recursive proof composition

ZKProof Standards 5.5
2 Aug 2023

Ying Tong (Geometry Research)

agenda

1. **overview**
   a) motivation
   b) constructions

2. **comparison**
   a) implementations
   b) recursion threshold
   c) support for lookup arguments

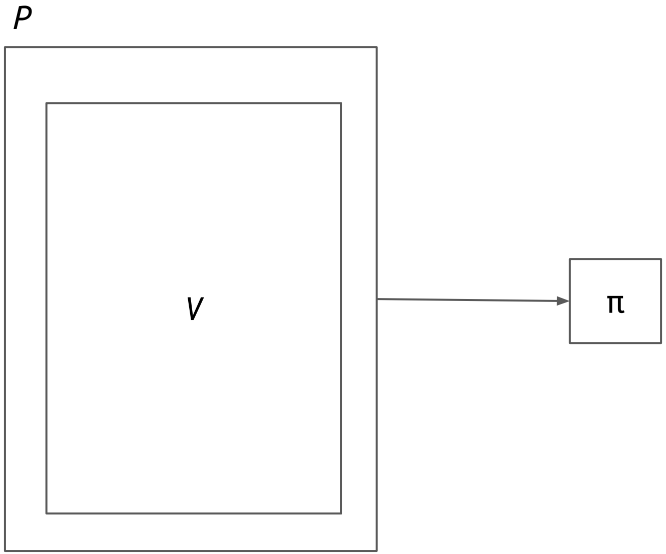
3. **future work**
   a) tooling & interfaces
   b) benchmarking
   c) standards & specifications
   d) security

agenda

1. **overview**
   a) motivation
   b) constructions

2. **comparison**
   a) recursion threshold
   b) zero-knowledgeness
   c) support for lookup arguments

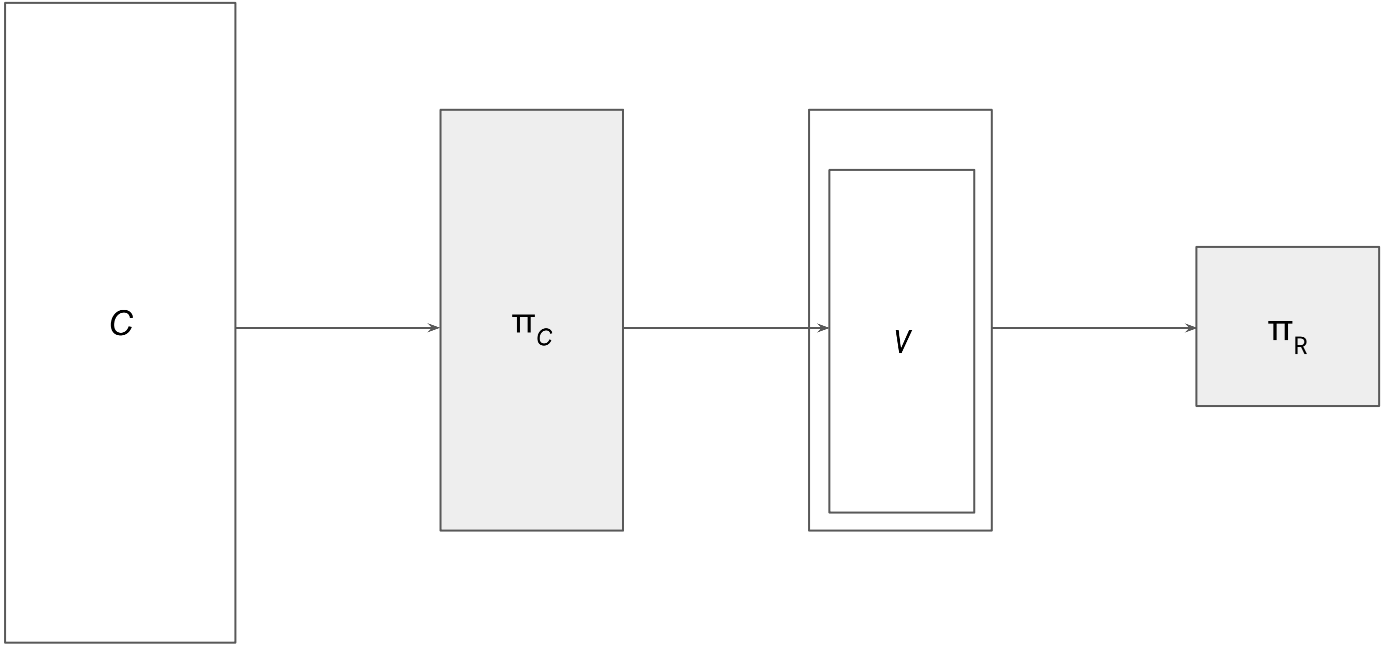
3. **future work**
   a) tooling & interfaces
   b) benchmarking
   c) standards & specifications



*a **recursive proof** is a proof that enforces the accepting computation of the **proof system's own verifier***

overview: *motivation*

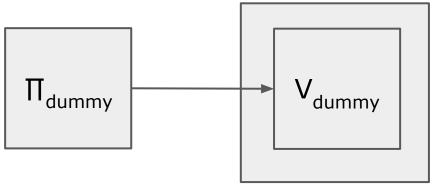
shrinking proof size



overview: *motivation*

shrinking proof size

```rust
// Start with a dummy proof of specified size
let inner = dummy_proof::<F, C, D>(config, log2_inner_size)?;
let (_, _, cd) = &inner;
```



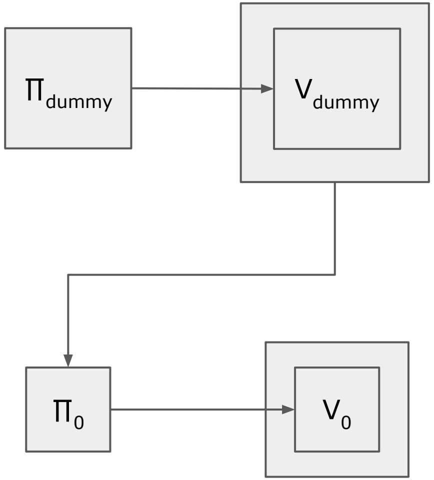
https://github.com/mir-protocol/plonky2/blob/main/plonky2/examples/bench_recursion.rs#L183

overview: *motivation*

shrinking proof size

```
Initial proof degree 16384 = 2^14
Degree before blinding & padding: 4028
Degree after blinding & padding: 4096
```

```
// Recursively verify the proof
let middle = recursive_proof::<F, C, C, D>(&inner, config, None)?;
let (_, _, cd) = &middle;
```



https://github.com/mir-protocol/plonky2/blob/main/plonky2/examples/bench_recursion.rs#L183

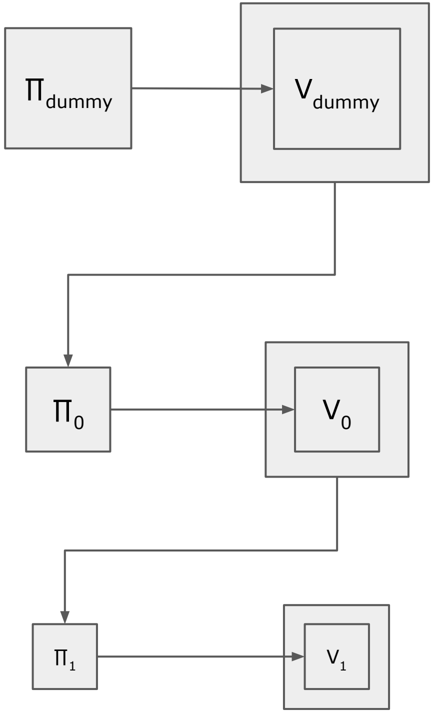
overview: *motivation*

shrinking proof size

```
Initial proof degree 16384 = 2^14
Degree before blinding & padding: 4028
Degree after blinding & padding: 4096
```

```
Single recursion proof degree 4096 = 2^12
Degree before blinding & padding: 3849
Degree after blinding & padding: 4096
```

```rust
// Add a second layer of recursion to shrink the proof size further
let outer = recursive_proof::<F, C, C, D>(&middle, config, None)?;
let (proof, vd, cd) = &outer;
```



https://github.com/mir-protocol/plonky2/blob/main/plonky2/examples/bench_recursion.rs#L183

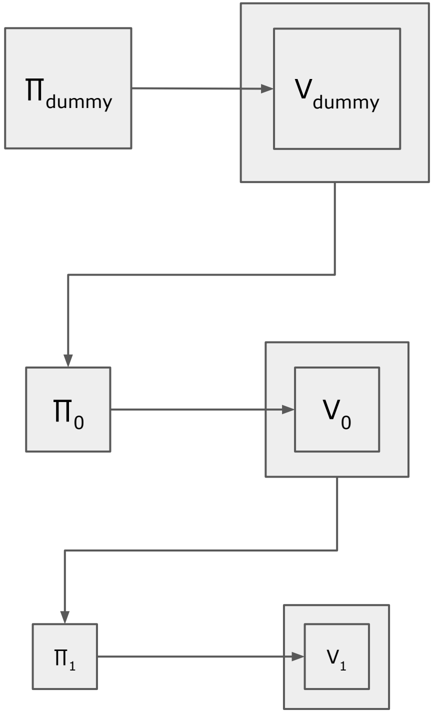
overview: *motivation*

shrinking proof size

```
Initial proof degree 16384 = 2^14
Degree before blinding & padding: 4028
Degree after blinding & padding: 4096
```

```
Single recursion proof degree 4096 = 2^12
Degree before blinding & padding: 3849
Degree after blinding & padding: 4096
```

```
Double recursion proof degree 4096 = 2^12
Proof length: 127184 bytes
0.2511s to compress proof
Compressed proof length: 115708 bytes
```



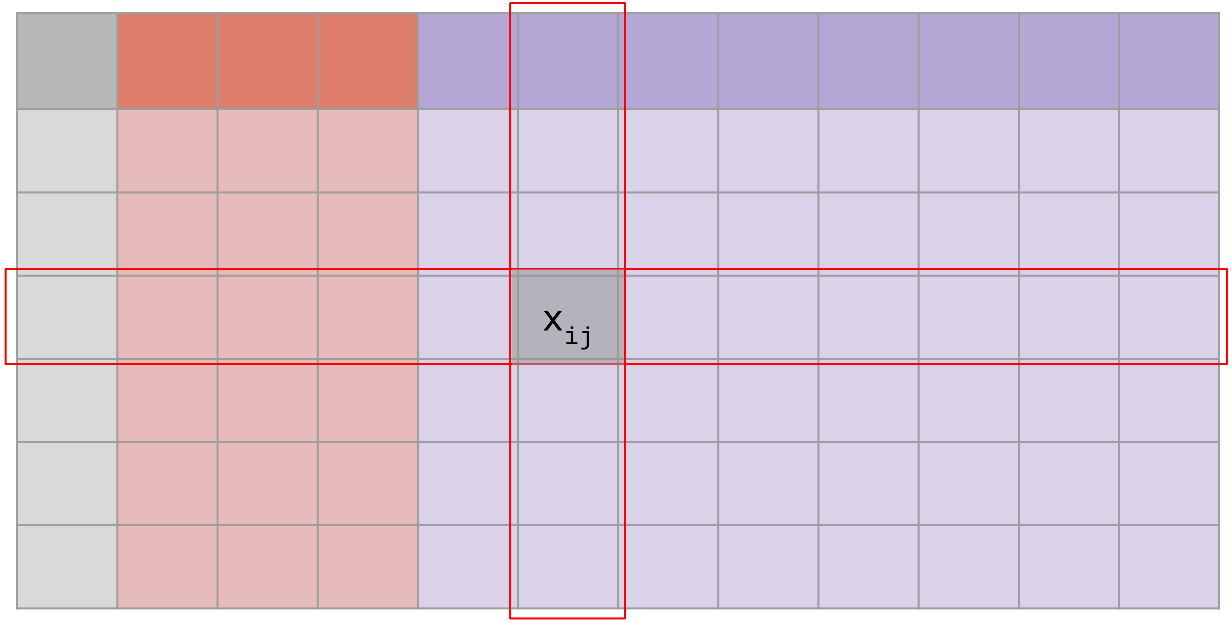
https://github.com/mir-protocol/plonky2/blob/main/plonky2/examples/bench_recursion.rs#L183

overview: *motivation*

shrinking proof size

overview: *motivation*

shrinking proof size

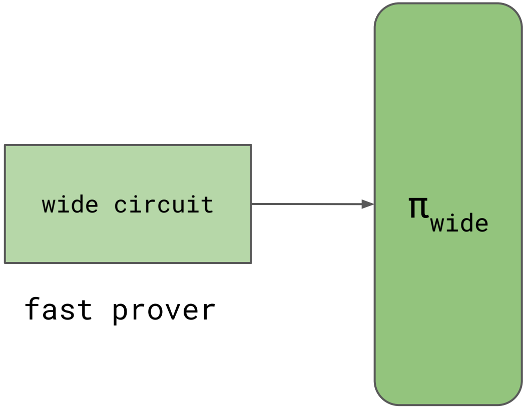


each column j corresponds to a Lagrange interpolation polynomial $p_j(X)$ evaluating to $\mathbf{p}_j(\omega^i) = \mathbf{x}_{ij}$, where ω is the n^{th} primitive root of unity.

overview: *motivation*

shrinking proof size

	fast prover	small proof / fast verifier
"wide" proof	✓	✗



fast prover

large proof

overview: *motivation*

shrinking proof size

	fast prover	small proof / fast verifier
"wide" proof	✔	❌
"narrow" proof	❌	✔

wide circuit

fast prover

π_{wide}

large proof

narrow circuit

slow prover

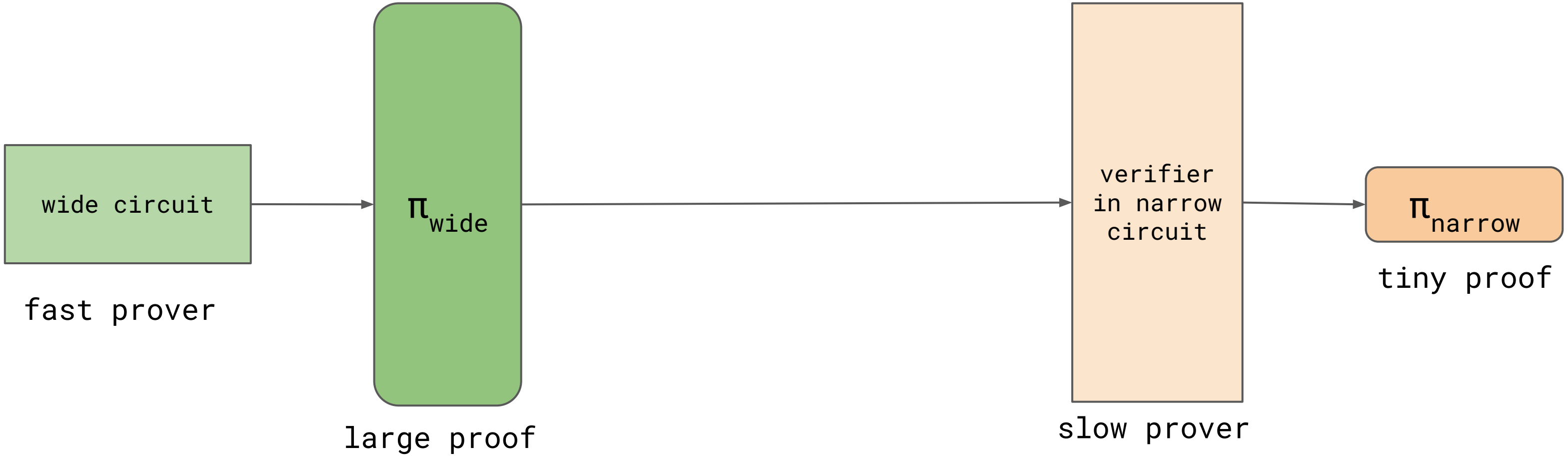
π_{narrow}

tiny proof

overview: *motivation*

shrinking proof size

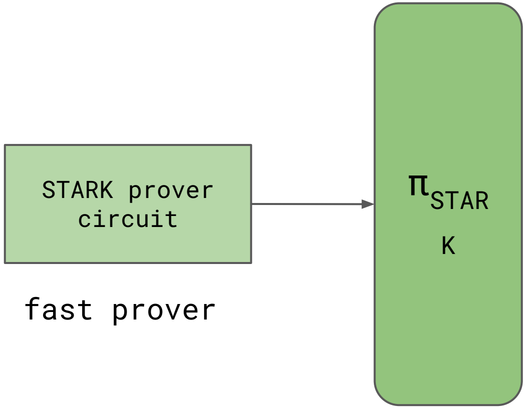
	fast prover	small proof / fast verifier
"wide" proof	✓	✗
"narrow" proof	✗	✓



