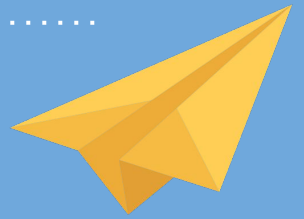
Foundations and Applications of Zero-Knowledge Proofs



Zero-knowledge SNARKs:

How fast can we go?

Anca Nitulescu

INPUT | OUTPUT

Outline

1. Proving Large Computations
2. Recursion Techniques
3. Folding Schemes
4. Faster Schemes?

zk-SNARK

Zero-Knowledge
does not leak anything about the witness

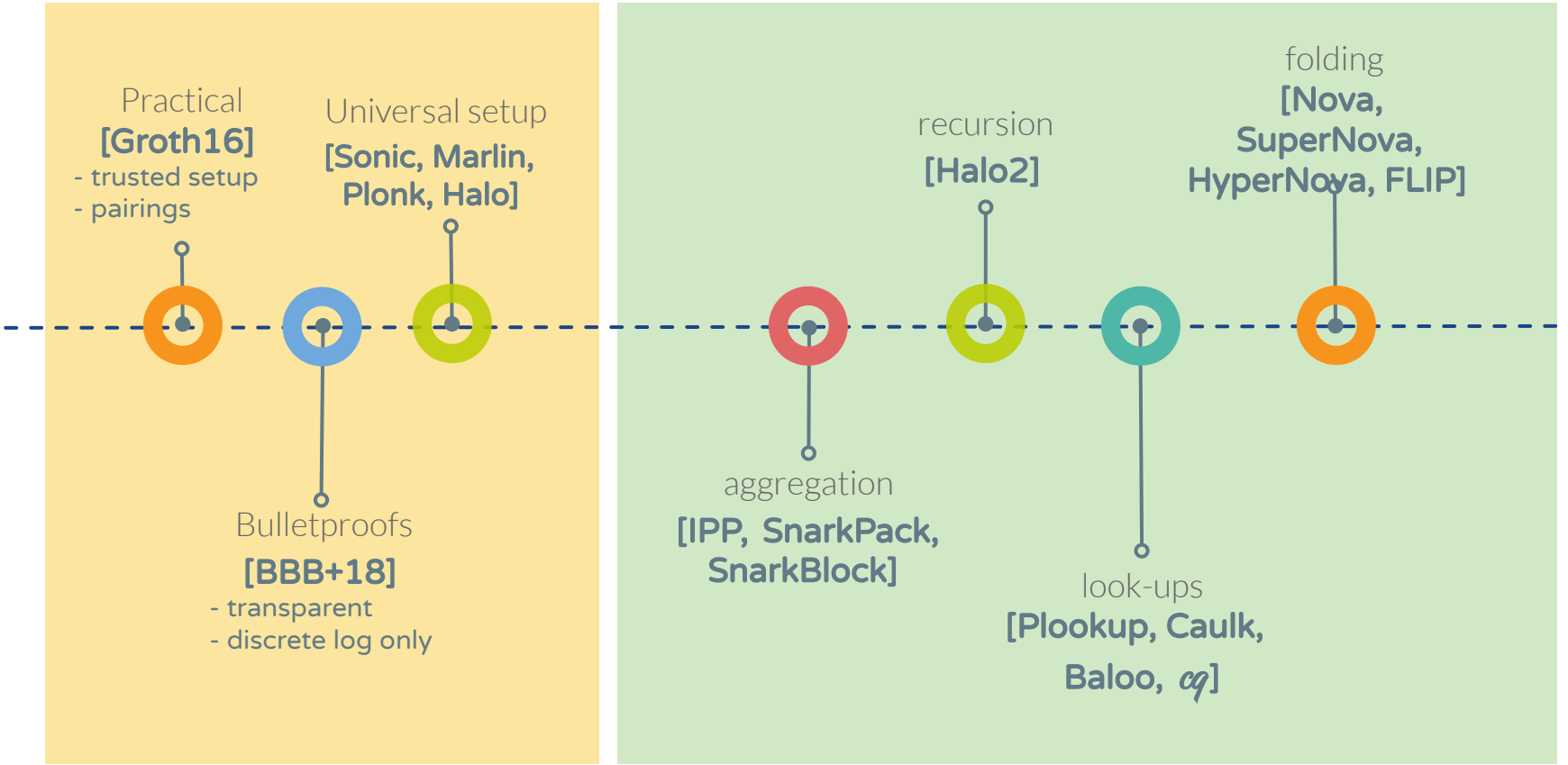
Succinctness
- proof size independent (sublinear) of NP witness size
- fast verification

Non-Interactivity
no exchange between prover and verifier

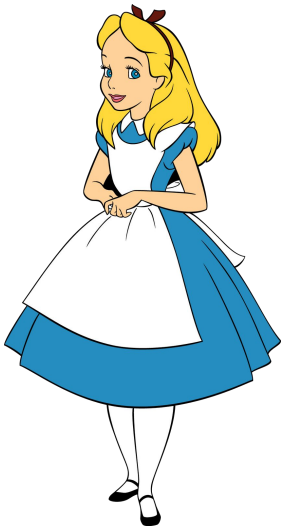
Knowledge Soundness
a witness can be efficiently extracted from the prover

Argument
soundness holds only against computationally bounded provers

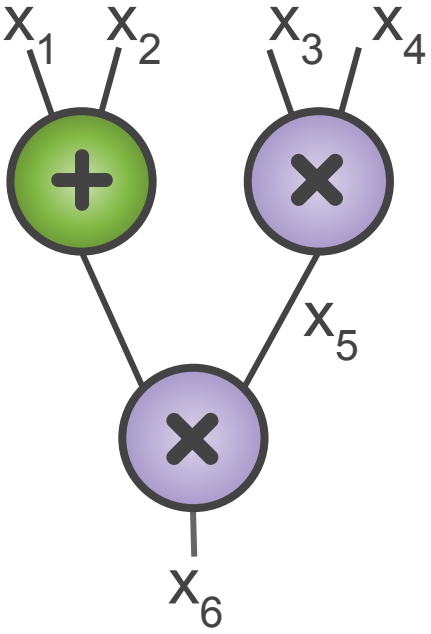
Recent zk-SNARK research



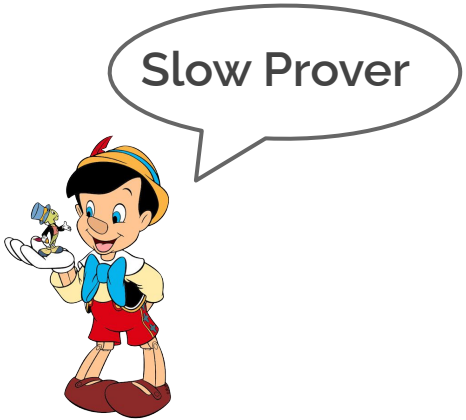
Prove Large Circuits



Verifier



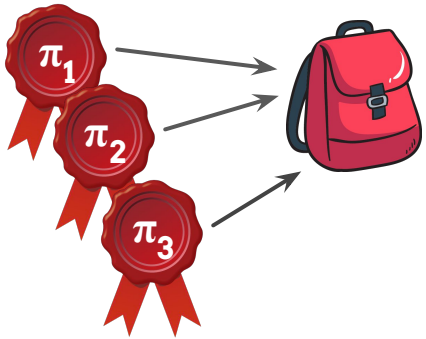
- proof size < witness size
- verification = sublinear(|C|)



