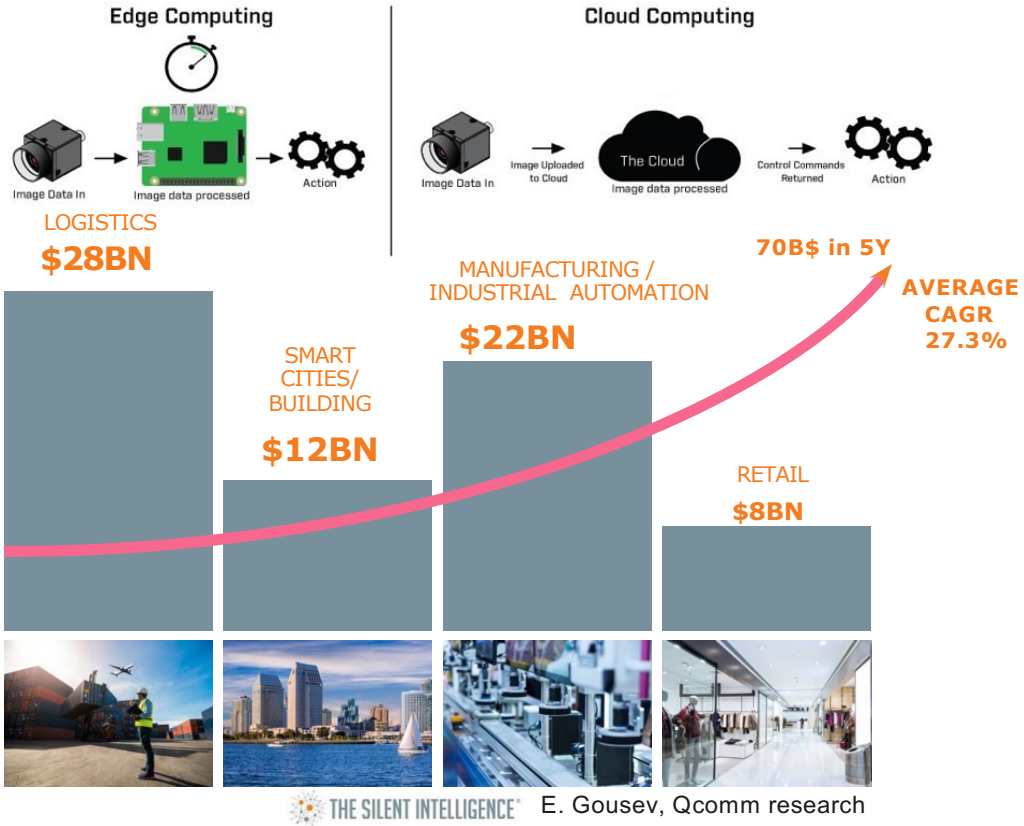
PULP: Embedding AI at the Extreme Edge of the IoT

Yvan Tortorella

yvan.tortorella@chips.it

EFCL Winter School 2026

Cloud → Edge → Near-Sensor Computing



#1 Customer Question on Amazon.com (out of 1,000+):

1. *I don't want any of my (private, personal) videos on any servers not in my control. Is this possible?*

Source: www.amazon.com/ask/questions/asin/B01M3VHG87/

#2 Customer Question on Amazon.com (out of 1,000+):

2. *How long does the battery charge last?*

Source: www.amazon.com/ask/questions/asin/B01M3VHG87/

Near-Sensor Computing challenge

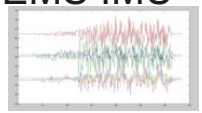
AI capabilities in the power envelope of an MCU:
100mW peak (10mW avg)

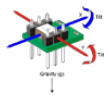
IoT End-Nodes Challenge

Chips-IT

Sense

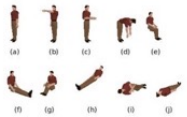
MEMS IMU



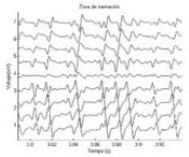


MEMS Microphone

ULP Imager



EMG/ECG/EIT



100 µW ÷ 2 mW

Analyze and Classify

µController

e.g. CortexM

IOs

1 ÷ 25 MOPS
1 ÷ 10 mW

Battery + Harvesting powered
→ a few mW power envelope

Transmit

Short range, medium BW

Low rate (periodic) data

SW update, commands

Long range, low BW

LoRa SEMTECH

Idle: ~1µW
Active: ~ 50mW

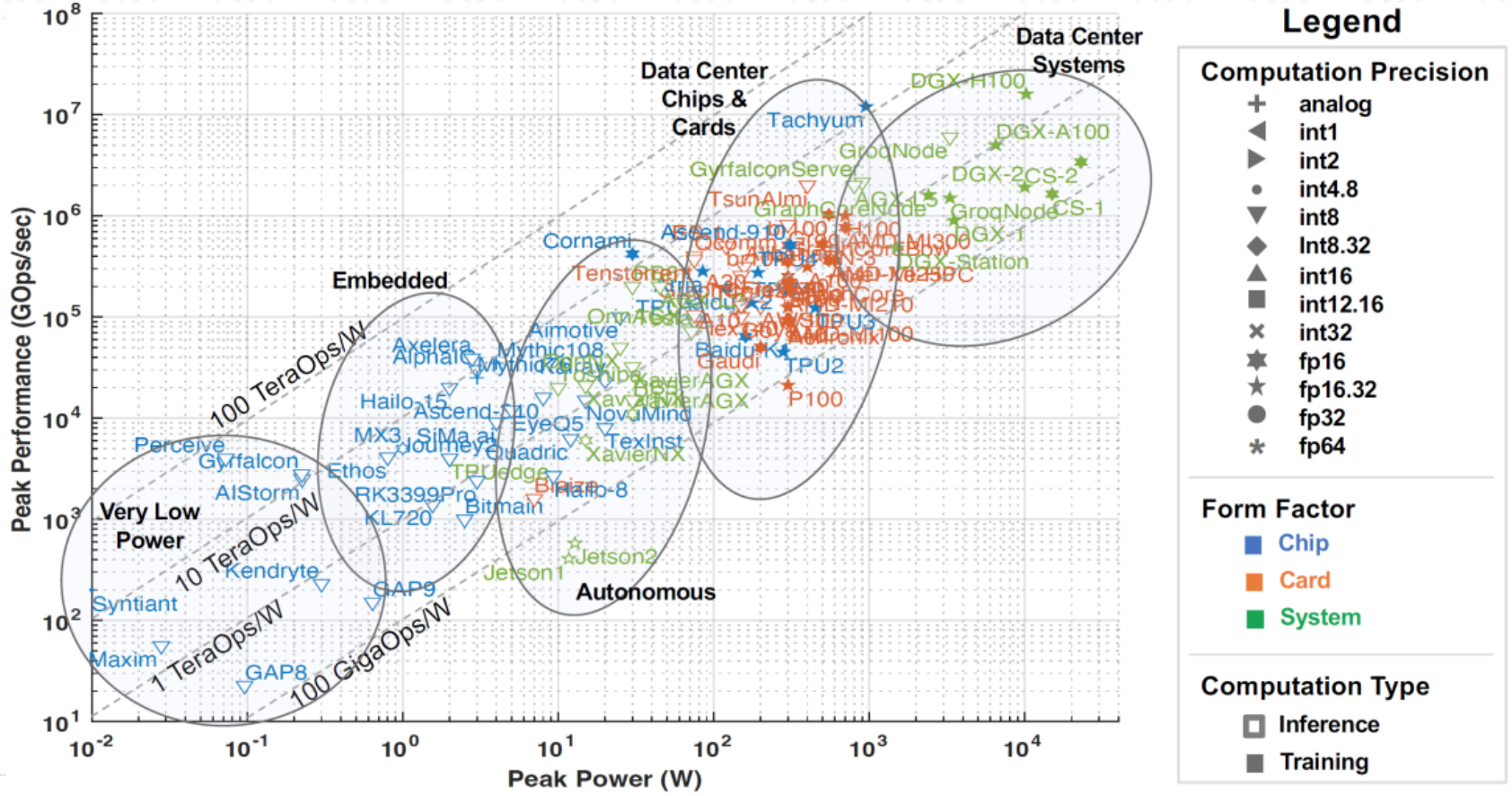
Near-Sensor Analytics Applications (NSAA)

Chips-IT

	INPUT BANDWIDTH	COMPUTATIONAL DEMAND	OUTPUT BANDWIDTH	COMPRESSION FACTOR
Image Tracking: [*Lagroce2014]	80 Kbps	1.34 GOPS	0.16 Kbps	500x
Voice/Sound Speech: [*VoiceControl]	256 Kbps	100 MOPS	0.02 Kbps	12800x
Inertial Kalman: [*Nilsson2014]	2.4 Kbps	7.7 MOPS	0.02 Kbps	120x
Biometrics SVM: [*Benatti2014]	16 Kbps	150 MOPS	0.08 Kbps	200x

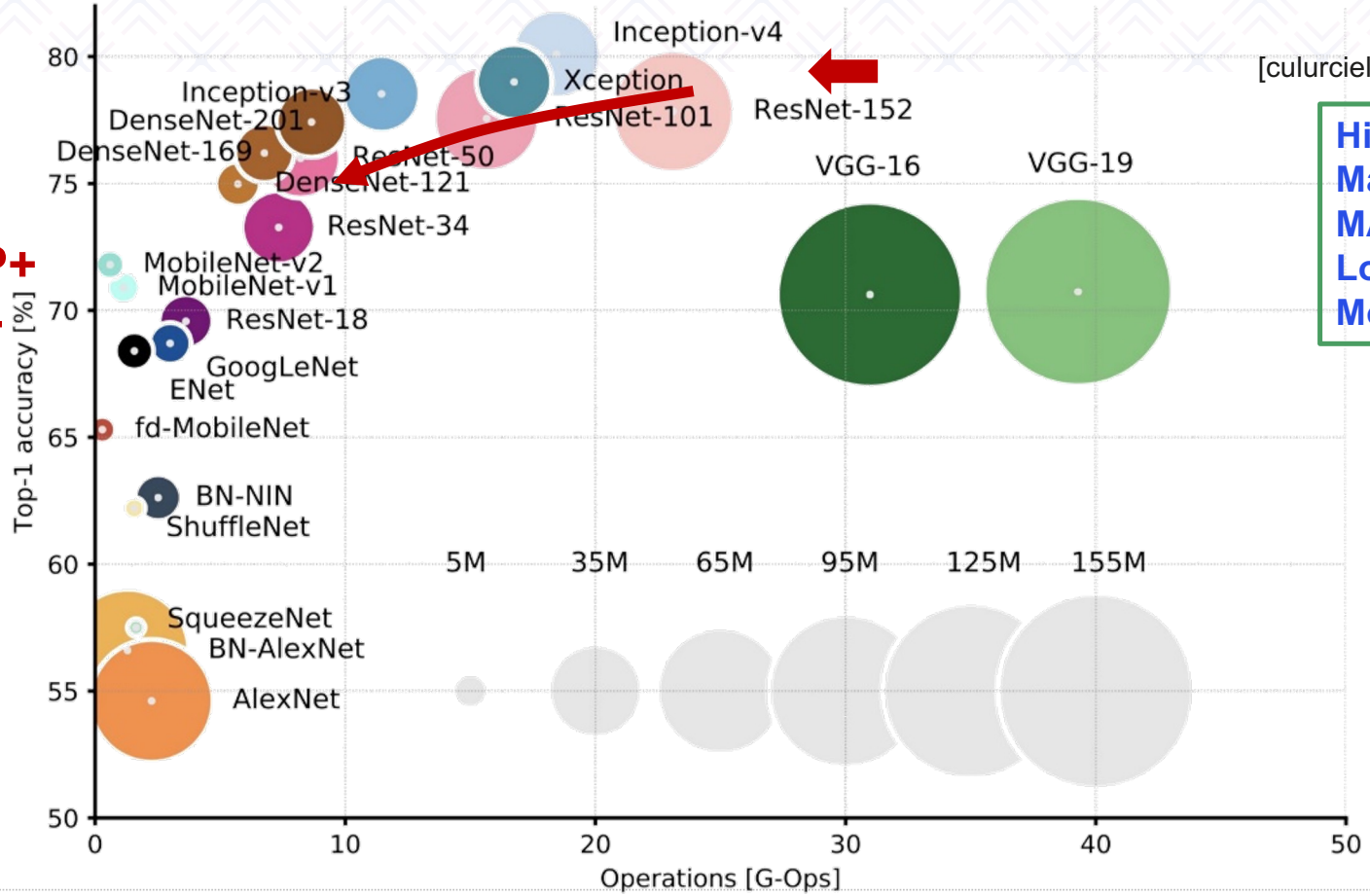
- ***Extremely compact output (single index, alarm, signature)***
- ***Computational power of ULP µControllers is not enough***
- ***Parallel worloads***

ML Processors from Tiny to Huge



AI Workloads from Cloud to Edge (Extreme?)

Chips-IT



GOP+
MB+

[culurciello18]

High OP/B ratio
Massive Parallelism
MAC-dominated
Low precision OK
Model redundancy

CloudML → TinyML

Chips-IT

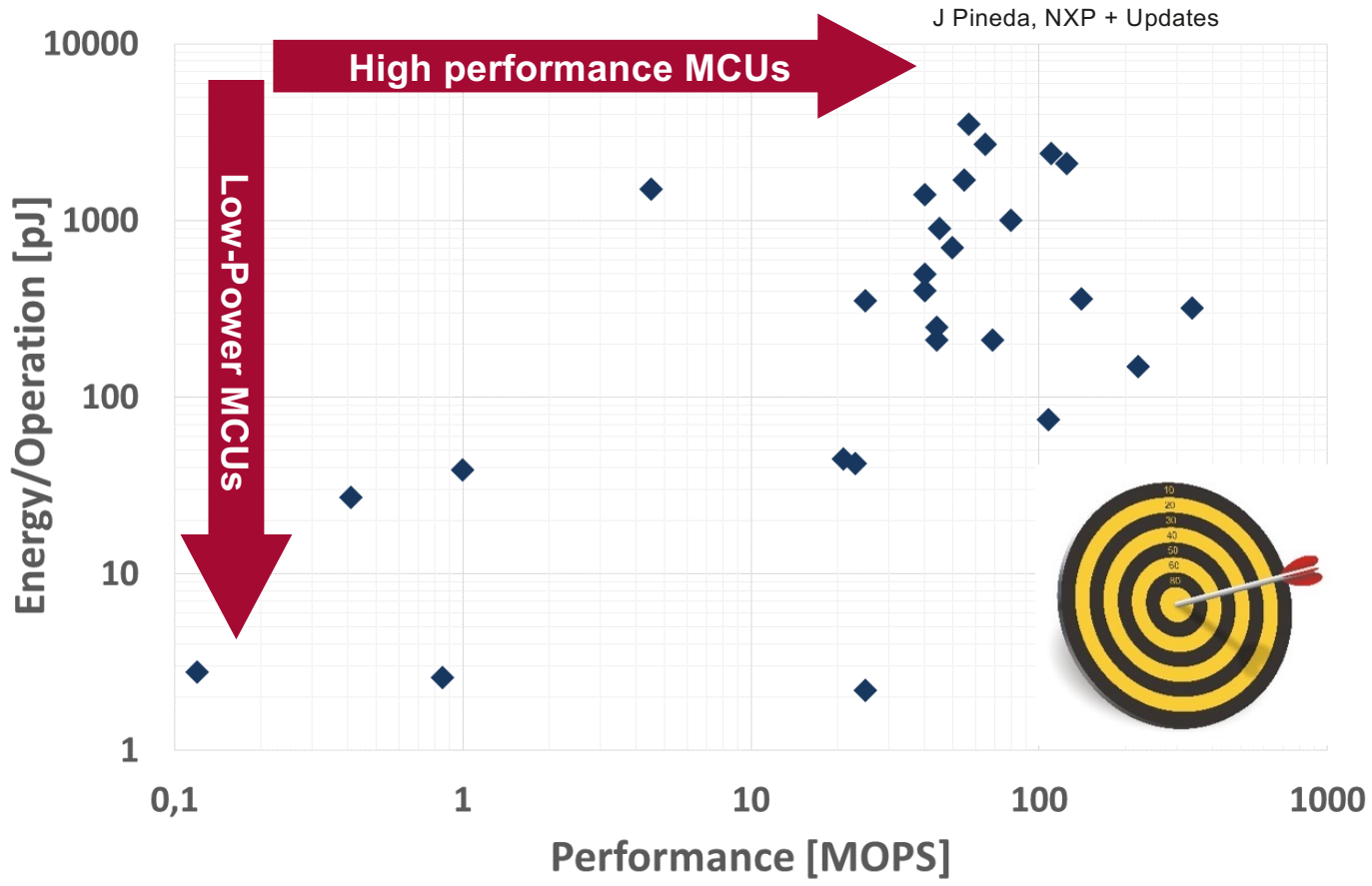
TinyML Opportunity

Anomaly Detection	Voice	Vision
✓ Sensor data analytics ✓ Classify states or behavior with sensors ✓ Predict potential failure behavior early ✓ Smartwatch, predictive maintenance, etc	✓ Keyword spotting – wake word ✓ Voice commands	✓ Image classification ✓ Face recognition ✓ Object detection ✓ Personalization based on face recognition

TinyML challenge

AI capabilities in the power envelope of an MCU: **10-mW peak (1mW avg)**

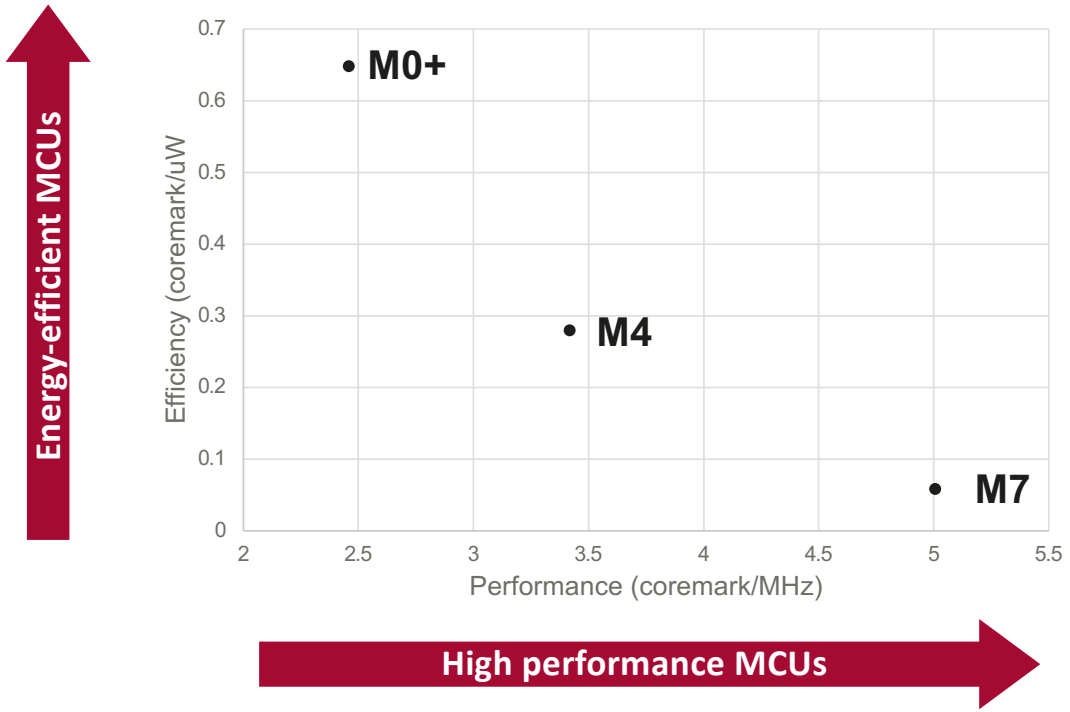
Energy efficiency @ GOPS is THE Challenge



