

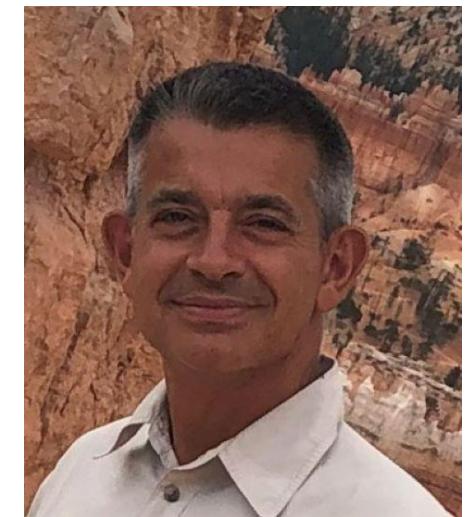
PULP PLATFORM

Open Source Hardware, the way it should be!

The Rise of Tightly-Coupled Processor Clusters

Luca Benini

<lbenini@iis.ee.ethz.ch,luca.Benini@unibo.it>



EuroHPC
Joint Undertaking



FNSNF

FONDS NATIONAL SUISSE
SCHWEIZERISCHER NATIONALFONDS
FONDO NAZIONALE SVIZZERO
SWISS NATIONAL SCIENCE FOUNDATION

ETH zürich



<http://pulp-platform.org>



@pulp_platform



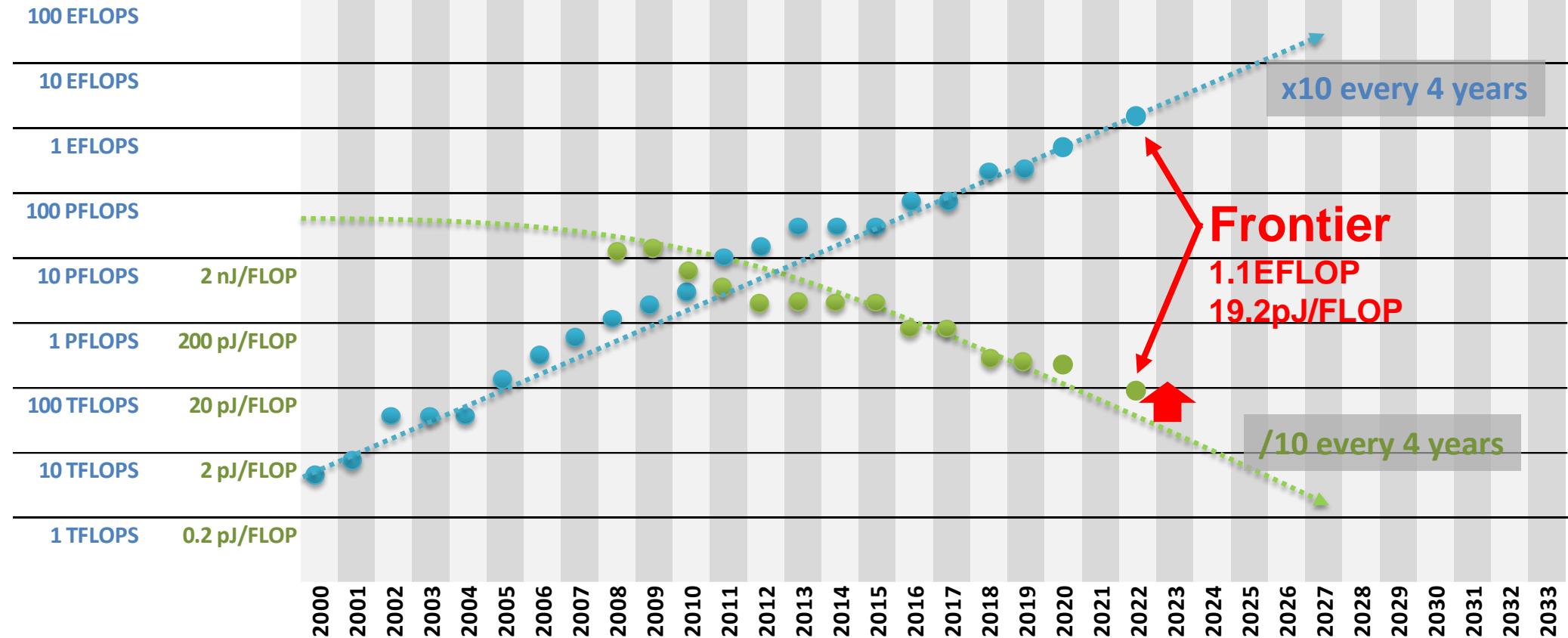
https://www.youtube.com/pulp_platform

Computing is Power Bound: HPC

HPC Performance

Energy / Op (for a 20MW budget)

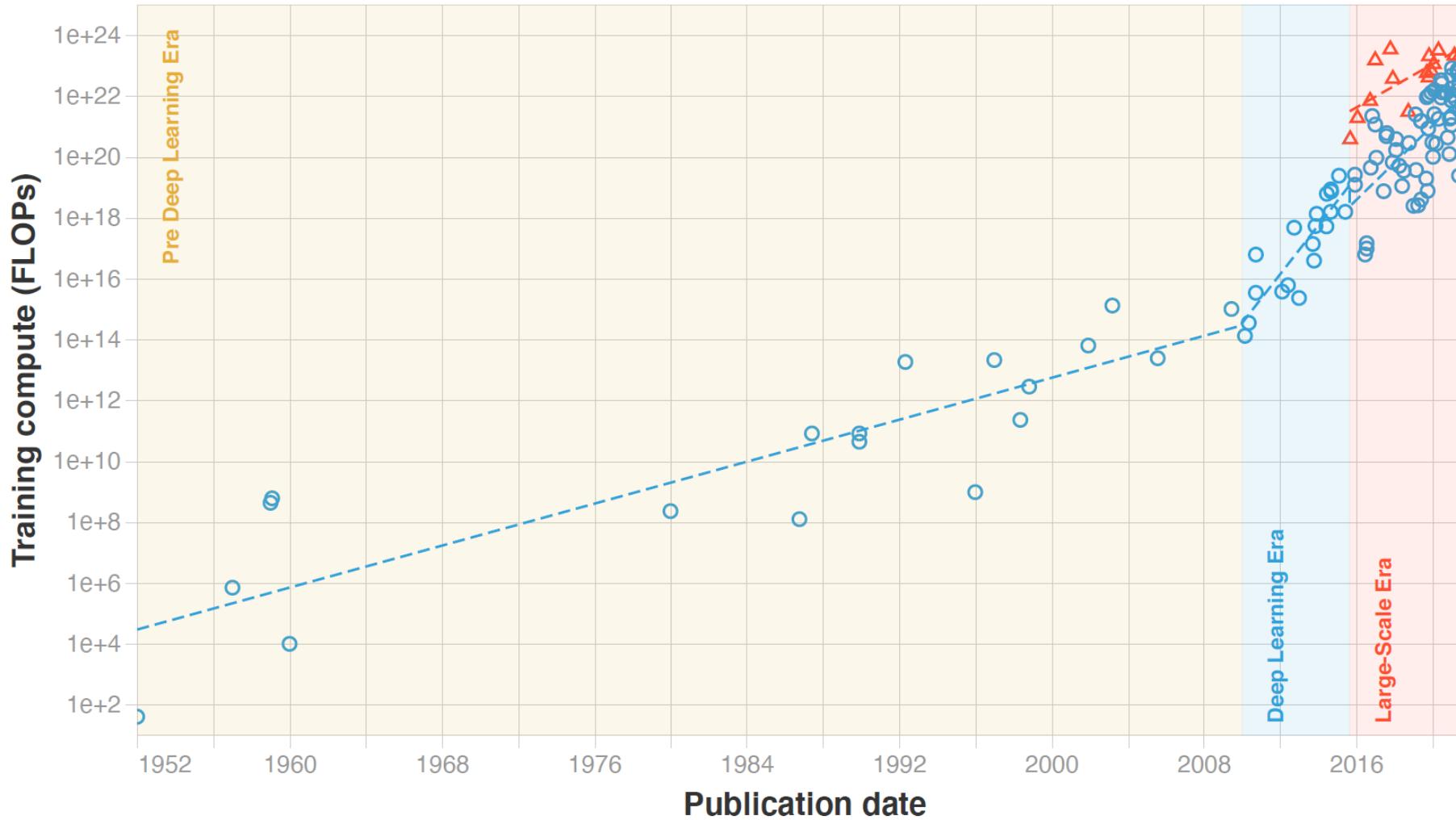
Copyright © European Processor Initiative 2019. EPI Tutorial/Barcelona/17-07-2019



HPC: 10x every 4 years

Computing is Power Bound: ML

(Largest datacenter <150MW)

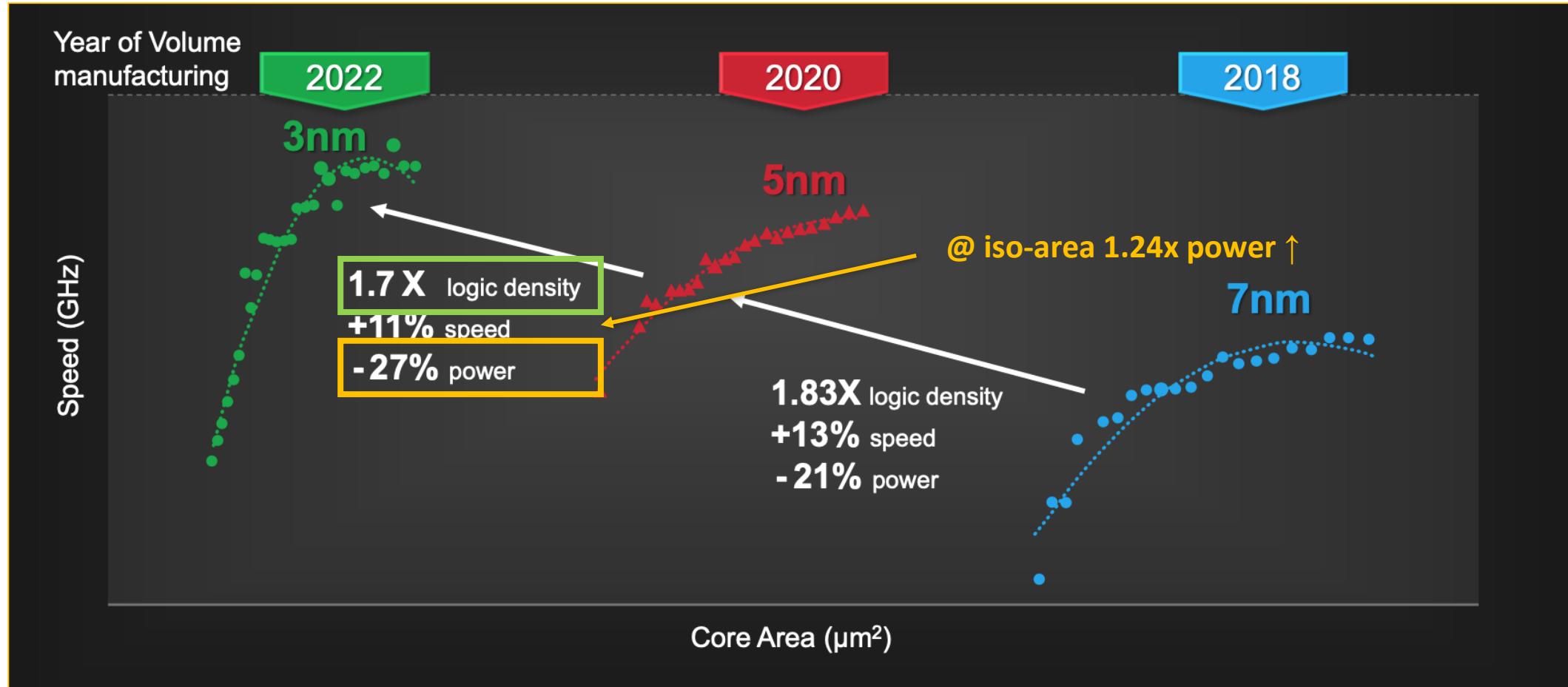


Sevilla 22: arXiv:2202.05924

Machine Learning (training): 10x every 2 years

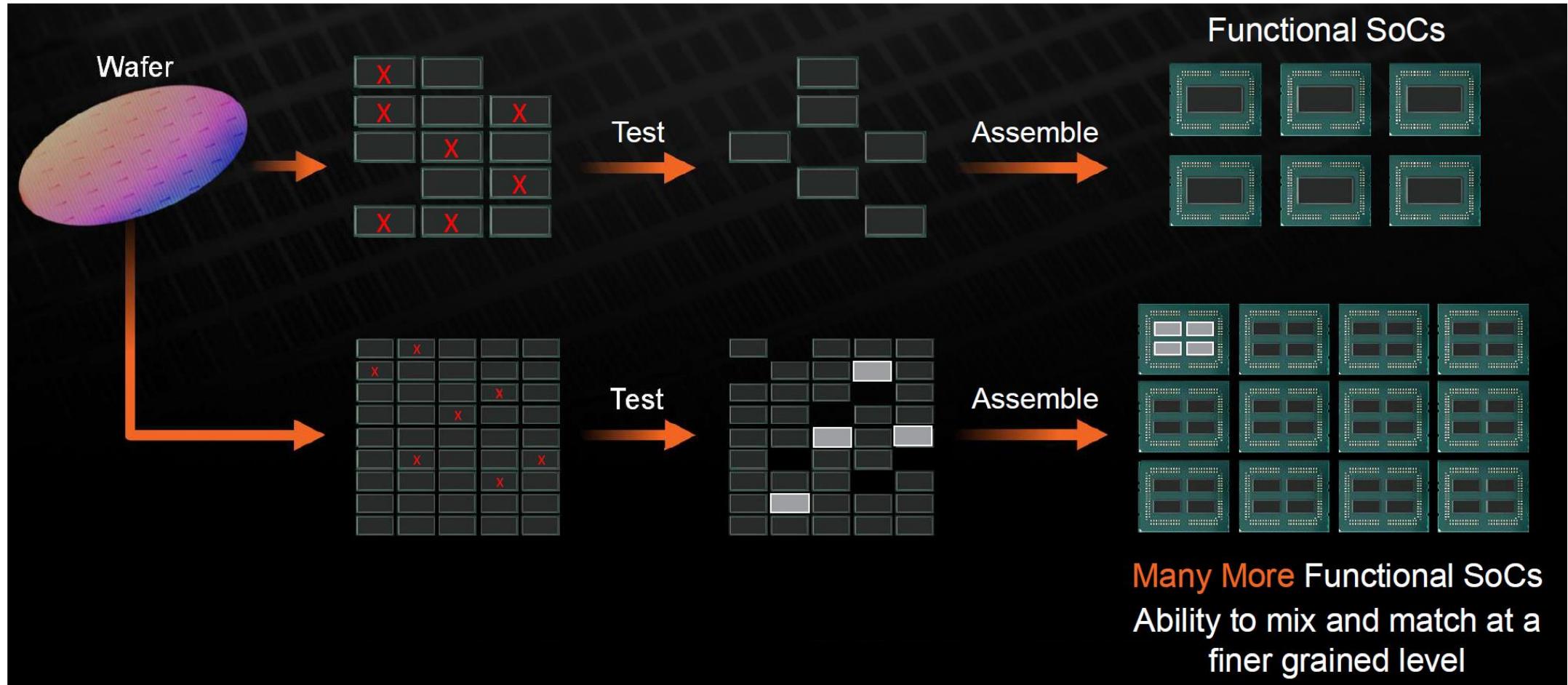
Technology Scaling?

TSMC, ISSCC21



Chiplets?