Prover

Prove Large Circuits

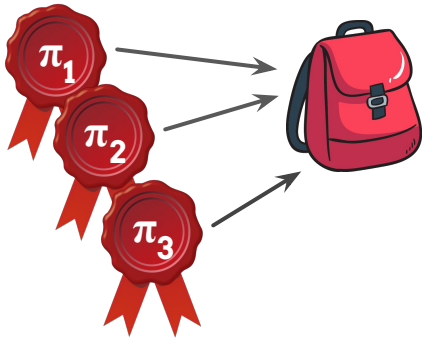
Aggregation



Not fast

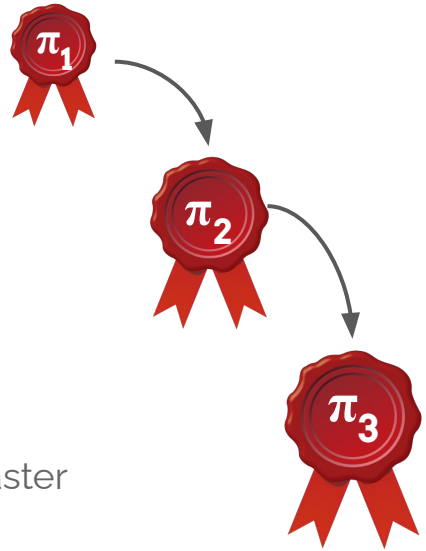
Prove Large Circuits

Aggregation



Not fast

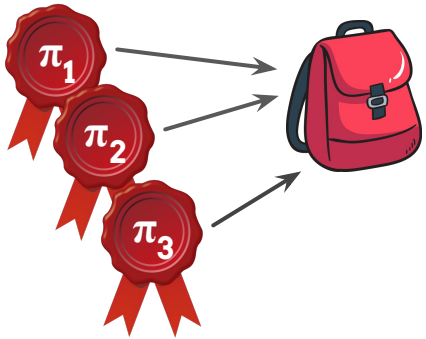
Recursion



Faster

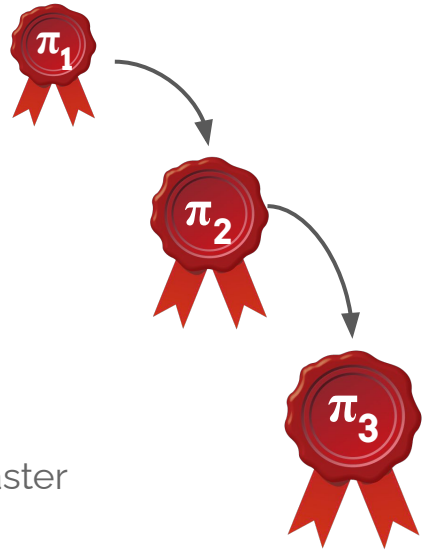
Prove Large Circuits

Aggregation



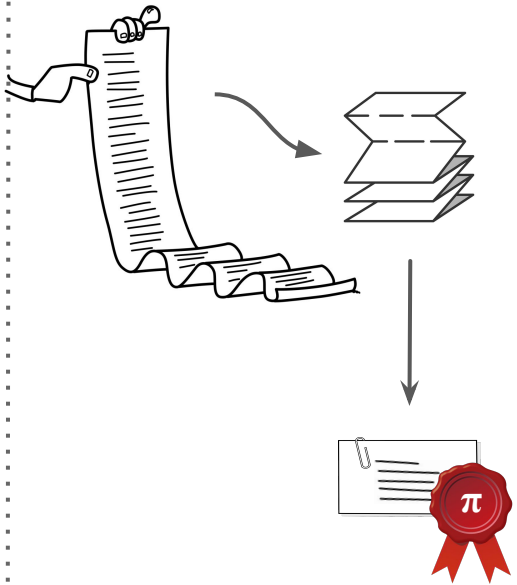
Not fast

Recursion



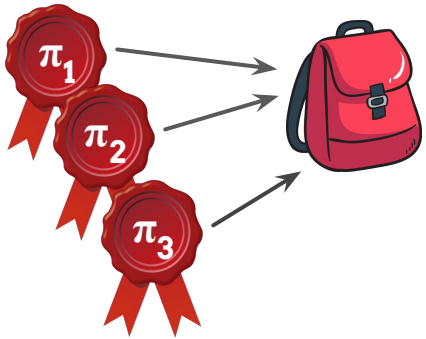
Faster

Folding



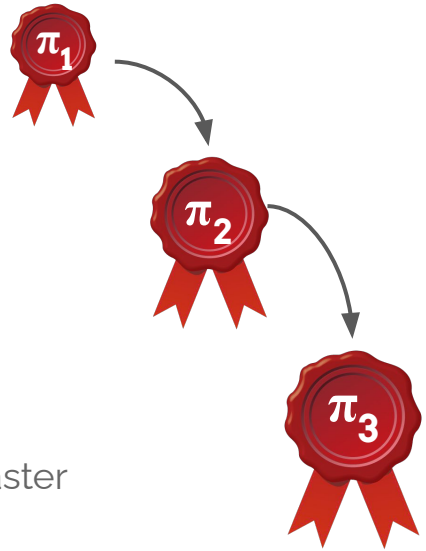
Prove Large Circuits

Aggregation



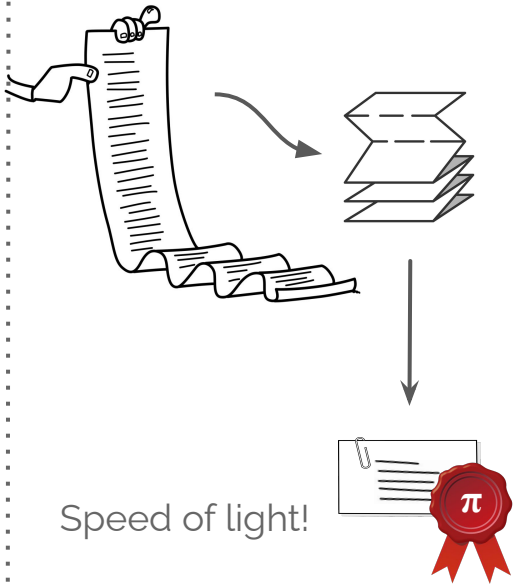
Not fast

Recursion



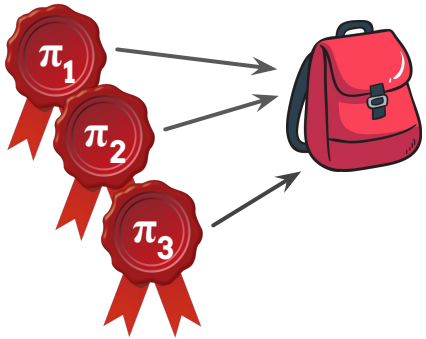
Faster

Folding



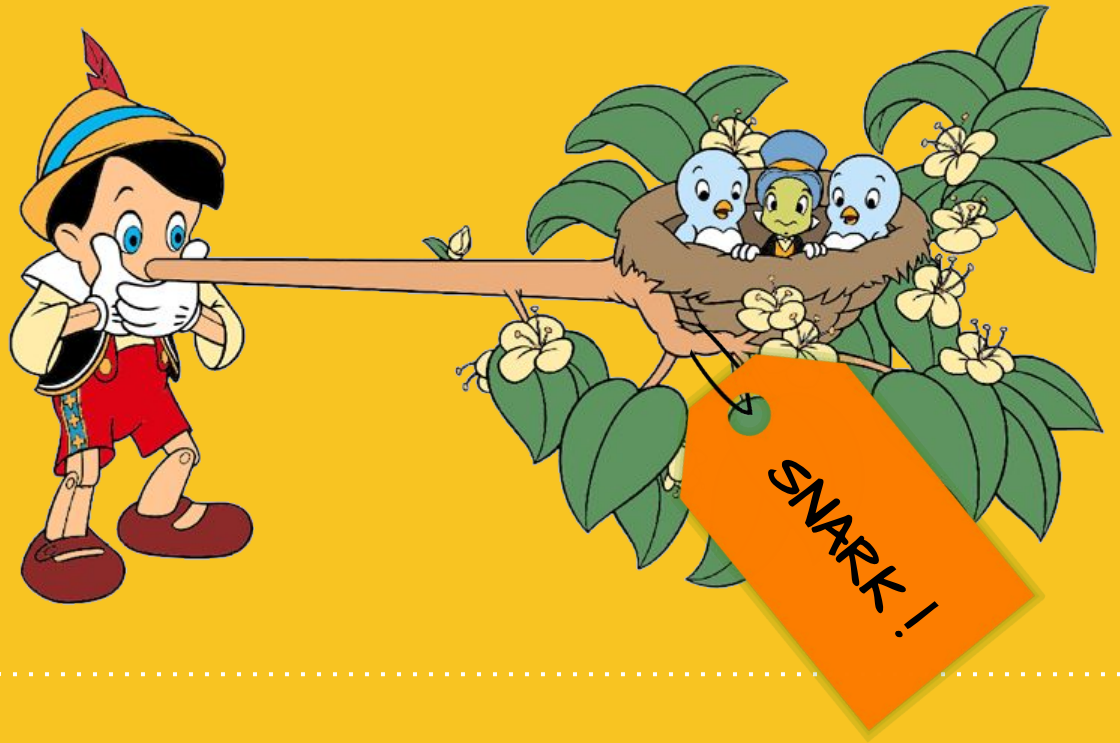
Speed of light!

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?

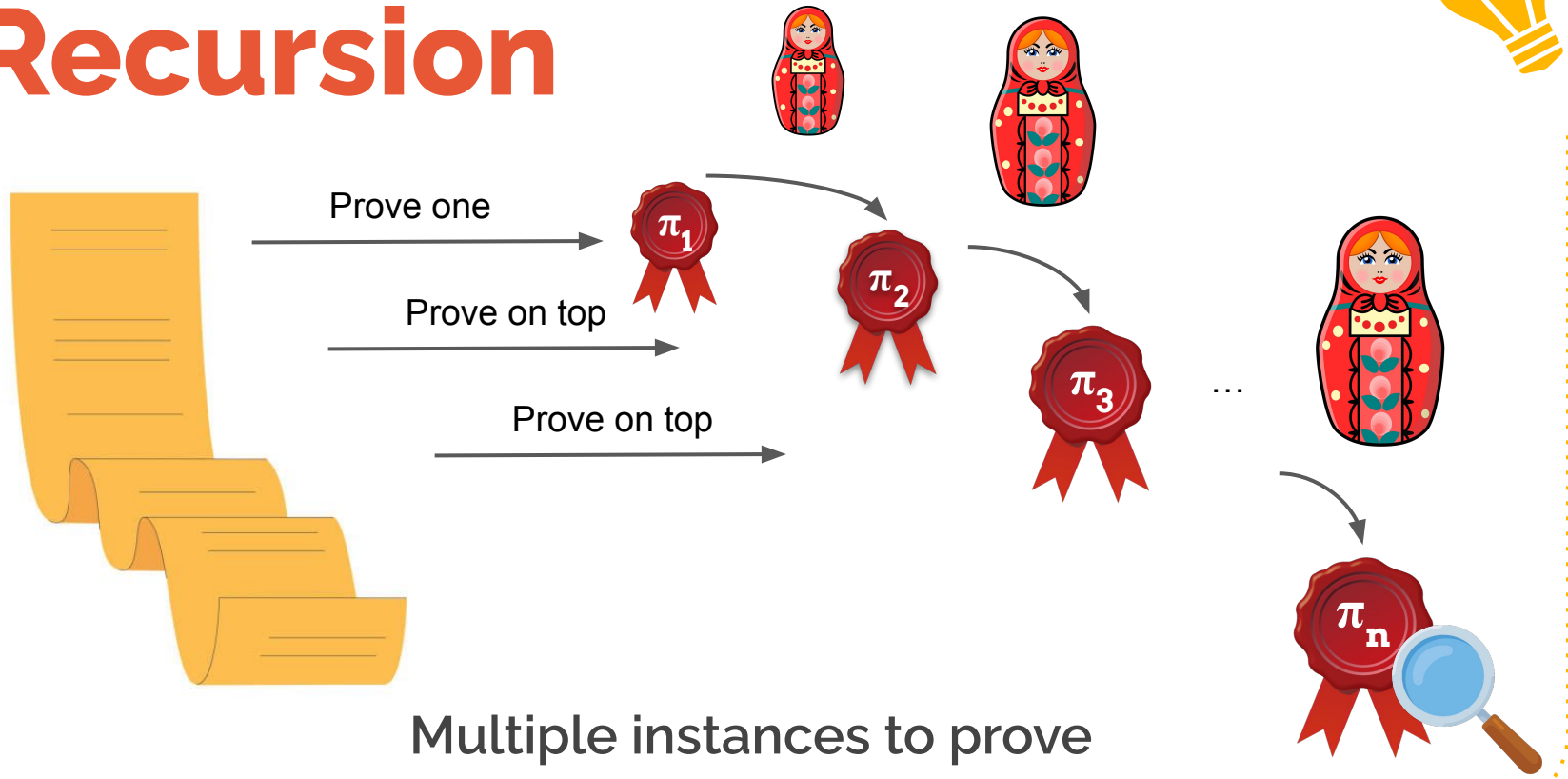
Expensive Prover



How faster can we go ?

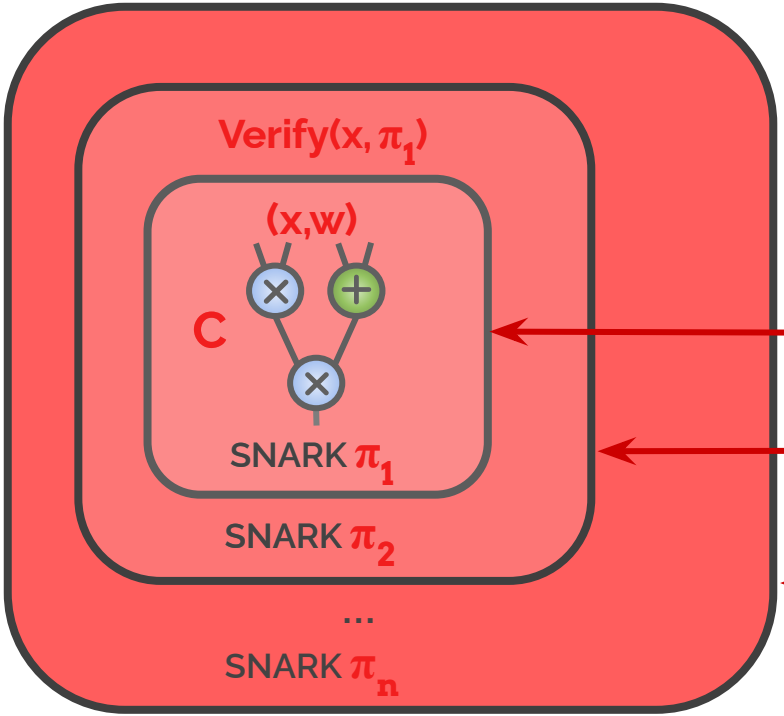
SNARK Recursion

Recursion



Multiple instances to prove

Proof Recursion



Really slow…

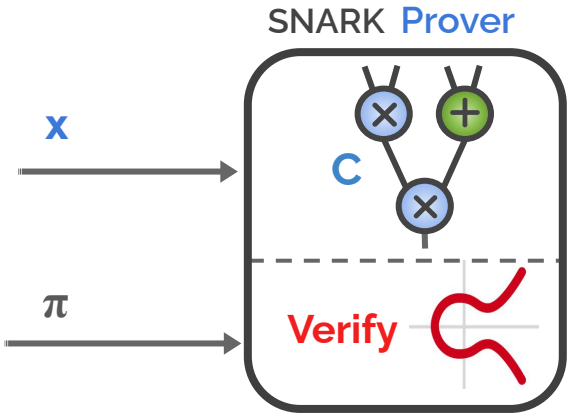
if Linear Verification

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

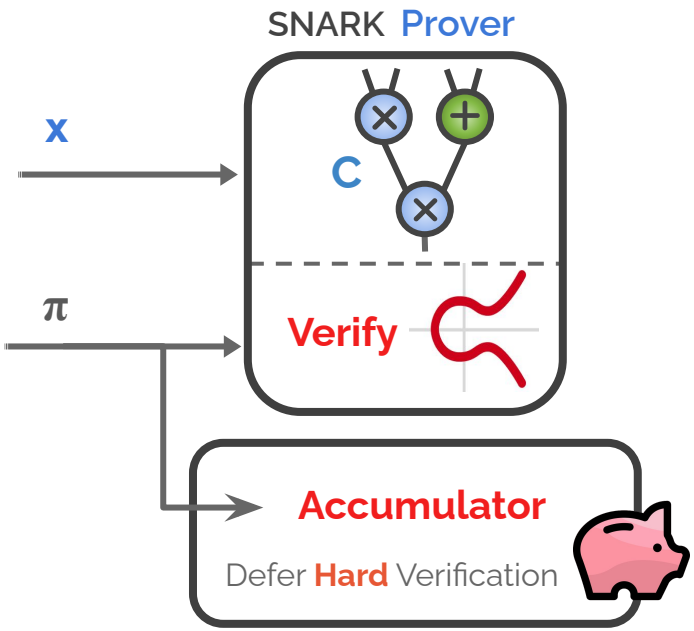
$|C_{Verify_2}| = 2|C_{Verify_1}| = 2^2|C_{SNARK}|$

