Advanced zk-STARKs

Alan Szepieniec
艾伦·佘丕涅茨
alan@neptune.cash

https://neptune.cash/

https://triton-vm.org/

https://asz.ink/presentations/2025-09-18-Advanced-zkSTARKs.pdf

Table of Contents

Retrospective

Optimizations
- Batching
- Quotient Segmentation
- Grinding

Enhancements
- Zero-Knowledge
- Randomized AIR with Preprocessing

VM Architecture
- Overview
- Communication Arguments
- Memory

Other Topics

Table of Contents

Retrospective

Optimizations

- Batching
- Quotient Segmentation
- Grinding

Enhancements

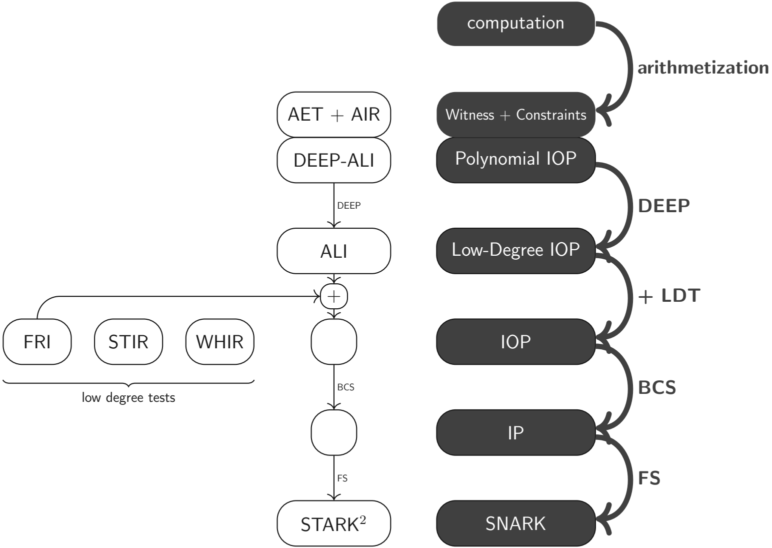
- Zero-Knowledge
- Randomized AIR with Preprocessing

VM Architecture

- Overview
- Communication Arguments
- Memory

Other Topics

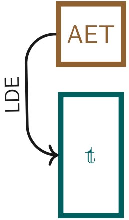
STARK Compilation Pipeline



STARK Diagram

algebraic execution trace

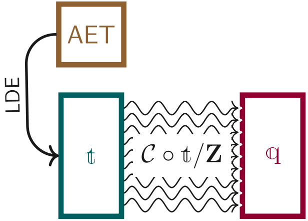
STARK Diagram



low-degree extension

low-degree extended trace

STARK Diagram

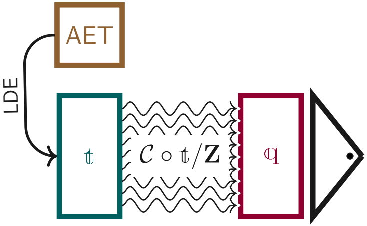


composition with AIR constraints

division by zerofiers

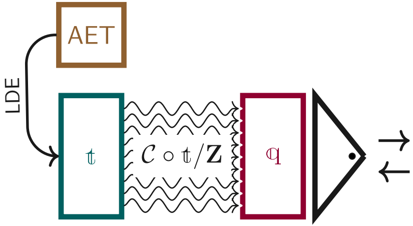
quotients

STARK Diagram



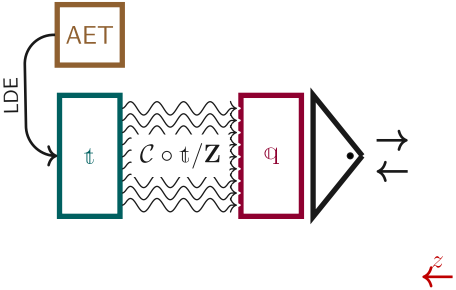
build Merkle tree

STARK Diagram



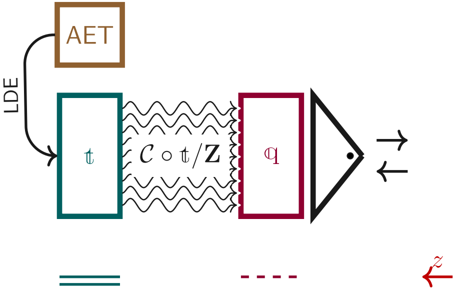
interact with verifier

STARK Diagram



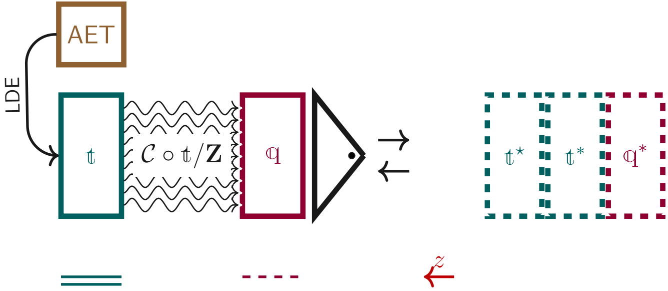
sample out-of-domain point

STARK Diagram



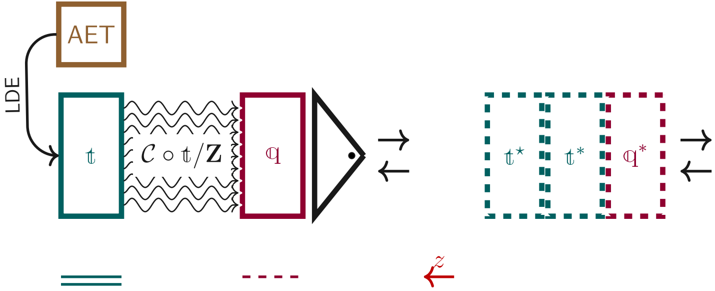
produce out-of-domain rows

STARK Diagram



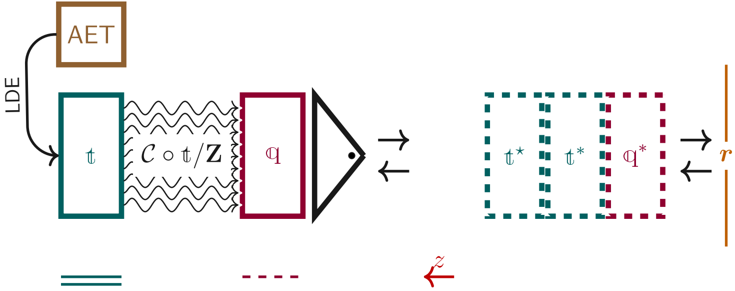
apply DEEP update

STARK Diagram



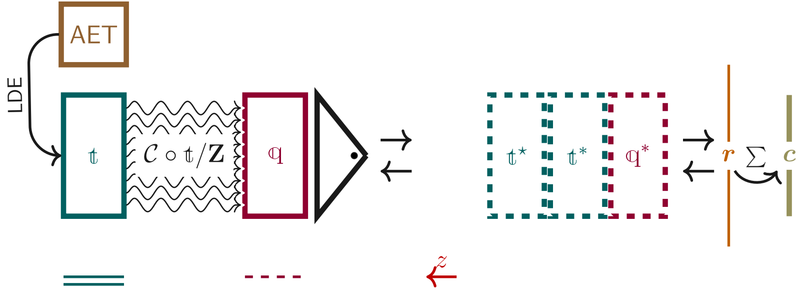
interact with verifier

STARK Diagram



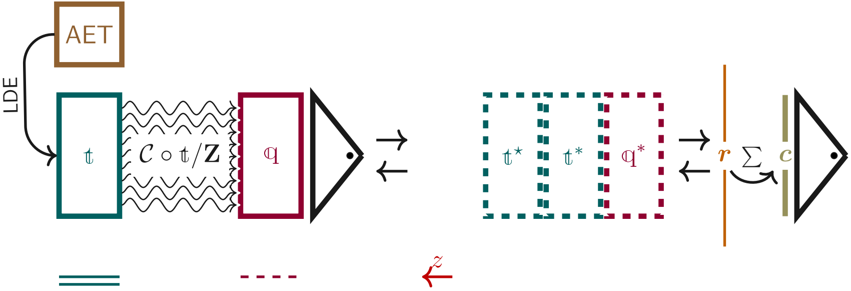
sample weights

STARK Diagram



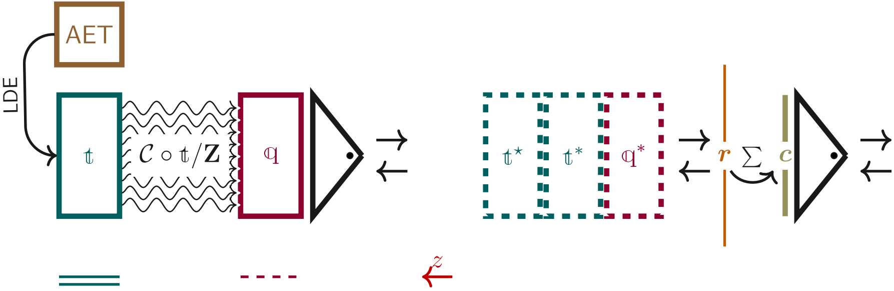
random linear combination

STARK Diagram



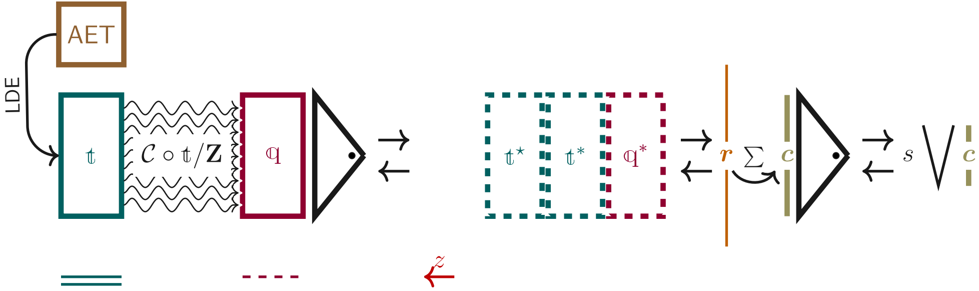
build Merkle tree

STARK Diagram



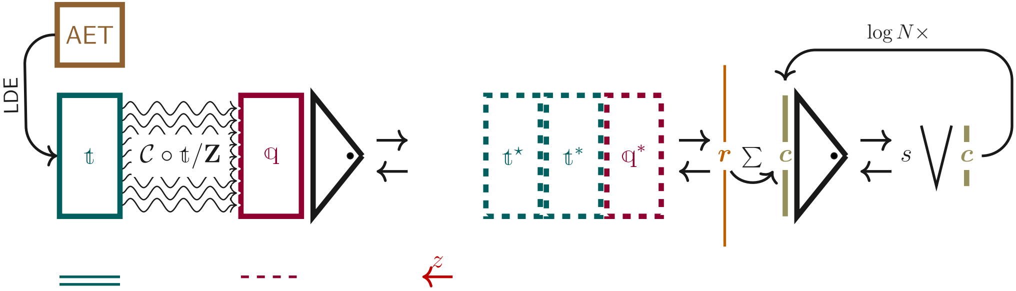
interact with verifier

STARK Diagram



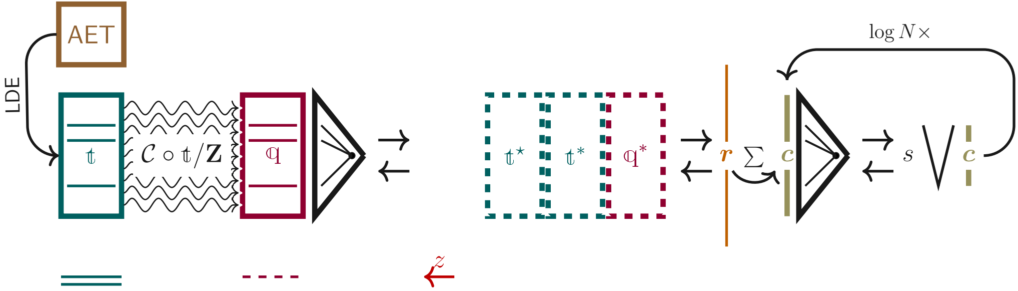
split-and-fold

STARK Diagram



rinse and repeat

STARK Diagram



obtain FRI indices

open indicated rows

STARK Diagram

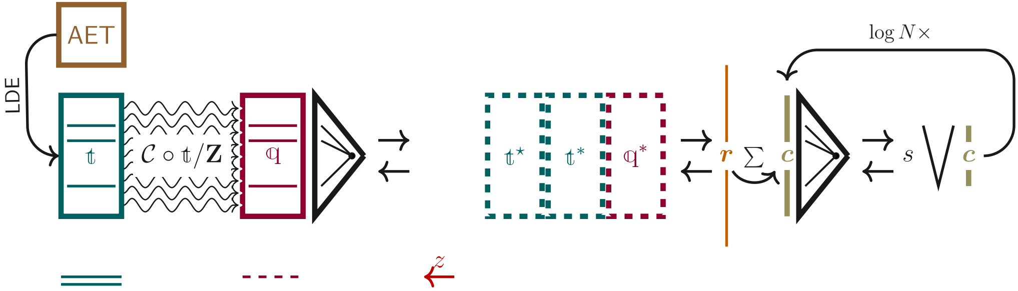


Table of Contents

Retrospective

Optimizations

- Batching
- Quotient Segmentation
- Grinding

Enhancements

- Zero-Knowledge
- Randomized AIR with Preprocessing

VM Architecture

- Overview
- Communication Arguments
- Memory

Other Topics

Table of Contents

- Retrospective
- Optimizations
  - Batching
  - Quotient Segmentation
  - Grinding
- Enhancements
  - Zero-Knowledge
  - Randomized AIR with Preprocessing
- VM Architecture
  - Overview
  - Communication Arguments
  - Memory
- Other Topics

Table of Contents

- Retrospective
- Optimizations
  - Batching
  - Quotient Segmentation
  - Grinding
- Enhancements
  - Zero-Knowledge
  - Randomized AIR with Preprocessing
- VM Architecture
  - Overview
  - Communication Arguments
  - Memory
- Other Topics

Batching

– linear $\sum_{i=0}^{n-1} r_i \boldsymbol{c}_i$ $\epsilon = \epsilon_{\mathsf{GAP}}(\delta_0)$

Batching

– linear	$\sum_{i=0}^{n-1} r_i \boldsymbol{c}_i$	$\epsilon = \epsilon_{\mathsf{GAP}}(\delta_0)$
– univariate	$\sum_{i=0}^{n-1} r^i \boldsymbol{c}_i$	$\epsilon = n \cdot \epsilon_{\mathsf{GAP}}(\delta_0)$

