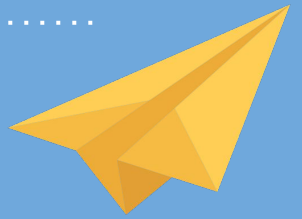
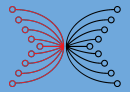


FLIP & Prove

R1CS instances efficiently

Joint work with
Nikitas Paslis, Carla Ràfols

Anca Nitulescu



INPUT | OUTPUT

Outline

1. Main Idea and Motivation
2. From Recursion to Folding
3. FLIP: Fold via IPA
4. Relaxed Groth16

zk-SNARK

Zero-Knowledge
does not leak anything about the witness

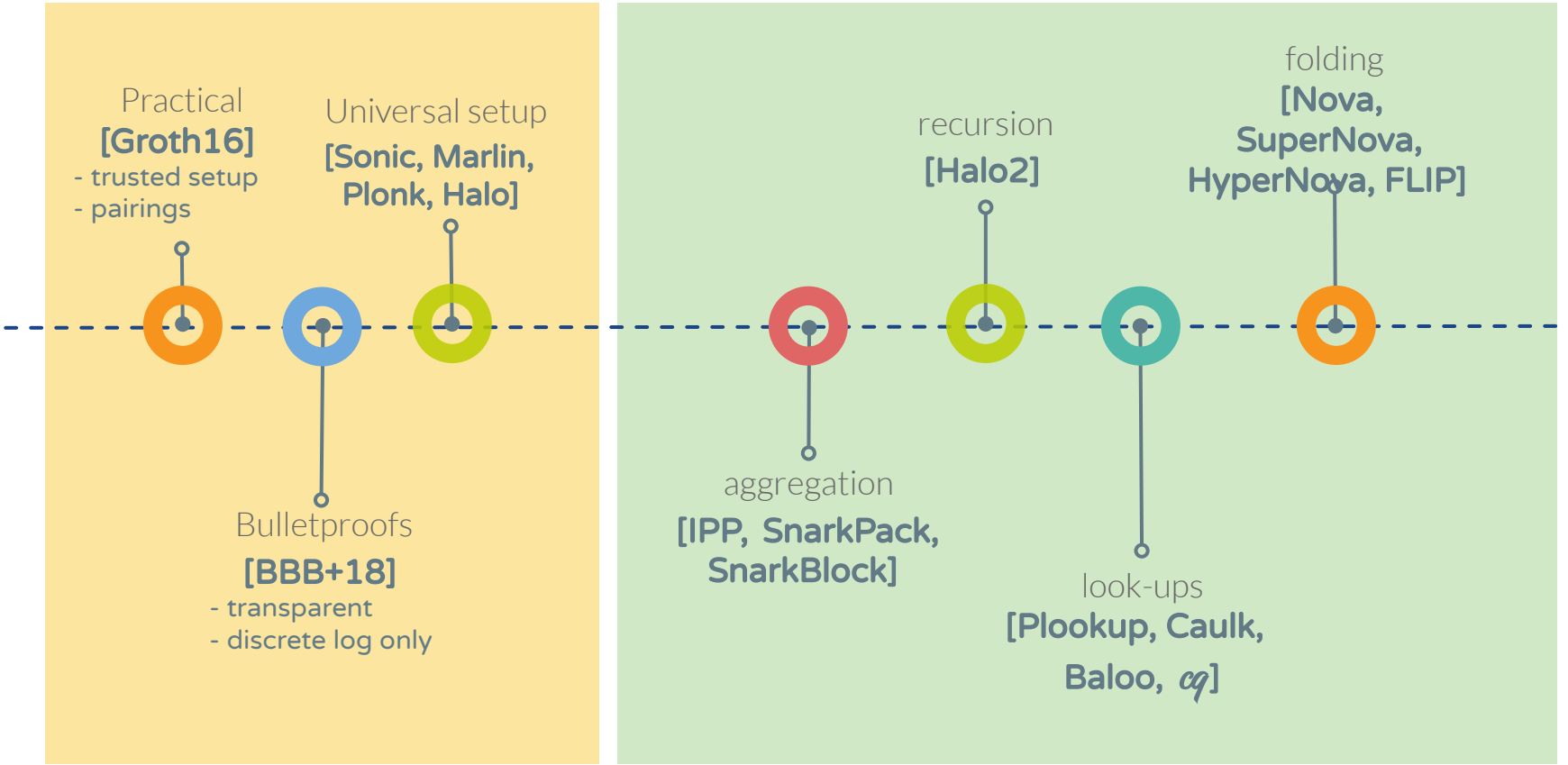
Succinctness
- proof size independent (sublinear) of NP witness size
- fast verification

Non-Interactivity
no exchange between prover and verifier

Knowledge Soundness
a witness can be efficiently extracted from the prover

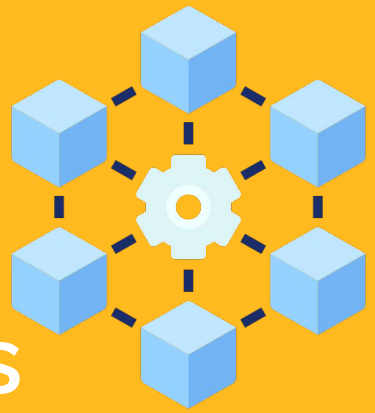
Argument
soundness holds only against computationally bounded provers

Recent zk-SNARK research



Applications

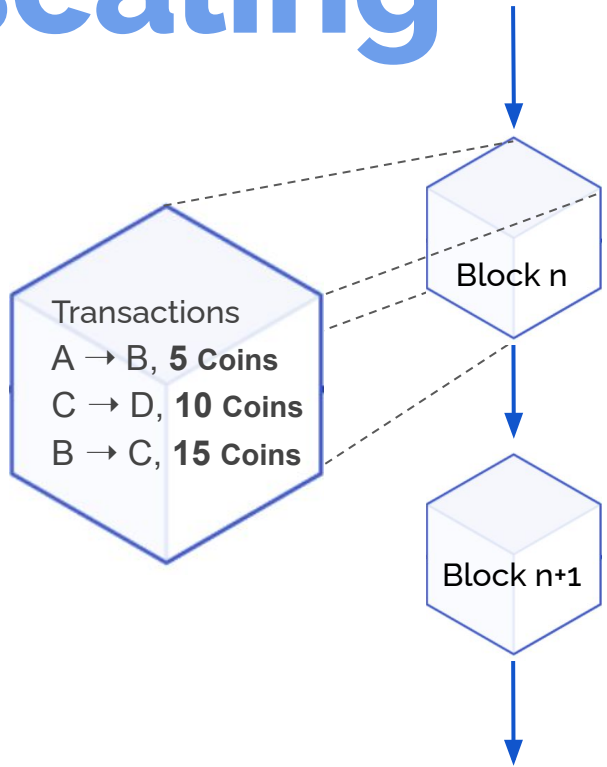
to descentralized systems



Blockchain scaling

ZK Roll-ups

- instead of posting all transaction data on-chain
- execute **tx** using off-chain computation
- a summary of the state changes is published to the chain and verified



Blockchain scaling

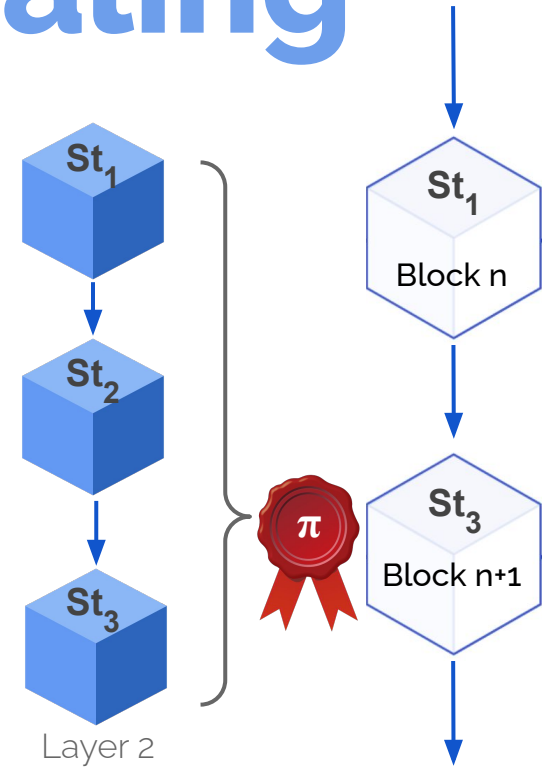
ZK Roll-ups

- instead of posting all transaction data on-chain
- execute **tx** using off-chain computation
- a summary of the state changes is published to the chain and verified

SNARK π proves the correctness of the changes

Statement: state changes proposed by the layer 2 are correct,

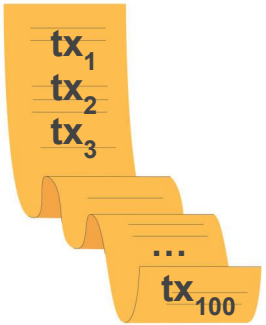
i.e. are the result of the execution of the given batch of transactions

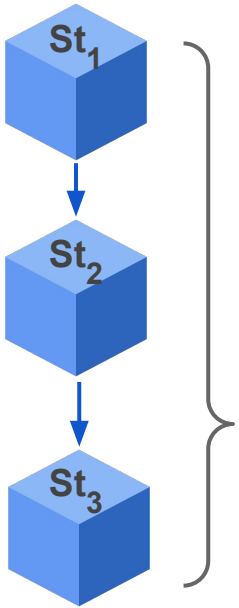


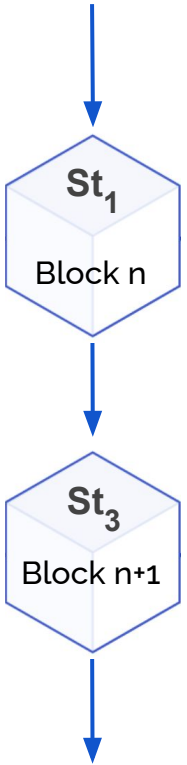
Blockchain scaling

ZK Roll-ups

Prove multiple instances:
- correctness of the execution of batch of **tx**





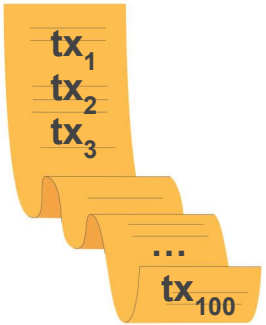


Blockchain scaling

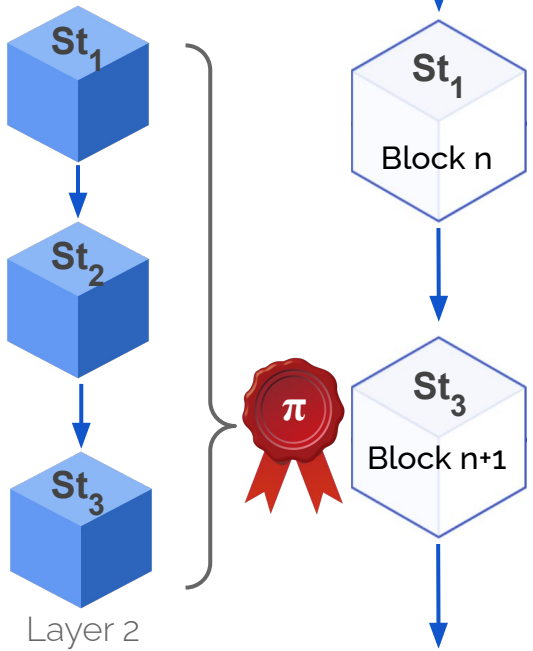
ZK Roll-ups

Example 100 **Tx**

Naively: generate state transition proof once all 100 Tx are submitted → **long delay**



Proving is slow

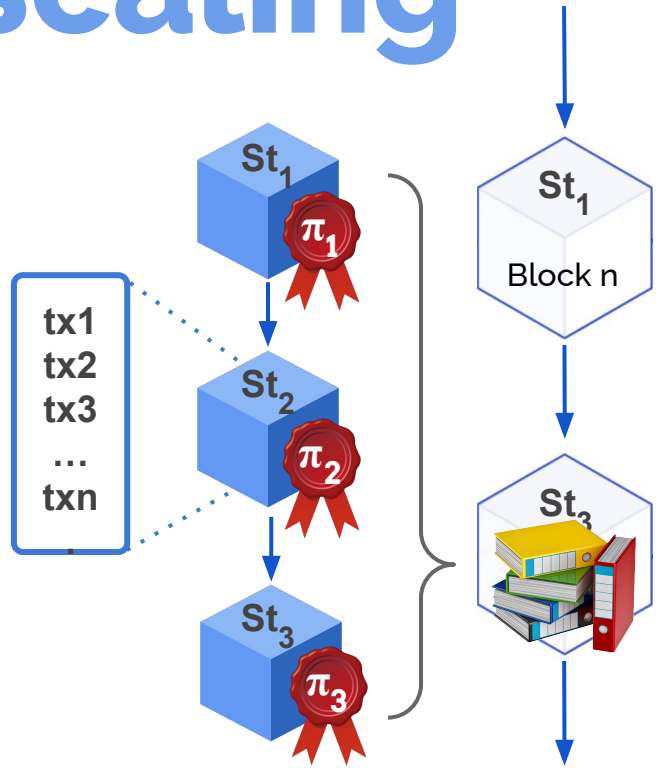


Layer 2

Blockchain scaling

ZK Roll-ups

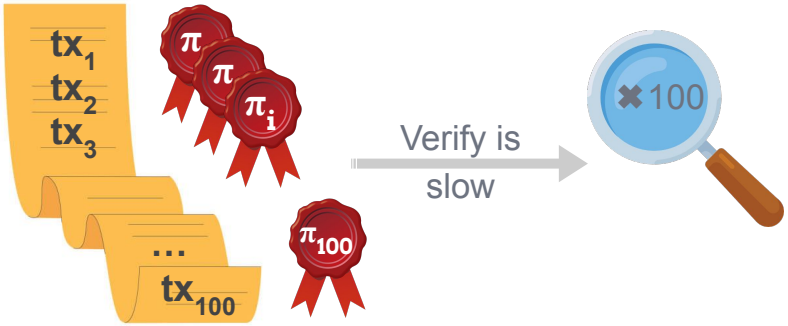
Periodically provide proofs for valid transaction batches

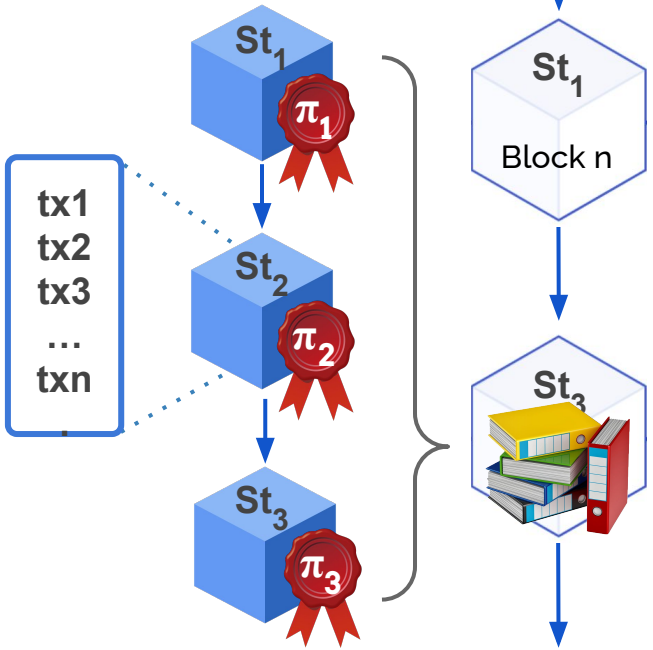


Blockchain scaling

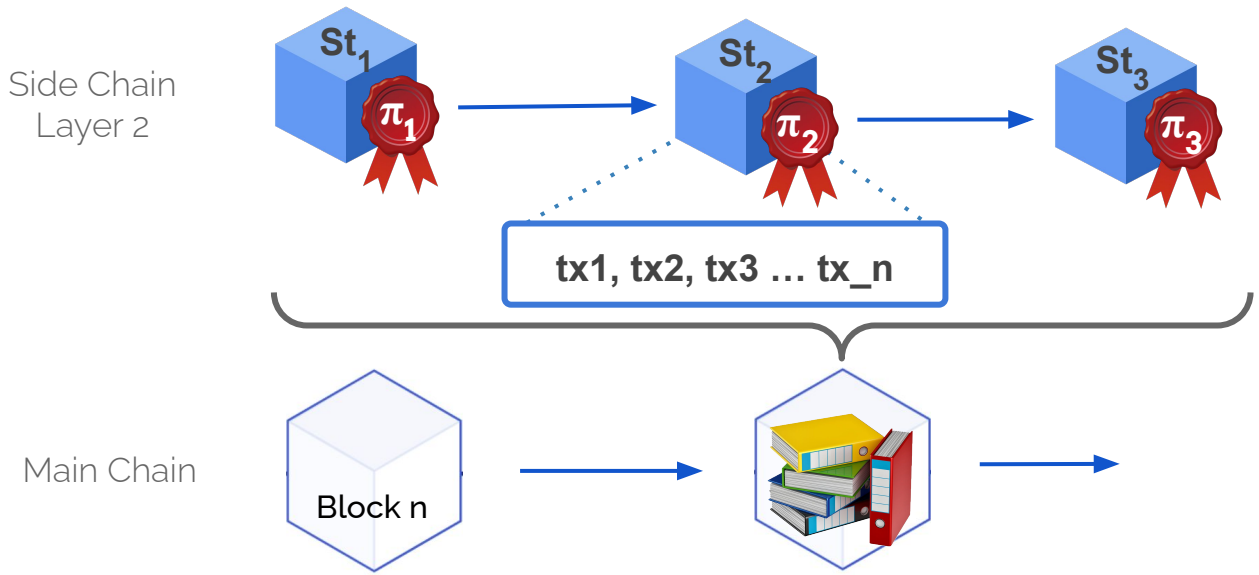
ZK Roll-ups

Periodically provide proofs for valid transaction batches

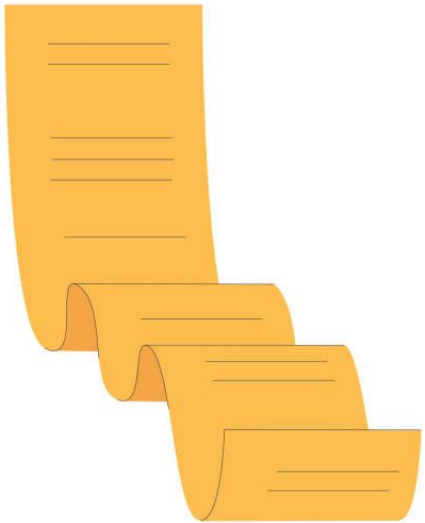




Prove & scale ?