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

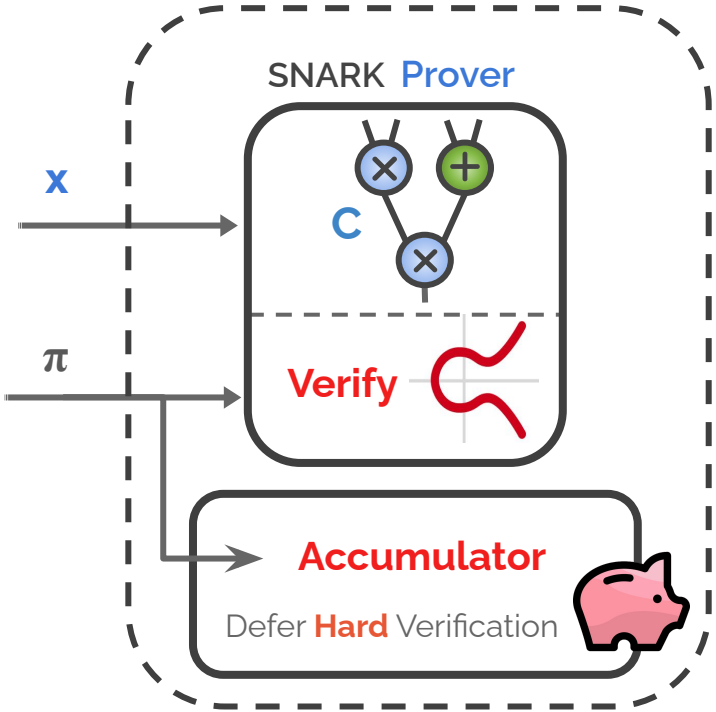
Atomic Accumulation



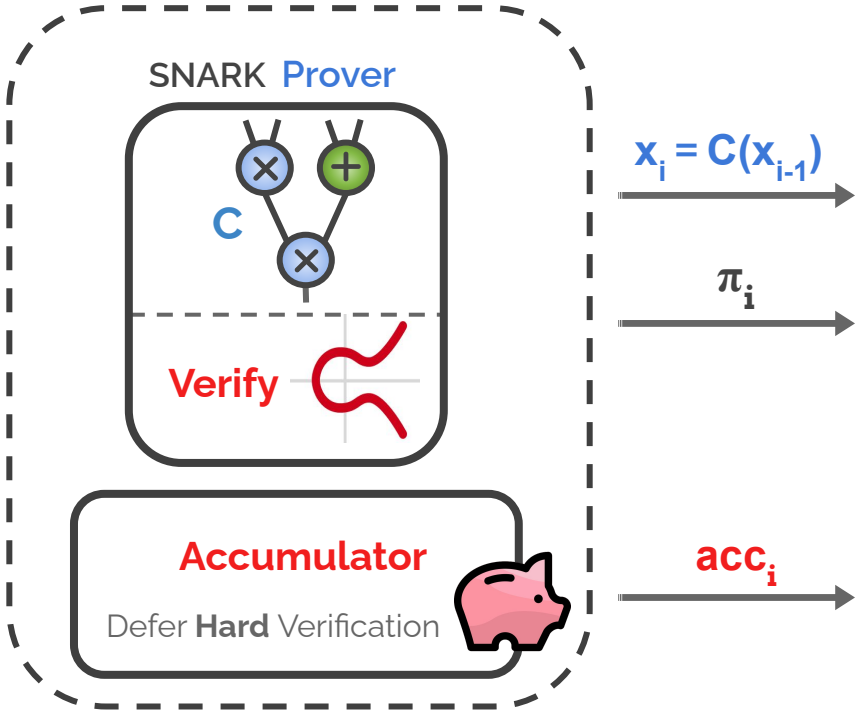
Atomic Accumulation



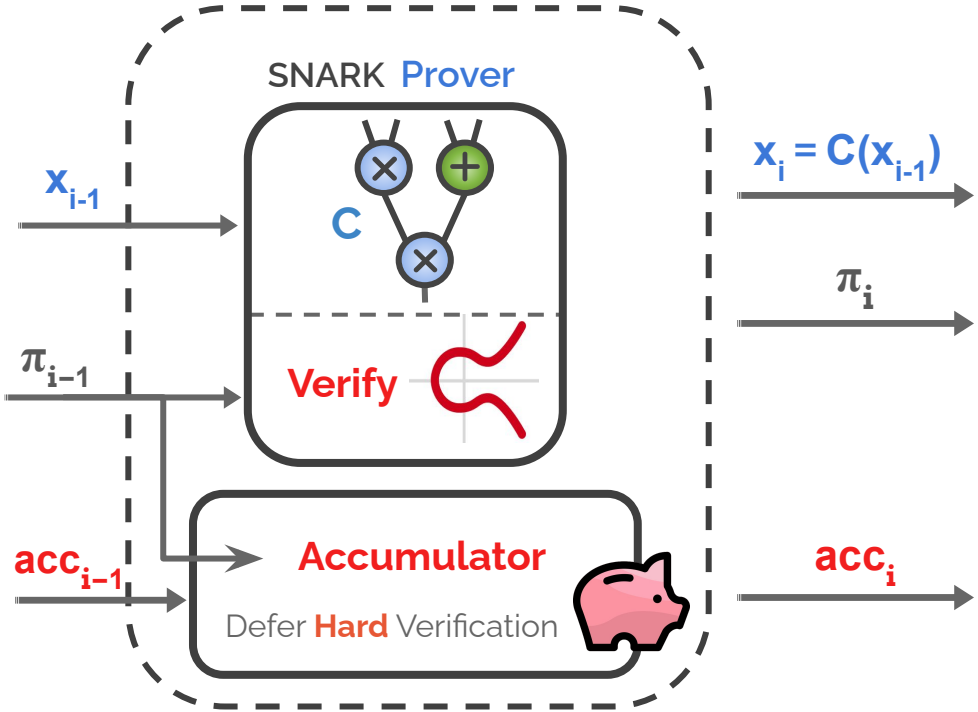
Atomic Accumulation



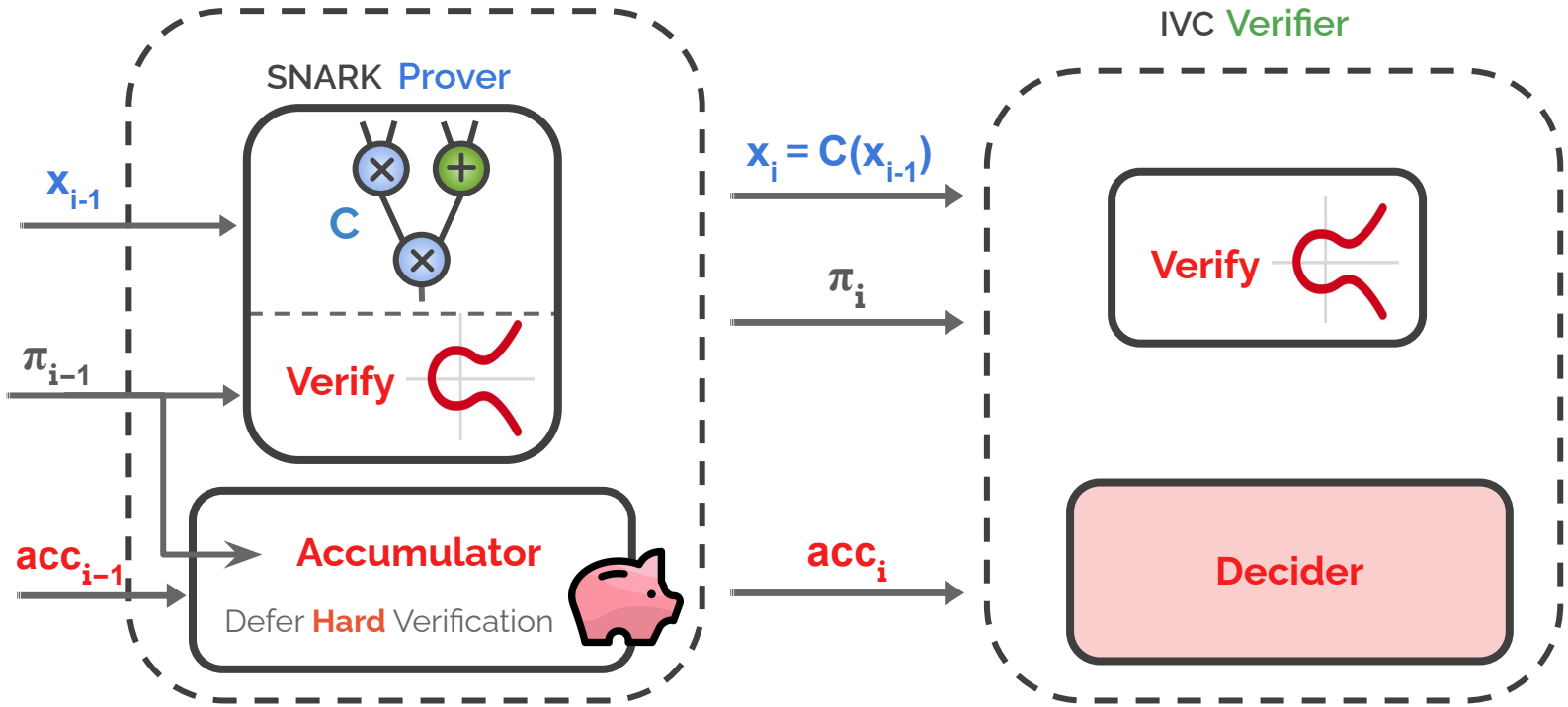
Atomic Accumulation



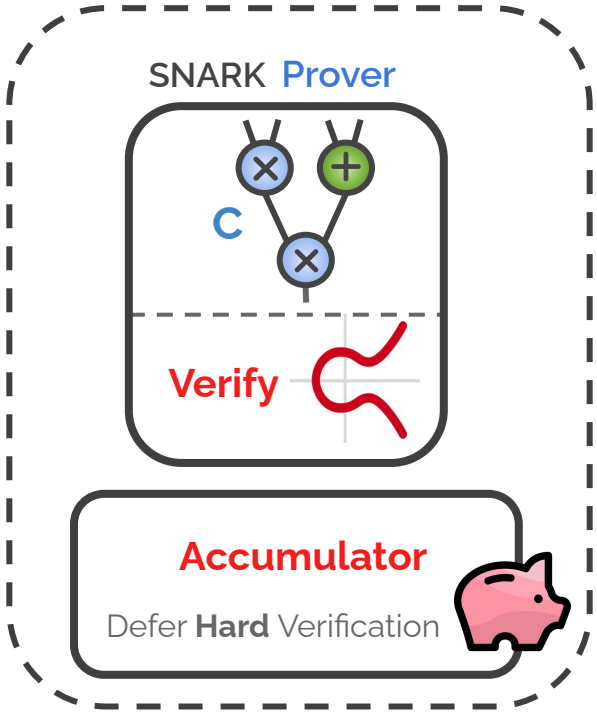
Atomic Accumulation



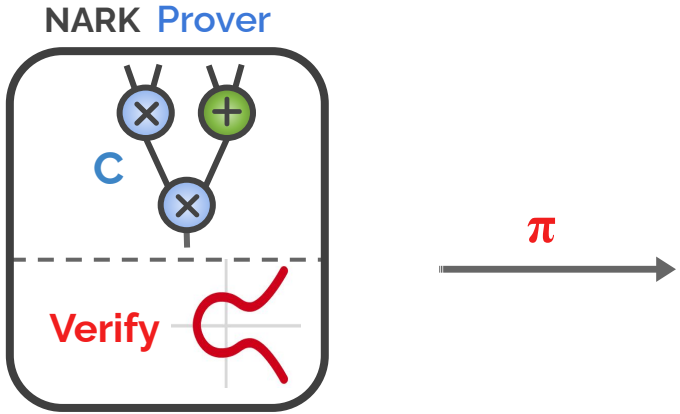
Atomic Accumulation



Atomic Accumulation



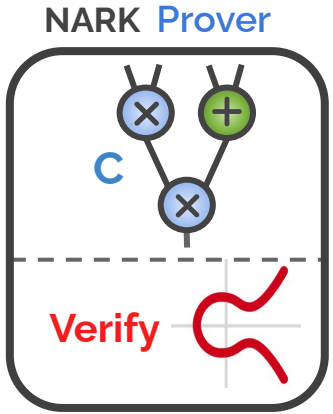
Split Accumulation



NARK:

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

Split Accumulation



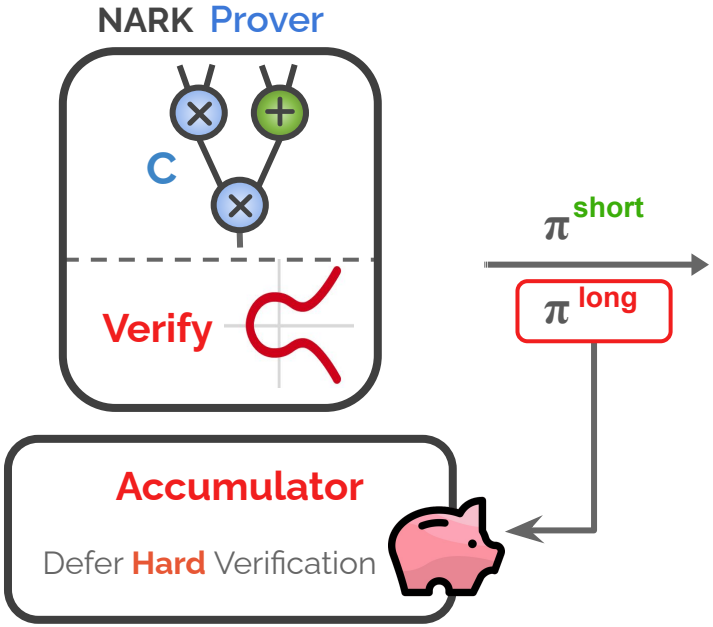
π^{short}

π^{long}

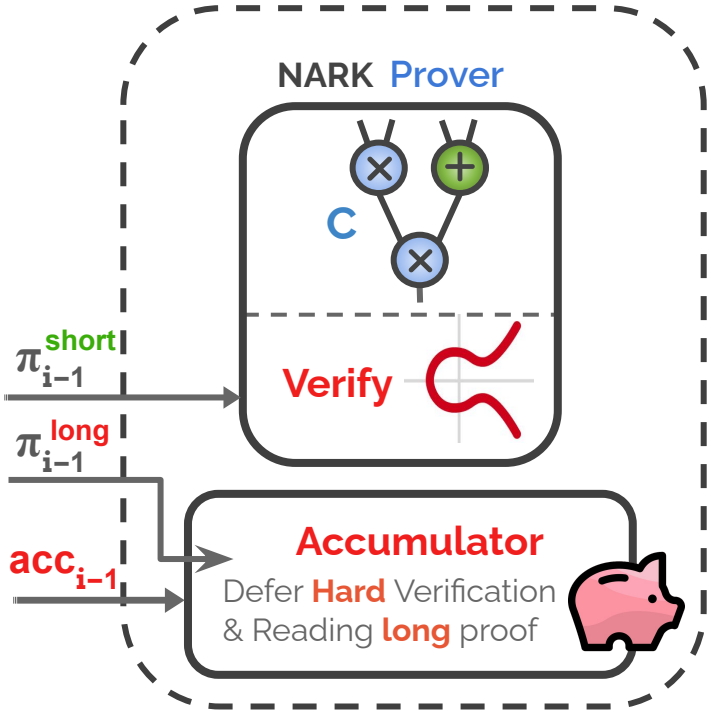
NARK:

