



# **The EPI vector RISC-V accelerator: Architecture, platform, tools and hands-on**

Filippo Mantovani\*, Fabio Banchelli, Pablo Vizcaino  
Barcelona Supercomputing Center (BSC)

# OUTLINE

1. Intro to EPAC and RISC-V vector tile
2. About “co-design”
3. The Software Development Vehicles (SDV)

Q&A

Break

4. SDV tutorial / Hands-on



# **EPAC: European Processor Initiative ACcelerator**

# EPI MAIN OBJECTIVE

To develop European microprocessor and accelerator technology

- Strengthen competitiveness of EU industry and science

SiPearl,  
Atos, CEA,  
UniBo, E4,  
UniPi, P&R, ...

**Rhea**

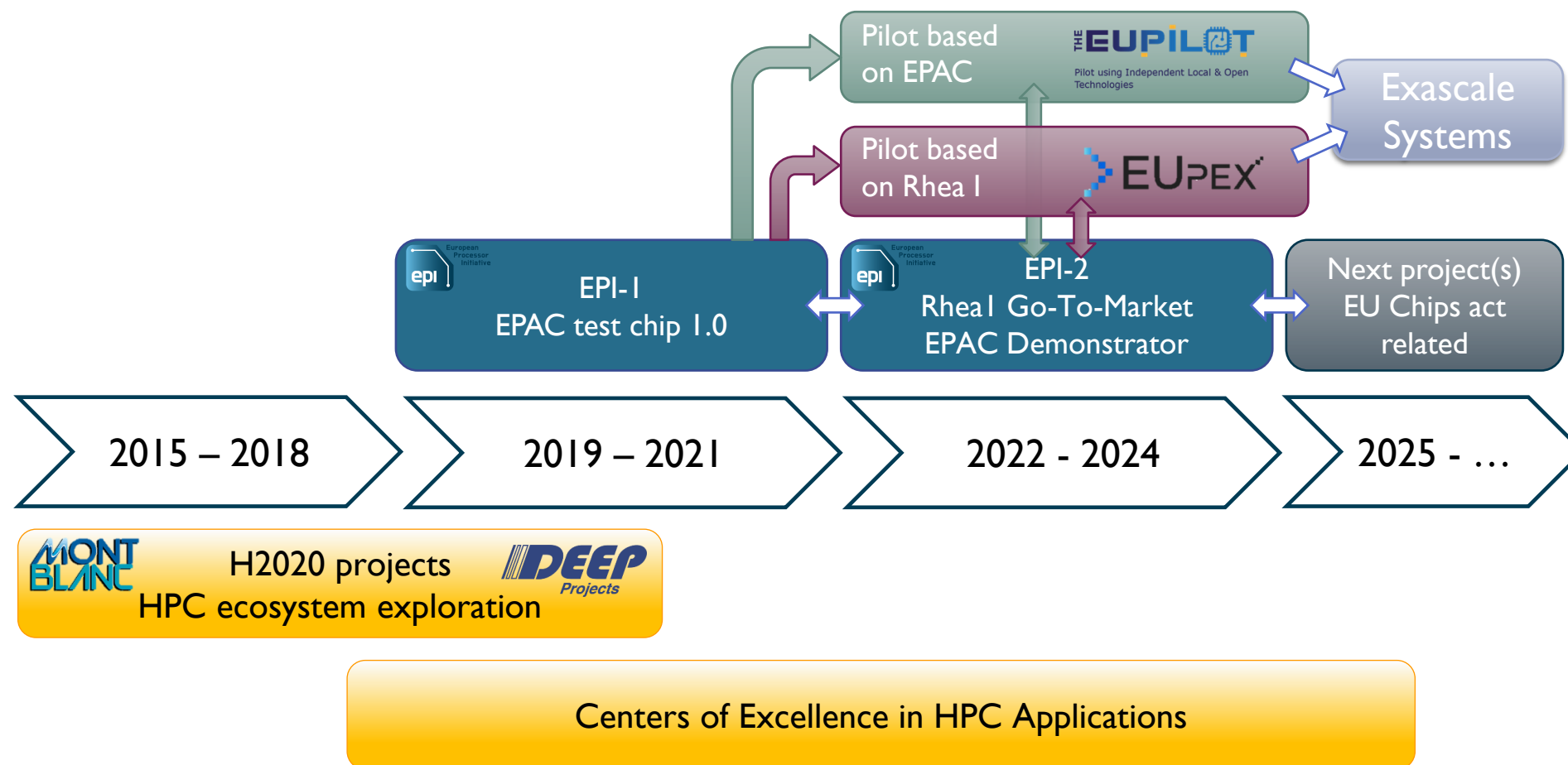
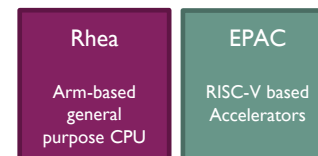
Arm-based  
general purpose  
CPU

**EPAC**

RISC-V based  
Accelerators

BSC, SemiDynamics,  
EXTOLL, FORTH,  
ETHZ, UniBo, UniZG,  
Chalmers, CEA, E4,  
Menta, ZPT, ...

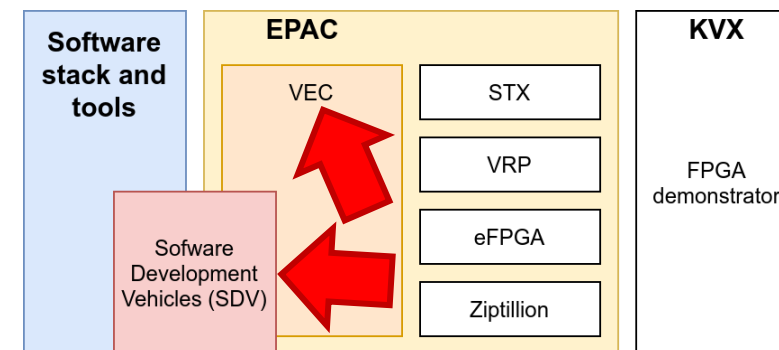
# OVERALL TECHNOLOGY ROADMAP



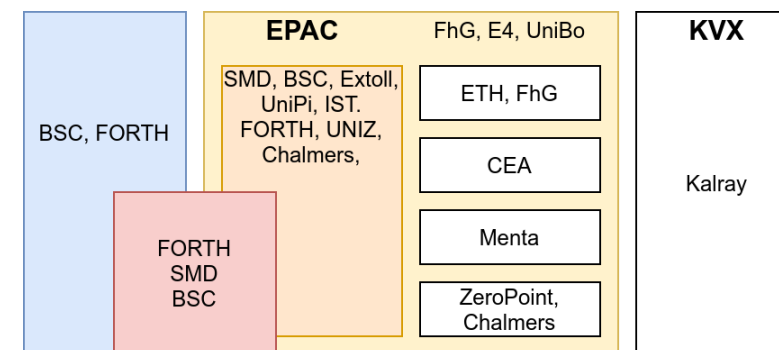
# EPAC: VISIONS AND COLLABORATIONS

- **VEC** - Self-hosted RISC-V CPU + wide VPU (256 double elements) supporting RVV 0.7.1 / 1.0
- **STX/SPARTA** - RISC-V CPU + specific cores for stencil and neural network computation
- **VRP** - RISC-V CPU with support for variable precision arithmetic (data size up to 512 bit)
- **eFPGA** - On-chip reconfigurable logic
- **Ziptilion** - IP compressing/decompressing data to/from the main memory
- **KVX** - FPGA demonstrator of the Kalray RISC-V CPU targeting HPC and ML

