geometry

Sangria is relaxed PLONK: A Nova-style folding scheme for PLONK

APR | 2023

Sangria: Overview

- Folding schemes were introduced in [Nova] and are the key ingredient to achieving cheap recursion. Nova only shows a folding scheme for R1CS.

- A folding scheme compresses two instances of a problem into a single instance of the same problem

- Sangria is a folding scheme for PLONKish circuits

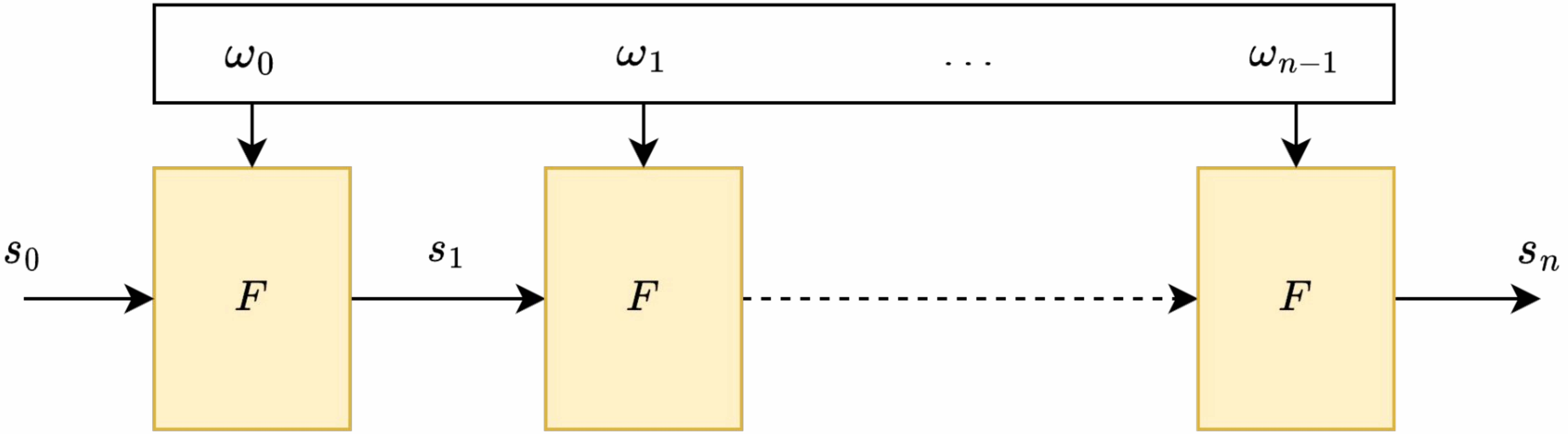
- Costs are similar to Nova:
  - Verifier work is constant in the depth of the circuit
  - Constants are worse than Nova, this is the price we pay to get a more flexible arithmetization

This Talk

- Motivation: why do we care about recursion? What are the known results?
- Some background: PLONK circuits and folding schemes
- Sangria: how to fold PLONK circuits
- Trade-offs and a new design space

Goal: Incrementally Verifiable Computation (IVC)

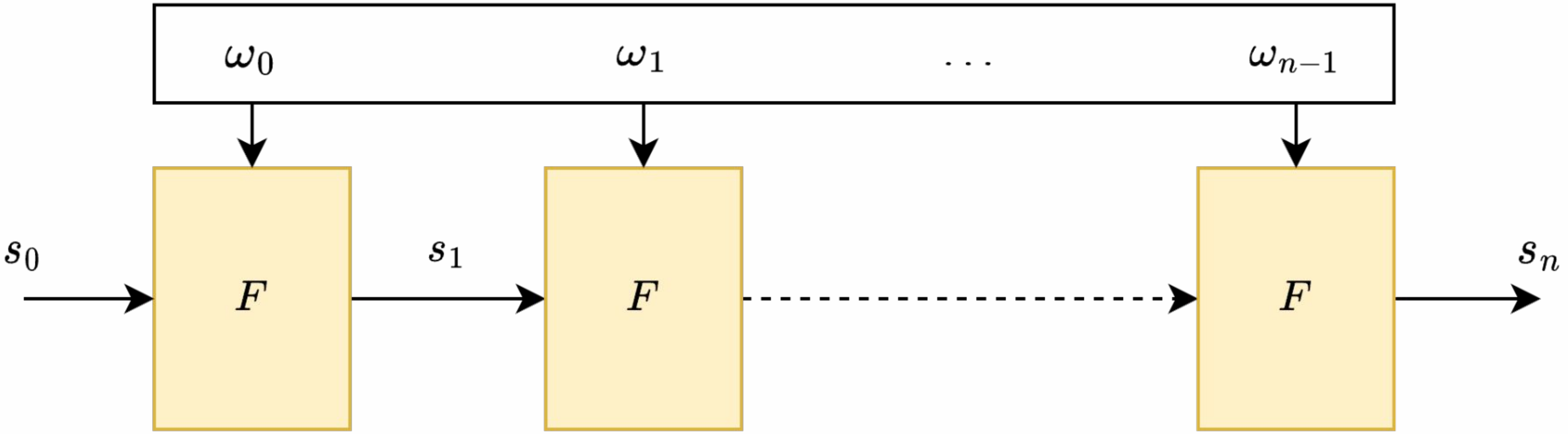
Imagine a long computation that results from *n* applications of a function *F*.



We want to produce a succinct proof that the prover knows $\omega_0, \omega_1, \ldots, \omega_{n-1}$ such that s_n is the correct final state

Goal: Incrementally Verifiable Computation (IVC)

Imagine a long computation that results from *n* applications of a function *F*.

