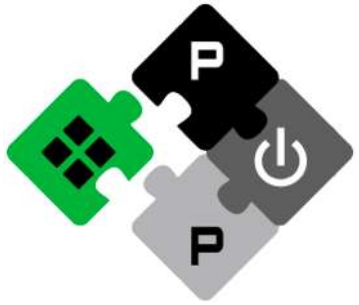


PULP PLATFORM

Open Source Hardware, the way it should be!

Open Platforms for Energy-Efficient Scalable Computing

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

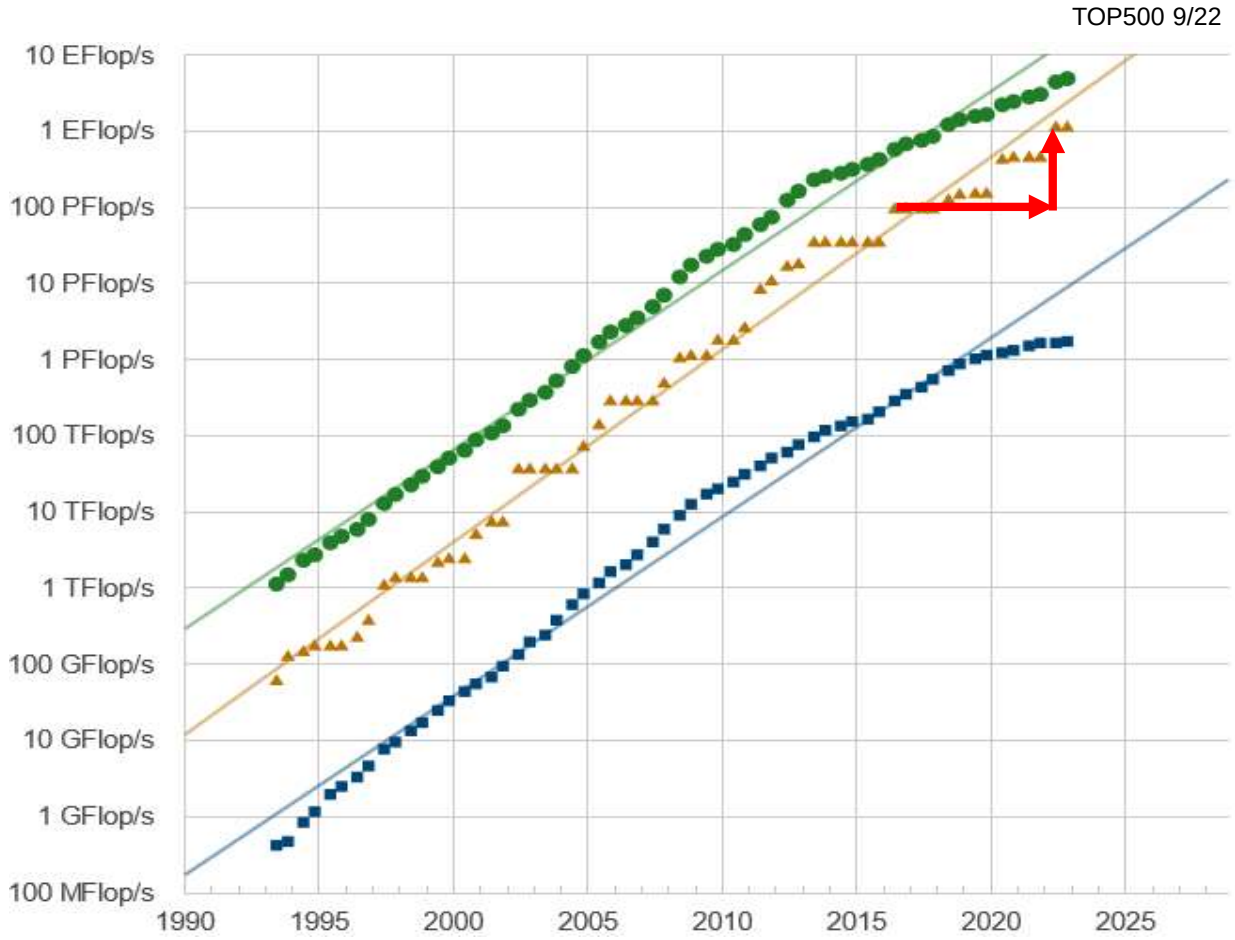
http://pulp-platform.org

@pulp_platform

https://www.youtube.com/pulp_platform

Computing is Power Bound: HPC

HPC Performance



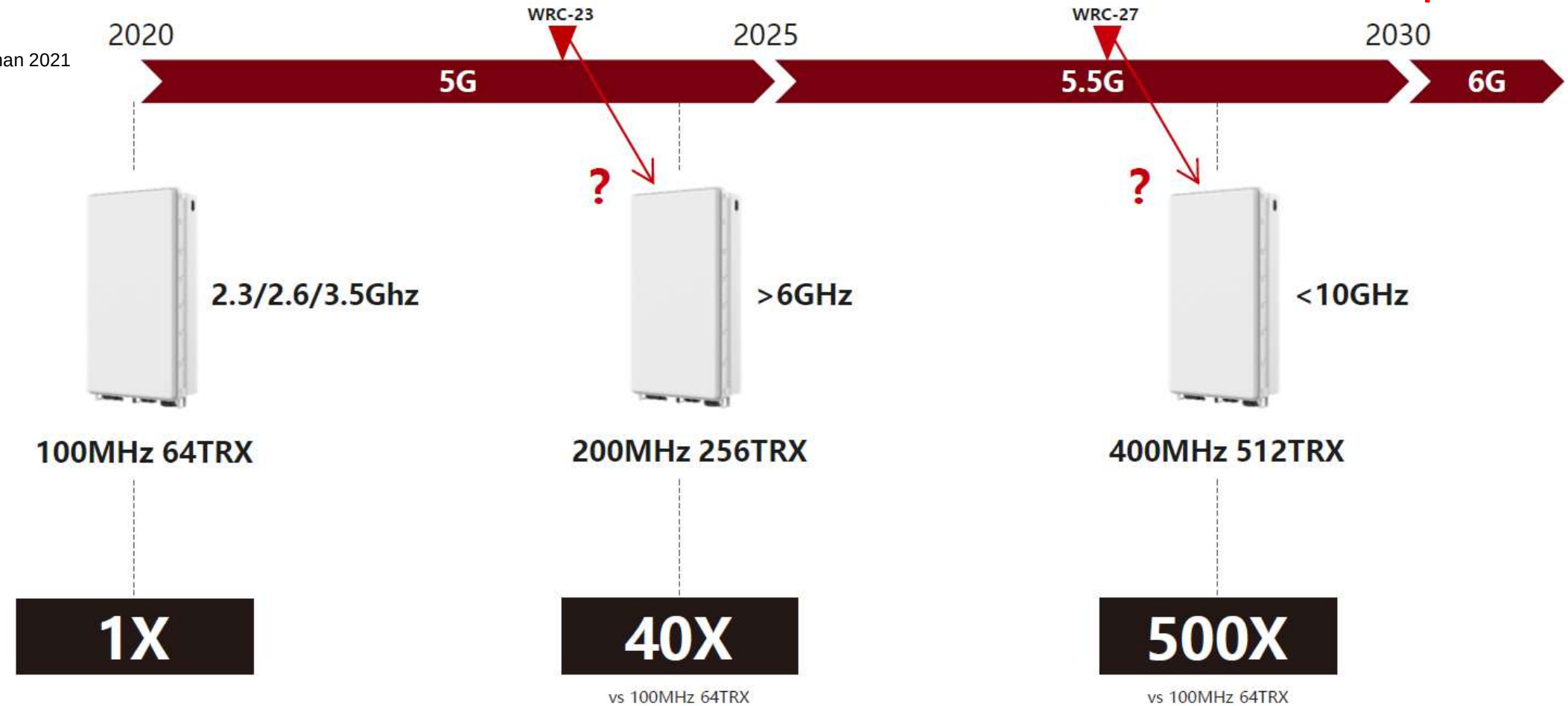
20MW budget (fixed)

HPC: 10x every ~5 years

Computing is Power Bound: 5G/6G

Base-station power ~ const

M. Khan 2021

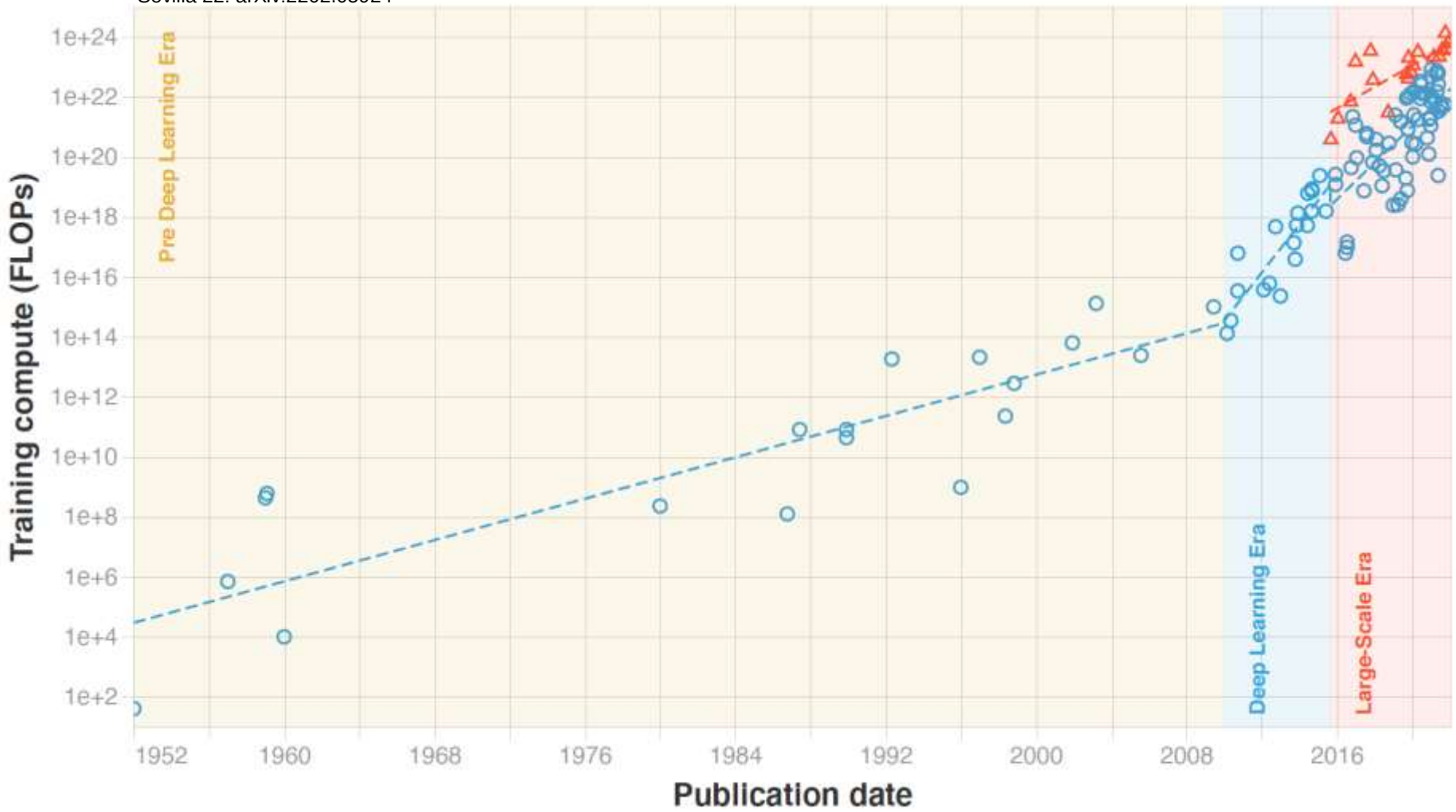


Baseband station processing: 10x every 3 years!

Computing is Power Bound: ML

Largest datacenter <150MW

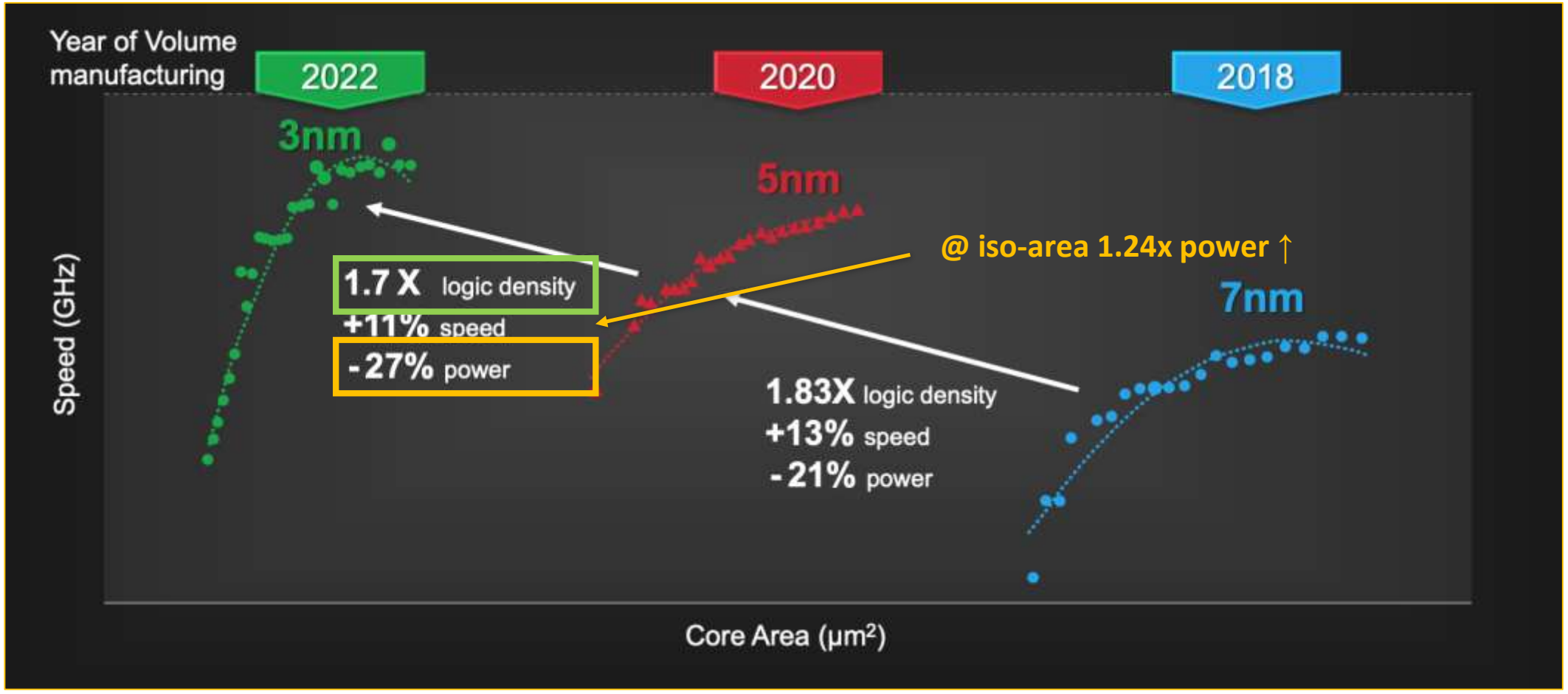
Sevilla 22: arXiv:2202.05924



Machine Learning (training): 10x every 2 years

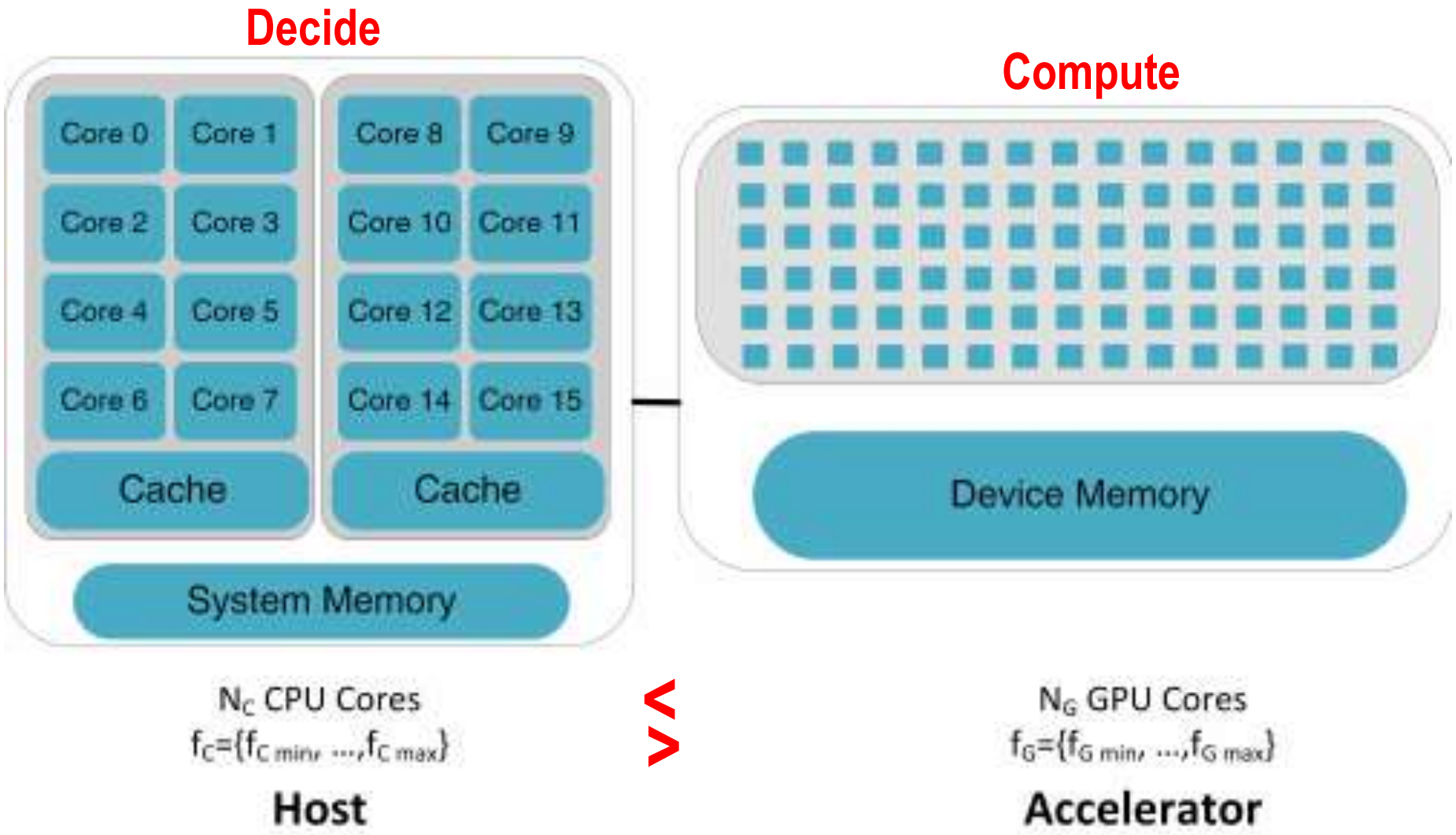
Technology Scaling?

TSMC, ISSCC21



Energy Efficiency $\left(\frac{1}{\text{Power}\cdot\text{Time}}\right)$ → **10x every 12 years…**

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

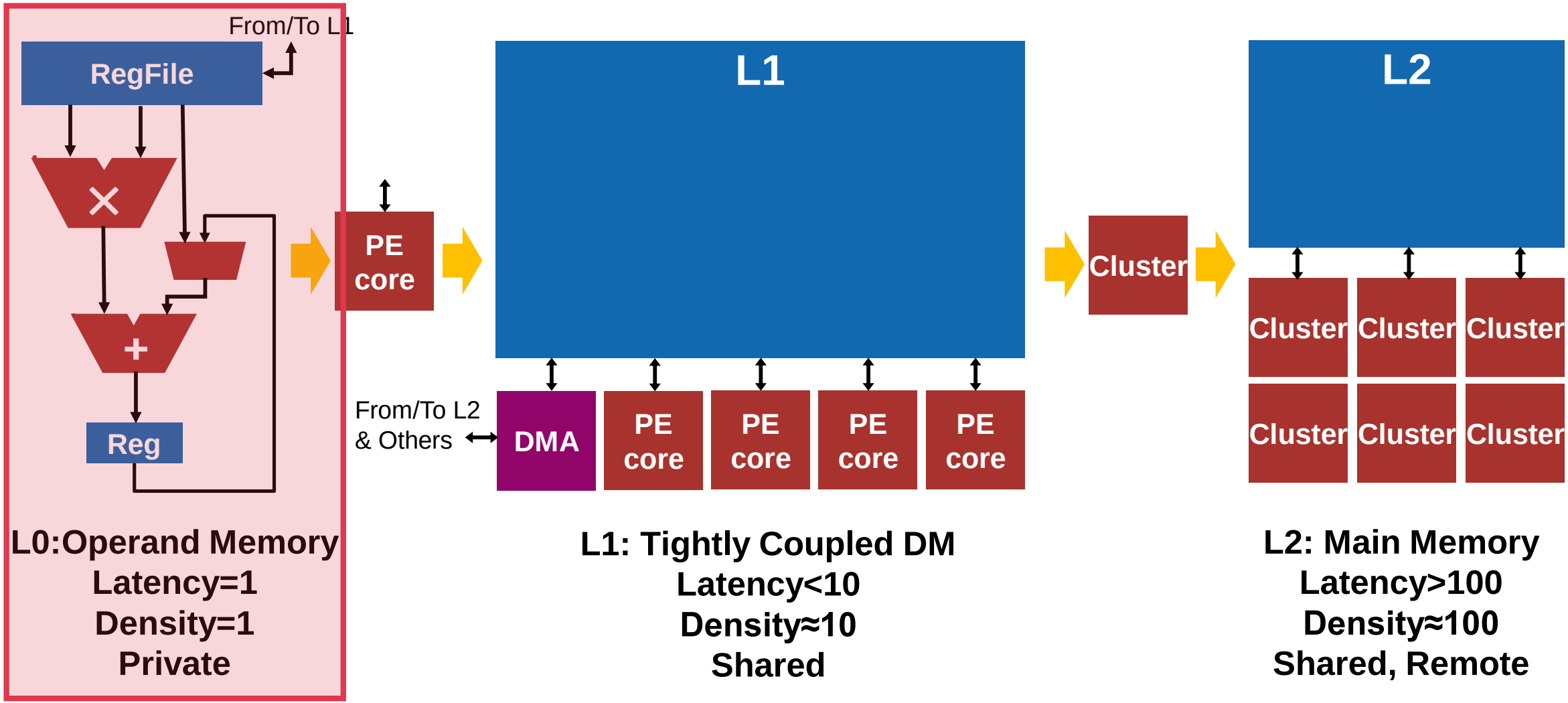
Compute (plough through numbers)

