Barcelona Supercomputing Center
Centro Nacional de Supercomputación

EXCELENCIA SEVERO OCHOA

The RISC-V “accelerator” in EPI

Prof. Jesús Labarta

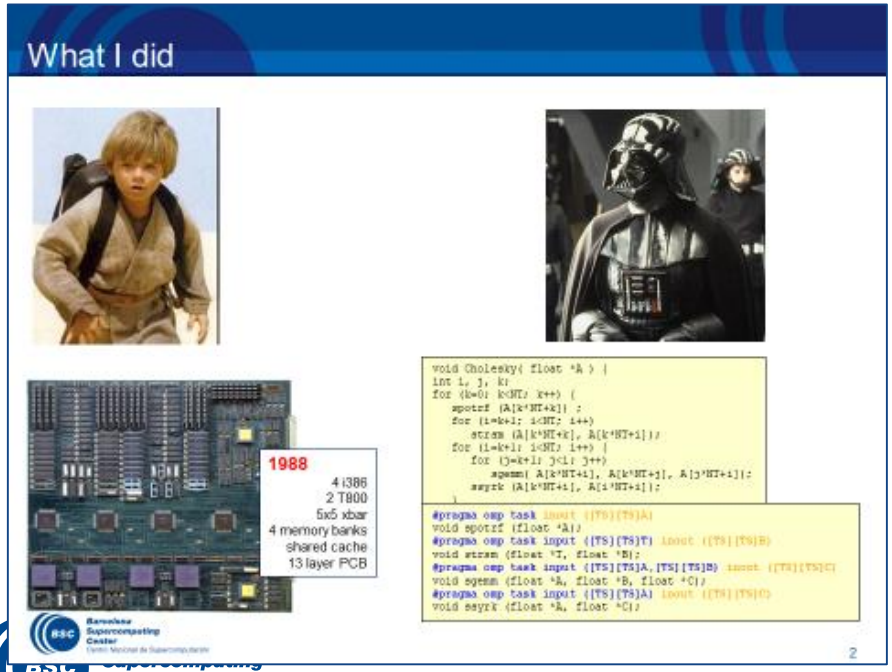
BSC & UPC

3B4HPC Summit

Barcelona, July 1st 2022

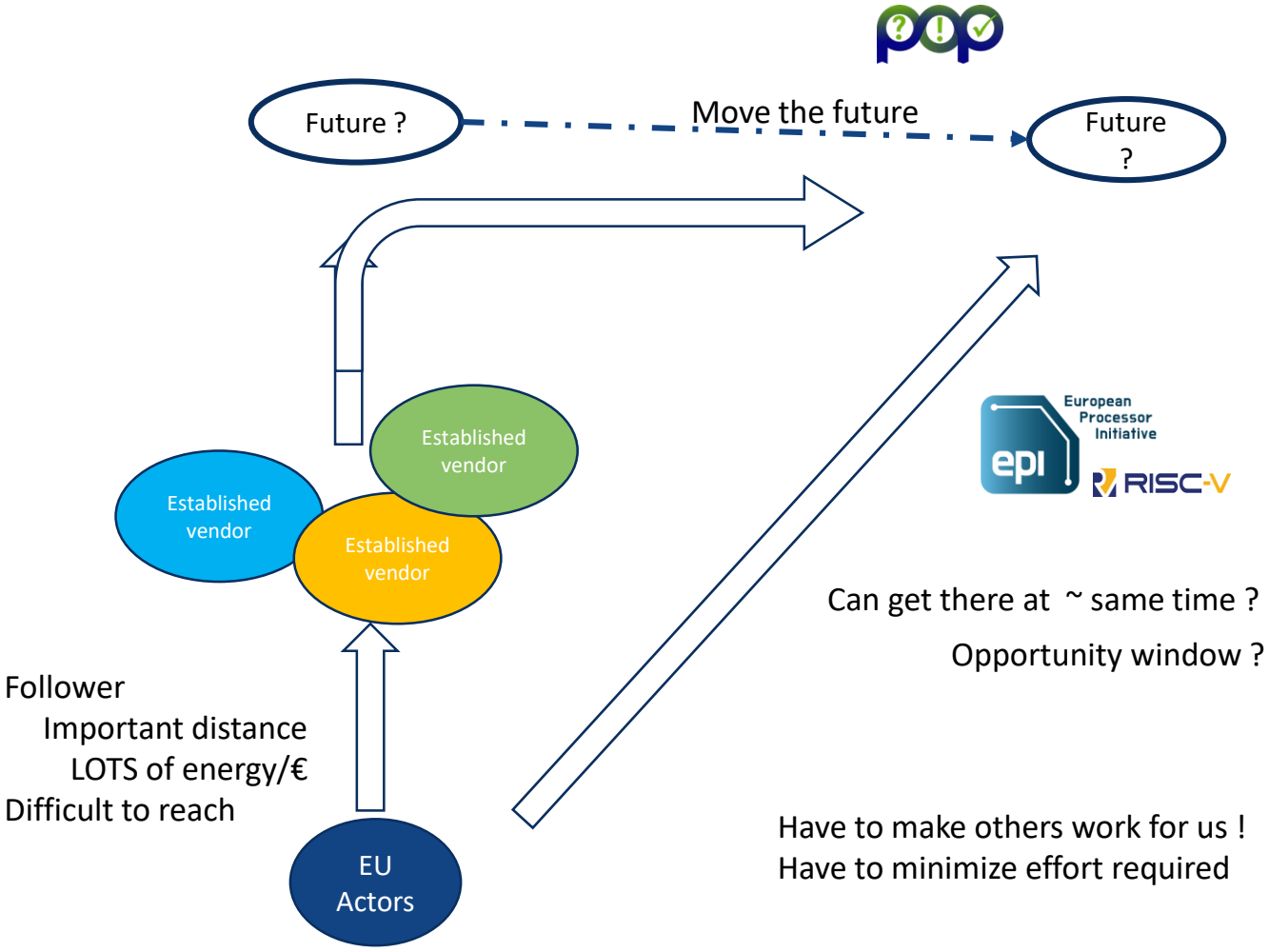
Disclaimer

- About myself



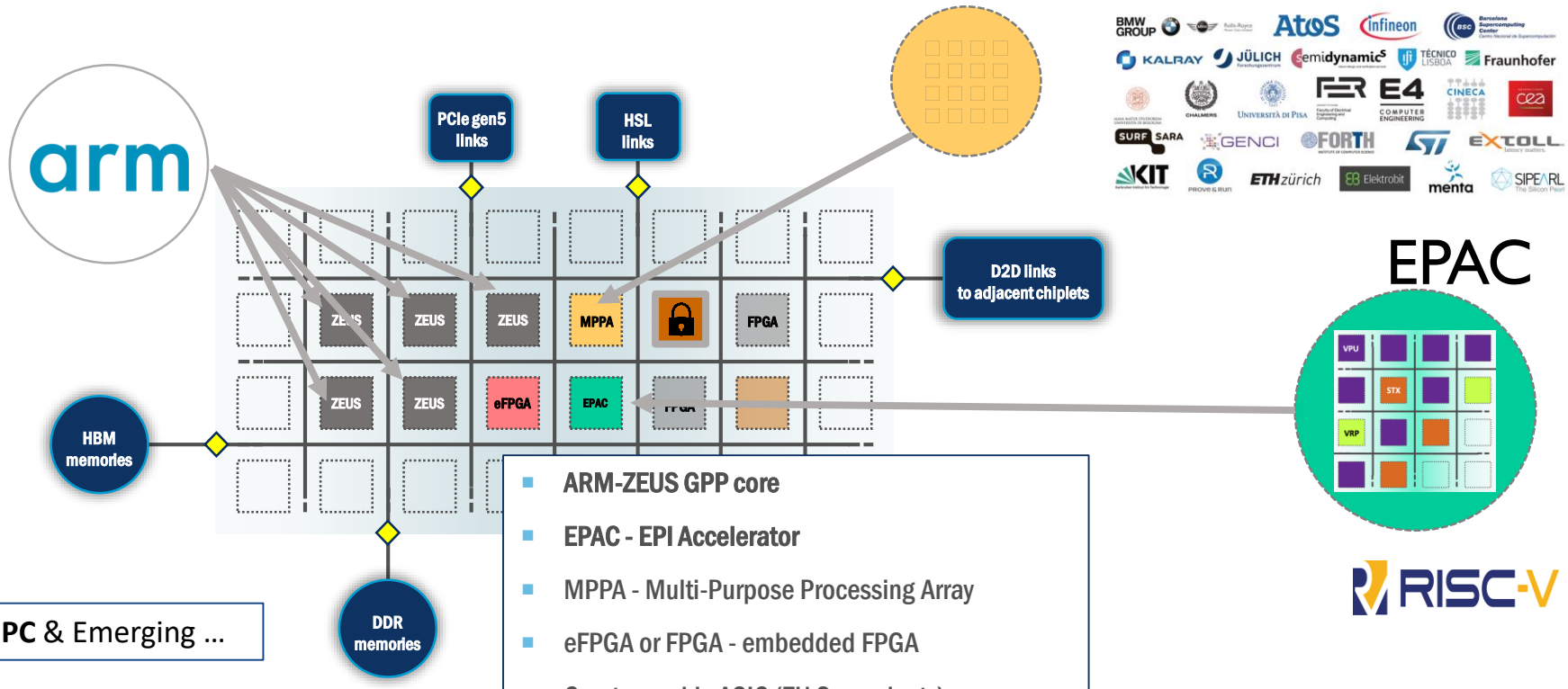
El abuelo cebolleta ataca de nuevo

The importance of a vision



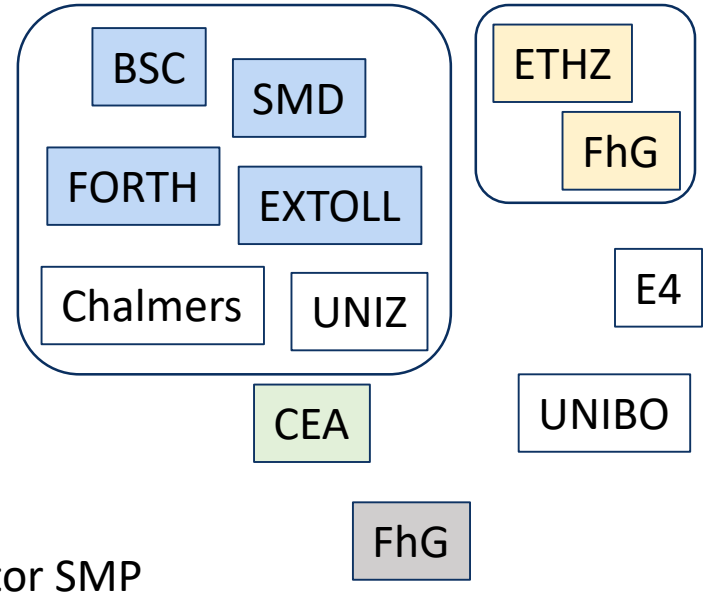
Thoughts from 2016

EPAC within EPI



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



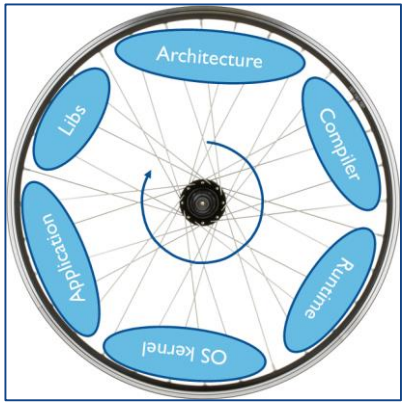
Holistic co-design

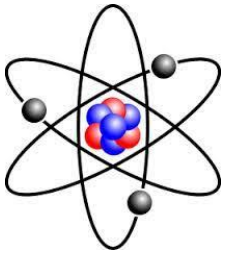
- Applications
- Libraries/Platforms
- Schedulers
- Compiler/Toolchain
- OS
- HW Systems
- CPUs/GPUs/ASICs

"As above, so below"

Similar concepts/mechanisms at all levels