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

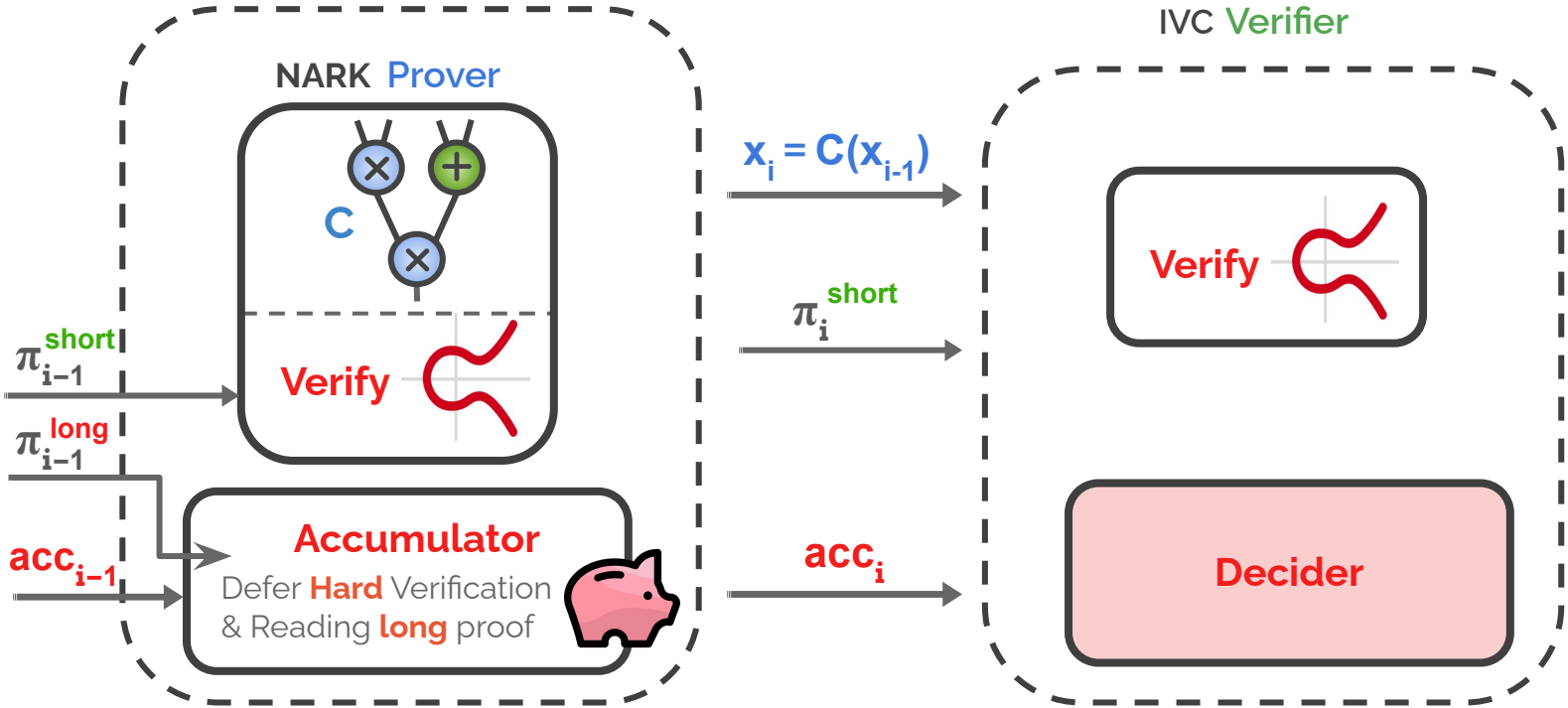
Split Accumulation



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

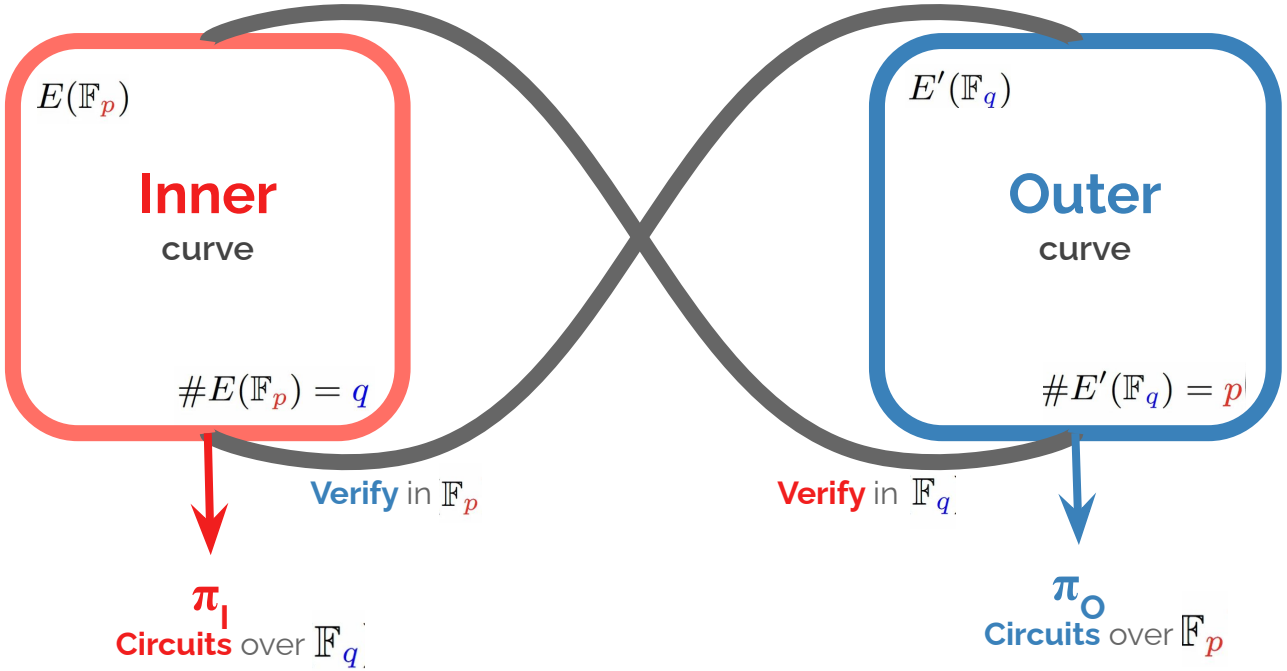
Recursion overview

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

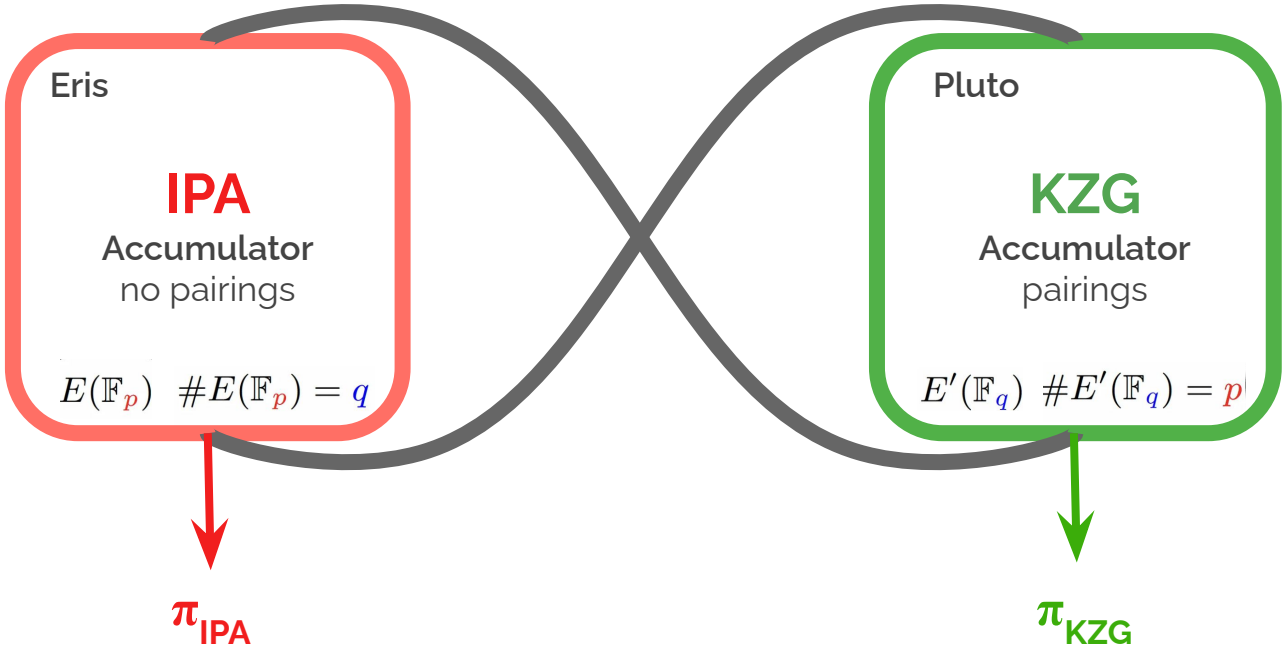
Prover computes FFTs
Needs cycles of curves

*also called folding

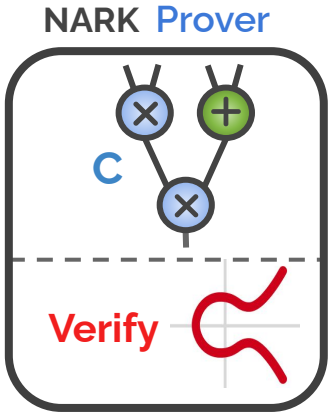
Cycles of Curves



Halo2 Curves



Folding Schemes



π

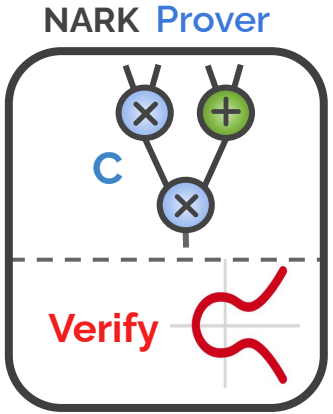
Why do we need a Prover if

|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!