- Modulate flow of **data**
- Embarassing 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**

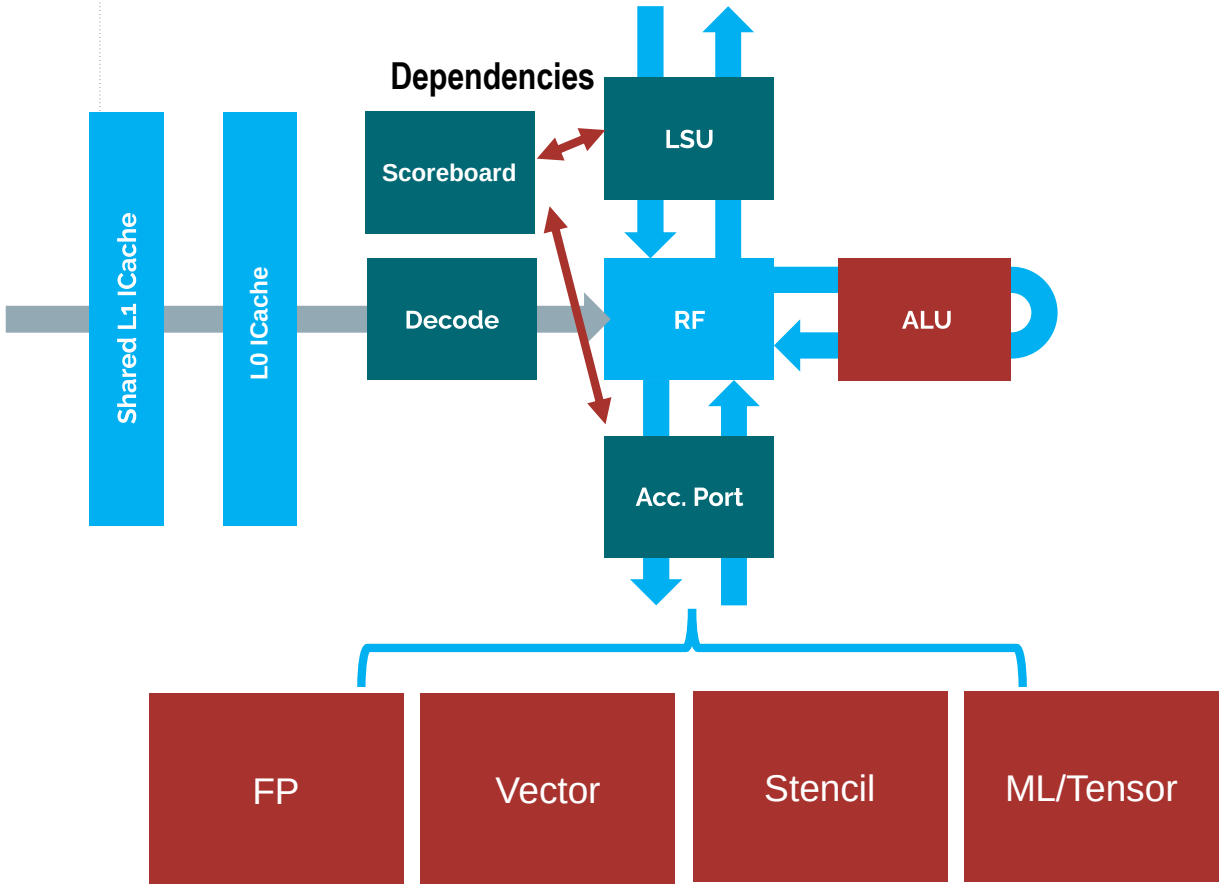
How to design an efficient "Compute" fabric?

Compute Efficiency: D (…and I) Movement is Key



PE: Snitch, a Tiny RISC-V 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!

F. Zaruba, F. Schuiki, T. Hoefler and L. Benini, "Snitch: A Tiny Pseudo Dual-Issue Processor for Area and Energy Efficient Execution of Floating-Point Intensive Workloads," in *IEEE Transactions on Computers*, vol. 70, no. 11, pp. 1845-1860, 1 Nov. 2021

Snitch PE: ISA Extension for efficient "Compute"

- **How can we remove the Von Neumann Bottleneck?**
- **Targeting "compute" code**

```
double sum = 0;
for (int i = 0; i < N; ++i) {
  sum += A[i] * B[i];
}
```

```
fld      ft0, 0(a1)          70 pJ
fld      ft1, 0(a2)          70 pJ
addi     a1, a1, 8           50 pJ
addi     a2, a2, 8           50 pJ
fmadd.d  fa0, ft0, ft1, fa0  80 pJ
bne      a1, a3, -5          50 pJ
```

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

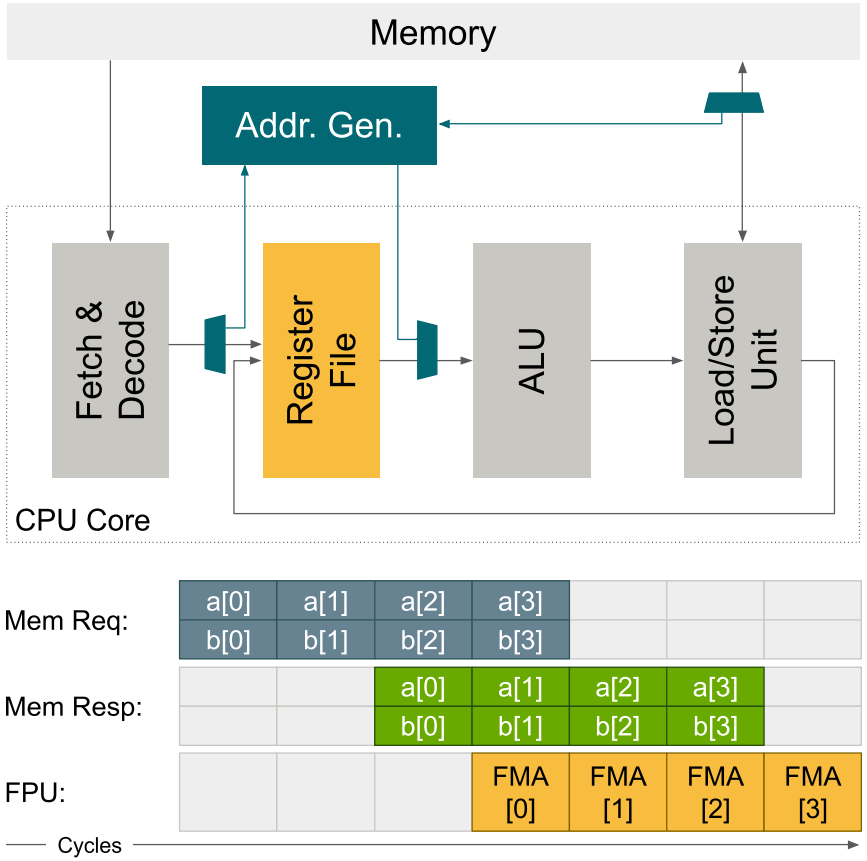
- Intuition: High FPU utilization ≈ high energy-efficiency
- Idea: Turn register read/writes into implicit memory loads/stores.
- Extension around the core's register file
- Address generation hardware

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

→

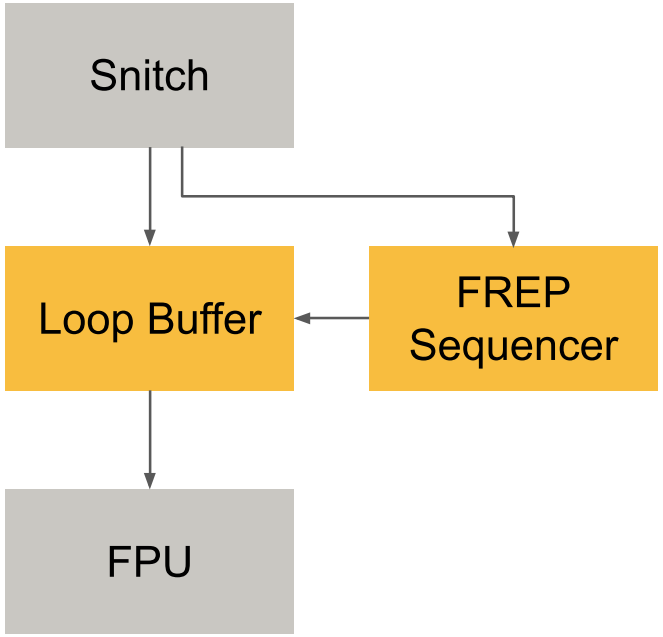
```
loop:
fld r0, %[a]
fld r1, %[b]
fmadd r2, r0, r1
```

- Increase FPU/ALU utilization by ~3x up to 100%
- SSRs ≠ memory operands
  - Perfect prefetching, latency-tolerant
  - 1-3 SSR (2-3KG/SSR)



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
```

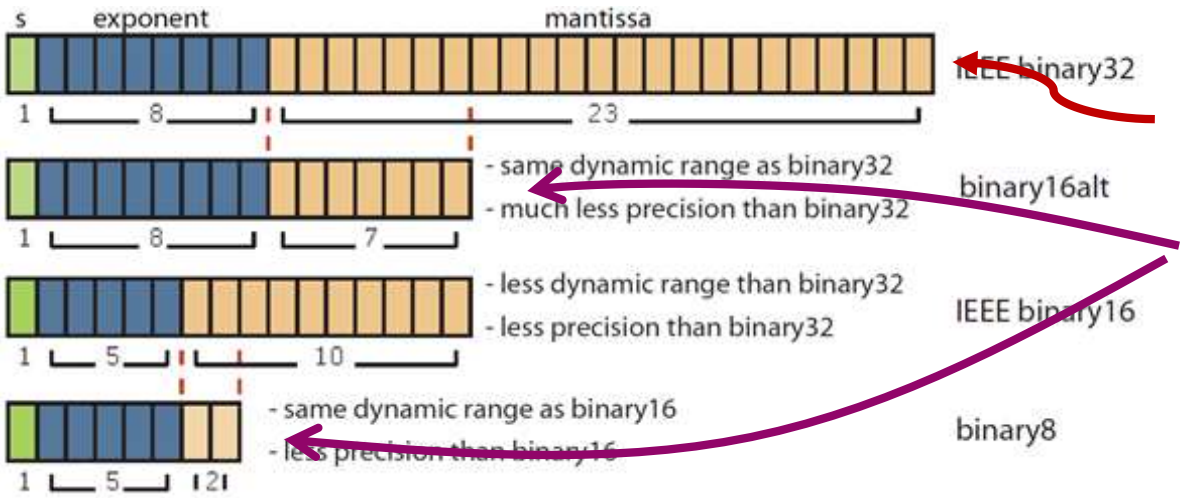
→

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

RISC-V ISA Extension for Target Workload

Mixed precision

Efficient DNN inference & training



Inference ≠ Training Quantization

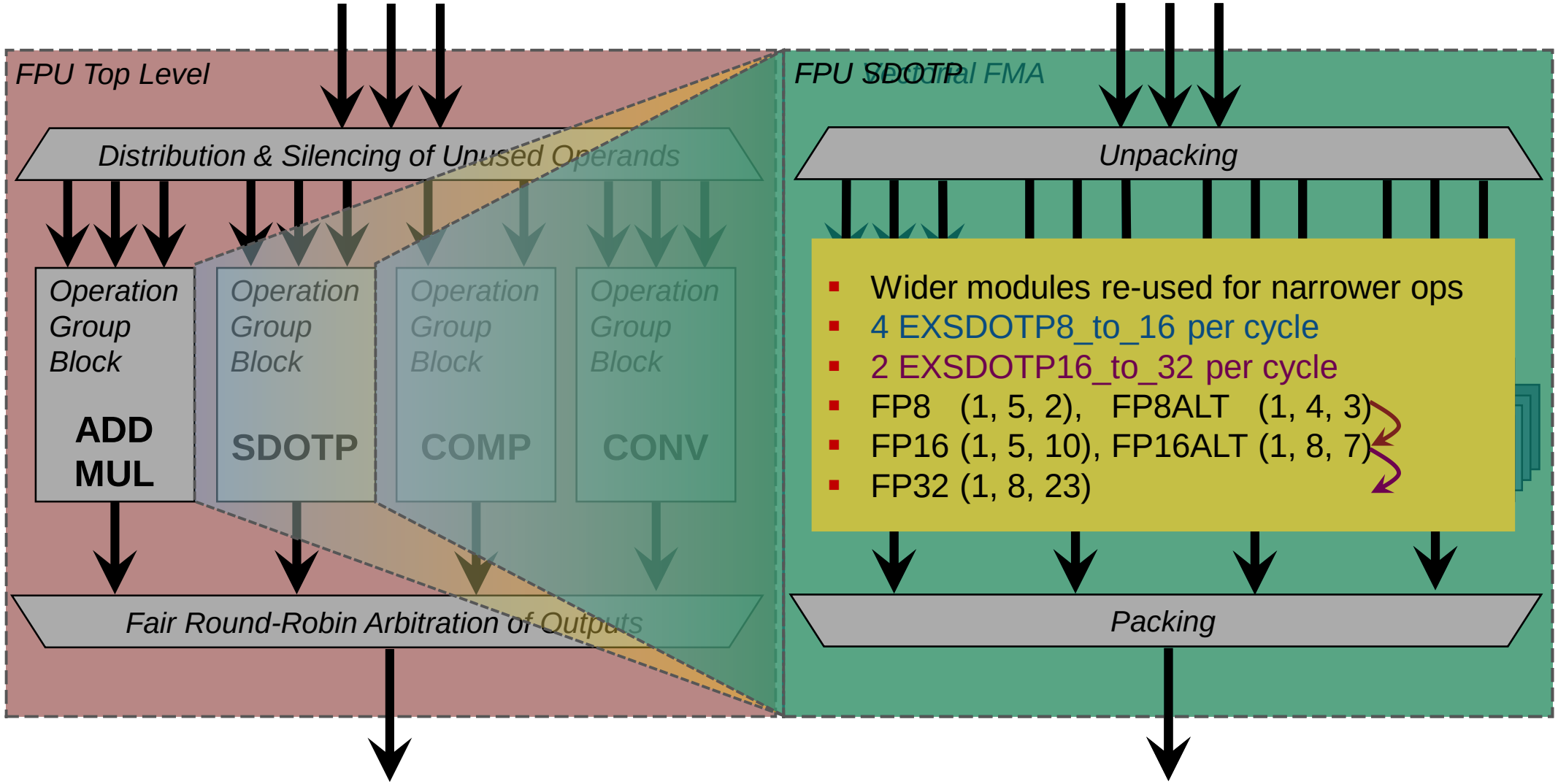
- Inference: INT8 quantization is SoA
- Training: High dynamic range needed for weights and weight updates

`fp32` is still standard for DNN training workloads. Low precision training with **`bf18`** and **`fp8`**

Support a wide variety of FP formats and instructions:

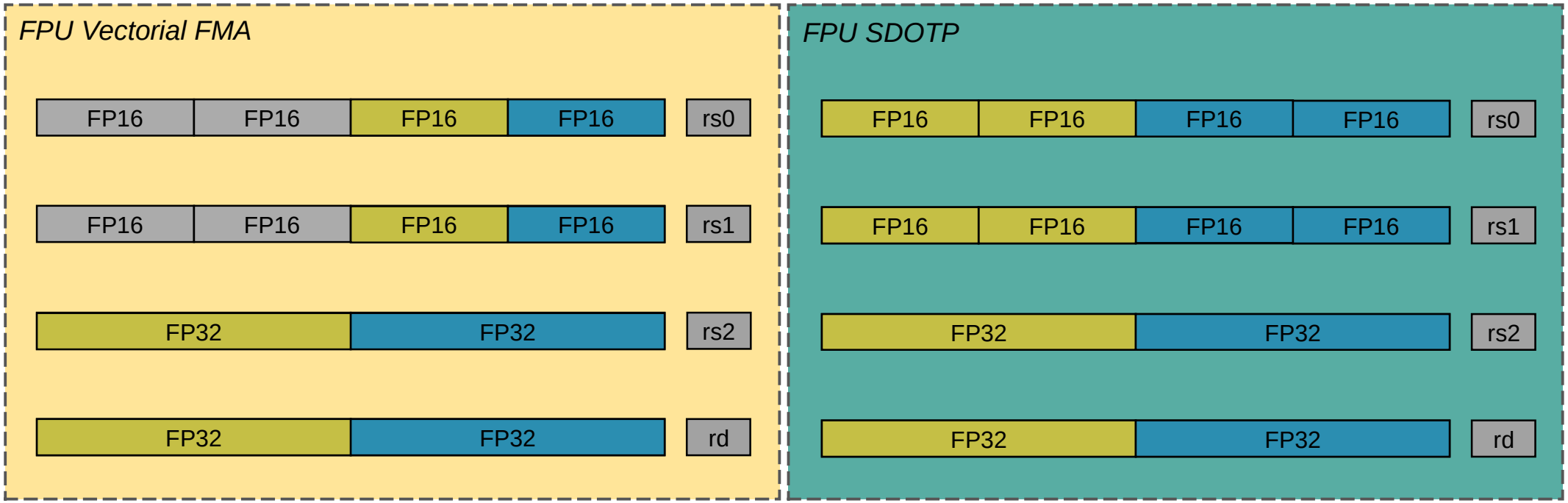
- Standard: **`fp64`**, **`fp32`**, **`fp16`**, **`bf16`**
- Low precision: **`fp8`**, ***`altfp8`***
  - `fp8` (1-4-3): forward prop.
  - *`altfp8`* (1-5-2): backward prop.
  - Exp. ops: accumulation

FPU and SDOTP unit



EXFMA vs EXSDOTP

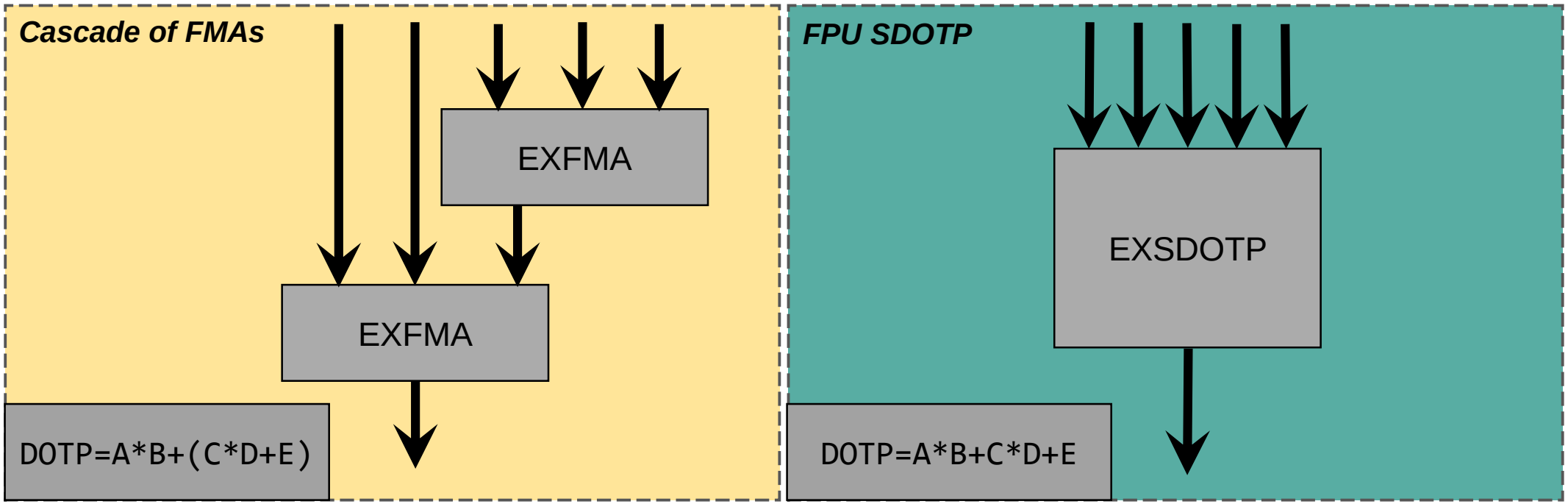
Expanding Dot Product w. accumulation



- $rd_{32} = rs0_{16} * rs1_{16} + rs2_{32}$
- Expanding FMA**: Unbalanced**
- Consumes only half of `rs0, rs1` and produces the whole `rd`

- $rd_{32} = rs0_{0,16} * rs1_{0,16} + rs0_{1,16} * rs1_{1,16} + rs2_{32}$
- Expanding SDOTP: **Balanced**
- The whole `rs0, rs1, rs2, rd` are used

Cascade of EXFMAs vs EXSDOTP

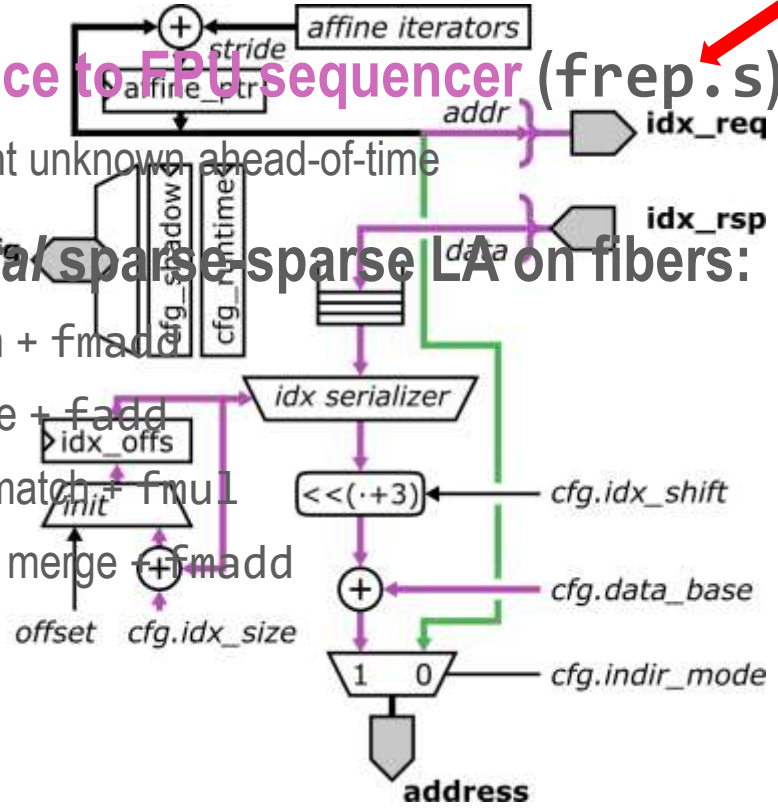


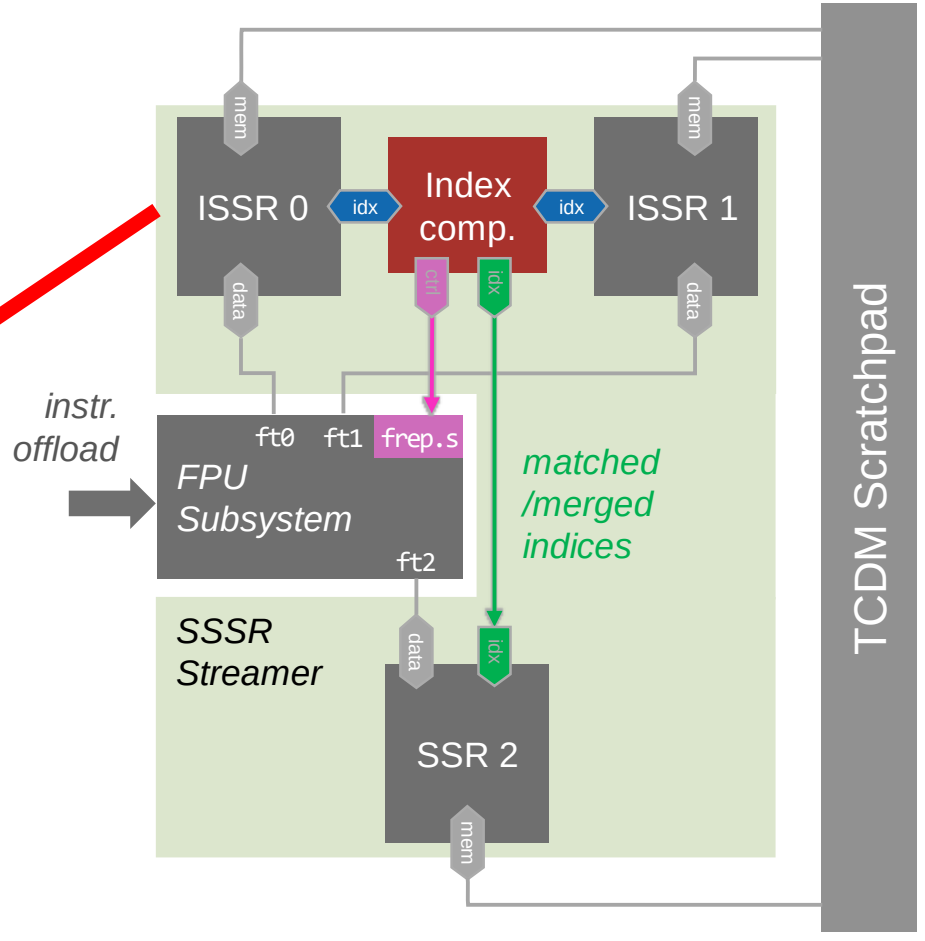
Non-distributive FP addition → **Precision Loss**

- **Fused** EXSDOTP (i.e. lossless)
- Single normalization and rounding step
- **Smaller** area and **shorter critical path**
- Product by-pass to compute **fused three-term addition** (vector inner sum)
- **Stochastic rounding** supported (+3% area)

What About Sparsity? Sparse SSR Streamer

- **Based on existing 3-SSR streamer**
  1. Extend 2 SSRs to ISSRs
  2. Add *index comparison unit* between ISSRs
  3. Forward result indices to 3rd SSR
- **Control interface to FPU sequencer (`frep.s`)**
  - Result index count unknown ahead-of-time
- **Enables *general* sparse-sparse LA on fibers:**
  - *dotp*: index match + `fmadd`
  - *vadd*: index merge + `fadd`
  - *elem-mul*: index match + `fmul`
  - *vec-mac*: index merge + `fmadd`



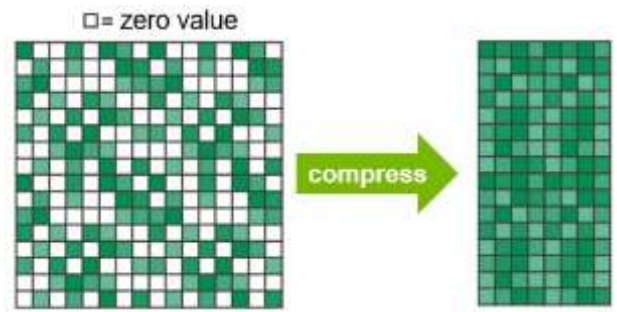


Sparse SSR Functional Benefits

- **Enable *continuous useful issues* in sparse LA**
- **Accelerate uses beyond sparse LA**
  - Dense workloads: dense LA, CONV, FFT, …
  - ISSRs as *dynamic pattern streamers*: stencil codes, codebook decoding, …
  - *Index matching* in graph operations
- **Provide SoA capabilities *without* common compromises**
  - Indirection, intersection, *and* union
  - Efficient inflight *or* AoT data transform
  - No sparsity or structural requirements (e.g. A100[1])
  - No imposed dataflow (e.g. OuterSpace[2])

