What does the SHARK say?

Mariana Raykova
Eran Tromer
Madars Virza

Zero-knowledge on the blockchain

Zero-knowledge on the blockchain

Privacy-preserving cryptocurrencies

Privacy-preserving smart contracts

Proof of regulatory compliance

Blockchain-based sovereign identity

Zero-knowledge on the blockchain

Privacy-preserving cryptocurrencies

Privacy-preserving smart contracts

Proof of regulatory compliance

Blockchain-based sovereign identity

see [ZKProof Standardization – Applications Track]

Zero-knowledge on the blockchain

Privacy-preserving cryptocurrencies

Privacy-preserving smart contracts

Proof of regulatory compliance

Blockchain-based sovereign identity

see [ZKProof Standardization – Applications Track]

Need zero-knowledge non-interactive arguments of knowledge, ideally succinct ones: zk-SNARK.

Which zk-SNARK?

Which zk-SNARK?

- QAP-based [GGPR13][PGHR13][BCGTV13][DFGK14][Groth16][GM17][BG18]…

  Fastest verification. Widely used.

Which zk-SNARK?

- QAP-based

[GGPR13][PGHR13][BCGTV13][DFGK14] [Groth16][GM17][BG18]…

Fastest verification. Widely used.

Require a Common Reference String (CRS)

Which zk-SNARK?

- QAP-based [GGPR13][PGHR13][BCGTV13][DFGK14][Groth16][GM17][BG18]…

  Fastest verification. Widely used.

  Require a ~~Common Reference String (CRS)~~ Structured Reference String (SRS)

  ZKProof Standardization

Which zk-SNARK?

- QAP-based [GGPR13][PGHR13][BCGTV13][DFGK14][Groth16][GM17][BG18]…

  Fastest verification. Widely used.

  Require a Structured Reference String (SRS)

Which zk-SNARK?

- QAP-based

[GGPR13][PGHR13][BCGTV13][DFGK14][Groth16][GM17][BG18]…

Fastest verification. Widely used.

Require a Structured Reference String (SRS)

Generating an SRS:

- Trusted setup

Which zk-SNARK?

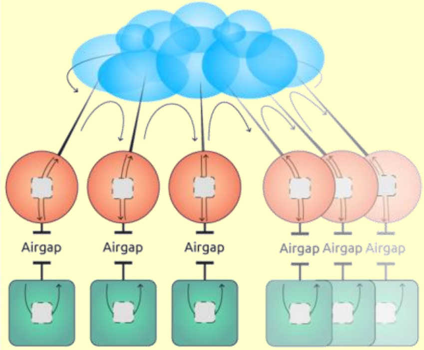
- QAP-based

[GGPR13][PGHR13][BCGTV13][DFGK14][Groth16][GM17][BG18]…

Fastest verification. Widely used.

Require a Structured Reference String (SRS)

Generating an SRS:



- Trusted setup
- MPC + destruction [BCGTV15][BGG17]
- Updatable SRS [GKMMM18, MBKM19]
- Scalable SRS generation ("Powers of Tau") [BGM18]

Avoiding a Structured Reference String

Other zk-SNARKs

- PCP-based (e.g., *libSTARK*)

[Micali94][BCGT13][BCS16][BBCGGHPRSTV17][BBHR18]

Asymptotically succinct but large constants.

Avoiding a Structured Reference String

Other zk-SNARKs

- PCP-based (e.g., *libSTARK*)

[Micali94][BCGT13][BCS16][BBCGGHPRSTV17][BBHR18]

Asymptotically succinct but large constants.

Non-succinct ZK:

Avoiding a Structured Reference String

Other zk-SNARKs

- PCP-based (e.g., *libSTARK*)

[Micali94][BCGT13][BCS16][BBCGGHPRSTV17][BBHR18]

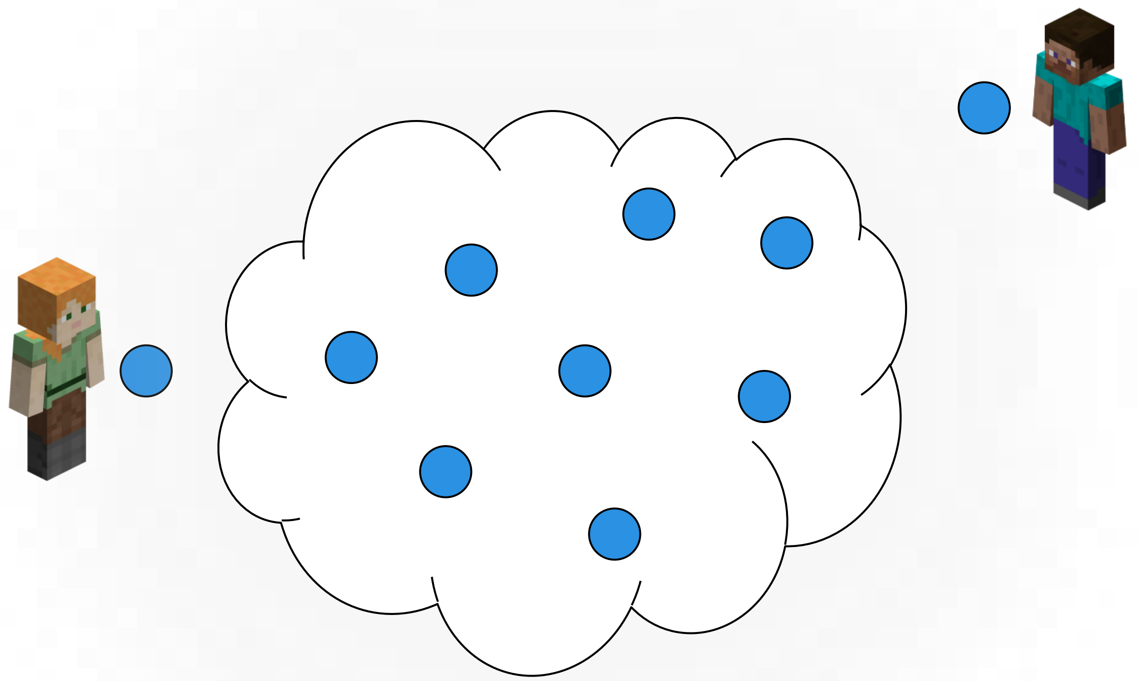
Asymptotically succinct but large constants.

Non-succinct ZK:

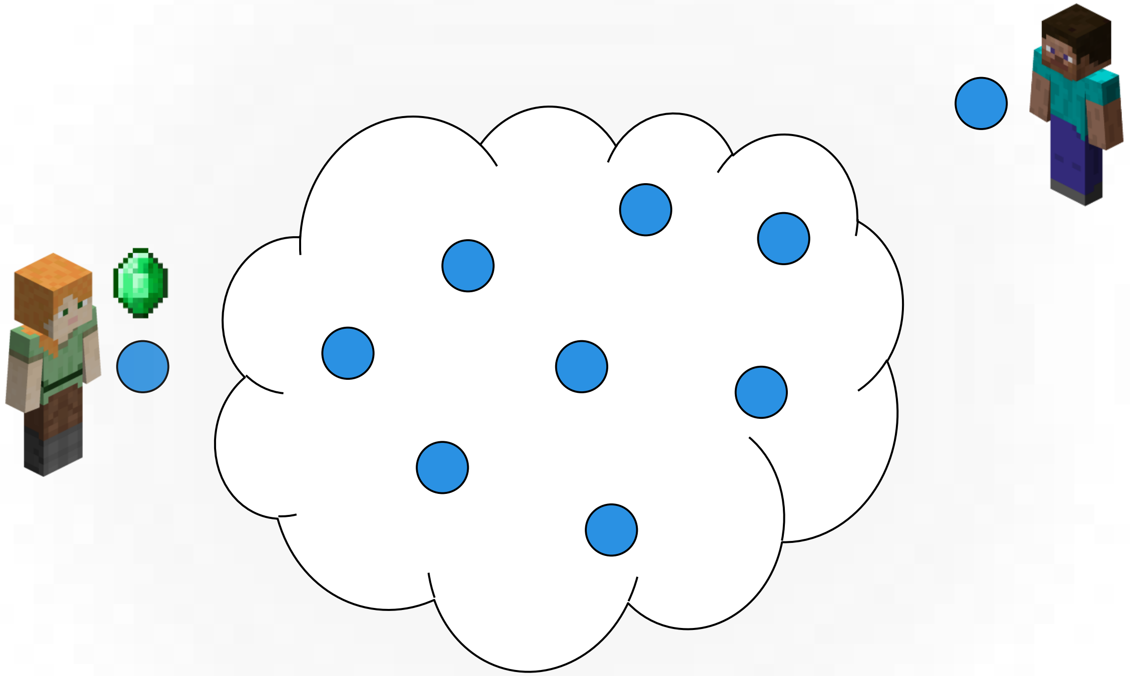
- Aurora [BCRSVW19]
- Bulletproofs [BCCGP16][BBBPWM17]
- Hyrax [WTSTW17]
- Ligero [AHIV17]
- ZKBoo(++) [GMO16][CDGORRSZ17]

Slow verification and/or large proofs (as statement grows).

