Mangrove

A Scalable Framework for Folding-based SNARKs

CRYPTO 2024

Generated with Google Gemini

Wilson Nguyen, Trisha Datta, Binyi Chen, Nirvan Tyagi, Dan Boneh

https://ia.cr/2024/416

What are SNARKs?

What are SNARKs?

Prover

(x, w)

Verifier

x

What are SNARKs?

Claim:

I know a w s.t.

$(x, w) \in \mathsf{R}$

Prover

(x, w)

Verifier

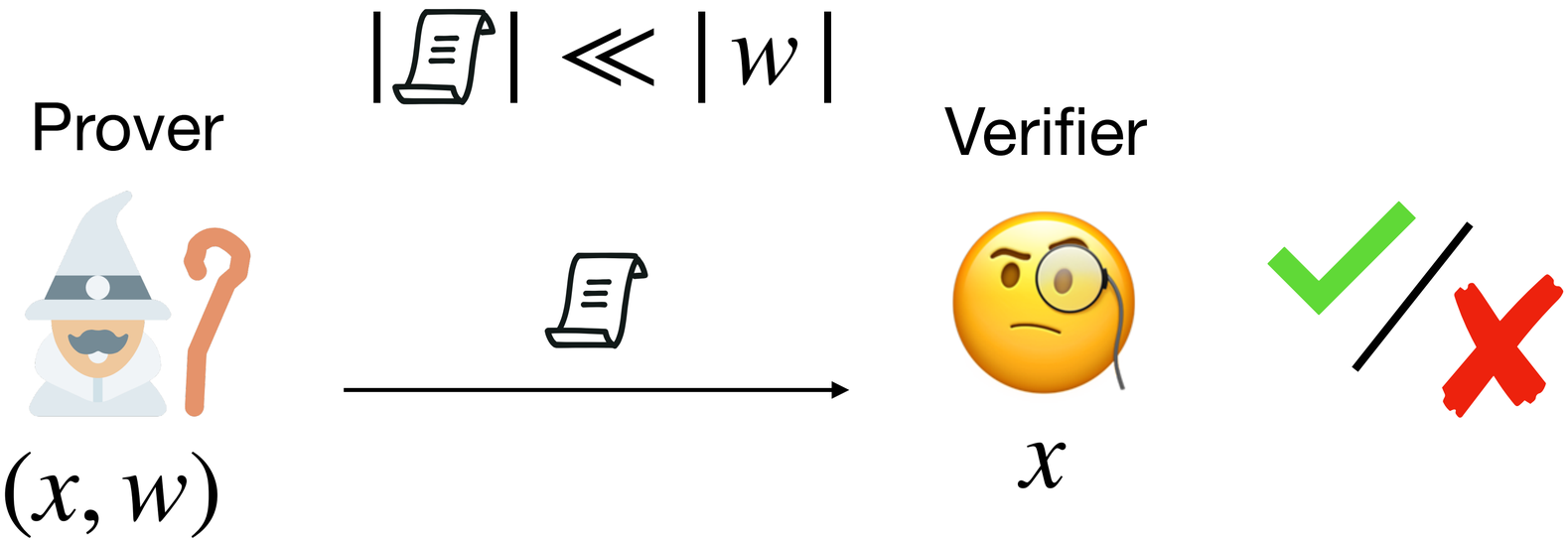
x

What are SNARKs?

Claim:

I know a w s.t.

$(x, w) \in \mathsf{R}$

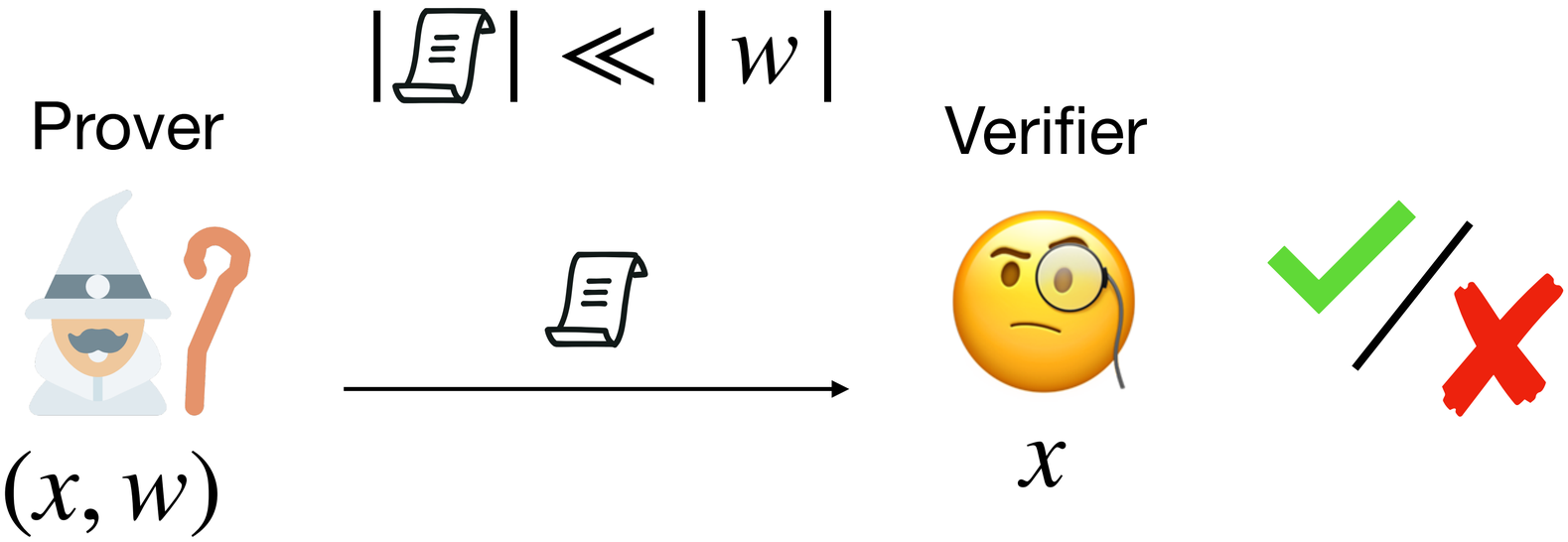


What are SNARKs?

Claim:

I know a w s.t.

$(x, w) \in \mathsf{R}$



$\mathsf{Setup}(1^\lambda) \to \mathsf{rs}$ Generate reference String

$\mathsf{Preprocess}(\mathsf{rs}, C) \to (\mathsf{pk}, \mathsf{vk})$ Generate proving and verifying keys

The Need for Scalability

Filecoin Network 

560 Trillion $\left(2^{49}\right)$
constraints per day [1, 2]

$$(a_0 z_6 + a_2 z_7) \cdot b_2 z_7 = c_3 z_1$$
$$(a_8 z_6 + z_5) \cdot b_5 z_8 = c_3 z_7$$
$$z_9 \cdot (b_7 z_1 + b_{10} z_2) = c_5 z_5$$

The Need for Scalability

Filecoin Network

560 Trillion (2^{49})
constraints per day [1, 2]

$$(a_0 z_6 + a_2 z_7) \cdot b_2 z_7 = c_3 z_1$$
$$(a_8 z_6 + z_5) \cdot b_5 z_8 = c_3 z_7$$
$$z_9 \cdot (b_7 z_1 + b_{10} z_2) = c_5 z_5$$

Hard Limit → **6-7 million** proofs, each for **100 million** constraints

The Need for Scalability

Filecoin Network

560 Trillion (2^{49})
constraints per day [1, 2]

$$(a_0 z_6 + a_2 z_7) \cdot b_2 z_7 = c_3 z_1$$
$$(a_8 z_6 + z_5) \cdot b_5 z_8 = c_3 z_7$$
$$z_9 \cdot (b_7 z_1 + b_{10} z_2) = c_5 z_5$$

Hard Limit → **6-7 million** proofs, each for **100 million** constraints

Verifiable ML

700 x 700 Matrix Mul
685 million constraints [4]

$$\left(A \right)\left(B \right) = \left(C \right)$$

Barriers to Scalability

Barriers to Scalability

Proving Memory

Monolithic SNARKs

- Plonk, Marlin, Groth16, STARKs
- Linear Memory (TBs RAM)[1,5,8]

Barriers to Scalability

Proving Memory

Monolithic SNARKs

- Plonk, Marlin, Groth16, STARKs
- Linear Memory (TBs RAM)[1,5,8]

Streaming SNARKs (**Gemini)**

< 1.2 GB for 2^{31} constraints [2]

< 1.7 days

Distributed SNARKs (**DIZK)**

< 12.8 hr for 2^{31} constraints [3,4]

5.2 TB RAM with 20 machines

Barriers to Scalability

Proving Memory

Monolithic SNARKs

- Plonk, Marlin, Groth16, STARKs
- Linear Memory (TBs RAM)[1,5,8]

Streaming SNARKs (**Gemini)**

< 1.2 GB for 2^{31} constraints [2]

< 1.7 days

Distributed SNARKs (**DIZK)**

< 12.8 hr for 2^{31} constraints [3,4]

5.2 TB RAM with 20 machines

Limited by SRS size

Barriers to Scalability

Proving Memory

Monolithic SNARKs

- Plonk, Marlin, Groth16, STARKs
- Linear Memory (TBs RAM)[1,5,8]

Streaming SNARKs (**Gemini)**

< 1.2 GB for 2^{31} constraints [2]

< 1.7 days

Distributed SNARKs (**DIZK)**

< 12.8 hr for 2^{31} constraints [3,4]

5.2 TB RAM with 20 machines

Structured Reference Strings

$$\underbrace{\left(G,\ \tau G,\ \tau^2 G,\ \ldots,\ \tau^n G\right)}_{\textit{Powers of Tau}}$$

- **Limit** the # of constraints proven
  - Largest SRS < 2^{27} constraints [1]
- **Difficult** setup ceremonies [1,6, 7]

Limited by SRS size

Scalable SNARKs

- Prove arbitrary number of constraints
- Transparent setup
- Minimal hardware requirements (can run even on a laptop)
- Memory efficient streaming prover
- Support for distributed proving

Mangrove

Techniques

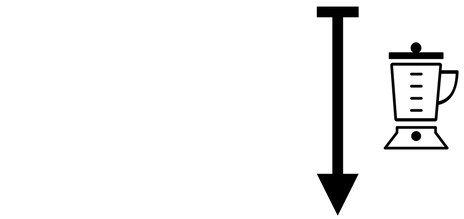
Techniques

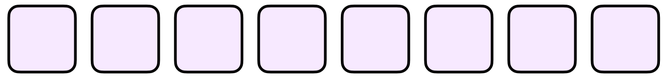
$$C(x, w) = 0$$ Arbitrary Circuit SAT

Techniques

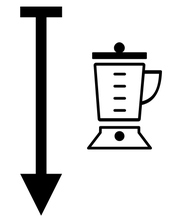
Uniform Compiler

$$C(x, w) = 0$$





Arbitrary Circuit SAT

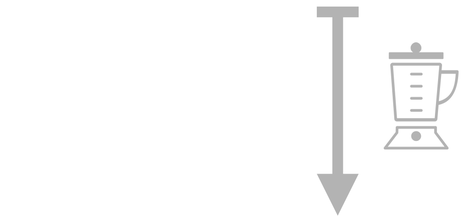


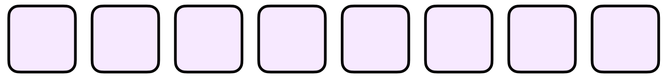
Small & Identical chunks
(Polynomial Relation)

Techniques

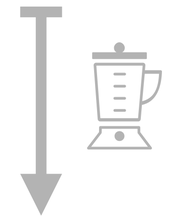
Uniform Compiler

$$C(x, w) = 0$$





Arbitrary Circuit SAT



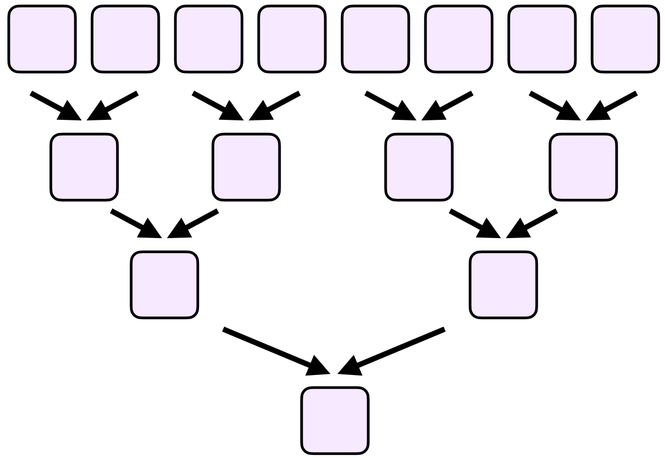
Small & Identical chunks
(Polynomial Relation)

Techniques

Uniform Compiler

$C(x, w) = 0$

Folding Scheme for Polynomial Relations



Arbitrary Circuit SAT

Small & Identical chunks (Polynomial Relation)

Single chunk

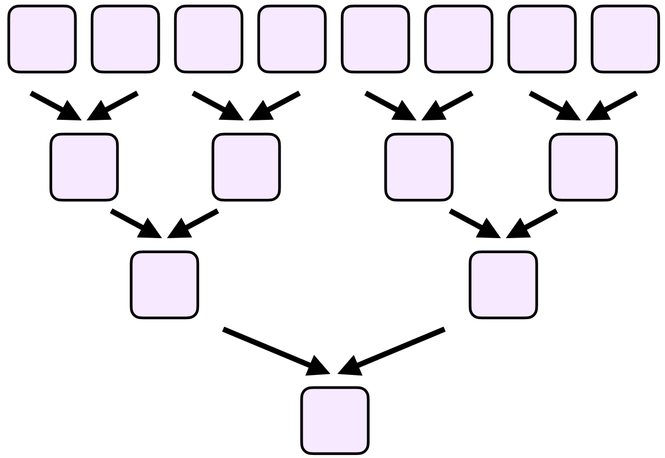
Techniques

Uniform Compiler

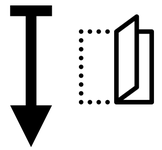
$C(x, w) = 0$

Arbitrary Circuit SAT

Folding Scheme for Polynomial Relations



Small & Identical chunks
(Polynomial Relation)



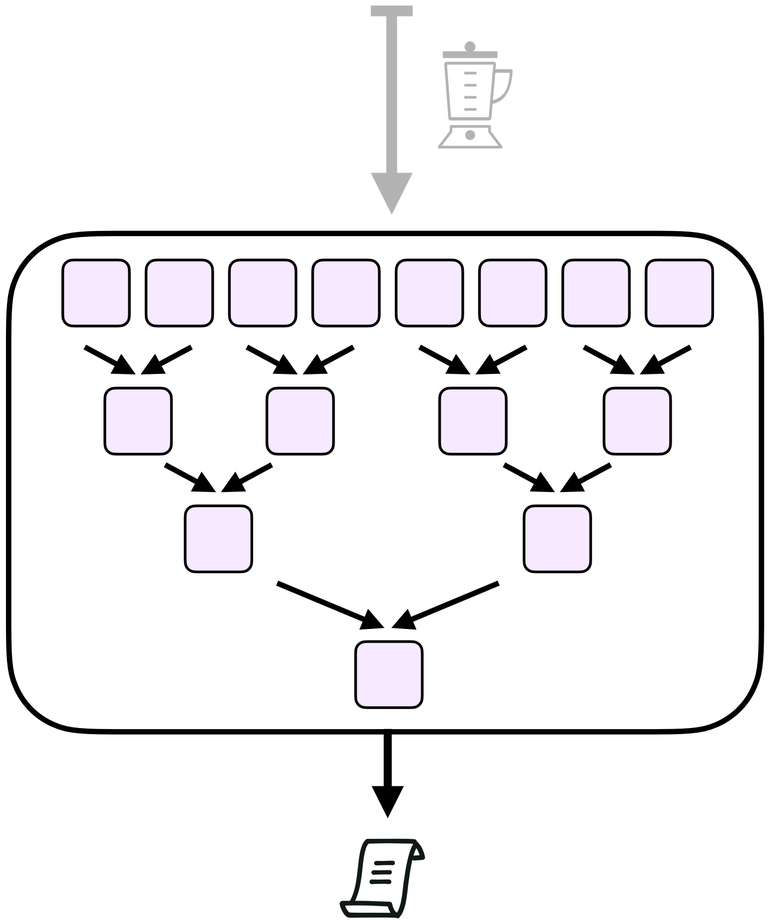
Single chunk

Techniques

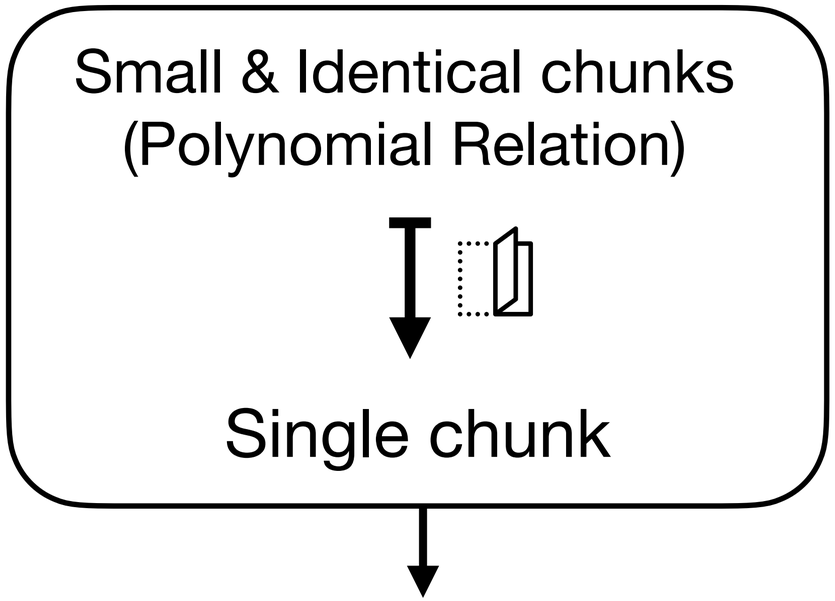
$C(x, w) = 0$

Arbitrary Circuit SAT

Uniform Compiler



Folding Scheme for Polynomial Relations



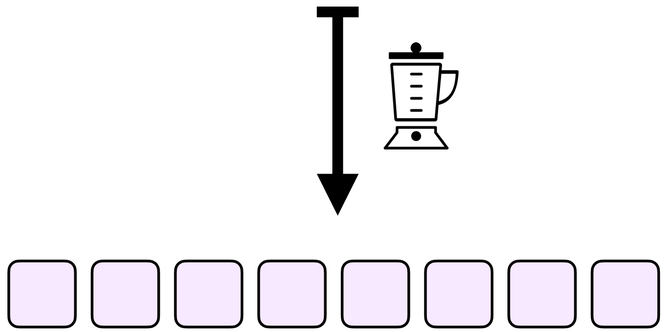
Proof Carrying Data [10]

