Complexity-Preserving Sublinear-Arguments from Symmetric Key Primitives

Laasya Bangalore
Muthu Venkitasubramaniam
Georgetown University

LIGERO

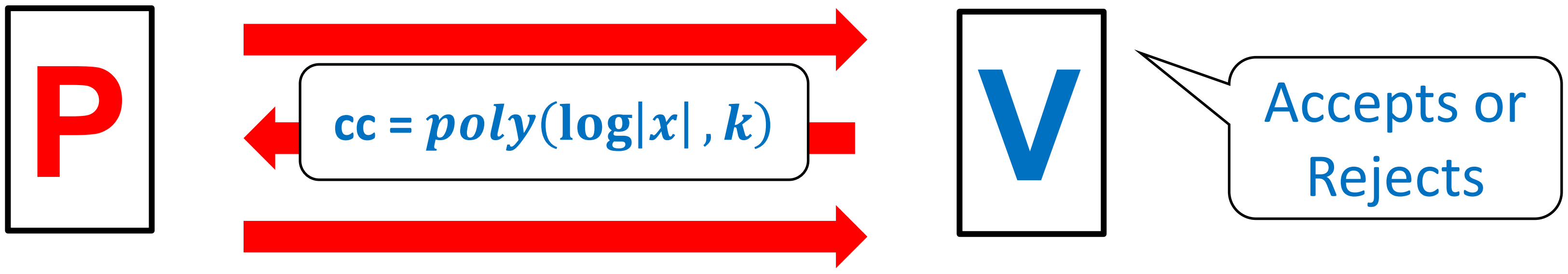
Rishabh Bhadauria
Carmit Hazay
Bar-Ilan University

GEORGETOWN UNIVERSITY

אוניברסיטת בר־אילן
Bar-Ilan University

Succinct (ZK) Arguments [Kil99,Mic00]

Given a language L



Completeness: $\forall x \in L,\quad \langle P, V\rangle(x) = 1$ w.p. 1

Soundness: $\forall x \notin L, \forall$ **PPT** $P^*,\ \langle P^*, V\rangle(x) = 1$ w.p. $\leq\ negl(k)$

Zero-knowledge: $\forall x \in L, \forall$PPT $V^*,\quad \exists$ PPT S, s.t.

$$\{view_{V^*}\langle P, V^*\rangle(x)\} \approx \{S(x)\}$$

Main Question

Do there exist complexity-preserving public-coin succinct* arguments for NP from minimal assumptions?

Main Question

Do there exist complexity-preserving public-coin succinct* arguments for NP from minimal assumptions?

This Talk **YES***

* sublinear

Main Question

Do there exist complexity-preserving public-coin succinct* arguments for NP from minimal assumptions?

This Talk **YES***

* sublinear

$\left.\begin{array}{l}\text{Time-preserving} \\ \text{Space-preserving}\end{array}\right\}$ Complexity-preserving

Main Question

Do there exist complexity-preserving public-coin succinct* arguments for NP from minimal assumptions?

This Talk **YES***

* sublinear

Taxonomy of Arguments

- Succinct* vs Non-succinct
- Interactive vs Non-interactive
- Trusted setup vs No setup (transparent)
- ZK vs (only) Integrity
- Public-Key Crypto vs (only) Symmetric-Key Crypto

Time-preserving
Space-preserving
$\Big\}$ Complexity-preserving

A SNARKy background

1992: Sublinear ZK for NP [Kil92]
2000: CSProofs [Mic00]
2007: IKO [IKO07], IVC [V07]
2008: IVC [Val08]
2010: Short-PB-NIZKA [Groth10], Preprocessing-Verifiable-Computation [GGP10]
2012: QSP [GGPR12], EfficientPCP [IMS12], Succinct-NIArgs-LIP [BCIOP12]
2013: Pinocchio [PGHR13], SNARKs-for-C [BCGTV13], ZK-vonNeumann [BCTV13]
2014: Geppetto [CFHKKNPZ14], CyclesOfCurves [BCTV14]
2015: IP4Muggles [GKR15], SNARKs-for-MapReduce [CTV15]
2016: ZKBoo [GMO16], BulletproofsPrequel [BCCGP16], Groth16 [Groth16], HybIntZK [CGM16]
2017: Ligero (AHIV17), ZKB++ and Picnic [CDGORRSZ17] (discuss alongside ZKBoo), Hyrax [WTsTW], zk-vSQL [ZGKPP17] (can add to existing vSQL section), Bulletproofs [BBBPWM17], SnarkySigs [GM17]
2018: Aurora [BCRSVW18], FRI [BBHR18], ZKStarks [BBHR18], Picnic2 [KKW18], vRAM [ZGKPP18], DIZK [WZCPS18], UpdatableNIZK [GKMM18], HybNIZK [AGM18]
2019: Fractal [COS19], Halo [BGH19], Plonk [GWC19], RedShift [KPV19], Spartan [Setty19], DeepFRI [BGKS19], LatticeZKPs [ESLL19], SubversionResistant [Bag19], Darks [BFS19], LatticeSnarkArithmetic [Nit19], ZKPSetMembership [BCFGD19]
2020: HaloInfinite [BDFG20], Quarks (Xiphos and Kopis) [SL20], Dory [Lee20],
Wolverine [WYKW20], Bulletproofs+ [CHJKS20], SPARKS [EFKP20], Plookup [GW20], SuperSonic [BFS20], CompressedSigma [AC20], LatticeZKviaOTC [LKS20], GeneralizedCompressedSigma [ACR20], PVZKfromBlockchain [SSV20], LinePointZK [DIO20], PublicCoinZKTime&Space [BHRRS20], Dory [Lee20], DoublyEfficientIP [ZLWZSXZ20], PqSnarks4Rsis-Rlwe [BCOS20], ZAPsAlgebraicLangs [CH20]
2021: Manta [CXZ21], Nova [KST21], Rinocchio [GNS21], Limbo [DGOT21], QuickSilver [YSWW21], Limbo [GOT21], IntRange [CKLR21], SubexpDDH [JJ21], Cerberus [LSTW21], ConstOverZKRamProgs [FKLOW21]
2022: NIZK Multiple Verifiers [YW22], Feta [BJOSS22], gOTzilla [BCGHM22], ZK UNSAT [LAHPTW22]
Credits: ZKProof.org

A SNARKy background

(Uniform) RAM program that takes time T and uses space S

- Succinct
- Public-coin or publicly verifiable
- Time-preserving
- Space-preserving
- “Black-box” in the underlying assumption

A SNARKy background

(Uniform) RAM program that takes time T and uses space S

- Succinct
- Public-coin or publicly verifiable
- Time-preserving
- Space-preserving
- “Black-box” in the underlying assumption

[BC12] Designated Verifier Sublinear Arguments
[HR18] Non-interactive Sublinear Arguments

A SNARKy background

(Uniform) RAM program that takes time T and uses space S

- Succinct
- Public-coin or publicly verifiable
- Time-preserving
- Space-preserving
- “Black-box” in the underlying assumption

[WTSTW18,XZZPS19,S20,SL20,KMP20,BCG20a,BCG20,…]
All require Prover to use space proportional to T!

A SNARKy background

(Uniform) RAM program that takes time T and uses space S

- Succinct
- Public-coin or publicly verifiable
- Time-preserving
- Space-preserving
- "Black-box" in the underlying assumption

[Val08,BCCT13] Complexity preserving via recursive composition
[EFKP20] SNARKs for parallel RAM comp. using CRH and SNARKs
[BGH21,BCMS20,COS20] "Heuristic" assumptions

A SNARKy background

(Uniform) RAM program that takes time T and uses space S

- Succinct
- Public-coin or publicly verifiable
- Time-preserving
- Space-preserving
- “Black-box” in the underlying assumption

[BBHR18,BFHVXZ20] *Folklore

A SNARKy background

(Uniform) RAM program that takes time T and uses space S

✓ • Succinct
✓ • Public-coin or publicly verifiable
✓ • Time-preserving
✓ • Space-preserving
✓ • "Black-box" in the underlying assumption

[BHRRS20] Based on hardness of discrete log
[BHRRS21] Based on hardness assumptions on hidden order groups

A SNARKy background

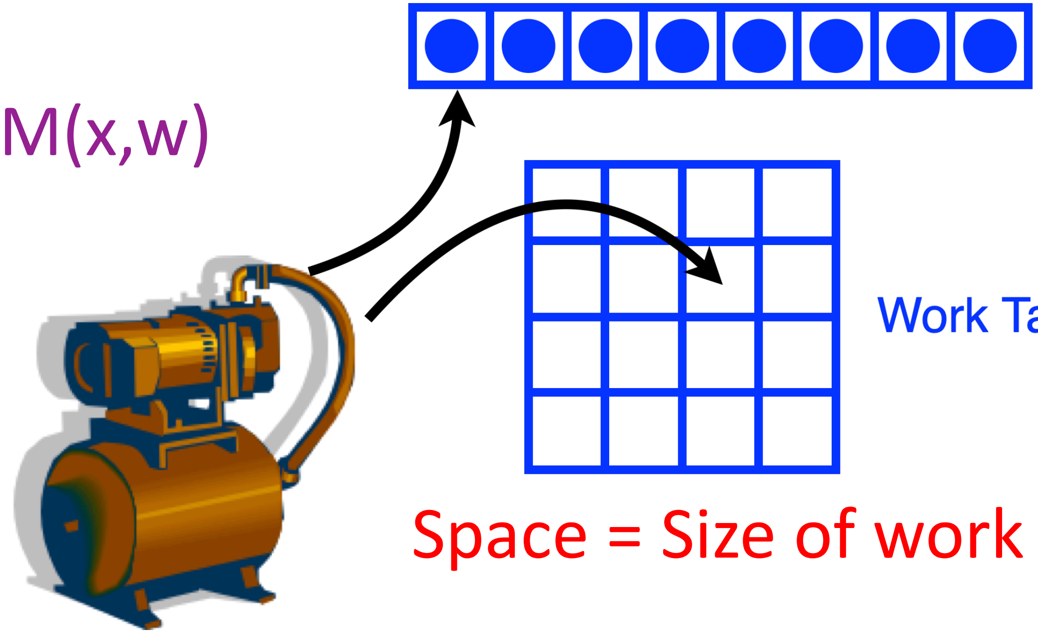
(Uniform) RAM program that takes time T and uses space S

- ✓ Succinct
- ✓ Public-coin or publicly verifiable
- ✓ Time-preserving
- ✓ Space-preserving
- ✓ “Black-box” in the underlying assumption

[BHRRS20] Based on hardness of discrete log
[BHRRS21] Based on hardness assumptions on hidden order groups

Main Question: Can we get based on symmetric cryptography?