Batching

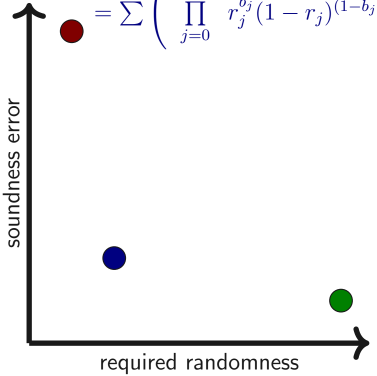
- linear $\sum_{i=0}^{n-1} r_i \boldsymbol{c}_i$ $\epsilon = \epsilon_{\mathsf{GAP}}(\delta_0)$
- univariate $\sum_{i=0}^{n-1} r^i \boldsymbol{c}_i$ $\epsilon = n \cdot \epsilon_{\mathsf{GAP}}(\delta_0)$
- multilinear $\sum_{i=0}^{n-1} \boldsymbol{r}^{\llcorner i \lrcorner} \boldsymbol{c}_i$ $\epsilon = \log n \cdot \epsilon_{\mathsf{GAP}}(\delta_0)$

$$= \sum \left(\prod_{j=0}^{\log n - 1} r_j^{b_j} (1 - r_j)^{(1-b_j)} \right) \boldsymbol{c}_i$$

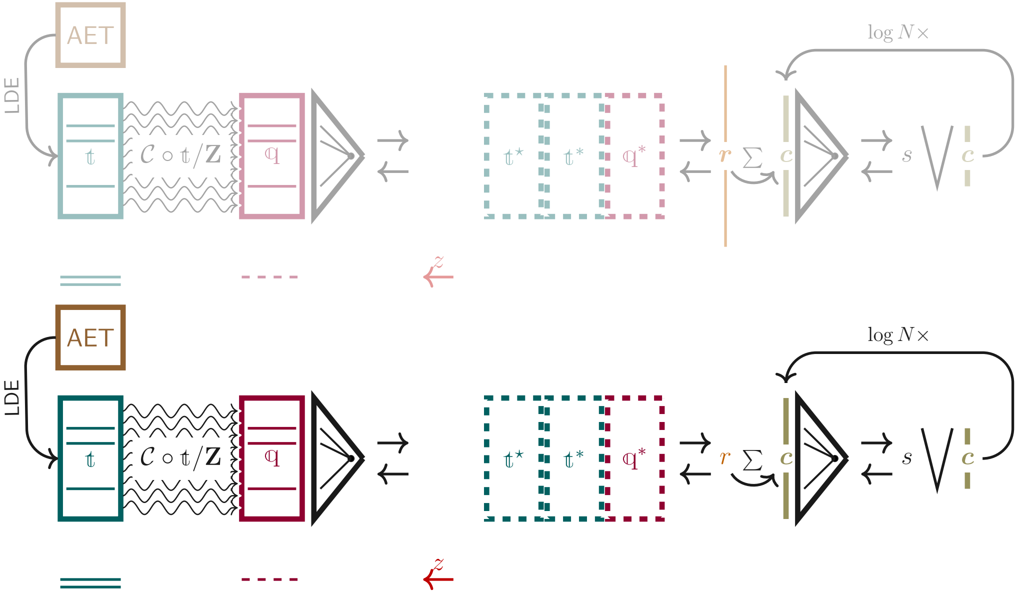
Batching

- linear $\sum_{i=0}^{n-1} r_i \boldsymbol{c}_i$ $\epsilon = \epsilon_{\mathsf{GAP}}(\delta_0)$
- univariate $\sum_{i=0}^{n-1} r^i \boldsymbol{c}_i$ $\epsilon = n \cdot \epsilon_{\mathsf{GAP}}(\delta_0)$
- multilinear $\sum_{i=0}^{n-1} \boldsymbol{r}^{\llcorner i \lrcorner} \boldsymbol{c}_i$ $\epsilon = \log n \cdot \epsilon_{\mathsf{GAP}}(\delta_0)$

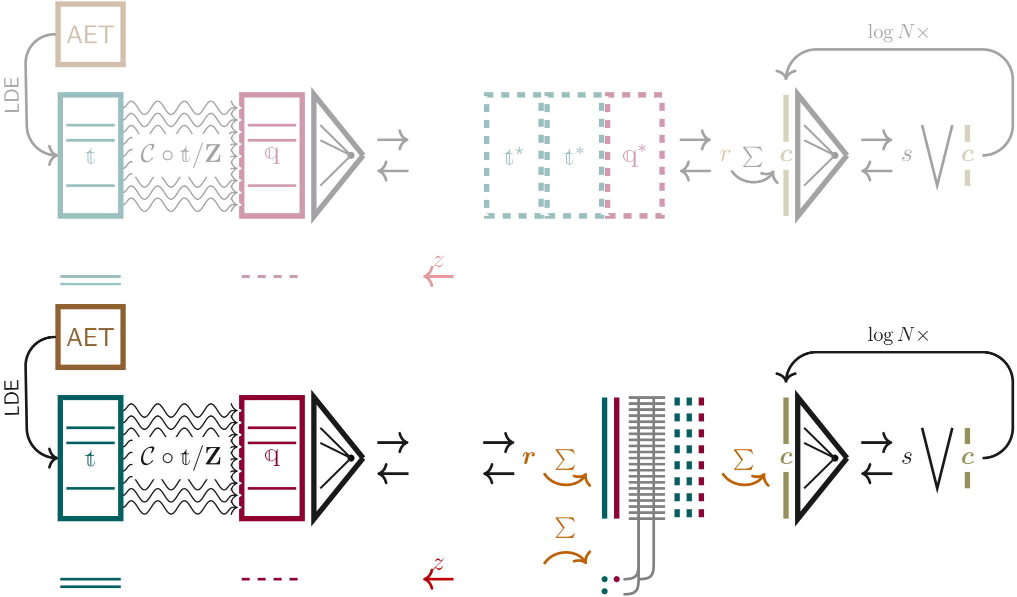
$$= \sum \left(\prod_{j=0}^{\log n - 1} r_j^{b_j} (1 - r_j)^{(1-b_j)} \right) \boldsymbol{c}_i$$



Univariate Batching



Batch Before DEEP



Batch Constraints

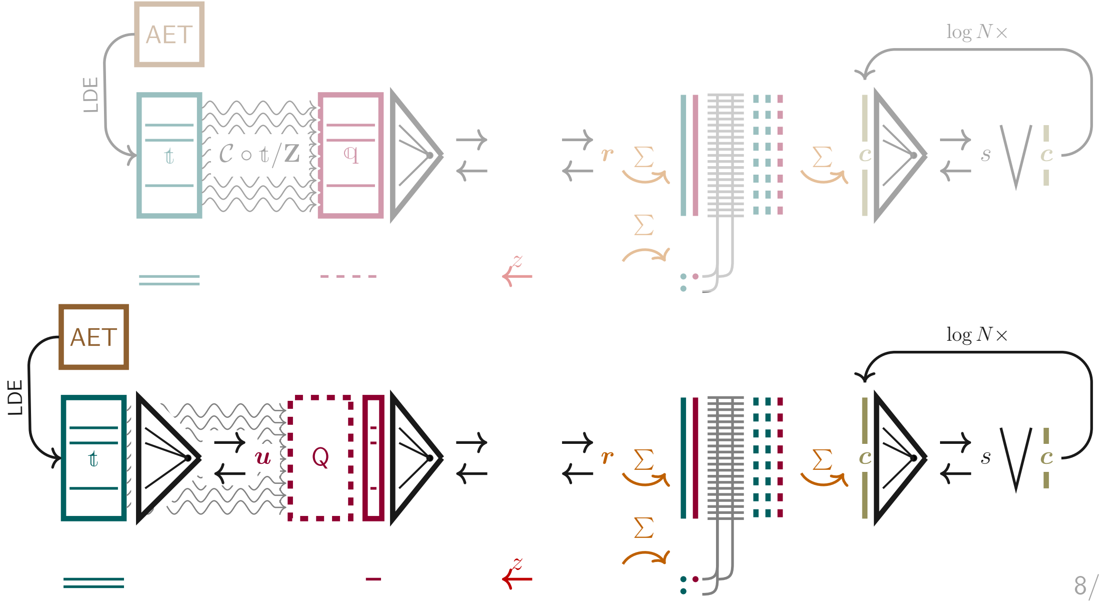


Table of Contents

- Retrospective
- Optimizations
  - Batching
  - Quotient Segmentation
  - Grinding
- Enhancements
  - Zero-Knowledge
  - Randomized AIR with Preprocessing
- VM Architecture
  - Overview
  - Communication Arguments
  - Memory
- Other Topics

Table of Contents

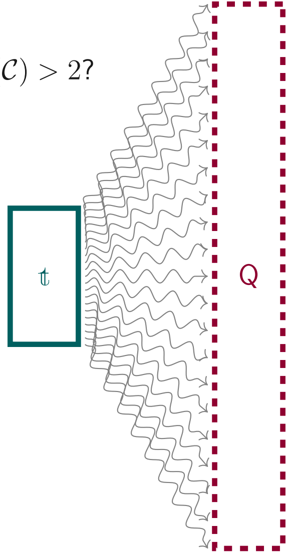
- Retrospective
- Optimizations
  - Batching
  - Quotient Segmentation
  - Grinding
- Enhancements
  - Zero-Knowledge
  - Randomized AIR with Preprocessing
- VM Architecture
  - Overview
  - Communication Arguments
  - Memory
- Other Topics

Quotient Segmentation

What if $\deg(\mathcal{C}) > 2$?

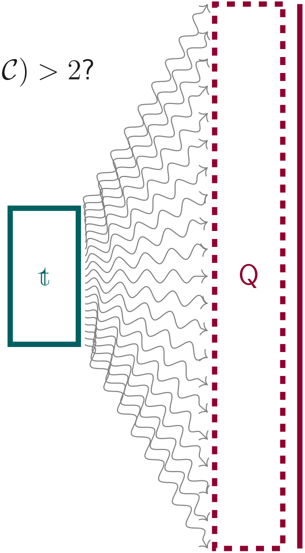
Quotient Segmentation

What if $\deg(\mathcal{C}) > 2$?



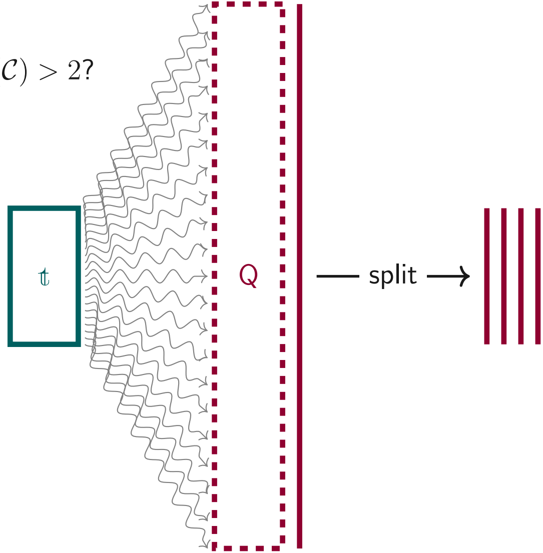
Quotient Segmentation

What if $\deg(\mathcal{C}) > 2$?



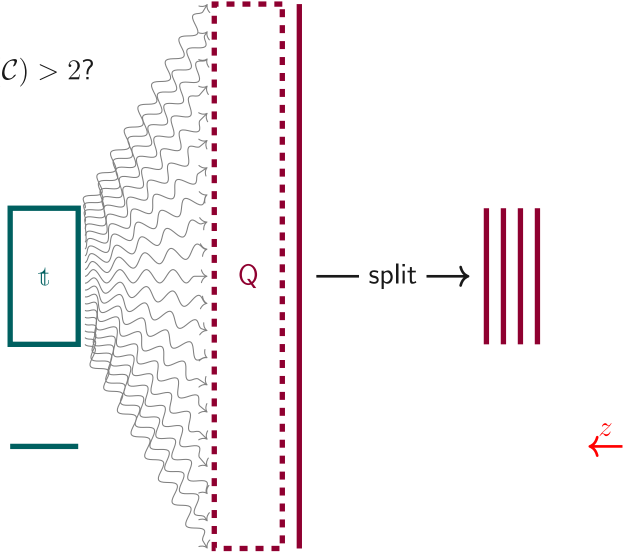
Quotient Segmentation

What if $\deg(\mathcal{C}) > 2$?



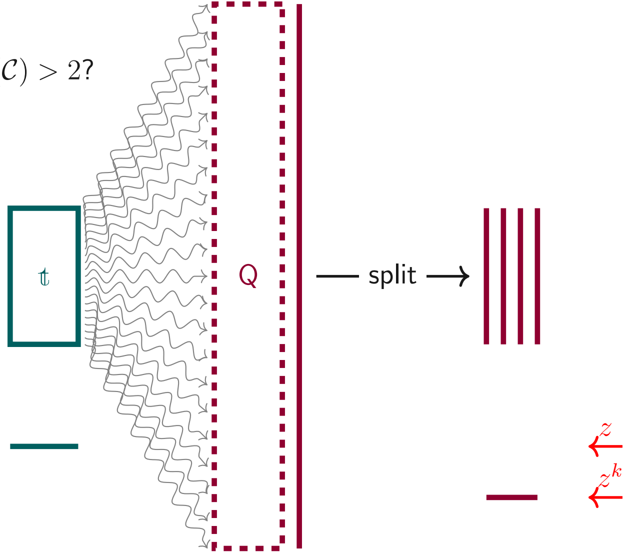
Quotient Segmentation

What if $\deg(\mathcal{C}) > 2$?

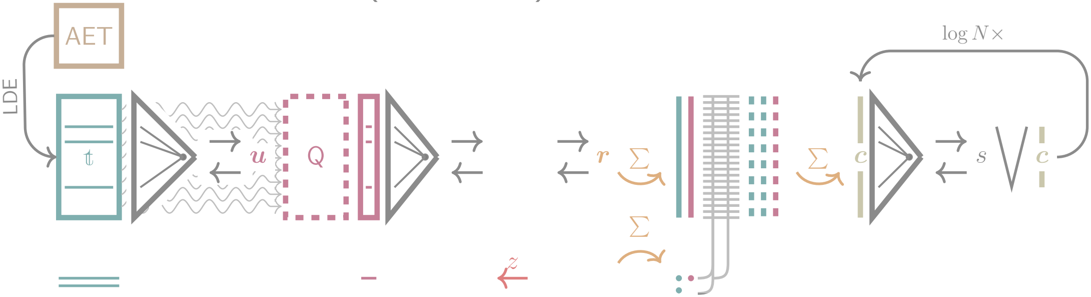


Quotient Segmentation

What if $\deg(\mathcal{C}) > 2$?



