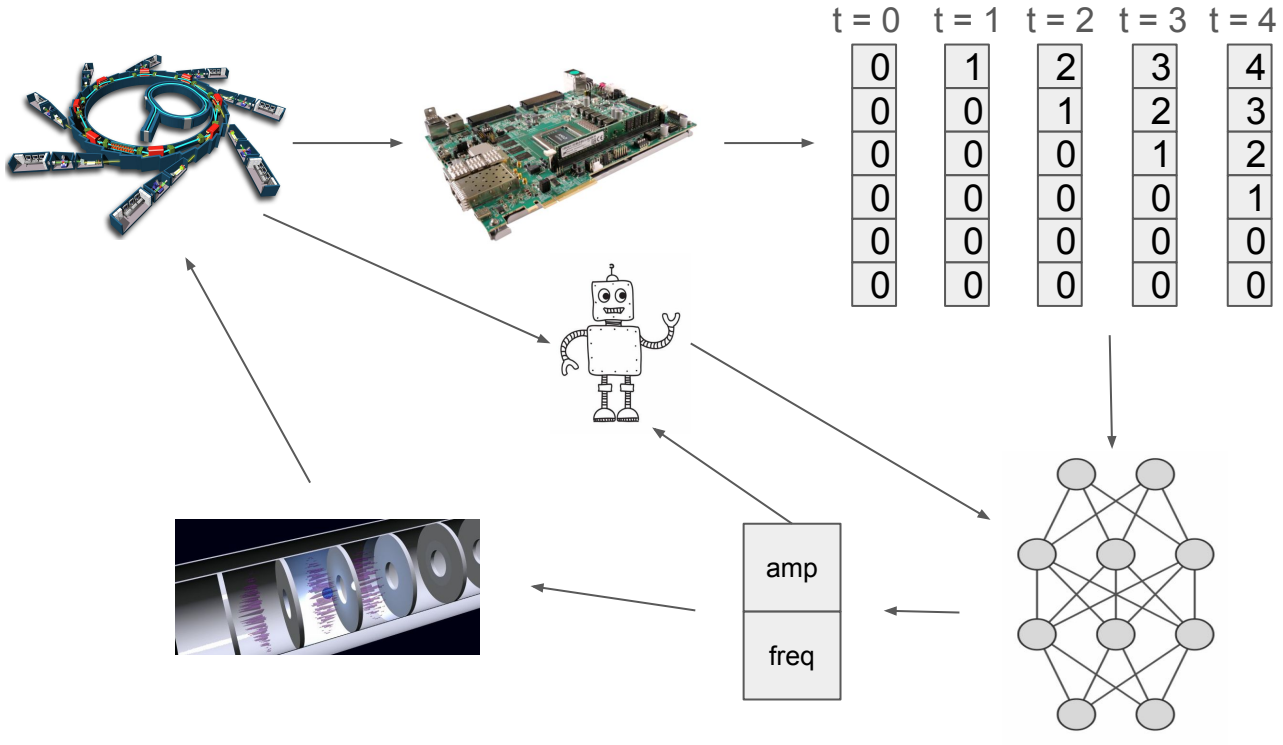
A low latency Gated Recurrent Unit Implementation for the AMD Versal AI Engine

Michail Sapkas, University of Padova

2nd FPGA Developers' Forum meeting
May 20 – 23, 2025 – CERN

The idea:

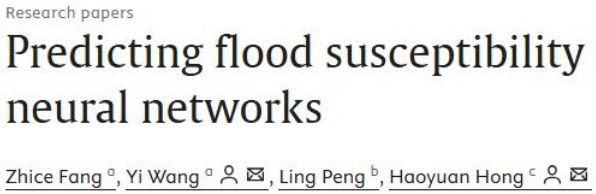


We need:

- To be faster than the timescale of the physics ~ tens of μs
- Our model must be sophisticated enough to capture time correlations and keep memory of previous modulations

The RNNs success story (the LSTM)

- Time series **analysis** / **prediction**

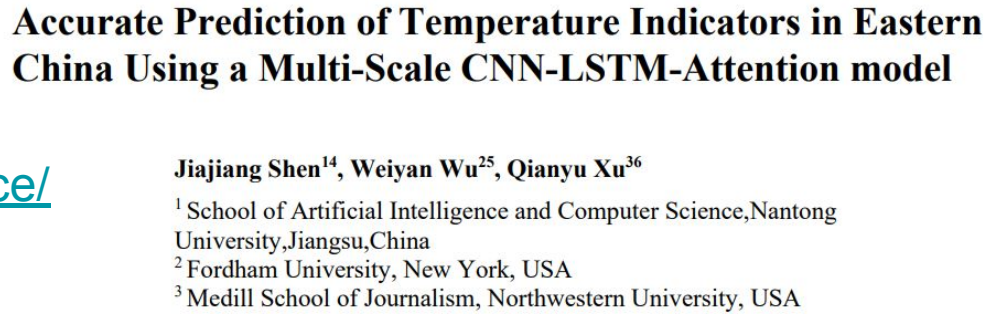


Research papers

Predicting flood susceptibility using LSTM neural networks

Zhice Fang [a], Yi Wang [a], Ling Peng [b], Haoyuan Hong [c]

https://www.sciencedirect.com/science/article/pii/S0022169420311951



Accurate Prediction of Temperature Indicators in Eastern China Using a Multi-Scale CNN-LSTM-Attention model

Jiajiang Shen[14], Weiyan Wu[25], Qianyu Xu[36]

[1] School of Artificial Intelligence and Computer Science,Nantong University,Jiangsu,China
[2] Fordham University, New York, USA
[3] Medill School of Journalism, Northwestern University, USA

https://arxiv.org/pdf/2412.07997

The RNNs success story

- Multiple “Generative AI” applications in **audio**, **imaging**, **text**

Generating Image Sequence from Description with LSTM Conditional GAN

Xu Ouyang[†], Xi Zhang[‡], Di Ma, Gady Agam
Illinois Institute of Technology
Chicago, IL 60616
{xouyang3, xzhang22, dma2}@hawk.iit.edu, agam@iit.edu

https://arxiv.org/abs/1806.03027

Google’s Neural Machine Translation System: Bridging the Gap between Human and Machine Translation

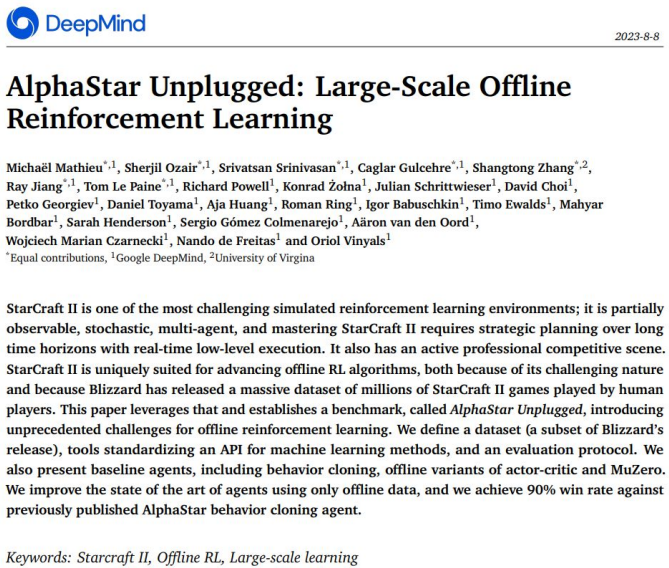
Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi
yonghui,schuster,zhifengc,qvl,mnorouzi@google.com

Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, Jeffrey Dean

https://arxiv.org/abs/1609.08144

The RNNs success story

- Reinforcement Learning in gaming



DeepMind

2023-8-8

AlphaStar Unplugged: Large-Scale Offline Reinforcement Learning

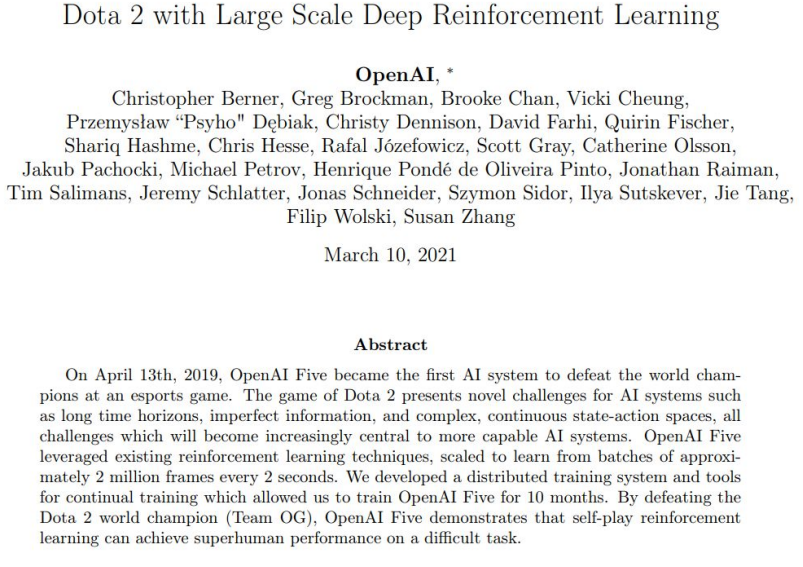
Michaël Mathieu[*,1], Sherjil Ozair[*,1], Srivatsan Srinivasan[*,1], Caglar Gulcehre[*,1], Shangtong Zhang[*,2], Ray Jiang[*,1], Tom Le Paine[*,1], Richard Powell[1], Konrad Żołna[1], Julian Schrittwieser[1], David Choi[1], Petko Georgiev[1], Daniel Toyama[1], Aja Huang[1], Roman Ring[1], Igor Babuschkin[1], Timo Ewalds[1], Mahyar Bordbar[1], Sarah Henderson[1], Sergio Gómez Colmenarejo[1], Aäron van den Oord[1], Wojciech Marian Czarnecki[1], Nando de Freitas[1] and Oriol Vinyals[1]

[*]Equal contributions, [1]Google DeepMind, [2]University of Virgina

StarCraft II is one of the most challenging simulated reinforcement learning environments; it is partially observable, stochastic, multi-agent, and mastering StarCraft II requires strategic planning over long time horizons with real-time low-level execution. It also has an active professional competitive scene. StarCraft II is uniquely suited for advancing offline RL algorithms, both because of its challenging nature and because Blizzard has released a massive dataset of millions of StarCraft II games played by human players. This paper leverages that and establishes a benchmark, called *AlphaStar Unplugged*, introducing unprecedented challenges for offline reinforcement learning. We define a dataset (a subset of Blizzard's release), tools standardizing an API for machine learning methods, and an evaluation protocol. We also present baseline agents, including behavior cloning, offline variants of actor-critic and MuZero. We improve the state of the art of agents using only offline data, and we achieve 90% win rate against previously published AlphaStar behavior cloning agent.

Keywords: Starcraft II, Offline RL, Large-scale learning

https://arxiv.org/abs/2308.03526



Dota 2 with Large Scale Deep Reinforcement Learning

OpenAI, *
Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław "Psyho" Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique Pondé de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, Susan Zhang

March 10, 2021

Abstract

On April 13th, 2019, OpenAI Five became the first AI system to defeat the world champions at an esports game. The game of Dota 2 presents novel challenges for AI systems such as long time horizons, imperfect information, and complex, continuous state-action spaces, all challenges which will become increasingly central to more capable AI systems. OpenAI Five leveraged existing reinforcement learning techniques, scaled to learn from batches of approximately 2 million frames every 2 seconds. We developed a distributed training system and tools for continual training which allowed us to train OpenAI Five for 10 months. By defeating the Dota 2 world champion (Team OG), OpenAI Five demonstrates that self-play reinforcement learning can achieve superhuman performance on a difficult task.

https://arxiv.org/abs/1912.06680

The RNNs success story

- as Controllers for Nuclear Fusion

https://www.sciencedirect.com/science/article/abs/pii/S0920379620306554

Review

Modeling and control of plasma horizontal displacement for HL-2A tokamak based on LSTM

Xianfei Fu [a], Bin Yang [a b], Shiqing Wang [a b]

Article | Open access | Published: 16 February 2022

Magnetic control of tokamak plasmas through deep reinforcement learning

Jonas Degrave, Federico Felici, Jonas Buchli, Michael Neunert, Brendan Tracey, Francesco Carpanese, Timo Ewalds, Roland Hafner, Abbas Abdolmaleki, Diego de las Casas, Craig Donner, Leslie Fritz, Cristian Galperti, Andrea Huber, James Keeling, Maria Tsimpoukelli, Jackie Kay, Antoine Merle, Jean-Marc Moret, Seb Noury, Federico Pesamosca, David Pfau, Olivier Sauter, Cristian Sommariva, ... Martin Riedmiller + Show authors

Nature **602**, 414–419 (2022) | Cite this article

https://www.nature.com/articles/s41586-021-04301-9

Physics > Plasma Physics

[Submitted on 21 Jul 2023 (v1), last revised 5 Oct 2023 (this version, v2)]

Towards practical reinforcement learning for tokamak magnetic control https://arxiv.org/abs/2307.11546

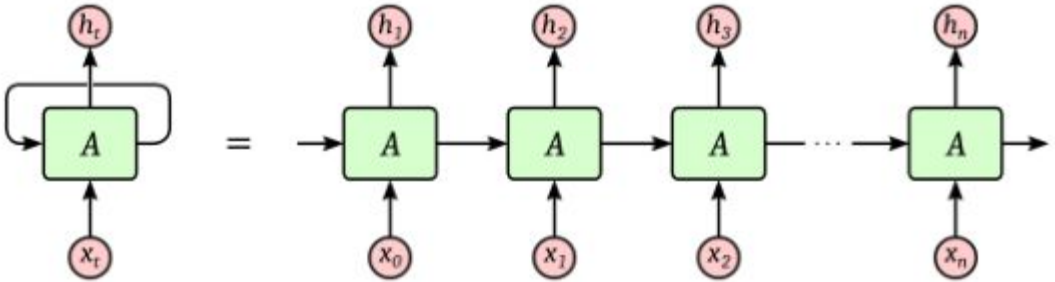
Brendan D. Tracey, Andrea Michi, Yuri Chervonyi, Ian Davies, Cosmin Paduraru, Nevena Lazic, Federico Felici, Ti Kurylowicz, Daniel J. Mankowitz, Martin Riedmiller, The TCV Team

Possible cases

- Particle track reconstruction
- Anomaly detection
- Classification

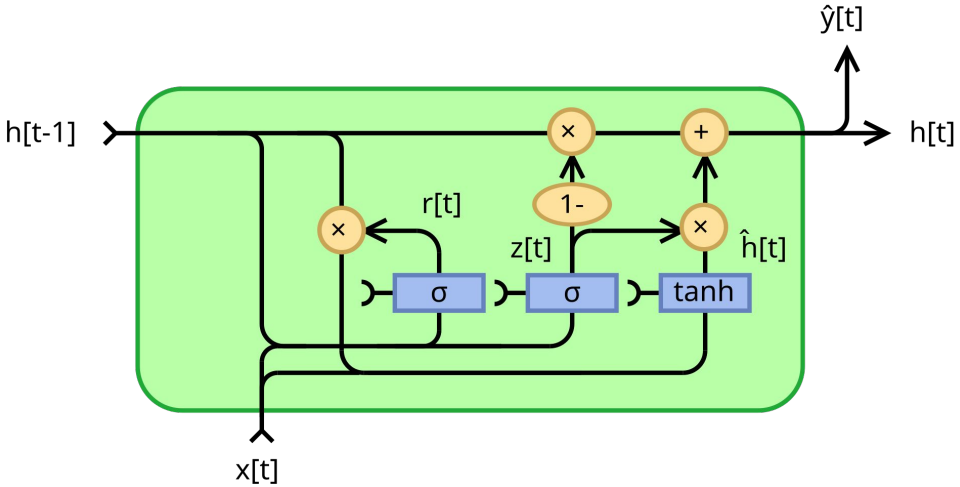
The Gated Recurrent Unit (LSTMs little cousin)

- This is the “temporal unrolled view” for RNNs



- The “hidden state” is an encoding of observations through time, some may even call it “memory”. At each time step (or after many), it can be fed to a Feed Forward NN to do regression or classification.

The Gated Recurrent Unit



$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z)$$
$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r)$$
$$\hat{h}_t = \phi(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h)$$
$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \hat{h}_t$$

The Gated Recurrent Unit

- These are called gates

$$
\begin{aligned}
z_t &= \sigma(W_z x_t + U_z h_{t-1} + b_z) \\
r_t &= \sigma(W_r x_t + U_r h_{t-1} + b_r) \\
\hat{h}_t &= \phi(W_h x_t + U_h (r_t \odot h_{t-1}) + b_h) \\
h_t &= (1 - z_t) \odot h_{t-1} + z_t \odot \hat{h}_t
\end{aligned}
$$

The Gated Recurrent Unit

- Matrix - Vector Multiplication, which in general, **W** and **U** not of same dimensions

$$
\begin{aligned}
z_t &= \sigma(W_z x_t + U_z h_{t-1} + b_z) \\
r_t &= \sigma(W_r x_t + U_r h_{t-1} + b_r) \\
\hat{h}_t &= \phi(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h) \\
h_t &= (1 - z_t) \odot h_{t-1} + z_t \odot \hat{h}_t
\end{aligned}
$$

The Gated Recurrent Unit

- Matrix - Vector Multiplication
- Activation Functions (sigmoid - tanh)

$$
\begin{aligned}
z_t &= \sigma(W_z x_t + U_z h_{t-1} + b_z) \\
r_t &= \sigma(W_r x_t + U_r h_{t-1} + b_r) \\
\hat{h}_t &= \phi(W_h x_t + U_h (r_t \odot h_{t-1}) + b_h) \\
h_t &= (1 - z_t) \odot h_{t-1} + z_t \odot \hat{h}_t
\end{aligned}
$$

The Gated Recurrent Unit

- Matrix - Vector Multiplication
- Activation Functions (sigmoid - tanh)
- Element wise multiplication

$$
\begin{aligned}
z_t &= \sigma(W_z x_t + U_z h_{t-1} + b_z) \\
r_t &= \sigma(W_r x_t + U_r h_{t-1} + b_r) \\
\hat{h}_t &= \phi(W_h x_t + U_h (r_t \odot h_{t-1}) + b_h) \\
h_t &= (1 - z_t) \odot h_{t-1} + z_t \odot \hat{h}_t
\end{aligned}
$$

The Gated Recurrent Unit: Computational Dependencies

- Previous **hidden state** (or initialize it the first time)

$$
\begin{aligned}
z_t &= \sigma(W_z x_t + U_z h_{t-1} + b_z) \\
r_t &= \sigma(W_r x_t + U_r h_{t-1} + b_r) \\
\hat{h}_t &= \phi(W_h x_t + U_h (r_t \odot h_{t-1}) + b_h) \\
h_t &= (1 - z_t) \odot h_{t-1} + z_t \odot \hat{h}_t
\end{aligned}
$$

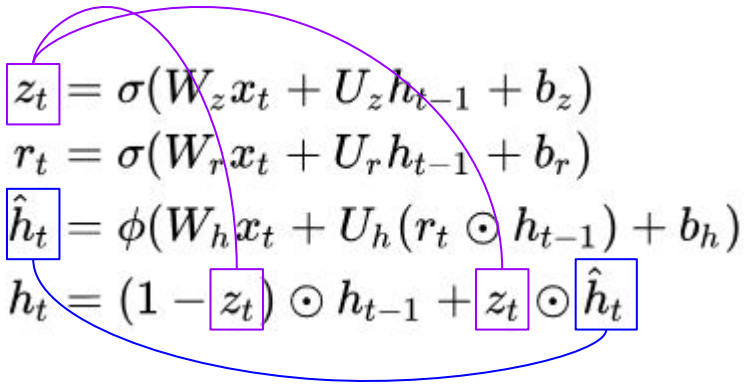
The Gated Recurrent Unit: Computational Dependencies

