Open-Source Heterogeneous Computing with Cluster-Coupled Accelerators: A Neural Engine Case Study

Arpan Prasad[*], Gianna Paulin[*], **Yvan Tortorella[+]**, Luca Bertaccini[*], Luca Benini[*+], Francesco Conti[+]
[*]Integrated Systems Laboratory, ETH Zürich
[+]Energy-Efficient Embedded Systems Laboratory, University of Bologna

{prasadar, pauling, lbertaccini, lbenini}@iis.ee.ethz.ch
{yvan.tortorella,f.conti}@unibo.it

PULP includes Cores+Interco+IO+HWPE → Open Platform

RISC-V Cores

- RI5CY 32b
- Ibex 32b
- Snitch 32b
- Ariane + Ara 64b

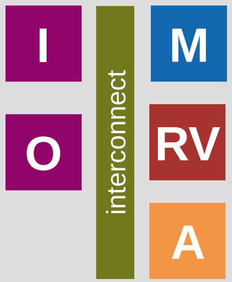
Peripherals

- JTAG
- SPI
- UART
- I2S
- DMA
- GPIO

Interconnect

- Logarithmic interconnect
- APB – Peripheral Bus
- AXI4 – Interconnect

Platforms



Single Core

- PULPissimo

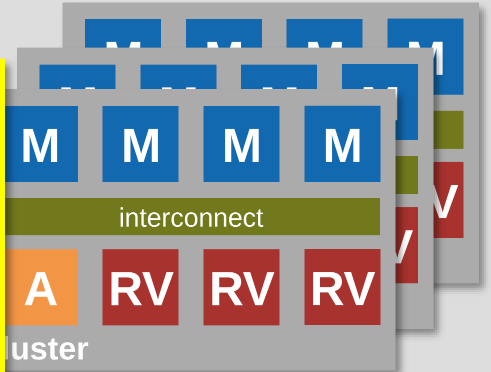
Open-Source Hardware > Open-Source ISA:

- core implementations
- system-level integration
- peripherals, <u>**accelerators**</u>

<u>**Everything Designed in System Verilog**</u>

Multi-core

- Mr. Wolf
- Vega



Multi-cluster

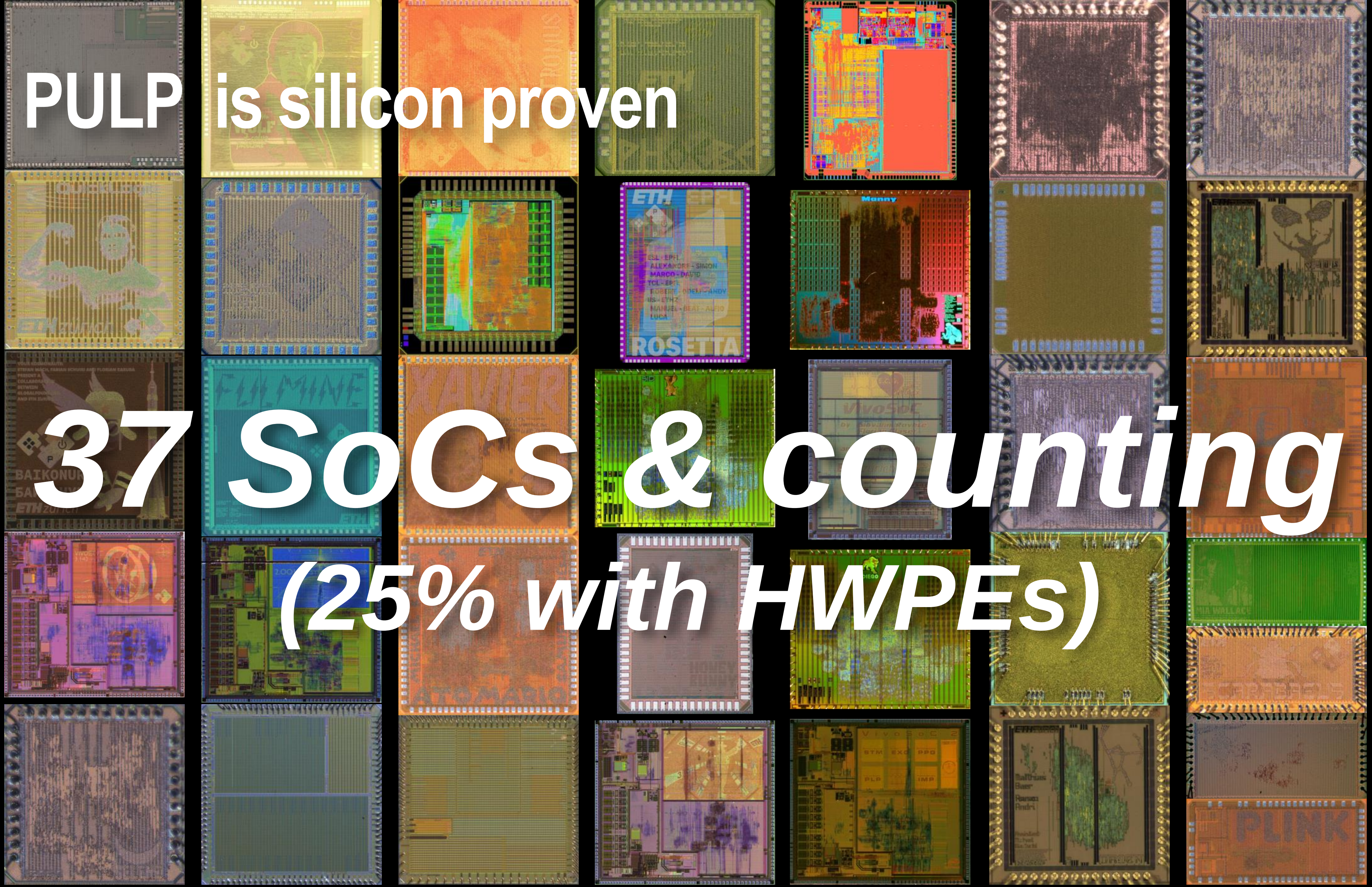
- HERO
- Open Piton

IOT → HPC

Hardware Processing Elements

- NE (DNN)
- NTX (ML)
- FFT
- RedMulE (MatMul)

http://asic.ethz.ch



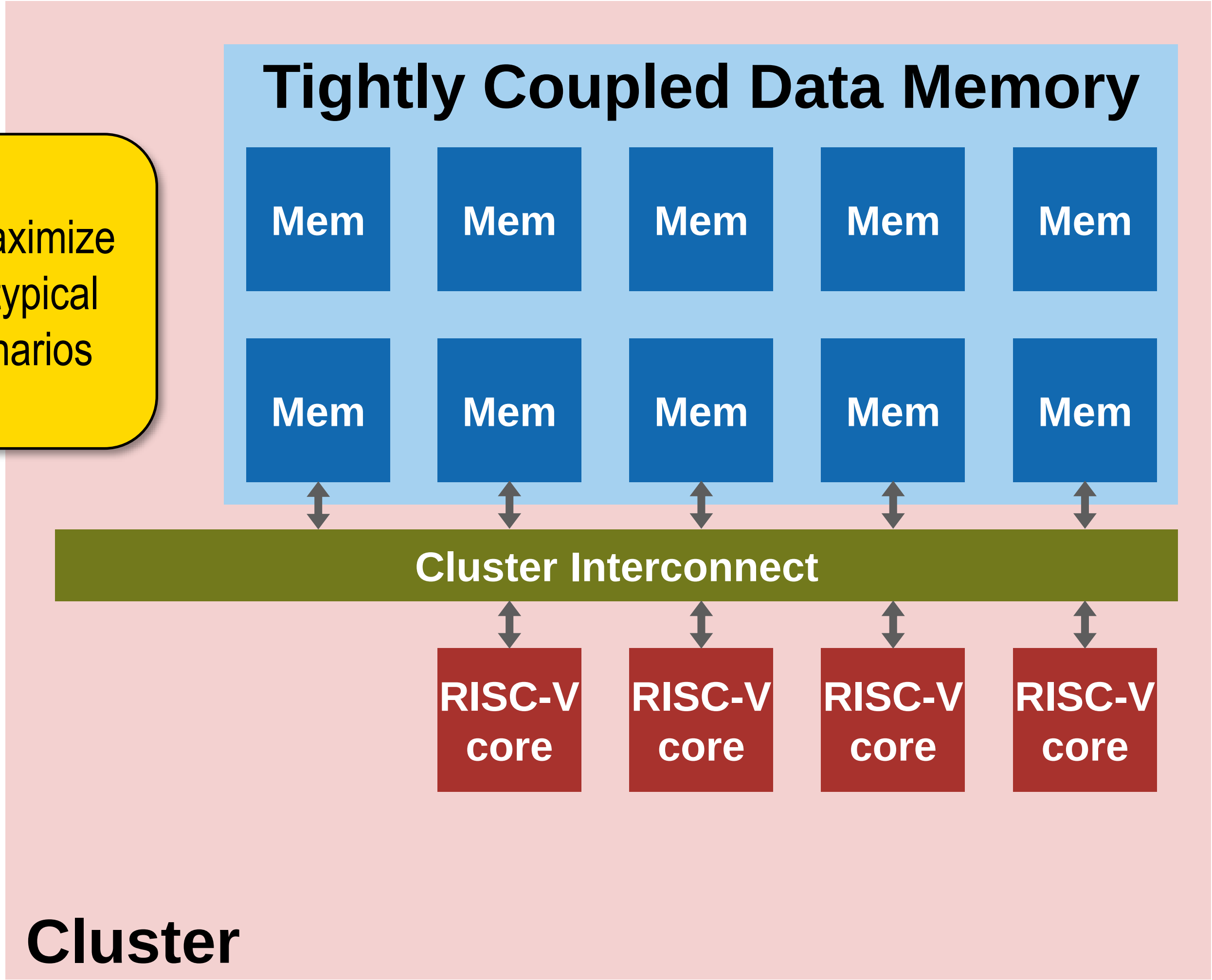
PULP is silicon proven

37 SoCs & counting

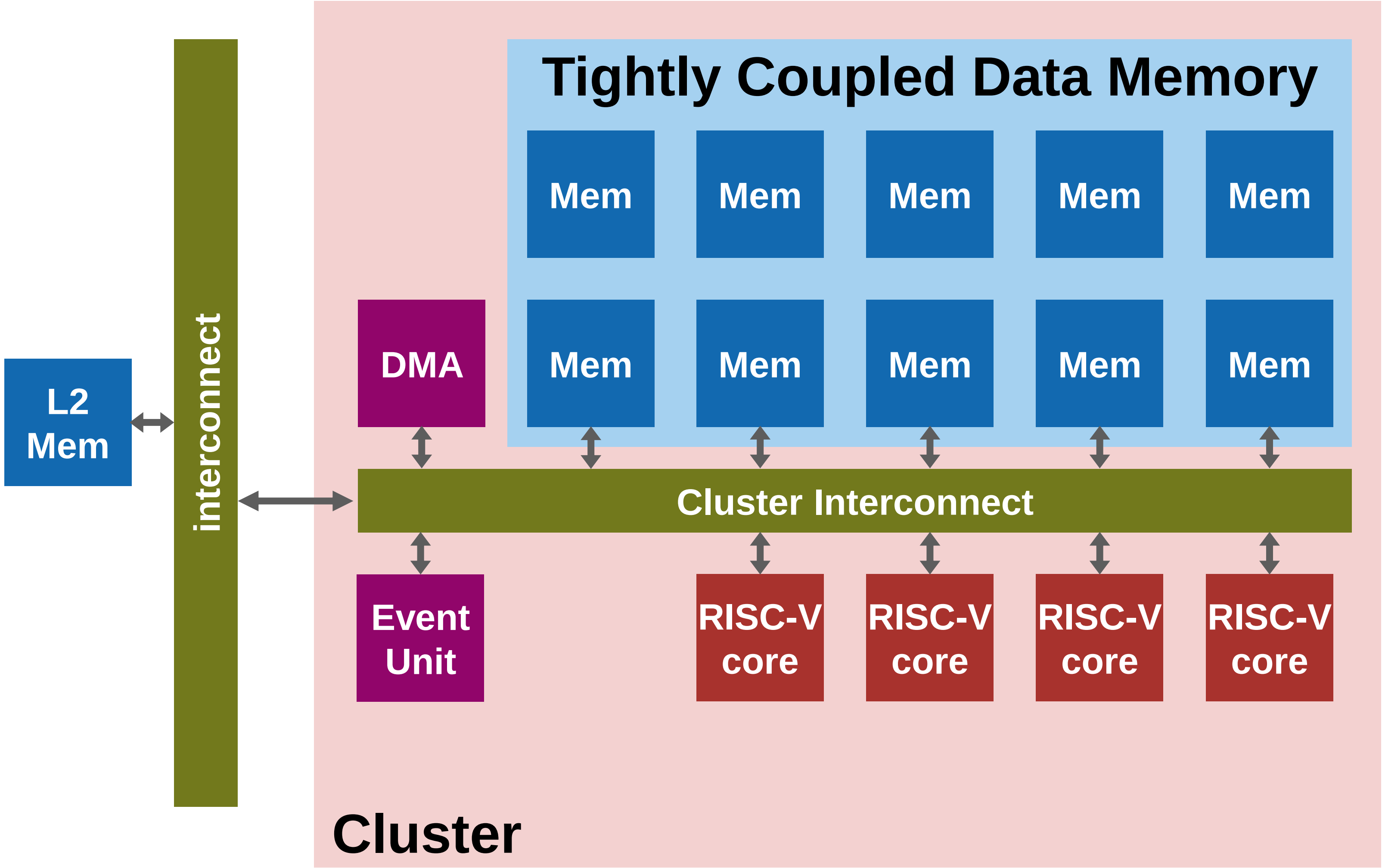
(25% with HWPEs)

PULP Cluster = Parallel RISC-V + Shared-L1 Scratchpad

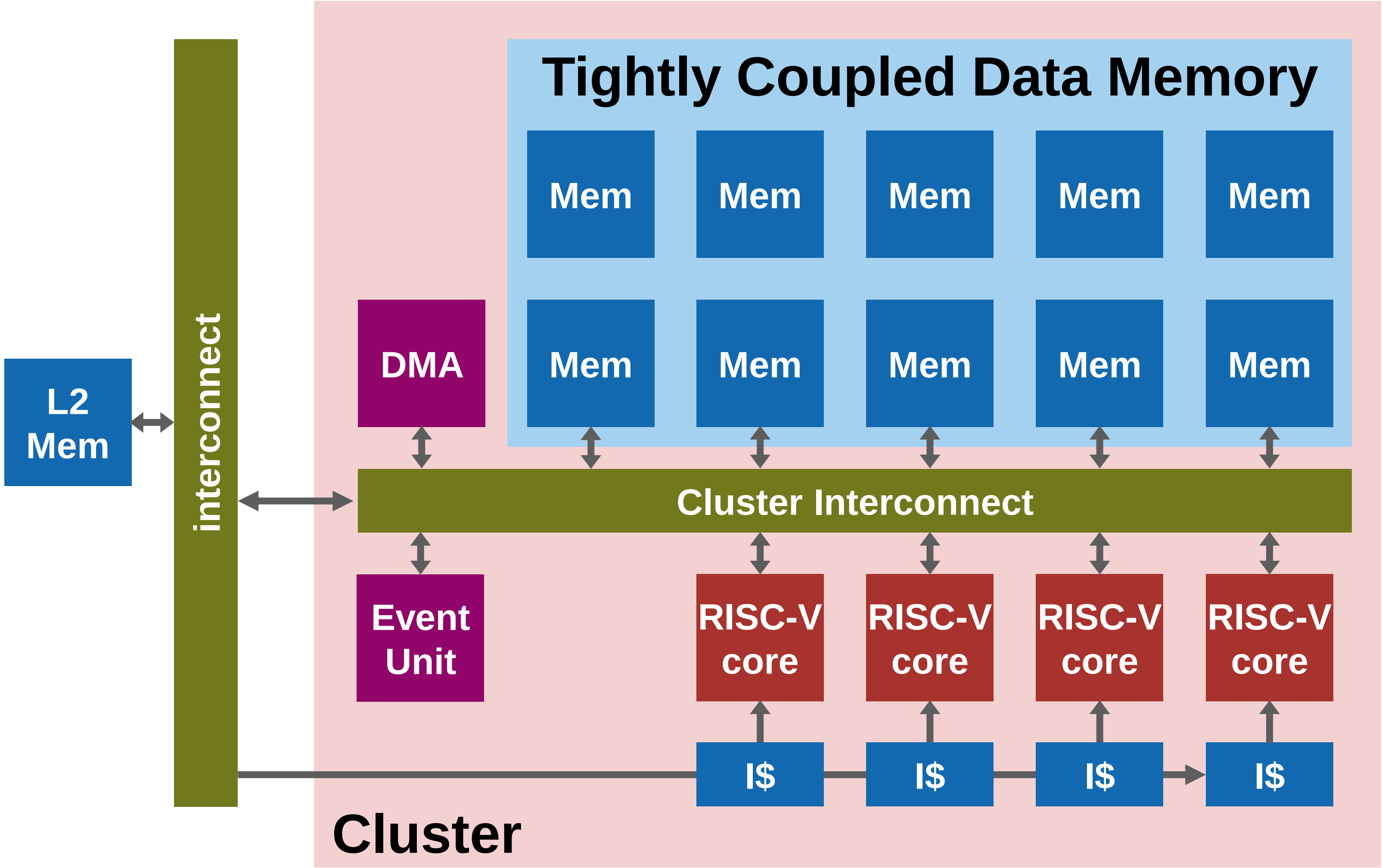
bank interleaving to maximize available bandwidth in typical parallel computing scenarios



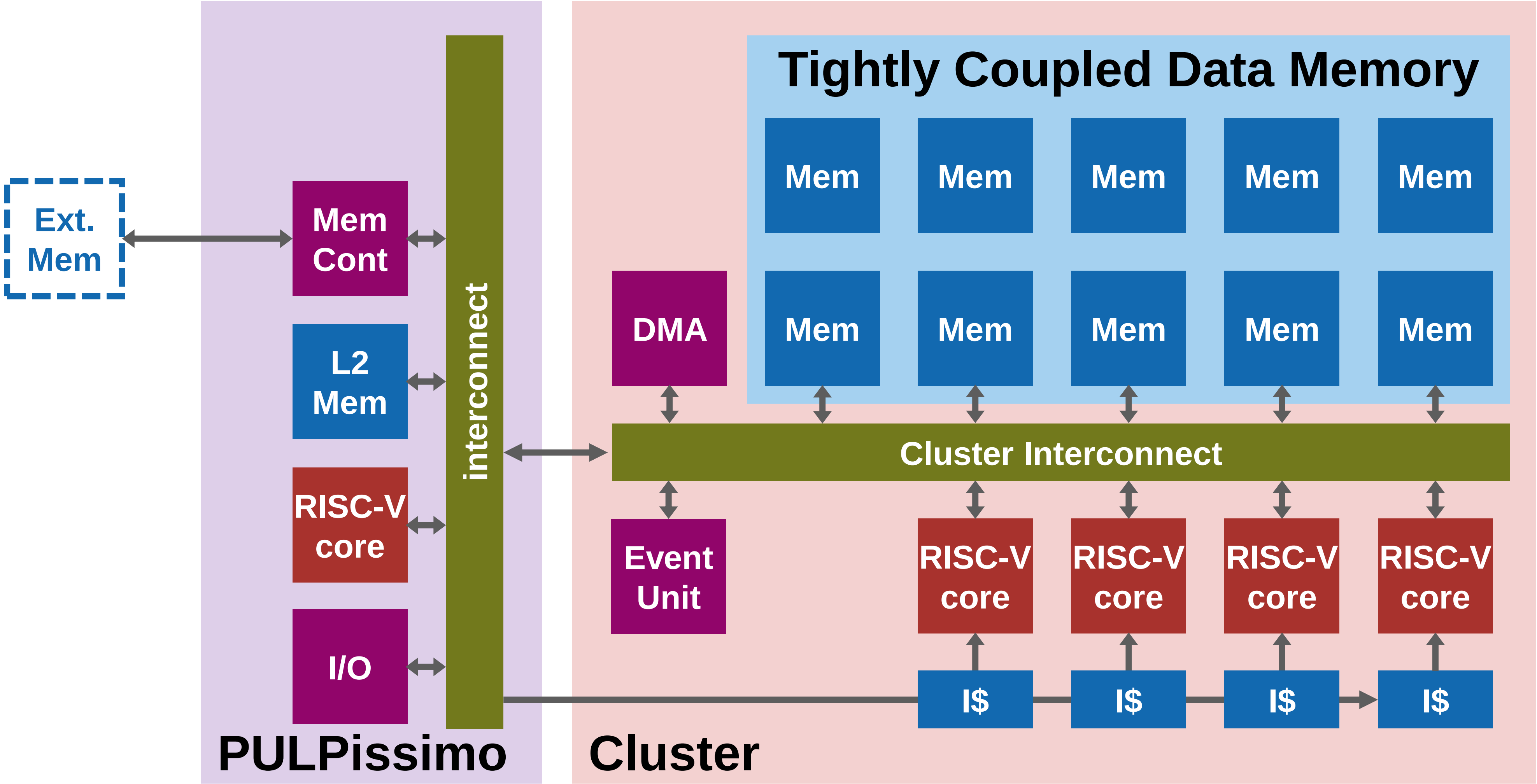
DMA for data transfer + EU to coordination cores