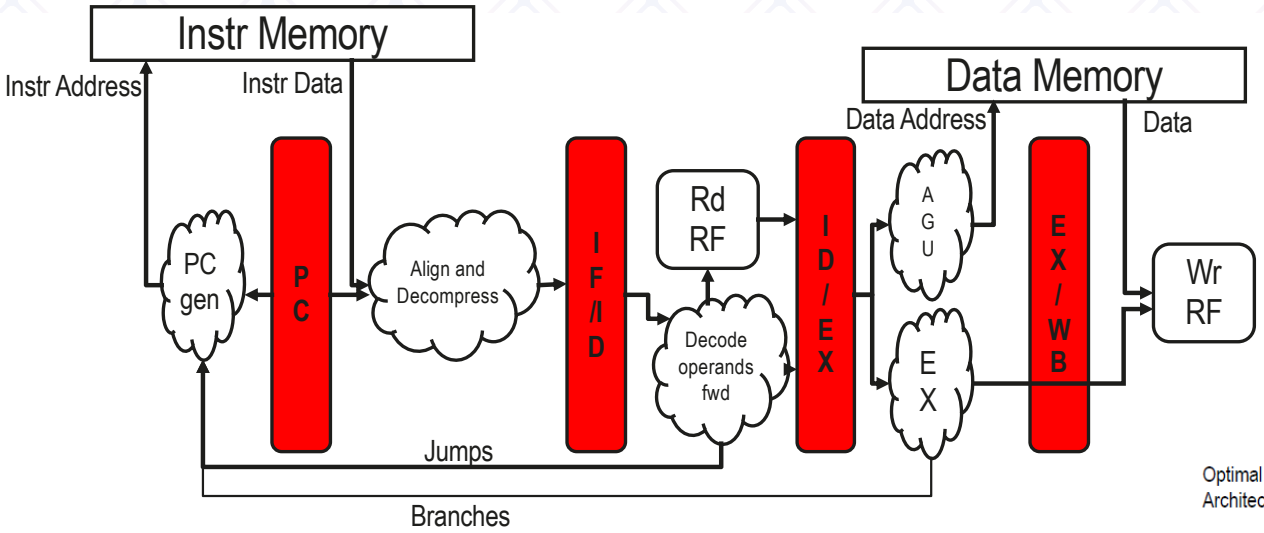
Energy efficiency @ GOPS is the Challenge

Chips-IT

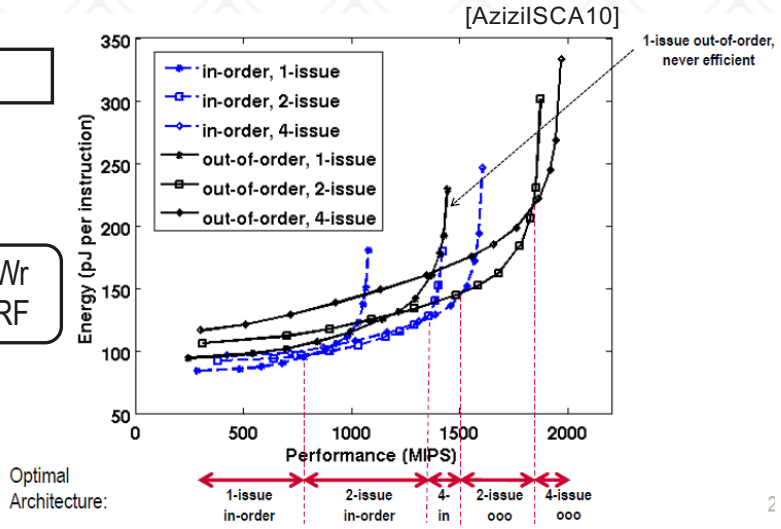
ARM Cortex-M MCUs: M0+, M4, M7 (40LP, typ, 1.1V)*



The Tunnel: High-Performance vs. Energy-Efficient



[AziziISCA10]

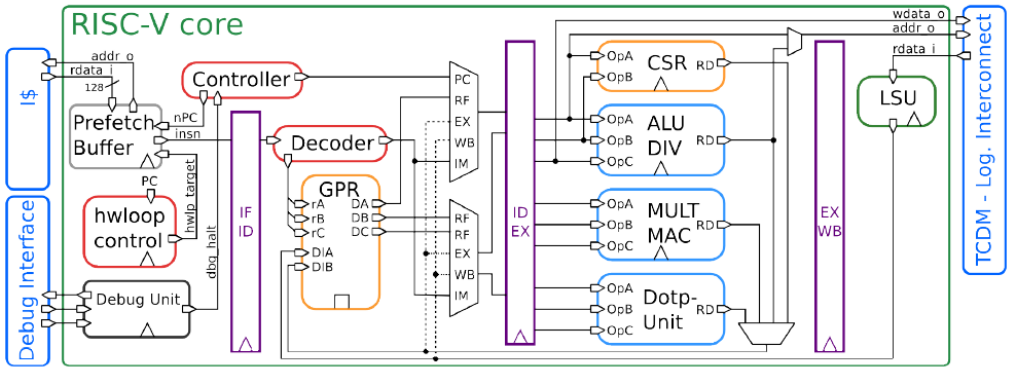


"Classical" core performance scaling trajectory

- Faster CLK -> deeper pipeline -> **IPC drops**
- Recover IPC -> superscalar -> **ILP bottleneck** (dependencies)
- Mitigate ILP bottlenecks -> OOO -> **huge power, area cost!**

RI5CY – Open MCU-class RISC-V Core (based on CV32E40P)

3-cycle ALU-OP, 4-cyle MEM-OP→IPC loss: LD-use, Branch



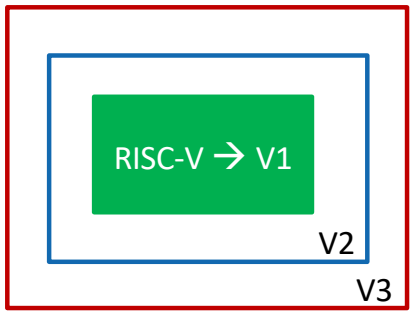
ISA is extensible *by construction* (great!)

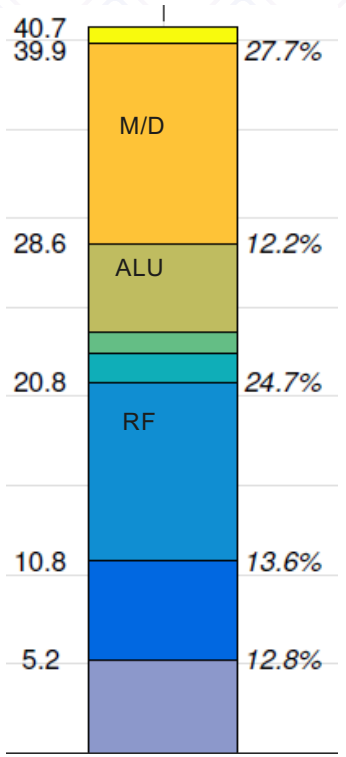
V1 Baseline RISC-V RV32IMC (not good for ML)

V2 HW loops
Post modified Load/Store
Mac

V3 SIMD 2/4 + DotProduct + Shuffling
Bit manipulation unit
Lightweight fixed point

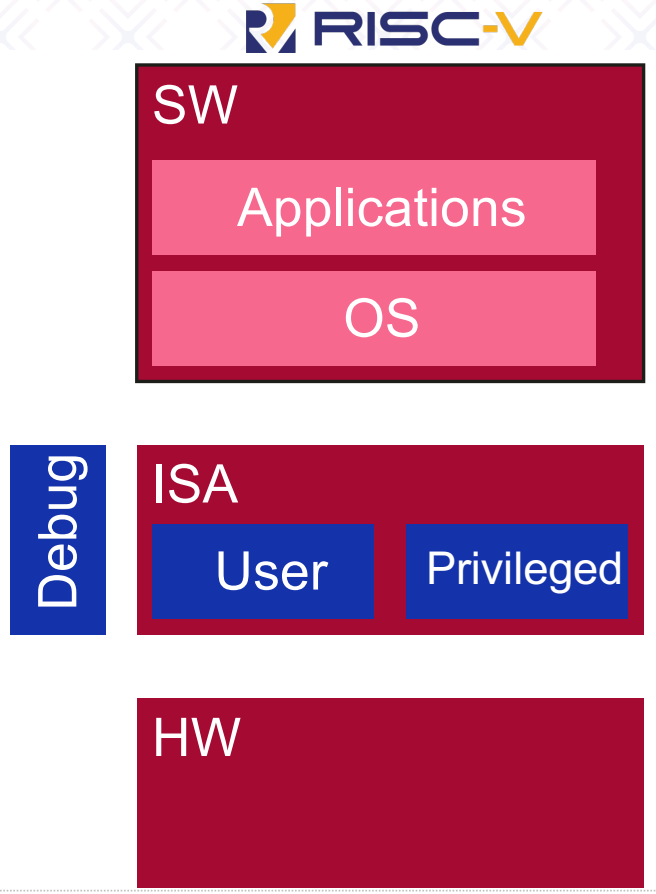
40 kGE
70% RF+DP





RISC-V Instruction Set Architecture

- Started by UC-Berkeley in 2010
- Contract between SW and HW
  - Partitioned into user and privileged spec
  - External Debug
- Standard governed by RISC-V foundation
  - ETHZ is a founding member of the foundation
  - Necessary for the continuity
- Defines 32, 64 and 128 bit ISA
  - No implementation, just the ISA
  - Different implementations (both open and close source)
- ETHZ+UNIBO specialized in **efficient implementations of RISC-V cores**



RISC-V Foundation Members

Chips-IT

RISC-V ISA Baseline and Extensions

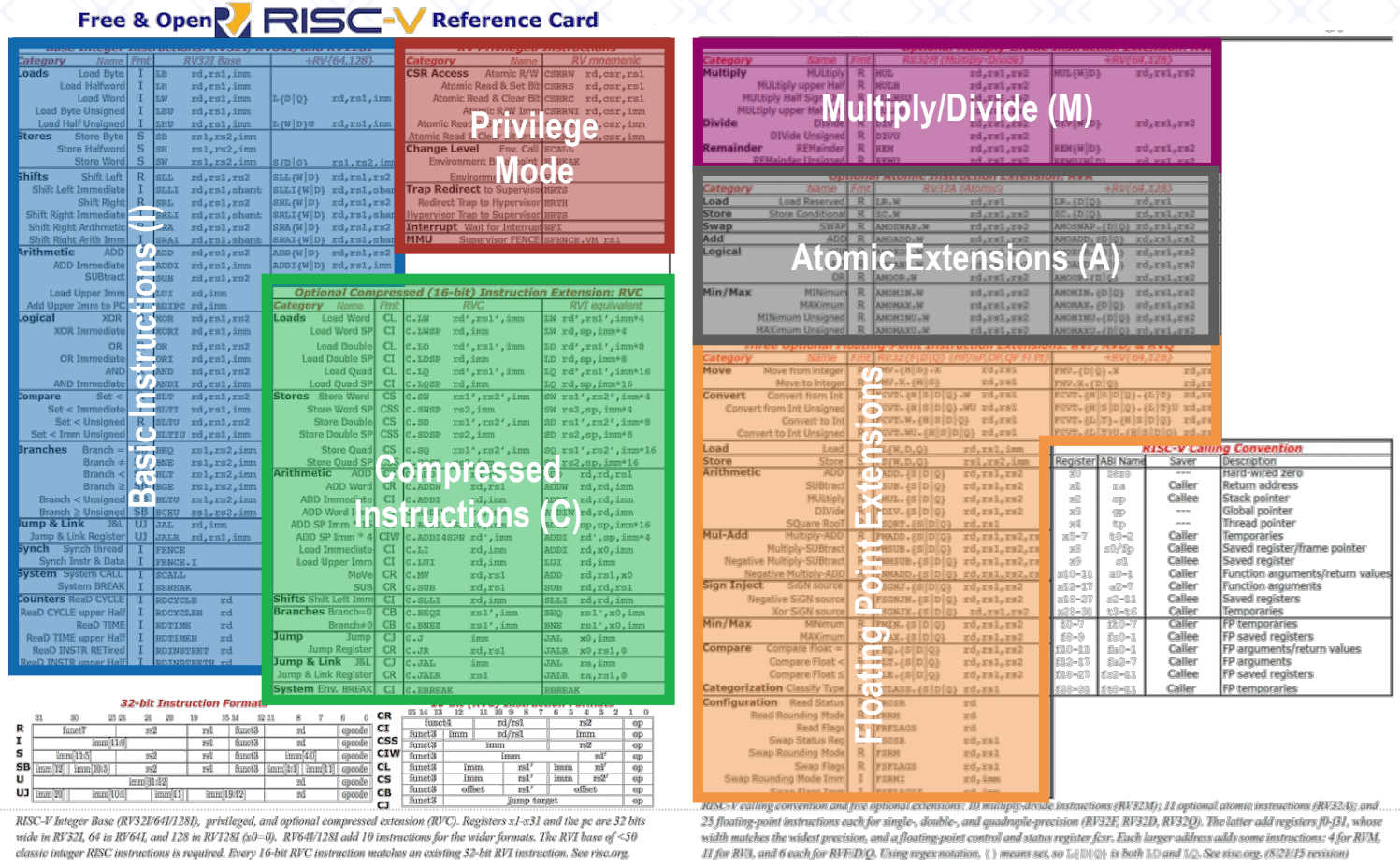
Chips-IT

I	Integer instructions (frozen)
E	Reduced number of registers
M	Multiplication and Division (frozen)
A	Atomic instructions (frozen)
F	Single-Precision Floating-Point (frozen)
D	Double-Precision Floating-Point (frozen)
C	Compressed Instructions (frozen)
X	Non Standard Extensions

- Kept very simple and extendable
  - Wide range of applications from IoT to HPC
- RV + word-width + extensions
  - RV32IMC: 32bit, integer, multiplication, compressed
- User specification:
  - Separated into extensions, only **I** is mandatory
- Privileged Specification (WIP):
  - Governs OS functionality: Exceptions, Interrupts
  - Virtual Addressing
  - Privilege Levels

Baseline + Extensions in a Page

Chips-IT



RISC-V Integer Base (RV32I/64I/128I), privileged, and optional compressed extension (RVC). Registers x1-x31 and the pc are 32 bits wide in RV32I, 64 in RV64I, and 128 in RV128I (x0=0). RV64I/128I add 10 instructions for the wider formats. The RVI base of <50 classic integer RISC instructions is required. Every 16-bit RVC instruction matches an existing 32-bit RVI instruction. See risc.org.

RISC-V calling convention and five optional extensions: 10 multiply-divide instructions (RV32M); 11 optional atomic instructions (RV32A); and 25 floating-point instructions each for single-, double-, and quadruple-precision (RV32F, RV32D, RV32Q). The latter add registers f0-f31, whose width matches the widest precision, and a floating-point control and status register fcsr. Each larger address adds some instructions: 4 for RVM, 11 for RVA, and 6 each for RVF/D/Q. Using regex notation, { } means set, so L{D|Q} is both LD and LQ. See risc.org. (8/21/15 revision)

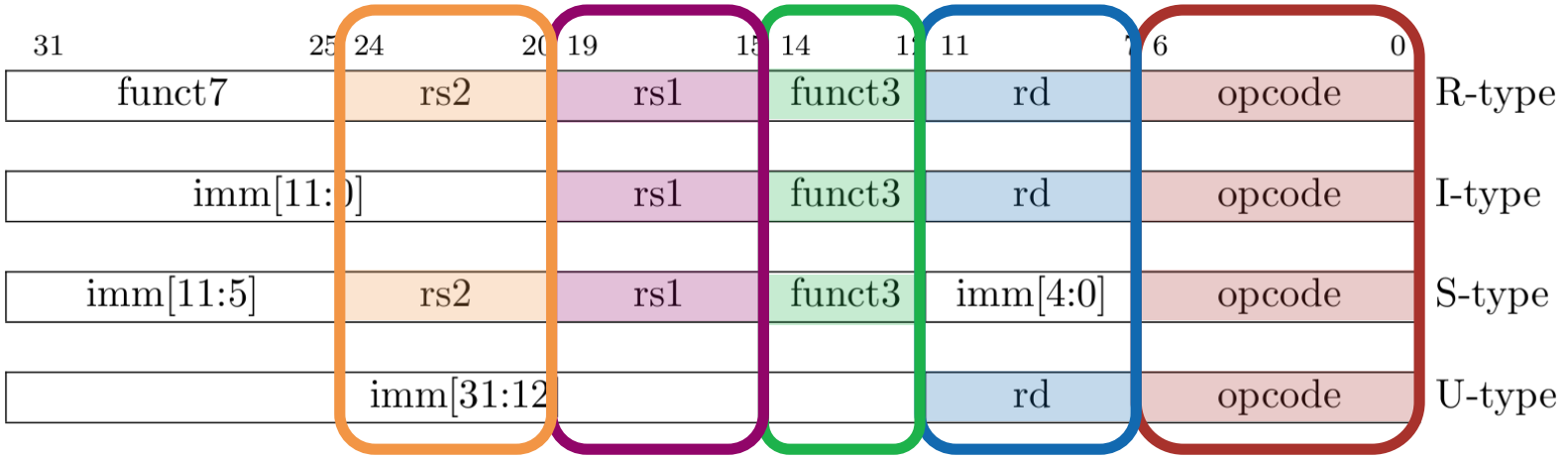
RISC-V Architectural State

Chips-IT

- There are 32 registers, each 32 / 64 / 128 bits long
  - Named x0 to x31
  - x0 is hard wired to zero
  - There is a standard '**E**' extension that uses only 16 registers (RV32E)
- In addition one program counter (PC)
  - Byte based addressing, program counter increments by 4/8/16
- For floating point operation 32 additional FP registers
- Additional Control Status Registers (CSRs)
  - Encoding for up to 4'096 registers are reserved. Not all are used.

RISC-V Instructions four basic types

- **R** register to register operations
- **I** operations with immediate/constant values
- **S / SB** operations with two source registers
- **U / UJ** operations with large immediate/constant value



RISC-V is a load/store architecture

Chips-IT

- All operations are on internal registers
  - Can not manipulate data in memory directly
- Load instructions to copy from memory to registers
- R-type or I-type instructions to operate on them
- Store instructions to copy from registers back to memory
- Branch and Jump instructions
- 1/3 ALU utilization if operands are from/to memory (LD, ALU, ST)

Encoding of the instructions, main groups

Chips-IT

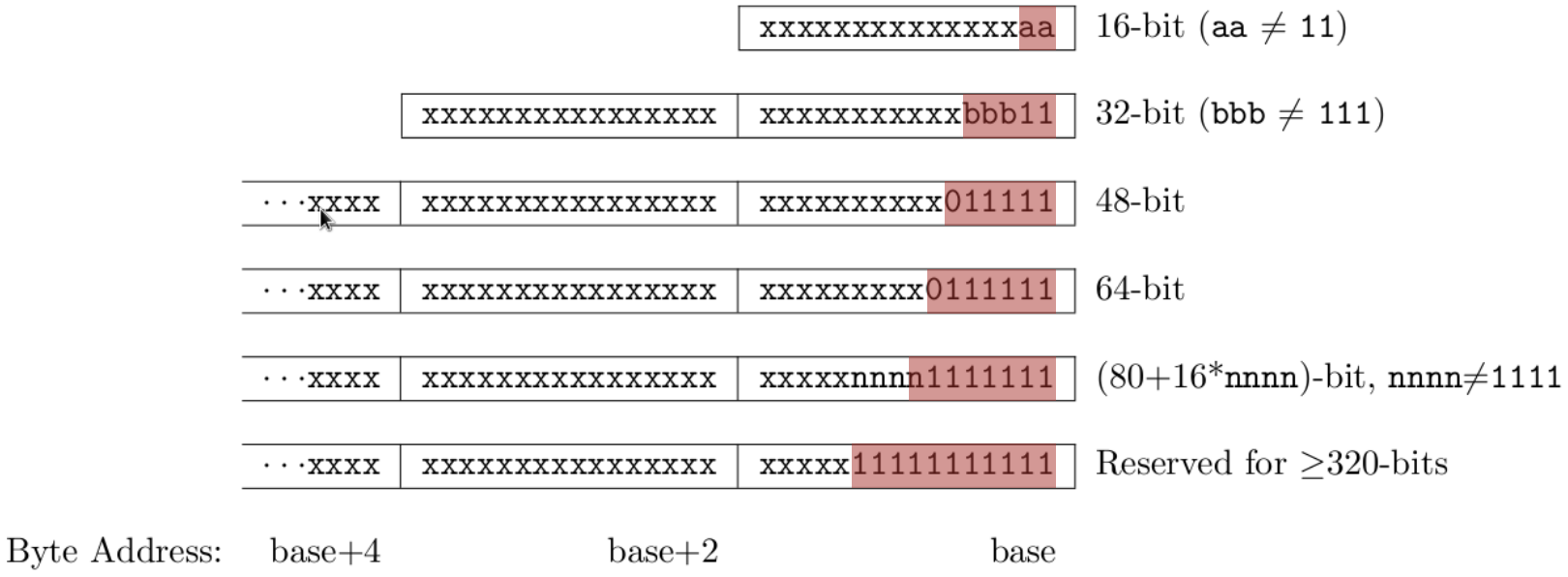
- Reserved opcodes for standard extensions
- Rest of opcodes free for custom implementations
- Standard extensions will be frozen/not change in the future

inst[4:2] / inst[6:5]	000	001	010	011	100	101	110	111 ($> 32b$)
00	LOAD	LOAD-FP	*custom-0*	MISC-MEM	OP-IMM	AUIPC	OP-IMM-32	$48b$
01	STORE	STORE-FP	*custom-1*	AMO	OP	LUI	OP-32	$64b$
10	MADD	MSUB	NMSUB	NMADD	OP-FP	*reserved*	*custom-2/rv128*	$48b$
11	BRANCH	JALR	*reserved*	JAL	SYSTEM	*reserved*	*custom-3/rv128*	$\geq 80b$

Extensibility is integral to RISC-V ISA design!

