



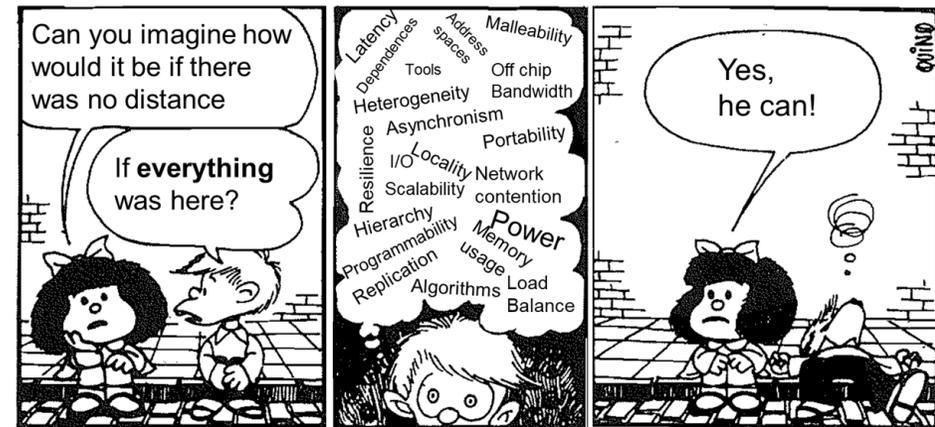
# THE ACCELERATOR TILE OF THE EUROPEAN PROCESSOR INITIATIVE (EPAC)

JESUS LABARTA (JESUS.LABARTA@BSC.ES)

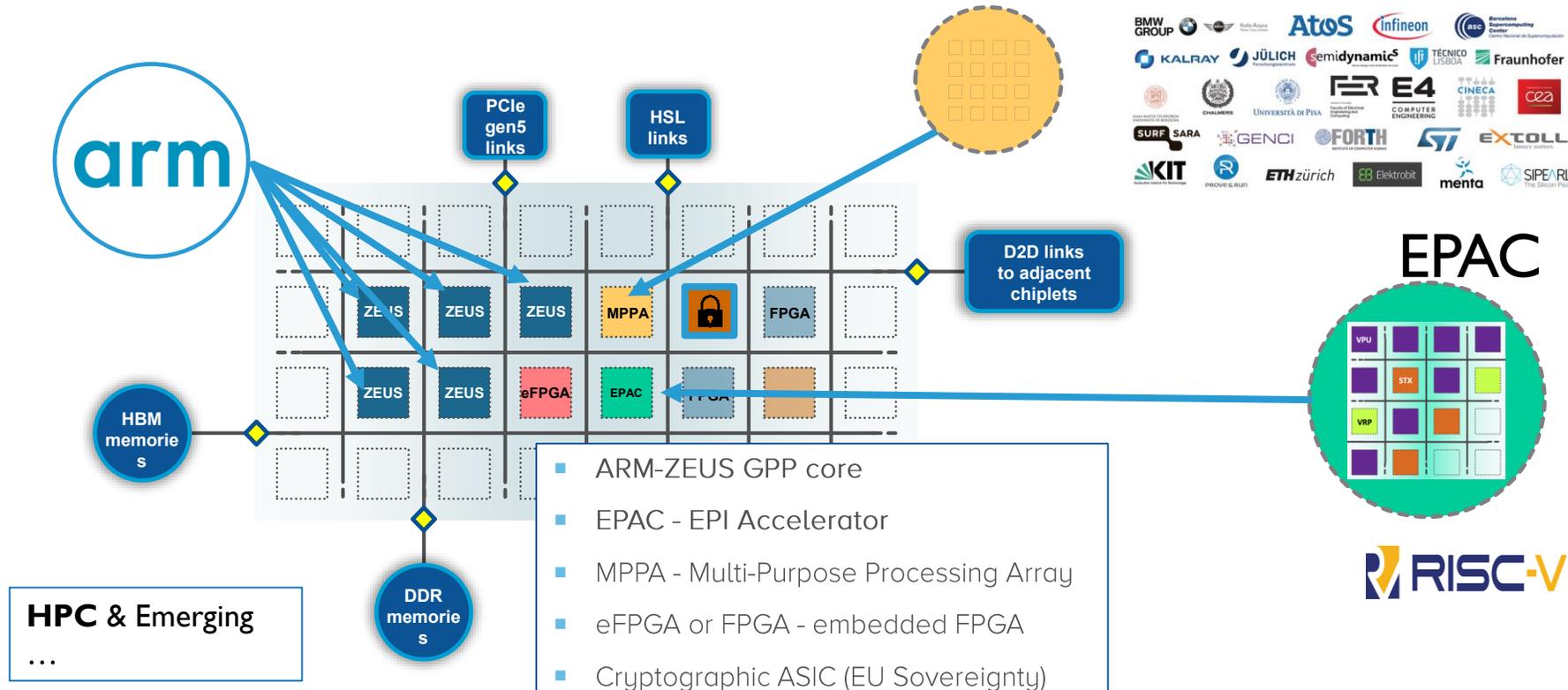


# EPI OBJECTIVES

- Components (low power microprocessor technologies) ...
  - ARM based SoC
  - RISC-V based accelerator
  
- ... to be combined to target
  - HPC
  - HPDA
  - Emerging
    - Automotive
    - ...

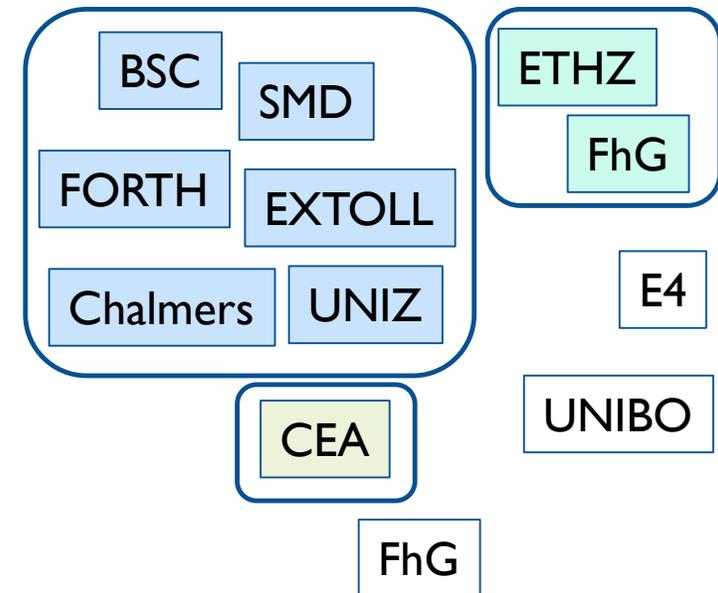


# OVERALL ARCHITECTURE



# VISIONS AND COLLABORATIONS

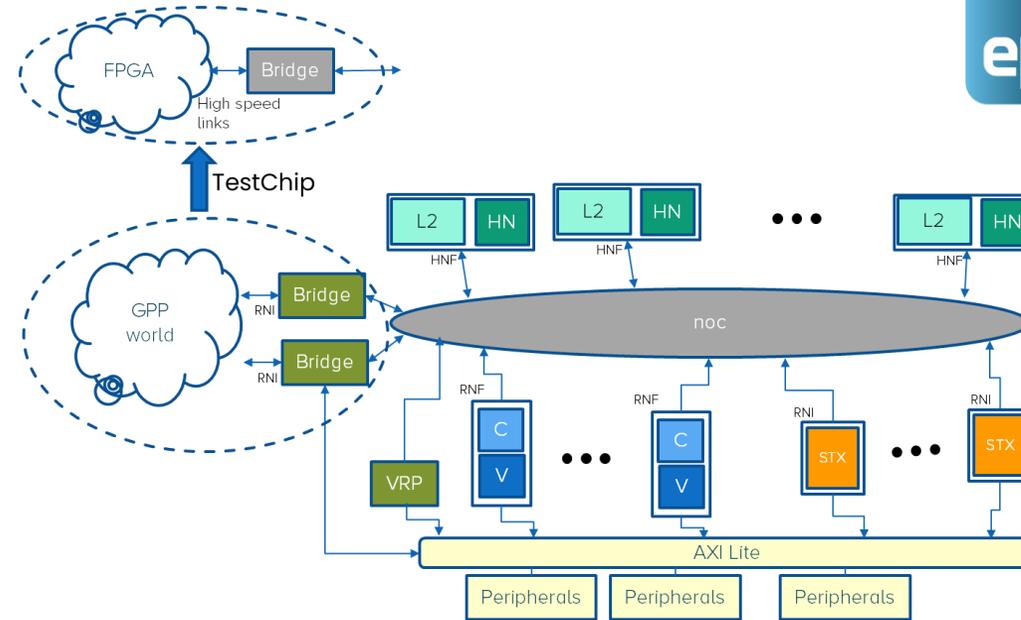
- STX:
  - Specific Accelerator devices
    - AI
    - Stencil
- RVV
  - ISA is important, RISC-V Vector
  - “Accelerator”
    - Easier entry, focus
    - → Standard self hosted, general purpose vector SMP
- VRP:
  - Extended precision arithmetic



# EPAC ARCHITECTURE

## RVV

- RV64GCV (→ 8x)
  - 2 way in order core
  - Decoupled VPU
    - 8 lanes
    - Long vectors (256 DP elements)
  - L1 - MESI coherency
- CHI interface NoC
  - 1 line / cycle
- L\$2: 256KB/module
  - Allocation control mechanisms
- No in tile L\$3