"Steered by a vision/principles"
"Steered by detailed insight"



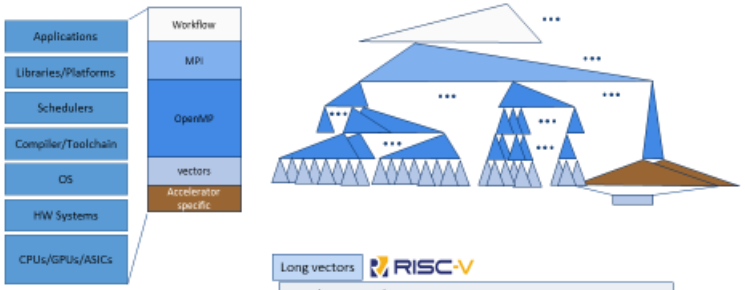


Vision

Balanced hierarchy

Expression & exploitation of Parallelism

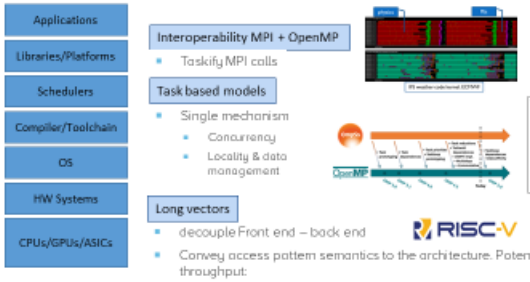
$$10^6 = 1 \times 10^6 = 10 \times 10^5 = 10^2 \times 10^2 \times 10^2$$



Long vectors RISC-V

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

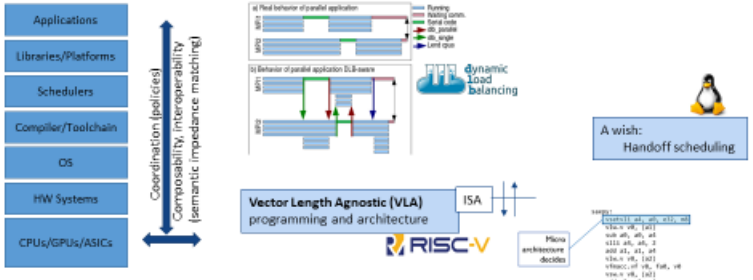
Task based models
- Single mechanism
  - Concurrency
  - Locality & data management

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

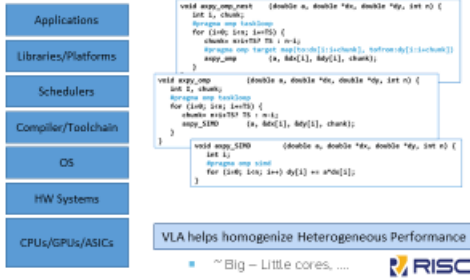
Task based computational workflows

RISC-V

Malleability & Coordinated scheduling



Homogenizing Heterogeneity

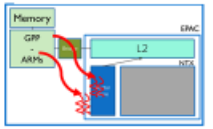


VLA helps homogenize Heterogeneous Performance
- ~Big – Little cores, ...

