PULP and AI Acceleration

Taichip Winter School – Frankfurt a.d. Oder – February 2025

Frank K. Gürkaynak kgf@iis.ee.ethz.ch

PULP Platform
Open Source Hardware, the way it should be!

@pulp_platform
pulp-platform.org
youtube.com/pulp_platform

What is on the menu today and who is cooking?

- **Frank K. Gürkaynak, ETH Zürich**
  - Senior scientist in the group of Luca Benini
  - Director of Microelectronics Design Center (dz.ethz.ch)
- **Open source and IC Design**
  - Active in the open source HW community
  - Community representative on the board of directors of RISC-V
  - Involved in IC Design since 1995
    - Energy efficient processor design
    - Cryptographic Hardware accelerators
- **Contact**
  - e-mail: kgf@iis.ee.ethz.ch
  - Homepage: https://iis.ee.ethz.ch/~kgf

What is on the menu today

- **Brief introduction into how computer architectures evolved**
  - How we have addressed the need for ever more computing, what are our issues
- **What are the characteristics of (present) AI/ML algorithms**
  - Why throwing more and more cores are not helping much
- **What is the PULP team doing about it**
  - Efficient cores (SIMD, Quantization)
  - Shared memory accelerators (PULP cluster)
  - Scaling to 100s and 1000s of cores (Mempool, Occamy, FlooNoC)
  - Vector processing (Ara, Spatz)
  - Heterogeneous acceleration (Kraken, ITA)

1970s

1975s

1980s

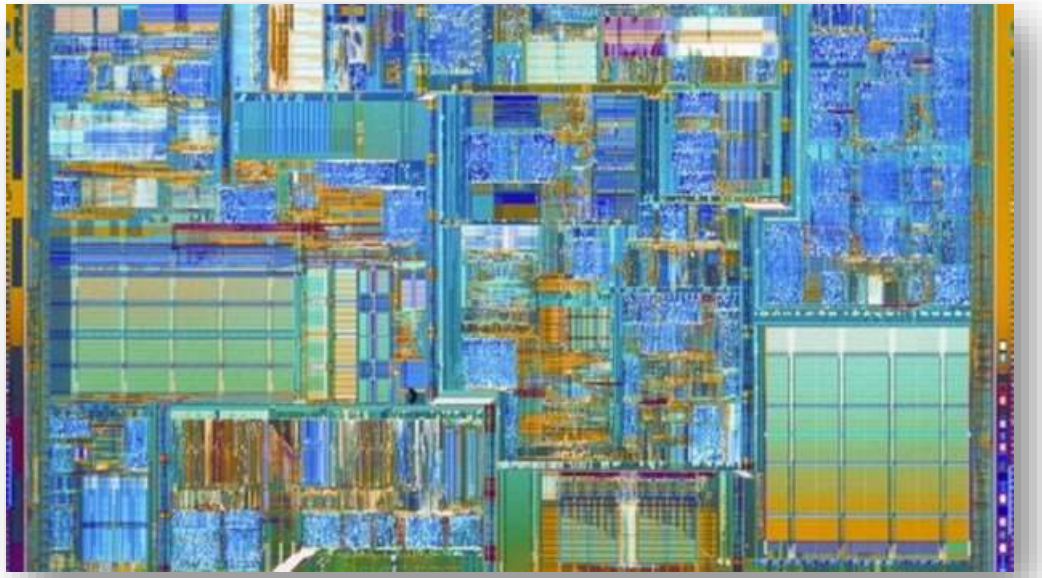
1990s

2000s

2010s

2020s

Emergence of Microprocessors



These slides “borrowed” from Viviane Potocnik

Intel 4004: First commercially available microprocessor, operating at **740 kHz**.

1970s

1975s

1980s

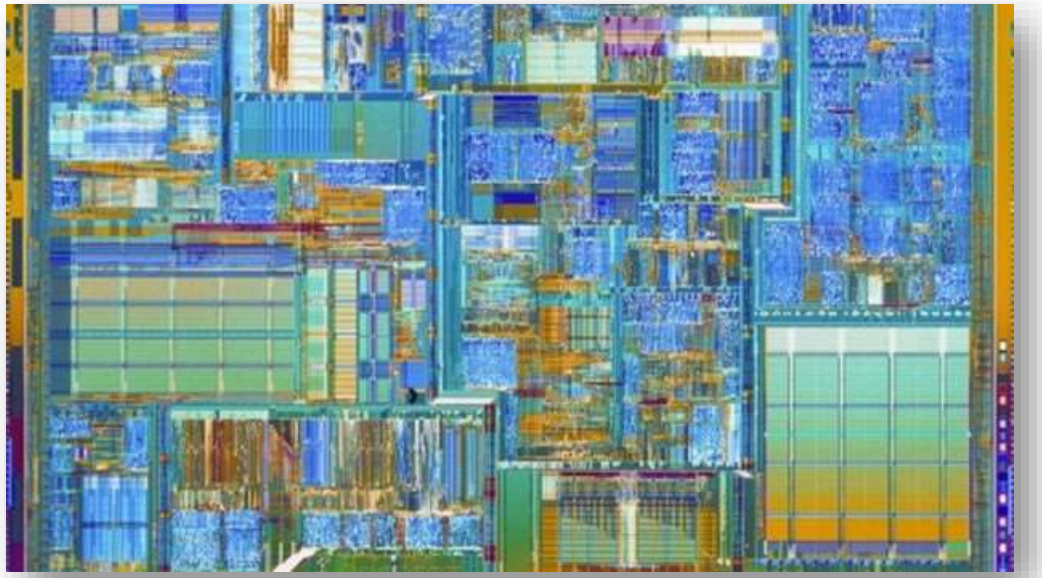
1990s

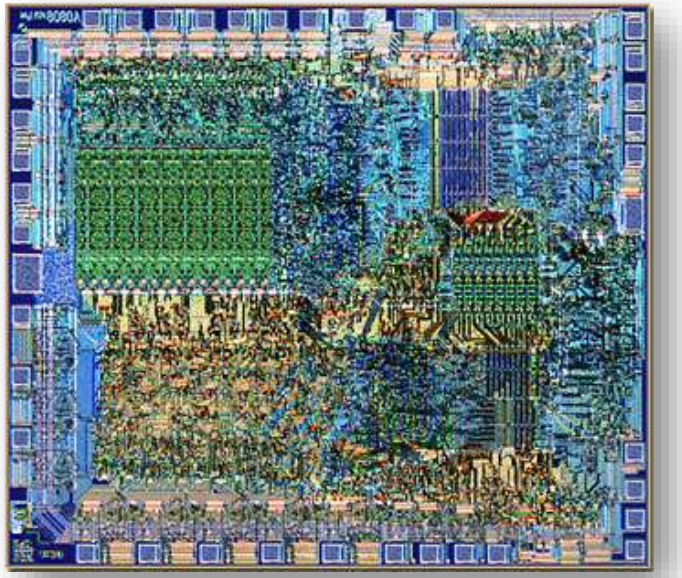
2000s

2010s

2020s

Emergence of Microprocessors



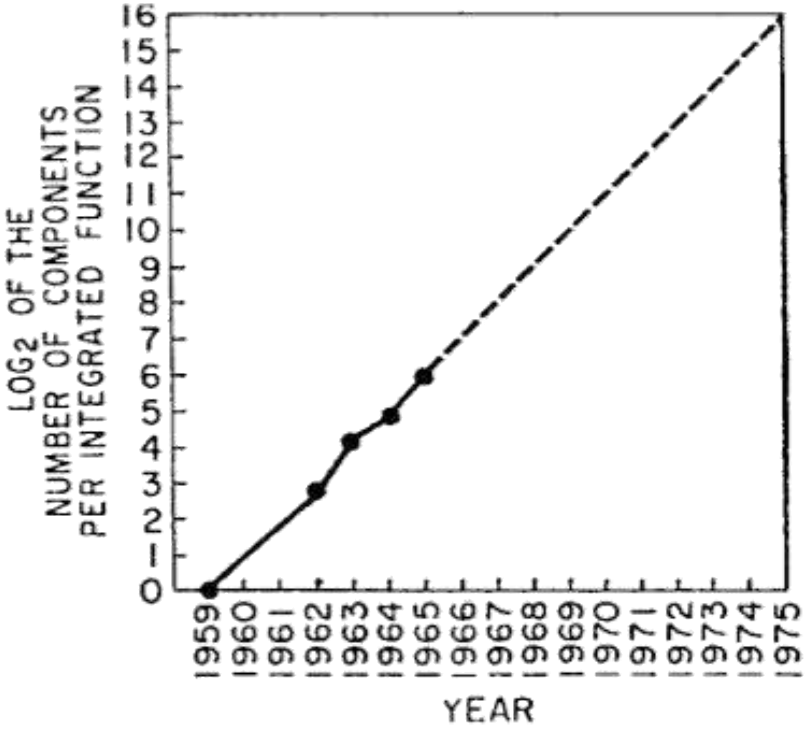


Intel 4004: First commercially available microprocessor, operating at **740 kHz**.

Intel 8080/8086: More powerful and set foundation for the **x86** architecture.

1970s
1975s
1980s
1990s
2000s
2010s
2020s

Moore's Law (Early Phase)

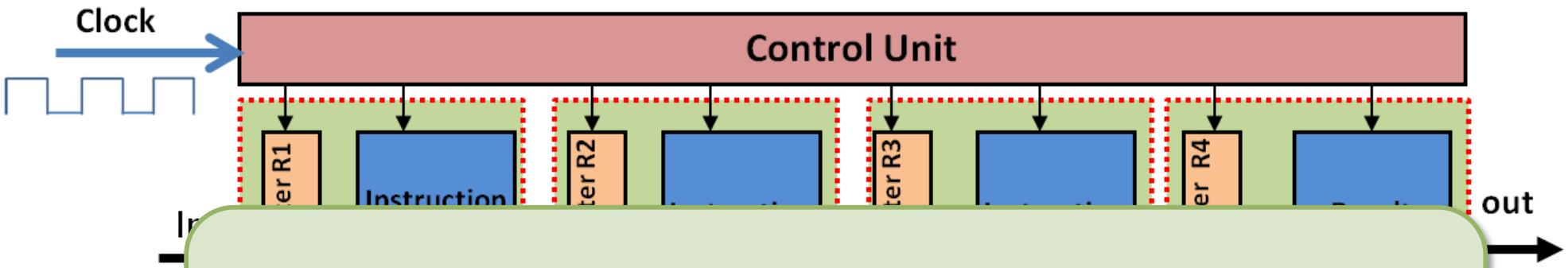


Transistor density **doubled** roughly **every 1.5–2 years**, enabling more complex yet still rudimentary chips.

Lower cost + higher integration: allowed CPUs to move from large mainframes to personal computers.

1970s
1975s
1980s
1990s
2000s
2010s
2020s

Basic Pipelining



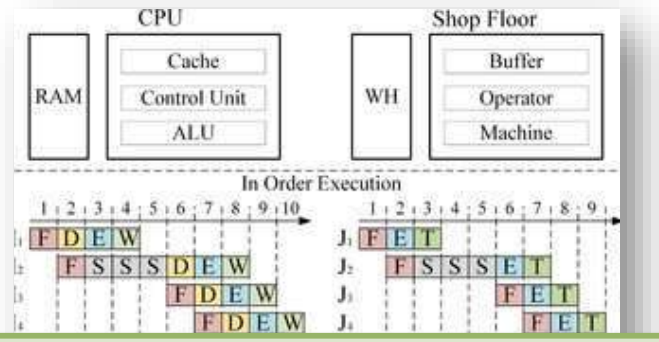
These systems are **too limited** for **large-scale data processing**. However, the basic idea of packing **more transistors** to improve performance lays groundwork for future, **more parallel designs**.

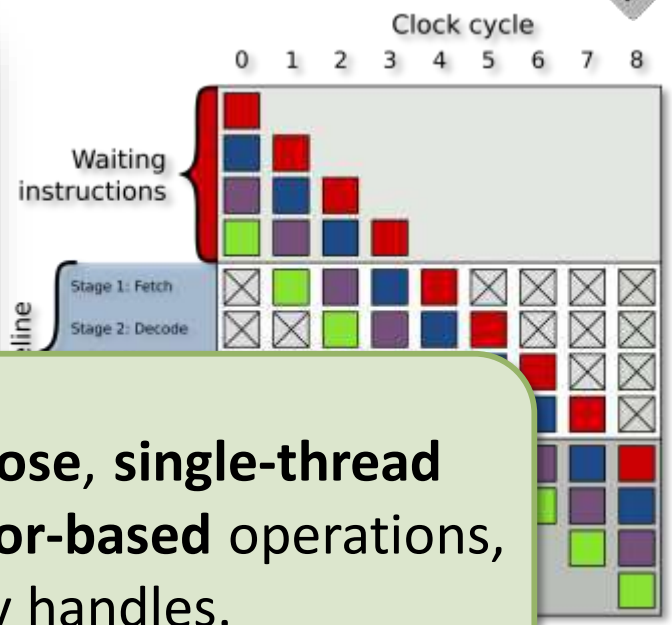
Some early CPUs started using simple pipelines (**fetch**, **decode**, **execute**).

Modest gains but established the principle of **overlapping instruction stages**.

1970s
1975s
1980s
1990s
2000s
2010s
2020s

Instruction Level Parallelism





ILP boosts performance **for general-purpose**, **single-thread workloads**. **ML** requires more **parallel**, **vector-based** operations, beyond what ILP alone efficiently handles.

Superscalar Approach

CPUs could dispatch multiple insns/cycle

Significant increase in single-threaded performance

Out-Of-Order Execution

HW scheduling of instructions to bypass stalls

Register renaming and large reorder buffers keeps pipeline full

Speculative Execution

Guessing the outcome of branches to minimize idle cycles

Increasing HW complexity (branch predictor, caches, …)

1970s

1975s

1980s

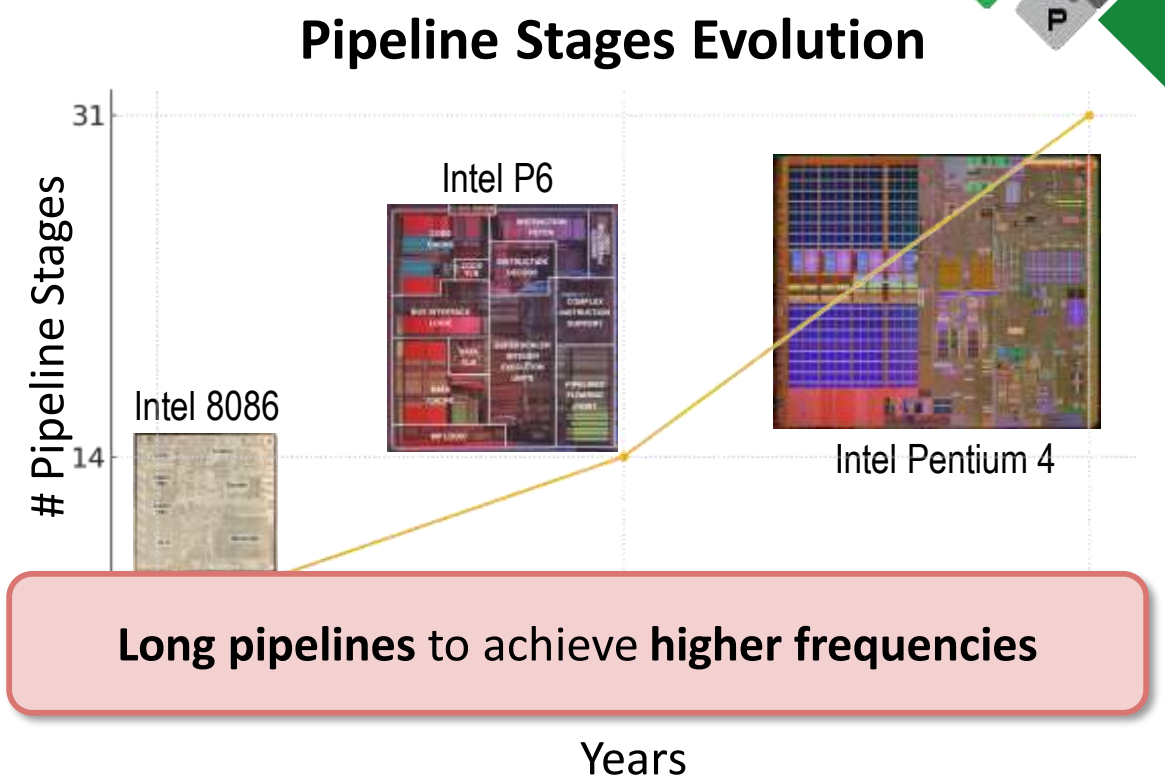
1990s

2000s

2010s

2020s

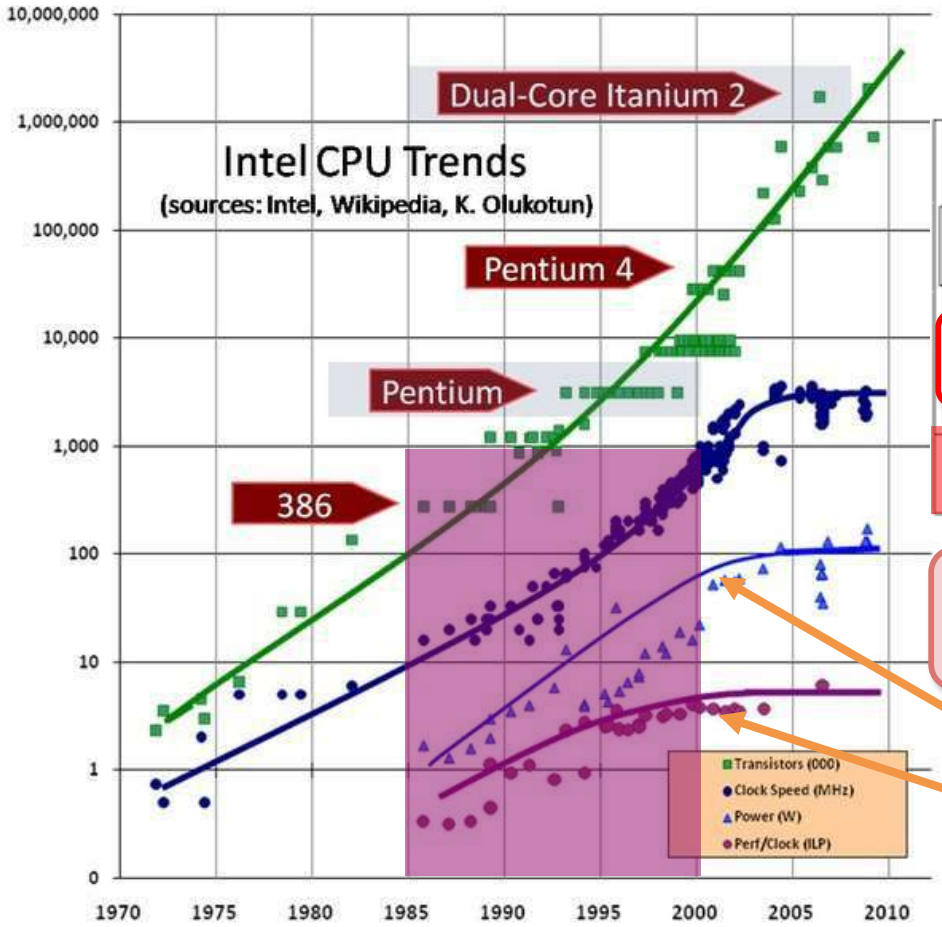
The Clock Frequency Race



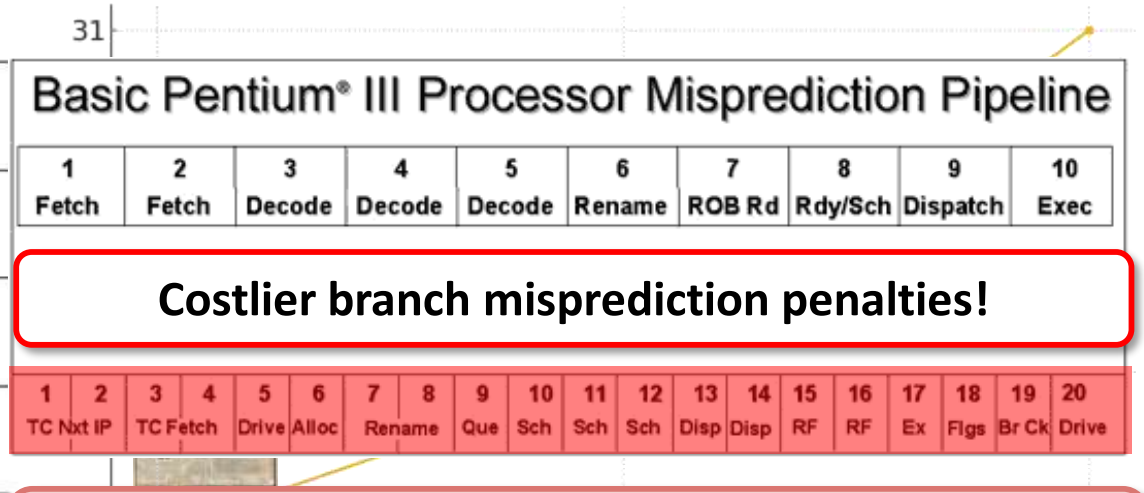
Long pipelines to achieve **higher frequencies**

1970s
1975s
1980s
1990s
2000s
2010s
2020s

The Clock Frequency Race



Pipeline Stages Evolution



Costlier branch misprediction penalties!

Long pipelines to achieve **higher frequencies**

Years

Each **incremental GHz** requires **disproportionately more power** and produces **more heat**.

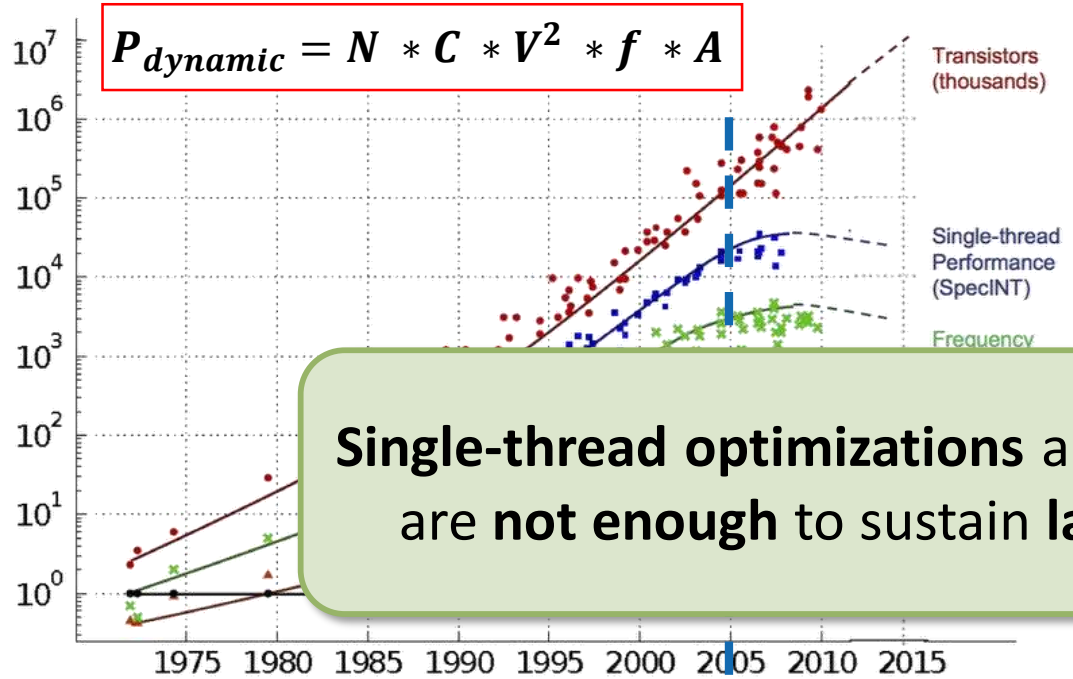
Initial **performance gains** with **increased clock speeds**

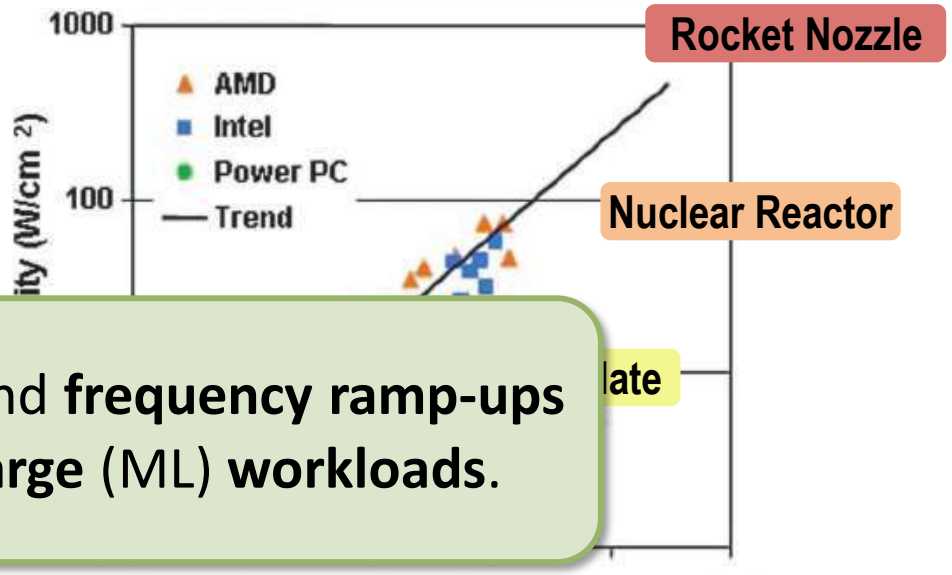
The End of Dennard Scaling

1970s
1975s
1980s
1990s
2000s
2010s
2020s

Power Wall: **higher frequencies** cause **exponential increase** in **heat** + **power** density

$$P_{dynamic} = N * C * V^2 * f * A$$

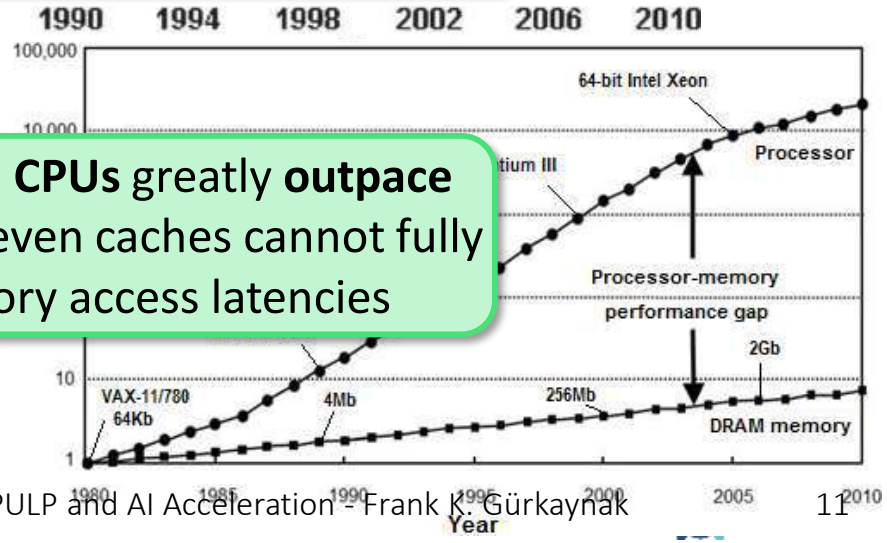




Single-thread optimizations and **frequency ramp-ups** are **not enough** to sustain **large** (ML) **workloads**.

Scaling down transistors allows **proportional voltage** + **power reductions**

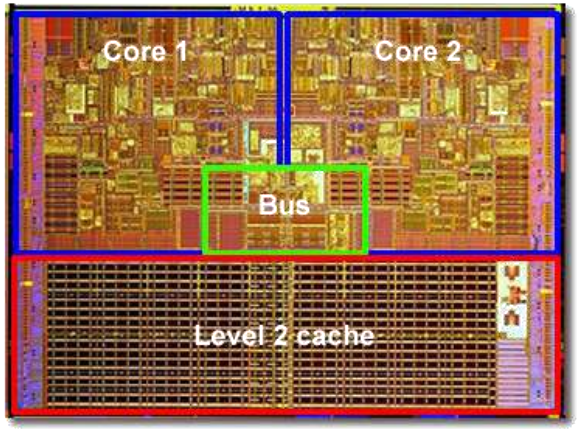
Memory Wall: **CPUs** greatly **outpace DRAM** speeds; even caches cannot fully mask memory access latencies



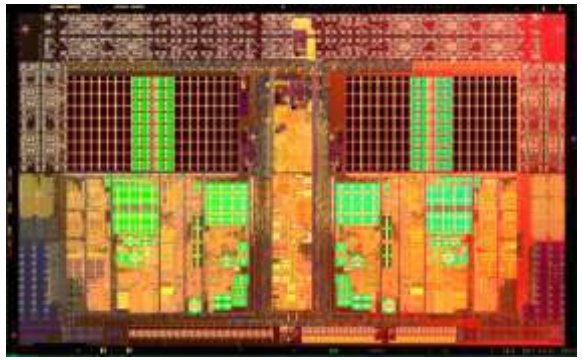
End of Dennard Scaling

1970s
1975s
1980s
1990s
2000s
2010s
2020s

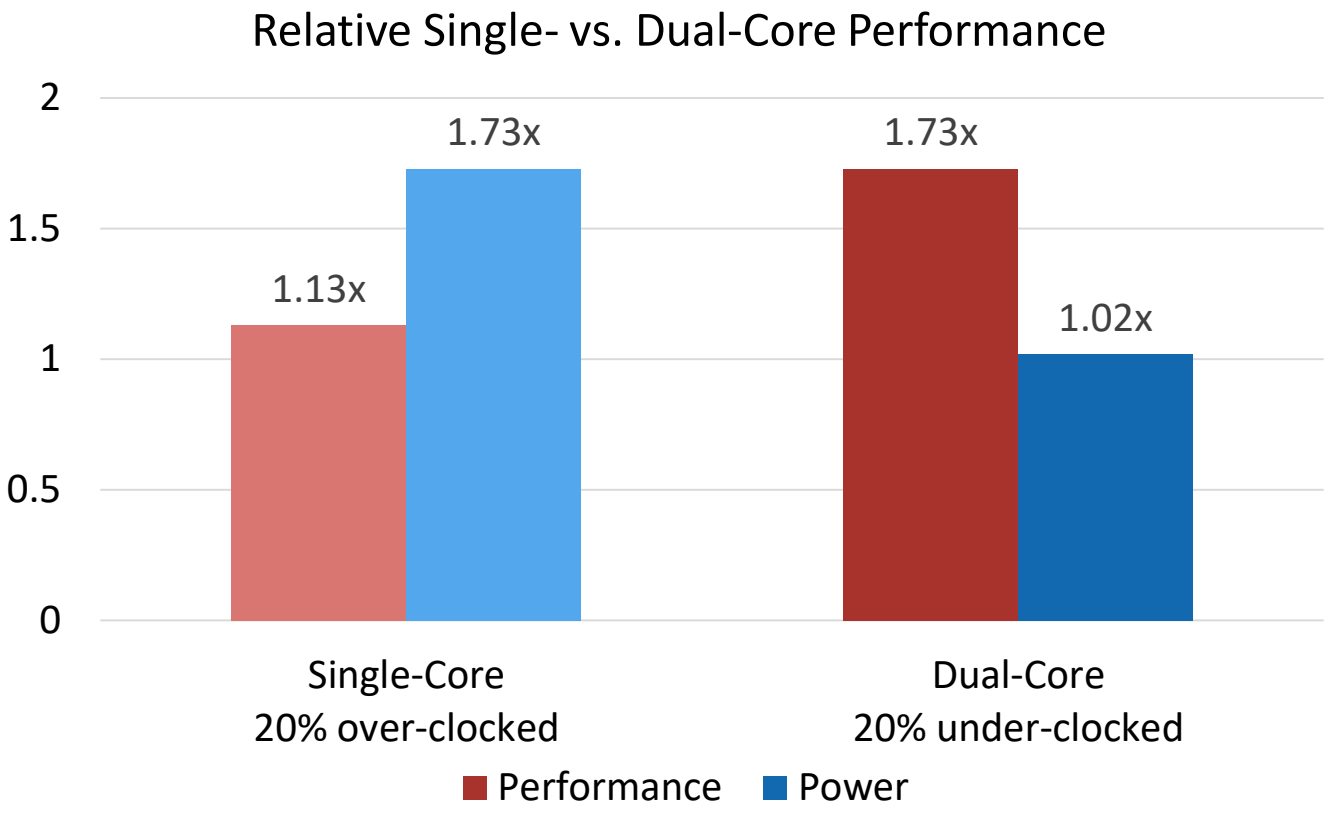
The Multicore Era



Intel Core 2 Duo

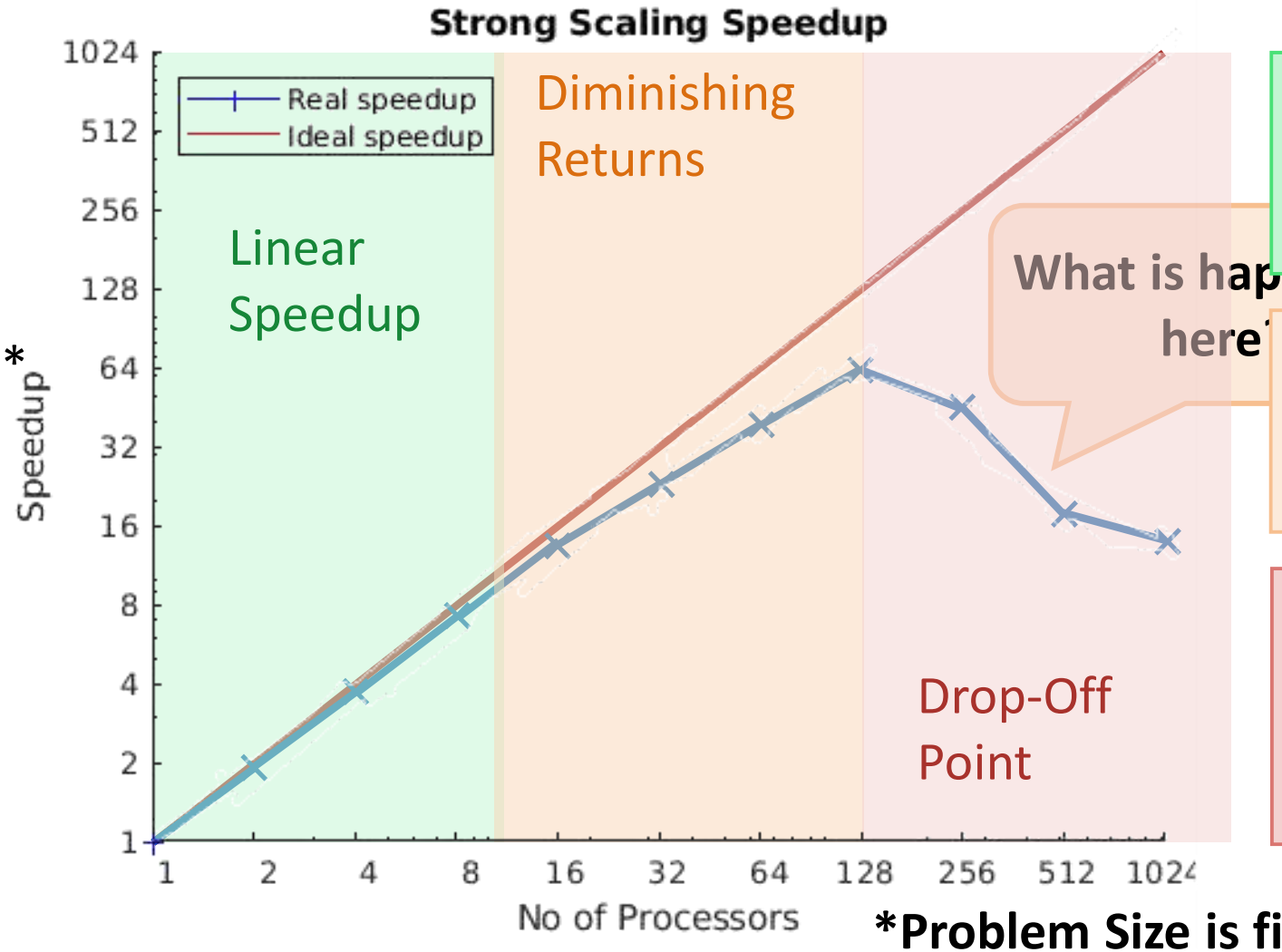


AMD Athlon X2



A **single-core system** delivers just a **13% performance boost** but consumes **73% more power**, while a **dual-core** setup provides a **73% performance increase** with only a **2% rise** in **power** consumption.

If your problem is too big, just throw more cores at it



What is happening here?

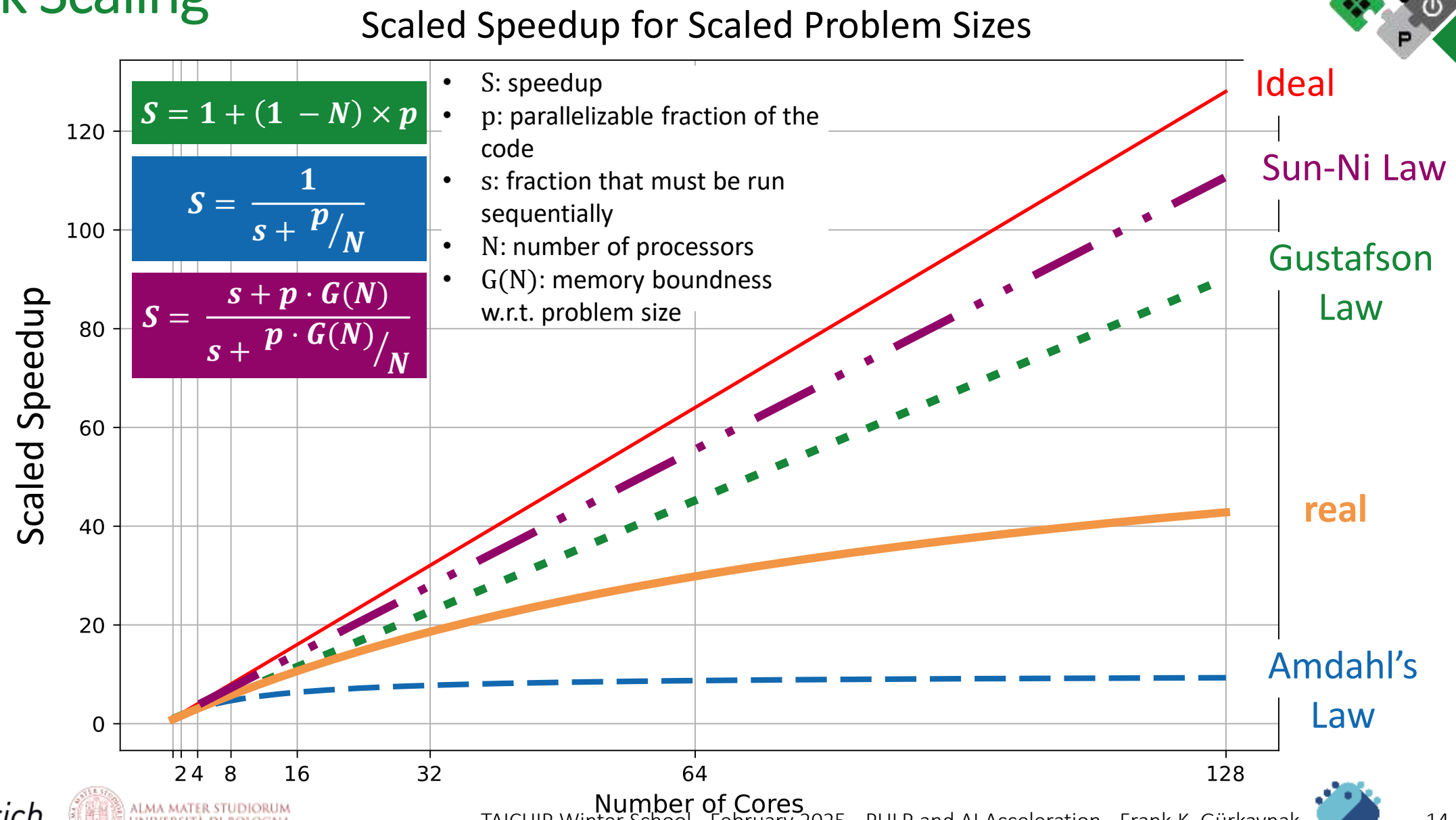
- At this stage, adding more processors leads to nearly ideal speedup.
- Tasks are well-parallelized, and overheads are minimal.

