ETH zürich

Marco Ferroni's Portfolio

Profile

I'm a 23 years old passionate student from a small town near Ravenna (IT).

- **Education**
  - **Master Degree (PRESENT):** Electrical Engineering at ETH Zurich (current GPA 5.55/6).
  - **Bachelor:** Electronics Engineering at Politecnico di Milano (110L/110 with 29.48/30 final GPA).
- **Scholarships**
  - **Best freshmen award:** 1000 euros scholarship awarded to the best freshmen of Politecnico di Milano (top 2.5%).
  - **Fee exemption:** in my Bachelor due to my high GPA (>29/30) I was granted a fee exemption.
- **Job experience**
  - **Circuit Design Intern at SynSense:** currently employed for six months as part of the IC design team, I am learning the full stack of asynchronous IC design.
  - **Academic guest at EMPA:** I've performed quantum simulations on CNTs

Me

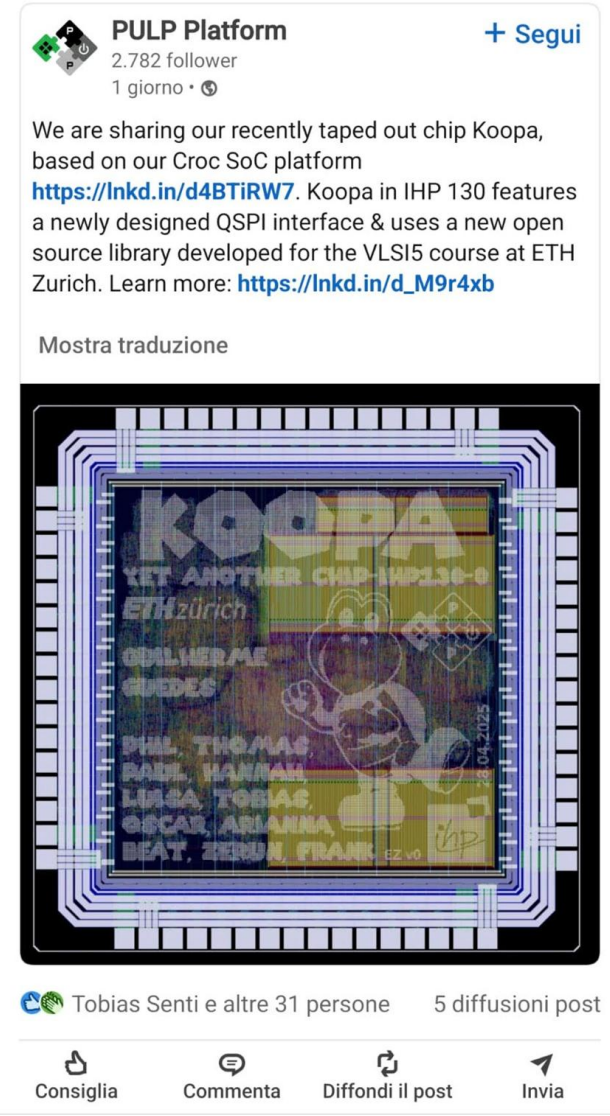
Me and my cat!

ETH zürich

Expanding the ez130 library

Expanding the ez130 library -Overview

- **Context**
  - **VLSI5:** first time the course was offered. Very limited slots.
  - **Scope:** expand an existing open source 130nm library of PULP
- **Goal**
  - Design 6 new standard cells
- **Workflow**
  - **Schematic design:** design exploration, logic restructuring
  - **Stick diagram:** share diffusions, optimize connections
  - **Physical design:** Cadence
  - **Characterization:** PEX, generate .lef and .gds
  - **Post-layout verification:** implemented a small design that used them



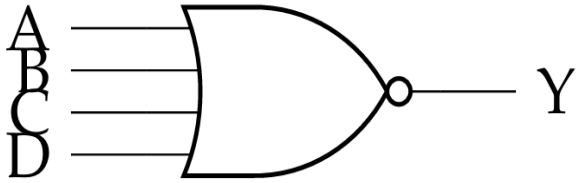
Our cells are already getting used

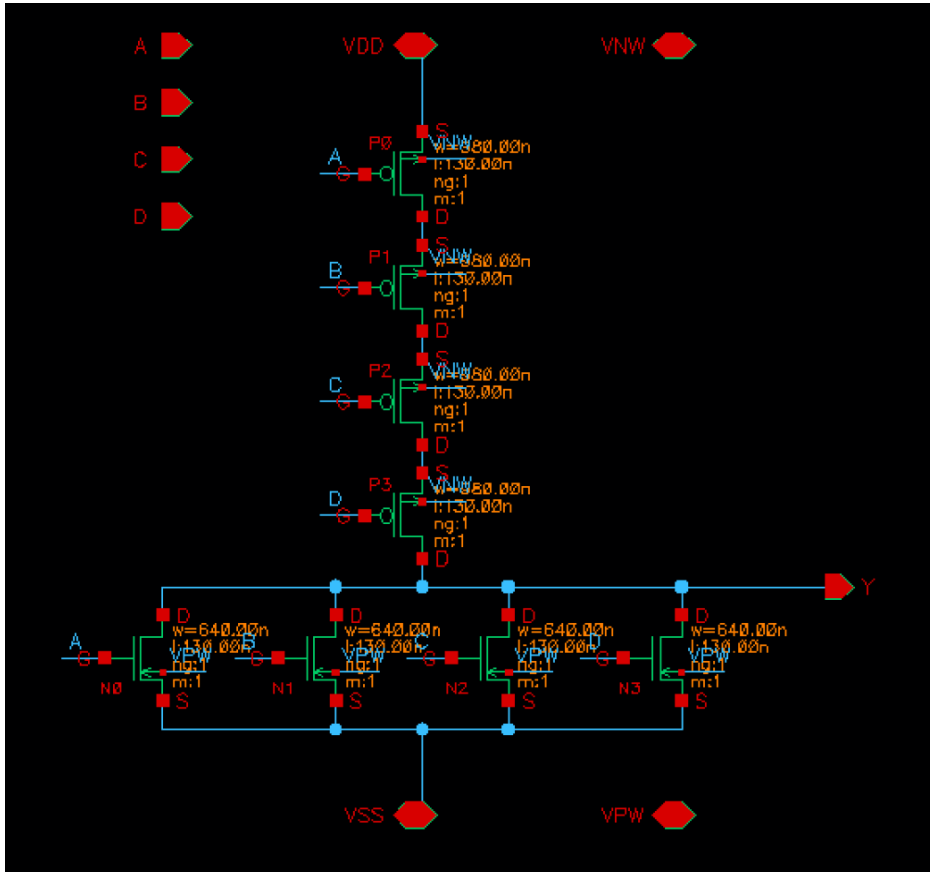
Expanding the ez130 library -NOR4X2 / X1

Logic function: $\overline{A + B + C + D}$

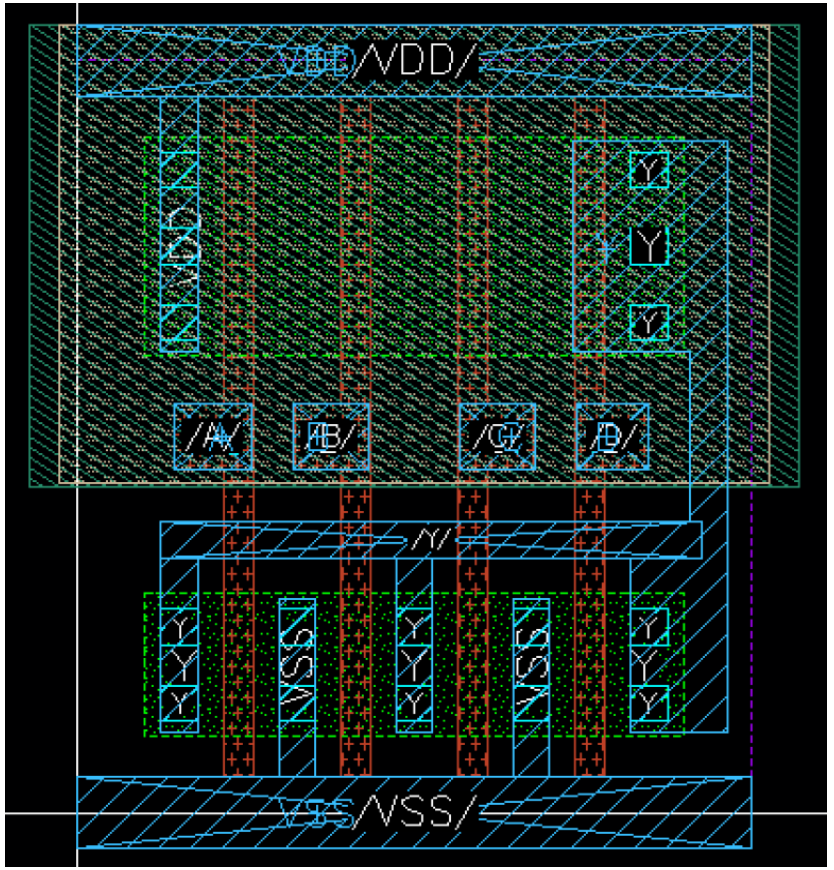
N° of transistors: 8

Area: 9.8784 µm²





NOR4X2 schematic



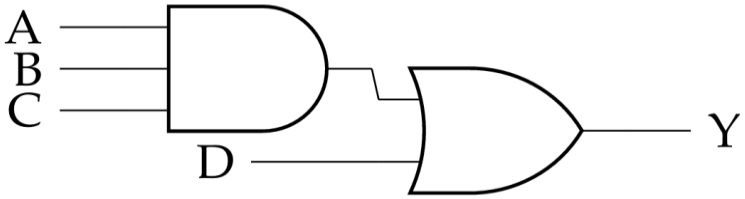
NOR4X2 layout

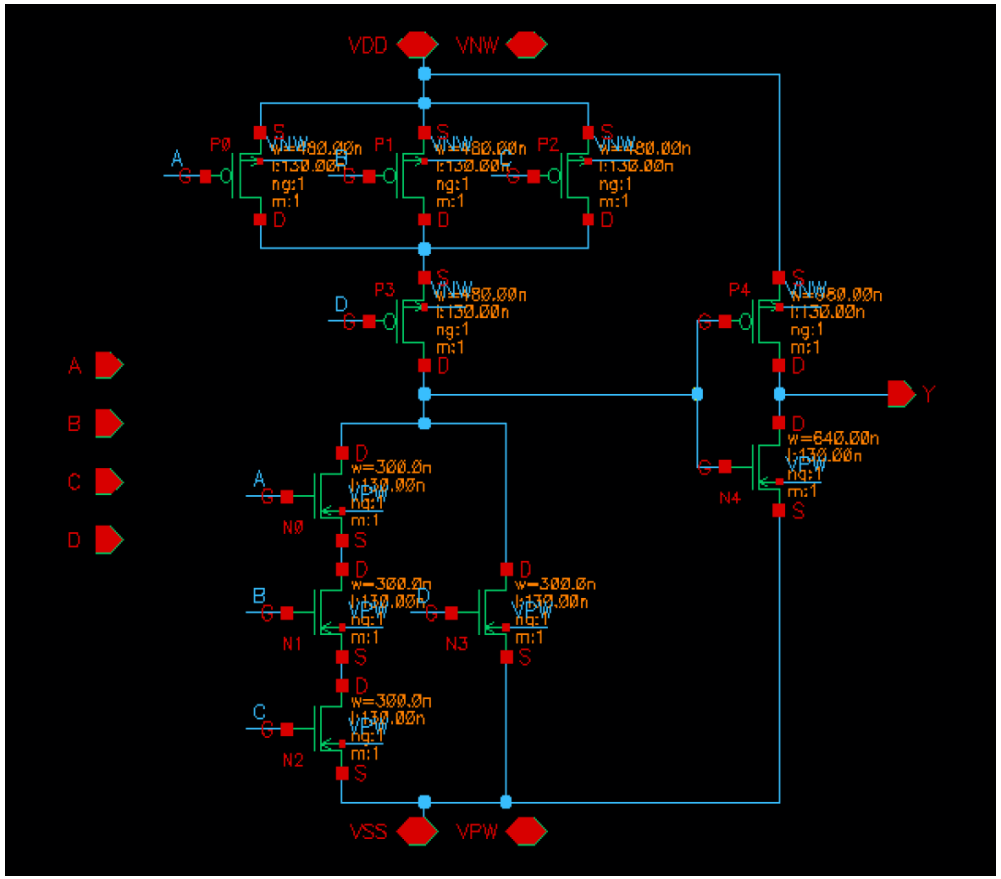
Expanding the ez130 library –AO31X2 / X1

Logic function: $A \cdot B \cdot C + D$

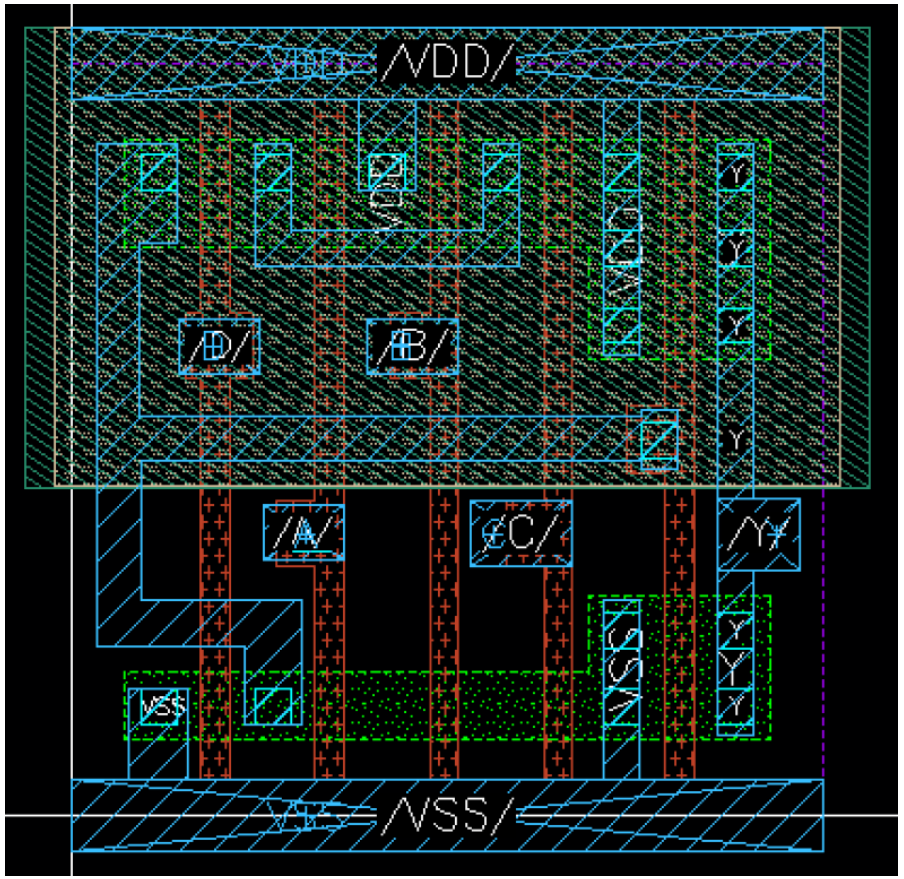
N° of transistors: 10

Area: 11.2896 µm²





AO31X2 schematic



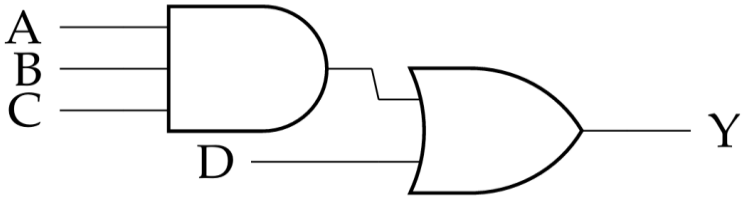
AO31X2 layout

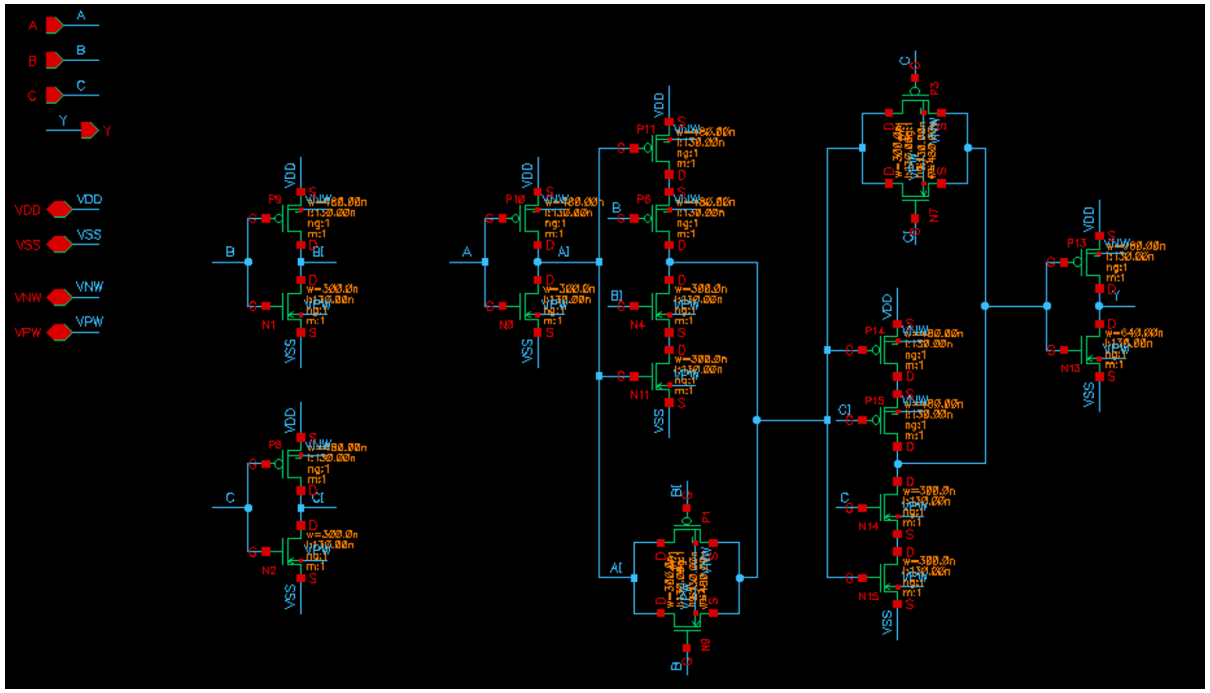
Expanding the ez130 library –XNOR3X2 / X1

