



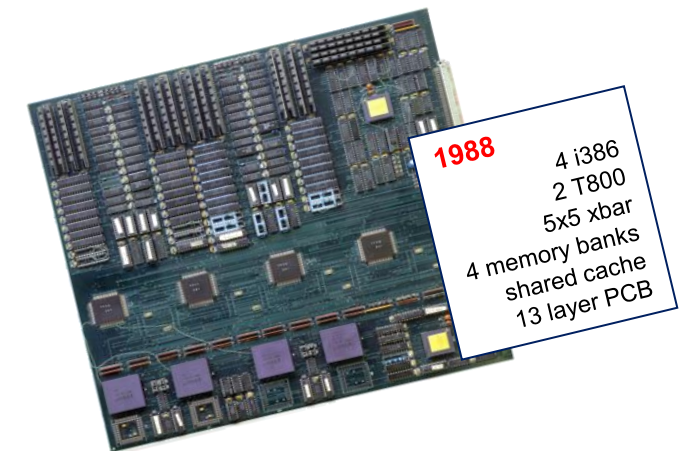
# THE RISC-V VECTOR PROCESSOR IN EPI

JESUS LABARTA (@BSC.ES)

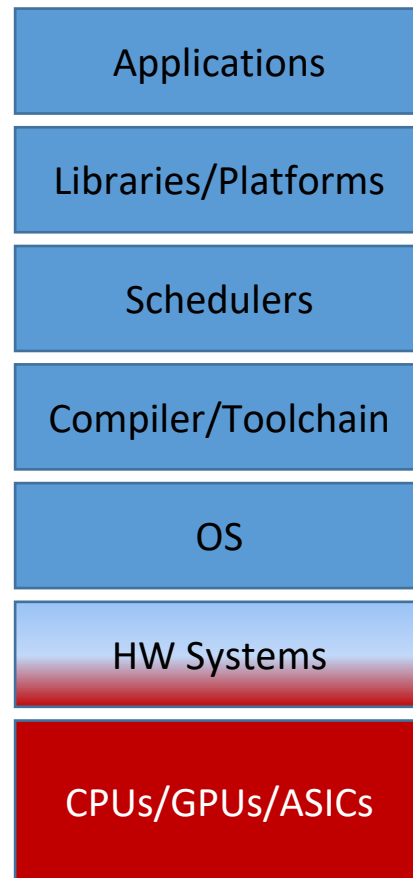
# DISCLAIMER

- Personal opinions
- I grew up in HPC-land
  - In a Computer Architecture Department ...
  - ... but moved to the dark side (Software) ...
  - ... back to the future?

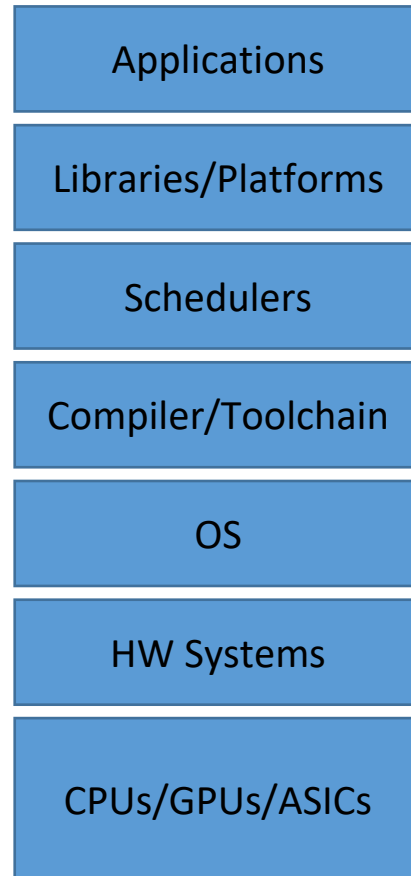
El abuelo cebolleta  
ataca de nuevo



# TOWARDS PROCESSOR DESIGN IN EUROPE



# TOWARDS PROCESSOR DESIGN IN EUROPE



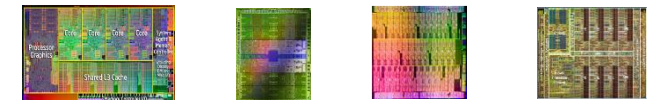
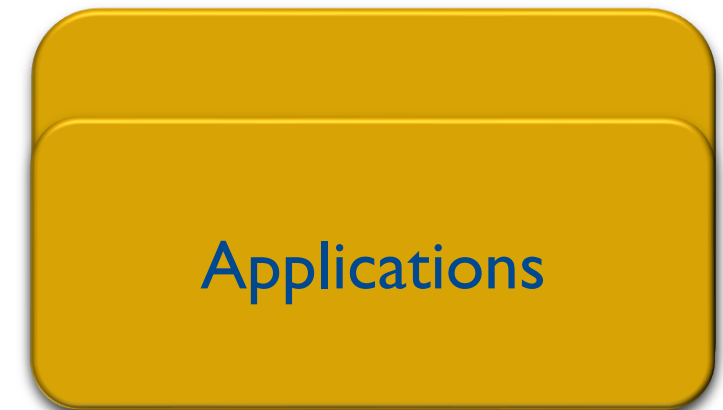
## BUT BEFORE ....

- About Osmotic Membranes in HPC programming
- Thoughts and vision on co-design
- The shoemaker son going barefoot

# VISION

- The multicore and memory revolution
  - ISA leak ...
  - Plethora of architectures
    - Heterogeneity
    - Memory hierarchies
- Complexity + variability = Divergence ...
  - ... between our mental models and actual system behavior

The power wall made us go multicore  
and the ISA interface to leak  
→ our world is shaking



## What programmers need ?

## HOPE !!!

# VISION IN THE PROGRAMMING REVOLUTION

Applications

PM: High-level, clean, abstract interface

Power to the runtime

ISA / API

Mem.. Ac.. Co...res cel... Vis... ..ory ..sual... Sto...rage ..lization ..ory Co...res mem..

General purpose

Task & data based

Forget about resources

Decouple:  
Minimal & sufficient permeability?

Intelligence  
&  
Resource management

“Reuse & expand” old architectural ideas  
under new constraints

# THE PM OSMOTIC MEMBRANE

Applications

PM: High-level, clean, abstract interface

Power to the runtime

ISA / API

Mem.. Ac.. Co...res cel... Vis... ..ory ..sual... Sto...rage ..lization ..ory Co...res mem..

General purpose

Task & data based

Forget about resources

Decouple:

**Minimal & sufficient permeability?**

Intelligence  
&

Resource management

“Reuse & expand” old architectural ideas  
under new constraints



# TOWARDS (SUPER)COMPUTER DESIGN

- Holistic design of complex systems ...

Applications

Libraries/Platforms

Schedulers

Compiler/Toolchain

OS

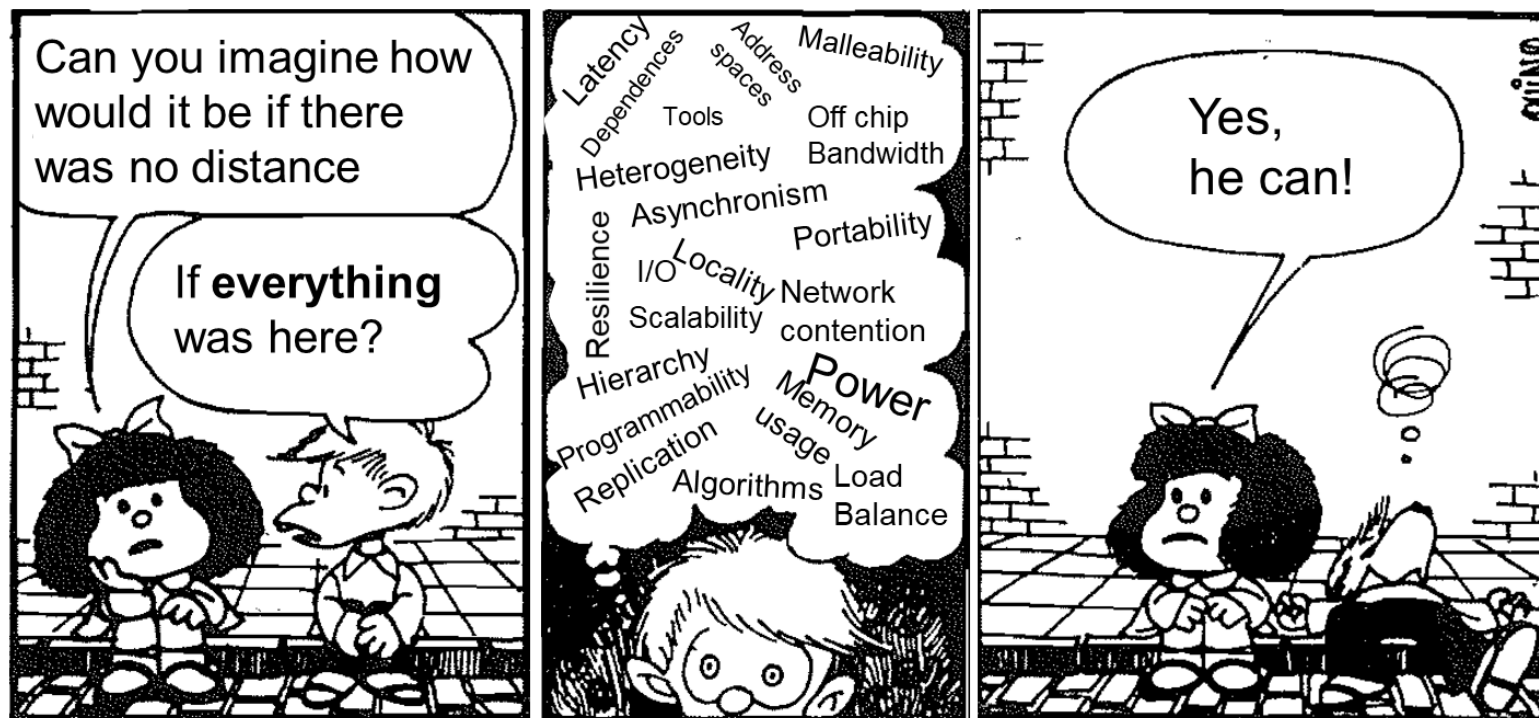
HW Systems

CPUs/GPUs/ASICs

# TOWARDS (SUPER)COMPUTER DESIGN

- Holistic design of complex systems ...

...a risk



Applications

Libraries/Platforms

Schedulers

Compiler/Toolchain

OS

HW Systems

CPUs/GPUs/ASICs

# CO-DESIGN

Applications

Libraries/Platforms

Schedulers

Compiler/Toolchain

OS

HW Systems

CPUs/GPUs/ASICs

A buzzword !!!!

Top → down !!!!  
and obsessed by hardware

Limited scope !!!!

# CO-DESIGN

Applications

Libraries/Platforms

Schedulers

Compiler/Toolchain

OS

HW Systems

CPUs/GPUs/ASICs

You have to DESIGN !!!!