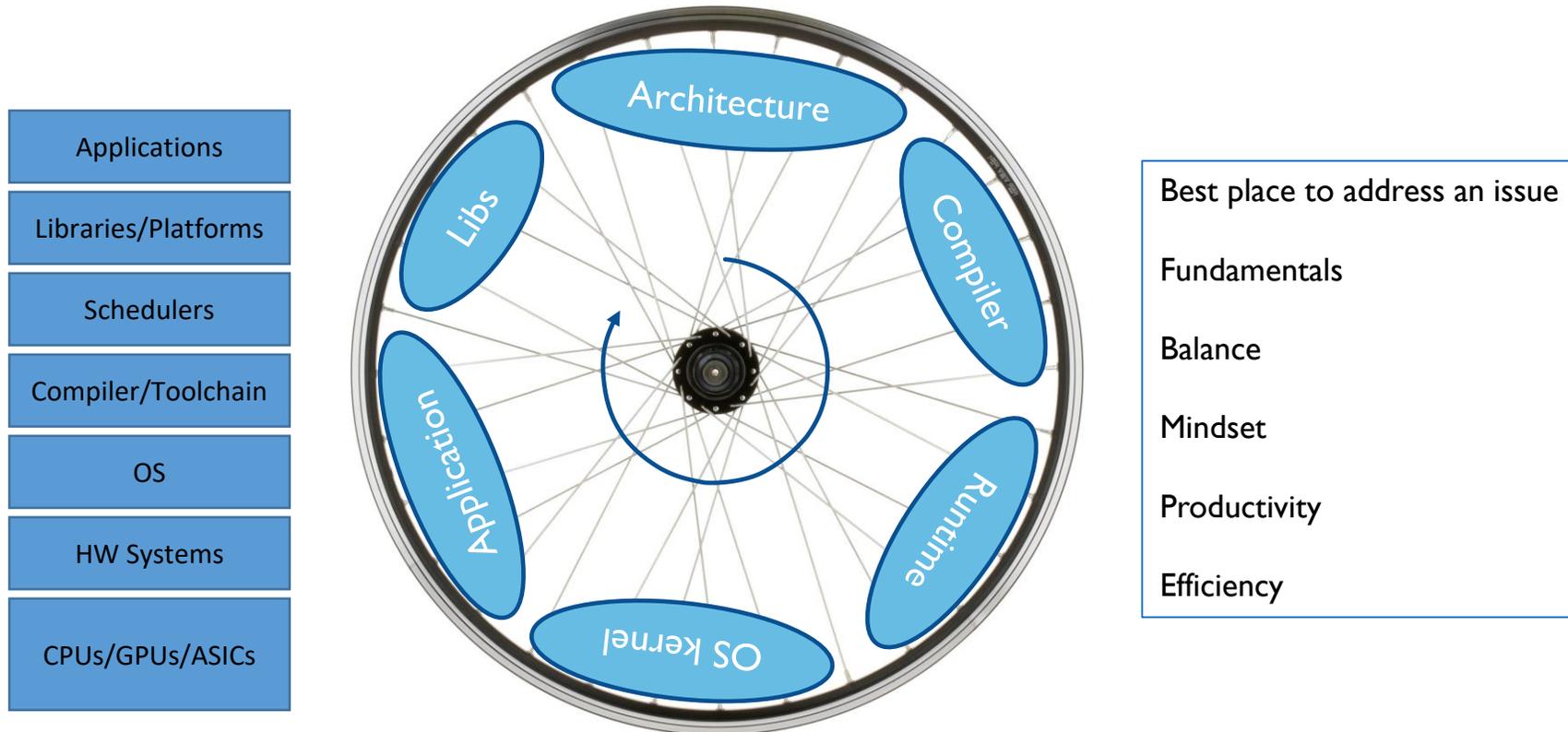
## STX

- DL and stencil specific accelerators
- Extensions to planned NTX
  - Programmable address generators →
  - lightweight RISC-V core + fat FPU + (Streaming Semantics & FREP)

