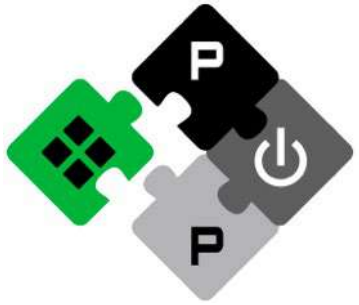


PULP PLATFORM

Open Source Hardware, the way it should be!

Paving the Road for RISC-V Supercomputers with Open Hardware

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

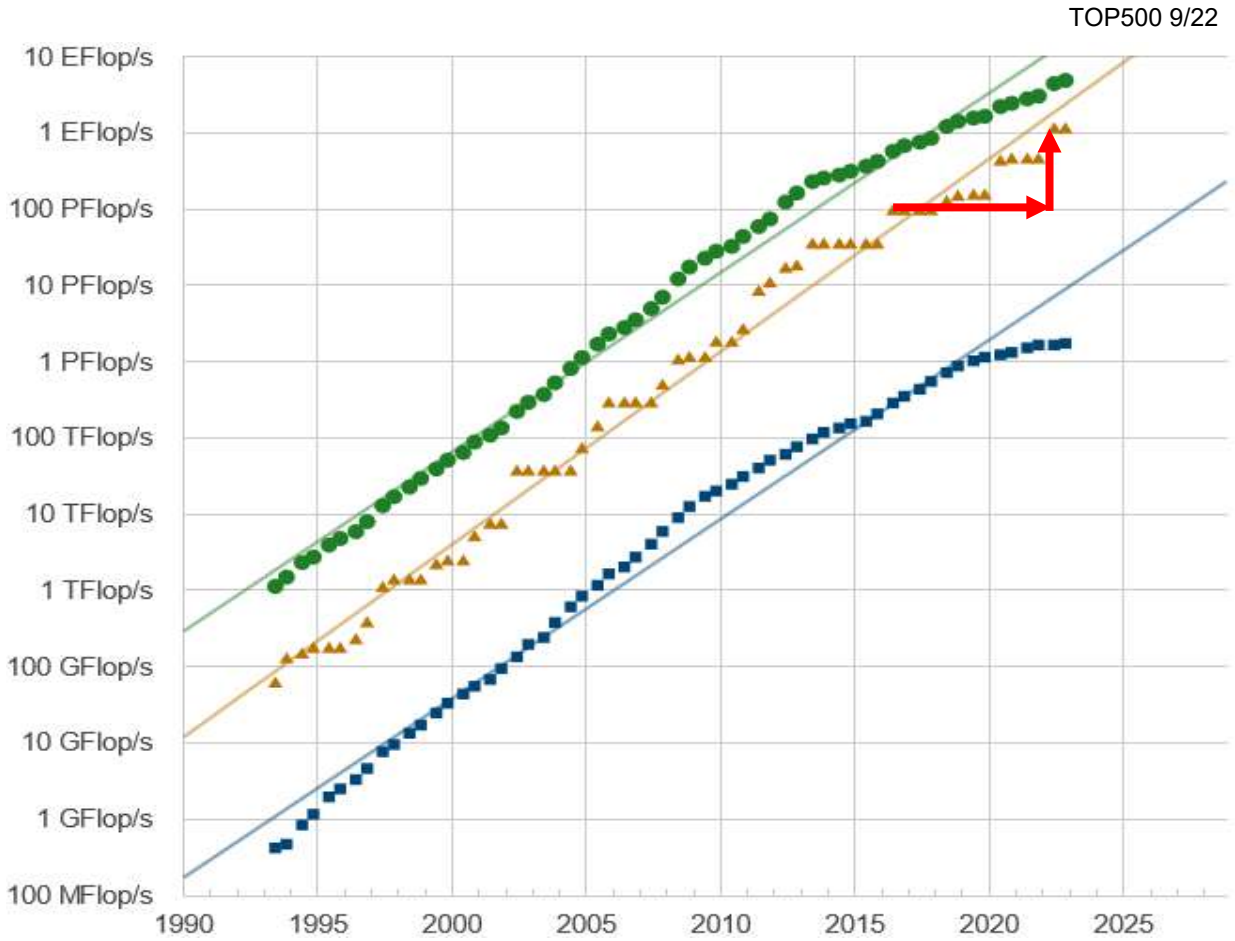
http://pulp-platform.org

@pulp_platform

https://www.youtube.com/pulp_platform

Computing is Power Bound: HPC

HPC Performance



20MW budget (fixed)

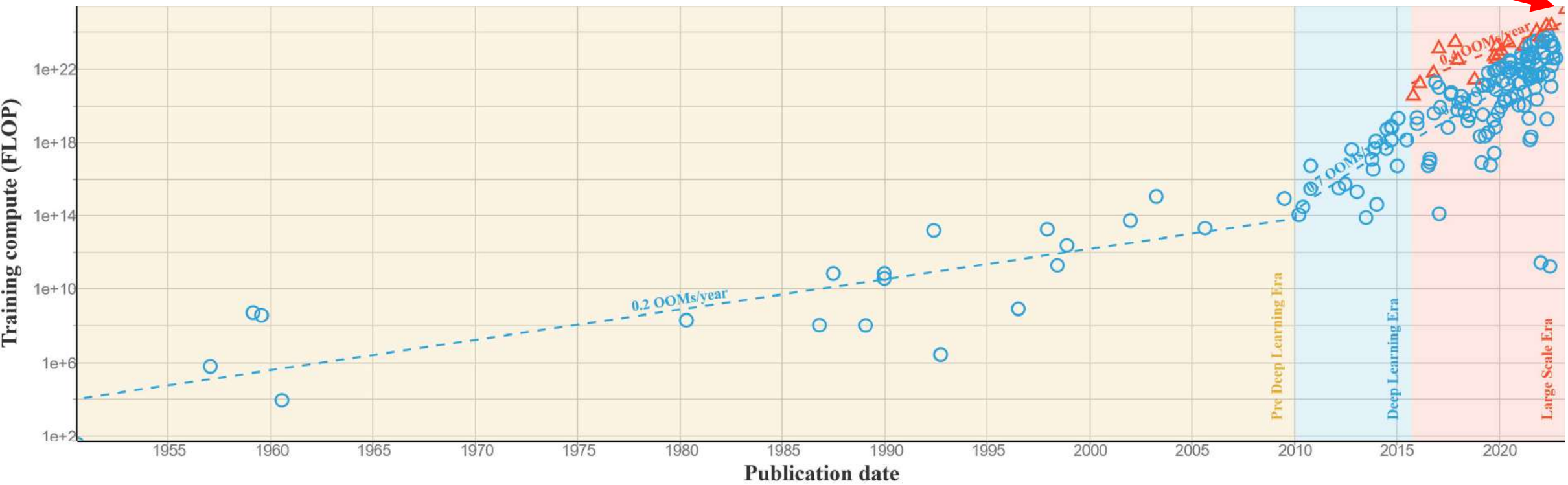
HPC: 10x every ~5 years

Computing is Power Bound: ML

Largest datacenter <150MW

GPT-4 (OpenAI'23)
Training Compute: 2.1E+25 (FLOP)

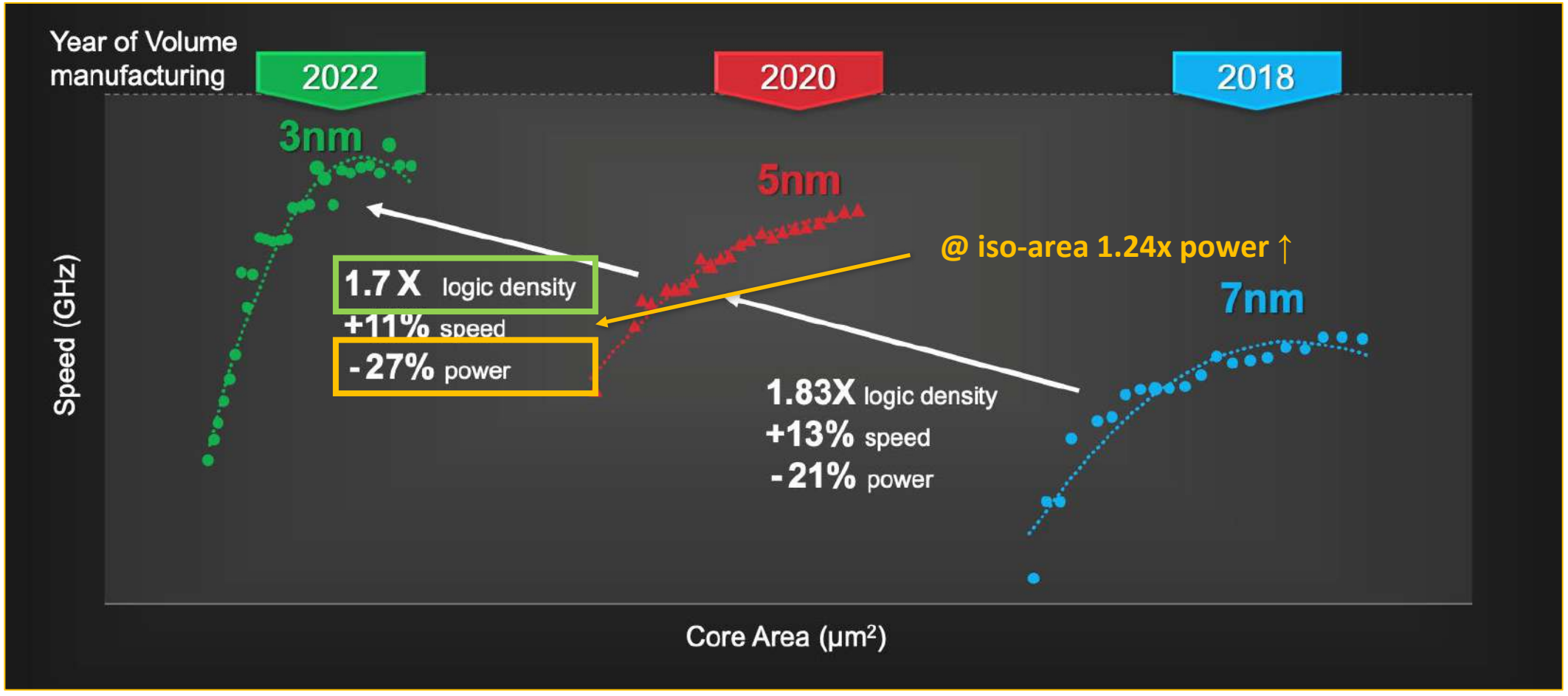
Sevilla 22: arXiv:2202.05924, epochai.org



Machine Learning (training): 10x every 2 years

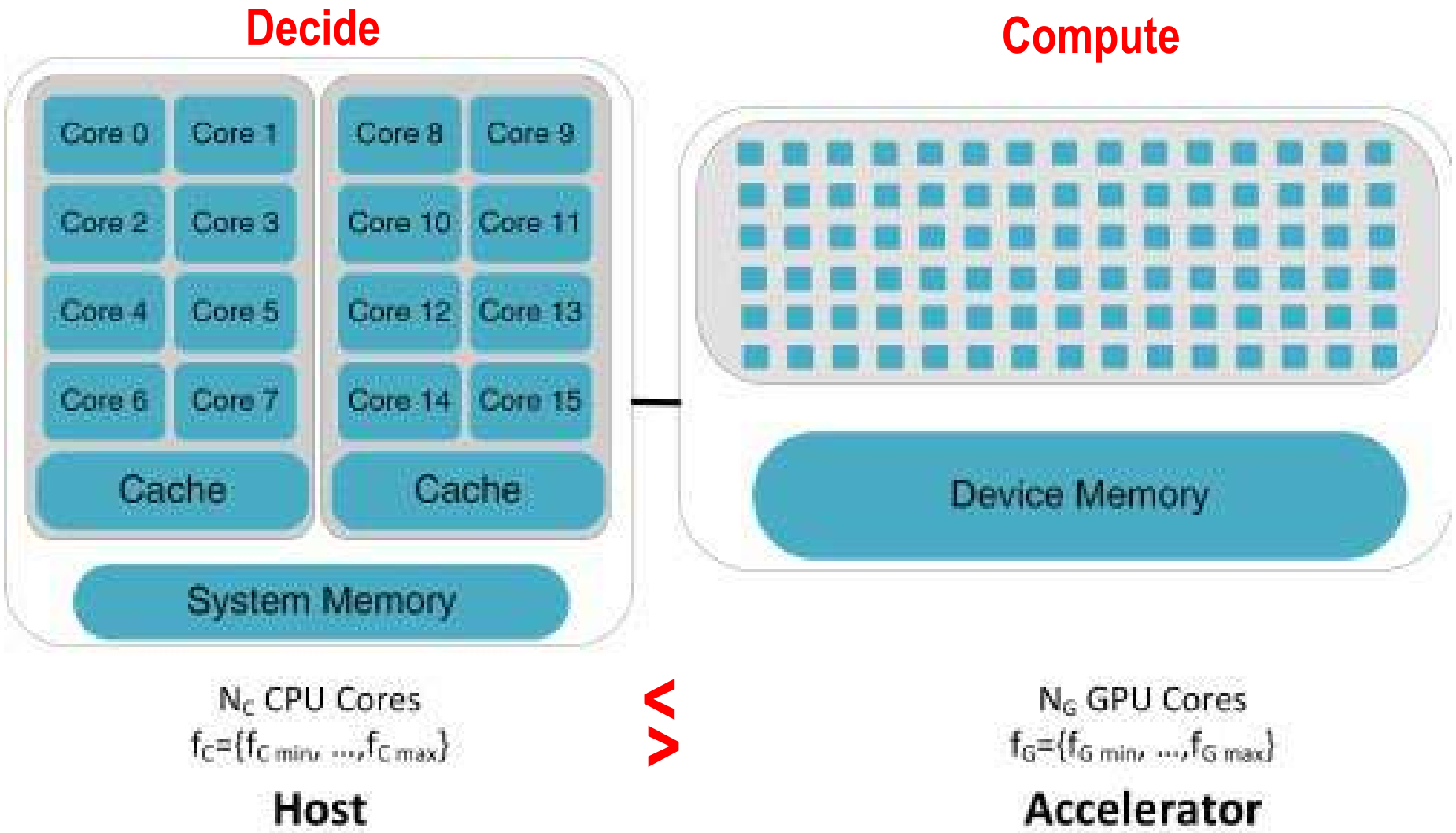
Technology Scaling?

