ETH zürich

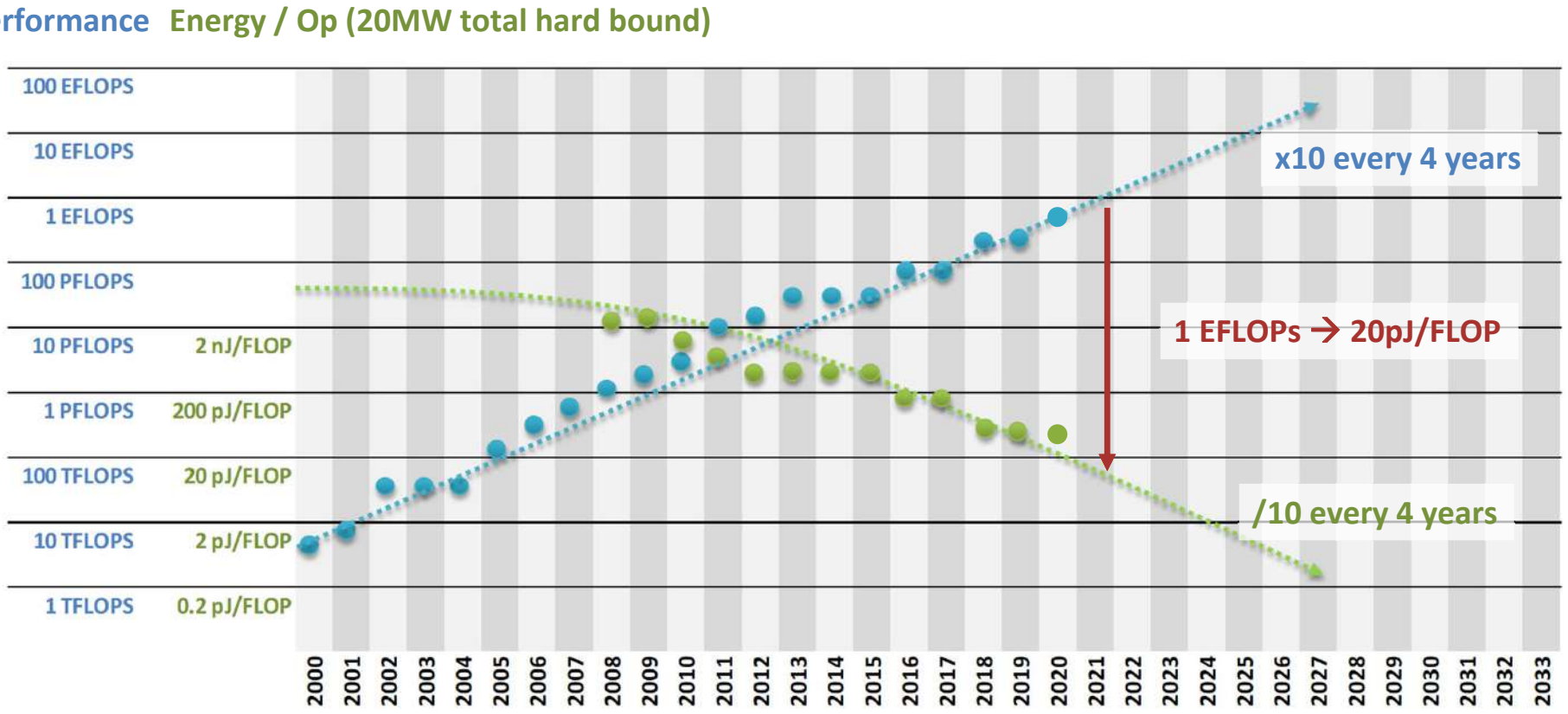
Integrated Systems Laboratory

Tricking Dr von Neumann with Magic Birds: from Snitch to Manticore and Occamy

Luca Benini ETHZ

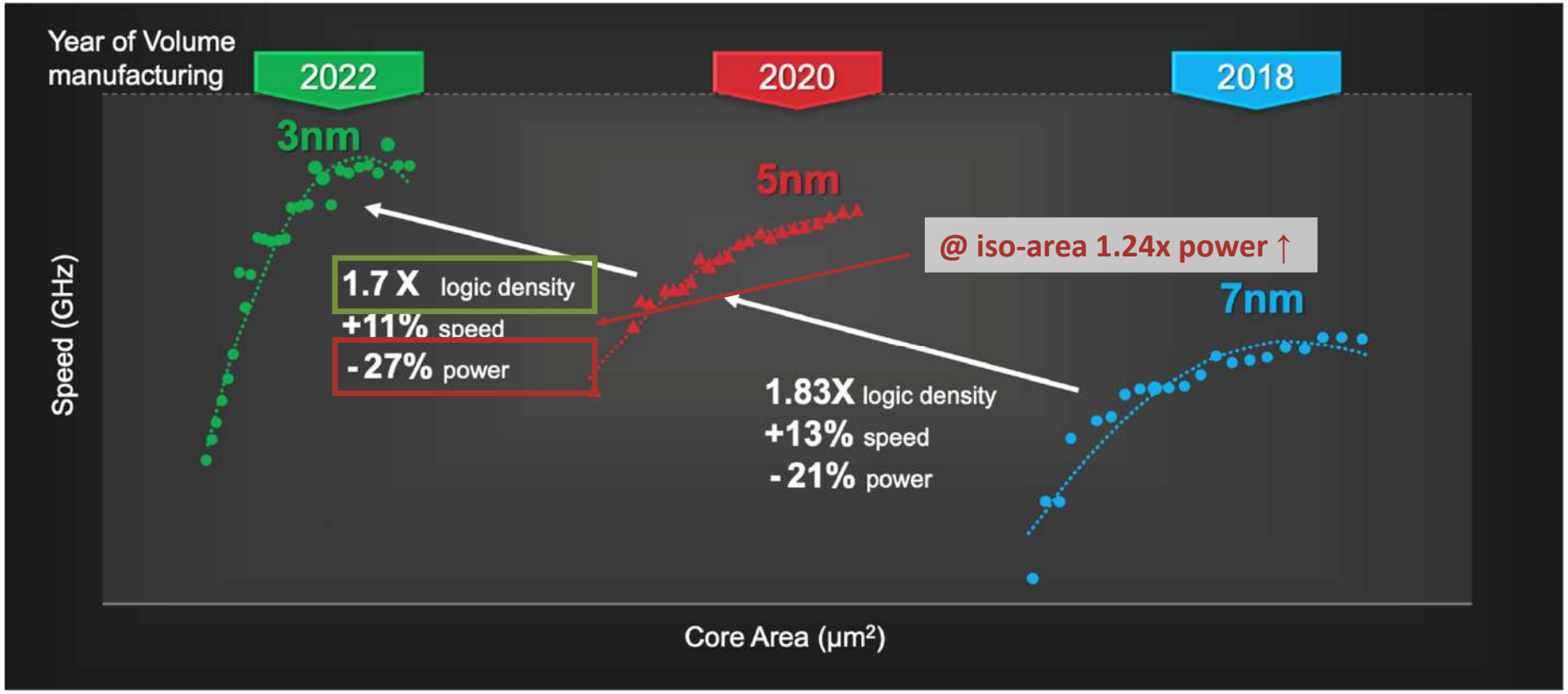
DITET

Energy Efficiency at Exascale



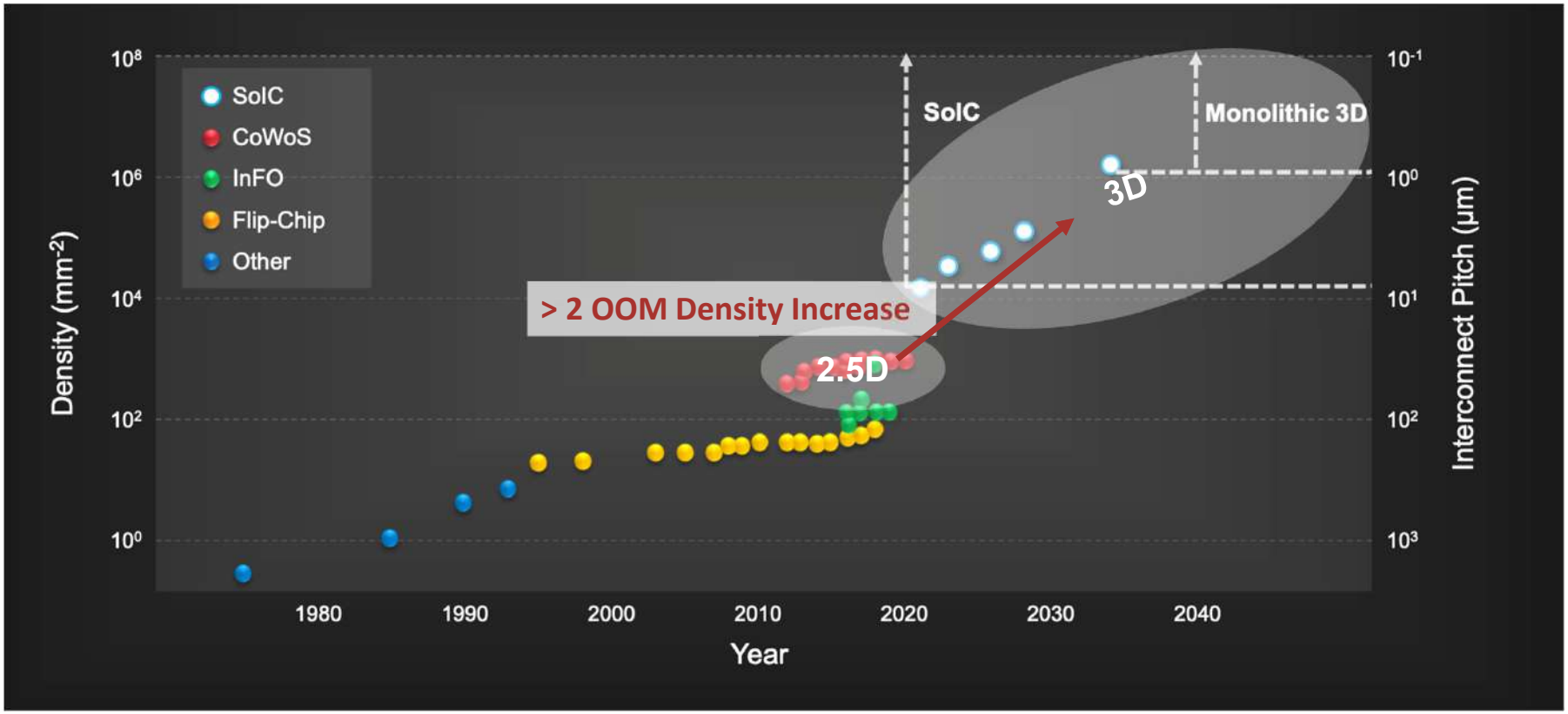
HPC is power bound → need 10x energy-efficiency improvement every 4 years

