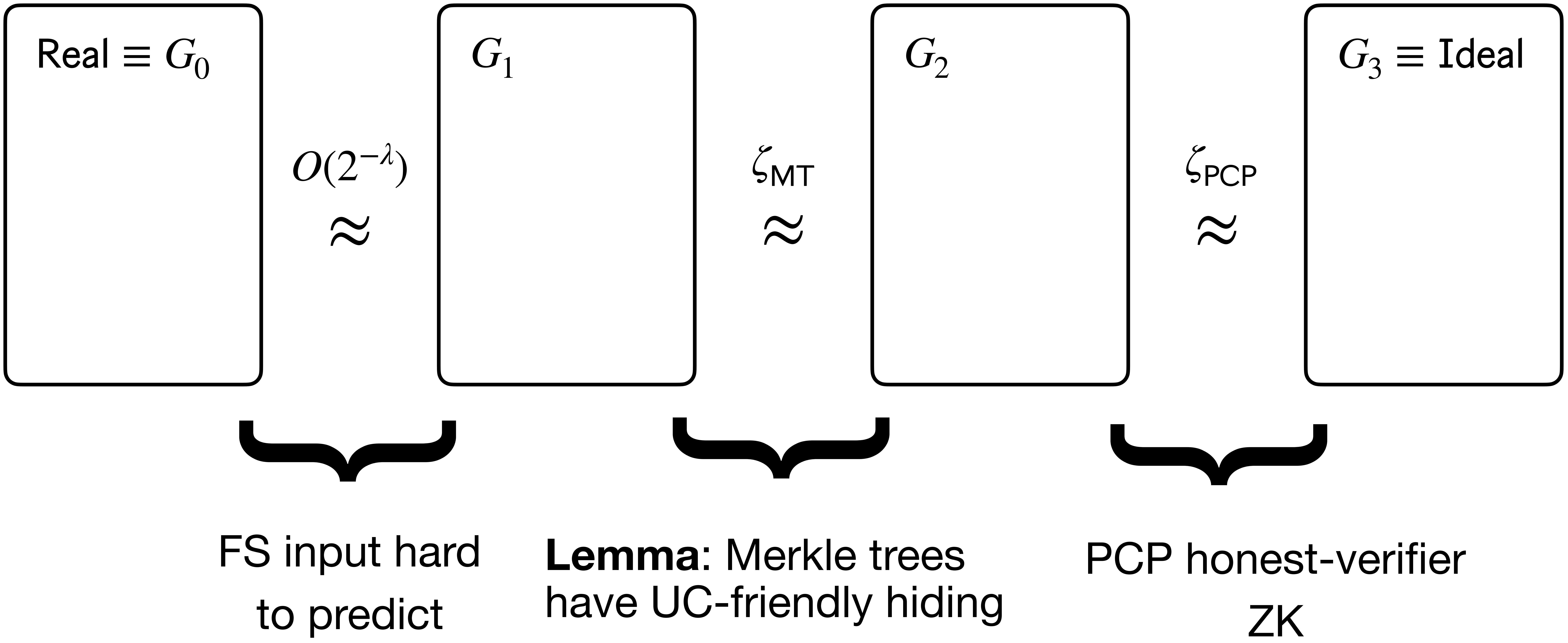
FS input hard to predict

Lemma: Merkle trees have UC-friendly hiding

PCP honest-verifier ZK

Micali has UC-friendly ZK

Follows similarly to standard Micali ZK + Merkle trees are UC-friendly.



$\mathsf{Real} \equiv G_0$

$O(2^{-\lambda})$ $\approx$

G_1

ζ_{MT} $\approx$

G_2

ζ_{PCP} $\approx$

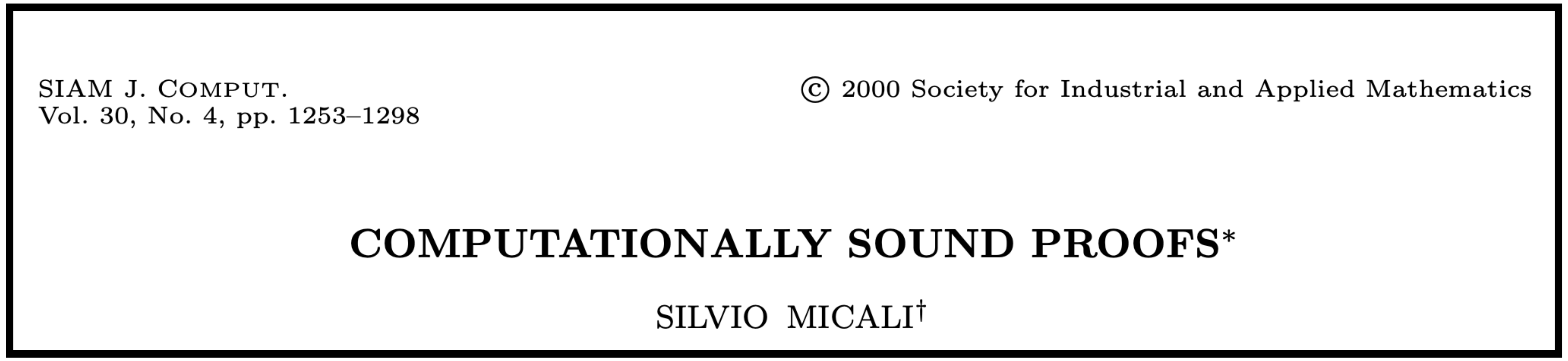
$G_3 \equiv \mathsf{Ideal}$

FS input hard to predict

Lemma: Merkle trees have UC-friendly hiding

PCP honest-verifier ZK

Micali’s construction I



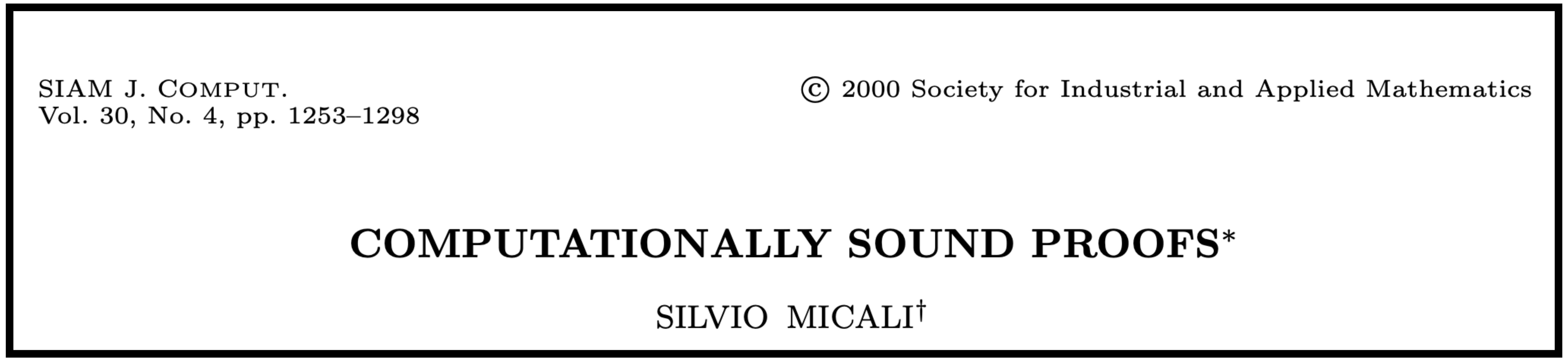
SIAM J. Comput.
Vol. 30, No. 4, pp. 1253–1298

© 2000 Society for Industrial and Applied Mathematics

COMPUTATIONALLY SOUND PROOFS*

SILVIO MICALI†

Micali’s construction I



SIAM J. Comput.
Vol. 30, No. 4, pp. 1253–1298

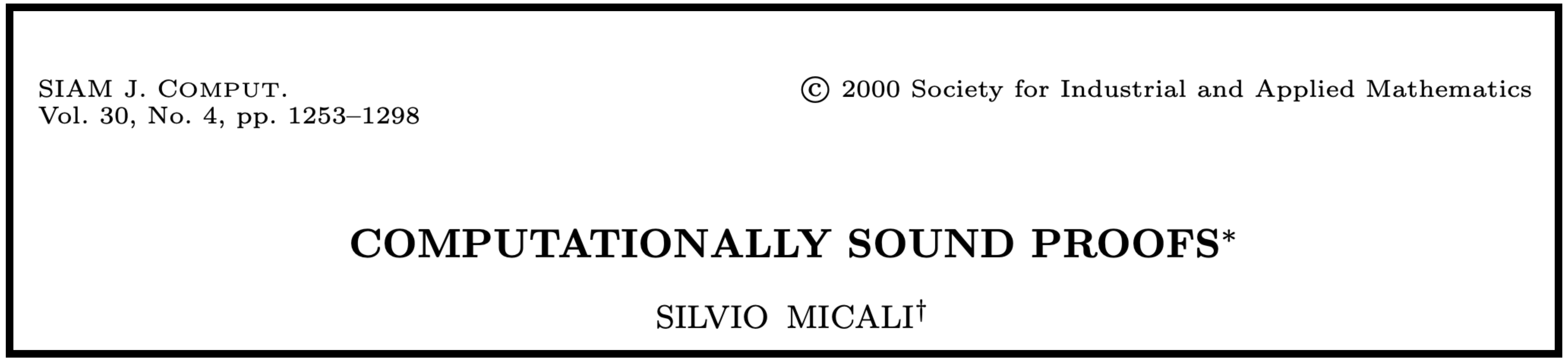
© 2000 Society for Industrial and Applied Mathematics

COMPUTATIONALLY SOUND PROOFS*

SILVIO MICALI†

Canonical construction of zkSNARK in the ROM

Micali’s construction I



SIAM J. Comput.
Vol. 30, No. 4, pp. 1253–1298

© 2000 Society for Industrial and Applied Mathematics

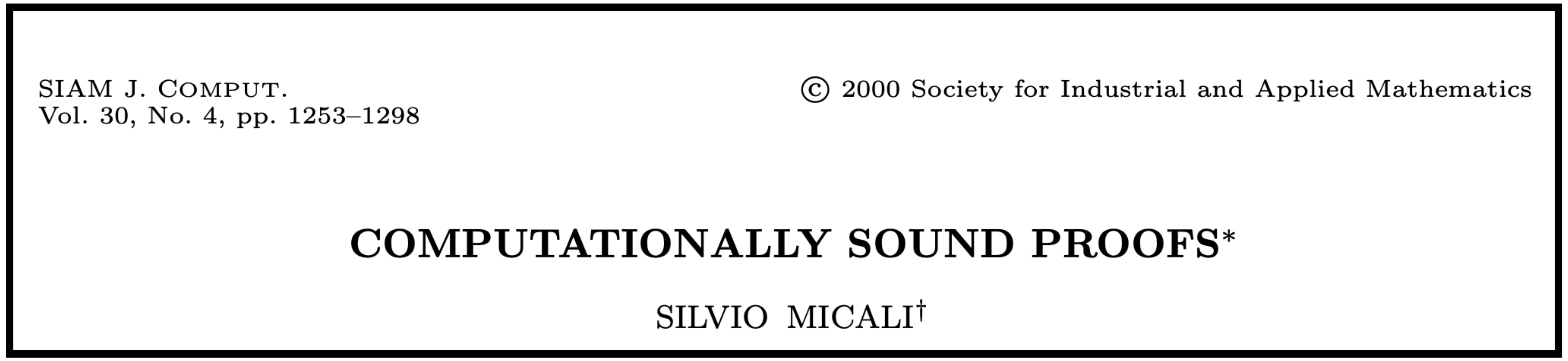
COMPUTATIONALLY SOUND PROOFS*

SILVIO MICALI†

Canonical construction of zkSNARK in the ROM

Straightline black-box extractor: compatible with UC!

Micali’s construction I



SIAM J. Comput.
Vol. 30, No. 4, pp. 1253–1298

© 2000 Society for Industrial and Applied Mathematics

COMPUTATIONALLY SOUND PROOFS[*]

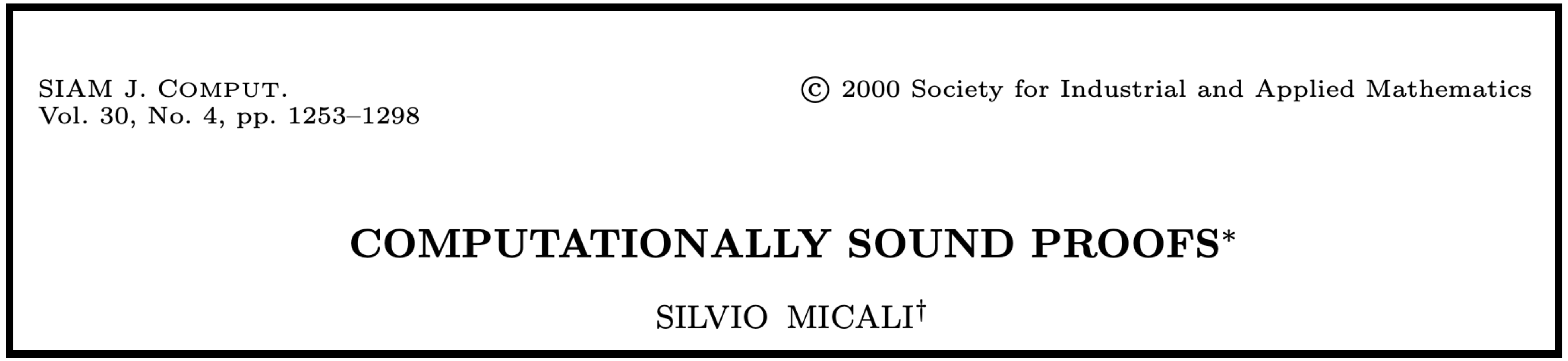
SILVIO MICALI[†]

Canonical construction of zkSNARK in the ROM

Straightline black-box extractor: compatible with UC!

Proofs are **non-malleable**: also required for UC-security!

Micali’s construction I



SIAM J. Comput.
Vol. 30, No. 4, pp. 1253–1298

© 2000 Society for Industrial and Applied Mathematics

COMPUTATIONALLY SOUND PROOFS*

SILVIO MICALI†

Canonical construction of zkSNARK in the ROM

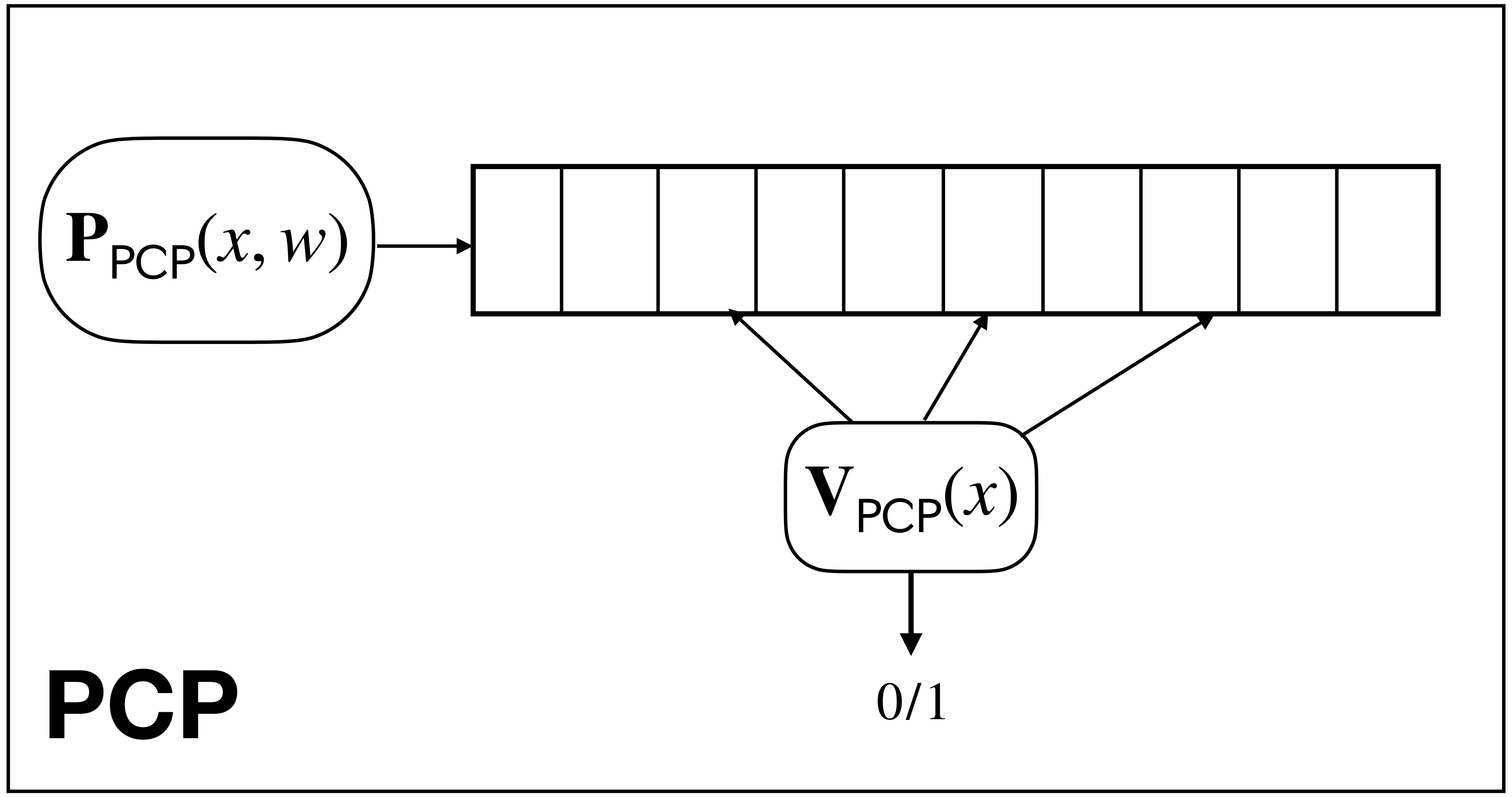
Straightline black-box extractor: compatible with UC!

Proofs are **non-malleable**: also required for UC-security!

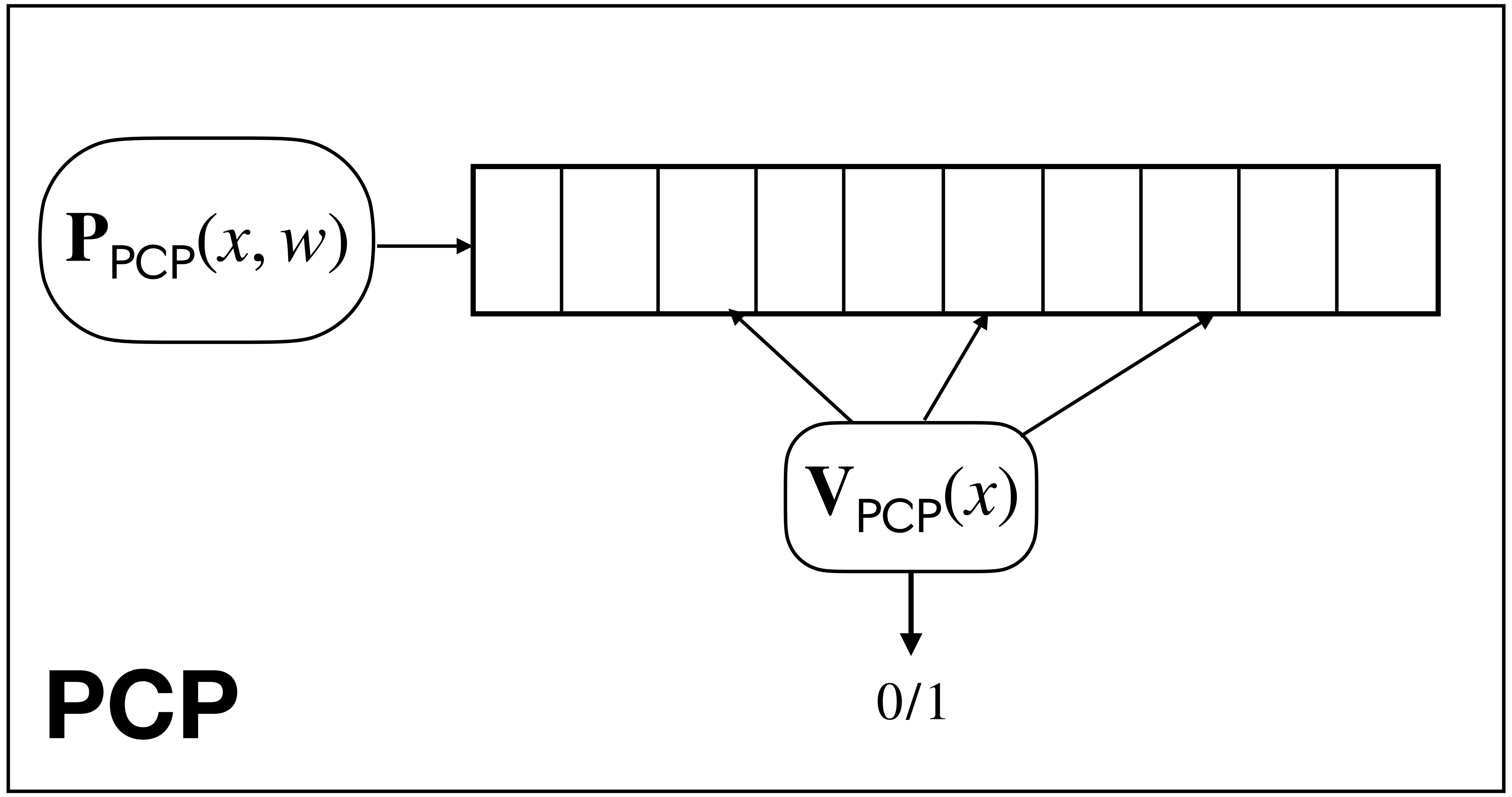
Stepping stone to BCS, which underlies **deployed** zkSNARKs

Micali’s construction II

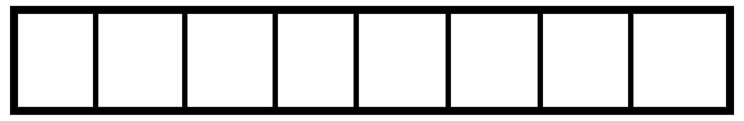
Micali's construction II


$\mathbf{P}_{\mathsf{PCP}}(x, w)$
$\mathbf{V}_{\mathsf{PCP}}(x)$
0/1
PCP

Micali's construction II


$\mathbf{P}_{\mathsf{PCP}}(x, w)$
$\mathbf{V}_{\mathsf{PCP}}(x)$
0/1
PCP

+



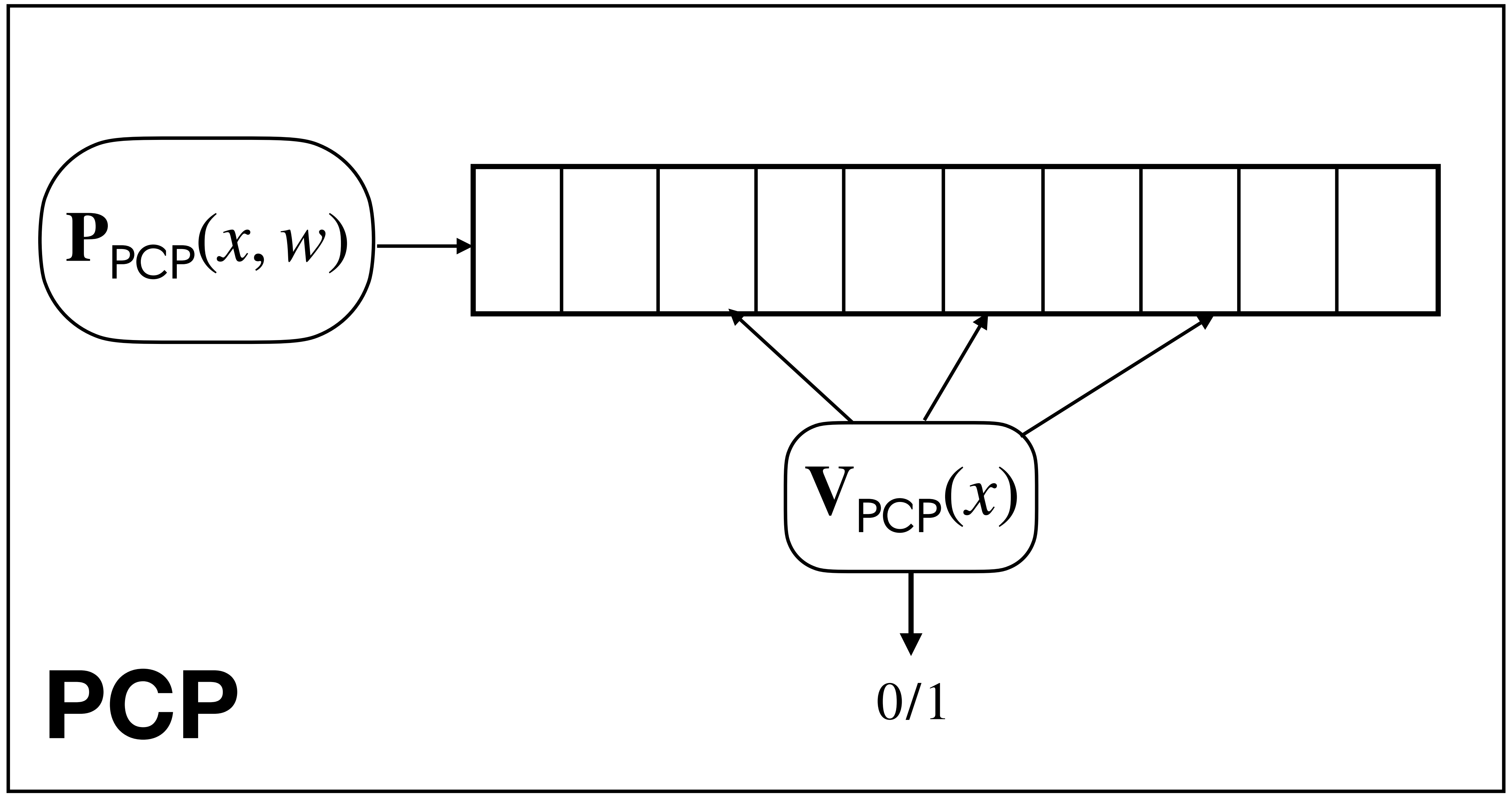
$\xrightarrow{\mathsf{Commit}^f} \mathsf{rt}, \mathsf{aux}$

$\mathsf{MTOpen}(\mathsf{rt}, I, \vec{a}, \mathsf{aux}) \to \pi$

$\mathsf{MTCheck}^f(\mathsf{rt}, I, \vec{a}, \pi) = 1 \iff$
$\tilde{\exists}\Pi : \mathsf{rt} = \mathsf{MTCommit}^f(\Pi) \wedge \Pi|_I = \vec{a}$

Merkle trees

Micali’s construction II


$\mathbf{P}_{\mathsf{PCP}}(x, w)$
$\mathbf{V}_{\mathsf{PCP}}(x)$
0/1
PCP

+

$\xrightarrow{\mathsf{Commit}^f} \mathsf{rt}, \mathsf{aux}$

$\mathsf{MTOpen}(\mathsf{rt}, I, \vec{a}, \mathsf{aux}) \to \pi$

$\mathsf{MTCheck}^f(\mathsf{rt}, I, \vec{a}, \pi) = 1 \iff$
$\tilde{\exists} \Pi : \mathsf{rt} = \mathsf{MTCommit}^f(\Pi) \wedge \Pi|_I = \vec{a}$

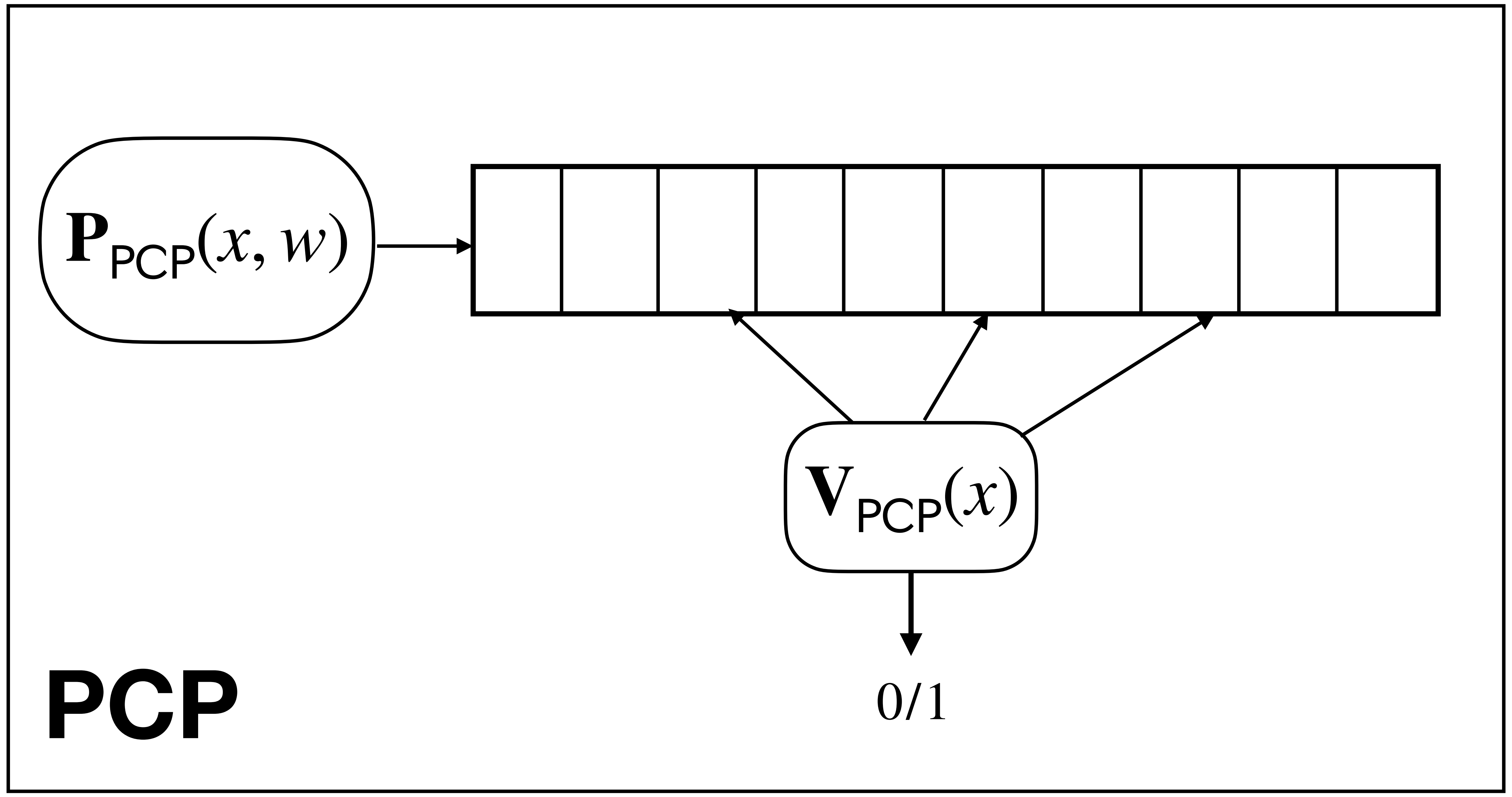
Merkle trees

+

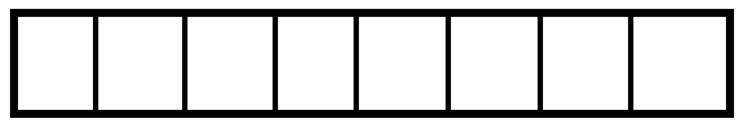
Fiat Shamir

=

Micali's construction II


$\mathbf{P}_{\mathsf{PCP}}(x, w)$
$\mathbf{V}_{\mathsf{PCP}}(x)$
0/1
PCP

+



$\xrightarrow{\mathsf{Commit}^f}$ rt, aux

$\mathsf{MTOpen}(\mathsf{rt}, I, \vec{a}, \mathsf{aux}) \to \pi$

$\mathsf{MTCheck}^f(\mathsf{rt}, I, \vec{a}, \pi) = 1 \iff$
$\tilde{\exists}\Pi : \mathsf{rt} = \mathsf{MTCommit}^f(\Pi) \wedge \Pi|_I = \vec{a}$

Merkle trees

+

Fiat Shamir

=

zkSNARK in the ROM

Micali’s construction III

Commit to PCP string using MT,
then apply FS transform

Micali’s construction III

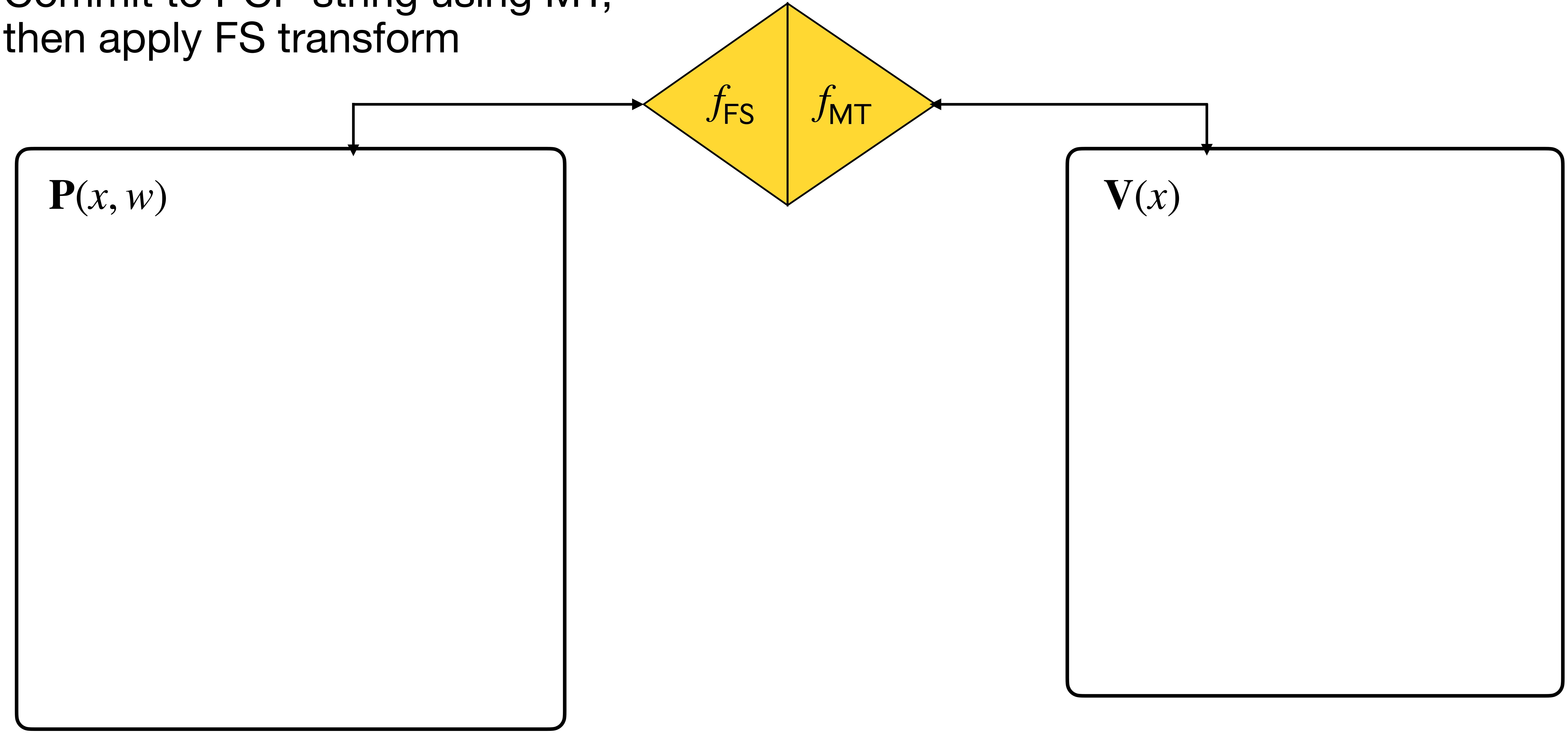
Commit to PCP string using MT,
then apply FS transform

$\mathbf{P}(x, w)$

$\mathbf{V}(x)$

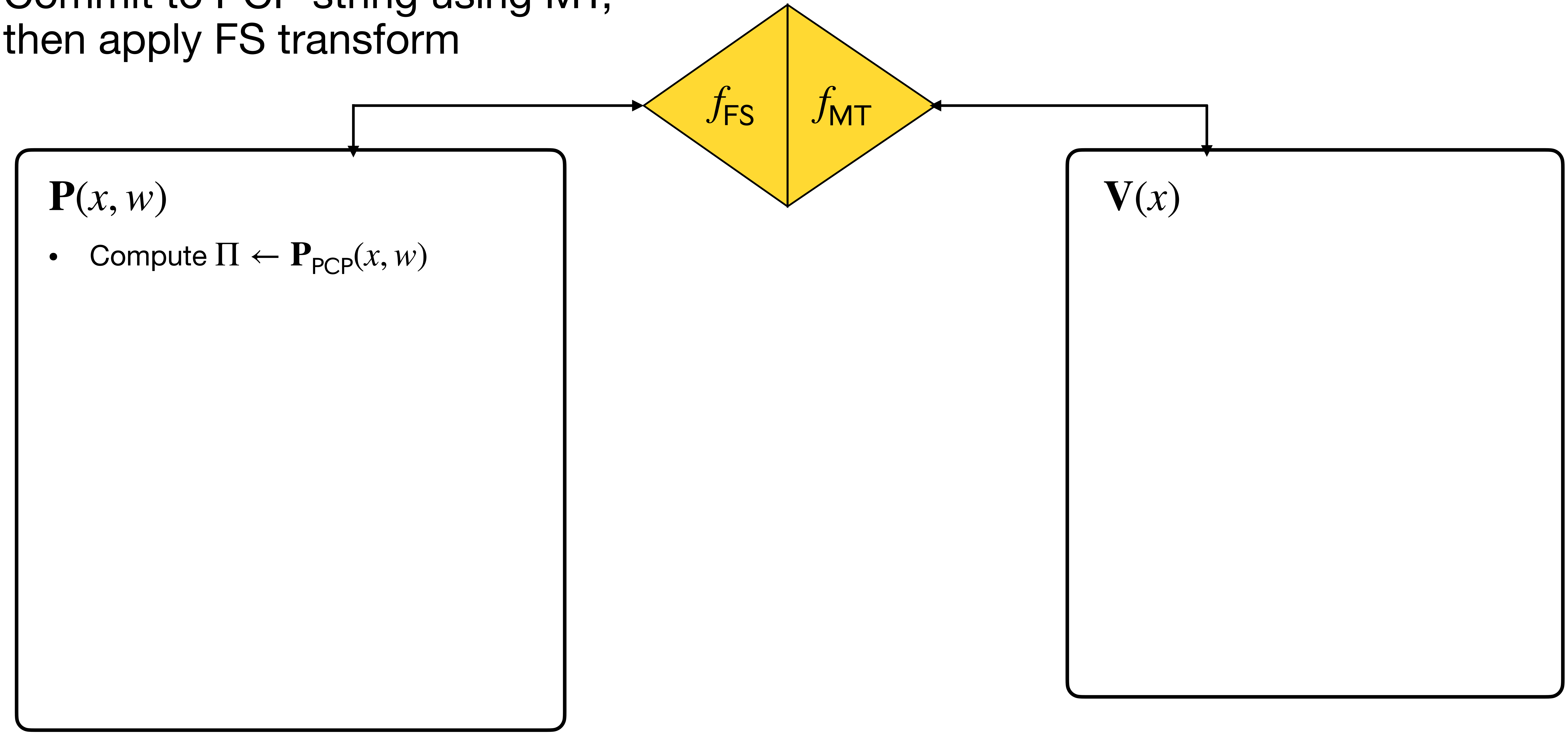
Micali’s construction III

Commit to PCP string using MT, then apply FS transform



Micali’s construction III

Commit to PCP string using MT,
then apply FS transform



f_{FS} f_{MT}

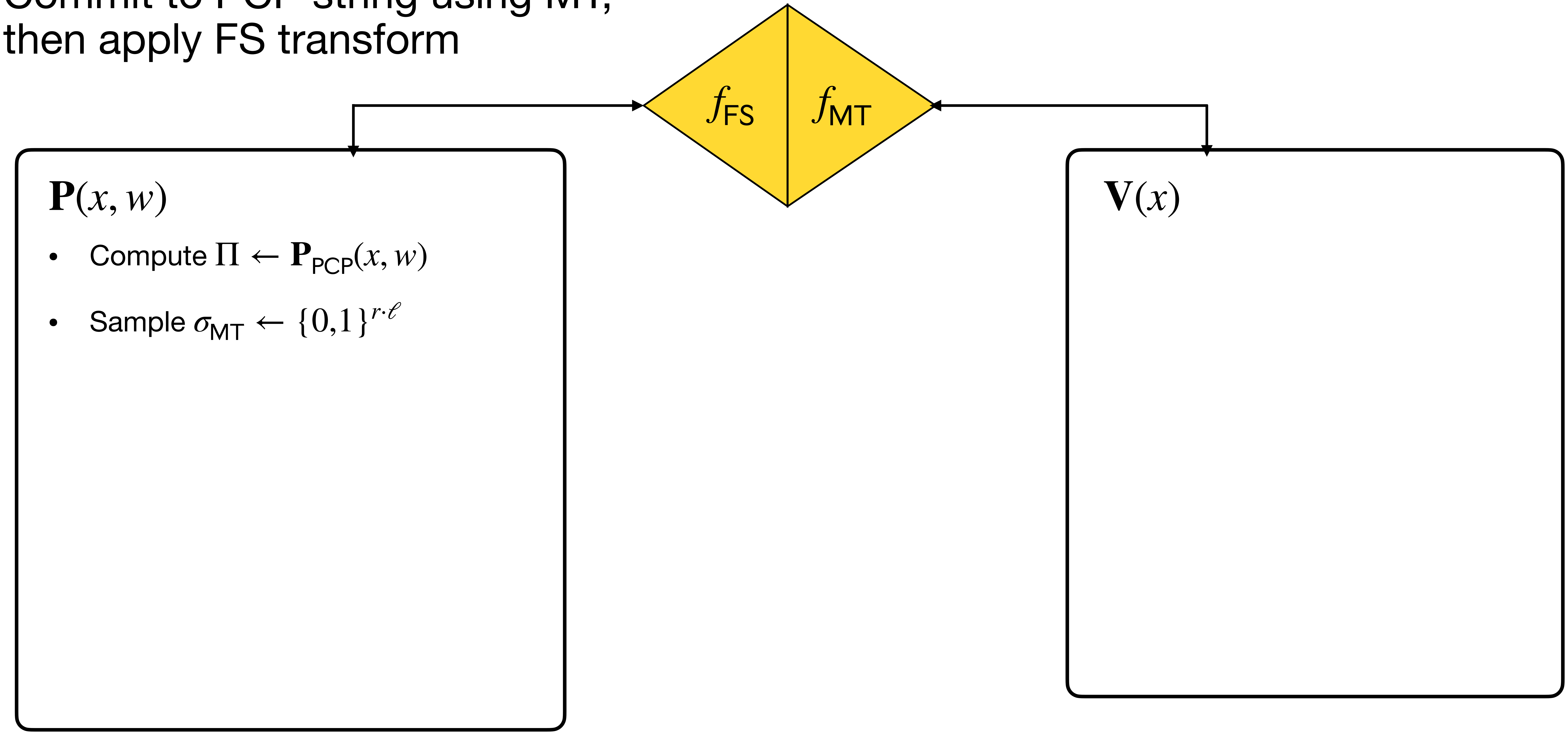
$\mathbf{P}(x, w)$

- Compute $\Pi \leftarrow \mathbf{P}_{\text{PCP}}(x, w)$

$\mathbf{V}(x)$

Micali’s construction III

Commit to PCP string using MT,
then apply FS transform



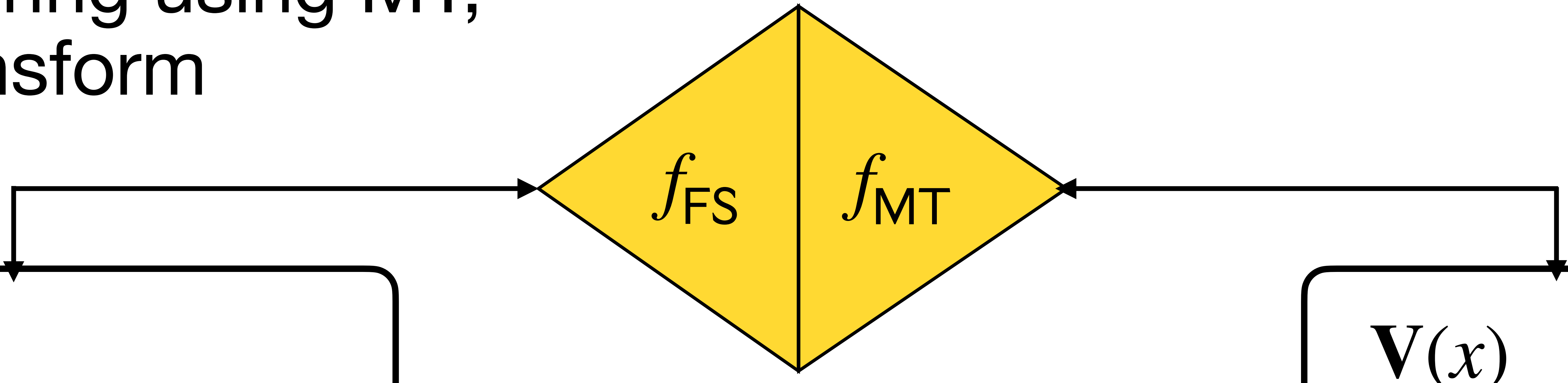
$\mathbf{P}(x, w)$

- Compute $\Pi \leftarrow \mathbf{P}_{\mathsf{PCP}}(x, w)$
- Sample $\sigma_{\mathsf{MT}} \leftarrow \{0,1\}^{r \cdot \ell}$

$\mathbf{V}(x)$

Micali’s construction III

Commit to PCP string using MT,
then apply FS transform



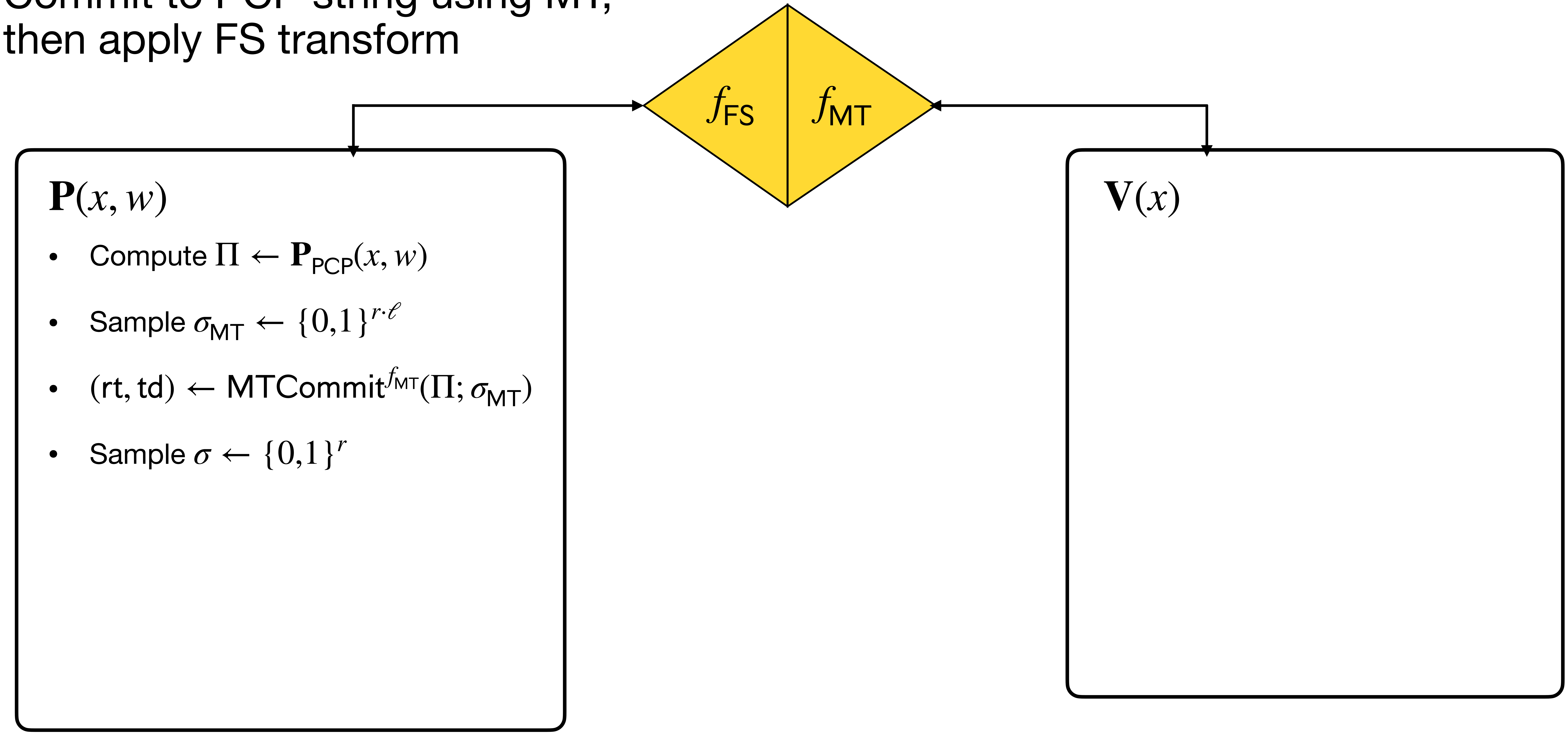
$\mathbf{P}(x, w)$

- Compute $\Pi \leftarrow \mathbf{P}_{\mathsf{PCP}}(x, w)$
- Sample $\sigma_{\mathsf{MT}} \leftarrow \{0,1\}^{r \cdot \ell}$
- $(\mathsf{rt}, \mathsf{td}) \leftarrow \mathsf{MTCommit}^{f_{\mathsf{MT}}}(\Pi; \sigma_{\mathsf{MT}})$

$\mathbf{V}(x)$

Micali’s construction III

Commit to PCP string using MT, then apply FS transform



f_{FS} f_{MT}

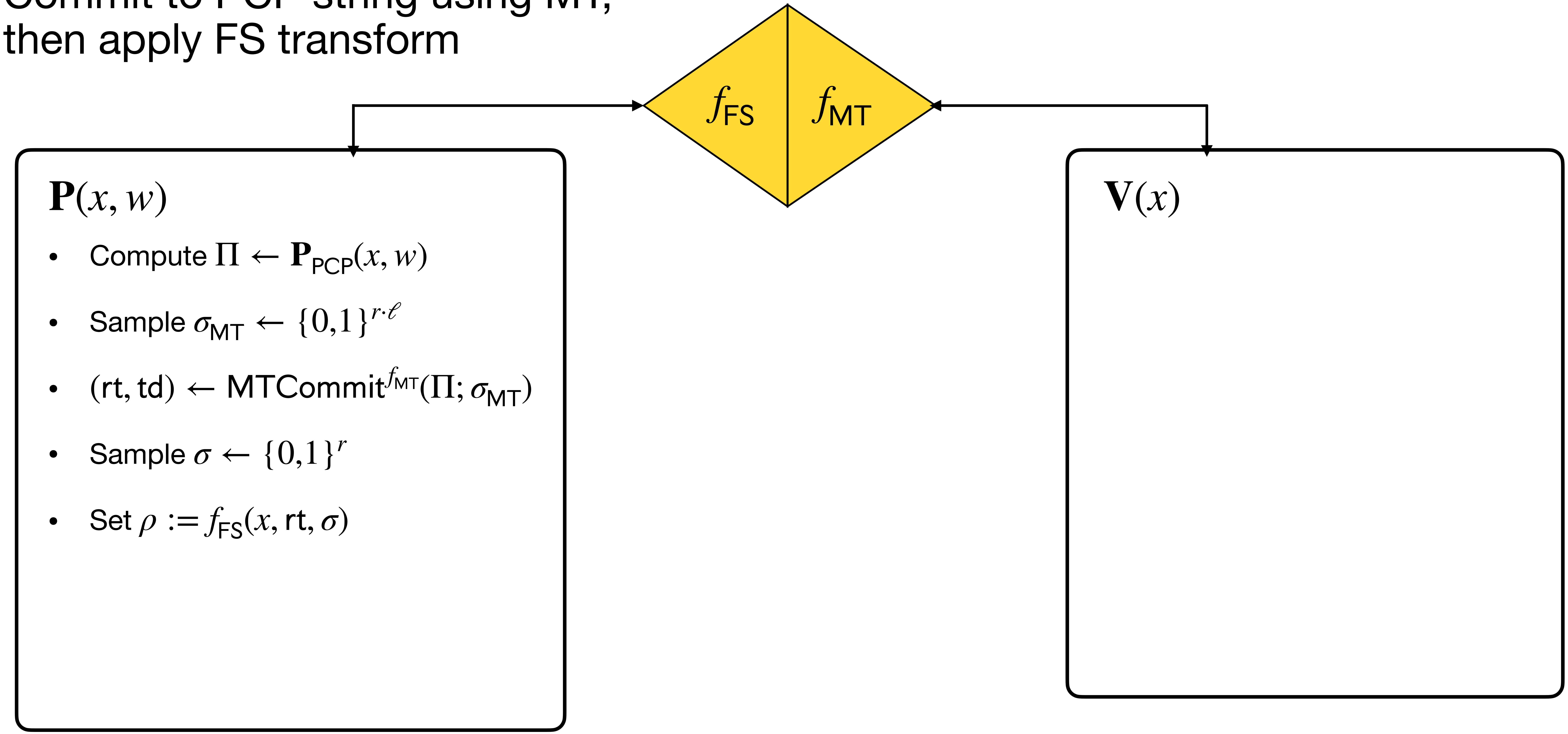
$\mathbf{P}(x, w)$

- Compute $\Pi \leftarrow \mathbf{P}_{\mathsf{PCP}}(x, w)$
- Sample $\sigma_{\mathsf{MT}} \leftarrow \{0,1\}^{r \cdot \ell}$
- $(\mathsf{rt}, \mathsf{td}) \leftarrow \mathsf{MTCommit}^{f_{\mathsf{MT}}}(\Pi; \sigma_{\mathsf{MT}})$
- Sample $\sigma \leftarrow \{0,1\}^{r}$