# HOLISTIC CO-DESIGN

Applications

Libraries/Platforms

Schedulers

Compiler/Toolchain

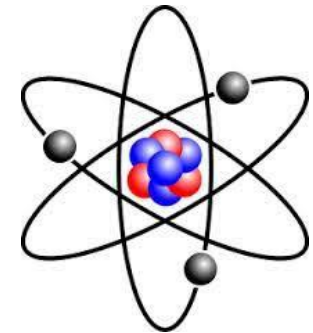
OS

HW Systems

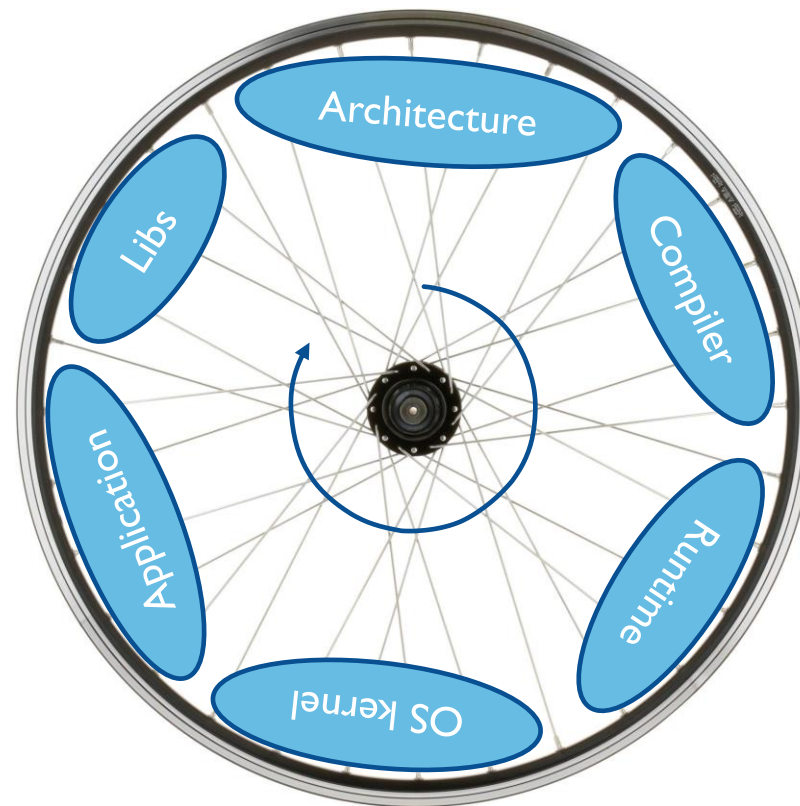
CPUs/GPUs/ASICs

“As above, so below”

Similar concepts/mechanisms at all levels



# HOLISTIC CO-DESIGN



Best place to address an issue

Fundamentals

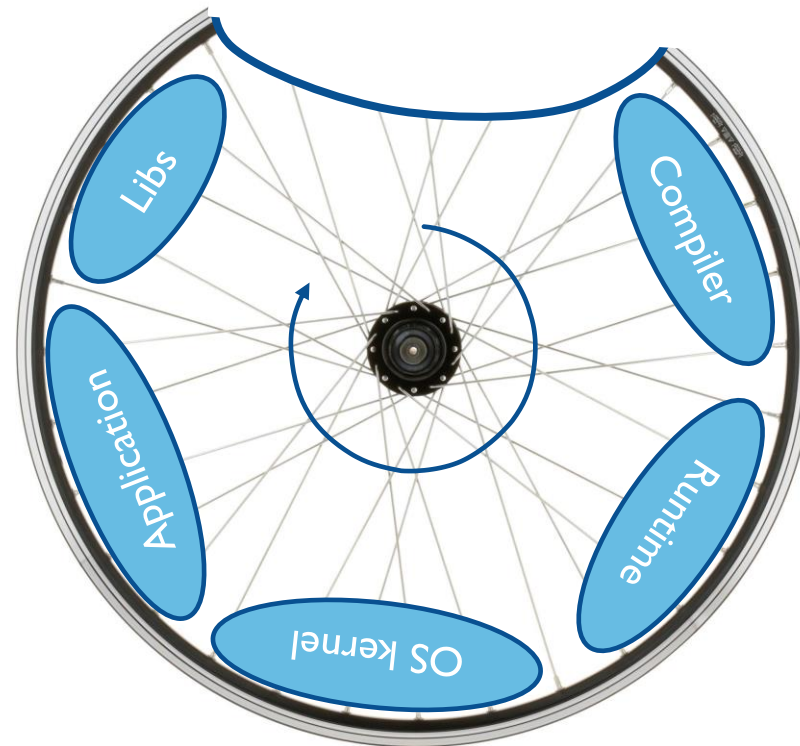
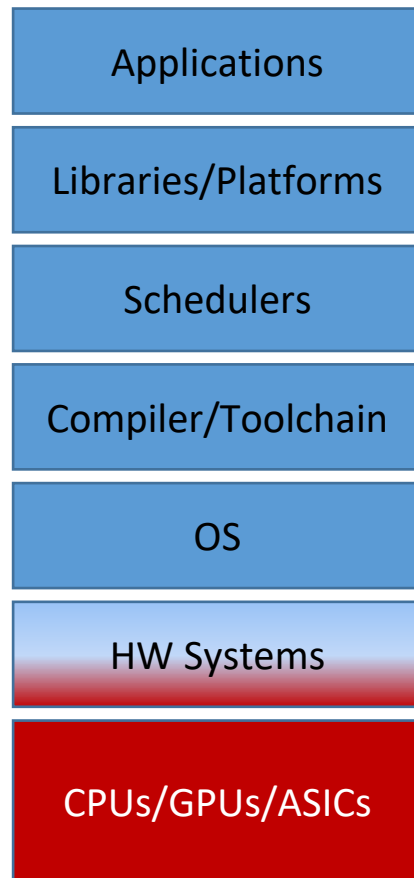
Balance

Mindset

Productivity

Efficiency

# HOLISTIC CO-DESIGN



Best place to address an issue

Fundamentals

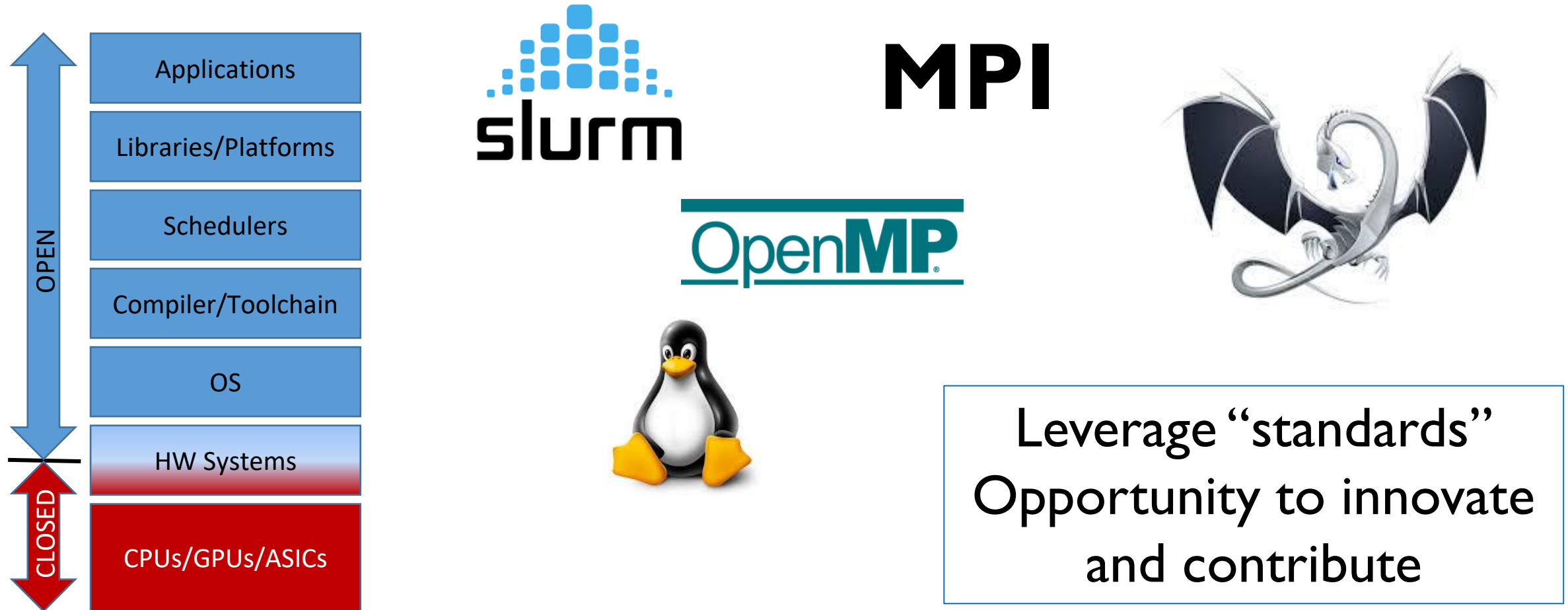
Balance

Mindset

Productivity

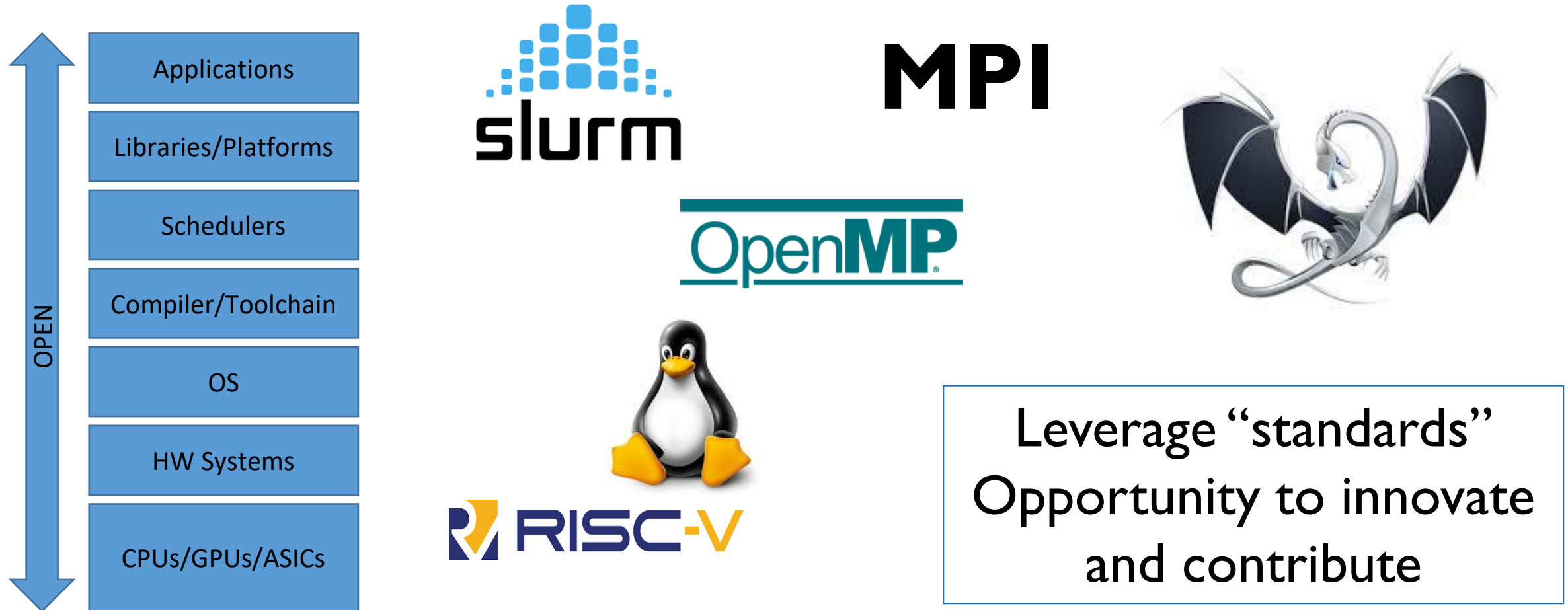
Efficiency

# LEVERAGE INTERFACES AND IMPLEMENTATIONS



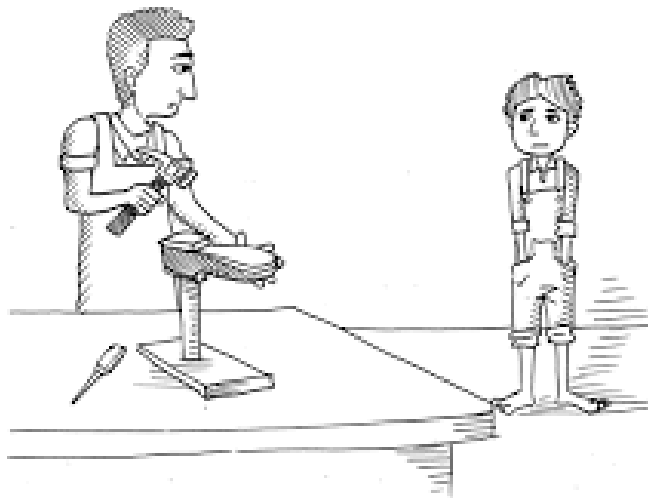


# LEVERAGE INTERFACES AND IMPLEMENTATIONS



## SHOEMAKER'S SON ...

...always goes barefoot



En casa del herrero ...