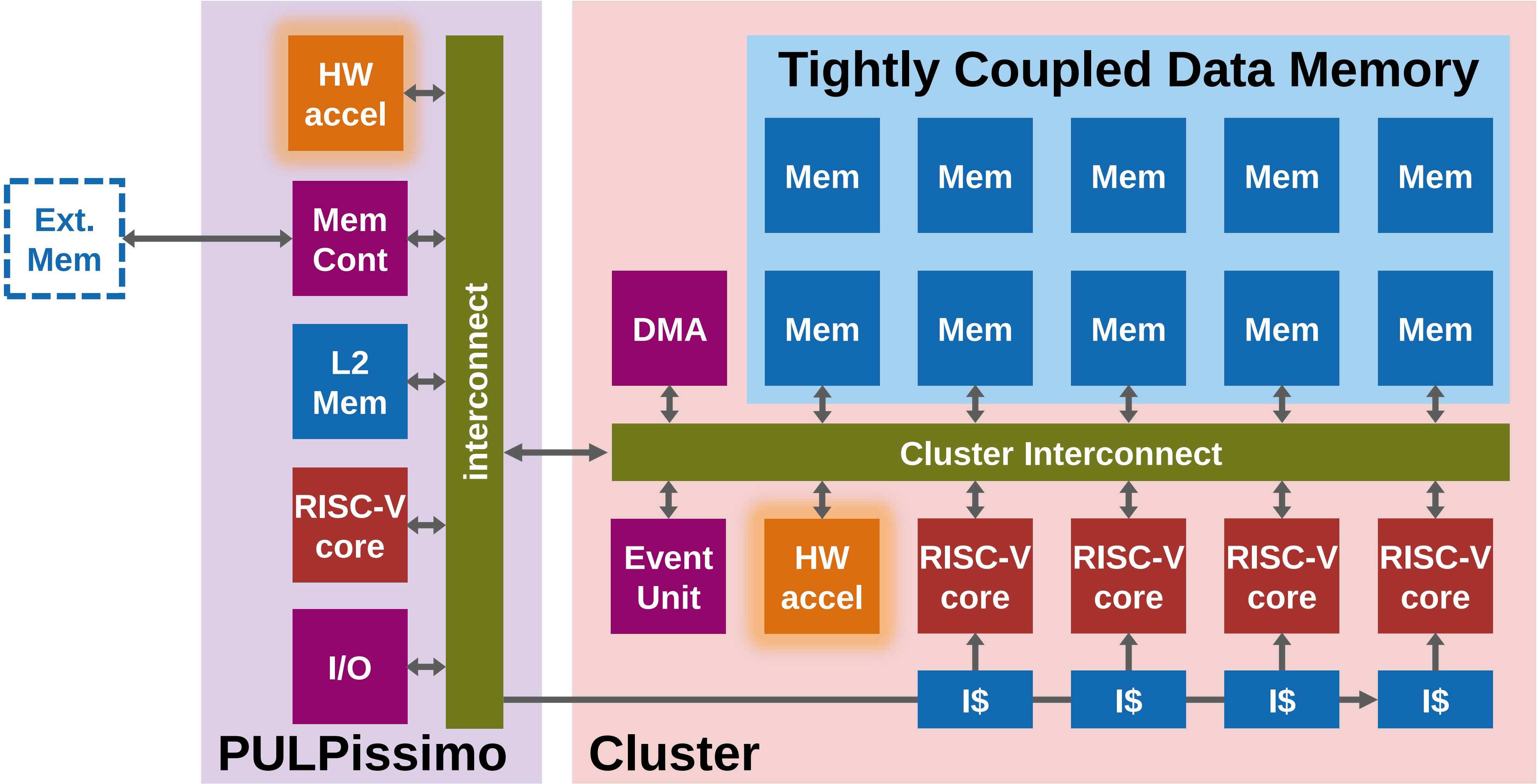
There is a (shared) instruction cache that fetches from L2



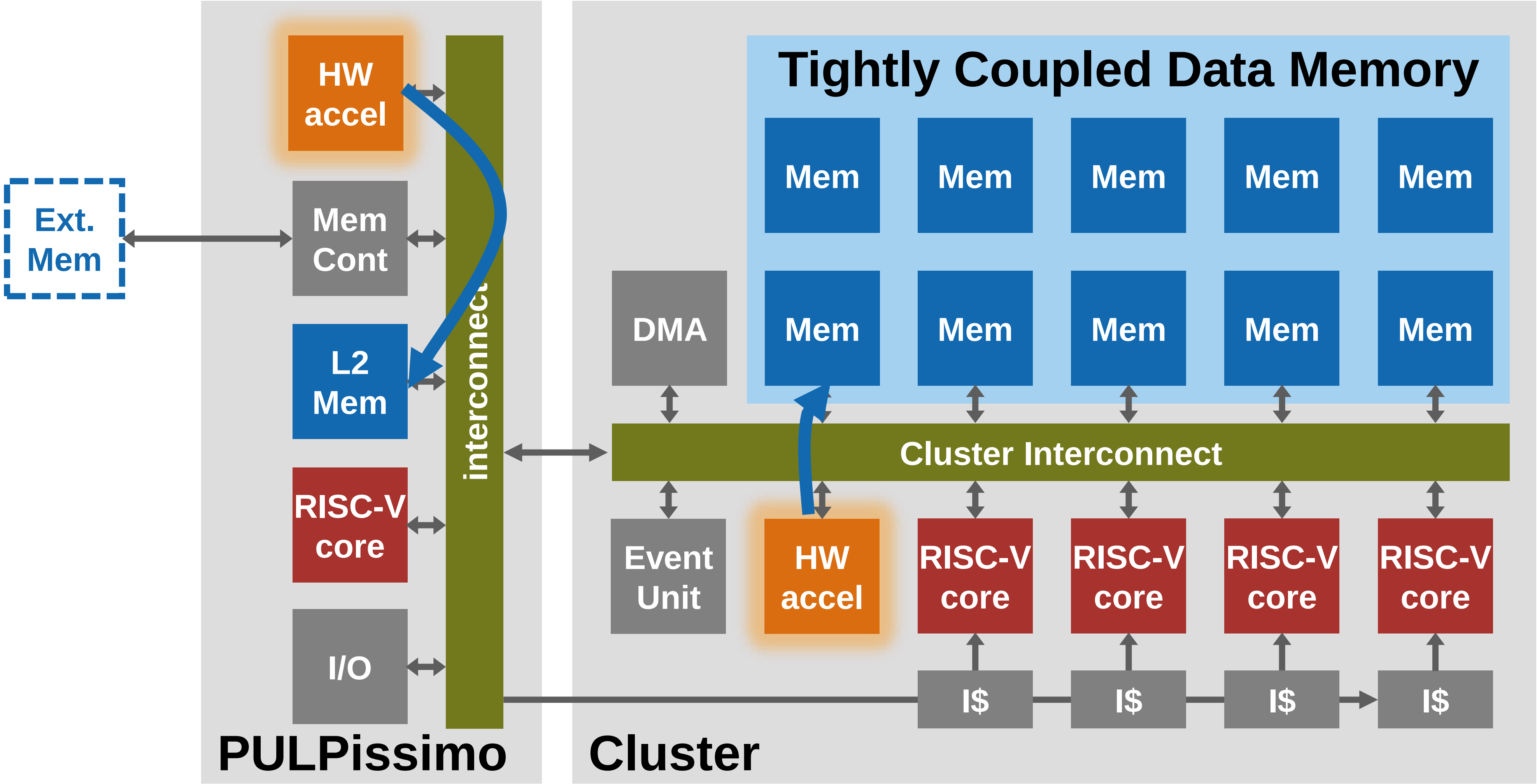
Microcontroller System (PULPissimo) for I/O



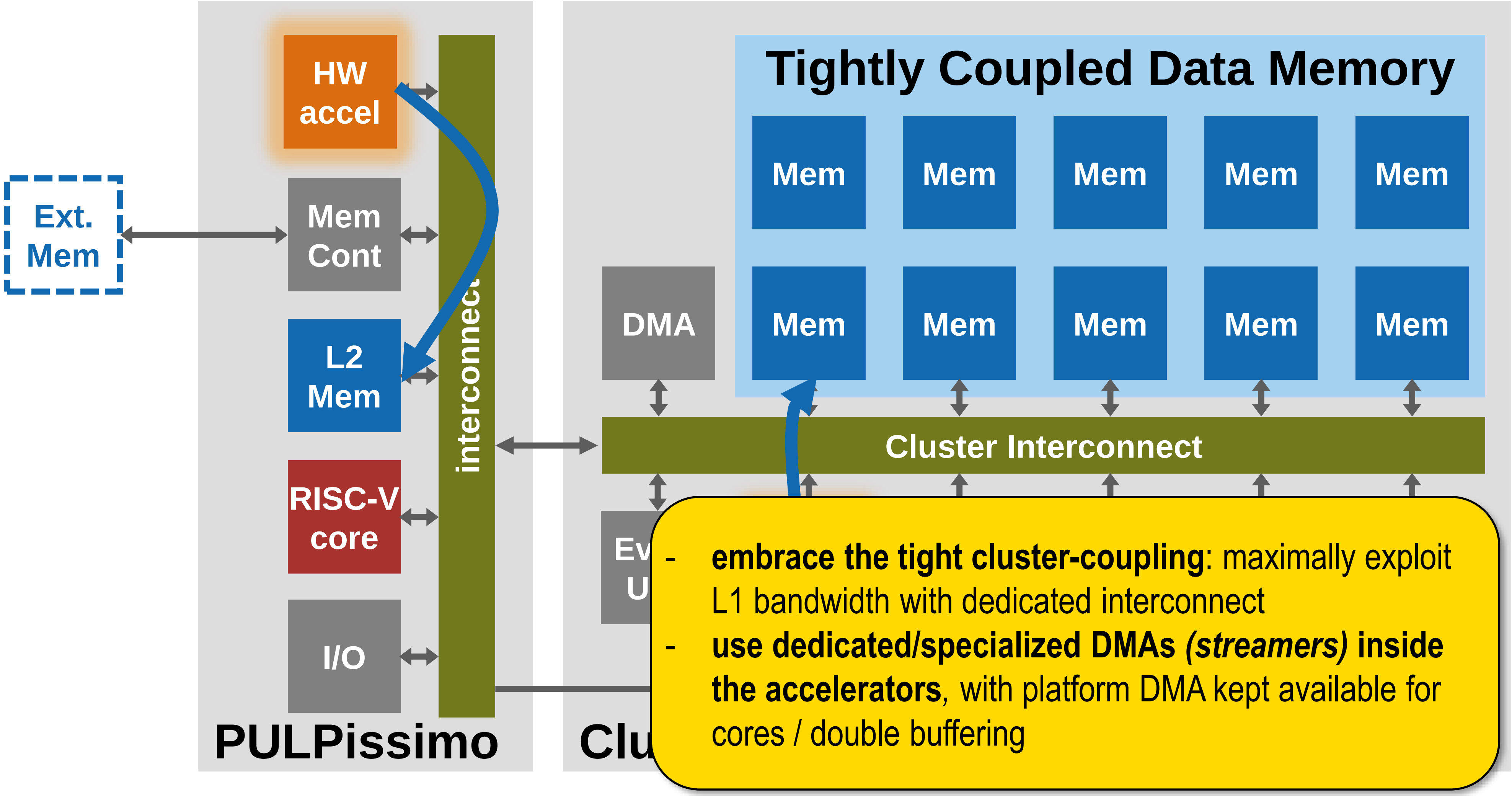
How to couple HW Accerators?



1. High-Bandwidth, Latency-Tolerant access to Memory

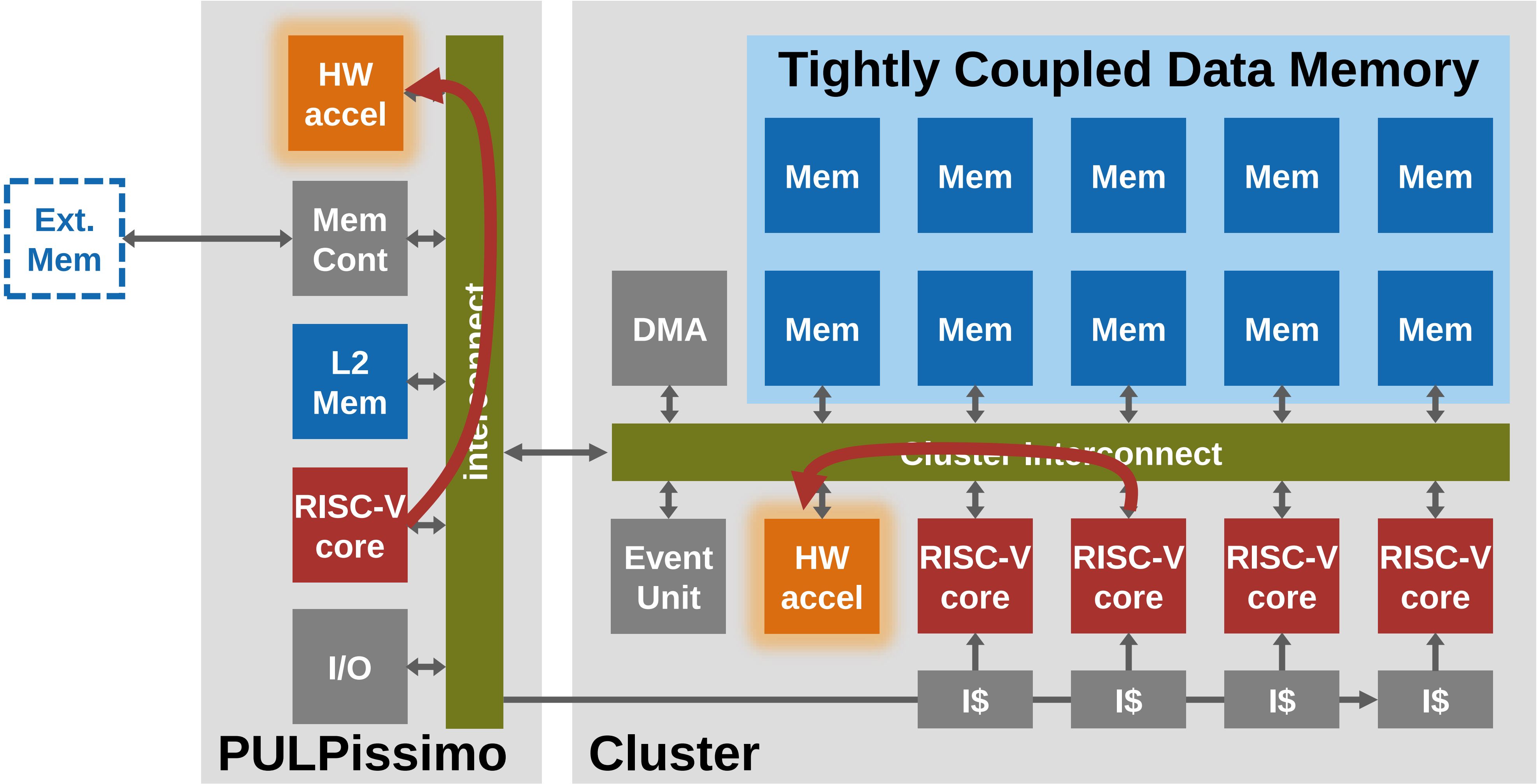


1. High-Bandwidth, Latency-Tolerant access to Memory

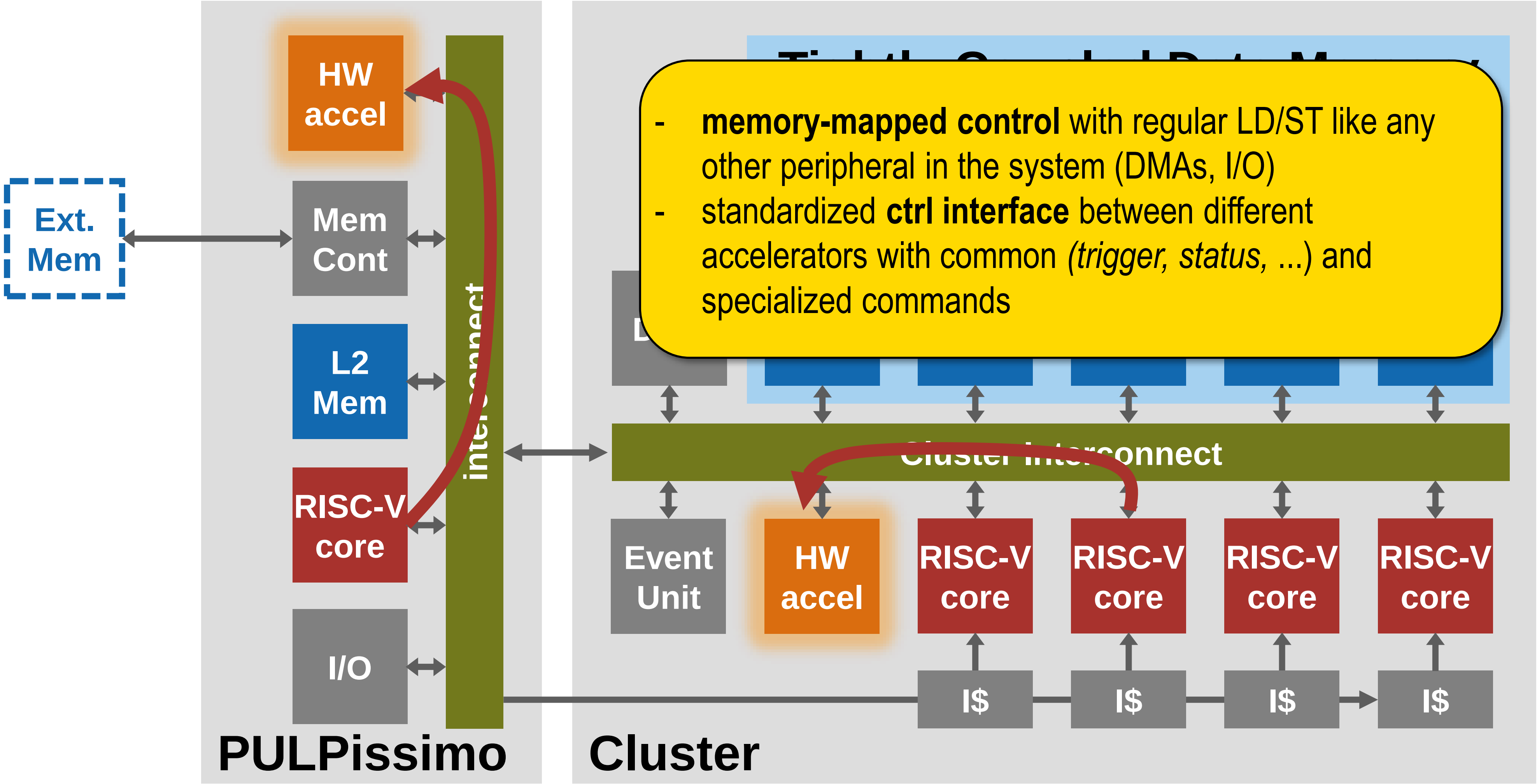


- **embrace the tight cluster-coupling**: maximally exploit L1 bandwidth with dedicated interconnect
- **use dedicated/specialized DMAs *(streamers)* inside the accelerators**, with platform DMA kept available for cores / double buffering

