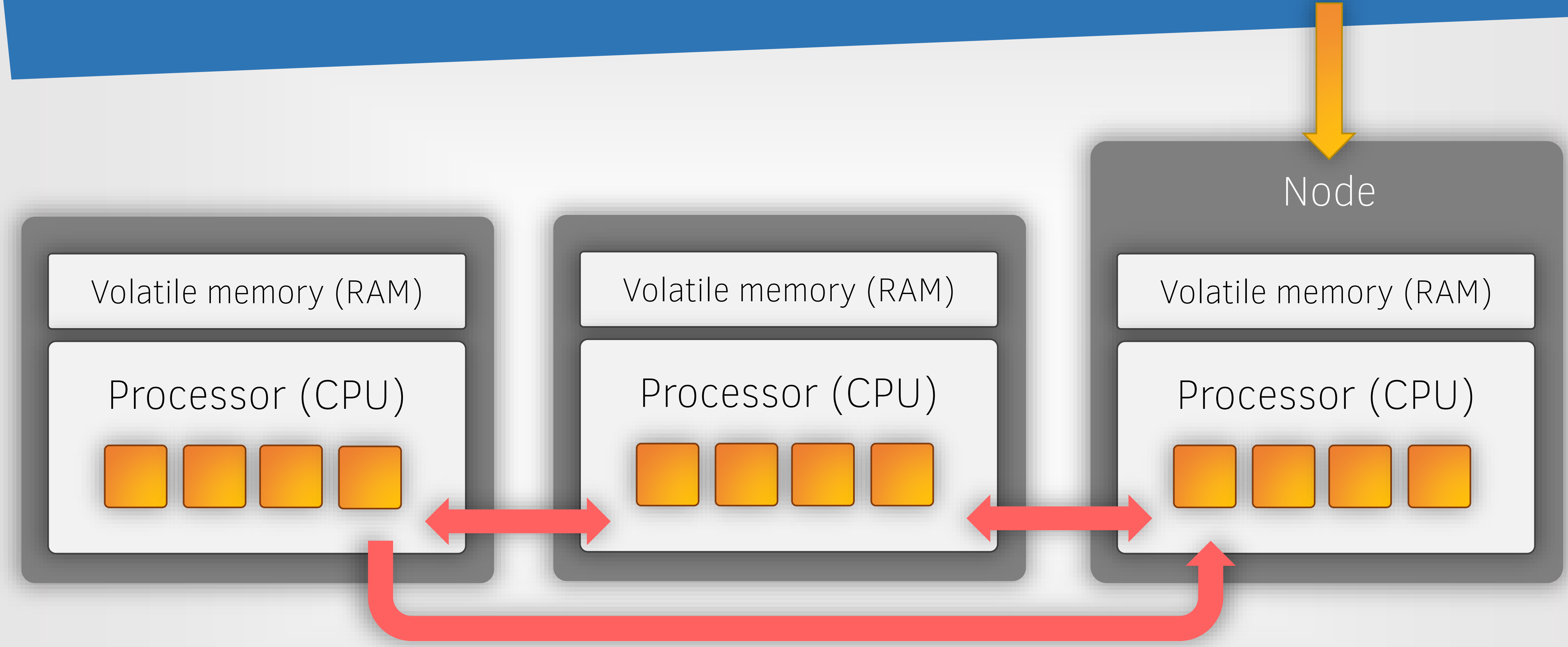
Smilei) Workshop

Node-level optimization (20 min)