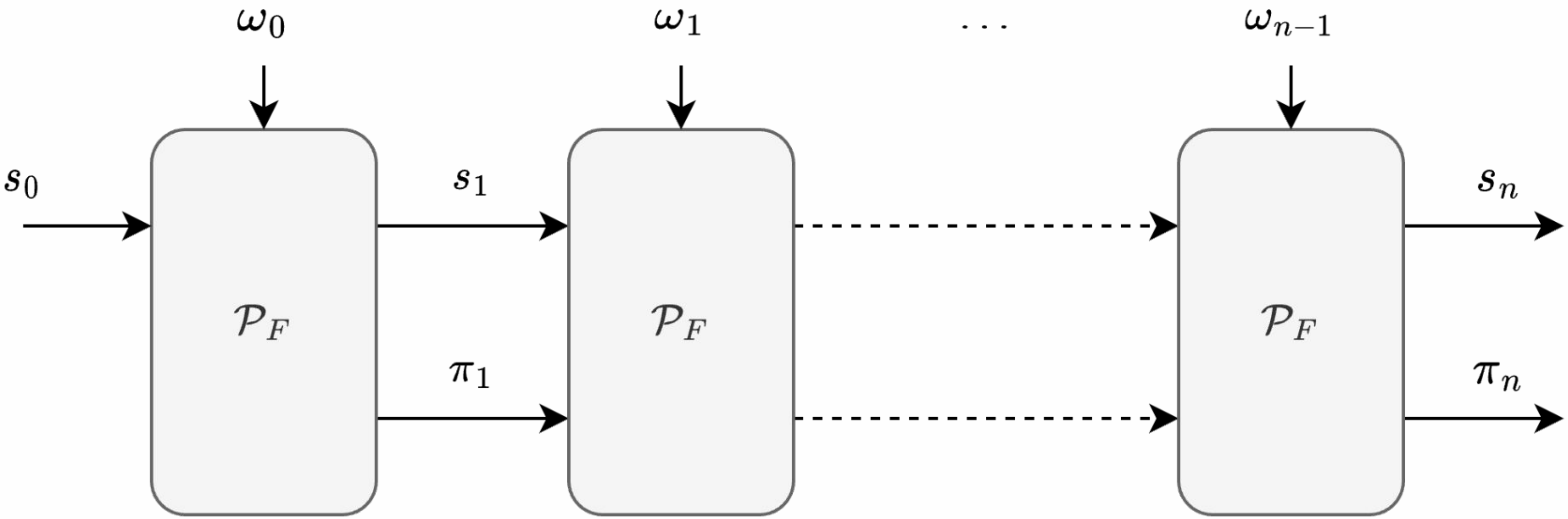


We want to produce a succinct proof that the prover knows $\omega_0, \omega_1, \ldots, \omega_{n-1}$ such that s_n is the correct final state

In practice, IVC can be used to build zkVMs, rollups and verifiable delay functions

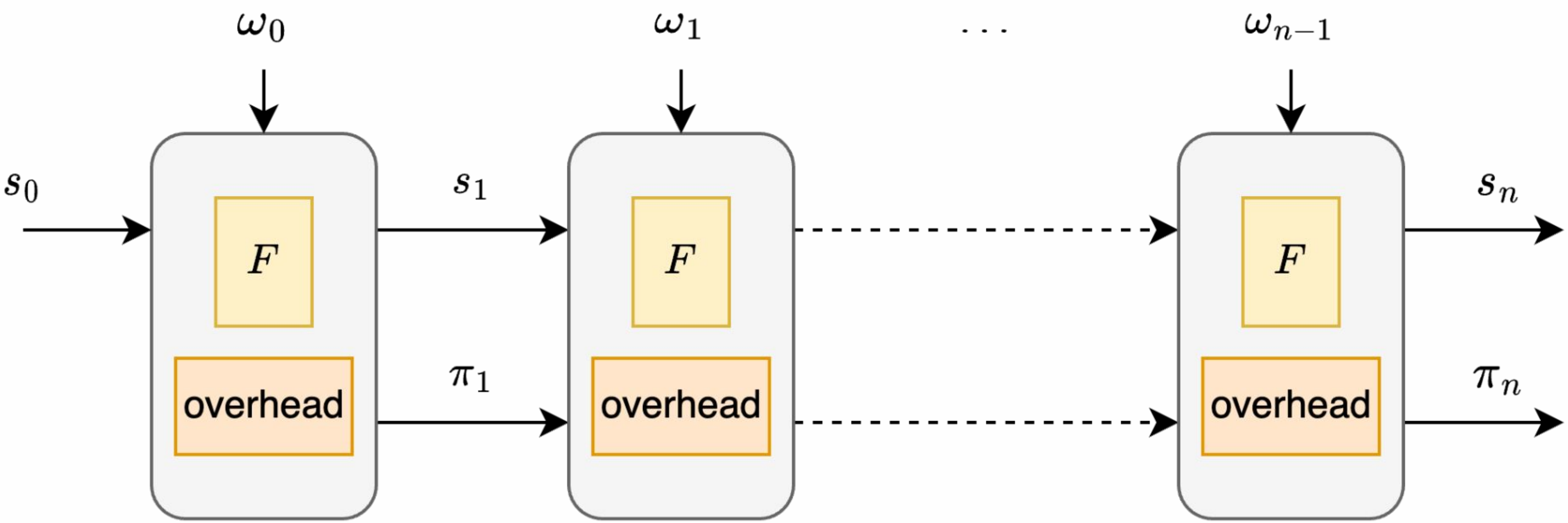
Realising IVC

To realise IVC, we define a prover the takes in a state **and** proof, and updates both



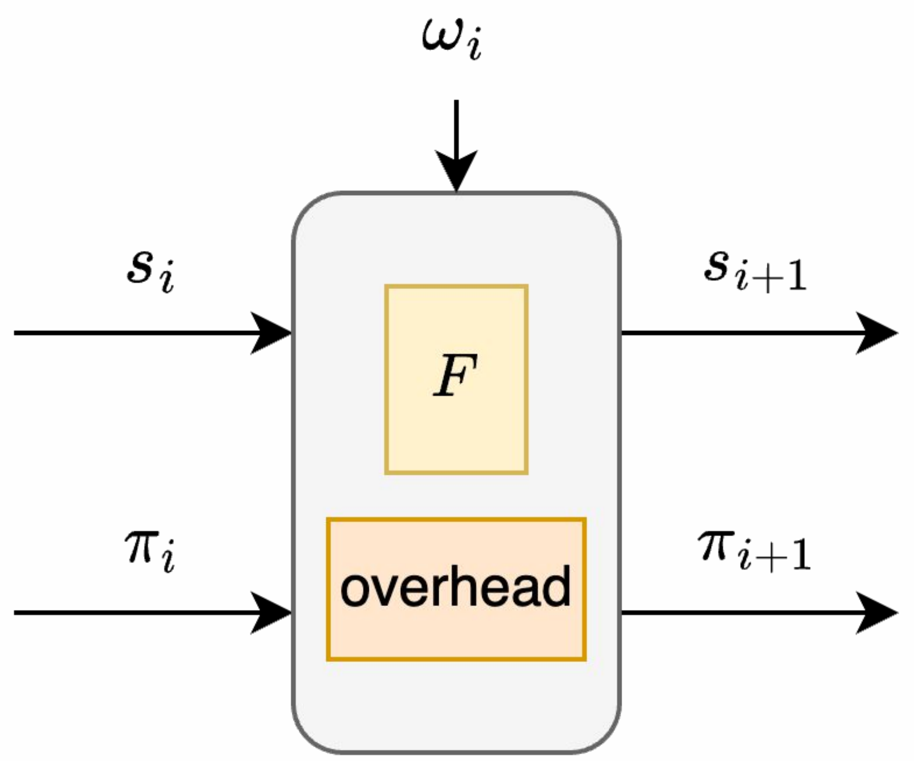
Realising IVC

To realise IVC, we define a prover the takes in a state **and** proof, and updates both



We must pay some overhead to update the proof. How small can we get it?