2. Low-Overhead SW Control + Communication

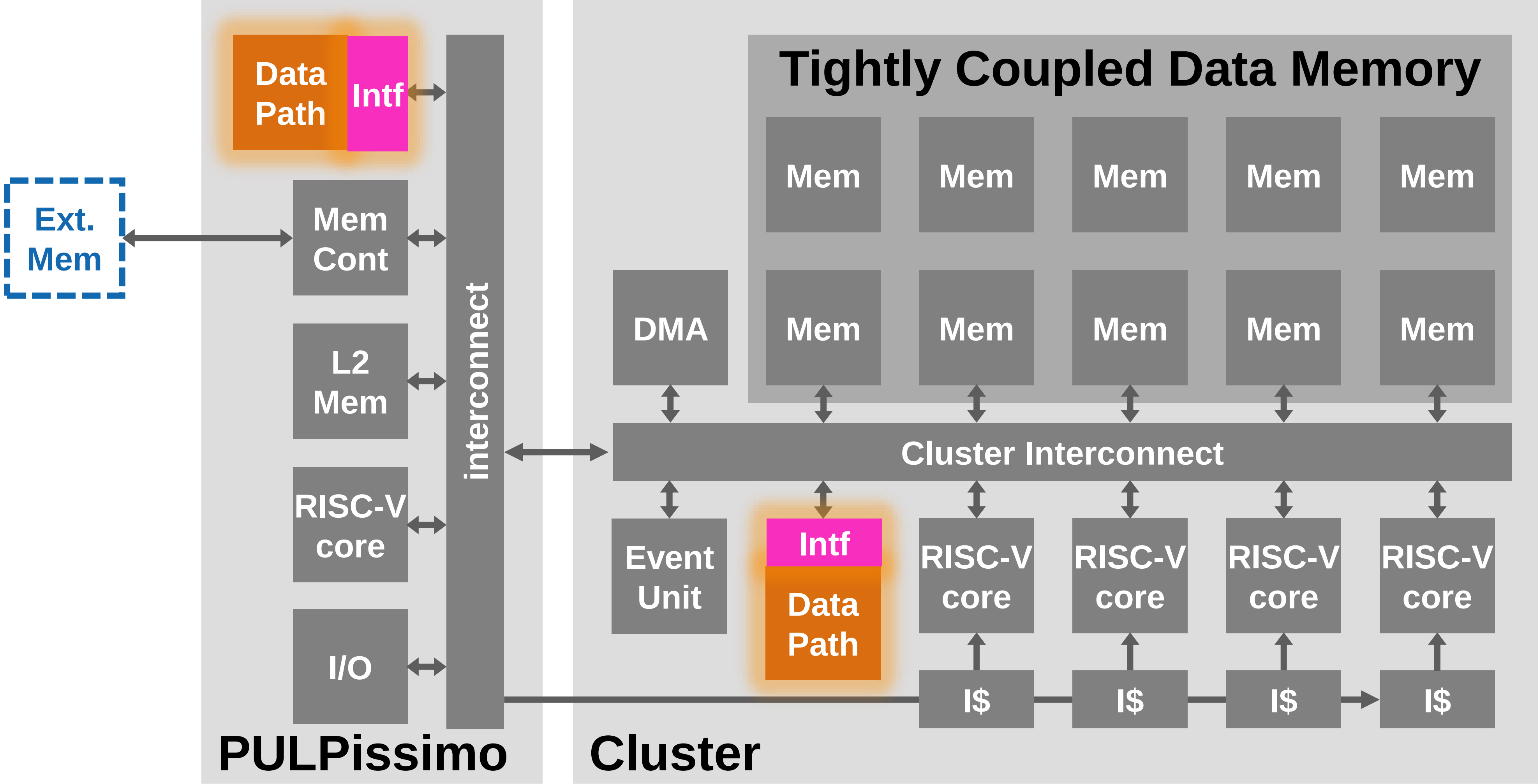


2. Low-Overhead SW Control + Communication

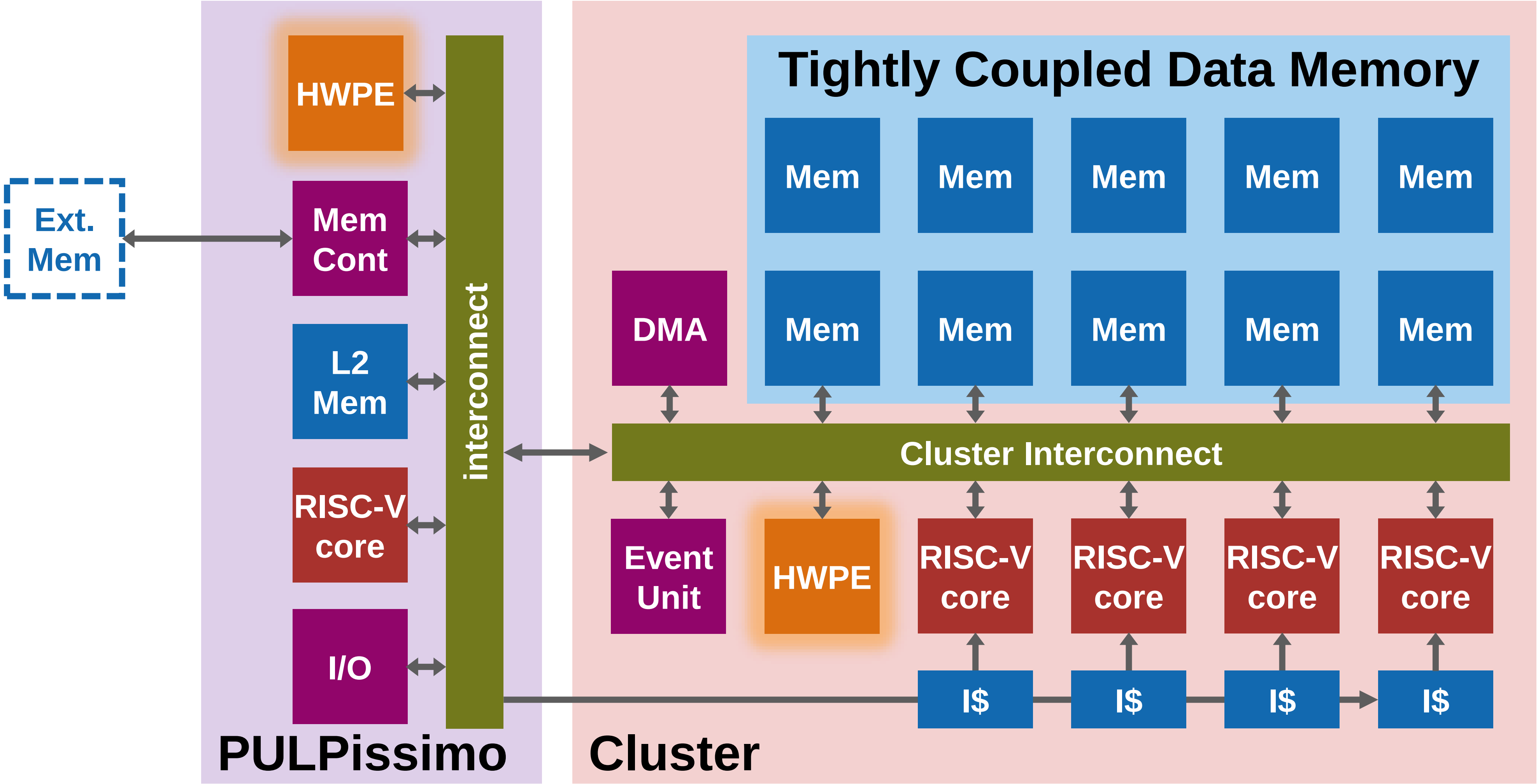


- **memory-mapped control** with regular LD/ST like any other peripheral in the system (DMAs, I/O)
- standardized **ctrl interface** between different accelerators with common *(trigger, status, ...)* and specialized commands

3. Standardized Interface → Redesign only Datapath+Ctrl



1+2+3 = PULP Hardware Processing Engines (HWPEs)



Open-source cluster: https://github.com/pulp-platform/pulp

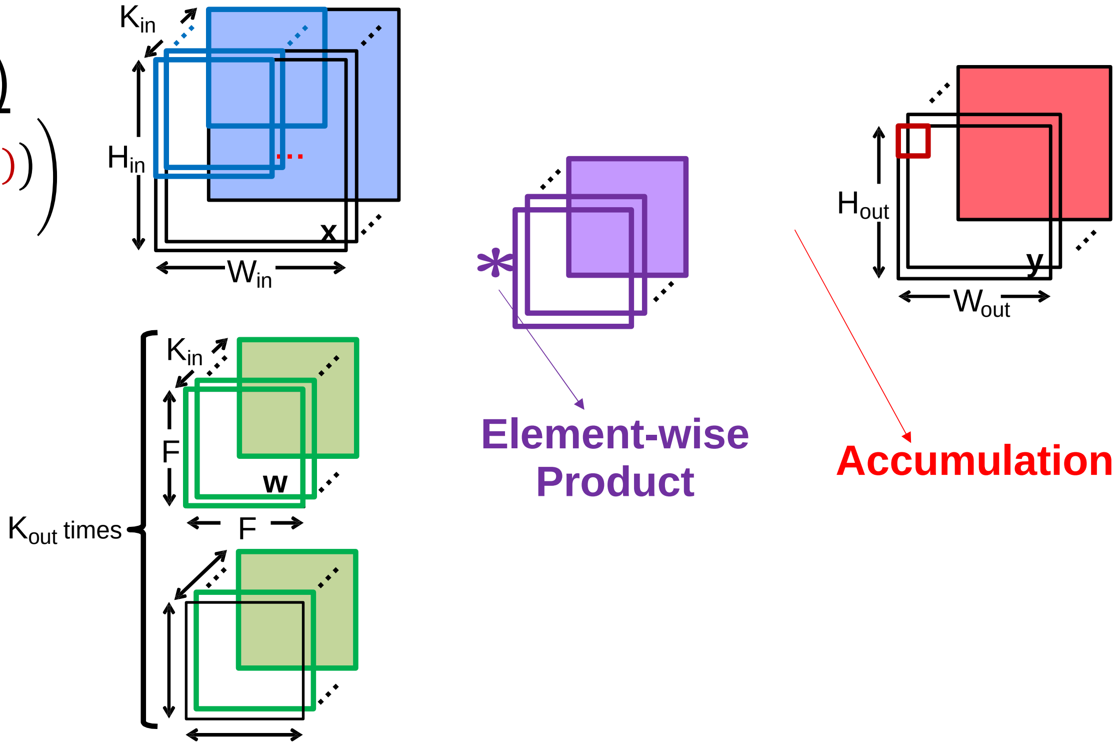
Case Study: the Neural Engine

Flexible weight precision Neural Engine (NE)

$$\mathbf{y}(k_{out}) = \boldsymbol{quant}\left(\sum_{i=0..\boldsymbol{M}} \sum_{filter} \sum_{k_{in}} 2^i \left(\mathbf{W_{bin}}(k_{out}, k_{in}, i) \otimes \mathbf{x}(k_{in})\right)\right)$$

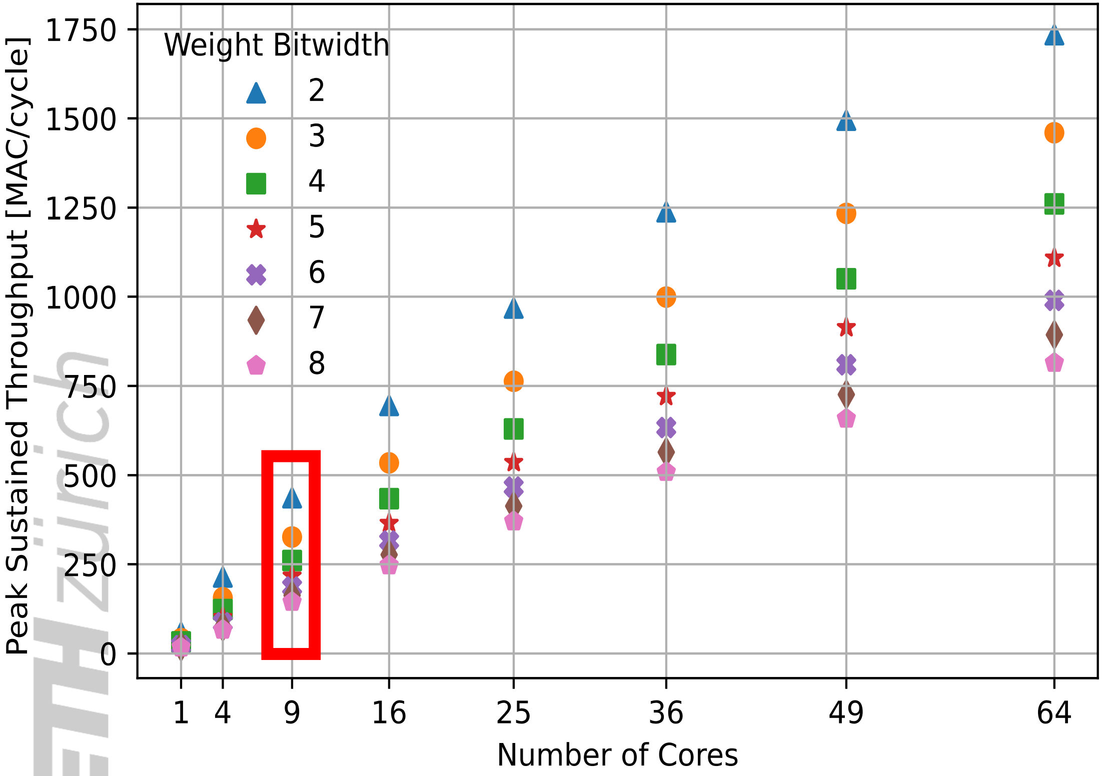
Key NE idea:

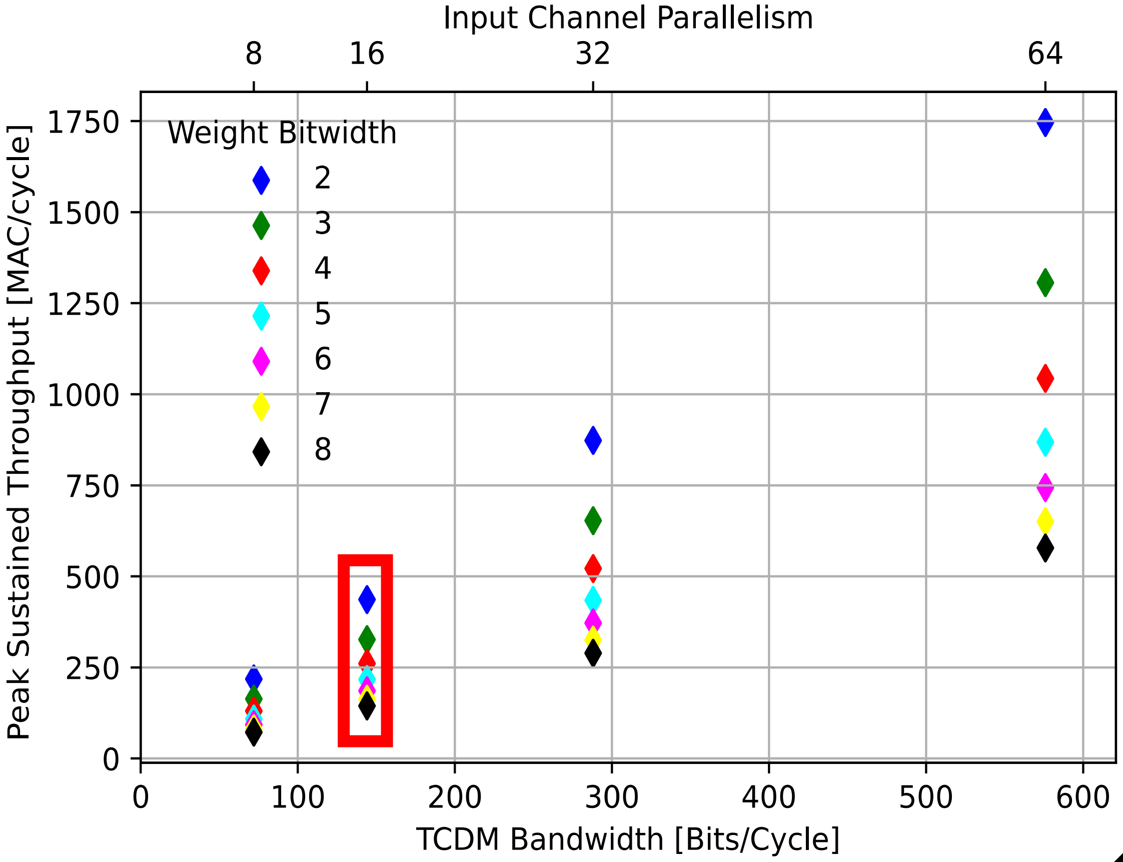
- many **cores**, each dedicated to the receptive field of 1 pixel across **OCP** out channels and **ICP** input channels
- splitting weights in **single bitwise** contributions: flexible weight prec (2-8b), fixed activation prec (8b)
- multiple iterations guided by SW tiling
- realized with **1x8b multipliers + adder trees**



Open-source instance (ICP=16): https://github.com/pulp-platform/ne16

NE Performance Scalability (CONV 3x3)





Performance scalable by changing **number of cores** and **ICP** (design time) or **weight bitwidth** (at run time)

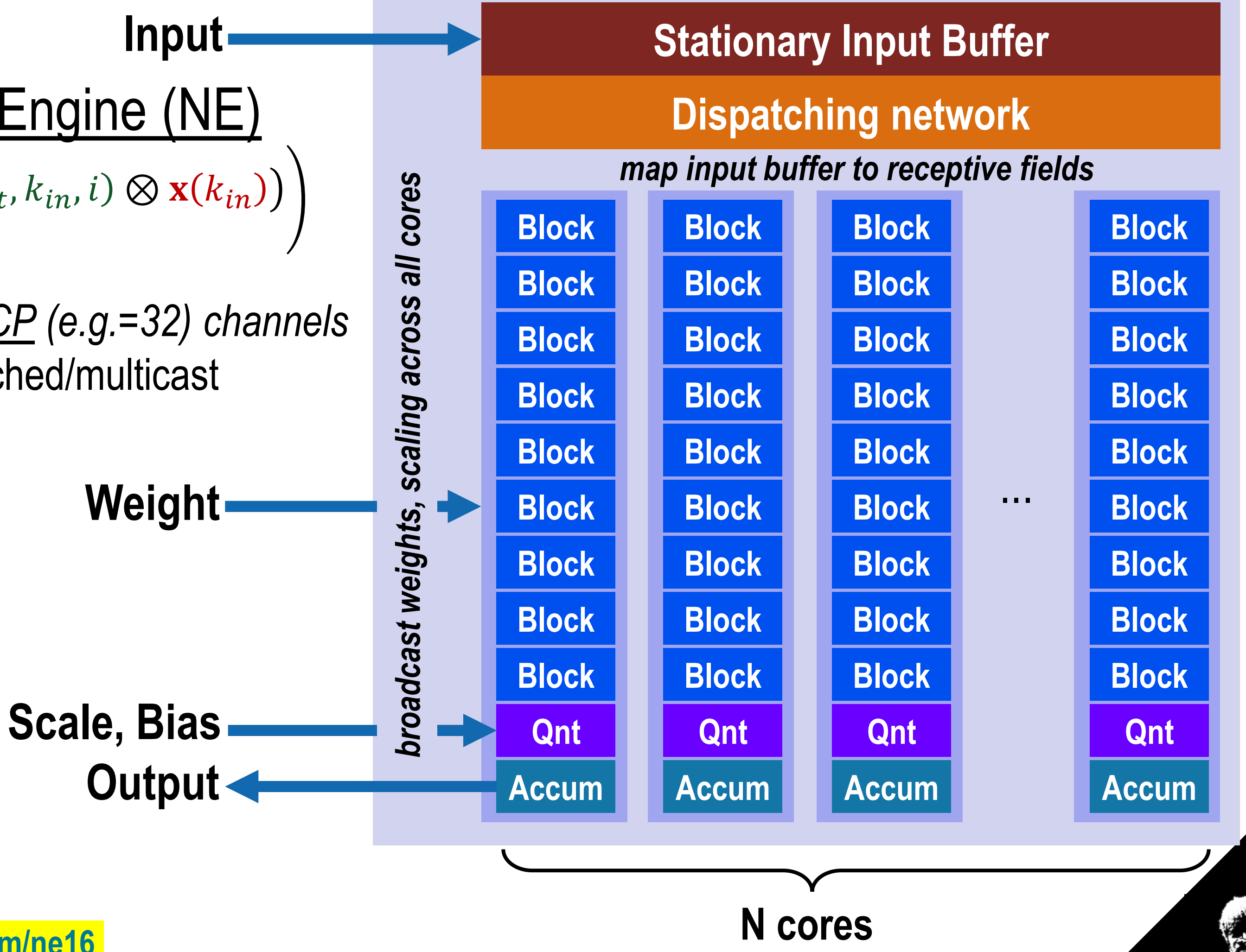
Case Study: the Neural Engine

Flexible weight precision Neural Engine (NE)

$$\mathbf{y}(k_{out}) = \boldsymbol{quant}\left(\sum_{i=0..\boldsymbol{M}} \sum_{filter} \sum_{k_{in}} 2^i \left(\mathbf{W}_{\mathbf{bin}}(k_{out}, k_{in}, i) \otimes \mathbf{x}(k_{in})\right)\right)$$

1 core = receptive field of 1 output px across <u>*OCP*</u> *(e.g.=32) channels*
weights broadcast between all cores inputs dispatched/multicast
according to mode (3x3, 1x1, DepthWise)

N cores, 9 blocks per core
each block contributes
for *ICP* (e.g.=32)

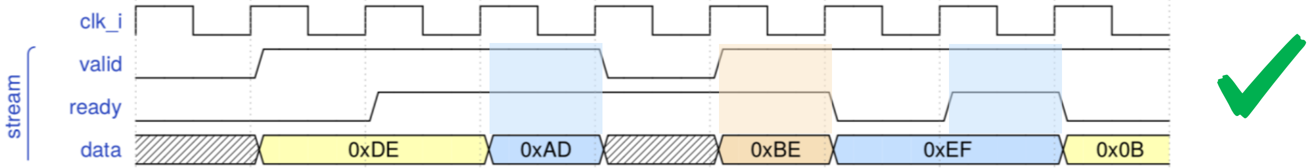


Open-source instance (ICP=16): https://github.com/pulp-platform/ne16

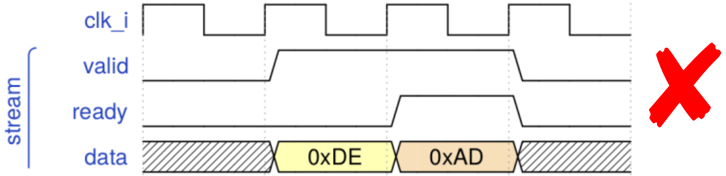
Lightweight, Latency-Tolerant *HWPE-Streams*

Monodirectional, minimal (only handshake + data + optional strobe), no assumptions on content. Positional information carried by order of data packets instead of address → **meant to represent data inside accelerator**

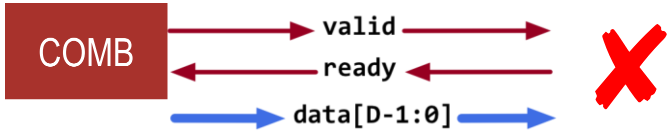
1. handshake happens when valid & ready = 1



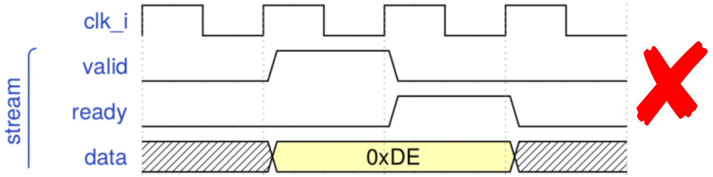
2. data/strobe can only change 1) following a handshake, 2) when valid is deasserted