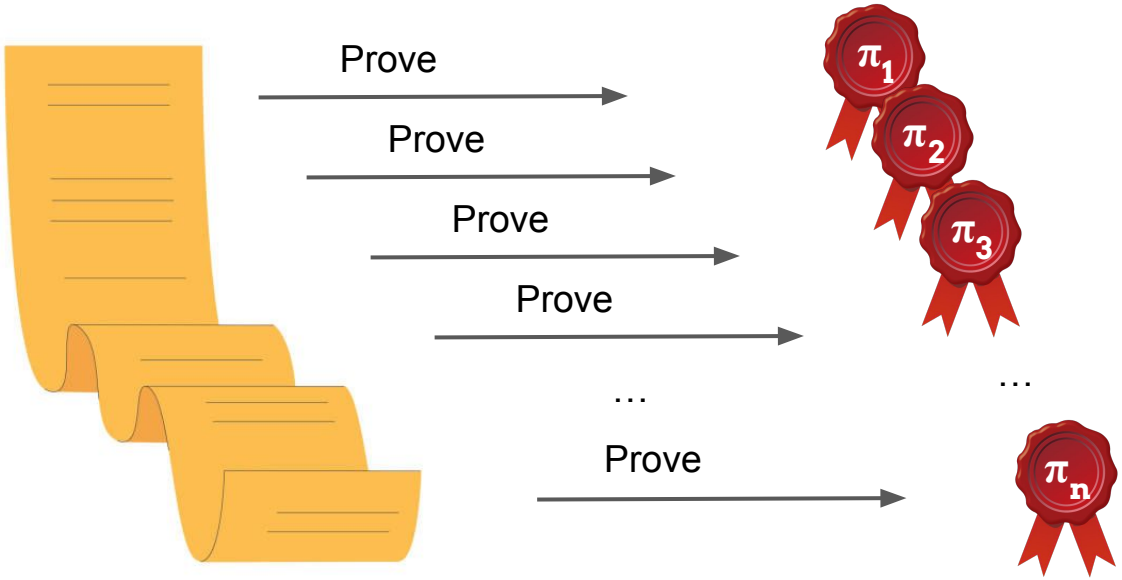

Side Chain
Layer 2
St1
π1
St2
π2
St3
π3
tx1, tx2, tx3 … tx_n
Main Chain
Block n

Different Strategies



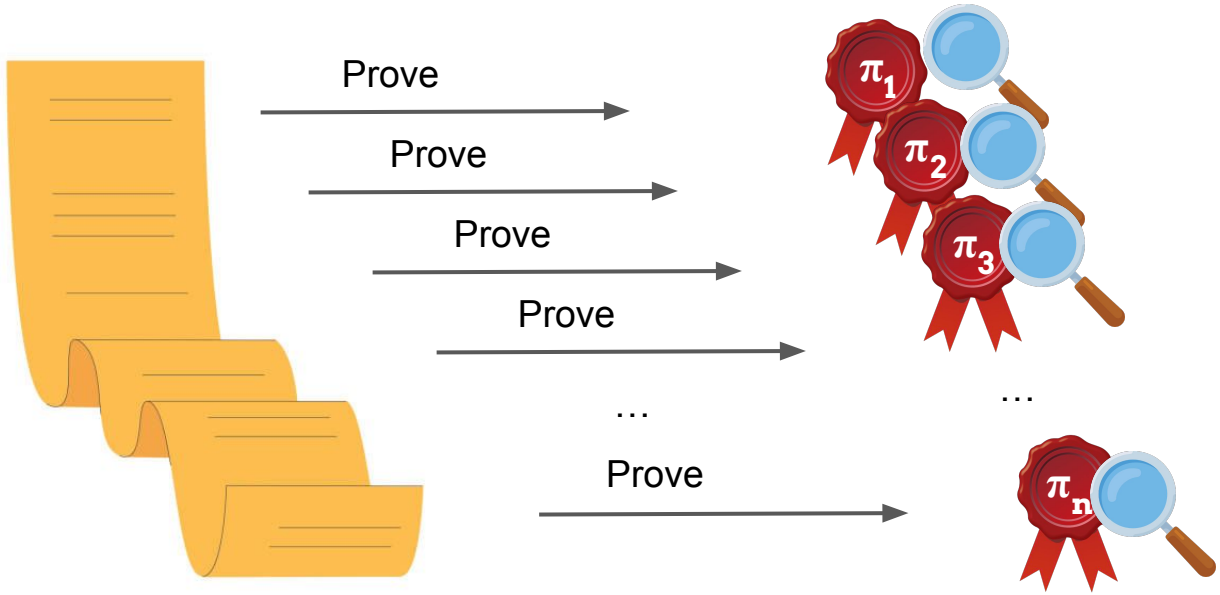
Multiple instances to prove

Naively



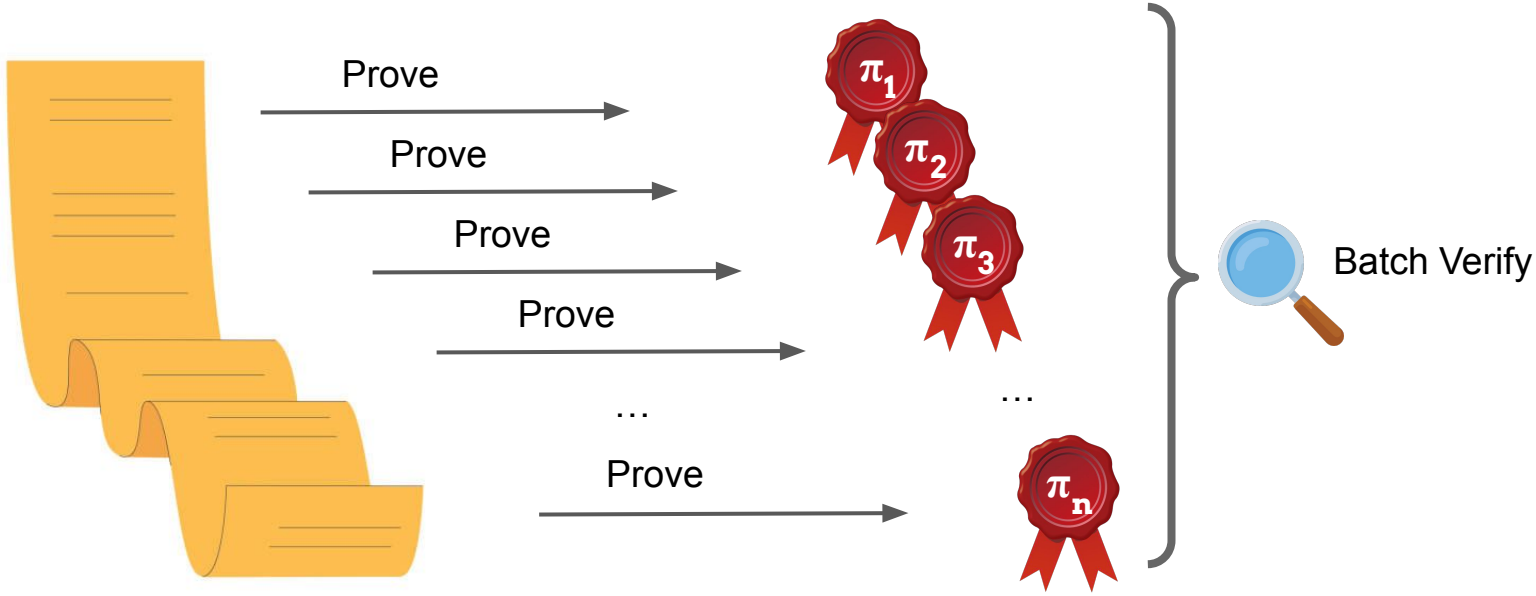
Multiple instances to prove

Naively



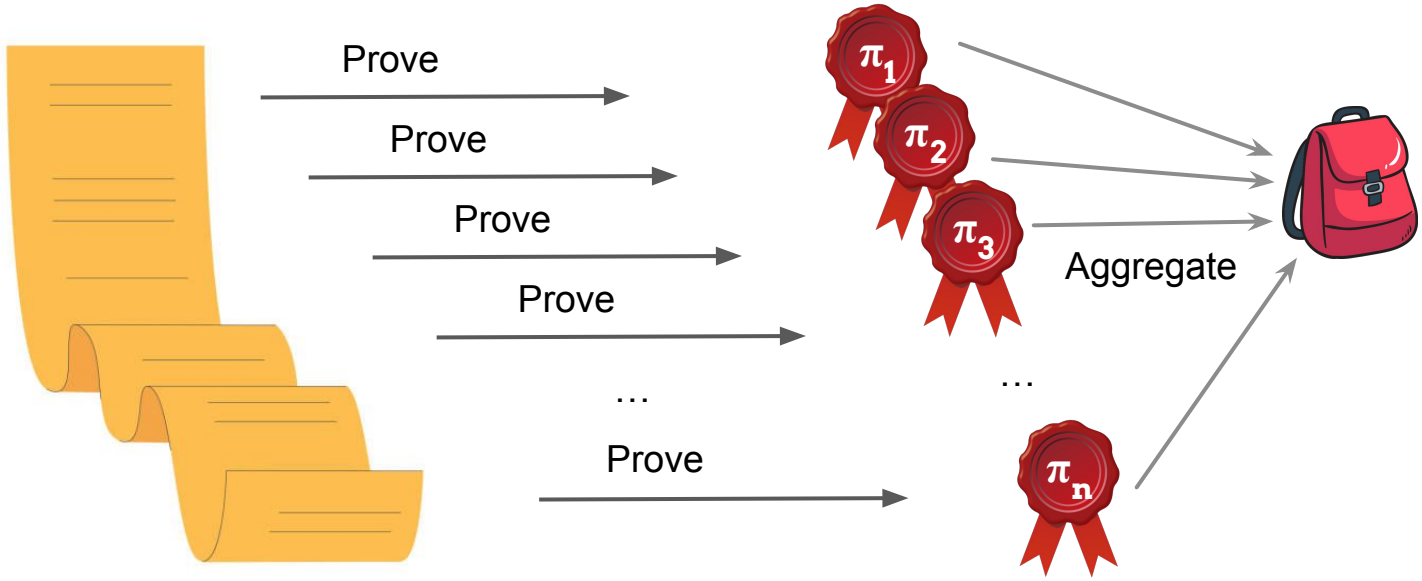
Multiple instances to prove

Batch Verify



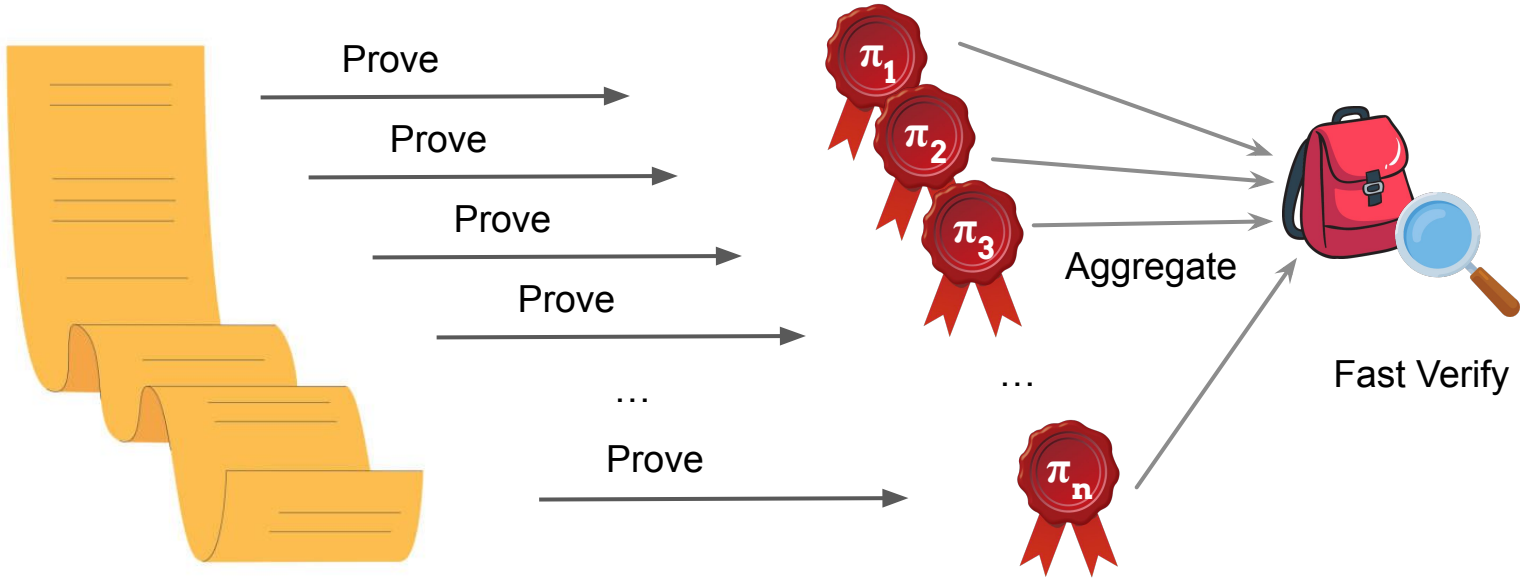
Multiple instances to prove

Aggregation



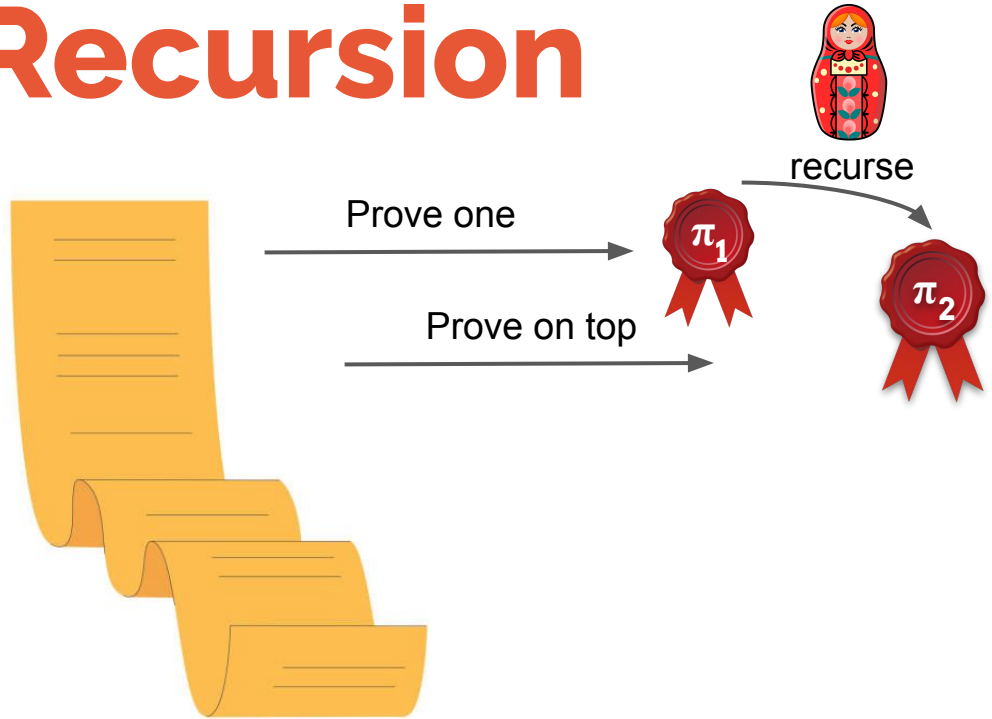
Multiple instances to prove

Aggregation



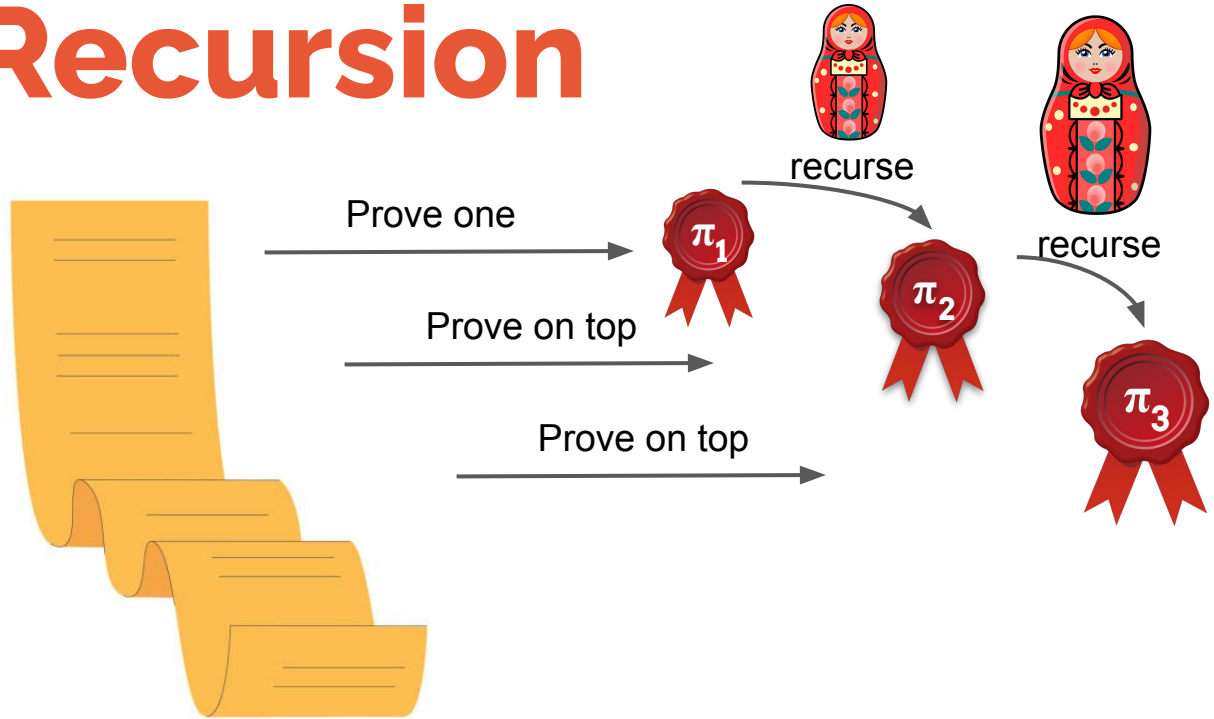
Multiple instances to prove

Recursion



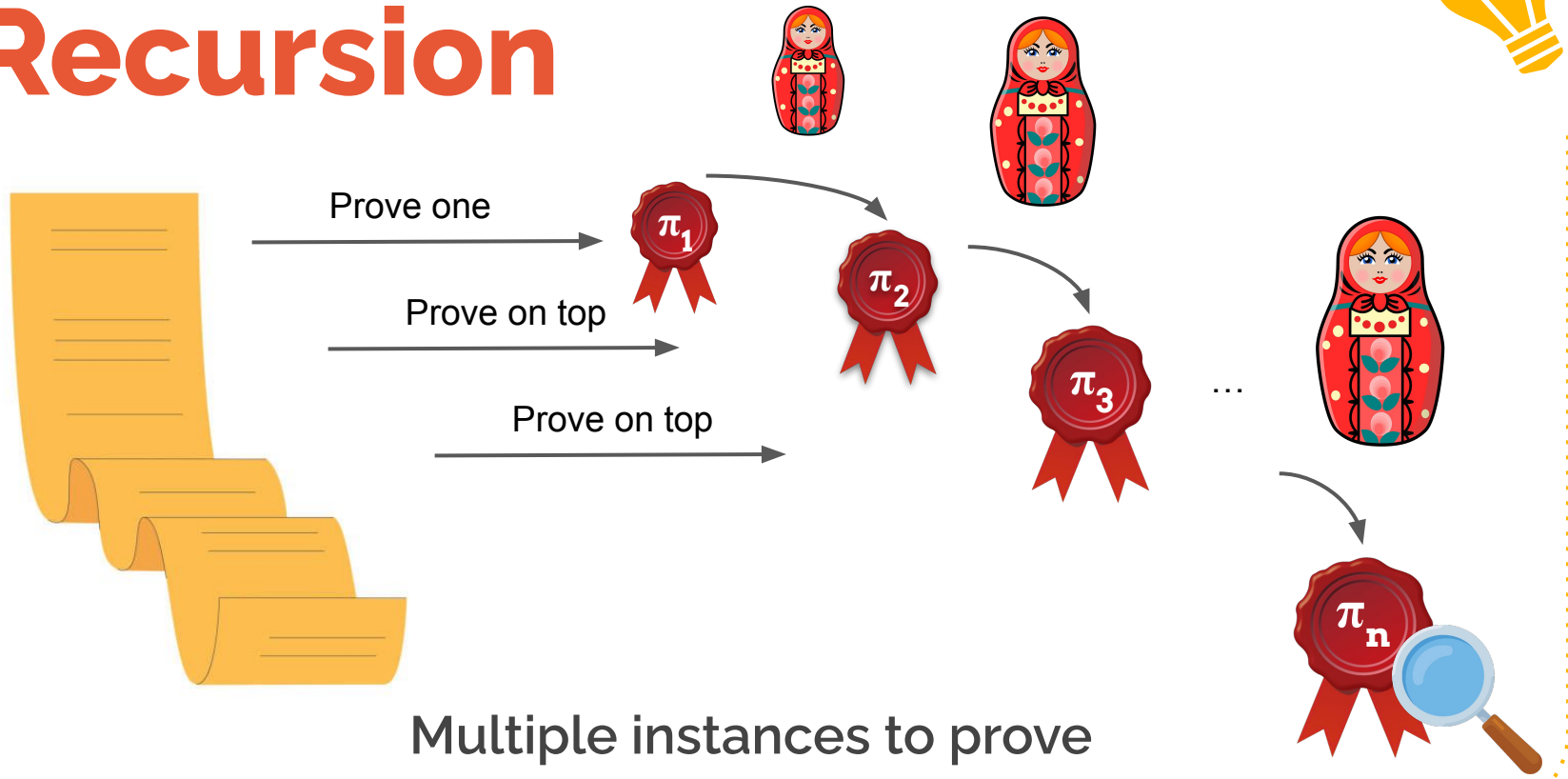
Multiple instances to prove

Recursion



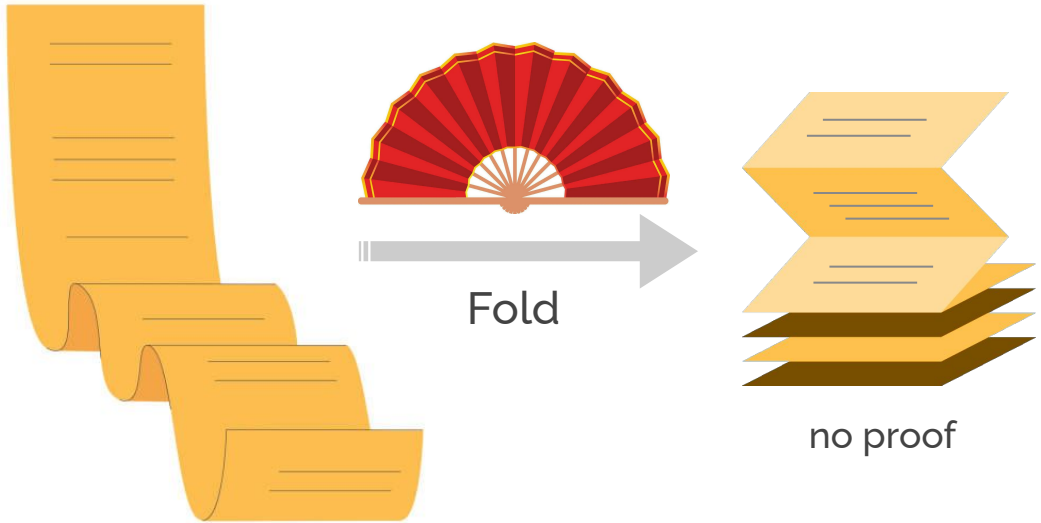
Multiple instances to prove

Recursion



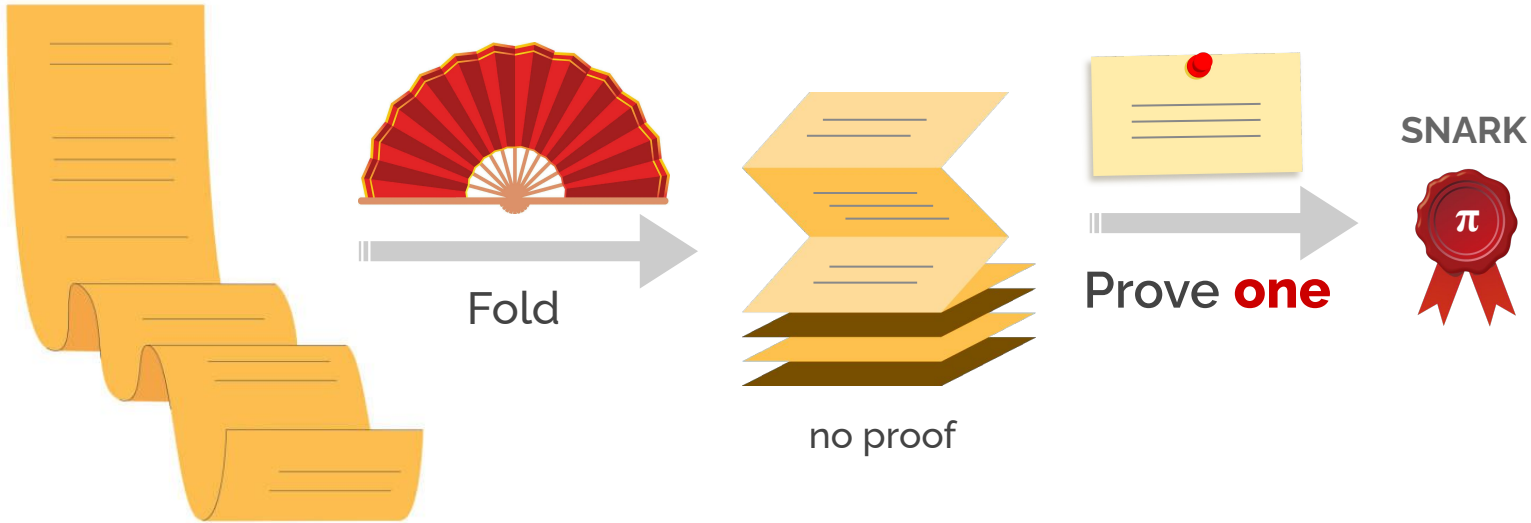
Multiple instances to prove

Folding



Multiple instances to prove

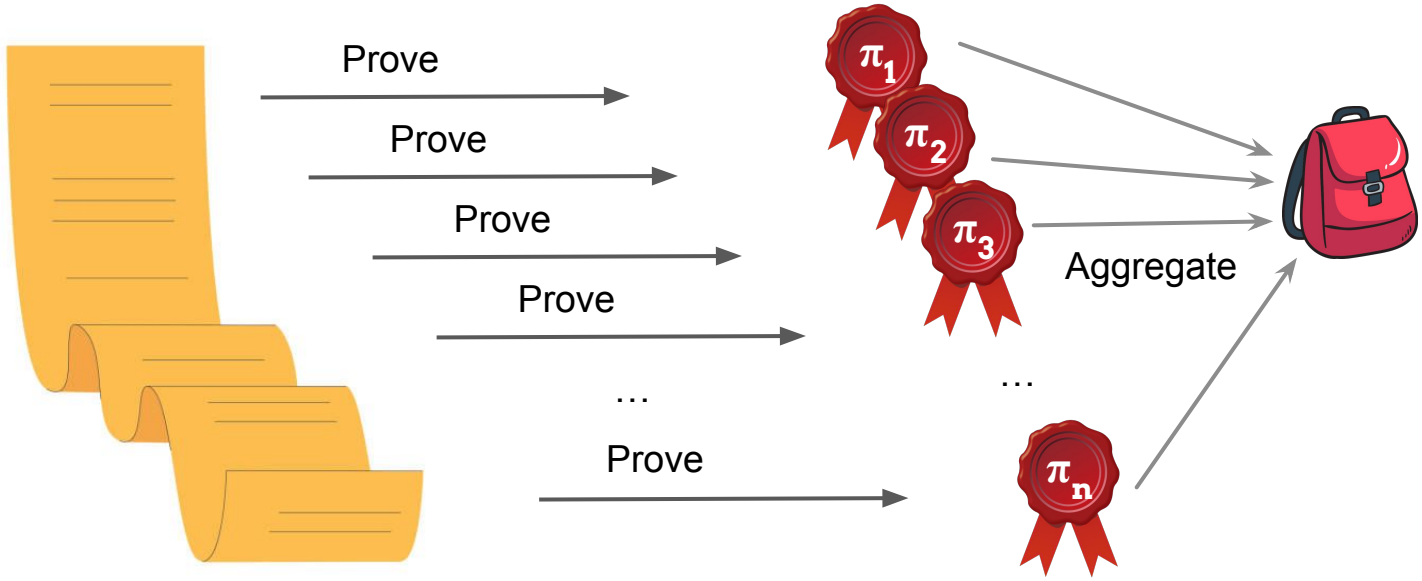
Folding



Multiple instances to prove

SNARK Aggregation

Aggregation



Multiple instances to prove

Distributed storage

Storage Providers

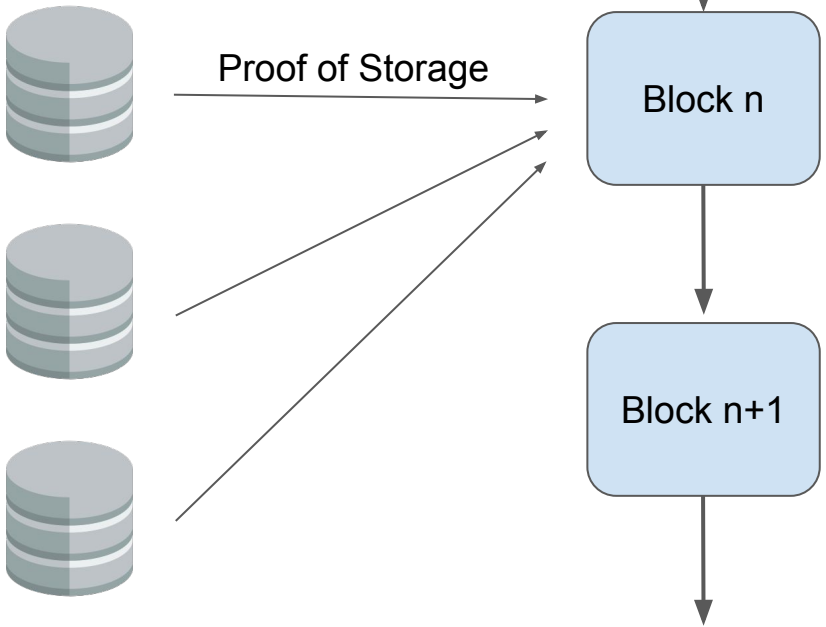
- onboard storage capacity
- earn block rewards
- regularly prove the storage