AMD, ISCAS22



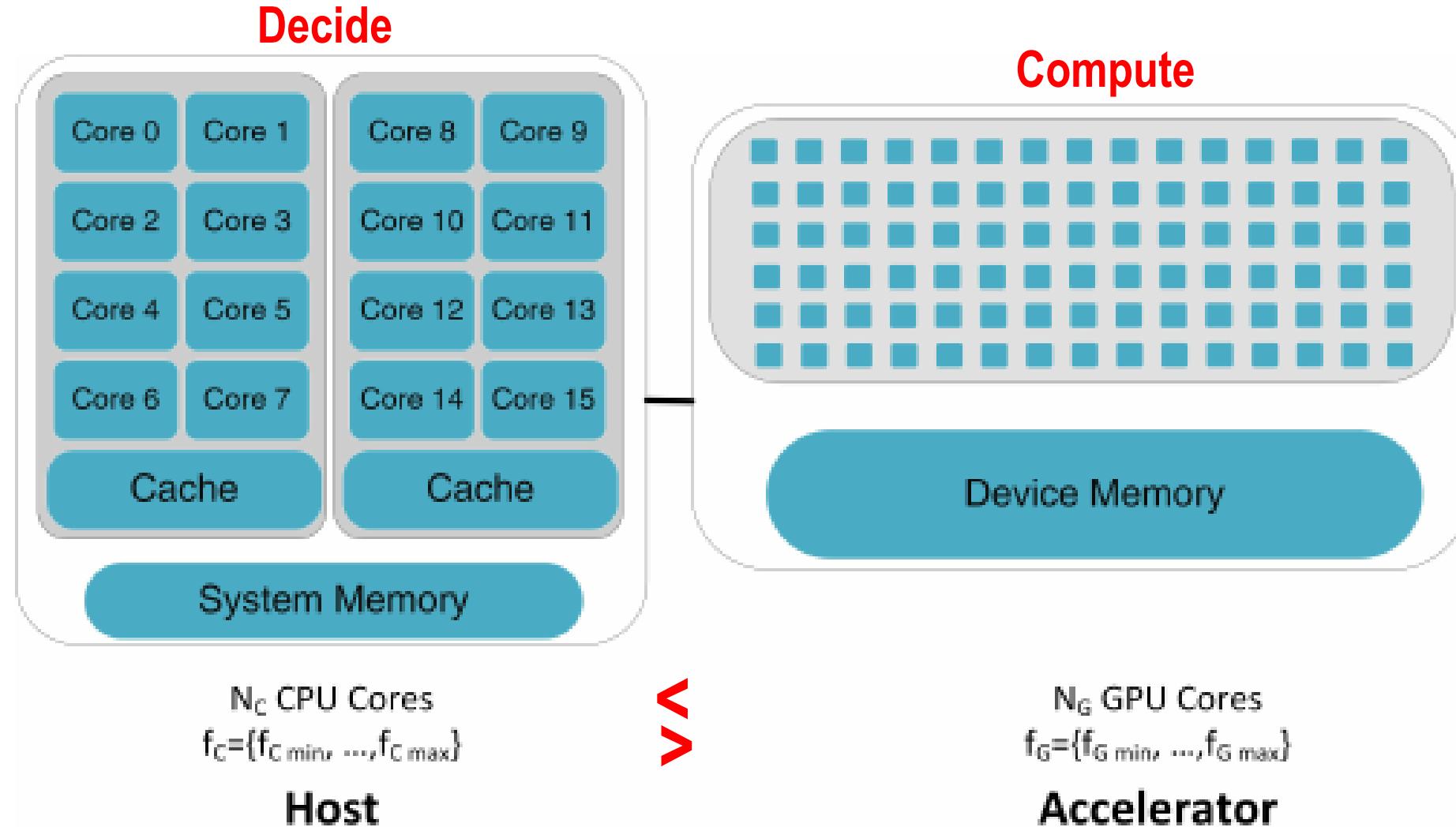
Good news for Cost+Density, no silver bullet for Energy



ETH zürich



Efficient Architecture: Heterogeneous+Parallel



Heterogeneous + Parallel... Why?

- Processors can do two kinds of useful work:

Decide (jump to different program part)

- Modulate flow of **instructions**
- Mostly sequential decisions:
 - Don't work too much
 - Be clever about the battles you pick
(latency is king)
- **Lots of decisions**
Little number crunching

Compute (plough through numbers)

- Modulate flow of **data**
- Embarrassing data parallel:
 - Don't think too much
 - Plough through the data
(throughput is king)
- **Few decisions**
Lots of number crunching

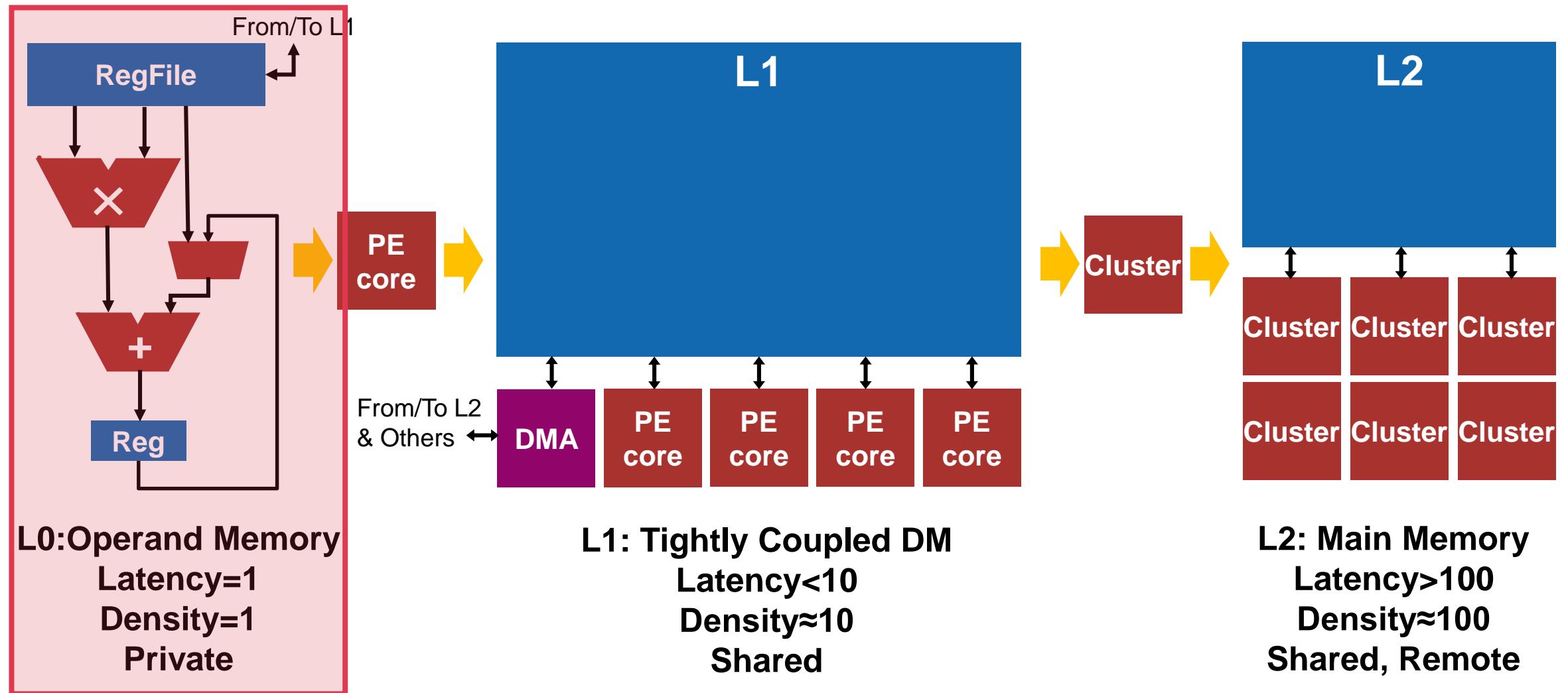
- Today's workloads are dominated by “Compute”:

- Tons of data, few (as fast as possible) decisions based on the computed values,
- **“Data-Oblivious Algorithms” (ML, or better DNNs are so!)**
- Large data footprint + sparsity

How to design an efficient “Compute” fabric?

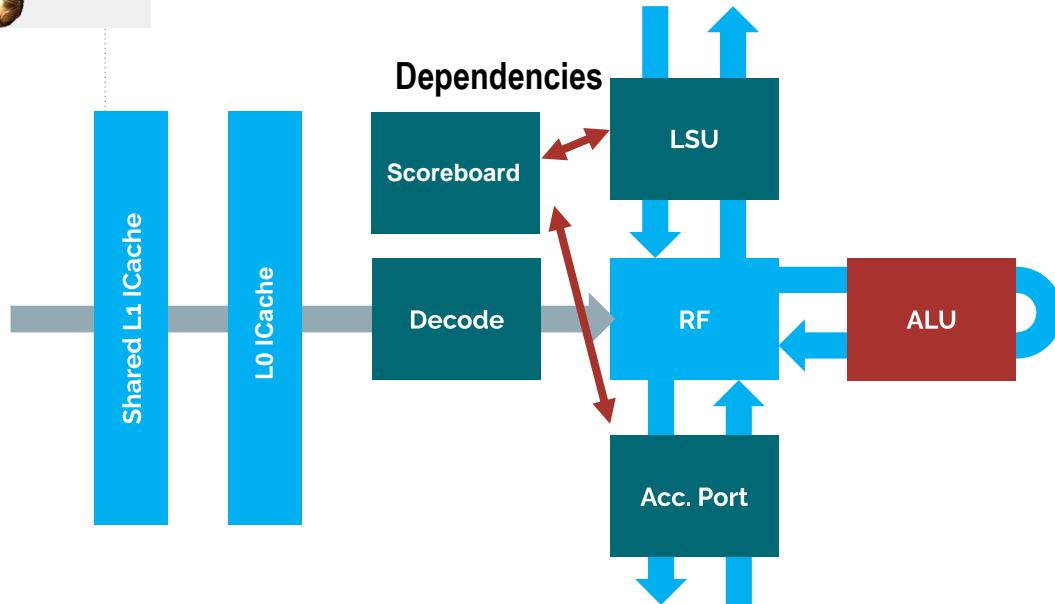


Compute Efficiency for PE, Cluster, SoC



PE: Snitch, a Tiny RISC-V Control Core

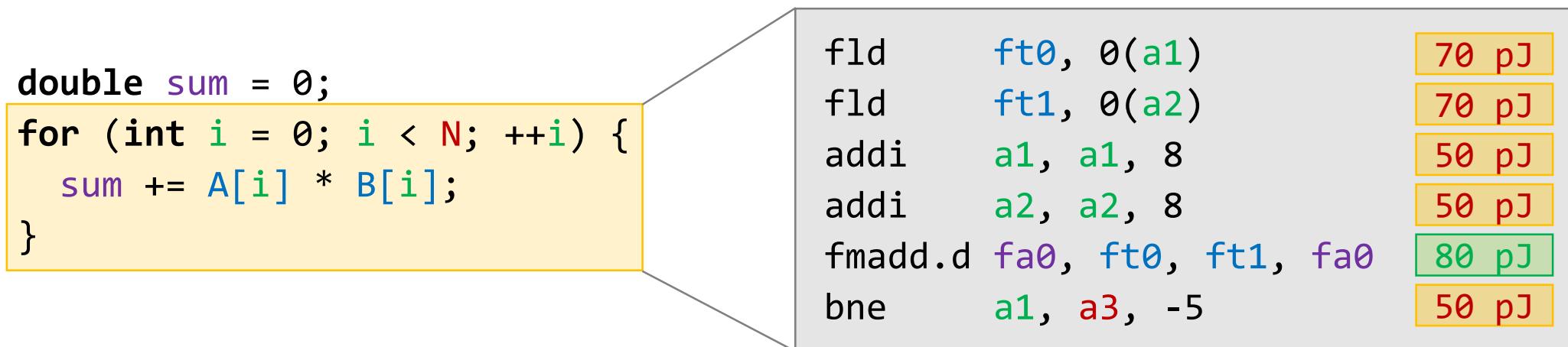
A versatile building block



- **Simplest core: around 20KGE**
 - Speed via simplicity (1GHZ+)
 - L0 Icache/buffer for low energy fetch
 - Shared L1 for instruction reuse (SPMD)
- **Extensible → Accelerator port**
 - Minimal baseline ISA (RISC-V)
 - Extensibility: Performance through ISA extensions (via accelerator port)
- **Latency-tolerant →Scoreboard**
 - Tracks instruction dependencies
 - Much simpler than OOO support!

Snitch PE: ISA Extension for efficient “Compute”

- How can we remove the Von Neumann Bottleneck?
- Targeting “compute” code



Memory access, operation, iteration control – can we do better?

Note: memory access (>1 cycle even for L1) → need latency tolerance for LD/ST

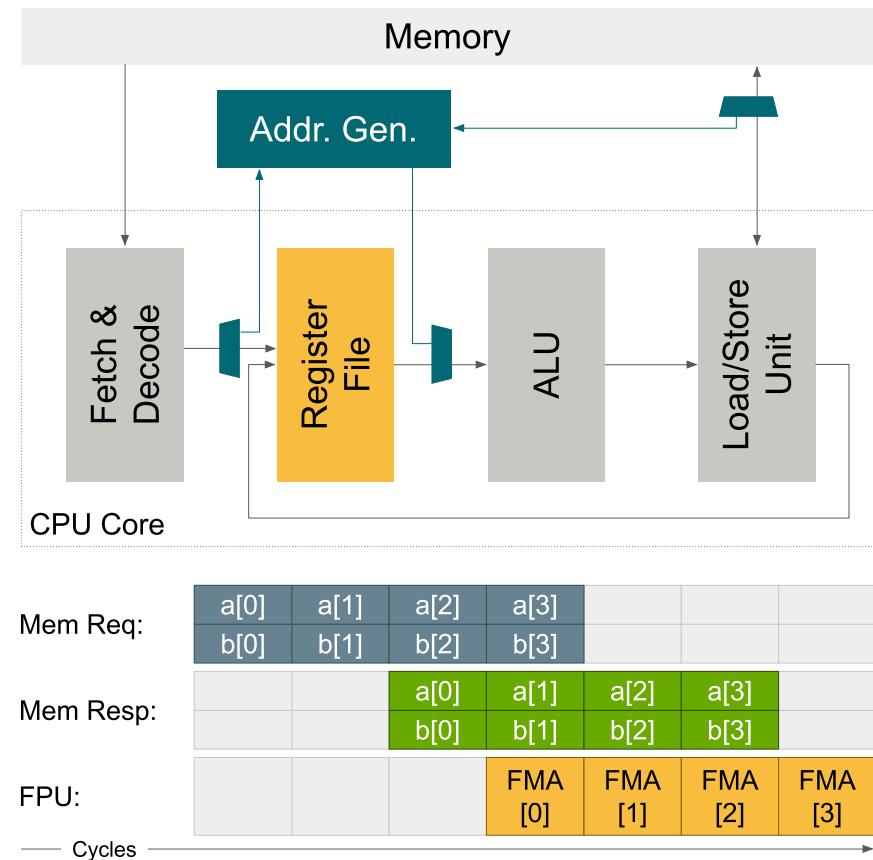
Stream Semantic Registers

LD/ST elision

- Intuition: High FPU utilization \approx high energy-efficiency
- Idea: Turn register read/writes into implicit memory loads/stores.
- Extension around the core's register file
- Address generation hardware

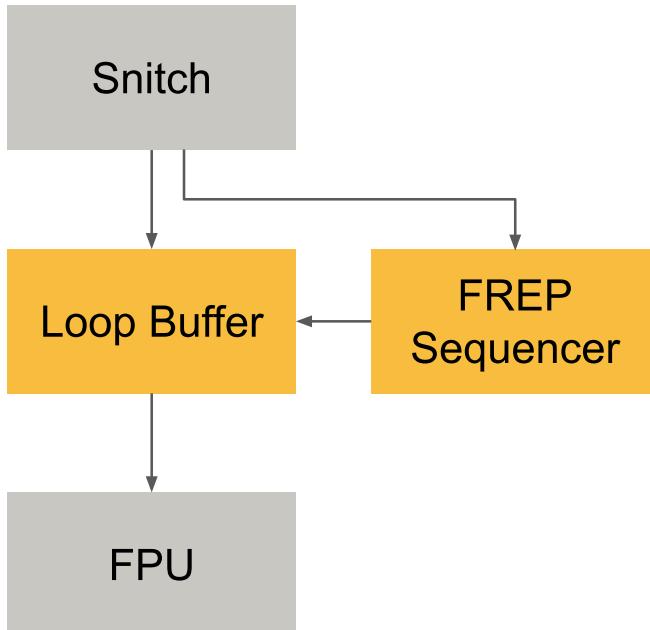
```
loop:  
  f1d r0, %[a]      scfg 0, %[a], ldA  
  f1d r1, %[b]      scfg 1, %[b], ldB  
  fmadd r2, r0, r1  
loop:  
  fmadd r2, ssr0, ssr1
```

- Increase FPU/ALU utilization by ~3x up to 100%
- SSRs \neq memory operands
 - Perfect prefetching, latency-tolerant



Floating-point Repetition Buffer

Remove control flow overhead in compute stream



- Programmable micro-loop buffer
- Sequencer steps through the buffer, independently of the FPU
- Integer core free to operate in parallel:
Pseudo-dual issue
- High area- and energy-efficiency

```
mv r0, zero
loop:
  addi r0, 1
  fmadd r2, ssr0, ssr1
  bne r0, r1, loop
```

→

```
frep r1, 1
loop:
  fmadd r2, ssr0, ssr1
```

SNITCH vs GPU PE

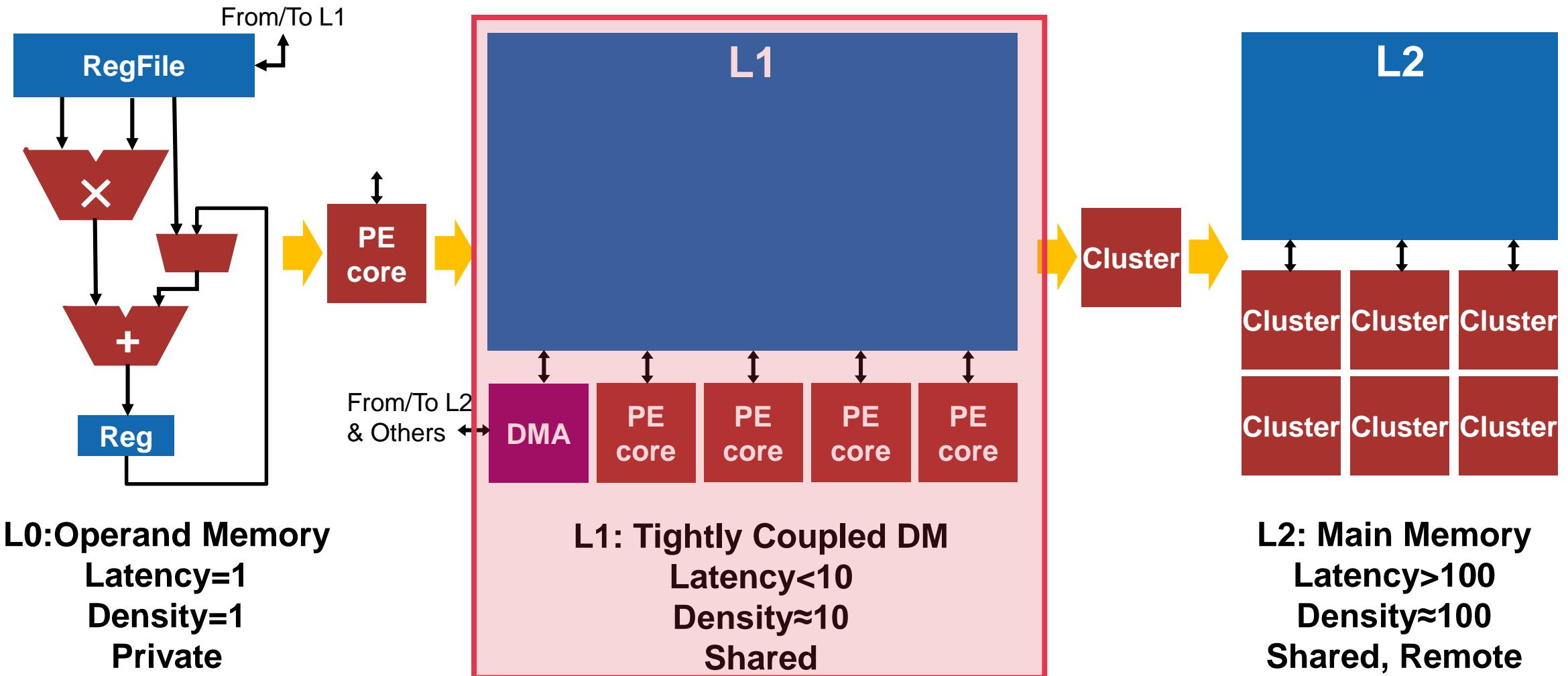
- Snitch dissipates significant fraction of power in its FPU (more is better):
 - GPUs not much (5% reported in [1])
- Compared to NVIDIA GPU :
 - Register file in GPU holds registers and thread-local data
 - Each register read/write is an SRAM access
 - Data *and* register accesses are in SRAM