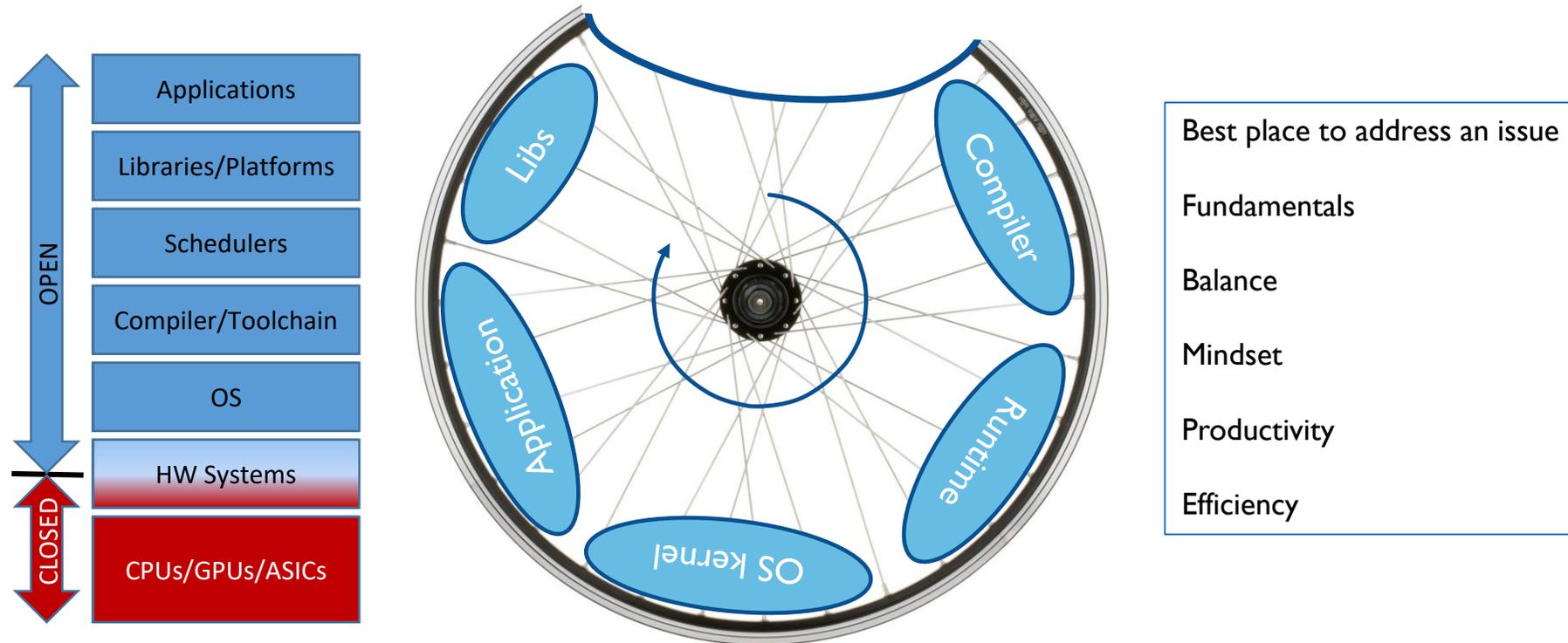
## VRP

- Variable precision processors

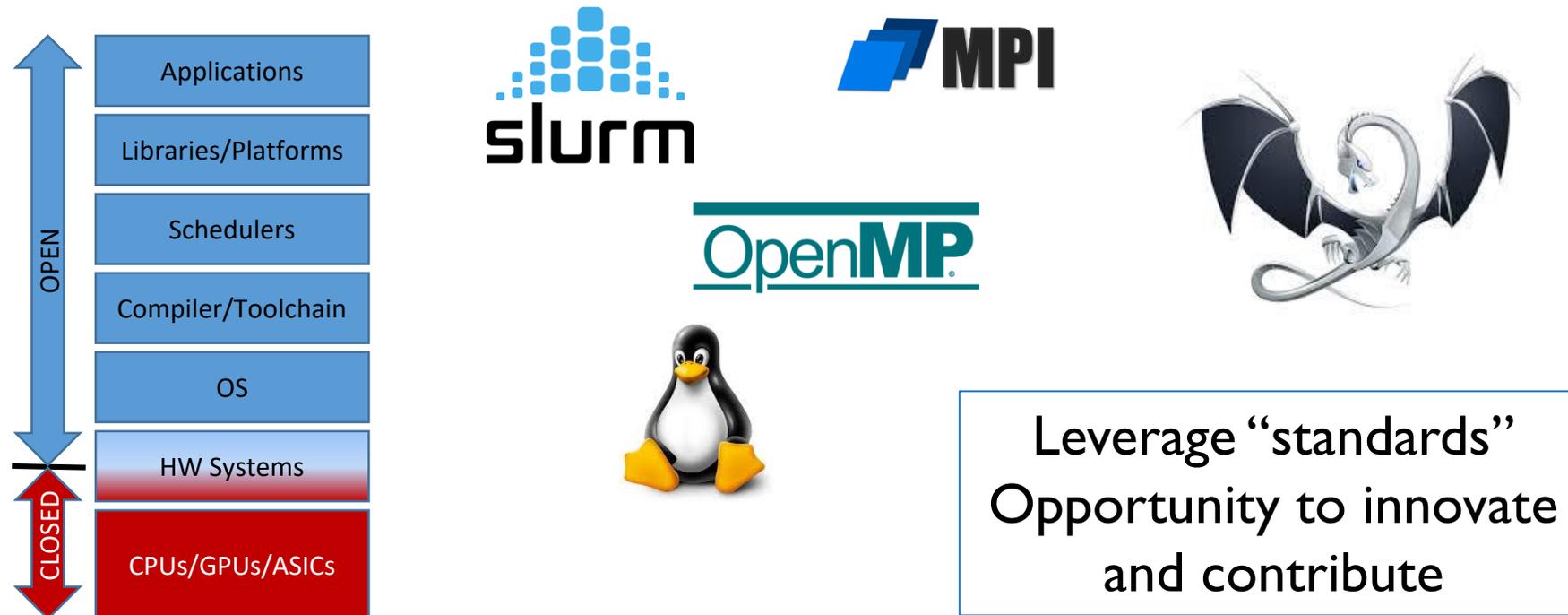
# HOLISTIC CO-DESIGN



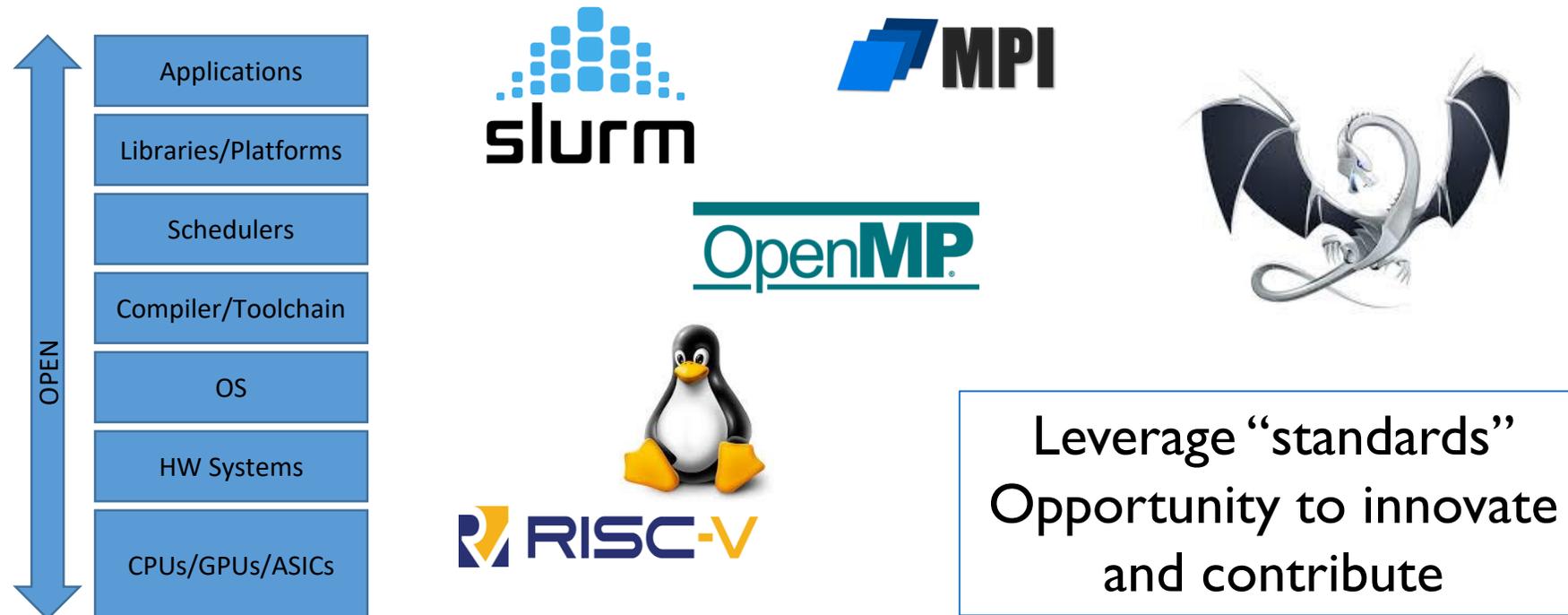
# HOLISTIC CO-DESIGN



# LEVERAGE INTERFACES AND IMPLEMENTATIONS



# LEVERAGE INTERFACES AND IMPLEMENTATIONS



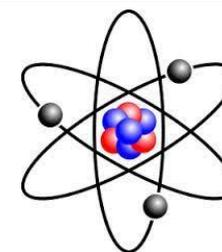
# HOLISTIC CO-DESIGN



“As above, so below”  
Similar concepts/mechanisms at all levels

“Steered by a vision”  
“Steered by detailed insight”

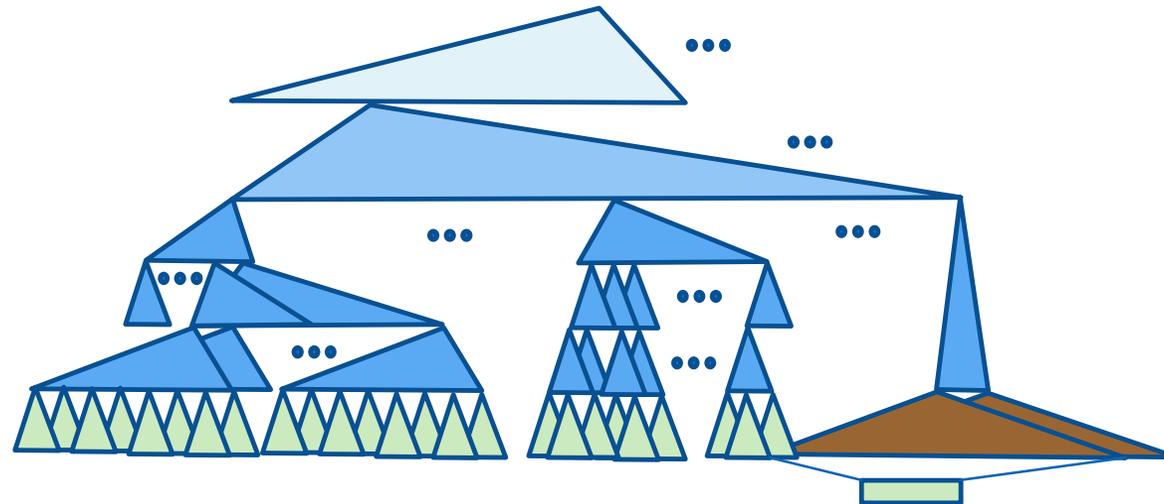
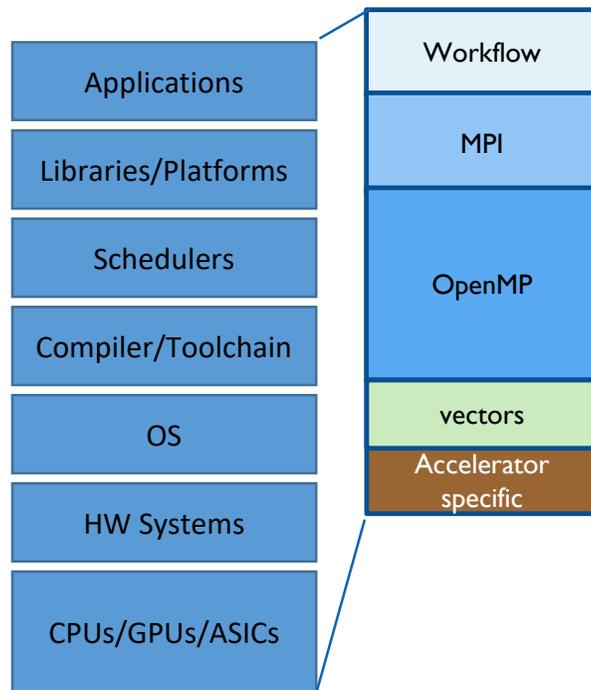
- Principles**
- Balanced hierarchy
  - Latency → throughput: asynchrony & overlap
  - Malleability and coordinated scheduling
  - Homogenize heterogeneity



# BALANCED HIERARCHY

$$10^6 = 1 \times 10^6 = 10 \times 10^5 = 10^2 \times 10^2 \times 10^2$$

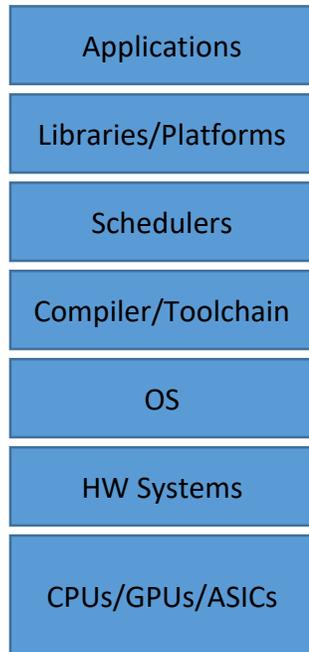
Expression & exploitation of Parallelism



Long vectors 

256 elements. 8 lanes per core  
“Limited” number of “general purpose” control flows within tile

# LATENCY → THROUGHPUT: ASYNCHRONY AND OVERLAP



## Interoperability MPI + OpenMP

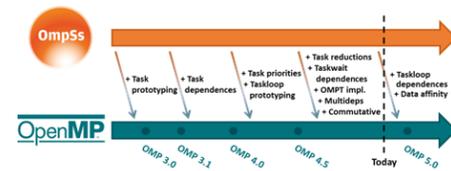
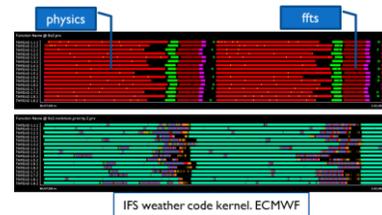
- Taskify MPI calls

## Task based models

- Single mechanism
  - Concurrency
  - Locality & data management

## Long vectors

- decouple Front end – back end
- Convey access pattern semantics to the architecture. Potential to optimize memory throughput



## Task based computational workflows



```

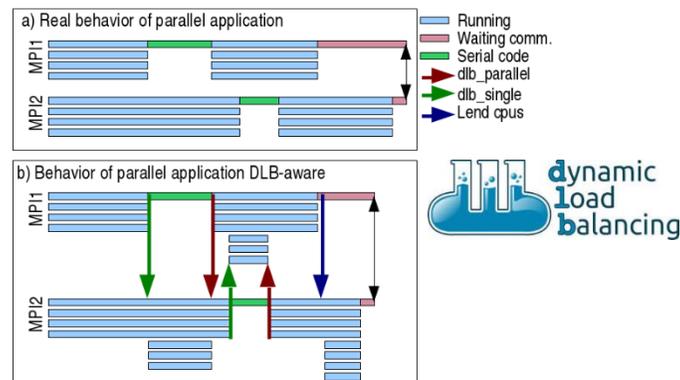
void Cholesky(int NT, float *A[NT][NT]) {
  for (int k=0; k<NT; k++) {
    #pragma omp task inout ([TS][TS] A[k][k])
    spotrf (A[k][k], TS) ;
    for (int i=k+1; i<NT; i++) {
      #pragma omp task in([TS][TS] A[k][k]) inout ([TS])
      stram (A[k][k], A[k][i], TS);
    }
    for (int i=k+1; i<NT; i++) {
      for (j=k+1; j<= i; j++) {
        #pragma omp task in([TS][TS] A[k][i]), [TS][TS] ()
        inout ([TS][TS] (**A[j][i]))
        sgmm( A[k][i], A[k][j], A[j][i], TS);
      }
      #pragma omp task in ([TS][TS] A[k][i]) inout([TS])
      sayrk (A[k][i], A[i][i], TS);
    }
  }
}
    
```



