ACCELERATOR

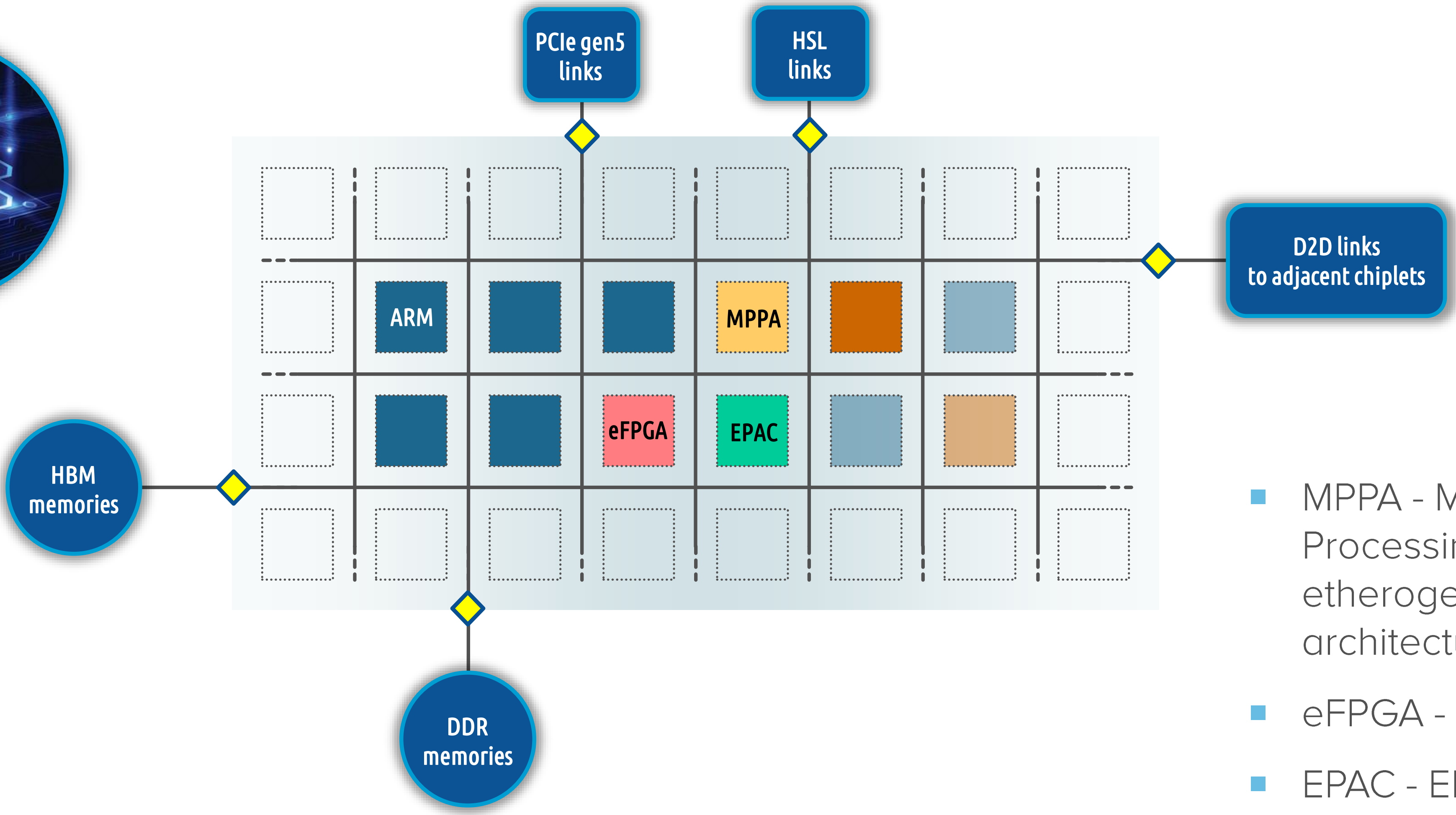
MAURO OLIVIERI

 EPI Tutorial/Barcelona/17-07-2019

ACCELERATOR - SUMMARY

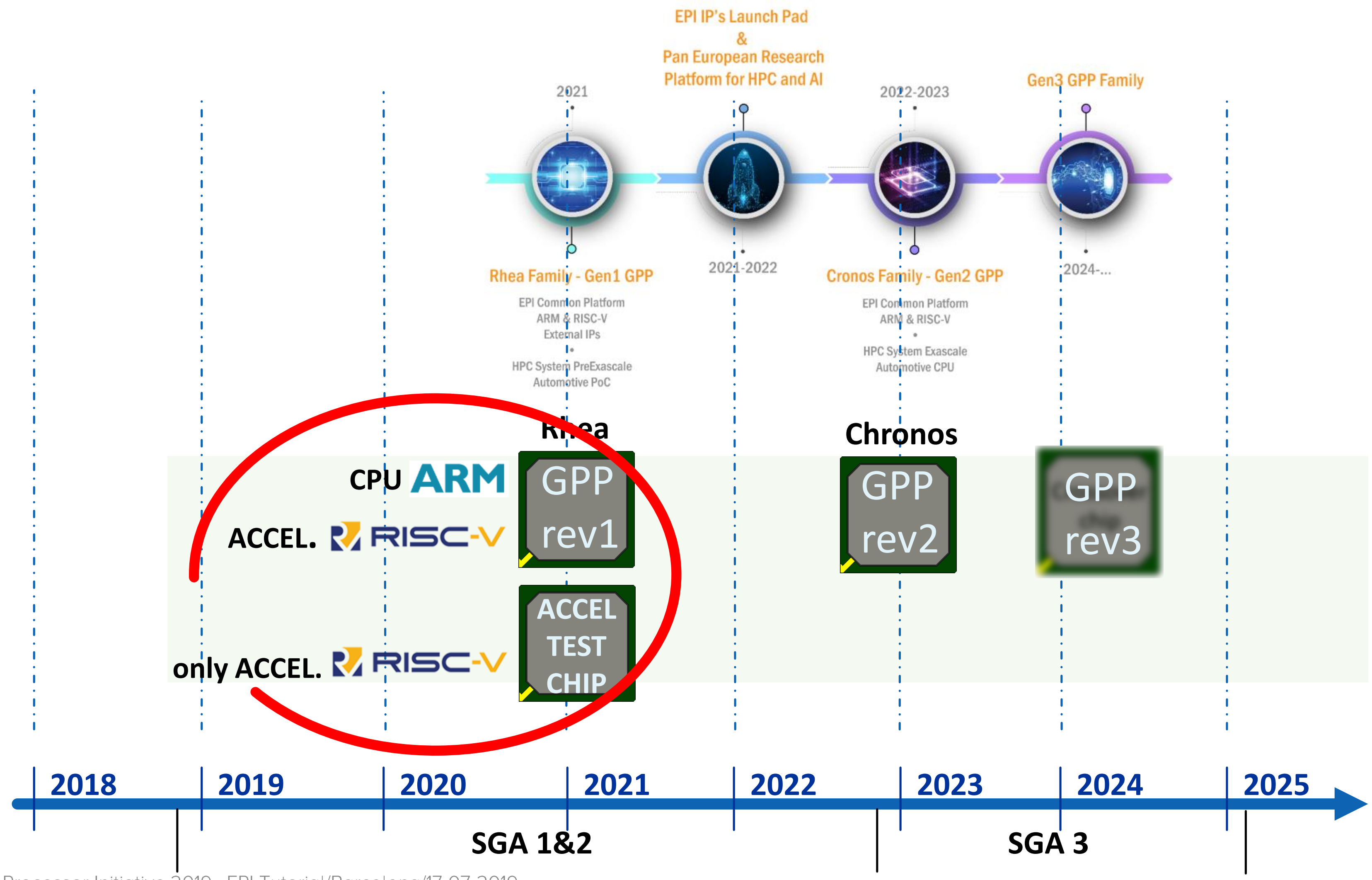
Start	Finish		Presenter
15:45	16:15	**EPI accelerator (EPAc) design**	Mauro Olivieri
		The accelerator roadmap	Contributors
		Vector processor accelerator	Adrian Cristal, Jesus Labarta
		Neural and Stencil processor accelerator	Luca Benini
16:15	16:30	**Q & A**	