Quotient Segmentation (Diagram)



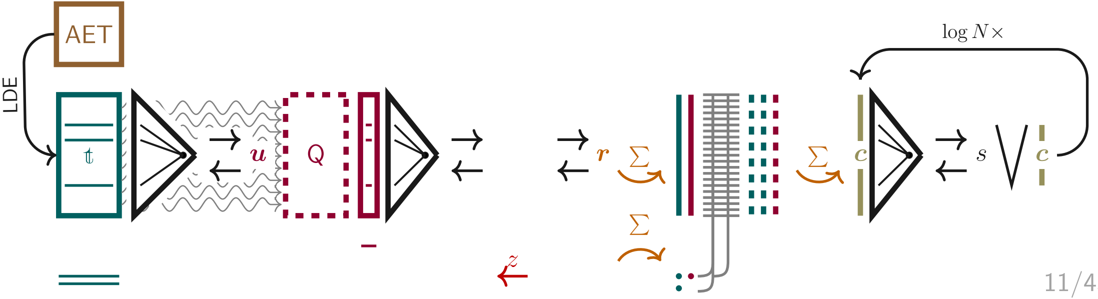


Table of Contents

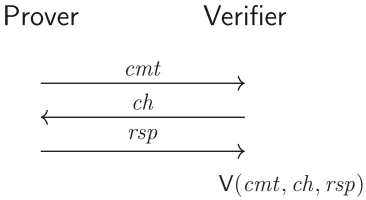
- Retrospective
- Optimizations
  - Batching
  - Quotient Segmentation
  - Grinding
- Enhancements
  - Zero-Knowledge
  - Randomized AIR with Preprocessing
- VM Architecture
  - Overview
  - Communication Arguments
  - Memory
- Other Topics

Table of Contents

- Retrospective
- Optimizations
  - Batching
  - Quotient Segmentation
  - Grinding
- Enhancements
  - Zero-Knowledge
  - Randomized AIR with Preprocessing
- VM Architecture
  - Overview
  - Communication Arguments
  - Memory
- Other Topics

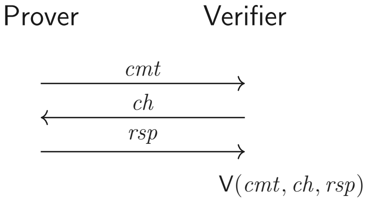
Grinding

No Grinding



Grinding

No Grinding



Prover

Verifier

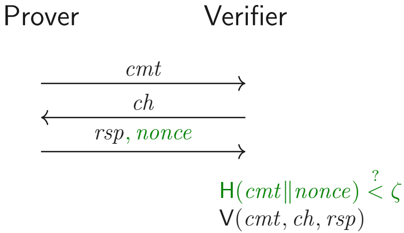
$\xrightarrow{cmt}$

$\xleftarrow{ch}$

$\xrightarrow{rsp}$

$\mathsf{V}(cmt, ch, rsp)$

With Grinding



Prover

Verifier

$\xrightarrow{cmt}$

$\xleftarrow{ch}$

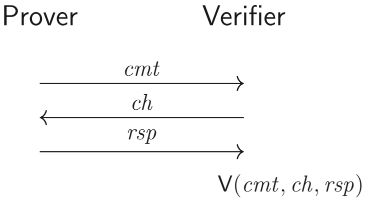
$\xrightarrow{rsp, nonce}$

$\mathsf{H}(cmt \| nonce) \overset{?}{<} \zeta$

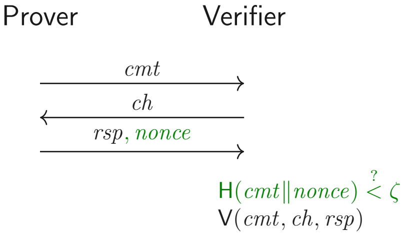
$\mathsf{V}(cmt, ch, rsp)$

Grinding

No Grinding

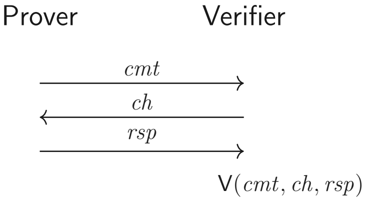
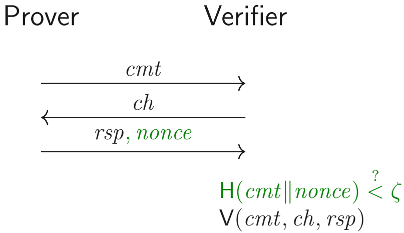


With Grinding


Prover
Verifier
cmt
ch
rsp, $nonce$
$\mathsf{H}(cmt \| nonce) \overset{?}{<} \zeta$
$\mathsf{V}(cmt, ch, rsp)$

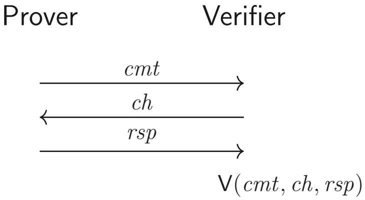
	No Grinding	With Grinding
Soundness Error	ϵ	$\epsilon \cdot \zeta$

Grinding

	No Grinding	With Grinding
	 Prover Verifier $\xrightarrow{cmt}$ $\xleftarrow{ch}$ $\xrightarrow{rsp}$ $\mathsf{V}(cmt, ch, rsp)$	 Prover Verifier $\xrightarrow{cmt}$ $\xleftarrow{ch}$ $\xrightarrow{rsp,\ nonce}$ $\mathsf{H}(cmt \| nonce) \overset{?}{<} \zeta$ $\mathsf{V}(cmt, ch, rsp)$
Soundness Error	ϵ	$\epsilon \cdot \zeta$
Prover Work	W	$W + \zeta^{-1}$

Grinding

No Grinding



Prover   Verifier

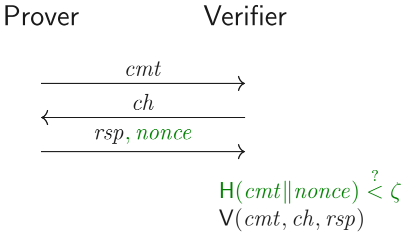
$\xrightarrow{cmt}$

$\xleftarrow{ch}$

$\xrightarrow{rsp}$

$\mathsf{V}(cmt, ch, rsp)$

With Grinding



Prover   Verifier

$\xrightarrow{cmt}$

$\xleftarrow{ch}$

$\xrightarrow{rsp, nonce}$

$\mathsf{H}(cmt \| nonce) \overset{?}{<} \zeta$

$\mathsf{V}(cmt, ch, rsp)$

	No Grinding		With Grinding	
Soundness Error	ϵ		$\epsilon \cdot \zeta$	+ 10 bits
Prover Work	W	$\approx 2^{30}$	$W + \zeta^{-1}$	+ 0.1%

Table of Contents

- Retrospective
- Optimizations
  - Batching
  - Quotient Segmentation
  - Grinding
- Enhancements
  - Zero-Knowledge
  - Randomized AIR with Preprocessing
- VM Architecture
  - Overview
  - Communication Arguments
  - Memory
- Other Topics

Table of Contents

Retrospective

Optimizations

- Batching
- Quotient Segmentation
- Grinding

Enhancements

- Zero-Knowledge
- Randomized AIR with Preprocessing

VM Architecture

- Overview
- Communication Arguments
- Memory

Other Topics

Table of Contents

Retrospective

Optimizations
- Batching
- Quotient Segmentation
- Grinding

Enhancements
- Zero-Knowledge
- Randomized AIR with Preprocessing

VM Architecture
- Overview
- Communication Arguments
- Memory

Other Topics

Table of Contents

Retrospective

Optimizations

- Batching
- Quotient Segmentation
- Grinding

Enhancements

- Zero-Knowledge
- Randomized AIR with Preprocessing

VM Architecture

- Overview
- Communication Arguments
- Memory

Other Topics

Zero Knowledge

zero-knowledge $\Leftrightarrow$ transcript is independent of witness

Zero Knowledge

zero-knowledge $\Leftrightarrow$ transcript is independent of witness

$\rightarrow$ *mask with randomness*

Zero Knowledge

zero-knowledge ⇔ transcript is independent of witness

→ *mask with randomness*

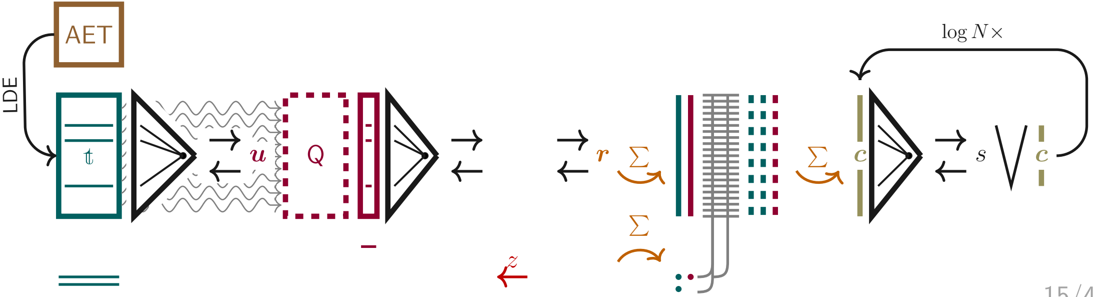
1. salted Merkle leafs (optional)
2. batch randomizer polynomial
3. trace randomizer values

Zero Knowledge

zero-knowledge ⇔ transcript is independent of witness

→ *mask with randomness*

1. salted Merkle leafs (optional)
2. batch randomizer polynomial
3. trace randomizer values

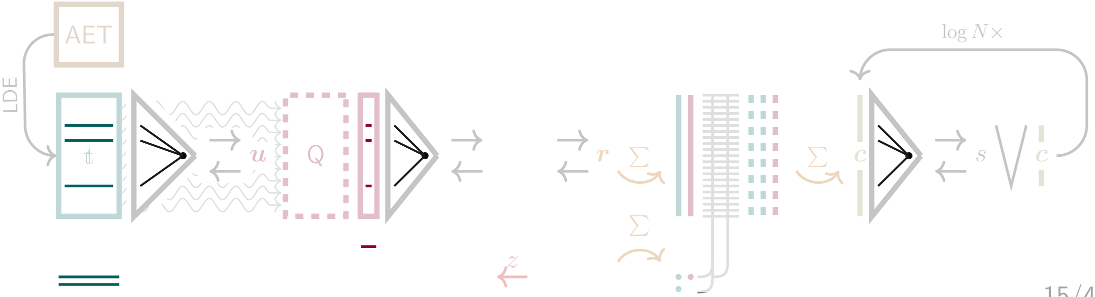


Zero Knowledge

zero-knowledge ⇔ transcript is independent of witness

→ *mask with randomness*

1. salted Merkle leafs (optional)
2. batch randomizer polynomial
3. trace randomizer values

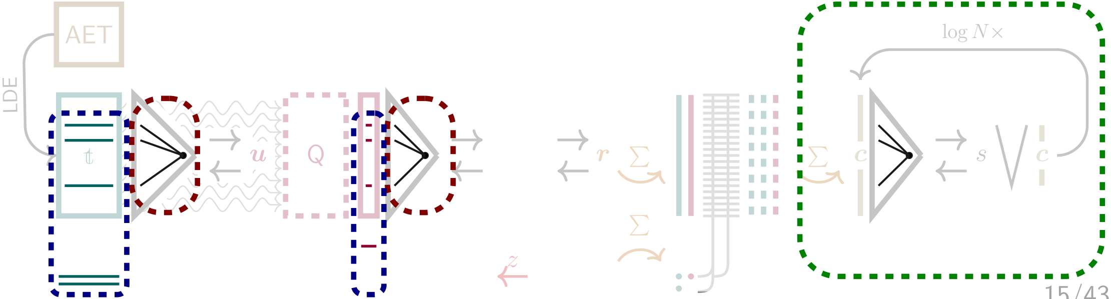


Zero Knowledge

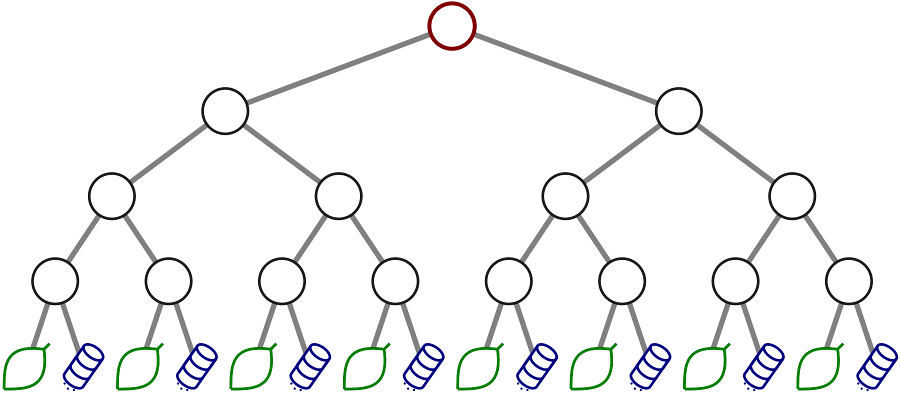
zero-knowledge ⇔ transcript is independent of witness

→ *mask with randomness*

1. salted Merkle leafs (optional)
2. batch randomizer polynomial
3. trace randomizer values



Salted Merkle Leafs



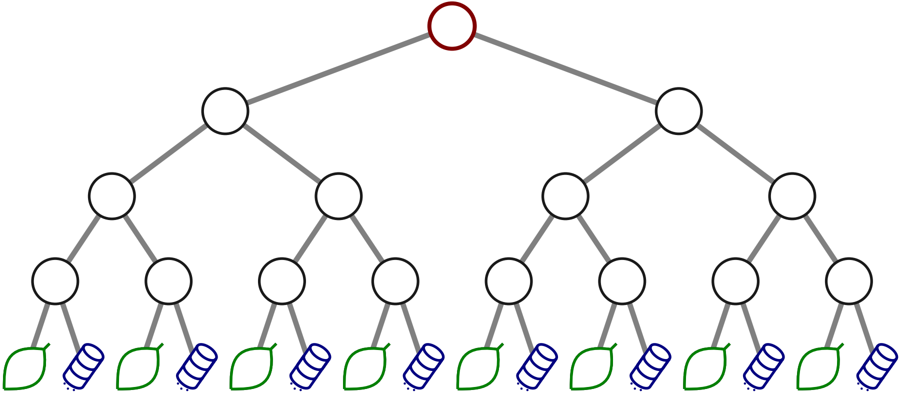
$$parent = \mathsf{H}(left\,child \| right\,child)$$

🍃 leafs

○ root

🧂 salts

Salted Merkle Leafs

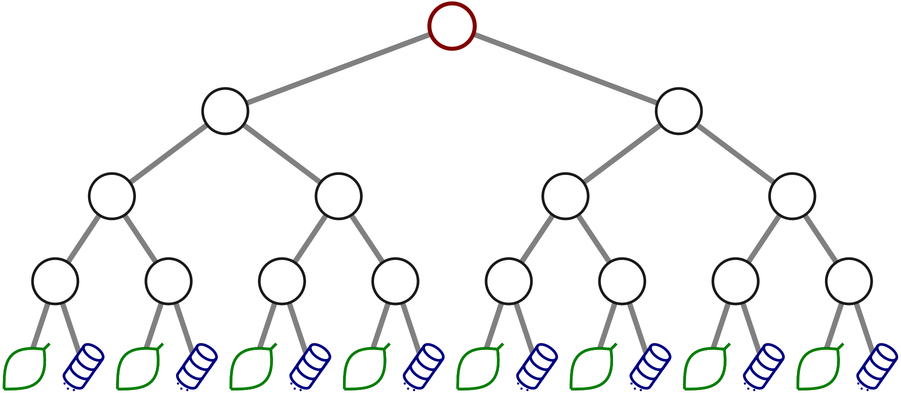


$$\mathit{parent} = \mathsf{H}(\mathit{left\,child} \| \mathit{right\,child})$$

- 🍃 leafs
- ○ root
- 🧂 salts

Motivation: make internal Merkle nodes ○ independent of un-opened data

Salted Merkle Leafs



$$parent = \mathsf{H}(left\,child \| right\,child)$$

leafs

root

salts

Motivation: make internal Merkle nodes ○ independent of un-opened data

→ in ROM: data = H(leaf) already independent

→ in standard model: concat-then-hash not enough ×

⟶ use perfectly-hiding + computationally-binding commitment scheme instead ✓

Batch Randomizer Polynomial

include uniformly random polynomial into batch

Batch Randomizer Polynomial

include uniformly random polynomial into batch

→ How: add *unconstrained* and *random* trace column

Batch Randomizer Polynomial

include uniformly random polynomial into batch

→ How: add *unconstrained* and *random* trace column

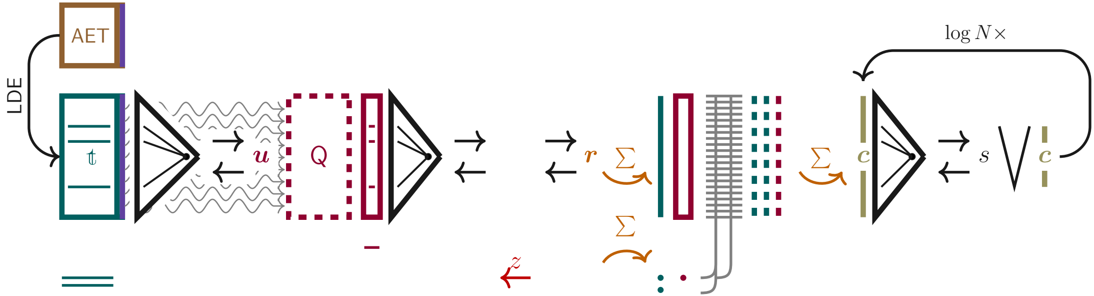
→ Why: make low-degree tested codeword independent of trace