RISC-V

Nested tasked/workshared

OpenMP
- Offload regular OpenMP



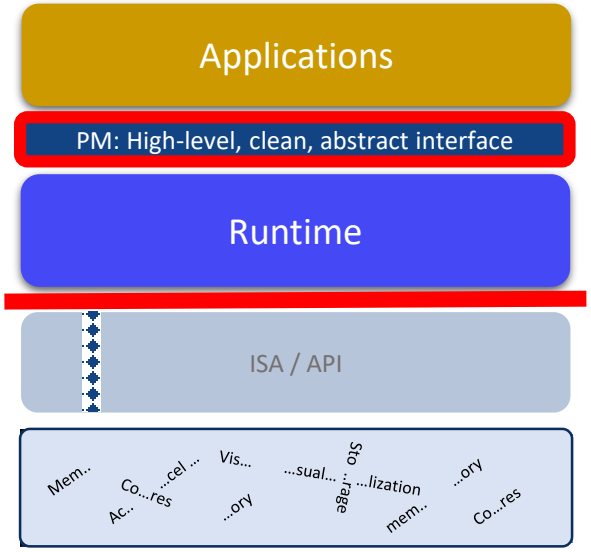
- HW support IO coherence

Long vectors

- Raise ISA semantic level
  - Vector instructions == tasks
  - “less words, more work”
  - The importance of ISA

RISC-V an enabler

- Parallelism: Asynchrony and overlap
  - Decouple Front end – back end
    - Less pressure, throughput orientation
  - OoO execution

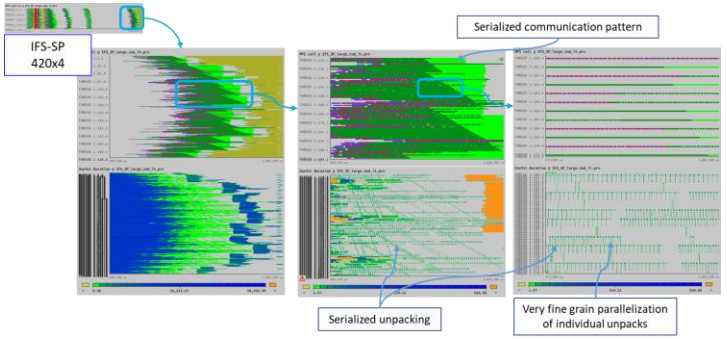


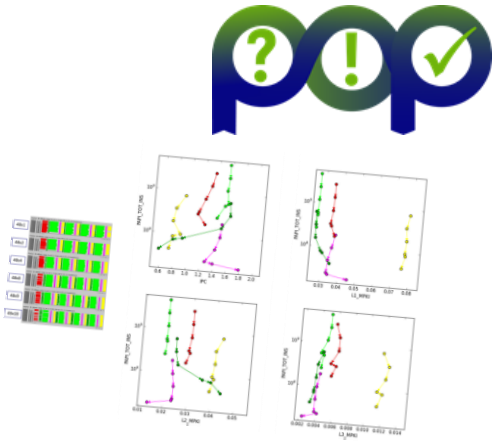
- Locality management
  - Hierarchy concept & hints

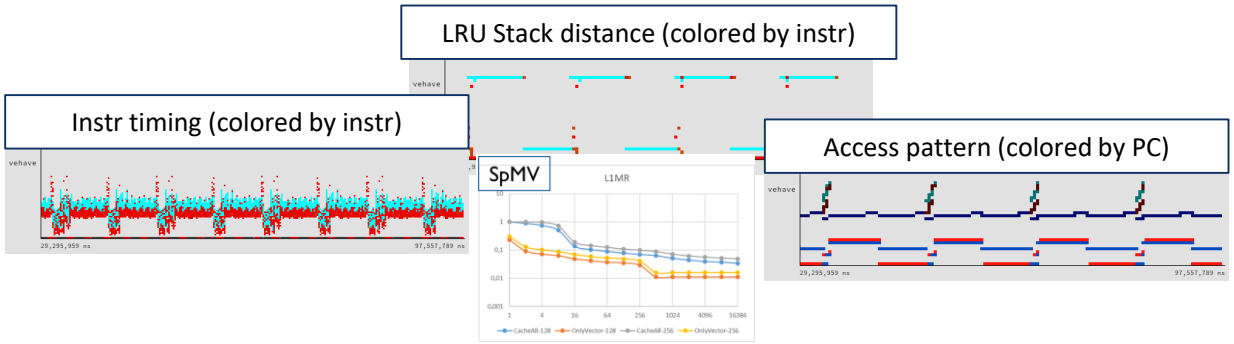
- Osmotic membrane
  - Convey access pattern semantics to the architecture.
  - Potential to optimize memory throughput

Detailed analysis and Insight on behavior

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



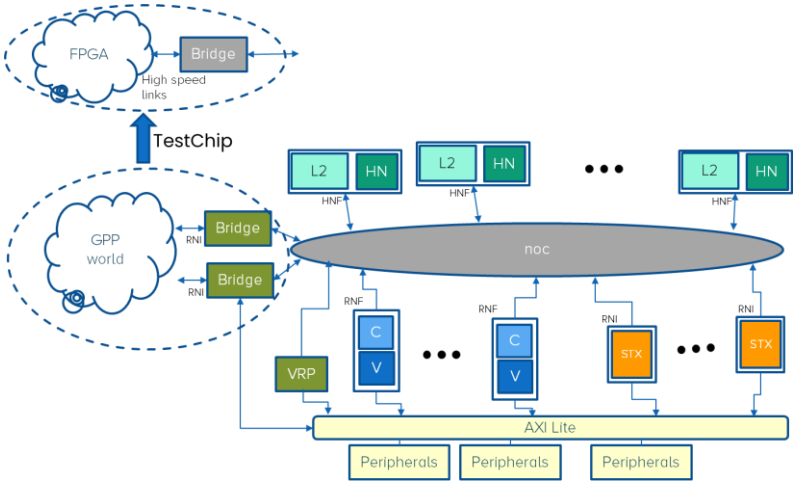




EPAC architecture

RVV

- RV64GCV (→ 8x)
  - 2 way in order core
  - Decoupled VPU
    - 8 lanes
    - Long vectors (256 DP elements)
  - → 128 MSHR
  - L1 - MESI coherency
- CHI interface NoC
  - 1 line / cycle
- L2$: 256KB/module
  - Allocation control mechanisms
- No in tile L3$