```
dotp:call    conf_ft0_mst_match
    call     conf_ft1_mst_match
    frep.s                          }
    fmadd.d fa1, ft0, ft1, fa1      }

vadd:call    conf_ft0_mst_merge     } compute only
    call     conf_ft1_mst_merge     }
    call     conf_ft2_slv_wb        }
    frep.s                          }
    fadd.d   ft2, ft0, ft1          }
```



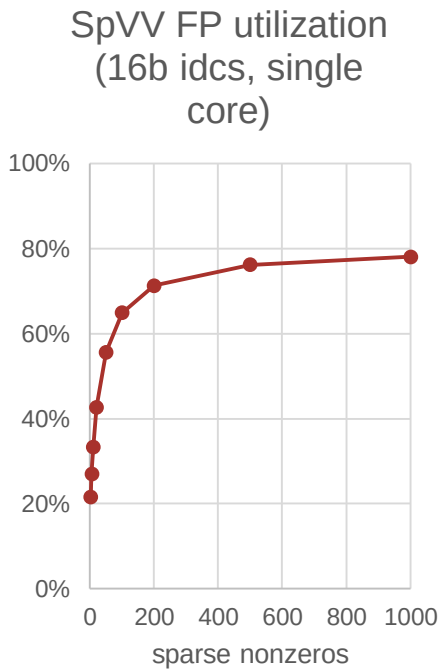
[1]

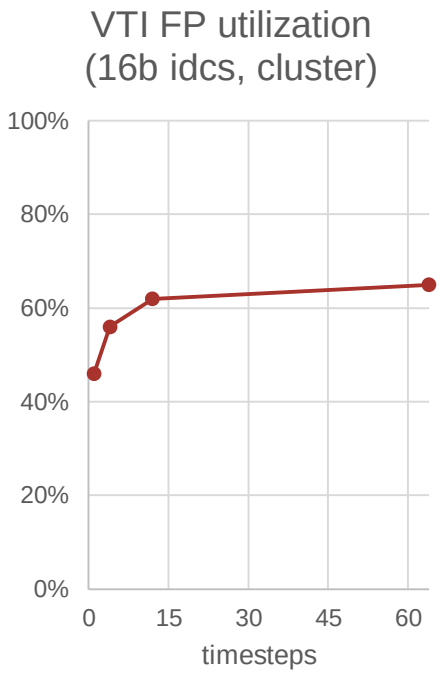
"2:4 structured pattern"

[1] Nvidia Corporation, "NVIDIA A100 Tensor Core GPU Architecture".
[2] S. Pal et al., "OuterSPACE: An Outer Product Based Sparse Matrix Multiplication Accelerator," in HPCA 2018, pp. 724-736.

Sparse SSR Performance Benefits

- **Notable single-core speedups over RV baseline**
  - *CsrMV*: up to **7.0×** faster, **79%** FP util.
  - *SpV+SpV*: up to **9.8×** faster / higher FP util.
  - *SpV·SpV*: up to **7.7×** faster / higher FP util.
  - *VTI (3D stencil code):* up to **2.9×** faster, **78%** FP util.
- **Significant benefits in multicore cluster:**
  - *CsrMV :* up to **5.0×** faster, **2.9x** less energy
  - *CsrMSpV :* up to **5.8×** faster, **3.0x** less energy
  - *VTI:* up to **2.7×** faster
- **Notably higher peak FP utilizations than SoA CPUs (69×), GPUs (2.8×) on CsrMV**

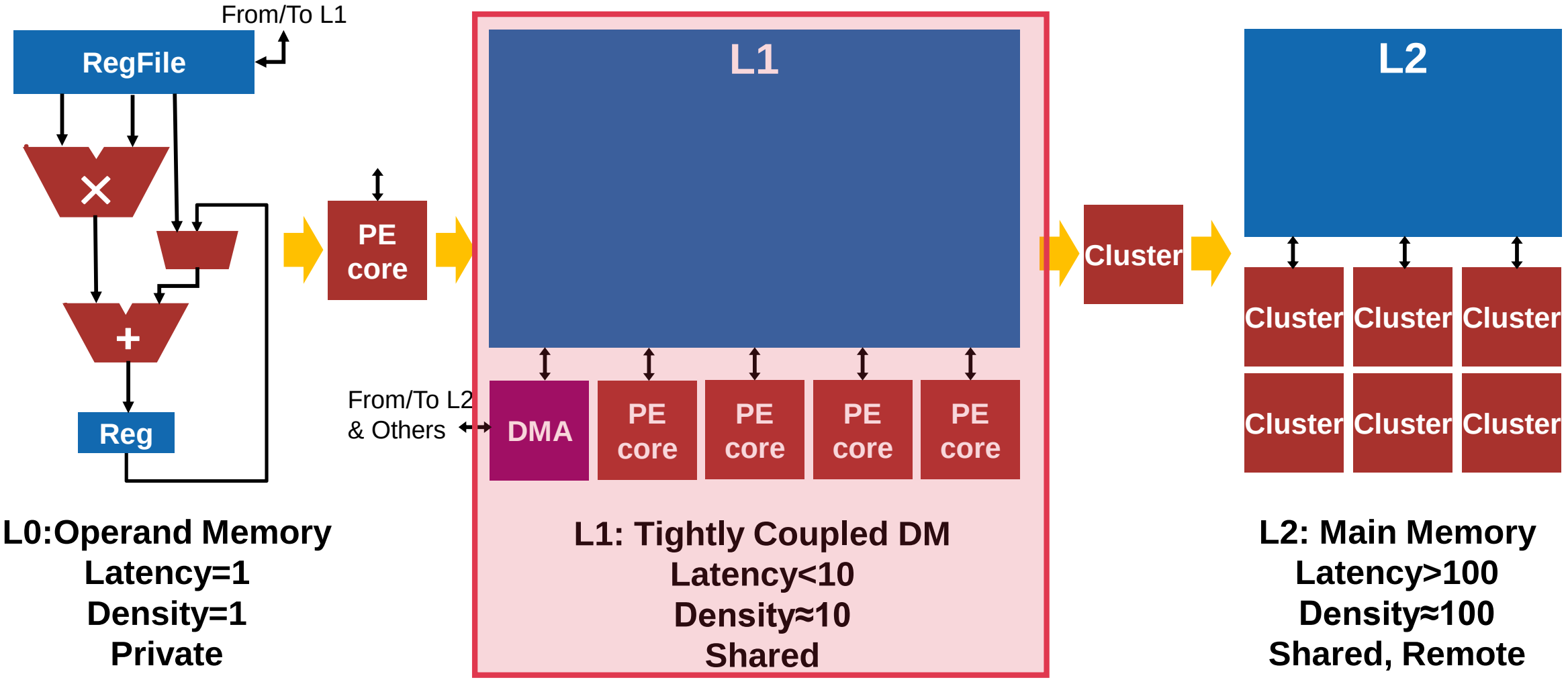




Efficient PE (snitch) architecture in perspective

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

Compute Efficiency: the Cluster (PEs + On-chip TCDM)



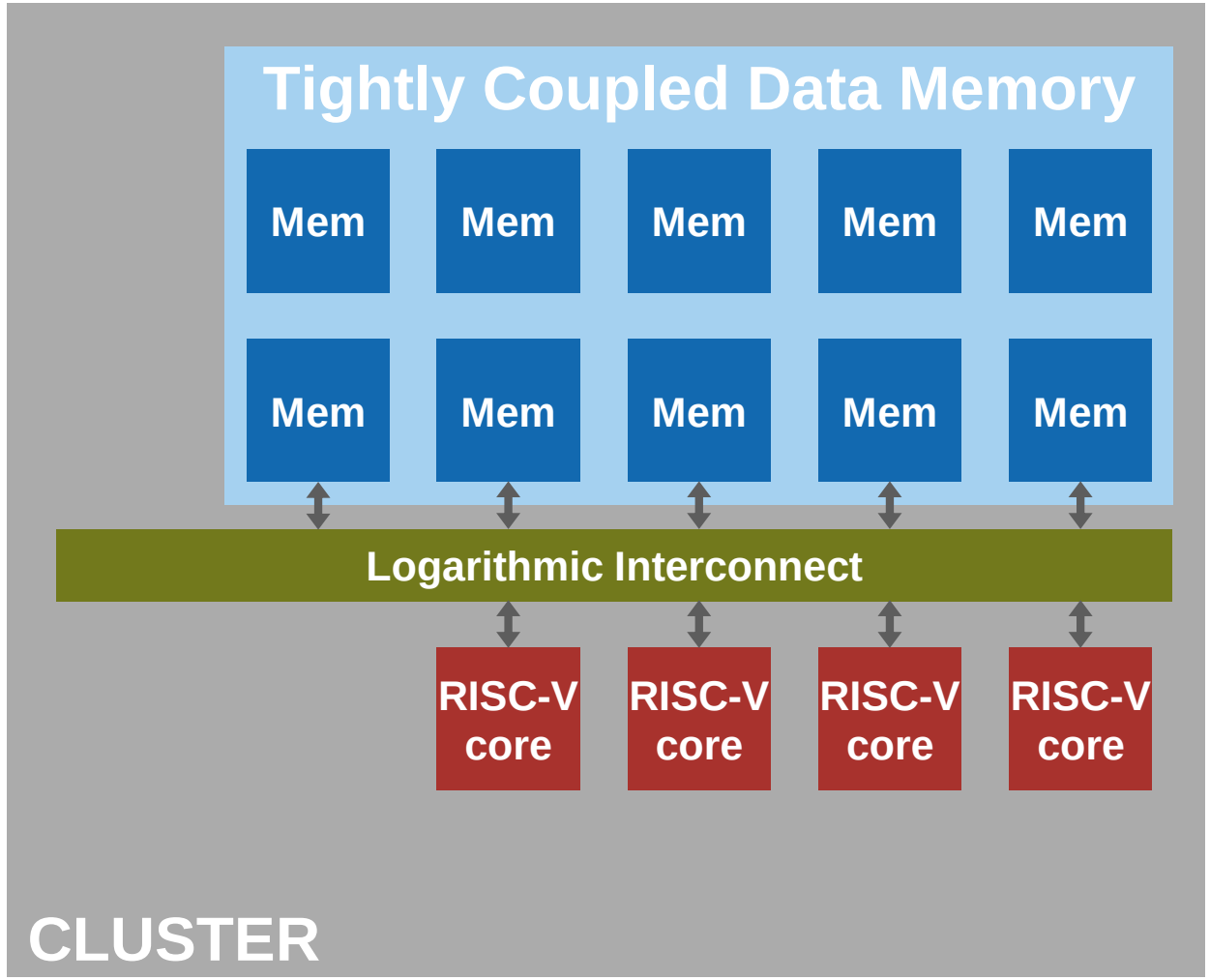
L0:Operand Memory
Latency=1
Density=1
Private

L1: Tightly Coupled DM
Latency<10
Density≈10
Shared

L2: Main Memory
Latency>100
Density≈100
Shared, Remote

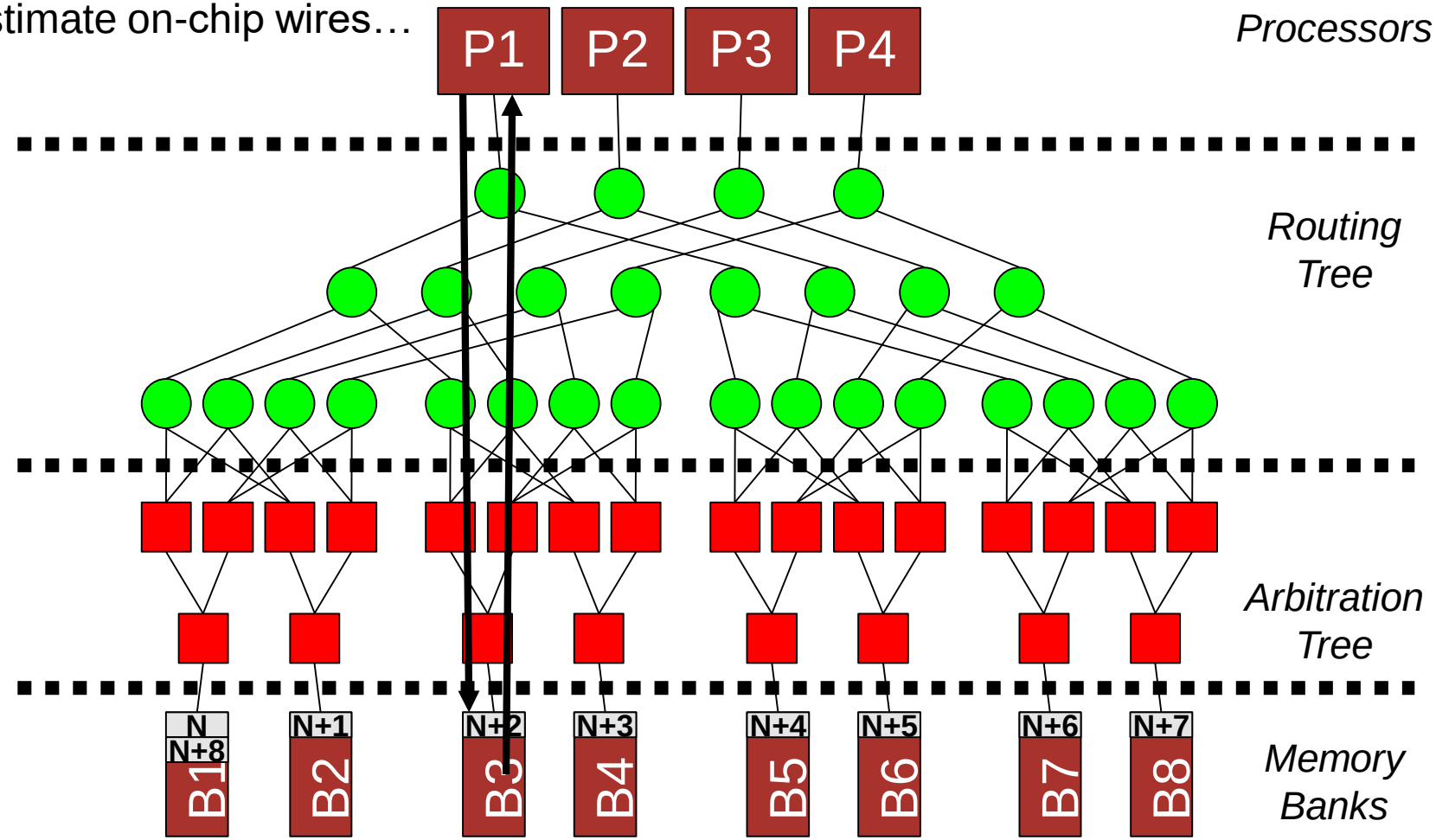
The Cluster: Design Challenges

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



High speed logarithmic interconnect

Do not underestimate on-chip wires…



World-level bank interleaving «emulates» multiported mem

@1GHz, 8-16 PEs, Latency: 2 cycles + stalls for banking conflicts

A. Rahimi, I. Loi, M. R. Kakoee and L. Benini, "A fully-synthesizable single-cycle interconnection network for Shared-L1 processor clusters," *2011 Design, Automation & Test in Europe*, 2011, pp. 1-6.

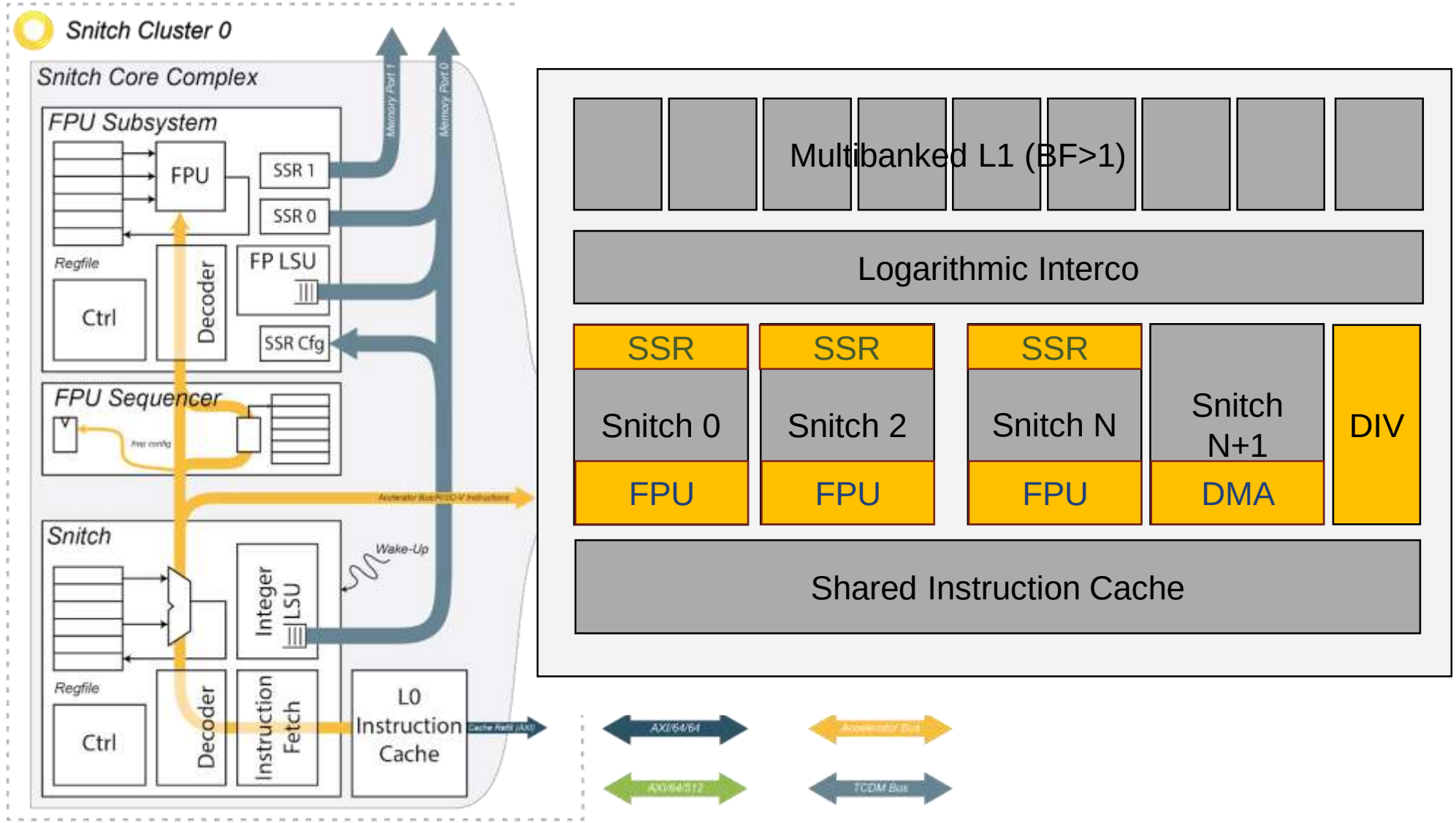
Efficient *Explicit* Global Data Mover

hide L2main 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 2-4-dim shapes
- Higher-dimensionality handled by SW
- Intrinsics/library for easy programming
- Domain-specific autotilers

```
// 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?