- Real speedup starts deviating from the ideal speedup
- Communication overhead, memory bottlenecks, or serial parts of the program limit scaling

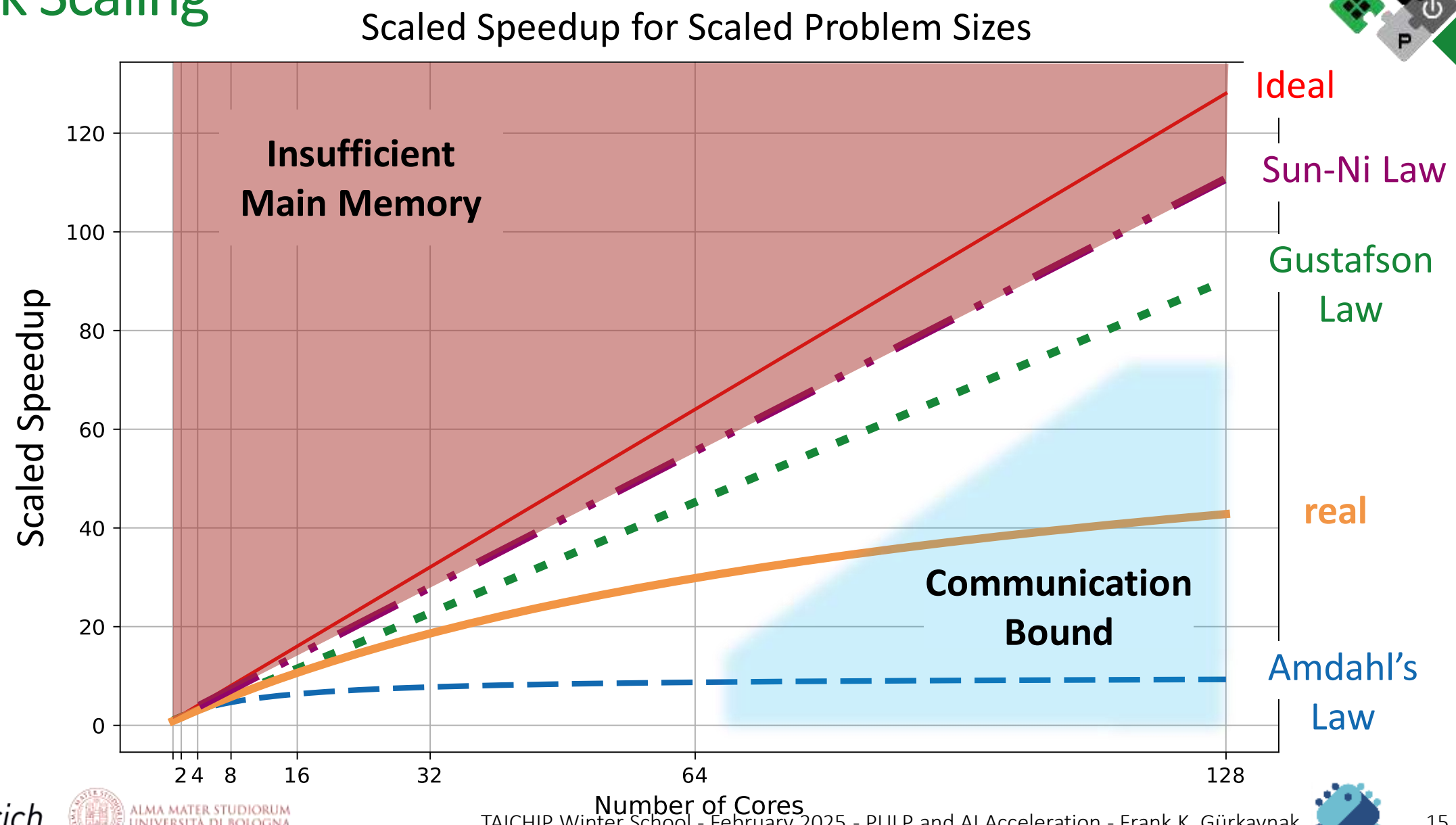
- Adding more processors causes speedup to drop
  - **Synchronization overhead** increases
  - **Communication** between processors becomes the bottleneck
  - Not enough parallel work left to distribute.

***Problem Size is fixed**

Weak Scaling

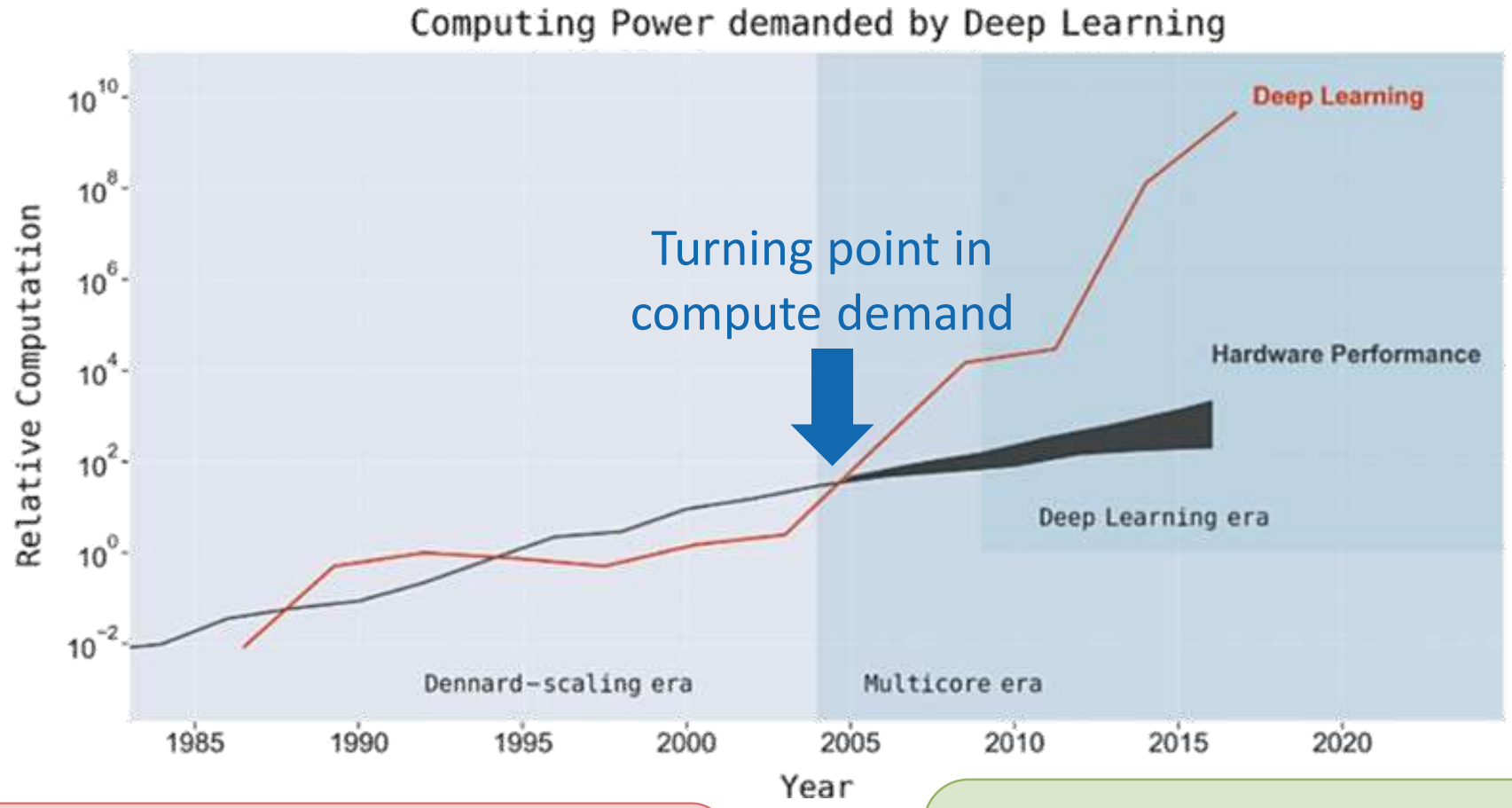


Weak Scaling



1970s
1975s
1980s
1990s
2000s
2010s
2020s

The Rise of ML



ML inference and **training** start to require **far more processing power** than traditional scaling can reliably provide.

The need arises to **understand** the **underlying computations** and thus, **computational requirements** in **ML/AI** models.

Classic Compute

Users, Sensors, …

Web, Mobile, Cloud …

Text, Data, Audio, Video, …

Input | **Applications** | **Output**

X86, M3, ARM, FPGAs, …

Classic CPUs (local or cloud)

Local or Cloud

Classic CPUs (local or cloud)

Python, Rust, C/C++, …

Programming Language

Humans

Human-Programmed Rules

ML

Users, Sensors, …

Web, Mobile, Cloud …

Text, Data, Audio, Video, …

Input | **Inference** | **Predictions**

GPUs, CPUs, Cloud…

Trained ML/AI Model

Multi-core AI chips, Cloud for large models

Tensorflow, Pytorch, MxNet

GPUs, FPGAs, ASICs, TPUs

Training Algorithm

Training Data

Classic Compute

Input | Applications | Output

Classic CPUs (loca

Predefined **Rules**, **Logic** and **Knowledge** are **coded** into programs **by humans.**

Programming

Classic CPUs (local or cloud)

Programming Language

Human-Programmed Rules

ML

Input | Inference | Predictions

GPUs, C

Trained

Teaches a **DNN** to **learn from examples** (training data), rather than being explicitly programmed.

ML - Training

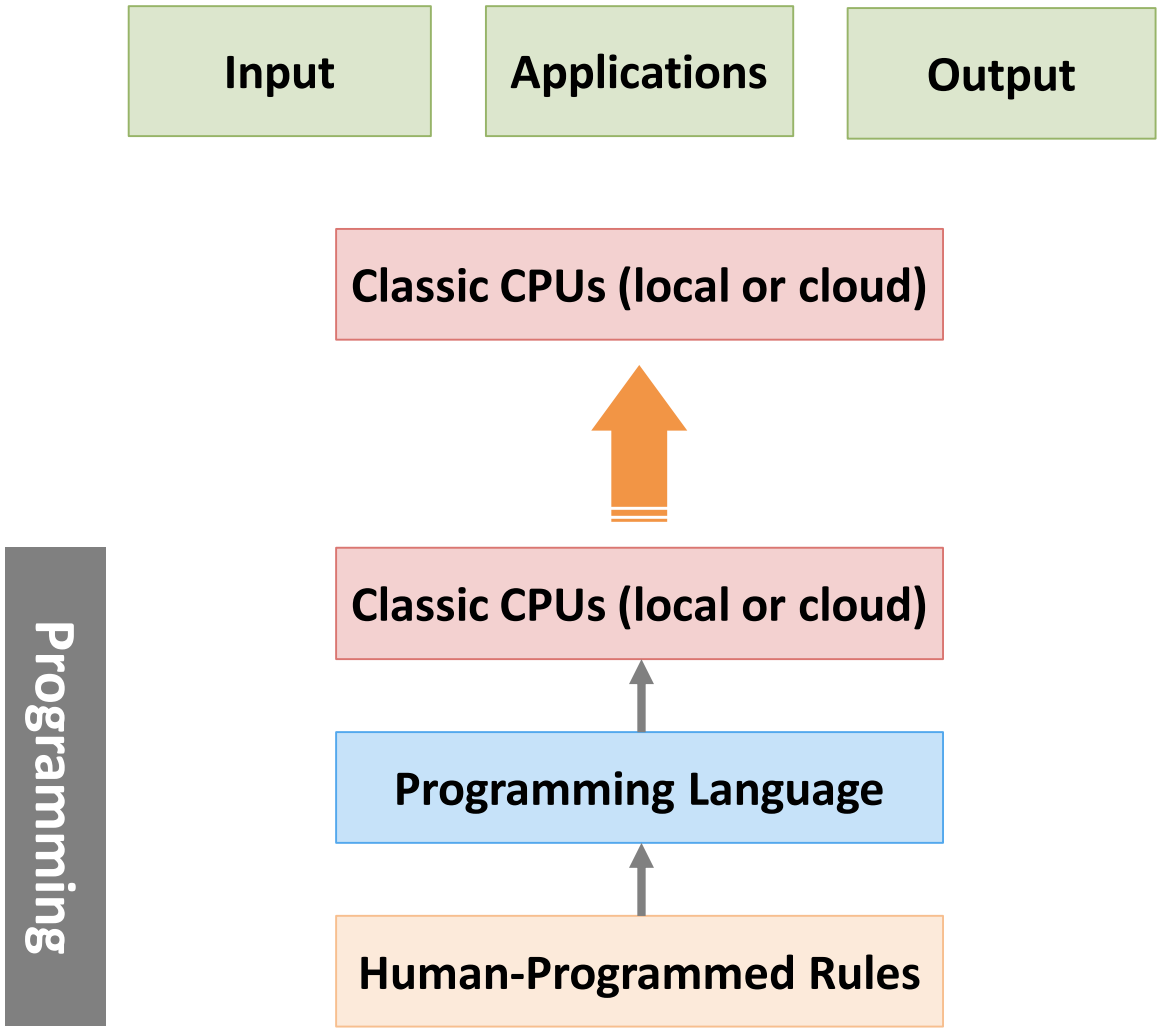
Tensorflow, Pytorch, MxNet

GPUs, FPGAs, ASICs, TPUs

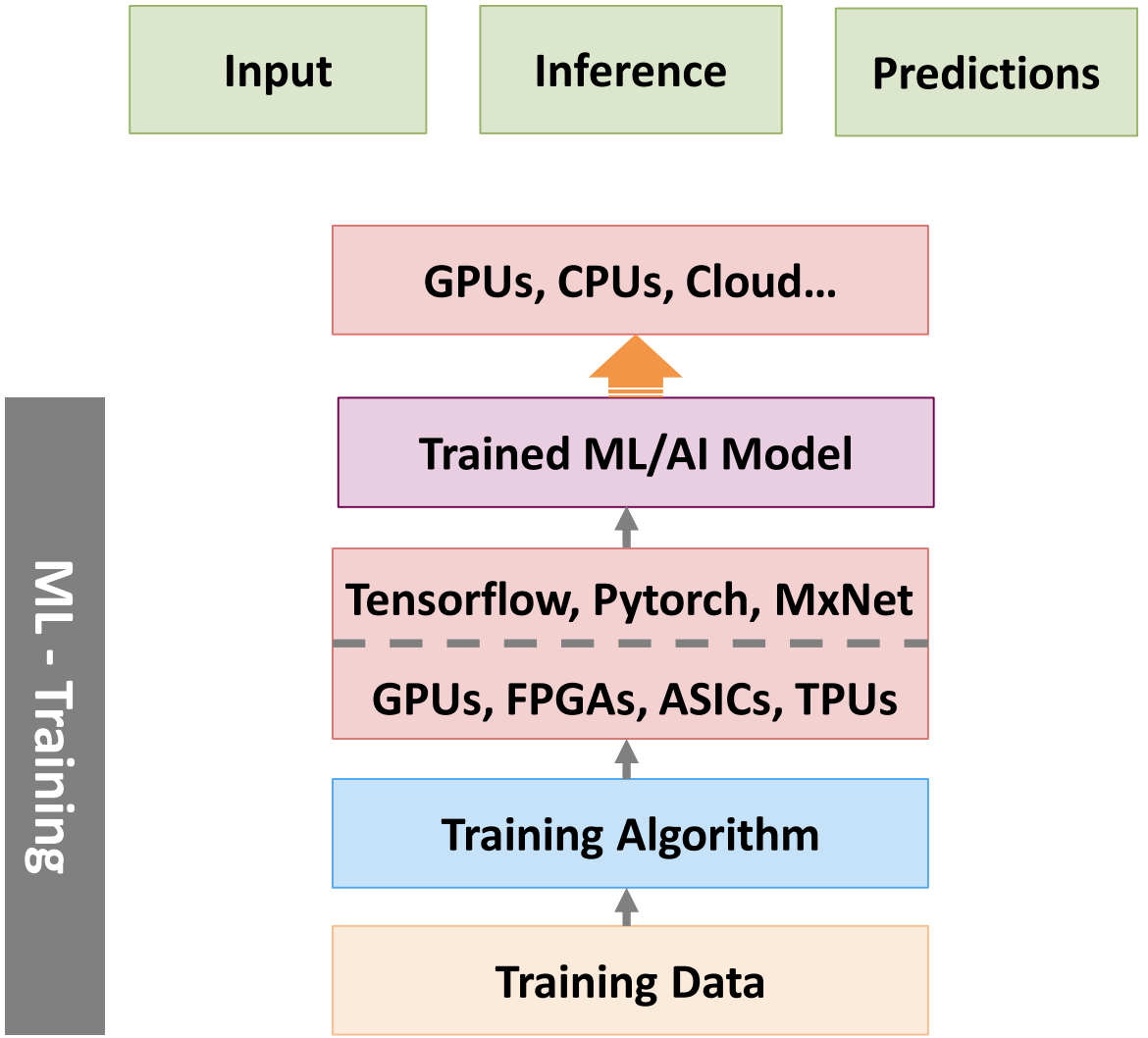
Training Algorithm

Training Data

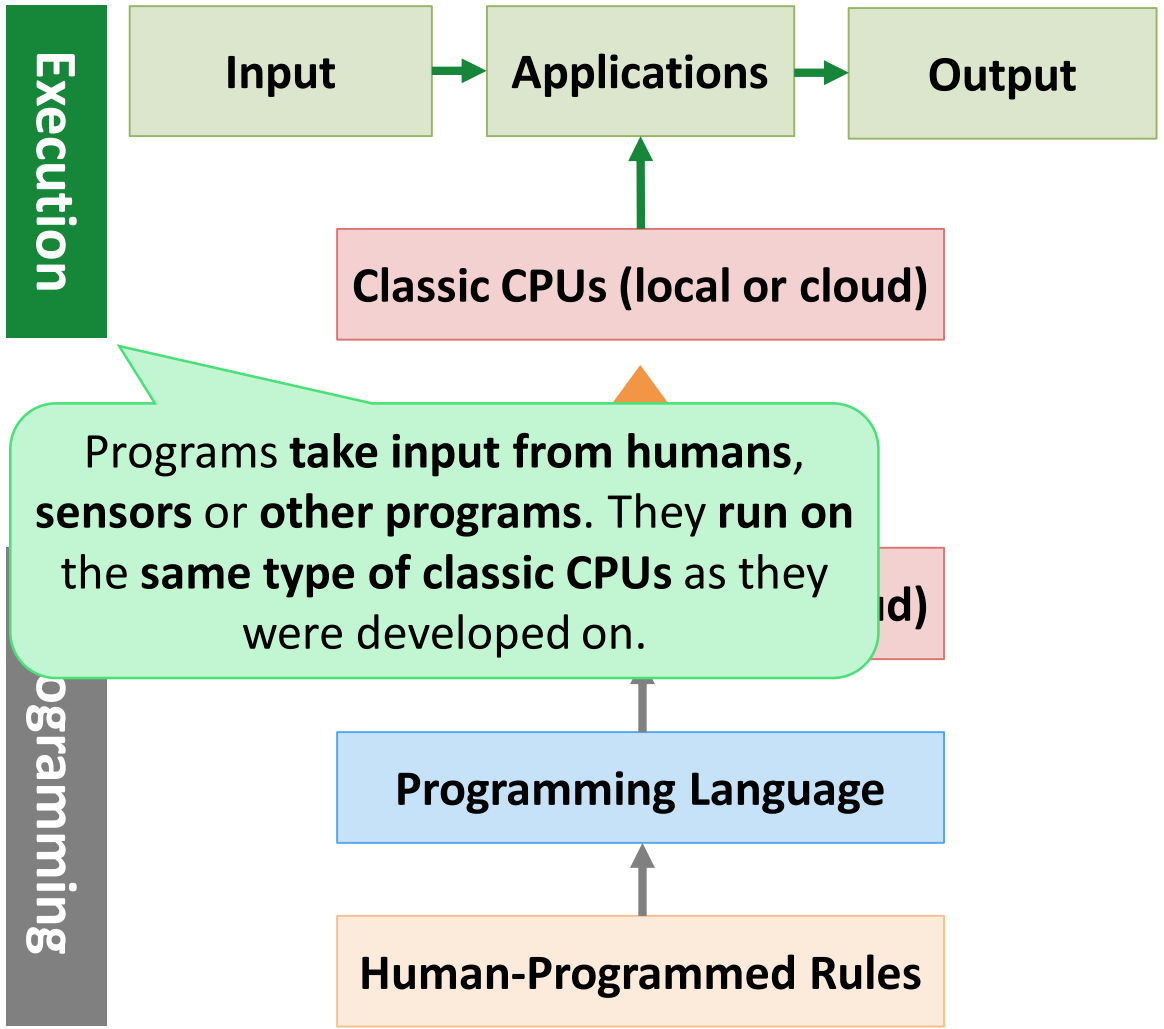
Classic Compute



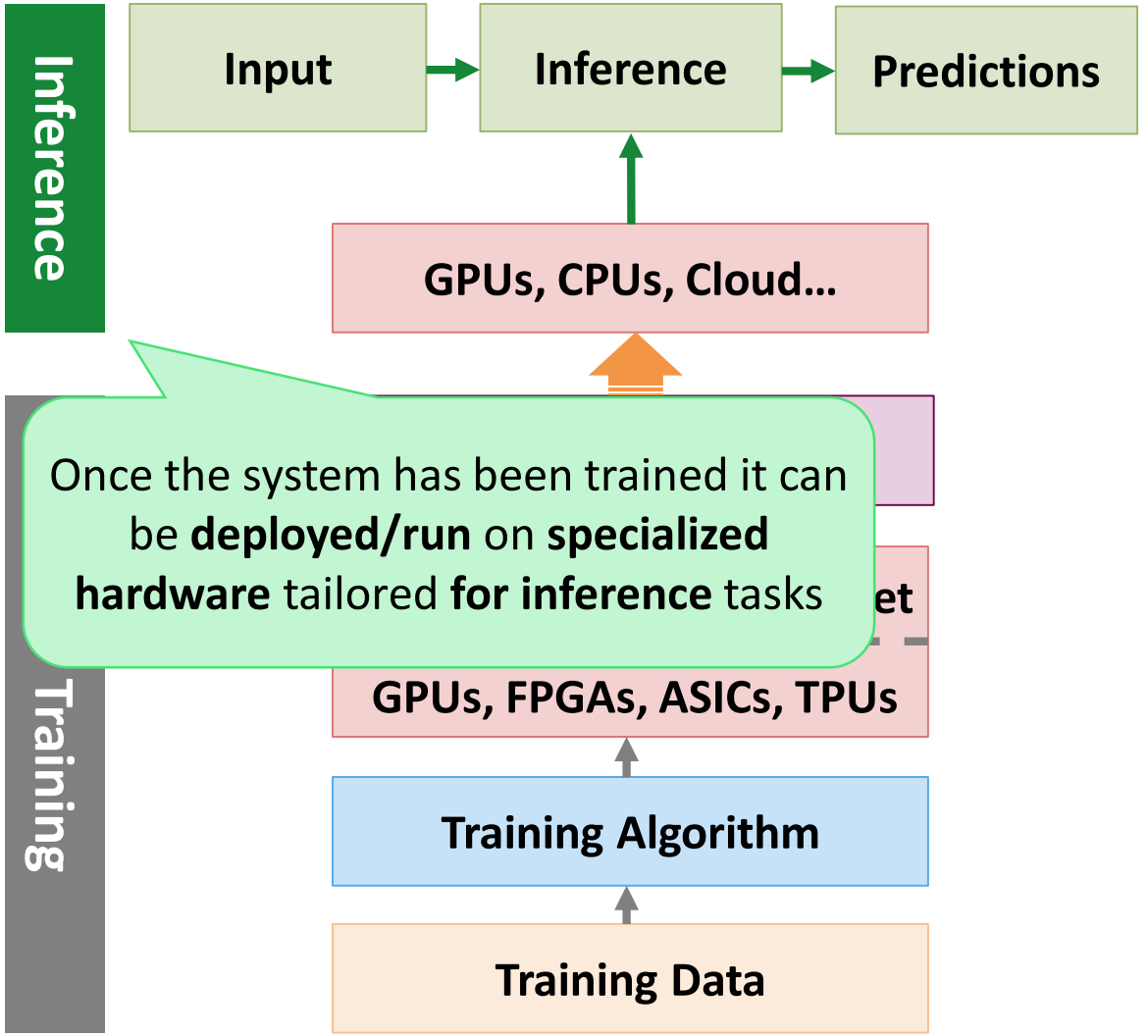
ML



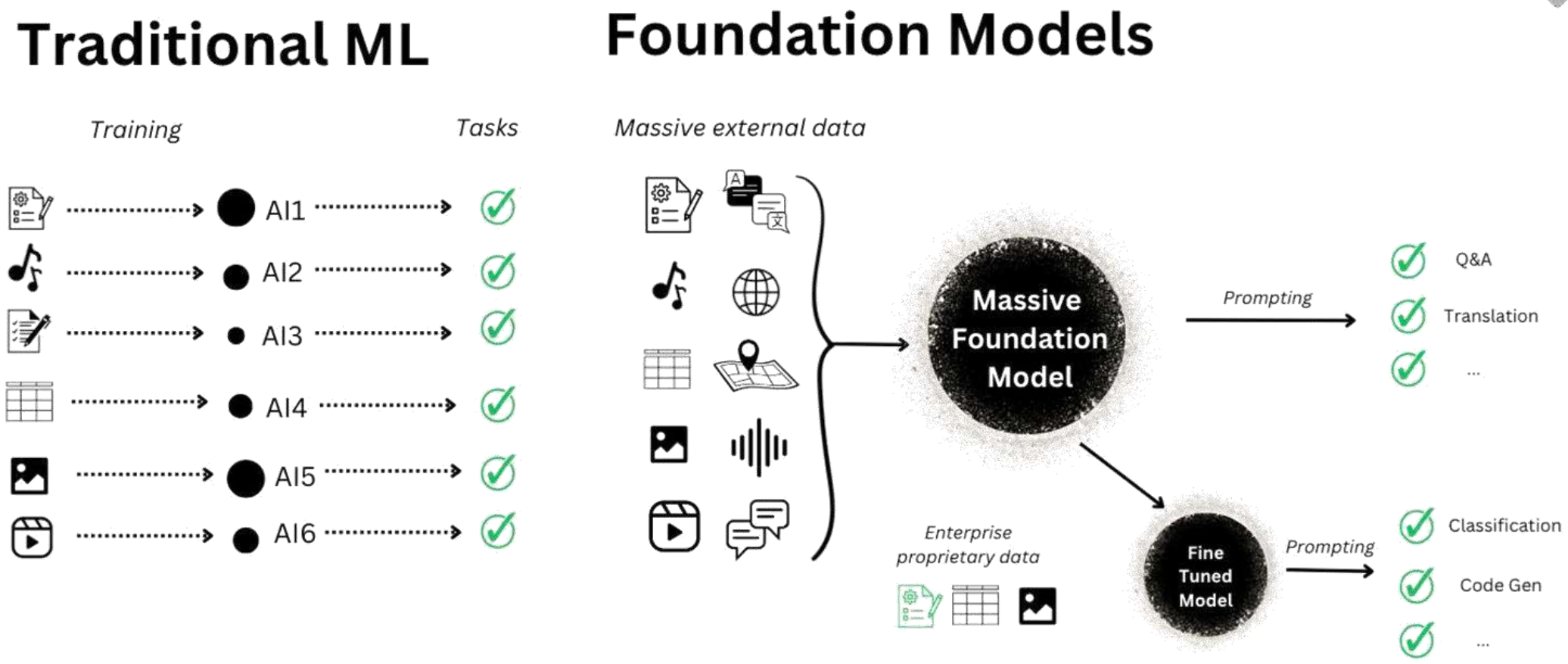
Classic Compute



ML



The Era of GenAI: ML is Yesterday's News

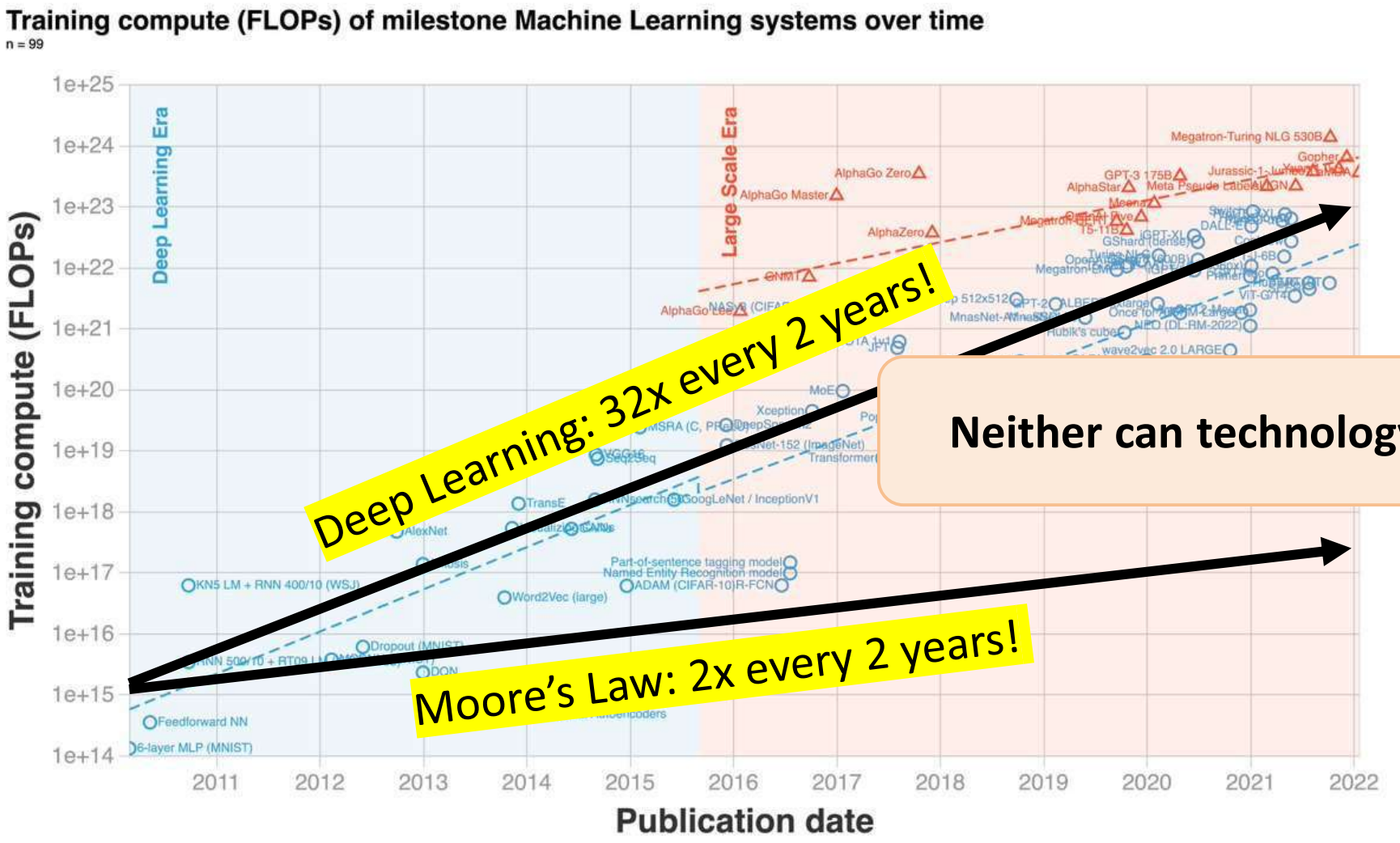


- Individual siloed models
- Require task-specific training
- Lots of human supervised training

- Massive multi-tasking model
- Adaptable with little or no training
- Pre-trained unsupervised learning

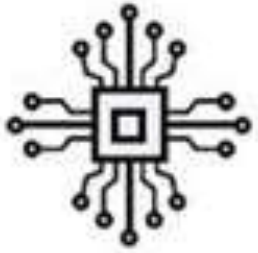
Source: Armand Ruiz

The Era of GenAI: Scaling Laws



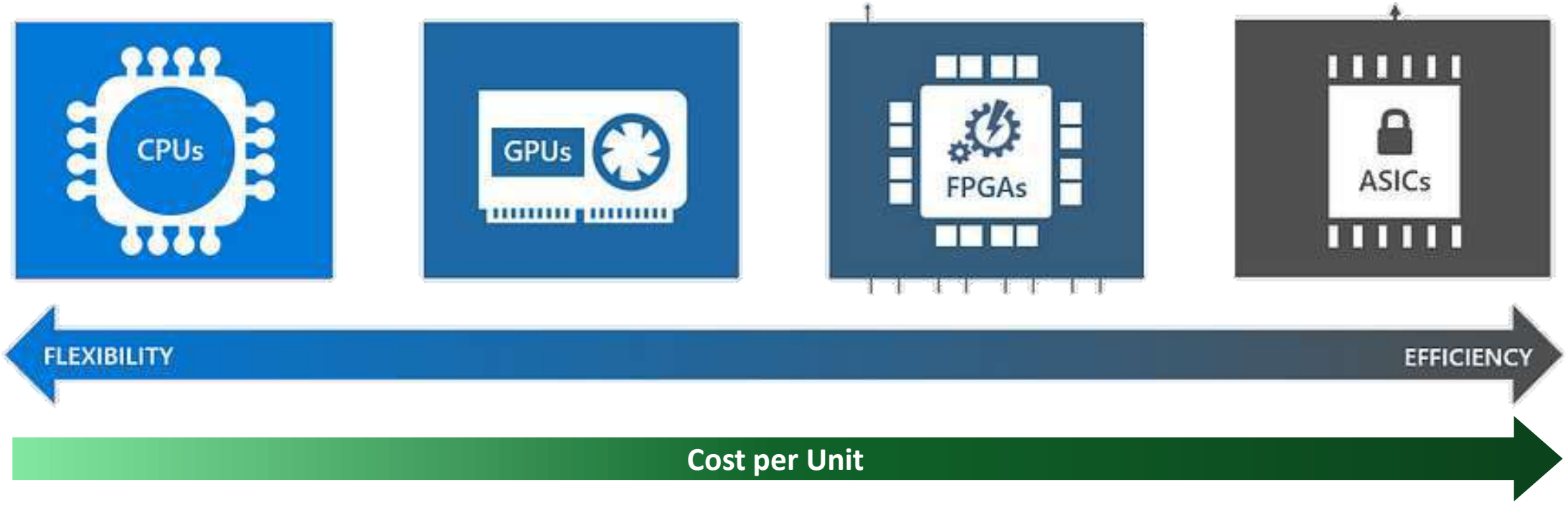
Neither can technology scaling…

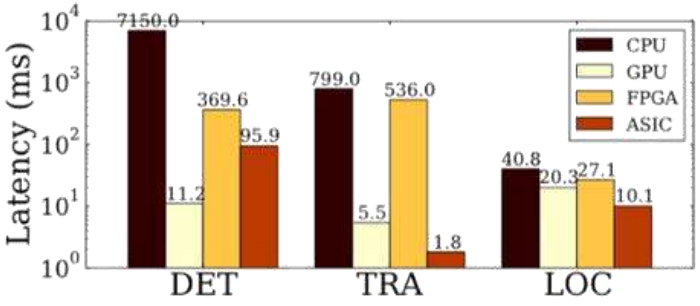
GenAI Efficiency: Learning from the Best Hardware

	Weight	Space	Processor Speed	Energy Efficiency
	3 pounds (1.4 kg)	1/6 basketball (80 cubic inches or 1,300 cm^3)	Up to 1,000,000 trillion operations per second	20 watts
	150 tons	Basketball court (cabinets over 4,350 square feet, or 400 m^2)	93,000 trillion operations per second	10 million watts

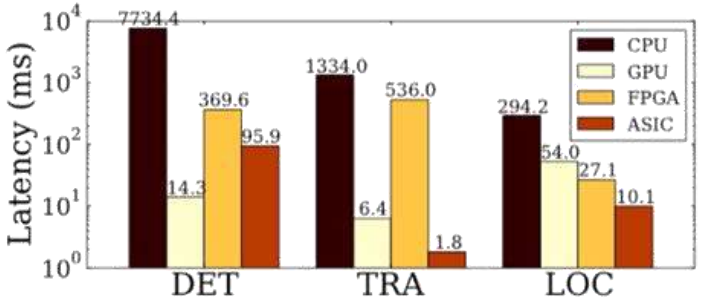
https://www.educba.com/artificial-intelligence-vs-human-intelligence/

ML & AI: The Energy Challenge

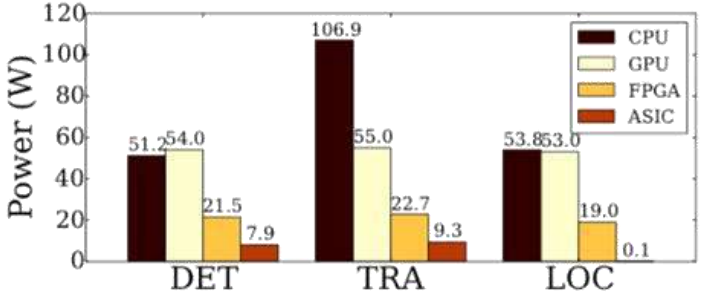




(a) Mean Latency Across Platforms

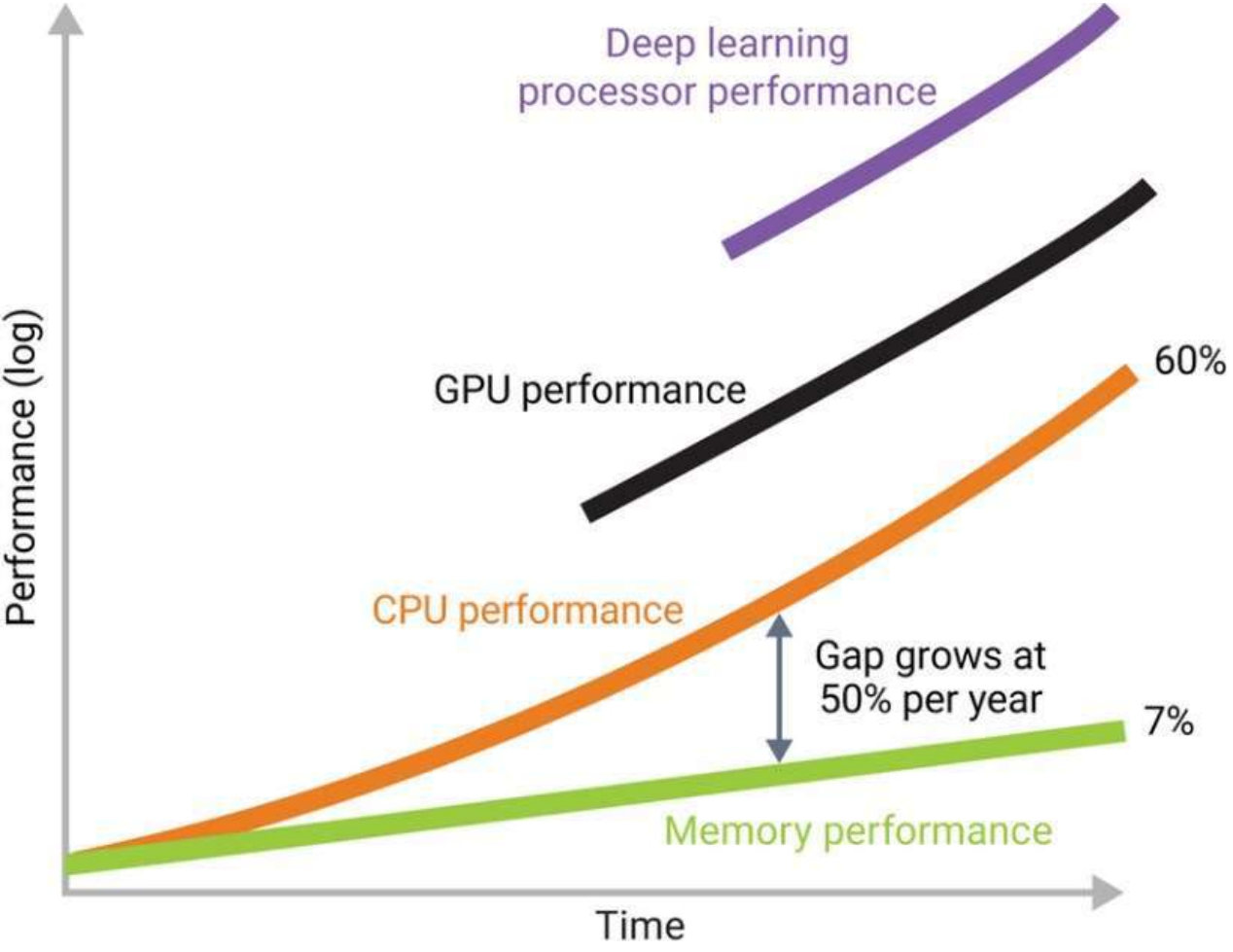


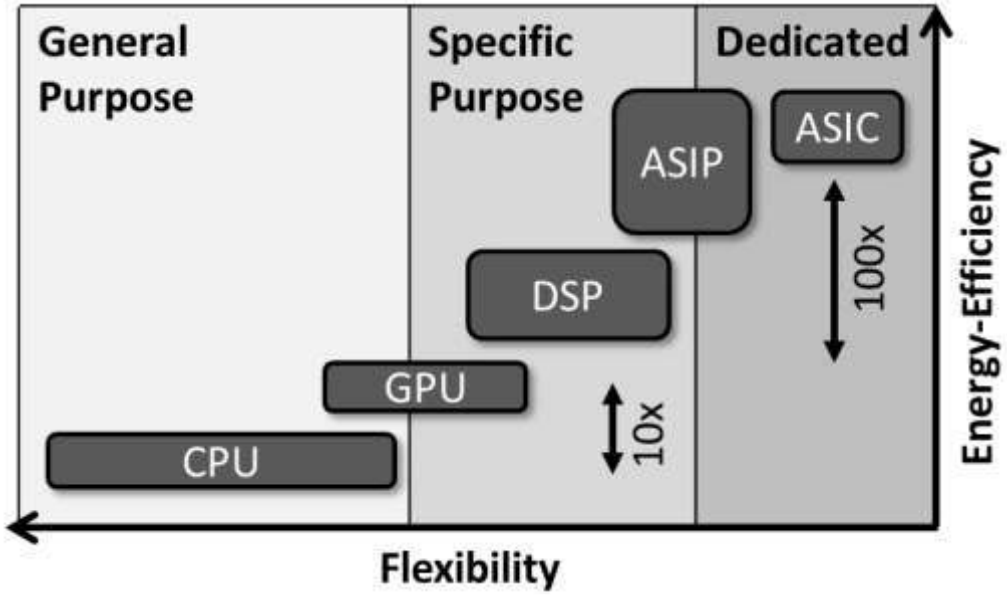
(b) 99.99th-Percentile Latency Across Platforms



