Advanced zk-STARKs

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

neptune

Triton VM

https://neptune.cash/

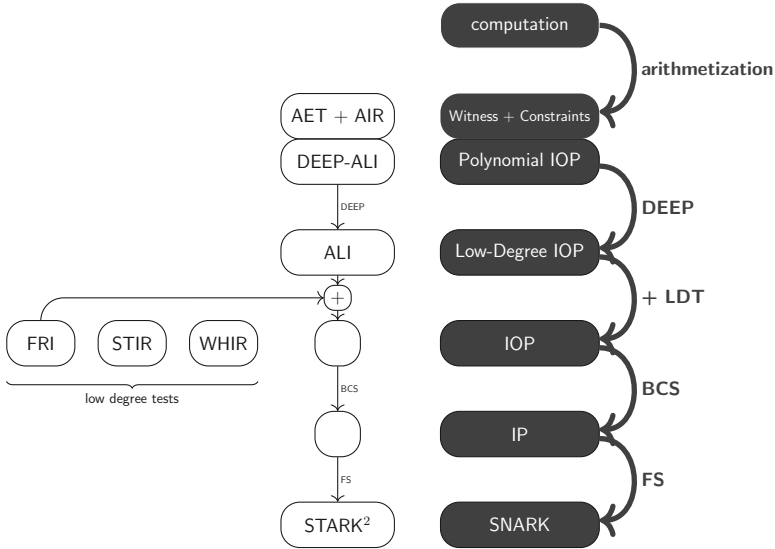
https://triton-vm.org/

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

Table of Contents

Table of Contents

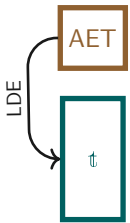
STARK Compilation Pipeline



STARK Diagram

algebraic execution trace

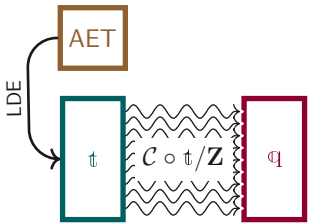
STARK Diagram



low-degree extension

low-degree extended trace

STARK Diagram

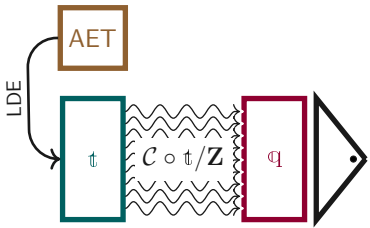


composition with AIR constraints

division by zerofiers

quotients

STARK Diagram

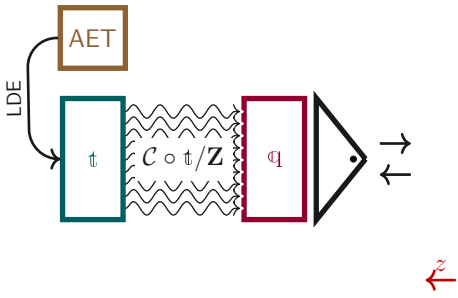


build Merkle tree

STARK Diagram

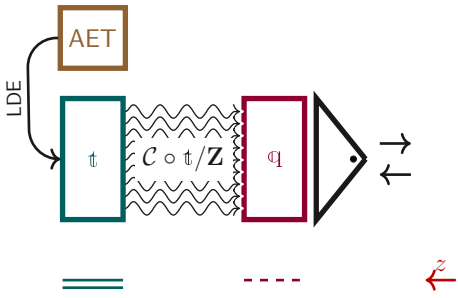
interact with verifier

STARK Diagram



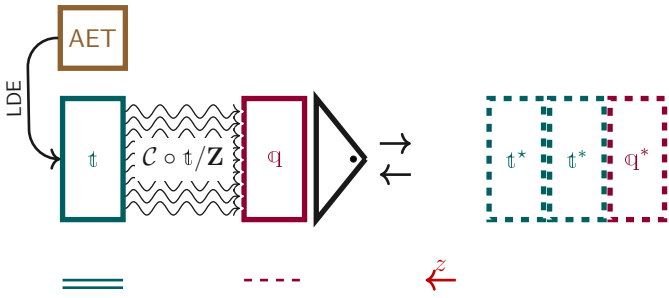
sample out-of-domain point

STARK Diagram

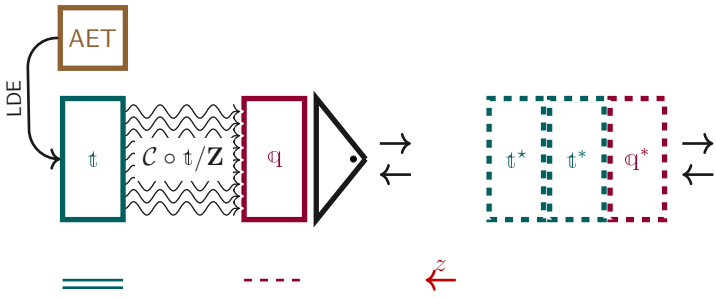


produce out-of-domain rows

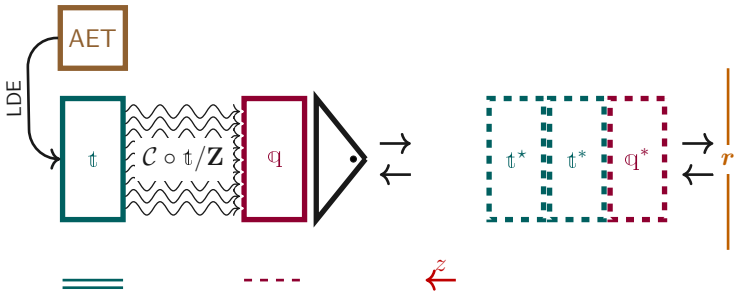
STARK Diagram



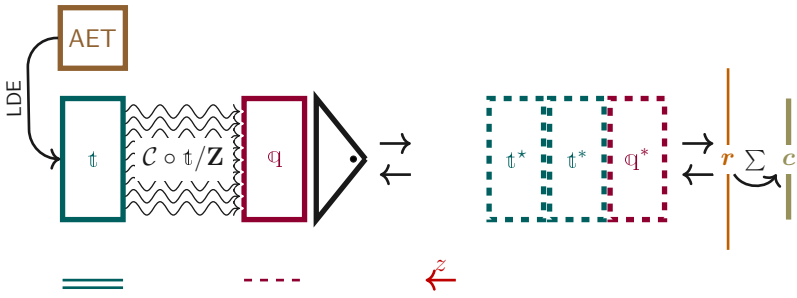
STARK Diagram



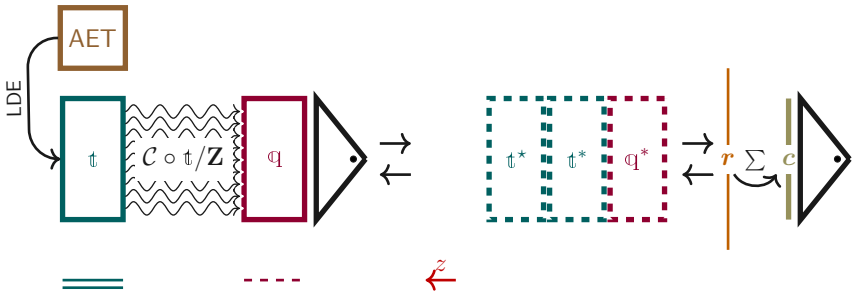
STARK Diagram



STARK Diagram

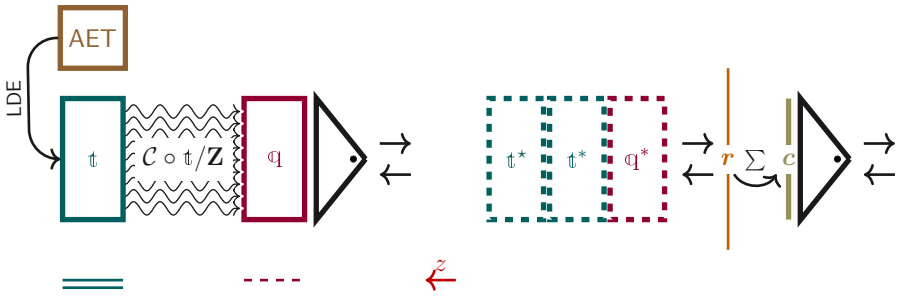


STARK Diagram



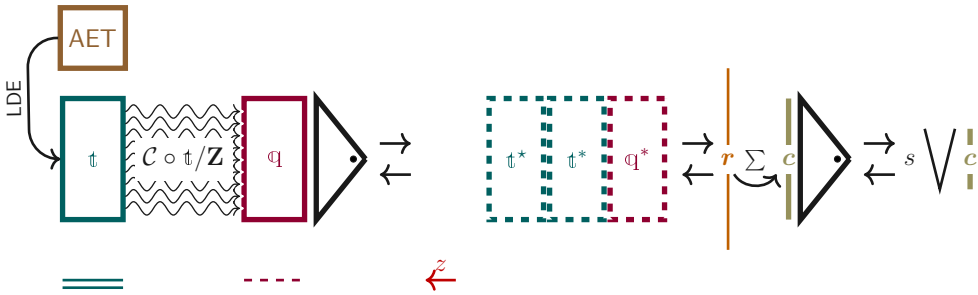
build Merkle tree

STARK Diagram



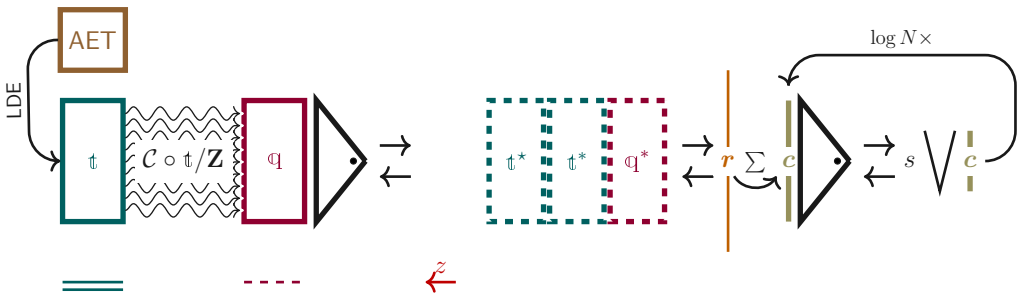
interact with verifier

STARK Diagram



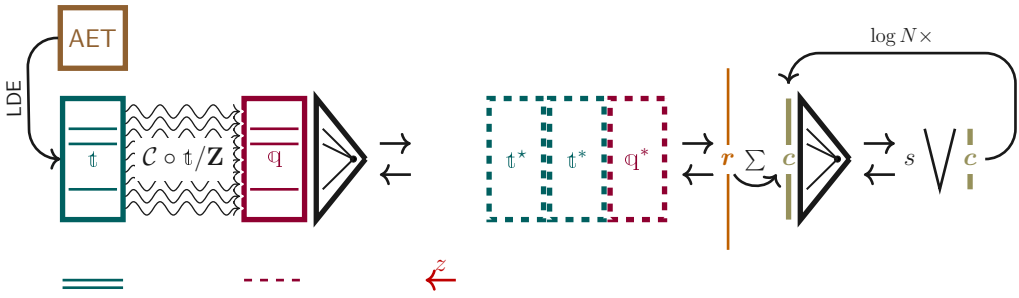
split-and-fold

STARK Diagram



rinse and repeat

STARK Diagram



obtain FRI indices
open indicated rows

STARK Diagram

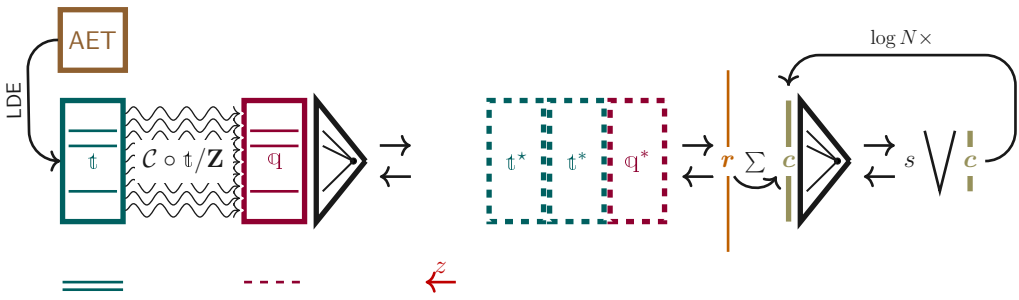


Table of Contents

Table of Contents

Table of Contents

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

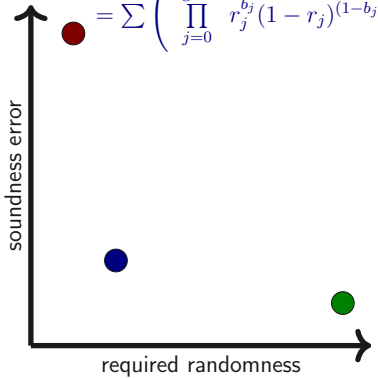
- linear $\sum_{i=0}^{n-1} r_i \boldsymbol{c}_i$ $\epsilon = \epsilon_{\mathsf{GAP}}(\delta_0)$
- univariate $\sum_{i=0}^{n-1} r^i \boldsymbol{c}_i$ $\epsilon = n \cdot \epsilon_{\mathsf{GAP}}(\delta_0)$
- multilinear $\sum_{i=0}^{n-1} \boldsymbol{r}^{\llcorner i \lrcorner} \boldsymbol{c}_i$ $\epsilon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