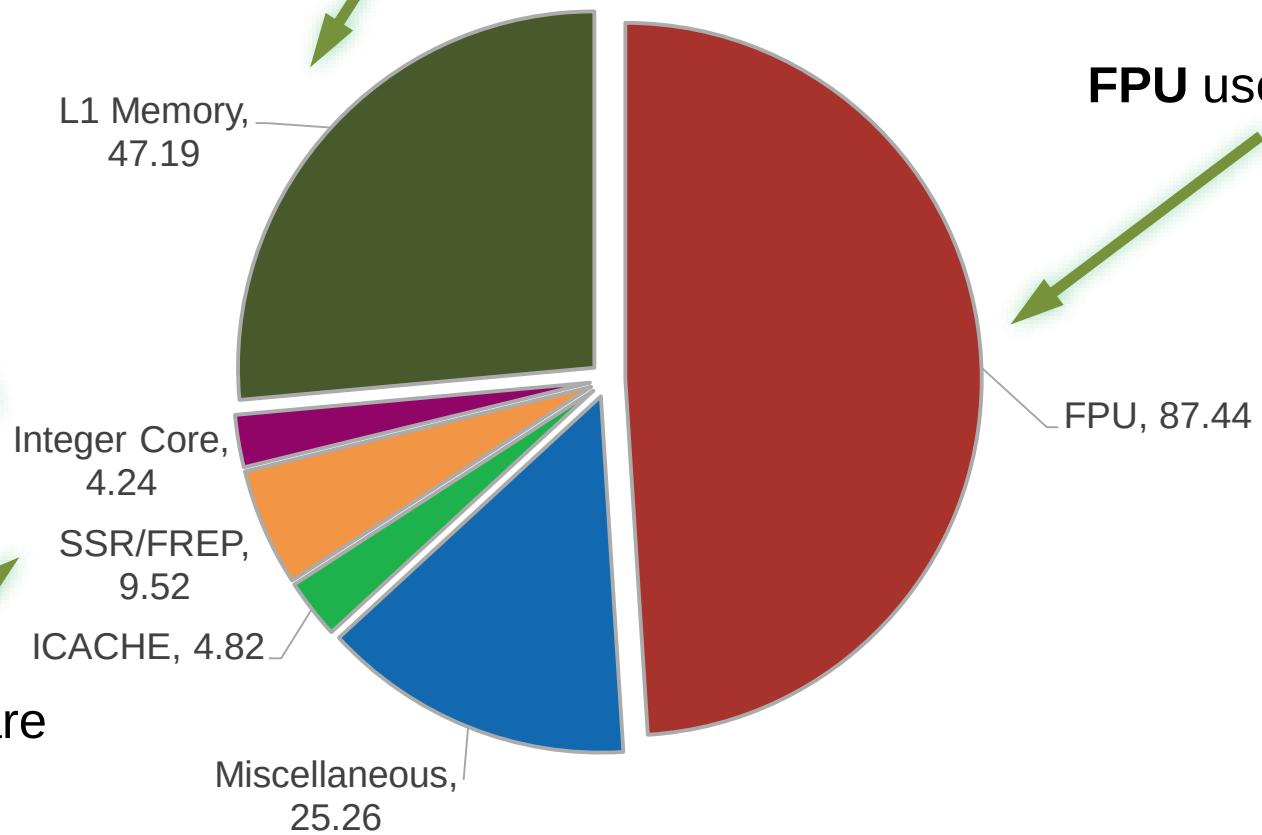
In an 8-core cluster

Inevitable to have local memory
(e.g., GPU/GPU L1 cache, vector register file)

FPU uses **50%** of power

Integer core uses
2% of power

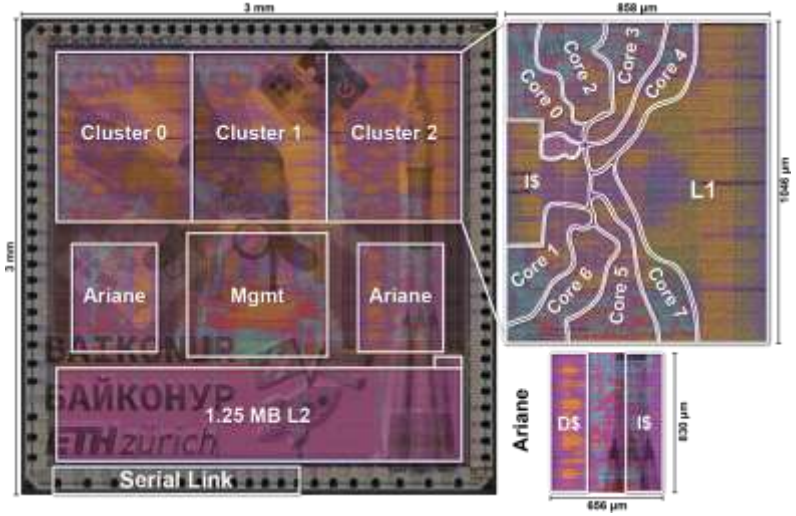
SSR/FREP hardware
uses **5%** of power

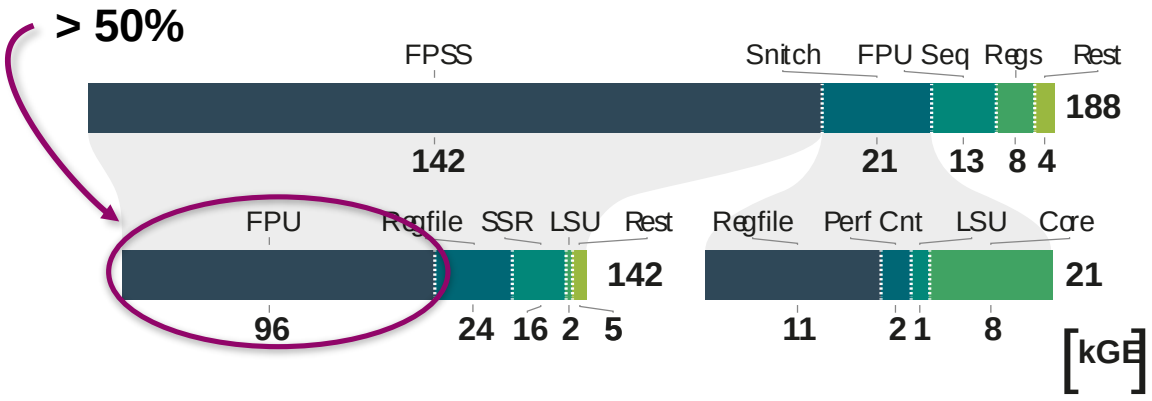


Spending energy where it contributes to the result → **High Efficiency**

Baikonur

The first Snitch-based test chip.





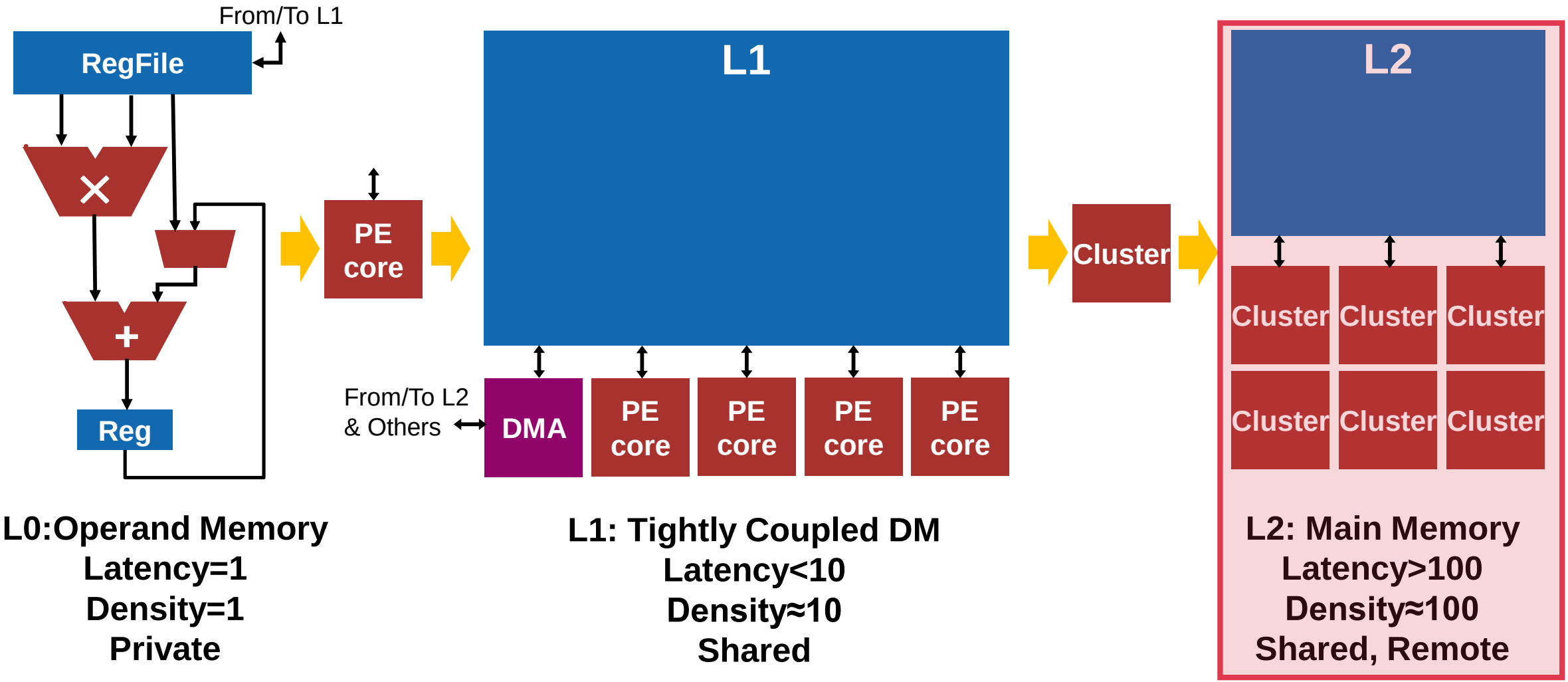
- **Snitch compute cluster:**
  - 8x RV32G Snitch cores
  - 8x large FPUs
  - > 40% energy spent on FPU
  - FPU utilization >80%+ (10% banking conflicts)
  - High energy-efficiency of **~80 Gdpflop/s/W**
  - **104 Gspflop/s/W** similar to accelerators
- **9 mm² prototype in GF 22FDX**
- **Testbed for 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: the Chip(let) (Clusters+Off-die Mem)



L0:Operand Memory
Latency=1
Density=1
Private

L1: Tightly Coupled DM
Latency<10
Density≈10
Shared

L2: Main Memory
Latency>100
Density≈100
Shared, Remote

From Snitch Cluster

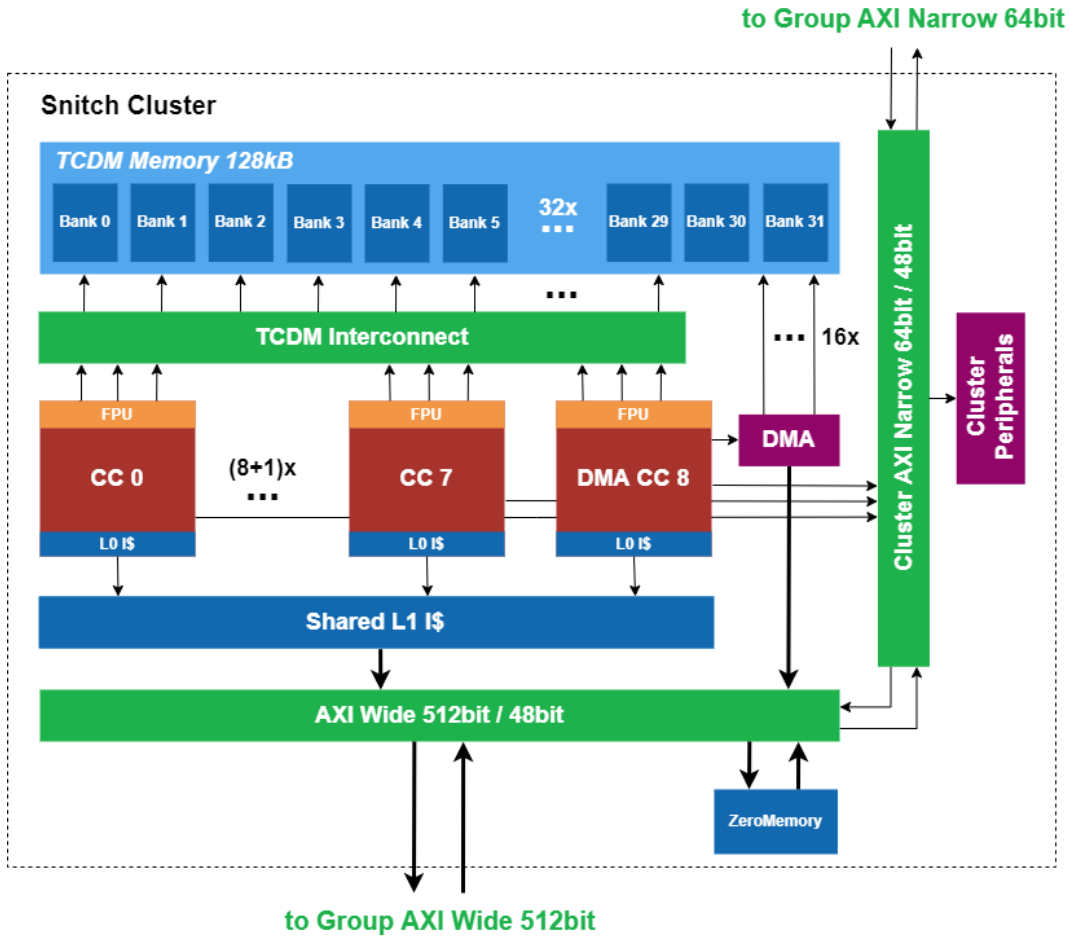
8 Snitch compute cores
- Single-stage, small Integer control core

9th Core: DMA
- 512 bit data interface
- Efficient data movement

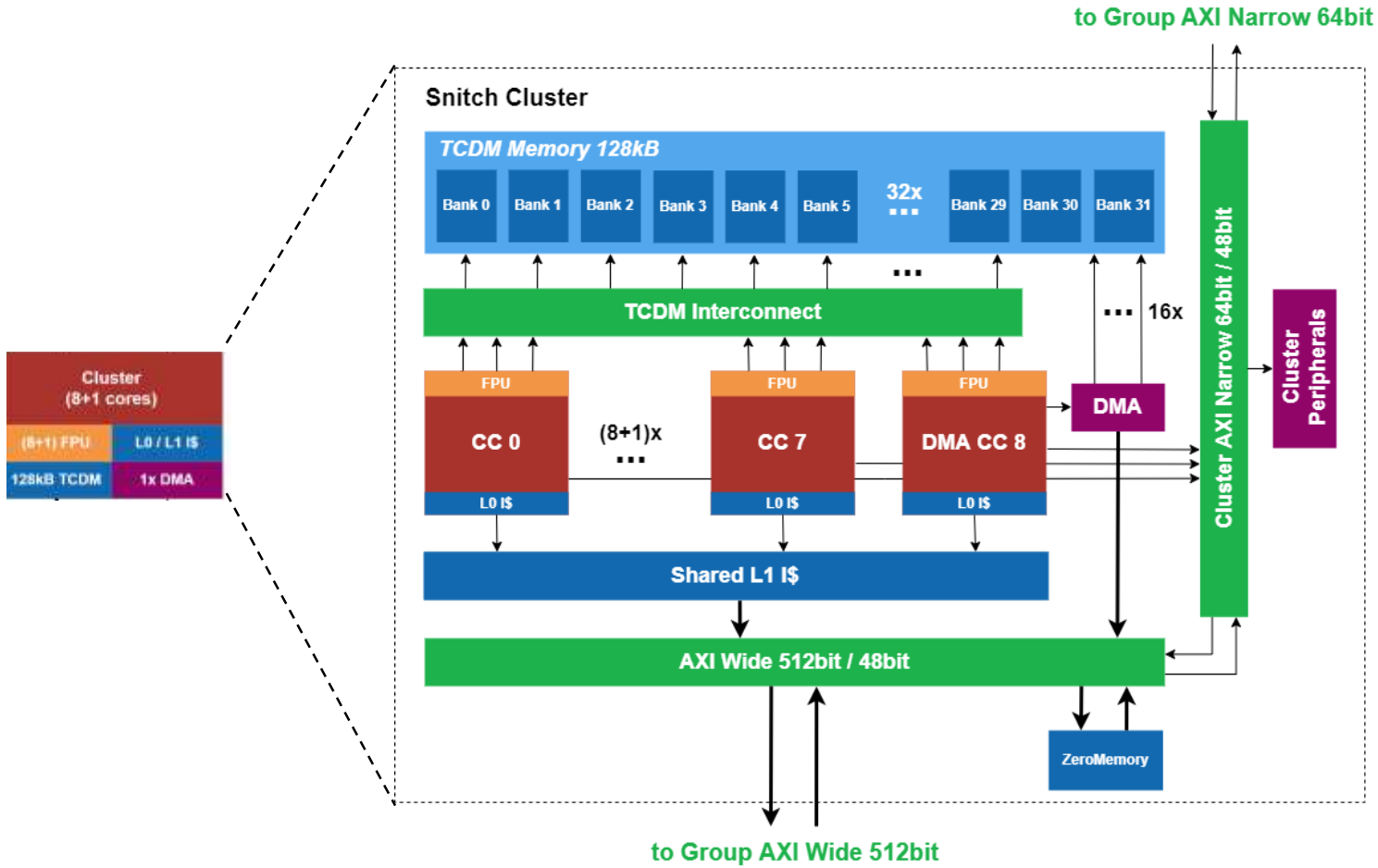
128 kB TCDM
- Scratchpad for predictable memory accesses
- 32 Banks

Custom ISA extensions
- Xfrep, Xssr
- **New: Xissr sparsity support**

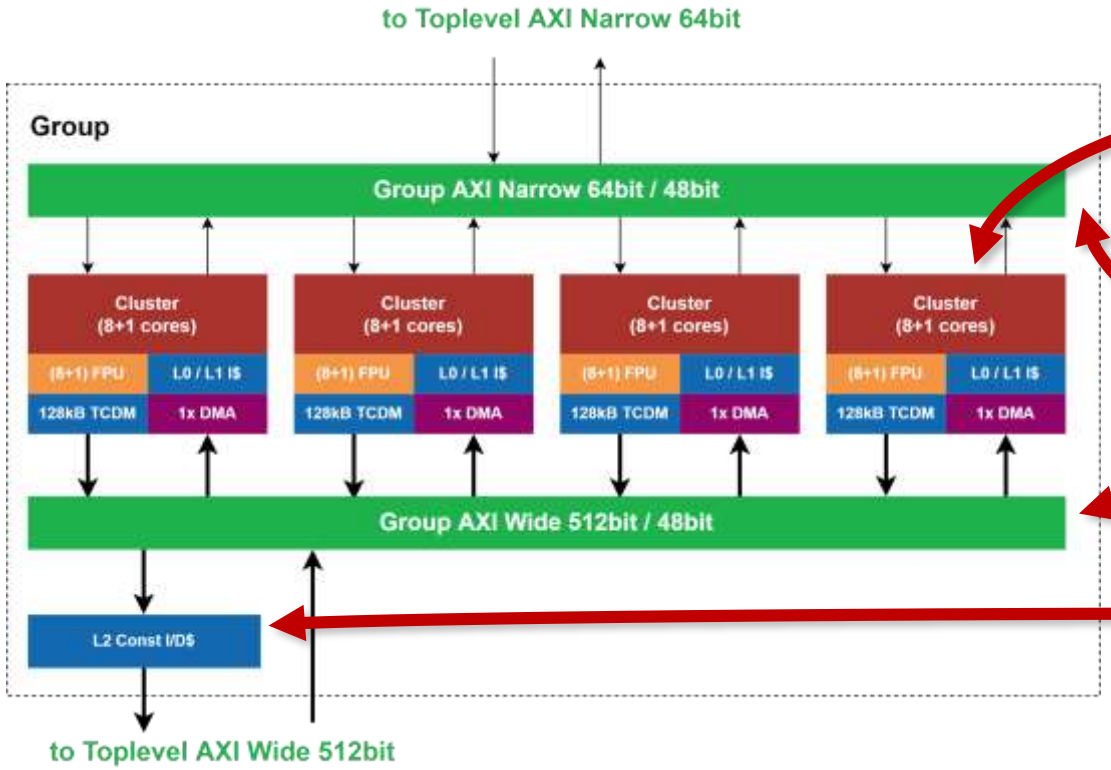
1 FPU per Snitch core
- Decoupled and heavily pipelined
- Multi-format FPU (+SIMD)
- **New: Minifloat support + SDOTP**