RECALL… THE GPP AND COMMON ARCHITECTURE

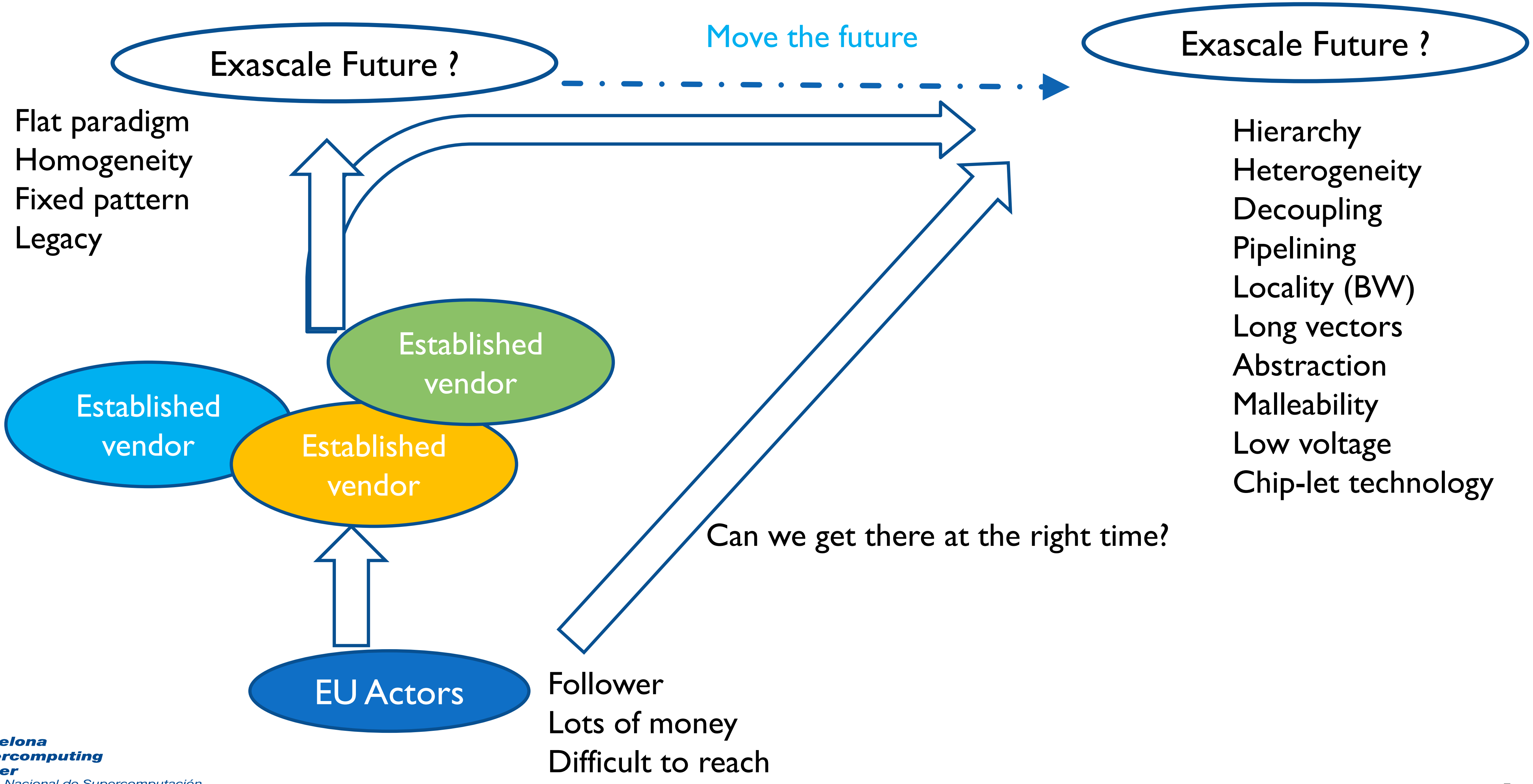


- MPPA - Multi-Purpose Processing Array etherogeneous architecture
- eFPGA - embedded FPGA
- EPAC - EPI Accelerator