Logic function: $\overline{A \oplus B \oplus C}$

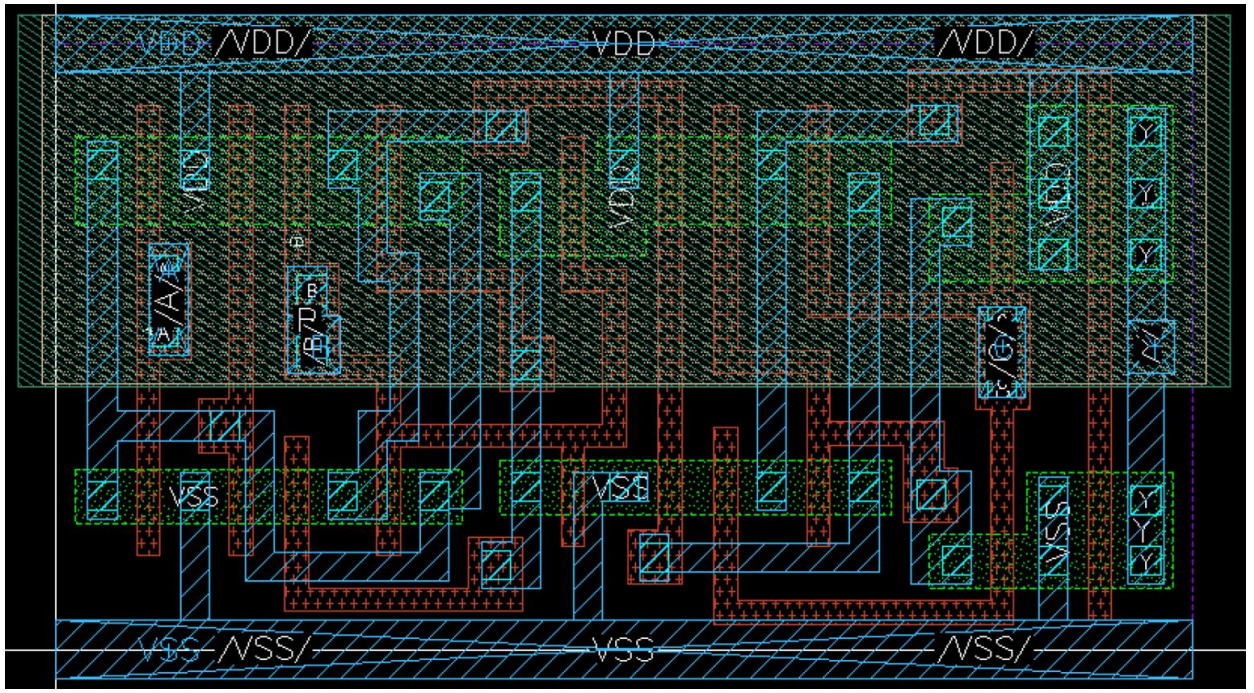
N° of transistors: 20

Area: 21.168 µm²



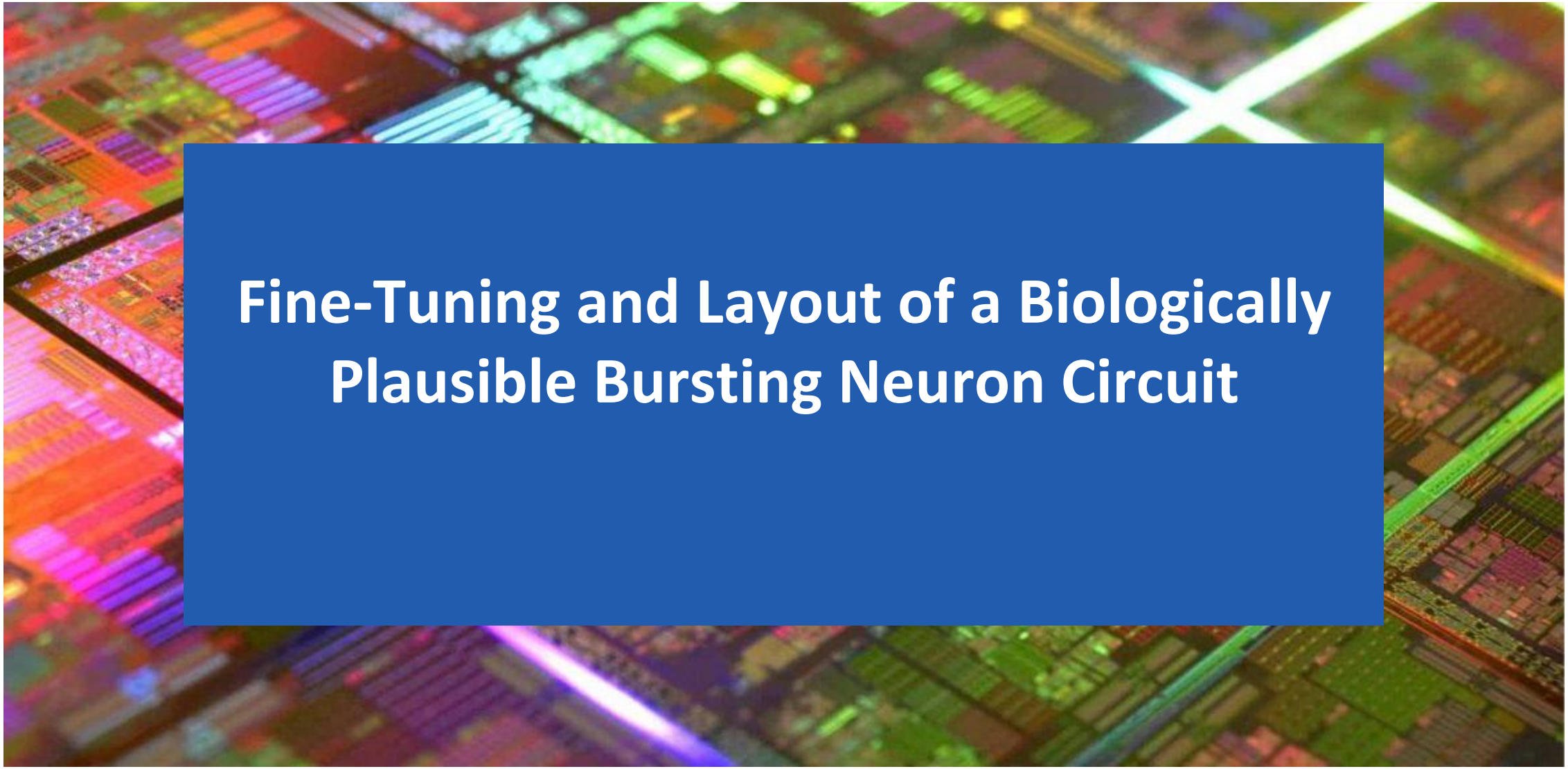


XNOR3X2 schematic



XNOR3X2 layout

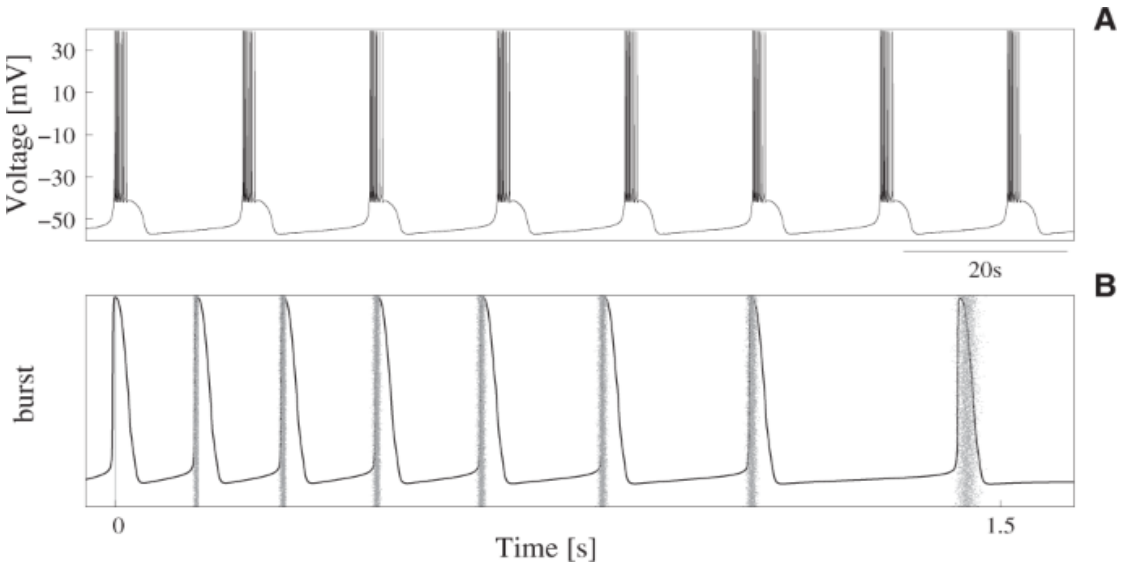
ETH zürich



Fine-Tuning and Layout of a Biologically Plausible Bursting Neuron Circuit

Bursting Neuron -Overview

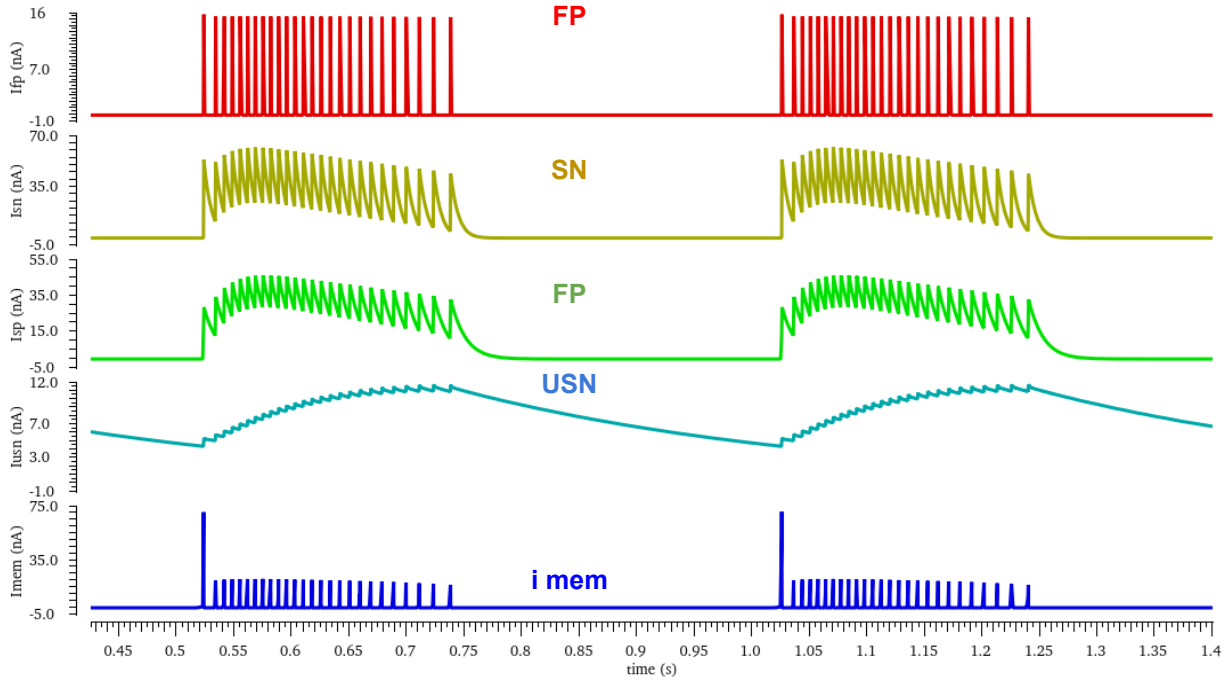
- **Context**
  - **Neuromorphic II:** analog IC design
  - **Scope:** learn of to design the layout of analog circuit
- **Goal**
  - Obtain a functioning layout of the given circuit
- **Workflow**
  - **Tuning:** find the right parameters and transistor sizes
  - **Simulations:** make sure the spiking activity is happening
  - **Physical implementation:** layout of the circuit
  - **Post-layout simulations:** Montecarlo simulations



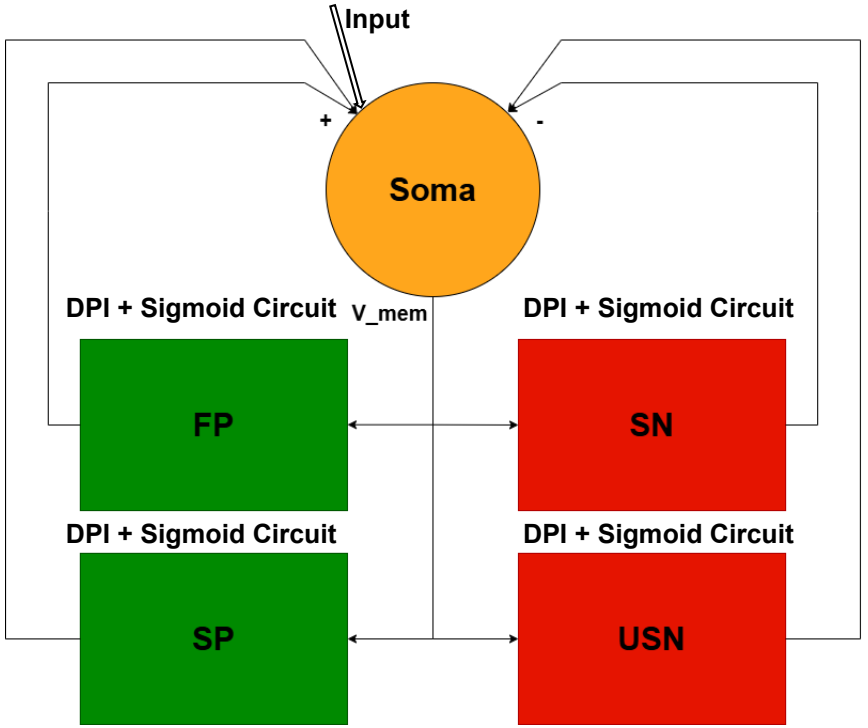
Bursting activity in biological neurons

Bursting Neuron –Circuit overview

- **Input: DC current**
- **Output: bursts in membrane current**
- **4 Feedback mechanisms (DPI + Sigmoid):**
  - FP (fast positive), SN (slow negative), SP (slow positive), USN (ultra slow negative)

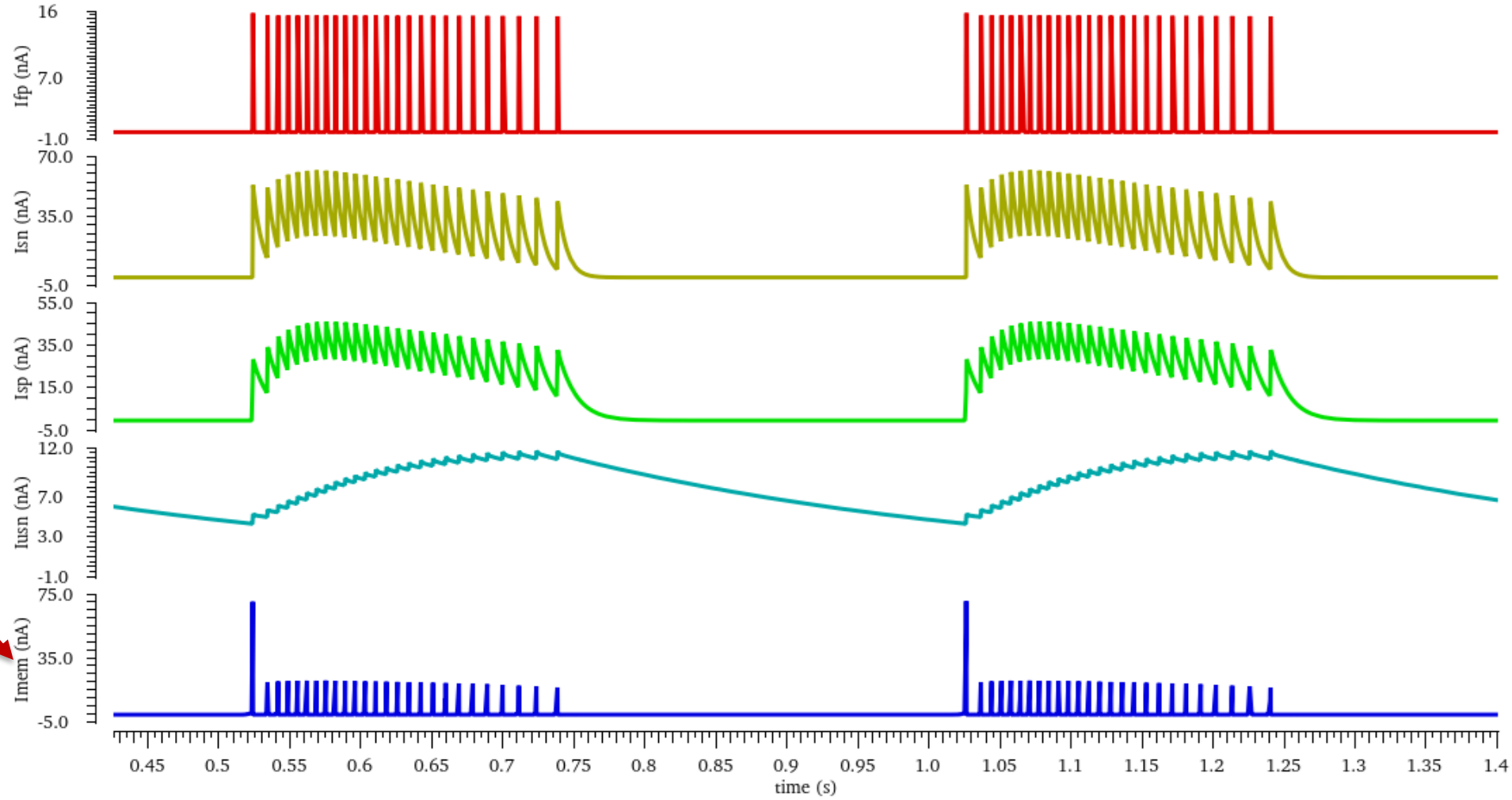


Bursting activity in our circuit



Model used

Bursting Neuron –Bursting activity

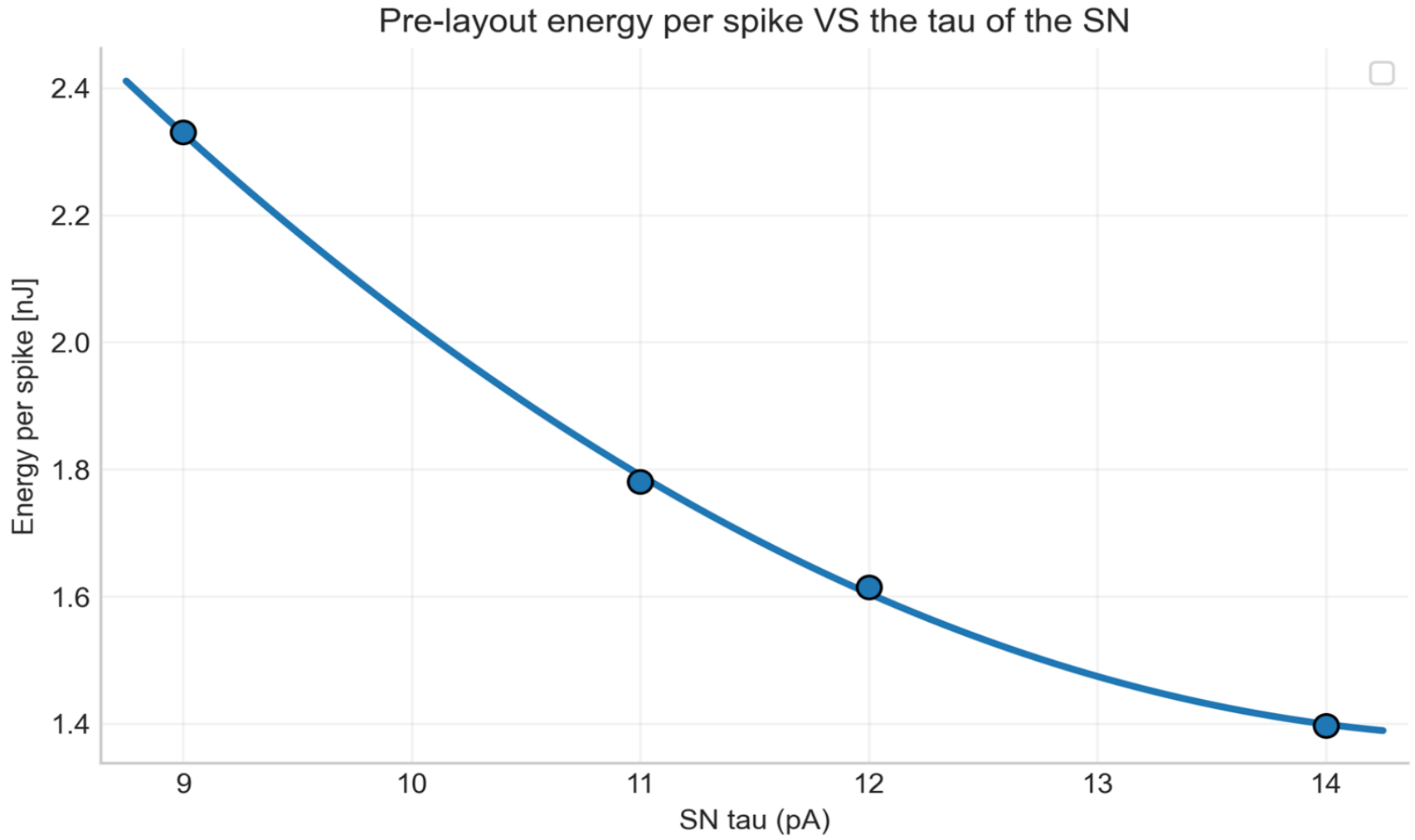


Membrane current

Bursting activity in our circuit

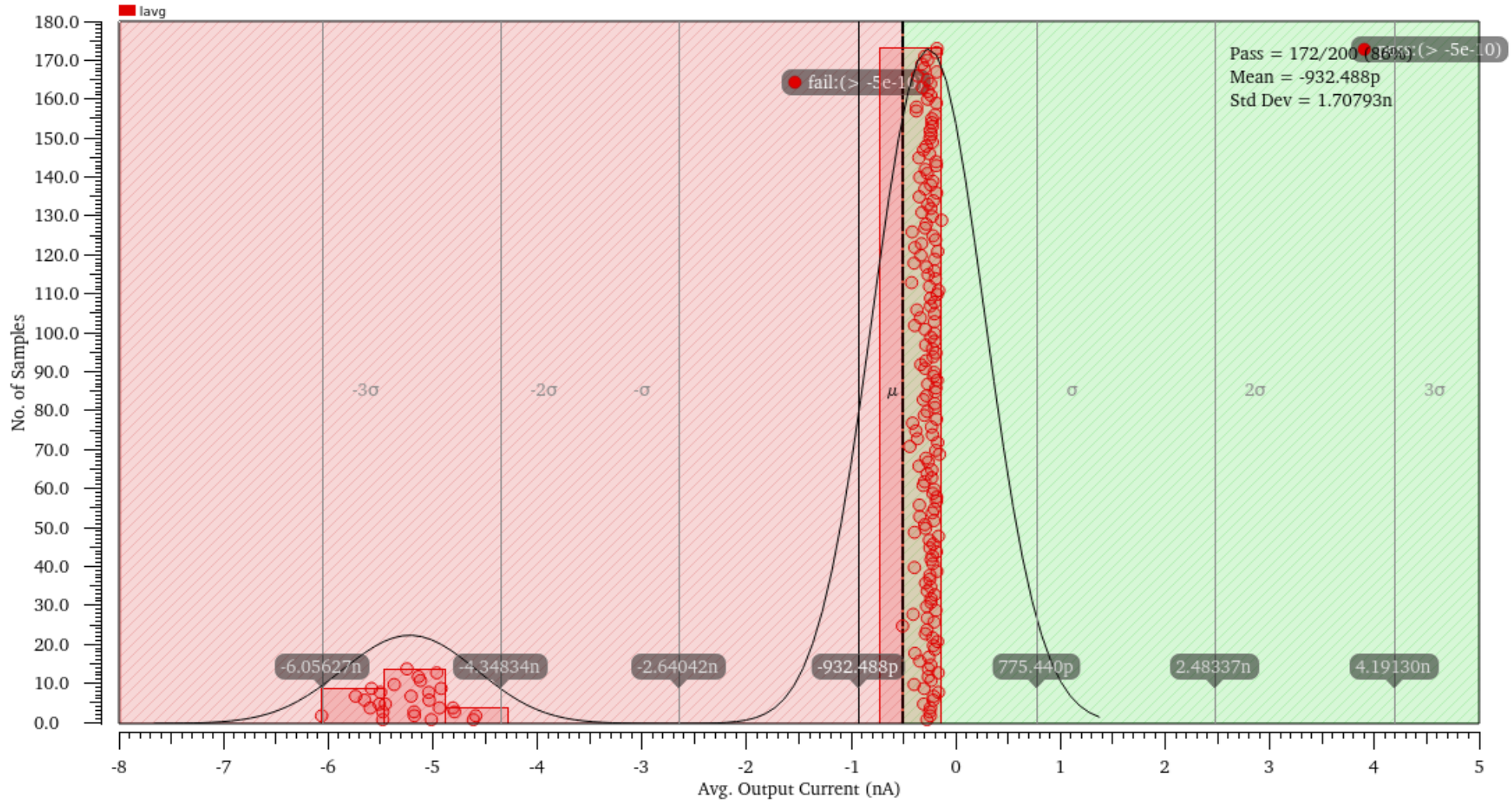
Bursting Neuron –Power analysis Pre-layout

The energy consumption for each spike decreases with the tau of the SN since it makes the spikes more narrow.