= **Provers**

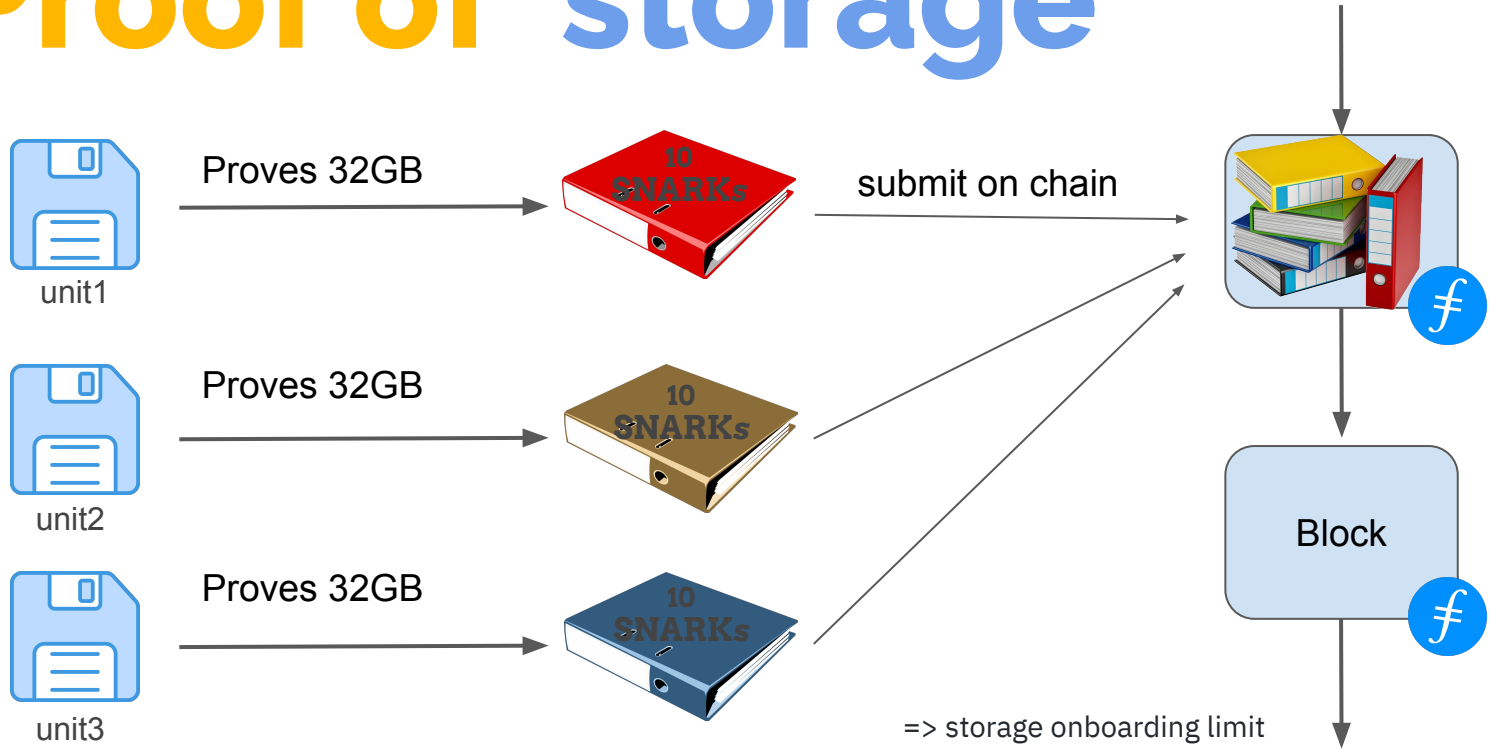
Nodes in network

- ensure data is being stored, maintained, and secured
- need to check proofs of space

= **Verifiers**



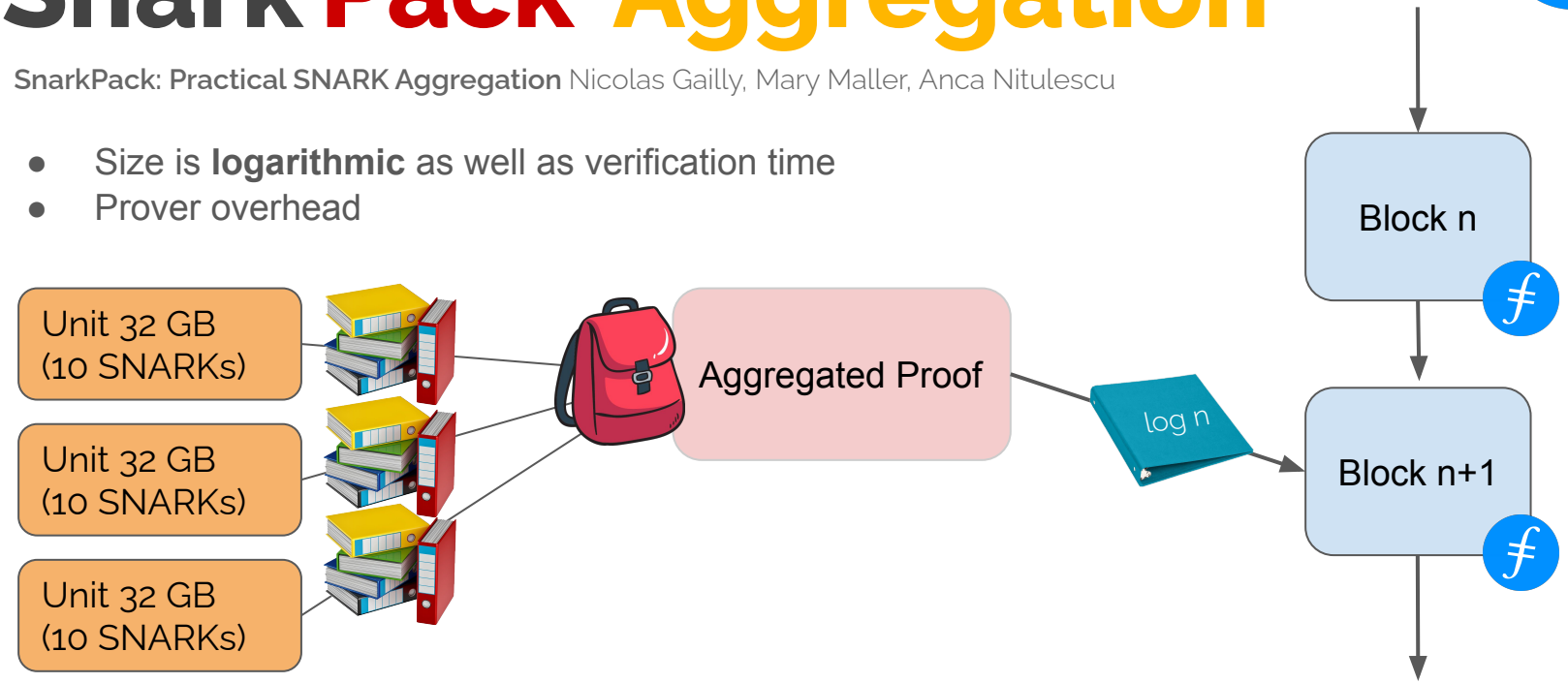
Proof of storage



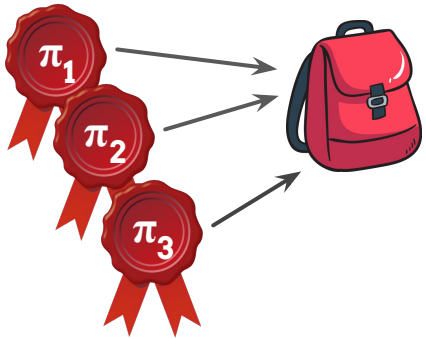
Snark Pack Aggregation

SnarkPack: Practical SNARK Aggregation Nicolas Gailly, Mary Maller, Anca Nitulescu

- Size is **logarithmic** as well as verification time
- Prover overhead



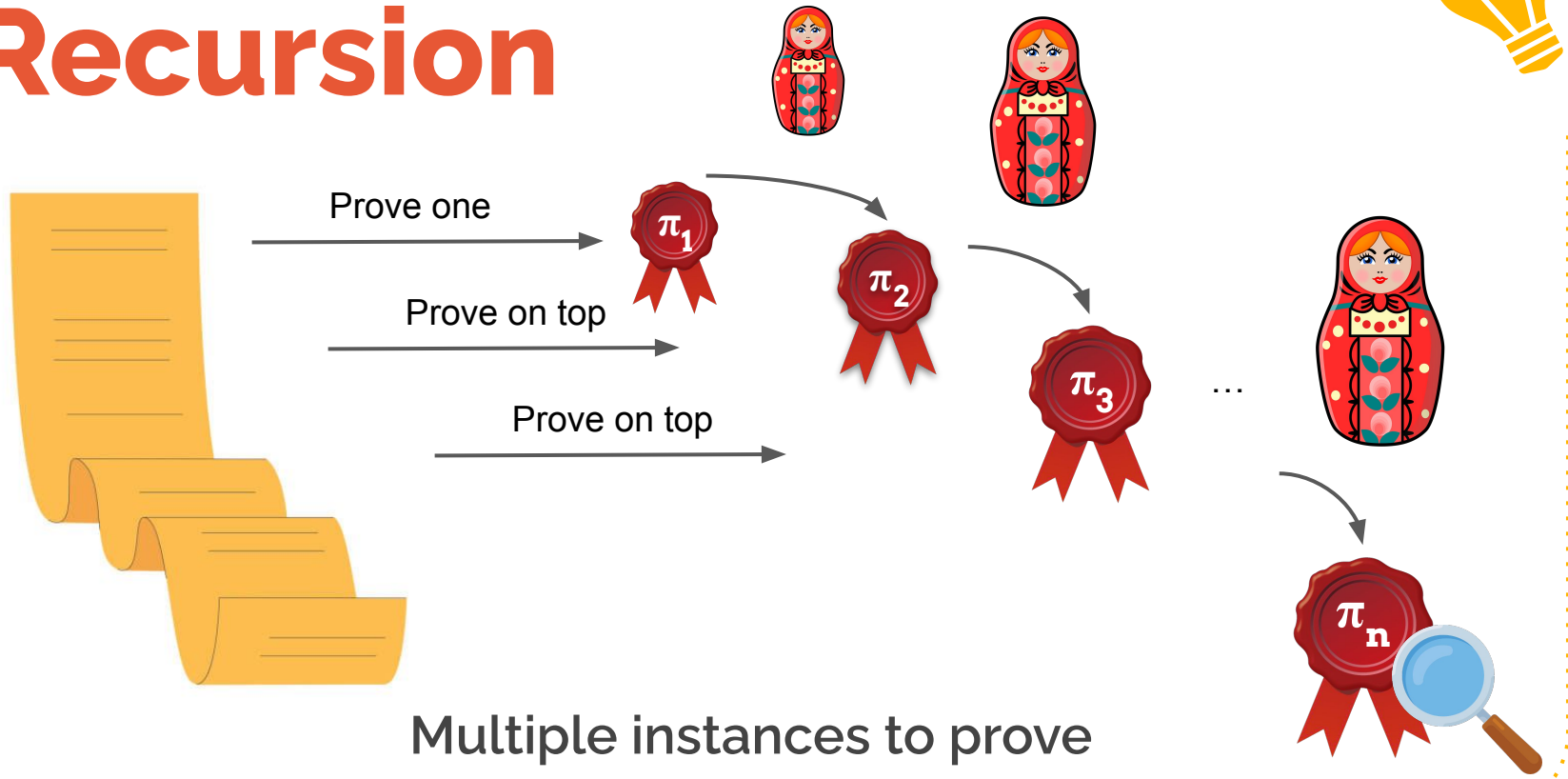
Snark Pack Aggregation



- **Groth16**: Verifier–only algebraic checks (pairings)
- **proofs** for same relation → extension to different relations
- **Extension to other SNARKs:**
  - Main blocker: **Random Oracle** in Universal SNARKs
  - *aPlonK* – Requires ahead coordination between provers!
- **Transparent Aggregation:** What about Aggregating SNARKs without a trusted setup?

SNARK Recursion

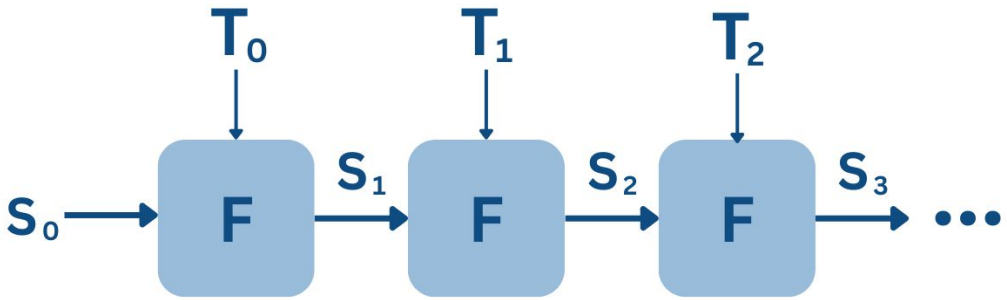
Recursion



Multiple instances to prove

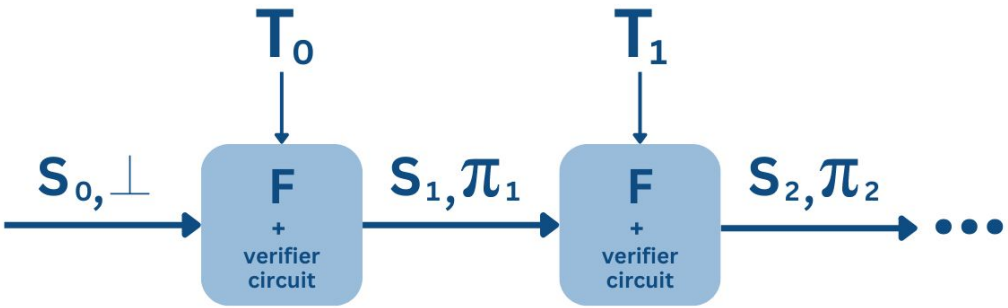
IVC

Incremental Verifiable Computation

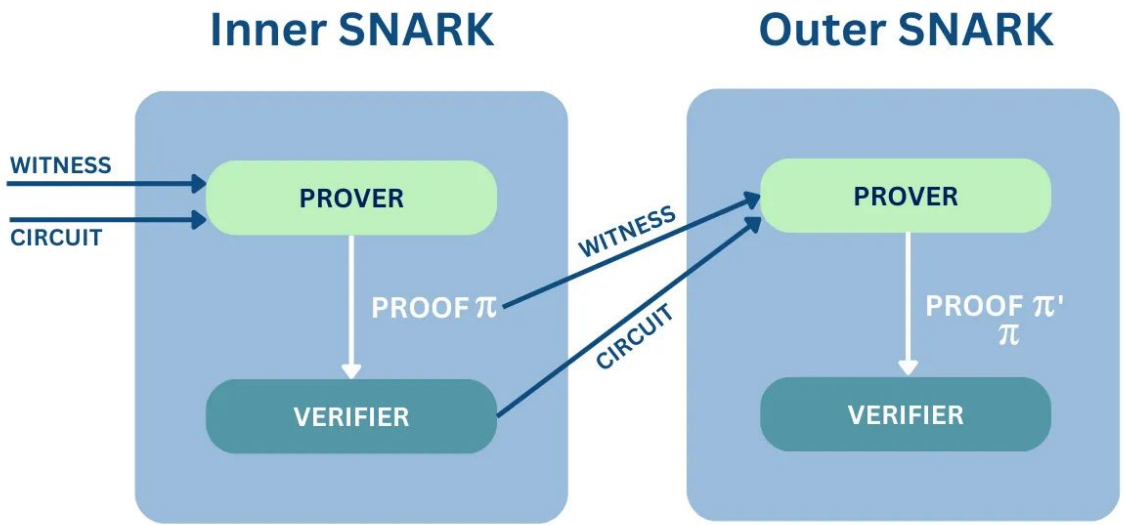


IVC

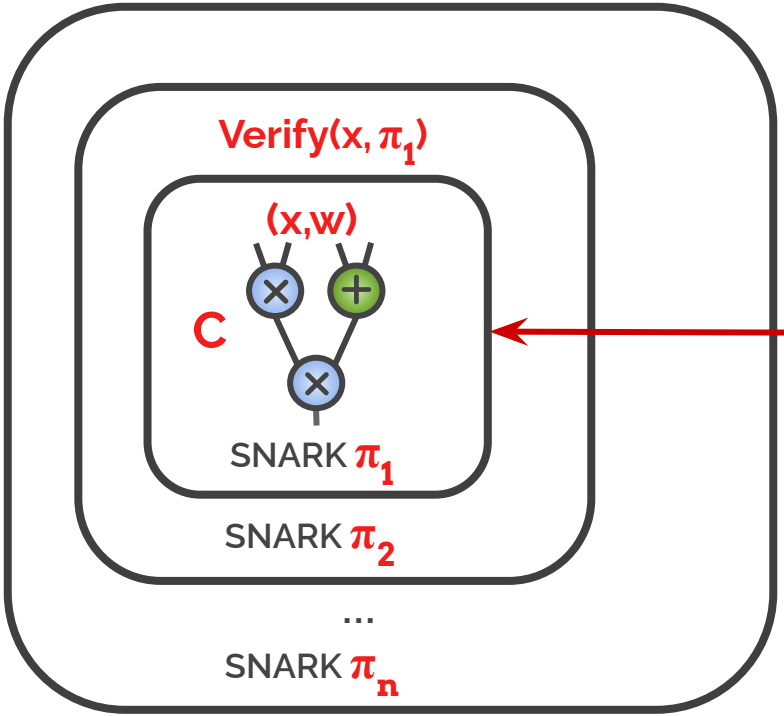
Incremental Verifiable Computation



Recursion


Inner SNARK
Outer SNARK
WITNESS
CIRCUIT
PROVER
PROOF π
VERIFIER
WITNESS
CIRCUIT
PROVER
PROOF π'
π
VERIFIER

Proof Recursion

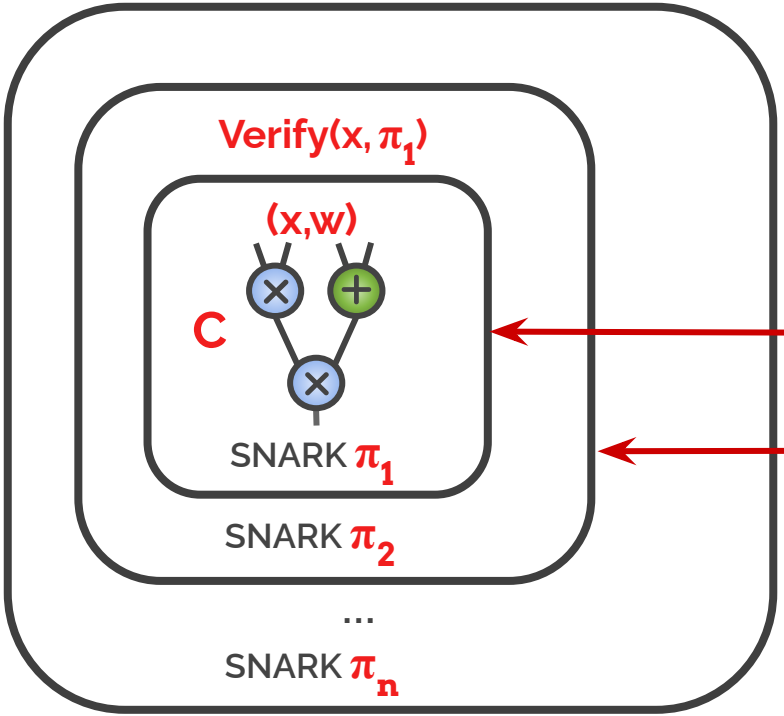


Verification Runtime

$|\text{Verify}| = 2|C_{\text{SNARK}}|$

$|C_{\text{Verify}_1}| = 2|C_{\text{SNARK}}|$

Proof Recursion



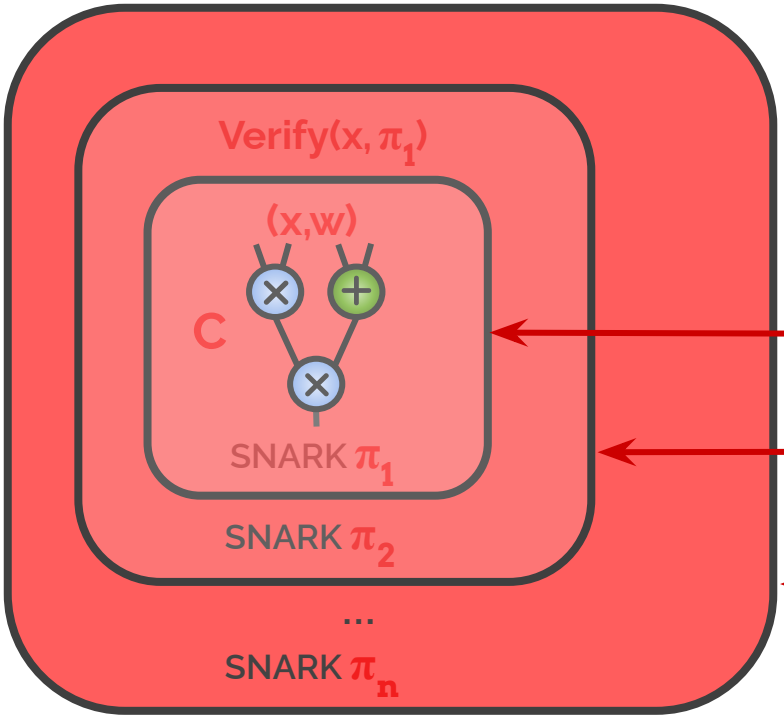
Verification Runtime

$|\text{Verify}| = 2|C_{\text{SNARK}}|$

$|C_{\text{Verify}_1}| = 2|C_{\text{SNARK}}|$

$|C_{\text{Verify}_2}| = 2|C_{\text{Verify}_1}| = 2^2|C_{\text{SNARK}}|$

Linear Verfication!!!