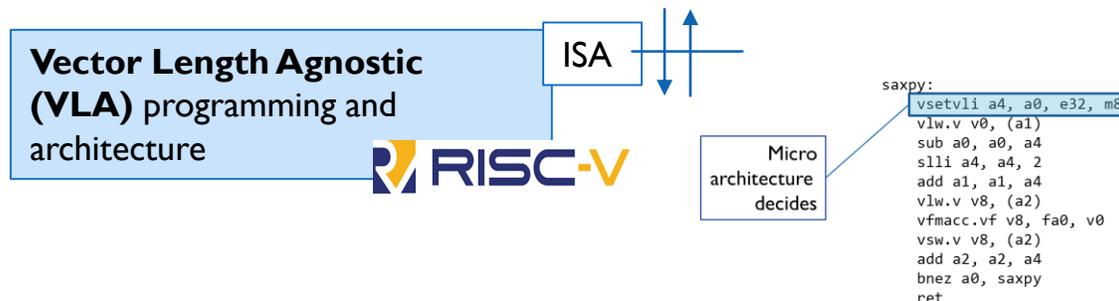
# MALLEABILITY & COORDINATED SCHEDULING



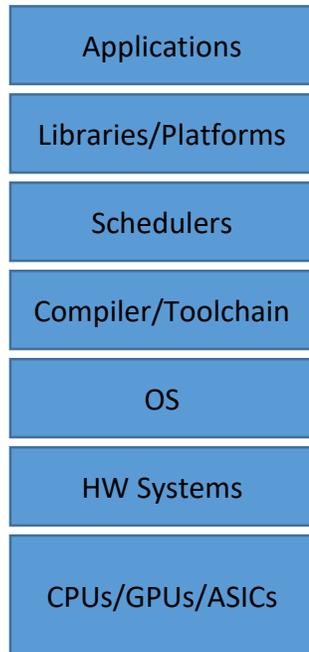
Coordination (policies)  
Composability, interoperability  
(semantic impedance matching)



A wish:  
Handoff scheduling



# HOMOGENIZING HETEROGENEITY



```

void axpy_omp_nest (double a, double *dx, double *dy, int n) {
    int i, chunk;
    #pragma omp taskloop
    for (i=0; i<n; i+=TS) {
        chunk= n>i+TS? TS : n-i;
        #pragma omp target map(to:dx[i:i+chunk], tofrom:dy[i:i+chunk])
        axpy_omp (a, &dx[i], &dy[i], chunk);
    }
}

void axpy_omp (double a, double *dx, double *dy, int n) {
    int I, chunk;
    #pragma omp taskloop
    for (i=0; i<n; i+=TS) {
        chunk= n>i+TS? TS : n-i;
        axpy_SIMD (a, &dx[i], &dy[i], chunk);
    }
}

void axpy_SIMD (double a, double *dx, double *dy, int n) {
    int i;
    #pragma omp simd
    for (i=0; i<n; i++) dy[i] += a*dx[i];
}
    
```

## VLA helps homogenize Heterogeneous Performance

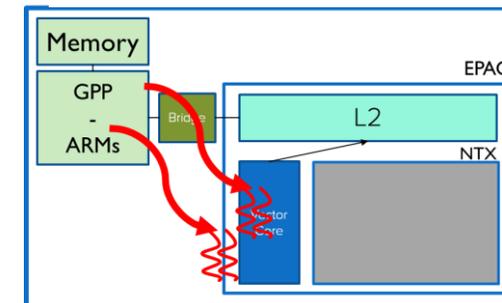
- Regular ISA
- ~ Big – Little cores, ....



## Nested tasked/workshared



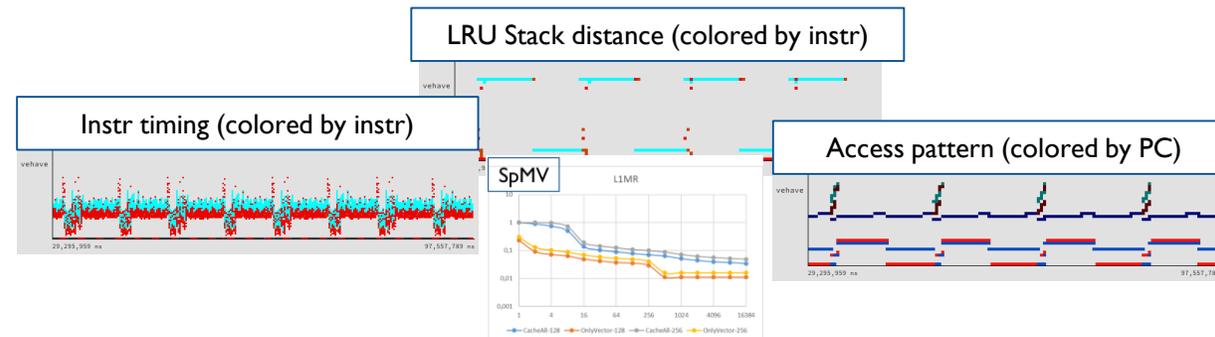
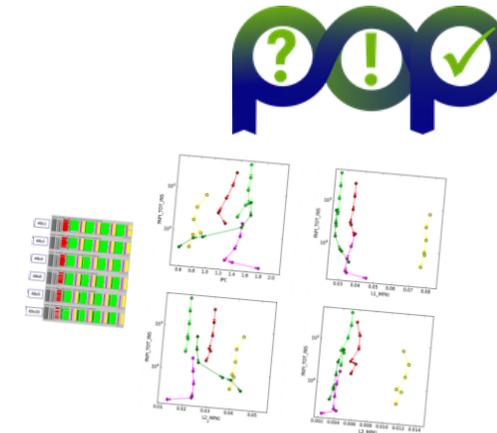
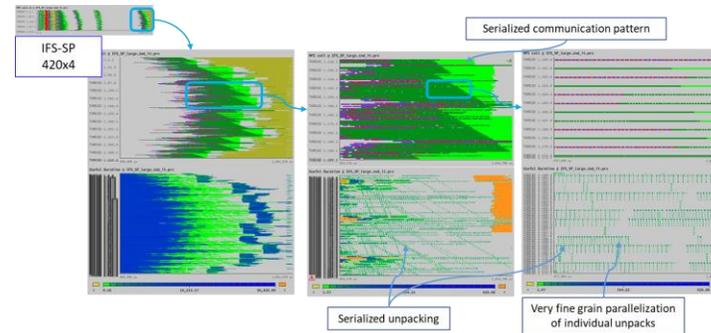
- Offload regular OpenMP



- HW support: IO coherence

# DETAILED ANALYSIS AND INSIGHT ON BEHAVIOR

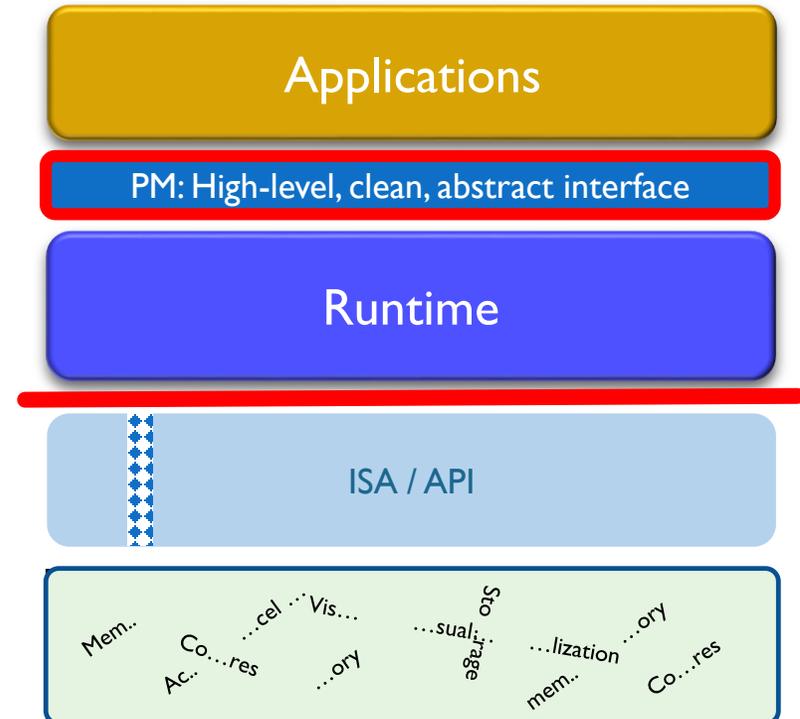
- Applications
- Libraries/Platforms
- Schedulers
- Compiler/Toolchain
- OS
- HW Systems
- CPUs/GPUs/ASICs