RAM Model



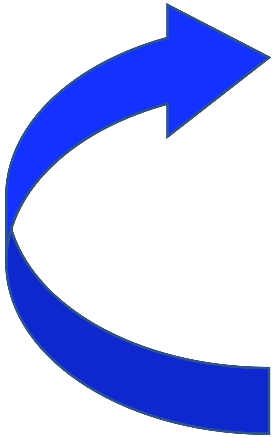
Input Tape - Linear Access

(Uniform) RAM program M(x,w)

- Runs in time T(|x|)
- Uses space S(|x|)
- It is of constant size

Work Tape - RAM Access

Space = Size of work tape



Modelled as a RAM

P has x, w on (two) input tapes with linear access.

V has x on input tape with linear access.

Main Result – Upper Bound

Theorem: Assume CRH exists. ∀ NP language verified by a uniform RAM machine running in **time** $\boldsymbol{T}$ and **space** $\boldsymbol{S}$ ∃ public-coin (ZK) arg. with soundness $\boldsymbol{negl(k)}$ such that:

	Time	Space
$\boxed{\mathbf{P}}$	$\boldsymbol{\tilde{O}(T)}$	$\boldsymbol{\tilde{O}(S)}$

where the CRH is used in a black-box way and $\boldsymbol{\tilde{O}(\quad)}$ ignores $\boldsymbol{poly(\log(T), k)}$

Main Result – Upper Bound

Theorem: Assume CRH exists. ∀ NP language verified by a uniform RAM machine running in **time** $\boldsymbol{T}$ and **space** $\boldsymbol{S}$ ∃ public-coin (ZK) arg. with soundness $\boldsymbol{negl(k)}$ such that:

	Time	Space
$\boxed{\mathbf{P}}$	$\tilde{\boldsymbol{O}}(\boldsymbol{T})$	$\tilde{\boldsymbol{O}}(\boldsymbol{S})$

$\longrightarrow$ Complexity preserving

where the CRH is used in a black-box way and $\tilde{\boldsymbol{O}}(\quad)$ ignores $\boldsymbol{poly}(\mathbf{log}(\boldsymbol{T}), \boldsymbol{k})$

Main Result – Upper Bound

Theorem: Assume CRH exists. ∀ NP language verified by a uniform RAM machine running in **time** $\boldsymbol{T}$ and **space** $\boldsymbol{S}$ ∃ public-coin (ZK) arg. with soundness $\boldsymbol{negl(k)}$ such that:

	Time	Space
P	$\tilde{O}(T)$	$\tilde{O}(S)$
V	$\tilde{O}\left(\frac{T}{S} + S\right)$	$\tilde{O}(1)$

→ Complexity preserving

where the CRH is used in a black-box way and $\tilde{O}(\;)$ ignores $\boldsymbol{poly(\log(T), k)}$

Main Result – Upper Bound

Theorem: Assume CRH exists. ∀ NP language verified by a uniform RAM machine running in **time** $\boldsymbol{T}$ and **space** $\boldsymbol{S}$ ∃ public-coin (ZK) arg. with soundness $\boldsymbol{negl(k)}$ such that:

	Time	Space	
P	$\tilde{O}(T)$	$\tilde{O}(S)$	→ Complexity preserving
V	$\tilde{O}\left(\frac{T}{S}+S\right)$	$\tilde{O}(1)$	

Proof Length is $\tilde{O}\left(\frac{T}{S}\right)$

where the CRH is used in a black-box way and $\tilde{O}(\quad)$ ignores $\boldsymbol{poly}(\mathbf{log}(\boldsymbol{T}), \boldsymbol{k})$

Main Result – Upper Bound

Theorem: Assume CRH exists. ∀ NP language verified by a uniform RAM machine running in **time** $\boldsymbol{T}$ and **space** $\boldsymbol{S}$ ∃ public-coin (ZK) arg. with soundness $\boldsymbol{negl(k)}$ such that:

	Time	Space	
P	$\tilde{O}(T)$	$\tilde{O}(S)$	→ Complexity preserving
V	$\tilde{O}\left(\frac{T}{S}+S\right)$	$\tilde{O}(1)$	Sublinear

Proof Length is $\tilde{O}\left(\frac{T}{S}\right)$

where the CRH is used in a black-box way and $\tilde{O}(\ )$ ignores $\boldsymbol{poly(\log(T), k)}$

Main Result – Lower Bound*

Theorem: Assume CRH exists. ∀ NP language verified by a uniform RAM machine running in **time** $\boldsymbol{T}$ and **space** $\boldsymbol{S}$ ∃ public-coin (ZK) arg. with soundness $\boldsymbol{negl(k)}$ such that:

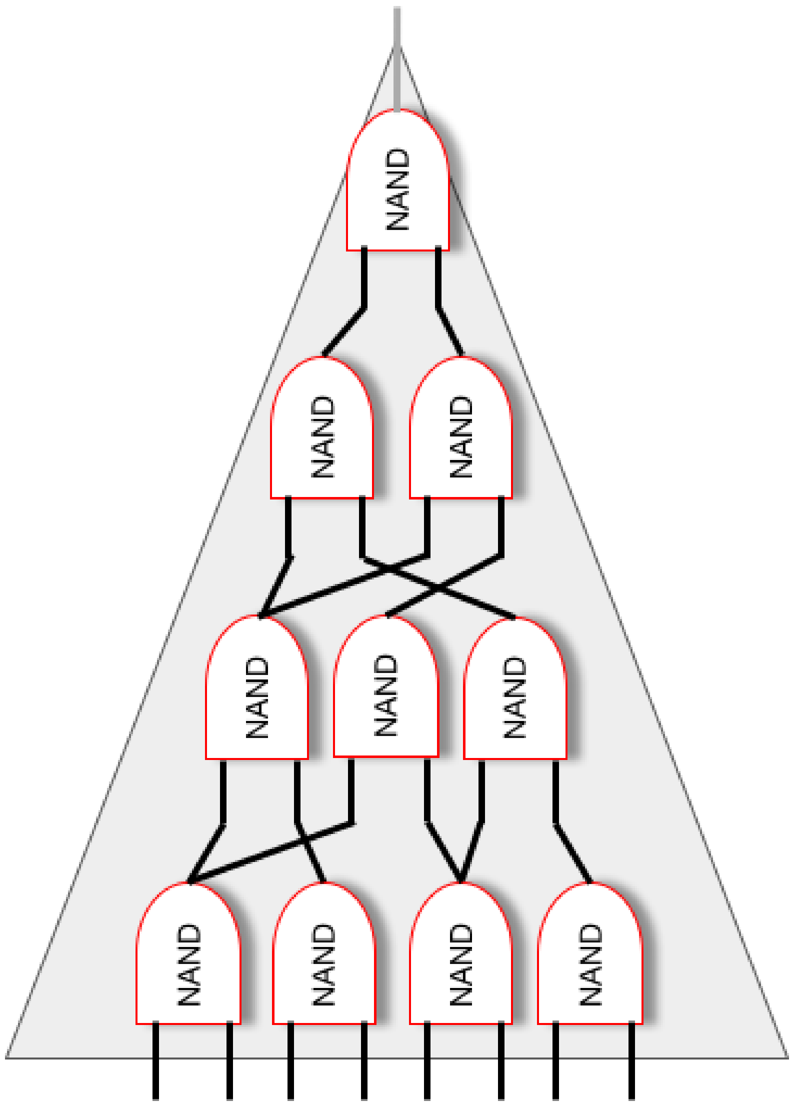
	Time	Space
P	$\tilde{O}(T)$	
V	$\tilde{O}\left(\frac{T}{S}+S\right)$	

Proof Length is $\tilde{O}\left(\frac{T}{S}\right)$

Our lower bound shows why it's hard to further improve the proof length.

where the CRH is used in a black-box way and $\tilde{O}(\quad)$ ignores $\boldsymbol{poly(\log(T), k)}$