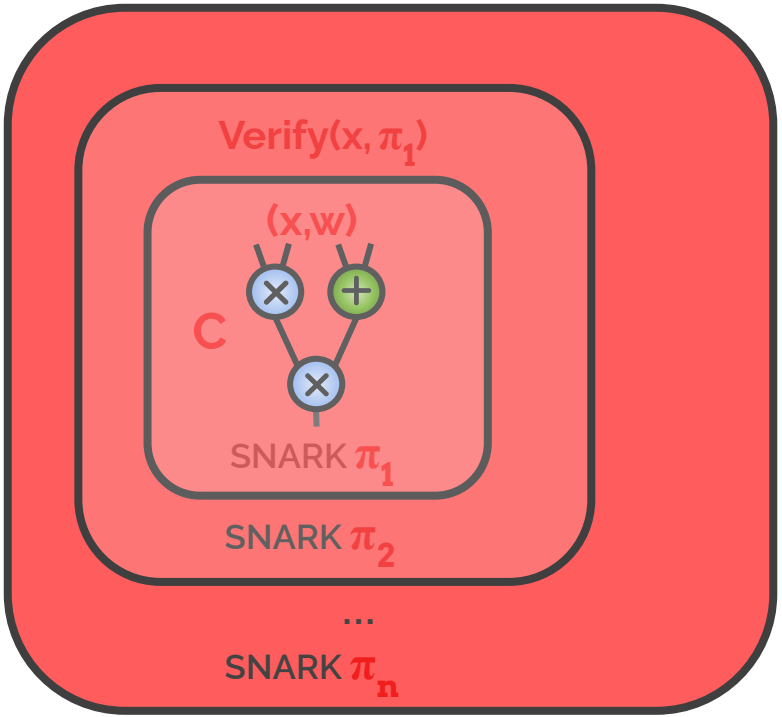


$|C_{Verify_1}| = 2|C_{SNARK}|$

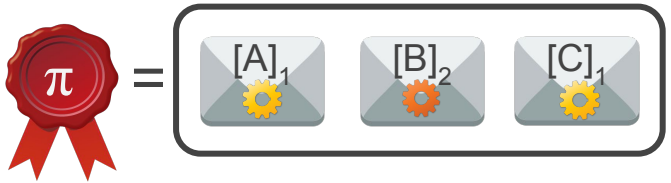
$|C_{Verify_2}| = 2|C_{Verify_1}| = 2^2|C_{SNARK}|$