GPU Assembly	Snitch Assembly
LDS R2, [R0]	FMAdd R2, SSR0, SSR1
LDS R3, [R1]	
FFMA R4, R2, R3, R2	
2 mem. acc. ("[...]"")	2 mem. acc. ("")
8 reg. acc.	4 reg access (no load instructions)
= 10 SRAM hits total	
= 2 SRAM hits total	

Efficient PE architecture in perspective

1. Minimize control overhead → Simple, shallow pipelines
2. Reduce VNB → amortize IF: SSR-FREP + SIMD (Vector processing)
3. Hide memory latency → non-blocking LD/ST+dependency tracking
4. Highly expressive, domain-specific instructions with SPUs

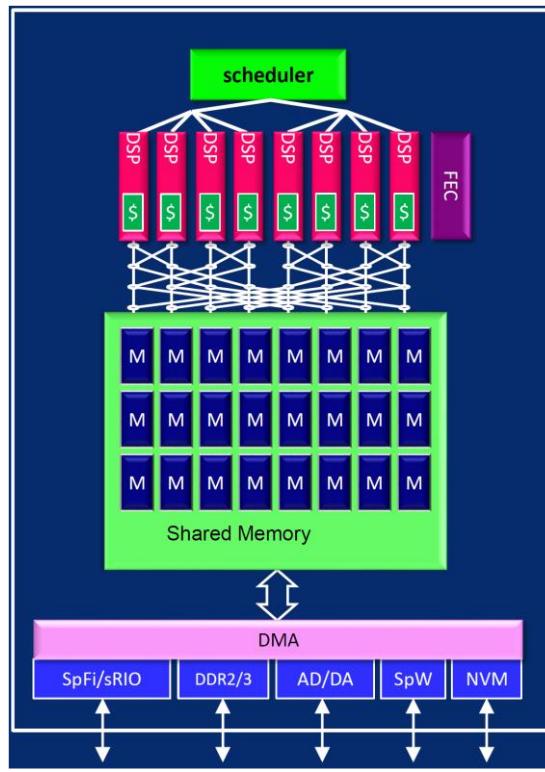
Compute Efficiency for PE, Cluster, SoC



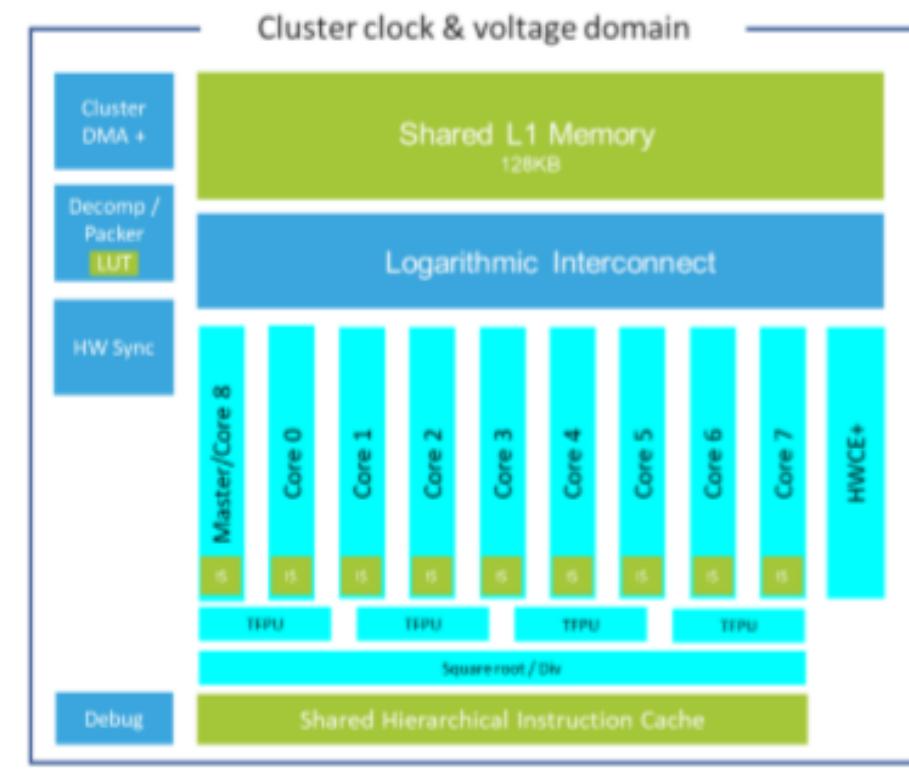
Tightly-Coupled Processors Clusters



HPC: Ampere's SP



HPE: RamonChips RC64

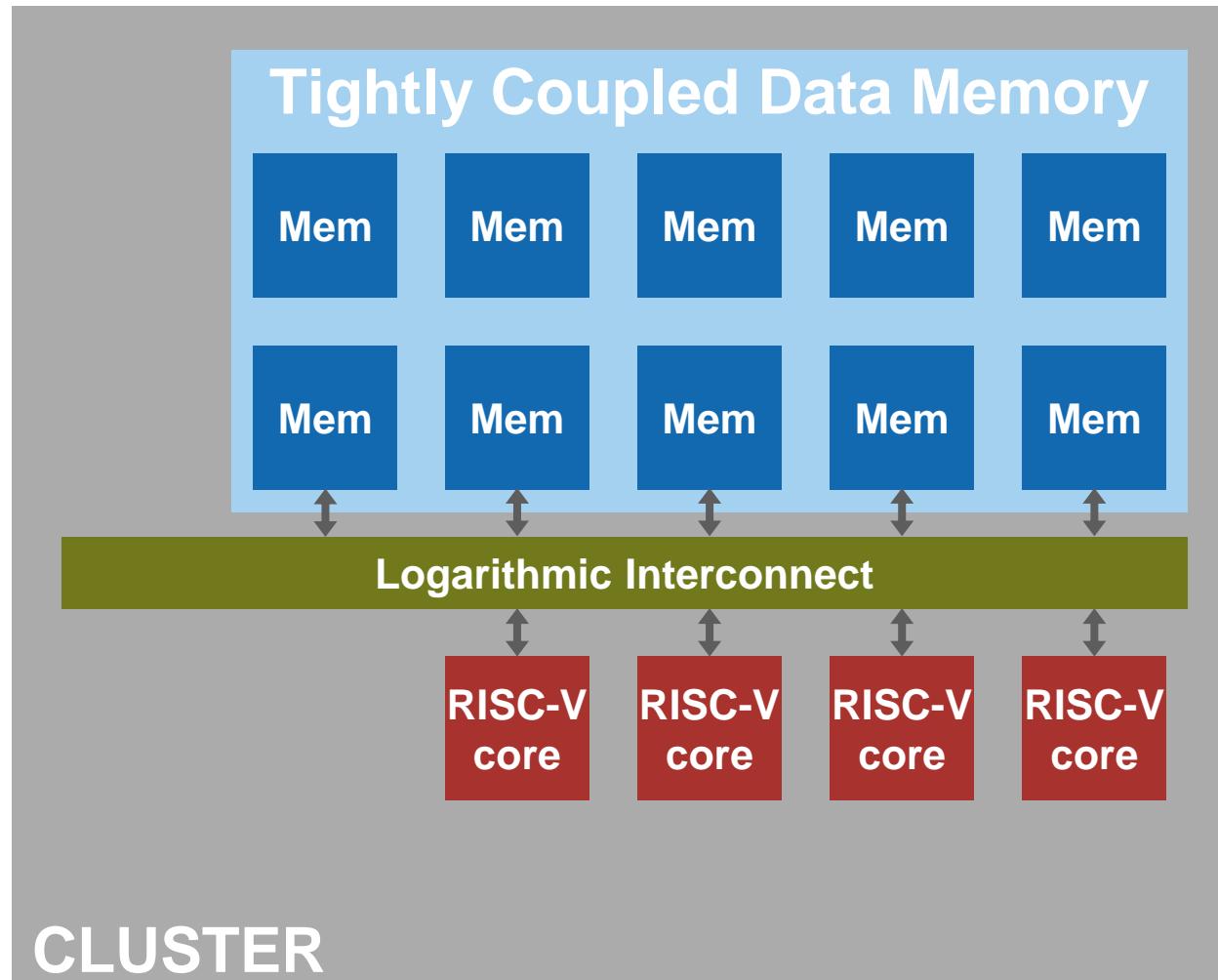


IoT: Greenwave's GAP9

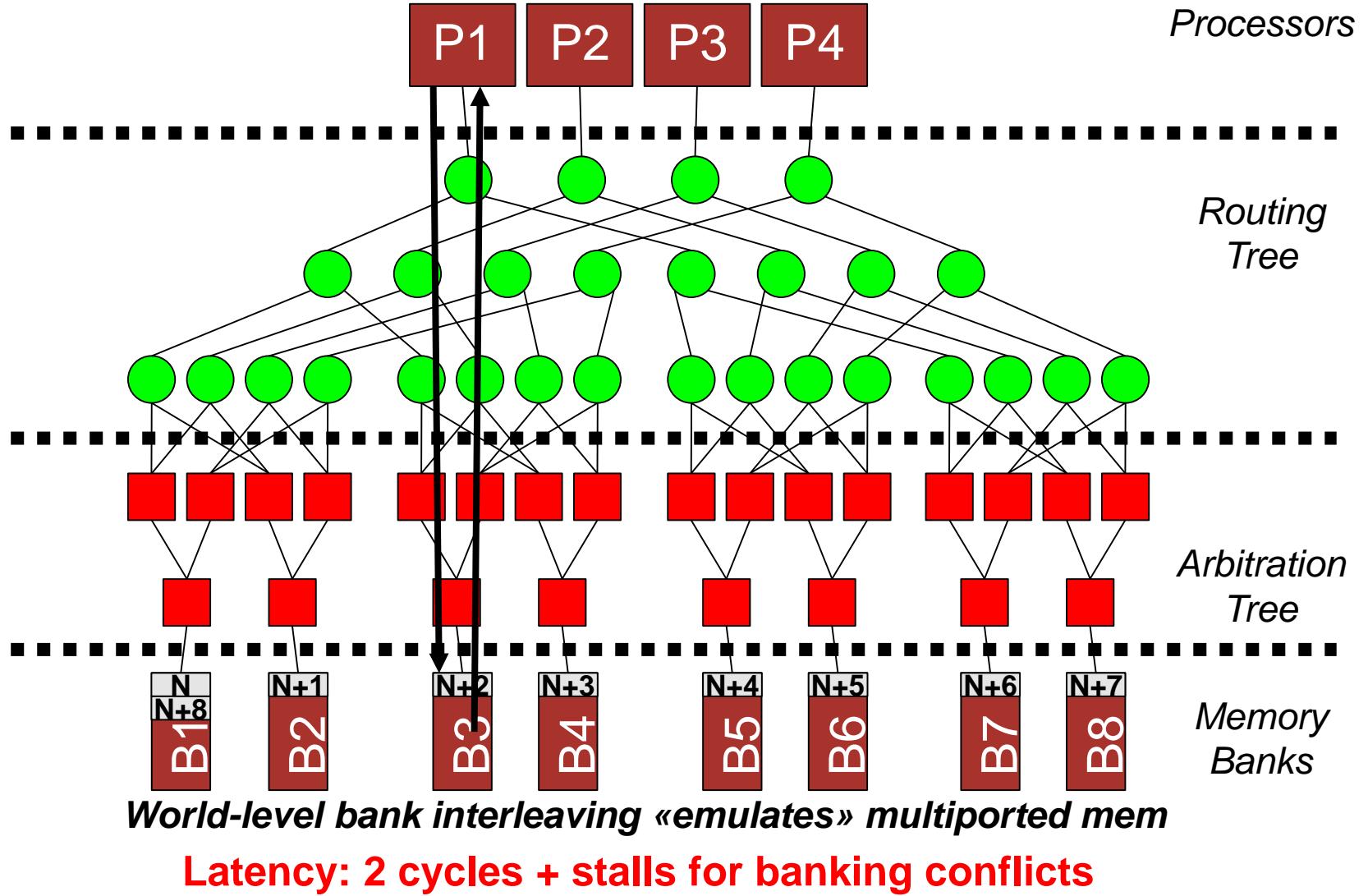
Pervasive Architectural Pattern for Data-Parallel Computing!

The Cluster: Design Challenges

- **Low latency access TCDM**
 - Multi-banked architecture
 - Fast logarithmic interconnect
- **Fast synchronization**
 - Atomics
 - Barriers
- **Efficient PE**
 - Hide TCDM “residual” latency
 - Remove Von Neumann Bottleneck