Ecole Polytechnique – March 2022
Mathieu Lobet

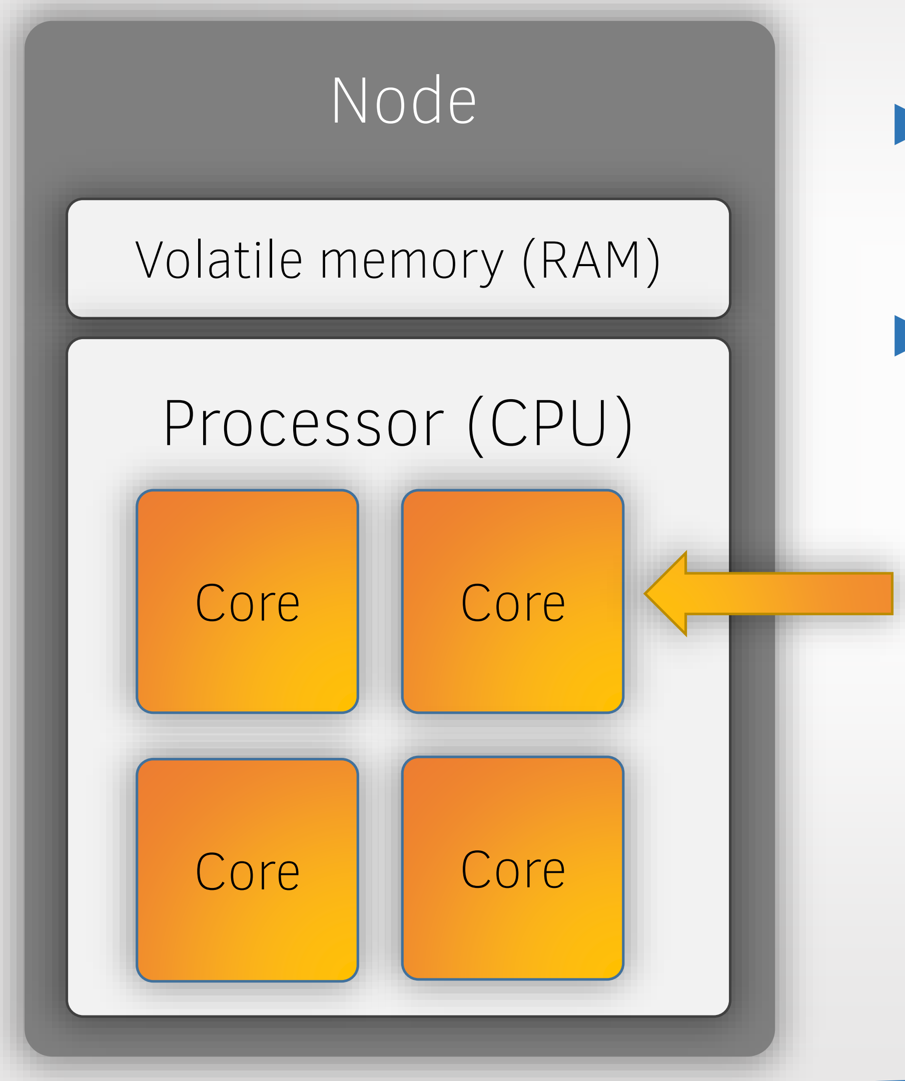
Focus on the node parallelism level



Smilei) Workshop

I. Vectorization

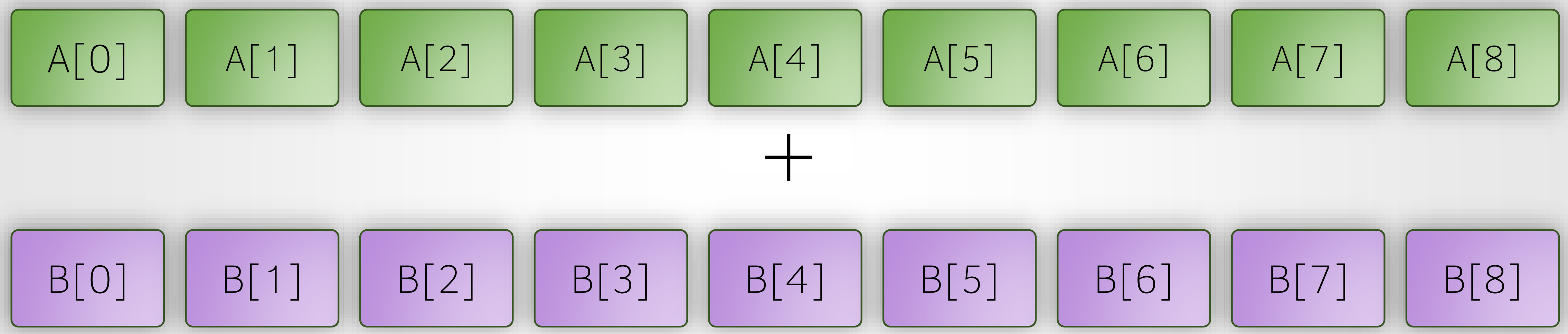
Vectorization is a component of each core



- Vectorization is a parallel computation located at the core level.
- It is refered to as SIMD for Single Instruction Multiple Data

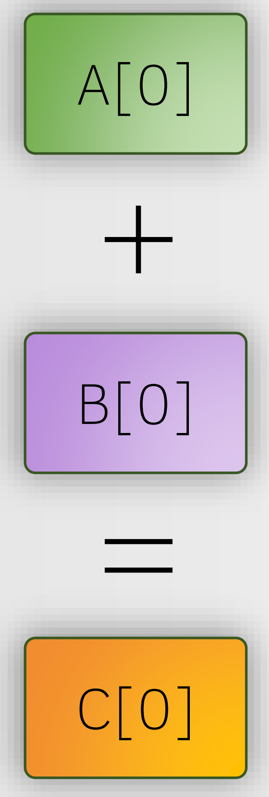
Understand the vectorized treatment of data

We want to sum vector A with vector B



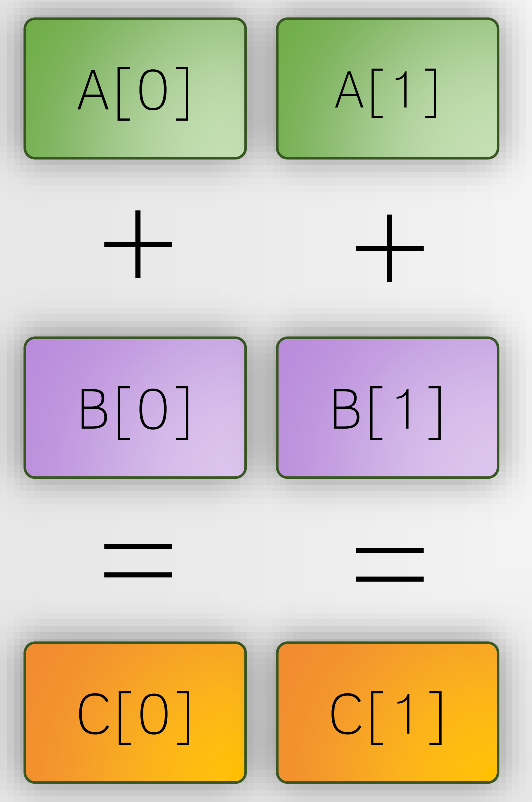
Understand the vectorized treatment of data

- In a scalar loop, the core will perform each sum one by one...



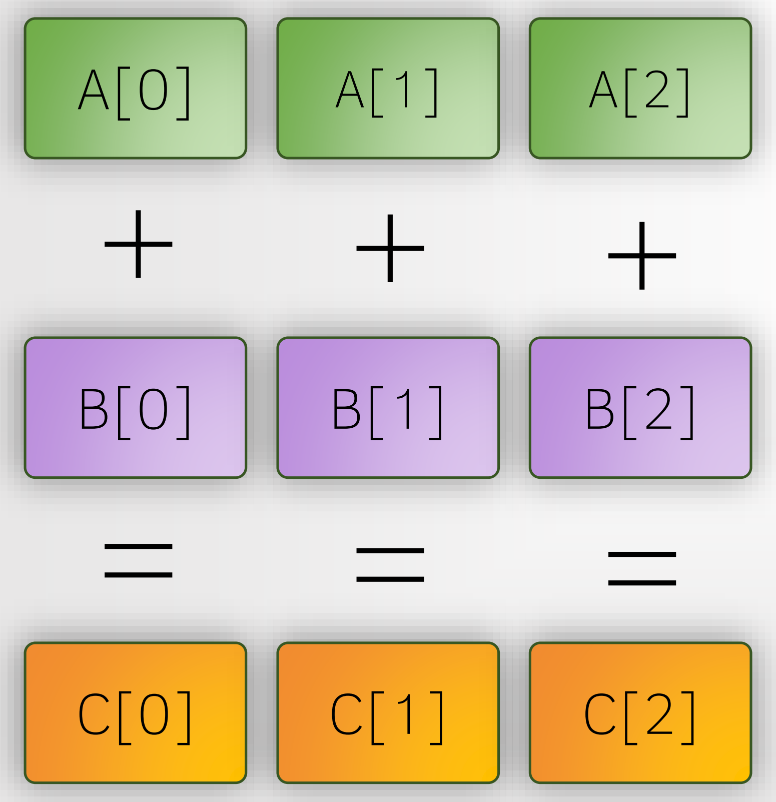
Understand the vectorized treatment of data

- In a scalar loop, the core will perform each sum one by one...