$|C_{Verify_3}| = 2|C_{Verify_2}| = 2^3|C_{SNARK}|$

Sublinear Verfication

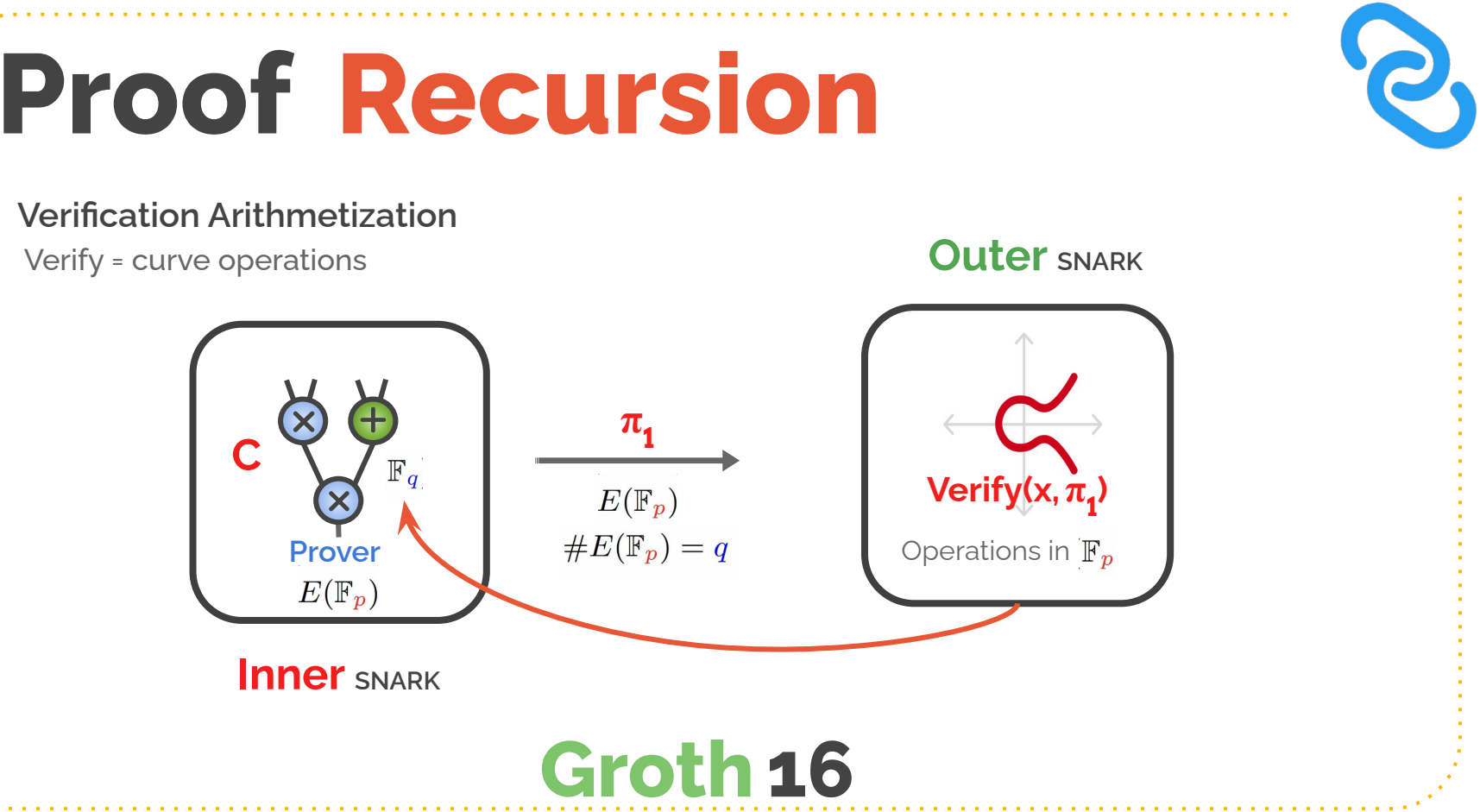


Groth 16



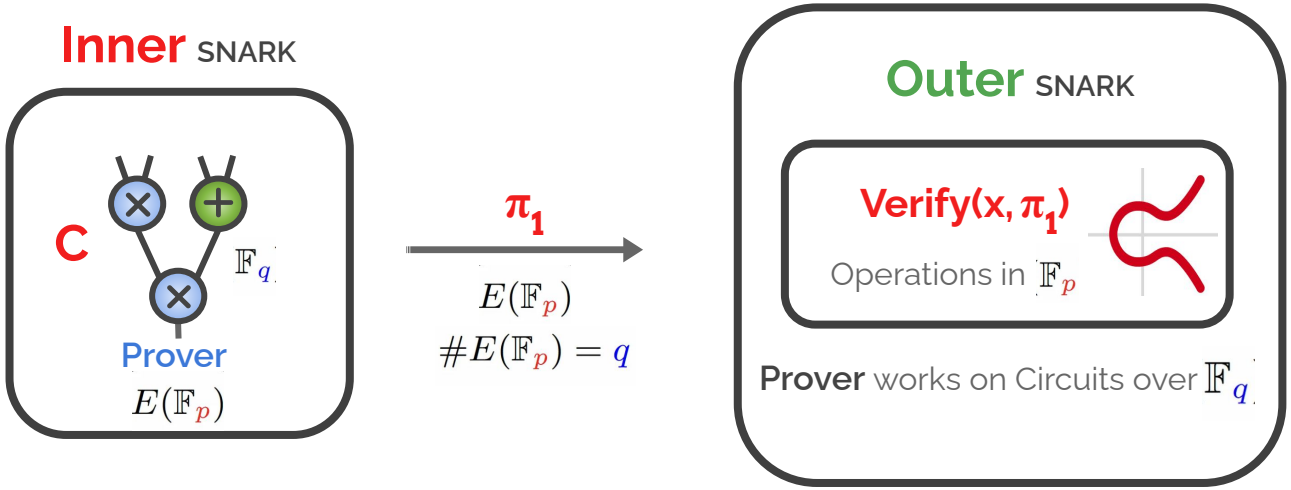
$e([A]_1, [B]_2) = e([C]_1, in)$

Proof Recursion

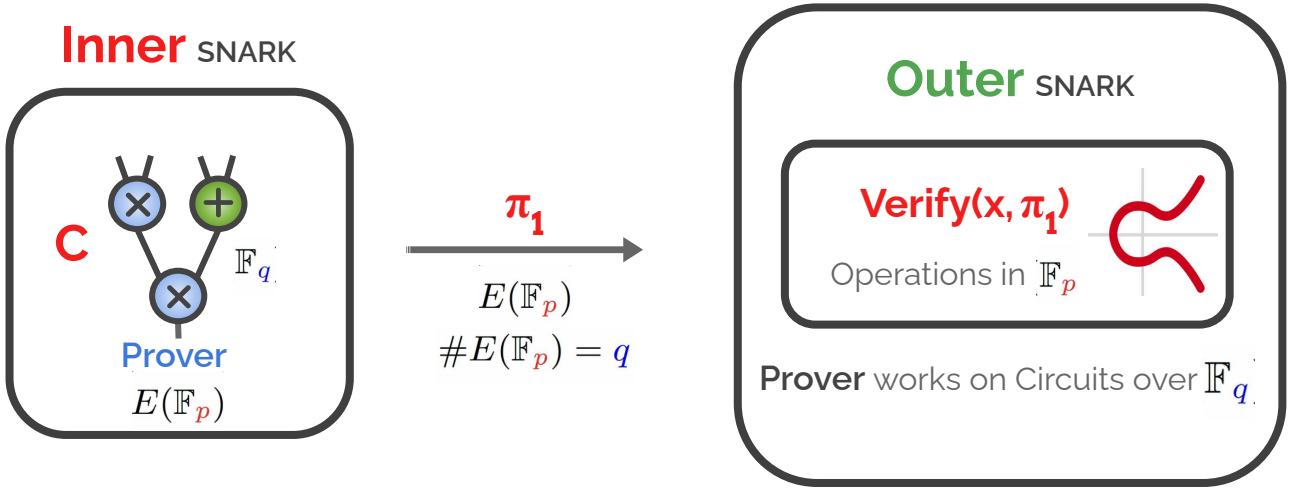


Groth 16

Proof Recursion



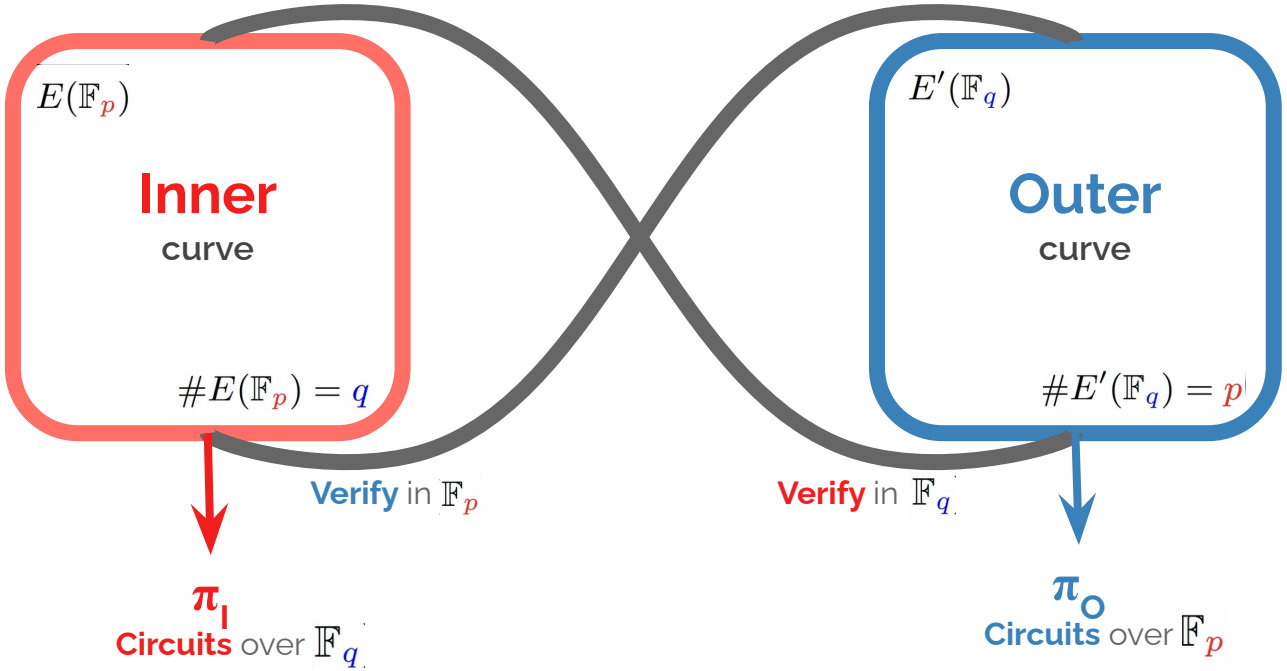
Proof Recursion



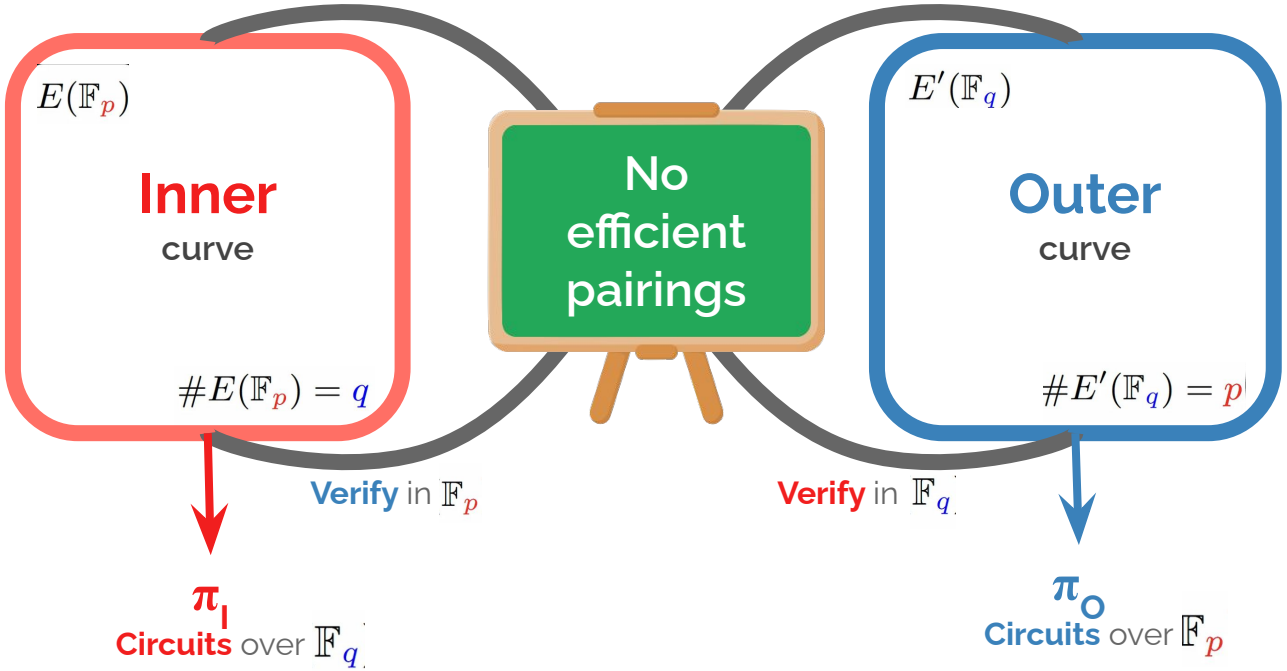
p = q

Dlog is easy to solve :(

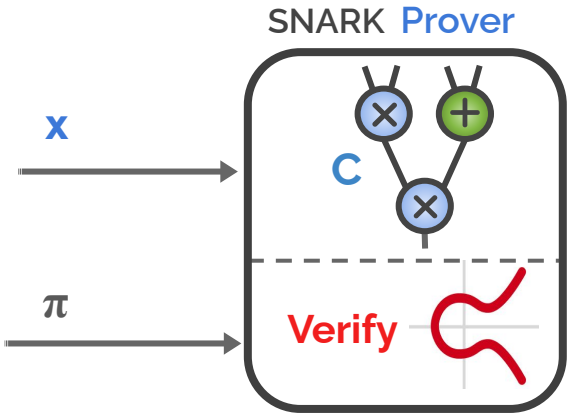
Cycles of Curves



Cycles of Curves

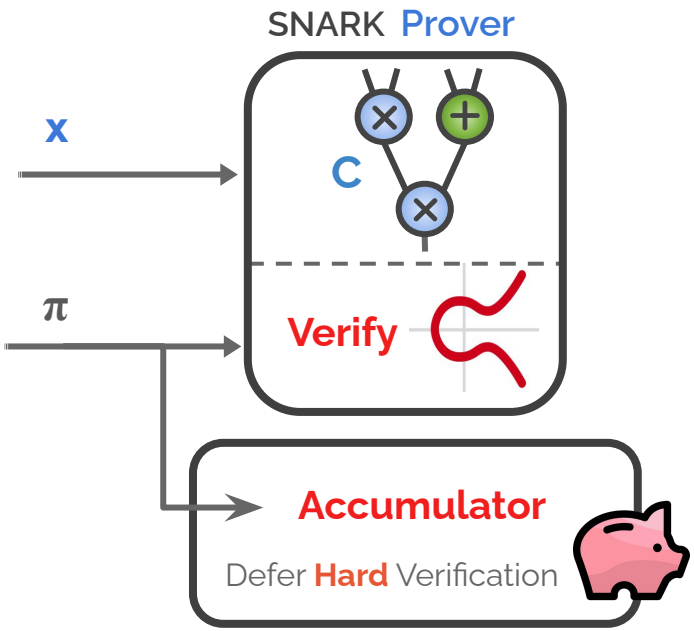


Atomic Accumulation

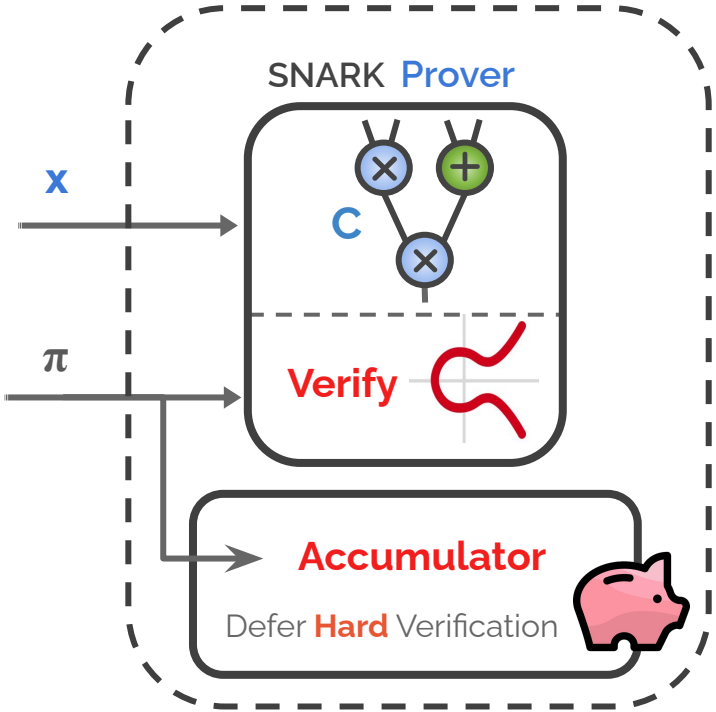


Back to Linear Verfication

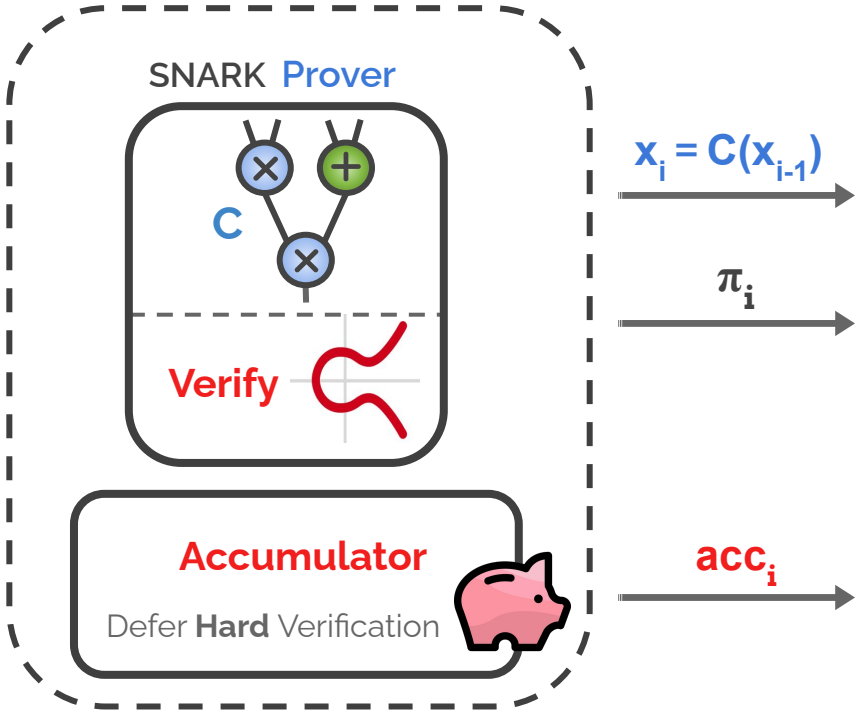
Atomic Accumulation



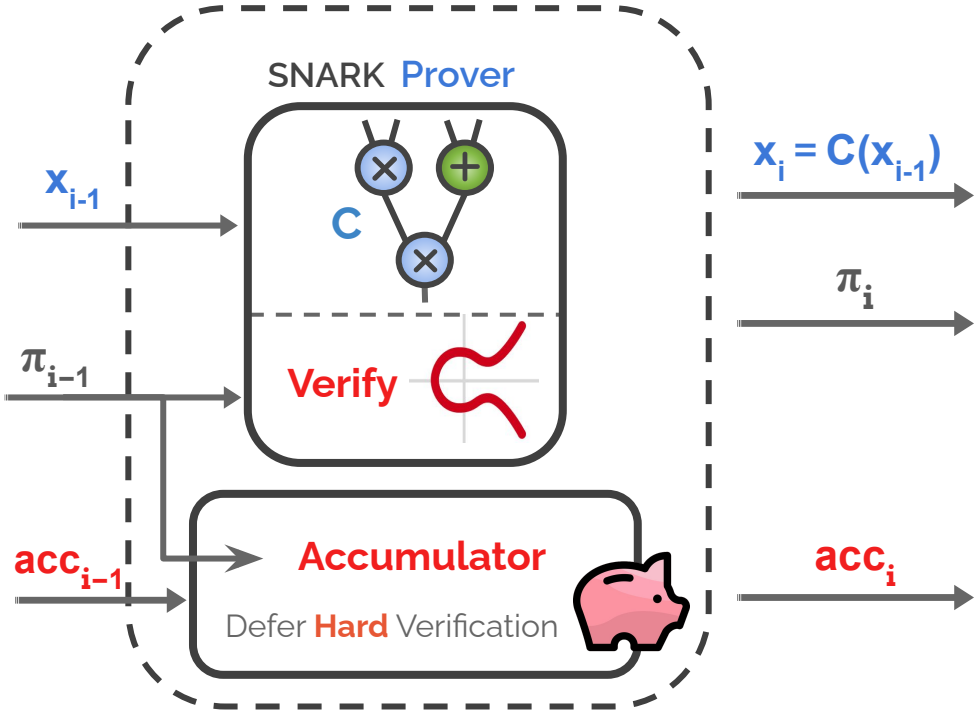
Atomic Accumulation



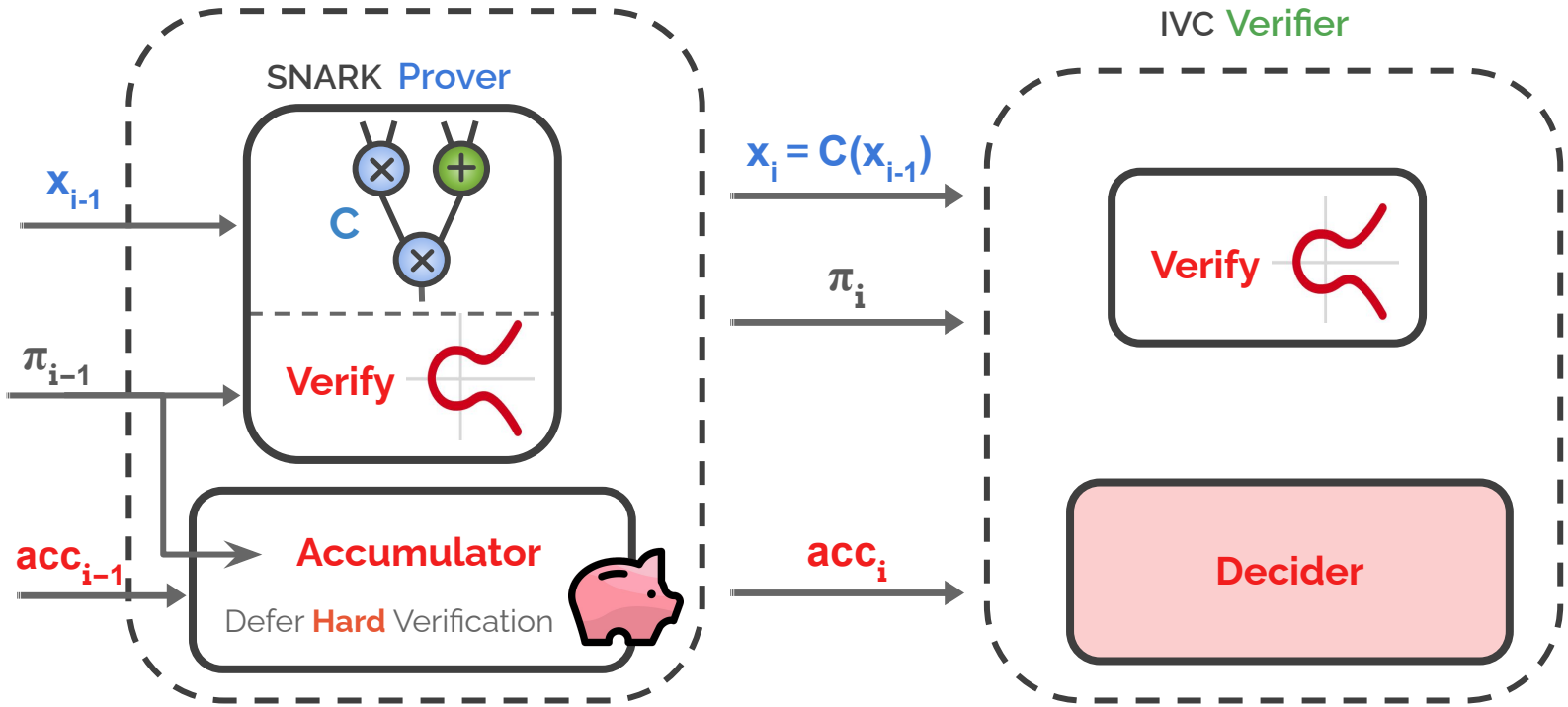
Atomic Accumulation



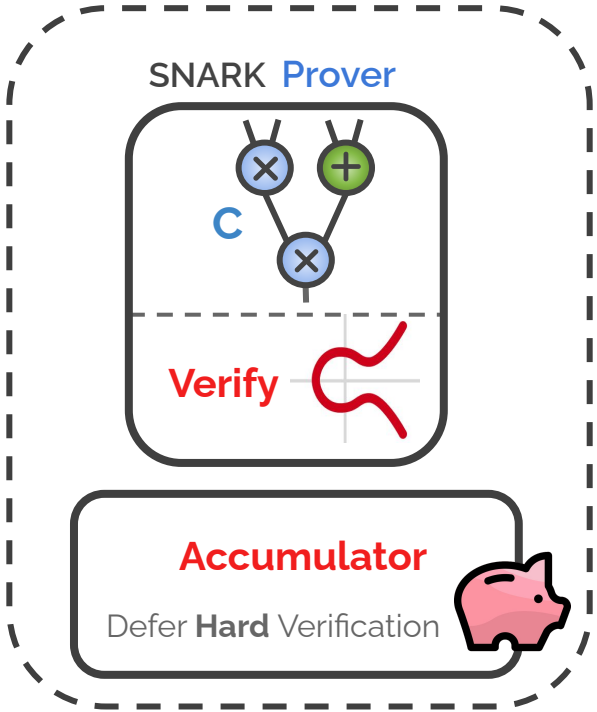
Atomic Accumulation



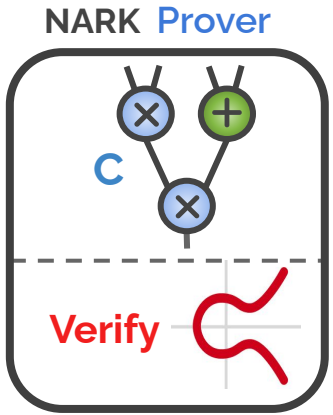
Atomic Accumulation



Atomic Accumulation



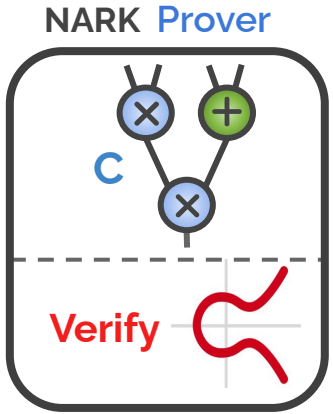
Split Accumulation



NARK:

$|\pi| = O(|C|)$

Split Accumulation



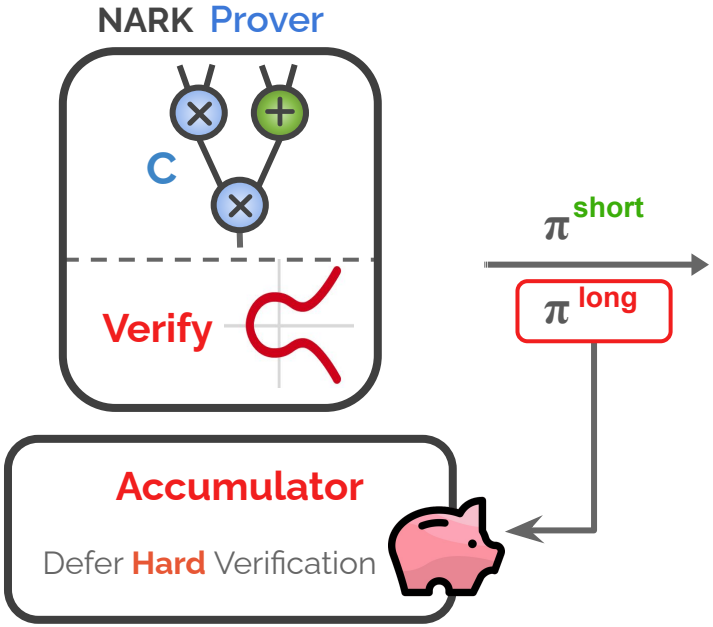
π^{short}

π^{long}

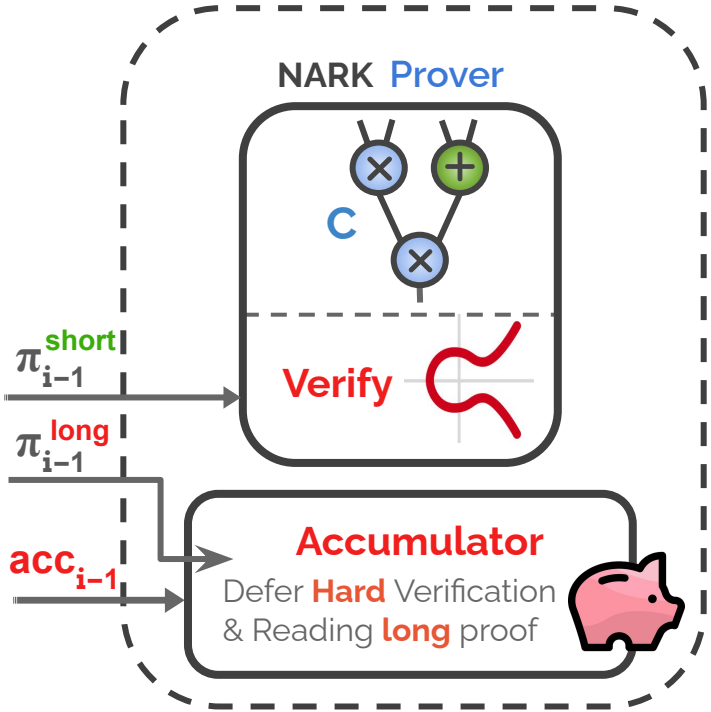
NARK:

$|\pi^{long}| = O(|C|)$

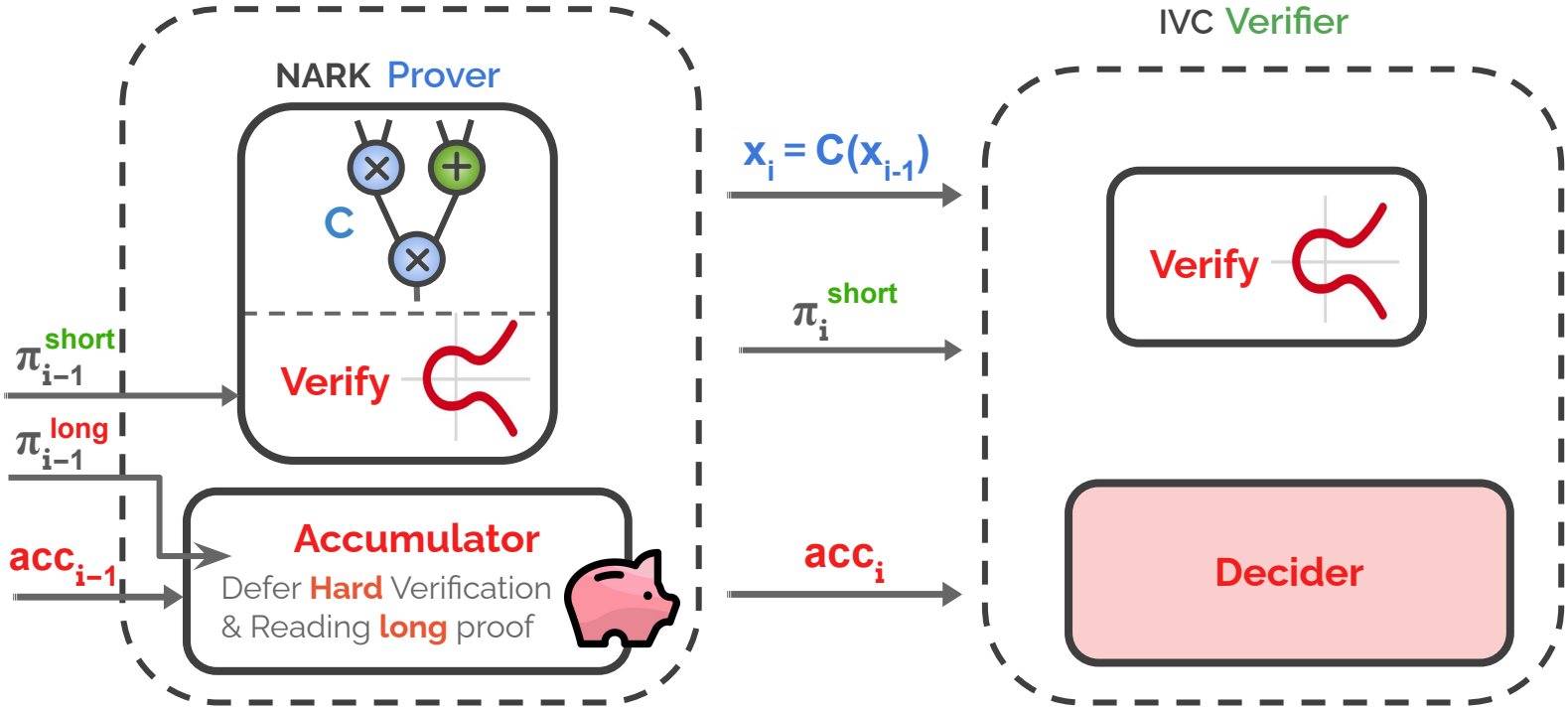
Split Accumulation



Split Accumulation



Split Accumulation

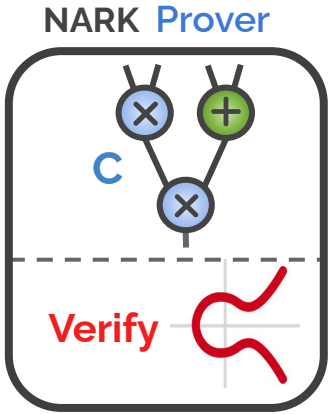


Recursion overview

Recursion technique	Overhead	Examples
Naive recursion	Read whole proof, run full verifier Defer nothing	Plonky2, recursive STARK, Fractal
Atomic Accumulation	Read whole proof, run partial verifier, Defer hard verification	Halo/Halo2, [BCMS20]
Split Accumulation*	Read partial proof, run partial verifier, Defer hard verification and reading whole proof	[BCLMS21] Proof-Carrying Data without Succinct Arguments

*also called folding

Folding Schemes



π

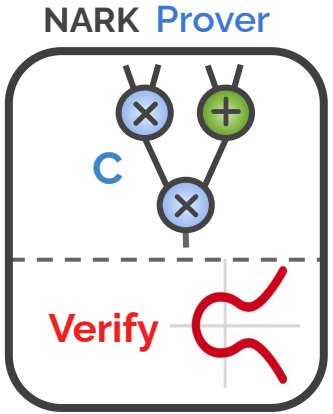
Why do we need a Prover if

|proof| = |witness| ?

NARK:

$|\pi| = O(|C|)$

Folding Schemes



NARK:

$|\pi| = O(|C|)$

π

Why do we need a Prover if

|proof| = |witness| ?

Idea: Fold directly the witness!

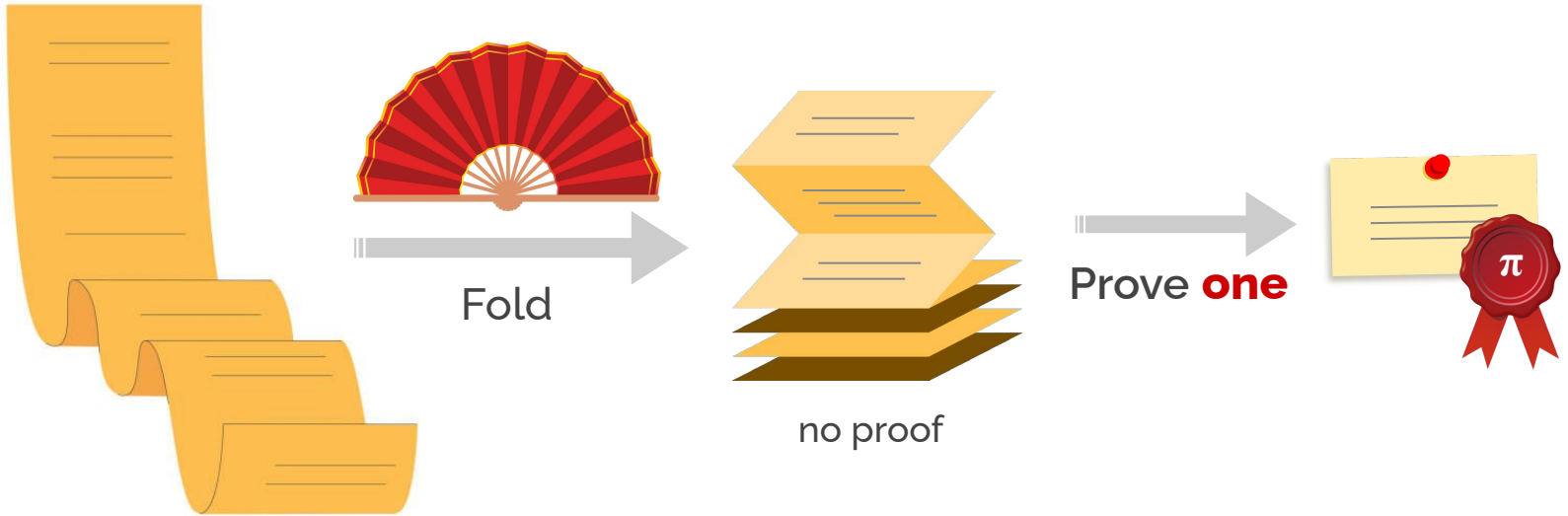
Recursion overview

Recursion technique	Overhead	Examples
Naive recursion	Read whole proof, run full verifier Defer nothing	Plonky2, recursive STARK, Fractal
Atomic Accumulation	Read whole proof, run partial verifier, Defer hard verification	Halo/Halo2, [BCMS20]
Split Accumulation*	Read partial proof, run partial verifier, Defer hard verification and reading whole proof	[BCLMS21] Proof-Carrying Data without Succinct Arguments
Folding Schemes	Read unproven instances-witness, compress them Defer proving	Nova

*also called folding

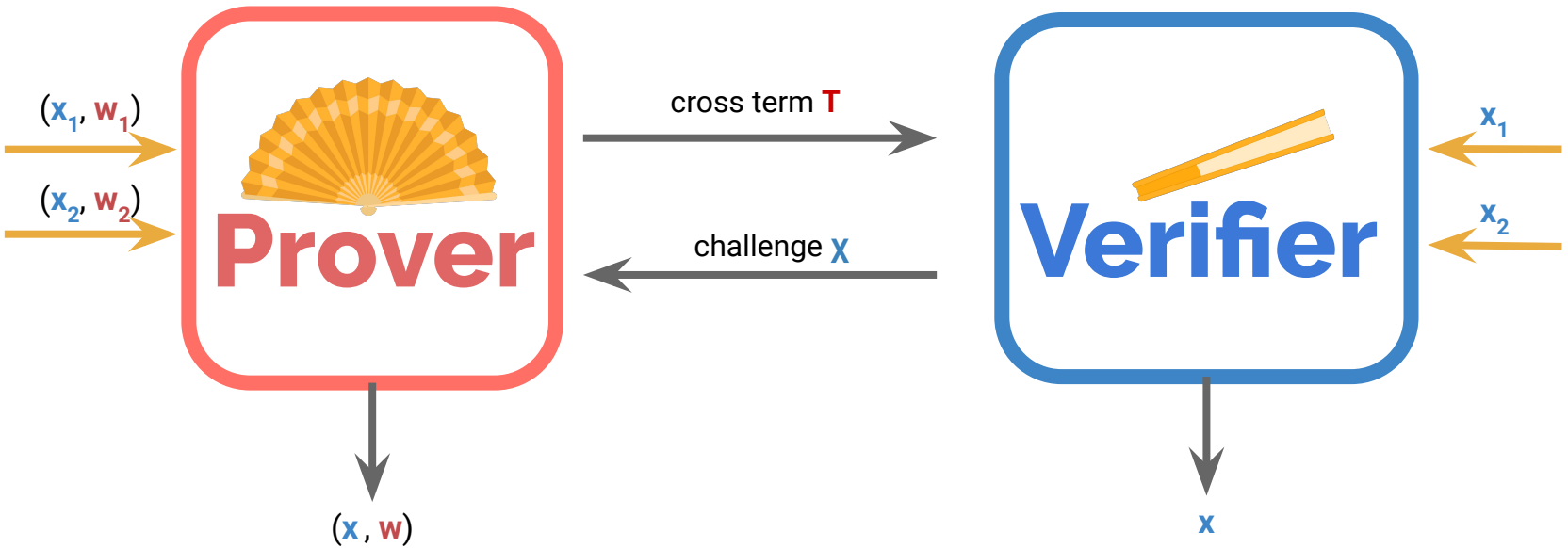
Folding Schemes

Folding



Multiple instances to prove

Folding idea



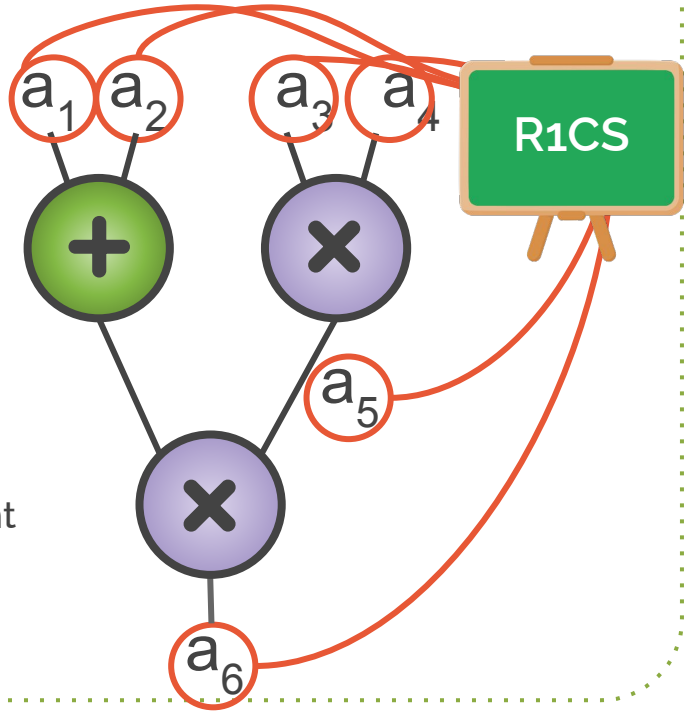
R1CS Rank-1 Constraint System

What are they?