TSMC, ISSCC21



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

Efficient Architecture: Heterogeneous+Parallel



Heterogeneous + Parallel… Why?

Decide

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

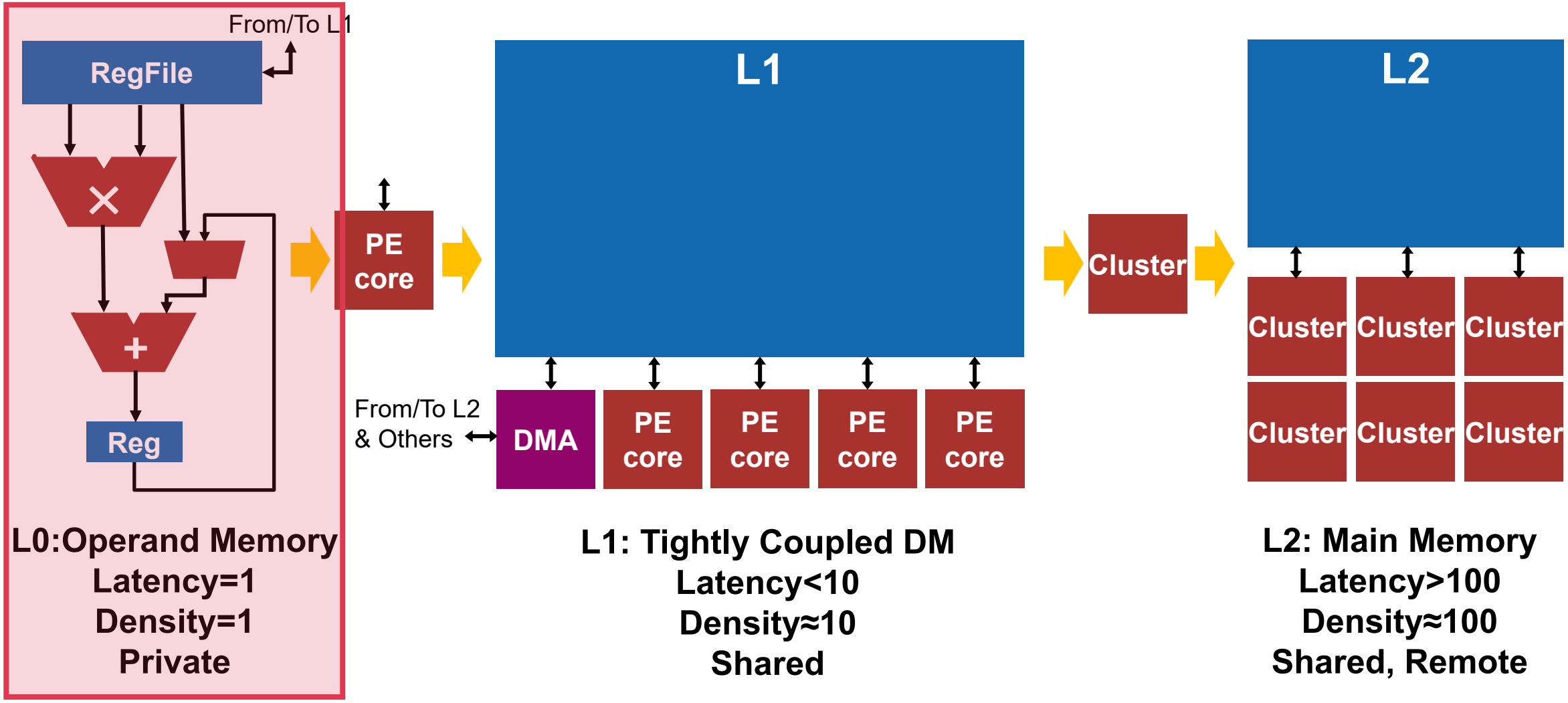
Compute (plough through numbers)

- 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**

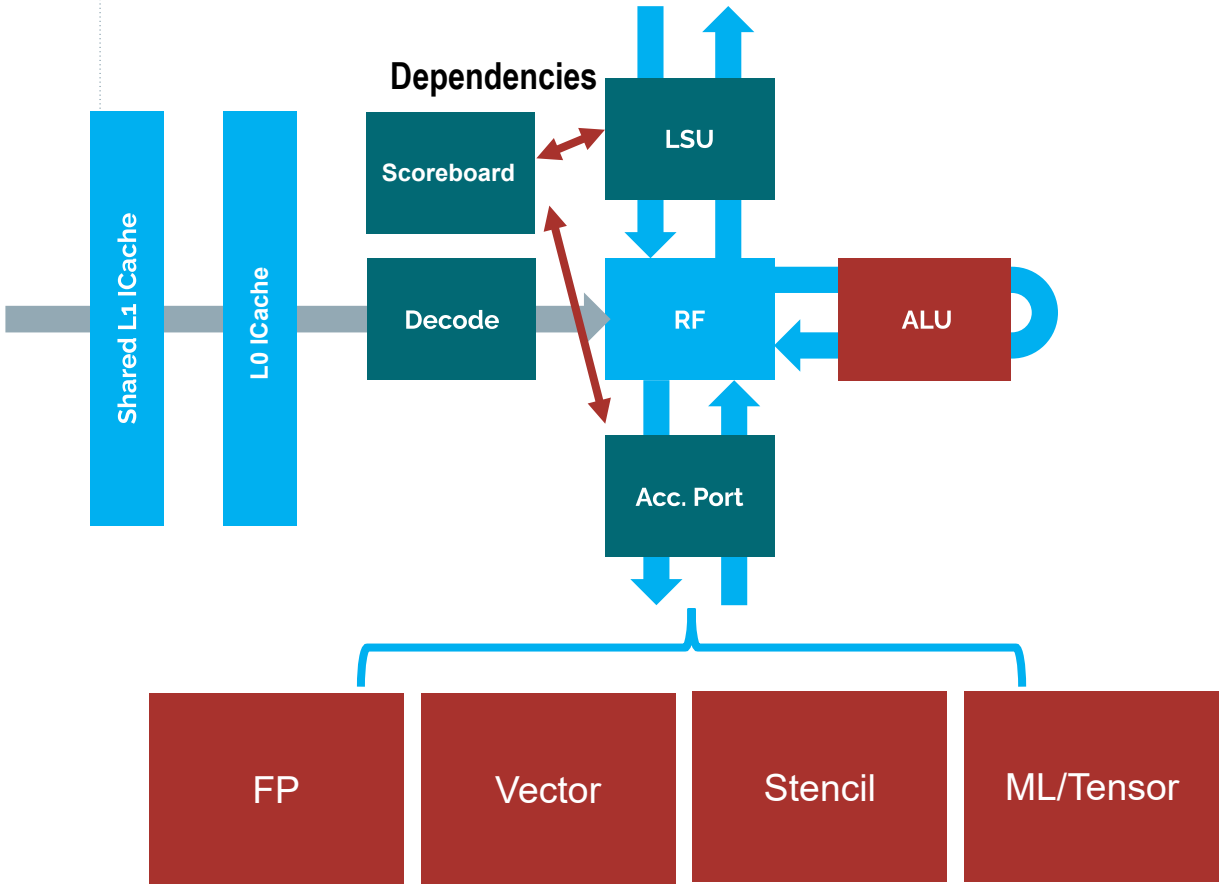
How to design an efficient "Compute" fabric?

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

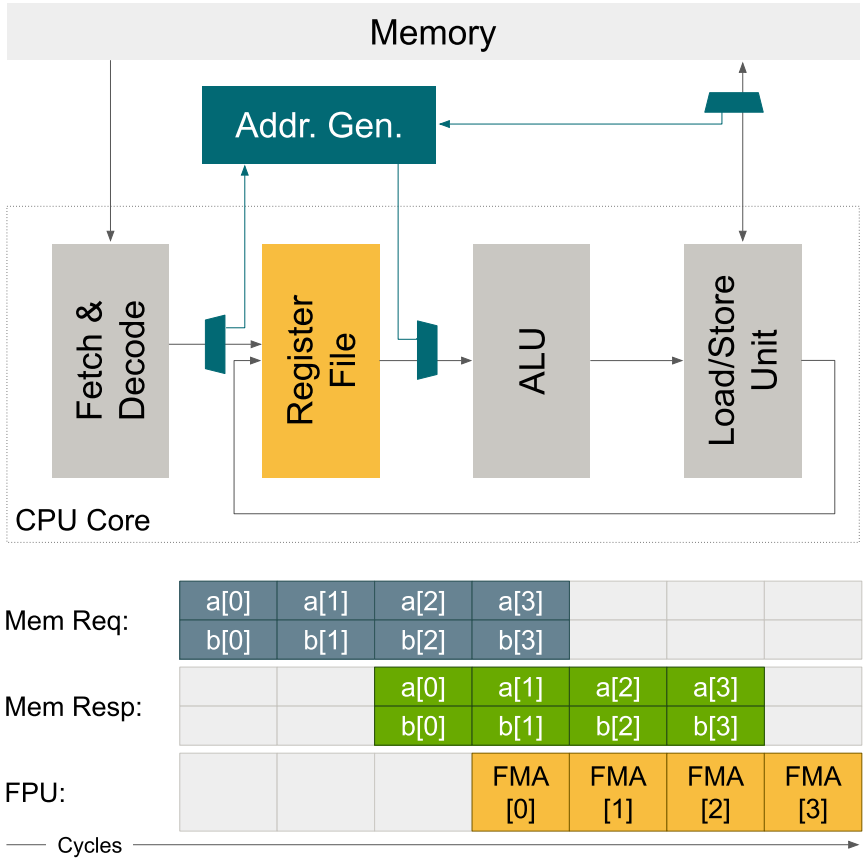
- Intuition: High FPU utilization ≈ high energy-efficiency
- Idea: Turn register read/writes into implicit memory loads/stores.
- Extension around the core's register file
- Address generation hardware

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

→

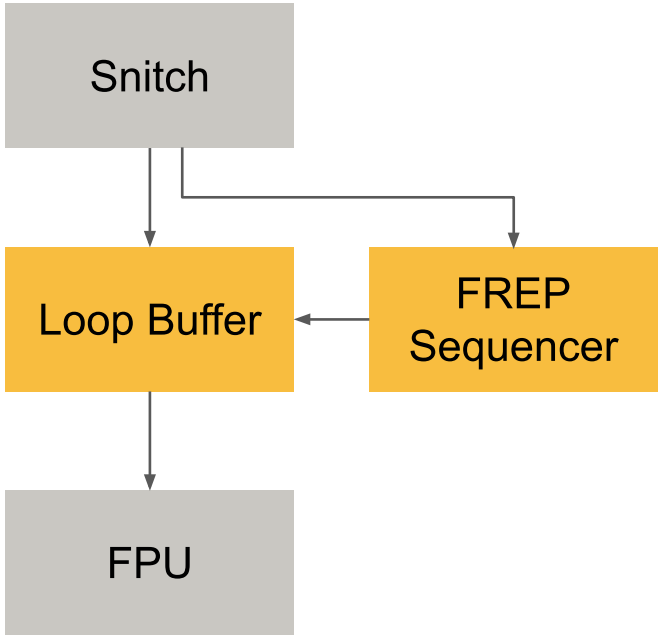
```
loop:
fld r0, %[a]
fld r1, %[b]
fmadd r2, r0, r1
```