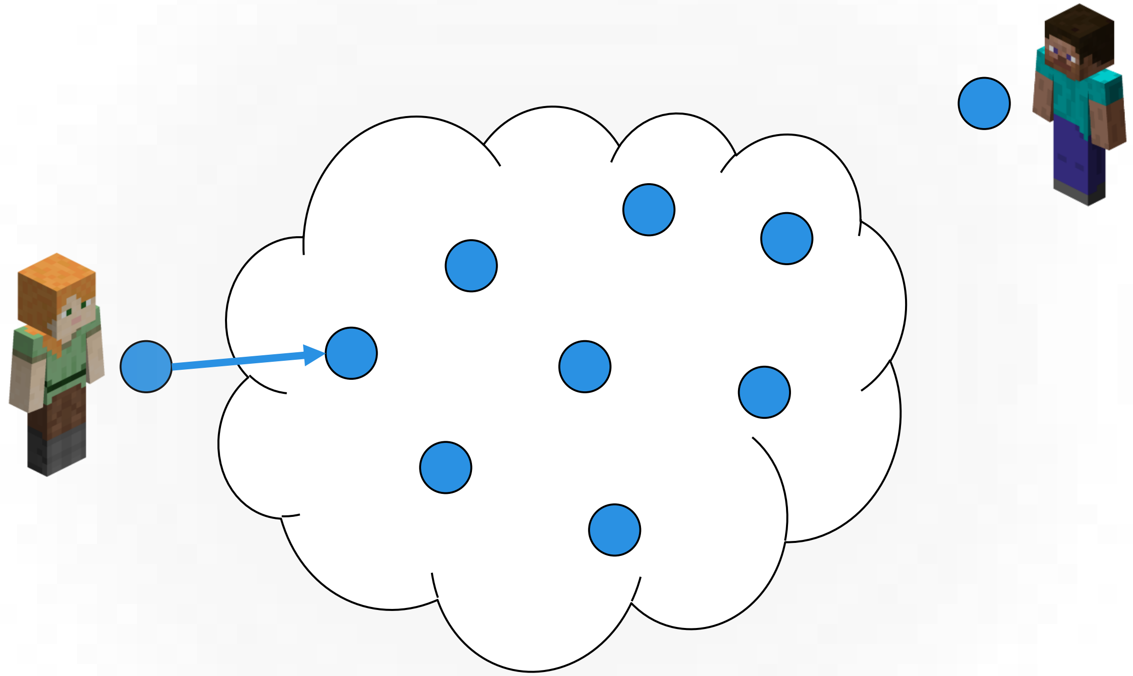
Proof transmission & verification speed



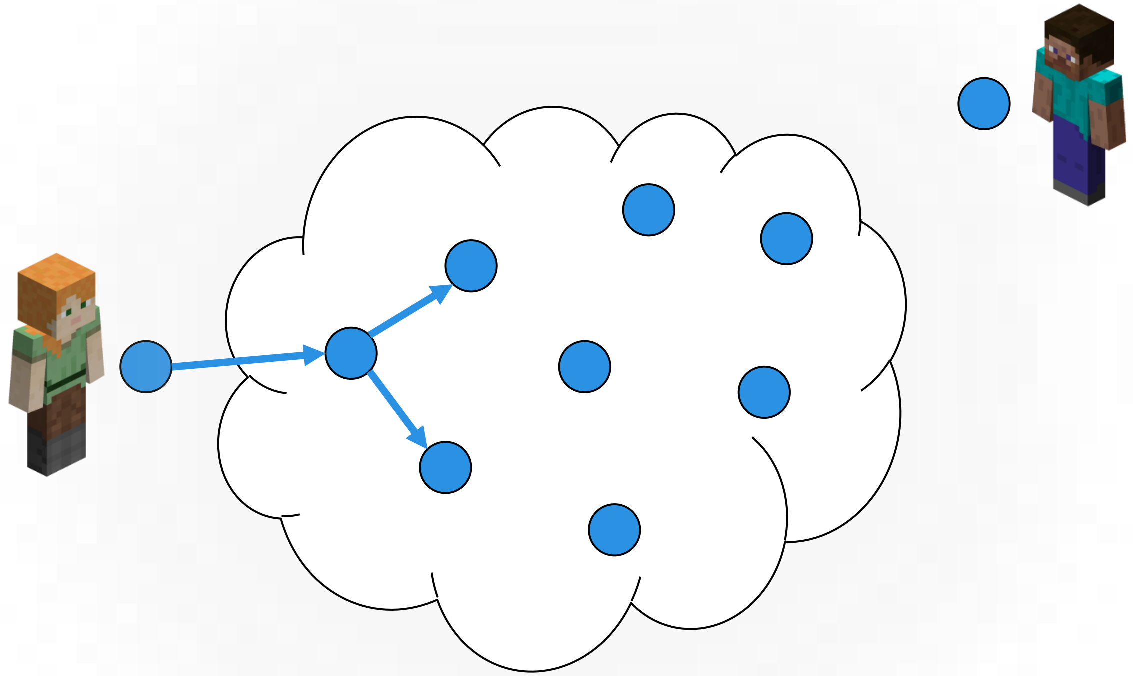
Proof transmission & verification speed



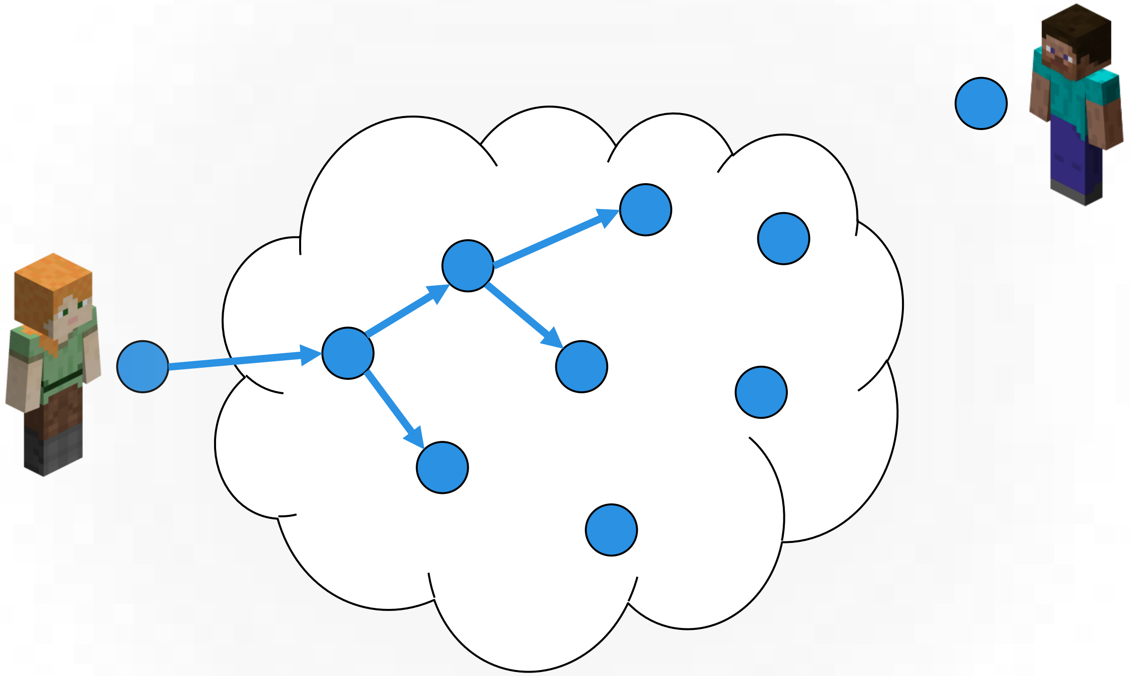
Proof transmission & verification speed



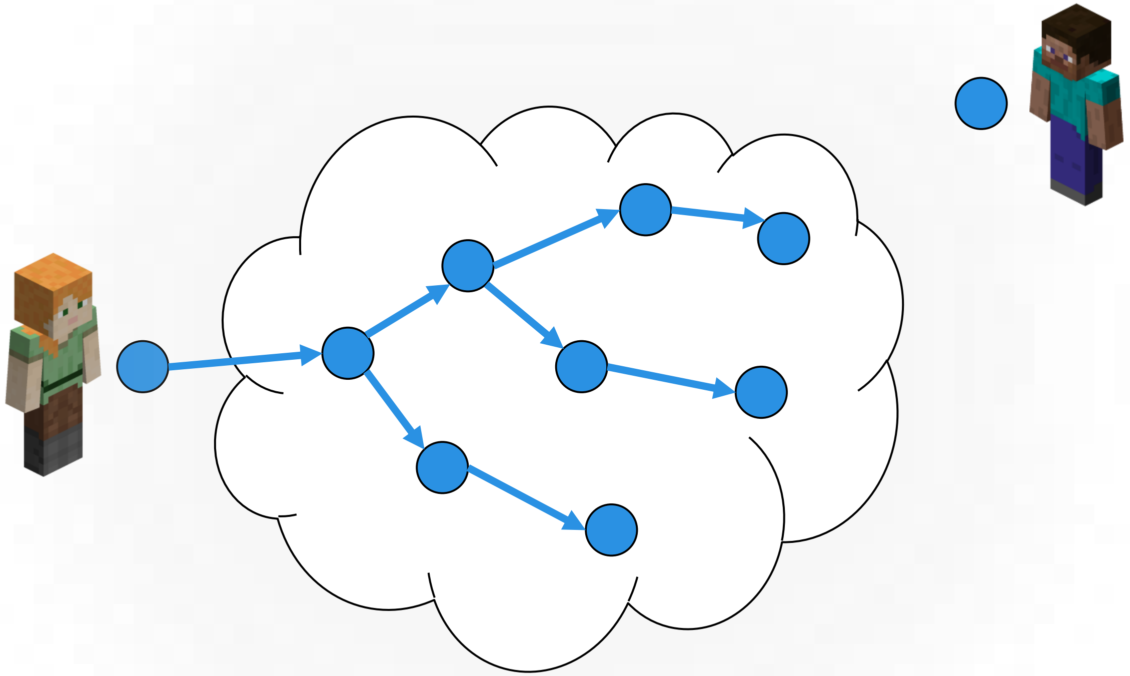
Proof transmission & verification speed



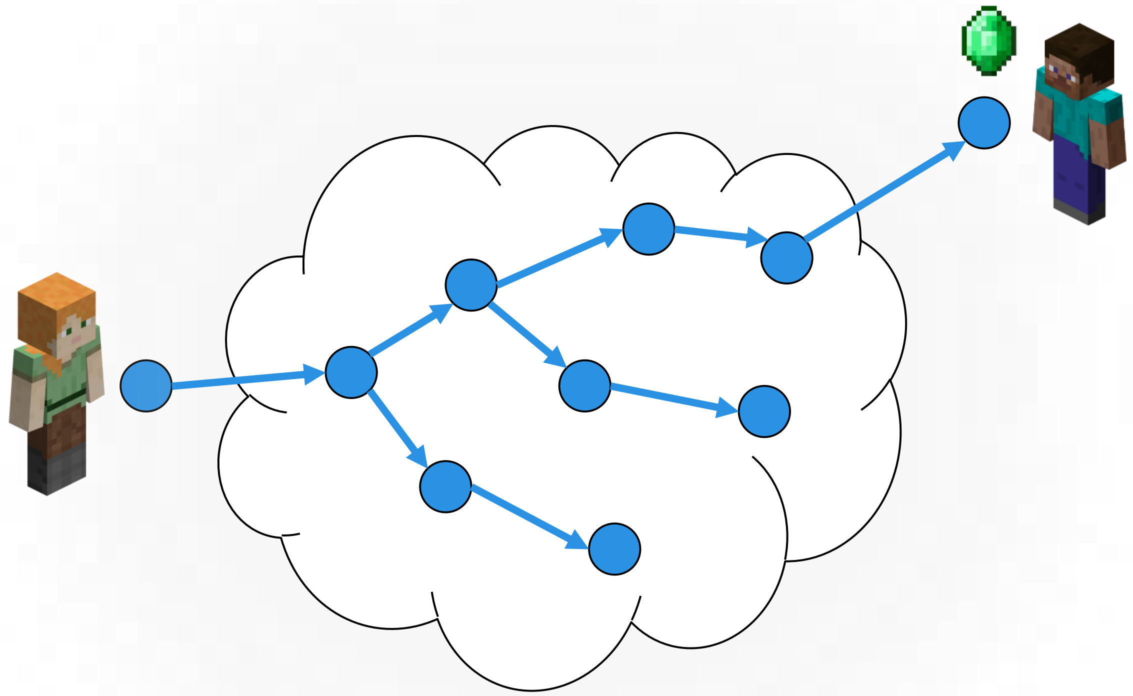
Proof transmission & verification speed



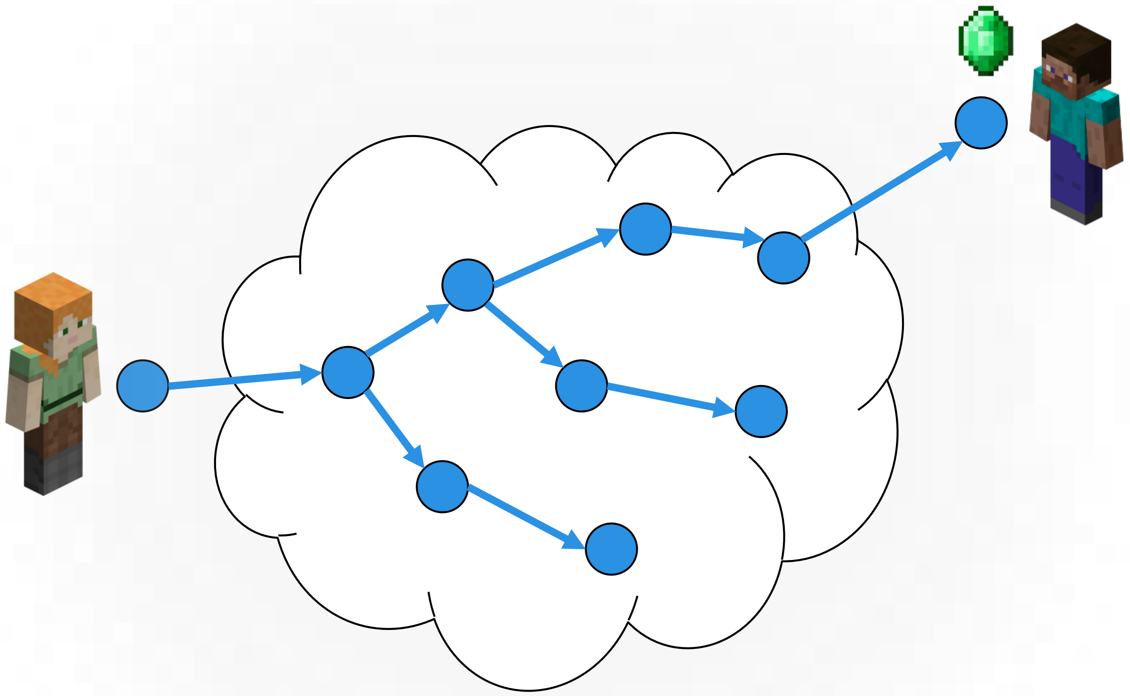
Proof transmission & verification speed



Proof transmission & verification speed

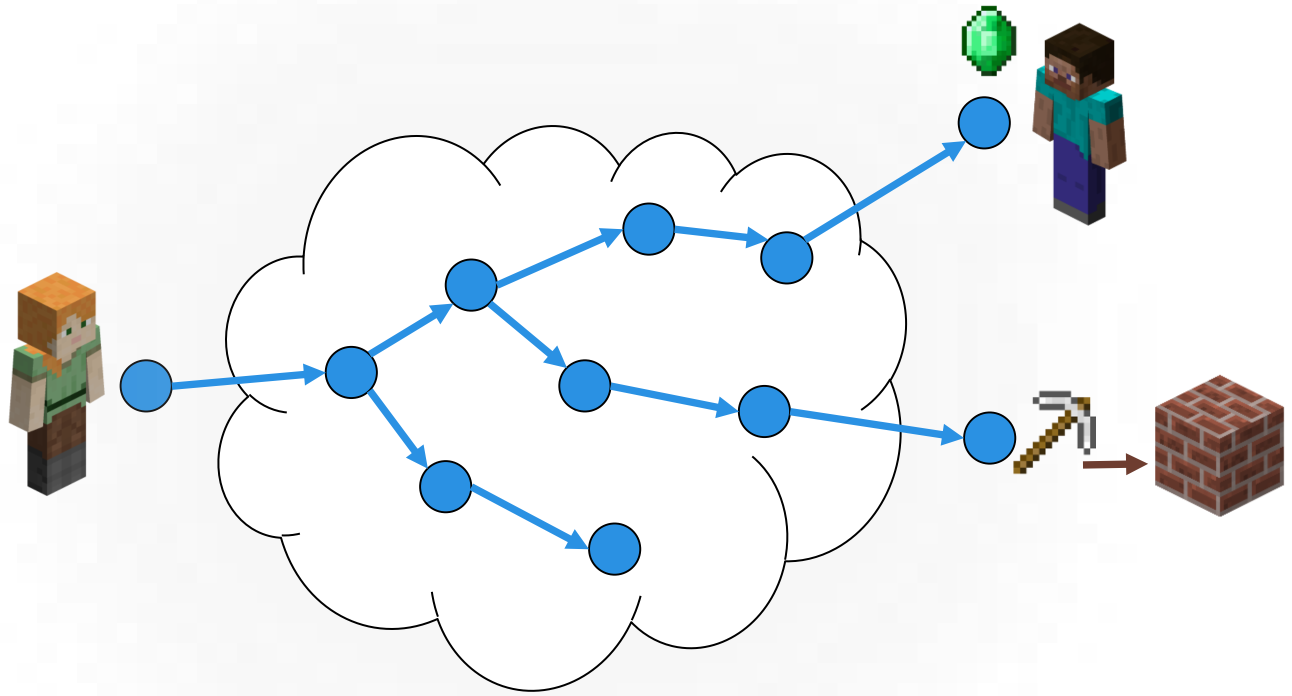


Proof transmission & verification speed



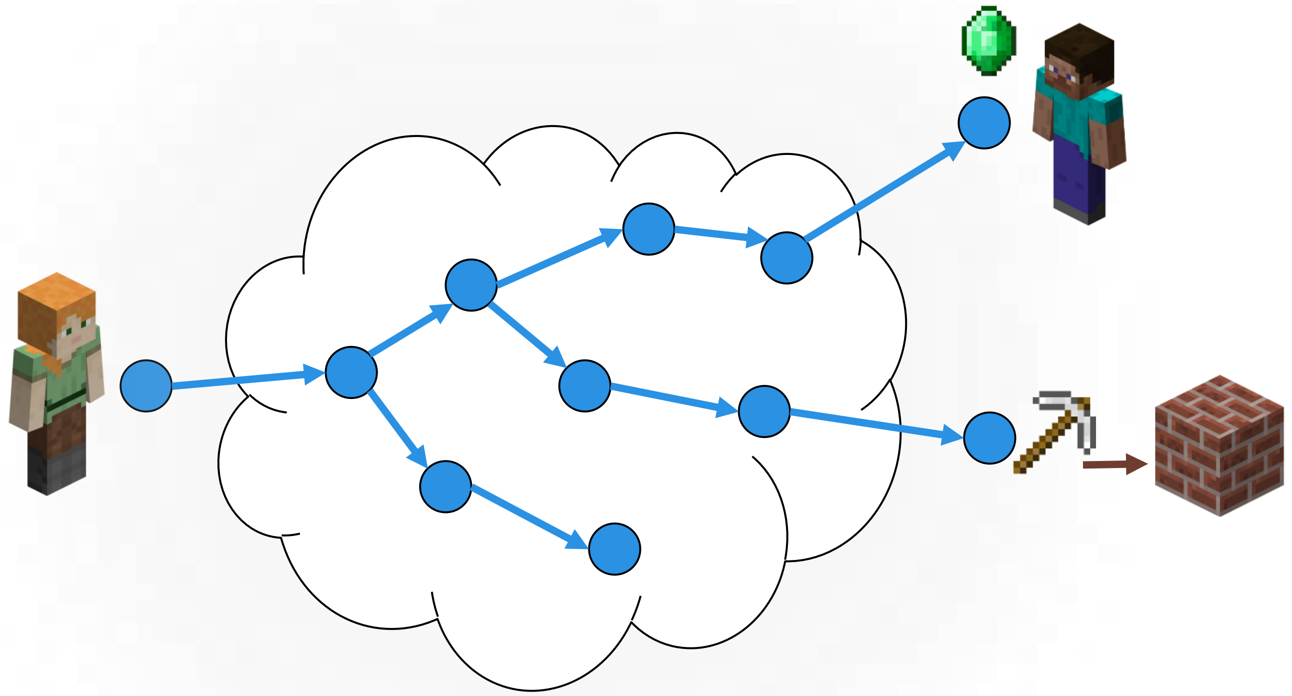
Seeing
transactions

Proof transmission & verification speed



Seeing transactions

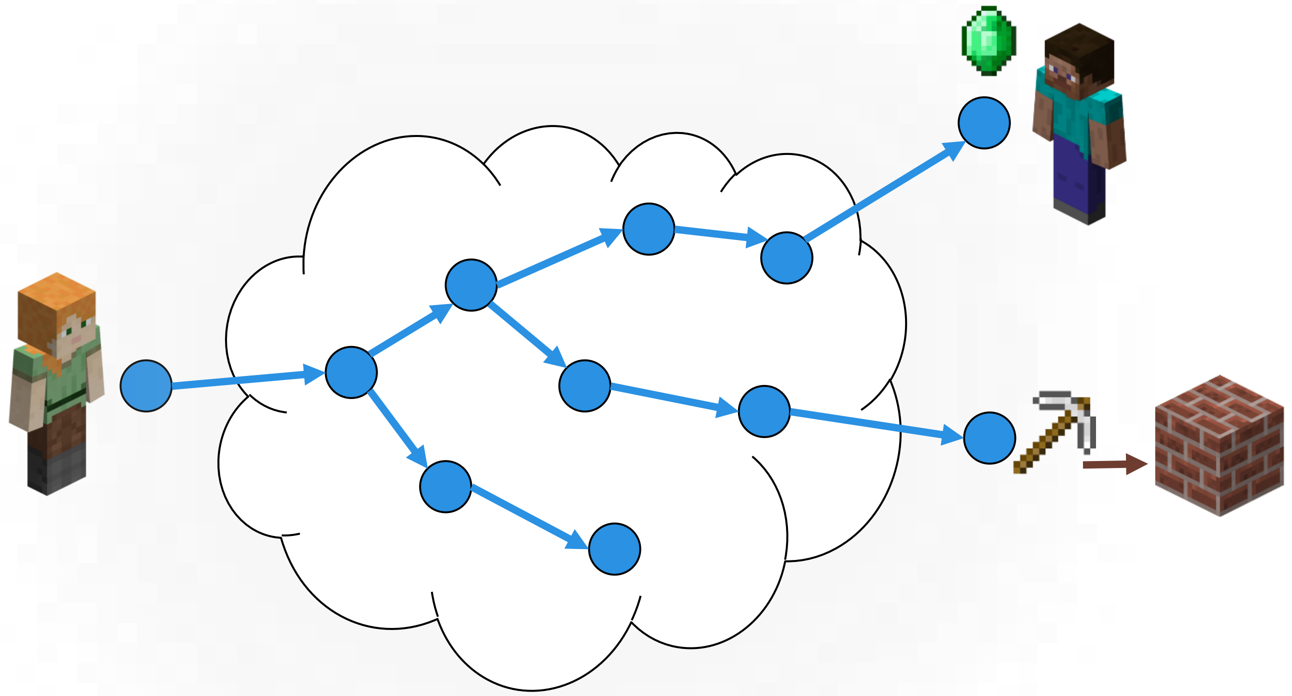
Proof transmission & verification speed



Seeing transactions

Including new transactions in a block

Proof transmission & verification speed

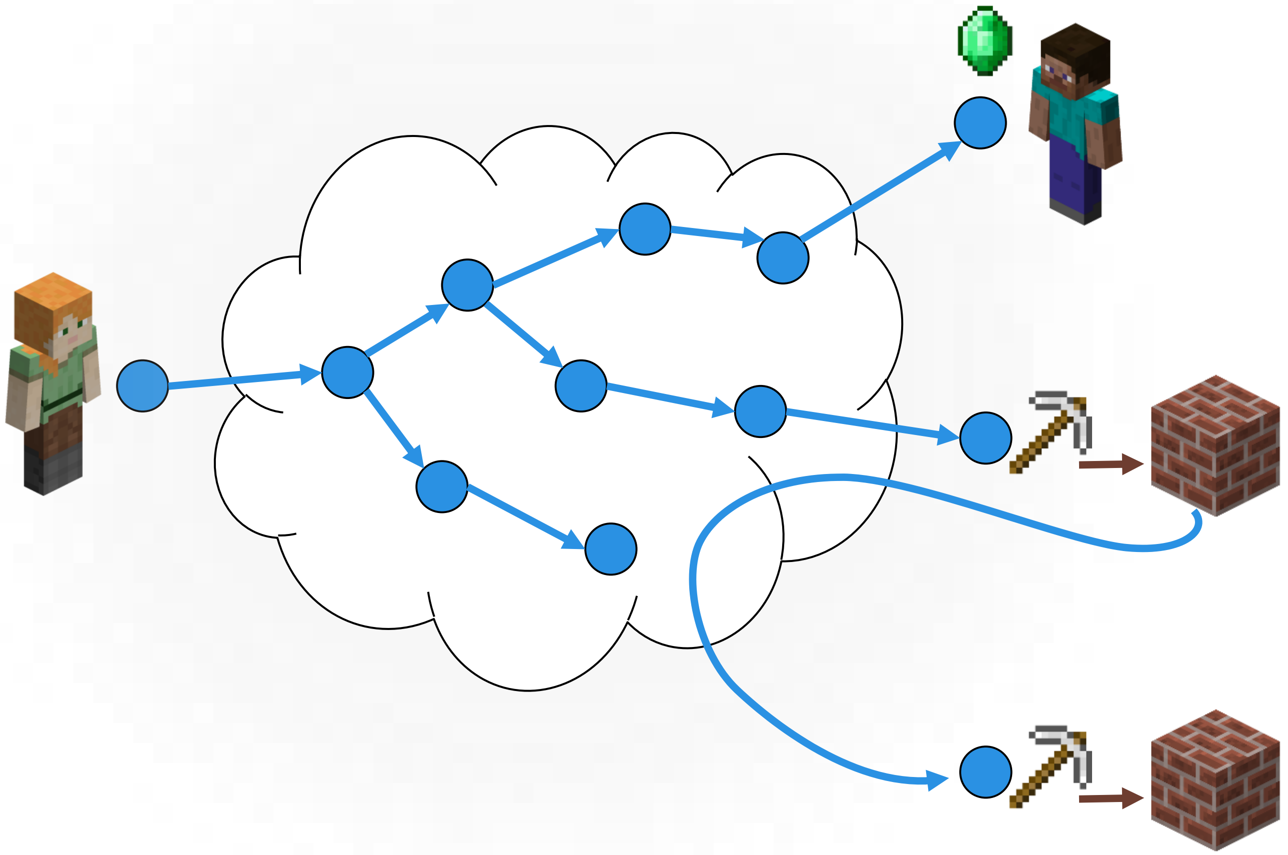


Seeing transactions

Including new transactions in a block

Slow verification
→ miners get a head start by not validating
→ double-spends and chain splits [July 2015 Bitcoin fork]

Proof transmission & verification speed

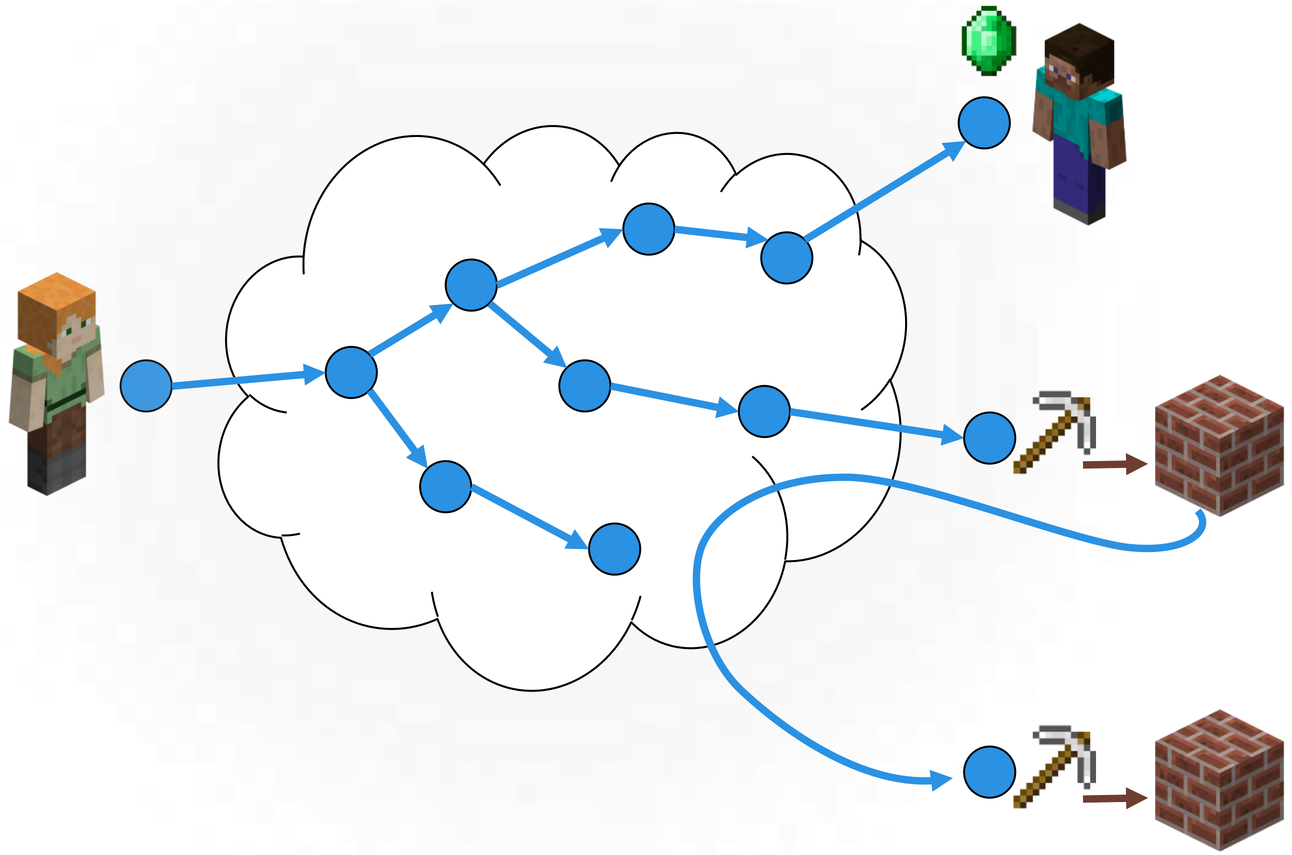


Seeing transactions

Including new transactions in a block

Slow verification
→ miners get a head start by not validating
→ double-spends and chain splits [July 2015 Bitcoin fork]

Proof transmission & verification speed



Seeing transactions

Including new transactions in a block

Verifying the previous block

Slow verification

→ miners get a head start by not validating

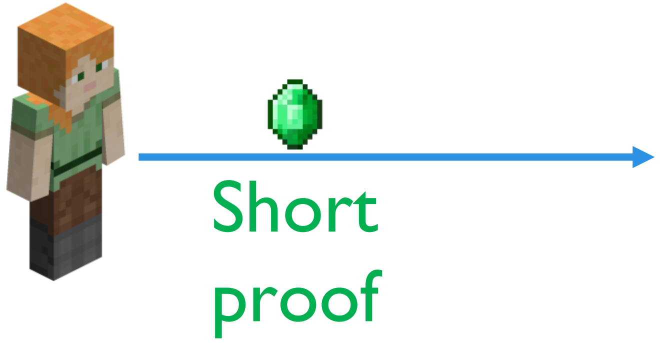
→ double-spends and chain splits [July 2015 Bitcoin fork]

zk-SHARK

zero-knowledge
succinct hybrid argument of knowledge

zk-SHARK

zero-knowledge
succinct hybrid argument of knowledge



zk-SHARK

zero-knowledge
succinct hybrid argument of knowledge

Short
proof

Prudent verification

Slow (comparable to Bulletproofs)

No SRS

zk-SHARK

zero-knowledge

succinct hybrid argument of knowledge

Short proof

Prudent verification

Slow (comparable to Bulletproofs)

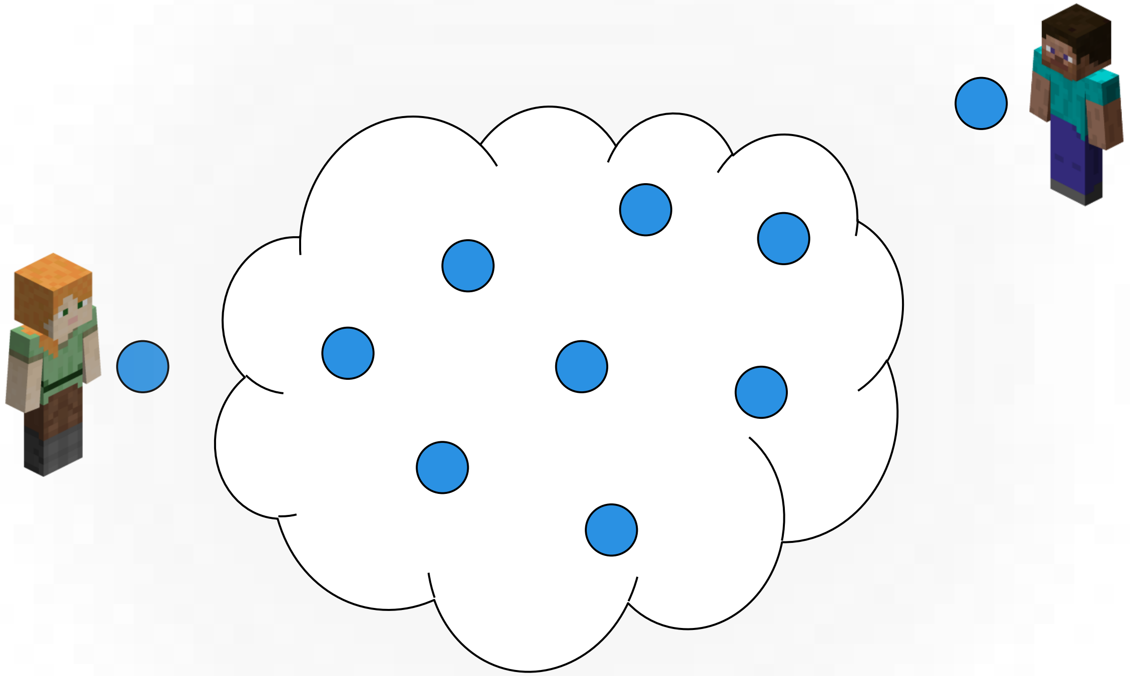
No SRS

Optimistic verification

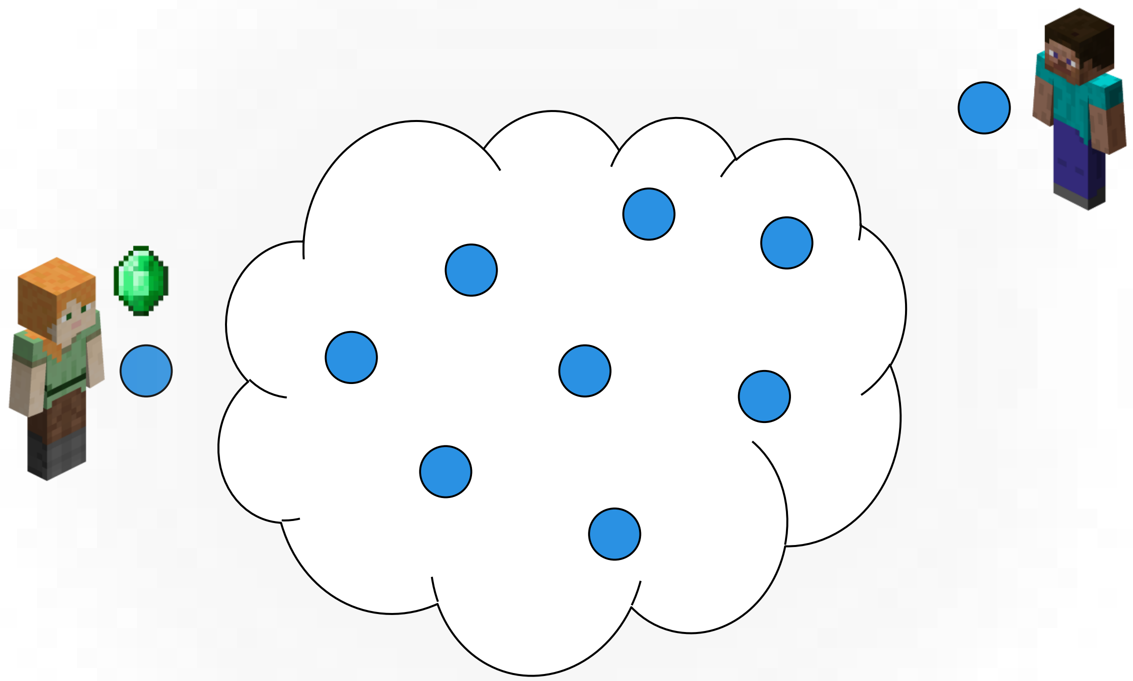
Fast (comparable to QAP-based SNARK)

Relies on SRS

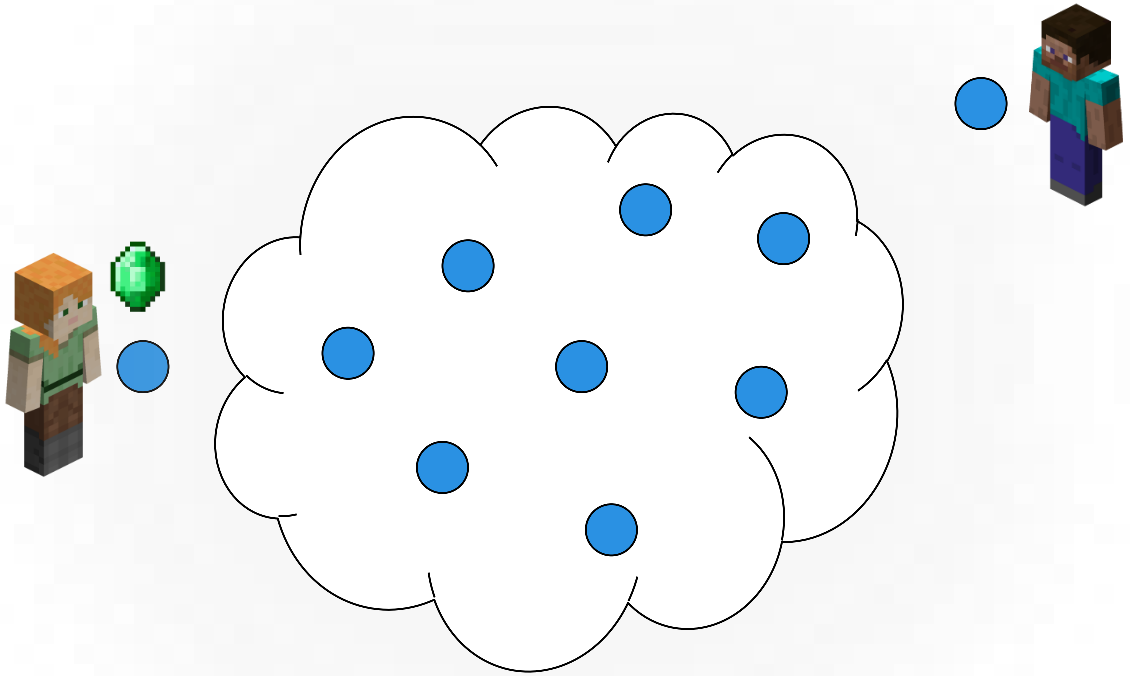
Proof transmission & verification speed



Proof transmission & verification speed

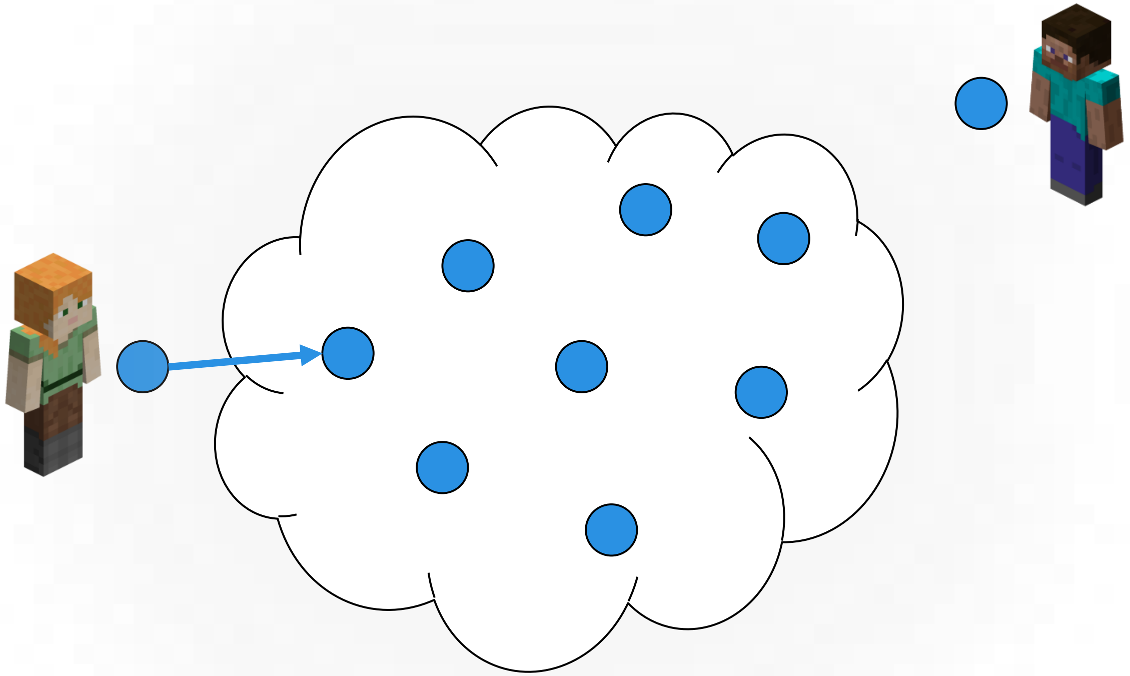


Proof transmission & verification speed



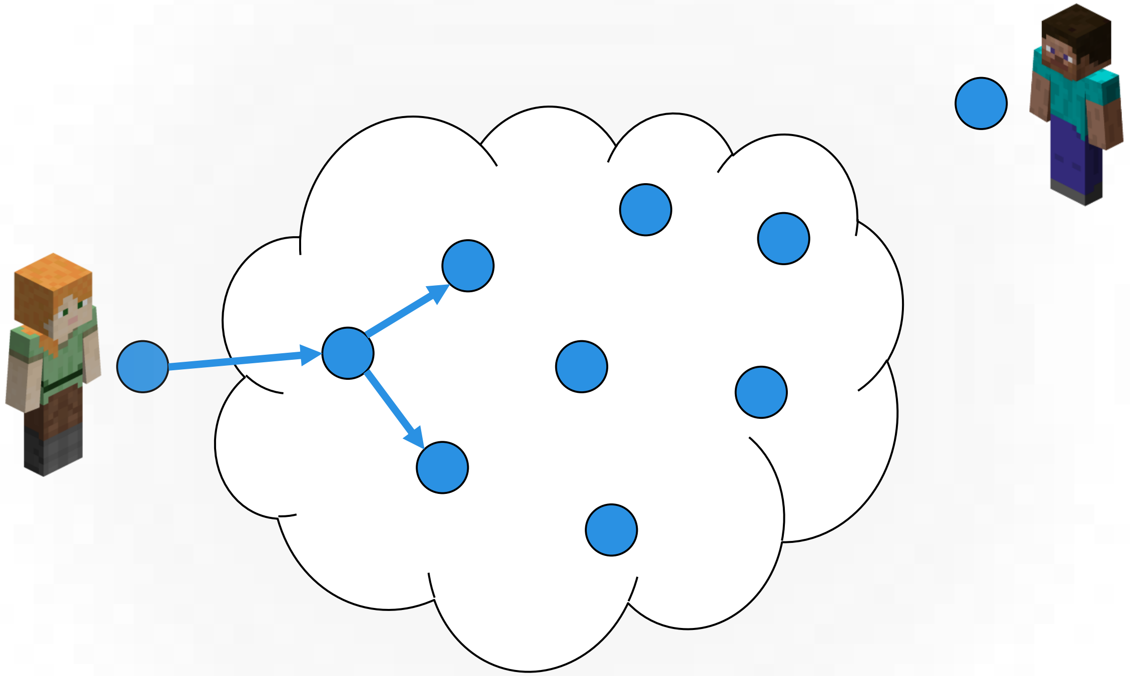
Optimistically verify during propagation.

Proof transmission & verification speed



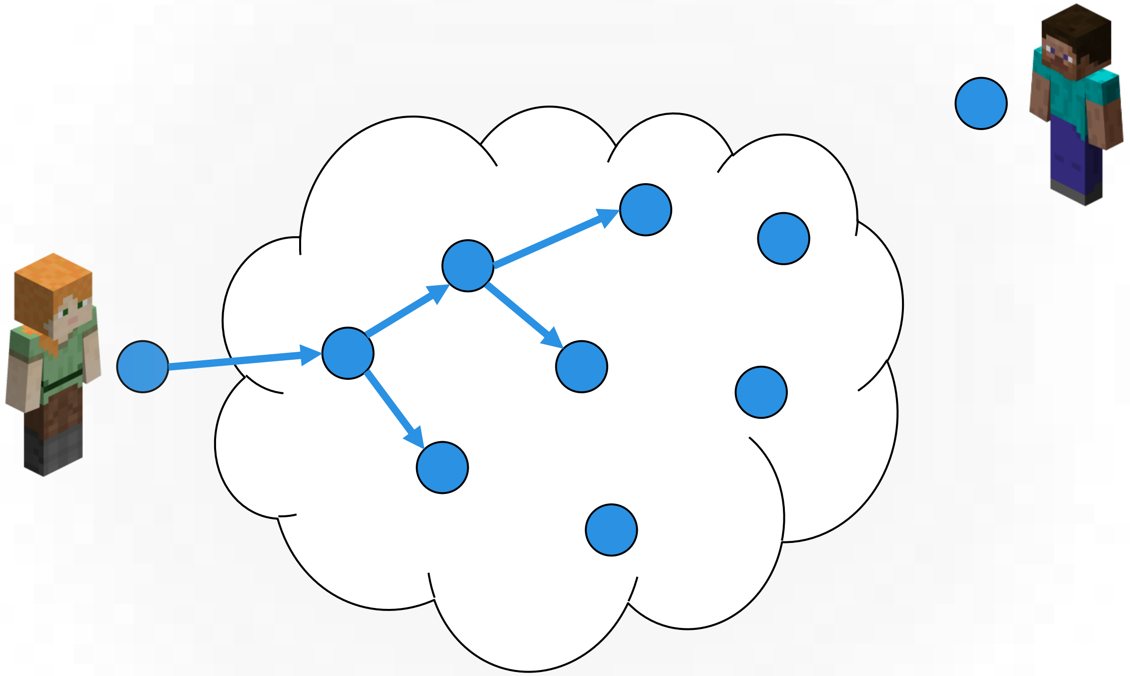
Optimistically verify during propagation.

Proof transmission & verification speed



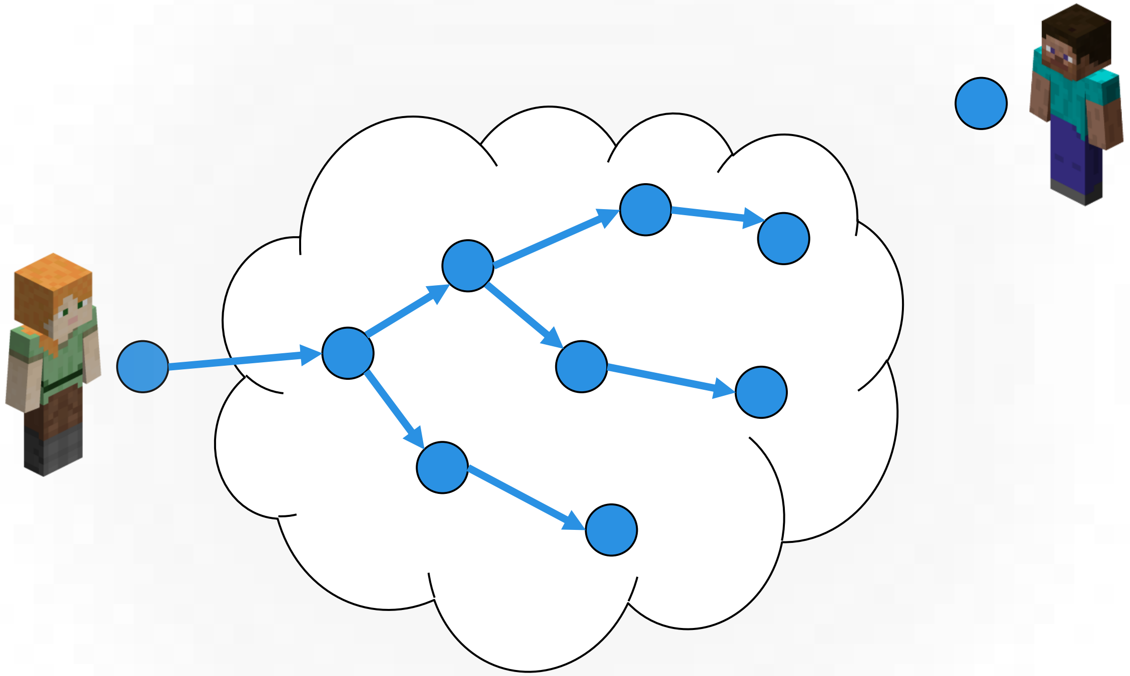
Optimistically verify during propagation.

Proof transmission & verification speed



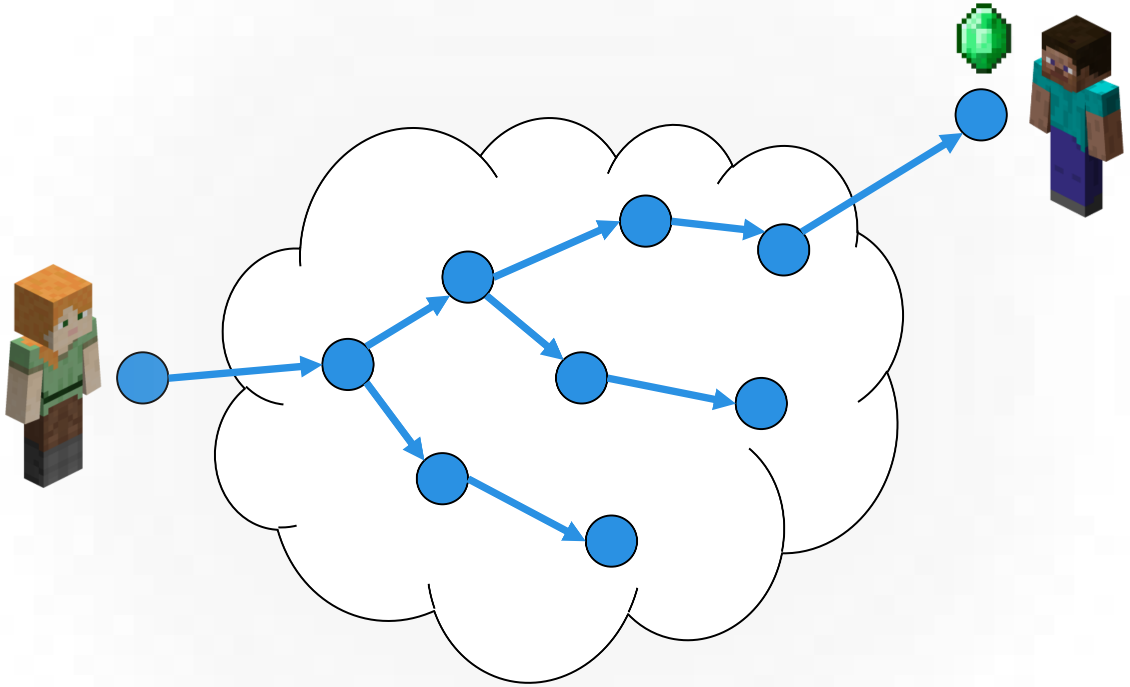
Optimistically verify during propagation.