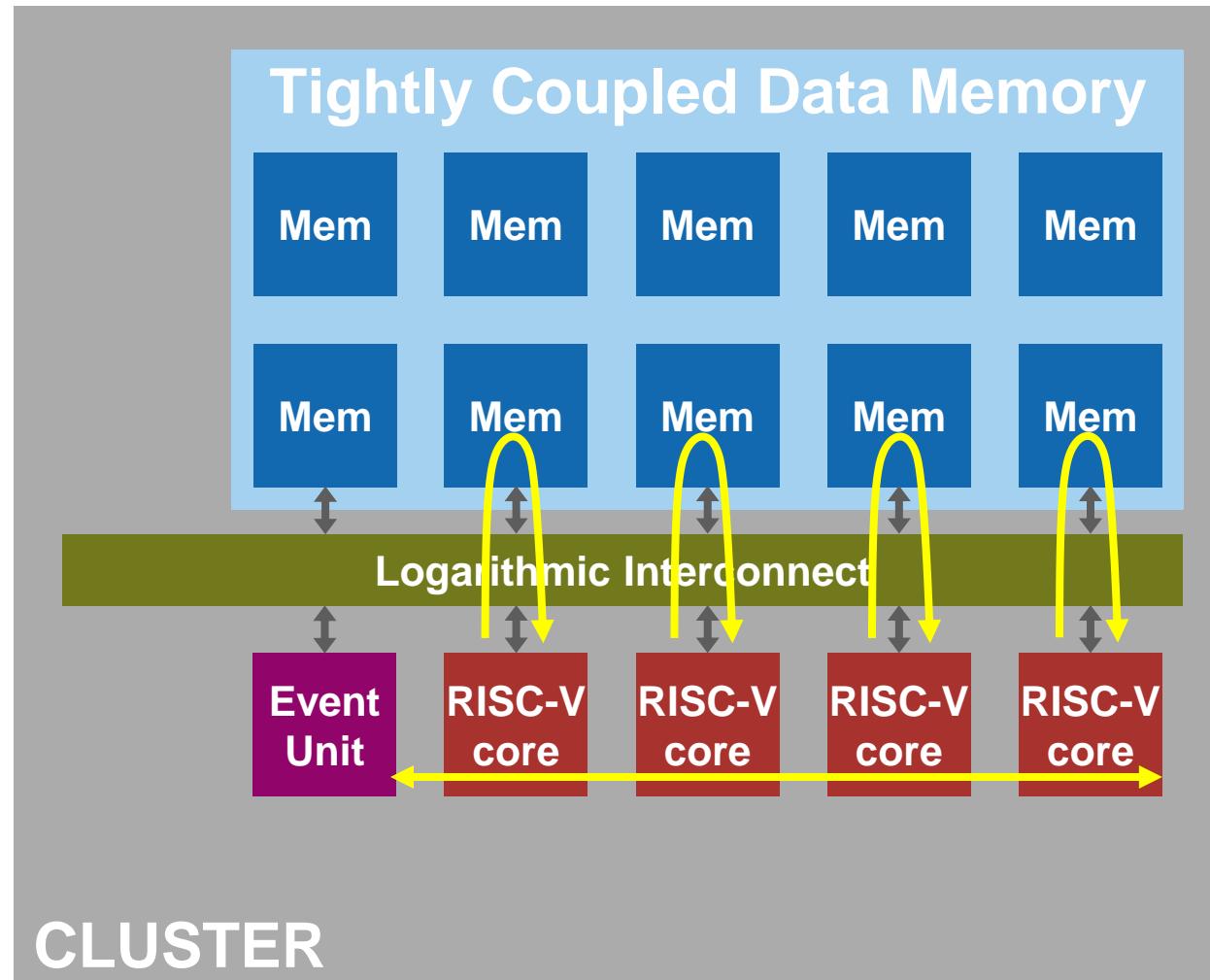


High speed logarithmic interconnect

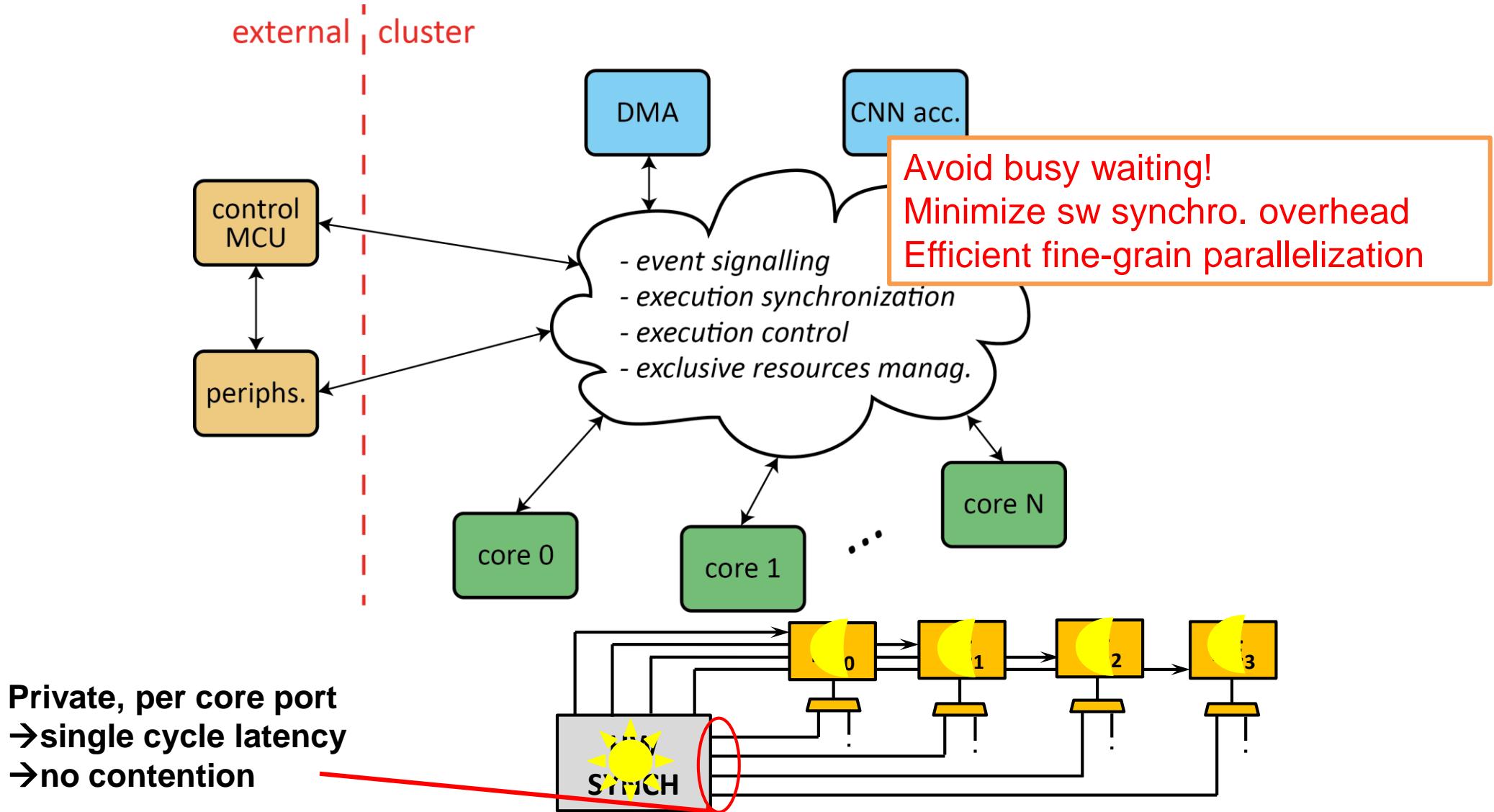


Fast synchronization and Atomics

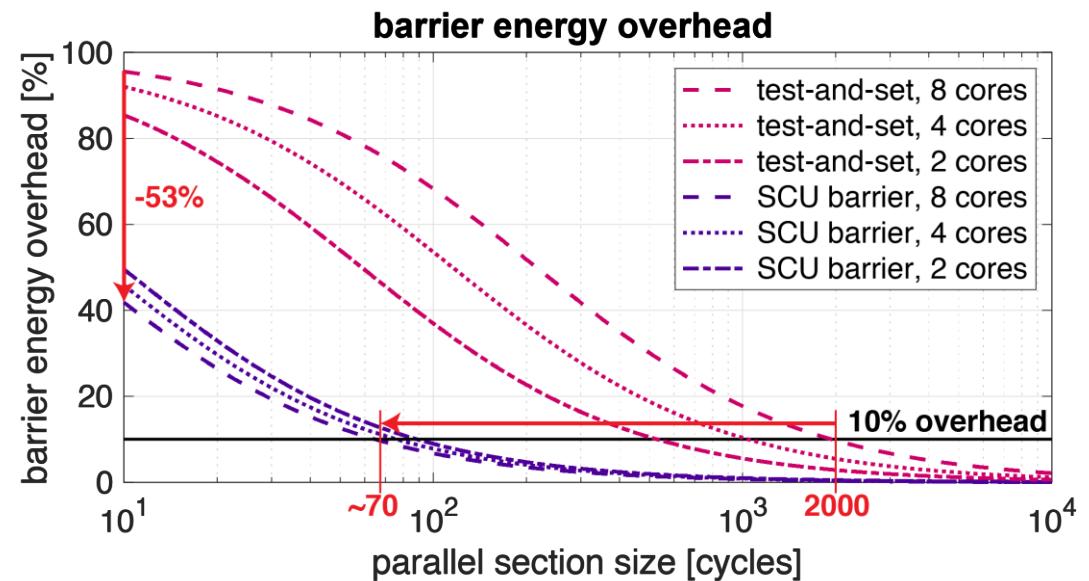
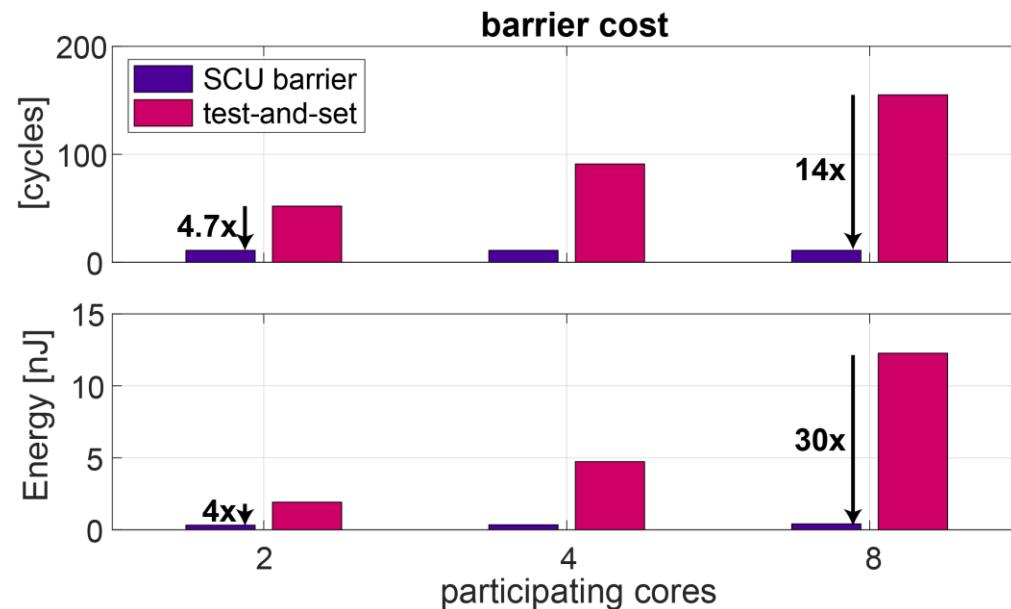
- **Atomic instructions**
 - Leverage fast memory access
 - Based on per-bank atomic adapters
- **Fast synchronization**
 - Atomics for generic primitives
 - Accelerating barriers in HW (make the common case fast)
 - Minimize idle power while waiting



Hardware-Accelerated, Event-Based Barrier



Barrier: Results



- Fully parallel access to SCU: Barrier cost constant
- Primitive energy cost: Down by up to 30x
- Minimum parallel section for 10% overhead in terms of ...
 - ... cycles: ~100 instead of > 1000 cycles
 - ... energy: ~70 instead of > 2000 cycles

Efficiently Global Data Mover

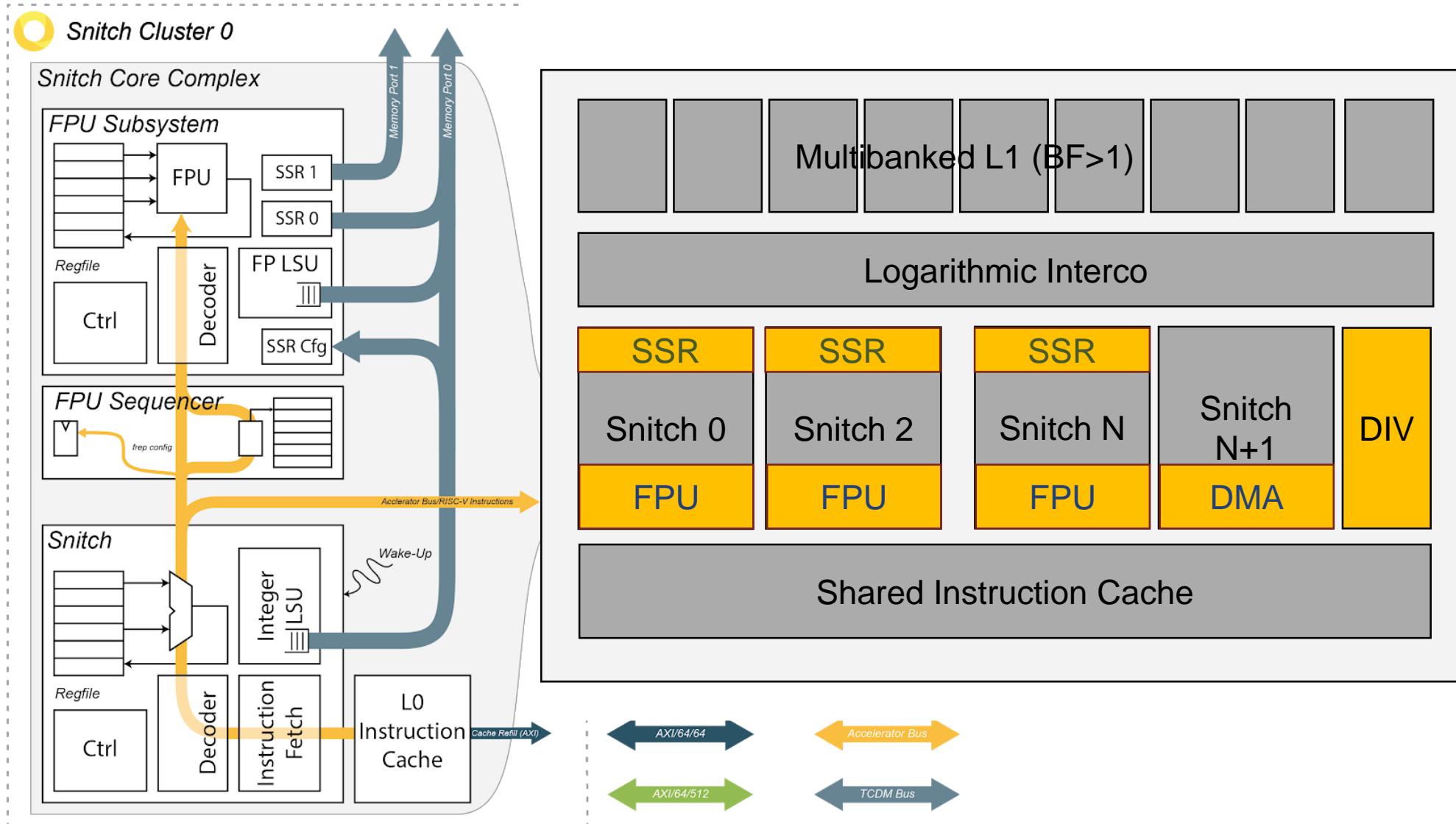
hide L2/main memory latency



- 64-bit AXI DMA – explicit double-buffered transfers – **better than D\$**
- Tightly coupled with Snitch (<10 cycles configuration)
- Operates on wide 512-bit data-bus
- Hardware support to copy 2D shapes
- Higher-dimensionality handled by SW
- Intrinsics/library for easy programming

```
// setup and start a 1D transfer, return transfer ID
uint32_t __builtin_sdma_start_oned(
    uint64_t src, uint64_t dst, uint32_t size, uint32_t cfg);
// setup and start a 2D transfer, return transfer ID
uint32_t __builtin_sdma_start_twod(
    uint64_t src, uint64_t dst, uint32_t size,
    uint32_t sstrd, uint32_t dstrd, uint32_t nreps, uint32_t cfg);
// return status of transfer ID tid
uint32_t __builtin_sdma_stat(uint32_t tid);
// wait for DMA to be idle (no transfers ongoing)
void __builtin_sdma_wait_for_idle(void);
```

Snitch Cluster Architecture

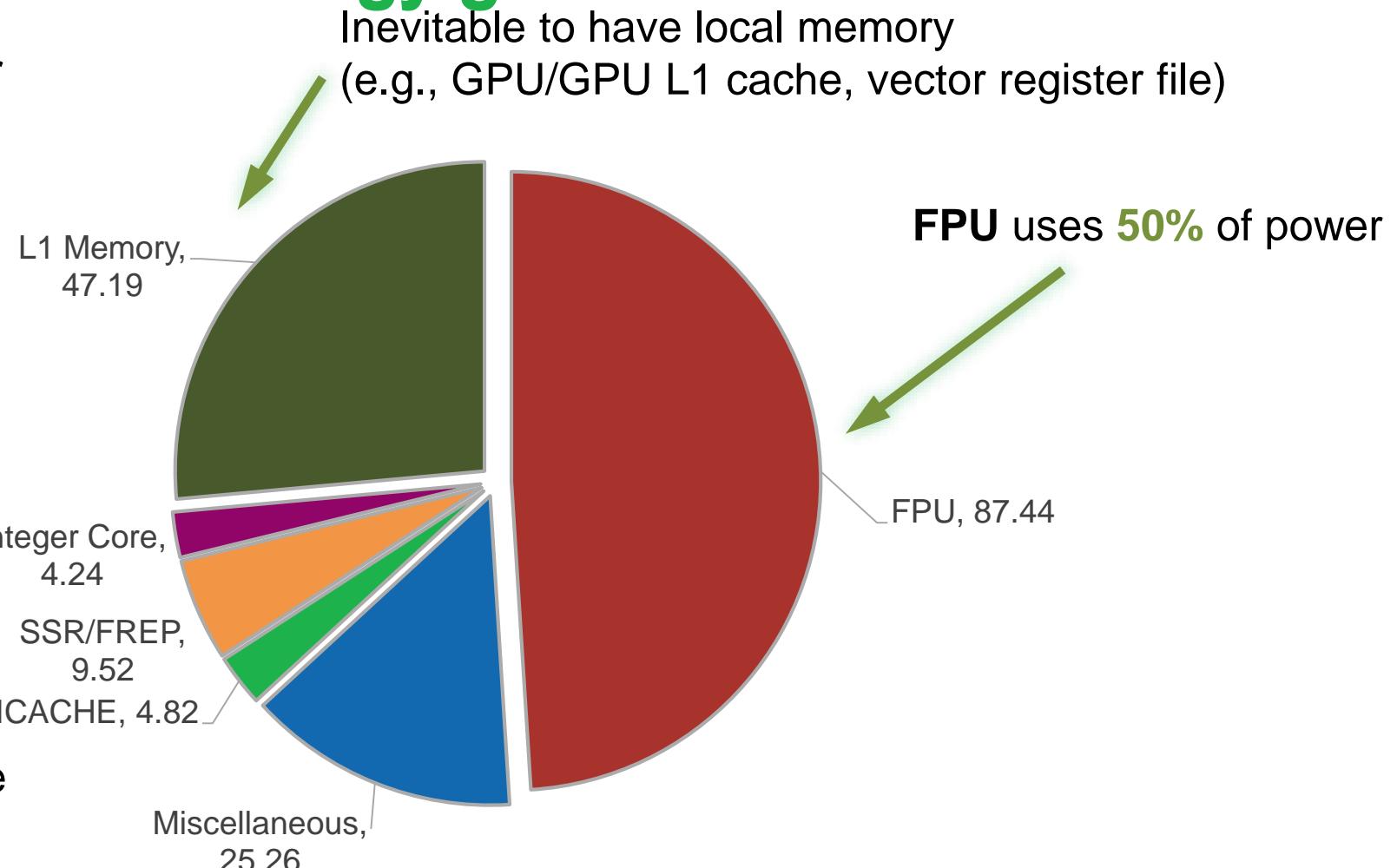


Where does the Energy go?