- 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
```

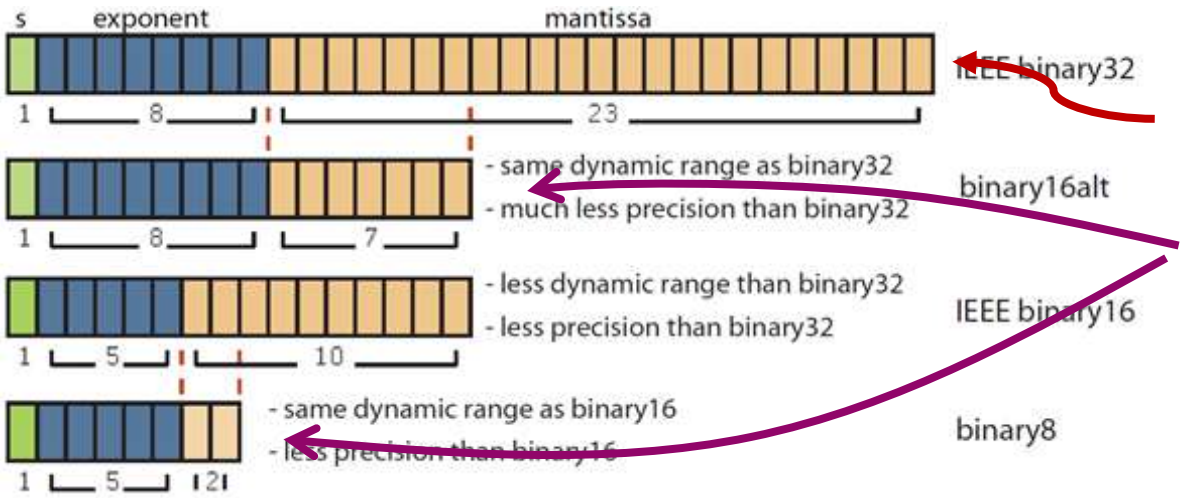
→

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

RISC-V ISA Extension for Target Workload

Mixed precision

Efficient DNN inference & training



Inference ≠ Training Quantization

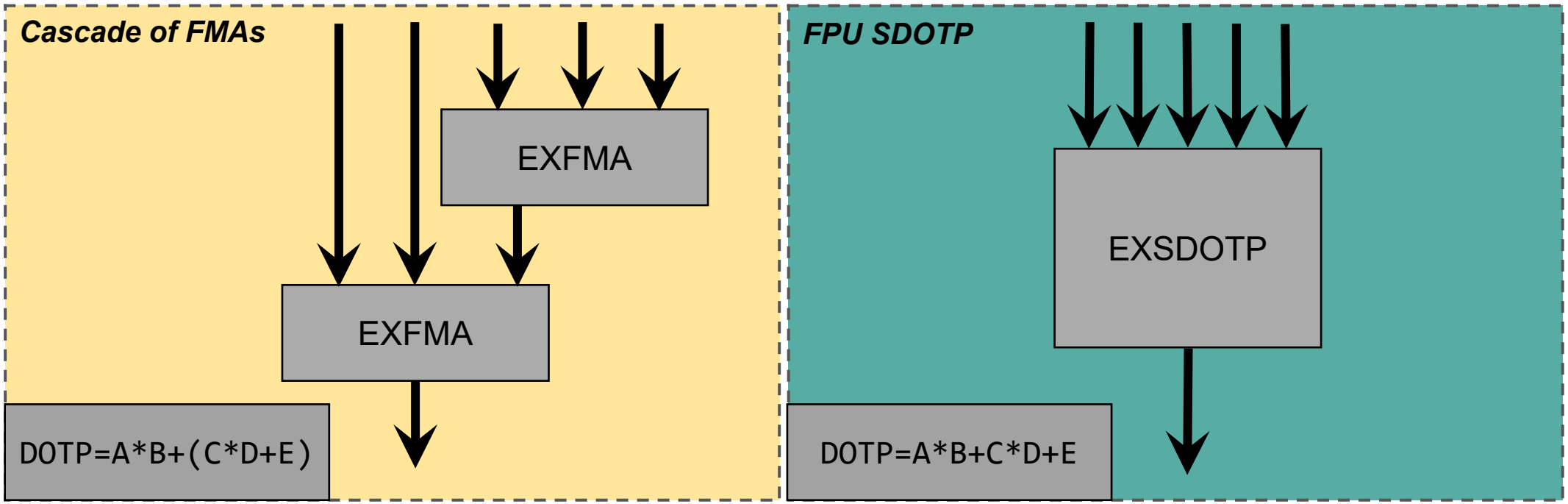
- Inference: INT8 quantization is SoA
- Training: High dynamic range needed for weights and weight updates

`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

Cascade of EXFMAs vs EXSDOTP

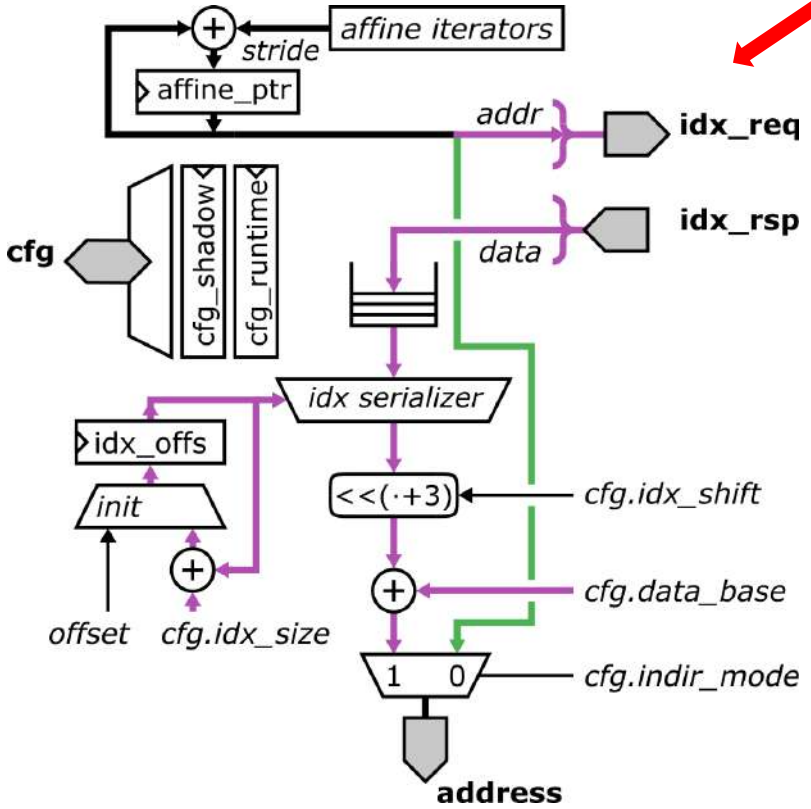


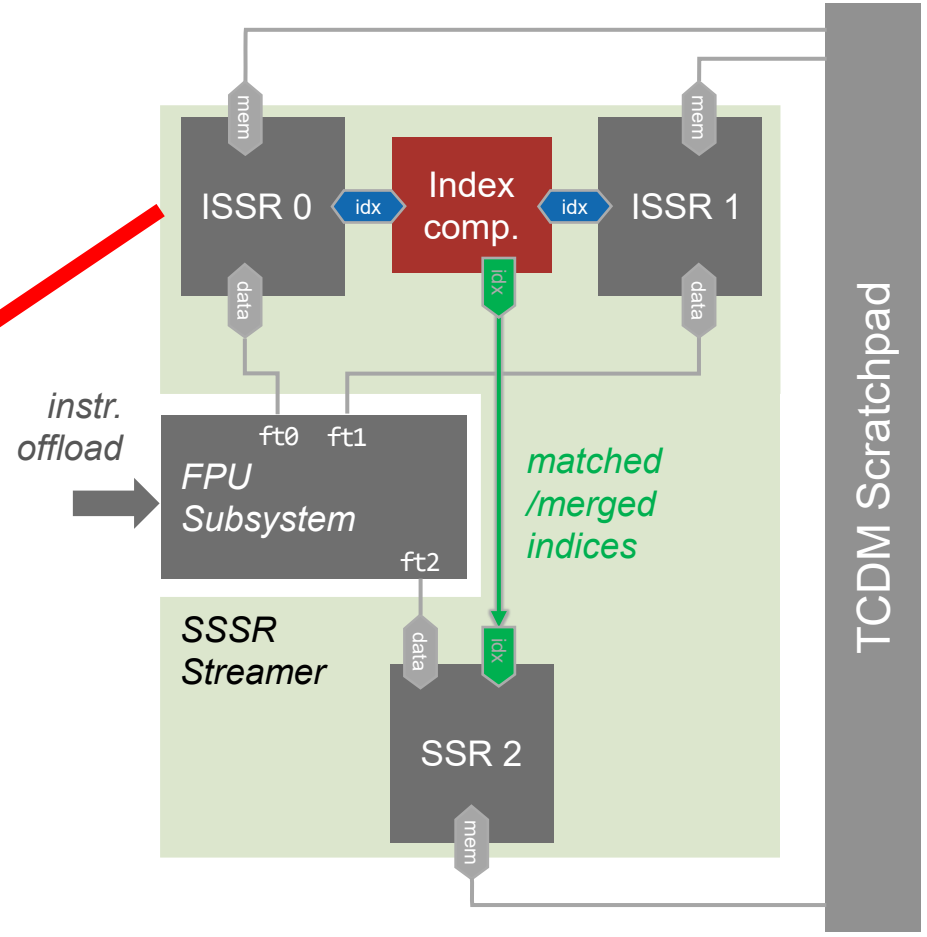
Non-distributive FP addition → **Precision Loss**

- **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? Indirect SSR Streamer

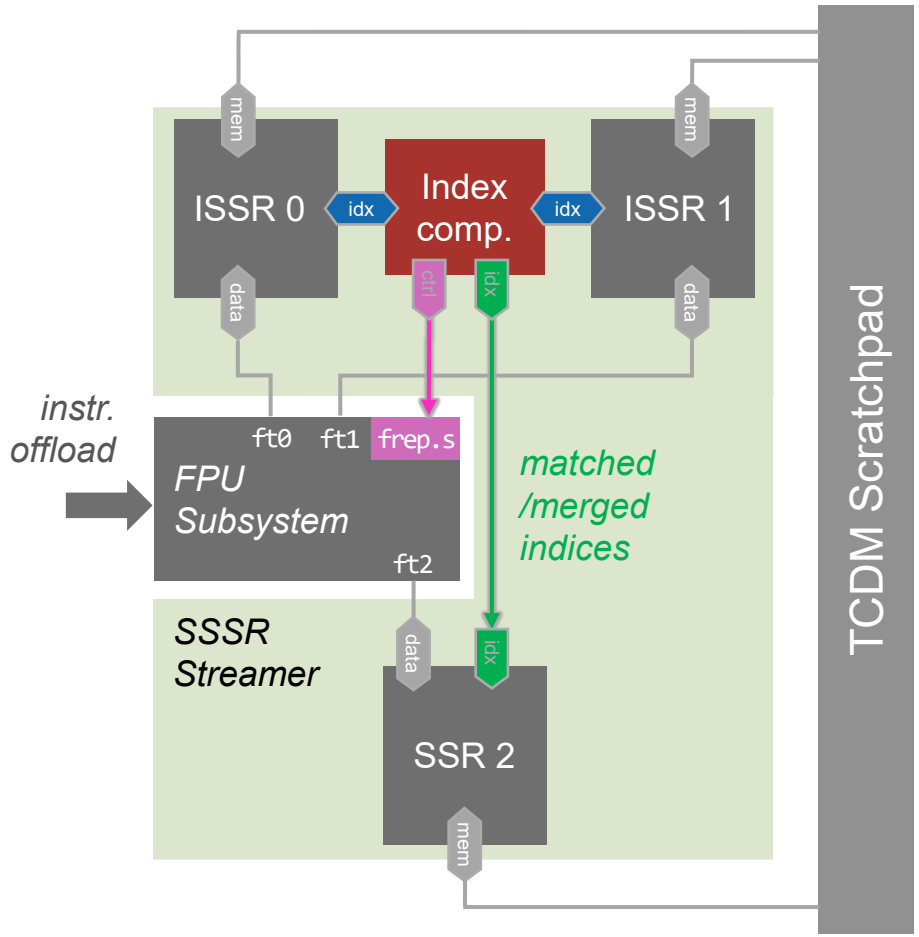
- **Based on existing 3-SSR streamer**
  1. Extend 2 SSRs to ISSRs
  2. Add *index comparison unit* between ISSRs
  3. Forward result indices to 3rd SSR





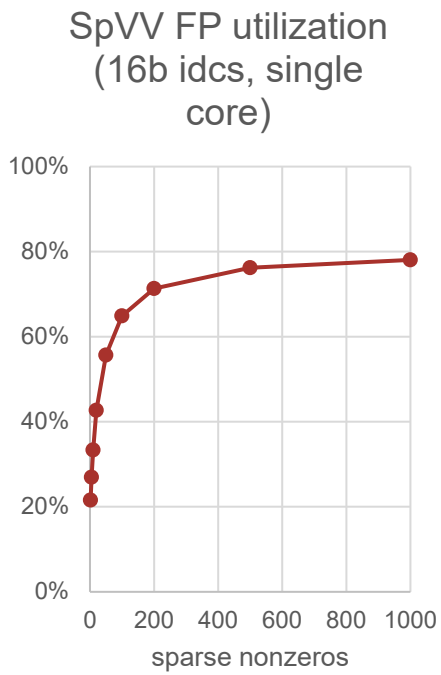
What About Sparsity? Indirect SSR Streamer

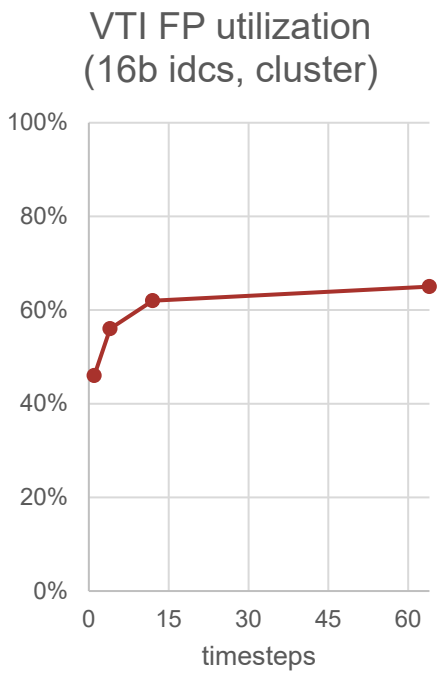
- **Based on existing 3-SSR streamer**
  1. Extend 2 SSRs to ISSRs
  2. Add *index comparison unit* between ISSRs
  3. Forward result indices to 3rd SSR
- **Control interface to FPU sequencer (`frep.s`)**
  - Result index count unknown ahead-of-time