Batch Randomizer Polynomial

include uniformly random polynomial into batch

→ How: add *unconstrained* and *random* trace column

→ Why: make low-degree tested codeword independent of trace



Trace Randomizer Values: Interleaving

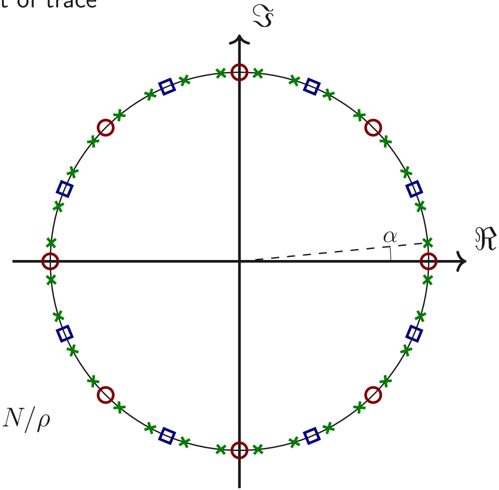
interleave trace with random rows

→ Why: make observed rows independent of trace

Trace Randomizer Values: Interleaving

interleave trace with random rows

→ Why: make observed rows independent of trace



○ trace values

□ randomizer values

× evaluation domain

$=$ coset of subgroup of order $2N/\rho$

code *rate* $\rho = \frac{\#\bigcirc + \#\square}{\#\times}$ (!!!)

Trace Randomizer Values: Interleaving

interleave trace with random rows

→ Why: make observed rows independent of trace

$$\mathsf{w} \cdot \#\square \geqslant \underbrace{t \cdot \mathsf{w}}_{\text{FRI}} + (\underbrace{2 \cdot \mathsf{w}}_{\text{DEEP}} + \underbrace{(t+1) \cdot k}_{\text{quotient segments}}) \cdot \underbrace{e}_{\text{extension degree}}$$

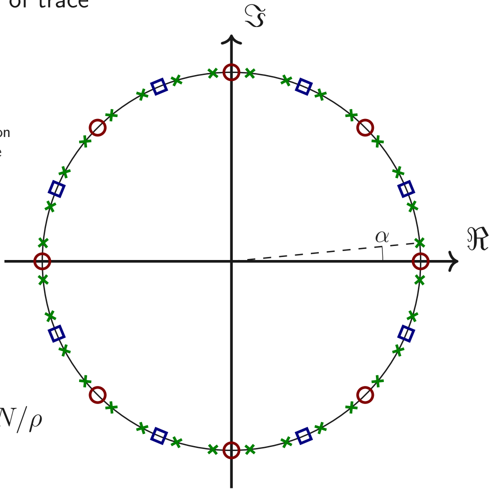
○ trace values

□ randomizer values

× evaluation domain

$=$ coset of subgroup of order $2N/\rho$

code *rate* $\rho = \frac{\#\bigcirc + \#\square}{\#\times}$ (!!!)



Trace Randomizer Values: Stingy

pad then concatenate random rows

→ Why: make observed rows independent of trace

$$\mathsf{w} \cdot \#\square \geqslant \underbrace{t \cdot \mathsf{w}}_{\text{FRI}} + (\underbrace{2 \cdot \mathsf{w}}_{\text{DEEP}} + \underbrace{(t+1) \cdot k}_{\substack{\text{quotient}\\\text{segments}}}) \cdot \underbrace{e}_{\substack{\text{extension}\\\text{degree}}}$$

Trace Randomizer Values: Stingy

pad then concatenate random rows

→ Why: make observed rows independent of trace

$$\mathsf{w} \cdot \#\square \geqslant \underbrace{t \cdot \mathsf{w}}_{\text{FRI}} + \big(\underbrace{2 \cdot \mathsf{w}}_{\text{DEEP}} + \underbrace{(t+1) \cdot k}_{\text{quotient segments}} \big) \cdot \underbrace{e}_{\text{extension degree}}$$

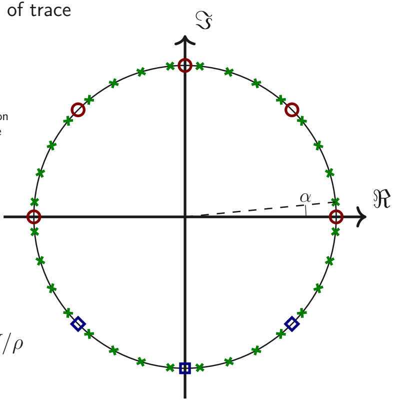
○ trace values

□ randomizer values

× evaluation domain

= coset of subgroup of order N/ρ

code *rate* $\rho = \frac{\#\bigcirc + \#\square}{\#\times}$



Trace Randomizer Values: Stingy

pad then concatenate random rows

→ Why: make observed rows independent of trace

$$\mathsf{w} \cdot \#\square \geqslant \underbrace{t \cdot \mathsf{w}}_{\text{FRI}} + (\underbrace{2 \cdot \mathsf{w}}_{\text{DEEP}} + \underbrace{(t+1) \cdot k}_{\text{quotient segments}}) \cdot \underbrace{e}_{\text{extension degree}}$$

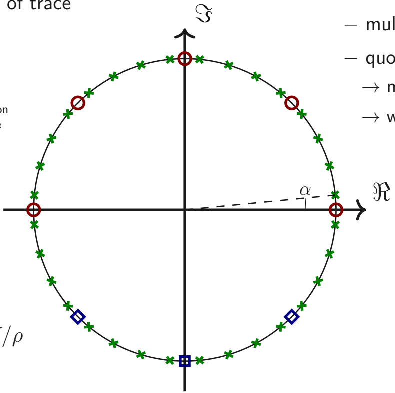
○ trace values

□ randomizer values

× evaluation domain

= coset of subgroup of order N/ρ

code *rate* $\rho = \frac{\#\bigcirc + \#\square}{\#\times}$



complications:

- multiply quotients by $\deg \#\square - 1$ $Z(X)$
- quotient degree increases
  - → more segments, or
  - → worse rate ⇒ more indices

Table of Contents

Retrospective

Optimizations

- Batching
- Quotient Segmentation
- Grinding

Enhancements

- Zero-Knowledge
- Randomized AIR with Preprocessing

VM Architecture

- Overview
- Communication Arguments
- Memory

Other Topics

Table of Contents

Retrospective

Optimizations
- Batching
- Quotient Segmentation
- Grinding

Enhancements
- Zero-Knowledge
- Randomized AIR with Preprocessing

VM Architecture
- Overview
- Communication Arguments
- Memory

Other Topics

Two-Stage DEEP-ALI

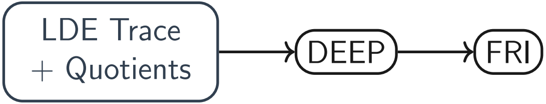
0-stage:

1-stage:

2-stage:

Two-Stage DEEP-ALI

0-stage:



LDE Trace
+ Quotients

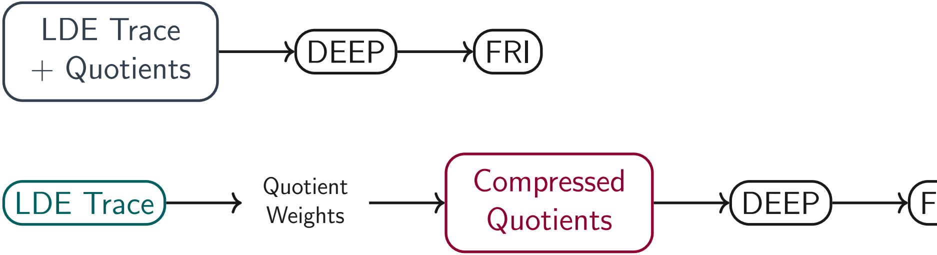
DEEP

FRI

1-stage:

2-stage:

Two-Stage DEEP-ALI



0-stage:

LDE Trace + Quotients

DEEP

FRI

1-stage:

LDE Trace

Quotient Weights

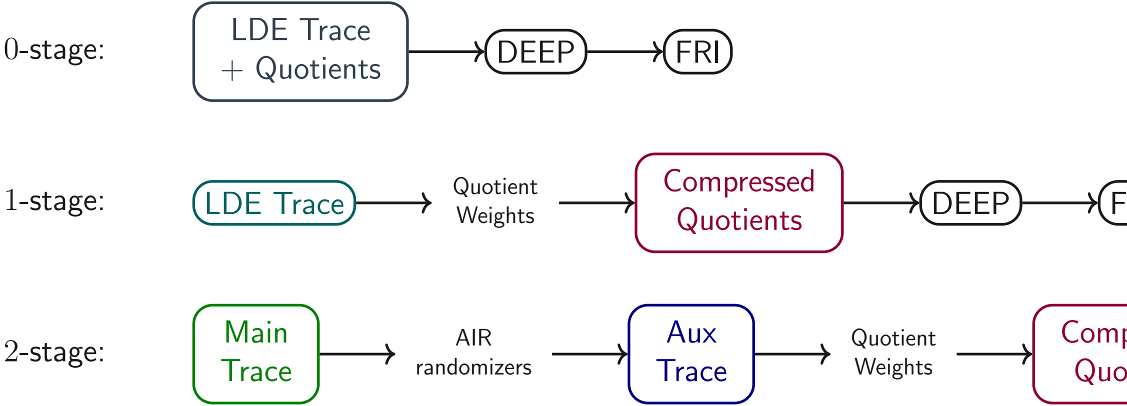
Compressed Quotients

DEEP

F

2-stage:

Two-Stage DEEP-ALI



0-stage:

LDE Trace + Quotients → DEEP → FRI

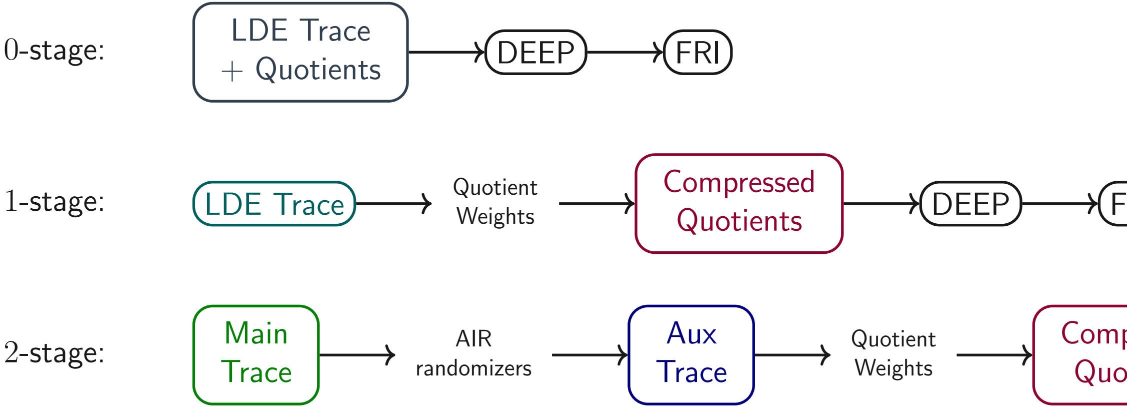
1-stage:

LDE Trace → Quotient Weights → Compressed Quotients → DEEP → F

2-stage:

Main Trace → AIR randomizers → Aux Trace → Quotient Weights → Comp Quo

Two-Stage DEEP-ALI



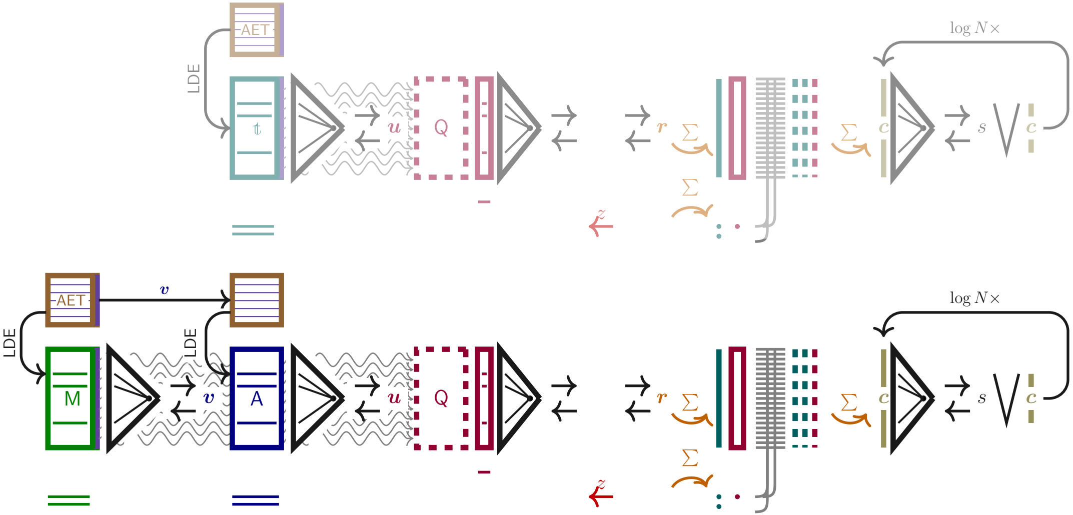
0-stage: LDE Trace + Quotients → DEEP → FRI

1-stage: LDE Trace → Quotient Weights → Compressed Quotients → DEEP → F

2-stage: Main Trace → AIR randomizers → Aux Trace → Quotient Weights → Comp Quo

randomized AIR $\gg$ *deterministic AIR*

Two-Stage DEEP-ALI (Diagram)



Preprocessing

pre-commit to separate "trace" table

- look-up tables ✓
- circuits ✓
- extra Merkle tree x
- need to know trace length beforehand x