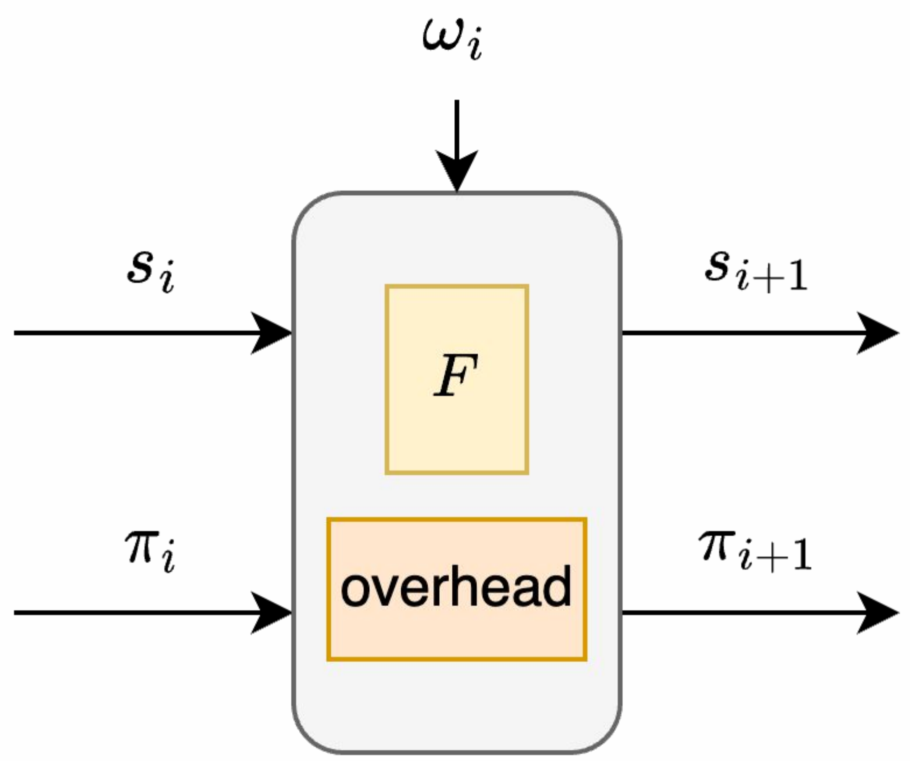
Recursion Overheads



Recent works have drastically reduced the recursion overhead:

	Overhead	Examples
SNARK of SNARK	Read whole proof, run full verifier, defer nothing	Plonky2, recursive STARK, Fractal, Groth16 + [BCTV14]

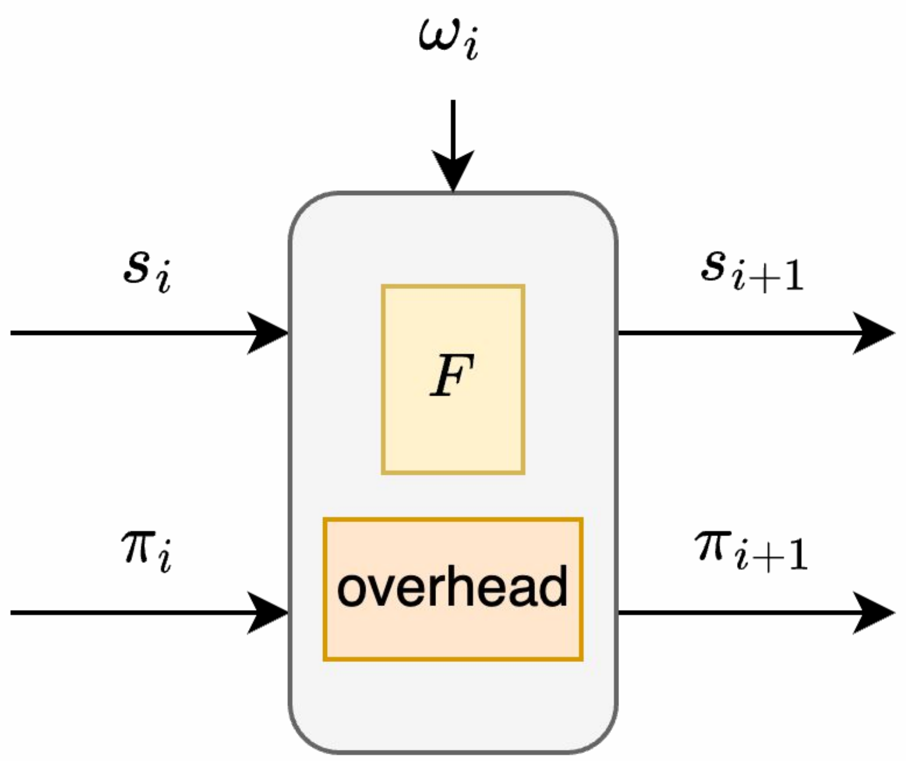
Recursion Overheads



Recent works have drastically reduced the recursion overhead:

	Overhead	**Examples**
SNARK of SNARK	Read whole proof, run full verifier, defer nothing	Plonky2, recursive STARK, Fractal, Groth16 + [BCTV14]
Accumulation (atomic)	Read whole proof, run partial verifier, defer hard verification	Halo/Halo2, [BCMS20]