overview: *motivation*

shrinking proof size

	fast prover	small proof / fast verifier
STARK	✓	✗



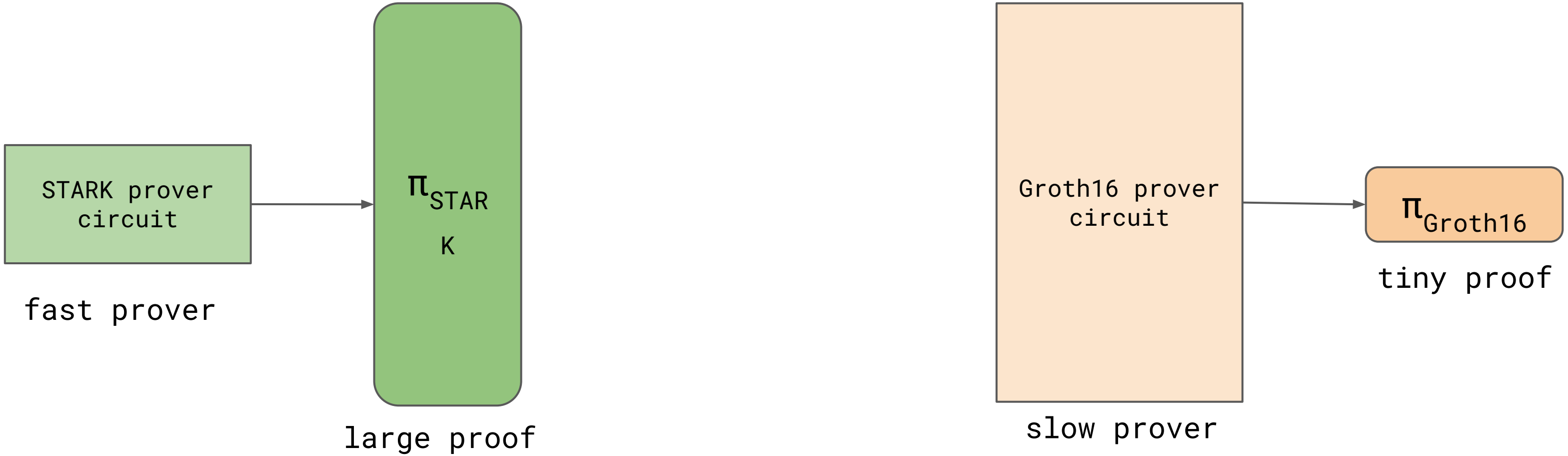
fast prover

large proof

overview: *motivation*

shrinking proof size

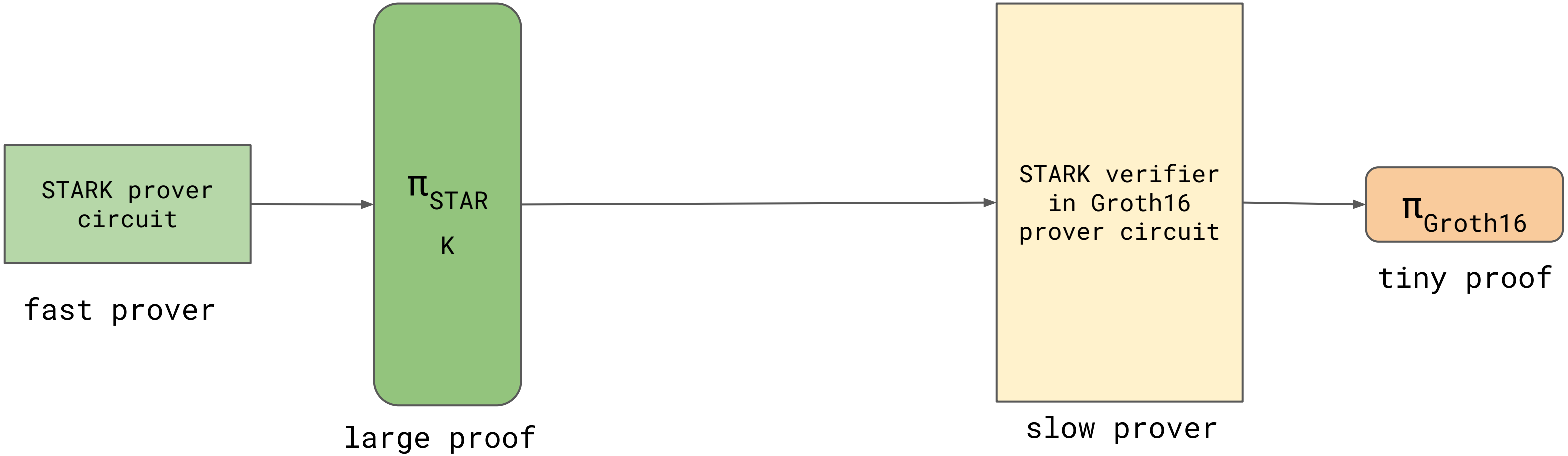
	fast prover	small proof / fast verifier
STARK	✓	✗
Groth16	✗	✓



overview: *motivation*

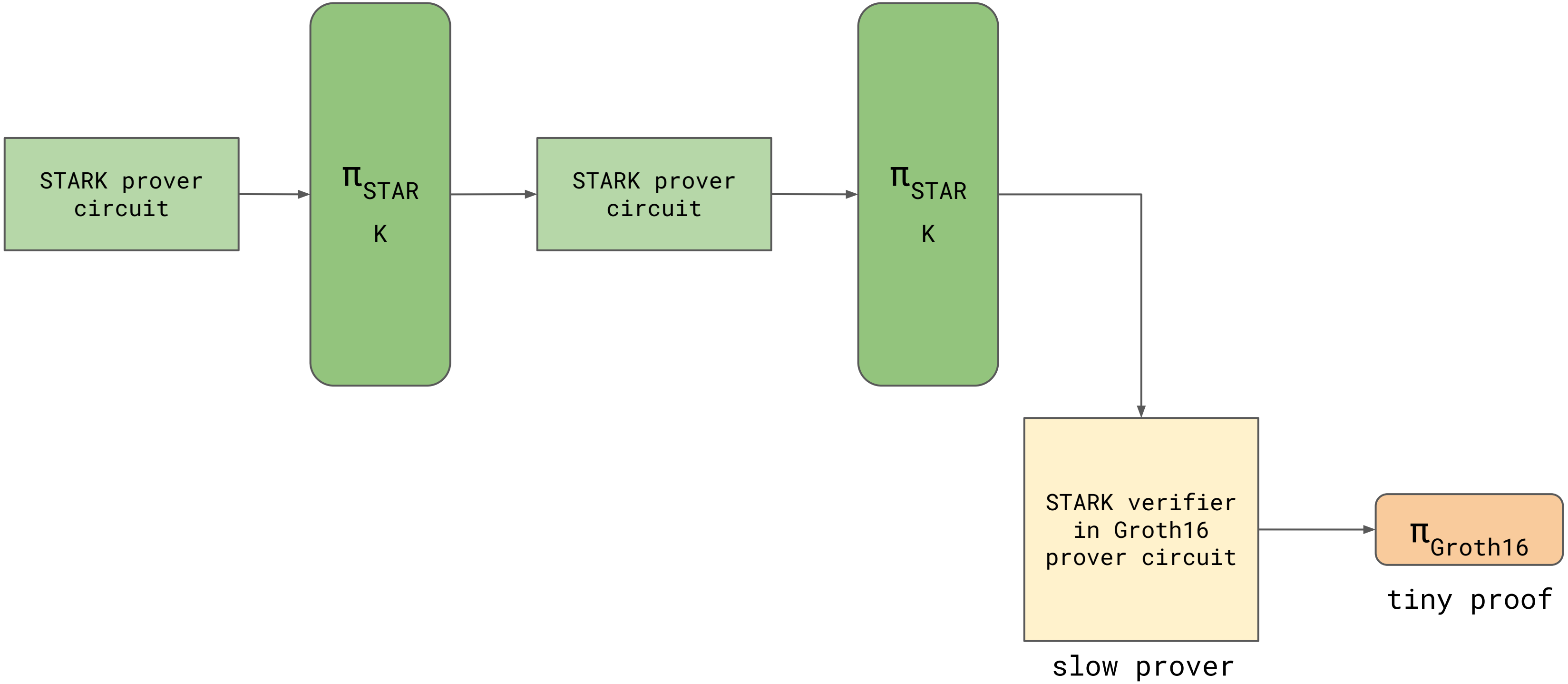
shrinking proof size

	fast prover	small proof / fast verifier
STARK	✓	✗
Groth16	✗	✓



overview: *motivation*

shrinking proof size



STARK prover circuit

π_{STARK}

STARK prover circuit

π_{STARK}

STARK verifier in Groth16 prover circuit

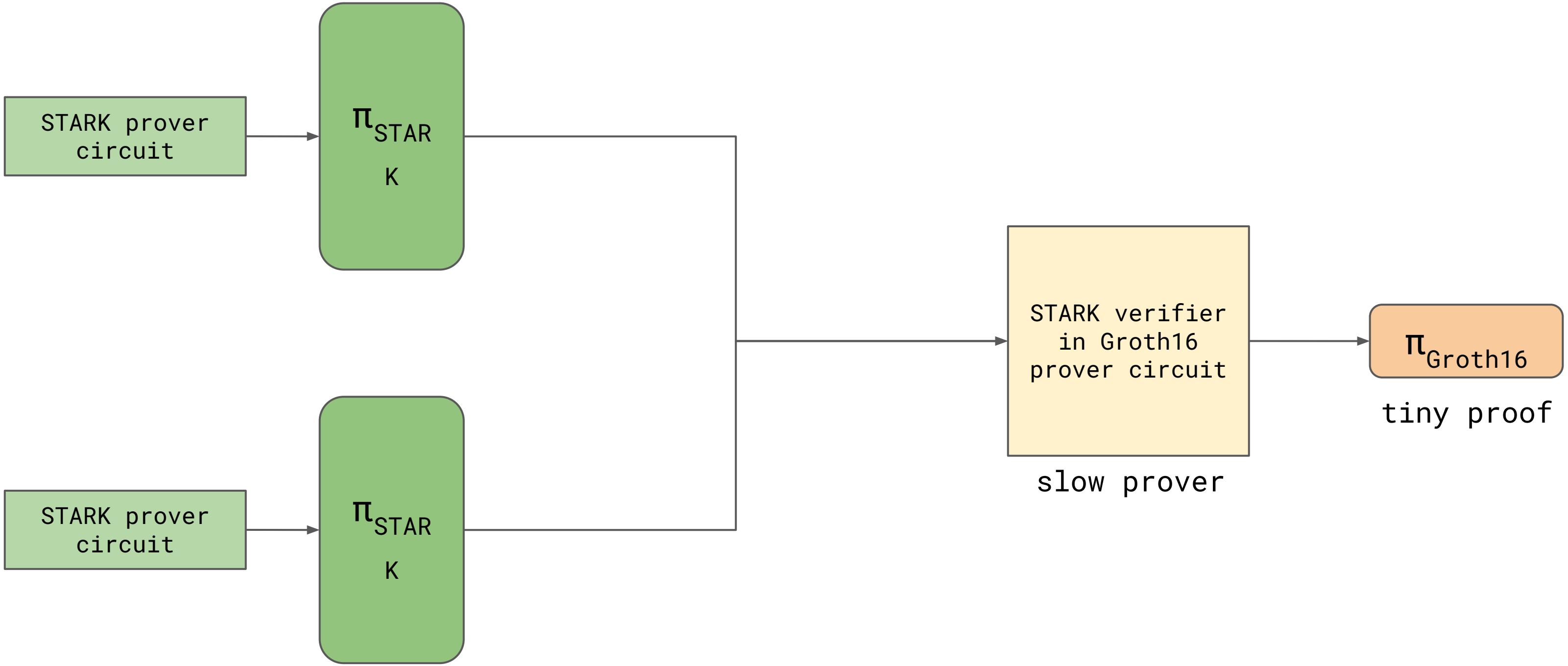
slow prover

π_{Groth16}

tiny proof

overview: *motivation*

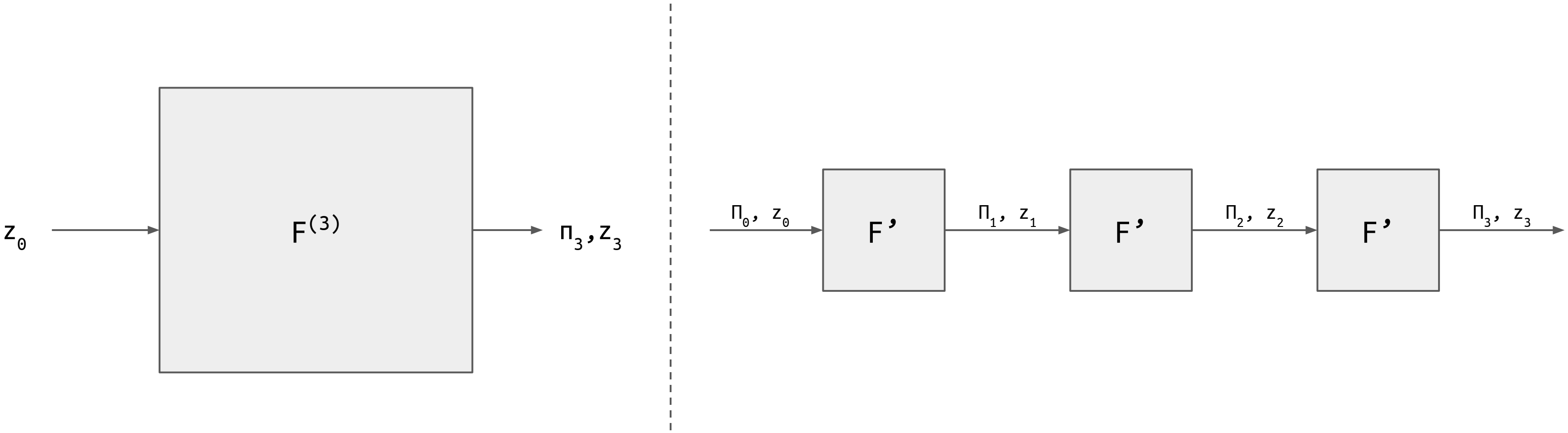
shrinking proof size



overview: *motivation*

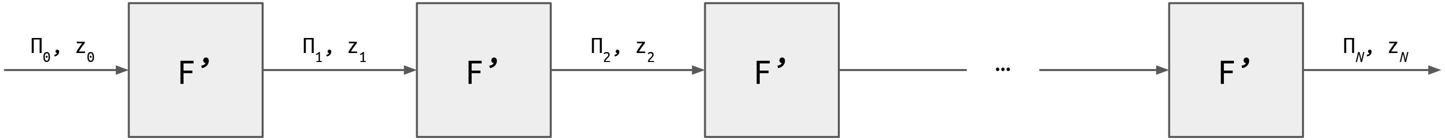
incrementally verifiable computation

break large circuit into N *repetitions of smaller circuit: reduces prover space complexity*



overview: *motivation*

incrementally verifiable computation



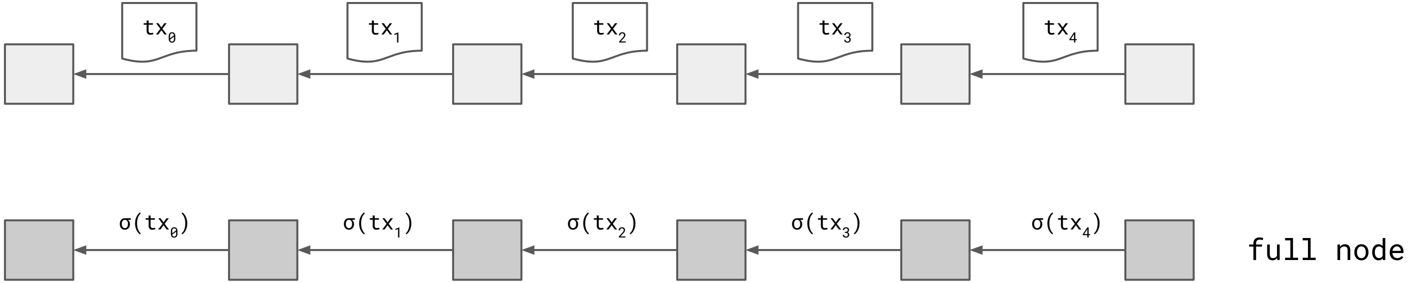
applications:

- verify chain of *N* blocks with a single proof (e.g. Mina Protocol)
- verify *N* steps of program in virtual machine (e.g. RISC Zero)
- verify inference of an *N*-layer neural network (e.g. Zator 🐊)

overview: *motivation*

e.g. succinct blockchain

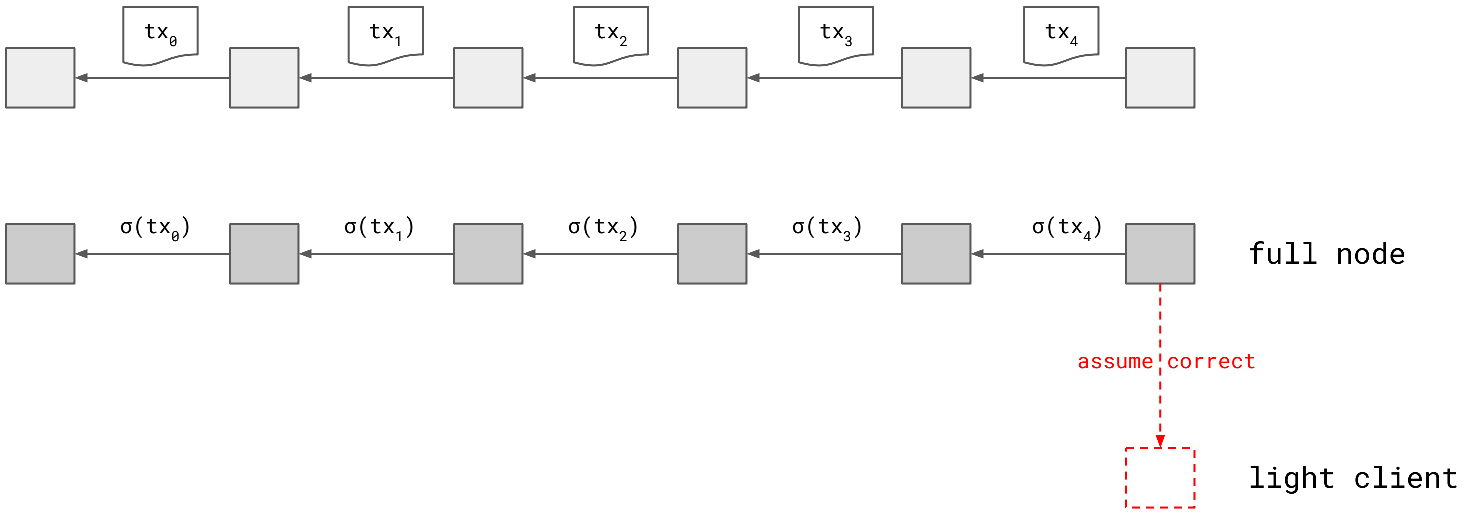
a blockchain in which each block can be verified in **constant time** regardless of the number of prior blocks in the history



overview: *motivation*

e.g. succinct blockchain

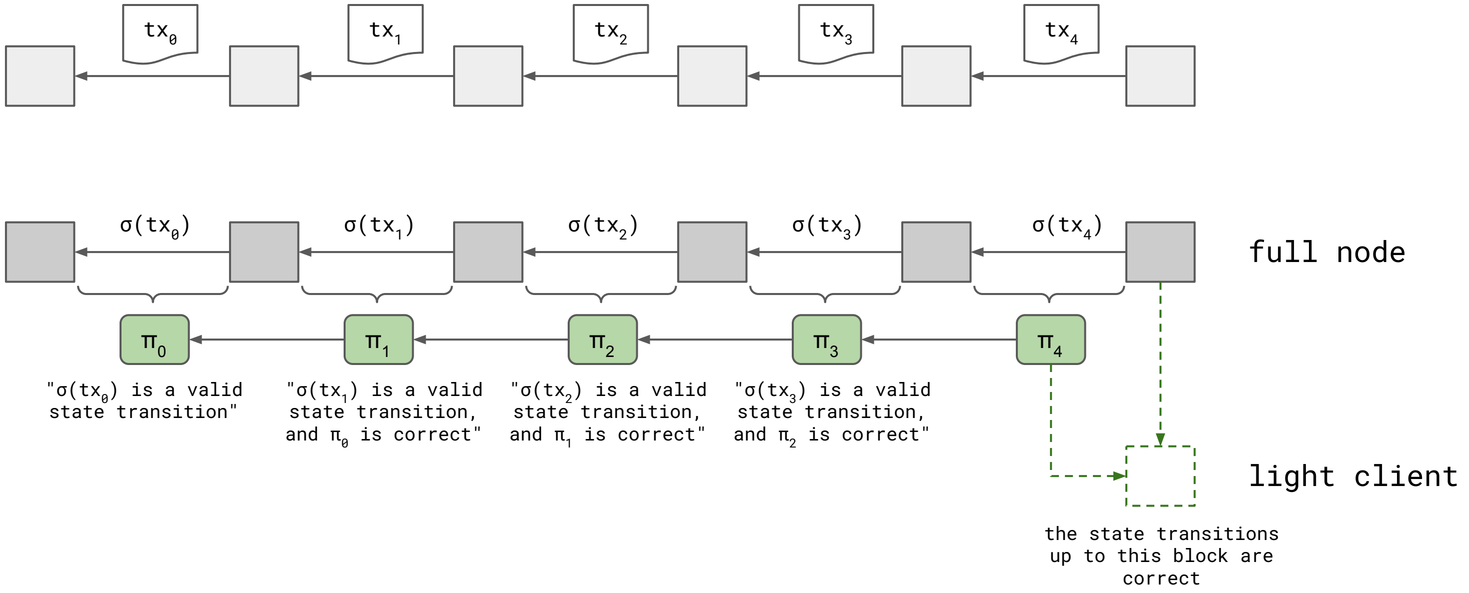
a blockchain in which each block can be verified in **constant time** regardless of the number of prior blocks in the history



overview: *motivation*

e.g. succinct blockchain

a blockchain in which each block can be verified in **constant time** regardless of the number of prior blocks in the history

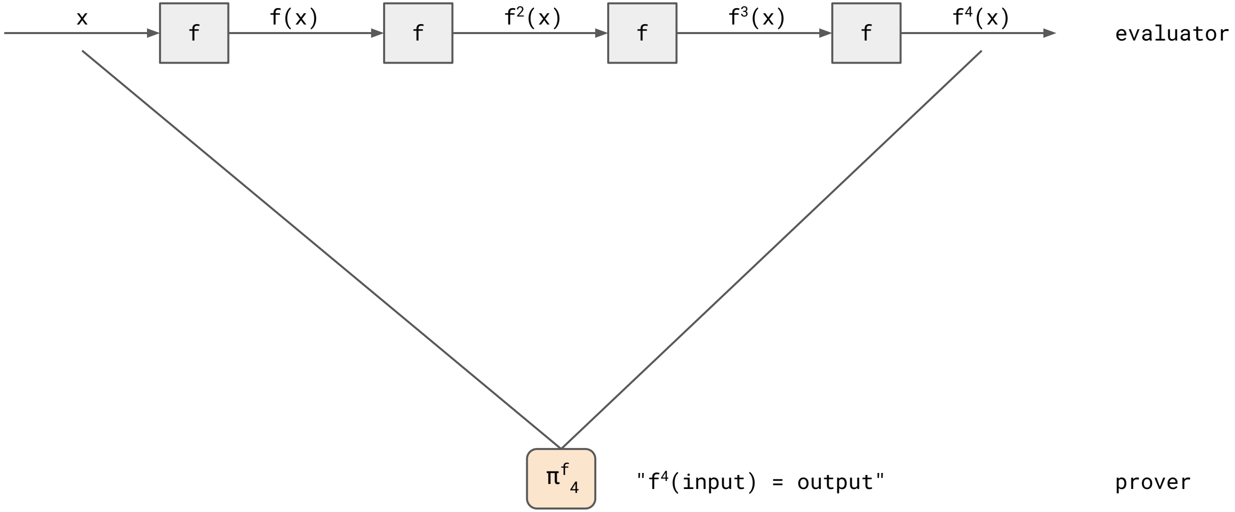


tx_0
tx_1
tx_2
tx_3
tx_4
$\sigma(tx_0)$
$\sigma(tx_1)$
$\sigma(tx_2)$
$\sigma(tx_3)$
$\sigma(tx_4)$
full node
π_0
π_1
π_2
π_3
π_4
"$\sigma(tx_0)$ is a valid state transition"
"$\sigma(tx_1)$ is a valid state transition, and π_0 is correct"
"$\sigma(tx_2)$ is a valid state transition, and π_1 is correct"
"$\sigma(tx_3)$ is a valid state transition, and π_2 is correct"
light client
the state transitions up to this block are correct

overview: *motivation*

e.g. parallelising the VDF prover

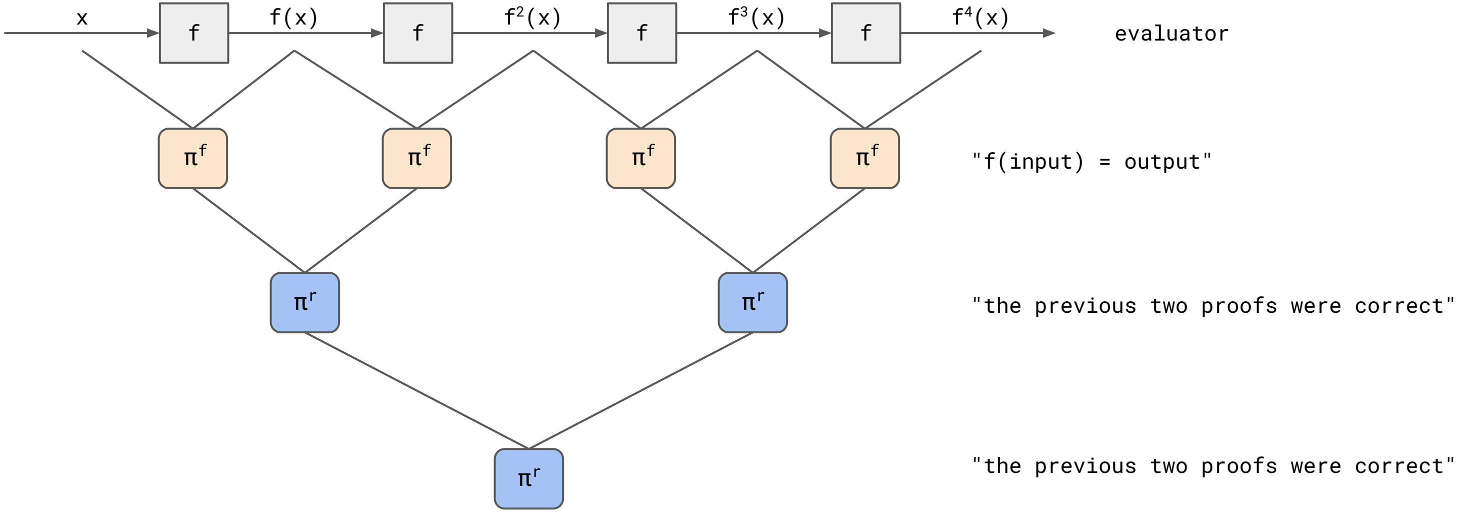
verifiable delay function [BBBF18]: a sequential computation that is slow to compute but efficient to verify



overview: *motivation*

e.g. parallelising the VDF prover

verifiable delay function [BBBF18]: a sequential computation that is slow to compute but efficient to verify



overview: *motivation*

proof-carrying data

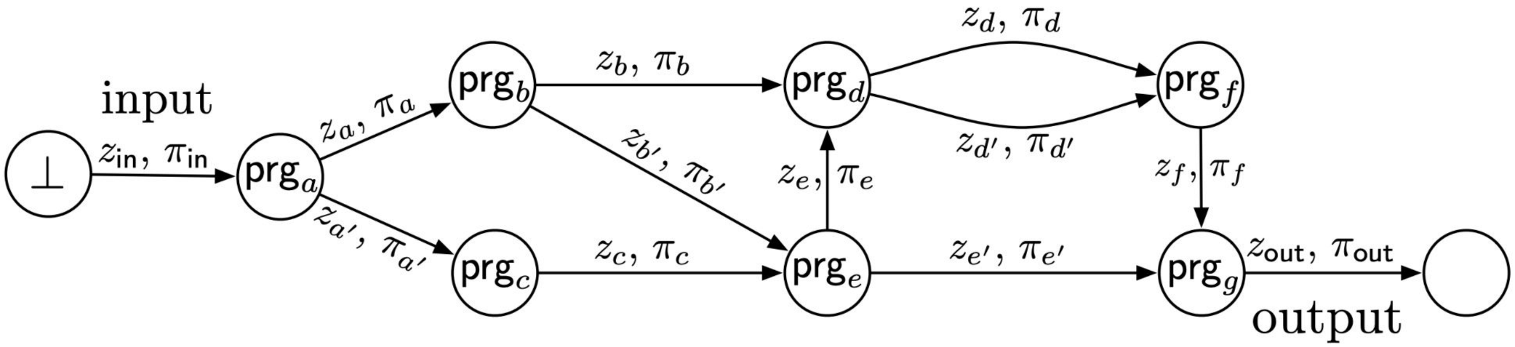
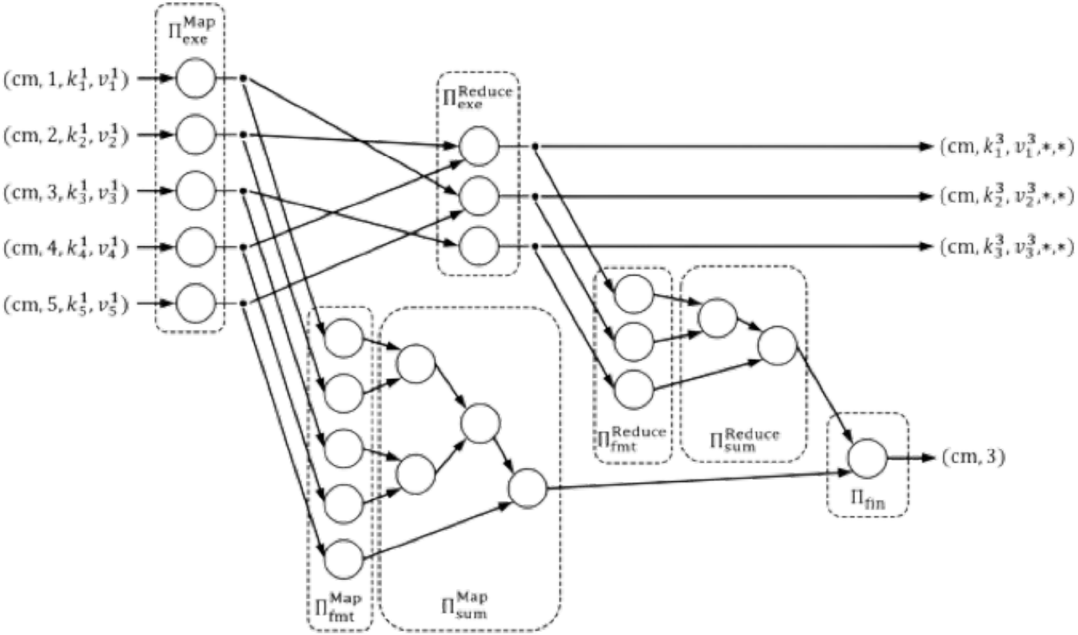


Figure 5: Example of an *augmented* distributed computation transcript. Programs are denoted by **prg**'s, data by z's, and proof strings by π's. The corresponding (non-augmented) distributed computation transcript is with the proof strings omitted.