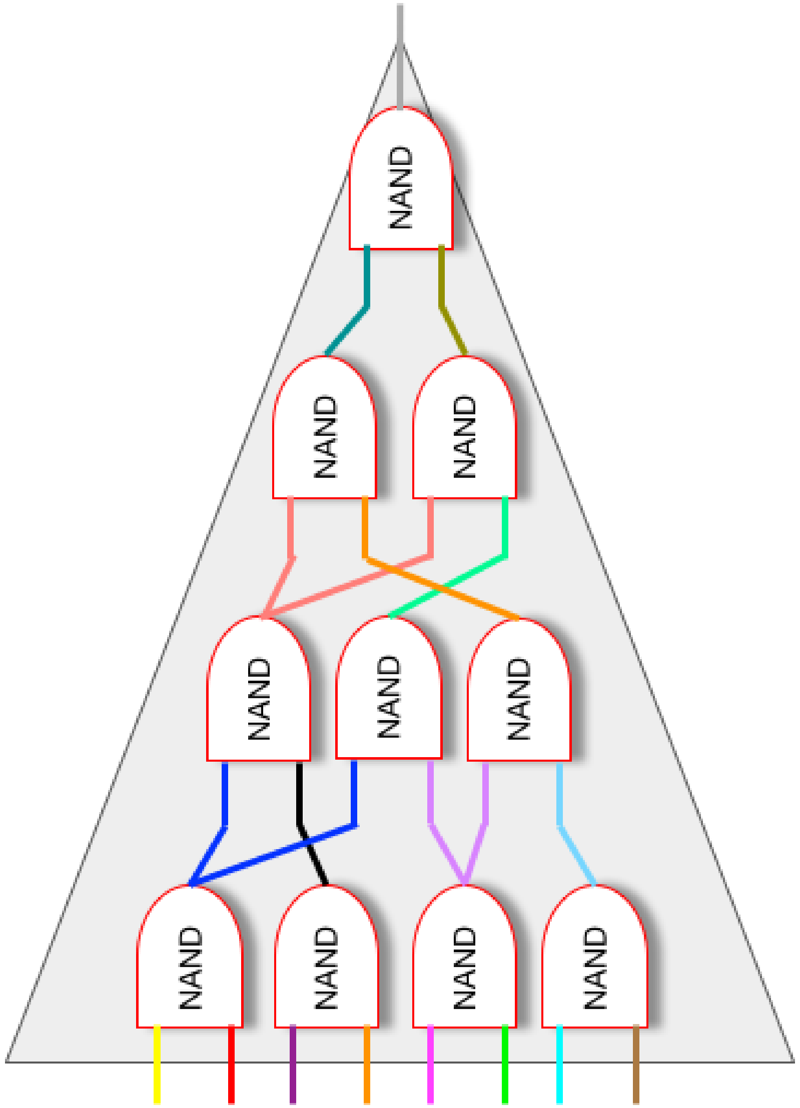
Ligero [AHIV17]
Proof Schematic



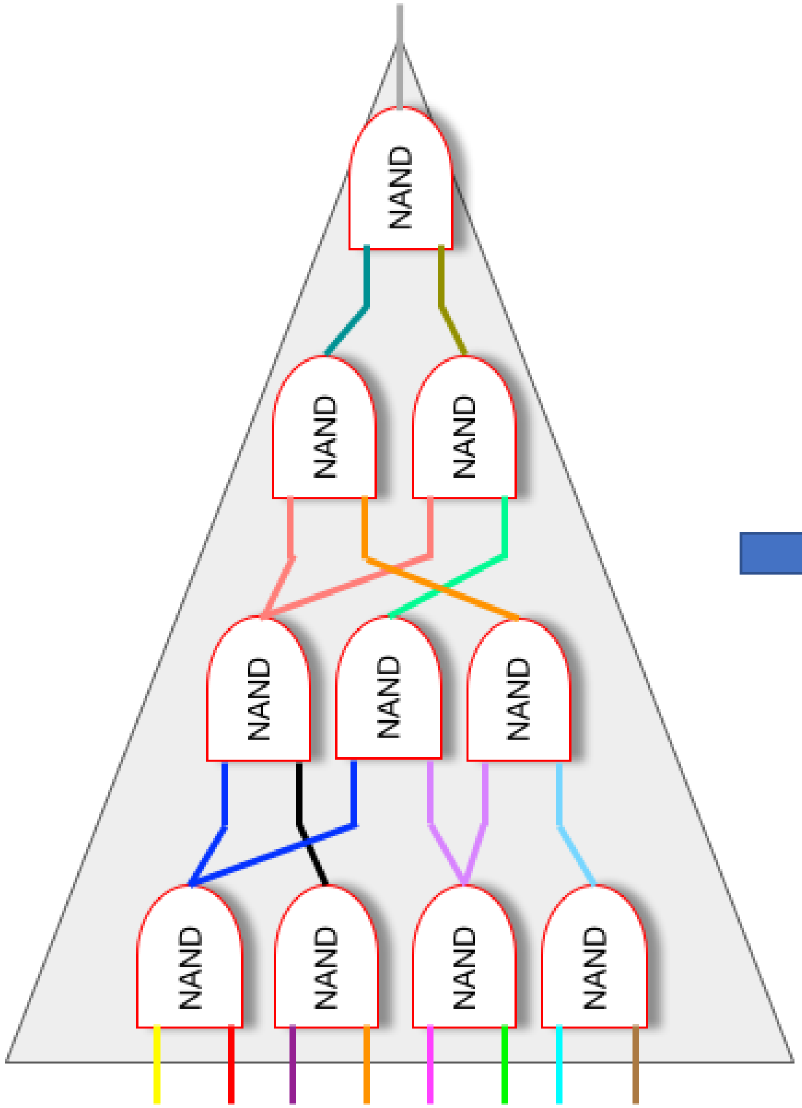
Given a circuit C

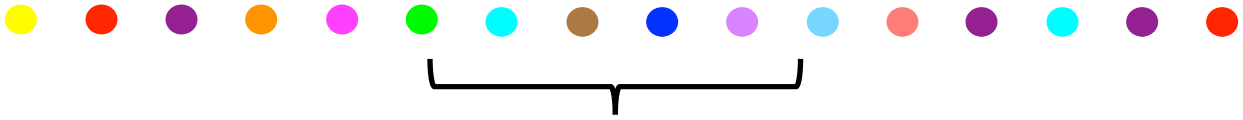
Prove that $\exists z$, such that $C(z) = 1$