Succinct Proof

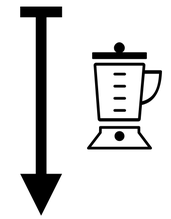
Techniques

Uniform Compiler

$$C(x, w) = 0$$

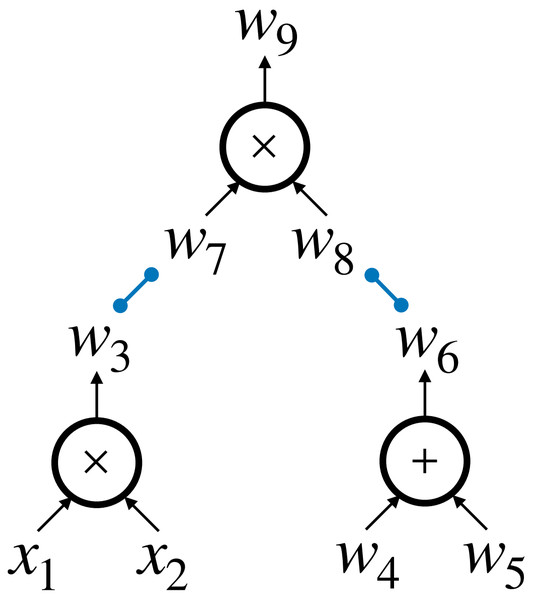


Arbitrary Circuit SAT

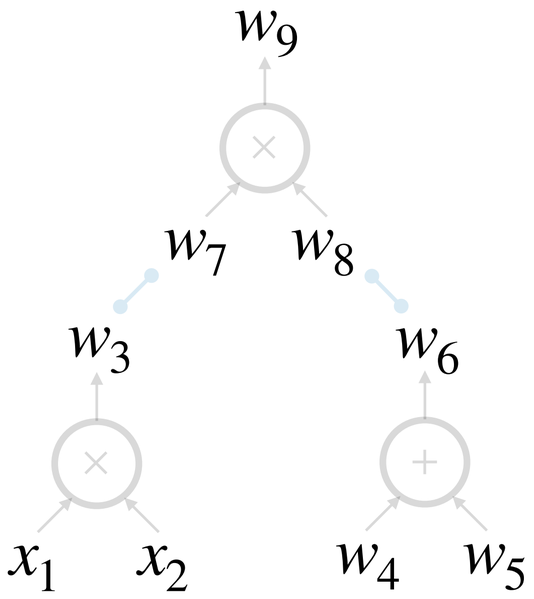


Small & Identical chunks
(Polynomial Relation)

Uniform Compiler - Encoding Circuits



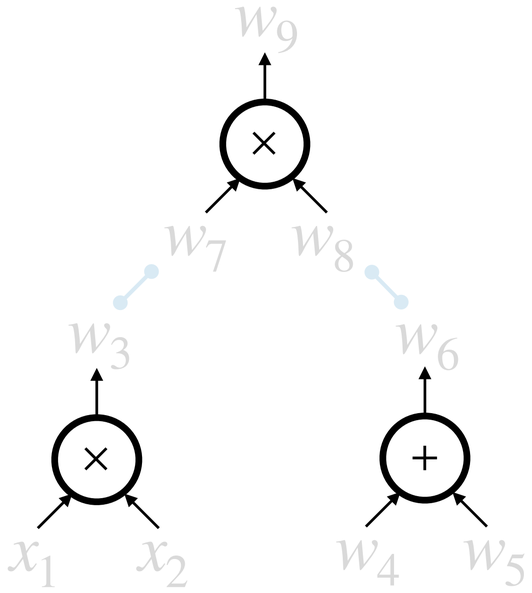
Uniform Compiler - Encoding Circuits



x_1	x_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9

Values $(x,\ w)$

Uniform Compiler - Encoding Circuits



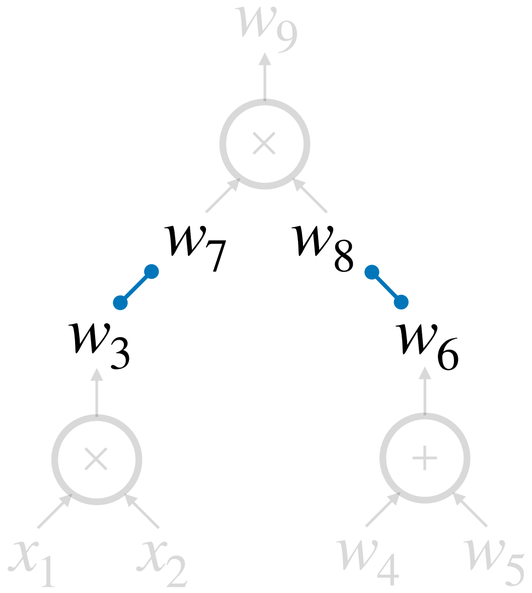
1	0	1

Binary Vector

x_1	x_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9

Values $(x,\ w)$

Uniform Compiler - Encoding Circuits



1	0	1

Binary Vector

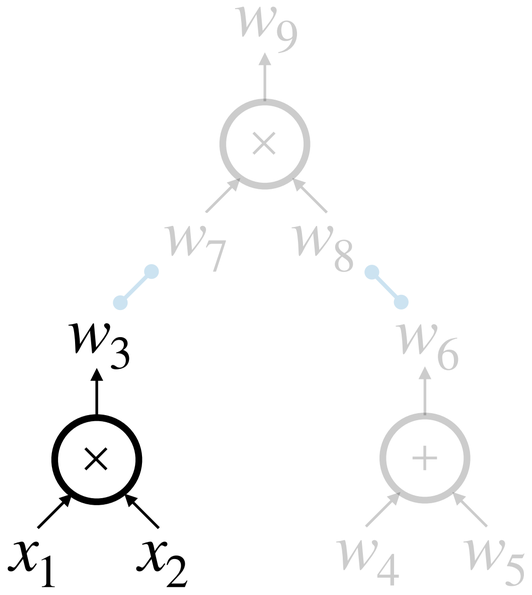
1	2	7	4	5	8	3	6	9

Permutation

x_1	x_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9

Values $(x,\ w)$

Uniform Compiler - Encoding Circuits



1	0	1

Binary Vector

1	2	7	4	5	8	3	6	9

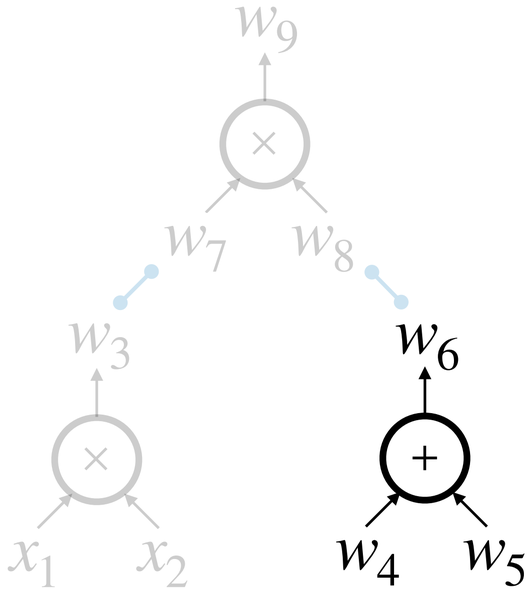
Permutation

x_1	x_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9

Values $(x,\ w)$

Gate Constraint | 1 | x_1 x x_2 = w_3 |
|---|---|

Uniform Compiler - Encoding Circuits



1	0	1

Binary Vector

1	2	7	4	5	8	3	6	9

Permutation

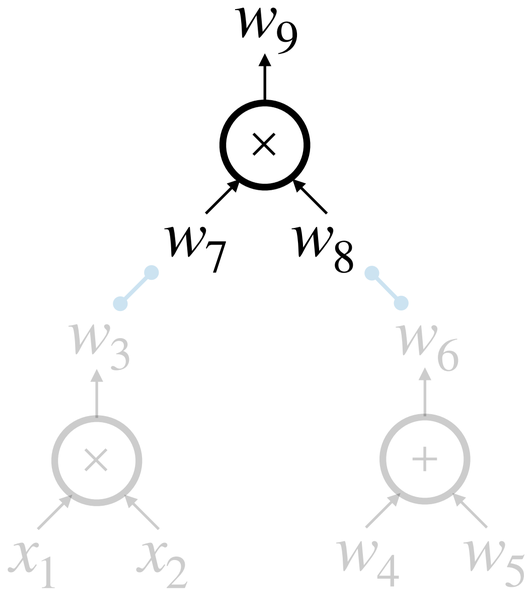
x_1	x_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9

Values $(x,\ w)$

Gate Constraint

1	x_1 x x_2 = w_3
0	w_4 + w_5 = w_6

Uniform Compiler - Encoding Circuits



1	0	1

Binary Vector

1	2	7	4	5	8	3	6	9

Permutation

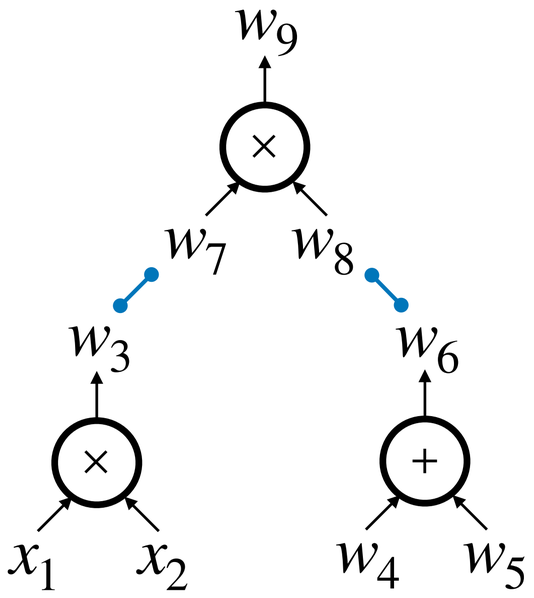
x_1	x_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9

Values $(x,\ w)$

Gate Constraint

1	x_1	x	x_2	=	w_3
0	w_4	+	w_5	=	w_6
1	w_7	x	w_8	=	w_9

Uniform Compiler - Encoding Circuits



1	0	1

Binary Vector

1	2	7	4	5	8	3	6	9

Permutation

x_1	x_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9

Values $(x,\ w)$

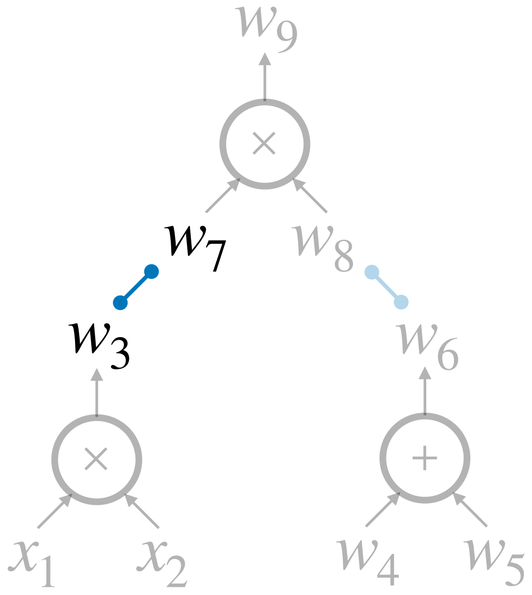
=

Copy Constraint:

x_1	x_2	w_7	w_4	w_5	w_8	w_3	w_6	w_9

Permuted Values

Uniform Compiler - Encoding Circuits



1	0	1	Binary Vector

1	2	7	4	5	8	3	6	9	Permutation

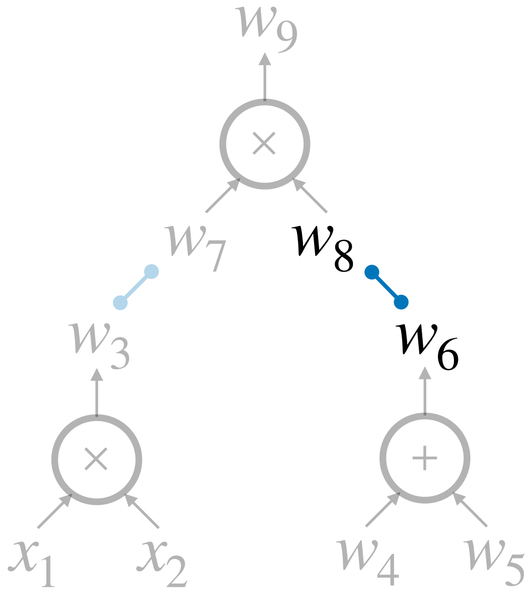
x_1	x_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	Values $(x,\ w)$

$=$ $=$

Copy Constraint:

x_1	x_2	w_7	w_4	w_5	w_8	w_3	w_6	w_9	Permuted Values

Uniform Compiler - Encoding Circuits



1	0	1

Binary Vector

1	2	7	4	5	8	3	6	9

Permutation

x_1	x_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9

Values $(x,\ w)$

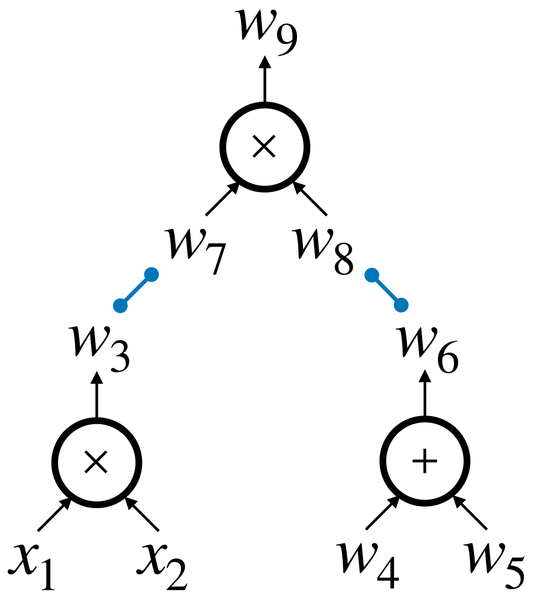
= =

Copy Constraint:

x_1	x_2	w_7	w_4	w_5	w_8	w_3	w_6	w_9

Permuted Values

Uniform Compiler - Chunking



1	0	1

Binary Vector

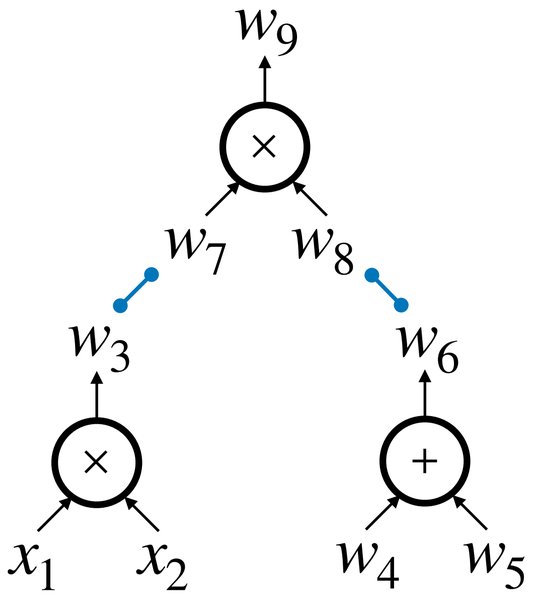
1	2	7	4	5	8	3	6	9

Permutation

x_1	x_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9

Values $(x,\ w)$

Uniform Compiler - Chunking

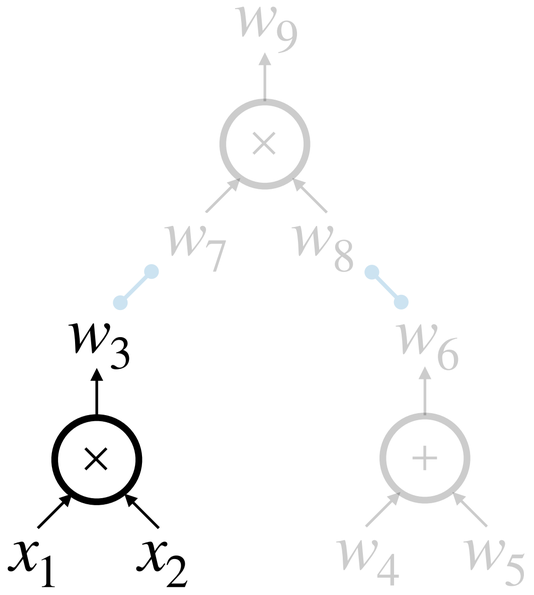


1	0	1	Binary Vector

1	2	7	4	5	8	3	6	9	Permutation

x_1	x_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	Values $(x,\ w)$

Uniform Compiler - Chunking



1	0	1

Binary Vector

1	2	7	4	5	8	3	6	9

Permutation

x_1	x_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9

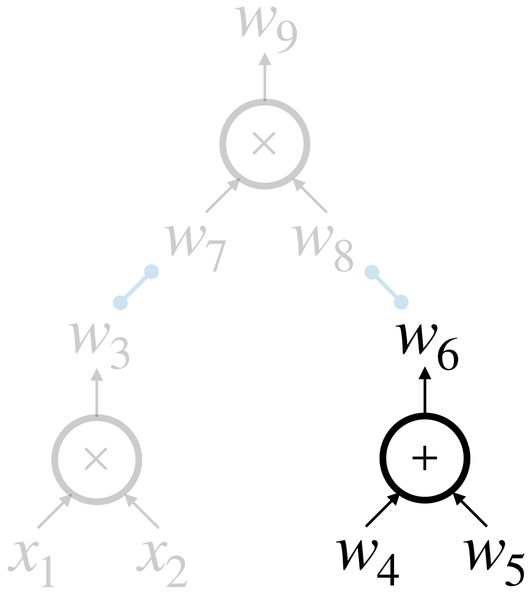
Values $(x,\ w)$

1	1	2	7
x_1	x_2	w_3	

0	4	5	8
w_4	w_5	w_6	

1	3	6	9
w_7	w_8	w_9	

Uniform Compiler - Chunking



1	0	1

Binary Vector

1	2	7	4	5	8	3	6	9

Permutation

x_1	x_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9

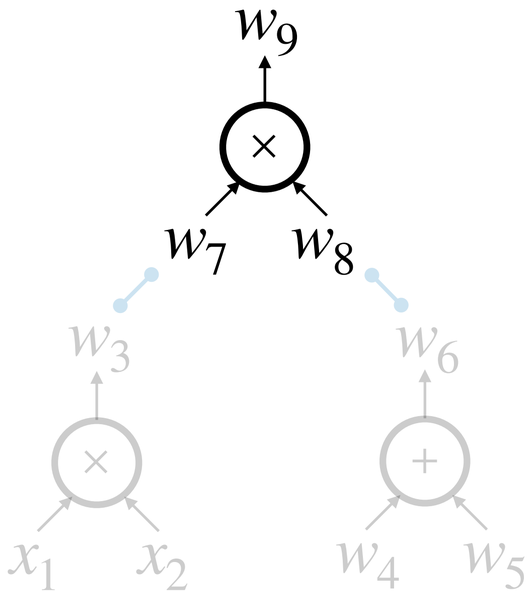
Values $(x,\ w)$

1	1	2	7
x_1	x_2	w_3	

0	4	5	8
w_4	w_5	w_6	

1	3	6	9
w_7	w_8	w_9	

Uniform Compiler - Chunking



1	0	1

Binary Vector

1	2	7	4	5	8	3	6	9

Permutation

x_1	x_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9

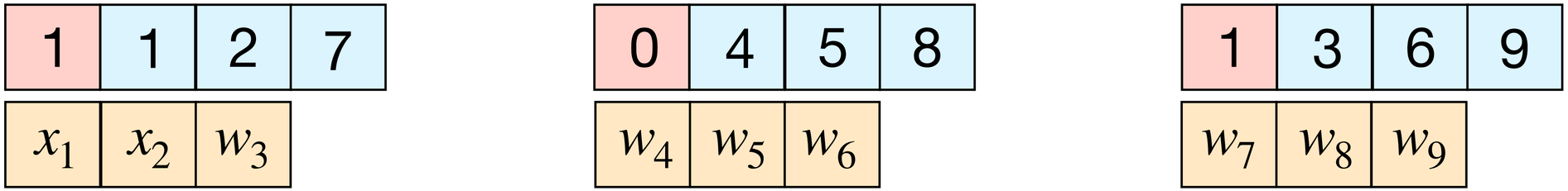
Values $(x,\ w)$

1	1	2	7
x_1	x_2	w_3	

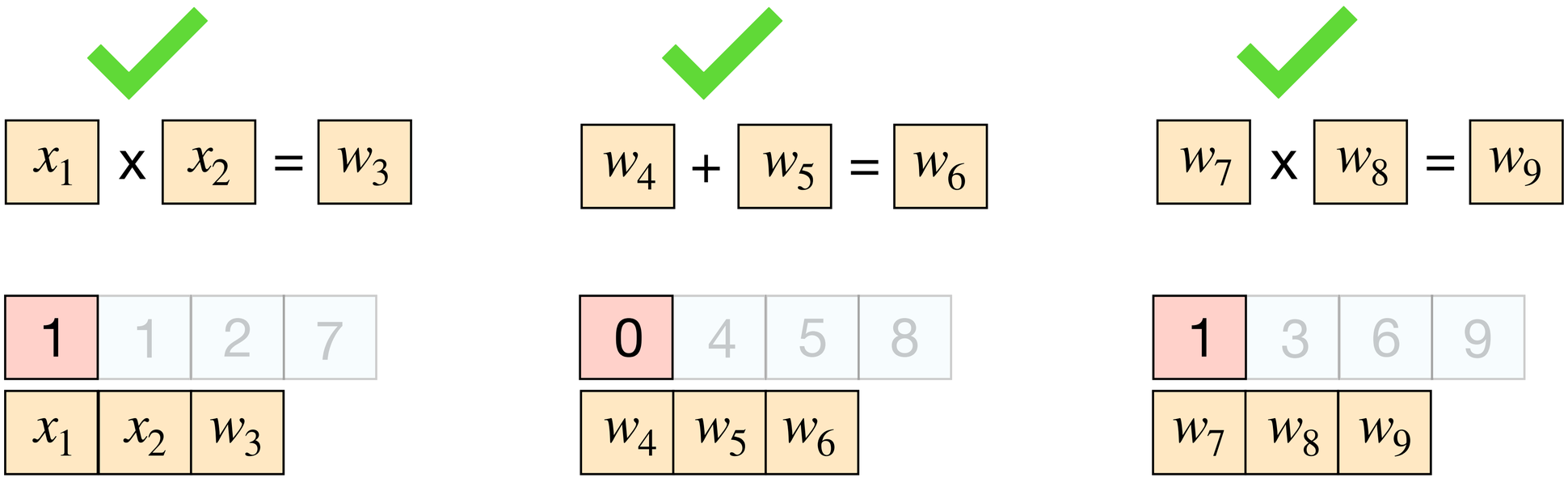
0	4	5	8
w_4	w_5	w_6	

1	3	6	9
w_7	w_8	w_9	

Uniform Compiler - Chunking

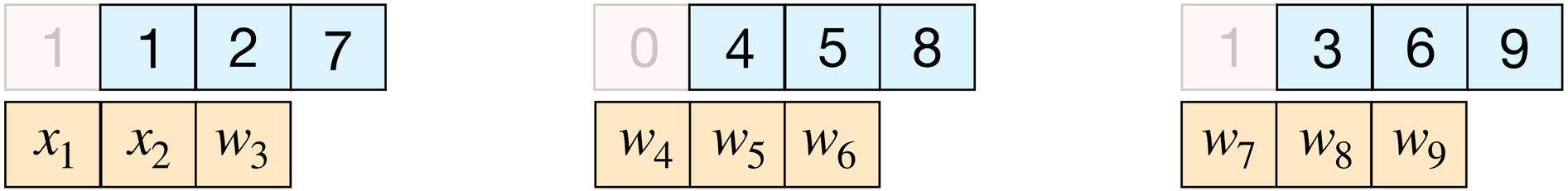


Uniform Compiler - Chunking

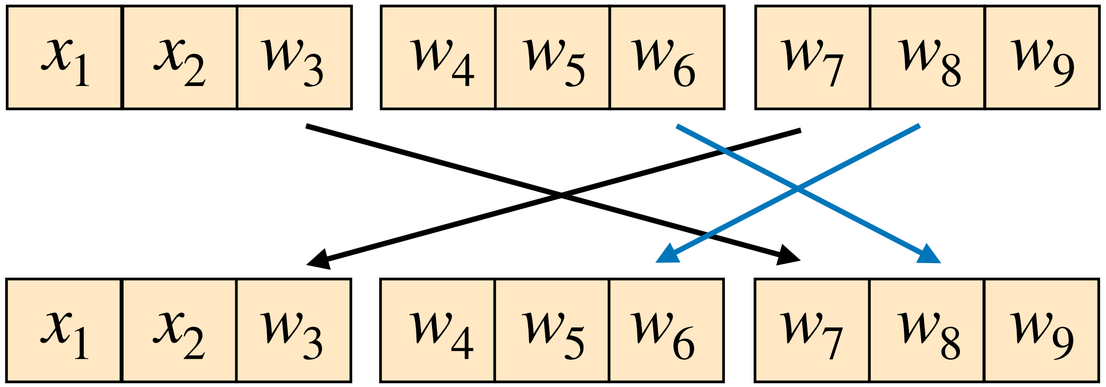


Uniform Compiler - Chunking

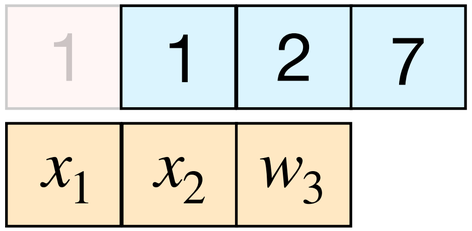
Copy

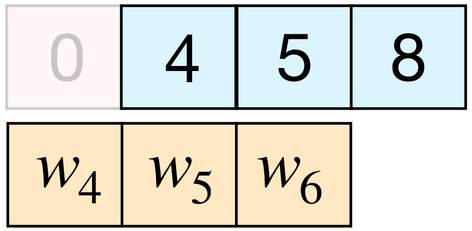


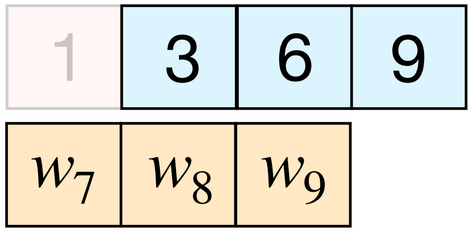
Uniform Compiler - Chunking



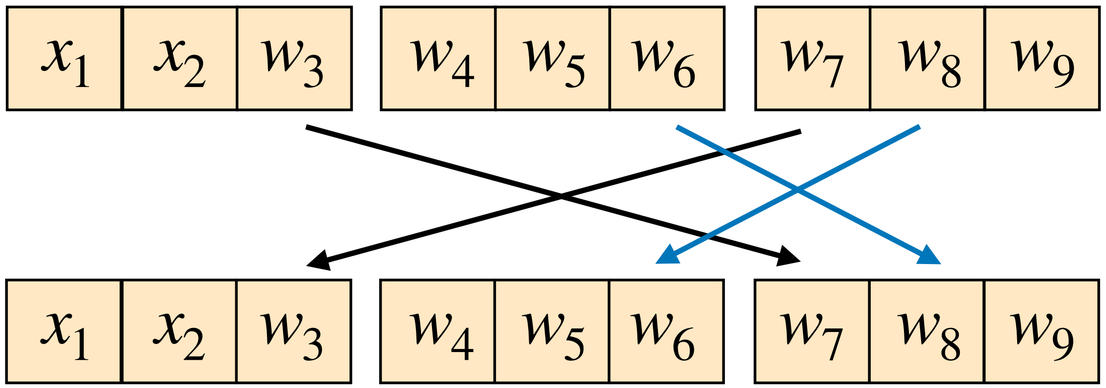
Copy





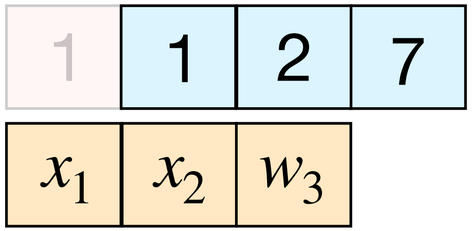


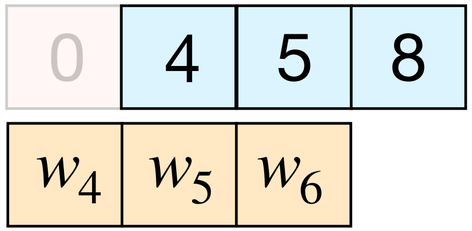
Uniform Compiler - Chunking

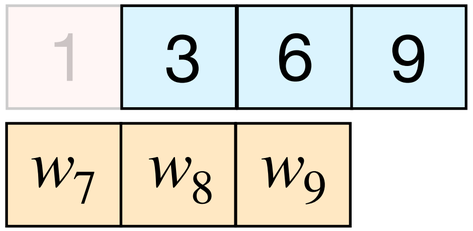


Problem:
Copy constraints
requires global access!

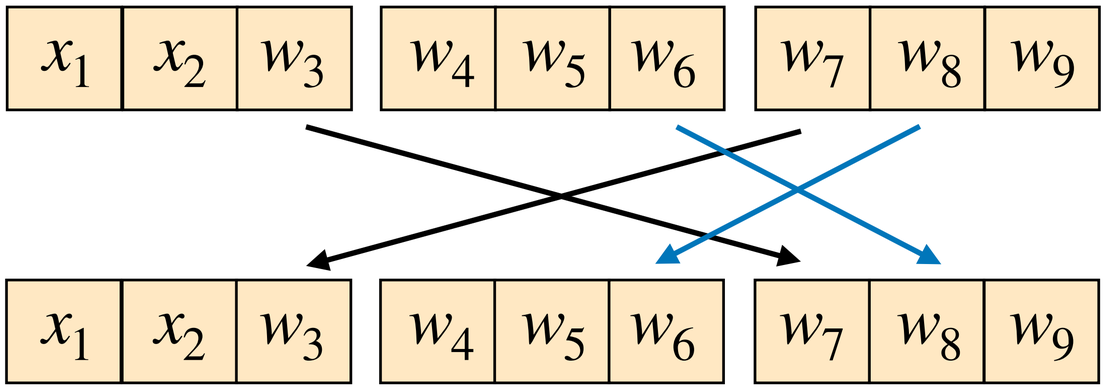
Copy







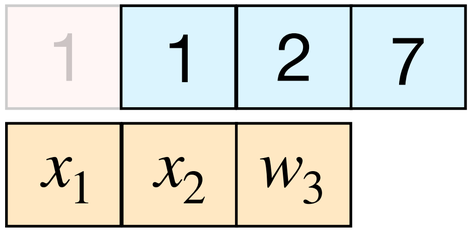
Uniform Compiler - Chunking

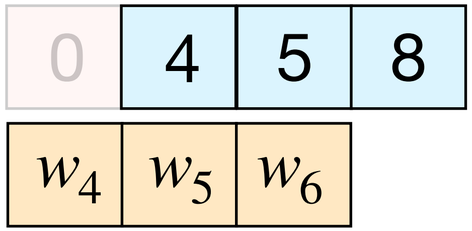


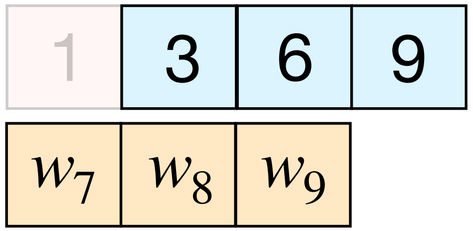
Problem:
Copy constraints
requires global access!

Solution:
Multiset Arguments
[11, 12, 13, 14]

Copy







Uniform Compiler - Multiset Argument

Prover

Verifier

$$v = \left(\begin{array}{|c|c|c|} \hline x_1 & x_2 & w_3 \\ \hline \end{array} \right.$$

1	1	2	7

w_4	w_5	w_6

0	4	5	8

w_7	w_8	w_9

1	3	6	9

$\left. \right)$

Uniform Compiler - Multiset Argument

Prover

$\text{Commit}(v) = h_v$

Verifier

$$v = \left(\begin{array}{|c|c|c|} \hline x_1 & x_2 & w_3 \\ \hline \end{array} \right.$$

1	1	2	7

0	4	5	8

$$\begin{array}{|c|c|c|} \hline w_4 & w_5 & w_6 \\ \hline \end{array}$$

1	3	6	9

$$\left. \begin{array}{|c|c|c|} \hline w_7 & w_8 & w_9 \\ \hline \end{array} \right)$$

Uniform Compiler - Multiset Argument

Prover

Verifier

$\text{Commit}(v) = h_v$

Randomness r

$$v = \left(\begin{array}{|c|c|c|} \hline 1 & 1 & 2 \\ \hline x_1 & x_2 & w_3 \\ \hline \end{array} \begin{array}{|c|c|c|} \hline 0 & 4 & 5 \\ \hline w_4 & w_5 & w_6 \\ \hline \end{array} \begin{array}{|c|c|c|} \hline 1 & 3 & 6 \\ \hline w_7 & w_8 & w_9 \\ \hline \end{array} \right)$$

Uniform Compiler - Multiset Argument

Prover

Verifier

$\text{Commit}(v) = h_v$

Randomness r

Local Products

1	1	2	7
x_1	x_2	w_3	p_1

0	4	5	8
w_4	w_5	w_6	p_2

1	3	6	9
w_7	w_8	w_9	p_3

Global Constraints $\prod p_i = 1$

Uniform Compiler - Multiset Argument

Prover

Verifier

$\text{Commit}(v) = h_v$

Randomness r

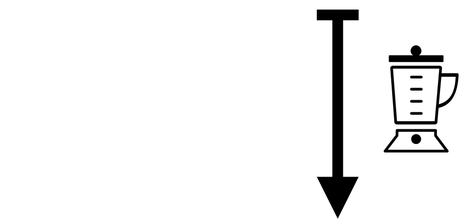
	1	1	2	7	0	4	5	8	1	3	6	9
Values	x_1	x_2	w_3	p_1	w_4	w_5	w_6	p_2	w_7	w_8	w_9	p_3

Global Constraints $\prod p_i = 1$ and $h_v = \text{Commit}(v)$

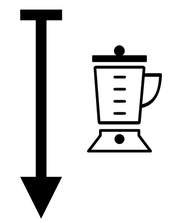
Techniques

Uniform Compiler

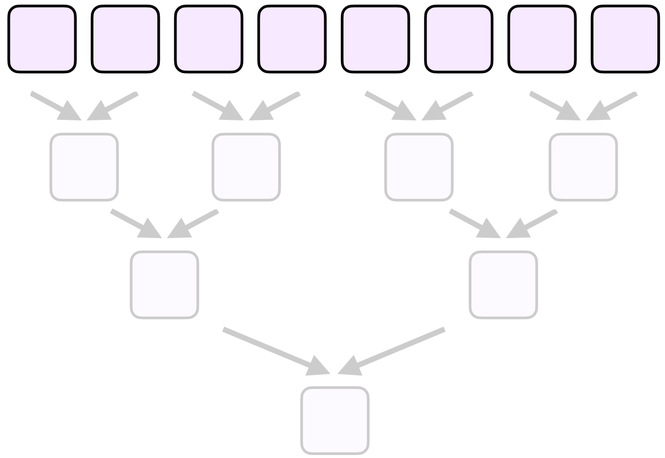
$C(x, w) = 0$

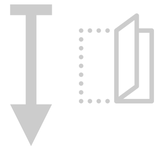


Arbitrary Circuit SAT



Small & Identical chunks
(Polynomial Relation)





Single chunk

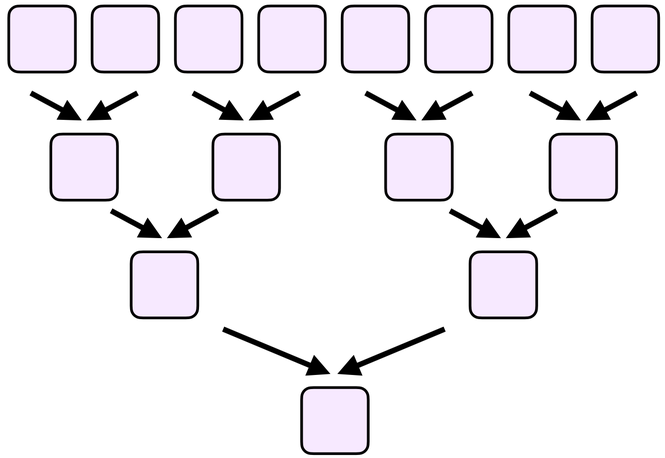
Folding Scheme
for Polynomial
Relations

Techniques

$C(x, w) = 0$

Arbitrary Circuit SAT

Uniform Compiler



Folding Scheme for Polynomial Relations

Small & Identical chunks (Polynomial Relation)

Single chunk

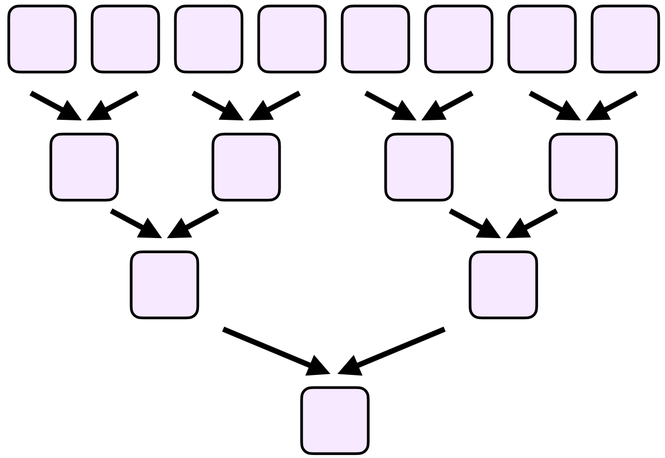
Techniques

Uniform Compiler

Circuit SAT

Produces a NARK
(not succinct)
with recursive structure

Folding Scheme
for Polynomial
Relations

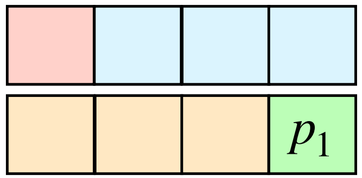


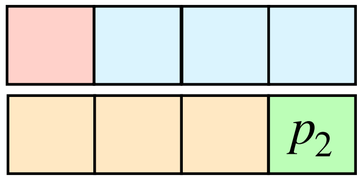
Small & Identical chunks
(Polynomial Relation)

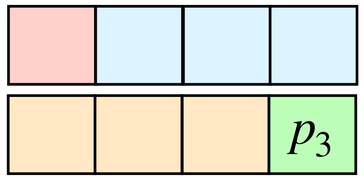
Single chunk

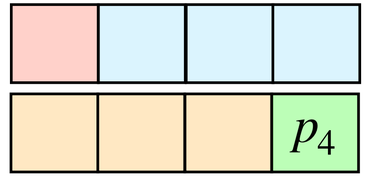
Global Constr $\prod p_i = 1$ and $h_v = \text{Commit}(v)$

Local Constr Gate Check + Local Product









Global Constr $\prod p_i = 1$ and $h_v = \text{Commit}(v)$

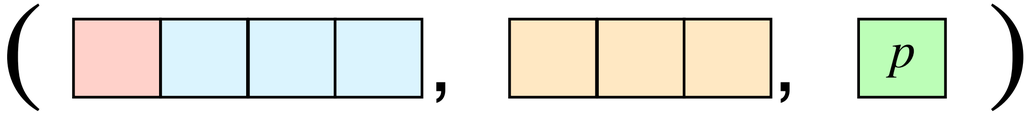
Local Constr Gate Check + Local Product

Global Constr $\prod p_i = 1$ and $h_v = \text{Commit}(v)$

Local Constr Gate Check + Local Product

Local Constraints

Circuit C Value v Prod



- Gate Check
- Local Product

Global Constr $\prod p_i = 1$ and $h_v = \text{Commit}(v)$

Local Constr Gate Check + Local Product

Local Constraints

Circuit C Value v Prod

$\left(\square, \square, p \right)$

Linearly Homomorphic (Pedersen)

- Gate Check
- Local Product

Global Constr $\prod p_i = 1$ and $h_v = \text{Commit}(v)$

Local Constr Gate Check + Local Product

Local Constraints

Circuit c Value v Prod

$\left(\square, \square, p \right)$

Linearly Homomorphic (Pedersen) ⟶ $(\overline{c}, \overline{v}, p)$

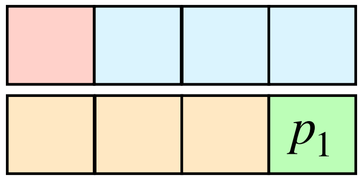
- Gate Check
- Local Product

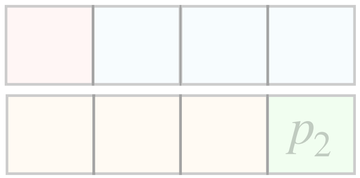
⟶ $f(c, v, p) \stackrel{?}{=} 0$

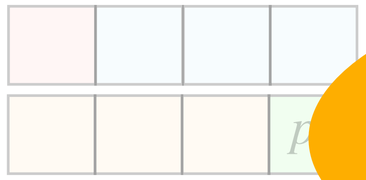
Polynomial Relation

Global Constr $\prod p_i = 1$ and $h_v = \text{Commit}(v)$

Local Constr Gate Check + Local Product







Can be batched with folding!