3. assertion of valid (0 to 1) cannot depend combinationally on ready

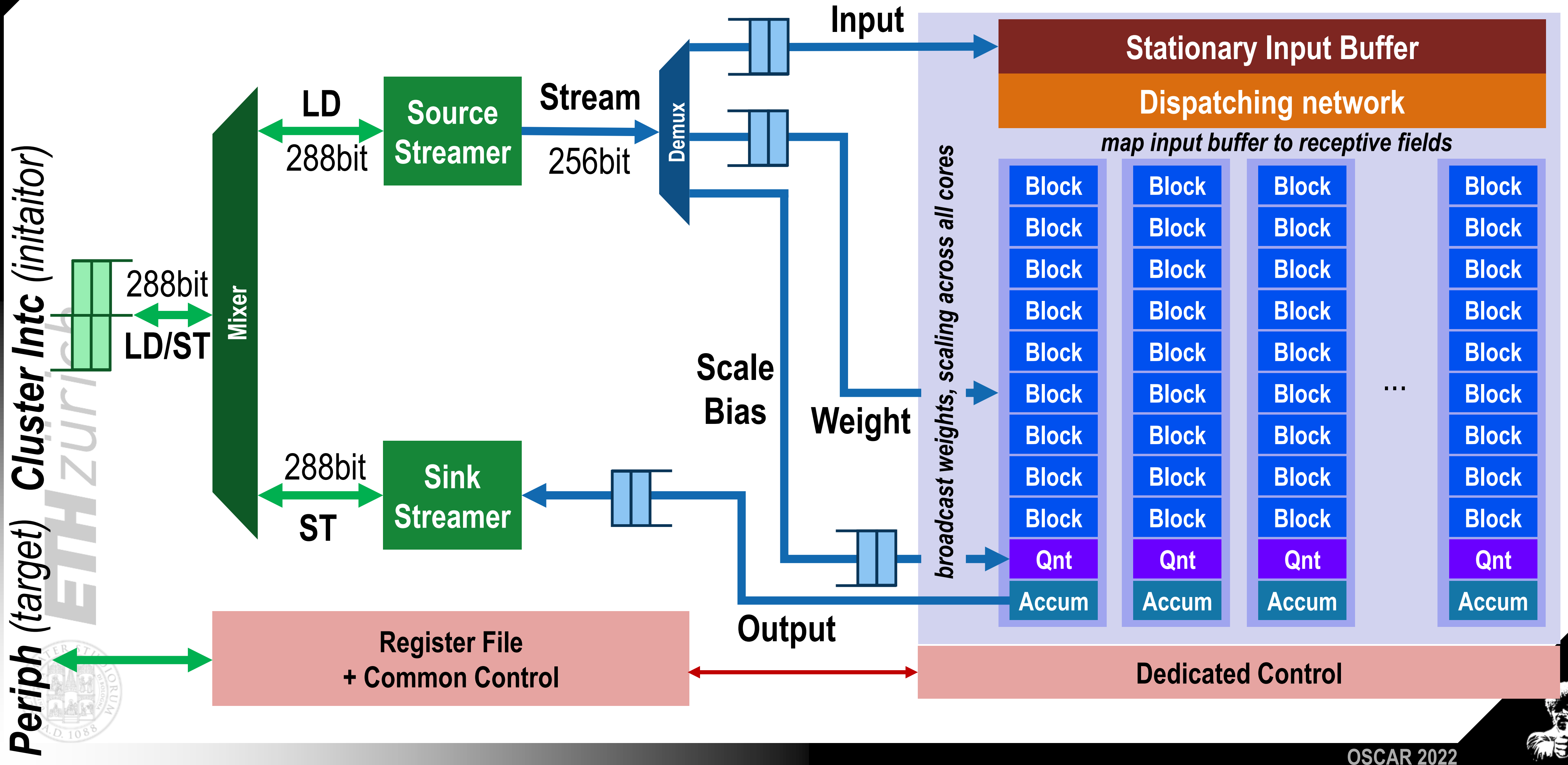


4. deassertion of valid (1 to 0) can only happen in the cycle after a handshake

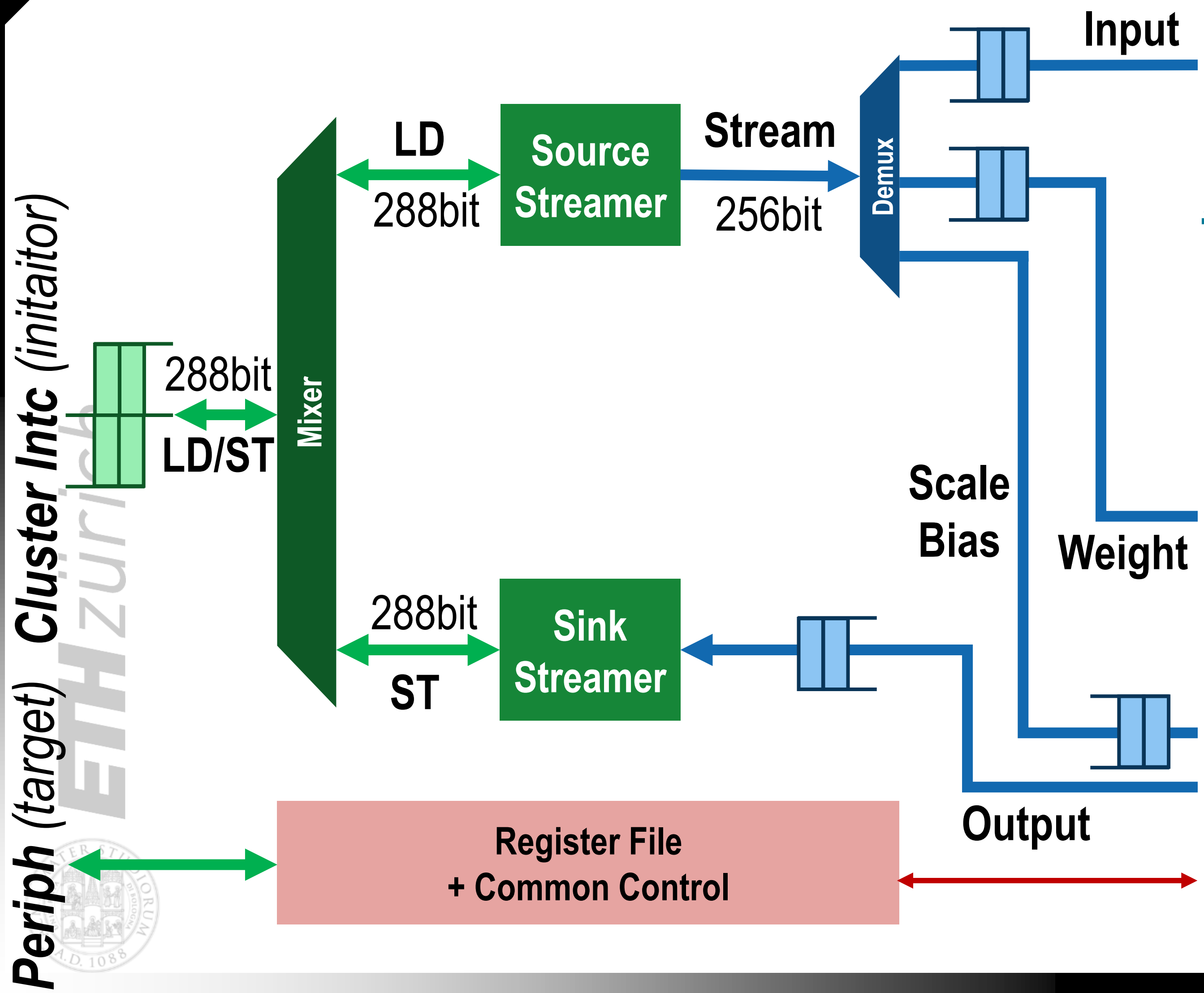


Open-source! https://github.com/pulp-platform/hwpe-stream

Case Study: the Neural Engine



Case Study: the Neural Engine



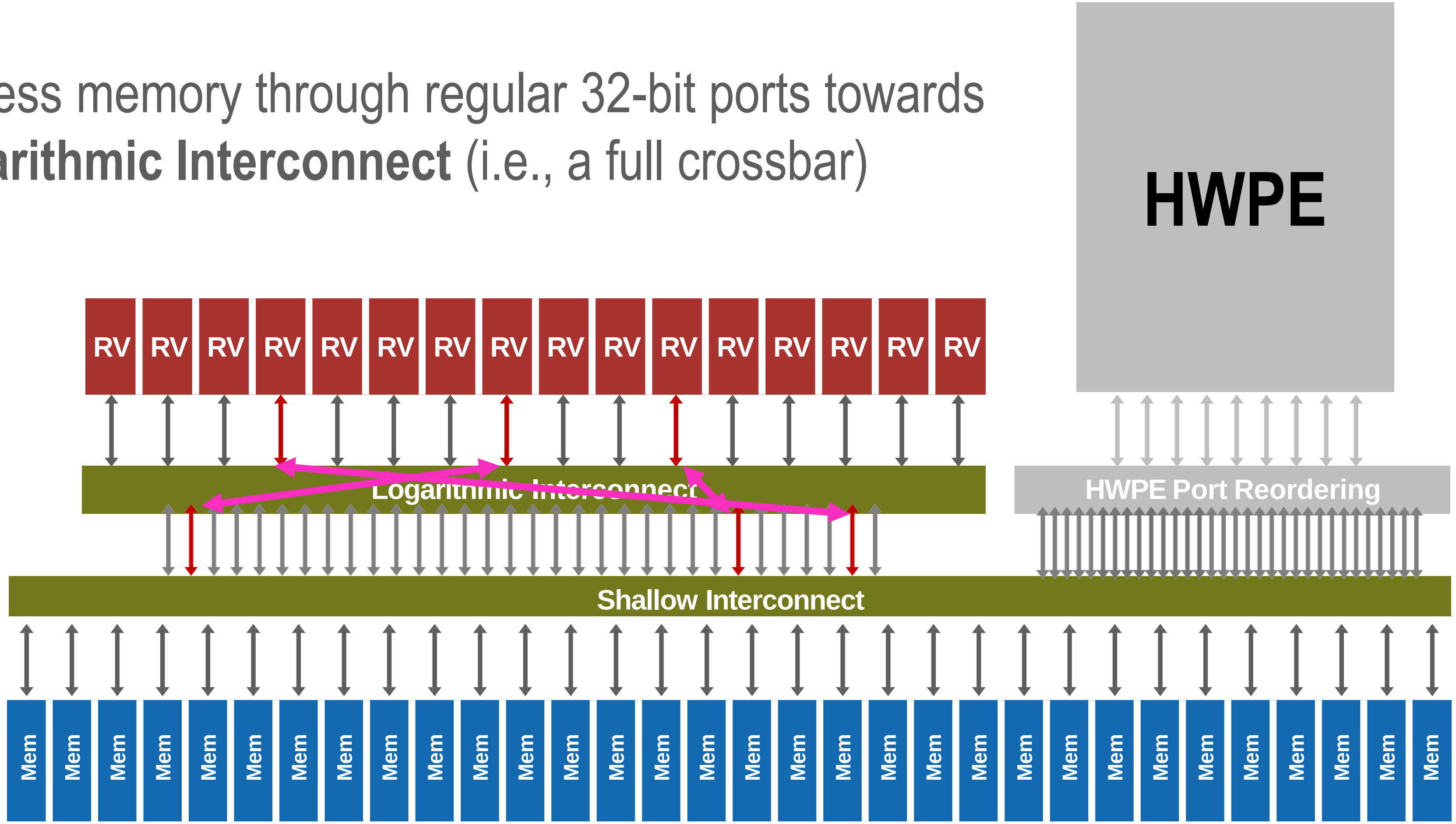
Fully built with open-source HWPE IPs
[***https://hwpe-doc.readthedocs.io***](https://hwpe-doc.readthedocs.io)

Interface designed by composition of open-source IPs for control + memory interfaces through ***HWPE-streams*** → specialized **streamers** convert from streams to mem (TCDM)

Memory access capabilities equivalent or superior to SW (misaligned access, delayed stores, high bandwidth) on part or all of the memory map

High-Bandwidth access: Heterogeneous Cluster Interconnect

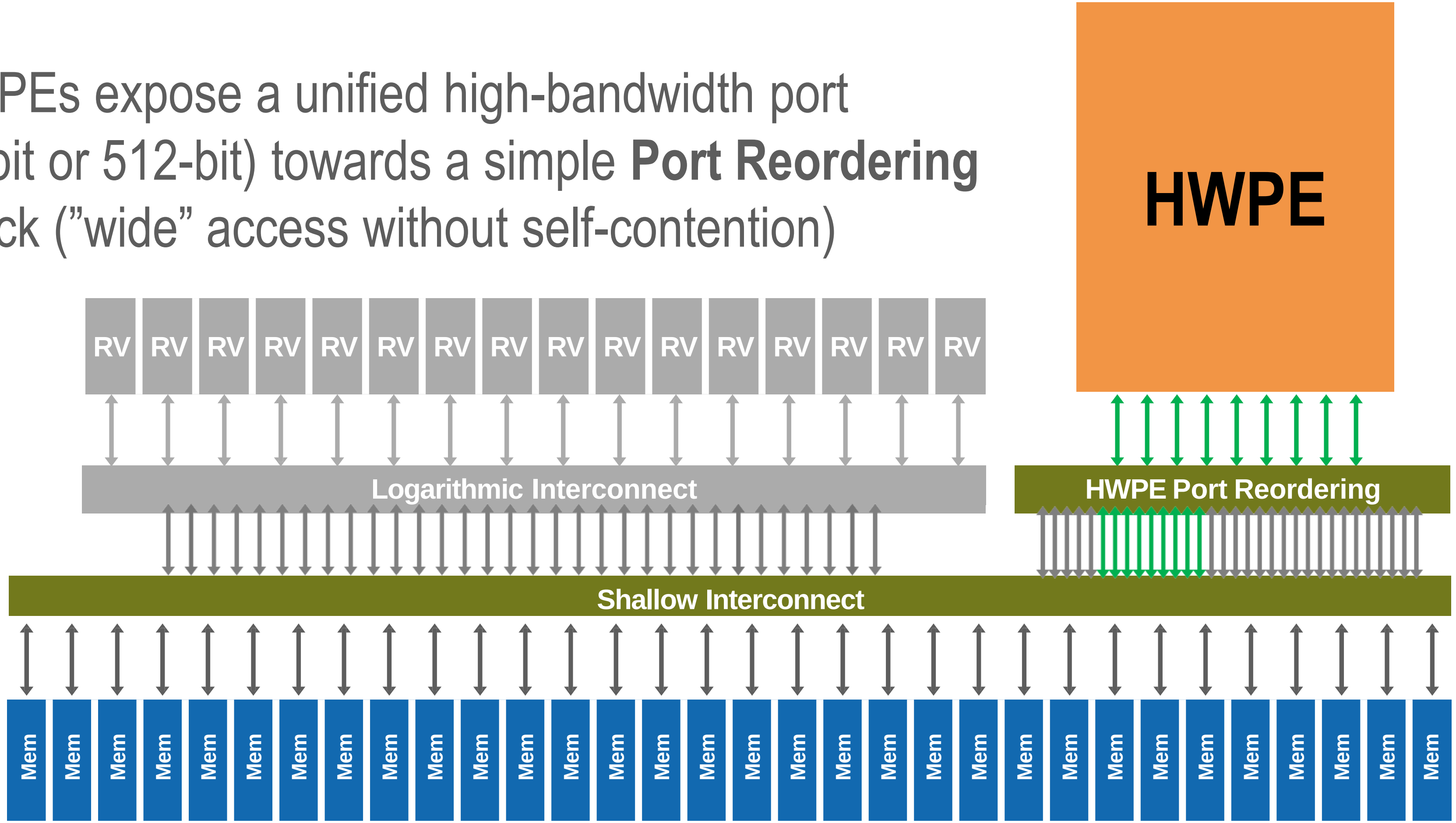
Cores access memory through regular 32-bit ports towards **Logarithmic Interconnect** (i.e., a full crossbar)



Open-source! https://github.com/pulp-platform/hci

High-Bandwidth access: Heterogeneous Cluster Interconnect

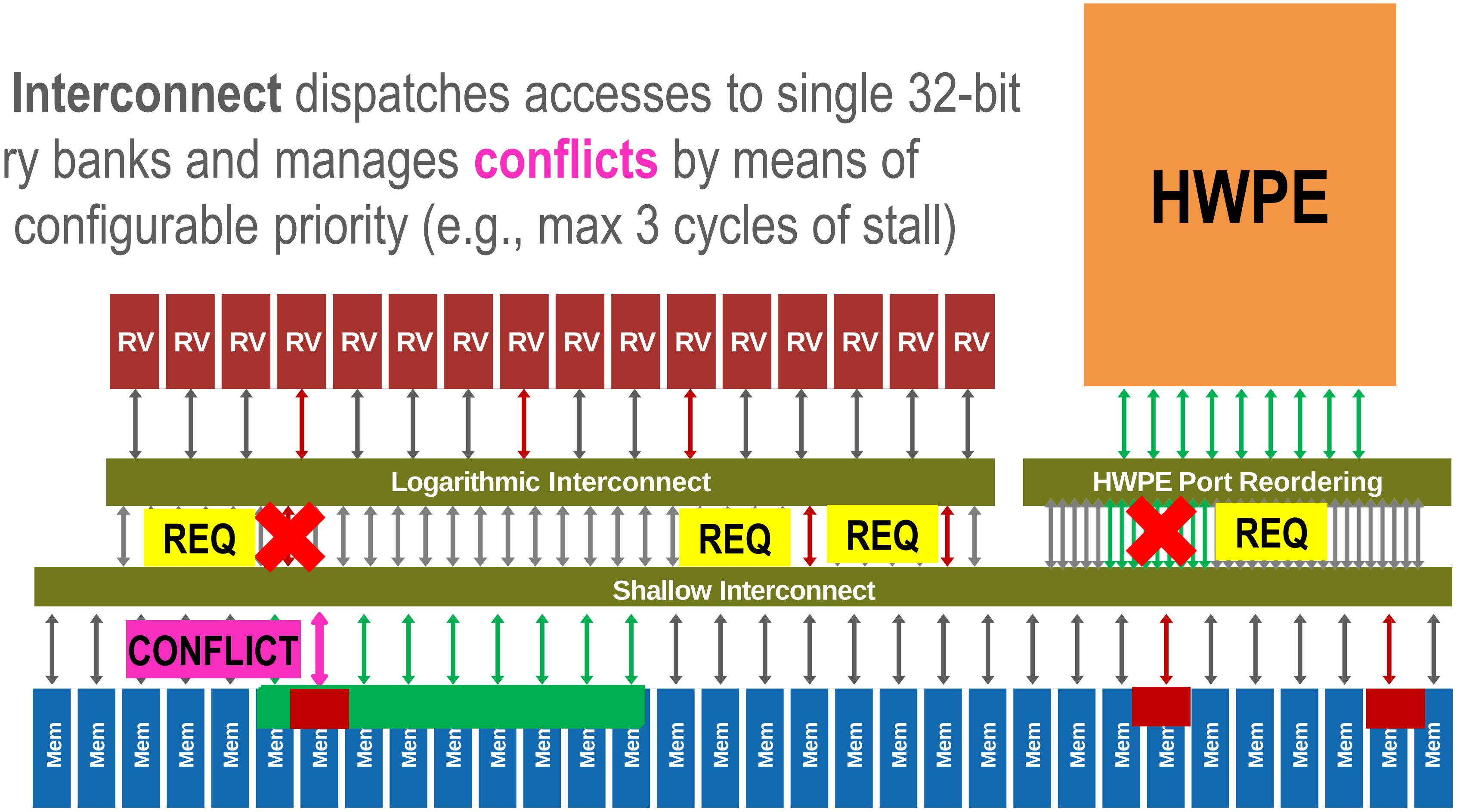
HWPEs expose a unified high-bandwidth port (e.g., 256-bit or 512-bit) towards a simple **Port Reordering** block (”wide” access without self-contention)



Open-source! https://github.com/pulp-platform/hci

High-Bandwidth access: Heterogeneous Cluster Interconnect

A **Shallow Interconnect** dispatches accesses to single 32-bit memory banks and manages **conflicts** by means of rotating configurable priority (e.g., max 3 cycles of stall)

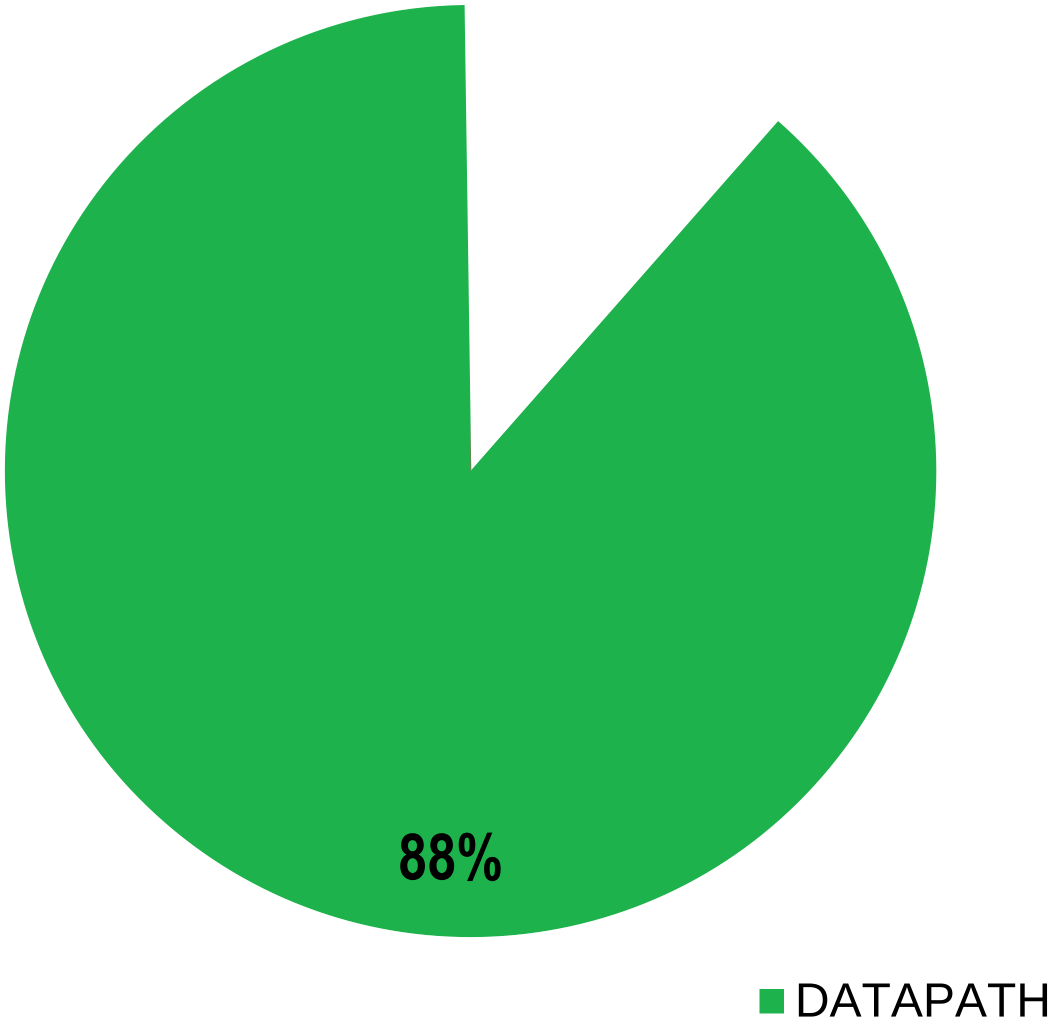

HWPE
RV RV RV RV RV RV RV RV RV RV RV RV RV RV RV RV
Logarithmic Interconnect
HWPE Port Reordering
REQ
REQ
REQ
REQ
Shallow Interconnect
CONFLICT
Mem Mem Mem Mem Mem Mem Mem Mem Mem Mem Mem Mem Mem Mem Mem Mem Mem Mem Mem Mem Mem Mem Mem Mem Mem Mem Mem Mem Mem Mem Mem Mem