... cuchara de palo



# DETAILED ANALYSIS AND **INSIGHT** ON BEHAVIOR

Applications

Libraries/Platforms

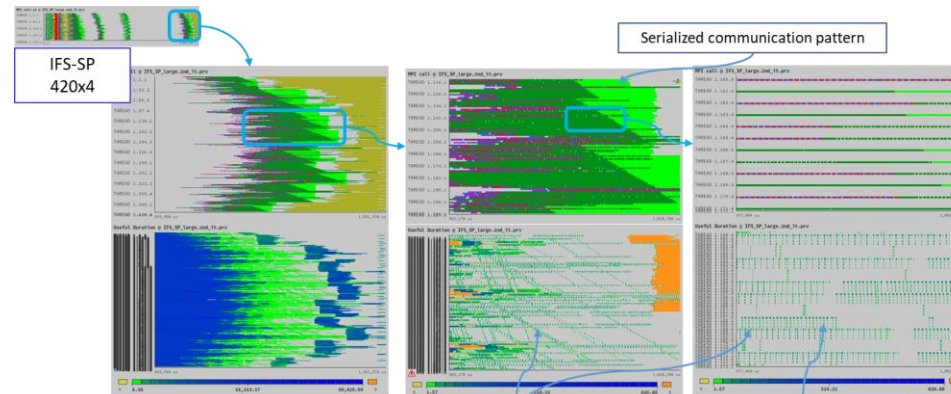
Schedulers

Compiler/Toolchain

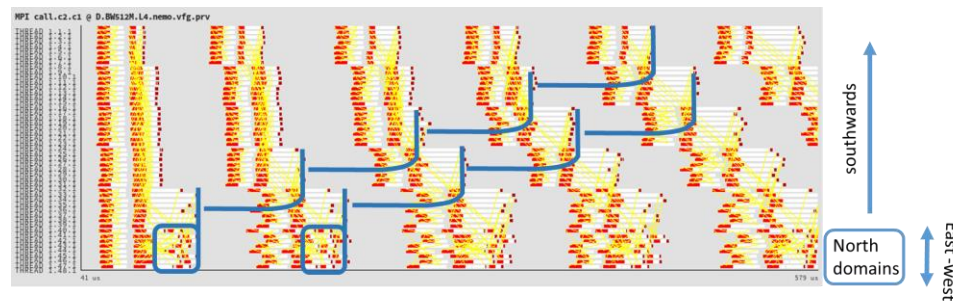
OS

HW Systems

CPUs/GPUs/ASICs



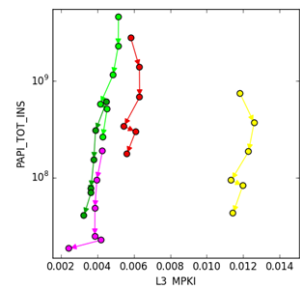
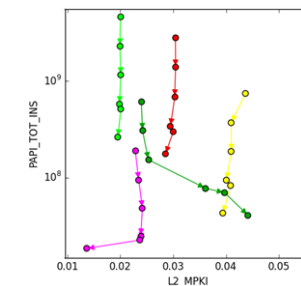
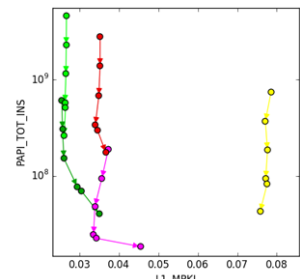
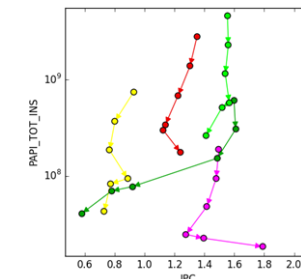
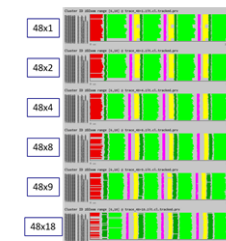
Detailed analysis



What if simulations



<https://pop-coe.eu/>

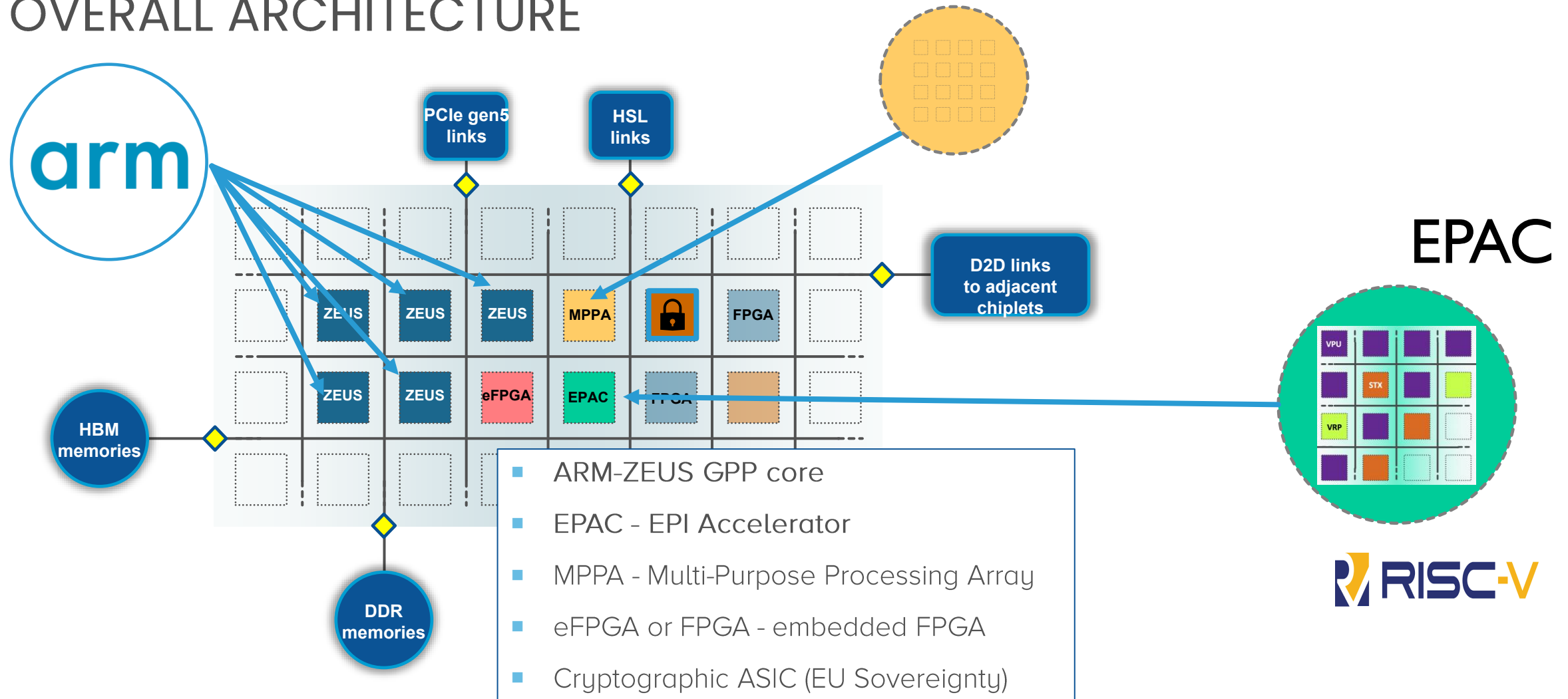


Performance analytics

# EPI

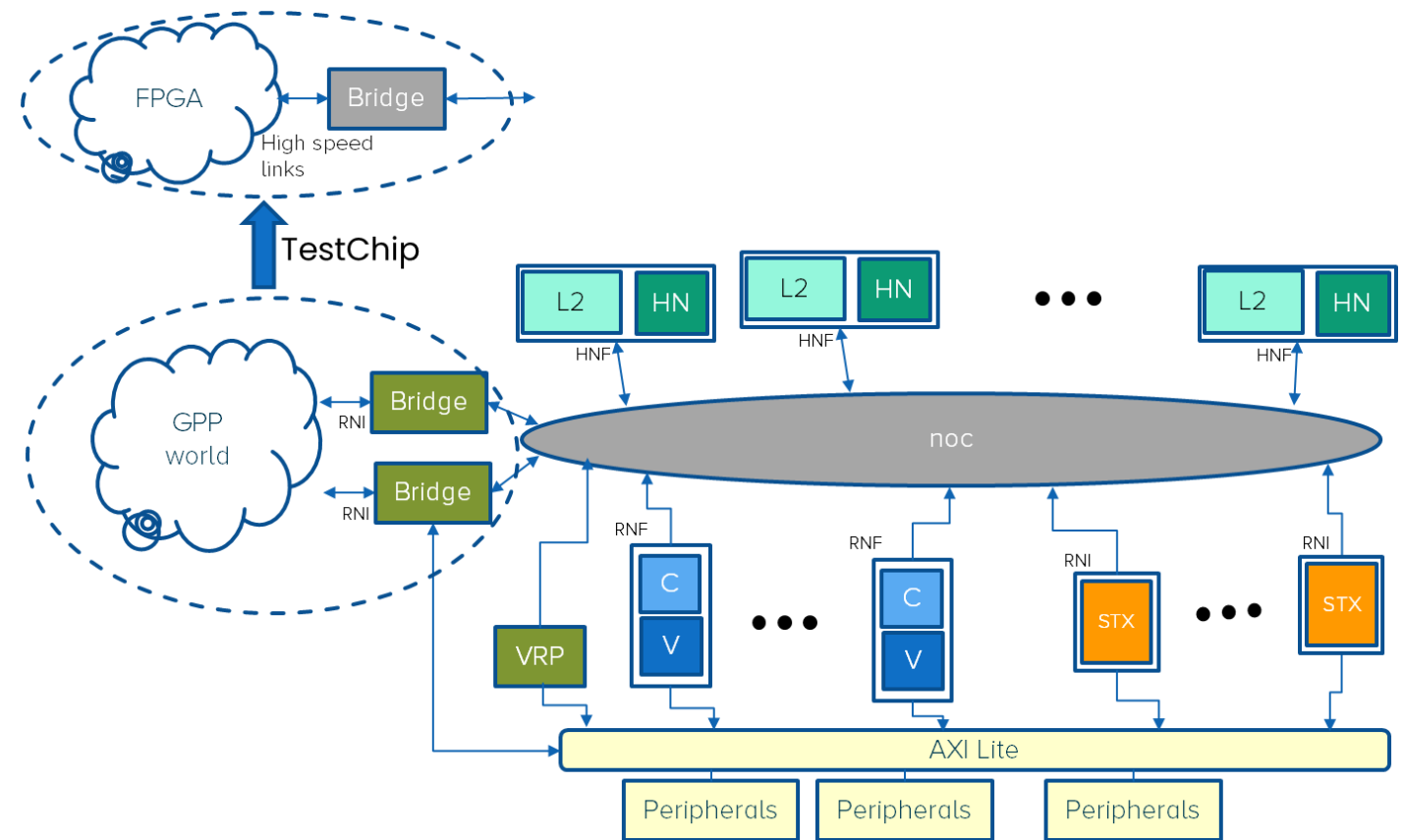
- The global architecture
- The RISC-V tile
- Guiding principles
- Where we are