RISC-V Instruction Length is Encoded

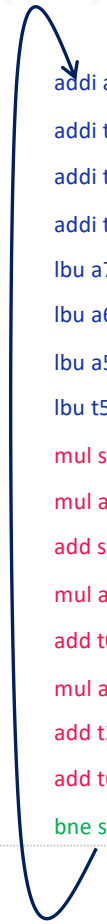
- LSB of the instruction tells how long the instruction is
- Supports instructions of 16, 32, 48, 64, 80, 96, … , 320 bit
  - Allows RISC-V to have Compressed instructions



RISC-V Basic Matrix Multiplication Loop

RV32IMC

```
addi a0, a0, 1
addi t1, t1, 1
addi t3, t3, 1
addi t4, t4, 1
lbu a7, -1(a0) (A)
lbu a6, -1(t4) (C)
lbu a5, -1(t3) (D)
lbu t5, -1(t1) (B)
mul s1, a7, a6 (A*C)
mul a7, a7, a5 (A*D)
add s0, s0, s1 (0 + A*C)
mul a6, a6, t5 (B*C)
add t0, t0, a7 (0 + A*D)
mul a5, a5, t5 (B*D)
add t2, t2, a6 (0 + B*C)
add t6, t6, a5 (0 + B*D)
bne s5, a0, 0x1C000BC
```



Lots of pointer updates!
Only one MACs every 4 cycles!
Extremely inefficient!

Xpulp: Post increment LD/ST

Chips-IT

- Automatic address update
  - Update base register with computed address after the memory access

  ⇒Save instructions to update address register
  - Post-increment:
    - Base address serves as memory address
- Offset can be stored in:
  - Register
  - Immediate

```
c = 0;
for(i=0;i<100;i++)
    c = c + a[i]*b[i];
```

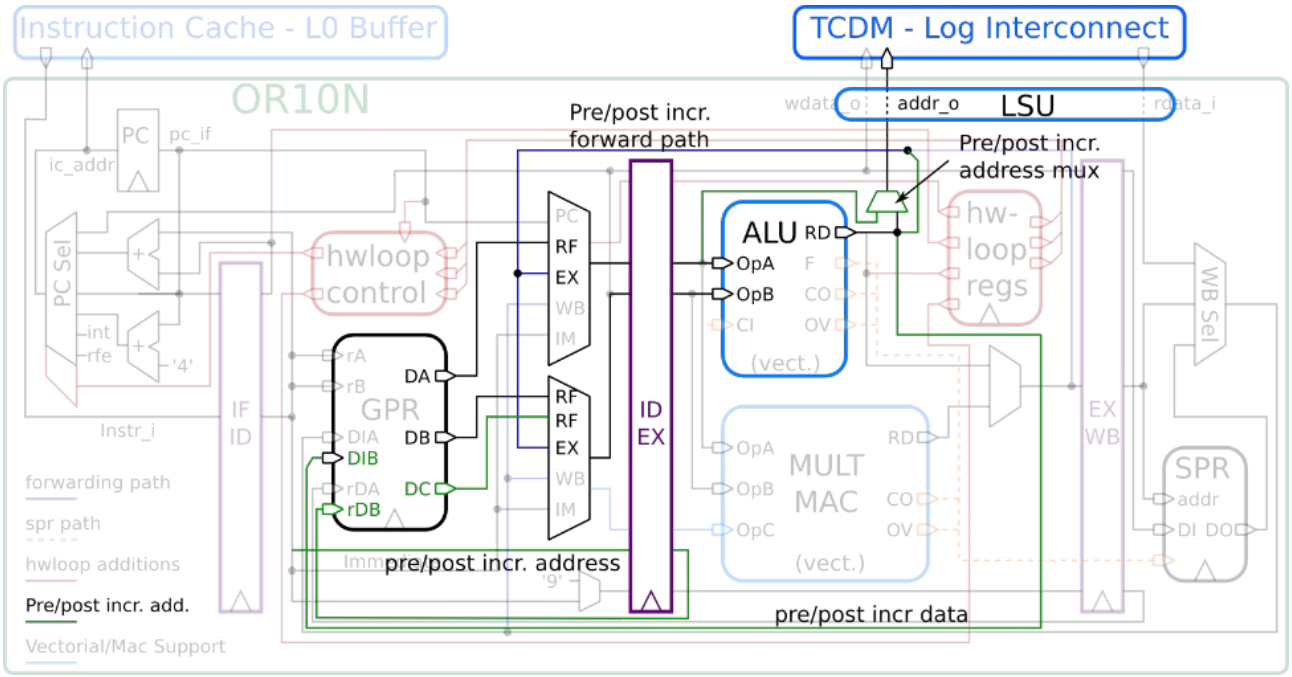
Original RISCV	Auto-incr load/store
addi x4, x0, 64 Lstart : *lb x2, 0(x10)* *lb x3, 0(x12)* *addi x10, x10, 1* *addi x12, x12, 1* ….. bne x2,x3, Lstart	addi x4, x0, 64 Lstart : *lb x2, 0(x10!)* *lb x3, 0(x12!)* ….. bne x2,x3, Lstart

⇒save 2 additional instructions to update the read addresses of the operands!

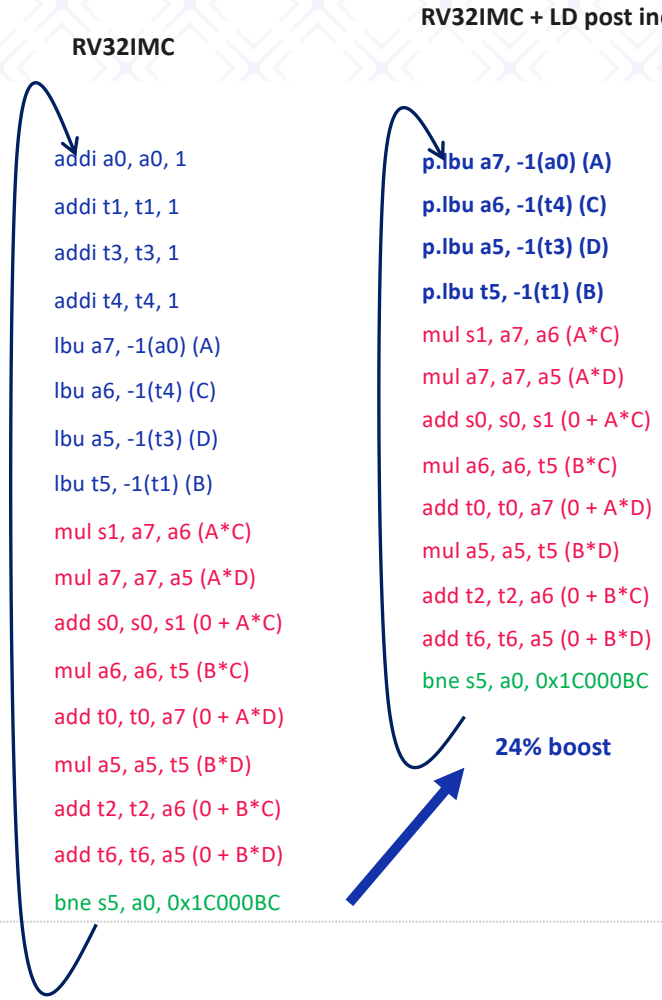
Xpulp: Post increment implementation

- Register file requires additional write port
- Register file requires additional read port if offset is stored in register

- Processor core area increases by 8-12 %
- Ports can be used for other instructions



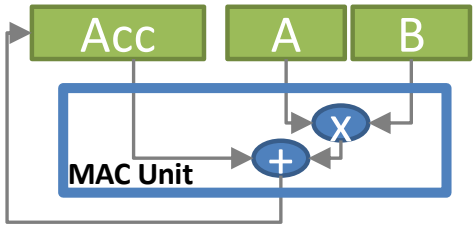
Xpulp: Post increment effect



Xpulp: Multiply-Accumulate (MAC)

Chips-IT

- Accumulation on 32 bit data p.mac

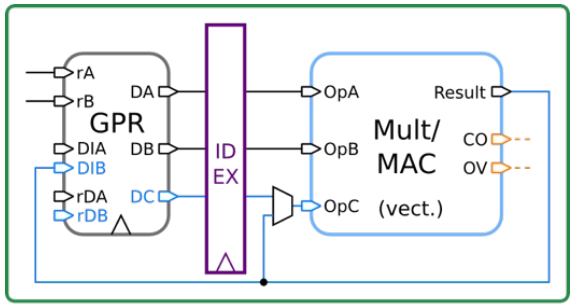


- Directly on the register file
- Pro:
  - Faster access to mac accumulation
  - Single cycle mult/mac
- Cons:
  - Additional read port on the register file
  - used for pre/post increment with register

```
int acc=0, coeff[N], inp[N];
for(int i=0; i<N; i++)
    acc += coeff[i] * inp[i];
```

acc = **__builtin_pulp_mac** (**inp**[i], **coeff**[i], **acc**);

Intrinsics: special functions that map directly to inlined DSP instructions.
However, the compiler can already place the p.mac instruction into the above code!



Xpulp: MAC effect

Xpulp: HW Loops Extensions

Chips-IT

Hardware loop setup with:

- 3 separate instructions
  - *lp.start, lp.end, lp.count, lp.counti*

  ⟹No restriction on start/end address
- Fast setup instructions
  - *lp.setup, lp.setupi*

  ⟹Start address= PC + 4

  ⟹End address= start address + offset

  ⟹Counter from immediate/register

```
c = 0;
for(i=0;i<100;i++)
    c = c + a[i]*b[i];
```

Original RISC-V

```
//initialize counter
mv   x4, 100
// init accumulator
mv   x5, 0
Lstart:
  //decrement counter
  addi  x4, x4, -1
  //load elements from mem
  lw x8, 0(x9)
  lw x10, 0(x11)
  //update memory pointers
  add x9, x9, 4
  add x11, x11, 4
  //mac
  mul x8, x8, x10
  add x5, x5, x8
bne  x4, x0, Lstart
```

HW Loop Ext

```
// init accumulator
mv   x5, 0
//set number iterations, start and end of
the loop
lp.setupi 100, Lend
  //load elements from mem
  lw x8, 0(x9)
  lw x10, 0(x11)
  //update memory pointers
  add x9, x9, 4
  add x11, x11, 4
  //mac
  mul x8, x8, x10
Lend:  add x5, x5, x8
```

No counter and branch overhead!

Xpulp: Hardware Loop Implementation

- Hardware loops or Zero Overhead Loops to remove branch overhead in for loops
- After configuration with start, end, count variables no more comparison and branches are required.
- Smaller loop benefit more
- Loop needs to be set up beforehand and is fully defined by:
  - Start address
  - End address
  - Counter

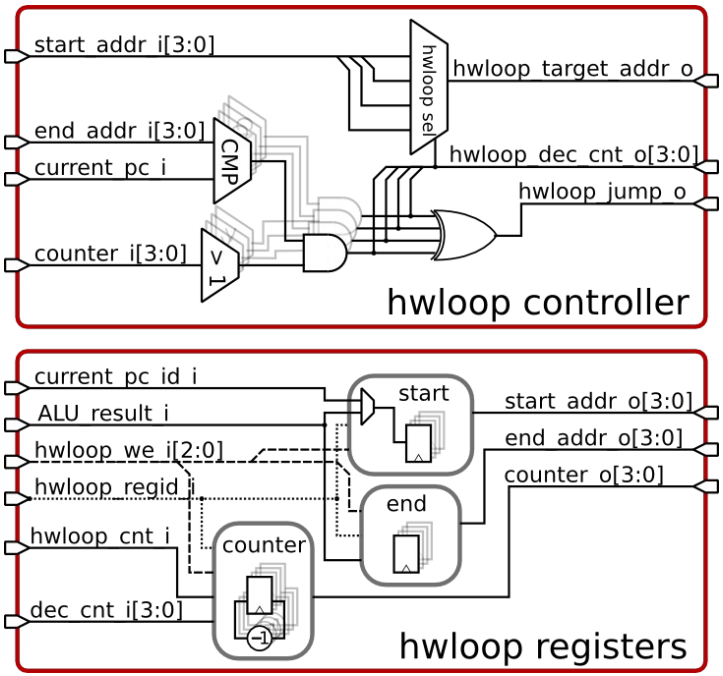
Two sets registers implemented to support nested loops.

Area costs:

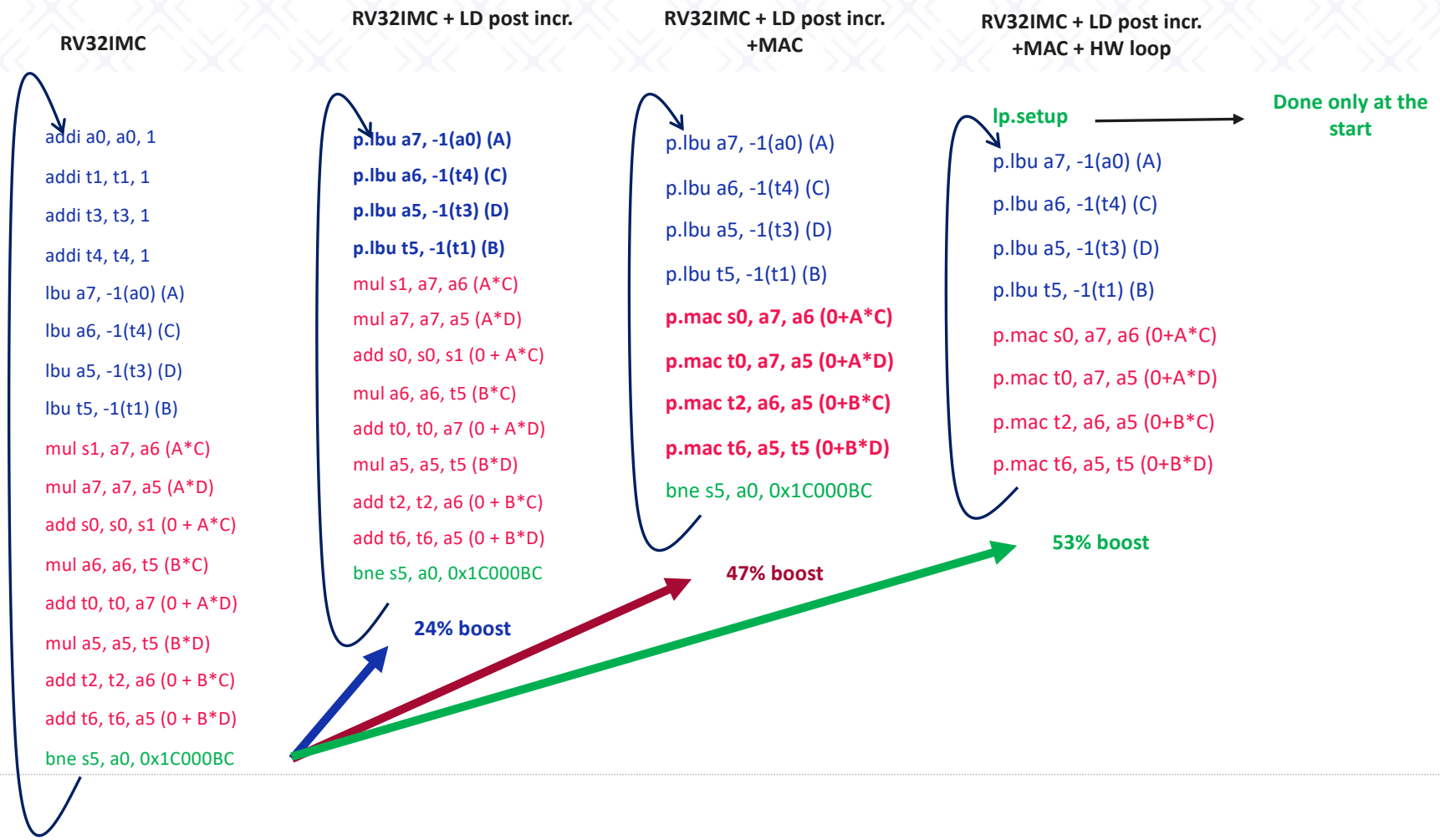
- Processor core area increases by 5%

Performance:

- Speedup can be up to factor 2!