$\mathbf{V}(x)$

Micali’s construction III

Commit to PCP string using MT,
then apply FS transform



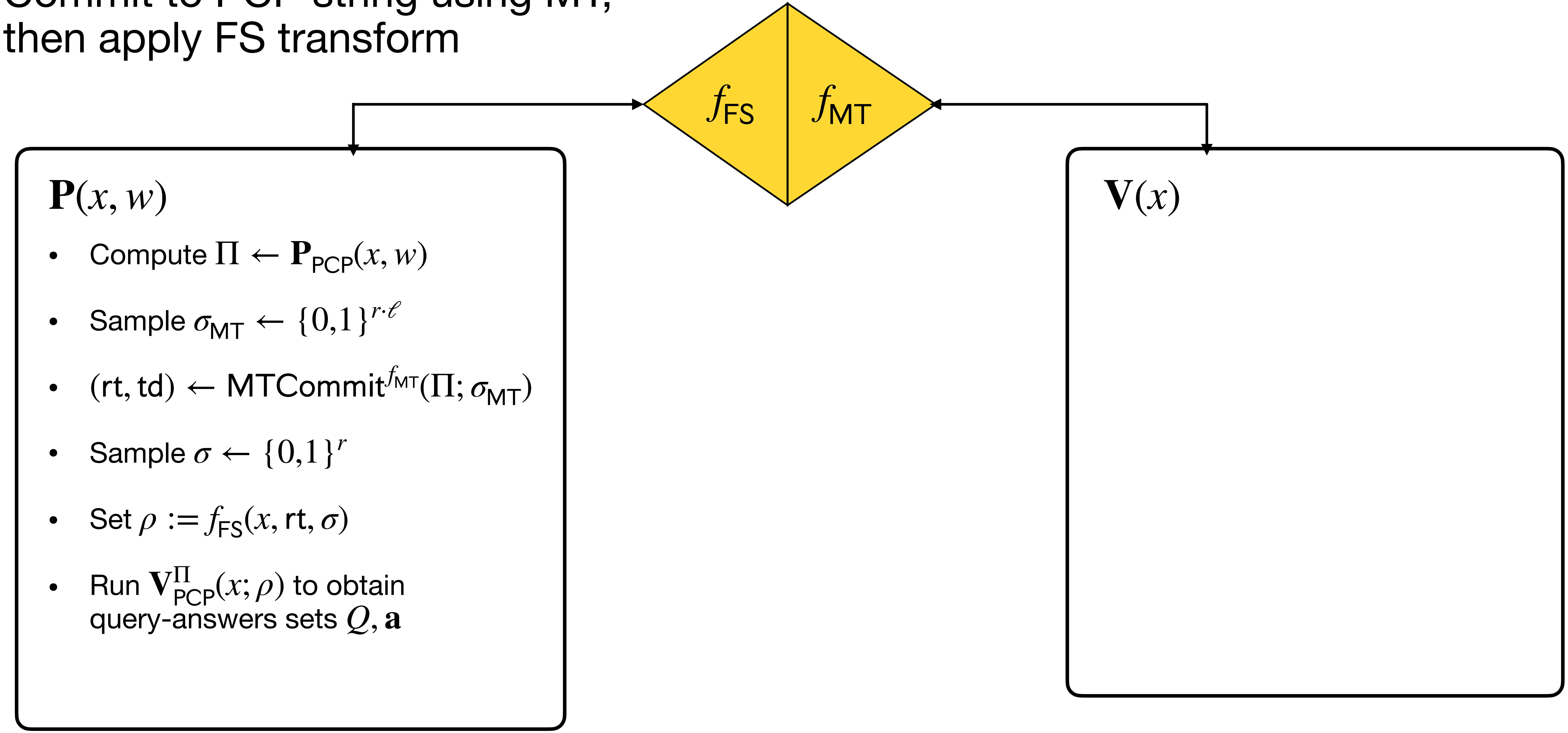
$\mathbf{P}(x, w)$

- Compute $\Pi \leftarrow \mathbf{P}_{\mathsf{PCP}}(x, w)$
- Sample $\sigma_{\mathsf{MT}} \leftarrow \{0,1\}^{r \cdot \ell}$
- $(\mathsf{rt}, \mathsf{td}) \leftarrow \mathsf{MTCommit}^{f_{\mathsf{MT}}}(\Pi; \sigma_{\mathsf{MT}})$
- Sample $\sigma \leftarrow \{0,1\}^{r}$
- Set $\rho := f_{\mathsf{FS}}(x, \mathsf{rt}, \sigma)$

$\mathbf{V}(x)$

Micali's construction III

Commit to PCP string using MT,
then apply FS transform



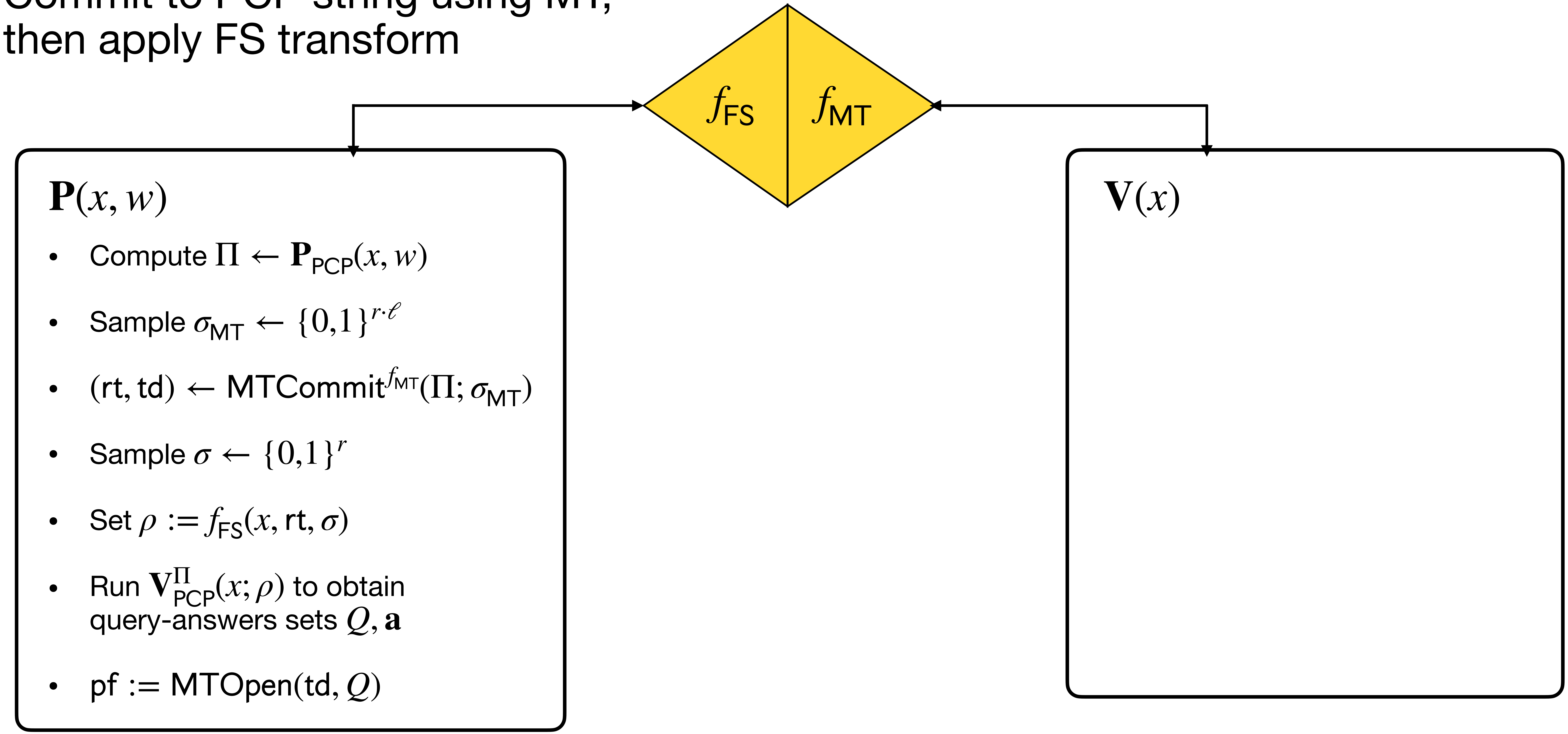
$\mathbf{P}(x, w)$

- Compute $\Pi \leftarrow \mathbf{P}_{\mathsf{PCP}}(x, w)$
- Sample $\sigma_{\mathsf{MT}} \leftarrow \{0,1\}^{r \cdot \ell}$
- $(\mathsf{rt}, \mathsf{td}) \leftarrow \mathsf{MTCommit}^{f_{\mathsf{MT}}}(\Pi; \sigma_{\mathsf{MT}})$
- Sample $\sigma \leftarrow \{0,1\}^r$
- Set $\rho := f_{\mathsf{FS}}(x, \mathsf{rt}, \sigma)$
- Run $\mathbf{V}_{\mathsf{PCP}}^{\Pi}(x; \rho)$ to obtain query-answers sets $Q, \mathbf{a}$

$\mathbf{V}(x)$

Micali’s construction III

Commit to PCP string using MT,
then apply FS transform



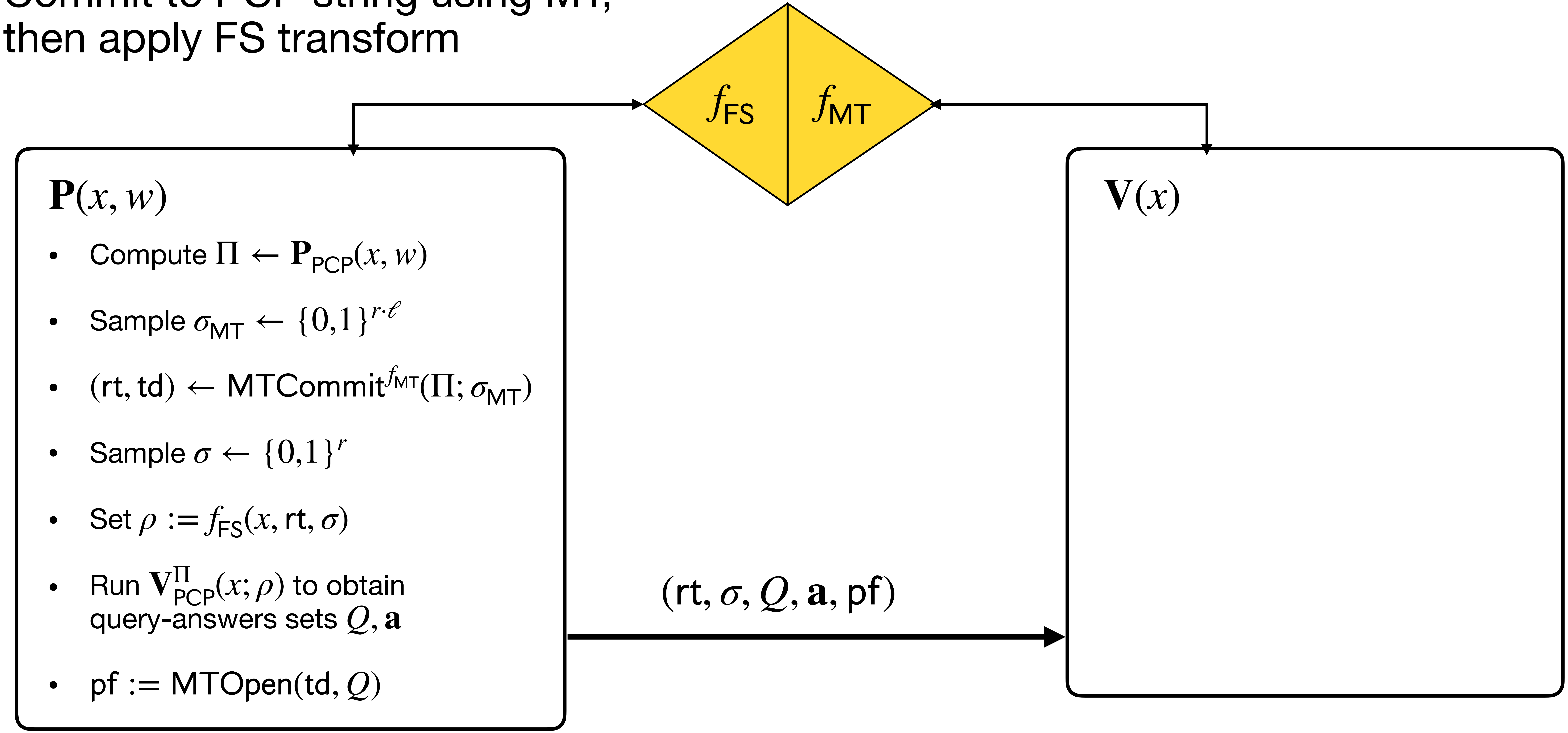
$\mathbf{P}(x, w)$

- Compute $\Pi \leftarrow \mathbf{P}_{\mathsf{PCP}}(x, w)$
- Sample $\sigma_{\mathsf{MT}} \leftarrow \{0,1\}^{r \cdot \ell}$
- $(\mathsf{rt}, \mathsf{td}) \leftarrow \mathsf{MTCommit}^{f_{\mathsf{MT}}}(\Pi; \sigma_{\mathsf{MT}})$
- Sample $\sigma \leftarrow \{0,1\}^{r}$
- Set $\rho := f_{\mathsf{FS}}(x, \mathsf{rt}, \sigma)$
- Run $\mathbf{V}^{\Pi}_{\mathsf{PCP}}(x; \rho)$ to obtain query-answers sets $Q, \mathbf{a}$
- $\mathsf{pf} := \mathsf{MTOpen}(\mathsf{td}, Q)$

$\mathbf{V}(x)$

Micali's construction III

Commit to PCP string using MT,
then apply FS transform



f_{FS} f_{MT}

$\mathbf{P}(x, w)$

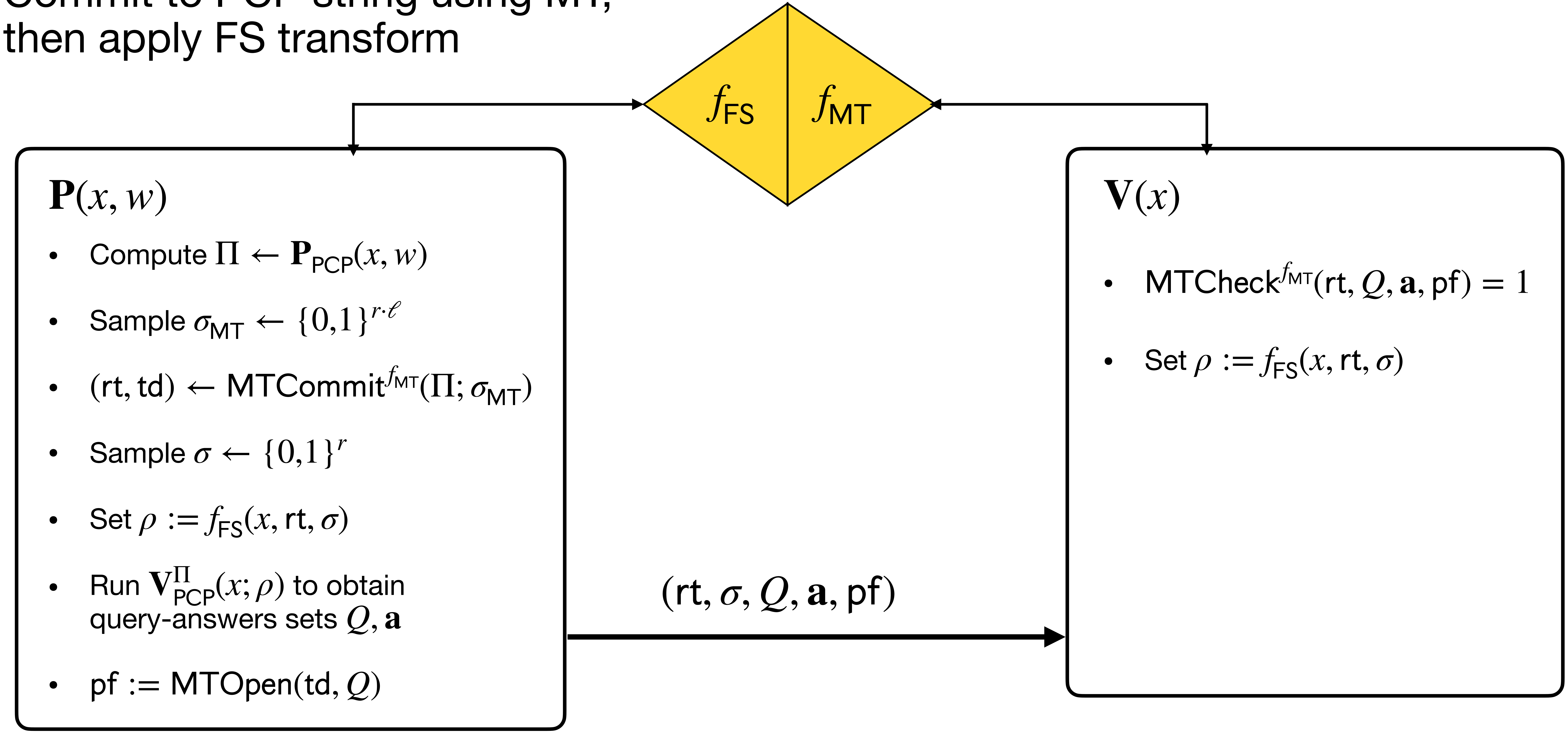
- Compute $\Pi \leftarrow \mathbf{P}_{\mathsf{PCP}}(x, w)$
- Sample $\sigma_{\mathsf{MT}} \leftarrow \{0,1\}^{r \cdot \ell}$
- $(\mathsf{rt}, \mathsf{td}) \leftarrow \mathsf{MTCommit}^{f_{\mathsf{MT}}}(\Pi; \sigma_{\mathsf{MT}})$
- Sample $\sigma \leftarrow \{0,1\}^r$
- Set $\rho := f_{\mathsf{FS}}(x, \mathsf{rt}, \sigma)$
- Run $\mathbf{V}^{\Pi}_{\mathsf{PCP}}(x; \rho)$ to obtain query-answers sets $Q, \mathbf{a}$
- $\mathsf{pf} := \mathsf{MTOpen}(\mathsf{td}, Q)$

$(\mathsf{rt}, \sigma, Q, \mathbf{a}, \mathsf{pf})$

$\mathbf{V}(x)$

Micali’s construction III

Commit to PCP string using MT,
then apply FS transform



$\mathbf{P}(x, w)$

- Compute $\Pi \leftarrow \mathbf{P}_{\mathsf{PCP}}(x, w)$
- Sample $\sigma_{\mathsf{MT}} \leftarrow \{0,1\}^{r \cdot \ell}$
- $(\mathsf{rt}, \mathsf{td}) \leftarrow \mathsf{MTCommit}^{f_{\mathsf{MT}}}(\Pi; \sigma_{\mathsf{MT}})$
- Sample $\sigma \leftarrow \{0,1\}^r$
- Set $\rho := f_{\mathsf{FS}}(x, \mathsf{rt}, \sigma)$
- Run $\mathbf{V}^{\Pi}_{\mathsf{PCP}}(x; \rho)$ to obtain query-answers sets $Q, \mathbf{a}$
- $\mathsf{pf} := \mathsf{MTOpen}(\mathsf{td}, Q)$

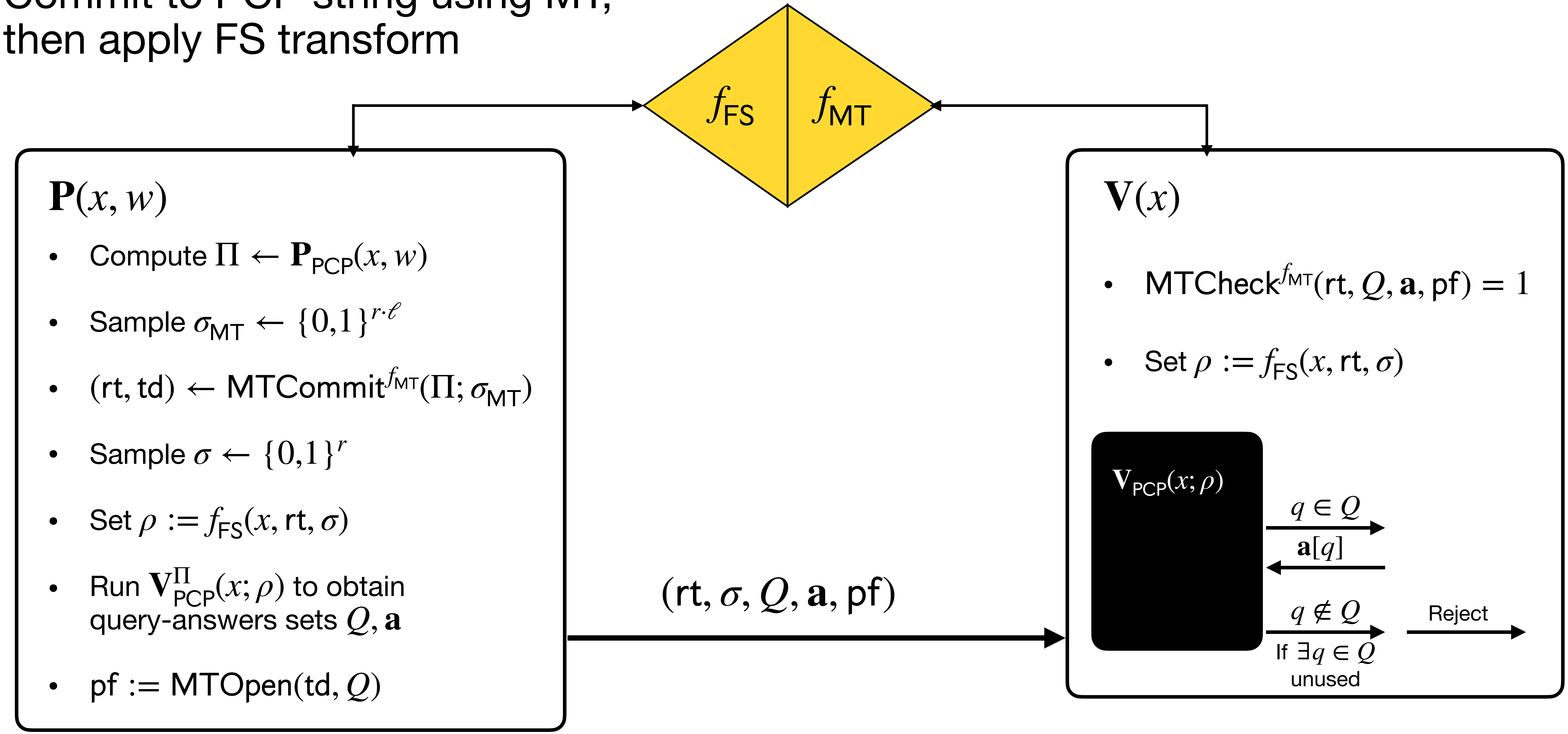
$(\mathsf{rt}, \sigma, Q, \mathbf{a}, \mathsf{pf})$

$\mathbf{V}(x)$

- $\mathsf{MTCheck}^{f_{\mathsf{MT}}}(\mathsf{rt}, Q, \mathbf{a}, \mathsf{pf}) = 1$
- Set $\rho := f_{\mathsf{FS}}(x, \mathsf{rt}, \sigma)$

Micali's construction III

Commit to PCP string using MT,
then apply FS transform



$\mathbf{P}(x, w)$

- Compute $\Pi \leftarrow \mathbf{P}_{\mathsf{PCP}}(x, w)$
- Sample $\sigma_{\mathsf{MT}} \leftarrow \{0,1\}^{r \cdot \ell}$
- $(\mathsf{rt}, \mathsf{td}) \leftarrow \mathsf{MTCommit}^{f_{\mathsf{MT}}}(\Pi; \sigma_{\mathsf{MT}})$
- Sample $\sigma \leftarrow \{0,1\}^{r}$
- Set $\rho := f_{\mathsf{FS}}(x, \mathsf{rt}, \sigma)$
- Run $\mathbf{V}^{\Pi}_{\mathsf{PCP}}(x; \rho)$ to obtain query-answers sets $Q, \mathbf{a}$
- $\mathsf{pf} := \mathsf{MTOpen}(\mathsf{td}, Q)$

$(\mathsf{rt}, \sigma, Q, \mathbf{a}, \mathsf{pf})$

$\mathbf{V}(x)$

- $\mathsf{MTCheck}^{f_{\mathsf{MT}}}(\mathsf{rt}, Q, \mathbf{a}, \mathsf{pf}) = 1$
- Set $\rho := f_{\mathsf{FS}}(x, \mathsf{rt}, \sigma)$

$\mathbf{V}_{\mathsf{PCP}}(x; \rho)$
$q \in Q$
$\mathbf{a}[q]$
$q \notin Q$
Reject
If $\exists q \in Q$ unused

UC Security I

[Canetti 2001]

UC Security I

[Canetti 2001]

- Motivation: Modular security analysis of protocols

UC Security I

[Canetti 2001]

- Motivation: Modular security analysis of protocols
- Why UC? ‘Gold-standard’ + vast literature

UC Security I

[Canetti 2001]

- Motivation: Modular security analysis of protocols
- Why UC? ‘Gold-standard’ + vast literature

Composition Theorem

UC Security I

π : protocol

φ : ideal functionality

ρ : calling protocol

- Motivation: Modular security analysis of protocols
- Why UC? ‘Gold-standard’ + vast literature

Composition Theorem

UC Security I

π : protocol

φ : ideal functionality

ρ : calling protocol

- Motivation: Modular security analysis of protocols
- Why UC? ‘Gold-standard’ + vast literature

Composition Theorem

$$\pi \approx_{\text{UC}} \varphi$$

UC Security I

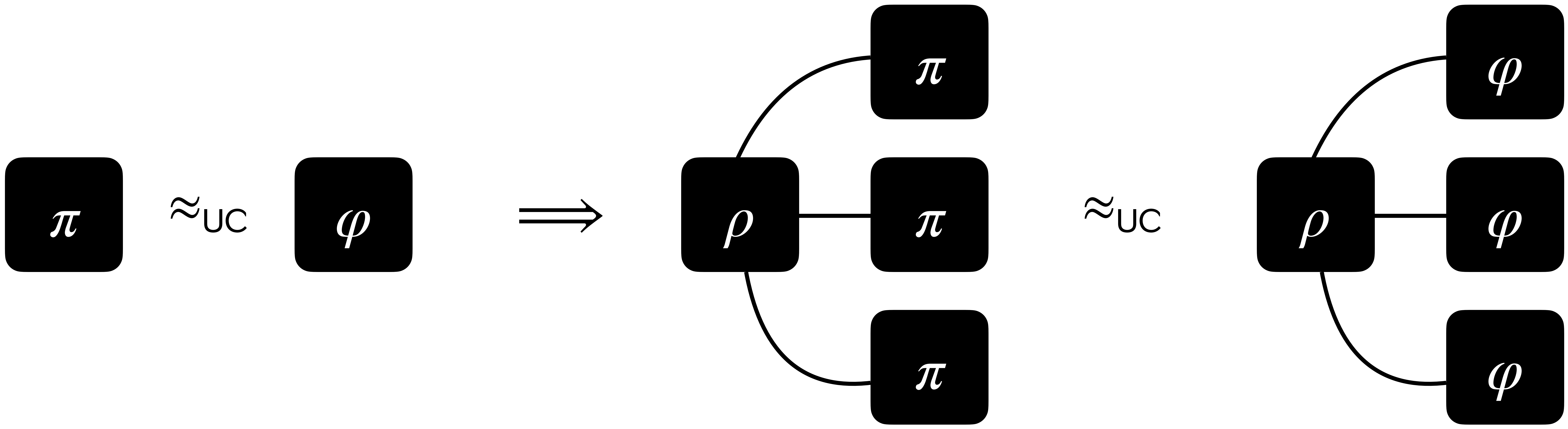
π : protocol

φ : ideal functionality

ρ : calling protocol

- Motivation: Modular security analysis of protocols
- Why UC? 'Gold-standard' + vast literature

Composition Theorem



UC Security II

UC Security II

π	: protocol	$\mathscr{E}$	: environment
$\mathscr{F}$	: ideal functionality	$\mathscr{A}$	: adversary
D	: dummy party	$\mathscr{S}$	: simulator

UC Security II

π	: protocol	$\mathscr{E}$	: environment
$\mathscr{F}$	: ideal functionality	$\mathscr{A}$	: adversary
D	: dummy party	$\mathscr{S}$	: simulator

$\pi \approx_{\text{UC}} \mathscr{F}$

UC Security II

Goal: Cannot distinguish protocol from idealized version.

π	: protocol	$\mathscr{E}$	: environment
$\mathscr{F}$	: ideal functionality	$\mathscr{A}$	: adversary
D	: dummy party	$\mathscr{S}$	: simulator

$$\pi \approx_{\text{UC}} \mathscr{F}$$

UC Security II

Goal: Cannot distinguish protocol from idealized version.

$$\pi \approx_{\text{UC}} \mathscr{F}$$
$$\iff$$
$$\forall \mathscr{A}, \exists \mathscr{S}, \forall \mathscr{E}$$

π	: protocol	$\mathscr{E}$	: environment
$\mathscr{F}$	: ideal functionality	$\mathscr{A}$	: adversary
D	: dummy party	$\mathscr{S}$	: simulator

π	: protocol	$\mathscr{E}$	: environment
$\mathscr{F}$	: ideal functionality	$\mathscr{A}$	: adversary
D	: dummy party	$\mathscr{S}$	: simulator

UC Security II

Goal: Cannot distinguish protocol from idealized version.

$$\pi \approx_{\text{UC}} \mathscr{F} \iff \forall \mathscr{A}, \exists \mathscr{S}, \forall \mathscr{E}$$

Real

UC Security II

Goal: Cannot distinguish protocol from idealized version.

π	: protocol	$\mathscr{E}$	: environment
$\mathscr{F}$	: ideal functionality	$\mathscr{A}$	: adversary
D	: dummy party	$\mathscr{S}$	: simulator

$$\pi \approx_{\text{UC}} \mathscr{F}$$
$$\iff$$
$$\forall \mathscr{A}, \exists \mathscr{S}, \forall \mathscr{E}$$

Real

$\approx$

Ideal

UC Security II

Goal: Cannot distinguish protocol from idealized version.

π : protocol

$\mathscr{F}$: ideal functionality

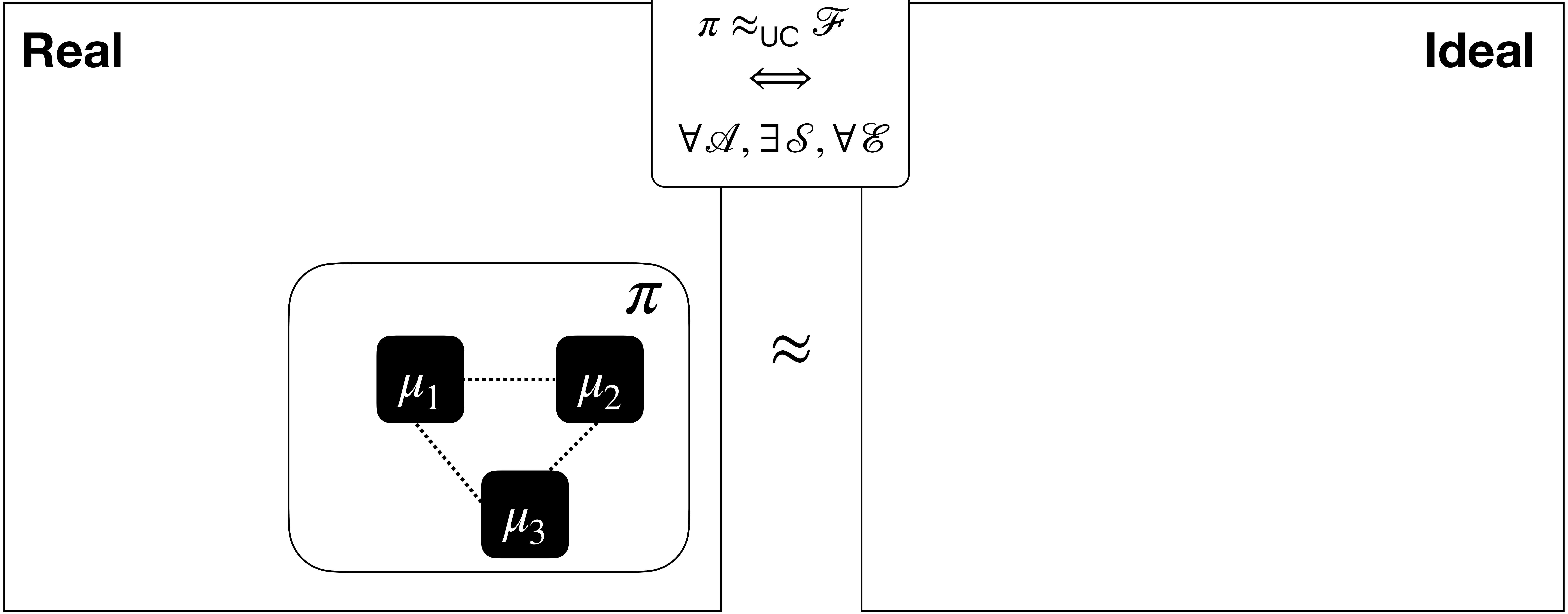
D : dummy party

$\mathscr{E}$: environment

$\mathscr{A}$: adversary

$\mathscr{S}$: simulator

$$\pi \approx_{\text{UC}} \mathscr{F} \iff \forall \mathscr{A}, \exists \mathscr{S}, \forall \mathscr{E}$$

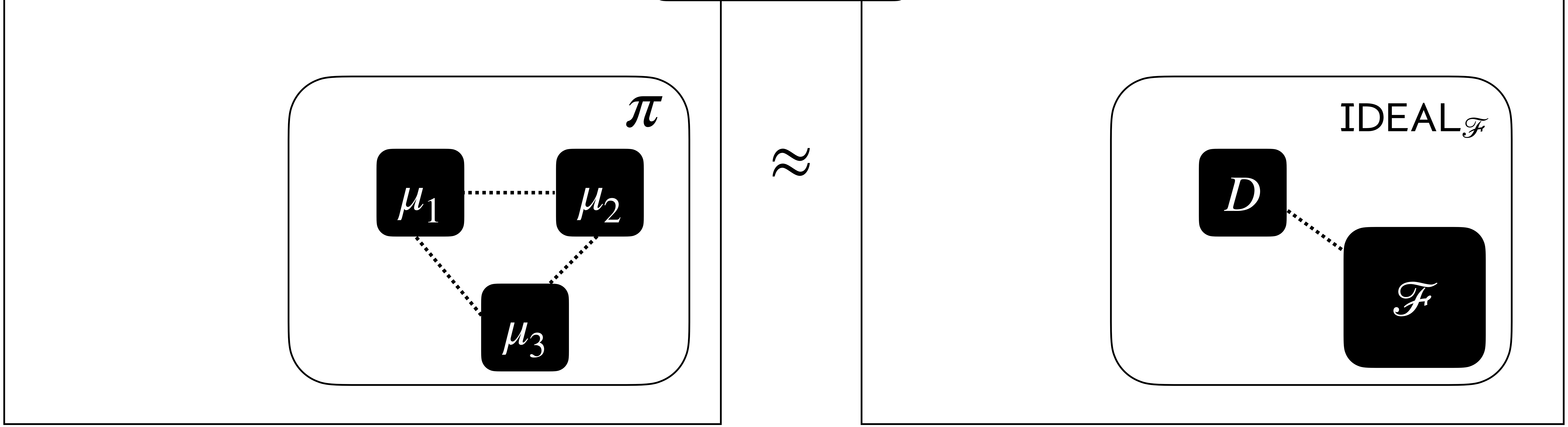


UC Security II

Goal: Cannot distinguish protocol from idealized version.

π	: protocol	$\mathscr{E}$	: environment
$\mathscr{F}$	: ideal functionality	$\mathscr{A}$	: adversary
D	: dummy party	$\mathscr{S}$	: simulator

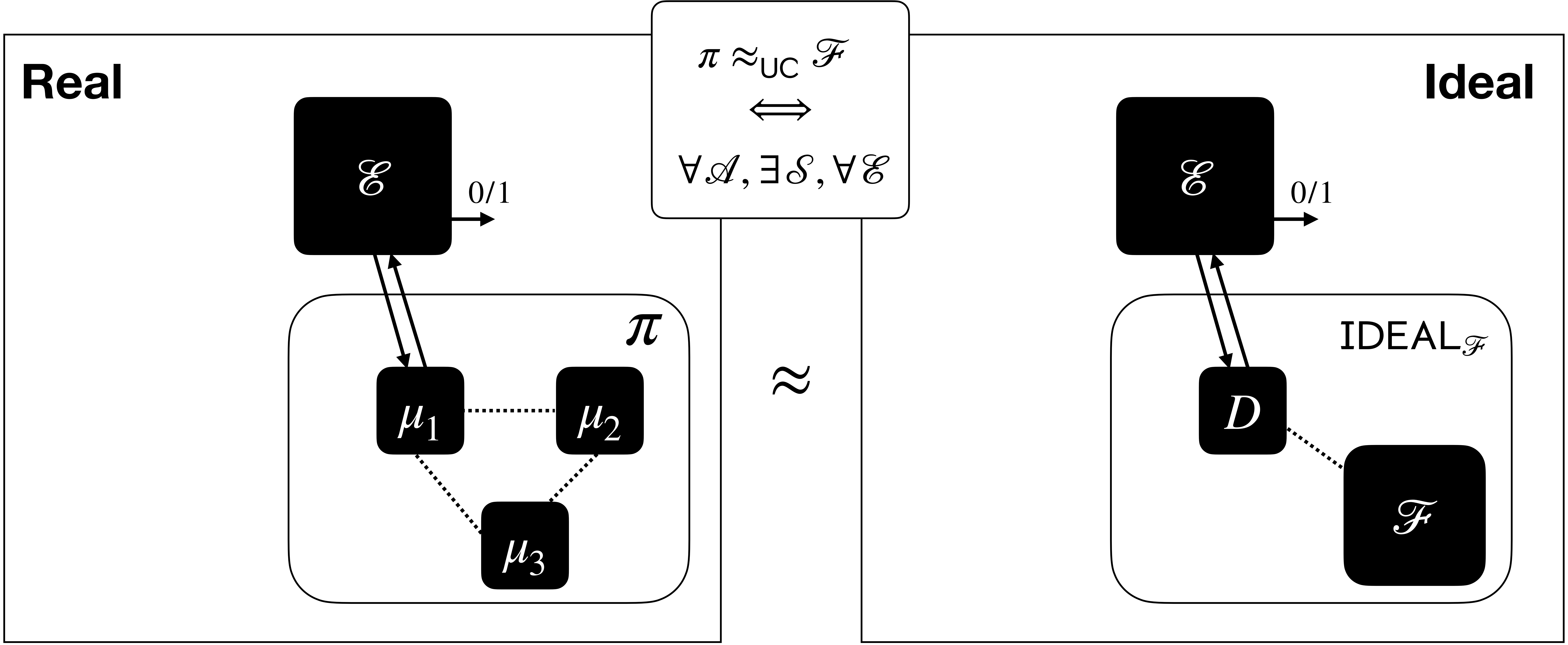
$$\pi \approx_{\text{UC}} \mathscr{F} \iff \forall \mathscr{A}, \exists \mathscr{S}, \forall \mathscr{E}$$



UC Security II

Goal: Cannot distinguish protocol from idealized version.

π	: protocol	$\mathscr{E}$	: environment
$\mathscr{F}$	: ideal functionality	$\mathscr{A}$	: adversary
D	: dummy party	$\mathscr{S}$	: simulator



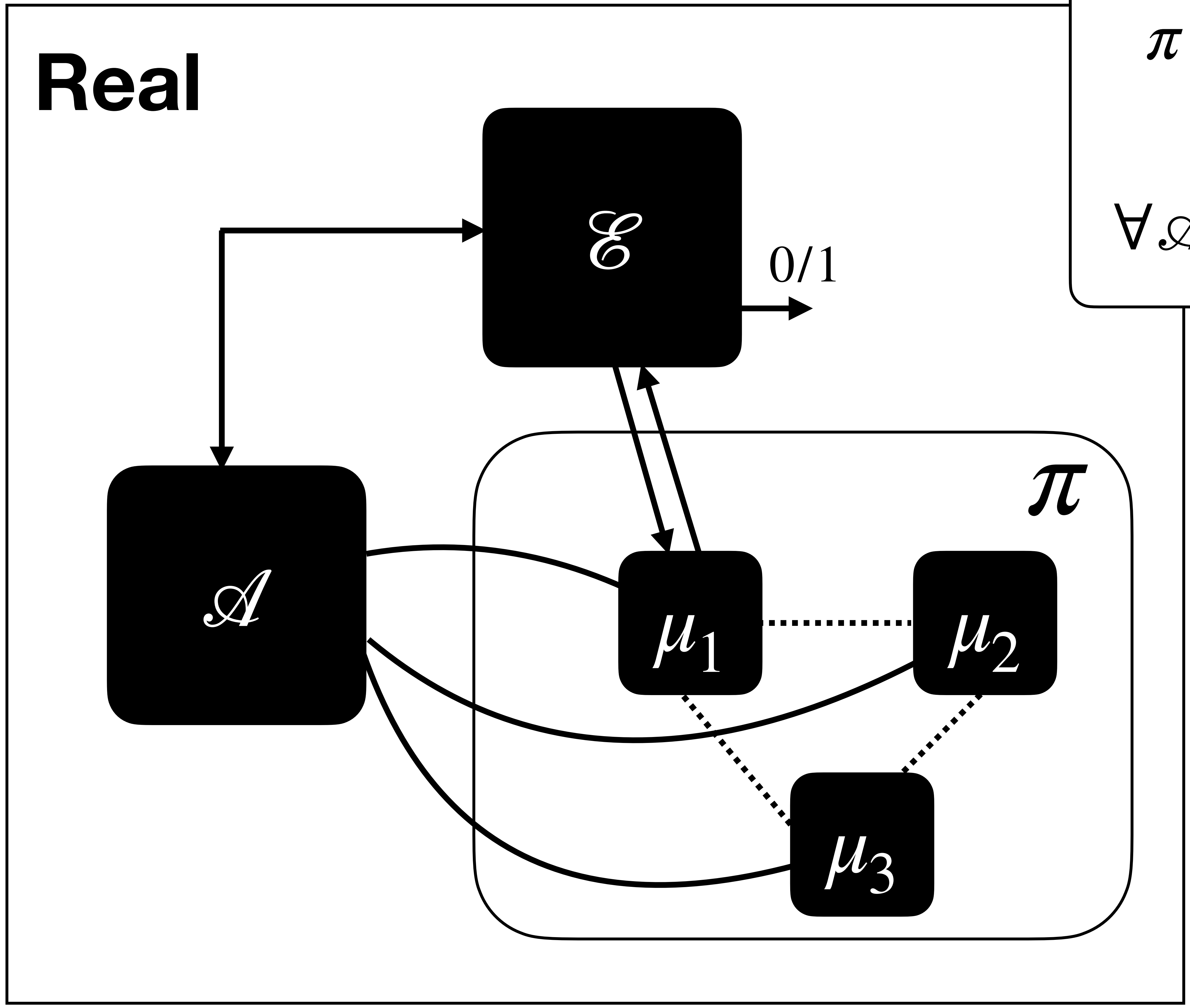
$$\pi \approx_{\text{UC}} \mathscr{F} \iff \forall \mathscr{A}, \exists \mathscr{S}, \forall \mathscr{E}$$

UC Security II

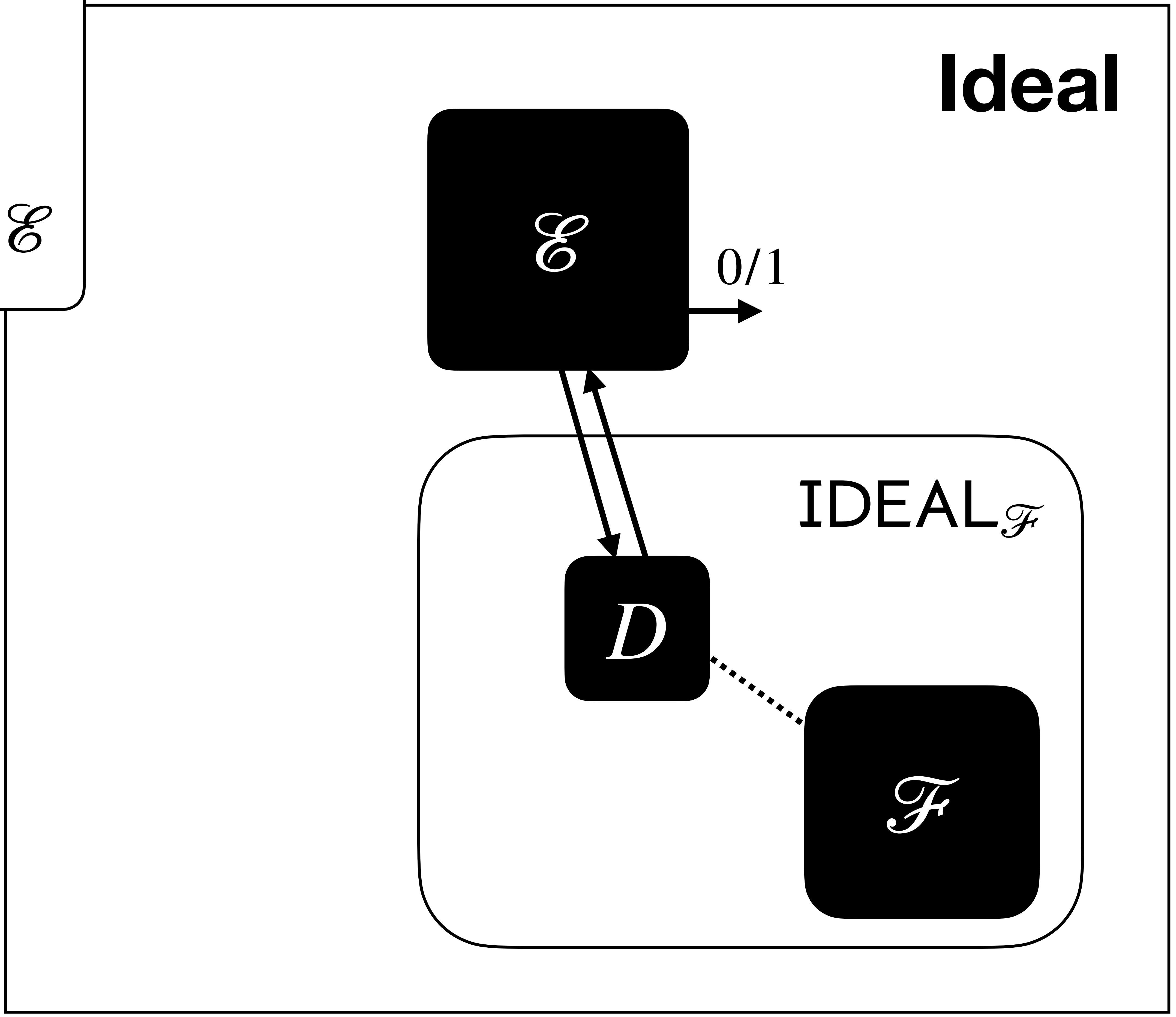
Goal: Cannot distinguish protocol from idealized version.

π	: protocol	$\mathscr{E}$	: environment
$\mathscr{F}$	: ideal functionality	$\mathscr{A}$	: adversary
D	: dummy party	$\mathscr{S}$	: simulator

$$\pi \approx_{\text{UC}} \mathscr{F}$$
$$\Longleftrightarrow$$
$$\forall \mathscr{A}, \exists \mathscr{S}, \forall \mathscr{E}$$



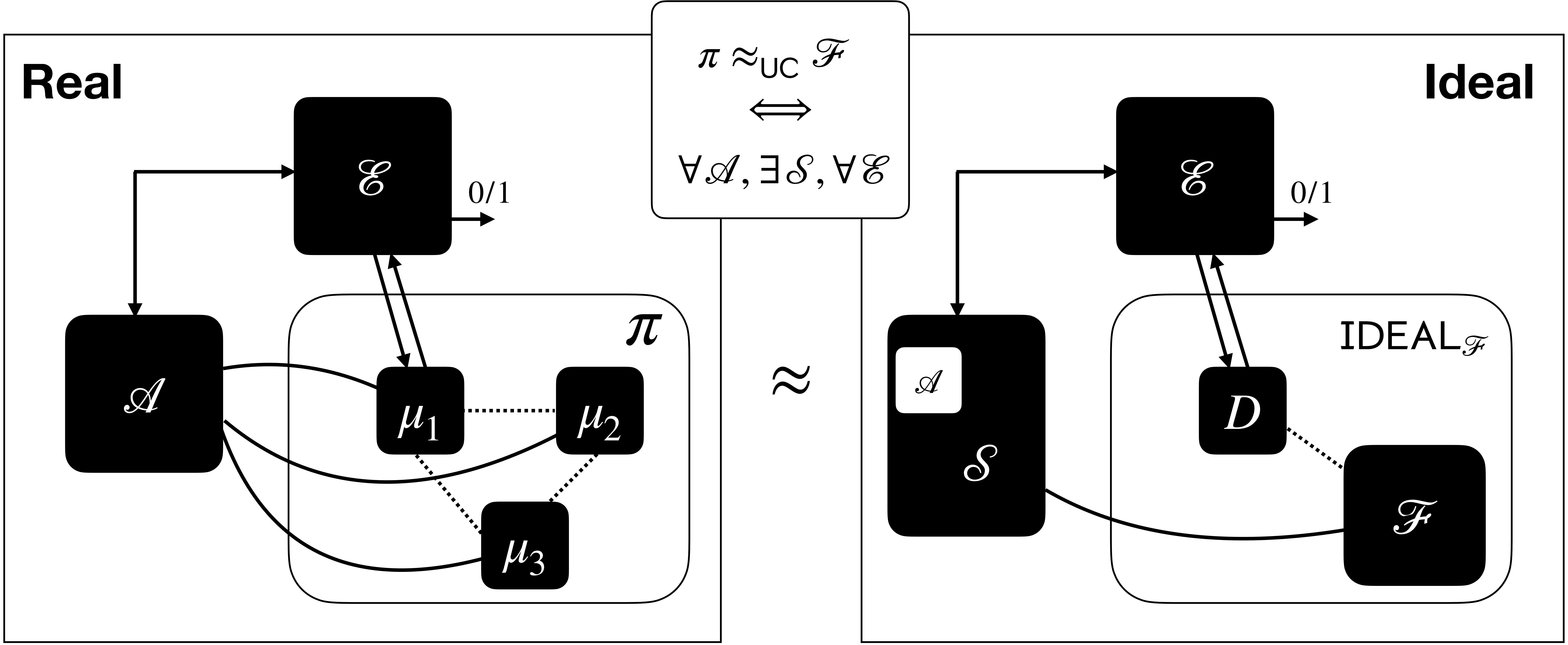
$\approx$



UC Security II

Goal: Cannot distinguish protocol from idealized version.