In an 8-core cluster

Integer core uses
2% of power

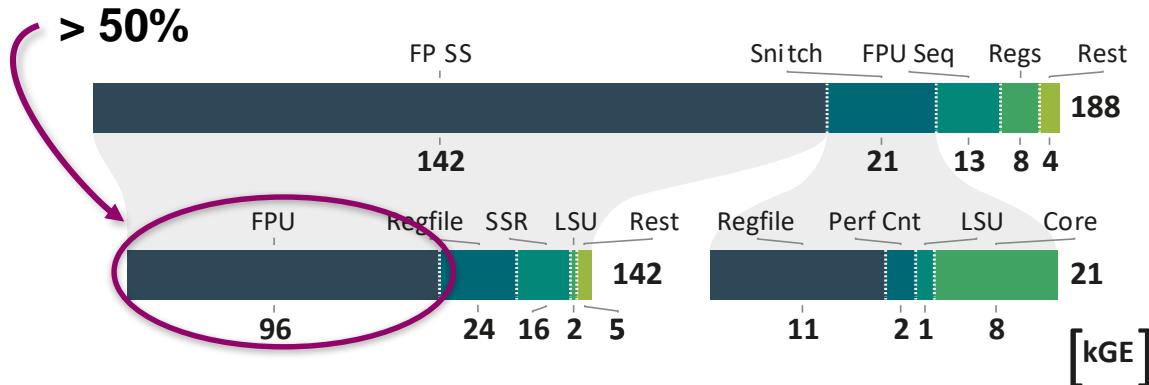
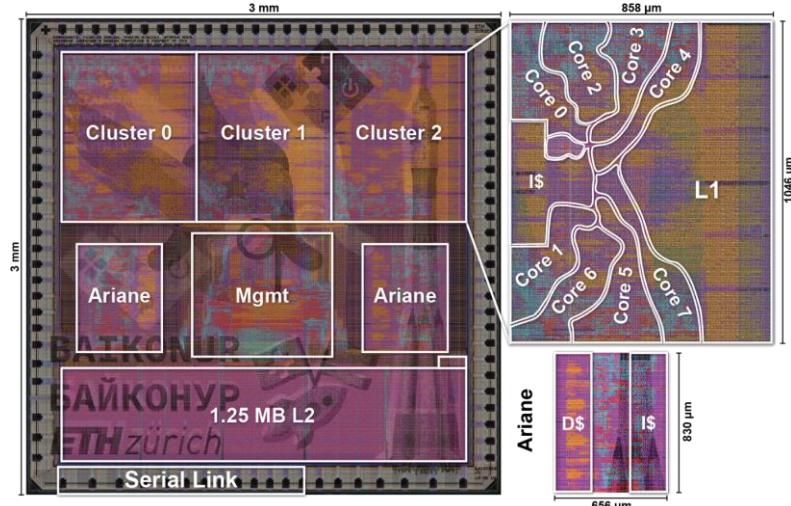
SSR/FREP hardware
uses 5% of power



Spending energy where it contributes to the result → **High Efficiency**

Baikonur

The first Snitch-based test chip.



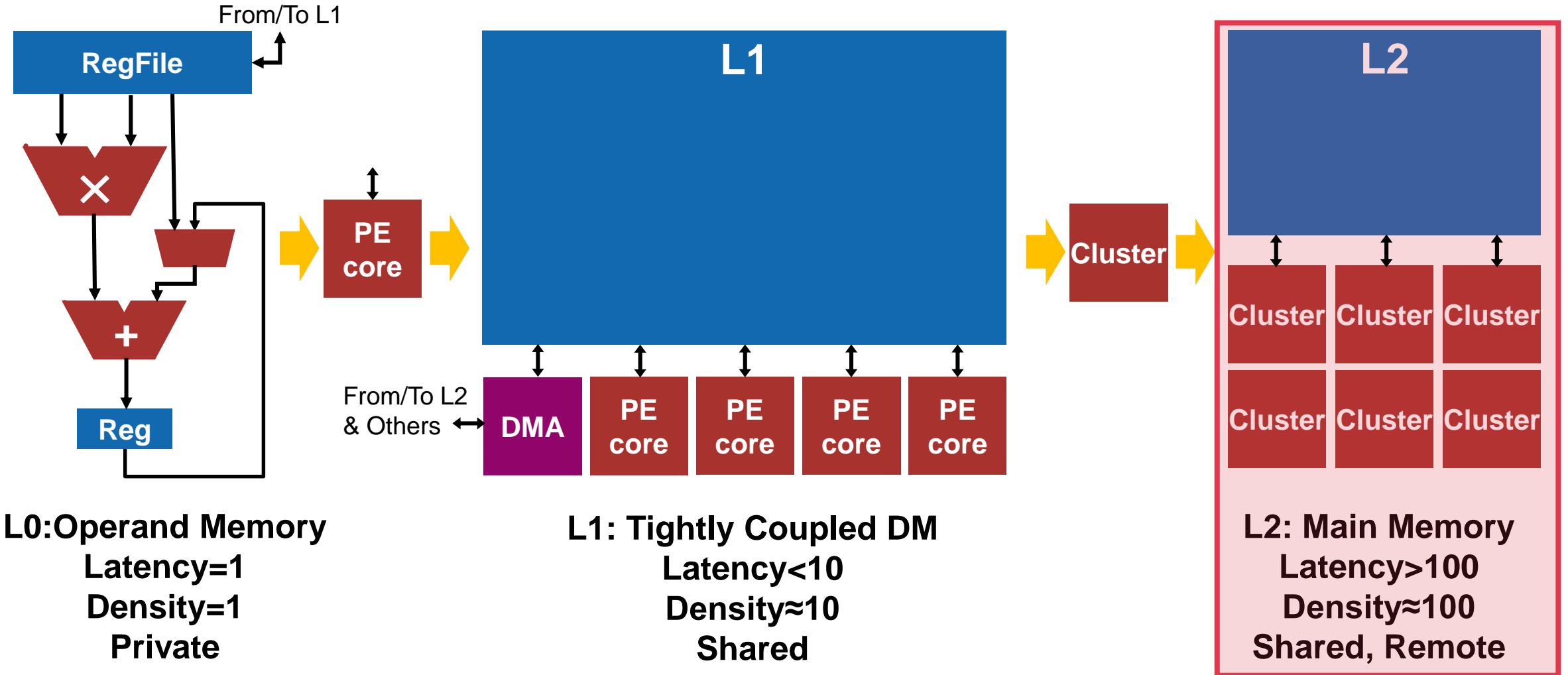
- **Snitch compute cluster:**
 - 8x RV32G Snitch cores
 - 8x large FPUs
 - > 40% energy spent on FPU
 - High energy-efficiency of ~80 Gdpflop/s/W
 - 104 Gspflop/s/W similar to accelerators
- **9 mm² prototype in GF 22FDX**
- **Testbed for key architectural components**
 - Snitch octa-core cluster
 - Ariane cores

Very efficient, versatile, compute cluster! How do we scale?

Efficient Cluster architecture in perspective

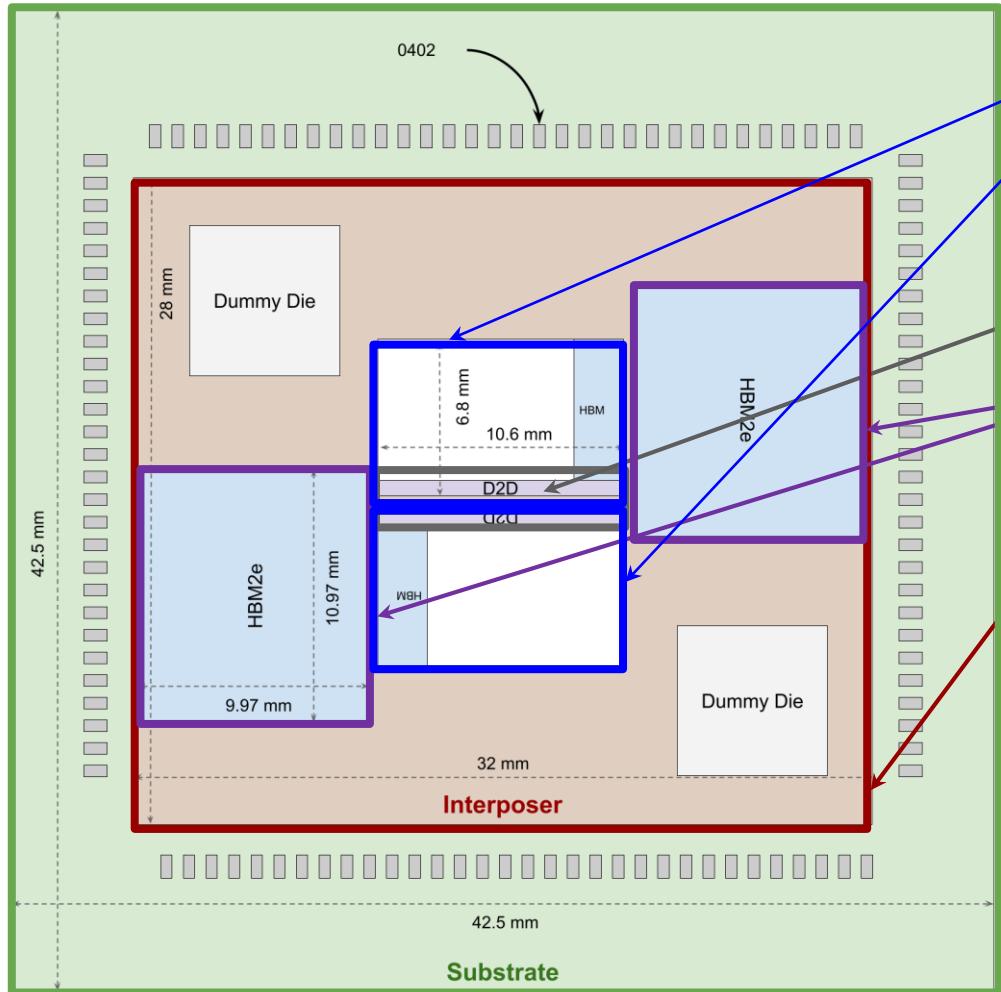
1. Memory pool – efficient sharing of L1 memory
2. Fast and parsimonious synchronization
3. Data Mover + Double buffering – explicitly managed block transfers at the boundary
4. **More cores and more memory per cluster... that would be nice!**

Compute Efficiency for PE, Cluster, SoC



Occamy: a 12LPP+ AI & Stream Computing Chiplet

Silicon implementation of all key IPs (core, Phys, chiplet)



Dual-chiplet configuration in 12LP+

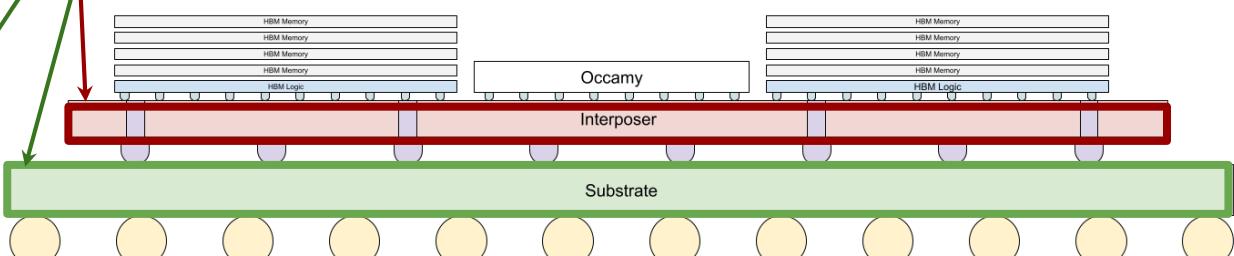
- Area: ~70mm²

Chip2chip link
HBM2e memories

Interposer

- 65nm tech, passive

Substrate



GlobalFoundries™
synopsys®
Rambus

From Snitch Cluster

8 Snitch compute cores

- Single-stage, small Integer control core

9th Core: DMA

- 512 bit data interface
- Efficient data movement

128 kB TCDM

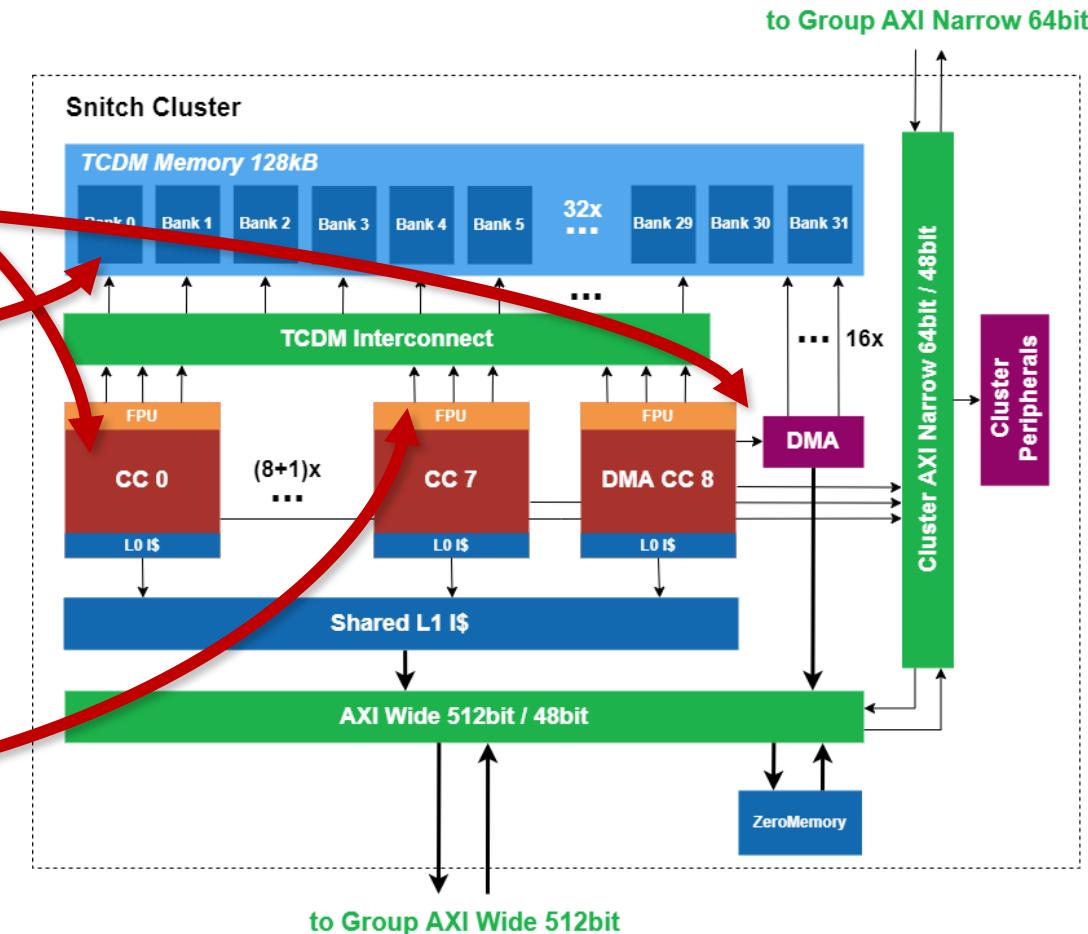
- Scratchpad for predictable memory accesses
- 32 Banks

Custom ISA extensions

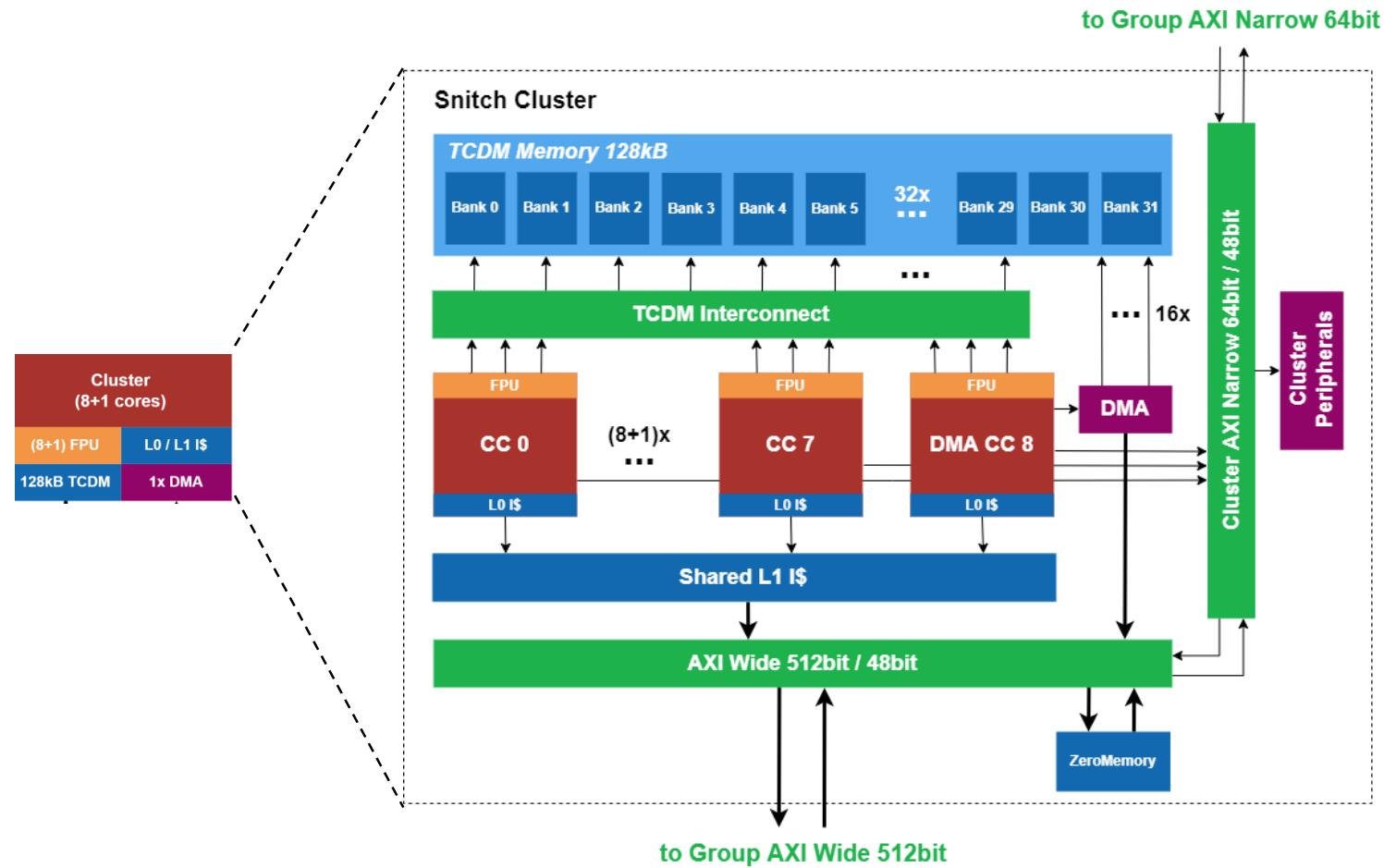
- Xfrep, Xssr
- **New: Xissr sparsity support**