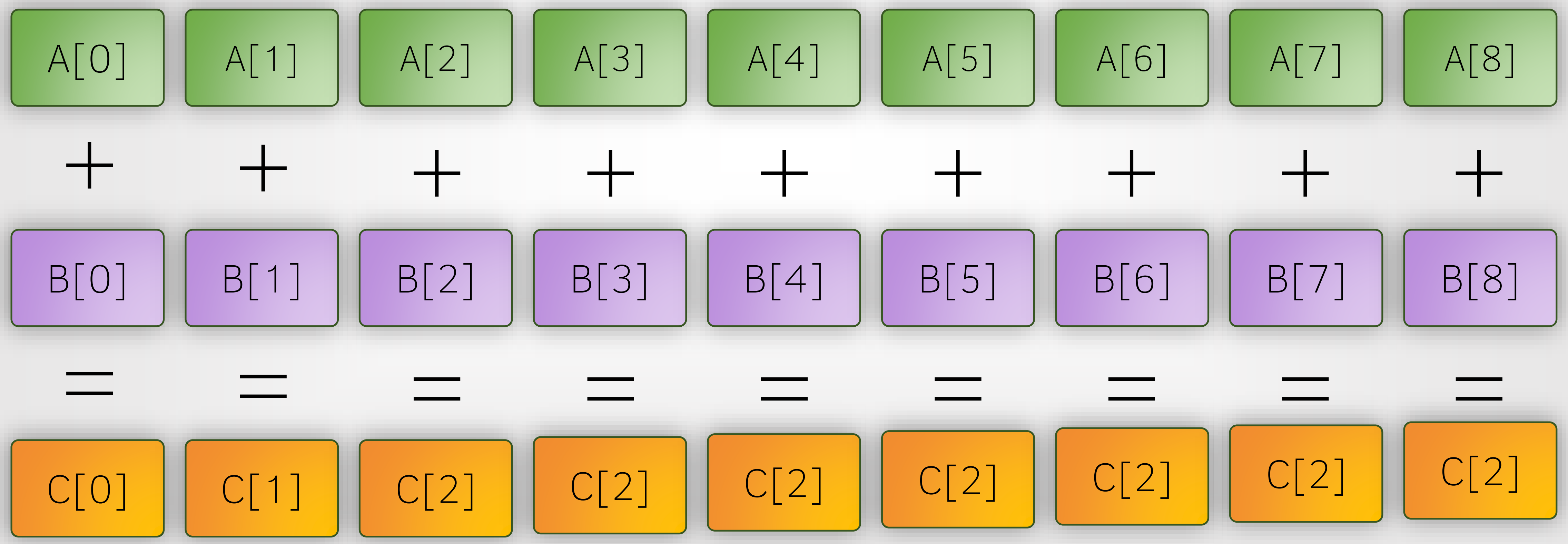
Understand the vectorized treatment of data

- In a scalar loop, the core will perform each sum one by one...



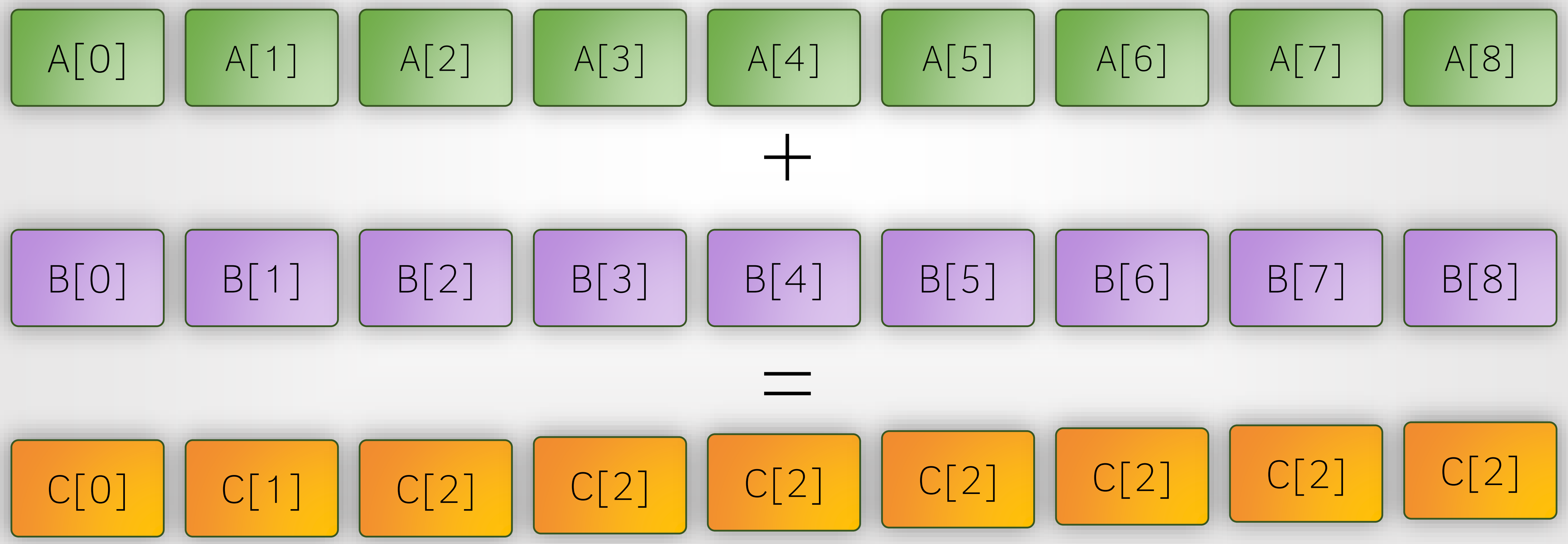
Understand the vectorized treatment of data

- In a scalar loop, the core will perform each sum one by one until the end



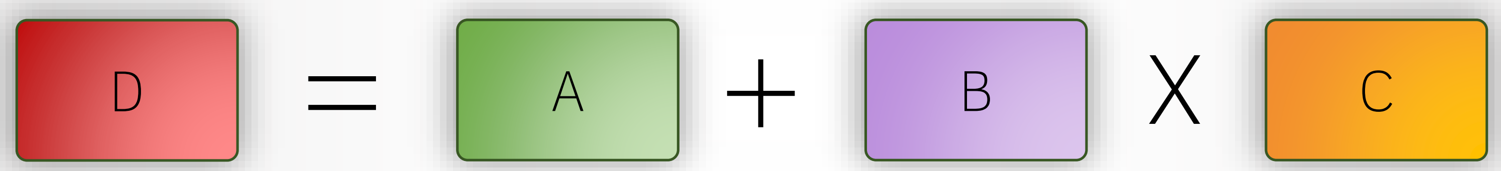
Understand the vectorized treatment of data

- The vectorized version performs the sum of all elements at once



Understand the vectorized treatment of data

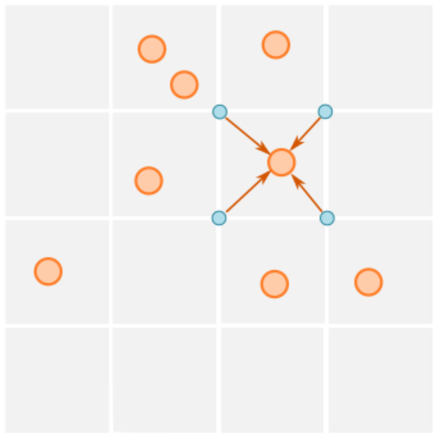
- Most modern processors perform both an addition and a multiplication in a single vectorized cycle (referred o as FMA for Fused-Add-Multiply instruction)



- If-branch can also be vectorized using masks
- Largest vectors are composed of 8 double-precision floats (AVX512 for instance)