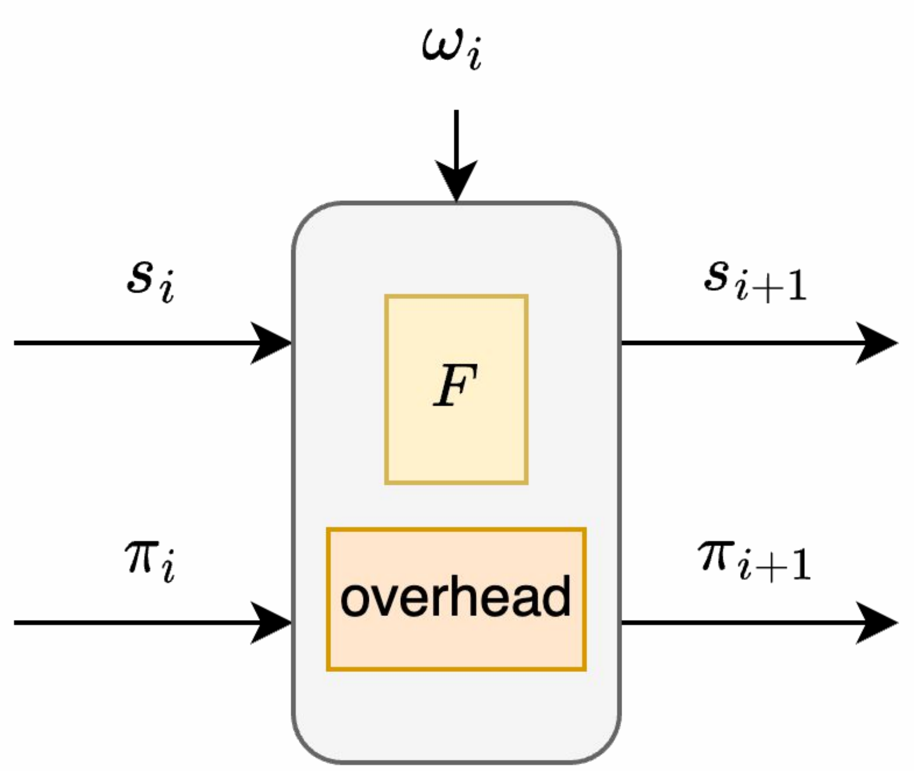
Recursion Overheads



Recent works have drastically reduced the recursion overhead:

	Overhead	Examples
SNARK of SNARK	Read whole proof, run full verifier, defer nothing	Plonky2, recursive STARK, Fractal, Groth16 + [BCTV14]
Accumulation (atomic)	Read whole proof, run partial verifier, defer hard verification	Halo/Halo2, [BCMS20]
Accumulation (split)	Read partial proof, run partial verifier, defer hard verification and reading whole proof	[BCLMS21]

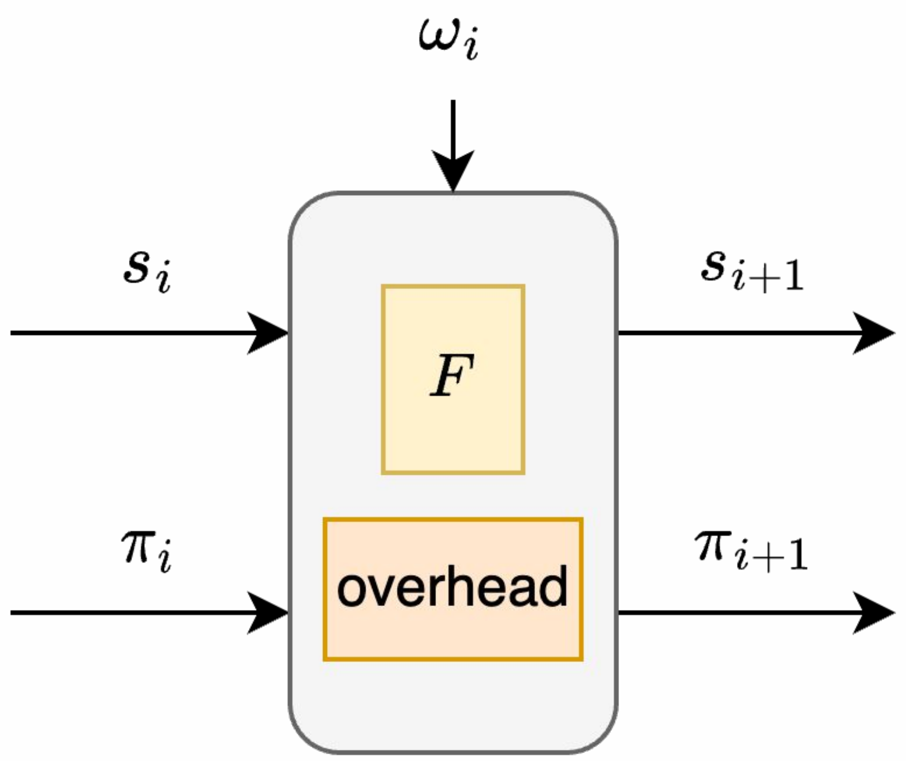
Recursion Overheads



Recent works have drastically reduced the recursion overhead:

	Overhead	Examples
SNARK of SNARK	Read whole proof, run full verifier, defer nothing	Plonky2, recursive STARK, Fractal, Groth16 + [BCTV14]
Accumulation (atomic)	Read whole proof, run partial verifier, defer hard verification	Halo/Halo2, [BCMS20]
Accumulation (split)	Read partial proof, run partial verifier, defer hard verification and reading whole proof	[BCLMS21]
Folding	Read an unproven instance, compress it into a running instance, **defer proving**	Nova

Recursion Overheads



Recent works have drastically reduced the recursion overhead:

	Overhead	Examples
SNARK of SNARK	Read whole proof, run full verifier, defer nothing	Plonky2, recursive STARK, Fractal, Groth16 + [BCTV14]
Accumulation (atomic)	Read whole proof, run partial verifier, defer hard verification	Halo/Halo2, [BCMS20]
Accumulation (split)	Read partial proof, run partial verifier, defer hard verification and reading whole proof	[BCLMS21]
Folding	Read an unproven instance, compress it into a running instance, **defer proving**	Nova

Unlikely we can do better than deferring the proving. However, we can maybe have a more efficient circuit representation

Definition: PLONK Arithmetization

$\mathbf{q_L}$	$\mathbf{q_R}$	$\mathbf{q_O}$	$\mathbf{q_M}$	$\mathbf{q_C}$	$\mathbf{a}$	$\mathbf{b}$	$\mathbf{c}$

- PLONK traces are like Sudokus
  - fixed-size grid
  - fill the cells with “numbers”
  - set of rules to follow

Definition: PLONK Arithmetization

q_L	q_R	q_O	q_M	q_C	a	b	c
1	0	0	0	0	5		
1	0	0	0	0	10		
-1	0	0	1	0			
-1	0	0	1	0			
1	1	-1	0	0			
0	0	-1	1	0			

- PLONK traces are like Sudokus
  - fixed-size grid
  - fill the cells with "numbers"
  - set of rules to follow

- Some values are already filled in: these are the **selectors** and **public inputs**

Definition: PLONK Arithmetization

q_L	q_R	q_O	q_M	q_C	a	b	c
1	0	0	0	0	5		
1	0	0	0	0	10		
-1	0	0	1	0			
-1	0	0	1	0			
1	1	-1	0	0			
0	0	-1	1	0			

- PLONK traces are like Sudokus
  - fixed-size grid
  - fill the cells with “numbers”
  - set of rules to follow
- Some values are already filled in: these are the **selectors** and **public inputs**
- Rule #1: copy constraints