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

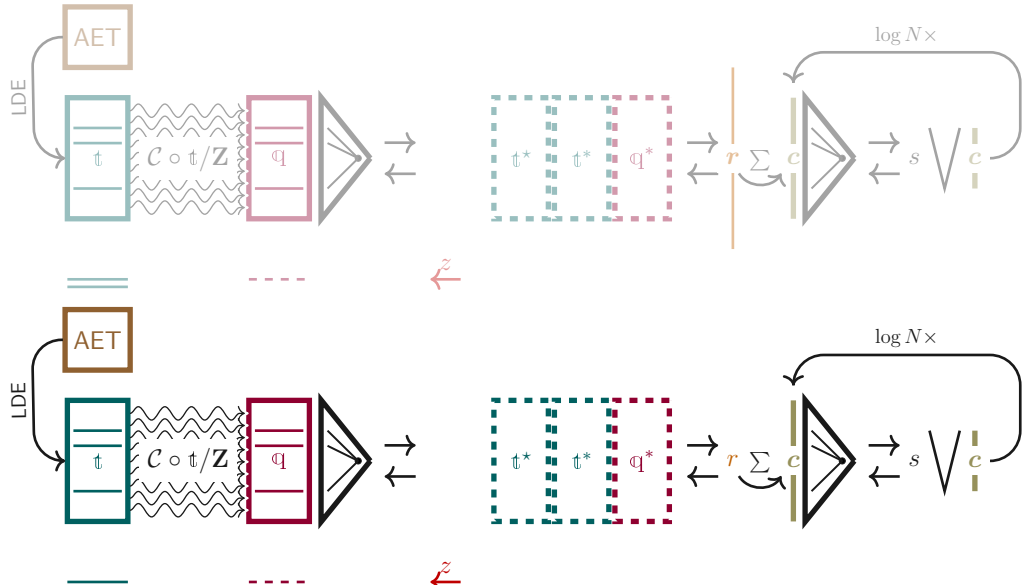
Batching

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

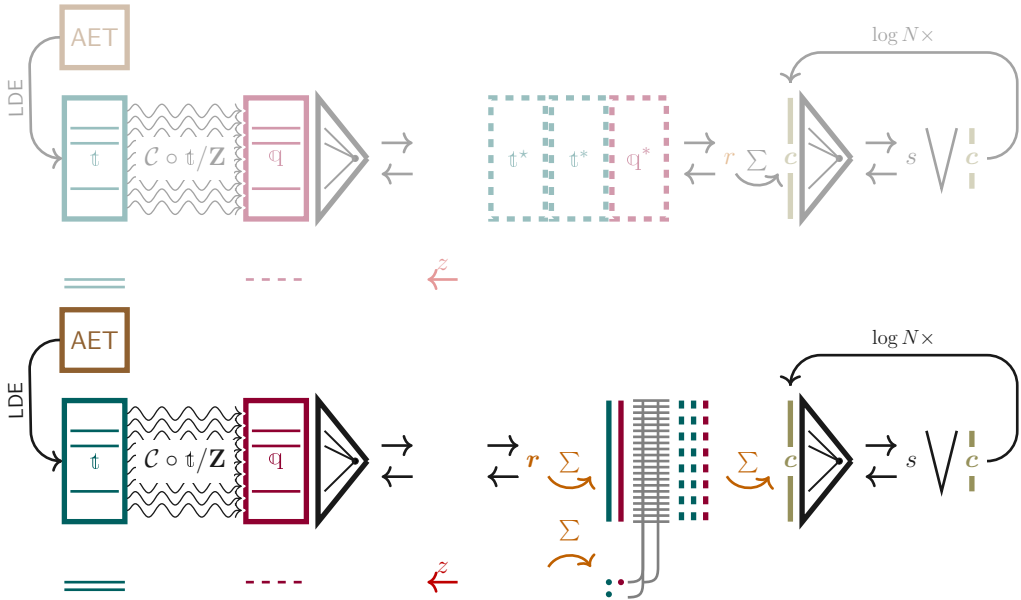
$$= \sum \left(\prod_{j=0}^{\log n - 1} r_j^{b_j} (1 - r_j)^{(1-b_j)} \right) \boldsymbol{c}_i$$



Univariate Batching



Batch Before DEEP



Batch Constraints

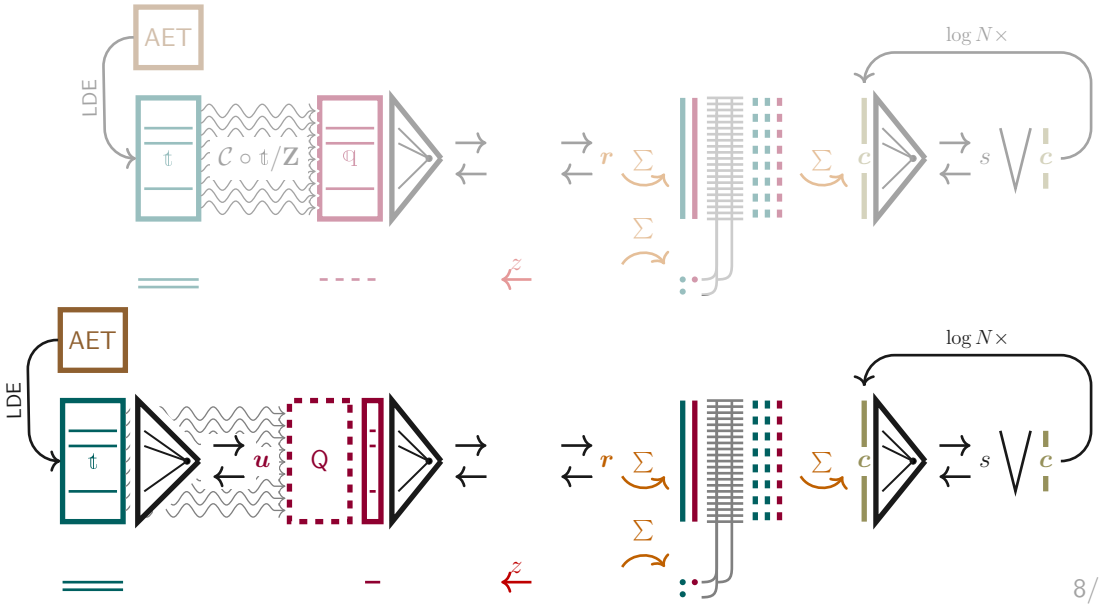


Table of Contents

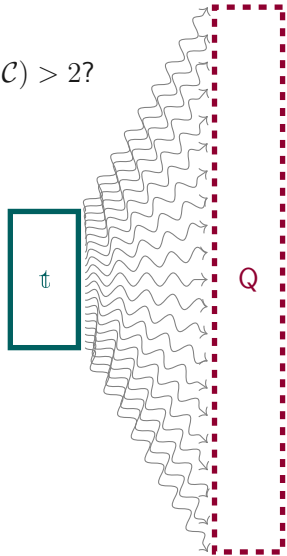
Table of Contents

Quotient Segmentation

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

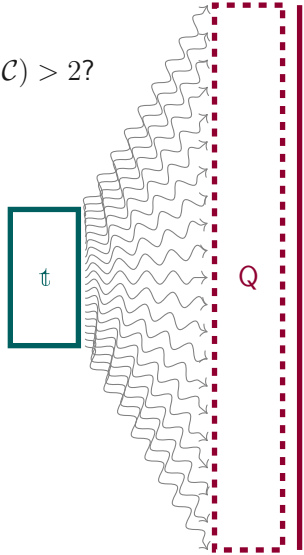
Quotient Segmentation

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



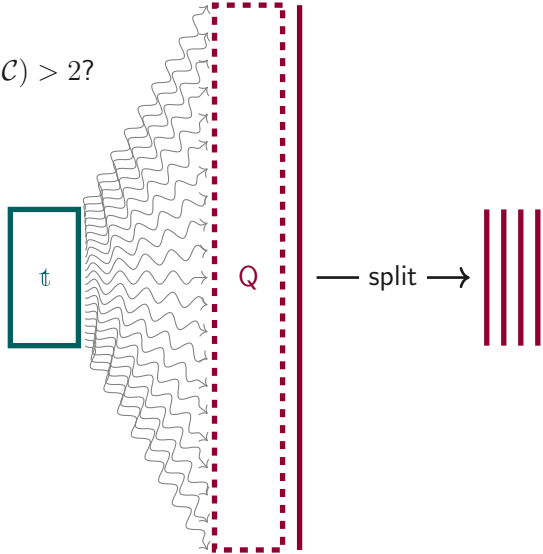
Quotient Segmentation

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

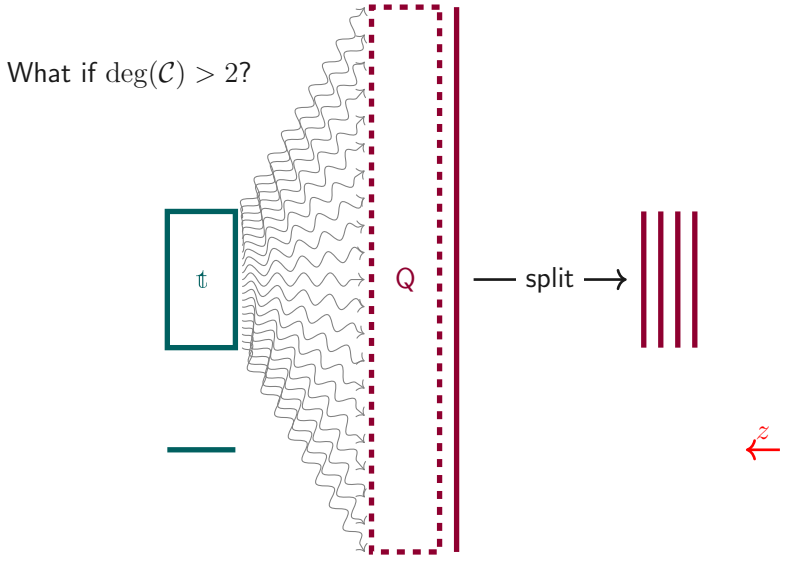


Quotient Segmentation

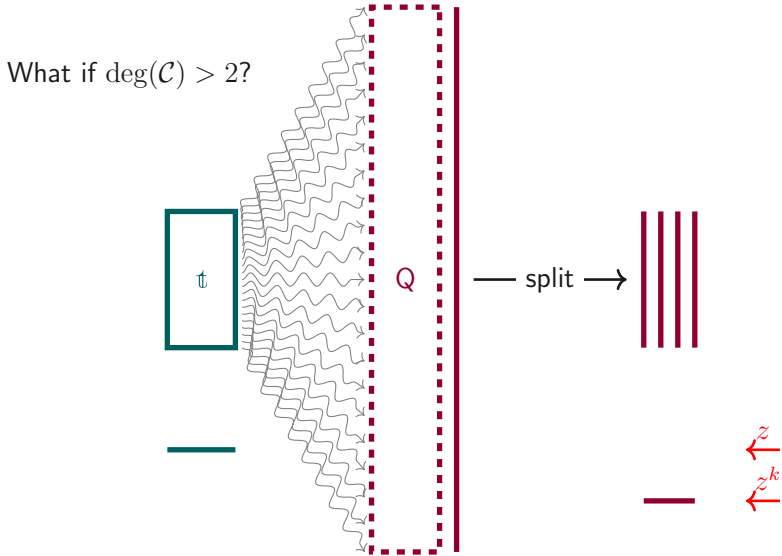
What if $\deg(\mathcal{C}) > 2$?