- **Candidate hidden state gate** needs **relevance gate (R) vector**

$$
\begin{aligned}
z_t &= \sigma(W_z x_t + U_z h_{t-1} + b_z) \\
r_t &= \sigma(W_r x_t + U_r h_{t-1} + b_r) \\
\hat{h}_t &= \phi(W_h x_t + U_h (r_t \odot h_{t-1}) + b_h) \\
h_t &= (1 - z_t) \odot h_{t-1} + z_t \odot \hat{h}_t
\end{aligned}
$$

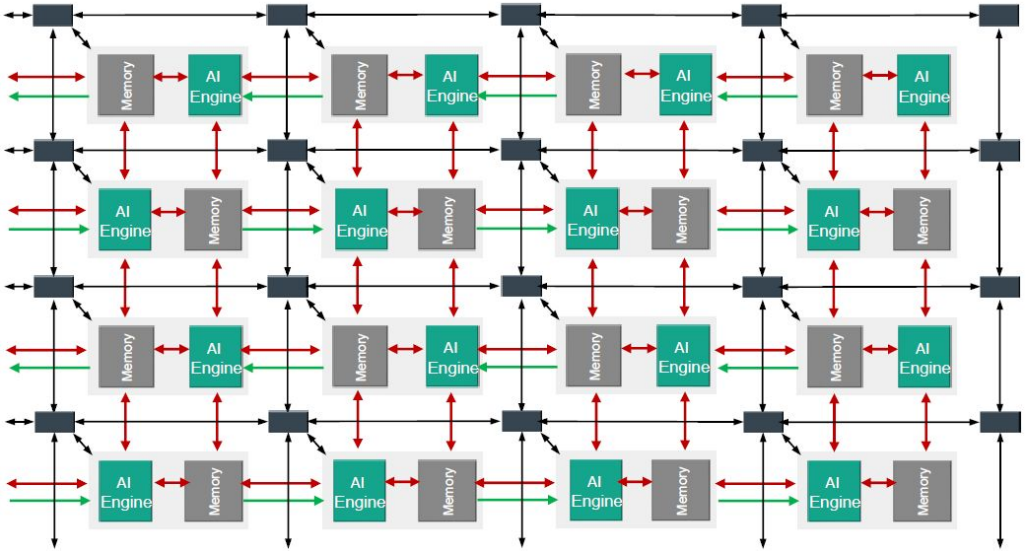
The Gated Recurrent Unit: Computational Dependencies

- **New hidden state gate** needs the **update gate (Z) vector AND**
- the **Candidate hidden state gate (h_hat) vector**

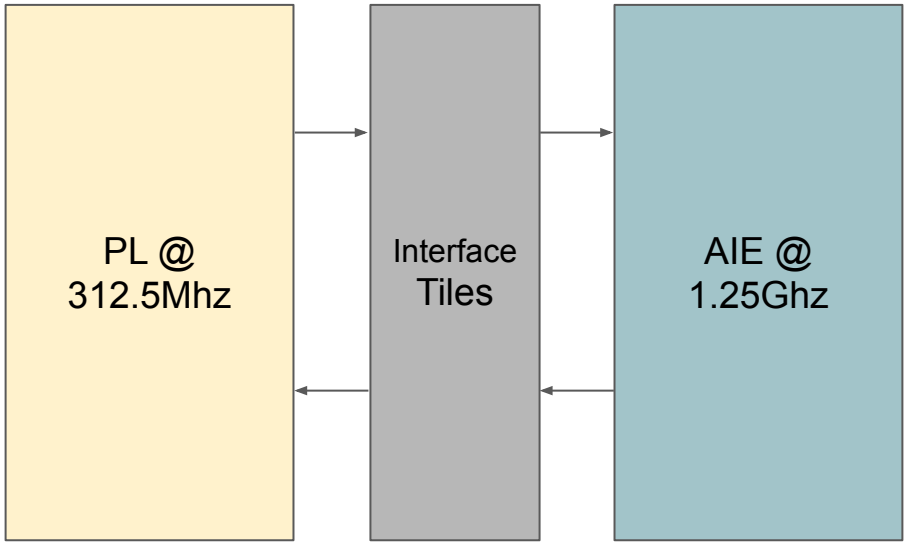


The Versal AIE

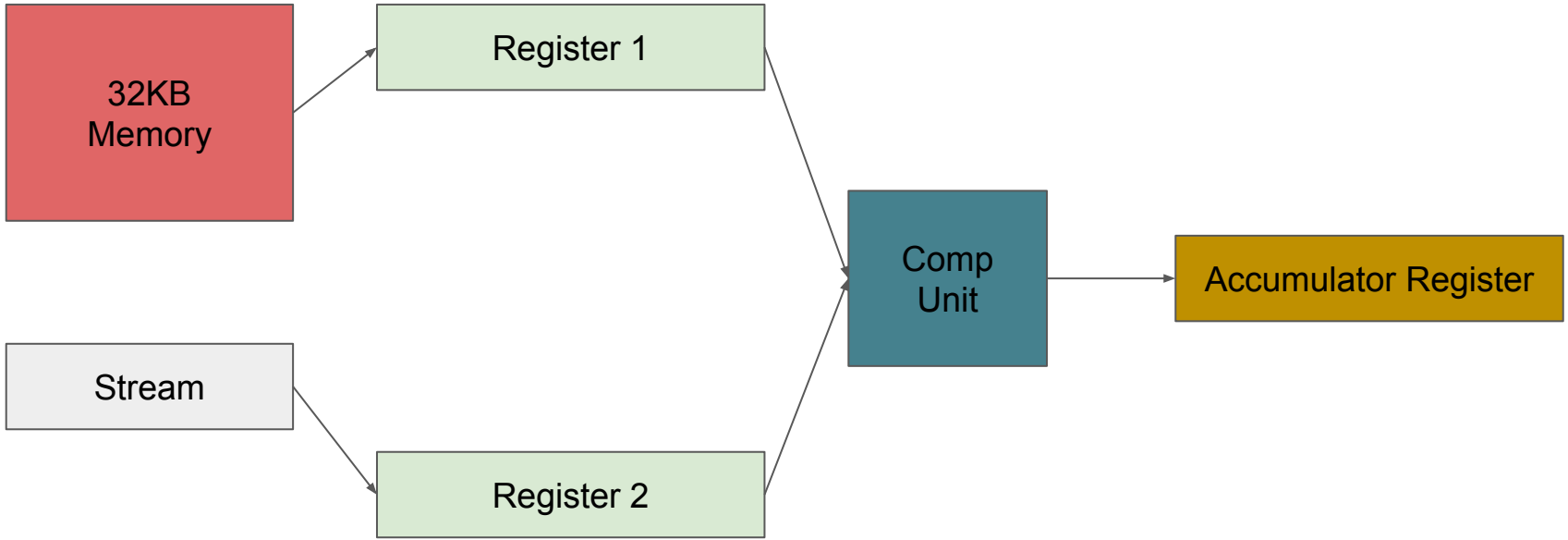
- 400 Vector Processors with local memory
- Interconnects / Cascade / Mem Share
- Everything written in C++ (chess compiler)



PL - AIE Connectivity

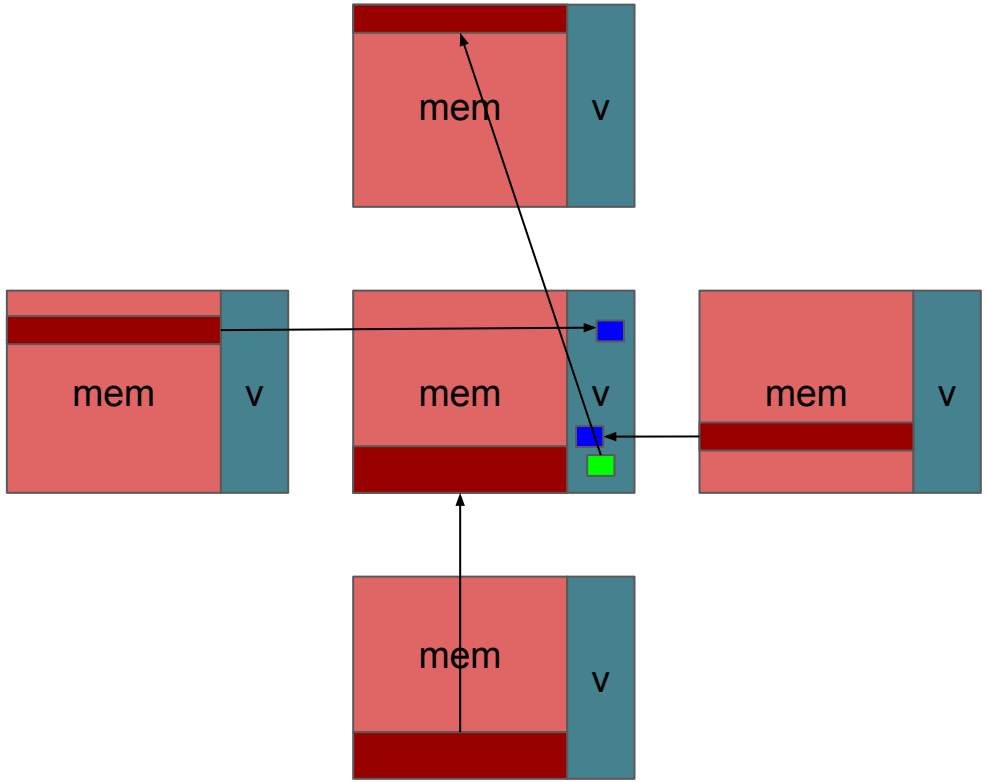


A (very) simplified inside of a AIE Tile



Tile - Tile Connectivity: Memory Sharing (Local)

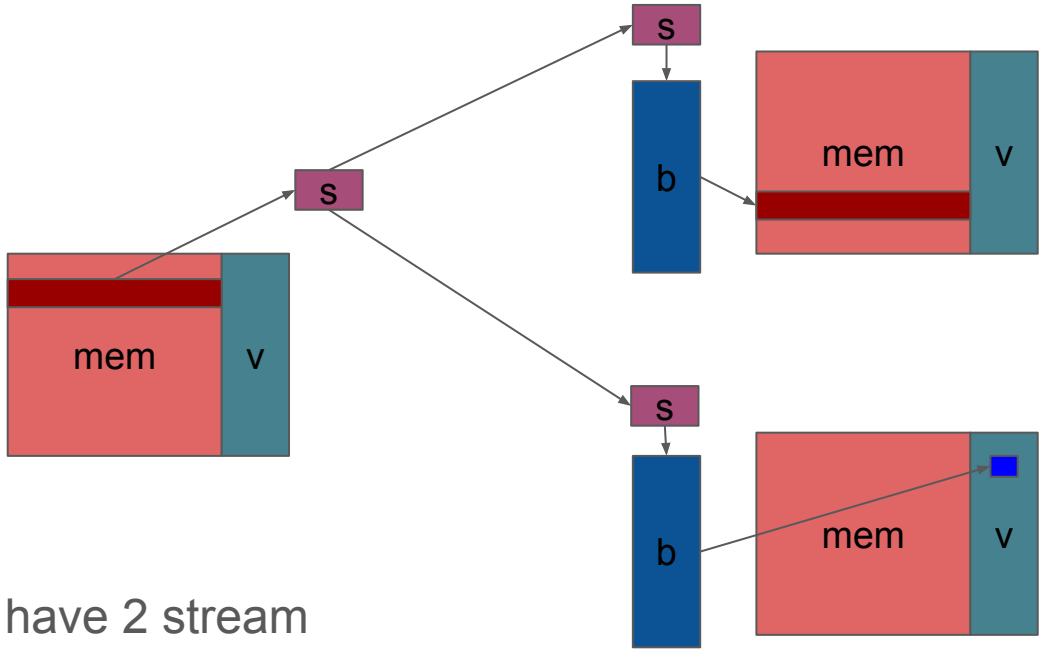
- Block of Memory banks
- Stored data
- Vectorized Processor
- VecProc Input Port
- VecProc Output Port



Tile - Tile Connectivity: AXI4 Stream Interconnect

Switch

Buffers

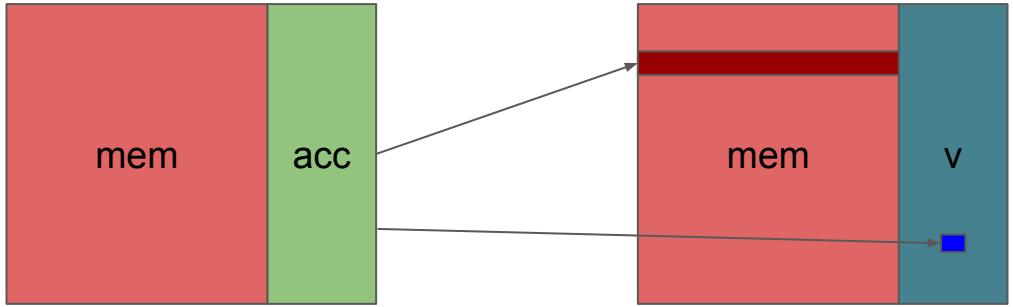


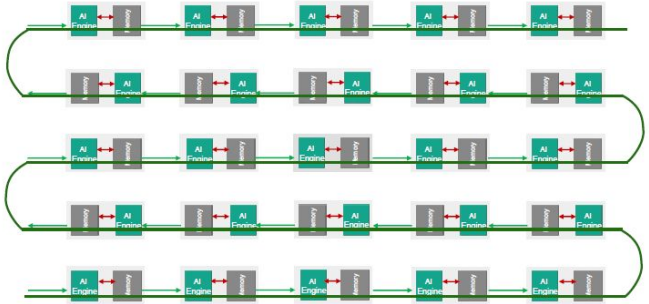
- Each tile can have 2 stream inputs and 2 stream outputs

Tile - Tile Connectivity: Cascade Stream (Local)

- Only neighbouring tiles (right for even, left for odds, up for edge)

Accumulator



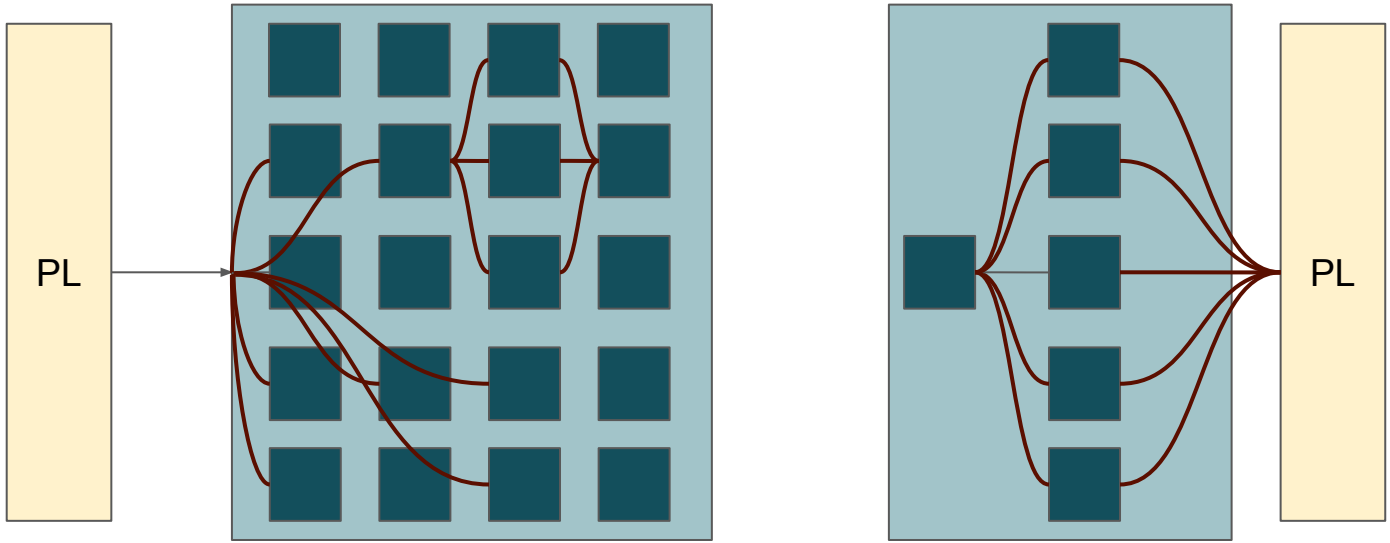


Packet Stream (split - merge)

- To use the same physical AXI4Stream for multiple source or destination
- Headers can either be sent from PL targeting a AIE Tile or from Tile to Tile

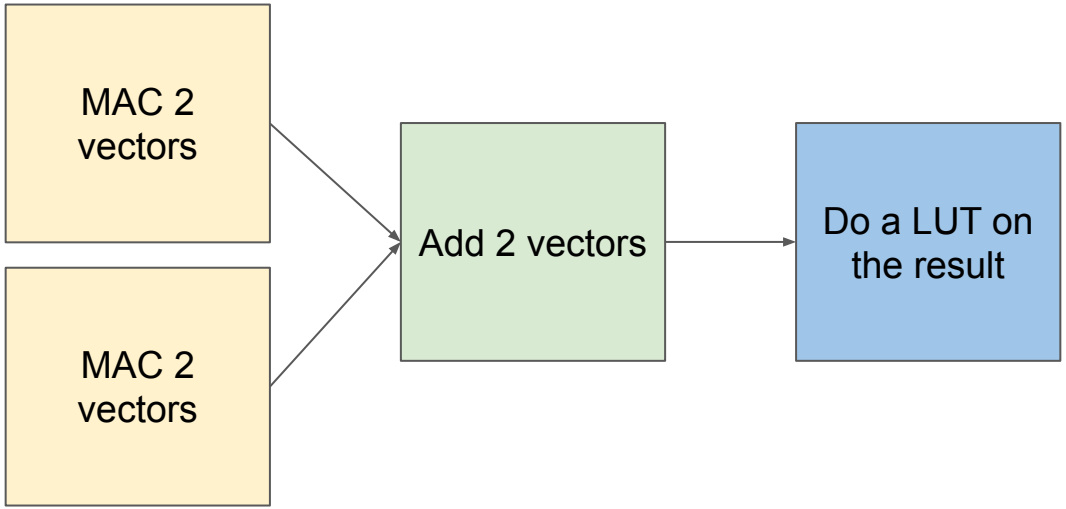
AIE Tile

Packet Stream Routings over AXI4Stream



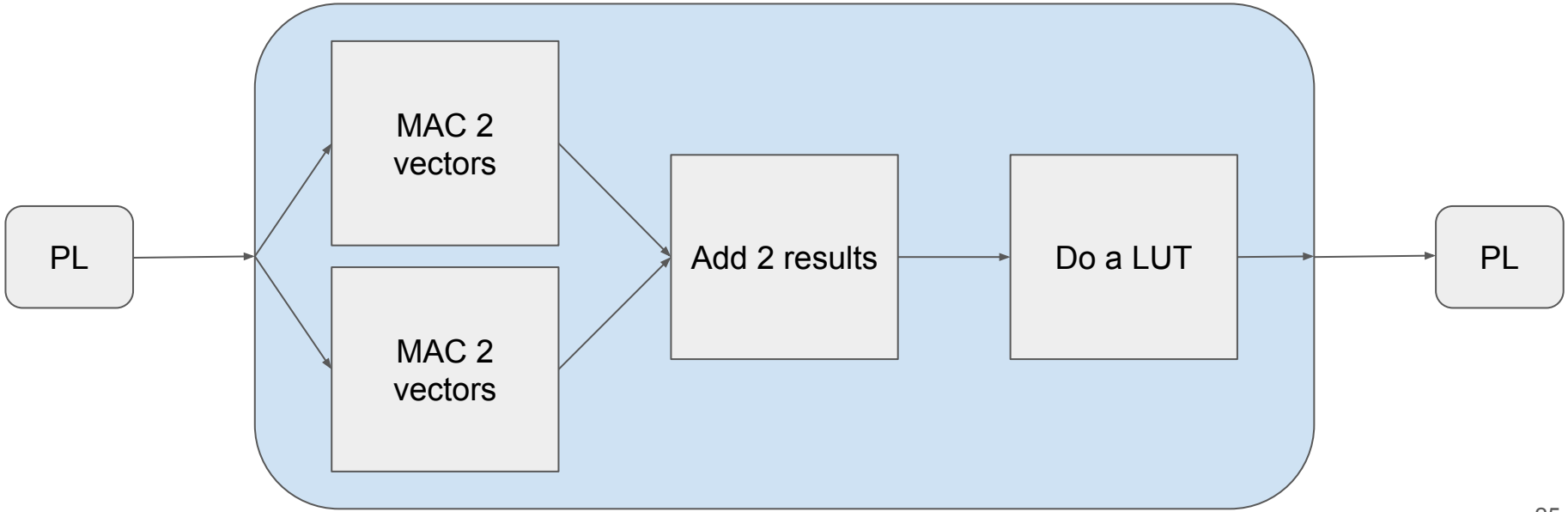
Programing paradigm

- You start with writing the program of a single tile called “kernel”
- A kernel is a function in C++ that also declares connectivity
- Chess compiler pragmas are essential