Open-source! https://github.com/pulp-platform/hci

Case study: NE area

Considering **NE**
with **ICP=16**, **OCP=32**, **9 cores**

Datapath accounts for **88%**
of area of full accelerator

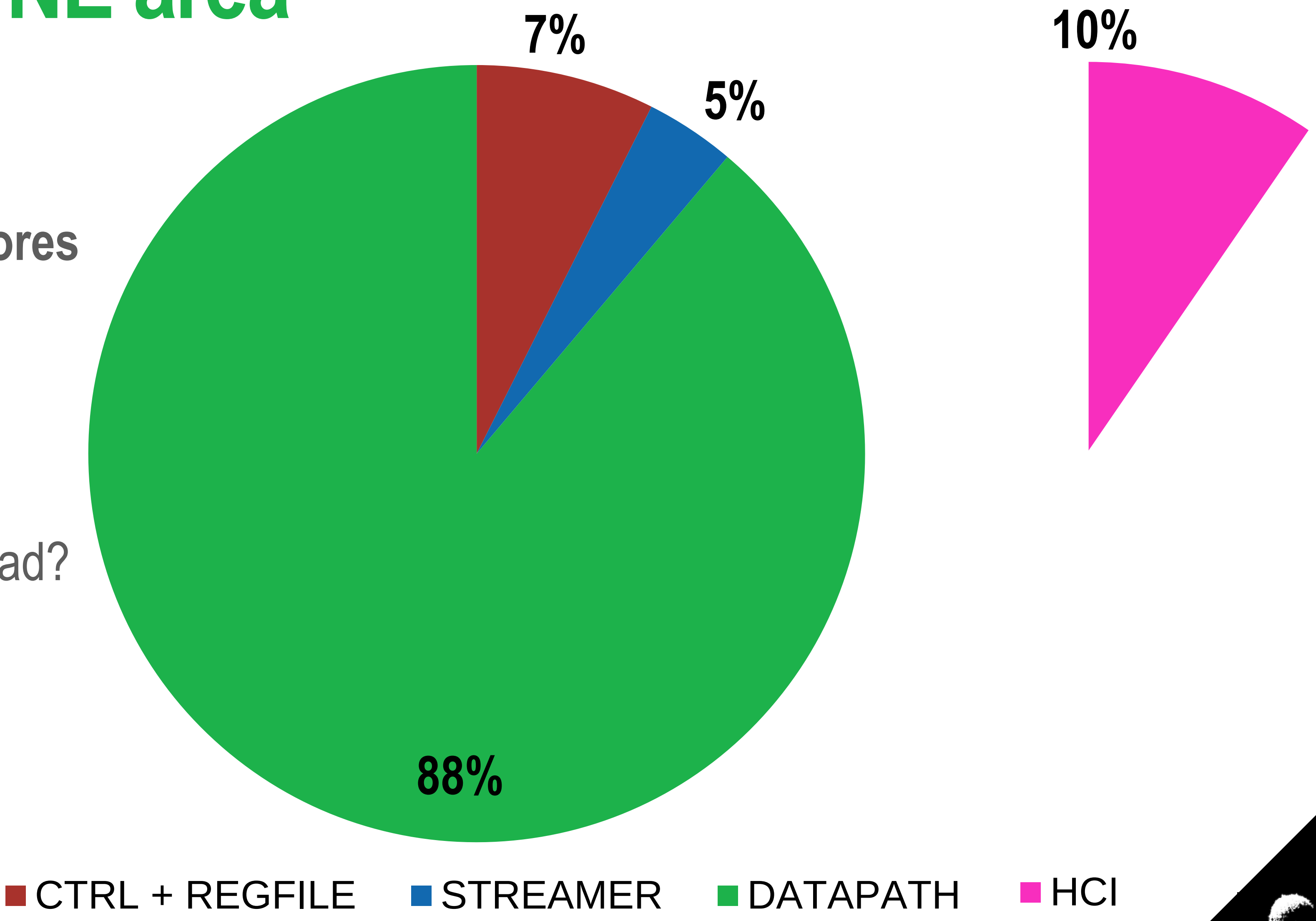


Case study: NE area

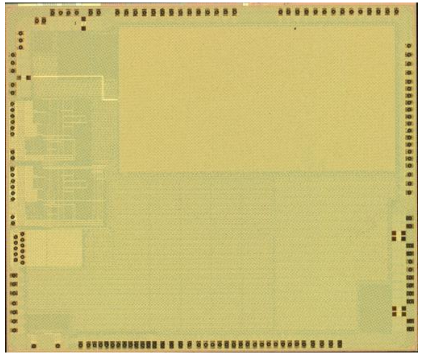
Considering **NE**
with **ICP=16**, **OCP=32**, **9 cores**

Datapath accounts for **88%**
of area of full accelerator

What about full HCI overhead?
Log intc + HWPE intc →
~10% of accelerator area!



HW-acceleration playground: success stories

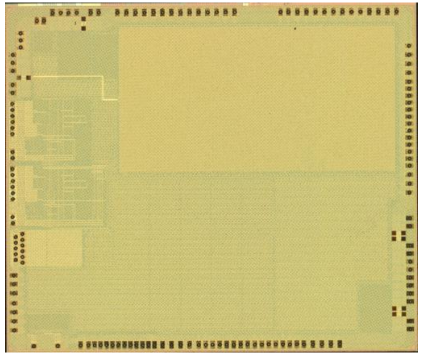


Vega 22nm, *ISSCC'21 with HW Conv Engine (UNIBO + ETHZ + GreenWaves)*

ECHOES
a PULP chip by
Mattia Sinigaglia, Luca Valente,
Luca Bertaccini, Angelo Garofalo,
Francesco Conti, Davide Rossi

Darkside 65nm, *accepted to ESSCIRC'22* + **Echoes 65nm**
Master student tape-outs with many HWPEs:
Transposer, FFT, Systolic Array, Depthwise Conv... (UNIBO+ETHZ)

HW-acceleration playground: success stories



Vega 22nm, *ISSCC'21 with HW Conv Engine (UNIBO + ETHZ + GreenWaves)*

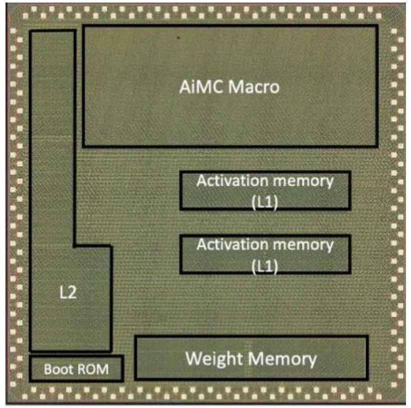
ECHOES
a PULP chip by
Mattia Sinigaglia, Luca Valente,
Luca Bertaccini, Angelo Garofalo,
Francesco Conti, Davide Rossi

Darkside 65nm, *accepted to ESSCIRC'22* + **Echoes 65nm**
Master student tape-outs with many HWPEs:
Transposer, FFT, Systolic Array, Depthwise Conv... (UNIBO+ETHZ)

Not only ETHZ/UNIBO:

GAP9 SoC ***(commercial)***
with NE16 (GreenWaves)

DIANA 22nm, *ISSCC'22*
PULPissimo + DNN AiMC
(KU Leuven + IMEC)
[Ueyoshi et al., ISSCC'22]



TinyVers 22nm, *VLSI'22 (to appear)*
PULPissimo + DNN HWPE
(KU Leuven + Magics)

Take-home message

- A set of templates, SystemVerilog IPs, and helpful tools that can be used to simplify accelerator development & usage
- Main research activities now targeted at easing usage of HWPEs, not only developing new ones

- HWPE home page https://hwpe-doc.readthedocs.io
- Main PULP home page https://pulp-platform.org
- Not only RTL: GVSOC supports HWPEs https://github.com/pulp-platform/pulp-sdk

PULP

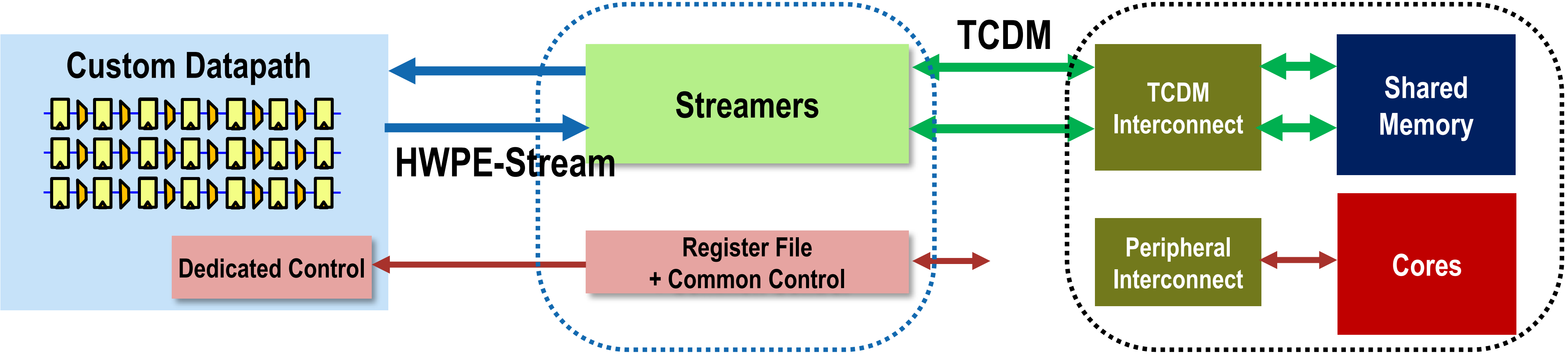
Parallel Ultra Low Power

HWPE Team: Francesco Conti, Yvan Tortorella, Arpan Prasad, Luca Bertaccini, Gianna Paulin, Alessio Burrello, Luka Macan, Luca Benini
And all the PULP team: Luca Benini, Alessandro Capotondi, Alessandro Ottaviano, Alessio Burrello, Alfio Di Mauro, Andrea Borghesi, Andrea Cossettini, Andreas Kurth, Angelo Garofalo, Antonio Pullini, Arpan Prasad, Bjoern Forsberg, Corrado Bonfanti, Cristian Cioflan, Daniele Palossi, Davide Rossi, Fabio Montagna, Florian Glaser, Florian Zaruba, Francesco Conti, Georg Rutishauser, Germain Haugou, Gianna Paulin, Giuseppe Tagliavini, Hanna Müller, Luca Bertaccini, Luca Valente, Luca Colagrande, Manuel Eggimann, Manuele Rusci, Marco Guermandi, Matheus Cavalcante, Matteo Perotti, Matteo Spallanzani, Michael Rogenmoser, Moritz Scherer, Moritz Schneider, Nazareno Bruschi, Nils Wistoff, Pasquale Davide Schiavone, Paul Scheffler, Philipp Mayer, Robert Balas, Samuel Riedel, Sergio Mazzola, Sergei Vostrikov, Simone Benatti, Stefan Mach, Thomas Benz, Thorir Ingolfsson, Tim Fischer, Victor Javier Kartsch Morinigo, Vlad Niculescu, Xiaying Wang, Yichao Zhang, Frank K. Gürkaynak, all our past collaborators and many more that we forgot to mention

http://pulp-platform.org @pulp_platform

BACKUP SLIDES

HWPEs are “first-class citizens” in the cluster



Built with open-source HWPE IPs
https://hwpe-doc.readthedocs.io

Open-source PULP cluster
https://github.com/pulp-platform/pulp

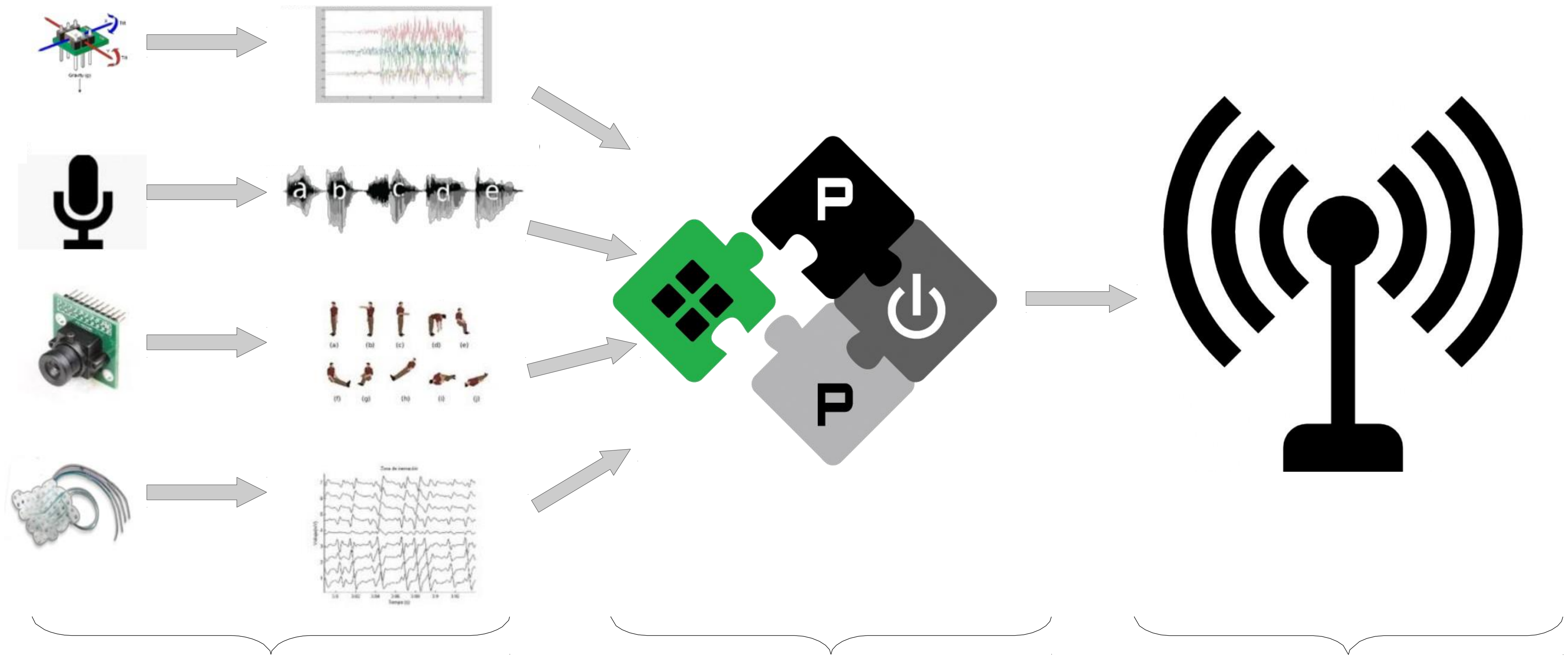
Divide et Impera design approach for HWPEs

Interface designed by composition of open-source IPs for control + memory interfaces through ***HWPE-streams*** → specialized **streamers** convert from streams to mem (TCDM)

Memory access capabilities equivalent or superior to SW (misaligned access, delayed stores, high) on part or all of the memory map

Edge Processing

Sensors Signals Processing Transmission



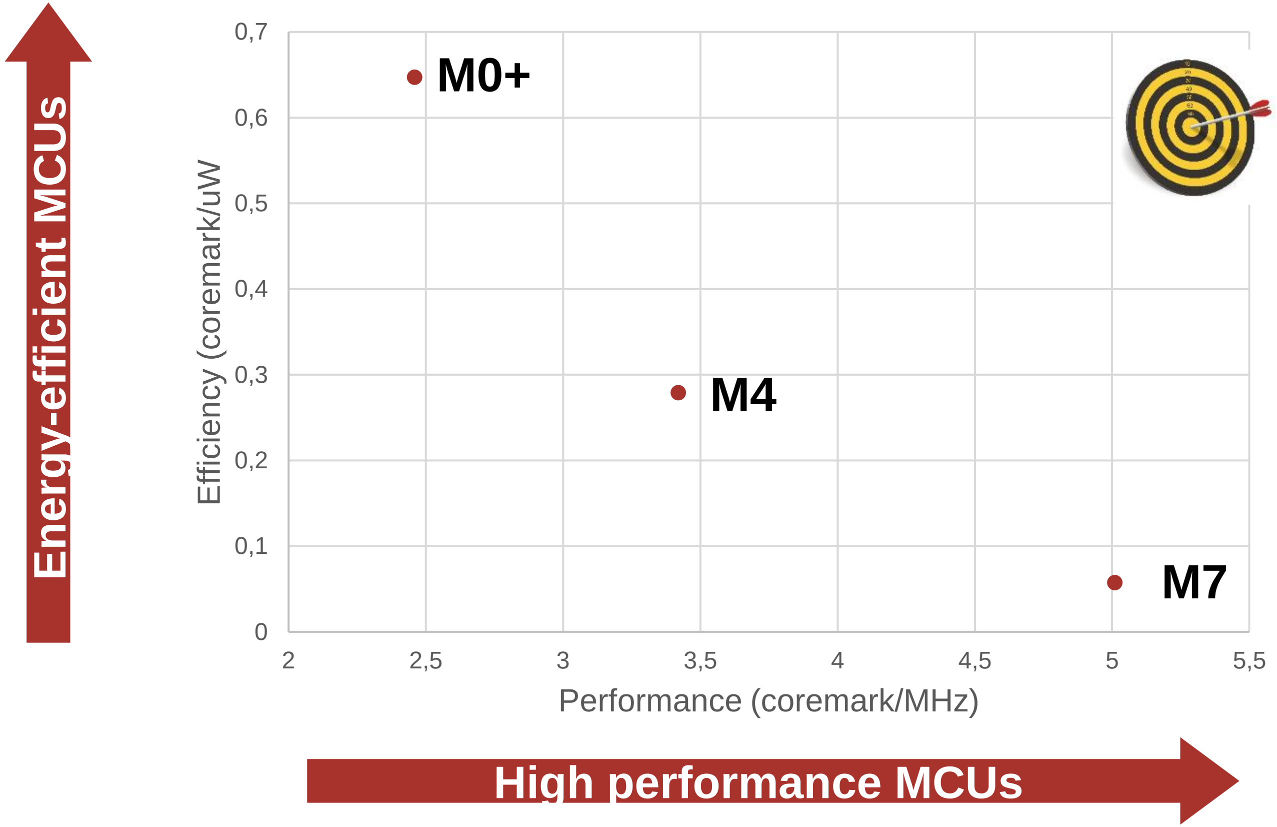
100µW - 1mW

1mW - 10mW

1mW (idle) - 50mW (active)

Energy efficiency @ GOPS is the Challenge

ARM Cortex-M MCUs: M0+, M4, M7 (40LP, typ, 1.1V)*



Energy-efficient MCUs

High performance MCUs

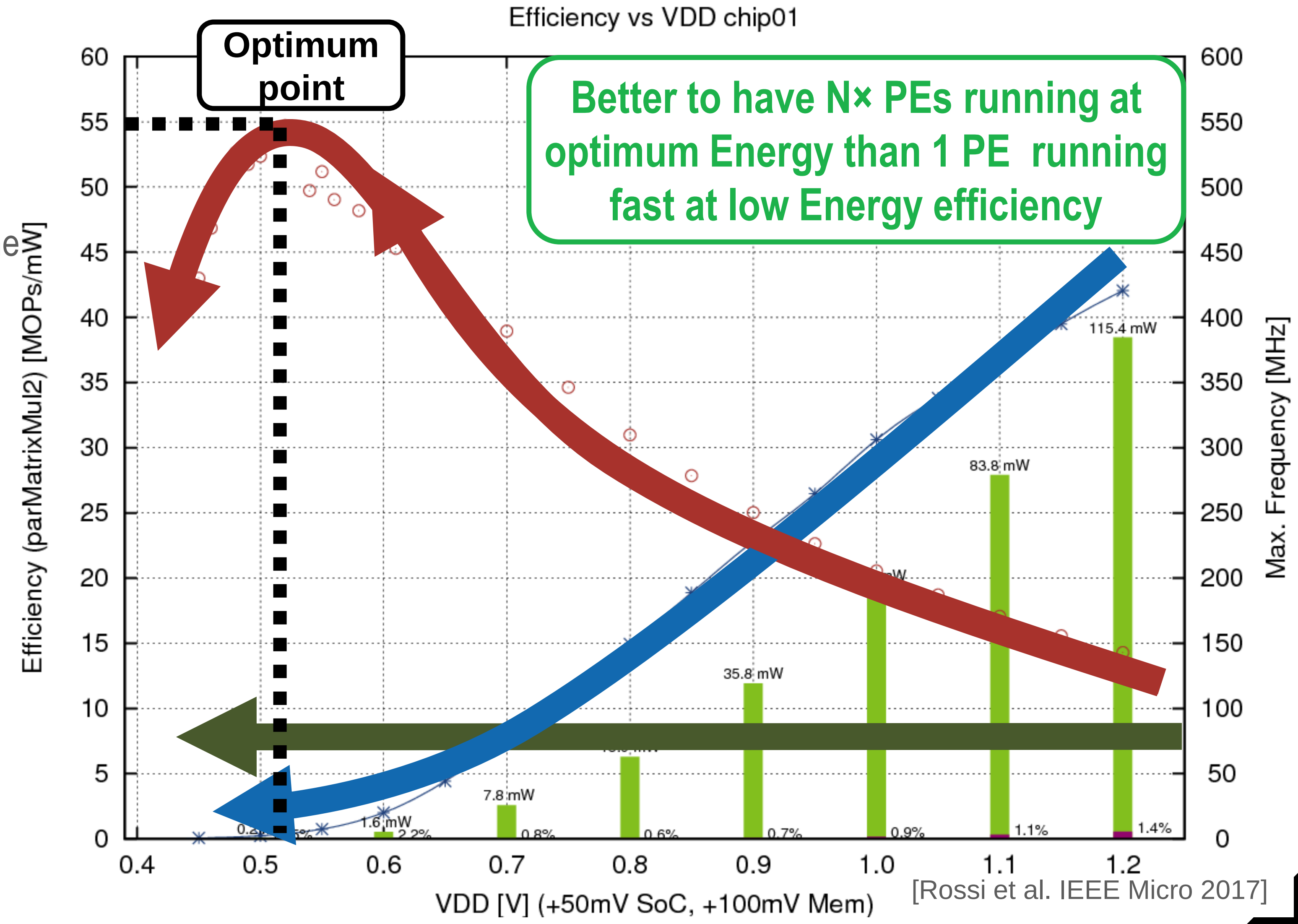
How??