Quotient Segmentation



Quotient Segmentation



Quotient Segmentation (Diagram)

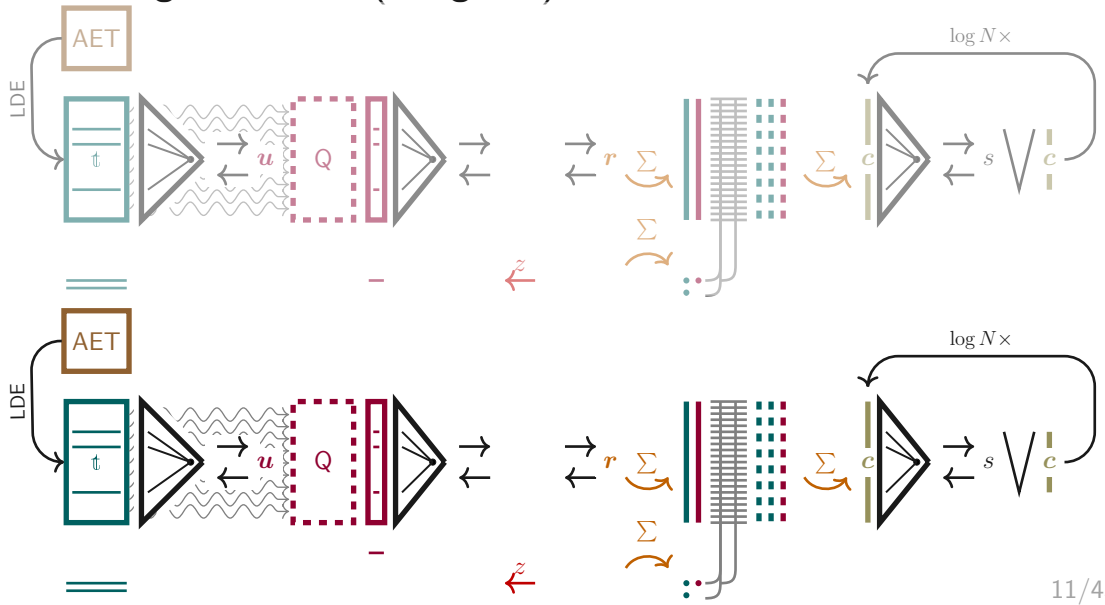
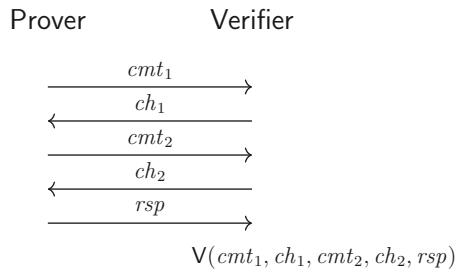


Table of Contents

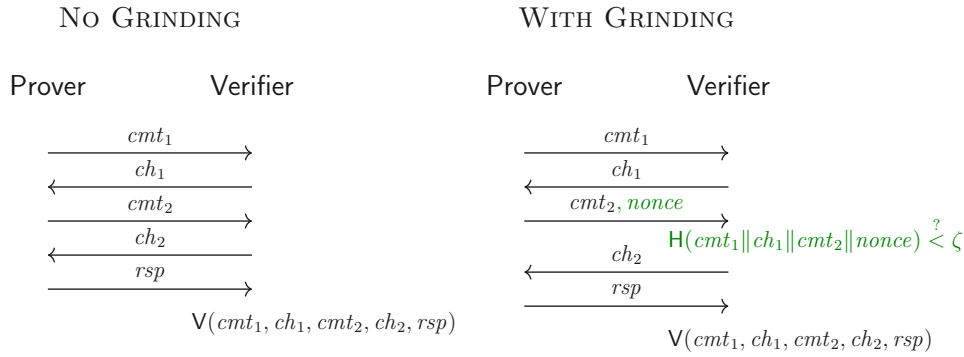
Table of Contents

Grinding

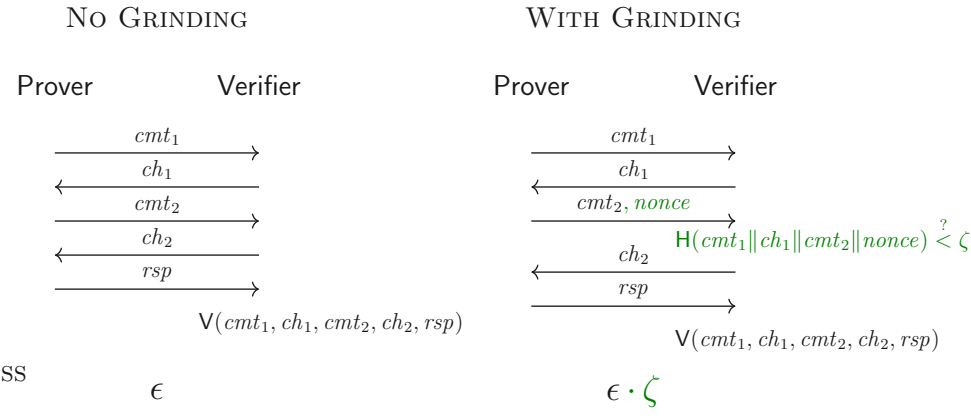
No Grinding



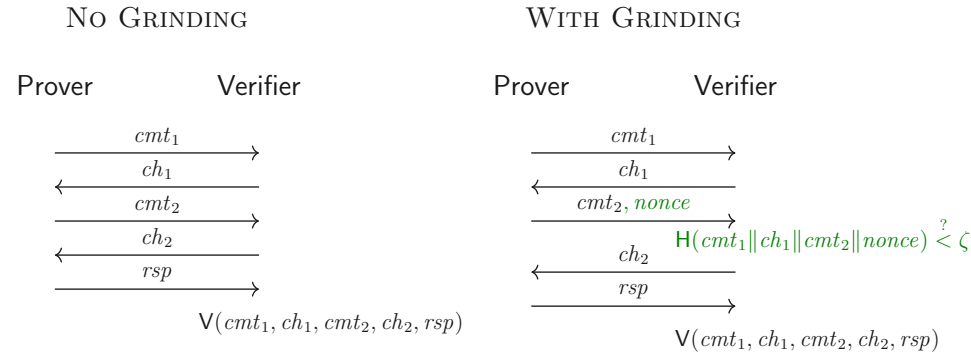
Grinding


No Grinding
Prover
Verifier
cmt_1
ch_1
cmt_2
ch_2
rsp
$\mathsf{V}(cmt_1, ch_1, cmt_2, ch_2, rsp)$
With Grinding
Prover
Verifier
cmt_1
ch_1
$cmt_2, nonce$
$\mathsf{H}(cmt_1 \| ch_1 \| cmt_2 \| nonce) \stackrel{?}{<} \zeta$
ch_2
rsp
$\mathsf{V}(cmt_1, ch_1, cmt_2, ch_2, rsp)$

Grinding


No Grinding
Prover
Verifier
cmt_1
ch_1
cmt_2
ch_2
rsp
$\mathsf{V}(cmt_1, ch_1, cmt_2, ch_2, rsp)$
With Grinding
Prover
Verifier
cmt_1
ch_1
$cmt_2, nonce$
$\mathsf{H}(cmt_1 \| ch_1 \| cmt_2 \| nonce) \stackrel{?}{<} \zeta$
ch_2
rsp
$\mathsf{V}(cmt_1, ch_1, cmt_2, ch_2, rsp)$
Soundness Error
ϵ
$\epsilon \cdot \zeta$

Grinding


No Grinding
Prover
Verifier
cmt_1
ch_1
cmt_2
ch_2
rsp
$\mathsf{V}(cmt_1, ch_1, cmt_2, ch_2, rsp)$
With Grinding
Prover
Verifier
cmt_1
ch_1
$cmt_2, nonce$
$\mathsf{H}(cmt_1 \| ch_1 \| cmt_2 \| nonce) \stackrel{?}{<} \zeta$
ch_2
rsp
$\mathsf{V}(cmt_1, ch_1, cmt_2, ch_2, rsp)$
Soundness Error
ϵ
$\epsilon \cdot \zeta$
Prover Work
W
$W + \zeta^{-1}$

Grinding

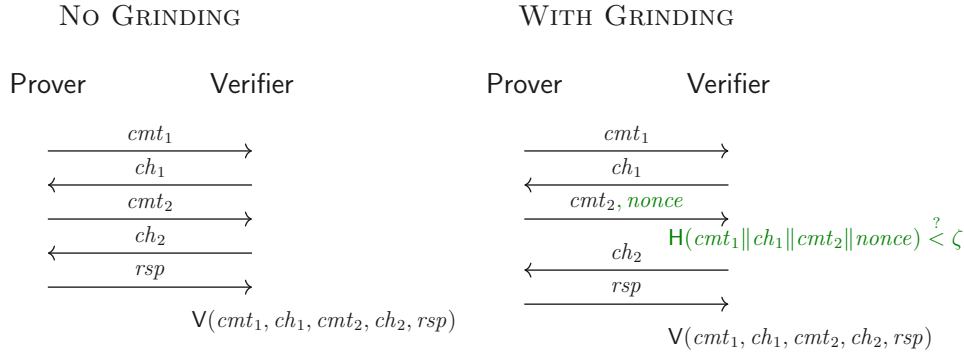

No Grinding
Prover
Verifier
cmt_1
ch_1
cmt_2
ch_2
rsp
$\mathsf{V}(cmt_1, ch_1, cmt_2, ch_2, rsp)$
With Grinding
Prover
Verifier
cmt_1
ch_1
$cmt_2, nonce$
$\mathsf{H}(cmt_1 \| ch_1 \| cmt_2 \| nonce) \stackrel{?}{<} \zeta$
ch_2
rsp
$\mathsf{V}(cmt_1, ch_1, cmt_2, ch_2, rsp)$
Soundness Error
ϵ
$\epsilon \cdot \zeta$
+ 10 bits
Prover Work
W
$\approx 2^{30}$
$W + \zeta^{-1}$
+ 0.1%

Table of Contents

Table of Contents

Table of Contents

Table of Contents

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 $\Leftrightarrow$ transcript is independent of witness

$\rightarrow$ *mask with randomness*

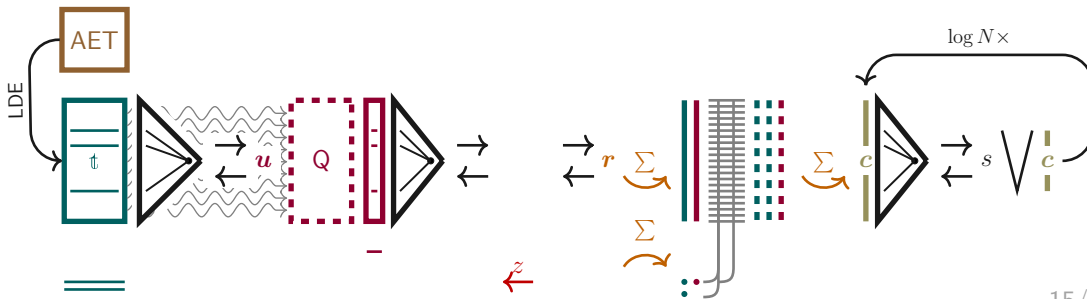
1. salted Merkle leafs (optional)
2. batch randomizer polynomial
3. trace randomizer values

Zero Knowledge

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

$\rightarrow$ *mask with randomness*

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

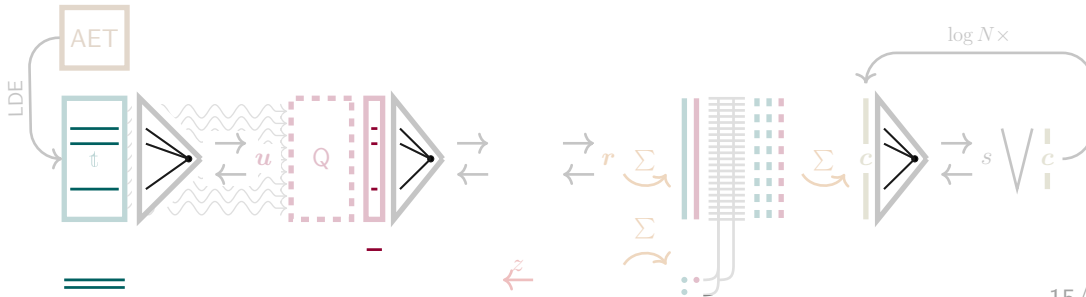


Zero Knowledge

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

$\rightarrow$ *mask with randomness*

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

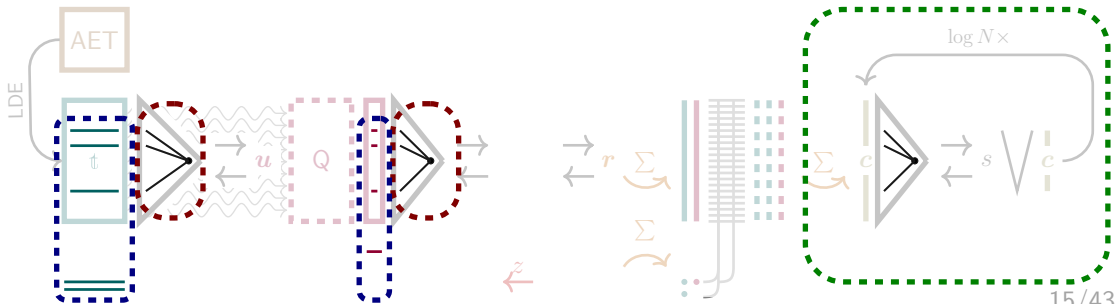


Zero Knowledge

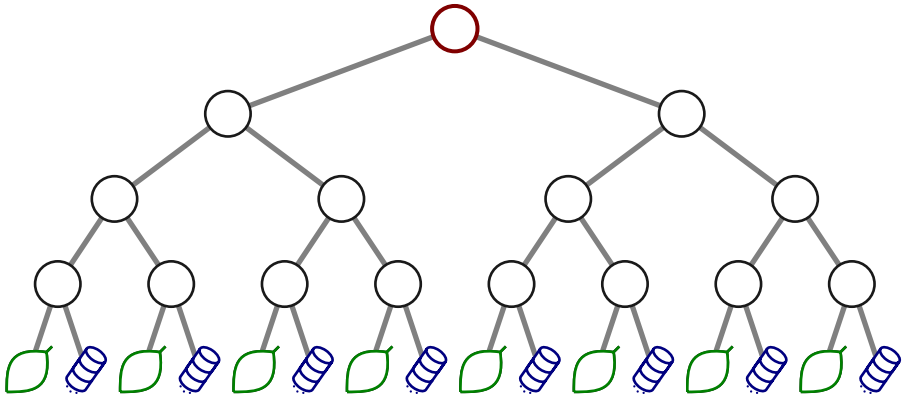
zero-knowledge $\Leftrightarrow$ transcript is independent of witness