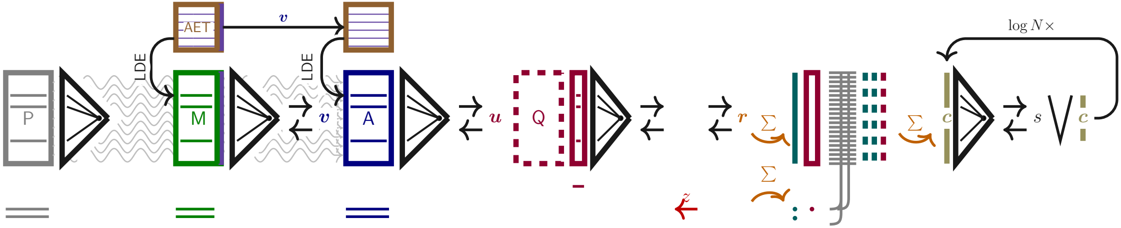


Table Of Contents

- Retrospective
- Optimizations
  - Batching
  - Quotient Segmentation
  - Grinding
- Enhancements
  - Zero-Knowledge
  - Randomized AIR with Preprocessing
- VM Architecture
  - Overview
  - Communication Arguments
  - Memory
- Other Topics

Table Of Contents

- Retrospective
- Optimizations
  - Batching
  - Quotient Segmentation
  - Grinding
- Enhancements
  - Zero-Knowledge
  - Randomized AIR with Preprocessing
- VM Architecture
  - Overview
  - Communication Arguments
  - Memory
- Other Topics

Table Of Contents

- Retrospective
- Optimizations
  - Batching
  - Quotient Segmentation
  - Grinding
- Enhancements
  - Zero-Knowledge
  - Randomized AIR with Preprocessing
- VM Architecture
  - Overview
  - Communication Arguments
  - Memory
- Other Topics

Table Of Contents

Retrospective

Optimizations
- Batching
- Quotient Segmentation
- Grinding

Enhancements
- Zero-Knowledge
- Randomized AIR with Preprocessing

VM Architecture
- Overview
- Communication Arguments
- Memory

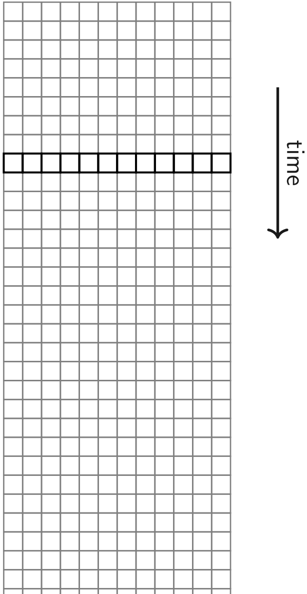
Other Topics

Processor (Example)

`clk`	*clock / cycle counter*
`ip`	*instruction pointer*
`ci`	*current instruction*
$\texttt{arg}_0$	*instruction argument 0*
$\texttt{arg}_1$	*instruction argument 1*
$\texttt{arg}_2$	*instruction argument 2*
`ramp`	*RAM pointer*
`ramv`	*RAM value*
$\texttt{reg}_0$	*register 0*
$\texttt{reg}_1$	*register 1*
$\texttt{reg}_2$	*register 2*
$\texttt{reg}_3$	*register 3*

Processor (Example)

`clk`	*clock / cycle counter*
`ip`	*instruction pointer*
`ci`	*current instruction*
$\mathtt{arg}_0$	*instruction argument 0*
$\mathtt{arg}_1$	*instruction argument 1*
$\mathtt{arg}_2$	*instruction argument 2*
`ramp`	*RAM pointer*
`ramv`	*RAM value*
$\mathtt{reg}_0$	*register 0*
$\mathtt{reg}_1$	*register 1*
$\mathtt{reg}_2$	*register 2*
$\mathtt{reg}_3$	*register 3*



Processor AIR Constraints (Example)

$\mathbf{jmp}\ a$ *jump to instruction* $\texttt{reg}_a$

$$\mathsf{dest} = \sum_{i=0}^{3} \mathsf{reg}_i \prod_{j \neq i} \frac{\mathsf{arg}_0 - j}{i - j}$$ *value of* $\texttt{reg}_a$

$$\mathsf{jump} = \mathsf{dest} - \mathsf{ip}^\star$$ *update value of* `ip` *(jump case)*

$$\mathsf{nojump} = \mathsf{ip} + 1 - \mathsf{ip}^\star$$ *update value of* `ip` *(no jump)*

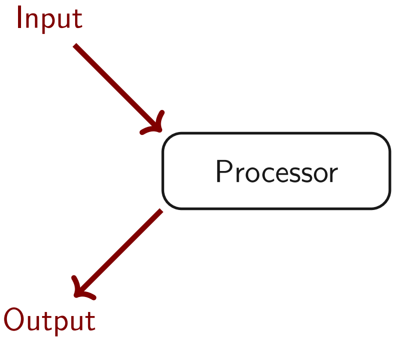
$$\mathsf{selector} = \prod_{instr \in \mathcal{I} \setminus \{\mathbf{jmp}\}} \frac{instr - \mathsf{ci}}{\mathsf{instr} - \mathbf{jmp}}$$ *1 iff* $\texttt{ci} = \mathbf{jmp}$

$$\boxed{\mathsf{selector} \cdot \mathsf{jump} + (1 - \mathsf{selector}) \cdot \mathsf{nojump}}$$

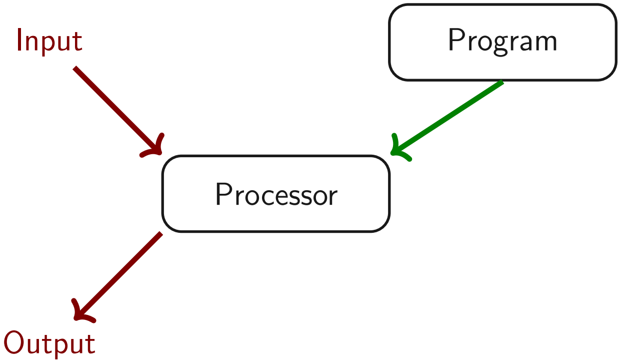
Communication Lines

Processor

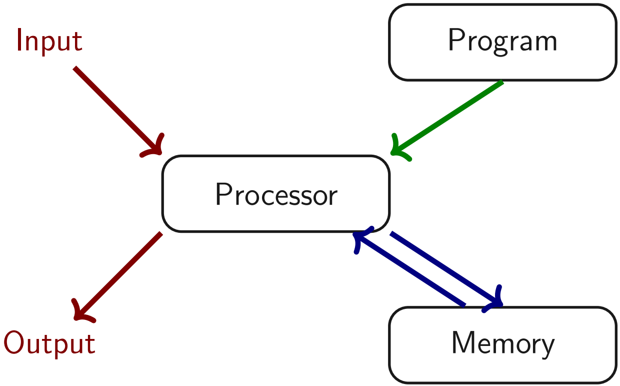
Communication Lines



Communication Lines



Communication Lines



Communication Lines

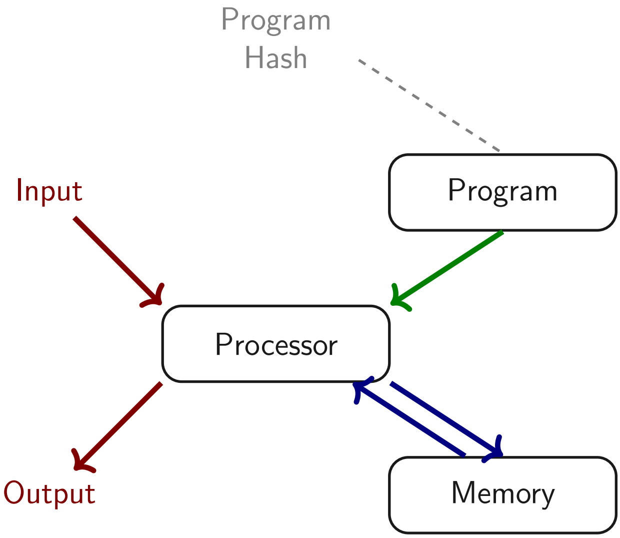


Table of Contents

Retrospective

Optimizations

- Batching
- Quotient Segmentation
- Grinding

Enhancements

- Zero-Knowledge
- Randomized AIR with Preprocessing

VM Architecture

- Overview
- Communication Arguments
- Memory

Other Topics

Table of Contents

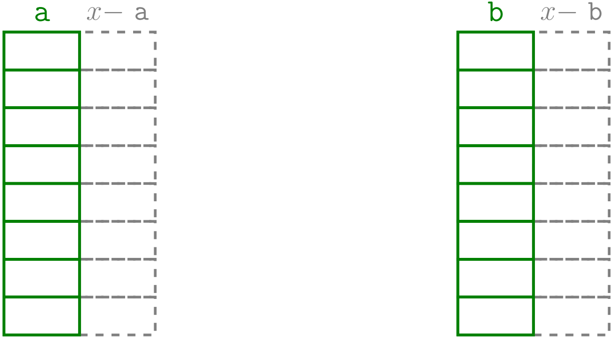
- Retrospective
- Optimizations
  - Batching
  - Quotient Segmentation
  - Grinding
- Enhancements
  - Zero-Knowledge
  - Randomized AIR with Preprocessing
- VM Architecture
  - Overview
  - Communication Arguments
  - Memory
- Other Topics

Permutation Argument

$b = \sigma(a)$ for some permutation σ

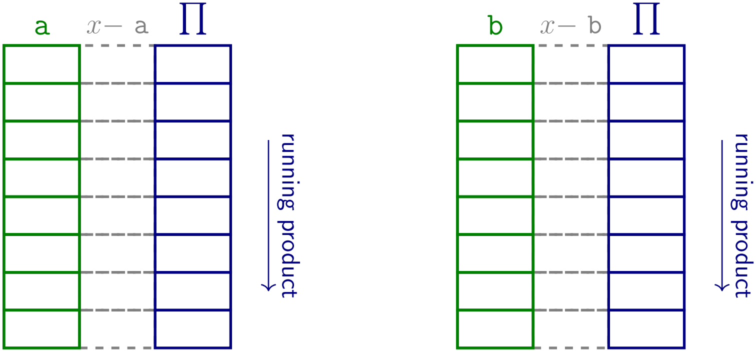
Permutation Argument

$b = \sigma(a)$ for some permutation σ



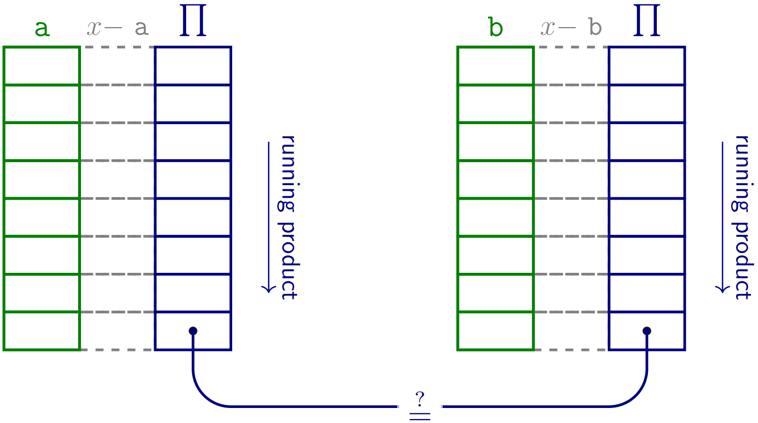
Permutation Argument

$b = \sigma(a)$ for some permutation σ



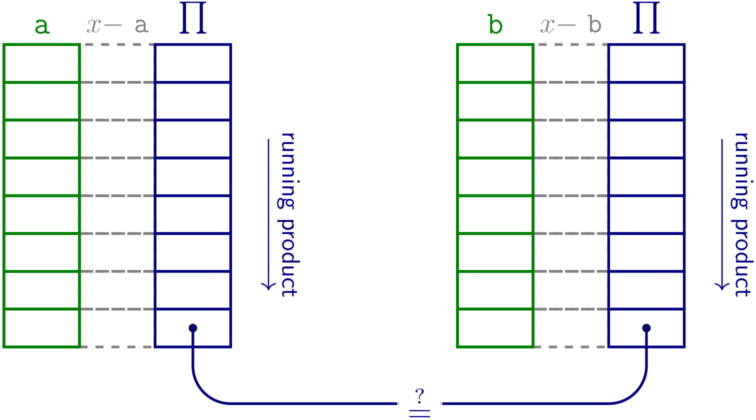
Permutation Argument

$b = \sigma(a)$ for some permutation σ



Permutation Argument

$b = \sigma(a)$ for some permutation σ



Soundness

$$\Pr_x[p_a(x) = p_b(x) \mid p_a(X) \neq p_b(X)] \leqslant \frac{N}{\mathbb{F}}$$

Evaluation Argument

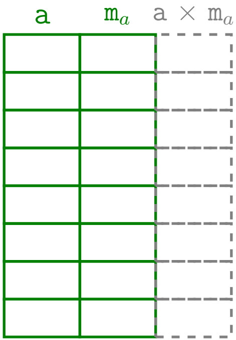
$$a[m_a] = b[m_b] \text{ for } m_a, m_b \subseteq \{0, \ldots, N-1\}$$

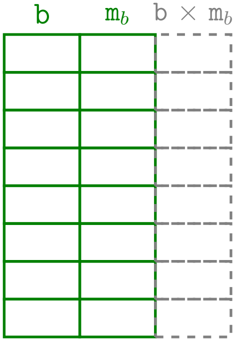
a	$\mathtt{m}_a$

b	$\mathtt{m}_b$

Evaluation Argument

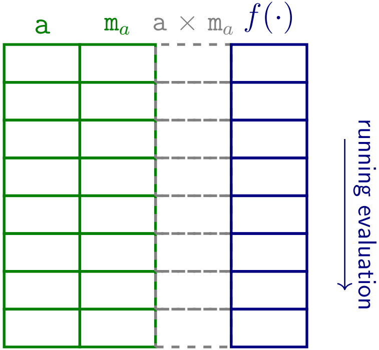
$$a[m_a] = b[m_b] \text{ for } m_a, m_b \subseteq \{0, \ldots, N-1\}$$

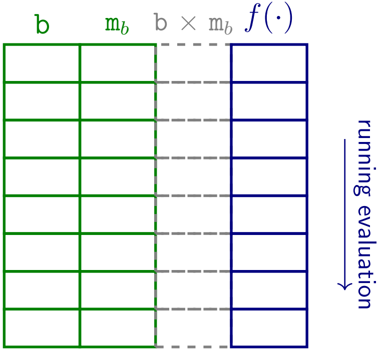




Evaluation Argument

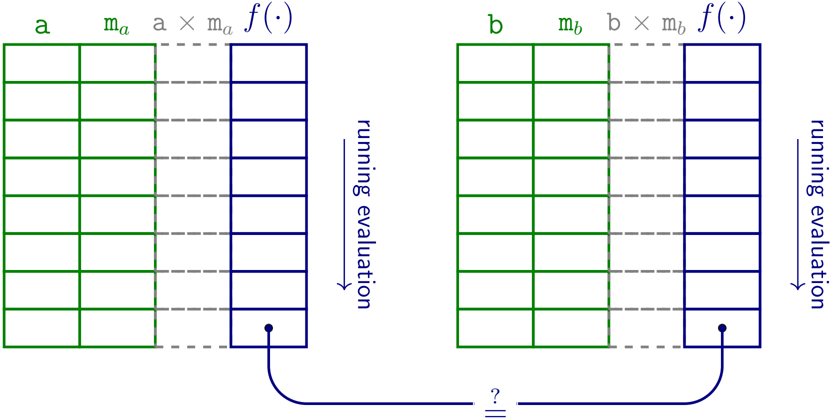
$$a[m_a] = b[m_b] \text{ for } m_a, m_b \subseteq \{0, \ldots, N-1\}$$