Local Constraints

Circuit c Value v Prod

$$\left(\square, \square, p \right)$$

Linearly Homomorphic (Pedersen)

$$(\overline{c}, \overline{v}, p)$$

Polynomial Relation

- Gate Check
- Local Product

$$f(c, v, p) \stackrel{?}{=} 0$$

Global Constr $\prod p_i = 1$ and $h_v = \text{Commit}(v)$

Local Constr Gate Check + Local Product

Polynomial Relation

$$(\overline{c}, \overline{v}, p)$$

$$f(c, v, p) \stackrel{?}{=} 0$$

Global Constr $\prod p_i = 1$ and $h_v = \text{Commit}(v)$

Local Constr Gate Check + Local Product

Claim: $(\overline{c}_1, \overline{v}_1, p_1), (\overline{c}_2, \overline{v}_2, p_2)$

Polynomial Relation

$(\overline{c}, \overline{v}, p)$

$f(c, v, p) \stackrel{?}{=} 0$

Global Constr $\prod p_i = 1$ and $h_v = \text{Commit}(v)$

Local Constr Gate Check + Local Product

Claim: $(\overline{c}_1, \overline{v}_1, p_1), (\overline{c}_2, \overline{v}_2, p_2)$

$\text{Fold}_V(\cdot, \cdot, \text{advice})$

Polynomial Relation

$(\overline{c}, \overline{v}, p)$

$f(c, v, p) \stackrel{?}{=} 0$

Global Constr $\prod p_i = 1$ and $h_v = \text{Commit}(v)$

Local Constr Gate Check + Local Product

Claim: $(\overline{c}_1, \overline{v}_1, p_1), (\overline{c}_2, \overline{v}_2, p_2)$

$\text{Fold}_V(\cdot, \cdot, \text{advice})$

New Claim: $(\overline{c}, \overline{v}, p) \in \mathcal{L}(\text{PolyRel})$

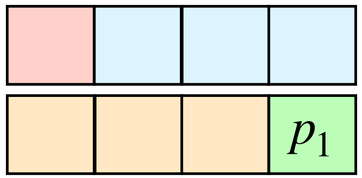
Polynomial Relation

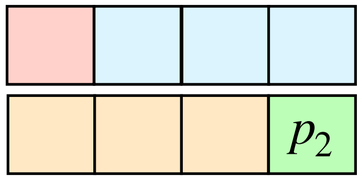
$(\overline{c}, \overline{v}, p)$

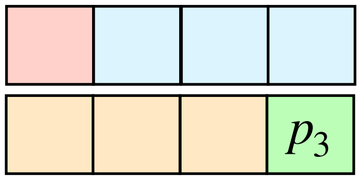
$f(c, v, p) \stackrel{?}{=} 0$

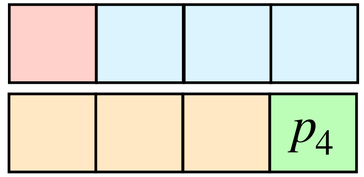
Global Constr $\prod p_i = 1$ and $h_v = \text{Commit}(v)$

Local Constr Gate Check + Local Product



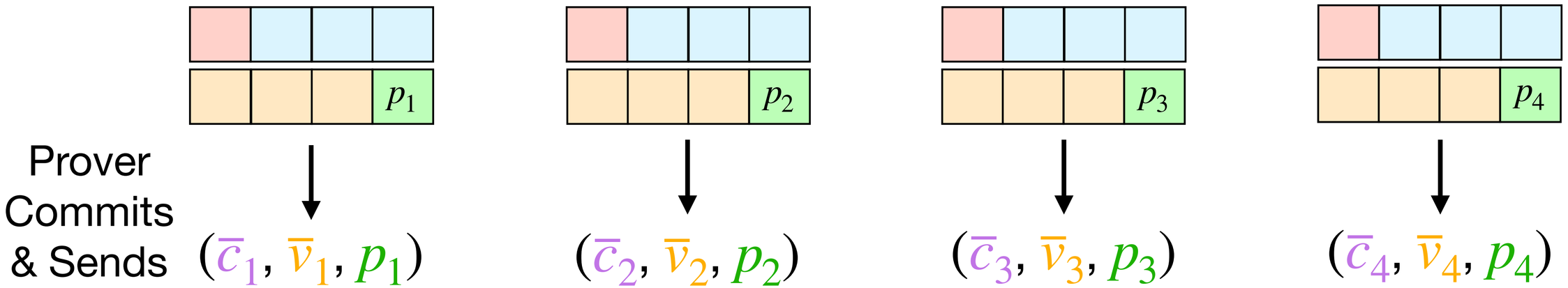






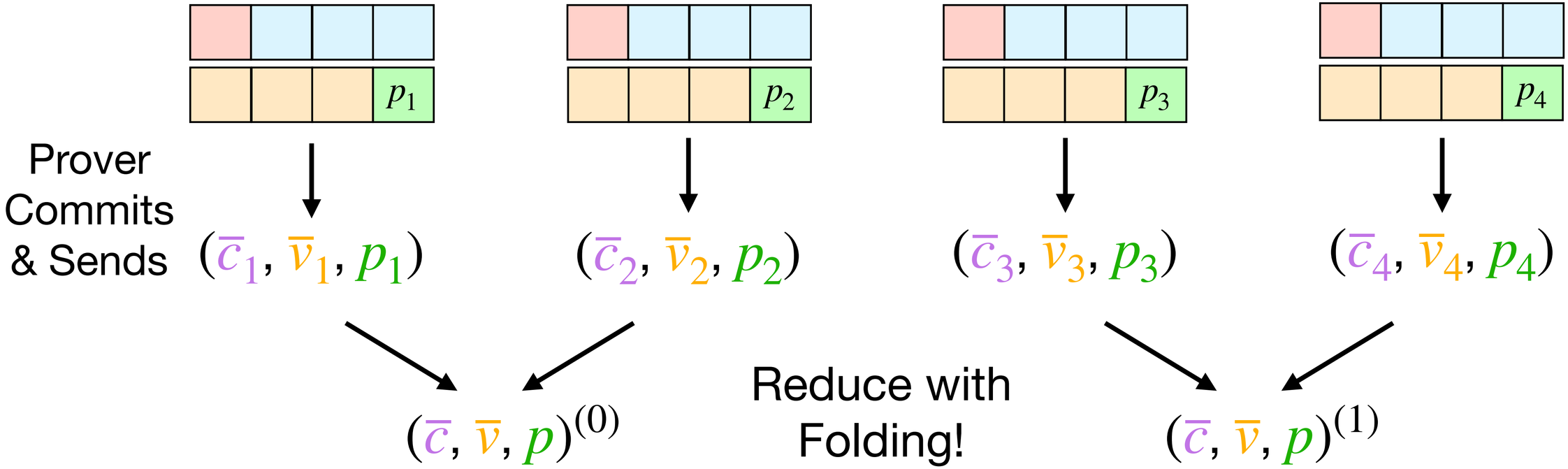
Global Constr $\prod p_i = 1$ and $h_v = \text{Commit}(v)$

Local Constr Gate Check + Local Product



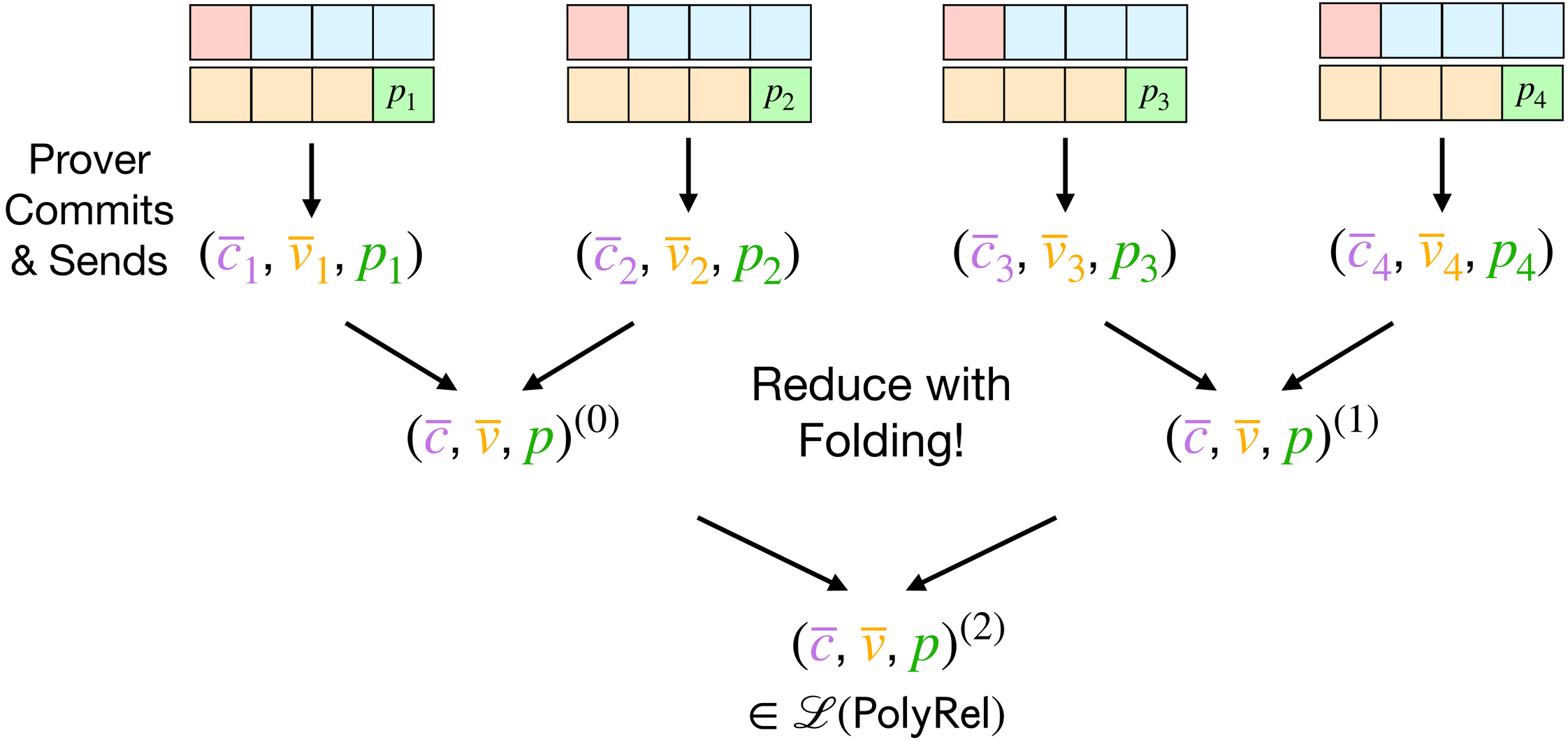
Global Constr $\prod p_i = 1$ and $h_v = \text{Commit}(v)$

Local Constr Gate Check + Local Product

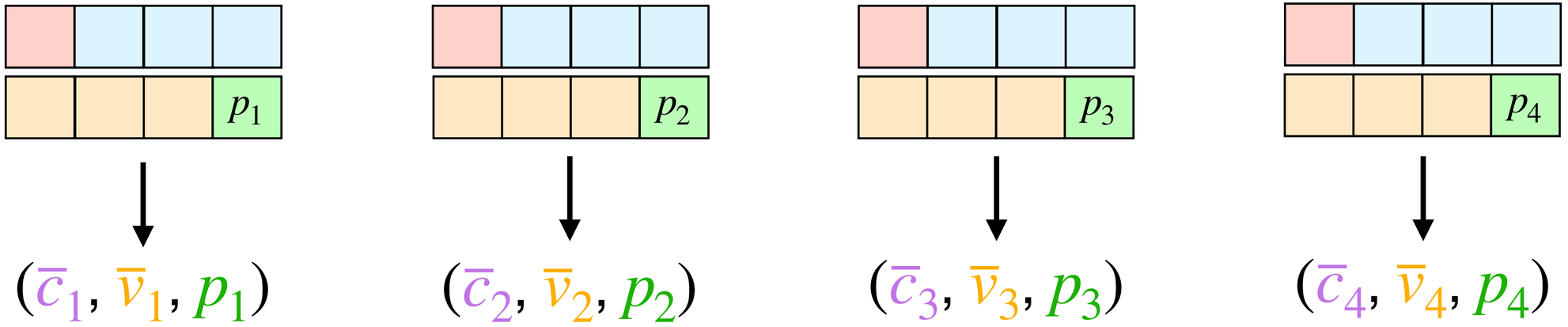


Global Constr $\prod p_i = 1$ and $h_v = \text{Commit}(v)$

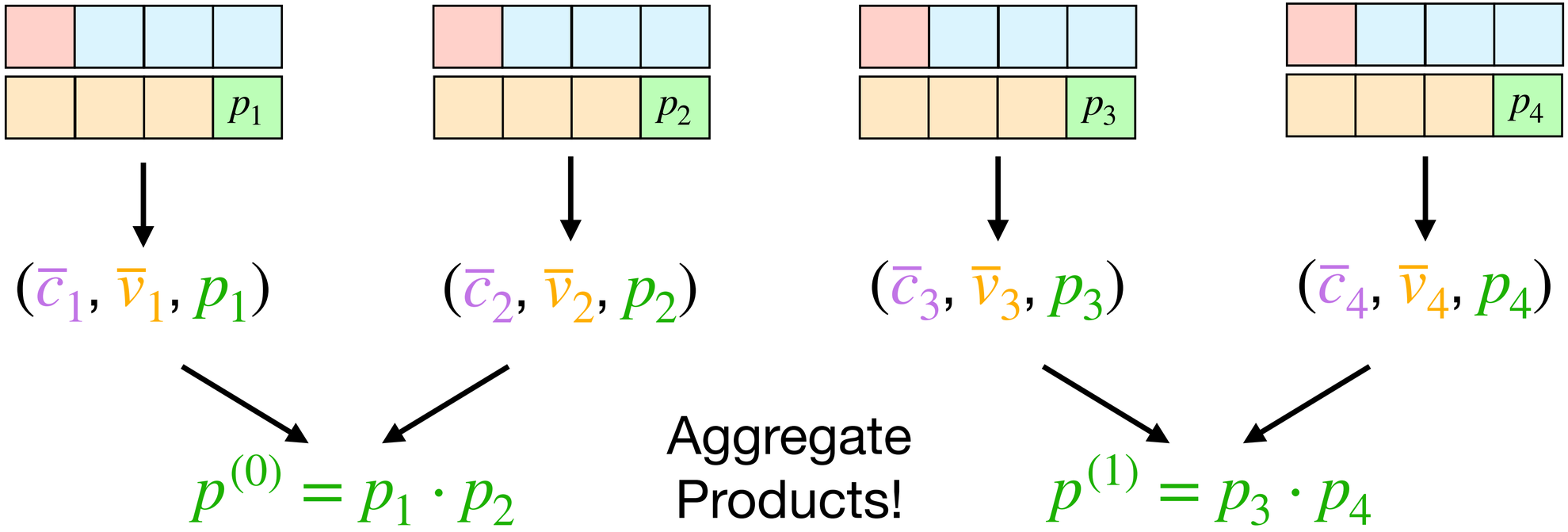
Local Constr Gate Check + Local Product



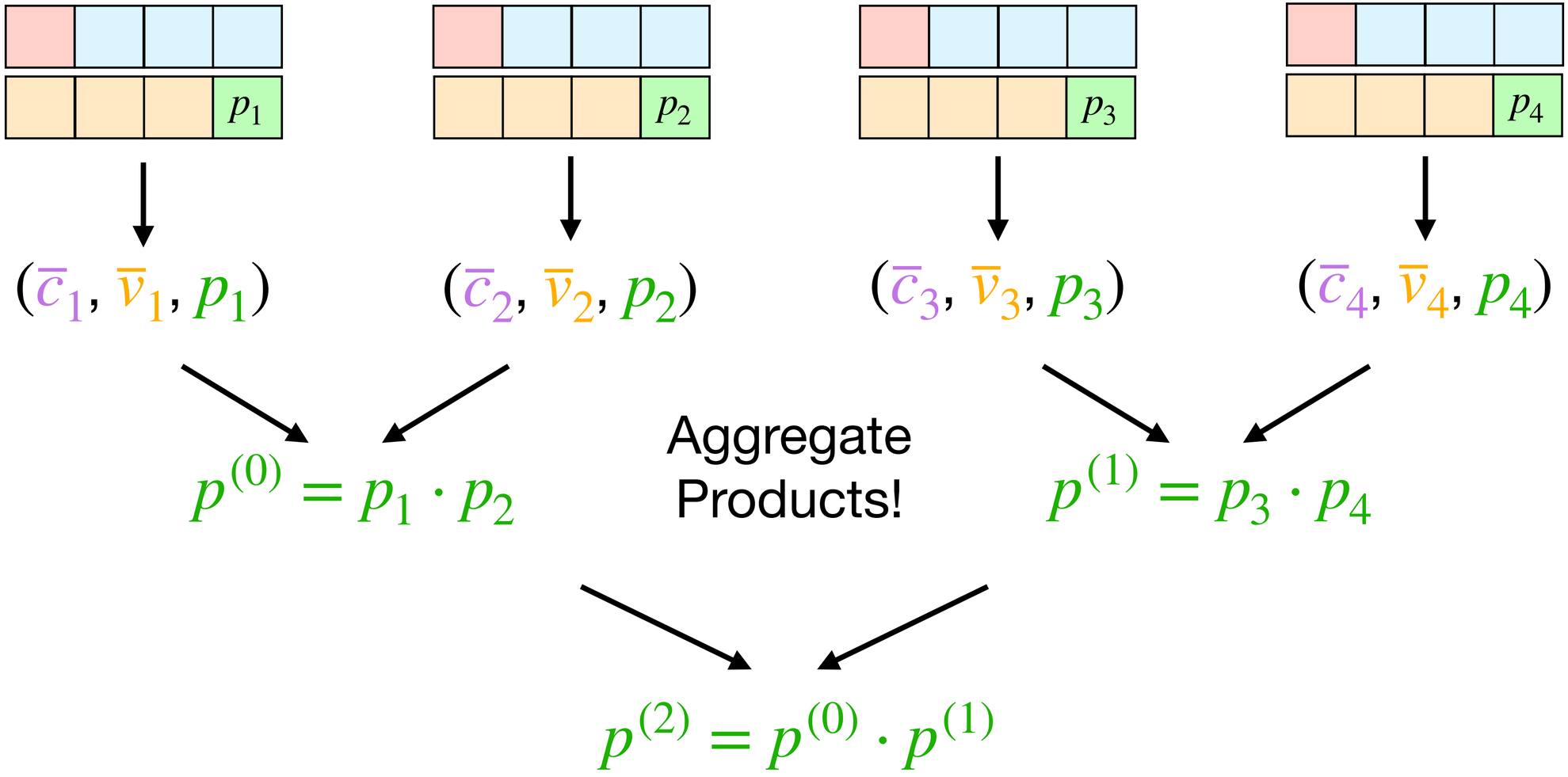
Global Constr $\prod p_i = 1$ and $h_v = \text{Commit}(v)$ **Local Constr** Gate Check + Local Product