Proof transmission & verification speed



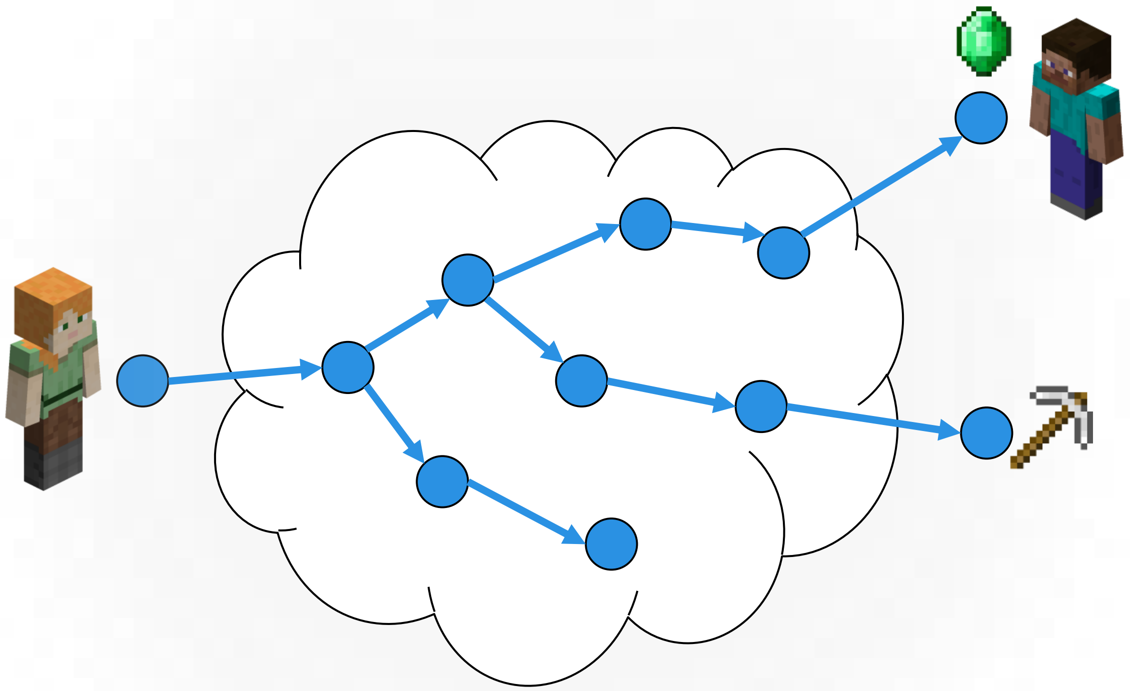
Optimistically verify during propagation.

Proof transmission & verification speed



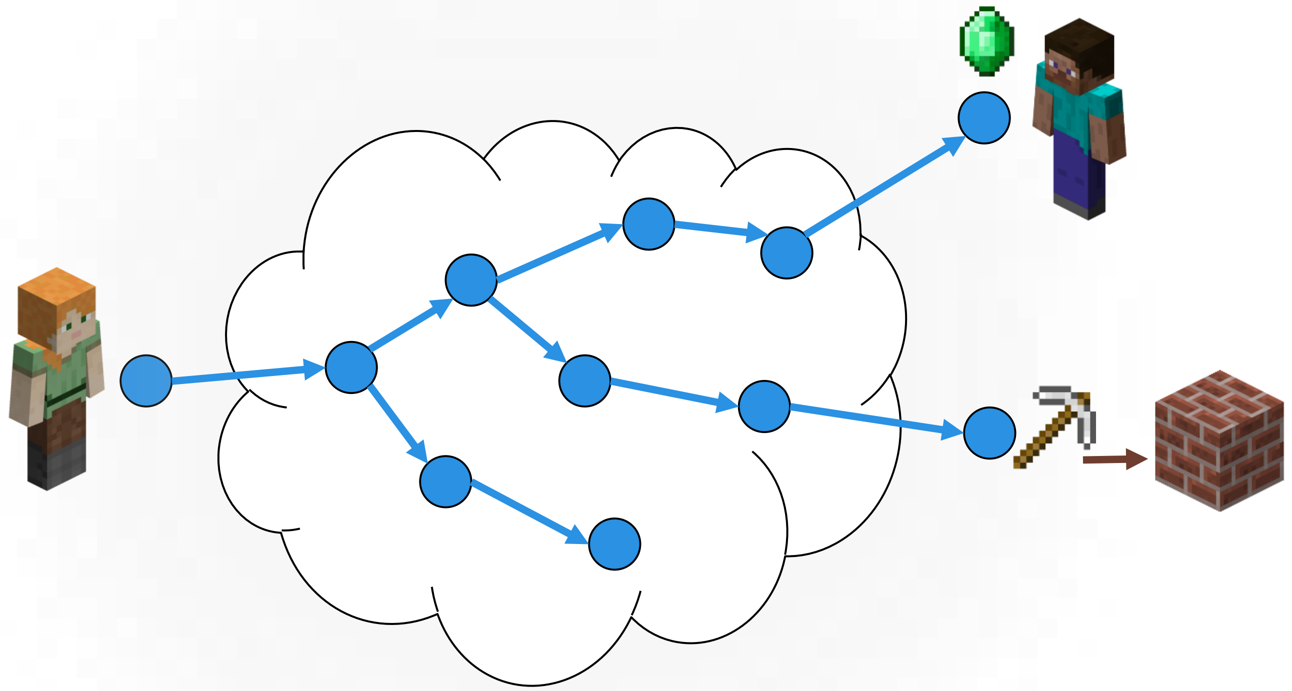
Optimistically verify during propagation.

Proof transmission & verification speed



Optimistically verify during propagation.

Proof transmission & verification speed

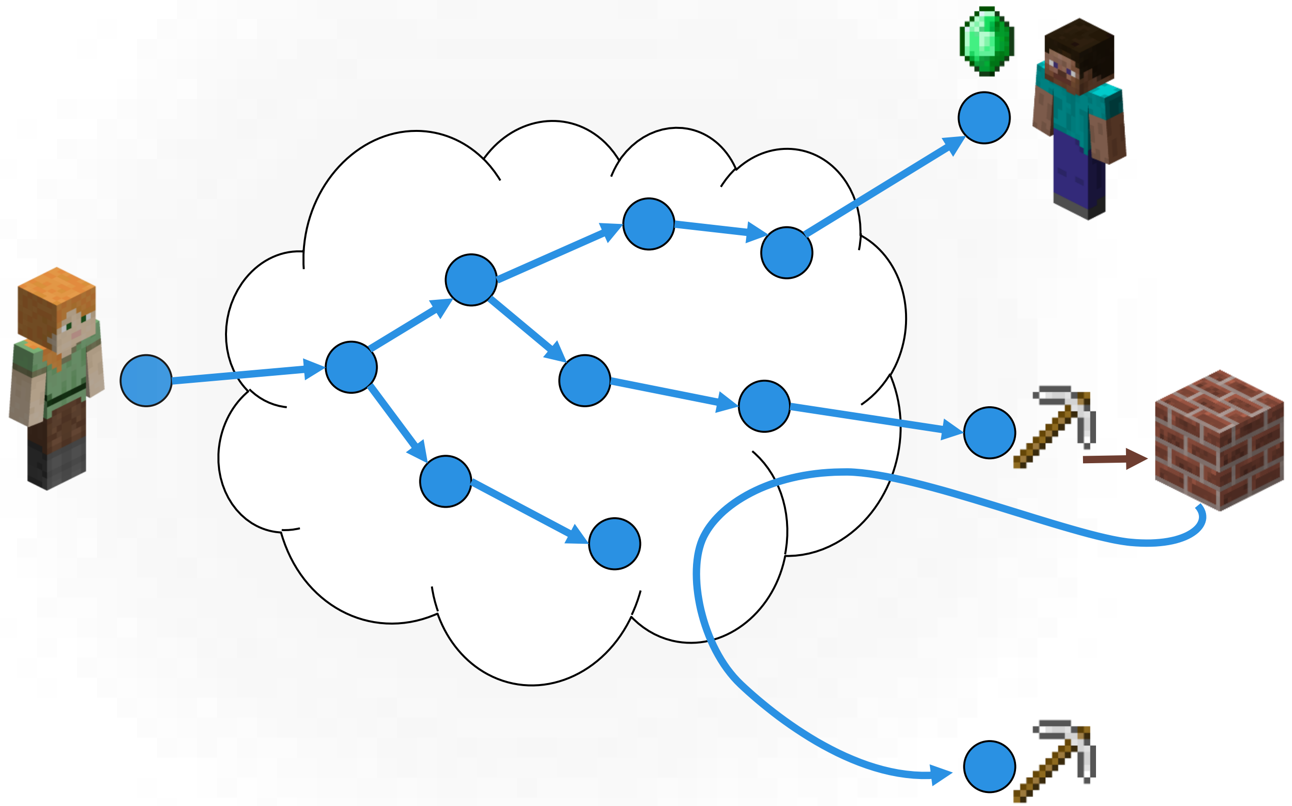


Optimistically verify during propagation.

Optimistically verify tx for inclusion in block template.

Prudently verify later, during PoW.

Proof transmission & verification speed

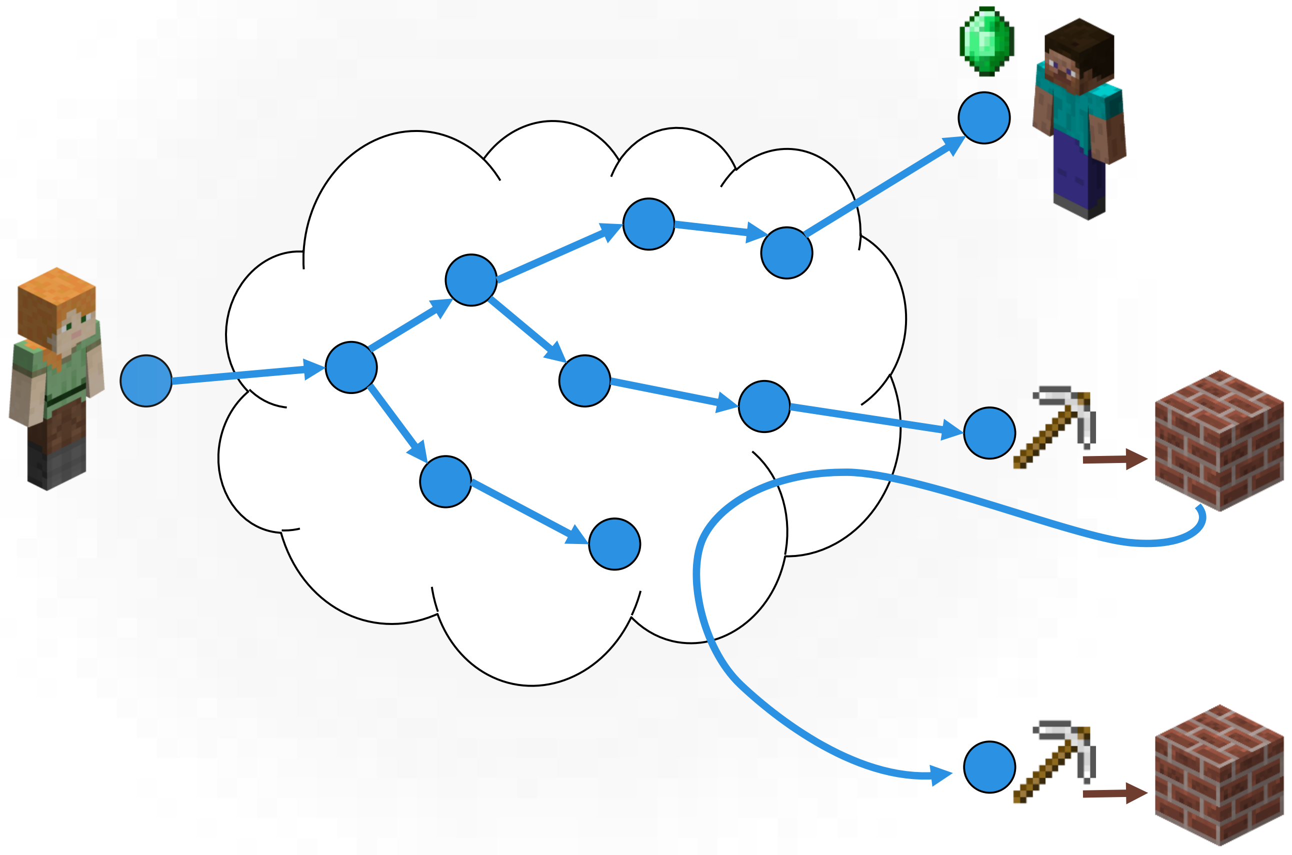


Optimistically verify during propagation.

Optimistically verify tx for inclusion in block template.

Prudently verify later, during PoW.

Proof transmission & verification speed



Optimistically verify during propagation.

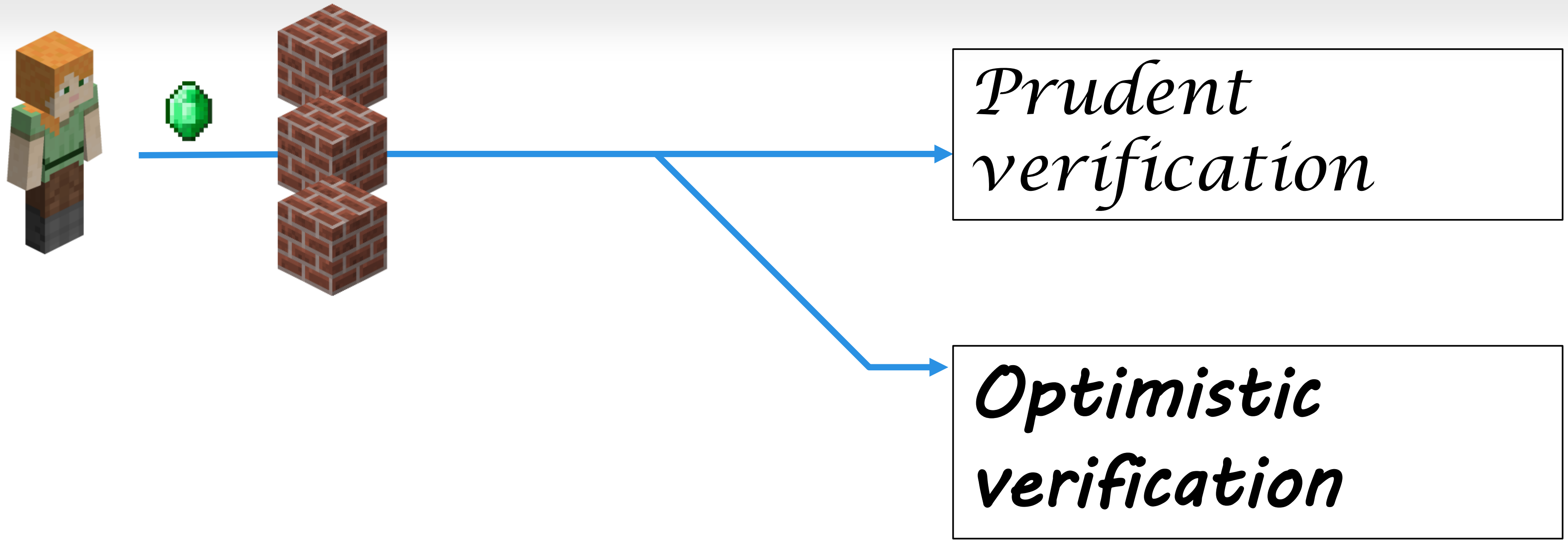
Optimistically verify tx for inclusion in block template.

Prudently verify later, during PoW.

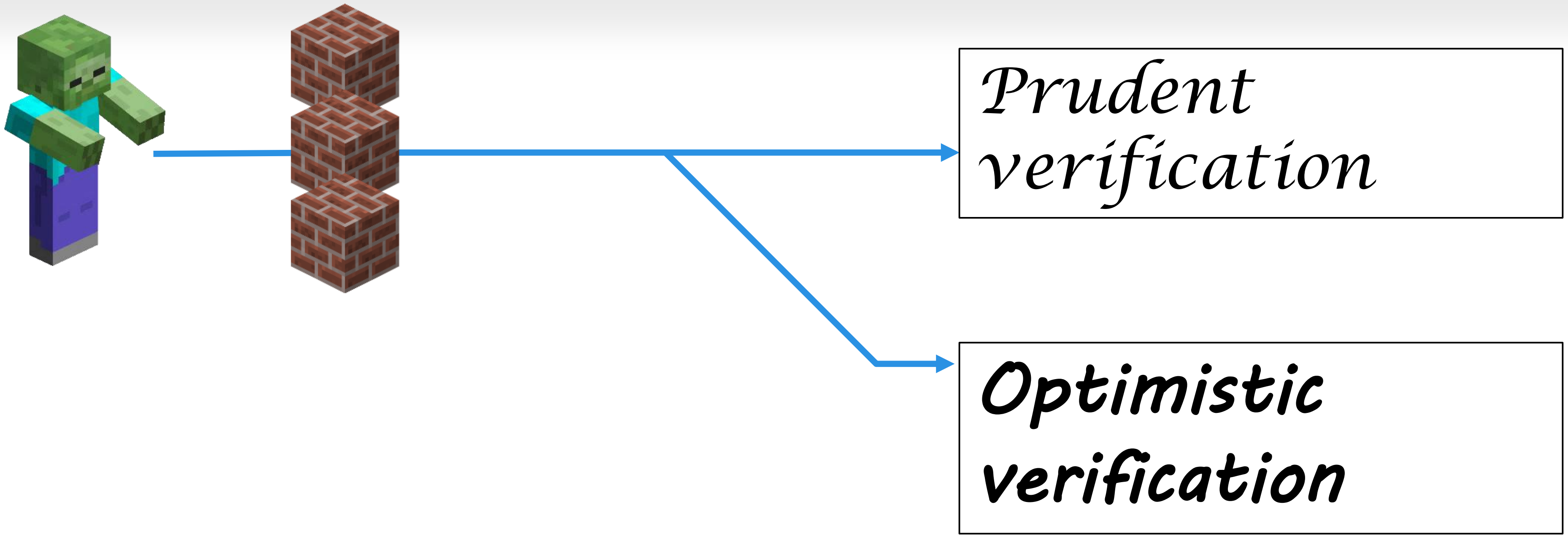
Optimistically verify incoming blocks.

Prudently verify later, during PoW.

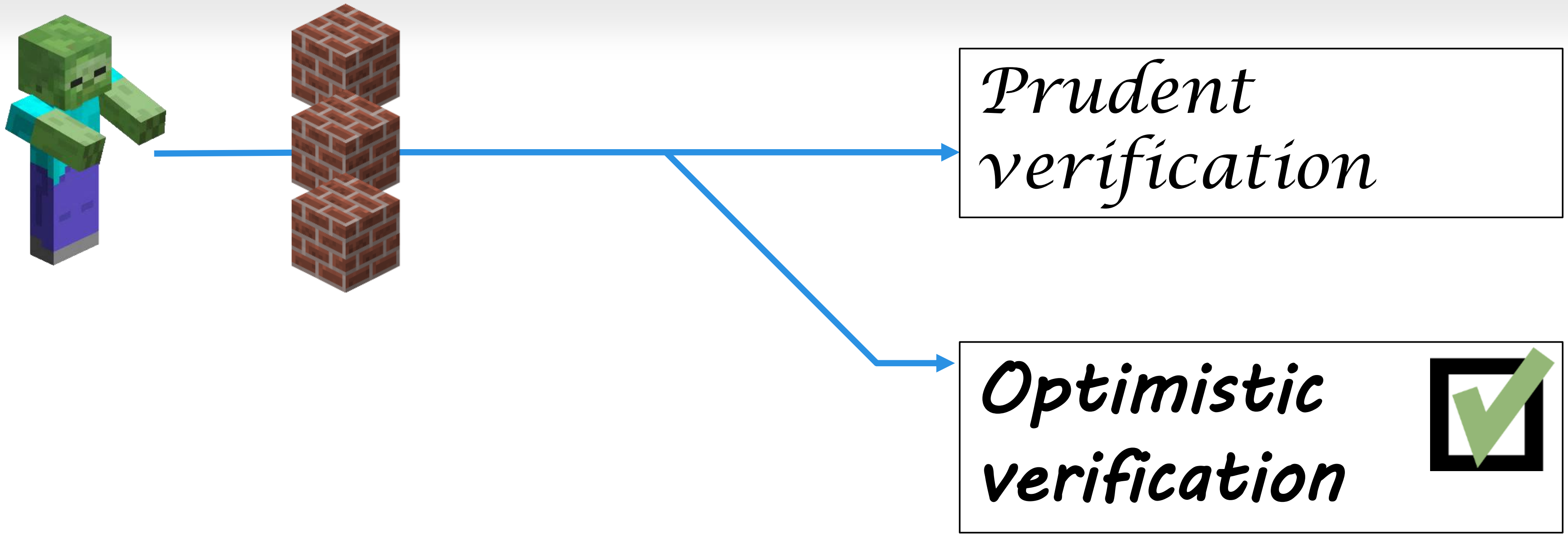
What if prudent verification fails?



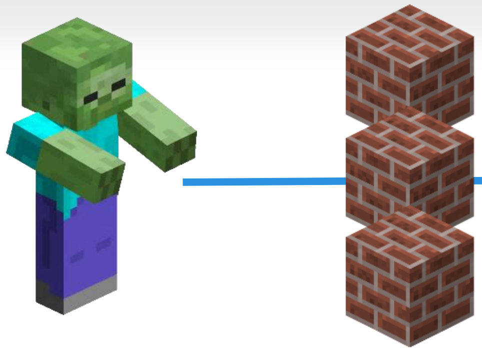
What if prudent verification fails?



What if prudent verification fails?



What if prudent verification fails?

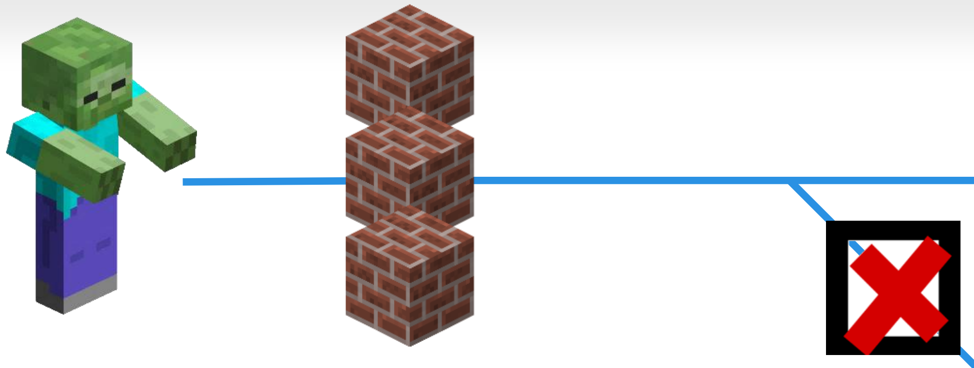


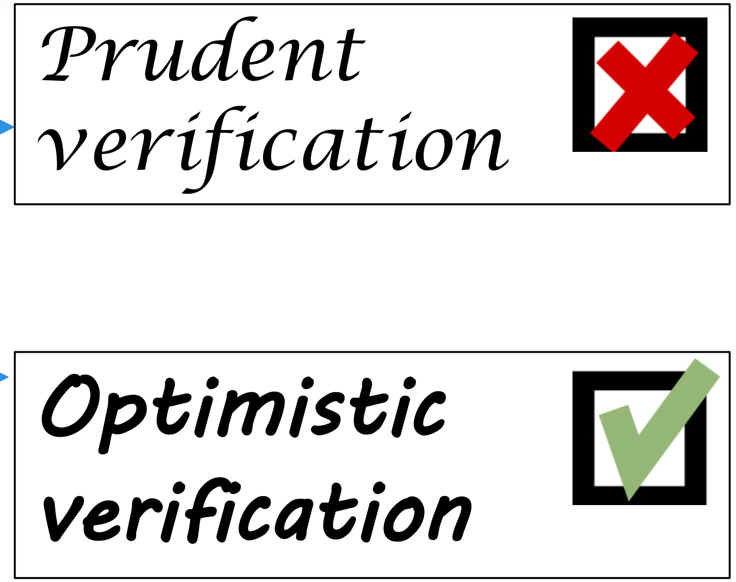
Prudent verification

Optimistic verification

- Detection
  - Real-time

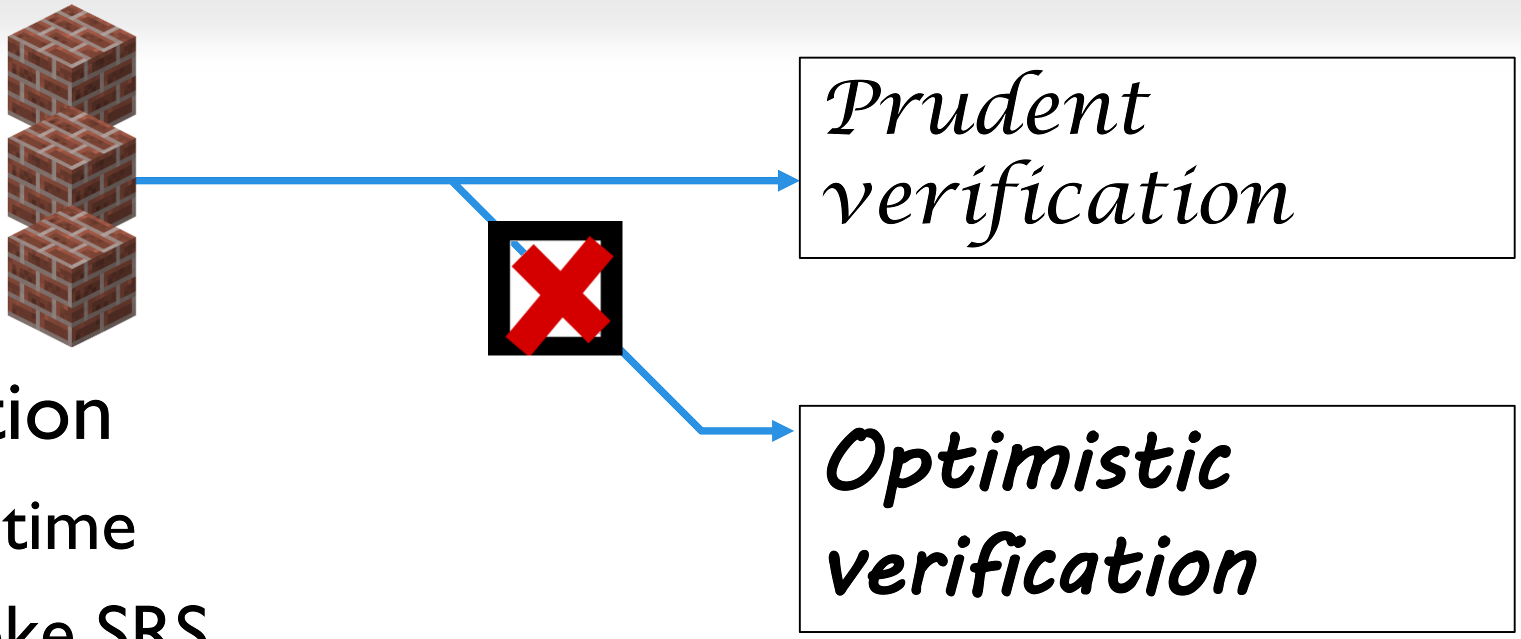
What if prudent verification fails?





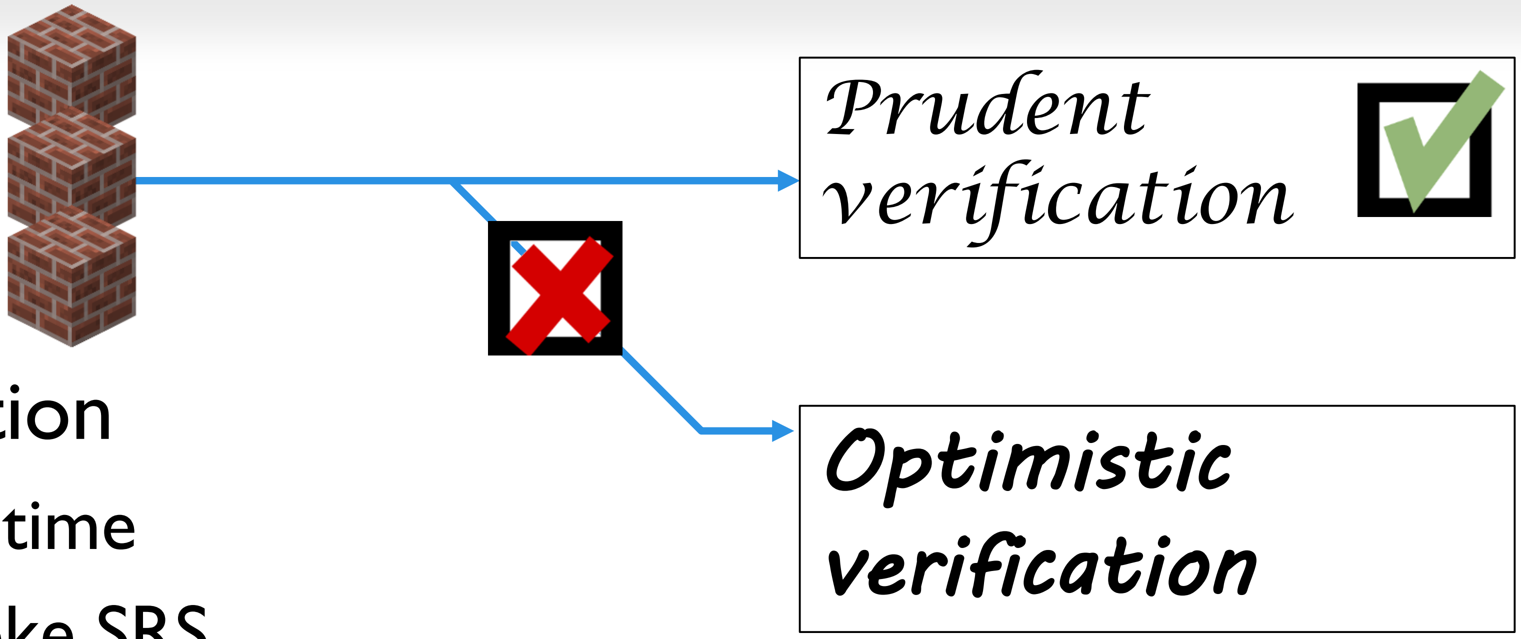
- Detection
  - Real-time
  - Revoke SRS
  - Such pair of proofs is a *fraud proof* for compromised SRS!

What if prudent verification fails?



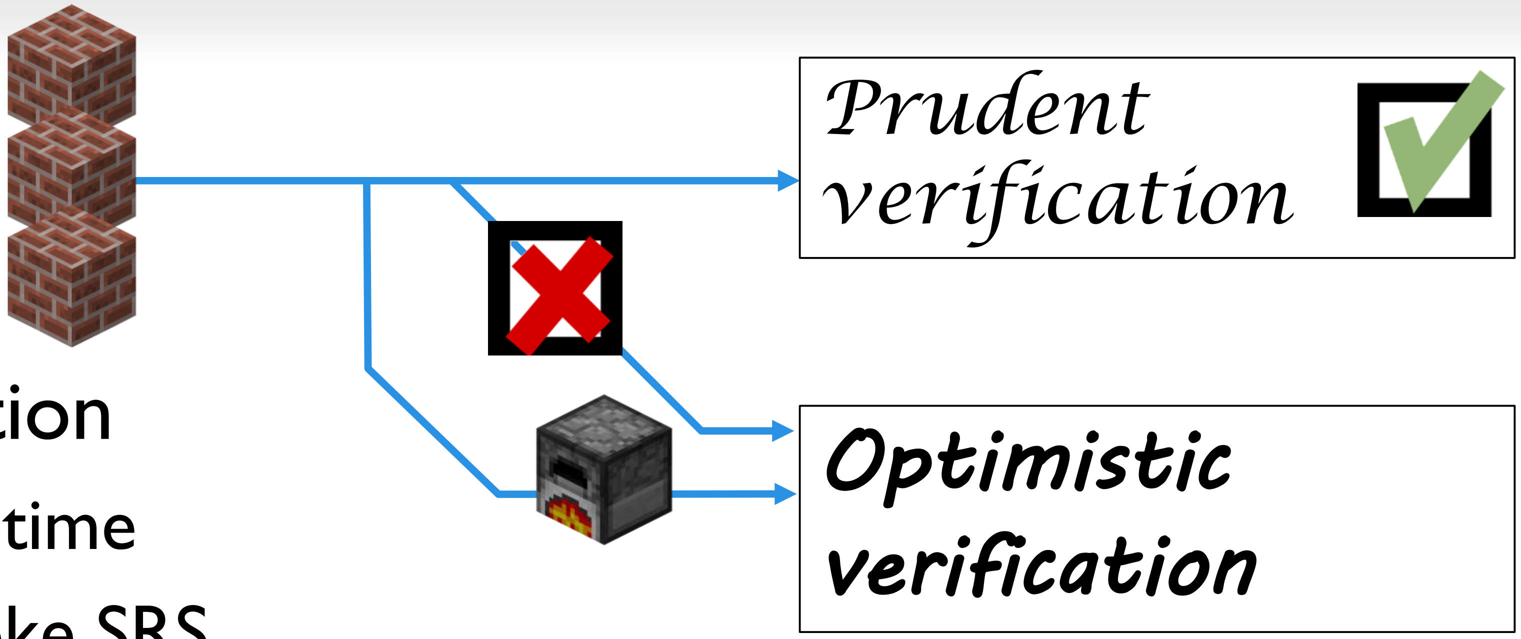
- Detection
  - Real-time
  - Revoke SRS
  - Such pair of proofs is a *fraud proof* for compromised SRS!
- Recovery

What if prudent verification fails?