- characterization of NP: it can represent the verification algorithm of arbitrary NP languages
- widely used for SNARKs
- capture arithmetic programs

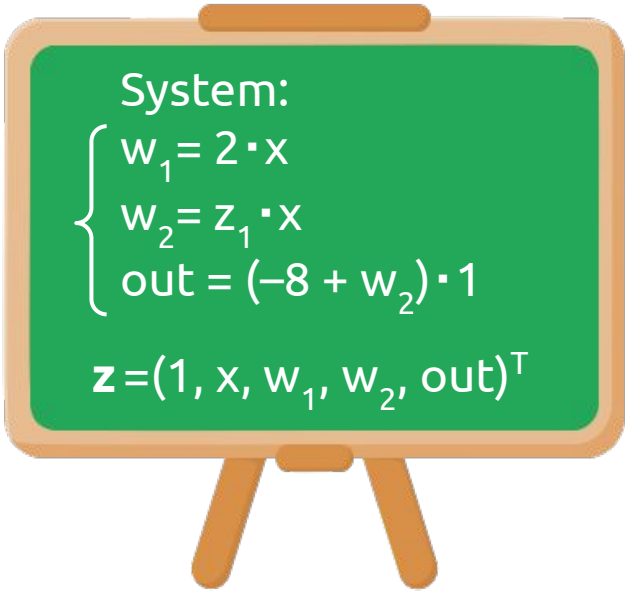
How to define R1CS?

- $\mathbf{A} \cdot \mathbf{z} \circ \mathbf{B} \cdot \mathbf{z} = \mathbf{C} \cdot \mathbf{z}$ where $\mathbf{z} = (\mathbf{x}, \mathbf{w})$
- any arithmetic circuit has a R1CS: each constraint corresponds to one logic gate



R1CS Instances

Example: Solution for equation $2x^2 - 8 = 0$



$\mathbf{z} = (1, x, w_1, w_2, out)^T$

$(2, 0, 0, 0, 0) \cdot \mathbf{z} \circ (0, 1, 0, 0, 0) \cdot \mathbf{z} = (0, 0, 1, 0, 0) \cdot \mathbf{z}$

$(0, 0, 1, 0, 0) \cdot \mathbf{z} \circ (0, 1, 0, 0, 0) \cdot \mathbf{z} = (0, 0, 0, 1, 0) \cdot \mathbf{z}$

$(-8, 0, 0, 1, 0) \cdot \mathbf{z} \circ (1, 0, 0, 0, 0) \cdot \mathbf{z} = (0, 0, 0, 0, 1) \cdot \mathbf{z}$

$$\left(A \cdot z\right) \circ \left(B \cdot z\right) = C \cdot z$$

Multiple Instances

$$\left(A \cdot z_1\right) \circ \left(B \cdot z_1\right) = C \cdot z_1$$

$$\left(A \cdot z_2\right) \circ \left(B \cdot z_2\right) = C \cdot z_2$$

Multiple Instances

$$\left(A \cdot z_1\right) \circ \left(B \cdot z_1\right) = C \cdot z_1$$

$$r \cdot \left(A \cdot z_2\right) \circ \left(B \cdot z_2\right) = C \cdot z_2$$

$$R1CS_1 + r \cdot R1CS_2$$

not R1CS

Multiple Instances

$$\left[A \cdot z_1\right] \circ \left[B \cdot z_1\right] = C \cdot z_1$$

$$r \cdot \left[A \cdot z_2\right] \circ \left[B \cdot z_2\right] = C \cdot z_2$$

$R1CS_1 + r \cdot R1CS_2$

not R1CS

$$\begin{aligned} AZ \circ BZ &= A(Z_1 + r \cdot Z_2) \circ B(Z_1 + r \cdot Z_2) \\ &= AZ_1 \circ BZ_1 + r \cdot (AZ_1 \circ BZ_2 + AZ_2 \circ BZ_1) + r^2 \cdot (AZ_2 \circ BZ_2) \\ &\neq CZ. \end{aligned}$$

Relaxed R1CS

Nova: Recursive Zero-Knowledge Arguments from Folding Schemes – Kothapalli, Setty,Tzialla

R1CS

$$\mathbf{z} = (1, \mathbf{x}, \mathbf{w}) \text{ s.t.}$$

$$(\mathbf{A} \cdot \mathbf{z}) \circ (\mathbf{B} \cdot \mathbf{z}) = \mathbf{C} \cdot \mathbf{z}$$

Relaxed R1CS

Nova: Recursive Zero-Knowledge Arguments from Folding Schemes – Kothapalli, Setty,Tzialla

R1CS

$\mathbf{z} = (1, \mathbf{x}, \mathbf{w})$ s.t.

$$(A \cdot z) \circ (B \cdot z) = C \cdot z$$

relaxed R1CS

instance = $(u, \mathbf{x}, \mathbf{e})$

$\mathbf{z'} = (u, \mathbf{x}, \mathbf{w})$ s.t.

$$(A \cdot z') \circ (B \cdot z') = uC \cdot z' + e$$

Relaxed R1CS

Nova: Recursive Zero-Knowledge Arguments from Folding Schemes – Kothapalli, Setty,Tzialla

R1CS

$$\mathbf{z} = (1, \mathbf{x}, \mathbf{w}) \text{ s.t.}$$

$$(\mathbf{A} \cdot \mathbf{z}) \circ (\mathbf{B} \cdot \mathbf{z}) = \mathbf{C} \cdot \mathbf{z}$$

relaxed R1CS

$$\text{instance} = (u, \mathbf{x}, \mathbf{e})$$

$$\mathbf{z'} = (u, \mathbf{x}, \mathbf{w}) \text{ s.t.}$$

$$(\mathbf{A} \cdot \mathbf{z'}) \circ (\mathbf{B} \cdot \mathbf{z'}) = u\mathbf{C} \cdot \mathbf{z'} + \mathbf{e}$$

$u = 1$

$\mathbf{e} = \mathbf{0}$

Relaxed R1CS

$z_1 = (u_1, x_1, w_1)$

$$\left(A \cdot z_1\right) \circ \left(B \cdot z_1\right) = u_1 C \cdot z_1 + e_1$$

$z_2 = (u_2, x_2, w_2)$

$$\left(A \cdot z_2\right) \circ \left(B \cdot z_2\right) = u_2 C \cdot z_2 + e_2$$

Relaxed R1CS

$$z_1 = (u_1, x_1, w_1)$$

$$\left(A \cdot z_1\right) \circ \left(B \cdot z_1\right) = u_1 C \cdot z_1 + e_1$$

$$r \cdot \left(A \cdot z_2\right) \circ \left(B \cdot z_2\right) = u_2 C \cdot z_2 + e_2$$

$$z_2 = (u_2, x_2, w_2)$$

relaxed R1CS

Relaxed R1CS

$z_1 = (u_1, x_1, w_1)$  $z_2 = (u_2, x_2, w_2)$

$$\left[A \cdot \underbrace{(z_1 + r z_2)}_{z}\right] \circ \left[B \cdot (z_1 + r z_2)\right] =$$

$$= (u_1 + r u_2) C \cdot (z_1 + r z_2) + e$$

$$\begin{aligned} AZ \circ BZ &= AZ_1 \circ BZ_1 + r \cdot (AZ_1 \circ BZ_2 + AZ_2 \circ BZ_1) + r^2 \cdot (AZ_2 \circ BZ_2) \\ &= (u_1 CZ_1 + E_1) + r \cdot (AZ_1 \circ BZ_2 + AZ_2 \circ BZ_1) + r^2 \cdot (u_2 CZ_2 + E_2) \\ &= (u_1 + r \cdot u_2) \cdot C(Z_1 + rZ_2) + E \\ &= uCZ + E. \end{aligned}$$

Committed Relaxed R1CS

Nova: Recursive Zero-Knowledge Arguments from Folding Schemes – Kothapalli, Setty,Tzialla

instance = (u, **x**, [**e**], [**w**])
witness = (**e, w**)

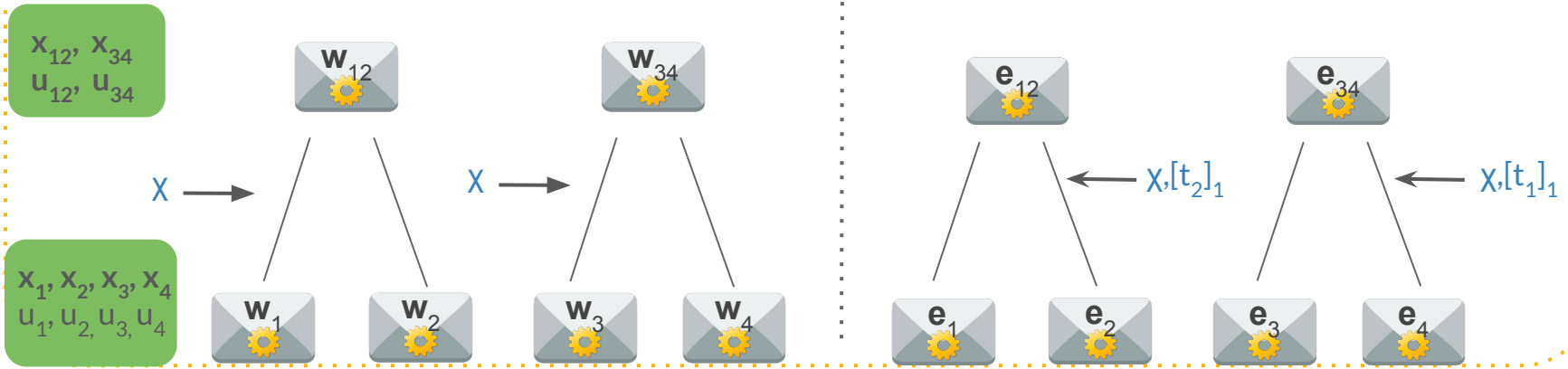
[**e**] = $\text{Com}_{ck}(\mathbf{e})$
s.t. [**w**] = $\text{Com}_{ck}(\mathbf{w})$

and for **z** = (u, **x, e**, **w**)

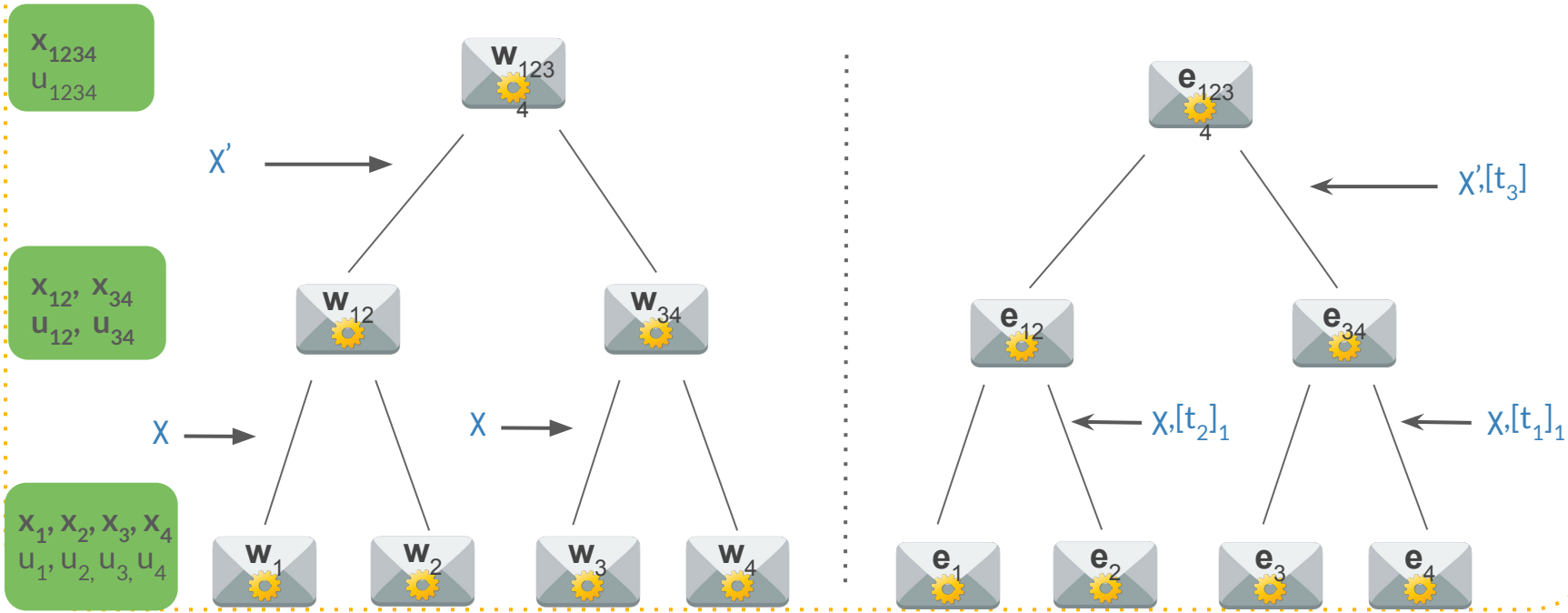
$$\left(\mathbf{A}\cdot\mathbf{z}\right)\circ\left(\mathbf{B}\cdot\mathbf{z}\right) = u\mathbf{C}\cdot\mathbf{z} + \mathbf{e}$$

Nova Folding

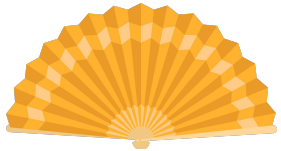
Nova Folding



Nova Folding



Prover



$x_i = (\mathbf{x}_i, u_i, [e_i]_1, [w_i]_1),\ w_i = (\mathbf{w}_i, \mathbf{e}_i)$

Verifier

$x_i = (\mathbf{x}_i, u_i, [e_i]_1, [w_i]_1)$

$\mathbf{z}_i = (u_i, \mathbf{x}_i^\top, \mathbf{w}_i^\top)^\top$

$\mathbf{t} = \mathbf{A}\mathbf{z}_1 \circ \mathbf{B}\mathbf{z}_2 + \mathbf{A}\mathbf{z}_2 \circ \mathbf{B}\mathbf{z}_1 - u_1\mathbf{C}\mathbf{z}_2 - u_2\mathbf{C}\mathbf{z}_1$

$[t]_1 = \mathsf{Com}_{\mathsf{ck}}(\mathsf{pp}, \mathbf{t})$ $\xrightarrow{\quad [t]_1 \quad}$

$\xleftarrow{\quad \chi \quad}$ $\chi \leftarrow \mathbb{F}$

$\mathbf{e} = \mathbf{e}_1 + \chi\mathbf{t} + \chi^2\mathbf{e}_2$

$\mathbf{w} = \mathbf{w}_1 + \chi\mathbf{w}_2$

$[e]_1 = [e_1]_1 + \chi[t] + \chi^2[e_2]_1$

$[w]_1 = [w_1]_1 + \chi[w_2]_1$

$u = u_1 + \chi u_2$

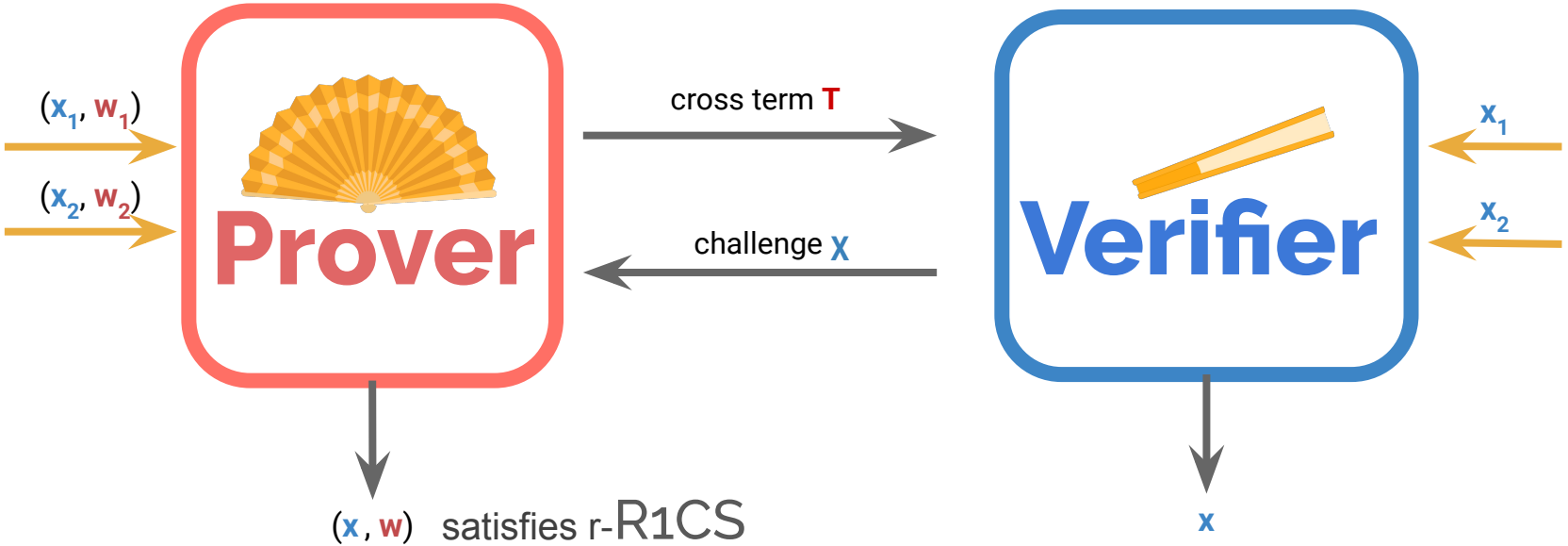
$\mathbf{x} = \mathbf{x}_1 + \chi\mathbf{x}_2$

$w = (\mathbf{w}, \mathbf{e})$

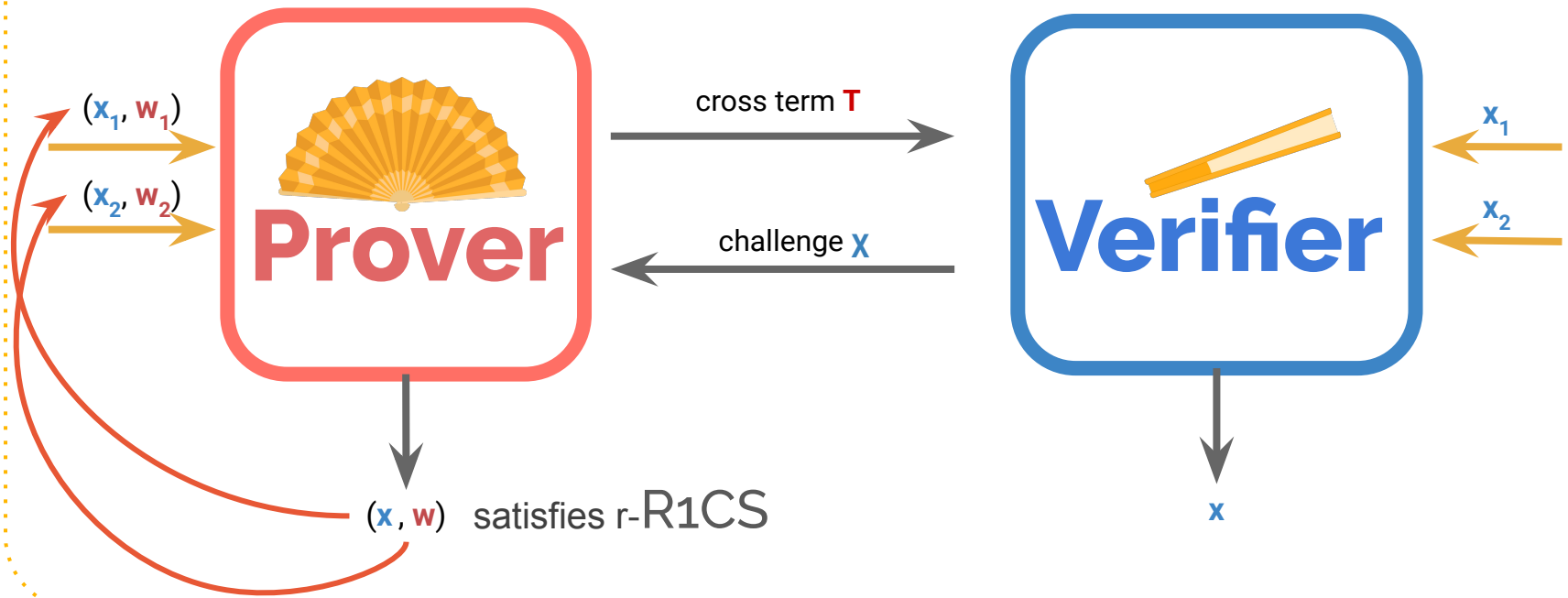
$x = (u, \mathbf{x}, [e]_1, [w]_1)$

Security of Folding

Folding Soundness



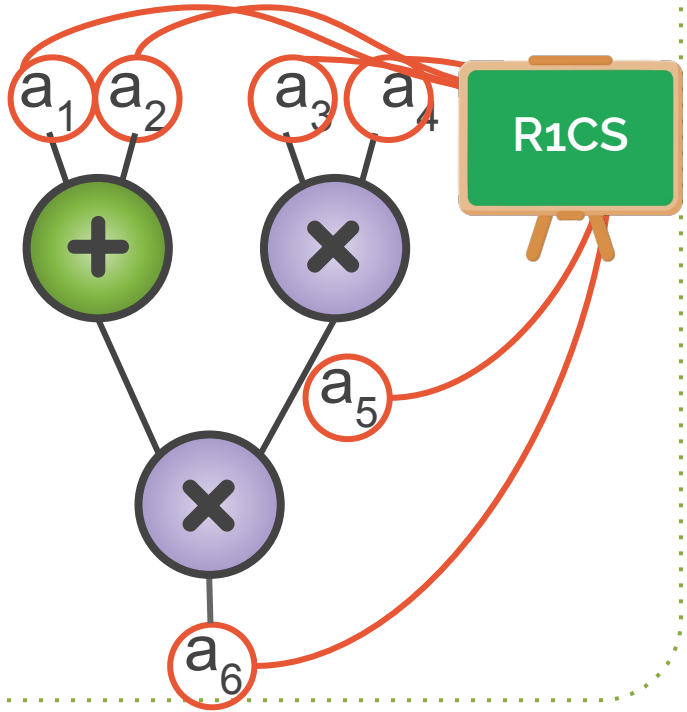
Folding Soundness



CR-R1CS Proof System

Spartan for CR-R1CS

- transparent setup
- sumcheck → very efficient prover
- logarithmic proof size
- adaptation to commit-and-prove