image from https://people.eecs.berkeley.edu/~alexch/docs/CT10.pdf

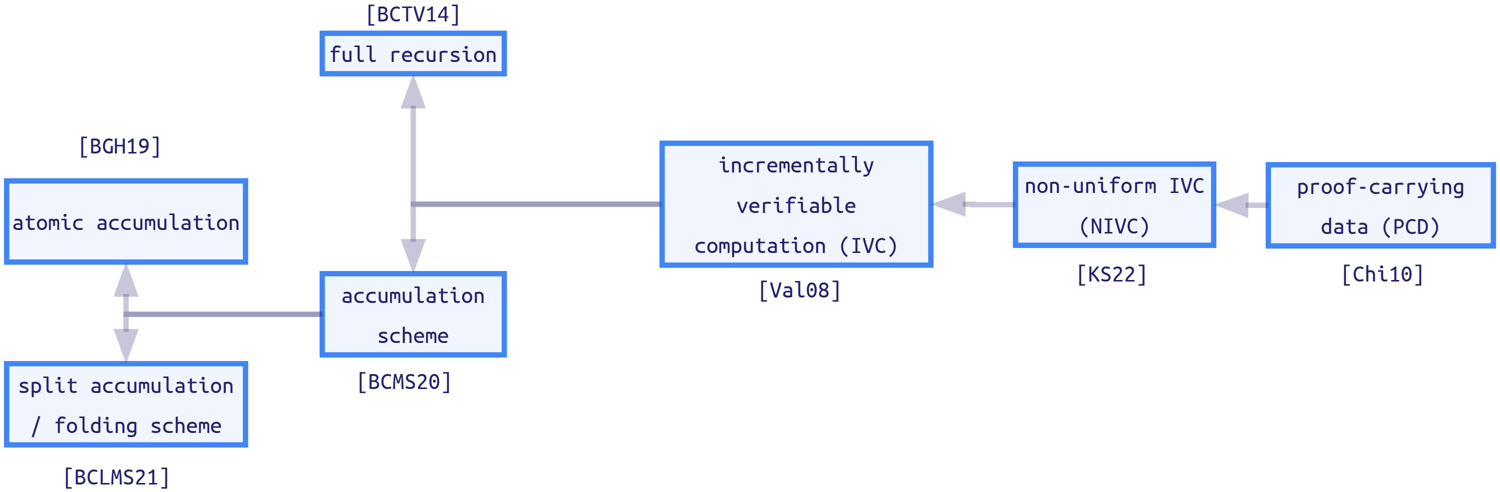
overview: *motivation*

proof-carrying data

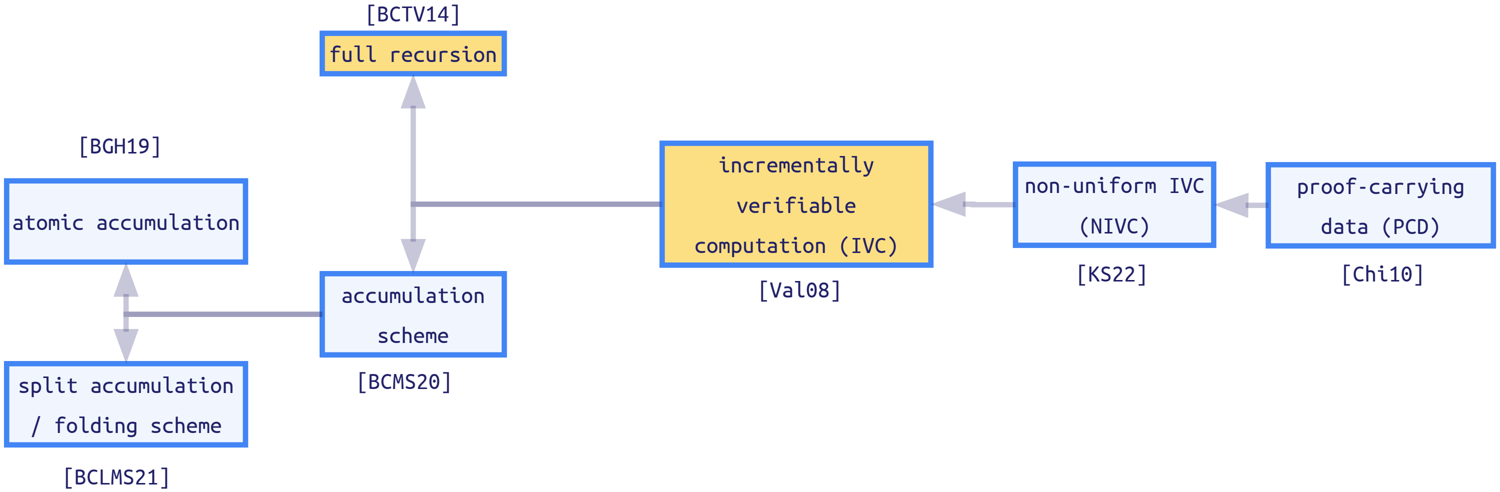

$\Pi^{\text{Map}}_{\text{exe}}$
$(\text{cm}, 1, k_1^1, v_1^1)$
$(\text{cm}, 2, k_2^1, v_2^1)$
$(\text{cm}, 3, k_3^1, v_3^1)$
$(\text{cm}, 4, k_4^1, v_4^1)$
$(\text{cm}, 5, k_5^1, v_5^1)$
$\Pi^{\text{Reduce}}_{\text{exe}}$
$(\text{cm}, k_1^3, v_1^3, \star, \star)$
$(\text{cm}, k_2^3, v_2^3, \star, \star)$
$(\text{cm}, k_3^3, v_3^3, \star, \star)$
$\Pi^{\text{Reduce}}_{\text{fmt}}$
$\Pi^{\text{Reduce}}_{\text{sum}}$
$(\text{cm}, 3)$
Π_{fin}
$\Pi^{\text{Map}}_{\text{fmt}}$
$\Pi^{\text{Map}}_{\text{sum}}$

e.g. MapReduce [CTV15]

overview: *constructions*

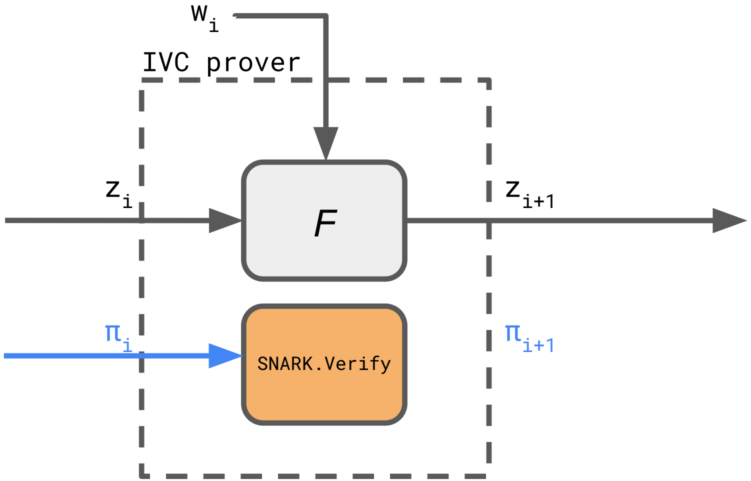


overview: *constructions*



overview: *constructions*

full recursion

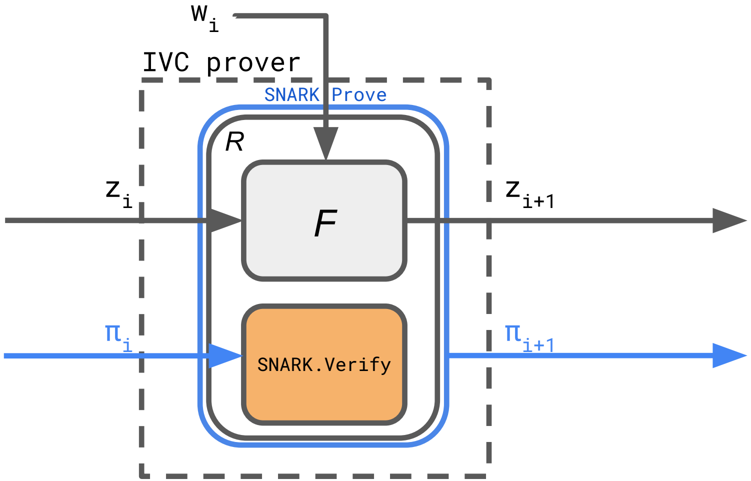


$z_{i+1} = F(z_i; w_i)$

$V(\pi_i) = 1$

overview: *constructions*

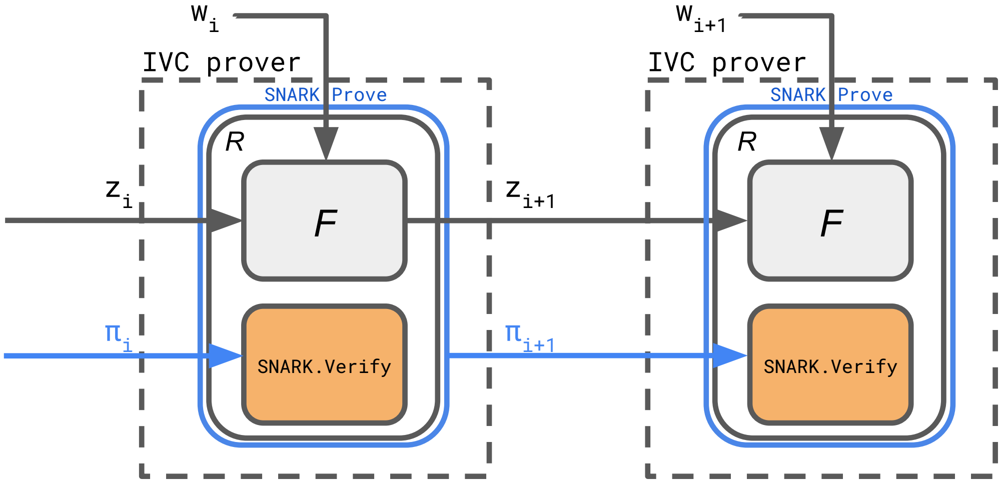
full recursion



$$\left.\begin{array}{l} z_{i+1} = F(z_i;\ w_i) \\ V(\pi_i) = 1 \end{array}\right\} \pi_{i+1}$$

overview: *constructions*

full recursion

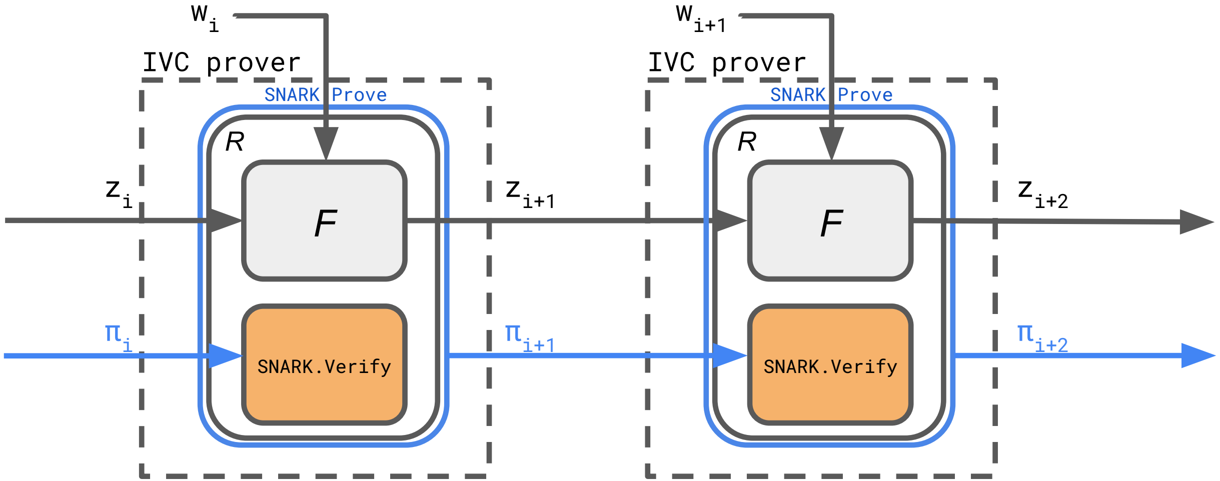


$$\left.\begin{array}{l} z_{i+1} = F(z_i;\ w_i) \\ V(\pi_i) = 1 \end{array}\right\} \pi_{i+1}$$

$$z_{i+2} = F(z_{i+1};\ w_{i+1})$$
$$V(\pi_{i+1}) = 1$$

overview: *constructions*

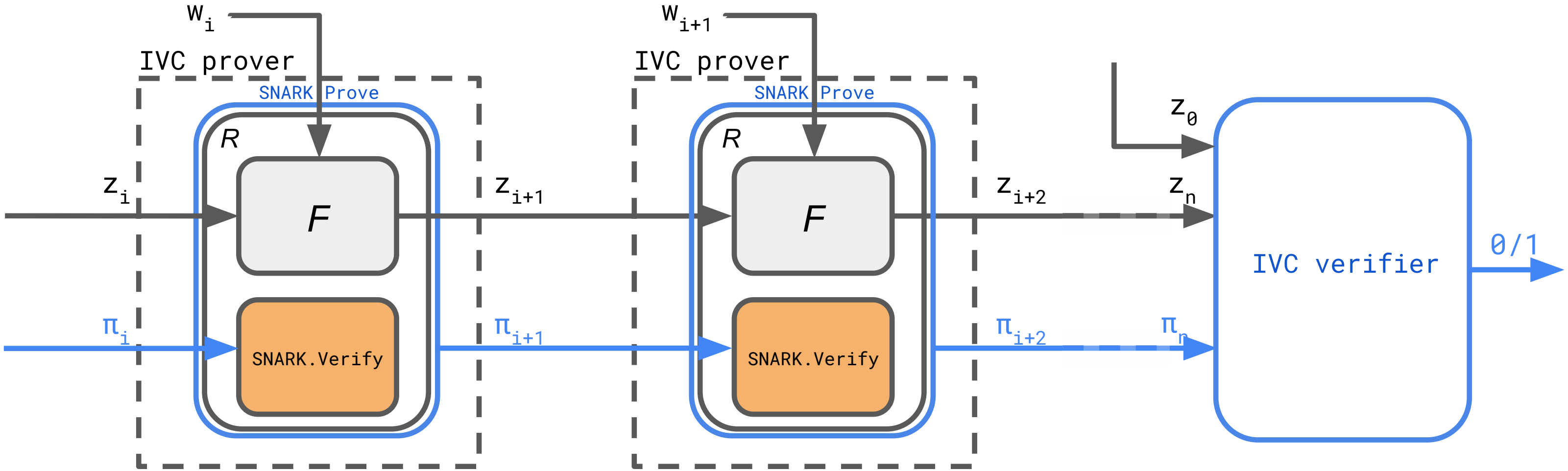
full recursion



$$\left.\begin{array}{l} z_{i+1} = F(z_i;\ w_i) \\ V(\pi_i) = 1 \end{array}\right\} \pi_{i+1} \qquad \left.\begin{array}{l} z_{i+2} = F(z_{i+1};\ w_{i+1}) \\ V(\pi_{i+1}) = 1 \end{array}\right\} \pi_{i+2}$$

overview: *constructions*

full recursion

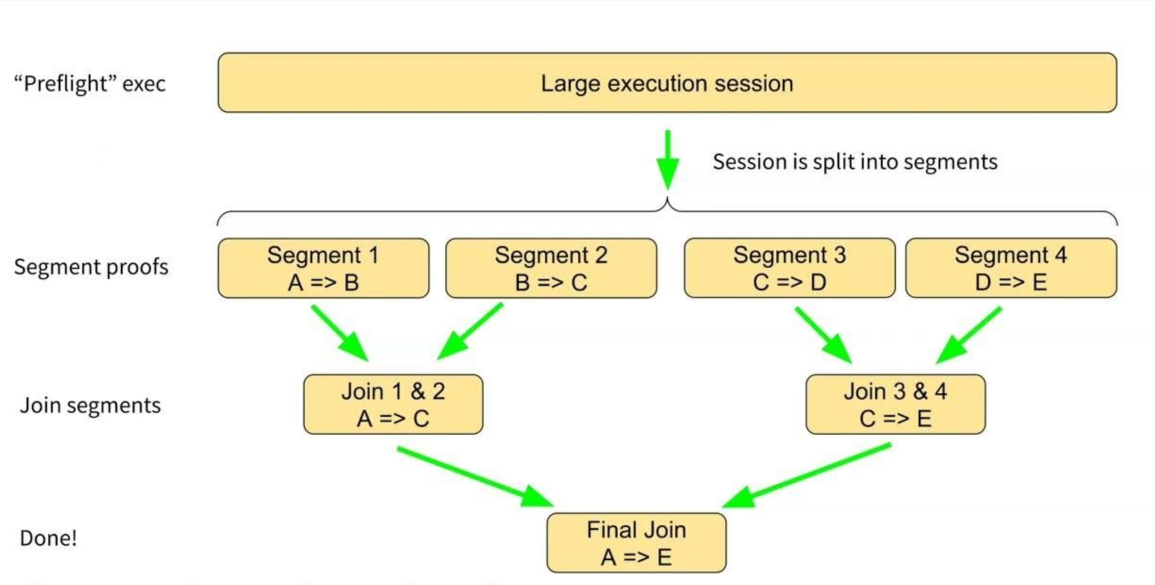


$$\left.\begin{array}{l} z_{i+1} = F(z_i;\ w_i) \\ V(\pi_i) = 1 \end{array}\right\} \pi_{i+1}$$

$$\left.\begin{array}{l} z_{i+2} = F(z_{i+1};\ w_{i+1}) \\ V(\pi_{i+1}) = 1 \end{array}\right\} \pi_{i+2}$$