# OVERALL ARCHITECTURE



# EPAC

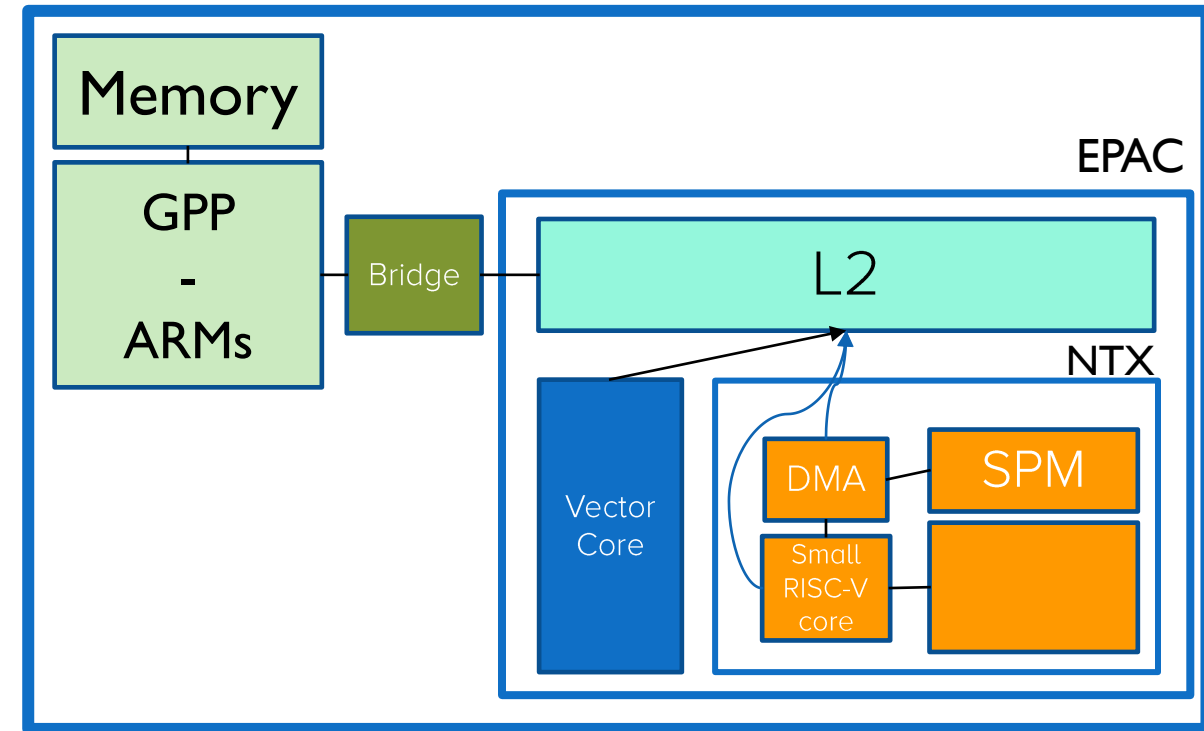
- RV64GCV (→ 8x)
  - 2 way in order core
  - Decoupled OoO VPU
    - 8 lanes
    - Long vectors (256 DP elements)
  - L1 - MESI coherency
- CHI interface NoC
  - 1 line / cycle (high B/F)
- L2: 256KB/module
  - Allocation control mechanisms
- Linux



- STX: DL and stencil specific accelerators
- VRP: variable precision processors

# EPI: A HIGHLY HETEROGENEOUS/HIERARCHICAL SYSTEM

EPI

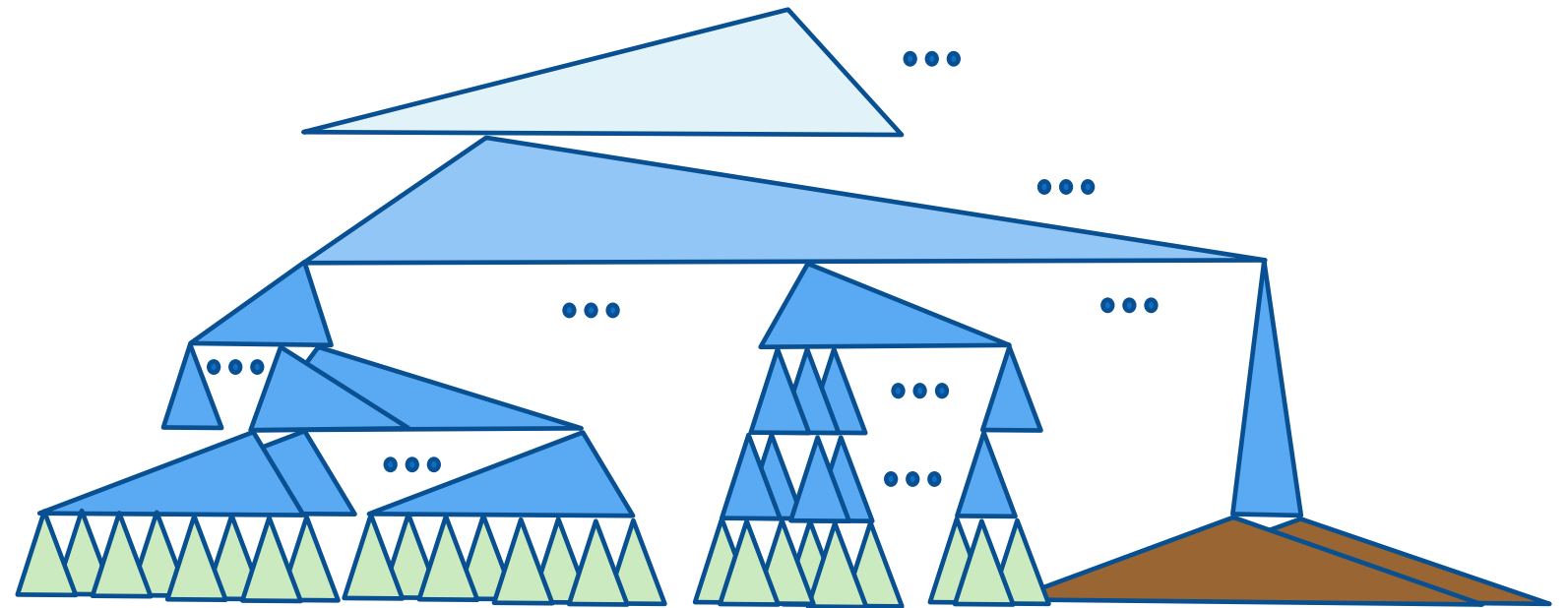
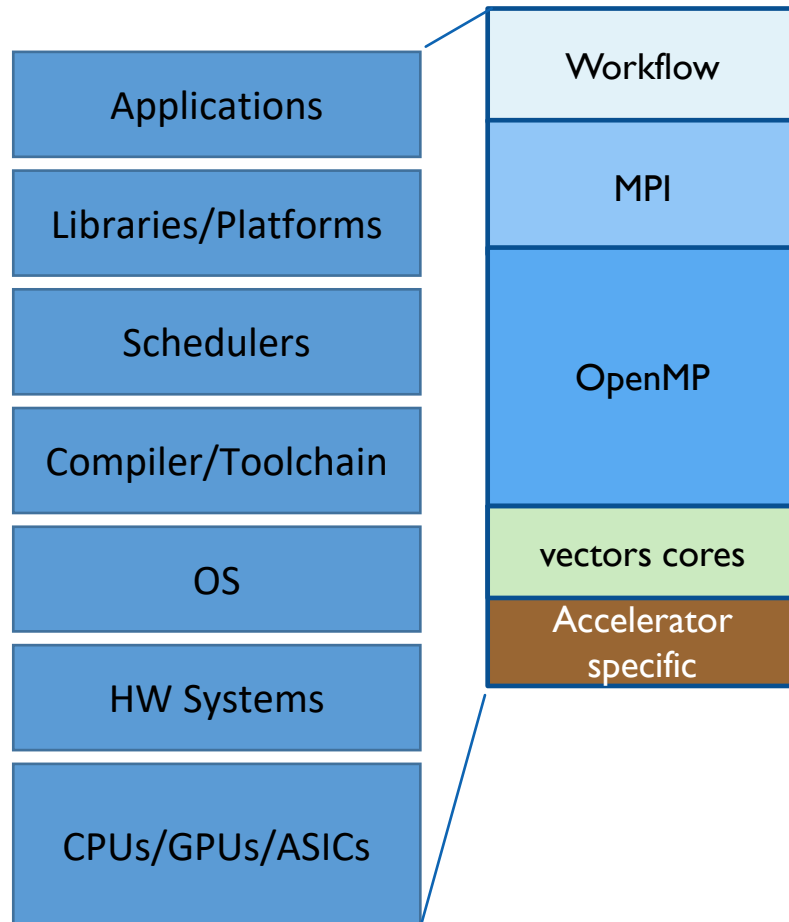


# FOUR PRINCIPLES

- Balanced hierarchy
- Latency → throughput
- Malleability and coordinated scheduling
- Homogenize heterogeneity

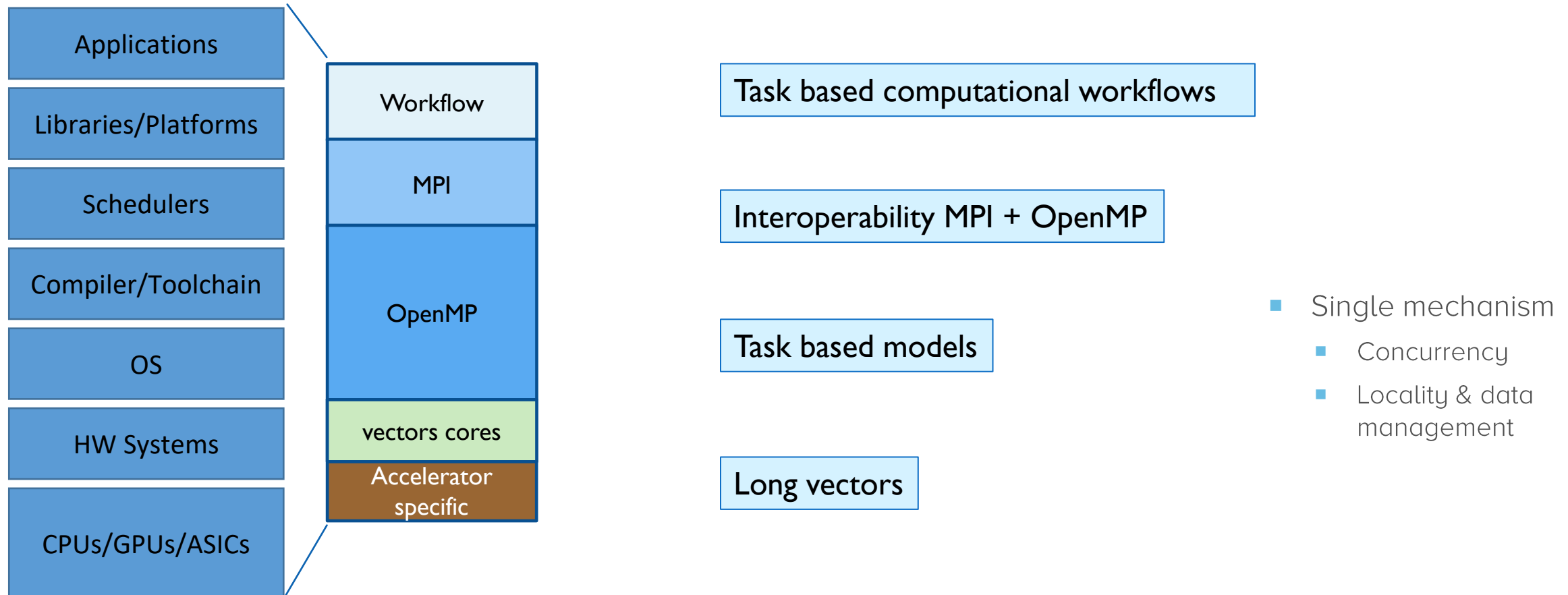


# BALANCED HIERARCHY



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

# LATENCY → THROUGHPUT: ASYNCHRONY AND OVERLAP



# INTEGRATE CONCURRENCY AND DATA

## Single mechanism

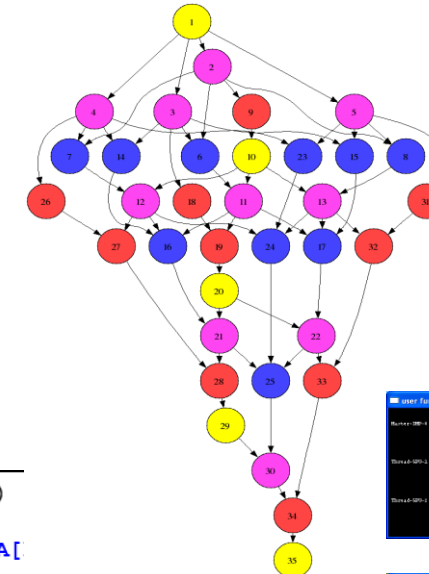
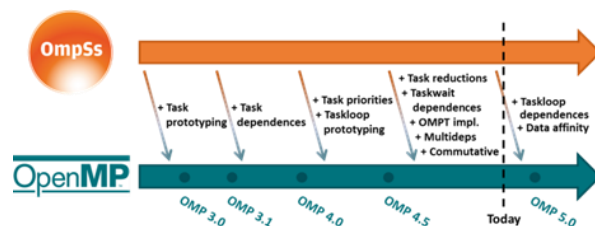
### Concurrency:

Dependencies built from data accesses

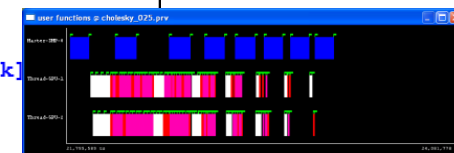
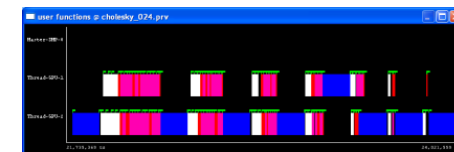
Lookahead: About instantiating work

### Locality & data management

From data accesses

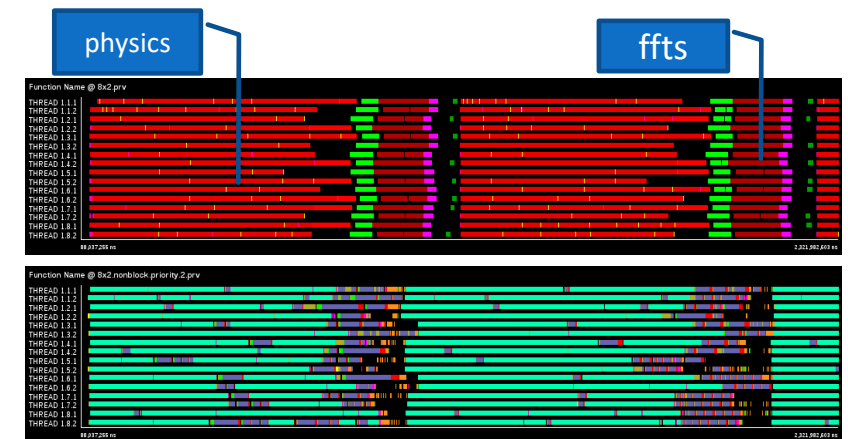
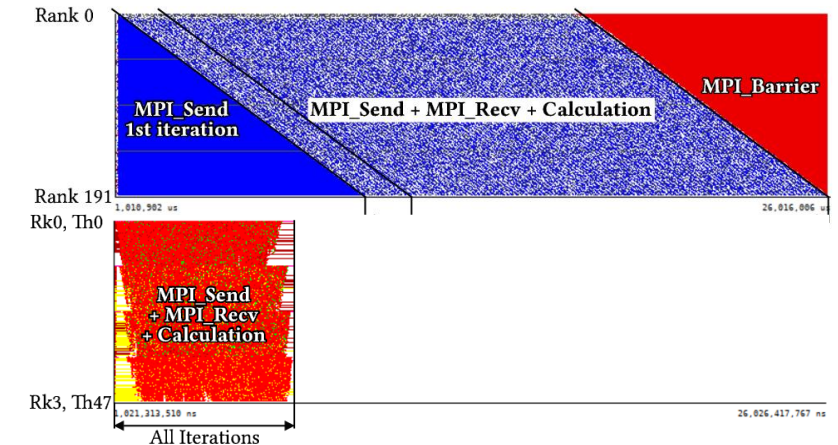
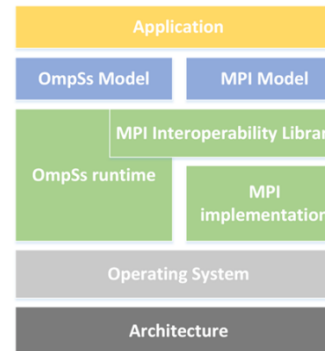


```
void Cholesky(int NT, float *A[NT][NT])
{
    for (int k=0; k<NT; k++) {
        #pragma omp task inout ([TS][TS](A[
        ● spotrf (A[k][k], TS) ;
        for (int i=k+1; i<NT; i++) {
            #pragma omp task in([TS][TS](A[k][k])) inout ([TS][TS](A[k]
            ● strsm (A[k][k], A[k][i],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



# MPI – OPENMP/OMPSS INTEROPERABILITY

- Taskifying MPI calls
  - Virtualize “communication resource”
- Opportunities
  - Overlap/out of order execution
    - Computation - communication
    - Communication - communication
    - Phases / iterations
  - Provide laxity for communications
    - Tolerate poorer communication
  - Migrate/aggregate load balance issues
    - Flexibility

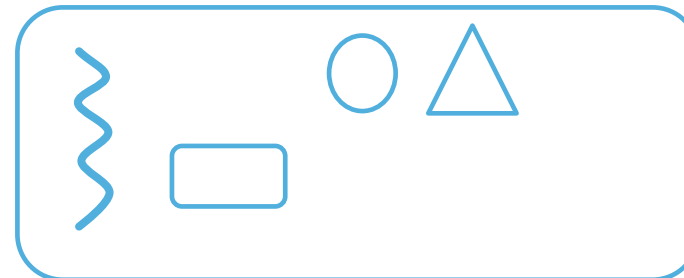
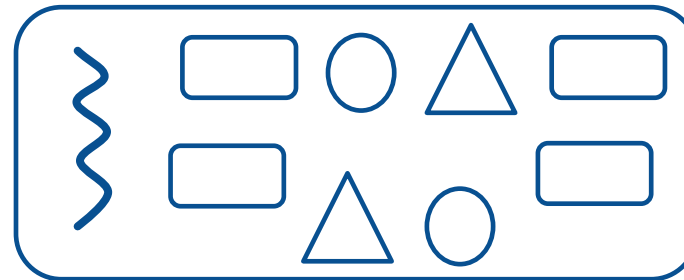


IFS weather code kernel. ECMWF

V. Marjanovic et al, “Overlapping Communication and Computation by using a Hybrid MPI/SMPs Approach” ICS 2010

K. Sala et al, "Improving the Interoperability between MPI and Task-Based Programming Models". Submitted

# PYCOMPSS



**Main program**

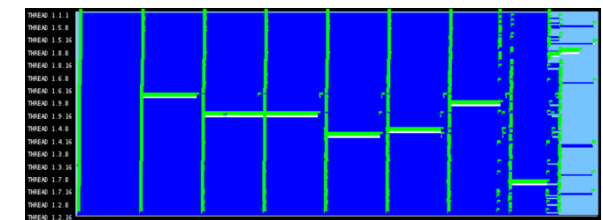
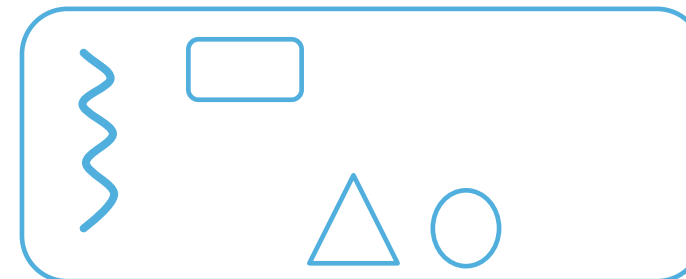
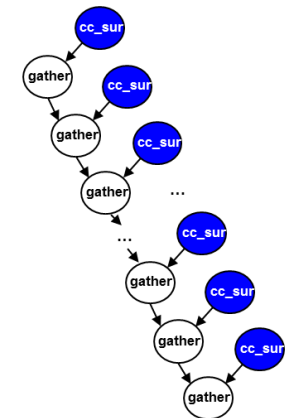
```
num_frags = ...
todo_list = init_list();
for i in range(num_frags):
    result = cc_sur(todo_list[i])
    gather(result, global_result)

compss_stop()
```

**Tasks definition**

```
@task(returns = list, result = IN, global_result = INOUT)
def gather(result, global_result):