ACCELERATOR STREAM IN EPI ROADMAP



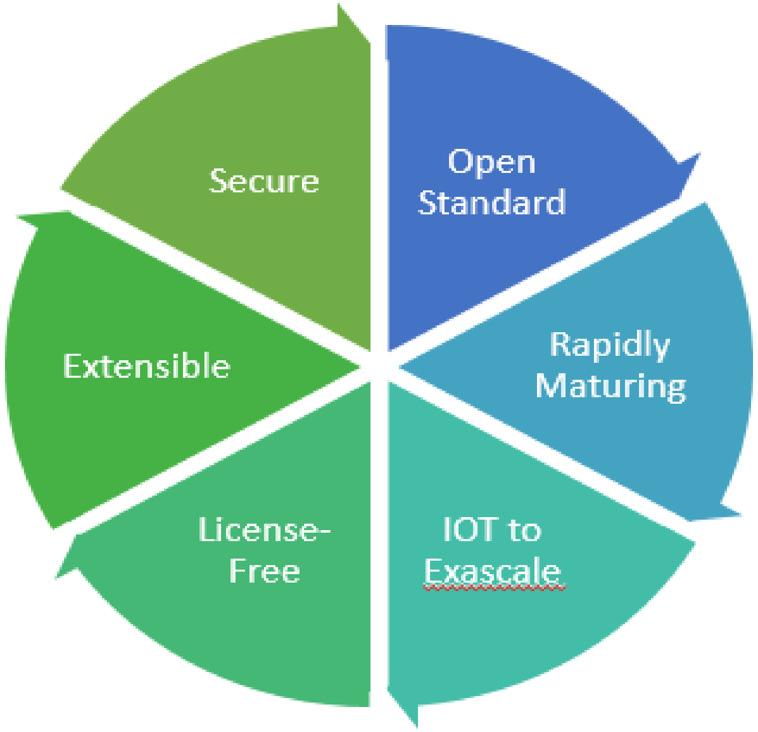
BSC'S VISION OF THE EXASCALE TRANSITION



ABOUT THE INSTRUCTION SET FOR THE ACCELERATOR

- In 2015, Mateo Valero said he believed a European Supercomputer based on ARM was possible (Mont-Blanc).
- Even though ARM is no longer European, it can form part of the short-term solution
- The fastest-growing movement in computing at the moment is Open-Source and is called RISC-V
- The future is Open and RISC-V is democratizing chip-design
- Very-high-performance general purpose RISC-V CPUs are not available yet. But the time is right to develop powerful RISC-V accelerator processors

RISC-V



RISC-V

THE ADVANTAGES....

- RISC-V has no legacy constraints
- RISC-V has (standard and non-standard) extensions
- RISCV extensions facilitate decoupling from hardware (avoid low level accelerator control, e.g. memory mapped)
- RISC-V is contributed by committees of experts doing a great job

THE LIMITATIONS....

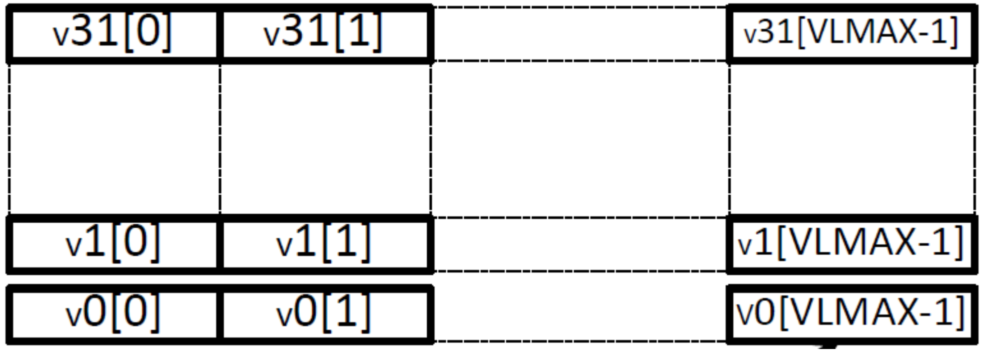
- Many people's contributions = many different requirements
- Some features may result less important to some members
- The eco-system issue (but smaller than – for example – in consumer market)
- The political issue question

RISC-V VECTOR EXTENSION

https://github.com/riscv/riscv-v-spec/

European Processor Initiative

32 vector registers



Main features and operations:

- Orthogonal set of vector operations, parity with scalar ISA
- Rich set of integer, fixed-point, and floating-point instructions
- Vector-vector, vector-scalar, and vector-immediate instructions
- Masking on (almost) every vector instruction
- Non-strided and strided loads and stores, gathers, scatters
- Reduction instructions (sum, min/max, and/or, ...)

Vector Control-Status registers:

Vtype

Vtype sets width of element in each vector register (e.g., 16-bit, 32-bit, ...)

Vl

Vector length CSR sets number of elements active in each instruction

Vstart

Resumption element after trap

fcsr

Fixed-point rounding mode and saturation flag fields

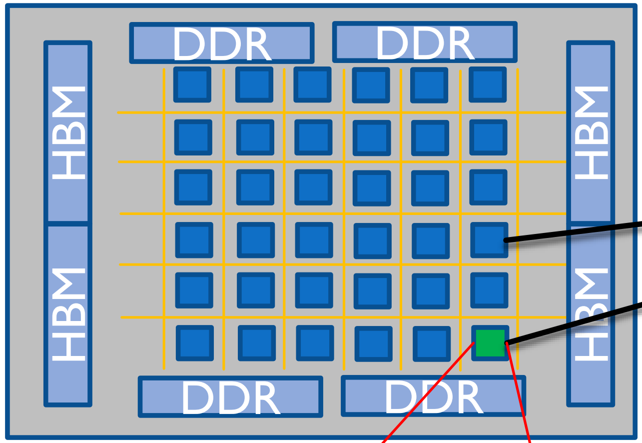
RISC-V VECTOR EXTENSION

https://github.com/riscv/riscv-v-spec/

Example vector instruction

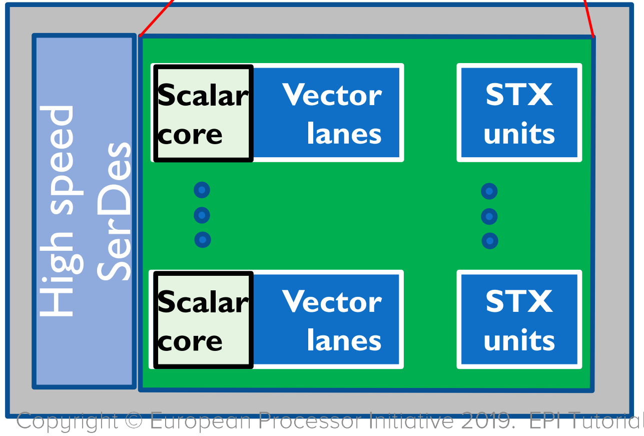
- **vfadd.vv vd, vs1, vs2 , vm**
  - Adds two vector registers, element by elements, and puts result into destination vector register
- **vsetvli rd, rs1, vtypei**
  - Sets vector length VL and element width and type VTYPE
- **vle.v vd, (rs1), vm**
  - Loads a vector from memory into destination vector register, unit-strided
- **vse.v vs3, (rs1), vm**
  - Stores a vector from source vector register to memory, unit-strided

1ST GENERATION EPI CHIPS



General Purpose Processor (GPP) chip

- 7 nm, chip-let technology
- ARM-SVE tiles
- EPAC RISC-V vector+AI accelerator tiles
- L1, L2, L3 cache subsystem + HBM + DDR