Try to avoid flying blind in the midst

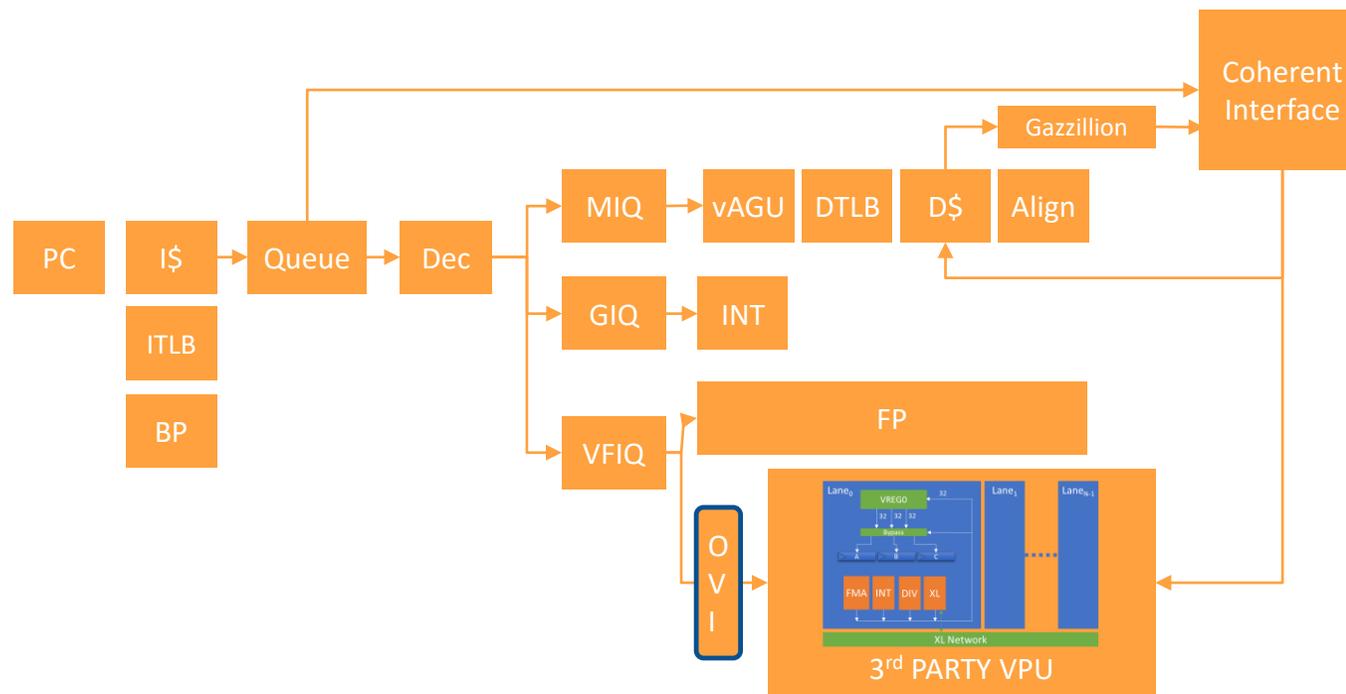
# LONG VECTORS

- Raise ISA semantic level
  - Vector instructions == tasks
  - “less words, more work”
  - The importance of ISA
  
- Parallelism
  - Decouple Front end – back end
    - Less pressure, throughput orientation
  - OoO execution
  
- Osmotic membrane
  - Convey access pattern semantics to the architecture.
  - Potential to optimize memory throughput.



# AVISPADO 220 WITH VPU

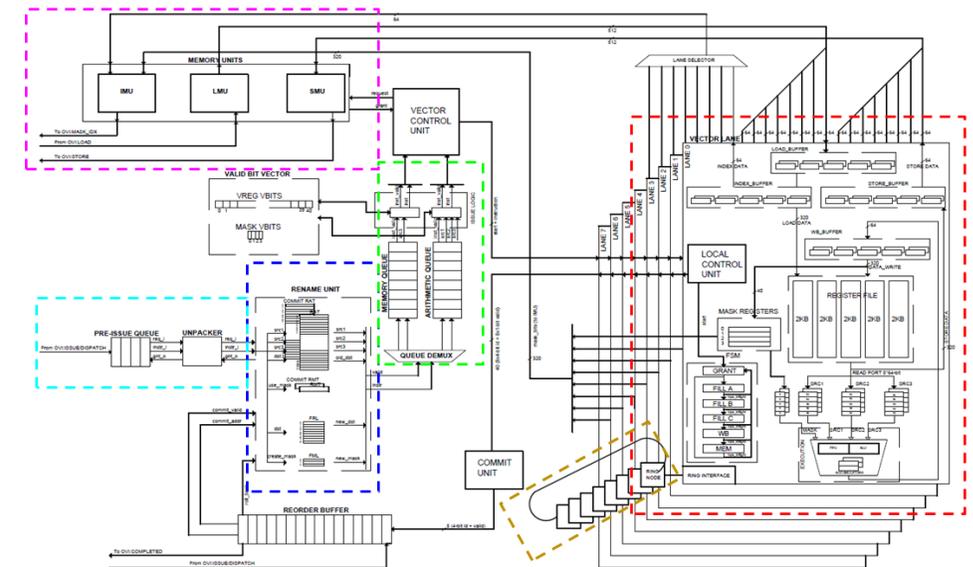
RISCV64GCV



Courtesy R. Espasa

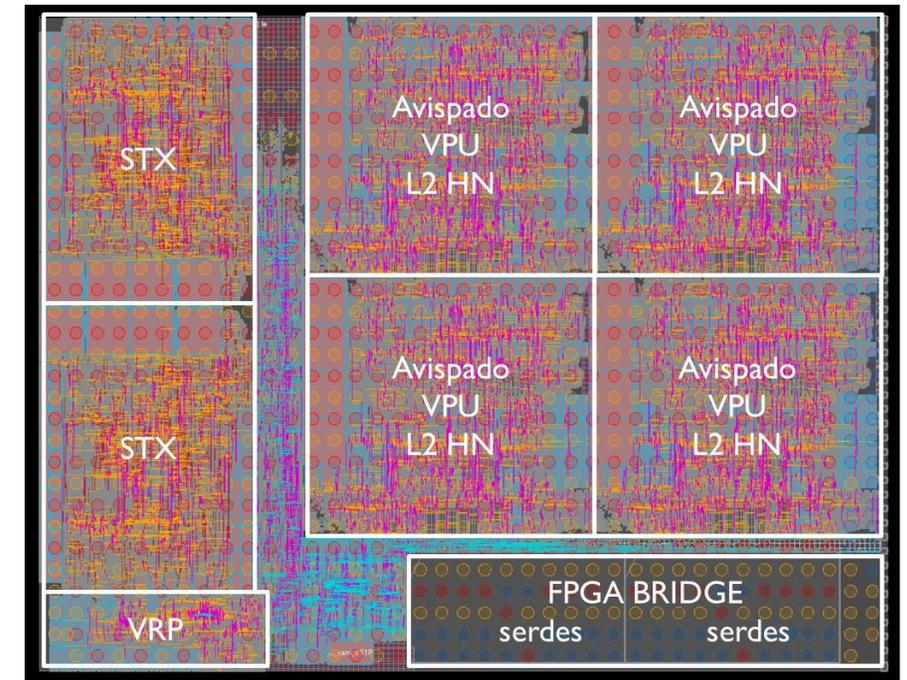
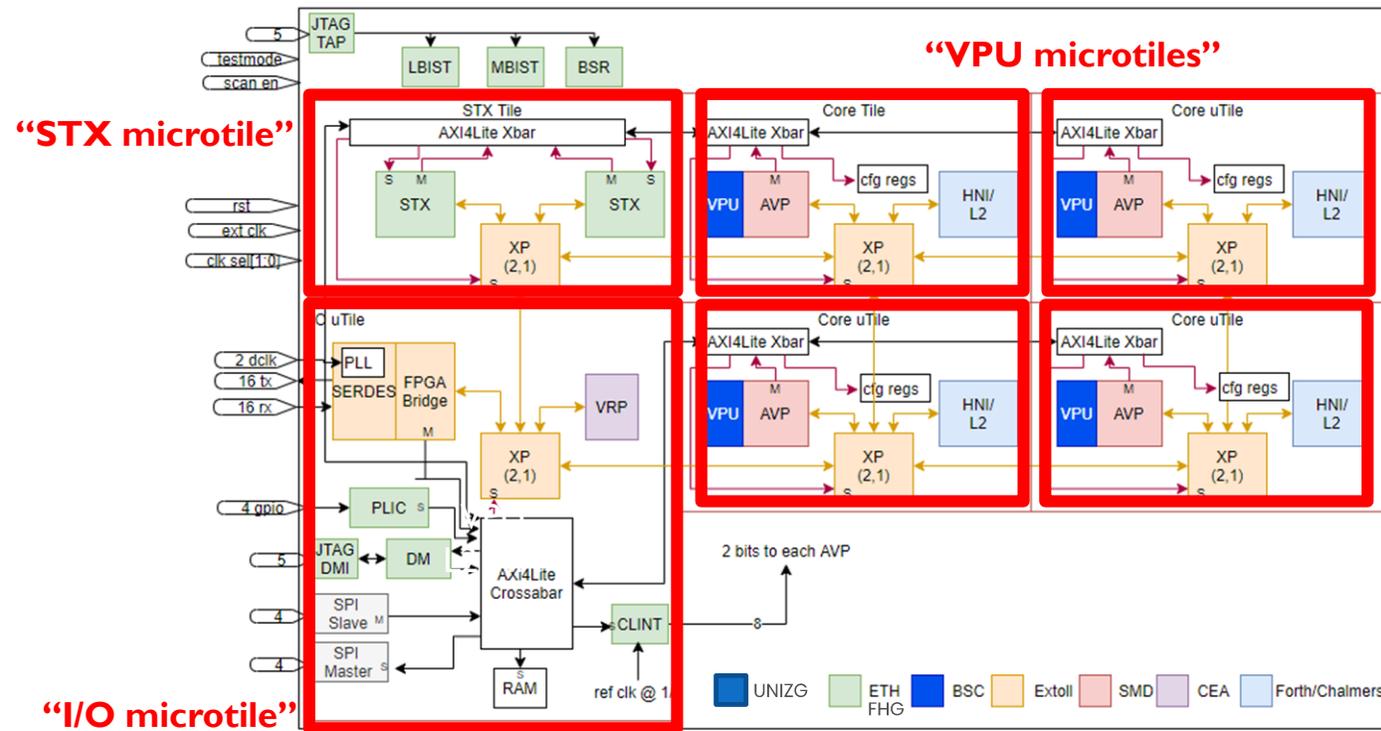
# VPU: A PROCESSOR IN ITSELF

- Hierarchical “accelerator” integration
  - Program & data served by scalar core (Coherence; ~punch tape program 😊)
  - Fine grain “offloading” of “vector tasks” (directly hardware supported)
  - Homogenized heterogeneity under single “standard” ISA interface defining program order
  