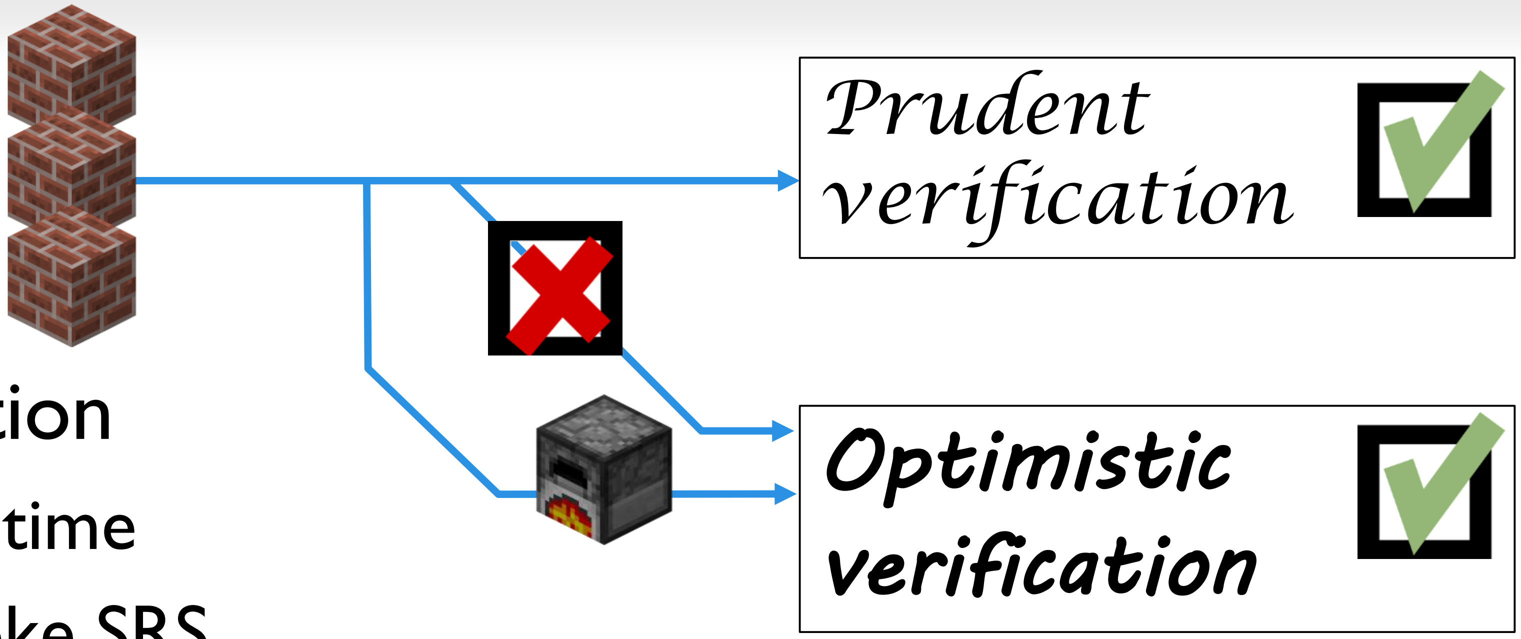
- Detection
  - Real-time
  - Revoke SRS
  - Such pair of proofs is a *fraud proof* for compromised SRS!
- Recovery
  - Soundness: prudently verify

What if prudent verification fails?



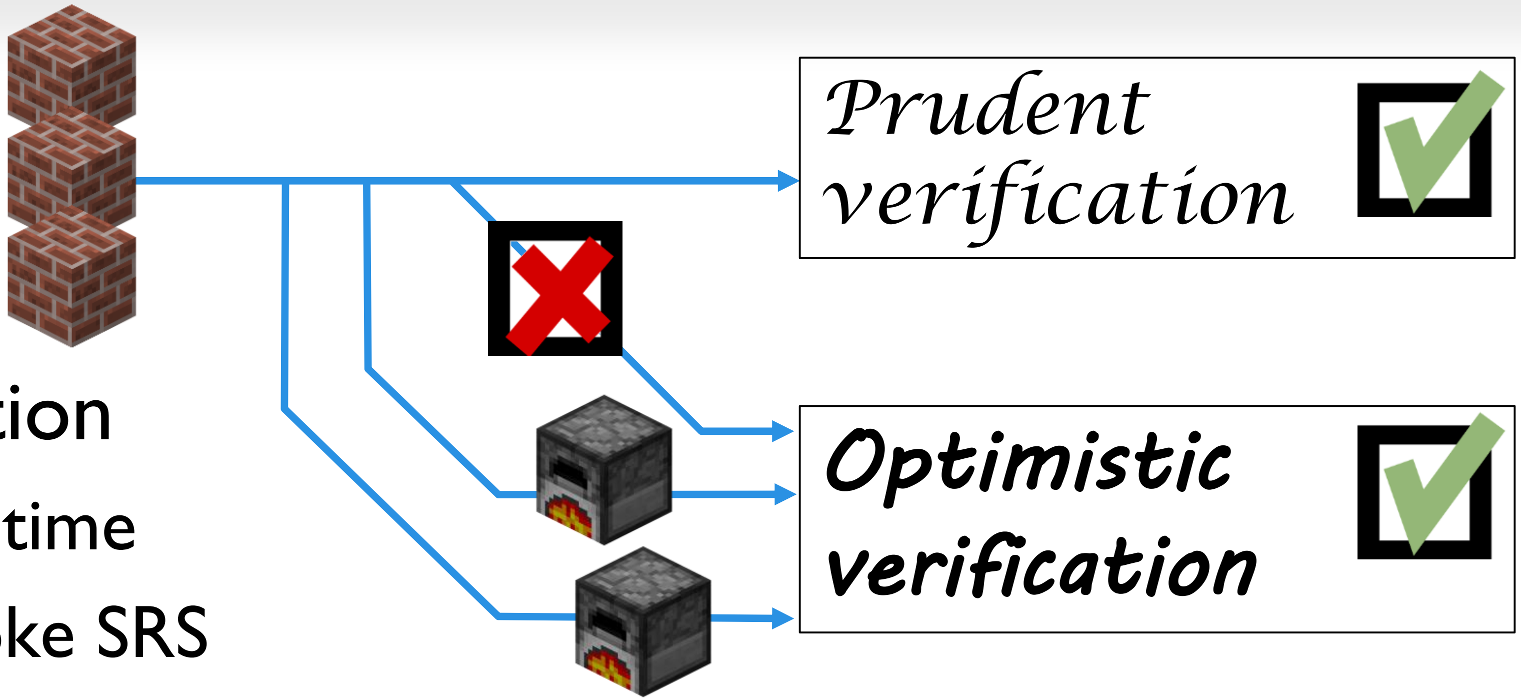
- Detection
  - Real-time
  - Revoke SRS
  - Such pair of proofs is a *fraud proof* for compromised SRS!
- Recovery
  - Soundness: prudently verify
  - Speed: regenerate SRS + refresh the optimistic proofs.

What if prudent verification fails?



- Detection
  - Real-time
  - Revoke SRS
  - Such pair of proofs is a *fraud proof* for compromised SRS!
- Recovery
  - Soundness: prudently verify
  - Speed: regenerate SRS + refresh the optimistic proofs.

What if prudent verification fails?



- Detection
  - Real-time
  - Revoke SRS
  - Such pair of proofs is a *fraud proof* for compromised SRS!
- Recovery
  - Soundness: prudently verify
  - Speed: regenerate SRS + refresh the optimistic proofs.

SHARK requirement: ***anyone can refresh***, without original sender.

A generic SHARK construction

A generic SHARK construction

Attempt – parallel composition:

A generic SHARK construction

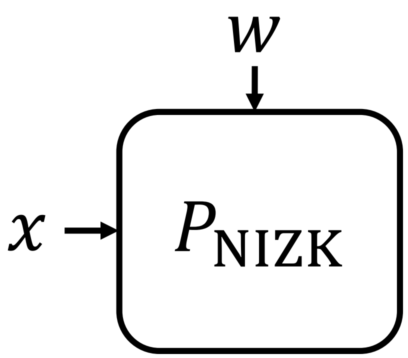
Attempt – parallel composition:

① Fix a language L and construct a NIZK for it:

A generic SHARK construction

Attempt – parallel composition:

① Fix a language L and construct a NIZK for it:

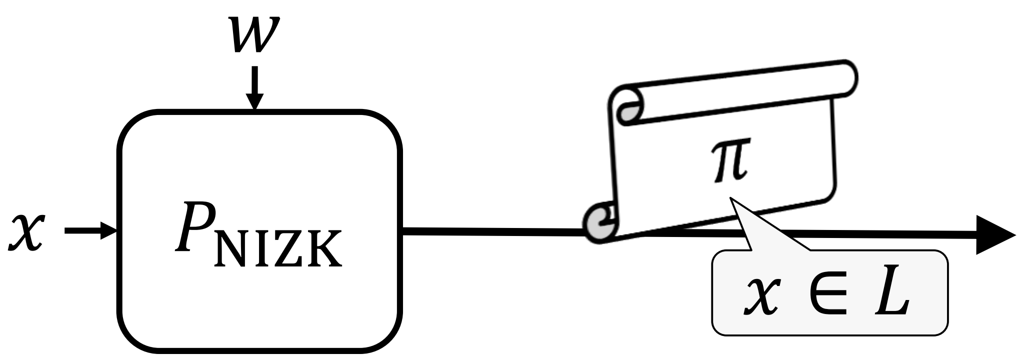


Not pictured: G_{NIZK}

A generic SHARK construction

Attempt – parallel composition:

① Fix a language L and construct a NIZK for it:

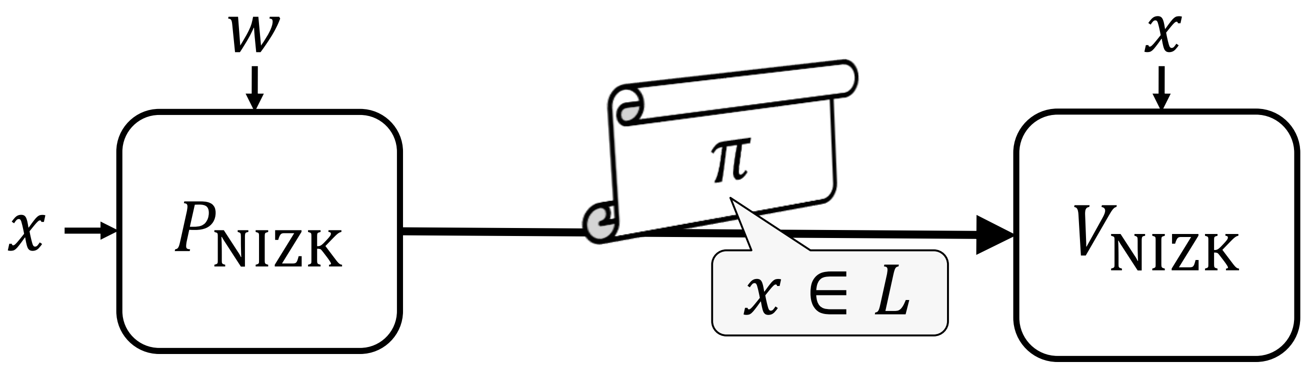


Not pictured: G_{NIZK}

A generic SHARK construction

Attempt – parallel composition:

① Fix a language L and construct a NIZK for it:

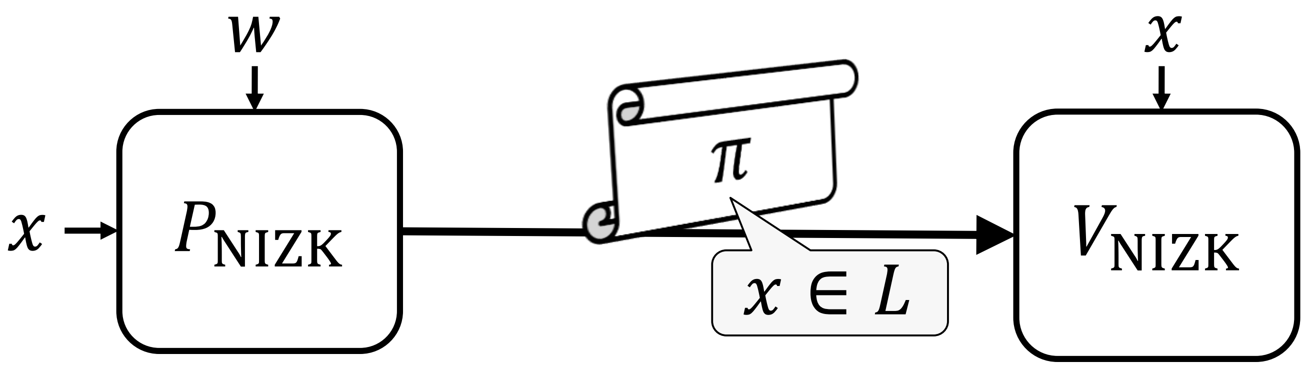


Not pictured: G_{NIZK}

A generic SHARK construction

Attempt – parallel composition:

① Fix a language L and construct a NIZK for it:



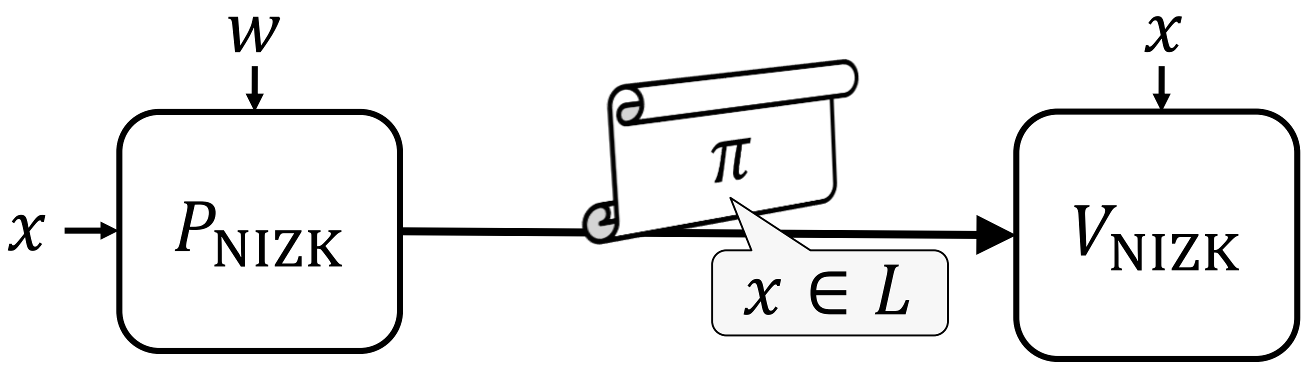
Call π "prudent proof"

Not pictured: G_{NIZK}

A generic SHARK construction

Attempt – parallel composition:

① Fix a language L and construct a NIZK for it:



Call π "prudent proof"

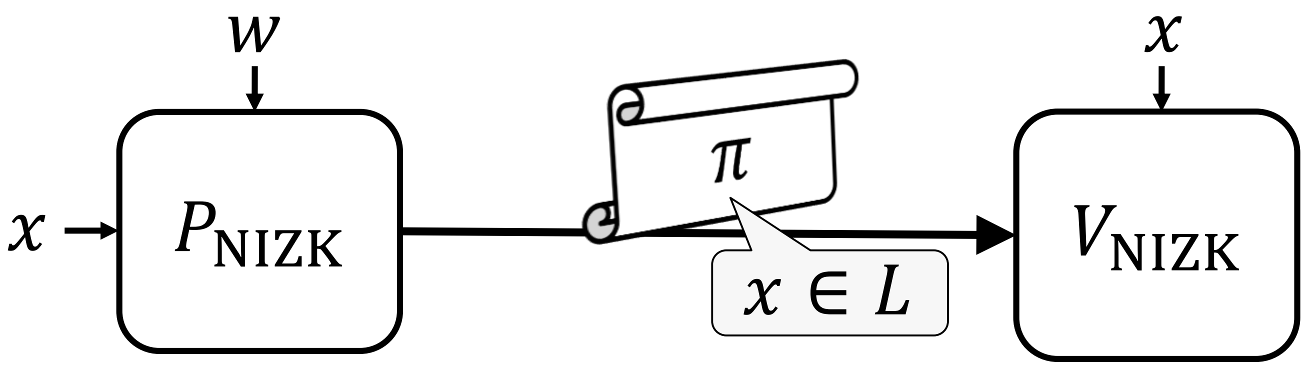
② Construct a SNARK for the same L:

Not pictured: G_{NIZK}

A generic SHARK construction

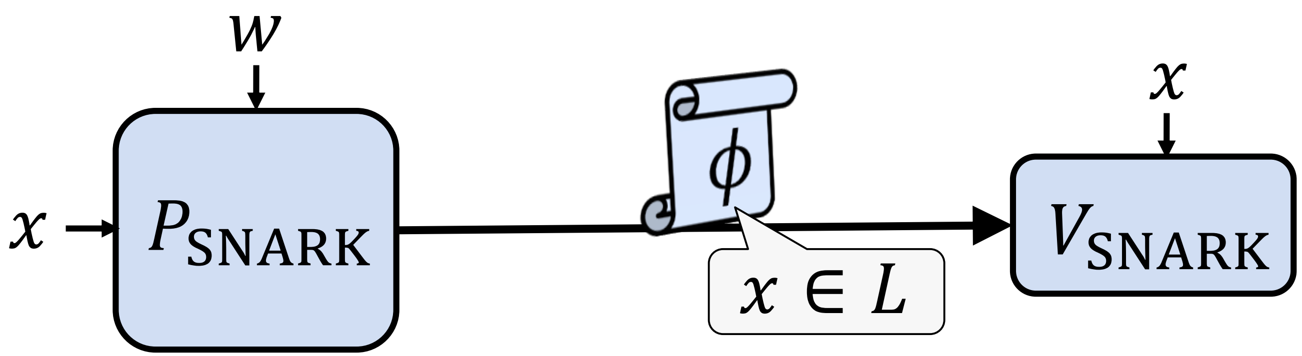
Attempt – parallel composition:

① Fix a language L and construct a NIZK for it:



Call π "prudent proof"

② Construct a SNARK for the same L:

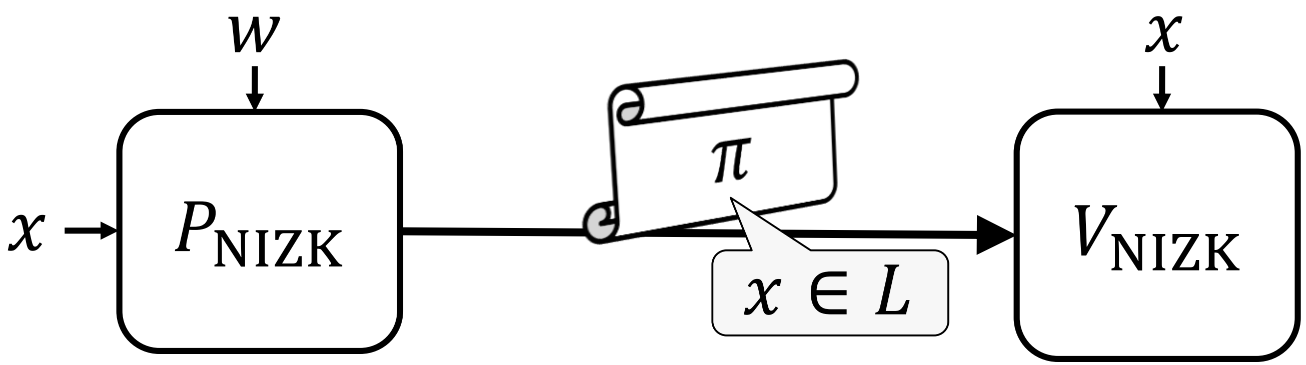


Not pictured: $G_{\mathrm{NIZK}}, G_{\mathrm{SNARK}}$

A generic SHARK construction

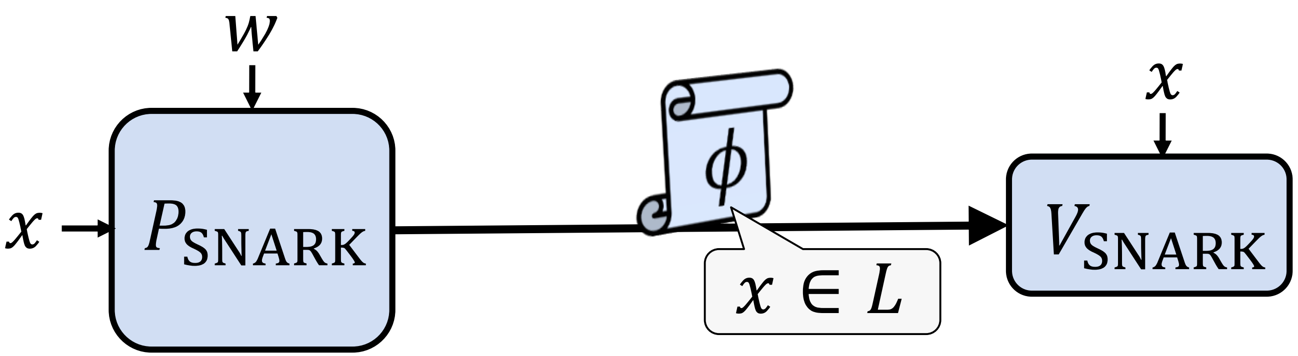
Attempt – parallel composition:

① Fix a language L and construct a NIZK for it:



Call π "prudent proof"

② Construct a SNARK for the same L:



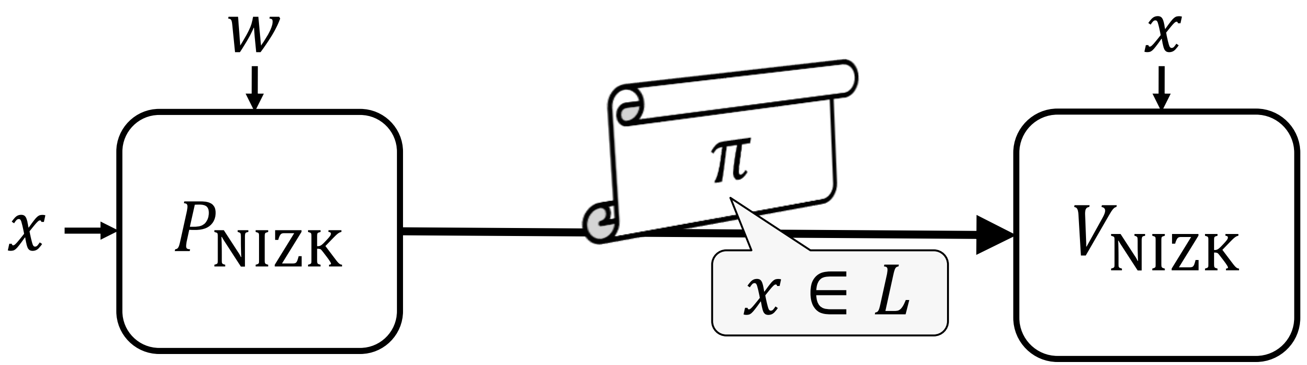
Optimistic proofs should be refreshable without w

Not pictured: $G_{\text{NIZK}}, G_{\text{SNARK}}$

A generic SHARK construction

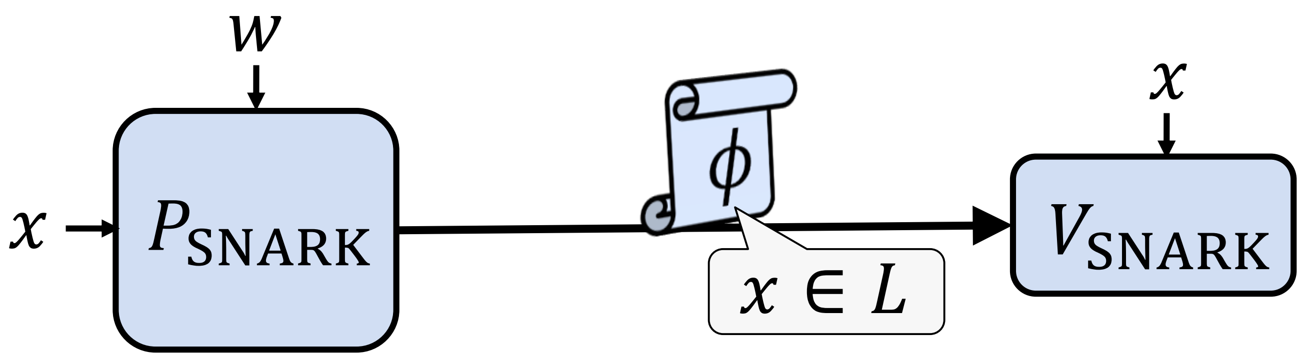
Attempt – parallel composition:

① Fix a language L and construct a NIZK for it:



Call π "prudent proof"

② Construct a SNARK for the same L:



Optimistic proofs should be refreshable without w

$\Rightarrow$ ϕ is **not** a SHARK optimistic proof

Not pictured: $G_{\text{NIZK}}, G_{\text{SNARK}}$

A generic SHARK construction

A generic SHARK construction

Construction:

A generic SHARK construction

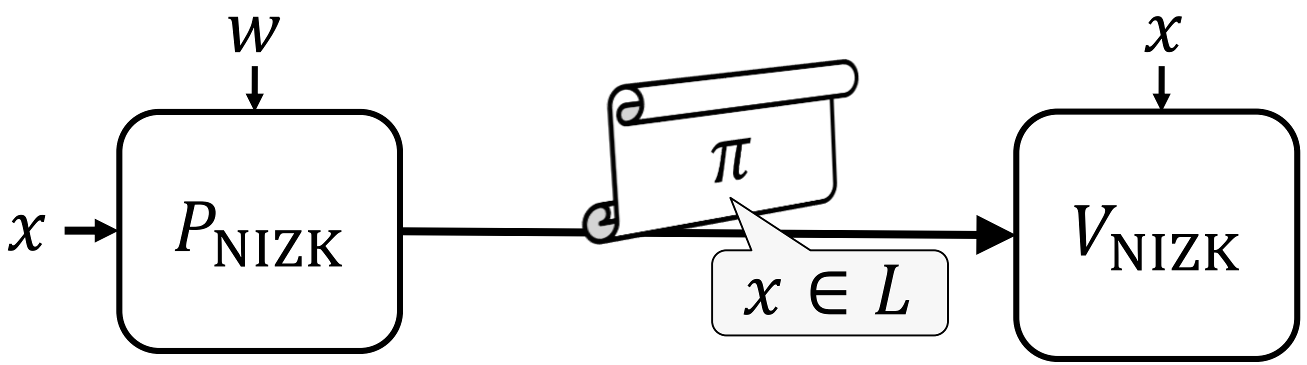
Construction:

1 Fix a language L and construct a NIZK for it:

A generic SHARK construction

Construction:

① Fix a language L and construct a NIZK for it:

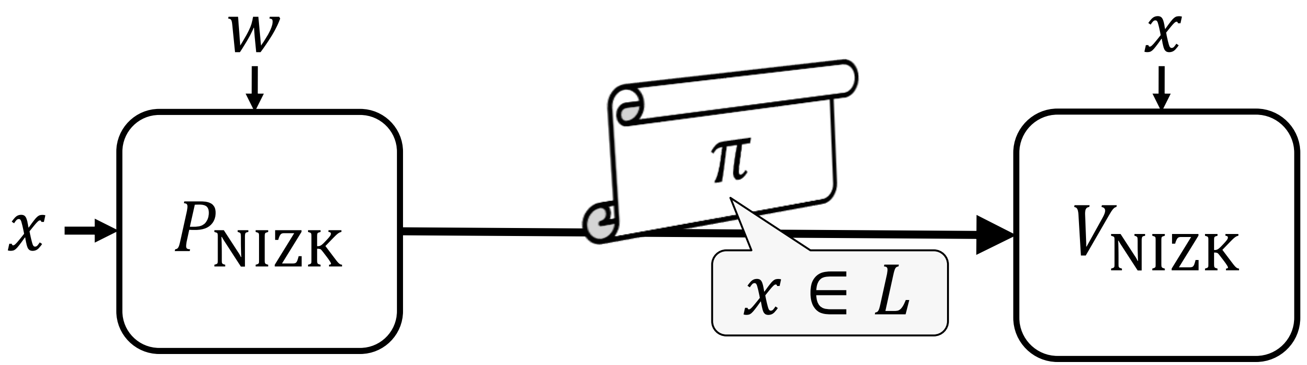


Not pictured: G_{NIZK}

A generic SHARK construction

Construction:

① Fix a language L and construct a NIZK for it:



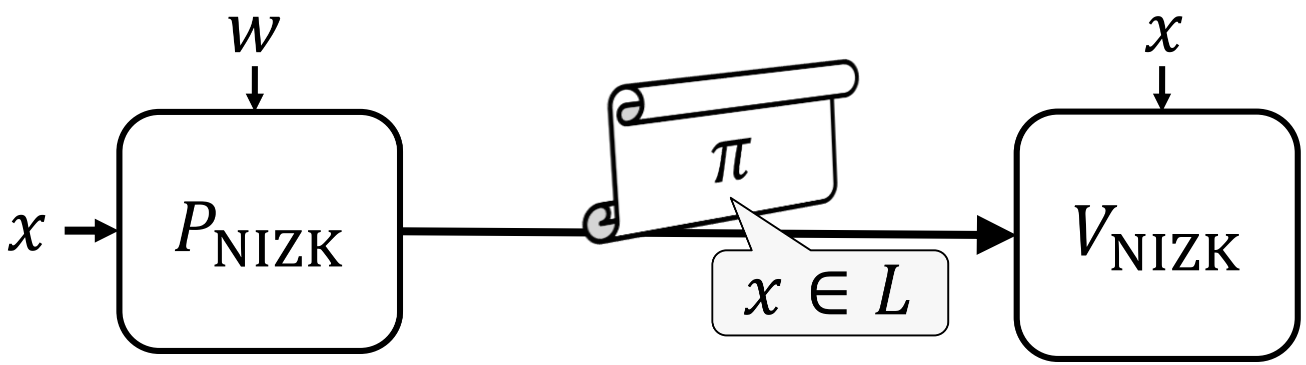
Call π “prudent proof”

Not pictured: G_{NIZK}

A generic SHARK construction

Construction:

① Fix a language L and construct a NIZK for it:



Call π "prudent proof"