Extended witness $\overline{w}$

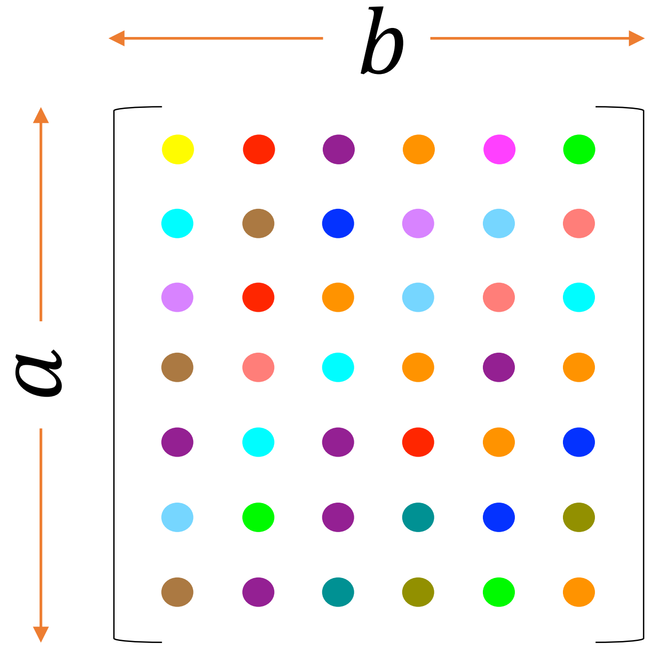


Extended witness $\overline{w}$



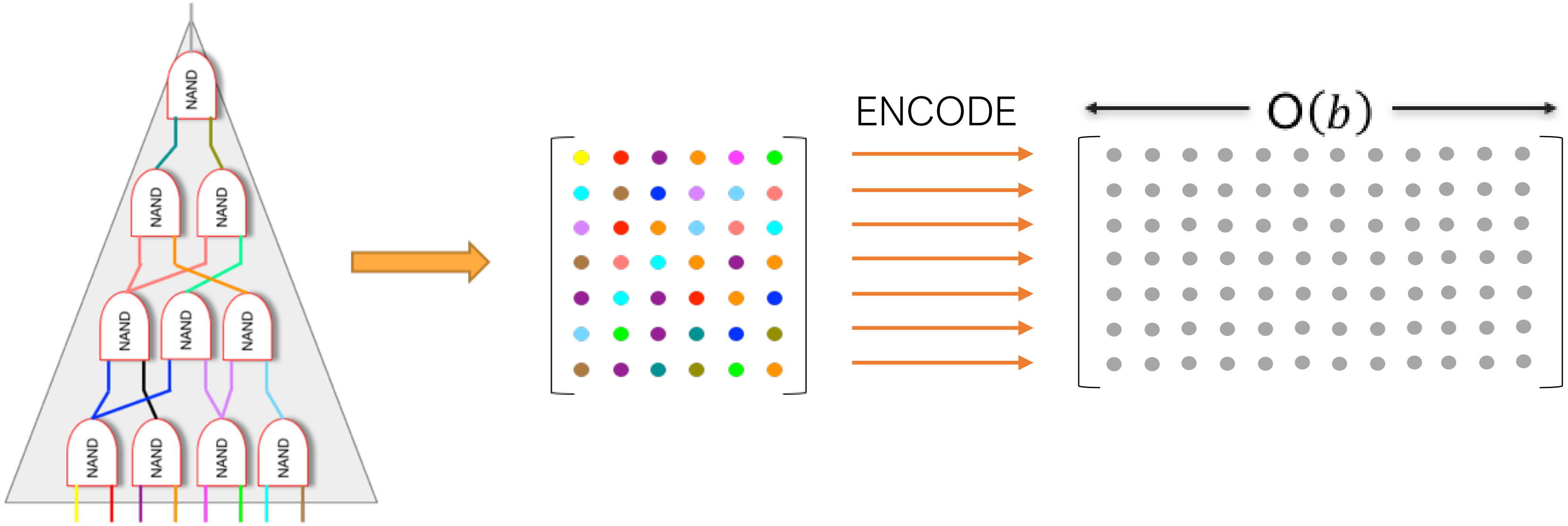


i^{th} chunk



i^{th} row $\overline{w}_i$
is i^{th} chunk of $\overline{w}$

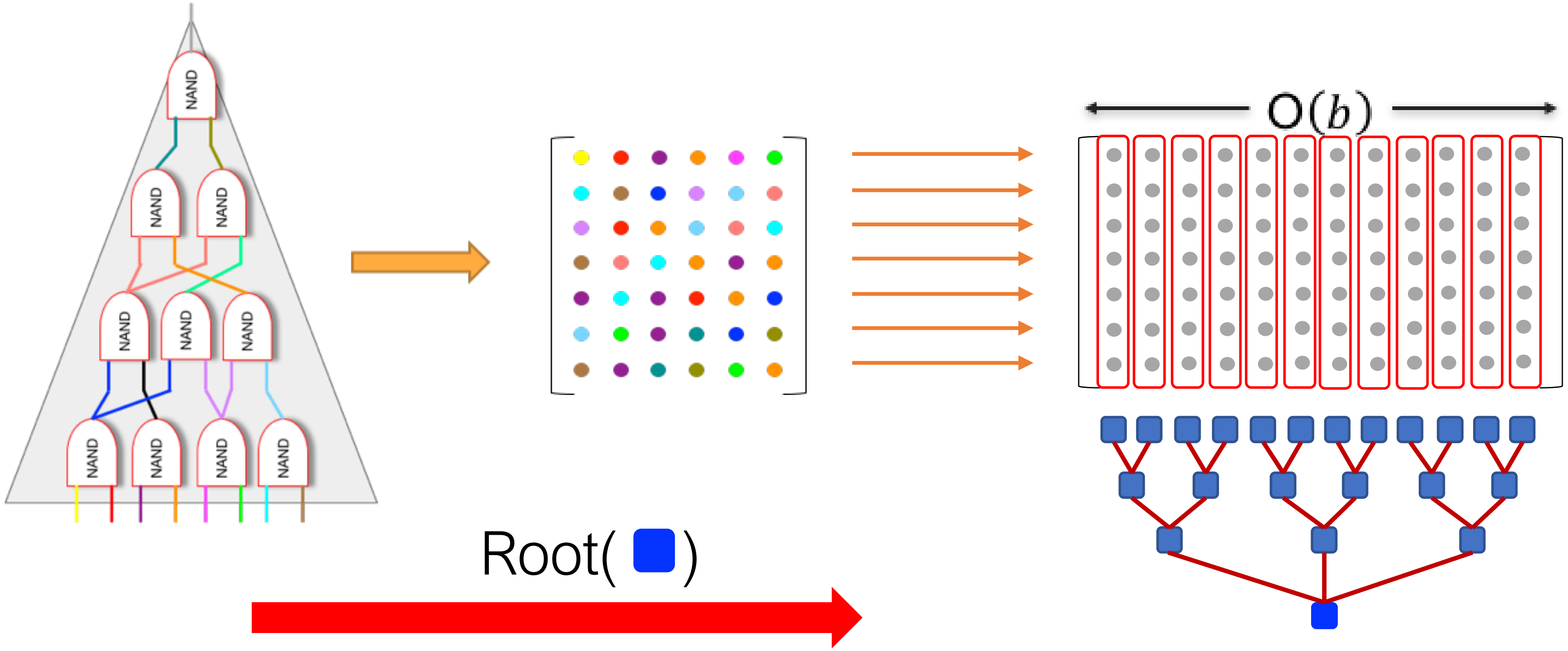
$$a \cdot b \geq X \cdot \#gates$$



P

V

Compute Merkle Root

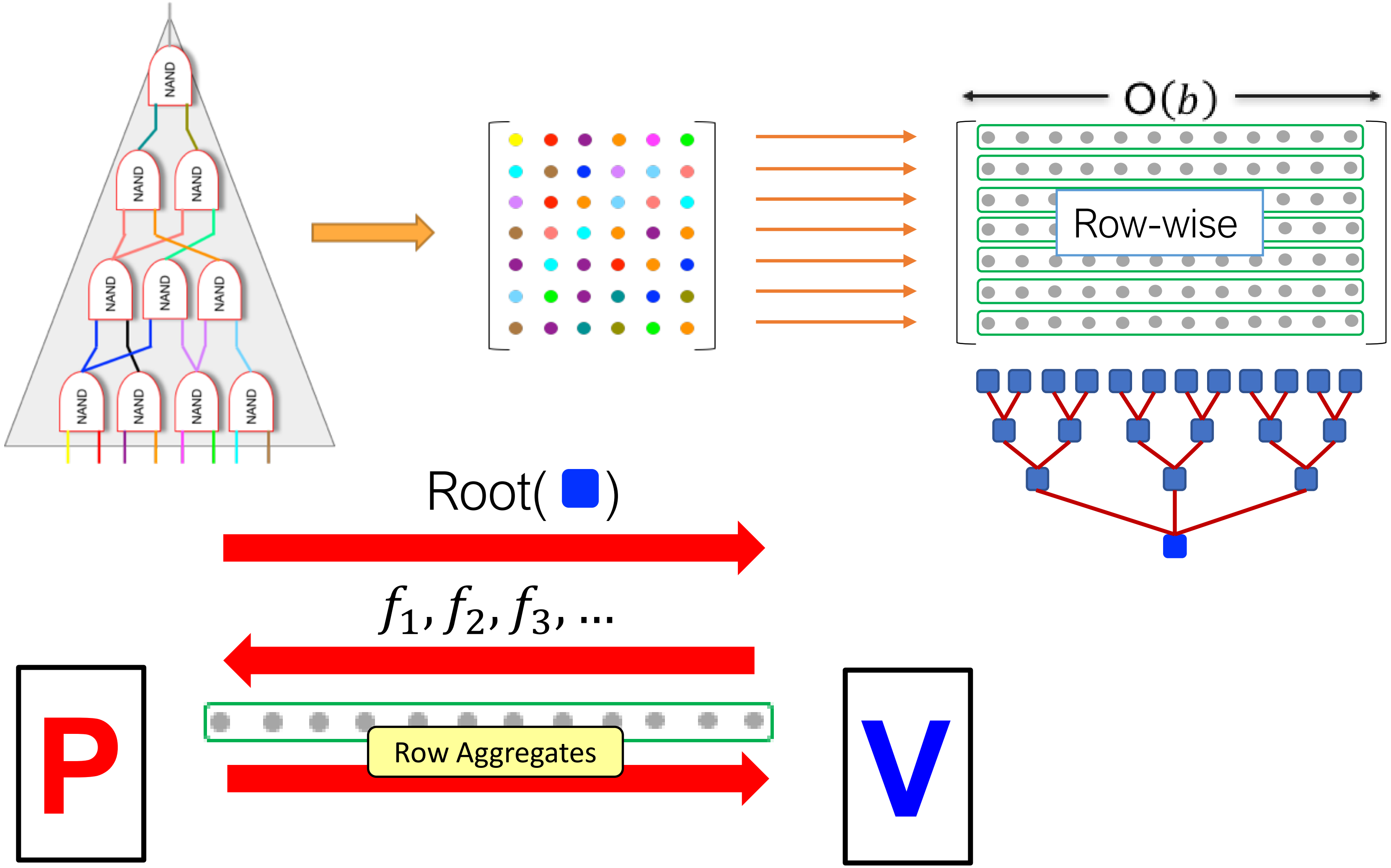


P

V

Compute Merkle Root

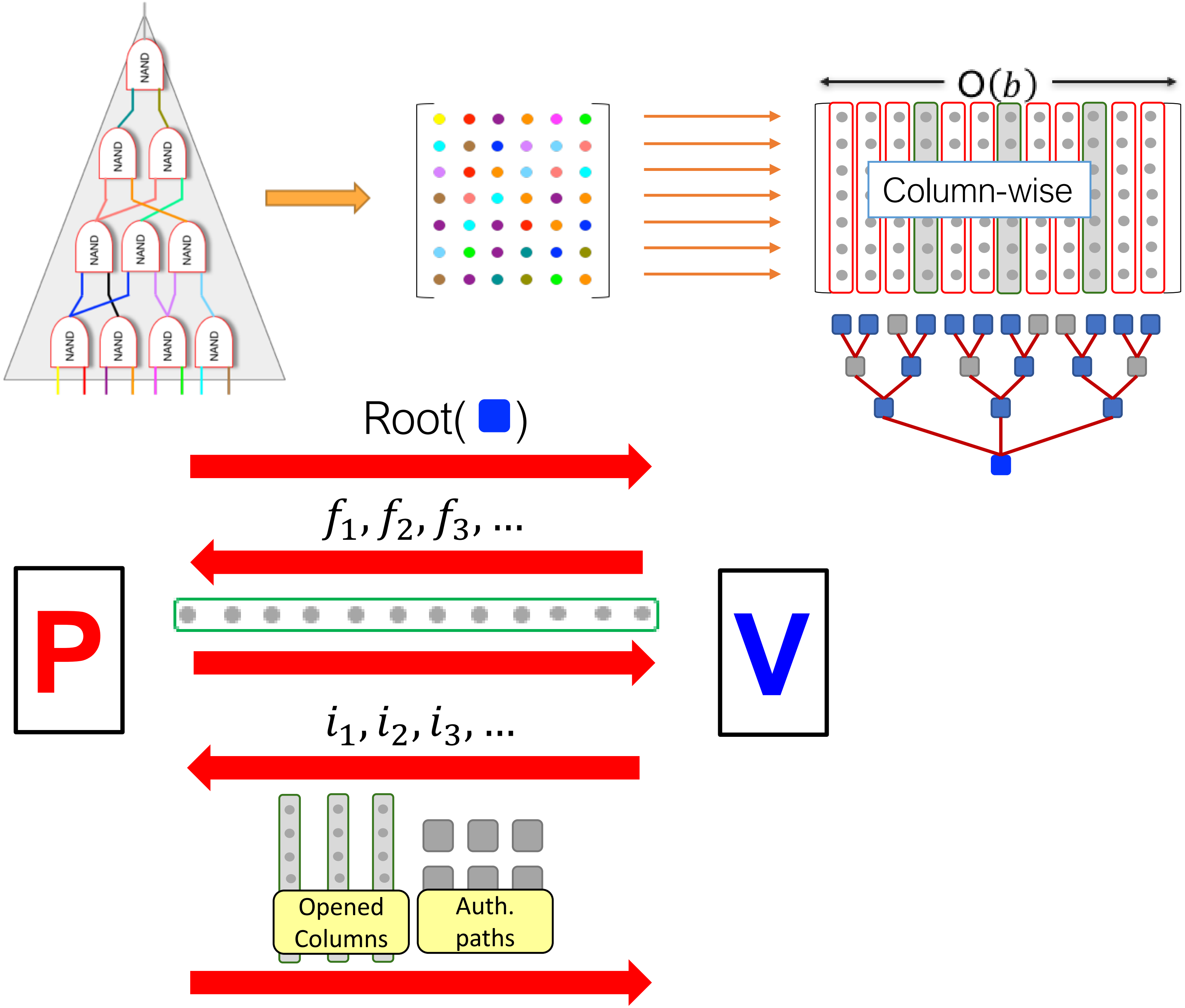
Compute Row Aggregates



Compute Merkle Root

Compute Row Aggregates

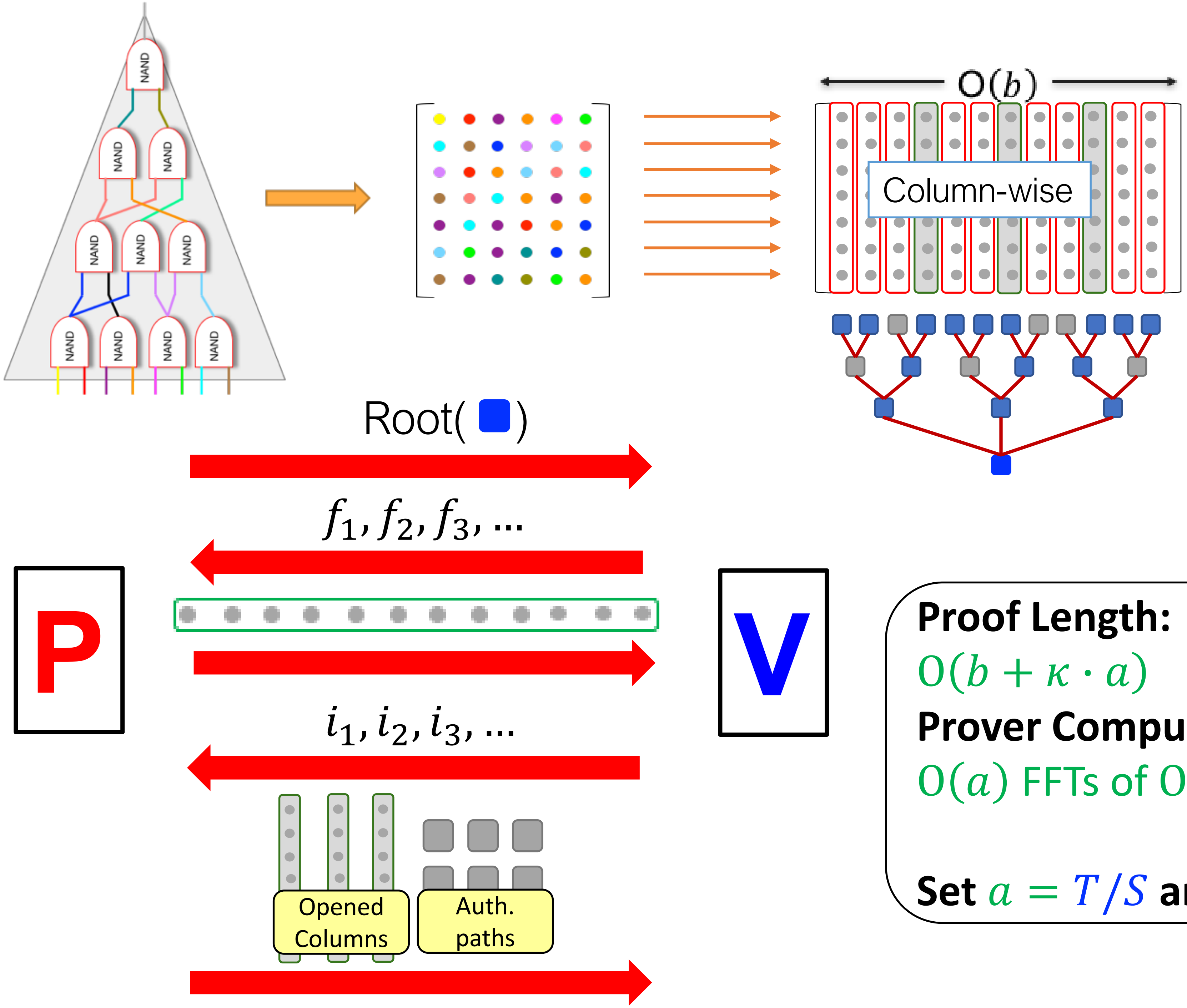
Decommit Columns



Compute Merkle Root

Compute Row Aggregates