STX

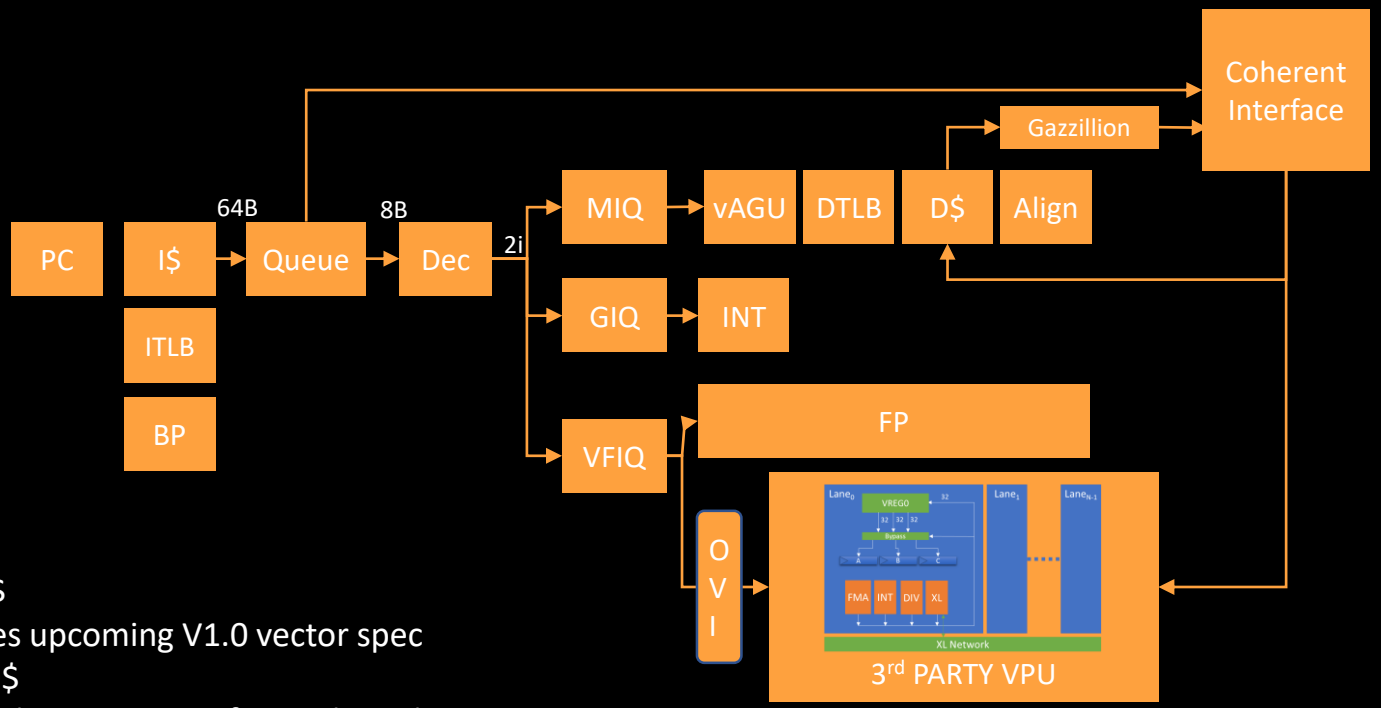
- DL and stencil specific accelerators
- Extensions to planned NTX
  - Programmable address generators →
  - lightweight RISC-v core + fat FPU + (Streaming Semantics & FREP)

VRP

- Variable precision processors

AVISPADO 220 with VPU

RISCV64GCV

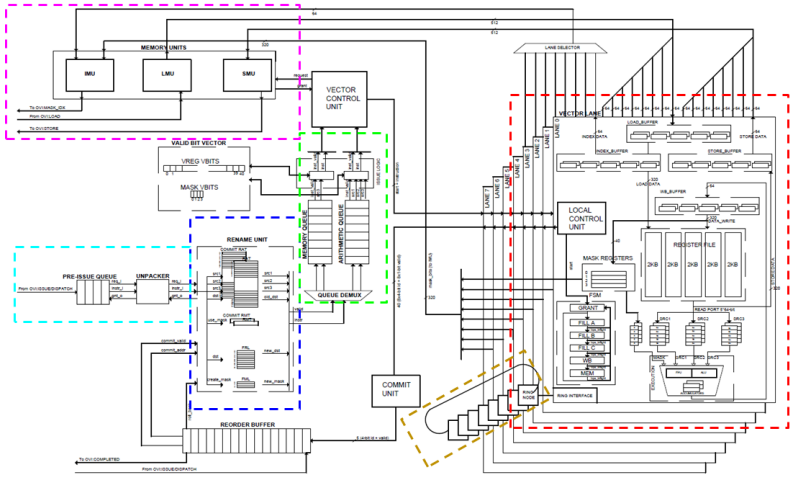


- SV48
- 16KB I$
- Decodes upcoming V1.0 vector spec
- 32KB D$
- Full hardware support for unaligned accesses
- Coherent (CHI)
- Vector Memory (vle, vlse, vlxe, vse, …) processed by MIQ/LSU

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

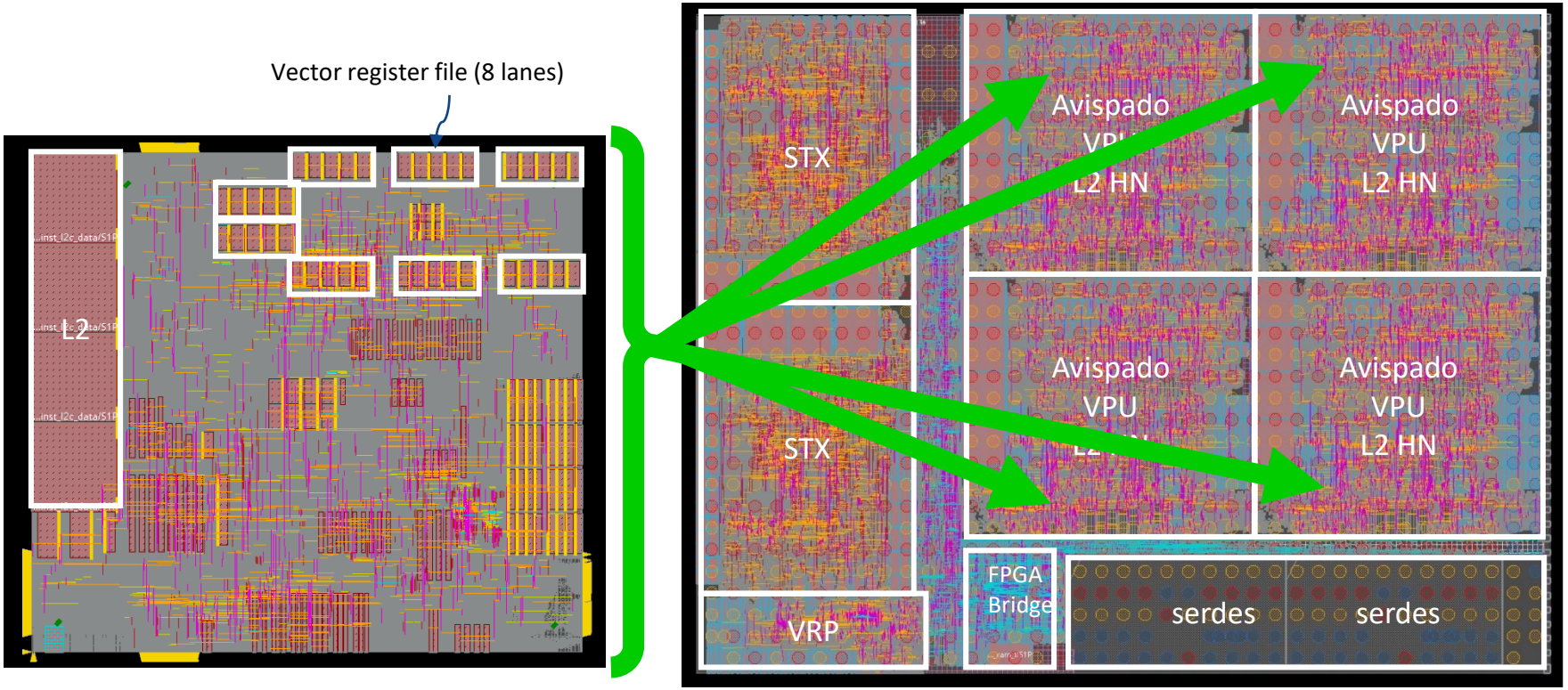


- Implementation
  - **#FUs << VL** (lanes=8, VL=256)
  - Some OoO
    - Resources to overlap?
      - L/S, FU, shuffling
    - Renaming
      - 40 physical registers
  - Single ported register file
    - Large state
    - 5 banks/lane providing sufficient bandwidth for 1 op/cycle (latency/BW trading)
  - Data shuffling: directional ring