π	: protocol	$\mathscr{E}$	: environment
$\mathscr{F}$	: ideal functionality	$\mathscr{A}$	: adversary
D	: dummy party	$\mathscr{S}$	: simulator



$$\pi \approx_{\mathsf{UC}} \mathscr{F}$$
$$\Longleftrightarrow$$
$$\forall \mathscr{A}, \exists \mathscr{S}, \forall \mathscr{E}$$

Argument functionality

[LR22]

Argument functionality

[LR22]

- Model a zkSNARK as an ideal functionality.

Argument functionality

[LR22]

- Model a zkSNARK as an ideal functionality.
- Prover generates simulated proofs (without using the witness).

Argument functionality

[LR22]

- Model a zkSNARK as an ideal functionality.
- Prover generates simulated proofs (without using the witness).
- Verifier aims to extract a witness from each accepting proof.

Argument functionality

[LR22]

- Model a zkSNARK as an ideal functionality.
- Prover generates simulated proofs (without using the witness).
- Verifier aims to extract a witness from each accepting proof.
- Proofs generated in Prove are always accepted by Verify.

Argument functionality

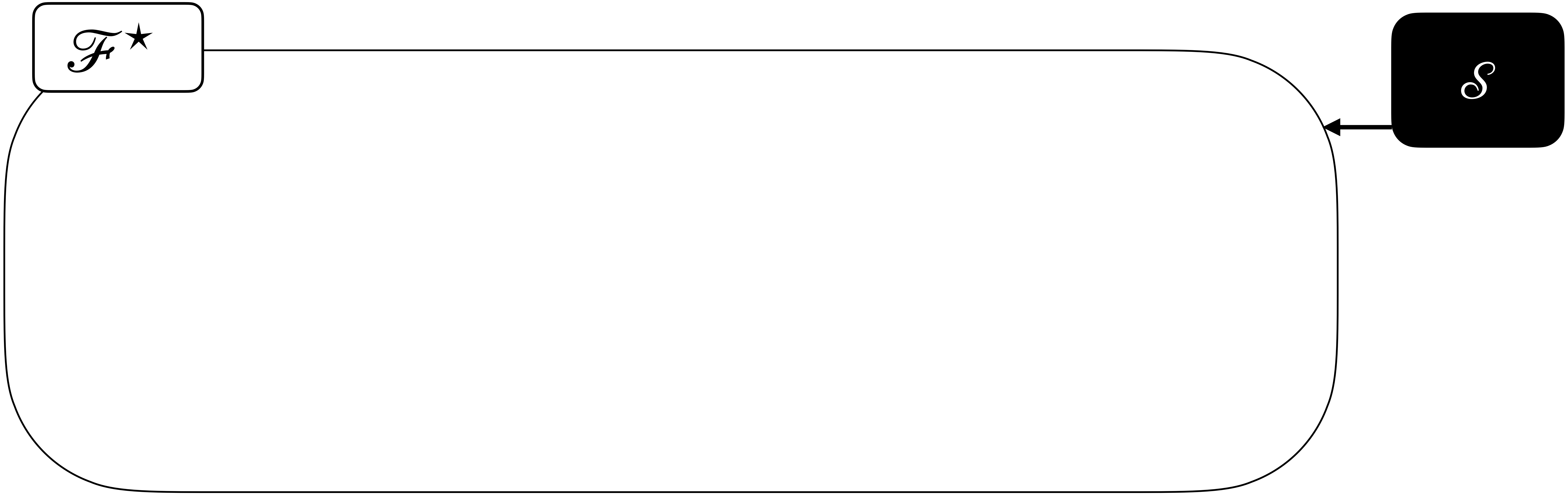
[LR22]

- Model a zkSNARK as an ideal functionality.
- Prover generates simulated proofs (without using the witness).
- Verifier aims to extract a witness from each accepting proof.
- Proofs generated in Prove are always accepted by $\mathtt{Verify}$.

$\mathscr{F}^{\star}$

Argument functionality [LR22]

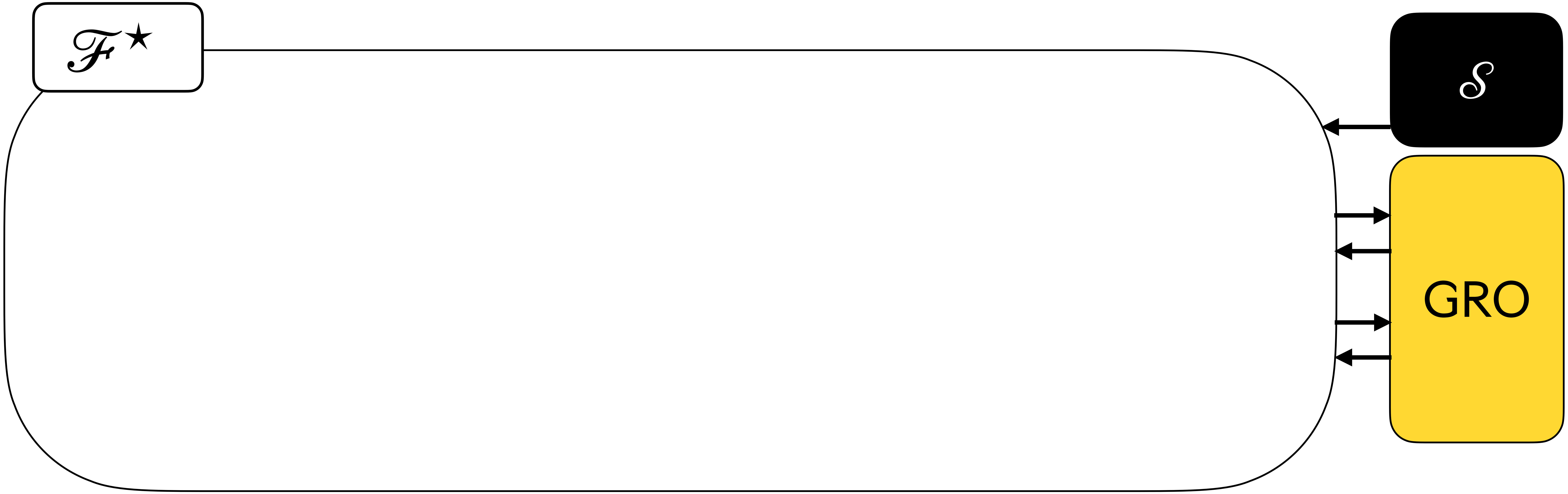
- Model a zkSNARK as an ideal functionality.
- Prover generates simulated proofs (without using the witness).
- Verifier aims to extract a witness from each accepting proof.
- Proofs generated in Prove are always accepted by Verify.



Argument functionality

[LR22]

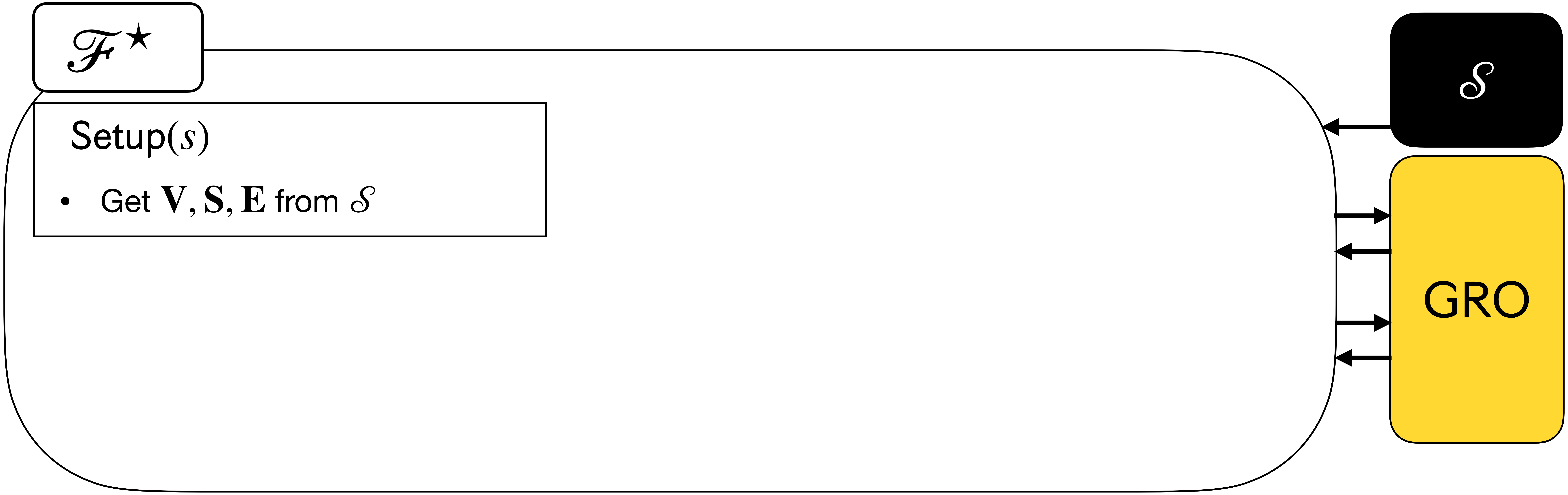
- Model a zkSNARK as an ideal functionality.
- Prover generates simulated proofs (without using the witness).
- Verifier aims to extract a witness from each accepting proof.
- Proofs generated in Prove are always accepted by Verify.



Argument functionality

[LR22]

- Model a zkSNARK as an ideal functionality.
- Prover generates simulated proofs (without using the witness).
- Verifier aims to extract a witness from each accepting proof.
- Proofs generated in Prove are always accepted by Verify.



$\mathscr{F}^\star$

$\mathsf{Setup}(s)$

- Get $\mathbf{V}, \mathbf{S}, \mathbf{E}$ from $\mathscr{S}$

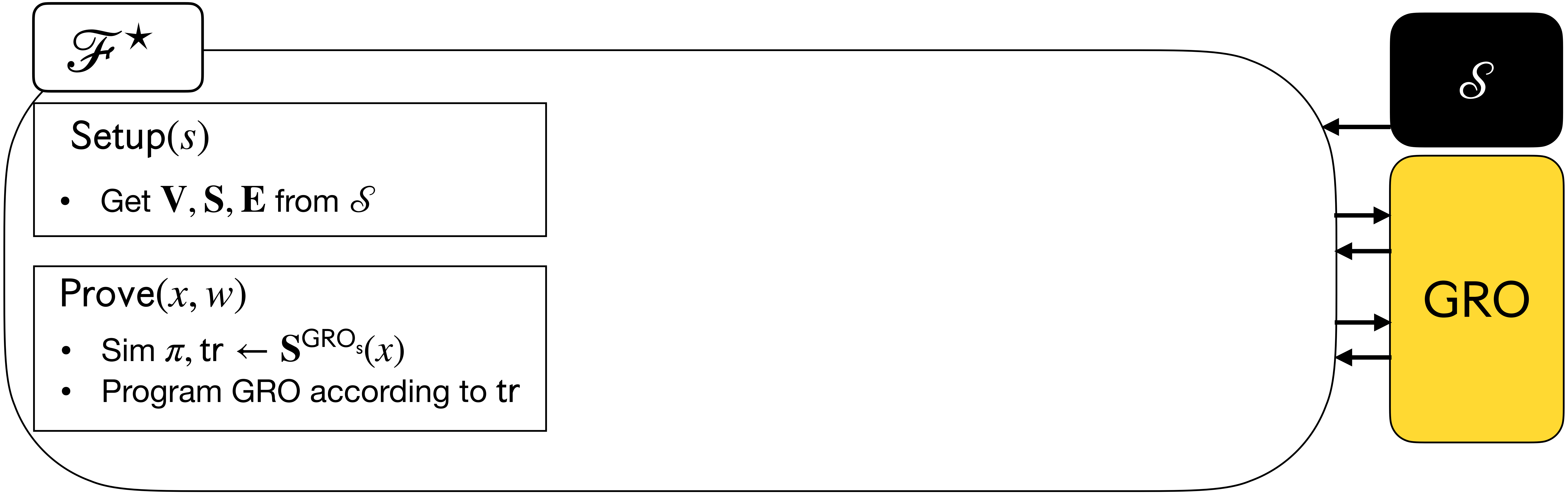
$\mathscr{S}$

GRO

Argument functionality

[LR22]

- Model a zkSNARK as an ideal functionality.
- Prover generates simulated proofs (without using the witness).
- Verifier aims to extract a witness from each accepting proof.
- Proofs generated in Prove are always accepted by Verify.



$\mathscr{F}^\star$

Setup(s)

- Get $\mathbf{V}, \mathbf{S}, \mathbf{E}$ from $\mathscr{S}$

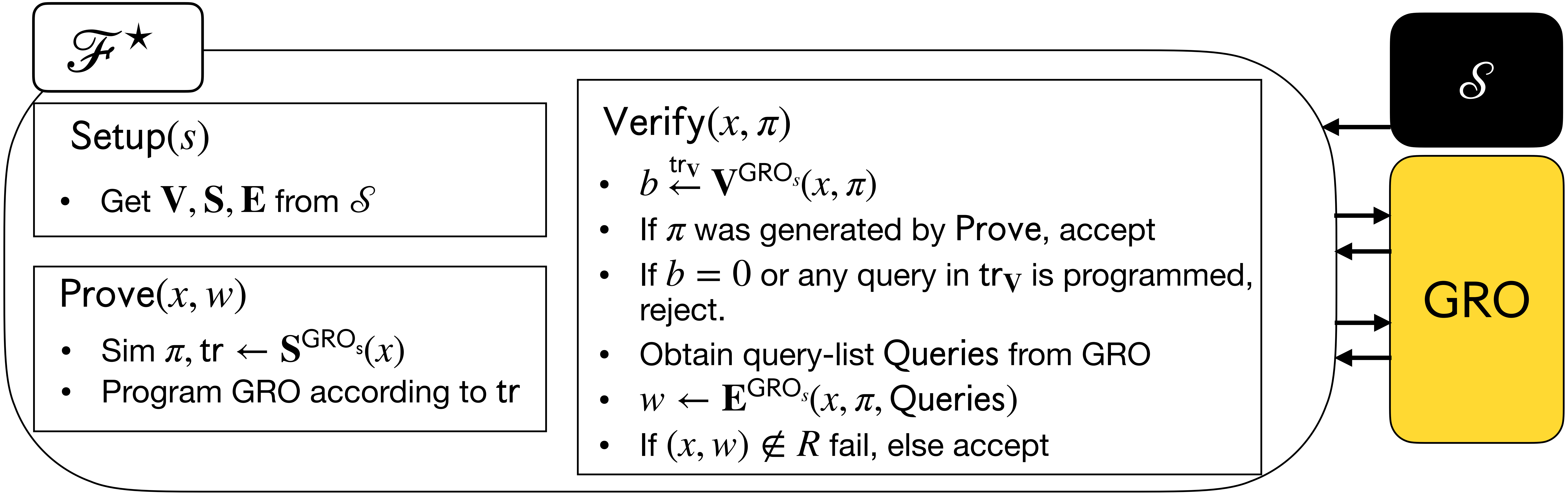
Prove(x, w)

- Sim π, tr $\leftarrow \mathbf{S}^{\mathsf{GRO}_s}(x)$
- Program GRO according to tr

Argument functionality

[LR22]

- Model a zkSNARK as an ideal functionality.
- Prover generates simulated proofs (without using the witness).
- Verifier aims to extract a witness from each accepting proof.
- Proofs generated in Prove are always accepted by Verify.



$\mathscr{F}^\star$

Setup(s)

- Get $\mathbf{V}, \mathbf{S}, \mathbf{E}$ from $\mathscr{S}$

Prove(x, w)

- Sim π, tr $\leftarrow \mathbf{S}^{\mathsf{GRO}_s}(x)$
- Program GRO according to tr

Verify(x, π)

- $b \overset{\mathsf{tr}_\mathbf{V}}{\leftarrow} \mathbf{V}^{\mathsf{GRO}_s}(x, \pi)$
- If π was generated by Prove, accept
- If $b = 0$ or any query in tr$_\mathbf{V}$ is programmed, reject.
- Obtain query-list Queries from GRO
- $w \leftarrow \mathbf{E}^{\mathsf{GRO}_s}(x, \pi, \text{Queries})$
- If $(x, w) \notin R$ fail, else accept

Wrapper protocol

Wrapper protocol

Converts an argument $\mathsf{ARG} = (\mathbf{P}, \mathbf{V})$ in the ROM into a protocol in the GROM

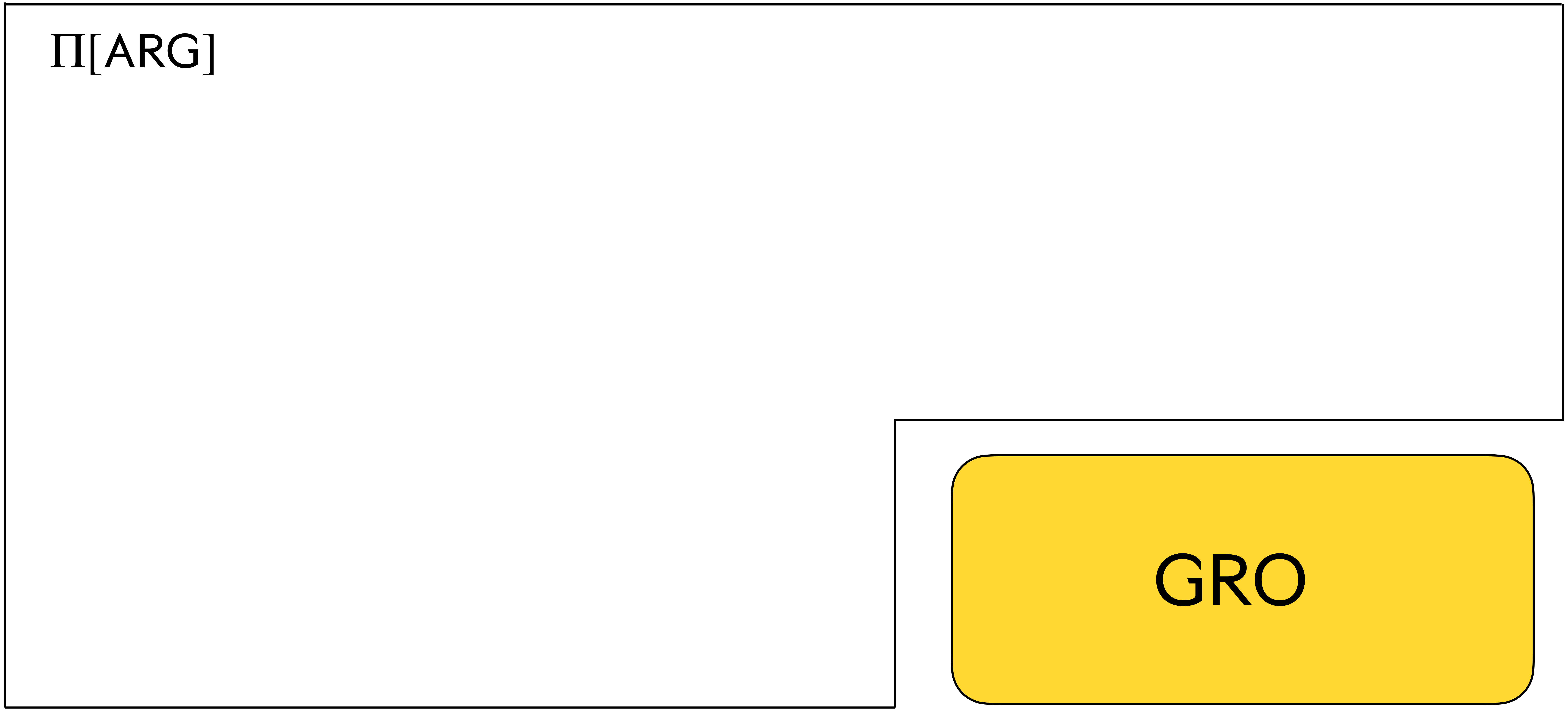
Wrapper protocol

Converts an argument $\mathsf{ARG} = (\mathbf{P}, \mathbf{V})$ in the ROM into a protocol in the GROM

$\Pi[\mathsf{ARG}]$

Wrapper protocol

Converts an argument $\mathrm{ARG} = (\mathbf{P}, \mathbf{V})$ in the ROM into a protocol in the GROM

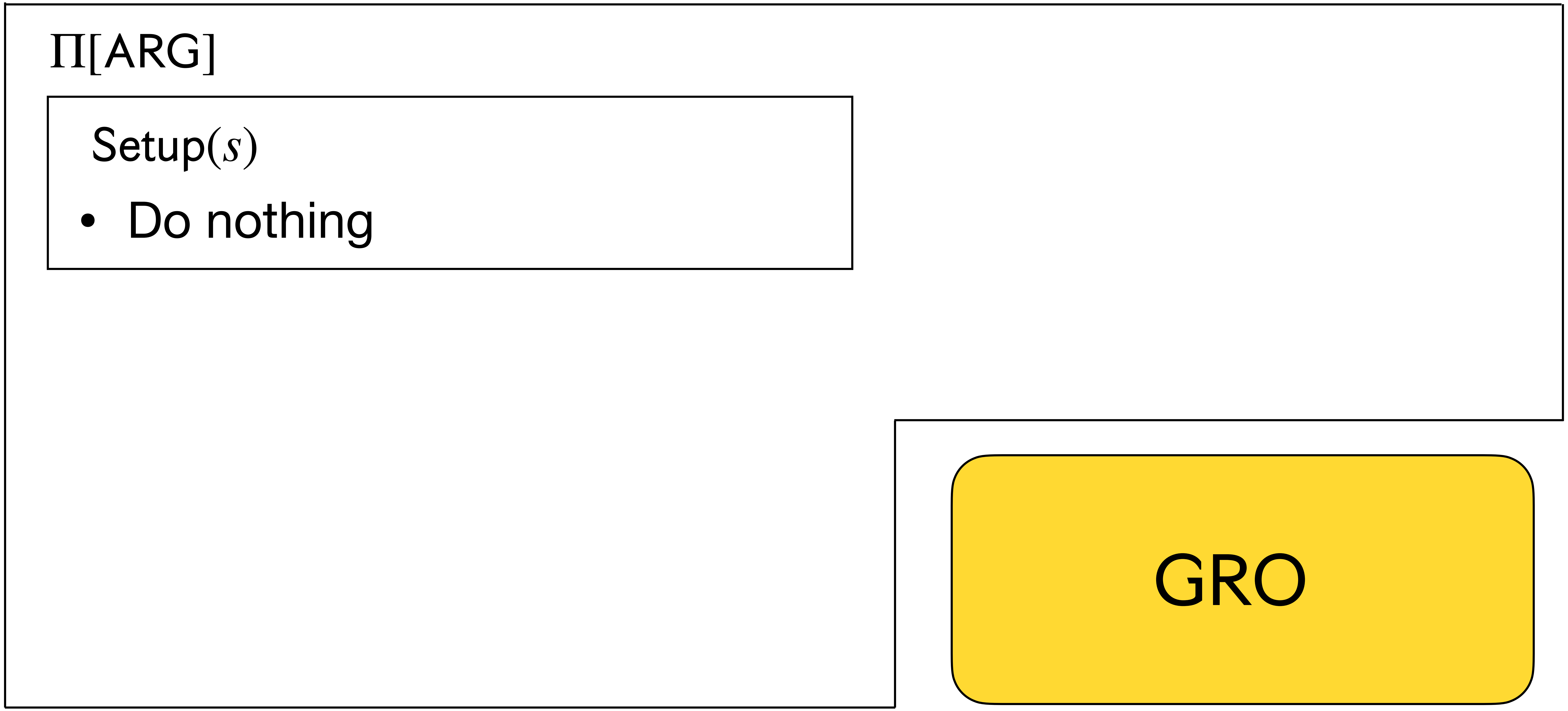


$\Pi[\mathrm{ARG}]$

GRO

Wrapper protocol

Converts an argument $\mathrm{ARG} = (\mathbf{P}, \mathbf{V})$ in the ROM into a protocol in the GROM



$\Pi[\mathrm{ARG}]$

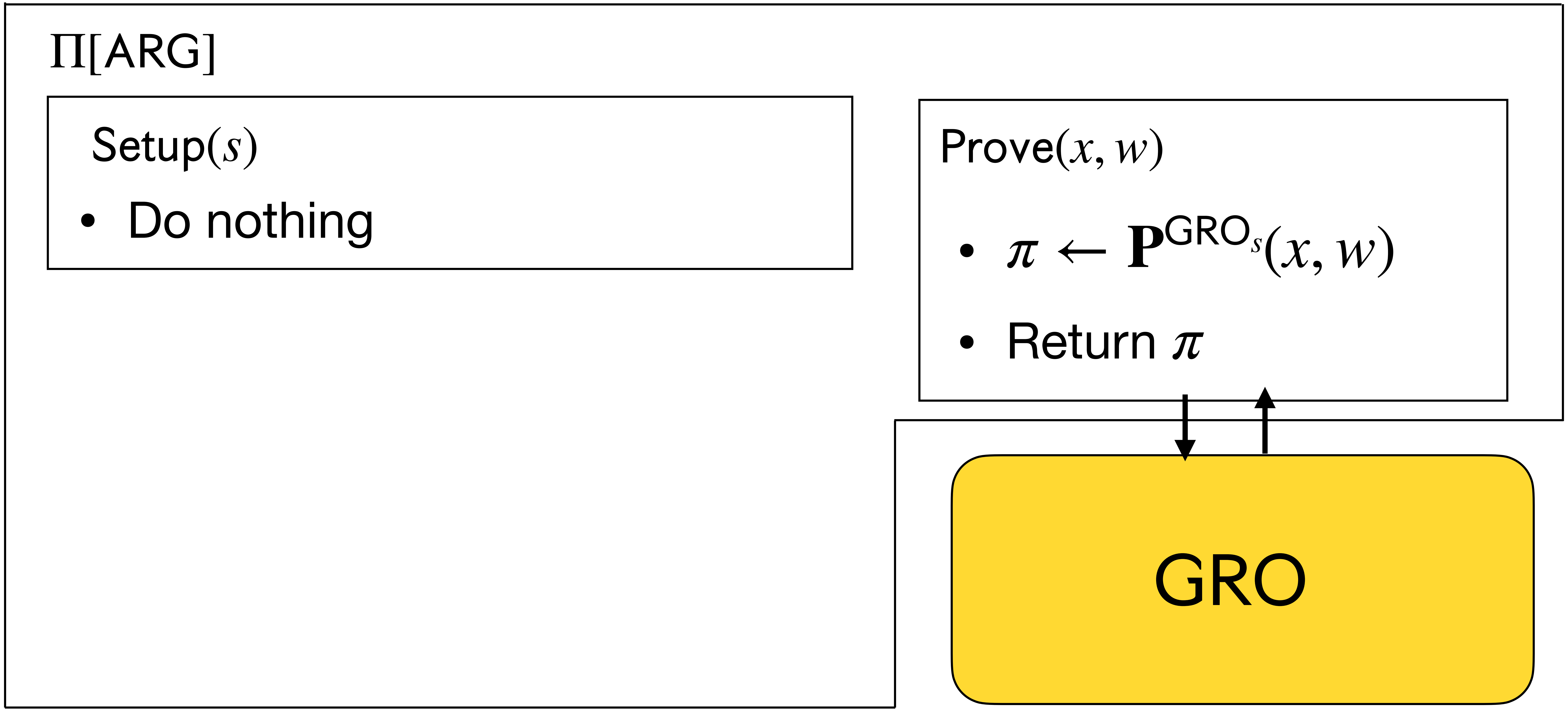
Setup(s)

- Do nothing

GRO

Wrapper protocol

Converts an argument $\mathsf{ARG} = (\mathbf{P}, \mathbf{V})$ in the ROM into a protocol in the GROM



$\Pi[\mathsf{ARG}]$

Setup(s)

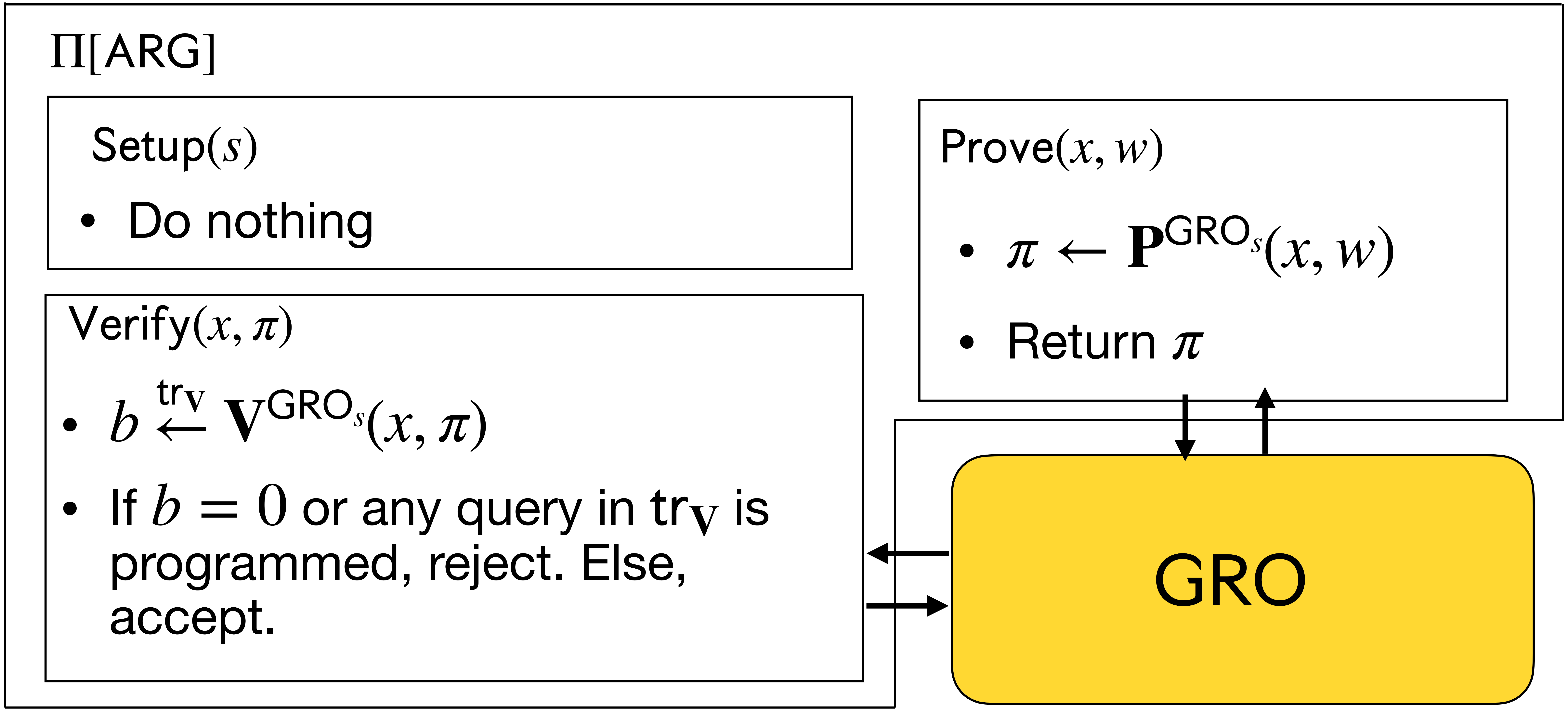
- Do nothing

Prove(x, w)

- $\pi \leftarrow \mathbf{P}^{\mathsf{GRO}_s}(x, w)$
- Return π

Wrapper protocol

Converts an argument ARG $= (\mathbf{P}, \mathbf{V})$ in the ROM into a protocol in the GROM



Π[ARG]

Setup(s)

- Do nothing

Prove(x, w)

- $\pi \leftarrow \mathbf{P}^{\mathsf{GRO}_s}(x, w)$
- Return π

Verify(x, π)

- $b \overset{\mathsf{tr}_{\mathbf{V}}}{\leftarrow} \mathbf{V}^{\mathsf{GRO}_s}(x, \pi)$
- If $b = 0$ or any query in $\mathsf{tr}_{\mathbf{V}}$ is programmed, reject. Else, accept.

Recap and Goal

Recap and Goal

Find an ARG in the ROM such that

Recap and Goal

Find an ARG in the ROM such that

in the GRO

Recap and Goal

Find an ARG in the ROM such that

$\Pi[\text{ARG}]$

in the

GRO

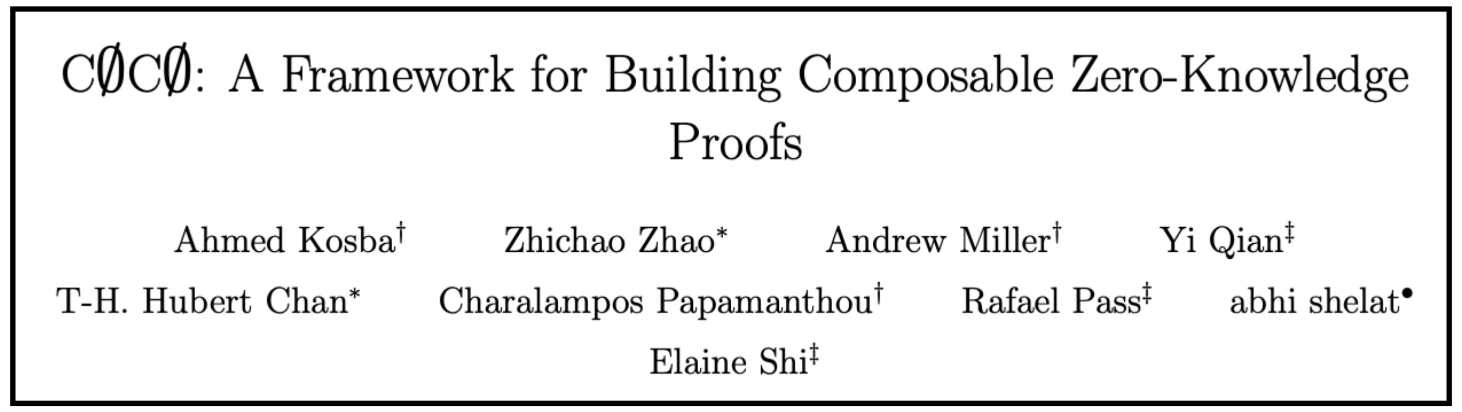
Recap and Goal

Find an ARG in the ROM such that

$\Pi[\mathsf{ARG}]$ $\approx_{\mathrm{UC}}$ $\mathcal{F}^{\star}$ in the GRO

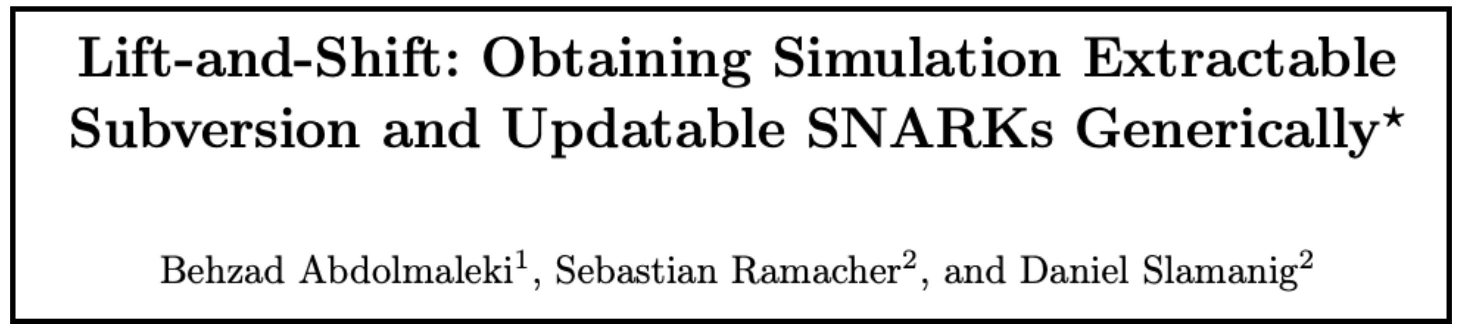
Related works

Related works



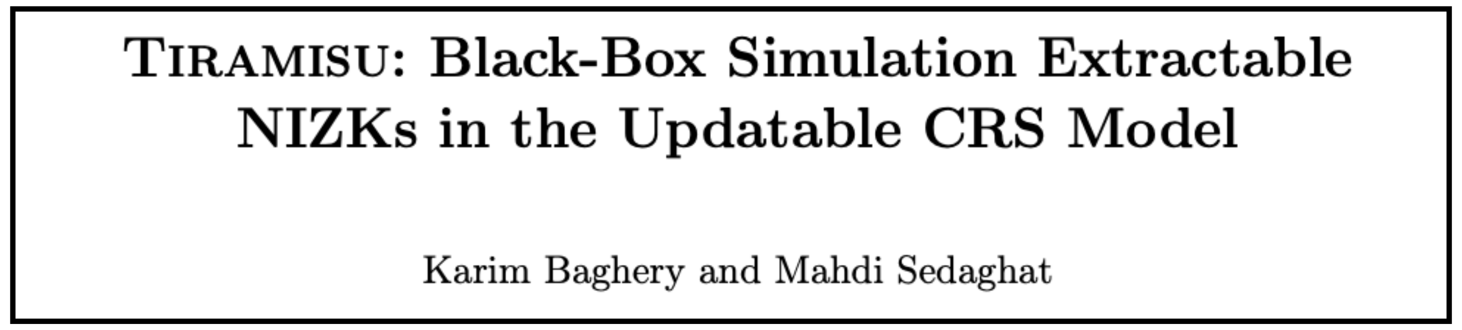
CØCØ: A Framework for Building Composable Zero-Knowledge Proofs

Ahmed Kosba[†] Zhichao Zhao[*] Andrew Miller[†] Yi Qian[‡]
T-H. Hubert Chan[*] Charalampos Papamanthou[†] Rafael Pass[‡] abhi shelat[•]
Elaine Shi[‡]



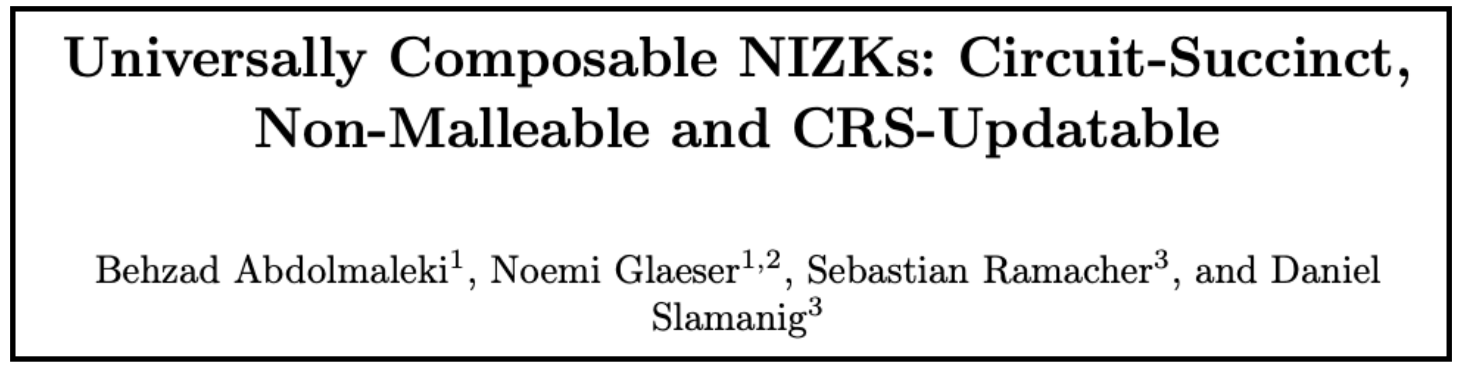
Lift-and-Shift: Obtaining Simulation Extractable Subversion and Updatable SNARKs Generically[⋆]

Behzad Abdolmaleki[1], Sebastian Ramacher[2], and Daniel Slamanig[2]



Tiramisu: Black-Box Simulation Extractable NIZKs in the Updatable CRS Model

Karim Baghery and Mahdi Sedaghat



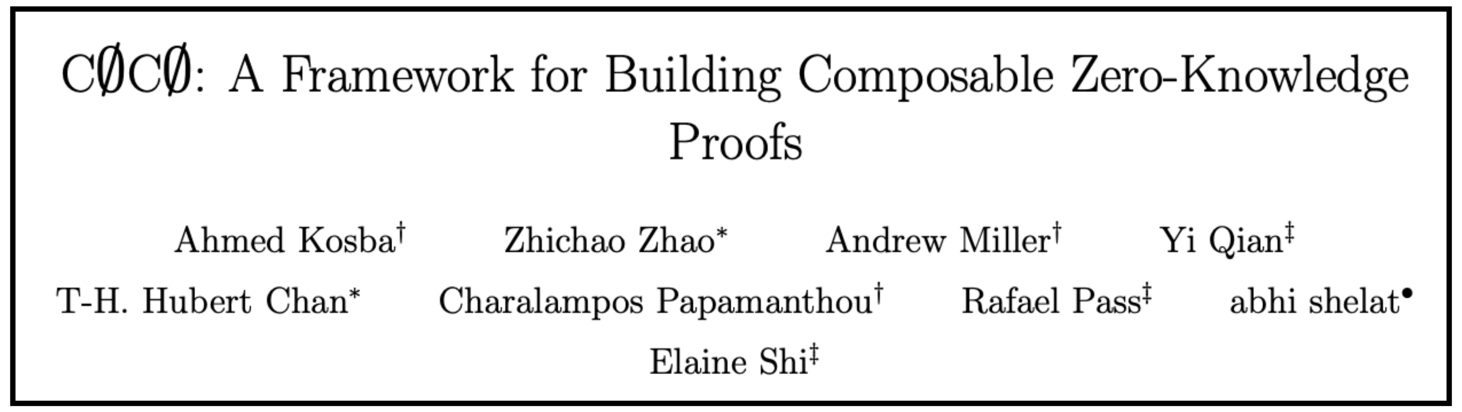
Universally Composable NIZKs: Circuit-Succinct, Non-Malleable and CRS-Updatable

Behzad Abdolmaleki[1], Noemi Glaeser[1,2], Sebastian Ramacher[3], and Daniel Slamanig[3]

Related works

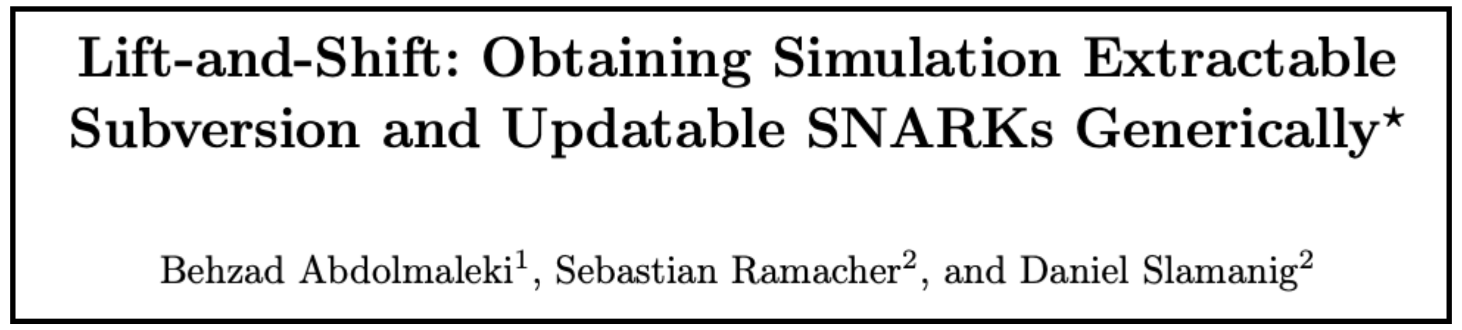
Append an encryption of the witness to the proof.

- Cannot be succinct $|\pi| \geq |w|$



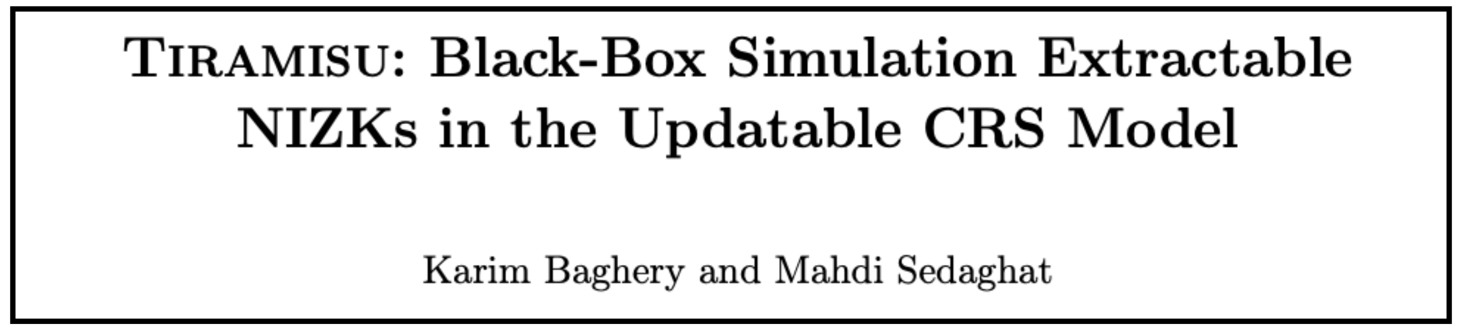
CØCØ: A Framework for Building Composable Zero-Knowledge Proofs

Ahmed Kosba[†] Zhichao Zhao[*] Andrew Miller[†] Yi Qian[‡]
T-H. Hubert Chan[*] Charalampos Papamanthou[†] Rafael Pass[‡] abhi shelat[•]
Elaine Shi[‡]



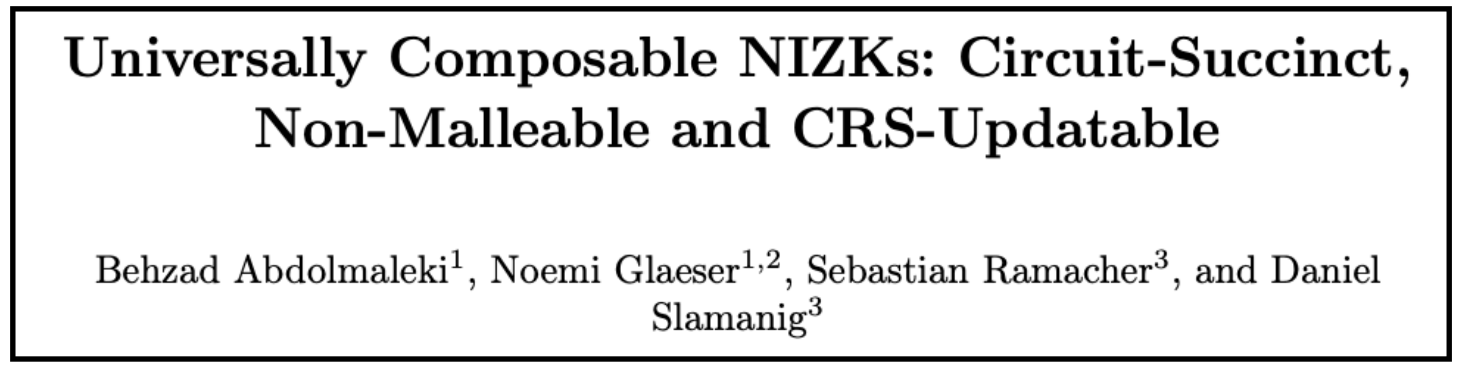
Lift-and-Shift: Obtaining Simulation Extractable Subversion and Updatable SNARKs Generically[⋆]

Behzad Abdolmaleki[1], Sebastian Ramacher[2], and Daniel Slamanig[2]



Tiramisu: Black-Box Simulation Extractable NIZKs in the Updatable CRS Model

Karim Baghery and Mahdi Sedaghat



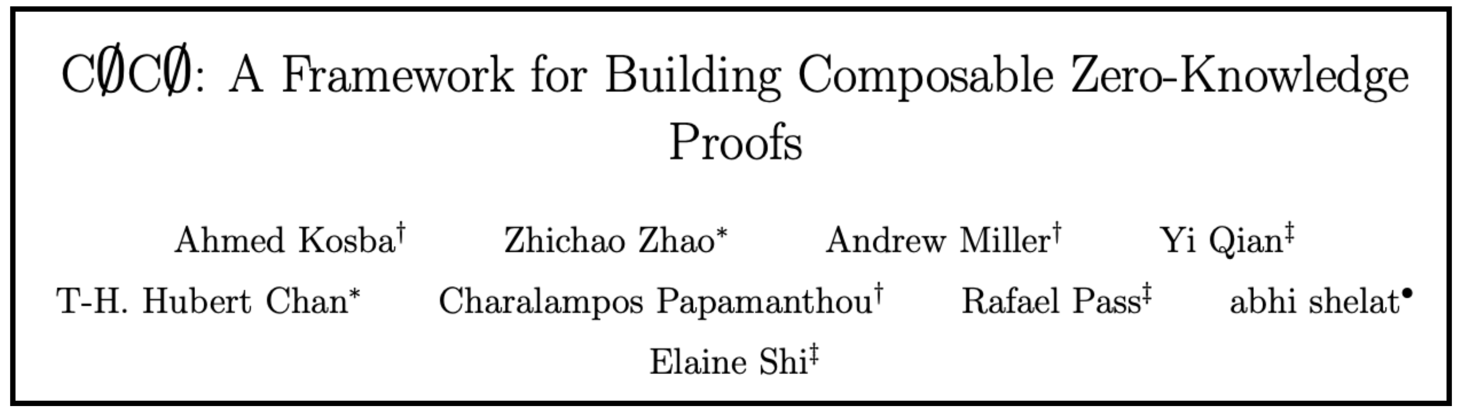
Universally Composable NIZKs: Circuit-Succinct, Non-Malleable and CRS-Updatable

Behzad Abdolmaleki[1], Noemi Glaeser[1,2], Sebastian Ramacher[3], and Daniel Slamanig[3]

Related works

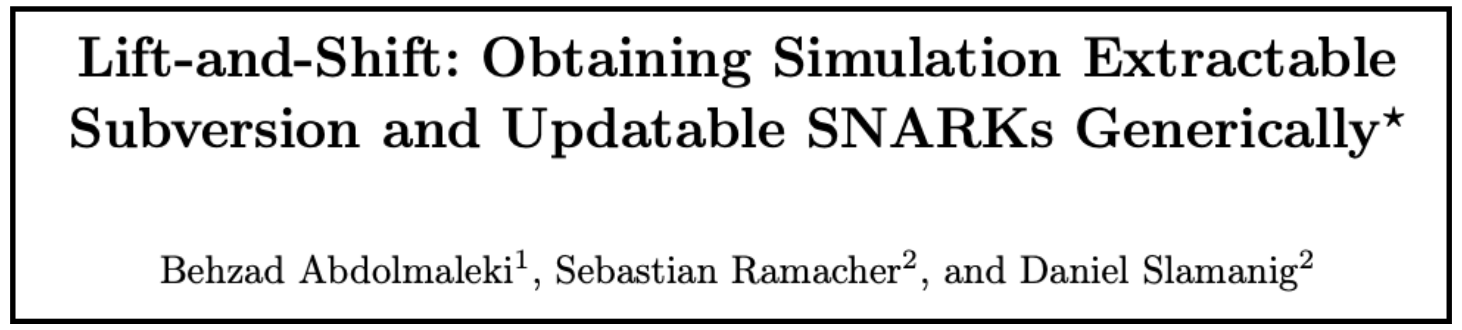
Append an encryption of the witness to the proof.