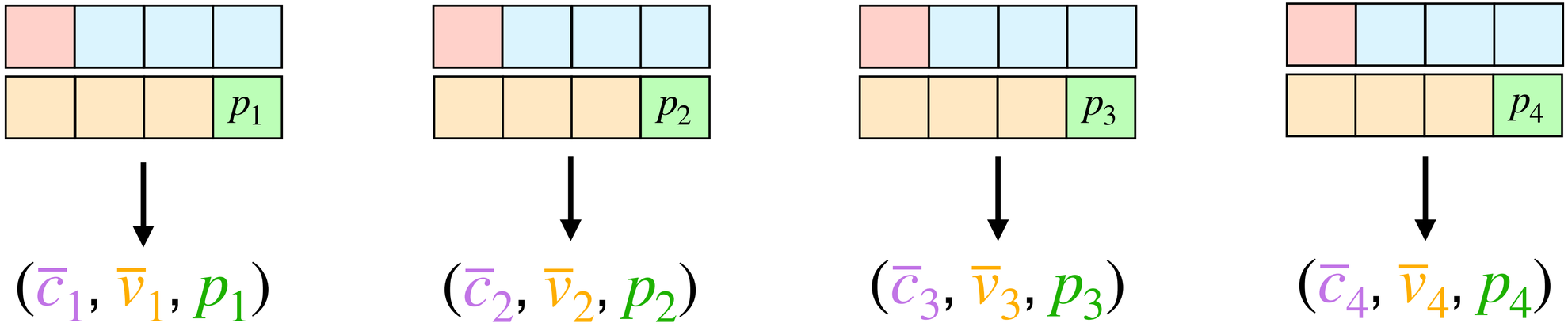
Global Constr $\prod p_i = 1$ and $h_v = \text{Commit}(v)$ **Local Constr** Gate Check + Local Product



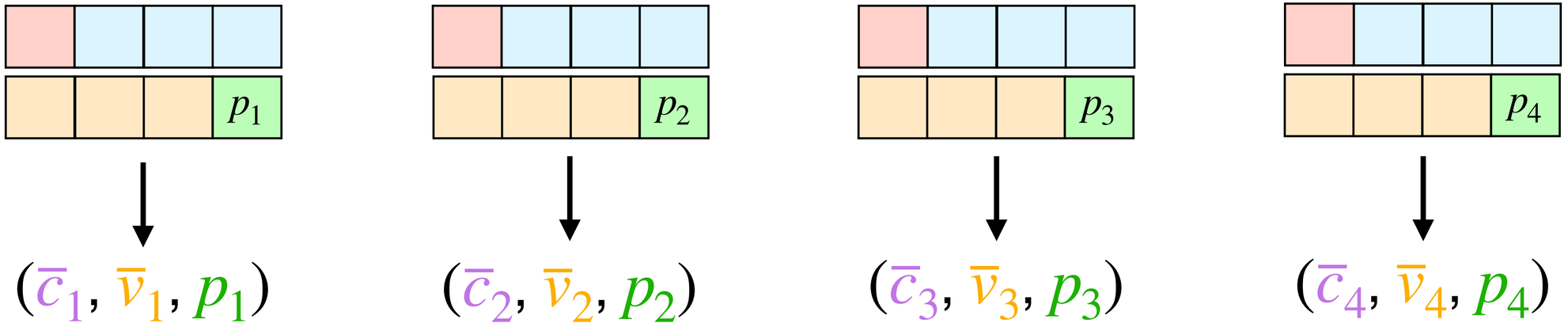
Global Constr $\prod p_i = 1$ and $h_v = \text{Commit}(v)$ **Local Constr** Gate Check + Local Product



Global Constr $\prod p_i = 1$ and $h_v = \text{Commit}(v)$ **Local Constr** Gate Check + Local Product

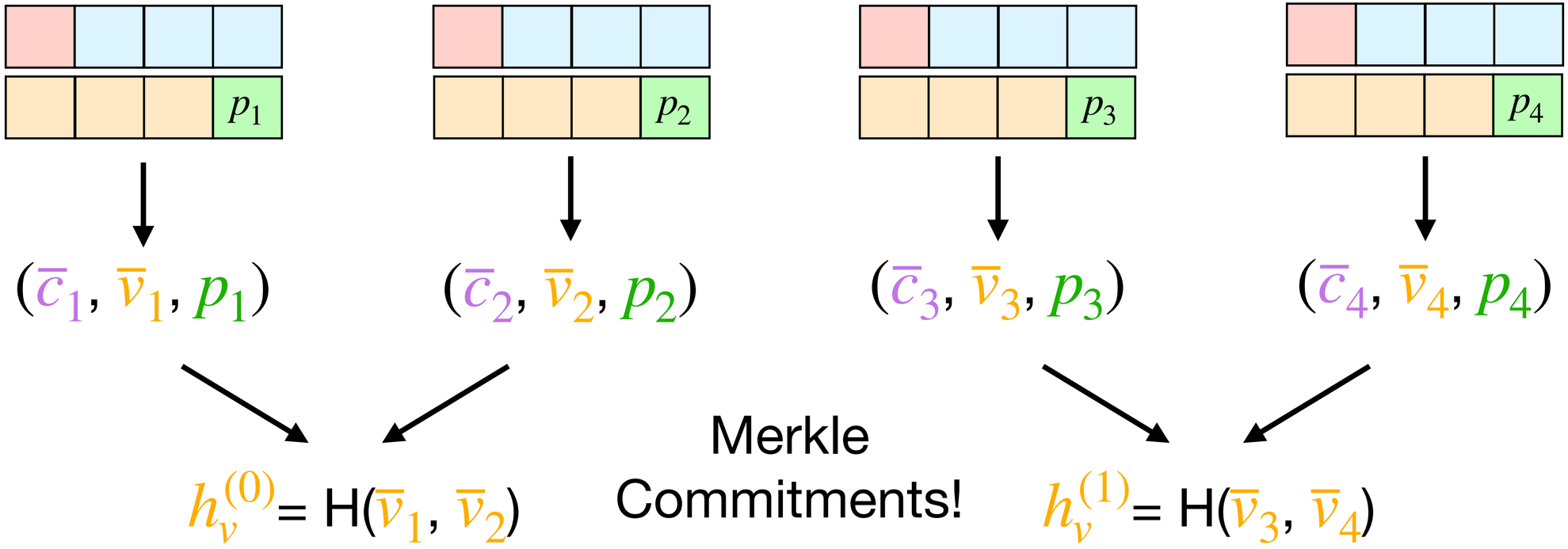


Global Constr $\prod p_i = 1$ and $h_v = \text{Commit}(v)$ **Local Constr** Gate Check + Local Product

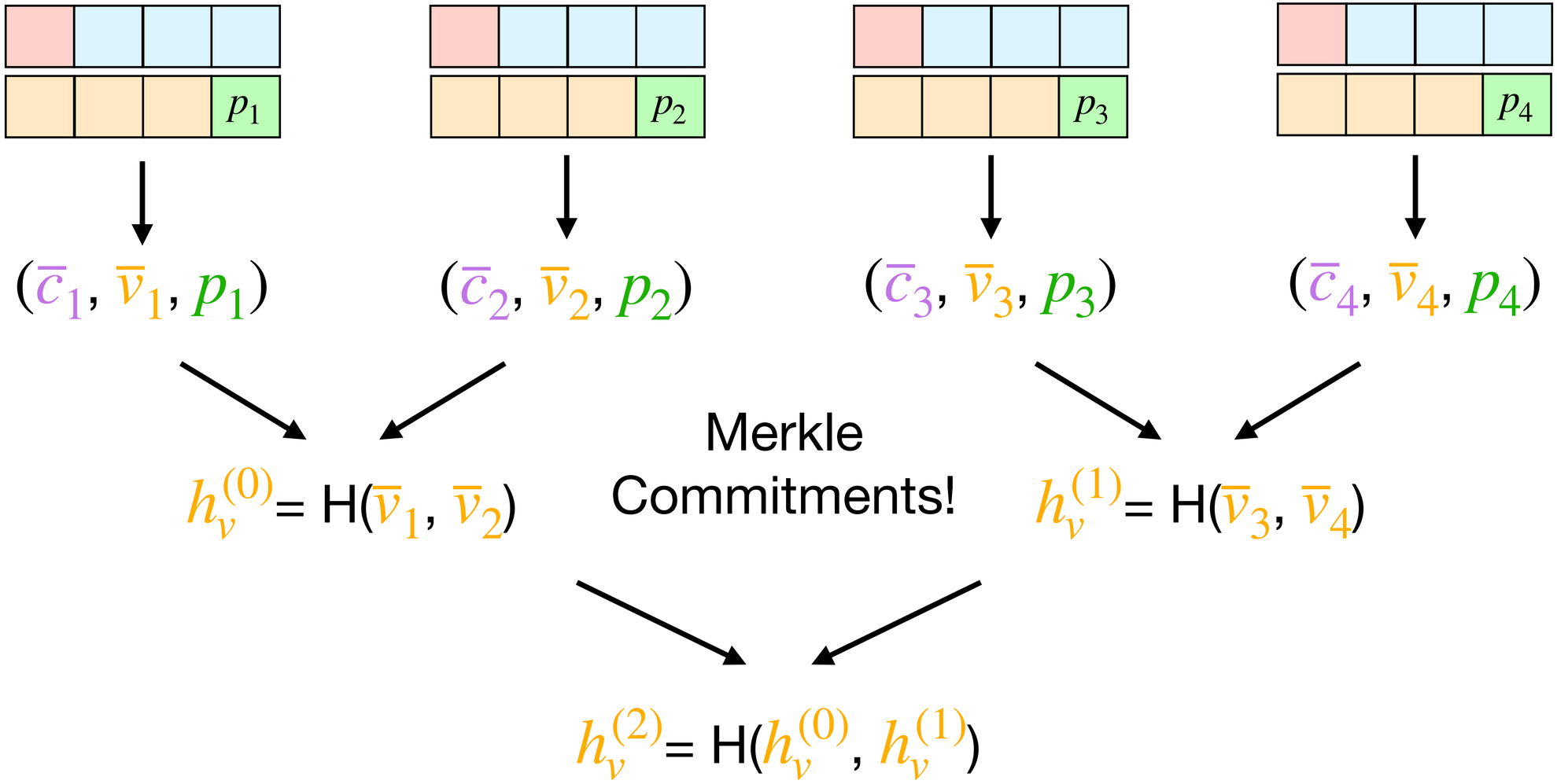


Merkle Commitments!

Global Constr $\prod p_i = 1$ and $h_v = \text{Commit}(v)$ **Local Constr** Gate Check + Local Product

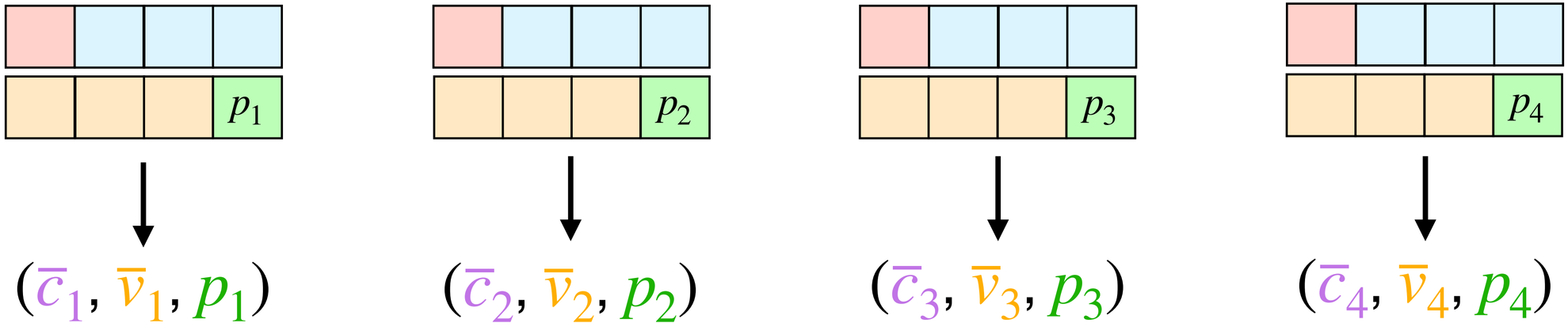


Global Constr $\prod p_i = 1$ and $h_v = \text{Commit}(v)$ **Local Constr** Gate Check + Local Product



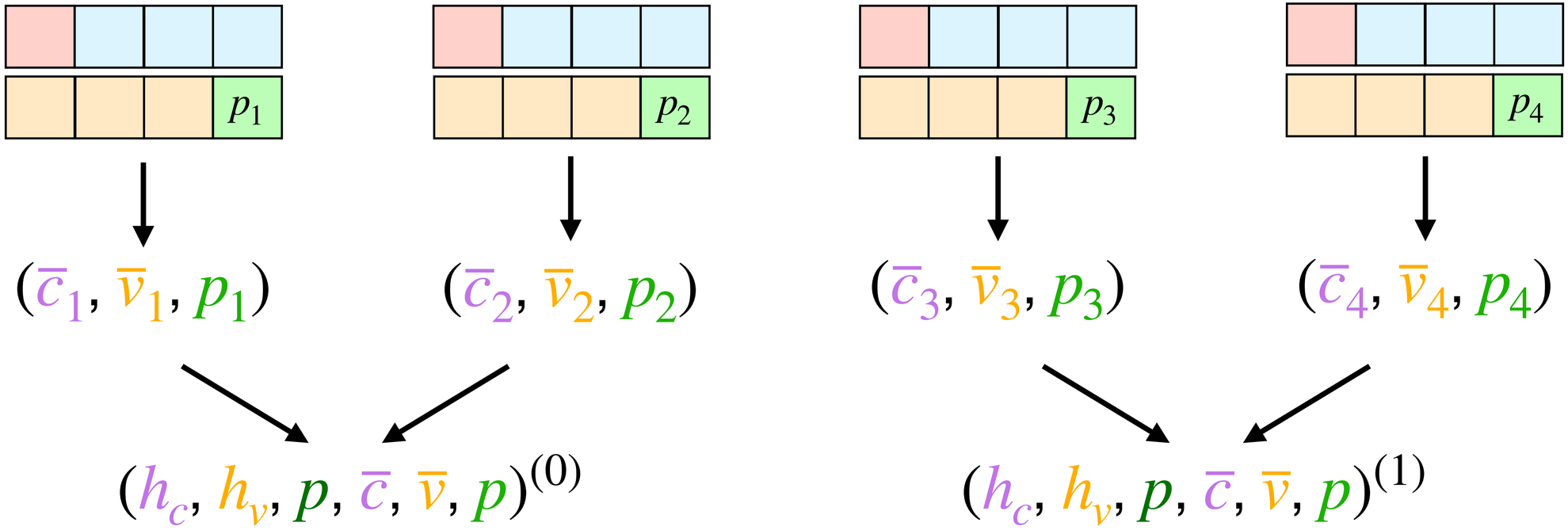
Global Constr $\prod p_i = 1$ and $h_v = \text{Commit}(v)$

Local Constr Gate Check + Local Product



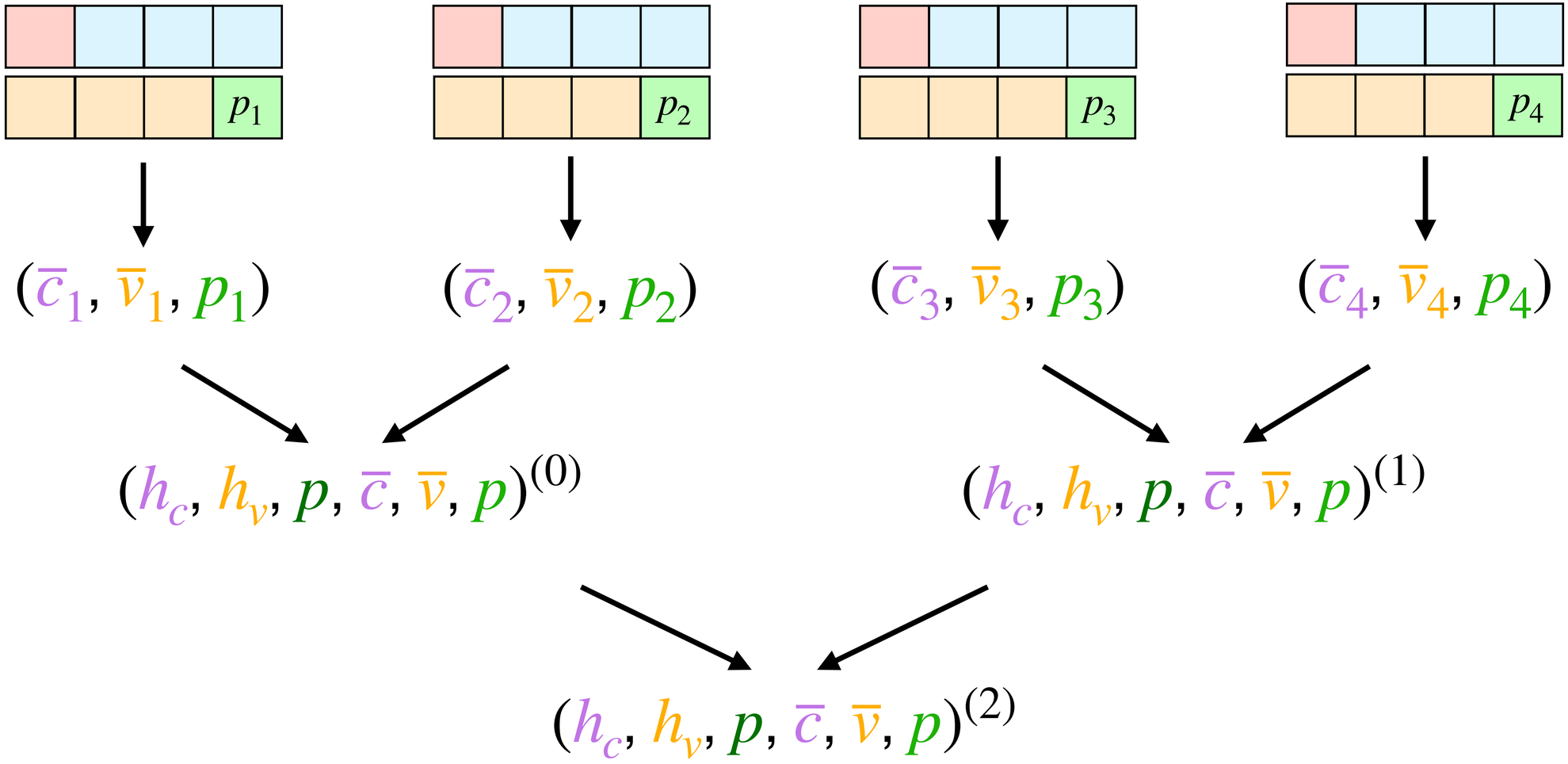
Global Constr $\prod p_i = 1$ and $h_v = \text{Commit}(v)$