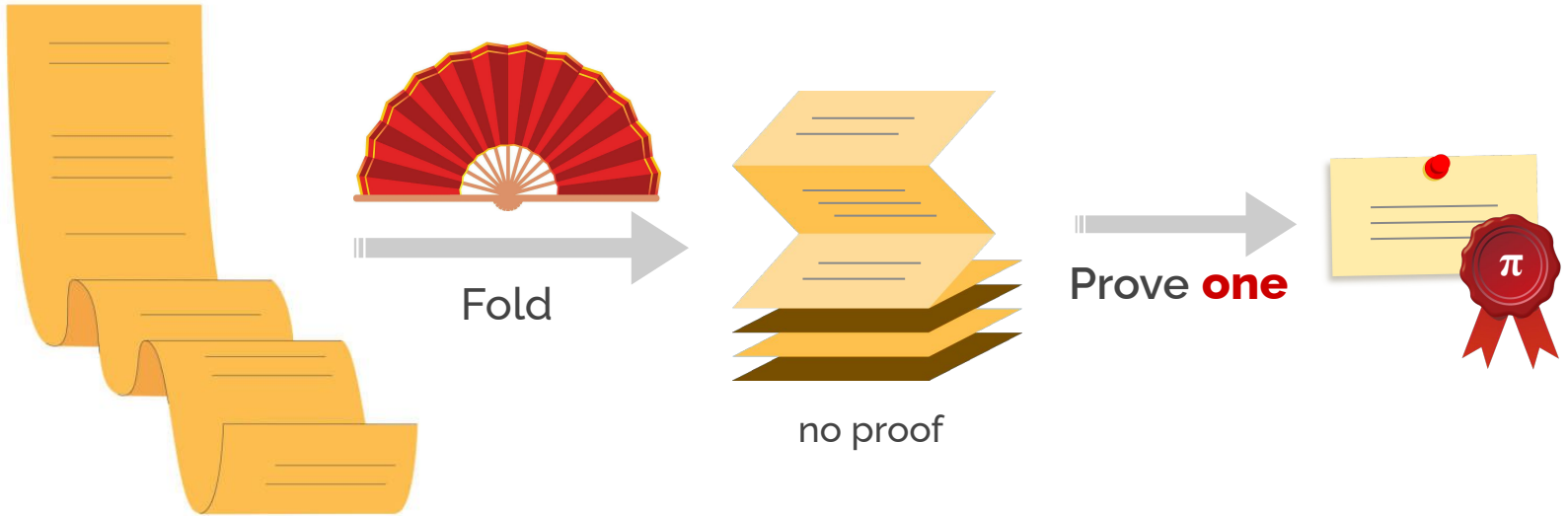
Recursion overview

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

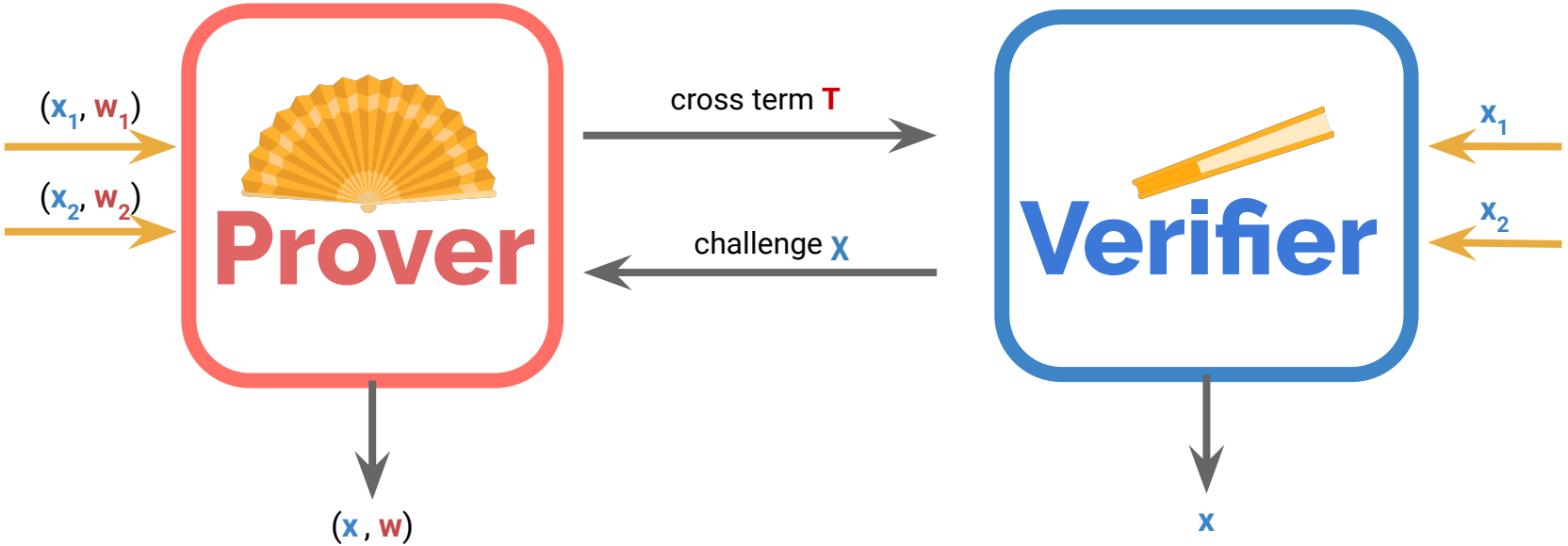
SNARK Folding

Folding



Multiple instances to prove

Folding idea



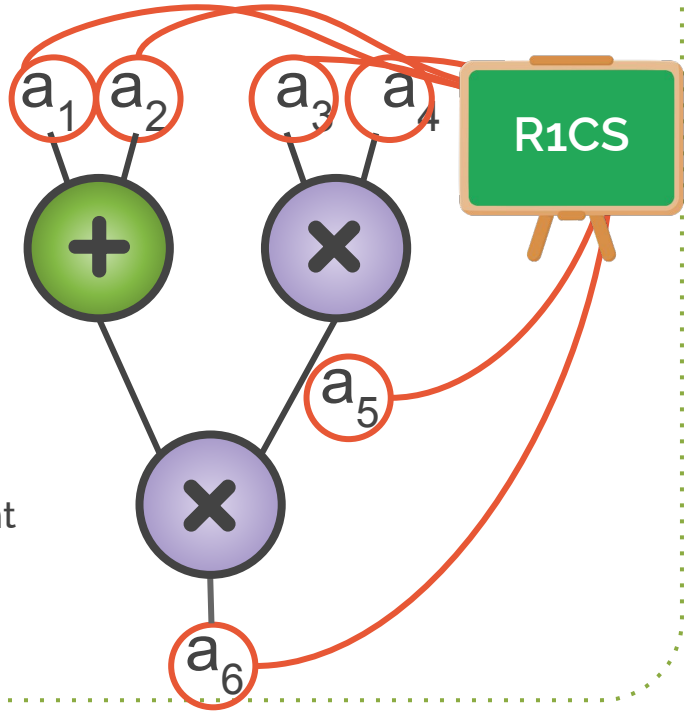
R1CS Rank-1 Constraint System

What are they?

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

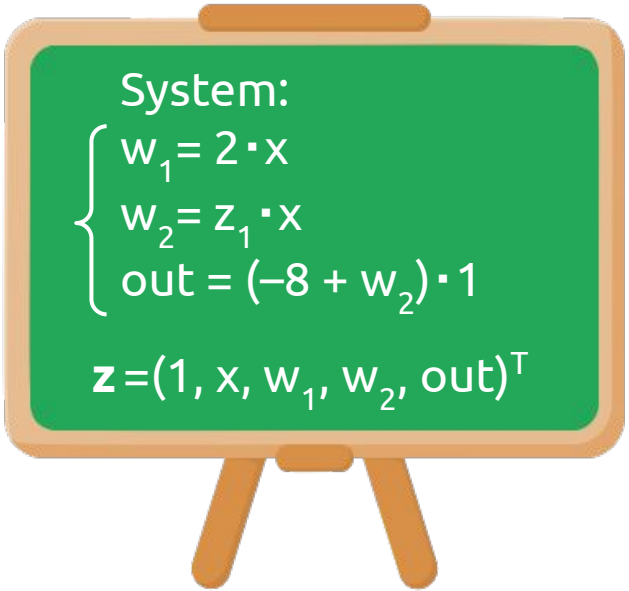
How to define R1CS?

- $\mathbf{A} \cdot \mathbf{z} \circ \mathbf{B} \cdot \mathbf{z} = \mathbf{C} \cdot \mathbf{z}$ 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 $2x^2 - 8 = 0$



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

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

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

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

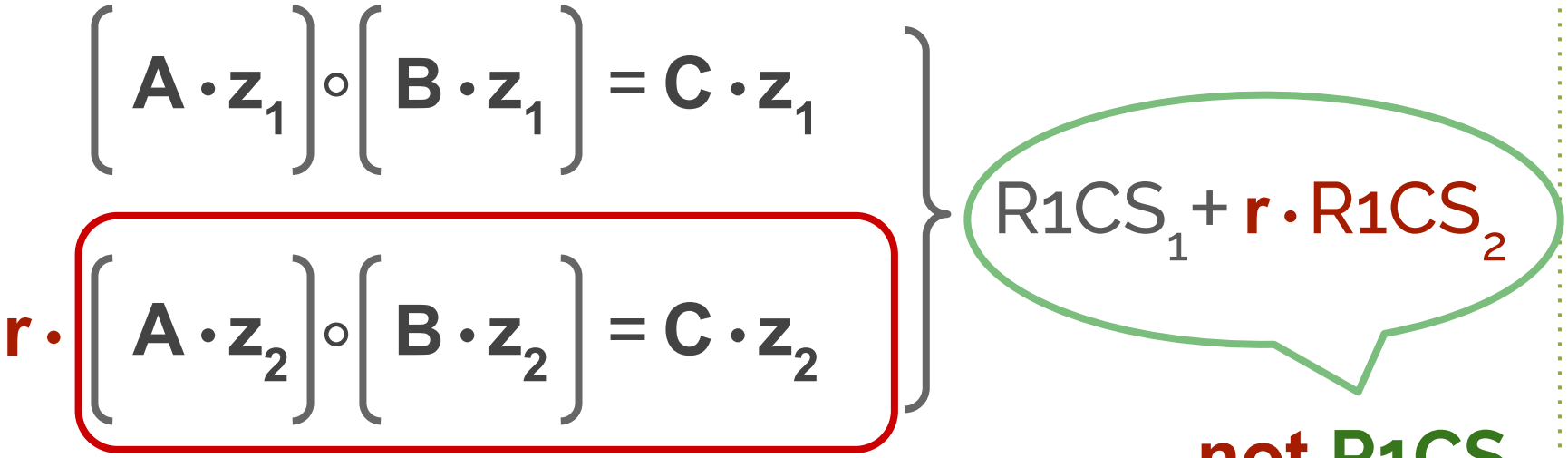
$$\left(A \cdot z\right) \circ \left(B \cdot z\right) = C \cdot z$$

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



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.

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

relaxed R1CS

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

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

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

Relaxed R1CS

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

R1CS

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

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

relaxed R1CS

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

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

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

$u = 1$

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

Relaxed R1CS

$$z_1 = (u_1, x_1, e_1, w_1)$$

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

$$z_2 = (u_2, x_2, e_2, w_2)$$

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

Relaxed R1CS

$$z_1 = (u_1, x_1, e_1, w_1)$$

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

$$r \cdot \left[z_2 = (u_2, x_2, e_2, w_2) \right]$$

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

relaxed R1CS

Relaxed R1CS

$z_1 = (u_1, x_1, e_1, w_1)$ $z_2 = (u_2, x_2, e_2, w_2)$

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

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

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

Committed Relaxed R1CS

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

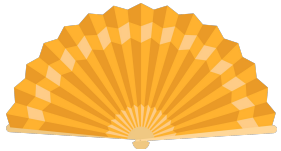
instance = (u, **x**, [**e**]) [**e**] = $\text{Com}_{ck}(\mathbf{e})$