- Cannot be succinct $|\pi| \geq |w|$



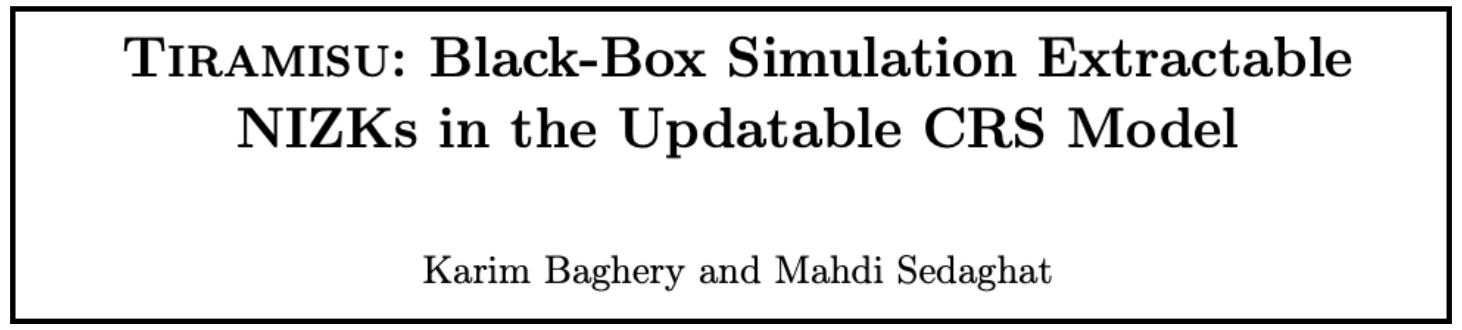
CØCØ: A Framework for Building Composable Zero-Knowledge Proofs

Ahmed Kosba[†] Zhichao Zhao[*] Andrew Miller[†] Yi Qian[‡]
T-H. Hubert Chan[*] Charalampos Papamanthou[†] Rafael Pass[‡] abhi shelat[•]
Elaine Shi[‡]



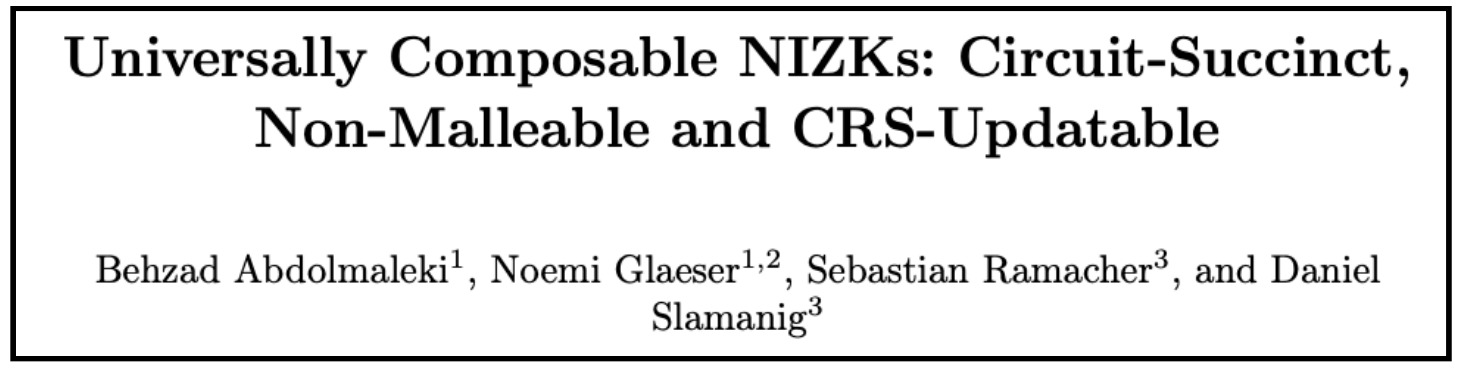
Lift-and-Shift: Obtaining Simulation Extractable Subversion and Updatable SNARKs Generically[*]

Behzad Abdolmaleki[1], Sebastian Ramacher[2], and Daniel Slamanig[2]



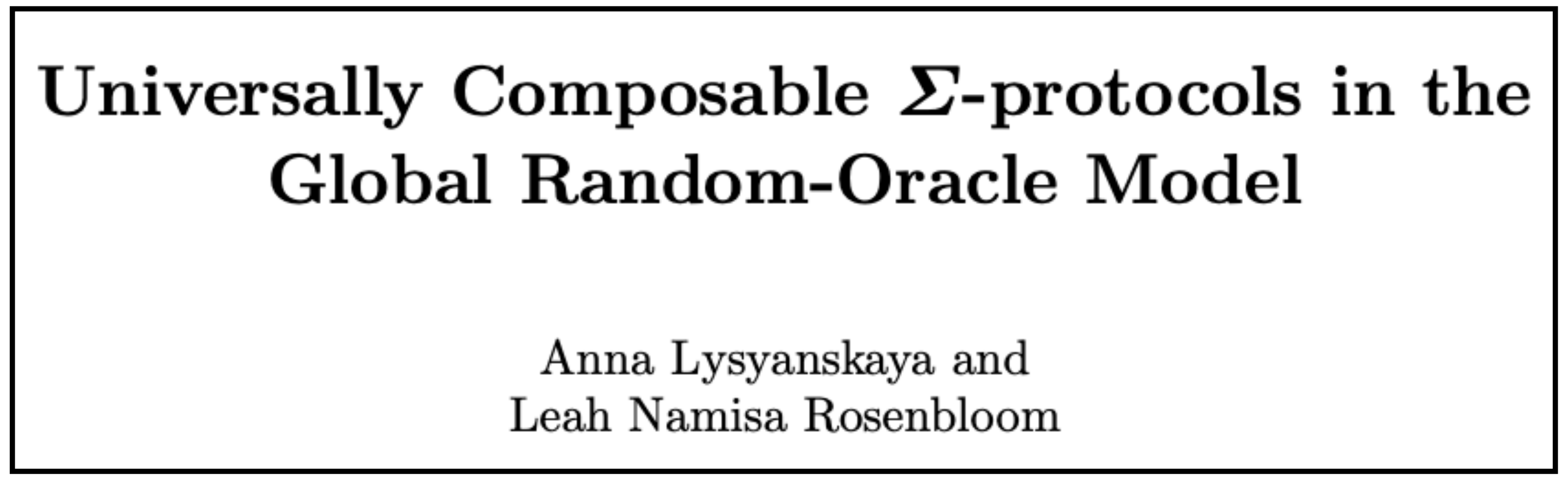
Tiramisu: Black-Box Simulation Extractable NIZKs in the Updatable CRS Model

Karim Baghery and Mahdi Sedaghat



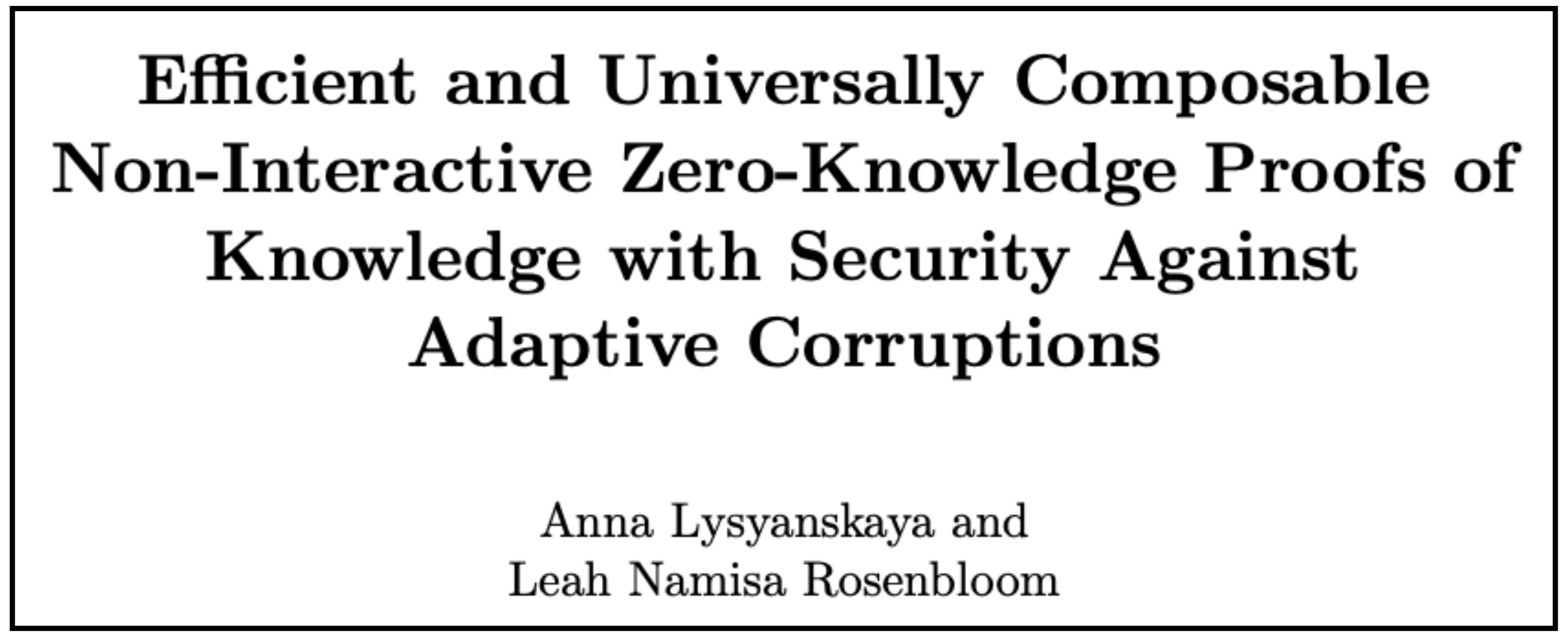
Universally Composable NIZKs: Circuit-Succinct, Non-Malleable and CRS-Updatable

Behzad Abdolmaleki[1], Noemi Glaeser[1,2], Sebastian Ramacher[3], and Daniel Slamanig[3]



Universally Composable Σ-protocols in the Global Random-Oracle Model

Anna Lysyanskaya and
Leah Namisa Rosenbloom



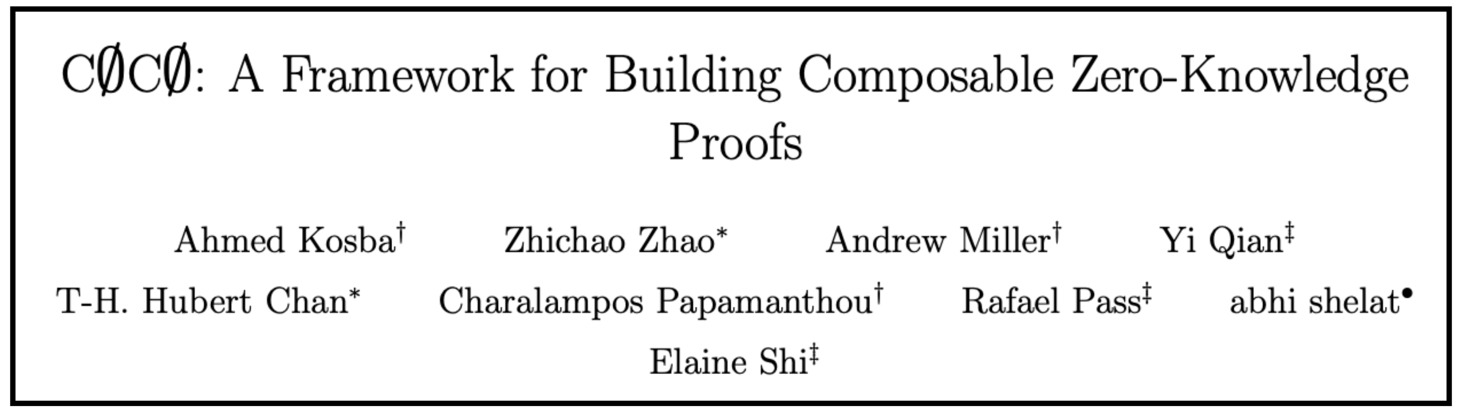
Efficient and Universally Composable Non-Interactive Zero-Knowledge Proofs of Knowledge with Security Against Adaptive Corruptions

Anna Lysyanskaya and
Leah Namisa Rosenbloom

Related works

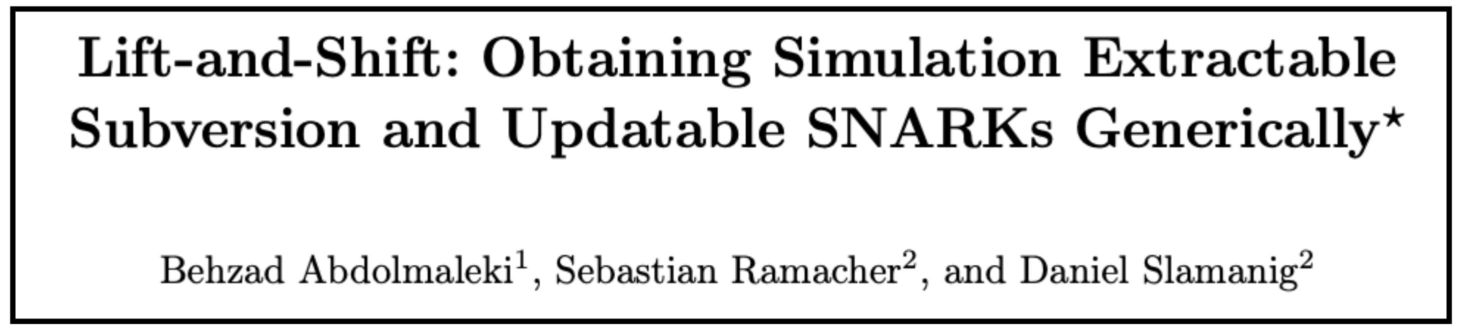
Append an encryption of the witness to the proof.

- Cannot be succinct $|\pi| \geq |w|$



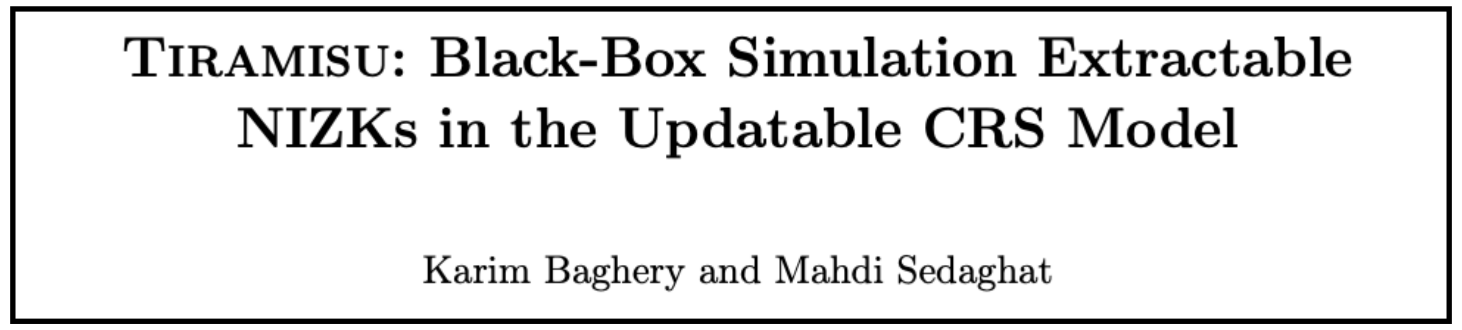
C∅C∅: A Framework for Building Composable Zero-Knowledge Proofs

Ahmed Kosba[†] Zhichao Zhao[*] Andrew Miller[†] Yi Qian[‡]
T-H. Hubert Chan[*] Charalampos Papamanthou[†] Rafael Pass[‡] abhi shelat[•]
Elaine Shi[‡]



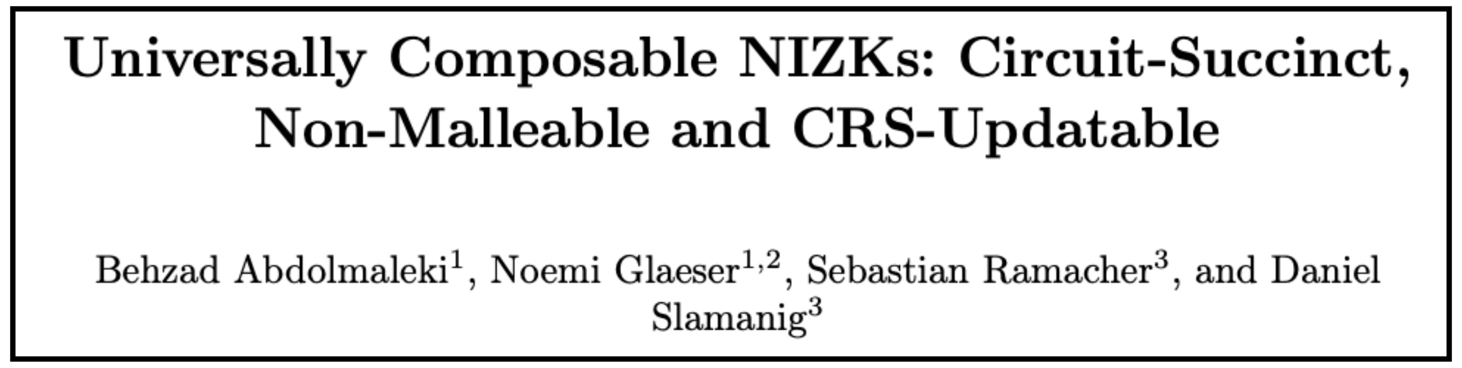
Lift-and-Shift: Obtaining Simulation Extractable Subversion and Updatable SNARKs Generically[⋆]

Behzad Abdolmaleki[1], Sebastian Ramacher[2], and Daniel Slamanig[2]



Tiramisu: Black-Box Simulation Extractable NIZKs in the Updatable CRS Model

Karim Baghery and Mahdi Sedaghat



Universally Composable NIZKs: Circuit-Succinct, Non-Malleable and CRS-Updatable

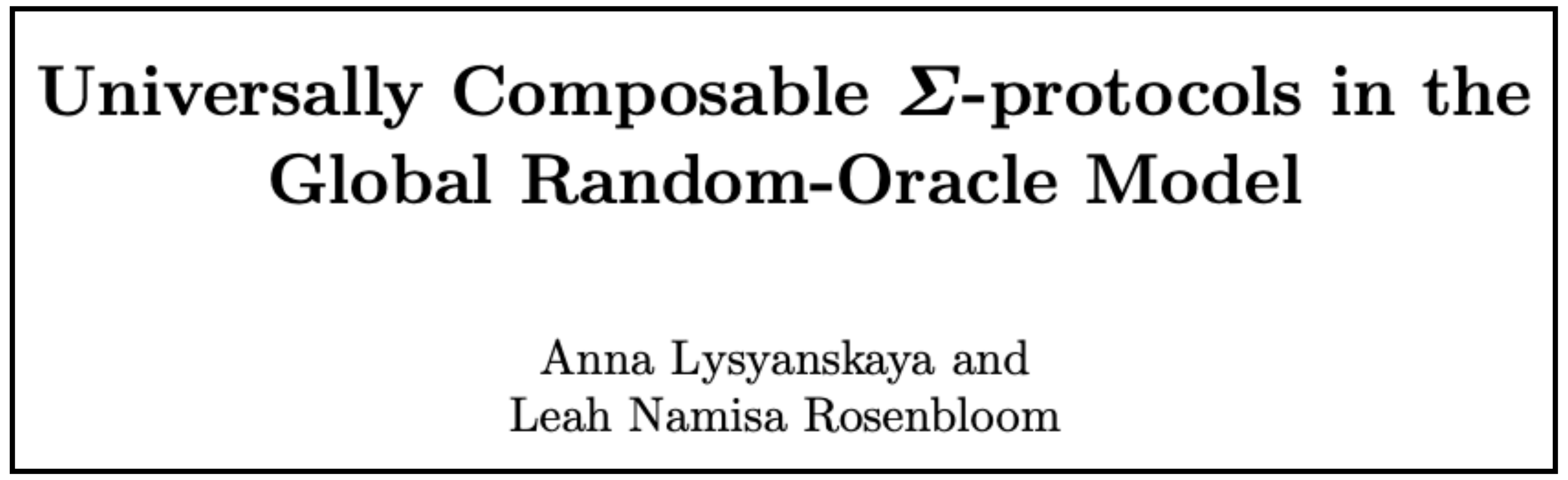
Behzad Abdolmaleki[1], Noemi Glaeser[1,2], Sebastian Ramacher[3], and Daniel Slamanig[3]

Compile Σ-protocol into NIZK

+ Techniques inspired this work

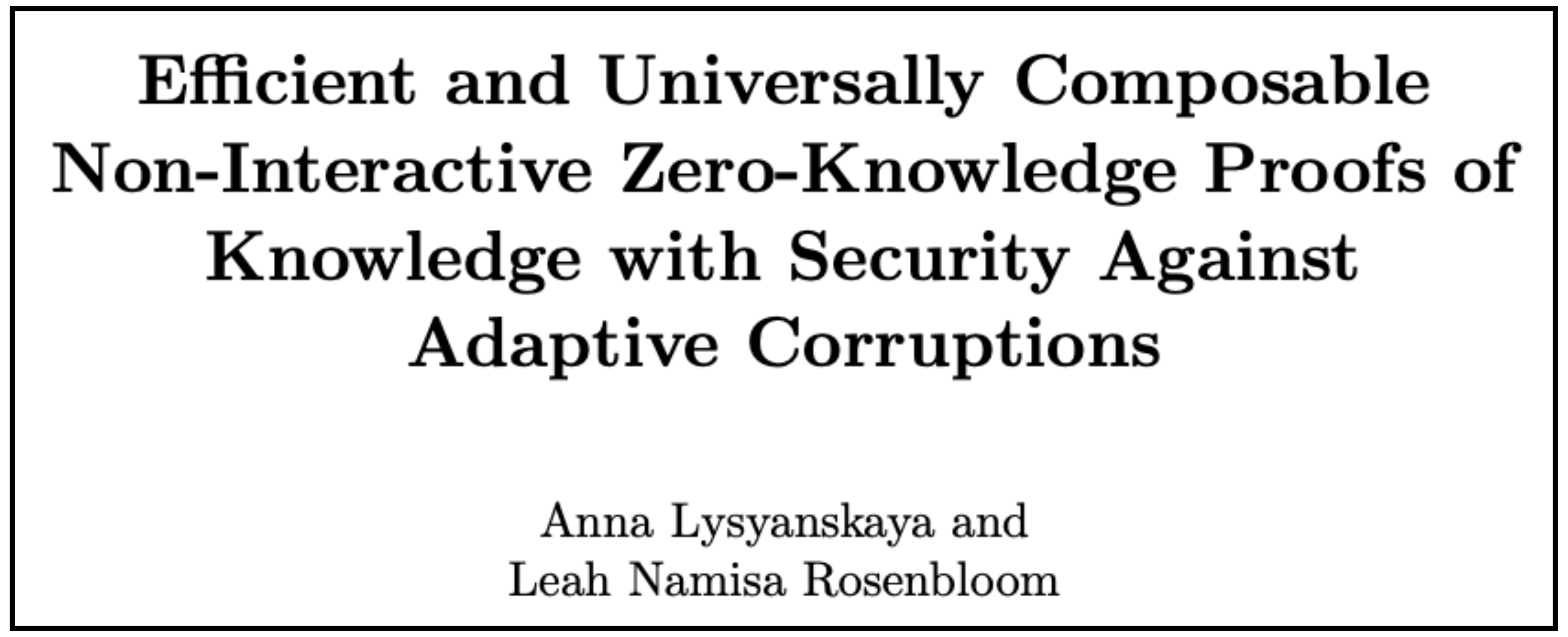
- Not succinct

- Expensive compilation (non-FS)



Universally Composable Σ-protocols in the Global Random-Oracle Model

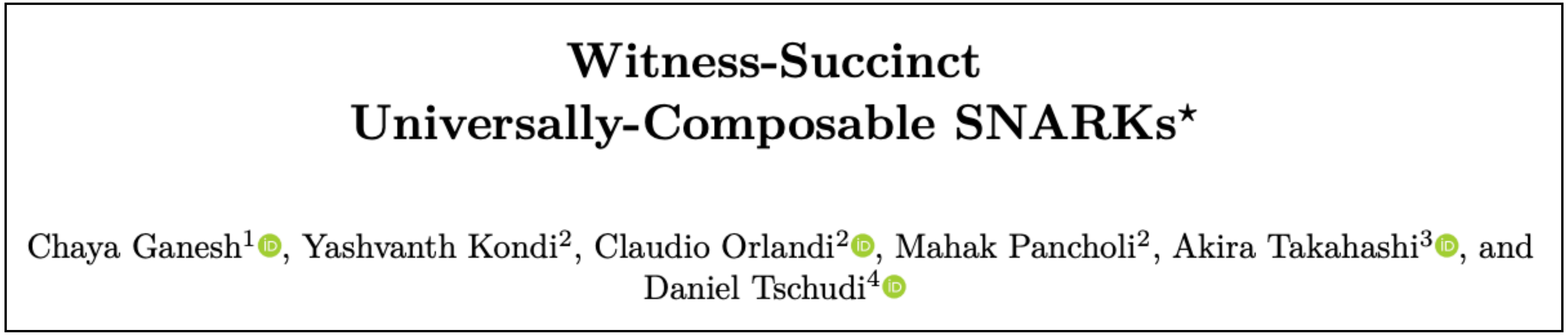
Anna Lysyanskaya and
Leah Namisa Rosenbloom



Efficient and Universally Composable Non-Interactive Zero-Knowledge Proofs of Knowledge with Security Against Adaptive Corruptions

Anna Lysyanskaya and
Leah Namisa Rosenbloom

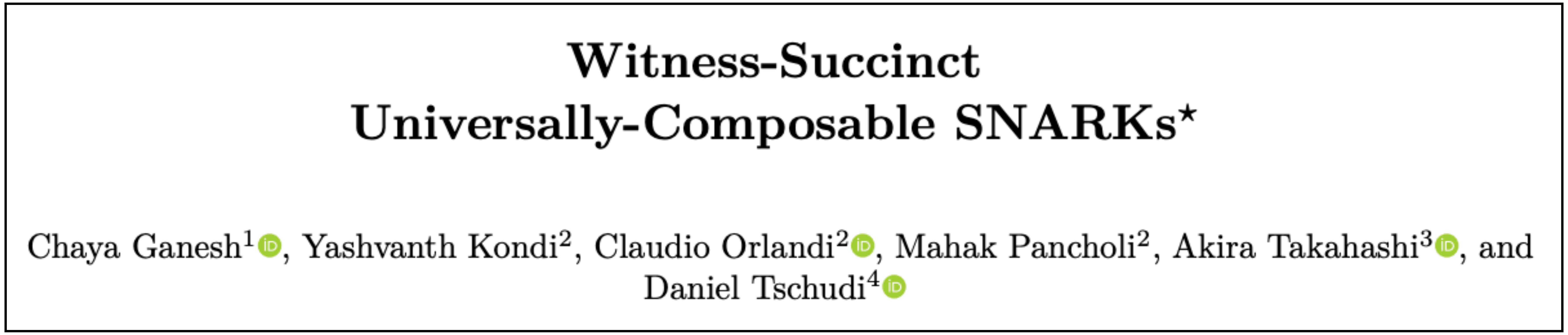
Succinct UC-secure zkSNARKs



Witness-Succinct Universally-Composable SNARKs[*]

Chaya Ganesh[1], Yashvanth Kondi[2], Claudio Orlandi[2], Mahak Pancholi[2], Akira Takahashi[3], and Daniel Tschudi[4]

Succinct UC-secure zkSNARKs

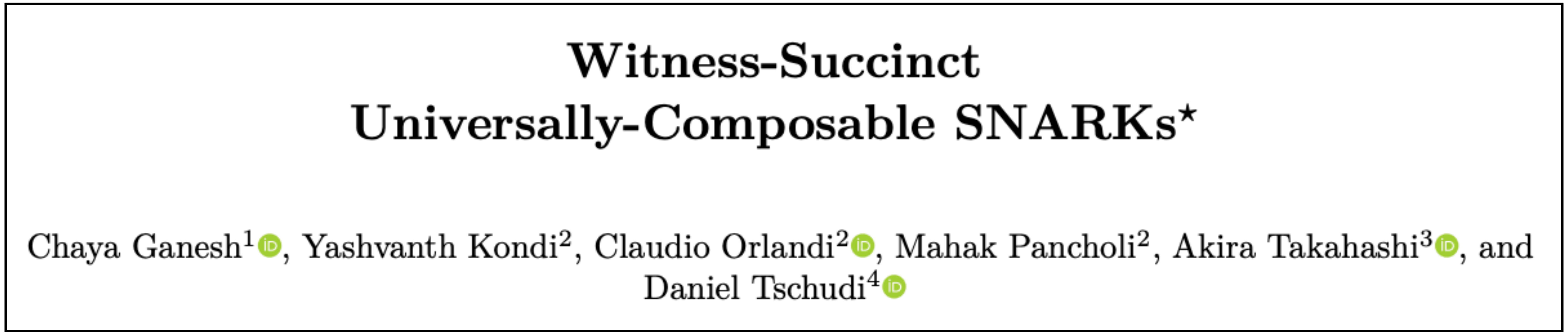


Witness-Succinct Universally-Composable SNARKs*

Chaya Ganesh[1], Yashvanth Kondi[2], Claudio Orlandi[2], Mahak Pancholi[2], Akira Takahashi[3], and Daniel Tschudi[4]

First UC-secure **S**NARK

Succinct UC-secure zkSNARKs



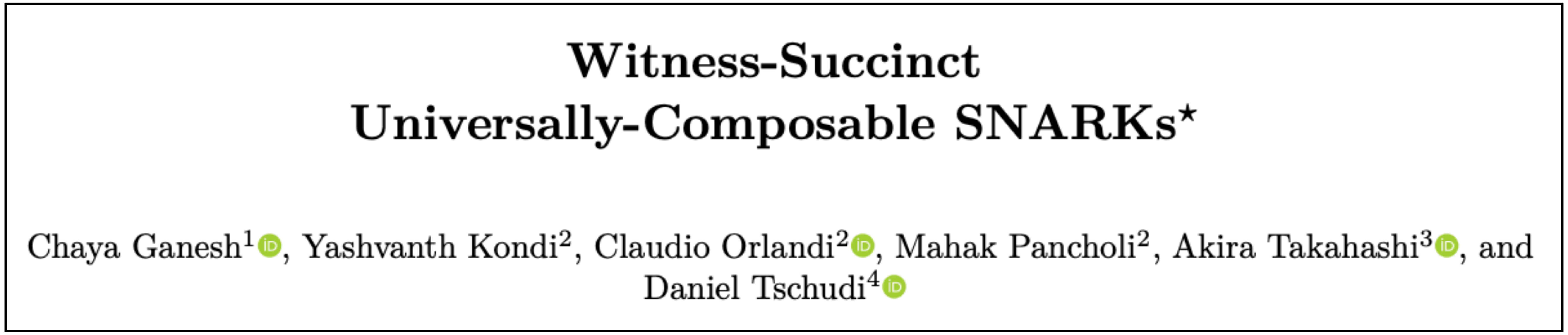
Witness-Succinct
Universally-Composable SNARKs*

Chaya Ganesh[1], Yashvanth Kondi[2], Claudio Orlandi[2], Mahak Pancholi[2], Akira Takahashi[3], and Daniel Tschudi[4]

First UC-secure **S**NARK

Combines simulation-extractable zkSNARK with a PCS

Succinct UC-secure zkSNARKs



Witness-Succinct Universally-Composable SNARKs⋆

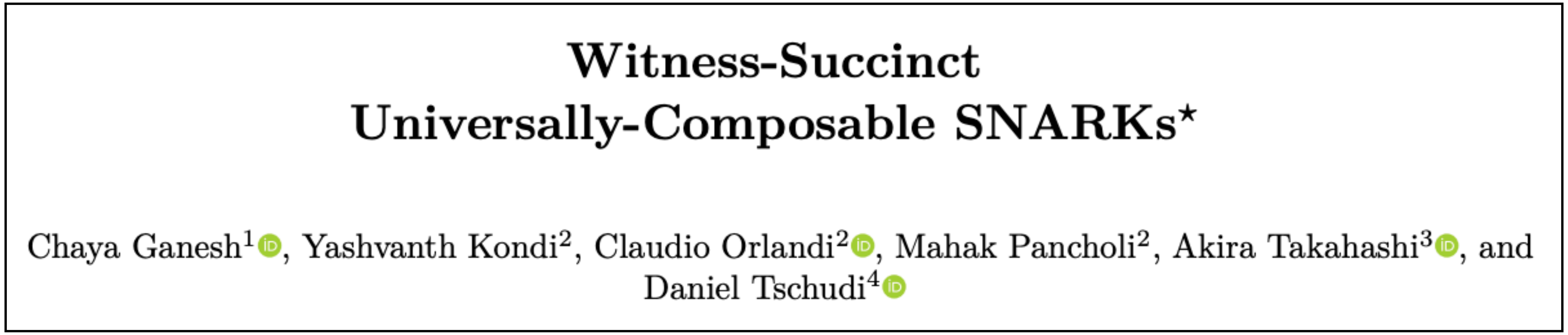
Chaya Ganesh[1], Yashvanth Kondi[2], Claudio Orlandi[2], Mahak Pancholi[2], Akira Takahashi[3], and Daniel Tschudi[4]

First UC-secure **S**NARK

Combines simulation-extractable zkSNARK with a PCS

Use Fischlin-like techniques to achieve straight-line extraction

Succinct UC-secure zkSNARKs



Witness-Succinct Universally-Composable SNARKs⋆

Chaya Ganesh[1], Yashvanth Kondi[2], Claudio Orlandi[2], Mahak Pancholi[2], Akira Takahashi[3], and Daniel Tschudi[4]

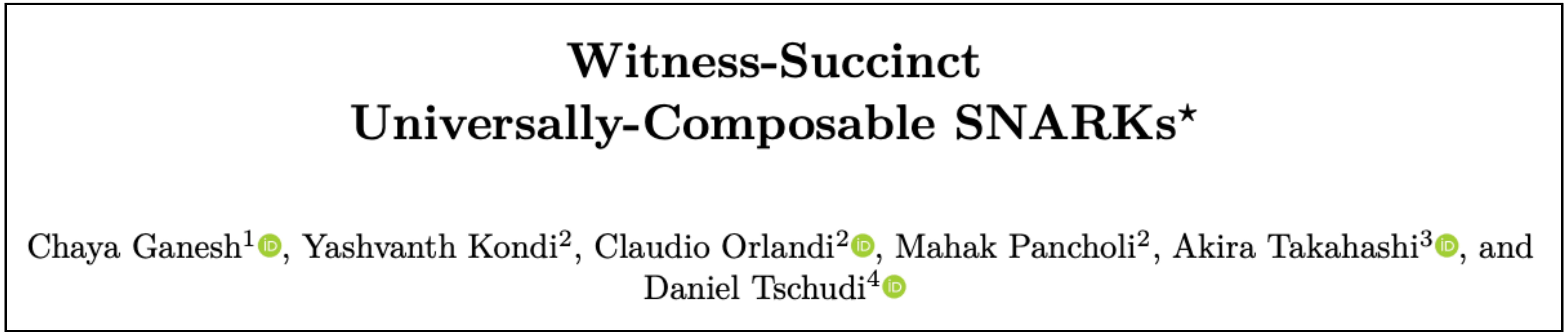
First UC-secure **S**NARK

+ Achieves succinct proofs

Combines simulation-extractable zkSNARK with a PCS

Use Fischlin-like techniques to achieve straight-line extraction

Succinct UC-secure zkSNARKs



Witness-Succinct Universally-Composable SNARKs*

Chaya Ganesh[1], Yashvanth Kondi[2], Claudio Orlandi[2], Mahak Pancholi[2], Akira Takahashi[3], and Daniel Tschudi[4]

First UC-secure **S**NARK

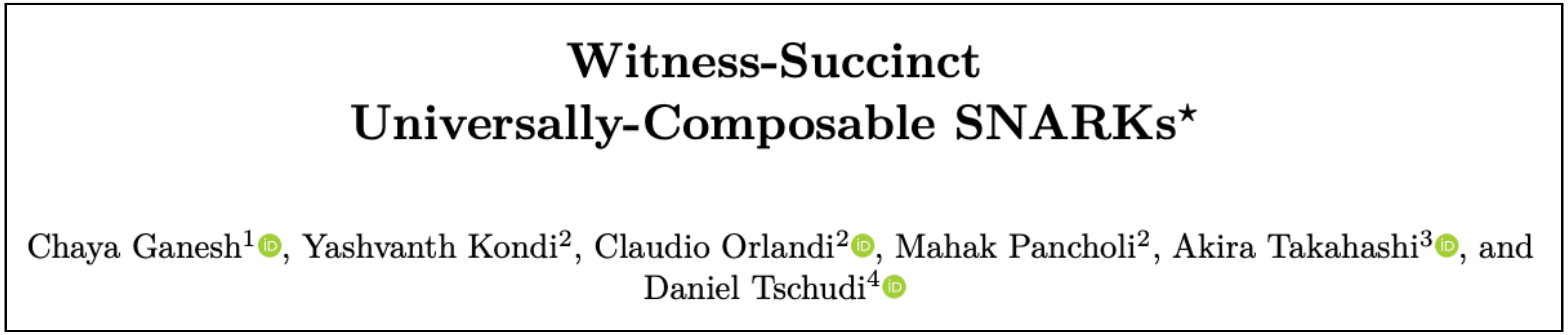
Combines simulation-extractable zkSNARK with a PCS

Use Fischlin-like techniques to achieve straight-line extraction

+ Achieves succinct proofs

+ UC-Secure in the (non-programmable) observable GROM

Succinct UC-secure zkSNARKs



Witness-Succinct Universally-Composable SNARKs*

Chaya Ganesh[1], Yashvanth Kondi[2], Claudio Orlandi[2], Mahak Pancholi[2], Akira Takahashi[3], and Daniel Tschudi[4]

First UC-secure **S**NARK

Combines simulation-extractable zkSNARK with a PCS

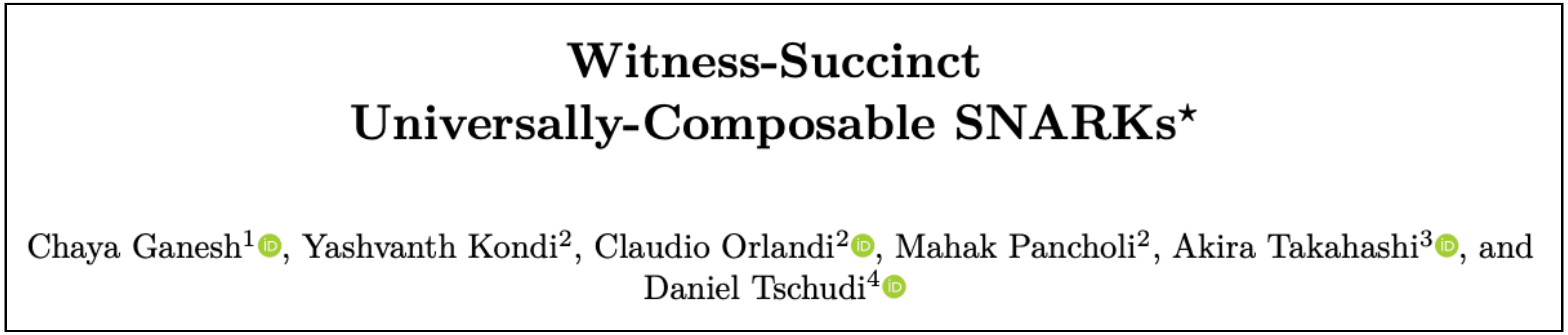
Use Fischlin-like techniques to achieve straight-line extraction

+ Achieves succinct proofs

+ UC-Secure in the (non-programmable) observable GROM

- Expensive non-standard construction

Succinct UC-secure zkSNARKs



Witness-Succinct Universally-Composable SNARKs*

Chaya Ganesh[1], Yashvanth Kondi[2], Claudio Orlandi[2], Mahak Pancholi[2], Akira Takahashi[3], and Daniel Tschudi[4]

First UC-secure **S**NARK

Combines simulation-extractable zkSNARK with a PCS

Use Fischlin-like techniques to achieve straight-line extraction

+ Achieves succinct proofs

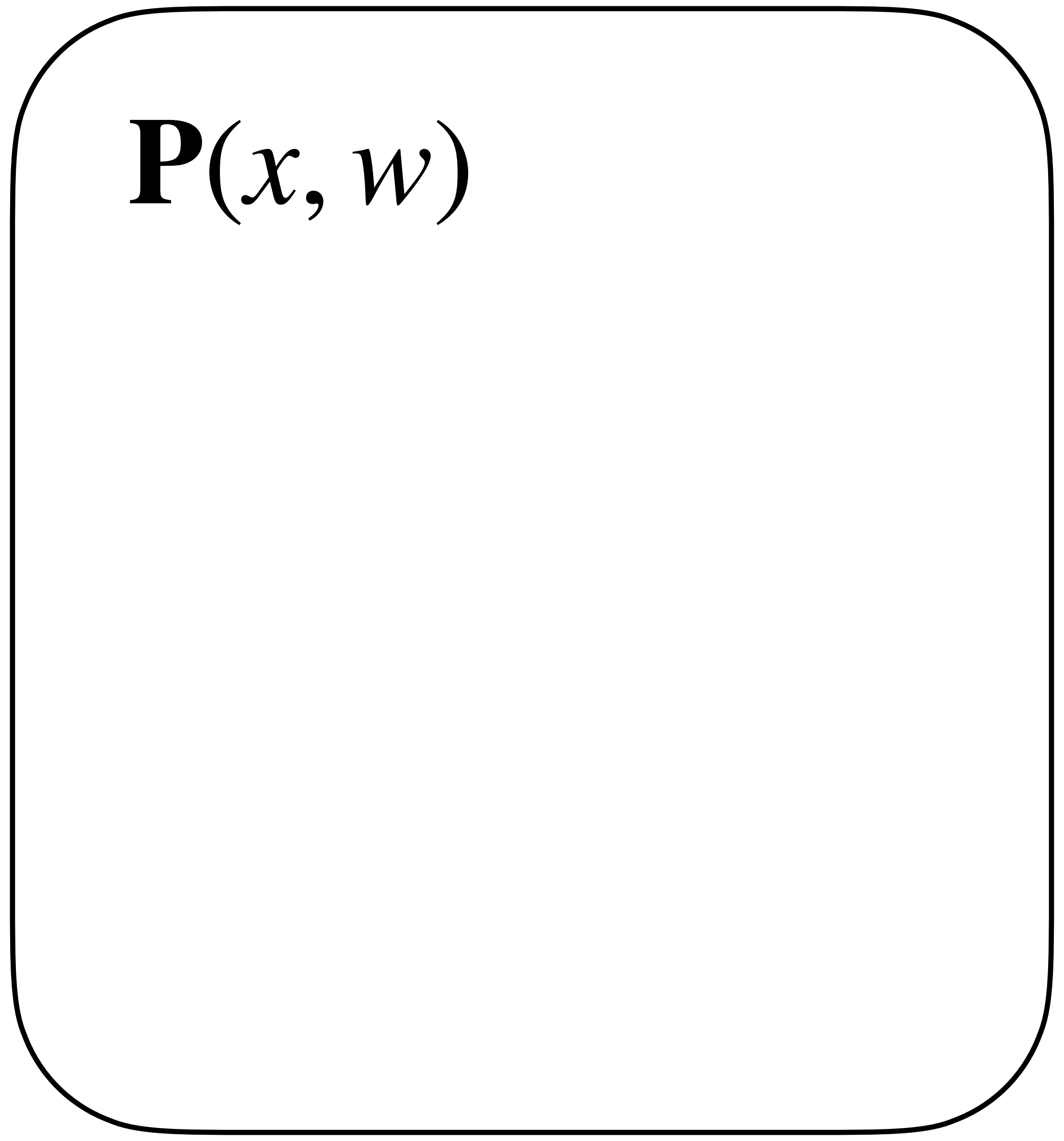
+ UC-Secure in the (non-programmable) observable GROM

- Expensive non-standard construction

- Focuses on asymptotic security

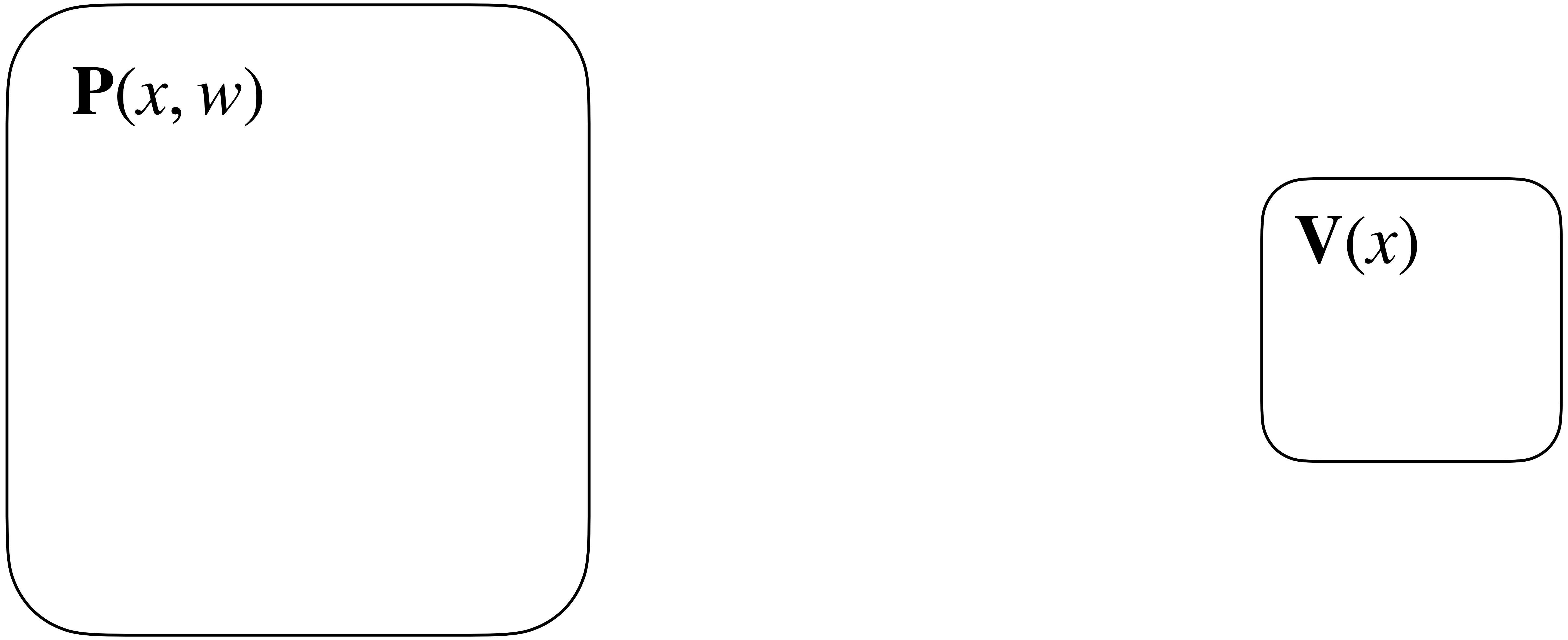
zkSNARKs (in the ROM)

zkSNARKs (in the ROM)



$\mathbf{P}(x, w)$

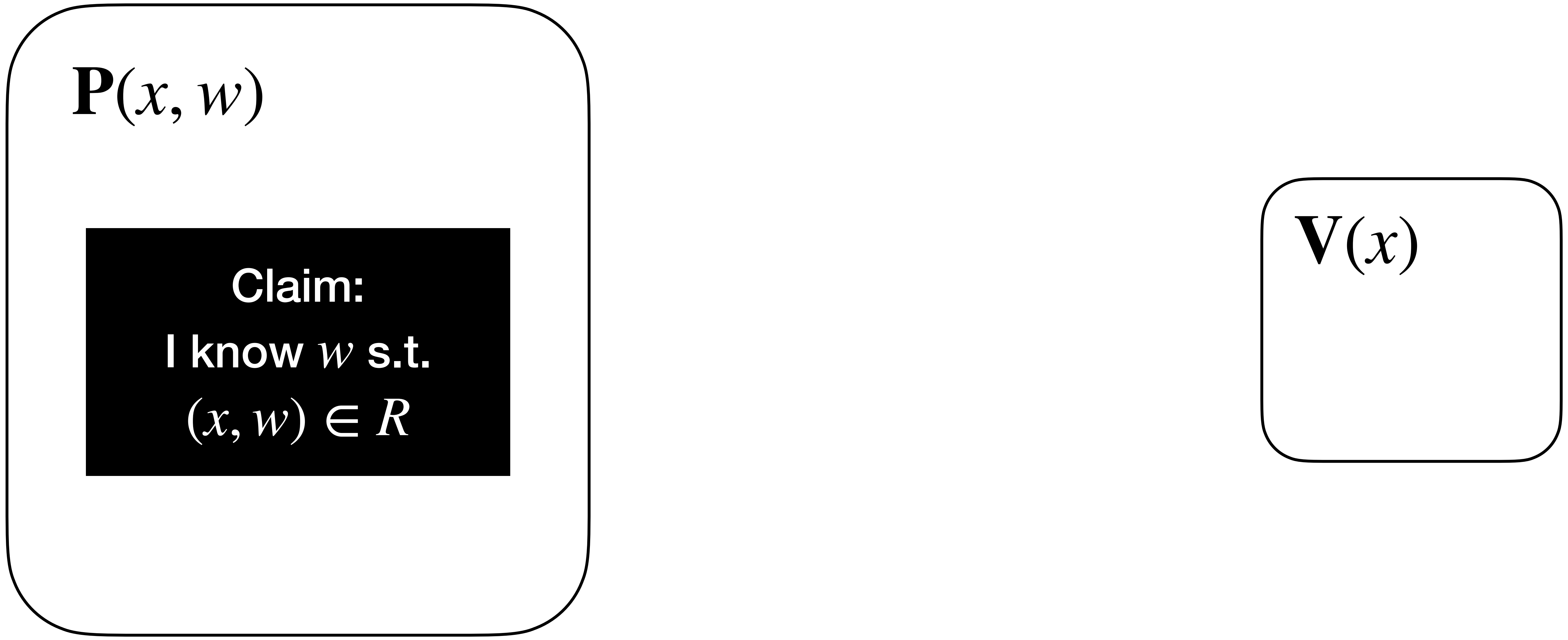
zkSNARKs (in the ROM)



$\mathbf{P}(x, w)$

$\mathbf{V}(x)$

zkSNARKs (in the ROM)

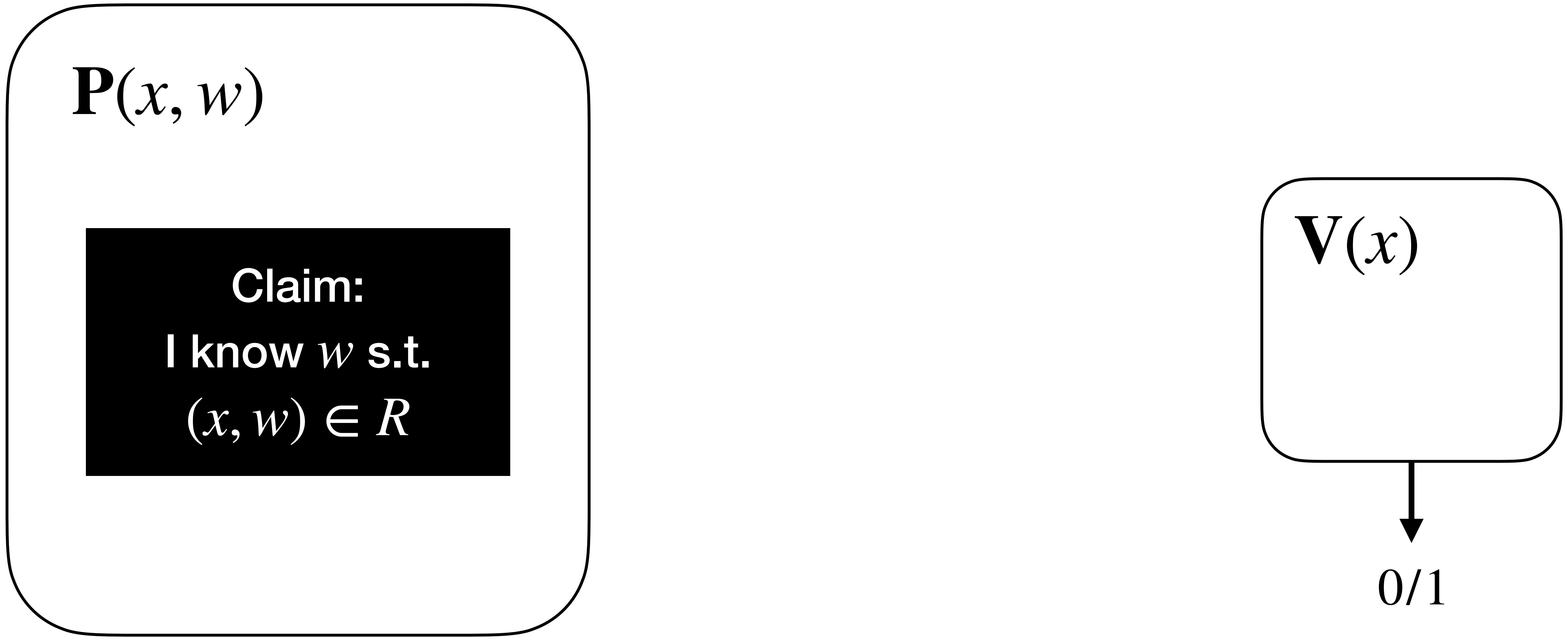


$\mathbf{P}(x, w)$

Claim:
I know w s.t.
$(x, w) \in R$

$\mathbf{V}(x)$

zkSNARKs (in the ROM)