Local Constr Gate Check + Local Product



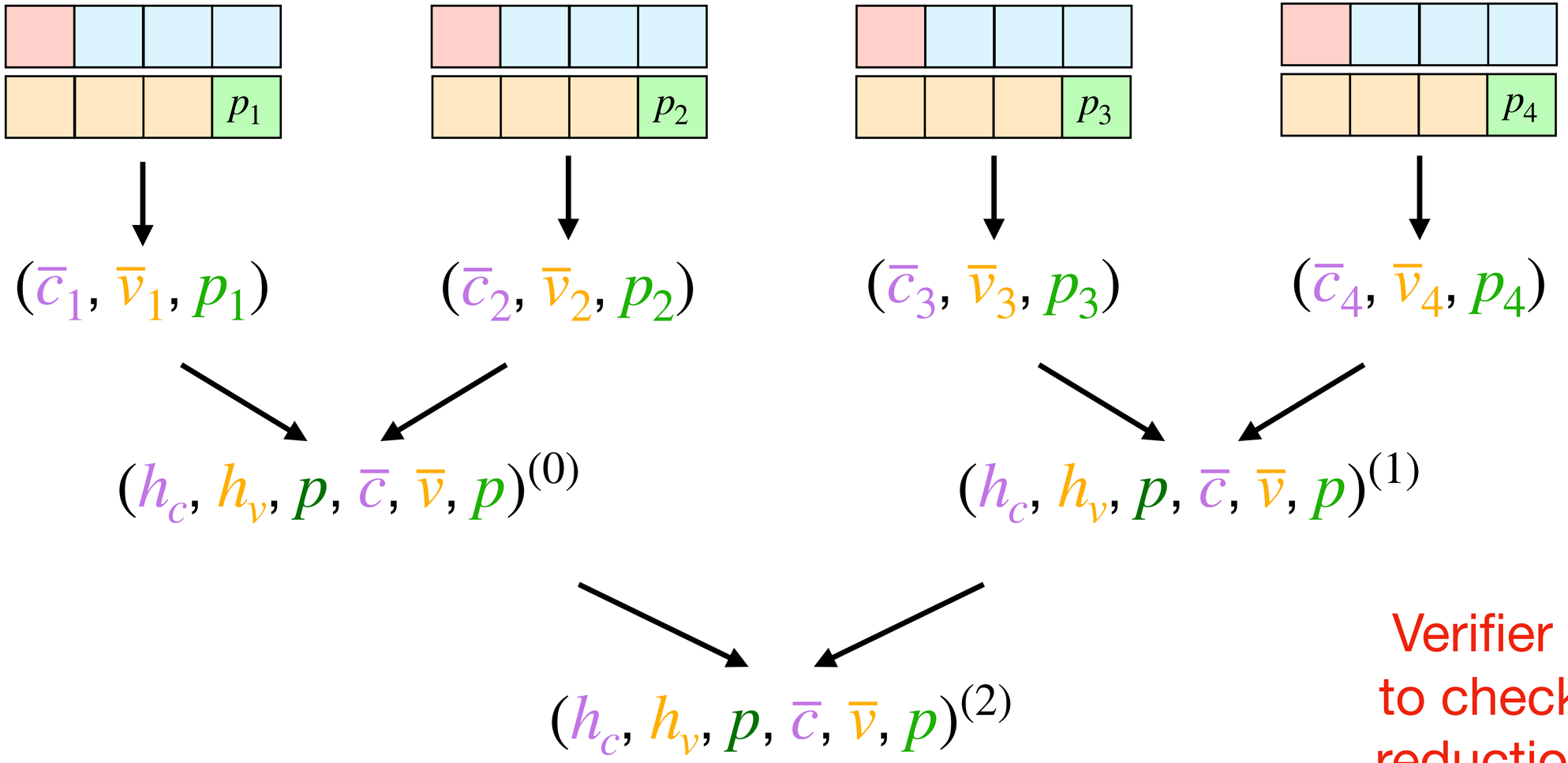
Global Constr $\prod p_i = 1$ and $h_v = \text{Commit}(v)$

Local Constr Gate Check + Local Product



Global Constr $\prod p_i = 1$ and $h_v = \text{Commit}(v)$

Local Constr Gate Check + Local Product



Verifier needs to check every reduction step!

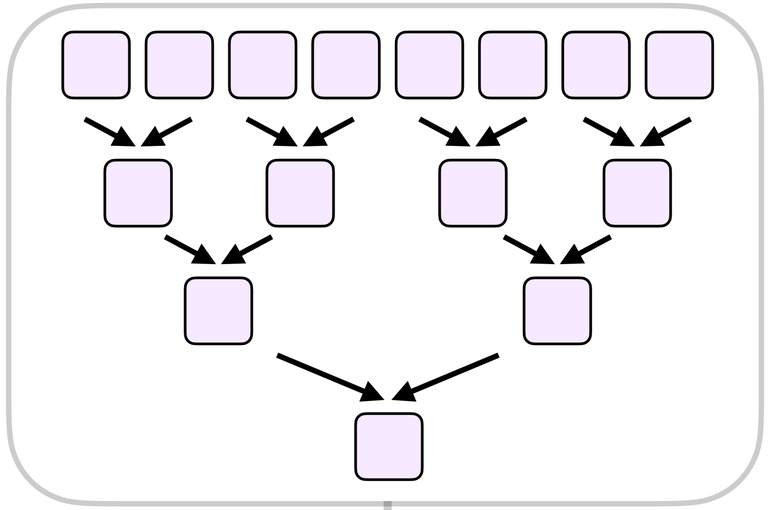
Techniques

$C(x, w) = 0$

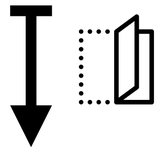
Arbitrary Circuit SAT

Uniform Compiler

Folding Scheme for Polynomial Relations



Small & Identical chunks (Polynomial Relation)



Single chunk

Proof Carrying Data [10]

Succinct Proof

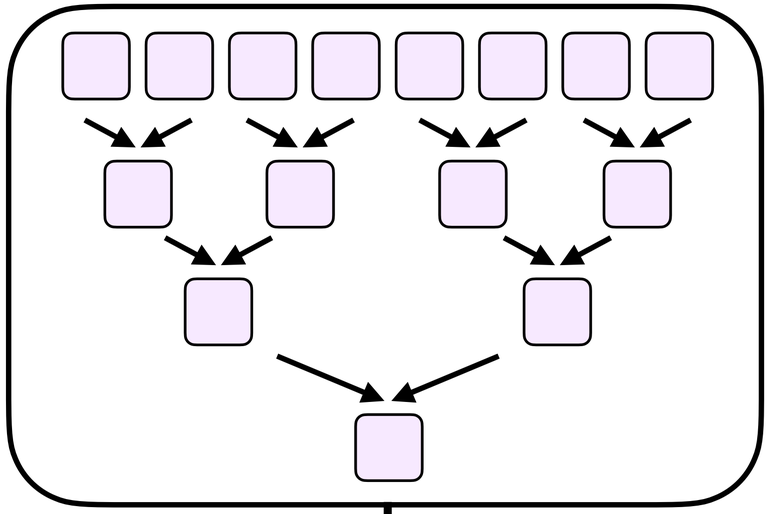
Techniques

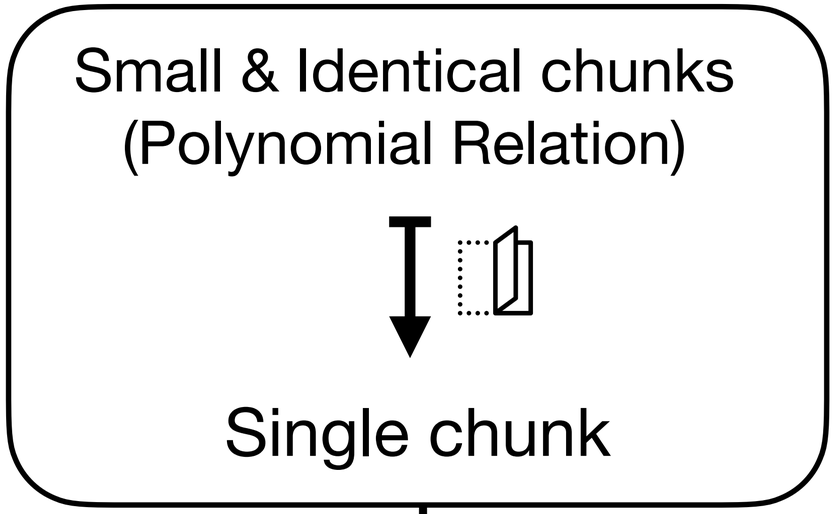
$C(x, w) = 0$

Arbitrary Circuit SAT

Uniform Compiler

Folding Scheme for Polynomial Relations

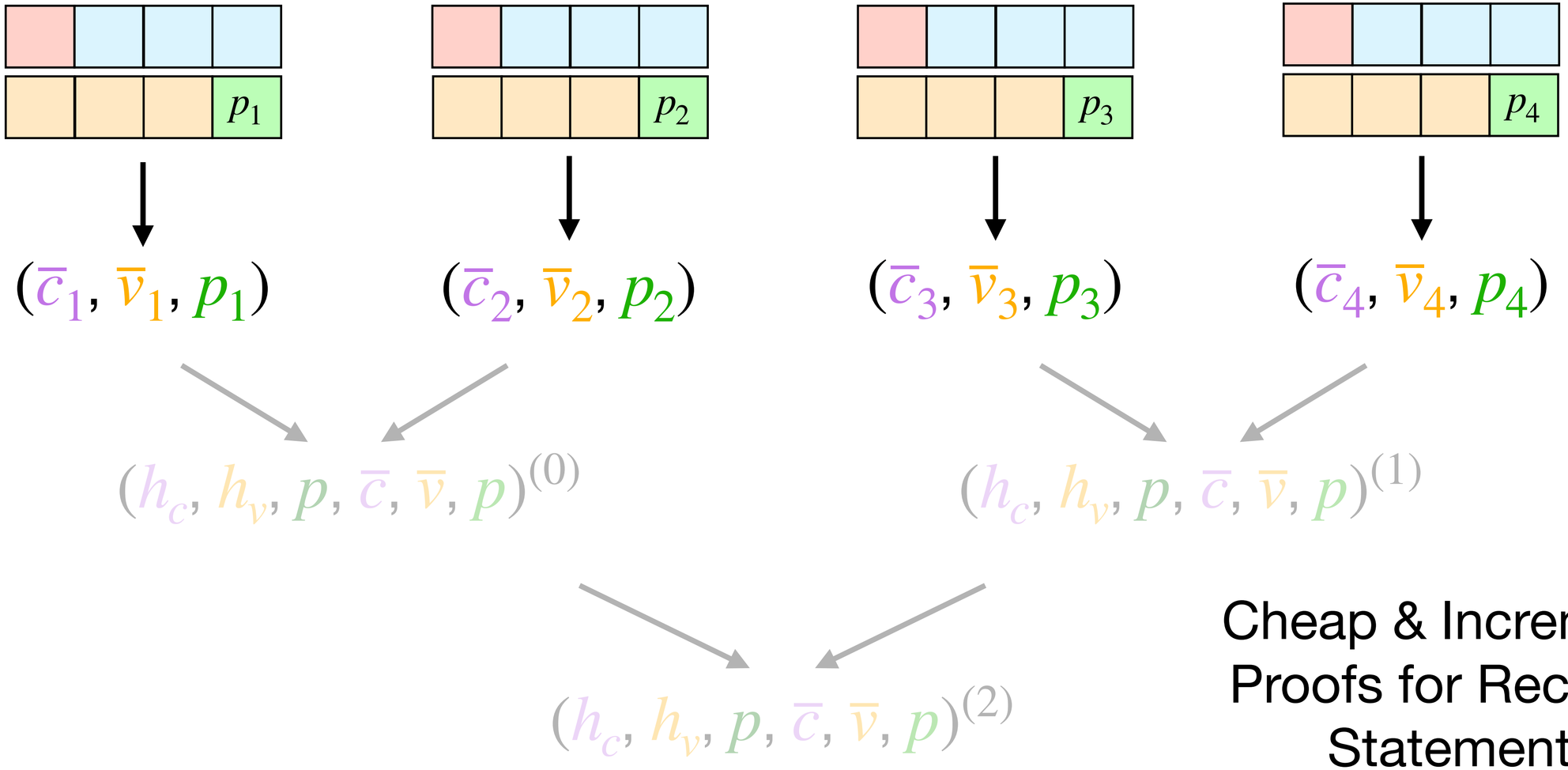




Proof Carrying Data [10]

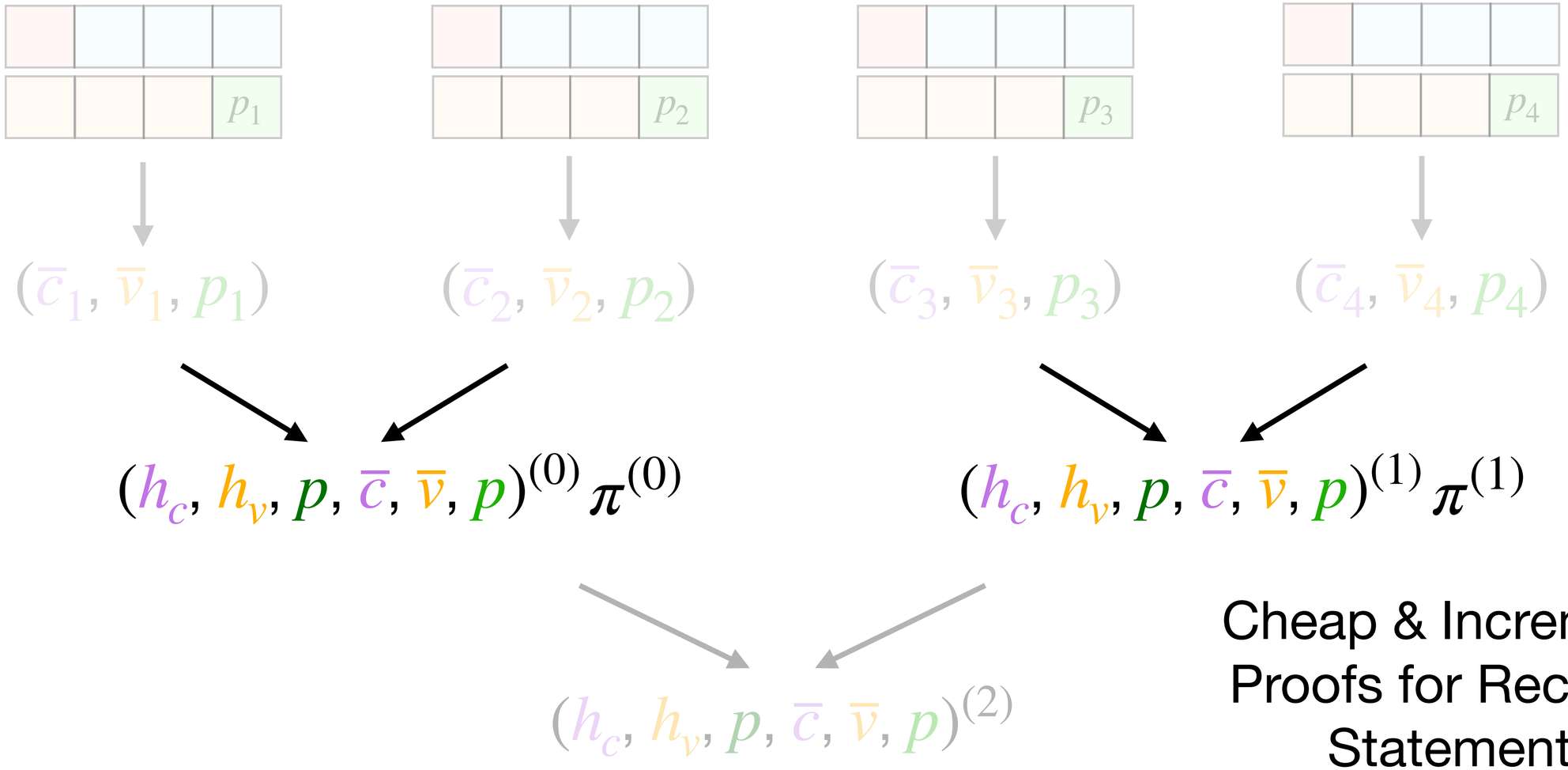
Succinct Proof

Recovering Succinctness - PCD [10]



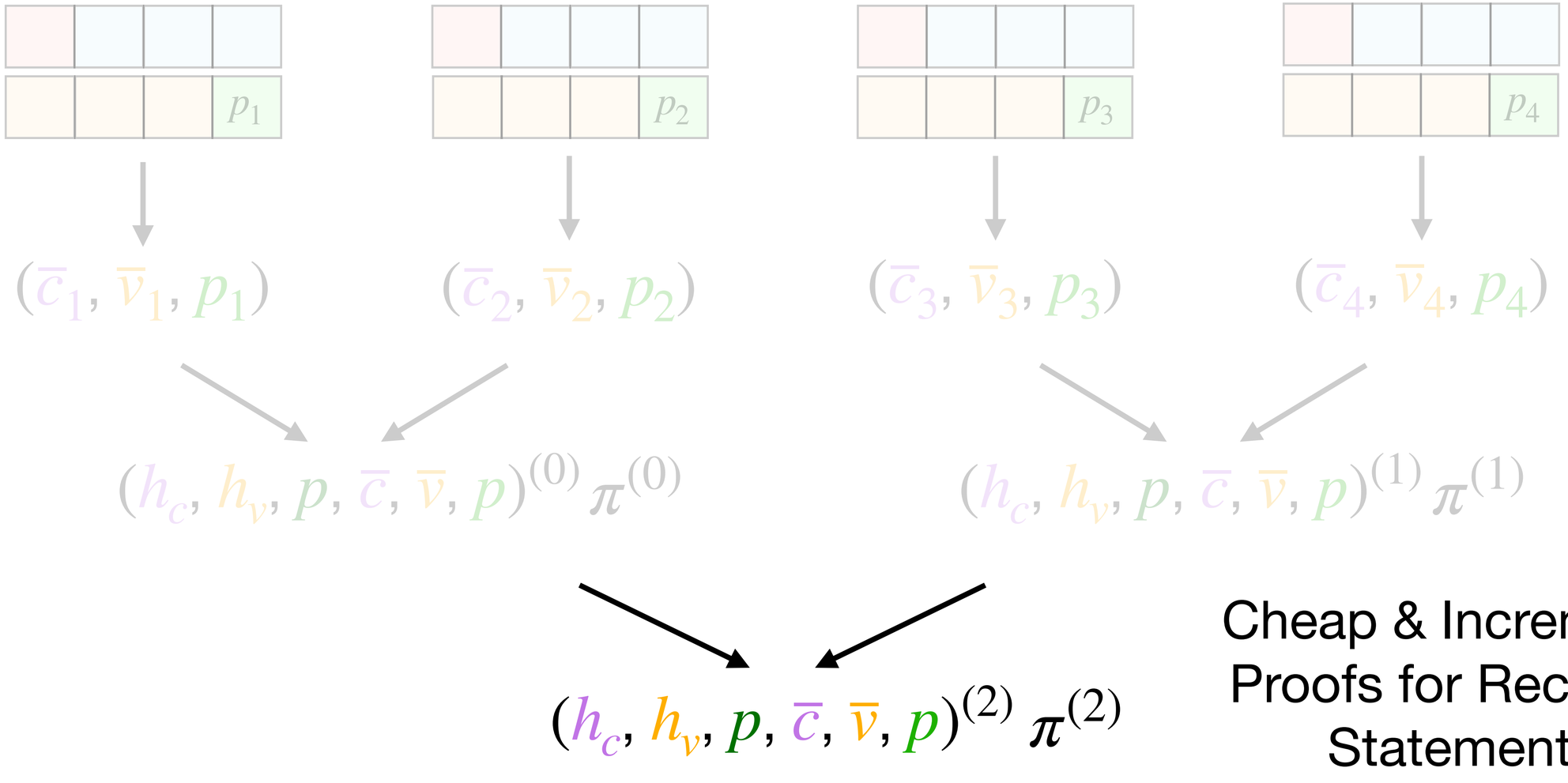
Cheap & Incremental Proofs for Recursive Statements!