Bursting Neuron –Montecarlo Pre-layout

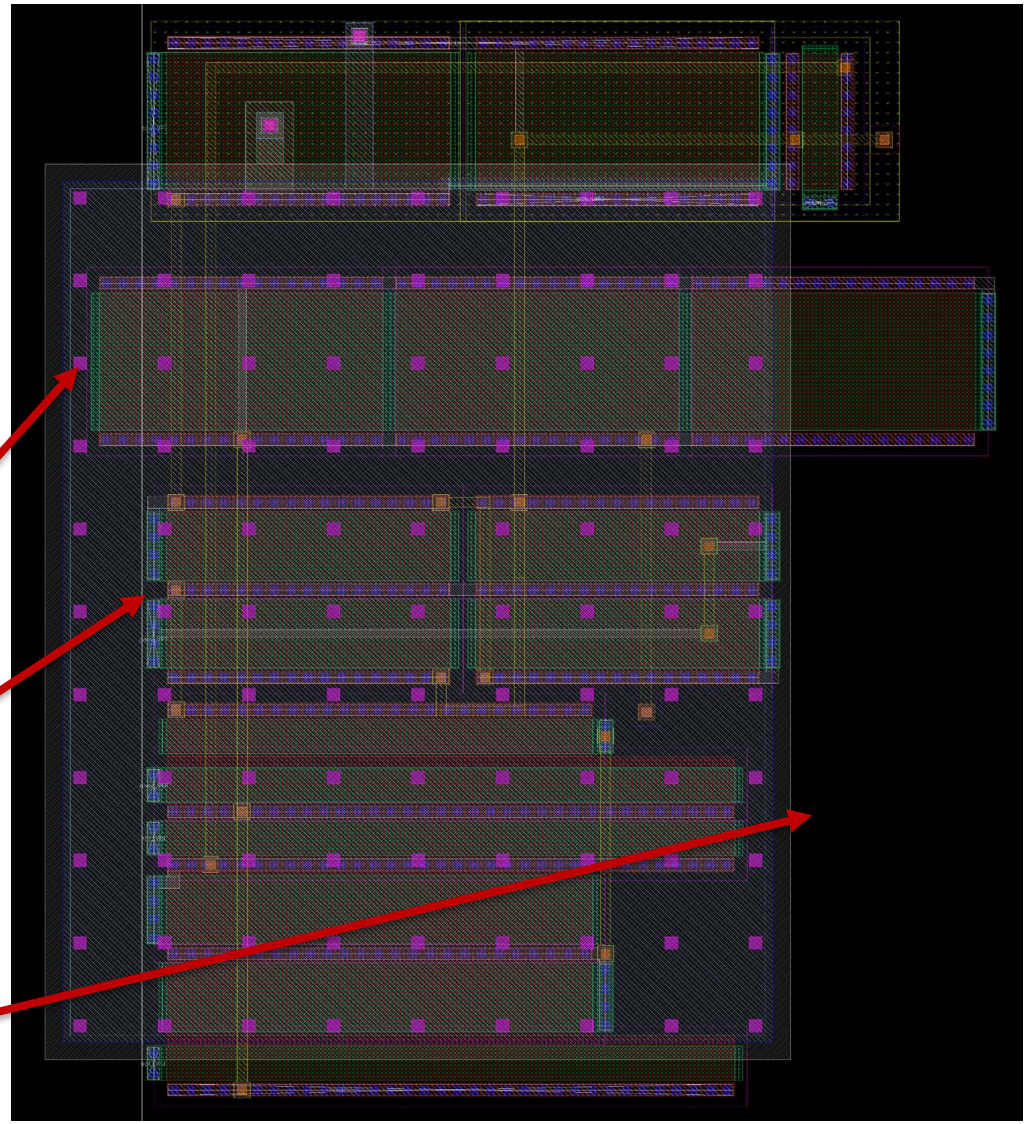
Passing samples: 86%



Montecarlo simulation results for mismatch + process variations

Bursting Neuron –Feedback layout

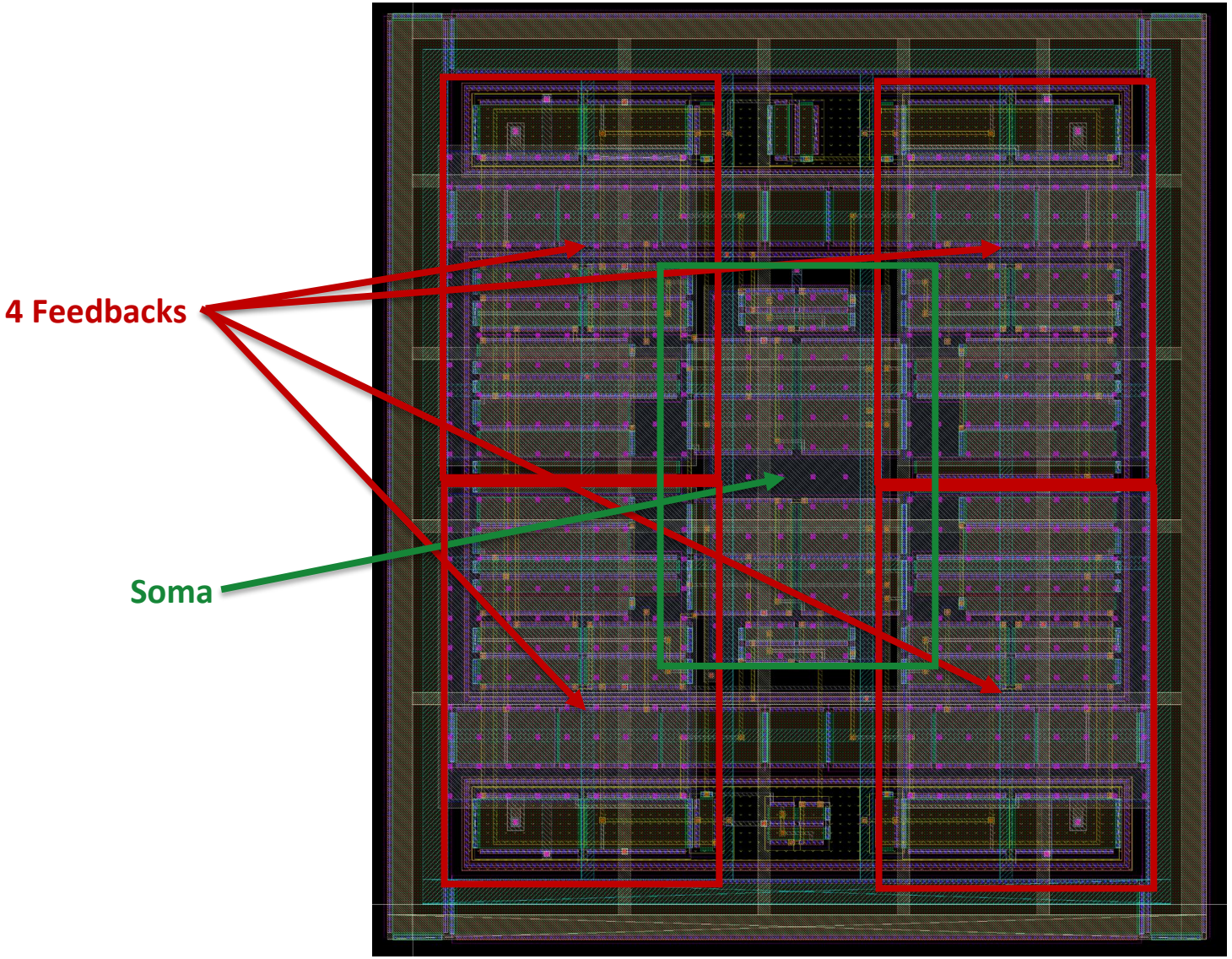
- **4 copies of the feedback block**
- **Features:**
  - Centroid arrangement for current mirrors
  - Stripe of dummy transistor
  - Lateral room for the Soma's block



Layout of one of the feedback mechanisms

Bursting Neuron –Final layout

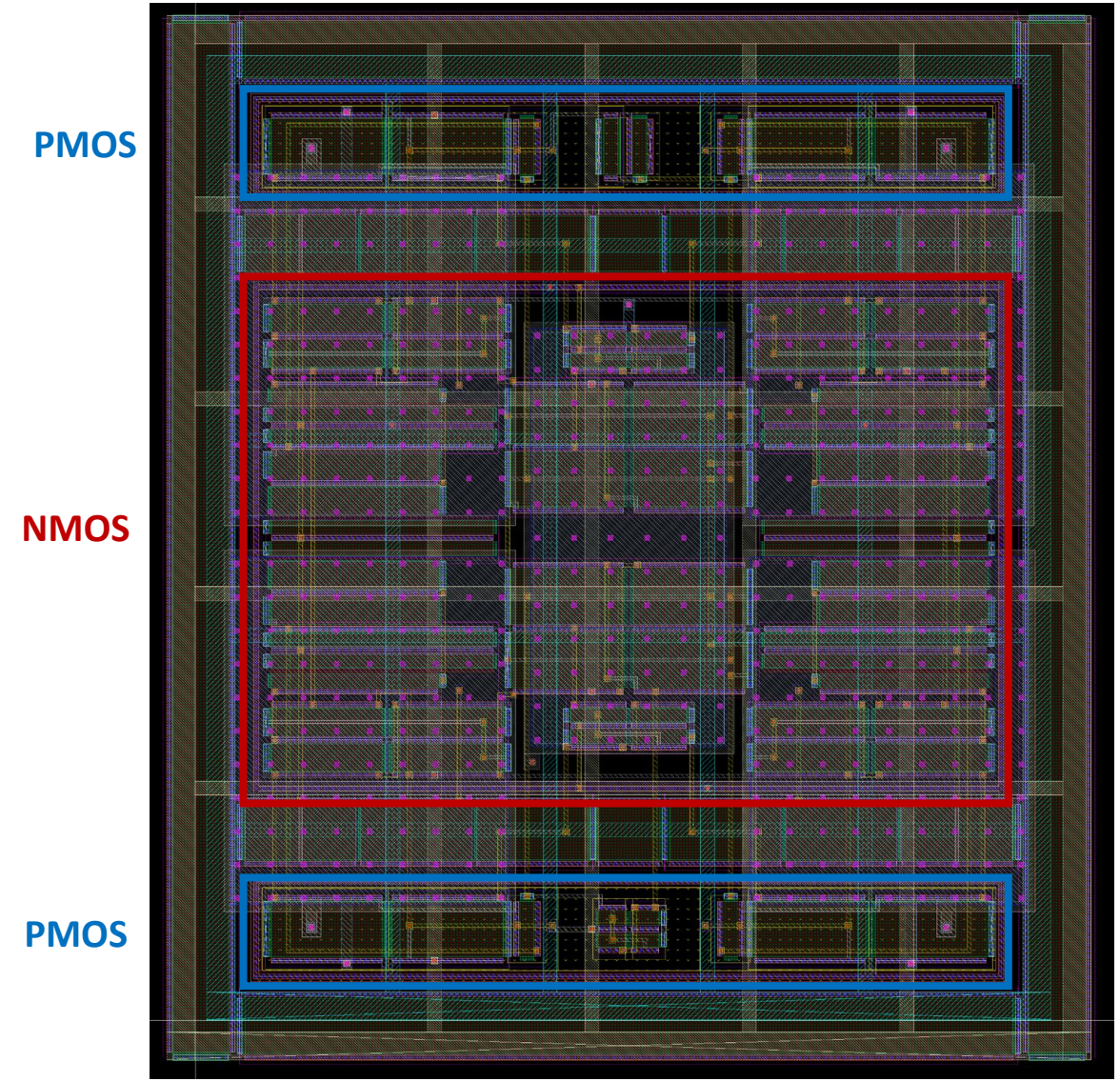
- **Structure**
  - 4 feedback blocks
  - Soma in the center
  - Two rows of PMOS
  - Central block of NMOS
- **Guard rings:**
  - P guard rings around NMOS
  - N guard rings around PMOS
- **Dummy transistors**
  - To separate PMOS and NMOS
  - All around
- **Power grid**
- **Area: 75.66 × 67.02 µm²**



Layout of the final neuron

Bursting Neuron –Final layout

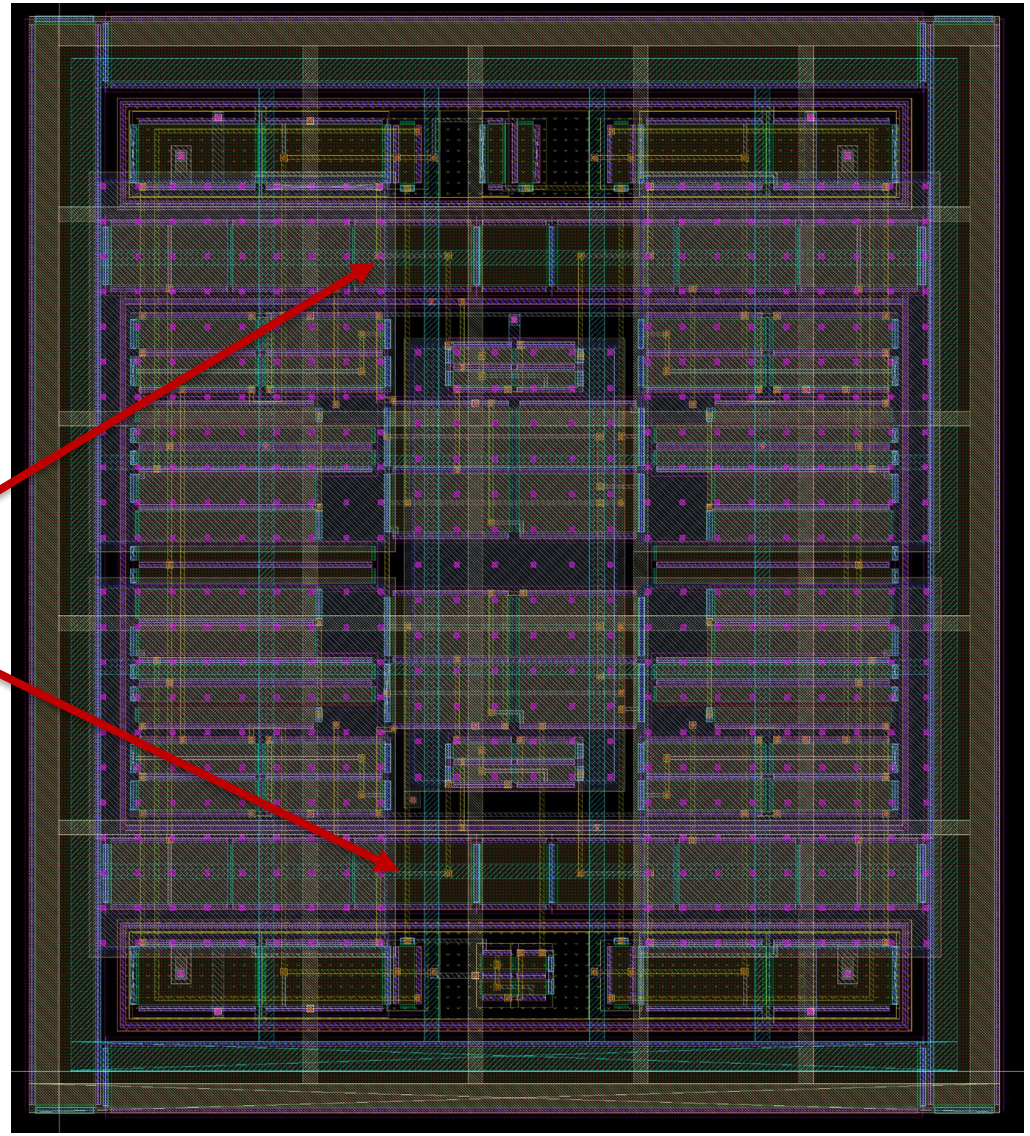
- **Structure**
  - 4 feedback blocks
  - Soma in the center
  - Two rows of PMOS
  - Central block of NMOS
- **Guard rings:**
  - P guard rings around NMOS
  - N guard rings around PMOS
- **Dummy transistors**
  - To separate PMOS and NMOS
  - All around
- **Power grid**
- **Area: 75.66 × 67.02 µm²**



Layout of the final neuron

Bursting Neuron –Final layout

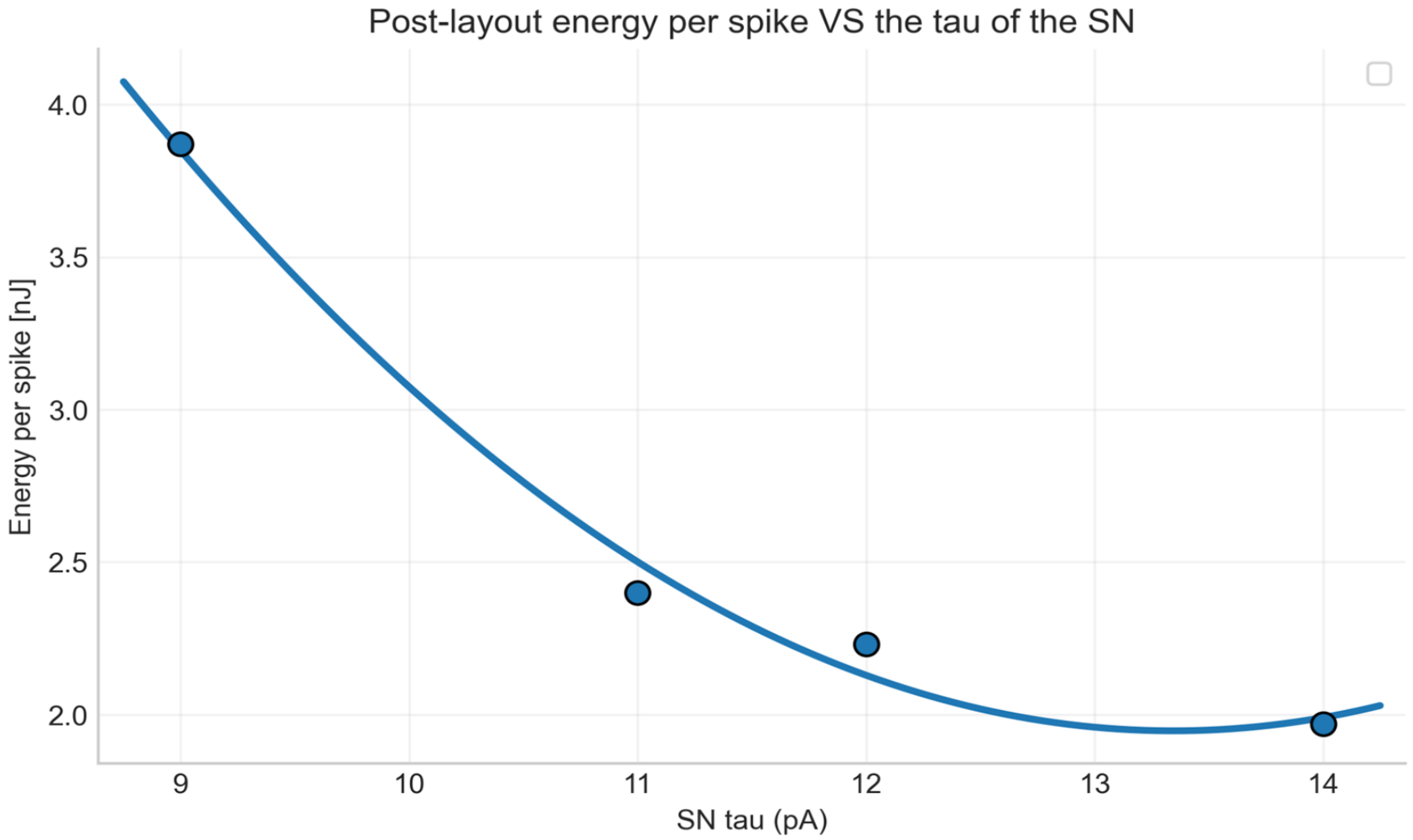
- **Structure**
  - 4 feedback blocks
  - Soma in the center
  - Two rows of PMOS
  - Central block of NMOS
- **Guard rings:**
  - P guard rings around NMOS
  - N guard rings around PMOS
- **Dummy transistors**
  - To separate PMOS and NMOS
  - All around
- **Power grid**
- **Area: 75.66 × 67.02 µm²**



Layout of the final neuron

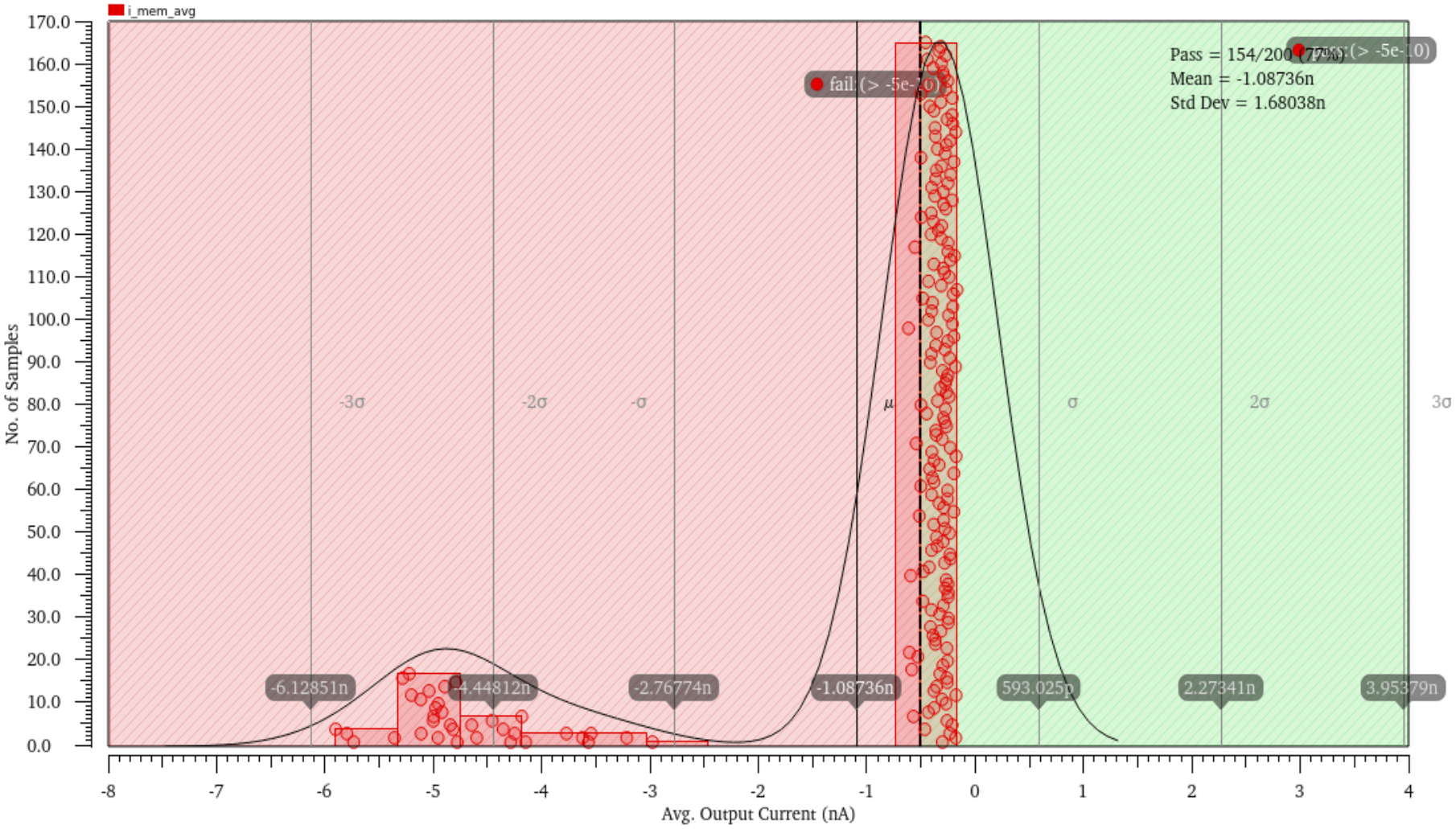
Bursting Neuron –Power analysis Post-layout

As expected, the power consumption is higher post layout, due to the fact that now parasitics are taken into account as well.