- Implementation
  - #FUs  $\ll$  VL (lanes=8, VL=256)
  - Some OoO
    - Resources to overlap?
      - L/S, FU, shuffling
    - Renaming
      - 40 physical registers
  - Single ported register file
    - Large state
    - 5 banks/lane providing sufficient bandwidth for 1 op/cycle (latency/BW trading)
  - Data shuffling: directional ring



“Vitruvius: An Area-Efficient Decoupled Vector Accelerator for High Performance Computing”  
 F. Minervini, O. Palomar. RISC-V Summit 2021

# EPAC TEST CHIP



# TEST CHIP AND BOARD



```

happy@epac$ axpy 1024
Running AXPY Scalar with 1024 array elements
init time: 45060 cycles

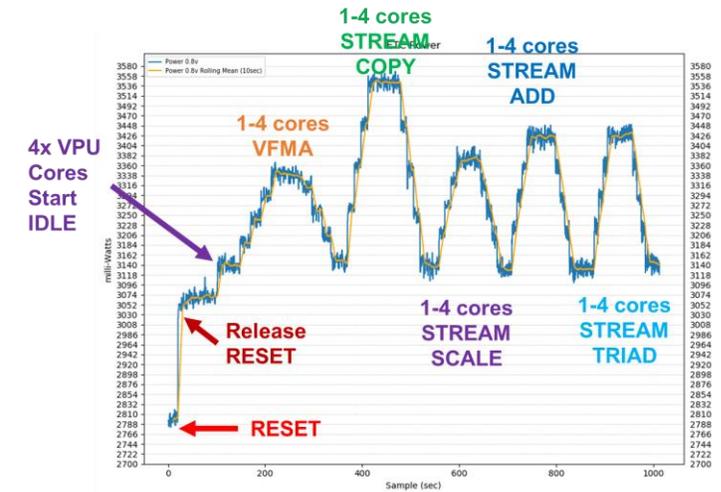
axpy scalar reference time: 23555 cycles

done
Result ok !!!
happy@epac$ vaxpy 1024
Running AXPY Vector with 1024 array elements
init time: 45043 cycles

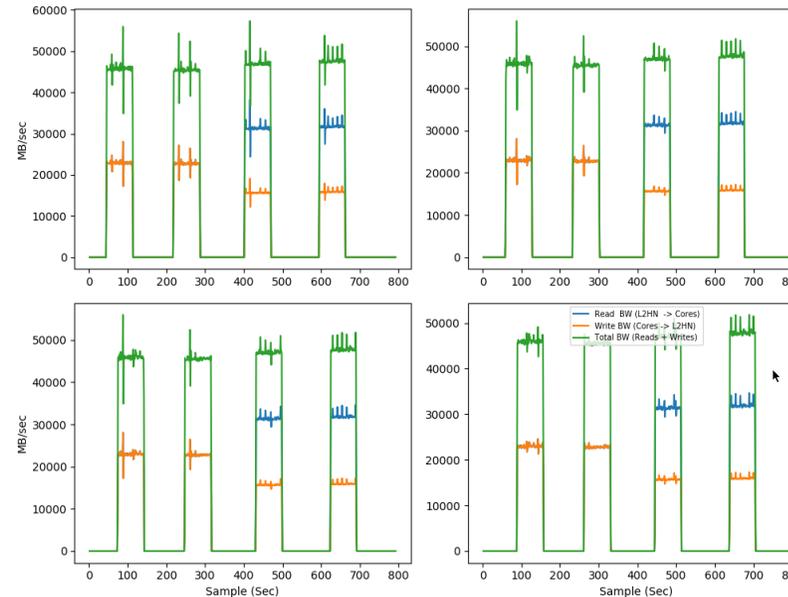
axpy vector time: 932 cycles

done
Result ok !!!
happy@epac$ █
  
```

~25x  
 while only 8x FPU's 😊  
 → Long vectors !!  
 → Memory Bandwidth



copy scale add triad



```

epac@EPAC:~/Desktop/etc_tools-master_v20211206/etc_tools-m
[sudo] password for epac:
Waiting for server ...
Connection with server established!
EPAC JTAG Console Client 0.1
Connecting to JTAG Console [3] ...
Press CTRL+A for exit

-----
| Welcome to EPAC TC Bring-Up Shell |
-----

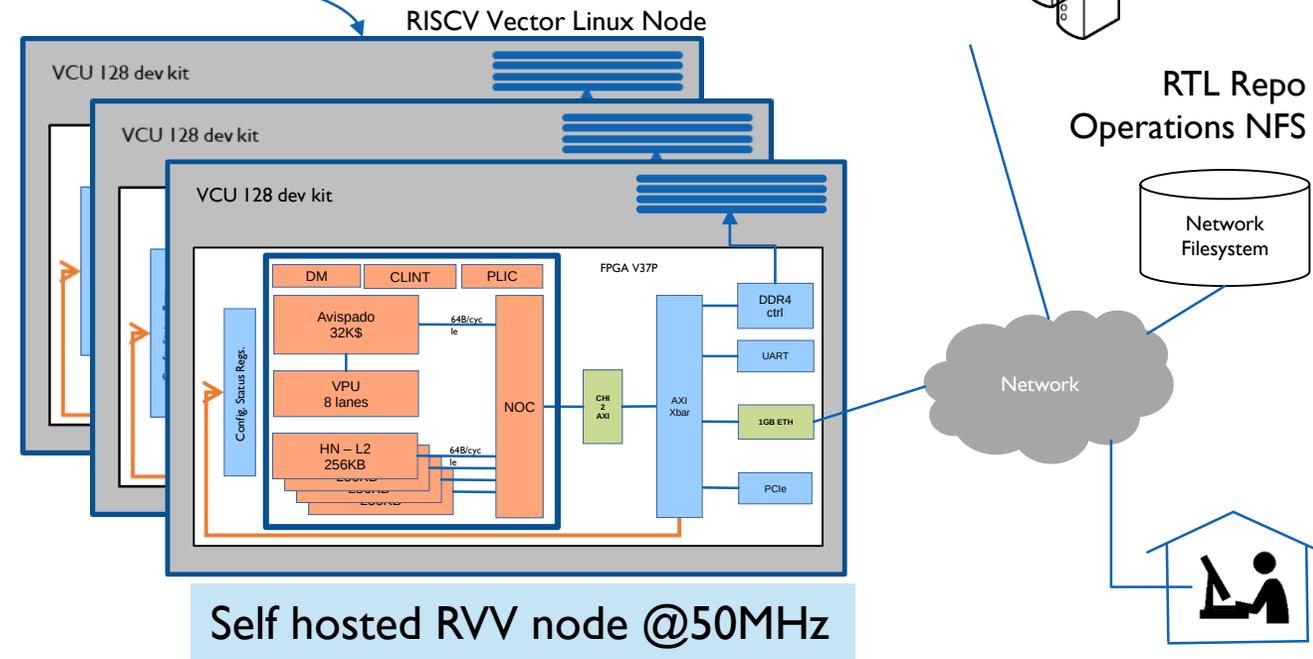
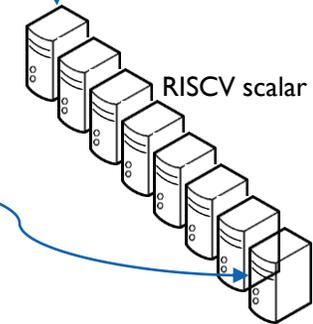
user@epac$ jpeg_benchmark

JPEG scalar reference time: 255667444 cycles

done
user@epac$ █
  
```

# EPI SDV ECOSYSTEM

- RISC-V cluster
  - Commercially available RISC-V platforms
  - Porting and configuring HPC software stack and increase productivity (e.g., SLURM, MPI, OpenMP, BSC tools, SDV1.2)
- SDV: RVV @ FPGA nodes
  - CI Infrastructure: Validation at “scale”
  - Software development and co-design steering
    - Test real “complex” codes @ real RTL
    - EPAC1.5 RTL improvement
    - Give to EPI partners and interested users easy access to the latest EPAC technology
      - Two step procedure



# SYSTEM SOFTWARE @ SDV – ENVIRONMENT

## OS



- Kernel: Efficient context switches, large pages, network drivers, PMU, DTB relocation patch,...
- Root file system: Busybox / Buildroot / Debian / Ubuntu
  - Dynamic linked libraries, package installations, start/shutdown services (LDAP, NFS)

## Compiler



- Intrinsics
- Automatic vectorization

## MPI



- OpenMPI @ 1 GbE

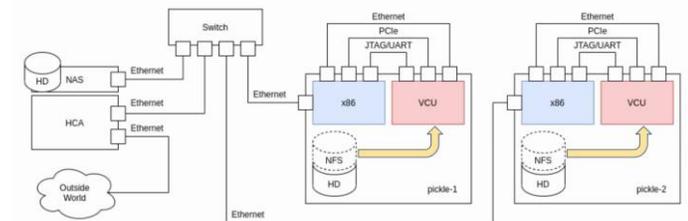
```
gramirez@epac:~/pt2pt$ mpirun -np 2 --hostfile hostfile ./osu_bw
# OSU MPI Bandwidth Test v5.8
# Size      Bandwidth (MB/s)
1           0.00
2           0.00
4           0.00
8           0.00
16          0.01
32          0.01
64          0.02
128         0.04
256         0.07
512         0.11
1024        0.15
2048        0.16
4096        0.17

gramirez@epac:~/pt2pt$ mpirun -np 2 --hostfile hostfile ./osu_latency
# OSU MPI Latency Test v5.8
# Size      Latency (us)
0           8060.92
1           3471.22
2           3515.94
4           3786.58
8           3568.45
16          3622.08
32          3708.37
64          3810.36
128         4183.88
256         4989.66
512         6430.31
1024        8450.97
2048        15445.54
4096        24106.76
```