Xpulp: HW Loop Effect



Xpulp: Packed SIMD (Single-Instruction, Multiple Data)

Chips-IT

Remember: DNN inference is OK with low-bitdwidth operands

- Packed-SIMD extensions
  - Make usage of resources the best in performance with little overhead
  - Target for embedded systems, RVV is for high performance
- pSIMD in 32bit machines
  - Vectors are either 4 8bits-elements or 2 16bits-elements
- pSIMD instructions

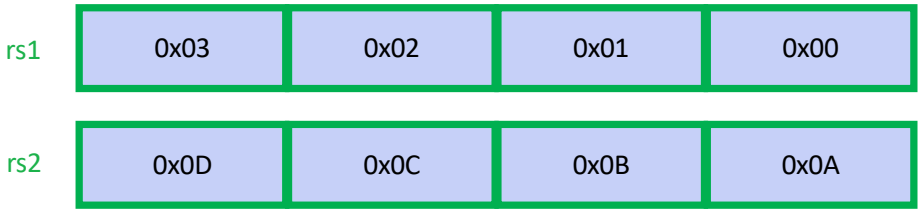
Computation	add, sub, shift, avg, abs, dot product
Compare	min, max, compare
Manipulate	extract, pack, shuffle

Xpulp: Packed SIMD

Chips-IT

Same Register-file

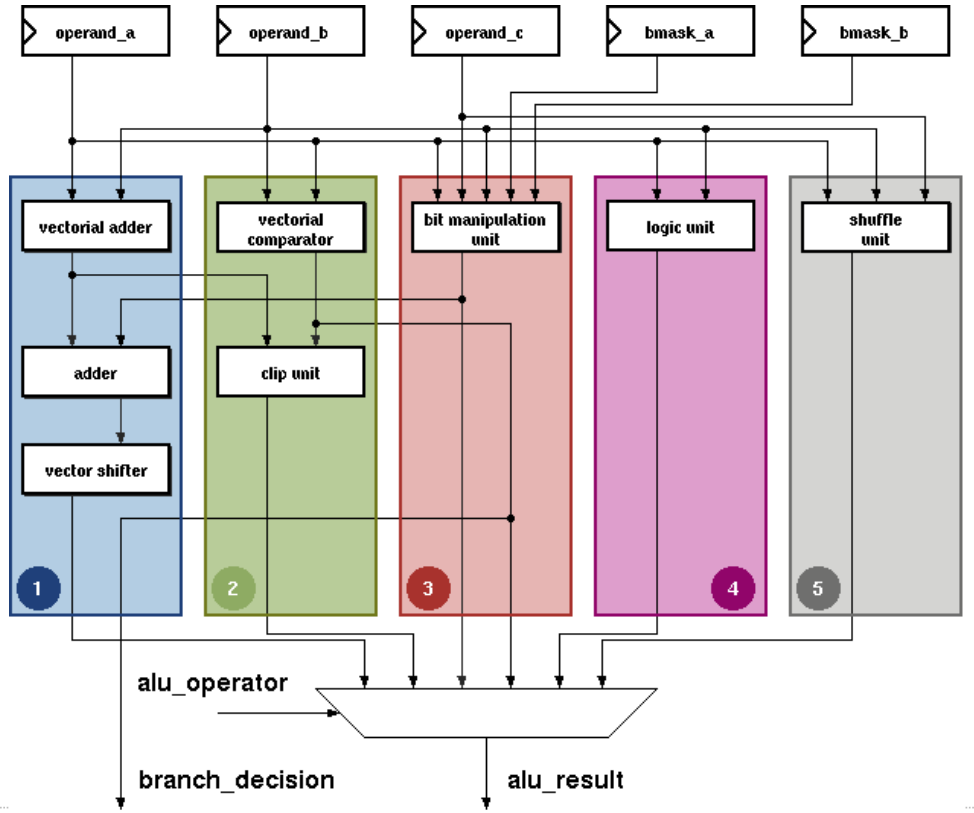
- The instruction encode how to interpret the content of the register



add rD, rs1, rs2	rD = 0x03020100 + 0x0D0C0B0A
add.h rD, rs1, rs2	rD[0] = 0x0100 + 0x0B0A rD[1] = 0x0302 + 0x0D0C
add.b rD, rs1, rs2	rD[0] = 0x00 + 0x0A rD[1] = 0x01 + 0x0B rD[2] = 0x02 + 0x0C rD[3] = 0x03 + 0x0D

Xpulp: Packed SIMD ALU architecture

- Advanced ALU for Xpulp extensions
- Optimized datapath to reduce resources
- Multiple-adders for round
- Adder followed by shifter for fixed point normalization
- Clip unit uses one adder as comparator and the main comparator



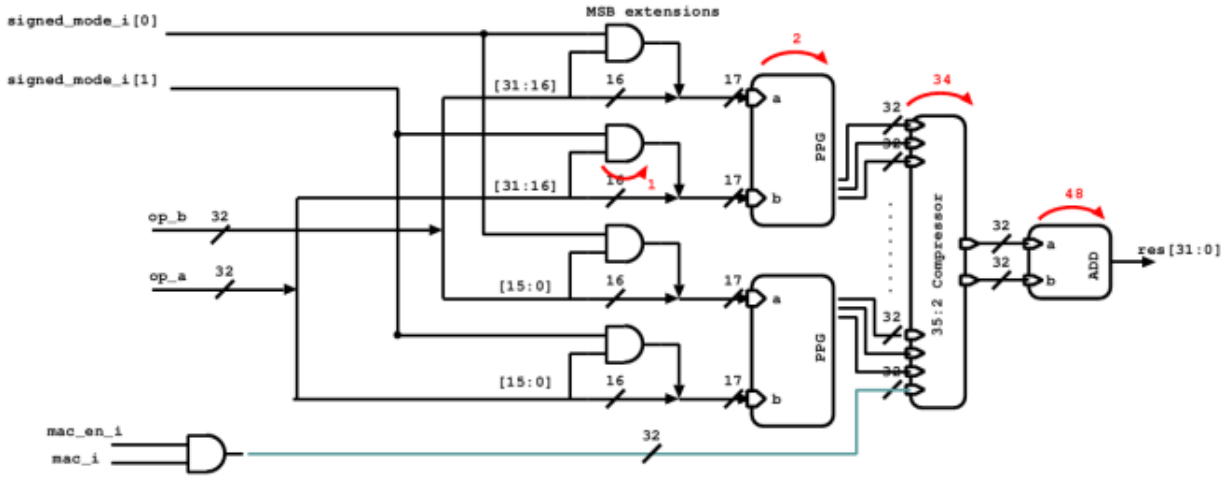
Expanding SIMD Dot Product

Chips-IT

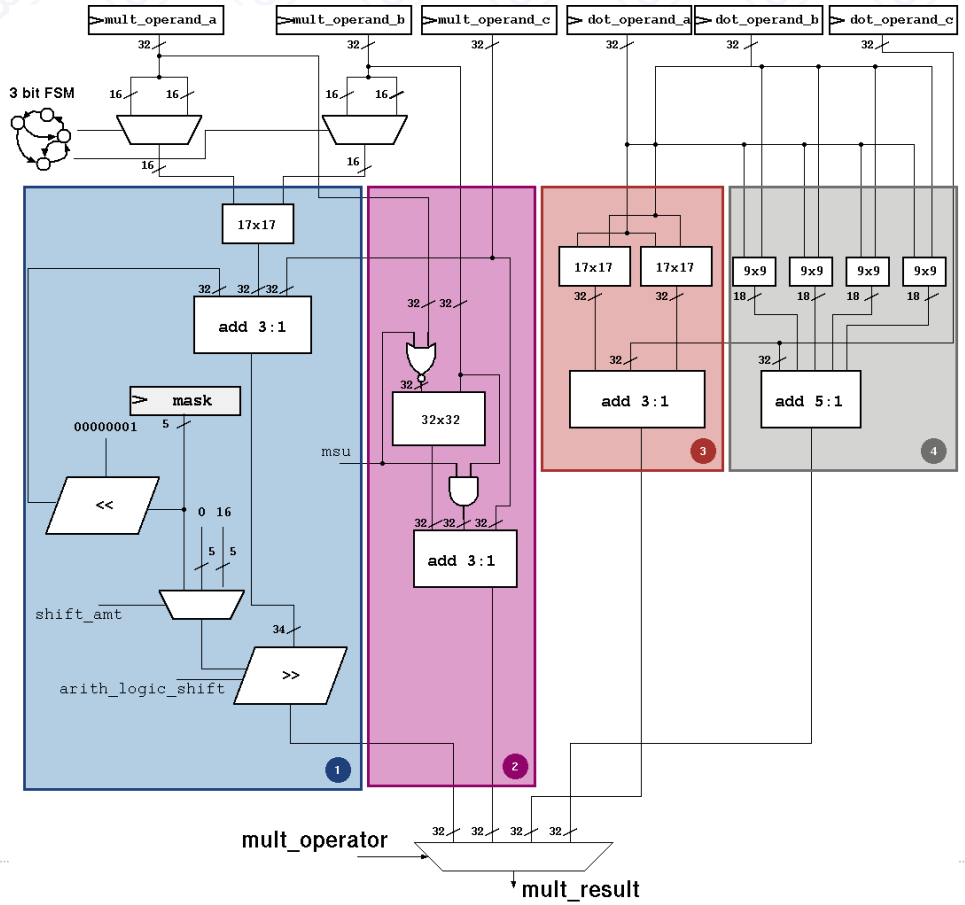
- Dot Product: (half word example)

  C[31:0] = A[31:16]*B[31:16] + A[15:0]*B[15:0] + C[31:0]

  → 2 multiplications, 1 addition, 1 accumulation in 1 cycle (2x for bytes)

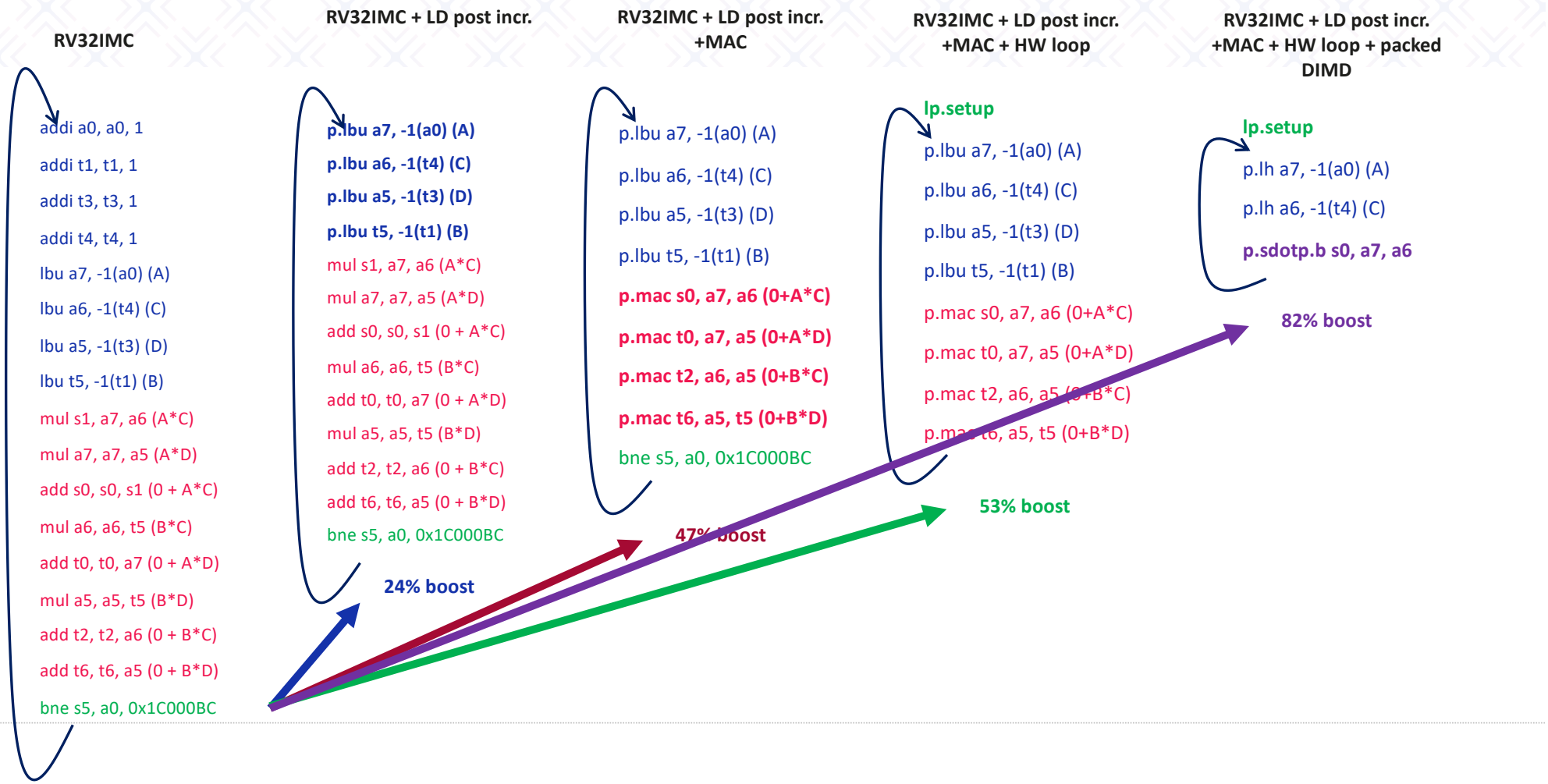


MUL architecture

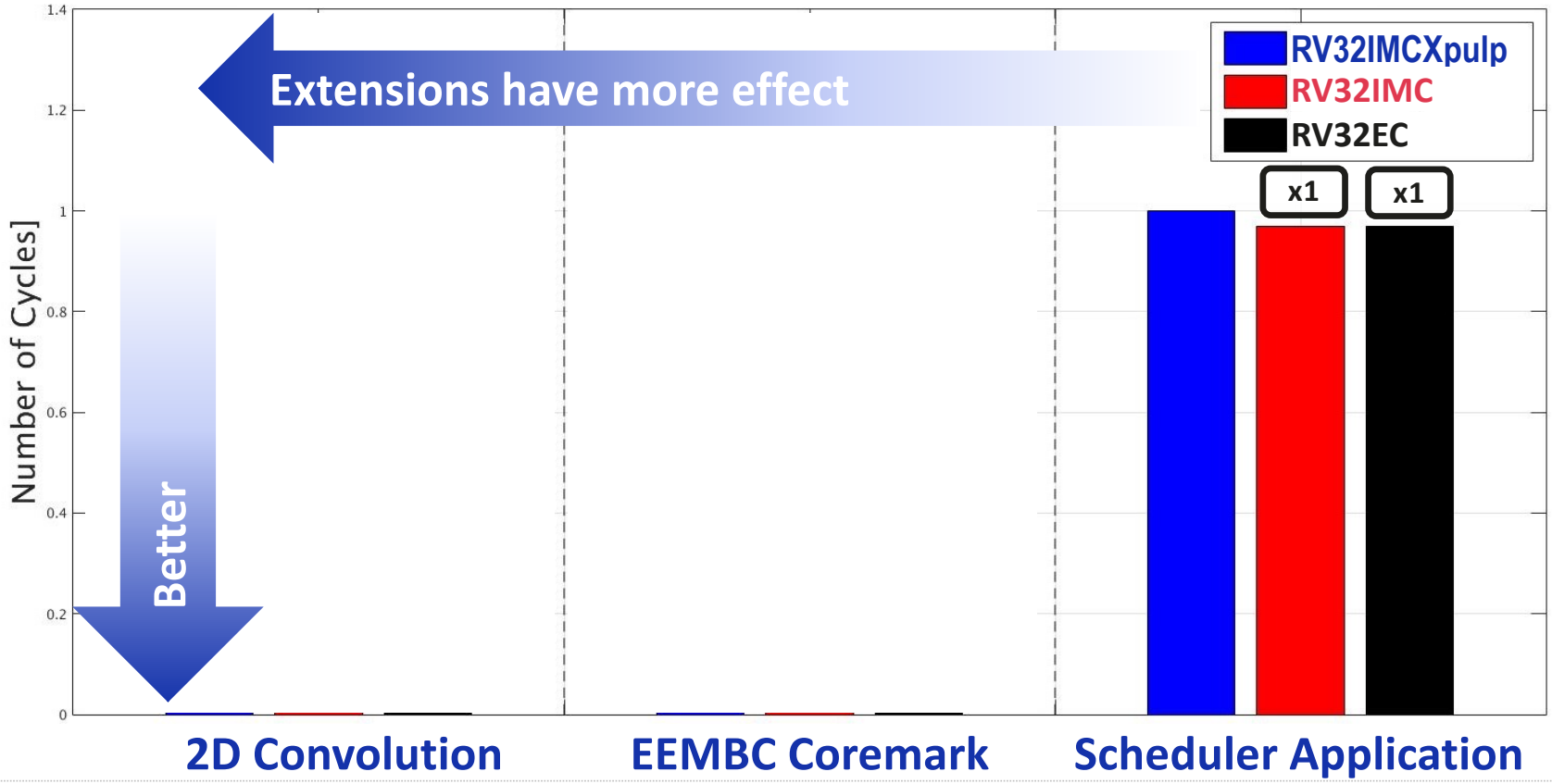


- **16x16b** with sign selection for short multiplications [with round and normalization]. 5 cycles FSM for higher 64-bits (mulh* instructions)
- **32x32b** single cycle MAC/MUL unit
- **16x16b** short parallel dot product
- **8x8b** byte parallel dot product
- *clock gating to reduce switching activity between the scalar and SIMD multipliers*

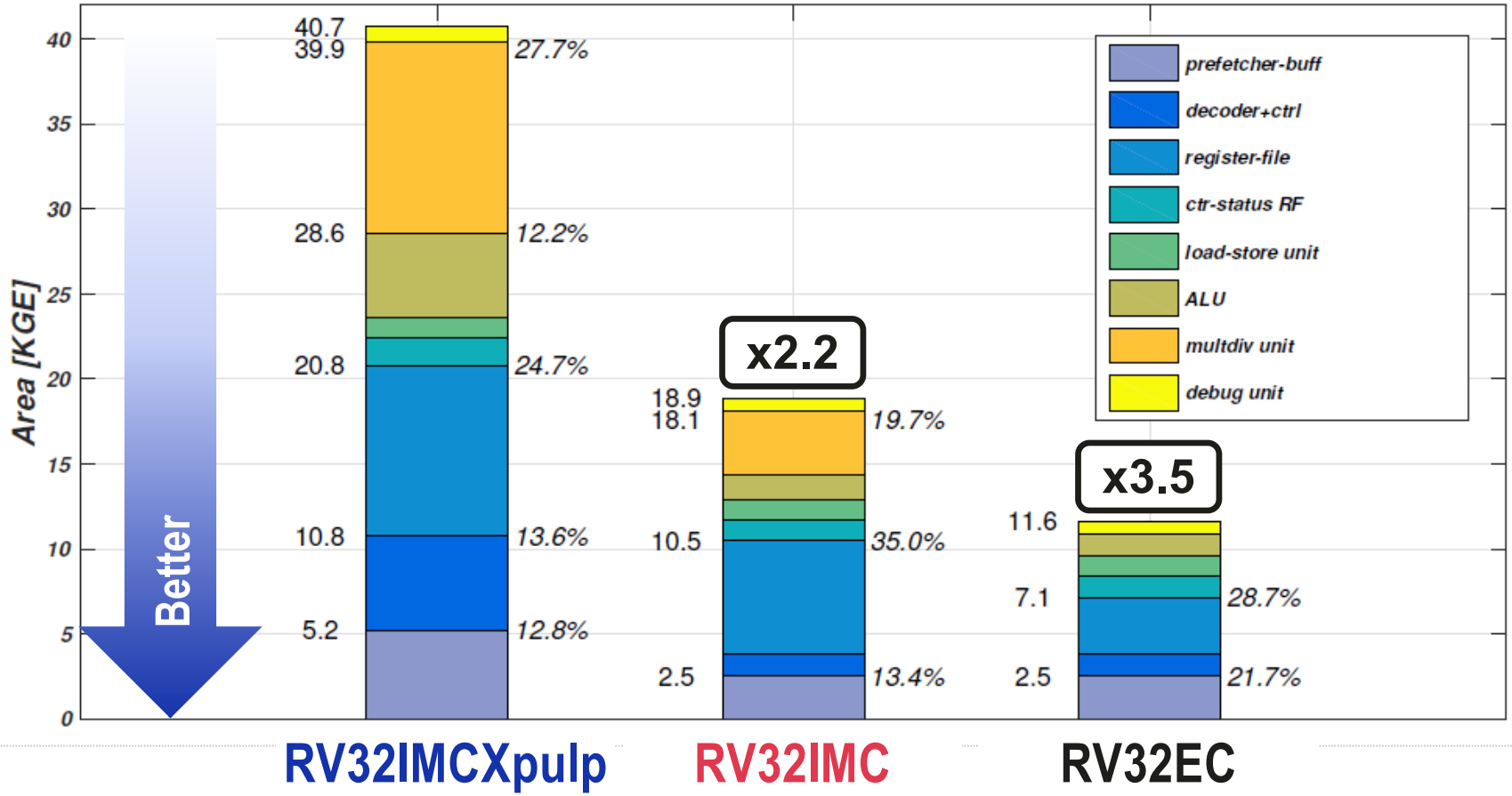
Xpulp: HW Loop Effect



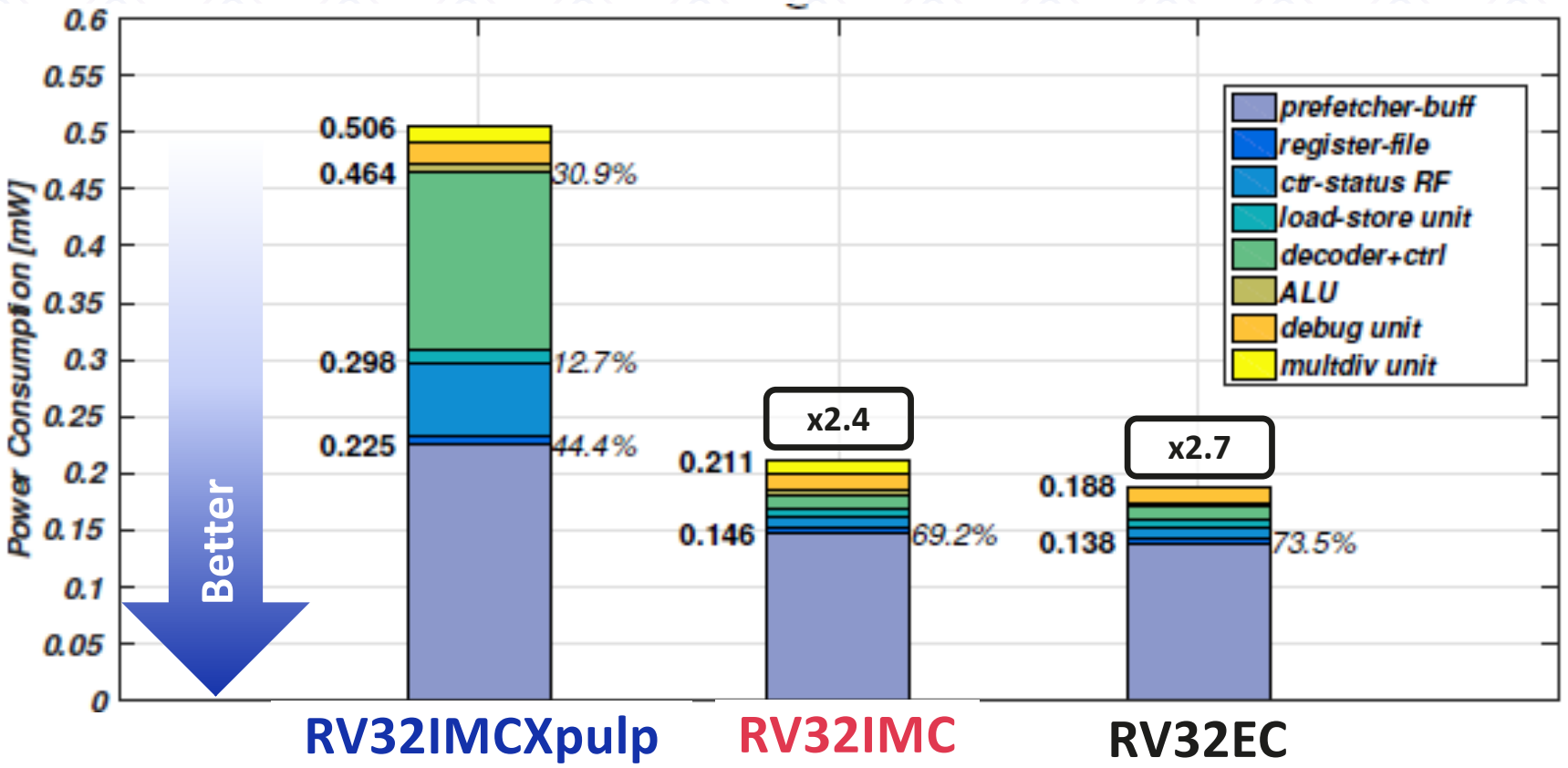
Runtime for three different applications