- **Enables *general* sparse-sparse LA on fibers:**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



ISSR Performance Benefits

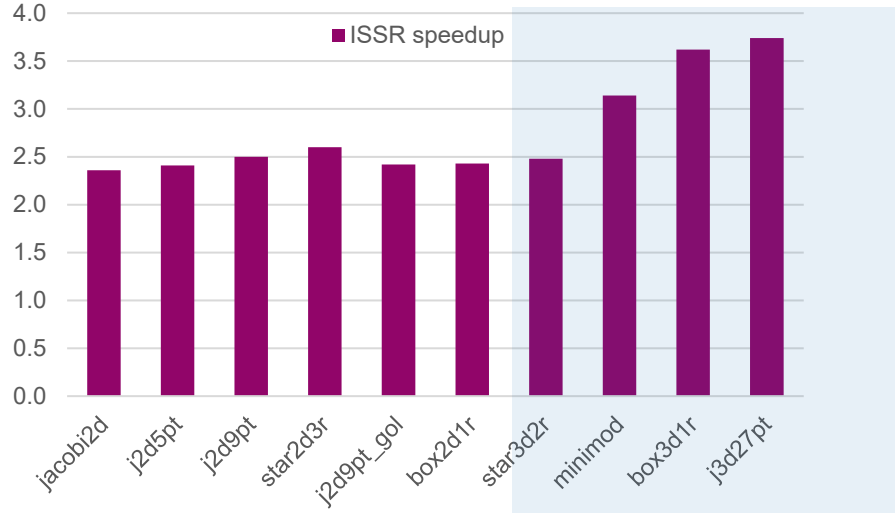
- **Notable single-core speedups over RV baseline**
  - *CsrMV*: up to **7.0×** faster, **79%** FP util.
  - *SpV+SpV*: up to **9.8×** faster / higher FP util.
  - *SpV·SpV*: up to **7.7×** faster / higher FP util.
  - *VTI (3D stencil code):* up to **2.9×** faster, **78%** FP util.
- **Significant benefits 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**

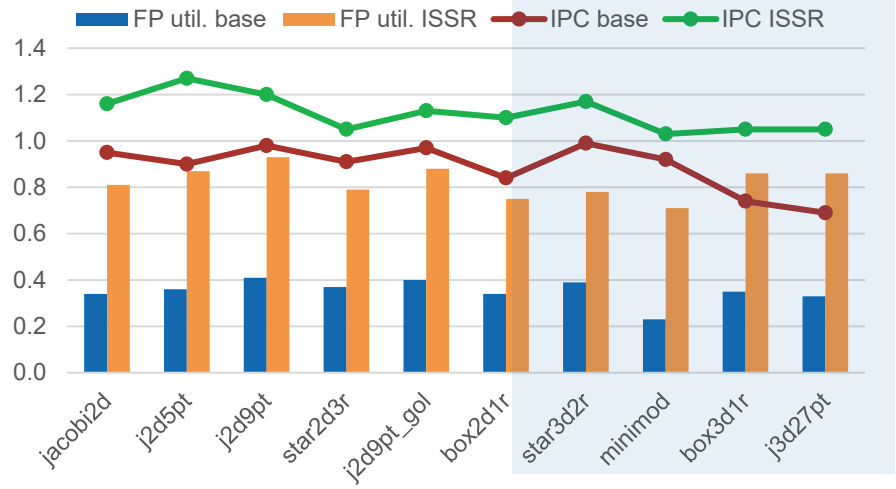




ISSR Performance on Stencils

- Various 2D/3D stencils on 8-worker-core cluster
  - FP64, $64^2/16^3$ grid chunks, up to 4× unroll
  - Tuned LLVM RV32G baseline vs ISSR-enhanced kernels
- Geomean **2.7×** speedups, **82%** FP utilization
  - ISSR **IPC consistently >1** as ISSRs enable pseudo-dual-issue
- Baseline perf. *degrades* for large (3D) stencils
  - Cannot maintain unroll *and* keep reusable inner-loop data in register file
  - ISSR streams **avoid this bottleneck**: 2.5× 2D → 3.2× 3D geomean speedup

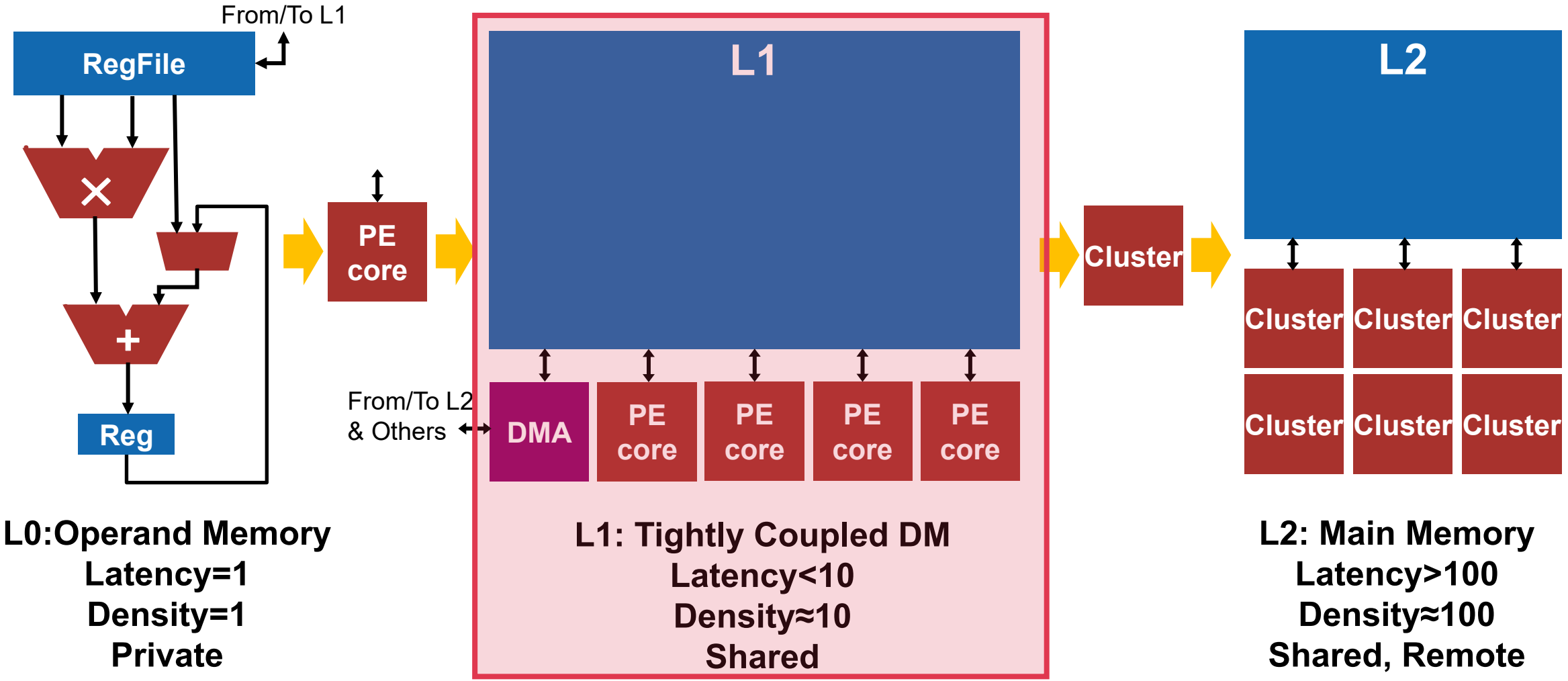




Efficient PE (snitch) architecture in perspective

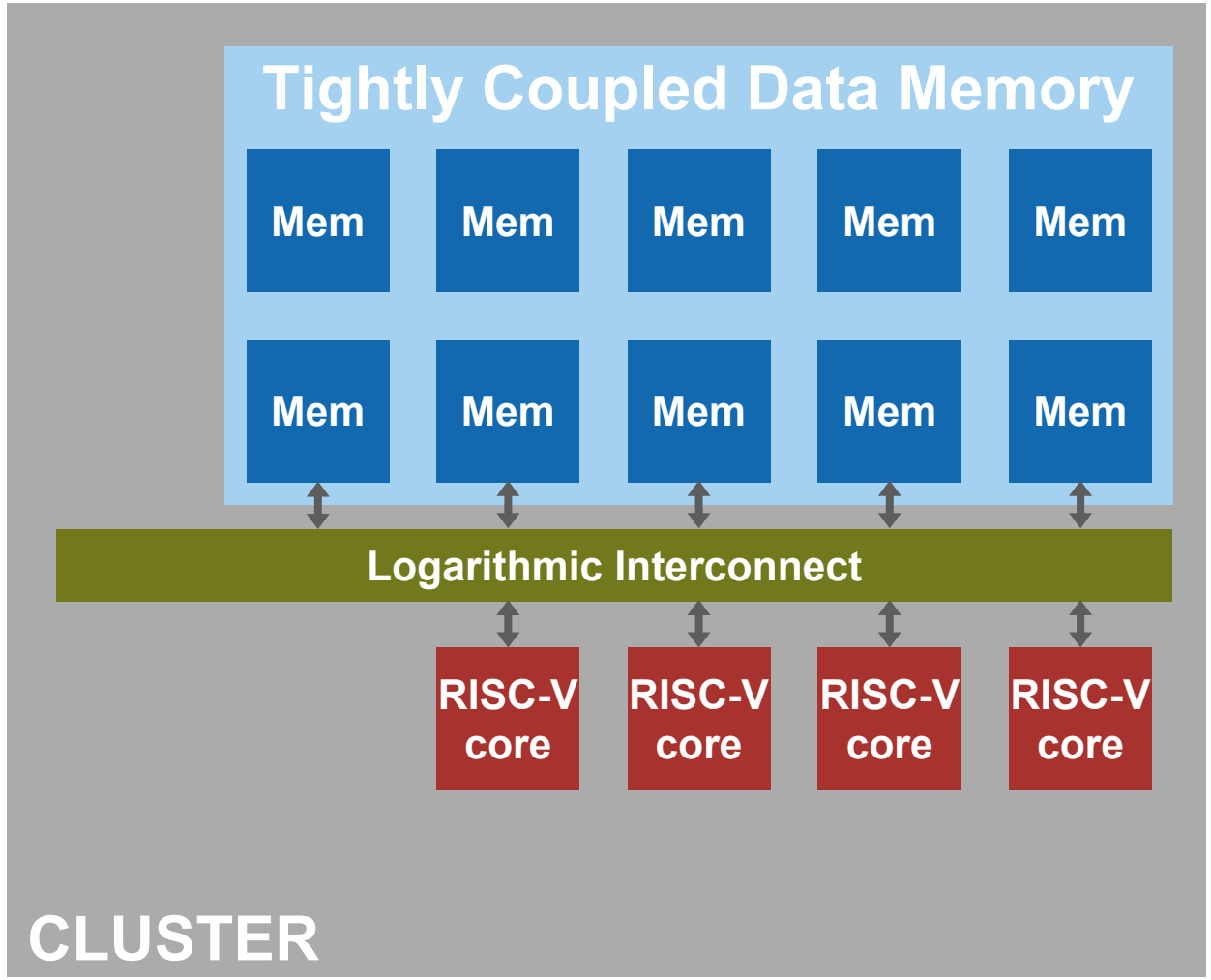
1. Minimize control overhead → Simple, shallow pipelines
2. Reduce VNB → amortize IF: SSR-FREP + SIMD (Vector processing)
3. Hide memory latency → non-blocking (indexed) LD/ST+dependency tracking