Definition: PLONK Arithmetization

$\mathbf{q_L}$	$\mathbf{q_R}$	$\mathbf{q_O}$	$\mathbf{q_M}$	$\mathbf{q_C}$	$\mathbf{a}$	$\mathbf{b}$	$\mathbf{c}$
1	0	0	0	0	5		
1	0	0	0	0	10		
-1	0	0	1	0			
-1	0	0	1	0			
1	1	-1	0	0			
0	0	-1	1	0			

$$(\mathbf{q_L})_i\mathbf{a}_i + (\mathbf{q_R})_i\mathbf{b}_i + (\mathbf{q_O})_i\mathbf{c}_i + (\mathbf{q_M})_i\mathbf{a}_i\mathbf{b}_i + (\mathbf{q_C})_i = 0$$

- PLONK traces are like Sudokus
  - fixed-size grid
  - fill the cells with “numbers”
  - set of rules to follow

- Some values are already filled in: these are the **selectors** and **public inputs**

- Rule #1: copy constraints
- Rule #2: gate equation, applies at each row

Definition: PLONK Arithmetization

$\mathbf{q_L}$	$\mathbf{q_R}$	$\mathbf{q_O}$	$\mathbf{q_M}$	$\mathbf{q_C}$	a	b	c
1	0	0	0	0	5		
1	0	0	0	0	10		
-1	0	0	1	0			
-1	0	0	1	0			
1	1	-1	0	0			
0	0	-1	1	0			

$$\mathbf{X} = (\mathbf{x_a}, \mathbf{x_b}, \mathbf{x_c})$$

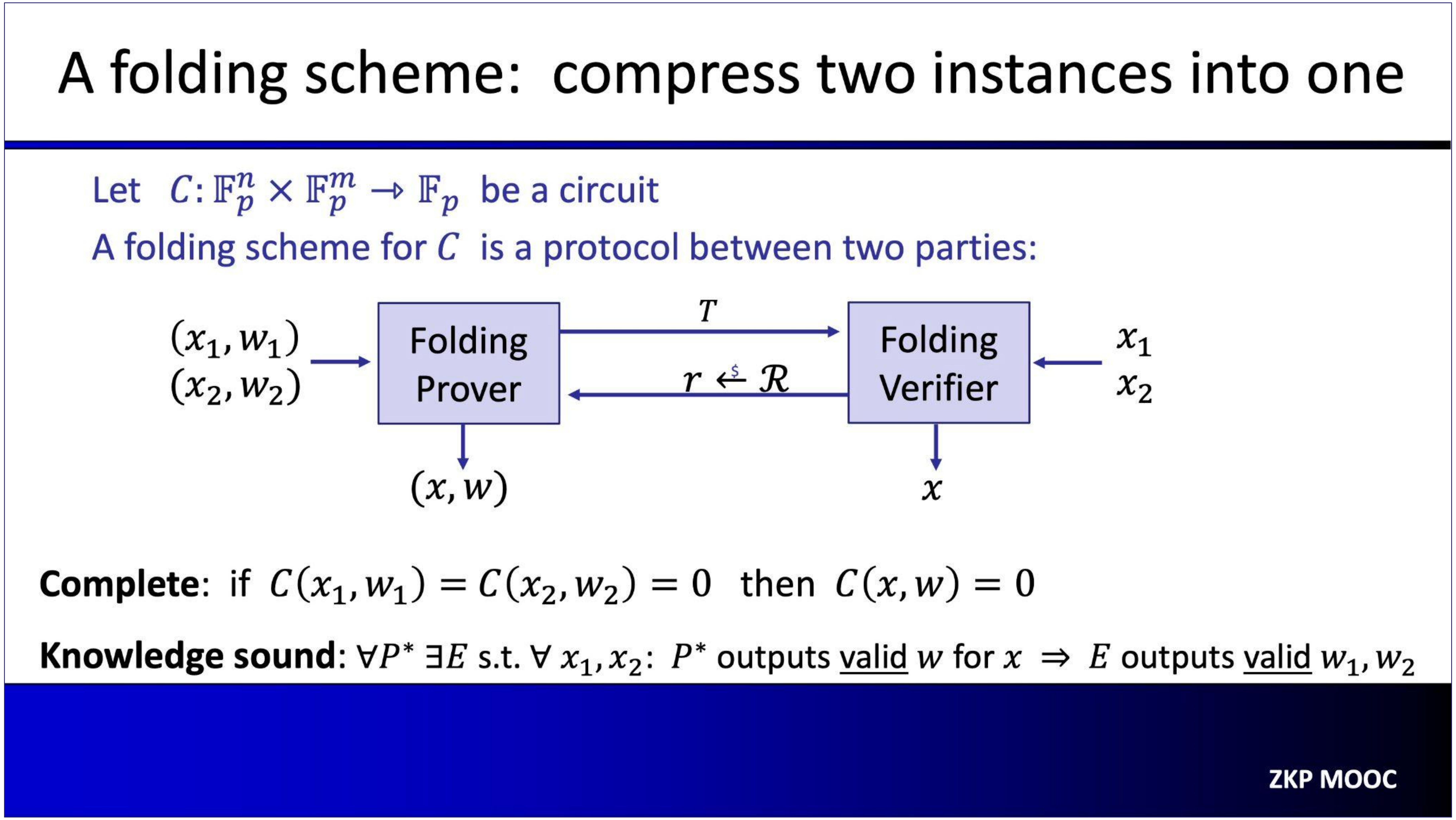
$$\mathbf{W} = (\mathbf{w_a}, \mathbf{w_b}, \mathbf{w_c})$$

$$(\mathbf{q_L})_i \mathbf{a}_i + (\mathbf{q_R})_i \mathbf{b}_i + (\mathbf{q_O})_i \mathbf{c}_i + (\mathbf{q_M})_i \mathbf{a}_i \mathbf{b}_i + (\mathbf{q_C})_i = 0$$

- PLONK traces are like Sudokus
  - fixed-size grid
  - fill the cells with “numbers”
  - set of rules to follow
- Some values are already filled in: these are the **selectors** and **public inputs**
- Rule #1: copy constraints
- Rule #2: gate equation, applies at each row
- Selectors and rules define a circuit
- Trace be divided into an instance and witness

Definition: Folding Schemes

Definition: Folding Schemes



Slide from ZKP MOOC, Lecture 10, Dan Boneh. [Video]() and [slides]().