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

Physical design

- 4 “VPU microtiles”
  - 2.517 mm2 each



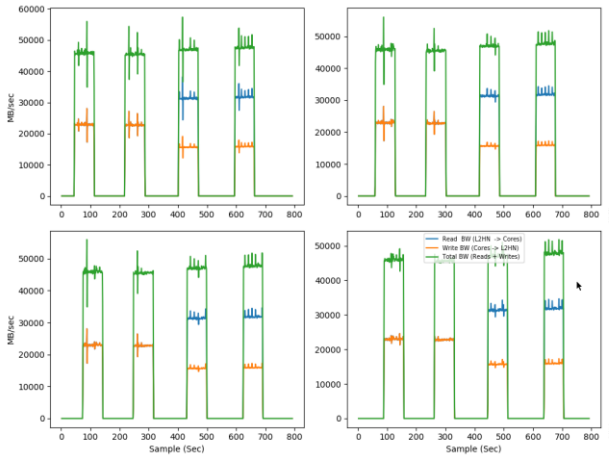
EPAC Test Chip

```
epac@EPAC:~/Desktop/etc_tools-master_v20211206/etc_tools-m
[sudo] password for epac:
Waiting for server ...
Waiting for server ...
Waiting for server ...
Waiting for server ...
Waiting for server ...
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$
```

```
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 FPUs
→ Long vectors !!
→ Memory Bandwidth

copy scale add triad



VRP

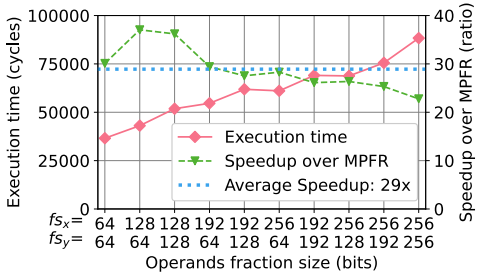


Table 1 Latency and energy per instruction

	Latency (cycles)		Energy (pJ/instruction)	
Instruction	Min	Max	Min	Max
VPADD/VPSUB	11	33	378.72	507.70
VPMUL	12	37	307.83	415.66
VPCMP	5	8	243.60	248.98