*data from ARMs web

Parallel & Accelerated + Near-threshold

- As VDD decreases, operating speed decreases
- However efficiency increases→ more work done per Joule
- Until leakage effects start to dominate
- Put more units in parallel to get performance up and keep them busy with a parallel workload

ML is massively parallel and scales well
(P/S ↑ with NN size)

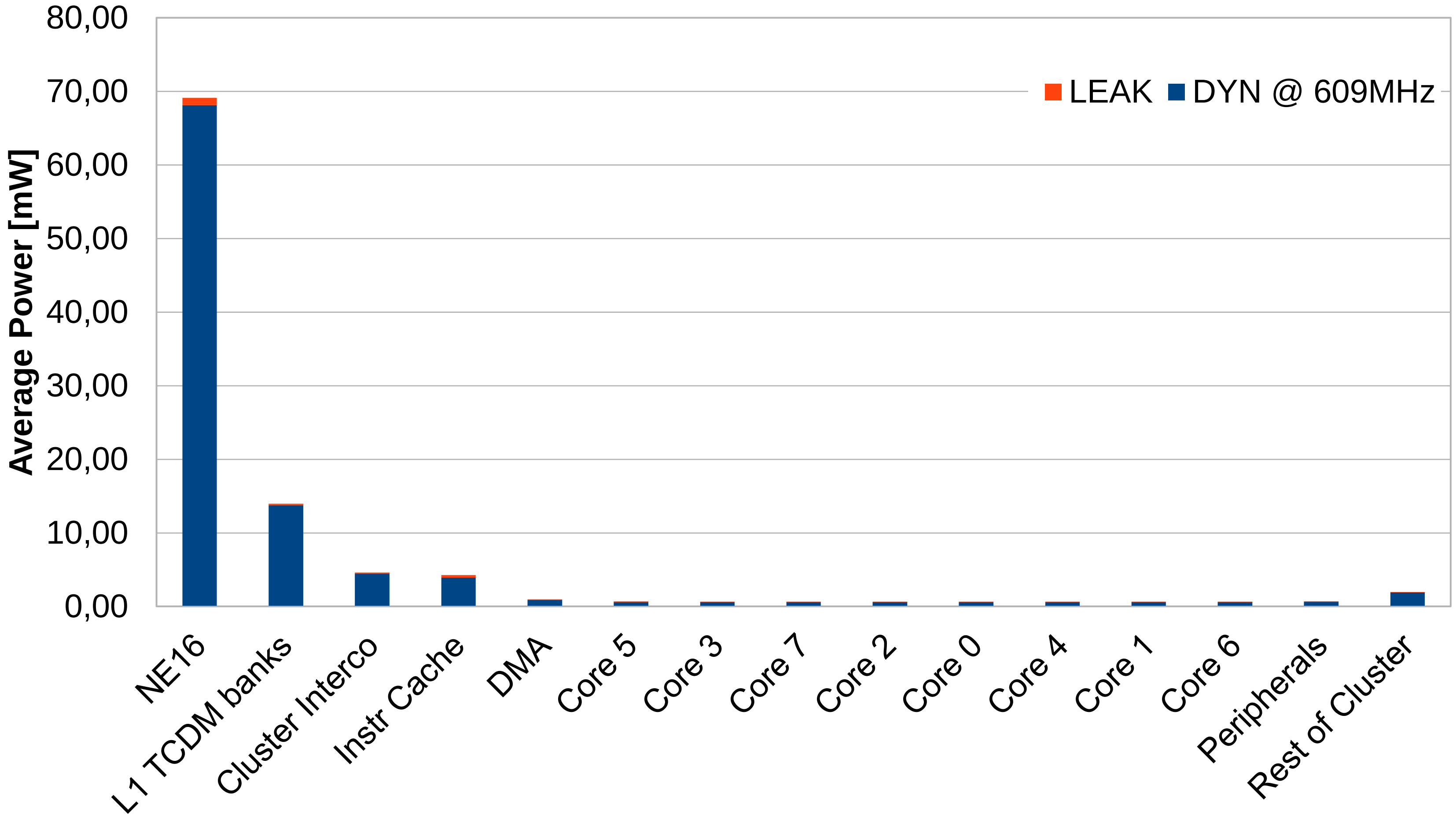


Optimum point

Better to have N× PEs running at optimum Energy than 1 PE running fast at low Energy efficiency

[Rossi et al. IEEE Micro 2017]

NE Power (ICP = 16)



HWPE Control

Memory-mapped, typically connected to Peripheral Interconnect or Peripheral Demux.
Executes a **queue** of **jobs** (typically 2 entries).

Control Registers	*trigger*	starts execution on HWPE, unlocks the ctrl
	acquire	starts a job offload on HWPE,returns a job ID handle, locks the ctrl
	finished jobs	returns no. of completed jobs
	status	each byte returns status of a job in the queue (1=working, 0=idle)
	running job	returns ID of the currently running job
	soft clear	clears the HWPE
	offloader ID	each byte returns ID+1 of a job in the queue
	sw synch	triggers an implementation-specific internal HWPE event (0 to 7 depending on written data)
	Generic Registers	
	Job-Dependent Registers	

Control registers used for standard HWPE control model, common to all HWPEs

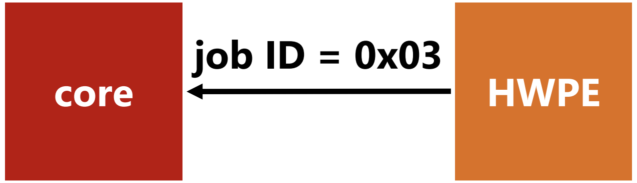
Generic registers and Job-dependent registers used for runtime parameters specific of a HW accelerator. Generics are constant in all jobs

HWPE Control

Memory-mapped, typically connected to Peripheral Interconnect or Peripheral Demux.
Executes a **queue** of **jobs** (typically 2 entries).

Control Registers

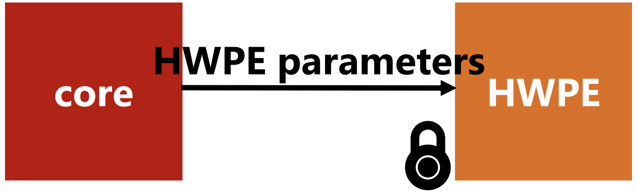
trigger	starts execution on HWPE, unlocks the ctrl
acquire	starts a job offload on HWPE,returns a job ID handle, locks the ctrl
finished jobs	returns no. of completed jobs
status	each byte returns status of a job in the queue (1=working, 0=idle)
running job	returns ID of the currently running job
soft clear	clears the HWPE
offloader ID	each byte returns ID+1 of a job in the queue
sw synch	triggers an implementation-specific internal HWPE event (0 to 7 depending on written data)
Generic Registers	
Job-Dependent Registers	



HWPE Control

Memory-mapped, typically connected to Peripheral Interconnect or Peripheral Demux.
Executes a **queue** of **jobs** (typically 2 entries).

Control Registers	*trigger*	starts execution on HWPE, unlocks the ctrl
	acquire	starts a job offload on HWPE,returns a job ID handle, locks the ctrl
	finished jobs	returns no. of completed jobs
	status	each byte returns status of a job in the queue (1=working, 0=idle)
	running job	returns ID of the currently running job
	soft clear	clears the HWPE
	offloader ID	each byte returns ID+1 of a job in the queue
	sw synch	triggers an implementation-specific internal HWPE event (0 to 7 depending on written data)
Generic Registers		
Job-Dependent Registers		

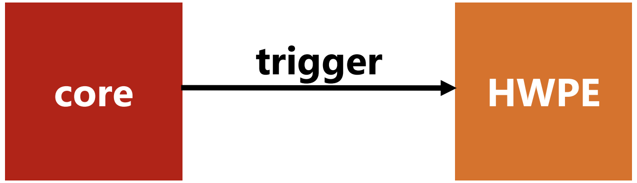


HWPE Control

Memory-mapped, typically connected to Peripheral Interconnect or Peripheral Demux.

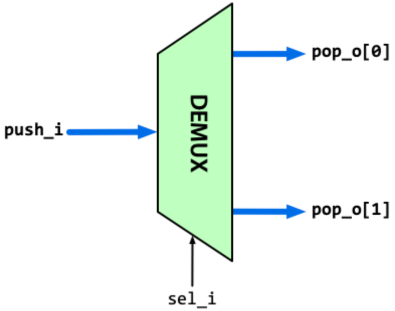
Executes a **queue** of **jobs** (typically 2 entries).

Control Registers	*trigger*	starts execution on HWPE, unlocks the ctrl
	acquire	starts a job offload on HWPE,returns a job ID handle, locks the ctrl
	finished jobs	returns no. of completed jobs
	status	each byte returns status of a job in the queue (1=working, 0=idle)
	running job	returns ID of the currently running job
	soft clear	clears the HWPE
	offloader ID	each byte returns ID+1 of a job in the queue
	sw synch	triggers an implementation-specific internal HWPE event (0 to 7 depending on written data)
	Generic Registers	
	Job-Dependent Registers	

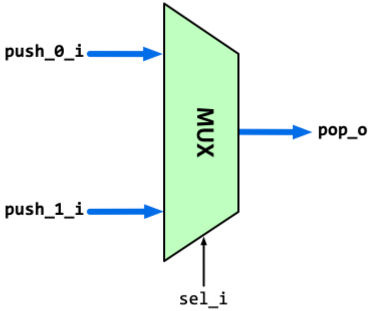


HWPE-Stream: a lightweight protocol for streams

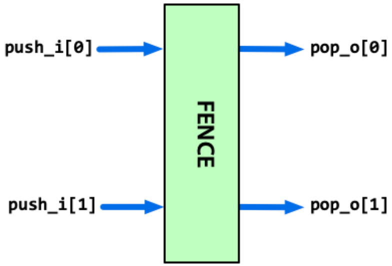
Monodirectional, **minimal** (only handshake + data + optional strobe), no assumptions on content. Positional information carried by *order* of data packets as opposed to *address*.



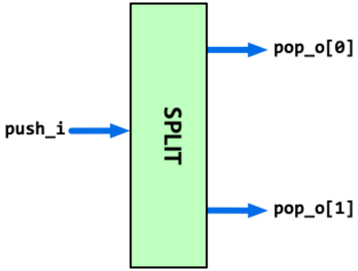
Demultiplexer



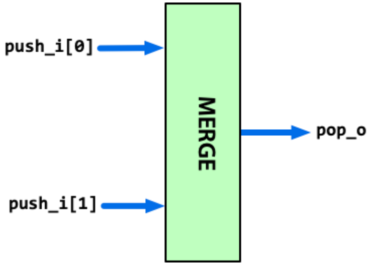
Multiplexer



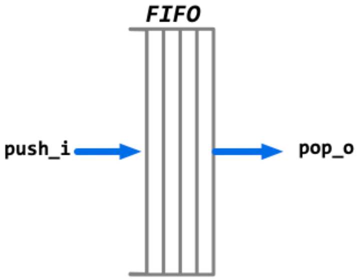
Fence



Splitter

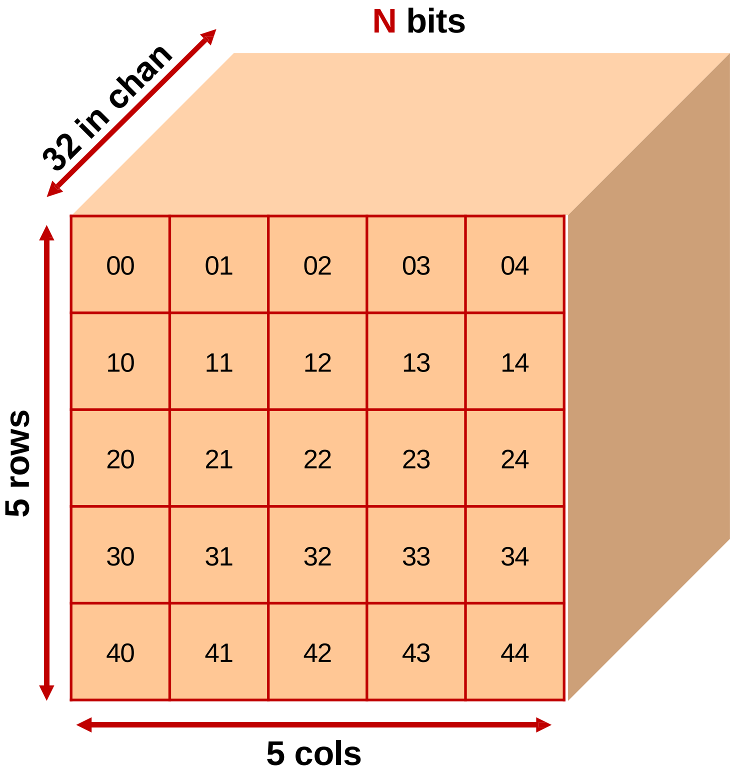


Merger



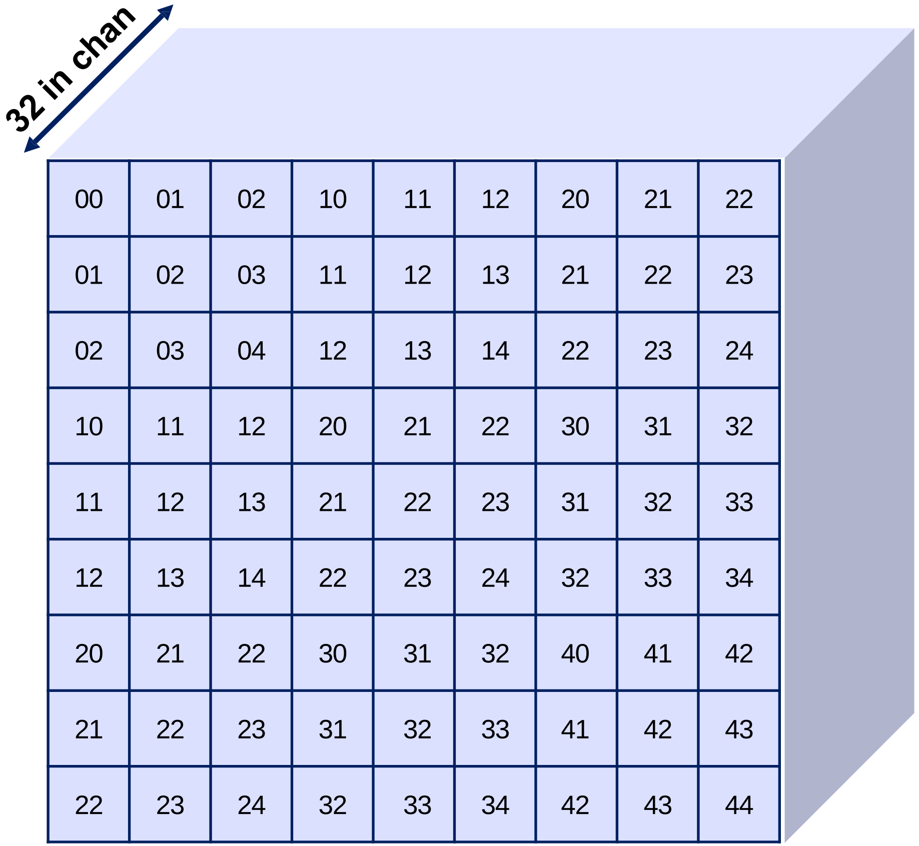
FIFO queues

RBE: 3x3 mapping



00	01	02	03	04
10	11	12	13	14
20	21	22	23	24
30	31	32	33	34
40	41	42	43	44

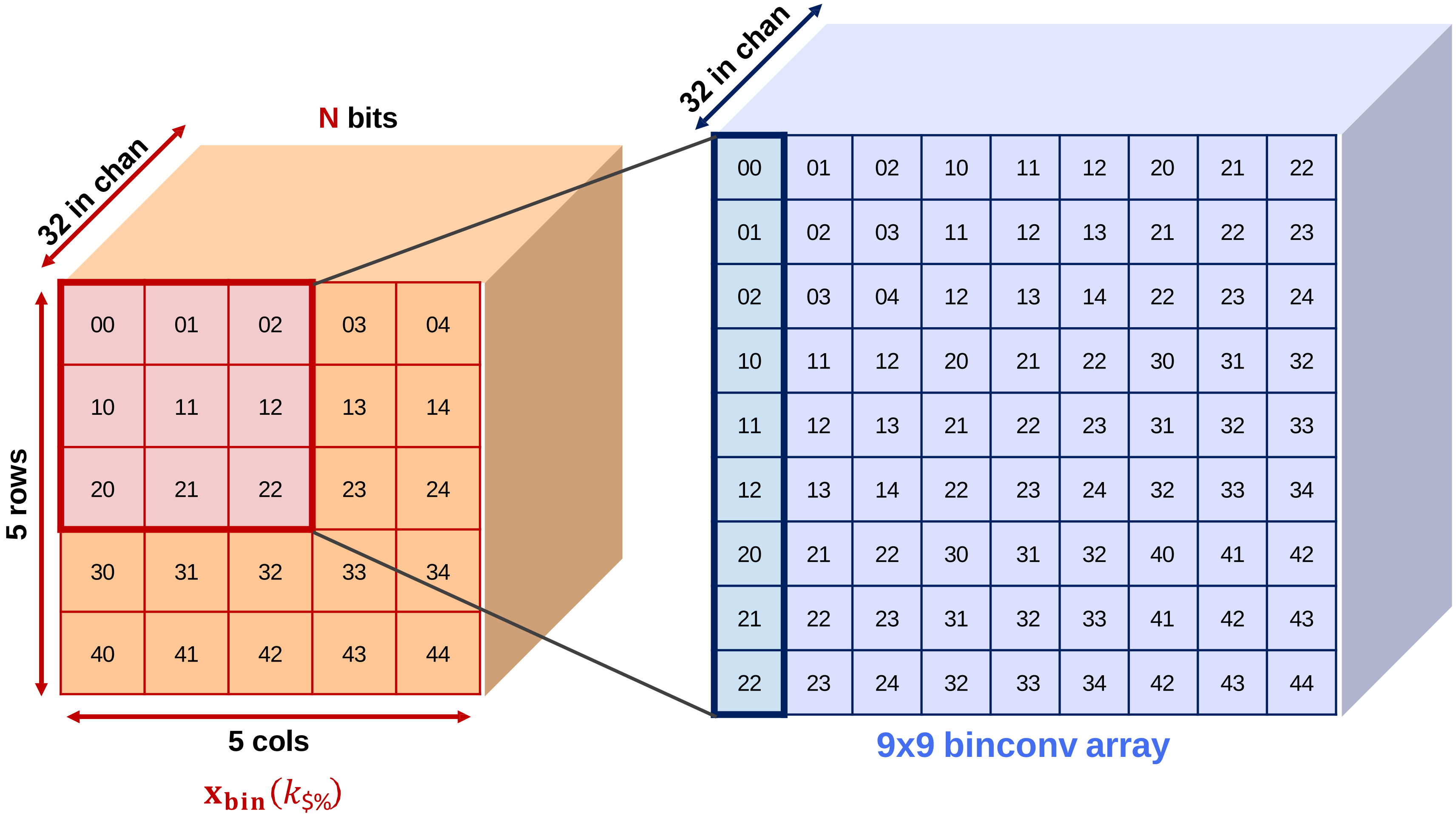
$\mathbf{x}_{\mathbf{bin}}(k_{\$\%})$



00	01	02	10	11	12	20	21	22
01	02	03	11	12	13	21	22	23
02	03	04	12	13	14	22	23	24
10	11	12	20	21	22	30	31	32
11	12	13	21	22	23	31	32	33
12	13	14	22	23	24	32	33	34
20	21	22	30	31	32	40	41	42
21	22	23	31	32	33	41	42	43
22	23	24	32	33	34	42	43	44