Technology Scaling



Unleashing the Future of Innovation, Dr. Mark Liu (Chairman of TSMC) at ISSCC21

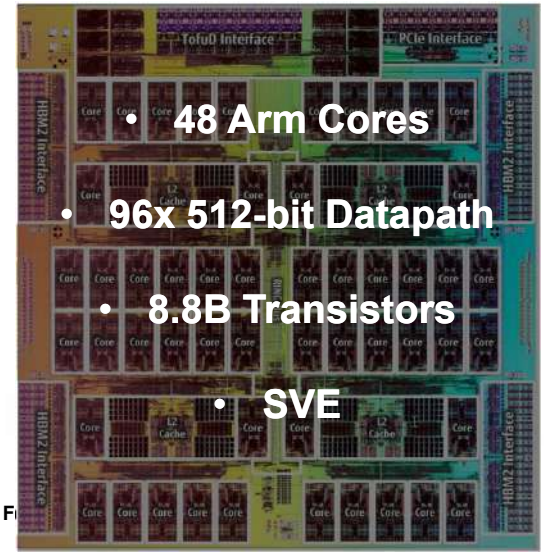
System Scaling

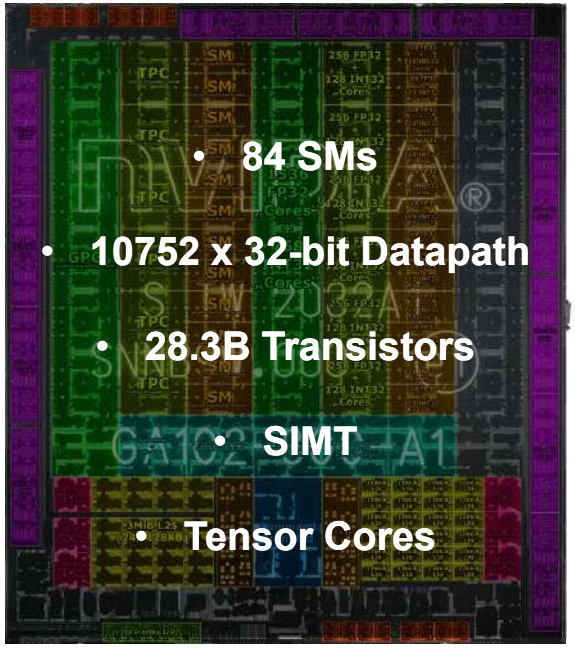


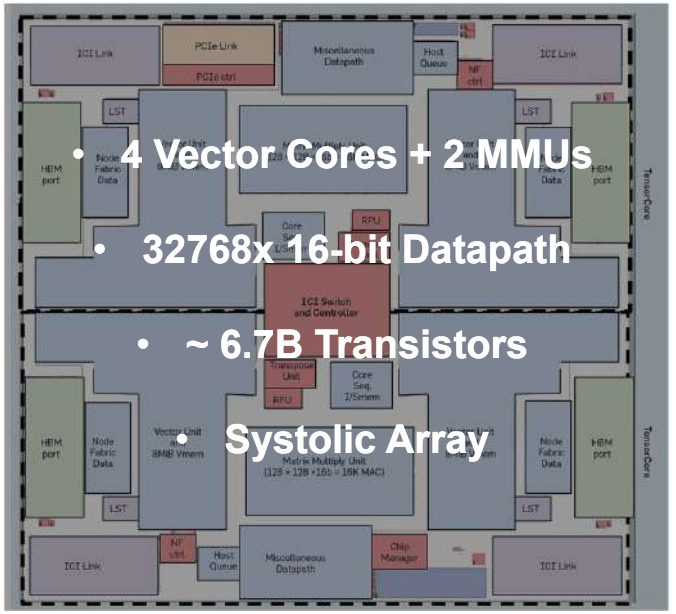
Density increases faster than power reduction → Architectural innovations are inevitable!

The Trend in Industry

Where are we headed?

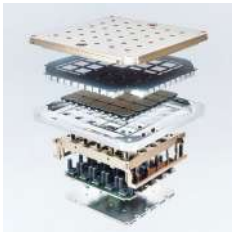


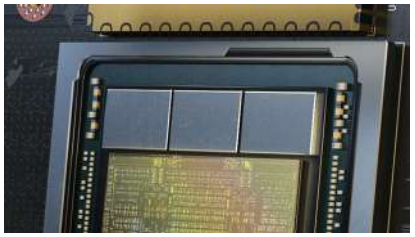




Energy-sustainable performance is the **primary** driver for architecture evolution

- Closer main memory → **2.5D (HBMx), CIM**
- From larger dies to more dies → **chiplets, wafer-scale**, **3D**
- Mitigate the *Von Neumann Bottleneck* → **more work with one instruction + high FU utilization**





Vector vs. SIMT vs. Systolic vs. ??

Not All Programs Are Created Equal

1. Processors can do two kinds of useful work:

Decide (jump to different program part)

- Modulate flow of **instructions**
- **Smarts**:
  - Don't work too much
  - Be clever about the battles you pick (e.g., search in a database)
- Lots of decisions
  Little number crunching

Compute (plough through numbers)

- Modulate flow of **data**
- **Diligence**:
  - Don't think too much
  - Just plough through the data (e.g., machine learning)
- Few decisions
  Lots of number crunching

2. Many of today's challenges are of the **diligence** kind:
   1. Tons of data, algorithm ploughs through, few decisions done based on the computed values
   2. **"Data-Oblivious Algorithms" (ML, or better DNNs are so!)**
   3. **Large data footprint + sparsity**

An Illustrative Example

- Dot product / scalar product
- Multiply elements of a vector and sum up

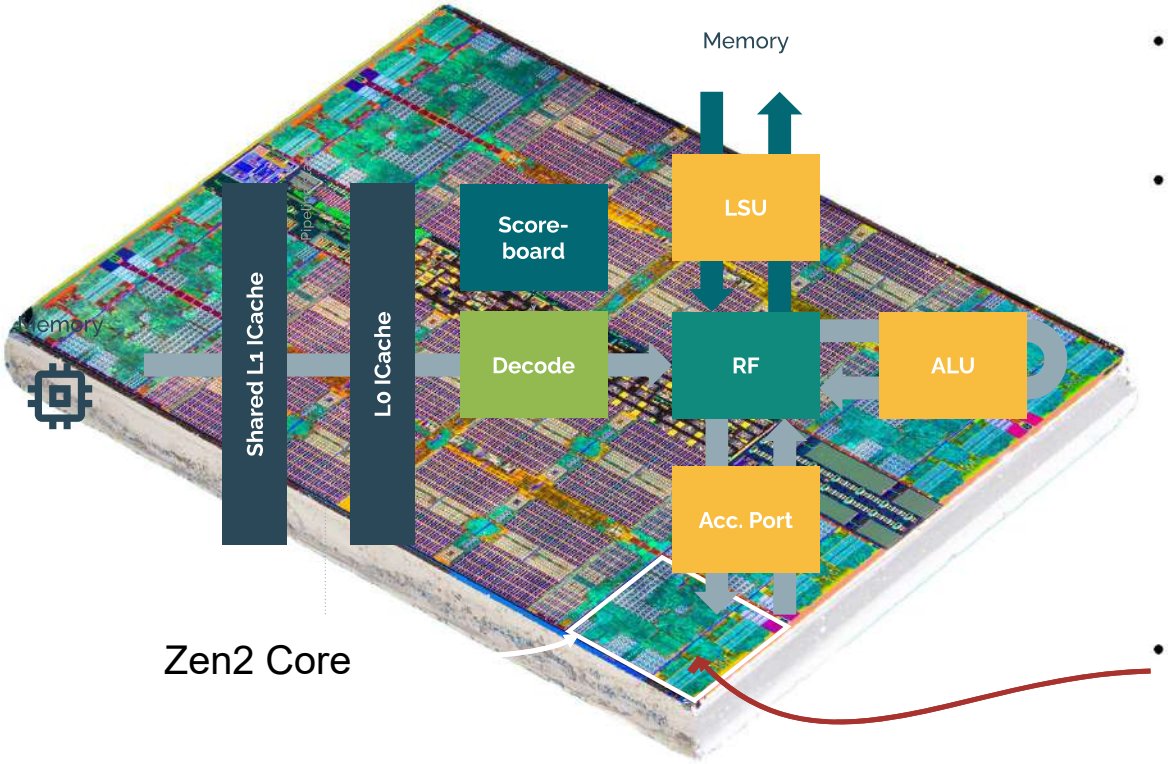
```
double sum = 0;
for (int i = 0; i < N; ++i) {
  sum += A[i] * B[i];
}
```

```
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?

Snitch: Tiny Control Core

A versatile building block.



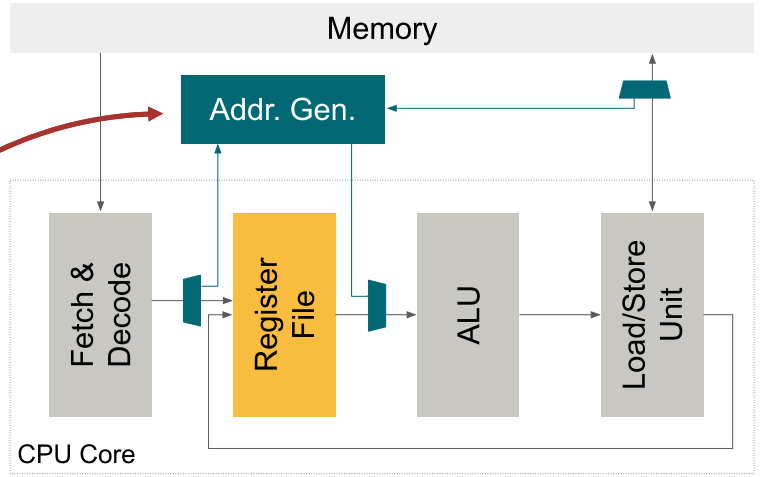
Zen2 Core

- Simplest core
- Focus on key features:
  - **Lightweight microarchitecture**
  - **Extensibility**: Performance through ISA extensions
  - **Latency tolerant**
  - **Competitive frequency**
- Around 15-25 kGE

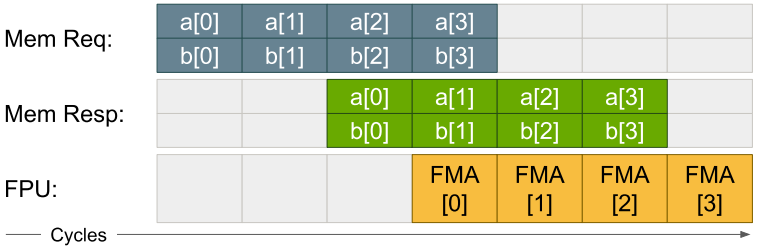
Stream Semantic Registers

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



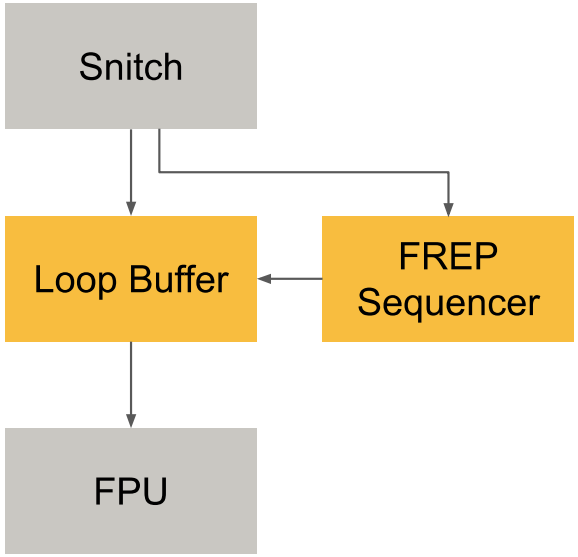