(c) Power Consumption Across Platforms

The Trend: Accelerators

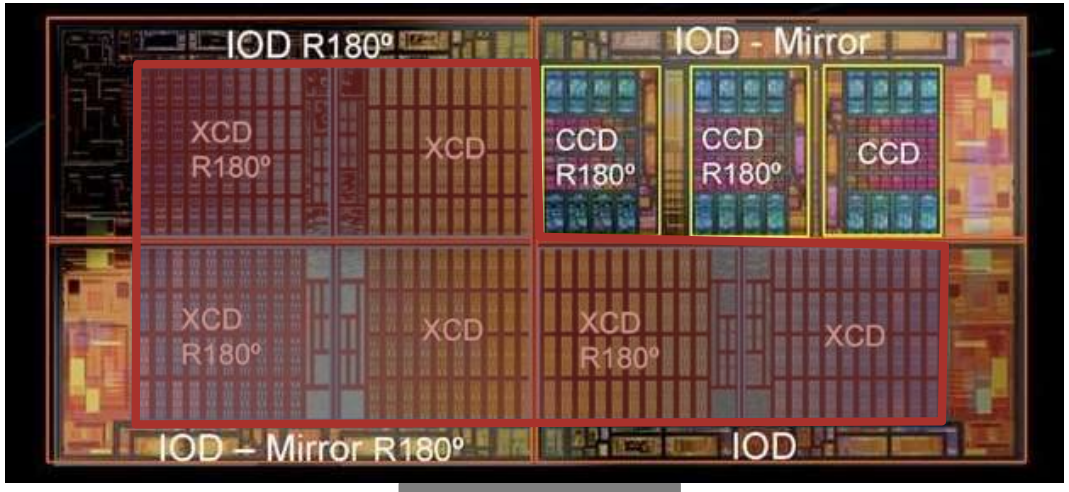




- Specialized Compute chip for specific workloads
- Today mostly for Machine learning
- GPUs dominate, but even more specialized accelerators are coming

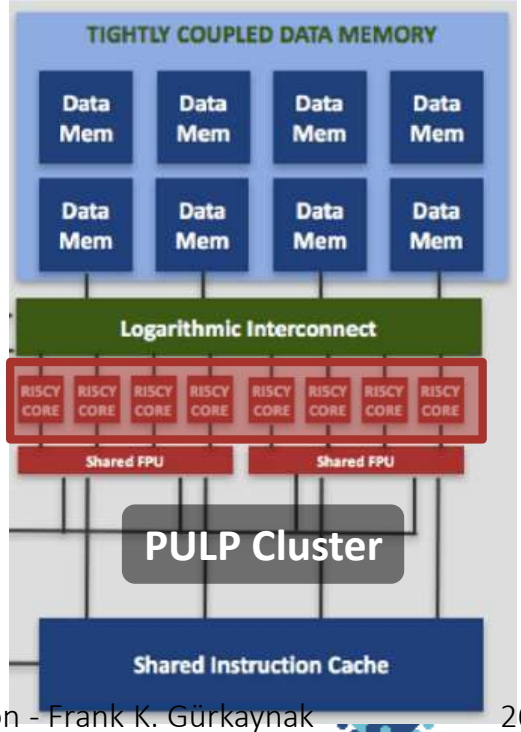
An Overview of Accelerators

Accelerator Type	Description	Applications	Models	Hardware
Highly Parallel Accelerators	Massive parallelism for matrix computations.	**CNN/DNNs**: Efficient image processing.	ResNet, EfficientNet	NVIDIA A100, AMD MI300, **PULP cluster**



AMD MI300

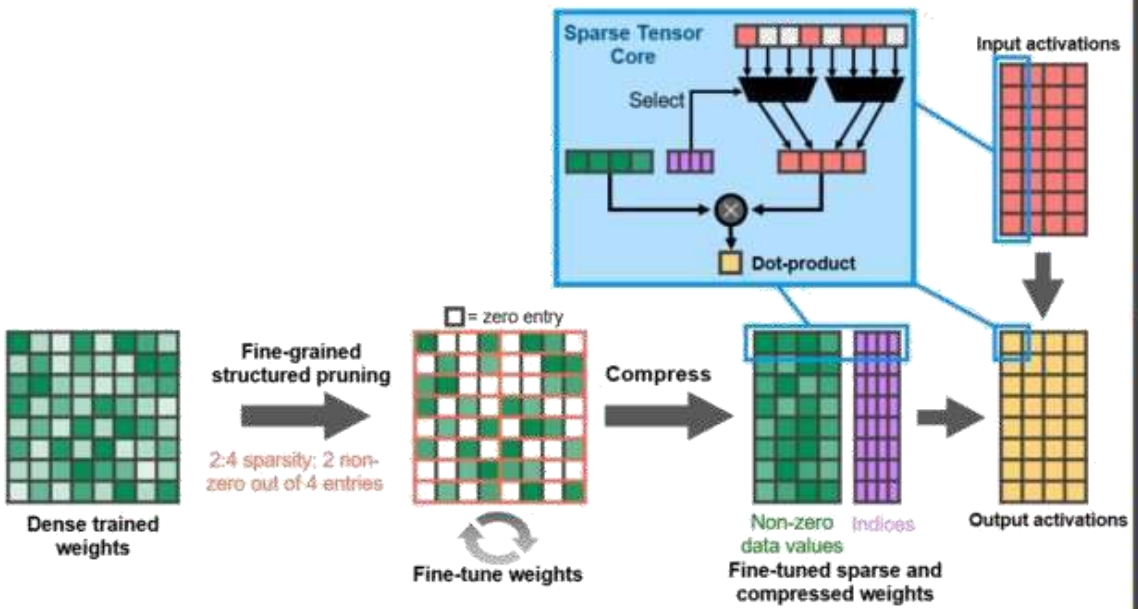
NVIDIA A100

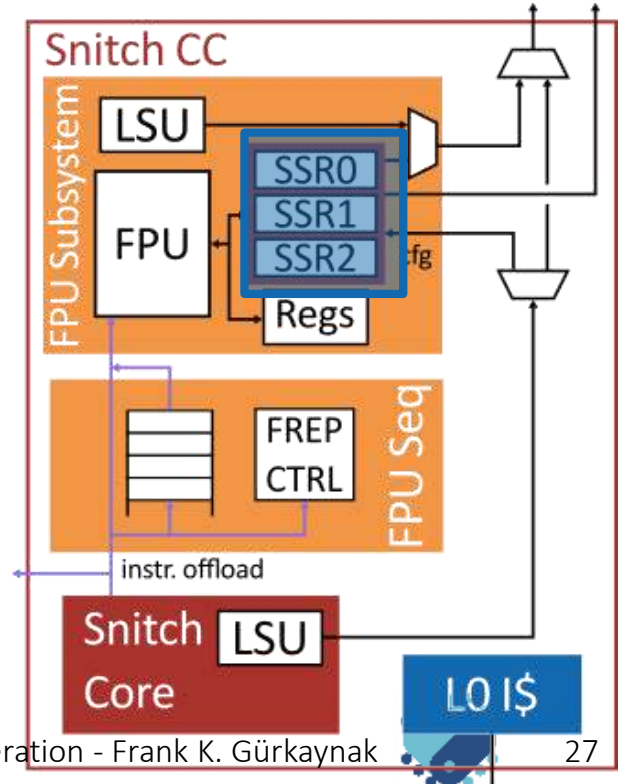


PULP Cluster

An Overview of Accelerators

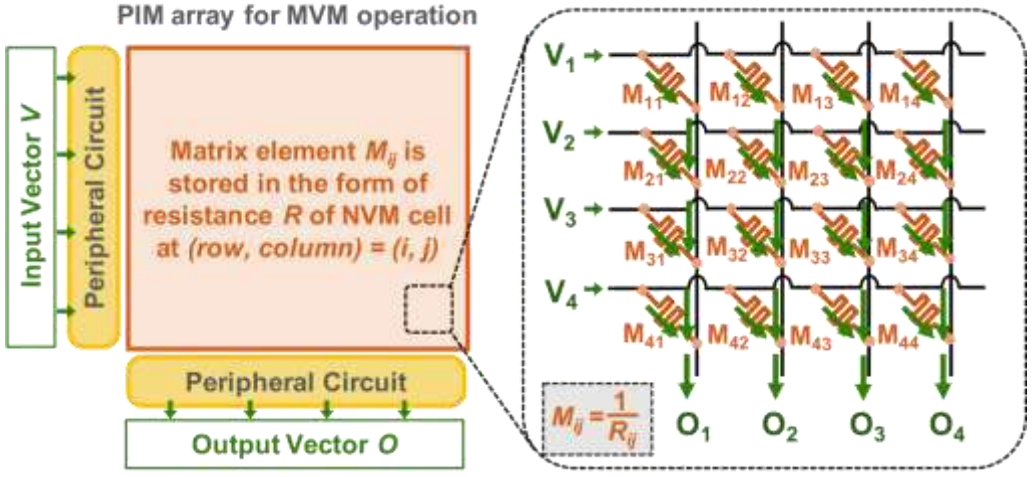
Accelerator Type	Description	Applications	Models	Hardware
Sparse Accelerators	Exploits sparsity to reduce computation and memory costs.	**Sparse Attention**: Efficient for NLP and vision tasks.	BERT, GPT, ViT	Tensor Cores, **Sparse Stream Semantic Registers (SSSRs)**

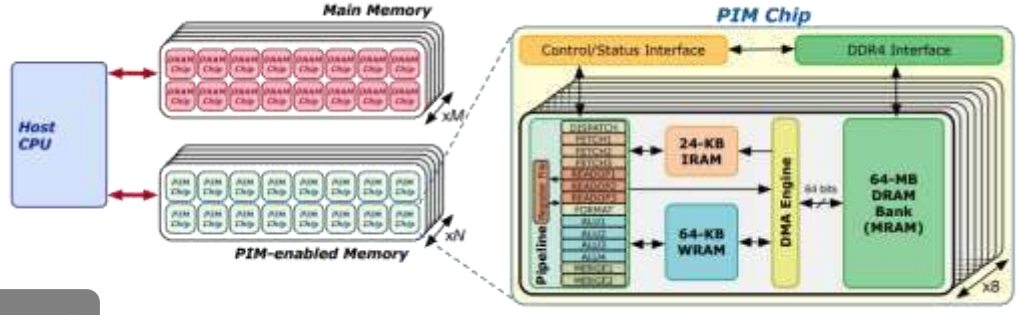




An Overview of Accelerators

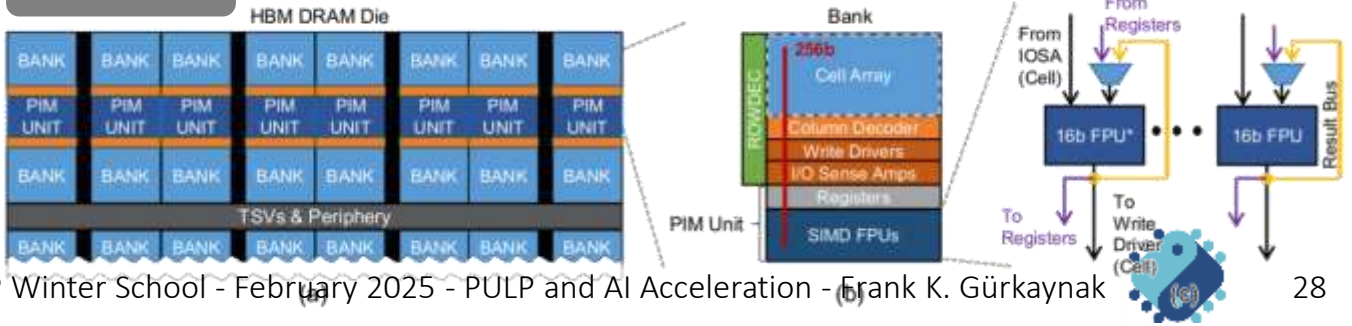
Accelerator Type	Description	Applications	Models	Hardware
Processing-in-Memory (PIM)	Computes near memory to reduce latency and bandwidth bottlenecks.	**GNNs**: Applications in recommendation systems and graph analytics.	GraphSAGE, GCN	Samsung FIMDRAM, UPMEM DPUs





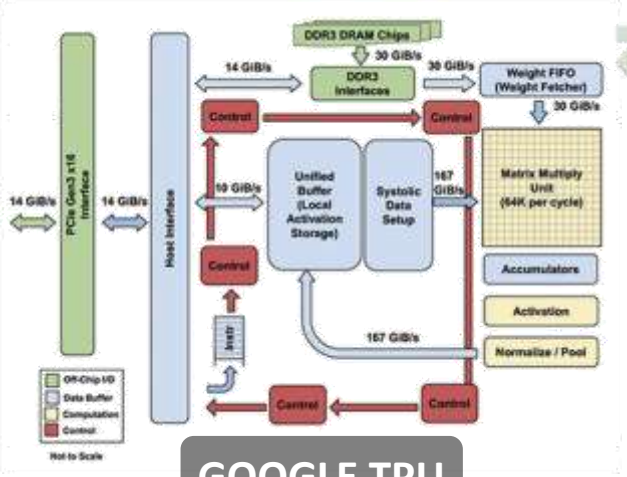
UPMEM

SAMSUNG

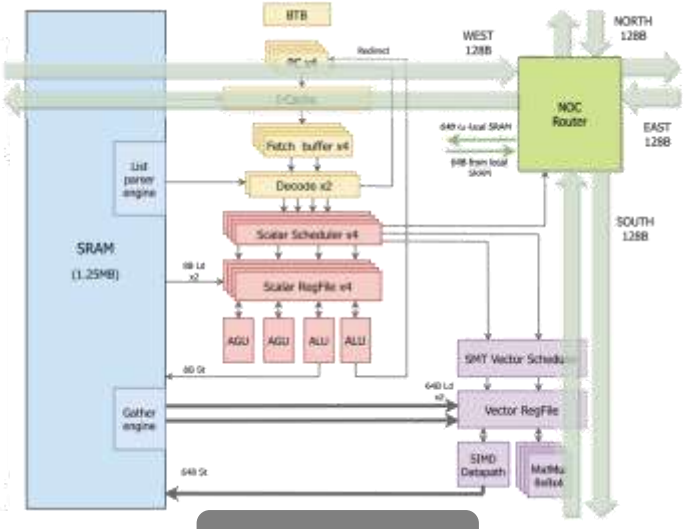


An Overview of Accelerators

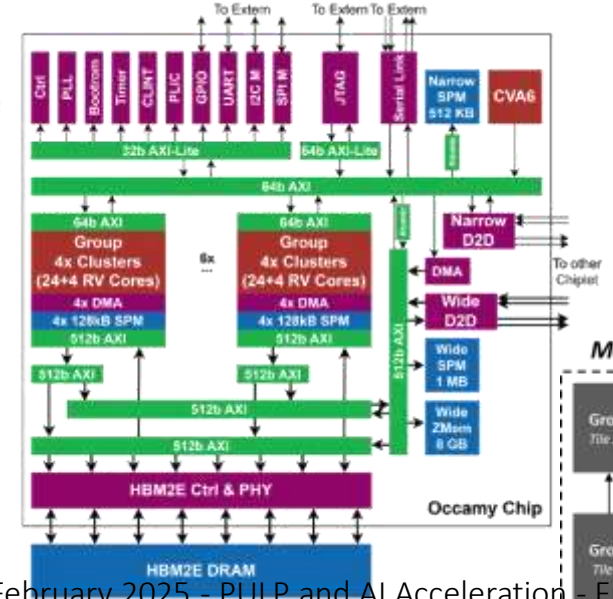
Accelerator Type	Description	Applications	Models	Hardware
Tensor Accelerators	Optimized for matrix multiplications and tensor-heavy workloads.	**Foundation Models**: Powering GPT-4, Stable Diffusion, and other large-scale AI models.	GPT-4, PaLM, MedSAM	Google TPU, Tesla Dojo, Habana Gaudi, **Occamy**, **MemPool**

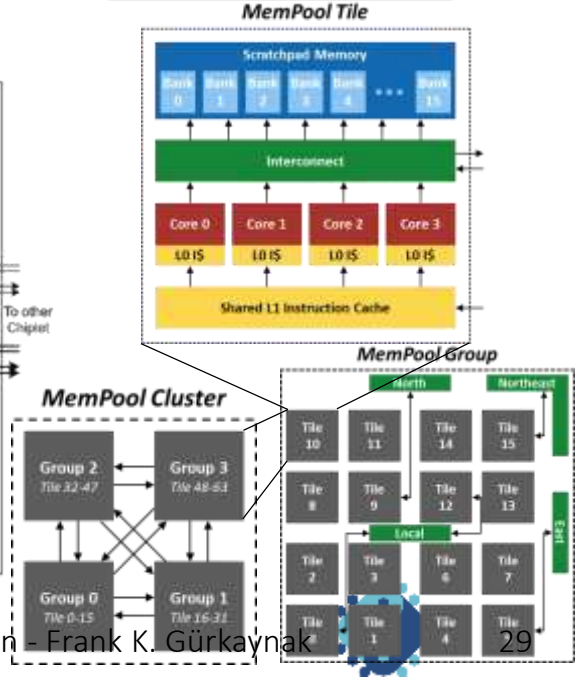


GOOGLE TPU



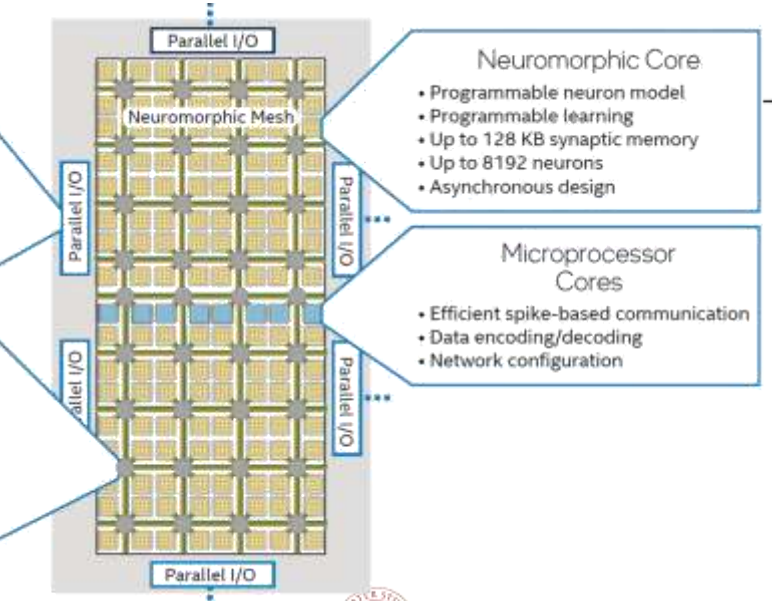
TESLA DOJO

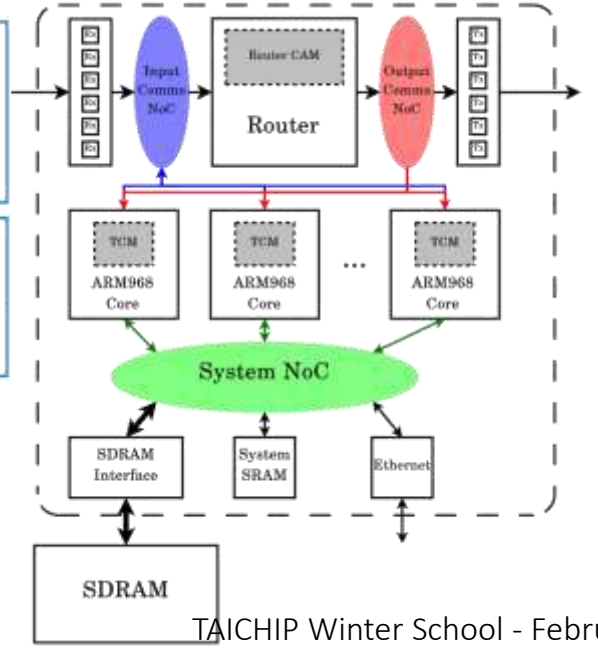


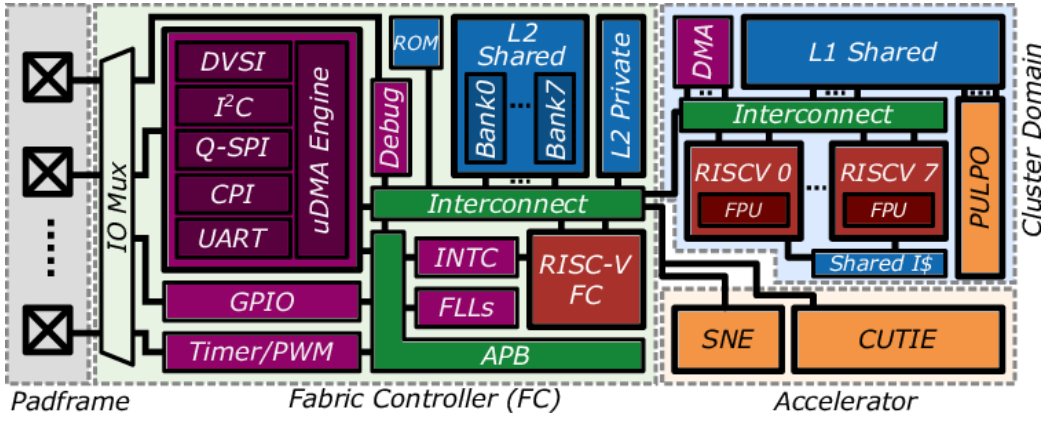


An Overview of Accelerators

Accelerator Type	Description	Applications	Models	Hardware
Neuromorphic Accelerators	Mimics the brain, using spikes for asynchronous event-driven processing.	**Spiking Neural Networks (SNNs)**: Robotics and vision-based event detection.	Spikformer, SSTFormer, SCNN	Intel Loihi, SpiNNaker, **Kraken**





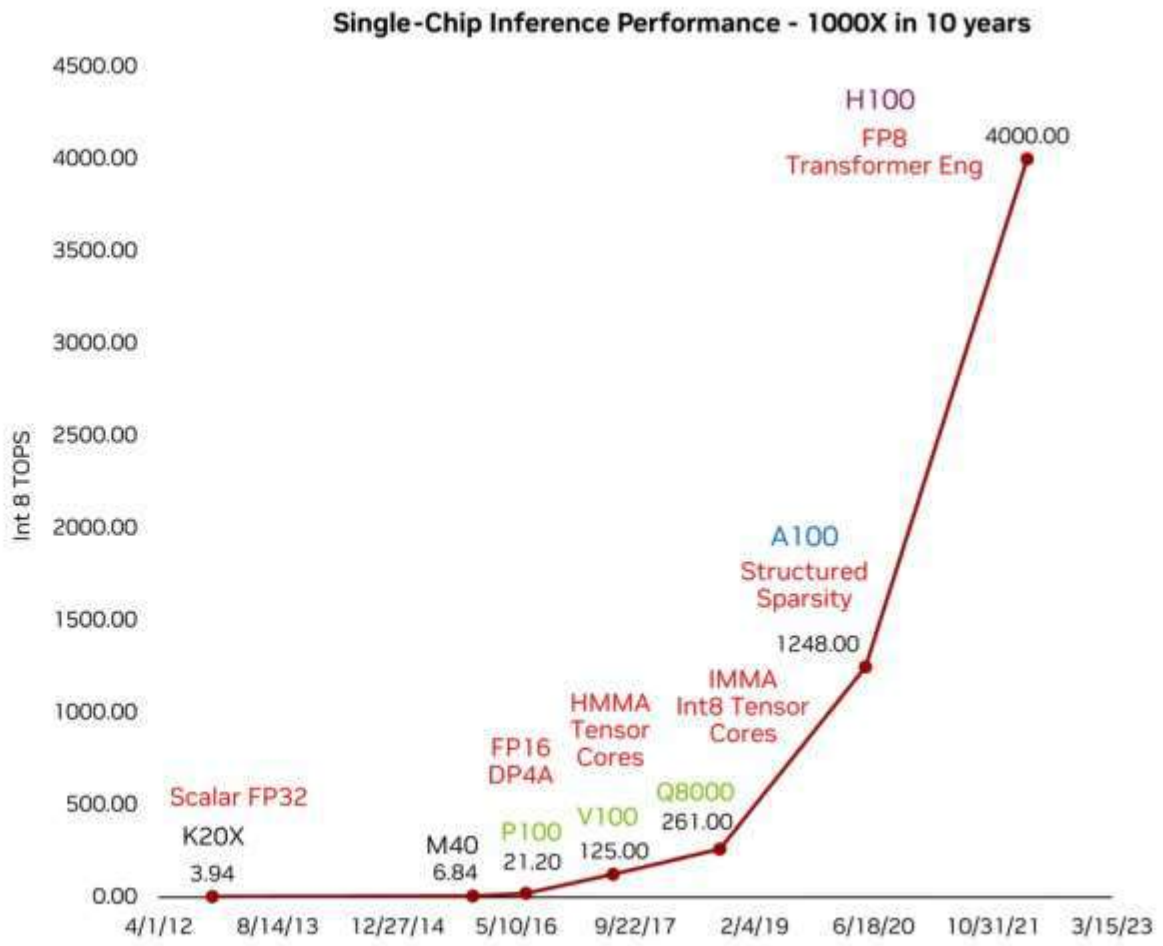


Looking up to the Leader

Dally HotChips 2023

Gains from

- Number Representation
  - FP32, FP16, Int8
  - (TF32, BF16)
  - ~16x
- Complex Instructions
  - DP4, HMMA, IMMA
  - ~12.5x
- Process
  - 28nm, 16nm, 7nm, 5nm
  - ~2.5x
- Sparsity
  - ~2x
- Model efficiency has also improved – overall gain > 1000x



Team of 100 people in ETH Zürich – University of Bologna

- **Research on open-source energy-efficient computing**

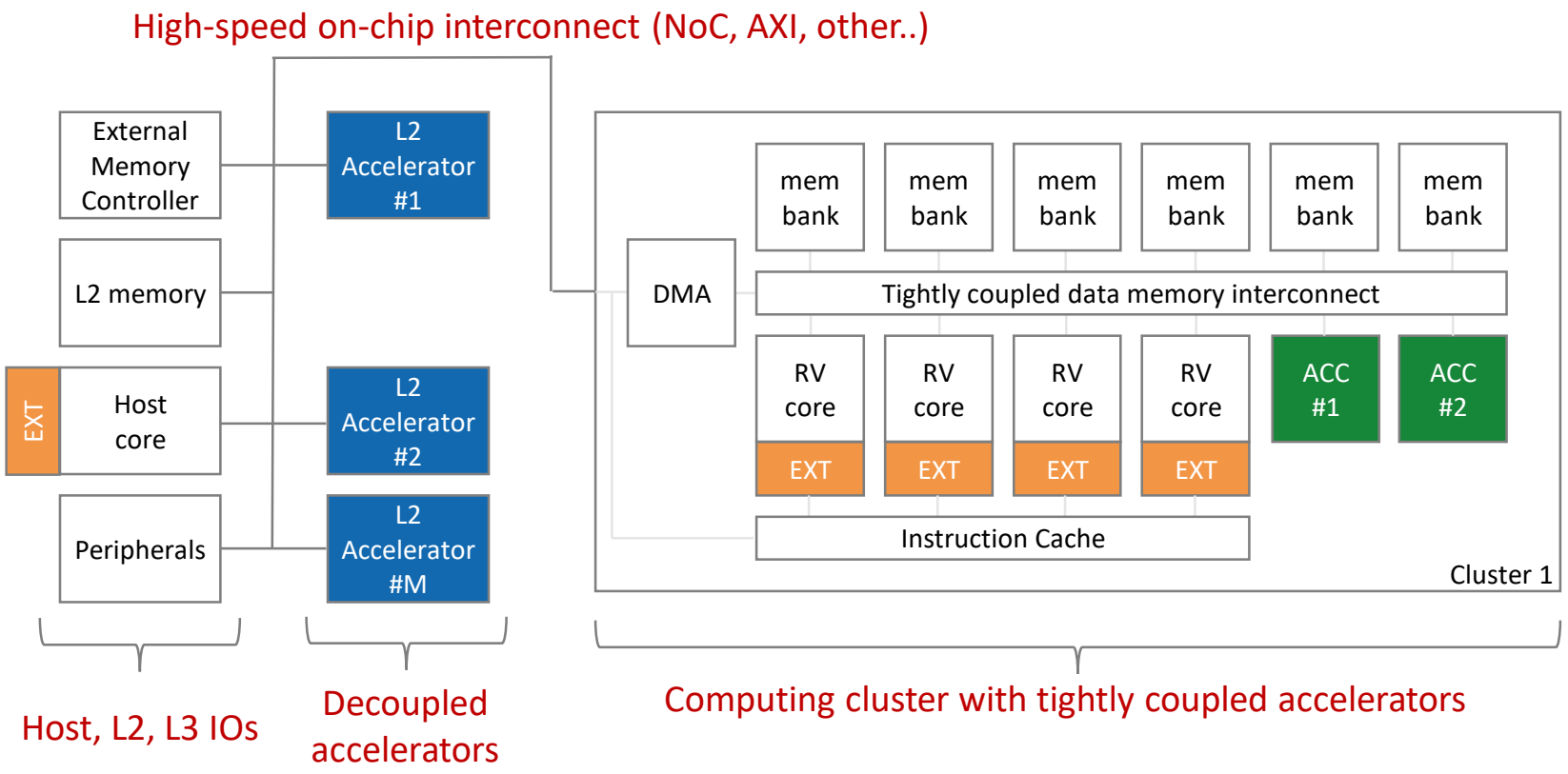
Our research focus: cluster-based many-core accelerators

Multiple Scales of acceleration

Extensions to processor cores
- Explore new extensions
- Efficient implementations

Shared-memory Accelerators
- Domain specific
- Local memory

Multiple Decoupled Accelerators
- Communication
- Synchronization



RISC-V is a key enabler → max agility, enabling SW build-up, without vendor lock-in

PULP creates a toolbox for efficient architectures

RISC-V Cores and Vector Units

RI5CY *CV32E*	Zero R *Ibex*	Snitch	Spatz	Ariane *CVA6*	ARA
RV32	RV32	RV32	RVV	RV64	RVV

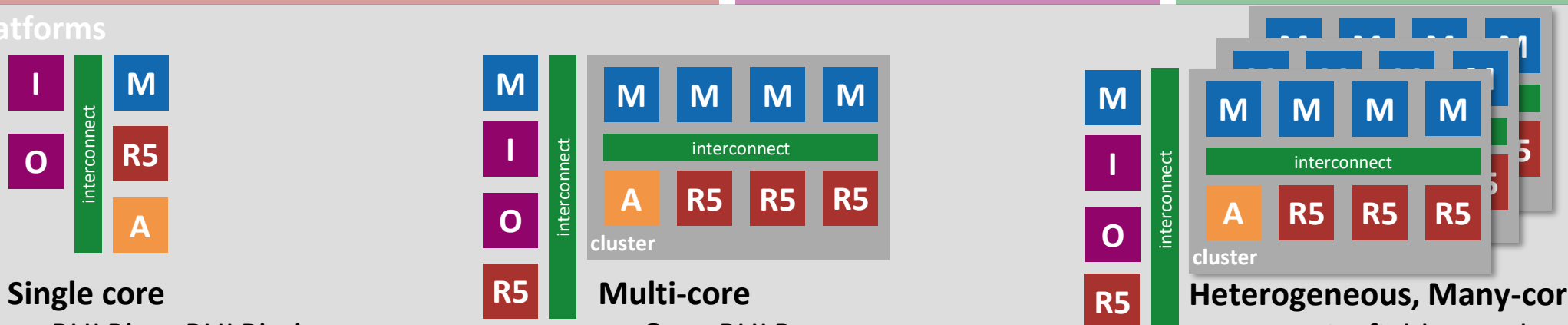
Peripherals

JTAG, SPI, UART, I2S, DMA, GPIO

Interconnects

LIC, HCI, APB, FlooNoC, AXI4

Platforms



Single core
- PULPino, PULPissimo
- Cheshire

Multi-core
- OpenPULP
- ControlPULP

Heterogeneous, Many-core
- Hero, Carfield, Astral
- Occamy, Mempool

IOT → **HPC**

Accelerators and ISA extensions

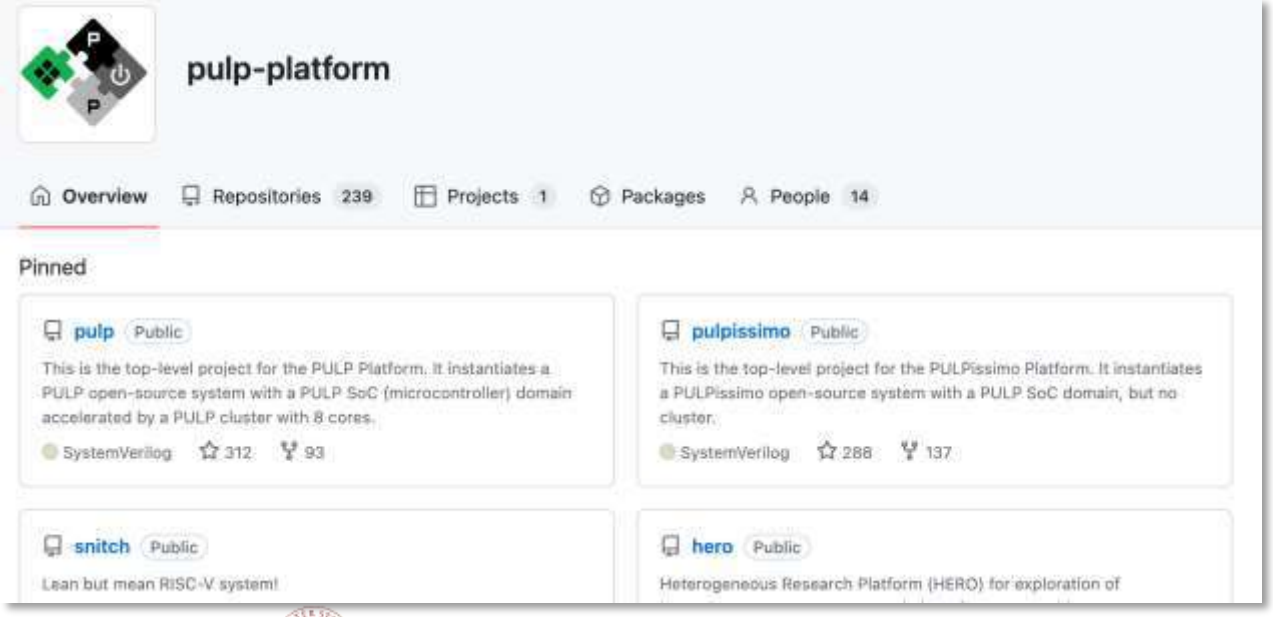
XpulpNN, XpulpTNN	ITA (Transformers)	RBE, NEUREKA (QNNs)	FFT (DSP)	REDMULE (FP-Tensor)

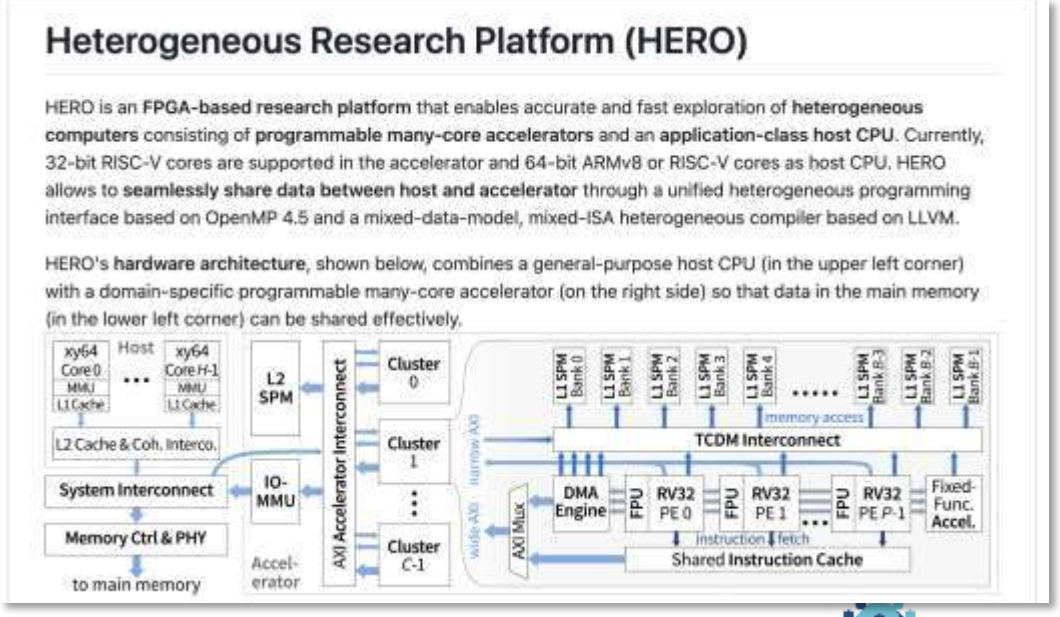
We make everything (we can) available openly

- **All our development is on GitHub using a permissive license**
  - HDL source code, testbenches, software development kit, virtual platform

https://github.com/pulp-platform

  - Allows anyone to use, change, and make products without restrictions.





RISC-V the open ISA

- **Originally developed at UC Berkeley as part of a class**
- **Open ISA managed by RISC-V international since 2015**
  - ETH Zürich is a founding member (currently Frank is in the Board of Directors)
  - Headquarters officially in Zurich
- **Simple Base ISA (RV32 / RV64 / RV128)**
  - Extensions to cover many aspects (vector, matrix..)
- **Open development**
  - Technical working groups where members discuss and propose new extensions
  - Public review and comments, ratified by the Board of Directors
- **Allows processors to be designed and extended easily**
  - While allowing a common SW infrastructure to be built around it.

RISC-V a free ISA to build SoA computer systems

- **It is FREE**
  - Everybody can build, sell, and make RISC-V cores available
  - The description is **FREE**, implementations can be FREE or proprietary
- **It is a modern design, no historical baggage**
  - Some common ISAs (ARM, Intel..) have been around for 20+ years Newer implementations, still need to be compatible to older designs.
  - RISC-V benefited form the mistakes made by others, cleaner design
  - Major design decisions have been properly motivated and explained
- **Reserved space for extensions, modular**
- **Open standard, you can help decide how it is developed**

Are RISC-V processors better than XYZ?

- **Actual performance depends on the implementation**
  - RISC-V does not specify implementation details (on purpose)
- **It is a modern design, should deliver comparable performance**
  - If implemented well, it should perform as good as other modern ISA implementations
  - In our experiments, we see no weaknesses when compared to other ISAs
  - It also is **not magically 2x better**
- **High-end processor performance is not much about ISA**
  - Implementation details like technology capabilities, memory hierarchy, pipelining, and power management are more important.