Decommit Columns



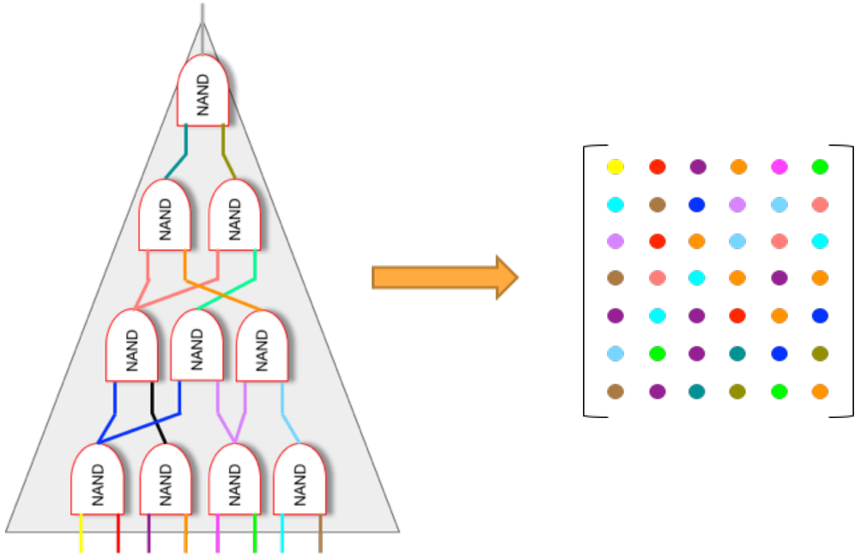
Proof Length:

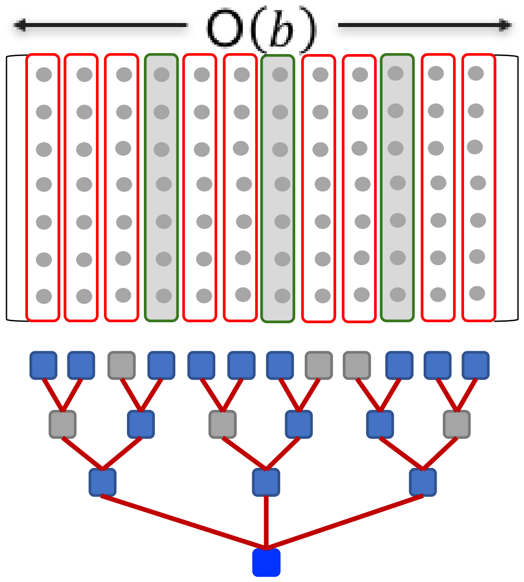
$O(b + \kappa \cdot a)$

Prover Computation:

$O(a)$ FFTs of $O(b)$

Set $a = T/S$ **and** $b = S$

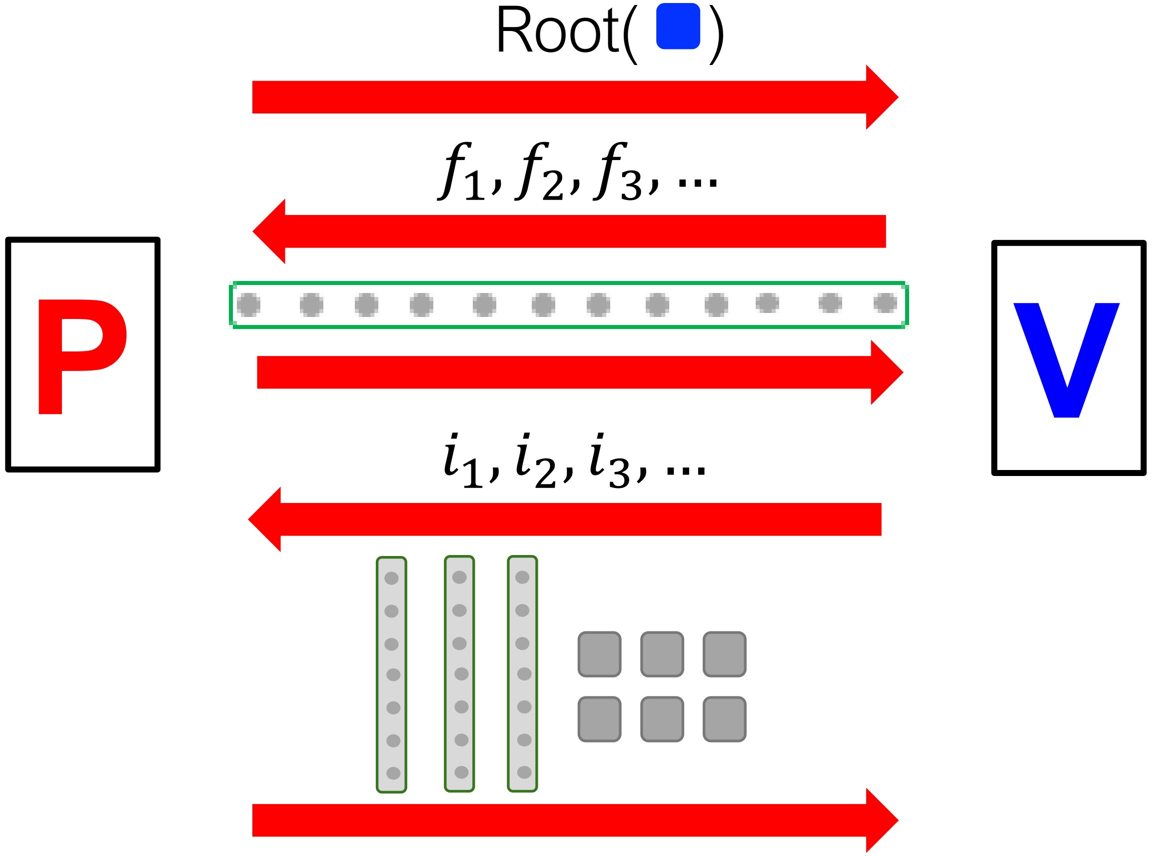




Compute Merkle Root

Compute Row Aggregates

Decommit Columns



Proof Length:

$\tilde{O}\left(\frac{T}{S} + S\right)$

Prover Computation:

$\tilde{\boldsymbol{O}}(T)$

Set $a = T/S$ **and** $b = S$

What about space-efficiency?

Compute Merkle Root

Compute Row Aggregates

Decommit Columns

$O(S)$ space if the matrix can be computed row by row

What about space-efficiency?