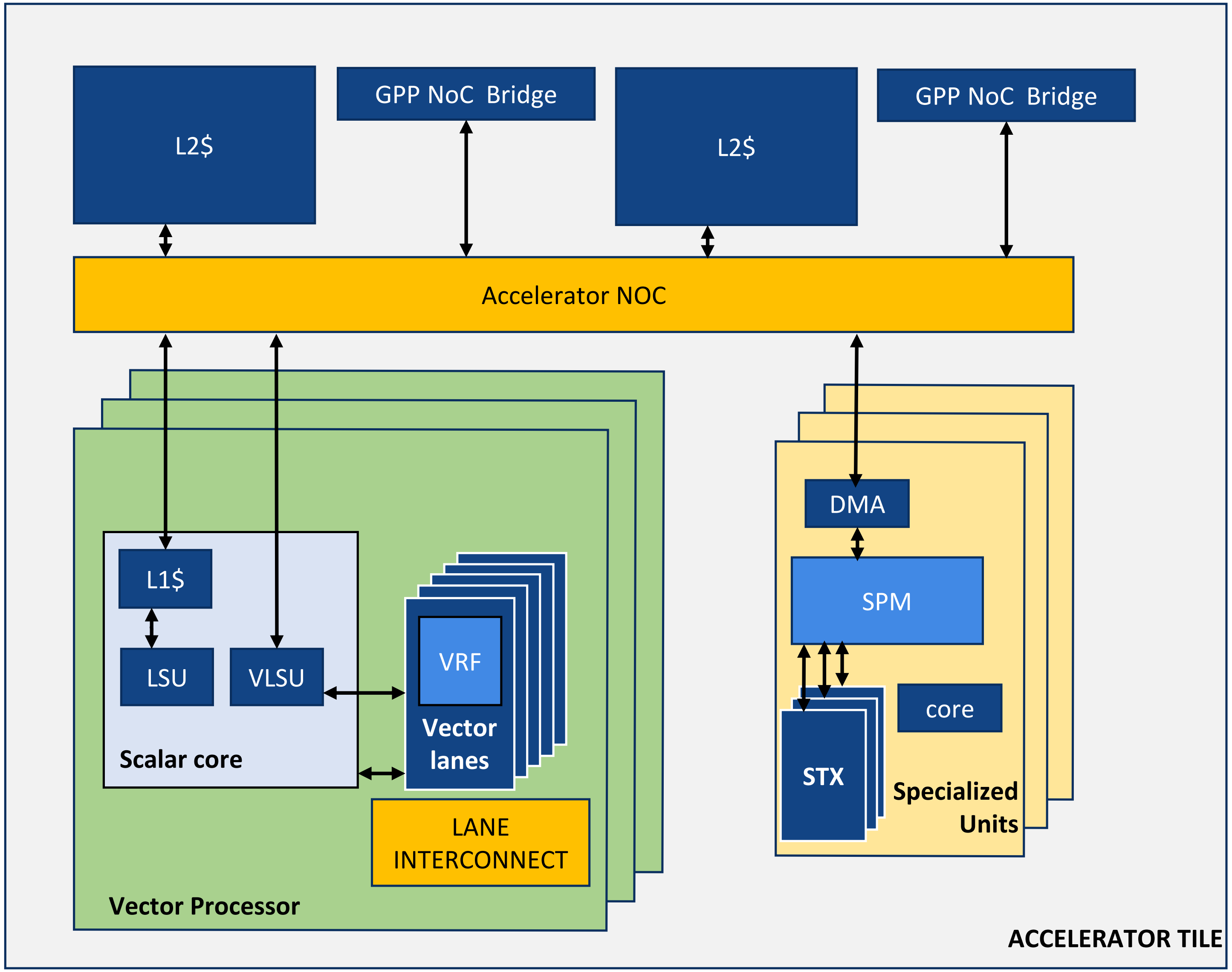


RISC-V Accelerator Demonstrator Test Chip

- 22 nm FDSOI
- Only one RISC-V accelerator tile
- On-chip L1, L2 + off-chip HBM + DDR PHY
- Targets 128 DP GFLOPS (vector processor only)

EPAC ARCHITECTURE VIEW

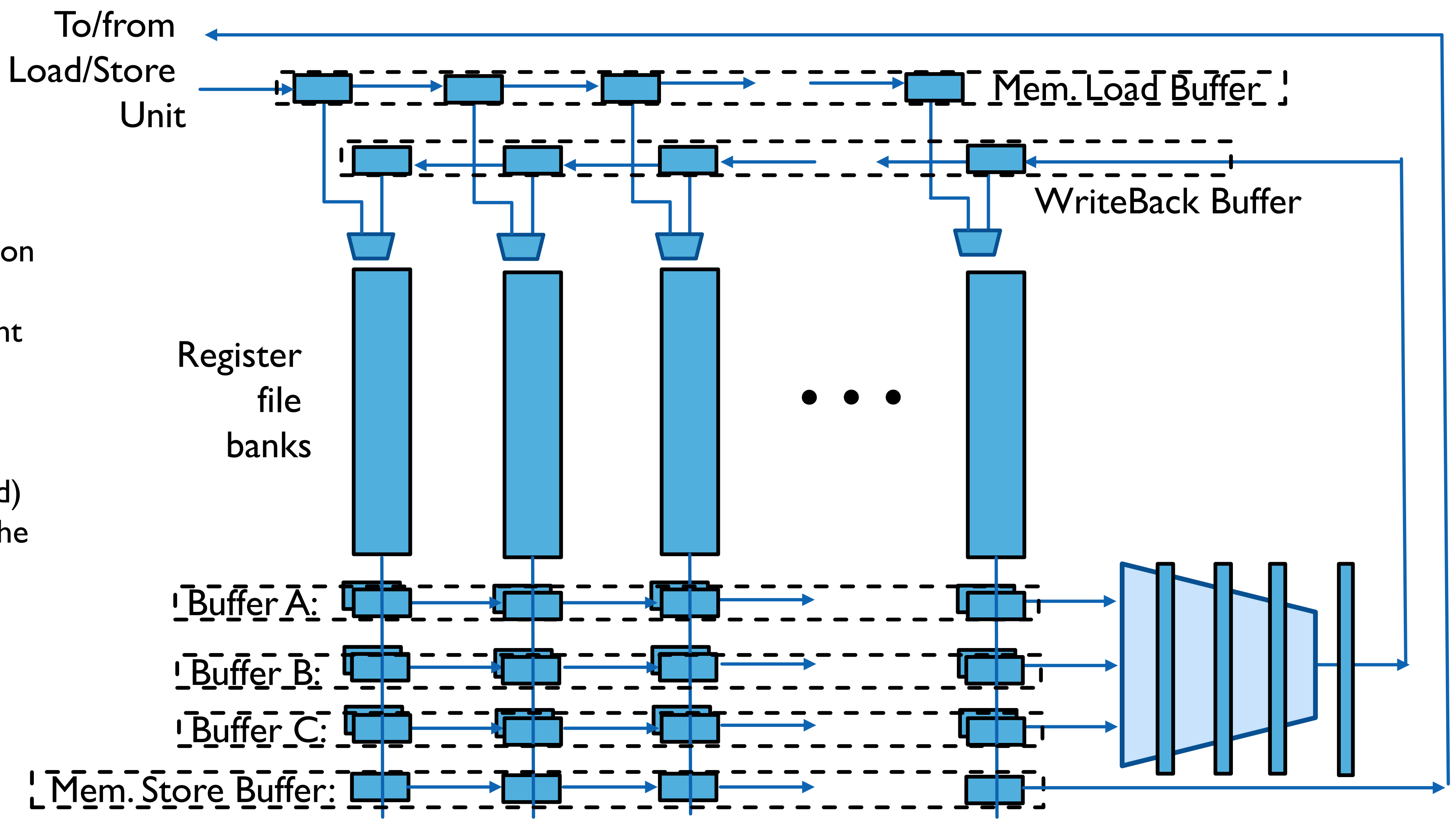
European Processor Initiative



- Up to 8 vector processors per tile
- The Vector Lanes act as tightly coupled (ISA mapped) acceleration units to the scalar core in the vector processor
- Heavily pipelined
- RISC-V vector extension compliant
- Up to 8 Specialized Units per tile
- The STX Units act as loosely coupled, memory mapped acceleration units to the scalar cores
- Fast single-cycle MACs in parallel
- Shared L2 cache banks
- Cache coherent NoC