overview: *constructions*

full recursion: small-field FRI

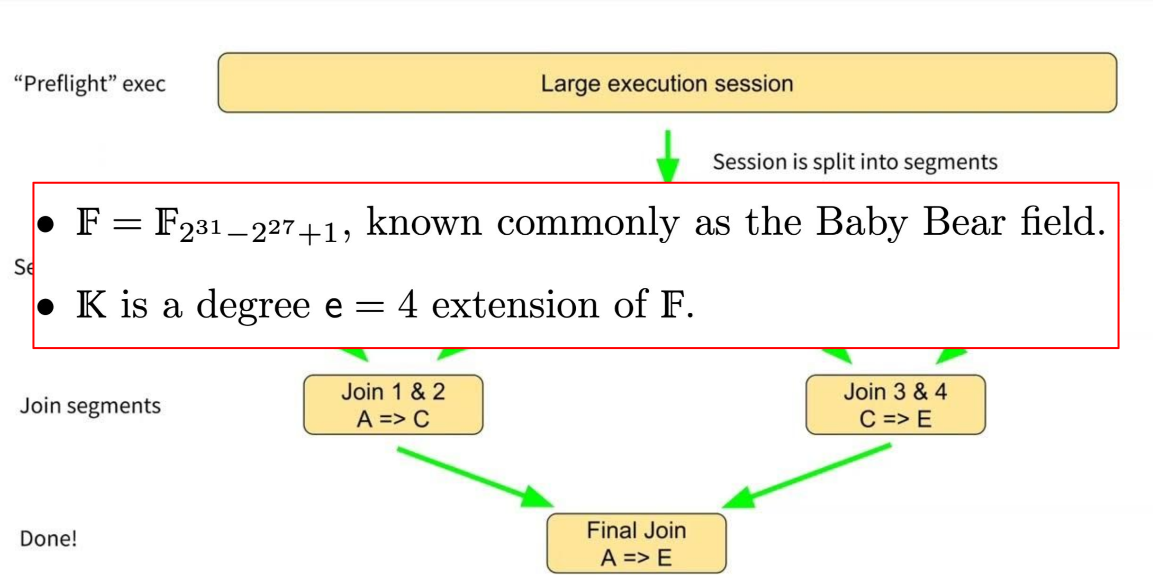


Segmented execution and proving

image from https://www.risczero.com/news/continuations

overview: *constructions*

full recursion: small-field FRI



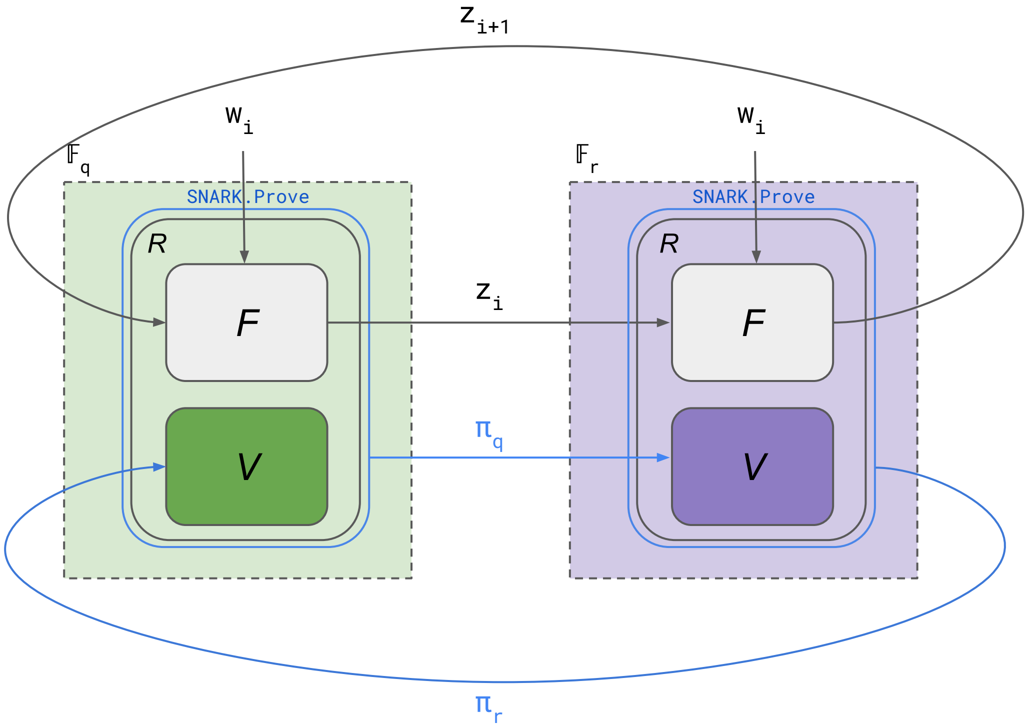
Segmented execution and proving

- $\mathbb{F} = \mathbb{F}_{2^{31}-2^{27}+1}$, known commonly as the Baby Bear field.
- $\mathbb{K}$ is a degree $\mathsf{e} = 4$ extension of $\mathbb{F}$.

image from https://www.risczero.com/news/continuations

overview: *constructions*

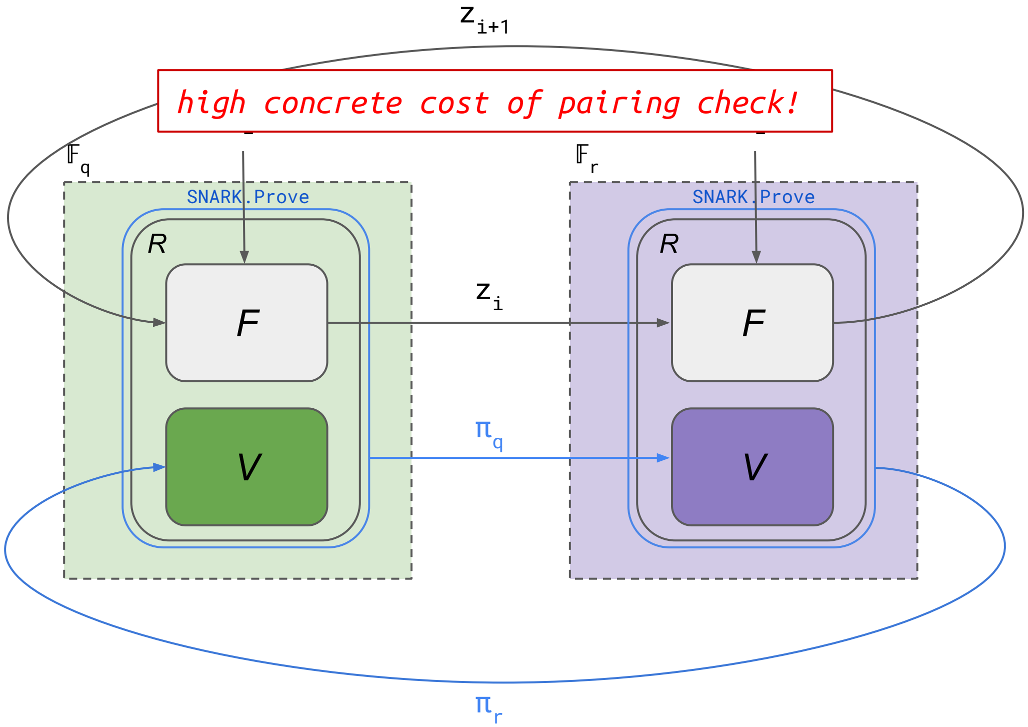
full recursion: pairings over a cycle of elliptic curves [BCTV14]



e.g. MNT4/6 curves

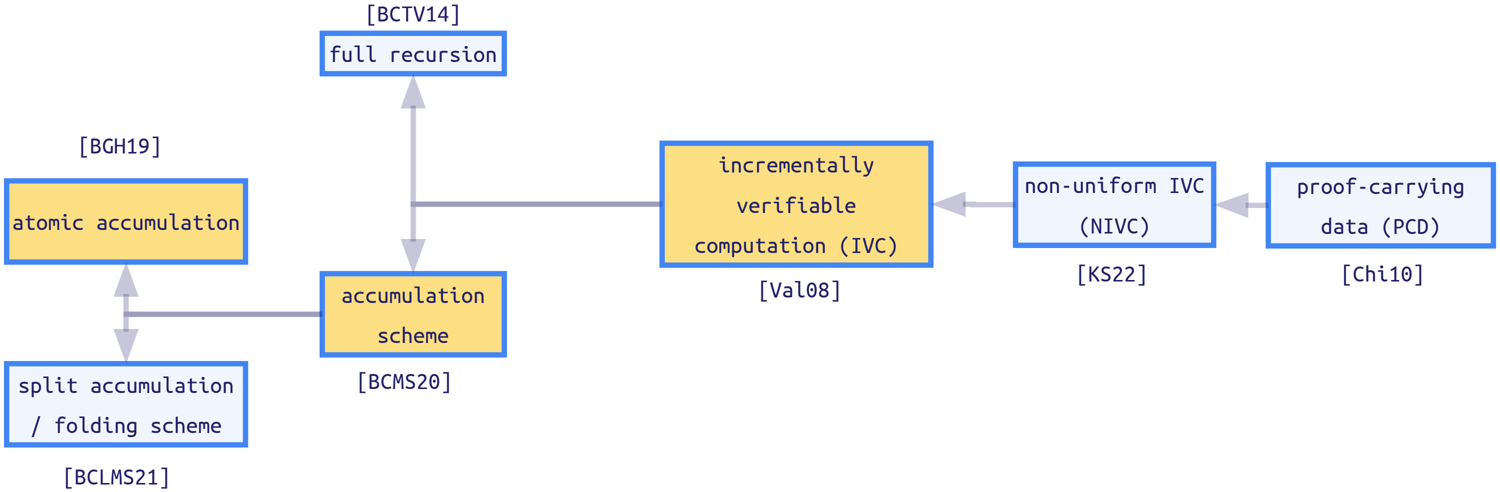
overview: *constructions*

full recursion: pairings over a cycle of elliptic curves [BCTV14]