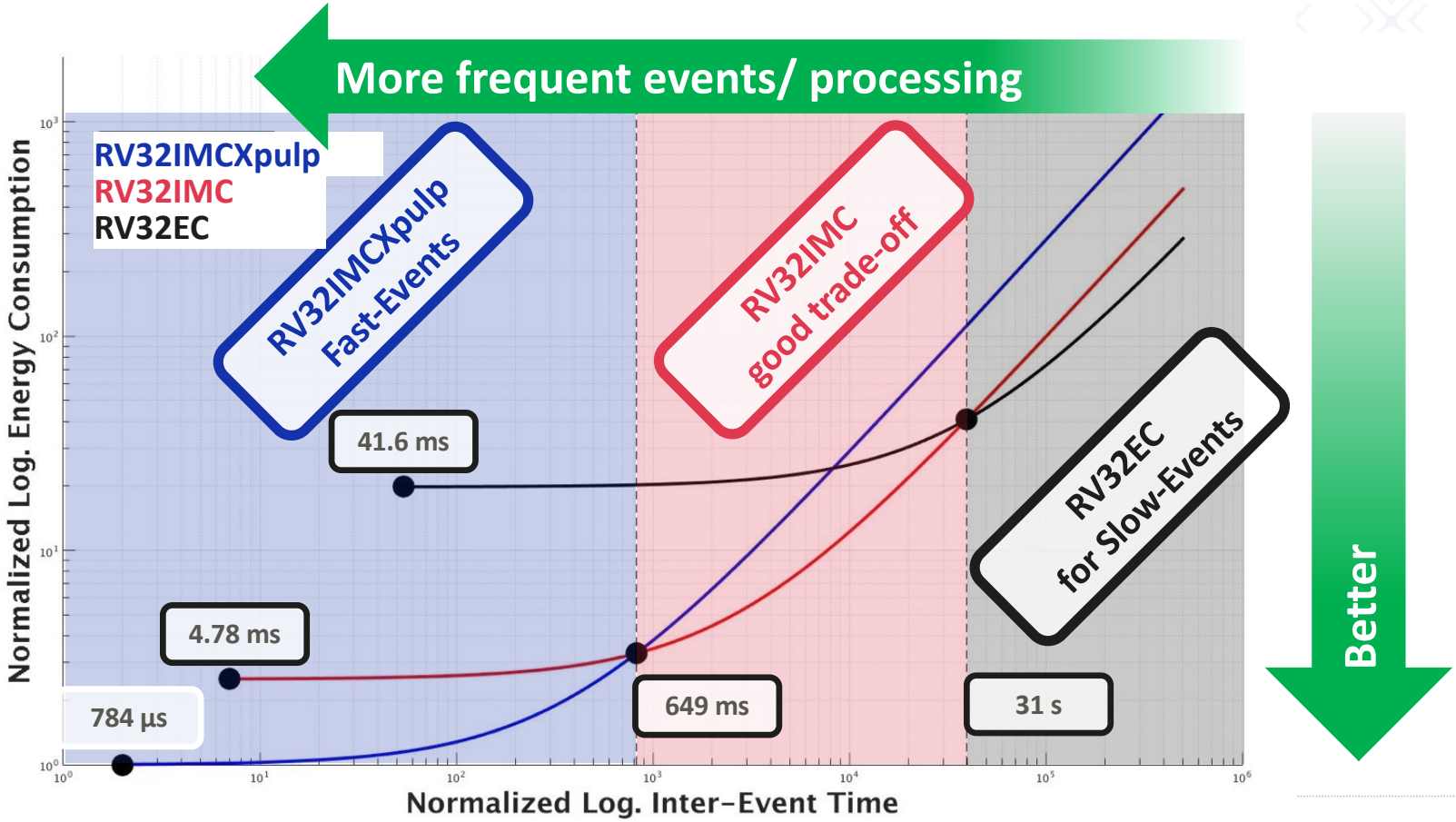
Different cores for different area budgets



Different cores for different power budgets



Energy Efficiency: 2D-Convolution @55MHz, 0.8V



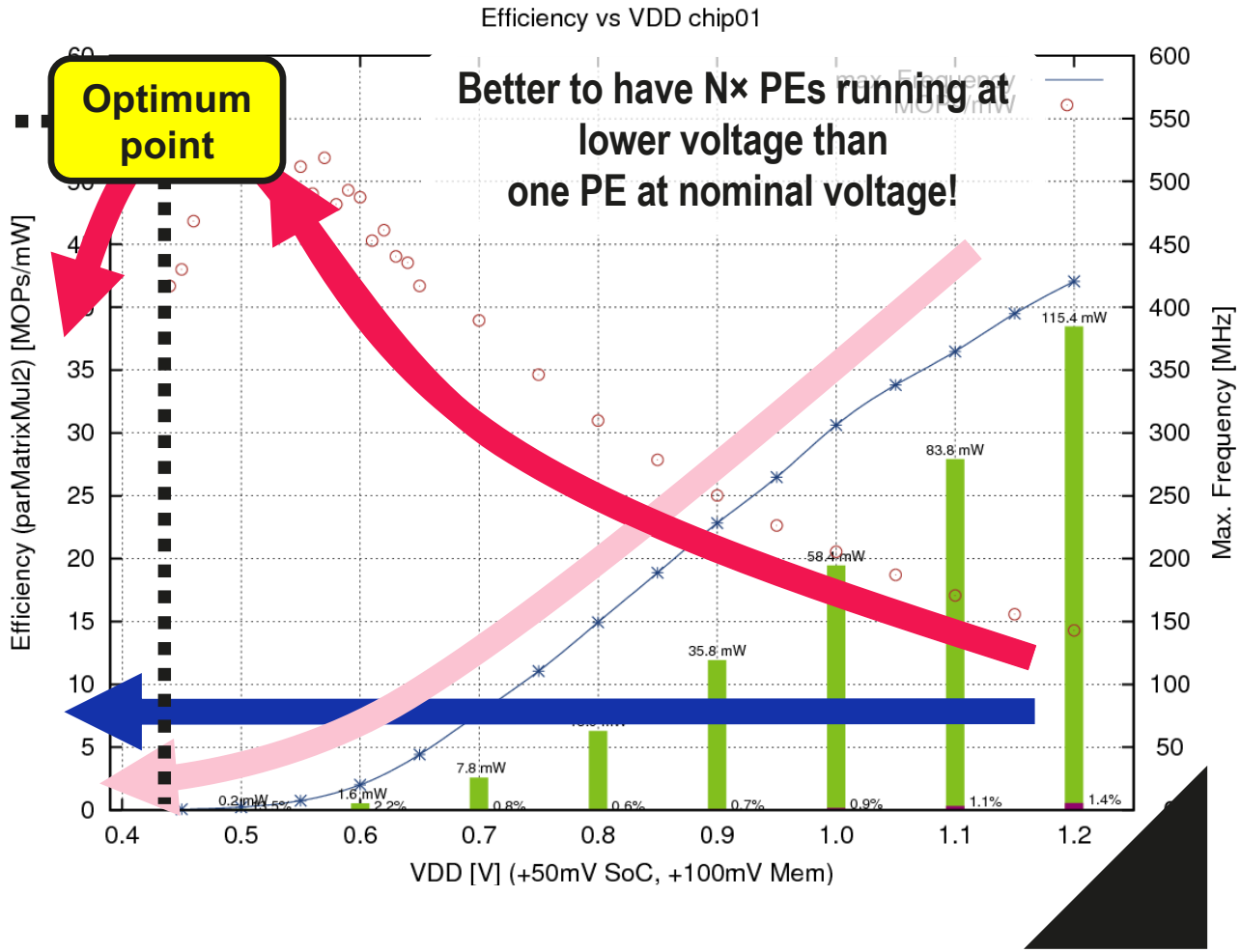
Thank you

Chips-IT

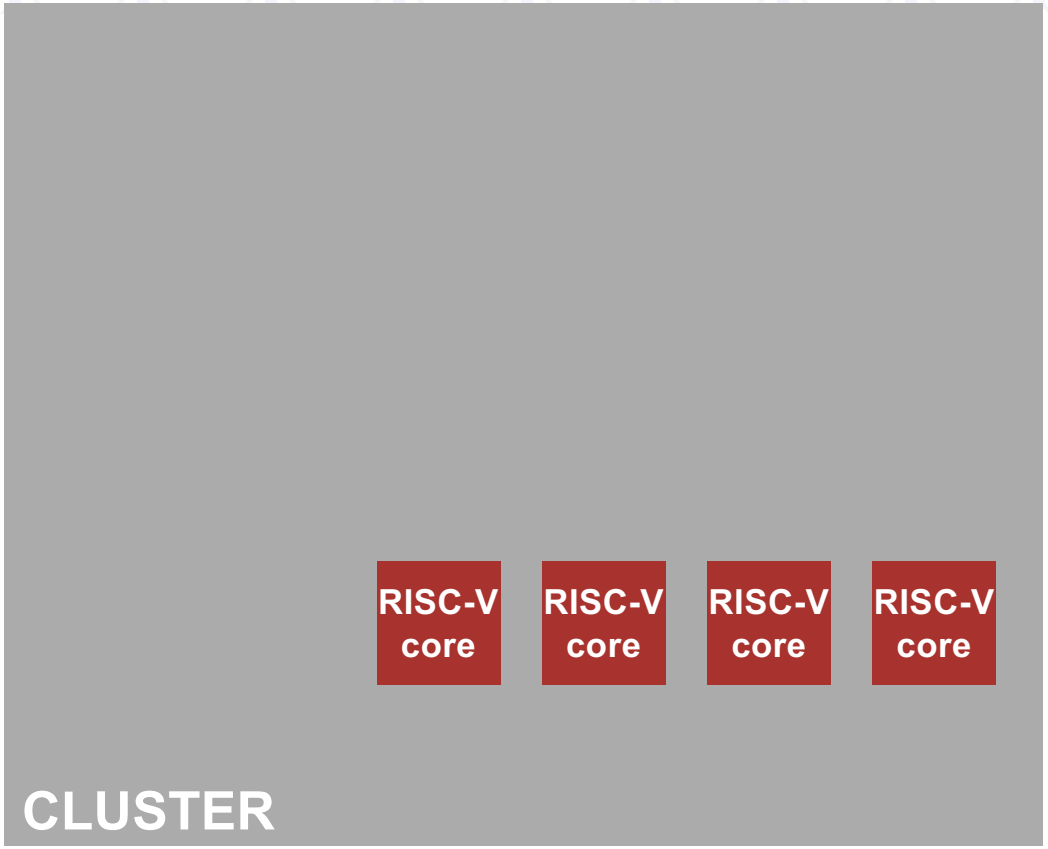
ML & Parallel, Near-threshold: a Marriage Made in Heaven

- As VDD decreases, decr
- However efficiency increases→ more w
- Until leakage effects start to dominate
- Put more units in parallel to get perform

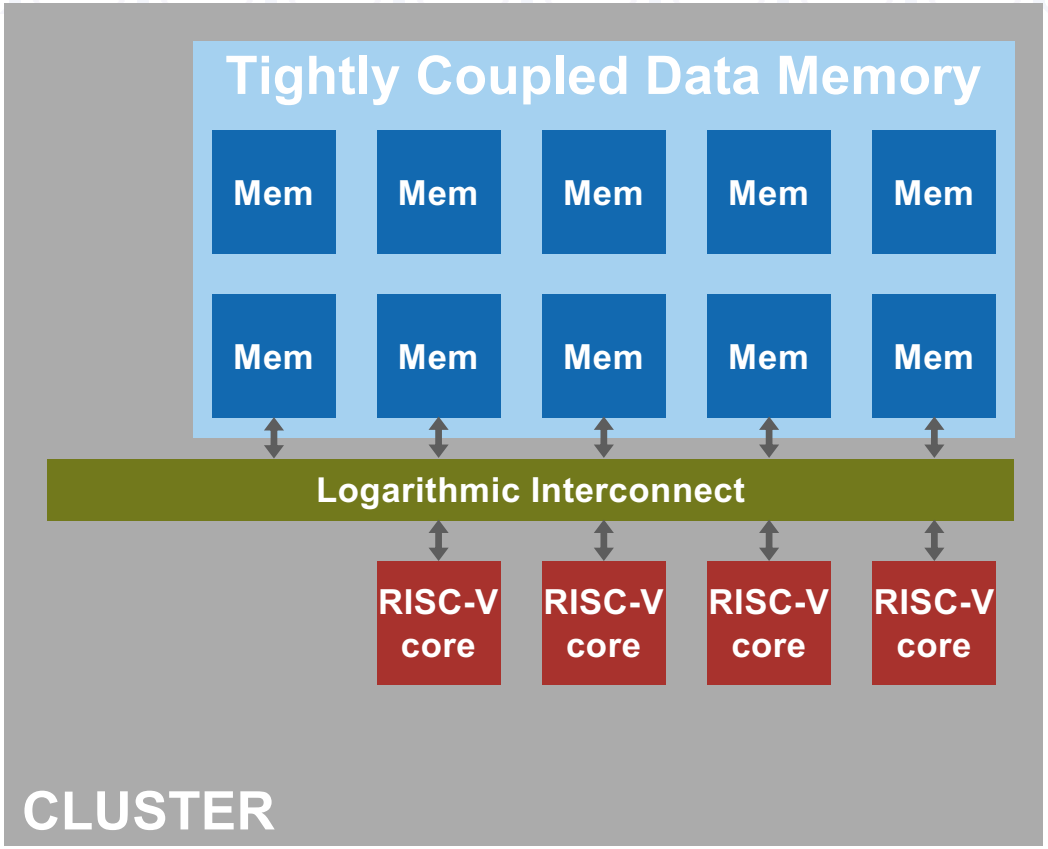
ML is massively parallel and scales well (P/S ↑ with NN size)



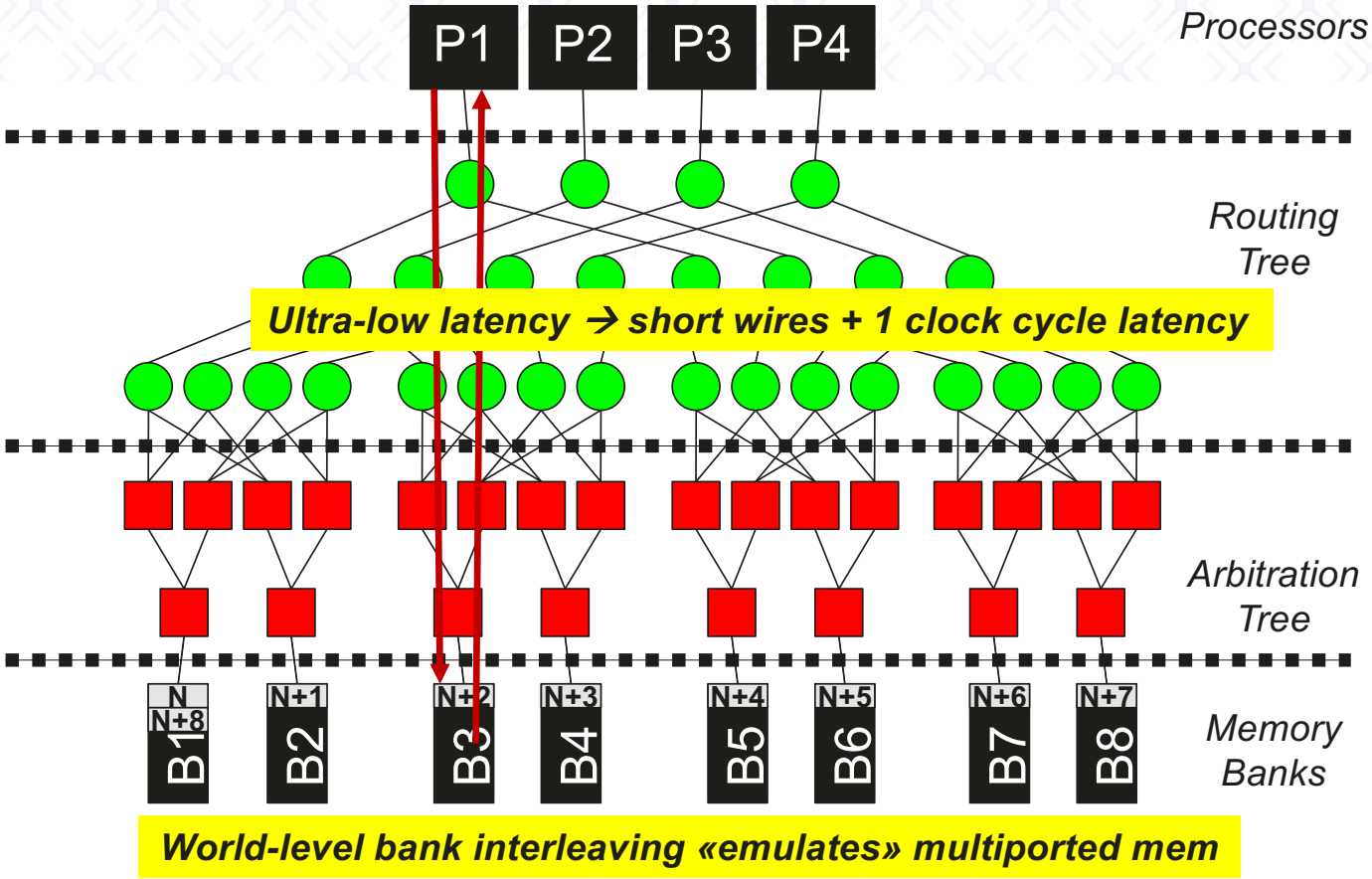
Multiple RI5CY Cores (1-16)