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

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

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

Prover



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

Verifier

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

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

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

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

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

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

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

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

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

$u = u_1 + \chi u_2$

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

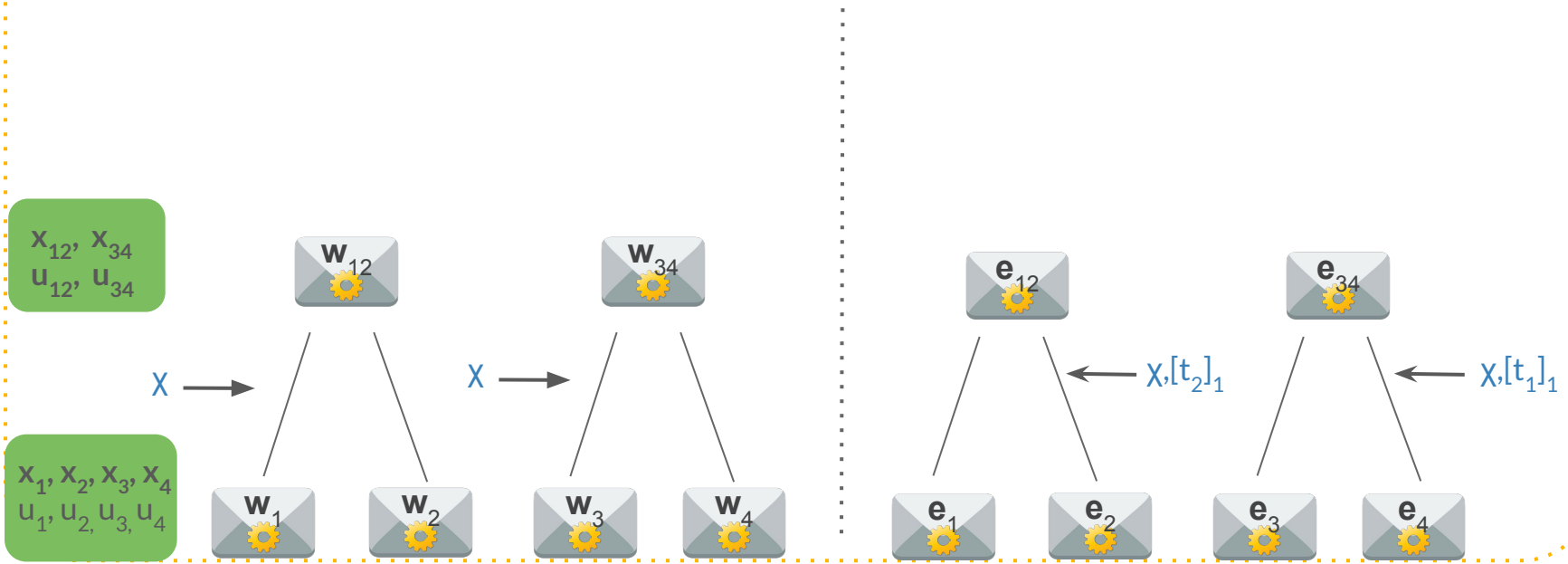
$w = (\mathbf{w}, \mathbf{e})$

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

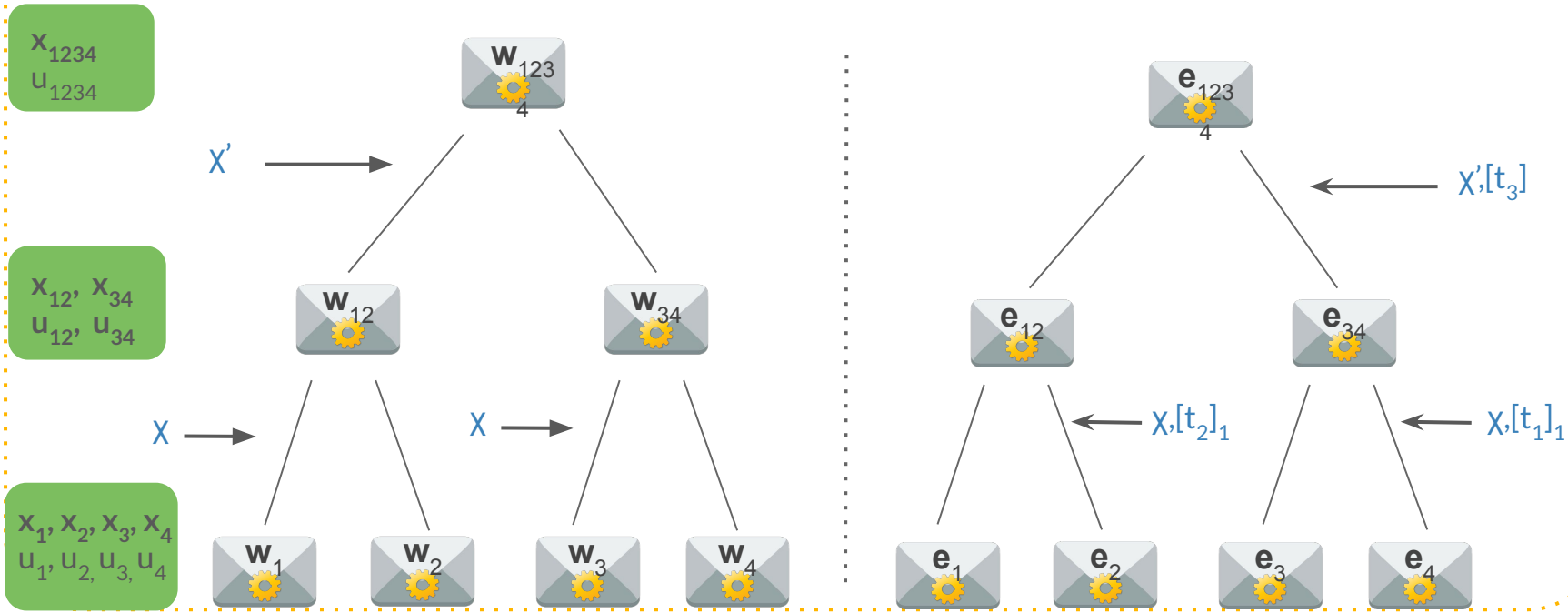
Nova Folding

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

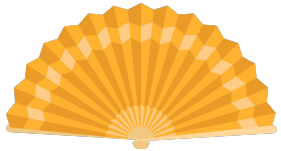
Nova Folding



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

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

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

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

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

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

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

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

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

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

$u = u_1 + \chi u_2$

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

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

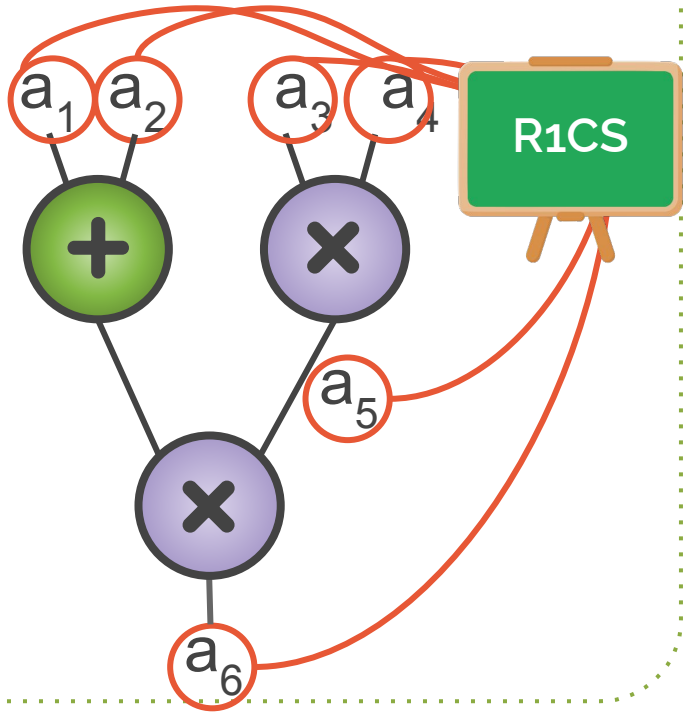
$x = (u, \mathbf{x}, [e]_1, [w]_1)$

Proving Final Instance

CR-R1CS Proof System

Spartan for CR-R1CS

- transparent setup
- sumcheck → very efficient prover
- logarithmic proof size



CR-R1CS Proof System

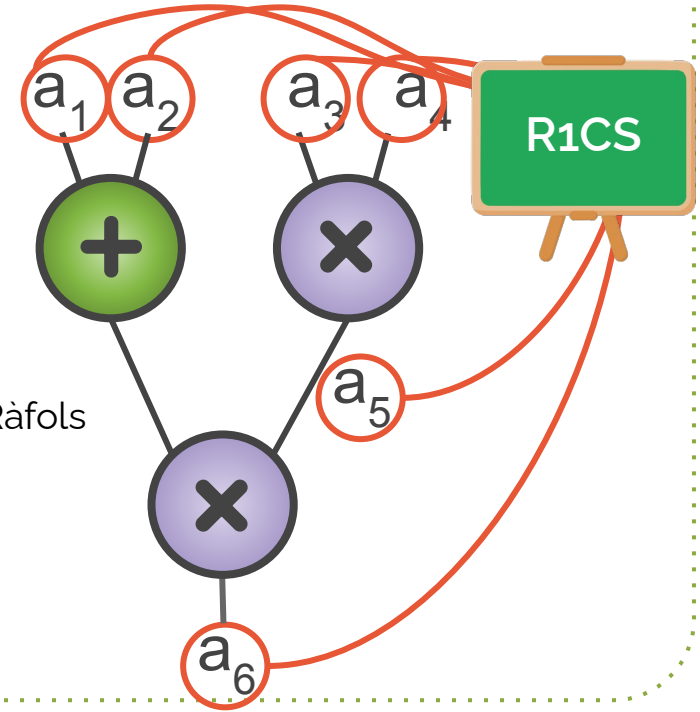
Spartan for CR-R1CS

- transparent setup
- sumcheck → very efficient prover
- logarithmic proof size

[FLIP-and-Prove R1CS](#) Anca Nitulescu, Nikitas Paslis, Carla Ràfols

Groth16 for CR-R1CS

- trusted circuit-dependent setup
- constant verifier, very small proof
- needs pairing-based curves



Prove large computations

Aggregation

SnarkPack

- Aggregation for **Groth16**
- R1CS arithmetization
- Circuit-Dependent Trusted Setup
- Pairing-friendly elliptic curve
- The prover must produce a SNARK at each step
- Logarithmic size final proof
- Logarithmic Verifier

Recursion

Halo2

- Recursion from cycle of curves
- Custom gates arithmetization
- Universal trusted setup
- FFT-friendly cycles of elliptic curves
- The prover must produce a SNARK at each step
- Accumulation: partially verify SNARK at each step, defer to end
- Logarithmic + size proof

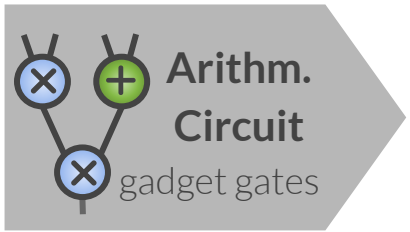
Folding

Nova

- Folding instances & witness
- relaxed-R1CS arithmetization
- Transparent/Universal setup
- Any plain cycle of elliptic curves
- The prover sends cross-terms
- One single final proof
- Needs a Commit-and-Prove SNARK for cr-R1CS

Arithmetization

SNARKs

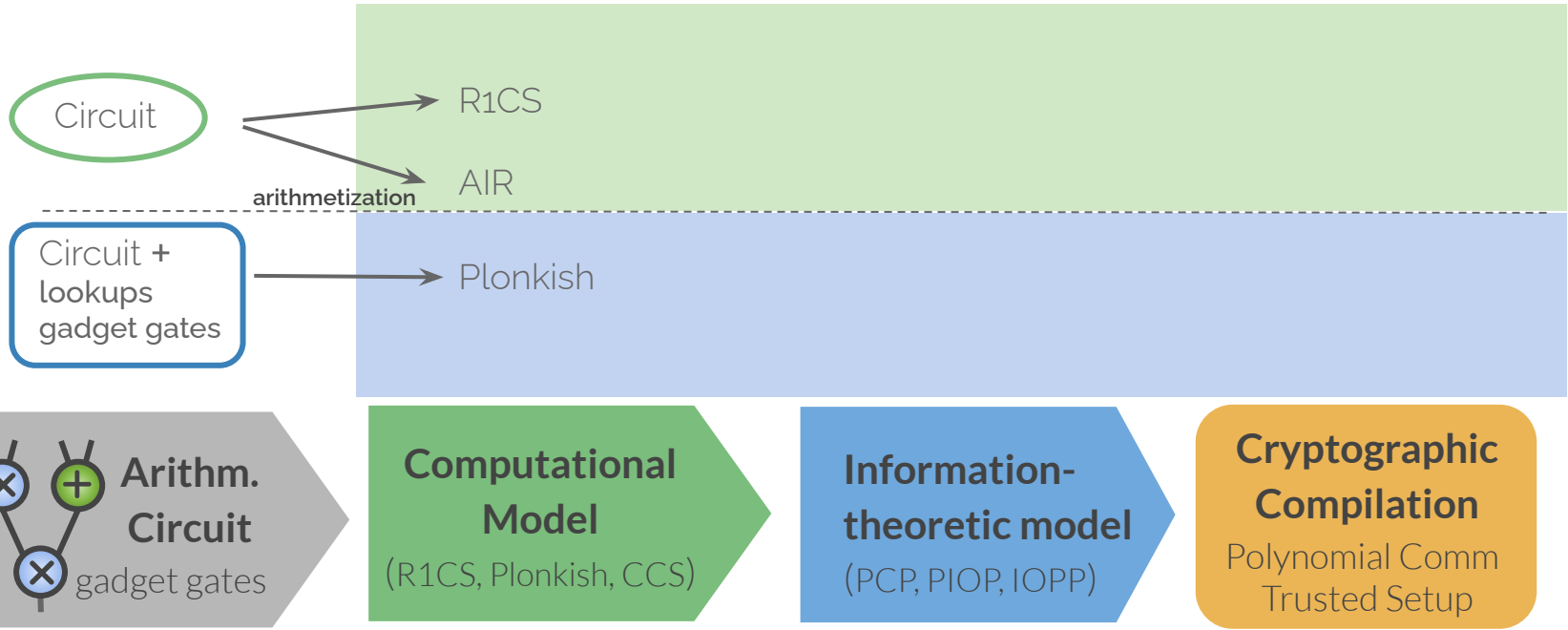


Computational Model
(R1CS, Plonkish, CCS)

Information-theoretic model
(PCP, PIOP, IOPP)

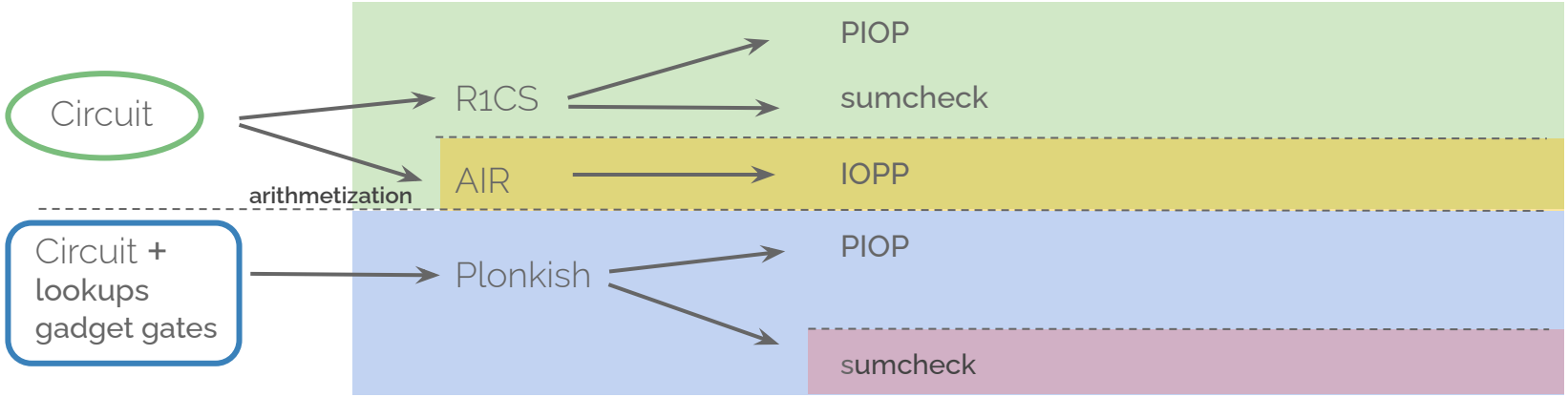
Cryptographic Compilation
Polynomial Comm
Trusted Setup

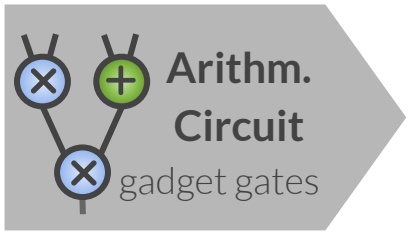
SNARKs



*SPARK PolyCommit for sparse multilinear polynomials

SNARKs





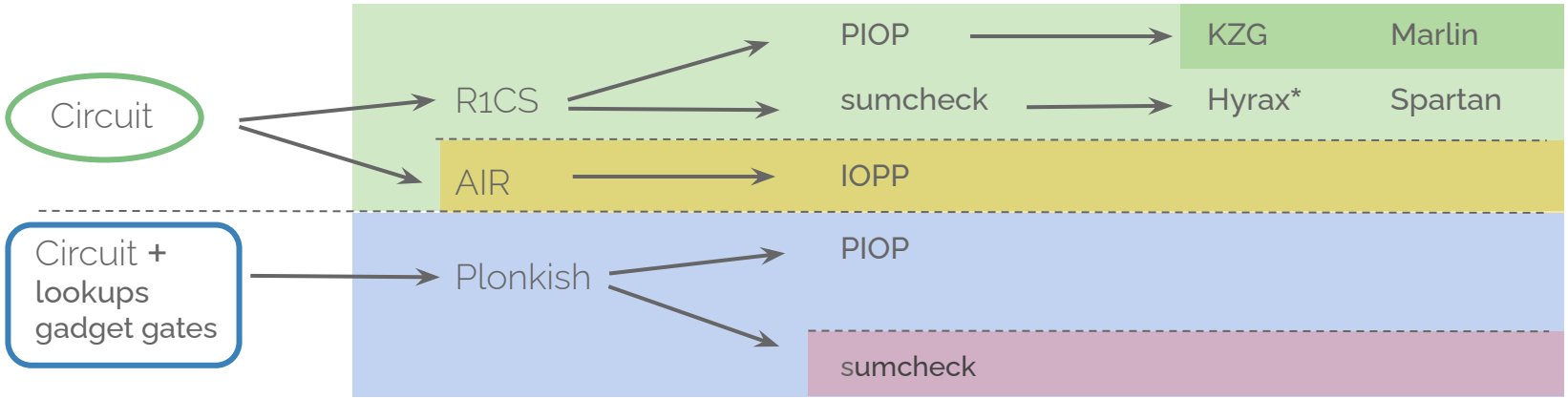
Computational Model
(R1CS, Plonkish, CCS)

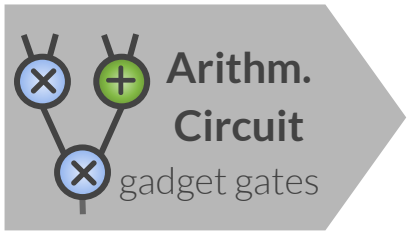
Information-theoretic model
(PCP, PIOP, IOPP)