VECTOR LANE MICROARCHITECTURE (SIMPLIFIED VIEW)

Buffer A, B, C: operand buffers,
WriteBack Buffer: holding operation results
Store Buffer: holding data to be sent to memory
Load buffer: holding data coming from memory
Buffers A,B,C are doubled (shadowed) to allow single-cycle full refill while the shadow buffer is being consumed.

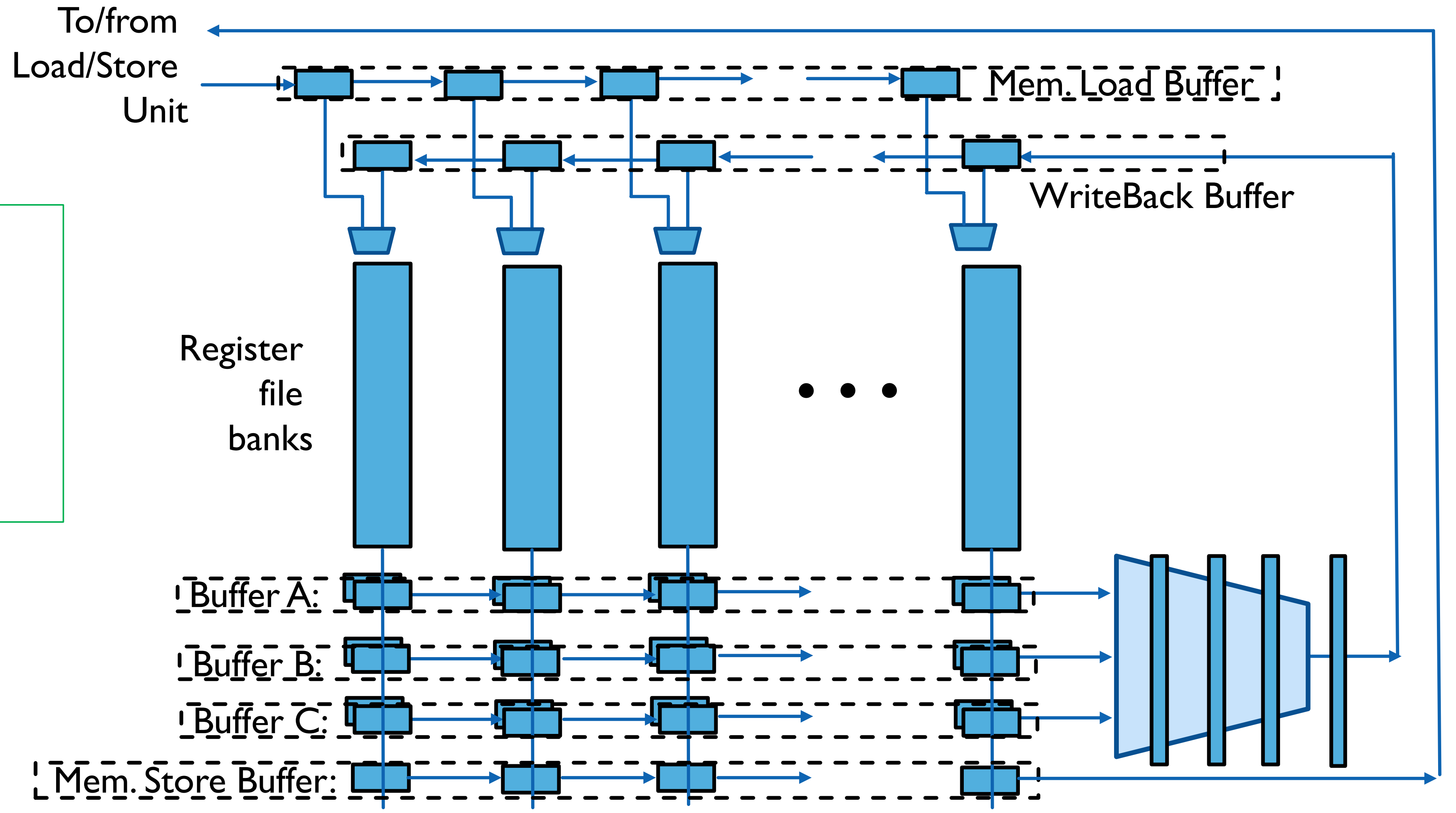


VECTOR ARITHMETIC OPERATION EXECUTION

vfadd.vv v0, v1, v2

```
for (i = 0; i < VL; ++i)
    v0[i] = v1[i] + v2[i];

v0[VL ...VLMAX] = 0;
```