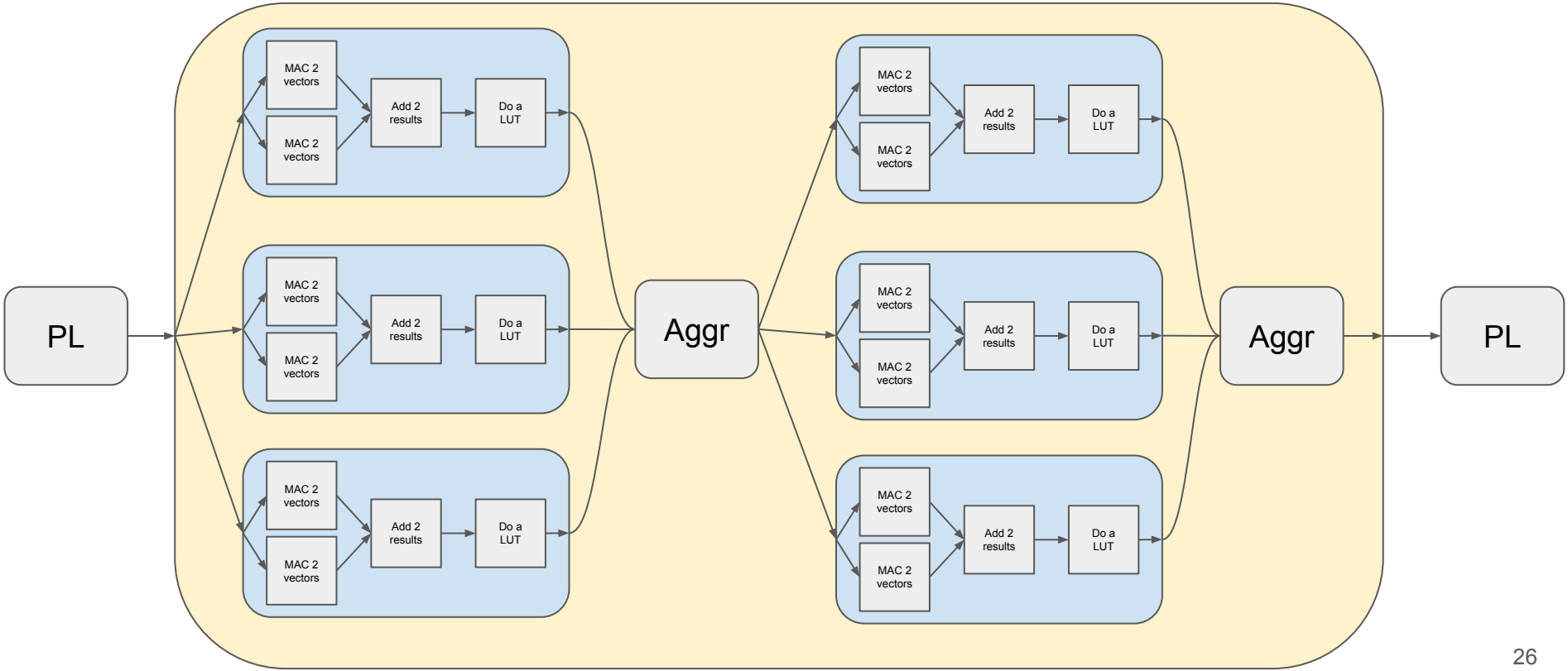


Organize multiple kernels into graphs

- Computational graphs are the top level abstractions that allow you to organize the execution of multiple kernels to obtain a result

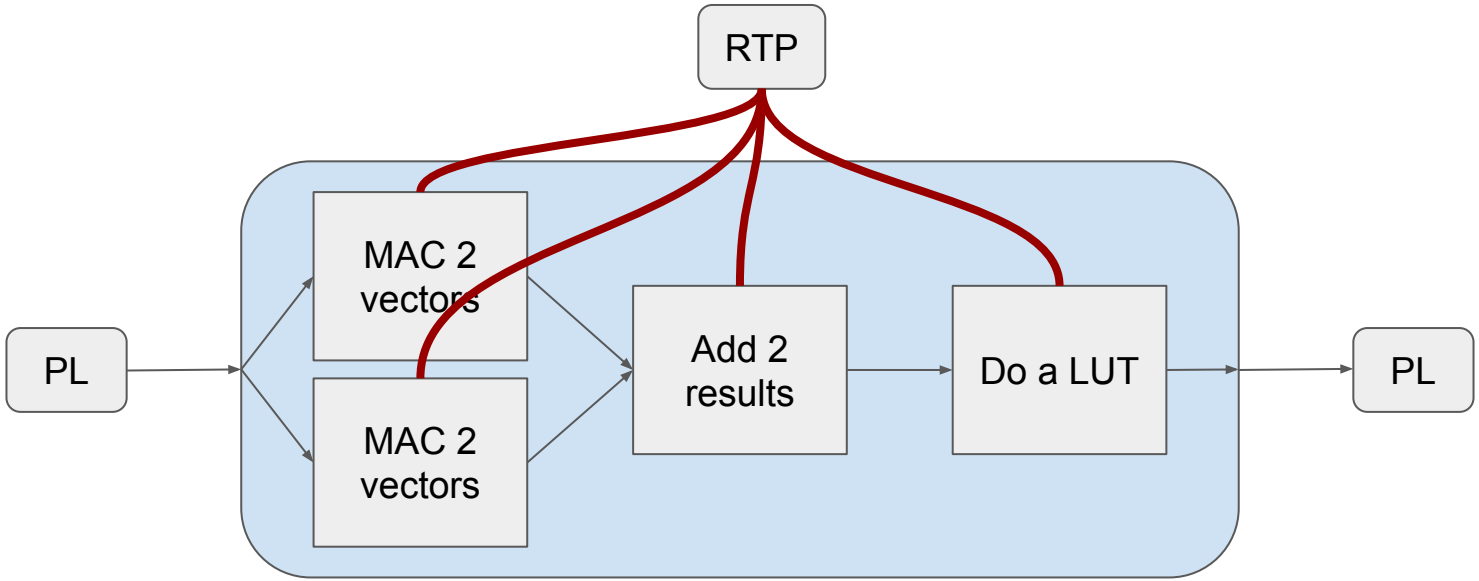


Organize into a top graph with multiple subgraphs



Run Time Parameters

- Passed to the memory of the AI Engine Tiles by the PS (txt file, shell, std::cin)
- Can be changed - read in run time



*** All numerical computations are done in FP32s ***

- No post-training quantization needed
- AIE native vector data types of **4 or 8 FP32s** “quantize” the MAC numerical operations as multiples of the vectors
- And therefore our matrices too are quantized
- In case of mismatch with the actual model: Padd with zeros

Distributive approach

- Let’s consider the operations of this gate

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z)$$

- The Matrix - Vector Operations can be calculated concurrently

$$W_z x_t \qquad\qquad U_z h_{t-1}$$

Distributive approach

- Let’s consider the operations of this gate

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z)$$

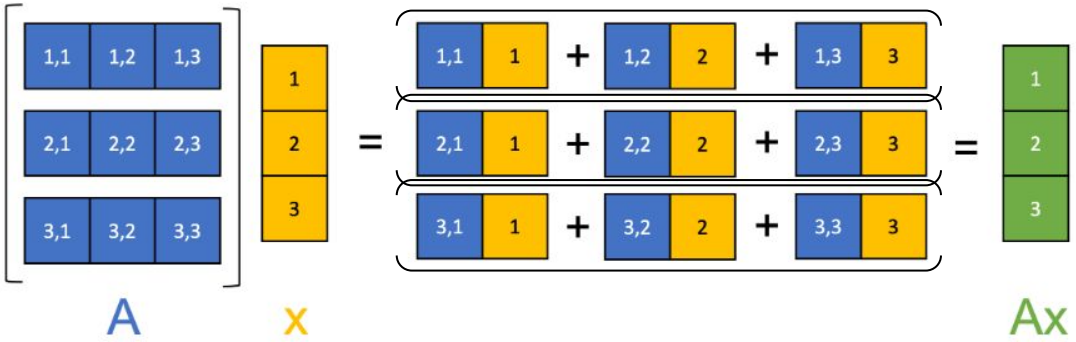
- The Matrix - Vector Operations can be calculated concurrently

$$W_z x_t \qquad\qquad U_z h_{t-1}$$

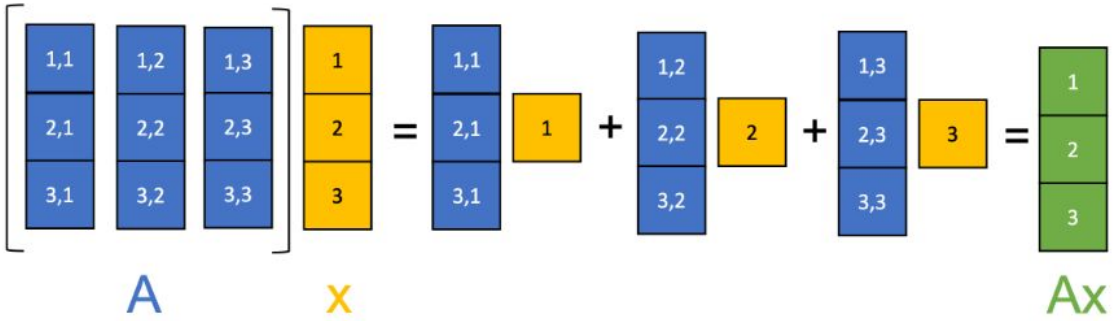
- The same applies for all these terms

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z)$$
$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r)$$
$$\hat{h}_t = \phi(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h)$$