Cryptographic Compilation
Polynomial Comm
Trusted Setup

*SPARK PolyCommit for sparse multilinear polynomials

SNARKs





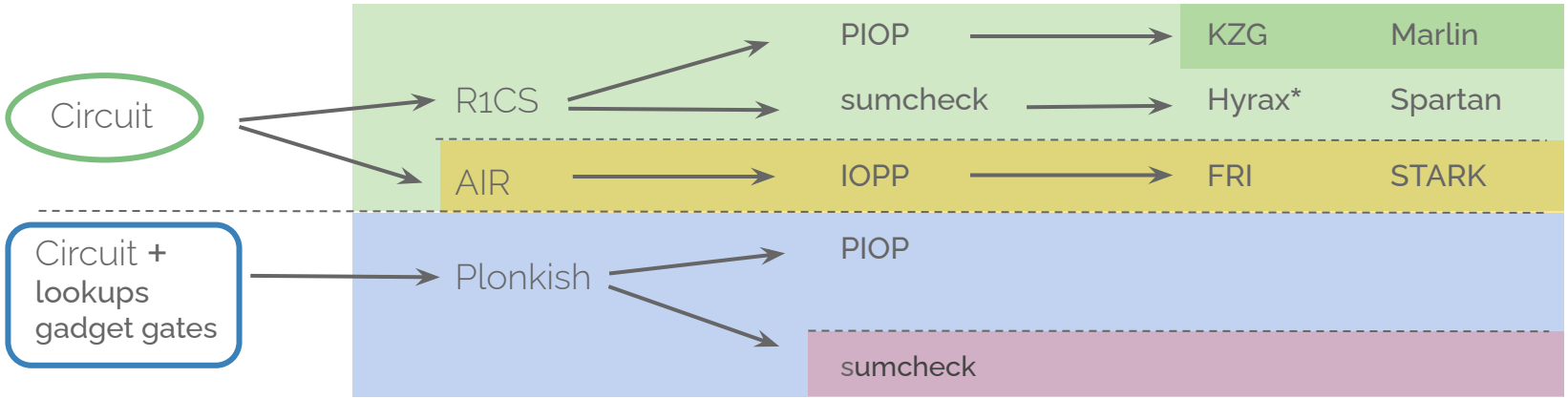
Computational Model
(R1CS, Plonkish, CCS)

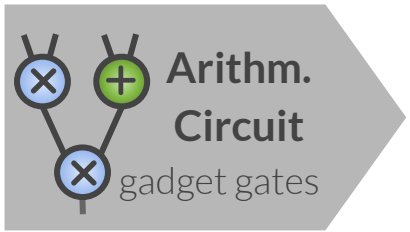
Information-theoretic model
(PCP, PIOP, IOPP)

Cryptographic Compilation
Polynomial Comm
Trusted Setup

*SPARK PolyCommit for sparse multilinear polynomials

SNARKs





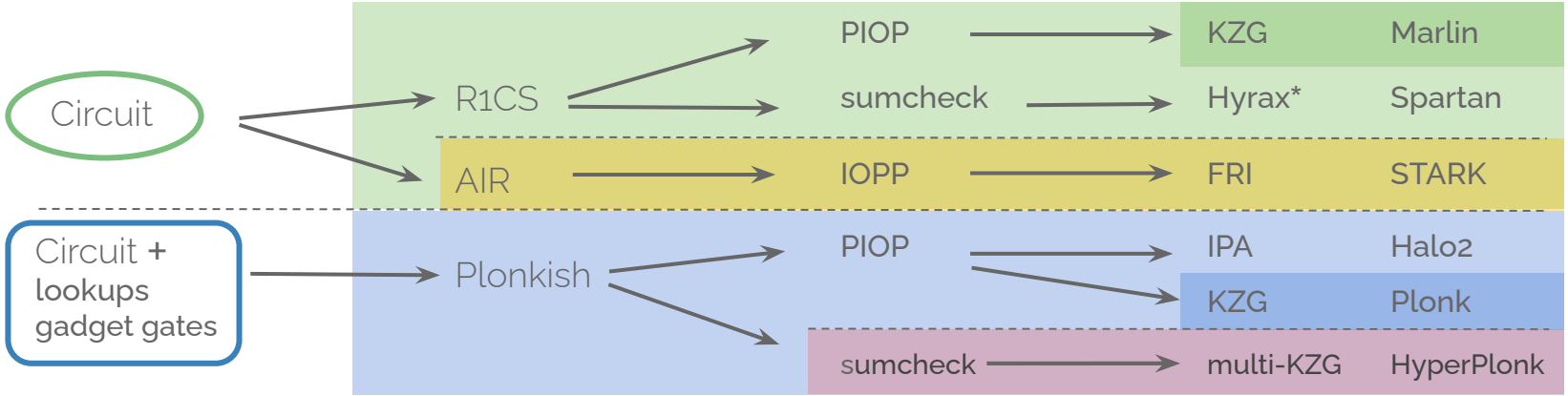
Computational Model
(R1CS, Plonkish, CCS)

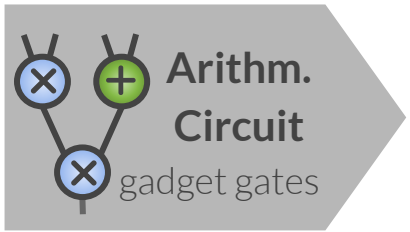
Information-theoretic model
(PCP, PIOP, IOPP)

Cryptographic Compilation
Polynomial Comm
Trusted Setup

*SPARK PolyCommit for sparse multilinear polynomials

SNARKs





Computational Model
(R1CS, Plonkish, CCS)

Information-theoretic model
(PCP, PIOP, IOPP)

Cryptographic Compilation
Polynomial Comm
Trusted Setup

*SPARK PolyCommit for sparse multilinear polynomials

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

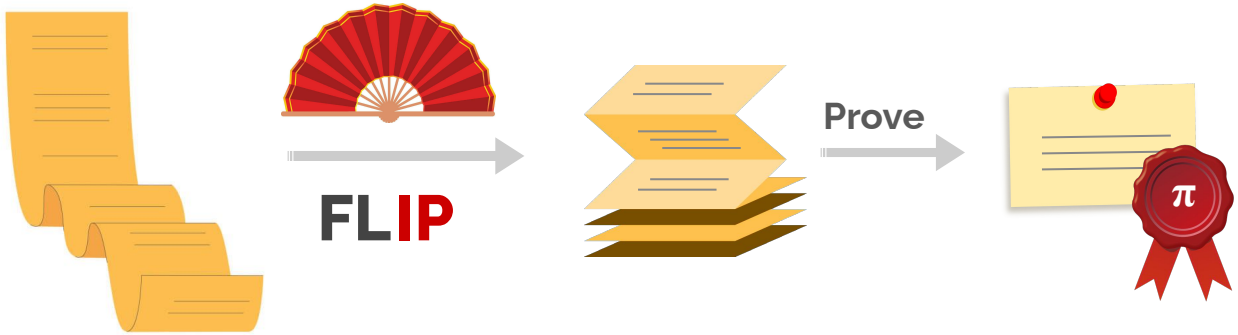
ProtoStar – Improves HyperNova

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

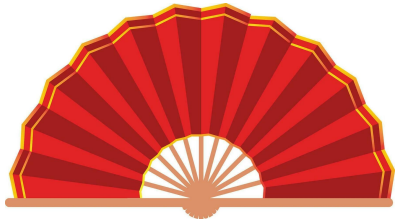
FLIP & Prove

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**) $\rightarrow$ 

$= [A(\tau)]_1$

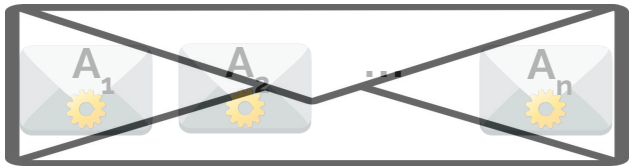
$A(X) = a_1 + a_2X + a_3X^2 + \ldots a_nX^{n-1}$

Target-Group Commitment

Vector Commitment

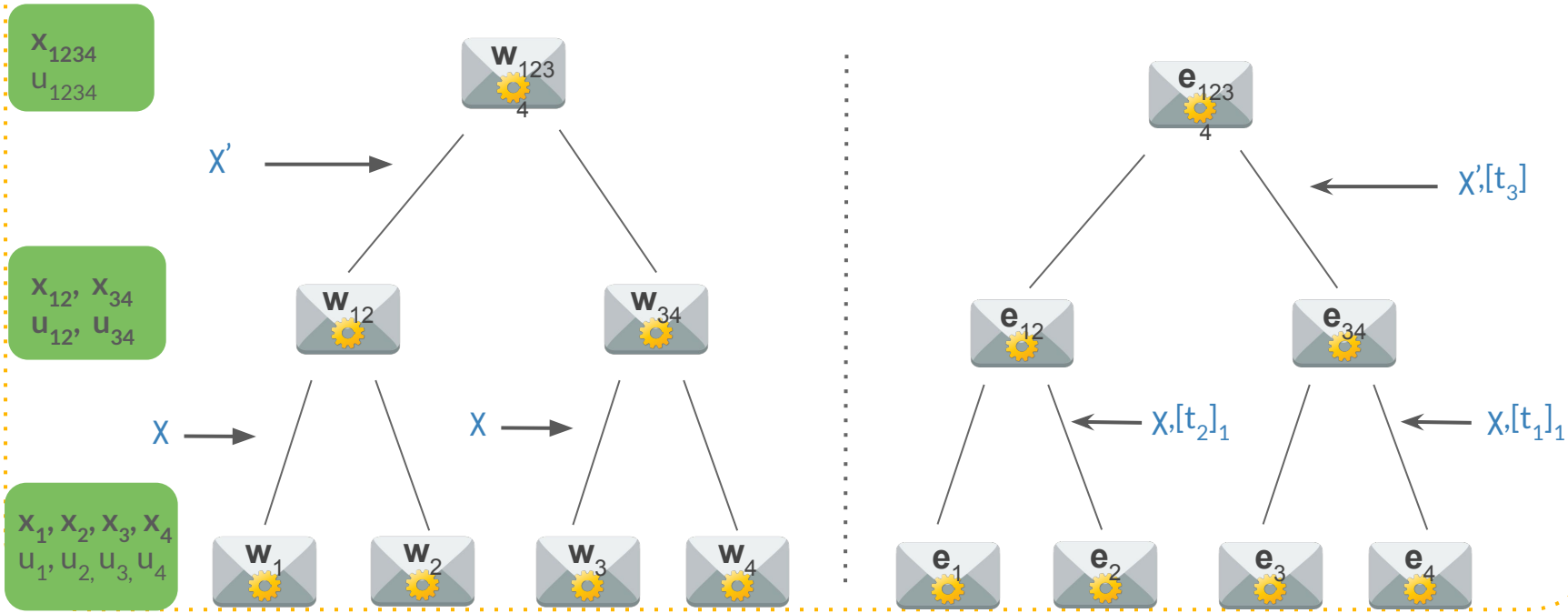
KeyGen(λ, n) $\rightarrow$ ck: $[1]_2, [\tau]_2, [\tau^2]_2, \ldots [\tau^n]_2$