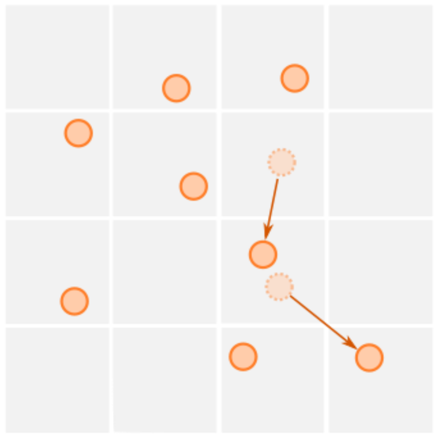
Vectorization bottlenecks

Field interpolation



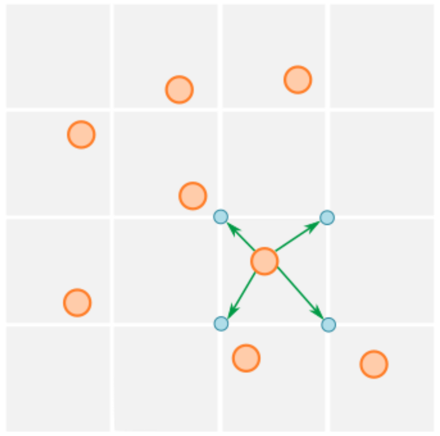
● particle
● grid node
← interpolation of fields at particle positions

Particle pusher



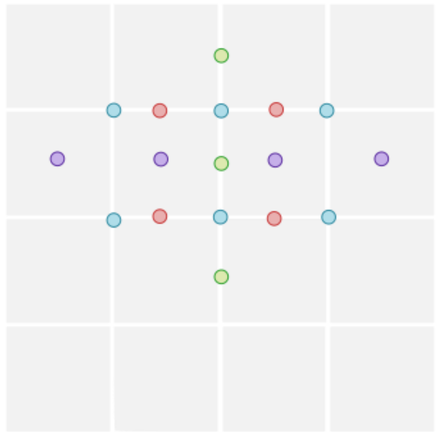
● particle
← particle movement

Charge and current projection



● particle
● grid node
← projection of particle current on the nearby nodes

Maxwell solver (staggered grid)



● Ex, Jx, By
● Ey, Jy, Bx
● Charge, Ez, Jz
● Bz

Vectorization bottlenecks

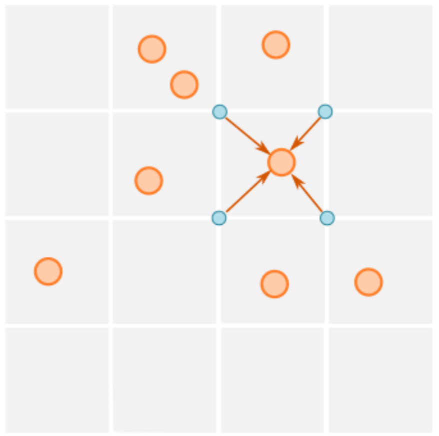
Bottleneck: need adaptation for efficient vectorization

Efficient vectorization

Bottleneck: need adaptation for vectorization to avoid memory race

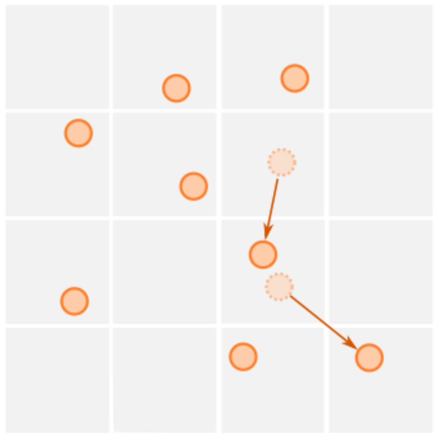
Negligible in term of computational time (stencil problem so vectorizable)

Field interpolation



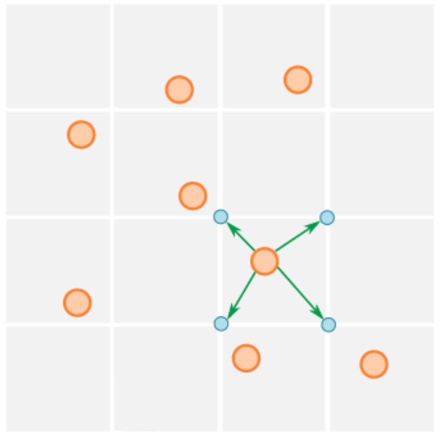
● particle
● grid node
← interpolation of fields at particle positions

Particle pusher



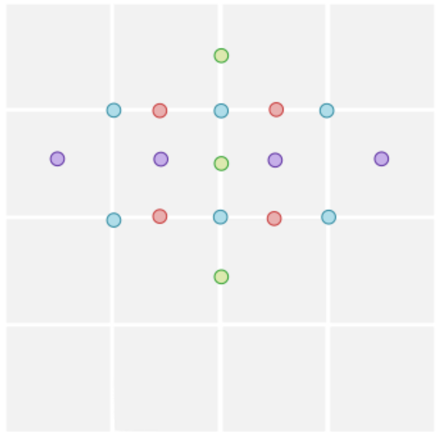
● particle
← particle movement

Charge and current projection



● particle
● grid node
← projection of particle current on the nearby nodes

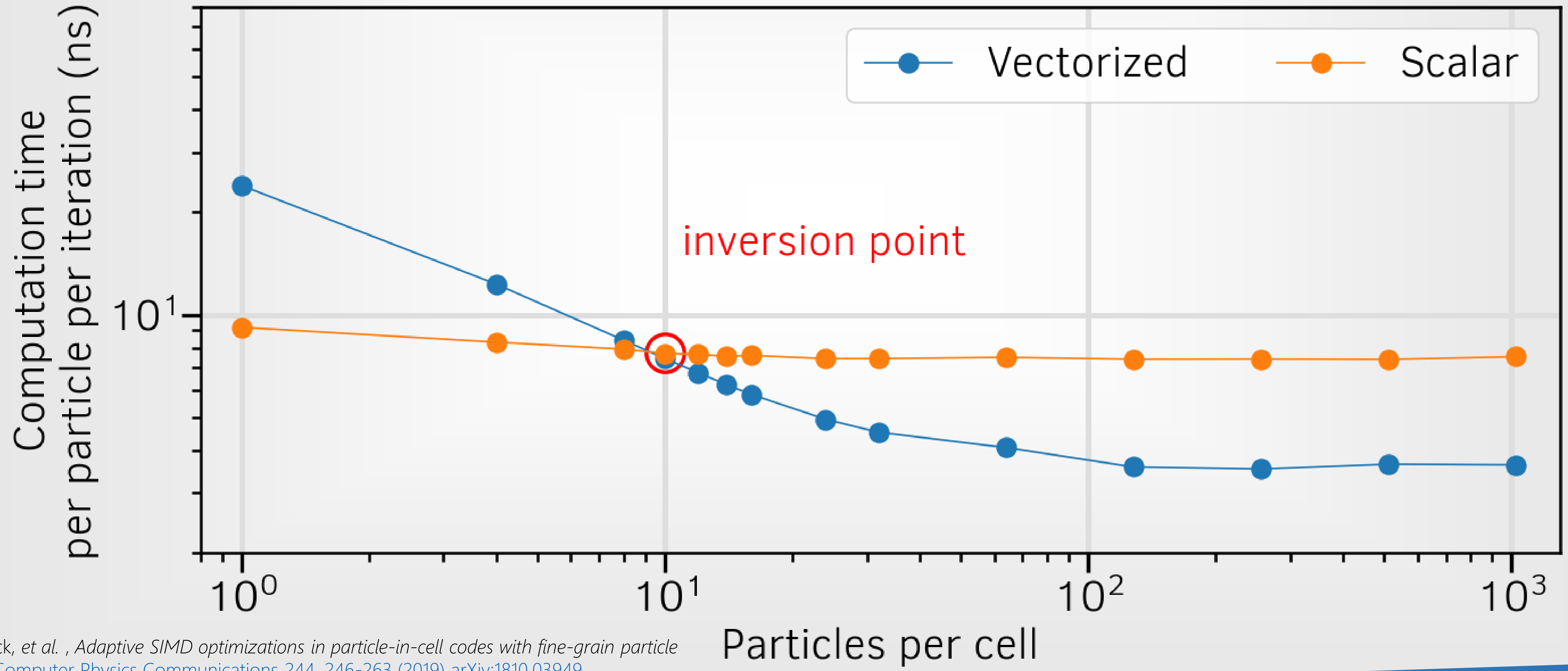
Maxwell solver (staggered grid)