```
loop:                       scfg 0, %[a], ldA
fld r0, %[a]                scfg 1, %[b], ldB
fld r1, %[b]        ⟶       loop:
fmadd r2, r0, r1            fmadd r2, ssr0, ssr1
```



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

Floating-point Repetition Buffer

Remove control flow overhead




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

→

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

- **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**

Efficiently Data Mover

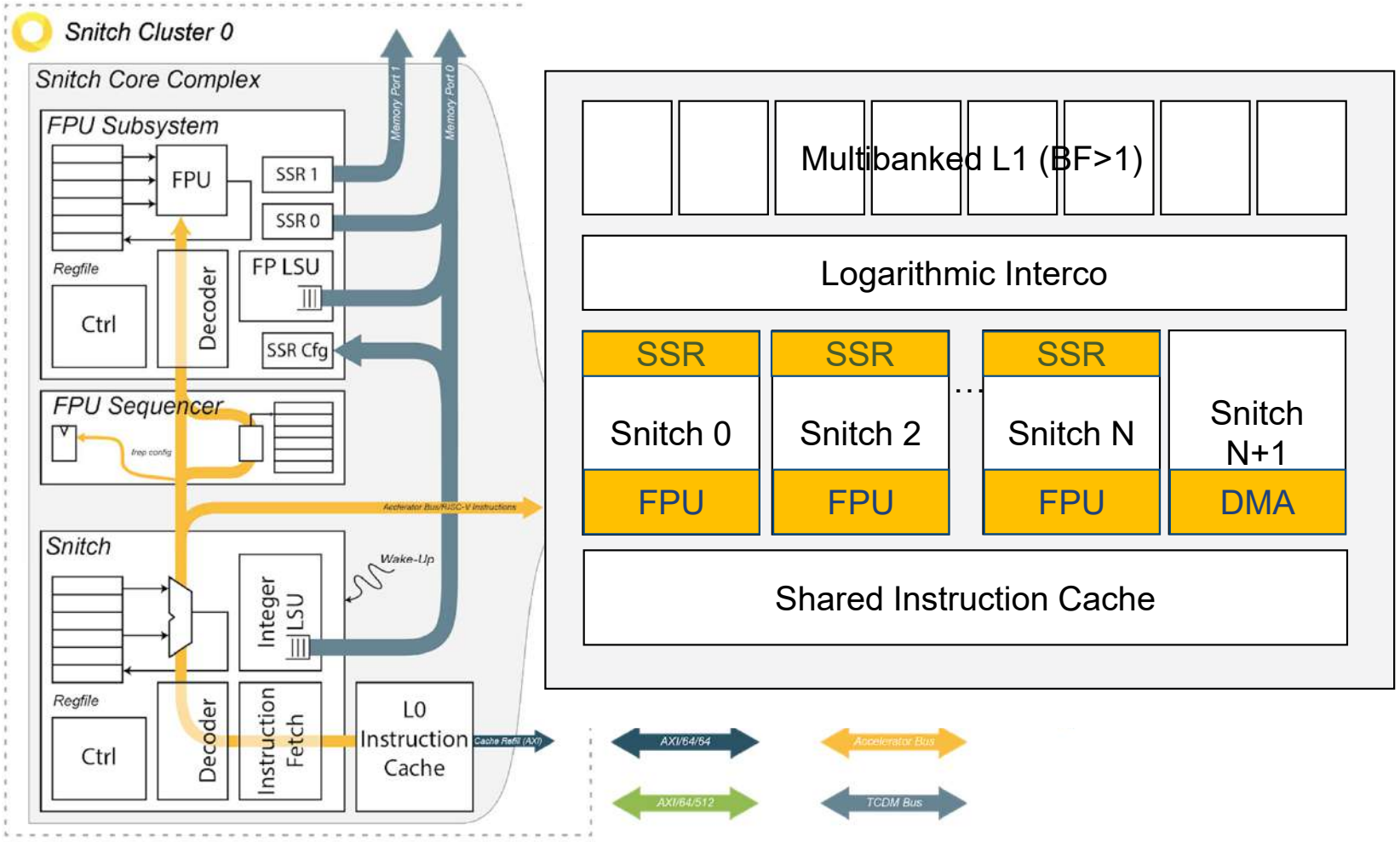
hide L2, main memory latency

- 64-bit AXI DMA
- Tightly coupled with Snitch (<10 cycles configuration
- Operates on wide 512-bit data-bus
- Hardware support to autonomously copy 2D shapes
- Higher-dimensionality can be 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



And where does the Energy go?

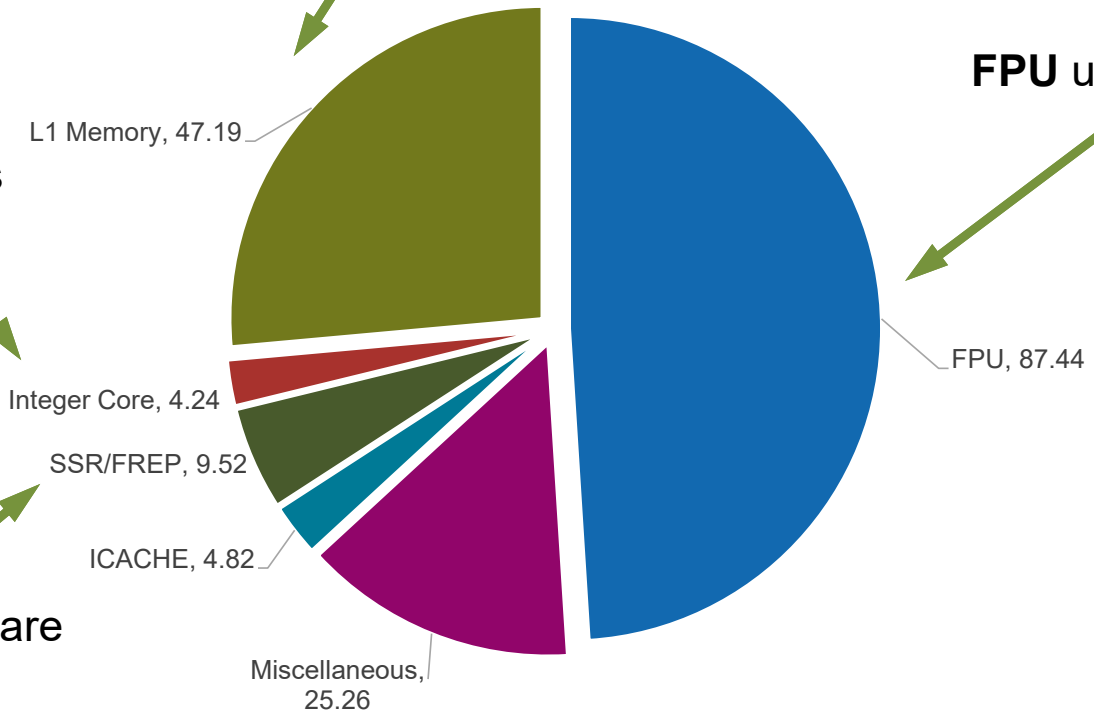
In an 8-core cluster

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

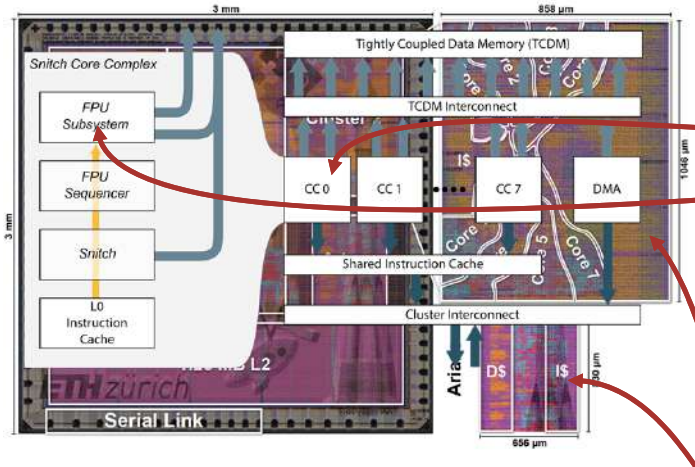
SSR/FREP hardware
uses **5%** of power

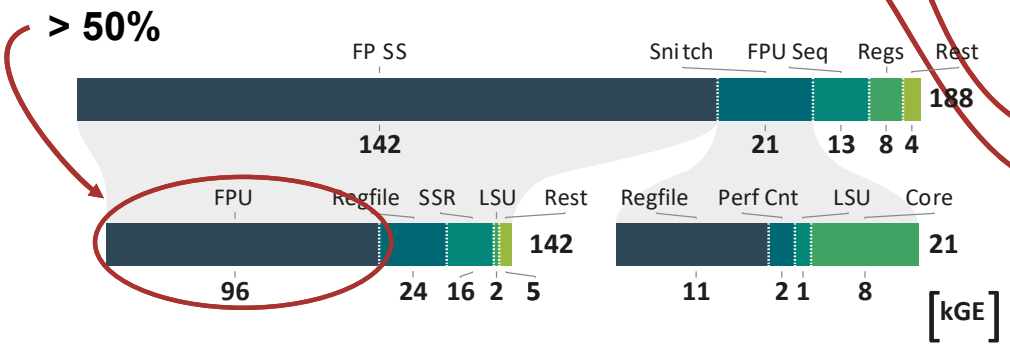


Spending energy where it contributes to the result → **High Efficiency**

Baikonur

The first Snitch-based test chip.



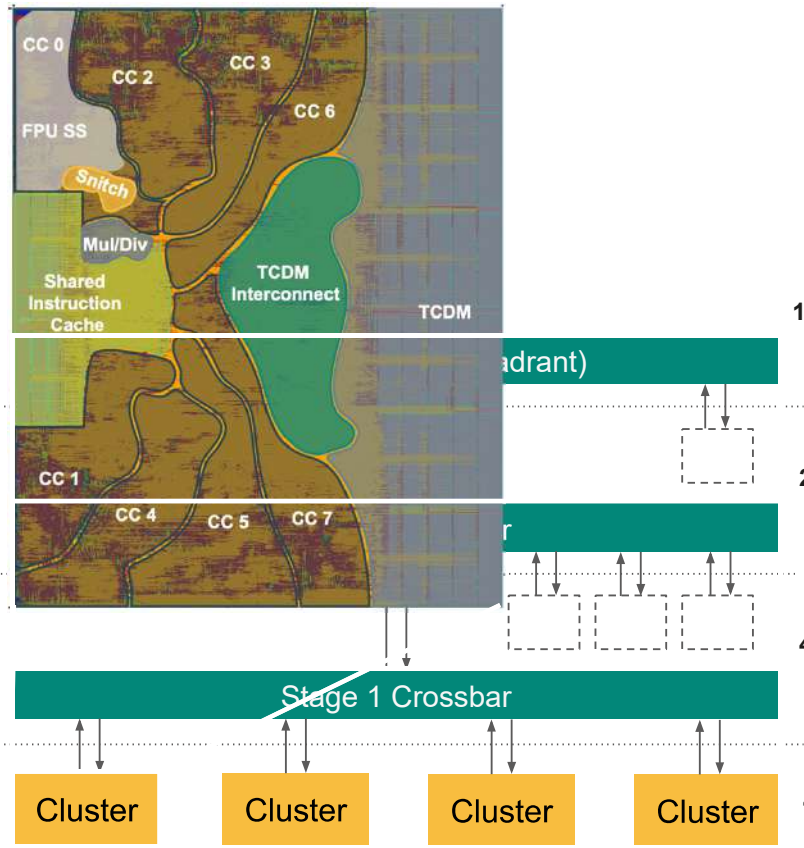


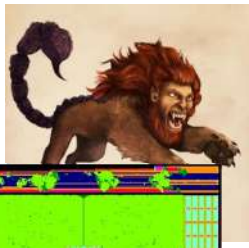
- Snitch compute cluster:
  - 8x RV32G Snitch cores
  - 8x large FPUs
  - > 40% energy spent on FPU
  - High energy-efficiency of **~80 Gdpflop/s/W**
  - **104 Gspflop/s/W** similar to accelerators
- 9 mm^2 prototype in 22nm
- Testbed for key architectural components
  - Snitch octa-core cluster
  - Ariane cores

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

Manticore

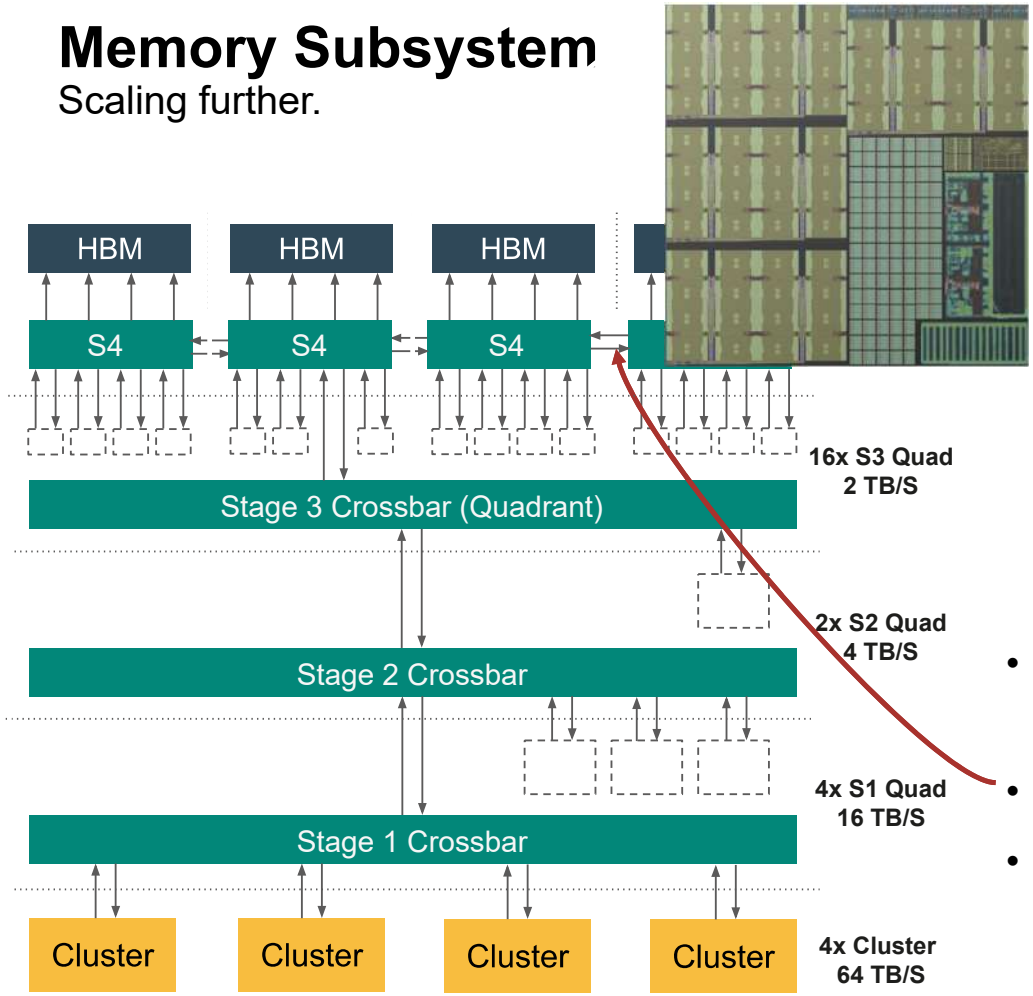
The scale-out architecture





Memory Subsystem

Scaling further.



HBM: Matching the bandwidth with the memory interface

Block-wise DMA accesses:

- High bus utilization ≈ high energy-efficiency
- Multi-dimensional blocks with Snitch DMA
- Good fit for parallel interfaces (HBM/HBI)
- Latency tolerance through double buffering

- Different to GPU memory hierarchy

- **Multi-chiplet design**
- **HBI**: Scaling across dies (NUMA)

Programming Model

Where we are headed.

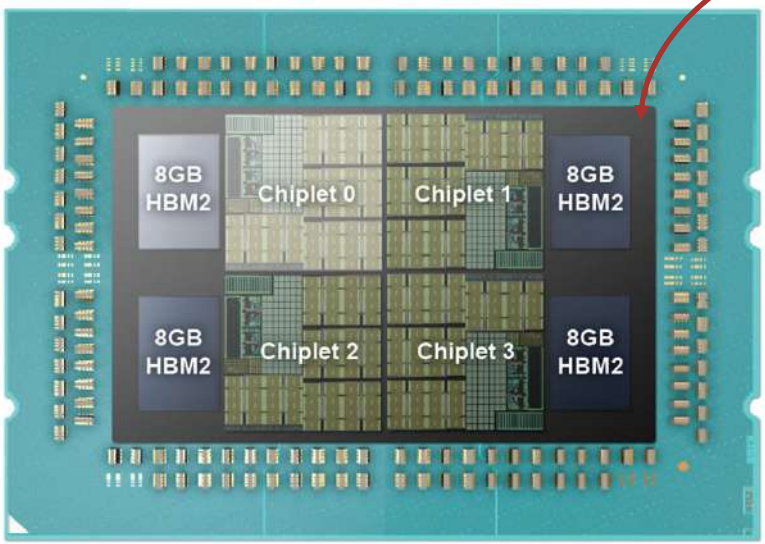
```
void main() {
  unsigned repetition = 2, bound = 4, stride = 8;
  static int data[8] = {1,2,3,4,5,6,7,8};