Recovering Succinctness - PCD [10]



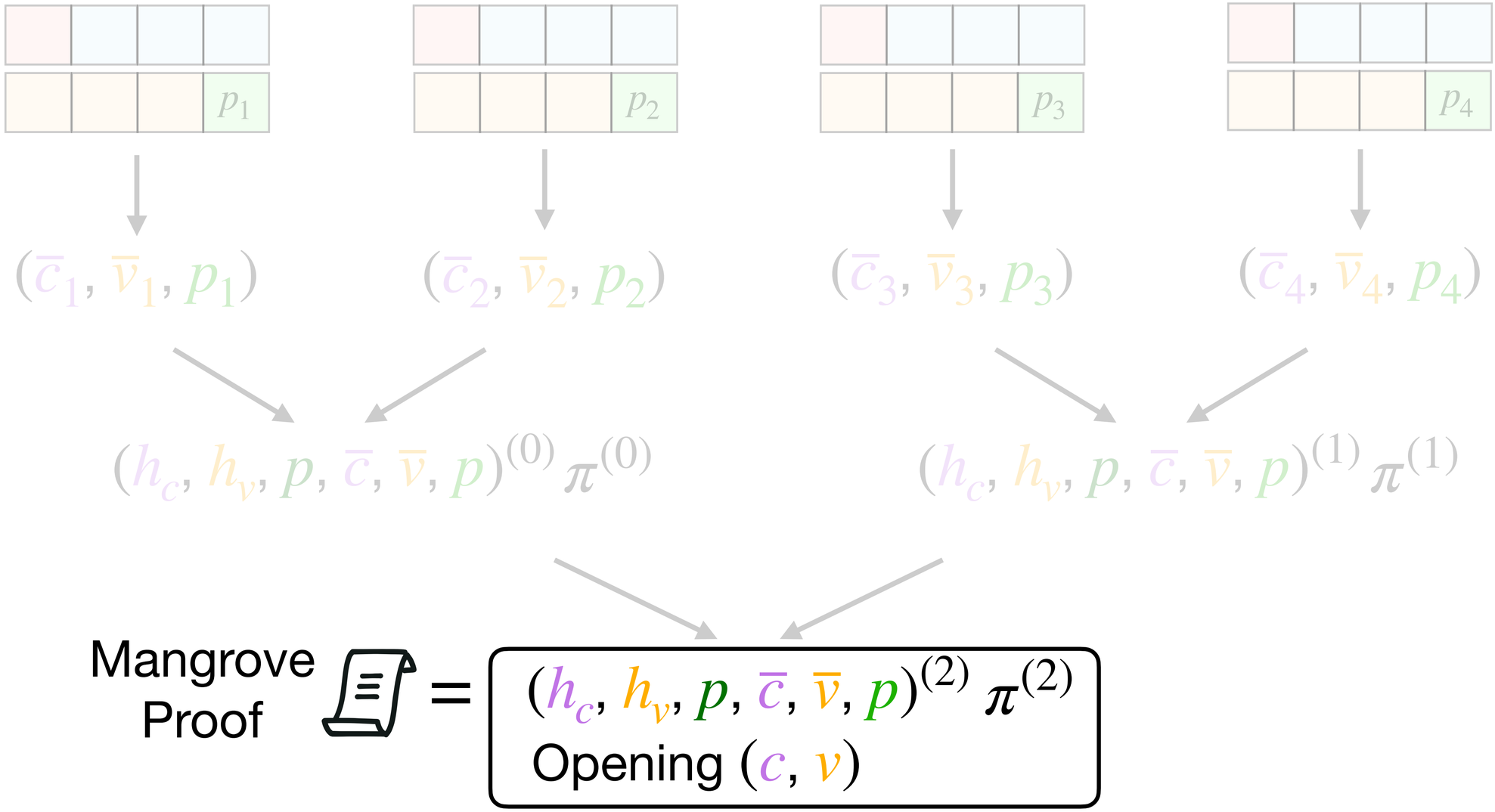
Cheap & Incremental Proofs for Recursive Statements!

Recovering Succinctness - PCD [10]

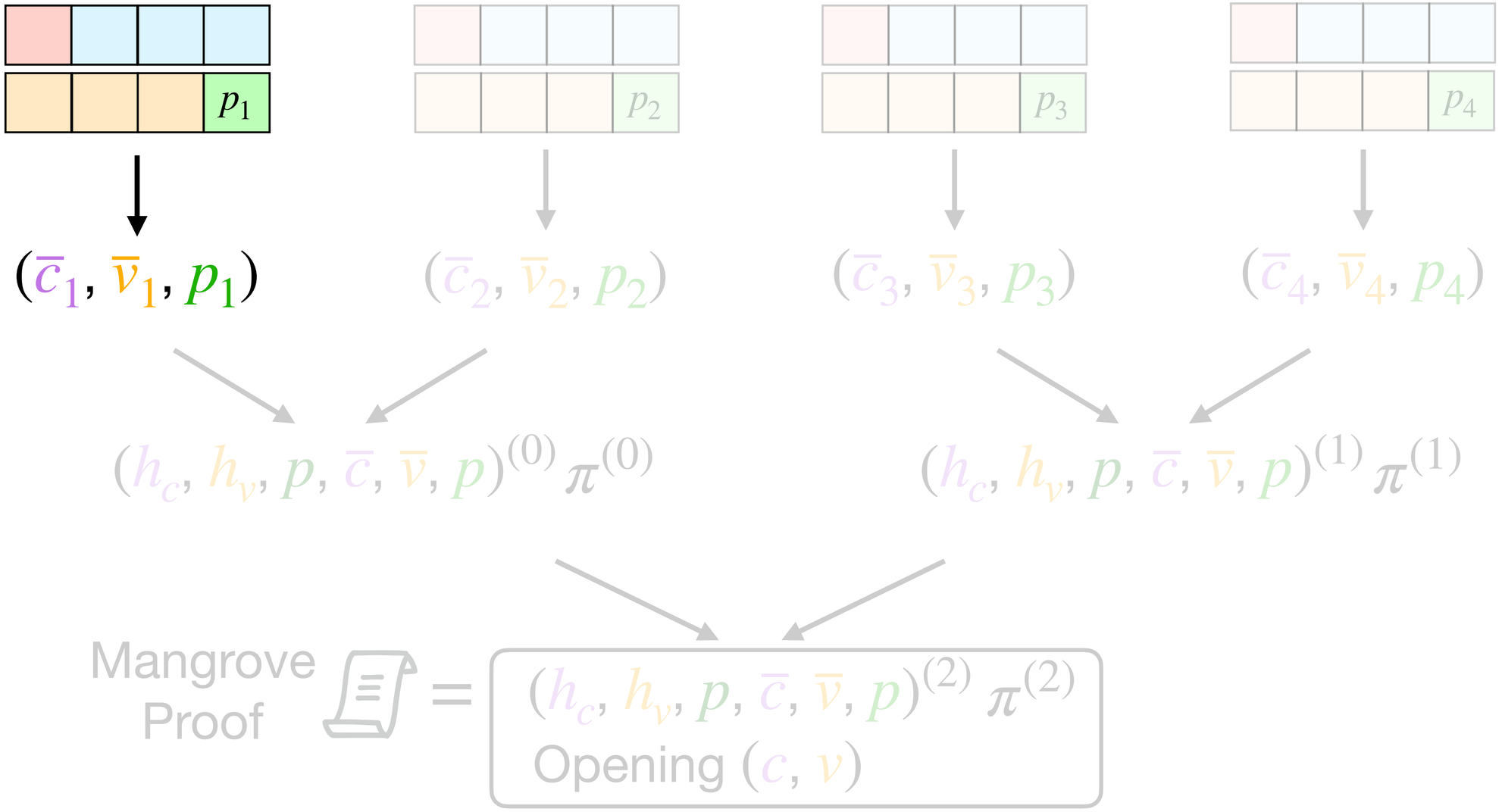


Cheap & Incremental Proofs for Recursive Statements!

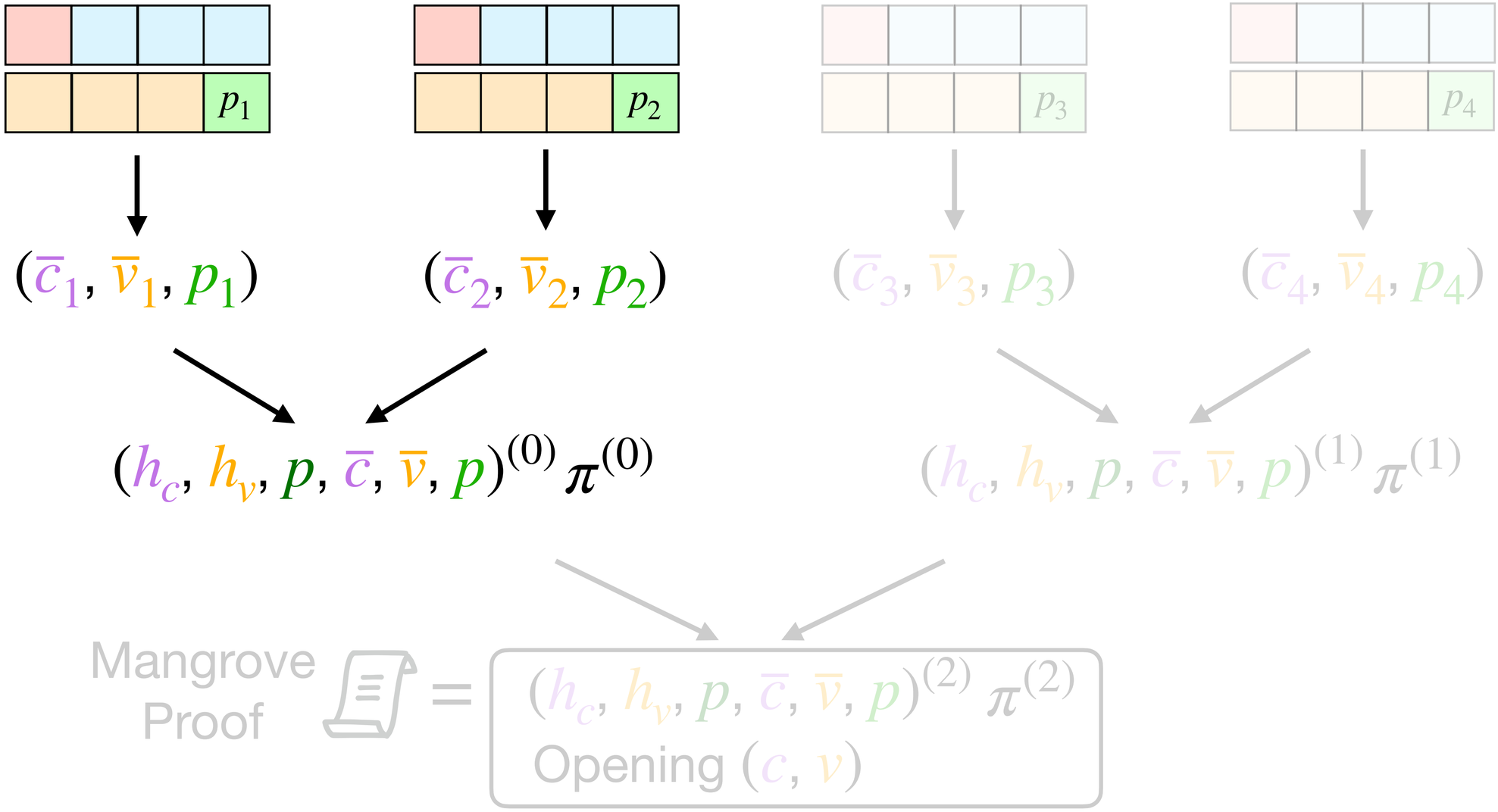
Recovering Succinctness - PCD [10]