How to perform Matrix - Vector multiplications

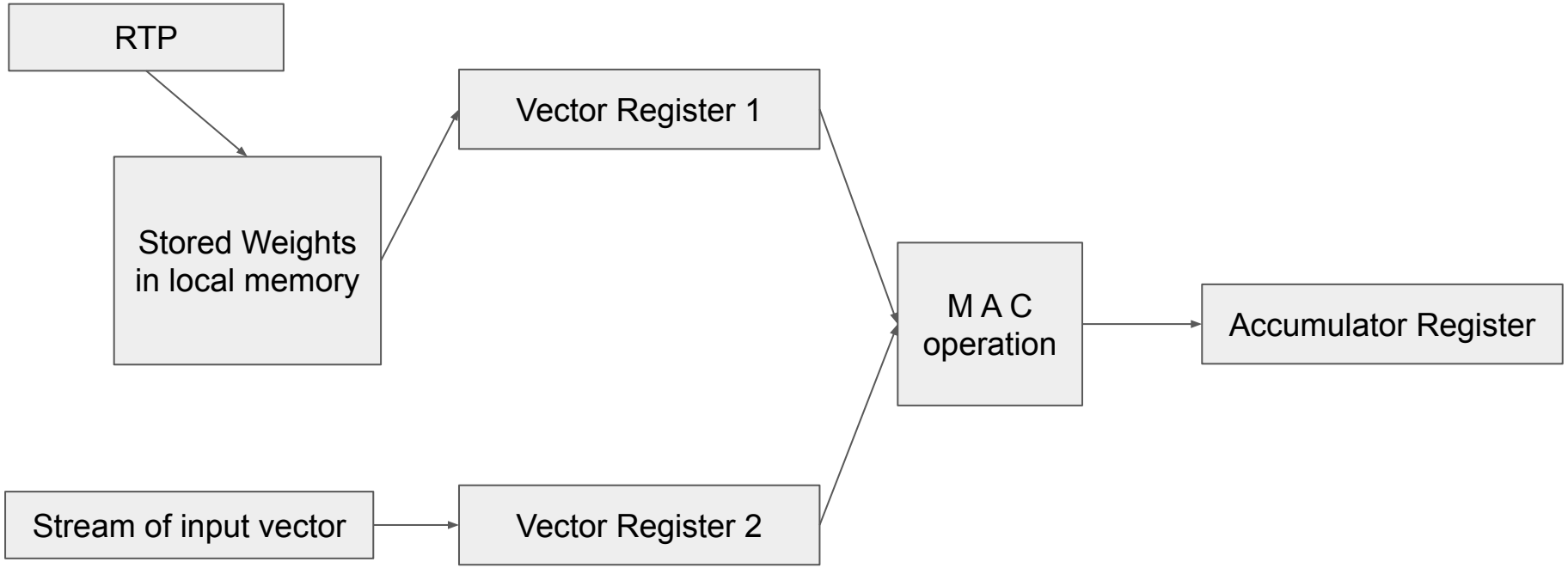


- Row wise you compute 1 element of the resulting vector

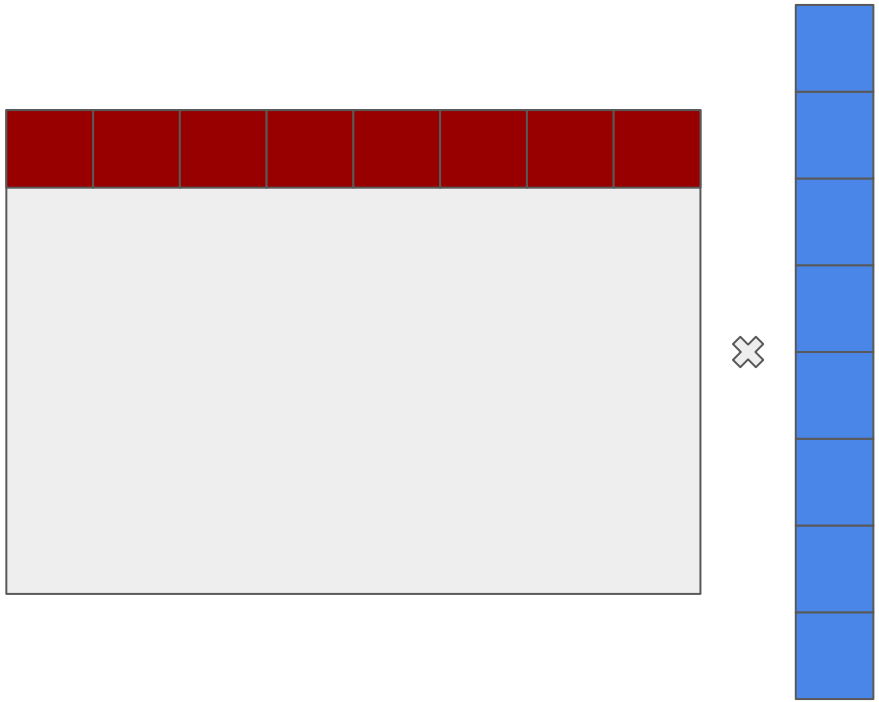


- Column wise you compute a partial column of the result

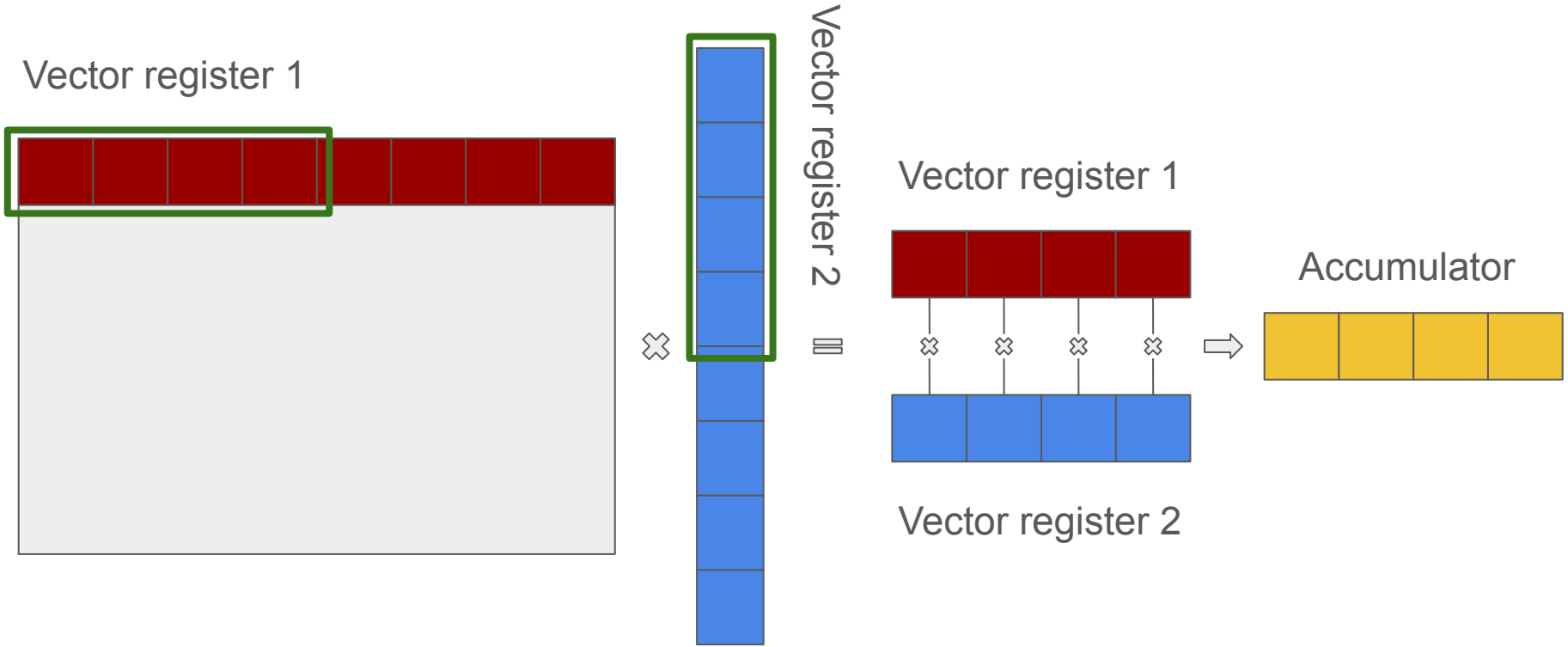
Remember? Inside a Tile



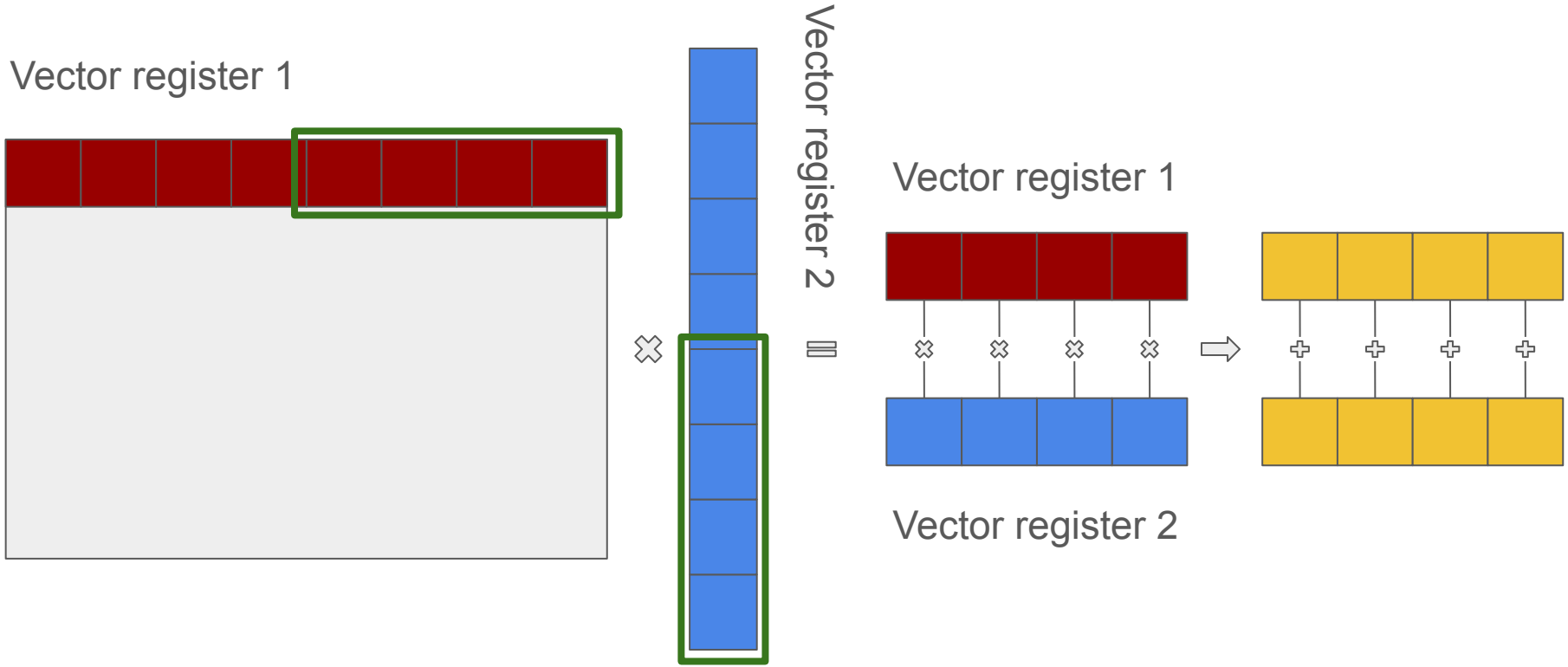
Multiply Accumulate: Rows



Multiply Accumulate: Rows

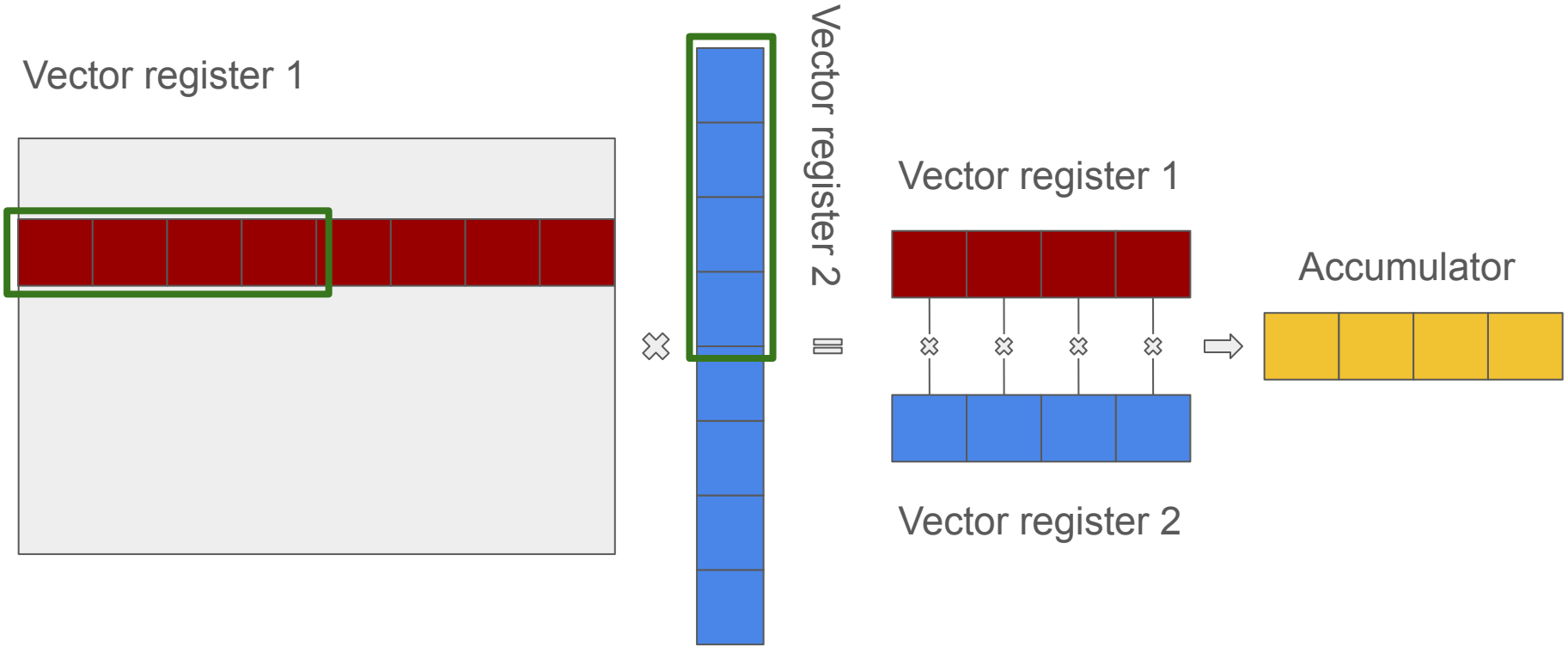


Multiply Accumulate: Rows

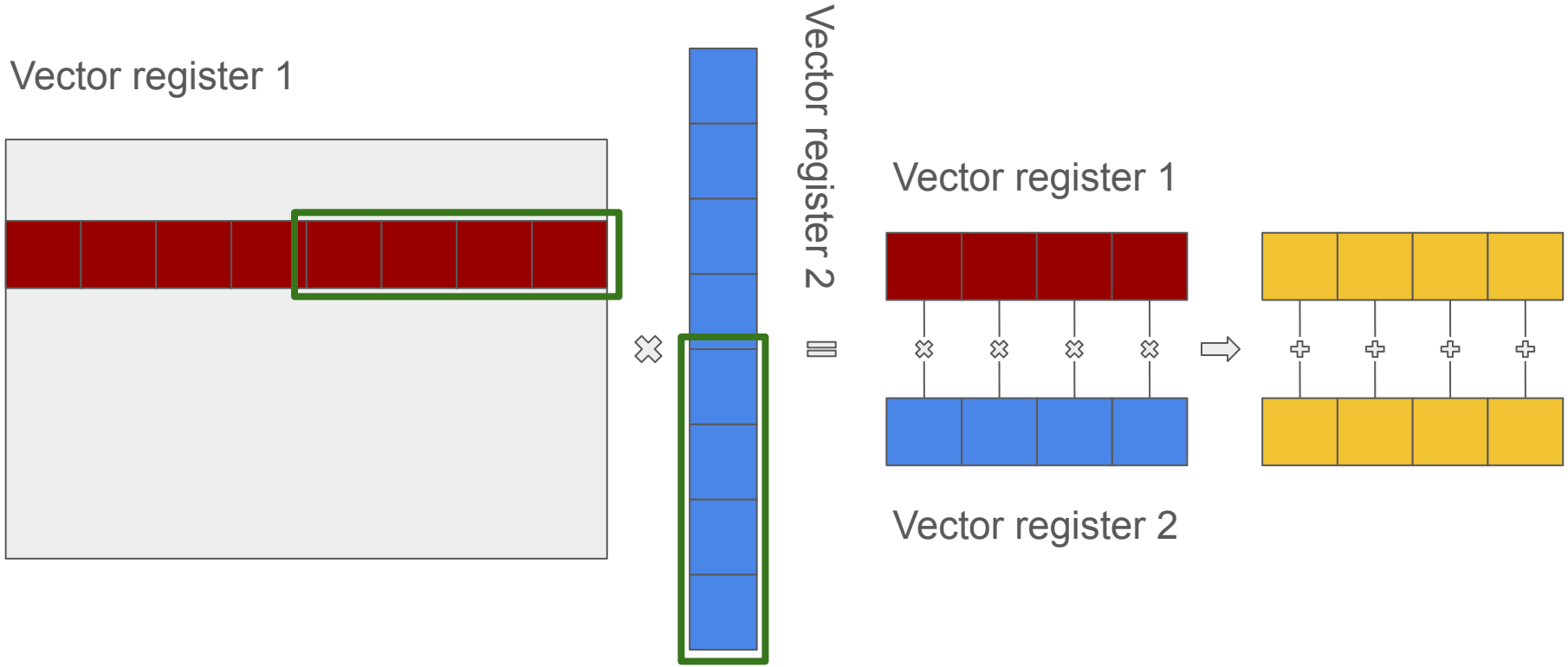


Multiply Accumulate: Rows

Multiply Accumulate: Rows

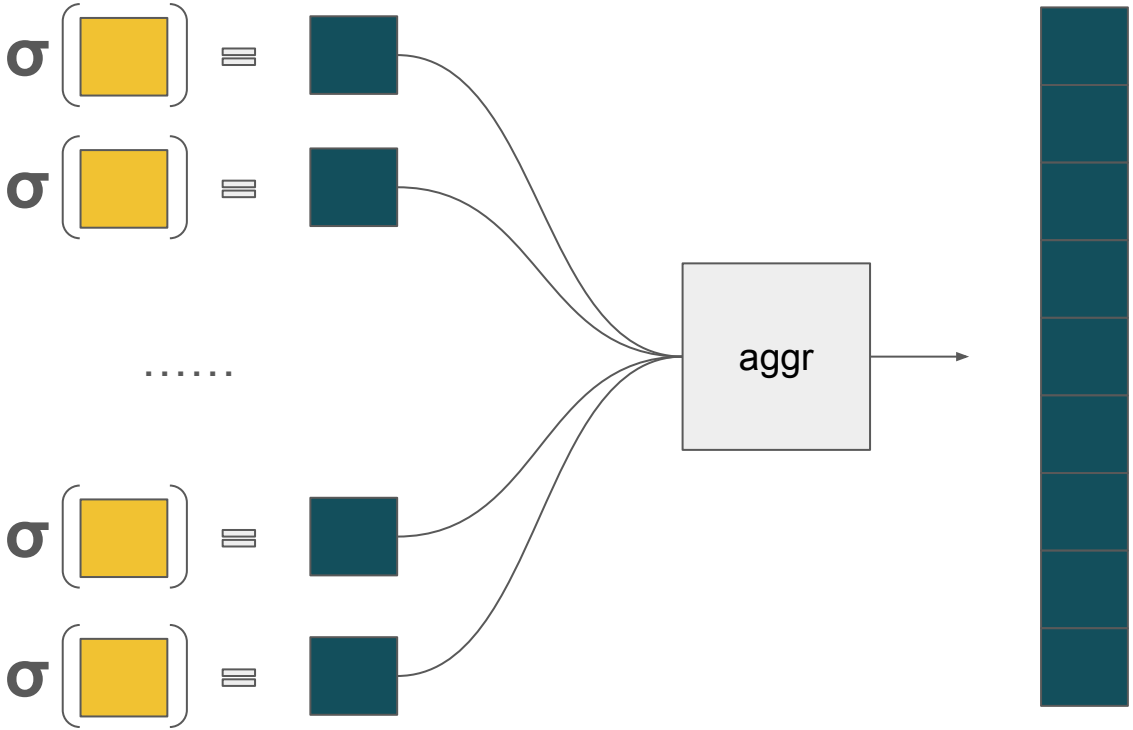


Multiply Accumulate: Rows



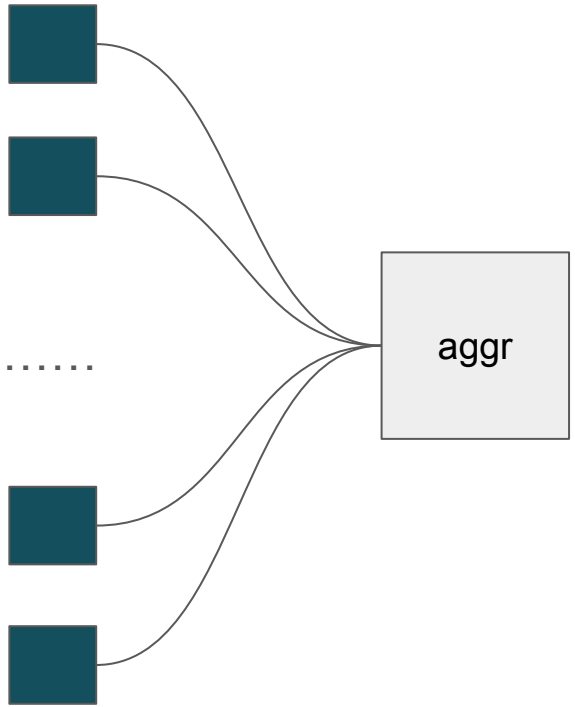
Multiply Accumulate: Rows

Aggregate partial results w Packet Stream: Merge

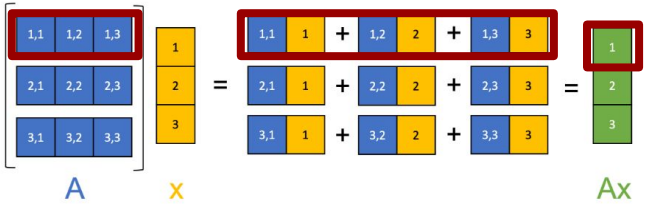


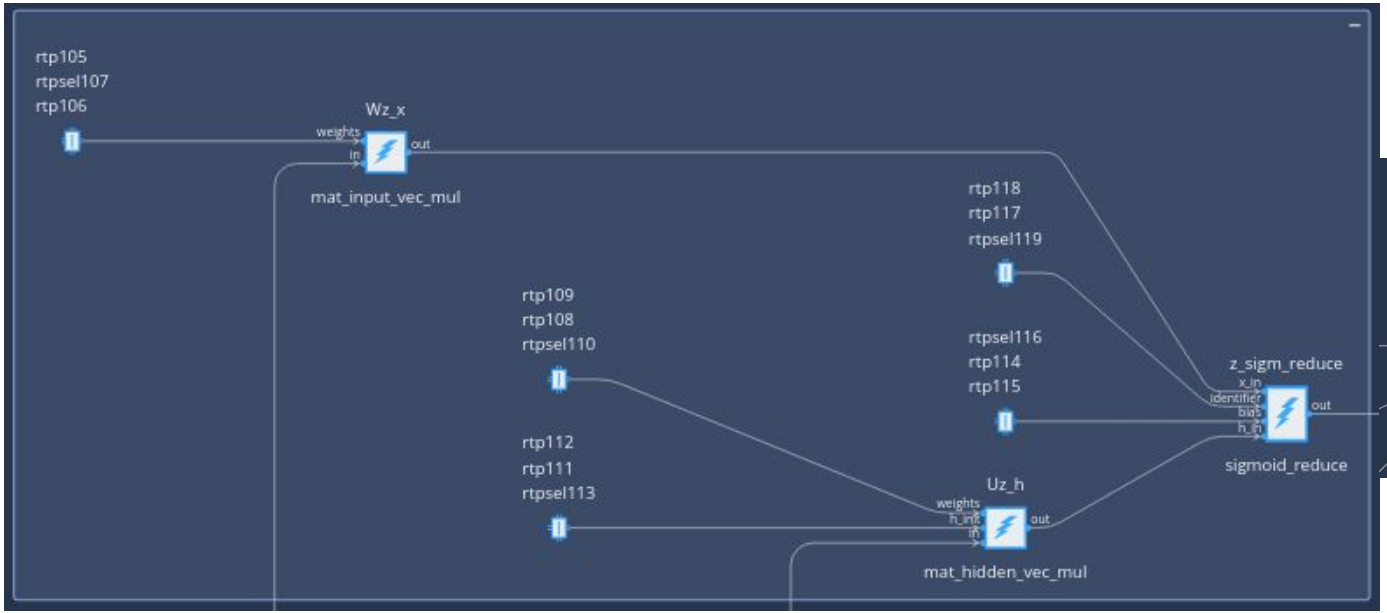
Aggregate partial results w Packet Stream: Merge

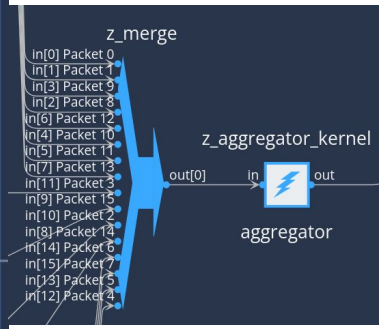
- Up to 32



Rows (First Way)