Exposing Quantization to SW: ISA Extensions

- **RISC-V has Reserved opcodes for standard extensions**
- **Rest of opcodes free for custom implementations**
- **Custom extensions can be standardized**
  - Standard extensions will be frozen/not change in the future

inst[4:2] / inst[6:5]	000	001	010	011	100	101	110	111 ($> 32b$)
00	LOAD	LOAD-FP	*custom-0*	MISC-MEM	OP-IMM	AUIPC	OP-IMM-32	$48b$
01	STORE	STORE-FP	*custom-1*	AMO	OP	LUI	OP-32	$64b$
10	MADD	MSUB	NMSUB	NMADD	OP-FP	*reserved*	*custom-2/rv128*	$48b$
11	BRANCH	JALR	*reserved*	JAL	SYSTEM	*reserved*	*custom-3/rv128*	$\geq 80b$

Extensibility is fundamental in the RISC-V ISA!

Achieving ~100% **dotp** Unit Utilization

8-bit Convolution

HW Loop

LD/ST with post increment

8-bit SIMD sdotp

8-bit sdotp + LD

RV32IMC

N

```
addi   a0,a0,1
addi   t1,t1,1
addi   t3,t3,1
addi   t4,t4,1
lbu    a7,-1(a0)
lbu    a6,-1(t4)
lbu    a5,-1(t3)
lbu    t5,-1(t1)
mul    s1,a7,a6
mul    a7,a7,a5
add    s0,s0,s1
mul    a6,a6,t5
add    t0,t0,a7
mul    a5,a5,t5
add    t2,t2,a6
add    t6,t6,a5
bne    s5,a0,1c000bc
```

RV32IMCXpulp

N/4

```
lp.setup
p.lw  w1, 4(a0!)
p.lw  w2, 4(a1!)
p.lw  x1, 4(a2!)
p.lw  x2, 4(a3!)
pv.sdotsp.b   s1, w1, x1
pv.sdotsp.b   s2, w1, x2
pv.sdotsp.b   s3, w2, x1
pv.sdotsp.b   s4, w2, x2
end
```

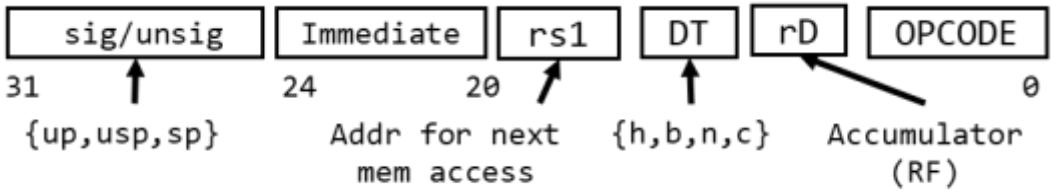
can we remove?

Yes! dotp+ld

N/4

```
Init NN-RF (outside of the loop)
lp.setup
pv.nnsdotup.h  s0,ax1,9
pv.nnsdotsp.b  s1, aw2, 0
pv.nnsdotsp.b  s2, aw4, 2
pv.nnsdotsp.b  s3, aw3, 4
pv.nnsdotsp.b  s4, ax1, 14
end
```

pv.nnsdot{up,usp,sp}.{h,b,n,c} rD, rs1, Imm

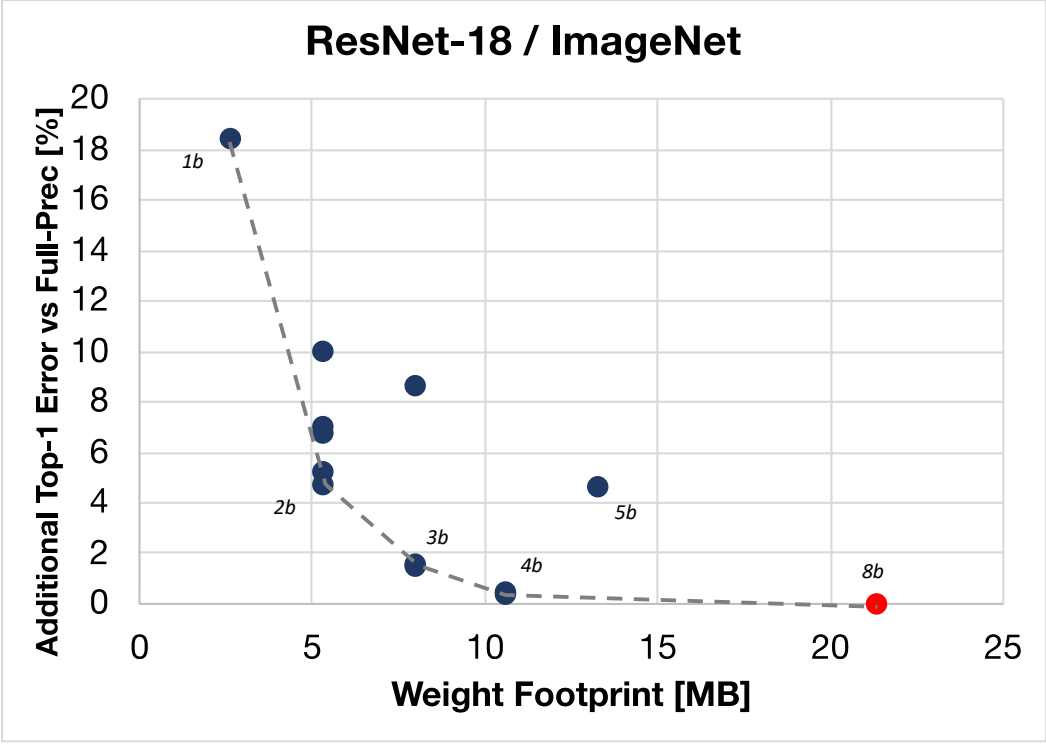


9x less instructions than RV32IMC

14.5x less instructions at an extra 3% area cost (~600GEs)

Next: Sub-byte precision

SoA Quantization Results



Quantizazion of a MobilenetV1_224_1.0 (*)

Quantization Method	Top1 Accuracy		Weight Memory Footprint	
Full-Precision	70.9%		16.27 MB	
INT-8	70.1%	**0.8%**	4.06 MB	**4x**
INT-4	66.46%	**4.4%**	2.35 MB	**7x**
Mixed-Precision	68%	**2.9%**	2.09 MB	**8x**

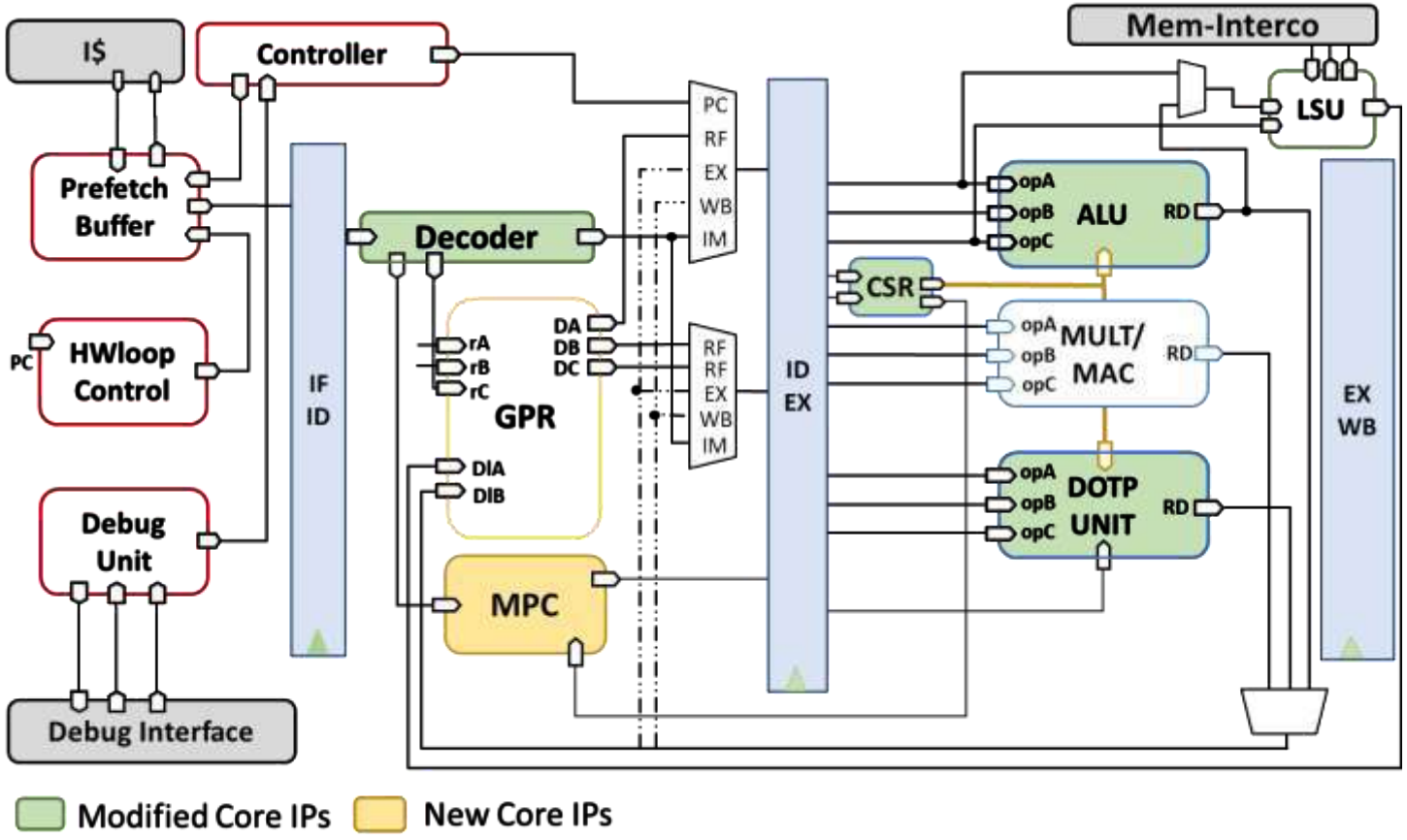
Courtesy of Rusci M. «Example on MobilenetV1_224_1.0.»C

Mixed-precision approach key to meet the memory constraints of tiny devices

Quantized Neural Networks (QNNs) are a natural target for execution on constrained extreme edge platforms.

(*) Rusci M. et al., Memory-Driven Mixed Low Precision Quantization For Enabling Deep Network Inference On Microcontrollers. . arXiv preprint arXiv:1905.13082.

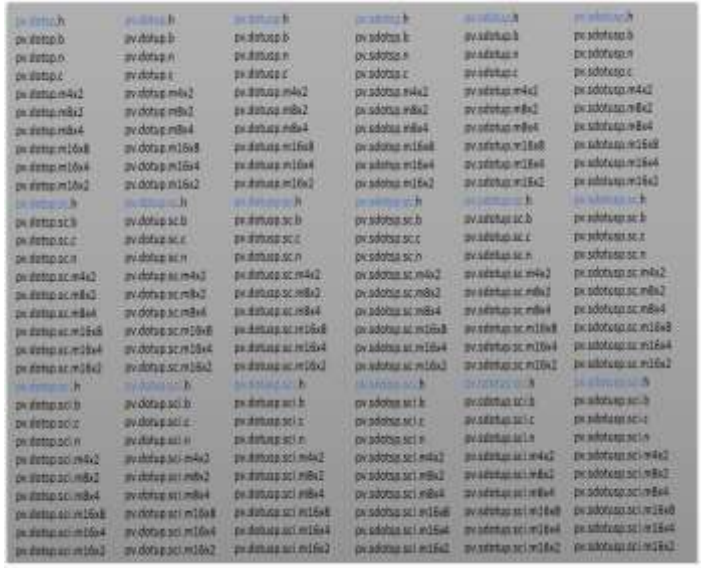
Processor HW Extension



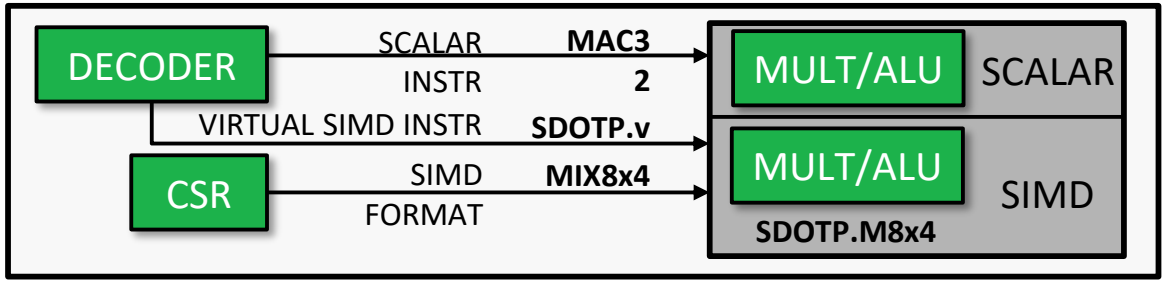
- **Goal**
  - HW support for mixed-precision SIMD instructions;
- **Challenge**
  - Enormous number of instructions to be encoded in the ISA;
- **Solution**
  - Status-based execution.

Virtual SIMD Instructions

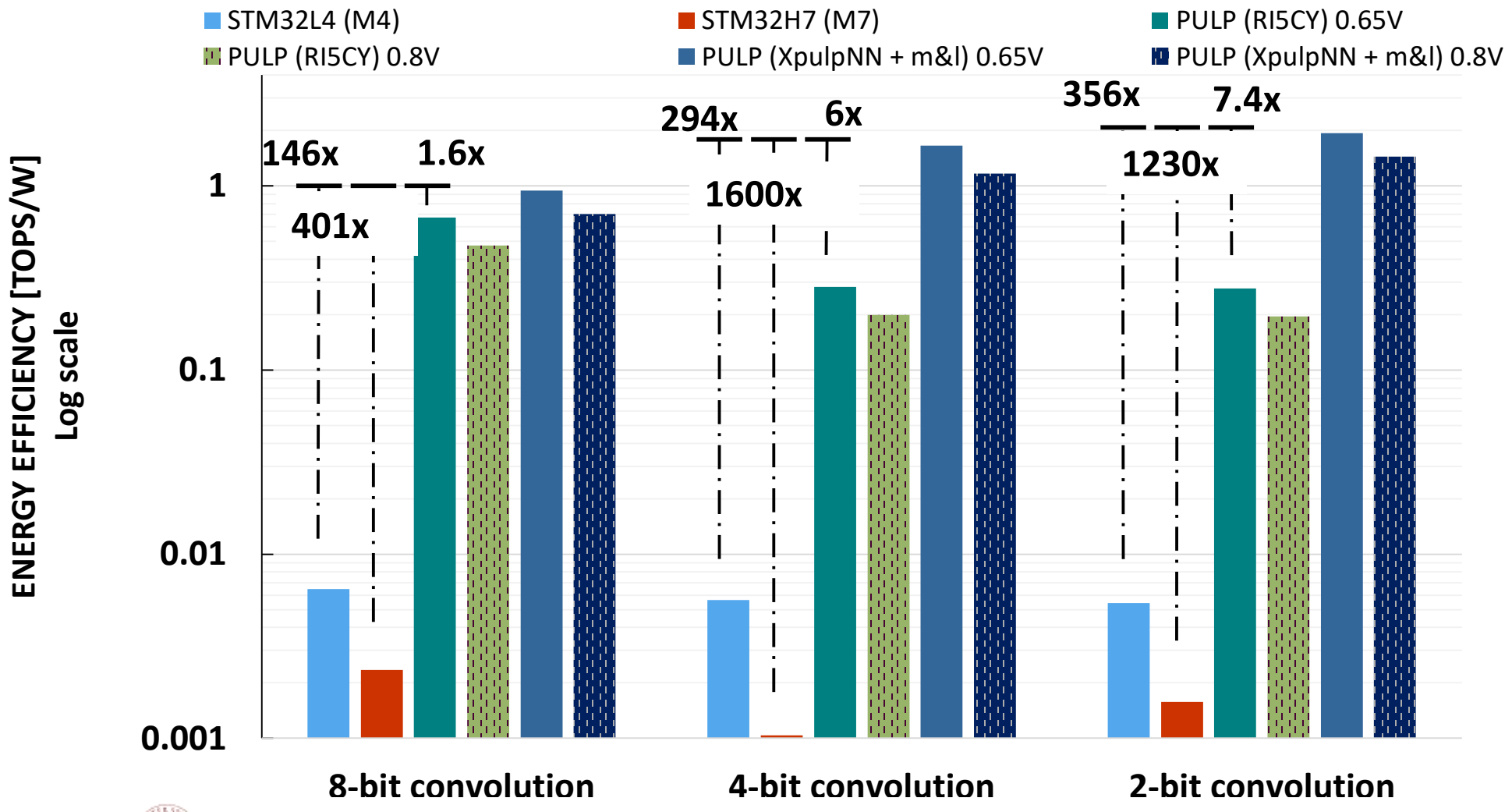
- **Encode operation as a virtual SIMD in the ISA (e.g. sdotsp.v)**
- **Format specified at runtime by a Control Register (e.g. 4x4)**
- **180→18 Instructions needed for SIMD DOTP**
- **Potential to avoid code replication for different formats**
- **Tiny Overhead on QNN for Switching format**
  - Format switch not frequent in DNN, e.g. every layer.



pv.dotsp.v	pv.sdotsp .v
pv.dotsp.sc .v	pv.sdotsp.sc .v
pv.dotsp.sci .v	pv.sdotsp.sci .v
pv.dotup .v	pv.sdotup .v
pv.dotup.sc .v	pv.sdotup.sc .v
pv.dotup.sci .v	pv.sdotup.sci .v
pv.dotusp .v	pv.sdotusp .v
pv.dotusp.sc .v	pv.sdotusp.sc .v
pv.dotusp.sci .v	pv.sdotusp.sci .v



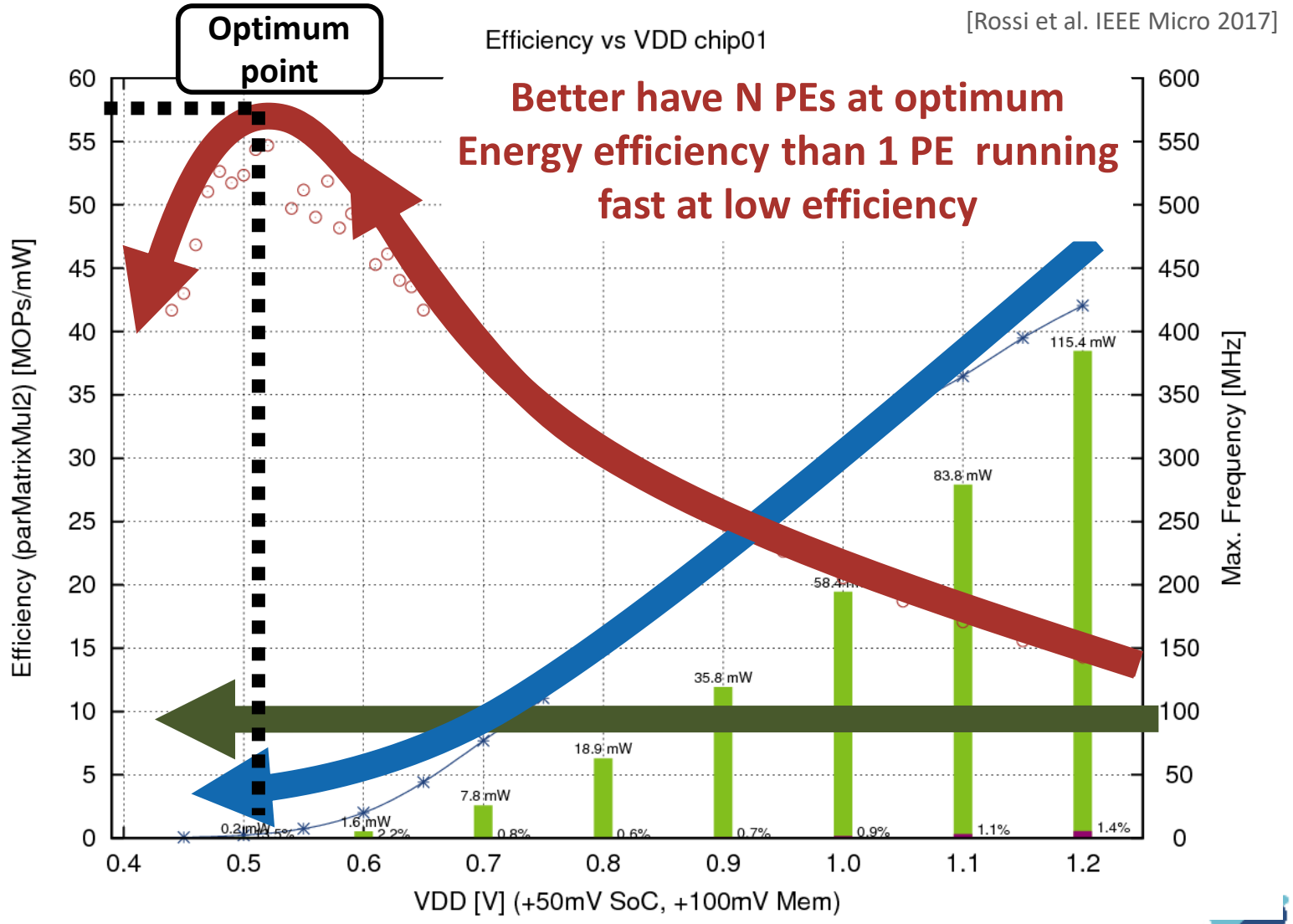
8-Cores x Cluster + XpulpNN + M&L (22nm)



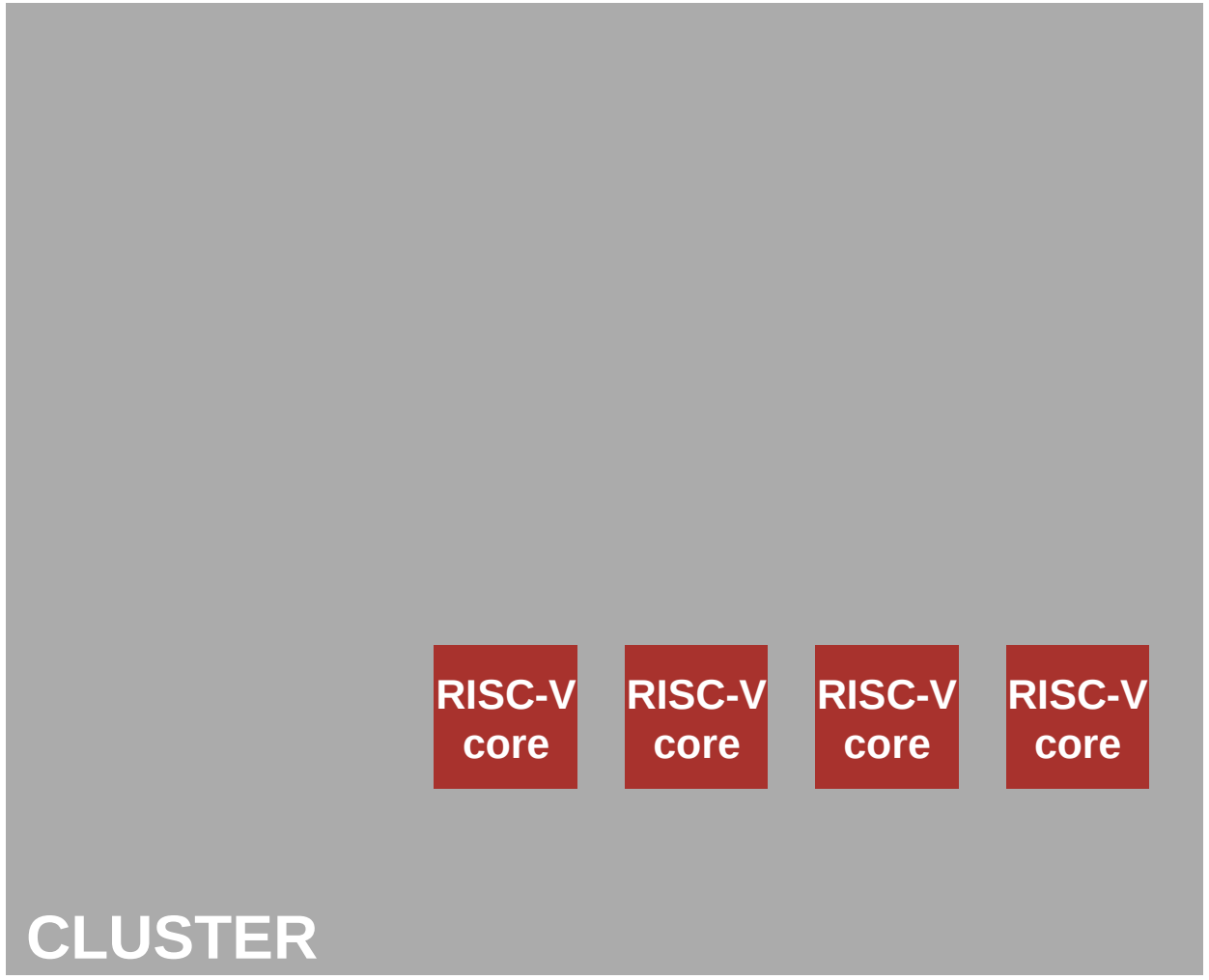
Parallel, Ultra-Low Power (PULP) PE Cluster

- As VDD decreases, operating speed decreases
- However efficiency increases→ more work done per Joule
- Run parallel to get performance and efficiency!

AI is parallel and scales
More parallel with NN size

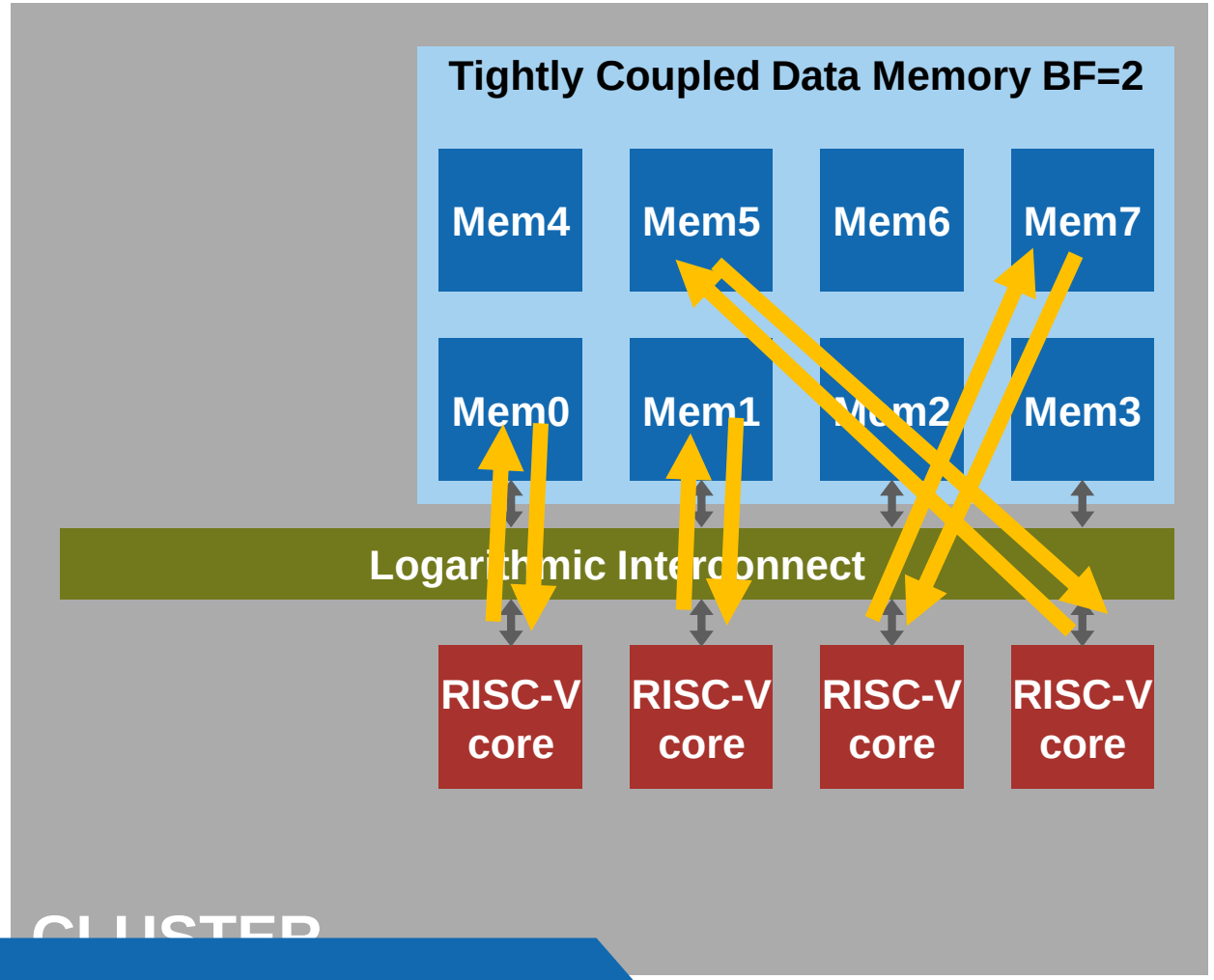


Let's design a cluster with multiple (4-16) RISC-V cores



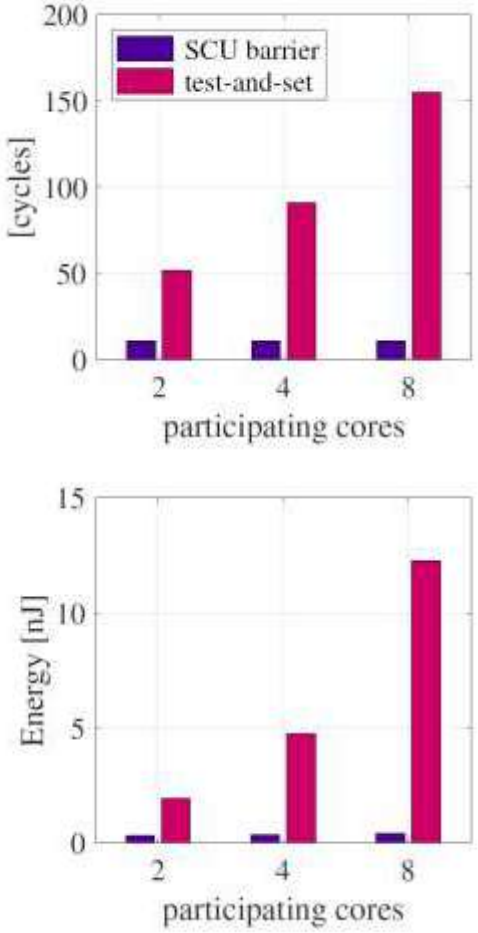
Low-Latency shared Tightly Coupled Data Memory

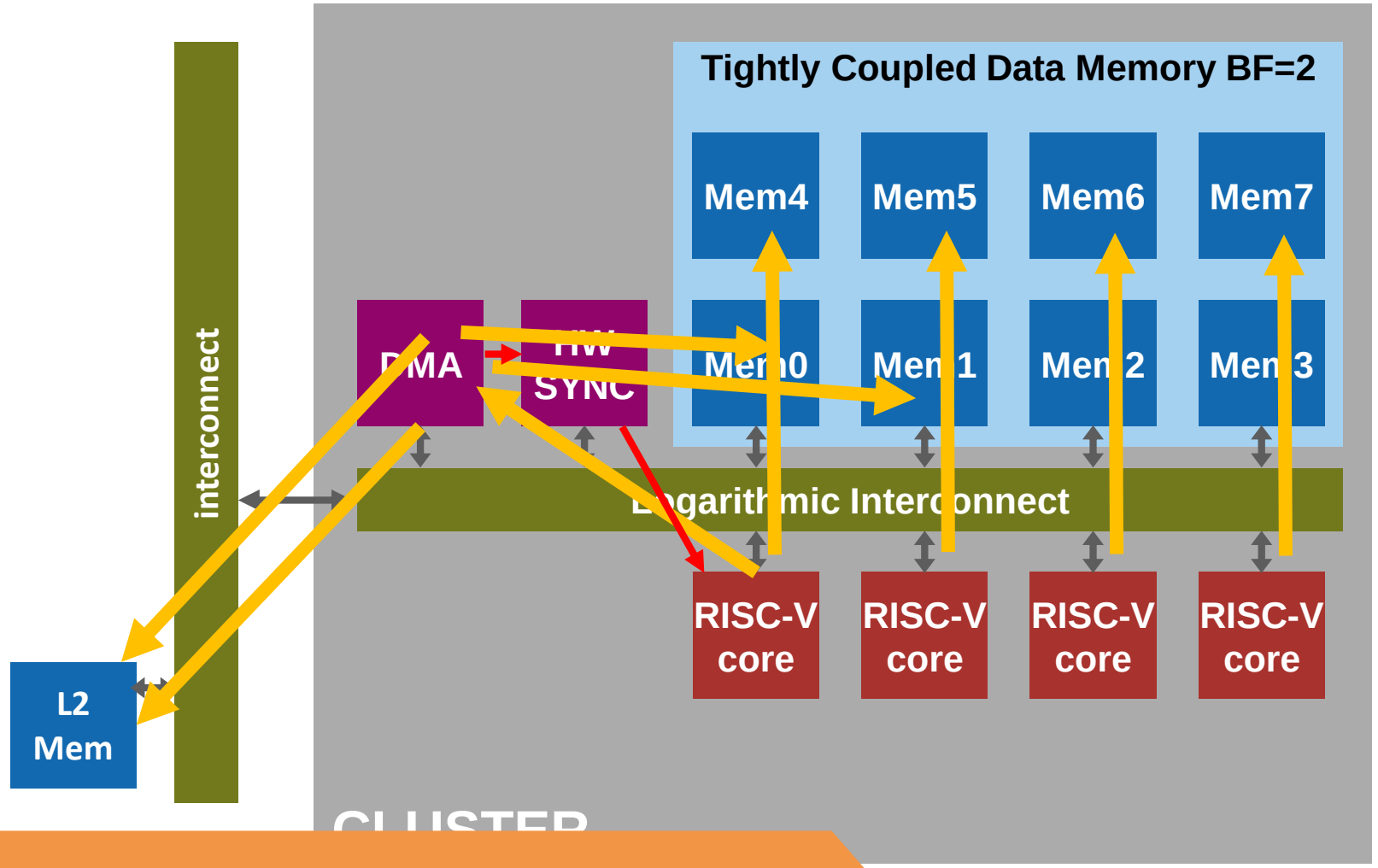
- Parallel memory access with low contention
  - Multi-banked, address-interleaved L1
- Fast interconnect with physical design awareness
  - Logarithmic depth of combinational switchboxes



Trade-off between memory size and latency

DMA based, non-blocking mem copy with fast sync.