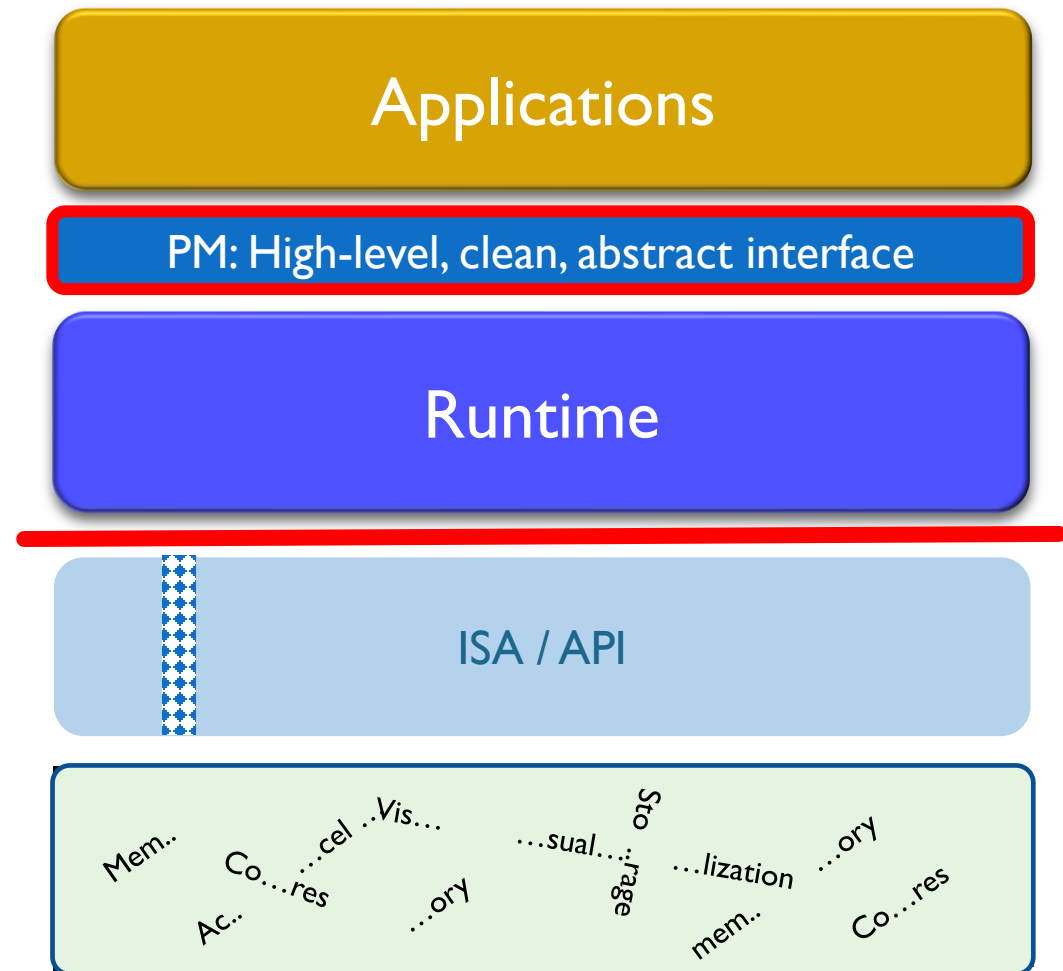
@task(returns = list)
def cc_sur():
```



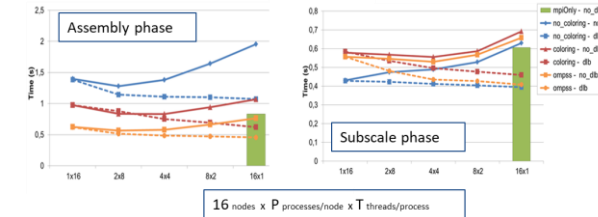
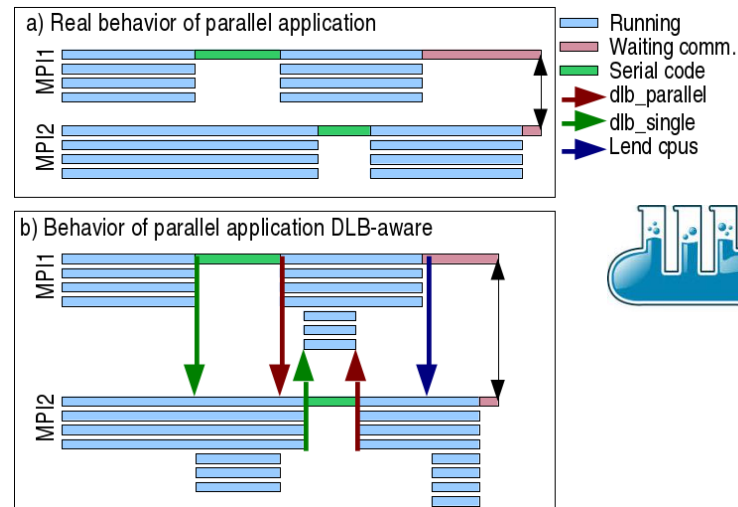
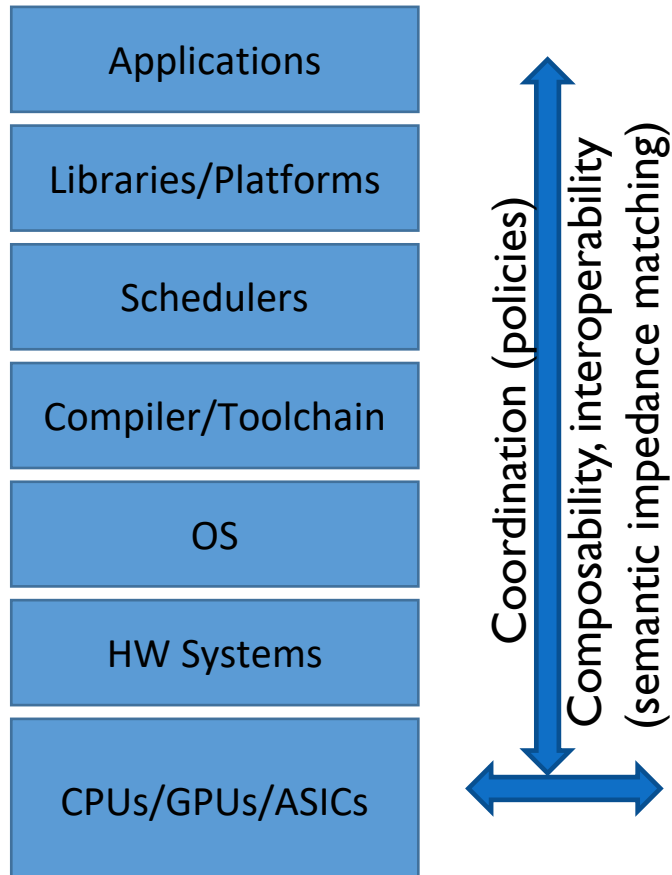
# LONG VECTORS



- 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.

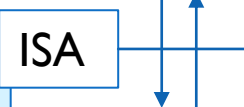


# MALLEABILITY & COORDINATED SCHEDULING



A wish:  
Handoff scheduling

**Vector Length Agnostic (VLA)**  
programming and architecture



saxpy:

```
vsetvli a4, a0, e32, m8
v1w.v v0, (a1)
sub a0, a0, a4
slli a4, a4, 2
add a1, a1, a4
v1w.v v8, (a2)
vfmacc.vf v8, fa0, v0
vsw.v v8, (a2)
add a2, a2, a4
bnez a0, saxpy
ret
```

# HOMOGENIZING HETEROGENEITY

Applications

Libraries/Platforms

Schedulers

Compiler/Toolchain

OS

HW Systems

CPUs/GPUs/ASICs

```
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];
}
```

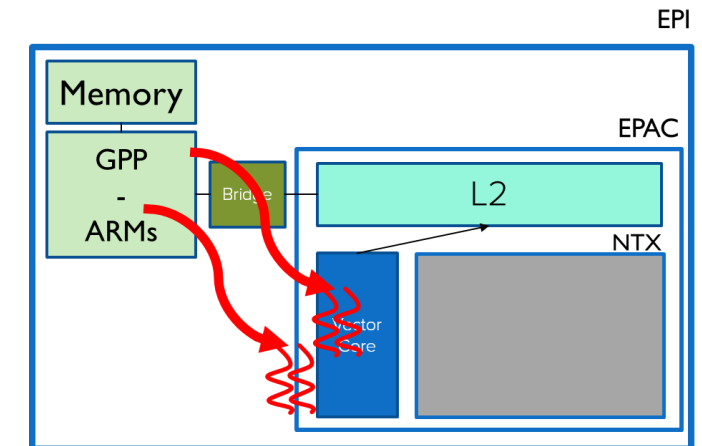
VLA helps homogenize Heterogeneous Performance

- ~ Big – Little cores, ....

Nested tasked/workshared

OpenMP

- Offload regular OpenMP
- Minimize CUDish influence

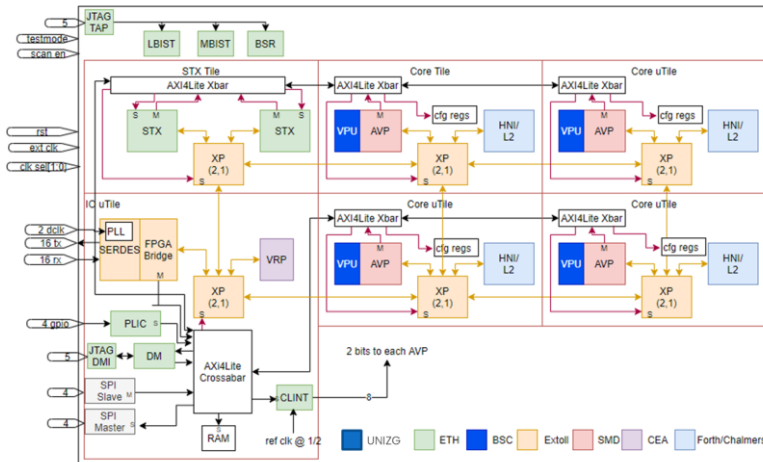


- HW support: IO coherence

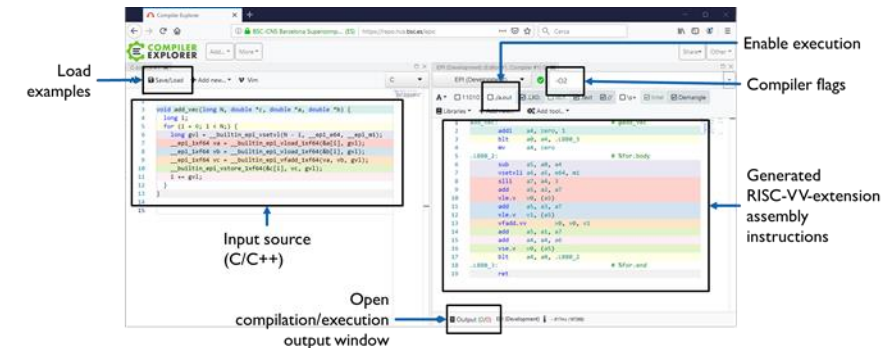


# WHERE WE ARE?

EPAC Test chip Tapeout @ 22nm  
@beginning of 2021



LLVM Vectorizing compiler  
& intrinsics



```
void axpy_SIMD
int i;
```

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

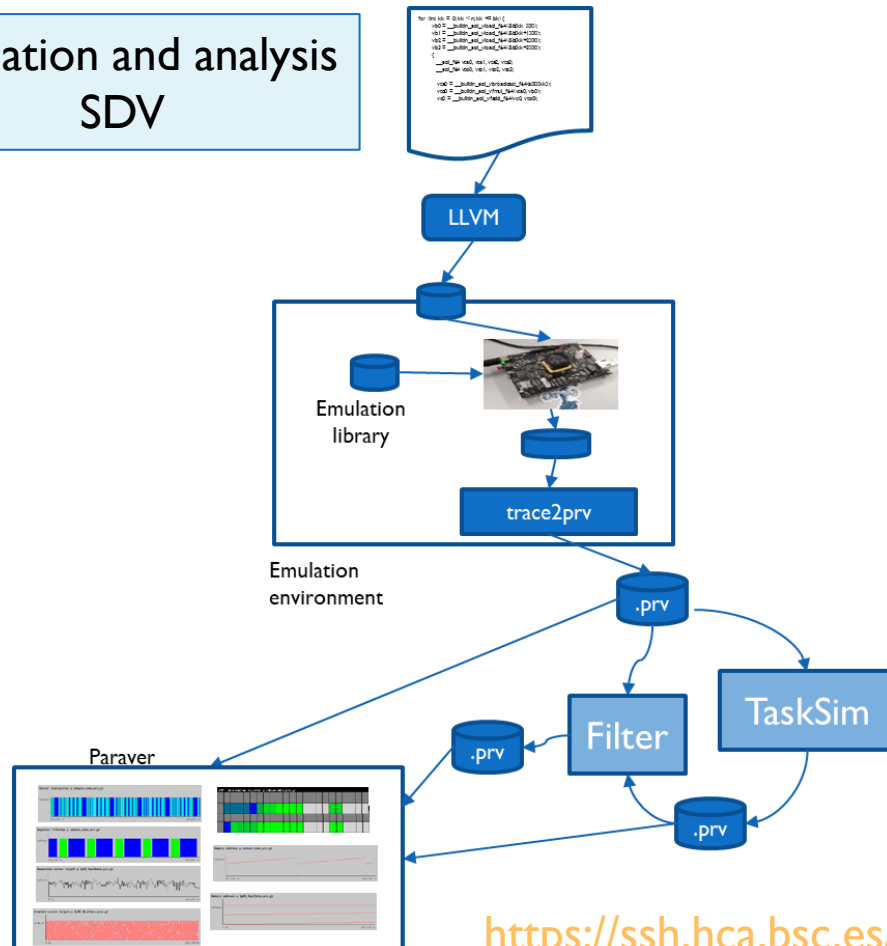
```
(double a, double *dx, double *dy, int n) {

void axpy_intrinsics (double a, double *dx, double *dy, int n) {
    int i;
    int gvl = __builtin_epi_vsetvl(n, __epi_e64, __epi_m1);
    __epi_1xf64 v_a = __builtin_epi_vbroadcast_1xf64(a, gvl);

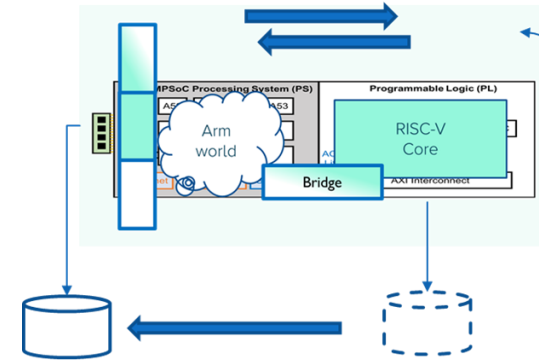
    for (i=0; i<n; ) {
        gvl = __builtin_epi_vsetvl(n - i, __epi_e64, __epi_m1);
        __epi_1xf64 v_dx = __builtin_epi_vload_1xf64(&dx[i], gvl);
        __epi_1xf64 v_dy = __builtin_epi_vload_1xf64(&dy[i], gvl);
        __epi_1xf64 v_res = __builtin_epi_vfmacc_1xf64(v_dy, v_a, v_dx, gvl);
        __builtin_epi_vstore_1xf64(&dy[i], v_res, gvl);
        i += gvl;
    }
}
```

# WHERE WE ARE ?

## Emulation and analysis SDV



## Heterogeneous System Software SDV



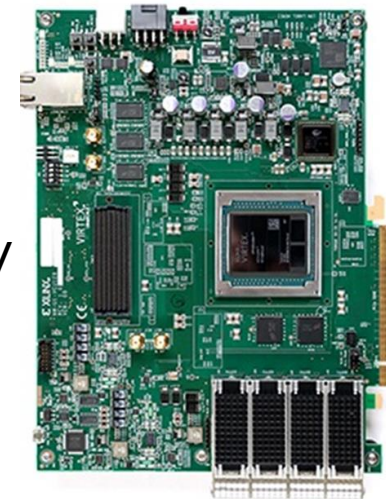
```
#on RISC-V side
$mkfs.ext4 -b 4096 /dev/vda
$mount -t ext4 /dev/vda /mnt/scratchfs
```