  __builtin_ssr_setup_1d(0, repetition, bound, stride, data);
  static volatile double d = 42.0;

  __builtin_ssr_enable();

  __builtin_ssr_push(0, d);
  volatile double e;
  e = __builtin_ssr_pop(0);
  __builtin_ssr_disable();
}
```

- Multiple layers of abstraction:
  - Hand-tuned assembly
  - LLVM intrinsics
  - FREP inference
  - High-level framework: DaCE
- Bare-metal runtime
- Basic OpenMP runtime
- Automatic schedule of data movement
- Inference of SSR and FREP
  - NumPy

OpenMP

Manticore Multi-Chip Concept

Summarizing



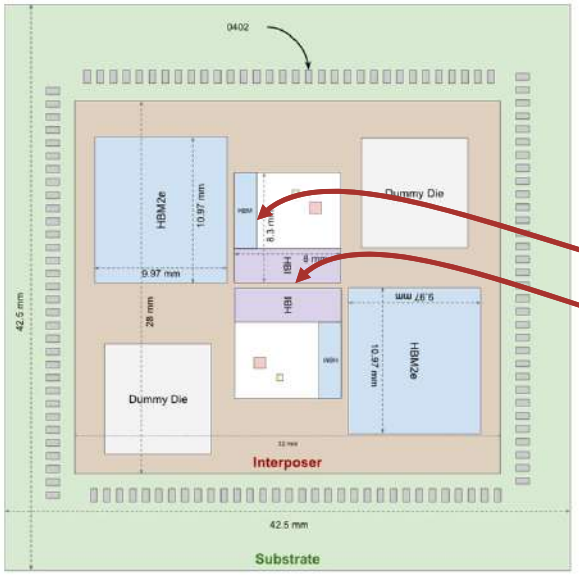
- Four chiplets and 8GB HBM2 on an interposer
  - Interposer enables **high-bandwidth**, **energy-efficient** parallel interfaces
- High D2D bandwidth
- High die to HBM bandwidth

- Total 4096 Snitch cores, peak > 8 Tdpflop/s
- Four CVA6 ”manager” cores

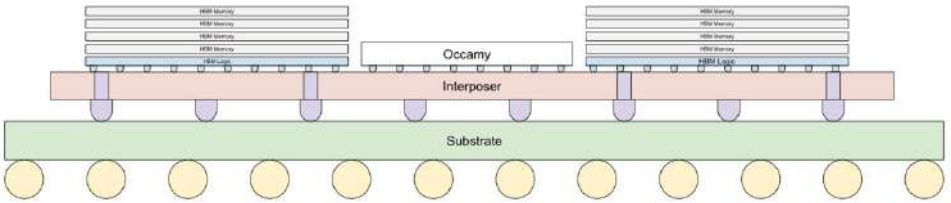
Outperforms SoA, **open** building-blocks, foundation of **next generation** high-performance computing systems!

Occamy: from Concept to silicon!

Silicon implementation of all key IPs (core, Phys, chiplet) of Manticore in GF12



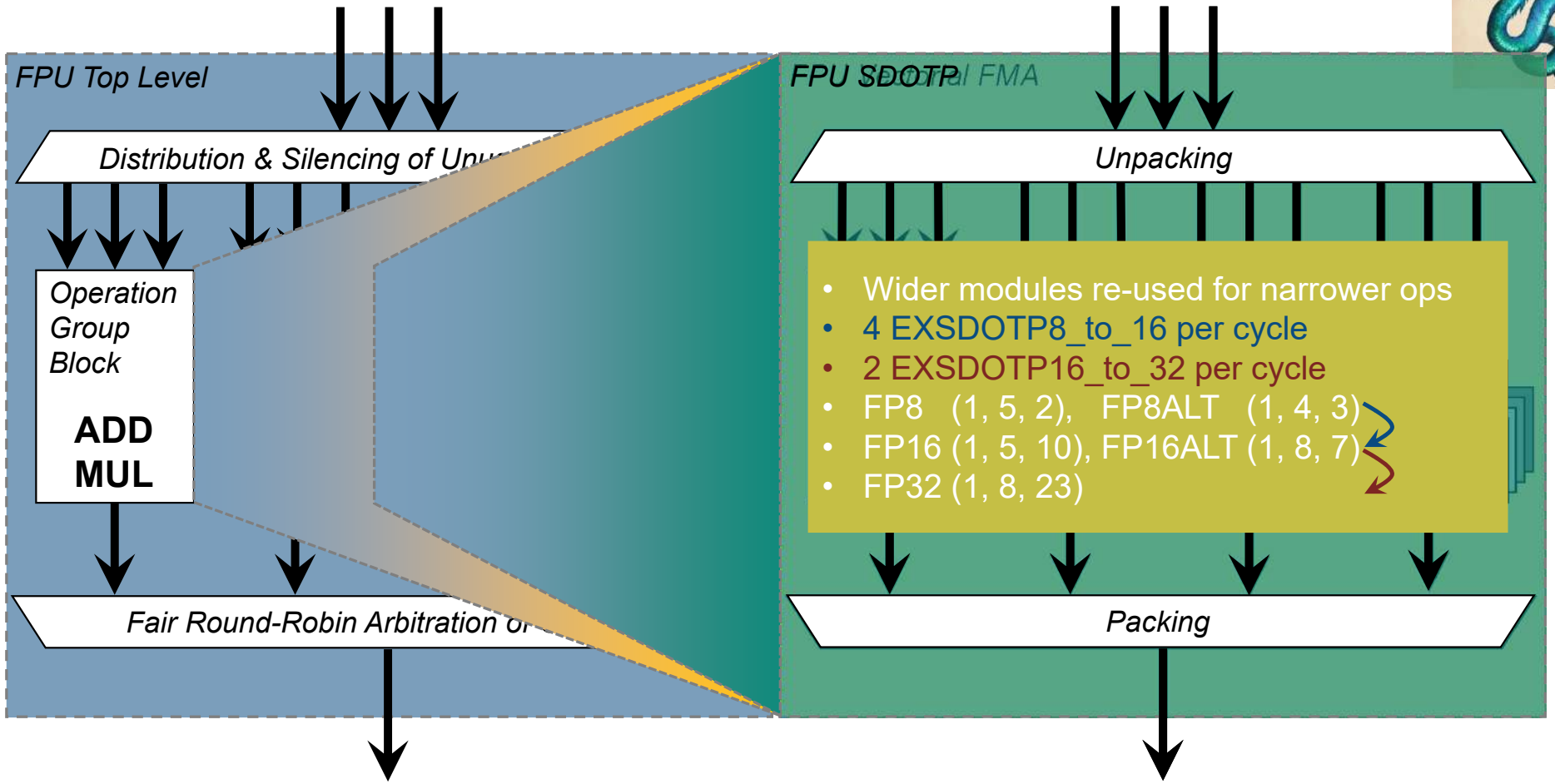
- Dual-chiplet configuration in 12nm FinFET
  - 8-10 L1 Quadrants, ~ 32 clusters/die
- Advanced high BW interfaces on interposer:
  - HBM2e for die-to-memory
  - 24 channel AIB for die-to-die
  - Controller developed in-house
- **Architectural improvements:**
  - **SIMD datatypes**
  - **Sparsity support**
  - **Virtual memory**
  - **Atomics and interrupts**
  - **Icache hierarchy**



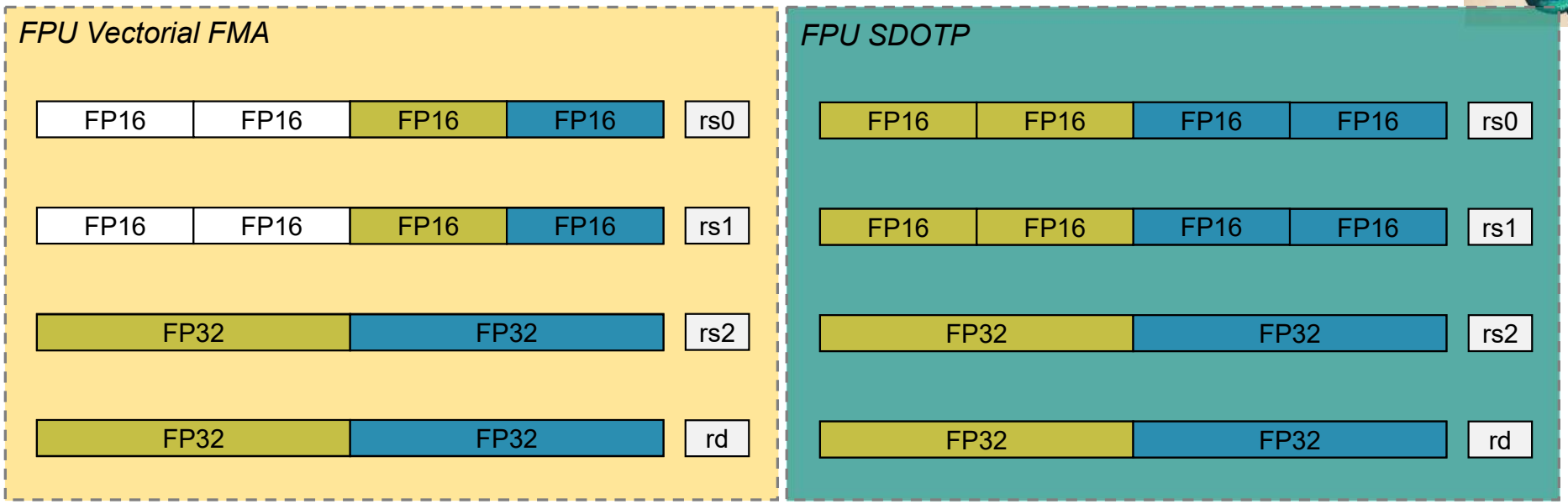
GlobalFoundries
SYNOPSYS®
Rambus

ETH zürich

FPU and new SDOTP unit



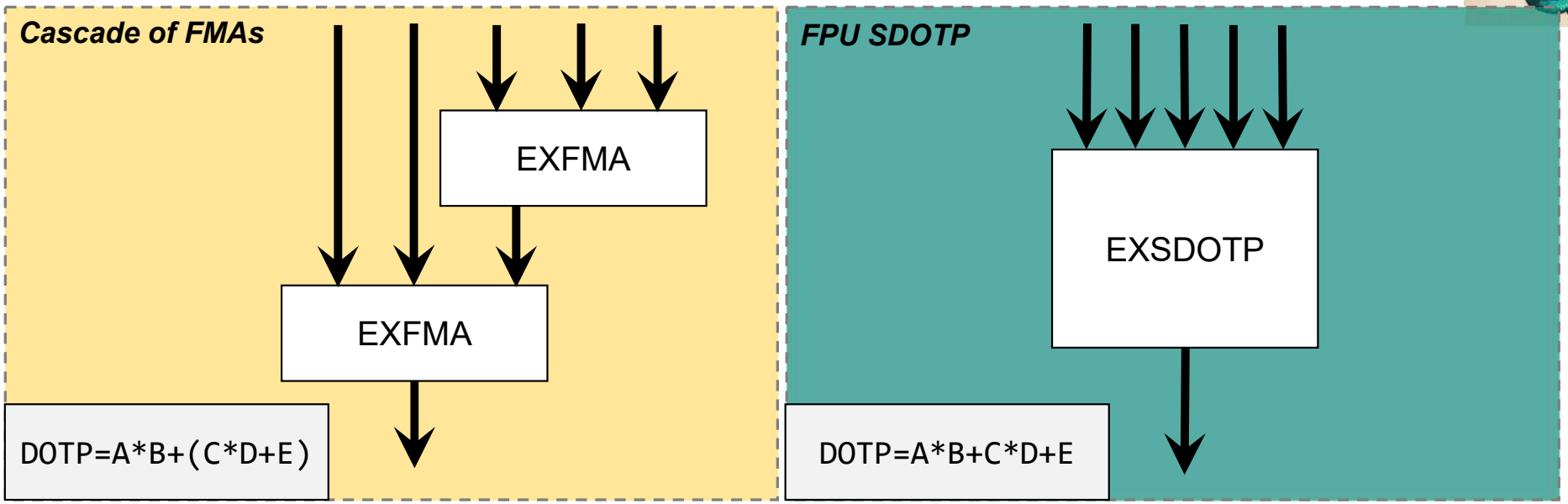
EXFMA vs EXSDOTP (Expanding Dot Product w. accumulation)



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

- $rd_{32} = rs0_{0,16} * rs1_{0,16} + rs0_{1,16} * rs1_{1,16} + rs2_{32}$
- Vectorial 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)

Sparse Linear Algebra Extensions

$$\vec{y} = \begin{bmatrix} 0 & 0 & 7 & 9 & 0 \\ 3 & 0 & 4 & 1 & 2 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 7 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 7 \\ 3 \\ 2 \\ 9 \end{bmatrix}$$

($\boldsymbol{A}$ and $\vec{x}$)

A_vals[] = {7, 9, 3, 4, 1, 2, 7}

A_idcs[] = {2, 3, 0, 2, 3, 4, 2}

A_ptrs[] = {0, 2, 3, 3, 4}