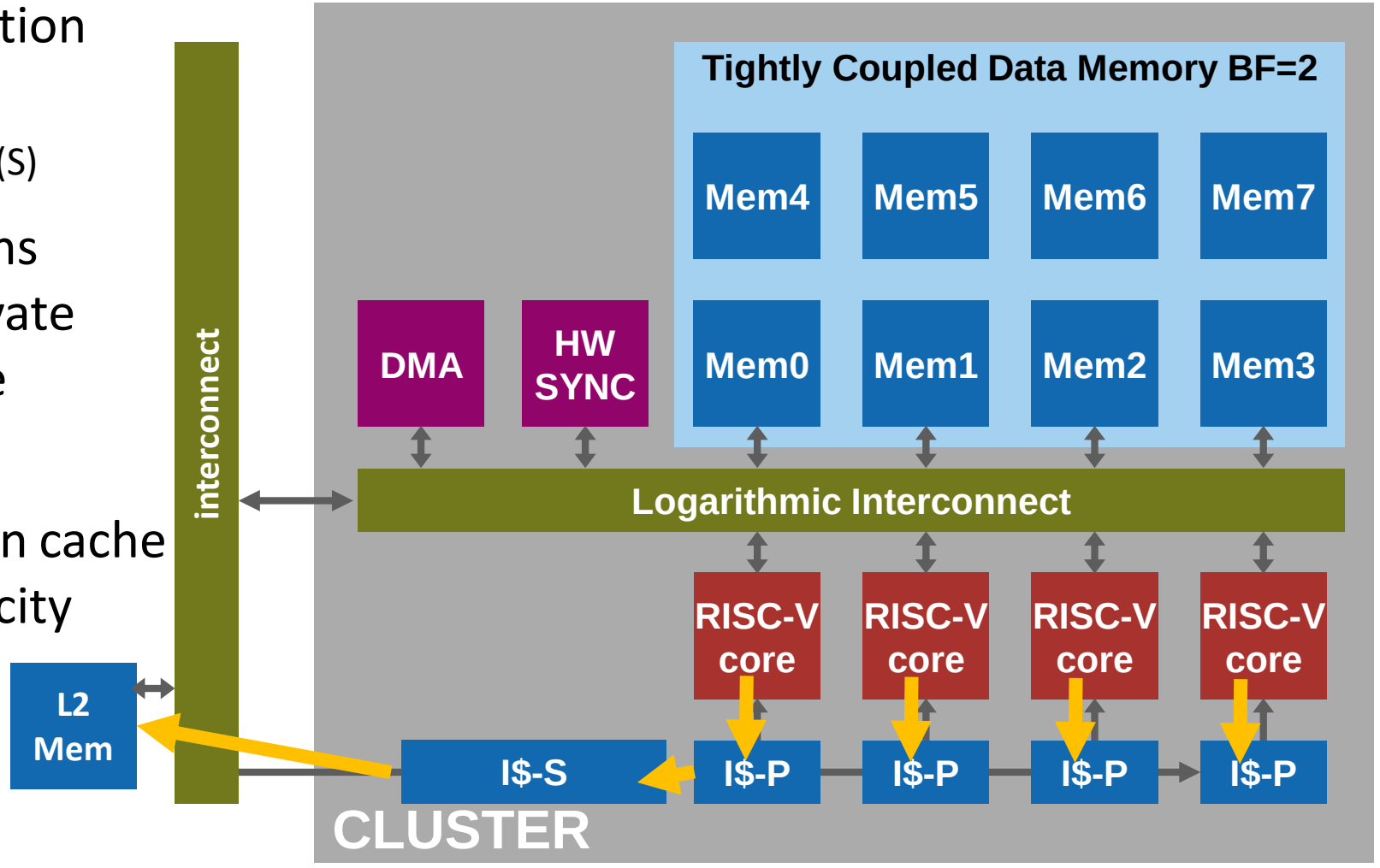


~15x latency and energy reduction for a barrier

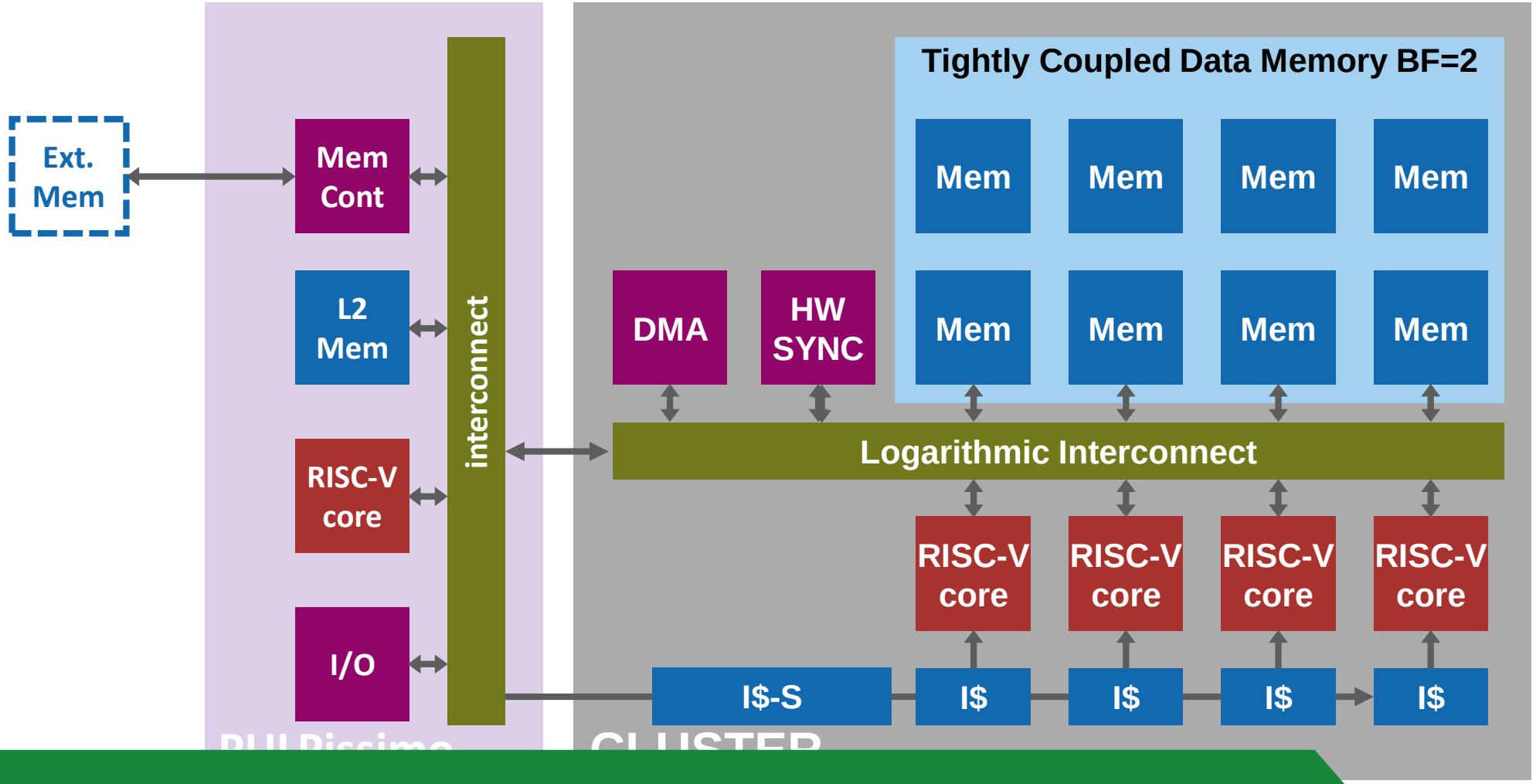
[Glaser TPDS20]

Shared instruction cache with private “loop buffer”

- Two-level instruction cache
  - Private (P) + Shared (S)
- Most instrunctions fetched form Private Instruction Cache
  - Low fetch energy
- Shared instruction cache to augment capacity
  - Reduces miss latency



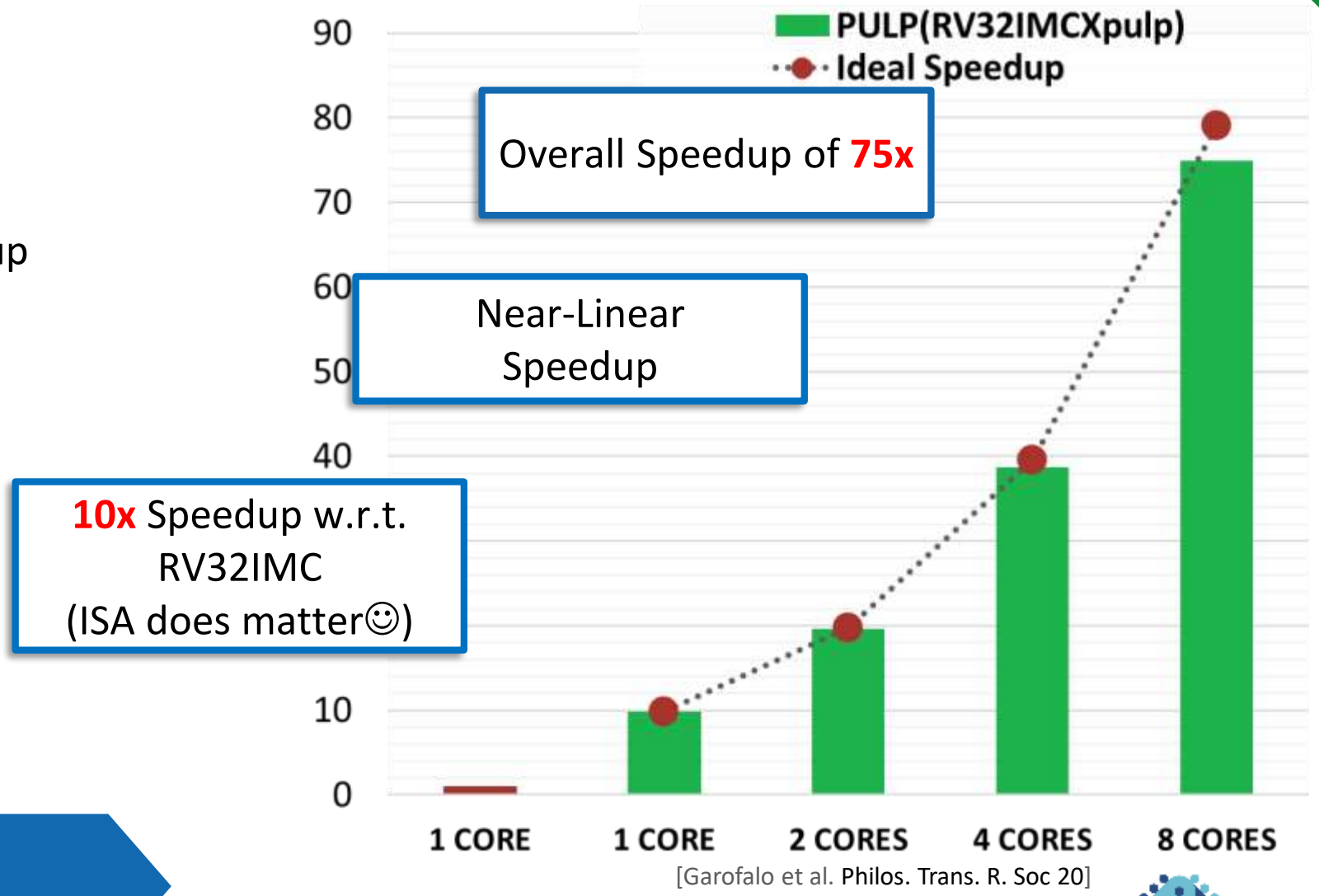
Host for sequential, I/O + Data-Parallel Accelerator Cluster



https://github.com/pulp-platform/pulp

Combining ISA extension + Efficient parallel execution

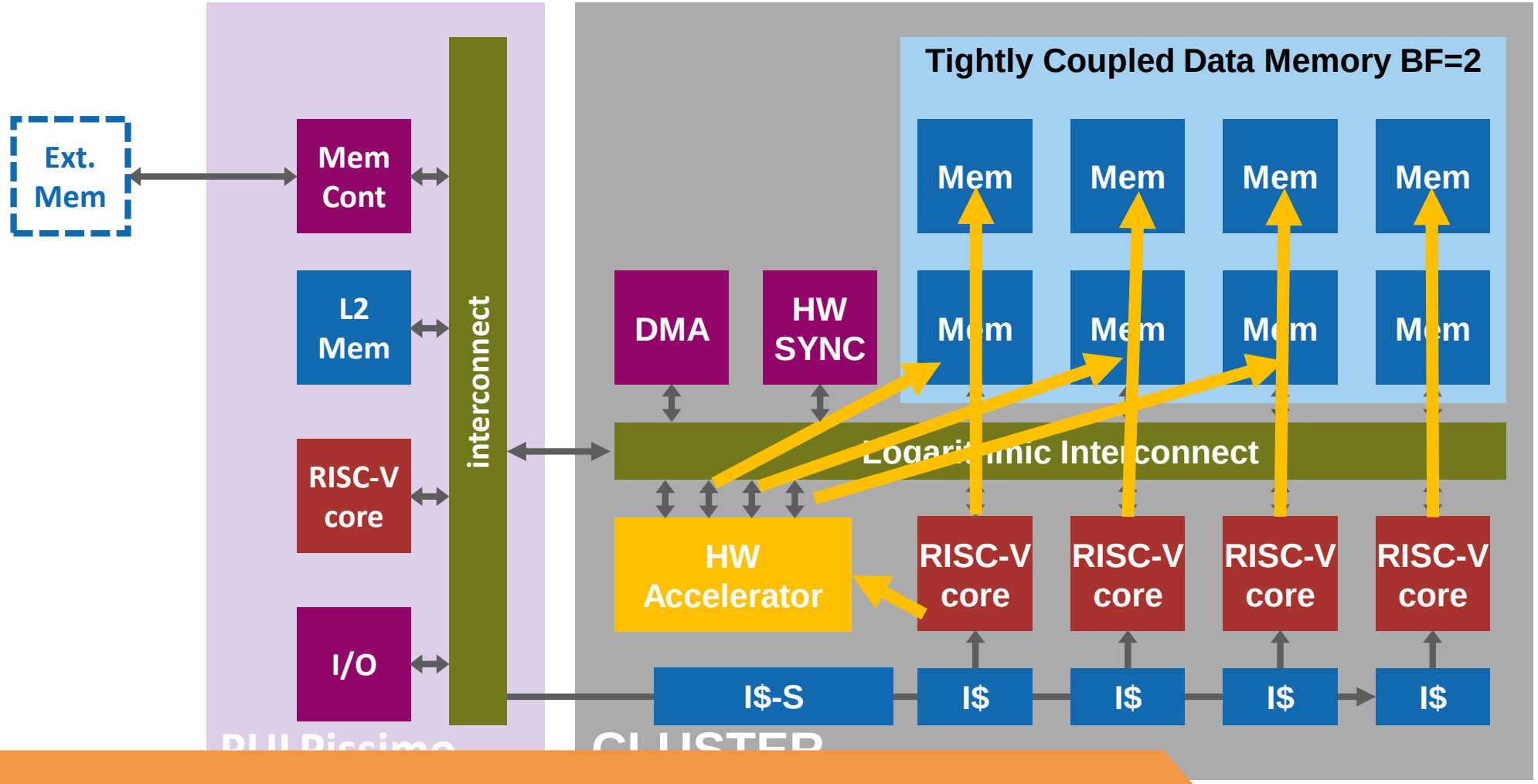
- **8-bit convolution**
  - Open source DNN library
- **10x through xPULP**
  - Extensions bring real speedup
- **Near-linear speedup**
  - Scales well for regular workloads
- **75x overall gain**
- **7-8 GMACs**
  - 250MHz
  - 4 MAC/Cycle (8bit)



[Garofalo et al. Philos. Trans. R. Soc 20]

More GOPS, less power

What's next? Tightly-coupled accelerators

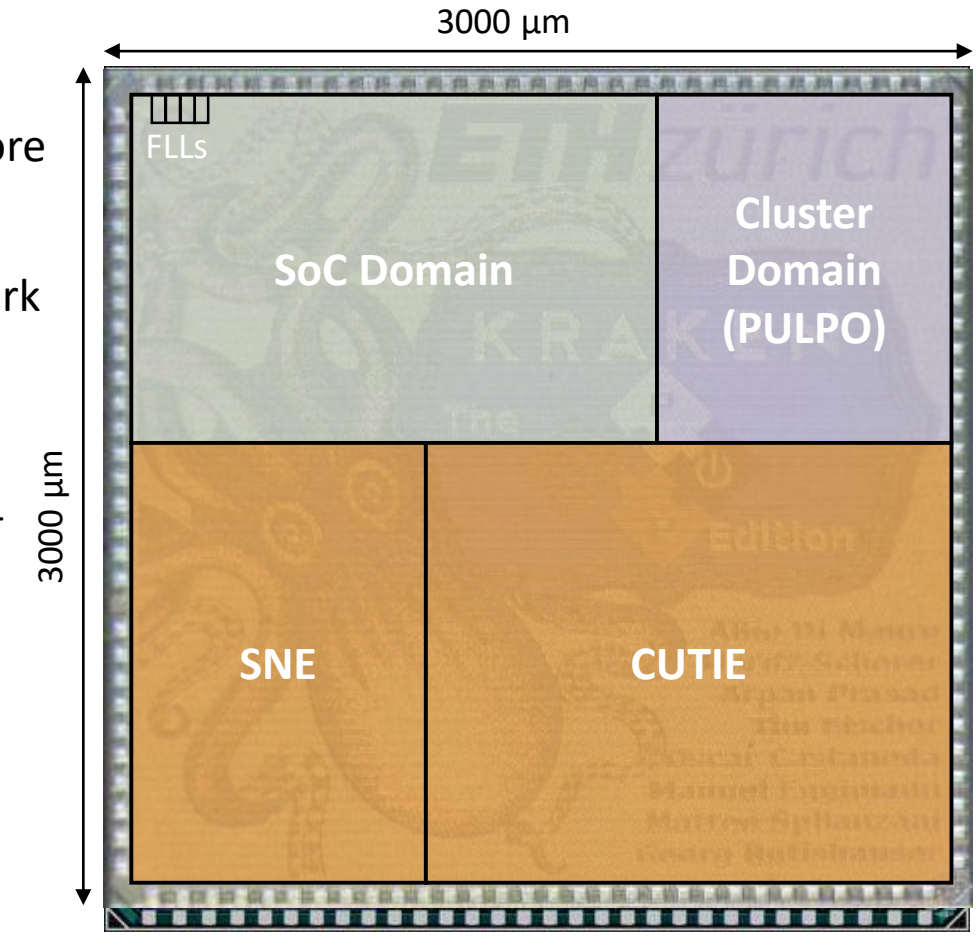


Acceleration with flexibility: zero-copy HW-SW cooperation

Kraken: 22FDX SoC, Multiple Heterogeneous Accelerators

The ***Kraken***: an "Extreme Edge" Brain

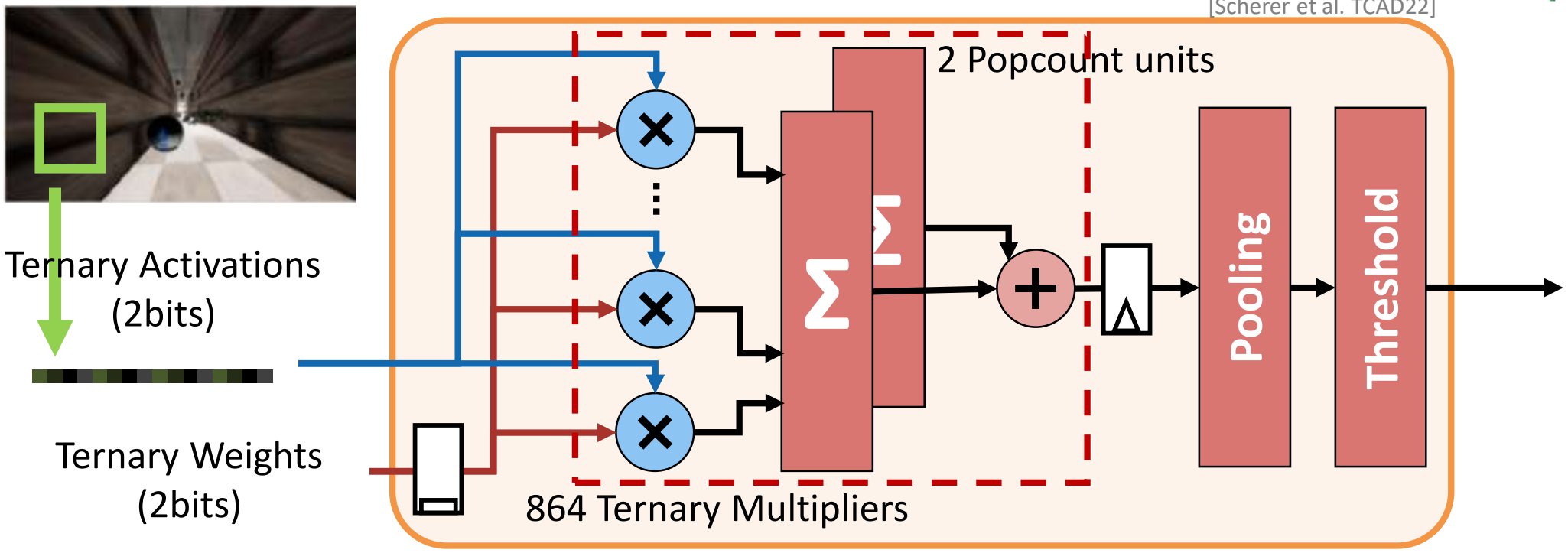
- **RISC-V Cluster**
  8 Compute cores +1 DMA core
- **CUTIE**
  Dense ternary-neural-network accelerator
- **SNE**
  Energy-proportional spiking-neural-network accelerator



Technology	22 nm FDSOI
Chip Area	9 mm^2
SRAM SoC	1 MiB
SRAM Cluster	128 KiB
VDD range	0.55 V - **0.8 V**
Cluster Freq	**~370 MHz**
SNE Freq	**~250 MHz**
CUTIE Freq	**~140 MHz**

CUTIE: Perception from Frame Sensors

[Scherer et al. TCAD22]



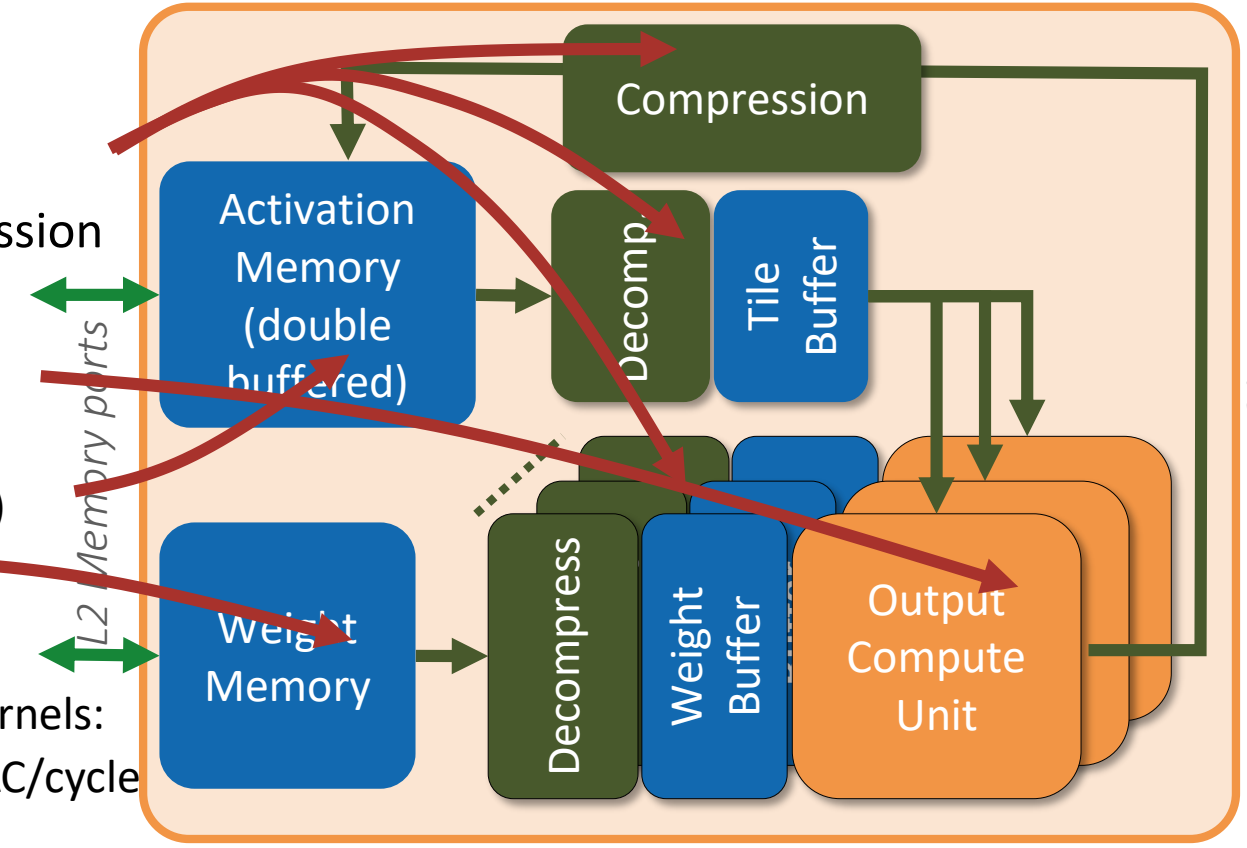
Output channel compute unit (OCU)

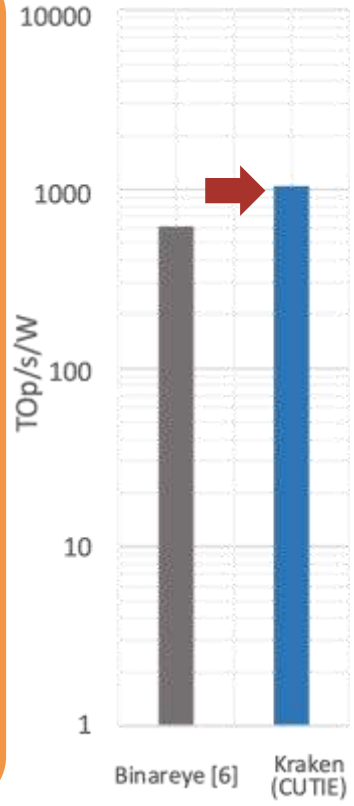
- **Completely Unrolled Ternary Neural Inference Engine**: K × K window, all input channels, cycle-by-cycle sliding
- One *Output Compute Unit* (OCU) computes one output activation per cycle!
- Zeros in weights and activations, spatial smoothness of activations reduce switching activity

Aggressive quantization and full specialization

Kraken`s CUTIE Implementation

- Data in 1.6 bits (Ternary value) with On-the-fly Compression/Decompression
- Configuration in Kraken
  - 96 channels (Output compute units)
  - 3 × 3 kernels
  - 64 × 64 pixels feature maps (158 KiB)
  - 9 layers of weights (117 KiB)
- Lots of TMAC/cycle
  - 96 OCUs, 96 Input channels, 3 × 3 kernels:
  - 96 × 96 × 3 × 3 = 82'944 Ternary-MAC/cycle

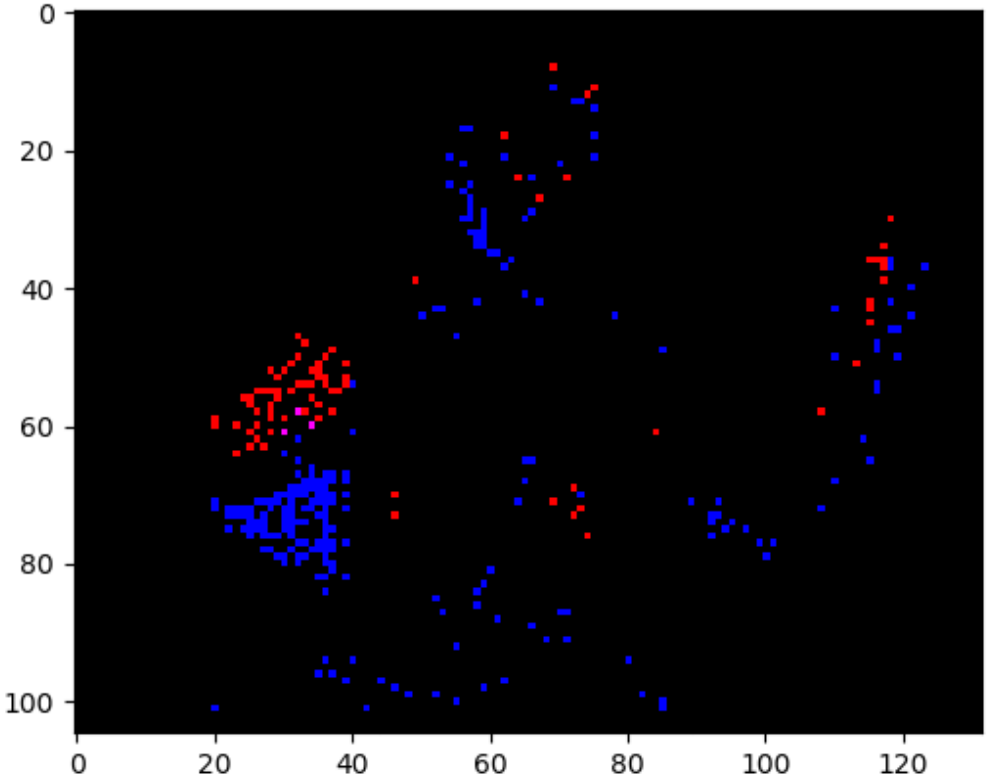


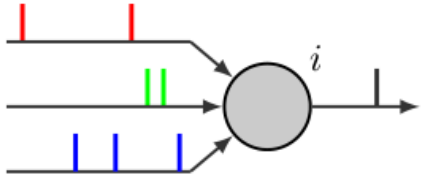


1fJ/MAC (1POP/s/W)
Ternary OPS

SNE: Perception on Event Sensors

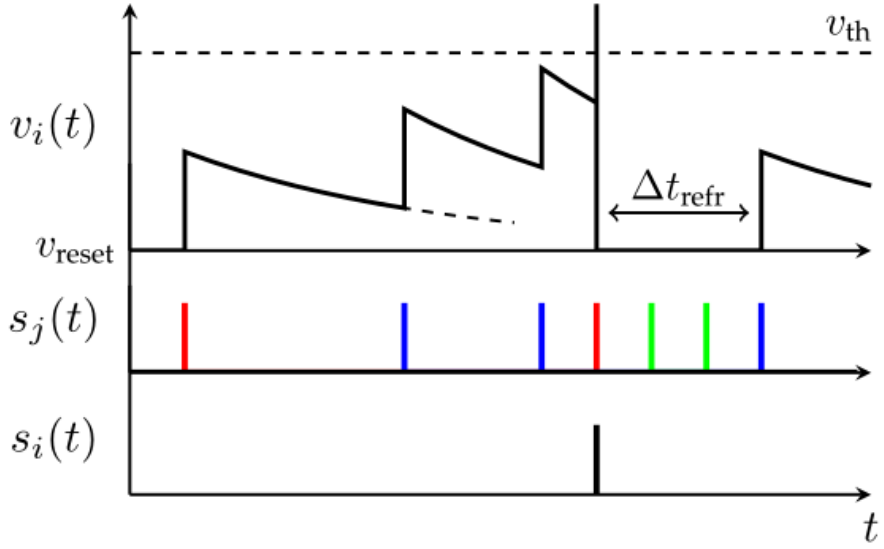
Event Sensors – DVS camera
Ultra-low latency
Energy- proportional interface





Spiking Neural Engine (SNE)

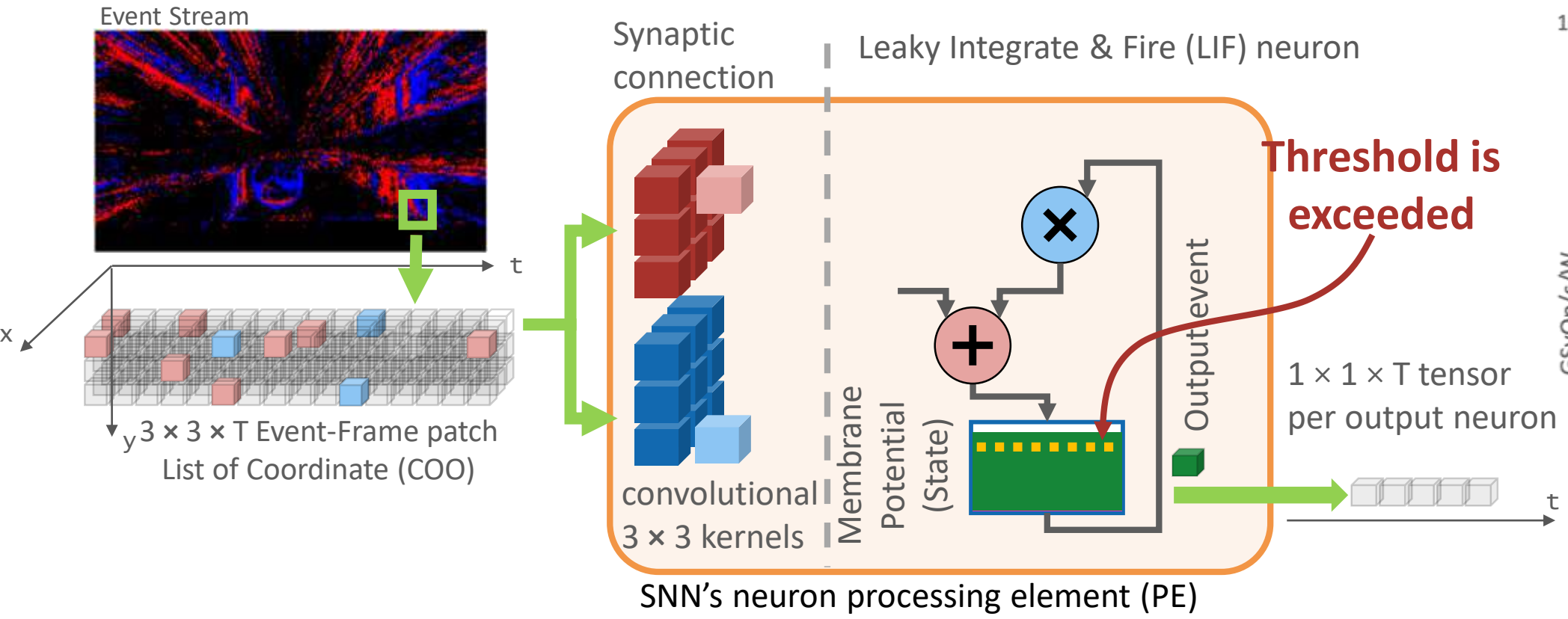
Leaky Integrate & Fire (LIF) neurons

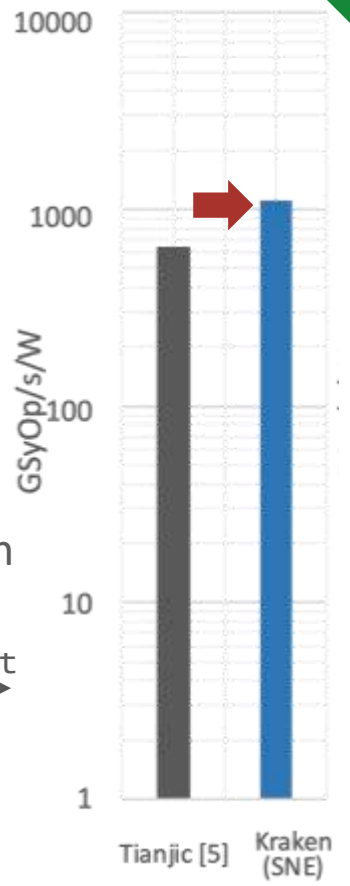


[Di Mauro et al. DATE22]

SNE works seamlessly with DVS (event-based) sensors

Event consumption, and output spikes generation





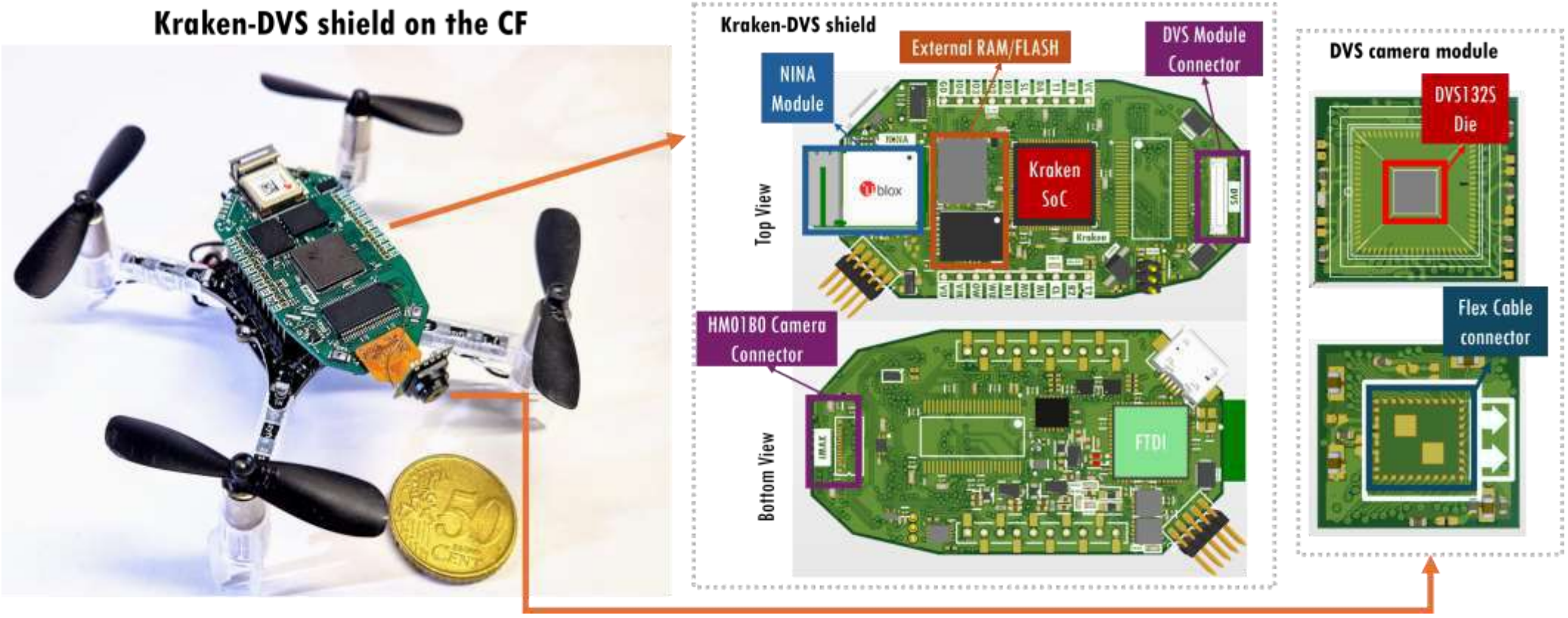
1TSyOp/s/W

A more complex dynamic than conventional DNNs neurons:

- Membrane Potential Accumulation/Activation 1× SynAcc = 1× 4b-ADD + 1× 8b-COMPARE
- Membrane Potential decay 1× SynDec = (1× 8b-MUL) + (1× 8b-MUL + 1× 8b-ADD)

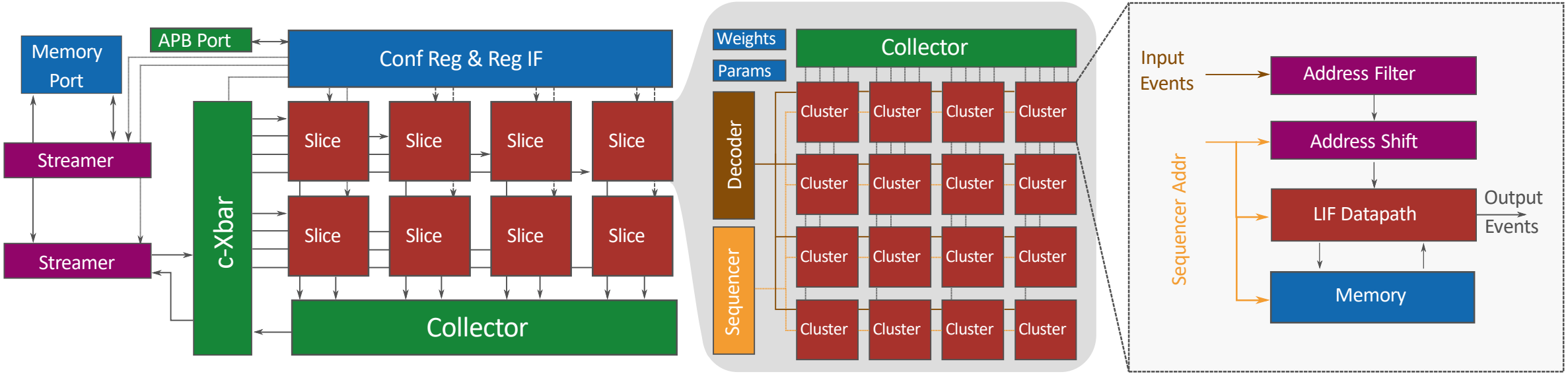
Kraken Shield and System Architecture

- 7g payload
- DVS and frame-based cameras → real-time multi-modal perception.
- Designed for integration into nano-UAV platforms



Spiking Neural Networks for Depth Estimation

SNN → SCNNs for depth estimation.



Depth Estimation

1.02k inferences/s

Energy Efficiency

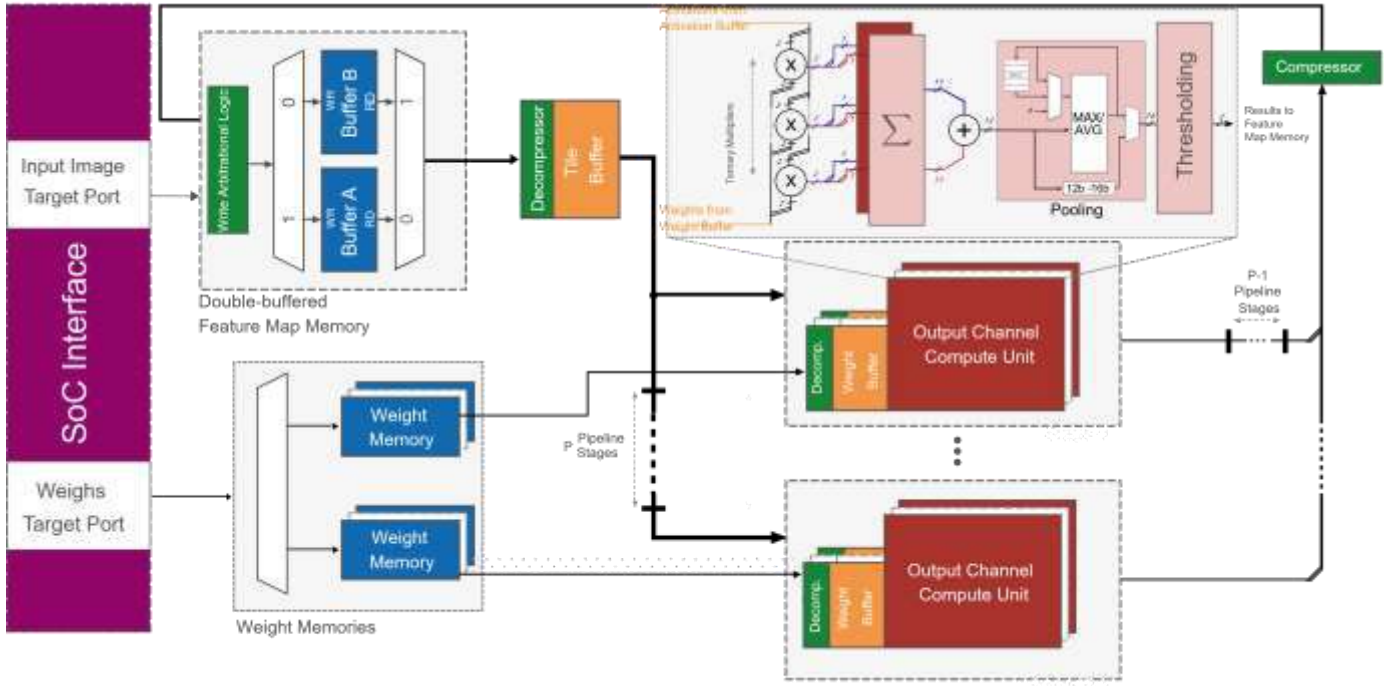
18 µJ per inference

Low Power

98mW @ (220MHz, 0.8V)

Ternary Neural Networks for Object Classification

CUTIE → TNN for object classification.