Bursting Neuron –Montecarlo Post-layout

Passing samples: 77%



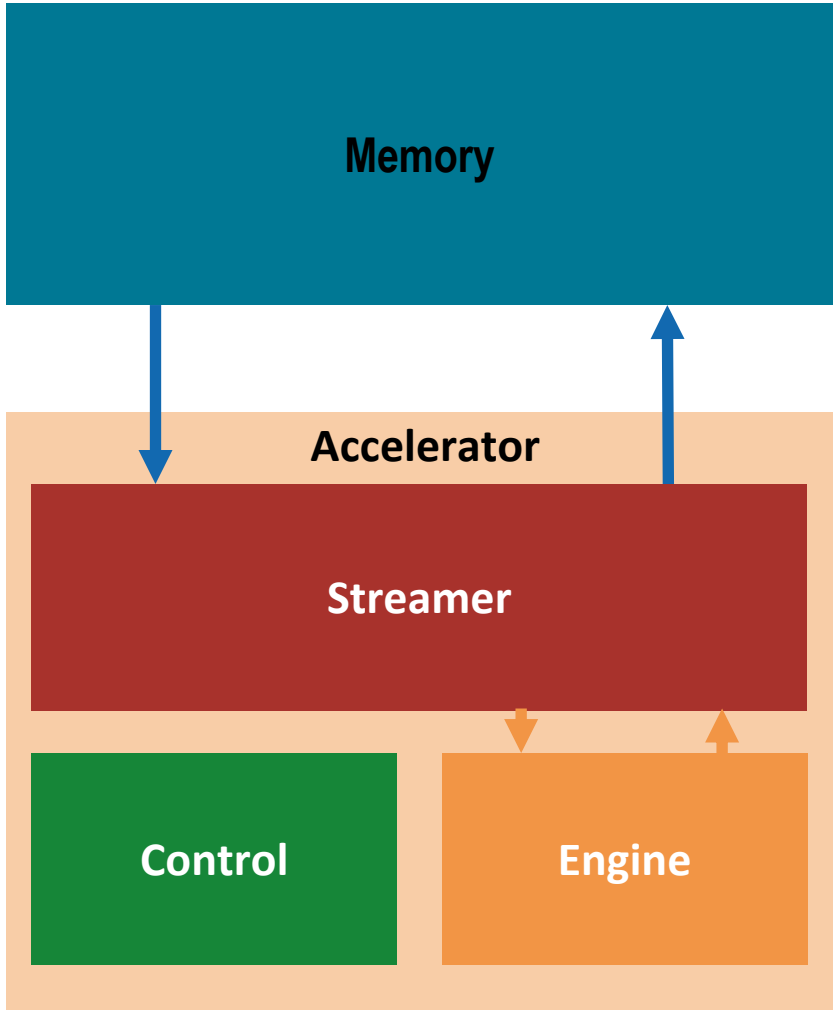
Montecarlo simulation results for mismatch + process variations

ETH zürich

Designing a SpMM accelerator

SpMM Accelerator-Overview

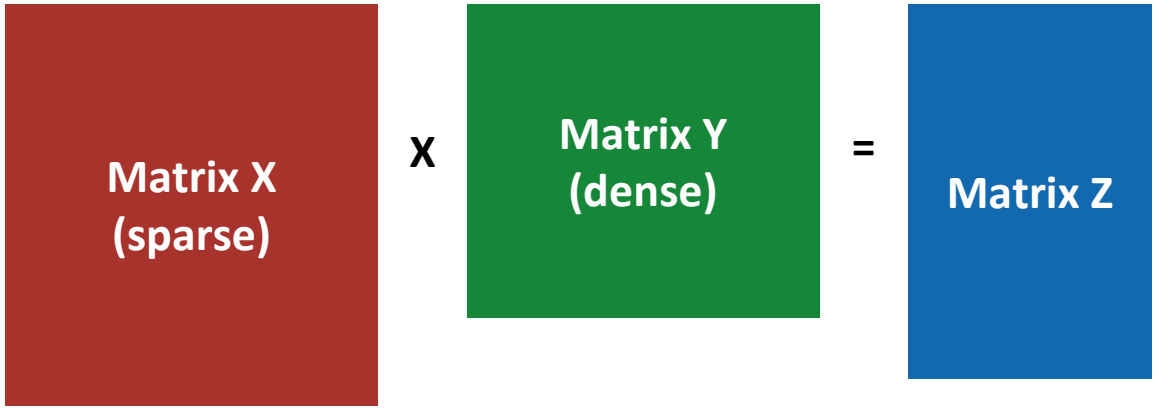
- **Context**
  - **Semester thesis:** under Institute for Integrated Systems (IIS)
  - **Scope:** learn the higher-end of an ASIC workflow
- **Goal**
  - Have the functioning RTL of the sparsity-aware streamer of the accelerator
- **Workflow**
  - **Analytical evaluation:** Gustavson and Outer product dataflows
  - **Architecture design:** design the architecture of the streamer
  - **Functional verification:** different matrix sizes and sparsity
  - **Synthesis:** area results



Structure of the accelerator

Case study

- **Assumptions on data**
  - X sparse
  - Y dense
- **X encoded with bitmap format**
  - The matrix is stored as its dense elements + a one-hot encoded mask of metadata
- **The engine takes chunks of the matrices**
  - The size of the engine determines the size of the tiles of X and Y.



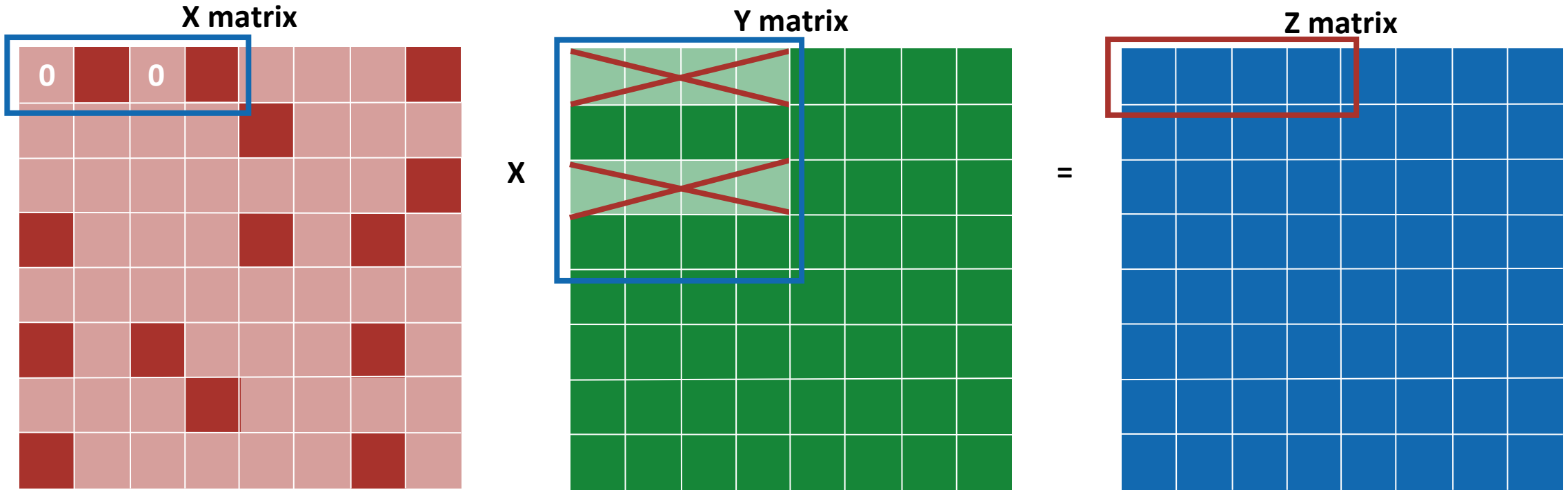
Naming conventions used

My contributions

- **Analytical evaluation**
  - Compared Gustavson vs. Outer Product dataflows under various reuse strategies for Sparse Matrix Multiplication (SpMM)
- **Sparsity-aware extension of the Hardware Processing Engine's (HWPE) streamer**
  - Integrated sparsity handling logic into the streamer
  - Functionally verified on arbitrary matrix, varying sizes and sparsity levels
- **Design Evaluation**
  - Performance
  - Area overhead

The idea

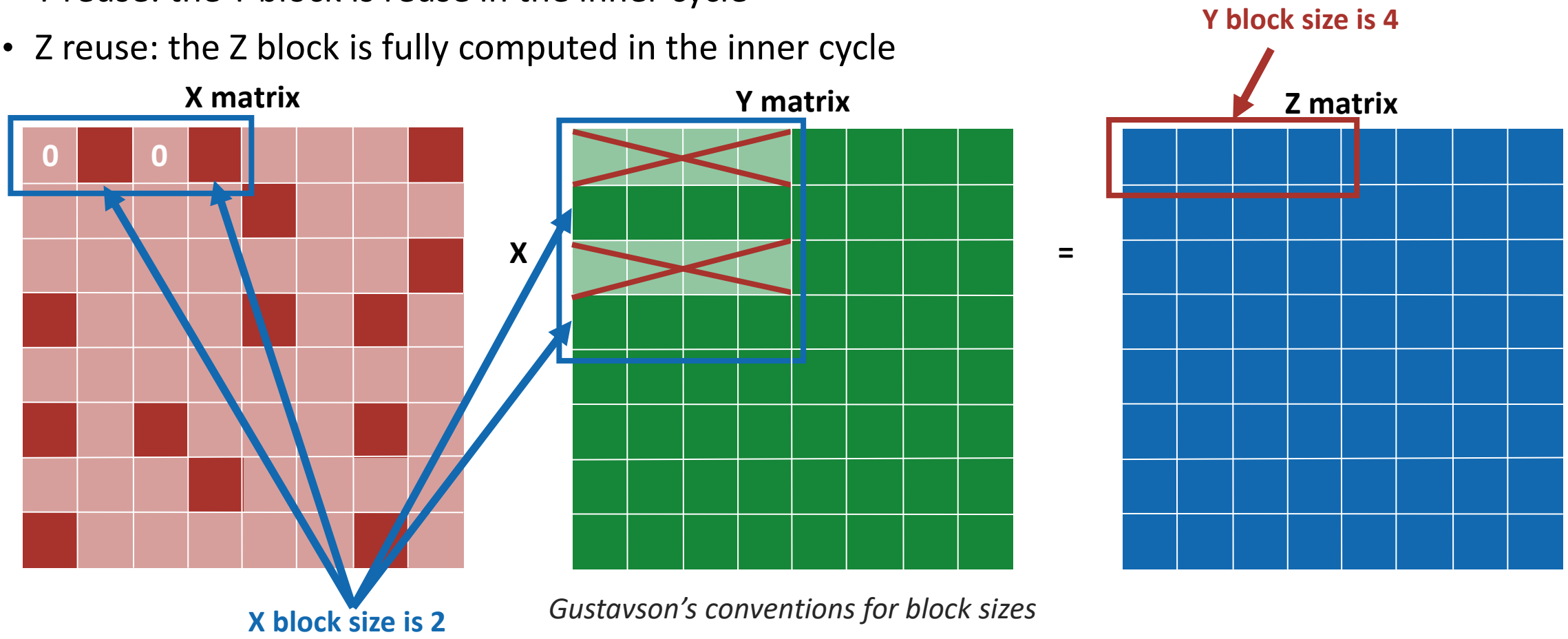
- **Avoid multiplications by zero**
  - If the element in the row of X is a zero, I can skip the correspondent row of Y
  - This way I avoid unnecessary loads and multiplications that consume bandwidth and cycles



Example with Gustavson

Dataflows options -Gustavson

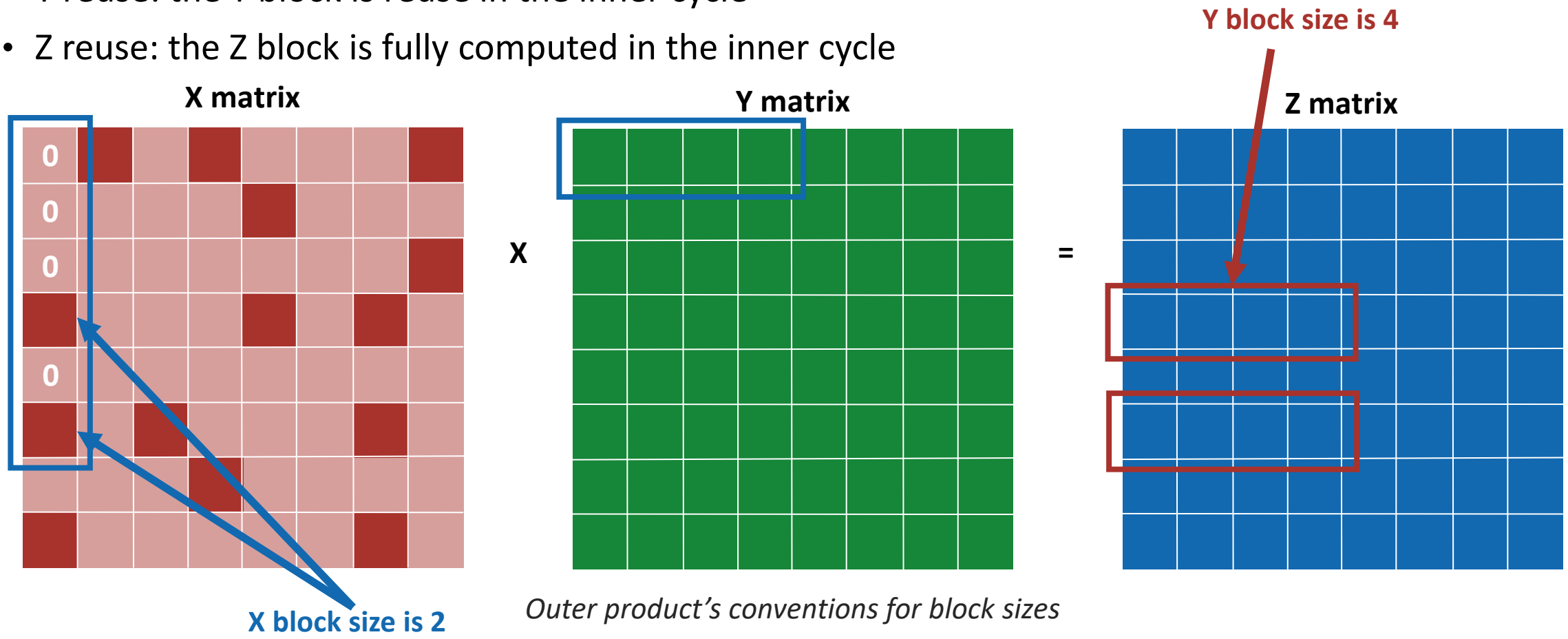
- **There are 3 possible options**
  - X reuse: the X block is reused in the inner cycle
  - Y reuse: the Y block is reuse in the inner cycle
  - Z reuse: the Z block is fully computed in the inner cycle



Gustavson's conventions for block sizes

Dataflows options –Outer product

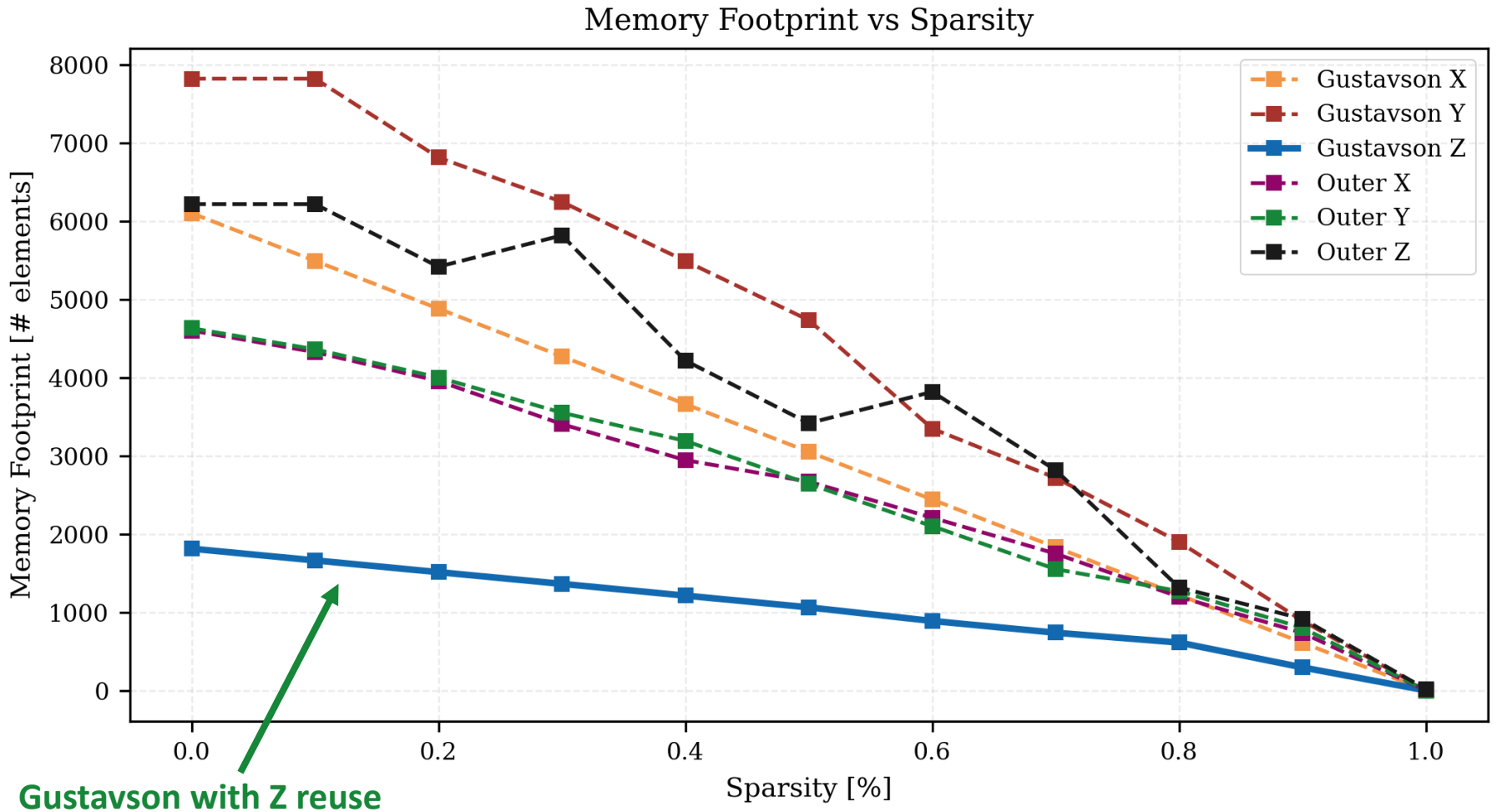
- **There are 3 possible options**
  - X reuse: the X block is reused in the inner cycle
  - Y reuse: the Y block is reuse in the inner cycle
  - Z reuse: the Z block is fully computed in the inner cycle



Outer product's conventions for block sizes

Analytical models comparison –Memory footprint

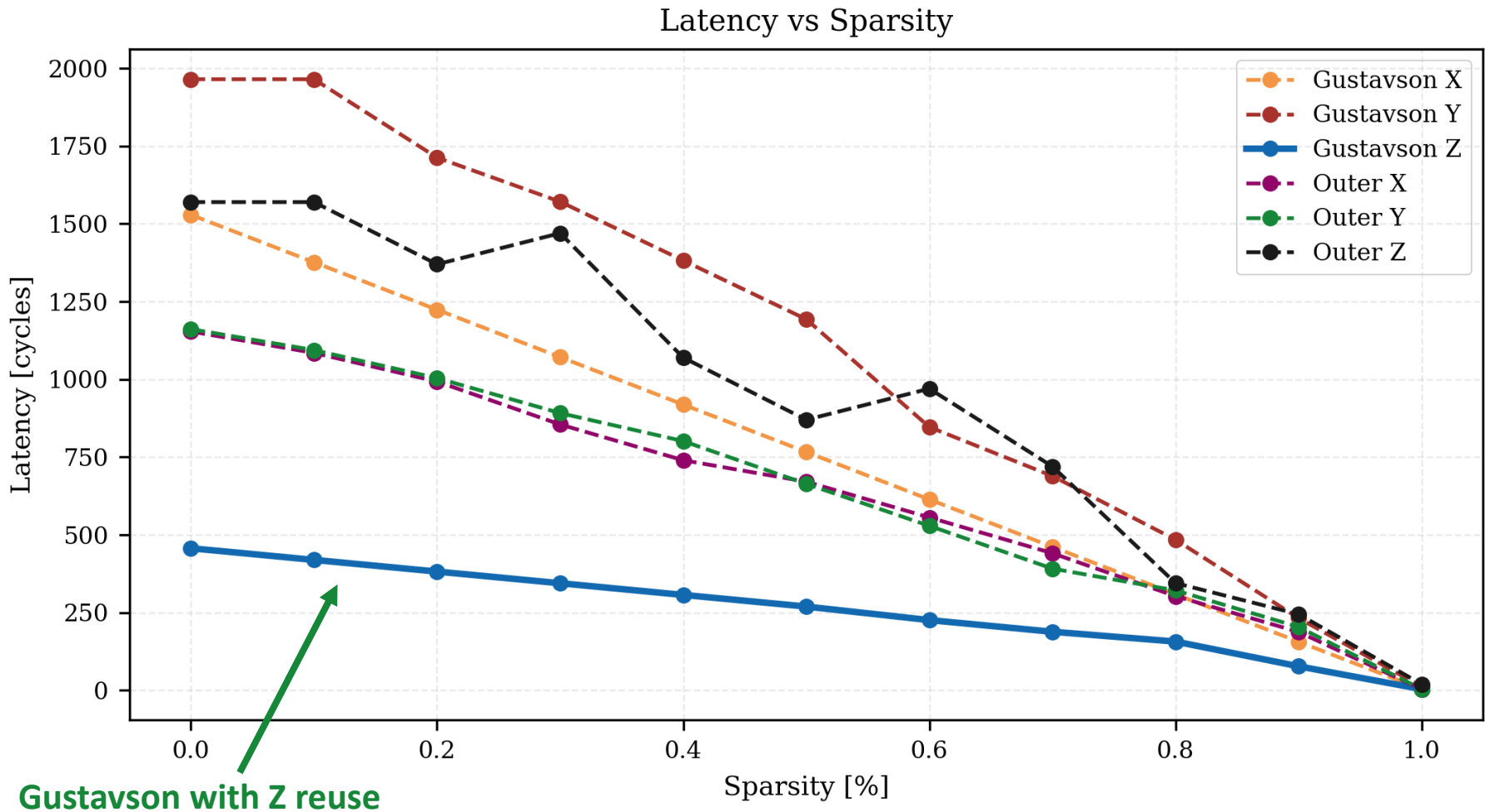
- **Assumptions:**
  - MAC units = 4
  - X rows = 32
  - Y columns = 32
  - Inner dim. = 32
  - Metadata chunk = 128b
  - Input data size = 32b
  - Output data size = 64b
  - BW = 128b