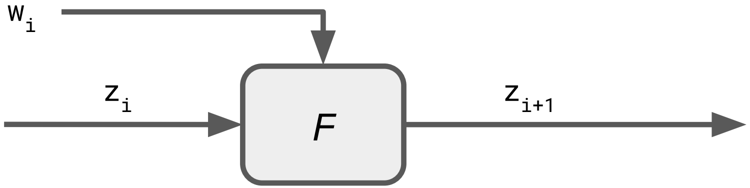
e.g. MNT4/6 curves

overview: *constructions*



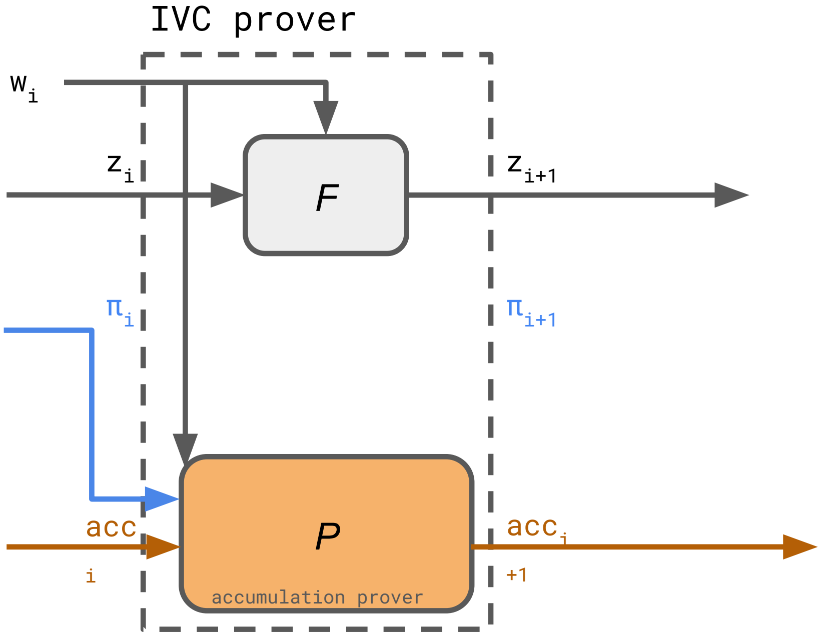
overview: *constructions*

atomic accumulation



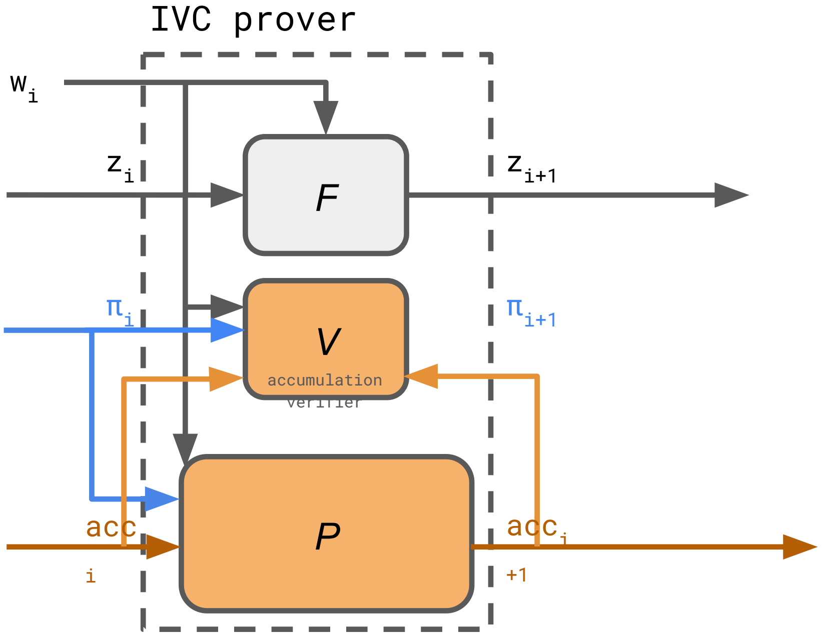
overview: *constructions*

atomic accumulation



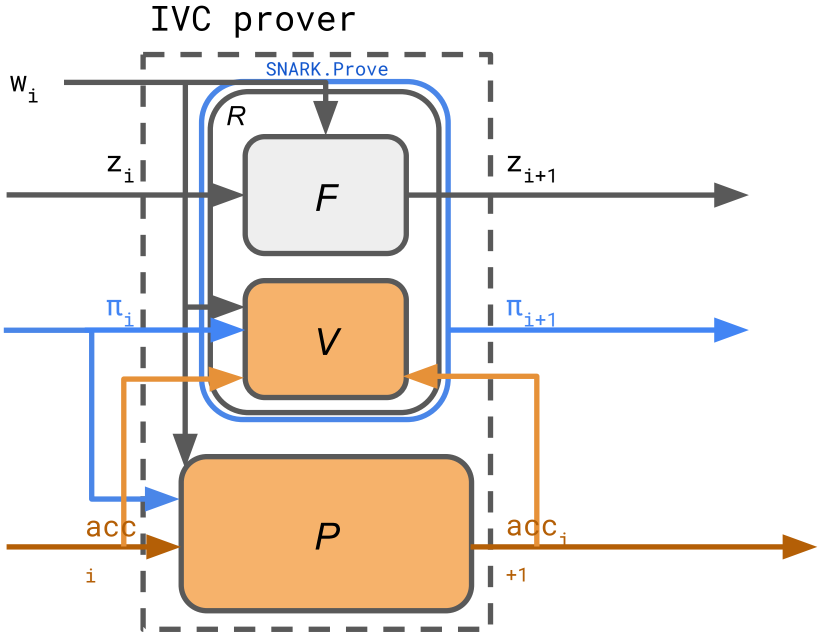
overview: *constructions*

atomic accumulation



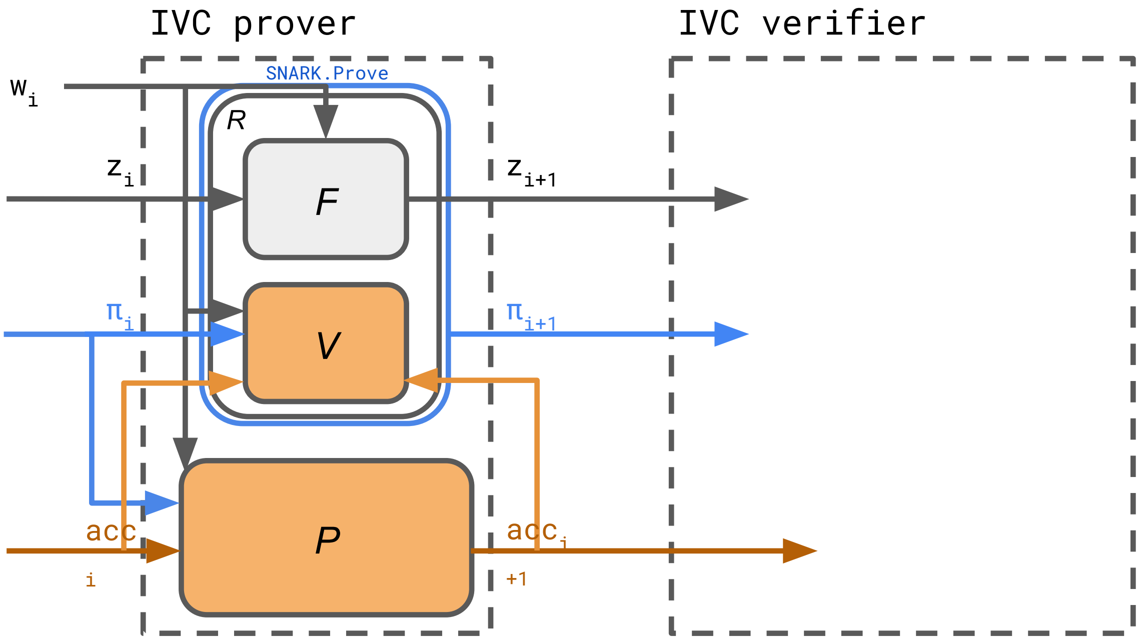
overview: *constructions*

atomic accumulation



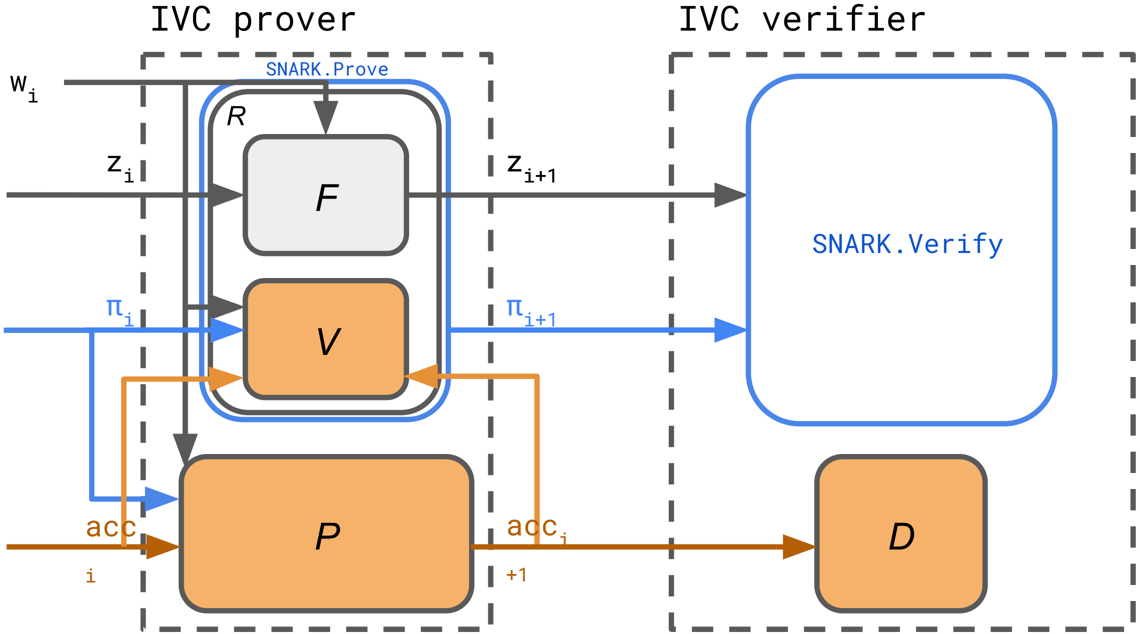
overview: *constructions*

atomic accumulation

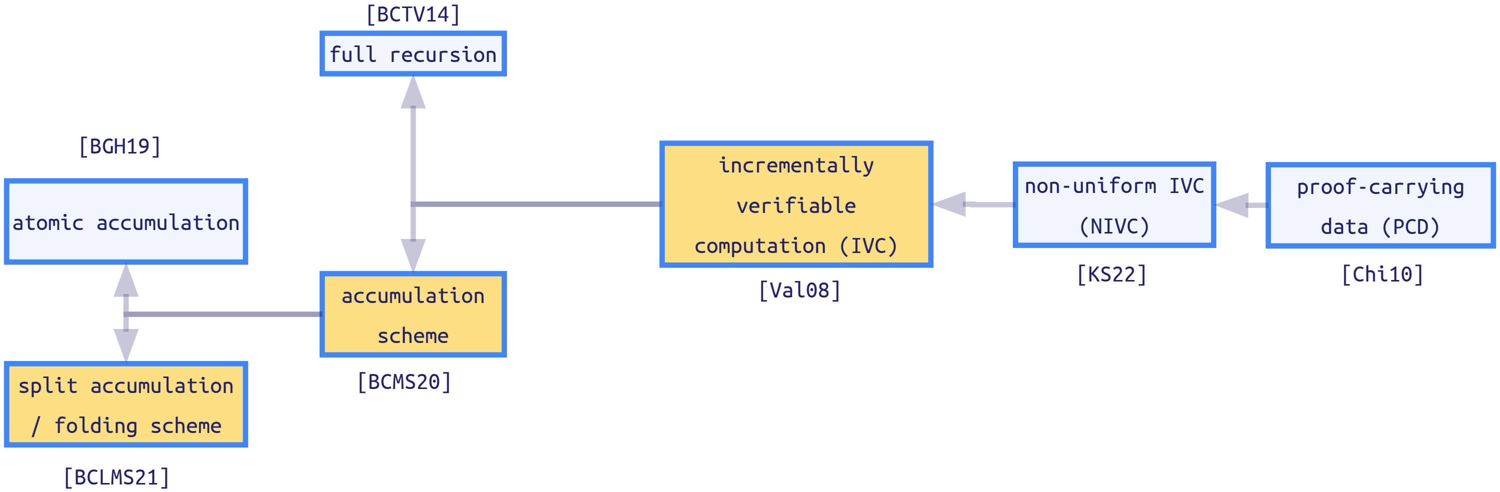


overview: *constructions*

atomic accumulation

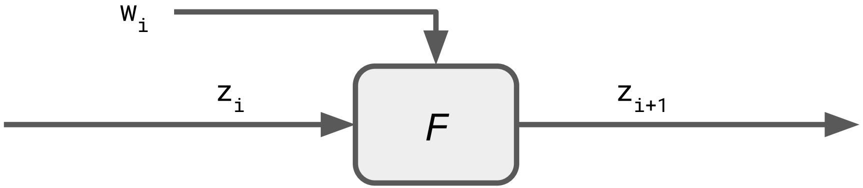


overview: *constructions*



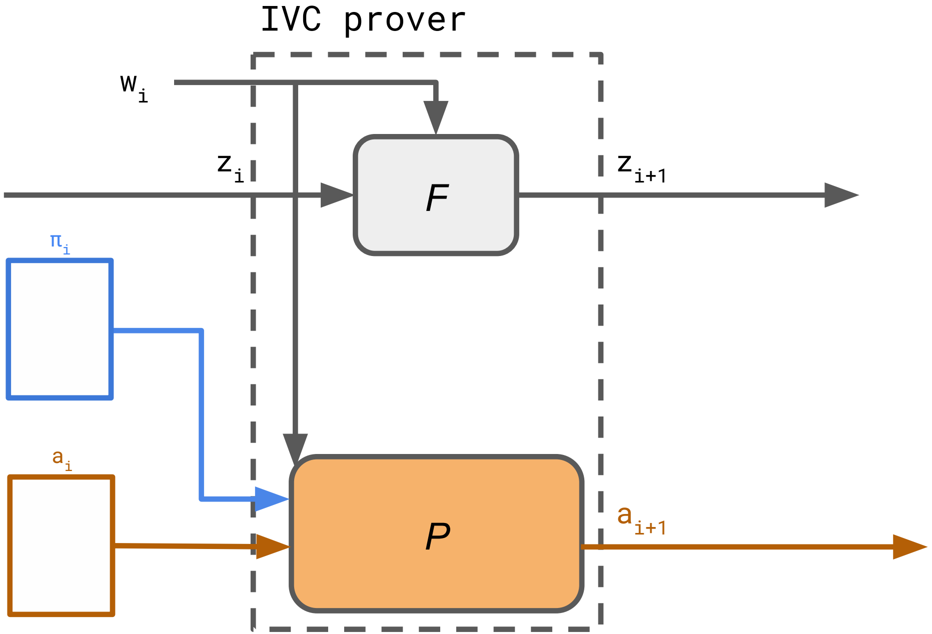
overview: *constructions*

split accumulation / folding



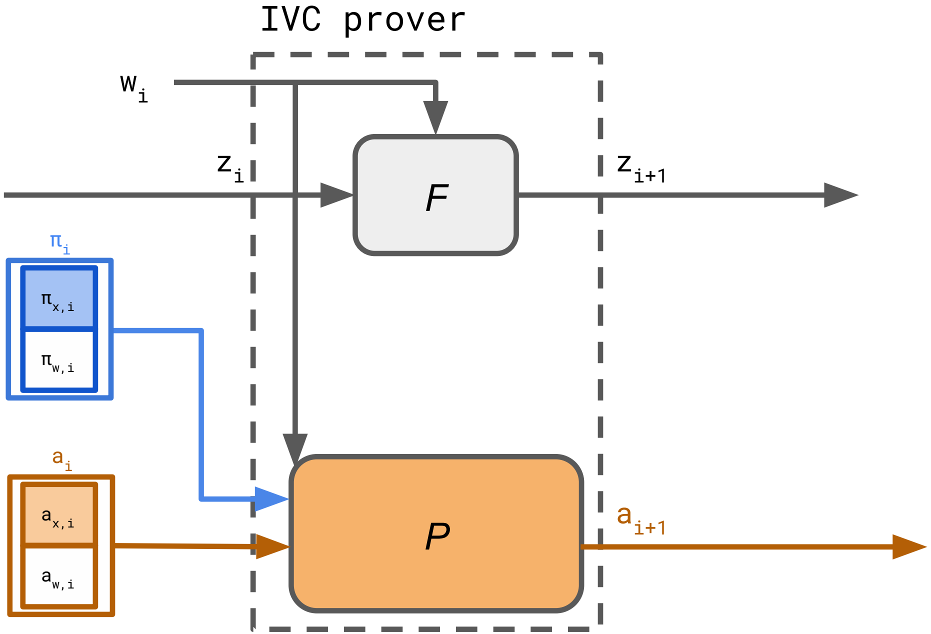
overview: *constructions*

split accumulation / folding



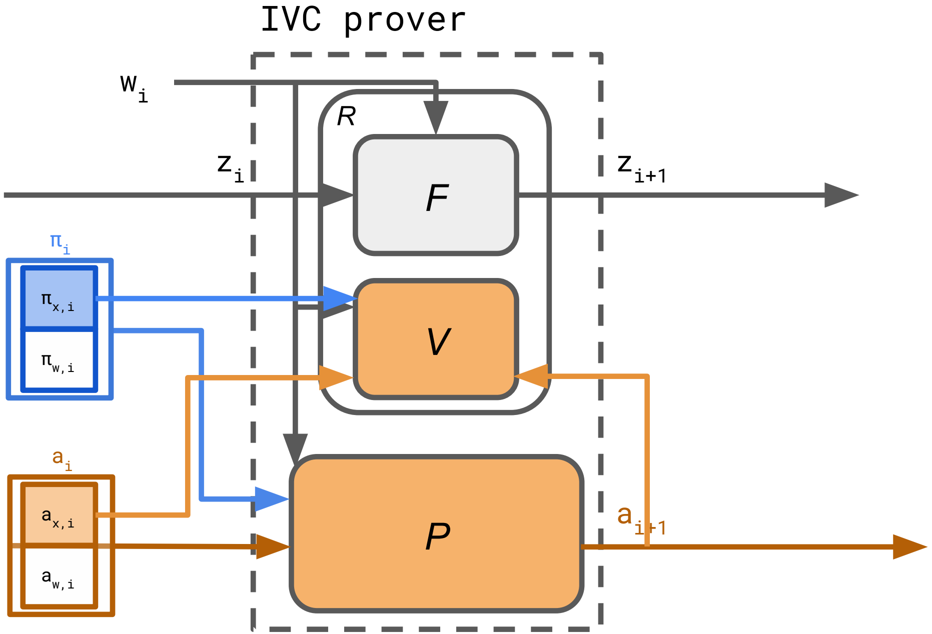
overview: *constructions*

split accumulation / folding



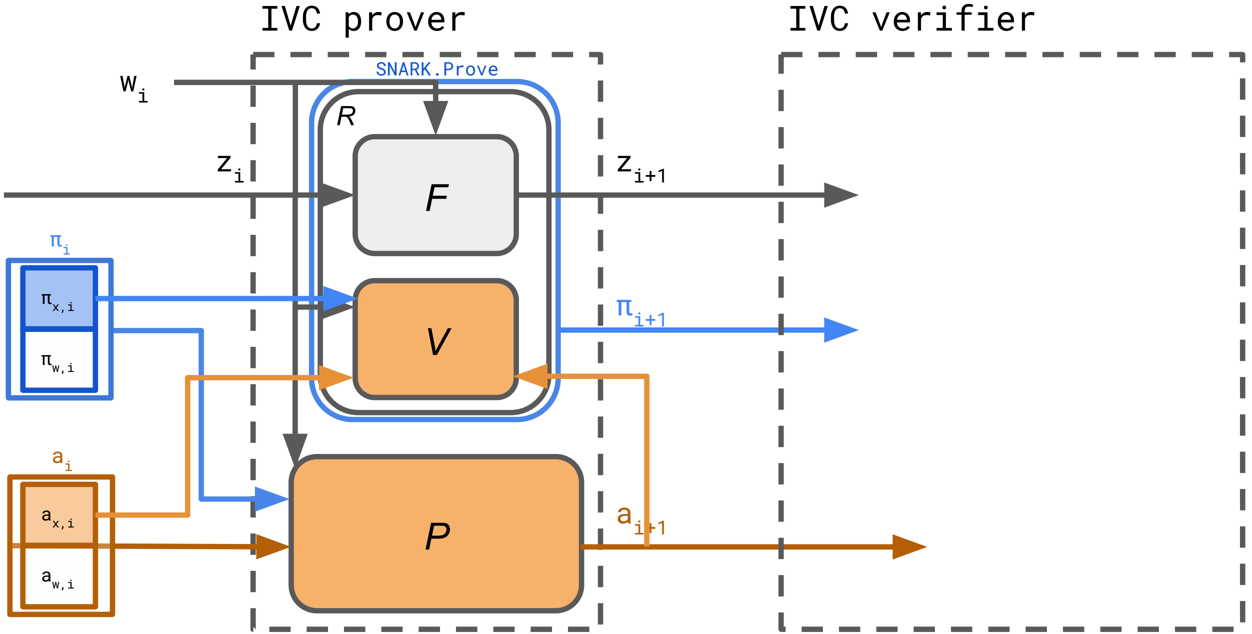
overview: *constructions*

split accumulation / folding



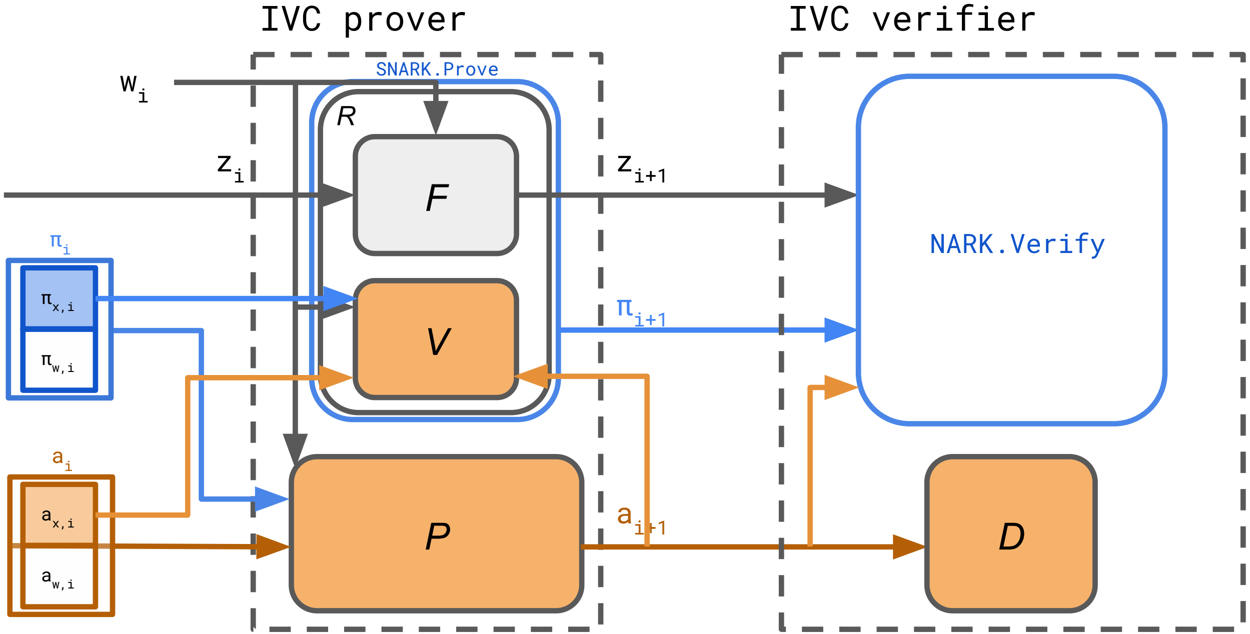
overview: *constructions*

split accumulation / folding



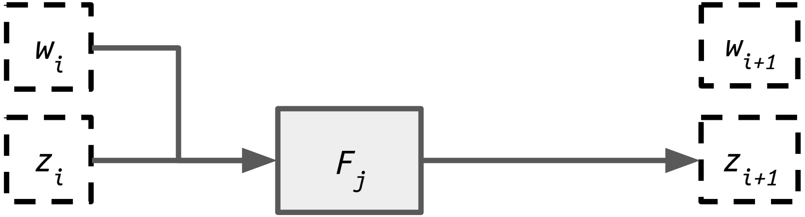
overview: *constructions*

split accumulation / folding



overview: *constructions*

non-uniform IVC (NIVC)