```
void main (...)
{
    ...
    gemTarget(A, B, C, size);
    ...
}

void gemTarget(double *A, double *B, double *C, long S) {
    #pragma omp target map(to:A[0:S*S],B[0:S*S],S) \
    map(from:C[0:S*S])
    {
        for(int i = 0; i < size; i++)
            for(int j = 0; j < size; j++)
                for(int k = 0; k < size; k++)
                    C[i*size + j] += A[i*size + k]*B[k*size + j];
        fp = fopen("/mnt/scratchfs/output_matrix.txt", "w+");
        for (int i=0; i<size*size; i++) fprintf(fp, "%lf ", C[i]);
    }
}
```

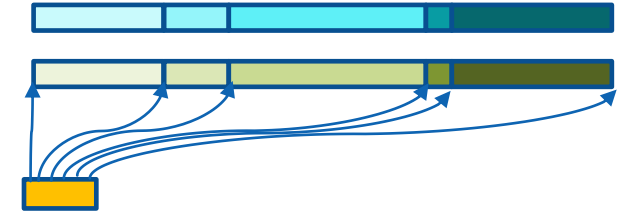
## EPAC SDV

RV64GCV  
Linux



# SPMV

```
void SpMV_ref(double *a, long *ia, long *ja,
              double *x, double *y, int nrows) {
    int row, idx;
    for (row=0; row<nrows; row++) {
        double sum = 0.0;
        for (idx=ia[row]; idx<ia[row+1]; idx++) {
            sum += a[idx] * x[ja[idx]];
        }
        y[row] = sum;
    }
}
```



# SPMV

Standard C/C++

Vector data types

Can be passed as arguments

Compiler does:  
Register allocation  
Instruction scheduling  
Spill/save/restore