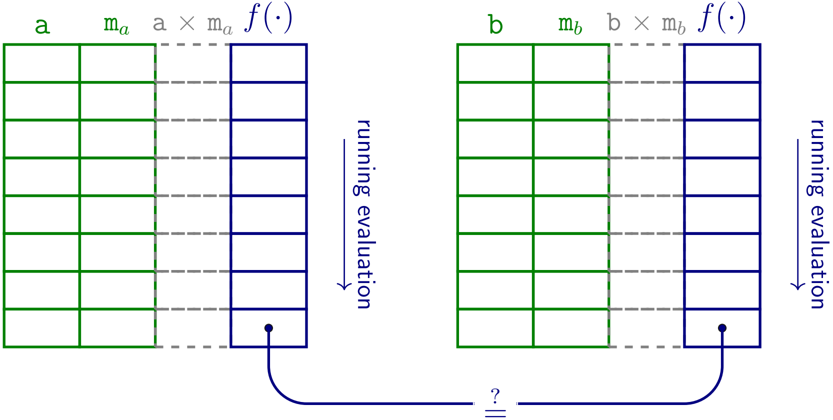
Evaluation Argument

$$a[m_a] = b[m_b] \text{ for } m_a, m_b \subseteq \{0, \ldots, N-1\}$$



Evaluation Argument

$$a[m_a] = b[m_b] \text{ for } m_a, m_b \subseteq \{0, \ldots, N-1\}$$



Soundness

$$\Pr_x[f_a(x) = f_b(x) \mid f_a(X) \neq f_b(X)] \leqslant \frac{N}{\mathbb{F}}$$

Lookup Argument

$a \equiv b$ as sets

a	$\mathtt{m}_a$

b	$\mathtt{m}_b$

Lookup Argument

$a \equiv b$ as sets

a	$\mathtt{m}_a$

b	$\mathtt{m}_b$

$$\log \frac{\mathsf{d}}{\mathsf{d}X}[f(X)] = \frac{f'(X)}{f(X)}$$

Lookup Argument

$a \equiv b$ as sets

a	$\mathtt{m}_a$

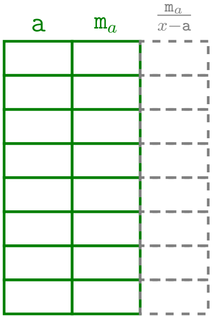
b	$\mathtt{m}_b$

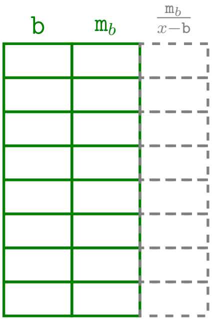
$$\log \frac{\mathsf{d}}{\mathsf{d}X}[f(X)] = \frac{f'(X)}{f(X)}$$

$$\log \frac{\mathsf{d}}{\mathsf{d}X}\left[\prod_i (X - a_i)^{m_i}\right]$$
$$= \sum_i \frac{m_i}{X - a_i}$$

Lookup Argument

$a \equiv b$ as sets



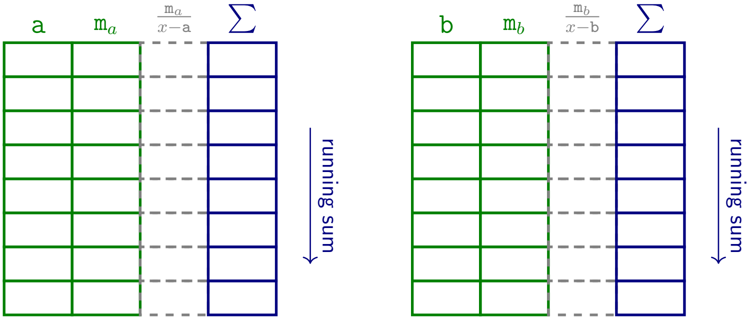


$$\log \tfrac{\mathsf{d}}{\mathsf{d}X}[f(X)] = \tfrac{f'(X)}{f(X)}$$

$$\log \tfrac{\mathsf{d}}{\mathsf{d}X}\left[\textstyle\prod_i (X - a_i)^{m_i}\right]$$
$$= \textstyle\sum_i \tfrac{m_i}{X - a_i}$$

Lookup Argument

$a \equiv b$ as sets

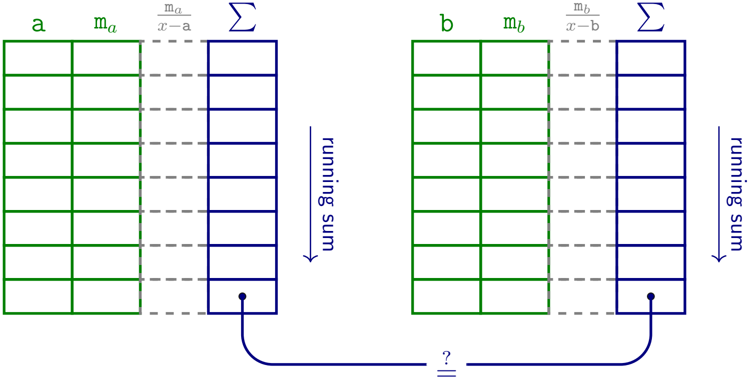


$$\log \frac{\mathsf{d}}{\mathsf{d}X}[f(X)] = \frac{f'(X)}{f(X)}$$

$$\log \frac{\mathsf{d}}{\mathsf{d}X}\left[\prod_i (X - a_i)^{m_i}\right]$$
$$= \sum_i \frac{m_i}{X - a_i}$$

Lookup Argument

$a \equiv b$ as sets

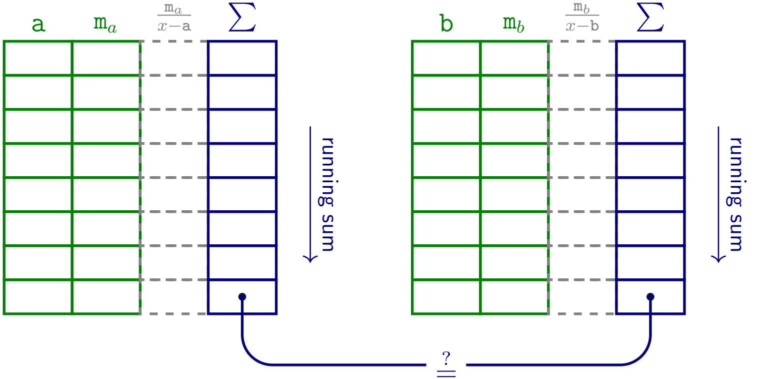


$$\log \frac{\mathsf{d}}{\mathsf{d}X}[f(X)] = \frac{f'(X)}{f(X)}$$

$$\log \frac{\mathsf{d}}{\mathsf{d}X}\left[\prod_i (X - a_i)^{m_i}\right]$$
$$= \sum_i \frac{m_i}{X-a_i}$$

Lookup Argument

$a \equiv b$ as sets



$$\log \frac{\mathsf{d}}{\mathsf{d}X}[f(X)] = \frac{f'(X)}{f(X)}$$

$$\log \frac{\mathsf{d}}{\mathsf{d}X}\left[\prod_i (X - a_i)^{m_i}\right]$$
$$= \sum_i \frac{m_i}{X-a_i}$$

Soundness

$$\Pr_x[S_a = S_b \mid a \not\equiv b]$$

$$= \Pr_x[S_a \cdot \prod_i (x - \mathtt{a}_i)^{\mathtt{m}_{a,i}} \cdot \prod_i (x - \mathtt{b}_i)^{\mathtt{m}_{b,i}} = S_b \cdot \prod_i (x - \mathtt{a}_i)^{\mathtt{m}_{a,i}} \cdot \prod_i (x - \mathtt{b}_i)^{\mathtt{m}_{b,i}} \mid a \not\equiv b]$$

$$\leqslant \frac{2N}{\mathbb{F}}$$

Table of Contents

- Retrospective
- Optimizations
  - Batching
  - Quotient Segmentation
  - Grinding
- Enhancements
  - Zero-Knowledge
  - Randomized AIR with Preprocessing
- VM Architecture
  - Overview
  - Communication Arguments
  - Memory
- Other Topics

Table of Contents

Retrospective

Optimizations

- Batching
- Quotient Segmentation
- Grinding

Enhancements

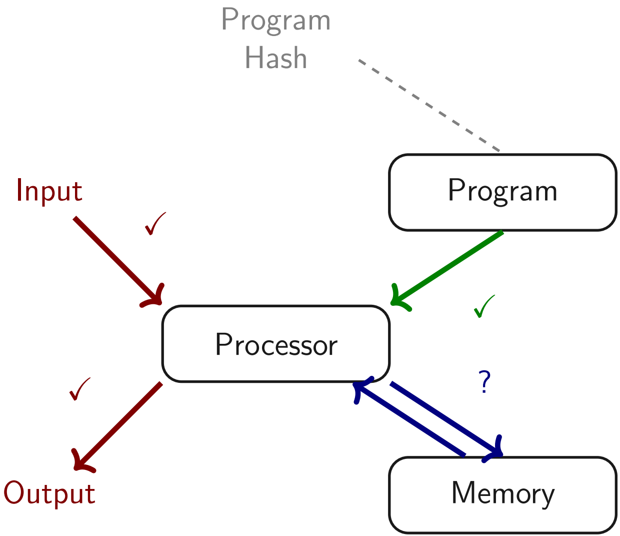
- Zero-Knowledge
- Randomized AIR with Preprocessing

VM Architecture

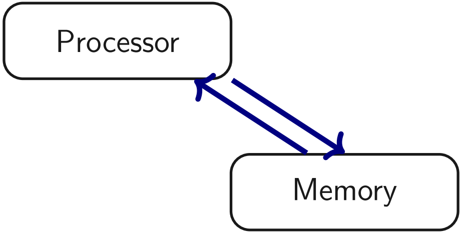
- Overview
- Communication Arguments
- Memory

Other Topics

Communication Lines Again



Memory — Problem Statement



Memory cells must have the same value as the previous time they were touched.

✓ random access
✓ read-write

Memory — Construction

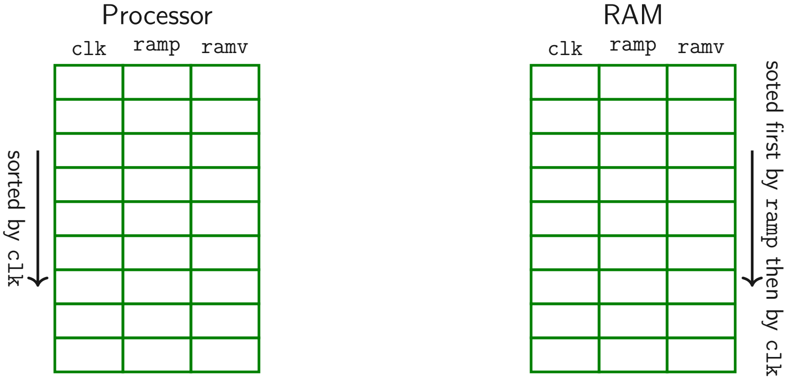
Processor

clk	ramp	ramv

RAM

clk	ramp	ramv

Memory — Construction



Processor

`clk` `ramp` `ramv`

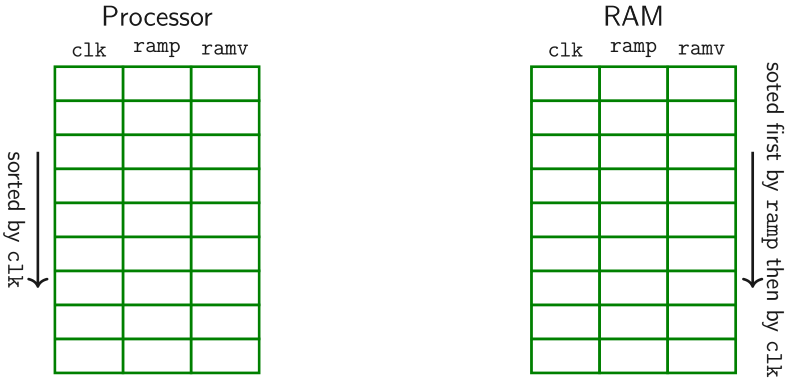
sorted by `clk`

RAM

`clk` `ramp` `ramv`

soted first by `ramp` then by `clk`

Memory — Construction



Processor

clk ramp ramv

sorted by clk

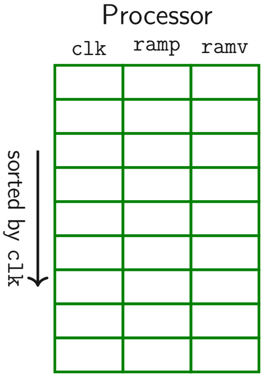
RAM

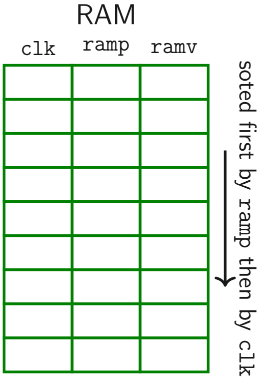
clk ramp ramv

soted first by ramp then by clk

1. same data, different order
2. within regions of constant ramp, correctly sorted by clk
3. correctly sorted by ramp

Memory — Construction

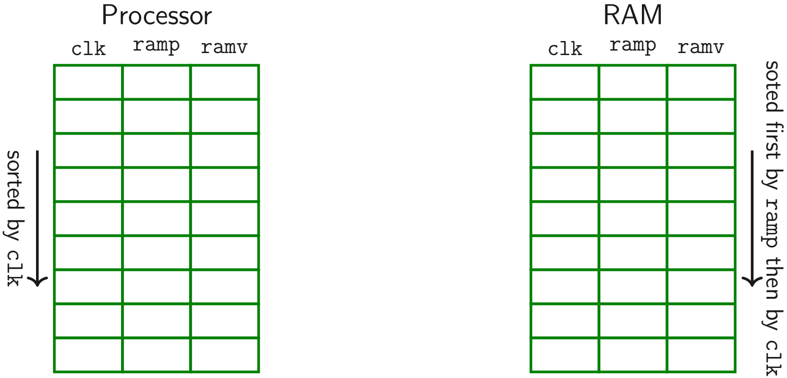




memory integrity ⇐

1. same data, different order
2. within regions of constant `ramp`, correctly sorted by `clk`
3. correctly sorted by `ramp`