```
for i in 0 to A_nrows:
  for j in A_ptrs[i] to A_ptrs[i+1]:
    y[i] += A_vals[j] * x[A_idcs[j]]
```

SpVV

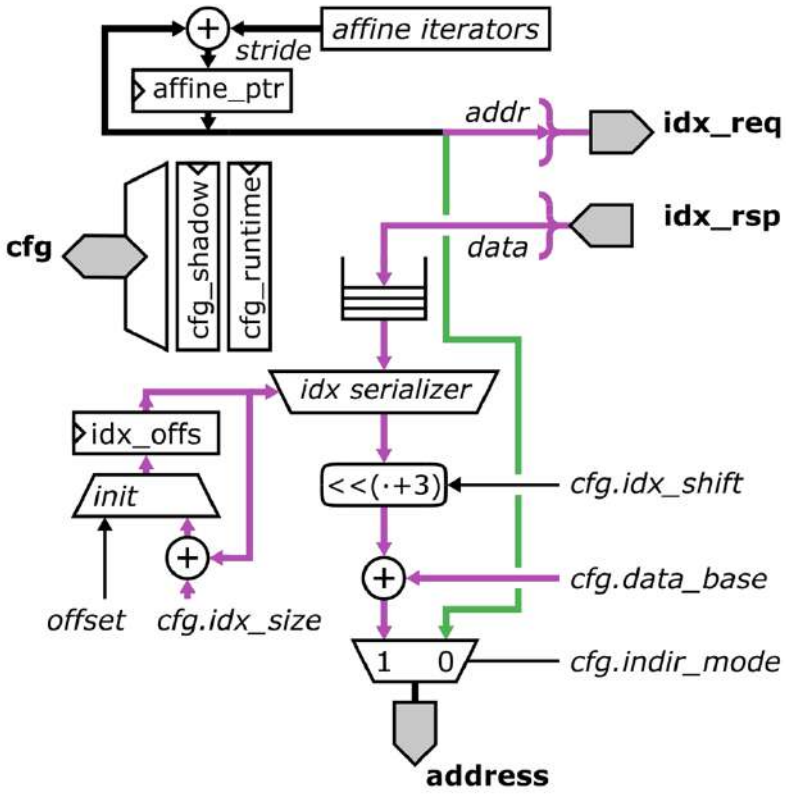
CsrMV

- Sparse tensors are *ubiquitous*: ML, physical simulation, math, CS, economics…
  - Wide range of sparse formats
- Example: *CSR Matrix* × *Vector* (CsrMV)
  - Large control-to-compute ratio
  - Chellenging memory access patterns
  - Low reuse opportunities

➢ Low FU utilizations on traditional architectures
  - Xeon Phi KNL: 0.7% FP64[1]
  - AGX Xavier (CuSPARSE[2]): 2.1% FP32

[1] B. Xie, J. Zhan, X. Liu, W. Gao, Z. Jia, X. He, and L. Zhang, "CVR: efficient vectorization of SpMV on x86 processors," in CGO '18, 2018, pp. 149–162.
[2] Nvidia Corporation, "Cuda Toolkit 10.0." [Online]. Available: https://developer.nvidia.com/cuda-toolkit

Indirection SSRs

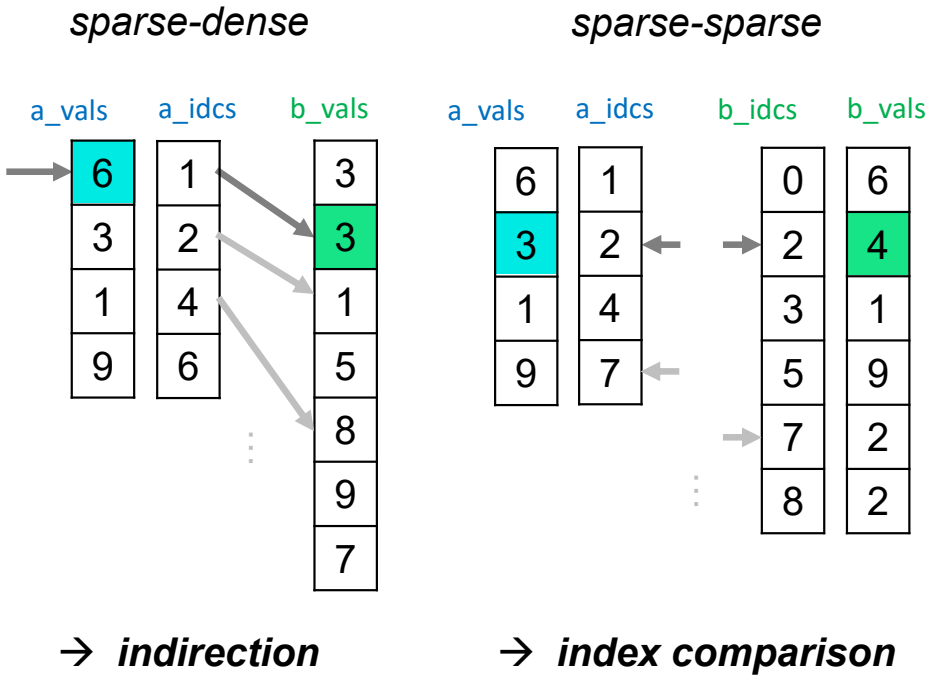


- Accelerate indirections in *sparse-dense LA* (e.g. CsrMV) with SSRs
  - Index lookup *inside* address generator
  - Emit stream of *indirected* addresses
- Originally: N nested loops for *affine iteration*
- Indirection:
  1. Fetch *64b index words* (reuse iterators)
  2. Serialize to *16b* or *32b* indices
  3. Shift to offset (+ extra shift for higher axes)
  4. Add data base address

Sparse-Sparse Linear Algebra

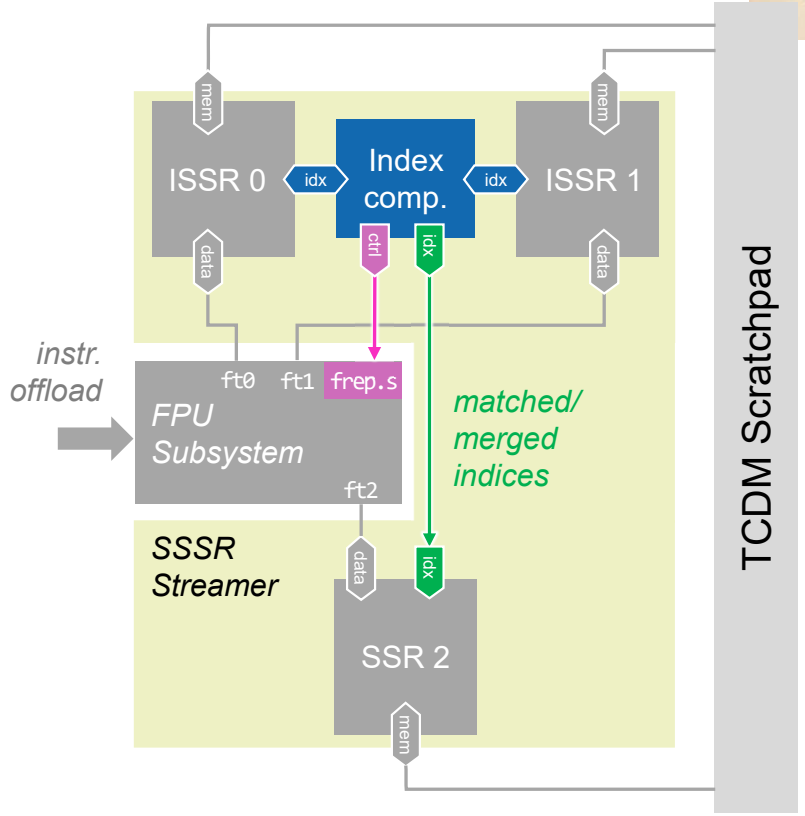
- Lowest sparse tensor axis: *sparse fiber*
  - value array + index array
  - Common to CSR/CSC, CSF, DCSR, …
- Indirection relies on one dense operand
  - b randomly addressed by indices of a
- *Sparse-sparse* LA requires *comparing indices*
  - Multiplication: *intersection* of indices
  - Addition: *union* of indices

➢ Extend ISSRs to enable index comparison
  - Reuse HW → multiple SSRs collaborate



Sparse SSR Streamer Architecture

- 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 Streamer Benefits

- Multi-purpose: backward-compatible to
  - 3 SSRs: dense LA, CONV, FFT, …
  - 2 ISSRs: Sparse-dense LA, codebook streaming, scatter-gather, …
- Notable projected single-core speedups
  - *CsrMV*: up to **7.2×** faster, **80%** FP util.
  - *SpV+SpV*: up to **12×** faster, **80%** FP util.
  - *SpV·SpV*: up to **11×** faster / higher FP util.
- *Multicore CsrMV*: up to **5.8×** faster, **2.7x** less energy
- *Versatile* sparse-sparse processing
  - Wide dynamic range for sparsity
  - Outer, inner, *or* row-wise SpGEMM
  - Flexibility in algorithms, formats used

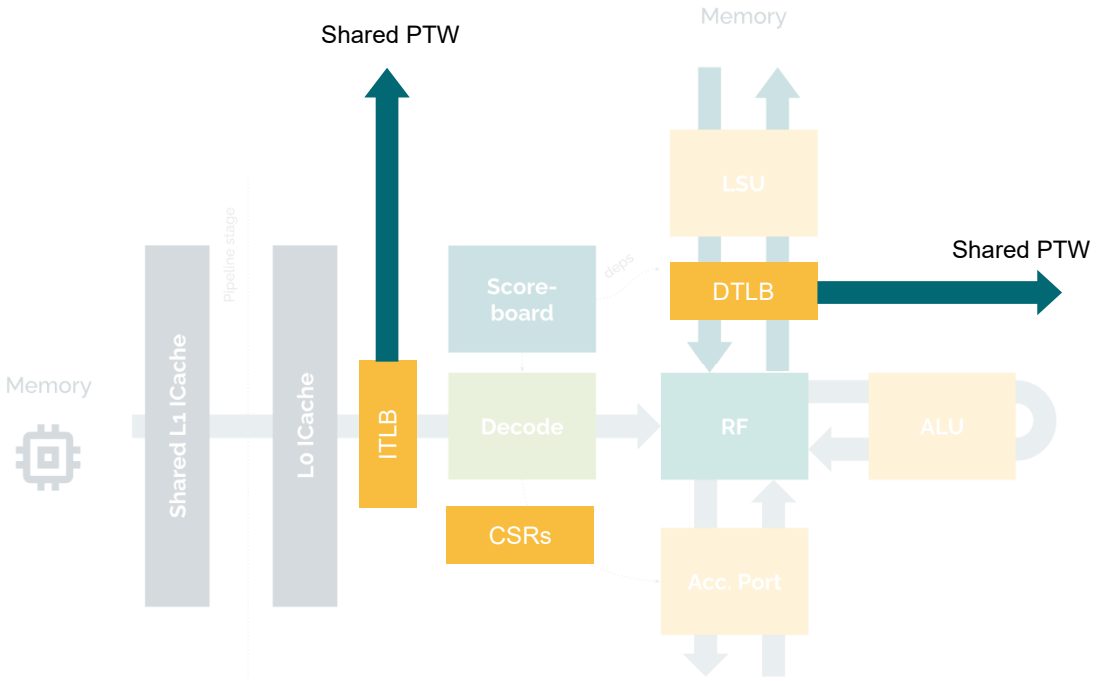
```
dotp:   call   conf_ft0_mst_match
        call   conf_ft1_mst_match      frep.s
        fmadd.d fa1, ft0, ft1, fa1