Other Arithmetization

R1CS

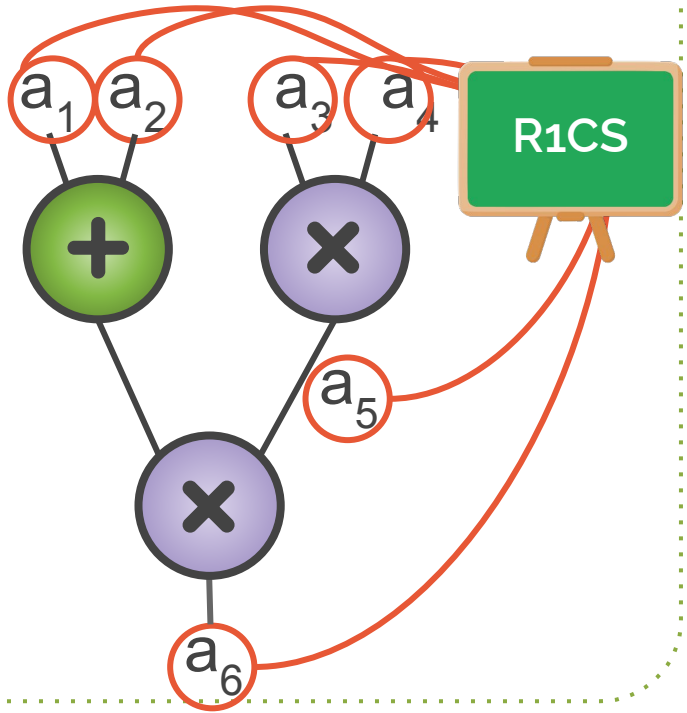
- only arithmetic gates
- one constraint per multiplication gate
- addition gates come for free

Plonkish

- arithmetic circuits + custom gates
- constraints for addition and multiplication
- permutation constraints (copy constraints)
- lookup tables

CCS

- generalization of the two



Nova Generalizations

Sangria – Nova for Plonk arithmetization

- verifier's work is constant in the depth of the circuit
- constants are worse than in Nova

SuperNova – generalized Nova

- different functions at every step
- the cost of proving a step is proportional only to the circuit size for that step

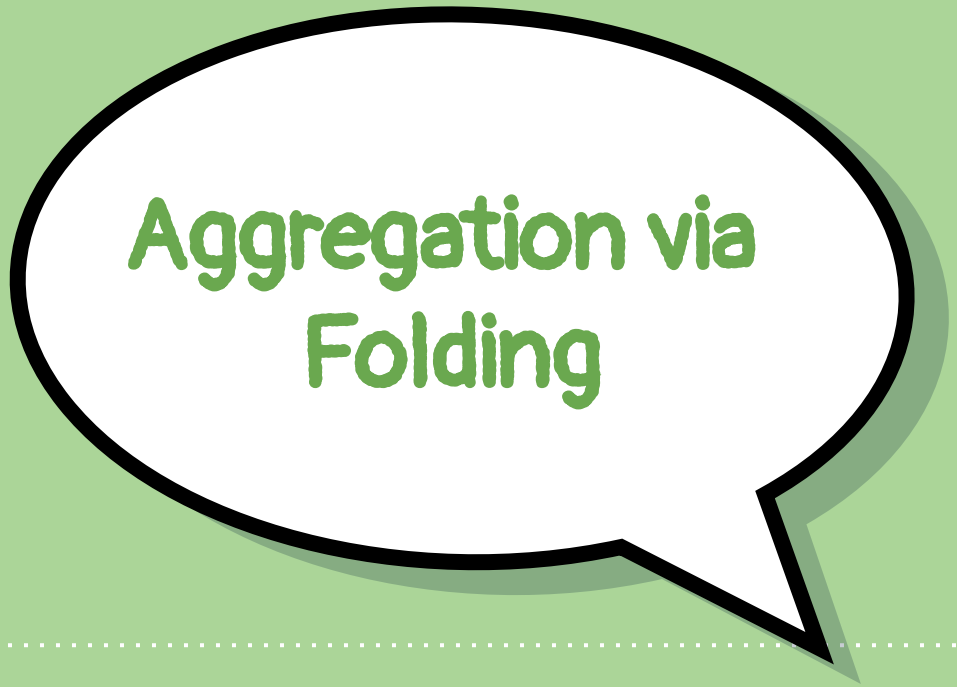
HyperNova – folding with sum-checks

- customizable constraint systems (CCS)
- supports lookups
- sum-check protocol for high-degree constraints polynomial

ProtoStar – Improves HyperNova

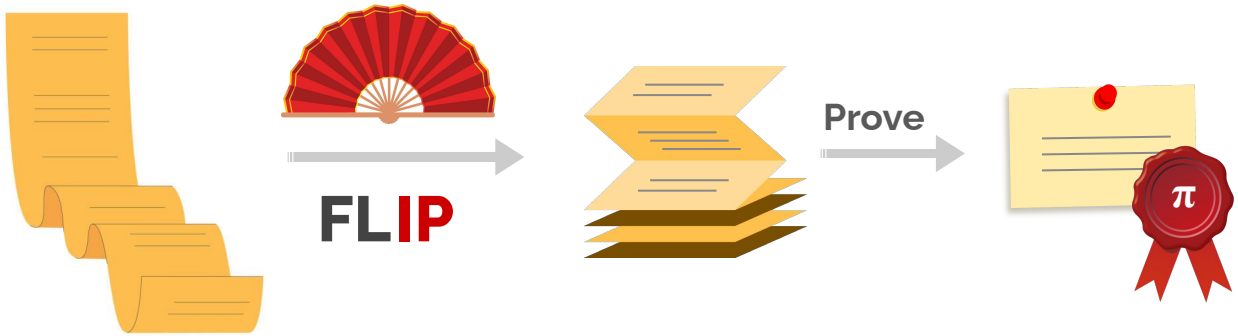
- supports lookups with smaller overhead
- sum-check protocol is not required.

FLIP & Prove

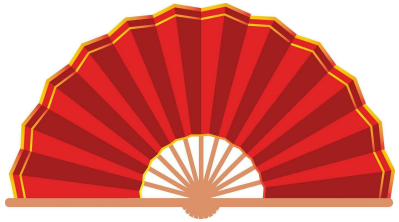


FLIP & Prove

- Size is **logarithmic** as well as verification time
- Reduced Prover time: **no FFTs, no cycles of curves**



FLIP-and-Prove R1CS Anca Nitulescu, Nikitas Paslis, Carla Ràfols



FLIP Construction

Background

Bilinear Groups

$$[a]_1 \in \mathbf{G}_1, [b]_2 \in \mathbf{G}_2$$
$$e : \mathbf{G}_1 \times \mathbf{G}_2 \rightarrow \mathbf{G}_T$$
$$e([a]_1, [b]_2) = [ab]_T$$

Vector Commitment

KeyGen(λ, n) $\rightarrow$ ck: $[1]_1, [\tau]_1, [\tau^2]_1, \ldots [\tau^n]_1$

Commit(ck, **a**) $\rightarrow$ 

$= [A(\tau)]_1$

$A(X) = a_1 + a_2X + a_3X^2 + \ldots a_nX^{n-1}$

Vector Commitment

KeyGen(λ, n) $\rightarrow$ ck: $[1]_1, [\tau]_1, [\tau^2]_1, \ldots [\tau^n]_1$

Commit(ck, **a**) $\rightarrow$ 

$= [A(\tau)]_1$

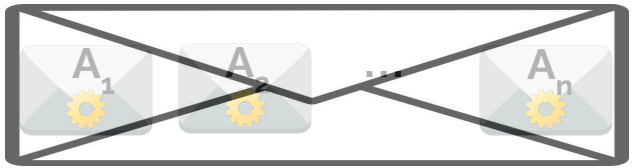
$A(X) = a_1 + a_2X + a_3X^2 + \ldots a_nX^{n-1}$

Target-Group Commitment

Vector Commitment

KeyGen(λ, n) $\rightarrow$ ck: $[1]_2, [\tau]_2, [\tau^2]_2, \ldots [\tau^n]_2$

Target-Group Commitment



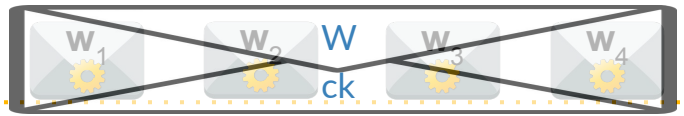
$$C = e([A_1]_1,[\tau]_2)e([A_2]_1,[\tau^2]_2)\ldots e([A_n],[\tau^n]_2)$$

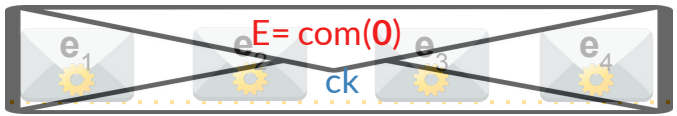
FLIP-style Folding

x_1, x_2, x_3, x_4
u_1, u_2, u_3, u_4

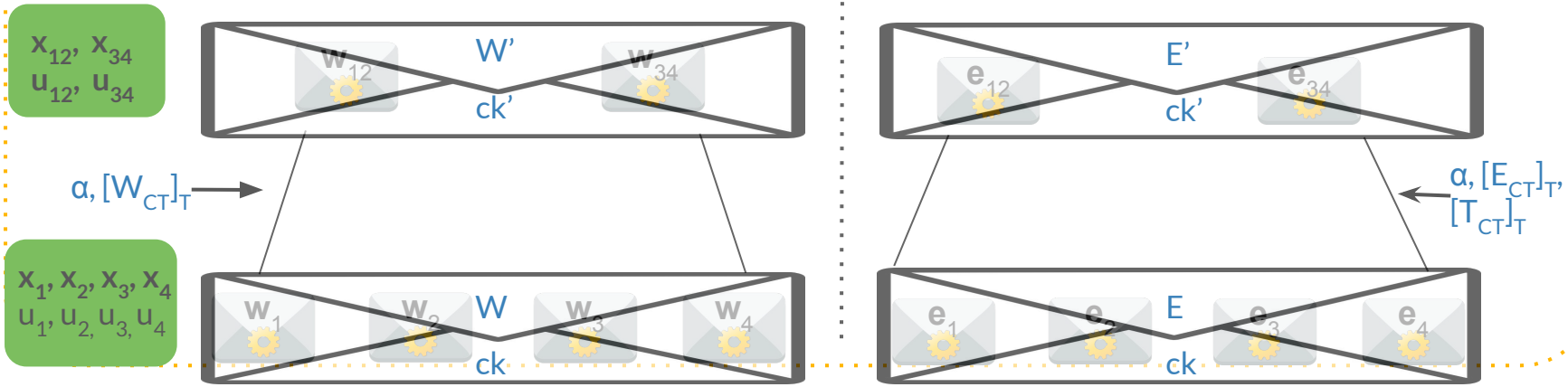
FLIP-style Folding

x_1, x_2, x_3, x_4
u_1, u_2, u_3, u_4

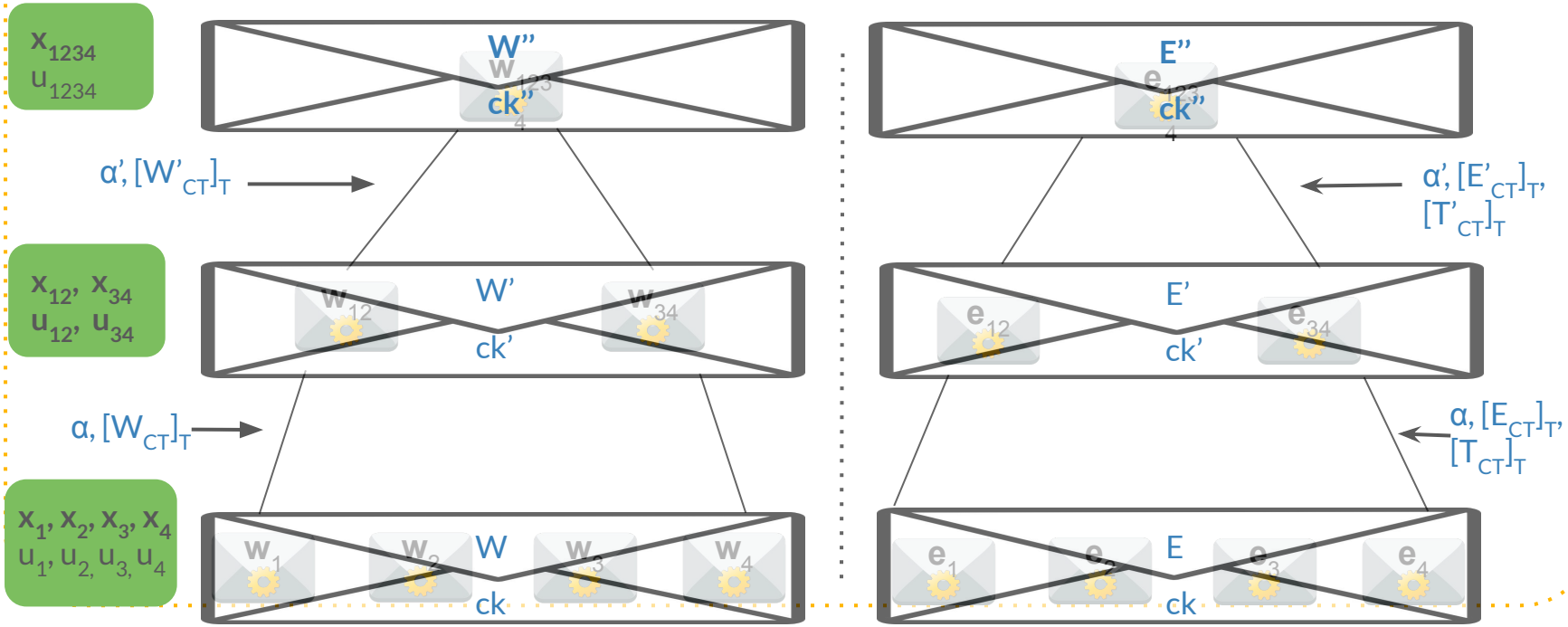




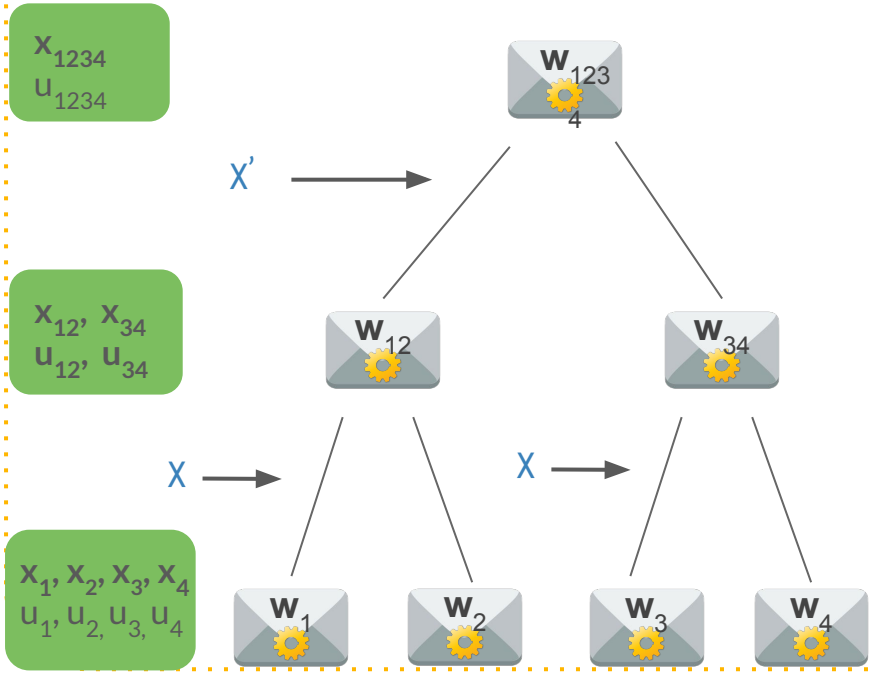
FLIP-style Folding



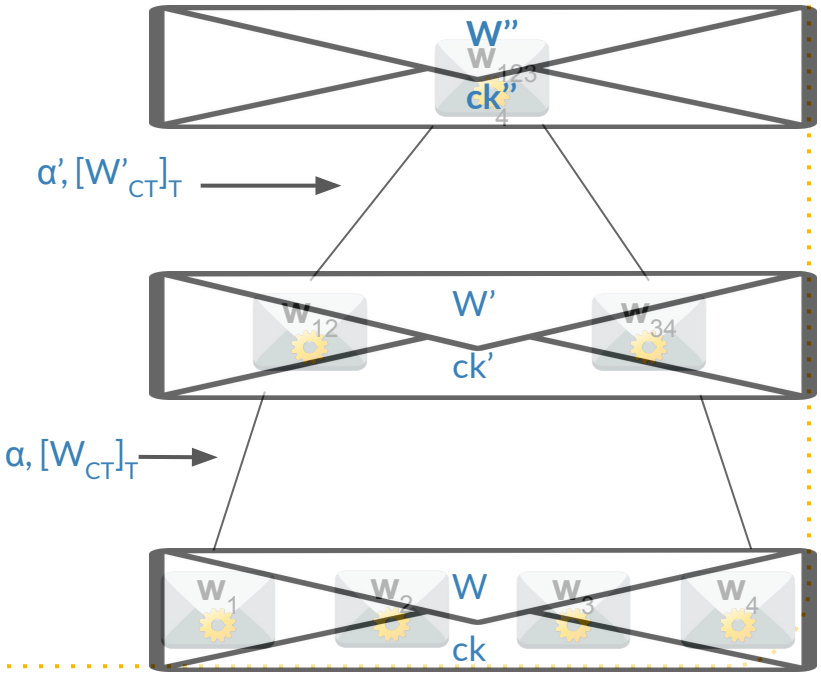
FLIP-style Folding



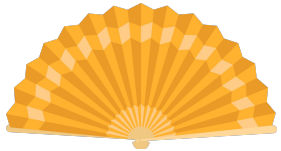
Nova-style



FLIP -style



Prover



$x_i = (\mathbf{x}_i, u_i, [e_i]_1, [w_i]_1),\ w_i = (\mathbf{w}_i, \mathbf{e}_i)$

$\mathbf{z}_i = (u_i, \mathbf{x}_i^\top, \mathbf{w}_i^\top)^\top$

$\mathbf{t}_i = \mathbf{A}\mathbf{z}_i \circ \mathbf{B}\mathbf{z}_{i+\nu/2} + \mathbf{A}\mathbf{z}_{i+\nu/2} \circ \mathbf{B}\mathbf{z}_i$
$\quad - u_i \mathbf{C}\mathbf{z}_{i+\nu/2} - u_{i+\nu/2}\mathbf{C}\mathbf{z}_i$

Compute:

$[t_i]_1 = \widetilde{\mathsf{Com}}(\widetilde{\mathsf{ck}}, \mathbf{t}_i; \mathbf{r}_{t_i}), \mathbf{r}_{t_i} \leftarrow \mathbb{F}^{\tilde{R}};$

$$[T_L]_T = \sum_{i=0}^{\nu/2-1} e([t_i]_1, [y_i]_2); \quad [T_R]_T = \sum_{i=0}^{\nu/2-1} e([t_{\nu/2+i}]_1, [y_{\nu/2+i}]_2)$$

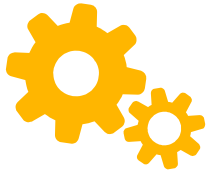
$$[E_{LR}]_T = \sum_{i=0}^{\nu/2-1} e([e_i]_1, [y_{\nu/2+i}]_2); \quad [E_{RL}]_T = \sum_{i=0}^{\nu/2-1} e([e_{\nu/2+i}]_1, [y_i]_2)$$

$$[W_{LR}]_T = \sum_{i=0}^{\nu/2-1} e([w_i]_1, [q_{\nu/2+i}]_2); \quad [W_{RL}]_T = \sum_{i=0}^{\nu/2-1} e([w_{\nu/2+i}]_1, [q_i]_2)$$

$\mathcal{C} = \{[T_L, T_R, E_{LR}, E_{RL}, W_{LR}, W_{RL}]_T\}$

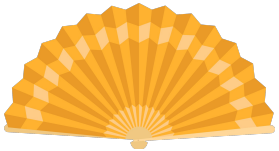
$\mathcal{C}$ ⟶

Verifier



$x_i = (\mathbf{x}_i, u_i, [e_i]_1, [w_i]_1)$

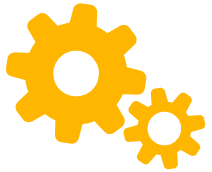
Prover



$x_i = (\mathbf{x}_i, u_i, [e_i]_1, [w_i]_1),\ w_i = (\mathbf{w}_i, \mathbf{e}_i)$

Verifier

$x_i = (\mathbf{x}_i, u_i, [e_i]_1, [w_i]_1)$



$\alpha \leftarrow \mathbb{F}$

Compute:

$$[E']_T = [E]_T + \alpha^{-2}[E_{LR}]_T + \alpha[T_L]_T + \alpha^{-1}[T_R]_T + \alpha^2[E_{RL}]$$

$$[W']_T = [W]_T + \alpha^{-1}[W_{LR}]_T + \alpha[W_{RL}]_T$$

$$\mathbf{x}'_i = \mathbf{x}_i + \alpha\mathbf{x}_{i+\nu/2},\ u'_i = u_i + \alpha u_{i+\nu/2}$$

$$\mathbf{w}'_i = \mathbf{w}_i + \alpha\mathbf{w}_{i+\nu/2}, \mathbf{r}_{w'_i} = \mathbf{r}_{w_i} + \alpha\mathbf{r}_{w_{i+\nu/2}}$$

$$\mathbf{e}'_i = \mathbf{e}_i + \alpha\mathbf{t}_i + \alpha^2\mathbf{e}_{i+\nu/2}, \mathbf{r}_{e'_i} = \mathbf{r}_{e_i} + \alpha\mathbf{r}_{t_i} + \alpha^2\mathbf{r}_{e_{i+\nu/2}}$$

$$[e'_i]_1 = [e_i]_1 + \alpha[t'_i]_1 + \alpha^2[e_{i+\nu/2}]_1$$

$$[w'_i]_1 = [w_i]_1 + \alpha[w_{i+\nu/2}]_1$$

$$[y'_i]_2 = [y_i]_2 + \alpha^{-2}[y_{i+\nu/2-1}]_2$$

$$[q'_i]_2 = [q_i]_2 + \alpha^{-1}[q_{i+\nu/2-1}]_2$$

Set:

$$\Phi' = ([W']_T, [E']_T, \{(u'_i, \mathbf{x}'_i)\}_{i=0}^{\nu/2-1})$$

$$w_{\Phi'} = \left\{(([e'_i]_1, [w'_i]_1); (\boldsymbol{w}'_i, \boldsymbol{e}'_i, \mathbf{r}_{w'_i}, \mathbf{r}_{e'_i}))\right\}$$

$$\mathsf{pk}_{\nu/2} = (\widetilde{\mathsf{ck}}, [\mathbf{y}']_2 \in \mathbb{G}_2^{\nu/2}, [\mathbf{q}']_2 \in \mathbb{G}_2^{\nu/2})$$

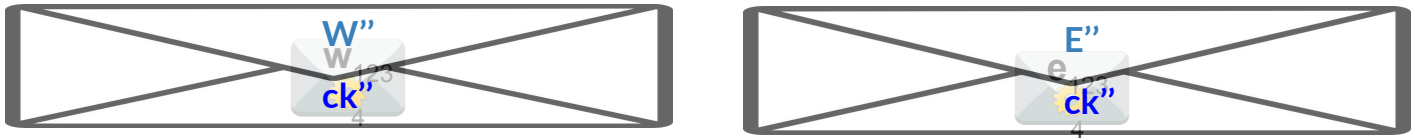
Compute:

Set:

$$\Phi' = ([W']_T, [E']_T$$

$$\{(u'_i, \mathbf{x}'_i)\}_{i=0}^{\nu/2-1}$$

Linear Verifier



$\mathbf{W} = (W_1, W_2, \ldots W_n)$

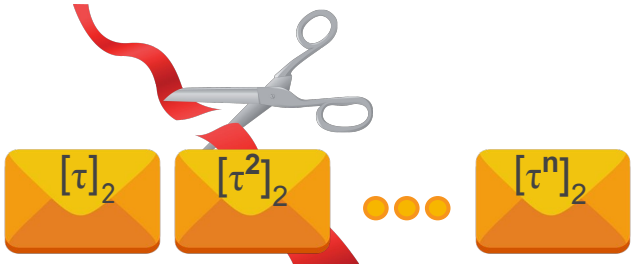
$\mathbf{W'} = \mathbf{W}_{left}\mathbf{W}_{right}$

W_{final}

ck_1 ck_2 ck_n → ck'_1 $ck'_{n/2}$ → ck_{final}

Linear verification

Solution: Structured Setup



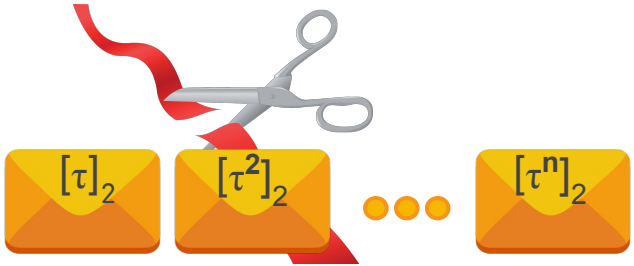
Logarithmic verification for folding commitment key **ck**

$\mathbf{W} = (W_1, W_2, \ldots W_n)$

$\mathbf{W'} = \mathbf{W}_{left}\mathbf{W}_{right}$

W_{final}

Solution: Structured Setup



Logarithmic verification for folding commitment key **ck**

W =(W_1, W_2, … W_n)

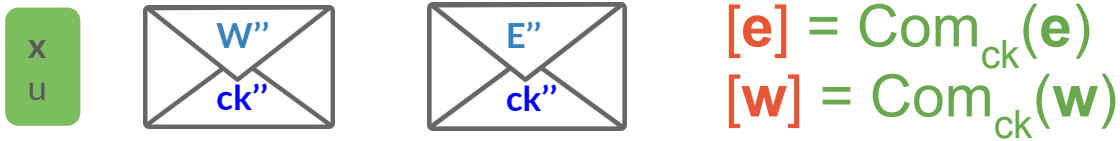
W'=$\mathbf{W}_{left}\mathbf{W}_{right}$

W_{final}

Fast polynomial check on ck_{final}: $g_\alpha(X) = \prod_{i=1}^{\mu}\left(1+\alpha_{\mu+1-i}^{-2}X^{2^{i-1}}\right)$

Proving Final Instance

Proving FLIP



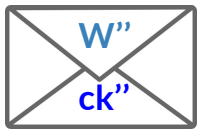
$[\mathbf{e}] = \mathrm{Com}_{ck}(\mathbf{e})$
$[\mathbf{w}] = \mathrm{Com}_{ck}(\mathbf{w})$

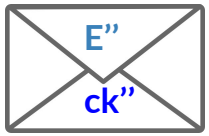
Committed Relaxed R1CS

for $\mathbf{z} = (u, \mathbf{x}, \mathbf{e}, \mathbf{w})$

$$\left(\mathbf{A}\cdot\mathbf{z}\right) \circ \left(\mathbf{B}\cdot\mathbf{z}\right) = u\mathbf{C}\cdot\mathbf{z} + \mathbf{e}$$

Proving FLIP



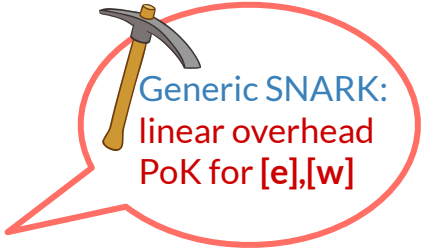


$[\mathbf{e}] = \mathrm{Com}_{ck}(\mathbf{e})$
$[\mathbf{w}] = \mathrm{Com}_{ck}(\mathbf{w})$

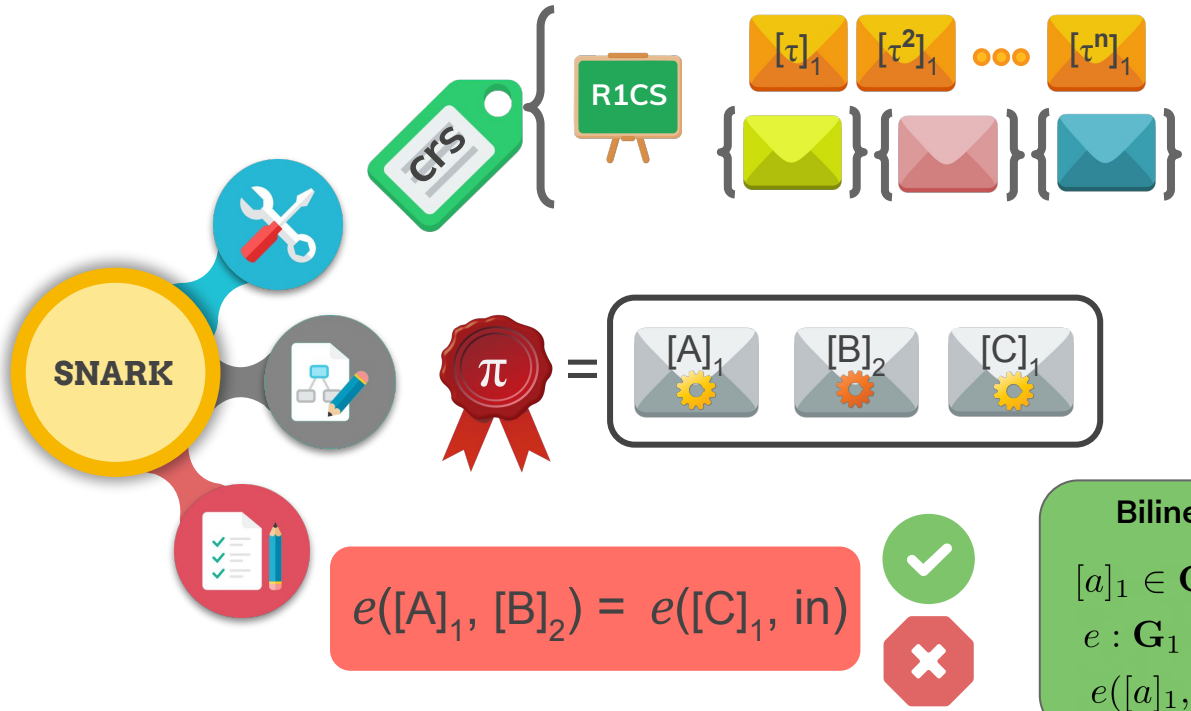
Committed Relaxed R1CS

for $\mathbf{z} = (u, \mathbf{x}, \mathbf{e}, \mathbf{w})$

$$(A \cdot z) \circ (B \cdot z) = uC \cdot z + e$$



Groth 16



Bilinear Groups

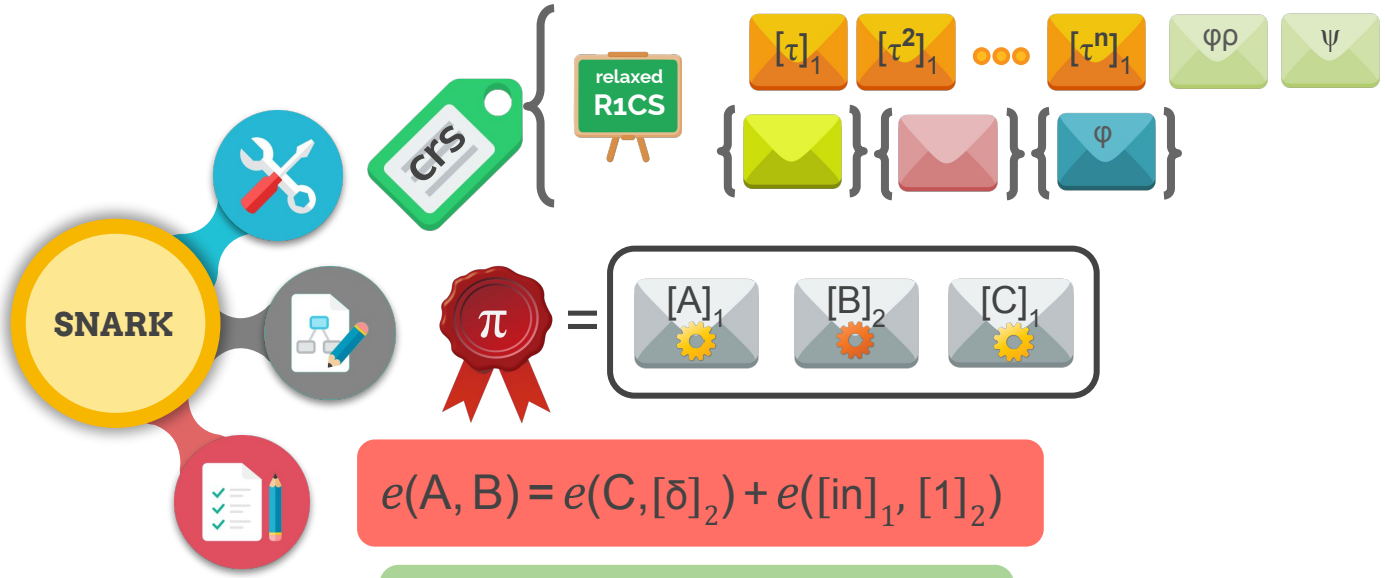
$[a]_1 \in \mathbf{G}_1, [b]_2 \in \mathbf{G}_2$

$e : \mathbf{G}_1 \times \mathbf{G}_2 \rightarrow \mathbf{G}_T$

$e([a]_1, [b]_2) = [ab]_T$

Relaxed Groth 16

for Committed Relaxed R1CS



$$e(A, B) = e(C, [\delta]_2) + e([in]_1, [1]_2)$$

$$+ e(u^{-1}[e]_1, [\psi]_2) - e([w]_1, [\varphi\rho]_2)$$

Groth 16

Groth.Setup$(1^\lambda, \mathcal{R})$

$\alpha, \beta, \gamma, \delta \leftarrow\$ \mathbb{Z}_p^*, \quad s \leftarrow\$ \mathbb{Z}_p^*,$

$$\mathsf{crs} = \left(\mathsf{QAP}, [\alpha, \beta, \delta]_{1,2}, \{[s^i]_{1,2}\}_{i=0}^{d-1}, \left\{\left[\frac{\beta v_j(s) + \alpha w_j(s) + y_j(s)}{\gamma}\right]_1\right\}_{j=0}^{t}, \left\{\left[\frac{s^i t(s)}{\delta}\right]_1\right\}_{i=0}^{d-2}, \left\{\left[\frac{\beta v_j(s) + \alpha w_j(s) + y_j(s)}{\delta}\right]_1\right\}_{j>t}, \right)$$

Groth.Prove(crs, u, w)

$u = (a_1, \ldots, a_t), \ a_0 = 1$

$w = (a_{t+1}, \ldots, a_m)$

$v(x) = \sum_{j=0}^{m} a_j v_j(x)$

$v_{\mathsf{mid}}(x) = \sum_{j>t} a_j v_j(x)$

$w(x) = \sum_{j=0}^{m} a_j w_j(x)$

$w_{\mathsf{mid}}(x) = \sum_{j>t} a_j w_j(x)$

$y(x) = \sum_{j=0}^{m} a_j y_j(x)$

$y_{\mathsf{mid}}(x) = \sum_{j>t} a_j y_j(x)$

$h(x) = \frac{(v(x)w(x) - y(x))}{t(x)}$

$$f_{\mathsf{mid}} = \frac{\beta v_{\mathsf{mid}}(s) + \alpha w_{\mathsf{mid}}(s) + y_{\mathsf{mid}}(s)}{\delta}$$

$r, u \leftarrow\$ \mathbb{Z}_p^*$

$A = [\alpha + v(s) + r\delta]_1,$

$B = [\beta + w(s) + u\delta]_2$

$$C = \left[f_{\mathsf{mid}} + \frac{t(s)h(s)}{\delta} + ua + rb - ur\delta\right]_2$$

return $(\pi = (A, B, C))$

Groth.Verify(vk, u, π)

$\pi = (A, B, C)$

$v_{io}(x) = \sum_{i=0}^{t} a_i v_i(x)$

$w_{io}(x) = \sum_{i=0}^{t} a_i w_i(x)$

$y_{io}(x) = \sum_{i=0}^{t} a_i y_i(x)$

$$f_{io} = \frac{\beta v_{io}(s) + \alpha w_{io}(s) + y_{io}(s)}{\gamma}$$

Check

$e(A, B) = e([\alpha]_1, [\beta]_2) \cdot e([f_{io}]_1, [\gamma]_2) \cdot e(C, [\delta]_2)$

Groth.Sim$(\mathsf{td}, u) \qquad \mathsf{td} = (s, \alpha, \beta, \gamma, \delta)$

$a, b \leftarrow\$ \mathbb{Z}_p^*$

$$c = \frac{ab - \alpha\beta - \beta v_{io}(s) + \alpha w_{io}(s) + y_{io}(s)}{\delta}$$

return $(\pi = ([a]_1, [b]_2, [c]_1))$

Relaxed Groth 16

- **Commit-and-Prove Groth16** for Pedersen-like commitments
- **Small proof size :** CaP LegoGroth16 has 2 additional group elements and 3 verification parings
- **Security Proof with auxiliar inputs: Strong** security model that takes into account the setup for FLIP and the intermediate values available from the Trusted Setup Ceremony
- **Can be used with Nova** as last step

$$\mathsf{srs} := \left(\begin{array}{l} \left[\mathsf{ck}, \widetilde{\mathsf{ck}}, \alpha, \beta, \delta, \{x^i\}_{i=0}^{l-1}, \{\sigma_j := u_j(x)\beta + v_j(x)\alpha + w_j(x)\}_{j=0}^{l} \right. \\ \left. \{x^i t(x)/\delta\}_{i=0}^{n-2}, \left\{ \sigma_j := \dfrac{u_j(x)\beta + v_j(x)\alpha + w_j(x) + \varphi \ell_j(x)}{\delta} \right\}_{j=l+1}^{m} \right]_1, \\ [\beta, \delta, [\varphi\rho]_2, [\psi]_2, \{x^i\}_{i=0}^{n-1}]_2, \ [\alpha\beta]_T \end{array} \right)$$

where

$$\mathsf{srs}_\mathsf{V} = \left(\{\sigma_j\}_{j=0}^{l}, [\delta]_2, [\varphi\rho]_2, [\psi]_2, [\alpha\beta]_T \right).$$

$$\mathsf{ck} = ([\ell_{l+1}(x)/\rho]_1, \ldots, [\ell_m(x)/\rho]_1, [\delta/\rho]_1) \in \mathbb{G}_1^{m-l+1},$$

$$\widetilde{\mathsf{ck}} = ([\ell_0(x)/\psi]_1, \ldots, [\ell_{n-1}(x)/\psi]_1, [\delta/\psi]_1) \in \mathbb{G}_1^{n+1},$$

Prove, $\pi \leftarrow \mathsf{Prove}(\mathsf{srs}_\mathsf{P}, \phi = (\mathbf{z}_{[:l]}, u, [e]_1, [w]_1), W = (\mathbf{z}_{[l:]}, \mathbf{e}, r_w, r_e))$**:** assuming $z_0 = 1$, it acts as follows,

1. Let $A^\dagger(x) \leftarrow \sum_{j=0}^{m} z_j u_j(x)$, $B^\dagger(x) \leftarrow \sum_{j=0}^{m} z_j v_j(x)$, $C^\dagger(x) \leftarrow \sum_{j=0}^{m} z_j w_j(x)$,
2. Let $e^\dagger(x) = \sum_{i=0}^{n-1} e_i \ell_i(x)$. Set $h(x) = \sum_{i=0}^{n-2} h_i x^i \leftarrow (u^{-1} A^\dagger(x) B^\dagger(x) - C^\dagger(x) - e^\dagger(x) u^{-1})/t(x)$.
3. Set $[h(x)t(x)/\delta]_1 \leftarrow \sum_{i=0}^{n-2} h_i \left[x^i t(x)/\delta\right]_1$.
4. Set $r_a \leftarrow_r \mathbb{Z}_p$. Set $[A]_1 \leftarrow u^{-1} \sum_{j=0}^{m} z_j \left[u_j(x)\right]_1 + [\alpha]_1 + r_a [\delta]_1$.
5. Set $r_b \leftarrow_r \mathbb{Z}_p$. Set $[B]_2 \leftarrow [u\beta]_2 + \sum_{j=0}^{m} z_j \left[v_j(x)\right]_2 + r_b [\delta]_2$.
6. Set $[C]_1 \leftarrow r_b [A]_1 + r_a [B]_1 - r_a r_b [\delta]_1 + r_w [\varphi]_1 - u^{-1} r_e [1]_1 + \sum_{j=l+1}^{m} z_j [\sigma_j]_1 + [h(x)t(x)/\delta]_1$,
7. Return $\pi := ([A, C]_1, [B]_2)$.

Verify, $\{1, 0\} \leftarrow \mathsf{Verify}(\mathsf{srs}_\mathsf{V}, \phi, \pi = ([A, C]_1, [B]_2))$**:** assuming $z_0 = 1$, checks:

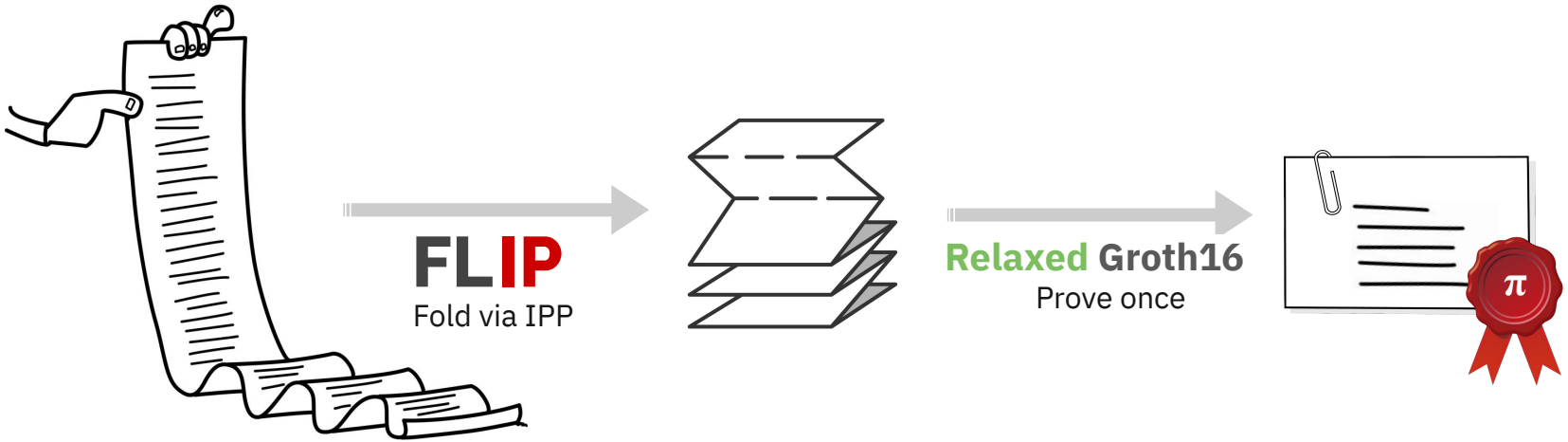
$$e([A]_1, [B]_2) - e([C]_1, [\delta]_2) - e(\sum_{j=0}^{l} z_j [\sigma_j]_1, [1]_2) - e(u^{-1} [e]_1, [\psi]_2) + e([w]_1, [\varphi\rho]_2) = u [\alpha\beta]_T.$$

Comparison

	Approach	Proof Size	FFTs	Cycle of curves	r-R1CS native	ZK
Aggreg	SnarkPack	$O(\log(m))\mathbb{G}_T + O(1)\mathbb{G}_{1,2}$	yes	no	no	no
IVC	Nova	$mO(1) + O(\log(n))\mathbb{G}$	no	yes	yes	no
	Nova with Groth16	$mO(1) + O(1)\mathbb{G}_{1,2}$	no	yes	no	no
	Nova with **Relaxed Groth16**	$mO(1) + O(1)\mathbb{G}_{1,2}$	no	yes	yes	yes
FLIP & Prove	**FLIP with Relaxed Groth16**	$O(\log(m))\mathbb{G}_T + O(1)\mathbb{G}_{1,2}$	no	no	yes	yes

Summary

Multiple instances to prove



Conclusions

FLIP & Prove aggregation:

- massive reduction in prover time compared to **Snark Pack**
  avoids **MSM** and **FFT** for generating **n** independent **SNARK**s
- better than recursion in some scenarios: no need for expensive arithmetization of verifier circuit inside the prover
- no need for (half) **cycles** of elliptic curves as in Nova folding
- **Relaxed Groth16** can be used as an alternative to Spartan for proving the last step of Nova

Open Questions

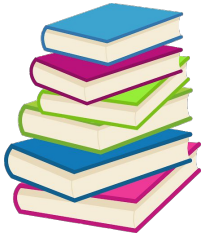
- **Improvements:** Lower memory usage for the Prover in **FLIP**
- **Extensions:** Aggregate instances for different relations
  - Other NP representations: Sangria for *PlonK* arithmetization
  - Instances for different **R1CS** relations: SuperNova

- **Nova-friendly IVC last step proving via Relaxed Groth16:** Nova works on a cycle of curves without pairings, we need a 1-chain to apply Groth16

Thanks

eprint.iacr.org/2024/1364

References



[KZG10] **Constant-Size Commitments to Polynomials and Their Applications** – A. Kate, G. Zaverucha, I. Goldberg, in ASIACRYPT'10.

[GGPRR13] **Quadratic span programs and succinct NIZKs without PCPs** – R. Gennaro, C. Gentry, Bryan Parno, and Mariana Raykova.

[Groth16] **On the Size of Pairing-based Non-interactive Arguments** – Jens Groth, EUROCRYPT 2016.

[BBB+18] **Bulletproofs: Short Proofs for Confidential Transactions and More** – B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwel

[GWC19] **PlonK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge** – Ariel Gabizon, Zachary J. Williamson, Oana Ciobotaru

[GMN20] **SnarkPack: Practical SNARK Aggregation** – Nicolas Gailly, Mary Maller, Anca Nitulescu , FC 2022

[BKM+22] **Caulk: Lookup Arguments in Sublinear Time** – A. Zapico, V. Buterin, D. Khovratovich, M. Maller, A. Nitulescu, M. Simkin, CCS 2022

[KST22] **Nova: Recursive Zero-Knowledge Arguments from Folding Schemes** – Abhiram Kothapalli, Srinath Setty, Ioanna Tzialla, CRYPTO 2022

[NPR24] **FLIP-and-Prove R1CS** – Anca Nitulescu, Nikitas Paslis, Carla Ràfols, eprint.iacr.org/2024/1364

Questions?

Credits

Special thanks to all those who made and released these resources for free:

- Presentation template by SlidesCarnival
- Clip arts by Iconfinder, Flaticon and juicy_fish
- Illustrations by Disneyclips