9x9 binconv array

RBE: 3x3 mapping

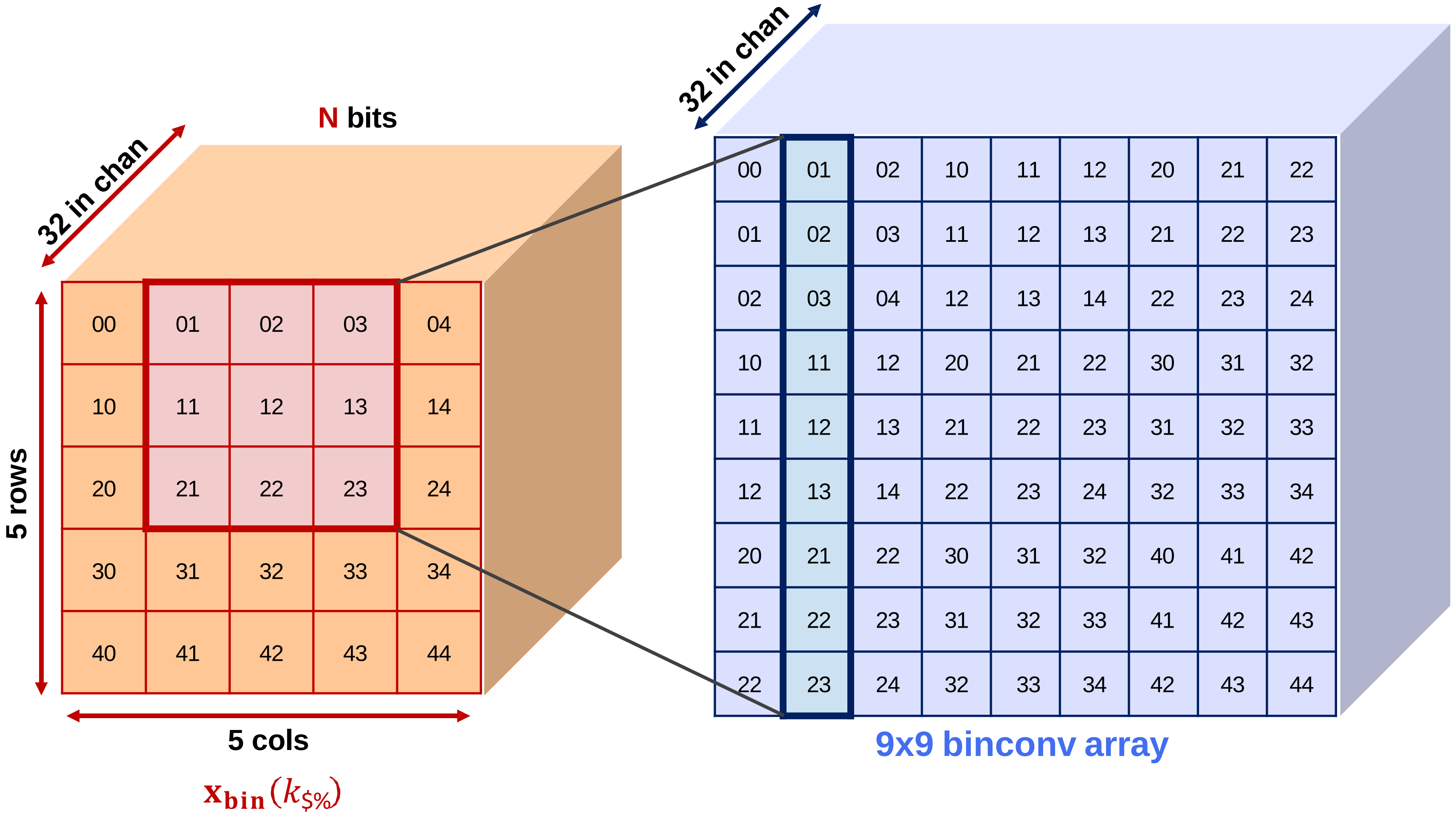


$\mathbf{x}_{\mathbf{bin}}(k_{\$\%})$

00	01	02	03	04
10	11	12	13	14
20	21	22	23	24
30	31	32	33	34
40	41	42	43	44

00	01	02	10	11	12	20	21	22
01	02	03	11	12	13	21	22	23
02	03	04	12	13	14	22	23	24
10	11	12	20	21	22	30	31	32
11	12	13	21	22	23	31	32	33
12	13	14	22	23	24	32	33	34
20	21	22	30	31	32	40	41	42
21	22	23	31	32	33	41	42	43
22	23	24	32	33	34	42	43	44

RBE: 3x3 mapping

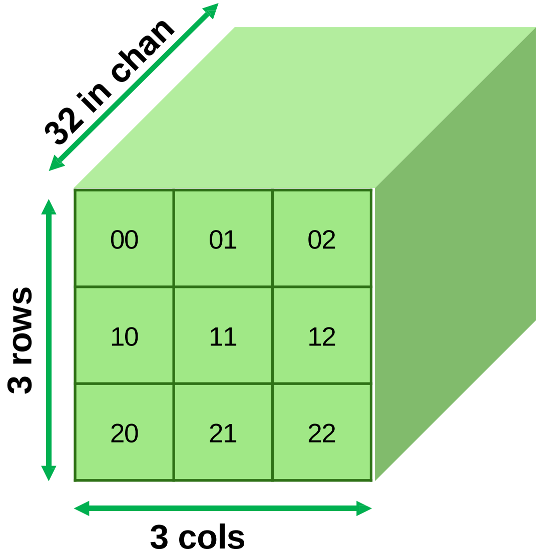


00	01	02	03	04
10	11	12	13	14
20	21	22	23	24
30	31	32	33	34
40	41	42	43	44

$\mathbf{x}_{\mathbf{bin}}(k_{\$\%})$

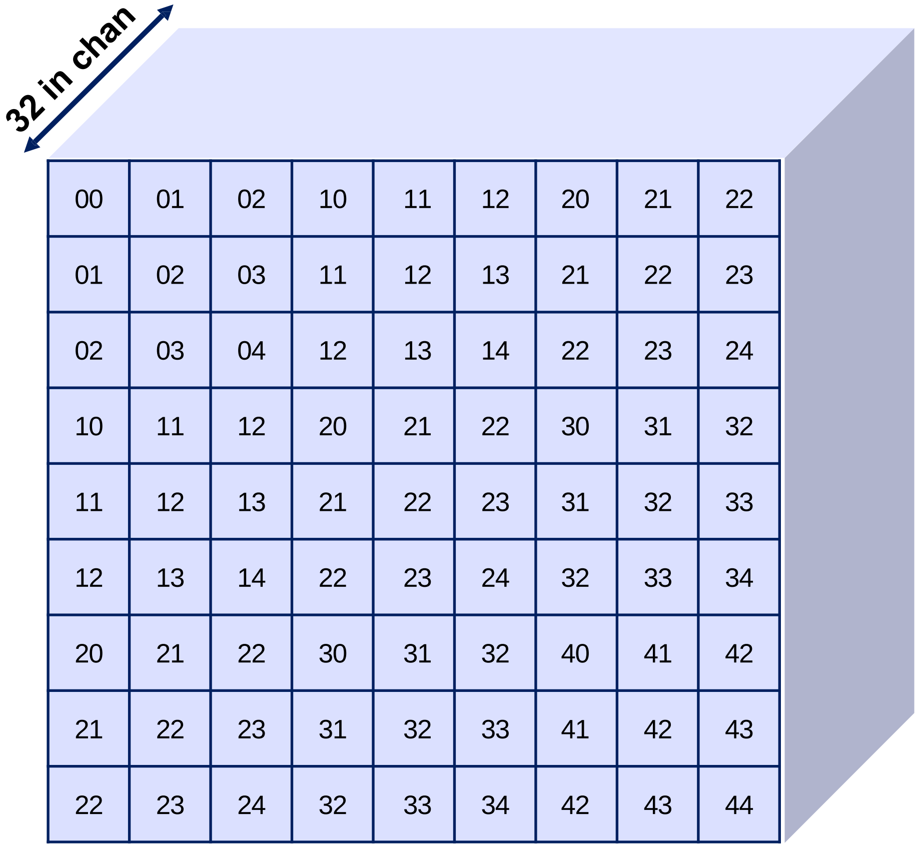
00	01	02	10	11	12	20	21	22
01	02	03	11	12	13	21	22	23
02	03	04	12	13	14	22	23	24
10	11	12	20	21	22	30	31	32
11	12	13	21	22	23	31	32	33
12	13	14	22	23	24	32	33	34
20	21	22	30	31	32	40	41	42
21	22	23	31	32	33	41	42	43
22	23	24	32	33	34	42	43	44

RBE: 3x3 mapping



00	01	02
10	11	12
20	21	22

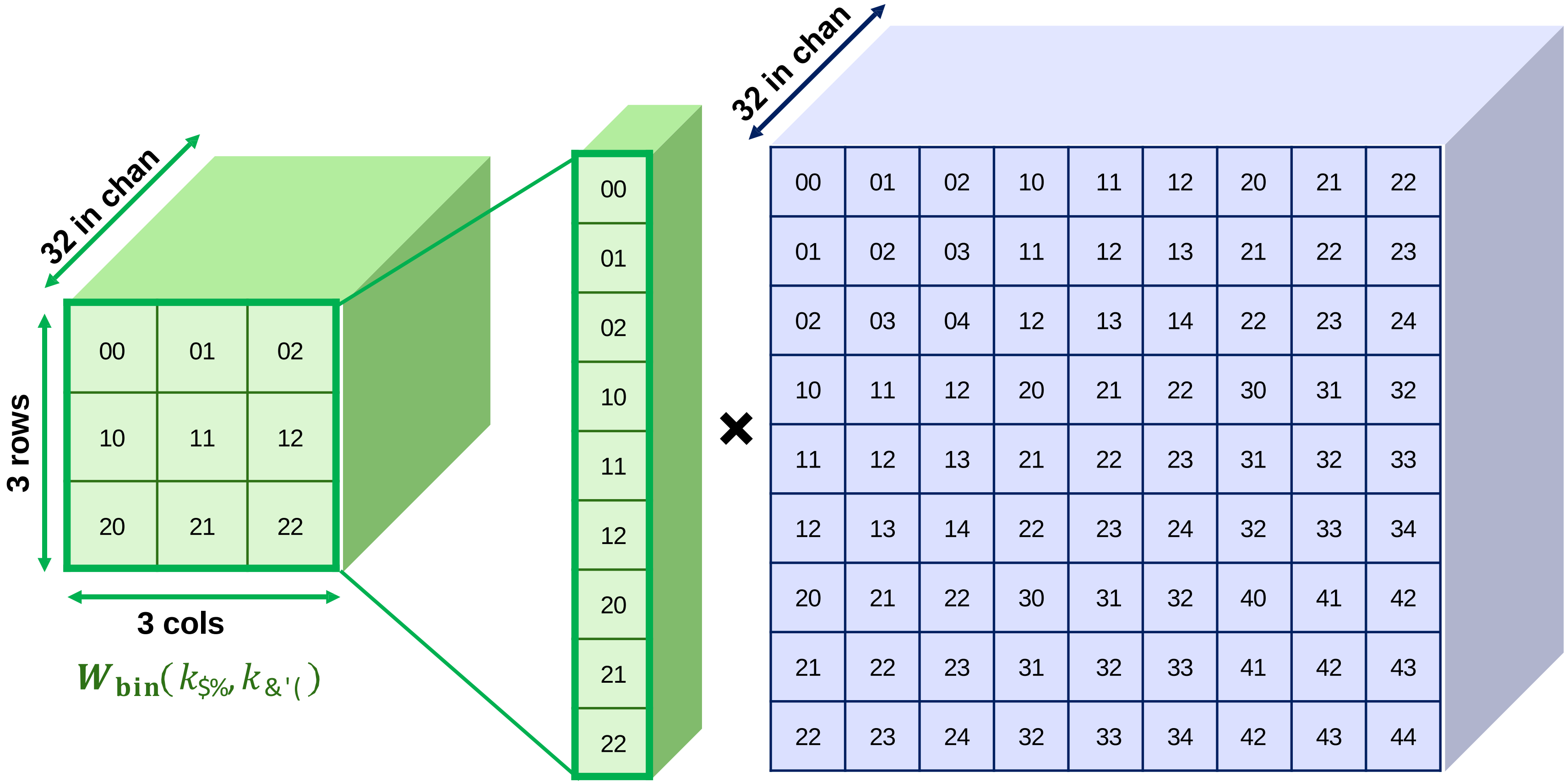
$W_{\text{bin}}(k_{\$\%}, k_{\&'(})$



00	01	02	10	11	12	20	21	22
01	02	03	11	12	13	21	22	23
02	03	04	12	13	14	22	23	24
10	11	12	20	21	22	30	31	32
11	12	13	21	22	23	31	32	33
12	13	14	22	23	24	32	33	34
20	21	22	30	31	32	40	41	42
21	22	23	31	32	33	41	42	43
22	23	24	32	33	34	42	43	44

9x9 binconv array

RBE: 3x3 mapping



00	01	02
10	11	12
20	21	22

$W_{\text{bin}}(k_{\$\%}, k_{\&'(})$

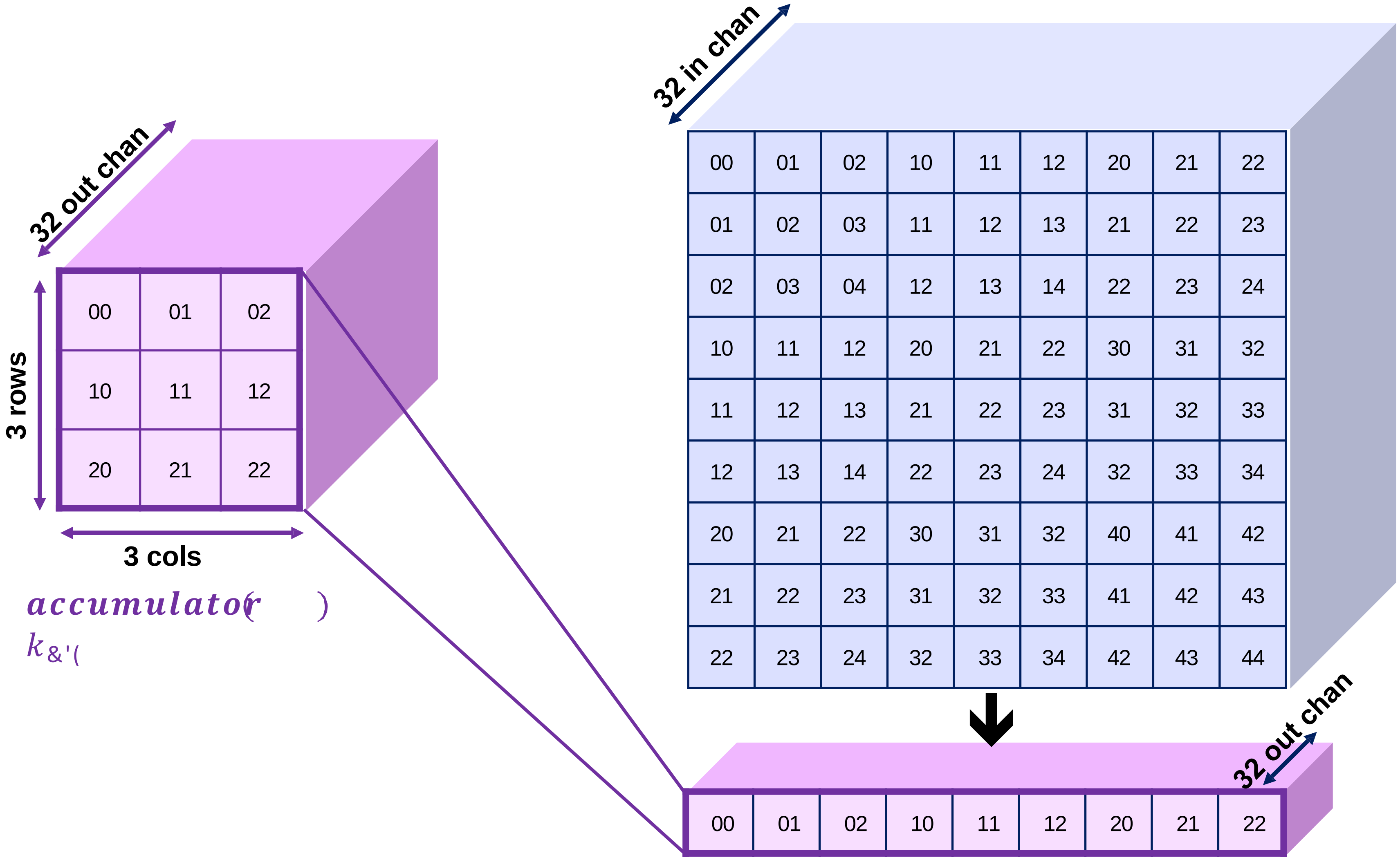
00
01
02
10
11
12
20
21
22

00	01	02	10	11	12	20	21	22
01	02	03	11	12	13	21	22	23
02	03	04	12	13	14	22	23	24
10	11	12	20	21	22	30	31	32
11	12	13	21	22	23	31	32	33
12	13	14	22	23	24	32	33	34
20	21	22	30	31	32	40	41	42
21	22	23	31	32	33	41	42	43
22	23	24	32	33	34	42	43	44

9x9 binconv array

32 out chan × M bits in 32×M cycles

RBE: 3x3 mapping



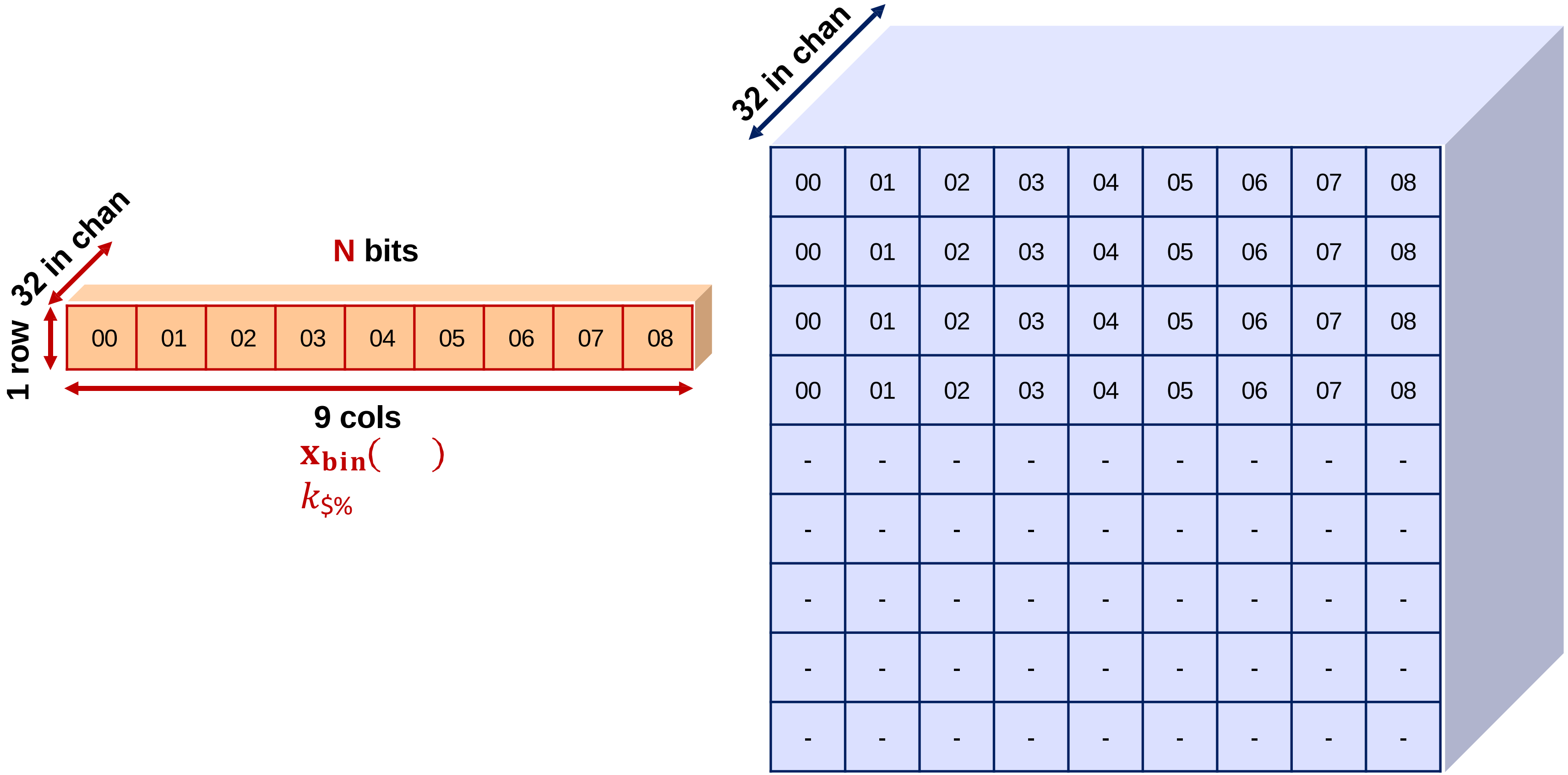
00	01	02
10	11	12
20	21	22

accumulator( )
k&'(

00	01	02	10	11	12	20	21	22
01	02	03	11	12	13	21	22	23
02	03	04	12	13	14	22	23	24
10	11	12	20	21	22	30	31	32
11	12	13	21	22	23	31	32	33
12	13	14	22	23	24	32	33	34
20	21	22	30	31	32	40	41	42
21	22	23	31	32	33	41	42	43
22	23	24	32	33	34	42	43	44

00	01	02	10	11	12	20	21	22

RBE: 1x1 mapping



00	01	02	03	04	05	06	07	08

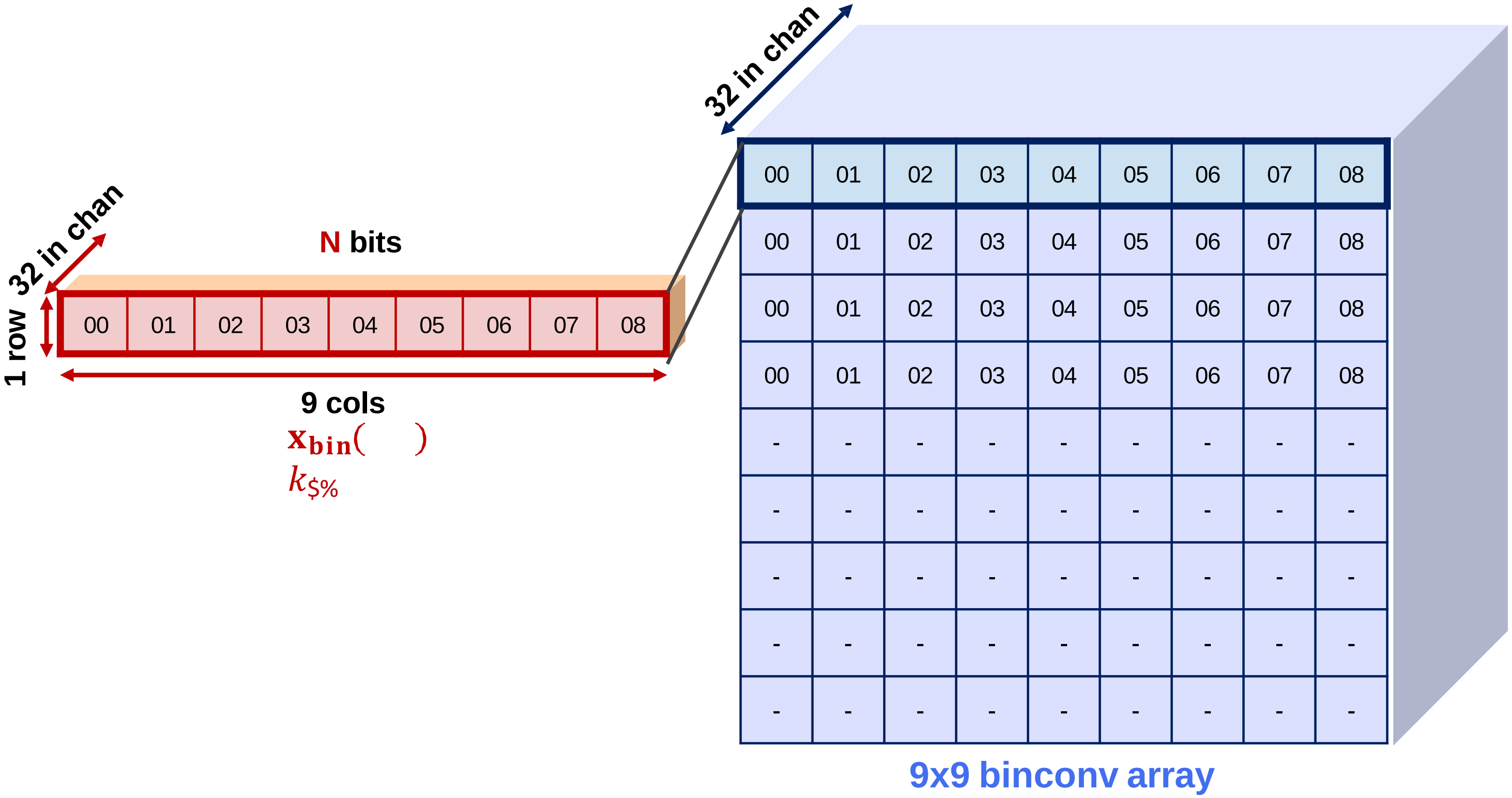
$\mathbf{x}_{\mathbf{bin}}(\quad)$

$k_{\$\%}$

00	01	02	03	04	05	06	07	08
00	01	02	03	04	05	06	07	08
00	01	02	03	04	05	06	07	08
00	01	02	03	04	05	06	07	08
-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-