$\rightarrow$ *mask with randomness*

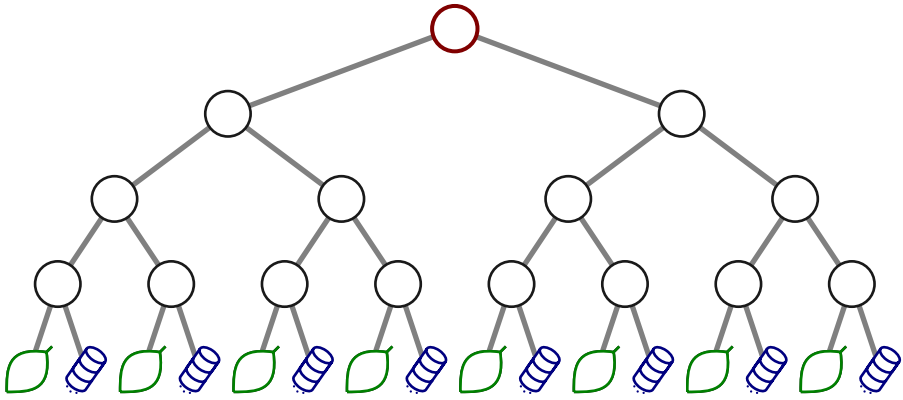
1. salted Merkle leafs (optional)
2. batch randomizer polynomial
3. trace randomizer values



Salted Merkle Leafs



Salted Merkle Leafs



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

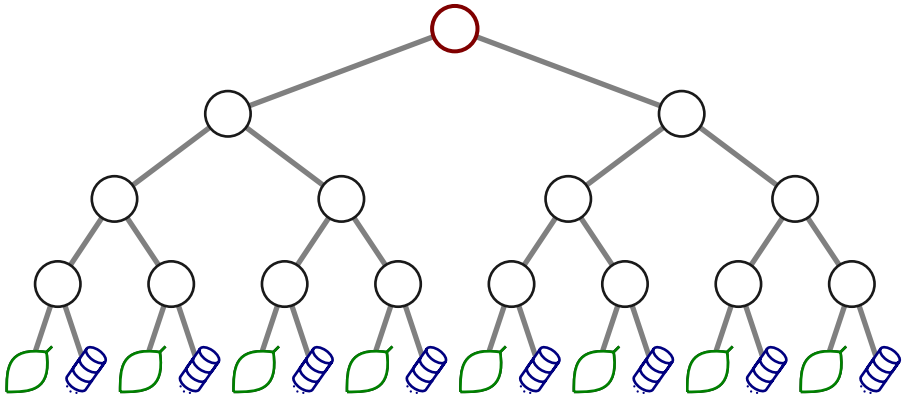
leafs

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

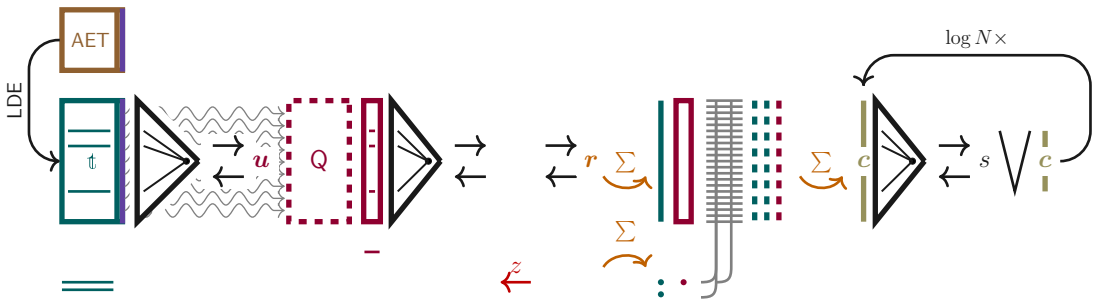
→ Why: make low-degree tested codeword independent of trace

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

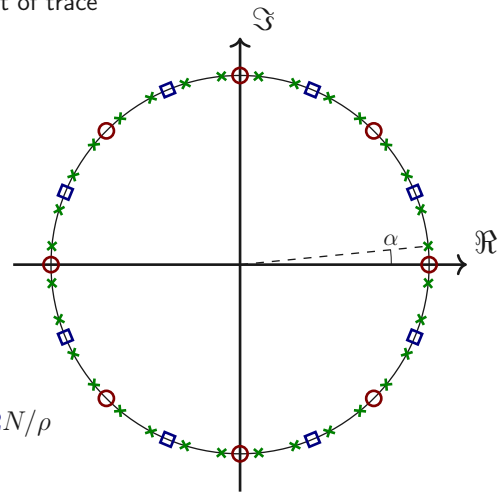
Trace Randomizer Values: Interleaving

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

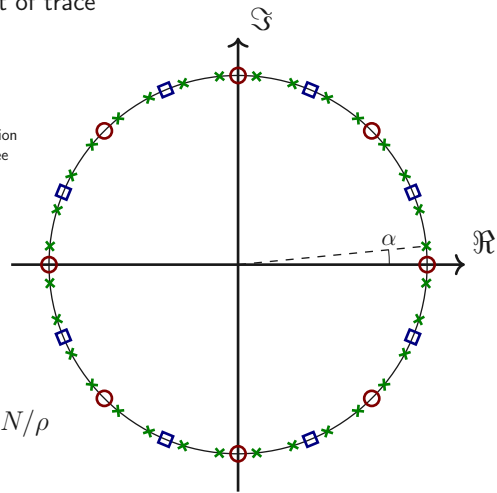
interleave trace with random rows

→ Why: make observed rows independent of trace

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

- ○ 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

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

Trace Randomizer Values: Stingy

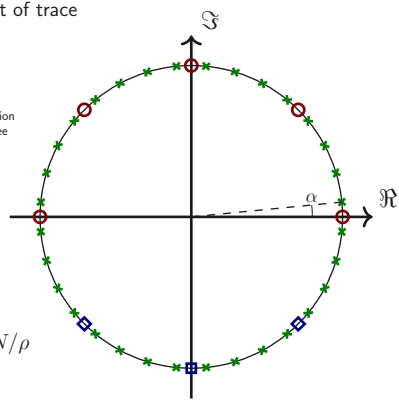
pad then concatenate random rows

→ Why: make observed rows independent of trace

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

- trace values
- randomizer values
- evaluation domain
  = coset of subgroup of order N/ρ

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



Trace Randomizer Values: Stingy

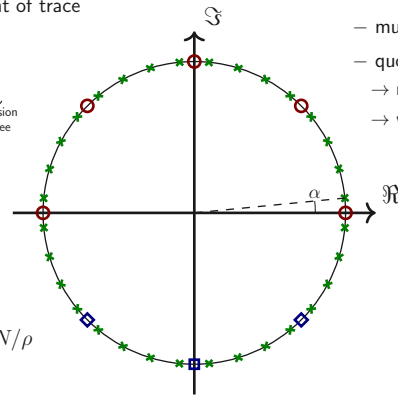
pad then concatenate random rows

→ Why: make observed rows independent of trace

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

- 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

Table of Contents

Two-Stage DEEP-ALI

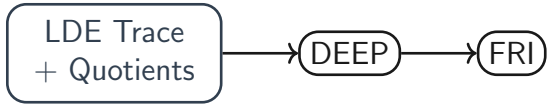
0-stage:

1-stage:

2-stage:

Two-Stage DEEP-ALI

0-stage:

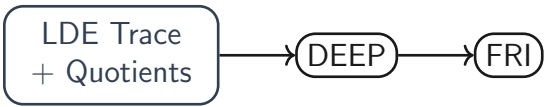


1-stage:

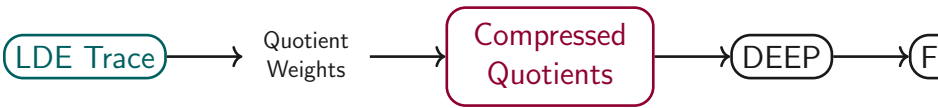
2-stage:

Two-Stage DEEP-ALI

0-stage:

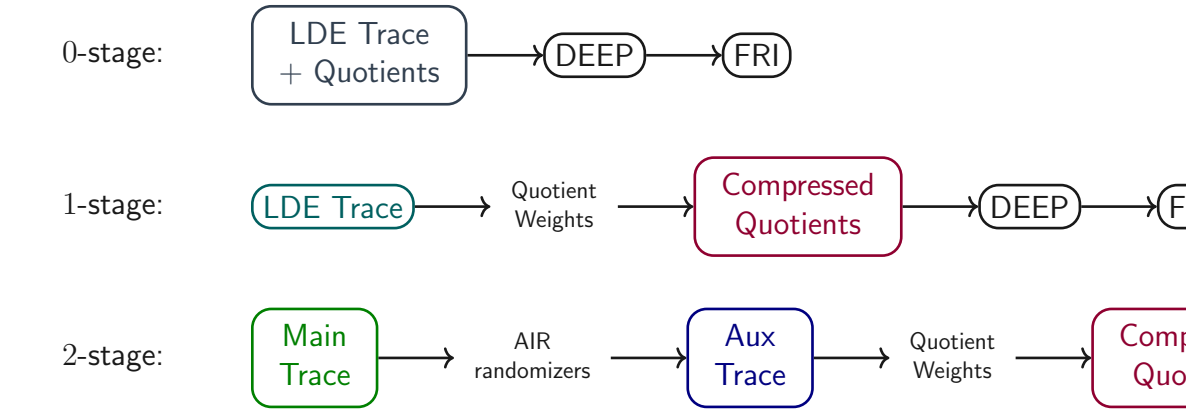


1-stage:

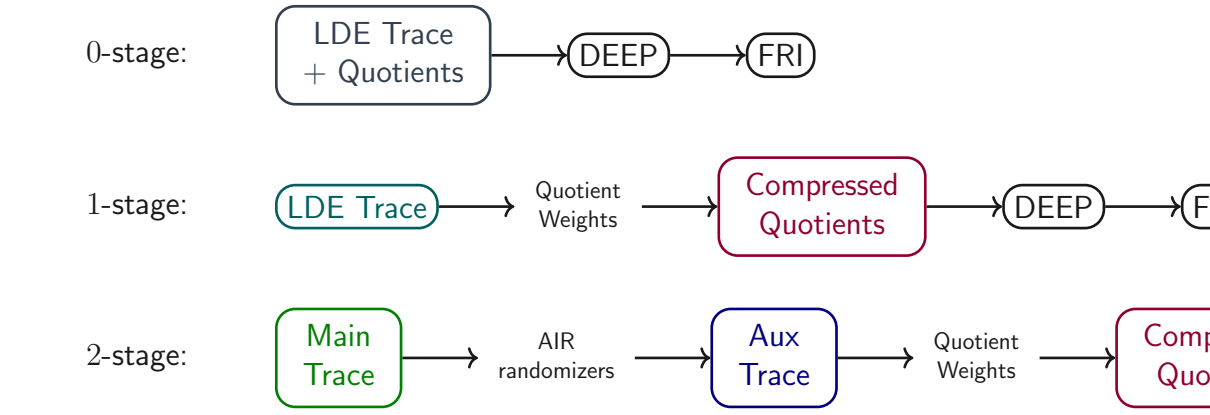


2-stage:

Two-Stage DEEP-ALI

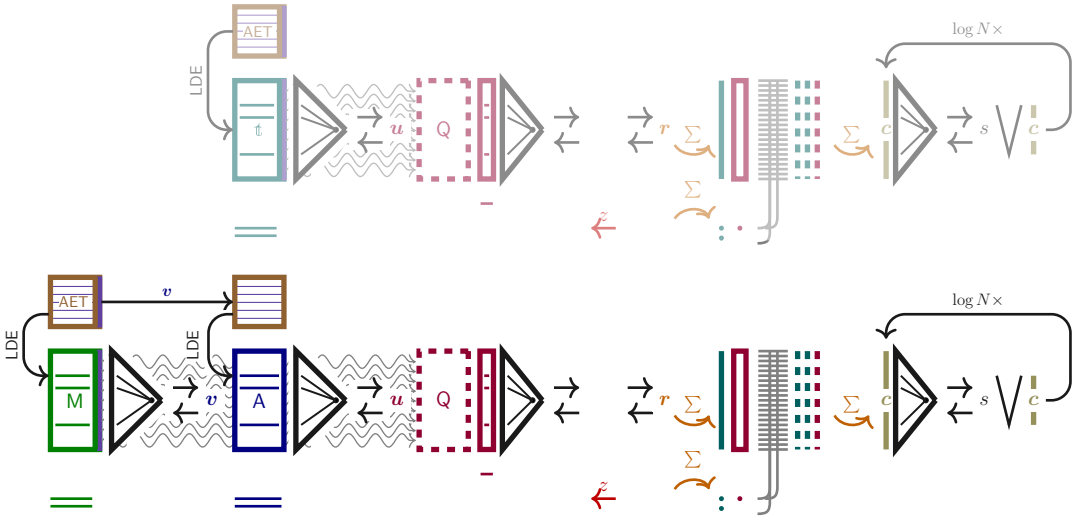


Two-Stage DEEP-ALI



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

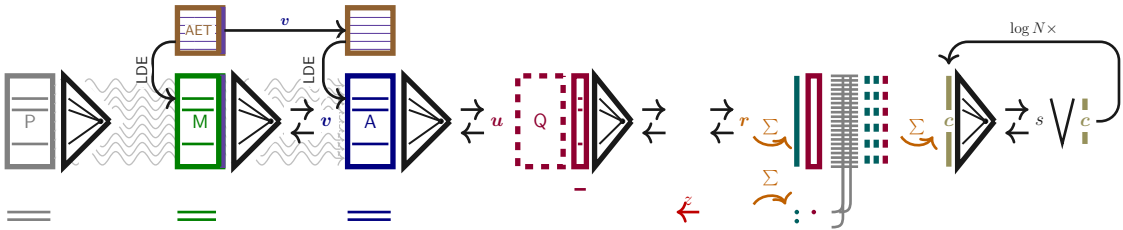


Table Of Contents

Table Of Contents

Table Of Contents

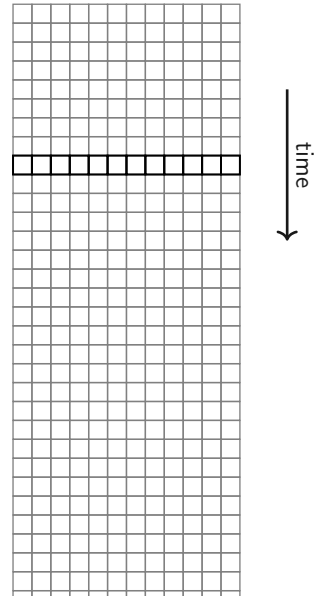
Table Of Contents

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*
$\texttt{arg}_0$	*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 AIR Constraints (Example)

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

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

$$\mathsf{jump} = \mathsf{dest} - \mathsf{ip}^\star \qquad \textit{update value of } \texttt{ip} \textit{ (jump case)}$$

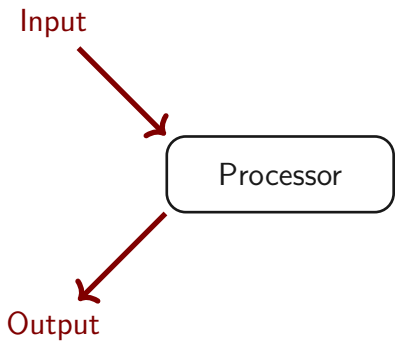
$$\mathsf{nojump} = \mathsf{ip} + 1 - \mathsf{ip}^\star \qquad \textit{update value of } \texttt{ip} \textit{ (no jump)}$$

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

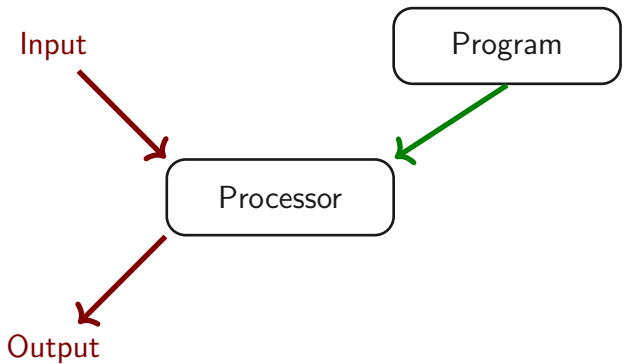
$$\boxed{\mathsf{selector} \cdot \mathsf{jump} + (1 - \mathsf{selector}) \cdot \mathsf{nojump}}$$

Communication Lines

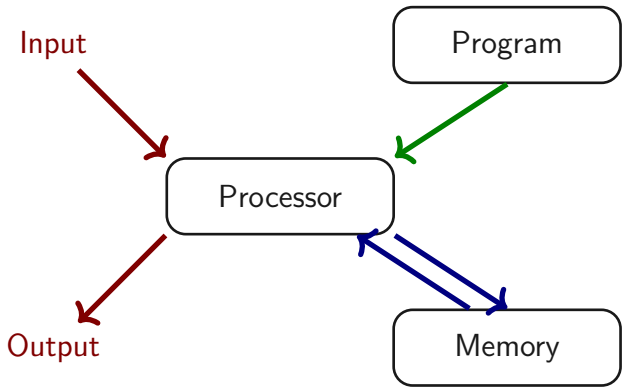
Communication Lines



Communication Lines



Communication Lines



Communication Lines

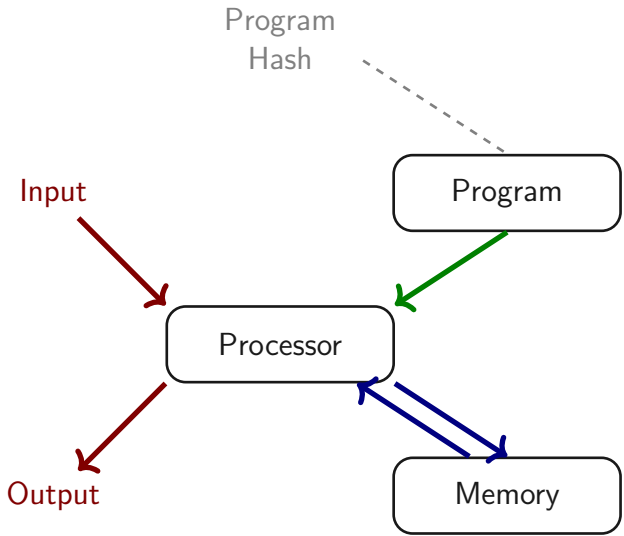


Table of Contents

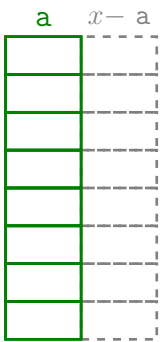
Table of Contents

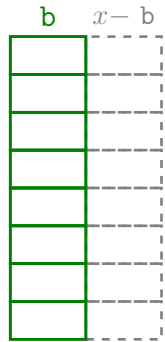
Permutation Argument

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

Permutation Argument

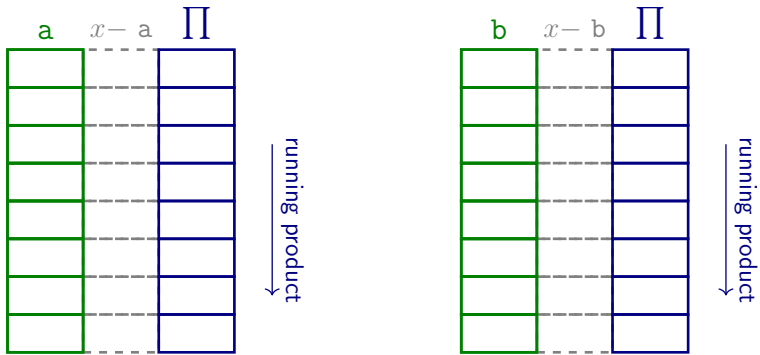
$b = \sigma(a)$ for some permutation σ





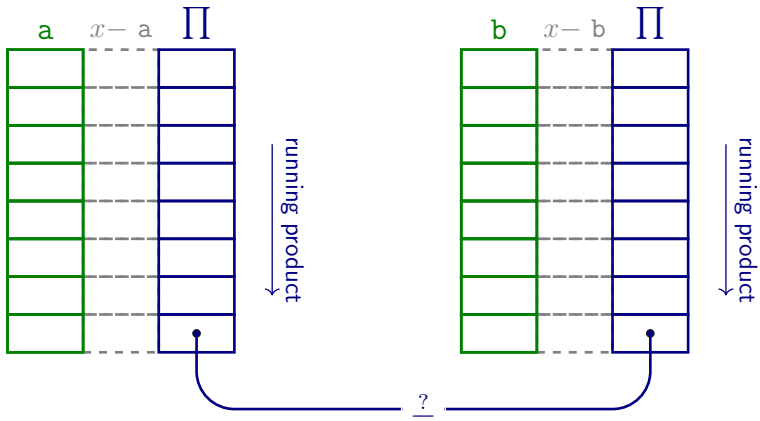
Permutation Argument

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



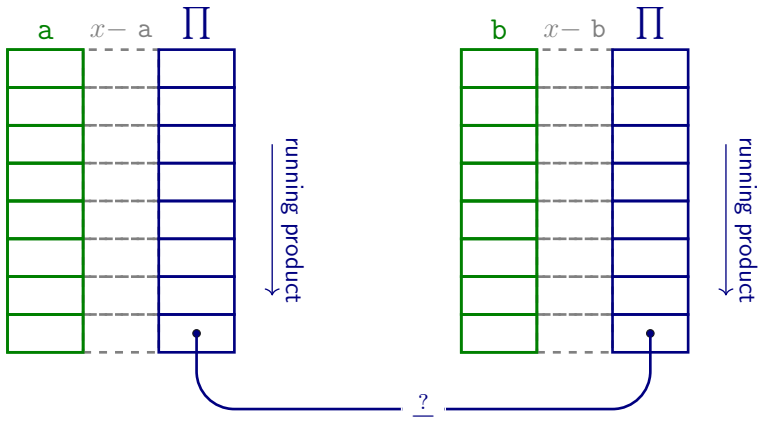
Permutation Argument

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



Permutation Argument

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

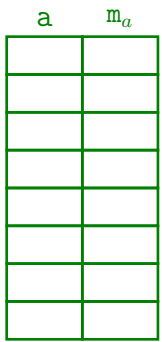


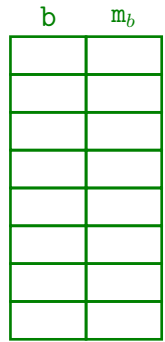
Soundness

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

Evaluation Argument

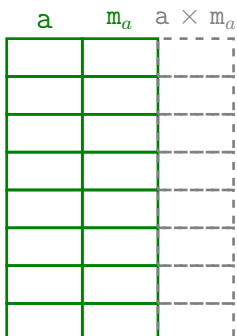
$$a[m_a] = b[m_b] \text{ for } m_a, m_b \subseteq \{0, \ldots, N-1\}$$

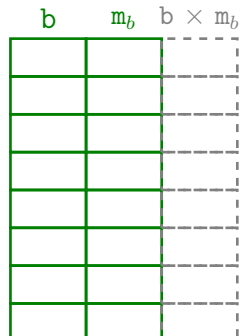




Evaluation Argument

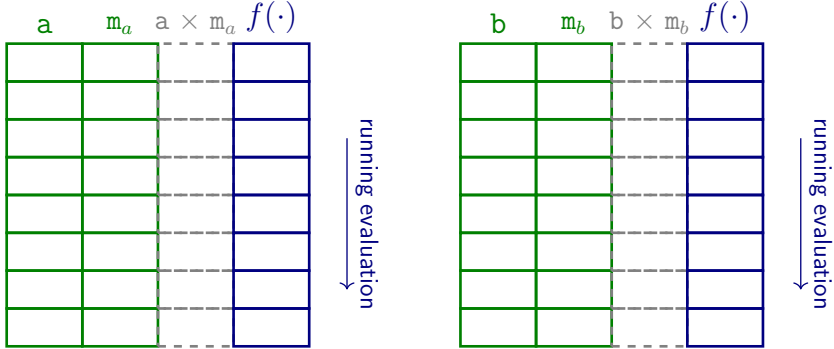
$$a[m_a] = b[m_b] \text{ for } m_a, m_b \subseteq \{0, \ldots, N-1\}$$