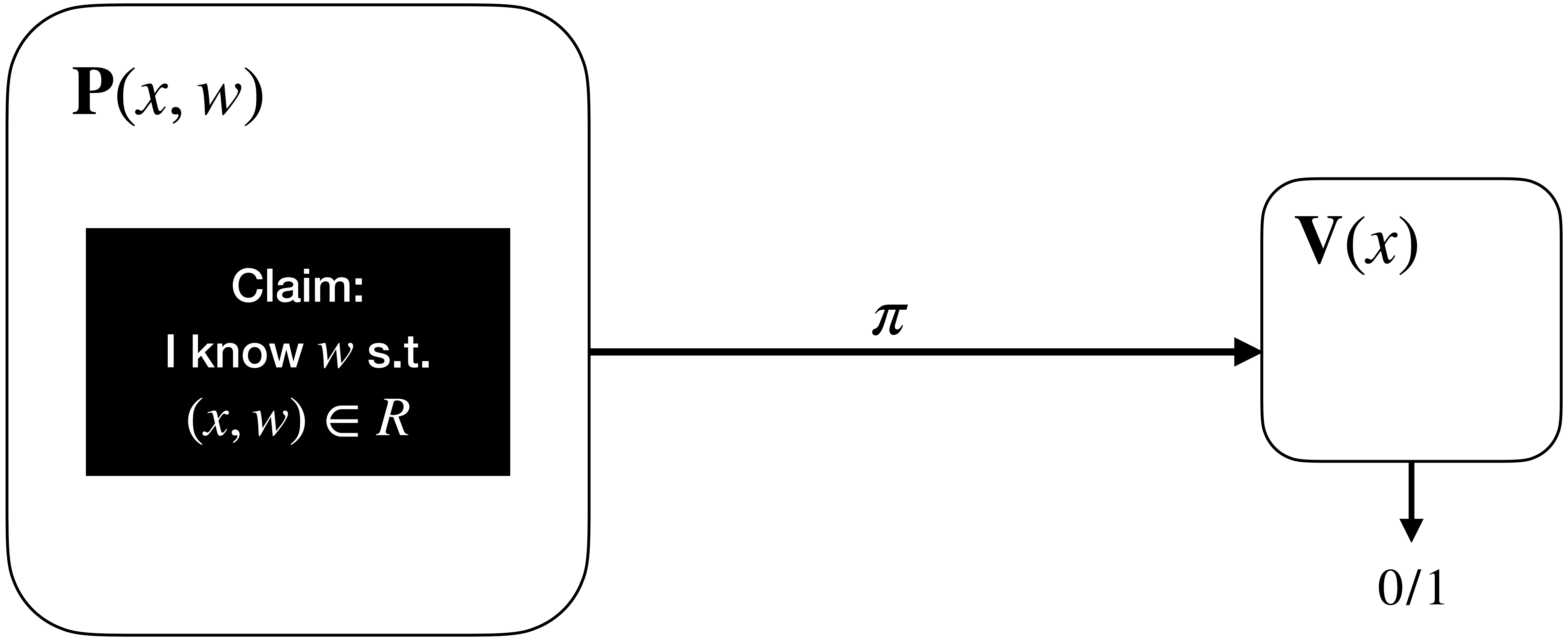
$\mathbf{P}(x, w)$

Claim:
I know w s.t.
$(x, w) \in R$

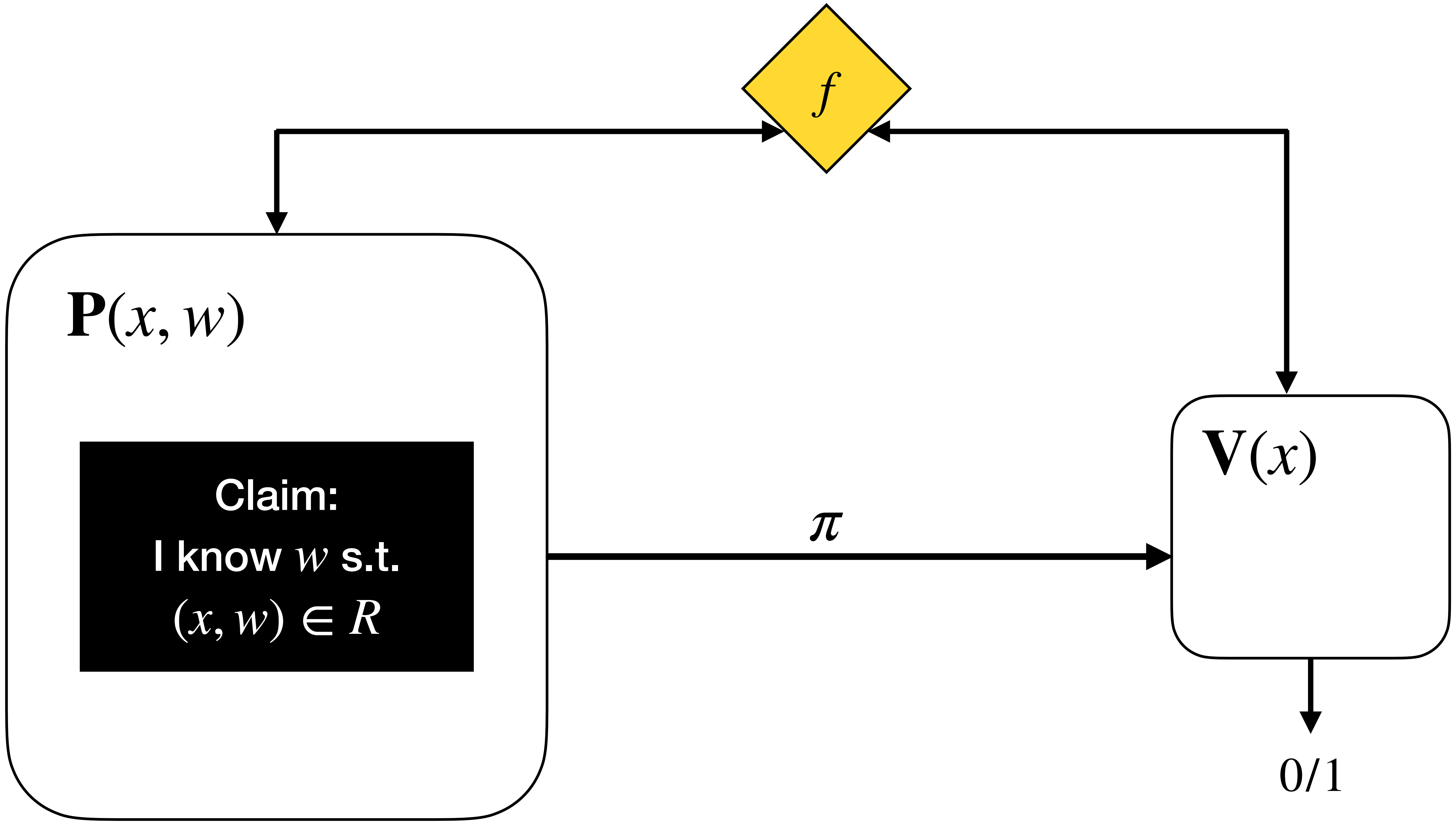
$\mathbf{V}(x)$

0/1

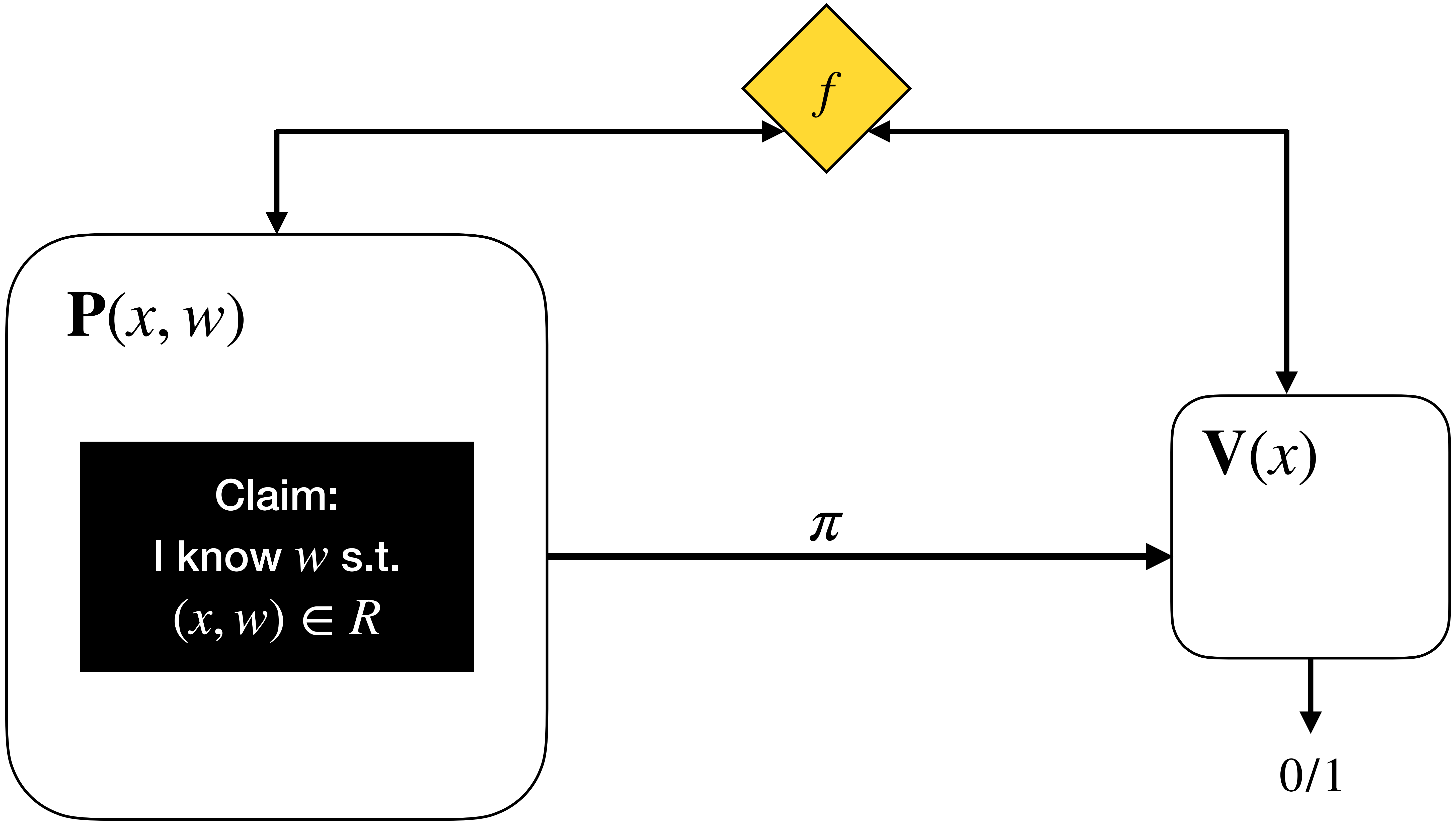
zkSNARKs (in the ROM)



zkSNARKs (in the ROM)

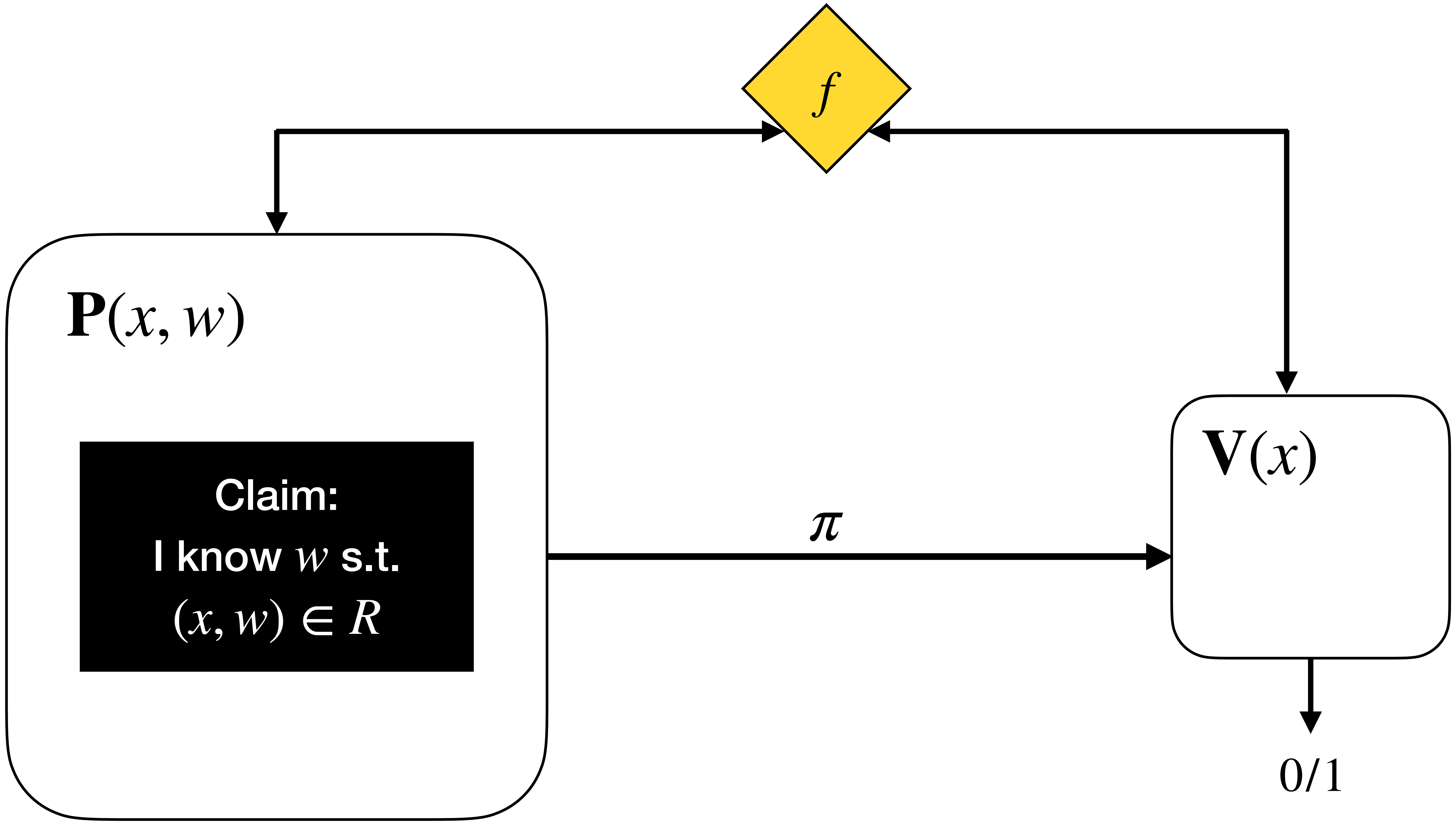


zkSNARKs (in the ROM)



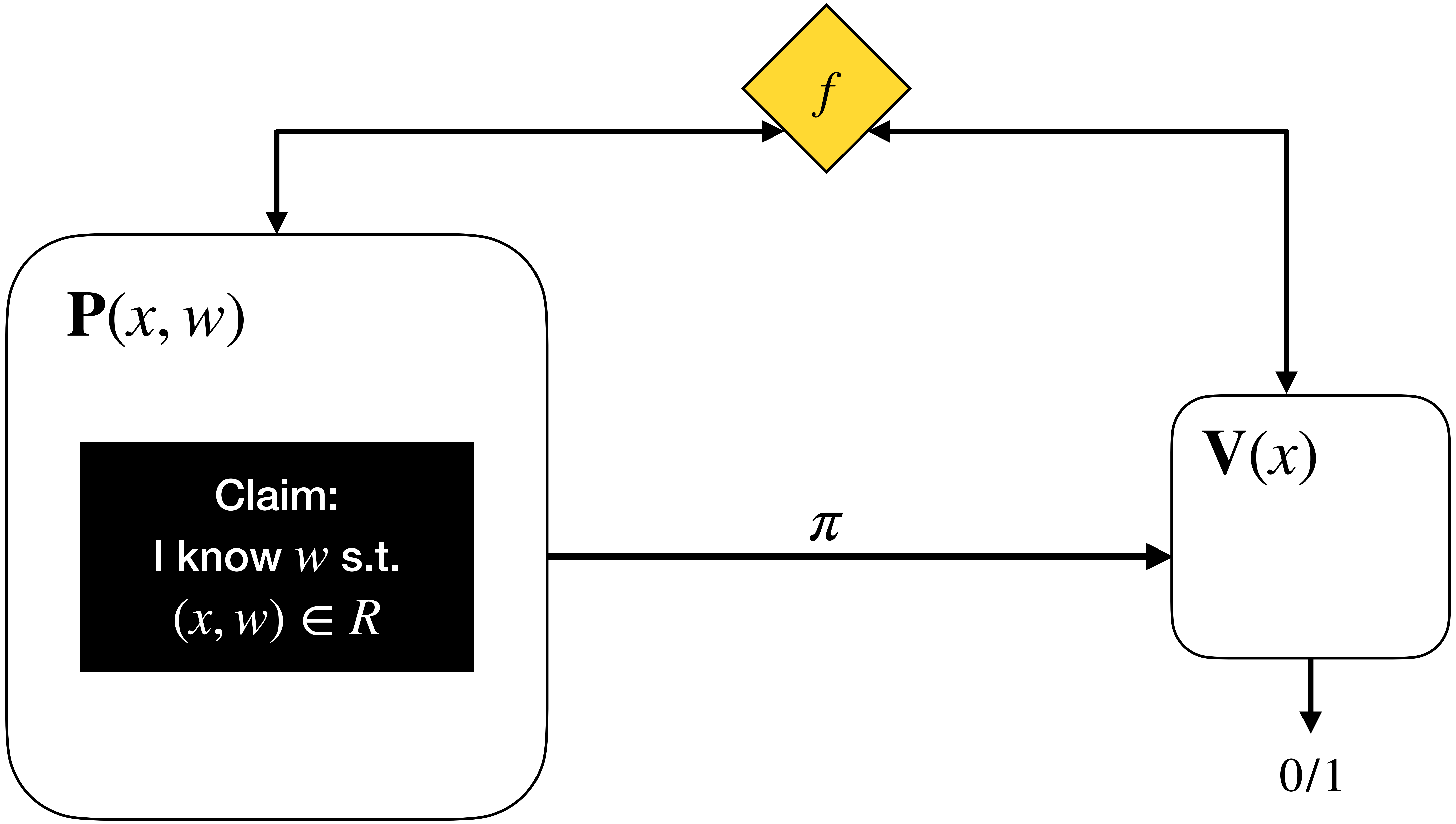
- Zero-Knowledge
  - $\exists \mathbf{S} : \ \mathbf{P}^f(x, w) \approx \mathbf{S}^f(x)$

zkSNARKs (in the ROM)



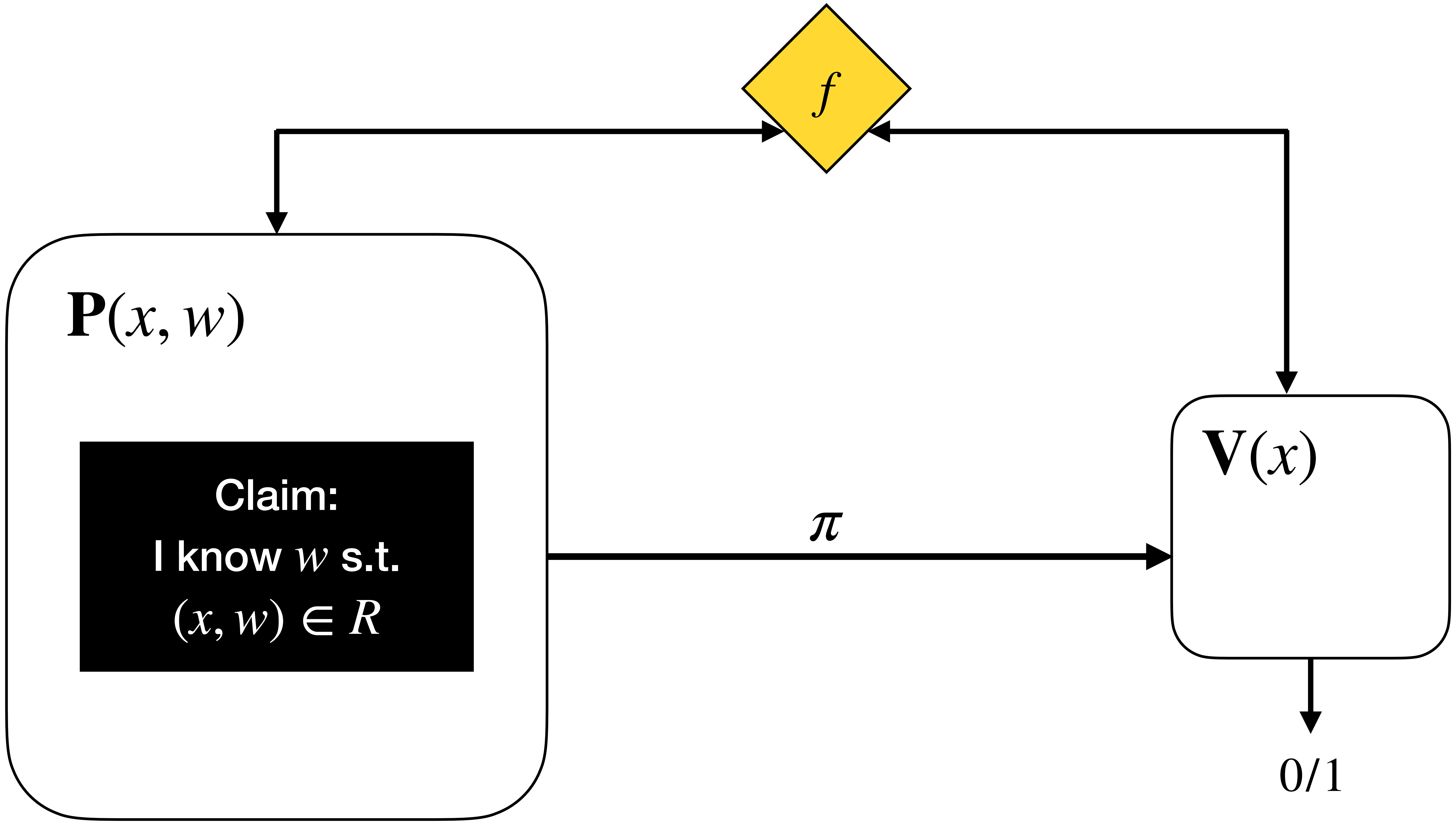
- Zero-Knowledge
  - $\exists \mathbf{S} : \mathbf{P}^f(x, w) \approx \mathbf{S}^f(x)$
- Succinct
  - $|\pi| \ll |w|$

zkSNARKs (in the ROM)



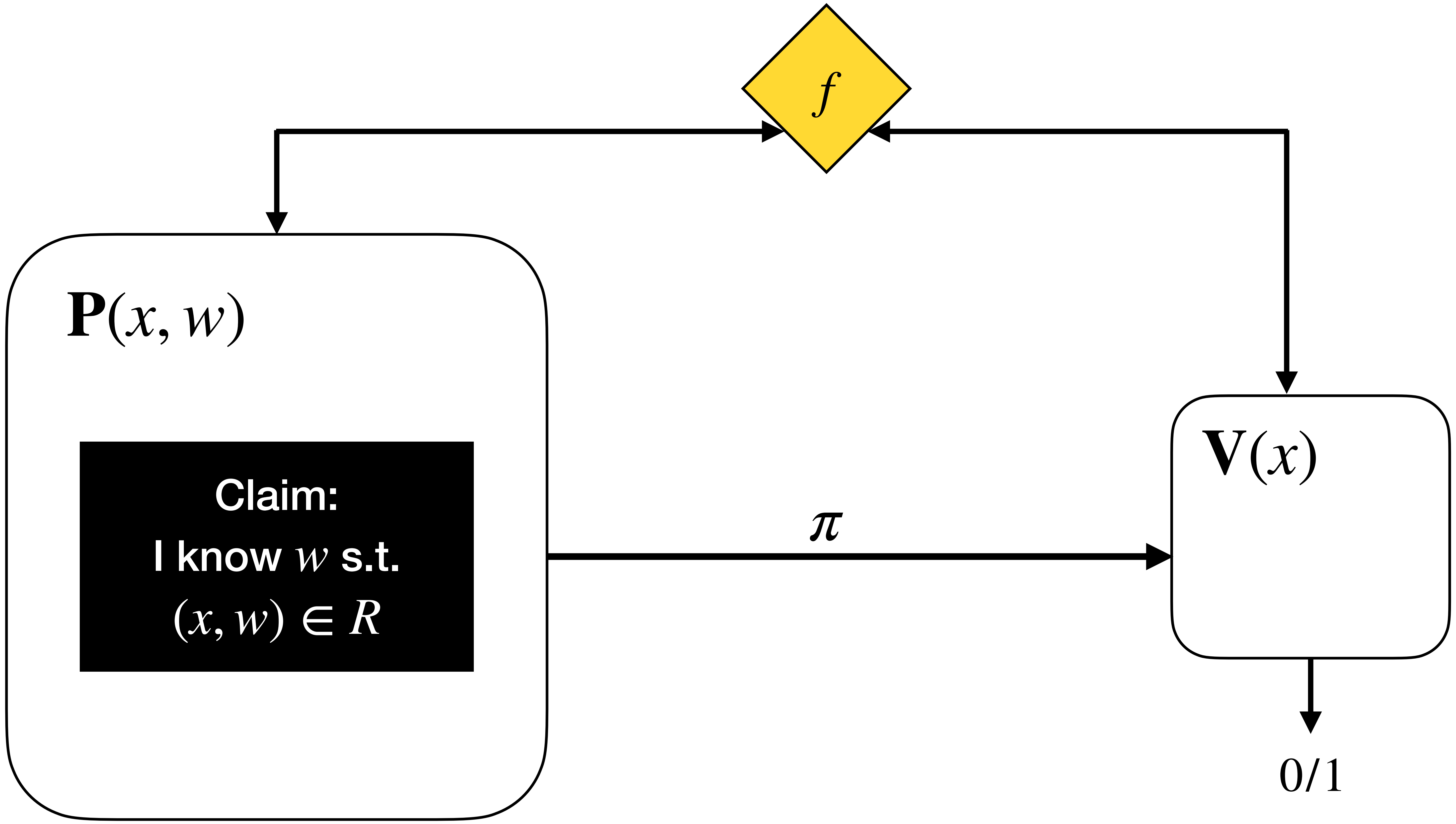
- Zero-Knowledge
  - ○ $\exists \mathbf{S} : \mathbf{P}^f(x, w) \approx \mathbf{S}^f(x)$
- Succinct
  - ○ $|\pi| \ll |w|$
- Non-interactive

zkSNARKs (in the ROM)



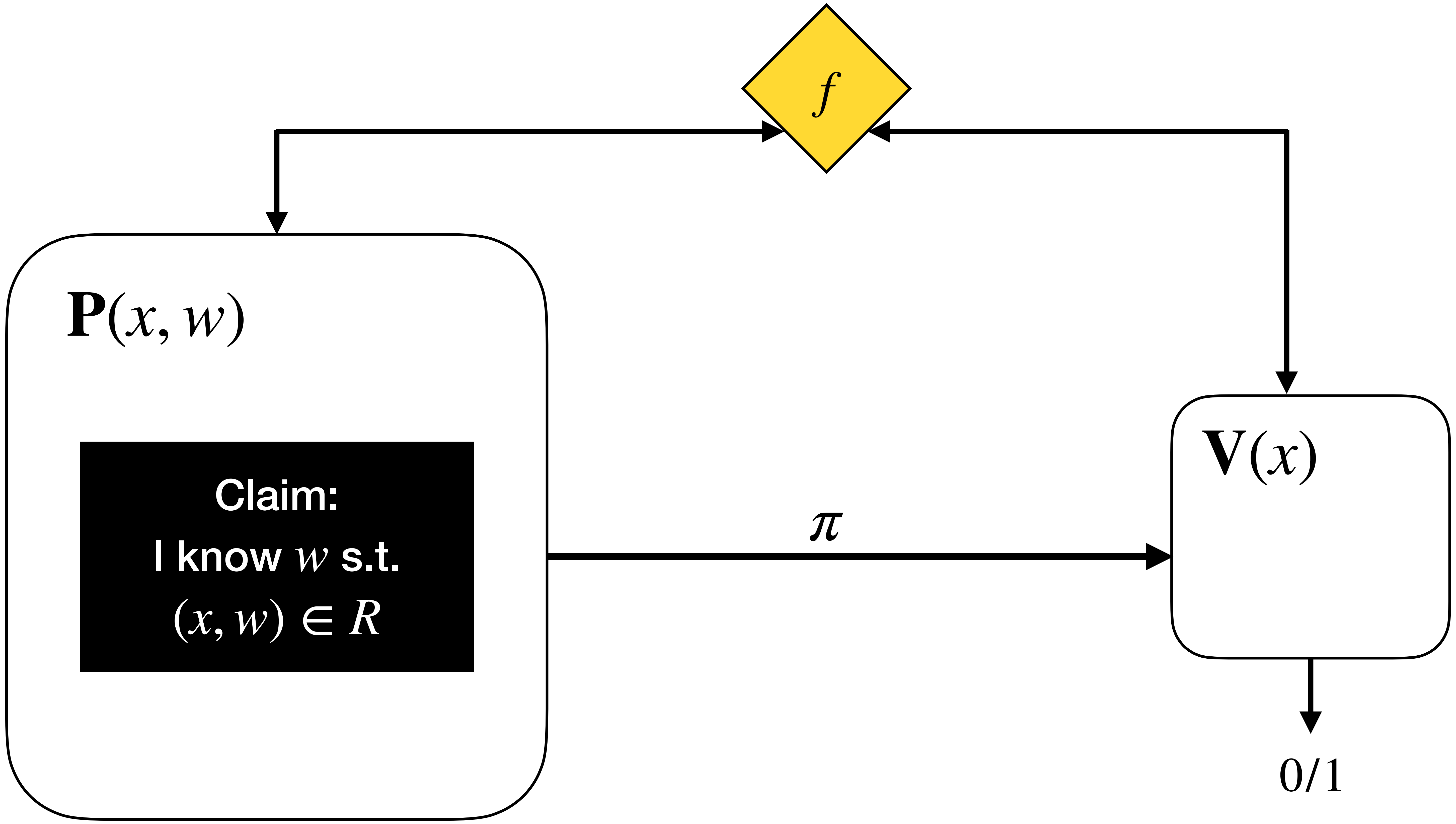
- Zero-Knowledge
  - $\exists \mathbf{S} : \ \mathbf{P}^f(x, w) \approx \mathbf{S}^f(x)$
- Succinct
  - $|\pi| \ll |w|$
- Non-interactive
- Argument of Knowledge

zkSNARKs (in the ROM)



- Zero-Knowledge
  - $\exists \mathbf{S}: \ \mathbf{P}^f(x, w) \approx \mathbf{S}^f(x)$
- Succinct
  - $|\pi| \ll |w|$
- Non-interactive
- Argument of Knowledge
  - $\exists \mathbf{E}: \ \mathbf{V}^f(x, \pi \leftarrow \tilde{\mathbf{P}}) = 1$

zkSNARKs (in the ROM)



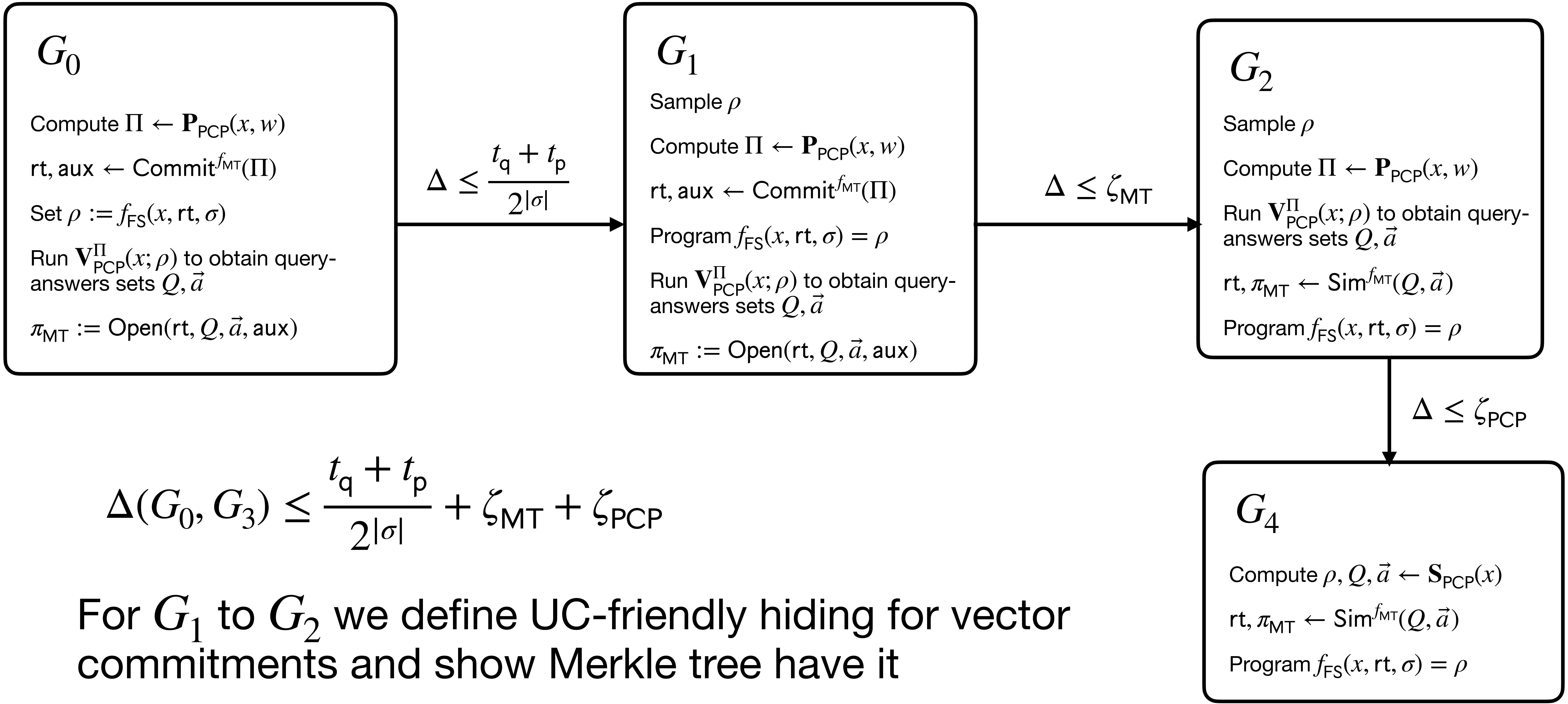
- Zero-Knowledge
  - $\exists \mathbf{S} : \ \mathbf{P}^f(x, w) \approx \mathbf{S}^f(x)$
- Succinct
  - $|\pi| \ll |w|$
- Non-interactive
- Argument of Knowledge
  - $\exists \mathbf{E} : \ \mathbf{V}^f(x, \pi \leftarrow \tilde{\mathbf{P}}) = 1$ $\implies (x, \mathbf{E}(x, \pi, \mathrm{tr}_{\tilde{\mathbf{P}}})) \in R$

What if we only care about scalability?

Dropping ZK

- Often, SNARKs are deployed without ZK
- We consider this out of scope for this work but (at an high level) believe that:
    - The techniques here would still work and can be simplified.
    - Remove UC-friendly ZK and move to non-programmable GROM.
    - UC-completeness then reduces to perfect completeness.
    - Knowledge sound PCP/IOP suffices for Micali/BCS.

Micali has UC-friendly ZK



G_0

Compute $\Pi \leftarrow \mathbf{P}_{\mathsf{PCP}}(x, w)$

rt, aux $\leftarrow$ Commit$^{f_{\mathsf{MT}}}(\Pi)$

Set $\rho := f_{\mathsf{FS}}(x, \mathsf{rt}, \sigma)$

Run $\mathbf{V}^{\Pi}_{\mathsf{PCP}}(x; \rho)$ to obtain query-answers sets $Q, \vec{a}$

$\pi_{\mathsf{MT}} :=$ Open(rt, $Q, \vec{a}$, aux)

$\Delta \leq \frac{t_{\mathsf{q}} + t_{\mathsf{p}}}{2^{|\sigma|}}$

G_1

Sample ρ

Compute $\Pi \leftarrow \mathbf{P}_{\mathsf{PCP}}(x, w)$

rt, aux $\leftarrow$ Commit$^{f_{\mathsf{MT}}}(\Pi)$

Program $f_{\mathsf{FS}}(x, \mathsf{rt}, \sigma) = \rho$

Run $\mathbf{V}^{\Pi}_{\mathsf{PCP}}(x; \rho)$ to obtain query-answers sets $Q, \vec{a}$

$\pi_{\mathsf{MT}} :=$ Open(rt, $Q, \vec{a}$, aux)

$\Delta \leq \zeta_{\mathsf{MT}}$

G_2

Sample ρ

Compute $\Pi \leftarrow \mathbf{P}_{\mathsf{PCP}}(x, w)$

Run $\mathbf{V}^{\Pi}_{\mathsf{PCP}}(x; \rho)$ to obtain query-answers sets $Q, \vec{a}$

rt, $\pi_{\mathsf{MT}} \leftarrow$ Sim$^{f_{\mathsf{MT}}}(Q, \vec{a})$

Program $f_{\mathsf{FS}}(x, \mathsf{rt}, \sigma) = \rho$

$\Delta \leq \zeta_{\mathsf{PCP}}$

G_4

Compute $\rho, Q, \vec{a} \leftarrow \mathbf{S}_{\mathsf{PCP}}(x)$

rt, $\pi_{\mathsf{MT}} \leftarrow$ Sim$^{f_{\mathsf{MT}}}(Q, \vec{a})$

Program $f_{\mathsf{FS}}(x, \mathsf{rt}, \sigma) = \rho$

$$\Delta(G_0, G_3) \leq \frac{t_{\mathsf{q}} + t_{\mathsf{p}}}{2^{|\sigma|}} + \zeta_{\mathsf{MT}} + \zeta_{\mathsf{PCP}}$$

For G_1 to G_2 we define UC-friendly hiding for vector commitments and show Merkle tree have it

Micali has UC-friendly completeness

Micali has UC-friendly completeness

UC-friendly completeness

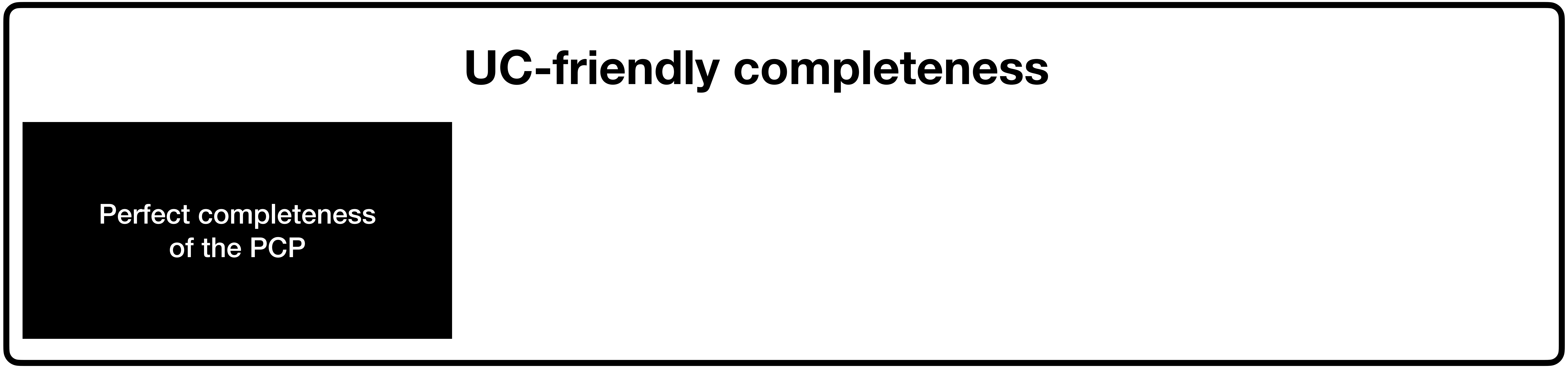
Micali has UC-friendly completeness

- Assuming **PCP perfect completeness**, honest proof are rejected only if the verifier queries a previously programmed point.

UC-friendly completeness

Micali has UC-friendly completeness

- Assuming **PCP perfect completeness**, honest proof are rejected only if the verifier queries a previously programmed point.



UC-friendly completeness

Perfect completeness of the PCP

Micali has UC-friendly completeness

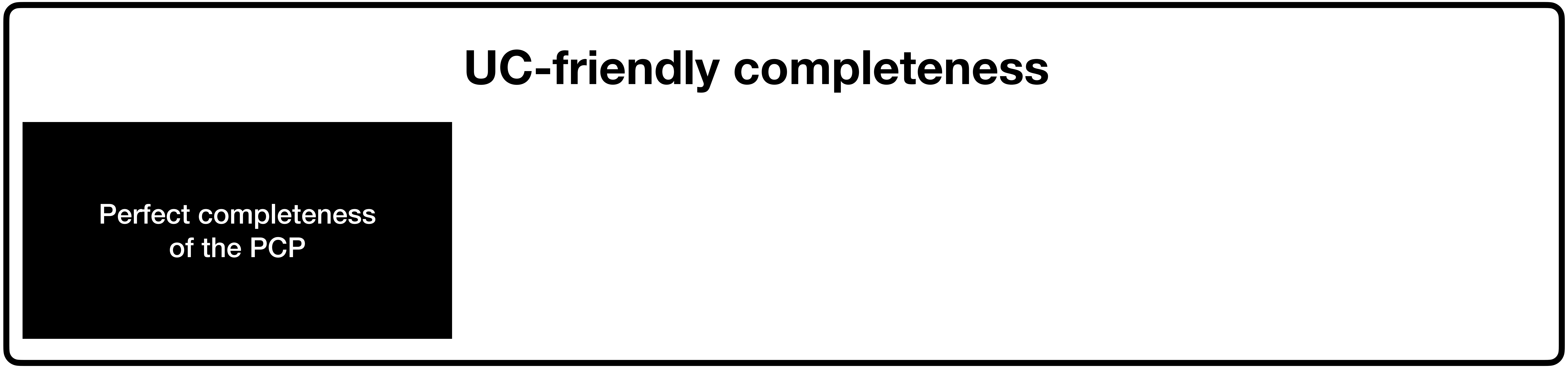
- Assuming **PCP perfect completeness**, honest proof are rejected only if the verifier queries a previously programmed point.
- Disallow this attack with two natural properties:

UC-friendly completeness

Perfect completeness of the PCP

Micali has UC-friendly completeness

- Assuming **PCP perfect completeness**, honest proof are rejected only if the verifier queries a previously programmed point.
- Disallow this attack with two natural properties:
  - **Monotone proofs** (verifier does not query points not previously queried by the prover)



UC-friendly completeness

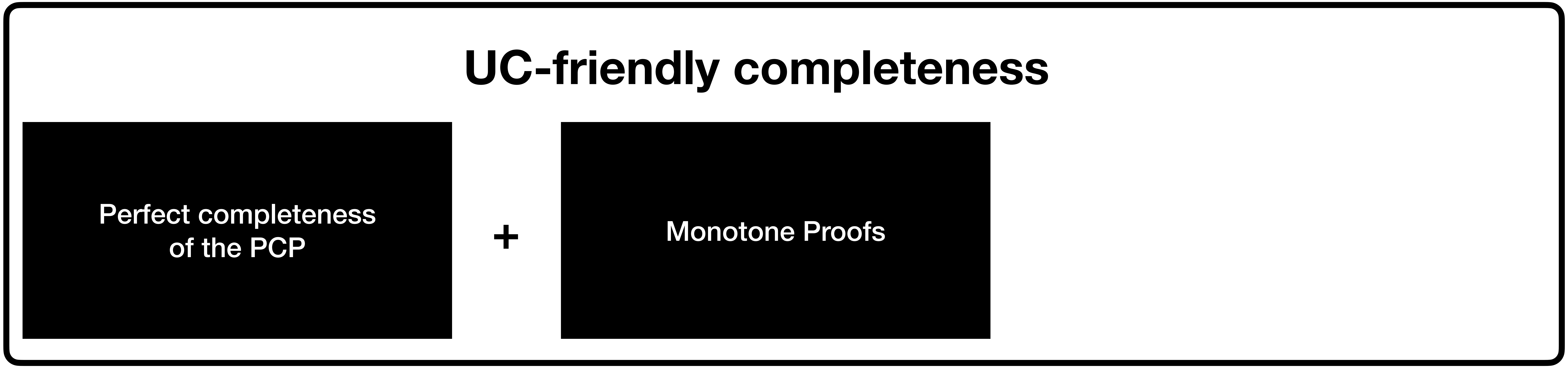
Perfect completeness of the PCP

Micali has UC-friendly completeness

- Assuming **PCP perfect completeness**, honest proof are rejected only if the verifier queries a previously programmed point.

- Disallow this attack with two natural properties:

  - **Monotone proofs** (verifier does not query points not previously queried by the prover)



UC-friendly completeness

Perfect completeness of the PCP

+

Monotone Proofs

Micali has UC-friendly completeness

- Assuming **PCP perfect completeness**, honest proof are rejected only if the verifier queries a previously programmed point.

- Disallow this attack with two natural properties:

  - **Monotone proofs** (verifier does not query points not previously queried by the prover)

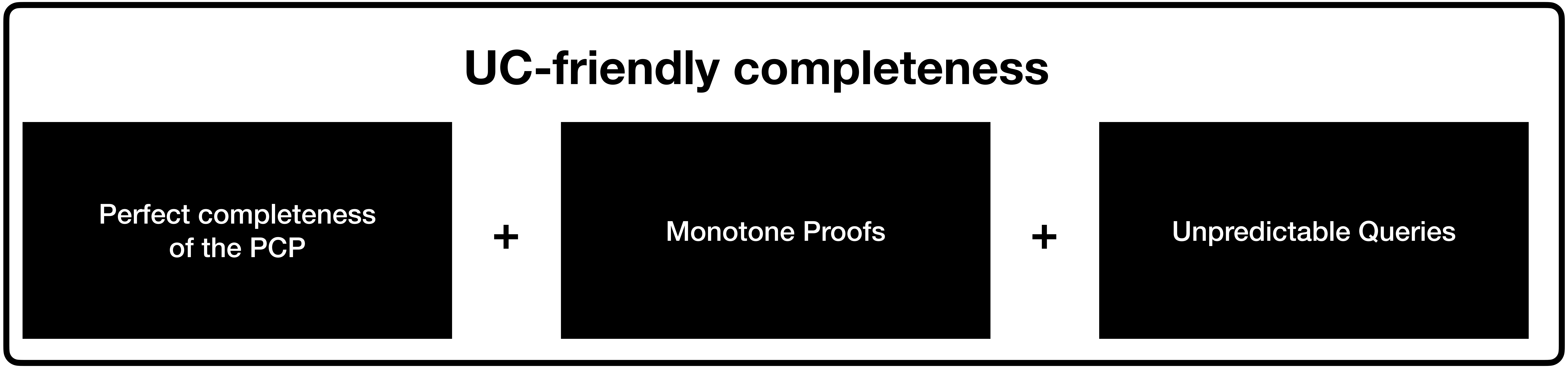
  - **Unpredictable queries** (hard to program points prover will query)

UC-friendly completeness

Perfect completeness of the PCP + Monotone Proofs

Micali has UC-friendly completeness

- Assuming **PCP perfect completeness**, honest proof are rejected only if the verifier queries a previously programmed point.
- Disallow this attack with two natural properties:
  - **Monotone proofs** (verifier does not query points not previously queried by the prover)
  - **Unpredictable queries** (hard to program points prover will query)



UC-friendly completeness

Perfect completeness of the PCP + Monotone Proofs + Unpredictable Queries

Micali has UC-friendly KS

Micali has UC-friendly KS

UC-friendly KS of Micali

Micali has UC-friendly KS

- UC-friendly KS implies simulation-extractability.

UC-friendly KS of Micali

Micali has UC-friendly KS

- UC-friendly KS implies simulation-extractability.
- Merkle trees are **non-malleable** already.

UC-friendly KS of Micali

Micali has UC-friendly KS

- UC-friendly KS implies simulation-extractability.
- Merkle trees are **non-malleable** already.
- In Micali, makes proofs **non-malleable**.