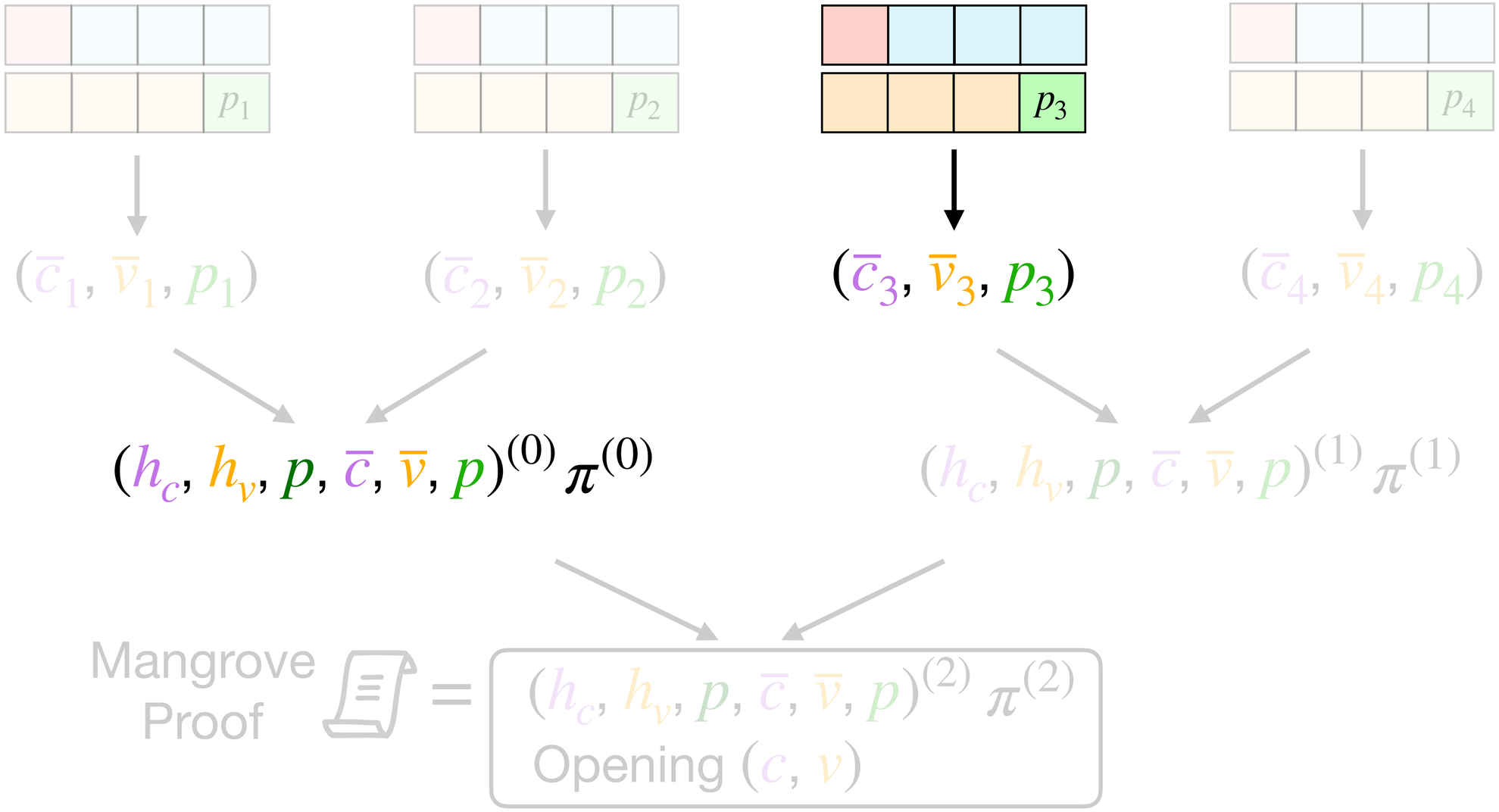
Streaming - Tree Evaluation



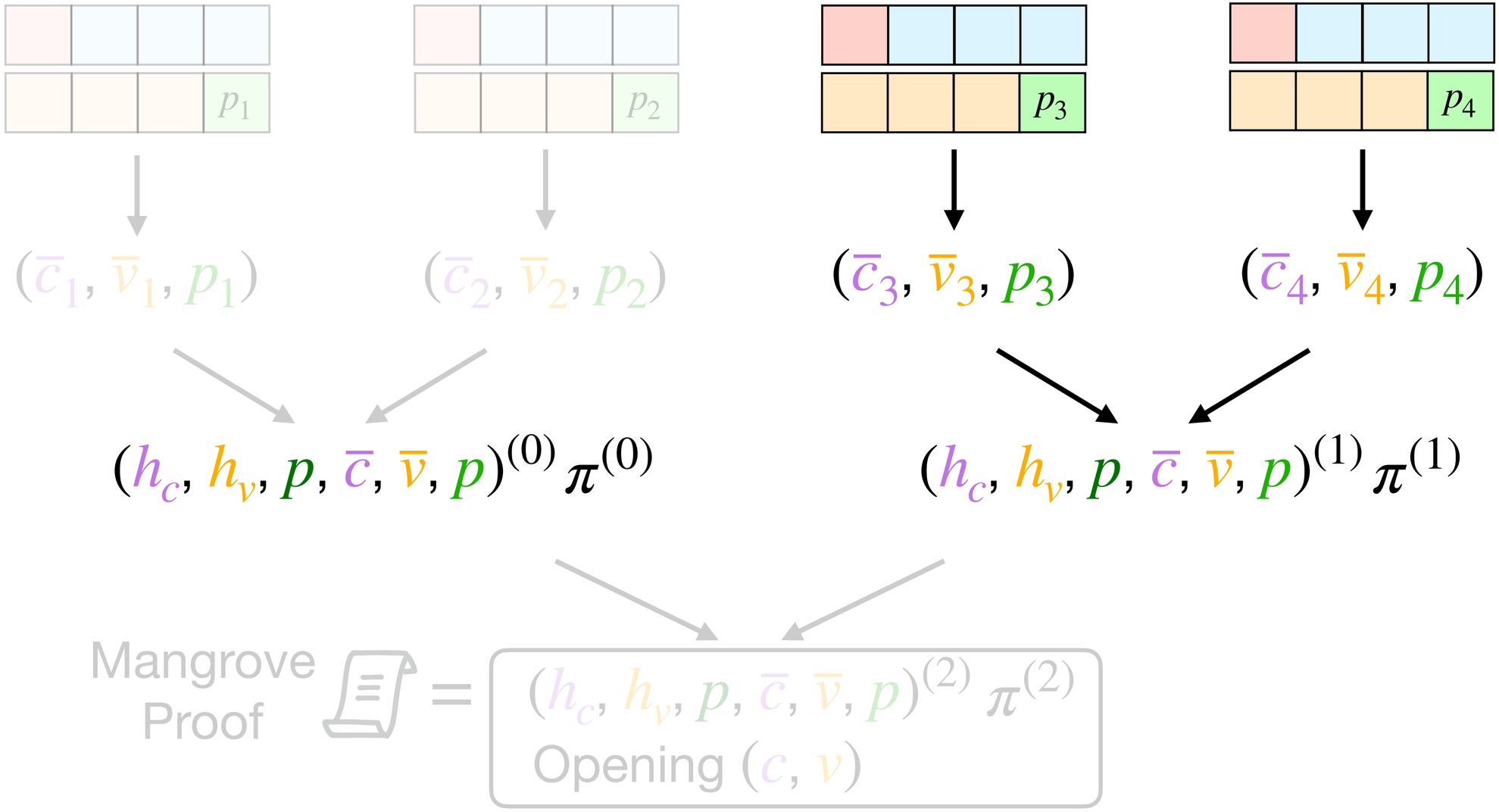
Streaming - Tree Evaluation



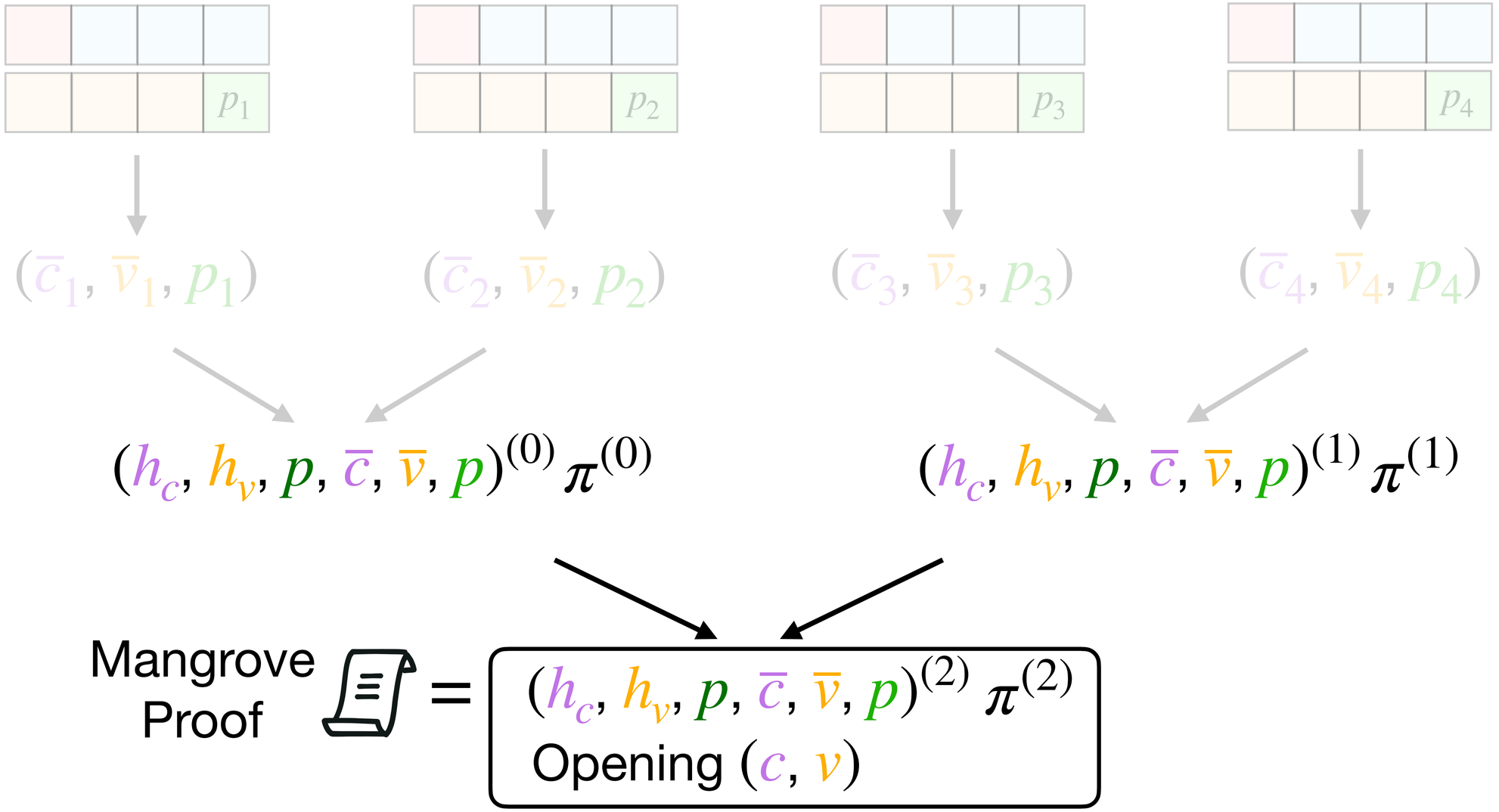
Streaming - Tree Evaluation



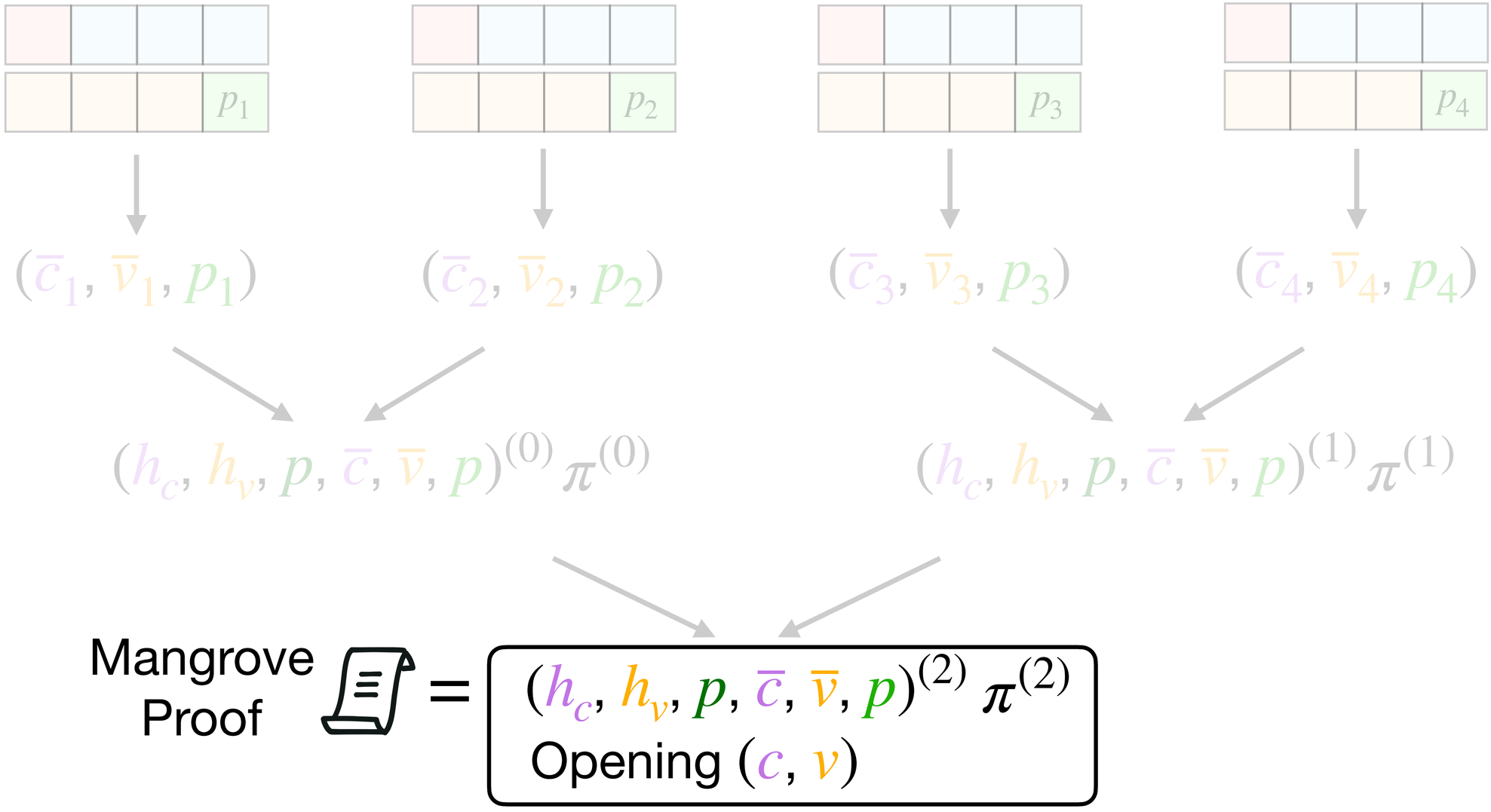
Streaming - Tree Evaluation



Streaming - Tree Evaluation

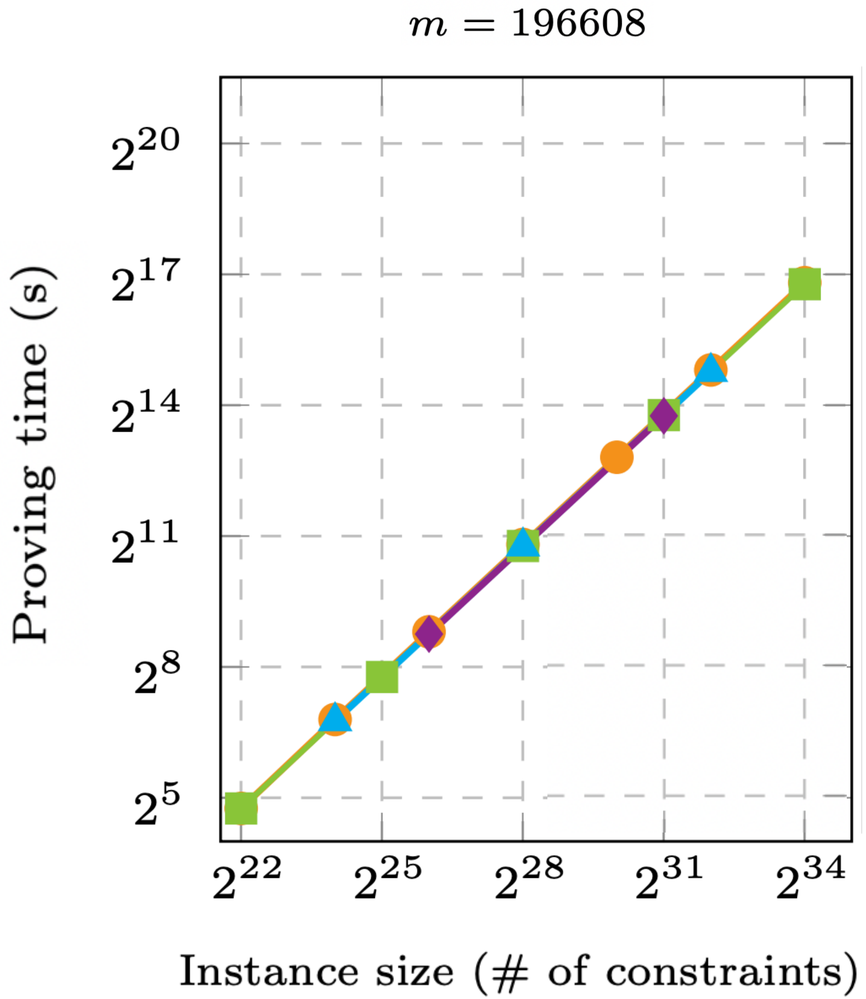


Streaming - Tree Evaluation



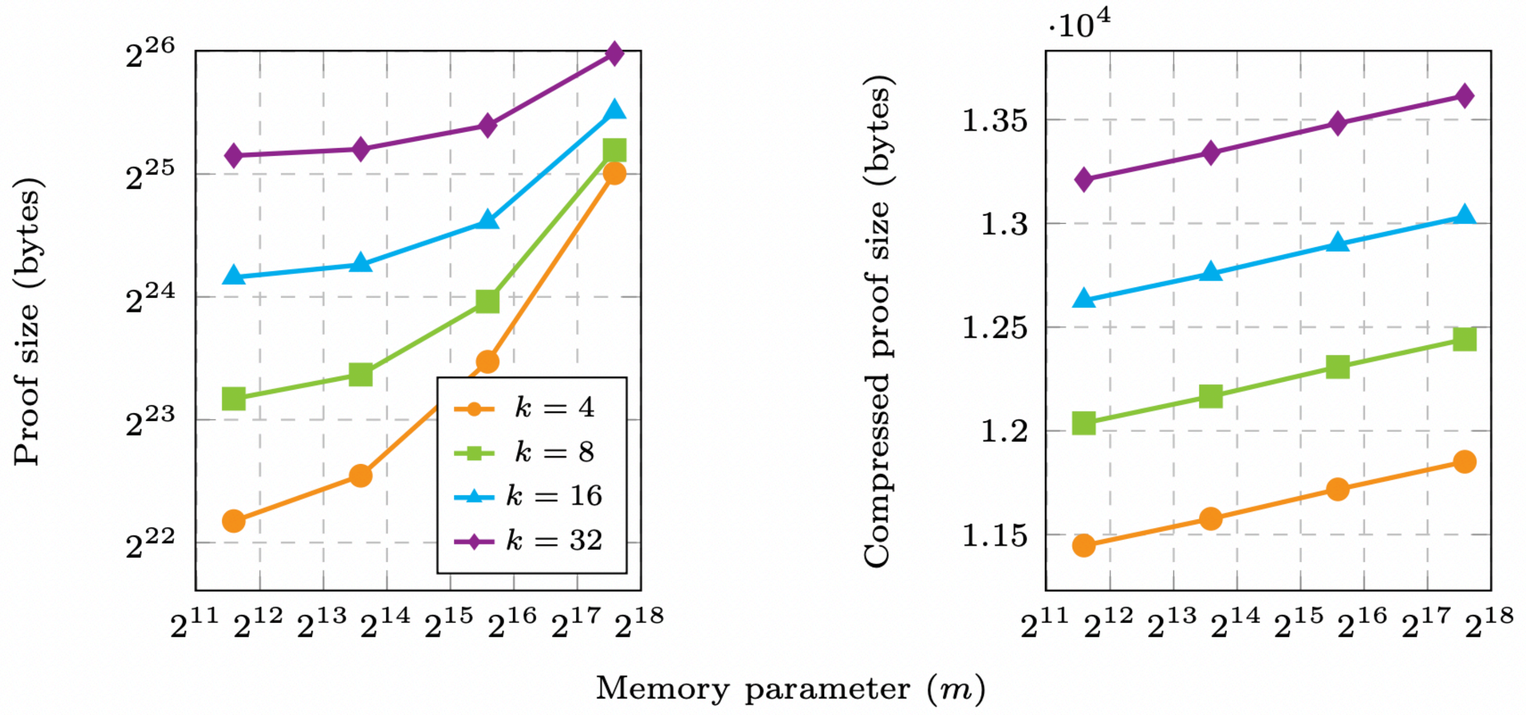
Performance!

- Estimated performance from benchmarks
- **Mangrove Prover** (Macbook Pro):
  - For 2^{24} constraints, 2 mins (400 MB)
  - For 2^{32} constraints, 8 hrs (800 MB)
- **Gemini** (EC2 c5.9xlarge):
  - For 2^{24} constraints, 19 mins (1.2 GB)
  - For 2^{32} constraints, 80 hrs (1.35 GB)



Performance - Proof Sizes

- **Mangrove** 34 MB (independent of instance size)
- **Gemini** 13 - 27 KB for 2^{12} - 2^{35} instance size
- **Compressed Proof** (Spartan [15])**:** < 1 min and < 14 KB



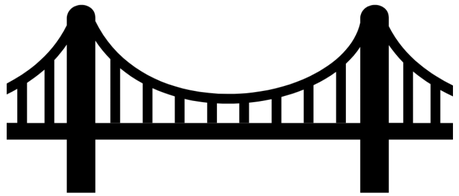
Takeaways!

ia.cr/2024/416

Takeaways!

Uniform Compiler

Arbitrary Circuits



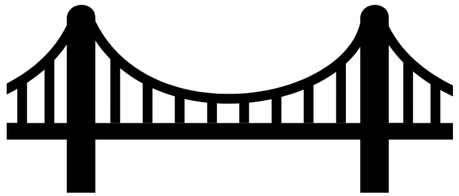
Repeated Computation

ia.cr/2024/416

Takeaways!

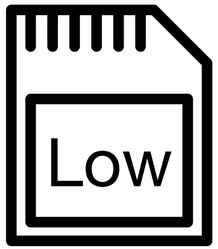
Uniform Compiler

Arbitrary Circuits



Repeated Computation

Proof Carrying Data



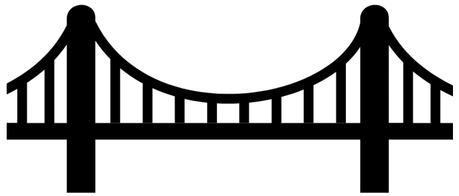
Repeated Computation

ia.cr/2024/416

Takeaways!

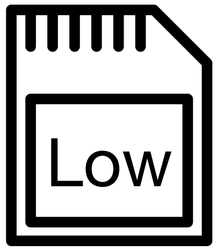
Uniform Compiler

Arbitrary Circuits



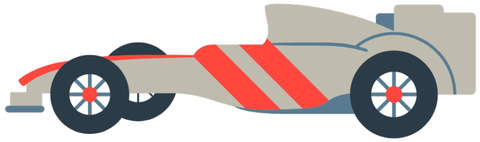
Repeated Computation

Proof Carrying Data



Repeated Computation

Conceptually Simple



~~SRS~~

ia.cr/2024/416

References

1. Filecoin: https://research.protocol.ai/sites/snarks/, https://filecoin.io/blog/posts/trusted-setup-complete/
2. Gemini: https://eprint.iacr.org/2022/420.pdf
3. Hekaton: https://eprint.iacr.org/2024/1208.pdf
4. DIZK: https://eprint.iacr.org/2018/691.pdf
5. Analyzing and Benchmarking ZK-Rollups: https://eprint.iacr.org/2024/889.pdf
6. a16z: https://a16zcrypto.com/posts/article/on-chain-trusted-setup-ceremony/
7. Zcash Setup Vulnerability: https://electriccoin.co/blog/zcash-counterfeiting-vulnerability-successfully-remediated/
8. Scroll: https://zk-learning.org/assets/lecture12.pdf
9. Algebraic Reductions of Knowledge: https://eprint.iacr.org/2022/009.pdf
10. Proof Carrying Data without Succinct Arguments: https://eprint.iacr.org/2020/1618.pdf
11. C. Andrew Neff A verifiable secret shuffle and its application to e-voting.
12. Bayer and Groth Efficient zero-knowledge argument for correctness of shuffle
13. Franzese, Katz, Lu, Ostrovsky, Wang, Weng: Constant-overhead zero-knowledge for RAM programs
14. Plonk: https://eprint.iacr.org/2019/953
15. Spartan: https://eprint.iacr.org/2019/550
16. Groth16: https://eprint.iacr.org/2016/260.pdf
17. Marlin: https://eprint.iacr.org/2019/1047.pdf
18. Images: https://www.svgrepo.com/