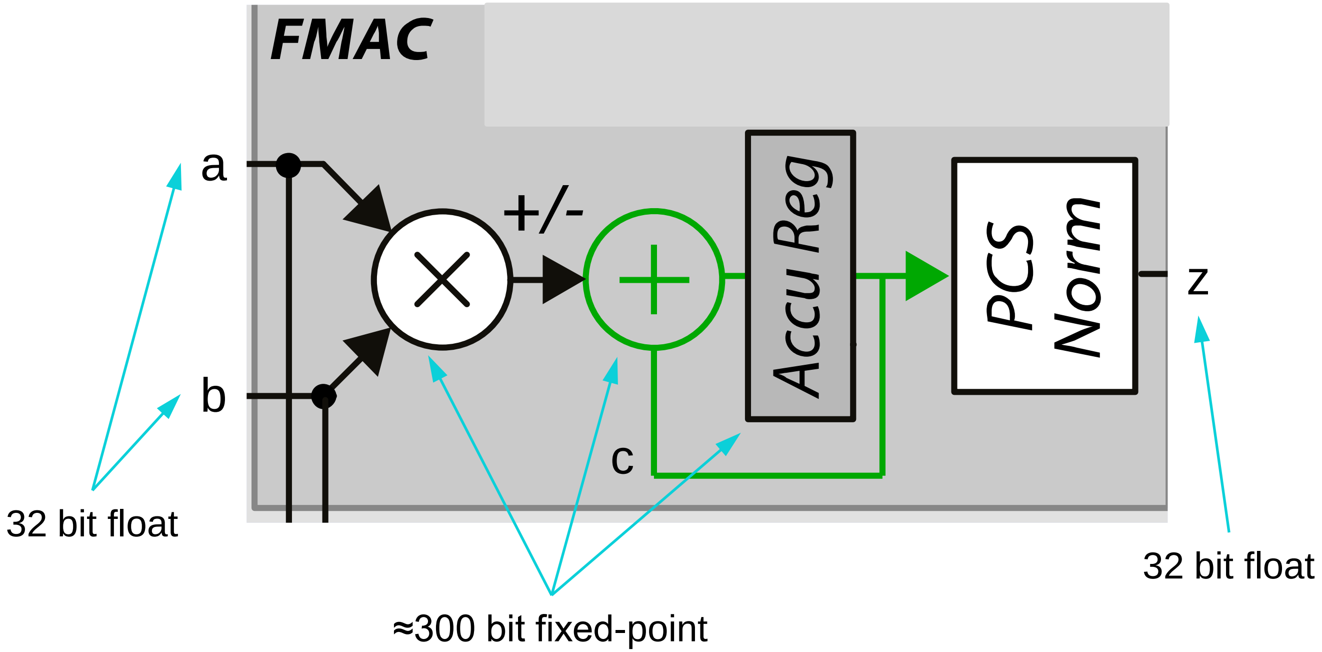
STENCIL AND NEURAL TRAINING ACCELERATOR (STX)

- Reference published design: STX [1]
- Streaming floating-point co-processor
- Efficiently performs float32 FMAC
  - Fast multiply-accumulate, single cycle
- Address generation unit ensures low control overhead
  - 5 nested hardware loops
  - 3 address generators
- Many common C/C++ loop nests map well to this architecture
- 8 STX paired up per associated processor core
- Floating-point operation makes accelerator a drop-in replacement for GPUs for training.

[1] Schuiki et al, "A Scalable Near-Memory Architecture for Training Deep Neural Networks on Large In-Memory Datasets,", in IEEE TC 2019.

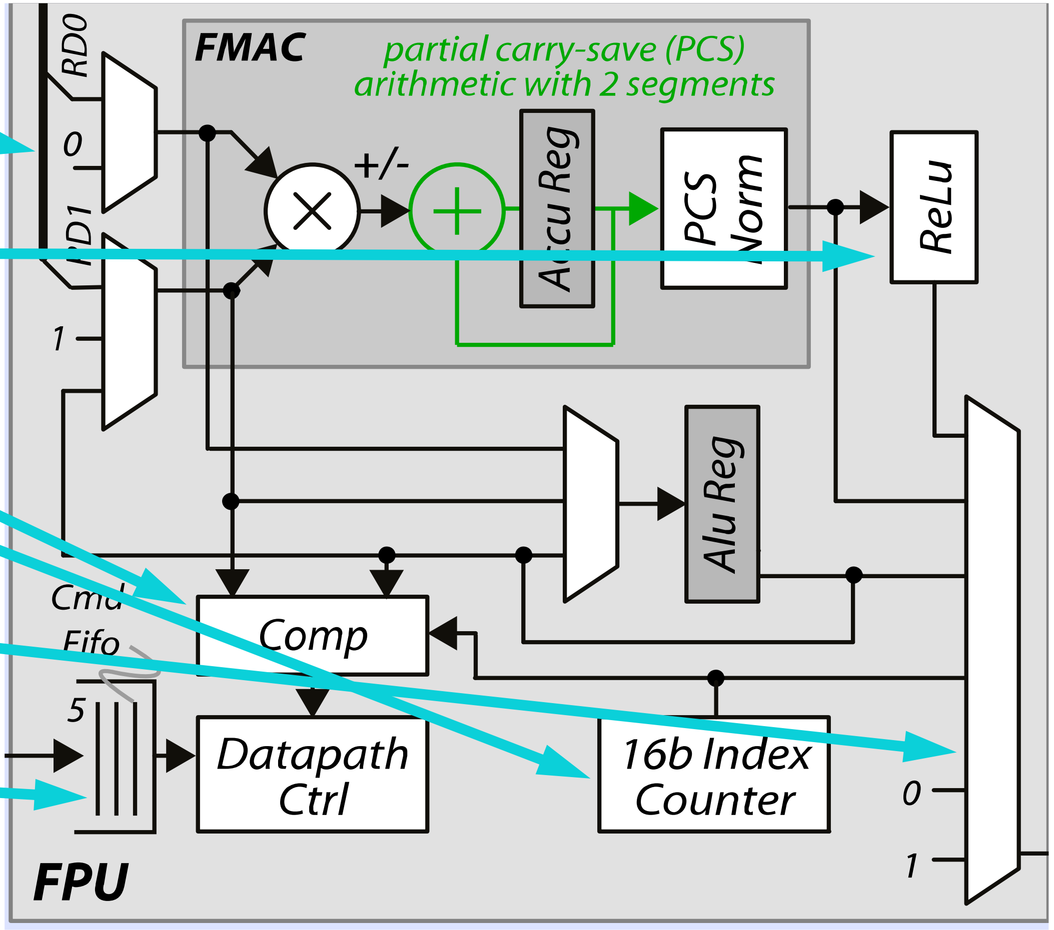
MICROARCHITECTURE OF BASIC FMAC UNIT

- Main data path is a **single-cycle partial carry save FMA**
- Expansion of float operands to fixed-point
- Multiplication and addition in fixed-point
  - Single-cycle
  - Tuneable performance by increasing number of partial sums
- Conversion to float after accumulation
  - Partial sums are accumulated
  - Conversion from fixed-point to float