VPLOAD	16	42	604.99	1550.30
VPSTORE	20	30	1023.88	1795.26
FADD	6	-	117.76	-
FMUL	6	-	126.70	-

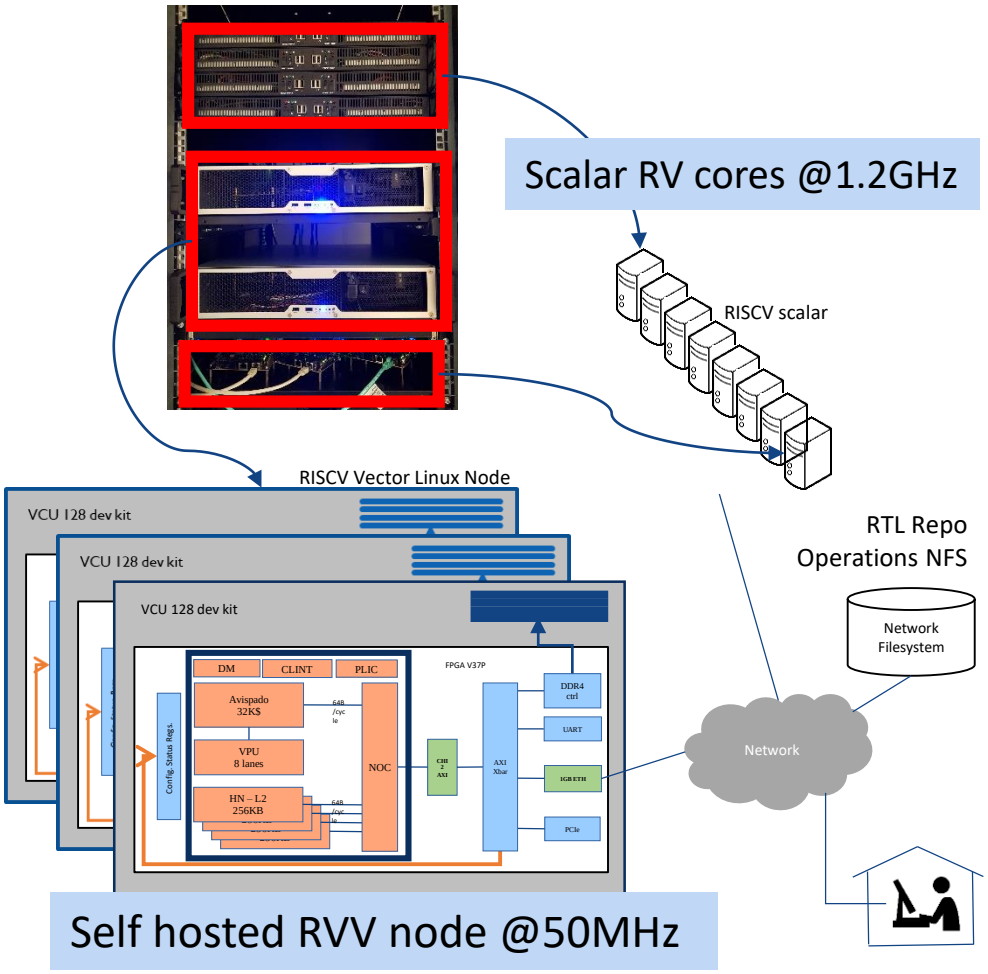
Min and max values use operands with significand sizes of 64b and 256b, respectively

RVV @ FPGA & ecosystem

- HPC software stack @ Commercially available RISC-V platforms
  - SLURM, MPI, OpenMP, BSC tools, RVV software emulator
- EPI SDV platforms
  - Linux
  - Test user codes @ real RTL
  - Give to EPI partners and external users early access to EPI technology

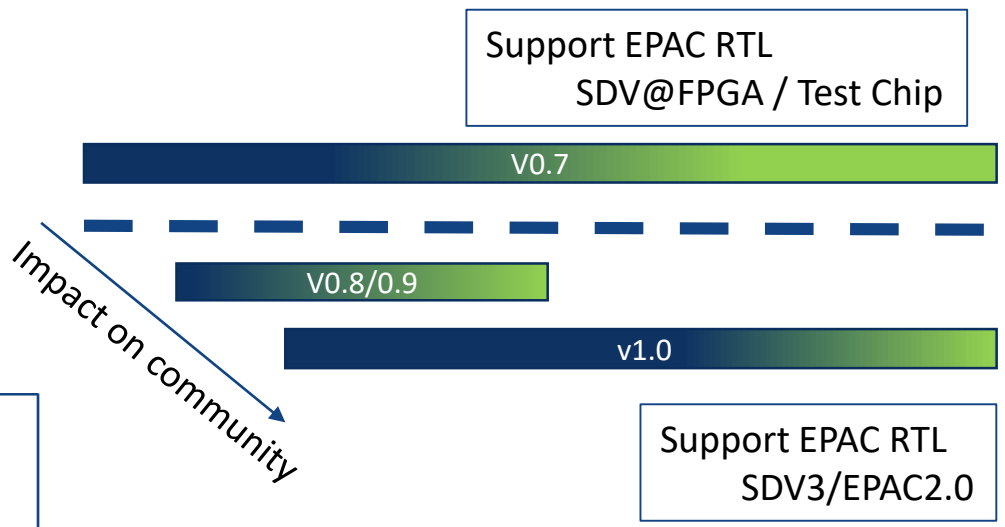
Contact : filippo.mantovani@bsc.es

- Holistic CI/CD framework
  - HW & SW
  - Functionality & performance



RISC-V Vector extension (RVV) Compiler

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

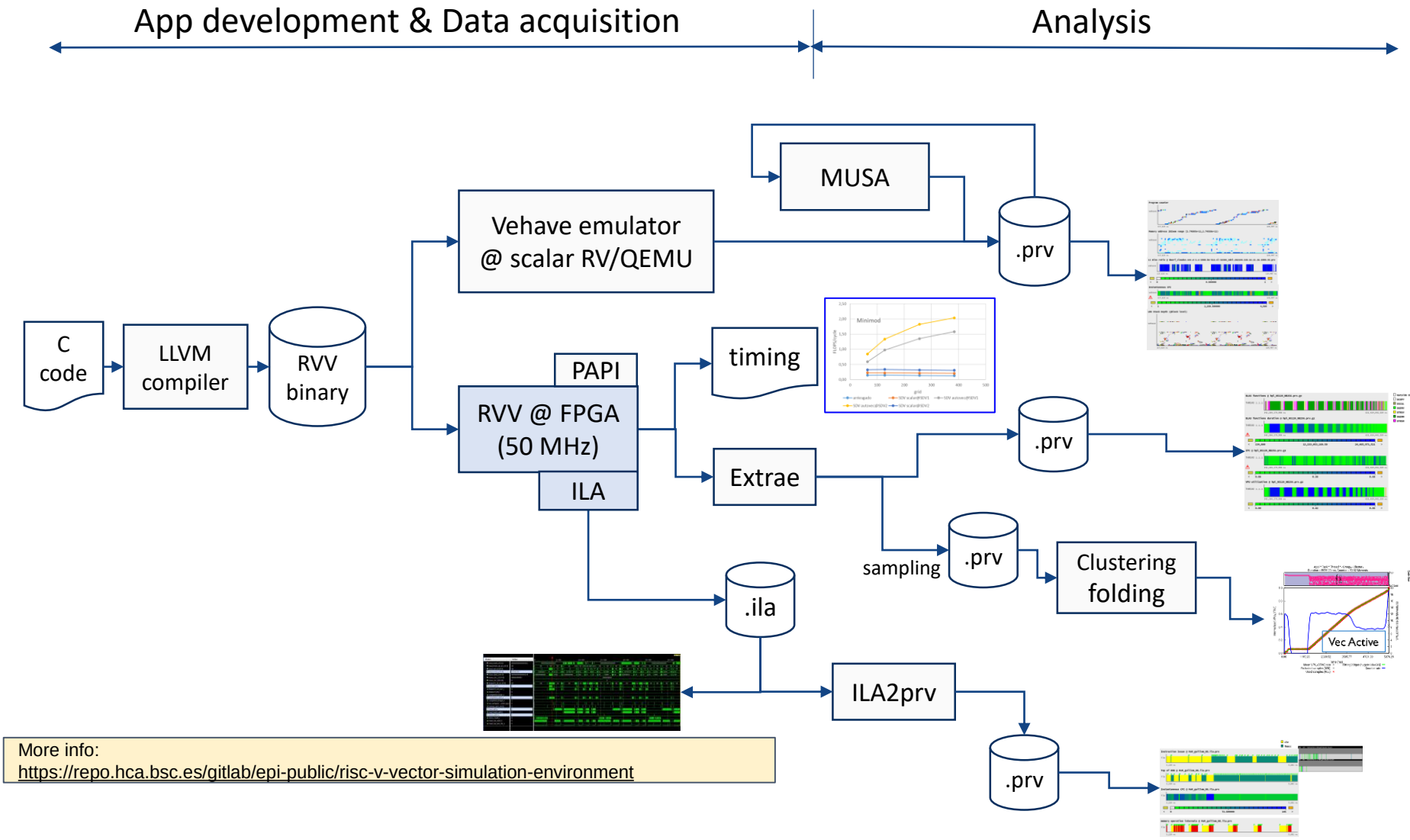