Compute Merkle Root

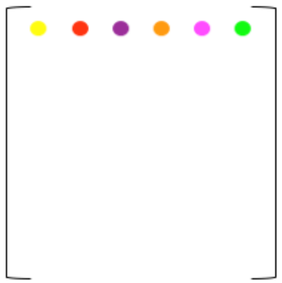
Compute Row Aggregates

Decommit Columns

$O(S)$ space if the matrix can be computed row by row

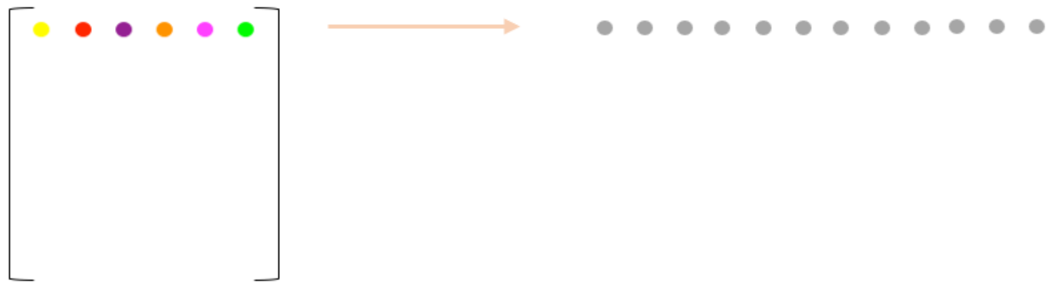
[BCGT13] RAM program => succinct circuits => Transcript generated in time T and space S

Computing Merkle Root Space Efficiently



Computing Merkle Root Space Efficiently

Computing Merkle Root Space Efficiently

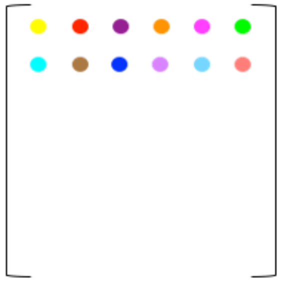


Computing Merkle Root Space Efficiently

Computing Merkle Root Space Efficiently

Computing Merkle Root Space Efficiently

Computing Merkle Root Space Efficiently

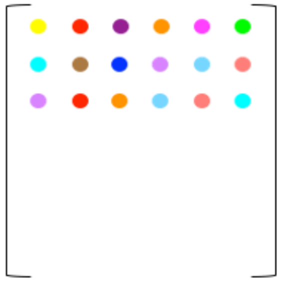


Computing Merkle Root Space Efficiently

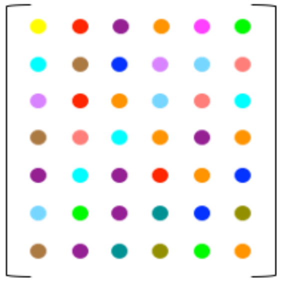
Computing Merkle Root Space Efficiently

Computing Merkle Root Space Efficiently

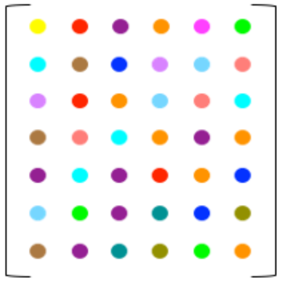
Computing Merkle Root Space Efficiently

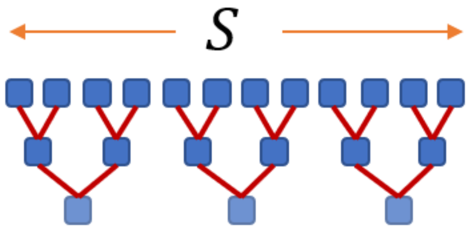


Computing Merkle Root Space Efficiently

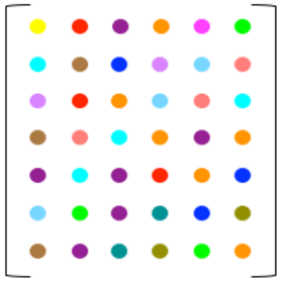


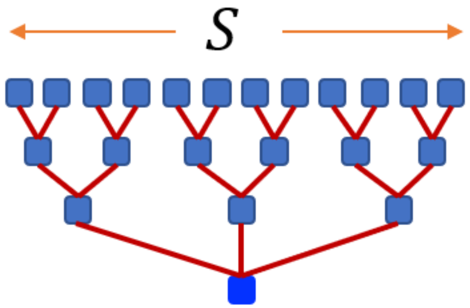
Computing Merkle Root Space Efficiently





Computing Merkle Root Space Efficiently





What about space-efficiency of Row Aggregates?

Compute Merkle Root

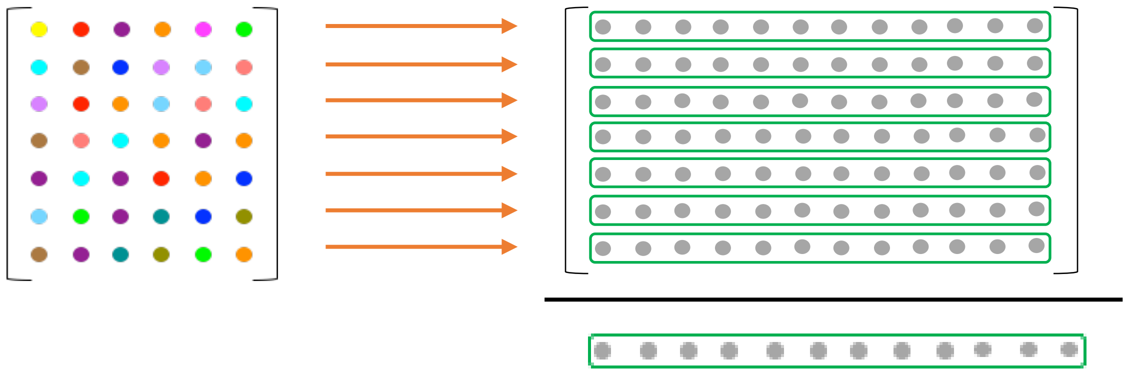
Compute Row Aggregates

Decommit Columns

Code test - the prover encoded each row correctly

Quadratic test - the prover computed multiplication gates correctly

Linear test - the prover computed “linear” gates correctly



Row Aggregate

What about space-efficiency of Row Aggregates?

Compute Merkle Root

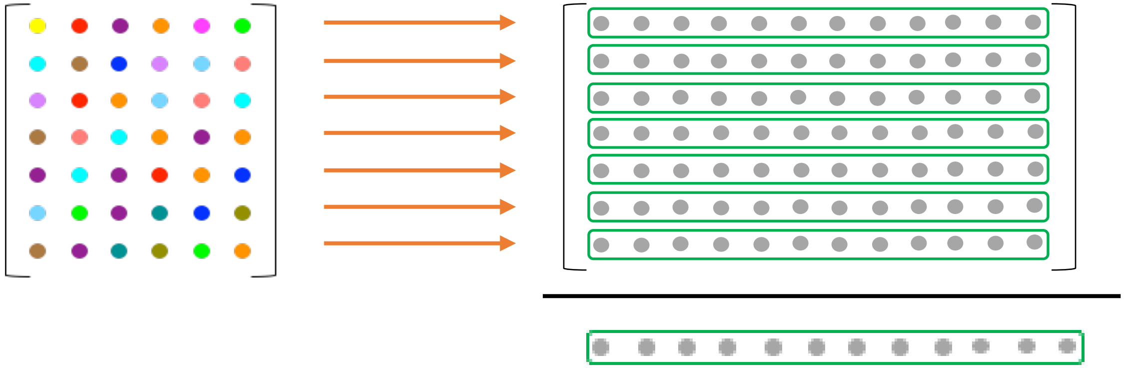
Compute Row Aggregates

Decommit Columns

Code test - the prover encoded each row correctly

Quadratic test - the prover computed multiplication gates correctly

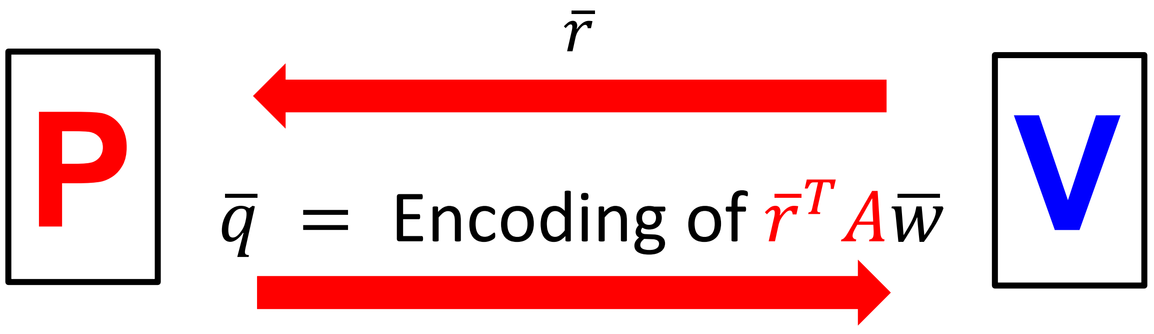
Linear test - the prover computed "linear" gates correctly



Row Aggregate

Linear gates were correct

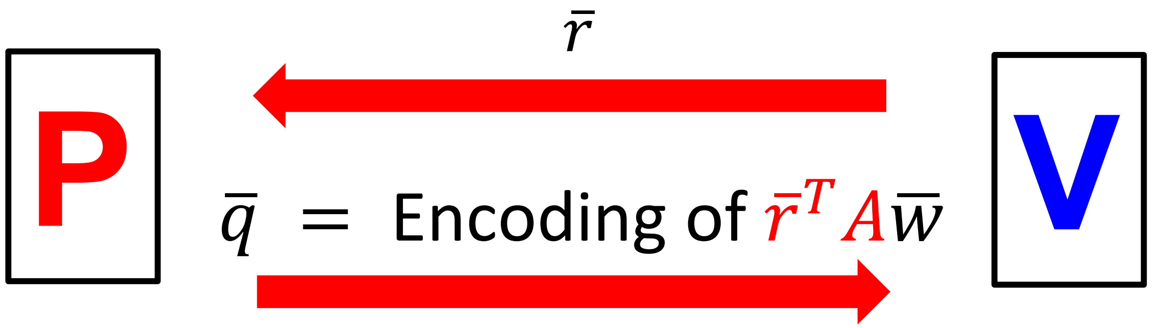
Linear constraints can be expressed as $A\bar{w} = \bar{b}$



Check if $\bar{q}$ is a valid codeword encoding $\bar{r}^T \bar{b}$ and agrees on revealed columns

Linear gates were correct

Linear constraints can be expressed as $A\bar{w} = \bar{b}$



Check if $\bar{q}$ is a valid codeword encoding $\bar{r}^T\bar{b}$ and agrees on revealed columns

Main Challenge: How to compute $\bar{r}^T A$ efficiently?