Low-Latency Shared TCDM

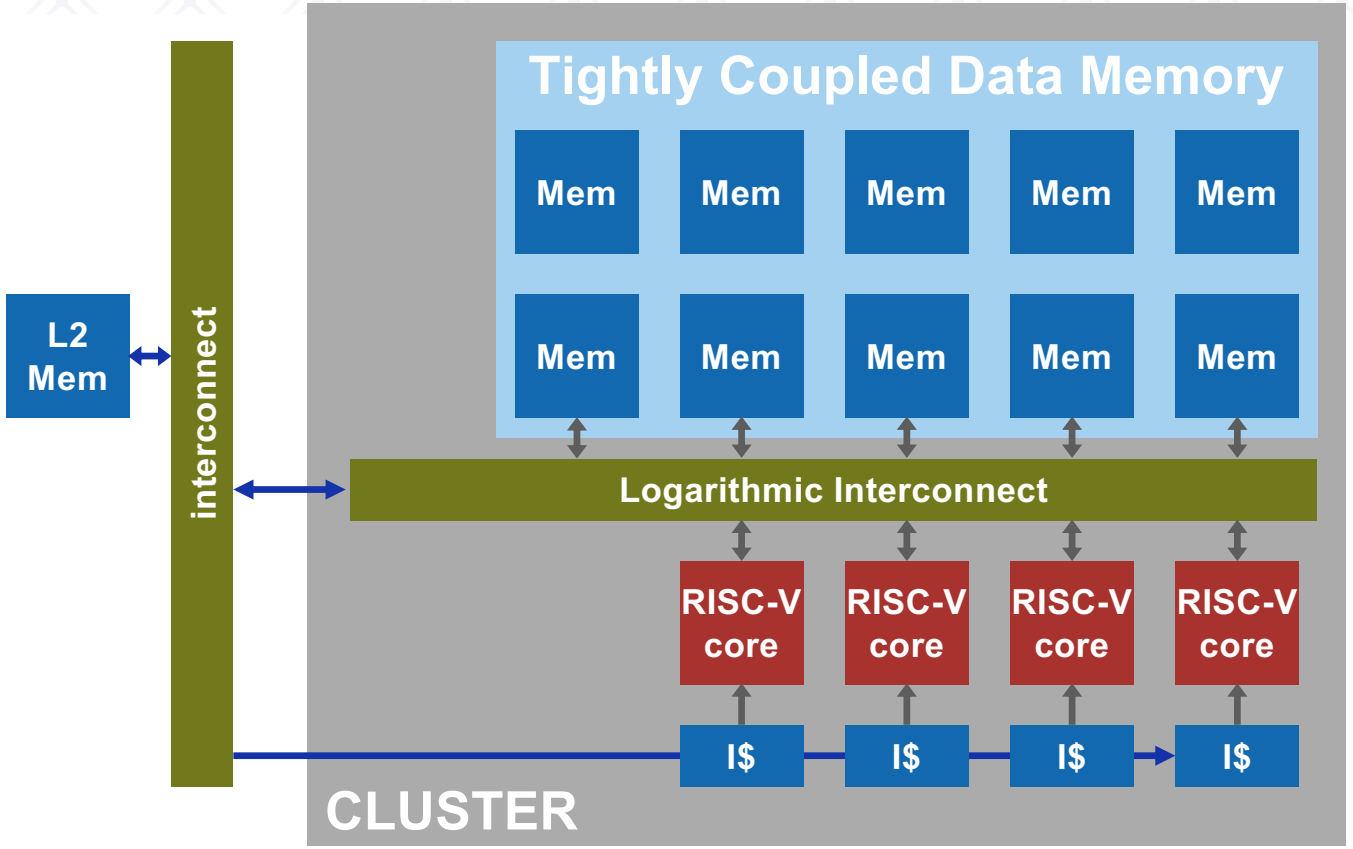


High speed single clock logarithmic interconnect

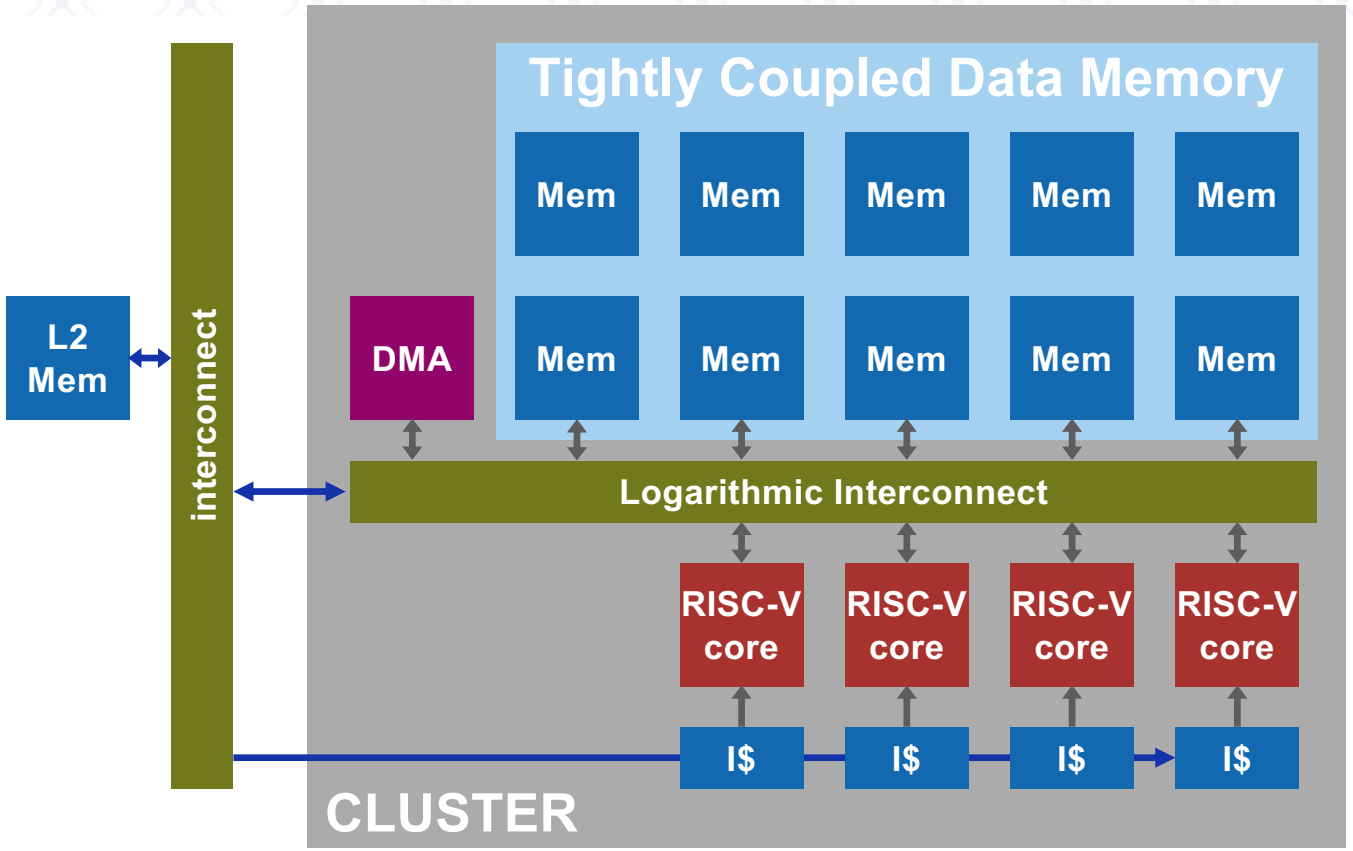


A. Rahimi, I. Loi, M. R. Kakoee and L. Benini, "A fully-synthesizable single-cycle interconnection network for Shared-L1 processor clusters," *2011 Design, Automation & Test in Europe*, 2011, pp. 1-6.

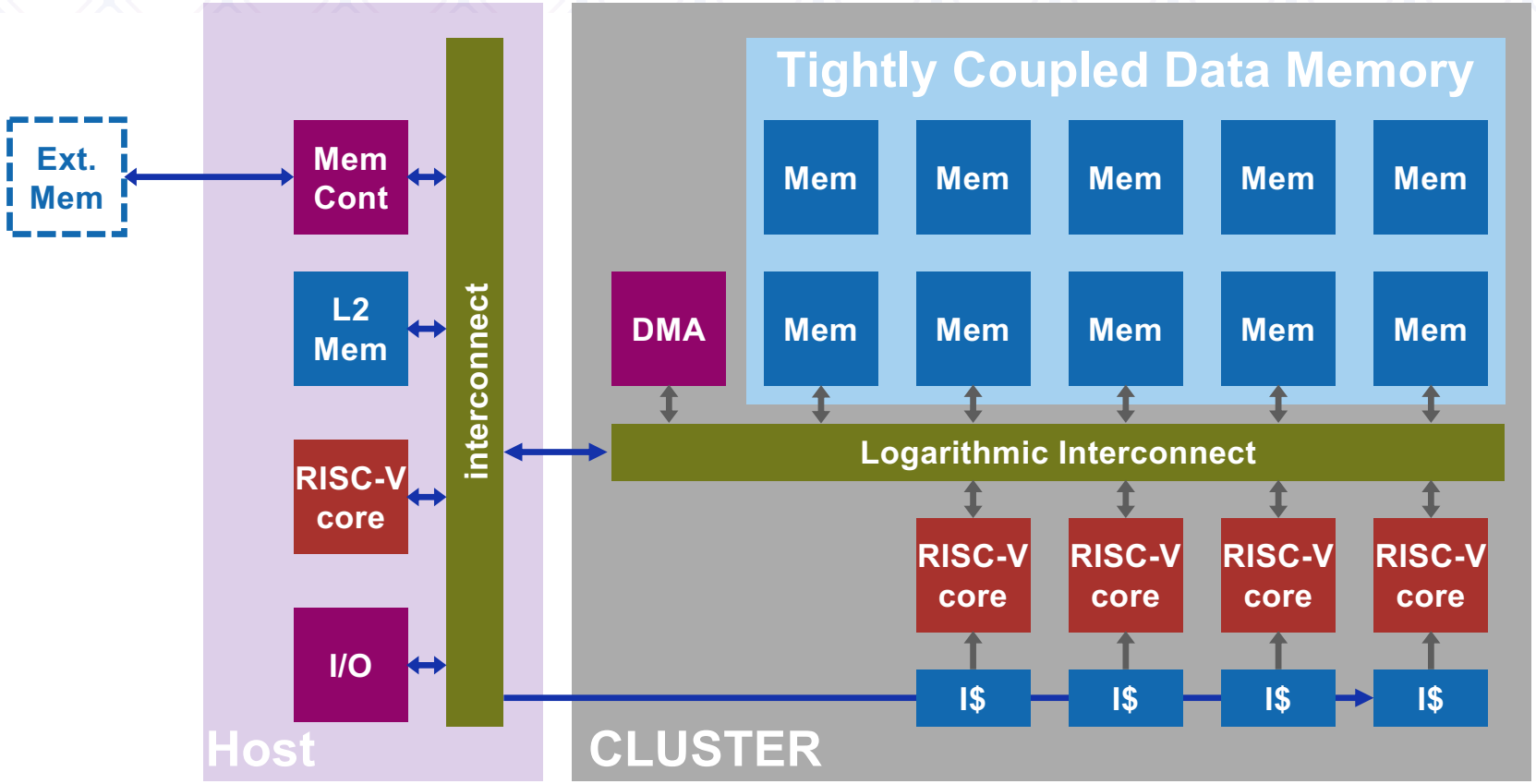
Shared instruction cache with private “loop buffer”



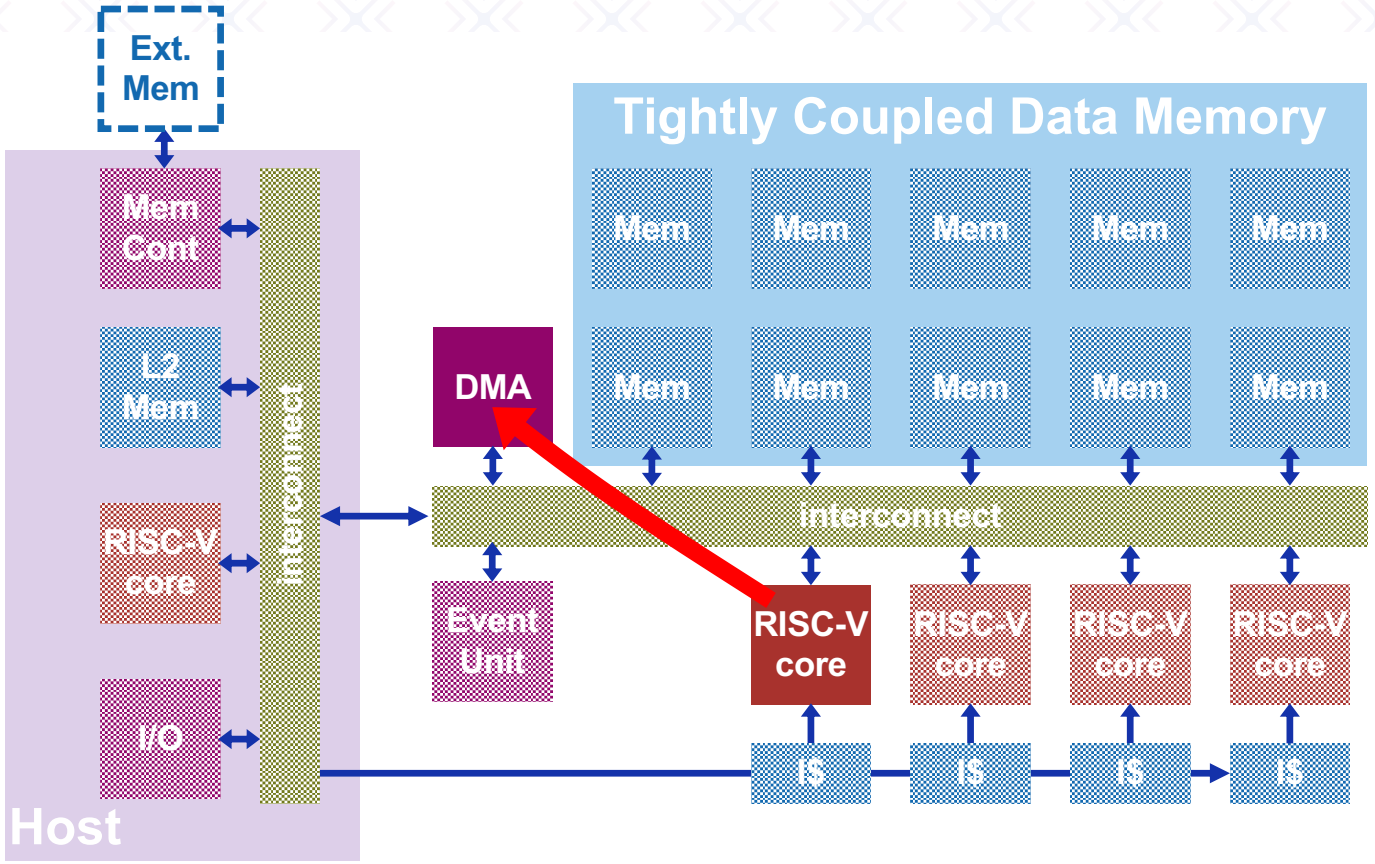
DMA for data transfers from/to L2



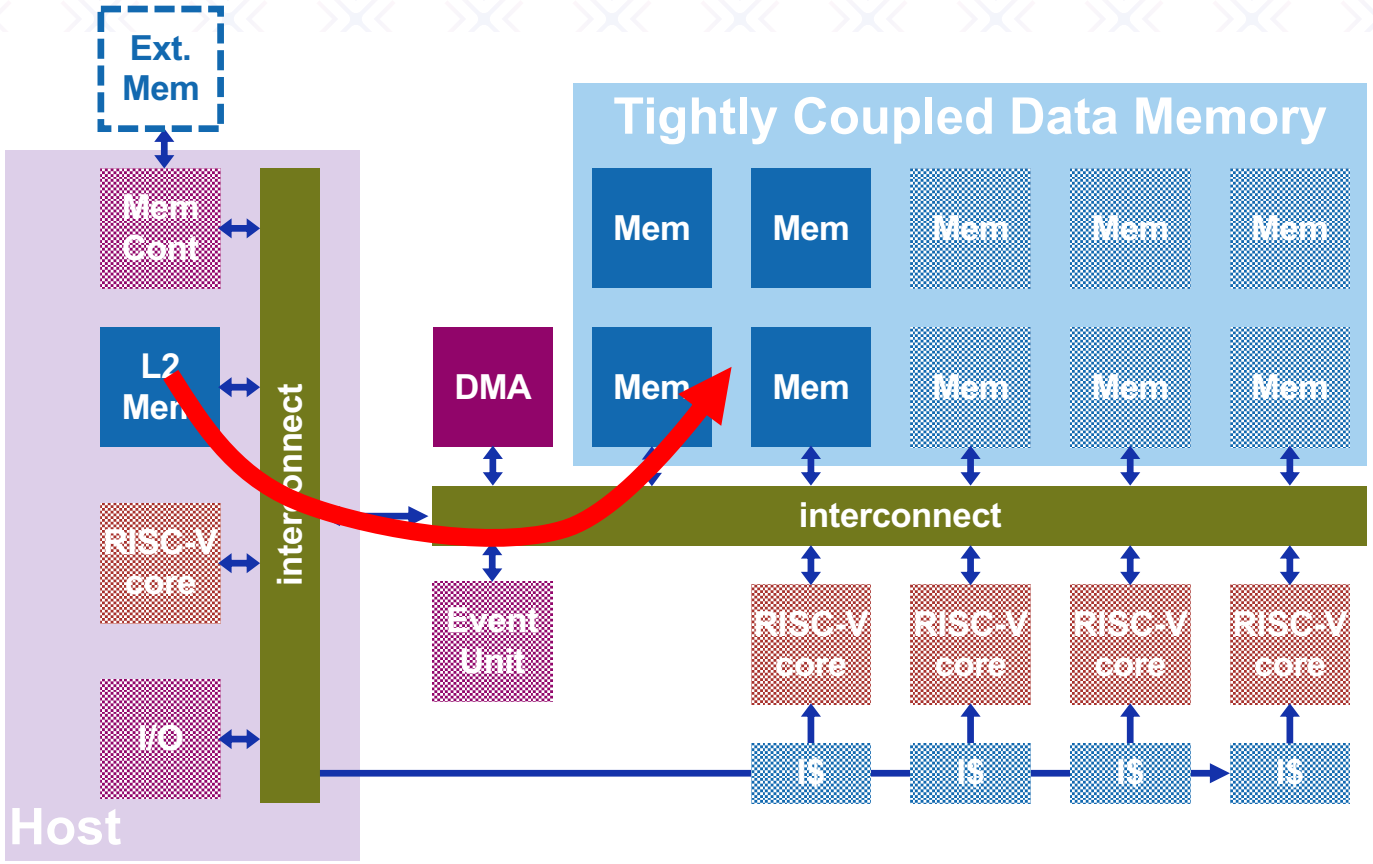
An additional Host controller is used for IO