```
void SpMV_vec(double *a, long *ia, long *ja, double *x, double *y, int nrows) {
    for (int row = 0; row < nrows; row++) {
        int nnz_row = ia[row + 1] - ia[row];
        unsigned long gvl;        // granted vector lengths
        int idx = ia[row];
        __epi_1xf64 v_a, v_x, v_prod, v_partial_res;
        __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 = _
gvl);
            colid  += gvl;
        }
        y[row] += __builtin_
    }
}
```

```
void target_inner_3d (…) {
  for (i) {
    for (j) {
      #pragma clang loop vectorize(enable)
      for (k) {
        lap = LAP;
        v[IDX3_l(i,j,k)] = 2.f*u[IDX3_l(i,j,k)]
                 -v[IDX3_l(i,j,k)]+vp[IDX3(i,j,k)]*lap;
      }}}}
```

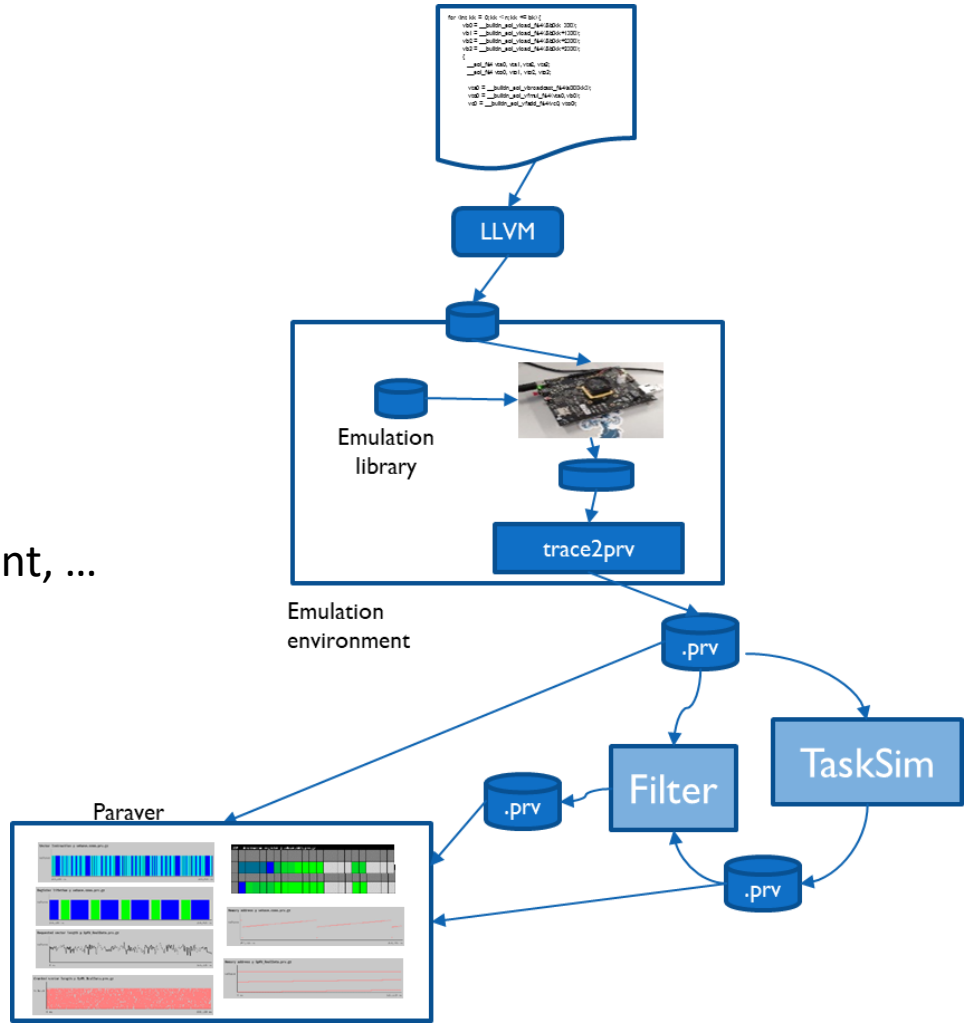
```
…
float complex A[n][n];
float complex templ, temp2;
…
for (int j = 0; j < n; j++)    {
   if (x[j] != ZERO || y[j] != ZERO)  {
      templ = alpha * conjunction(creal(y[j]), cimag(y[j]));
      temp2 = conjunction(creal(alpha * x[j]), cimag(alpha * x[j]));
      #pragma clang loop vectorize(assume_safety)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

SDV flows


App development & Data acquisition
Analysis
C code
LLVM compiler
RVV binary
Vehave emulator @ scalar RV/QEMU
MUSA
.prv
PAPI
RVV @ FPGA (50 MHz)
ILA
timing
Extrae
.prv
sampling
.prv
Clustering folding
Vec Active
.ila
ILA2prv
.prv
More info:
https://repo.hca.bsc.es/gitlab/epi-public/risc-v-vector-simulation-environment

RVV Software emulation

- Functional
  - Full OS, vectorising compiler, …
  - On QEMU & scalar RISC-V cores
- Timing model
  - Microarchitectural parameters
    - MAXVL, OoO, Locality management, …

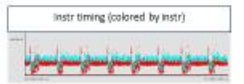


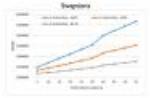
- https://repo.hca.bsc.es/gitlab/epi-public/risc-v-vector-simulation-environment

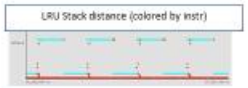
RVV Software emulation

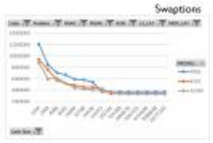
RVV Software emulation

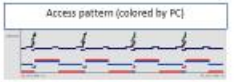
- Detailed analysis & insight

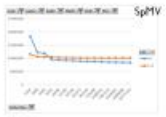












RVV Software emulation

- Yolo: AI

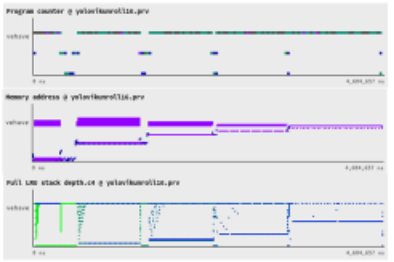
Program counter

Instruction? Order? Footprint?
Dependences? ROB size?

Memory address

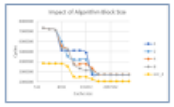
Cache management hints

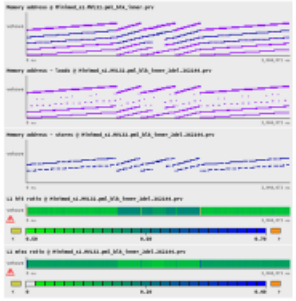
LRU stack depth



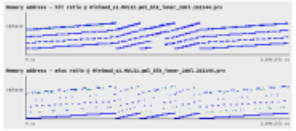
RVV Software emulation

- Detailed analysis & insight





256K cache

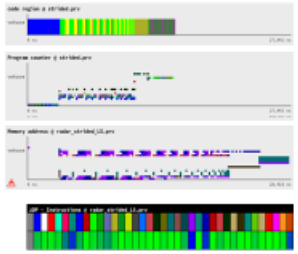


RVV Software emulation

- Radar miniapp

- Directives + automatic vectorization
  - Incremental way

- Steering compiler optimizations
  - Complex data types
  - Indexed → strided
  - Reuse through registers
  - Avoid optimizations generating extremely short vector lengths

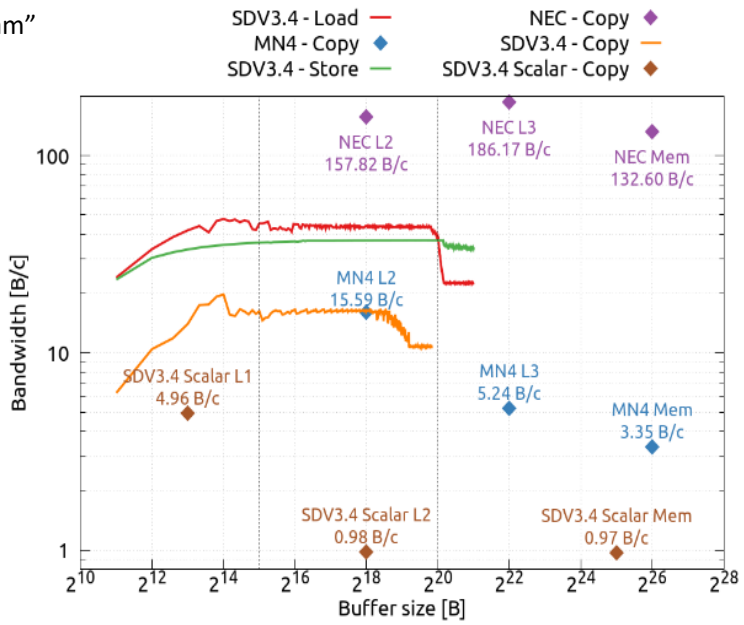


- Steering code refactoring
  - With productivity in mind !
    - Interchange, collapse, ...