Object Classification

10k inferences/s

Energy Efficiency

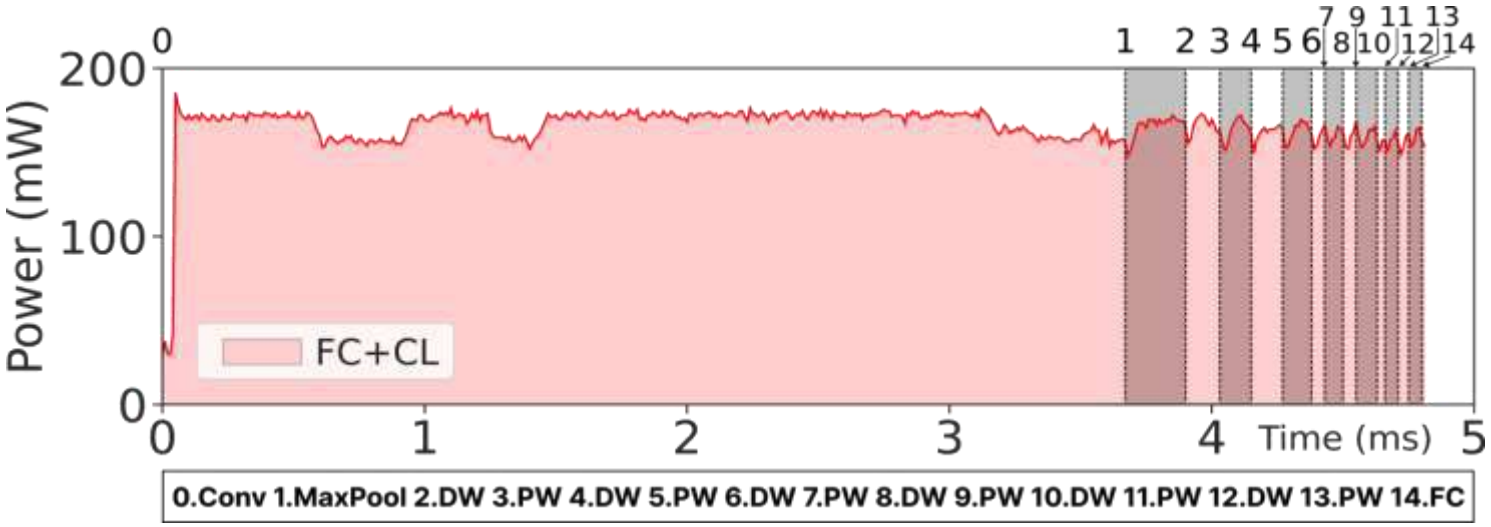
6 µJ per inference

Low Power

110mW @ (330MHz, 0.8V)

Kraken Power Consumption (all Included)

Combined power consumption of SNE, CUTIE, PULP cluster

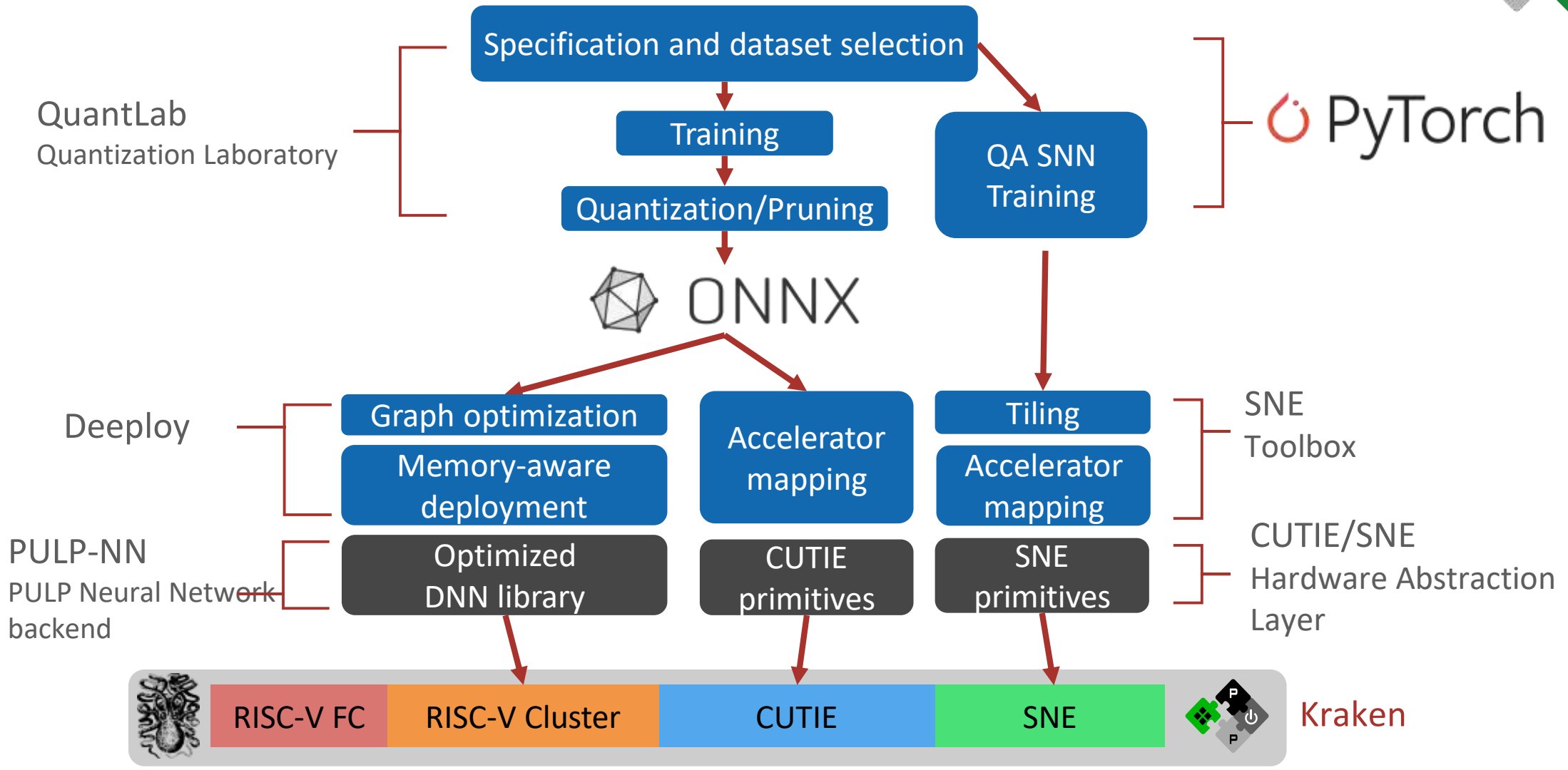


Model	Inference/s	µJ/inf	Power (mW)
SNE	1.02k	18	98
CUTIE	10k	6	110
PULP	221	750	165

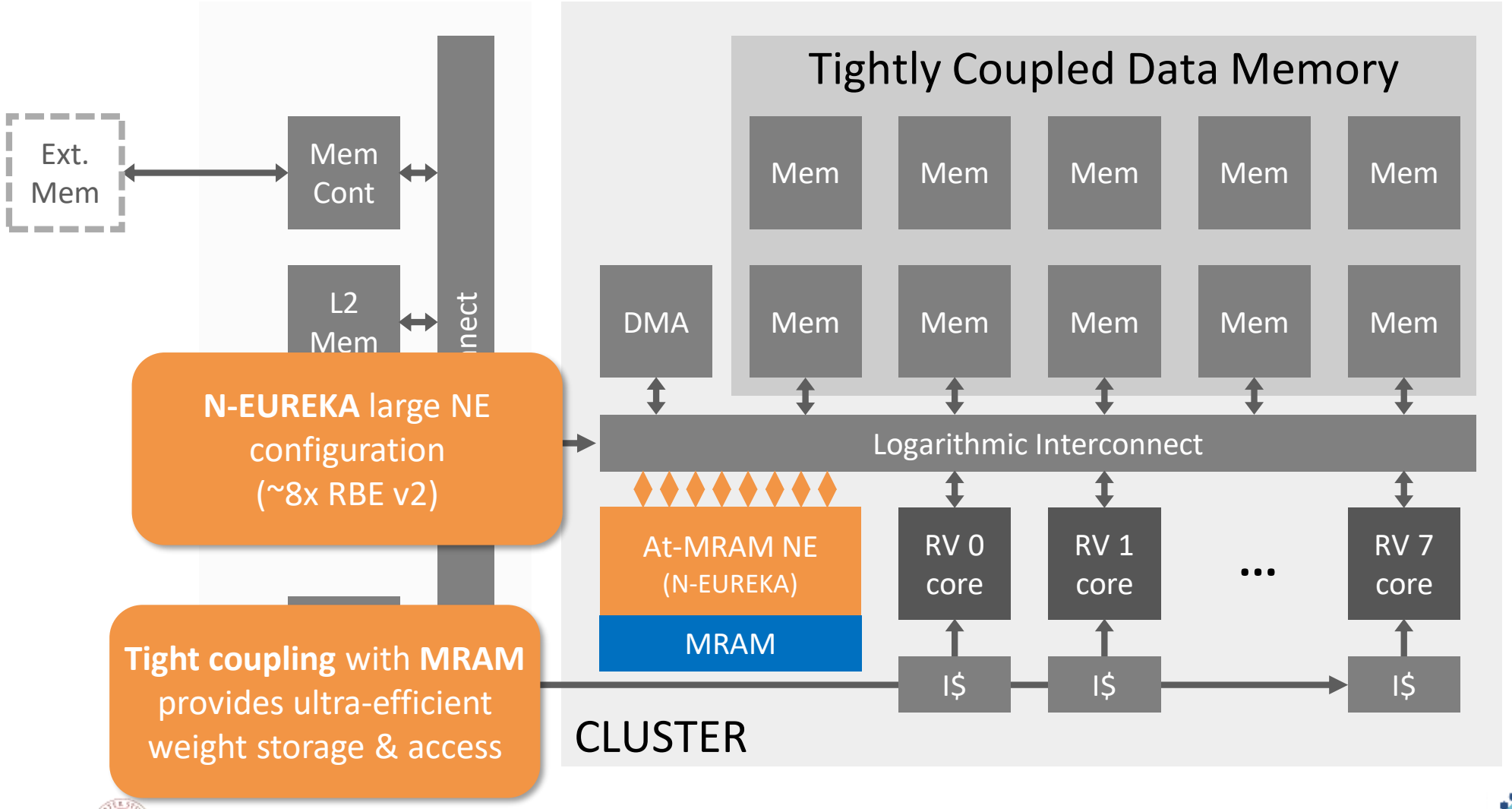
Kraken power waveform executing Tiny-PULP-Dronet at FC@280 MHz, CL@300 MHz, Vdd@0.8 V

P=373mW, representing just 5% of the UAV's power budget

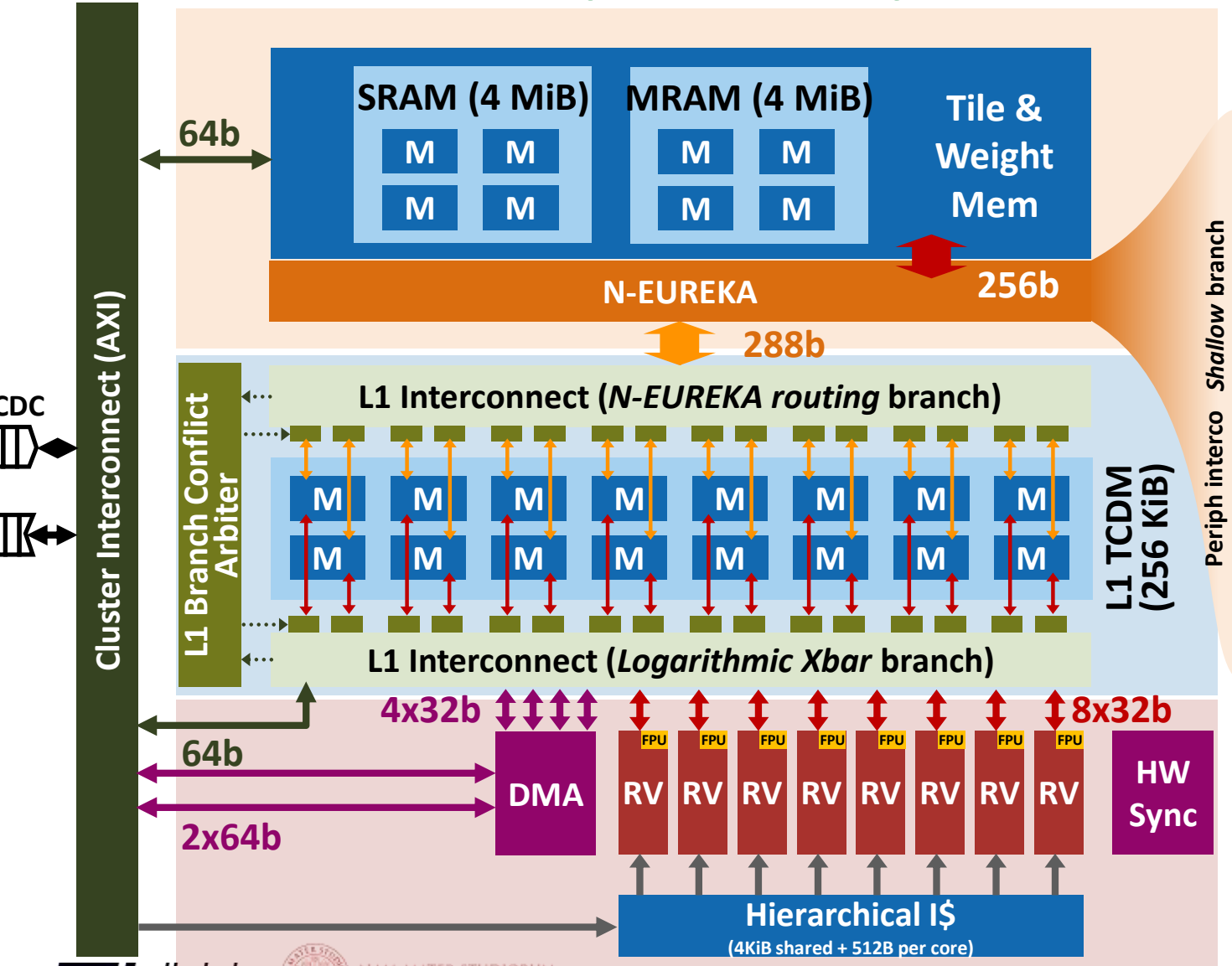
How to deploy applications to PULP/Kraken?

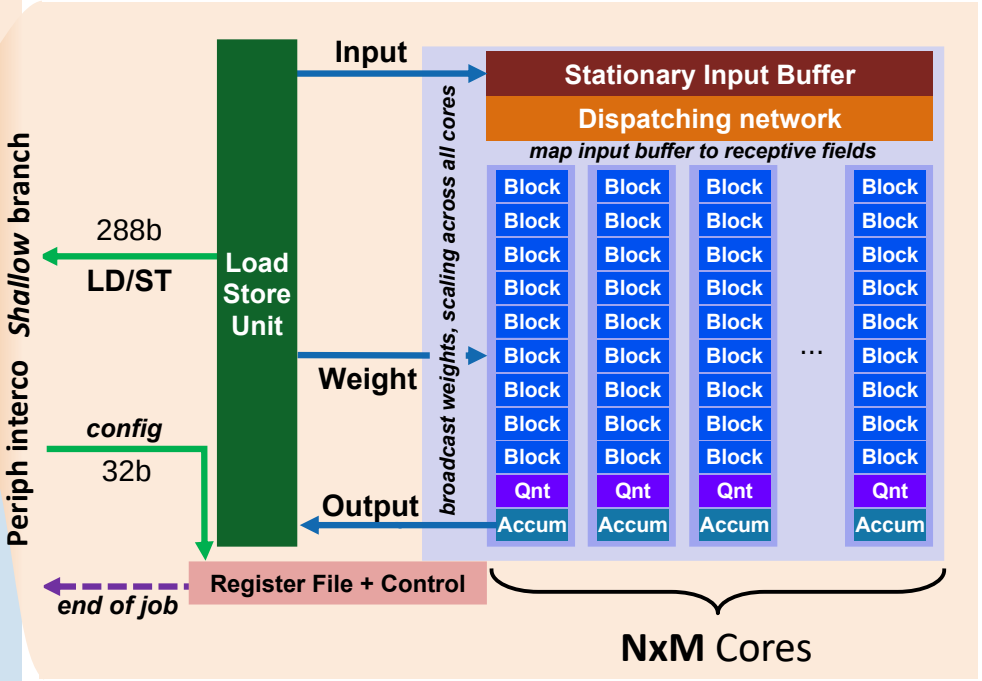


Siracusa: Higher performance cluster with N-Eureka



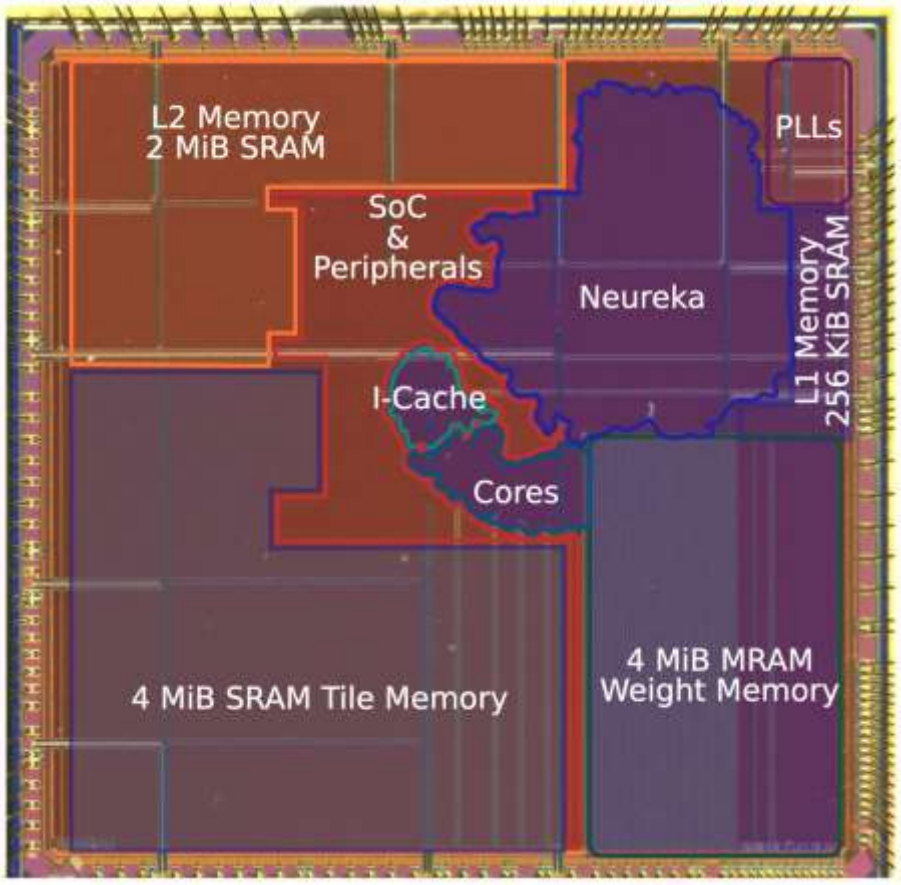
Siracusa: Memory Hierarchy and Dataflows





1 Core = receptive field of 1×1 pixel in/output across 32 out-channels
Output stationary, Input quasi-stationary
Parametric number of Cores (N×M out-pixel)
8b activations, 2-8b weights

Siracusa: 16nm SoC, Tightly Coupled at MRAM Accelerator



	Vega [1]	Diana [2]	Marsellus [3]	[4]	[5]	*Siracusa*
Technology	22nm FDX	22nm FDX	22nm FDX	40nm	22nm	16nm FinFET
Area	10mm²	10.24mm²	8.7mm²	25mm²	8.76mm²	16mm²
On-chip mem	1728 KB SRAM 4 MB MRAM (L3)	896 KB SRAM	1152 KB SRAM	768 KB	1428 KB	6400 KB SRAM 4 MB MRAM (L1)
Peak Perf 8b	32.2 GOPS	140 GOPS	90 GOPS	N/A	146 GOPS	**698 GOPS**
Peak Eff 8b	1.3 TOPS/W	2.07 TOPS/W	1.8 TOPS/W	0.94 TOPS/W	0.7 TOPS/W	**2.68 TOPS/W**
Peak Eff (WxAb)	1.3 TOPS/W	4.1TOPS/W (2x2b) **600 TOPS/W (analog)**	12.4 TOPS/W (2x2b)	60.6 TOPS/W (1x1b)	0.7 TOPS/W	8.84 TOPS/W (2x8b)
Area Eff	3.2 GOPS/mm²	21.2 GOPS/mm²	47.4 GOPS/mm²	N/A	58.3 GOPS/mm²	**65.2 GOPS/mm²**

[1] D. Rossi et al., JSSC'21
[2] P. Houshmand et al., JSSC'23
[3] F. Conti et al., JSSC'23
[4] M. Chang et al., ISSCC'22
[5] Q. Zhang et al., VLSI Symposium'22

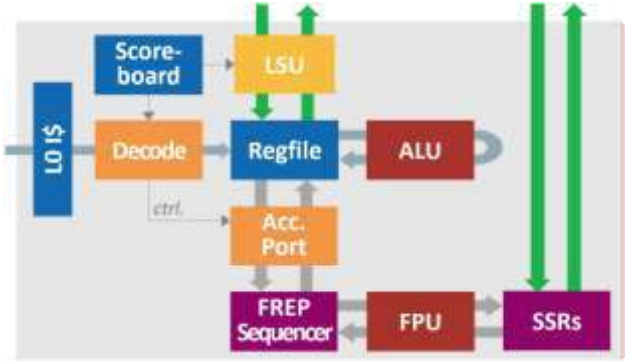
Balance efficiency, peak performance, area efficiency without compromises in precision

N-EUREKA 36-cores configuration

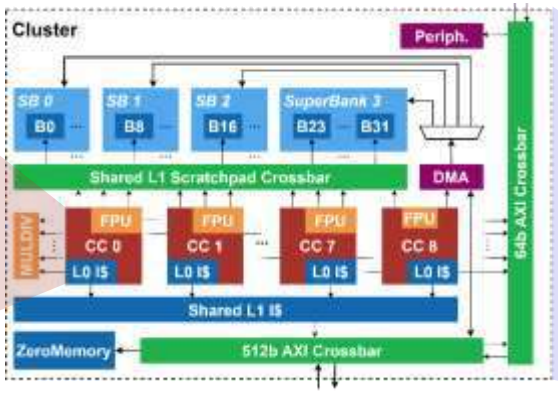
[A. Prasad et al., "Siracusa: a 16nm Heterogeneous RISC-V SoC for Extended Reality with At-MRAM Neural Engine," IEEE Journal of Solid-State Circuits]

Achieving Scale through Hierarchical Design

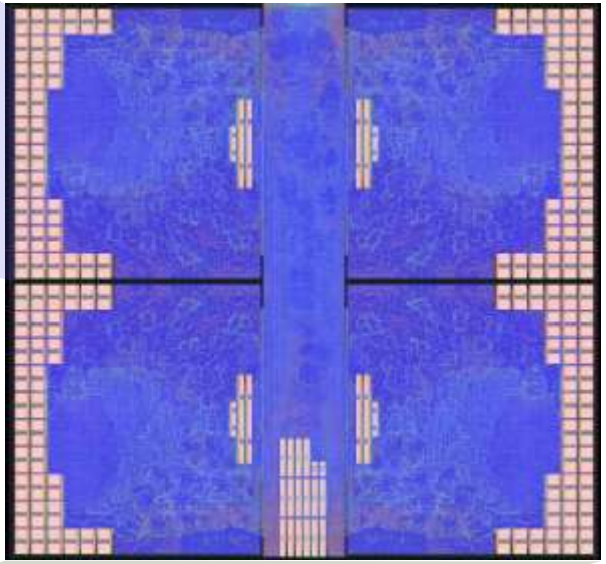
Snitch Core



Snitch Cluster

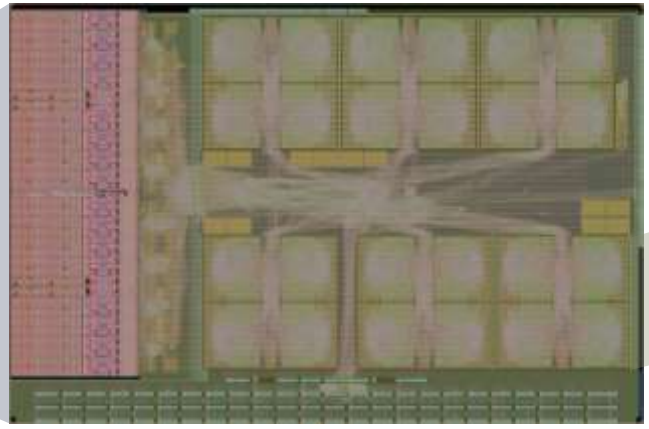


Occamy Group



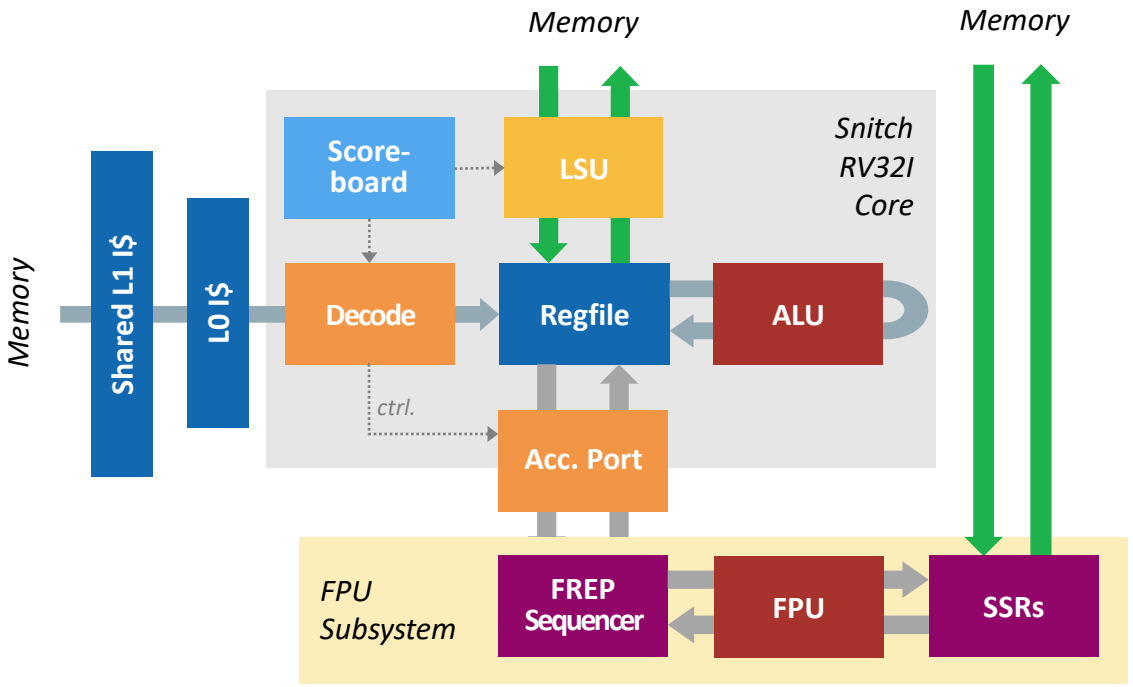
Occamy System

Occamy Chiplet



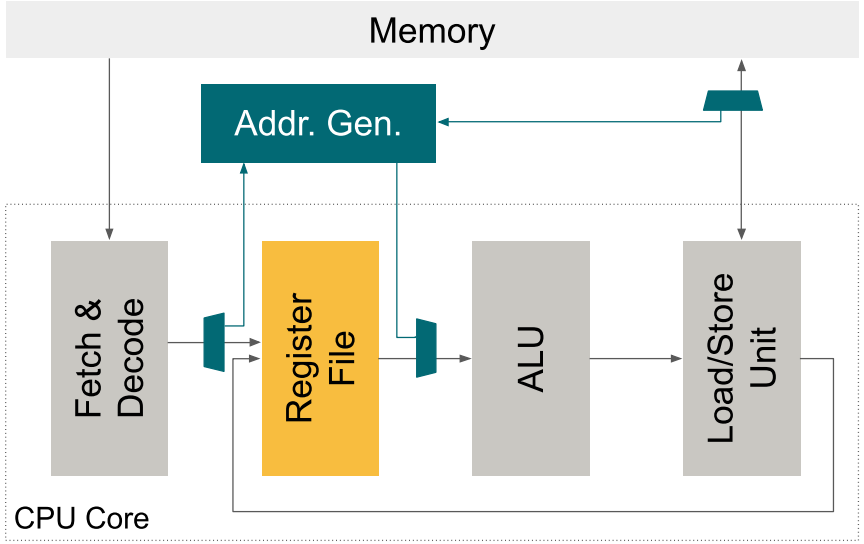
Snitch Core: Tiny Integer Control Core with Large FPU

- ***Snitch*: tiny, extensible RV core**
  - *Extensible* through accelerator port
  - *Latency-tolerant* through scoreboard
    → can issue ~10 non-blocking memOPs
- **Usually paired with *FPU subsystem***
  - Large, pipelined double-precision FPU
  - FP8-FP64 SIMD capable
  - Multiple subsystems supported (e.g. Spatz vector unit, INT-SIMD,…)
- **ISA extensions for *near-ideal FPU util.***
  - ***SSRs***: map *memory streams* to *FP registers*
  - ***FREP:*** dedicated *HW loop* for FPU
    → FPU & int. core can compute *in parallel*



SSR & FREP: the Key for PE efficiency

- **SSR**: Link register read/writes into implicit LD/ST
  - Extension around the core's register file
  - Address generators (2-3KGE/SSR)
  - Configured out of inner loop (LD/ST elision)
  - Staggering: generators prefetch from memory (latency tolerant!)
- **FREP**: L0 instruction buffer (no I$ access)
  - Pseudo-dual issue (Int pipeline can proceed in parallel)
  - No boundary checking for loop (similar HW loop in DSPs)
- **Boost FPU utilization → 100% (once setup is amortized)**



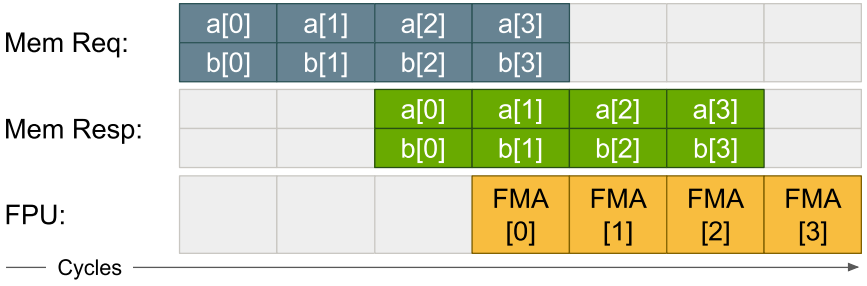
dotp: 30% FPU

```
loop:
fld r0, %[a]
fld r1, %[b]
fmadd r2, r0, r1
```

→

dotp: 90% FPU

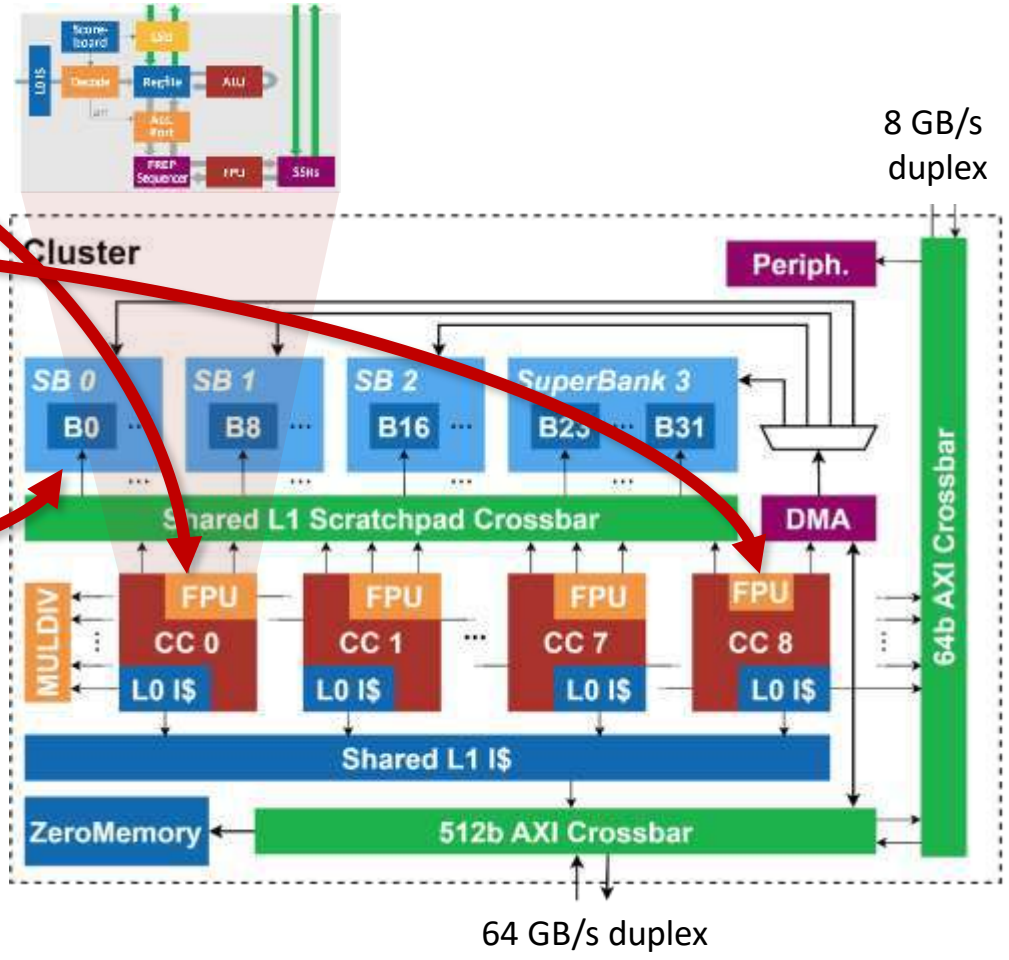
```
scfg 0, %[a], ldA
scfg 1, %[b], ldB
loop:
fmadd r2, ssr0, ssr1
```



Less expensive than OoO (CPU) and Multi-threading (GPU), complements SIMD/Vec/Mat instructions

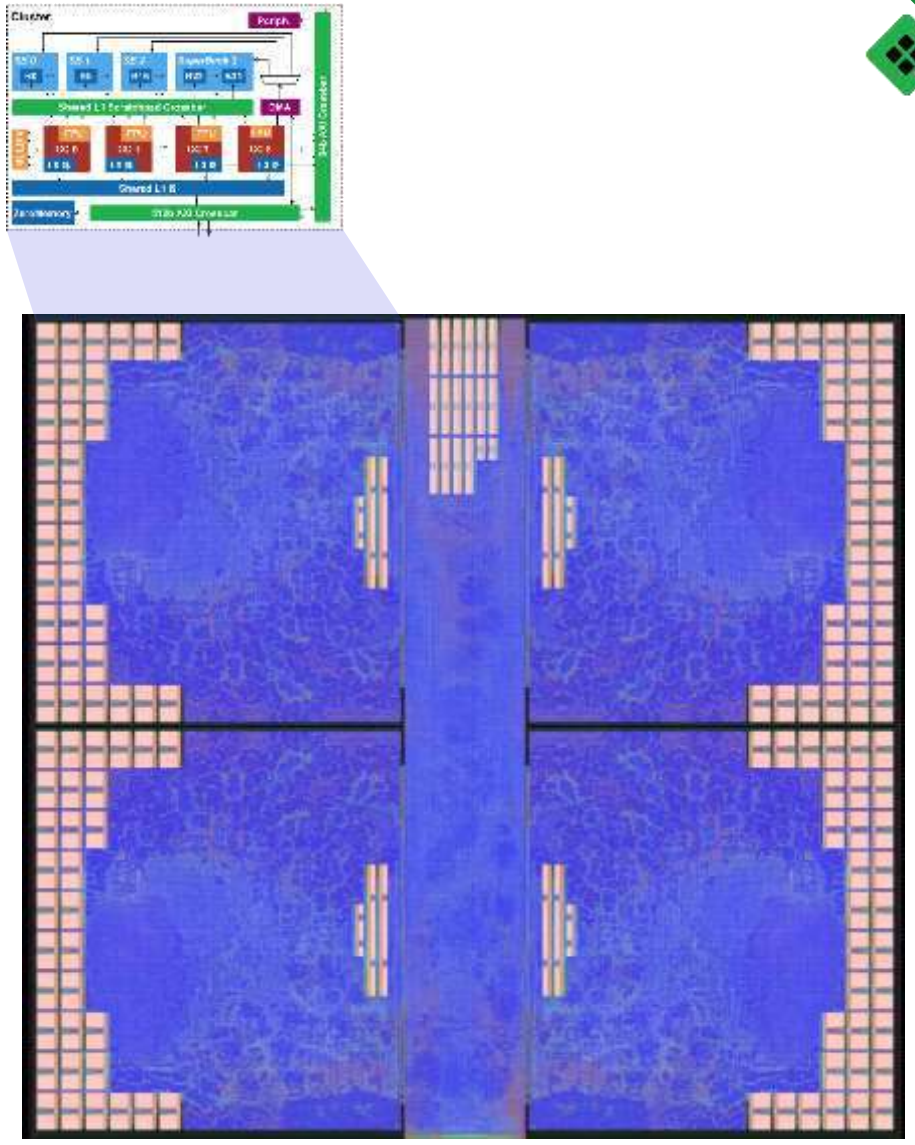
Snitch Cluster: The Fundamental Compute Block

- **8 Snitch compute cores**
  - SIMD 64b FPU with SSRs & FREP
- **9th Core: DMA engine**
  - 512b interface to interconnect
  - HW support for autonomous ≤ 2D transfers, higher dimensions through SW
  - *Latency-tolerance block transfers* (100s of cycles)
- **128 KiB TCDM**
  - 32-bank, low-latency shared scratchpad
  - Double-buffer large chunks with DMA
- **Shared I-cache and peripherals**



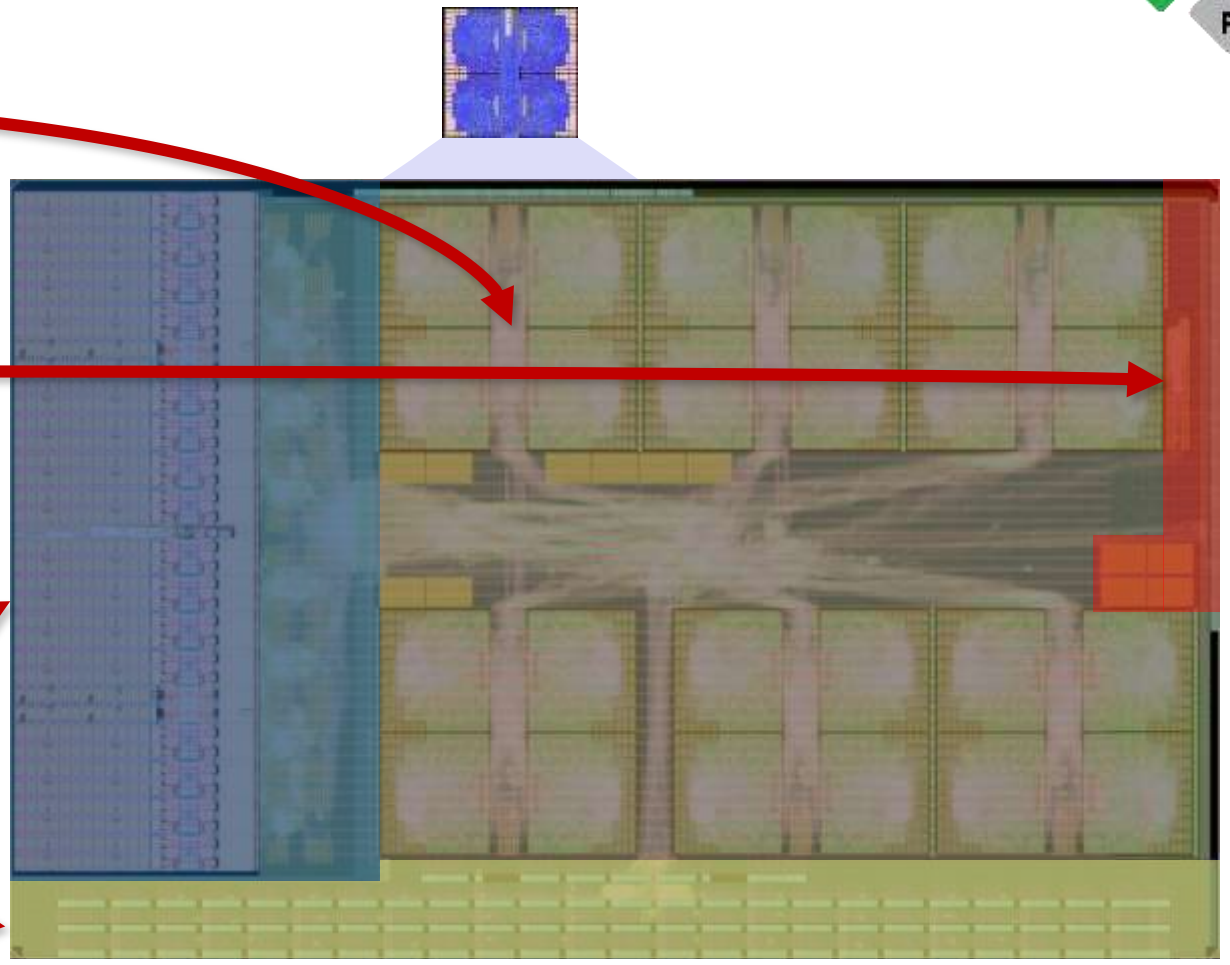
Four Clusters form a Group

- **Hierarchically shared bandwidth**
  - Clusters fully connected through crossbars → high-bandwidth local data exchange
  - Single shared 64b / 512b ports to top
- **Shared resources**
  - 32 KiB constant cache
  - IOTLB for address remap & access control
  - SW-controlled clock gating and reset
- **Simplified physical implementation**
  - 6 groups → 24 clusters per chiplet



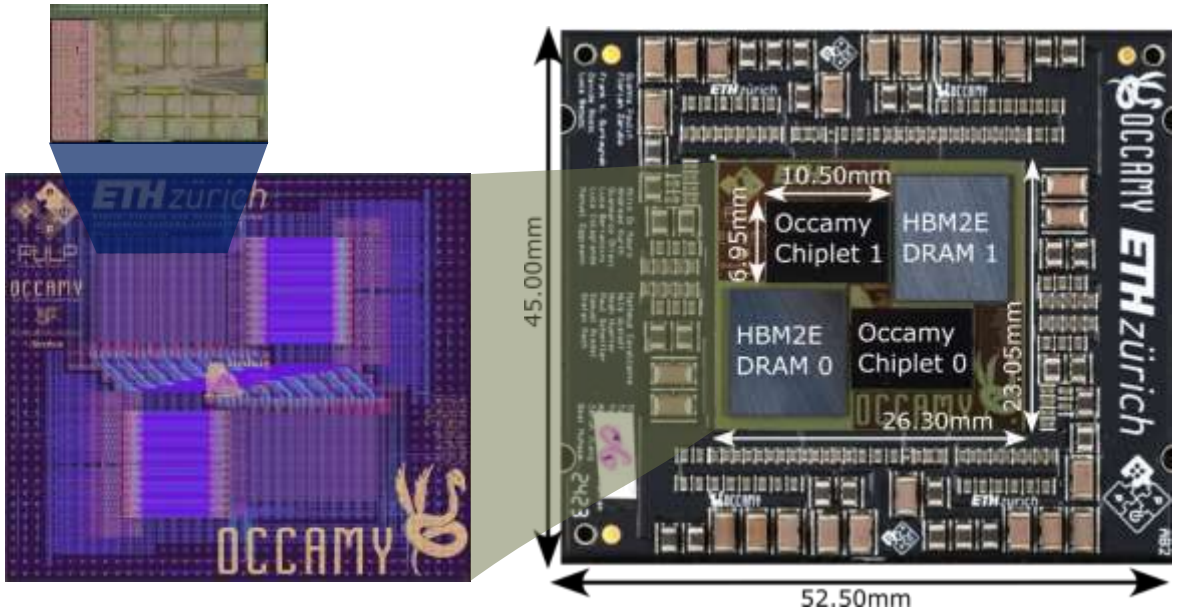
Occamy Chiplet: Six Groups with HBM and D2D Link

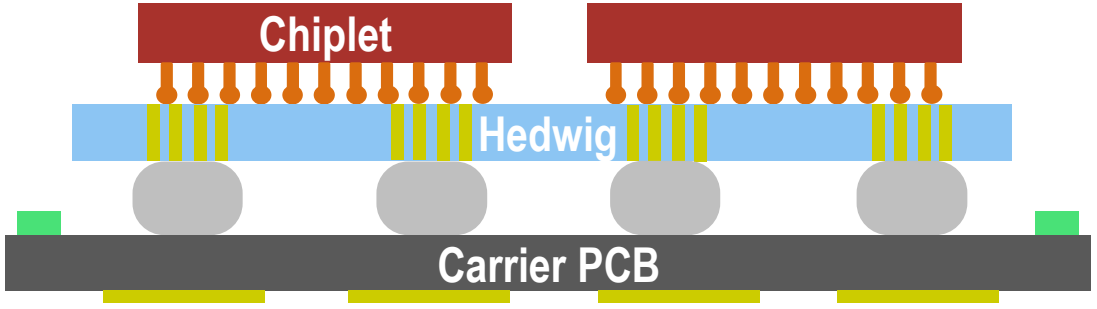
- **6 fully connected groups**
  - 24 clusters, 216 cores total
  - 512b NoC for data, 64b for messages
- **Autonomous 64b host domain**
  - CVA6 RV64GC Linux-capable core
  - Rich peripherals (SPI, I2C, UART…)
- **16 GiB, 410 GB/s HBM2E**
  - Optional page-level interleaving
- **12.8 GiB/s die-to-die link**
  - Fully digital and fault-tolerant



Occamy 2.5D System: Chiplets on Passive Interposer

- ***Hedwig* interposer**
  - 65nm, passive (BEOL only)
  - Connects 2× 73 mm^2 Occamy chiplet (GF 12LP+) and 2× Micron HBM2E
  - Distributes power and IOs
- **Carrier PCB**
  - RO4350B (low CTE, high stability)
  - LGA 2011 pinout adapted to fit ZIF socket
  - Stabilizes assembly and power
- ➢ **Occamy system module**
  - 432+2 RISC-V cores, 32 GiB HBM2E
  - 768 DP-GFLOP/s peak performance (HPC)
  - 6144 FP8-GFLOP/s peak performance (ML)





[1] https://reviews.llvm.org/D132828

LLVM Snitch Extensions

- **Why extend LLVM?**
  - Make **extensions** accessible (intrinsics, inference)
  - Improve **scheduling** and **register allocation** (default tuned for renaming OoO machines)
  - Improve **reassociation** (many RAW stalls)
- **LLVM 15 with Snitch extensions:**
  - Tuned in-order **machine model**
  - Xssr, Xfrep, Xdma **assembly** and **intrinsics**
  - Tuned **tree height reduction** pass[1] (efficient reassociation of unrolled FP math)
  - **SSR inference** based on scalar evolutions
  - **FREP inference** loop pragma

$\vec{a} \cdot \vec{b}$ →

naïve register allocation
→ RAW stalls, low IPC

FREP?

```
...
fmadd.d  fa0, ft0, ft1, fa0
fmadd.d  fa0, ft2, ft3, fa0
fmadd.d  fa0, ft4, ft5, fa0
...
```

SSRs?