To Snitch Group



Snitch Group



4 Clusters per Group

- Single-stage, small Integer control core

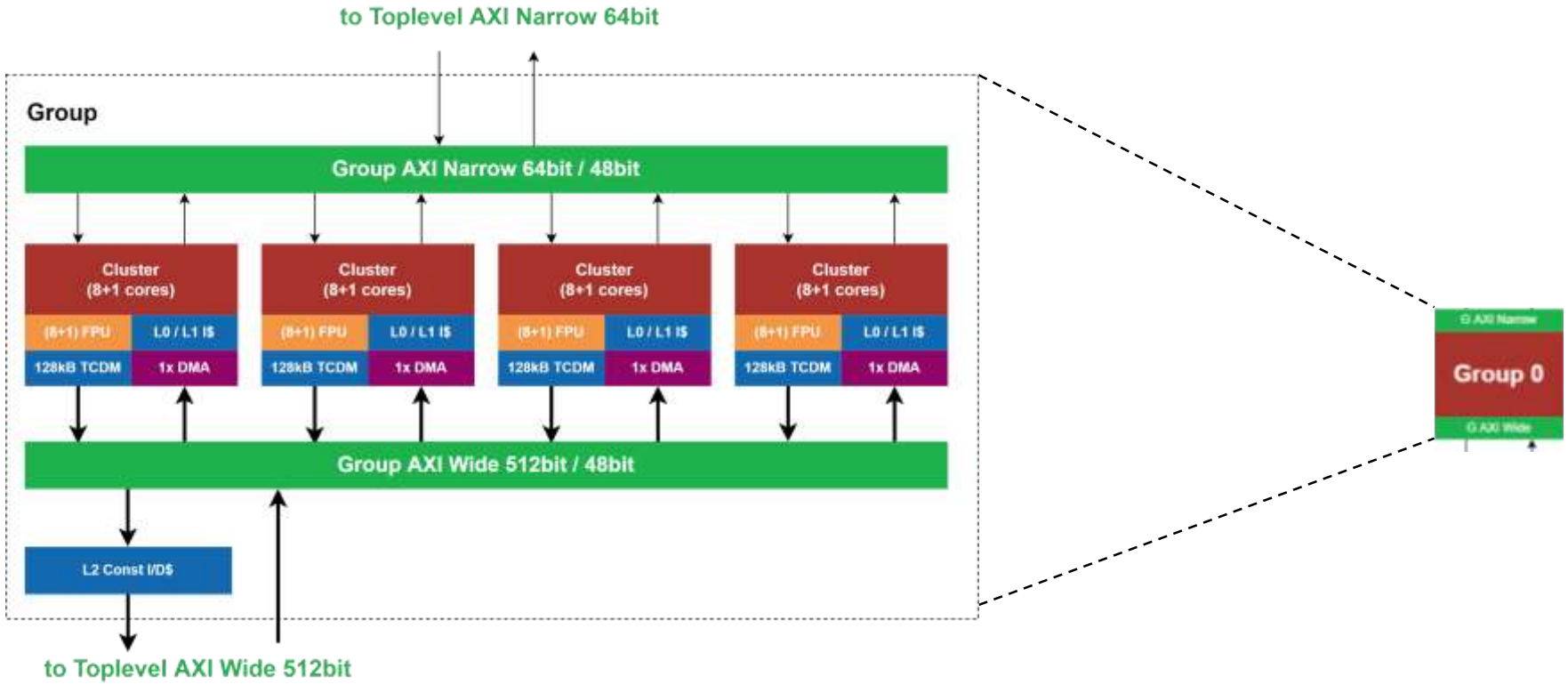
2 AXI Routers

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

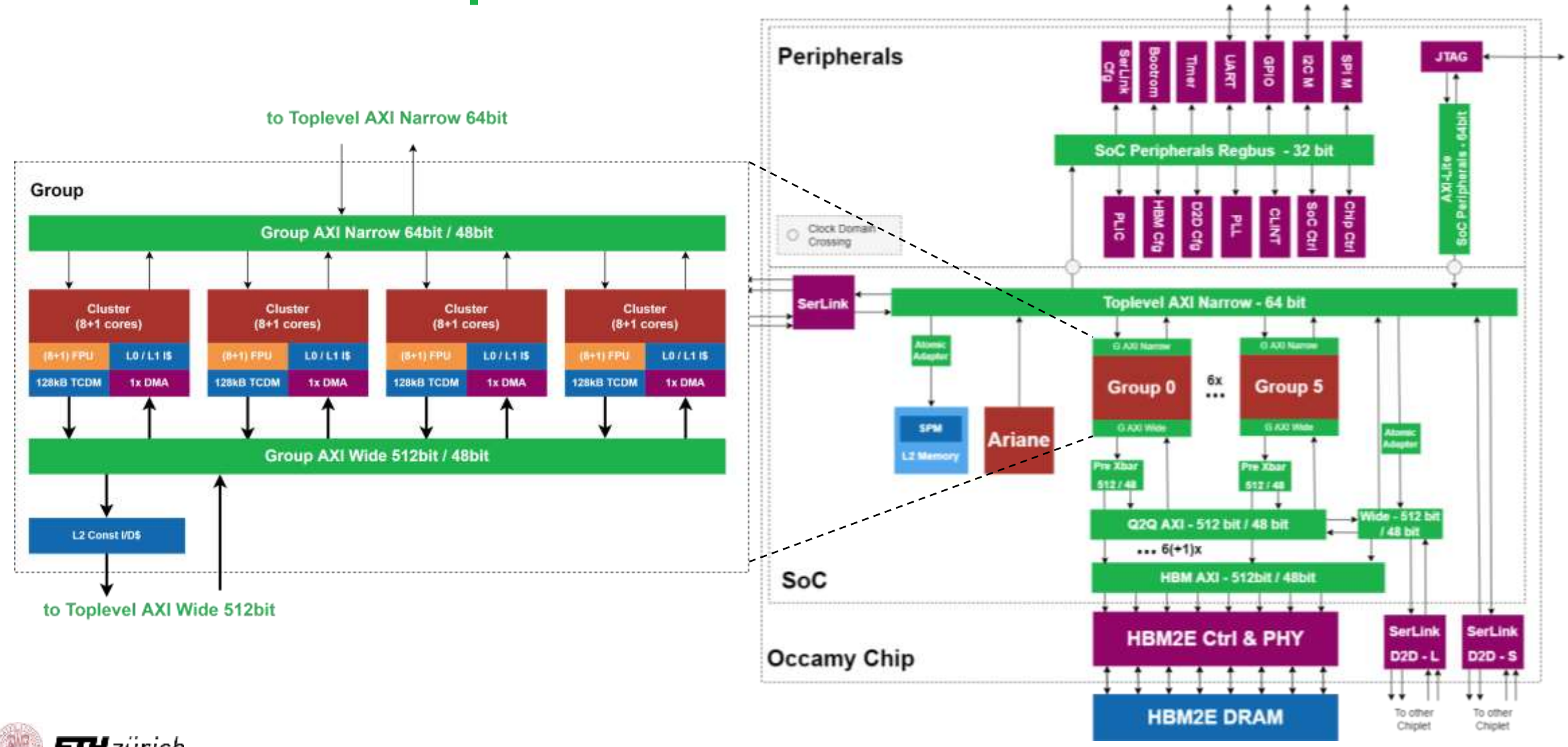
Constant Cache

- D/I-Cache hierarchy

From Snitch Group



To Multi-Group



Occamy – Chiplet Architecture

2 AXI Routers

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

Peripherals

- Complex address space management

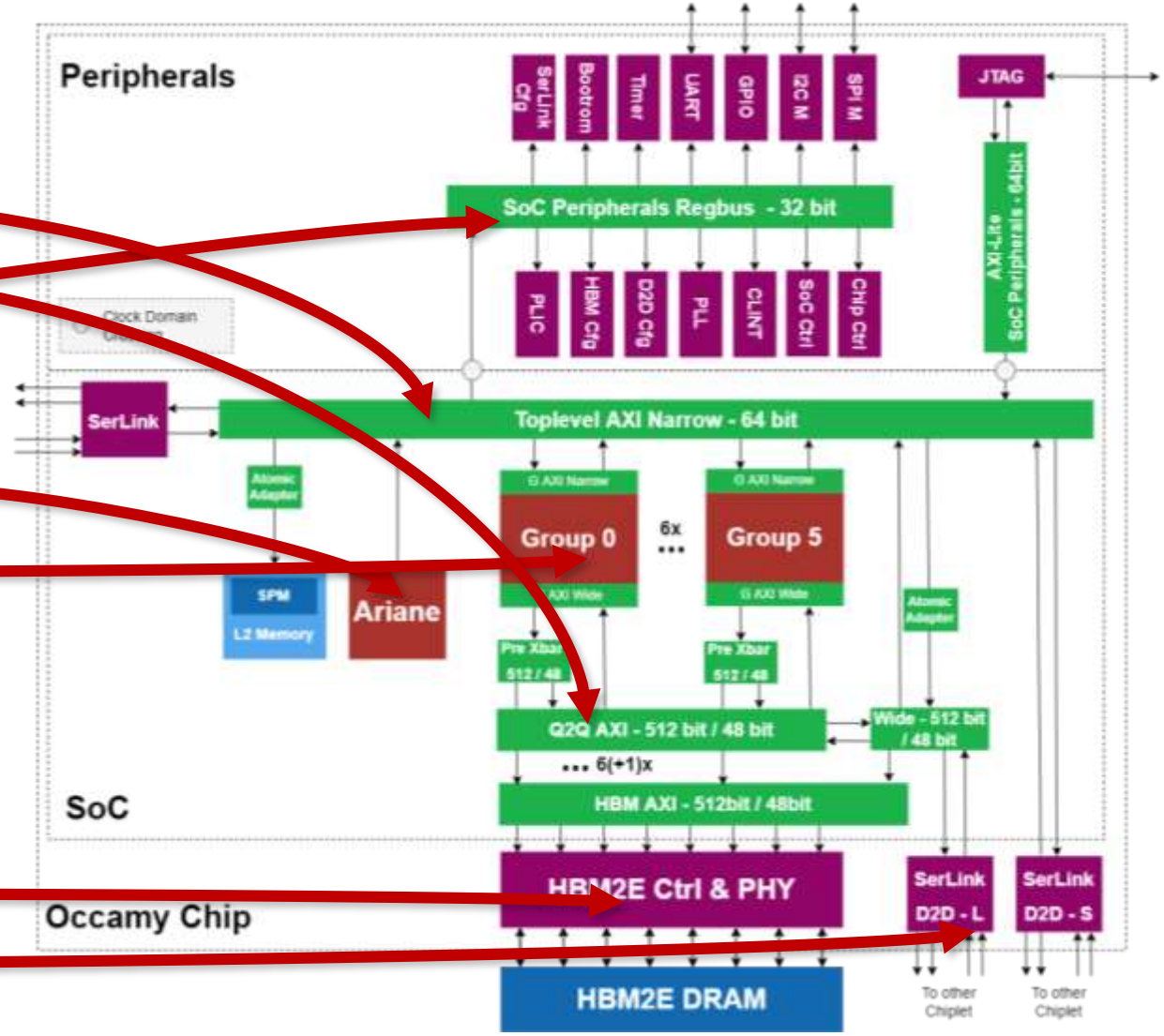
Linux-capable manager core CVA6

6 Groups: 216 cores/die

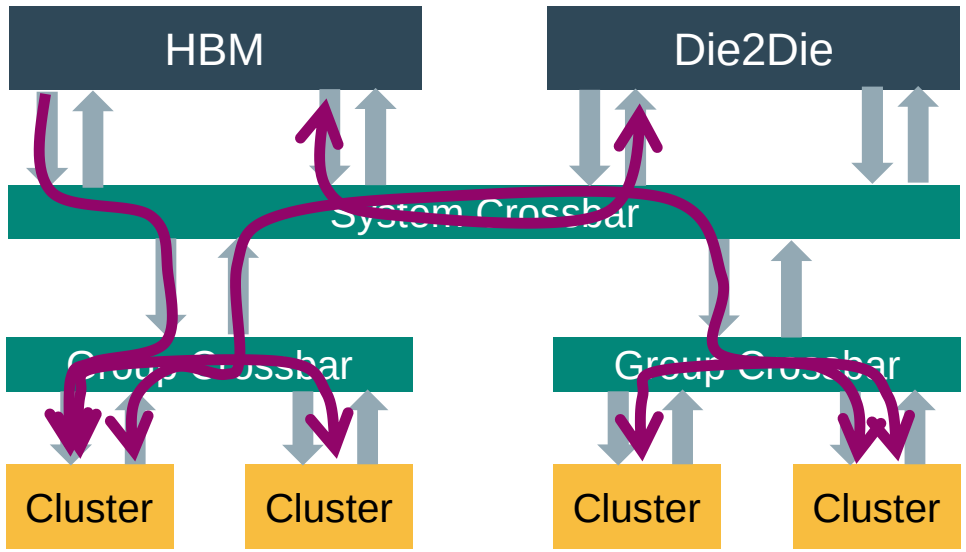
- 4 cluster / group:
  - 8 compute cores / cluster
  - 1 DMA core / cluster
- 512bit Constant Cache

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!

[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.

Silicon: Chiplets + 2HBM2e

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

Dual Chiplet Occamy:

- Technology: GF12LP+
- Area: 73mm2

Interposer Hedwig:

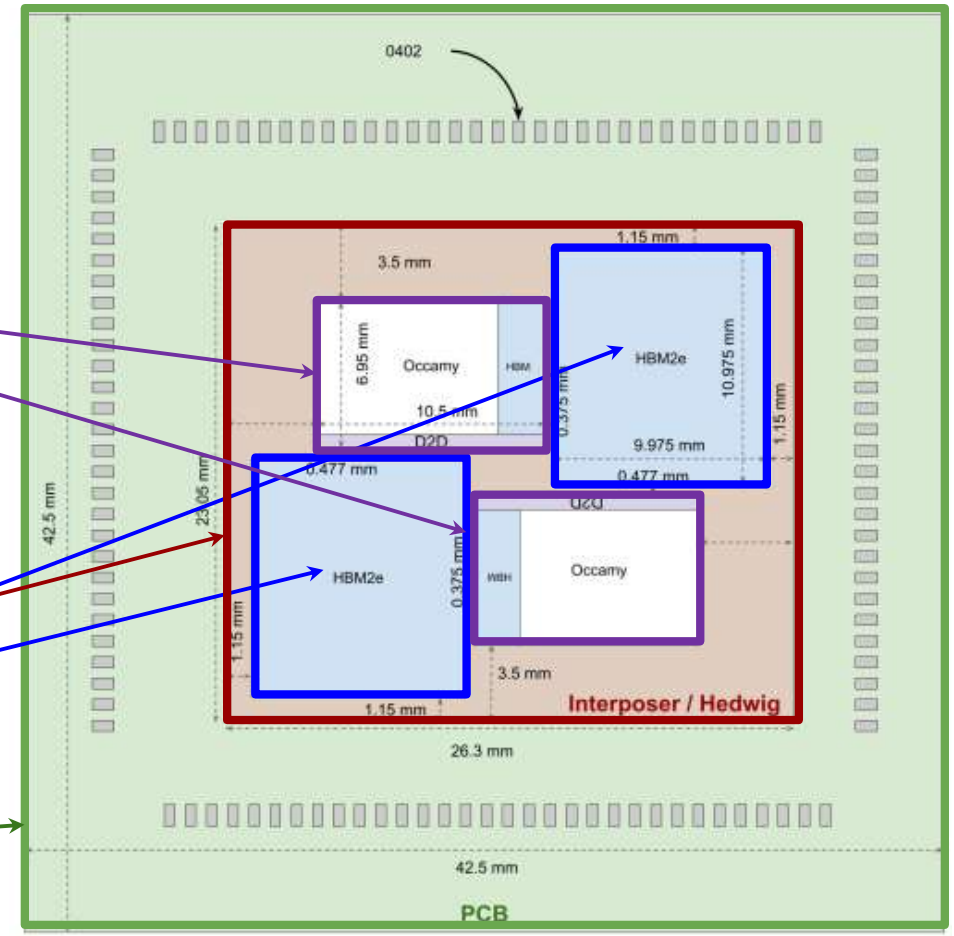
- Technology: 65nm, passive (only BEOL)
- Area: 26.3mm x 23.05mm

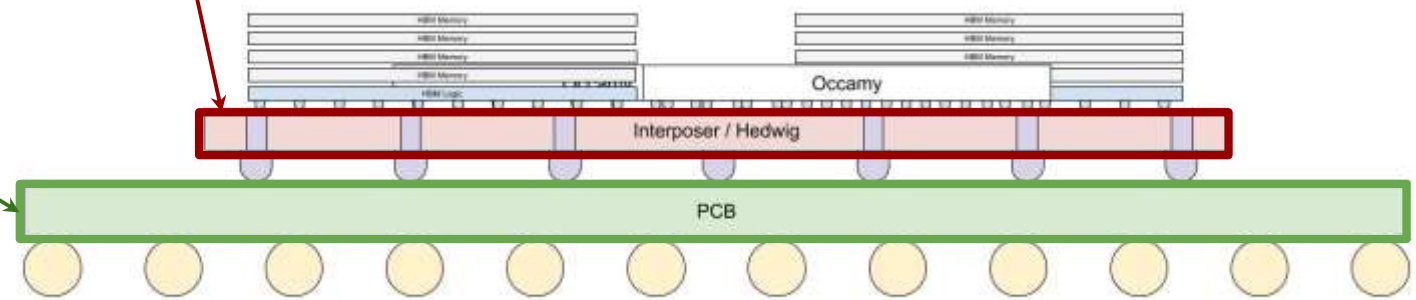
HBM2e:

- 16GB HBM2e (Micron)

Fan-out PCB:

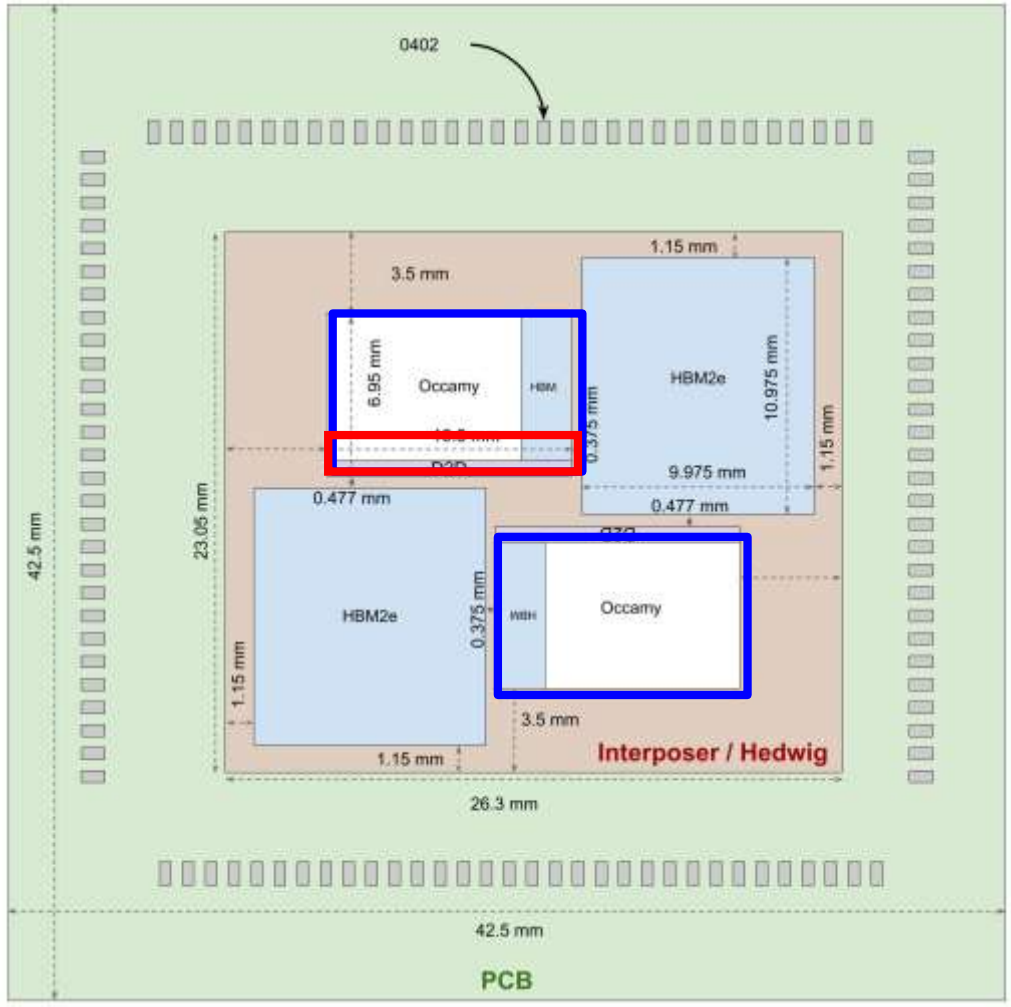
- Low-CTE

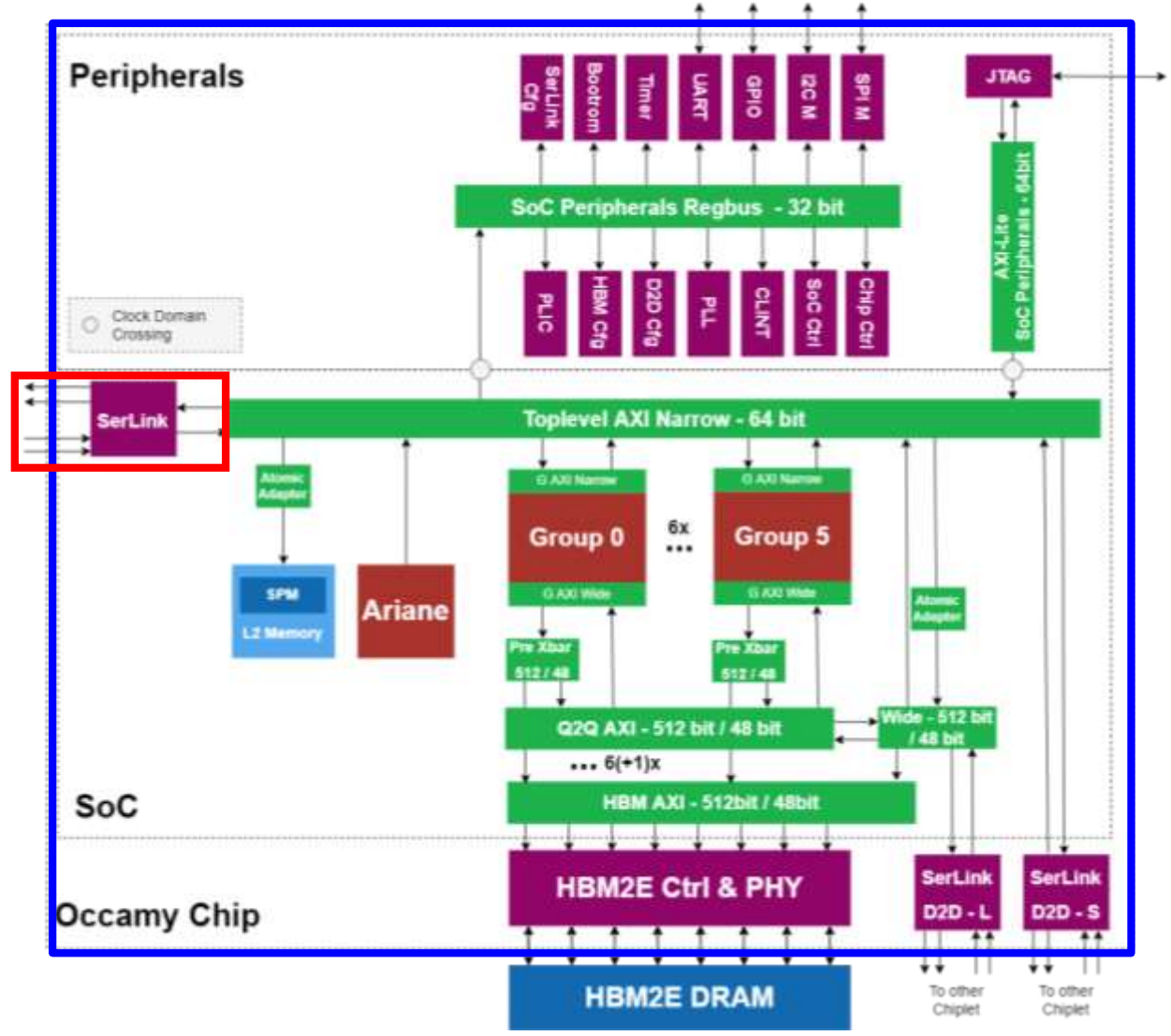




GlobalFoundries
SYNOPSYS
EUROPRACTICE
IC SERVICE
Rambus
Micron
ETH zürich
Fraunhofer IZM