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

Continuous useful issues

Snitch Cluster
Evolution

- VM Support in Snitch
- Added FP formats
- Easier generation of different cluster configurations
- Event counter
  - Slimmed down
  - Selectable event mask
  - Multiple events selectable: DMA, Icache, TCDM conflicts, etc.

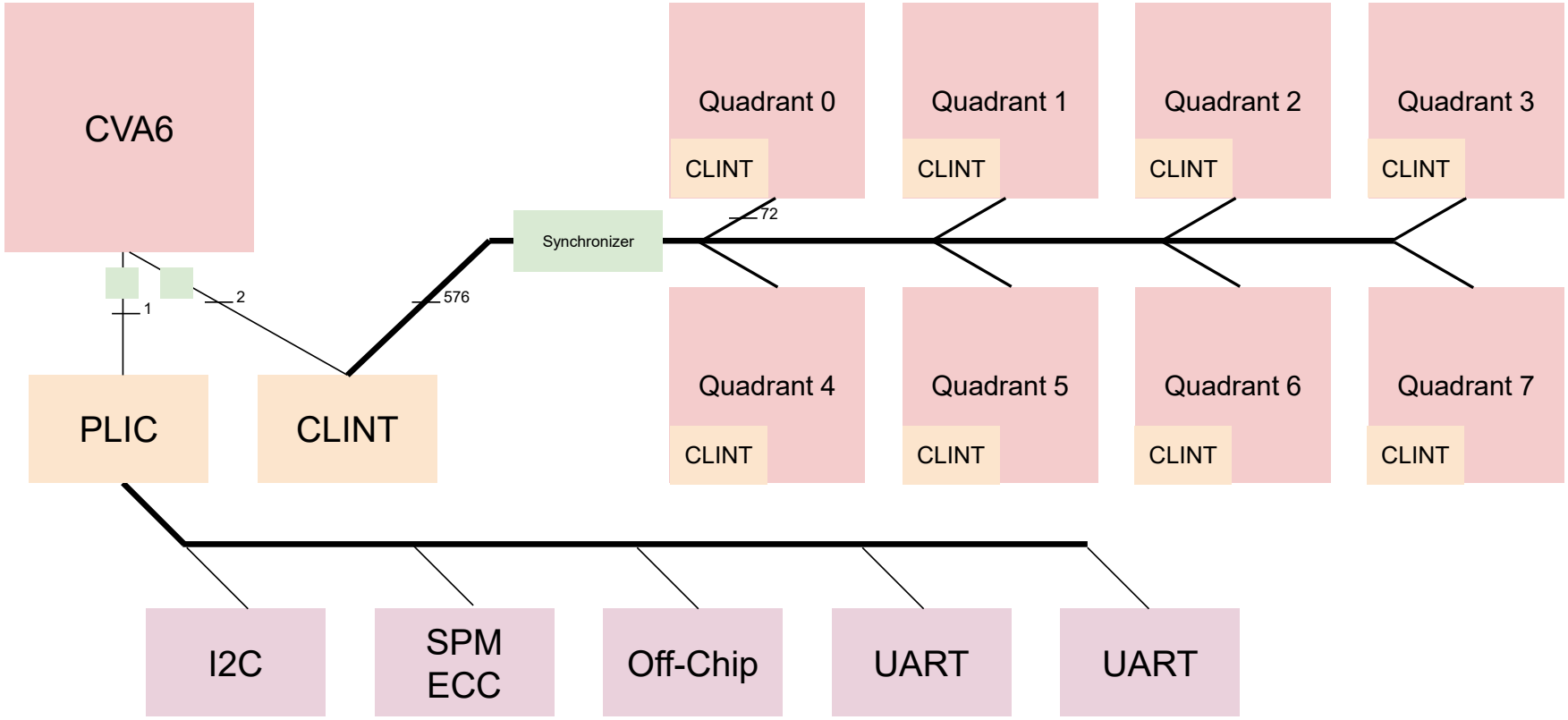
Virtual Memory



- Extended privilege space
  - CSRs
  - Exception capabilities
  - ~3 kGE
- Separate ITLB and DTLB
  - Fully set-associative
  - Configurable size
  - 1-entry configuration (~1 kGE)
- PTW shared by all cores of a hive/cluster
  - Very low overhead
  - Very small (~2 kGE)

- **Addresses can be larger than 32-bit (HPC, big models, etc.)**
- **Memory protection, isolation, and compartmentalization (programming model!)**

Interrupt Subsystem



Atomic and LS/SC Support

1. Cluster-local TCDM memory
2. Foreign cluster TCDM memory

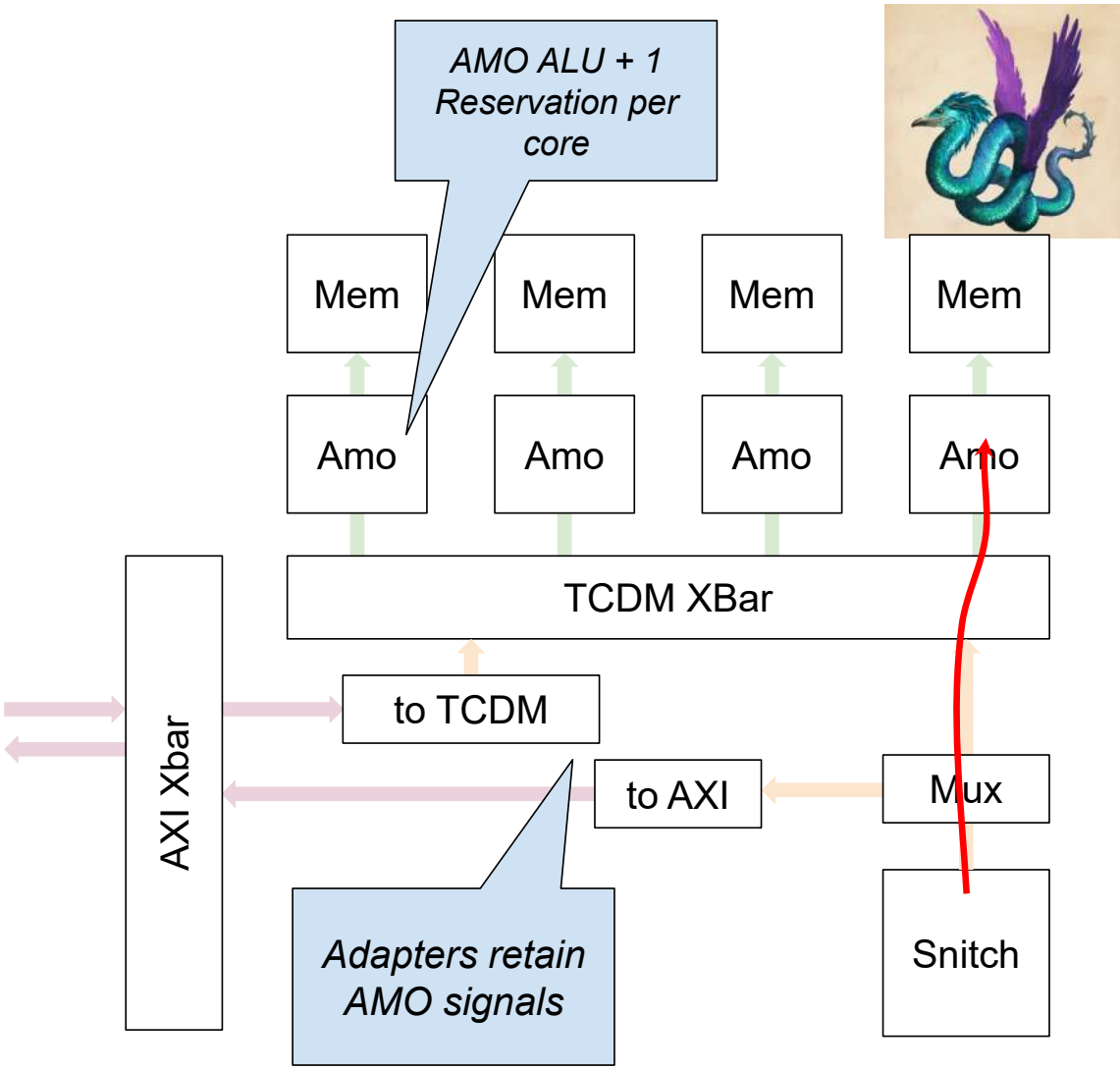
Interface Specifications:

→ Fixed Response Latency Mem Interface

→ TCDM

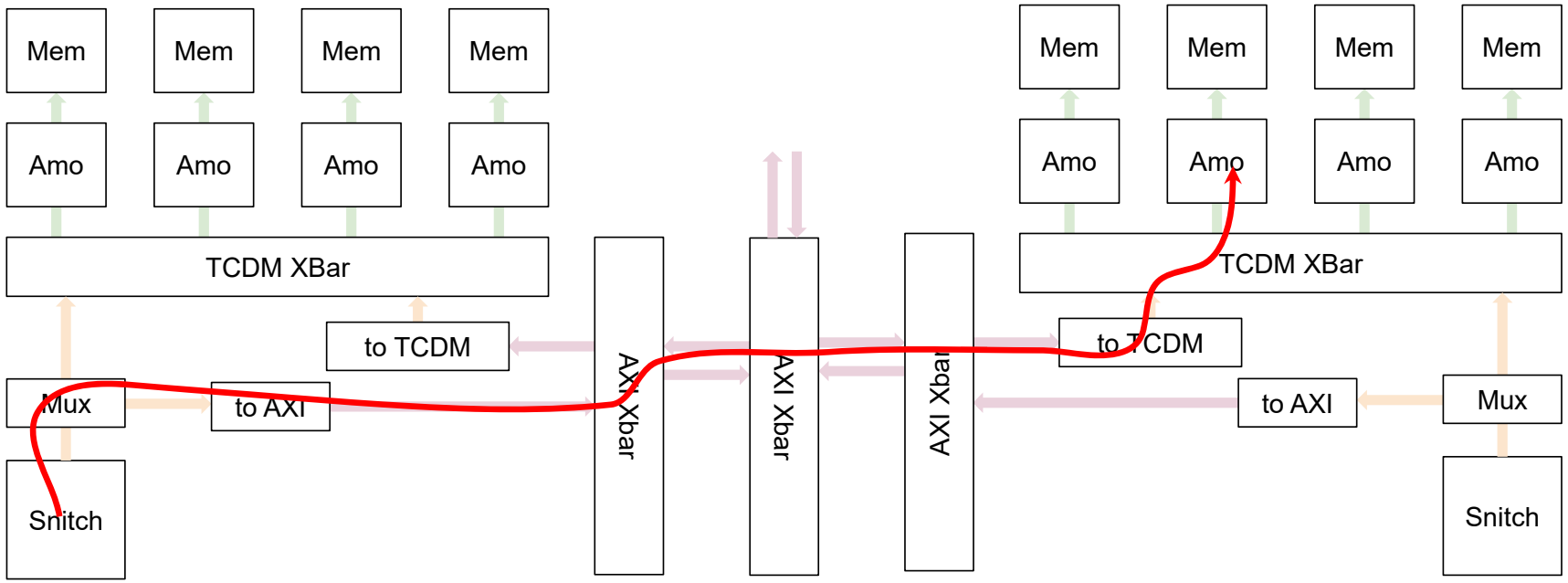
→ AXI

- AMO signals as auxiliary payload



Atomic/LRSC Support in Occamy

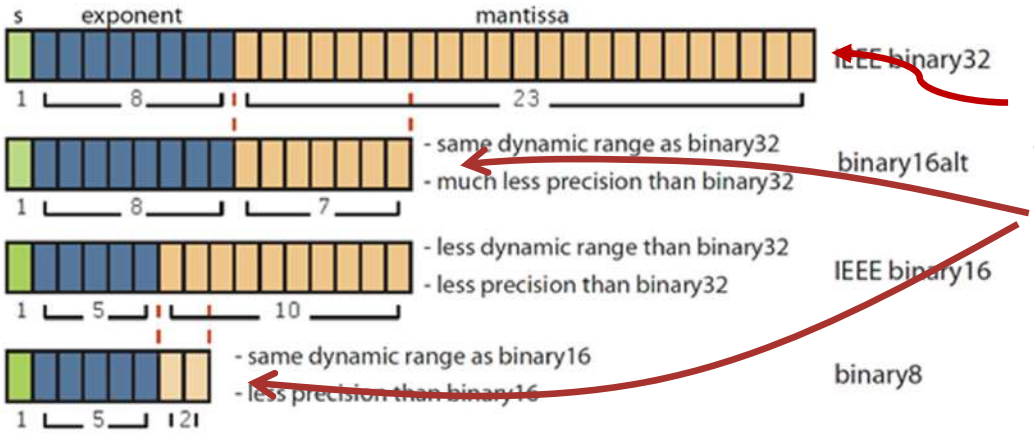
1. Cluster-local TCDM memory
2. Foreign cluster TCDM memory



Target Workload

Mixed precision training, sparse NNs, graphs …

Efficient DNN training platform



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

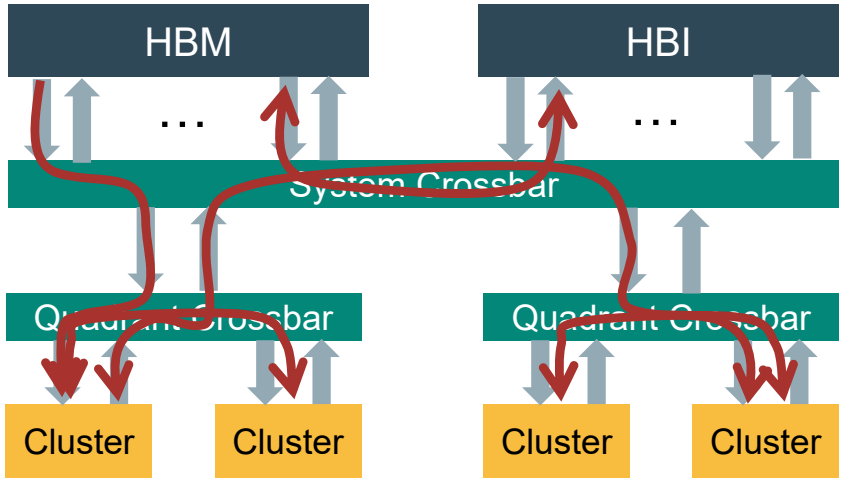
Low precision training with `bf18` and `fp8` are getting traction.

***Occamy* supports wide variety of FP formats and instructions:**

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

Efficiency

How to efficiently move data



Problem: HBM Accesses are not ideal in terms of

- Access energy
- Congestion
- High latency

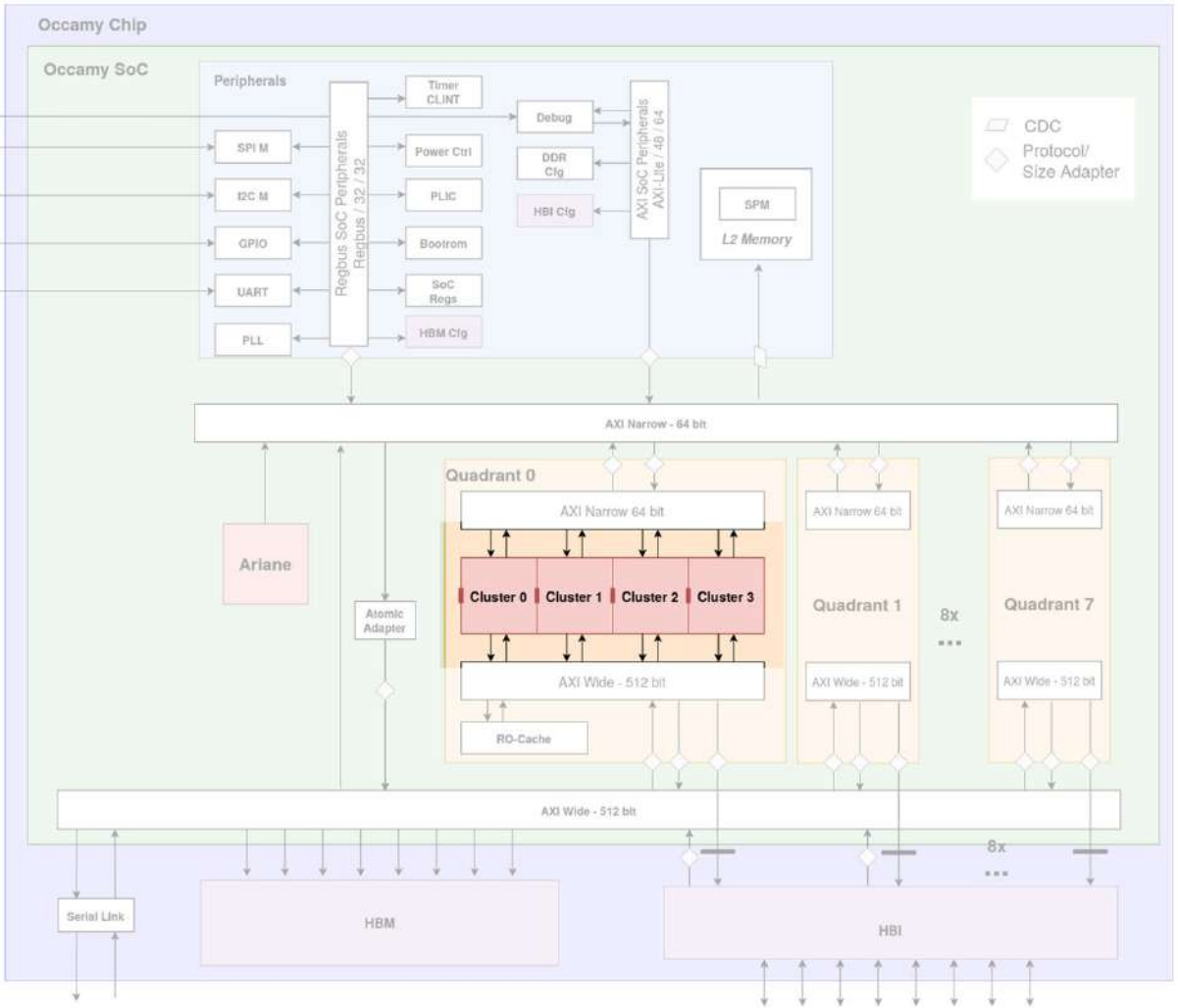
Instead reuse data on lower levels of the memory hierarchy

- Between **clusters**
- Across **quadrants**

Smartly distribute workload

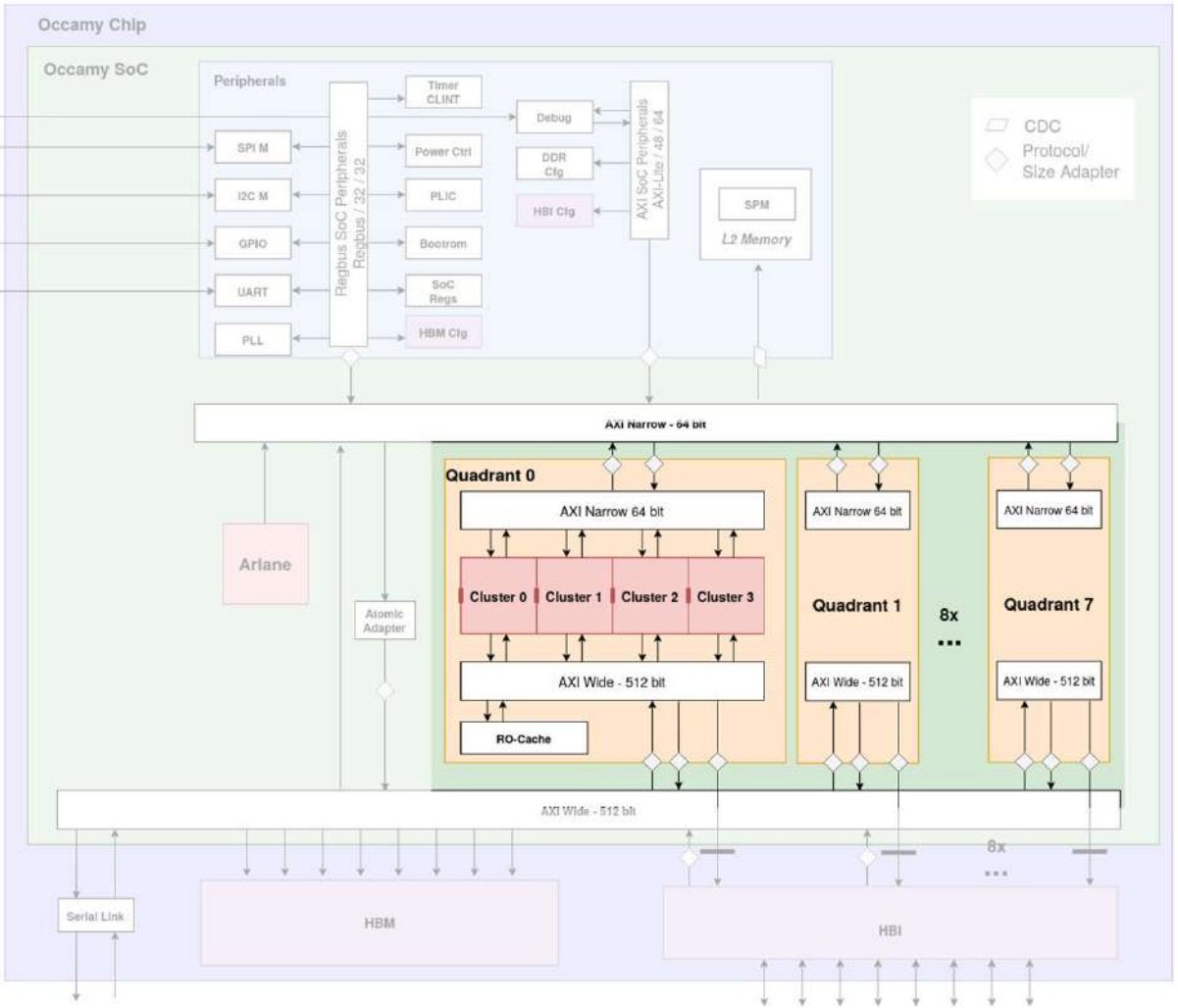
- **Clusters**: *DORY* framework for tiling strategy
- **Chiplets**: E.g. Layer pipelining