1 FPU per Snitch core

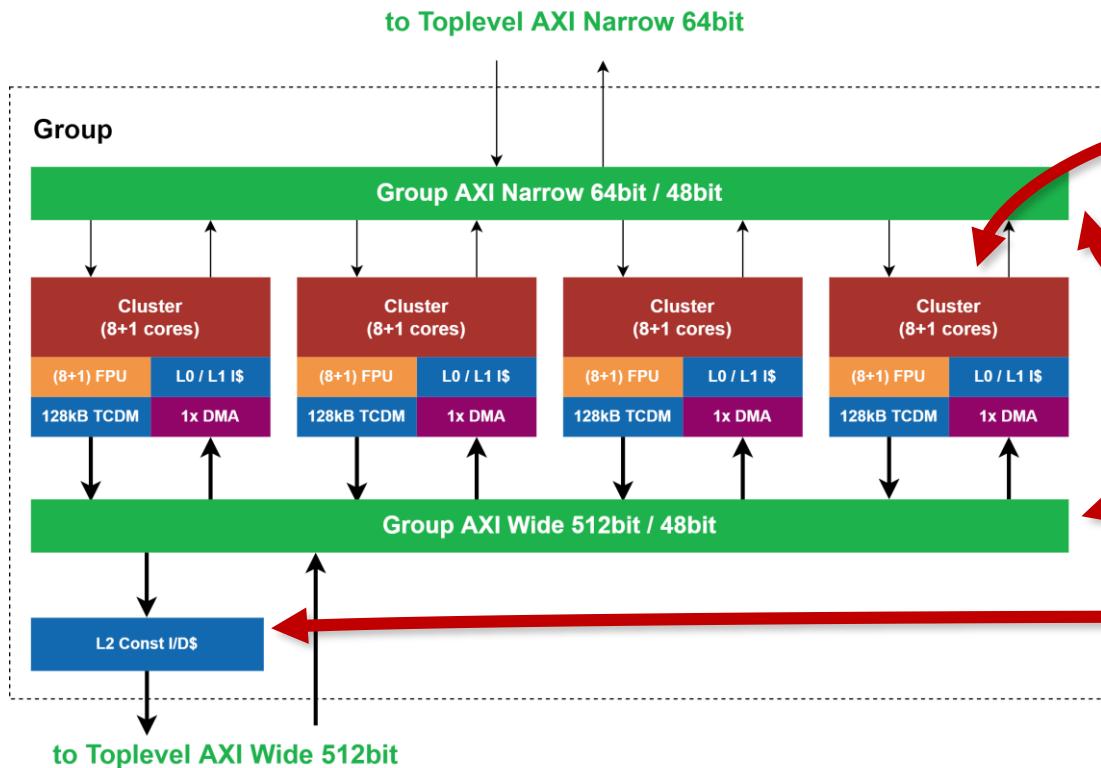
- Decoupled and heavily pipelined
- Multi-format FPU (+SIMD)
- **New: Minifloat support + SDOTP**



To Snitch Group



Snitch Group



4 Clusters per Group

- Single-stage, small Integer control core

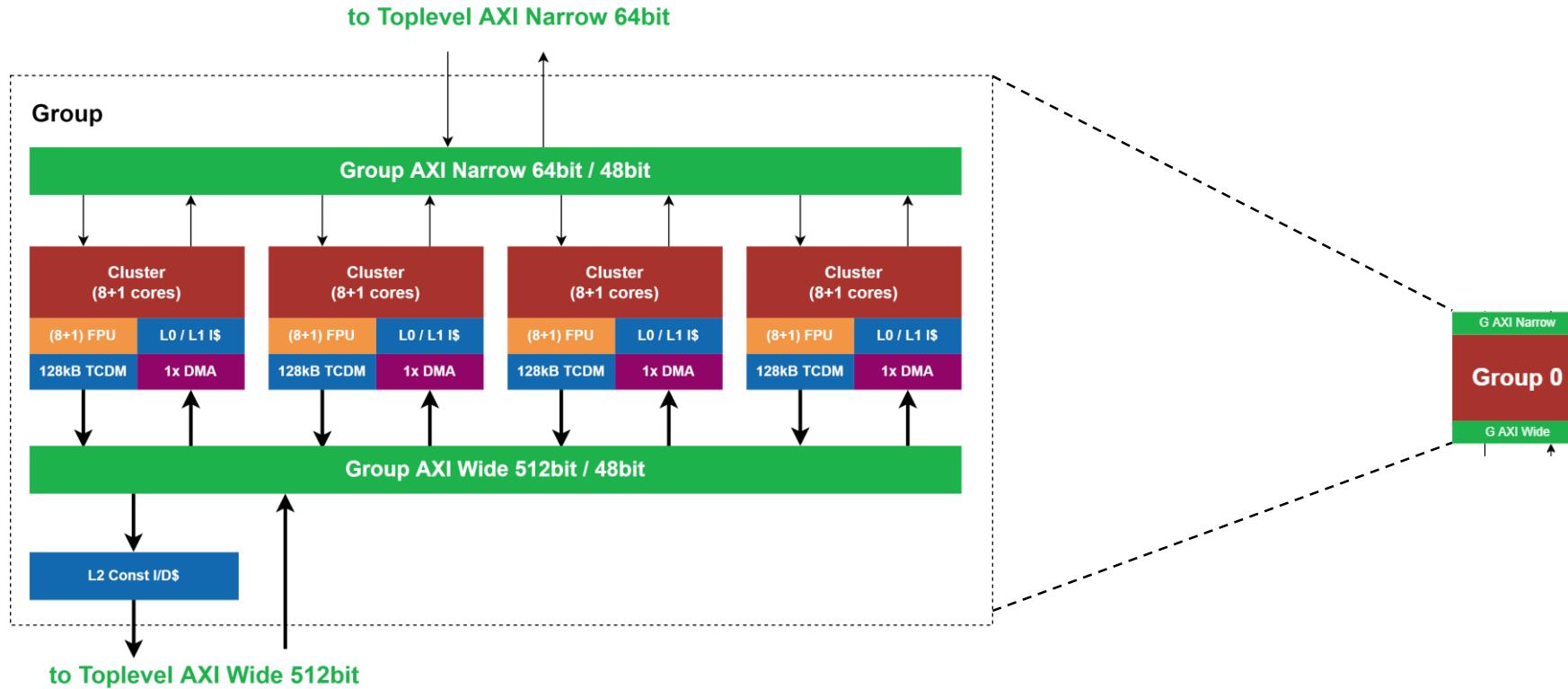
2 AXI Busses

- 64-bit narrow interface: config
- 512-bit wide interface: DMA

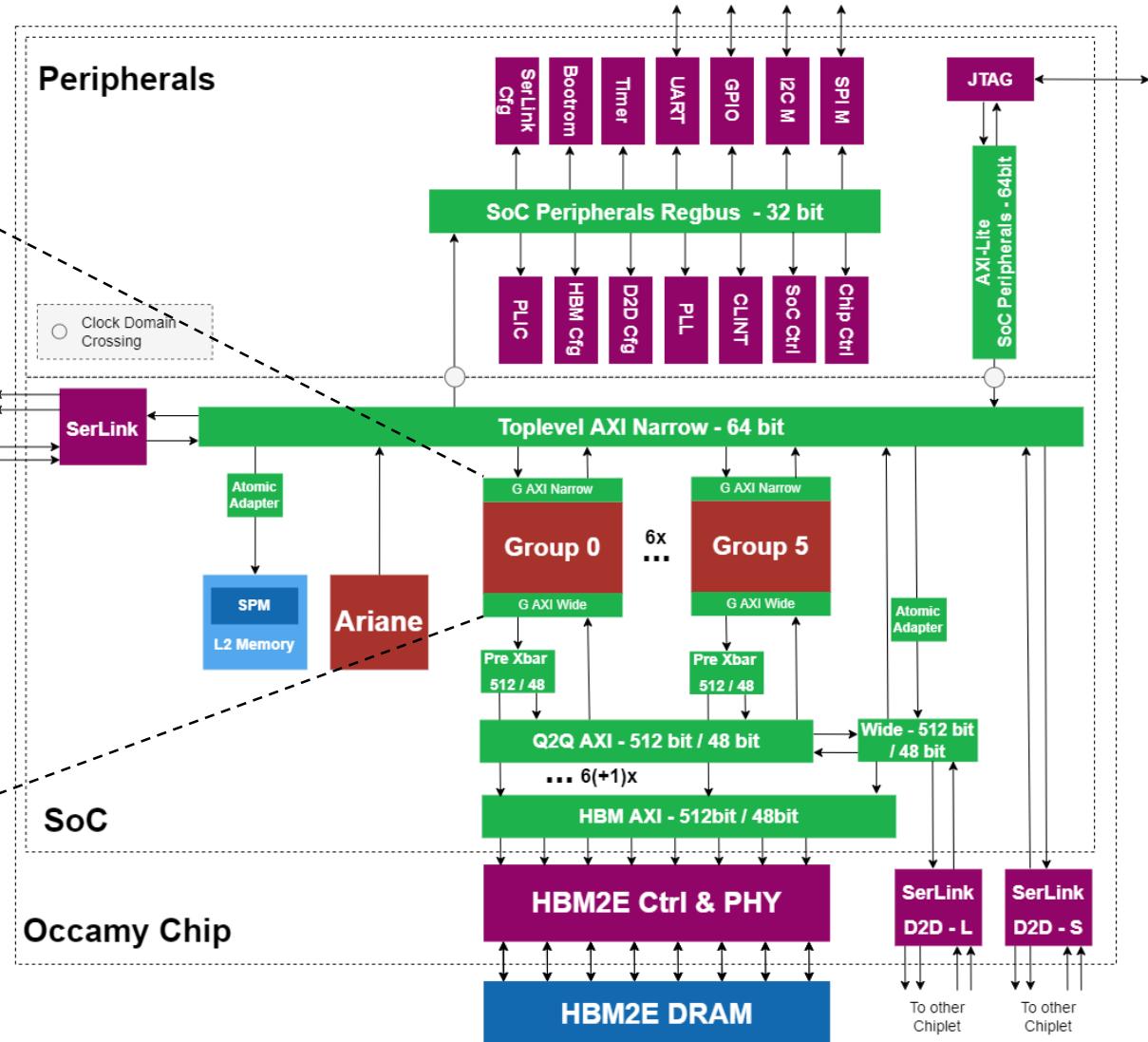
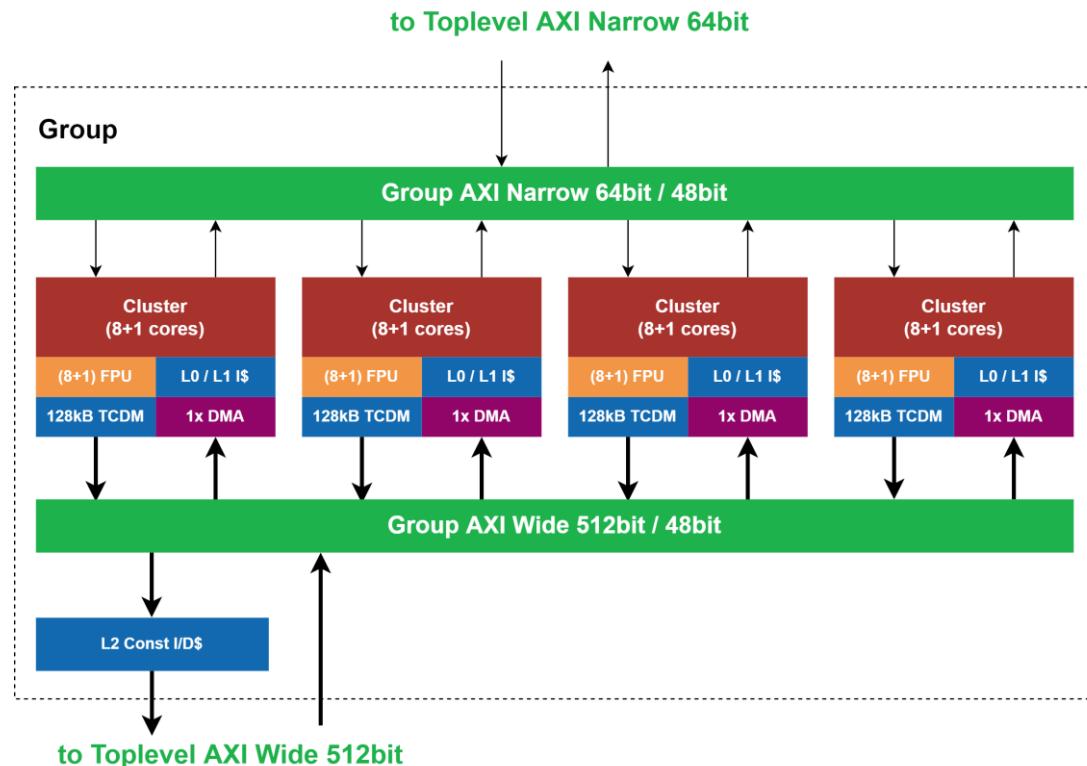
Constant Cache