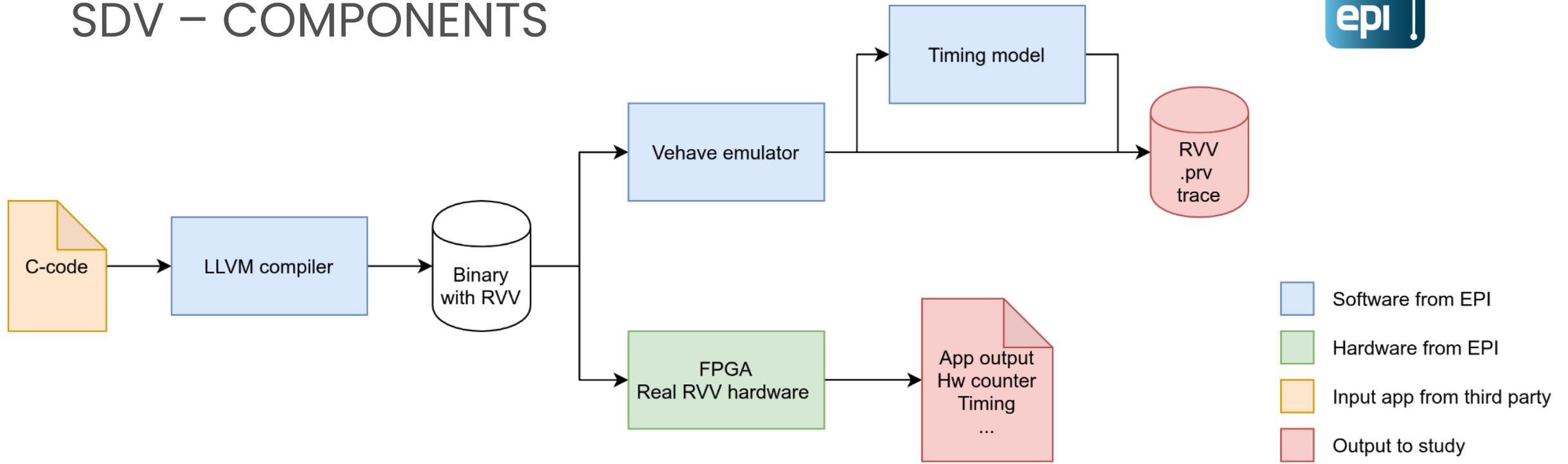
```
...
float complex A[n][n];
float complex temp1, temp2;
...
for (int j = 0; j < n; j++) {
    if (x[j] != ZERO || y[j] != ZERO) {
        temp1 = alpha * conjunction(creal(y[j]), cimag(y[j]));
        temp2 = conjunction(creal(alpha * x[j]), cimag(alpha * x[j]));
        #pragma clang loop vectorize(assume_safety)
        for (int i = 0; i <= j - 1; i++) A[i][j] = A[i][j] + x[i] * temp1 + y[i] * temp2;
        A[j][j] = creal(A[j][j]) + creal(x[j] * temp2);
    } else A[j][j] = creal(A[j][j]);
}

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;
            }
        }
    }
}

void SpMV_vec(double *a, long n, long m) {
    for (int row = 0; row < n; row++) {
        int nnz_row = ia[row];
        unsigned long gvl;
        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,
                __epi_m1);
            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_vsl1_1xi64(v_idx_row, v_three, gvl);
            v_x = __builtin_epi_vload_indexed_1xf64(x, v_idx_row, gvl);
            v_prod = __builtin_epi_vfmul_1xf64(v_a, v_x, gvl);
            v_partial_res = __builtin_epi_vfredsum_1xf64(v_prod,
                v_partial_res, gvl);
            colid += gvl;
        }
        y[row] += __builtin_epi_vgetfirst_1xf64(v_partial_res,gvl);
    }
}
```



# SDV – COMPONENTS



RVV = RISC-V with Vector Extension Support

Vector support is provided through:

- Inline intrinsics
- Auto-vectorization by the compiler

Current FPGA implementation 1 core @ 50 MHz  
Expect to scale to dual/quad core by end of 2022

Full linux support

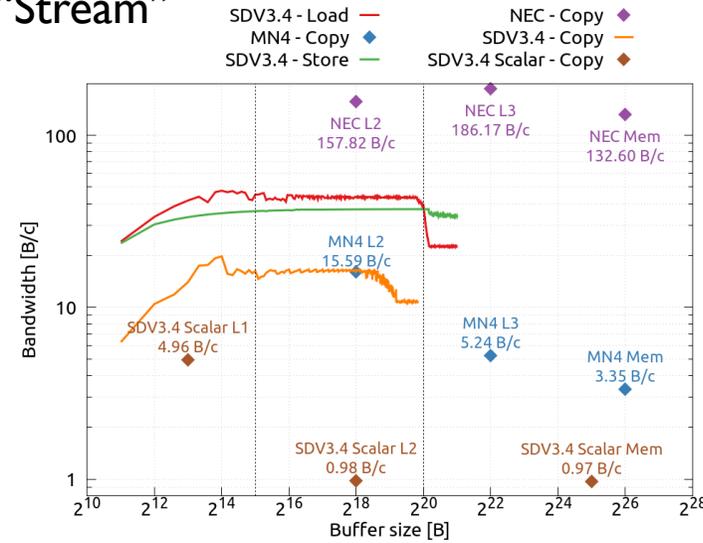
More info: <https://repo.hca.bsc.es/gitlab/epi-public/risc-v-vector-simulation-environment>

# SDV – VECTOR PERFORMANCE

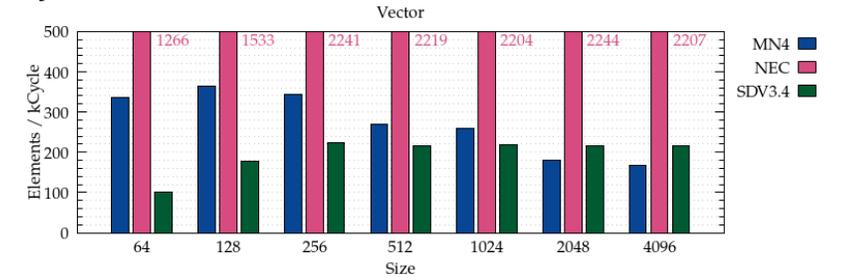
- EPAC ...
- ... vs. state of the art

(Xeon, NEC, A64FX, FU740, ...)

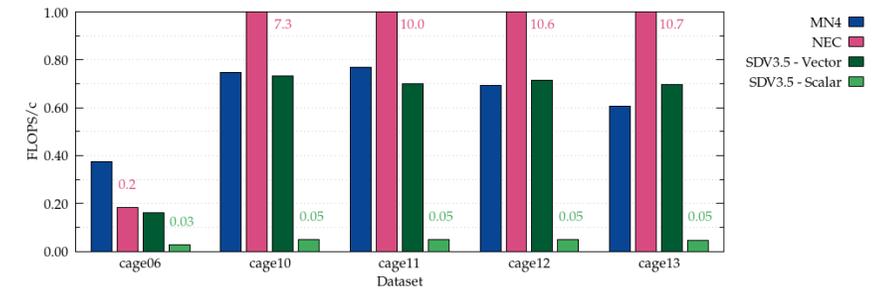
## “Stream”