SSR *Repeat* *Bound* *Stride* *Data*

```
double sum=0.0, a[N], b[N];

__builtin_ssr_setup_1d(      0,   0,  N-1,   8,   a);
__builtin_ssr_setup_1d(      1,   0,  N-1,   8,   b);

__builtin_ssr_enable();

#pragma frep infer
for (unsigned i = 0; i < N; ++i)
  sum += __builtin_ssr_pop(0) * __builtin_ssr_pop(1);

__builtin_ssr_disable();
```

Runtime Support

Data Movement

1D/2D DMA transfers for block transfers

Intrinsics for:

1. Setup
2. Coarse and Fine-Grained Synchronization

Dual-chiplet Connectivity

Fault-tolerant double-data-rate die-to-die links for robust inter-chiplet data exchange

Extensions

Sparsity-capable SUs

Specialized streaming units facilitate indirection and merging, tailored for sparse computations.

FREP

Decouples the floating-point and integer pipeline by sequencing instructions from a micro-loop buffer

Synchronization

DMA Synchronization

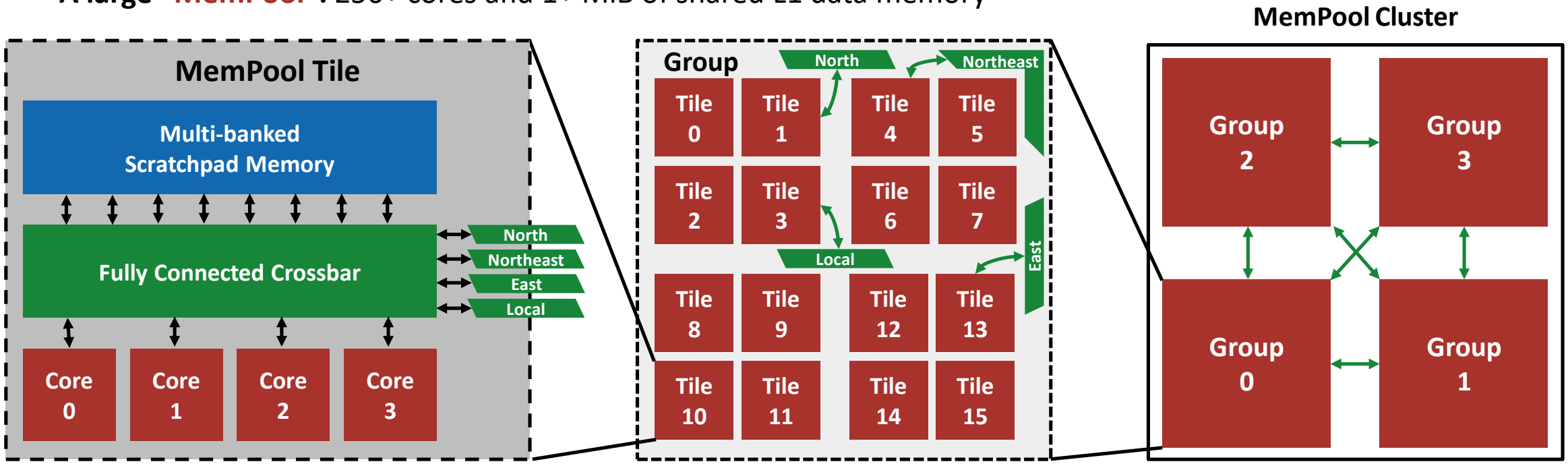
Can be exploited for different applications that require, e.g. double buffering or full synch.

Hierarchical Synch.

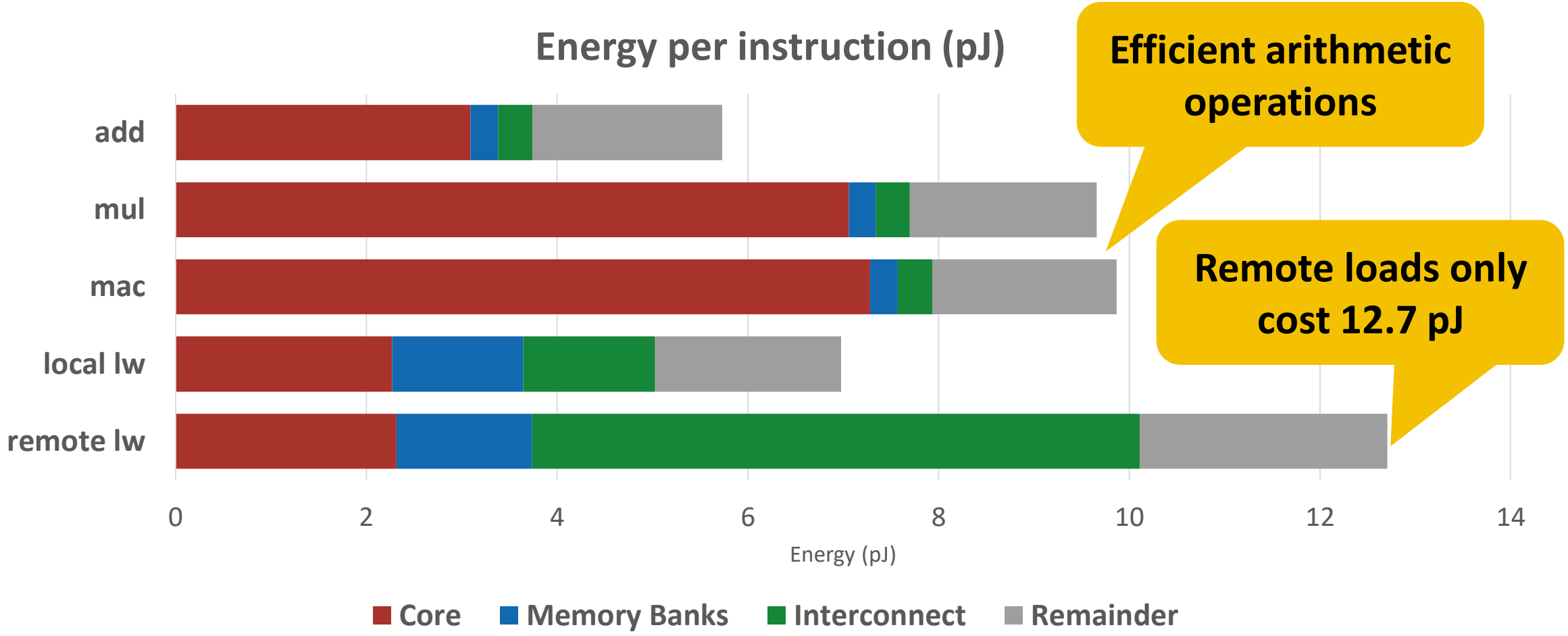
Flexibility to synchronize on cluster, and global level or partial synchronization.

Matmul Benefits from Large Shared-L1 clusters

- **Why?**
  - Better global latency tolerance if $L1_{size} > 2\times L2_{latency} \times L2_{bandwidth}$ (Little's law + double buffer)
  - Smaller data partitioning overhead
  - Larger Compute/Boundary bandwidth ratio: N^3/N^2 for MMUL grows linearly with N!
- **A large "MemPool":** 256+ cores and 1+ MiB of shared L1 data memory



What is the cost of traversing MemPool?

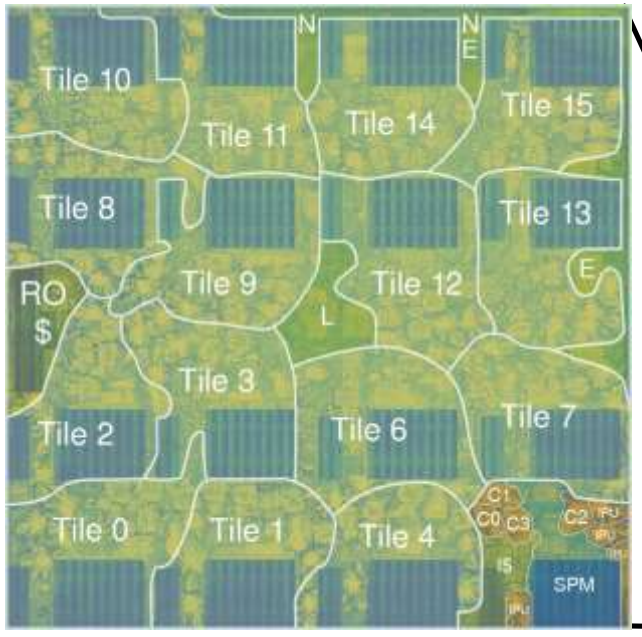


MemPool Cluster: A physical-aware design

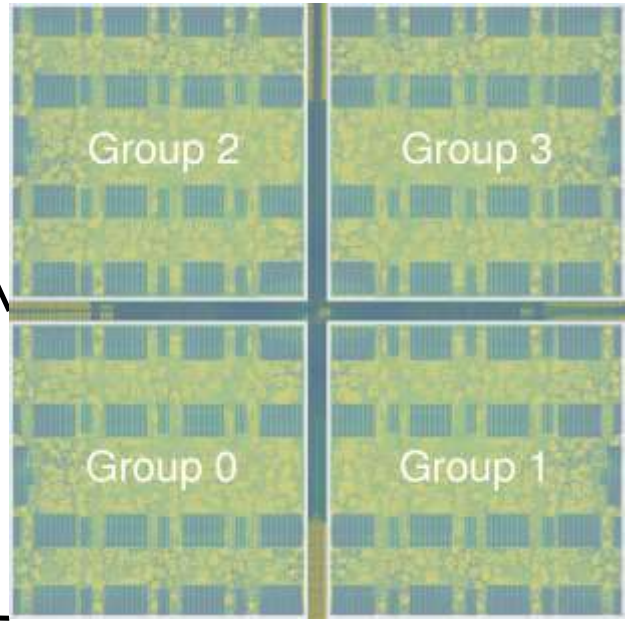
- **A Scalable Manycore Architecture with Low-Latency Shared L1 Memory**
  - 256+ cores
  - 1+ MiB of shared L1 data memory
  - ≤ 8 cycle latency (Snitch can handle it)
- **Hierarchical design**
- **Implemented in GF22**
  - Targeting 500 MHz (SS/0.72V/125°C)
  - Reaching 600 MHz (TT/0.80V/25°C)
  - Targeting iso-frequency with PULP
- **Cluster area of 13 mm²**
  - 5 mm diagonal
  - Round trip in 5 cycles
- **Terapool: 1024 Cores!**

MemPool
&
Terapool

MemPool Group

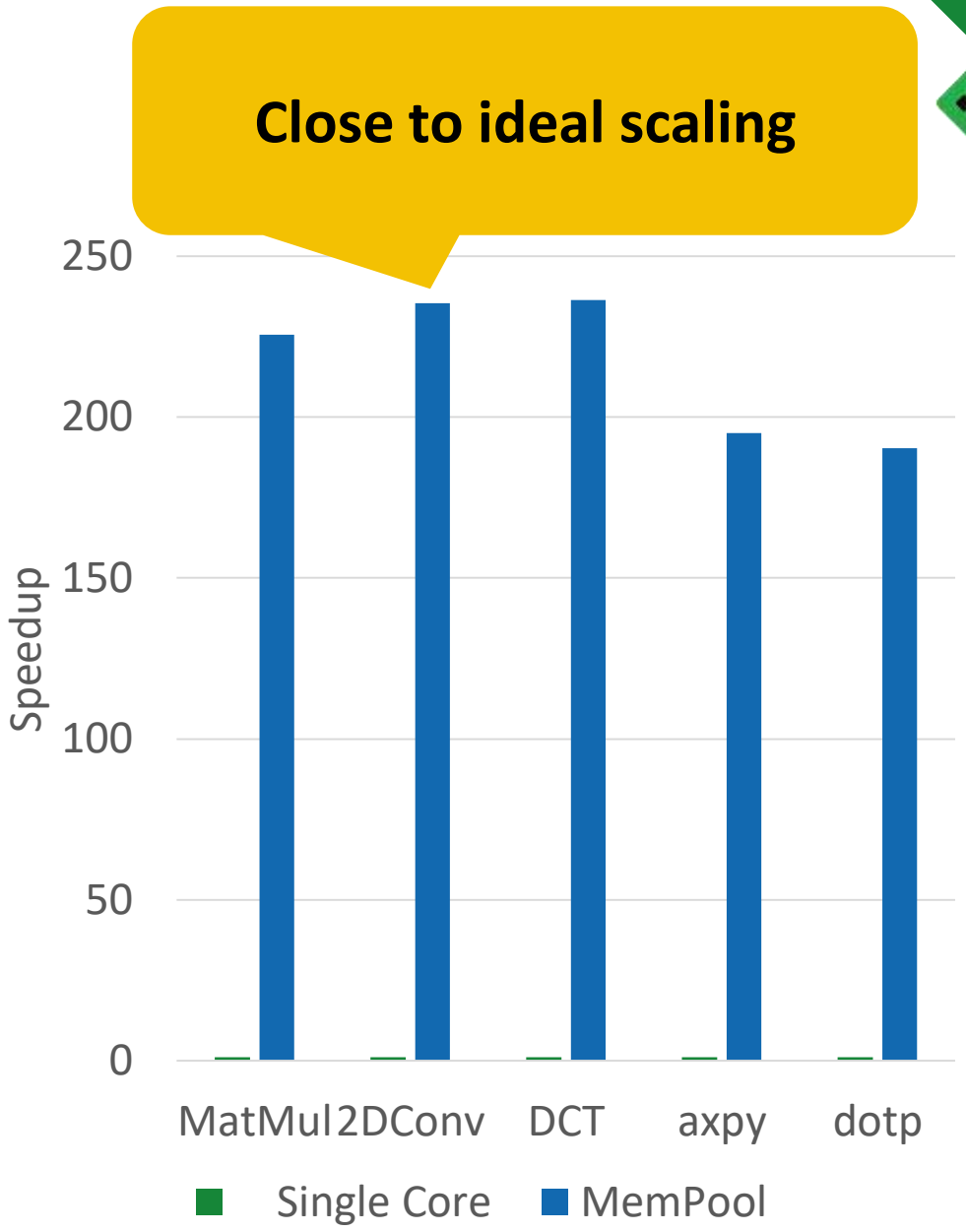


MemPool Cluster



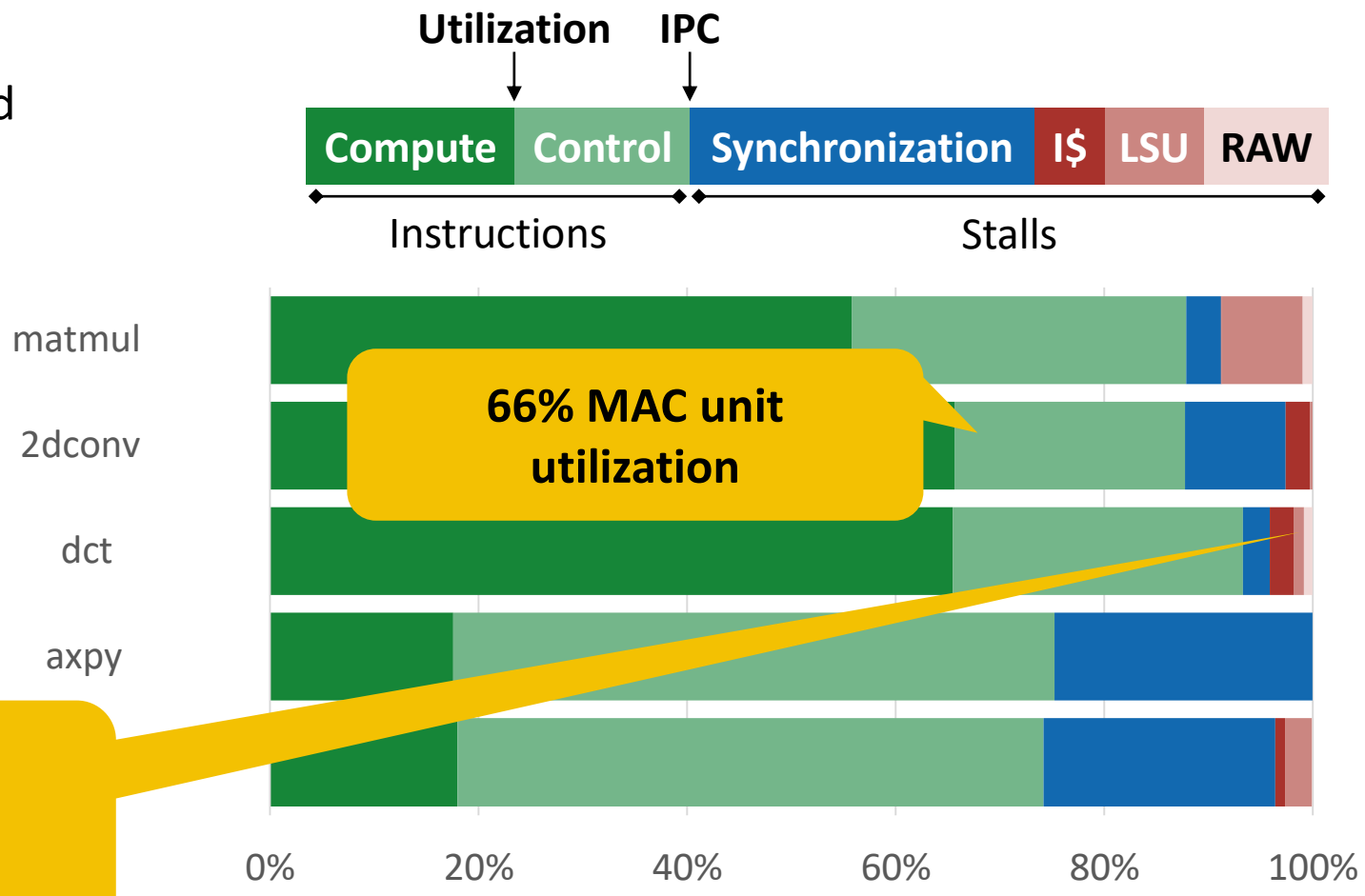
How well do we scale?

- **Evaluate highly optimized kernels**
- **Baseline: single core system**
  - No synchronization overhead or contention
- **Compute-heavy kernels achieve more than 88% of the ideal speedup**
- **Memory-bound kernels still achieve 75% of the ideal speedup**



Where do we lose performance?

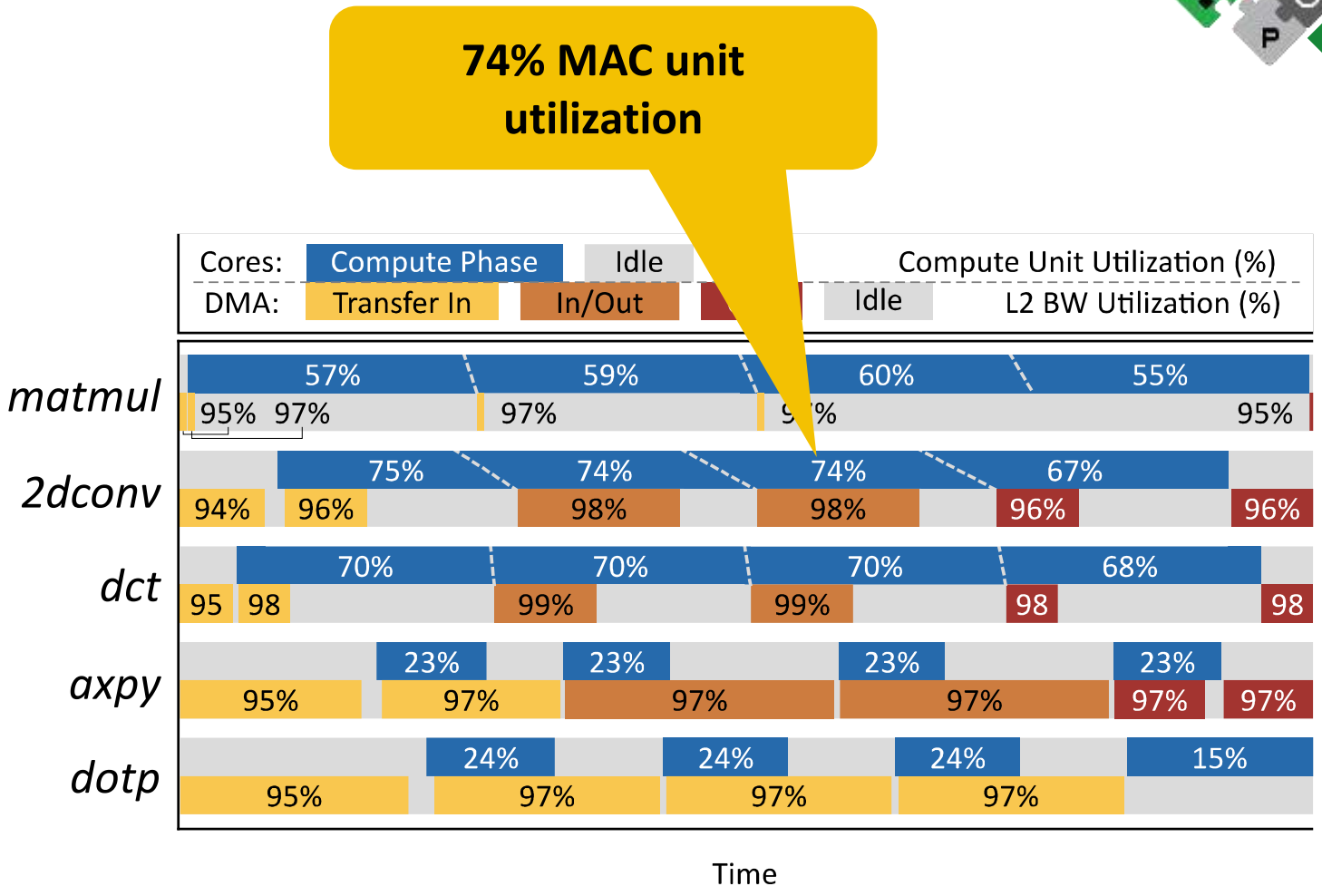
- **Compute-heavy kernels:**
  - Mainly synchronization overhead
  - Interconnect congestion
  - Achieve up to **66% MAC unit utilization**
- **Memory-bound kernels:**
  - Short execution time
  - Synchronization overhead
  - Still achieve **IPC of 75%**



Minimal architectural stalls

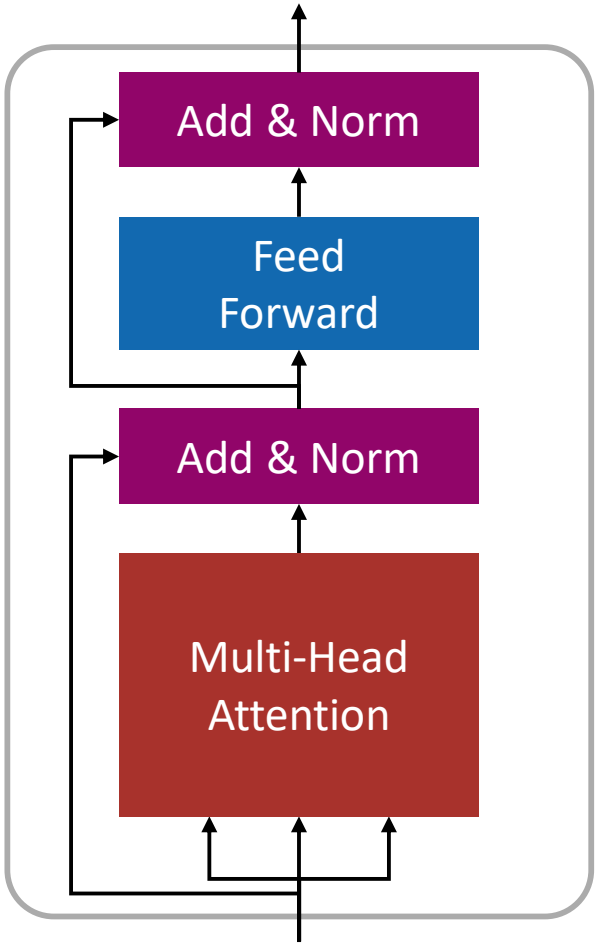
System Performance

- **Double-buffer the kernels**
  - Overlap compute and data transfer phase
- **Compute-bound kernels:**
  - Eliminate strict barriers
  - Achieve up to **74% MAC unit utilization**
- **Memory-bound kernels:**
  - Performance restricted by the L2 bandwidth
  - Almost at the roofline with **97% DMA utilization**

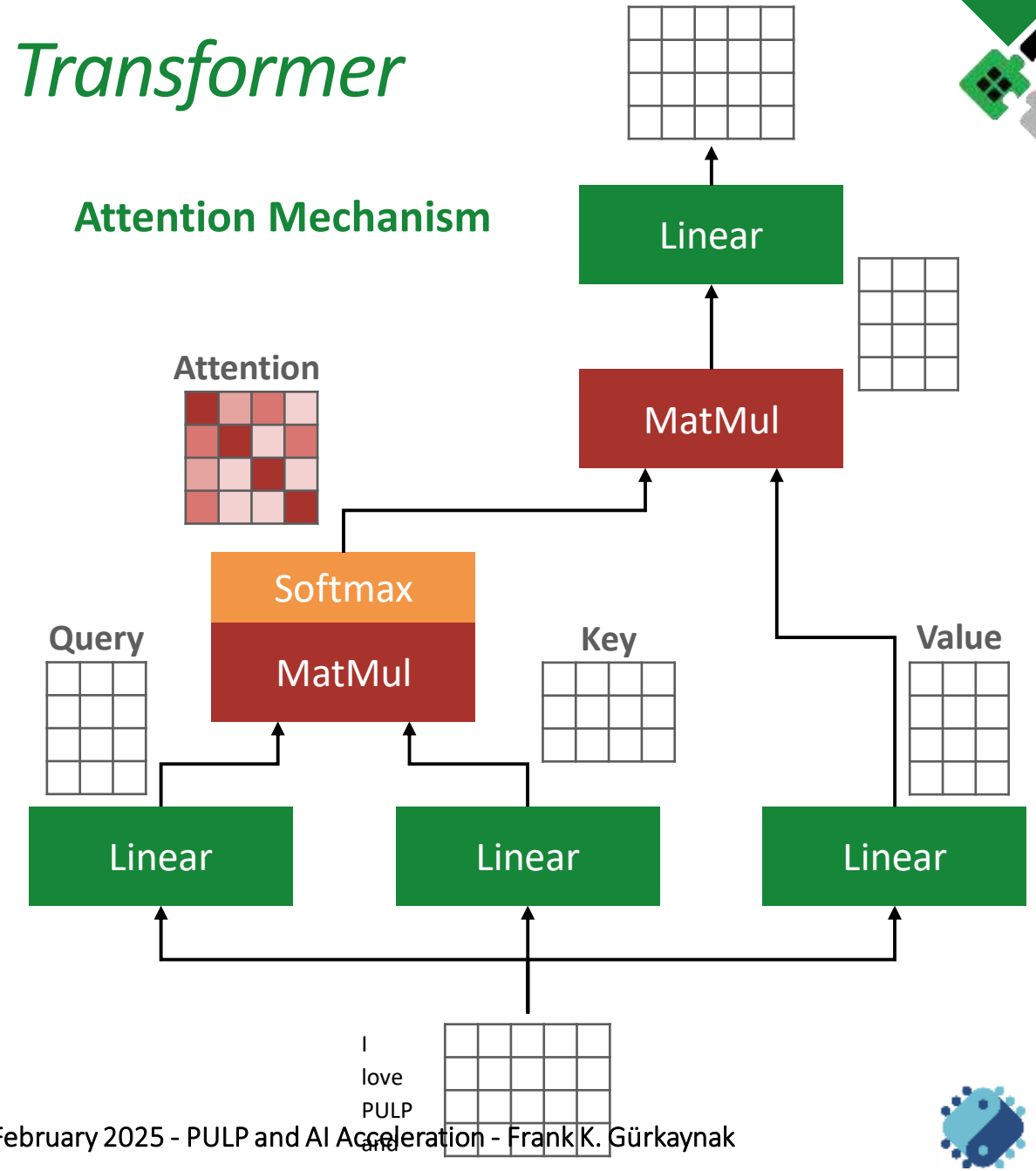


Accelerator for Attention — *Transformer*

Encoder Block of Transformers

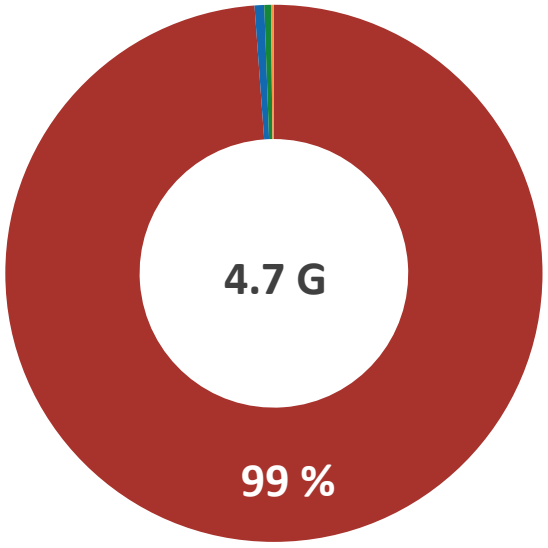


Attention Mechanism



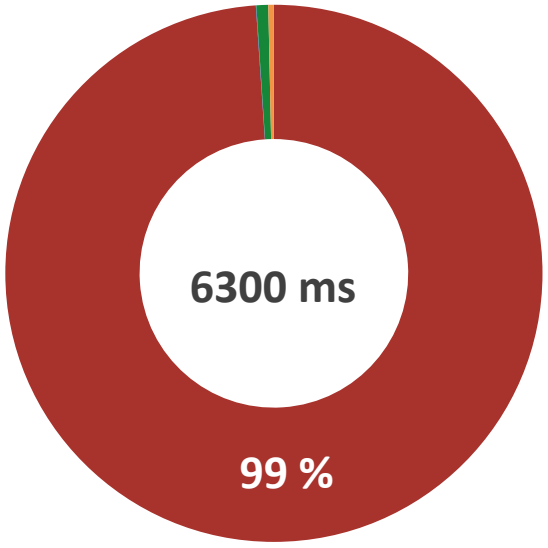
Accelerator for Attention — MobileBERT

Arithmetic Operations



GEMM Add
Softmax NoNorm

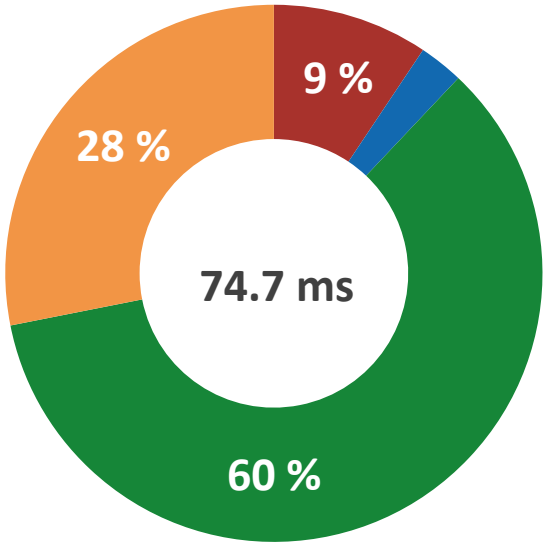
Latency
Everything on Cluster



GEMM Add
Softmax NoNorm

84x

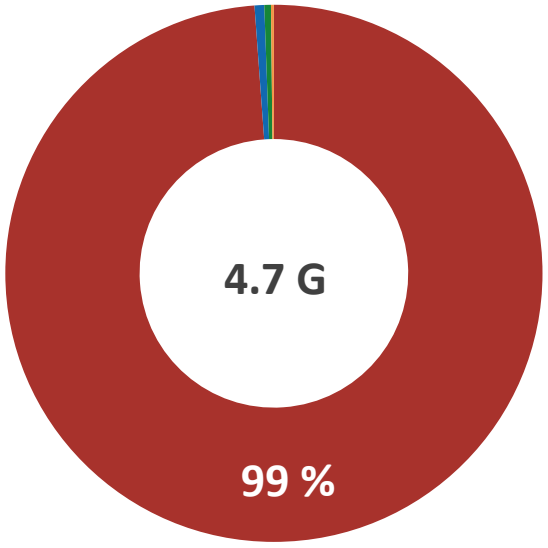
Latency
GEMM on Accelerator



GEMM Add
Softmax NoNorm

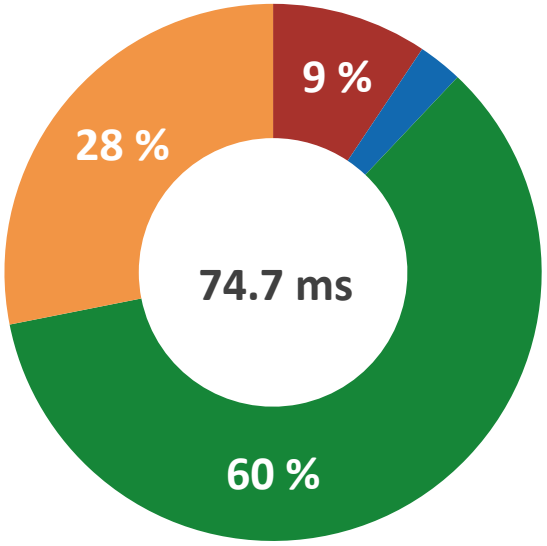
Accelerator for Attention — MobileBERT

Arithmetic Operations



GEMM Add
Softmax NoNorm

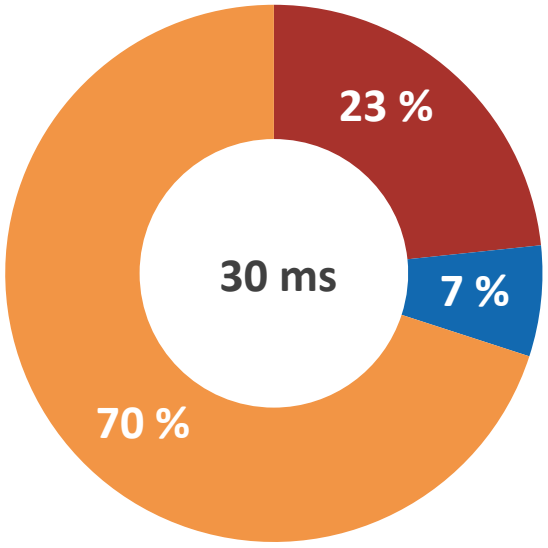
Latency
GEMM on Accelerator



GEMM Add
Softmax NoNorm

2.5x

Latency
GEMM & Attention on Accelerator
Softmax fused with GEMM

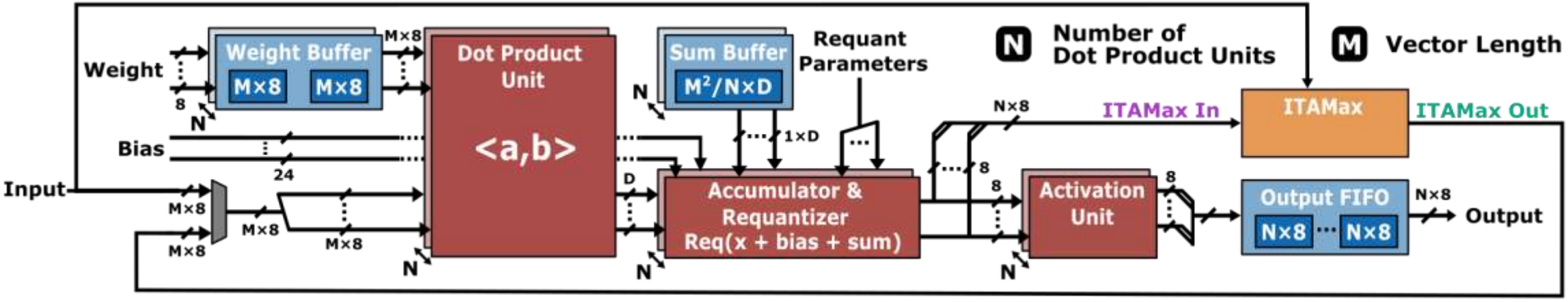


GEMM Add
Softmax NoNorm

Integer Transformer Accelerator — Dataflow

Heterogeneous Accelerated Architecture

- *Attention* accelerator for Transformers!
- INT8 quantized networks
- Fused $Q \times K^T$ and A$\times V$ computation
- Special *Softmax* unit!