UC-friendly KS of Micali

Micali has UC-friendly KS

- UC-friendly KS implies simulation-extractability.
- Merkle trees are **non-malleable** already.
- In Micali, makes proofs **non-malleable**.
- Reduce to **state-restoration KS** (implied by KS of PCP)

UC-friendly KS of Micali

Micali has UC-friendly KS

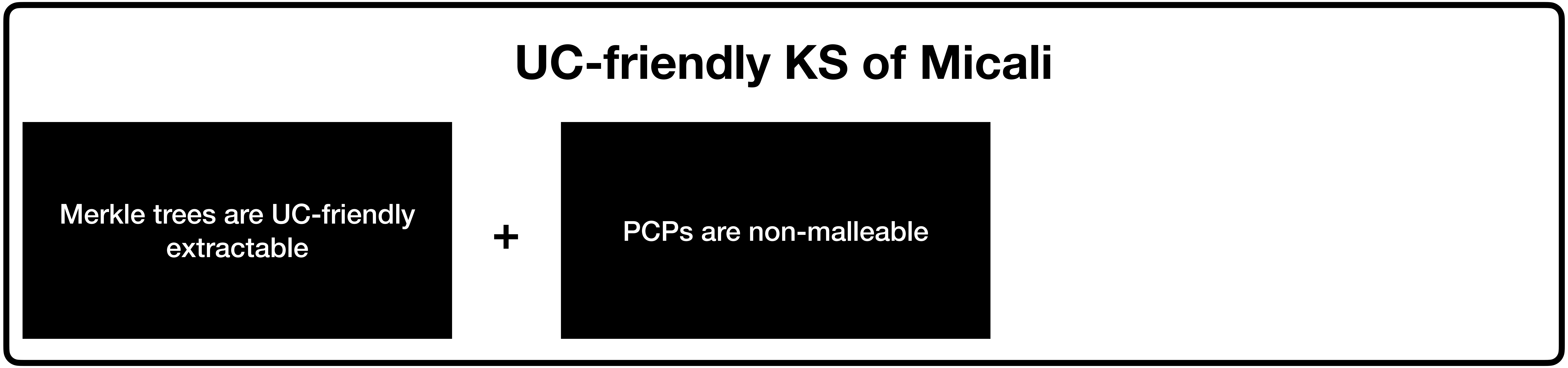
- UC-friendly KS implies simulation-extractability.
- Merkle trees are **non-malleable** already.
- In Micali, makes proofs **non-malleable**.
- Reduce to **state-restoration KS** (implied by KS of PCP)

UC-friendly KS of Micali

Merkle trees are UC-friendly extractable

Micali has UC-friendly KS

- UC-friendly KS implies simulation-extractability.
- Merkle trees are **non-malleable** already.
- In Micali, makes proofs **non-malleable**.
- Reduce to **state-restoration KS** (implied by KS of PCP)



UC-friendly KS of Micali

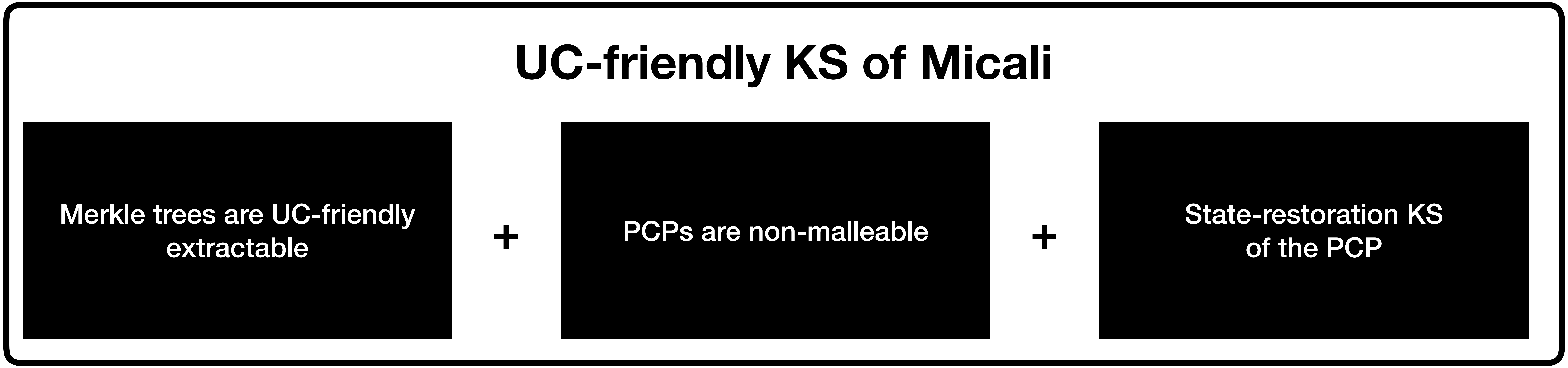
Merkle trees are UC-friendly extractable

+

PCPs are non-malleable

Micali has UC-friendly KS

- UC-friendly KS implies simulation-extractability.
- Merkle trees are **non-malleable** already.
- In Micali, makes proofs **non-malleable**.
- Reduce to **state-restoration KS** (implied by KS of PCP)



UC-friendly KS of Micali

Merkle trees are UC-friendly extractable + PCPs are non-malleable + State-restoration KS of the PCP

Related works

Known UC-secure zkSNARKs

Related works

Known UC-secure zkSNARKs

Non-Witness Succinct

Related works

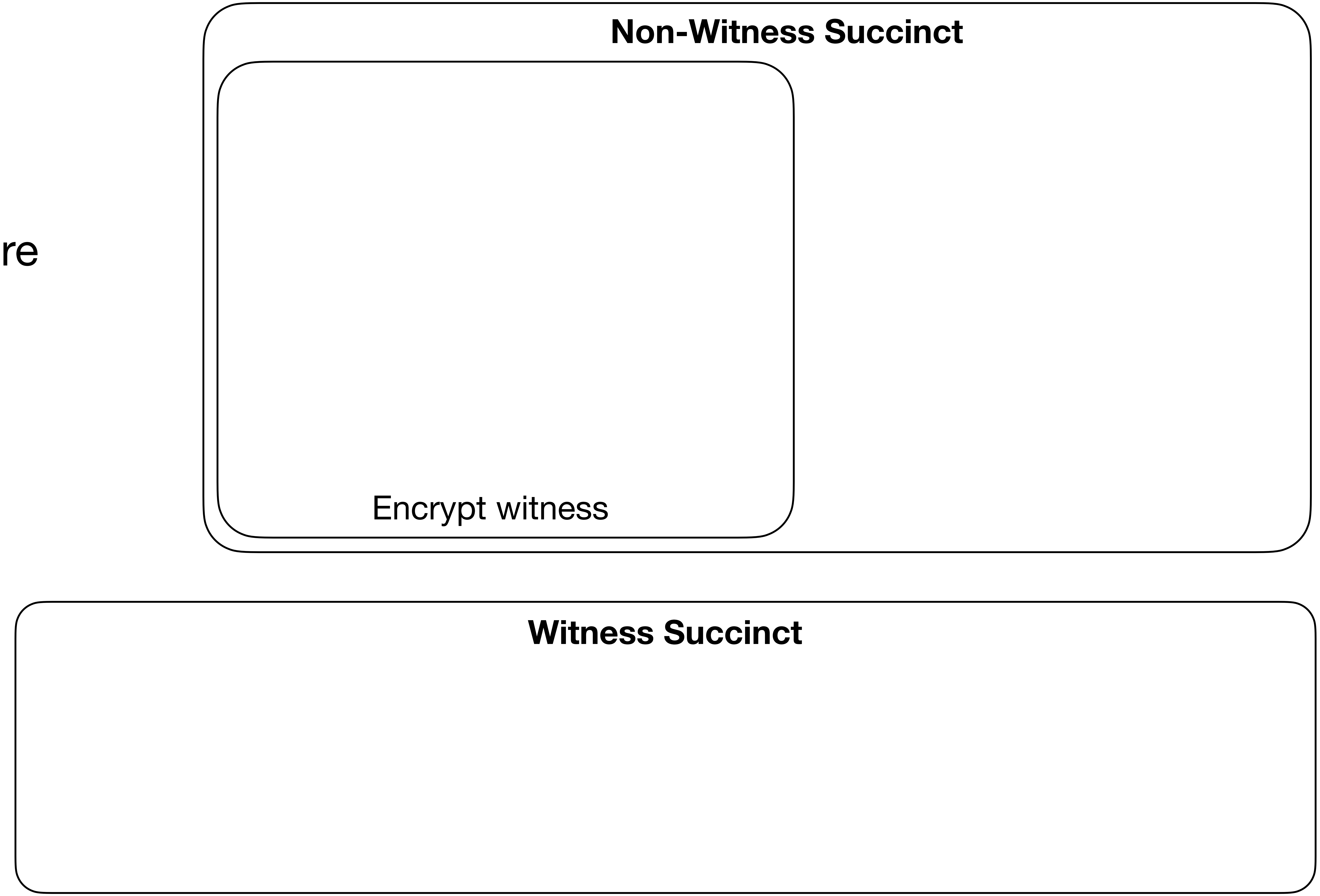
Known UC-secure zkSNARKs

Non-Witness Succinct

Witness Succinct

Related works

Known UC-secure zkSNARKs



Non-Witness Succinct

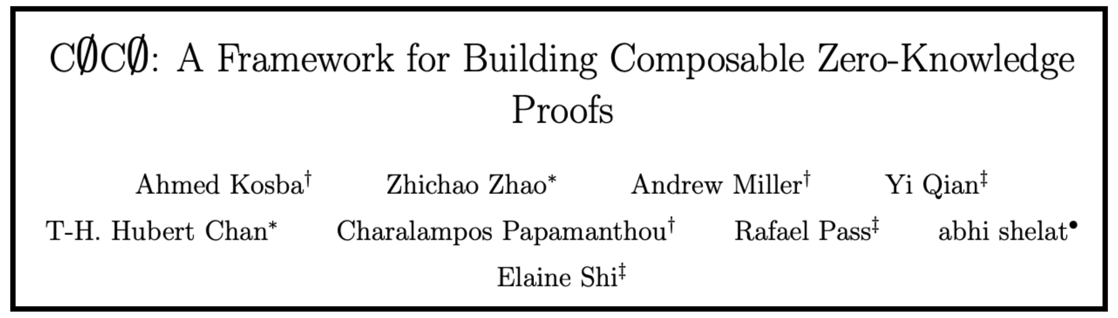
Encrypt witness

Witness Succinct

Related works

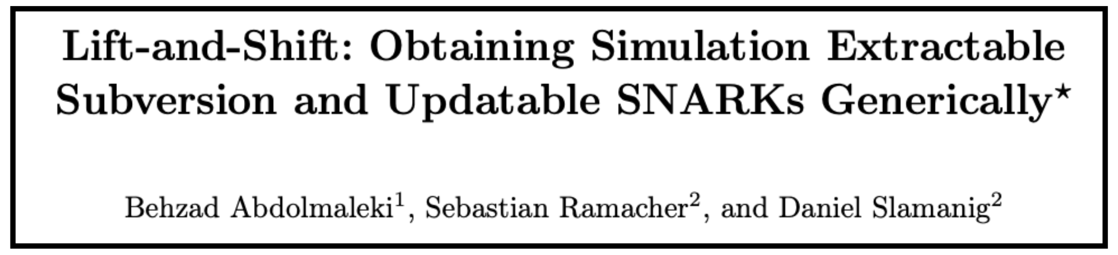
Known UC-secure zkSNARKs

Non-Witness Succinct



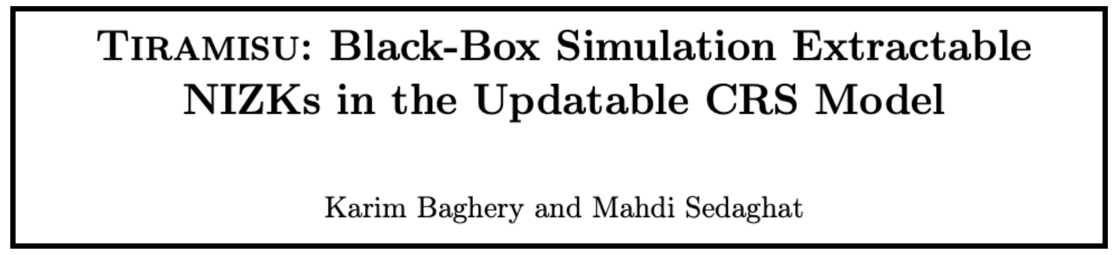
CØCØ: A Framework for Building Composable Zero-Knowledge Proofs

Ahmed Kosba[†] Zhichao Zhao[*] Andrew Miller[†] Yi Qian[‡]
T-H. Hubert Chan[*] Charalampos Papamanthou[†] Rafael Pass[‡] abhi shelat[•]
Elaine Shi[‡]



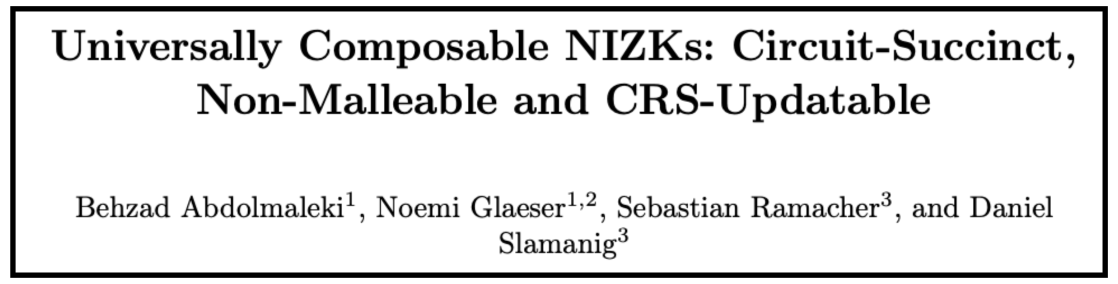
Lift-and-Shift: Obtaining Simulation Extractable Subversion and Updatable SNARKs Generically[⋆]

Behzad Abdolmaleki[1], Sebastian Ramacher[2], and Daniel Slamanig[2]



Tiramisu: Black-Box Simulation Extractable NIZKs in the Updatable CRS Model

Karim Baghery and Mahdi Sedaghat



Universally Composable NIZKs: Circuit-Succinct, Non-Malleable and CRS-Updatable

Behzad Abdolmaleki[1], Noemi Glaeser[1,2], Sebastian Ramacher[3], and Daniel Slamanig[3]

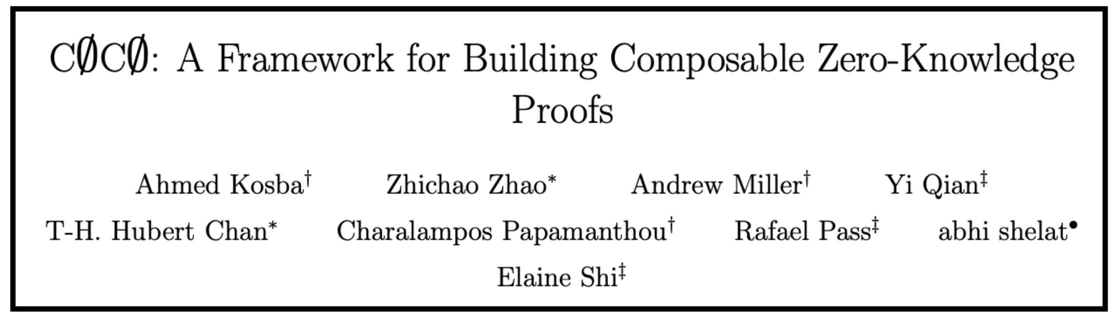
Encrypt witness

Witness Succinct

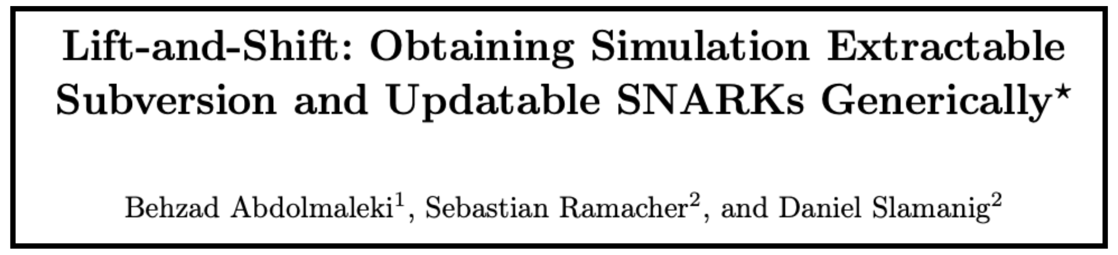
Related works

Known UC-secure zkSNARKs

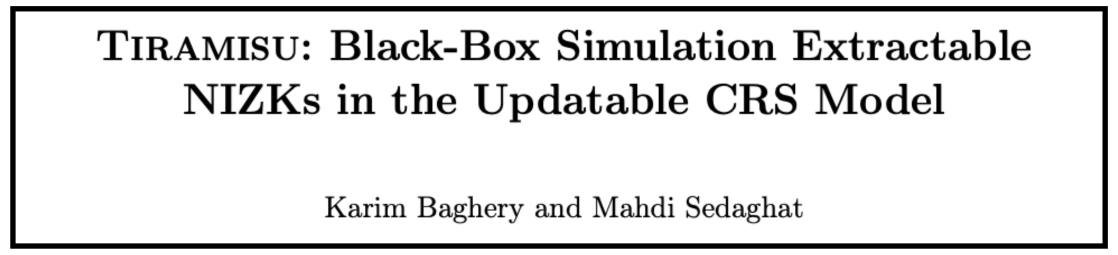
Non-Witness Succinct


CØCØ: A Framework for Building Composable Zero-Knowledge Proofs

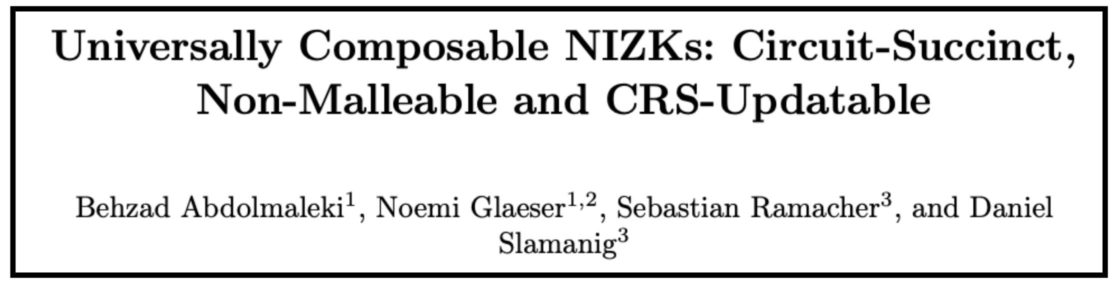
Ahmed Kosba[†] Zhichao Zhao[*] Andrew Miller[†] Yi Qian[‡]
T-H. Hubert Chan[*] Charalampos Papamanthou[†] Rafael Pass[‡] abhi shelat[•]
Elaine Shi[‡]


Lift-and-Shift: Obtaining Simulation Extractable Subversion and Updatable SNARKs Generically[⋆]

Behzad Abdolmaleki[1], Sebastian Ramacher[2], and Daniel Slamanig[2]


Tiramisu: Black-Box Simulation Extractable NIZKs in the Updatable CRS Model

Karim Baghery and Mahdi Sedaghat


Universally Composable NIZKs: Circuit-Succinct, Non-Malleable and CRS-Updatable

Behzad Abdolmaleki[1], Noemi Glaeser[1,2], Sebastian Ramacher[3], and Daniel Slamanig[3]

Encrypt witness

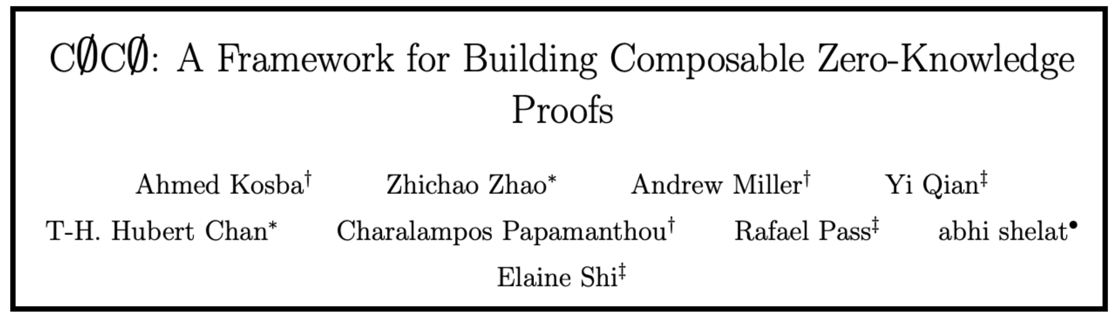
Compile Σ-protocol

Witness Succinct

Related works

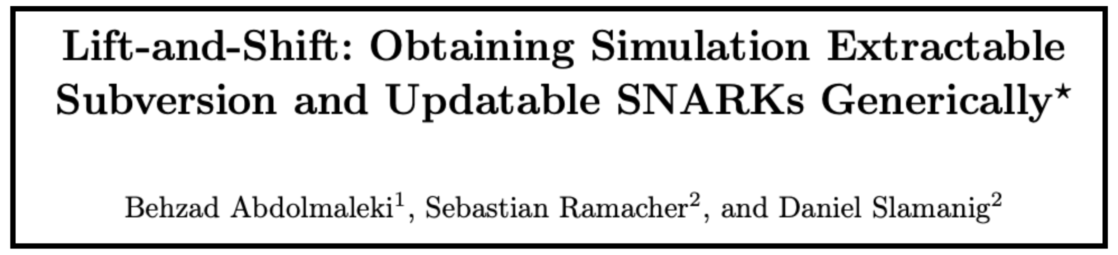
Known UC-secure zkSNARKs

Non-Witness Succinct



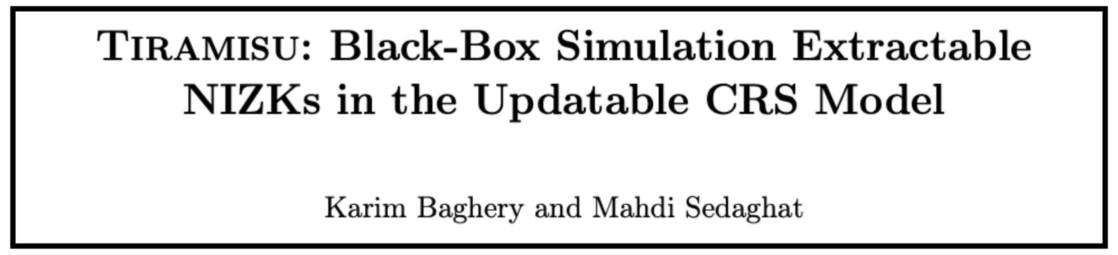
C∅C∅: A Framework for Building Composable Zero-Knowledge Proofs

Ahmed Kosba[†] Zhichao Zhao[*] Andrew Miller[†] Yi Qian[‡]
T-H. Hubert Chan[*] Charalampos Papamanthou[†] Rafael Pass[‡] abhi shelat[•]
Elaine Shi[‡]



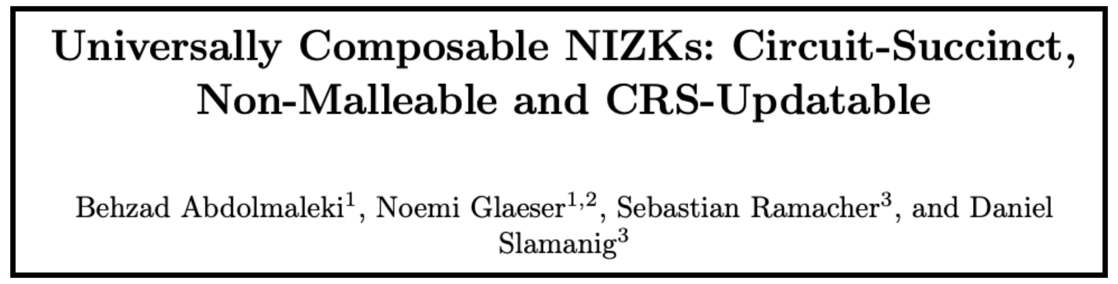
Lift-and-Shift: Obtaining Simulation Extractable Subversion and Updatable SNARKs Generically[⋆]

Behzad Abdolmaleki[1], Sebastian Ramacher[2], and Daniel Slamanig[2]



Tiramisu: Black-Box Simulation Extractable NIZKs in the Updatable CRS Model

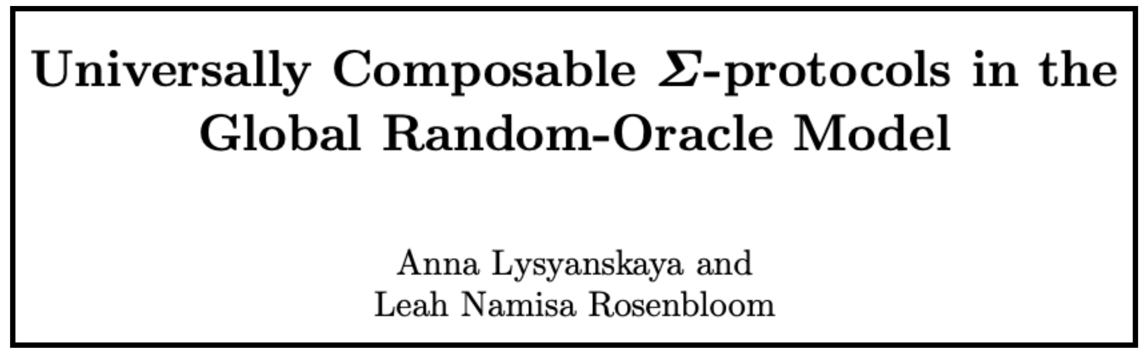
Karim Baghery and Mahdi Sedaghat



Universally Composable NIZKs: Circuit-Succinct, Non-Malleable and CRS-Updatable

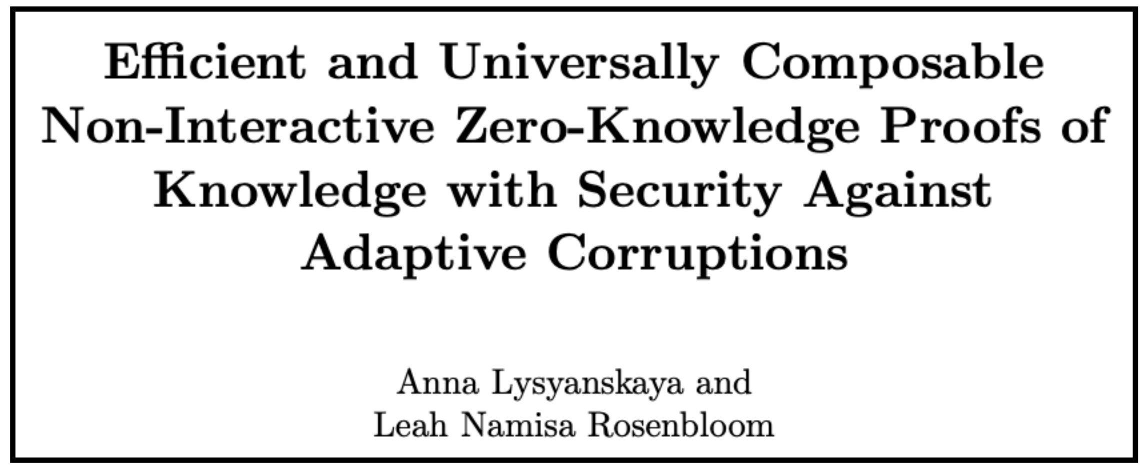
Behzad Abdolmaleki[1], Noemi Glaeser[1,2], Sebastian Ramacher[3], and Daniel Slamanig[3]

Encrypt witness



Universally Composable Σ-protocols in the Global Random-Oracle Model

Anna Lysyanskaya and
Leah Namisa Rosenbloom



Efficient and Universally Composable Non-Interactive Zero-Knowledge Proofs of Knowledge with Security Against Adaptive Corruptions

Anna Lysyanskaya and
Leah Namisa Rosenbloom

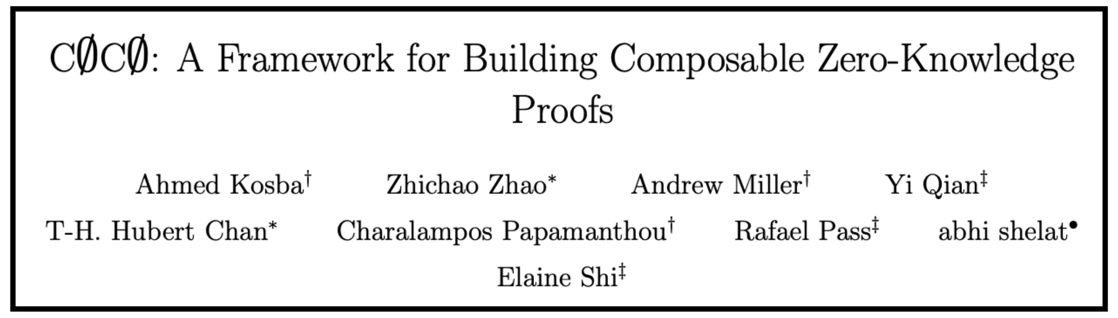
Compile Σ-protocol

Witness Succinct

Related works

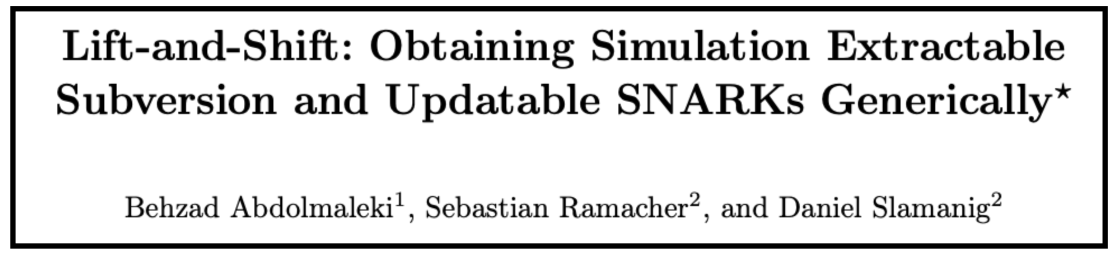
Known UC-secure zkSNARKs

Non-Witness Succinct



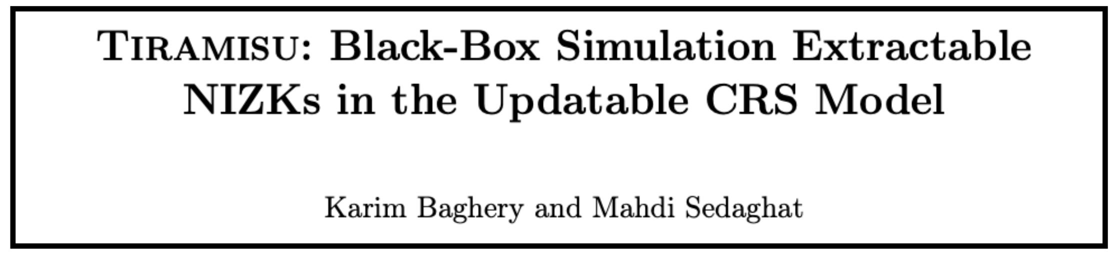
C∅C∅: A Framework for Building Composable Zero-Knowledge Proofs

Ahmed Kosba[†] Zhichao Zhao[*] Andrew Miller[†] Yi Qian[‡] T-H. Hubert Chan[*] Charalampos Papamanthou[†] Rafael Pass[‡] abhi shelat[•] Elaine Shi[‡]



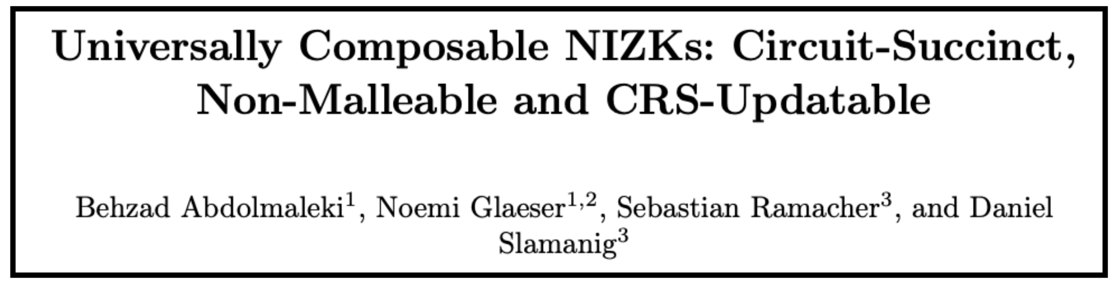
Lift-and-Shift: Obtaining Simulation Extractable Subversion and Updatable SNARKs Generically[⋆]

Behzad Abdolmaleki[1], Sebastian Ramacher[2], and Daniel Slamanig[2]



Tiramisu: Black-Box Simulation Extractable NIZKs in the Updatable CRS Model

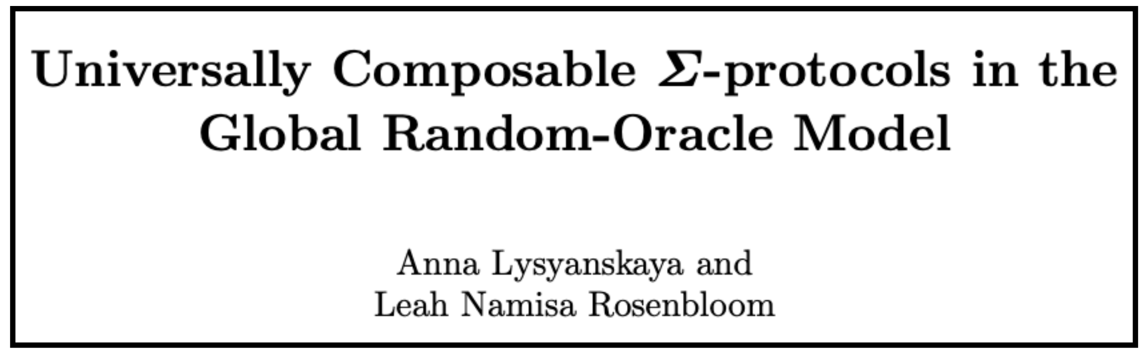
Karim Baghery and Mahdi Sedaghat



Universally Composable NIZKs: Circuit-Succinct, Non-Malleable and CRS-Updatable

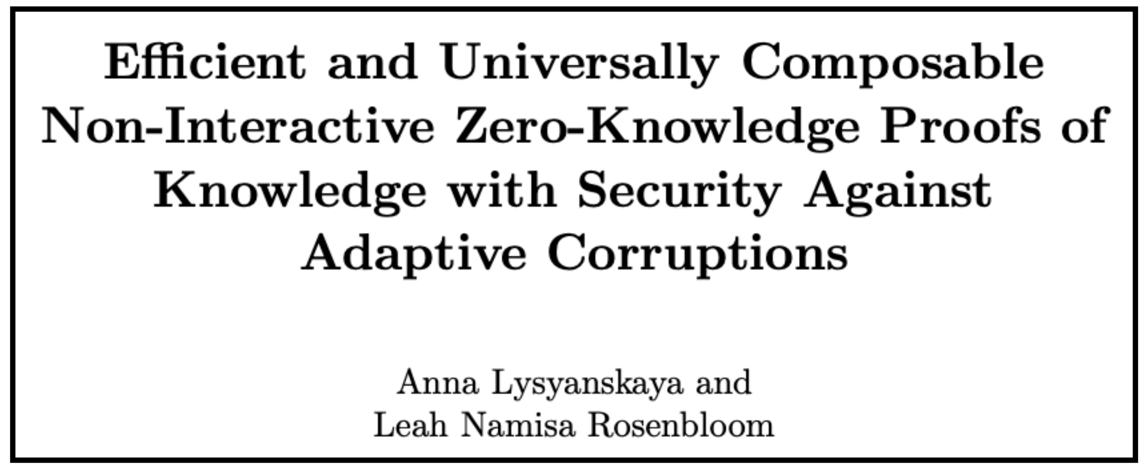
Behzad Abdolmaleki[1], Noemi Glaeser[1,2], Sebastian Ramacher[3], and Daniel Slamanig[3]

Encrypt witness



Universally Composable Σ-protocols in the Global Random-Oracle Model

Anna Lysyanskaya and Leah Namisa Rosenbloom

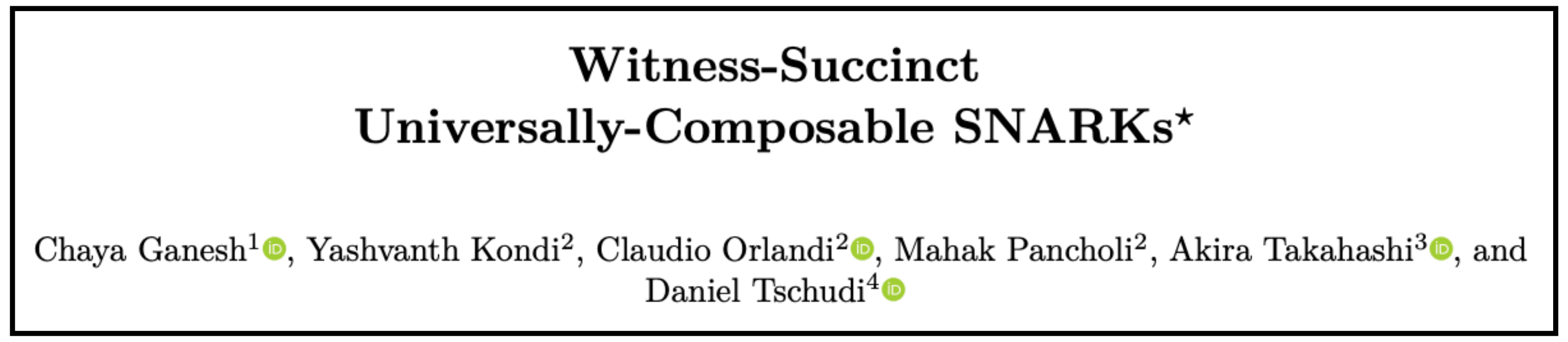


Efficient and Universally Composable Non-Interactive Zero-Knowledge Proofs of Knowledge with Security Against Adaptive Corruptions

Anna Lysyanskaya and Leah Namisa Rosenbloom

Compile Σ-protocol

Witness Succinct



Witness-Succinct Universally-Composable SNARKs[⋆]

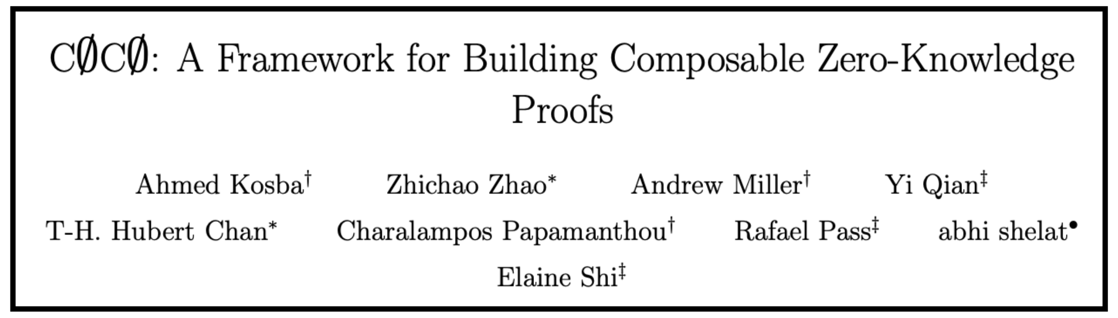
Chaya Ganesh[1], Yashvanth Kondi[2], Claudio Orlandi[2], Mahak Pancholi[2], Akira Takahashi[3], and Daniel Tschudi[4]

Commit witness using PCS

Related works

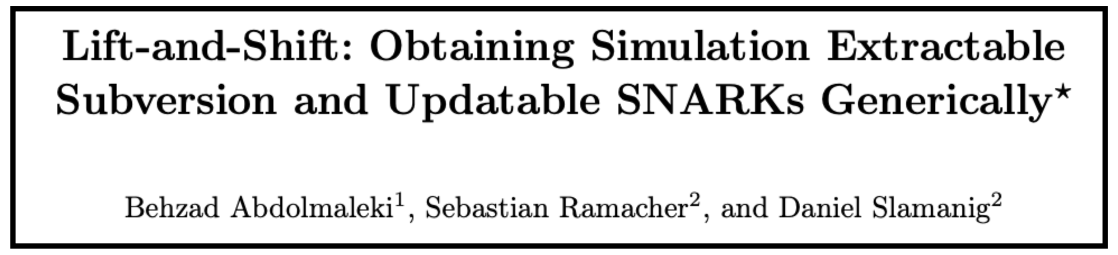
Known UC-secure zkSNARKs

Non-Witness Succinct



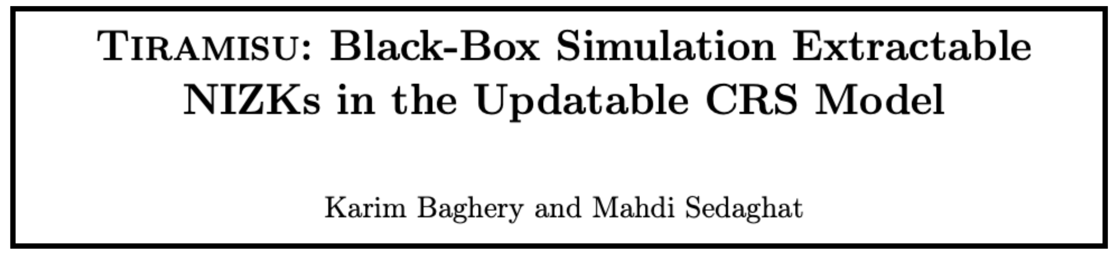
C∅C∅: A Framework for Building Composable Zero-Knowledge Proofs

Ahmed Kosba[†] Zhichao Zhao[*] Andrew Miller[†] Yi Qian[‡] T-H. Hubert Chan[*] Charalampos Papamanthou[†] Rafael Pass[‡] abhi shelat[•] Elaine Shi[‡]



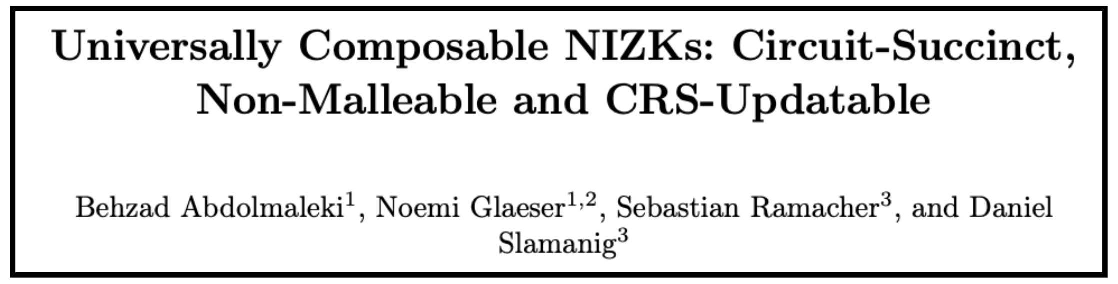
Lift-and-Shift: Obtaining Simulation Extractable Subversion and Updatable SNARKs Generically[⋆]

Behzad Abdolmaleki[1], Sebastian Ramacher[2], and Daniel Slamanig[2]



TIRAMISU: Black-Box Simulation Extractable NIZKs in the Updatable CRS Model

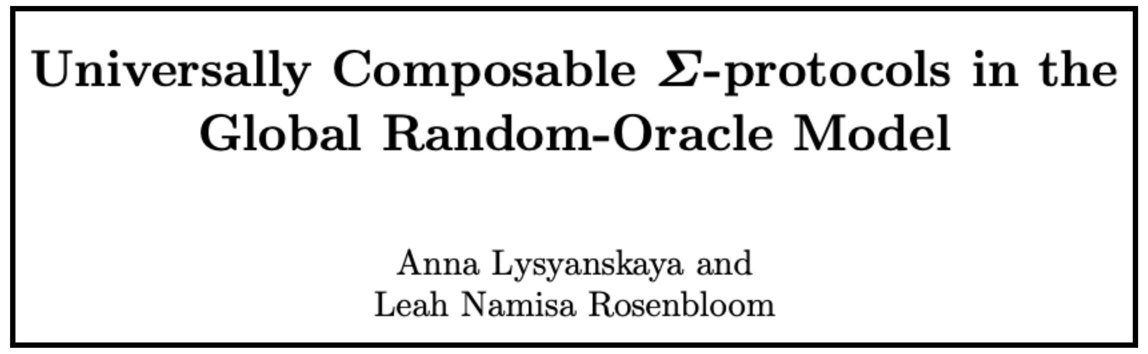
Karim Baghery and Mahdi Sedaghat



Universally Composable NIZKs: Circuit-Succinct, Non-Malleable and CRS-Updatable

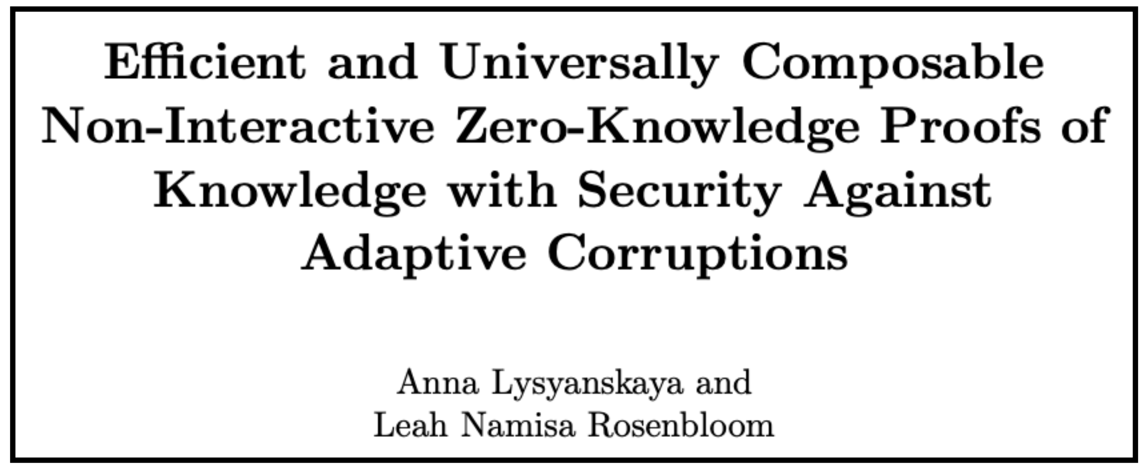
Behzad Abdolmaleki[1], Noemi Glaeser[1,2], Sebastian Ramacher[3], and Daniel Slamanig[3]

Encrypt witness



Universally Composable Σ-protocols in the Global Random-Oracle Model

Anna Lysyanskaya and
Leah Namisa Rosenbloom

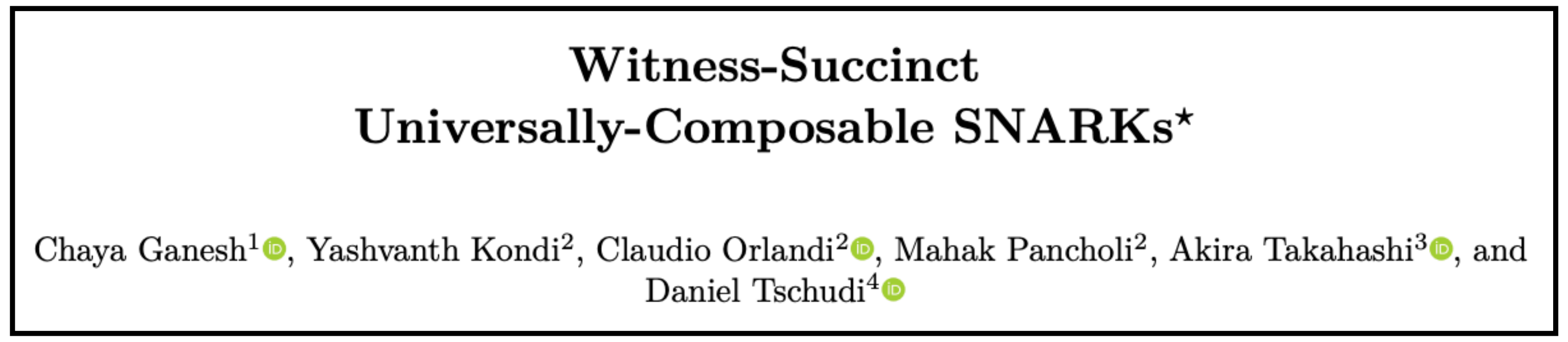


Efficient and Universally Composable Non-Interactive Zero-Knowledge Proofs of Knowledge with Security Against Adaptive Corruptions

Anna Lysyanskaya and
Leah Namisa Rosenbloom

Compile Σ-protocol

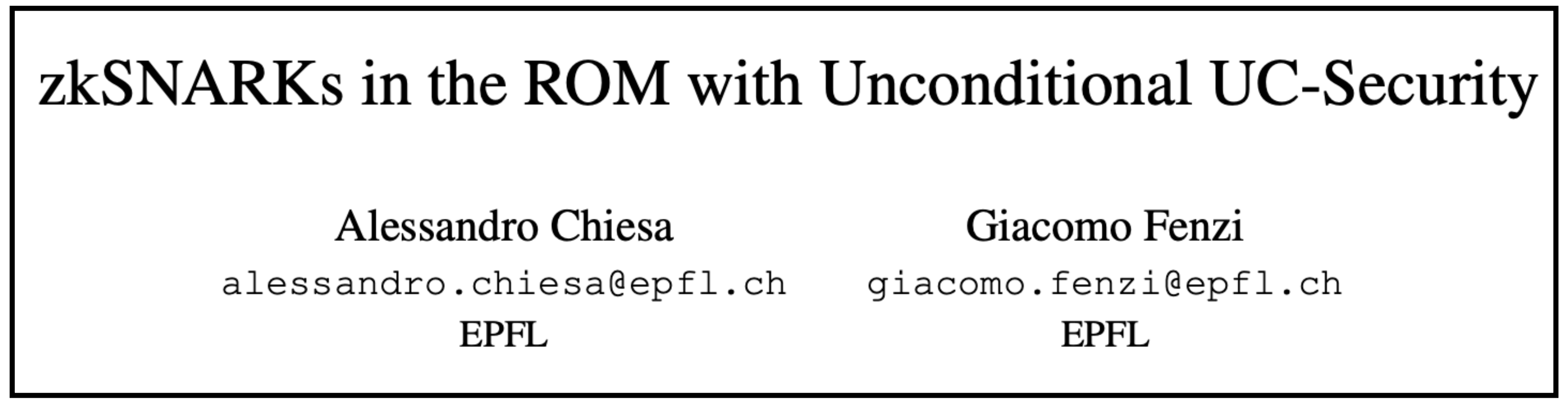
Witness Succinct



Witness-Succinct Universally-Composable SNARKs[⋆]

Chaya Ganesh[1], Yashvanth Kondi[2], Claudio Orlandi[2], Mahak Pancholi[2], Akira Takahashi[3], and Daniel Tschudi[4]

Commit witness using PCS



zkSNARKs in the ROM with Unconditional UC-Security

Alessandro Chiesa
alessandro.chiesa@epfl.ch
EPFL

Giacomo Fenzi
giacomo.fenzi@epfl.ch
EPFL

This work!