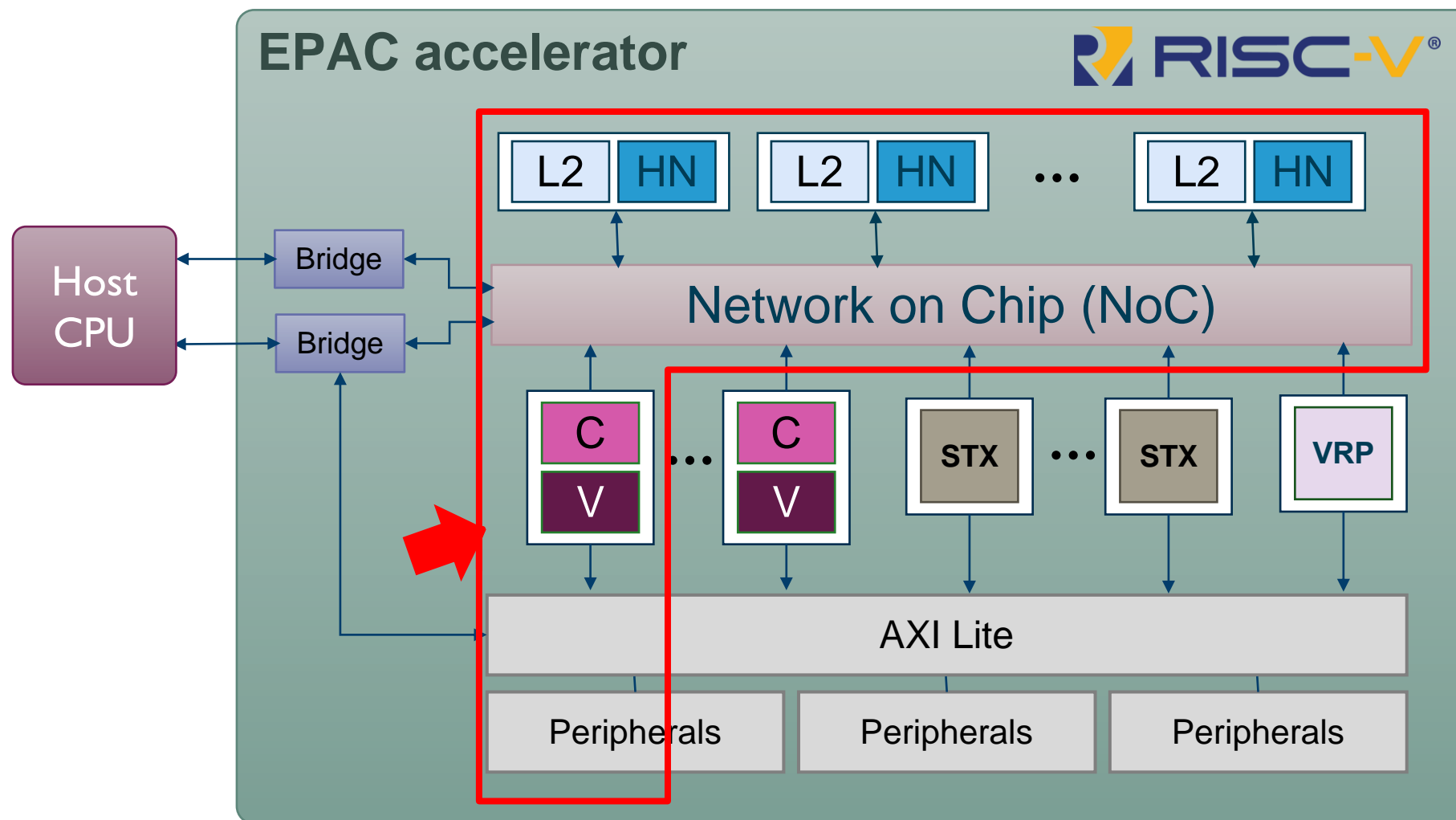
## VISION



## COLLABORATION



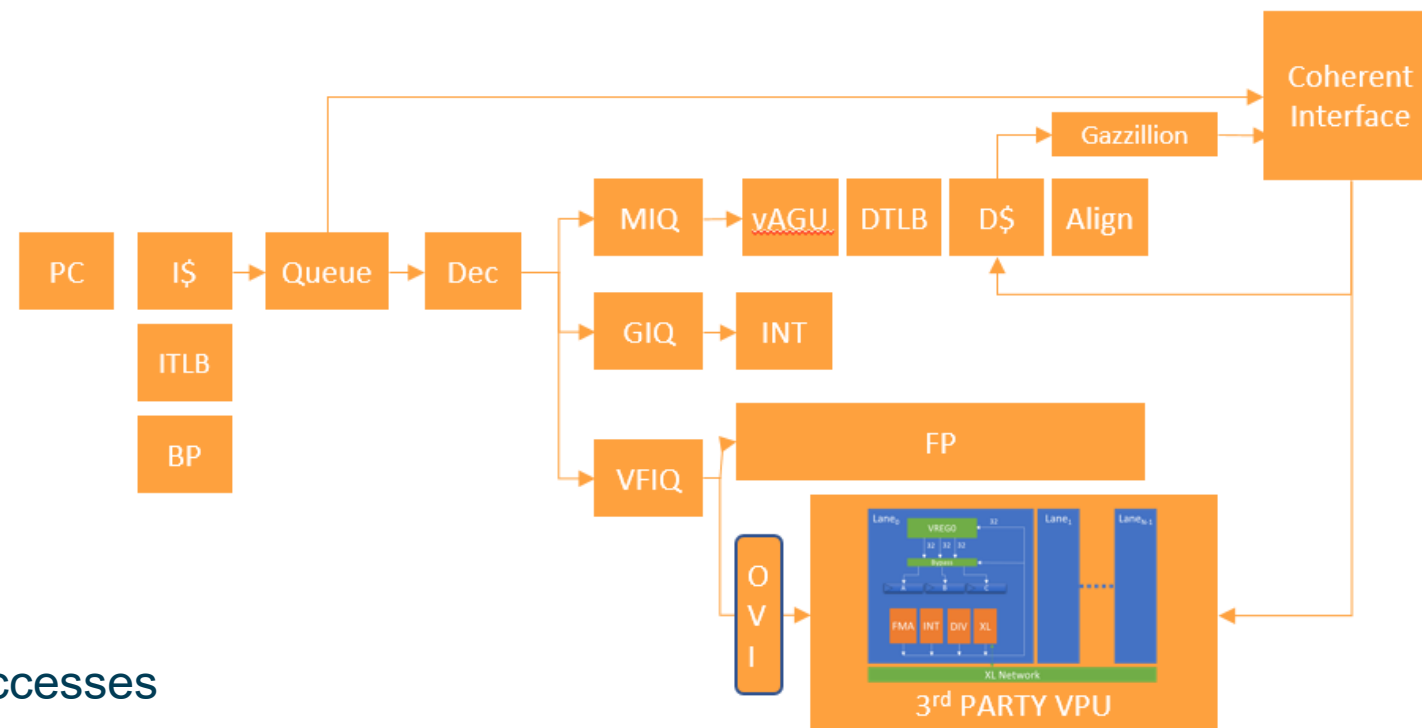
# EPAC: A RISC-V ACCELERATOR



# EPAC-VEC: AVISPADO 220 WITH VPU

## RISCV64GCV

- 16 kB instruction cache
- 32 kB data cache
- Decodes v0.7, v1.0 vector extension
- Full hardware support for unaligned accesses
- Cache coherent (CHI)
- Vector memory accesses (vle, vlse, vlxe, vse, ...) processed by a dedicated queue (MIQ/LSU)



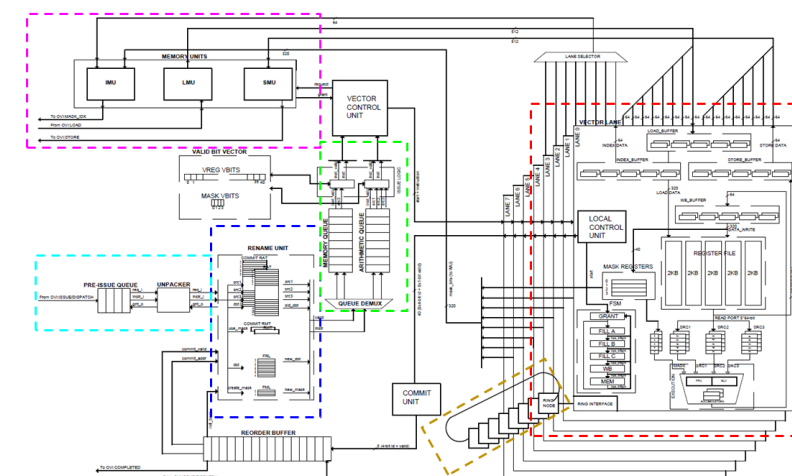
Courtesy:  **semidynamicS**  
silicon design and verification services



# EPAC-VEC: VECTOR PROCESSING UNIT (VITRUVIUS)

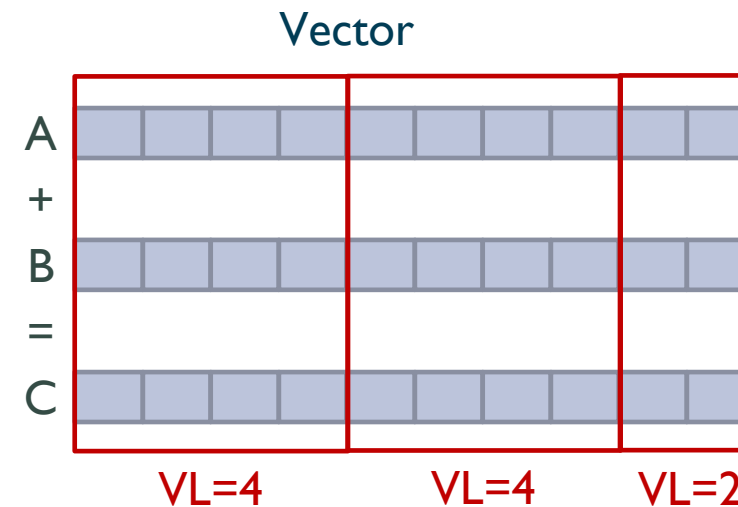
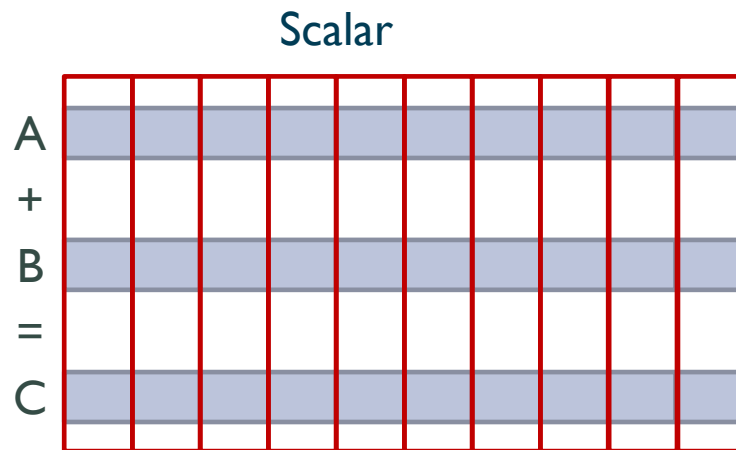
Architecture	Vector register size (1 cell = 1 double element)																								
Intel AVX512	D1	D2	D3	D4	D5	D6	D7	D8																	
Arm Neon	D1	D2																							
Arm SVE @ A64FX	D1	D2	D3	D4	D5	D6	D7	D8																	
NEC Aurora SX	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12	D13	D14	D15	D16	...								
RISC-V EPAC Vec	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12	D13	D14	D15	D16	...								