Toward **Attention**-based **TinyML**

Heterogeneous Accelerated Architecture

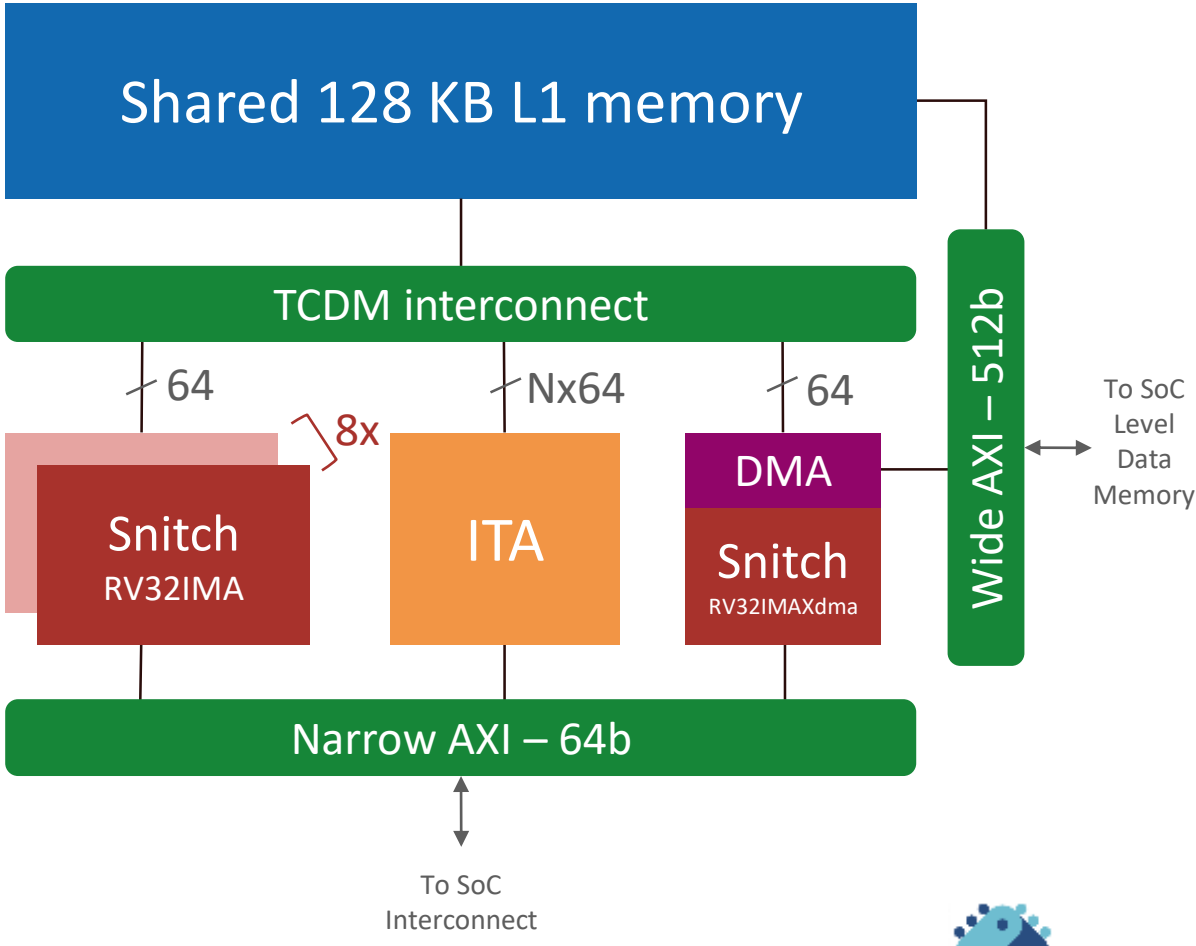
- Snitch multi-core cluster with an Attention accelerator (ITA)
- Communication via shared L1 memory

End-to-end Deployment with an Automated Flow using Deeploy

- Map DNN layers to kernel templates
- Solve tiling and memory allocation as one CP problem
- Requires a very minimal accelerator model
- Collaborative execution between cores and ITA.

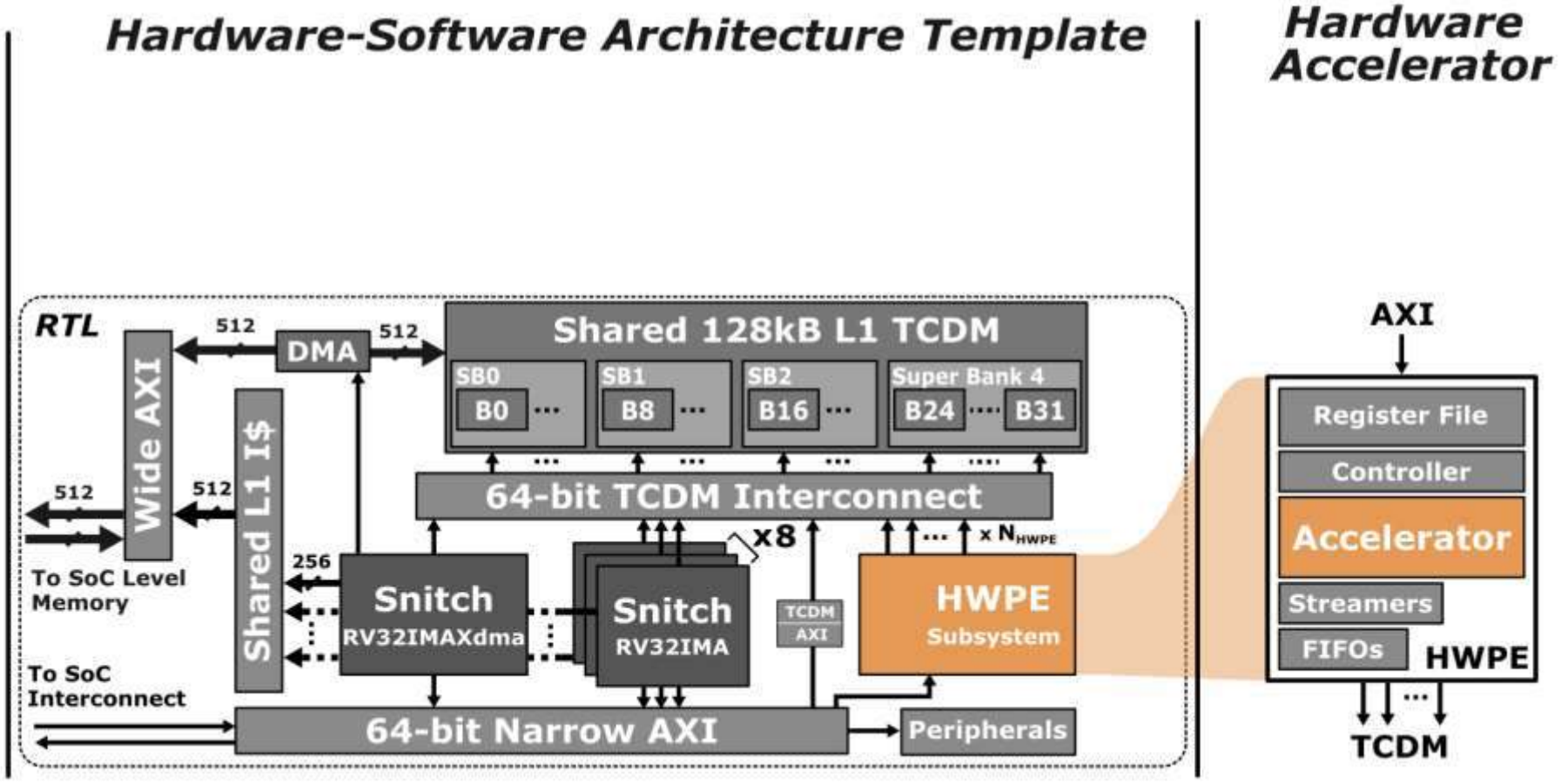
Wiese, Philip, et al. **"Toward Attention-based TinyML: A Heterogeneous Accelerated Architecture and Automated Deployment Flow"**, 2024

Snitch Cluster with Integer Transformer Accelerator (ITA)



Toward **Attention**-based **TinyML**

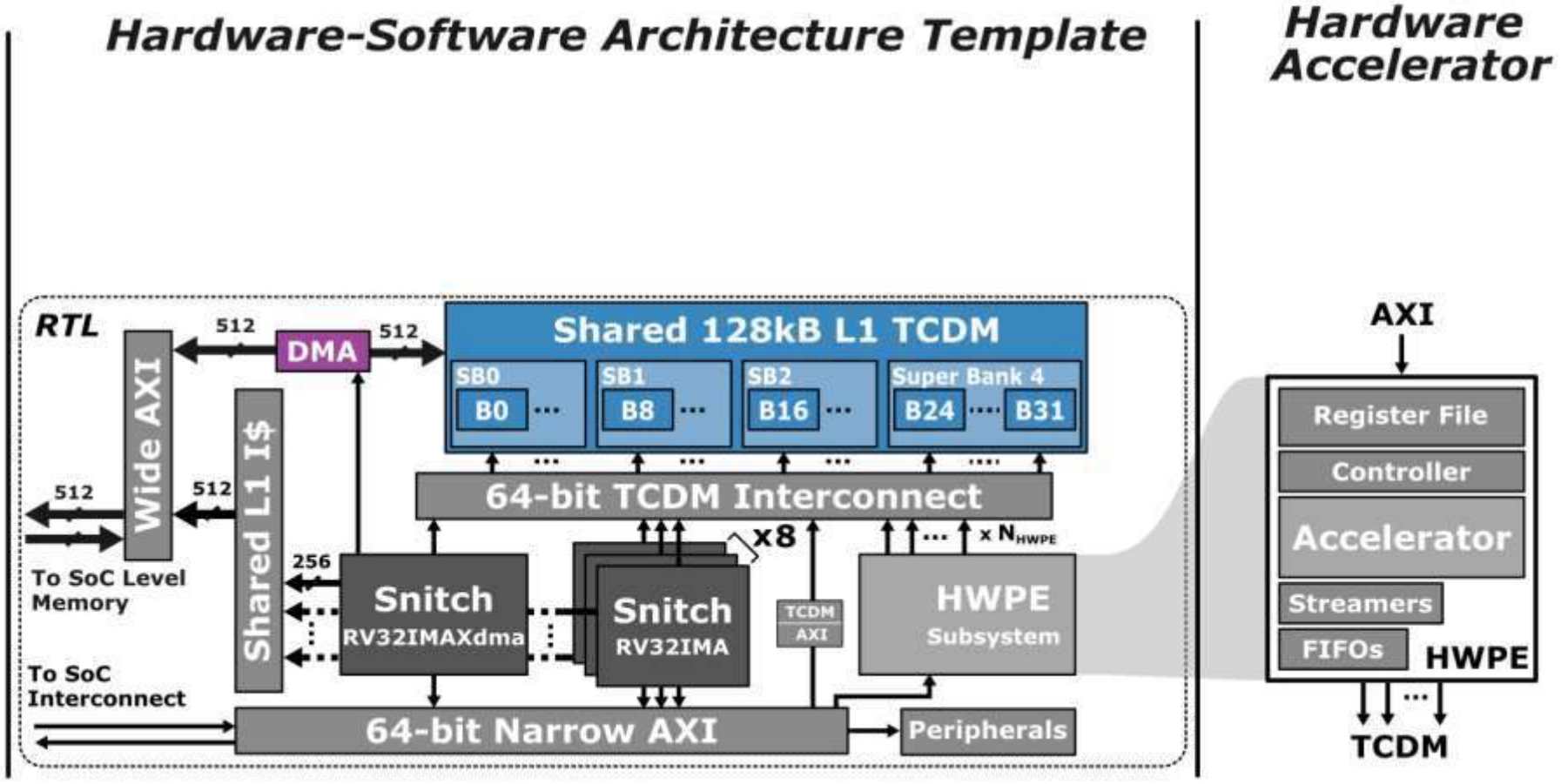
Step 1: Integrate Accelerator as HWPE Engine



Hardware Processing Engines, https://hwpe-doc.readthedocs.io/

Toward **Attention**-based **TinyML**

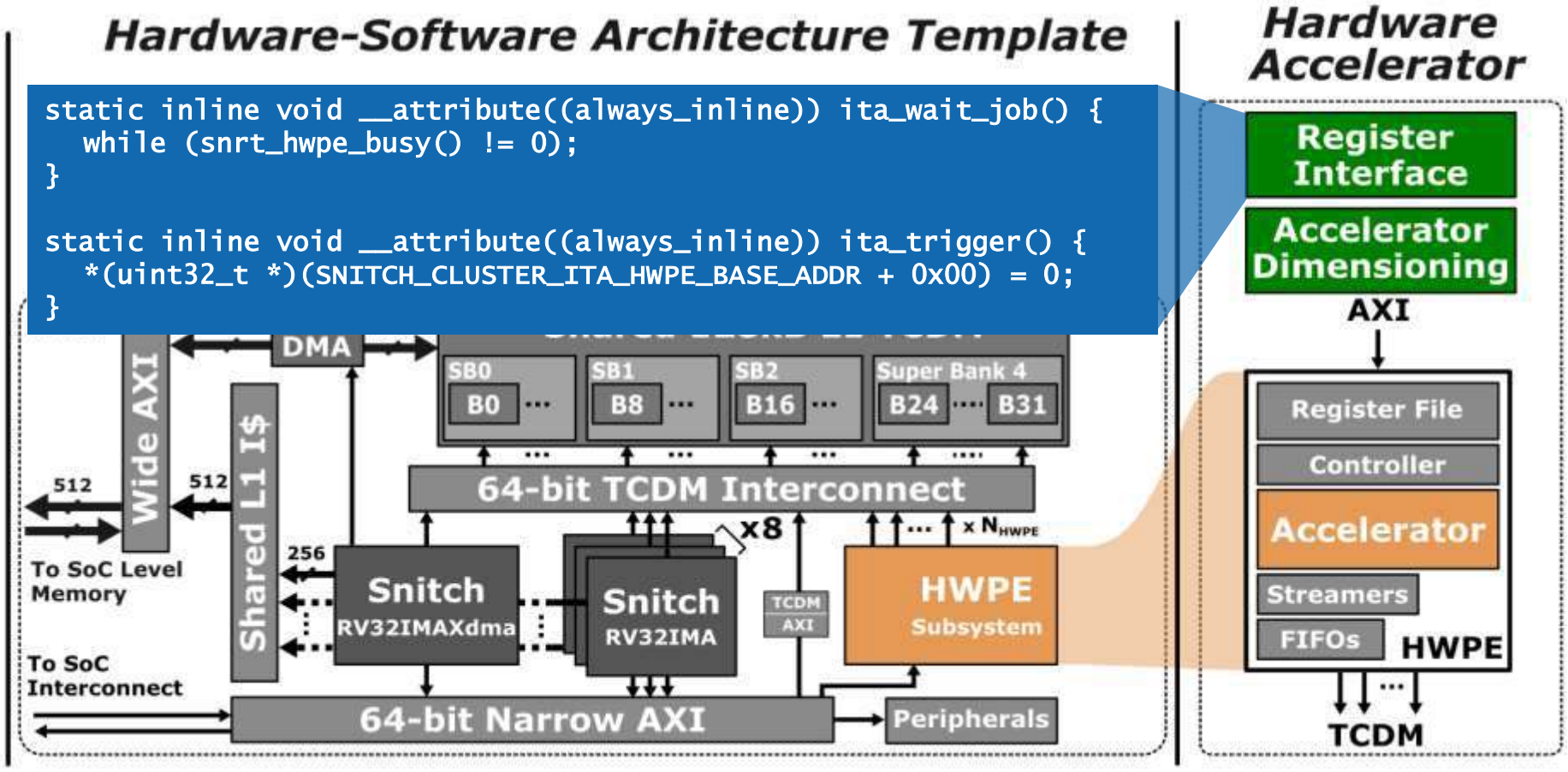
Step 2: Tune Interconnect Bandwidth



Hardware Processing Engines, https://hwpe-doc.readthedocs.io/

Toward **Attention**-based **TinyML**

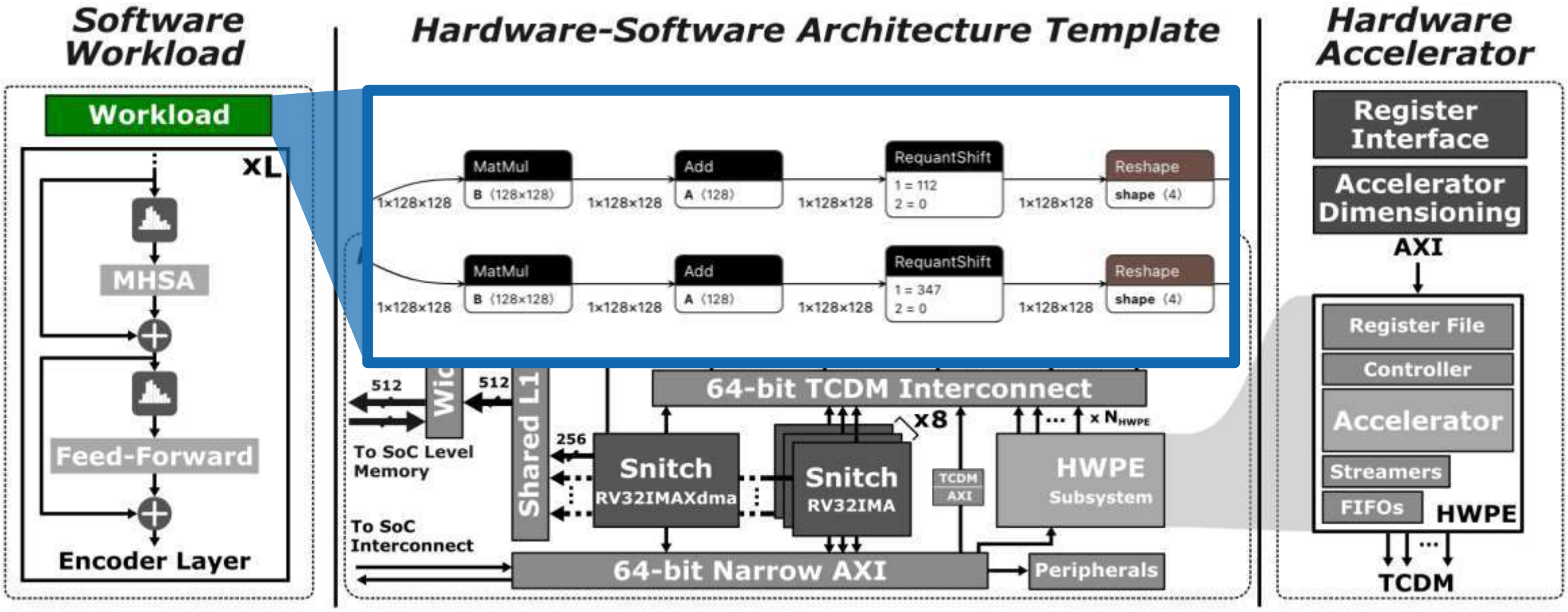
Step 3: Develop Register Interface & Extract Accelerator Constraints



Hardware Processing Engines, https://hwpe-doc.readthedocs.io/

Toward **Attention**-based **TinyML**

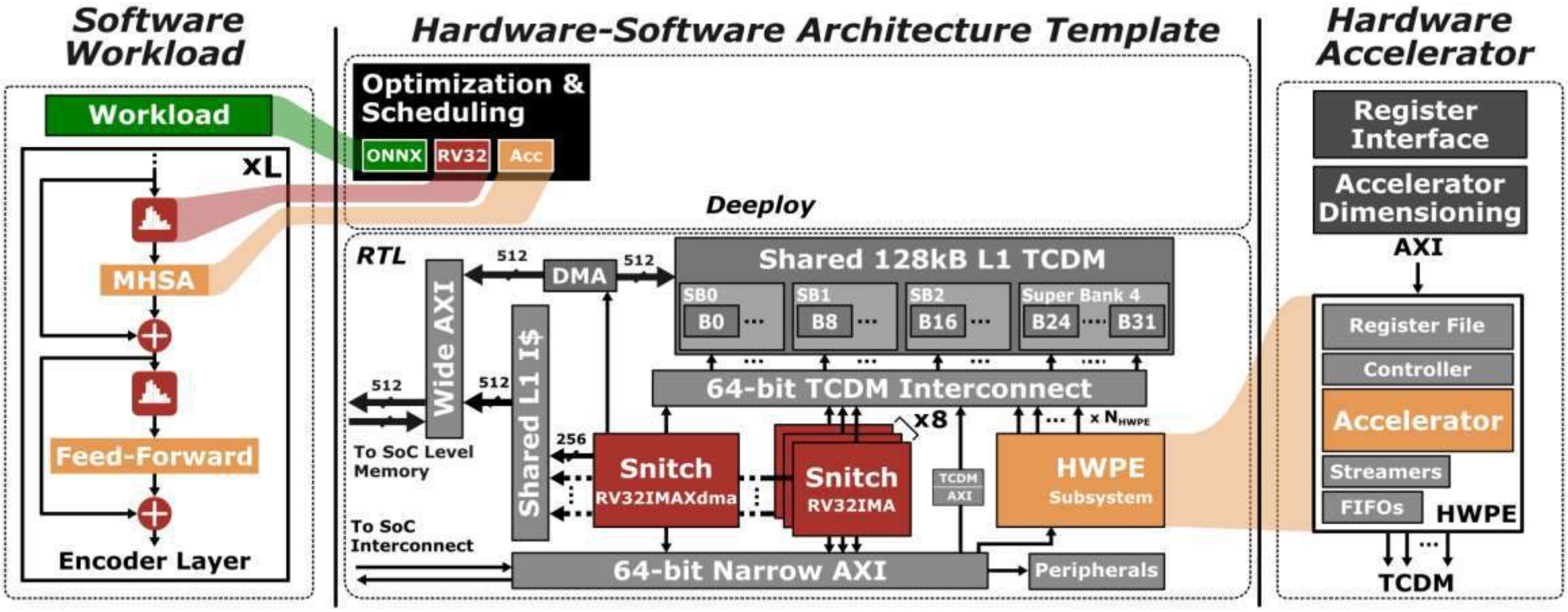
Step 4: Specify Workload



ONNX, Deeploy, https://github.com/pulp-platform/Deeploy

Toward **Attention**-based **TinyML**

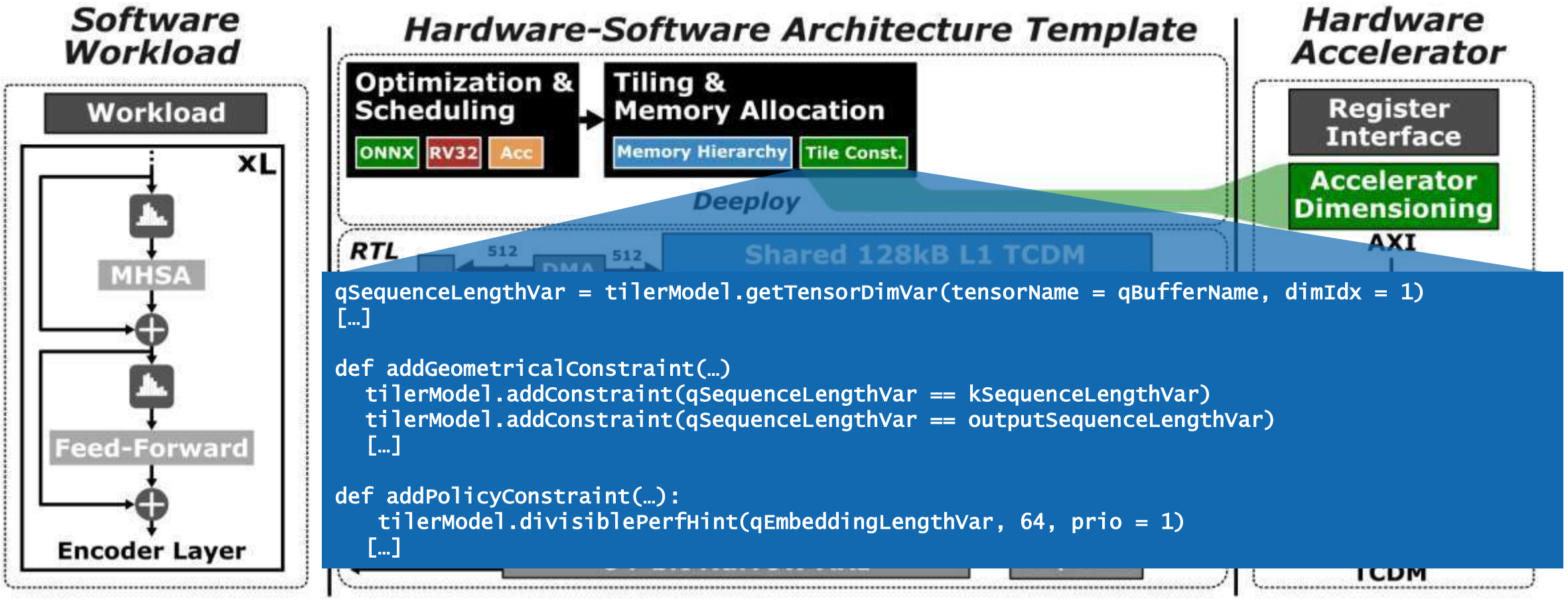
Step 5: Use Deeploy to Optimize and Schedule Operators



Deeploy, https://github.com/pulp-platform/Deeploy

Toward **Attention**-based **TinyML**

Step 6: Use Deeploy to Tile & Calculate Static Memory Allocation

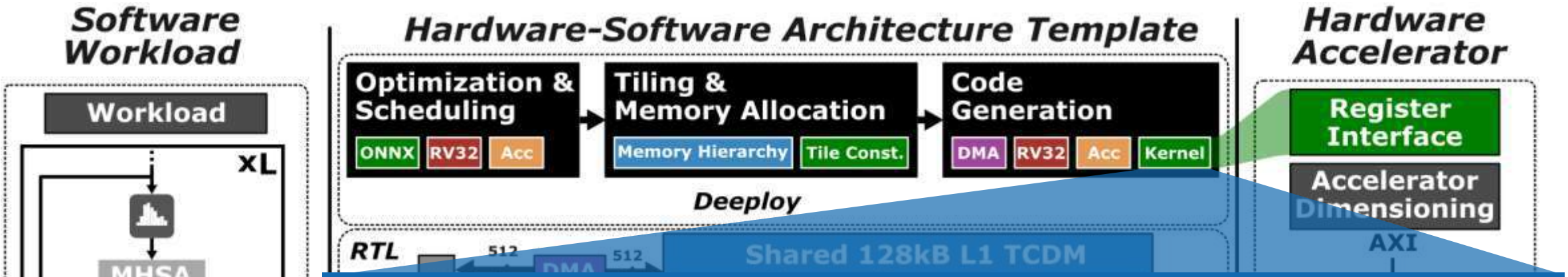



```
qSequenceLengthVar = tilerModel.getTensorDimVar(tensorName = qBufferName, dimIdx = 1)
[…]

def addGeometricalConstraint(…)
  tilerModel.addConstraint(qSequenceLengthVar == kSequenceLengthVar)
  tilerModel.addConstraint(qSequenceLengthVar == outputSequenceLengthVar)
  […]

def addPolicyConstraint(…):
   tilerModel.divisiblePerfHint(qEmbeddingLengthVar, 64, prio = 1)
  […]
```

Deeploy, https://github.com/pulp-platform/Deeploy

Toward **Attention**-based **TinyML**

Step 7: Use Deeploy to Generate Code




```
class ItaMHSATemplate(NodeTemplate):
  def computeTransientBuffersSize(…):
    buffersSize = [(self.nameMangleTransientBuffer("qp", nodeRep), H * S * P), …]

ItaMHSATemplateStr = r"""
  ita_soft_clear();
  ita_acquire_job();

  // Step 1
  ita_write_regs(
    (uint32_t)${q},
    (uint32_t)${wq_weight},
    (uint32_t)${wk_weight},
  […]
"""
```

Deeploy, https://github.com/

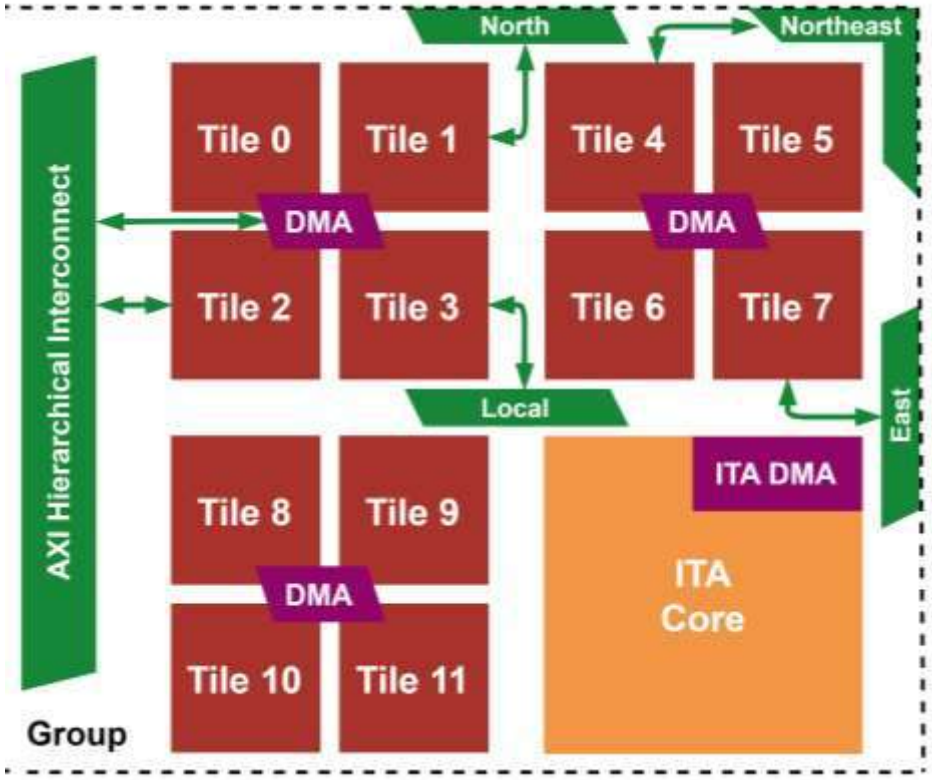
MemPool + Integer Transformer Accelerator (ITA)

Tightly coupled Acceleration Enginee

- Matmul & Softmax
- Reduce pressure on memory and interconnect

Collaborative Execution

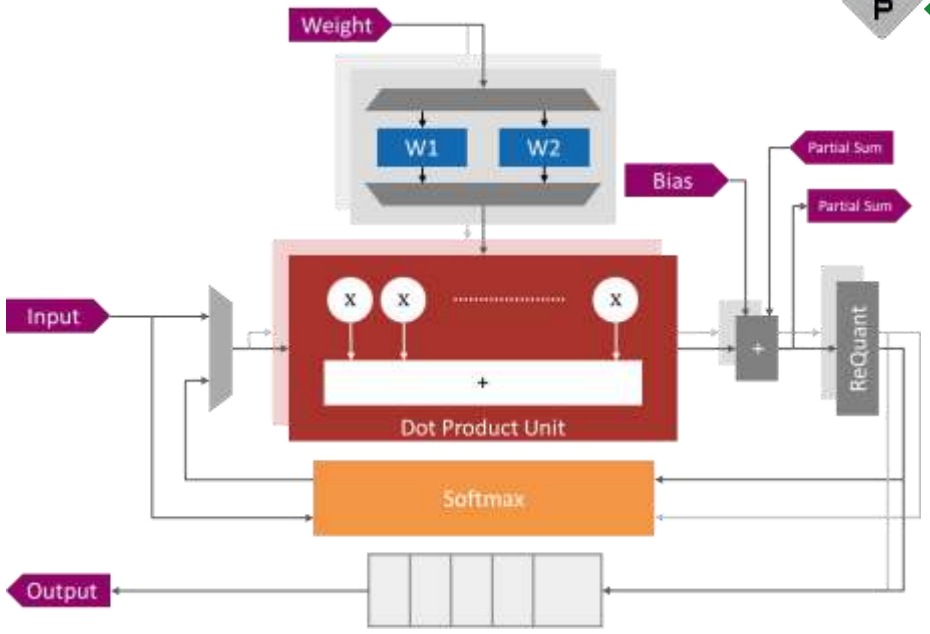
- Cores prepare activations for the next attention head
- Final head accumulation computed in cores
- Nonlinearity in cores (PACE)



MemPool + Integer Transformer Accelerator (ITA)

Integer Attention Accelerator

- 8-bit inputs, weights & outputs
- Builtin data marshaling & pipelined operation
- Streaming partial Softmax **adding no additional latency**
- Fused Q× K^T, Softmax and A × V computation
- Support for hardware-aware Softmax approximation in QuantLib



Dot Product Units	Q	K	V	$Q.K^T$	A.V	Output
Softmax				DA	EN	
				DI		

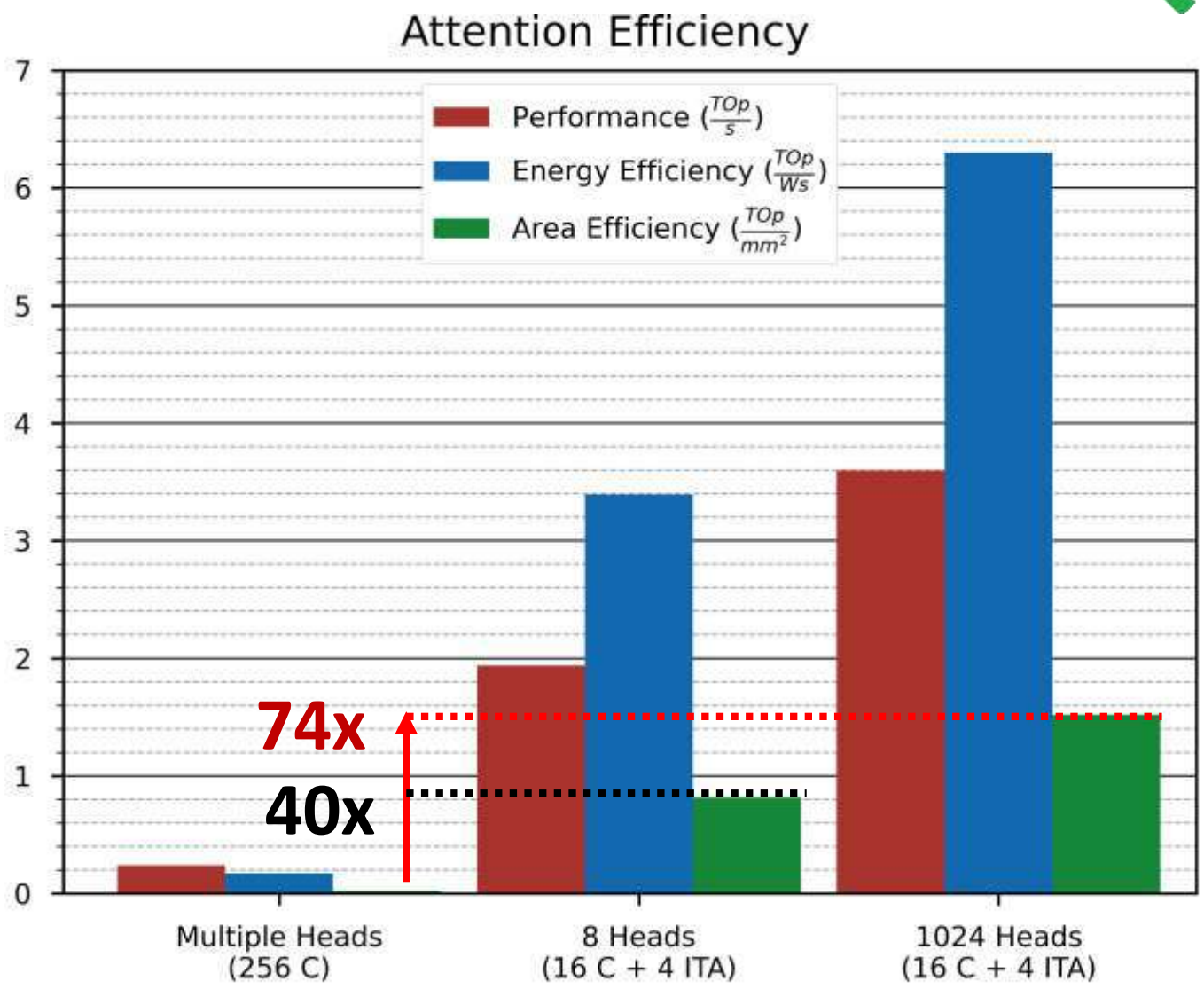
$$e^{a_i - a_{max_{n+1}}} = e^{a_i - a_{max_n}} \cdot e^{a_{max_n} - a_{max_{n+1}}}$$

Attention on ITA

Performance
increase of **15x**

Energy Efficiency
increase of **36x**

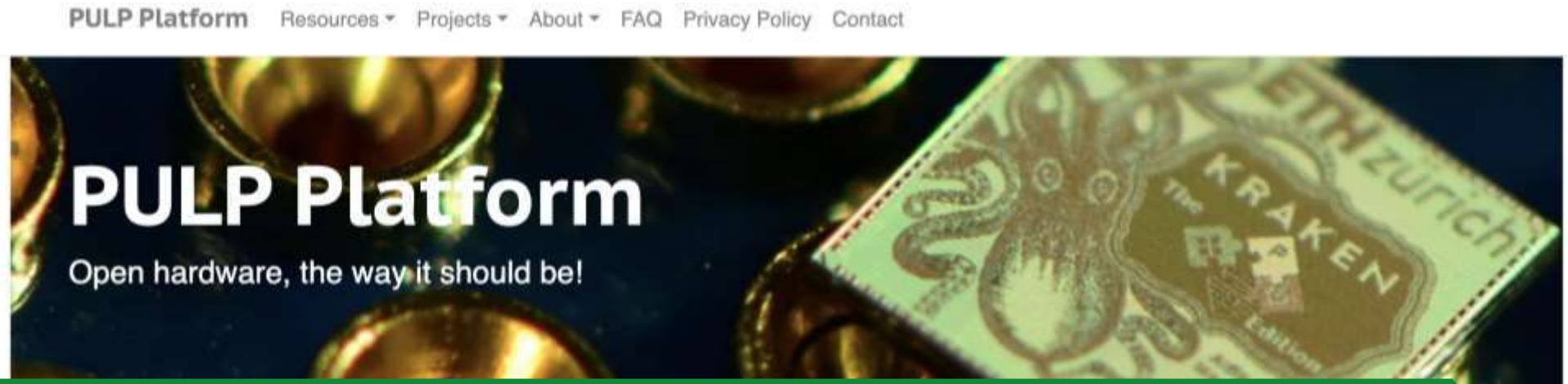
Area Efficiency
increase of **74x**



Final words

- **There is a lot of work to be done in efficient computer architectures**
  - These are exciting times, lot of new opportunities
- **There is plenty to do also for embedded AI**
  - The large datacenter chip discussions are between few big players
  - But AI/ML is not only in the datacenter, there is plenty to do on edge devices
- **There is a 50 year history of computing architectures**
  - Important to understand what has already been done, and what the limits are
  - Changing paradigms allow us to take a look at some older problems once again
- **Acceleration requires matching of compute and memory resources**
  - What and how much can be stored and replenished determines much of performance
  - Understanding data movement in and off chip to keep compute units occupied essential

Our WWW page contains a wide collection of talks/papers



Most of our talks: https://pulp-platform.org/conferences.html

And most of our papers: https://pulp-platform.org/publications.html

VLSI-SoC 2024 in Tangier.

28 August 2024

Luca Benini received the 2024 TCMM Open Source Hardware Contribution Award for his work on the PULP platform. The award was presented at Hot Chips 2024