4. Highly expressive, domain-specific instruction extensions (thanks, RISC-V!)

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



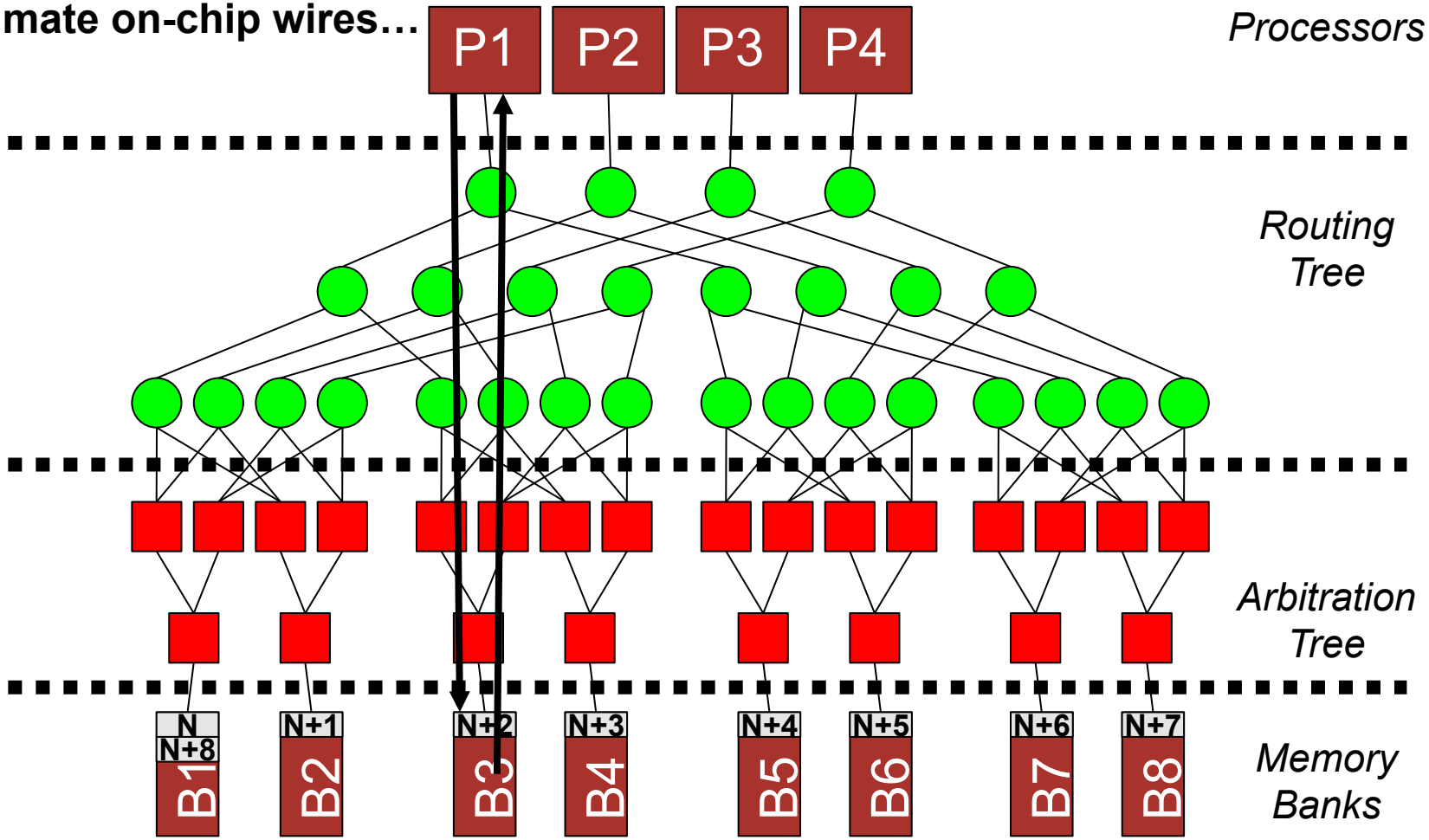
The Cluster: Design Challenges

- **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

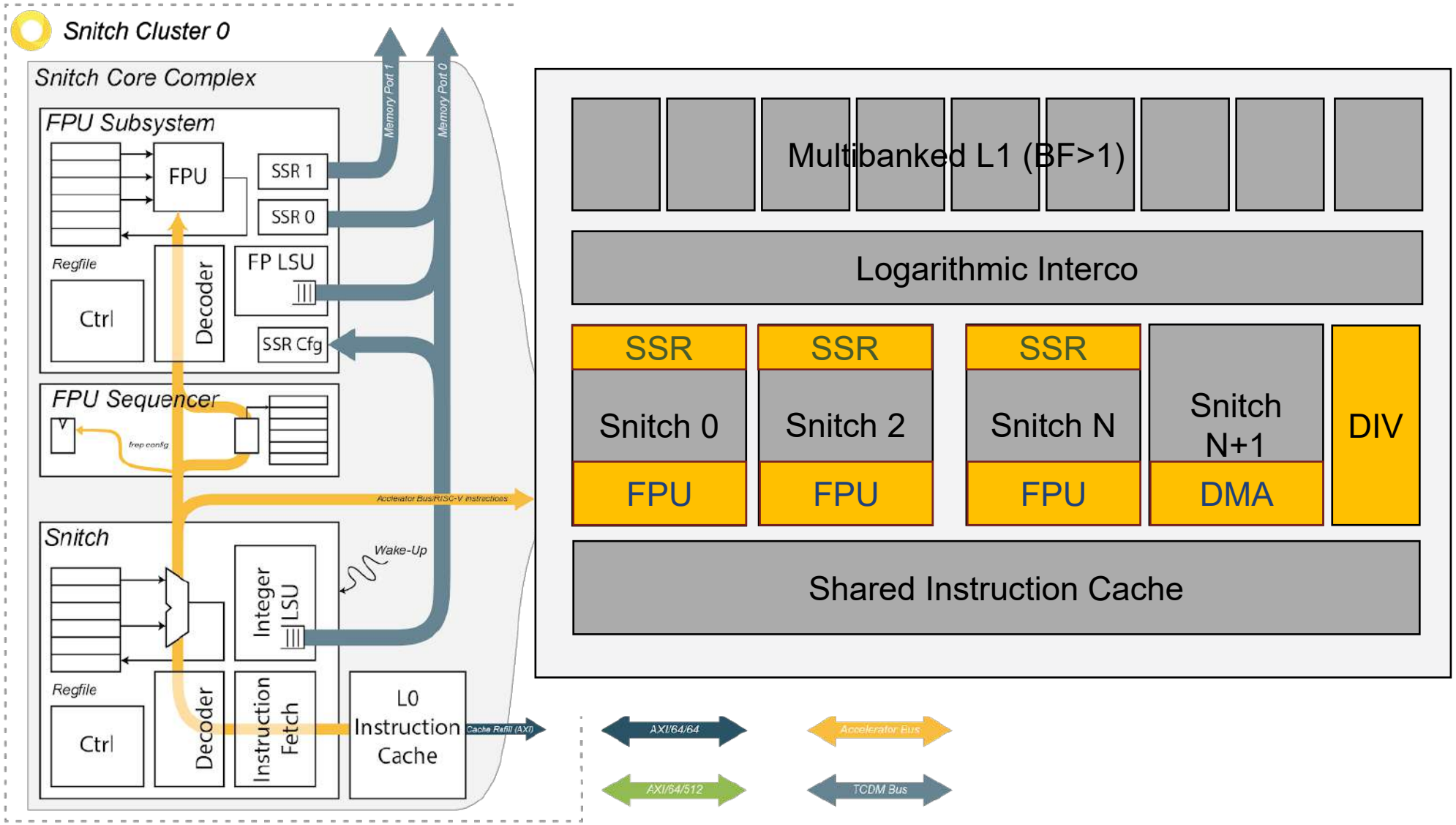
Efficient *Explicit* Global Data Mover

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?

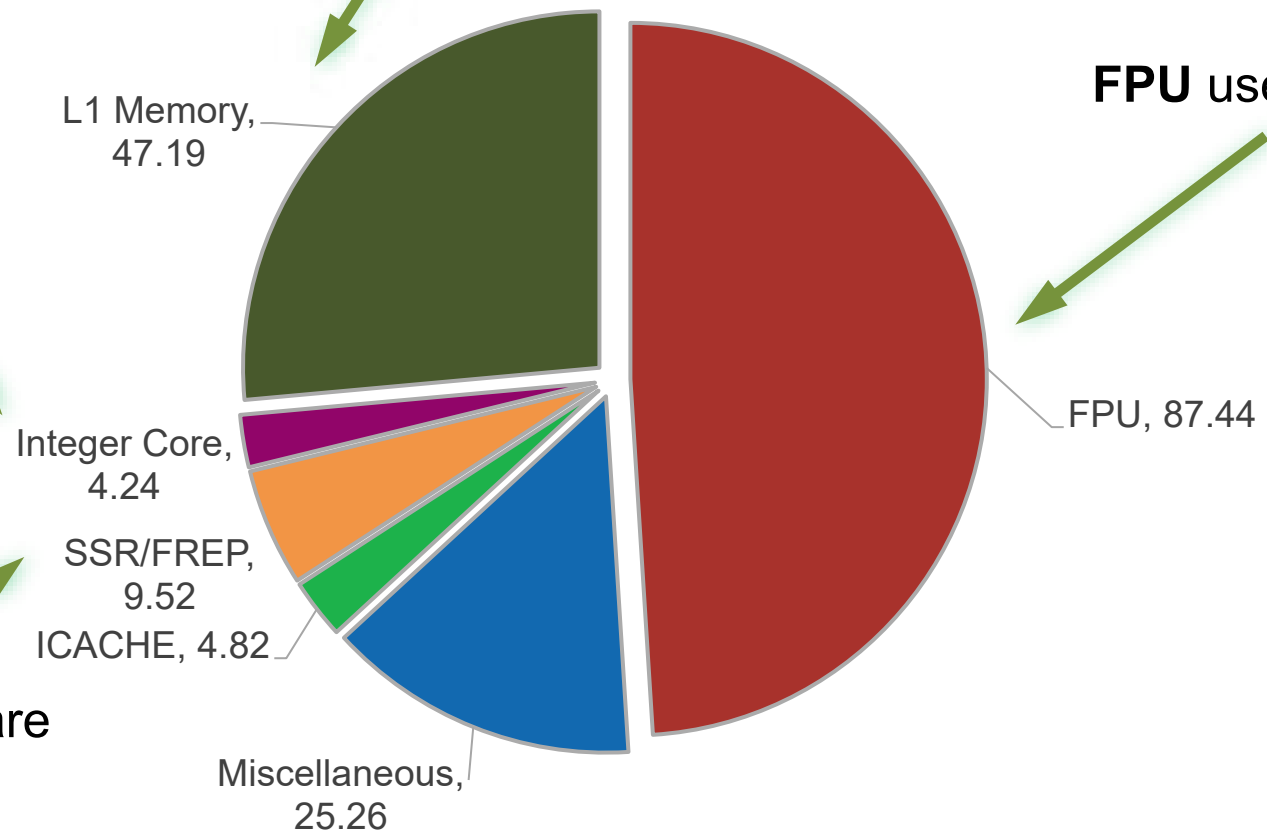
In an 8-core cluster

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



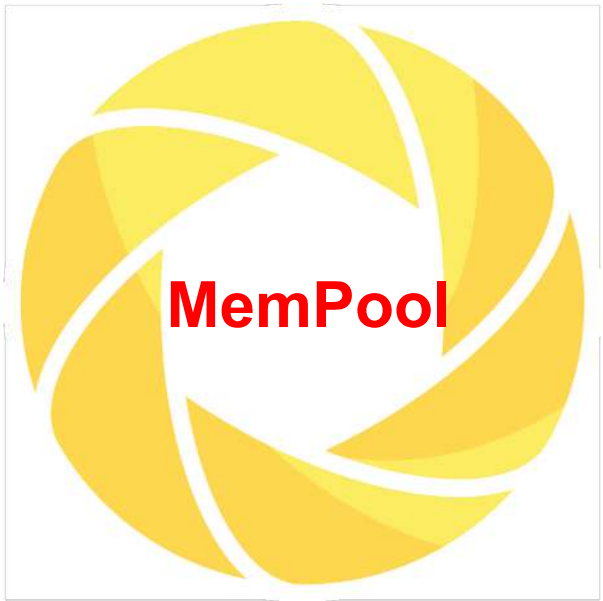
Spending energy where it contributes to the result → **High Efficiency**

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!

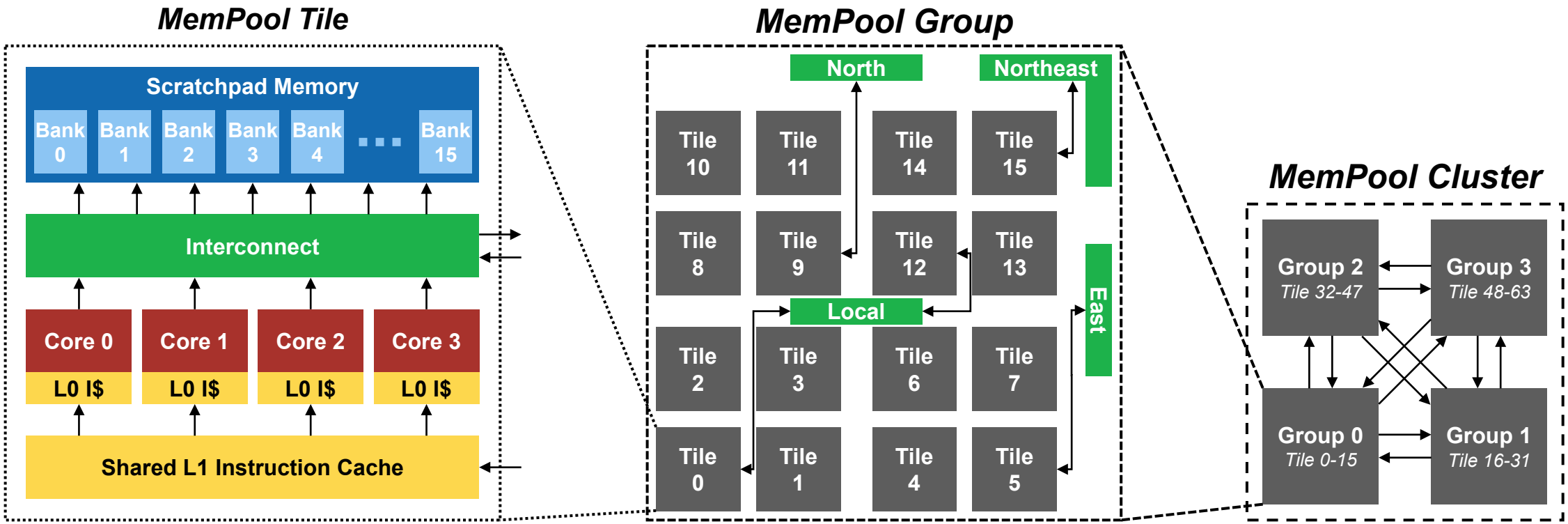
Back to the cluster… Can we make it Bigger?

- **Why?**
  - Better global latency tolerance if $L1_{size} > 2*L2_{latency}*L2_{bandwidth}$ (Little's law + double buffer)
  - 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
  - ≤ 10 cycles L1 latency (without contention)
- **Physical-aware design**
  - WC Frequency > 500 Mhz
  - Targeting iso-frequency with small cluster



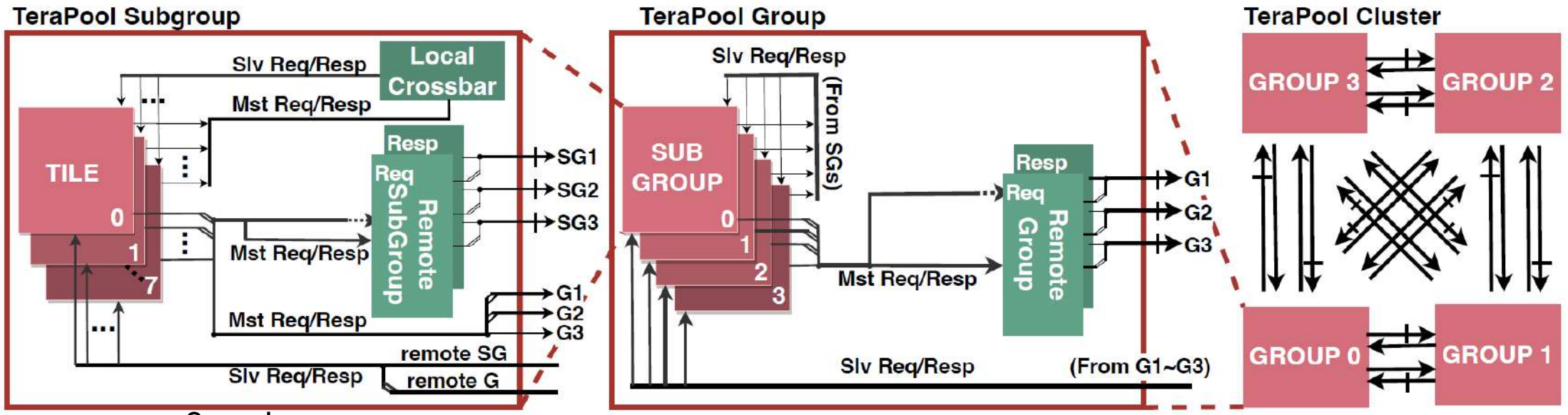
Hierarchical Physical Architecture

- **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

Can we push it further? Mempool → Terapool



GF12 0.8V 16 Snitch/Tile, Multi-stage Interconnect 0.23 req/core/cycle