- D/I-Cache hierarchy

From Snitch Group



To Multi-Group



Occamy – Chiplet Architecture

2 AXI Busses

- 64-bit narrow interface: config
- 512-bit wide interface: DMA

Peripherals

- Complex address space management

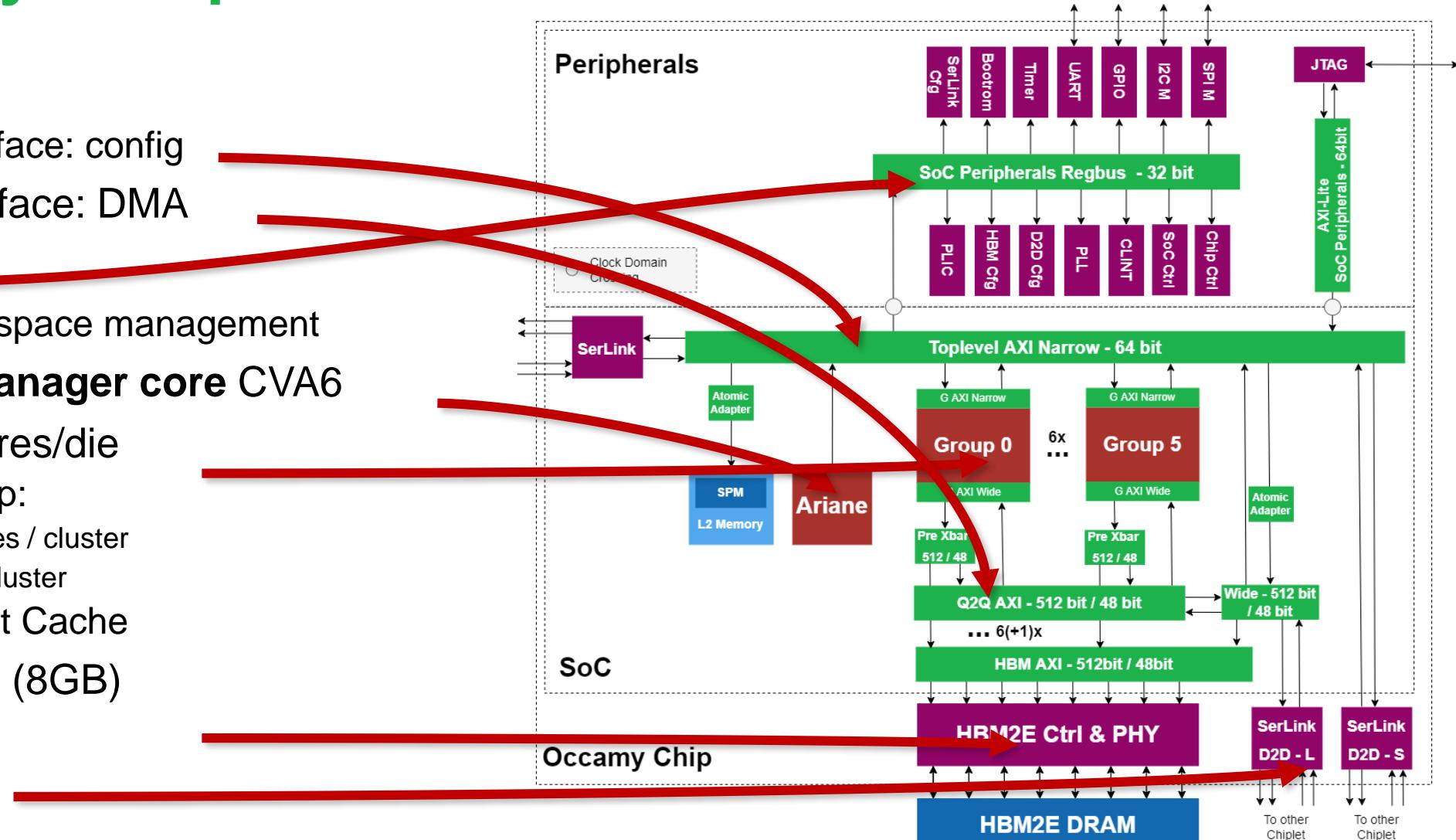
Linux-capable manager core CVA6

6 Groups: 216 cores/die

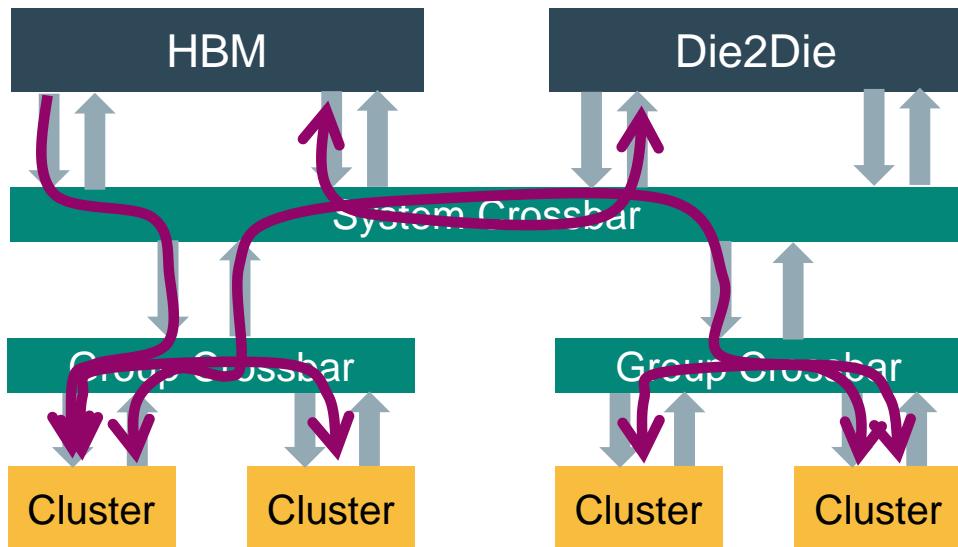
- 4 cluster / group:
 - 8 compute cores / cluster
 - 1 DMA core / cluster
- 512bit Constant Cache

8-channel HBM2e (8GB)

D2D serial link



Occamy NoC: Efficient and Flexible Data Movement



Problem: HBM Accesses are critical in terms of

- Access energy
- Congestion
- High latency

Instead reuse data on lower levels of the memory hierarchy

- Between **clusters**
- Across **groups**

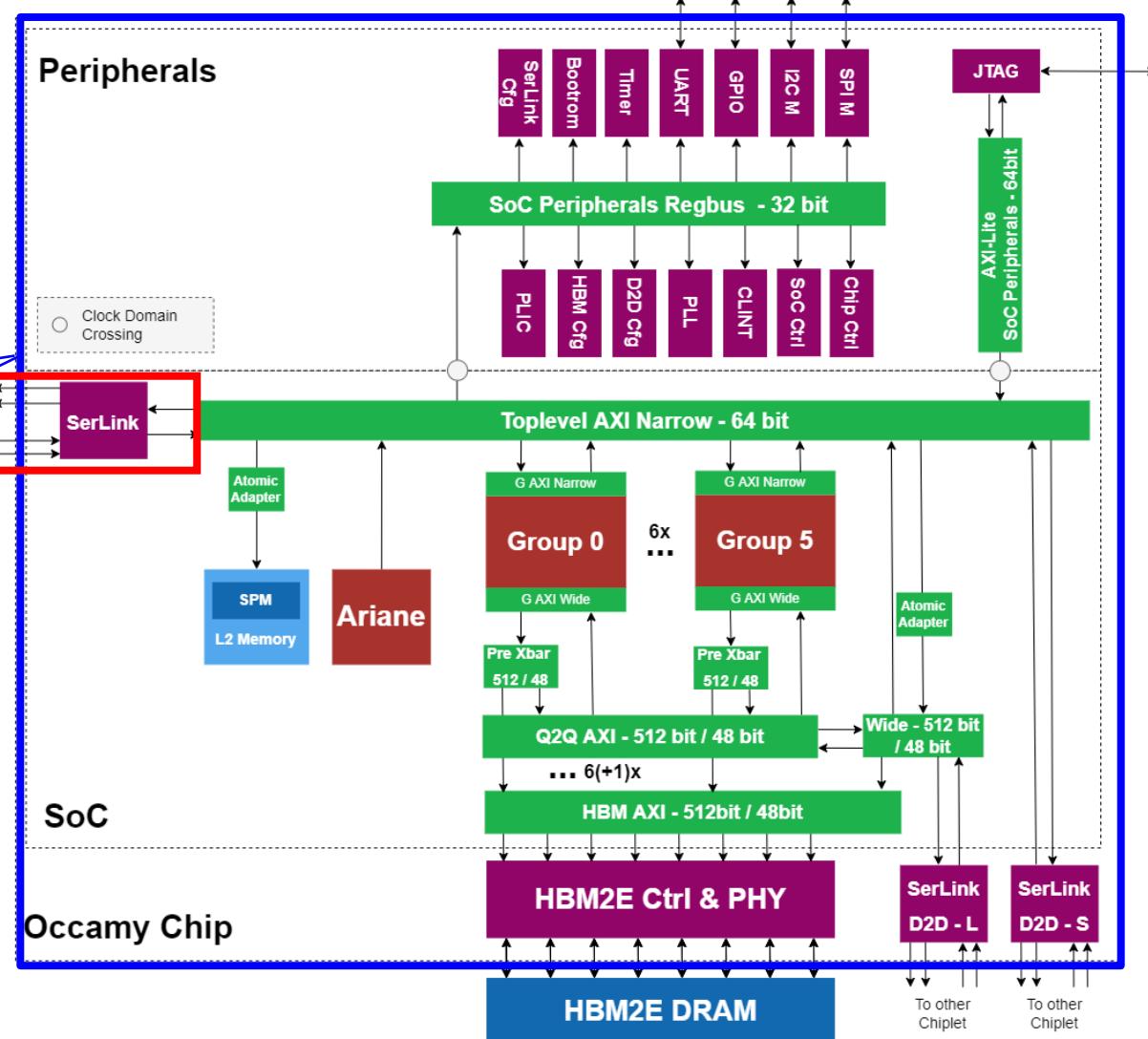
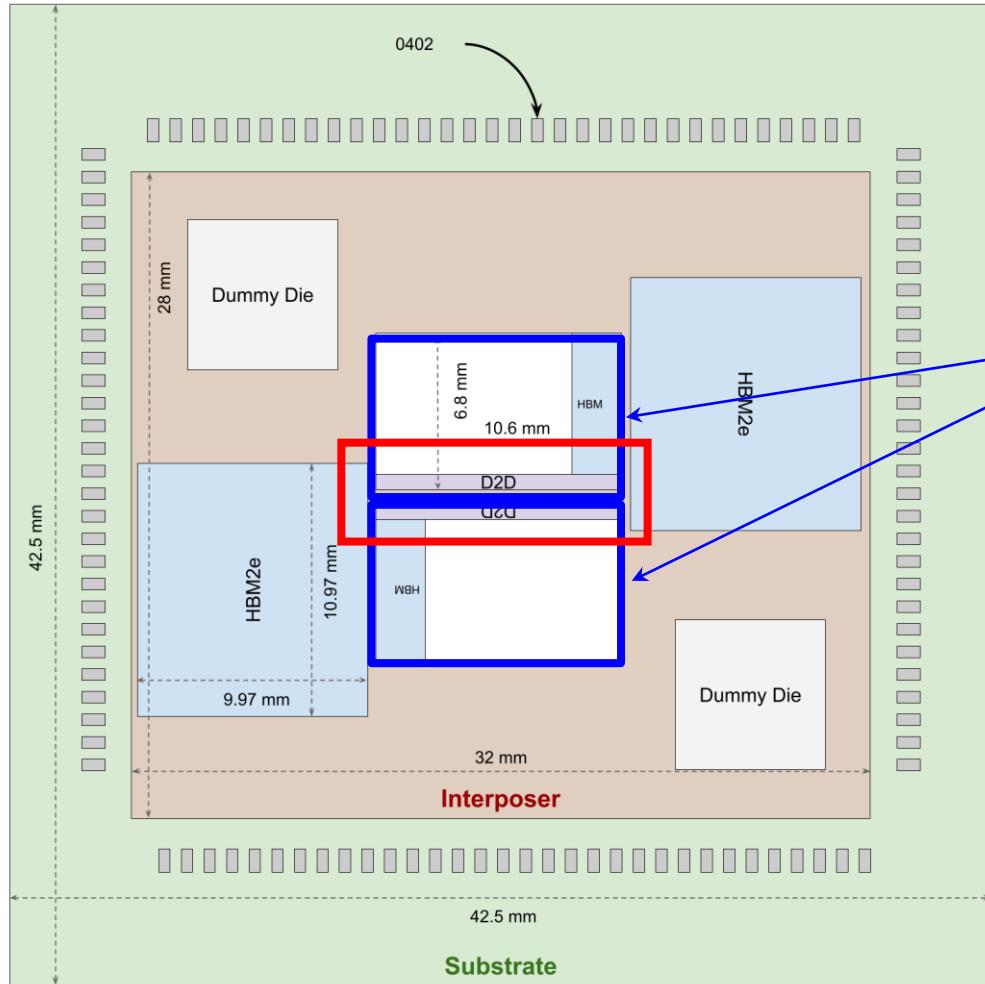
Smartly distribute workload

- **Clusters:** DORY framework for tiling strategy [1]
- **Chiplets:** E.g. Layer pipelining

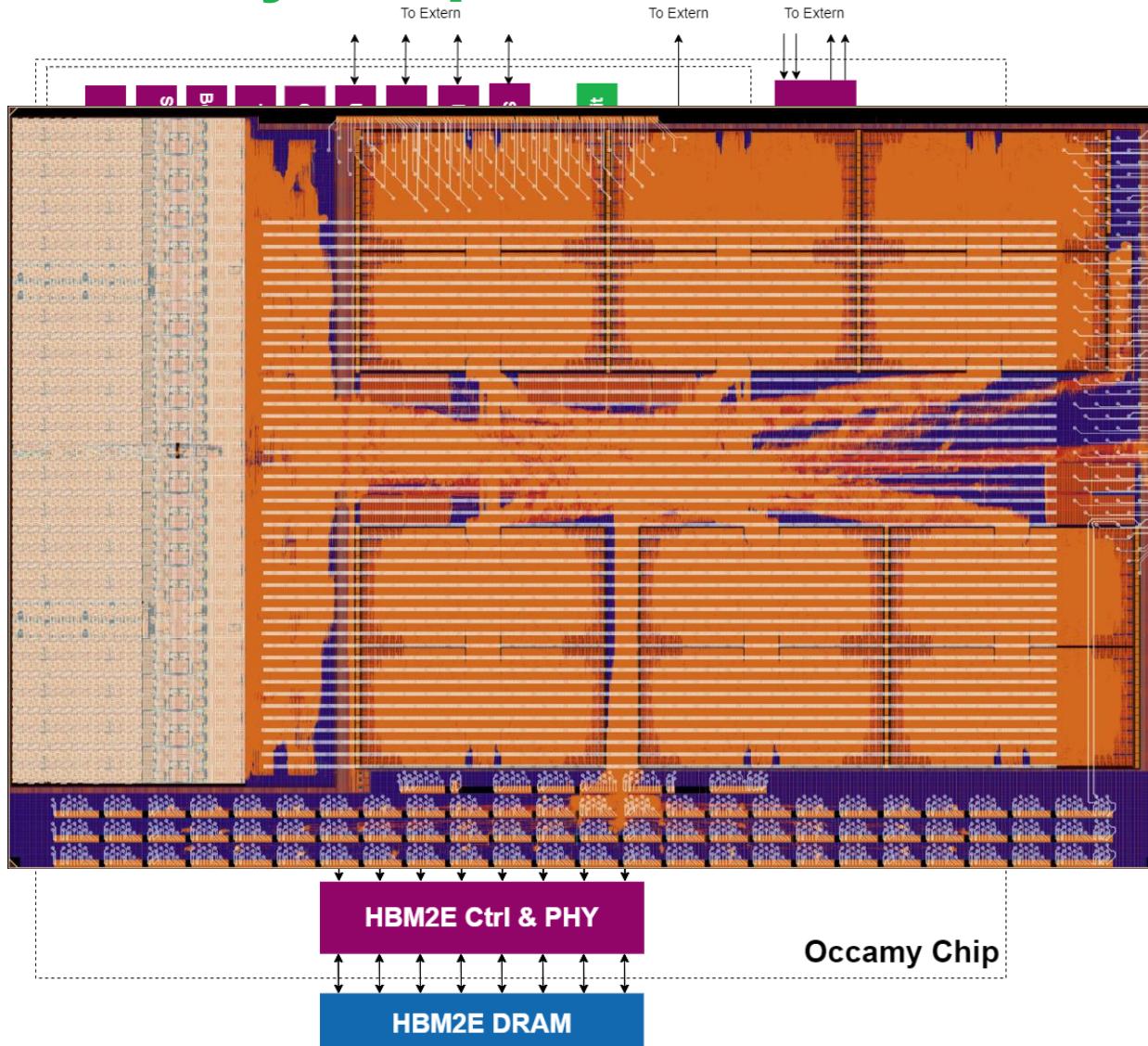
Big trend today!

[1] Burrello, Alessio, et al. "Dory: Automatic end-to-end deployment of real-world dnns on low-cost iot mcus." IEEE Transactions on Computers 70.8 (2021): 1253-1268.

Occamy's C2C Link



Occamy Chiplet



- GF12, target **1GHz** (typ)
- 2 AXI NoCs (multi-hierarchy)
 - 64-bit
 - 512-bit with “interleaved” mode
- Peripherals
- Linux-capable manager core CVA6
- 6 Quadrants: 216 cores/chiplet
 - 4 cluster / quadrant:
 - 8 compute +1 DMA core / cluster
 - 1 multi-format FPU / core
(FP64,x2 32, x4 16/alt, x8 8/alt)
- 8-channel HBM2e (8GB) **512GB/s**
- D2D link (Wide, Narrow) **70+2GB/s**
- System-level DMA
- SPM (2MB wide, 512KB narrow)

Peak 384 GDPflop/s per chiplet – Taped-out in July

Efficient SoC architecture in Perspective

1. Multi-cluster single-die scaling → strong latency tolerance, modularity
2. NoC for flexible Clus2Clus, Clus2Mem traffic → reduce pressure to Main memory
3. Full chiplet architecture: HBM2e, NoC-wrapped C2C, multi-chiplet ready

Back to the cluster... Can we make it Bigger?