Chip-Level Architecture



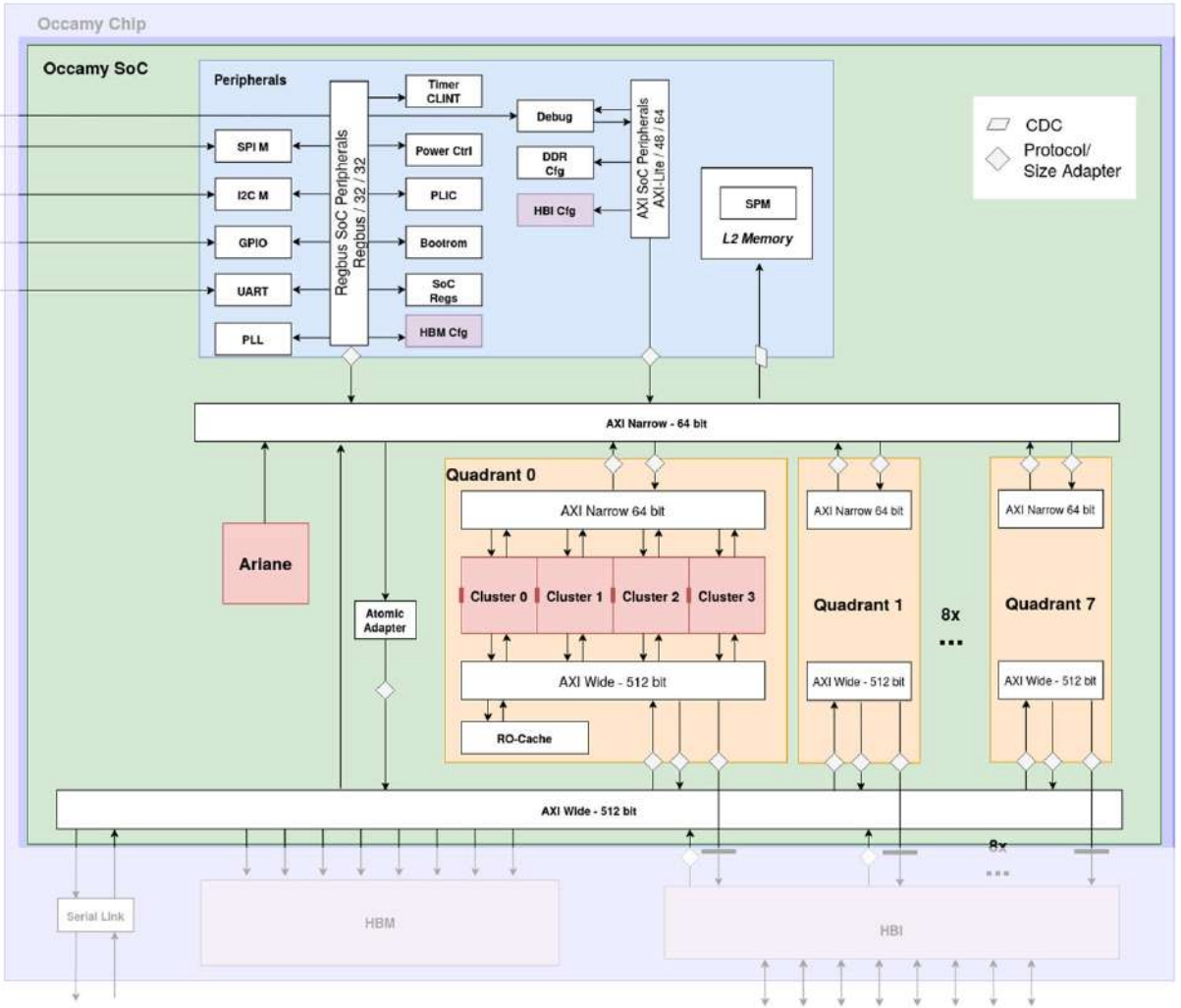
- 32 Snitch Cluster (4 x 8)
  - 64 bit narrow Interface: config
  - 512 bit wide interface: DMA

Chip-Level Architecture



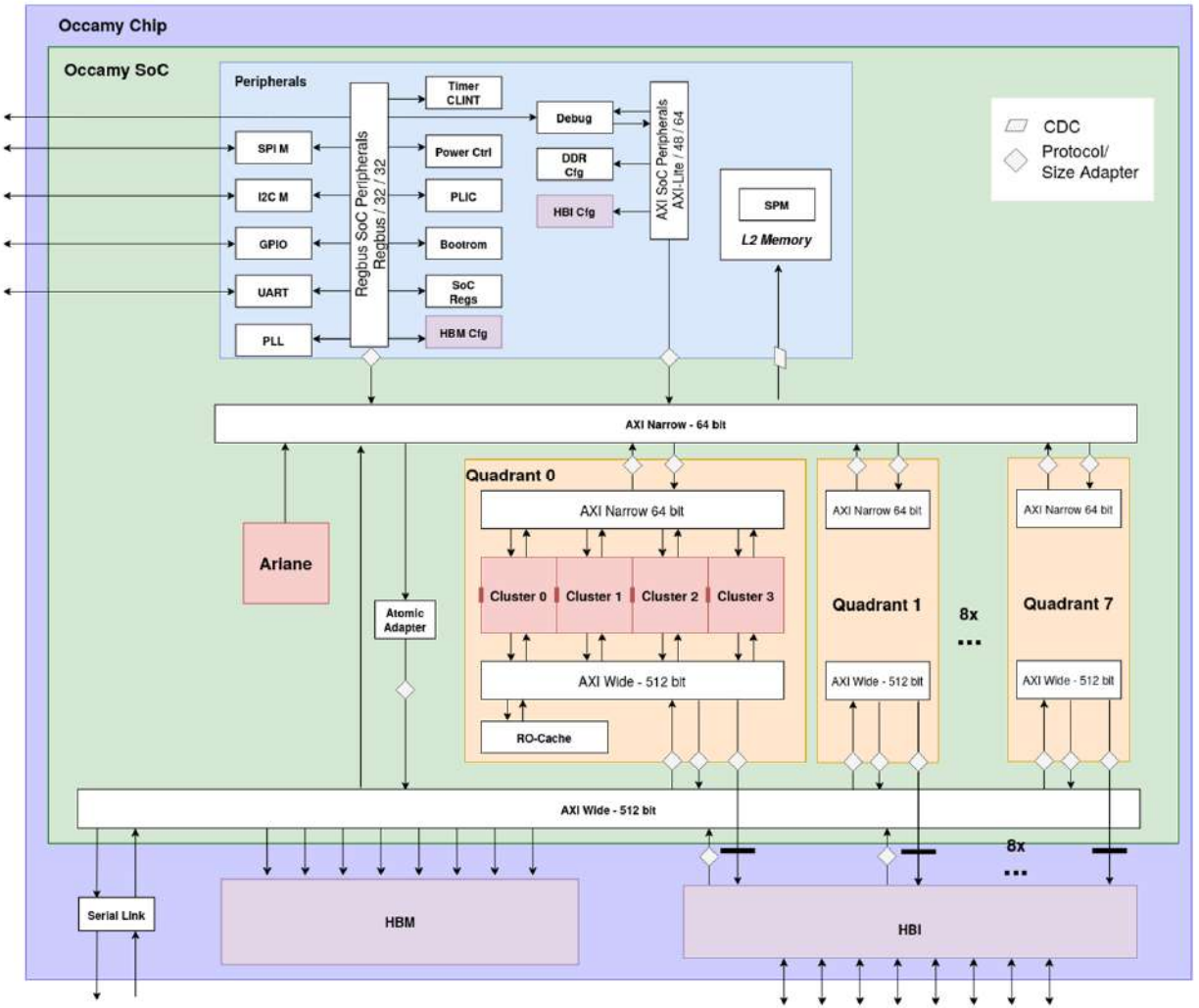
- 32 Snitch Cluster (4 x 8)
  - 64 bit narrow Interface: config
  - 512 bit wide interface: DMA
- 8 Quadrants
  - One level of NoC hierarchy
  - 512 bit RO-cache each
  - Direct master interface to HBI

Chip-Level Architecture



- 32 Snitch Cluster (4 x 8)
  - 64 bit narrow Interface: config
  - 512 bit wide interface: DMA
- 8 Quadrants
  - One level of NoC hierarchy
  - 512 bit RO-cache each
  - Direct master interface to HBI
- Occamy SoC
  - 64 bit narrow X-bar: config, peripherals, and Ariane core
  - 512 bit wide X-bar
  - Peripherals: AXI-Lite or Regbus
  - Only FOS hardware
  - Target: SoC @ 1GHz

Chip-Level Architecture



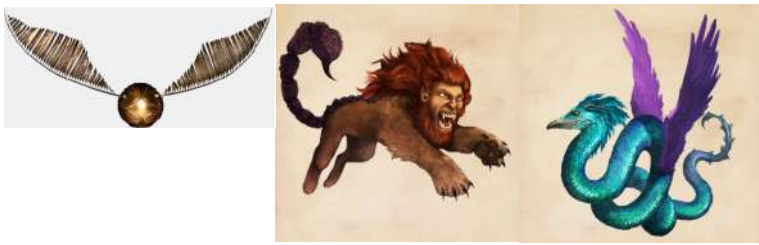
- 32 Snitch Cluster (4 x 8)
  - 64 bit narrow Interface: config
  - 512 bit wide interface: DMA
- 8 Quadrants
  - One level of NoC hierarchy
  - 512 bit RO-cache each
  - Direct master interface to HBI
- Occamy SoC
  - 64 bit narrow X-bar: config, peripherals, and Ariane core
  - 512 bit wide X-bar
  - Peripherals: AXI-Lite or Regbus
  - Only FOS hardware
  - Target: SoC @ 1GHz
- Occamy Chip
  - SoC + non-free IPs

System-Level Architecture (Occamy System)

- Combines 2 Occamy Chips
- 512 Snitch cores
- HBI D2D
  - In-house AXI4 SerDes
  - Optimized for DMA transfers
  - 16 (+ 8 redundant) channels used
  - Achieves full bandwidth
- HBM
  - One memory chip per Occamy chip
- Peripherals

Conclusion

Evolution and future perspectives



- **Snitch** → RISC-V-based, small, fast, latency-tolerant, extensible core → our “atom”
- **Cluster** → Efficient (70% L1+FPU), 1-8cores, L1TCDM+LIC, I$, DMA, SYNC, plugs → our “molecule”
- **Manticore** → Scalable & efficient architecture, NOC, L2 mems, Memory controllers → our “fabric”
- **Occamy** → silicon demonstration of all the key IPs and 2.5D integration
- Flexiblity >> accelerator (e.g. systolic array)
  - Dense and sparse workloads
  - Multi-precision FP support
  - Open, extensible ISA , scales to thousands of PEs
- Utilization ≈ accelerator: 70%+ logic area to execution units, 80%+ utilization
- Efficient implementation is possible: 1GHz (WC) in GF12 with a standard design flow
- **Future**: Larger clusters, 3D archi, compute in Network and near-memory (L2, Main), multi-chiplet scaling …


Thank you!
Gianna Paulin, Luca Bertaccini, Nils Wistoff, Noah Hütter, Paul Scheffler, Samuel Riedel, Thomas Benz, Tim Fischer, Luca Colagrande, Yichao Zhang, Matheus Cavalcante, Andreas Kurth, Florian Zaruba, Stefan Mach, Fabian Schuiki, Zerun Jiang, Beat Muheim, Frank K. Gürkaynak, Luca Benini
pulp-platform.org
@pulp_platform
github.com/pulp-platform/snitch
Thestral V1.0