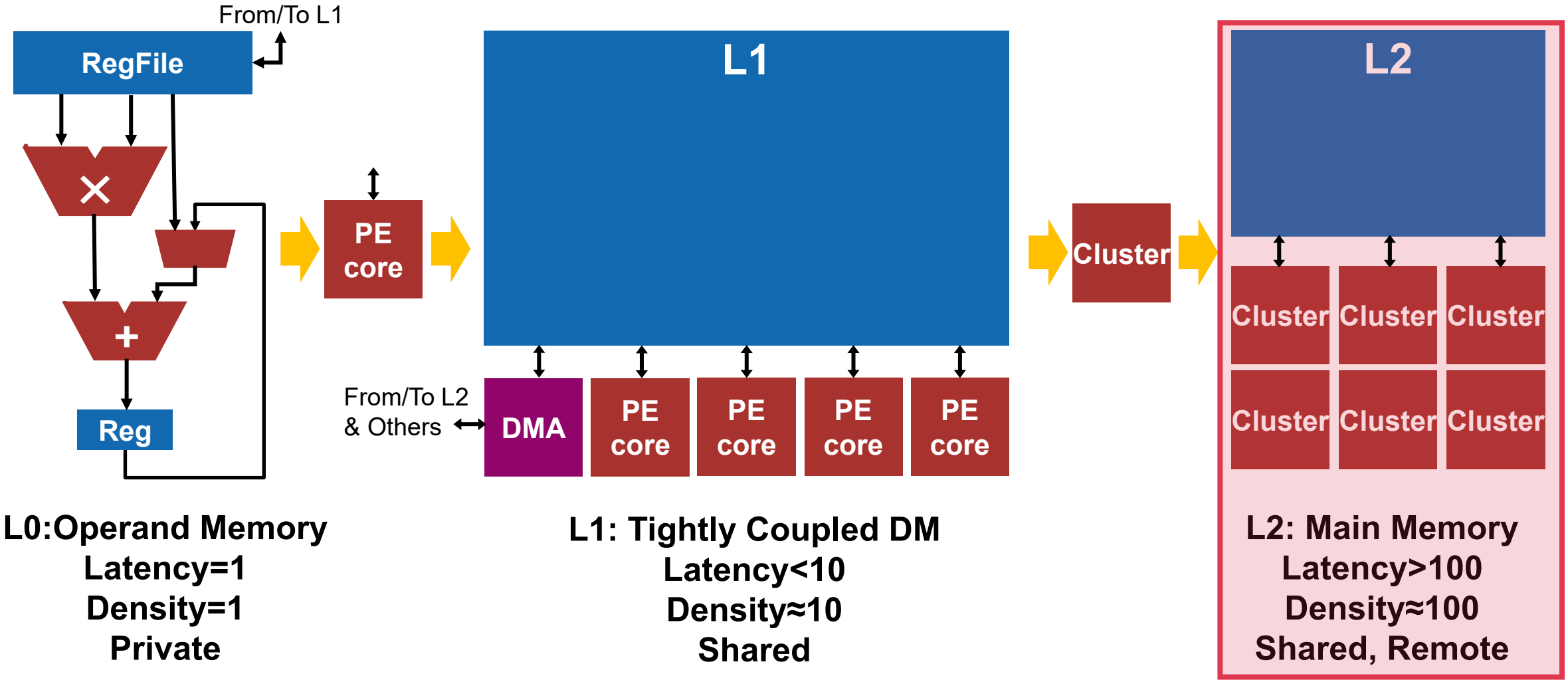
1024 Cores 4MB, 4096Banks!

69mm2, 3.8W, 900MHz 0.6TOPS (MMUL) → @5nm: 23mm^2, 2.2W, 1.2GHz, 1TOPS

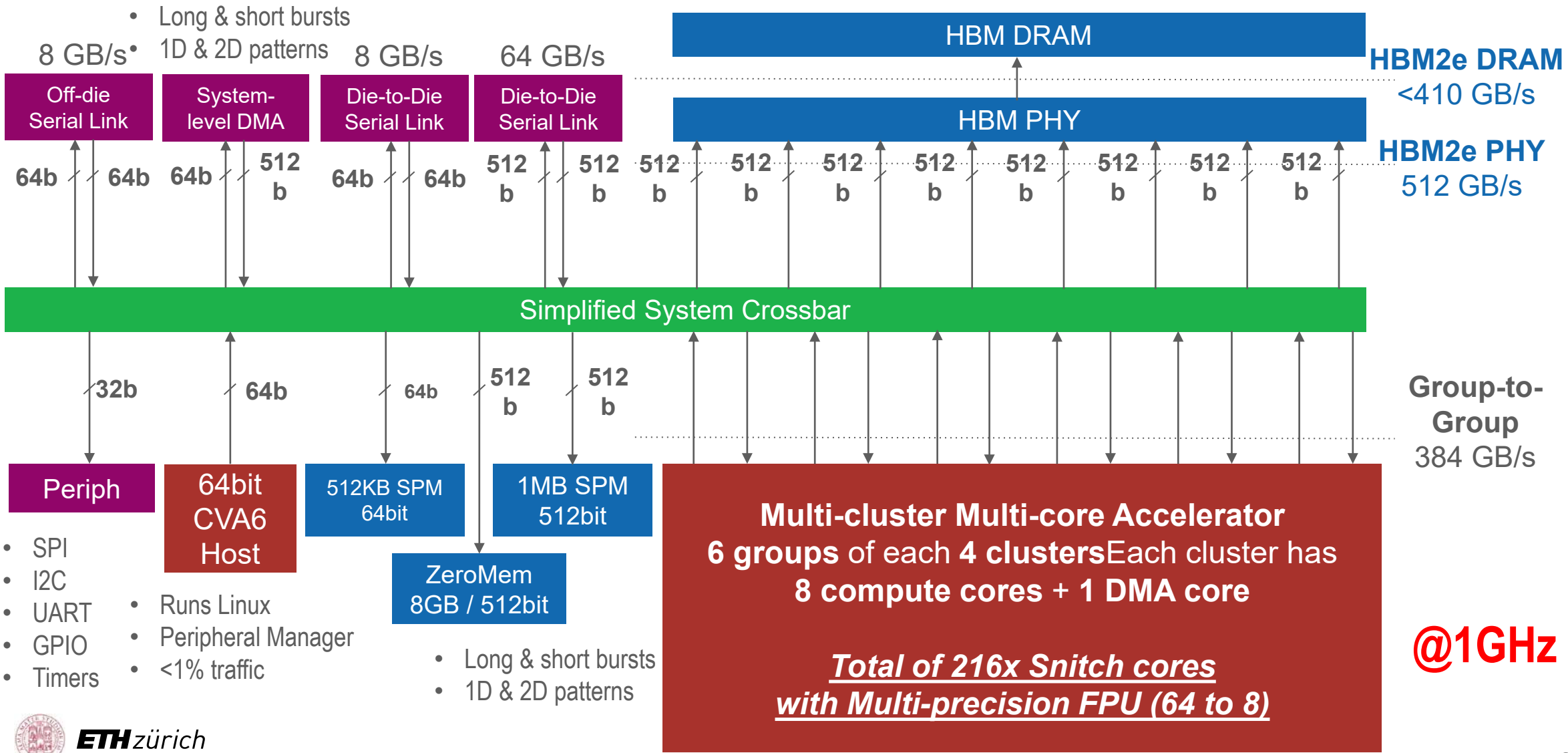
4MB can hide a latency of 500ns for a BW of 4TBps

… need more? → **Terapool-3D**

Compute Efficiency: the Chip(let) (Clusters+Off-die Mem)

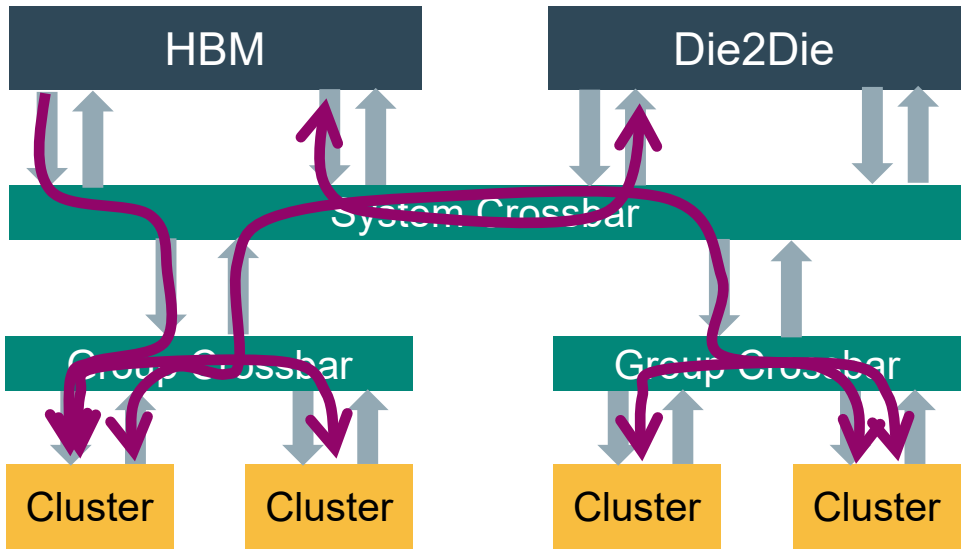


Occamy: RISC-V goes HPC Chiplet!



ETH zürich

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**: Tiling, Depth-First
- **Chiplets**: E.g. Layer pipelining

Big trend!

High-Performance, General-Purpose

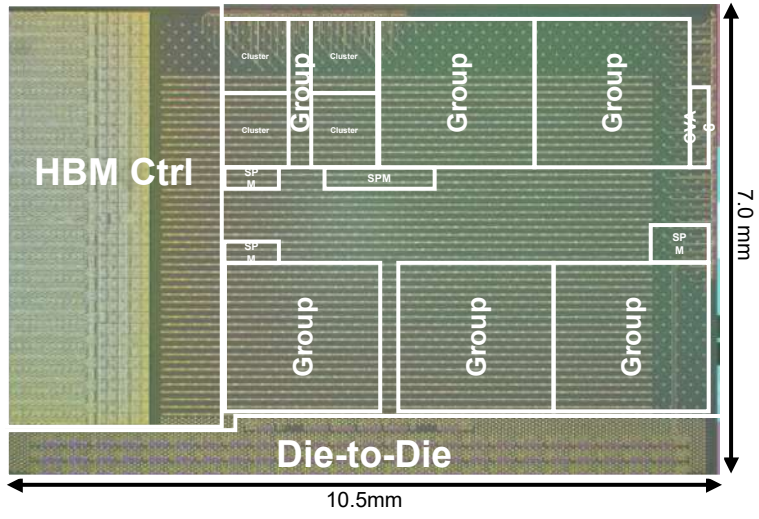
*Our **scalable architecture** is **general-purpose** and **high-performance***

Peak chiplet performance @1GHz:

- FP64: 384 GFLOp/s
- FP32: 768 GFLOp/s
- FP16: 1.536 TFLOp/s
- FP8: 3.072 TFLOp/s

Preliminary measured results:

- Dense Kernels:
  - GEMMS: ≥ 80% FPU utilization (also for SIMD MiniFloat)
  - Conv2d: ≥ 75% PFU utilization (also for SIMD MiniFloat)
- Stencils Kernels: ≤ **60%** FPU utilization
- Sparse Kernels: ≤ **50%** FPU utilization

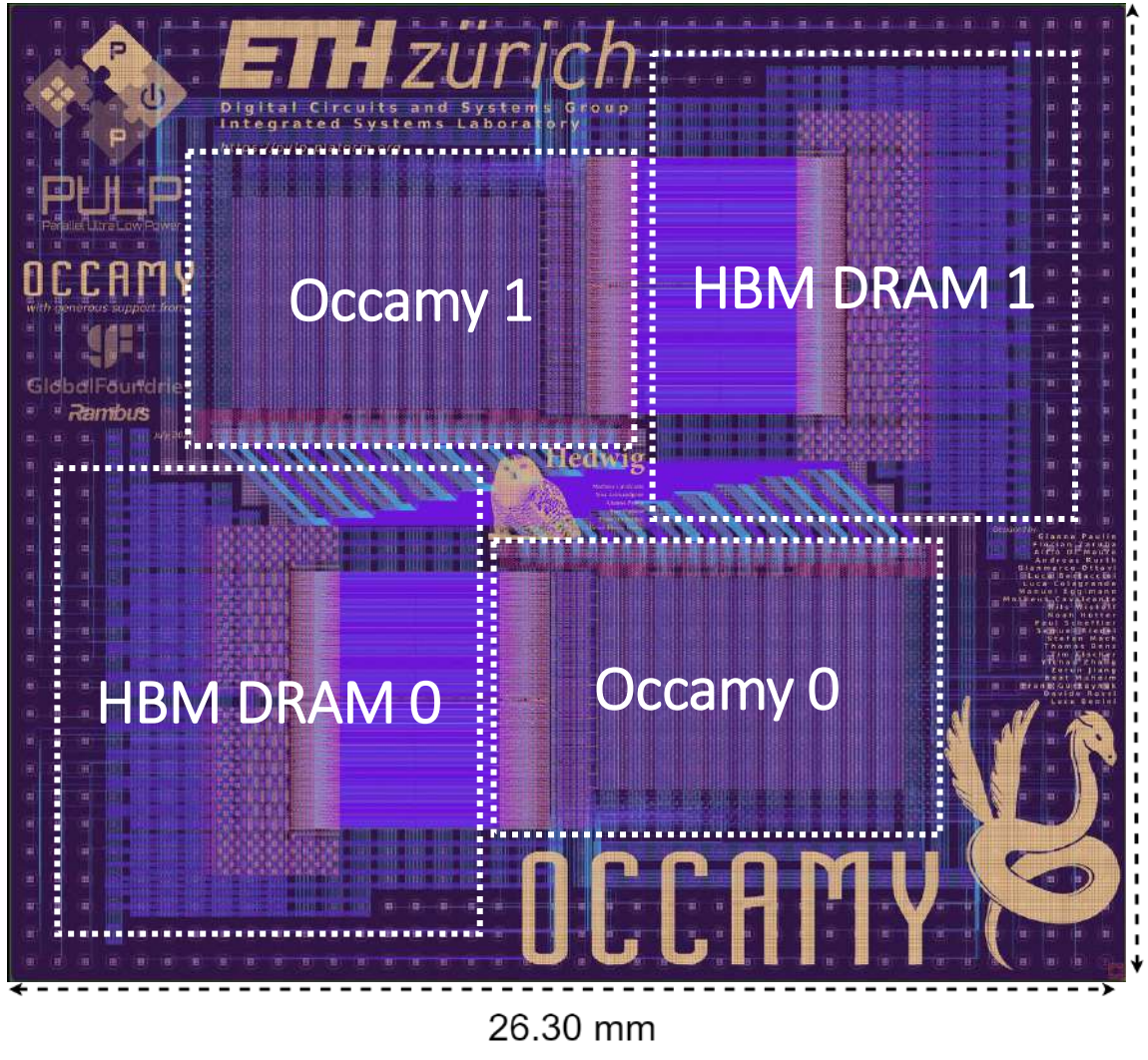


Chiplet taped out: 1st July 22

Silicon Interposer: Hedwig (65nm, passive, GF)

Taped out: 15th of October 2022

- **Interlocked** die arrangement
  - Prevent bending, increase stability
- **Compact** die arrangement