● Ex, Jx, By
● Ey, Jy, Bx
● Charge, Ez, Jz
● Bz

Vectorized versus scalar operator implementations

Thermal plasma 3D benchmark on a Skylake node (2 MPIs x 24 OMPs)

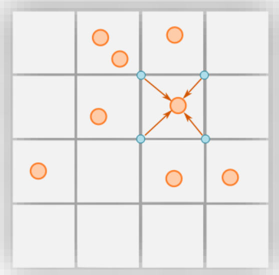


[1] A. Beck, *et al.* , *Adaptive SIMD optimizations in particle-in-cell codes with fine-grain particle sorting*, Computer Physics Communications 244, 246-263 (2019) arXiv:1810.03949

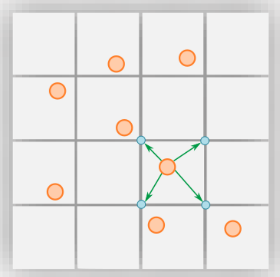
An adaptive method in time and space

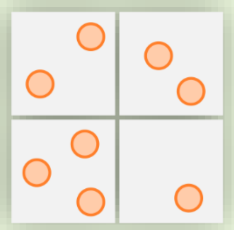
For many operators

Field interpolation



Charge and current projection





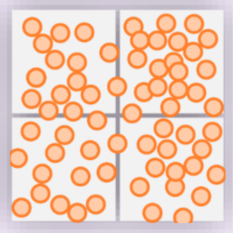
Scalar (classical) implementation

➢ more efficient for few particles per cell

adaptation in time and space at the patch level

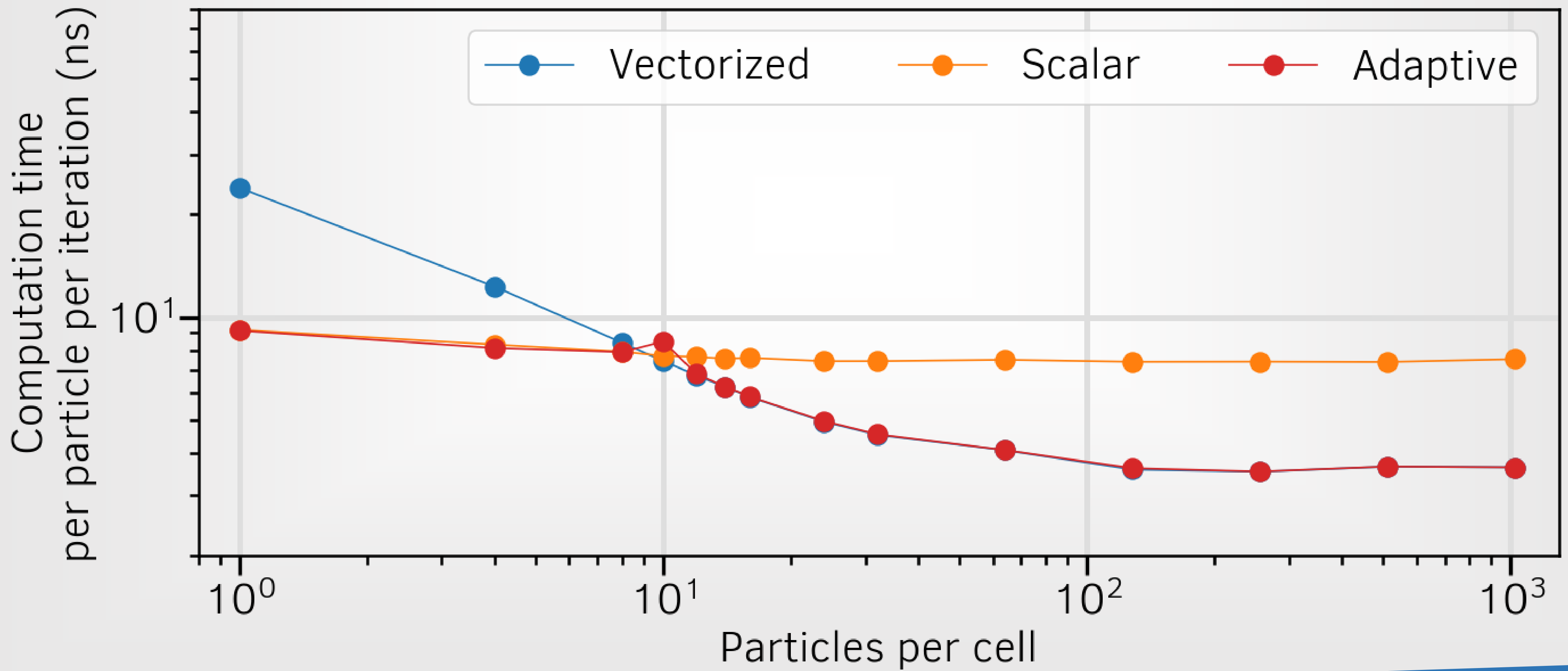
Vectorized implementation

➢ More efficient for many particles per cell

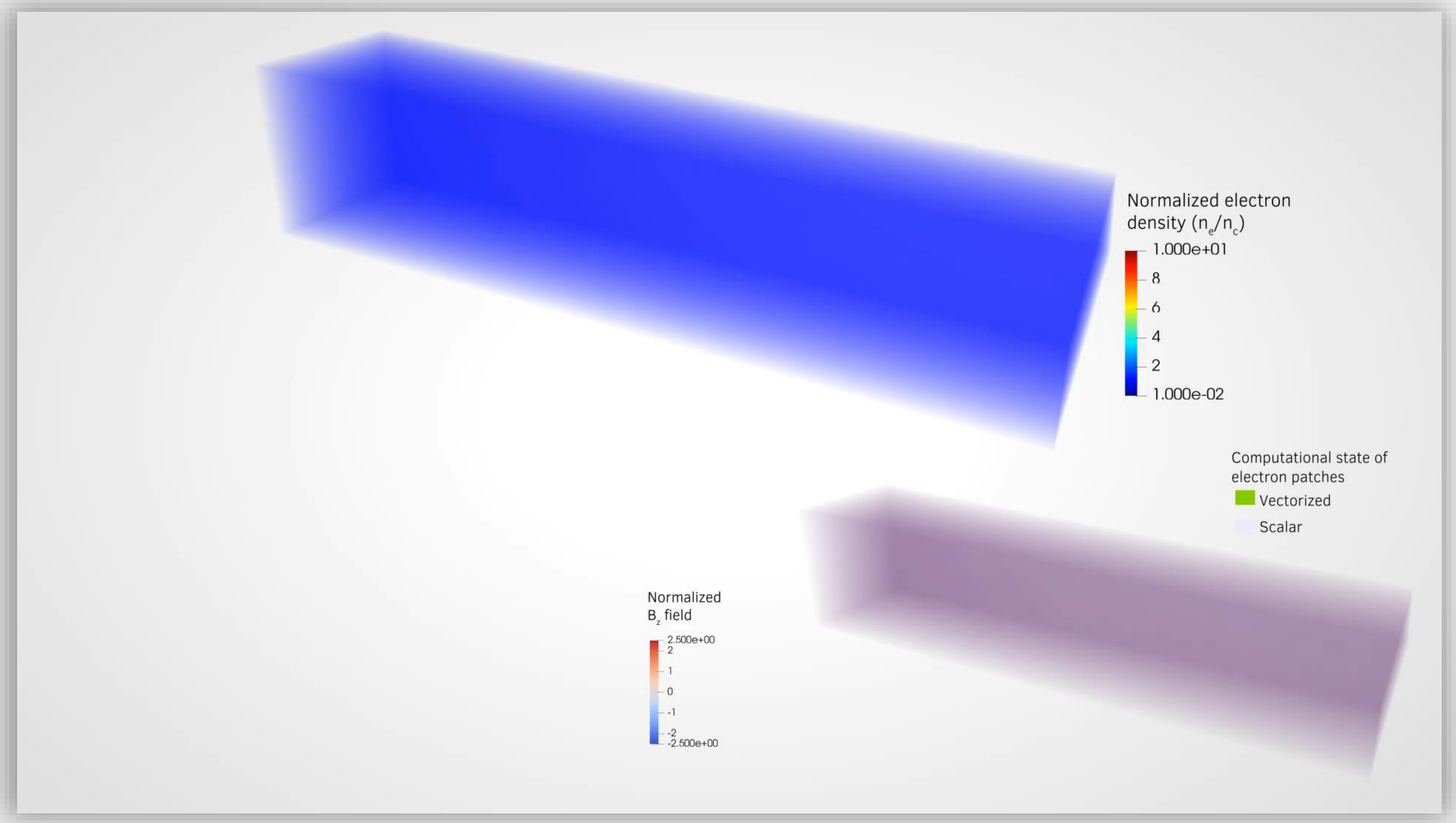