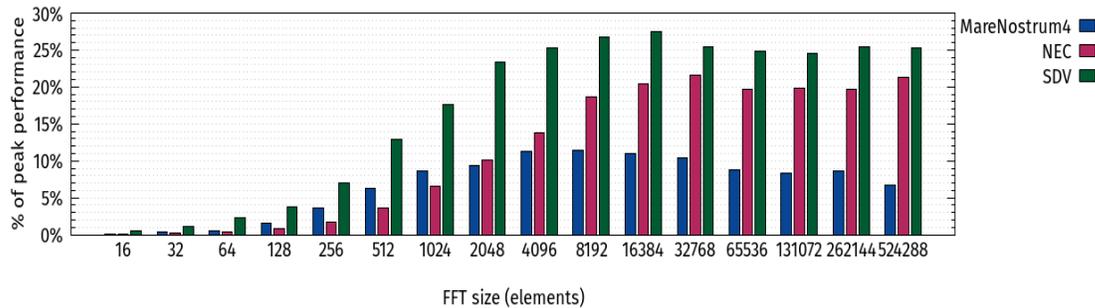
## Jacobi 2D



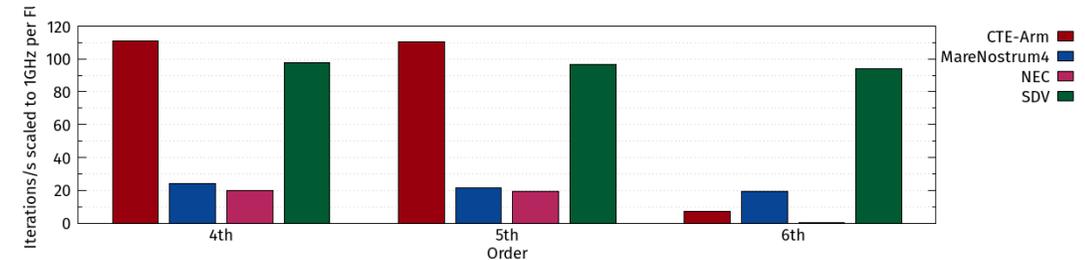
## SpMV



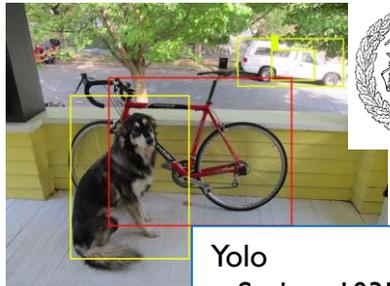
## FFT



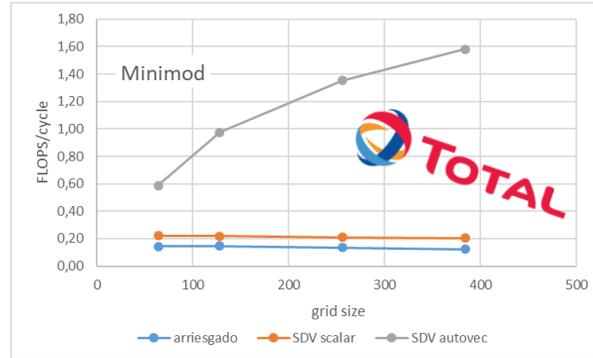
## HACC



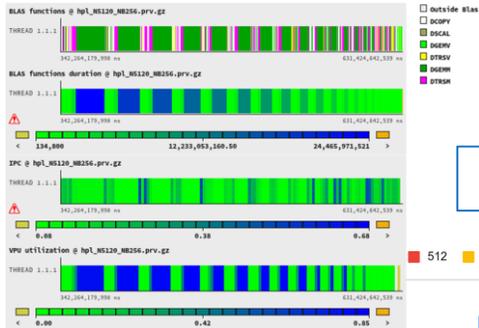
# SDV – VECTOR IN HPC AND BEYOND



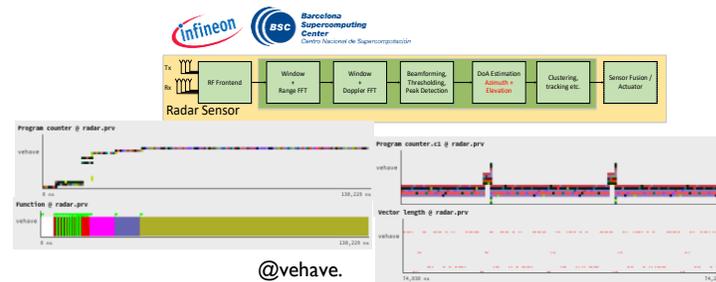
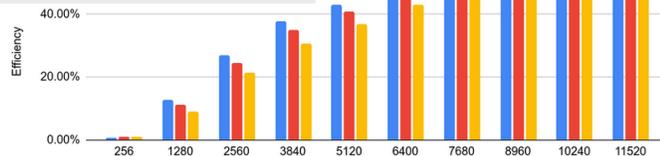
**Yolo**  
 Scalar: 1035.5 s  
 Vector: 29.3 s



**Bolt**  
 Intel (2.5GHz): 0.0303 s  
 Unmatched (1.2GHz): 0.1634 s  
 SDV Scalar (50MHz): 5.8462 s  
 SDV Vector (50MHz): 3.7104 s



**Linpack**



@vehave.

THE EUPILOT

- GROMACS
- EC-EARTH
- OneDNN

UNIVERSIDAD COMPLUTENSE MADRID

- BLIS

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

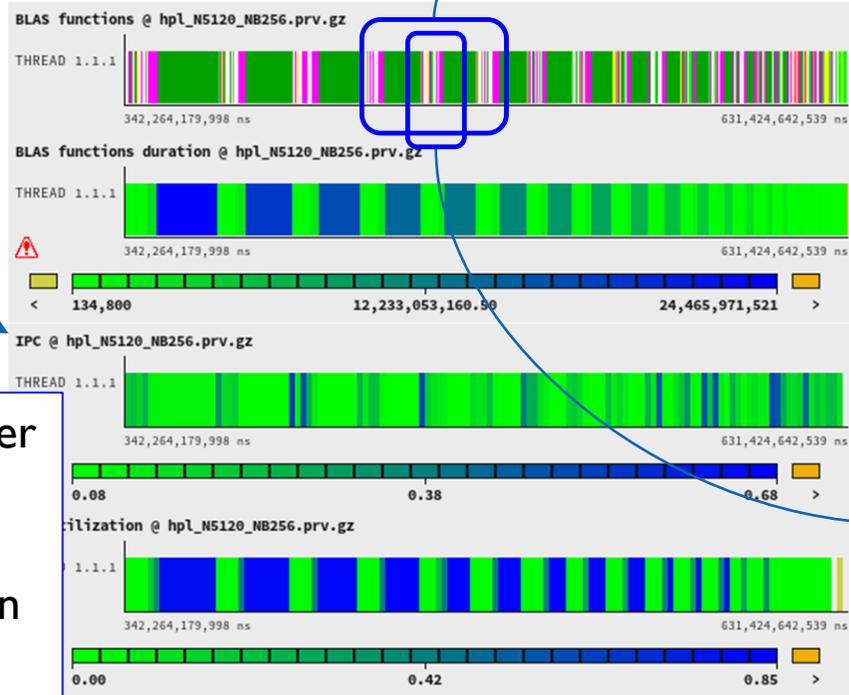
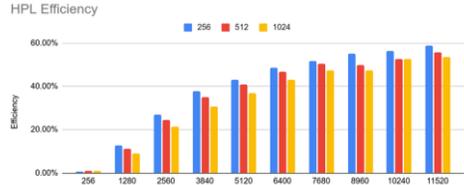
- Ginkgo

MEEP

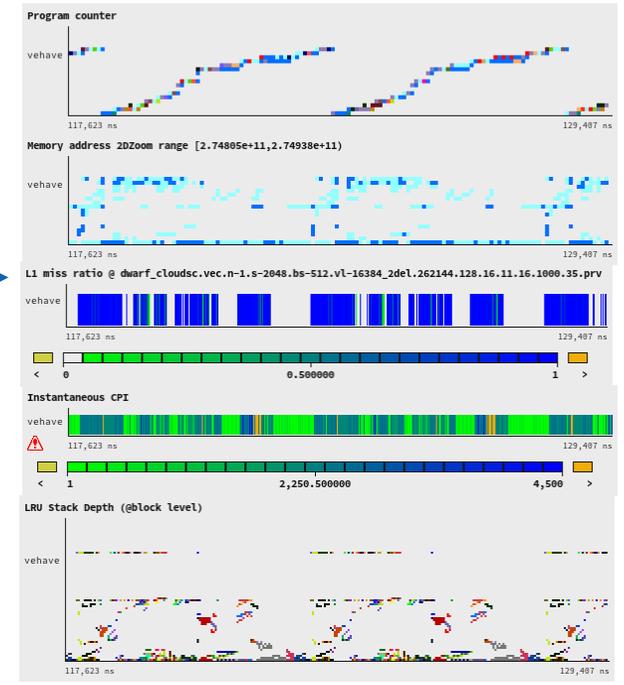
- Climate dwarfs
- TFLite
- Containers
- PyCOMPSs
- Spark

- Pytorch
- PostgreSQL
- Sorting

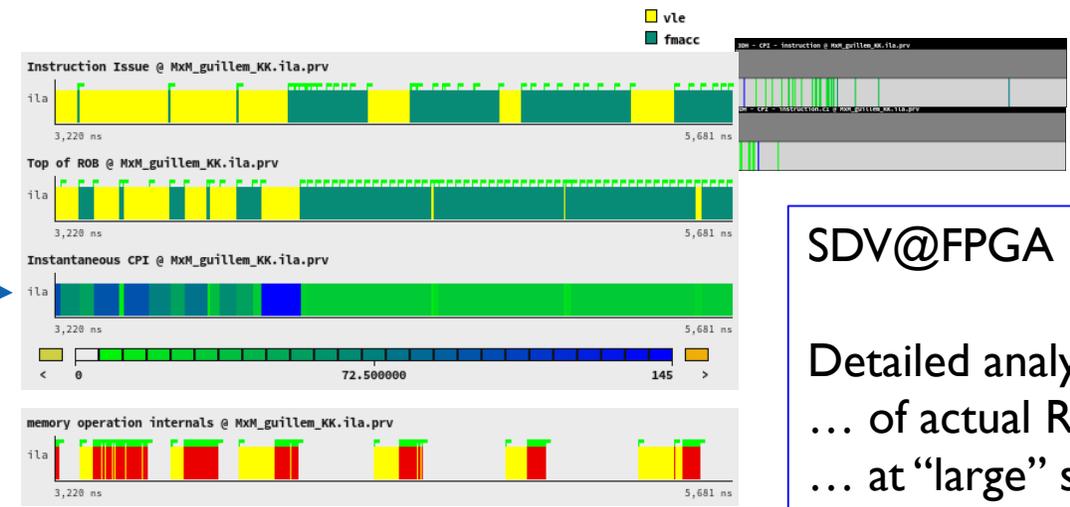
# LOD IN BEHAVIOR ANALYSIS



Extrac + Paraver  
Standard HPC instrumentation analytics and visualization



Vehave + MUSA  
Detailed analysis ...  
... of flexible what-ifs  
at ISA/ $\mu$ Arch level

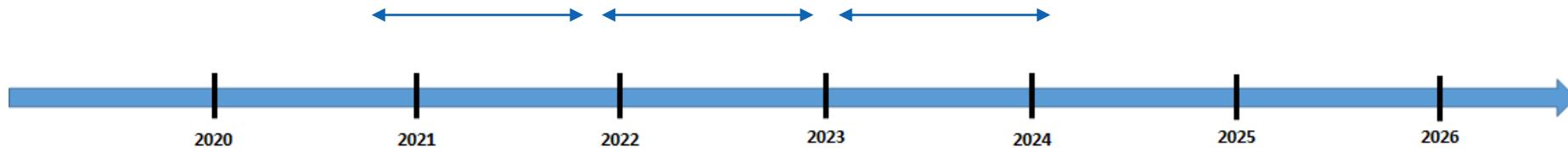


SDV@FPGA  
Detailed analysis ...  
... of actual RTL ...  
... at "large" scale

# THE EPI/EUPILOT RVV CHIP ROADMAP



~ yearly RTL dev. cycle



# THE IMPORTANCE OF A VISION

- RISC-V: also an option for HPC
- Holistic throughput oriented vision based on long vectors and task based models
- Hierarchical concurrency and locality exploitation
  - Not massive concurrency at a given level
  - Push behaviour exploitation to low levels
- Co-ordination between levels
- Make it all look very close to classical sequential programming to ensure productivity

### EPAC & Sagrada Familia ?

- There is something “special” ...
- ...showing the way ...
- ... sustaining the effort

Sagrada Familia 1975

**RISC-V**



# THANKS

[HTTPS://WWW.EUROPEAN-PROCESSOR-INITIATIVE.EU/](https://www.european-processor-initiative.eu/)

The European Processor Initiative (EPI) has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement EPI-SGA1: 826647 and under EPI-SGA2: 101036168. Please see <http://www.european-processor-initiative.eu> for more information.