A folding scheme for PLONK: Attempt #1

- We use the cryptographer's best friend: a random linear combination

$$
\begin{array}{c}\texttt{trace}'\\ \begin{array}{|c|c|c|}\hline \mathbf{a}'_1 & \mathbf{b}'_1 & \mathbf{c}'_1\\ \hline \mathbf{a}'_2 & \mathbf{b}'_2 & \mathbf{c}'_2\\ \hline \mathbf{a}'_3 & \mathbf{b}'_3 & \mathbf{c}'_3\\ \hline \mathbf{a}'_4 & \mathbf{b}'_4 & \mathbf{c}'_4\\ \hline \end{array}\end{array}
+ \boldsymbol{r}
\begin{array}{c}\texttt{trace}''\\ \begin{array}{|c|c|c|}\hline \mathbf{a}''_1 & \mathbf{b}''_1 & \mathbf{c}''_1\\ \hline \mathbf{a}''_2 & \mathbf{b}''_2 & \mathbf{c}''_2\\ \hline \mathbf{a}''_3 & \mathbf{b}''_3 & \mathbf{c}''_3\\ \hline \mathbf{a}''_4 & \mathbf{b}''_4 & \mathbf{c}''_4\\ \hline \end{array}\end{array}
=
\begin{array}{c}\texttt{final\_trace}\\ \begin{array}{|c|c|c|}\hline \mathbf{a}_1 & \mathbf{b}_1 & \mathbf{c}_1\\ \hline \mathbf{a}_2 & \mathbf{b}_2 & \mathbf{c}_2\\ \hline \mathbf{a}_3 & \mathbf{b}_3 & \mathbf{c}_3\\ \hline \mathbf{a}_4 & \mathbf{b}_4 & \mathbf{c}_4\\ \hline \end{array}\end{array}
$$

- Copy constraints are preserved
- The gate equation is not linear, we are going to run into some trouble

A folding scheme for PLONK: Attempt #2

- Apply Nova's insights:
  - the gate equation can be relaxed
  - the Prover works over the trace while the Verifier works with commitments

A folding scheme for PLONK: Attempt #2

- Apply Nova's insights:
  - the gate equation can be relaxed
  - the Prover works over the trace while the Verifier works with commitments

- Relaxed PLONK arithmetization:

Witness: PLONK witness $\mathbf{W} = (\mathbf{w_a}, \mathbf{w_b}, \mathbf{w_c})$ and an error vector **e**

Instance: public inputs $\mathbf{X} = (\mathbf{x_a}, \mathbf{x_b}, \mathbf{x_c})$, a scalar *u* and commitments $\boxed{\mathbf{w_a}}$ $\boxed{\mathbf{w_b}}$ $\boxed{\mathbf{w_c}}$ $\boxed{\mathbf{e}}$

Relaxed gate equation:

$$u\,[(\mathbf{q_L})_i\mathbf{a}_i + (\mathbf{q_R})_i\mathbf{b}_i + (\mathbf{q_O})_i\mathbf{c}_i] + (\mathbf{q_M})_i\mathbf{a}_i\mathbf{b}_i + u^2(\mathbf{q_C})_i + \mathbf{e}_i$$

A folding scheme for PLONK: Attempt #2

- Folding:

$$\mathtt{final\_trace} = \mathtt{trace}' + r * \mathtt{trace}''$$

$$u = u' + ru''$$

$$\mathbf{e} = ?$$

A folding scheme for PLONK: Attempt #2

- Folding:

$$\texttt{final\_trace} = \texttt{trace}' + r * \texttt{trace}''$$

$$u = u' + ru''$$

$$\mathbf{e} = ?$$

- Plugging in the new values to the relaxed gate equation yields:

$$\text{Gate}(\texttt{final\_trace}) = \text{Gate}(\texttt{trace}') + r\mathbf{t} + r^2 * \text{Gate}(\texttt{trace}'')$$

A folding scheme for PLONK: Attempt #2

- Folding:

$$\texttt{final\_trace} = \texttt{trace}' + r * \texttt{trace}''$$

$$u = u' + ru''$$

$$\mathbf{e} = ?$$

- Plugging in the new values to the relaxed gate equation yields:

$$\mathrm{Gate}(\texttt{final\_trace}) = \mathrm{Gate}(\texttt{trace}') + r\mathbf{t} + r^2 * \mathrm{Gate}(\texttt{trace}'')$$

- We can get rid of ***t*** by defining the error term and its folding as:

$$\mathbf{e} = \mathbf{e}' - r\mathbf{t} + r^2\mathbf{e}''$$

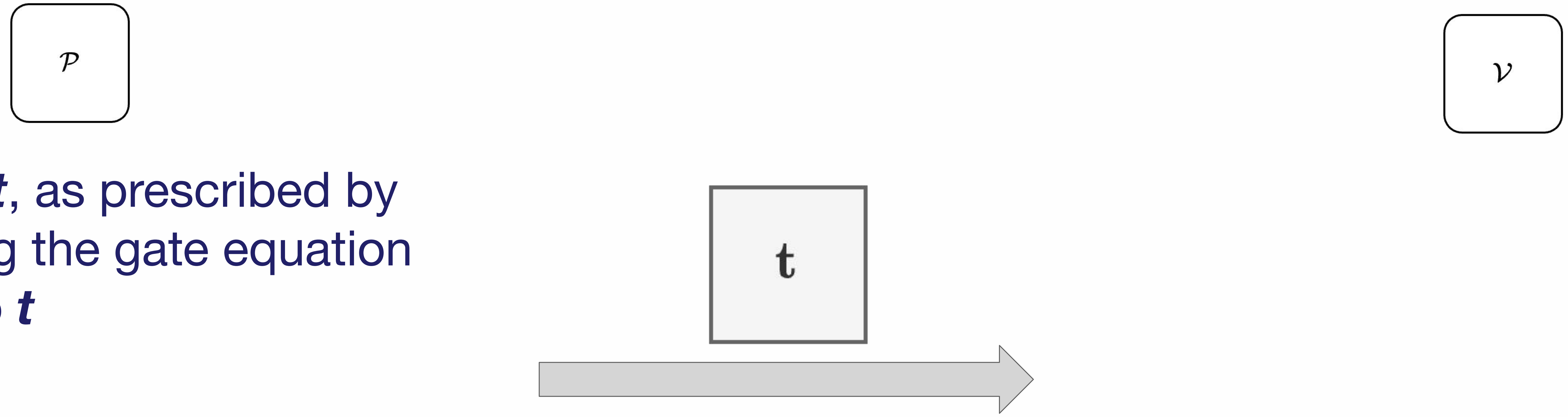
A folding scheme for PLONK: Sangria

$\mathcal{P}$

$\mathcal{V}$

- Compute $\boldsymbol{t}$, as prescribed by distributing the gate equation
- Commit to $\boldsymbol{t}$