- Implementation
  - Long vectors: 256 DP elements
    - #Functional Units (FUs) << Vector Length (VL)
    - 1 vector instruction can take several (32) cycles
  - 8 Lanes per core
    - FMA/lane: 2 DP Flop/cycle
  - 40 physical registers, some out of order
  - Vector length agnostic (VLA) programming and architecture



# WHAT'S SPECIAL?

- The scalar in-order RISC-V core can release several requests of cache lines to the memory
- The core is connected to a Vector Processing Unit (VPU)

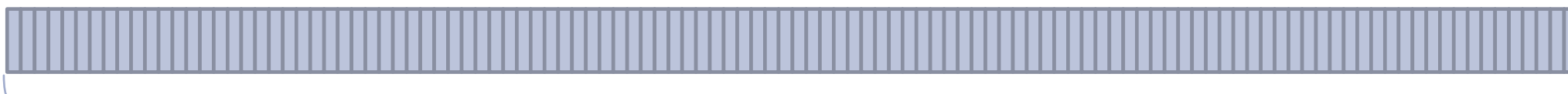


# HOW WIDE IS “YOUR” VECTOR?

Intel AVX 512



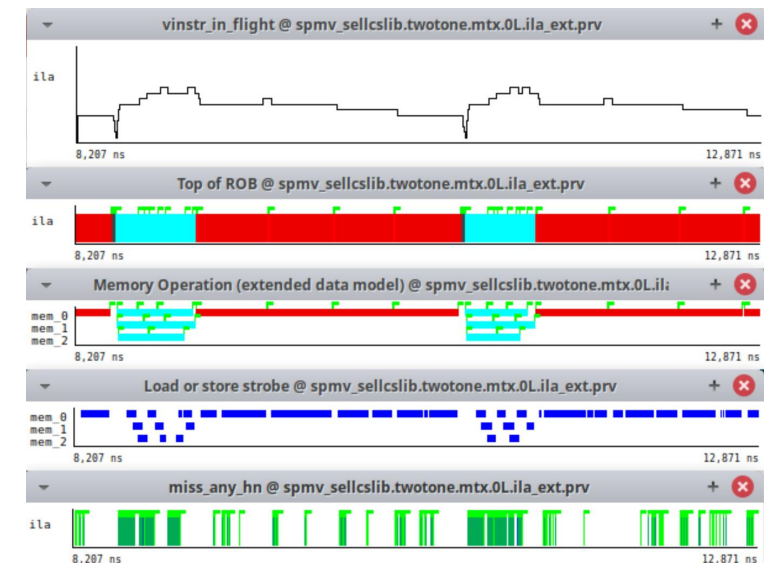
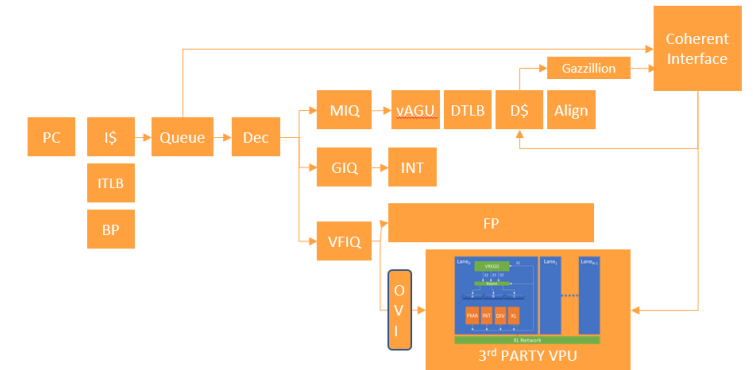
RISC-V EPAC VEC



256 DP elements  
16 kbits

# WHAT'S SPECIAL? – PART 2

- Vector instructions
  - Less instructions (including scalar instructions for controlling a loop)
- Several cycles for a single vector instruction
  - Enables overlap: other functional units can do useful work meanwhile
  - Makes easier to keep all functional units busy
- Vector accelerator
  - Launch a vec. Instruction ~= Launch a kernel (in GPU terms)
  - i.e., a few cycle for decoding vs. several cycles for firing up a thread
- Coalescing on load instructions
  - Compared to scalar flow, pay overhead of load instruction start-up only once
  - Saturate the memory bandwidth with less cores

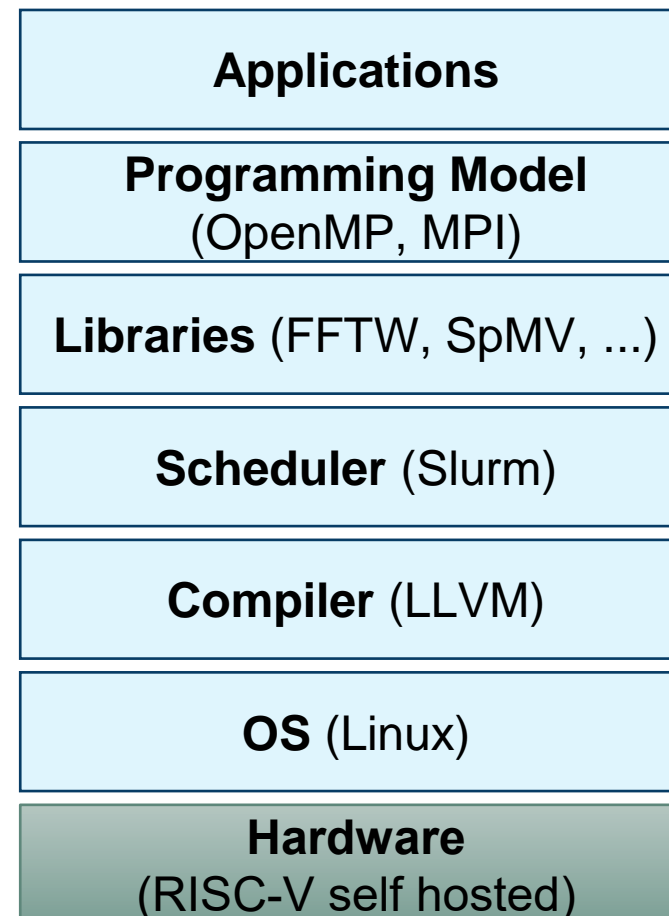


We have tools to measure each of these effects

# HOW DO I PROGRAM IT?

Like a standard HPC system!

- Compile your code
  - Yes, we give you a compiler!
- Link libraries
- Write a job script
  - It's SLURM you already know it!
- Submit a job to the queue
- Wait for the results
- Analyse execution traces and study how well your code is vectorized
  - Sure, we help you!



# HOW TO USE THE V-EXTENSION?

- **Assembler**
  - Always a valid option but not the most pleasant
- **C/C++ builtins (intrinsics)**
  - Low-level mapping to the instructions
  - Allows embedding it into an existing C/C++ codebase
  - Allows relatively quick experimentation
- **#pragma omp simd** (aka “Guided vectorization”)
  - Relies on vectorization capabilities of the compiler
    - Usually works but gets complicated if the code calls functions
  - Also usable in Fortran
- **Autovectorization**
  - Leave it to the compiler

# RISC-V VECTOR EXTENSION (RVV) COMPILER

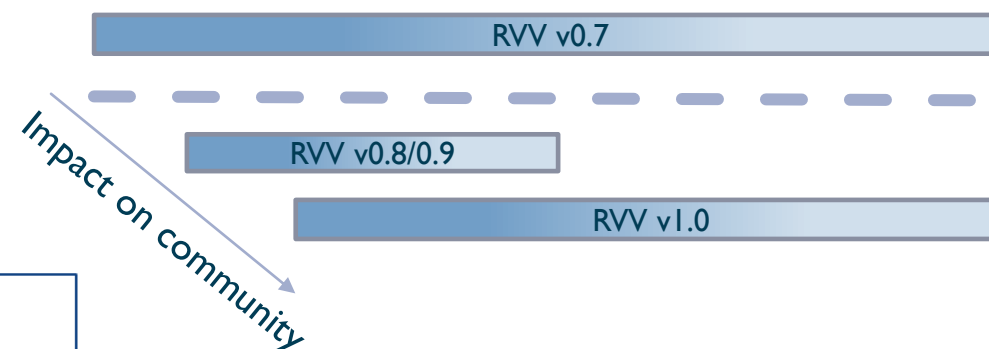
- LLVM support for the evolution of the RISC-V Vector (RVV) Extension
- Intrinsics
- Autovectorization

```
void SpMV_vec(double *a, long *ia, long *ja, double *x, double *y, int nrows) {
    for (int row = 0; row < nrows; row++) {
        int nnz_row = ia[row + 1] - ia[row];
        unsigned long gvl; // granted vector lengths
        int idx = ia[row];
        __epi_1xf64 v_a, v_x, v_prod, v_partial_res;
        __epi_1xi64 v_idx_row, v_three;
        y[row] = 0.0;
        v_partial_res = __builtin_epi_vbroadcast_1xf64(0.0, gvl);
        for (int colid = 0; colid < nnz_row; ) { // blocking on MAXVL
            gvl = __builtin_epi_vsetvl(nnz_row - colid, __epi_e64,
                v_a = __builtin_epi_vload_1xf64(&a[idx+colid], gvl);
                v_idx_row = __builtin_epi_vload_1xi64(&ja[idx+colid], gvl);
                v_three = __builtin_epi_vbroadcast_1xi64(3, gvl);
                v_idx_row = __builtin_epi_vsll_1xi64(v_idx_row, v_three, gvl);
                v_x = __builtin_epi_vload_indexed_1xf64(x, v_idx_row, g
                v_prod = __bu
                v_partial_res =
            }
            colid += gvl;
        }
        y[row] += __builtin_
    }
}

void target_inner_3d (...) {
    for (i) {
        for (j) {
            #pragma clang loop vectorize(enable)
            for (k) {
                lap = LAP;
                v[IDX3_1(i,j,k)] = 2.f*u[IDX3_1(i,j,k)]
                    -v[IDX3_1(i,j,k)]+vp[IDX3(i,j,k)]*lap;
            }
        }
    }

    float complex A[n][n];
    float complex templ, temp2;

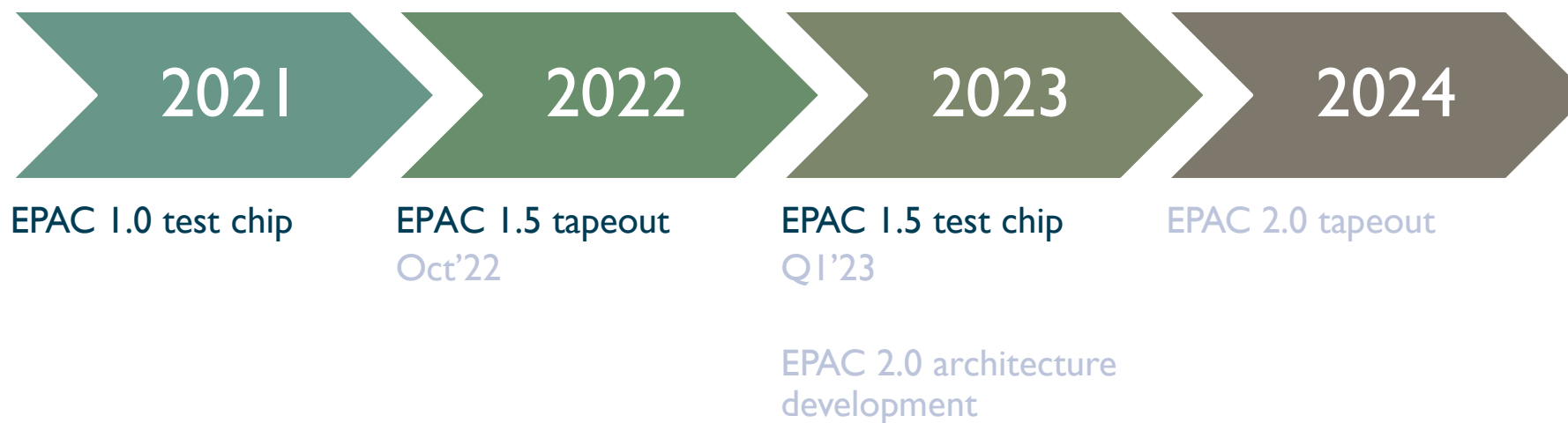
    for (int j = 0; j < n; j++) {
        if (x[j] != ZERO || y[j] != ZERO) {
            templ = alpha * conjunction(creal(y[j]), cimag(y[j]));
            temp2 = conjunction(creal(alpha * x[j]), cimag(alpha * x[j]));
            #pragma clang loop vectorize(assume_safety)
            for (int i = 0; i <= j - 1; i++) A[i][j] = A[i][j] + x[i] * templ + y[i] * temp2;
            A[j][j] = creal(A[j][j]) + creal(x[j] * templ + y[j] * temp2);
        } else A[j][j] = creal(A[j][j]);
    }
}
```