An adaptive method in time and space

Thermal plasma 3D benchmark on a Skylake node (2 MPIs x 24 OMPs)



Adaptive SIMD vectorization on large-scale simulations



Midly-relativistic collisionless shock simulation case:

- Mesh size: 2728 x 192 x 192 cells
- Patch size: 8 x 8 x 8 cells
- Domain size: 300 x 28.5 x 28.5 $(c/\omega)^3$
- Reconfiguration every 8 iterations
- 64 Intel Skylake processors (1536 cores) on Irene Joliot-Curie

Smilei) Workshop

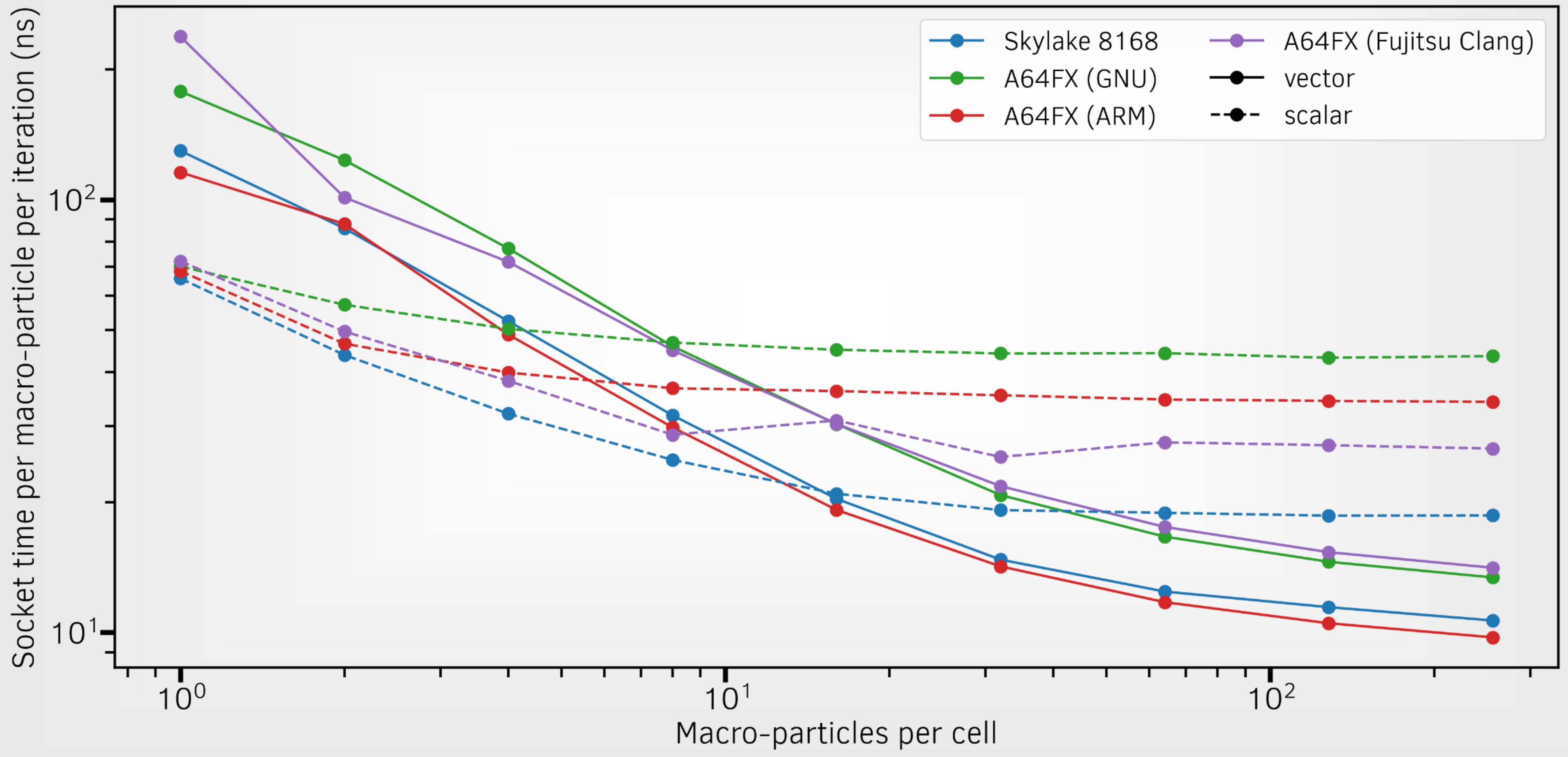
III. Adaptation to ARM-based processors

Many code adaptation to improve performance on ARM-based processors

- Code optimization (Fujitsu compiler, LLVM, GNU and ARM-clang) for the Fujitsu A64FX processor to fill the gap between the x86 processor efficiency
- Improved performance also on x86 processors especially with the GNU and LLVM compilers
- Still room for performance improvement on the most recent processors