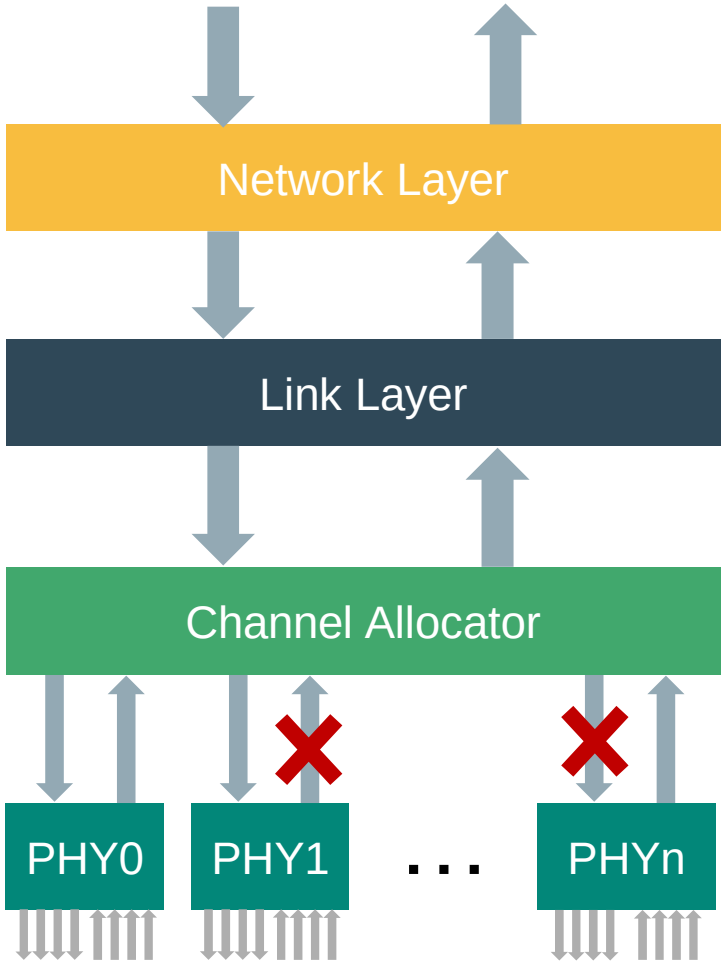
Occamy's C2C Link





C2C Link

Scalable and fault tolerant Chiplet2Chiplet Link



Network Layer

- Full AXI4 interface
- AXI4 to AXI stream converter

Data Link Layer

- Credit-based flow control
- RX synchronisation

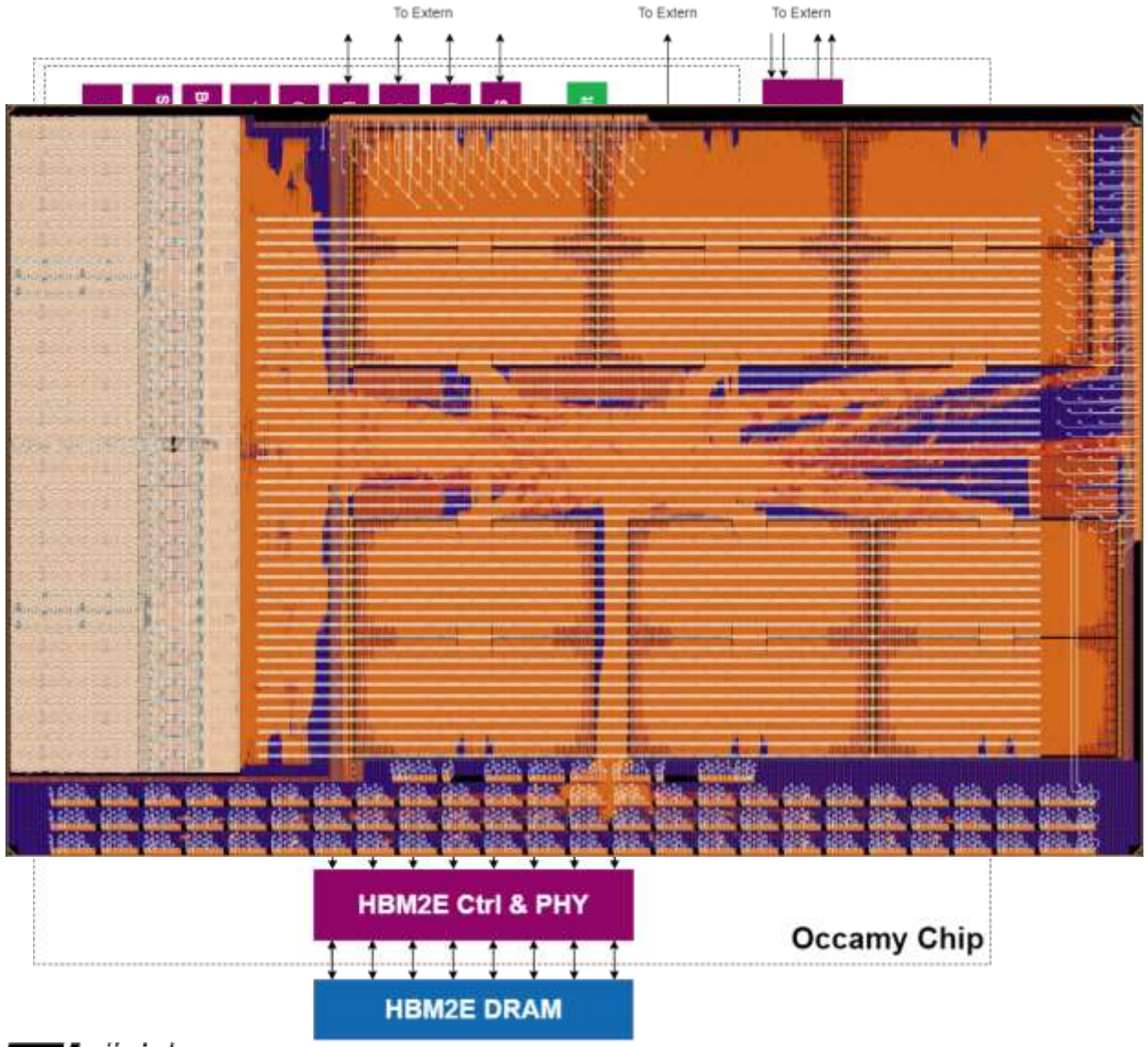
Channel Allocator

- Chops and reshuffles payload
- **Fault tolerance** mechanisms

Physical layer (decoupled)

- Source-synchronous DDR sampling
- **Scalable** number of channels
  (38 x 8-bit full-duplex DDR channels in Occamy)
- Compatible with fast serial phy (e.g. HBI)

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)

ETH zürich

Peak 384 GDPflop/s per chiplet – Taped-out in July (fully OSHW)

Efficient Chiplet architecture in Perspective

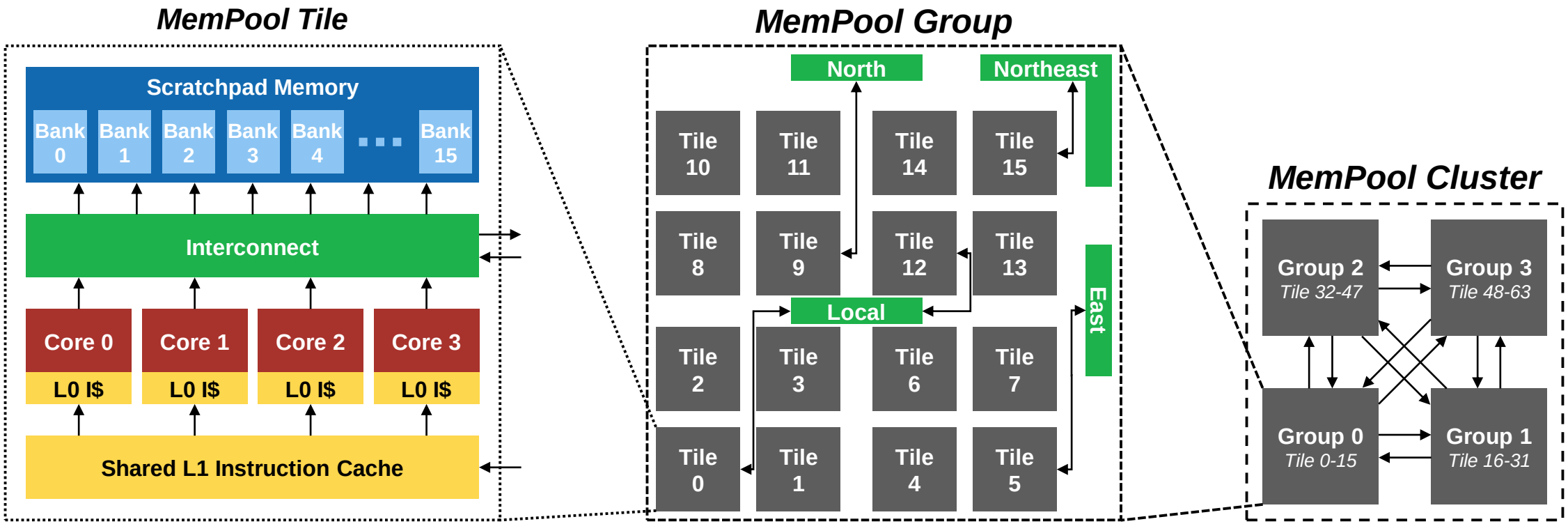
1. Multi-cluster single-die scaling → strong latency tolerance, modularity
2. NoC for flexible Clus2Clus, Clus2Mem traffic → reduce pressure to Main memory
3. Top level NoC Routes to “local main memory” / “global main memory” balanced BW
4. Modular chiplet architecture: HBM2e, NoC-wrapped C2C, multi-chiplet ready

Back to the cluster… Can we make it Bigger?

- **Why?**
  - Better global latency tolerance if **L1 > 2*Latency*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

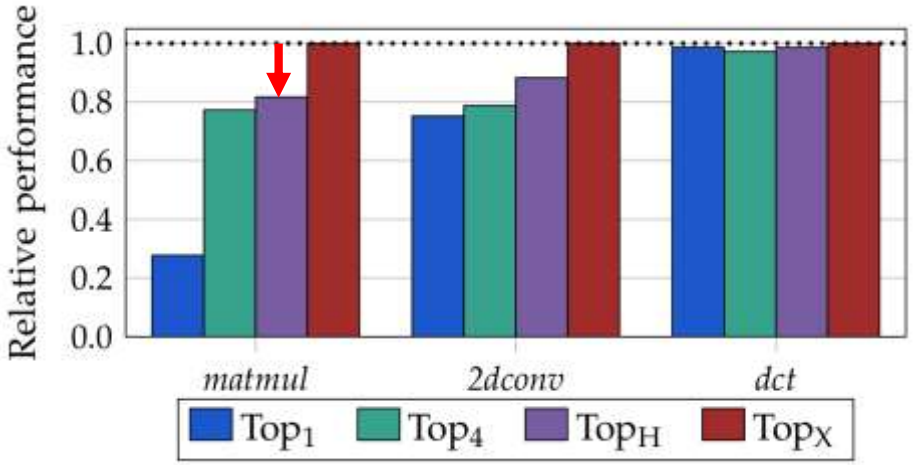
- **Tile**
  - 4 32-bit cores
  - 16 banks
  - **Single cycle** memory access
- **Group**
  - 64 cores
  - 256 banks
  - **3-cycles latency**
- **Cluster**
  - 256 cores
  - 1 MiB of memory (1024 banks)
  - **5-cycles latency**



TopH: Butterfly Multi-stage Interconnect 0.3req/core/cycle

Benchmarks: can we compete with the impossible?

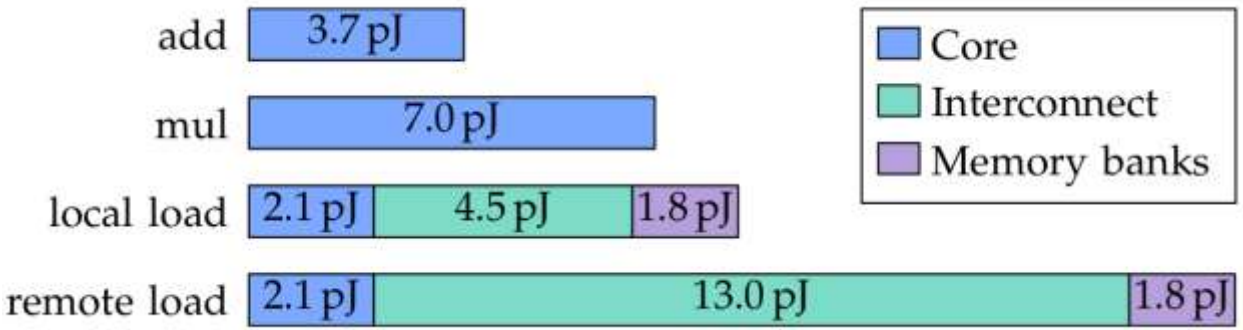
- Baseline: idealized Top_X
  - Fully connected logarithmic crossbar between 256 cores and 1024 banks
- Cycle-accurate RTL simulation
  - *Matmul*
    - Multiplication of two 64 × 64 matrices
  - *2dconv*
    - 2D Convolution with a 3 × 3 kernel
  - *dct*
    - 2D Discrete Cosine Transform on 8 × 8 blocks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

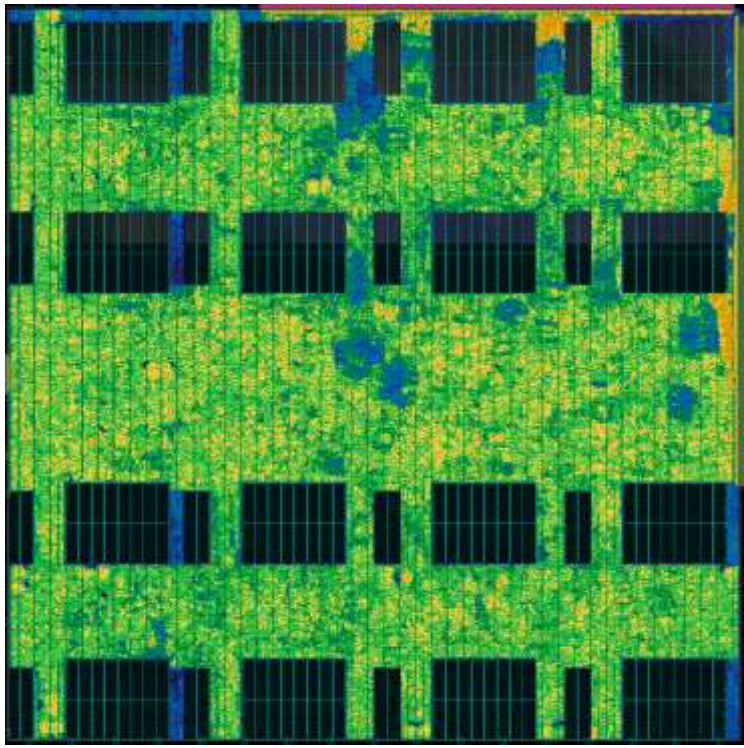


- Top_H has a performance penalty of at most 20%, on all kernels

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

- Breakdown of the energy consumption per instruction:



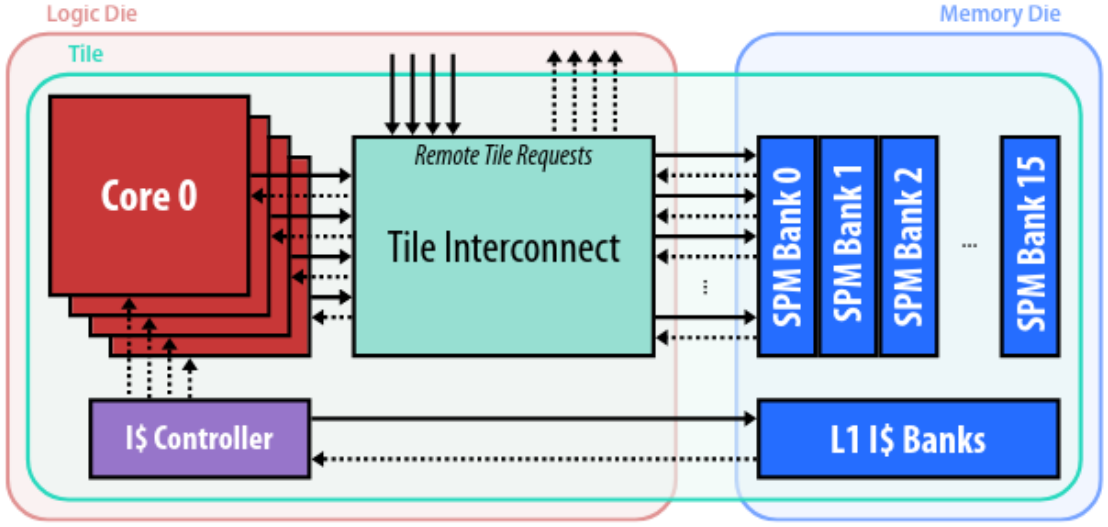


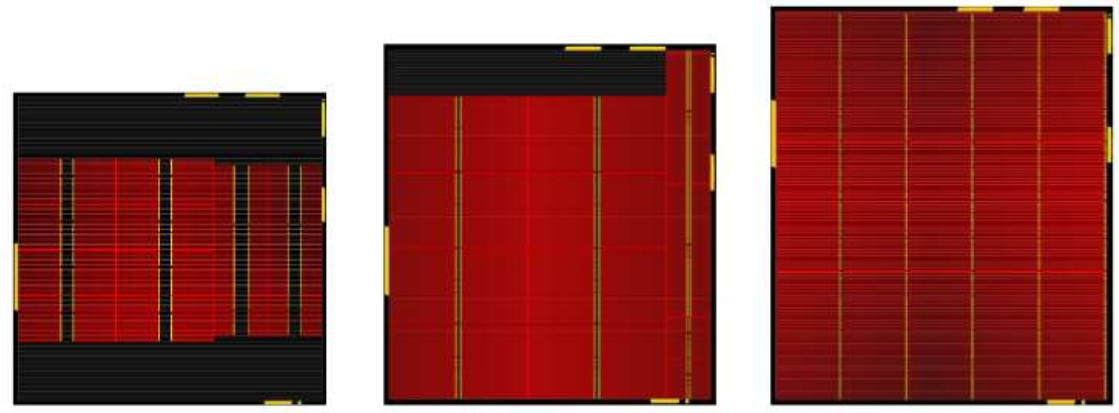
- Local loads consume about as much energy as a *mul*
  - About half of it, 4.5pJ, by the interconnect
- Remote loads: twice the energy of a local load
  - Despite crossing the whole cluster, twice!

Terapool (1024 cores) seems viable @ 7 cycles latency

MemPool-3D

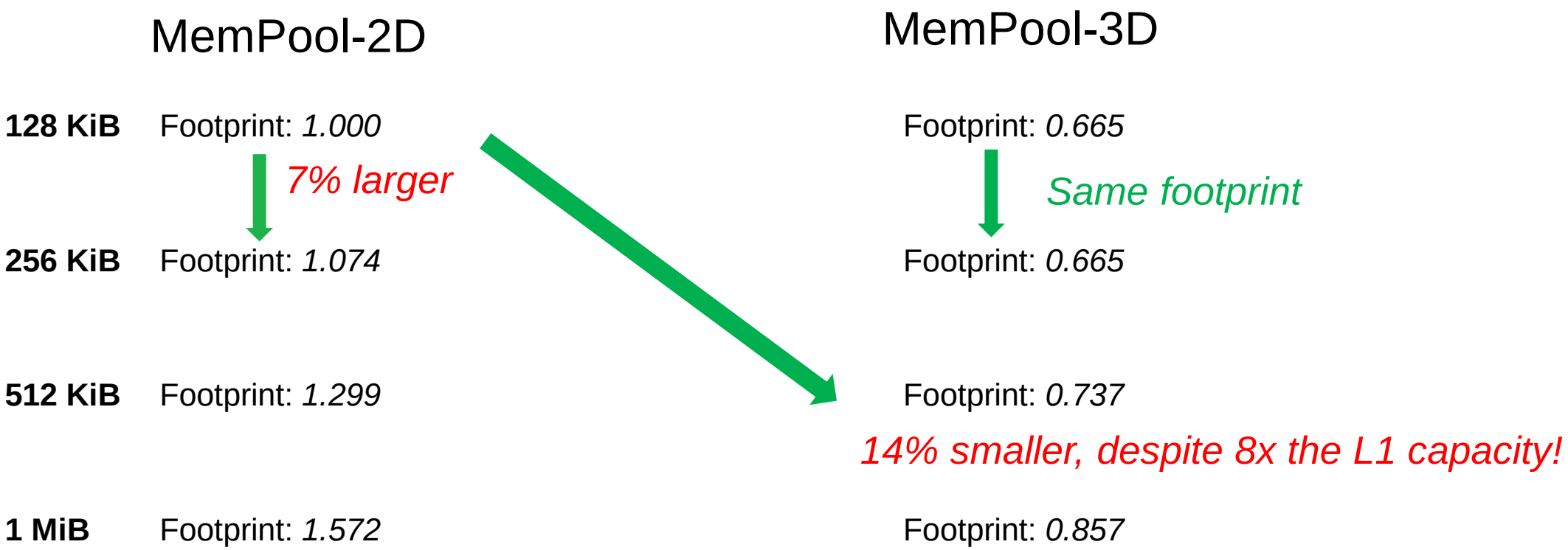
- Memory-on-Logic implementation of MemPool
- Implementation of MemPool with 1, 2, 4, 8 MiB of L1
- Leading to a higher utilization of the memory die





(a) MemPool-3D$_{1\,\text{MiB}}$. (b) MemPool-3D$_{4\,\text{MiB}}$. (c) MemPool-3D$_{8\,\text{MiB}}$.