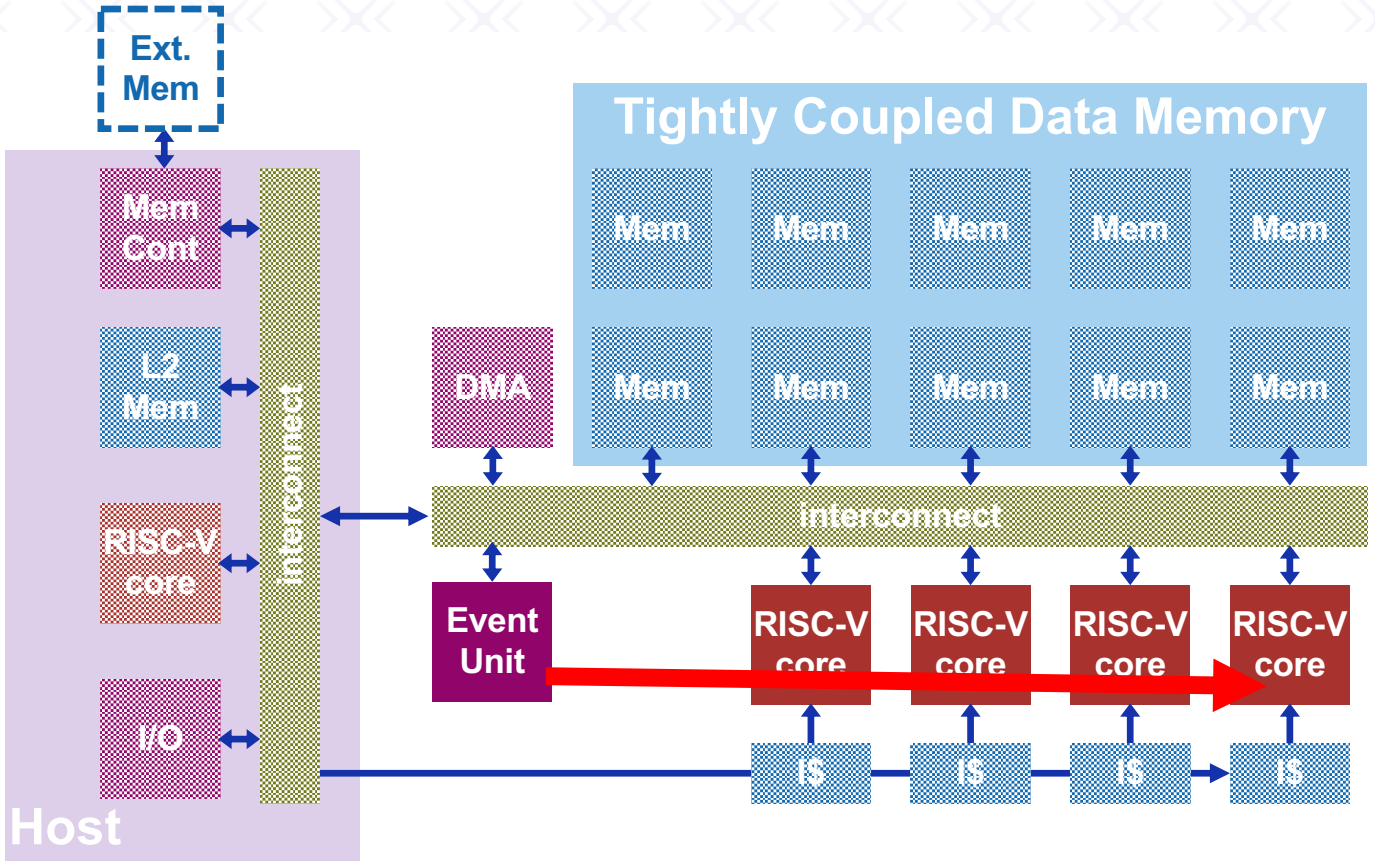
How do we work: Initiate a DMA transfer



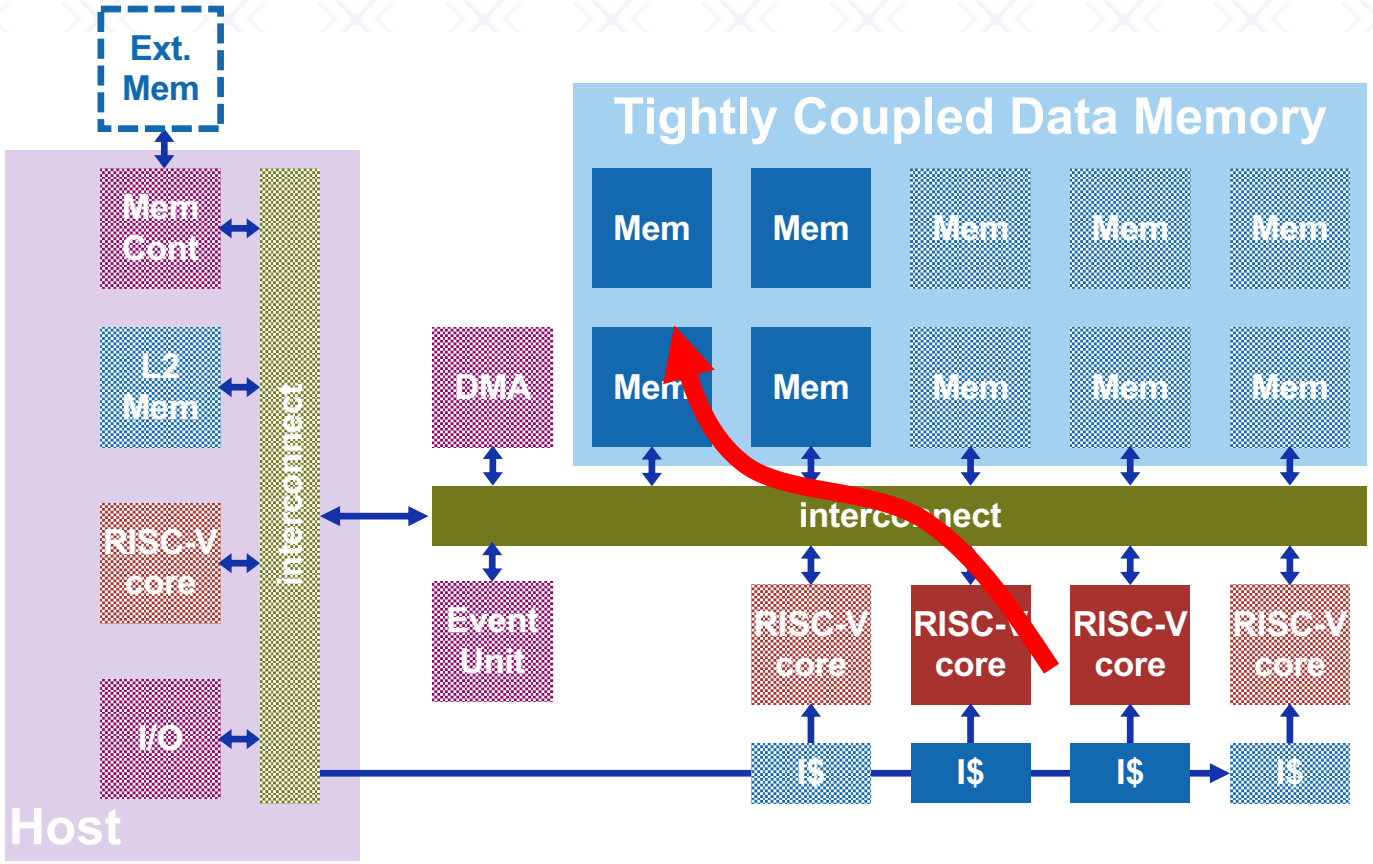
Data copied from L2 into TCDM



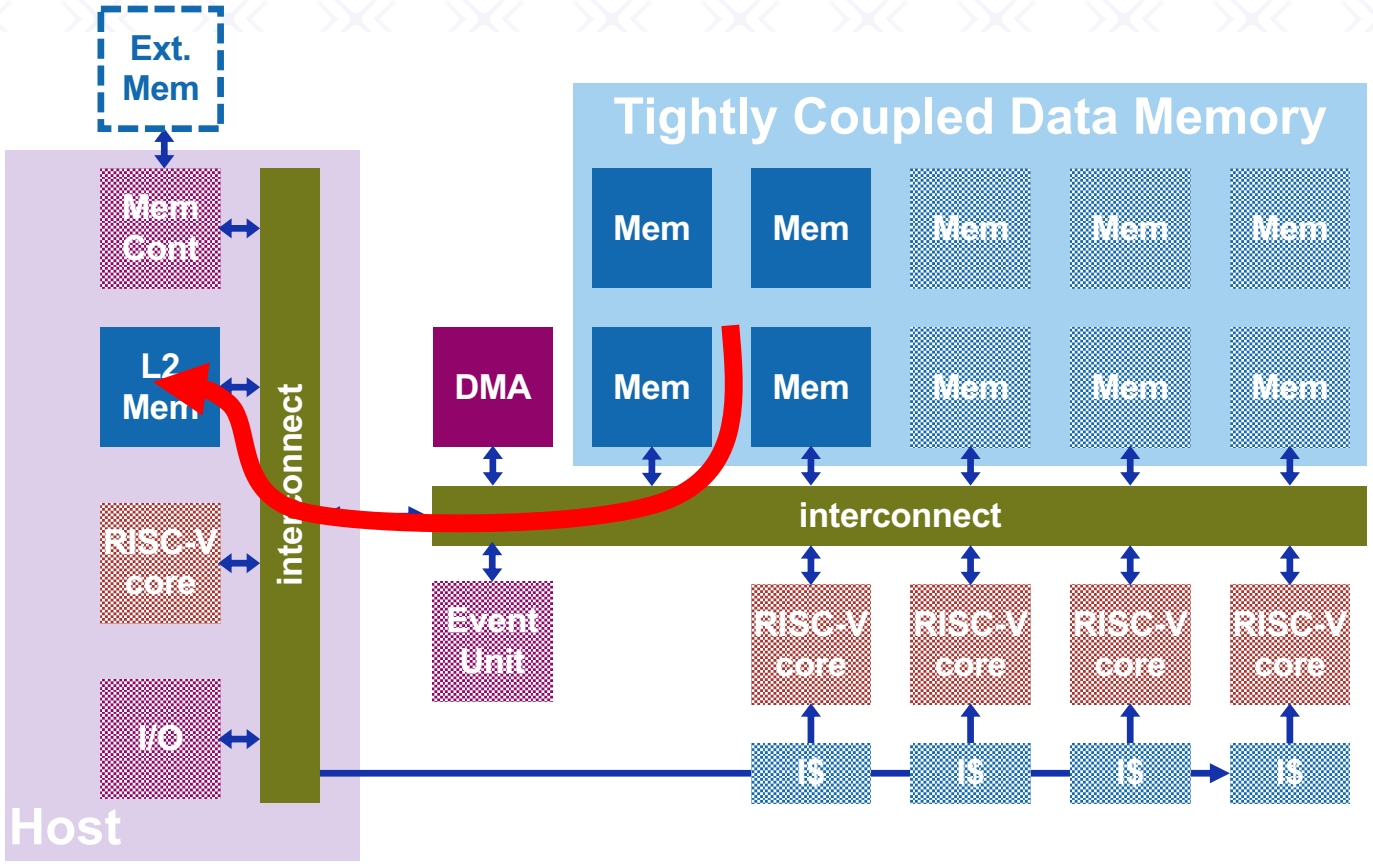
Once data is transferred, event unit notifies cores



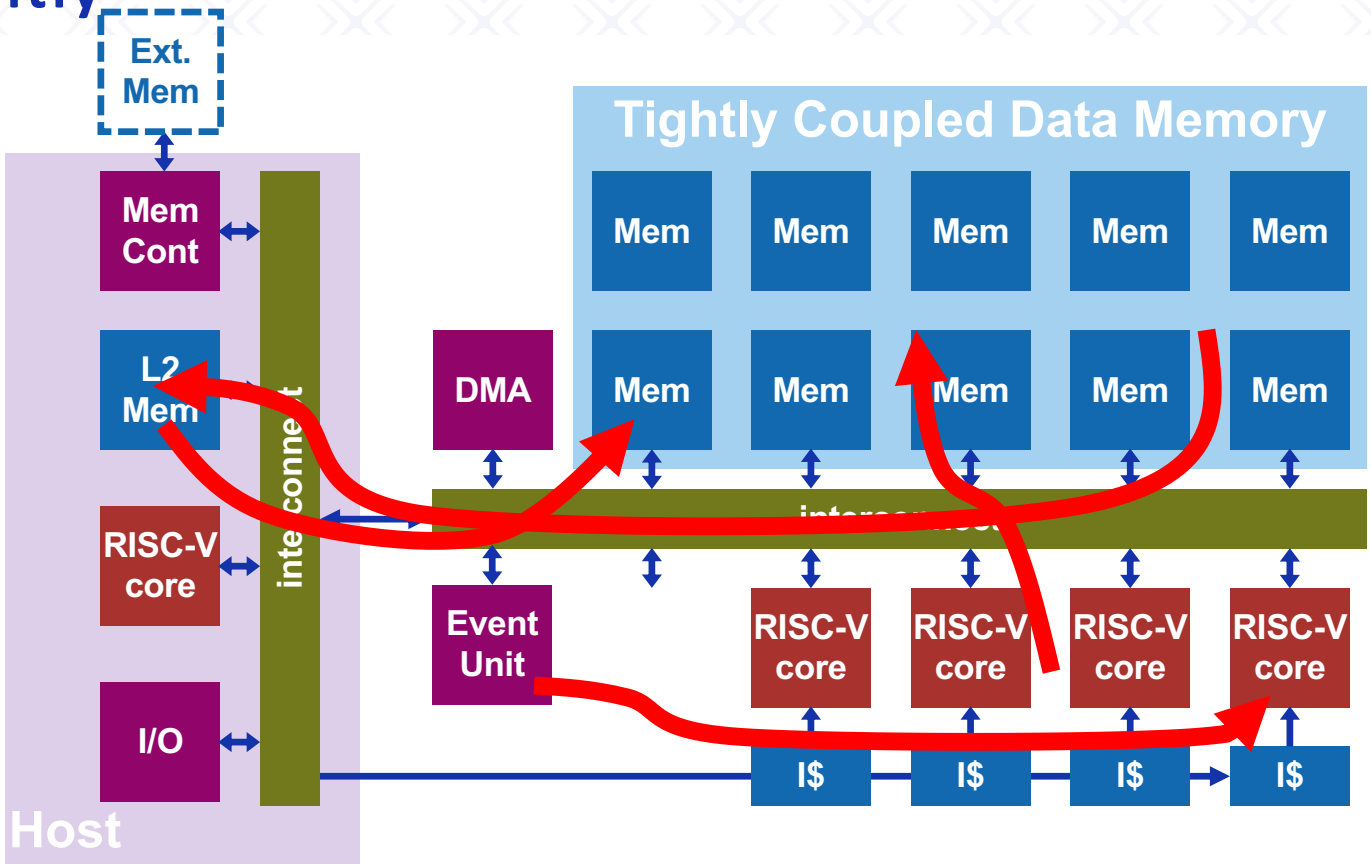
Cores can work on the data transferred



Once our work is done, DMA copies data back



During normal operation all of these occur concurrently



Not All Programs Are Created Equal

Chips-IT

- Processors can do two kinds of useful work:

Decide (jump through program parts)

- 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

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

Not All Programs Are Created Equal

Chips-IT

- Processors can do two kinds of useful work:

Decide (jump through program parts)

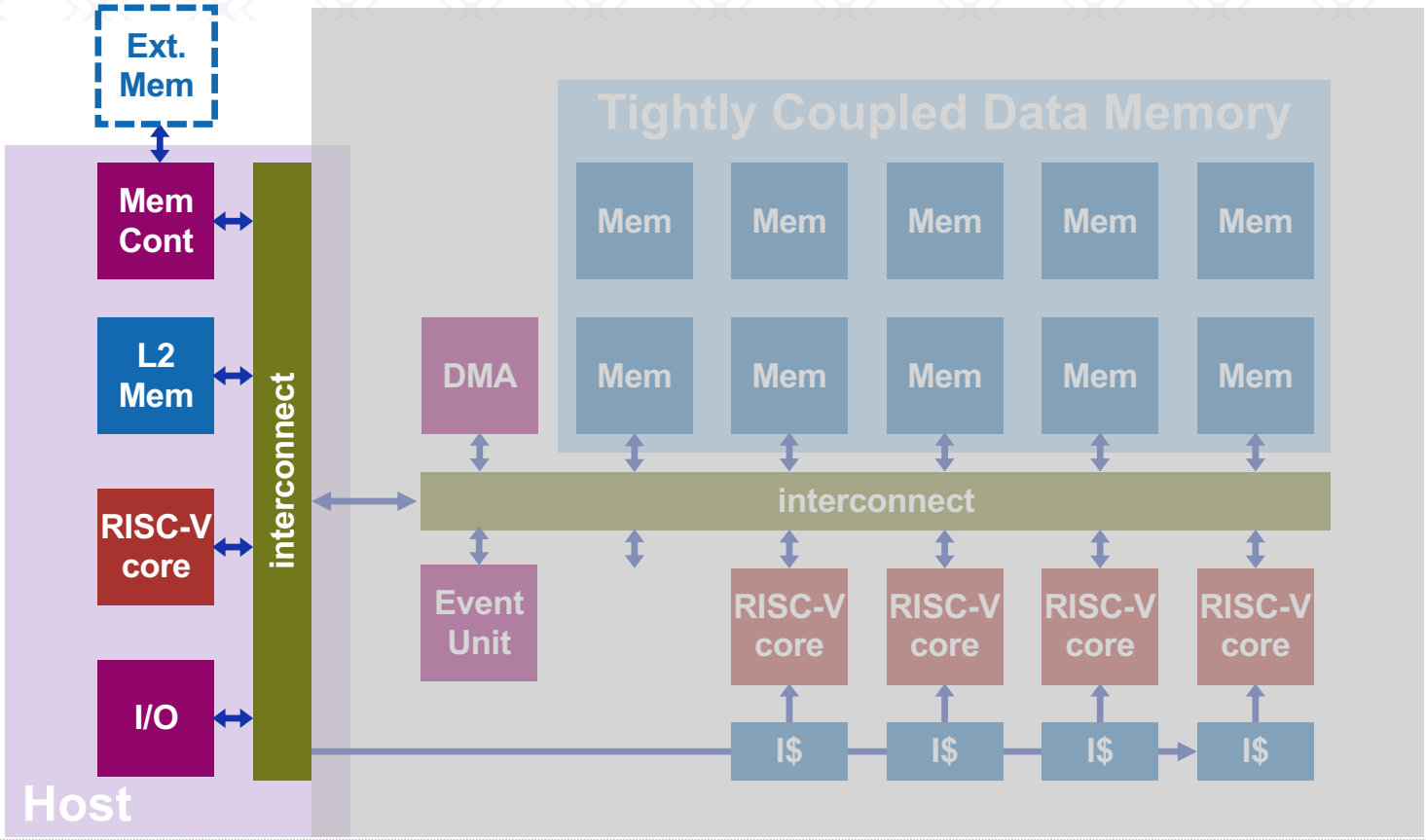
- Modulate flow of **instructions**
- **Smarts**:
  - Don't work too much
  - Be clever about the battles you pick (e.g., search in a database)
  - Lots of decisions
    Little number crunching

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

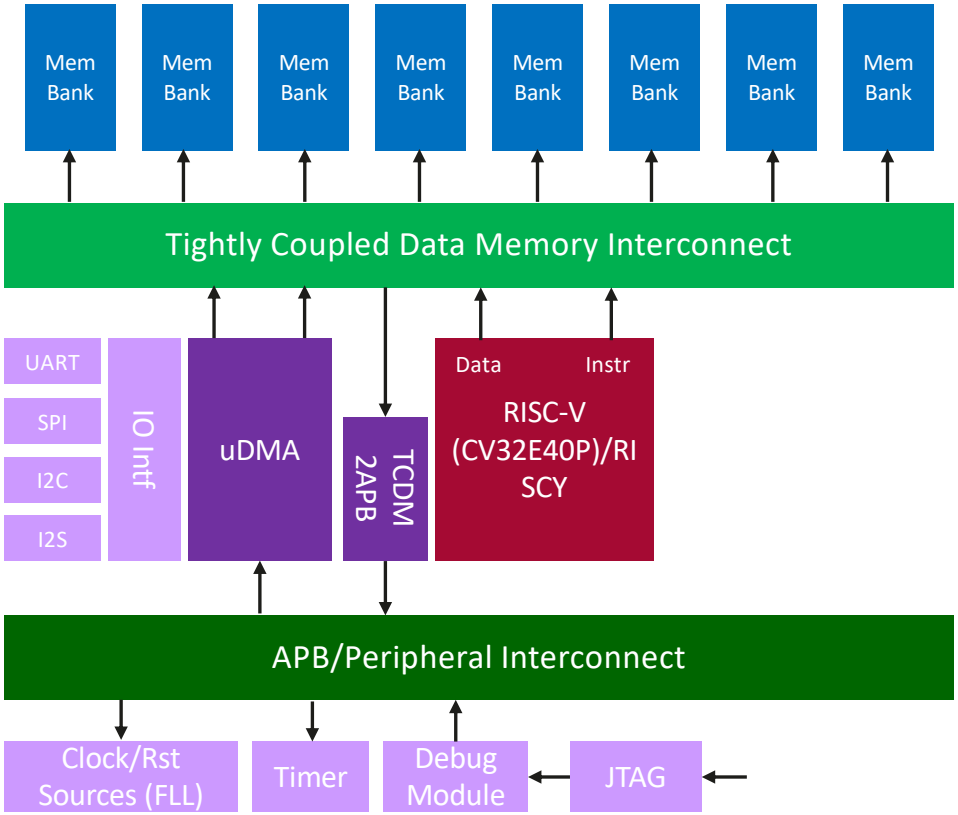
- But still decision making is crucial in computing!