M. Lobet, *et al.* , Simple adaptations to speed-up the Particle-In-Cell code Smilei on the ARM-based Fujitsu A64FX processor, https://dl.acm.org/doi/pdf/10.1145/3503470.3503475

Skylake and A64FX runtime comparison



Lower is better

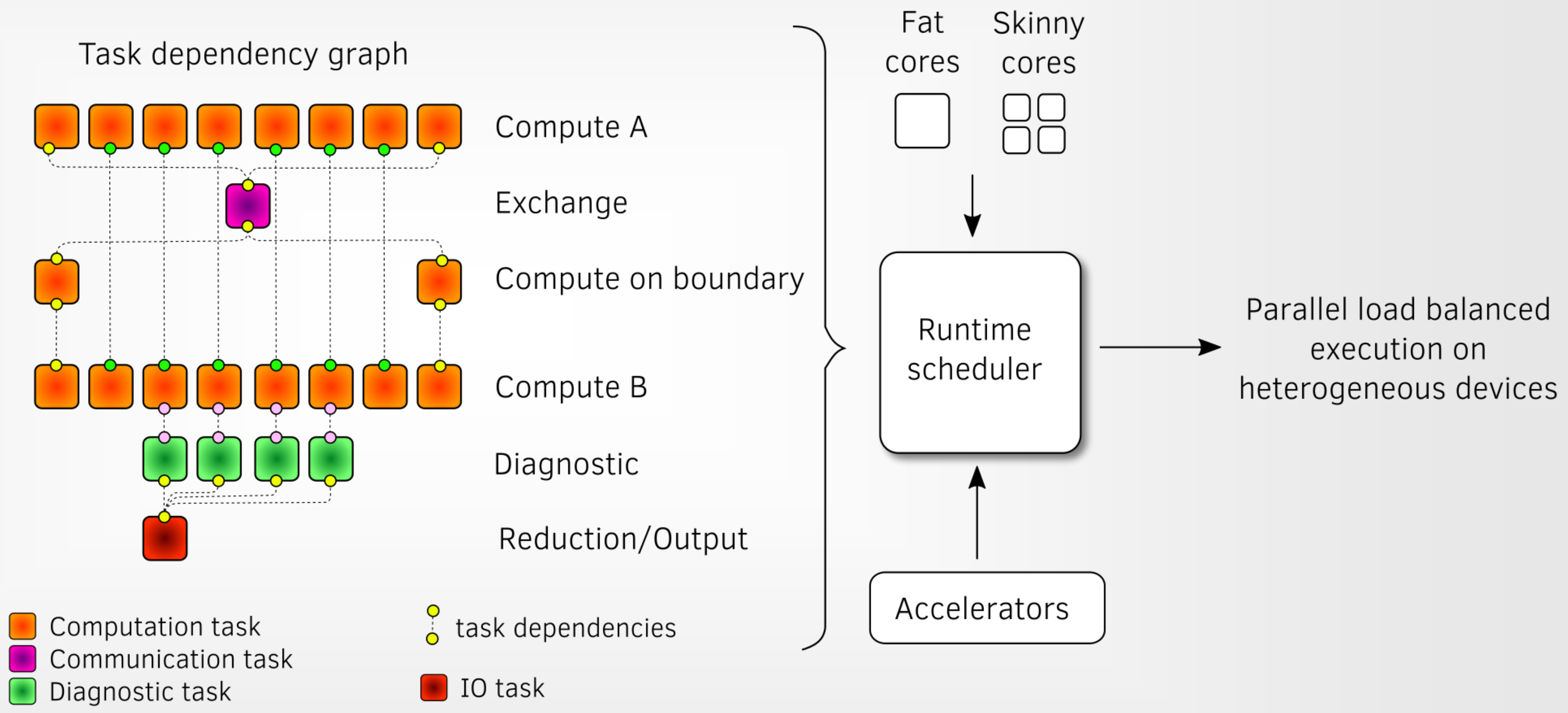
= runtime (ns) / iterations / macro-particles per socket

Smilei) Workshop

III. Task-based programming (prospective work)

Notion of asynchronism and task-based programming (work in progress)

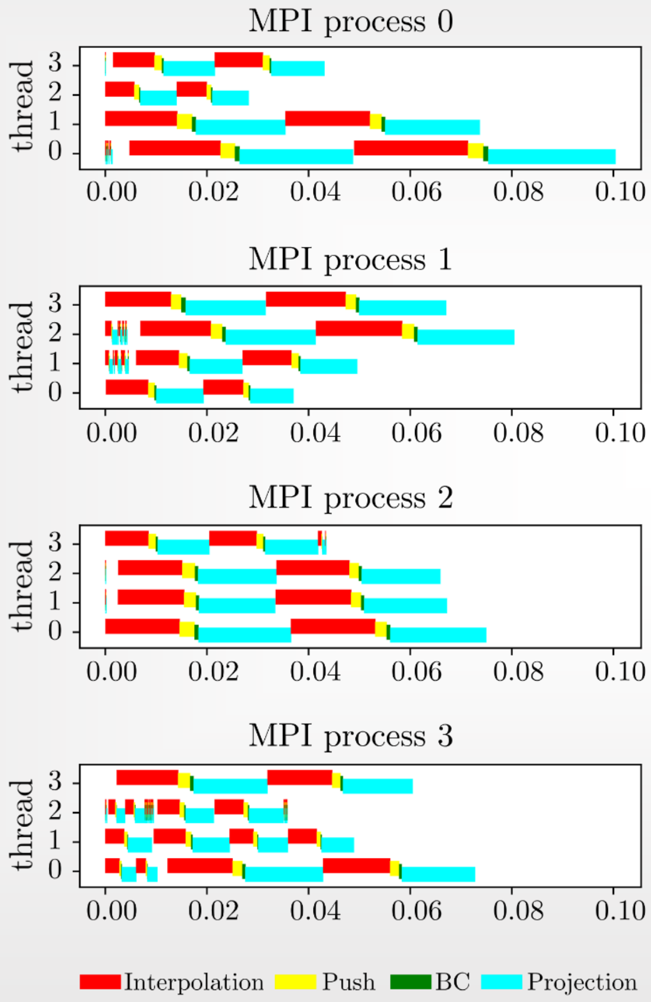
- ▶ An application can be divided temporally and spatially into inter-dependent tasks of different natures (computation, communication, IO)
- ▶ Using a smart runtime scheduler, tasks can then be run concurrently/asynchonously in parallel on a large number of cores/ on several architectures.
- ▶ To use a task-based model can speedup the code by overlapping IO and communication with computation.
- ▶ Factor of performances: grain size (fine, coarse), scheduler, dependency graph
- ▶ OpenMP task, OMPSS, StarPU, libKOMP, Eventify...