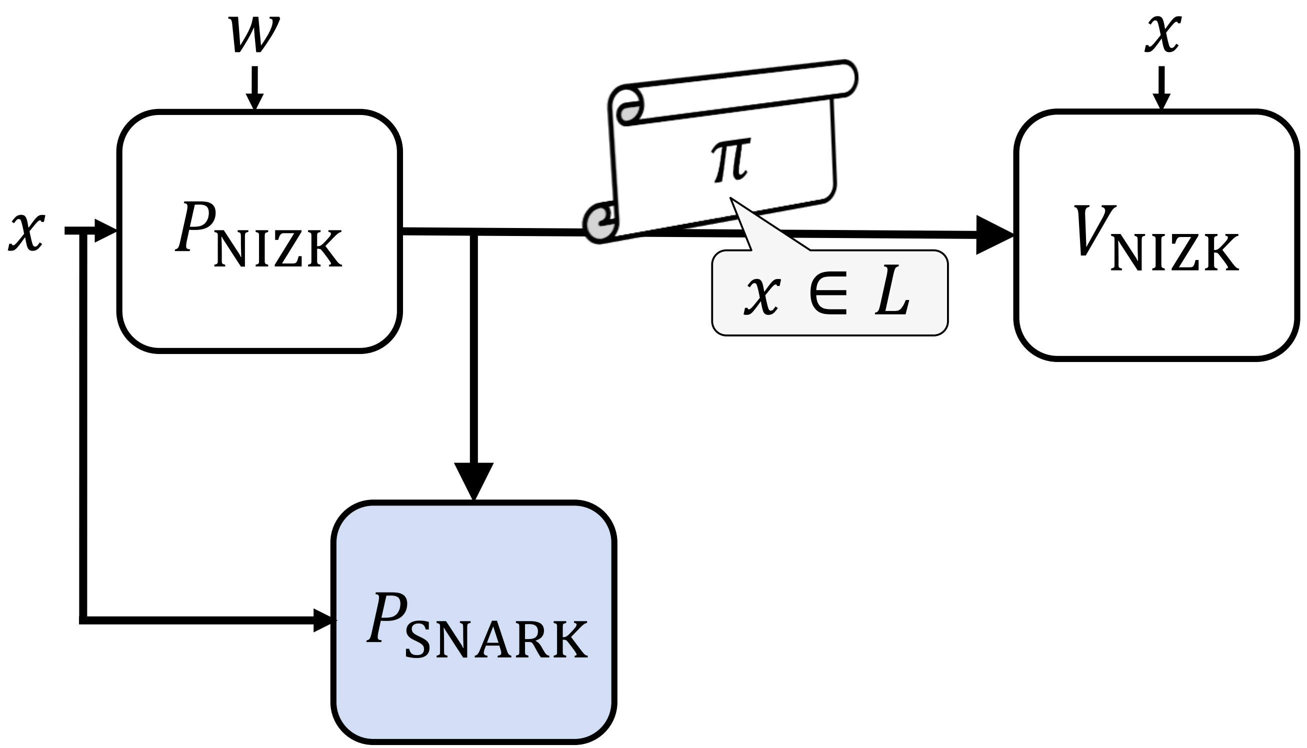
② Construct a SNARK for "V_{NIZK} accepts"

Not pictured: G_{NIZK}

A generic SHARK construction

Construction:

① Fix a language L and construct a NIZK for it:



Call π "prudent proof"

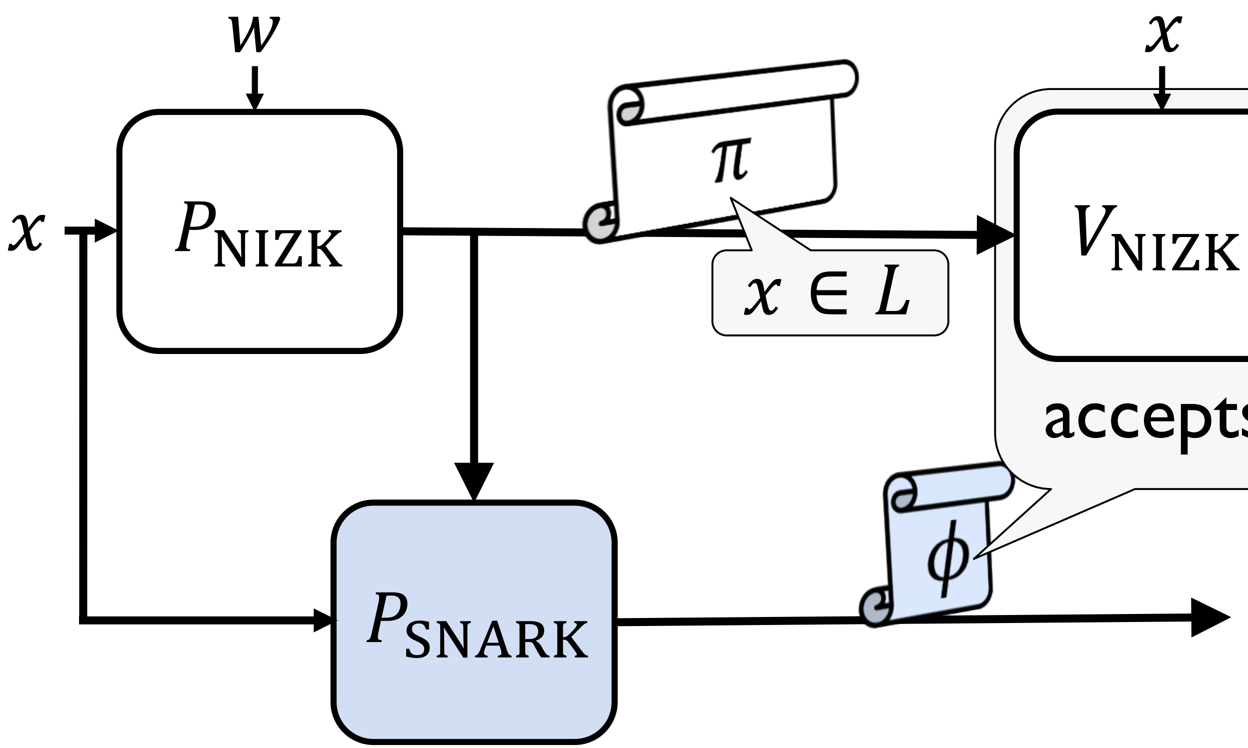
② Construct a SNARK for "V_{NIZK} accepts"

Not pictured: G_{NIZK}, G_{SNARK}

A generic SHARK construction

Construction:

① Fix a language L and construct a NIZK for it:



Call π "prudent proof"

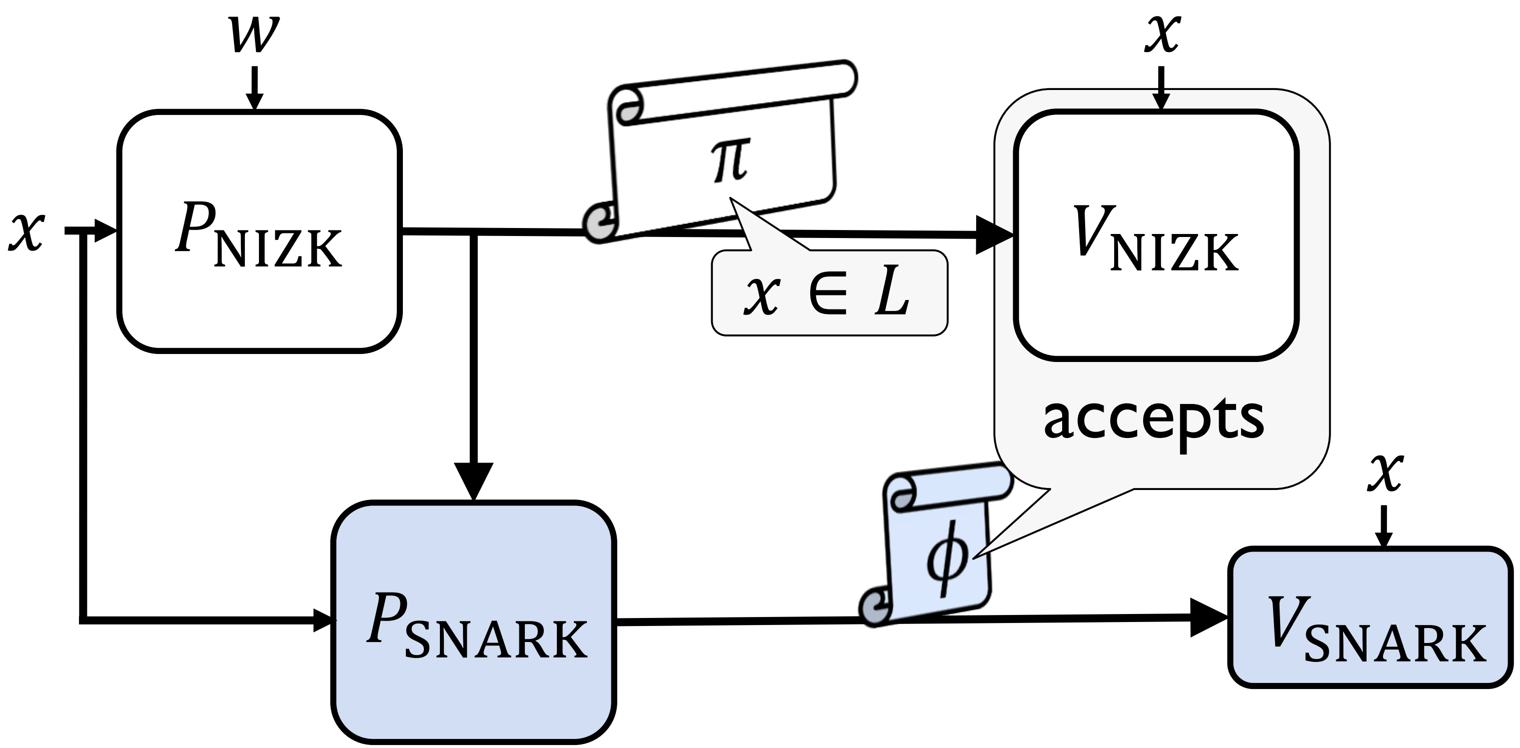
② Construct a SNARK for "V_{NIZK} accepts"

Not pictured: G_{NIZK}, G_{SNARK}

A generic SHARK construction

Construction:

① Fix a language L and construct a NIZK for it:



Call π "prudent proof"

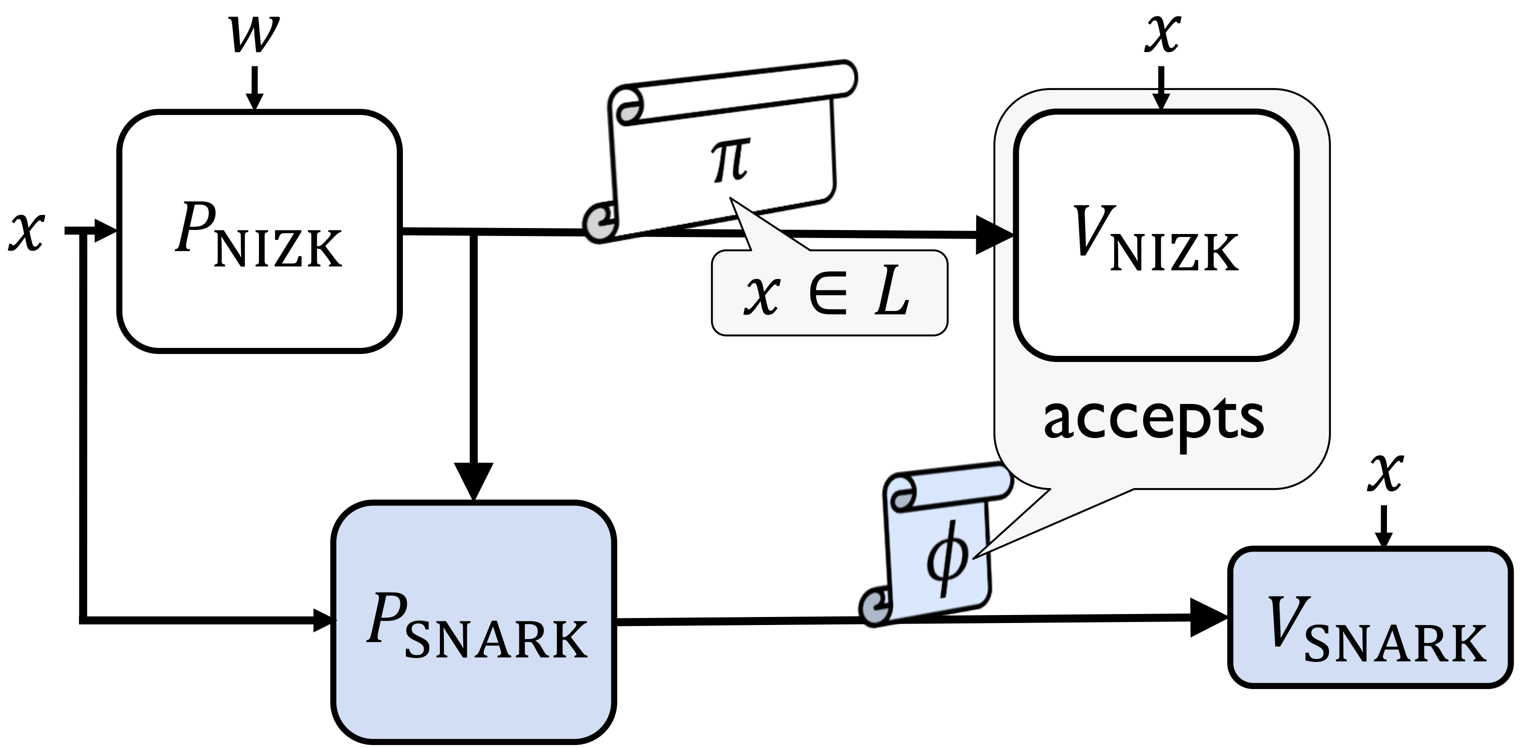
② Construct a SNARK for "V_{NIZK} accepts"

Not pictured: G_{NIZK}, G_{SNARK}

A generic SHARK construction

Construction:

① Fix a language L and construct a NIZK for it:



Call π "prudent proof"

Call ϕ "optimistic proof"

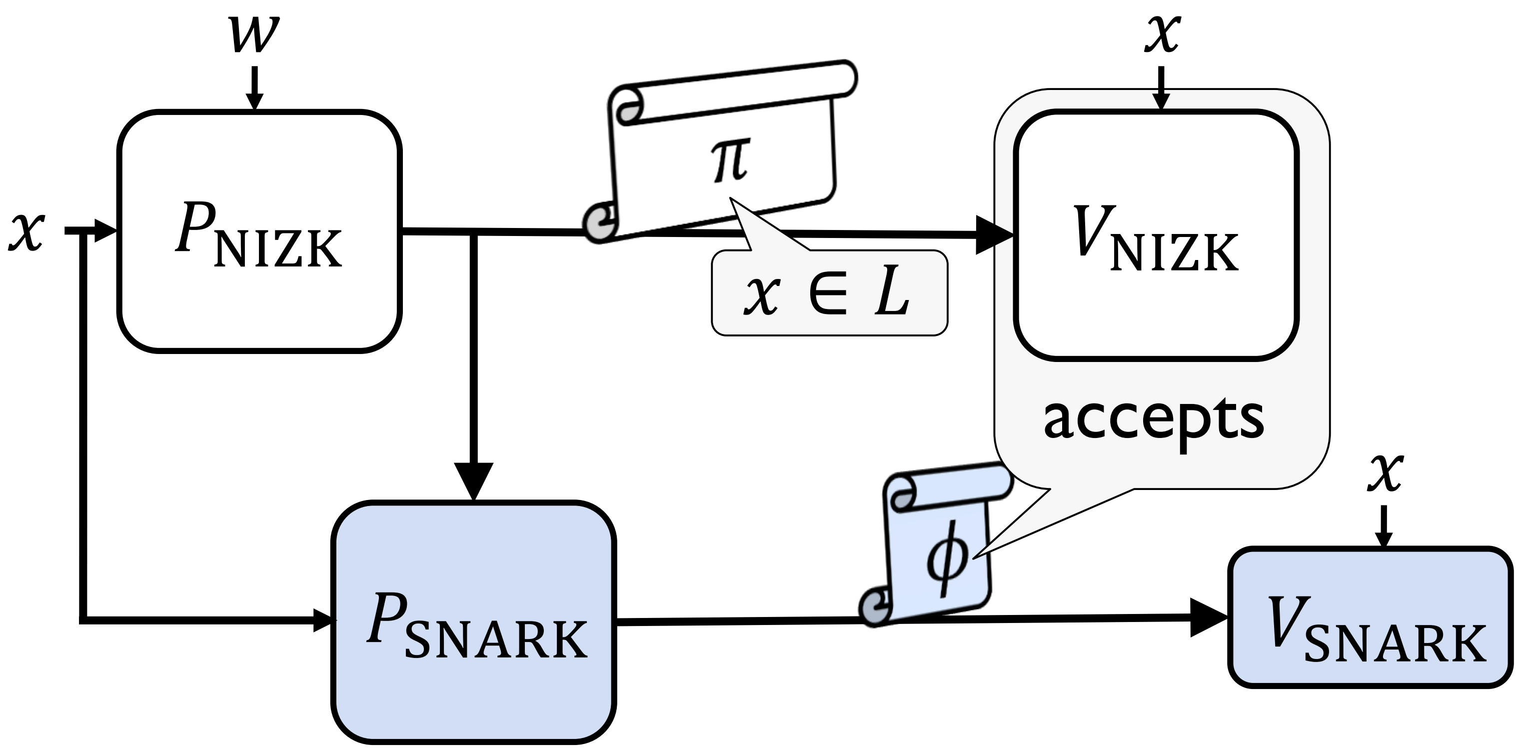
② Construct a SNARK for "V_{NIZK} accepts"

Not pictured: $G_{\mathrm{NIZK}}, G_{\mathrm{SNARK}}$

A generic SHARK construction

Construction:

① Fix a language L and construct a NIZK for it:



Call π "prudent proof"

Call ϕ "optimistic proof"

② Construct a SNARK for "V_{NIZK} accepts"

Not pictured: G_{NIZK}, G_{SNARK}

Challenge: efficient SHARK construction

Challenge: efficient SHARK construction

Generically combining
state-of-the-art components:

Challenge: efficient SHARK construction

Generically combining

state-of-the-art components:

- Bulletproofs NIZK + Groth16 SNARK

Challenge: efficient SHARK construction

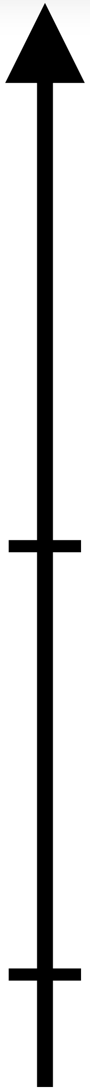
$O(|C|)$-size circuit

Generically combining

state-of-the-art components:

- Bulletproofs NIZK + Groth16 SNARK

Challenge: efficient SHARK construction



Generically combining state-of-the-art components:

- Bulletproofs NIZK + Groth16 SNARK

Challenge: efficient SHARK construction

$O(\lambda|C|)$ group ops in V_{NIZK}

$O(|C|)$-size circuit

Generically combining state-of-the-art components:

- Bulletproofs NIZK + Groth16 SNARK

some speed-up via multiexp

Challenge: efficient SHARK construction

$O(\lambda^2|C|)$ group ops in P_{SNARK}

$O(\lambda|C|)$ group ops in V_{NIZK}

$O(|C|)$-size circuit

Generically combining state-of-the-art components:

- Bulletproofs NIZK + Groth16 SNARK

some speed-up via multiexp

Challenge: efficient SHARK construction

$O(\lambda^2|C|)$ group ops in P_{SNARK}

$O(\lambda|C|)$ group ops in V_{NIZK}

$O(|C|)$-size circuit

Generically combining state-of-the-art components:

- Bulletproofs NIZK + Groth16 SNARK

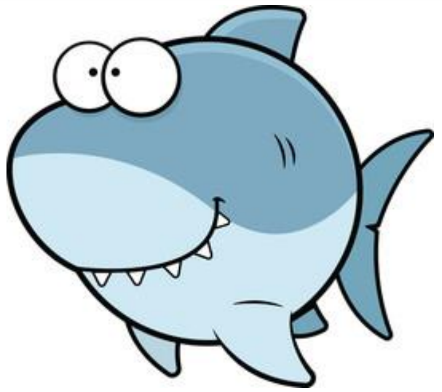
some speed-up via multiexp

Challenge: efficient SHARK construction

$O(\lambda^2|C|)$ group ops in P_{SNARK}

$O(\lambda|C|)$ group ops in V_{NIZK}

$O(|C|)$-size circuit



Generically combining state-of-the-art components:

- Bulletproofs NIZK + Groth16 SNARK

some speed-up via multiexp

Technical contribution: efficient SHARK construction

Technical contribution: efficient SHARK construction

NIZK for R1CS

Technical contribution: efficient SHARK construction

$\approx$ "arithmetic circuits with bilinear gates"

NIZK for R1CS

Technical contribution: efficient SHARK construction

$\approx$ "arithmetic circuits with bilinear gates"

NIZK for R1CS

- a new compilation technique for linear PCPs

Technical contribution: efficient SHARK construction

$\approx$ "arithmetic circuits with bilinear gates"

NIZK for R1CS

- a new compilation technique for linear PCPs
- public coin NIZK from LPCPs $\Rightarrow$ prudent mode

Technical contribution: efficient SHARK construction

$\approx$ "arithmetic circuits with bilinear gates"

NIZK for R1CS

- a new compilation technique for linear PCPs
- public coin NIZK from LPCPs $\Rightarrow$ prudent mode
- an optimized variant of Bulletproofs' inner product argument

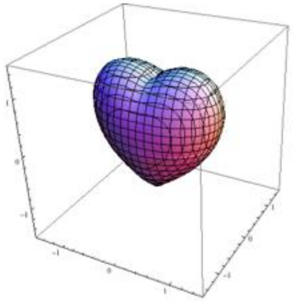
Technical contribution: efficient SHARK construction

$\approx$ "arithmetic circuits with bilinear gates"

NIZK for R1CS

- a new compilation technique for linear PCPs
- public coin NIZK from LPCPs $\Rightarrow$ prudent mode
- an optimized variant of Bulletproofs' inner product argument

$\Rightarrow V_{\text{NIZK}}$ has an "algebraic heart"



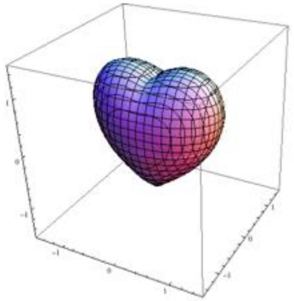
Technical contribution: efficient SHARK construction

$\approx$ "arithmetic circuits with bilinear gates"

NIZK for R1CS

- a new compilation technique for linear PCPs
- public coin NIZK from LPCPs $\Rightarrow$ prudent mode
- an optimized variant of Bulletproofs' inner product argument

$\Rightarrow V_{\mathrm{NIZK}}$ has an "algebraic heart"



A *special-purpose* SNARK

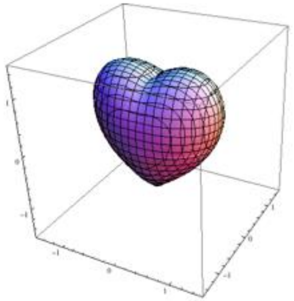
Technical contribution: efficient SHARK construction

$\approx$ "arithmetic circuits with bilinear gates"

NIZK for R1CS

- a new compilation technique for linear PCPs
- public coin NIZK from LPCPs $\Rightarrow$ prudent mode
- an optimized variant of Bulletproofs' inner product argument

$\Rightarrow V_{\mathrm{NIZK}}$ has an "algebraic heart"



A *special-purpose* SNARK

for "encoded polynomial delegation", a problem we introduce

Technical contribution: efficient SHARK construction

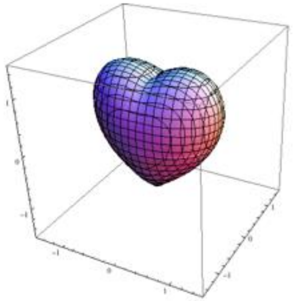
≈ "arithmetic circuits with bilinear gates"

Our SHARK

NIZK for R1CS

- a new compilation technique for linear PCPs
- public coin NIZK from LPCPs ⇒ prudent mode
- an optimized variant of Bulletproofs' inner product argument

⇒ V_{NIZK} has an "algebraic heart"



A *special-purpose* SNARK

for "encoded polynomial delegation", a problem we introduce

Linear PCP paradigm

[GGPR12]
[BCIOP13]

Linear PCP paradigm

[GGPR12]
[BCIOP13]

① Design a proof system sound against linear provers

Linear PCP paradigm

[GGPR12]
[BCIOP13]

① Design a proof system sound against linear provers

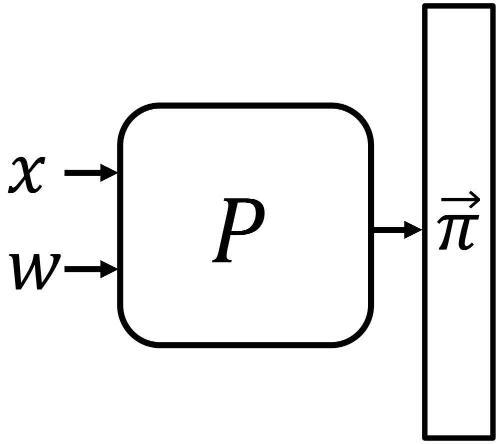
② Force prover to be linear using a cryptographic encoding

Linear PCP paradigm

[GGPR12]
[BCIOP13]

① Design a proof system sound against linear provers

② Force prover to be linear using a cryptographic encoding

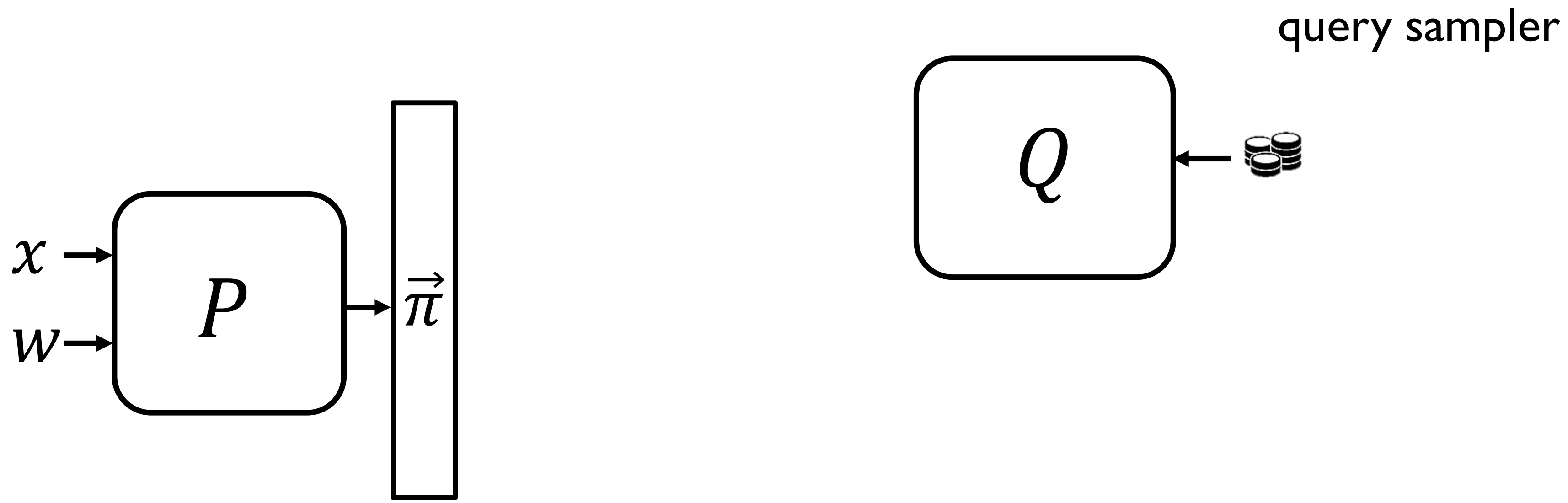


Linear PCP paradigm

[GGPR12]
[BCIOP13]

① Design a proof system sound against linear provers

② Force prover to be linear using a cryptographic encoding

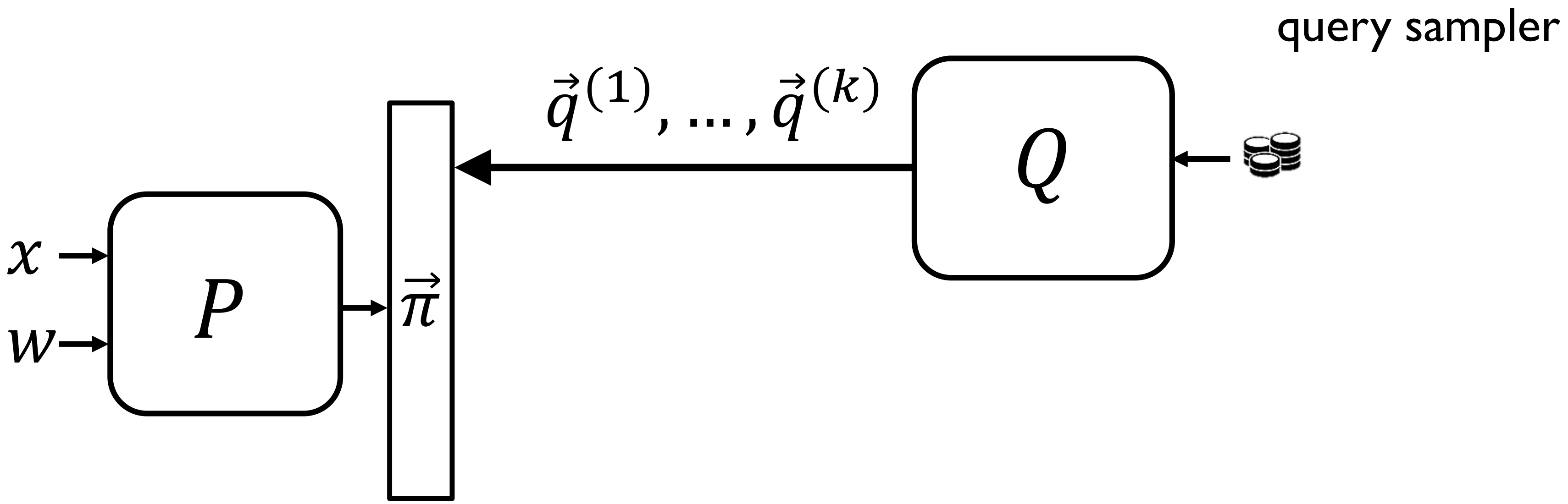


Linear PCP paradigm

[GGPR12]
[BCIOP13]