Gustavson with Z reuse has the lowest memory footprint

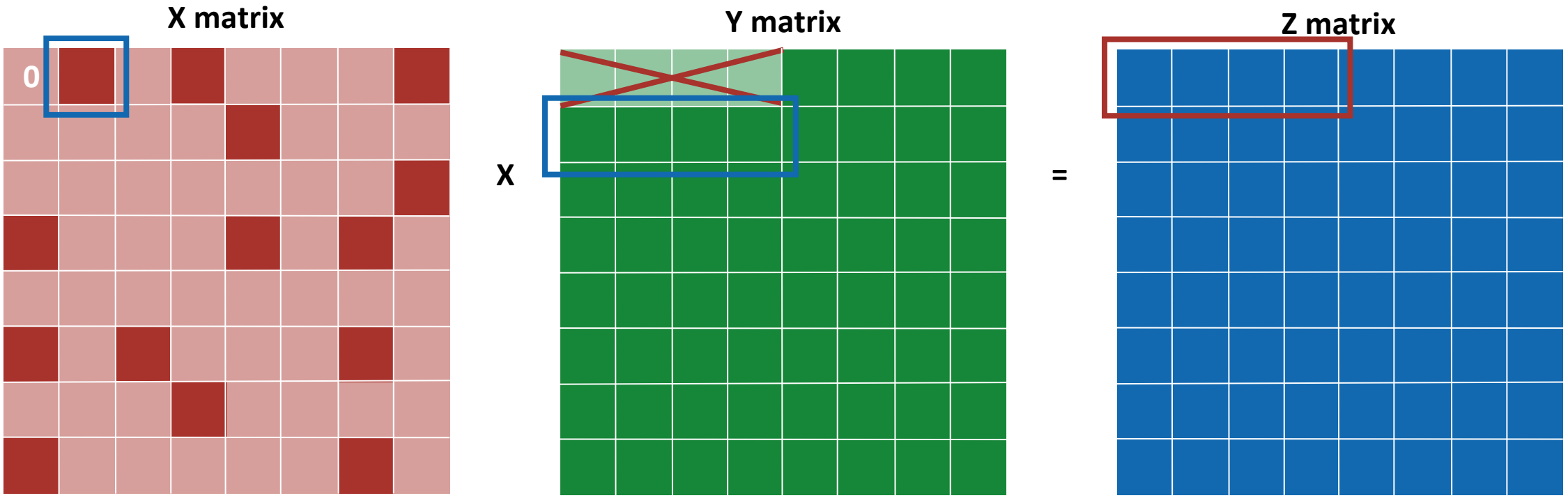
Analytical models comparison –Latency

- **Assumptions:**
  - MAC units = 4
  - X rows = 32
  - Y columns = 32
  - Inner dim. = 32
  - Metadata chunk = 128b
  - Input data size = 32b
  - Output data size = 64b
  - BW = 128b



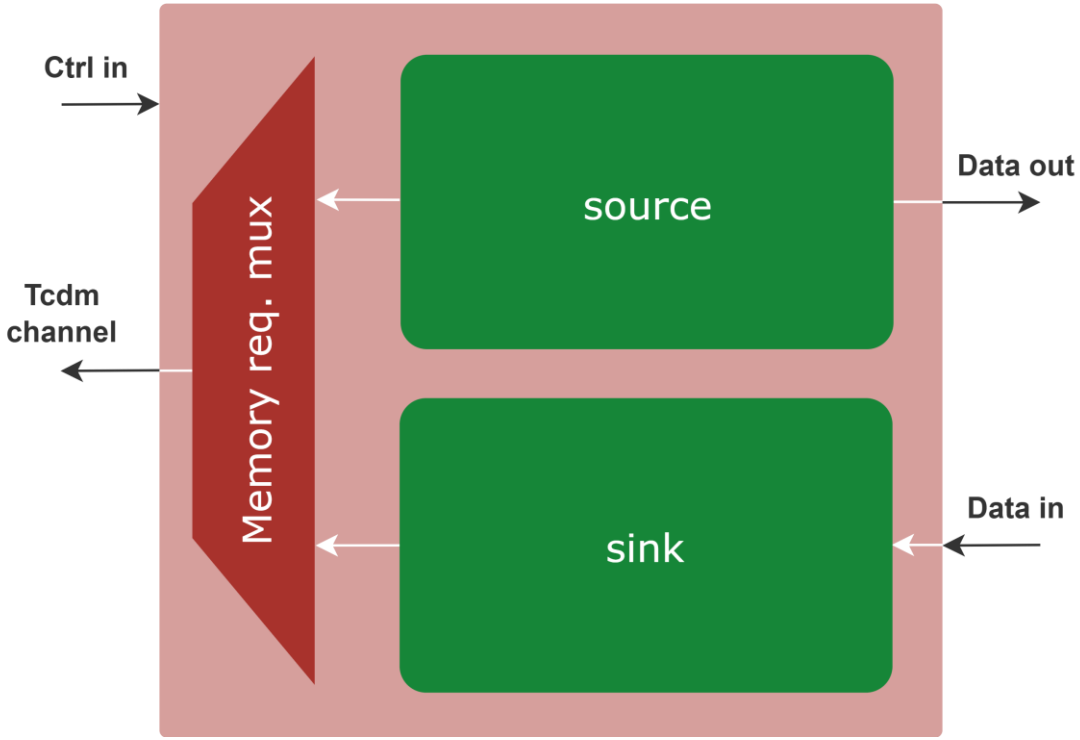
Final choice

- **Overall the Gustavson dataflow with Z reuse seems to be the best**
  - Never loads chunks of Z from memory
  - Reuses the metadata as much as possible