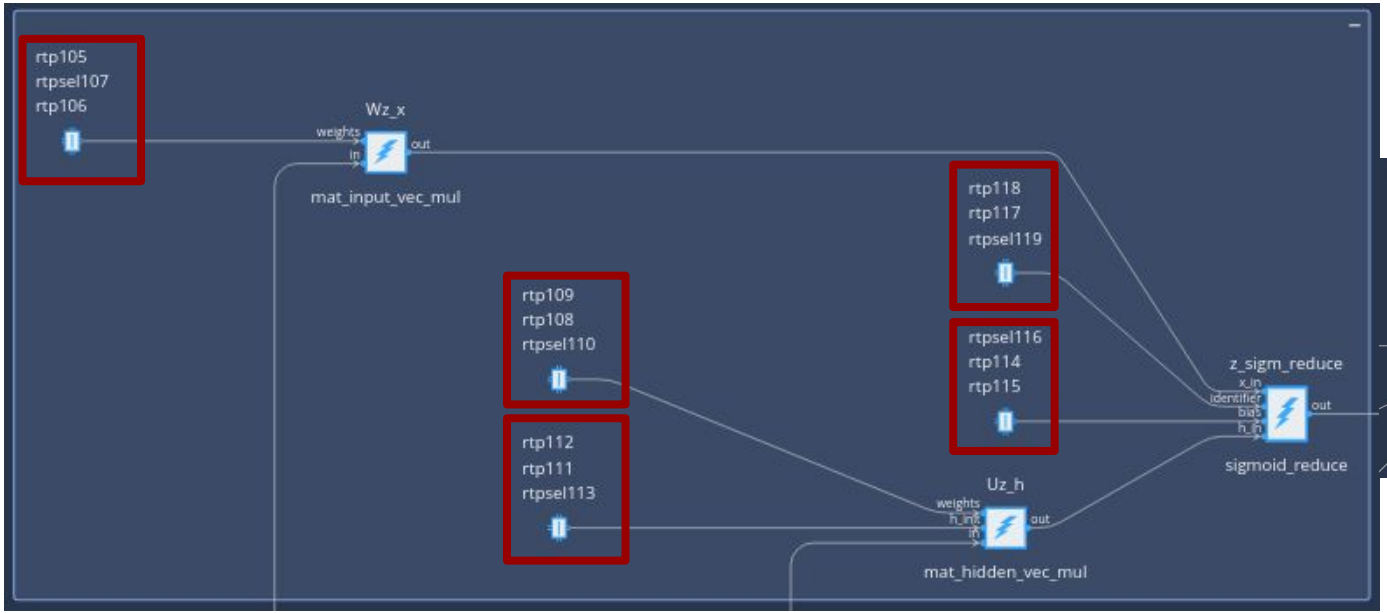


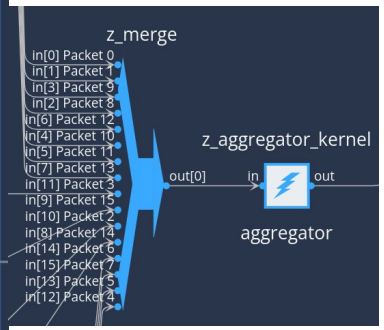




$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z)$$

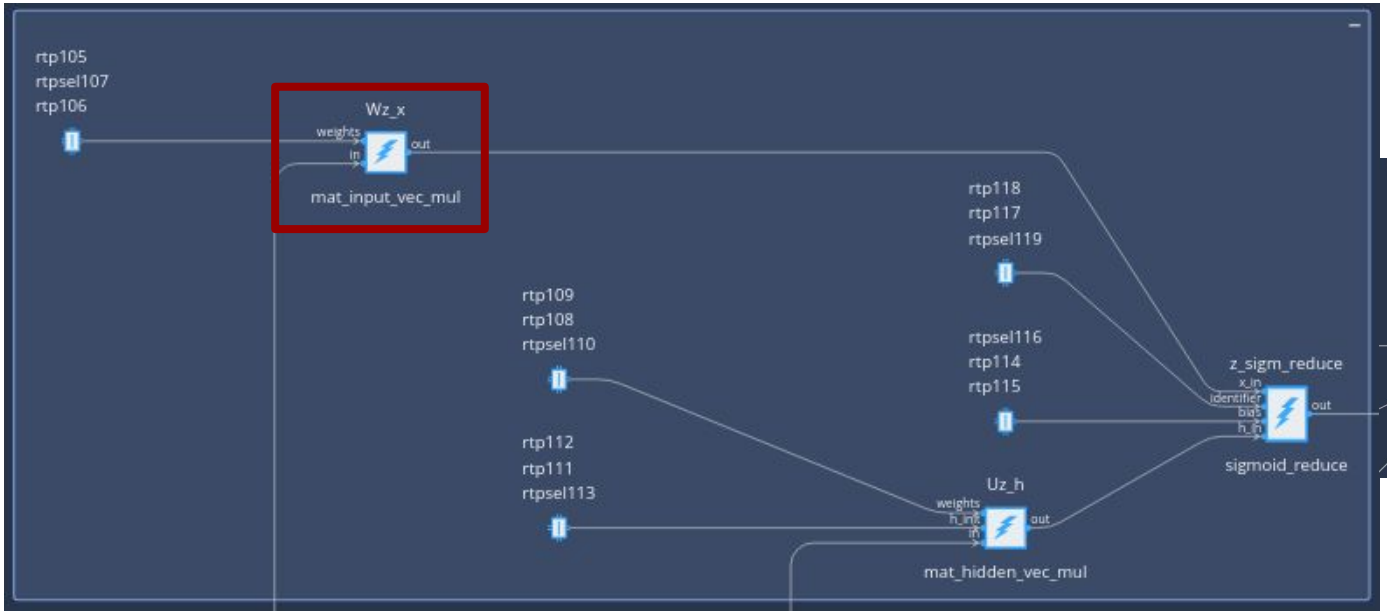
Rows (First Way)

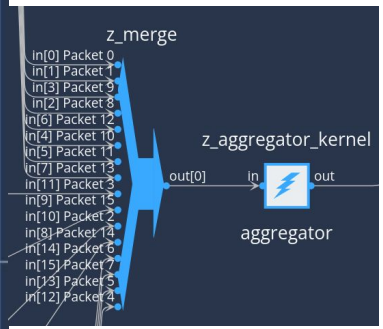




$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z)$$

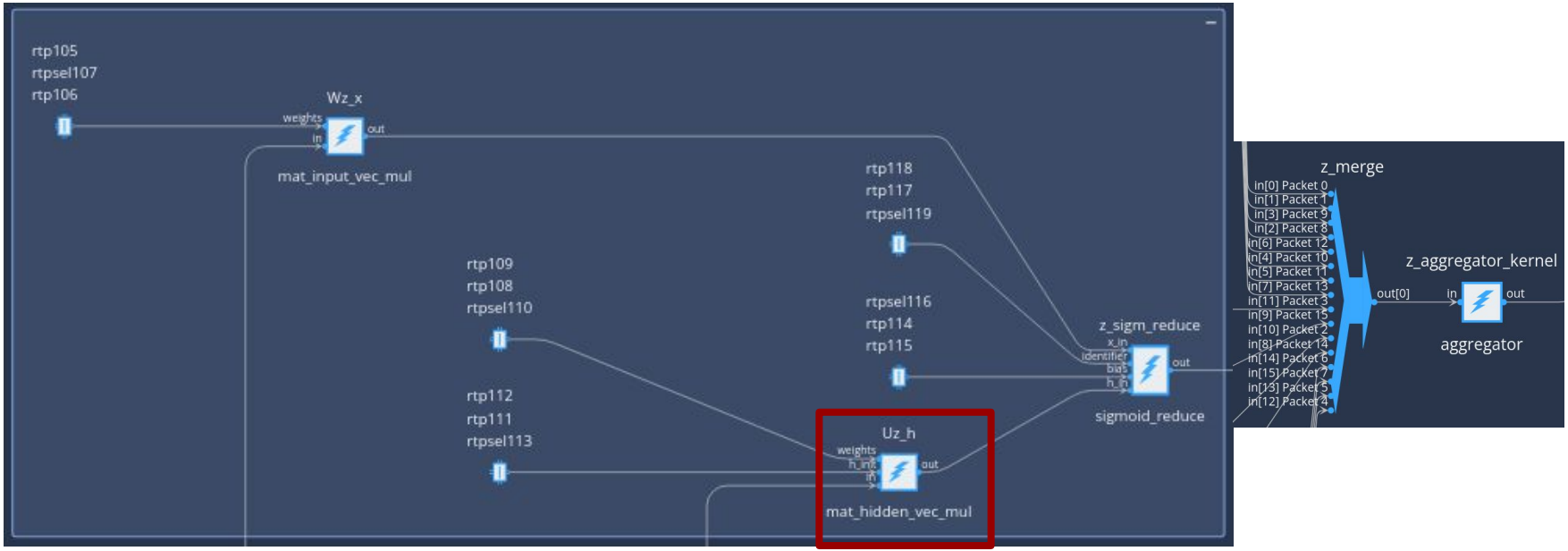
Rows (First Way)





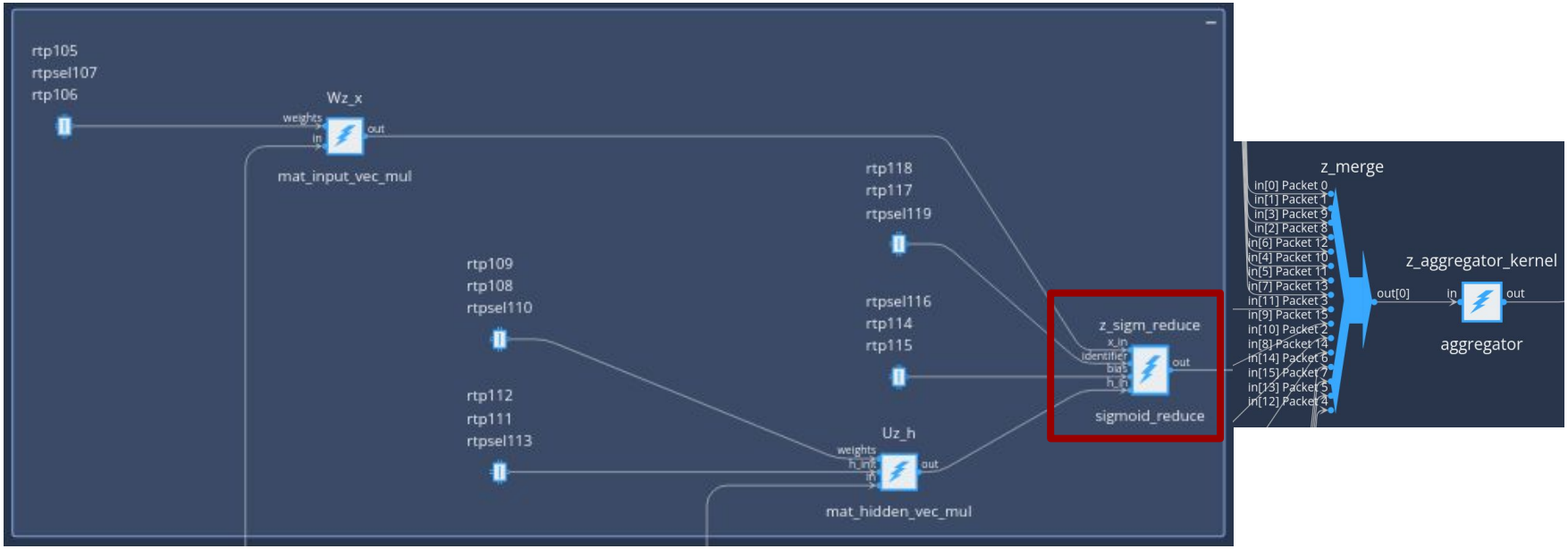
$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z)$$

Rows (First Way)



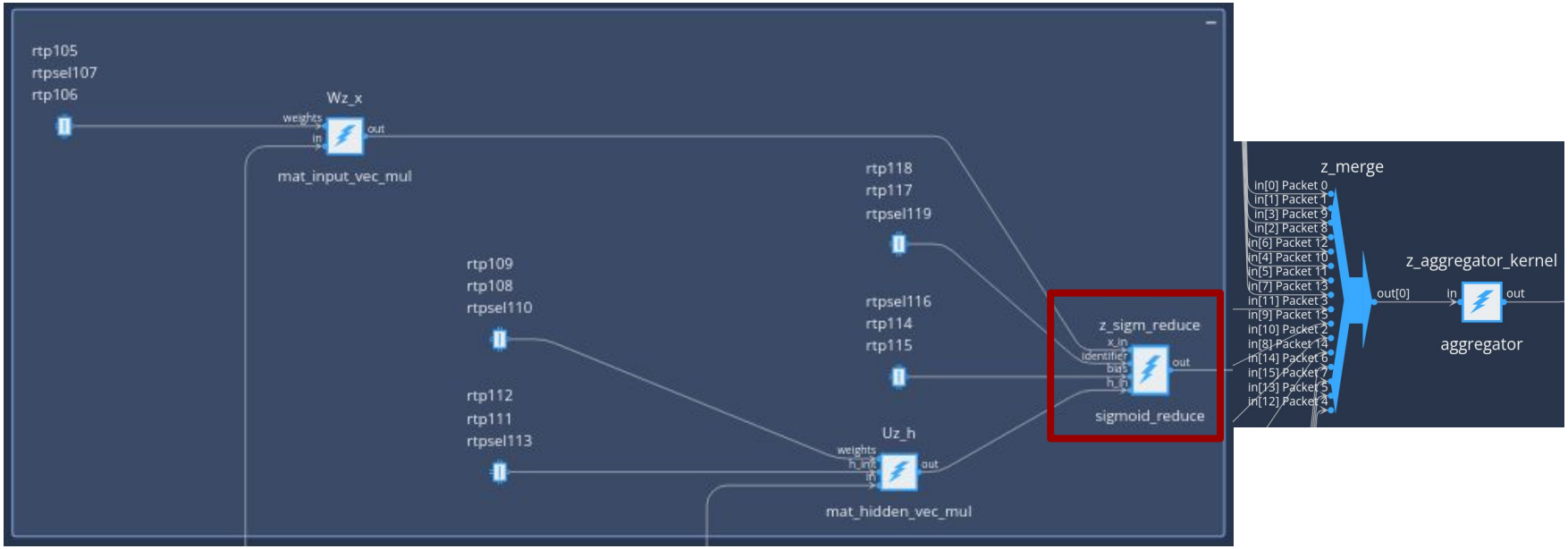
$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z)$$

Rows (First Way)



$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z)$$

Rows (First Way)



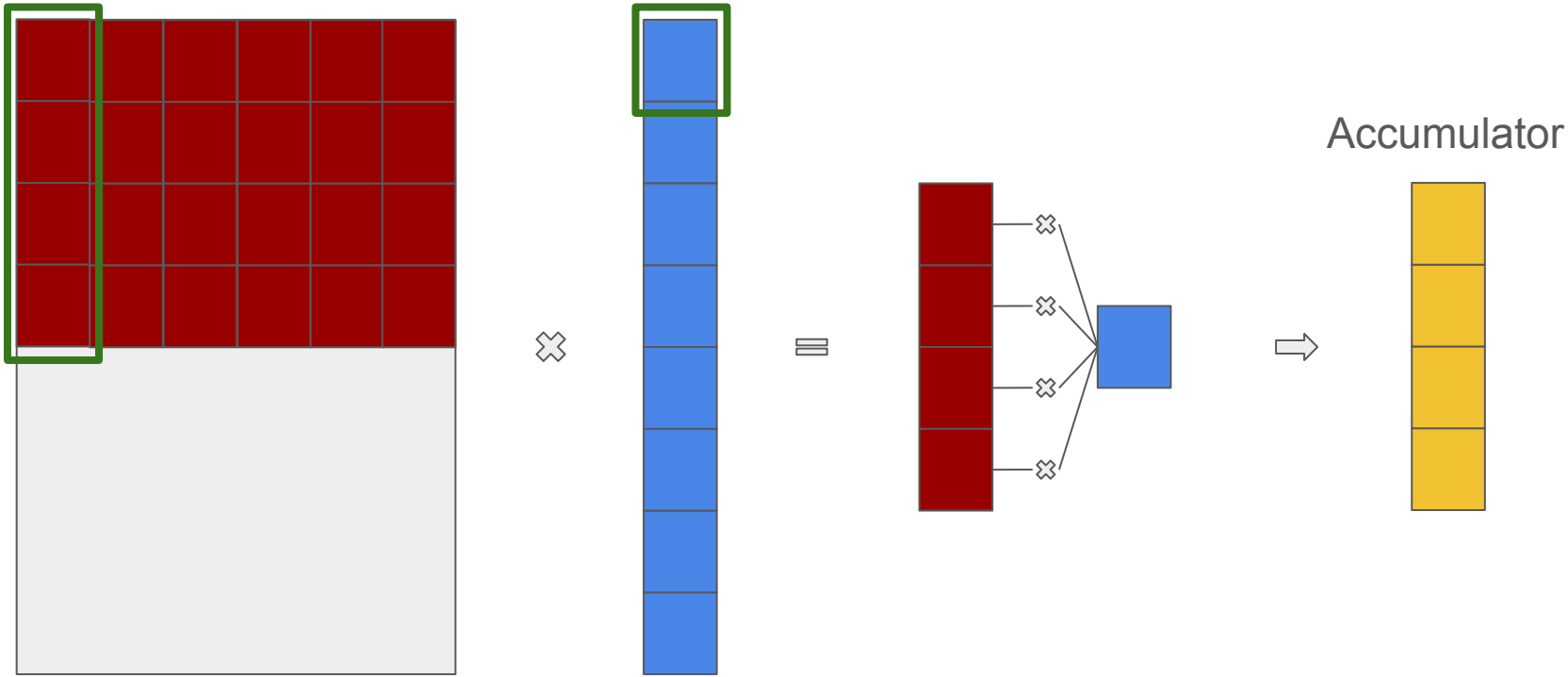
$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z)$$

Same for all 3 gates

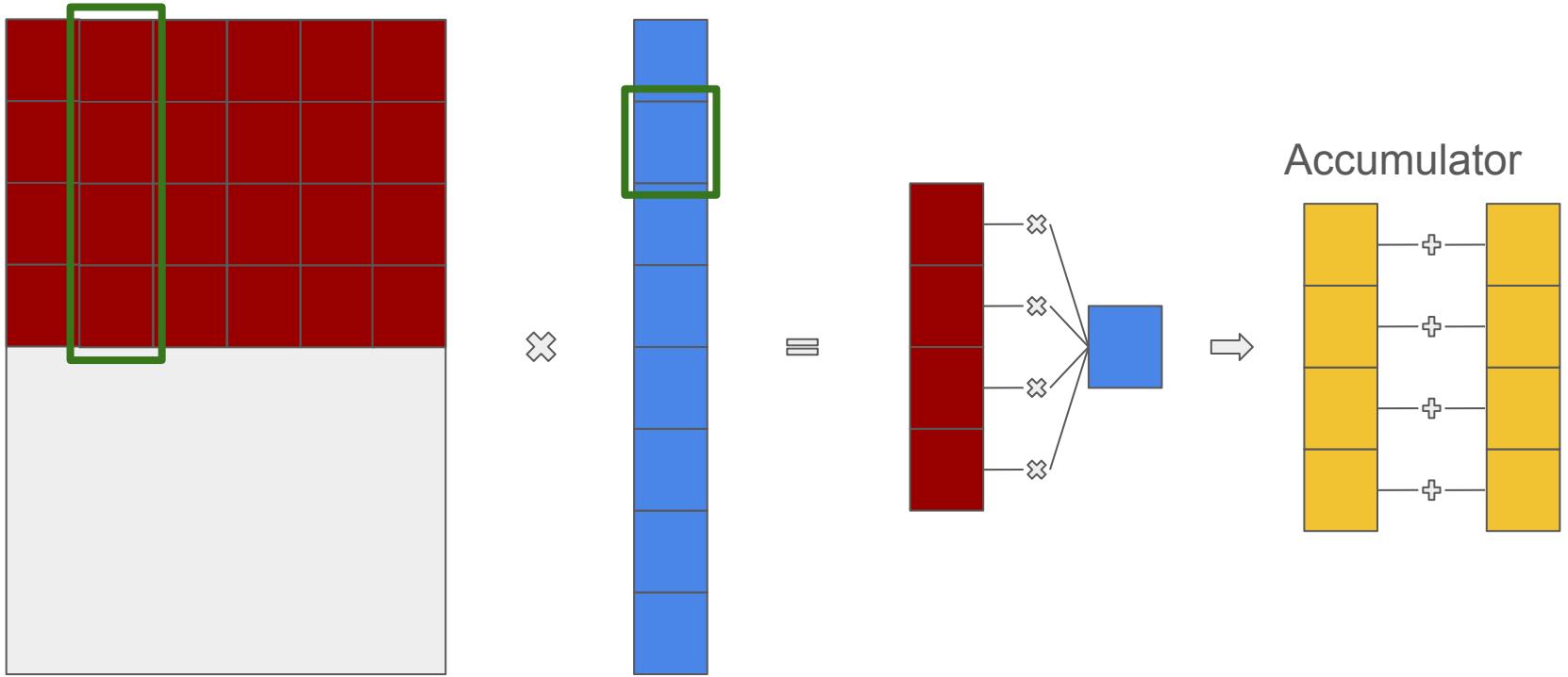
$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z)$$
$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r)$$
$$\hat{h}_t = \phi(W_h x_t + U_h (r_t \odot h_{t-1}) + b_h)$$

Trivial $\longrightarrow$ $h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \hat{h}_t$