Support EPAC 1.0  
and EPAC 1.5

Support EPAC 2.0

# EPAC TIMELINE

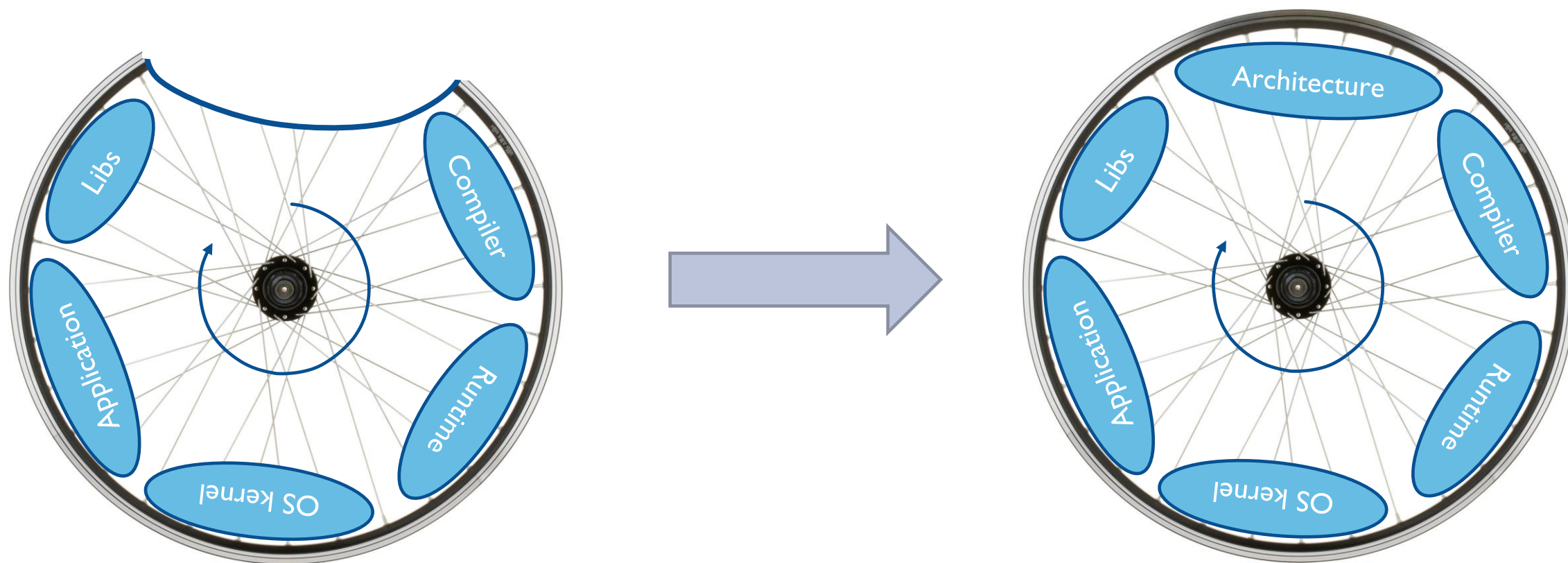






# ABOUT CO-DESIGN

# THE WHEEL OF CO-DESIGN

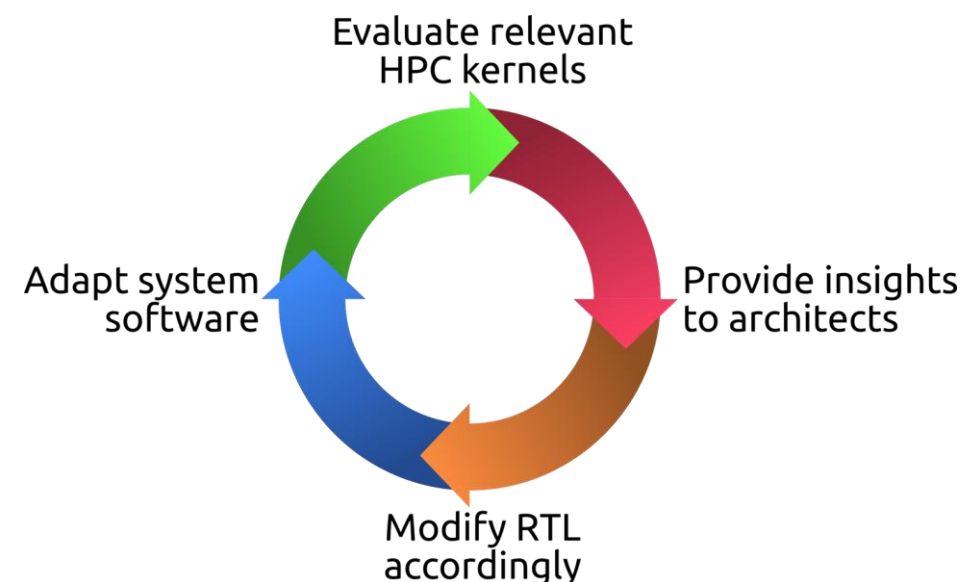
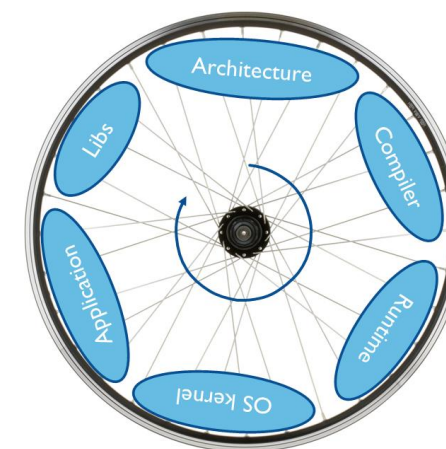


# CO-DESIGN WITH EPAC-VEC

- Test new architectures
  - while being developed
  - before the chip is ready
  
- Influence design decisions
  - Architecture
  - System software (e.g., compiler, libraries)
  - Applications (we are here today!)



SDV: Software Development Vehicles



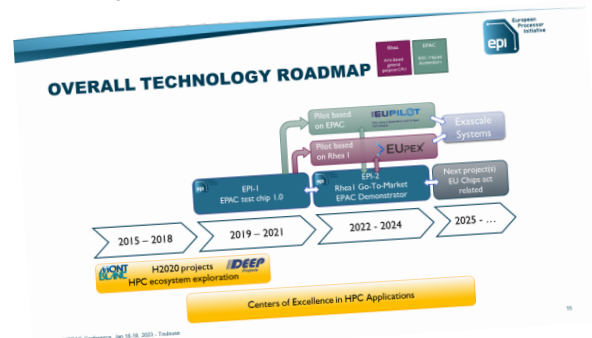
# WHO GAINS WHAT?

## Code developer

- Port the code to RISC-V
  - Recompile, analyse compiler output, understand dependencies
- Gain a vectorized code that
  - Leverage RISC-V vector extensions
  - Still run on your today's cluster (portability)
- Be ready for tomorrow's European systems

## EPI project

- Provide feedback to the compiler team
  - Improve the compiler
  - Teach how to write “vector friendly code”
- Provide feedback to the HW design team
- Give priorities to the porting of the HPC system software stack





# Software development vehicles (SDV)

# RISC-V PLATFORMS: COMMERCIAL AND FPGA-BASED



## Cluster of 10 HiFive Unmatched

- Operated as a standard HPC cluster (SLURM, modules, ...)



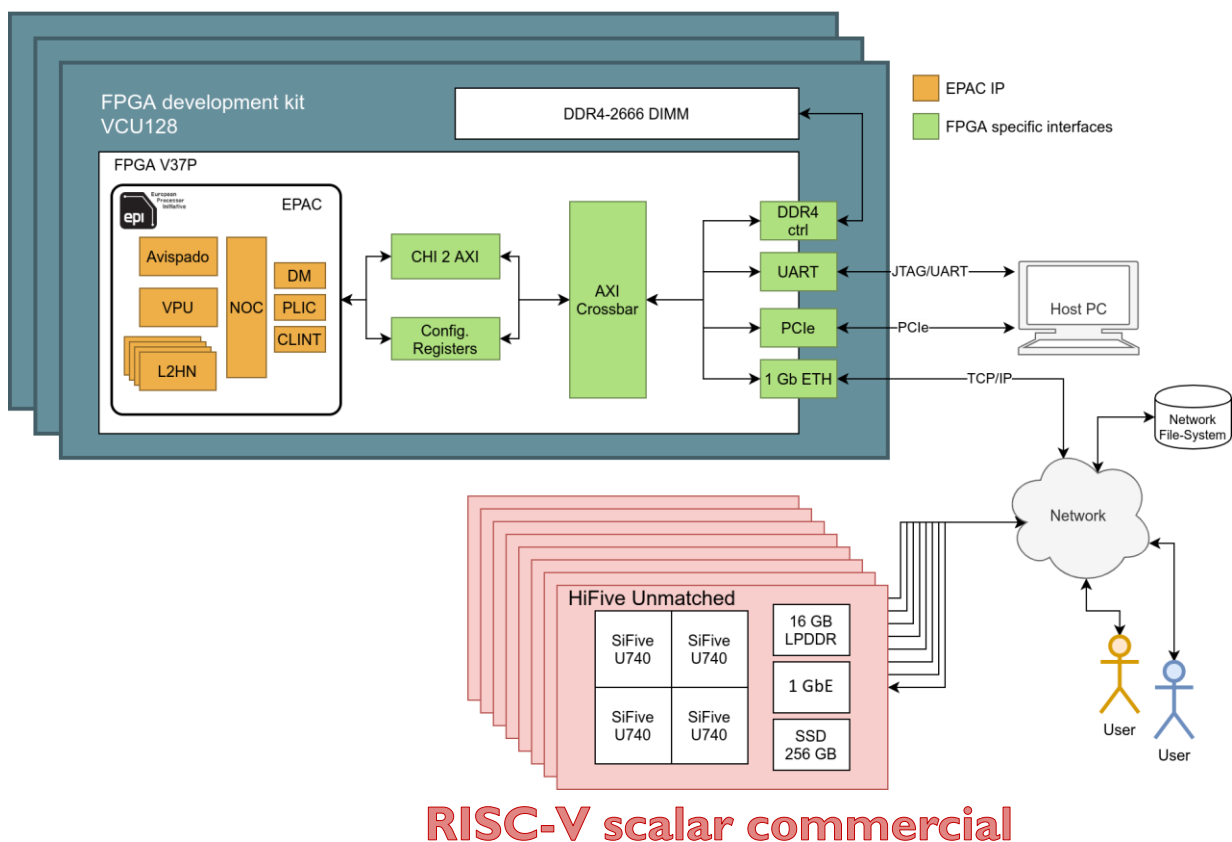
## Cluster of 4 host+FPGA (VCUI28)