  - No *dummy dies* or *stitching* needed
- **Fairly low I/O pin count** due to no high-bandwidth periphery
  - Off-package connectivity: ~200 wires
  - Array of **40 x 35 (-1) C4s** (total of 1'399 C4 bumps)
    - Diameter: 400µm, Pitch: 650µm
- **Die-to-Die: ~600 wires**
- **HBM: ~1700 wires**



Approaching 1T(DP)-FLOP

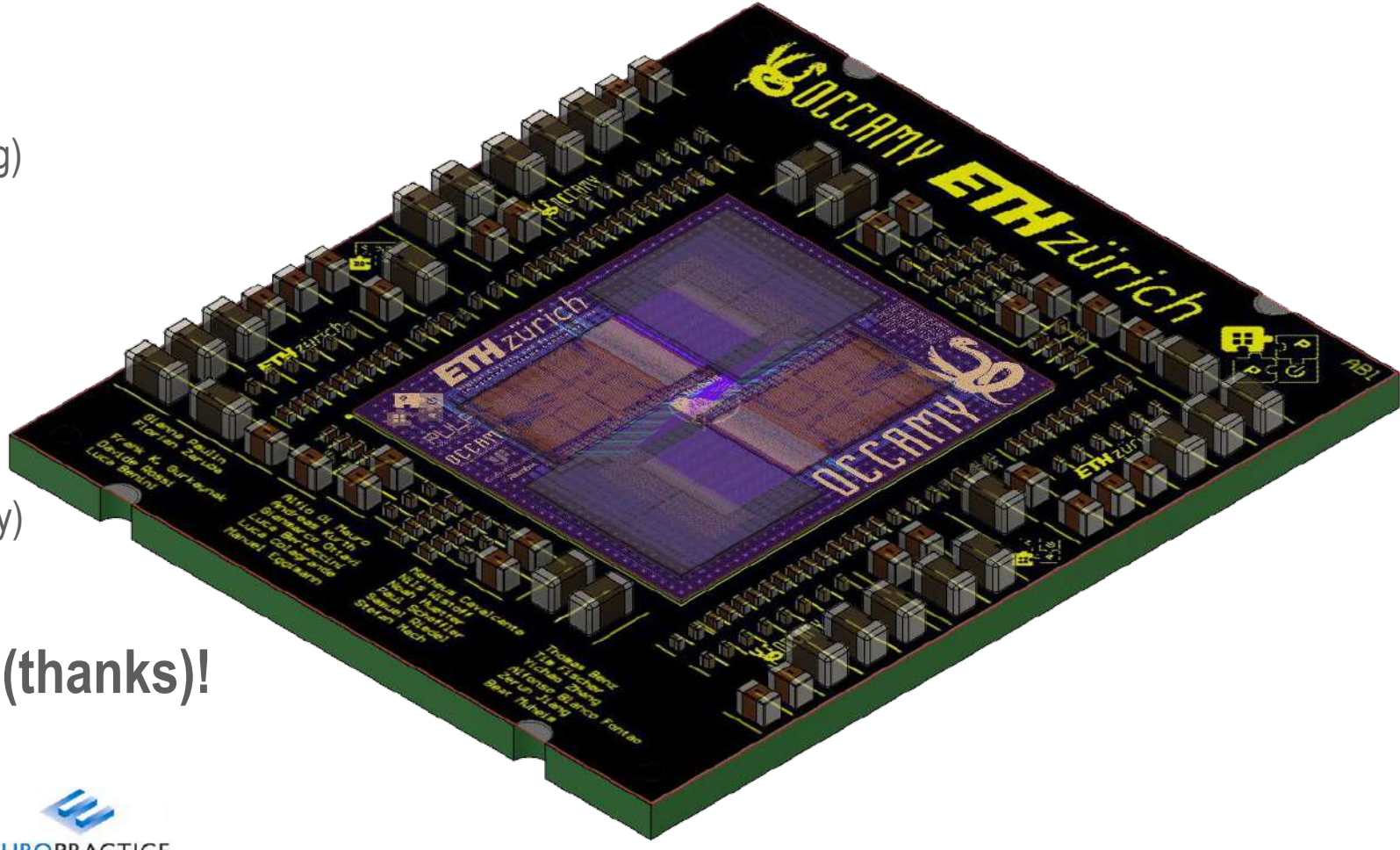
Dual Chiplet System Occamy:

- >430+ RV Cores
- 0.8 T DP-FLOP/s (no overclocking)
- 32GB of HBM2e DRAM
- Low tens of W (est.)

Aggressive 2.5D Integration

Carrier PCB:

- RO4350B (Low-CTE, high stability)
- 52.5mm x 45mm



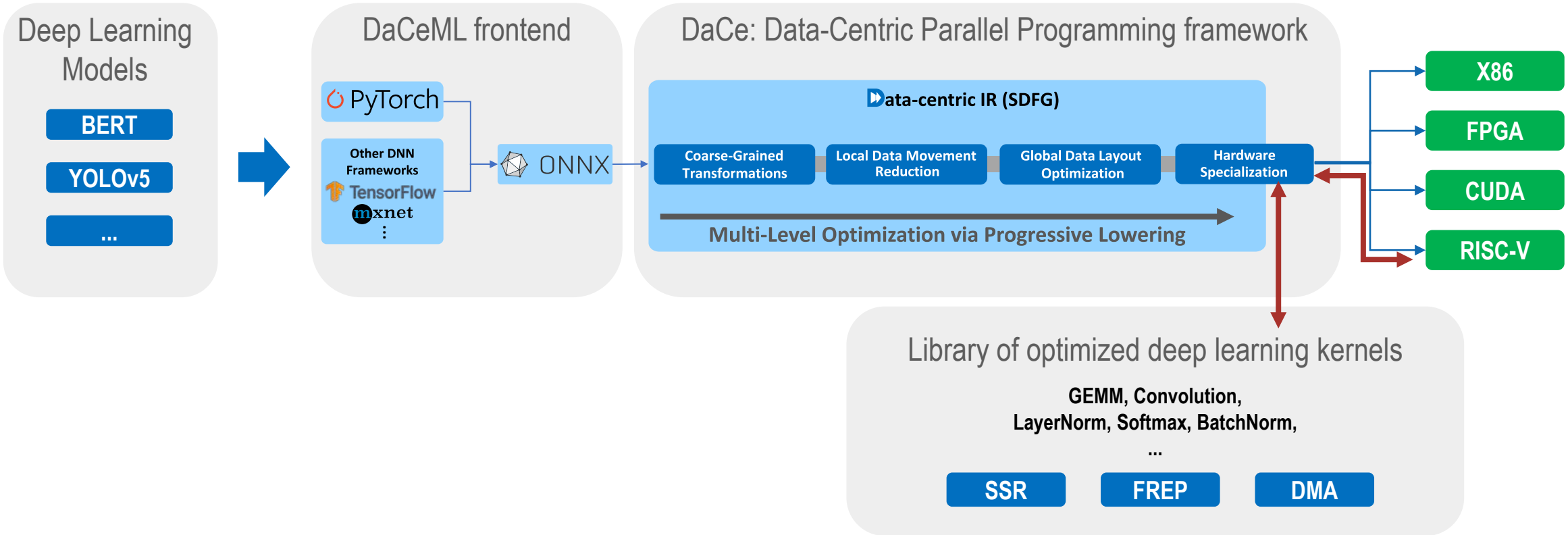
Industry partners are key (thanks)!

GlobalFoundries
SYNOPSYS®
EUROPRACTICE IC SERVICE
Rambus
Micron
Fraunhofer IZM
ETH zürich

Programming Occamy: DACE

Highly expressive DSL family – high-level transformations, support for explicitly managed memory

DaCeML: Data-Centric Machine Learning



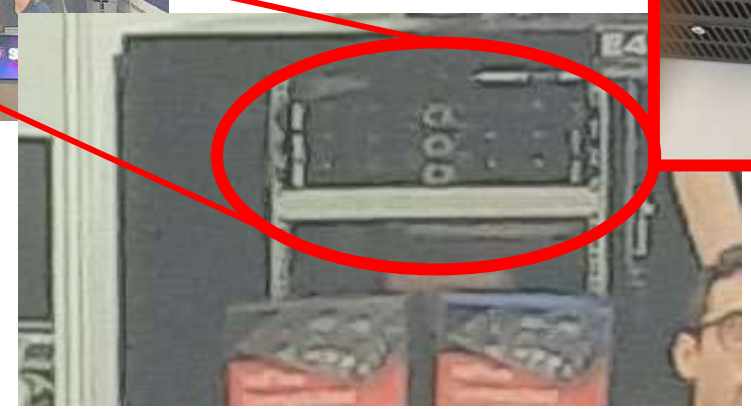
Efficient Chiplet architecture in Perspective

1. Multi-cluster single-die scaling → strong latency tolerance, modularity
2. NoC for flexible Clus2Clus, Clus2Mem, C2C 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

System Level: Monte Cimone, the first RISC-V Cluster

4x E4 RV007 1U Custom Server Blades:

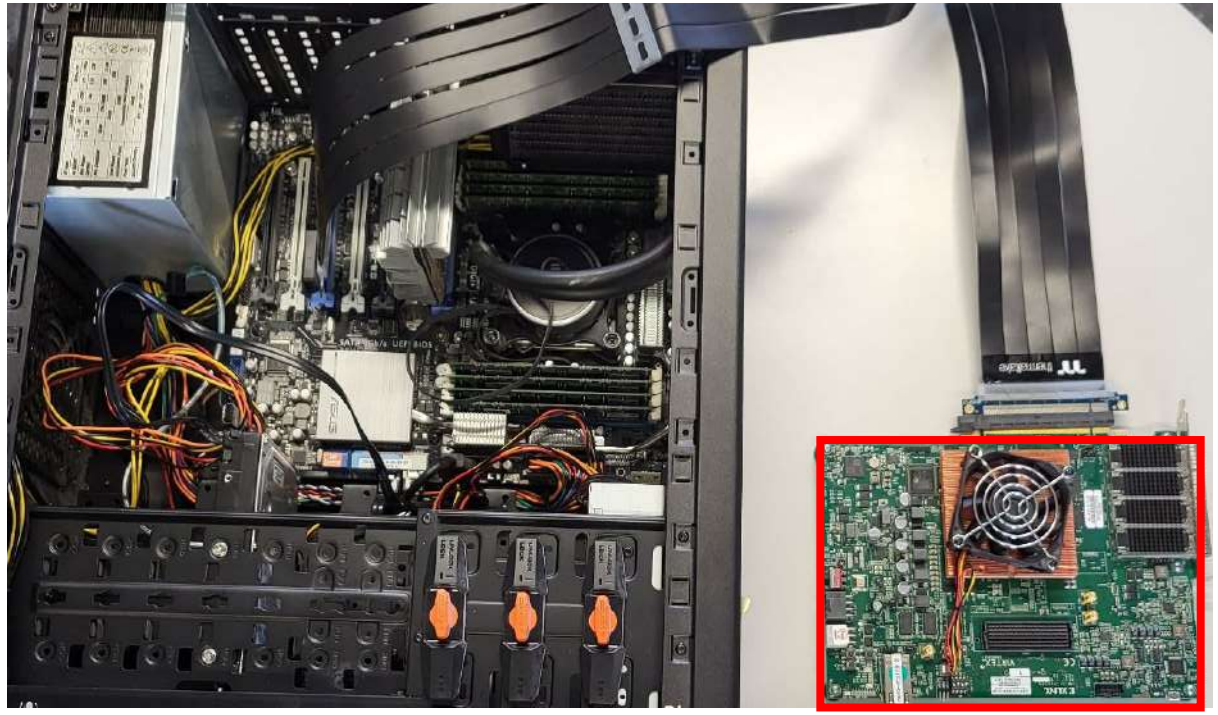
- **2x SiFive U740 SoC** with **4x U74 RV64GCB** cores
- 16GB of **DDR4**