Linear gates were correct

Linear constraints can be expressed as $A\bar{w} = \bar{b}$

$\bar{r}$

P $\longleftarrow$ **V**

$\bar{q}$ = Encoding of $\bar{r}^T A\bar{w}$ $\longrightarrow$

Check if $\bar{q}$ is a valid codeword encoding $\bar{r}^T \bar{b}$ and agrees on revealed columns

Main Challenge: How to compute $\bar{r}^T A$ efficiently?

Size of r: $\tilde{O}(T)$

Size of A: $\tilde{O}(T) \times \tilde{O}(T)$

Linear gates were correct

Linear constraints can be expressed as $A\bar{w} = \bar{b}$

$\bar{r}$

P

$\bar{q} = $ Encoding of $\bar{r}^T A\bar{w}$

V

Check if $\bar{q}$ is a valid codeword encoding $\bar{r}^T\bar{b}$ and agrees on revealed columns

Main Challenge: How to compute $\bar{r}^T A$ efficiently?

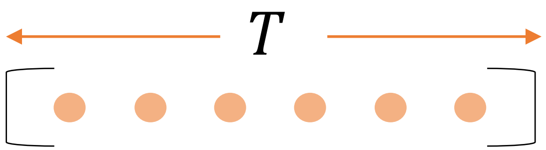
Size of r: $\tilde{O}(T)$

Size of A: $\tilde{O}(T) \times \tilde{O}(T)$

Naive Approach: $\tilde{O}(T^2)$ time & $\tilde{O}(T^2)$ space.

How to compute $\bar{r}^T A$ efficiently?

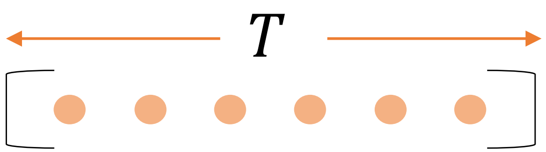
r



r requires $O(T)$ space!

How to compute $\bar{r}^T A$ efficiently?

r

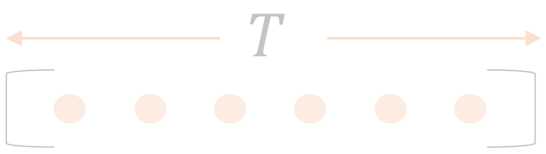


r requires $O(T)$ space!

- Generate r from a seed
- Required space: $\tilde{O}(1)$

How to compute $\bar{r}^T A$ efficiently?

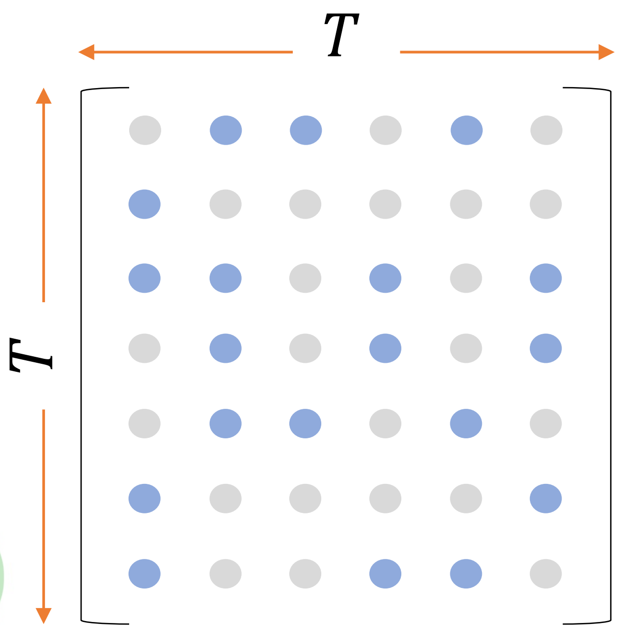
r



r requires $O(T)$ space!

- Generate r from a seed
- Required space: $\tilde{O}(1)$

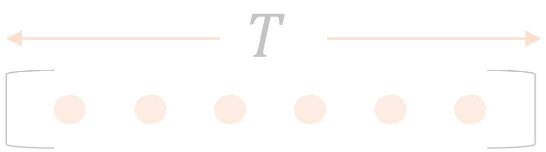
A



A is a large matrix!

How to compute $\bar{r}^T A$ efficiently?

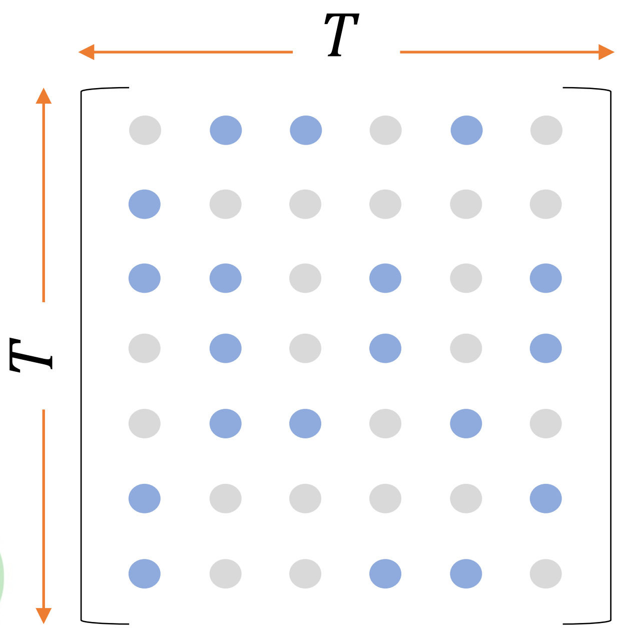
r



r requires $O(T)$ space!

- Generate r from a seed
- Required space: $\tilde{O}(1)$

A

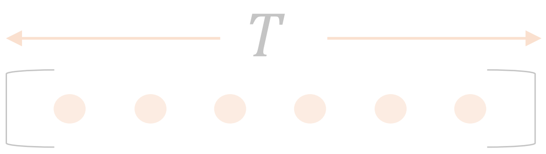


A is a large matrix!

- A is sparse and structured

How to compute $\bar{r}^T A$ efficiently?

r

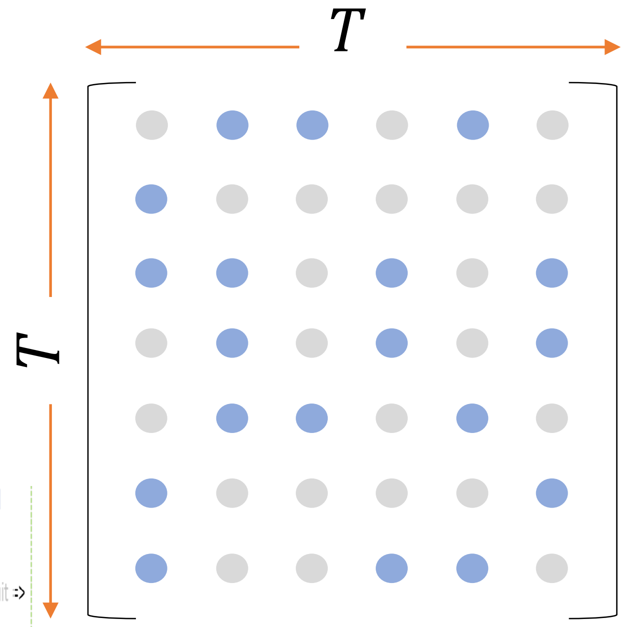


r requires $O(T)$ space!

- Generate r from a seed
- Required space: $\tilde{O}(1)$

- A is sparse and structured
- [BCGT13] Uniform RAM machine => Succinct Circuit => Succinct matrix

A

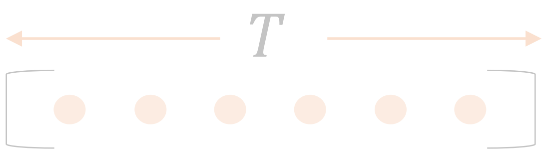


A is a large matrix!

- A is sparse and structured
- [BCGT13] Uniform RAM machine => Succinct Circuit => Succinct matrix

How to compute $\bar{r}^T A$ efficiently?

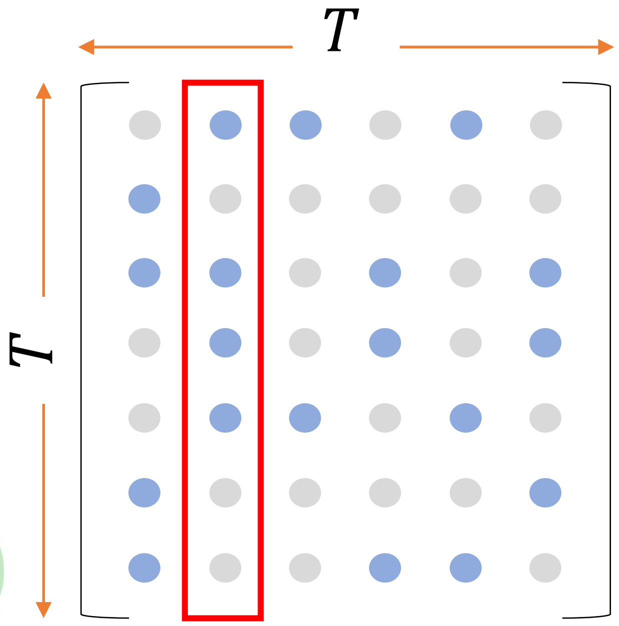
r



r requires $O(T)$ space!

- Generate r from a seed
- Required space: $\tilde{O}(1)$

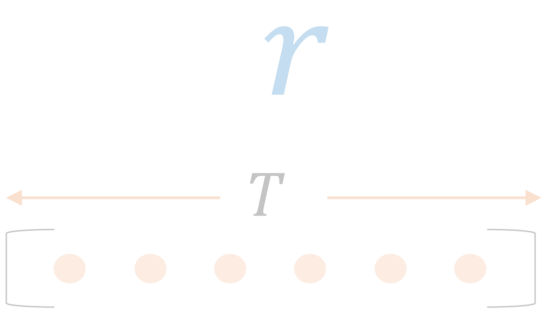
A



A is a large matrix!