Memory — Construction



Processor

clk ramp ramv

sorted by clk

RAM

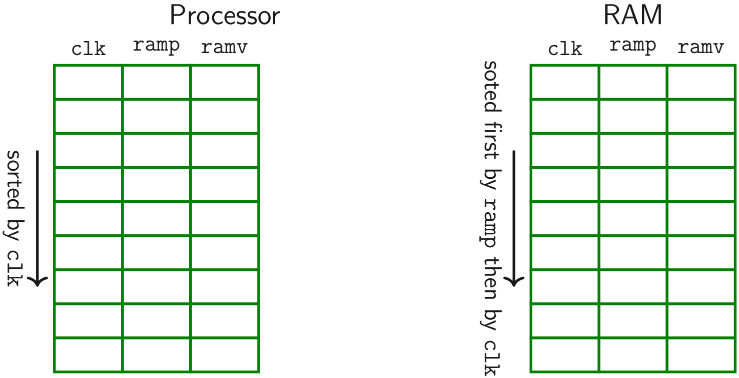
clk ramp ramv

soted first by ramp then by clk

memory integrity ⇐

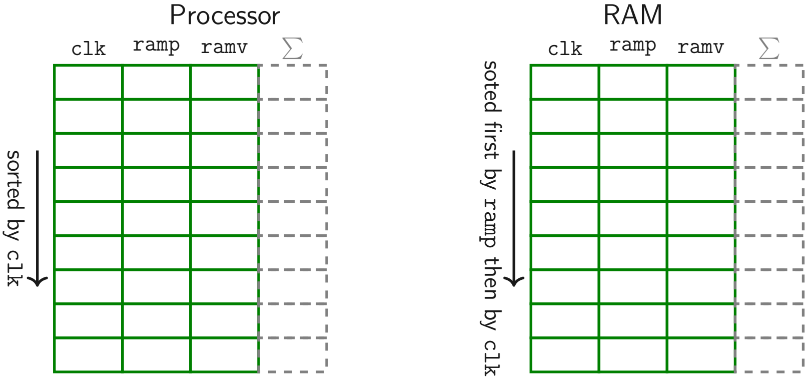
1. same data, different order
2. within regions of constant ramp, correctly sorted by clk
3. ~~correctly sorted by ramp~~ regions of constant ramp are *contiguous*

Memory — Permutation



1. same data, different order
2. within regions of constant `ramp`, correctly sorted by `clk`
3. ~~correctly sorted by `ramp`~~ regions of constant `ramp` are *contiguous*

Memory — Permutation

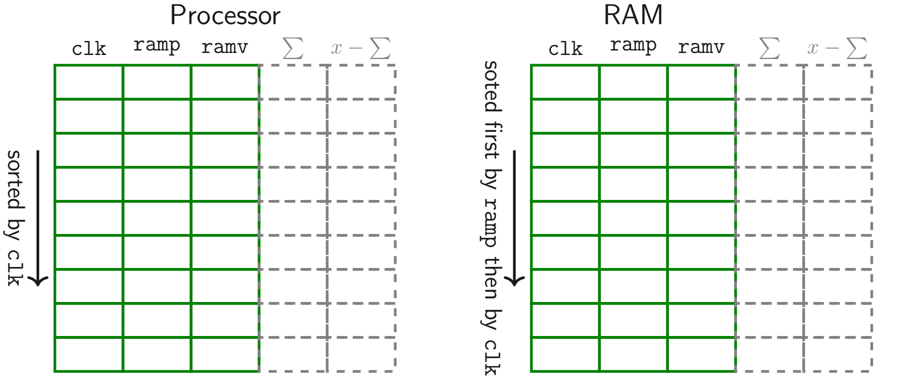


→ 1. same data, different order

2. within regions of constant `ramp`, correctly sorted by `clk`

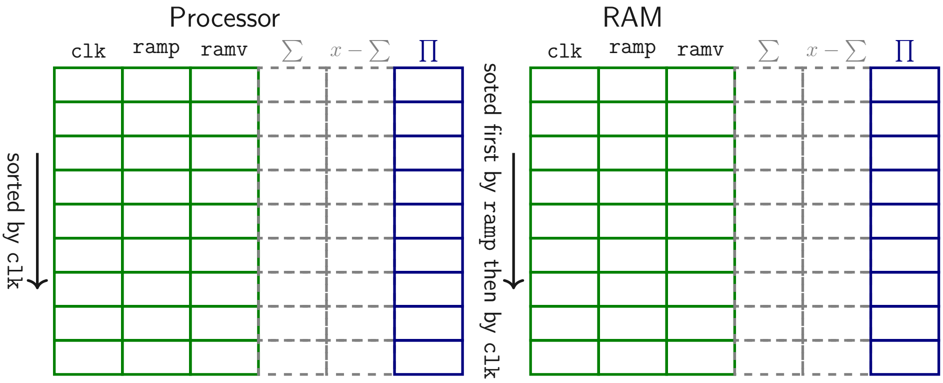
3. ~~correctly sorted by `ramp`~~ regions of constant `ramp` are *contiguous*

Memory — Permutation



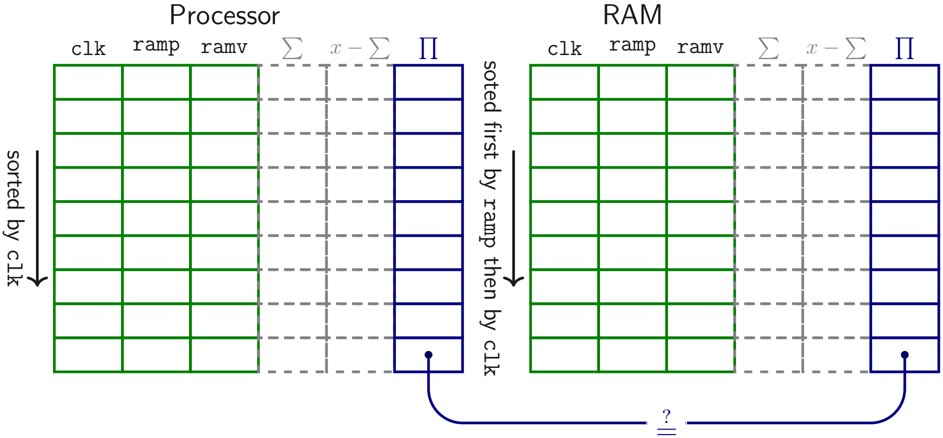
1. same data, different order
2. within regions of constant `ramp`, correctly sorted by `clk`
3. ~~correctly sorted by `ramp`~~ regions of constant `ramp` are *contiguous*

Memory — Permutation



1. same data, different order
2. within regions of constant `ramp`, correctly sorted by `clk`
3. ~~correctly sorted by `ramp`~~ regions of constant `ramp` are *contiguous*

Memory — Permutation

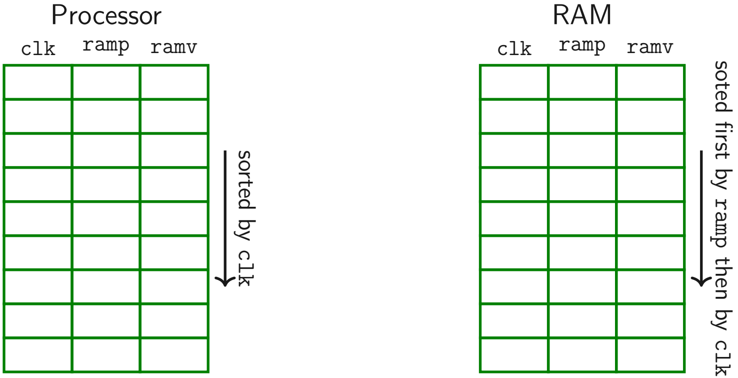


→ 1. same data, different order

2. within regions of constant ramp, correctly sorted by clk

3. ~~correctly sorted by ramp~~ regions of constant ramp are *contiguous*

Memory — Lookup



Processor

clk ramp ramv

sorted by clk

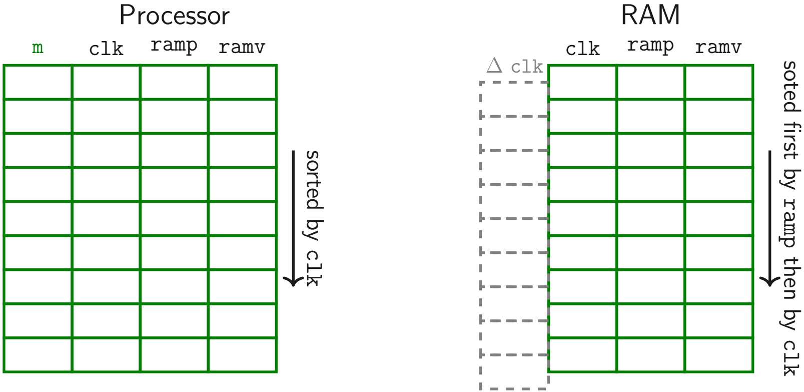
RAM

clk ramp ramv

soted first by ramp then by clk

1. same data, different order
2. within regions of constant ramp, correctly sorted by clk
3. ~~correctly sorted by ramp~~ regions of constant ramp are *contiguous*

Memory — Lookup

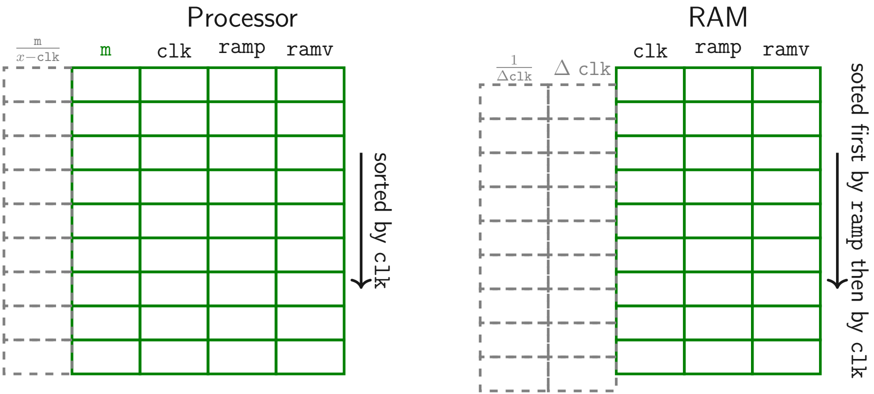


1. same data, different order

→ 2. within regions of constant `ramp`, correctly sorted by `clk`

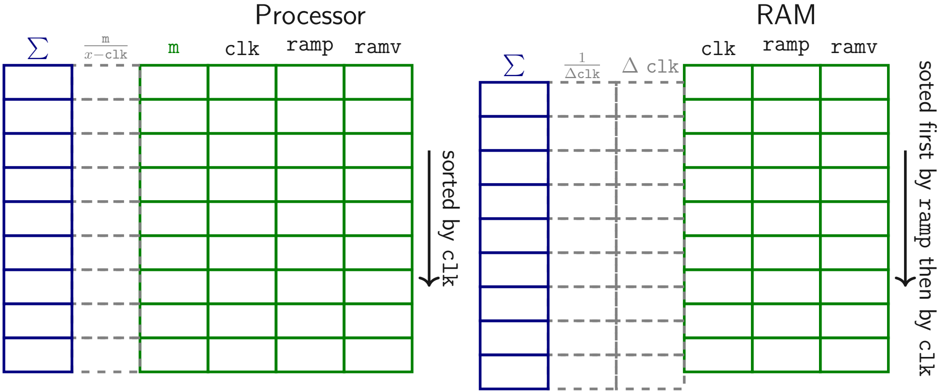
3. ~~correctly sorted by ramp~~ regions of constant `ramp` are *contiguous*

Memory — Lookup


Processor
$\frac{\texttt{m}}{x-\texttt{clk}}$
m
clk
ramp
ramv
sorted by clk
RAM
$\frac{1}{\Delta\texttt{clk}}$
Δ clk
clk
ramp
ramv
soted first by ramp then by clk

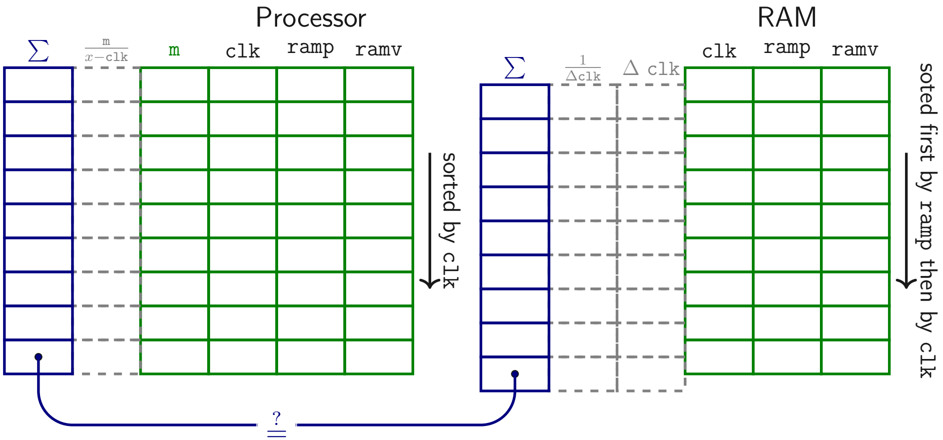
1. same data, different order
2. → within regions of constant `ramp`, correctly sorted by `clk`
3. ~~correctly sorted by `ramp`~~ regions of constant `ramp` are *contiguous*

Memory — Lookup


Processor
$\sum$
$\frac{\texttt{m}}{x-\texttt{clk}}$
m
clk
ramp
ramv
sorted by clk
RAM
$\sum$
$\frac{1}{\Delta\texttt{clk}}$
Δ clk
clk
ramp
ramv
soted first by ramp then by clk

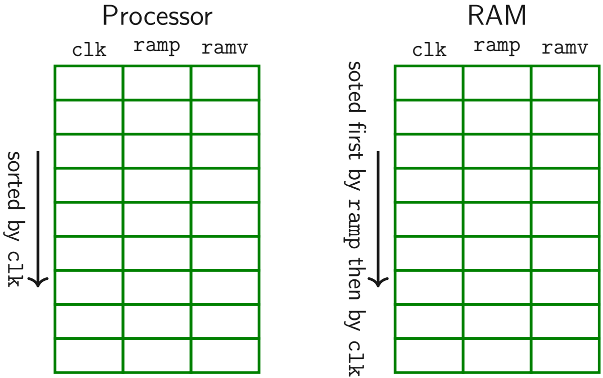
1. same data, different order
2. → within regions of constant `ramp`, correctly sorted by `clk`
3. ~~correctly sorted by ramp~~ regions of constant `ramp` are *contiguous*

Memory — Lookup


Processor
$\sum$
$\frac{\mathtt{m}}{x-\mathtt{clk}}$
m
clk
ramp
ramv
sorted by clk
RAM
$\sum$
$\frac{1}{\Delta\mathtt{clk}}$
Δ clk
clk
ramp
ramv
soted first by ramp then by clk
$\overset{?}{=}$

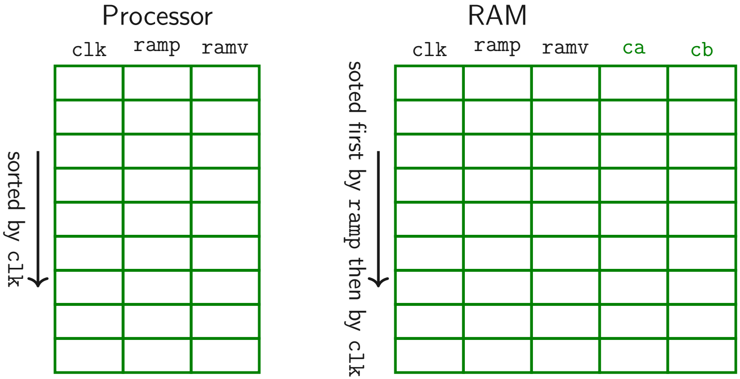
1. same data, different order
2. within regions of constant `ramp`, correctly sorted by `clk`
3. ~~correctly sorted by ramp~~ regions of constant `ramp` are *contiguous*