9x9 binconv array

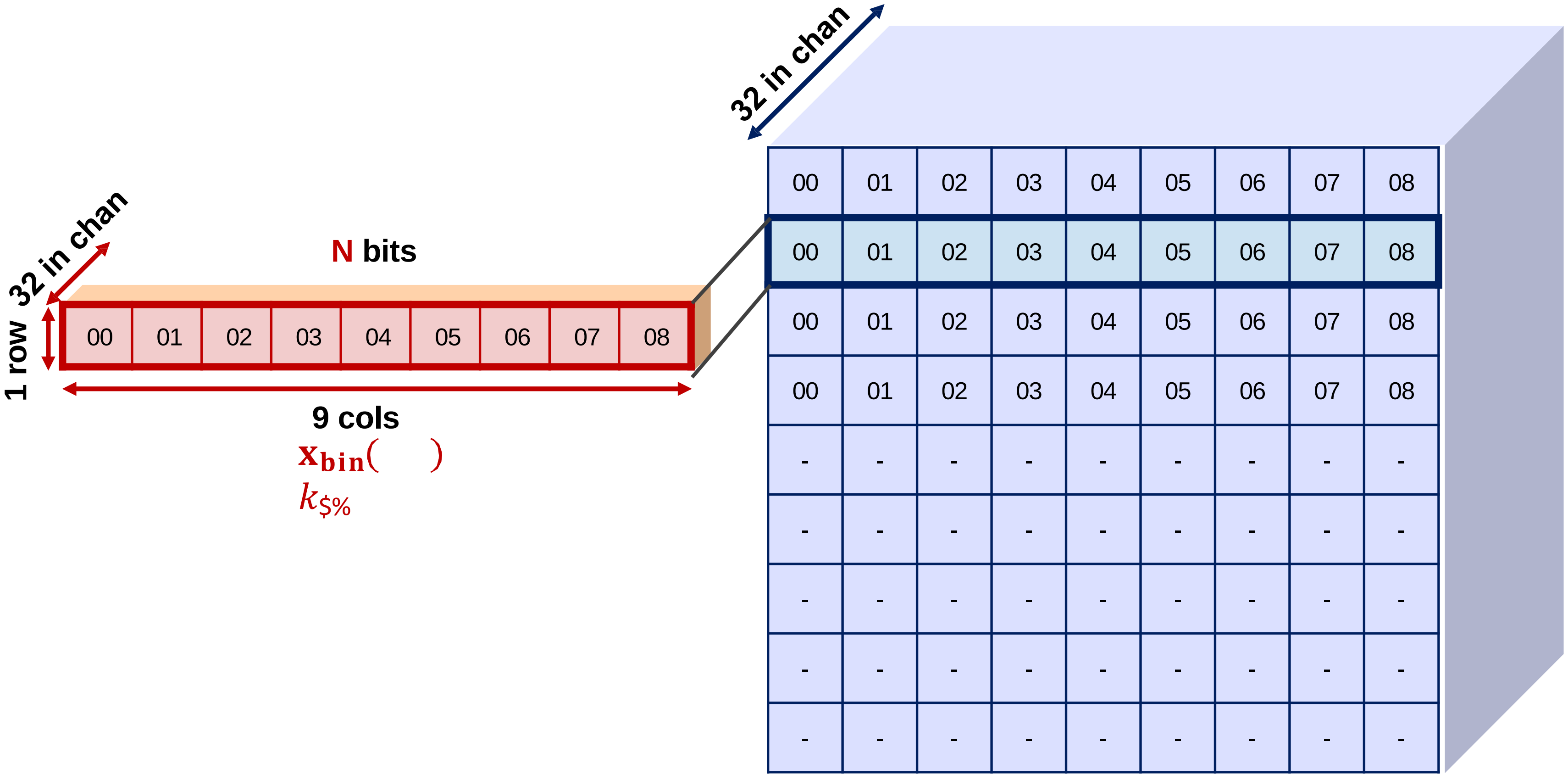
RBE: 1x1 mapping



00	01	02	03	04	05	06	07	08

00	01	02	03	04	05	06	07	08
00	01	02	03	04	05	06	07	08
00	01	02	03	04	05	06	07	08
00	01	02	03	04	05	06	07	08
-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-

RBE: 1x1 mapping



00	01	02	03	04	05	06	07	08

9 cols

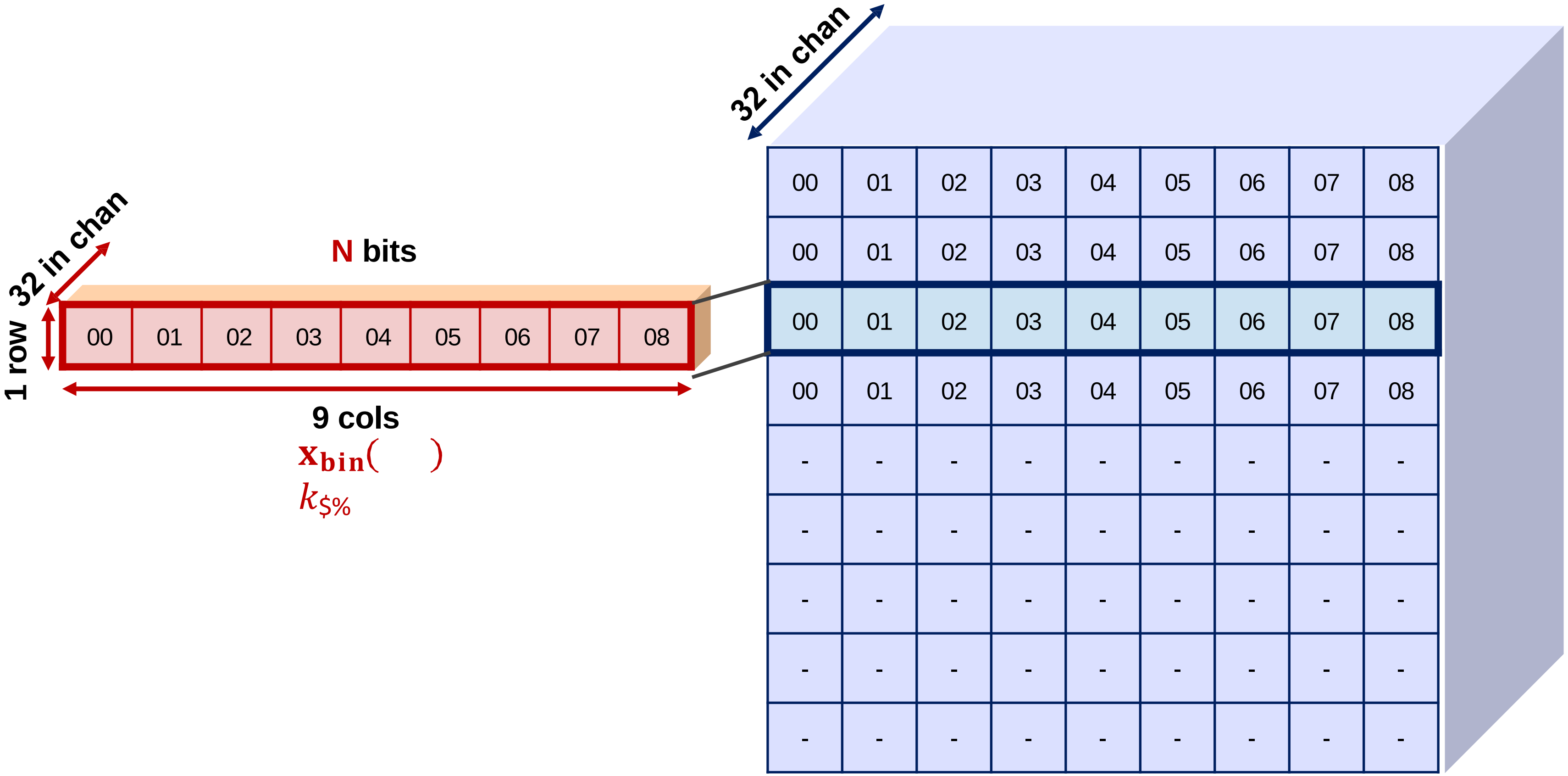
$\mathbf{x}_{\mathbf{bin}}(\quad)$

$k_{\$\%}$

00	01	02	03	04	05	06	07	08
00	01	02	03	04	05	06	07	08
00	01	02	03	04	05	06	07	08
00	01	02	03	04	05	06	07	08
-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-

9x9 binconv array

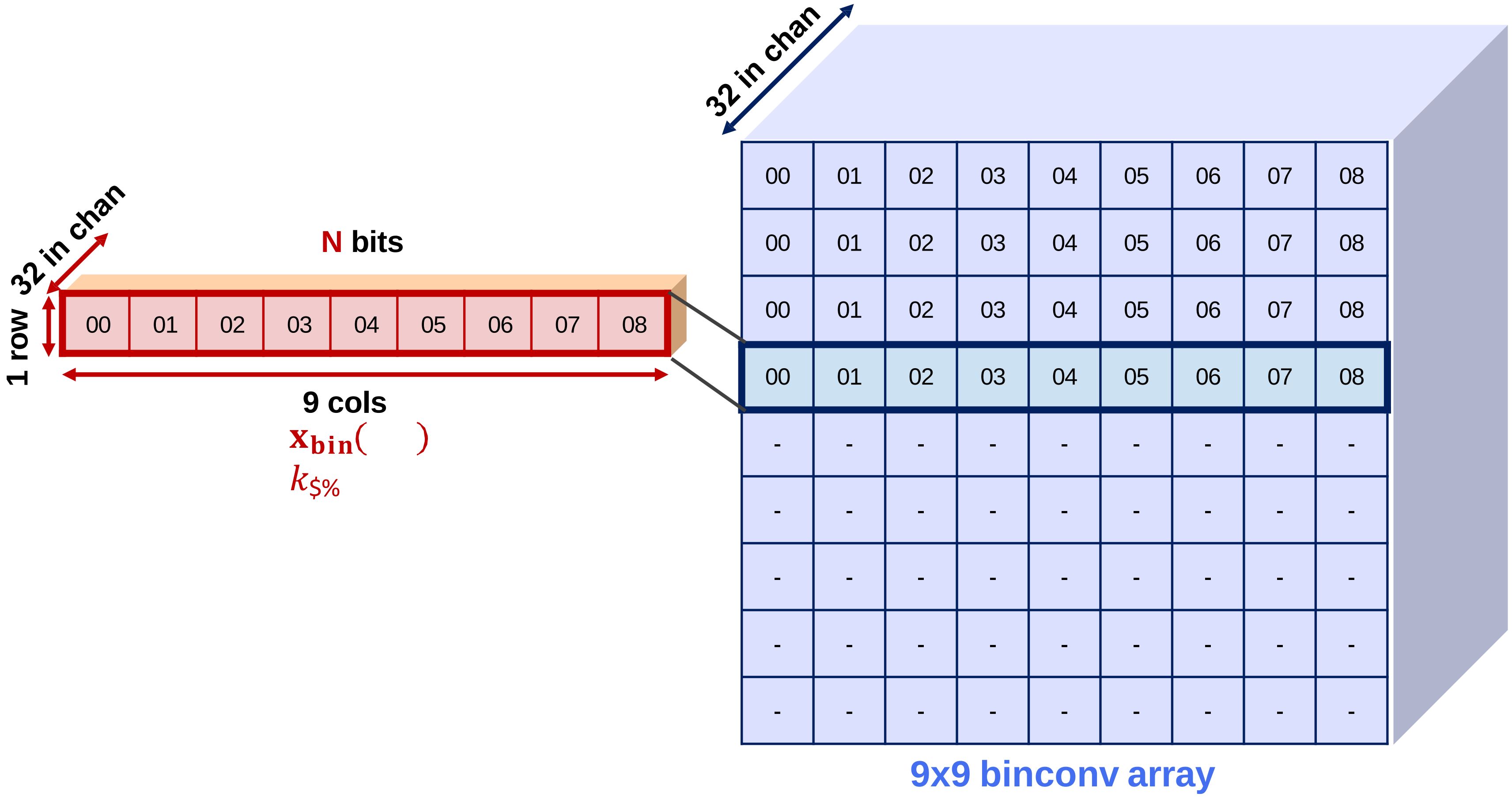
RBE: 1x1 mapping



00	01	02	03	04	05	06	07	08
00	01	02	03	04	05	06	07	08
00	01	02	03	04	05	06	07	08
00	01	02	03	04	05	06	07	08
-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-

9x9 binconv array

RBE: 1x1 mapping



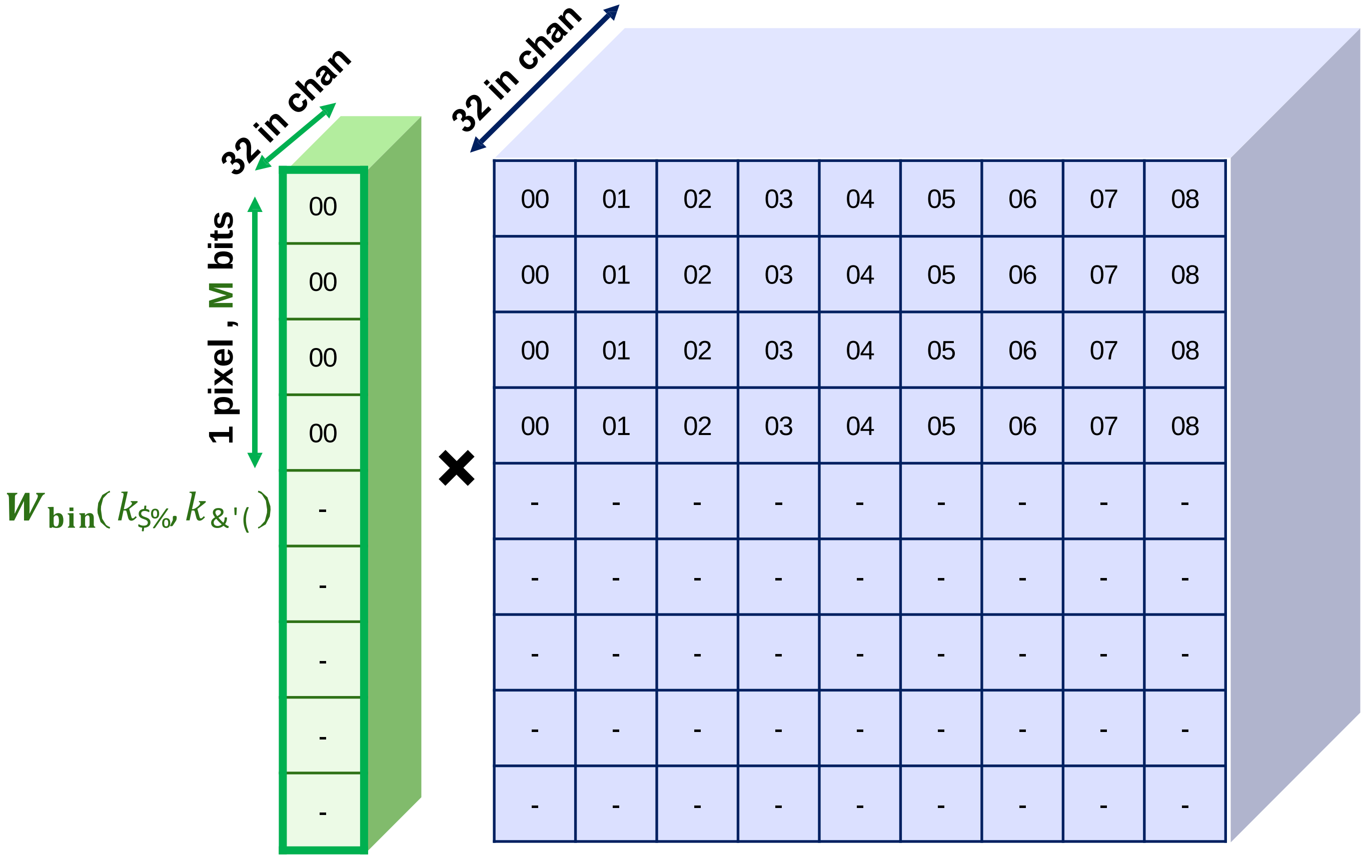
00	01	02	03	04	05	06	07	08

$\mathbf{x}_{\mathbf{bin}}(\quad)$

$k_{\$\%}$

00	01	02	03	04	05	06	07	08
00	01	02	03	04	05	06	07	08
00	01	02	03	04	05	06	07	08
00	01	02	03	04	05	06	07	08
-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-

RBE: 1x1 mapping



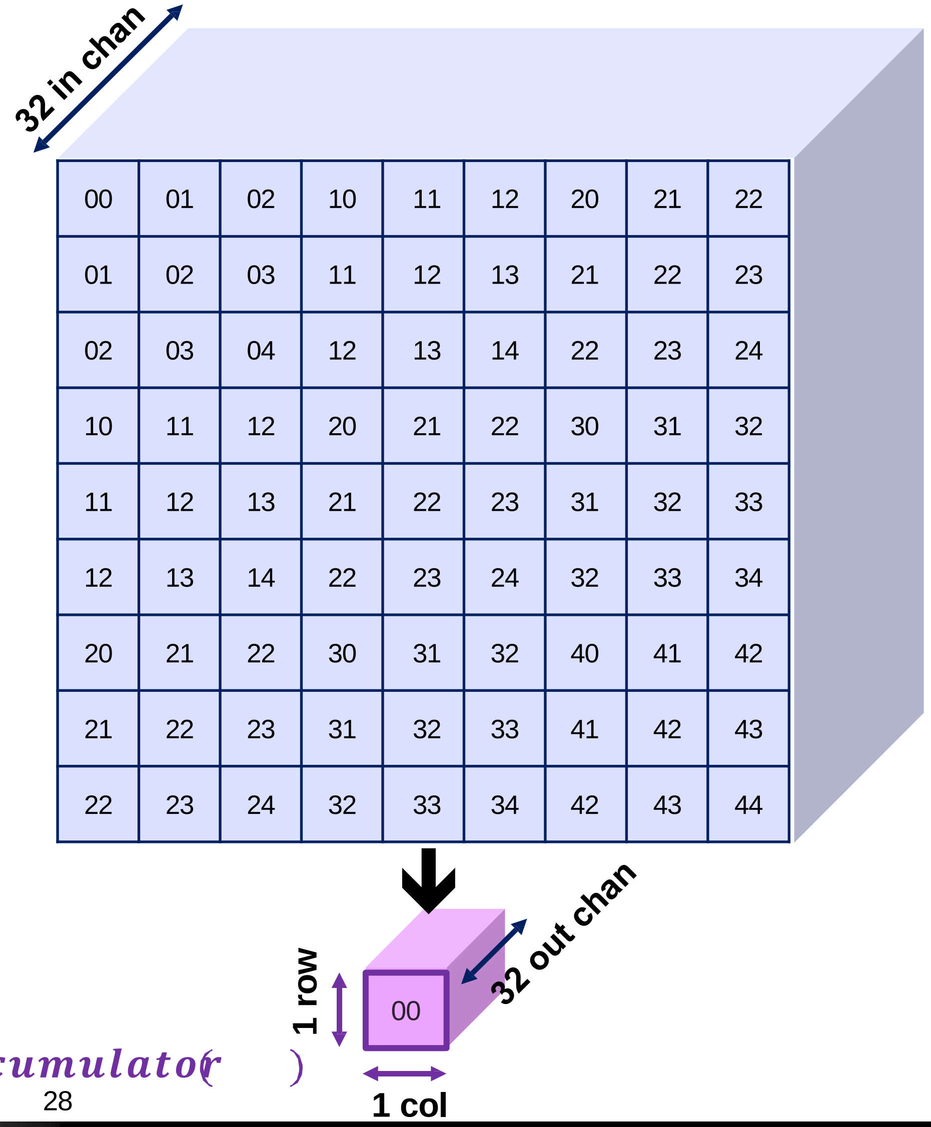
00
00
00
00
-
-
-
-
-

00	01	02	03	04	05	06	07	08
00	01	02	03	04	05	06	07	08
00	01	02	03	04	05	06	07	08
00	01	02	03	04	05	06	07	08
-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-

9x9 binconv array

32 out chan in 32 cycles

RBE: 1x1 mapping



32 in chan

00	01	02	10	11	12	20	21	22
01	02	03	11	12	13	21	22	23
02	03	04	12	13	14	22	23	24
10	11	12	20	21	22	30	31	32
11	12	13	21	22	23	31	32	33
12	13	14	22	23	24	32	33	34
20	21	22	30	31	32	40	41	42
21	22	23	31	32	33	41	42	43
22	23	24	32	33	34	42	43	44

1 row

00

32 out chan

1 col

$accumulator($ $)$

$k_{\&'(}$ 28