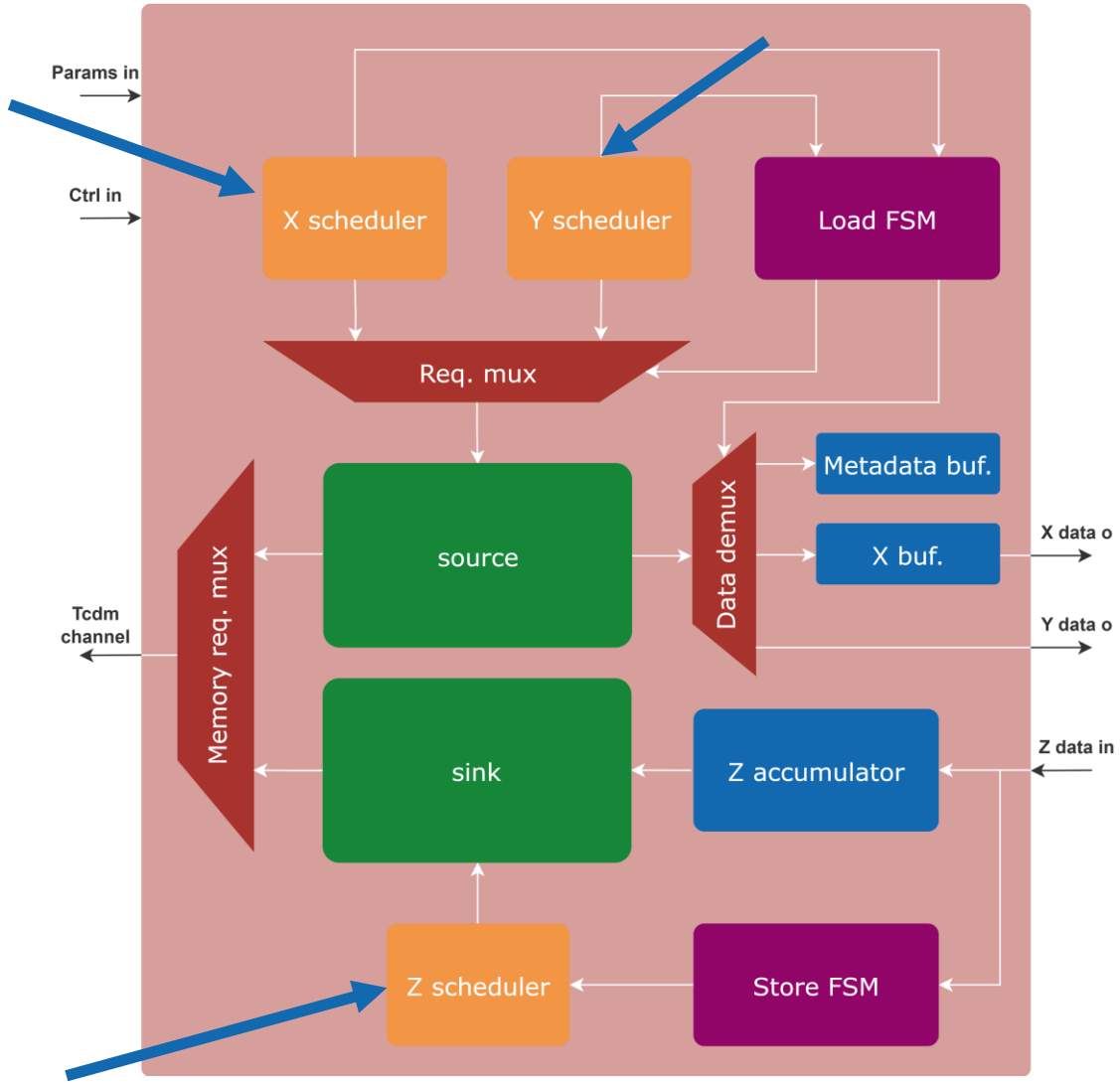
Gustavson dataflow with Z stationary

Accelerator's streamer architecture VS datamover one



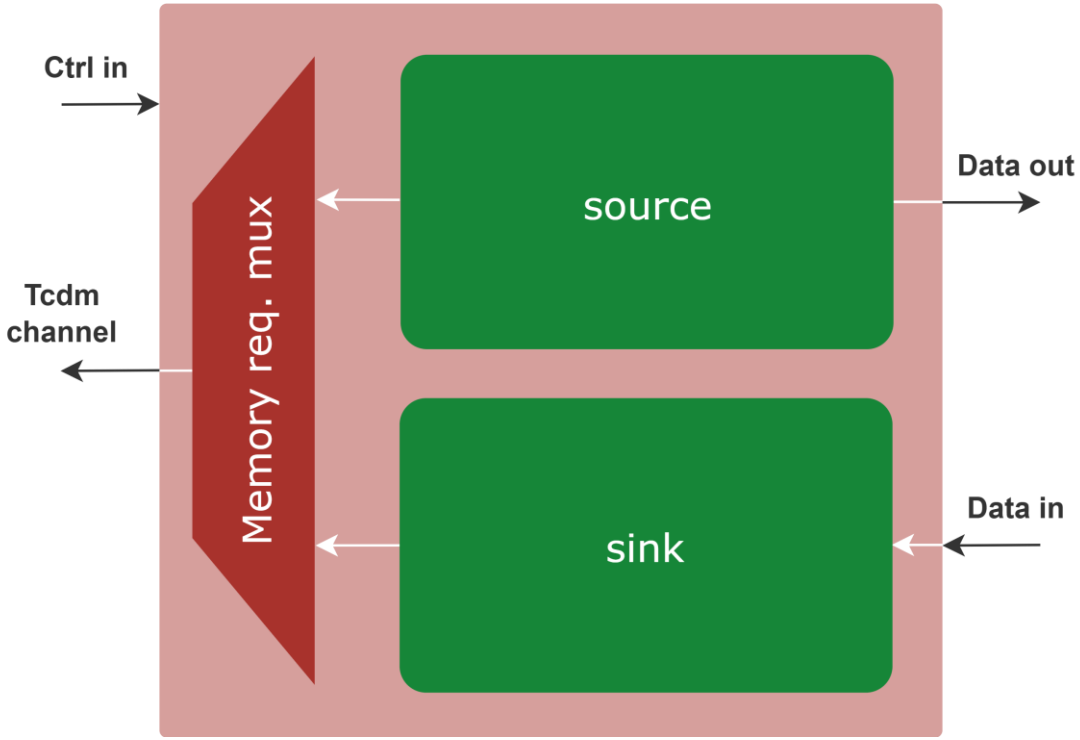
Datamover's streamer architecture

VS



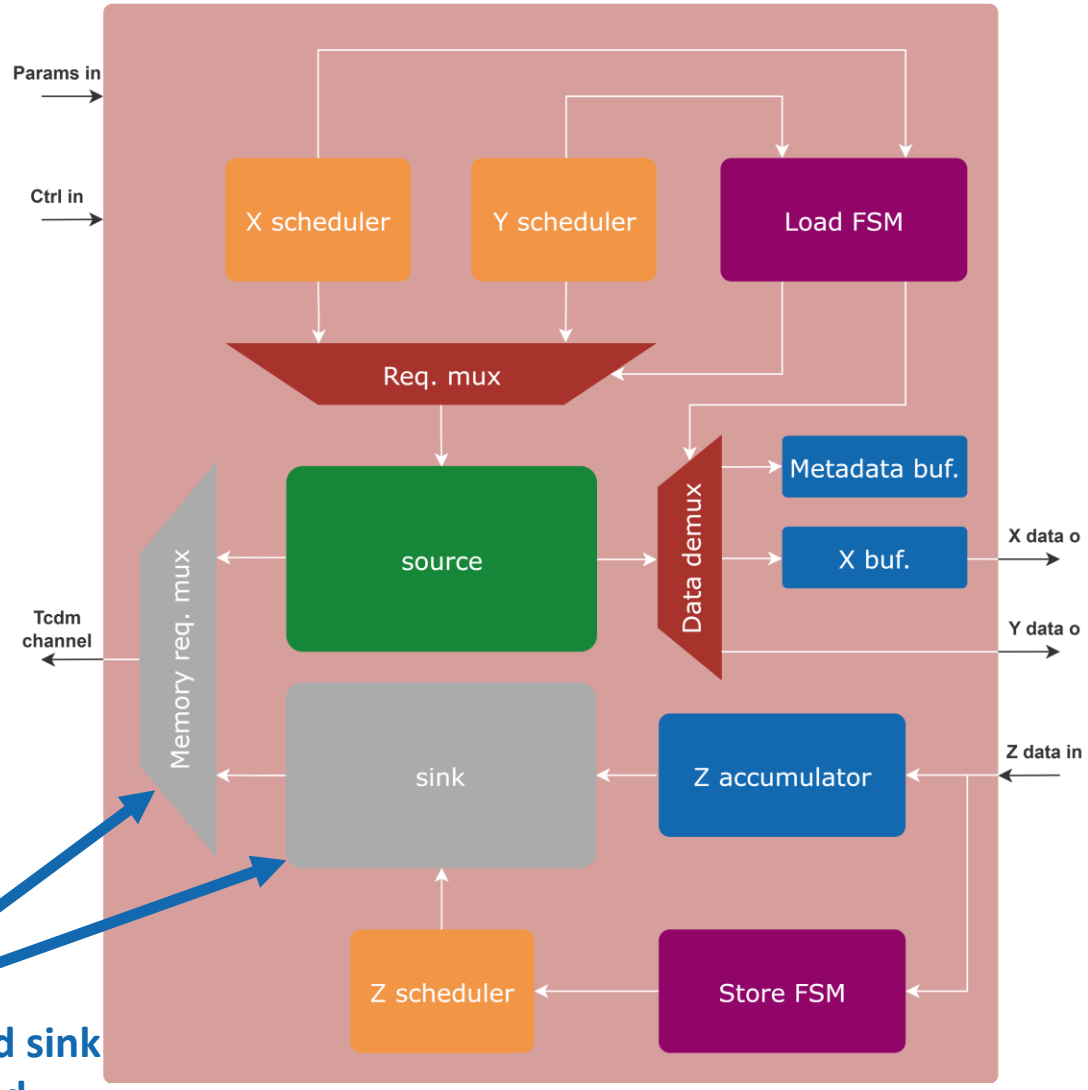
Accelerator's streamer architecture

Accelerator's streamer architecture VS datamover one



Datamover's streamer architecture

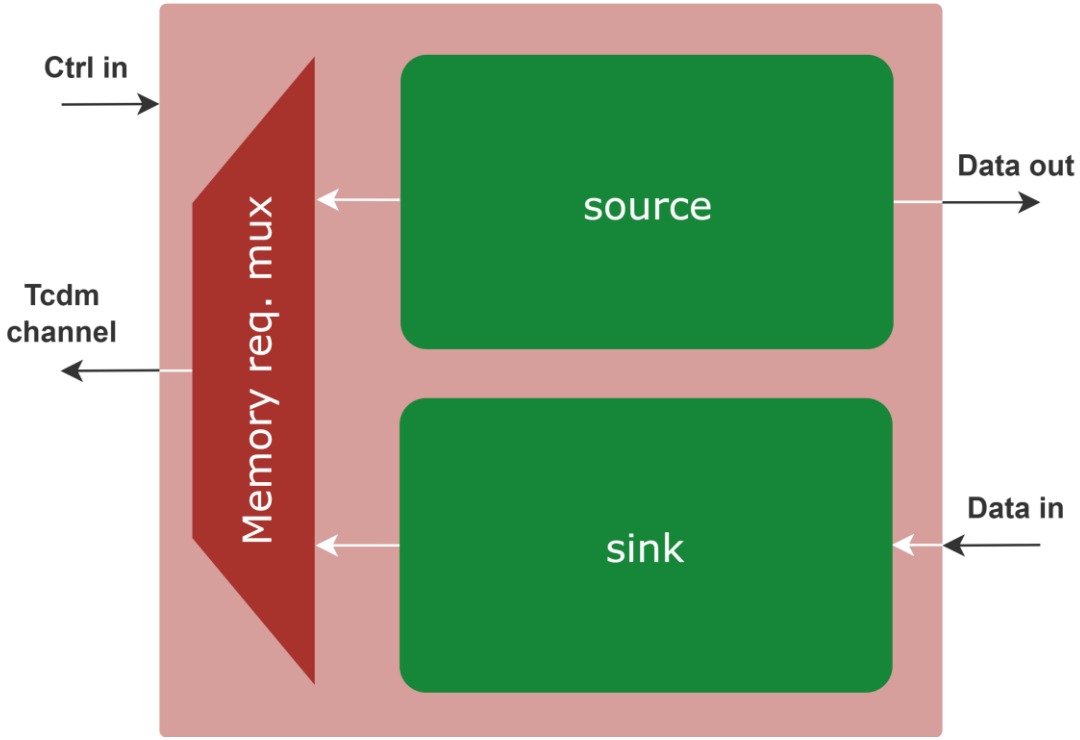
VS



The memory mux and sink were not modified

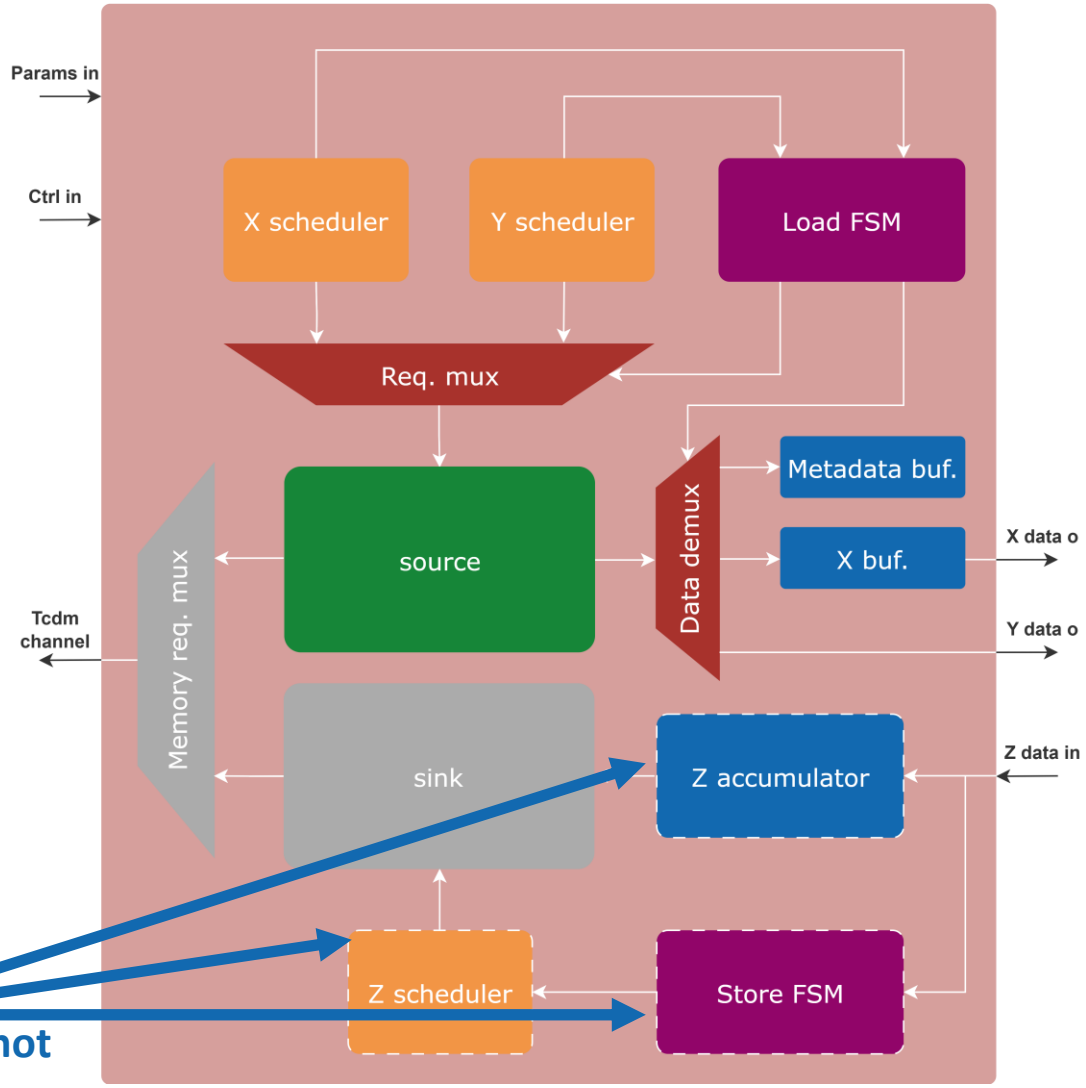
Accelerator's streamer architecture

Accelerator's streamer architecture VS datamover one



Datamover's streamer architecture

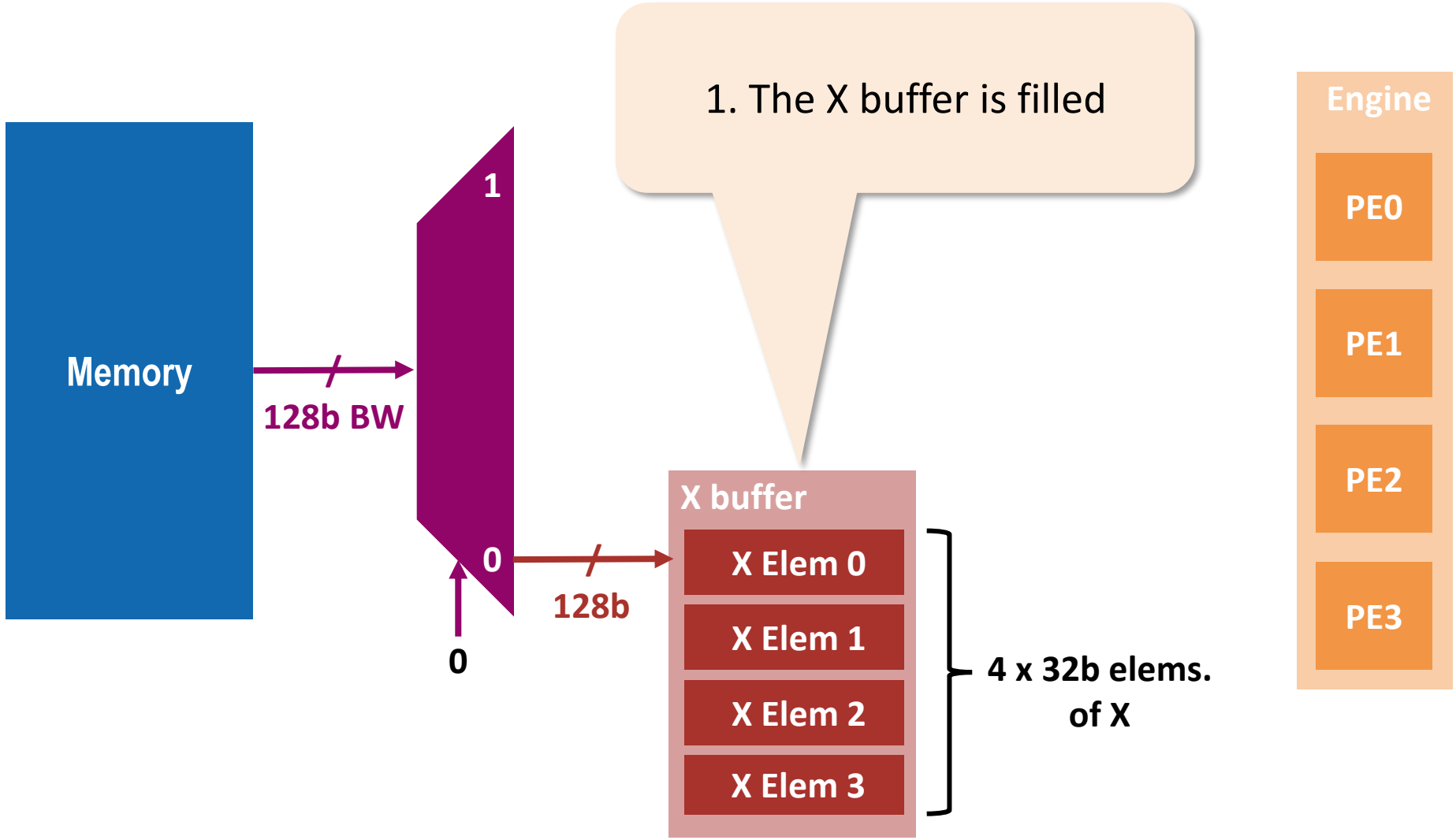
VS



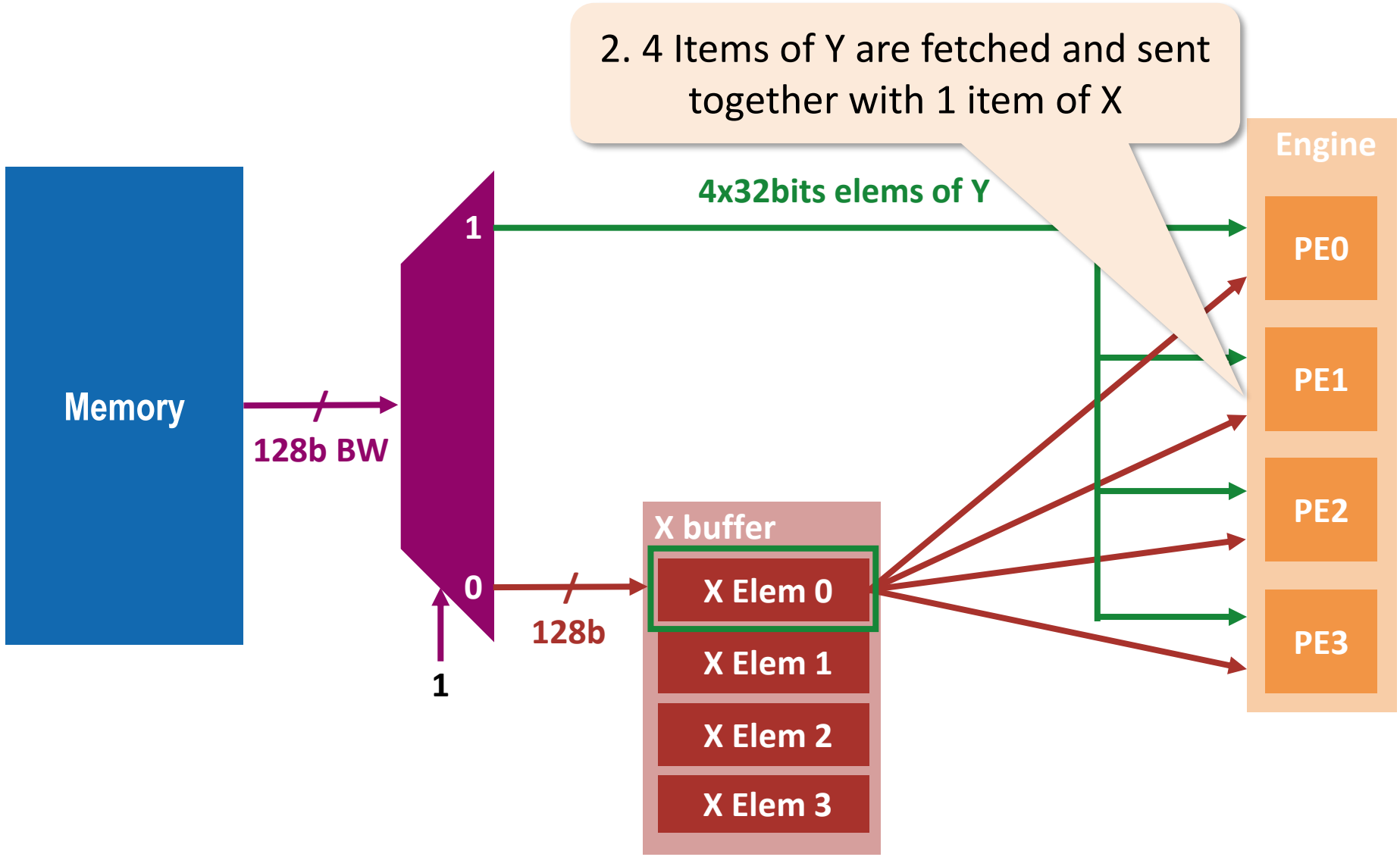
The storing part is not sparsity aware

Accelerator's streamer architecture

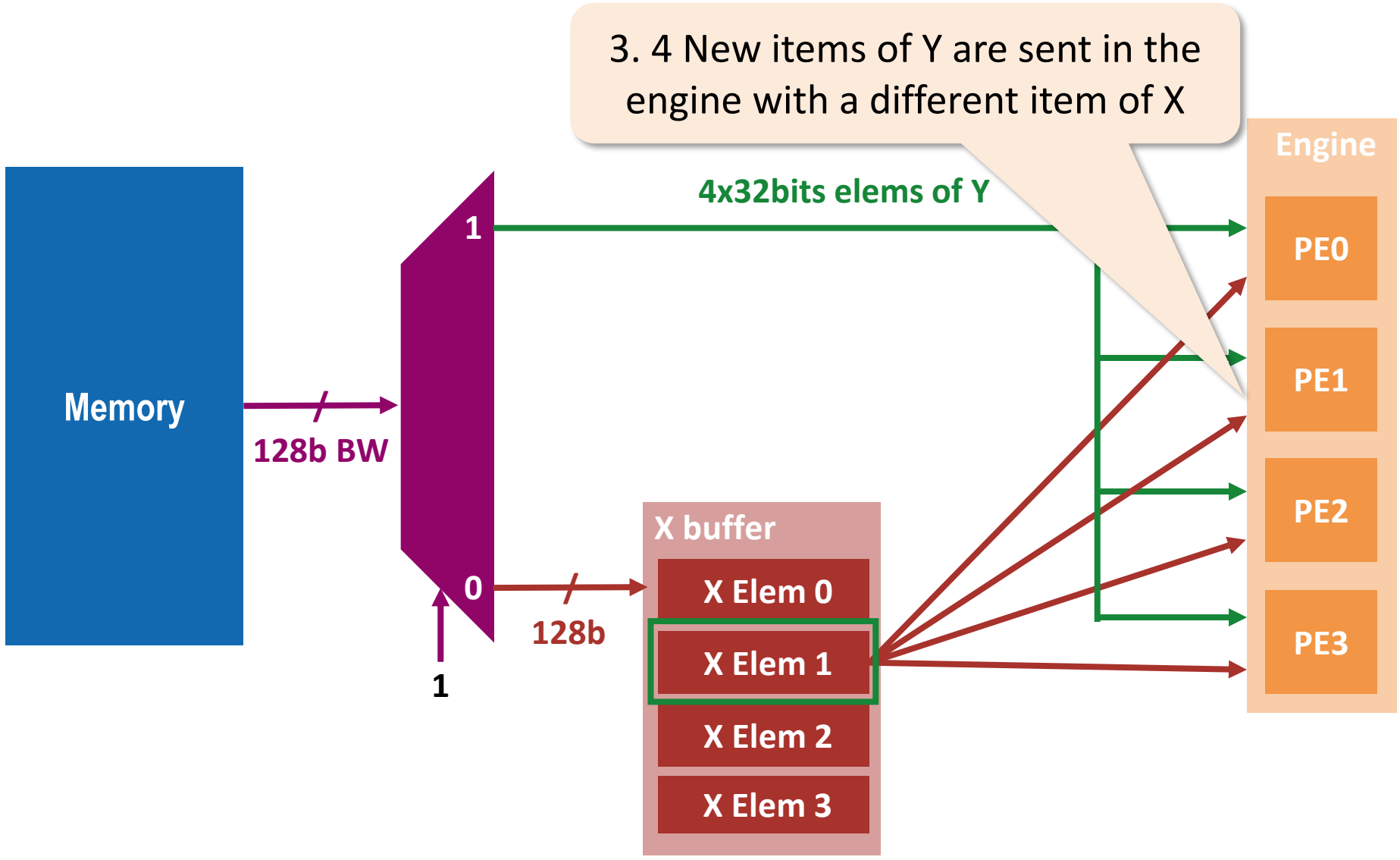
Streamer behavior –Loading part



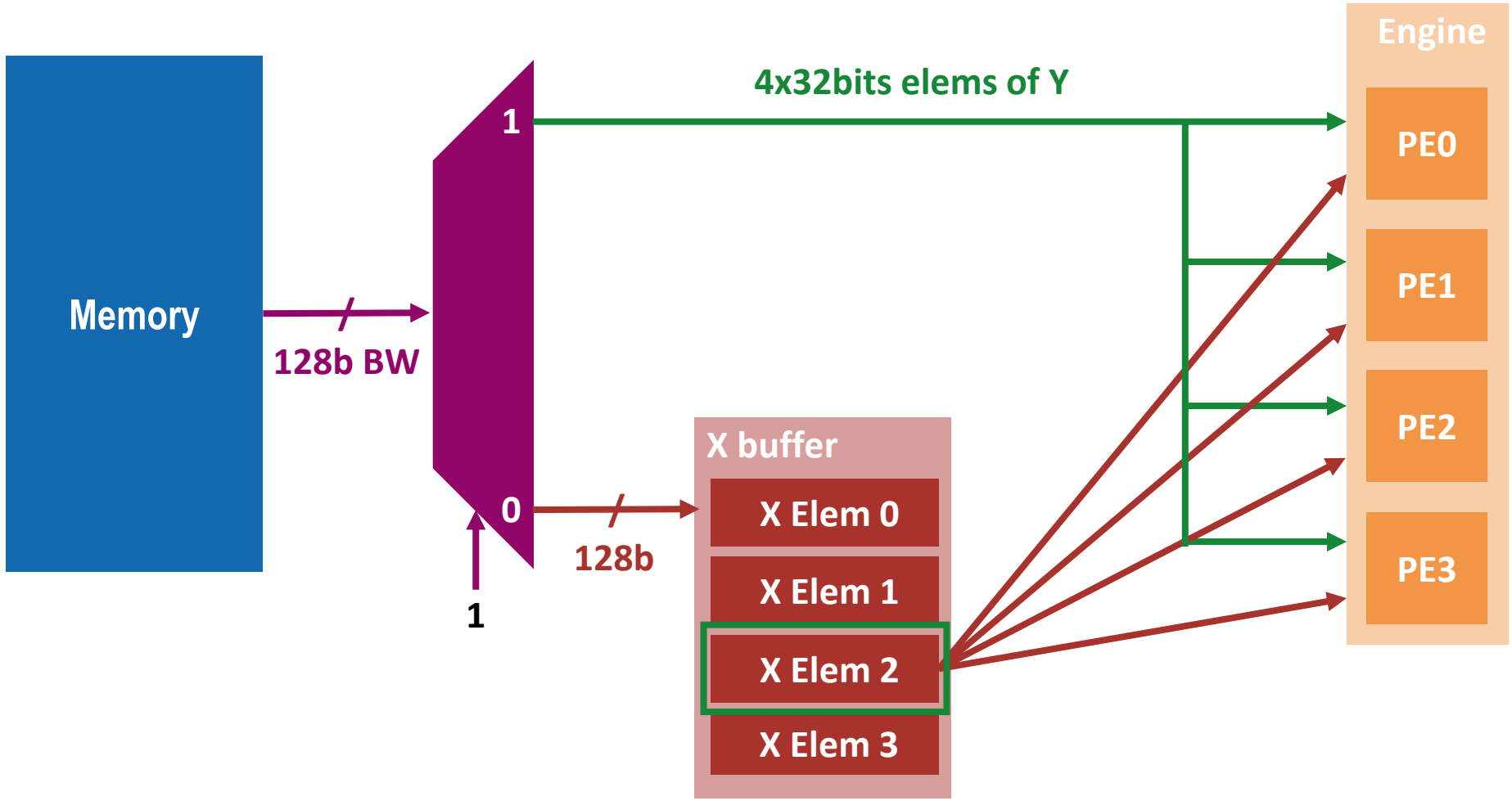