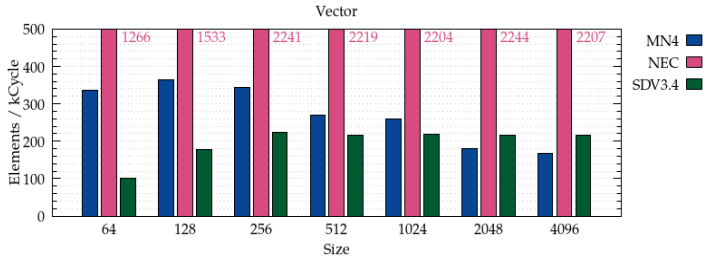
SDV@FPGA – vector performance

- EPAC – RVV vs. state of the art

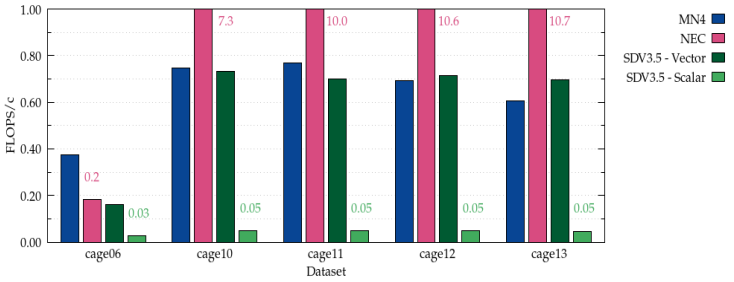
"Stream"



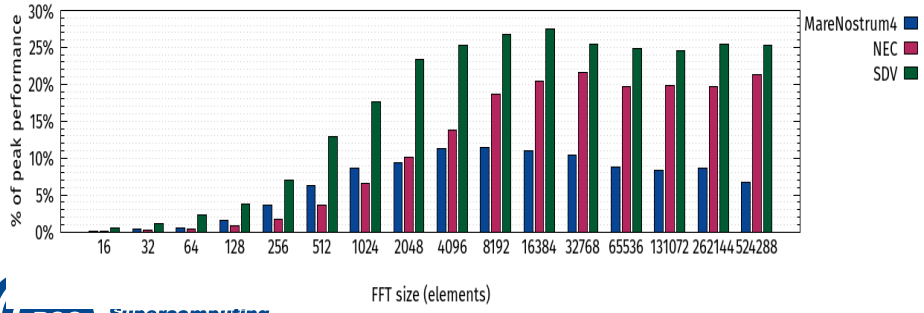
Jacobi 2D



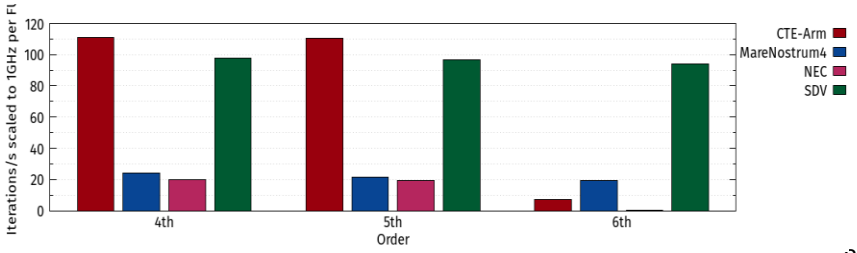
SpMV



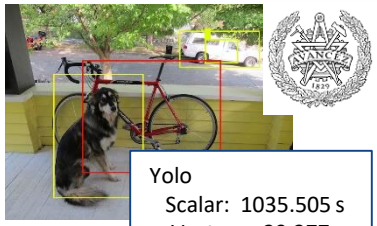
FFT



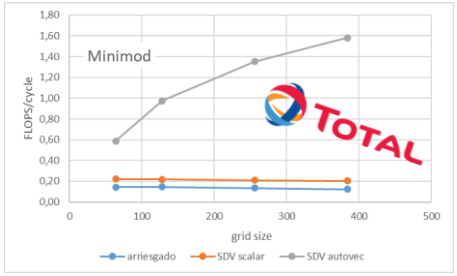
HACC

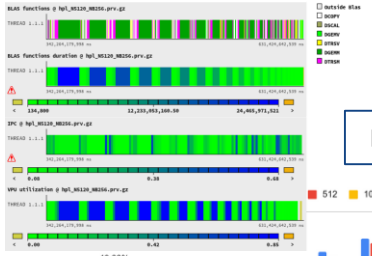


SDV – vector in HPC and beyond

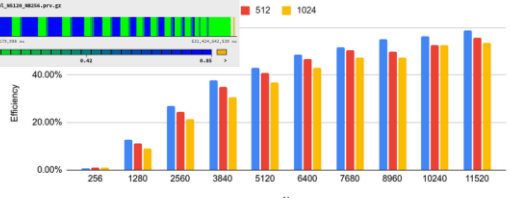


Yolo
Scalar: 1035.505 s
Vector: 29.377 s

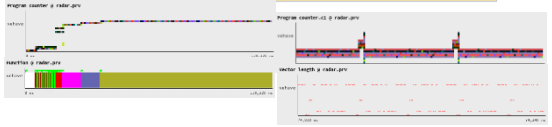




Linpack



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



@vehave. WIP: backporting vectorization to 0.7

- GROMACS
- EC-EARTH
- oneDNN

- BLIS

- Ginkgo

- Climate dwarfs
- TFLite
- Containers
- PyCOMPSs
- Spark

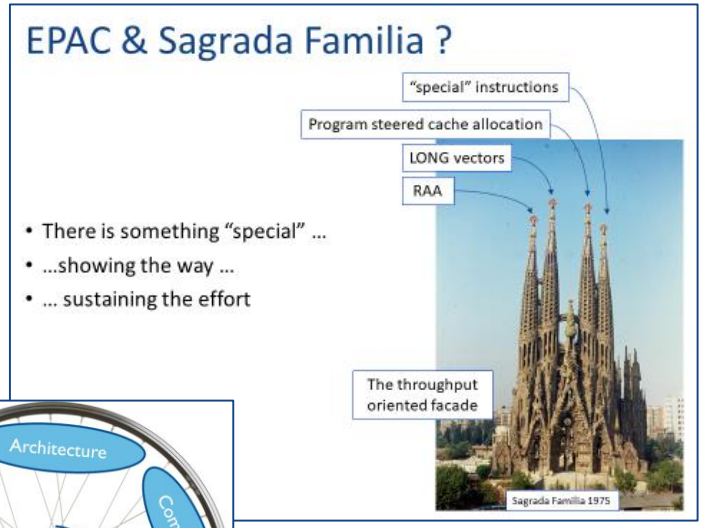
- Pytorch
- PostgressSQL
- Sorting

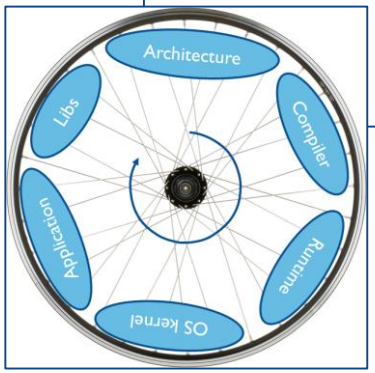
EPI SDV roadmap

When	Today	June 2022	Dec. 2022	Q1 2023	Q1 2024
HW available	RISC-V scalar platform x 8 Vehave: Simulator of vec. Instr.	3 FPGA Single core + VEC acc.	X FPGA single core + VEC acc. as a multinode	X FPGA VEC acc. multicore	EPAC chip 1.5 Multicore and multinode	EPAC chip 2.0 Multicore and multinode
What can we do with it?	- Validate code on RISC-V - Test compiler infrastructure - Analyze vectorizing potential and optimum vector length	- Test VEC acc. architecture	- Test MPI applications on VEC acc.	- Test OpenMP applications on VEC acc.	- Test applications on EPAC 1.5	- Test applications on EPAC 2.0

The importance of a vision

- 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





RISC-V

European Processor Initiative

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

EXCELENCIA SEVERO OCHOA

Thanks