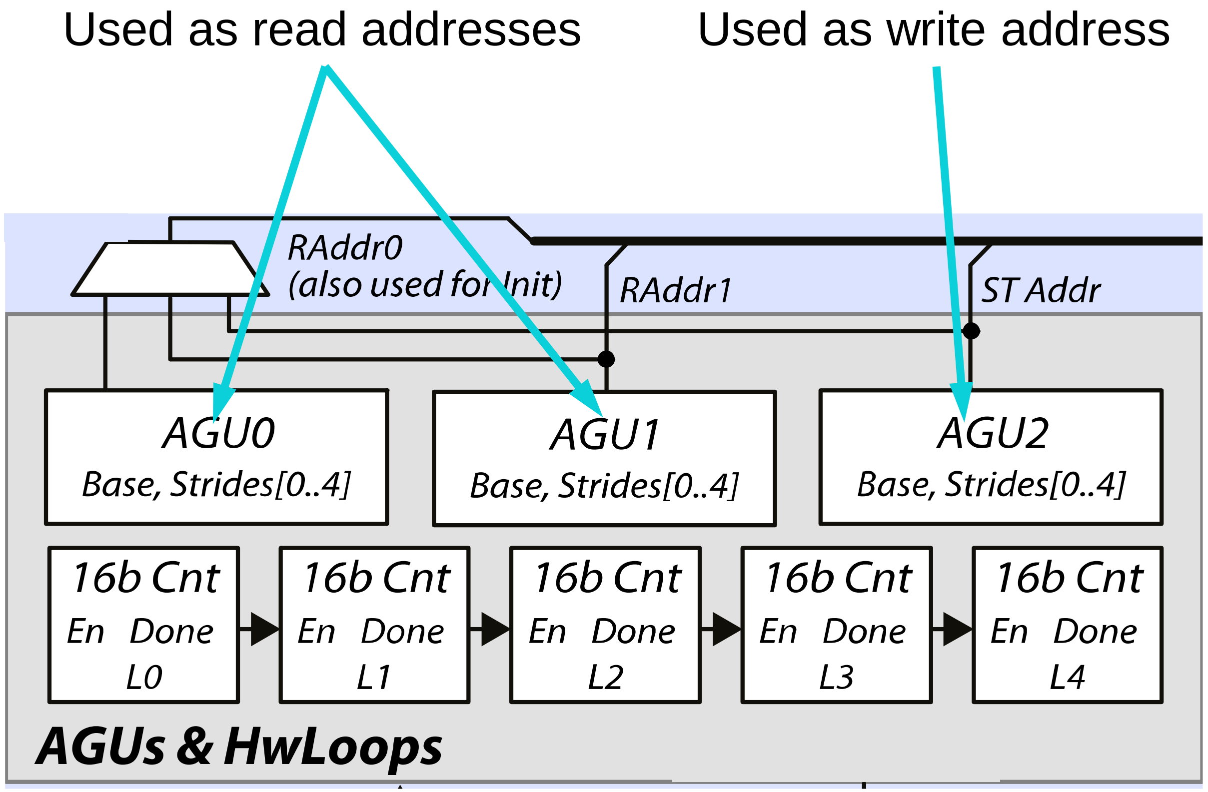
MICROARCHITECTURE OF THE STX DATA PATH

- FMA operands arrive as **memory streams**
  - Maskable to 0/1 to disable add/mul
- Optional **ReLU** on FMA result
- Comparator for finding **max/min**
- Index counter for finding **argmax/argmin**
  - Enables maxpool derivatives
- Output can be **masked** to 0/1
  - Enables ReLU derivatives
- Fire-and-forget datapath
  - Command pushed into FIFO
  - Consumes fixed number of input items
  - Produces fixed number of output items



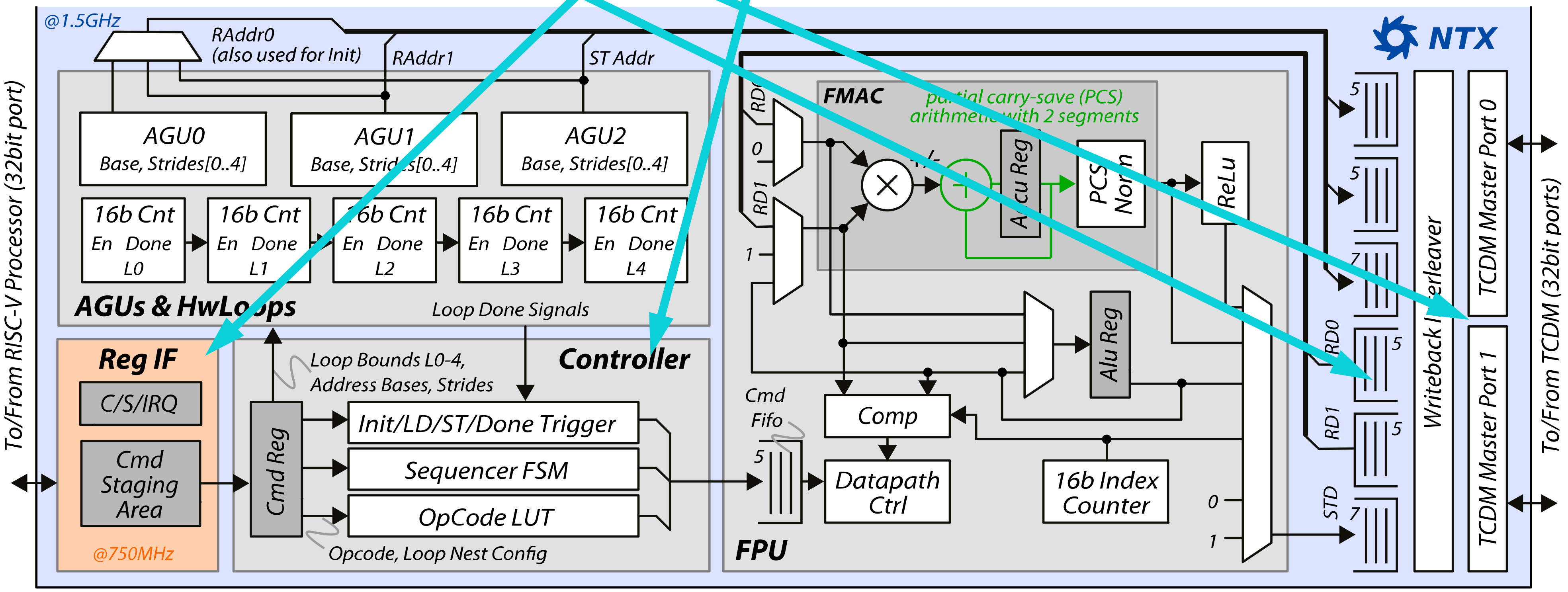
MICROARCHITECTURE OF ADDRESS GENERATION UNIT

- 5 nested hardware loop counters
  - 16 bit counter register
  - Configurable number of iterations
  - Once last iteration reached:
    - Reset counter to 0
    - Enable next counter for one cycle
- 3 address generation units
  - 32 bit address register
  - Each has 5 configurable strides, one per loop
  - One stride added to register per cycle
  - Stride corresponds to the highest enabled loop