- Operated as a standard HPC cluster once the FPGA is configured



# RISC-V PLATFORMS: COMMERCIAL AND FPGA-BASED

## Self hosted RISC-V vector node @ 50 MHz

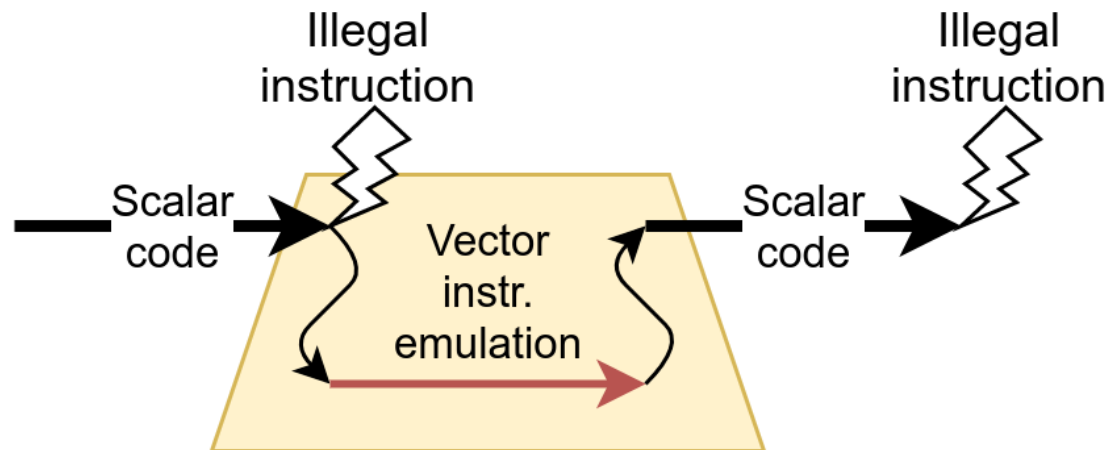


Scalar RISC-V commercial platforms coupled with EPAC development platforms

- Hardware/Software infrastructure for Continuous Integration and RTL check
- Playground to demonstrate a full HPC software stack
  - Linux, compiler, libraries, job scheduler, MPI
- Platform to test latest RTL with complex codes
  - Advanced performance analysis tools
  - Accurate timing available

# SOFTWARE EMULATOR: VEHAVE

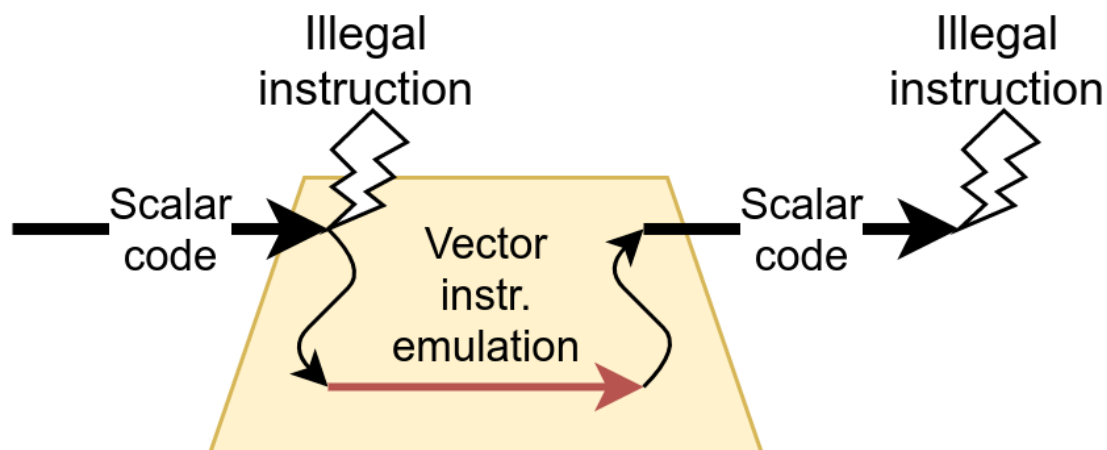
- Compile using the RISC-V Vector extension (RVV) Compiler
- Obtain a binary with vector instructions
- Run in a commercial RISC-V platform (scalar CPU)
- Obtain a trace with detailed information about the vectorization



Commercial RISC-V platform (scalar CPU)



# SOFTWARE EMULATOR: VEHAVE



## PROS:

- Useful to understand the potential vectorization of the code
- Easy to use and accessible with no need of hardware infrastructure
- It supports RVV-0.7 and RVV-1.0
- Output compatible with Paraver

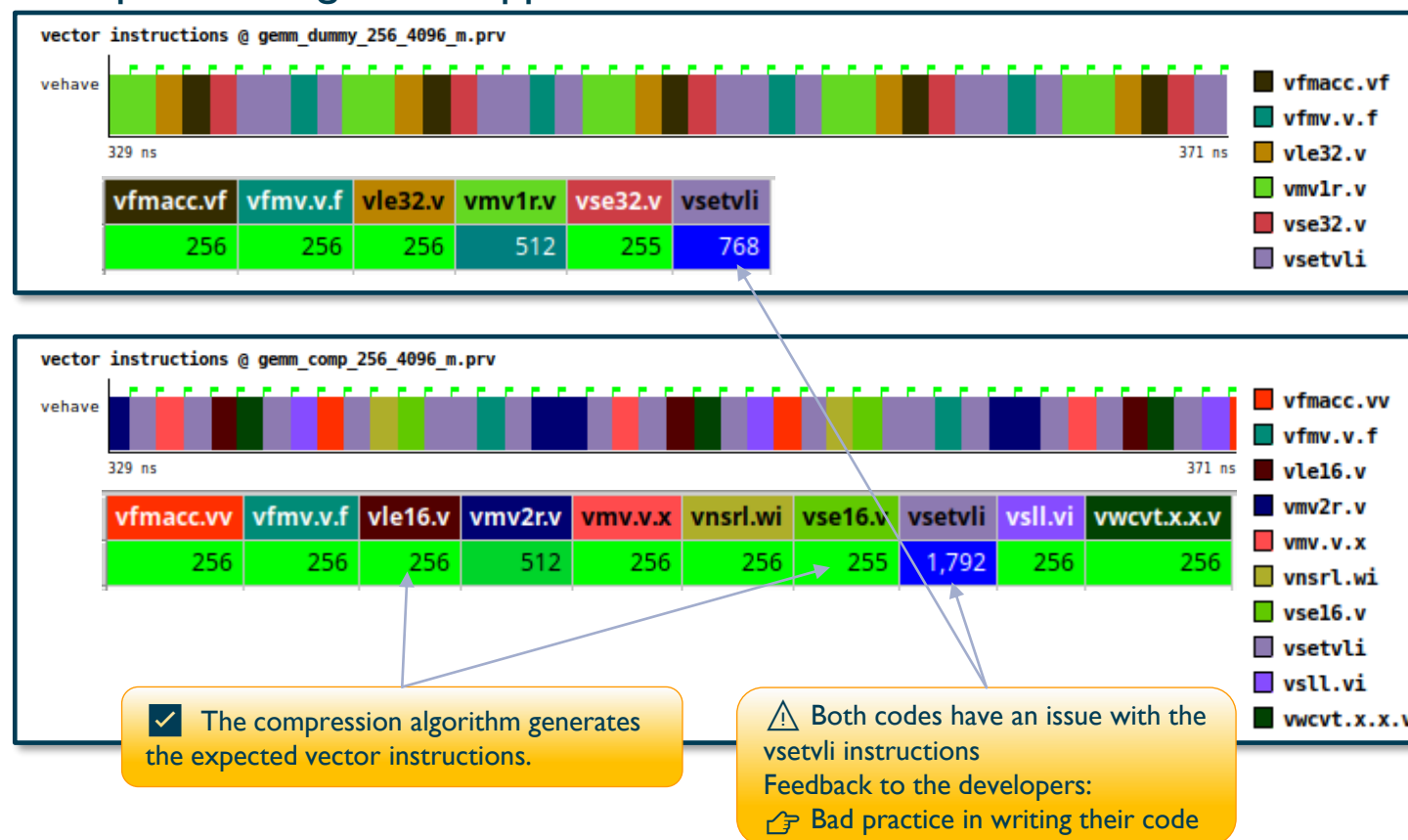
## CONS:

- Slow
- No information about performance (no timing)

# 1<sup>ST</sup> STEP: STUDY WITH VEHAVE

- Compile an application
  - Relying on autovectorization, with intrinsics, pragmas or assembly
  - Study the output of the compiler
- Run with emulation enabled
  - Collect execution traces
- Visualize and study traces
- How well the code is vectorized?
  - Feedback to the compiler team
  - Guidelines to the application developer