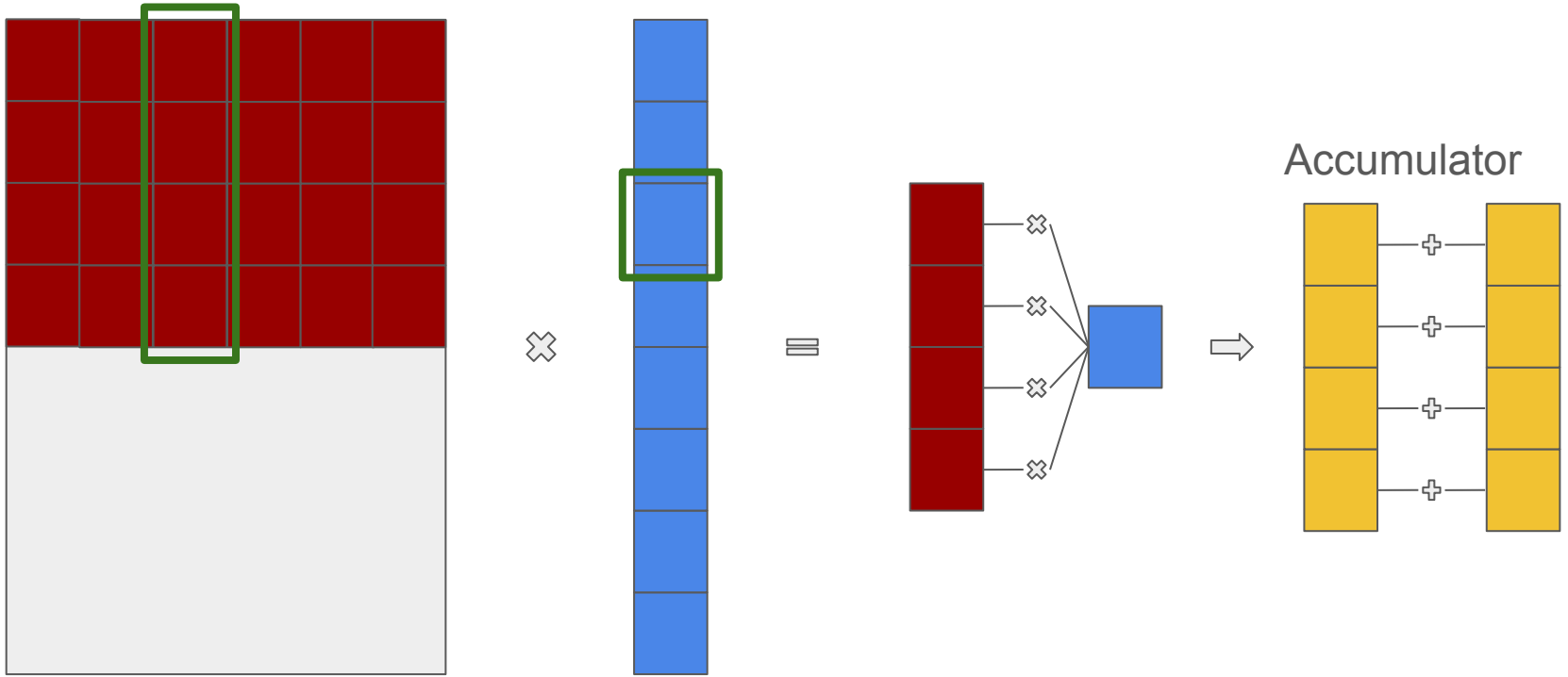
Multiply Accumulate: Columns



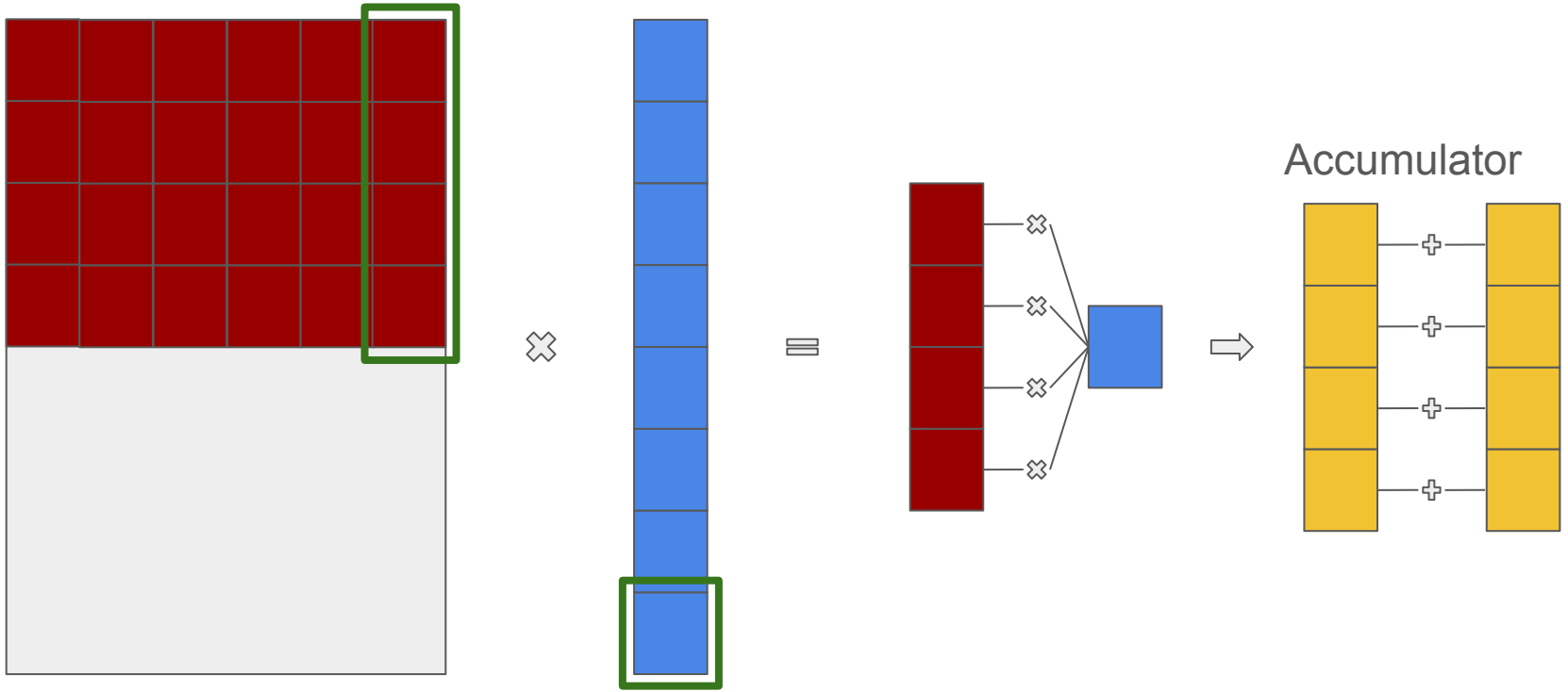
Multiply Accumulate: Columns



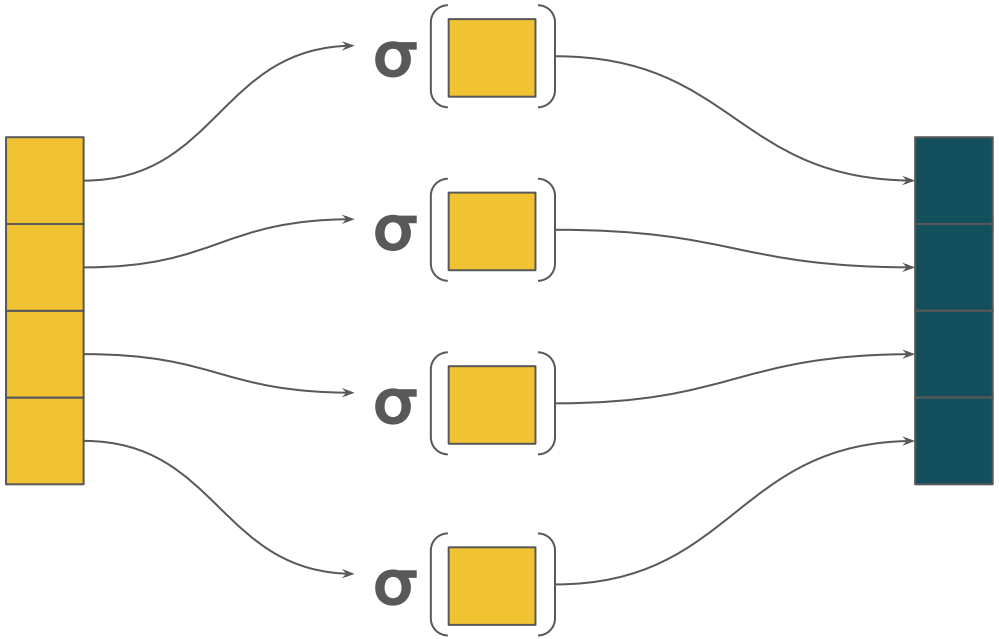
Multiply Accumulate: Columns



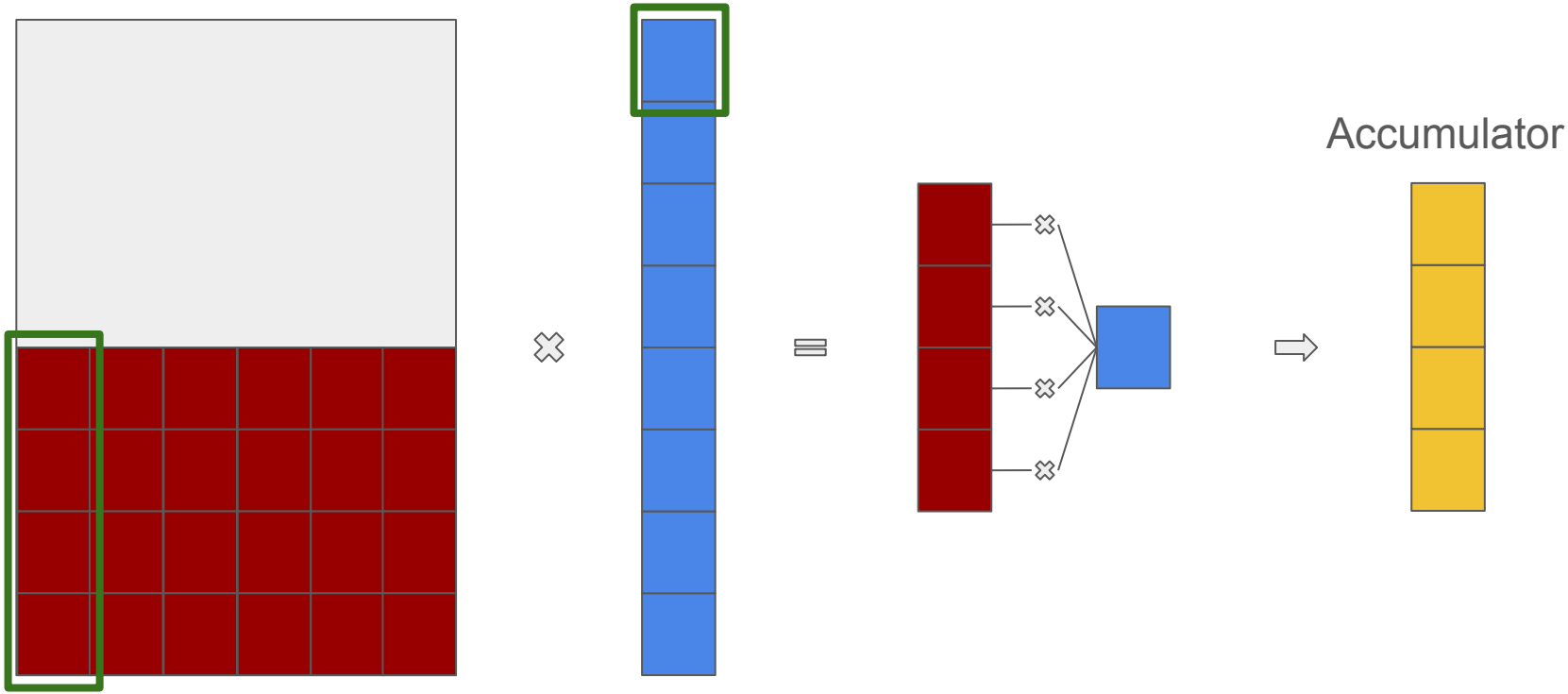
Multiply Accumulate: Columns



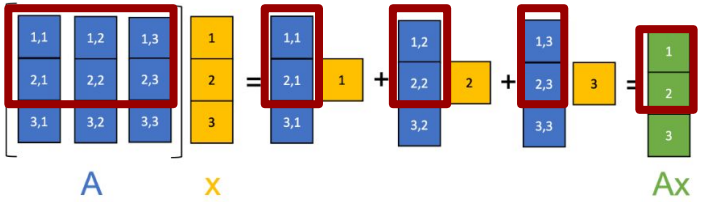
Multiply Accumulate: Columns

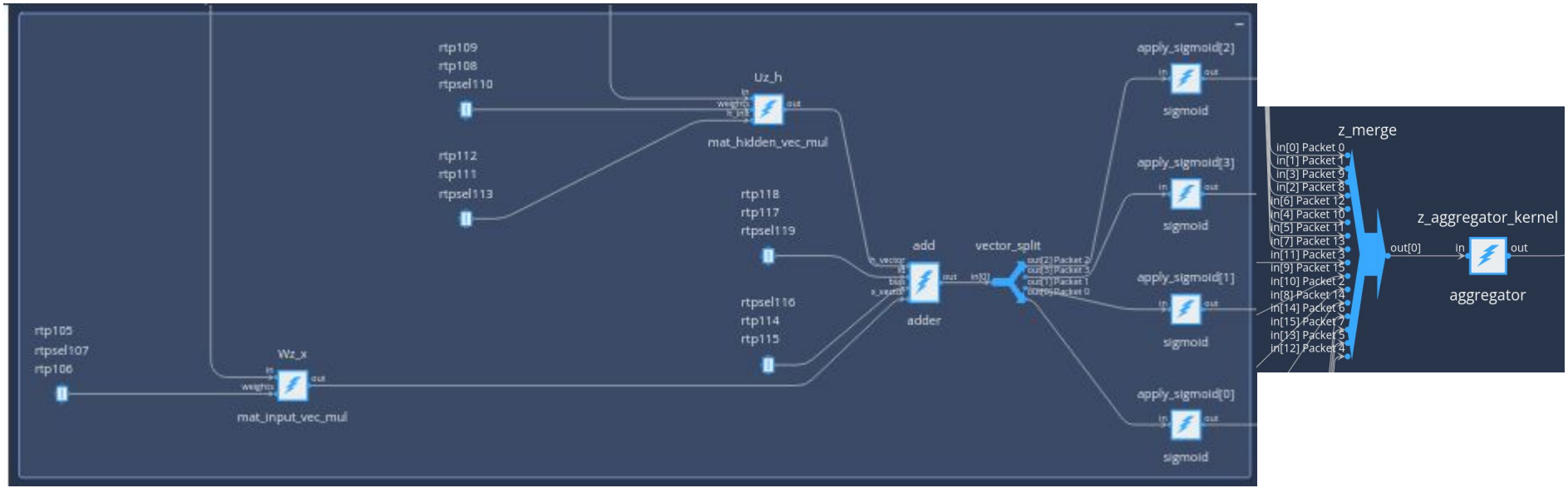


Multiply Accumulate: Columns



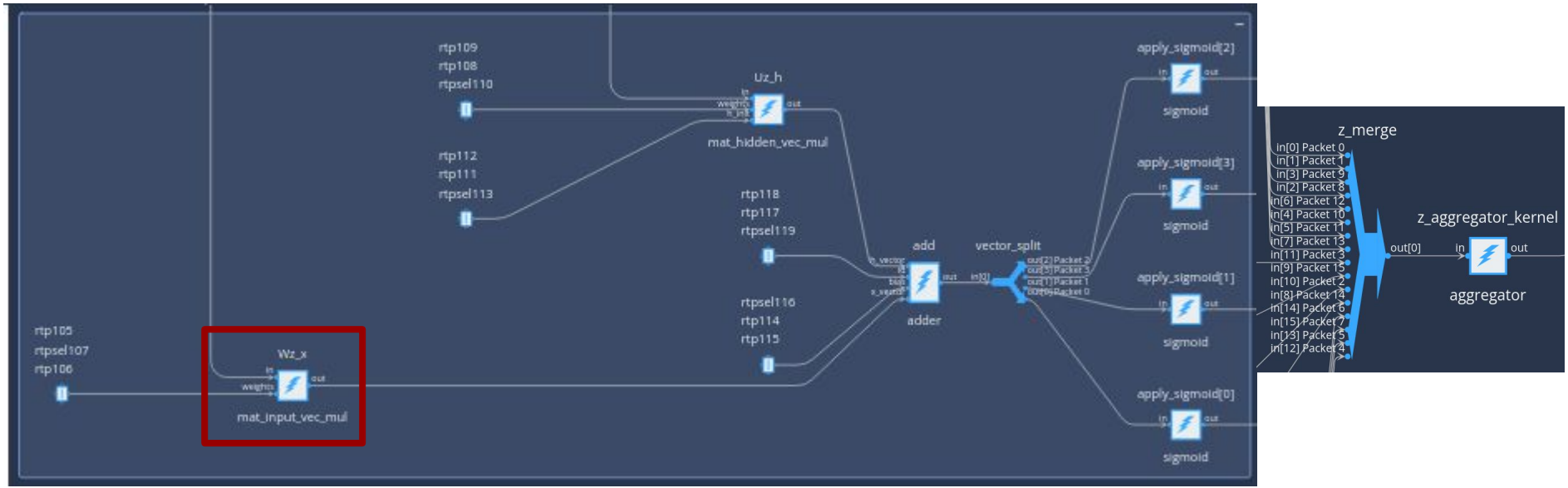
Columns (Second Way)





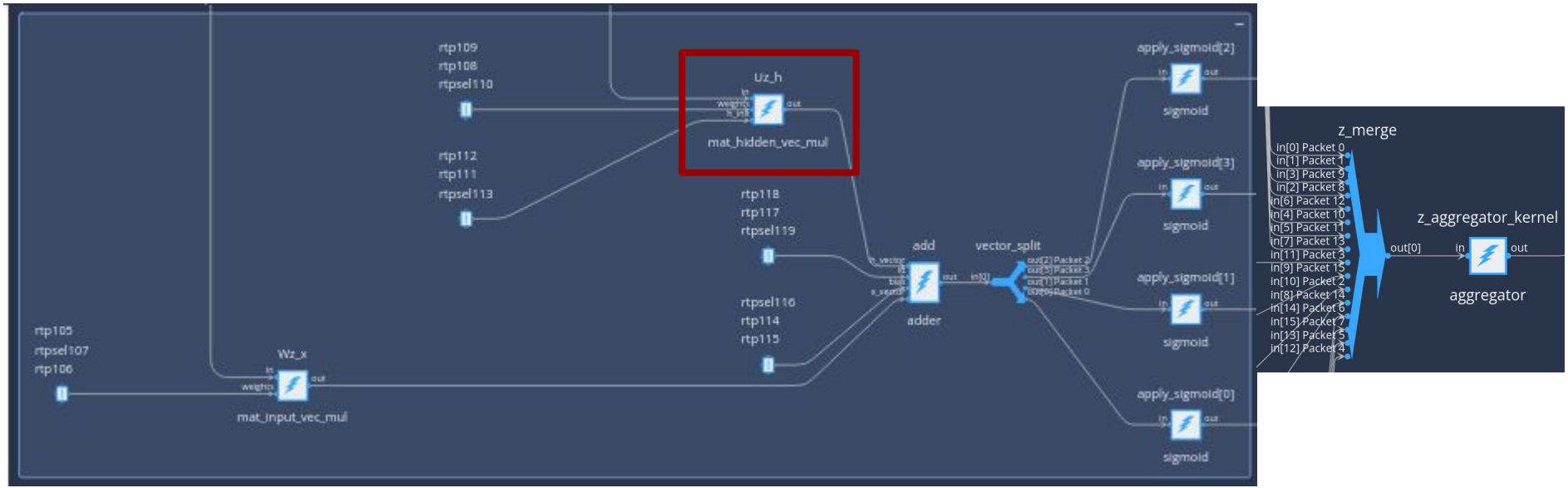
$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z)$$