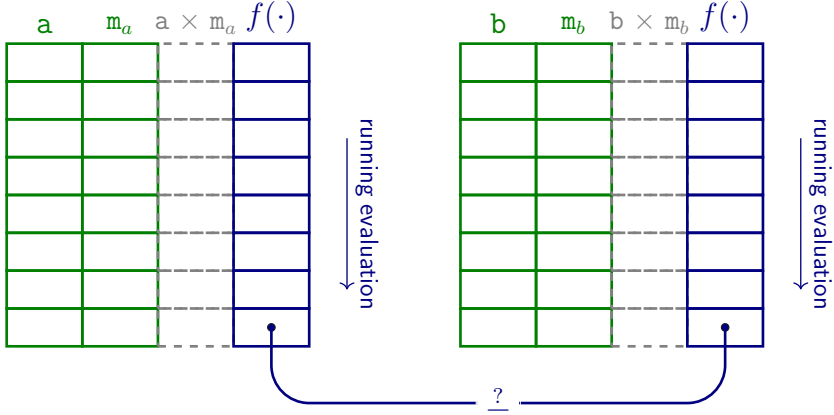
Evaluation Argument

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



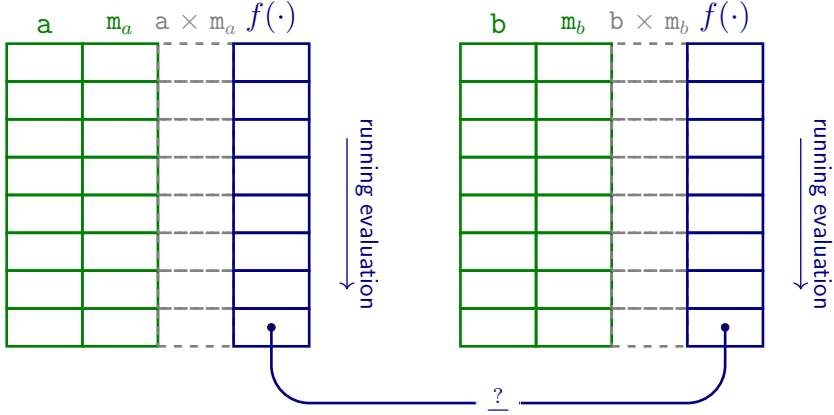
Evaluation Argument

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



Evaluation Argument

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

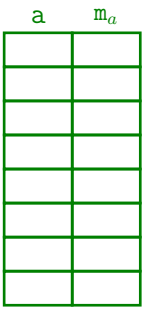


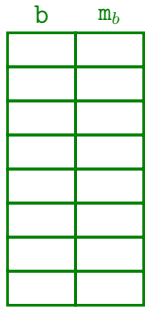
Soundness

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

Lookup Argument

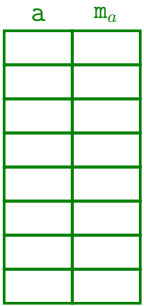
$a \equiv b$ as sets

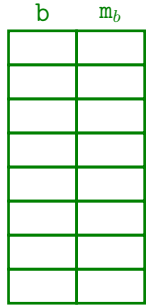




Lookup Argument

$a \equiv b$ as sets

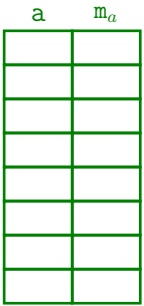


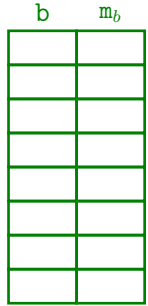


$$\log \tfrac{\mathtt{d}}{\mathtt{d}X}[f(X)] = \tfrac{f'(X)}{f(X)}$$

Lookup Argument

$a \equiv b$ as sets



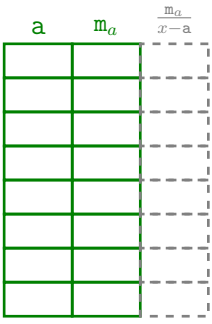


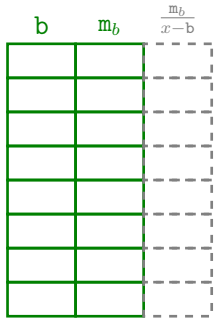
$$\log \tfrac{\mathtt{d}}{\mathtt{d}X}[f(X)] = \tfrac{f'(X)}{f(X)}$$

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

Lookup Argument

$a \equiv b$ as sets



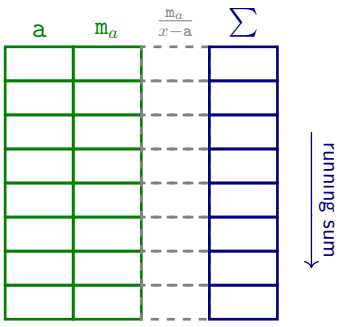


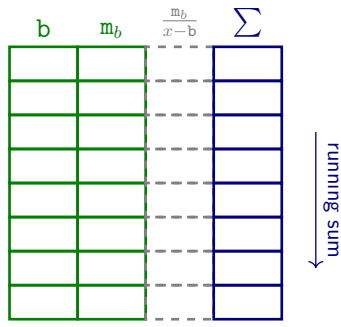
$$\log \tfrac{\mathtt{d}}{\mathtt{d}X}[f(X)] = \tfrac{f'(X)}{f(X)}$$

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

Lookup Argument

$a \equiv b$ as sets

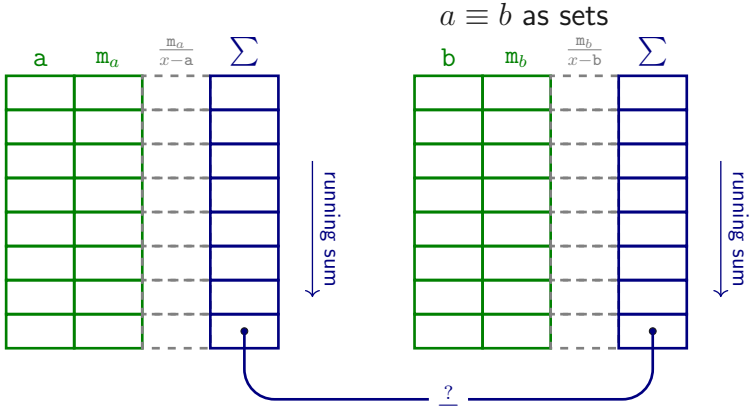




$$\log \tfrac{\mathtt{d}}{\mathtt{d}X}[f(X)] = \tfrac{f'(X)}{f(X)}$$

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

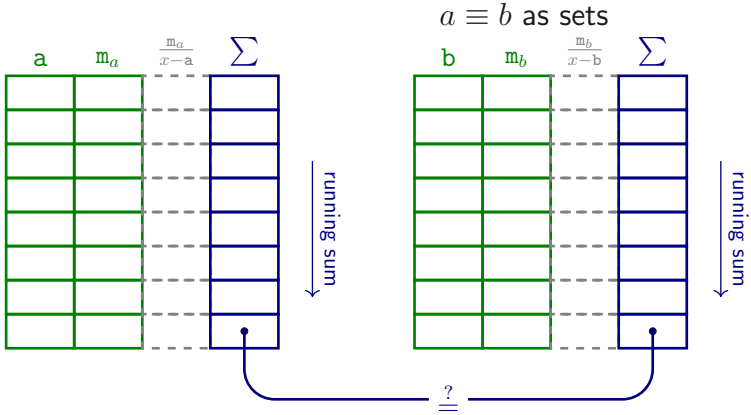
Lookup Argument



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

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

Lookup Argument



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

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

Soundness

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

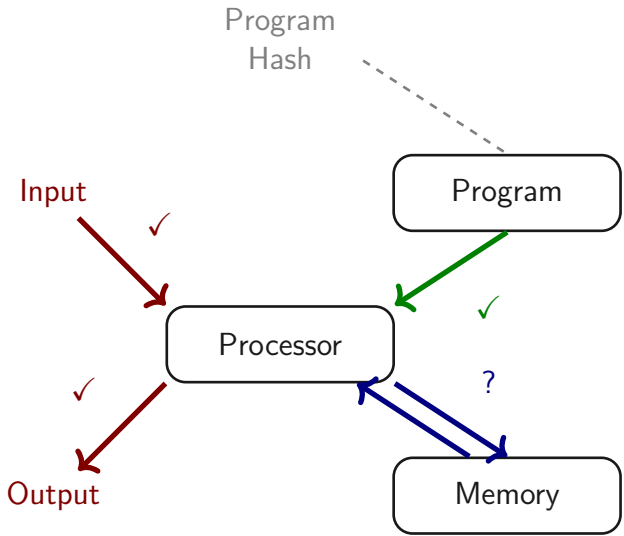
$$= \Pr_x[S_a \cdot \textstyle\prod_i (x-\mathtt{a}_i)^{\mathtt{m}_{a,i}} \cdot \prod_i (x-\mathtt{b}_i)^{\mathtt{m}_{b,i}} = S_b \cdot \prod_i (x-\mathtt{a}_i)^{\mathtt{m}_{a,i}} \cdot \prod_i (x-\mathtt{b}_i)^{\mathtt{m}_{b,i}} \mid a \not\equiv b]$$