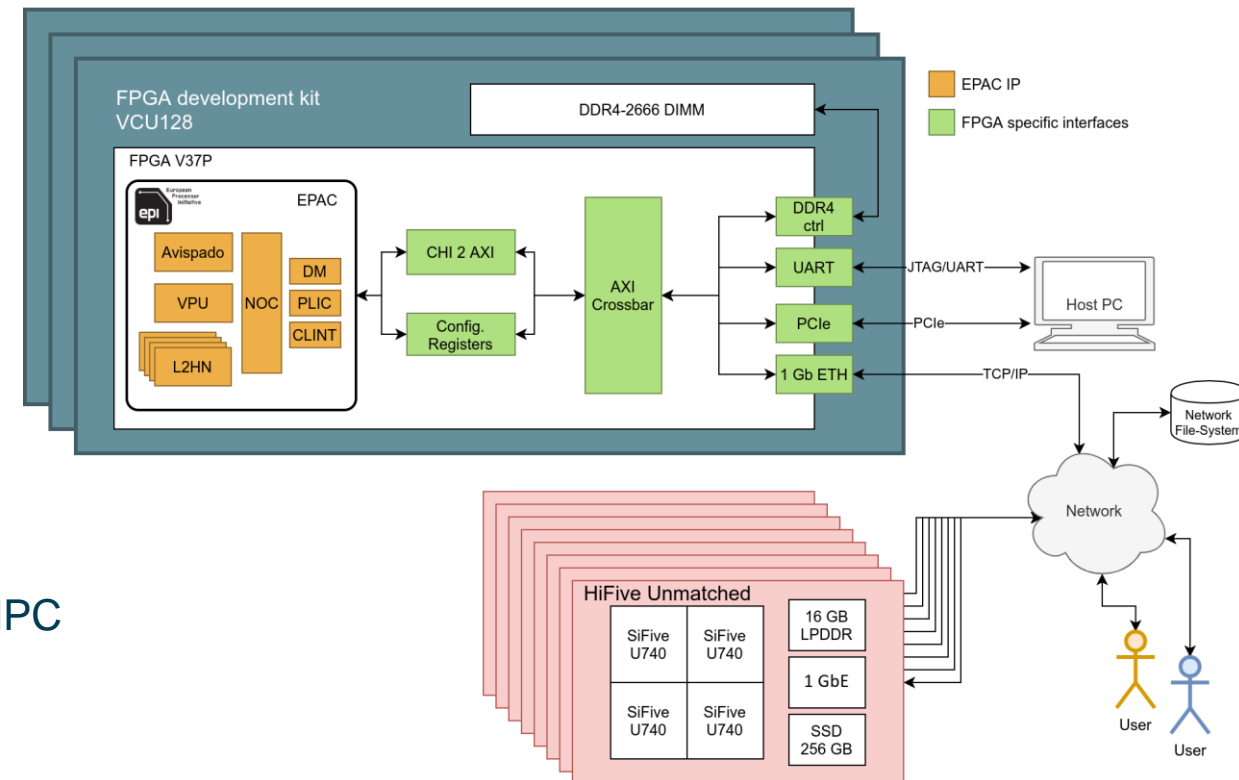
## Compression algorithm applied to a GEMM



# 2<sup>ND</sup> STEP: RUN ON FPGA-BASED SOFTWARE DEVELOPMENT VEHICLES

- Same RTL of EPAC mapped into FPGA
  - One tile (i.e., single core)
  - Running at 50 MHz
- Full HPC software stack and execution environment
  - Binary compatibility
  - Shared storage (NFS)
  - Multi-user / Multi-node (via MPI)
  - Standard debug and performance analysis tools for HPC

## Self hosted RISC-V vector node @ 50 MHz

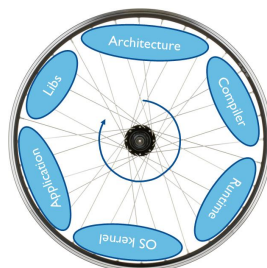


## RISC-V scalar commercial

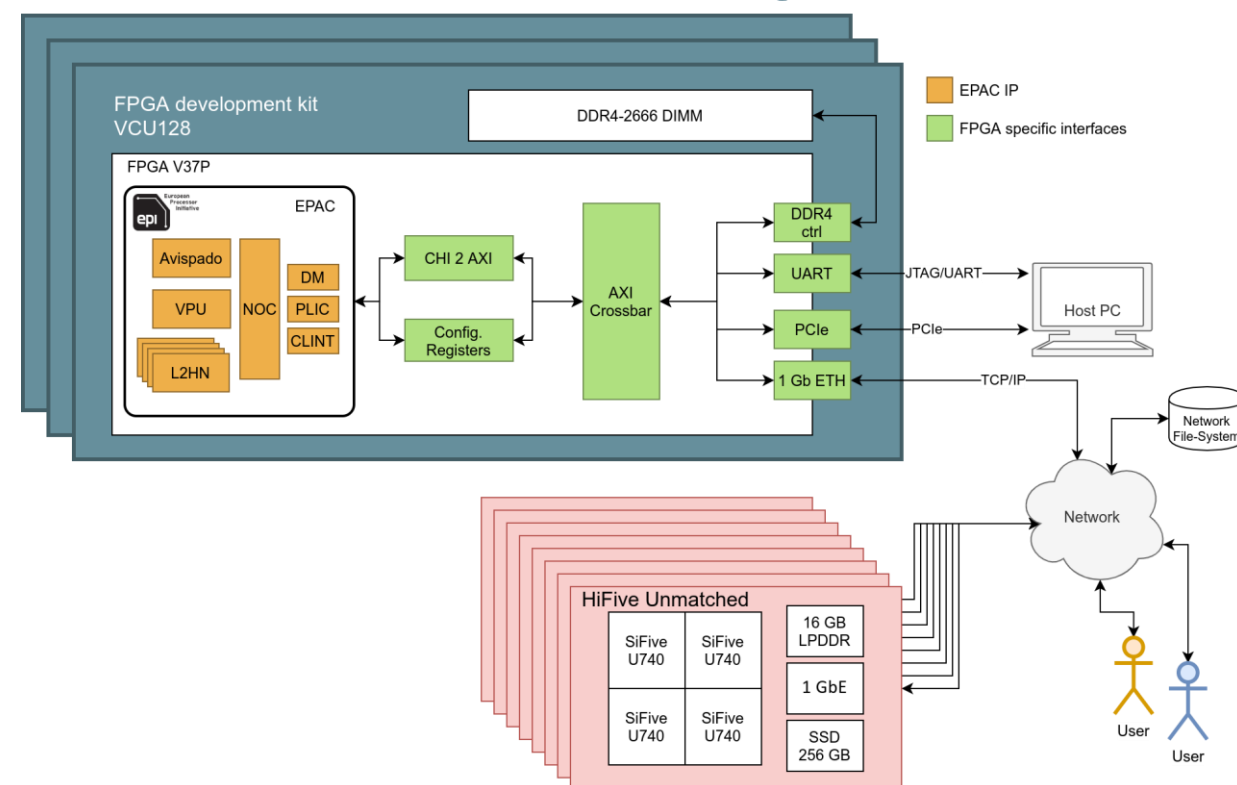
# 2<sup>ND</sup> STEP: RUN ON FPGA-BASED SOFTWARE DEVELOPMENT VEHICLES

Added values:

- Test the platform **before hardware is available**
- **Test** the effect of hardware design **on complex codes** before freezing the architectural specs
- Develop **system-software, libraries** and applications on top of a full HPC software stack
- Allow co-design between hardware and **all software layers**
  - In the end, SDV enables RISC-V in HPC!



## Self hosted RISC-V vector node @ 50 MHz



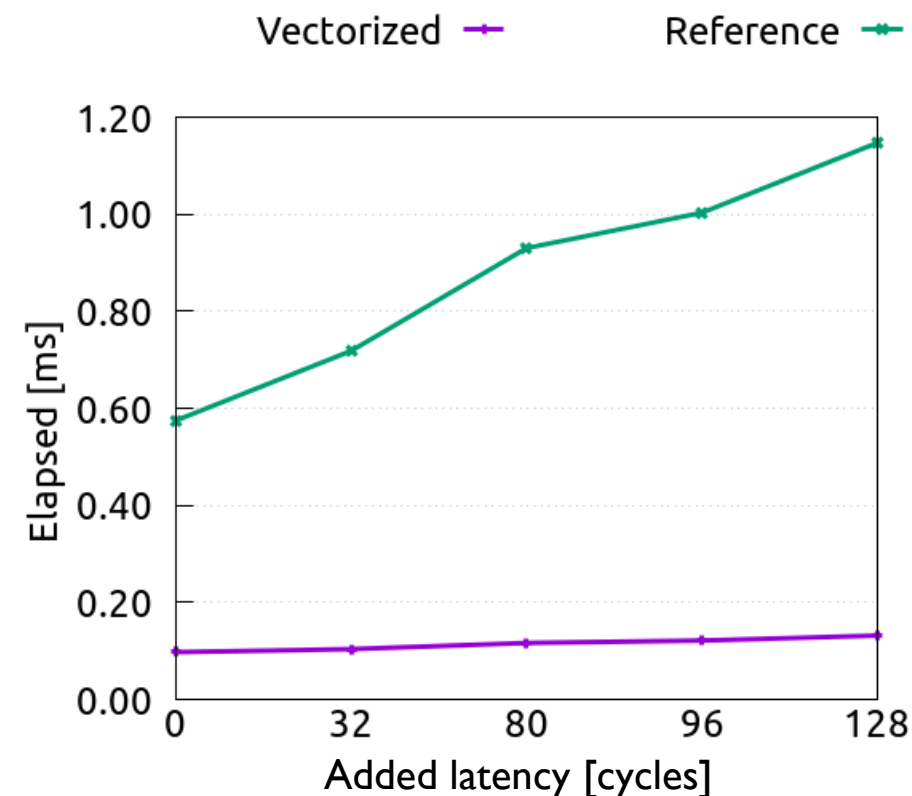
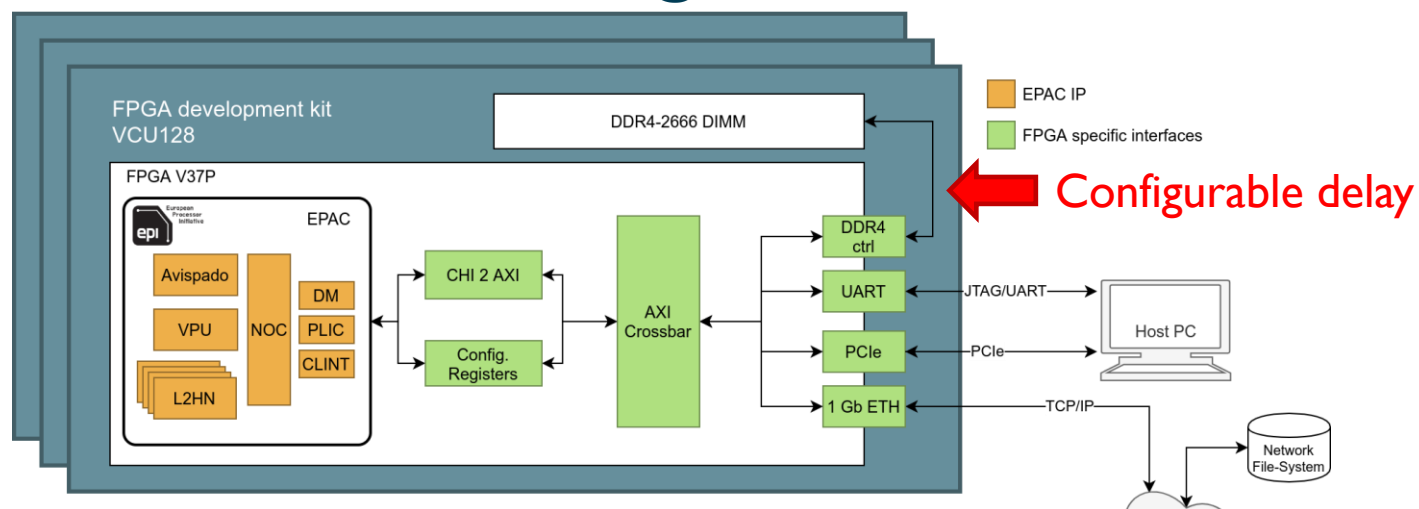
## RISC-V scalar commercial