Columns (Second Way)



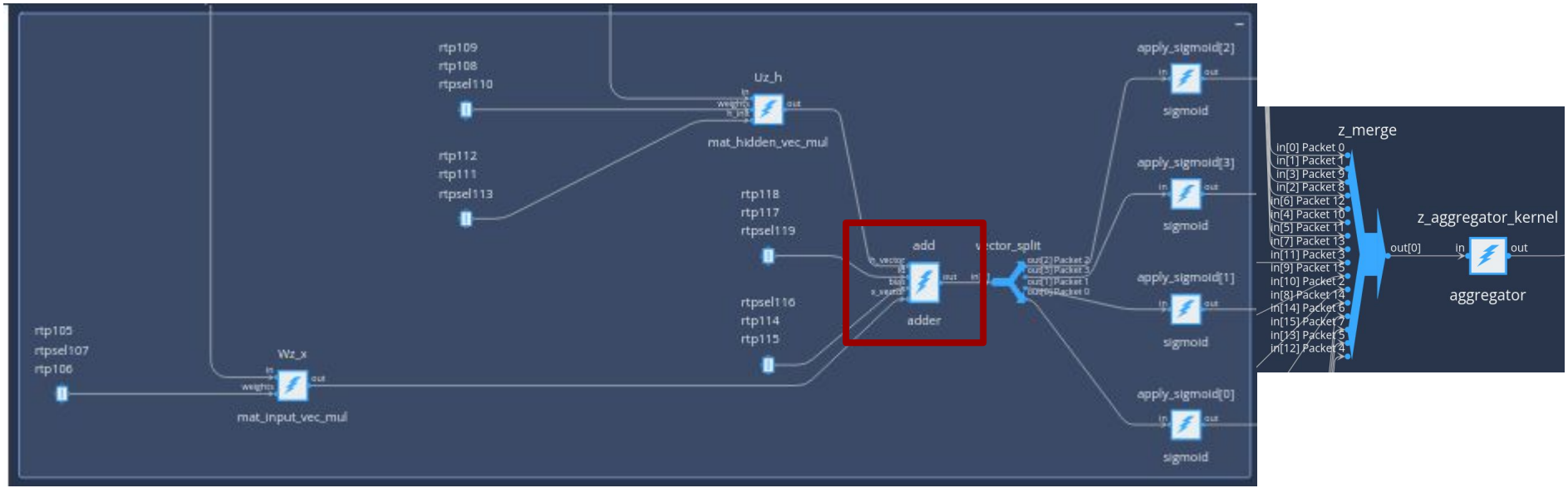
$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z)$$

Columns (Second Way)



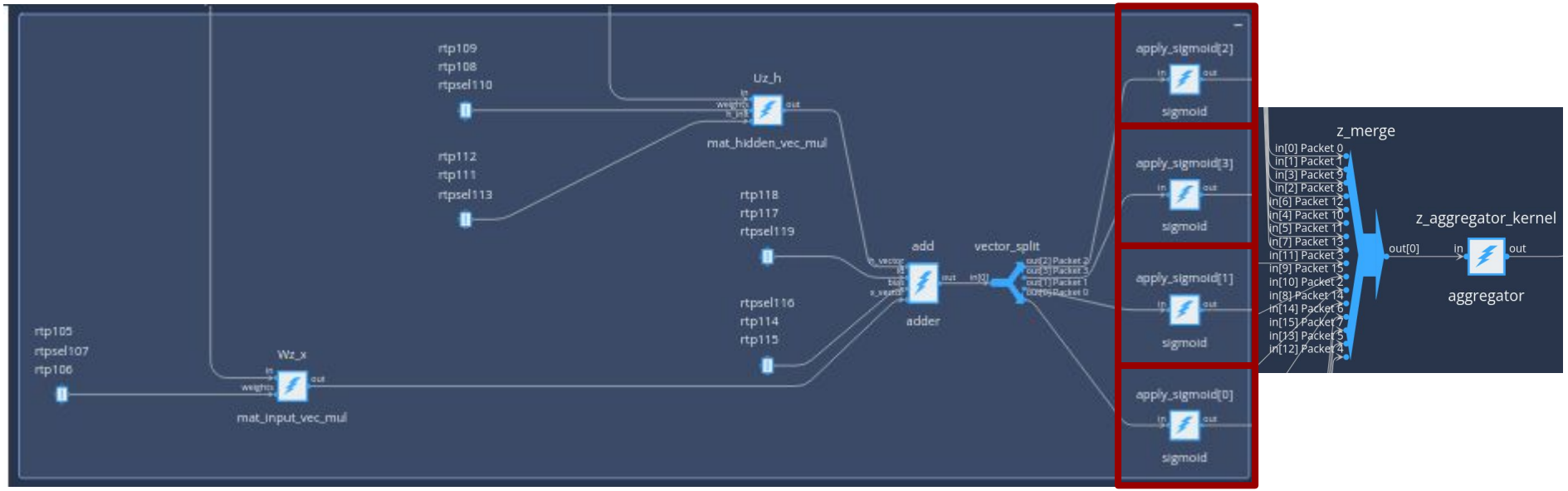
$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z)$$

Columns (Second Way)



$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z)$$

Columns (Second Way)



$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z)$$

Regarding the activation functions

Base Solution - Look Up Tables

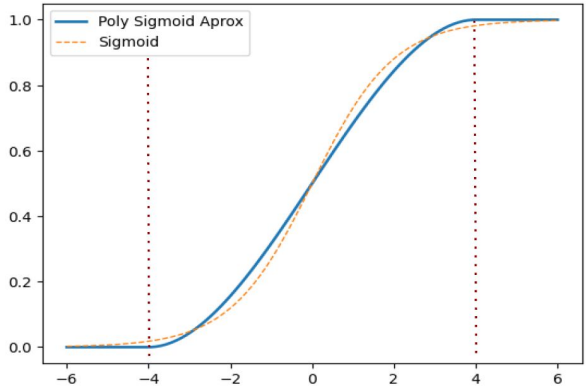
- This is the only solution in FPGA fabric and has the potential to be very fast
- Lose in precision

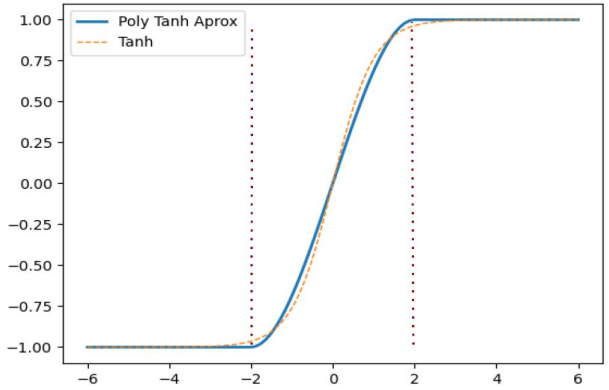
- In the AI Engines we have less than 32KB of available memory for LUTs
- ~ 8000 indices (- space we need for stack)
- **Need to scale a random float to the index range and cast to int**
- Random accessing memory is expensive, around ~ 70ns
- What happens to floating point computations with quantized activation functions?

Regarding the activation functions

Cool Solution - Compute a spline

- Implement in the AI Engine
- Change True Sigm and True Tanh with splines (here is 3rd order)





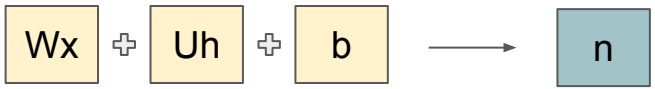
$$\sigma(x) = \frac{1}{2} + \frac{3}{16}x - \frac{1}{256}x^3$$

$$\tanh(x) = \frac{12}{16}x - \frac{1}{16}x^3$$

Regarding the activation functions

Cool Solution - Compute a spline

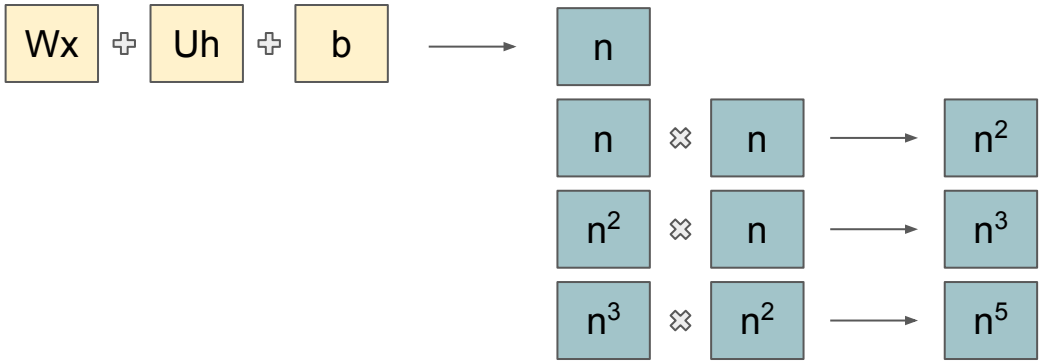
- Train the model with the modified functions
- Here is how to implement a kernel that computes a single spline:



Regarding the activation functions

Cool Solution - Compute a spline

- Train the model with the modified functions
- Here is how to implement a kernel that computes a single spline:

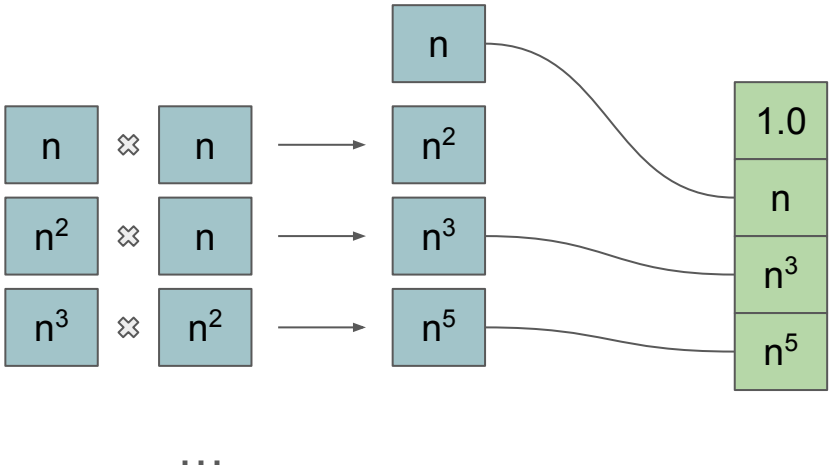


$$\sigma(x) = \frac{1}{2} + \frac{3}{16}x - \frac{1}{256}x^3$$

Regarding the activation functions

Cool Solution - Compute a spline

- Train the model with the modified functions
- Here is how to implement a kernel that computes a single spline:

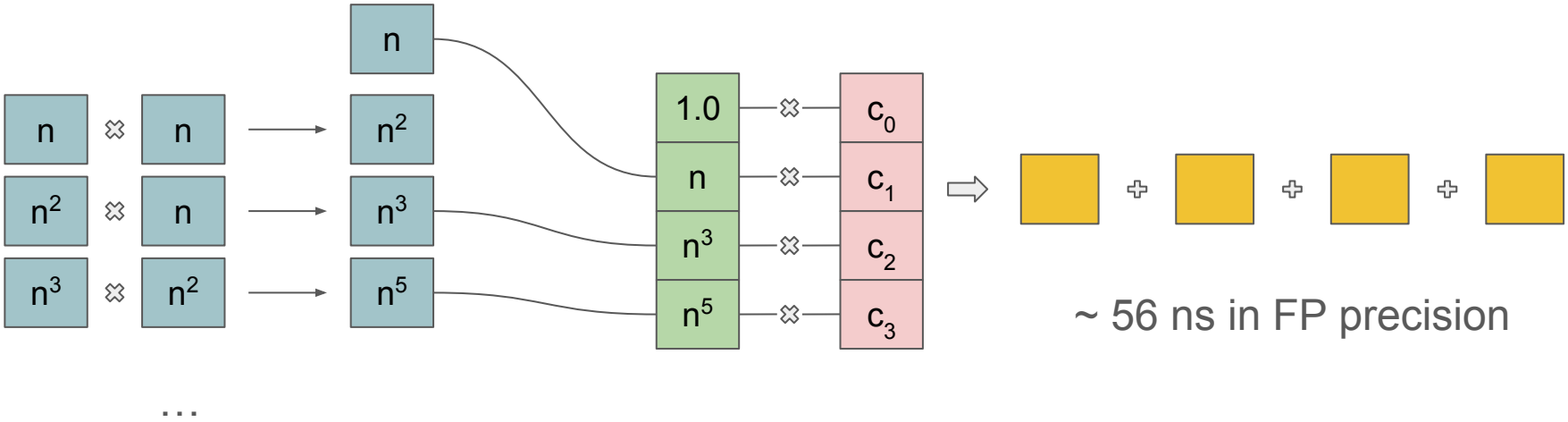


$$\sigma(x) = \frac{1}{2} + \frac{3}{16}x - \frac{1}{256}x^3$$

Regarding the activation functions

Cool Solution - Compute a spline

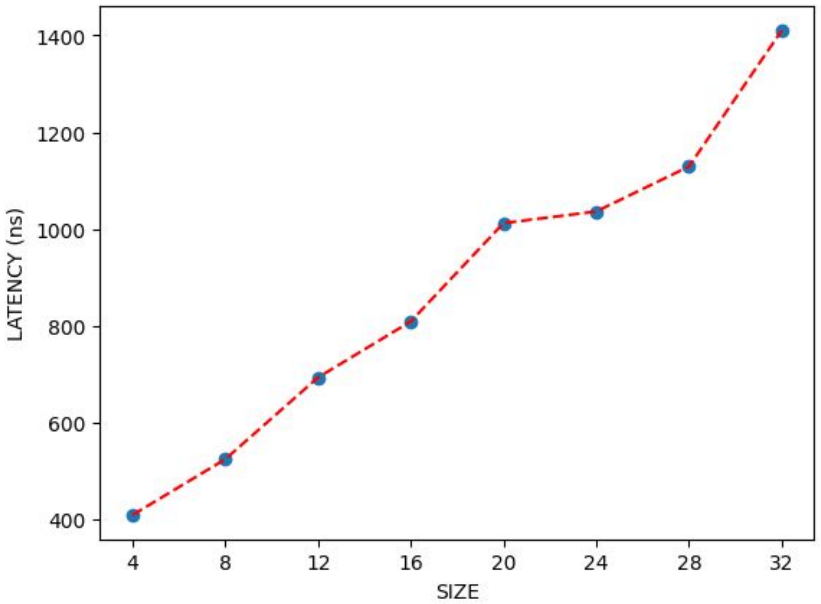
- Train the model with the modified functions
- Here is how to implement a kernel that computes a single spline:



Latencies (AIE hw emulation) for Row implementation

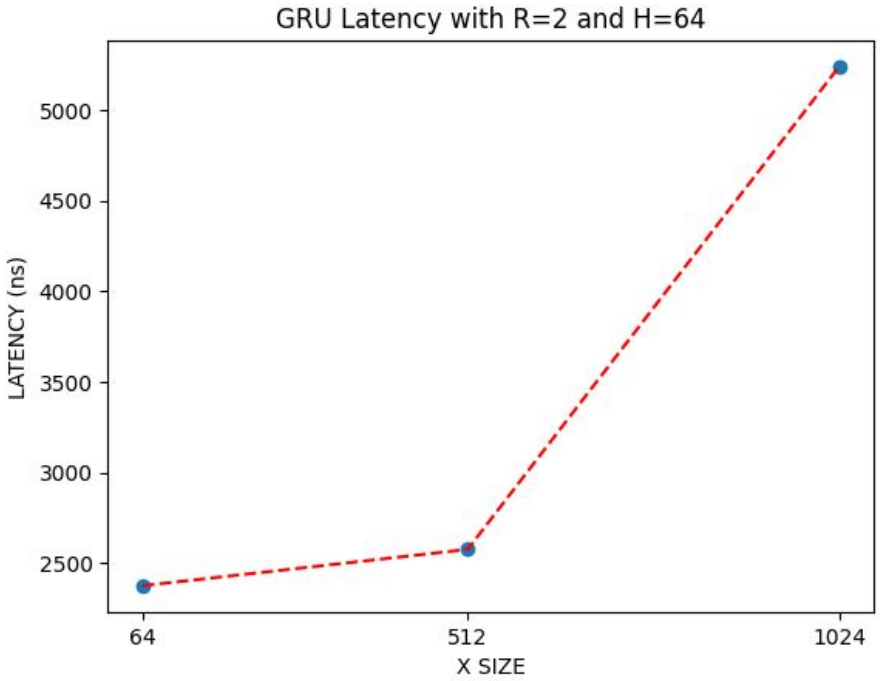
X_size = H_size	Latency (ns)	Resources
4	408.8	40
8	524	76
12	693	112
16	808.8	148
20	1012	184
24	1036	220
28	1129	256
32	1410.4	292

- In FP32 precision and LUTs



Latencies (AIE hw emulation) for Row implementation

X_size	Latency (ns)	Resources
64	2376	292
512	2576	292
1024	5238	292



Matrix Dimensions Problem

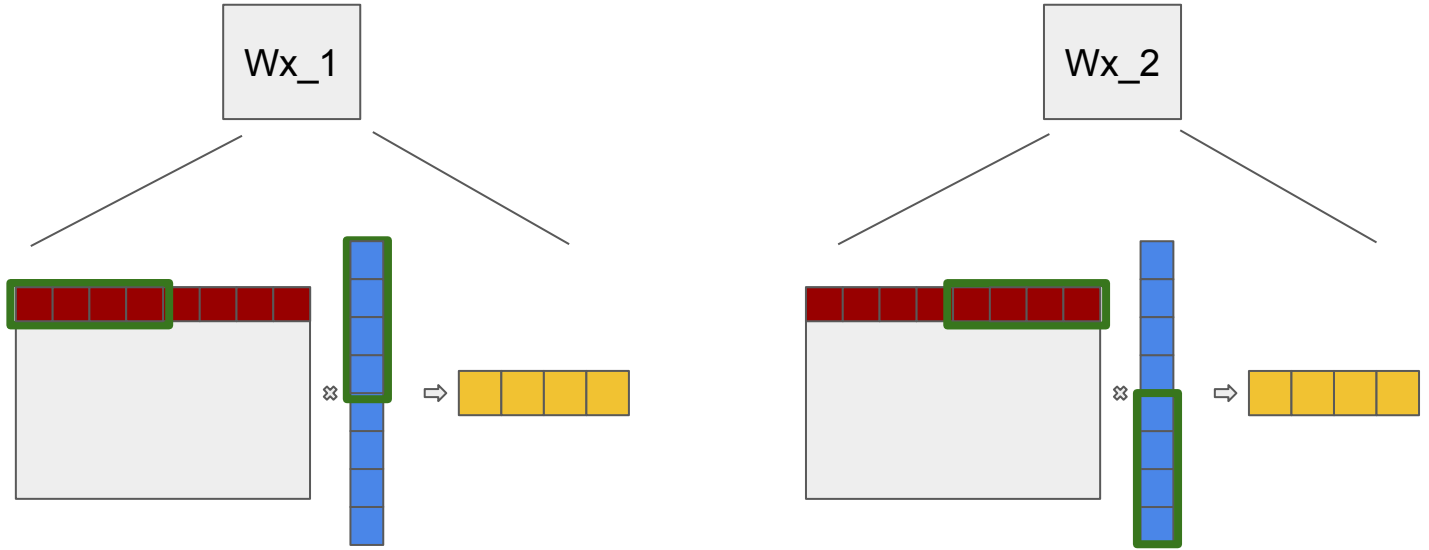
W: H x X

U: H x H

- Matrices don’t have the same size
- Means that we need to further distribute the computations so that all Matrix-Vector kernels are doing the same amount of MAC operations