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

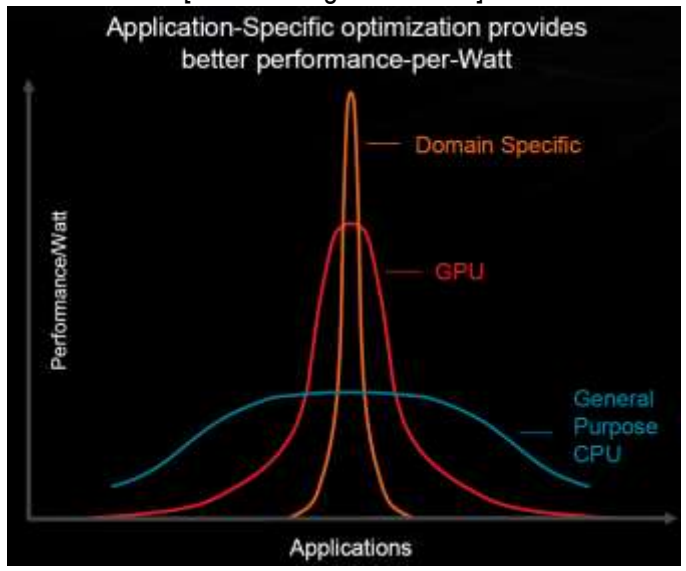


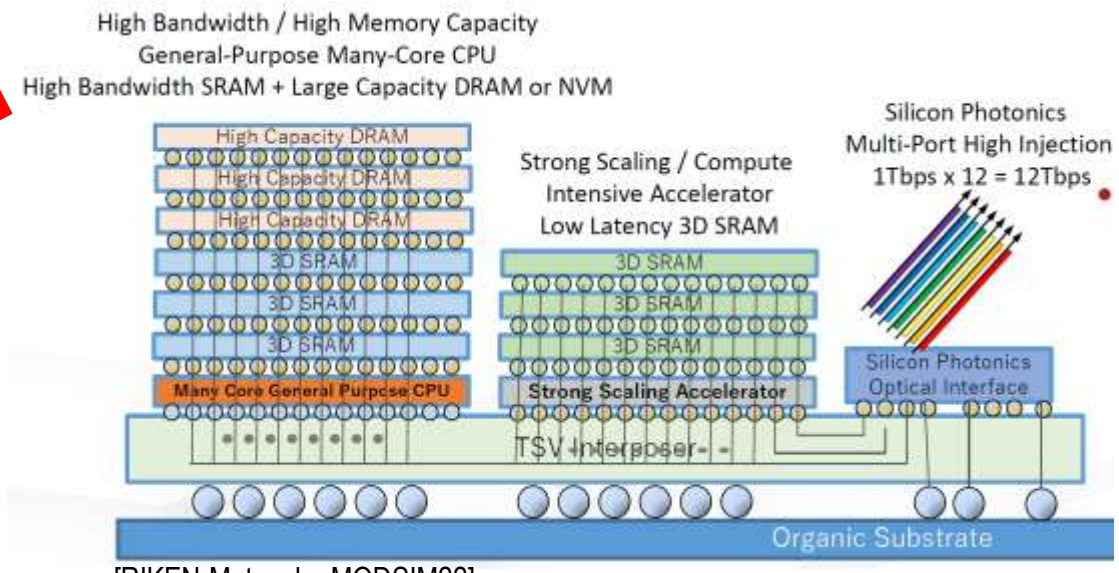
Similar trends for **Wire length** and **#Buffers**

Conclusion

- Energy efficiency quest: PE, Cluster, SoC
- Key ideas
  - Deep PE optimization → 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
  - “Commoditized” chiplets: 2.5D, 3D
  - Computing “at” memory (DRAM mempool)
  - Coming: optical IO and smart NICs, swiches
- Challenges:
  - RV Host?
  - RV HPC software ecosystem?

[AMD Naffziger ISCAS22]





[RIKEN Matsuoka MODSIM22]

Monte Cimone: the first RISC-V Cluster

Designed for HPC “pipe cleaning”

RISC-V Challenge… SW ecosystem (?)

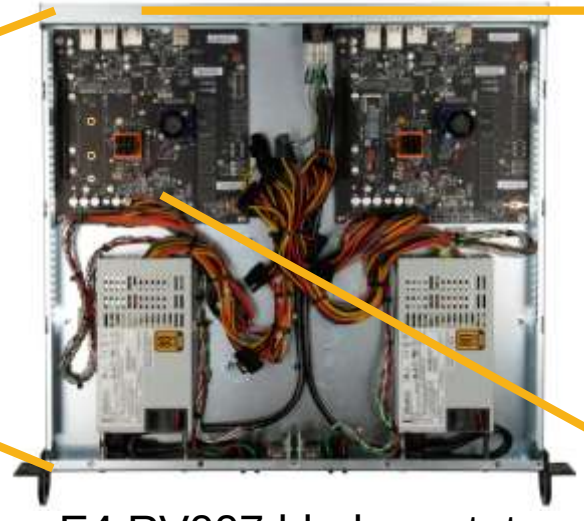
In 2021 we designed and built **Monte Cimone**, the **first physical prototype** and test-bed of a **complete RISC-V (RV64) compute cluster** integrating **compute, interconnect,** *a **complete software stack for HPC** and a full-featured **system monitoring infrastructure**.*

1. Ported and assessed the maturity of a HPC software stack composed of:
   - SLURM job scheduler, NAS filesystem, Spack package manager
   - compilers toolchains, scientific and communication libraries,
   - a set of HPC benchmarks and applications,
   - ExaMon datacenter automation and monitoring framework.

2. Characterized the HPL and STREAM benchmarks w. the toolchain and libraries installed by Spack: Compared against Armida (ARMv8a ThunderX2) and Marconi100 (ppc64le, IBM Power9)

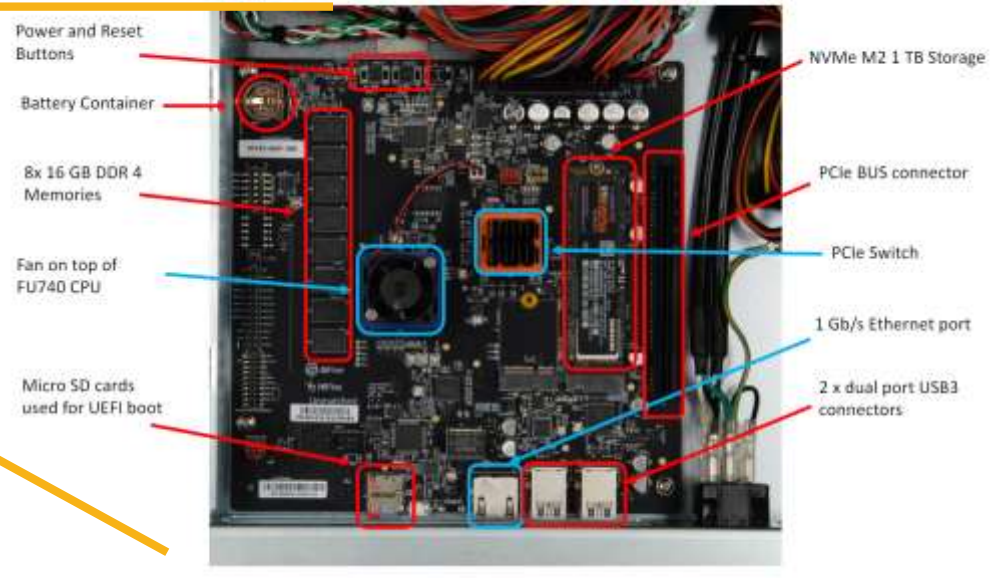
3. Measured and analyzed MC power and energy using Examon

Monte Cimone Hardware



E4 RV007 blade prototypes

SiFive HiFive Unmatched board

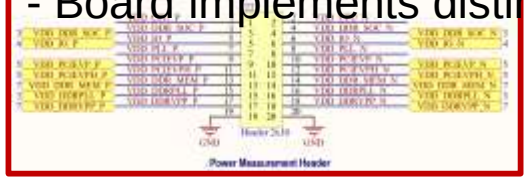


4x E4 RV007 1U Custom Server Blades:

- **2x SiFive U740 SoC** with **4x U74 RV64GCB** cores
- 16GB of **DDR4**
- 1TB node-local **NVME storage**
- **PCIe expansion** card **w/InfiniBand HCAs**
- **Ethernet + IB parallel networks**

SiFive U740 SoC w. 7 separated power rails:

- Core complex, IOs, PLLs, DDR subsystem and PCIe one.
- Board implements distinct shunt resistors

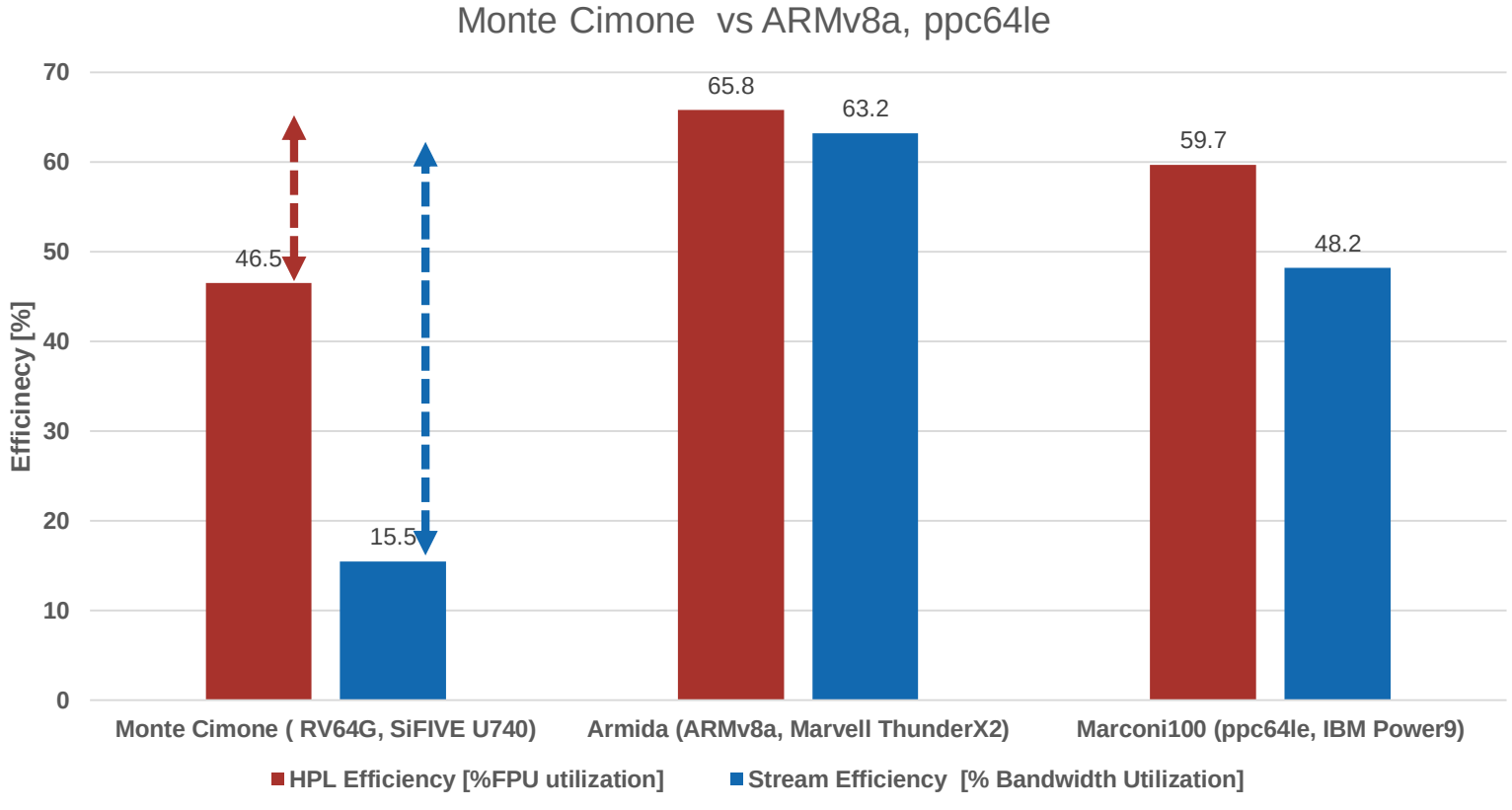


Performance Characterization

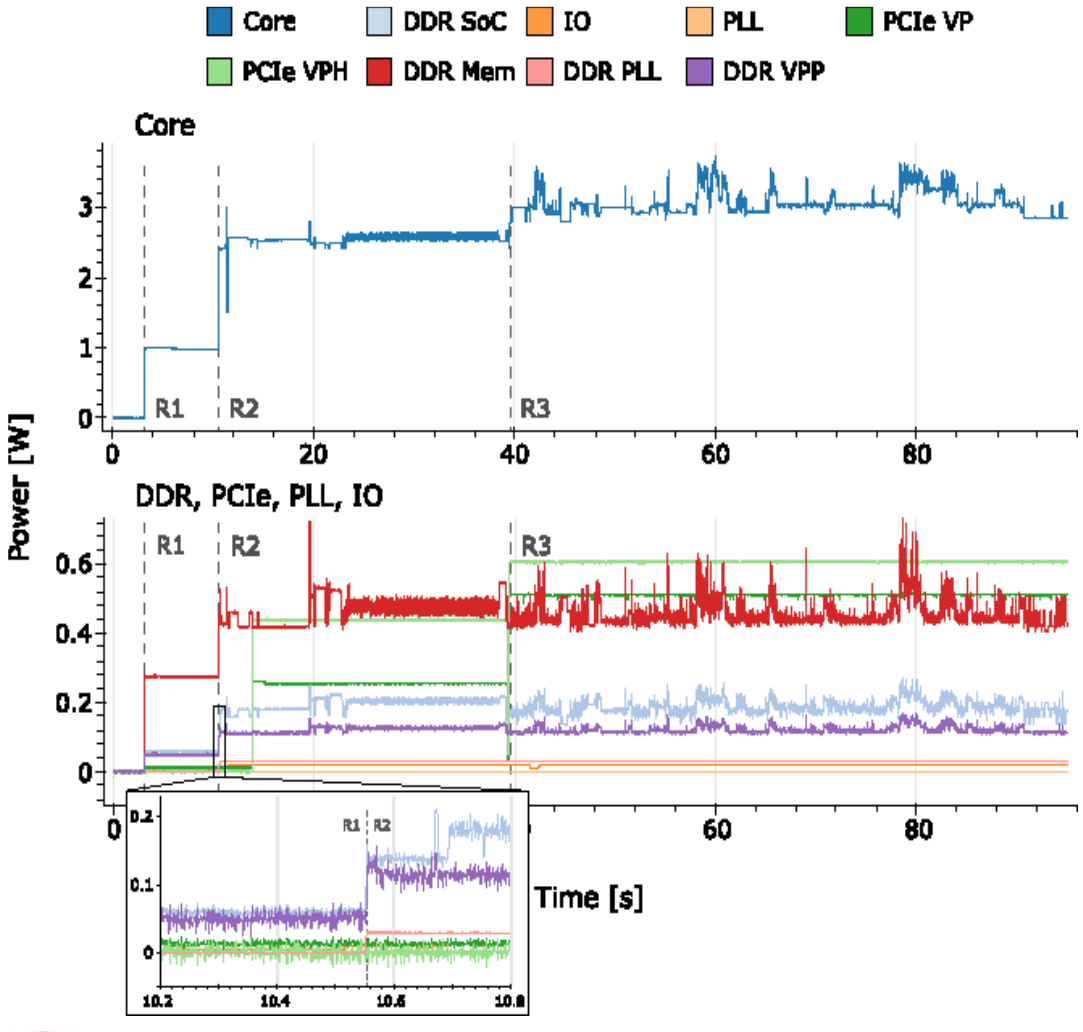
Monte Cimone vs ARMv8a, PPC64le:

HPL and Stream benchmarks on two SoA computing nodes:

- *Marconi100 (ppc64le, IBM Power9)*
- *Armida (ARMv8a, Marvell ThunderX2)*
- *Same benchmarking boundary conditions*
  - *Vanilla unoptimized libraries*
  - *software stack deployed via SPACK package manager*



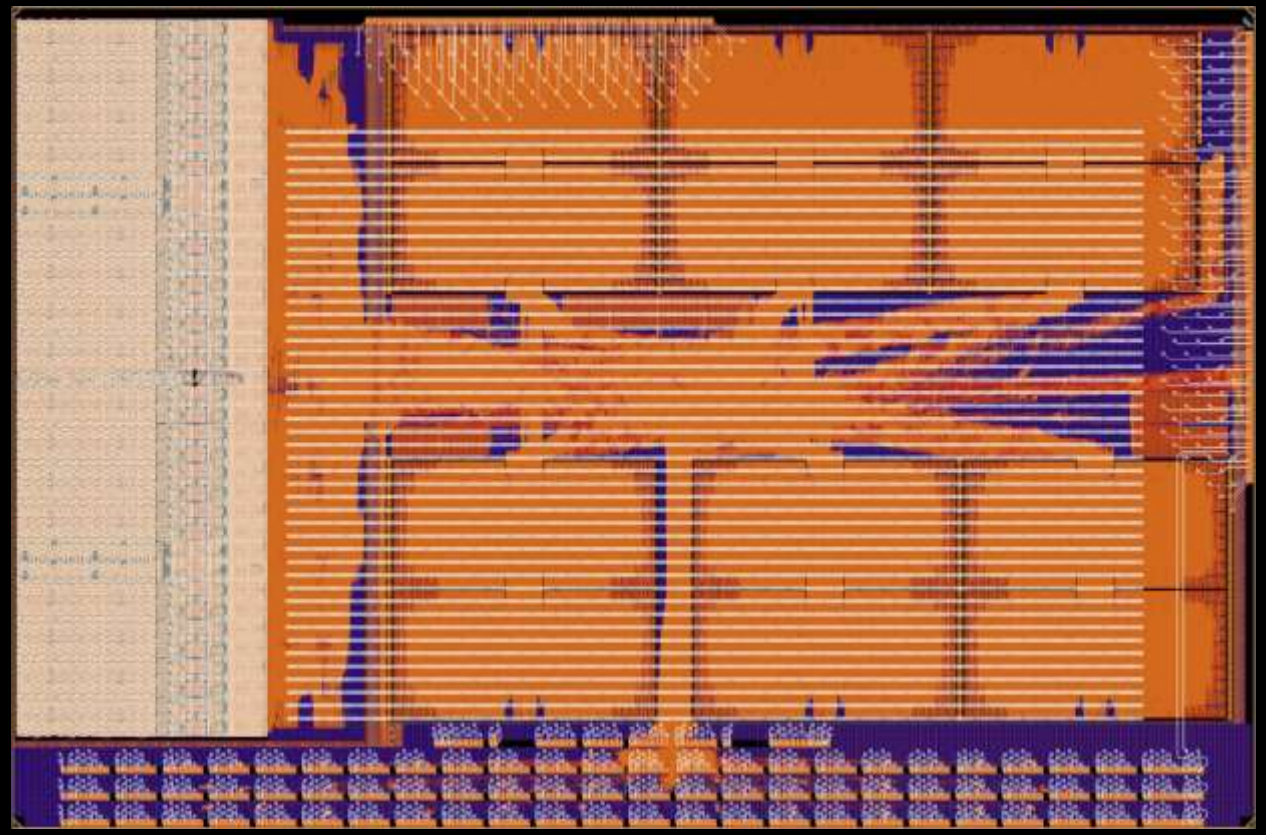
Power Characterization



Line	Idle		HPL		STREAM.L2		STREAM.DDR		QE		Boot R1	Boot R2
	[mW]	[%]	[mW]	[%]	[mW]	[%]	[mW]	[%]	[mW]	[%]	[mW]	[mW]
core	3075	64	4097	69	3714	68	3287	62	3825	67	984	2561
ddr_soc	139	3	177	3	170	3	232	4	176	3	59	197
io	20	0	20	0	20	0	20	0	20	0	5	20
pll	1	0	1	0	1	0	1	0	1	0	0	2
pcievp	521	11	527	9	524	10	522	10	530	9	12	231
pcievph	555	12	554	9	554	10	555	10	561	10	1	395
ddr_mem	404	8	440	7	401	7	592	11	434	8	275	467
ddr_pll	28	1	28	1	28	1	28	1	28	1	0	29
ddr_vpp	67	1	90	2	73	1	98	2	95	2	49	122
Total	4810	100	5935	100	5486	100	5336	100	5670	100	1385	4024

Core complex @ boot process:

- 0.981W of leakage only power (32% of the Idle power)
- 0.514W O. S. idle power (17% of the Idle power)
- 1.577W of dynamic and clock tree power (51% of the idle power).