Target-Group Commitment



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

Nova-style Folding

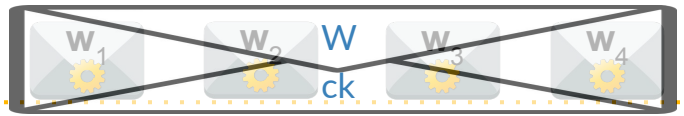


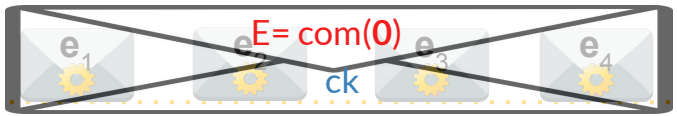
FLIP-style Folding

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

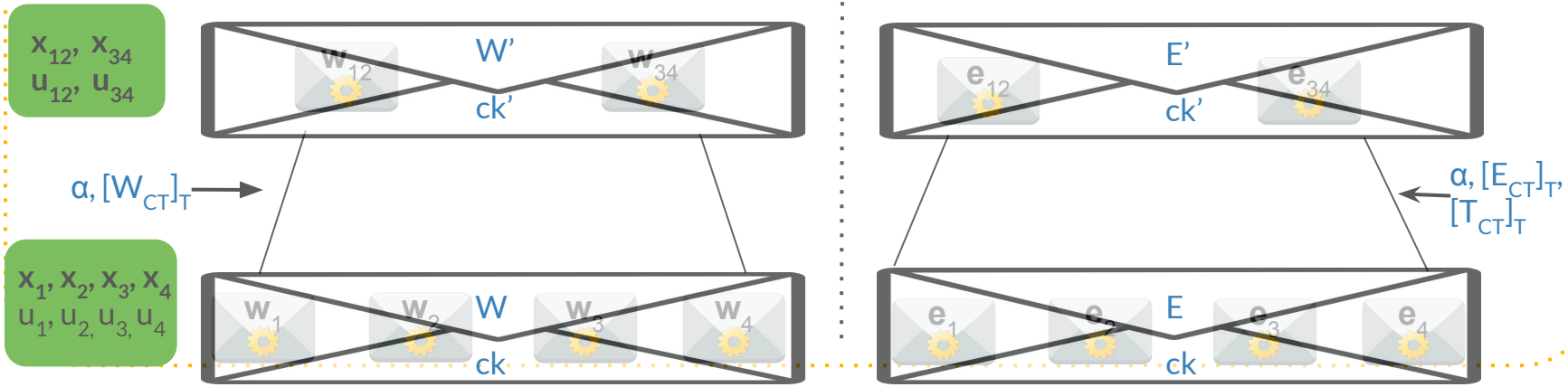
FLIP-style Folding

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

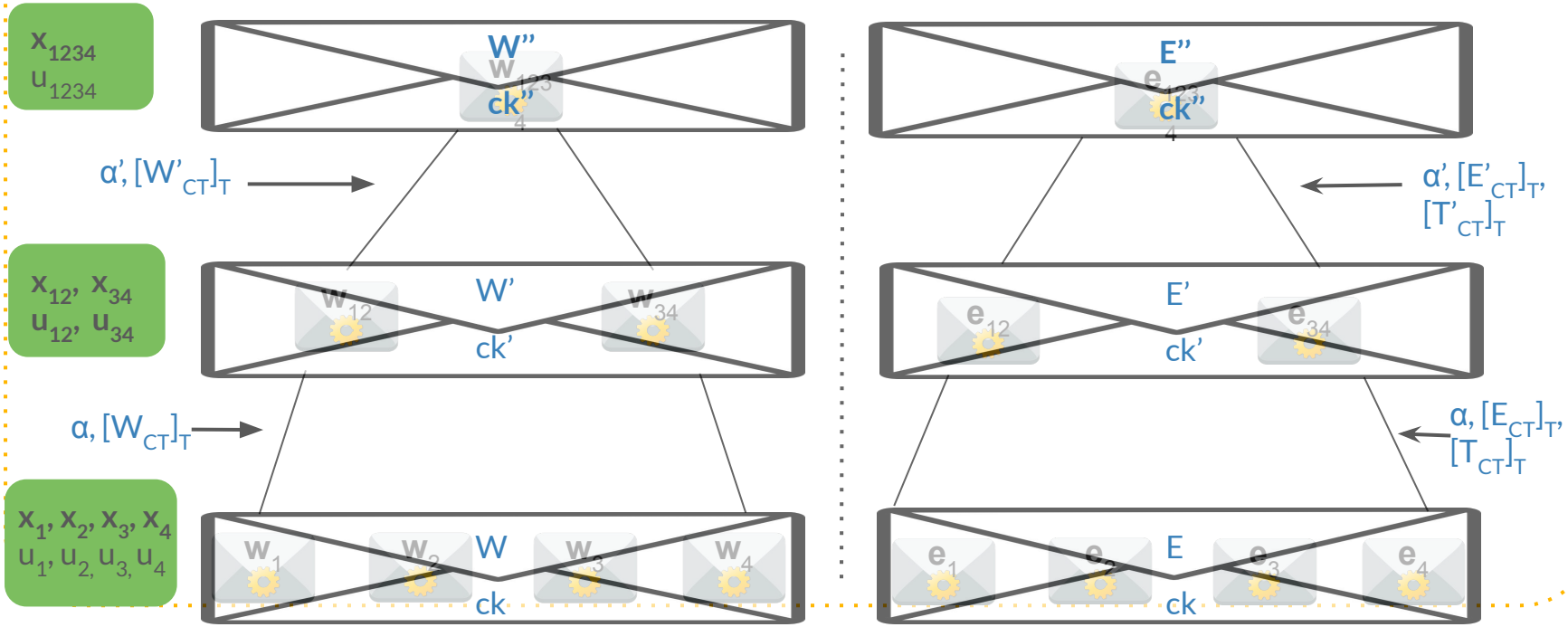




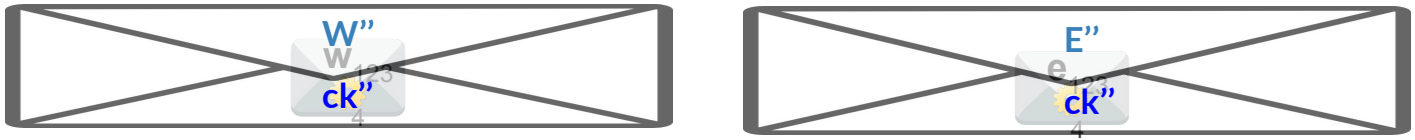
FLIP-style Folding



FLIP-style Folding



FLIP-style Folding



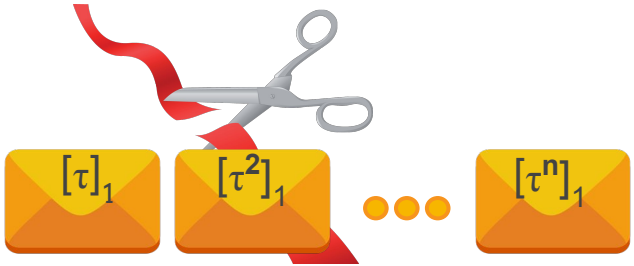
$\mathbf{W} = (W_1, W_2, \ldots W_n)$

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

W_{final}

Liniar verification

Solution: Structured Setup



Logarithmic verification for folding commitment key **ck**

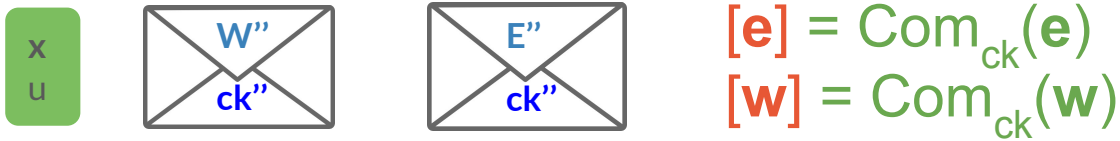
$\mathbf{W} = (W_1, W_2, \ldots W_n)$

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

W_{final}

Proving Final Instance

Proving FLIP



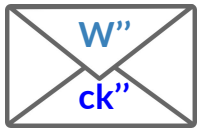
$[\mathbf{e}] = \mathrm{Com}_{ck}(\mathbf{e})$
$[\mathbf{w}] = \mathrm{Com}_{ck}(\mathbf{w})$

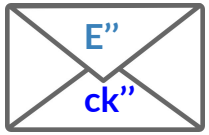
Committed Relaxed R1CS

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



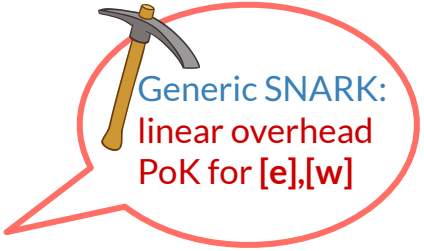


$[\mathbf{e}] = \mathrm{Com}_{ck}(\mathbf{e})$
$[\mathbf{w}] = \mathrm{Com}_{ck}(\mathbf{w})$

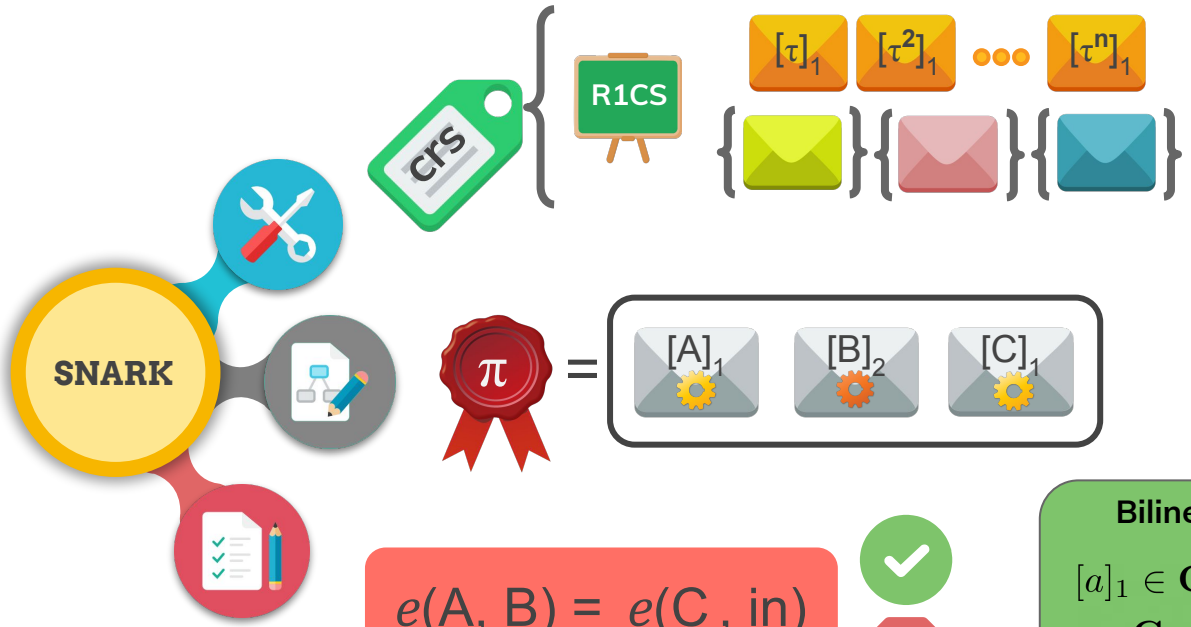
Committed Relaxed R1CS

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

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



Groth 16

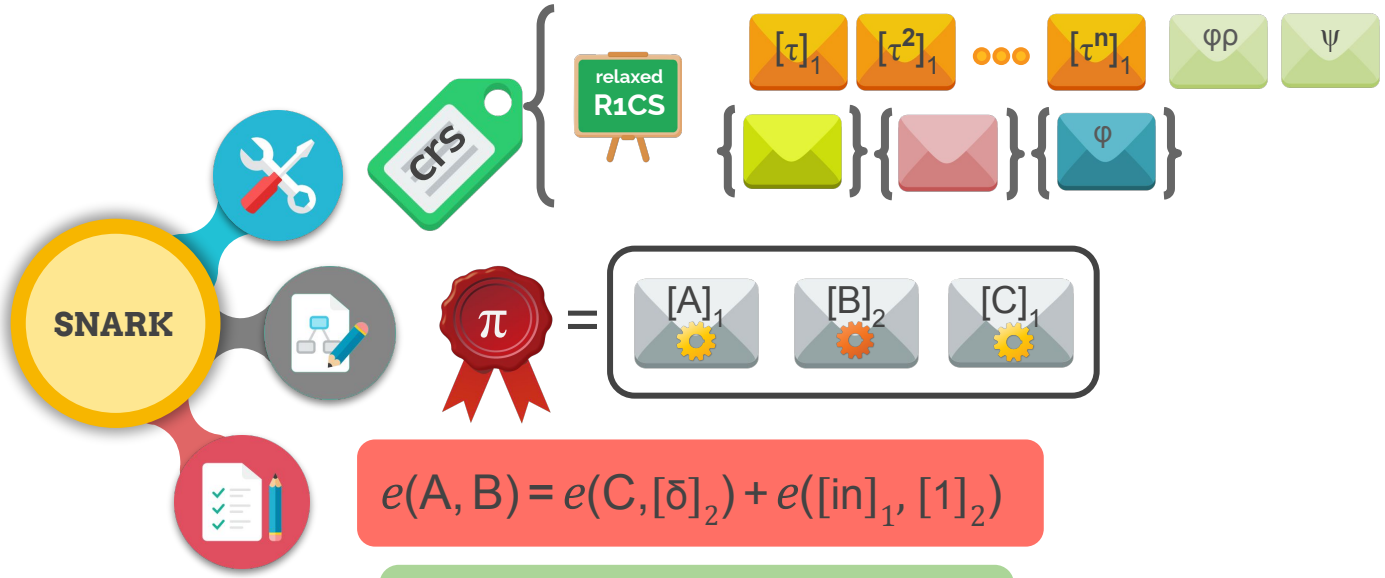


Bilinear Groups

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

Relaxed Groth 16

for Committed Relaxed R1CS

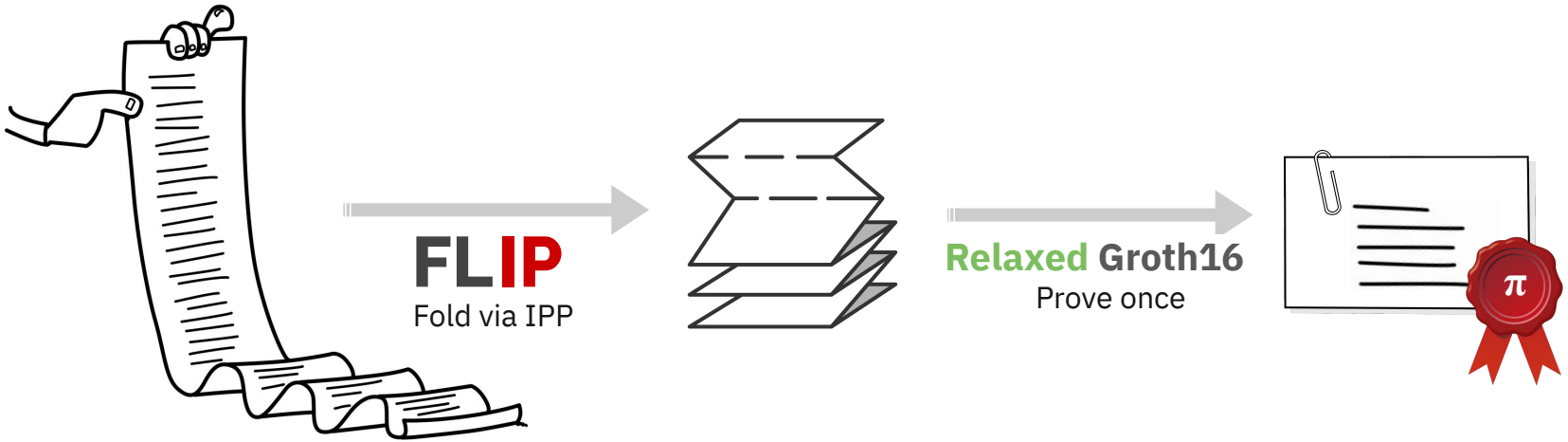


$$e(A, B) = e(C, [\delta]_2) + e([in]_1, [1]_2)$$

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

Summary

Multiple instances to prove



Thanks

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