Streamer behavior –Loading part



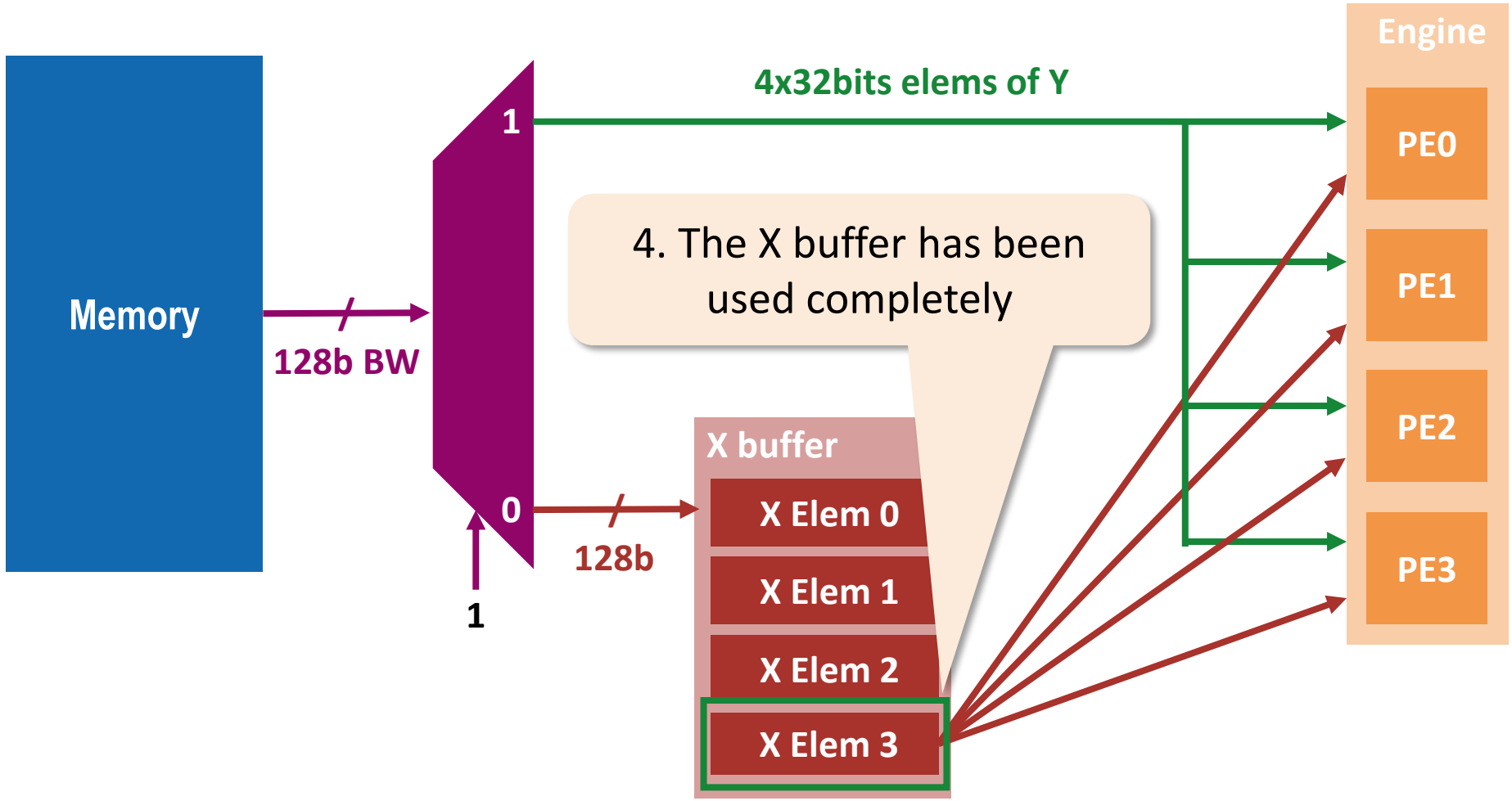
Streamer behavior –Loading part



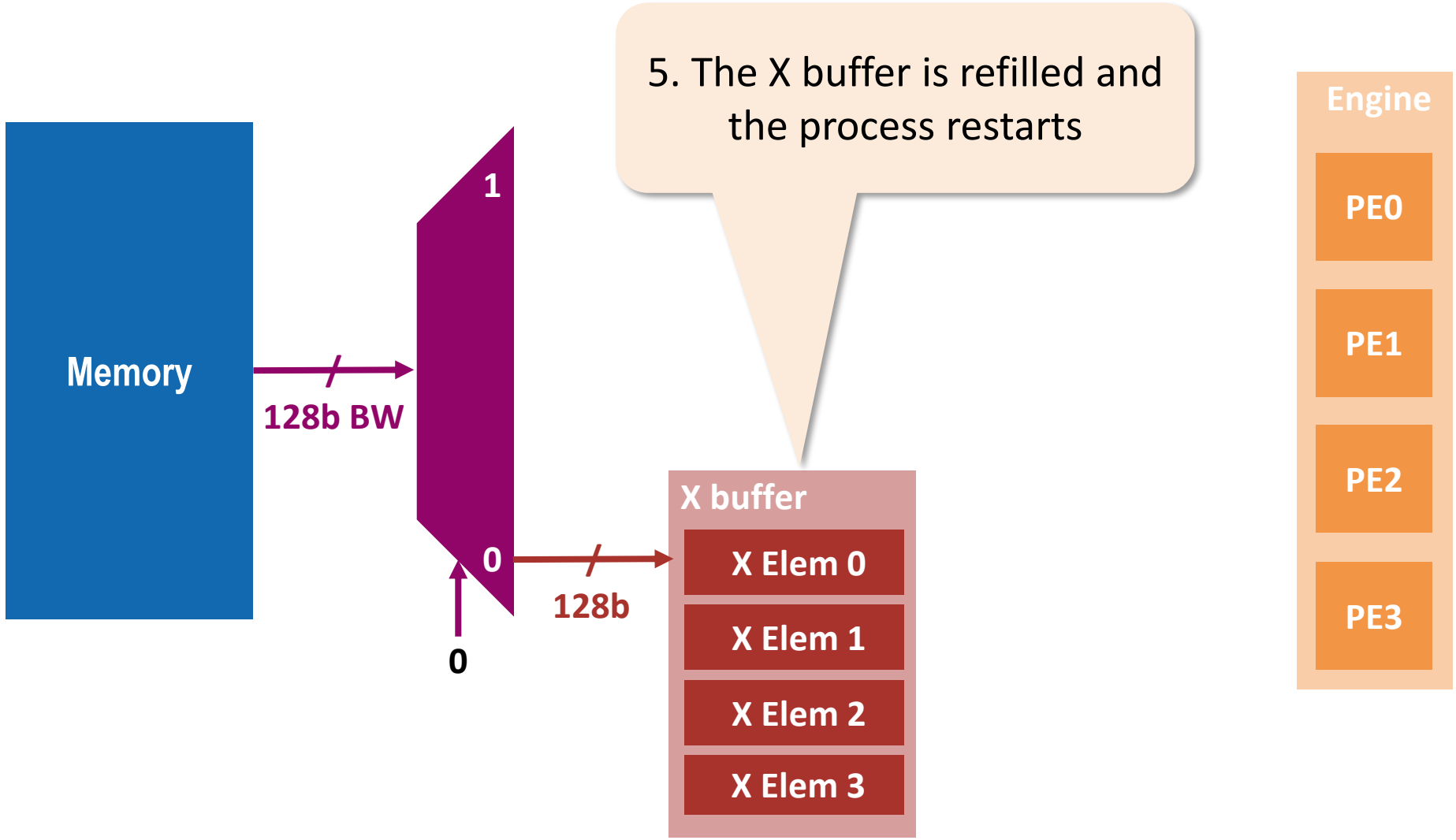
Streamer behavior –Loading part



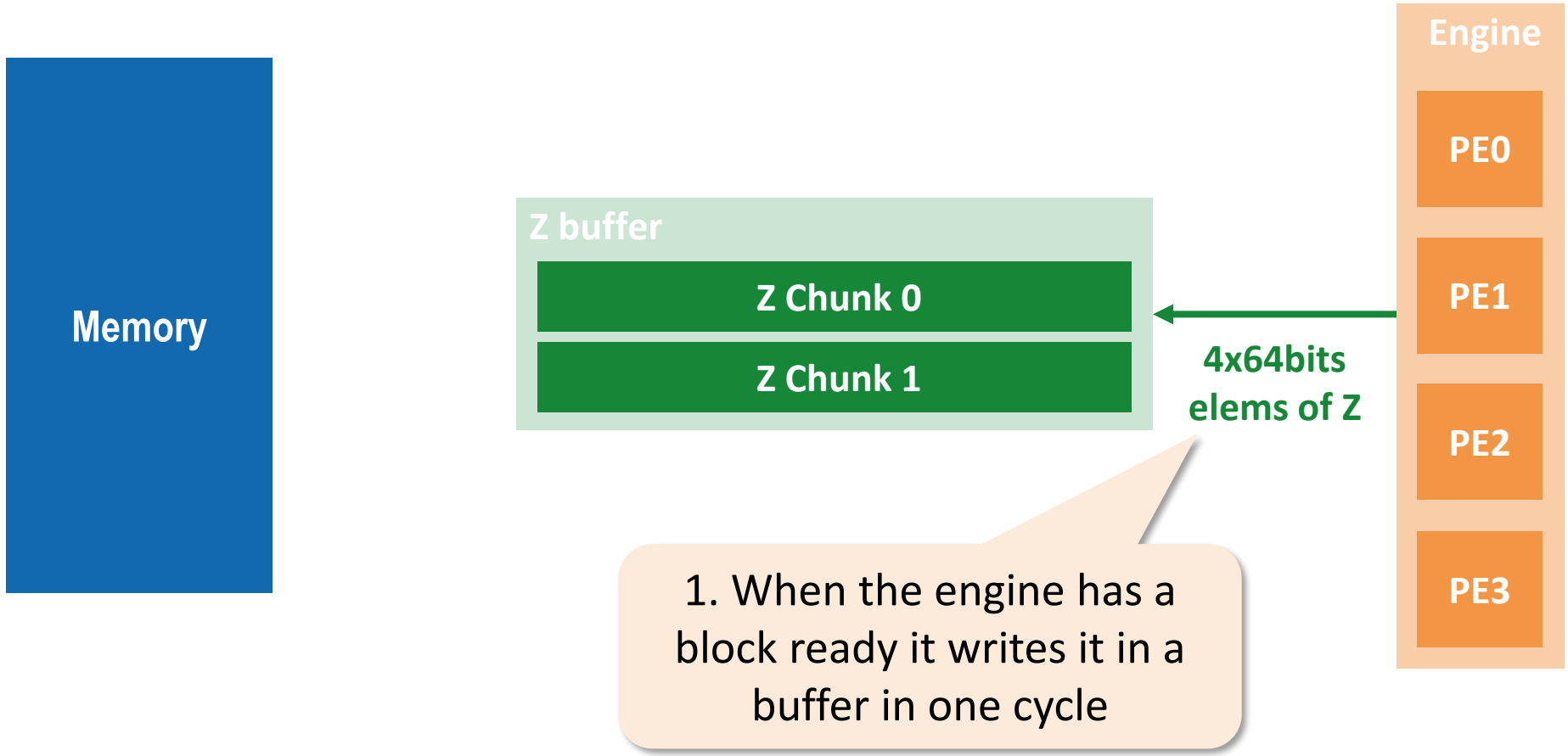
Streamer behavior –Loading part



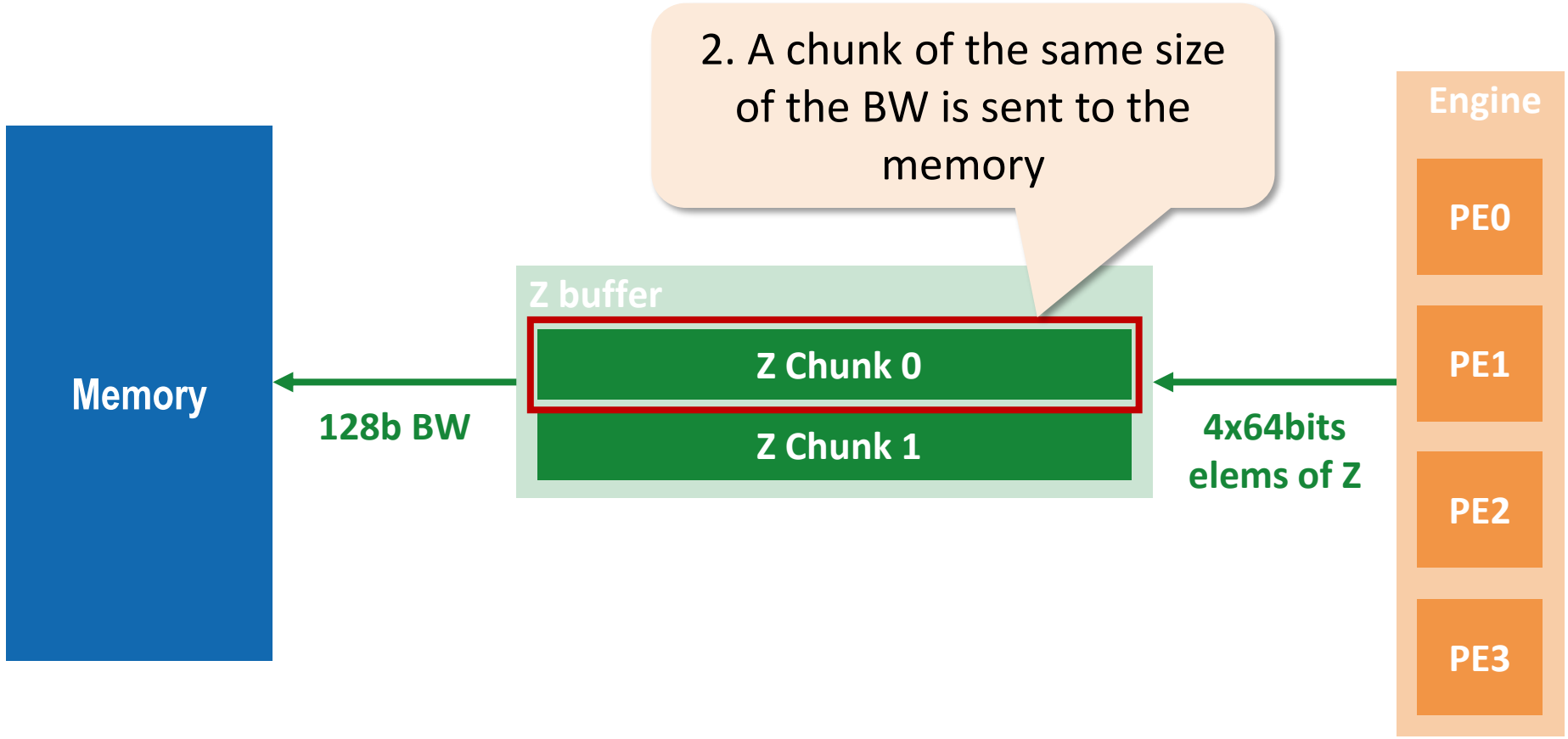
Streamer behavior –Loading part



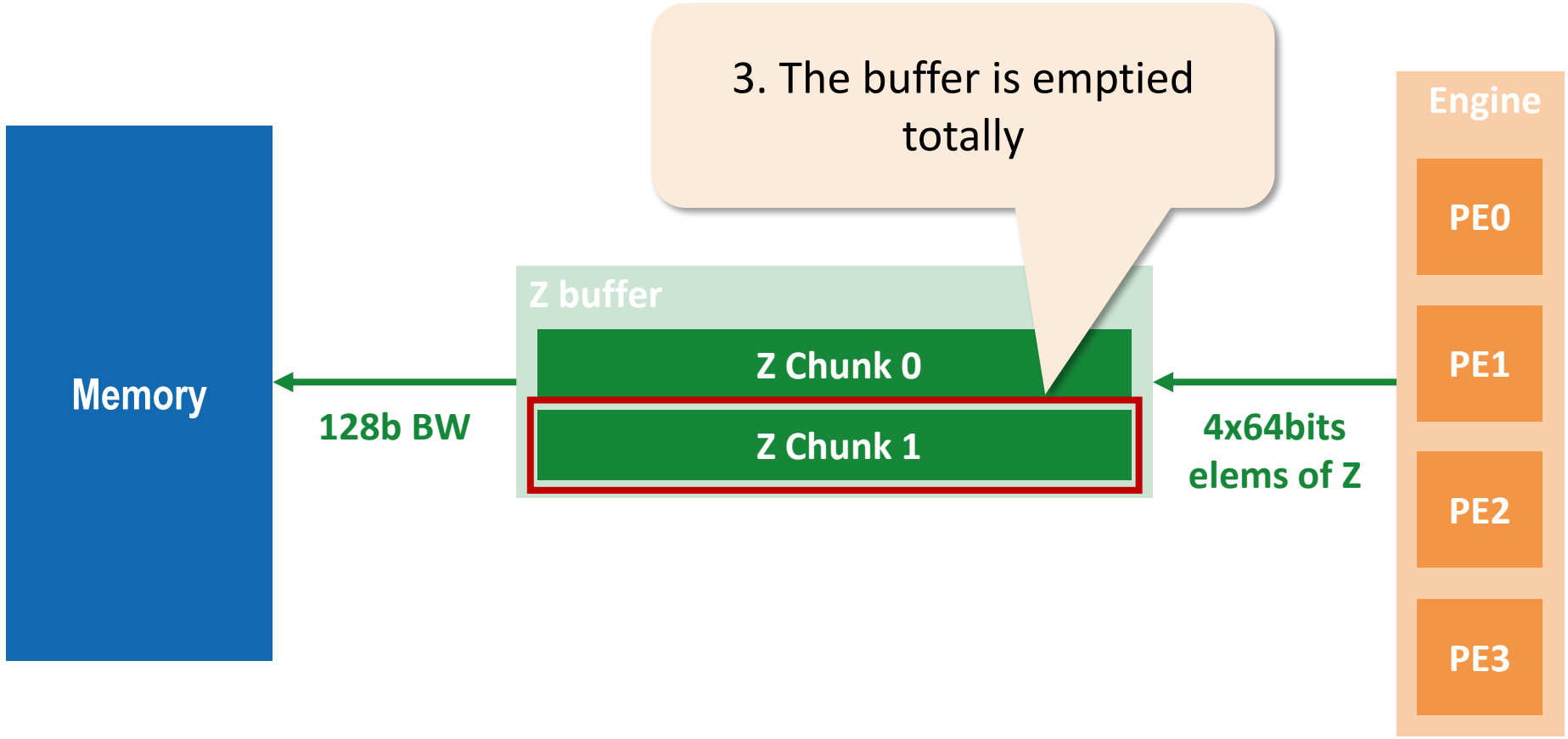
Streamer behavior –Storing part



Streamer behavior –Storing part

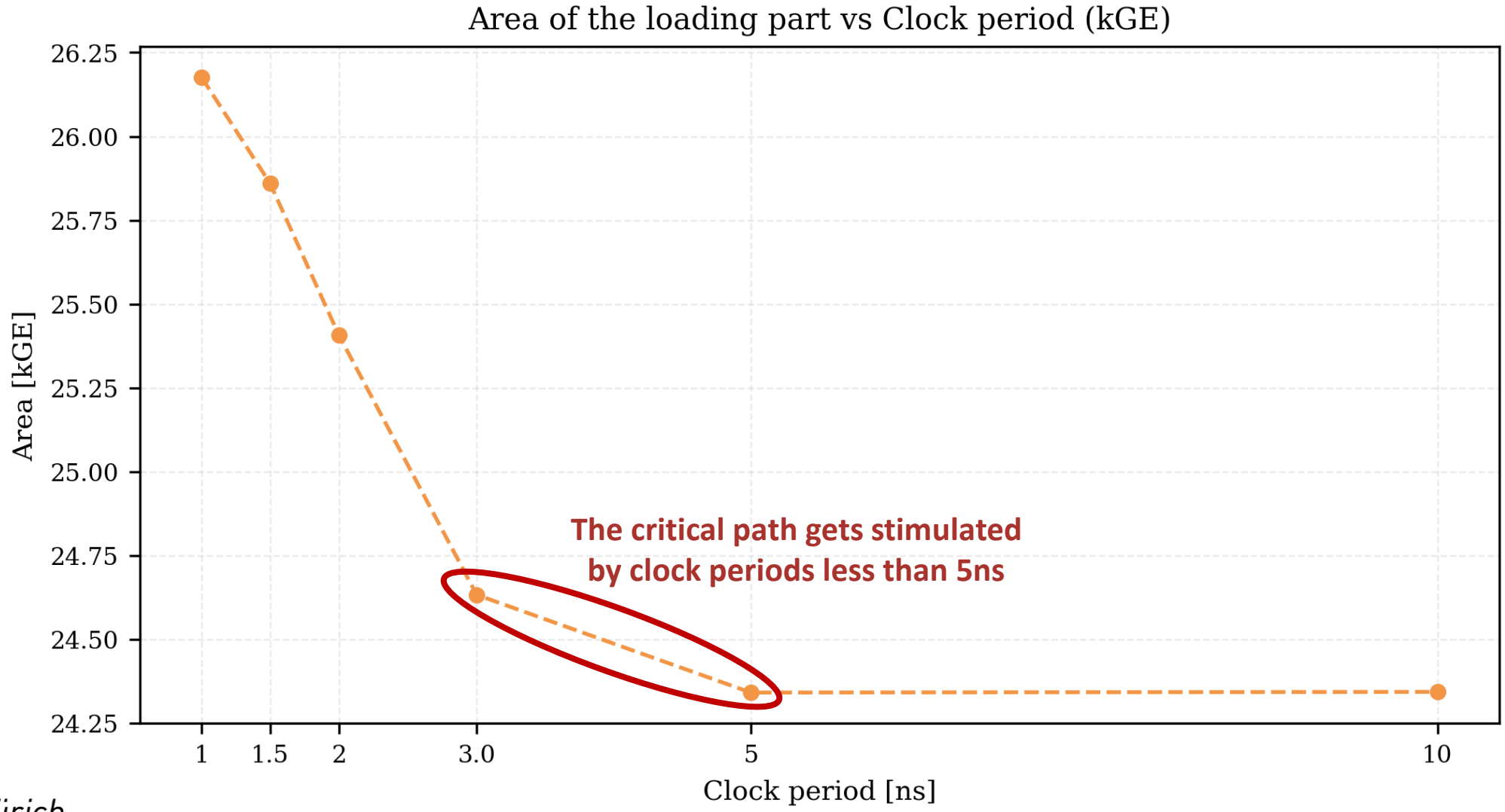


Streamer behavior –Storing part



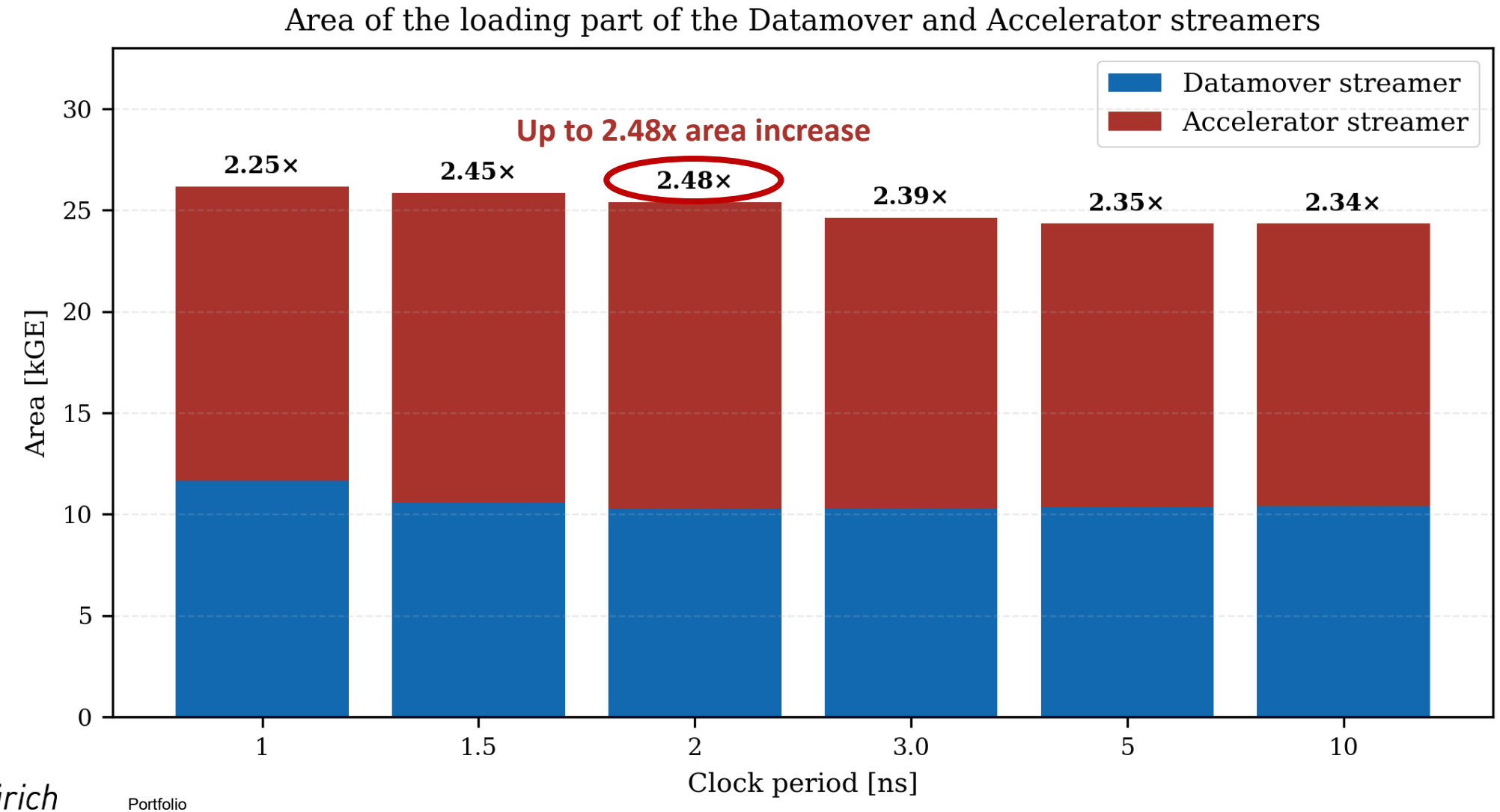
Synthesis results- AT plot

The plot reports the area considering only the loading part which was made sparsity-aware; synthesis was performed using the slow corner of the cells.



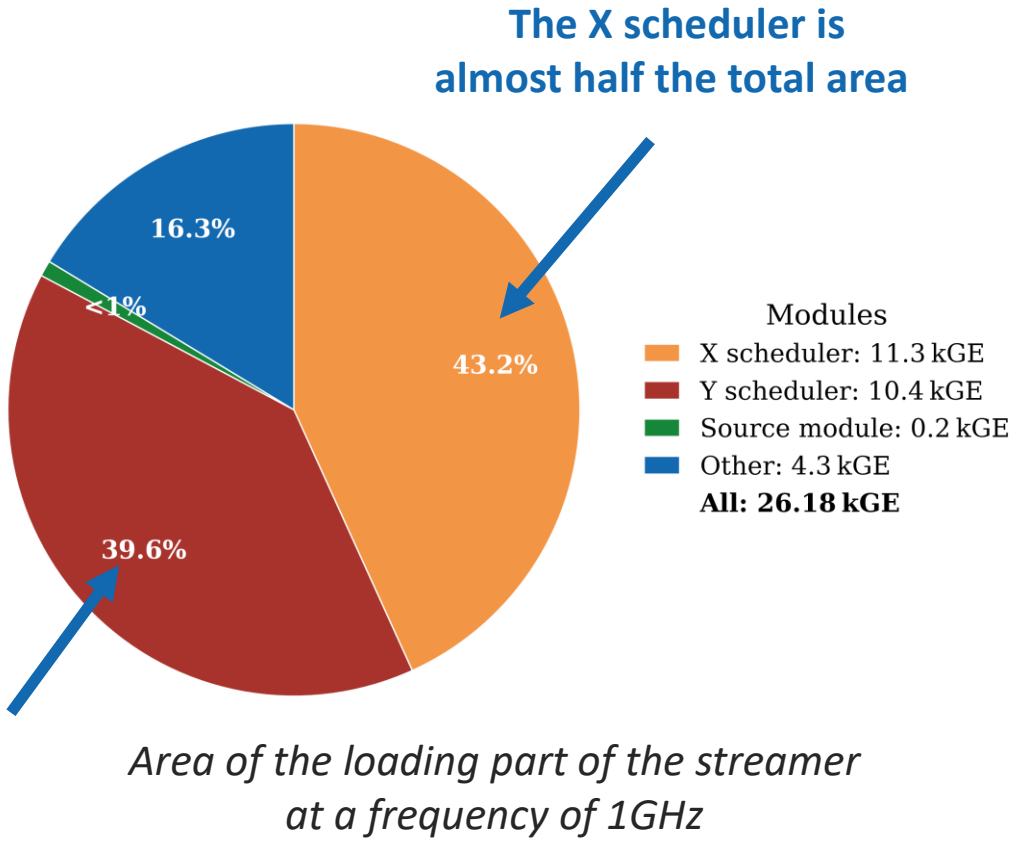
Synthesis results- Streamer VS Datamover

The plot reports the area difference only considering the loading part of the two designs, which was made sparsity-aware.