Memory — Contiguity



1. same data, different order
2. within regions of constant ramp, correctly sorted by clk
3. ~~correctly sorted by ramp~~ regions of constant ramp are *contiguous*

Memory — Contiguity



Processor

clk ramp ramv

sorted by clk

RAM

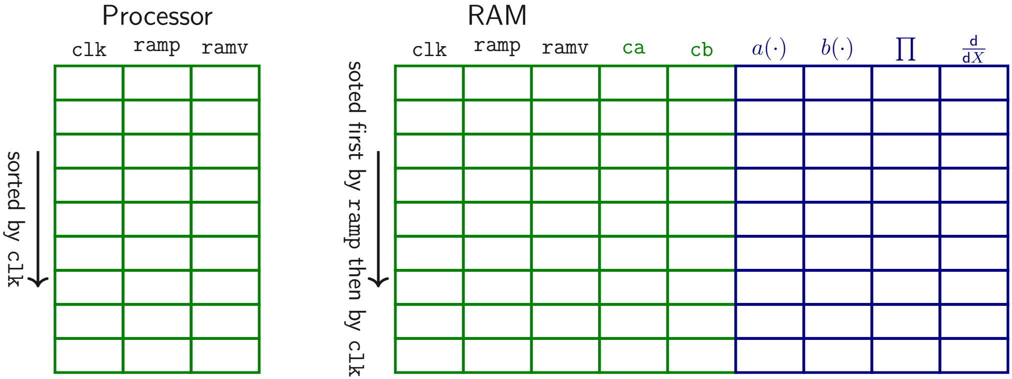
clk ramp ramv ca cb

soted first by ramp then by clk

1. same data, different order
2. within regions of constant ramp, correctly sorted by clk

→ 3. ~~correctly sorted by ramp~~ regions of constant ramp are *contiguous*

Memory — Contiguity



Processor

`clk` `ramp` `ramv`

sorted by `clk`

RAM

`clk` `ramp` `ramv` `ca` `cb` $a(\cdot)$ $b(\cdot)$ $\prod$ $\frac{\mathrm{d}}{\mathrm{d}X}$

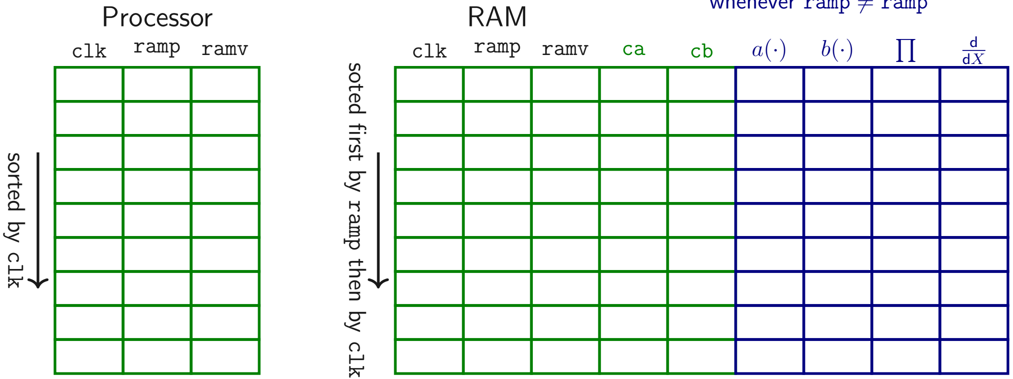
soted first by `ramp` then by `clk`

1. same data, different order
2. within regions of constant `ramp`, correctly sorted by `clk`
3. ~~correctly sorted by `ramp`~~ regions of constant `ramp` are *contiguous*

Memory — Contiguity

$\prod$ accumulates one factor $X - \mathtt{ramp}$

whenever $\mathtt{ramp} \neq \mathtt{ramp}^{\star}$



1. same data, different order
2. within regions of constant `ramp`, correctly sorted by `clk`
3. ~~correctly sorted by `ramp`~~ regions of constant `ramp` are *contiguous*

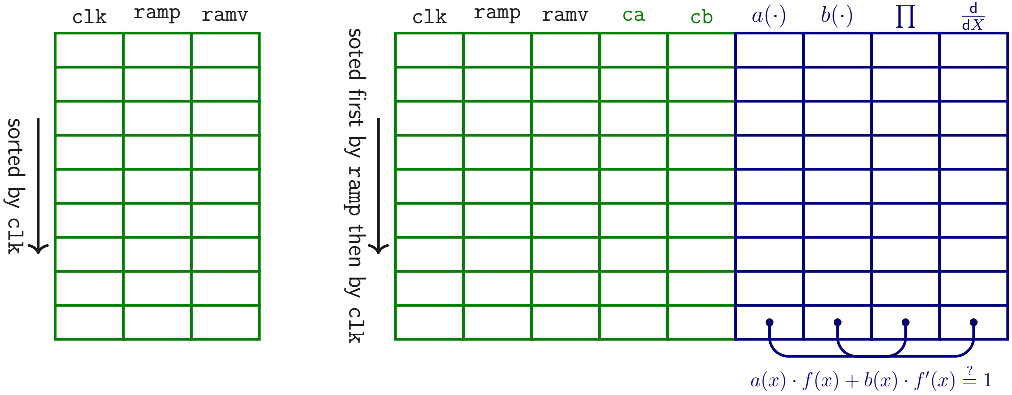
Memory — Contiguity

$\prod$ accumulates one factor $X - \texttt{ramp}$

whenever $\texttt{ramp} \neq \texttt{ramp}^\star$

Processor

RAM



clk ramp ramv

sorted by clk

soted first by ramp then by clk

clk ramp ramv ca cb $a(\cdot)$ $b(\cdot)$ $\prod$ $\frac{\mathrm{d}}{\mathrm{d}X}$

$a(x) \cdot f(x) + b(x) \cdot f'(x) \overset{?}{=} 1$

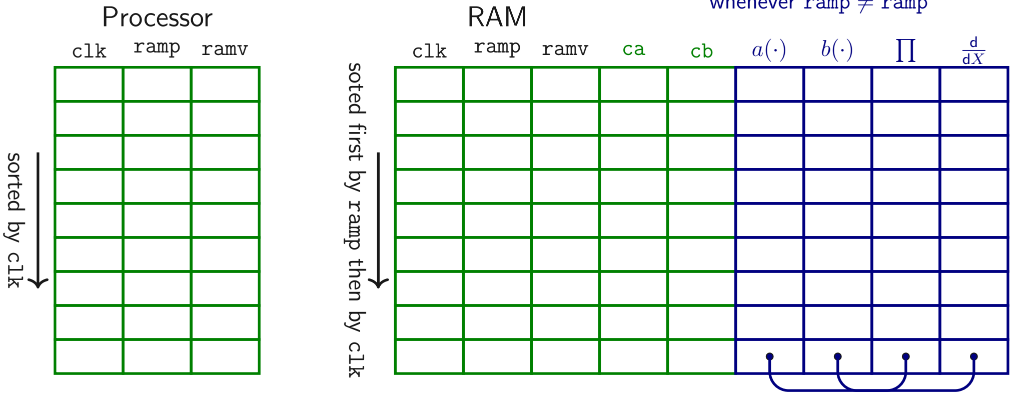
1. same data, different order
2. within regions of constant ramp, correctly sorted by clk

→ 3. ~~correctly sorted by ramp~~ regions of constant ramp are *contiguous*

Memory — Contiguity

$\prod$ accumulates one factor $X -$ `ramp`

whenever `ramp` $\neq$ `ramp`*



Processor

clk ramp ramv

sorted by clk

RAM

clk ramp ramv ca cb $a(\cdot)$ $b(\cdot)$ $\prod$ $\frac{\mathrm{d}}{\mathrm{d}X}$

soted first by ramp then by clk

$$a(x) \cdot f(x) + b(x) \cdot f'(x) \overset{?}{=} 1$$

not contiguous

$\Rightarrow$ repeated factors

$\Rightarrow$ $\gcd \neq 1$

1. same data, different order
2. within regions of constant ramp, correctly sorted by clk
3. ~~correctly sorted by ramp~~ regions of constant ramp are *contiguous*

Table of Contents

- Retrospective
- Optimizations
  - Batching
  - Quotient Segmentation
  - Grinding
- Enhancements
  - Zero-Knowledge
  - Randomized AIR with Preprocessing
- VM Architecture
  - Overview
  - Communication Arguments
  - Memory
- Other Topics

Table of Contents

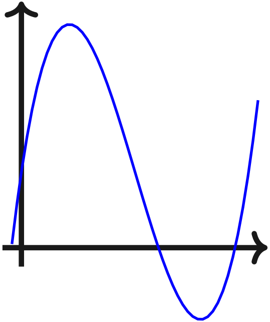
- Retrospective
- Optimizations
  - Batching
  - Quotient Segmentation
  - Grinding
- Enhancements
  - Zero-Knowledge
  - Randomized AIR with Preprocessing
- VM Architecture
  - Overview
  - Communication Arguments
  - Memory
- Other Topics

Table of Contents

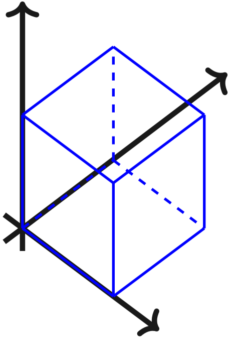
- Retrospective
- Optimizations
  - Batching
  - Quotient Segmentation
  - Grinding
- Enhancements
  - Zero-Knowledge
  - Randomized AIR with Preprocessing
- VM Architecture
  - Overview
  - Communication Arguments
  - Memory
- Other Topics

Univariate versus Multilinear

Univariate

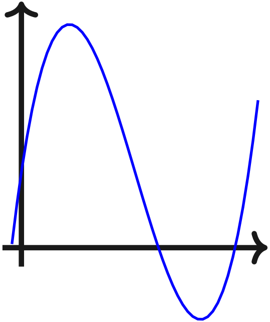


Multilinear



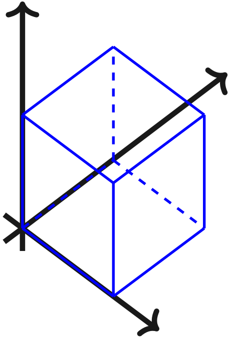
Univariate versus Multilinear

Univariate



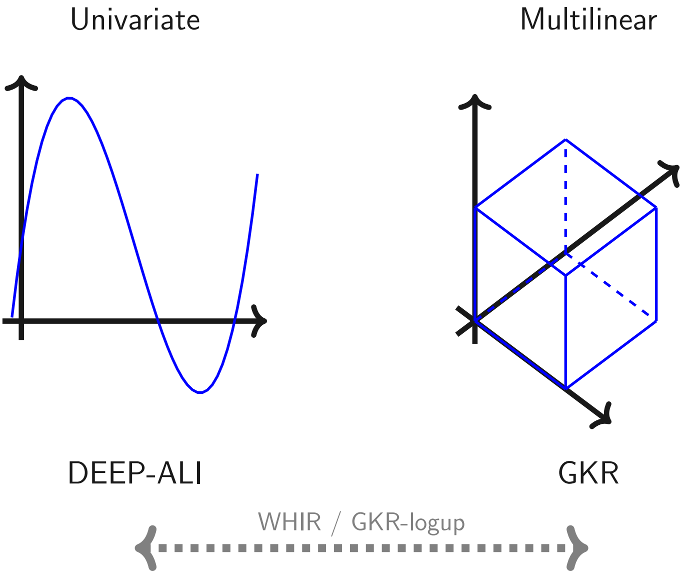
DEEP-ALI

Multilinear



GKR

Univariate versus Multilinear



Univariate

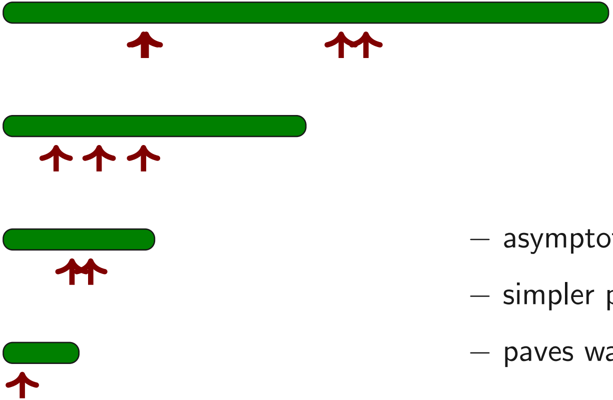
Multilinear

DEEP-ALI

GKR

WHIR / GKR-logup

STIR

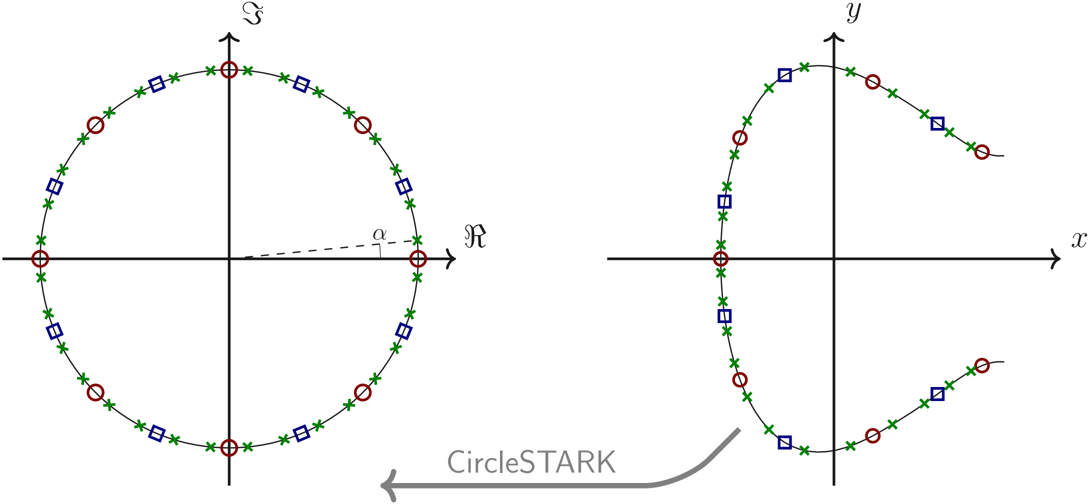


— asymptotically and concretely better than FRI

— simpler proof of soundness

— paves way for aggregation

ECFFT



Structured Fields

Arbitrary Fields

Table of Contents

Retrospective

Optimizations

- Batching
- Quotient Segmentation
- Grinding

Enhancements

- Zero-Knowledge
- Randomized AIR with Preprocessing

VM Architecture

- Overview
- Communication Arguments
- Memory

Other Topics

Table of Contents

- Retrospective
- Optimizations
  - Batching
  - Quotient Segmentation
  - Grinding
- Enhancements
  - Zero-Knowledge
  - Randomized AIR with Preprocessing
- VM Architecture
  - Overview
  - Communication Arguments
  - Memory
- Other Topics

Advanced zk-STARKs

Alan Szepieniec
艾伦·佘丕涅茨

alan@neptune.cash

https://neptune.cash/

https://triton-vm.org/

https://asz.ink/presentations/2025-09-18-Advanced-zkSTARKs.pdf