A folding scheme for PLONK: Sangria



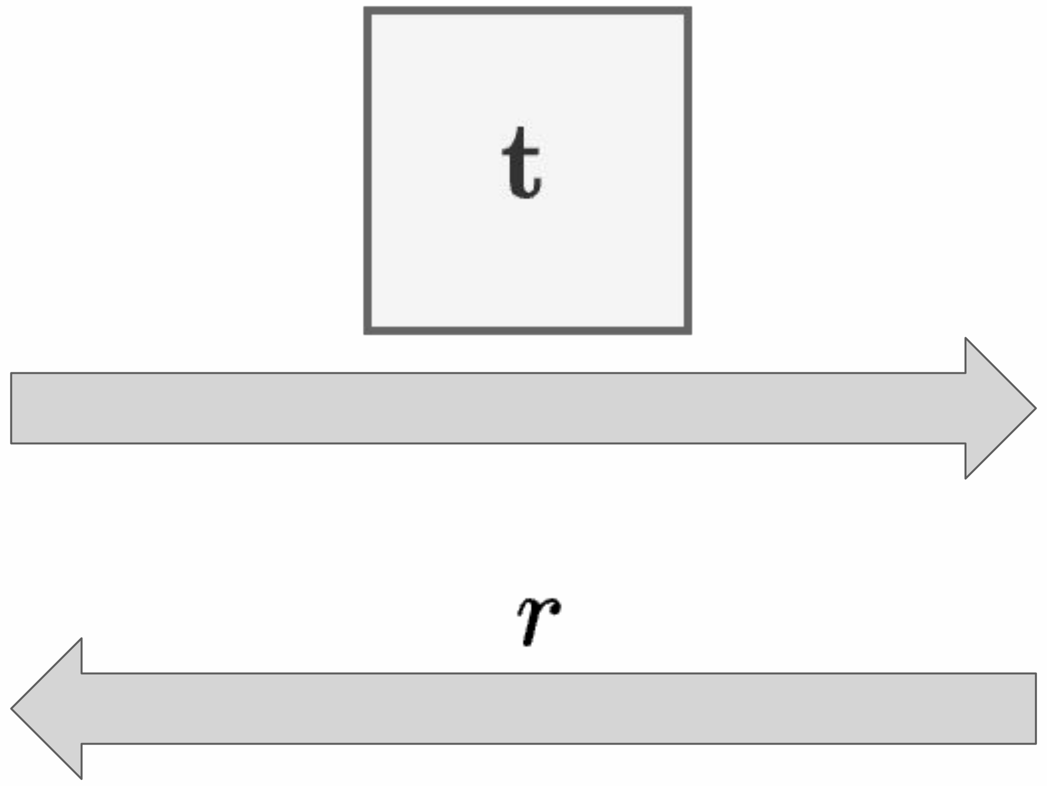
- Compute ***t***, as prescribed by distributing the gate equation
- Commit to ***t***

A folding scheme for PLONK: Sangria

$\mathcal{P}$

$\mathcal{V}$

- Compute $\boldsymbol{t}$, as prescribed by distributing the gate equation
- Commit to $\boldsymbol{t}$



- Sample a random value r

A folding scheme for PLONK: Sangria

$\mathcal{P}$

$\mathcal{V}$

- Compute ***t***, as prescribed by distributing the gate equation
- Commit to ***t***

$\xrightarrow{\ \mathbf{t}\ }$

- Sample a random value *r*
- Compute linear combinations of the public inputs and commitments:

$$\boxed{\mathbf{w_a}} = \boxed{\mathbf{w'_a}} + r\,\boxed{\mathbf{w''_a}}$$

$$\boxed{\mathbf{w_b}} = \boxed{\mathbf{w'_b}} + r\,\boxed{\mathbf{w''_b}}$$

$$\boxed{\mathbf{w_c}} = \boxed{\mathbf{w'_c}} + r\,\boxed{\mathbf{w''_c}}$$

$$\boxed{\mathbf{e}} = \boxed{\mathbf{e'}} - r\,\boxed{\mathbf{t}} + r^2\,\boxed{\mathbf{e''}}$$

$\xleftarrow{\ \boldsymbol{r}\ }$

- Compute linear combinations of the trace values

$$\texttt{final\_trace} = \texttt{trace}' + r * \texttt{trace}''$$

$$\mathbf{e} = \mathbf{e}' - r\mathbf{t} + r^2\mathbf{e}''$$

The verifier $\mathcal{V}$ is given the verifier key vk *and two committed relaxed PLONK instances,* $\left(\mathbf{X}', u', \overline{W_a'}, \overline{W_b'}, \overline{W_c'}, \overline{E'}\right)$ *and* $\left(\mathbf{X}'', u'', \overline{W_a''}, \overline{W_b''}, \overline{W_c''}, \overline{E''}\right)$. *The prover $\mathcal{P}$ is given the prover key* pk *and both instances with their corresponding witnesses* $(\mathbf{W}', \mathbf{e}', r_a', r_b', r_c', r_e')$ *and* $(\mathbf{W}'', \mathbf{e}'', r_a'', r_b'', r_c'', r_e'')$.

The Sangria folding scheme proceeds as follows:

1. $\mathcal{P}$ *samples* r_t *at random and sends* $\overline{T} = \mathsf{Com}(\mathsf{pp}_E, \mathbf{t}; r_t)$ *where* $\mathbf{t}$ *is computed as:*

$$\mathbf{t} := u''(\mathbf{q_L} \circ \mathbf{a}' + \mathbf{q_R} \circ \mathbf{b}' + \mathbf{q_O} \circ \mathbf{c}') + u'(\mathbf{q_L} \circ \mathbf{a}'' + \mathbf{q_R} \circ \mathbf{b}'' + \mathbf{q_O} \circ \mathbf{c}'') \tag{7}$$

$$+ \mathbf{q_M} \circ (\mathbf{a}' \circ \mathbf{b}'' + \mathbf{a}'' \circ \mathbf{b}') \tag{8}$$

$$+ 2ru'u''\mathbf{q_C} \tag{9}$$

2. $\mathcal{V}$ *samples the challenge* r *at random.*

3. $\mathcal{P}$ *and* $\mathcal{V}$ *output the folded instance* $(\mathbf{X}, u, \overline{W_a}, \overline{W_b}, \overline{W_c}, \overline{E})$ *where:*

$$\mathbf{X} \leftarrow \mathbf{X}' + r\mathbf{X}'' \tag{10}$$

$$u \leftarrow u' + ru'' \tag{11}$$

$$\overline{W_a} \leftarrow \overline{W_a'} + r\overline{W_a''} \tag{12}$$

$$\overline{W_b} \leftarrow \overline{W_b'} + r\overline{W_b''} \tag{13}$$

$$\overline{W_c} \leftarrow \overline{W_c'} + r\overline{W_c''} \tag{14}$$

$$\overline{E} \leftarrow \overline{E'} - r\overline{T} + r^2\overline{E''} \tag{15}$$