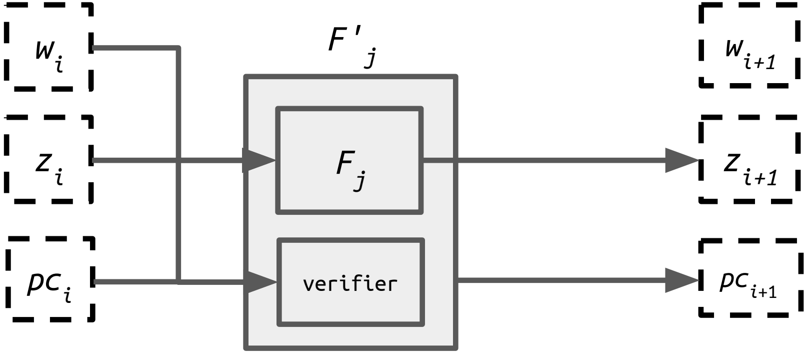
$F_j(w_i, z_i) = z_{i+1}$

overview: *constructions*

non-uniform IVC (NIVC)

verifier:

- $\varphi(w_i, z_i, pc_i) = pc_{i+1}$

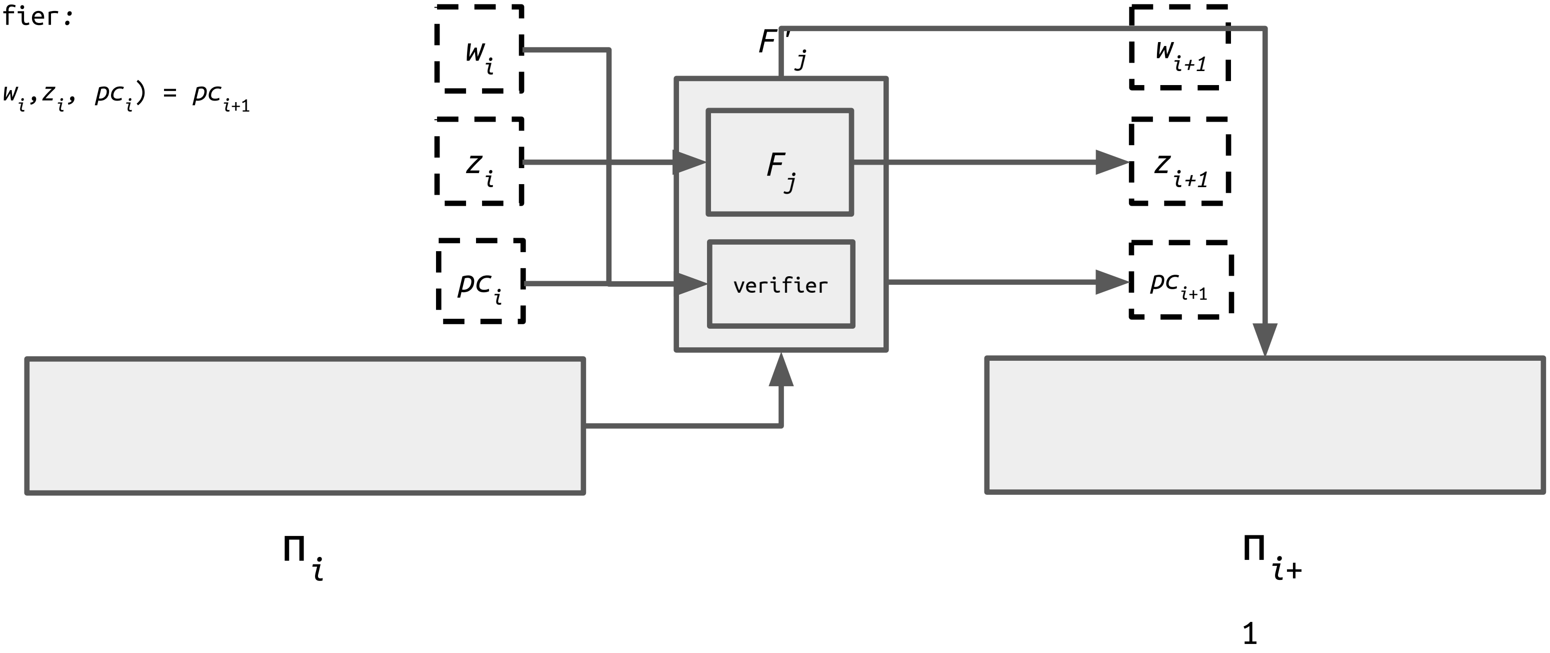


overview: *constructions*

non-uniform IVC (NIVC)

verifier:

- $\varphi(w_i, z_i, pc_i) = pc_{i+1}$

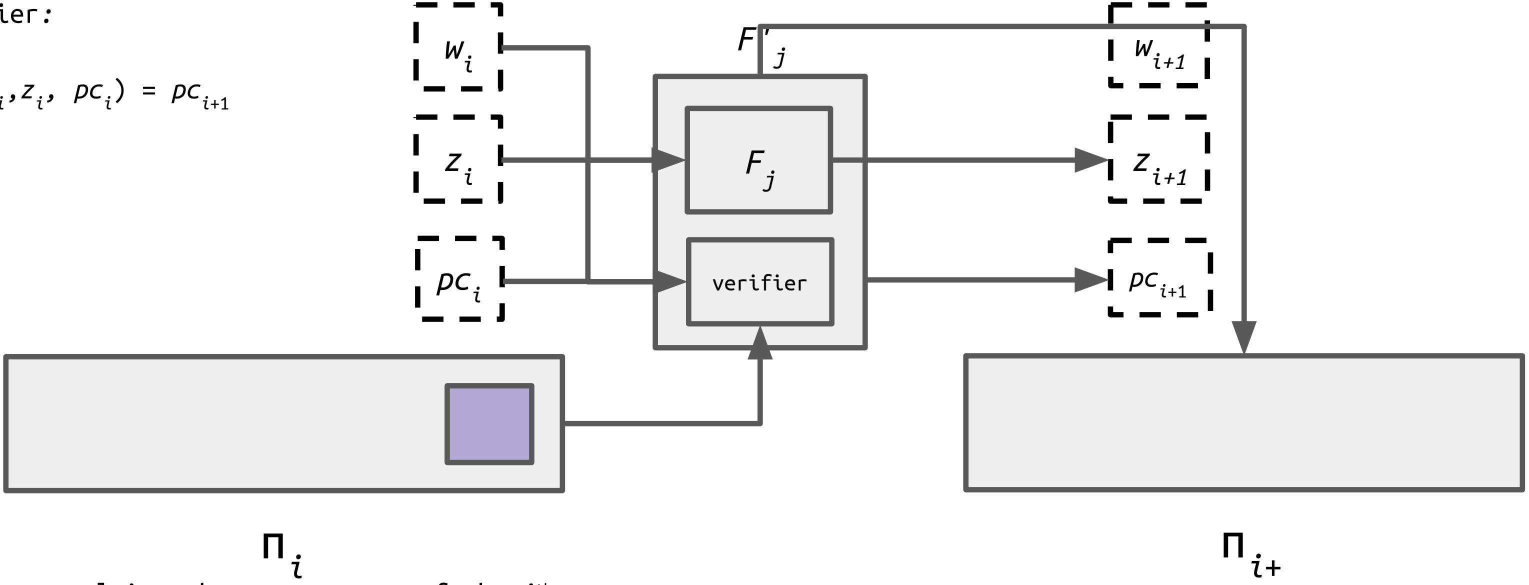


overview: *constructions*

non-uniform IVC (NIVC)

verifier:

- $\varphi(w_i, z_i, pc_i) = pc_{i+1}$



u_i claims the correctness of the i^{th} step

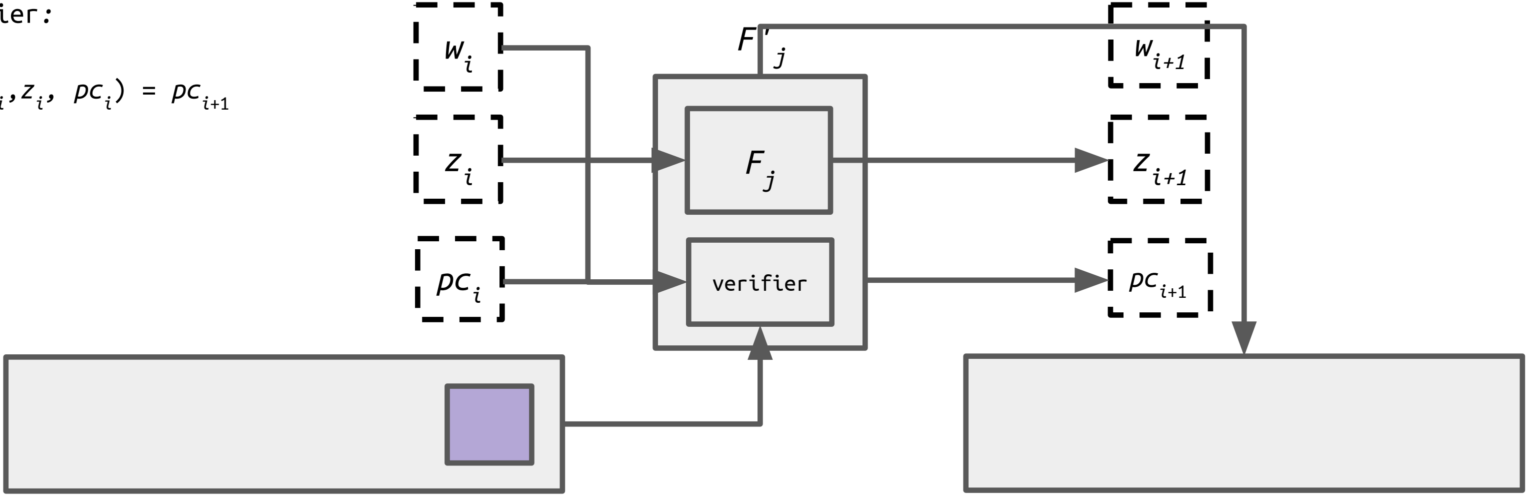
1

overview: *constructions*

non-uniform IVC (NIVC)

verifier*:*

- $\varphi(w_i, z_i, pc_i) = pc_{i+1}$



Π_i

u_i claims the correctness of the i^{th} step

Π_{i+}

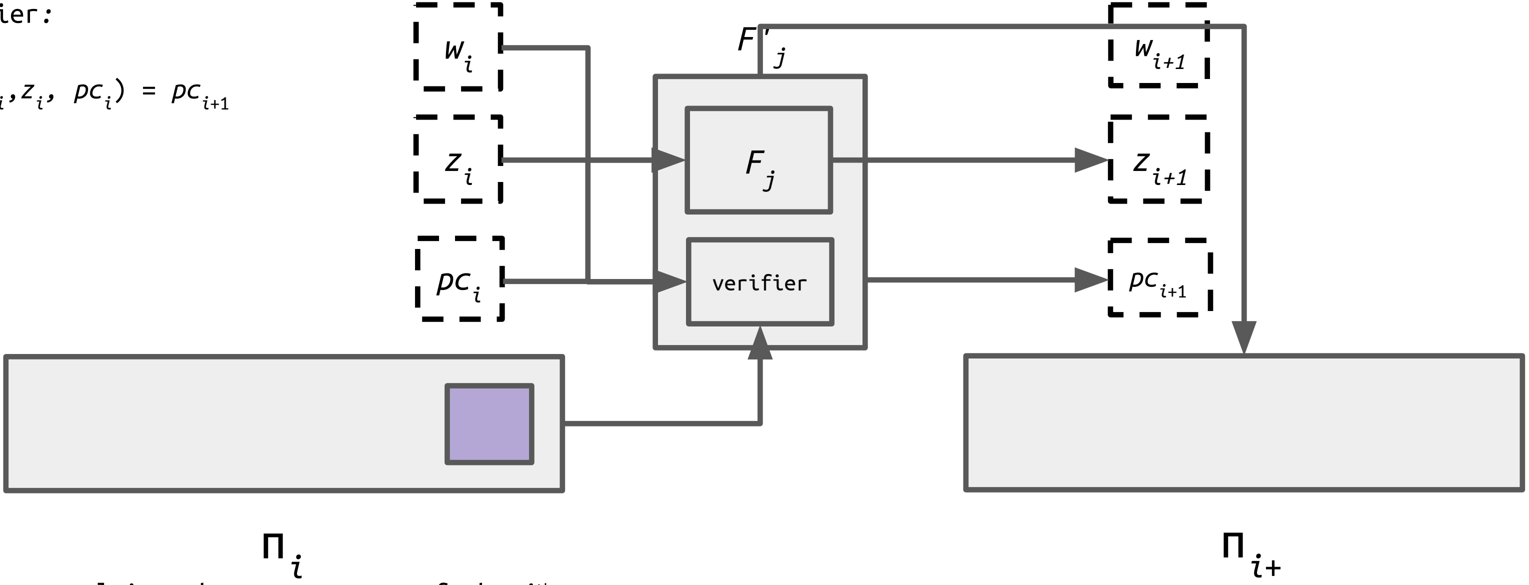
1

overview: *constructions*

non-uniform IVC (NIVC)

verifier:

- $\varphi(w_i, z_i, pc_i) = pc_{i+1}$



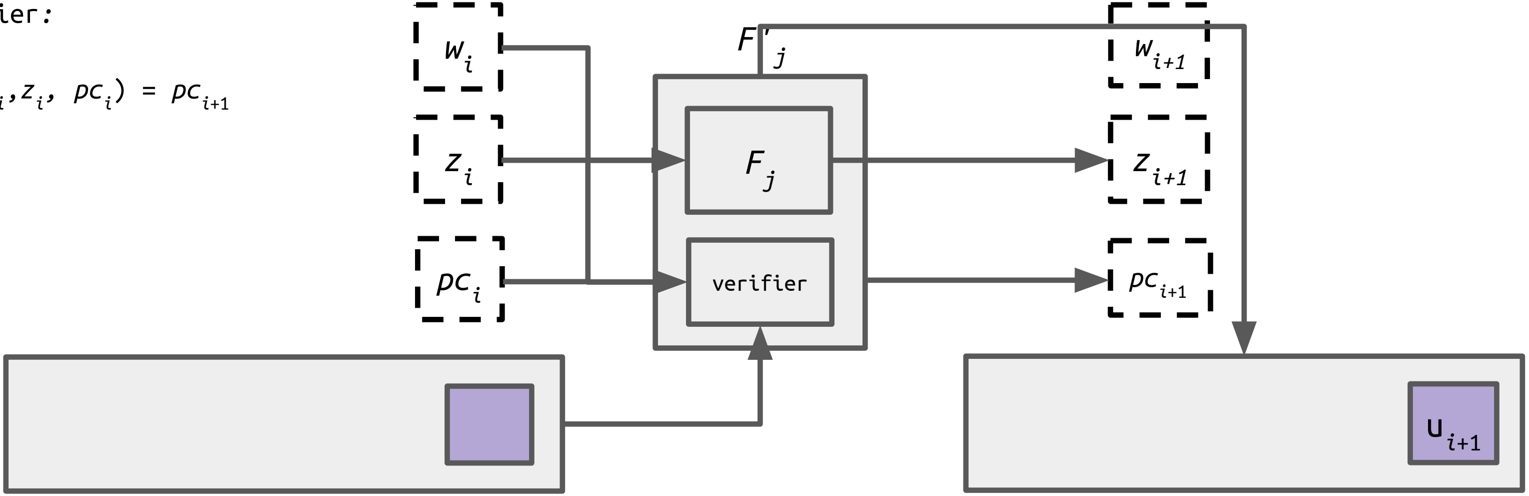
u_i claims the correctness of the i^{th} step

overview: *constructions*

non-uniform IVC (NIVC)

verifier:

- $\varphi(w_i, z_i, pc_i) = pc_{i+1}$



Π_i

u_i claims the correctness of the i^{th} step

Π_{i+}

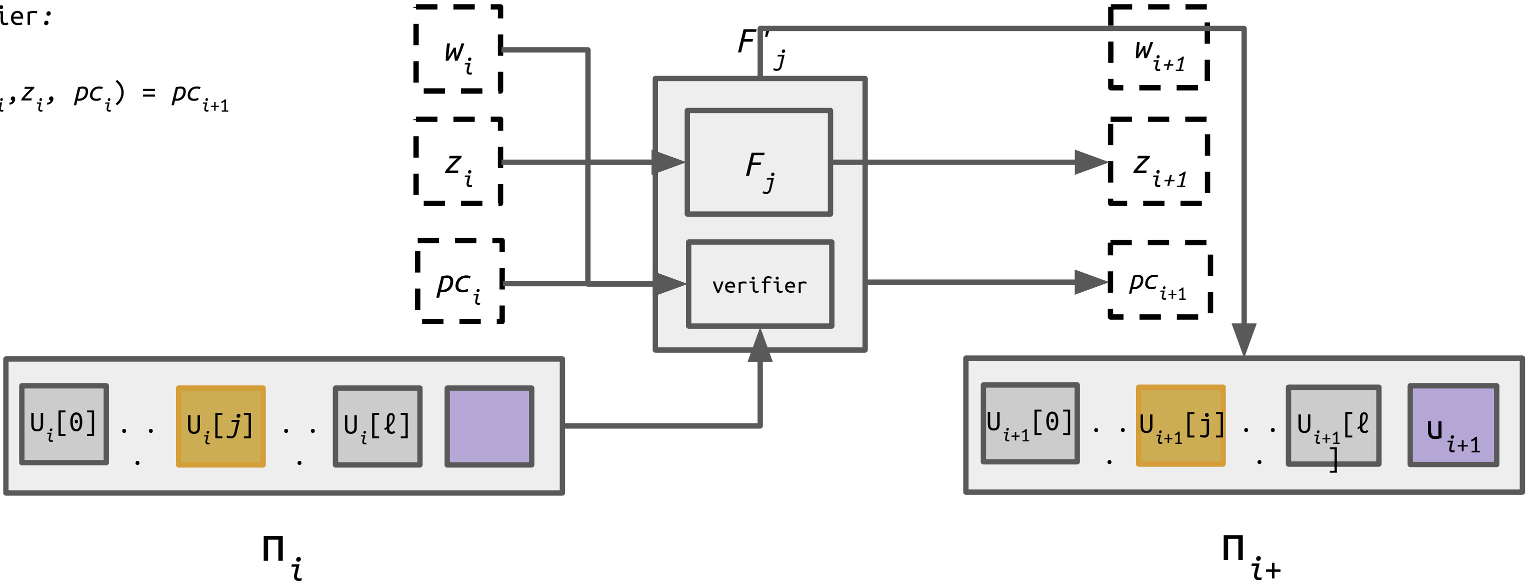
1

overview: *constructions*

non-uniform IVC (NIVC)

verifier:

- $\varphi(w_i, z_i, pc_i) = pc_{i+1}$



u_i claims the correctness of the i^{th} step

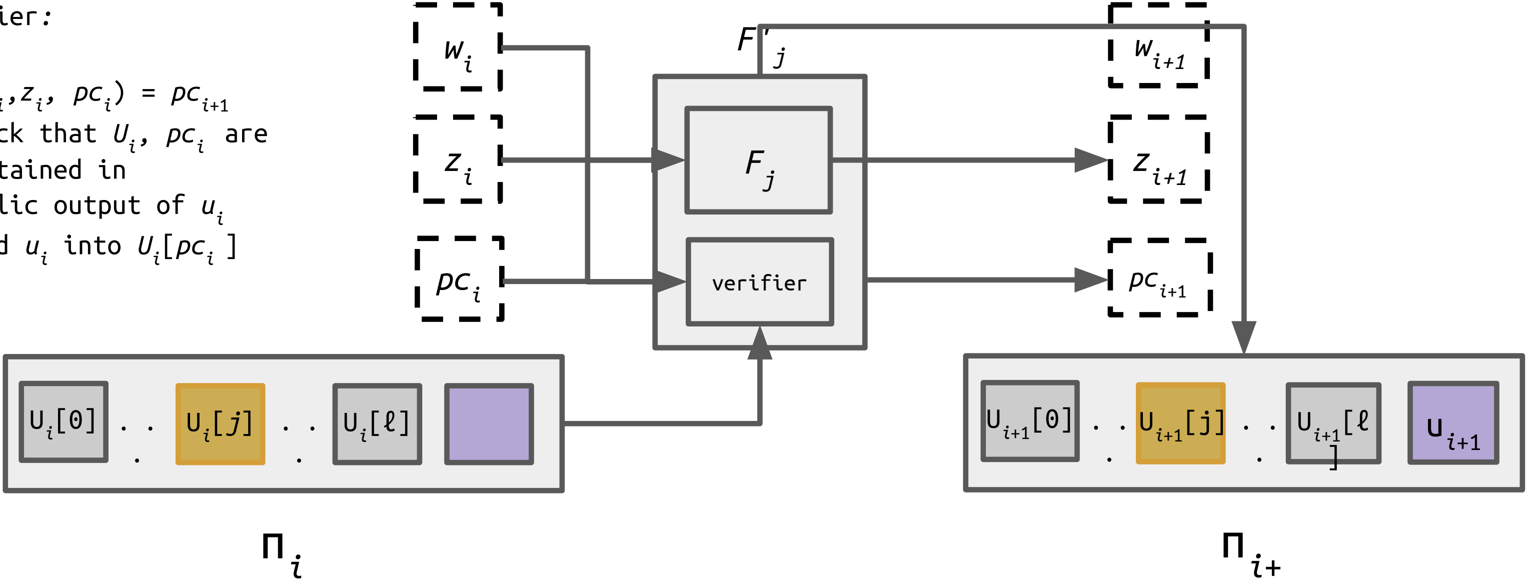
$U_i[j]$ attests to all prior i -1 iterations of F'_j

overview: *constructions*

non-uniform IVC (NIVC)

verifier:

- $\varphi(w_i, z_i, pc_i) = pc_{i+1}$
- check that U_i, pc_i are contained in public output of u_i
- fold u_i into $U_i[pc_i]$



u_i claims the correctness of the i^{th} step

$U_i[j]$ attests to all prior i -1 iterations of F'_j

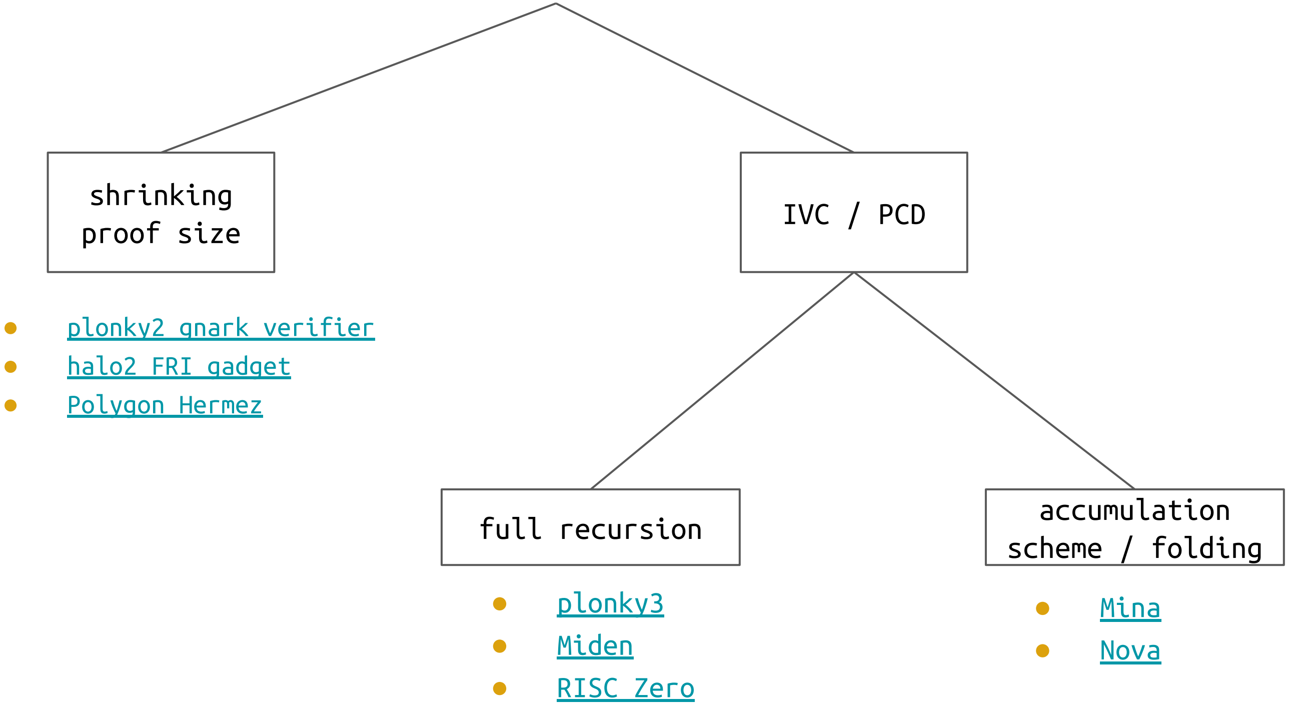
agenda

1. **overview**
   a) motivation
   b) constructions

2. **comparison**
   a) implementations
   b) recursion threshold
   c) support for lookup arguments

3. **future work**
   a) tooling & interfaces
   b) benchmarking
   c) standards & specifications
   d) security

comparison: *implementations*



shrinking proof size

- plonky2 gnark verifier
- halo2 FRI gadget
- Polygon Hermez

IVC / PCD

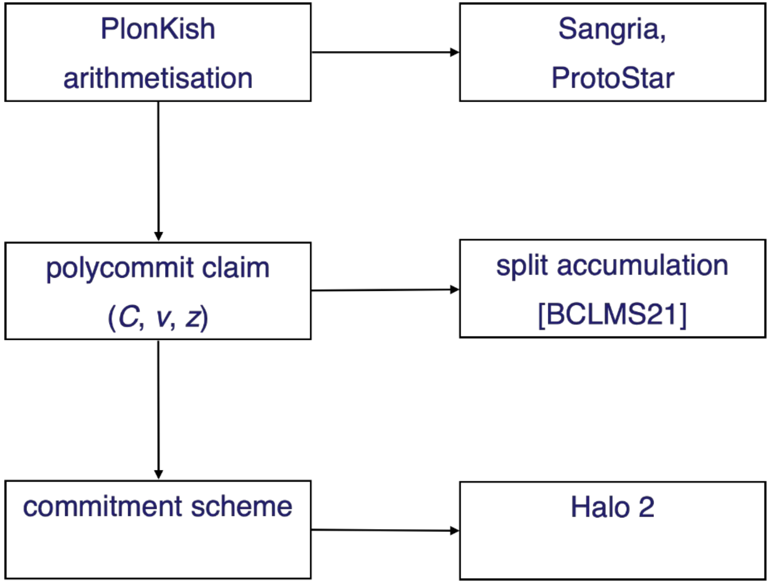
full recursion

- plonky3
- Miden
- RISC Zero

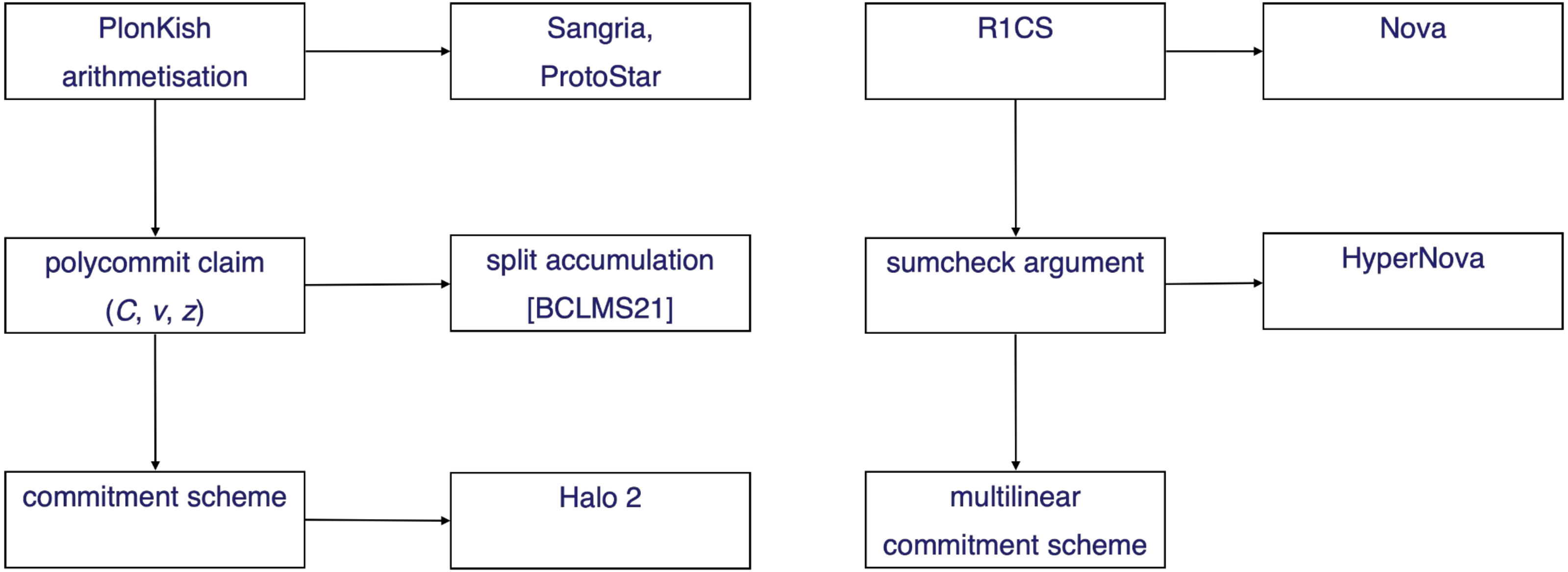
accumulation scheme / folding

- Mina
- Nova

comparison: *recursion threshold*



comparison: *recursion threshold*



comparison: *recursion threshold*

protocol	relation	accumulator	“reduce”	“combine”
halo2-IPA	PlonKish	IPA polycommit opening proofs	*P*: vanishing argument, multiopen argument, IPA	*P*: random linear combination and opening proof
			V: produce challenges, check multiopen argument, check logarithmic part of IPA	*V*: random linear combination and partial opening proof
BCLMS21	R1CS	Hadamard product vector commitment claims	*P*: commit to matrix-vector product	*P*: commit to error term
			V: none	*V*: add commitments w/ error
Nova	R1CS	committed relaxed R1CS	*P*: commit to witness	*P*: commit to error term
			V: none	*V*: add commitments w/ error
Sangria	PlonK	committed relaxed PlonK	*P*: commit to witness	*P*: commit to error term
			V: none	*V*: add commitments w/ error

taken from Nico Mohnblatt’s presentation

comparison: *recursion threshold*

protocol	**relation**	**accumulator**	**“reduce”**	**“combine”**
Nova	R1CS	committed relaxed R1CS	P: commit to witness	P: commit to error term
			V: none	V: add commitments w/ error
HyperNova	CCS	linearised committed CCS	P: commit to witness	P: random linear combination
			P and V: run the sumcheck protocol	V: random linear combination
ProtoStar	any relation w/ algebraic verifier	commitments to all messages and compressed verifier check	P: commit to each message	P: compute the compressed cross terms
			V: produce random challenges	V: add commitments and compressed cross terms

taken from Nico Mohnblatt’s presentation

comparison: *recursion threshold*

protocol	relation	accumulator	“reduce”	“combine”
Nova	R1CS	committed relaxed R1CS	*P*: commit to witness	*P*: commit to error term
			V: none	*V*: add commitments w/ error
HyperNova	CCS	linearised committed CCS	*P*: commit to witness	*P*: random linear combination
			P and *V*: run the sumcheck protocol	*V*: random linear combination
ProtoStar	any relation w/ algebraic verifier	commitments to all messages and compressed verifier check	*P*: commit to each message	*P*: compute the compressed cross terms
			V: produce random challenges	*V*: add commitments and compressed cross terms

need additively homomorphic commitments!

taken from Nico Mohnblatt’s presentation

comparison: *support for lookup arguments*

cq only works with KZG commitment scheme!

agenda

1. **overview**
   a) motivation
   b) constructions

2. **comparison**
   a) implementations
   b) recursion threshold
   c) support for lookup arguments

3. **future work**
   a) tooling & interfaces
   b) benchmarking
   c) standards & specifications
   d) security

future work

tooling &
interfaces

future work

tooling & interfaces

benchmarking

future work

tooling & interfaces

benchmarking

standards & specifications

future work

- tooling & interfaces
- benchmarking
- standards & specifications
- security