- 1TB node-local **NVME storage**
- **PCIe expansion** card **w/InfiniBand HCAs**
- **Ethernet + IB parallel networks**



Designed for HPC “pipe cleaning”

Preparing for Occamy: Accelerator on PCIe cards

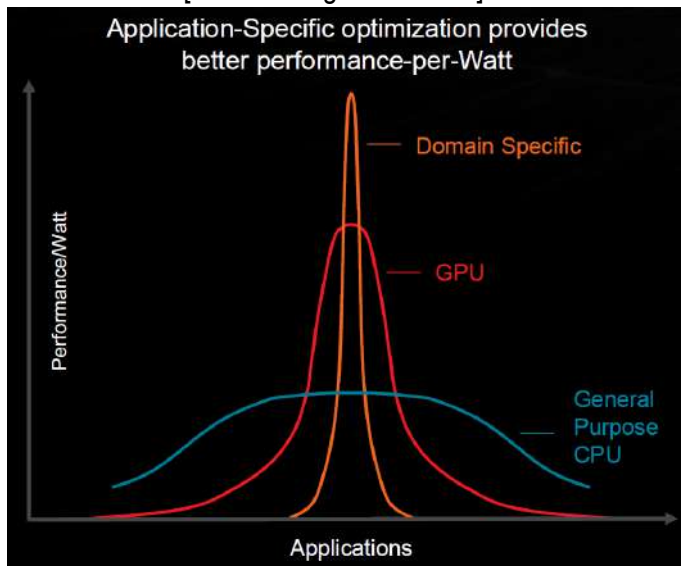
- Currently using FPGA-mapped “tiny Occamy”
  - VCU128 with HBM
- Supporting hybrid usage
  - Boot directly on standalone CVA6
  - Do not boot and let the Host control the cluster
  - HW probing by on-board device tree overlays
- High SW stack re-usability for both modes
  - Same Linux drivers to map the cluster
  - Same OpenMP offloading runtime

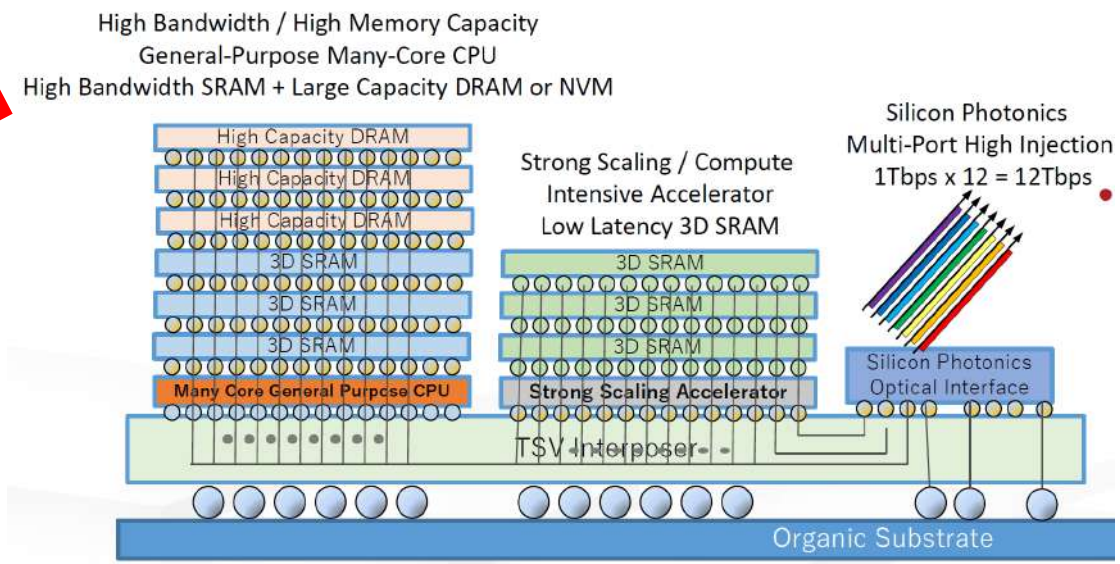


Conclusion

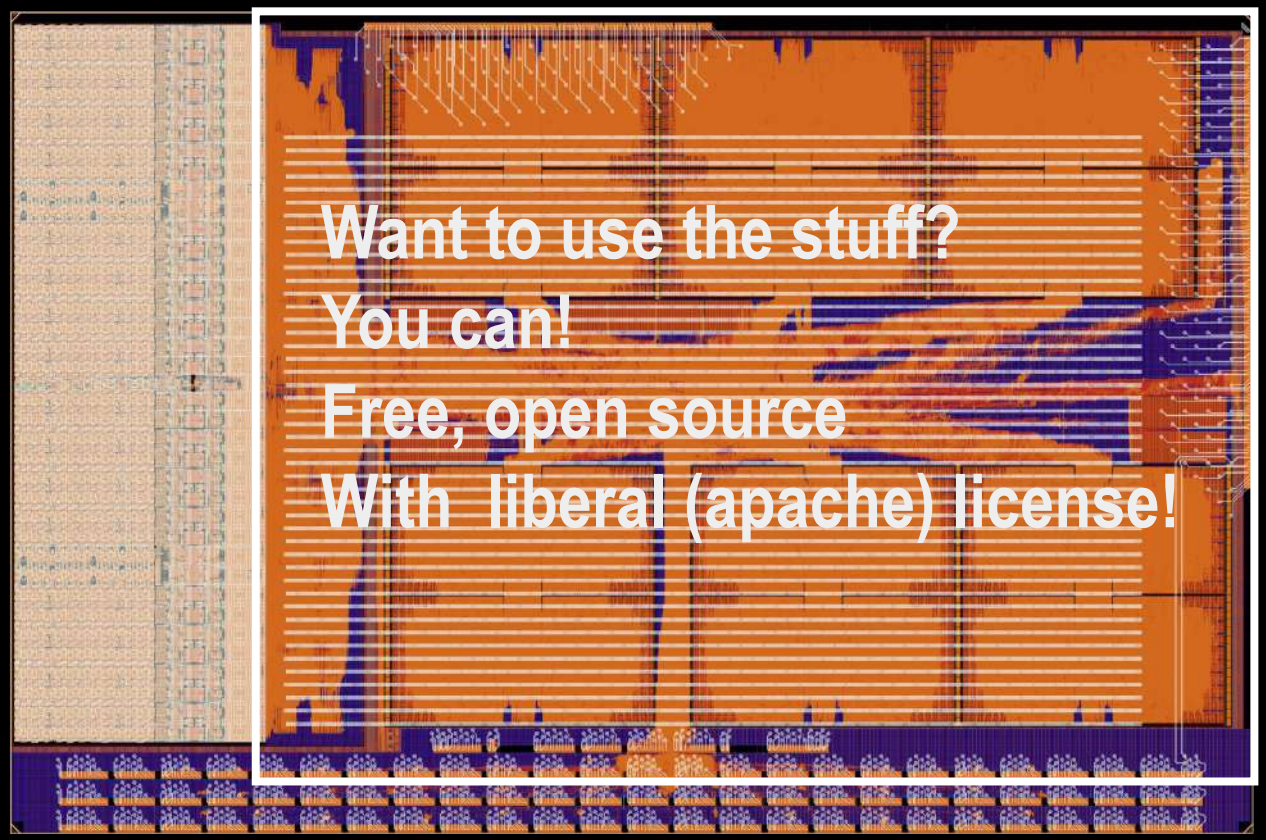
- Energy efficiency quest: PE, Cluster, SoC, System
- Key ideas
  - Deep PE optimization → extensible ISAs (RISC-V!)
  - VNB removal + Latency hiding: large OOO processors not needed
  - Low-overhead work distribution. Latency hiding → large “mempool”
  - Heterogeneous architecture →host+accelerator(s)
- Game-changing technologies
  - “Commoditized” chiplets: 2.5D, 3D
  - Computing “at” memory (DRAM mempool)
  - Coming: optical IO and smart NICs, swiches
- Challenges:
  - High performance RV Host?
  - RV HPC software ecosystem?

[AMD Naffziger ISCAS22]





[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 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