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

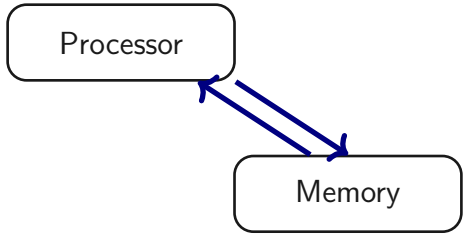
Table of Contents

Table of Contents

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

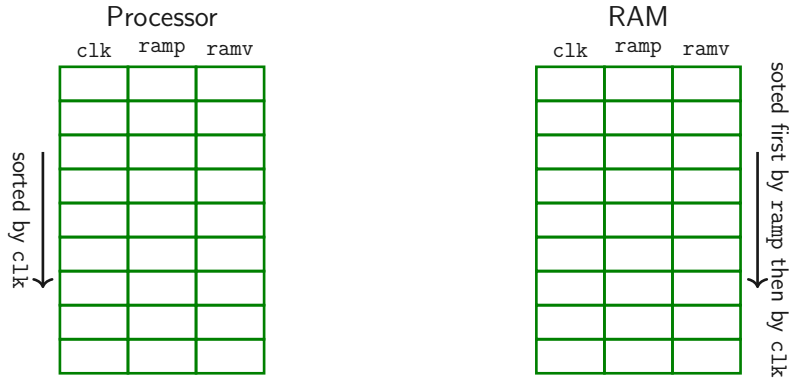
Processor

clk	ramp	ramv

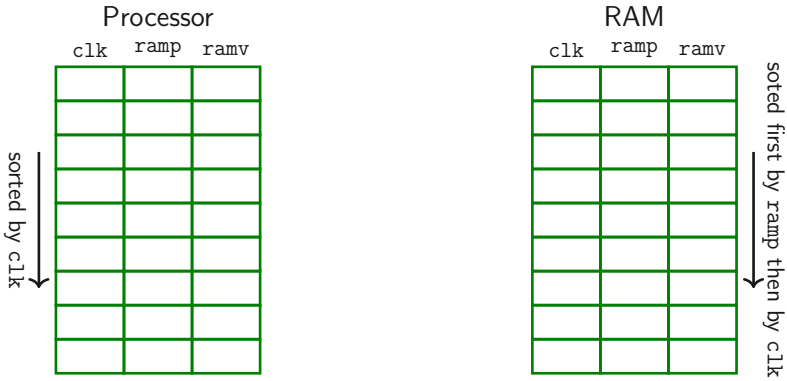
RAM

clk	ramp	ramv

Memory — Construction

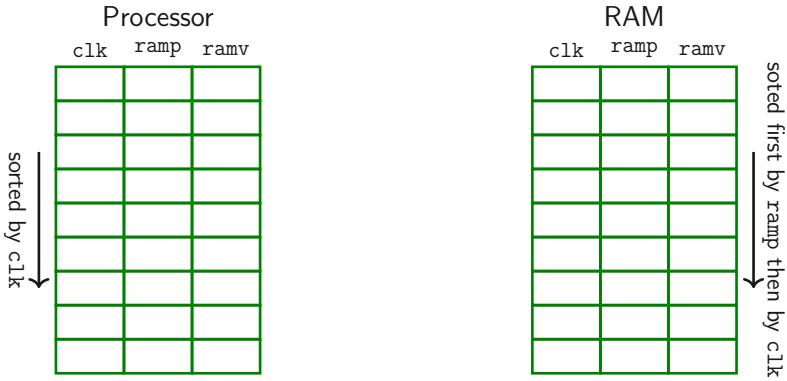


Memory — Construction



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

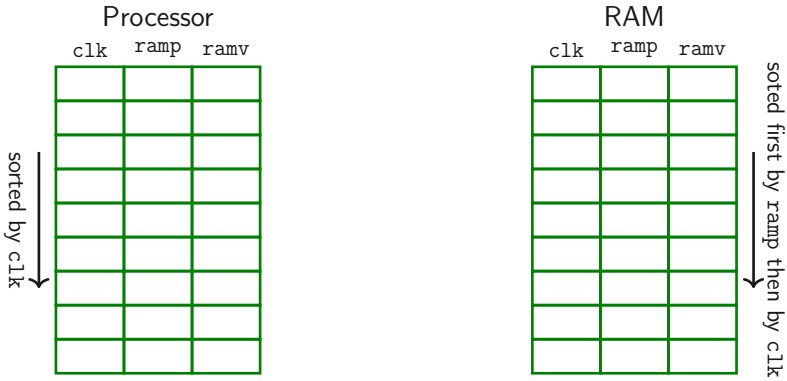
Memory — Construction



memory integrity $\Leftarrow$

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

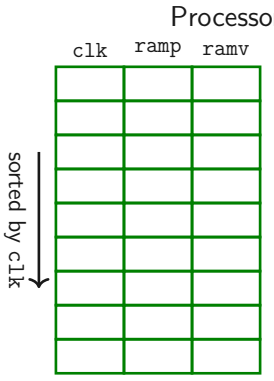
Memory — Construction

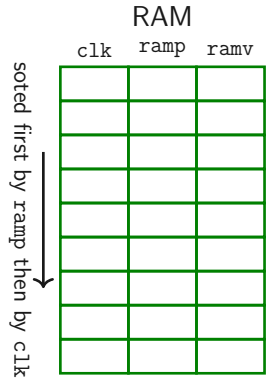


memory integrity $\Leftarrow$

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



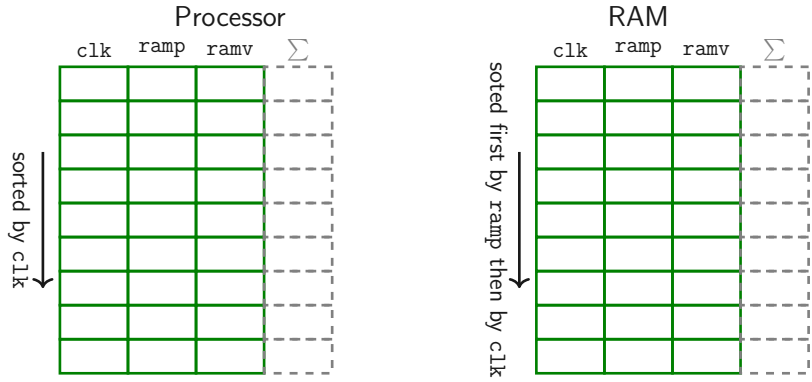


→ 1. same data, different order

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

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

Memory — Permutation

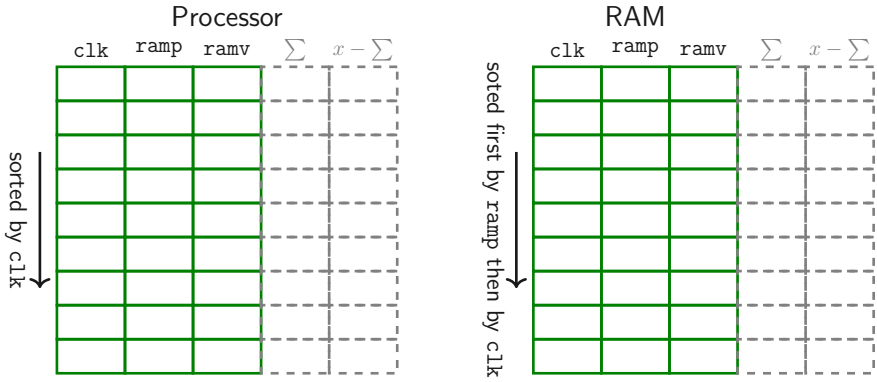


→ 1. same data, different order

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

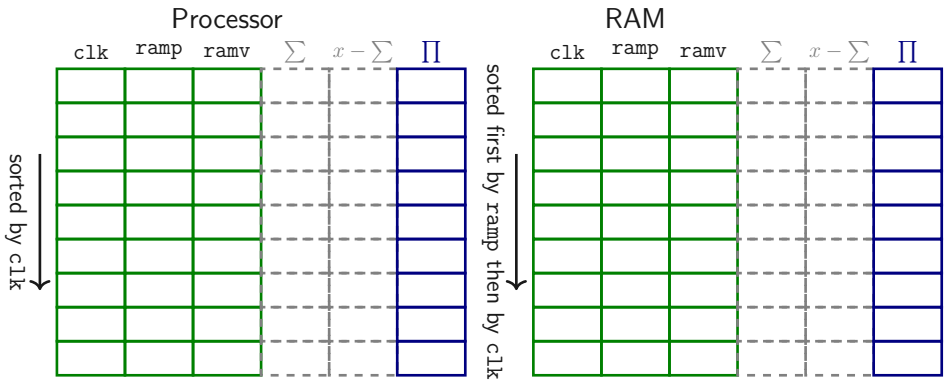
3. ~~correctly sorted by `ramp`~~ regions of constant `ramp` are *contiguous*

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

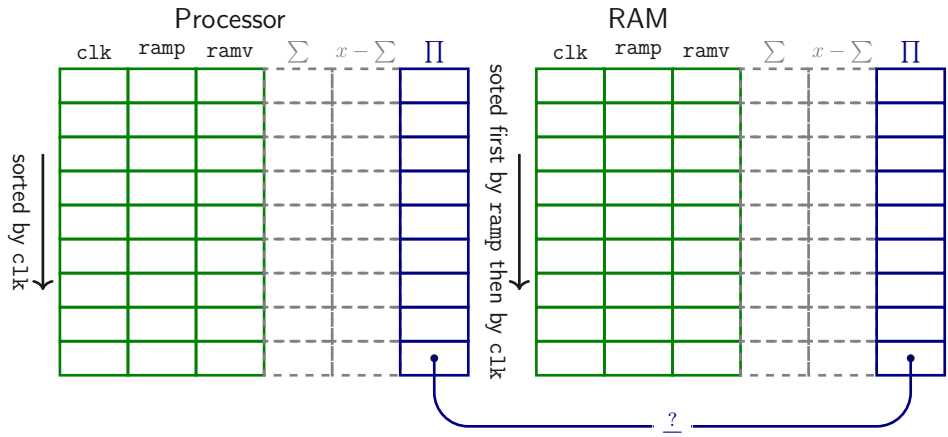


→ 1. same data, different order

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

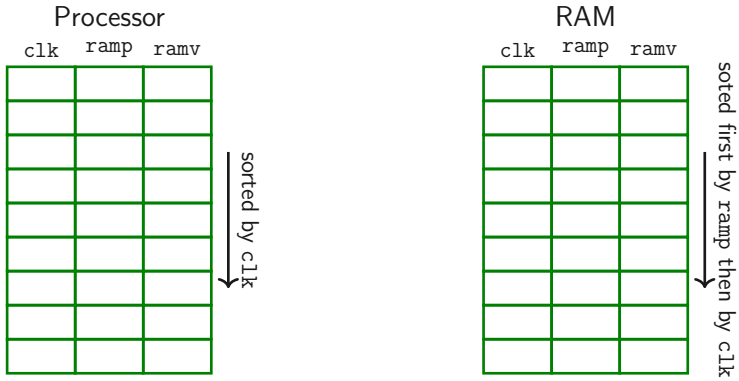
3. ~~correctly sorted by `ramp`~~ regions of constant `ramp` are *contiguous*

Memory — Permutation



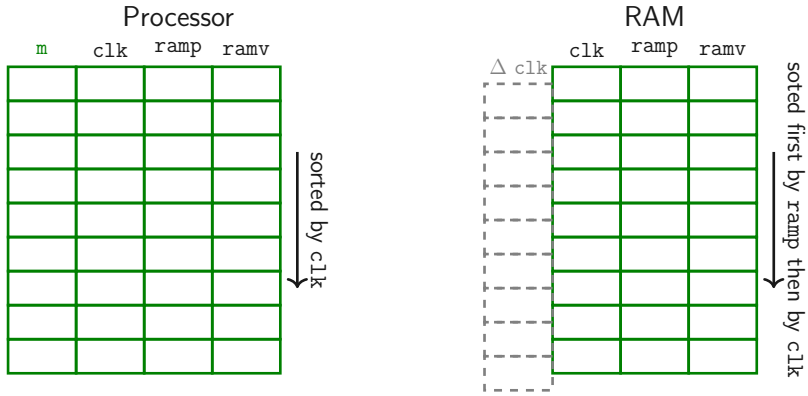
1. same data, different order
2. within regions of constant `ramp`, correctly sorted by `clk`
3. ~~correctly sorted by `ramp`~~ regions of constant `ramp` are *contiguous*

Memory — Lookup



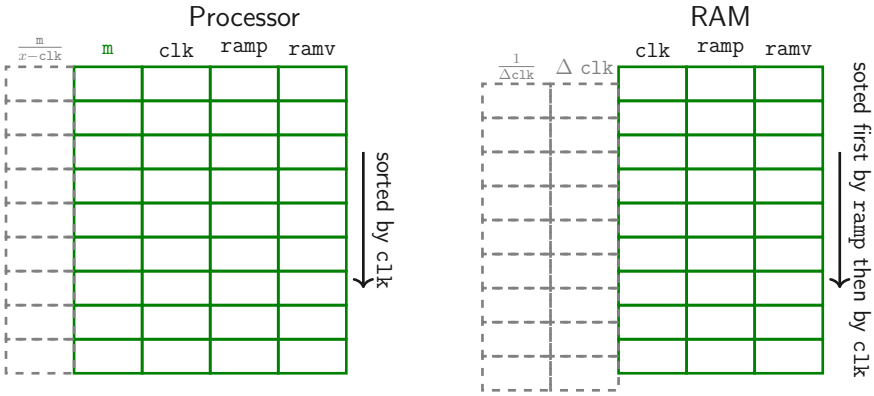
1. same data, different order
2. within regions of constant `ramp`, correctly sorted by `clk`
3. ~~correctly sorted by ramp~~ regions of constant `ramp` are *contiguous*

Memory — Lookup



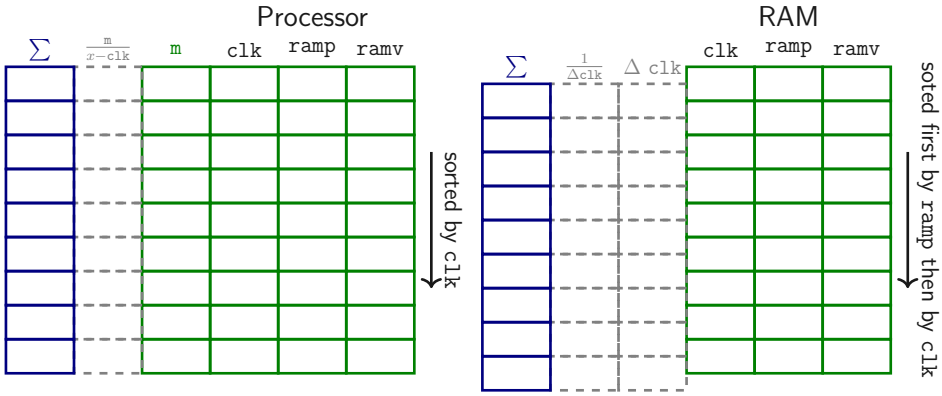
1. same data, different order
2. within regions of constant `ramp`, correctly sorted by `clk`
3. ~~correctly sorted by ramp~~ regions of constant `ramp` are *contiguous*

Memory — Lookup



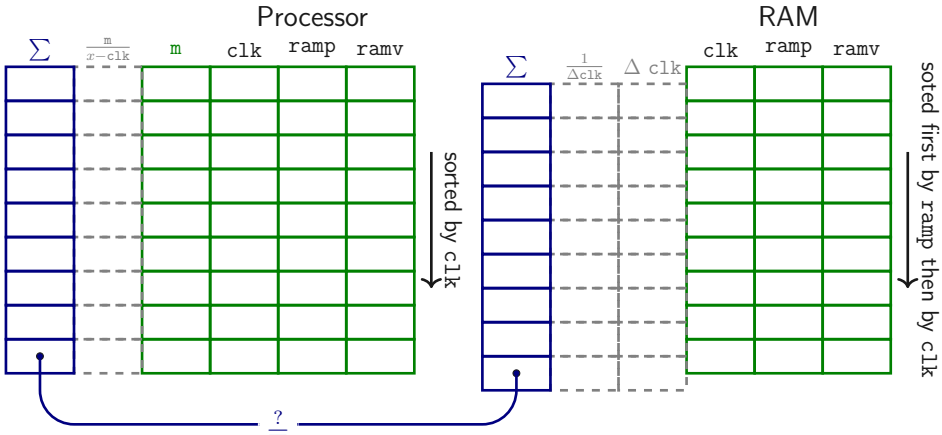
1. same data, different order
2. within regions of constant `ramp`, correctly sorted by `clk`
3. ~~correctly sorted by ramp~~ regions of constant `ramp` are *contiguous*