```
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, __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_vsll_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);
    }
}
```

Intrinsic operations

... C variables

... vector variables

... vector length

VLA  
programming

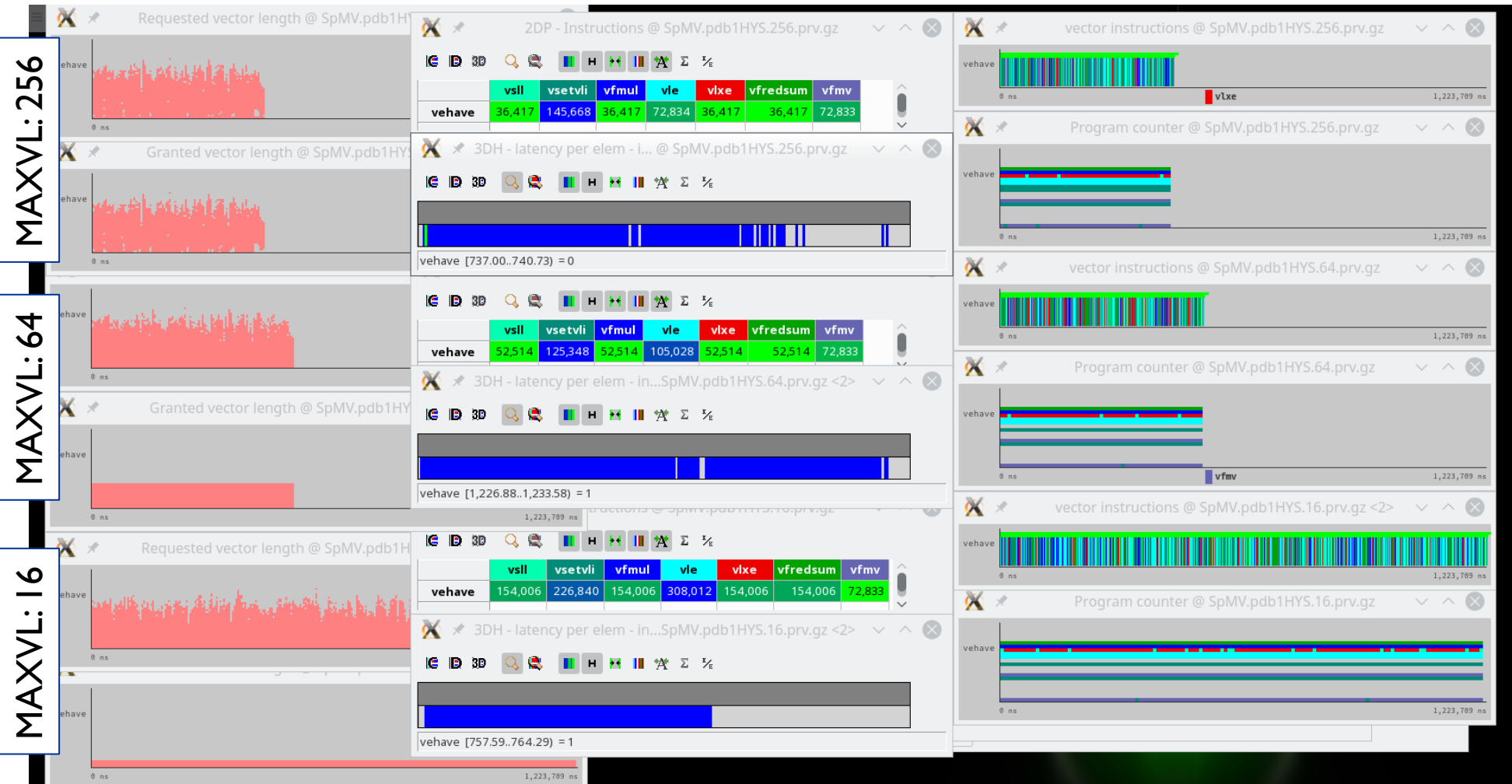
Sometimes “too” intelligent (eg. bcast; concept of cheap/expensive)

# SPMV

Matrix: pdb1HYS

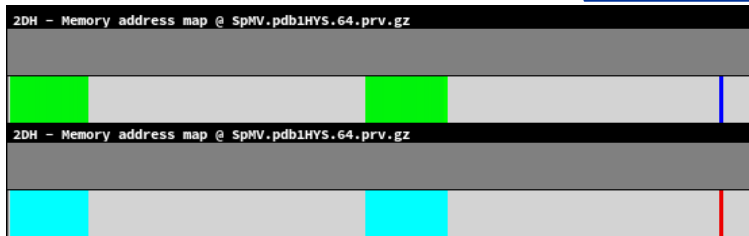
rvl: ~ 20 - 70 ?

vle: always miss  
vlse: fair hit

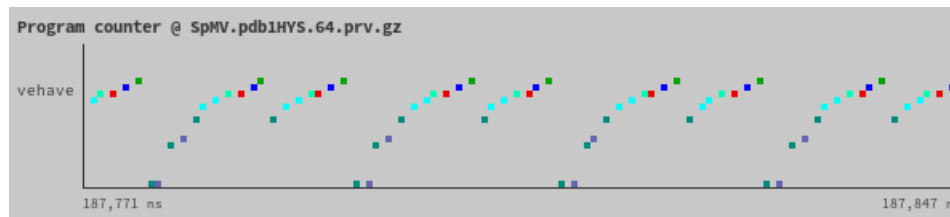


# SPMV

## Memory address map and timeline

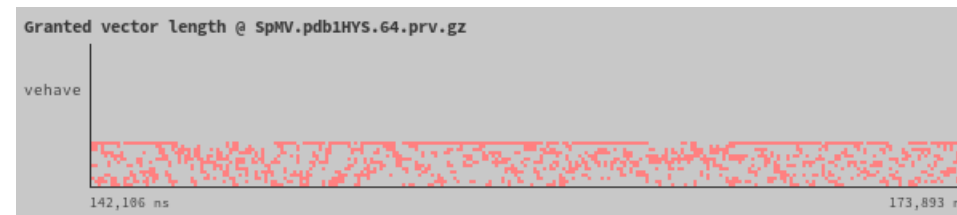
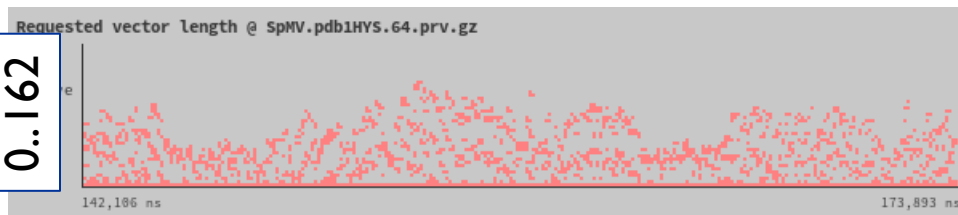


## PC and instruction



Identified some  
compilation issues

## Requested and granted vector lengths



Sometimes  
very small

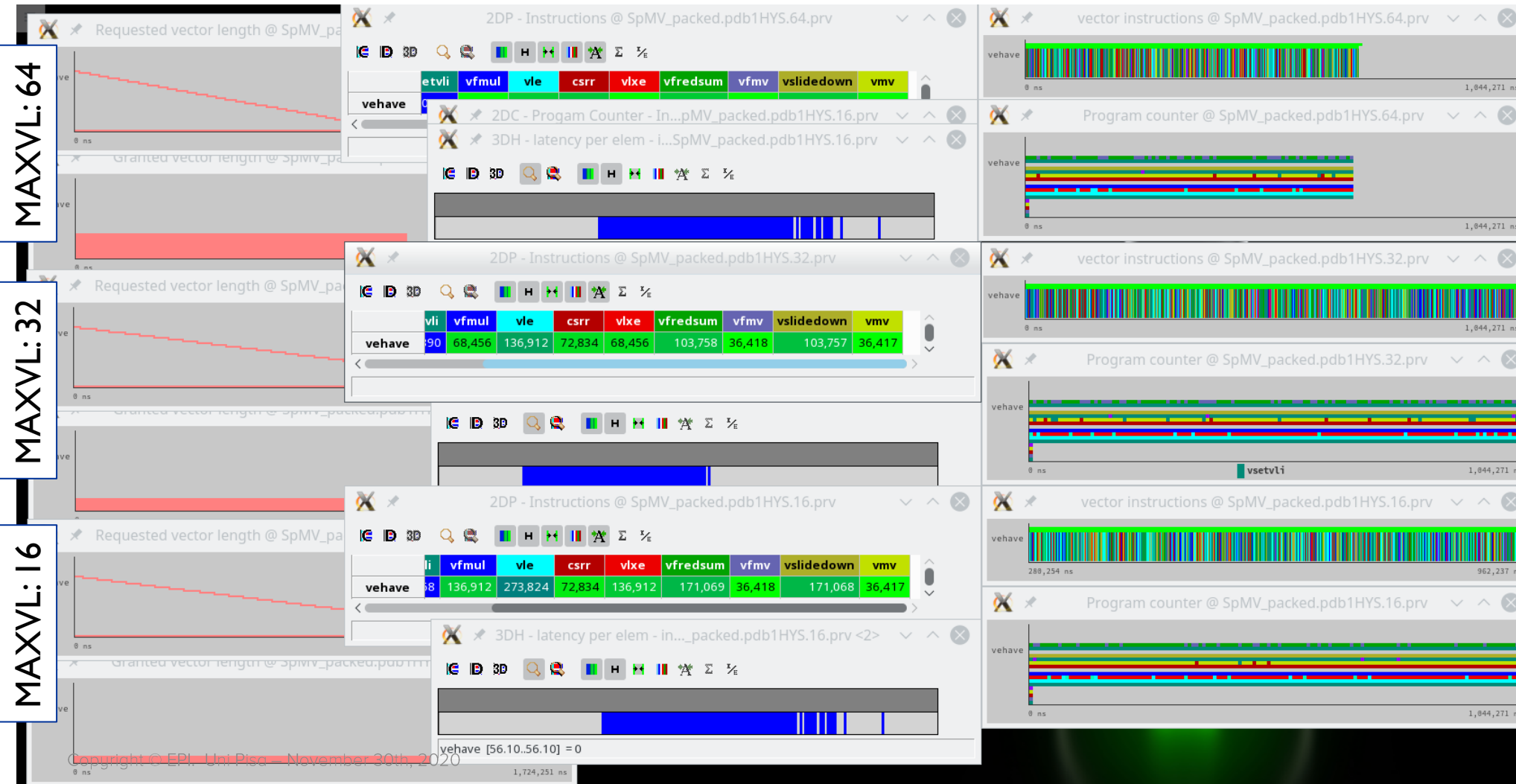
0..162

# SPMV\_PACKED

Matrix: pdb1HYS

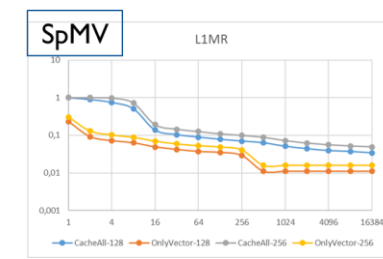
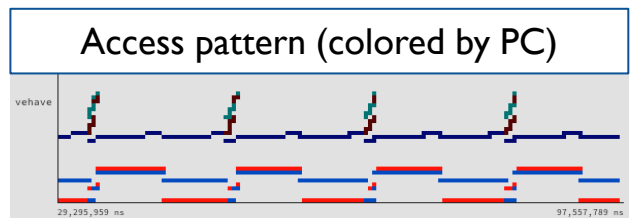
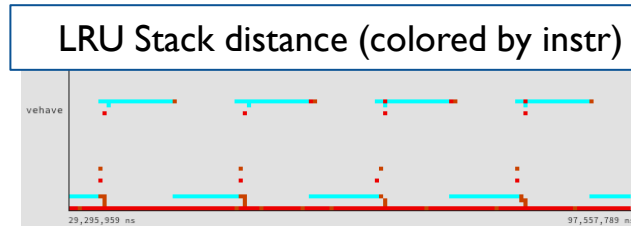
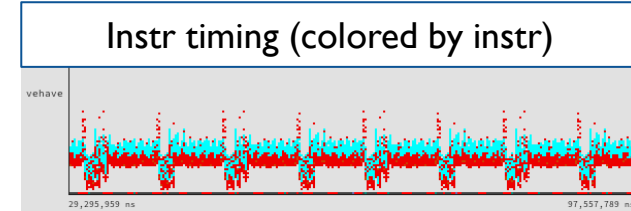
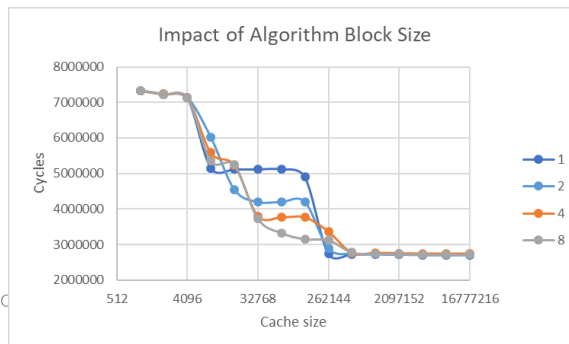
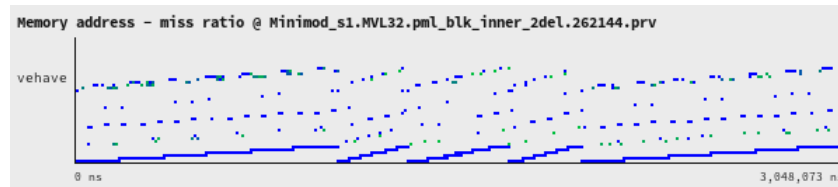
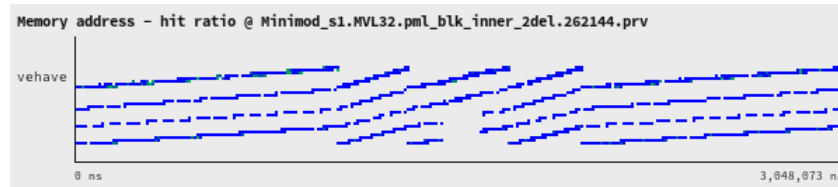
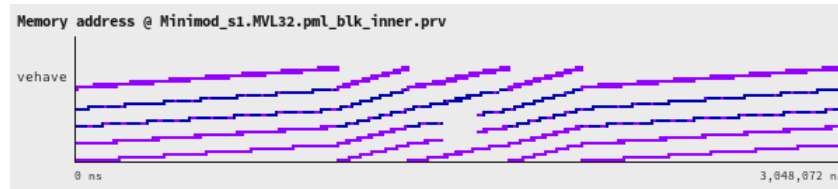
rvl: ~ 20 – 2M

Register mv !!



# MICROSCOPIC INSIGHT OF AGGREGATED BEHAVIORS

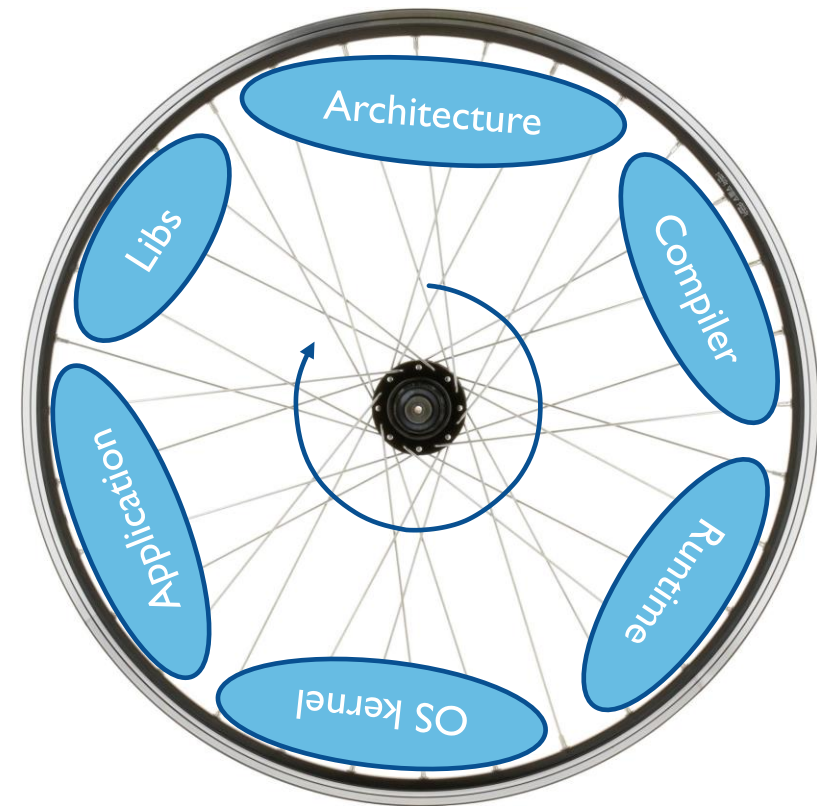
256K cache



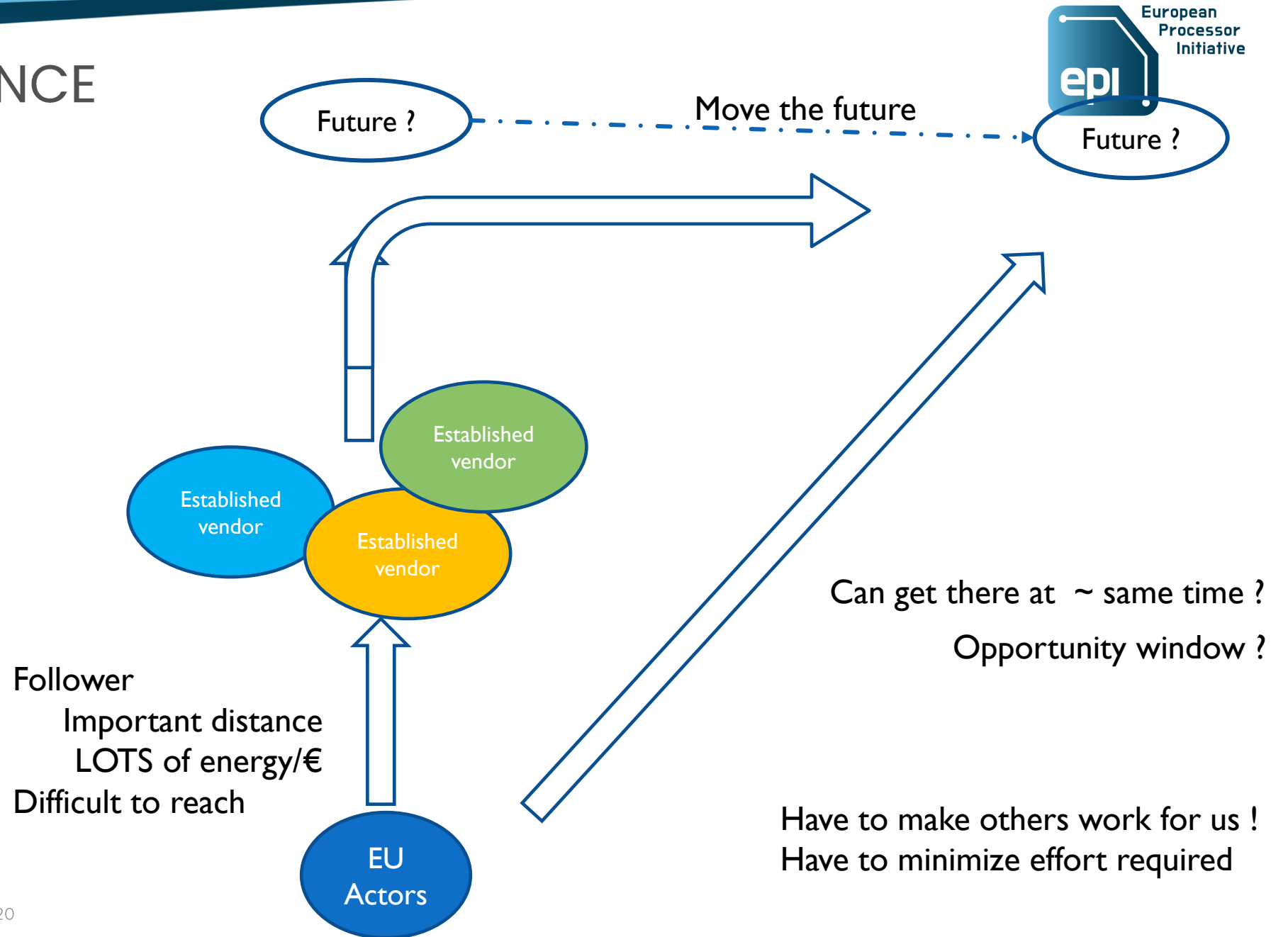


# MANY QUESTIONS

- Vector length?
- Register usage?
  - Efficient usage of state? How to improve?
  - Dependences?
- Address pattern
  - Reuse?
  - Stride?
- Instruction mix
  - Footprint
  - Scalar / vector interleave?
- Spill code
- ...



# THE IMPORTANCE OF A VISION



# EPAC

- 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
- Contact us if you are interested in evaluating the framework and provide co-design input