# EXAMPLE OF STUDY ON FPGA-SDV #1

## TEST OF HARDWARE CONFIGURATIONS

- How sensitive are vectorized/non vectorized codes to the latency accessing the memory?

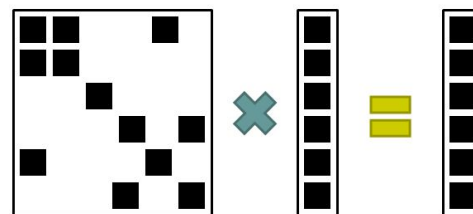
Self hosted RISC-V vector node @ 50 MHz



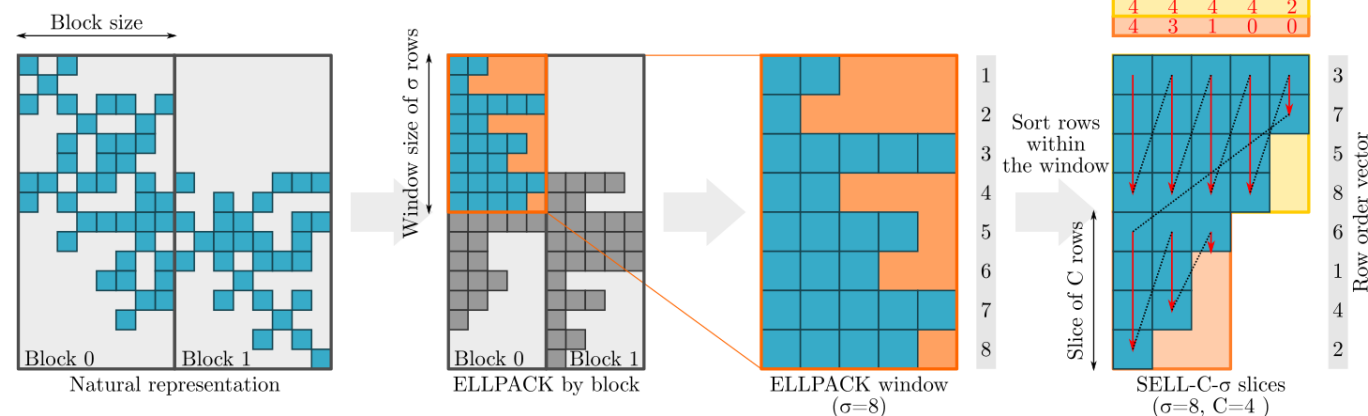
# EXAMPLE OF CO-DESIGN STUDY #2

## DEVELOPMENT OF OPTIMIZED LINEAR ALGEBRA LIBRARIES

- Sparse Matrix-Vector multiplication



- Naïve implementation struggles taking advantage of the vector unit
- Implementation of “vector friendly” SpMV algorithm: Sell-C-sigma



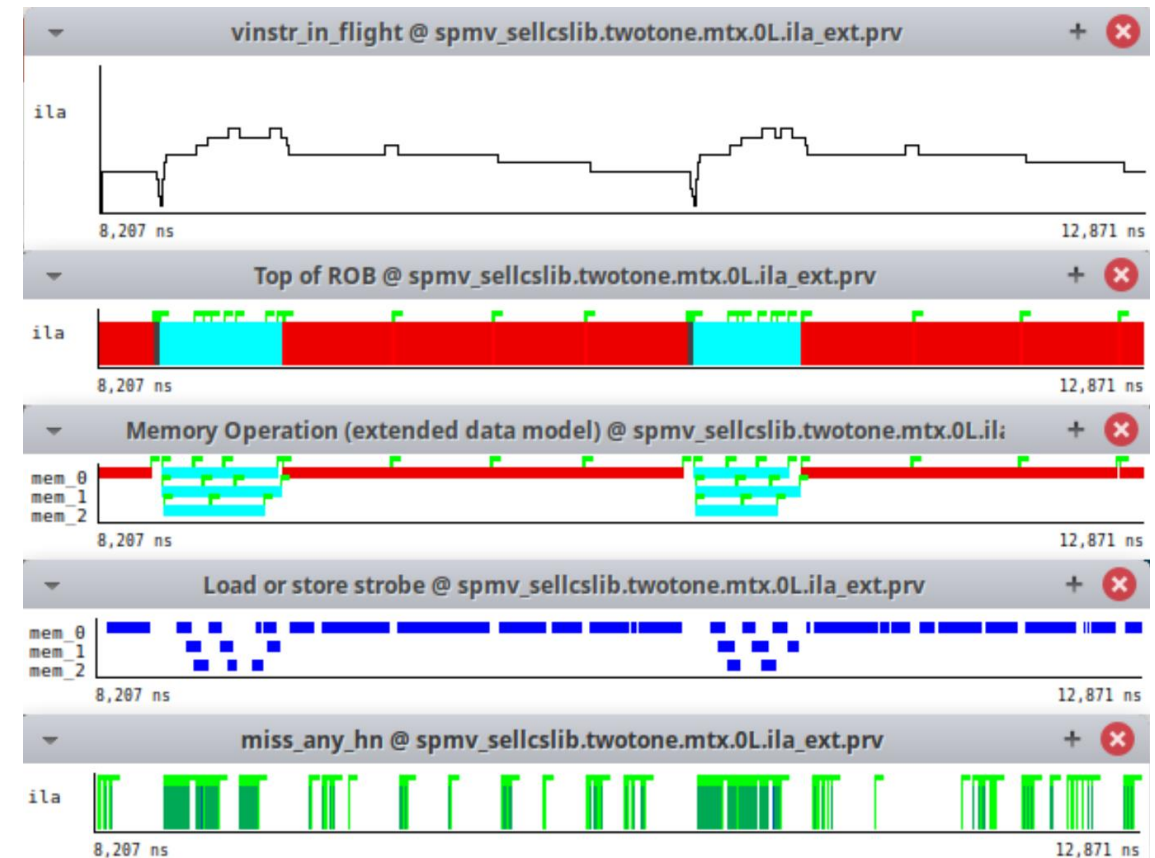
✎ Gómez, Constantino, Filippo Mantovani, Erich Focht, and Marc Casas. "Efficiently running SpMV on long vector architectures." In Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, pp. 292-303. 2021.

✎ Vizcaino, Pablo, Filippo Mantovani, Roger Ferrer, and Jesus Labarta. "Acceleration with long vector architectures: Implementation and evaluation of the FFT kernel on NEC SX-Aurora and RISC-V vector extension." Concurrency and Computation: Practice and Experience (2022): e7424.