4. $\mathcal{P}$ *outputs the folded witness* $(\mathbf{W}, \mathbf{e}, r_a, r_b, r_c, r_e)$ *where:*

$$\mathbf{W} \leftarrow \mathbf{W}' + r\mathbf{W}'' \tag{16}$$

$$r_a \leftarrow r_a' + r \cdot r_a'' \tag{17}$$

$$r_b \leftarrow r_b' + r \cdot r_b'' \tag{18}$$

$$r_c \leftarrow r_c' + r \cdot r_c'' \tag{19}$$

$$\mathbf{e} \leftarrow \mathbf{e}' - r\mathbf{t} + r^2\mathbf{e}'' \tag{20}$$

$$r_e \leftarrow r_e' - r \cdot r_t + r^2 \cdot r_e'' \tag{21}$$

Performance

- The verifier’s work is dominated by the additions of commitments

$$\boxed{\mathbf{w_a}} = \boxed{\mathbf{w}'_\mathbf{a}} + r\ \boxed{\mathbf{w}''_\mathbf{a}}$$

$$\boxed{\mathbf{w_b}} = \boxed{\mathbf{w}'_\mathbf{b}} + r\ \boxed{\mathbf{w}''_\mathbf{b}}$$

$$\boxed{\mathbf{w_c}} = \boxed{\mathbf{w}'_\mathbf{c}} + r\ \boxed{\mathbf{w}''_\mathbf{c}}$$

$$\boxed{\mathbf{e}} = \boxed{\mathbf{e}'} - r\ \boxed{\mathbf{t}} + r^2\ \boxed{\mathbf{e}''}$$

- In practice we will be folding a standard PLONK instance into a relaxed PLONK instance:
  - the standard instance will not have an error term
  - the verifier will only have to perform **4 point additions**

Wide Circuits and Custom Gates (TurboPLONK)

- We can deal with **wider circuits** (circuits with larger fan-in fan-out at each gate) by committing to each extra column
  - Cost: The verifier will have to perform **1 extra point addition per trace column**

Wide Circuits and Custom Gates (TurboPLONK)

- We can deal with **wider circuits** (circuits with larger fan-in fan-out at each gate) by committing to each extra column
  - Cost: The verifier will have to perform **1 extra point addition per trace column**

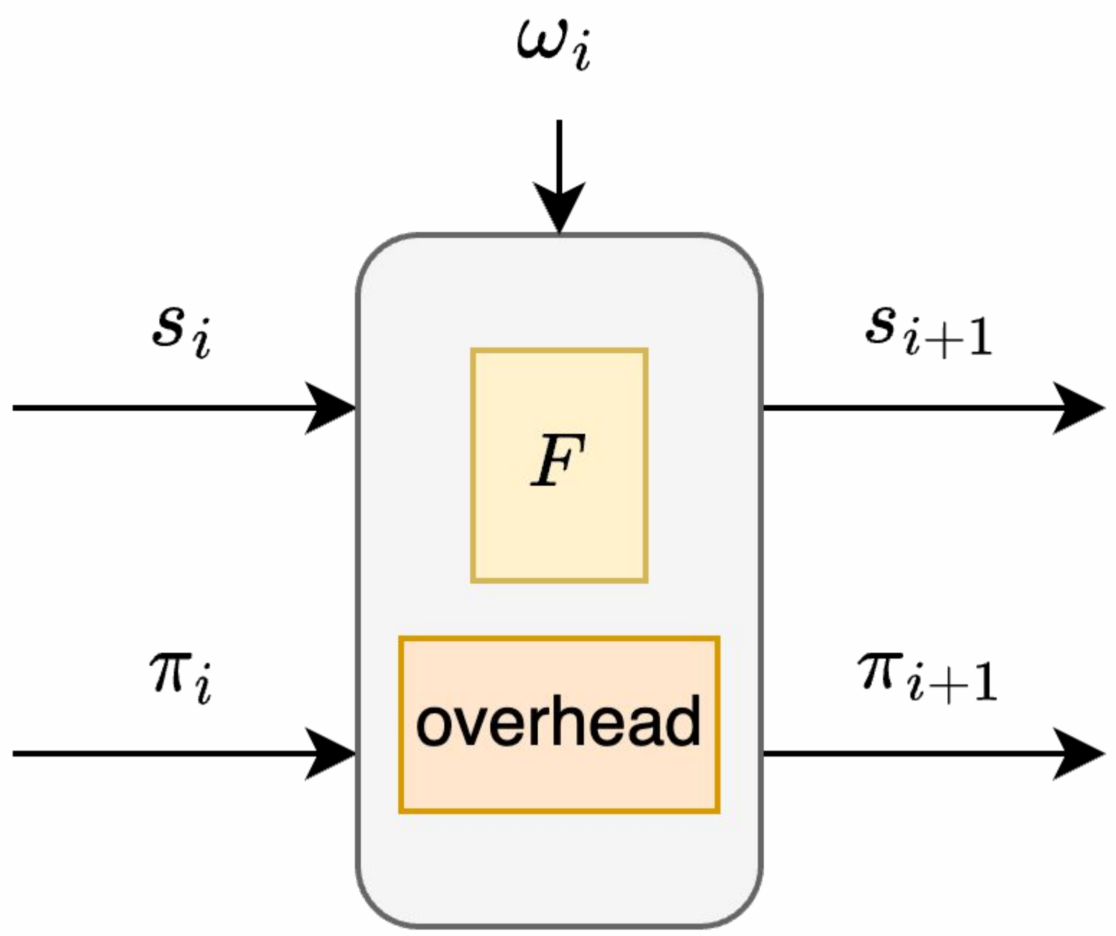
- Dealing with **higher degree gates**:
  - A gate of degree *d* will be relaxed with powers of *u* up to u^d
  - the error's folding equation will be of the form

$$\mathbf{e} = \mathbf{e}' - r\mathbf{t_1} - r^2\mathbf{t_2} \cdots - r^{d-1}\mathbf{t_{d-1}} + r^d\mathbf{e}''$$

  - Cost: each additional degree requires the Prover's message to include 1 new commitment. The Verifier will perform **1 extra point addition per degree**

New Design Space

Back to our original question, how small can we make the recursion overhead?



- Nova's circuit has approximately 20k constraints of which 12k are for point addition

<u>Open question:</u> can the benefit of wider circuits and custom gates outweigh the cost of the extra point additions that the folding verifier (recursion overhead) will perform?

Upcoming Work & Comments

- Sangria is a folding scheme for PLONK and piggybacks on Nova's results to achieve IVC. We can also piggyback on SuperNova!
- Implementation in progress for standard PLONK
- Folding lookup-enabled traces (ultraPLONK)
- (Hopefully) some circuit wizardry very soon, get in touch!

hello@geometry.xyz

@__geometry__

LONDON ST. HELIER SINGAPORE BELGRADE TEL AVIV BOSTON