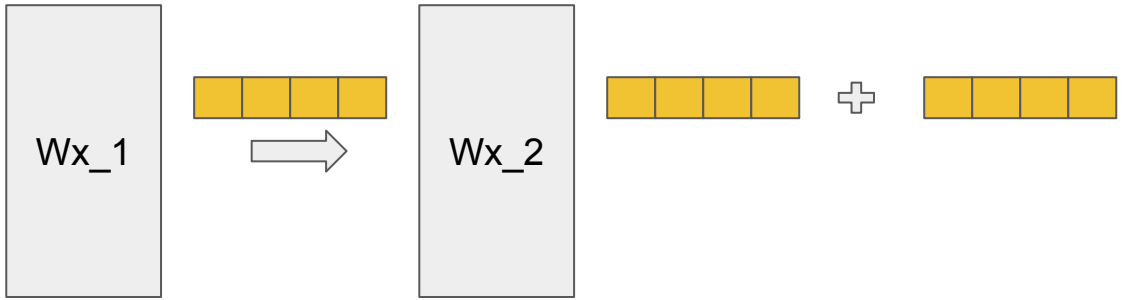
- We are working on this

Solution: Further Partially Compute (need more tiles)



- **We are working on this

And use Cascade



Free Running Kernels

- All kernels are “free running”, meaning that they never stop and stall when they are waiting to accept inputs or pass outputs
- The compiler cannot clearly understand this behavior and schedules reads and writes on the same cycle, stalling the whole pipeline

Solution:

- Use scheduling pragma to direct the compiler

Memory allocation

- LUTs get allocated in stack memory and the tool thinks that every kernel will be using the same stack

Possible Solution:

- Manually allocate stack for each kernel
- We are working on this

(or just use the spline thing)

Data Congestion

- Correct Tile placement can optimize further data aggregation

- It is arguable if the Mapper / Placer provided by the tool is doing the best optimization

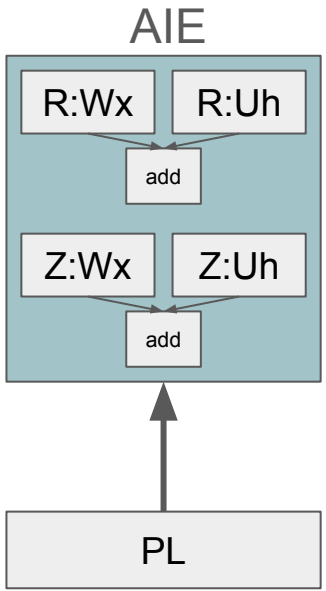
Outside the Hardware

- You need to reshape the trained parameters correctly (sometimes it's not that trivial and it depends on how you designed the MAC operations)
- Then pass them as RTPs in the Hardware

- In case you want to replace LUTs with computing splines for Activations, you need to design your own pytorch activation function classes

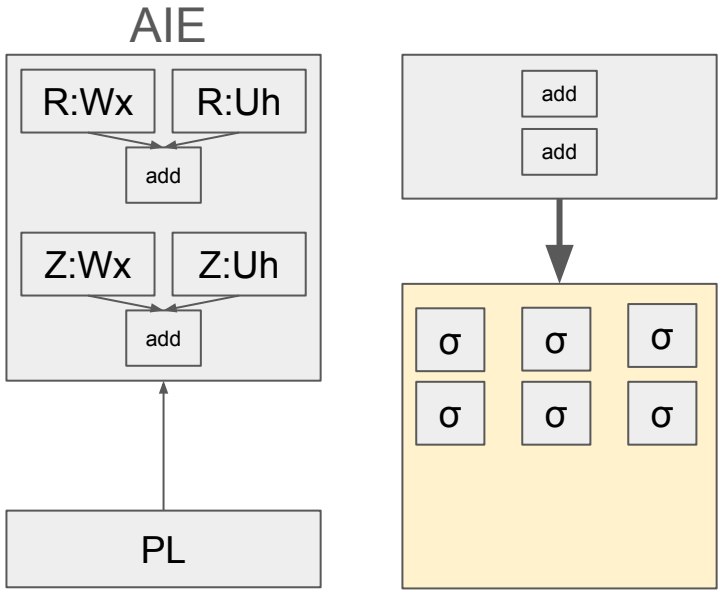
Going forward: Hybrid Solution

- Use the AI Engine only to distribute Matrix - Vector calculations



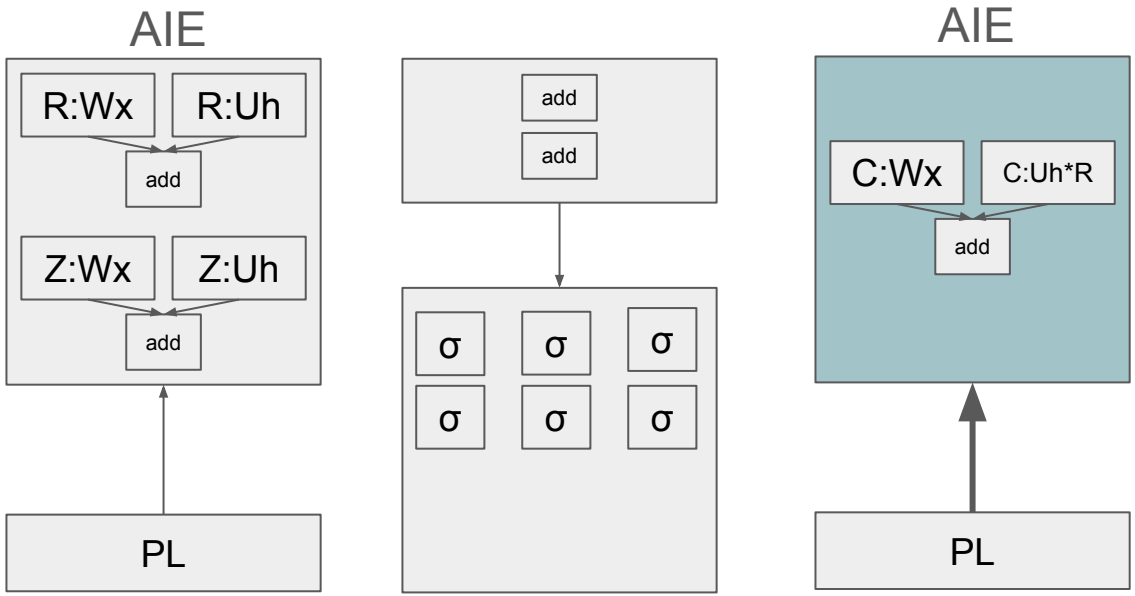
Going forward: Hybrid Solution

- Use the AI Engine only to distribute Matrix - Vector calculations



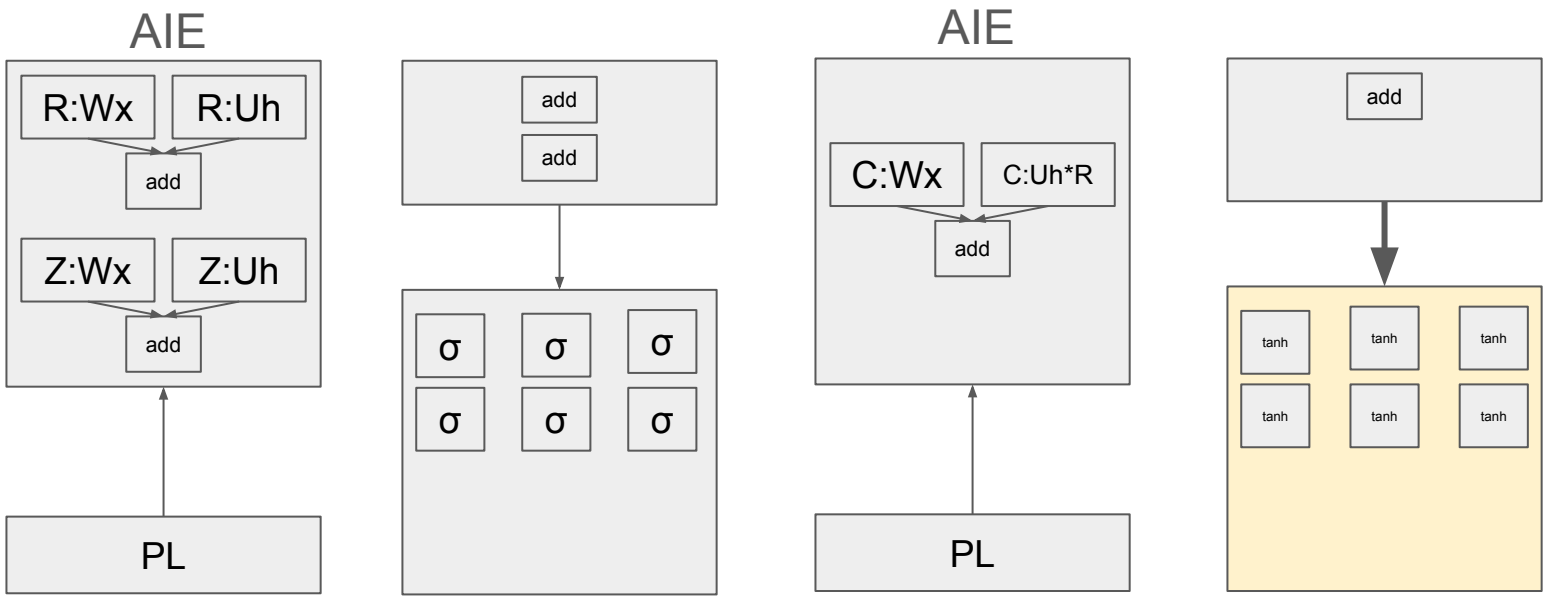
Going forward: Hybrid Solution

- Use the AI Engine only to distribute Matrix - Vector calculations



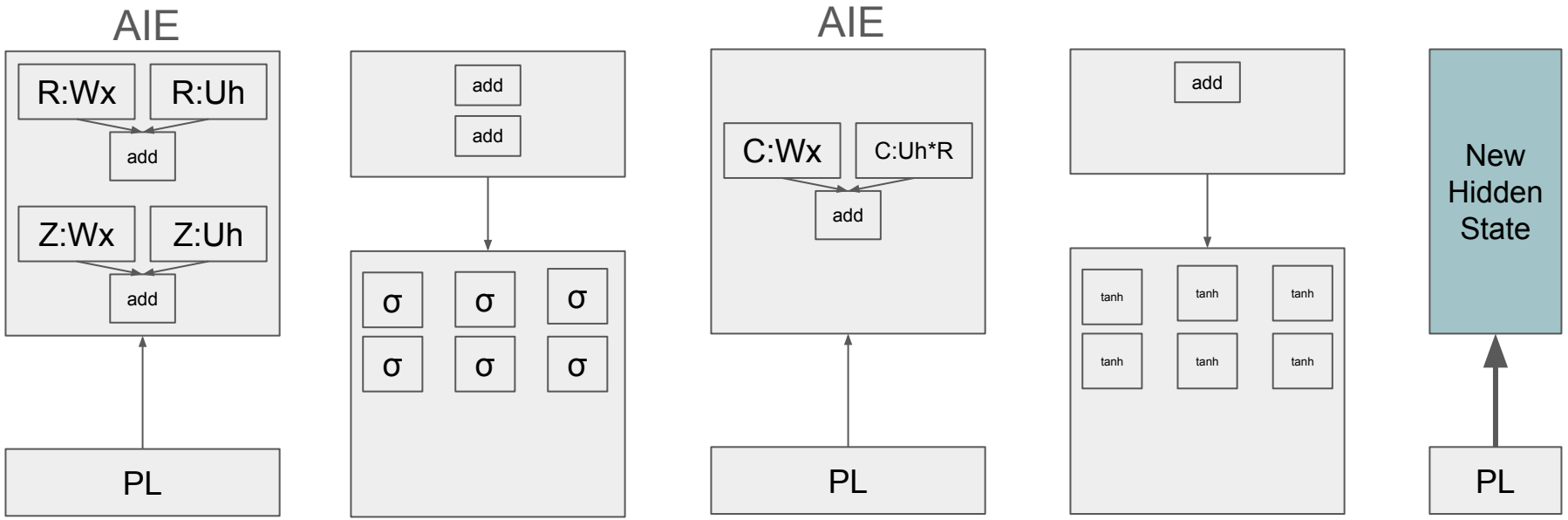
Going forward: Hybrid Solution

- Use the AI Engine only to distribute Matrix - Vector calculations



Going forward: Hybrid Solution

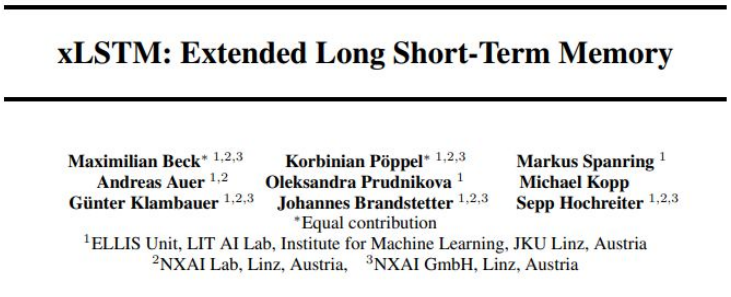
- Use the AI Engine only to distribute Matrix - Vector calculations



Going forward: Implementing other models

- Once we unlock the GRU we could easily design an LSTM on the same principles

- Or even more cutting edge models, such as xLSTM and RWKV



xLSTM: Extended Long Short-Term Memory

Maximilian Beck* [1,2,3] Korbinian Pöppel* [1,2,3] Markus Spanring [1]
Andreas Auer [1,2] Oleksandra Prudnikova [1] Michael Kopp
Günter Klambauer [1,2,3] Johannes Brandstetter [1,2,3] Sepp Hochreiter [1,2,3]
*Equal contribution
[1]ELLIS Unit, LIT AI Lab, Institute for Machine Learning, JKU Linz, Austria
[2]NXAI Lab, Linz, Austria, [3]NXAI GmbH, Linz, Austria

https://arxiv.org/pdf/2405.04517

RWKV: Reinventing RNNs for the Transformer Era

Bo Peng[1,2*] Eric Alcaide[2,3,4*] Quentin Anthony[2,5*]
Alon Albalak[2,6] Samuel Arcadinho[2,7] Stella Biderman[2,8] Huanqi Cao[9] Xin Cheng[10]
Michael Chung[11] Xingjian Du[1] Matteo Grella[12] Kranthi Kiran GV[2,13] Xuzheng He[2]
Haowen Hou[14] Jiaju Lin[1] Przemysław Kazienko[15] Jan Kocoń[15] Jiaming Kong[16]
Bartłomiej Koptyra[15] Hayden Lau[2] Krishna Sri Ipsit Mantri[17] Ferdinand Mom[18,19]
Atsushi Saito[2,20] Guangyu Song[21] Xiangru Tang[22] Bolun Wang[23] Johan S. Wind[24]
Stanisław Woźniak[15] Ruichong Zhang[9] Zhenyuan Zhang[2] Qihang Zhao[25,26]
Peng Zhou[23] Qinghua Zhou[5] Jian Zhu[27] Rui-Jie Zhu[28,29]
[1]Generative AI Commons [2]EleutherAI [3]U. of Barcelona [4]Charm Therapeutics [5]Ohio State U. [6]U. of C., Santa Barbara
[7]Zendesk [8]Booz Allen Hamilton [9]Tsinghua University [10]Peking University [11]Storyteller.io [12]Crisis24 [13]New York U.
[14]National U. of Singapore [15]Wroclaw U. of Science and Technology [16]Databaker Technology [17]Purdue U. [18]Criteo AI Lab
[19]Epita [20]Nextremer [21]Moves [22]Yale U. [23]RuoxinTech [24]U. of Oslo [25]U. of Science and Technology of China
[26]Kuaishou Technology [27]U. of British Columbia [28]U. of C., Santa Cruz [29]U. of Electronic Science and Technology of China

https://arxiv.org/pdf/2305.13048

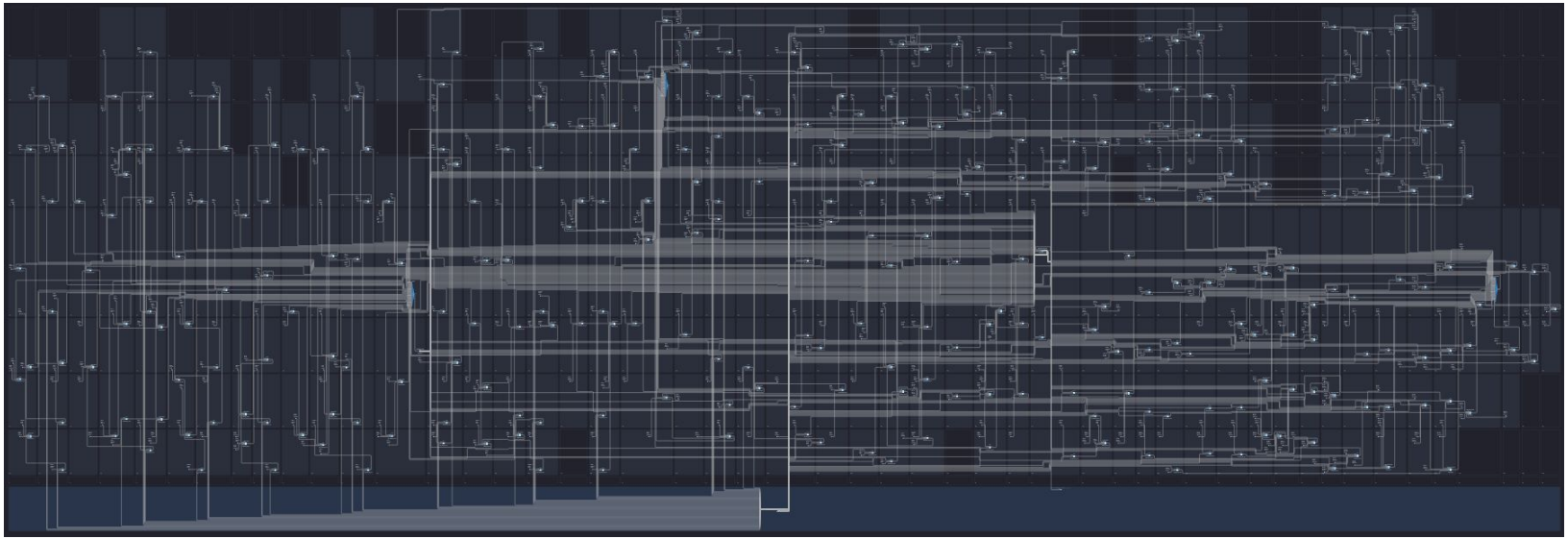
A final remark

- In an reinforcement learning framework these powerful models are “machine agnostic”, they just need time to explore the action space (and hp tuning) and need one forward pass

- They just work towards their single goal which is represented by a Loss Function:
  - Minimize standard deviation of some signal (control instabilities)
  - Maximize gate fidelity (qubits)
  - Weighted combinations of metrics / loss functions appropriate for the problem

- The faster you act, the better you control phase space trajectories in dynamical systems

Thank you for your attention



https://github.com/Msapkas/gru_aie