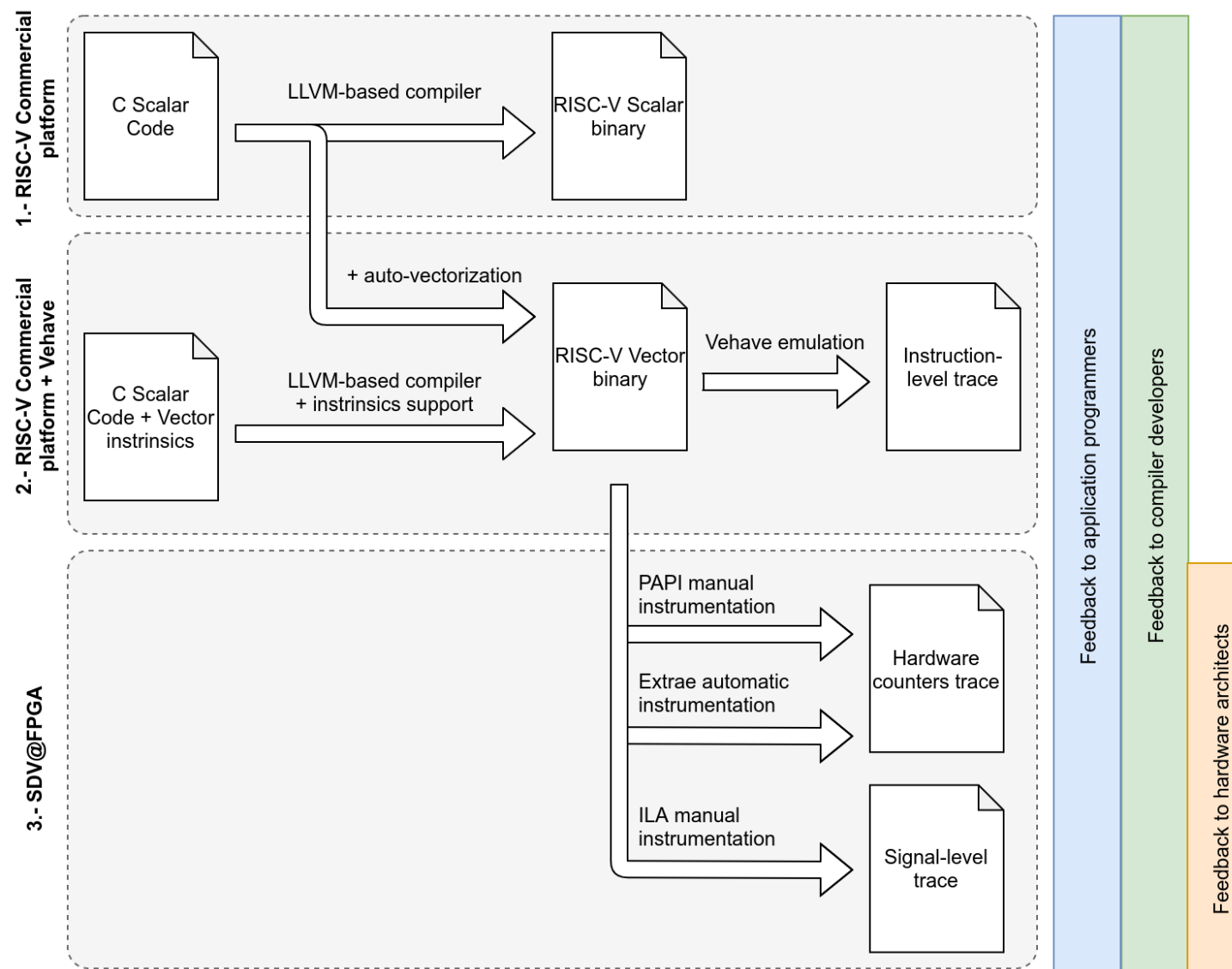
# 3<sup>RD</sup> STEP: SIGNAL ANALYSIS ON FPGA-SDV

## INTEGRATED LOGIC ANALYSER

- Fine grained analysis (at level of instructions) is possible
- Graphical representation of timelines
- In depth study can help highlighting:
  1. Low usage of the vector unit
    - Feedback to the **code developer**
  2. Suboptimal saturation or resources (FU, mem)
    - Feedback to the **RTL implementation team**
  3. Suboptimal overlap of instructions
    - Feedback to the **compiler team** (improve scheduling)



# CO-DESIGN WITH SDV





# WHAT'S NEXT? (TODAY)

- Tutorial where we will learn
  - SDV platforms and tools
  - How to login, compile and run
  - How to emulate vector binaries
  - How to submit jobs to the SDV platforms
- We will provide
  - A temporary account
  - Full access to SDV
  - A toy code for playing with SDV
  - Our support to play with your codes

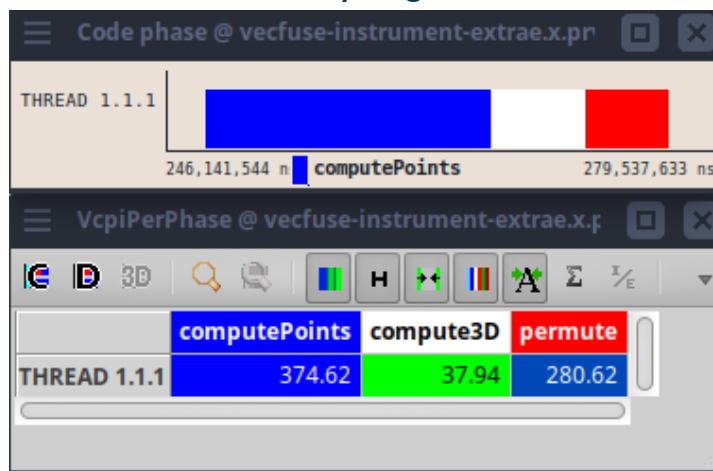


## Vectorizing

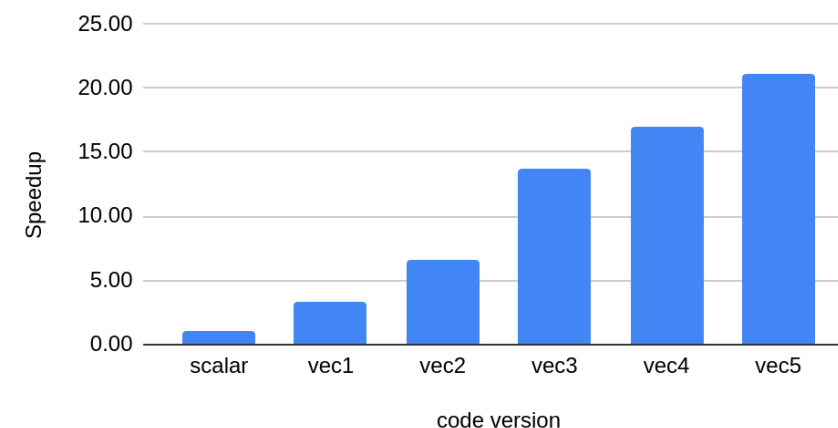
```
#pragma clang loop vectorize(enable)
for(int jk=0; jk<M*L; ++jk){
    double a = A[jk];
    double b = B[jk];
    double c = C[jk];
    double x1 = a*px + b*py + c*pz;
    double x2 = a*py + b*pz + c*px;
    double x3 = a*pz + b*px + c*py;
    double res = x1*x2*x3;
    C[jk] = res;
}
```

Institute for Advanced Simulation Seminar, Apr 25, 2023 - Jülich

## Analyzing

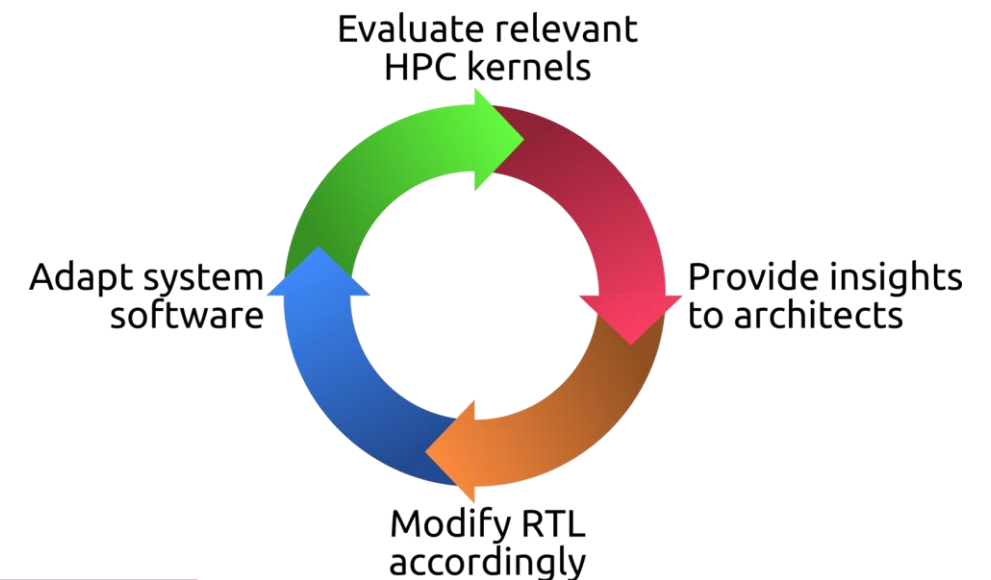


## Improving



# SUMMARY

- EPI develops the first RISC-V based accelerator targeting HPC leveraging the V-extension
  - RTL design of a Vector Unit
  - LLVM compiler support for the V-extension
- While RTL is becoming actual hardware, EPI develops tools for boosting the co-design cycle: Software Development Vehicles
  - Emulator of V-instructions (Vehave)
  - Commercial RISC-V-based clusters
  - FPGA-SDV: self-hosted RISC-V core coupled with a VPU on FPGA
  - Performance analysis tools
- We can leverage SDVs to:
  - Influence hardware design
  - Improve compiler autovectorization and system-software support
  - Prepare and tune codes and libraries for long-vector architectures



Interested in testing your code on EPAC?  
✉ Filippo Mantovani [filippo.mantovani@bsc.es](mailto:filippo.mantovani@bsc.es)

# THE “GOOD” LIES...

- Arm works on battery, so it's energy efficient
  - This is 10 years old, but I still hear it...
- RISC-V is a free CPU
  - No! RISC-V is an open ISA. CPUs are ISA implementations and may or may not be free!
- RISC-V will do to the hardware what Linux did to the OS
  - No! For building hardware you need \$\$ (a lot) for CAD tools, tapeouts, etc
- A closed ISA is a safer business decision for a company
  - No! Within companies, projects refocus... And companies may disappear
- RISC-V can be customized, everybody can put his own preferred instructions in the ISA
  - No! Well, in principle yes, but this is not the point of the ISA and it does not fragment the software

Good read  <https://www.eetimes.com/examining-the-top-five-fallacies-about-risc-v/>

Good movie  <https://www.youtube.com/watch?v=iFlcJFcOJKk>

# ACKNOWLEDGMENT AND CONTACTS




- Stream 3 leader and responsible for SDV:
  - Filippo Mantovani [filippo.mantovani@bsc.es](mailto:filippo.mantovani@bsc.es)
- EPI General Manager:
  - Etienne Walter [etienne.walter@atos.net](mailto:etienne.walter@atos.net)



[@EuProcessor](https://twitter.com/EuProcessor)



[European Processor Initiative](https://www.linkedin.com/company/european-processor-initiative/)

 F. Mantovani, P. Vizcaino, F. Banchelli et al., "Software Development Vehicles to enable extended and early co-design: a RISC-V and HPC case of study", First International workshop on RISC-V for HPC, ISC23, 25 May 2023.

# EPI FUNDING



This project has received funding from the European High Performance Computing Joint Undertaking (JU) under Framework Partnership Agreement No 800928 and Specific Grant Agreement No 101036168 EPI-SGA2. The JU receives support from the European Union's Horizon 2020 research and innovation programme and from Croatia, France, Germany, Greece, Italy, Netherlands, Portugal, Spain, Sweden, and Switzerland.

SPONSORED BY THE



Financé  
par