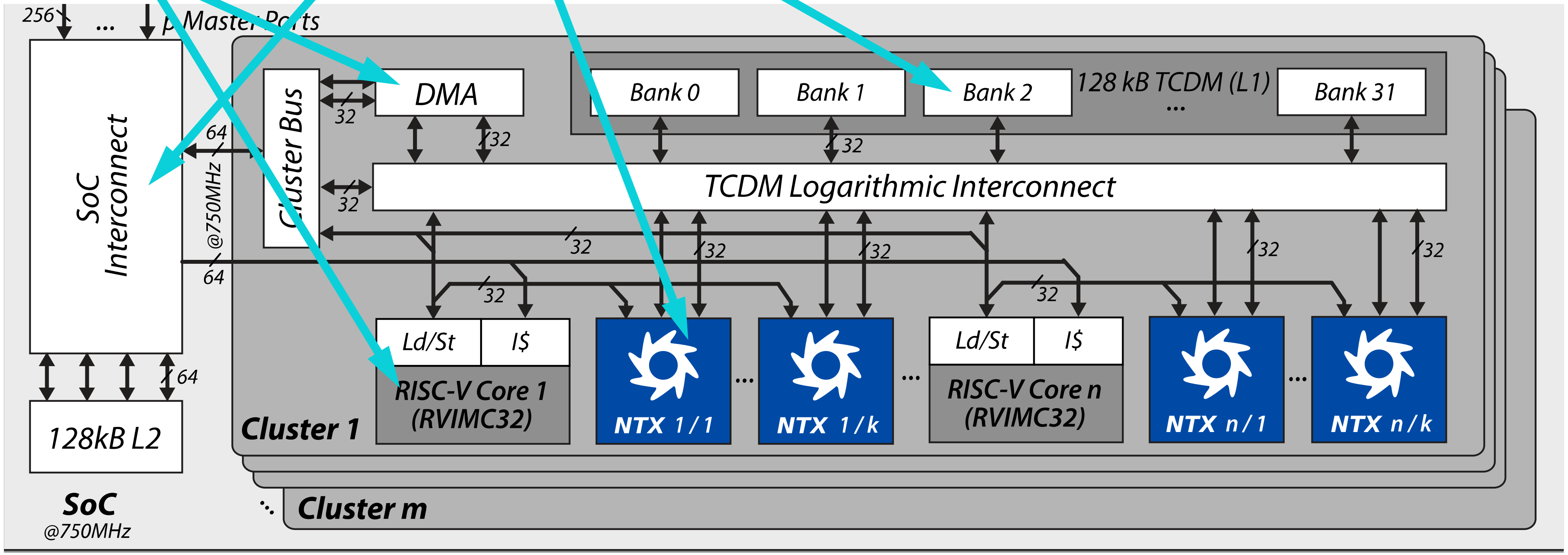
OVERALL STX COPROCESSOR MICROARCHITECTURE

- Processor configures operation via **memory-mapped registers**
- Controller issues AGU, HWL, and FPU micro-commands based on configuration
- Reads/writes data via **2 memory ports** (2 operand and 1 writeback streams)
- FIFOs help buffer data path and memory latencies



ARCHITECTURE PROCESSING CLUSTER

- 1 local processor core controls 8 STX coprocessors
- Attached to 128 kB shared SPM via a logarithmic interconnect
- DMA engine used to transfer data (double buffering)
- Multiple clusters connected via interconnect (crossbar/NoC)



- No explicit load/store instructions
- No explicit address calculation instructions

- Simple example: Dot product over 1024 elements
- With single RV32IF:
  - 5122 instructions executed
- With single STX (plus RV32I):
  - 10 instructions executed
  - 1024 idle cycles while STX executes (can be used)
- STX reduces instruction bandwidth by 512x
  - Even more when using more nested loops
- STX amortizes single instruction stream over 8 FPUs
  - Data/Inst. bandwidth ratio of 16 (worst case, usually higher)

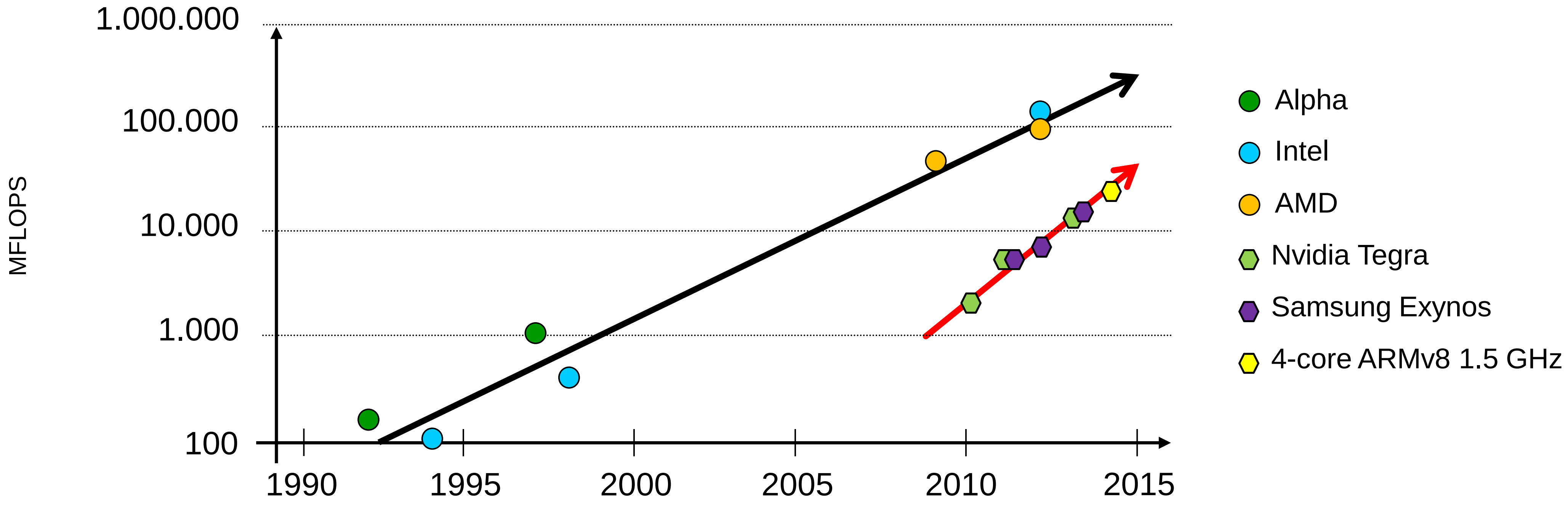
Single RV32IF Core:

```
Setup     — lp.setupi L0, 1024, 5
          ┌ flw       ft0, 0(a0)
          │ flw       ft1, 0(a1)
Hot Loop  ┤ fmadd     ft2, ft0, ft1, ft2
          │ addi      a0, a0, 4
          └ addi      a1, a1, 4
Writeback — fsw       ft2, 0(a2)
```

Single STX:

```
          ┌ sw  a0, STX_AGU0_PTR
          │ sw  a1, STX_AGU1_PTR
          │ sw  a2, STX_AGU2_PTR
          │ li  t1, 1024
          │ sw  t1, STX_BOUND_L0
Setup     ┤ li  t1, 4
          │ sw  t1, STX_AGU0_S0
          │ sw  t1, STX_AGU1_S0
          │ li  t1, STX_MAC_CMD
          └ sw  t1, STX_CMD
Idle      — wfi
```

ABOUT THE INSTRUCTION SET



- General purpose CPUs killed Vector processors
  - They were not faster …
  - … but they were significantly cheaper

- History may be about to repeat itself …
  - Mobile processor are not faster …
  - … but they are significantly greener

  - What next?