Challenge I

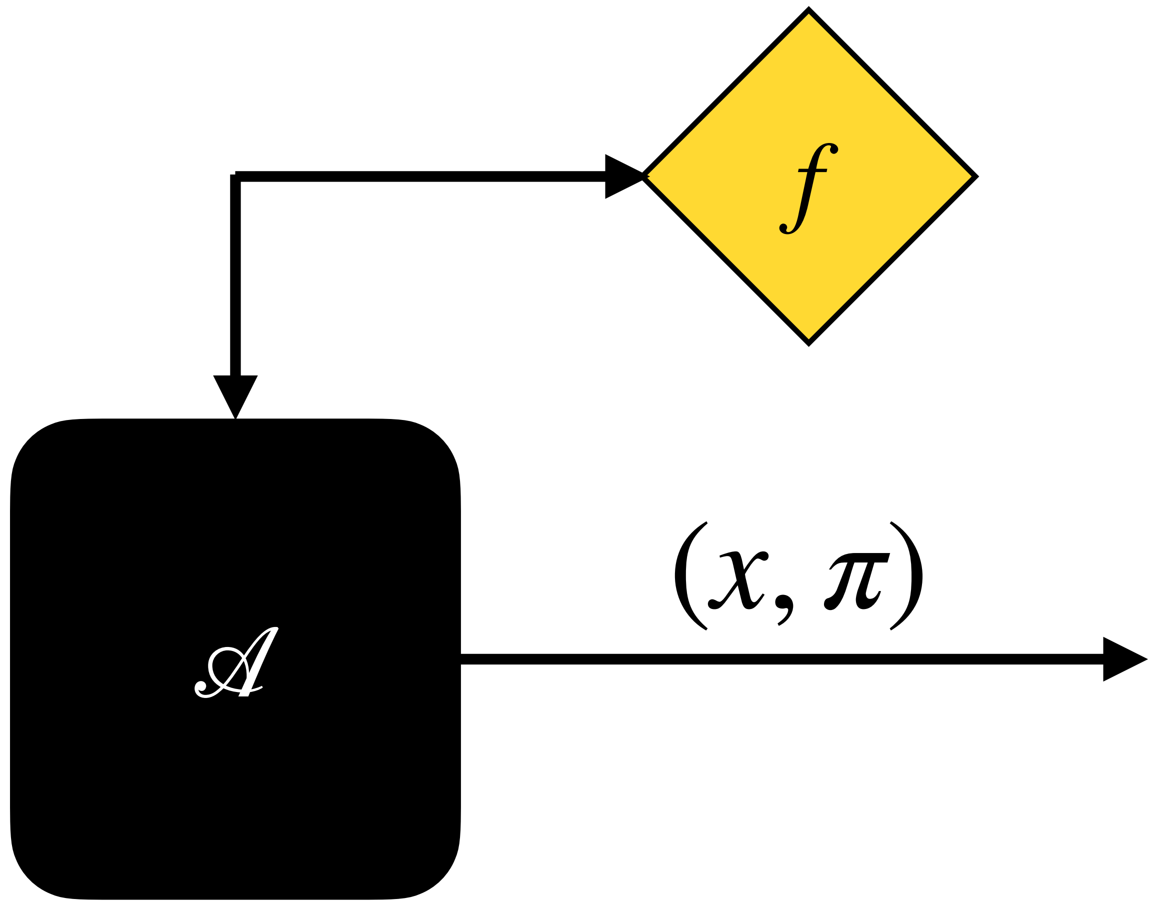
Challenge I

Rewinding

extractor

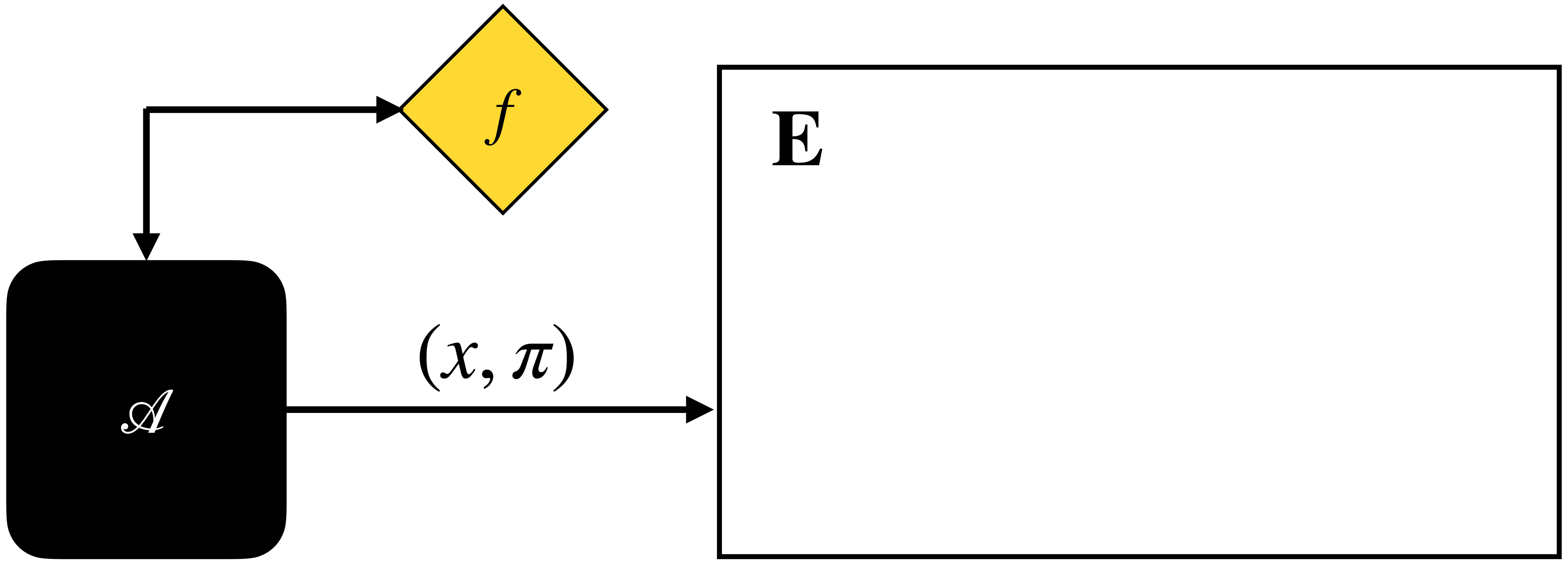
Challenge I

Rewinding extractor



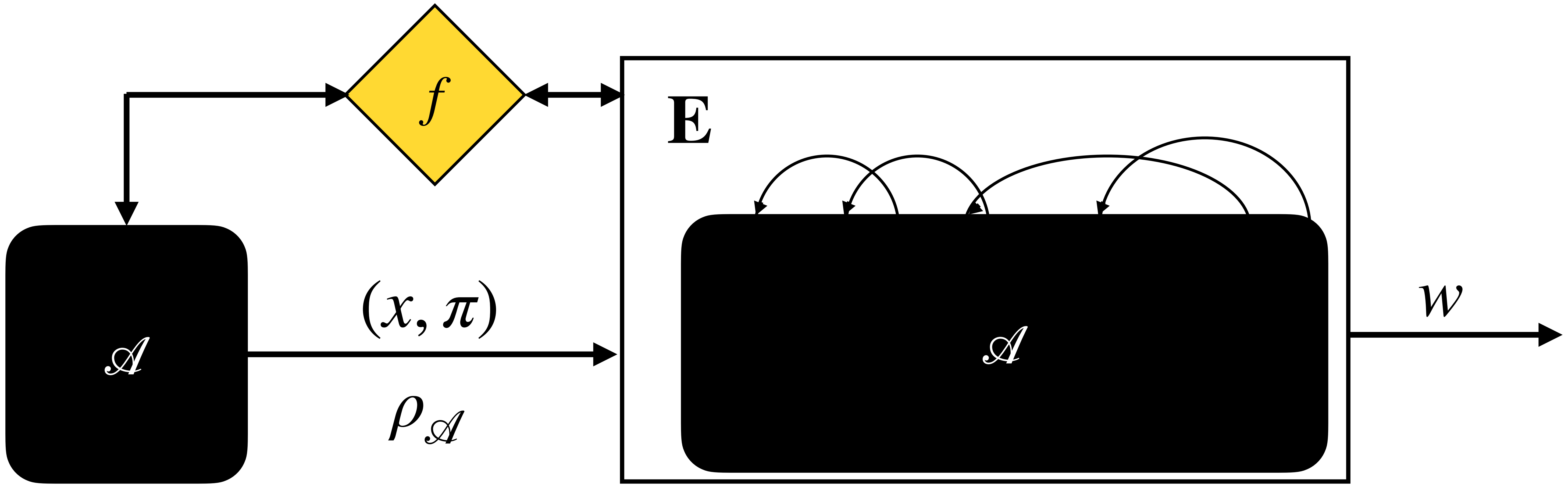
Challenge I

Rewinding extractor



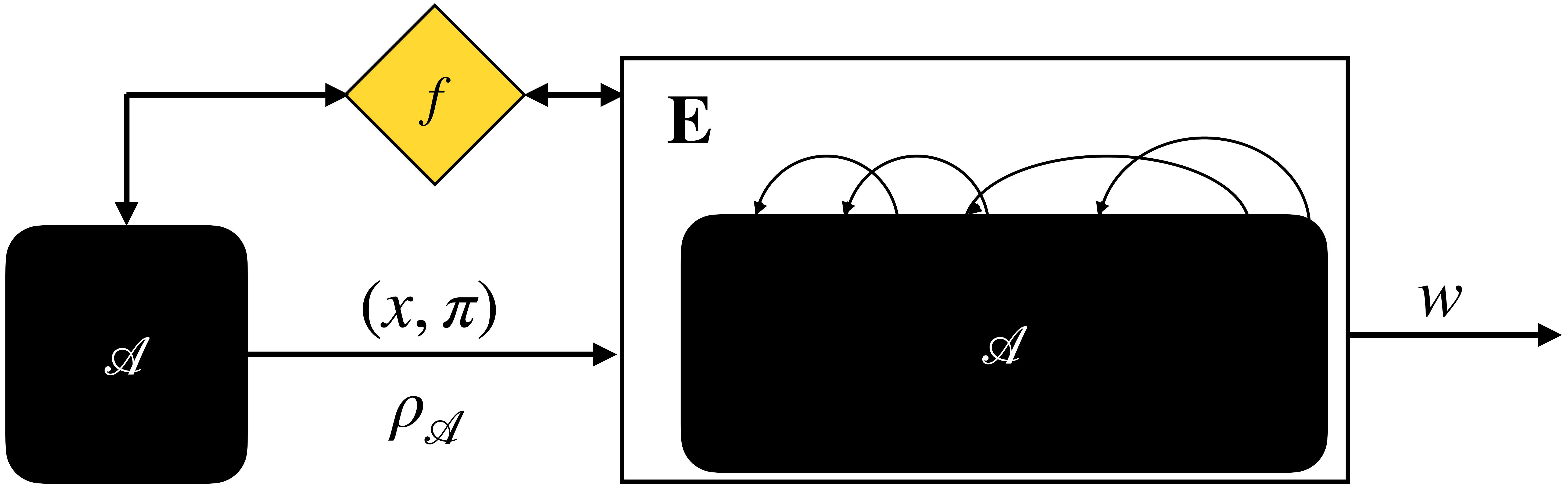
Challenge I

Rewinding extractor



Challenge I

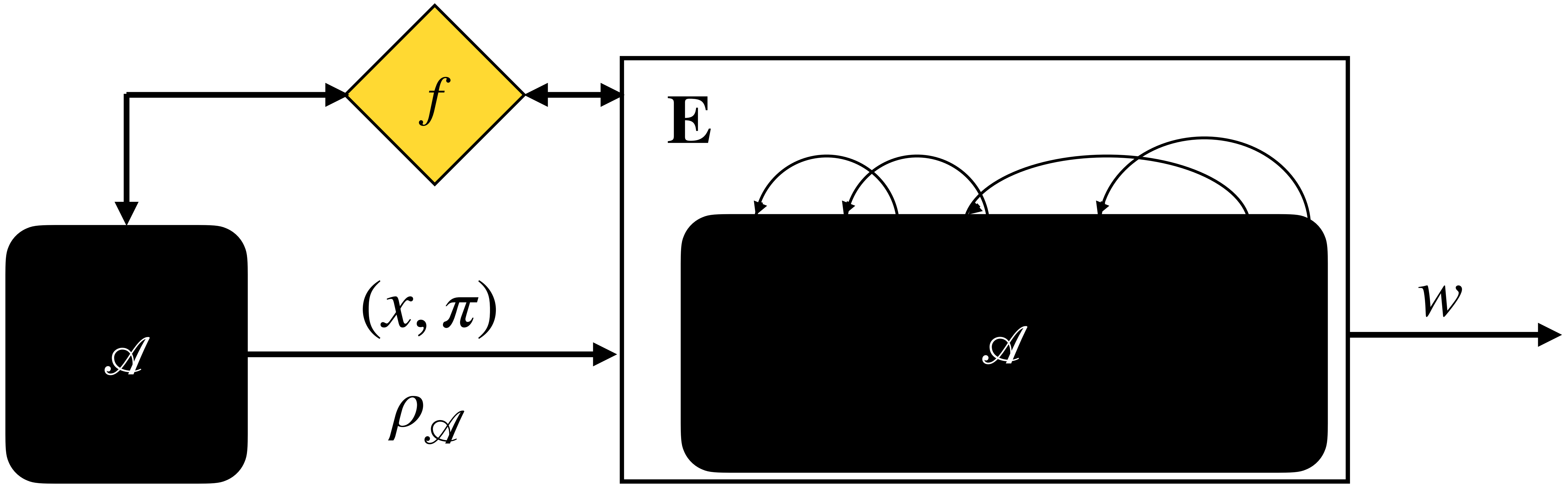
Rewinding extractor



Not allowed in UC!

Challenge I

Rewinding extractor

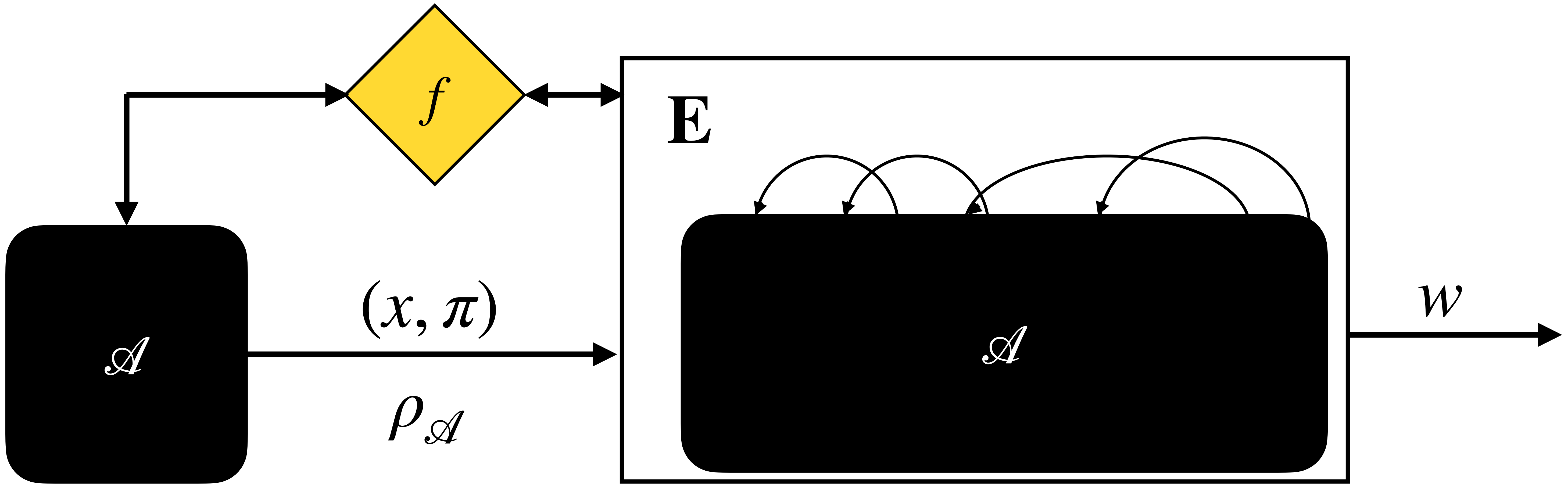


Not allowed in UC!

For UC-security, extractor must be **black-box** and **straight-line**, as we cannot rewind the environment, and security is $\exists \mathscr{S} \forall \mathscr{E}$

Challenge I

Rewinding extractor



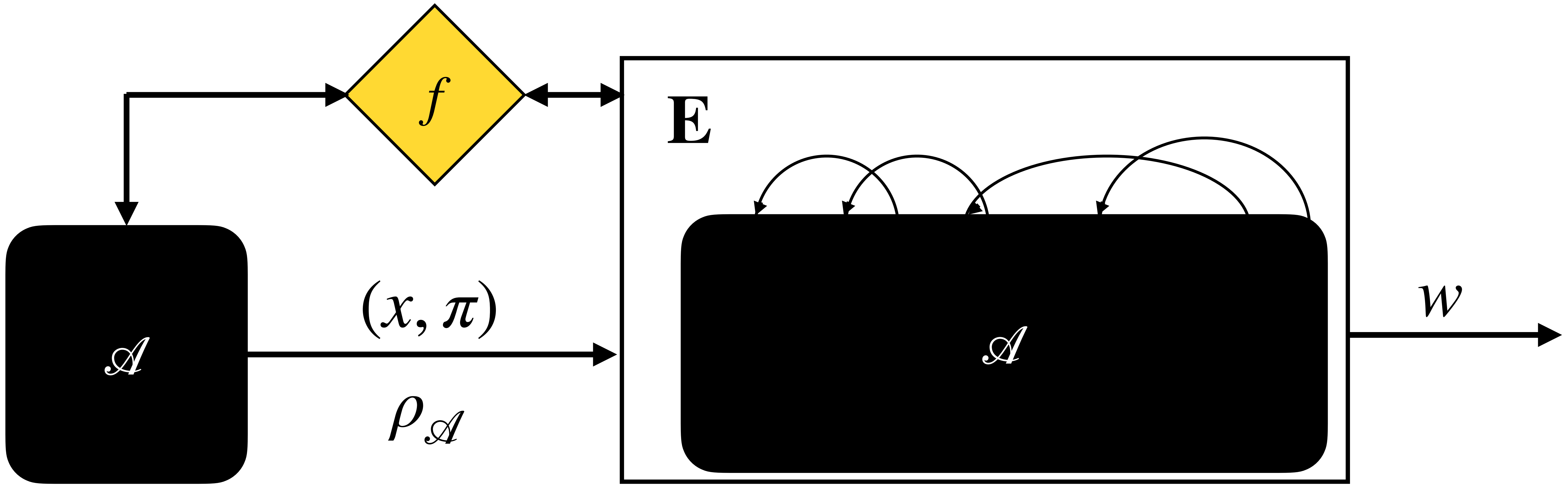
Not allowed in UC!

Straightline (black-box) extractor

For UC-security, extractor must be **black-box** and **straight-line**, as we cannot rewind the environment, and security is $\exists \mathscr{S} \forall \mathscr{E}$

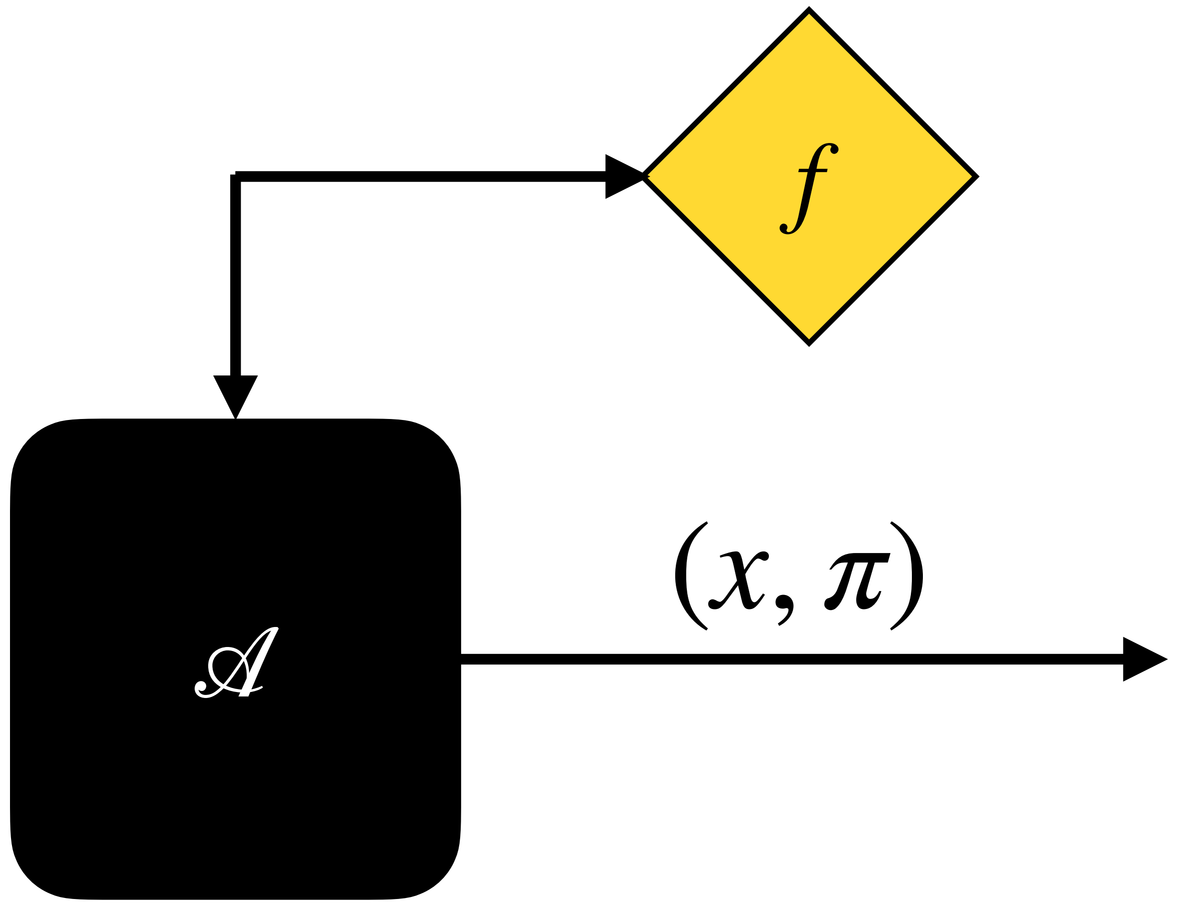
Challenge I

Rewinding extractor



Not allowed in UC!

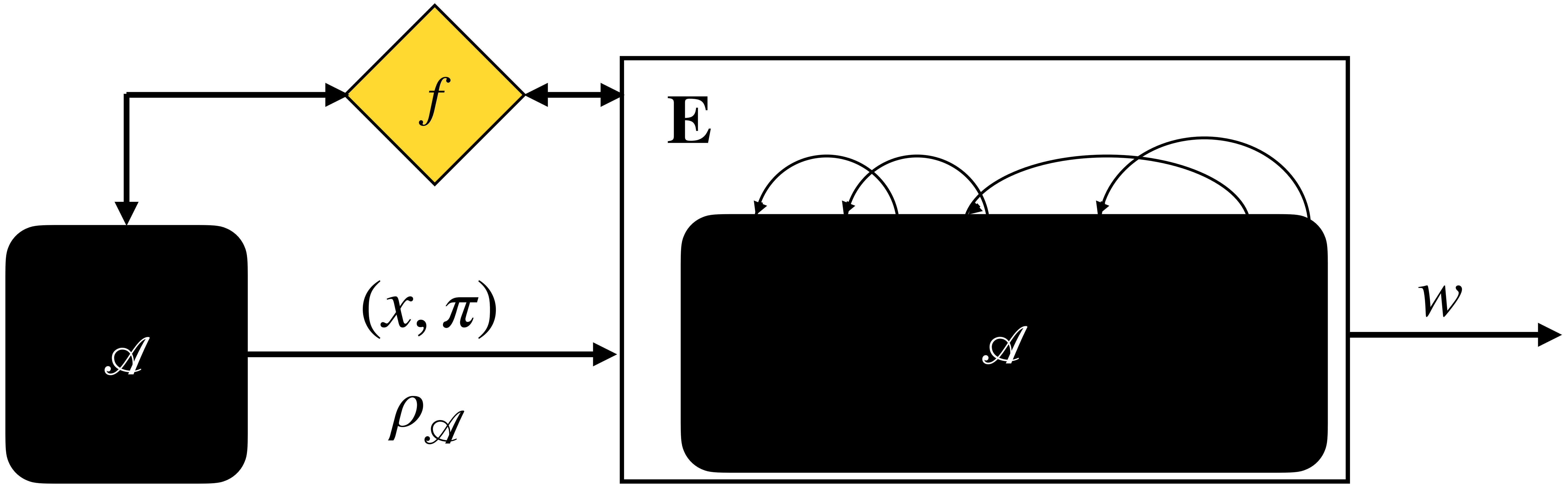
Straightline (black-box) extractor



For UC-security, extractor must be **black-box** and **straight-line**, as we cannot rewind the environment, and security is $\exists \mathscr{S} \forall \mathscr{E}$

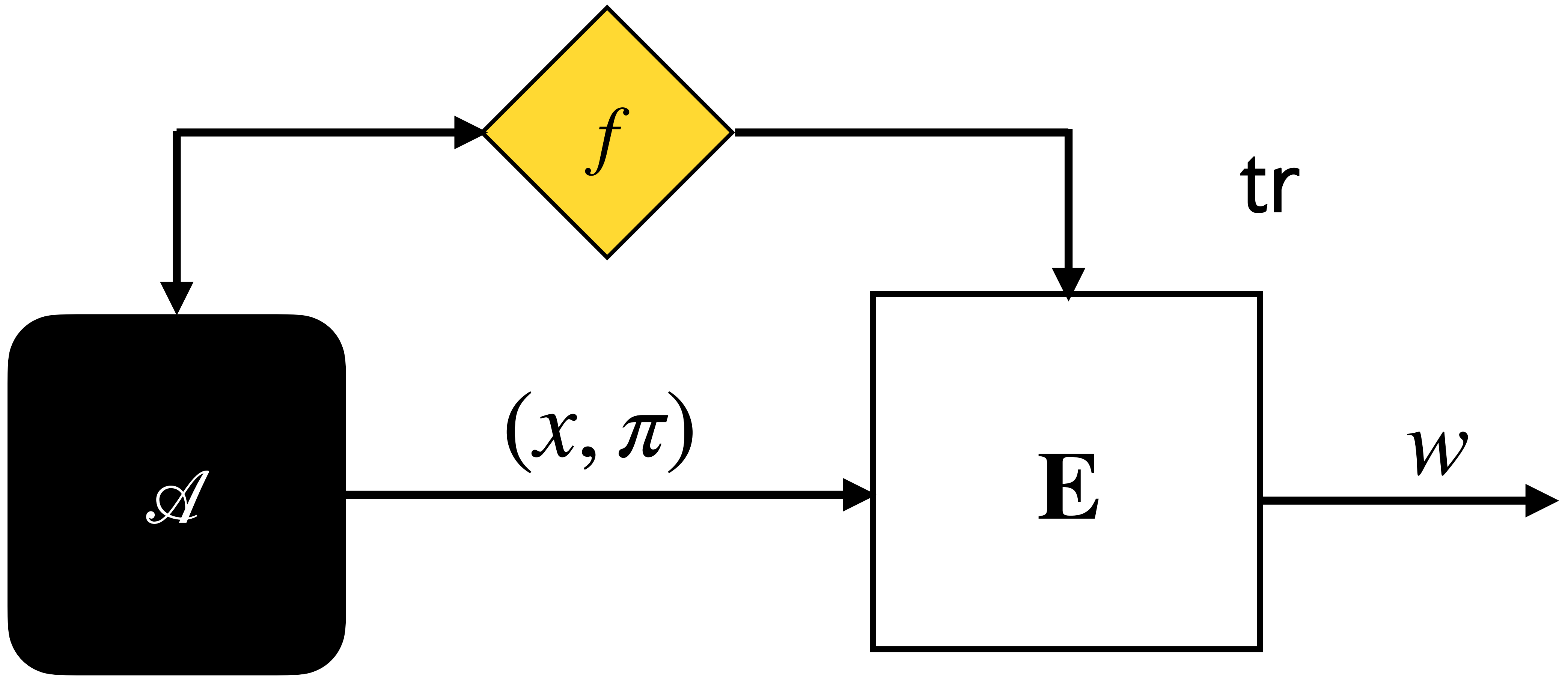
Challenge I

Rewinding extractor



Not allowed in UC!

Straightline (black-box) extractor



For UC-security, extractor must be **black-box** and **straight-line**, as we cannot rewind the environment, and security is $\exists \mathscr{S} \forall \mathscr{E}$

Challenge II

Challenge II

Our $\mathscr{F}_{\text{ARG}}$ gives access to simulated proofs.

Challenge II

Our $\mathscr{F}_{\mathrm{ARG}}$ gives access to simulated proofs.

Attack: The adversary could use them to “forge” new proofs.

Challenge II

Our $\mathscr{F}_{\mathrm{ARG}}$ gives access to simulated proofs.

Attack: The adversary could use them to “forge” new proofs.

Want: $\exists \mathbf{E}$ straightline s.t. $\forall \mathscr{A}$

Challenge II

Our $\mathscr{F}_{\mathrm{ARG}}$ gives access to simulated proofs.

Attack: The adversary could use them to “forge” new proofs.

Want: $\exists \mathbf{E}$ straightline s.t. $\forall \mathscr{A}$

$$\Pr\left[\quad \middle| \quad\right] \leq \kappa$$

Challenge II

Our $\mathscr{F}_{\mathrm{ARG}}$ gives access to simulated proofs.

Attack: The adversary could use them to “forge” new proofs.

Want: $\exists \mathbf{E}$ straightline s.t. $\forall \mathscr{A}$

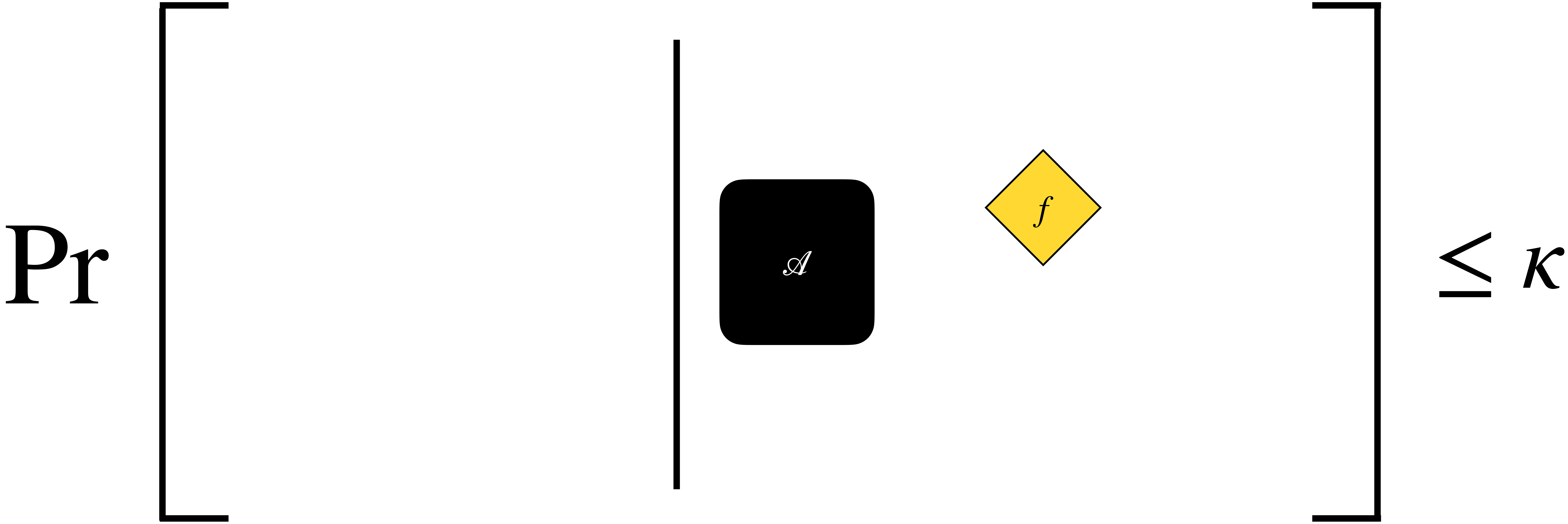
$$\Pr\left[\quad \middle| \; \boxed{\mathscr{A}} \quad \right] \leq \kappa$$

Challenge II

Our $\mathscr{F}_{\text{ARG}}$ gives access to simulated proofs.

Attack: The adversary could use them to “forge” new proofs.

Want: $\exists \mathbf{E}$ straightline s.t. $\forall \mathscr{A}$



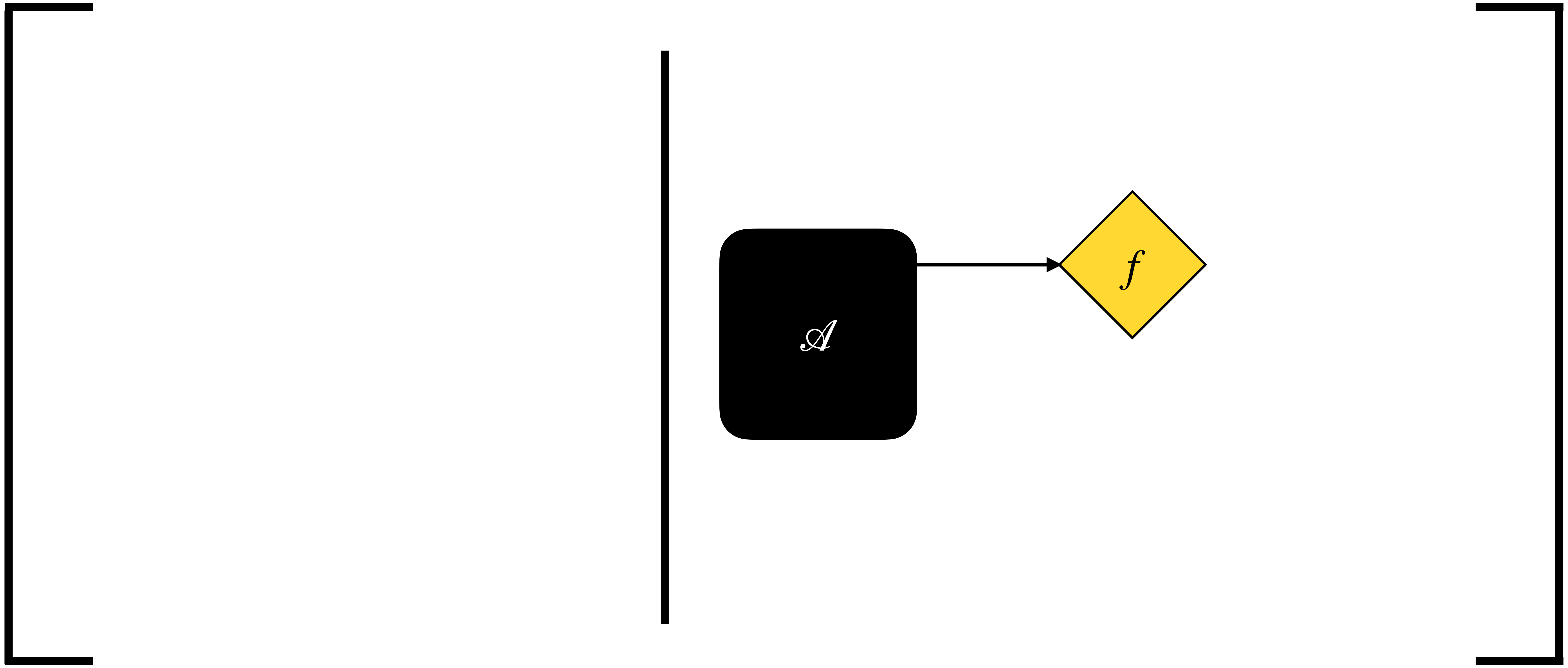
Challenge II

Our $\mathscr{F}_{\mathrm{ARG}}$ gives access to simulated proofs.

Attack: The adversary could use them to “forge” new proofs.

Want: $\exists \mathbf{E}$ straightline s.t. $\forall \mathscr{A}$

$\Pr$



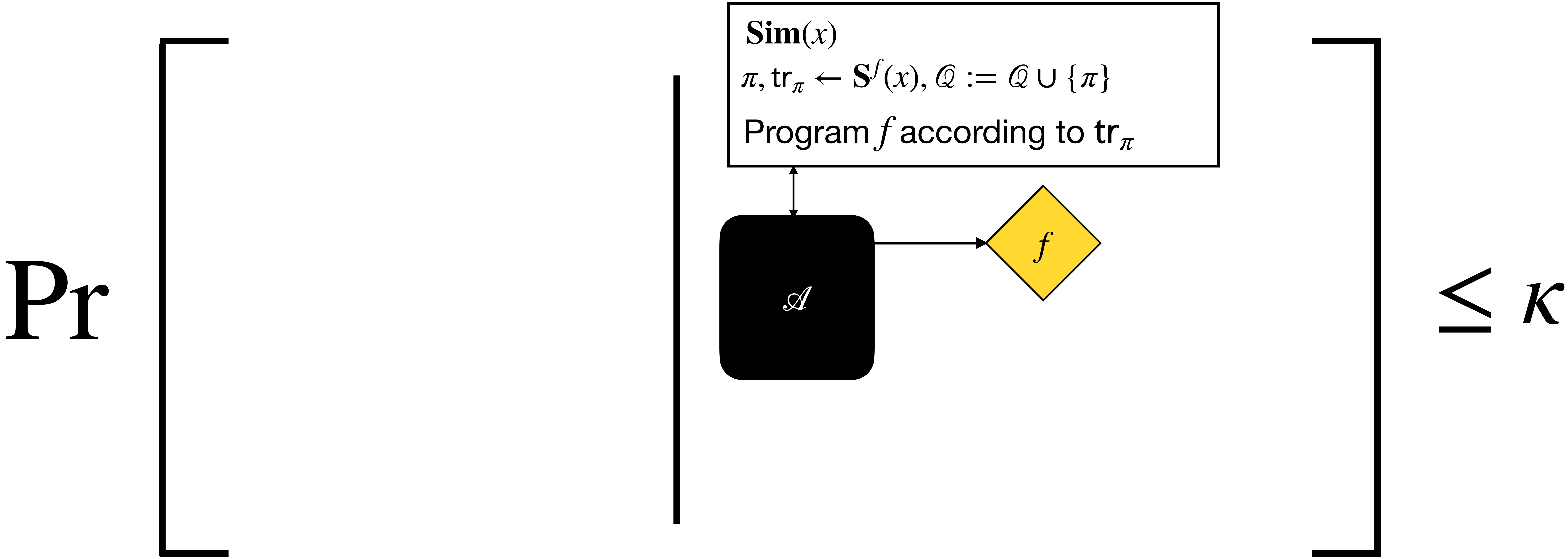
$\leq \kappa$

Challenge II

Our $\mathscr{F}_{\mathrm{ARG}}$ gives access to simulated proofs.

Attack: The adversary could use them to “forge” new proofs.

Want: $\exists \mathbf{E}$ straightline s.t. $\forall \mathscr{A}$



$\mathbf{Sim}(x)$

$\pi, \mathsf{tr}_\pi \leftarrow \mathbf{S}^f(x), \mathscr{Q} := \mathscr{Q} \cup \{\pi\}$

Program f according to tr_π

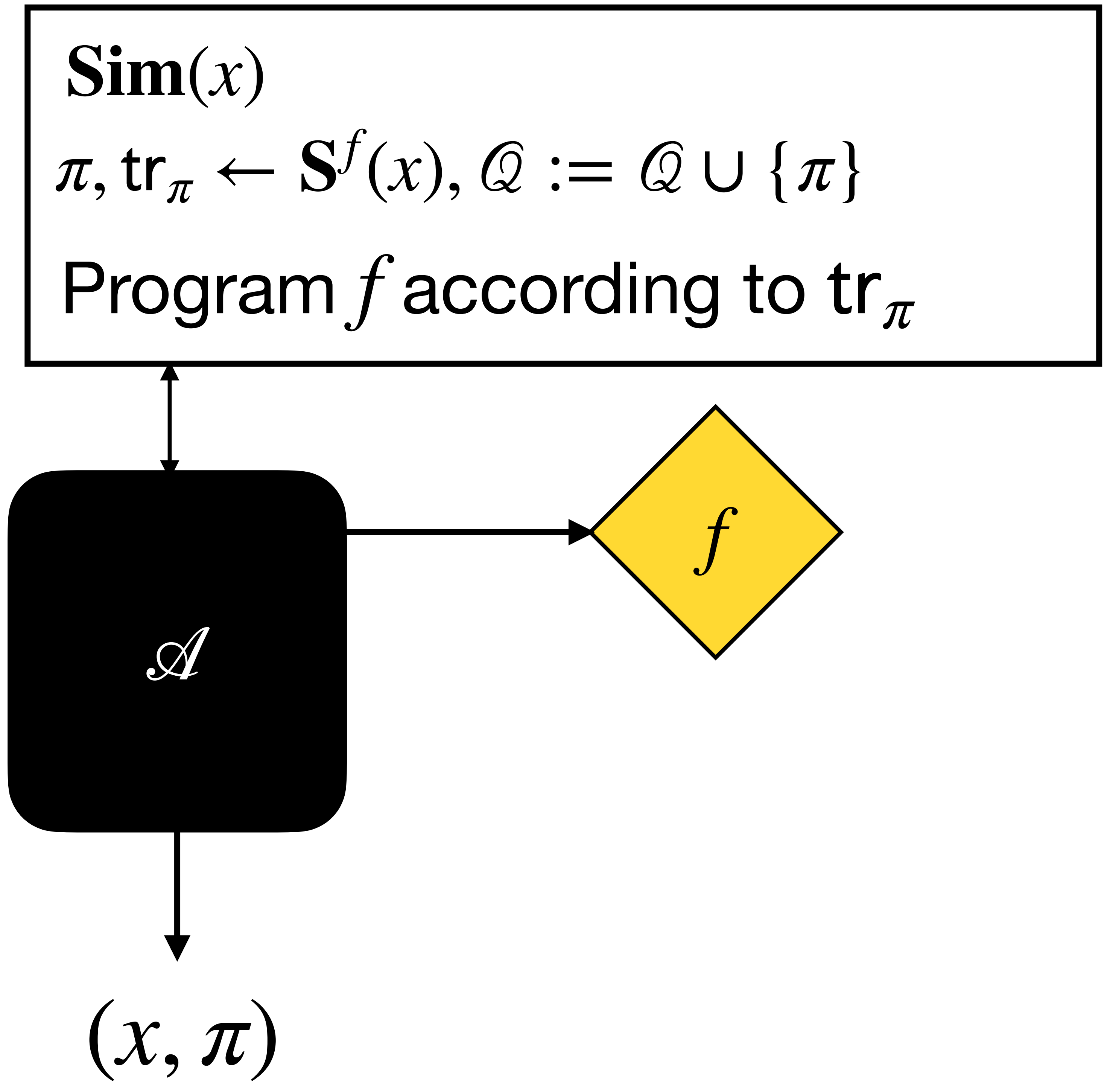
Challenge II

Our $\mathscr{F}_{\mathrm{ARG}}$ gives access to simulated proofs.

Attack: The adversary could use them to “forge” new proofs.

Want: $\exists \mathbf{E}$ straightline s.t. $\forall \mathscr{A}$

$$\Pr\left[\quad \middle| \quad \right] \leq \kappa$$



Sim(x)

$\pi, \mathsf{tr}_\pi \leftarrow \mathbf{S}^f(x)$, $\mathscr{Q} := \mathscr{Q} \cup \{\pi\}$

Program f according to tr_π

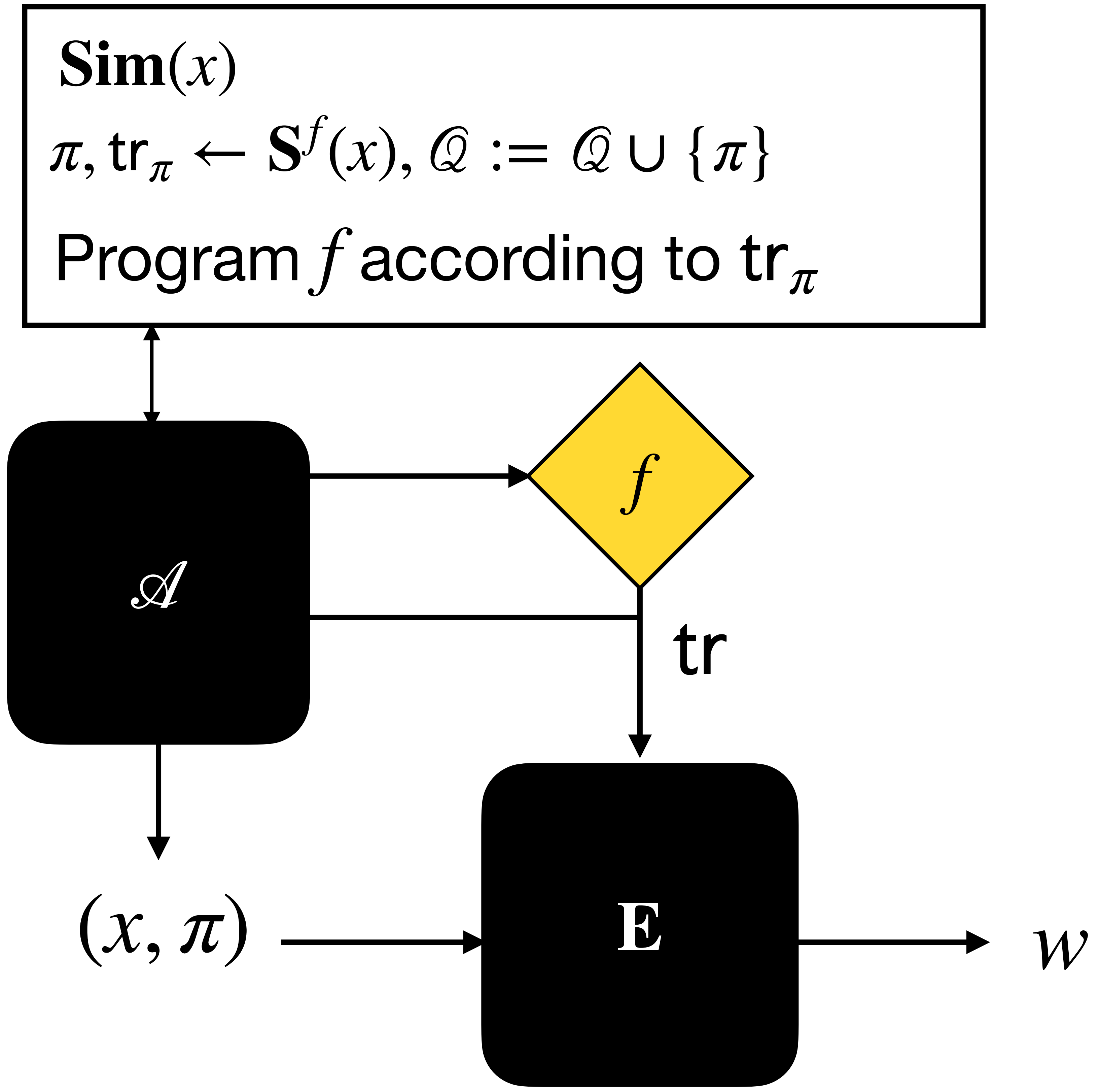
Challenge II

Our $\mathscr{F}_{\mathrm{ARG}}$ gives access to simulated proofs.

Attack: The adversary could use them to “forge” new proofs.

Want: $\exists \mathbf{E}$ straightline s.t. $\forall \mathscr{A}$

$$\Pr\left[\quad \middle| \quad \right] \leq \kappa$$



Sim(x)

$\pi, \mathsf{tr}_\pi \leftarrow \mathbf{S}^f(x), \mathscr{Q} := \mathscr{Q} \cup \{\pi\}$

Program f according to tr_π

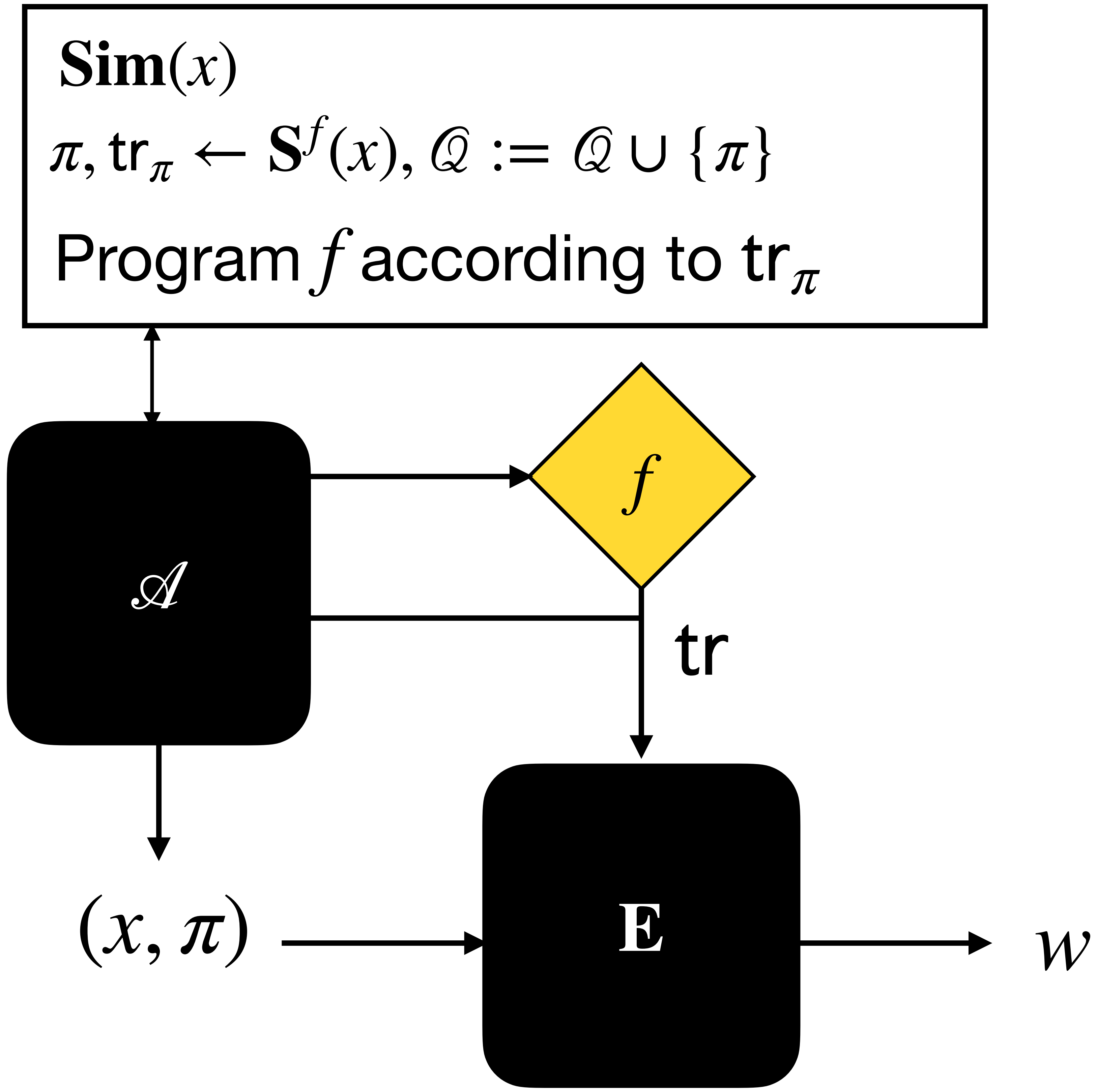
Challenge II

Our $\mathscr{F}_{\text{ARG}}$ gives access to simulated proofs.

Attack: The adversary could use them to “forge” new proofs.