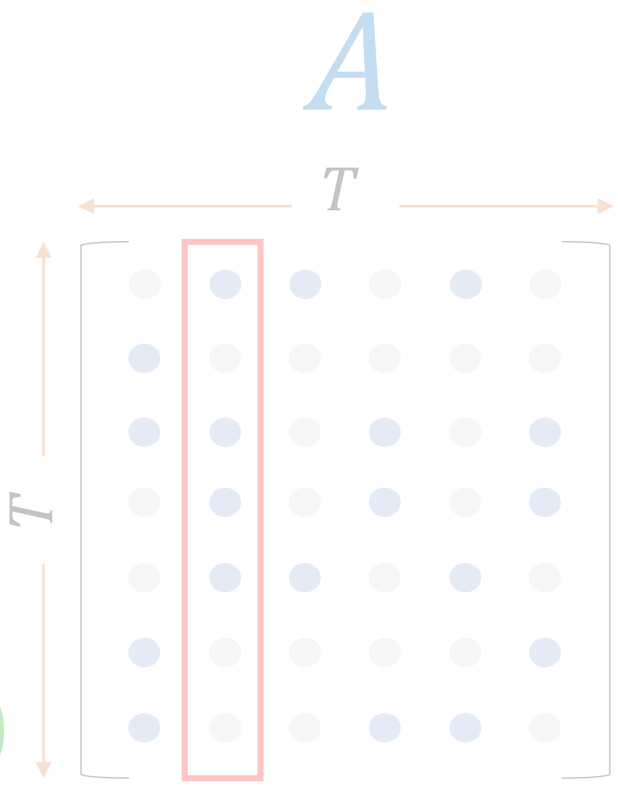
- A is sparse and structured
- [BCGT13] Uniform RAM machine => Succinct Circuit => Succinct matrix
- Non-zero elements in each column can be computed in $\tilde{O}(1)$ time without storing A.

How to compute $\bar{r}^T A$ efficiently?



r requires $O(T)$ space!

- Generate r from a seed
- Required space: $\tilde{O}(1)$

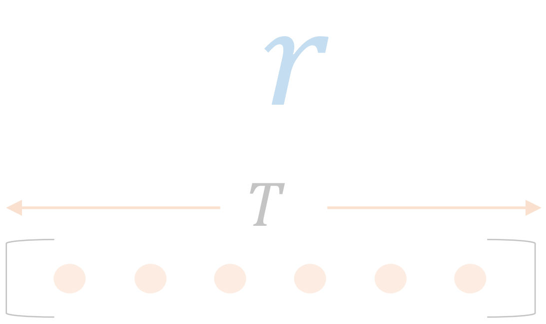


A is a large matrix!

- A is sparse and structured
- [BCGT13] Uniform RAM machine => Succinct Circuit => Succinct matrix
- Non-zero elements in each column can be computed in $\tilde{O}(1)$ time without storing A.

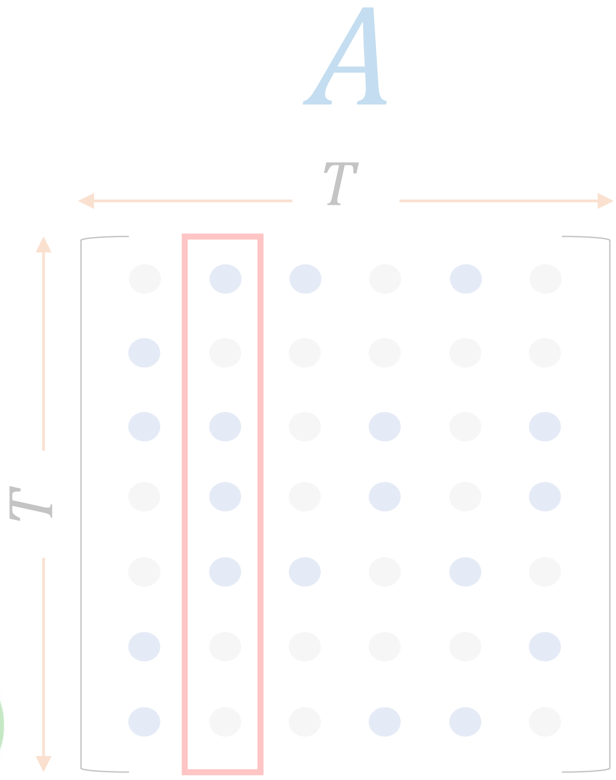
Hence $r^T A$ can be computed row-by-row in time $\tilde{O}(T)$ and space $\tilde{O}(S)$

How to compute $\bar{r}^T A$ efficiently?



r requires $O(T)$ space!

- Generate r from a seed
- Required space: $\tilde{O}(1)$



A is a large matrix!

- A is sparse and structured
- [BCGT13] Uniform RAM machine => Succinct Circuit => Succinct matrix
- Non-zero elements in each column can be computed in $\tilde{O}(1)$ time without storing A.

Hence $r^T A$ can be computed row-by-row in time $\tilde{O}(T)$ and space $\tilde{O}(S)$

P is complexity-preserving.

Main Result – Lower Bound*

Theorem: Assume CRH exists. ∀ NP language verified by a uniform RAM machine running in **time** $\boldsymbol{T}$ and **space** $\boldsymbol{S}$ ∃ public-coin (ZK) arg. with soundness $\boldsymbol{negl(k)}$ such that:

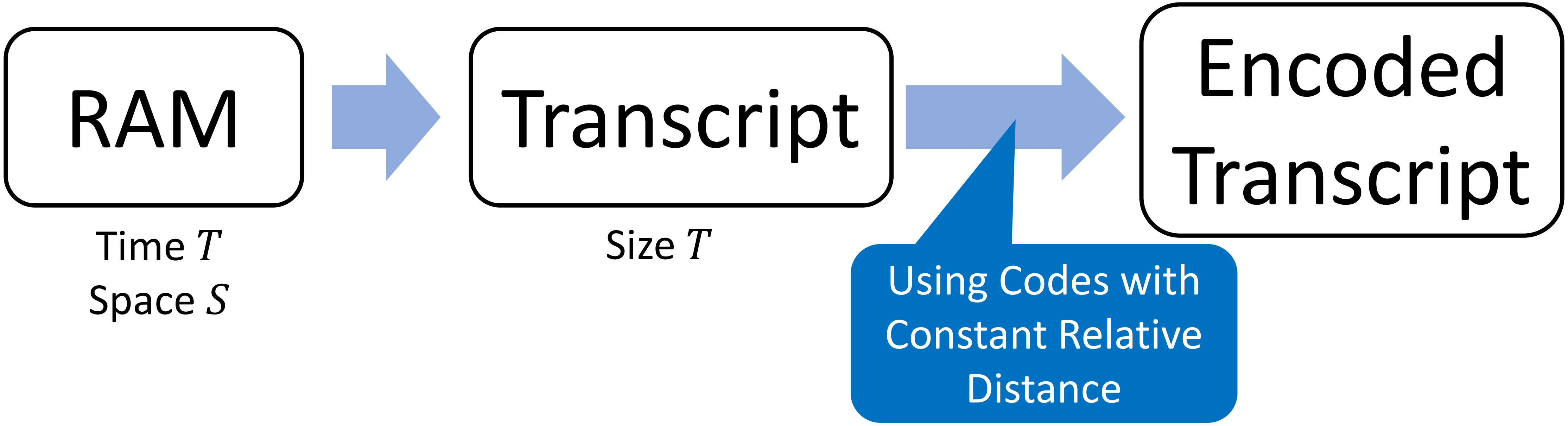
	Time	Space
P	$\tilde{O}(T)$	
V	$\tilde{O}\left(\frac{T}{S}+S\right)$	

Our lower bound shows why it's hard to further improve the proof length.

Proof Length is $\tilde{\boldsymbol{O}}\left(\frac{\boldsymbol{T}}{\boldsymbol{S}}\right)$

where the CRH is used in a black-box way and $\tilde{\boldsymbol{O}}(\quad)$ ignores $\boldsymbol{poly}(\mathbf{log}(\boldsymbol{T}), \boldsymbol{k})$

Current Techniques*: Most ZK-SNARKs rely on Codes…



Current Techniques*: Most ZK-SNARKs rely on Codes…

RAM → Transcript → Encoded Transcript

Time T
Space S

Size T

Using Codes with Constant Relative Distance

"Lower bound" on the encoding algorithm

provides evidence

Current Techniques*: Most ZK-SNARKs rely on Codes…

RAM → Transcript → Encoded Transcript

Time T

Space S

Size T

Using Codes with Constant Relative Distance

"Lower bound" on the encoding algorithm

provides evidence

"Hard" to reduce the proof length below $\tilde{O}\left(\frac{T}{S}\right)$

Lower Bound*

Theorem: Let C be $[n, m, \delta m]$ code over $\mathbb{F}$ then any r-pass Encode RAM algorithm requires space

$$S > \frac{\delta n}{r} \log|\mathbb{F}|$$

Lower Bound*

Theorem: Let C be $[n, m, \delta m]$ code over $\mathbb{F}$ then any r-pass Encode RAM algorithm requires space

$$S > \frac{\delta n}{r} \log|\mathbb{F}|$$

[YRST02] Proved the lower bound for 1-pass algorithm

Lower Bound*

Theorem: Let C be $[n, m, \delta m]$ code over $\mathbb{F}$ then any r-pass Encode RAM algorithm requires space

$$S > \frac{\delta n}{r} \log|\mathbb{F}|$$

[YRST02] Proved the lower bound for 1-pass algorithm

Setting $n = T$ and $|\mathbb{F}| = \log T$ we get that $\delta < \frac{S \cdot r}{T \cdot \log T}$

=> Minimum number of oracle queries is $\tilde{O}\left(\frac{1}{\delta}\right) \quad = \tilde{O}\left(\frac{T}{S}\right)$

Summary

- First construction of complexity-preserving sublinear (ZK) arguments from CRH in a black-box way

- Improving the communication will require new techniques – i.e., getting around constant-distance codes.

Thank You

Additional Slides

Warmup Lower Bound

Simplifying assumptions:

1. Only one pass over the message
2. No work tape
3. Codeword head is independent of the contents of the message

Warmup Lower Bound

Simplifying assumptions:

1. Only one pass over the message
2. No work tape
3. Codeword head is independent of the contents of the message