① Design a proof system sound against linear provers

② Force prover to be linear using a cryptographic encoding

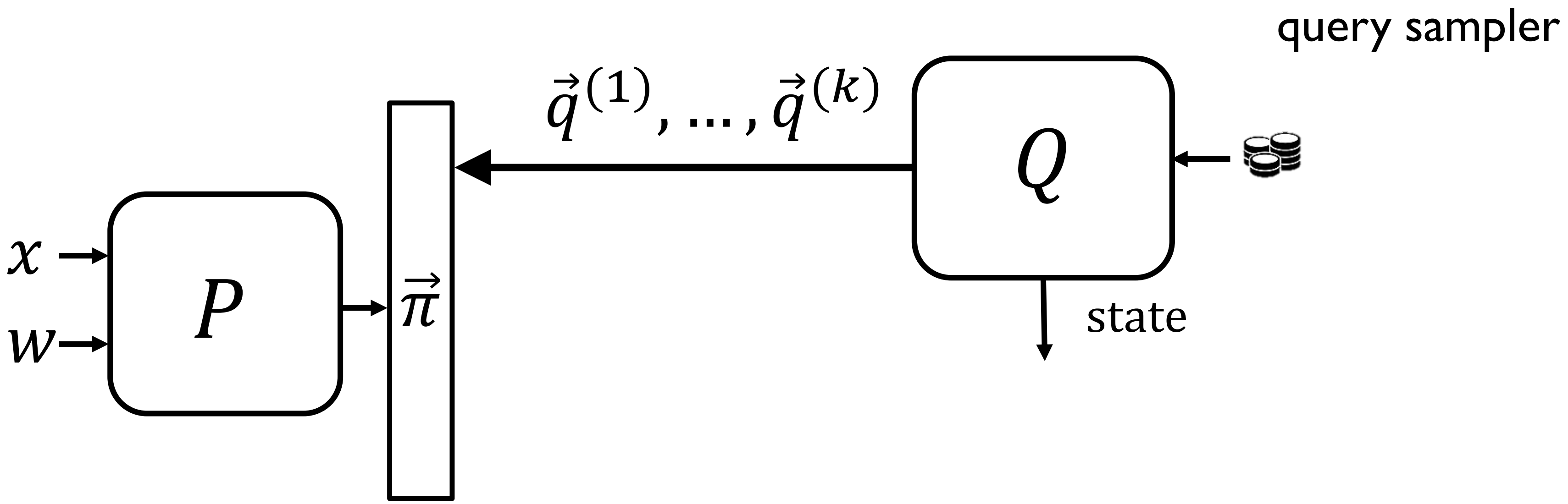


Linear PCP paradigm

[GGPR12]
[BCIOP13]

① Design a proof system sound against linear provers

② Force prover to be linear using a cryptographic encoding

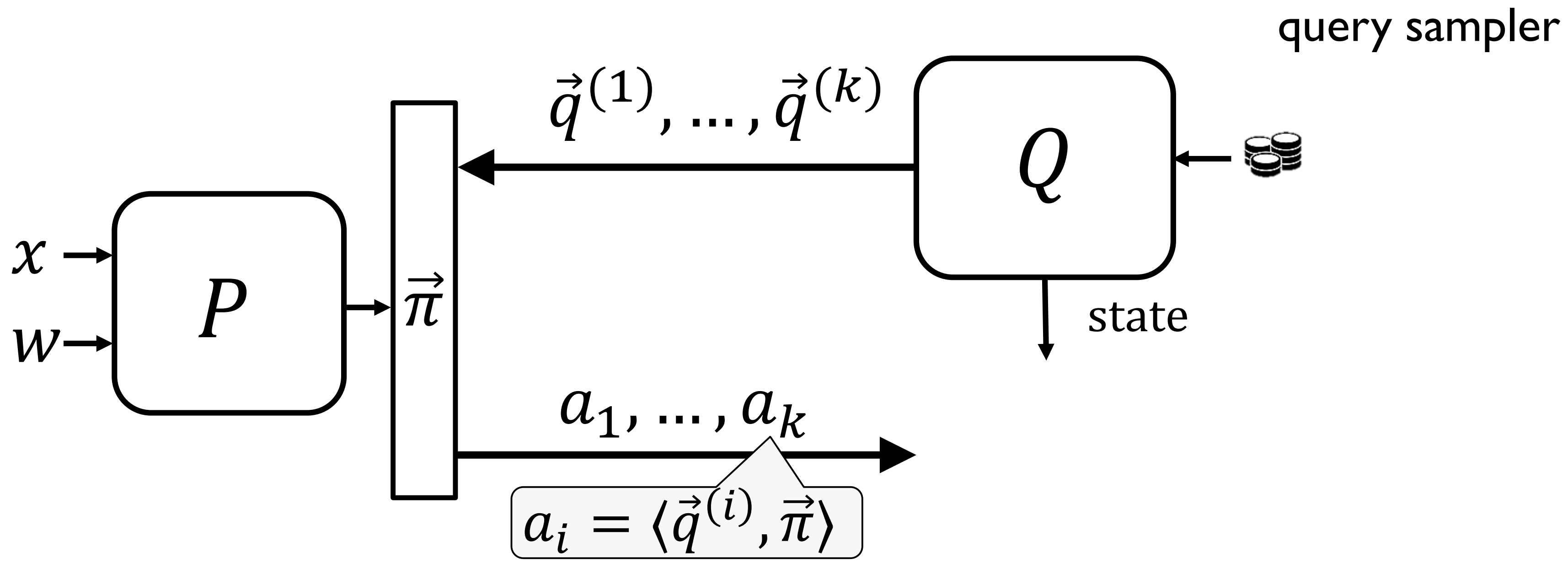


Linear PCP paradigm

[GGPR12]
[BCIOP13]

① Design a proof system sound against linear provers

② Force prover to be linear using a cryptographic encoding

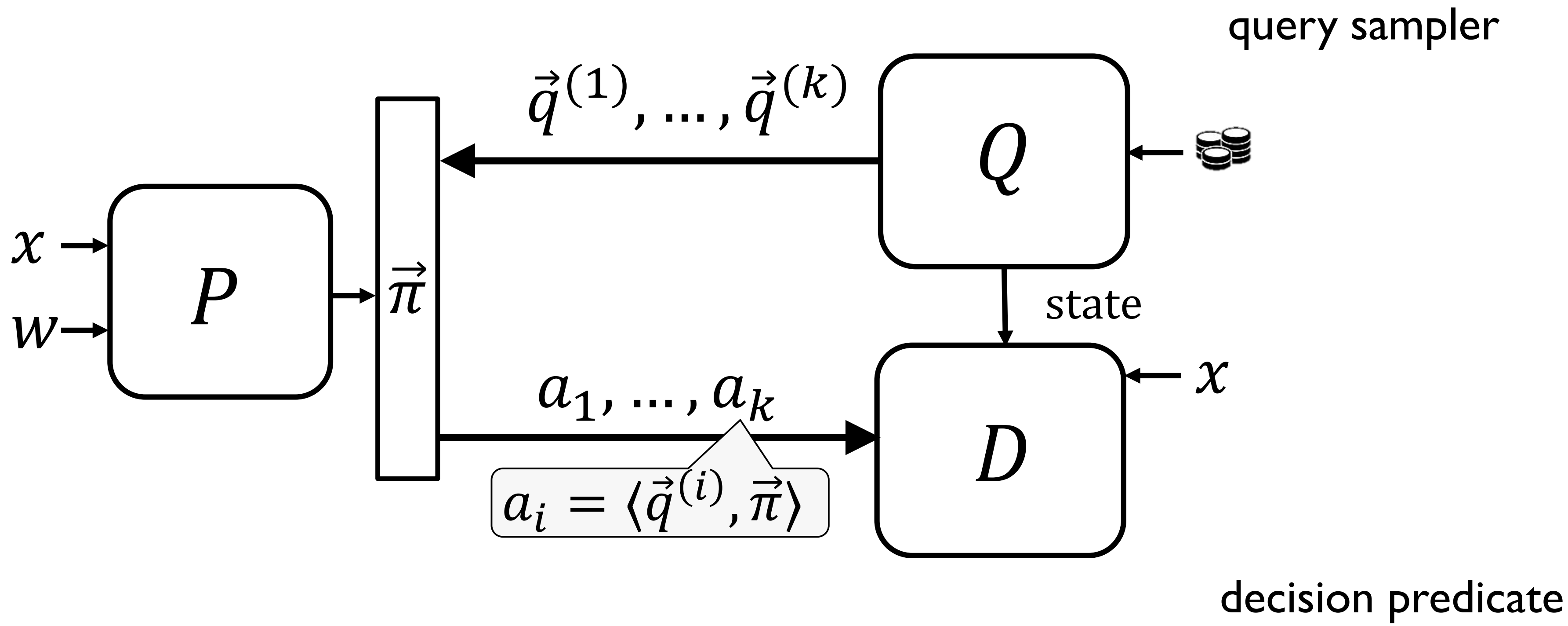


Linear PCP paradigm

[GGPR12]
[BCIOP13]

① Design a proof system sound against linear provers

② Force prover to be linear using a cryptographic encoding

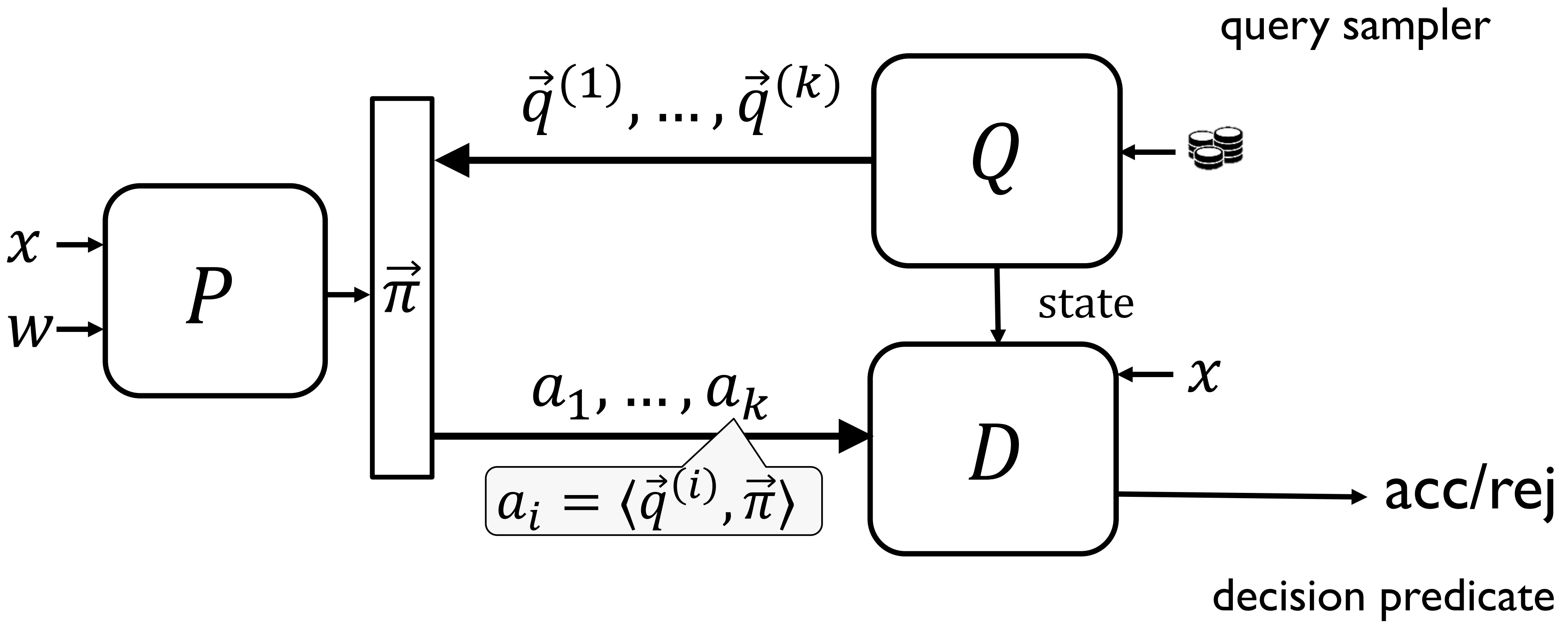


Linear PCP paradigm

[GGPR12]
[BCIOP13]

① Design a proof system sound against linear provers

② Force prover to be linear using a cryptographic encoding

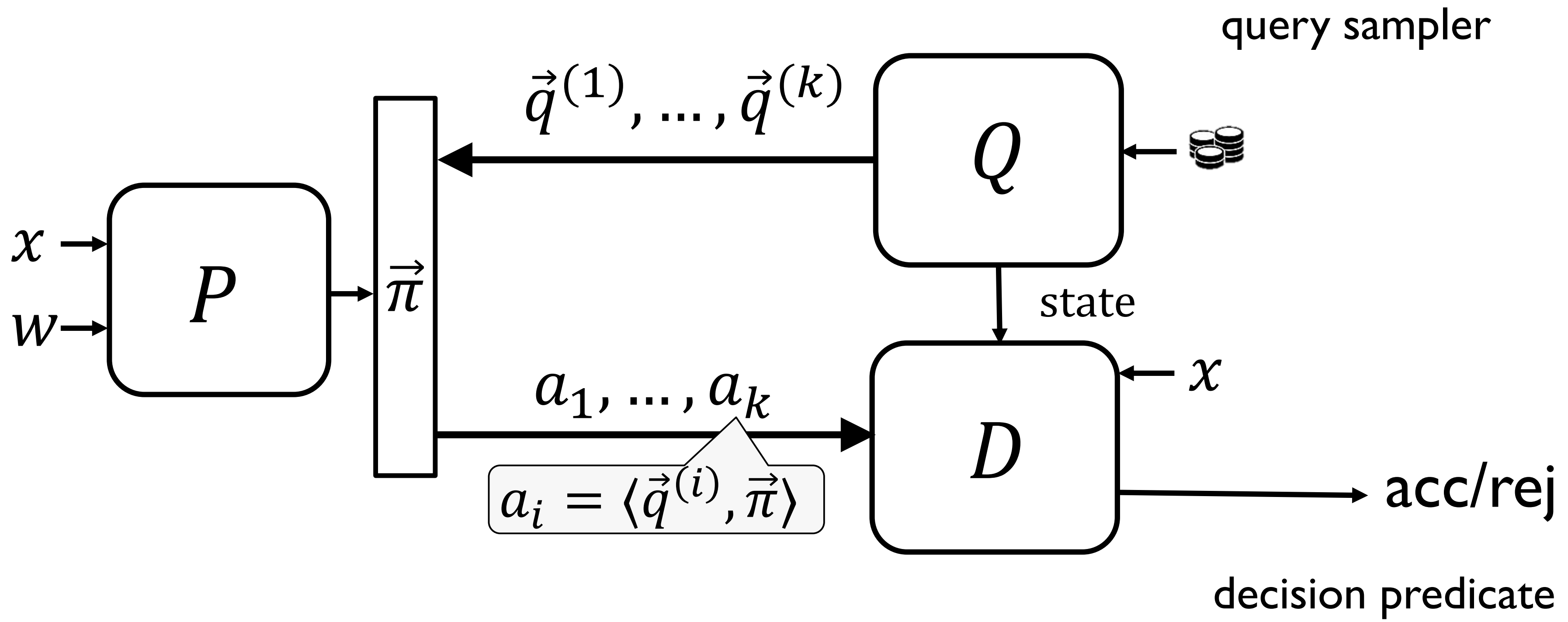


Linear PCP paradigm

[GGPR12]
[BCIOP13]

① Design a proof system sound against linear provers

② Force prover to be linear using a cryptographic encoding



Can define natural notions of completeness, PoK, ZK.

Our compilation technique

Our compilation technique

P V

Our compilation technique

Observe that P_{LPCP} does not need to know queries a priori.

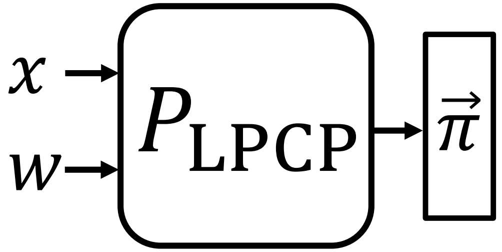
P V

Our compilation technique

Observe that P_{LPCP} does not need to know queries a priori.

P $\qquad\qquad\qquad\qquad$ V

① Run:



Our compilation technique

Observe that P_{LPCP} does not need to know queries a priori.

P $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ V

$$\xrightarrow{\mathbf{cm}_{\vec{\pi}} := \mathrm{Commit}(\vec{\pi})}$$

1 Run:

$x, w \rightarrow \boxed{P_{\mathrm{LPCP}}} \rightarrow \boxed{\vec{\pi}}$

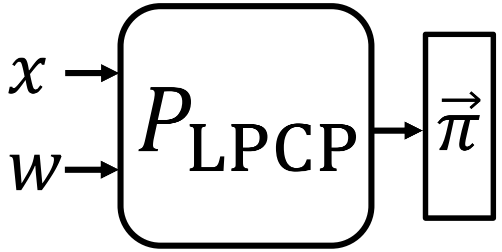
Our compilation technique

Observe that P_{LPCP} does not need to know queries a priori.

P $\qquad\qquad$ V

$\mathbf{cm}_{\vec{\pi}} := \mathrm{Commit}(\vec{\pi})$ $\longrightarrow$

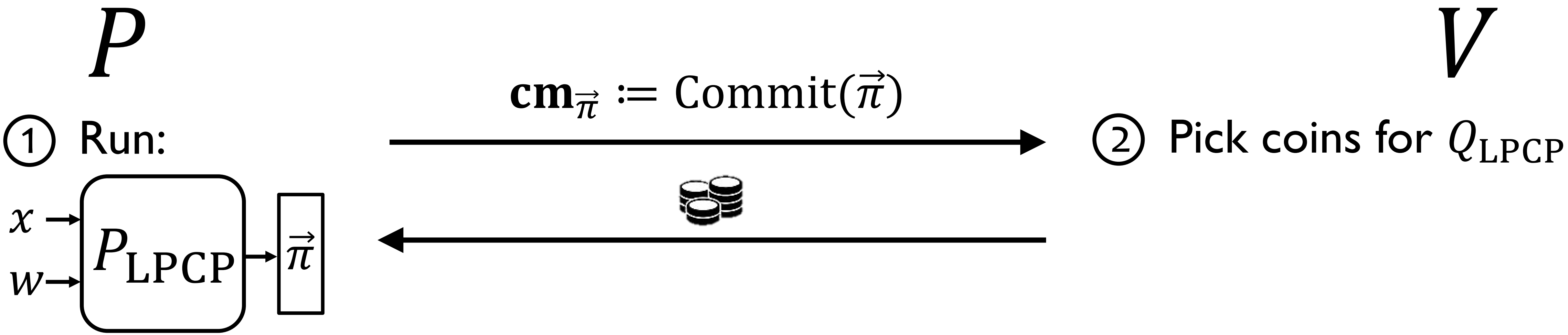
① Run:



② Pick coins for Q_{LPCP}

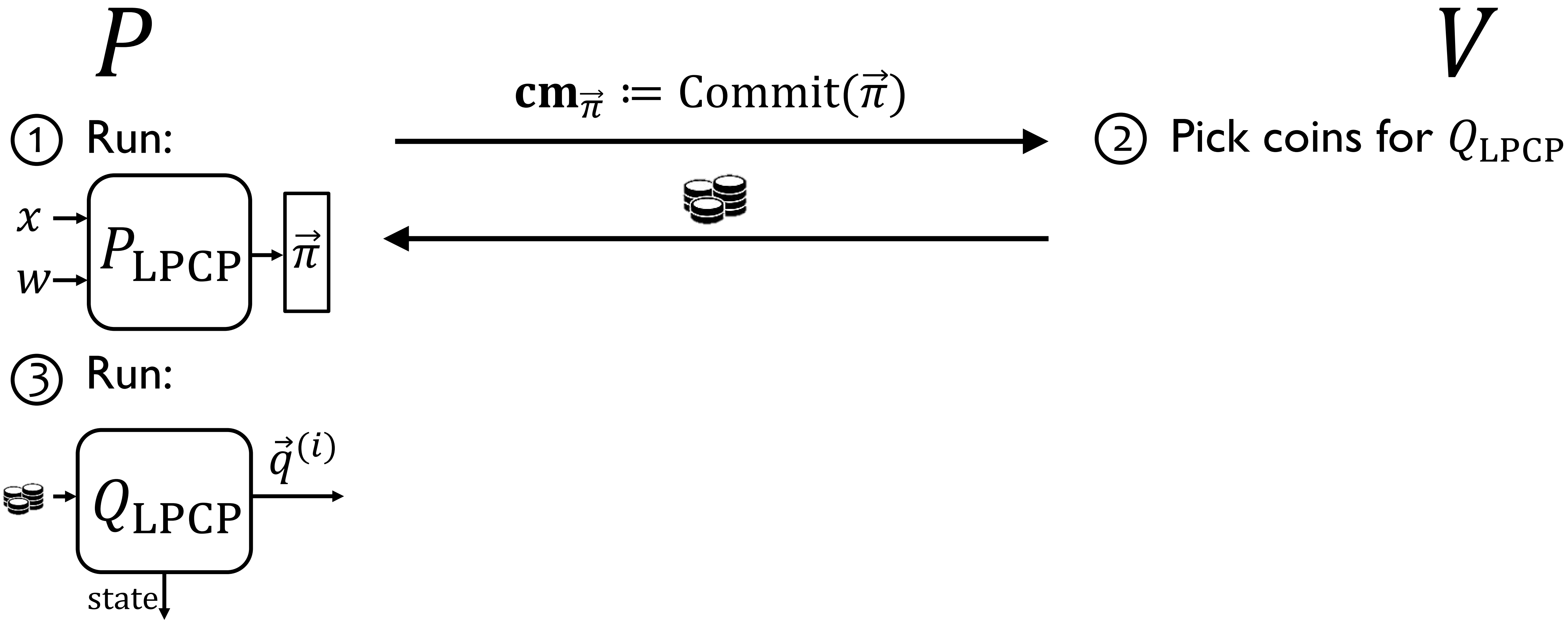
Our compilation technique

Observe that P_{LPCP} does not need to know queries a priori.



Our compilation technique

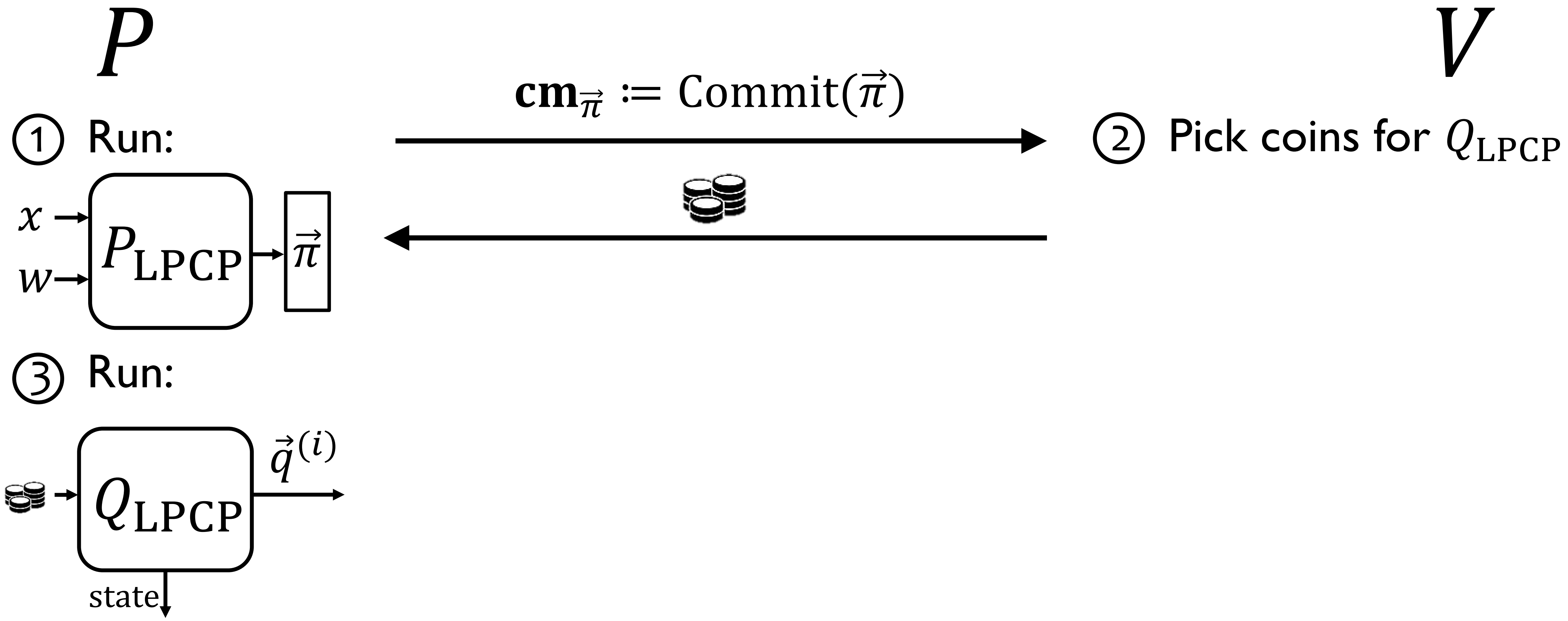
Observe that P_{LPCP} does not need to know queries a priori.



Our compilation technique

Observe that P_{LPCP} does not need to know queries a priori.

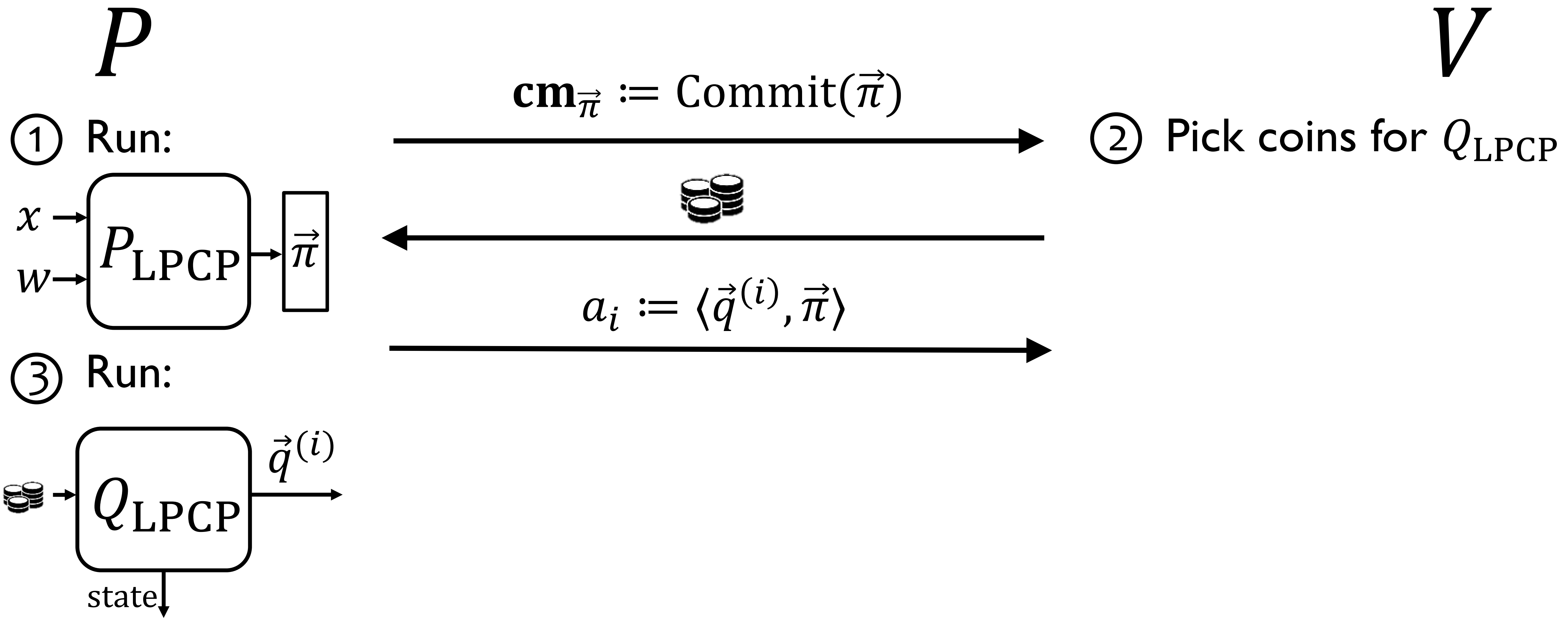
Public-coin so can't encrypt queries or use functional commitments. [BCIOP13] [LRY16]



Our compilation technique

Observe that P_{LPCP} does not need to know queries a priori.

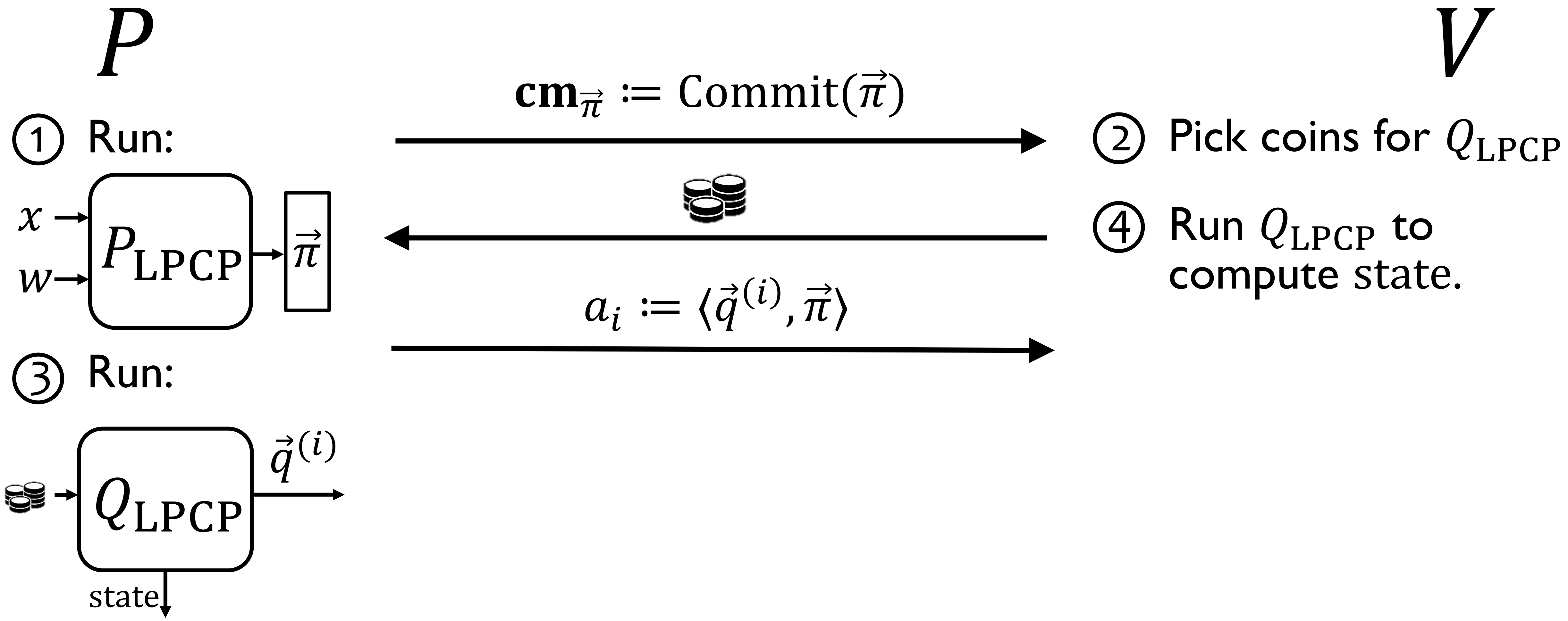
Public-coin so can't encrypt queries or use functional commitments. [BCIOP13] [LRY16]



Our compilation technique

Observe that P_{LPCP} does not need to know queries a priori.