Want: $\exists \mathbf{E}$ straightline s.t. $\forall \mathscr{A}$

$$\Pr\left[\begin{array}{c} \mathbf{V}^f(x,\pi) = 1 \\ (x,w) \notin R \\ \pi \notin \mathscr{Q} \end{array}\ \middle|\right.$$



$\mathbf{Sim}(x)$

$\pi, \mathsf{tr}_\pi \leftarrow \mathbf{S}^f(x)$, $\mathscr{Q} := \mathscr{Q} \cup \{\pi\}$

Program f according to tr_π

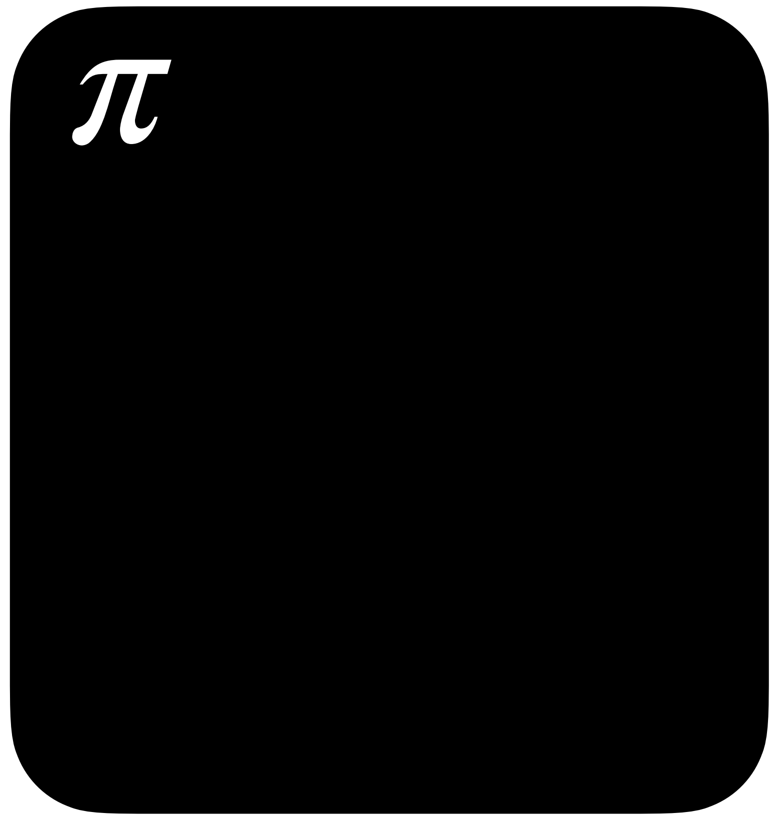
$$\left.\vphantom{\begin{array}{c}a\\b\\c\end{array}}\right] \leq \kappa$$

UC with Budgets

Plain UC only models adversaries that are **computationally** bounded

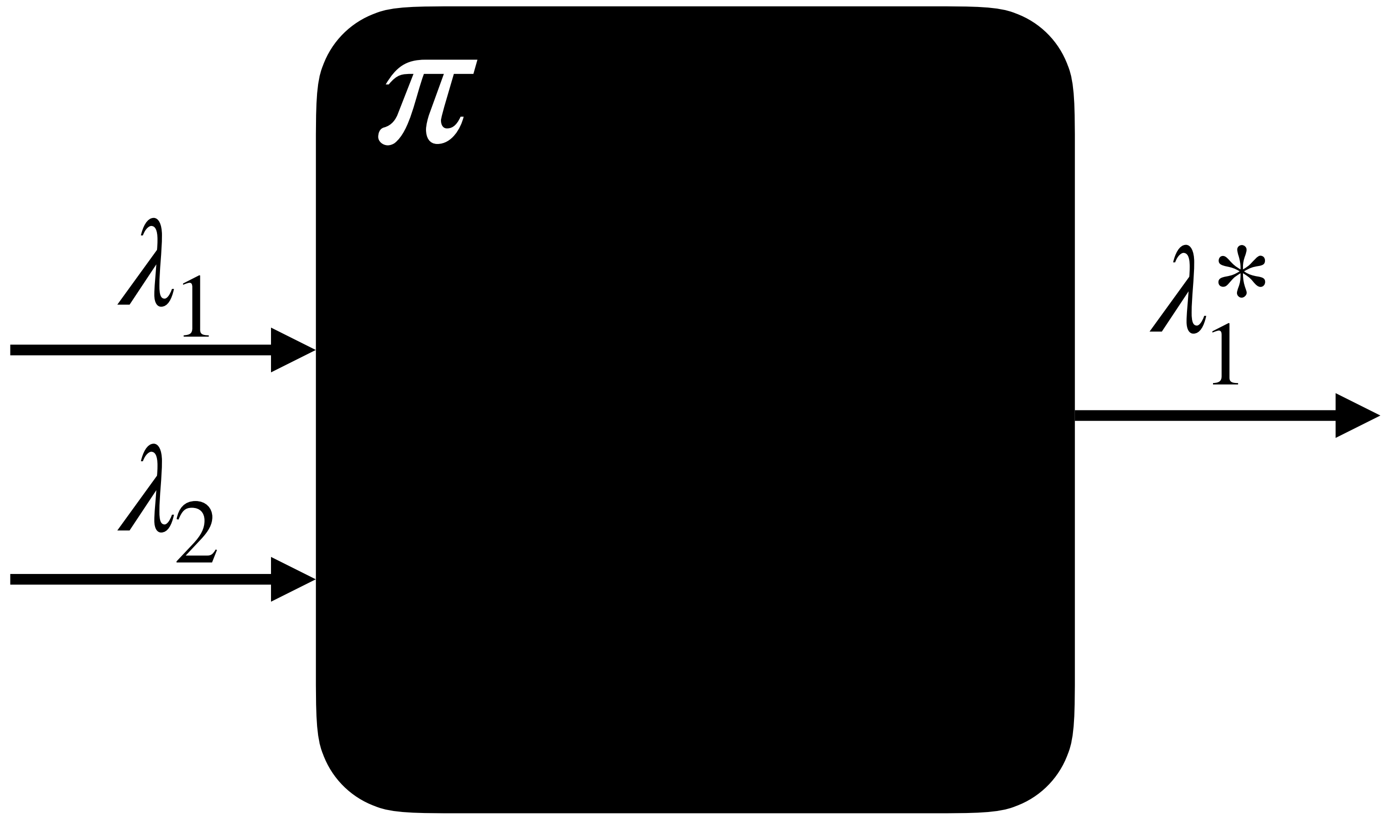
UC with Budgets

Plain UC only models adversaries that are **computationally** bounded



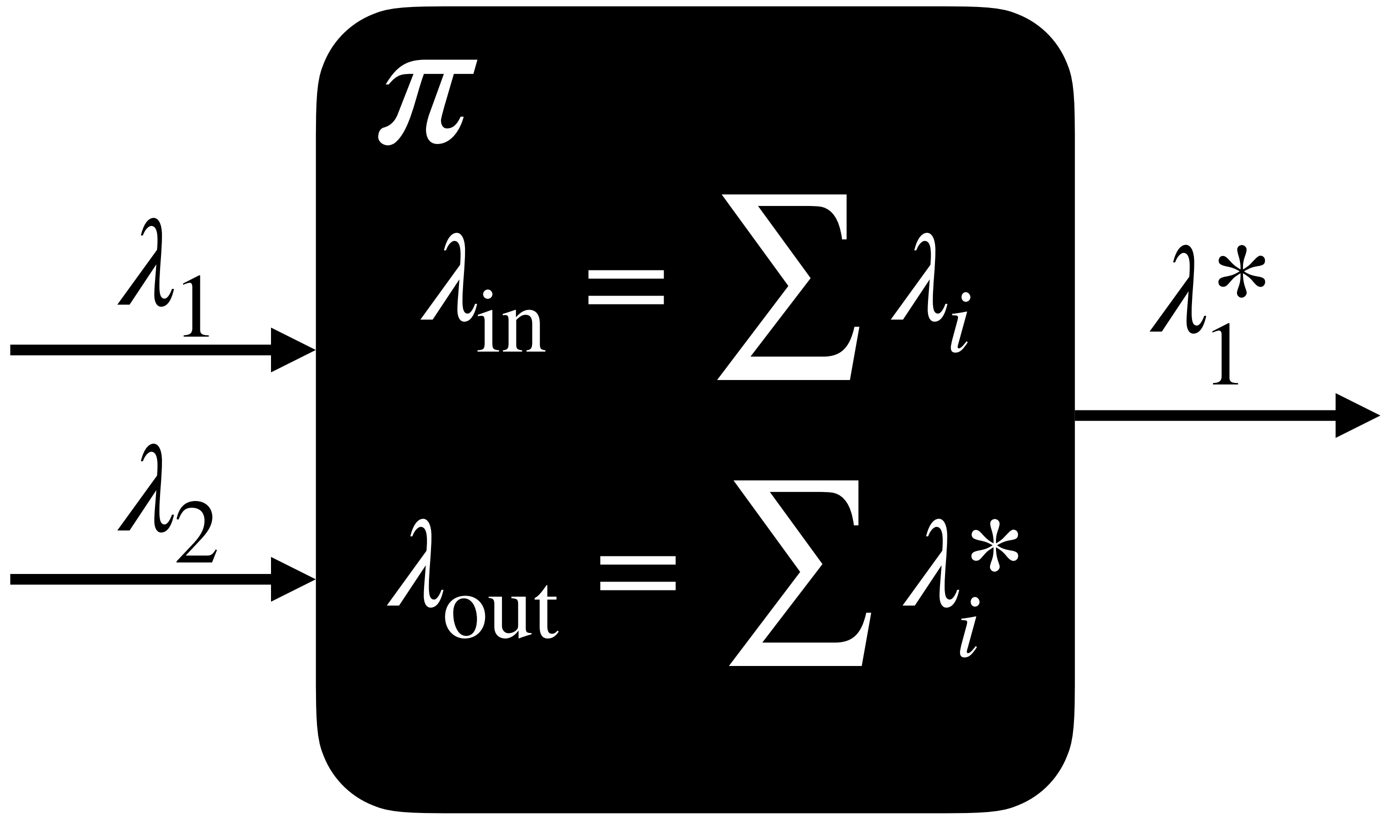
UC with Budgets

Plain UC only models adversaries that are **computationally** bounded



UC with Budgets

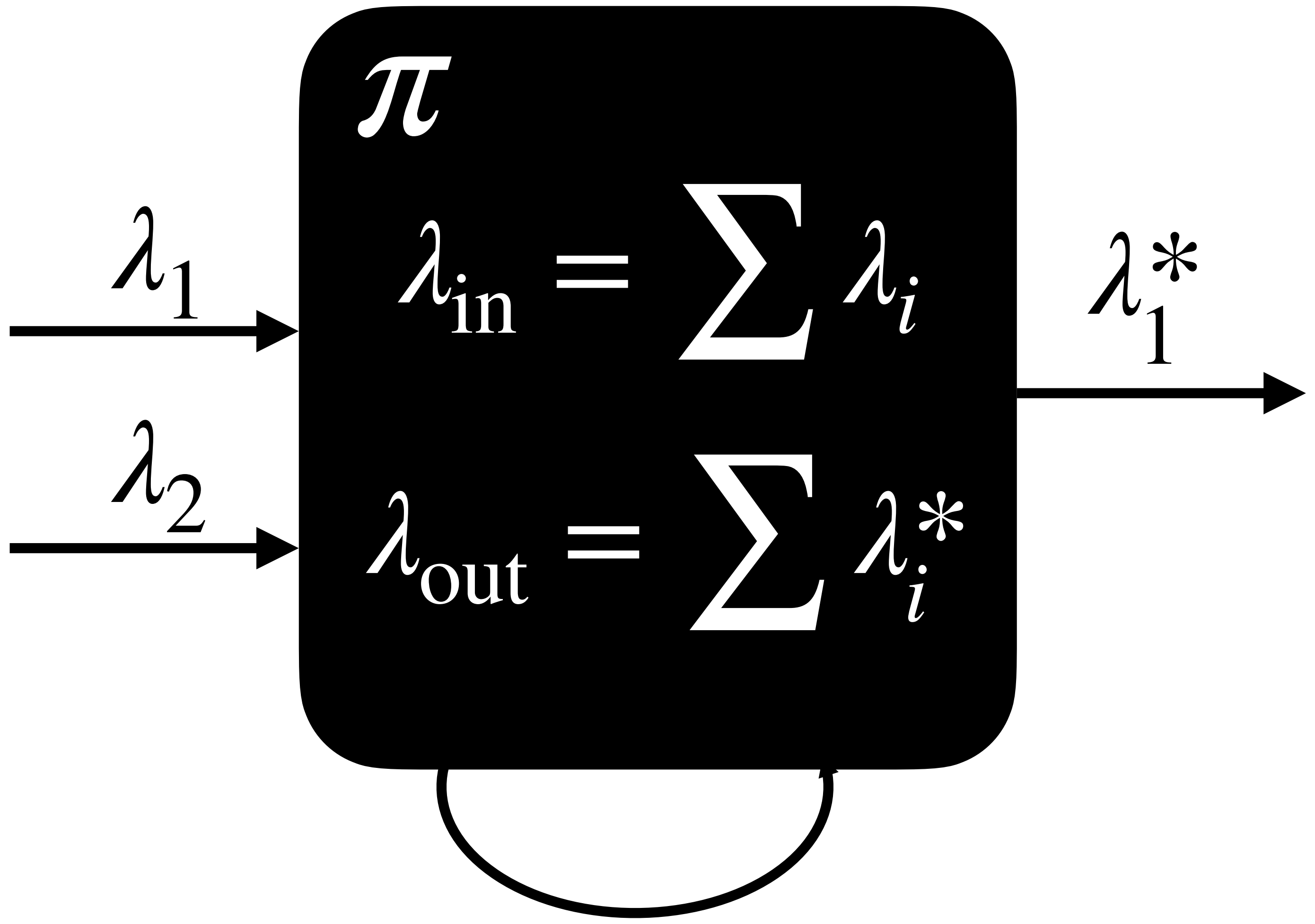
Plain UC only models adversaries that are **computationally** bounded



π
λ_1
λ_2
$\lambda_{\text{in}} = \sum \lambda_i$
$\lambda_{\text{out}} = \sum \lambda_i^*$
λ_1^*

UC with Budgets

Plain UC only models adversaries that are **computationally** bounded

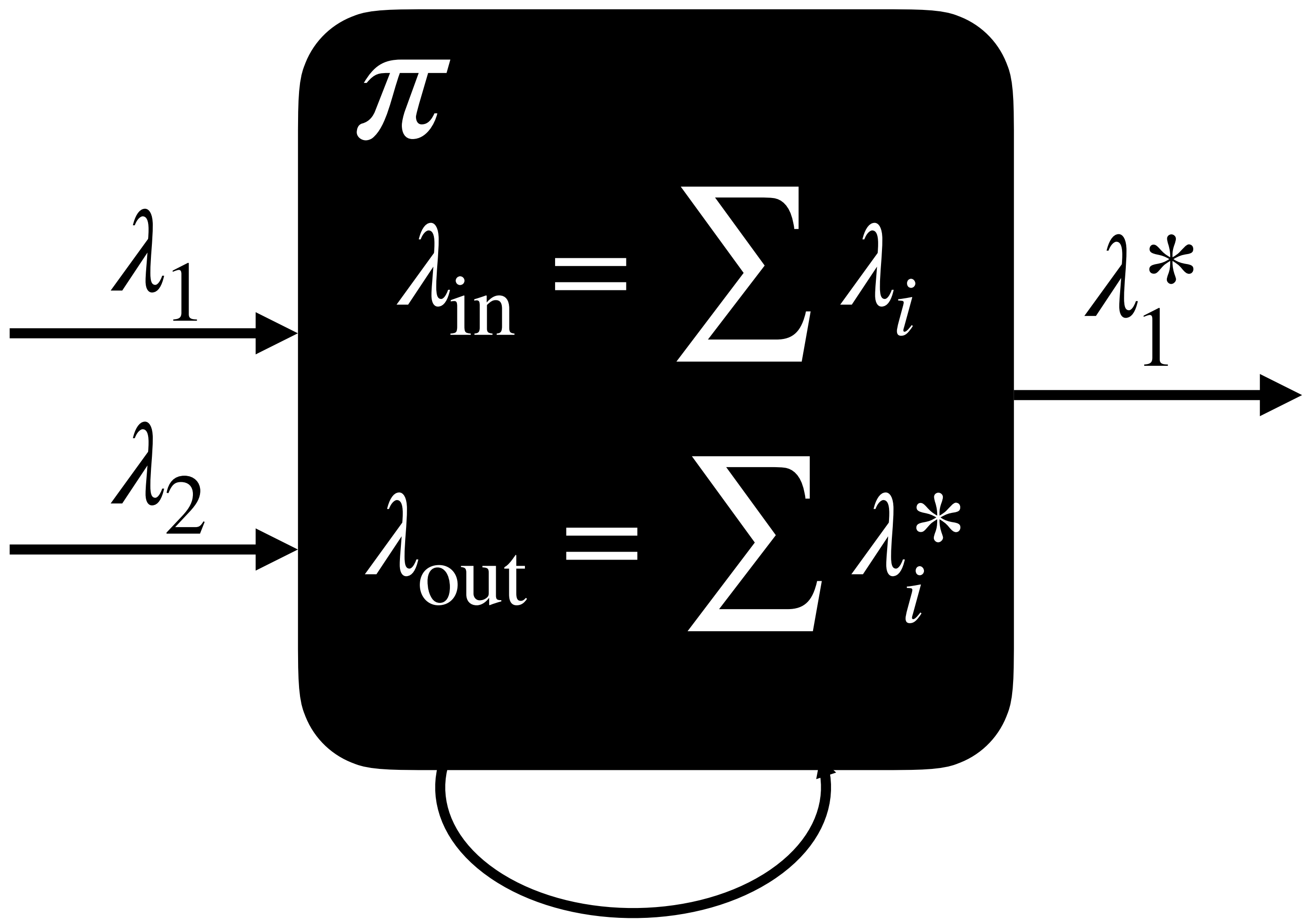

π
λ_1
λ_2
$\lambda_{\text{in}} = \sum \lambda_i$
$\lambda_{\text{out}} = \sum \lambda_i^*$
λ_1^*

$\text{time}(\pi) \leq p(\lambda_{\text{in}} - \lambda_{\text{out}})$

UC with Budgets

We consider adversaries that are **resource** bounded and computationally **unbounded**

Plain UC only models adversaries that are **computationally** bounded

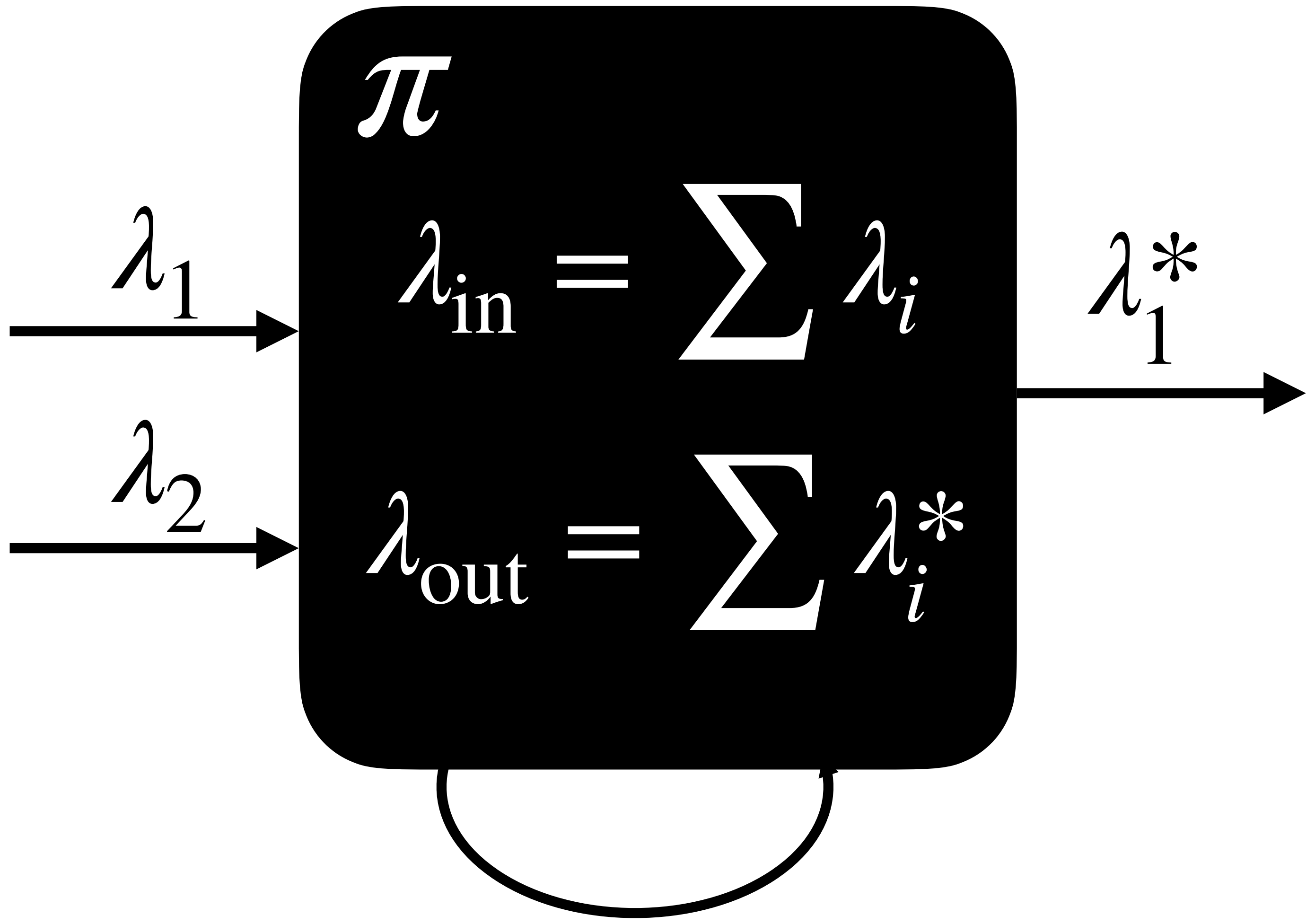

π
λ_1
λ_2
$\lambda_{\text{in}} = \sum \lambda_i$
$\lambda_{\text{out}} = \sum \lambda_i^*$
λ_1^*

time$(\pi) \leq p(\lambda_{\text{in}} - \lambda_{\text{out}})$

UC with Budgets

We consider adversaries that are **resource** bounded and computationally **unbounded**

Plain UC only models adversaries that are **computationally** bounded



$$\lambda_{\text{in}} = \sum \lambda_i$$

$$\lambda_{\text{out}} = \sum \lambda_i^*$$

$$\text{time}(\pi) \leq p(\lambda_{\text{in}} - \lambda_{\text{out}})$$

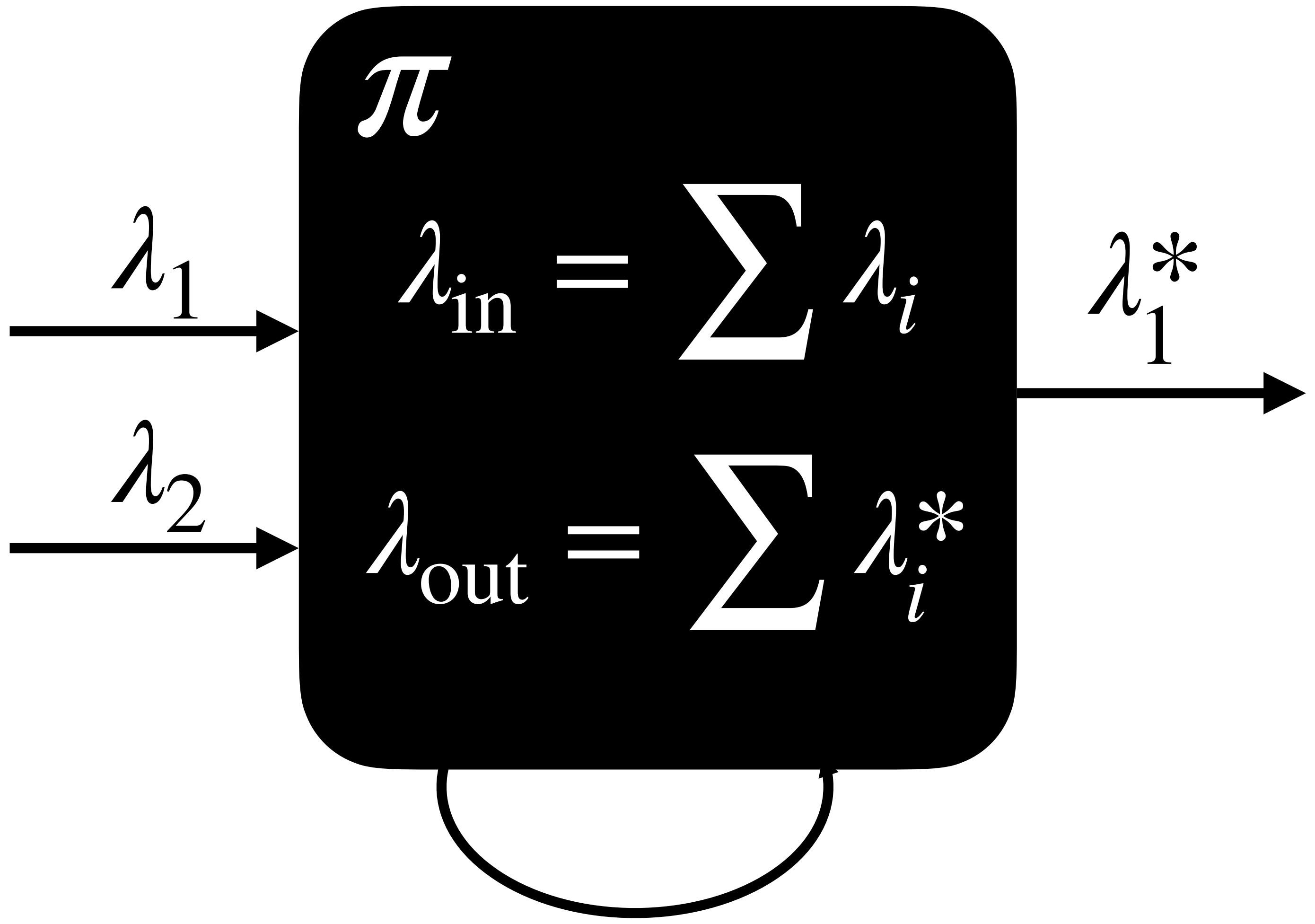
π

$$\mathscr{B} = \mathscr{B}_{\text{start}} + \sum \mathscr{B}_{\text{in}}$$

$$\mathscr{B} = (t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}})$$

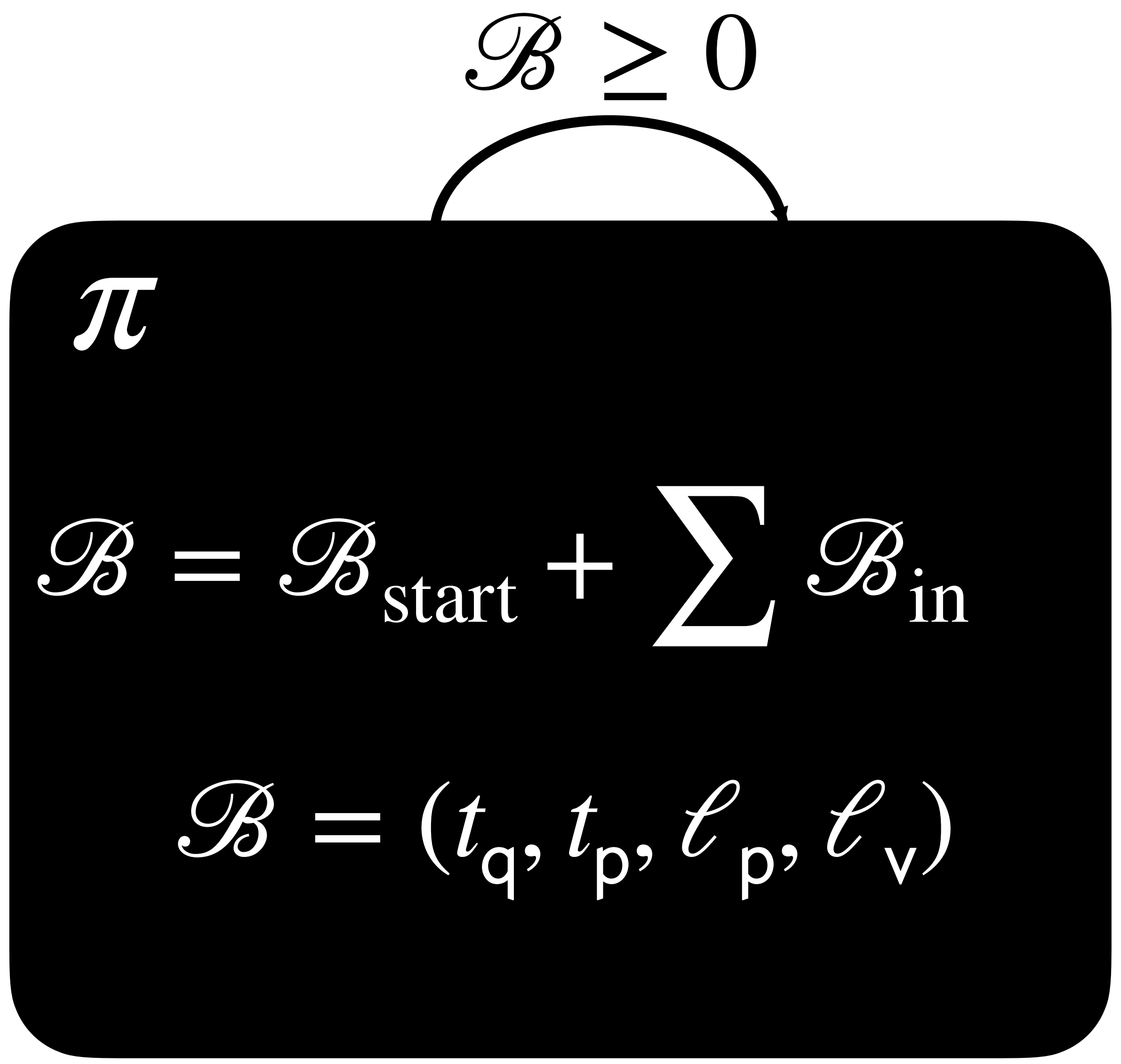
UC with Budgets

Plain UC only models adversaries that are **computationally** bounded


π
λ_1
λ_2
$\lambda_{\text{in}} = \sum \lambda_i$
$\lambda_{\text{out}} = \sum \lambda_i^*$
λ_1^*

$\text{time}(\pi) \leq p(\lambda_{\text{in}} - \lambda_{\text{out}})$

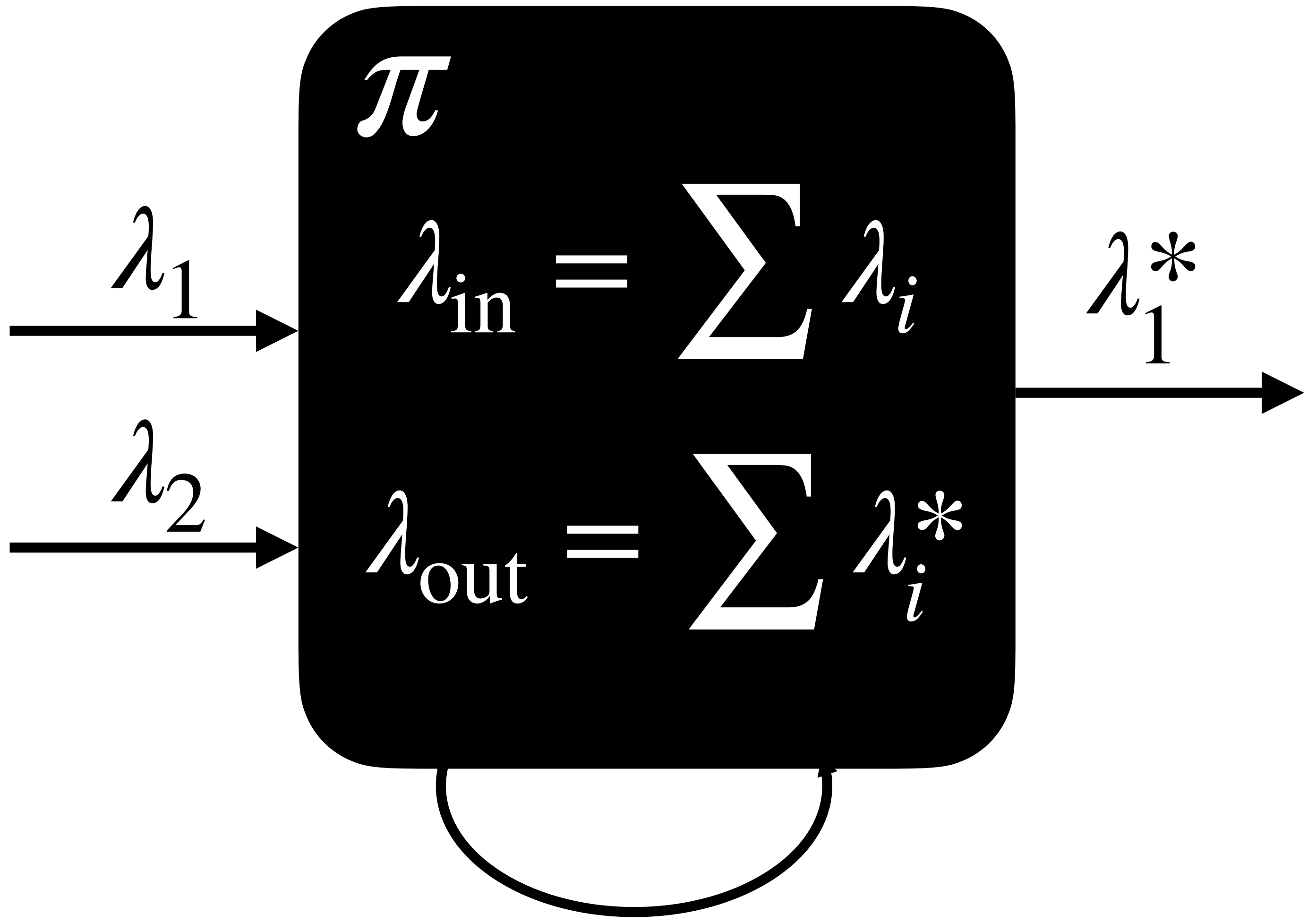
We consider adversaries that are **resource** bounded and computationally **unbounded**


$\mathscr{B} \geq 0$
π
$\mathscr{B} = \mathscr{B}_{\text{start}} + \sum \mathscr{B}_{\text{in}}$
$\mathscr{B} = (t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}})$

UC with Budgets

We consider adversaries that are **resource** bounded and computationally **unbounded**

Plain UC only models adversaries that are **computationally** bounded



π

λ_1

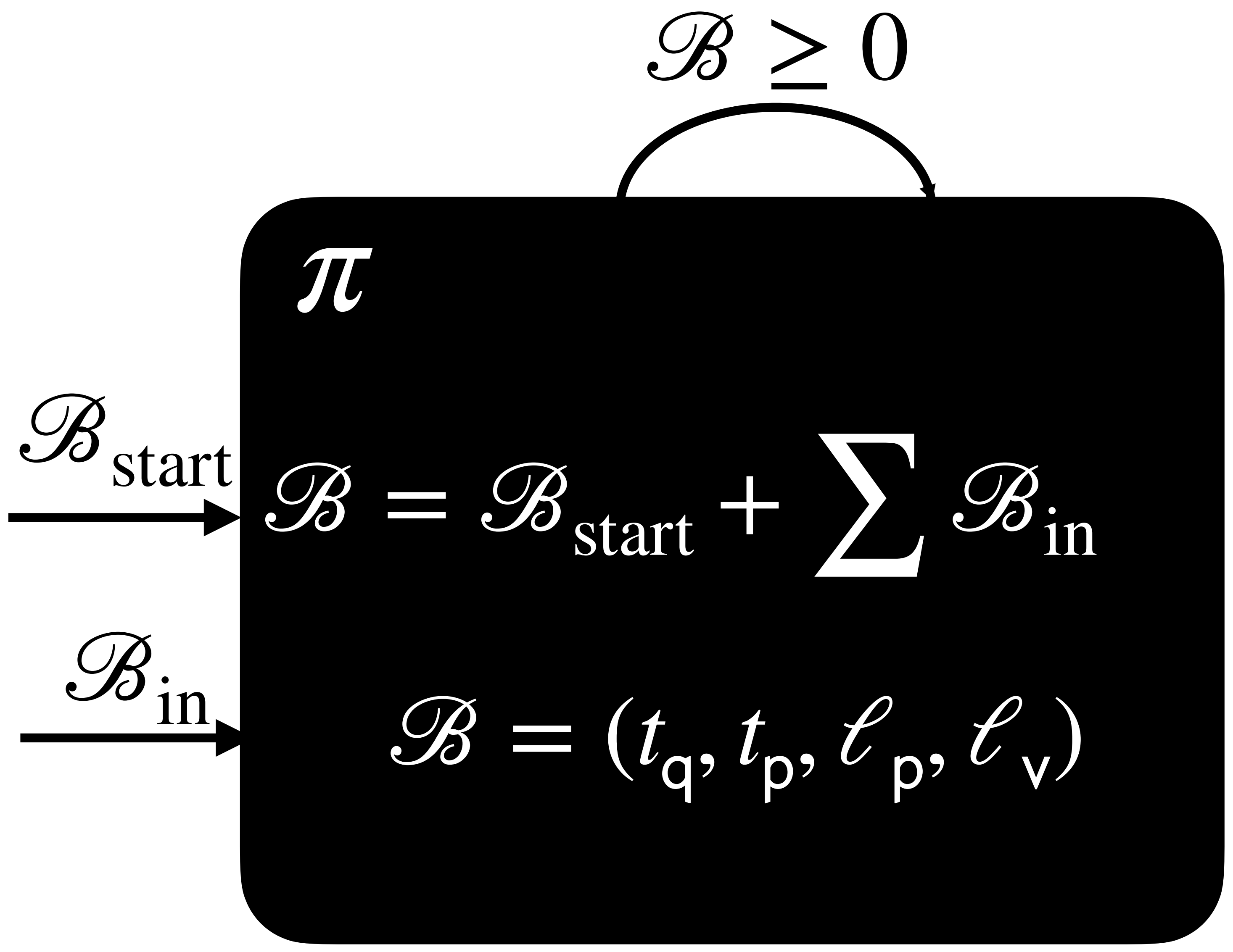
λ_2

$\lambda_{\text{in}} = \sum \lambda_i$

$\lambda_{\text{out}} = \sum \lambda_i^*$

λ_1^*

time$(\pi) \leq p(\lambda_{\text{in}} - \lambda_{\text{out}})$



$\mathscr{B} \geq 0$

π

$\mathscr{B}_{\text{start}}$

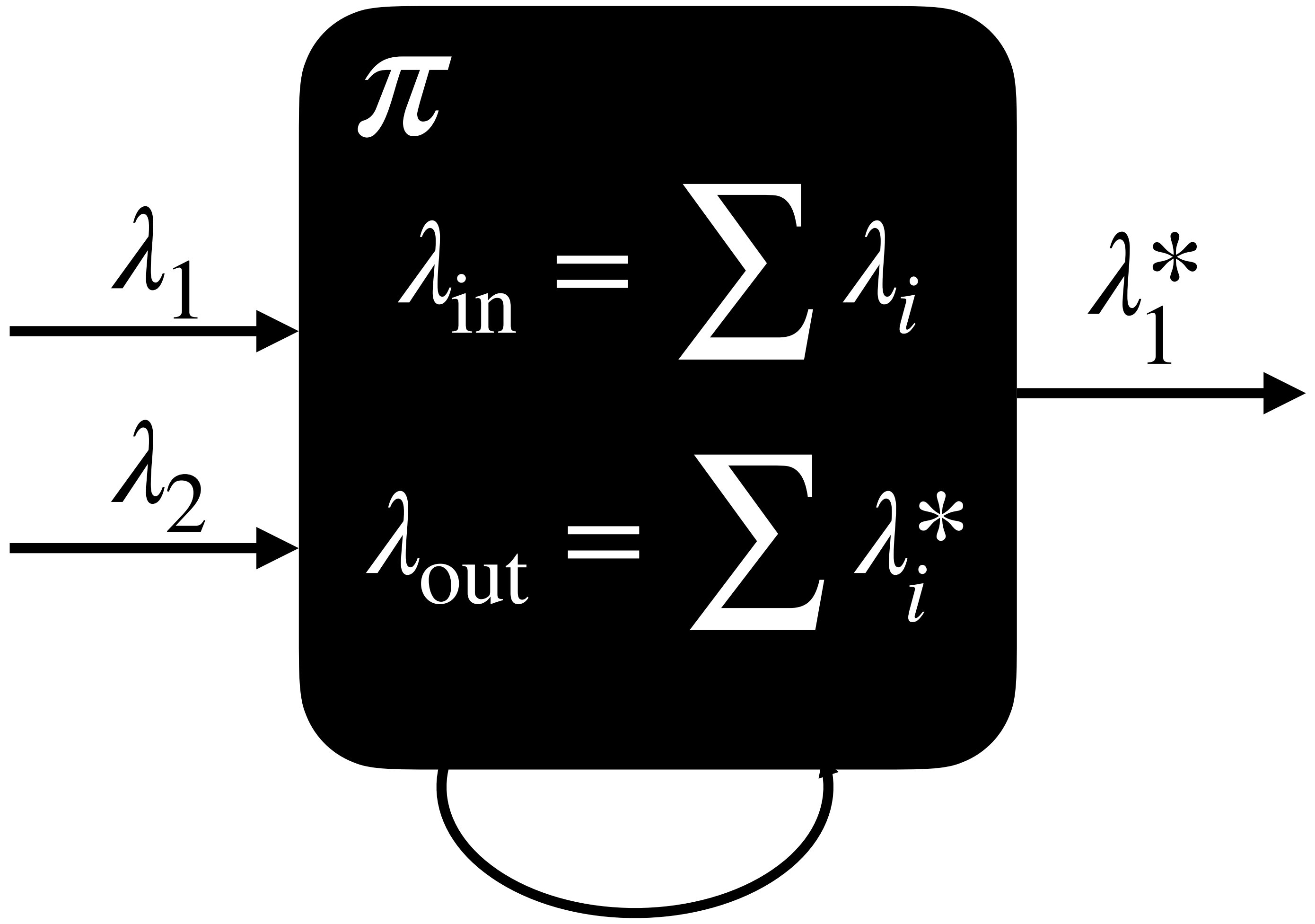
$\mathscr{B}_{\text{in}}$

$\mathscr{B} = \mathscr{B}_{\text{start}} + \sum \mathscr{B}_{\text{in}}$

$\mathscr{B} = (t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}})$

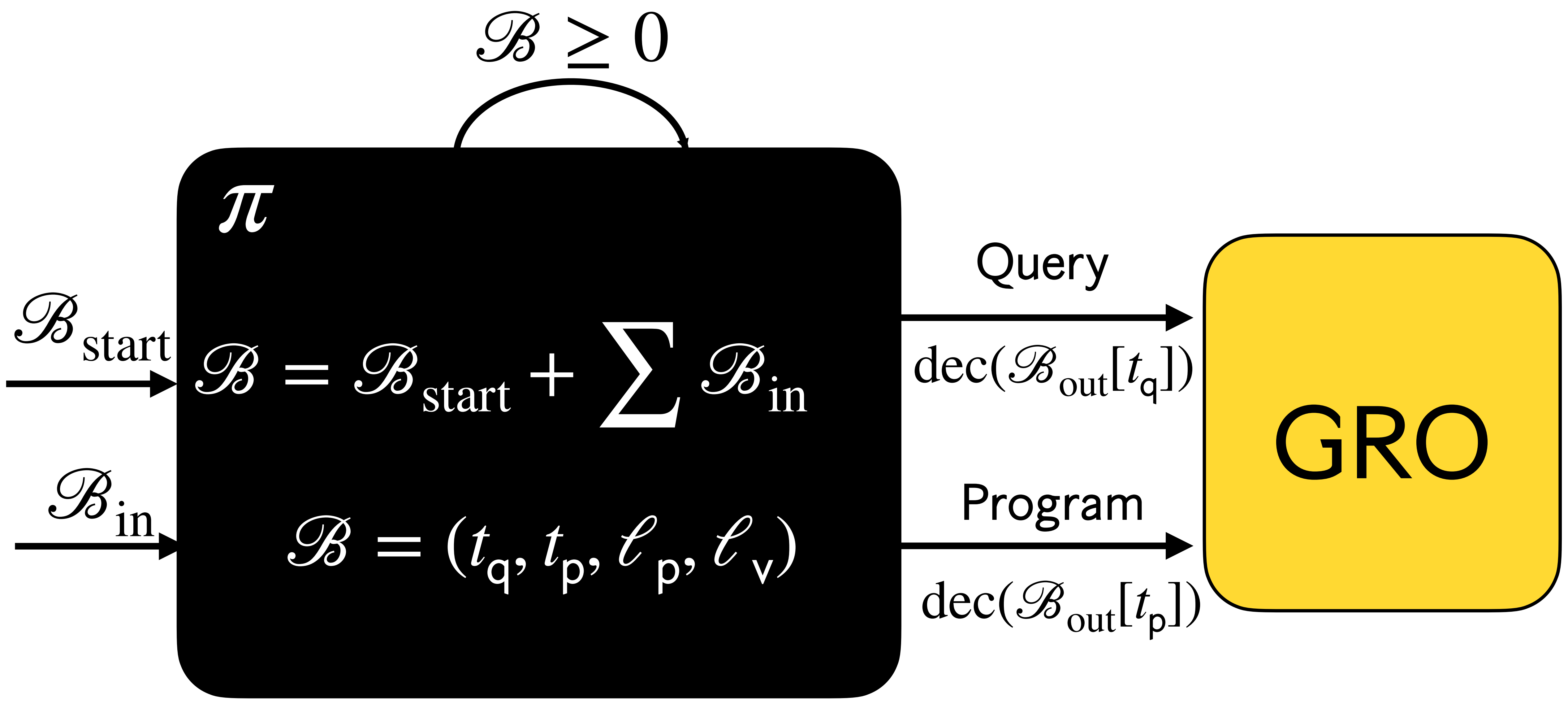
UC with Budgets

Plain UC only models adversaries that are **computationally** bounded


π
λ_1
λ_2
$\lambda_{\text{in}} = \sum \lambda_i$
$\lambda_{\text{out}} = \sum \lambda_i^*$
λ_1^*

$\text{time}(\pi) \leq p(\lambda_{\text{in}} - \lambda_{\text{out}})$

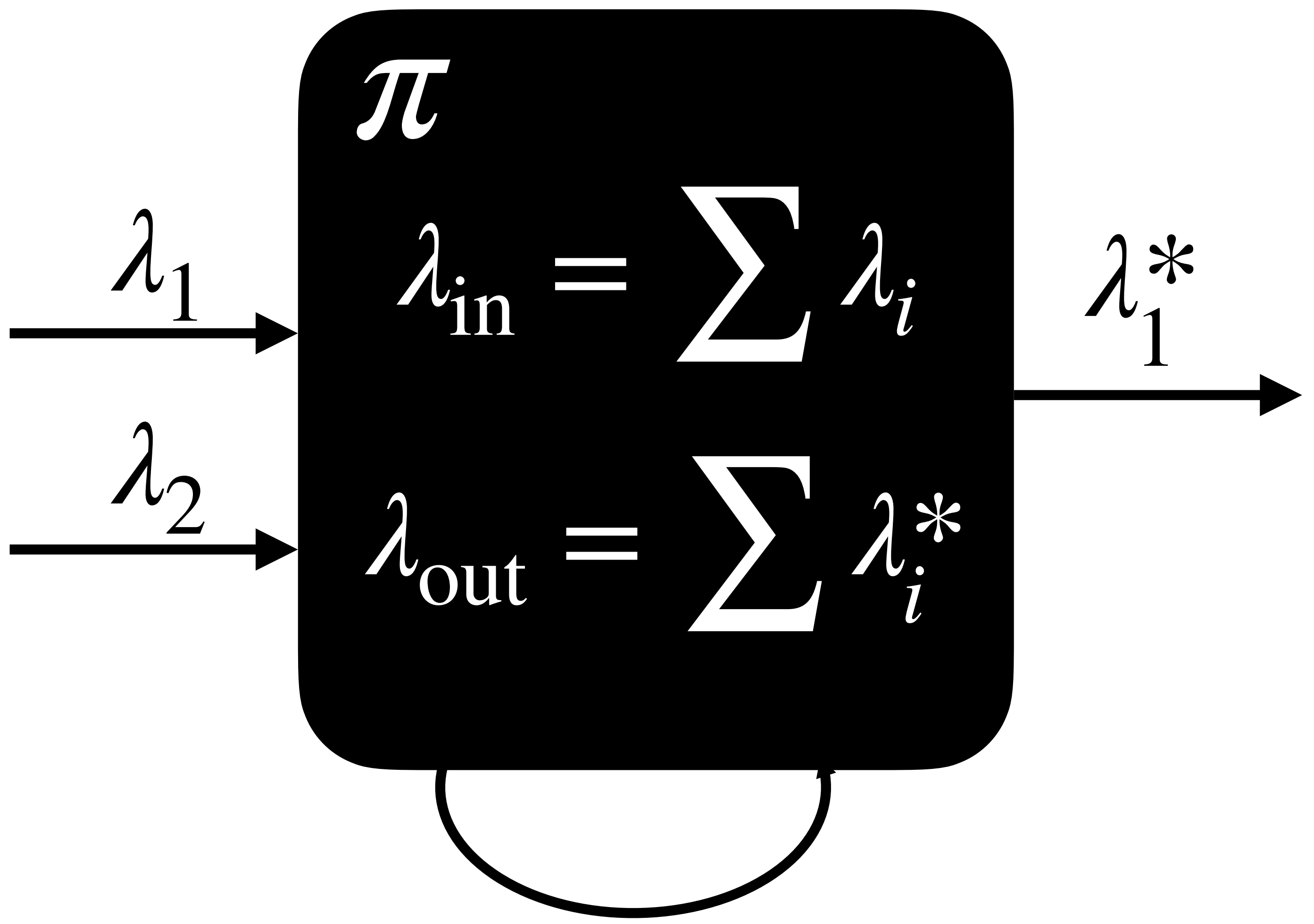
We consider adversaries that are **resource** bounded and computationally **unbounded**


$\mathscr{B} \geq 0$
π
$\mathscr{B}_{\text{start}}$
$\mathscr{B}_{\text{in}}$
$\mathscr{B} = \mathscr{B}_{\text{start}} + \sum \mathscr{B}_{\text{in}}$
$\mathscr{B} = (t_{\text{q}}, t_{\text{p}}, \ell_{\text{p}}, \ell_{\text{v}})$
Query
$\text{dec}(\mathscr{B}_{\text{out}}[t_{\text{q}}])$
Program
$\text{dec}(\mathscr{B}_{\text{out}}[t_{\text{p}}])$
GRO

$\text{dec}(x) := x \leftarrow x - 1$

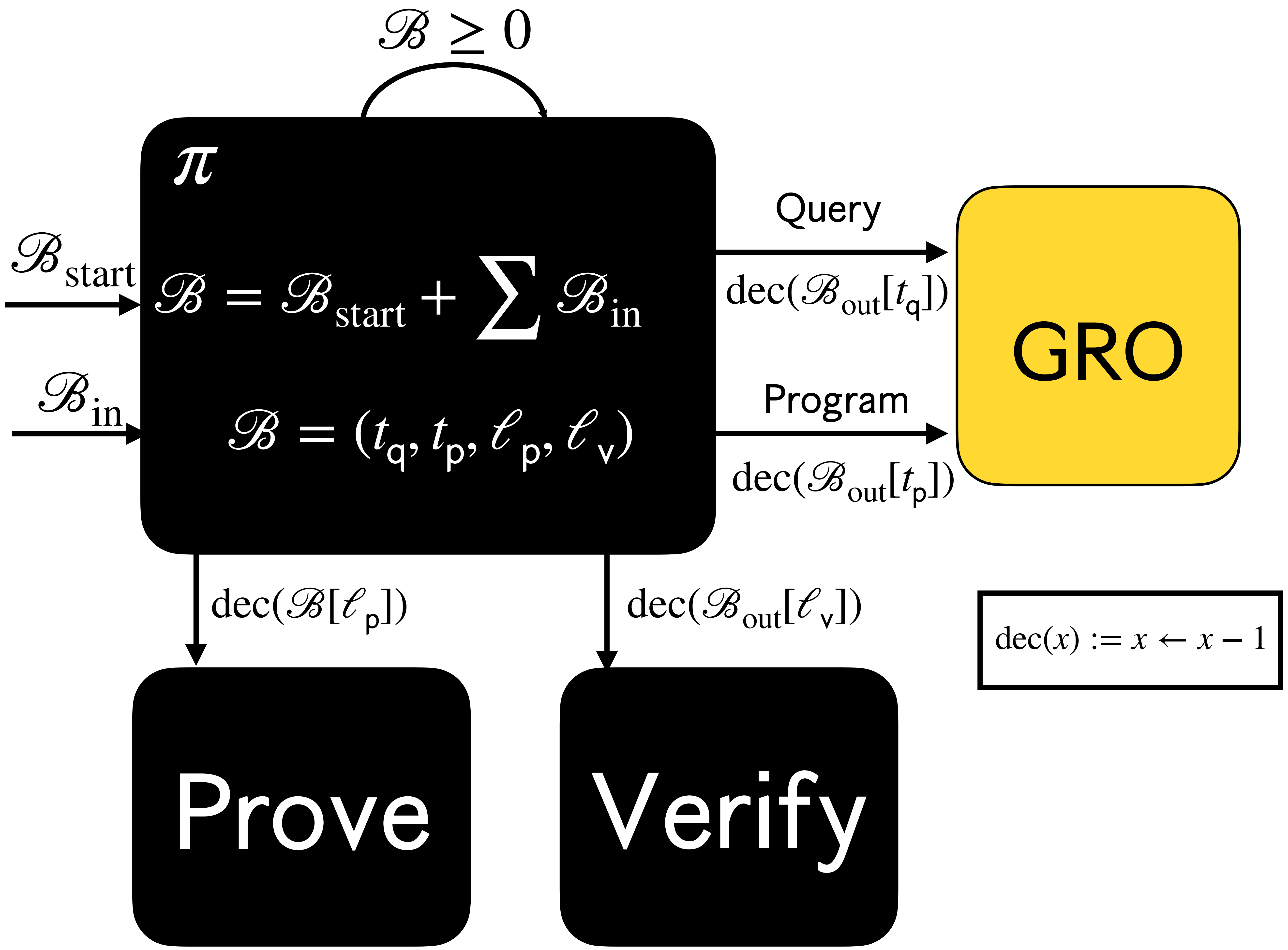
UC with Budgets

Plain UC only models adversaries that are **computationally** bounded


π
λ_1
λ_2
$\lambda_{\text{in}} = \sum \lambda_i$
$\lambda_{\text{out}} = \sum \lambda_i^*$
λ_1^*

$\text{time}(\pi) \leq p(\lambda_{\text{in}} - \lambda_{\text{out}})$

We consider adversaries that are **resource** bounded and computationally **unbounded**


$\mathscr{B} \geq 0$
π
$\mathscr{B}_{\text{start}}$
$\mathscr{B}_{\text{in}}$
$\mathscr{B} = \mathscr{B}_{\text{start}} + \sum \mathscr{B}_{\text{in}}$
$\mathscr{B} = (t_{\text{q}}, t_{\text{p}}, \ell_{\text{p}}, \ell_{\text{v}})$
Query
$\text{dec}(\mathscr{B}_{\text{out}}[t_{\text{q}}])$
Program
$\text{dec}(\mathscr{B}_{\text{out}}[t_{\text{p}}])$
GRO
$\text{dec}(\mathscr{B}[\ell_{\text{p}}])$
$\text{dec}(\mathscr{B}_{\text{out}}[\ell_{\text{v}}])$
Prove
Verify
$\text{dec}(x) := x \leftarrow x - 1$