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 Zanghieri, 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

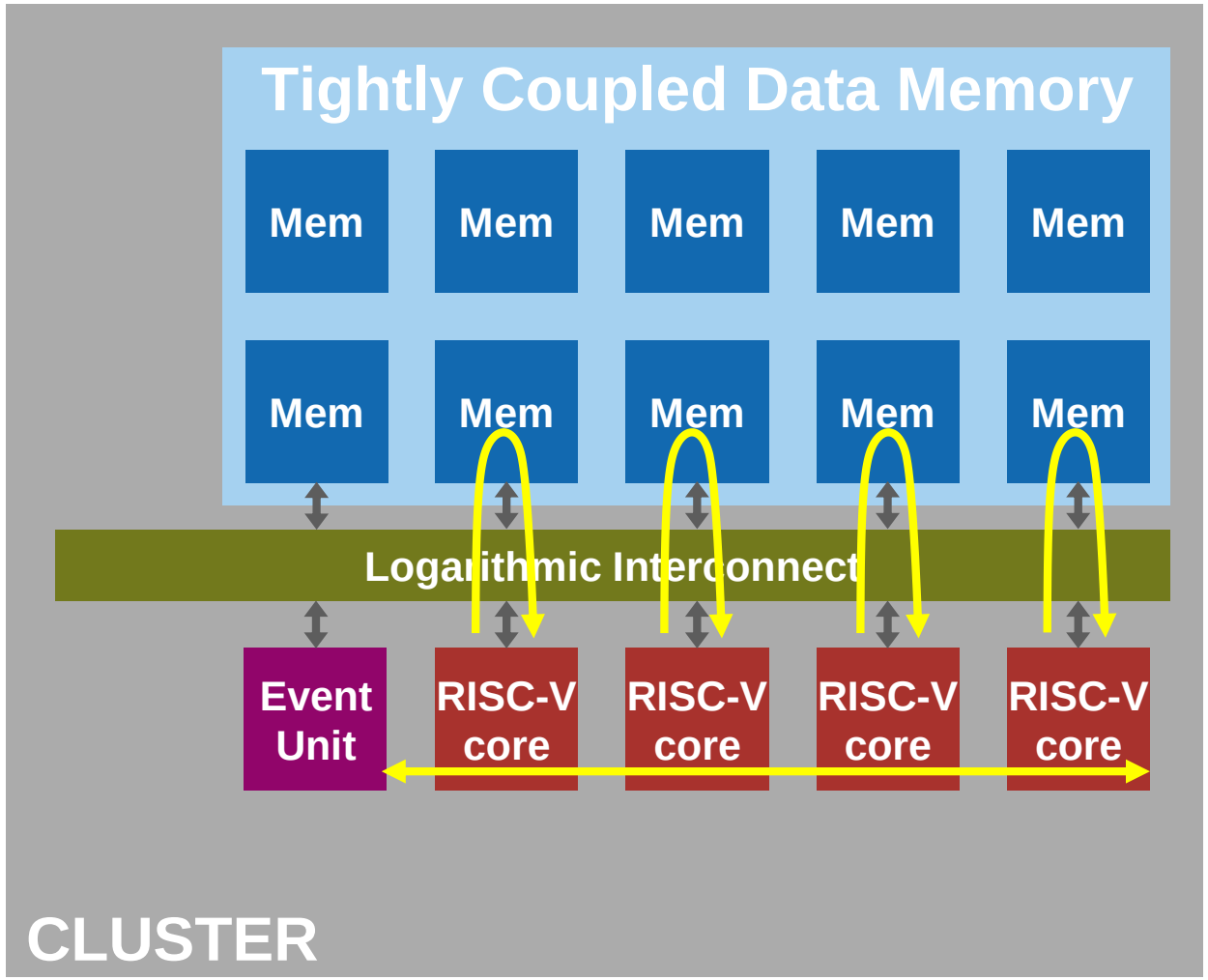
Parallel Ultra Low Power

http://pulp-platform.org

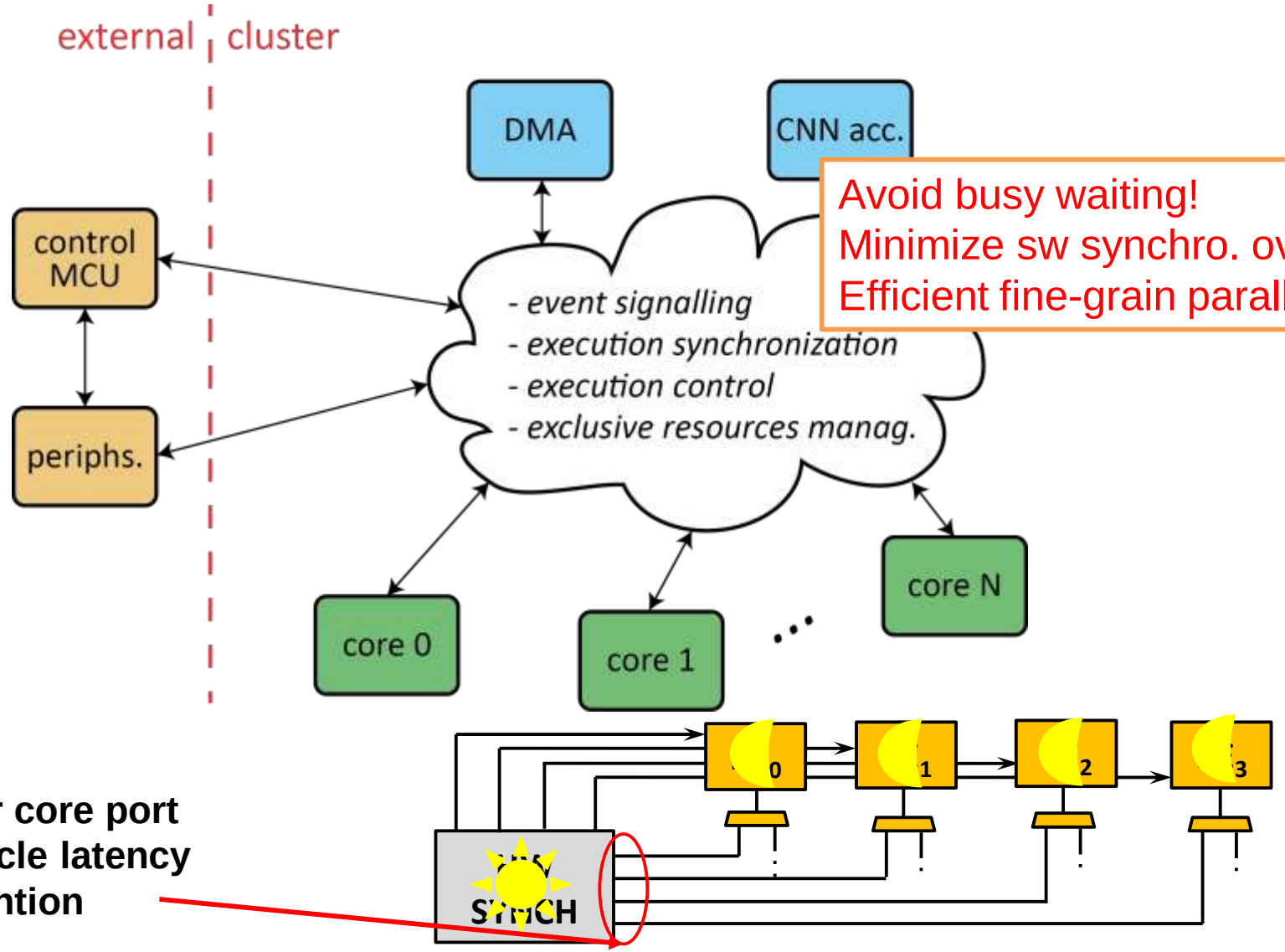
@pulp_platform

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

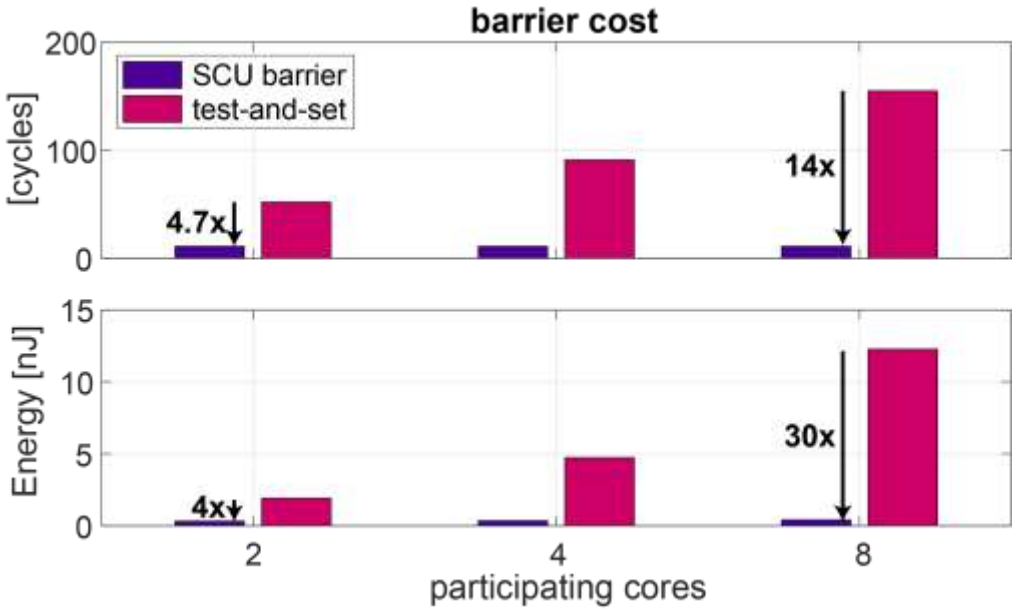


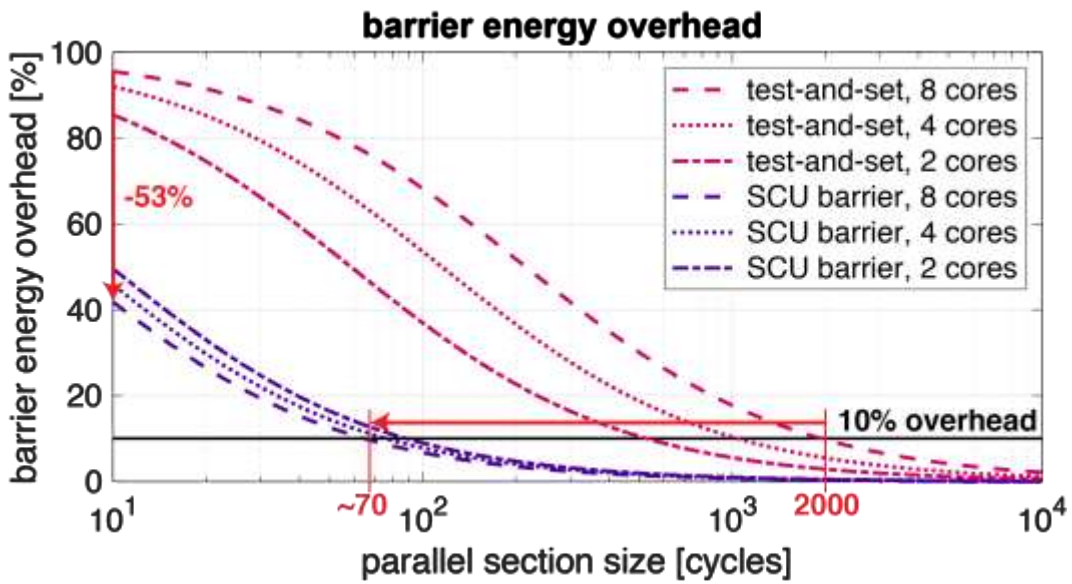
Avoid busy waiting!
Minimize sw synchro. overhead
Efficient fine-grain parallelization

Private, per core port
→single cycle latency
→no contention

F. Glaser, et al, "Energy-Efficient Hardware-Accelerated Synchronization for Shared-L1-Memory Multiprocessor Clusters," in IEEE TPDS, vol. 32, no. 3, pp. 633-648, 1 March 2021.

Barrier: Results

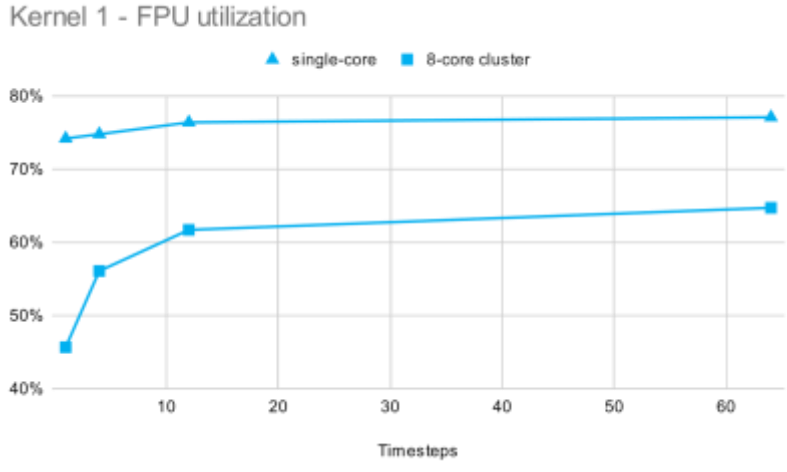


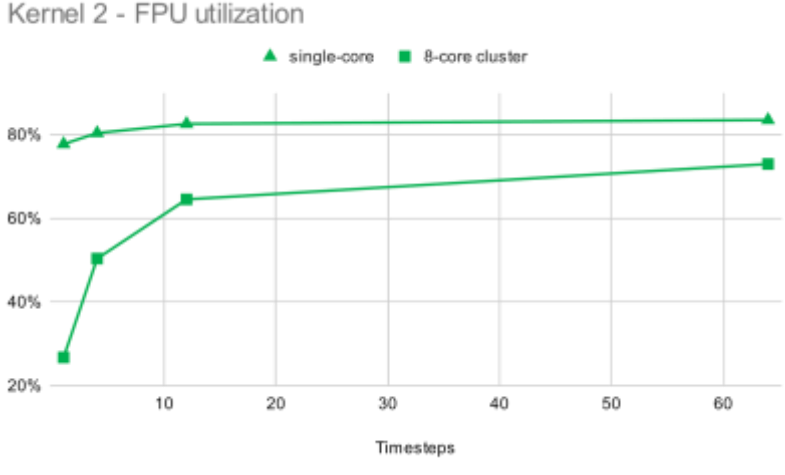


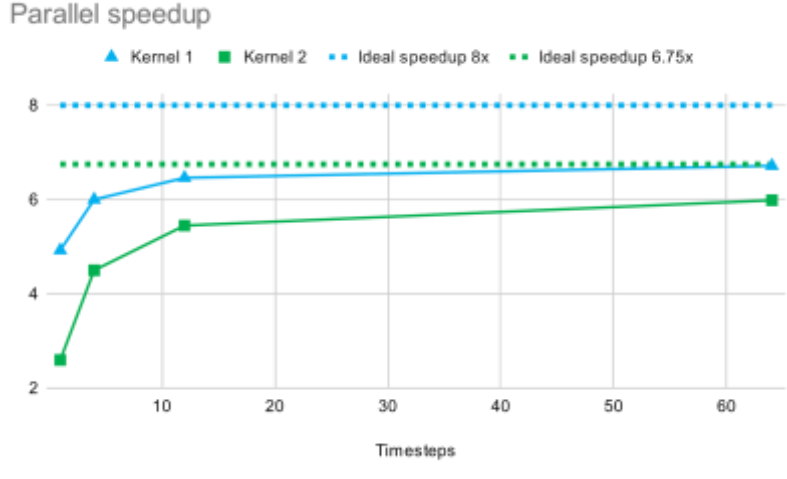
- Fully parallel access to SCU: Barrier cost constant
- Primitive energy cost: Down by up to 30x
- Minimum parallel section for 10% overhead in terms of …
  - … cycles: ~100 instead of > 1000 cycles
  - … energy: ~70 instead of > 2000 cycles

Stencils: Performance Results

- Kernel 1 (16x4x2 output tile, radius 3, 12 timesteps)
  - Single-core FPU utilization: **77%**
  - 8-core parallel speedup: **6.5×** (ideal limit 8×)
- Kernel 2 (3x3x3 output tile, radius 3, 12 timesteps)
  - Single-core FPU utilization: **83%**
  - 8-core parallel speedup: **5.4×** (ideal limit 6.75×)

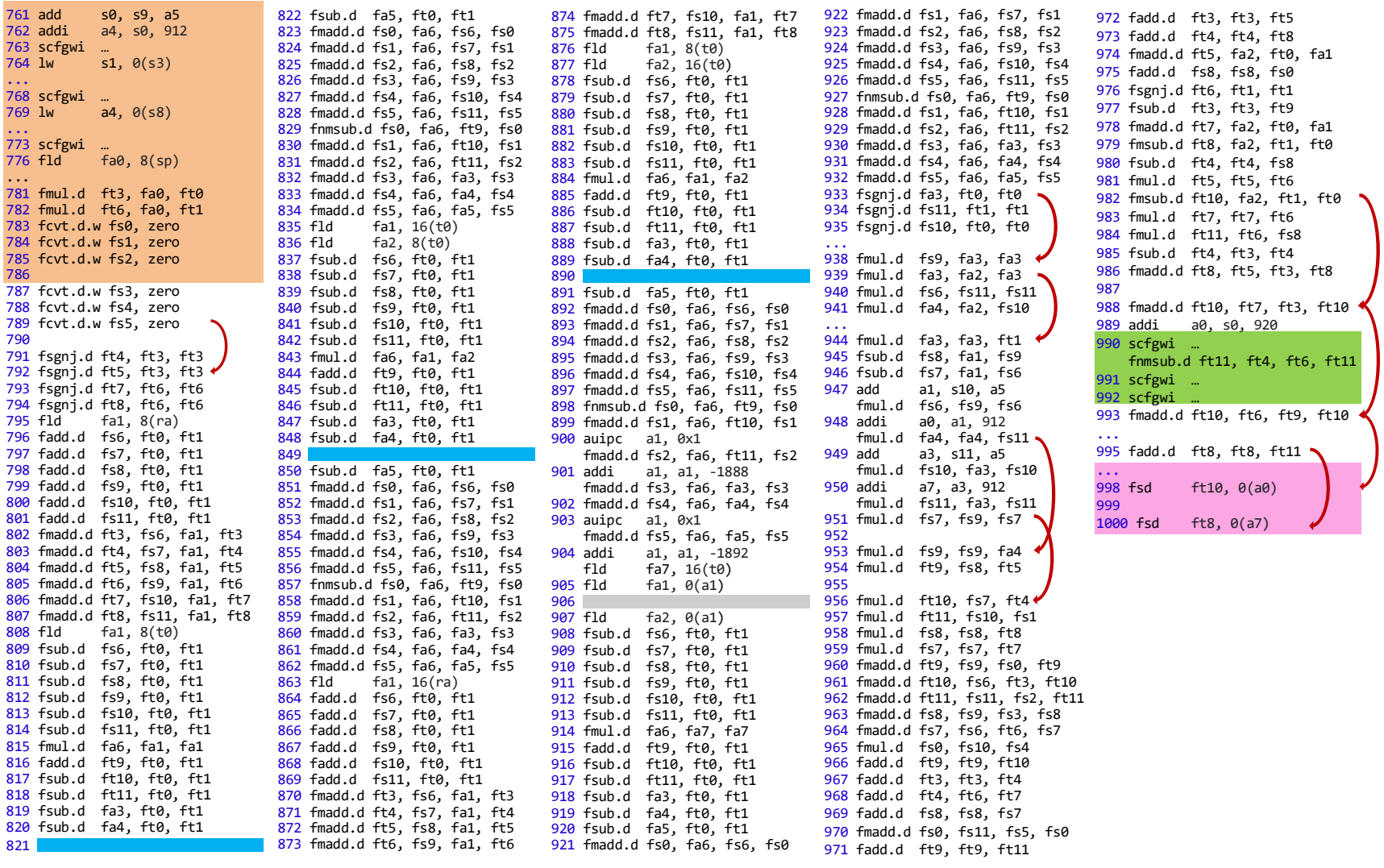




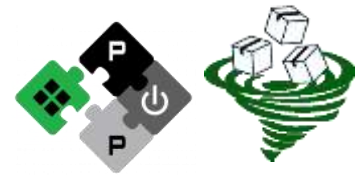


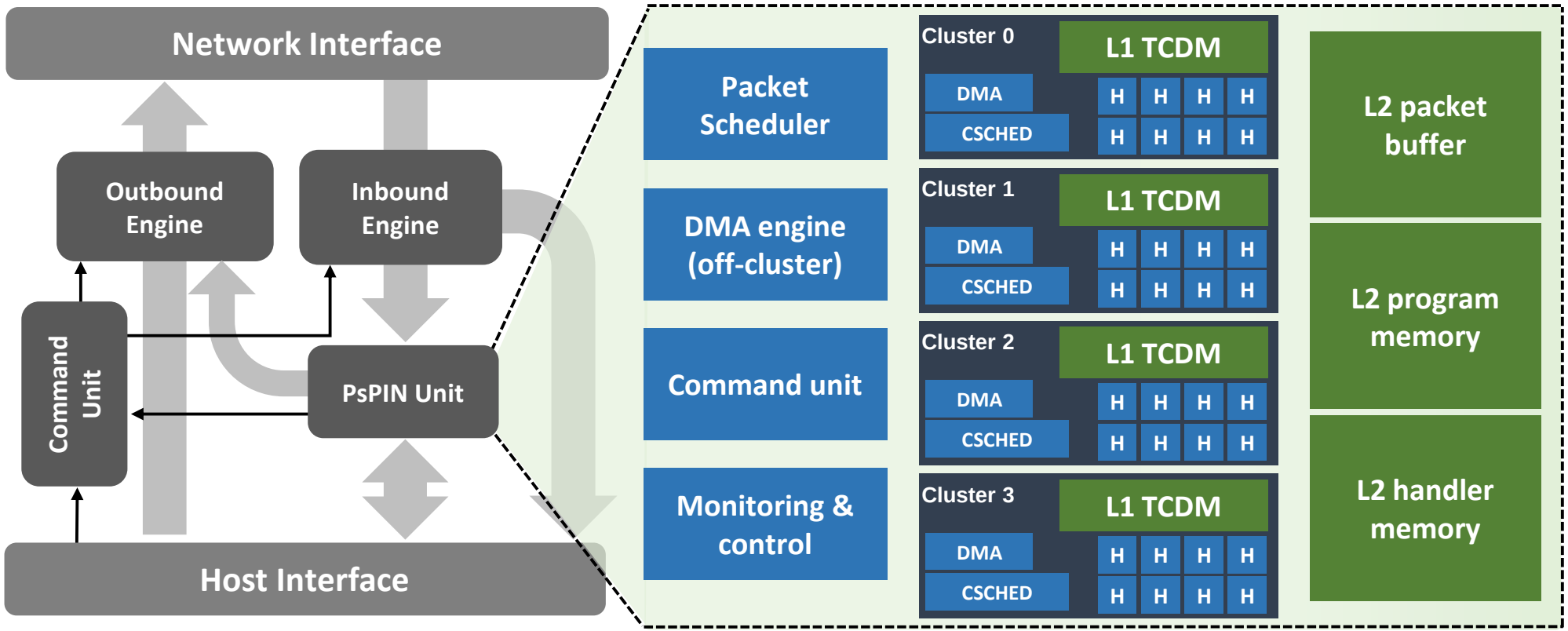
High Single-Core Efficiency

- Instructions almost entirely useful FP
  - Pseudo dual-issue hides SSR config
- Remaining stalls:
  - 1st point: cold I$, regs
  - RAW dependencies (out of registers)
  - In-order stores (out of SSRs)
  - ISSR index bottleneck (limited buffers)
  - PDI knock-on (not a real stall)



PsPIN: A PULP-powered implementation of sPIN





Stochastic Rounding

- Stochastic rounding to avoid stagnation when accumulating small quantities into a large accumulator
- Rounding decision taken comparing *p_rsr* bits against the output of an LFSR
- When using *p_rsr* = 12 bits, the area overhead amounts to ~4% on the ExSdotp unit, ~1% on the full extended FPU.