Memory — Lookup



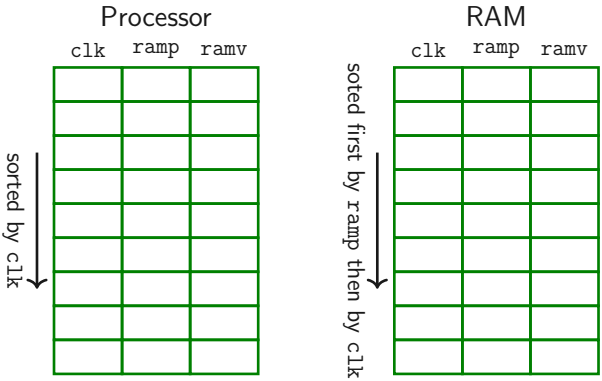
1. same data, different order
2. within regions of constant `ramp`, correctly sorted by `clk`
3. ~~correctly sorted by ramp~~ regions of constant `ramp` are *contiguous*

Memory — Lookup



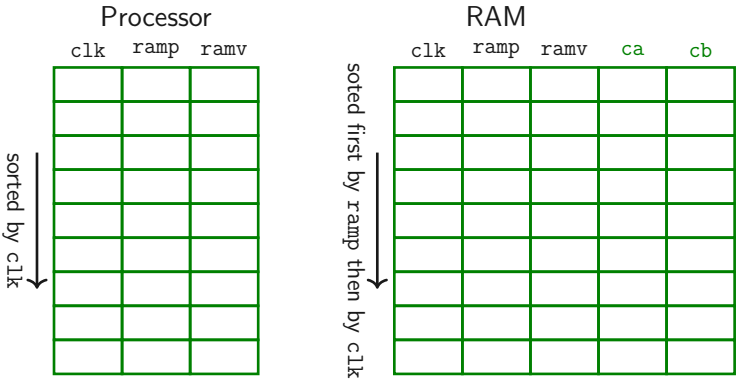
1. same data, different order
2. within regions of constant `ramp`, correctly sorted by `clk`
3. ~~correctly sorted by ramp~~ regions of constant `ramp` are *contiguous*

Memory — Contiguity



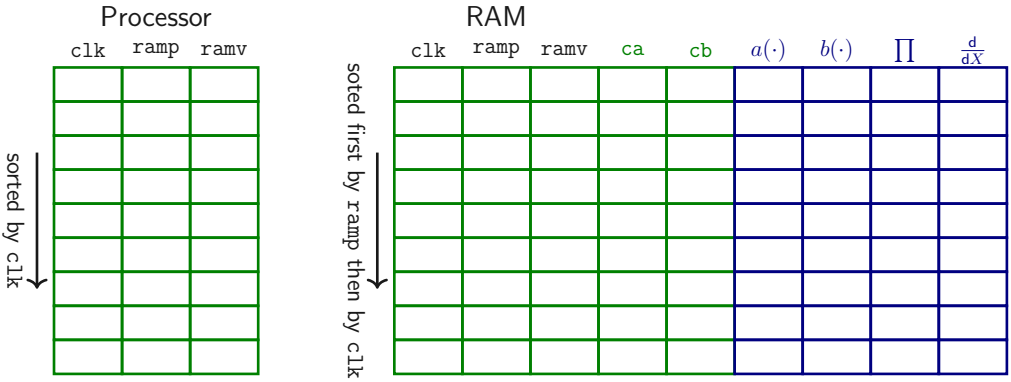
1. same data, different order
2. within regions of constant `ramp`, correctly sorted by `clk`
3. ~~correctly sorted by ramp~~ regions of constant `ramp` are *contiguous*

Memory — Contiguity



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*

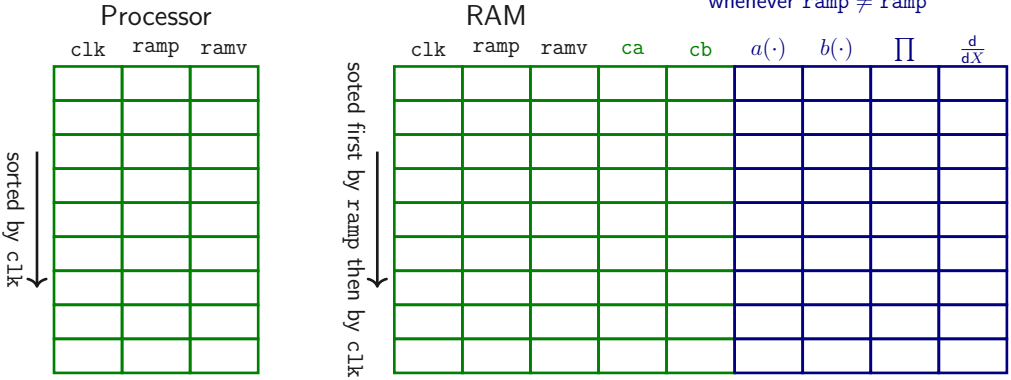
Memory — Contiguity



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*

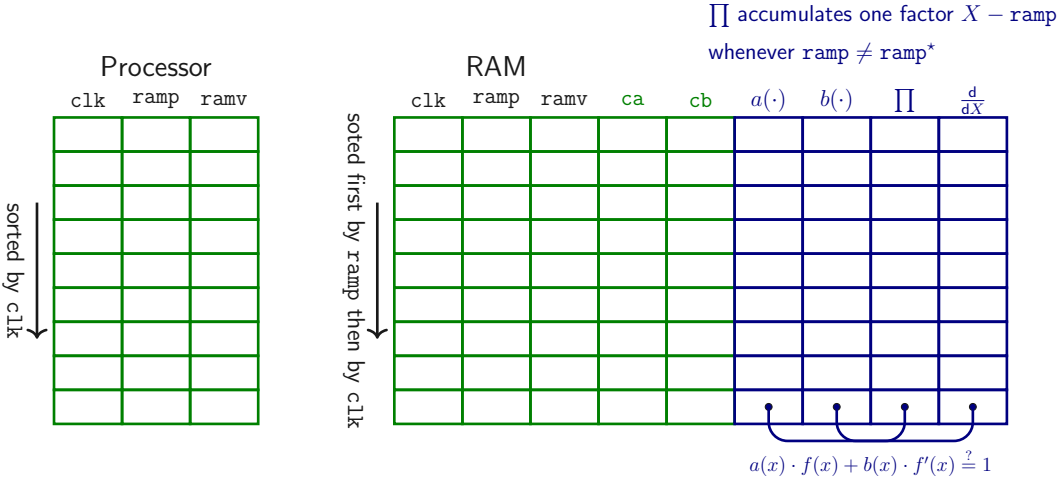
Memory — Contiguity

$\prod$ accumulates one factor $X - \texttt{ramp}$ whenever $\texttt{ramp} \neq \texttt{ramp}^\star$



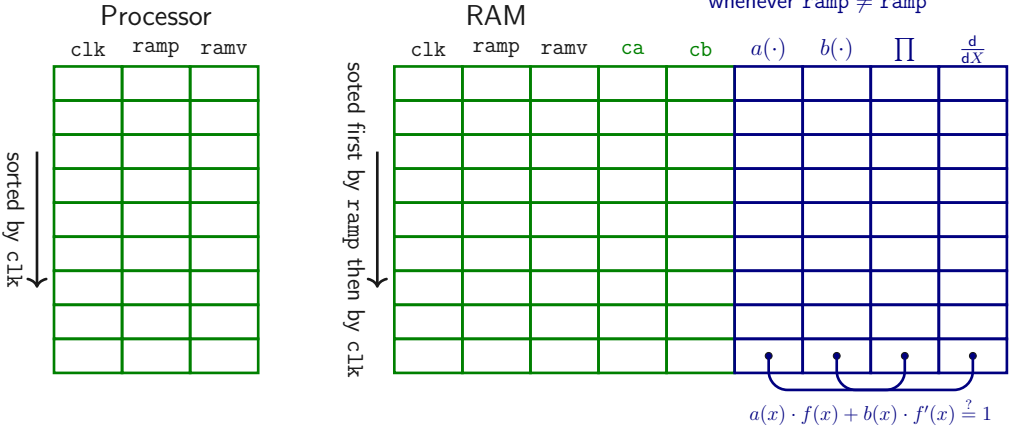
1. same data, different order
2. within regions of constant `ramp`, correctly sorted by `clk`
3. ~~correctly sorted by ramp~~ regions of constant `ramp` are *contiguous*

Memory — Contiguity



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*

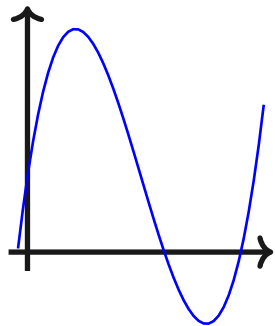
Table of Contents

Table of Contents

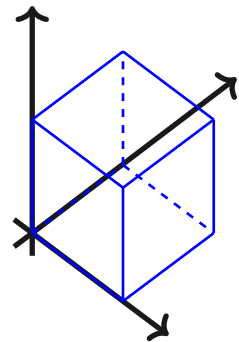
Table of Contents

Univariate versus Multilinear

Univariate

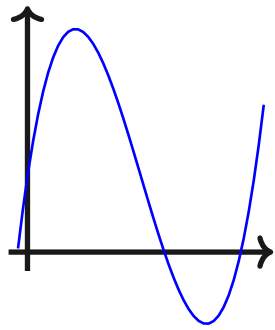


Multilinear



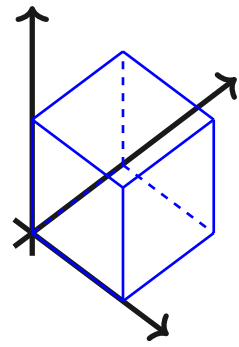
Univariate versus Multilinear

Univariate



DEEP-ALI

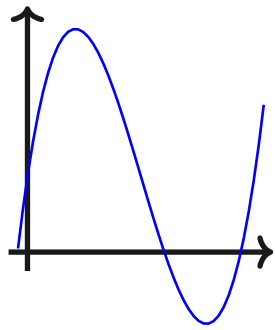
Multilinear



GKR

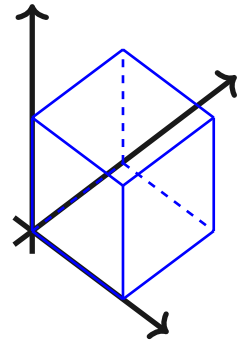
Univariate versus Multilinear

Univariate



DEEP-ALI

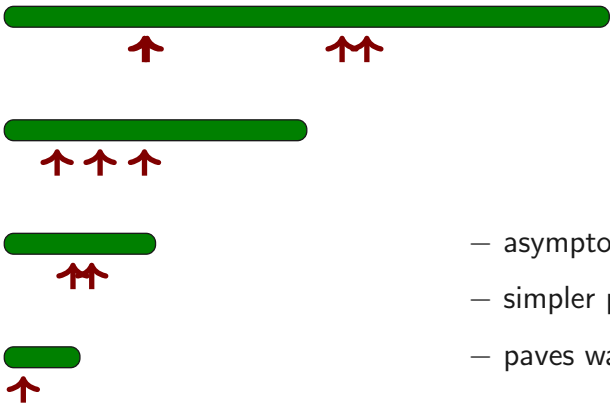
Multilinear



GKR

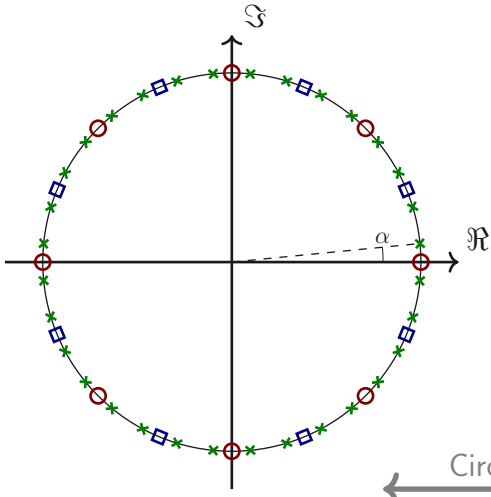
WHIR / GKR-logup

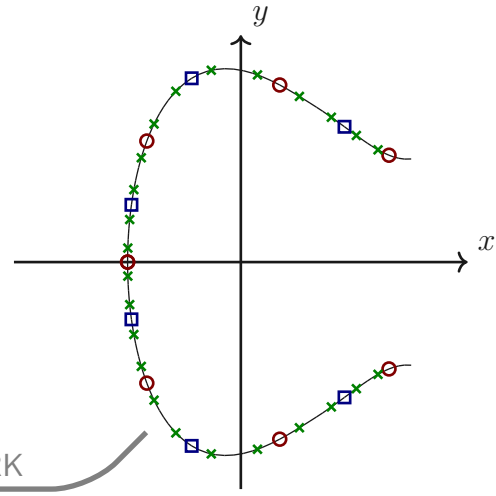
STIR



- asymptotically and concretely better than FRI
- simpler proof of soundness
- paves way for aggregation

ECFFT





CircleSTARK

Structured Fields

Arbitrary Fields

Table of Contents

Table of Contents

Advanced zk-STARKs

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

neptune

Triton VM

https://neptune.cash/

https://triton-vm.org/

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