Synthesis results- Area contributions

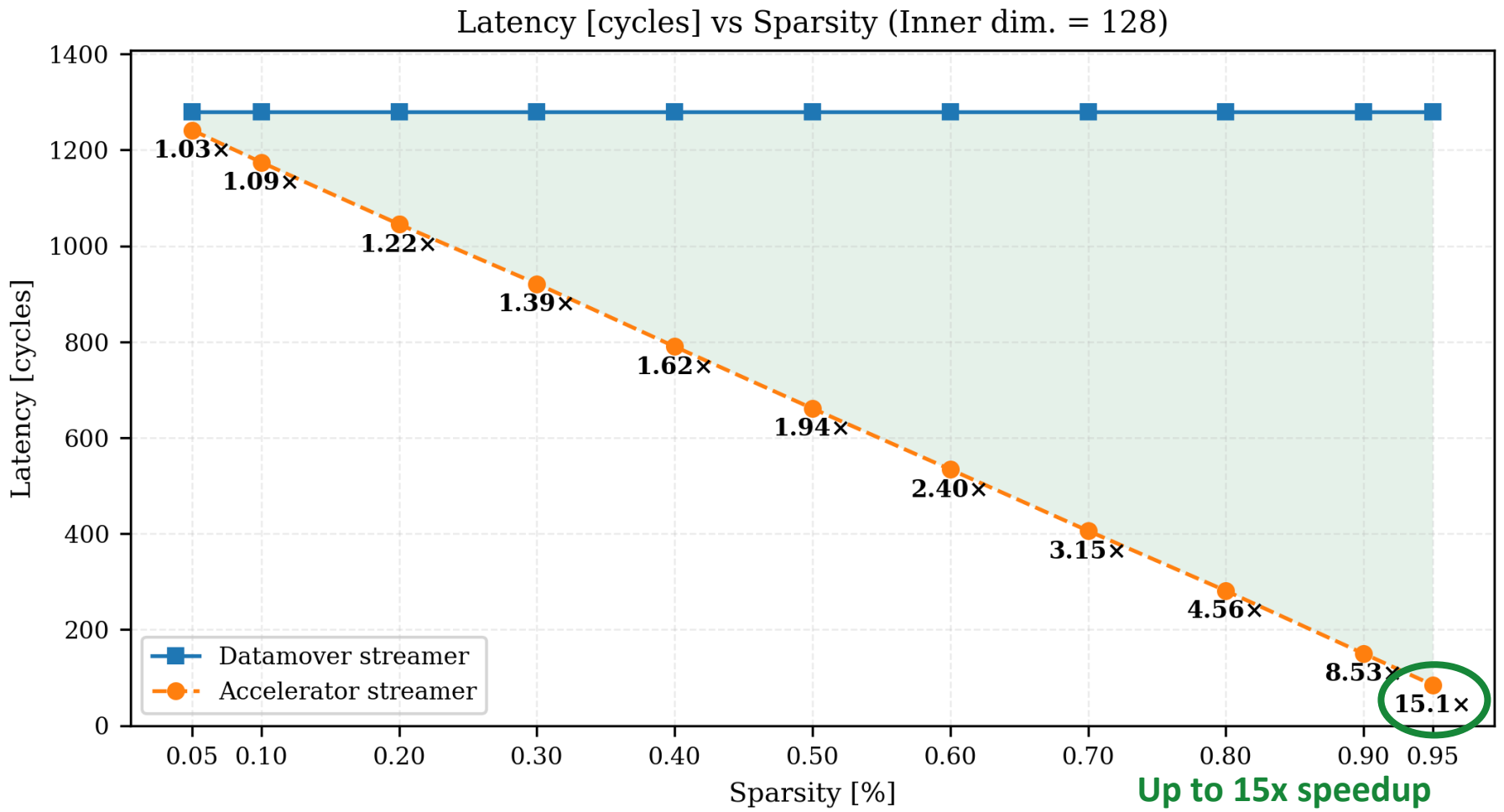
- **Adding intelligence is expensive**
  - The biggest area contributions come from the schedulers
- **Different function of the source module**
  - The source was reduced of about 10 times because it's not "smart" anymore.



Area of the loading part of the streamer at a frequency of 1GHz

RTL results –Latency VS sparsity

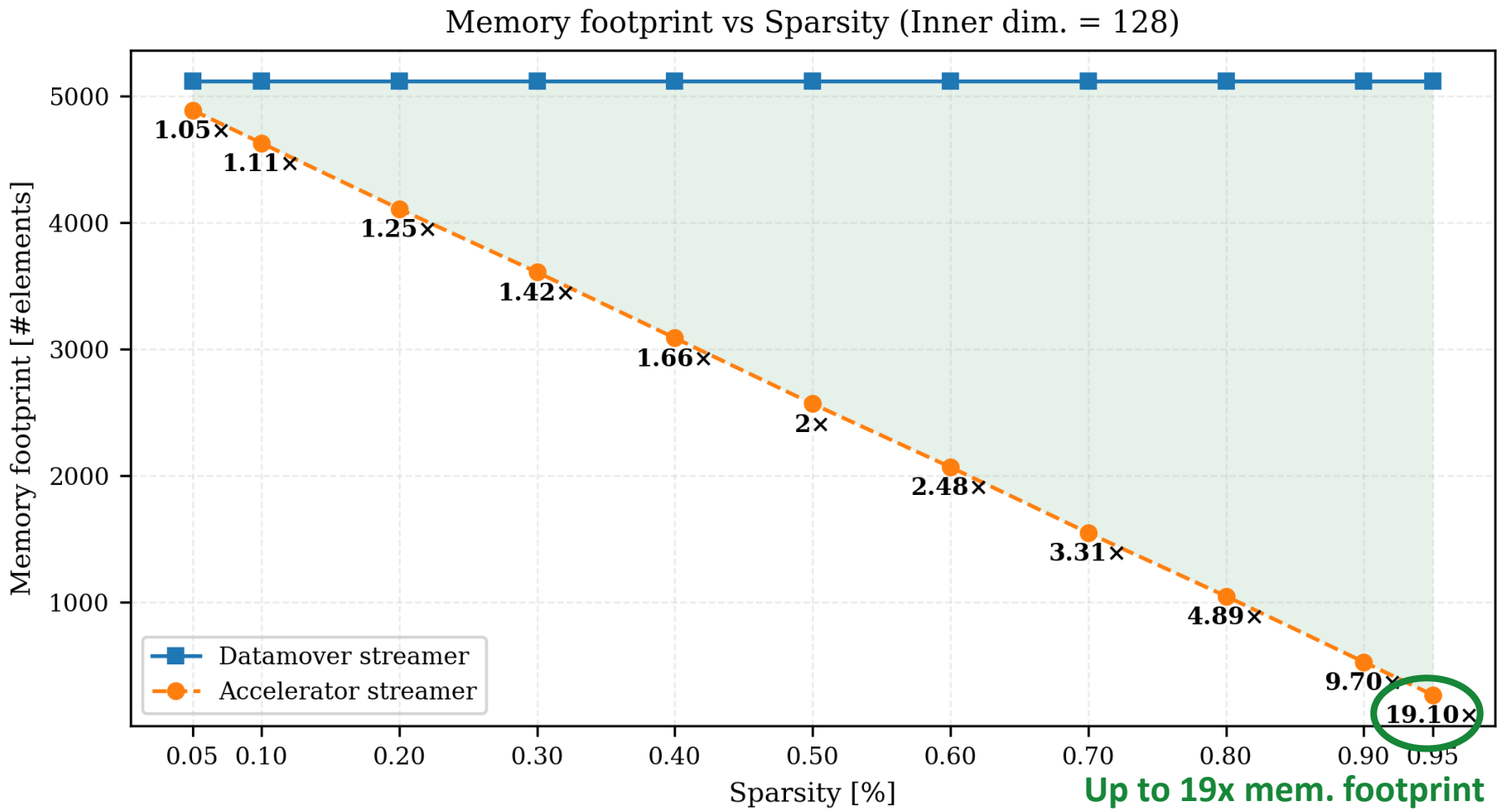
- **Assumptions:**
  - MAC units = 4
  - X rows = 2
  - Y columns = 16
  - Inner dim. = 128
  - Metadata chunk = 128b
  - Input data size = 32b
  - Output data size = 64b
  - BW = 128b



Up to 15x speedup with 95% sparsity

RTL results –Memory footprint VS sparsity

- **Assumptions:**
  - MAC units = 4
  - X rows = 2
  - Y columns = 16
  - Inner dim. = 128
  - Metadata chunk = 128b
  - Input data size = 32b
  - Output data size = 64b
  - BW = 128b



Up to 19x mem. footprint decrease with 95% sparsity

ETH zürich

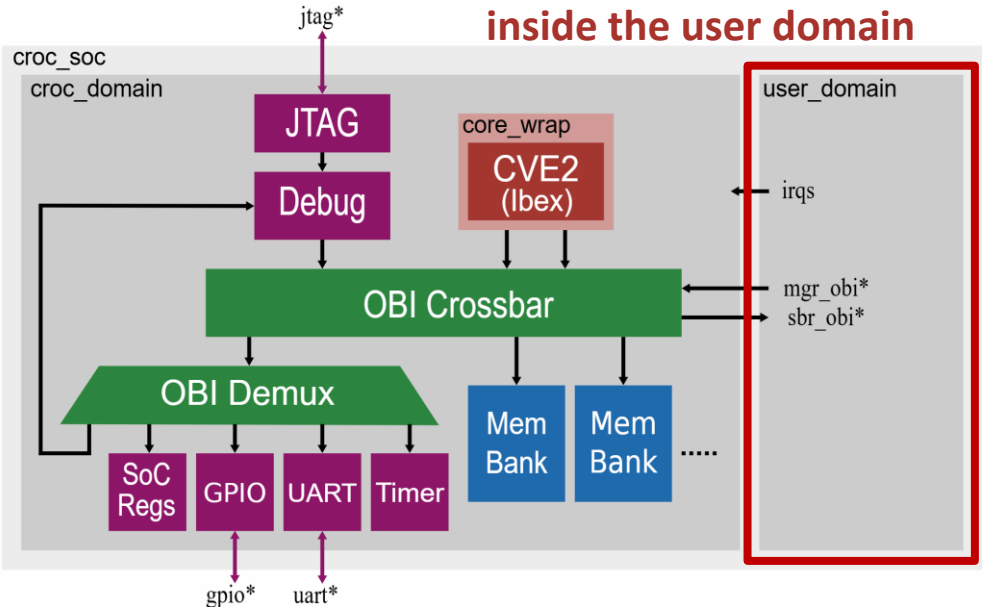
Expanding an existing SoC

E-Bee-P-Chip -Overview

- **Context**
  - **VLSI2:** two-people project work
  - **Scope:** learn the full stack of an ASIC workflow
- **Goal**
  - Improve an existing SoC (Croc) by adding something
- **Workflow**
  - **Implement RTL:** implemented an accelerator for data compression (EBPC), increased RAM size
  - **Functional verification:** verified the functionality of the RTL with a C testbench
  - **Physical implementation:** complete backend of the ASIC
  - **Results:** comparison with the base SoC to measure speedup, area and power analysis

Logo of our chip

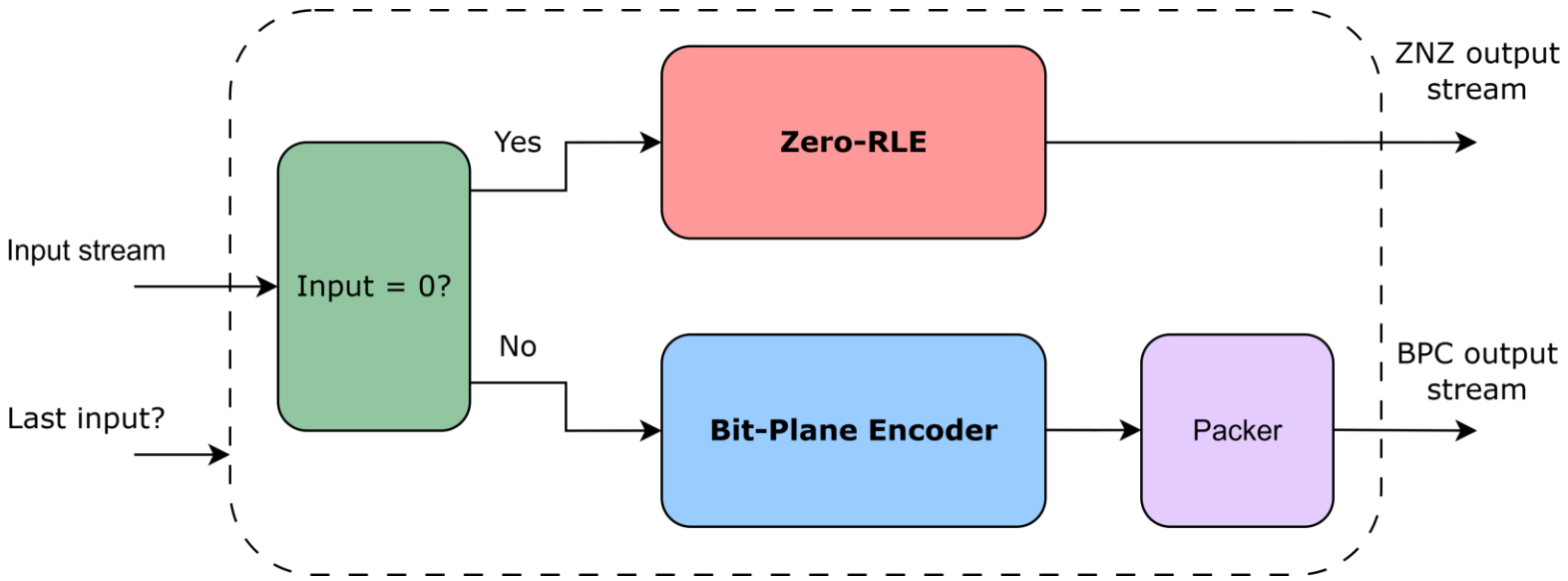
The module was implemented inside the user domain



Structure of the Croc SoC

E-Bee-P-Chip -EBPC

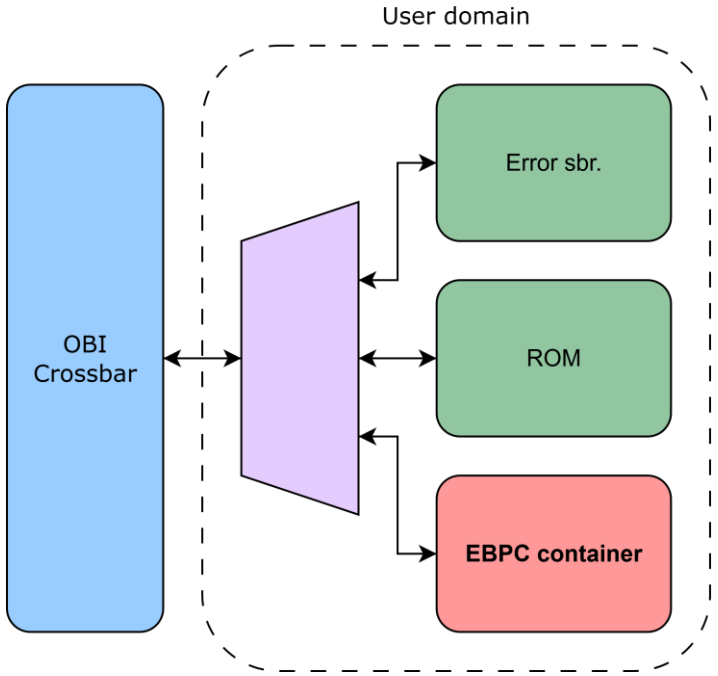
- **Ideal conditions**
  - A stream of 16-bit data items that are both sparse and correlated
- **Two modules**
  - **ZRLE:** compresses data items equal to 0
  - **BPC:** compresses nonzero-correlated data



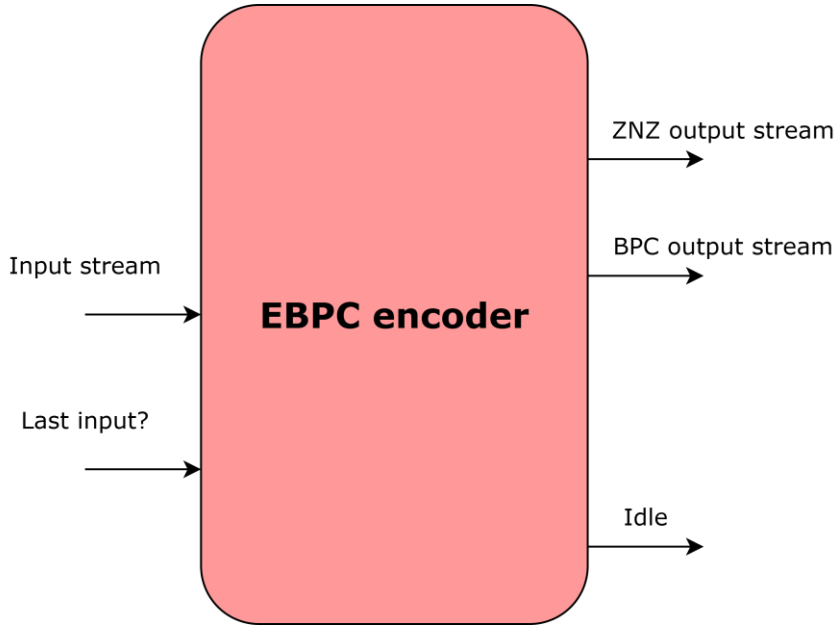
Structure of the Croc SoC

E-Bee-P-Chip –RTL implementation

- **We took a ready-to use IP from this [Github Repository]**
- **Implemented inside the user domain**
  - Inside a container that decodes the addresses of the OBI protocol
  - The container is connected to a sbr. Demux



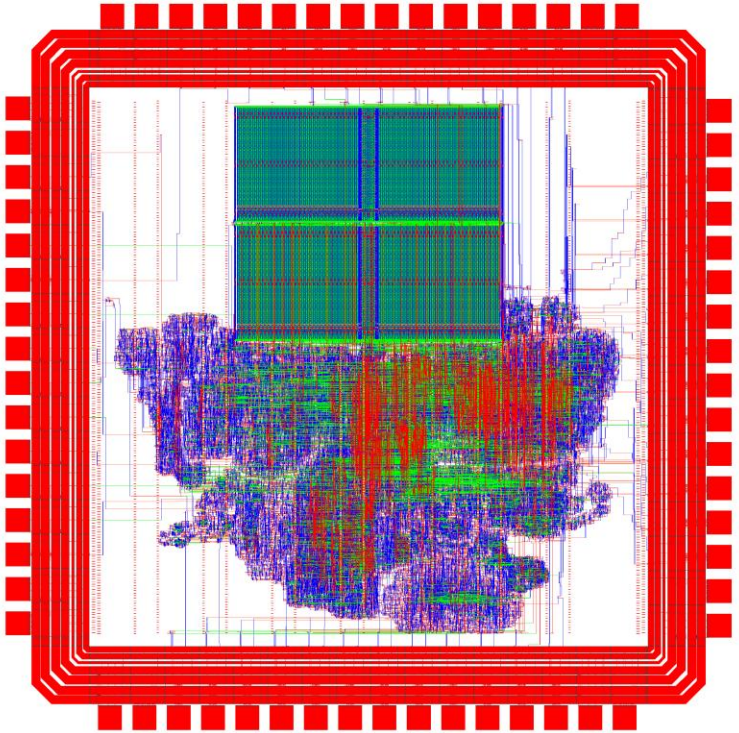
Integration of the module in the User Domain



Integration of the module in the User Domain

E-Bee-P-Chip –Backend

- **Synthesis**
  - Yosys
- **Physical backend**
  - Openroad
- **RAM banks**
  - From 2x2KB to 2x8KB
- **Clock frequency**
  - From 72MHz to 70MHz



Final layout

	Active area [mm^2]	Core area [mm^2]	Core utilization
Croc	0.78	1.76	44.4%
E-Bee-P-Chip	1.25	2.51	49.6%

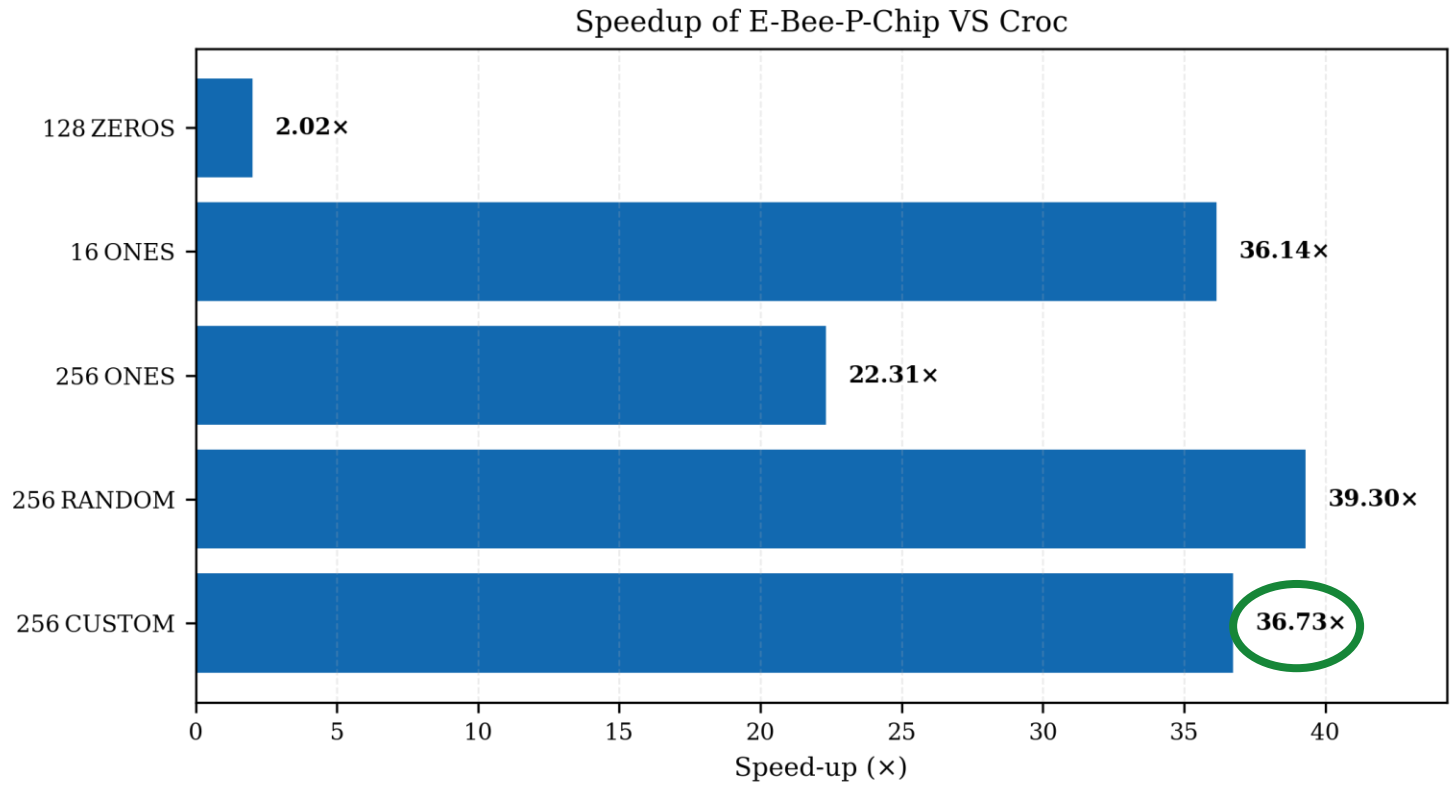
Area results

	Power [mW]
Croc	51.0
E-Bee-P-Chip	51.7

Power results

E-Bee-P-Chip –Benchmarks

- **Method**
  - C testbench that uses our accelerator VS C testbench that only uses core
- **Different use cases**
  - Custom is made of 256 16-bit sparse and correlated data items



Up to 37x speedup for the realistic case