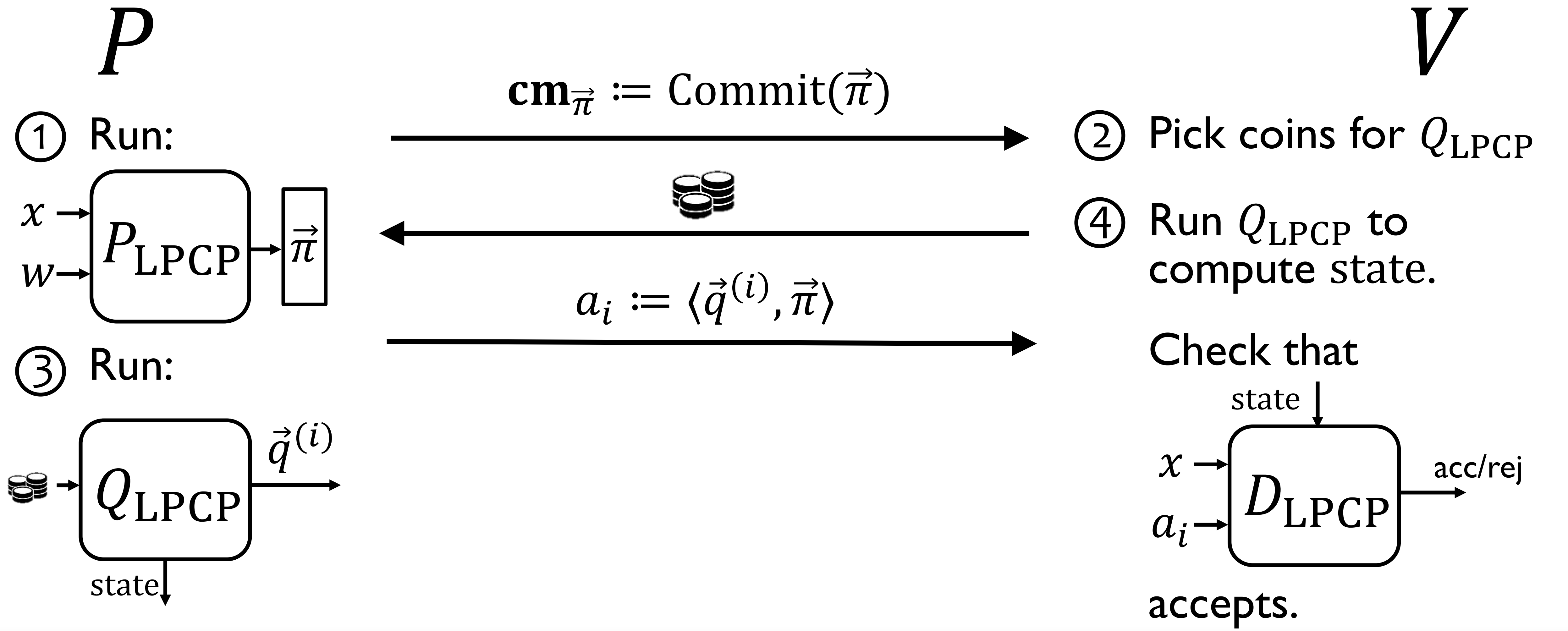
Public-coin so can't encrypt queries or use functional commitments. [BCIOP13] [LRY16]



Our compilation technique

Observe that P_{LPCP} does not need to know queries a priori.

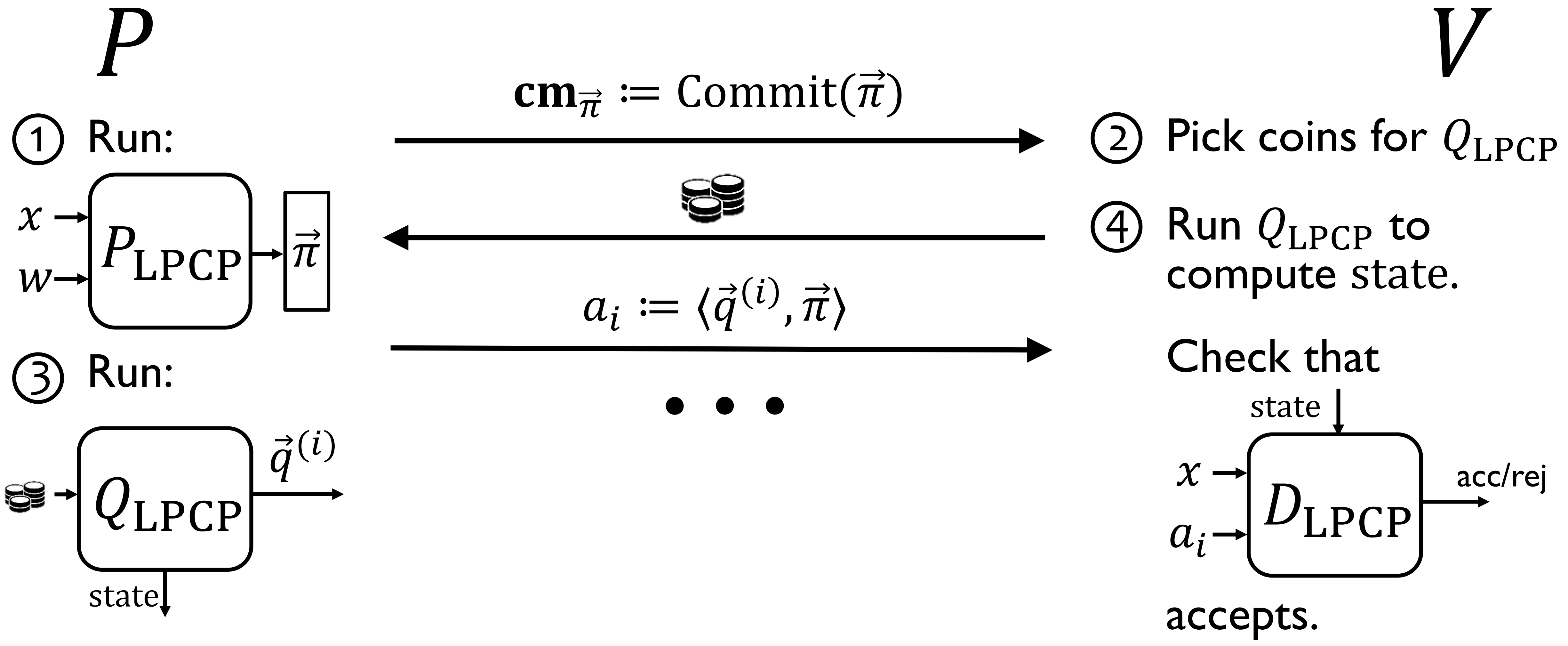
Public-coin so can't encrypt queries or use functional commitments. [BCIOP13] [LRY16]



Our compilation technique

Observe that P_{LPCP} does not need to know queries a priori.

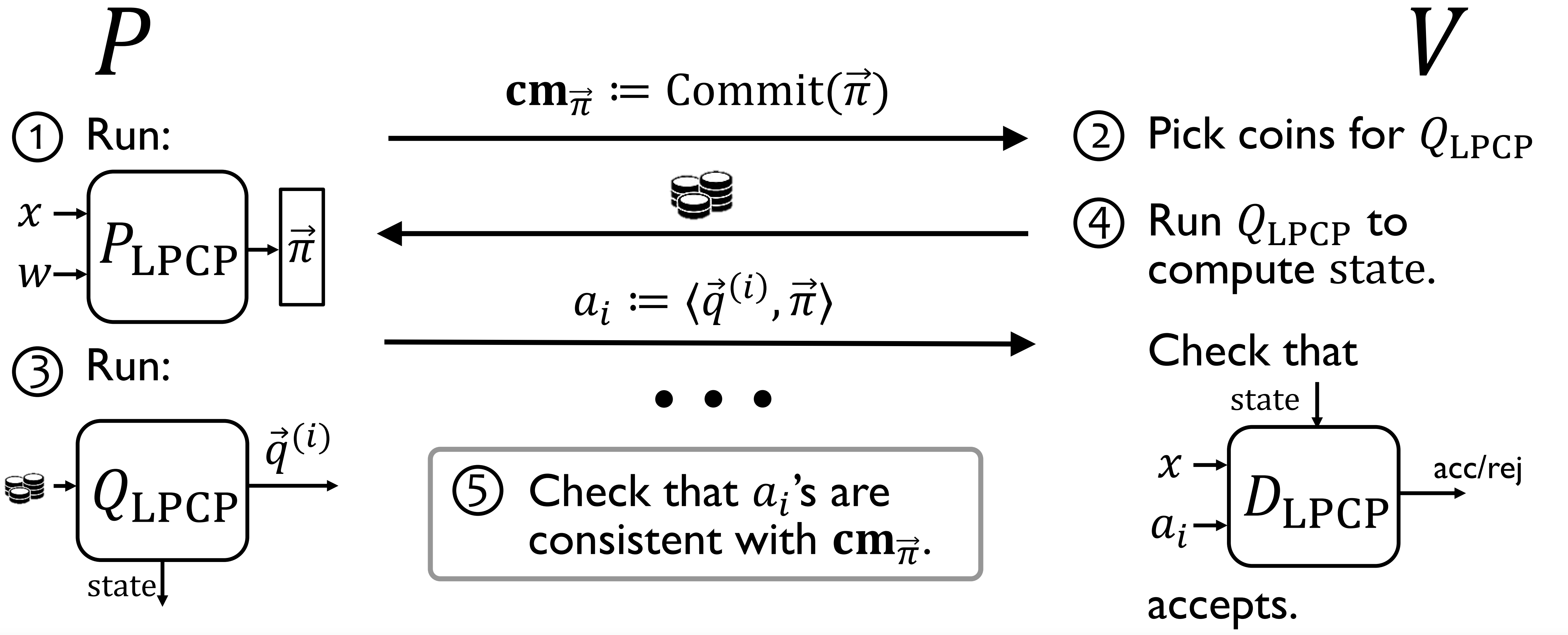
Public-coin so can't encrypt queries or use functional commitments. [BCIOP13] [LRY16]



Our compilation technique

Observe that P_{LPCP} does not need to know queries a priori.

Public-coin so can't encrypt queries or use functional commitments. [BCIOP13] [LRY16]



Answer consistency via inner product arguments

Answer consistency via inner product arguments

Verifier knows: , a's, and commitment to proof $\mathbf{cm}_{\vec{\pi}} := \mathrm{Commit}(\vec{\pi})$

Answer consistency via inner product arguments

Verifier knows: , a's, and commitment to proof $\mathbf{cm}_{\vec{\pi}} := \mathrm{Commit}(\vec{\pi})$

Goal: for every query $\vec{q}$ check $a = \langle \vec{q}, \vec{\pi} \rangle$ for a pre-committed $\vec{\pi}$

Answer consistency via inner product arguments

Verifier knows: , a's, and commitment to proof $\mathbf{cm}_{\vec{\pi}} := \mathrm{Commit}(\vec{\pi})$

Goal: for every query $\vec{q}$ check $a = \langle \vec{q}, \vec{\pi} \rangle$ for a pre-committed $\vec{\pi}$

Technique: **inner-product arguments** [BCCGP16,BBBPWM18]

Answer consistency via inner product arguments

Verifier knows: , a's, and commitment to proof $\mathbf{cm}_{\vec{\pi}} := \mathrm{Commit}(\vec{\pi})$

Goal: for every query $\vec{q}$ check $a = \langle \vec{q}, \vec{\pi} \rangle$ for a pre-committed $\vec{\pi}$

Technique: **inner-product arguments** [BCCGP16,BBBPWM18]

Input: Two vector Pedersen commitments $\mathbf{cm}_{\vec{u}}$, $\mathbf{cm}_{\vec{v}}$, and $z \in \mathbb{F}$.

Answer consistency via inner product arguments

Verifier knows: , a's, and commitment to proof $\mathbf{cm}_{\vec{\pi}} := \mathrm{Commit}(\vec{\pi})$

Goal: for every query $\vec{q}$ check $a = \langle \vec{q}, \vec{\pi} \rangle$ for a pre-committed $\vec{\pi}$

Technique: **inner-product arguments** [BCCGP16,BBBPWM18]

Input: Two vector Pedersen commitments $\mathbf{cm}_{\vec{u}}$, $\mathbf{cm}_{\vec{v}}$, and $z \in \mathbb{F}$.

Prove: The decommitments $\vec{u}, \vec{v} \in \mathbb{F}^n$ have the specified inner-product $z = \langle \vec{u}, \vec{v} \rangle$

Answer consistency via inner product arguments

Verifier knows: , a's, and commitment to proof $\mathbf{cm}_{\vec{\pi}} := \mathrm{Commit}(\vec{\pi})$

Goal: for every query $\vec{q}$ check $a = \langle \vec{q}, \vec{\pi} \rangle$ for a pre-committed $\vec{\pi}$

Technique: **inner-product arguments** [BCCGP16,BBBPWM18]

Input: Two vector Pedersen commitments $\mathbf{cm}_{\vec{u}}$, $\mathbf{cm}_{\vec{v}}$, and $z \in \mathbb{F}$.

Prove: The decommitments $\vec{u}, \vec{v} \in \mathbb{F}^n$ have the specified inner-product $z = \langle \vec{u}, \vec{v} \rangle$

So the verifier:

Answer consistency via inner product arguments

Verifier knows: , a's, and commitment to proof $\mathbf{cm}_{\vec{\pi}} := \mathrm{Commit}(\vec{\pi})$

Goal: for every query $\vec{q}$ check $a = \langle \vec{q}, \vec{\pi} \rangle$ for a pre-committed $\vec{\pi}$

Technique: **inner-product arguments** [BCCGP16,BBBPWM18]

Input: Two vector Pedersen commitments $\mathbf{cm}_{\vec{u}}$, $\mathbf{cm}_{\vec{v}}$, and $z \in \mathbb{F}$.

Prove: The decommitments $\vec{u}, \vec{v} \in \mathbb{F}^n$ have the specified inner-product $z = \langle \vec{u}, \vec{v} \rangle$

So the verifier:
- computes commitments to queries: $\mathbf{cm}_{\vec{q}} := \mathrm{Commit}(\vec{q})$

Answer consistency via inner product arguments

Verifier knows: , a's, and commitment to proof $\mathbf{cm}_{\vec{\pi}} := \mathrm{Commit}(\vec{\pi})$

Goal: for every query $\vec{q}$ check $a = \langle \vec{q}, \vec{\pi} \rangle$ for a pre-committed $\vec{\pi}$

Technique: **inner-product arguments** [BCCGP16,BBBPWM18]

Input: Two vector Pedersen commitments $\mathbf{cm}_{\vec{u}}$, $\mathbf{cm}_{\vec{v}}$, and $z \in \mathbb{F}$.

Prove: The decommitments $\vec{u}, \vec{v} \in \mathbb{F}^n$ have the specified inner-product $z = \langle \vec{u}, \vec{v} \rangle$

So the verifier:

- computes commitments to queries: $\mathbf{cm}_{\vec{q}} := \mathrm{Commit}(\vec{q})$
- engages in IP arguments for $(\mathbf{cm}_{\vec{q}}, \mathbf{cm}_{\vec{\pi}}, a)$

Answer consistency via inner product arguments

Verifier knows: , a's, and commitment to proof $\mathbf{cm}_{\vec{\pi}} := \mathrm{Commit}(\vec{\pi})$

Goal: for every query $\vec{q}$ check $a = \langle \vec{q}, \vec{\pi} \rangle$ for a pre-committed $\vec{\pi}$

Technique: **inner-product arguments** [BCCGP16,BBBPWM18]

Input: Two vector Pedersen commitments $\mathbf{cm}_{\vec{u}}$, $\mathbf{cm}_{\vec{v}}$, and $z \in \mathbb{F}$.

Prove: The decommitments $\vec{u}, \vec{v} \in \mathbb{F}^n$ have the specified inner-product $z = \langle \vec{u}, \vec{v} \rangle$

So the verifier:

- computes commitments to queries: $\mathbf{cm}_{\vec{q}} := \mathrm{Commit}(\vec{q})$
- engages in IP arguments for $(\mathbf{cm}_{\vec{q}}, \mathbf{cm}_{\vec{\pi}}, a)$

Result: NIZK from linear PCPs!

Answer consistency via inner product arguments

Verifier knows: , a's, and commitment to proof $\mathbf{cm}_{\vec{\pi}} := \mathrm{Commit}(\vec{\pi})$

Goal: for every query $\vec{q}$ check $a = \langle \vec{q}, \vec{\pi} \rangle$ for a pre-committed $\vec{\pi}$

Technique: **inner-product arguments** [BCCGP16,BBBPWM18]

Input: Two vector Pedersen commitments $\mathbf{cm}_{\vec{u}}$, $\mathbf{cm}_{\vec{v}}$, and $z \in \mathbb{F}$.

Prove: The decommitments $\vec{u}, \vec{v} \in \mathbb{F}^n$ have the specified inner-product $z = \langle \vec{u}, \vec{v} \rangle$

So the verifier:
- computes commitments to queries: $\mathbf{cm}_{\vec{q}} := \mathrm{Commit}(\vec{q})$
- engages in IP arguments for $(\mathbf{cm}_{\vec{q}}, \mathbf{cm}_{\vec{\pi}}, a)$

Result: NIZK from linear PCPs!

SHARK prudent proofs

The heart of our NIZK verifier

The heart of our NIZK verifier

V_{NIZK}

The heart of our NIZK verifier

V_{NIZK}

- check that LPCP decision predicate accepts (cheap)

The heart of our NIZK verifier

V_{NIZK}

- check that LPCP decision predicate accepts (cheap)
- compute Pedersen commitment to each LPCP query $\vec{q}$ (costly)

The heart of our NIZK verifier

V_{NIZK}

- check that LPCP decision predicate accepts (cheap)
- compute Pedersen commitment to each LPCP query $\vec{q}$ (costly)
- check that each inner product argument verifier accepts (costly)

The heart of our NIZK verifier

V_{NIZK}

- check that LPCP decision predicate accepts (cheap)
- compute Pedersen commitment to each LPCP query $\vec{q}$ (costly)
- check that each inner product argument verifier accepts (costly)

(Not pictured: Fiat-Shamir transform, …)

The heart of our NIZK verifier

V_{NIZK}

- check that LPCP decision predicate accepts (cheap)
- compute Pedersen commitment to each LPCP query $\vec{q}$ (costly)
- check that each inner product argument verifier accepts (costly)

we will make P_{SNARK} do both

(Not pictured: Fiat-Shamir transform, …)

The heart of our NIZK verifier

V_{NIZK}

- check that LPCP decision predicate accepts (cheap)
- compute Pedersen commitment to each LPCP query $\vec{q}$ (costly)
- check that each inner product argument verifier accepts (costly)

we will make P_{SNARK} do both

A new building block:
"encoded polynomial delegation"

(Not pictured: Fiat-Shamir transform, …)

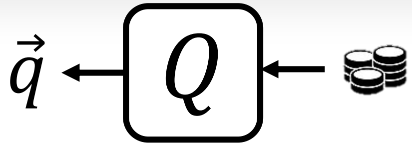
Outsourcing vector commitment computation

Outsourcing vector commitment computation

Goal: compute $\mathbf{cm}_{\vec{q}} := \mathrm{Commit}(\vec{q})$

Outsourcing vector commitment computation

Goal: compute $\mathbf{cm}_{\vec{q}} := \mathrm{Commit}(\vec{q})$



Outsourcing vector commitment computation

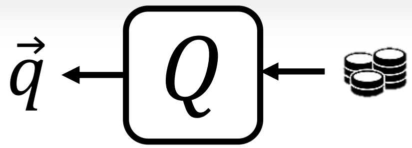
Goal: compute $\mathbf{cm}_{\vec{q}} := \mathrm{Commit}(\vec{q})$

$\vec{q} \leftarrow \boxed{Q} \leftarrow$

Most efficient linear PCP: quadratic arithmetic programs of [GGPR12]

Outsourcing vector commitment computation

Goal: compute $\mathbf{cm}_{\vec{q}} := \mathrm{Commit}(\vec{q})$



Most efficient linear PCP: quadratic arithmetic programs of [GGPR12]

Each query $\vec{q}$ has nice algebraic structure:

Outsourcing vector commitment computation

Goal: compute $\mathbf{cm}_{\vec{q}} := \mathrm{Commit}(\vec{q})$

$\vec{q} \leftarrow \boxed{Q} \leftarrow$ 

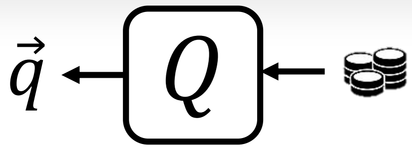
Most efficient linear PCP: quadratic arithmetic programs of [GGPR12]

Each query $\vec{q}$ has nice algebraic structure:

$\tau =$ 

Outsourcing vector commitment computation

Goal: compute $\mathbf{cm}_{\vec{q}} \coloneqq \mathrm{Commit}(\vec{q})$



Most efficient linear PCP: quadratic arithmetic programs of [GGPR12]

Each query $\vec{q}$ has nice algebraic structure:

$\tau =$ 

$$\vec{q} = (p_1(\tau), p_2(\tau), \dots, p_n(\tau))$$

Outsourcing vector commitment computation

Goal: compute $\mathbf{cm}_{\vec{q}} \coloneqq \mathrm{Commit}(\vec{q})$

$\vec{q} \leftarrow \boxed{Q} \leftarrow$ 

Most efficient linear PCP: quadratic arithmetic programs of [GGPR12]

Each query $\vec{q}$ has nice algebraic structure:

$\tau =$ 

$$\vec{q} = (p_1(\tau), p_2(\tau), \dots, p_n(\tau))$$

$$p_i(\tau) = p_{i,0} + p_{i,1}\tau + \cdots + p_{i,d}\tau^d$$

Outsourcing vector commitment computation

Goal: compute $\mathbf{cm}_{\vec{q}} := \mathrm{Commit}(\vec{q})$

$\vec{q} \leftarrow \boxed{Q} \leftarrow$ 

Most efficient linear PCP: quadratic arithmetic programs of [GGPR12]

Each query $\vec{q}$ has nice algebraic structure:

$\tau =$ 

$$\vec{q} = (p_1(\tau), p_2(\tau), \dots, p_n(\tau))$$

$$= (\ p_{1,0},\ \ p_{2,0},\ \dots,\ \ p_{n,0})$$

$$p_i(\tau) = p_{i,0} + p_{i,1}\tau + \cdots + p_{i,d}\tau^d$$

Outsourcing vector commitment computation

Goal: compute $\mathbf{cm}_{\vec{q}} := \mathrm{Commit}(\vec{q})$

$$\vec{q} \leftarrow \boxed{Q} \leftarrow$$

Most efficient linear PCP: quadratic arithmetic programs of [GGPR12]

Each query $\vec{q}$ has nice algebraic structure:

$$\tau =$$

$$\begin{aligned}
\vec{q} &= (p_1(\tau), p_2(\tau), \dots, p_n(\tau)) \\
&= (\ p_{1,0},\ p_{2,0}, \dots ,\ p_{n,0}) + \\
\tau \cdot &(\ p_{1,1},\ p_{2,1}, \dots ,\ p_{n,1})
\end{aligned}$$

$$p_i(\tau) = p_{i,0} + p_{i,1}\tau + \cdots + p_{i,d}\tau^d$$

Outsourcing vector commitment computation

Goal: compute $\mathbf{cm}_{\vec{q}} := \mathrm{Commit}(\vec{q})$

$$\vec{q} \leftarrow \boxed{Q} \leftarrow$$

Most efficient linear PCP: quadratic arithmetic programs of [GGPR12]

Each query $\vec{q}$ has nice algebraic structure:

$\tau =$

$$\vec{q} = (p_1(\tau), p_2(\tau), \dots, p_n(\tau))$$

$$= (\ p_{1,0},\ p_{2,0}, \dots ,\ p_{n,0}) \ +$$

$$\tau \cdot (\ p_{1,1},\ p_{2,1}, \dots ,\ p_{n,1}) \ +$$

$$\dots$$

$$\tau^d \cdot (\ p_{1,d},\ p_{2,d}, \dots ,\ p_{n,d})$$

$$p_i(\tau) = p_{i,0} + p_{i,1}\tau + \cdots + p_{i,d}\tau^d$$

Outsourcing vector commitment computation

Goal: compute $\mathbf{cm}_{\vec{q}} \coloneqq \mathrm{Commit}(\vec{q})$

$\vec{q} \leftarrow \boxed{Q} \leftarrow$ 

Most efficient linear PCP: quadratic arithmetic programs of [GGPR12]

Each query $\vec{q}$ has nice algebraic structure:

$\tau =$ 

$$
\begin{aligned}
\vec{q} &= (p_1(\tau), p_2(\tau), \dots, p_n(\tau)) \\
&= (\ p_{1,0},\ p_{2,0},\ \dots,\ p_{n,0}) \ + \\
\tau \cdot &(\ p_{1,1},\ p_{2,1},\ \dots,\ p_{n,1}) \ + \\
&\qquad \dots \\
\tau^d \cdot &(\ p_{1,d},\ p_{2,d},\ \dots,\ p_{n,d})
\end{aligned}
$$

$$\mathbf{cm}_{\vec{q}} = \mathrm{Commit}(p_1(\tau), p_2(\tau), \dots, p_n(\tau))$$

$$p_i(\tau) = p_{i,0} + p_{i,1}\tau + \cdots + p_{i,d}\tau^d$$

Outsourcing vector commitment computation

Goal: compute $\mathbf{cm}_{\vec{q}} := \mathrm{Commit}(\vec{q})$

$\vec{q} \leftarrow \boxed{Q} \leftarrow$ 

Most efficient linear PCP: quadratic arithmetic programs of [GGPR12]

Each query $\vec{q}$ has nice algebraic structure:

$\tau =$ 

$$\begin{aligned}\vec{q} &= (p_1(\tau), p_2(\tau), \ldots, p_n(\tau)) \\ &= (p_{1,0}, p_{2,0}, \ldots, p_{n,0}) + \\ &\tau \cdot (p_{1,1}, p_{2,1}, \ldots, p_{n,1}) + \\ &\quad \ldots \\ &\tau^d \cdot (p_{1,d}, p_{2,d}, \ldots, p_{n,d})\end{aligned}$$

$$\begin{aligned}\mathbf{cm}_{\vec{q}} &= \mathrm{Commit}(p_1(\tau), p_2(\tau), \ldots, p_n(\tau)) \\ &= \mathrm{Commit}(p_{1,0}, p_{2,0}, \ldots, p_{n,0})\end{aligned}$$

$$p_i(\tau) = p_{i,0} + p_{i,1}\tau + \cdots + p_{i,d}\tau^d$$

Outsourcing vector commitment computation

Goal: compute $\mathbf{cm}_{\vec{q}} \coloneqq \mathrm{Commit}(\vec{q})$

$$\vec{q} \leftarrow \boxed{Q} \leftarrow$$

Most efficient linear PCP: quadratic arithmetic programs of [GGPR12]

Each query $\vec{q}$ has nice algebraic structure:

$\tau =$

$$\begin{aligned}
\vec{q} &= (p_1(\tau), p_2(\tau), \dots, p_n(\tau)) \\
&= (\ p_{1,0},\ p_{2,0},\ \dots,\ p_{n,0}) \ + \\
&\quad \tau \cdot (\ p_{1,1},\ p_{2,1},\ \dots,\ p_{n,1}) \ + \\
&\quad \dots \\
&\quad \tau^d \cdot (\ p_{1,d},\ p_{2,d},\ \dots,\ p_{n,d})
\end{aligned}$$

$$\begin{aligned}
\mathbf{cm}_{\vec{q}} &= \mathrm{Commit}(p_1(\tau), p_2(\tau), \dots, p_n(\tau)) \\
&= \mathrm{Commit}(p_{1,0}, p_{2,0}, \dots, p_{n,0}) + \\
&\quad \tau \cdot \mathrm{Commit}(p_{1,1}, p_{2,1}, \dots, p_{n,1})
\end{aligned}$$

$$p_i(\tau) = p_{i,0} + p_{i,1}\tau + \cdots + p_{i,d}\tau^d$$

Outsourcing vector commitment computation

Goal: compute $\mathbf{cm}_{\vec{q}} := \mathrm{Commit}(\vec{q})$

$\vec{q} \leftarrow \boxed{Q} \leftarrow$ 

Most efficient linear PCP: quadratic arithmetic programs of [GGPR12]

Each query $\vec{q}$ has nice algebraic structure:

$\tau =$ 

$$\begin{aligned}\vec{q} &= (p_1(\tau), p_2(\tau), \dots, p_n(\tau)) \\ &= (\ p_{1,0},\ p_{2,0},\ \dots,\ p_{n,0}) \ + \\ &\quad \tau \cdot (\ p_{1,1},\ p_{2,1},\ \dots,\ p_{n,1}) \ + \\ &\quad \dots \\ &\quad \tau^d \cdot (\ p_{1,d},\ p_{2,d},\ \dots,\ p_{n,d})\end{aligned}$$

$$\begin{aligned}\mathbf{cm}_{\vec{q}} &= \mathrm{Commit}(p_1(\tau), p_2(\tau), \dots, p_n(\tau)) \\ &= \mathrm{Commit}(p_{1,0}, p_{2,0}, \dots, p_{n,0}) + \\ &\quad \tau \cdot \mathrm{Commit}(p_{1,1}, p_{2,1}, \dots, p_{n,1}) + \\ &\quad \dots \\ &\quad \tau^d \cdot \mathrm{Commit}(p_{1,d}, p_{2,d}, \dots, p_{n,d})\end{aligned}$$

$$p_i(\tau) = p_{i,0} + p_{i,1}\tau + \cdots + p_{i,d}\tau^d$$

Encoded polynomial delegation

Encoded polynomial delegation

Goal: outsource computation of

$$
\begin{aligned}
\mathbf{cm}_{\vec{q}} &= \mathrm{Com}(p_1(\tau), p_2(\tau), \dots, p_n(\tau)) \\
&= \mathrm{Com}(p_{1,0}, p_{2,0}, \dots, p_{n,0}) + \\
&\quad \tau \cdot \mathrm{Com}(p_{1,1}, p_{2,1}, \dots, p_{n,1}) + \\
&\quad \dots \\
&\quad \tau^d \cdot \mathrm{Com}(p_{1,d}, p_{2,d}, \dots, p_{n,d})
\end{aligned}
$$

Encoded polynomial delegation

Goal: outsource computation of

$$\mathbf{cm}_{\vec{q}} = \mathrm{Com}(p_1(\tau), p_2(\tau), \dots, p_n(\tau))$$

$$= \mathrm{Com}(p_{1,0}, p_{2,0}, \dots, p_{n,0}) +$$

$$\tau \cdot \mathrm{Com}(p_{1,1}, p_{2,1}, \dots, p_{n,1}) +$$

$$\dots$$

$$\tau^d \cdot \mathrm{Com}(p_{1,d}, p_{2,d}, \dots, p_{n,d})$$

Com( )'s are fully determined by L!

Encoded polynomial delegation

Goal: outsource computation of

$$\mathbf{cm}_{\vec{q}} = \mathrm{Com}(p_1(\tau), p_2(\tau), \dots, p_n(\tau))$$

$$= \mathrm{Com}(p_{1,0}, p_{2,0}, \dots, p_{n,0}) +$$

$$\tau \cdot \mathrm{Com}(p_{1,1}, p_{2,1}, \dots, p_{n,1}) +$$

$$\dots$$

$$\tau^d \cdot \mathrm{Com}(p_{1,d}, p_{2,d}, \dots, p_{n,d})$$

$\Rightarrow$

Fixed parameters $U_0, U_1, \dots, U_d \in \mathbb{G}$

$\mathrm{Com}(\quad)$'s are fully determined by L!