- **Why?**

- Better global latency tolerance if $L1 > 2 * \text{Latency} * \text{Bandwidth}$
- Easier to program (data-parallel, functional pipeline...)
- Smaller data partitioning overhead

- **An efficient many-core cluster with low-latency shared L1**

- 256+ cores
- 1+ MiB of shared L1 data memory
- ≤ 5 cycles latency (without contention)

- **Physical-aware design**

- WC Frequency > 500 Mhz
- Targeting iso-frequency with small cluster



Hierarchical Physical Architecture

- **Tile**

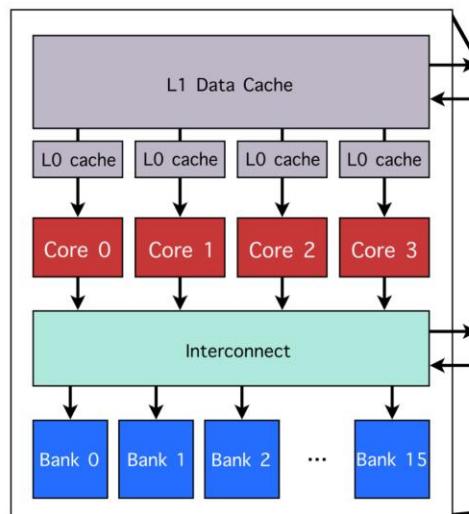
- 4 32-bit cores
- 16 banks
- **Single cycle memory access**

- **Group**

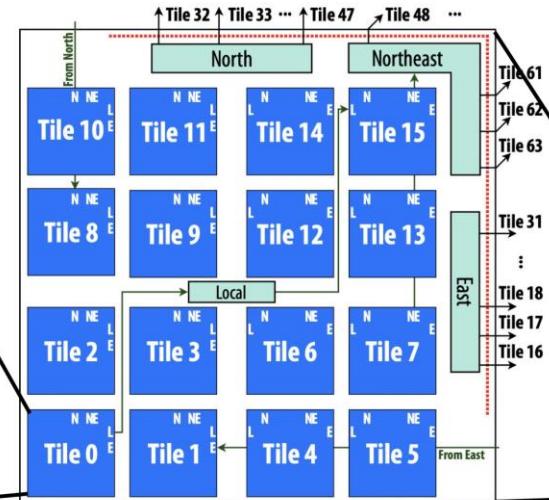
- 64 cores
- 256 banks
- **3-cycles latency**

- **Cluster**

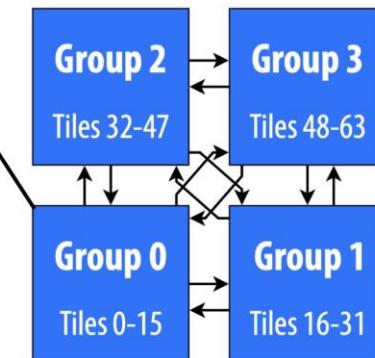
- 256 cores
- 1 MiB of memory (1024 banks)
- **5-cycles latency**



MemPool Tile



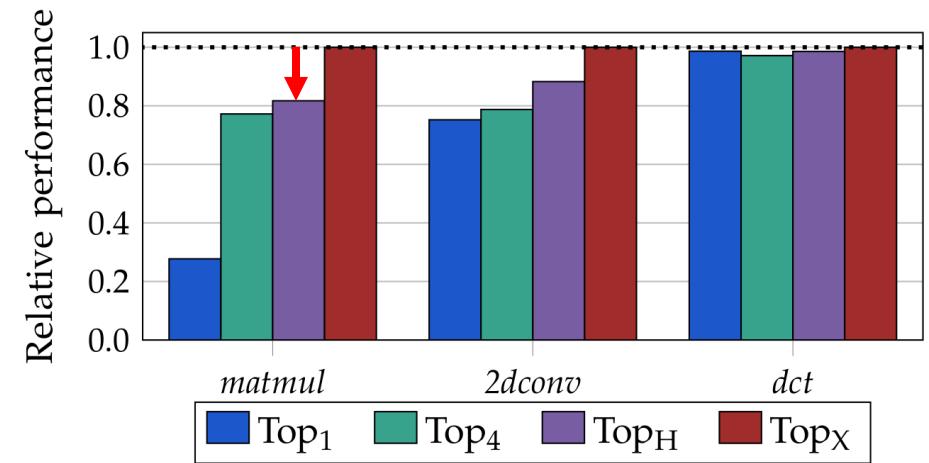
Group



Cluster

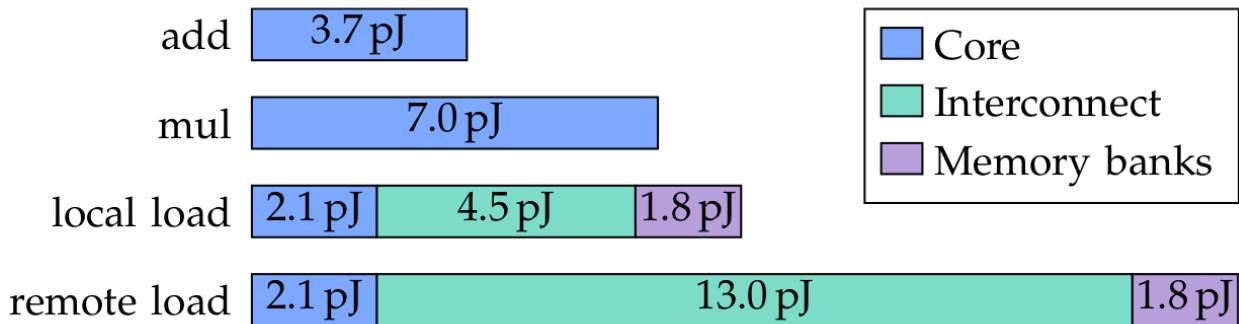
Benchmarks: can we compete with the impossible?

- Baseline: idealized Top_X
 - Fully connected logarithmic crossbar between 256 cores and 1024 banks
- Cycle-accurate RTL simulation
 - *Matmul*
 - Multiplication of two 64×64 matrices
 - *2dconv*
 - 2D Convolution with a 3×3 kernel
 - *dct*
 - 2D Discrete Cosine Transform on 8×8 blocks in local memory
- Top_H has a performance penalty of at most 20%, on all kernels

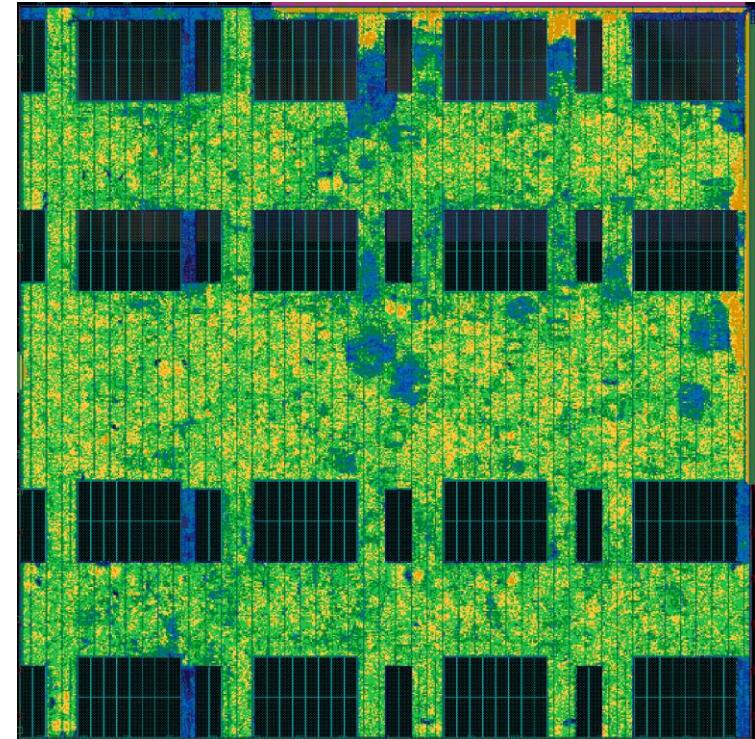


Energy Analysis (500 MHz, TT, 0.80 V, 25 °C)

- Breakdown of the energy consumption per instruction:

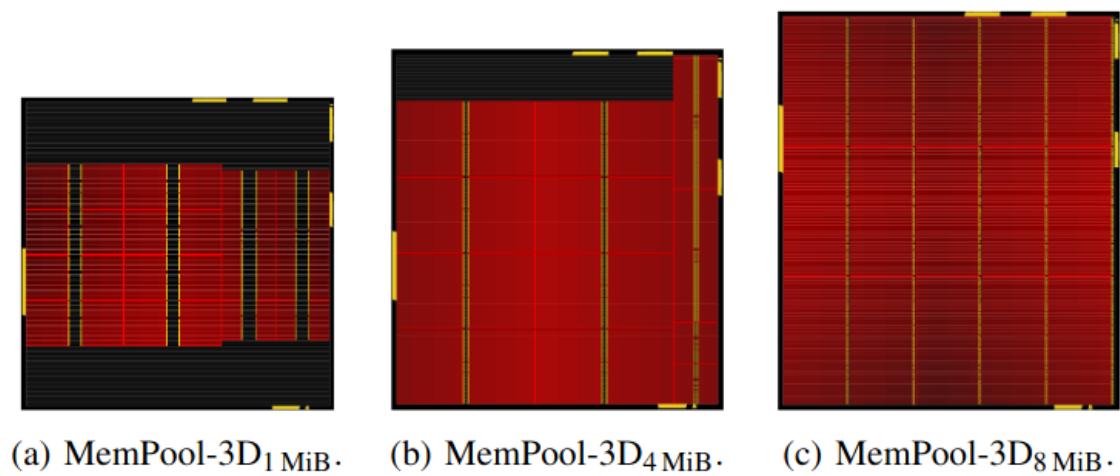
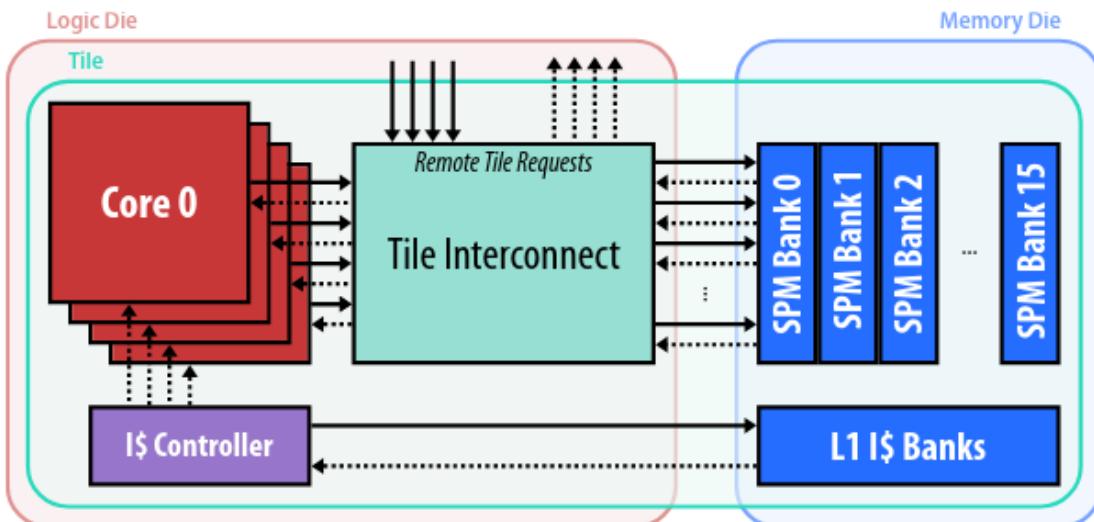


- Local loads consume about as much energy as a *mul*
 - About half of it, 4.5pJ, by the interconnect
- Remote loads: twice the energy of a local load
 - Despite crossing the whole cluster, twice!

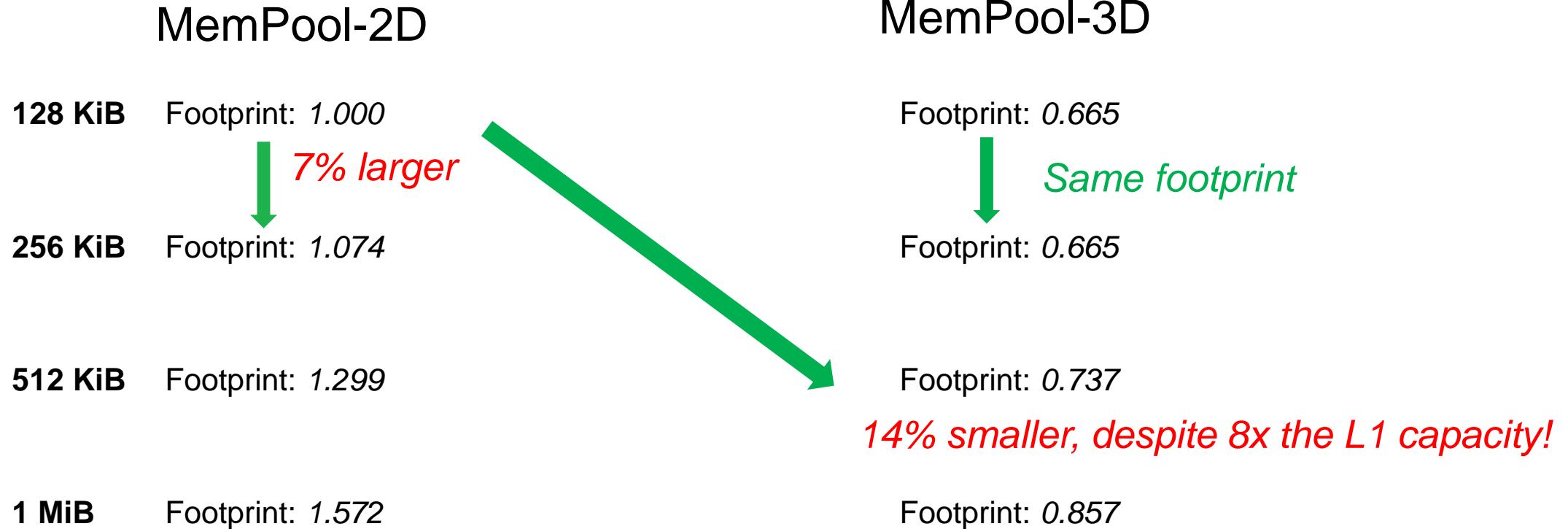


MemPool-3D

- Memory-on-Logic implementation of MemPool
- Implementation of MemPool with 1, 2, 4, 8 MiB of L1
- Leading to a higher utilization of the memory die



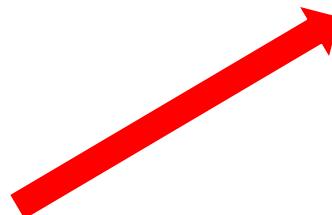
Groups: MemPool-2D vs. MemPool-3D



Similar trends for **Wire length** and **#Buffers**

Conclusion

- Energy efficiency quest: PE, Cluster, SoC



- Key ideas

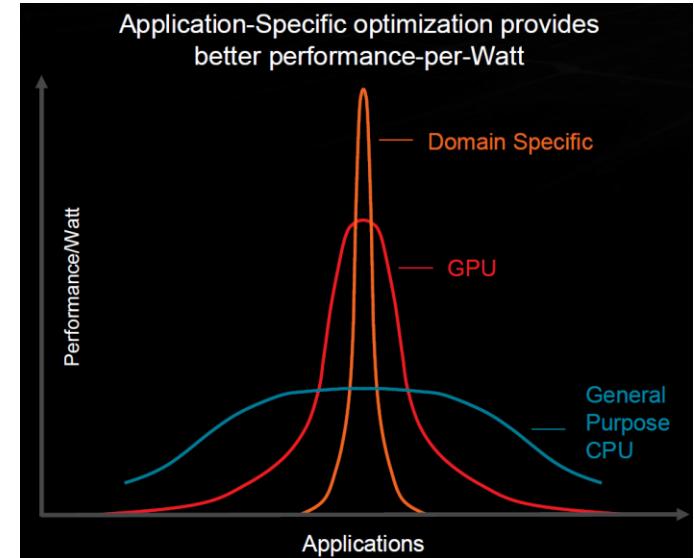
- Application specific PEs → extensible ISAs (RISC-V!)
- VNB removal + Latency hiding: large OOO processors not needed
- Low-overhead work distribution → large “mempool”
- Heterogeneous architecture → host+accelerator(s)

- Game-changing technologies

- Chiplets: 2.5D, 3D, (WFI?)
- Computing “at” memory,
- Coming (less than ten years): optical IO

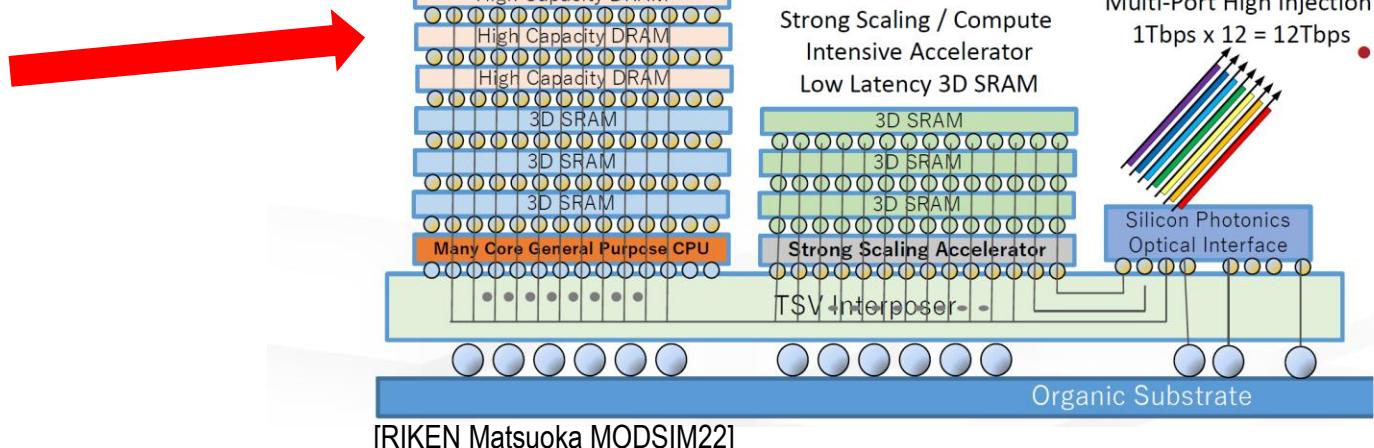
[AMD Naffziger ISCAS22]

Application-Specific optimization provides better performance-per-Watt



High Bandwidth / High Memory Capacity
General-Purpose Many-Core CPU

High Bandwidth SRAM + Large Capacity DRAM or NVM



[RIKEN Matsuoka MODSIM22]

Luca Benini, Alessandro Capotondi, Alessandro Ottaviano,
Alessandro Nadalini, 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, Davide Nadalini, Fabio Montagna, Florian Glaser,
Florian Zaruba, Francesco Conti, Frank K. Gürkaynak,
Georg Rutishauser, Germain Haugou, Gianna Paulin,
Gianmarco Ottavi, Giuseppe Tagliavini, Hanna Müller,
Lorenzo Lamberti, Luca Bertaccini, Luca Valente, Luca
Colagrande, Luka Macan, Manuel Eggimann, Manuele
Rusci, Marco Guermandi, Marcello Zangheri, Matheus
Cavalcante, Matteo Perotti, Matteo Spallanzani, Mattia
Sinigaglia, 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, Yvan Tortorella, all our past
collaborators **and many more that we forgot to mention**



PULP

Parallel Ultra Low Power



<http://pulp-platform.org>



@pulp_platform