Host controllers can be of various kind



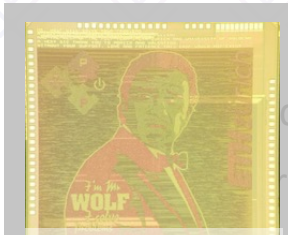
PULPissimo: embedded IoT controller

- Compact and tiny micro controller
- Extreme ultra-low power
- Simple control tasks and peripherals management

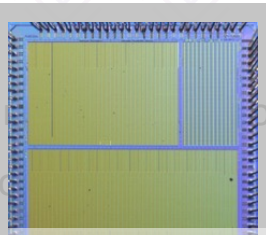


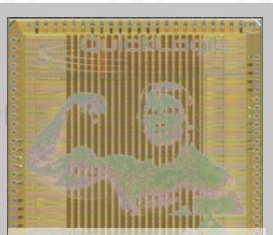
PULPissimo: embedded IoT controller

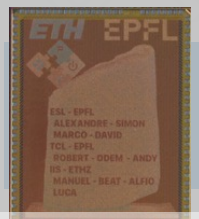
Chips-IT



Mr. Wolf 40nm

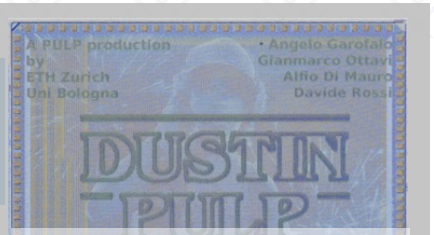

Poseidon 22nm


Arnold 22nm

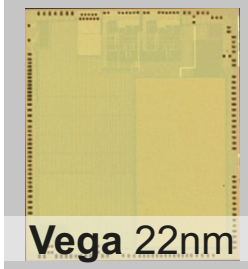


Rosetta 65nm

Xavier 65nm



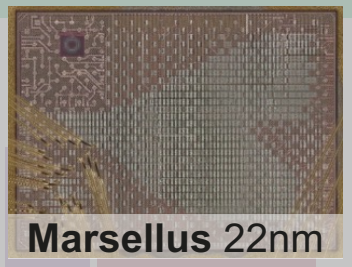
Dustin 65nm


Vega 22nm

Darkside 65nm

Echoes 65nm

Kraken 22nm

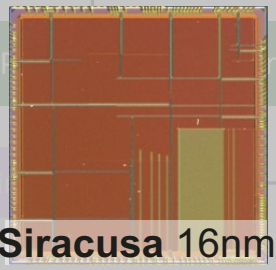

Marsellus 22nm

Cerberus 65nm

Eclipse 65nm

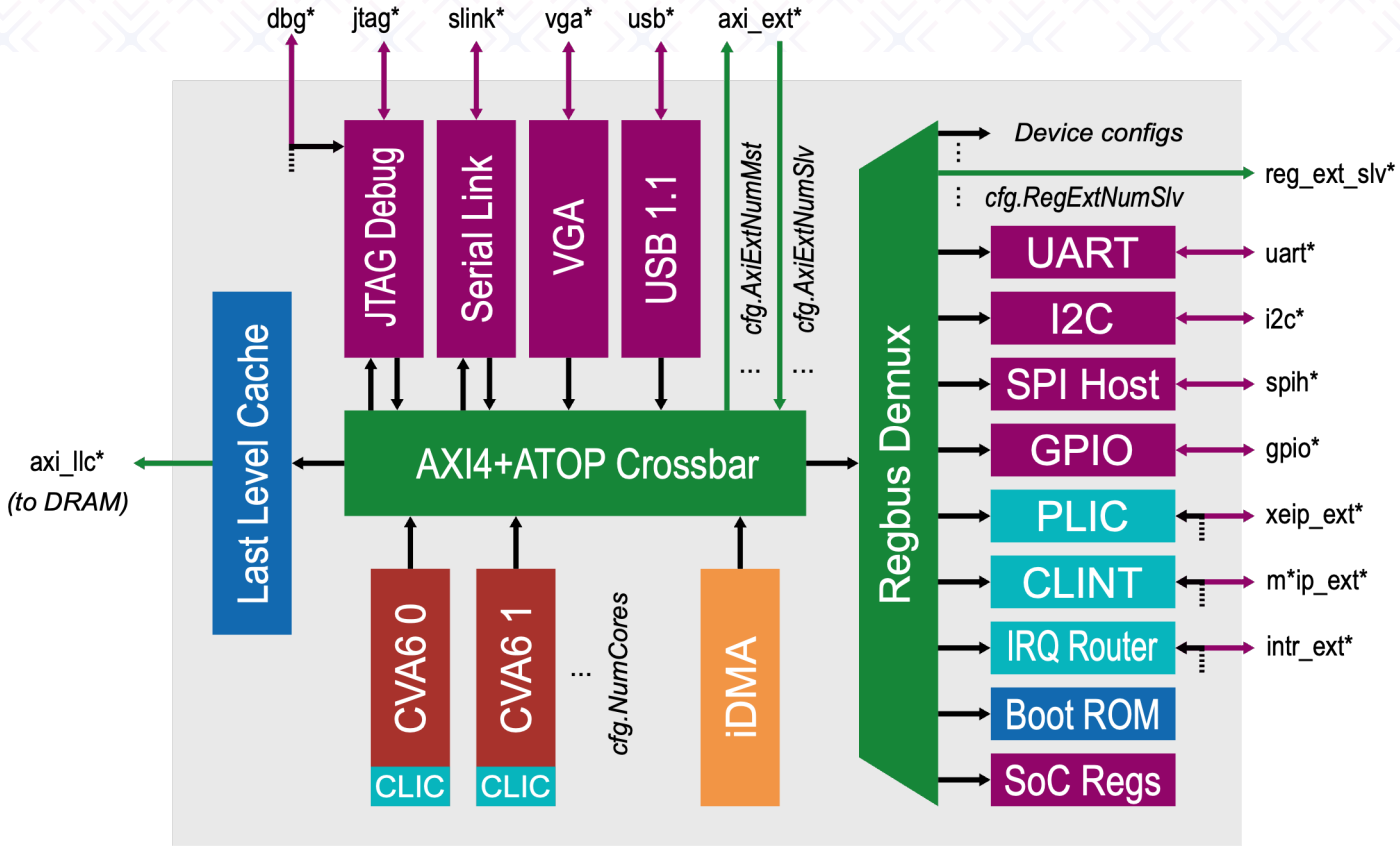
Kairos 65nm

Trikarenos 65nm


Siracusa 16nm

...

Cheshire: Not only extreme edge IoT

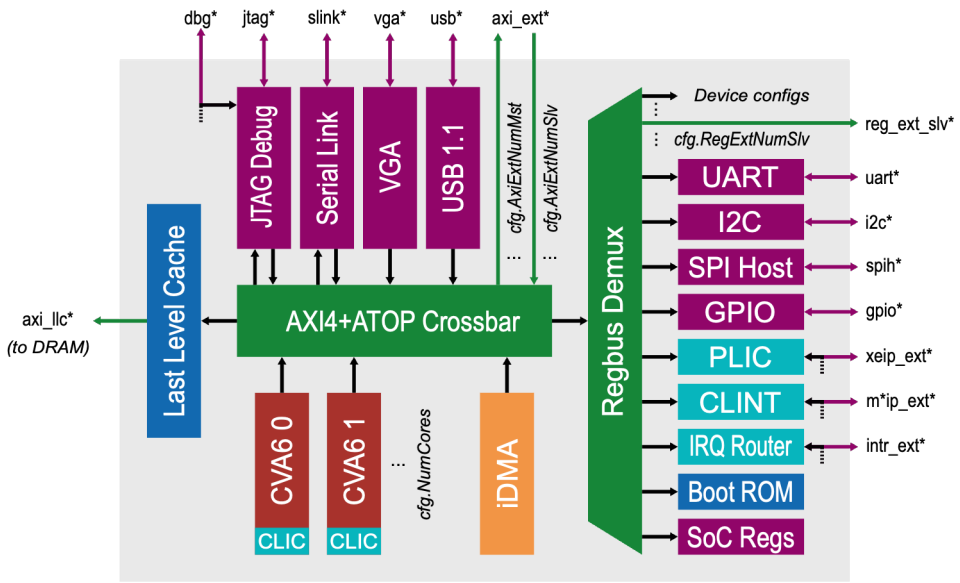


Cheshire: Not only extreme edge IoT

Chips-IT

Modular Platform

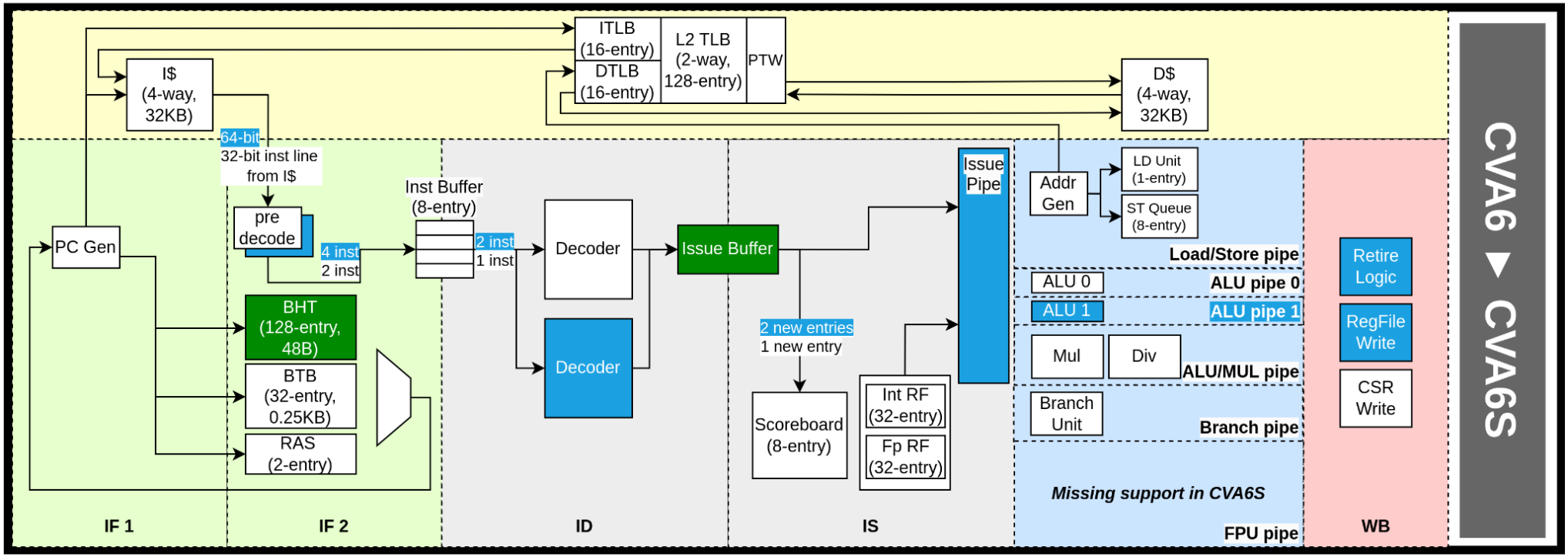
- Based on 64-bit Application-class processor
- Linux-capable and configurable for a wide range of applciations
- Standard interrupt, debug, and memory interfaces
- Access to external DRAMs (HyperRAMs)



CVA6(S+): Application-class 64-bit RISC-V core

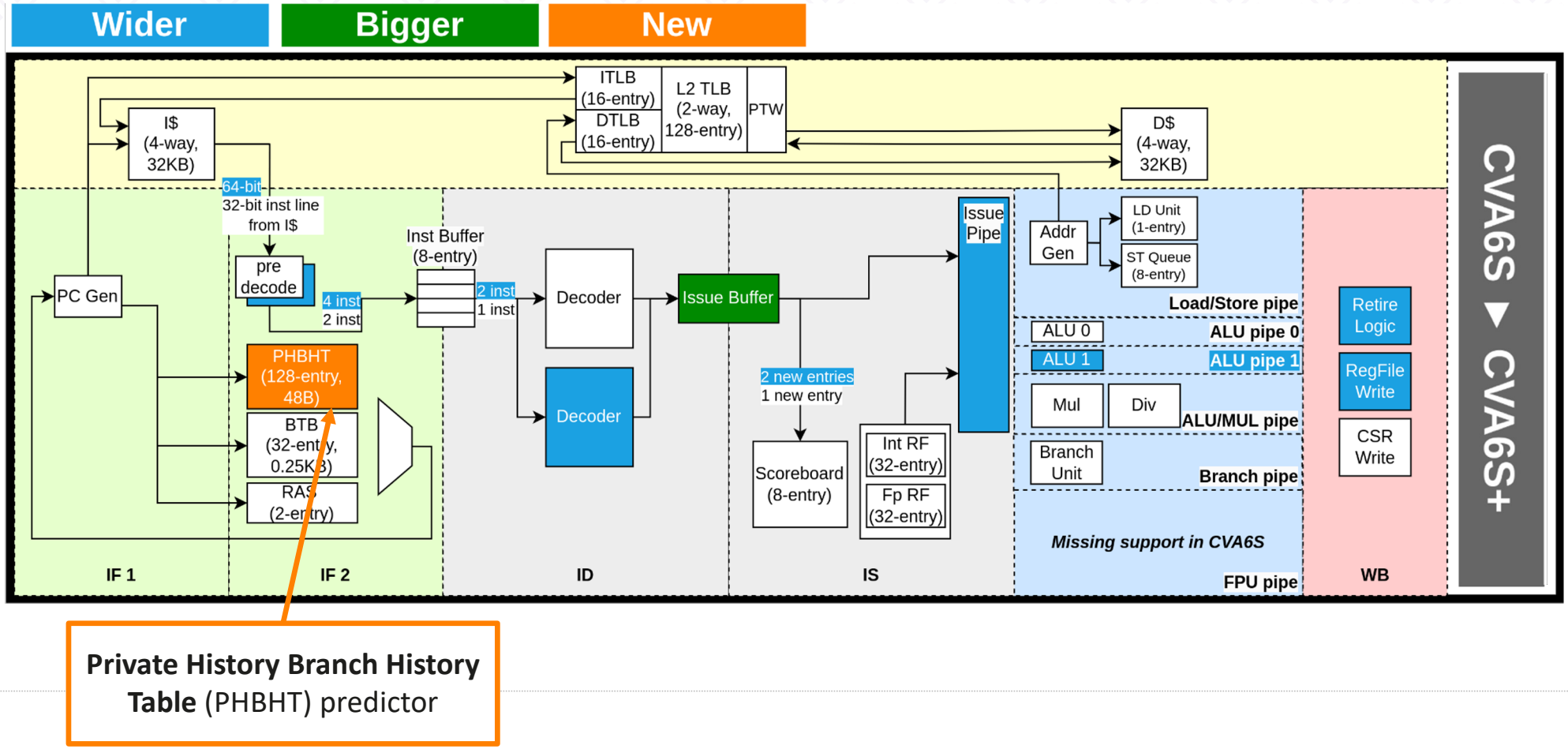
Chips-IT

Wider **Bigger**



CVA6(S+): Application-class 64-bit RISC-V core

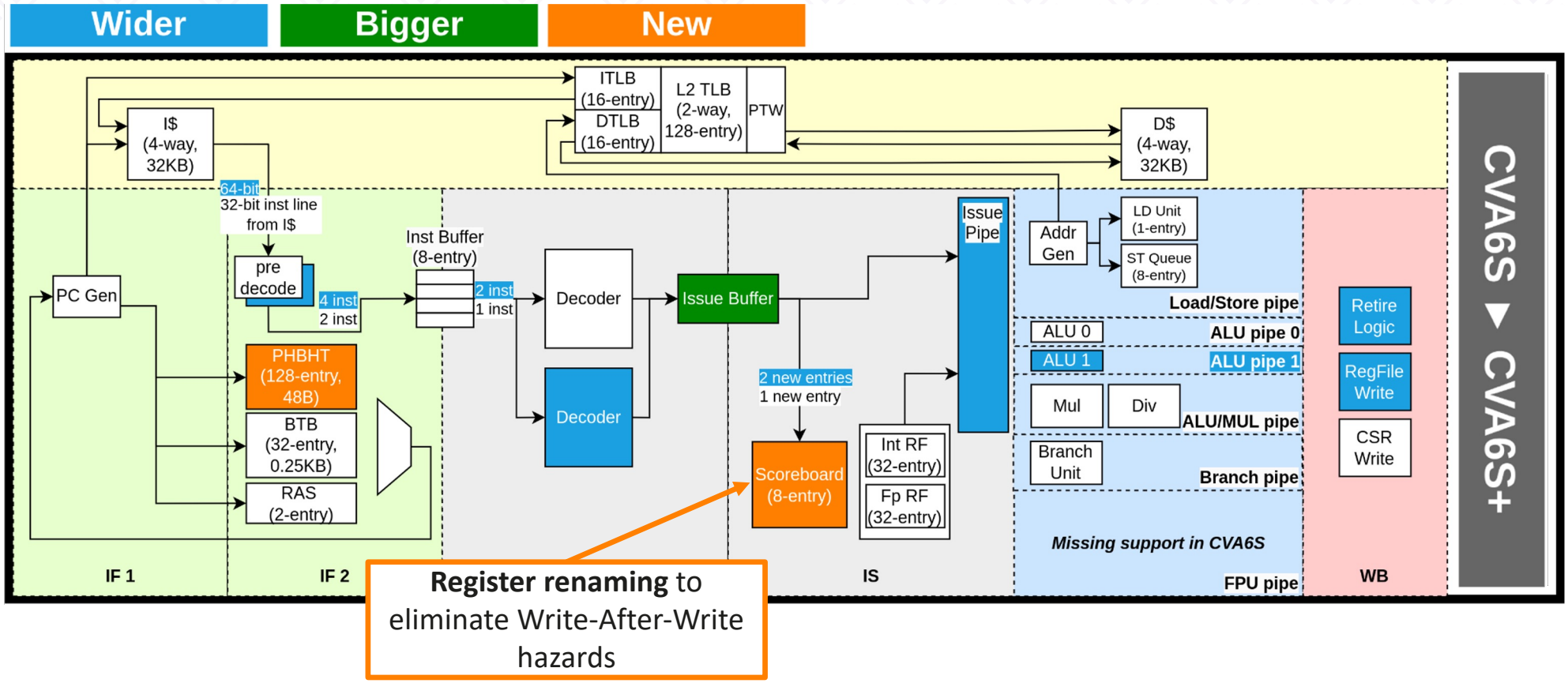
Chips-IT



Private History Branch History Table (PHBHT) predictor

CVA6(S+): Application-class 64-bit RISC-V core

Chips-IT



CVA6(S+): Application-class 64-bit RISC-V core

Chips-IT