Encoded polynomial delegation

Goal: outsource computation of

$$\mathbf{cm}_{\vec{q}} = \mathrm{Com}(p_1(\tau), p_2(\tau), \ldots, p_n(\tau))$$
$$= \mathrm{Com}(p_{1,0}, p_{2,0}, \ldots, p_{n,0}) +$$
$$\tau \cdot \mathrm{Com}(p_{1,1}, p_{2,1}, \ldots, p_{n,1}) +$$
$$\ldots$$
$$\tau^d \cdot \mathrm{Com}(p_{1,d}, p_{2,d}, \ldots, p_{n,d})$$

$\mathrm{Com}(\quad)$'s are fully determined by L!

$\Rightarrow$

Fixed parameters $U_0, U_1, \ldots, U_d \in \mathbb{G}$

For outsourcing $\mathbf{cm}_{\vec{q}}$ set

$$U_k = \mathrm{Com}(p_{1,k}, p_{2,k}, \ldots, p_{n,k})$$

Encoded polynomial delegation

Goal: outsource computation of

$$\mathbf{cm}_{\vec{q}} = \mathrm{Com}(p_1(\tau), p_2(\tau), \ldots, p_n(\tau))$$
$$= \mathrm{Com}(p_{1,0}, p_{2,0}, \ldots, p_{n,0}) +$$
$$\tau \cdot \mathrm{Com}(p_{1,1}, p_{2,1}, \ldots, p_{n,1}) +$$
$$\ldots$$
$$\tau^d \cdot \mathrm{Com}(p_{1,d}, p_{2,d}, \ldots, p_{n,d})$$

Com( )'s are fully determined by L!

$\Rightarrow$

Fixed parameters $U_0, U_1, \ldots, U_d \in \mathbb{G}$

Goal: given input $\tau \in \mathbb{F}$,

For outsourcing $\mathbf{cm}_{\vec{q}}$ set

$$U_k = \mathrm{Com}(p_{1,k}, p_{2,k}, \ldots, p_{n,k})$$

Encoded polynomial delegation

Goal: outsource computation of

$$\mathbf{cm}_{\vec{q}} = \mathrm{Com}(p_1(\tau), p_2(\tau), \dots, p_n(\tau))$$
$$= \mathrm{Com}(p_{1,0}, p_{2,0}, \dots, p_{n,0}) +$$
$$\tau \cdot \mathrm{Com}(p_{1,1}, p_{2,1}, \dots, p_{n,1}) +$$
$$\dots$$
$$\tau^d \cdot \mathrm{Com}(p_{1,d}, p_{2,d}, \dots, p_{n,d})$$

$\mathrm{Com}(\quad)$'s are fully determined by L!

$\Rightarrow$

Fixed parameters $U_0, U_1, \dots, U_d \in \mathbb{G}$

Goal: given input $\tau \in \mathbb{F}$,
outsource this computation:

$$U := U_0 + \tau \cdot U_1 + \cdots + \tau^d \cdot U_d \, .$$

For outsourcing $\mathbf{cm}_{\vec{q}}$ set

$$U_k = \mathrm{Com}(p_{1,k}, p_{2,k}, \dots, p_{n,k})$$

Encoded polynomial delegation

Goal: outsource computation of

$$\mathbf{cm}_{\vec{q}} = \mathrm{Com}(p_1(\tau), p_2(\tau), \dots, p_n(\tau))$$
$$= \mathrm{Com}(p_{1,0}, p_{2,0}, \dots, p_{n,0}) +$$
$$\tau \cdot \mathrm{Com}(p_{1,1}, p_{2,1}, \dots, p_{n,1}) +$$
$$\dots$$
$$\tau^d \cdot \mathrm{Com}(p_{1,d}, p_{2,d}, \dots, p_{n,d})$$

$\mathrm{Com}(\quad)$'s are fully determined by L!

$\Rightarrow$

Encoded polynomial delegation

Fixed parameters $U_0, U_1, \dots, U_d \in \mathbb{G}$

Goal: given input $\tau \in \mathbb{F}$,
outsource this computation:

$$U := U_0 + \tau \cdot U_1 + \cdots + \tau^d \cdot U_d \,.$$

For outsourcing $\mathbf{cm}_{\vec{q}}$ set

$$U_k = \mathrm{Com}(p_{1,k}, p_{2,k}, \dots, p_{n,k})$$

Encoded polynomial delegation

Goal: outsource computation of

$$\mathbf{cm}_{\vec{q}} = \mathrm{Com}(p_1(\tau), p_2(\tau), \dots, p_n(\tau))$$
$$= \mathrm{Com}(p_{1,0}, p_{2,0}, \dots, p_{n,0}) +$$
$$\tau \cdot \mathrm{Com}(p_{1,1}, p_{2,1}, \dots, p_{n,1}) +$$
$$\dots$$
$$\tau^d \cdot \mathrm{Com}(p_{1,d}, p_{2,d}, \dots, p_{n,d})$$

Com( )'s are fully determined by L!

$\Rightarrow$

Encoded polynomial delegation

Fixed parameters $U_0, U_1, \dots, U_d \in \mathbb{G}$

Goal: given input $\tau \in \mathbb{F}$,
outsource this computation:

$$U := U_0 + \tau \cdot U_1 + \cdots + \tau^d \cdot U_d \,.$$

For outsourcing $\mathbf{cm}_{\vec{q}}$ set
$U_k = \mathrm{Com}(p_{1,k}, p_{2,k}, \dots, p_{n,k})$

New building block: SNARK for encoded polynomial delegation in pairing groups

Encoded polynomial delegation

Goal: outsource computation of

$$\mathbf{cm}_{\vec{q}} = \mathrm{Com}(p_1(\tau), p_2(\tau), \dots, p_n(\tau))$$
$$= \mathrm{Com}(p_{1,0}, p_{2,0}, \dots, p_{n,0}) +$$
$$\tau \cdot \mathrm{Com}(p_{1,1}, p_{2,1}, \dots, p_{n,1}) +$$
$$\dots$$
$$\tau^d \cdot \mathrm{Com}(p_{1,d}, p_{2,d}, \dots, p_{n,d})$$

Com( )'s are fully determined by L!

$\Rightarrow$

Encoded polynomial delegation

Fixed parameters $U_0, U_1, \dots, U_d \in \mathbb{G}$

Goal: given input $\tau \in \mathbb{F}$, outsource this computation:

$$U := U_0 + \tau \cdot U_1 + \cdots + \tau^d \cdot U_d \, .$$

For outsourcing $\mathbf{cm}_{\vec{q}}$ set

$$U_k = \mathrm{Com}(p_{1,k}, p_{2,k}, \dots, p_{n,k})$$

New building block: SNARK for encoded polynomial delegation in pairing groups + multilinear variant for our optimized IP argument.

Encoded polynomial delegation

☑ SHARK optimistic proofs

Goal: outsource computation of

$$\mathbf{cm}_{\vec{q}} = \mathrm{Com}(p_1(\tau), p_2(\tau), \dots, p_n(\tau))$$
$$= \mathrm{Com}(p_{1,0}, p_{2,0}, \dots, p_{n,0}) +$$
$$\tau \cdot \mathrm{Com}(p_{1,1}, p_{2,1}, \dots, p_{n,1}) +$$
$$\dots$$
$$\tau^d \cdot \mathrm{Com}(p_{1,d}, p_{2,d}, \dots, p_{n,d})$$

Com( )'s are fully determined by L!

$\Rightarrow$

Encoded polynomial delegation

Fixed parameters $U_0, U_1, \dots, U_d \in \mathbb{G}$

Goal: given input $\tau \in \mathbb{F}$, outsource this computation:

$$U := U_0 + \tau \cdot U_1 + \cdots + \tau^d \cdot U_d\ .$$

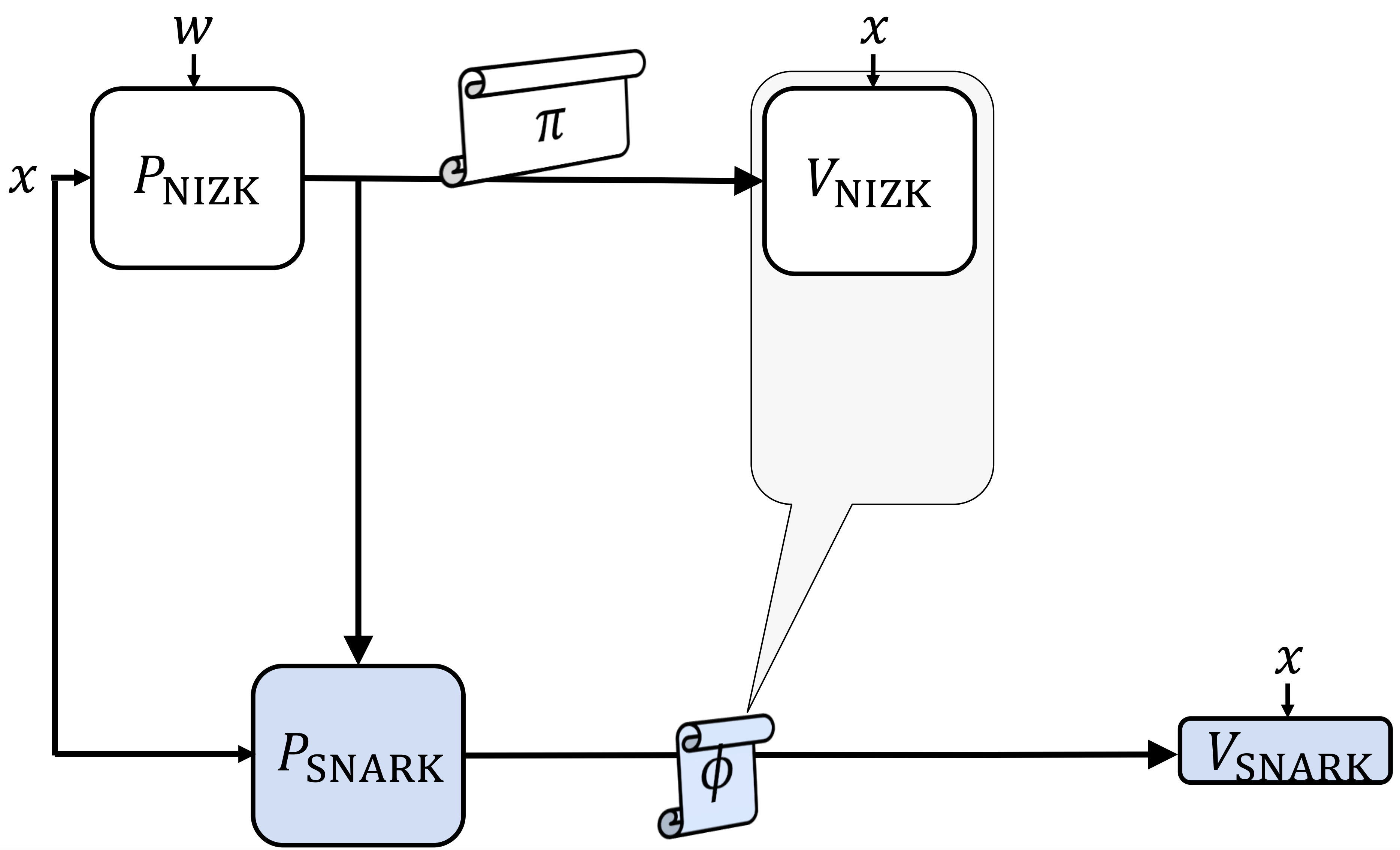
For outsourcing $\mathbf{cm}_{\vec{q}}$ set

$$U_k = \mathrm{Com}(p_{1,k}, p_{2,k}, \dots, p_{n,k})$$

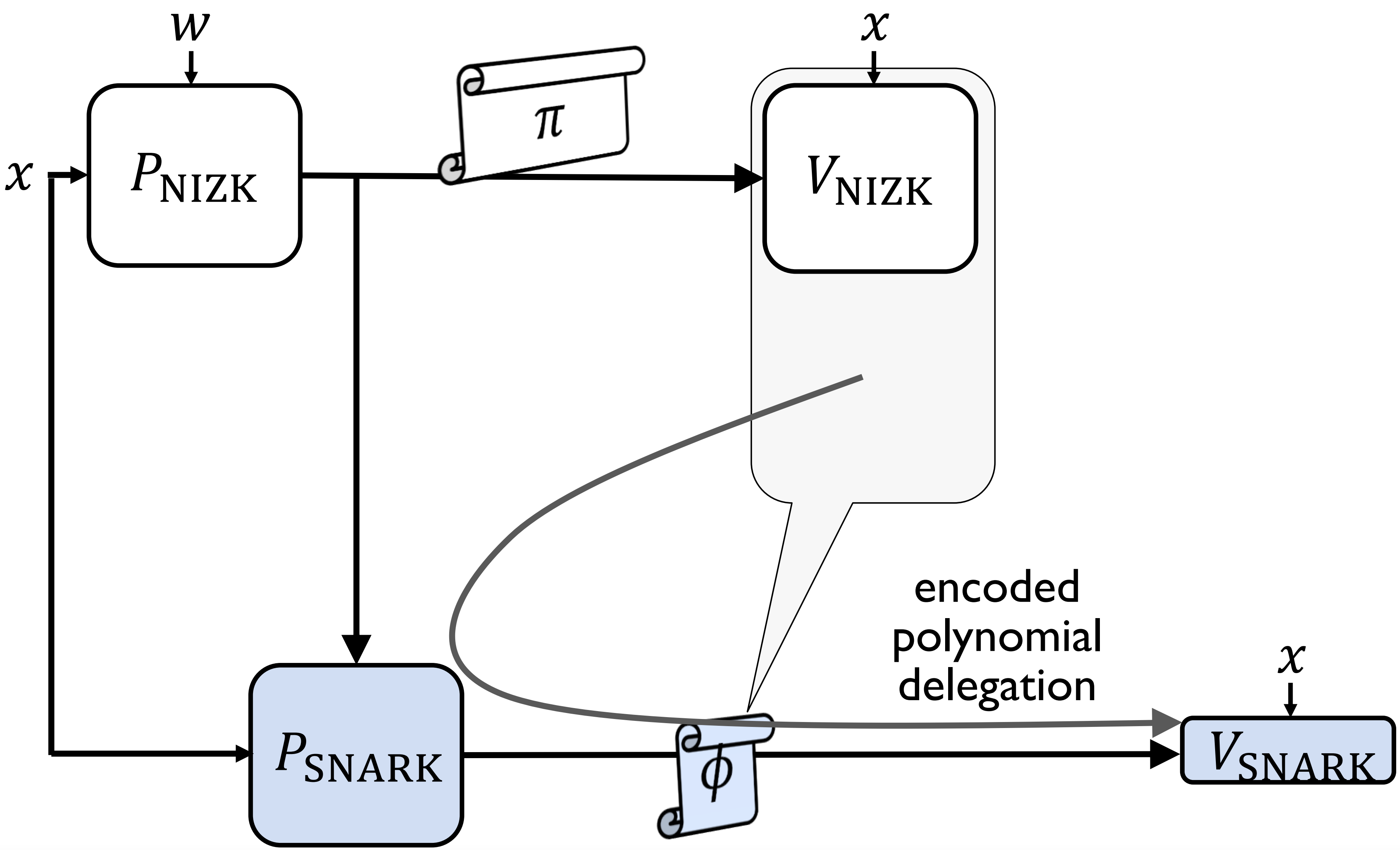
New building block: SNARK for encoded polynomial delegation in pairing groups + multilinear variant for our optimized IP argument.

Putting things together

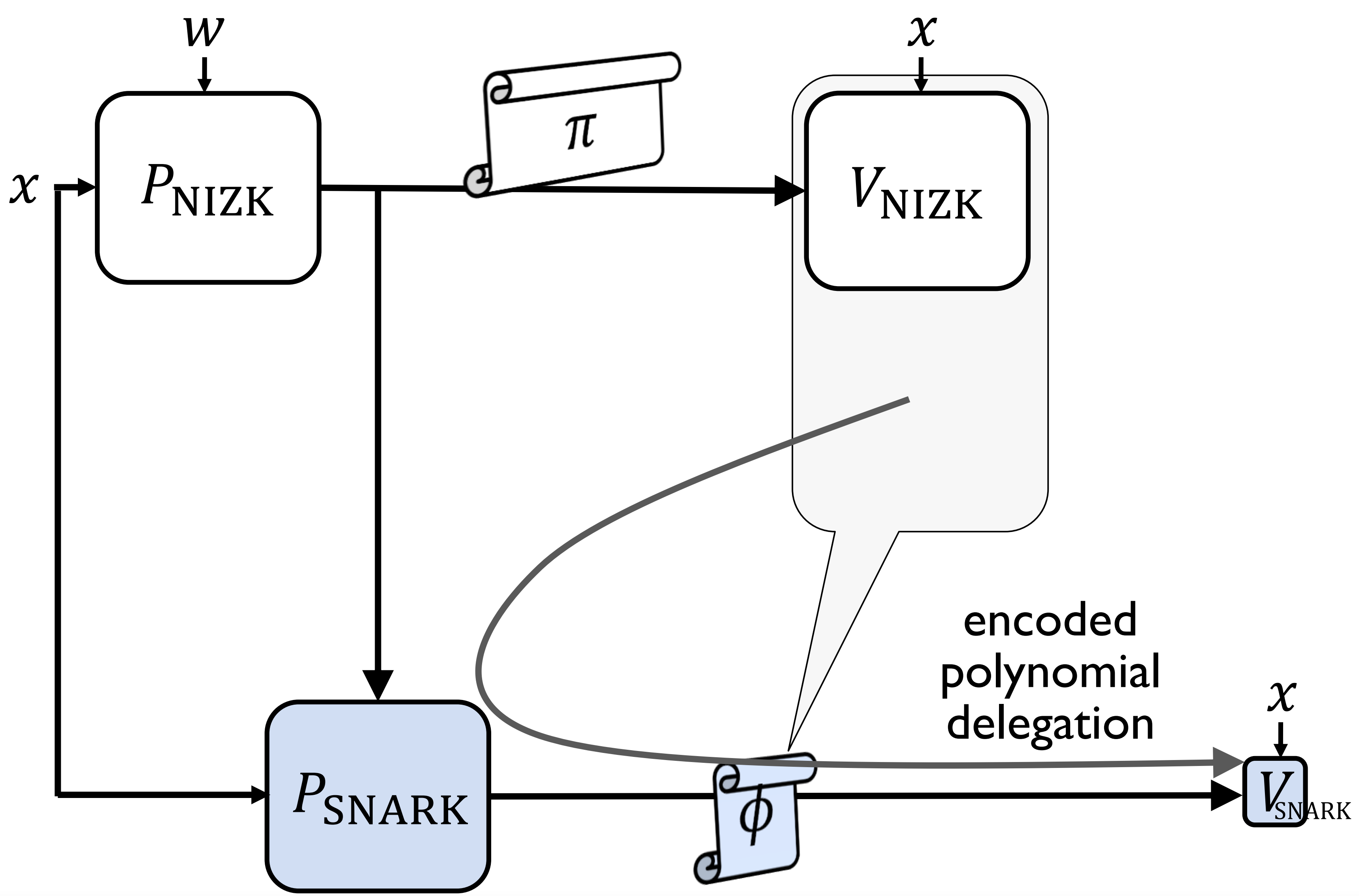
Putting things together



Putting things together



Putting things together



Comparison with generic instantiation

Comparison with generic instantiation

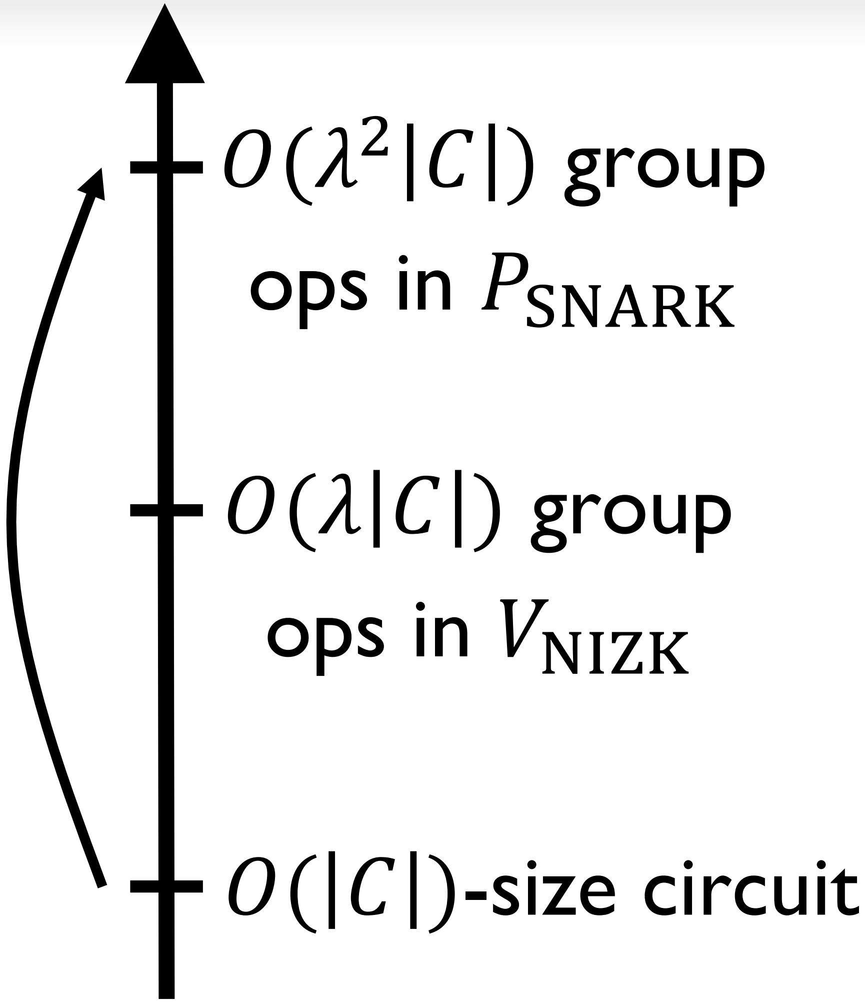
$O(\lambda^2|C|)$ group ops in P_{SNARK}

$O(\lambda|C|)$ group ops in V_{NIZK}

$O(|C|)$-size circuit

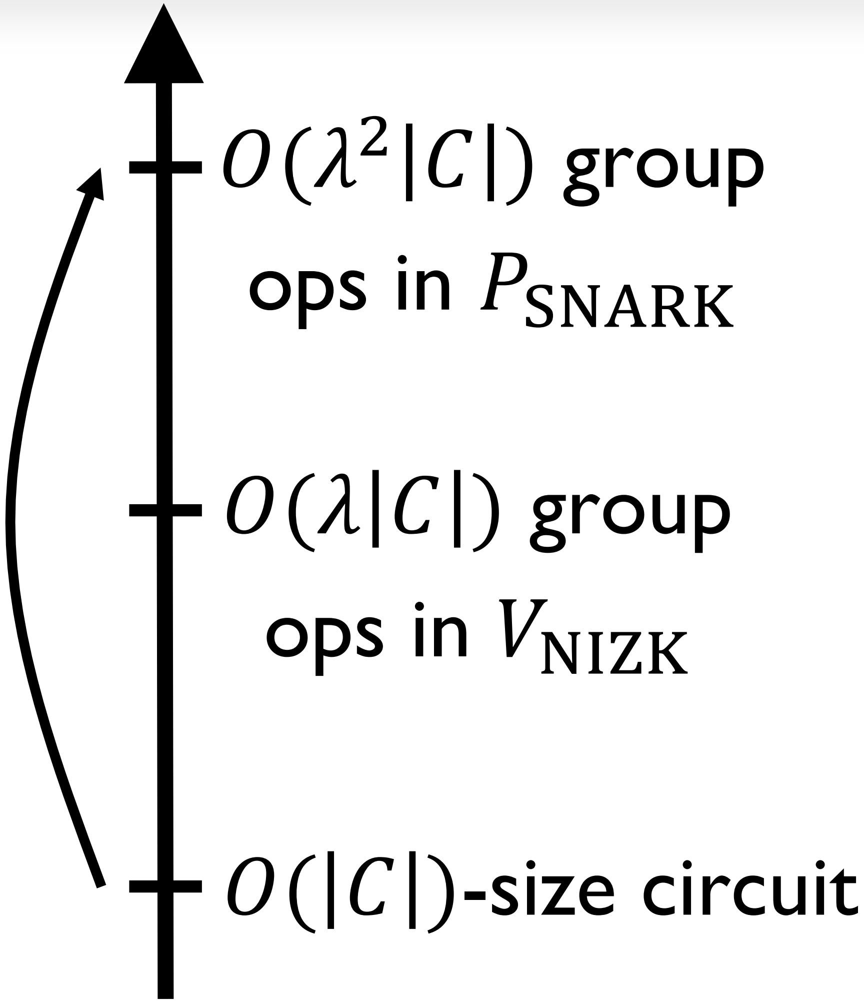
Generic instantiation

Comparison with generic instantiation



Generic instantiation

Comparison with generic instantiation



Generic instantiation

This work

Comparison with generic instantiation

$O(\lambda^2|C|)$ group ops in P_{SNARK}

$O(\lambda|C|)$ group ops in V_{NIZK}

$O(|C|)$-size circuit

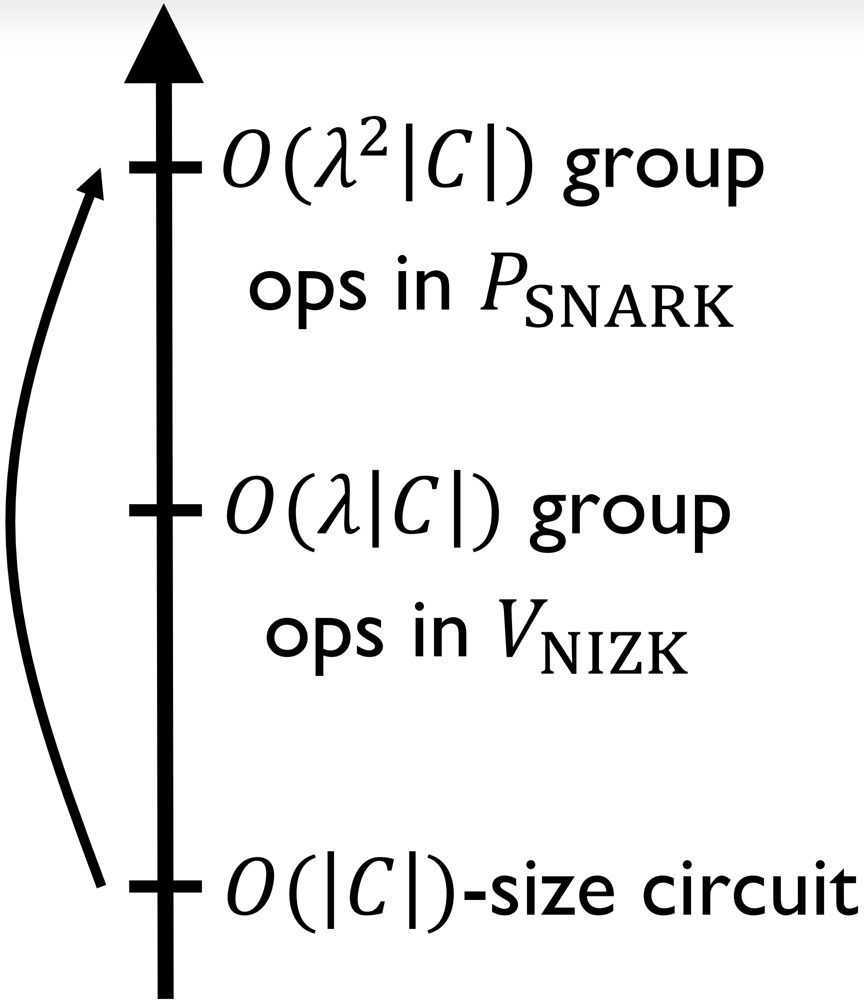
Generic instantiation

$O(\lambda|C|)$ group ops in V_{NIZK}

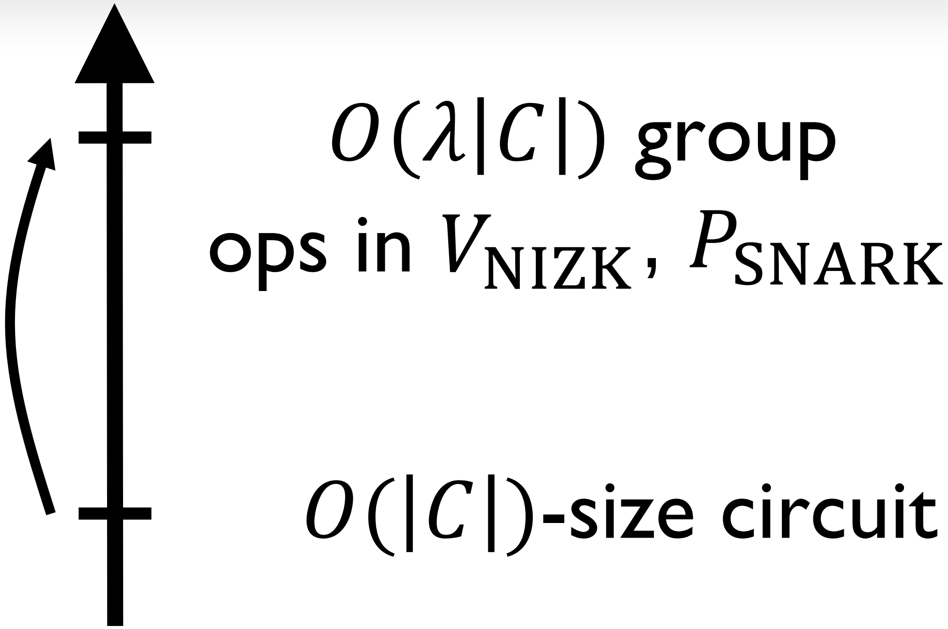
$O(|C|)$-size circuit

This work

Comparison with generic instantiation



Generic instantiation



This work

Comparison with generic instantiation

$O(\lambda^2|C|)$ group ops in P_{SNARK}

$O(\lambda|C|)$ group ops in V_{NIZK}

$O(|C|)$-size circuit

Generic instantiation

$O(\lambda|C|)$ group ops in V_{NIZK}, P_{SNARK}

$O(|C|)$-size circuit

$+$ competitive proof size and verification time

This work

Comparison with generic instantiation

$O(\lambda^2|C|)$ group ops in P_{SNARK}

$O(\lambda|C|)$ group ops in V_{NIZK}

$O(|C|)$-size circuit

Generic instantiation

$O(\lambda|C|)$ group ops in V_{NIZK}, P_{SNARK}

$O(|C|)$-size circuit

$+$ competitive proof size and verification time

e.g. 5 batchable pairings and
$6 \times \mathbb{G}_1$ for an optimistic proof

This work

Swimming with SHARKs

Swimming with SHARKs

- New primitive: private-coin setup needed for performance but not for soundness or ZK

Swimming with SHARKs

- New primitive: private-coin setup needed for performance but not for soundness or ZK

- Compromised setup can be quickly detected and easily replaced

Swimming with SHARKs

- New primitive: private-coin setup needed for performance but not for soundness or ZK

- Compromised setup can be quickly detected and easily replaced

- Speed competitive with best current zk-SNARKs

Swimming with SHARKs

- New primitive: private-coin setup needed for performance but not for soundness or ZK

- Compromised setup can be quickly detected and easily replaced

- Speed competitive with best current zk-SNARKs

- New building blocks along the way:
    - Optimized inner product argument
    - SNARK for encoded polynomial delegation