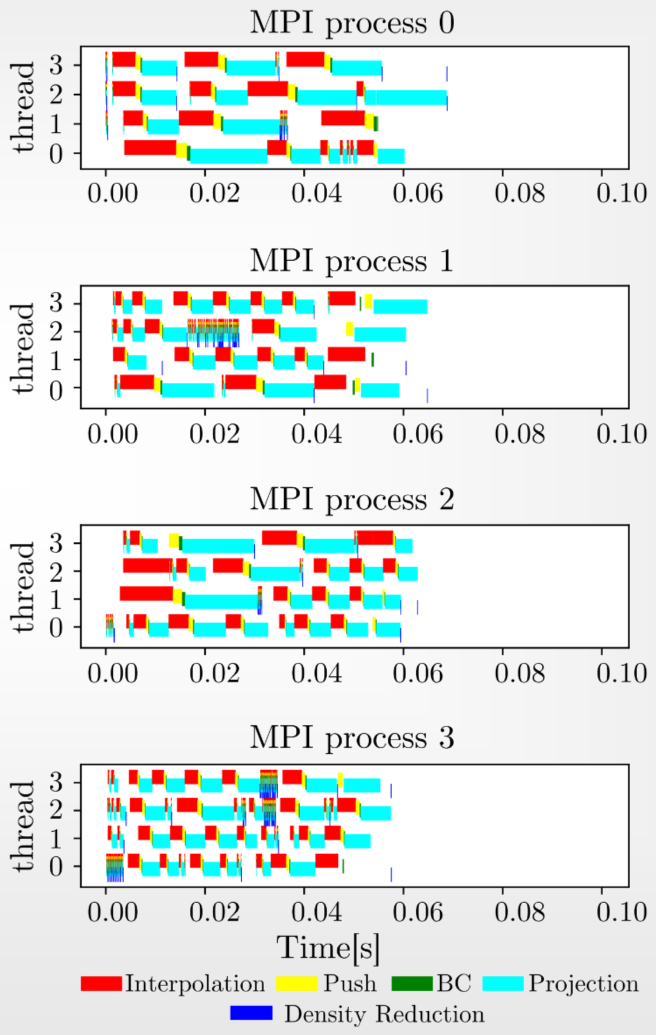


Profiling of the PIC iteration with and without tasks

Without task
(Master version)

4 MPI task + 4
OpenMP threads





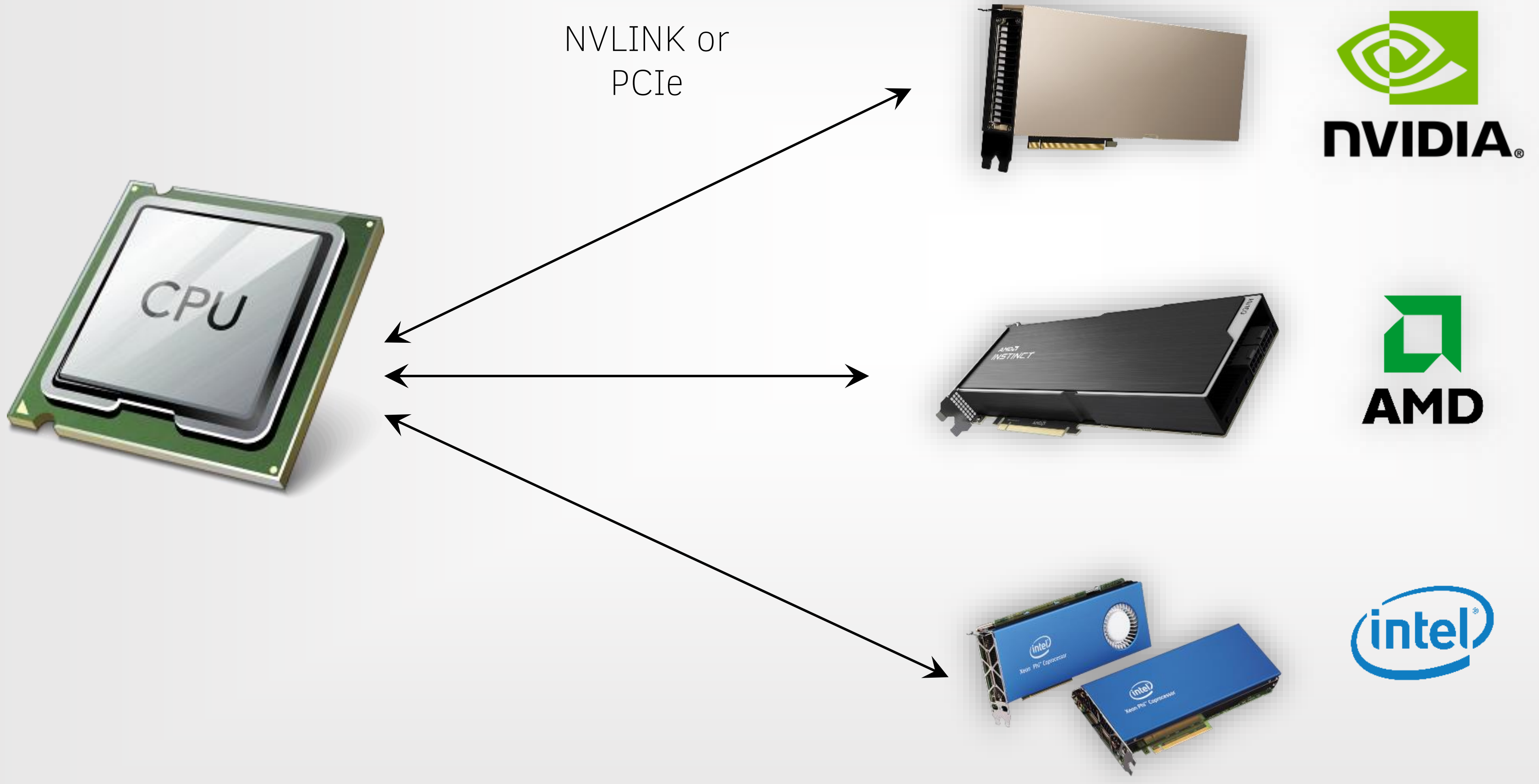
With task

Smilei) Workshop

III. GPU computing (work in progress)

How to program on GPUs

- ▶ So far, A GPU always works as an accelerator and needs a CPU for system tasks (IO, network communication…)



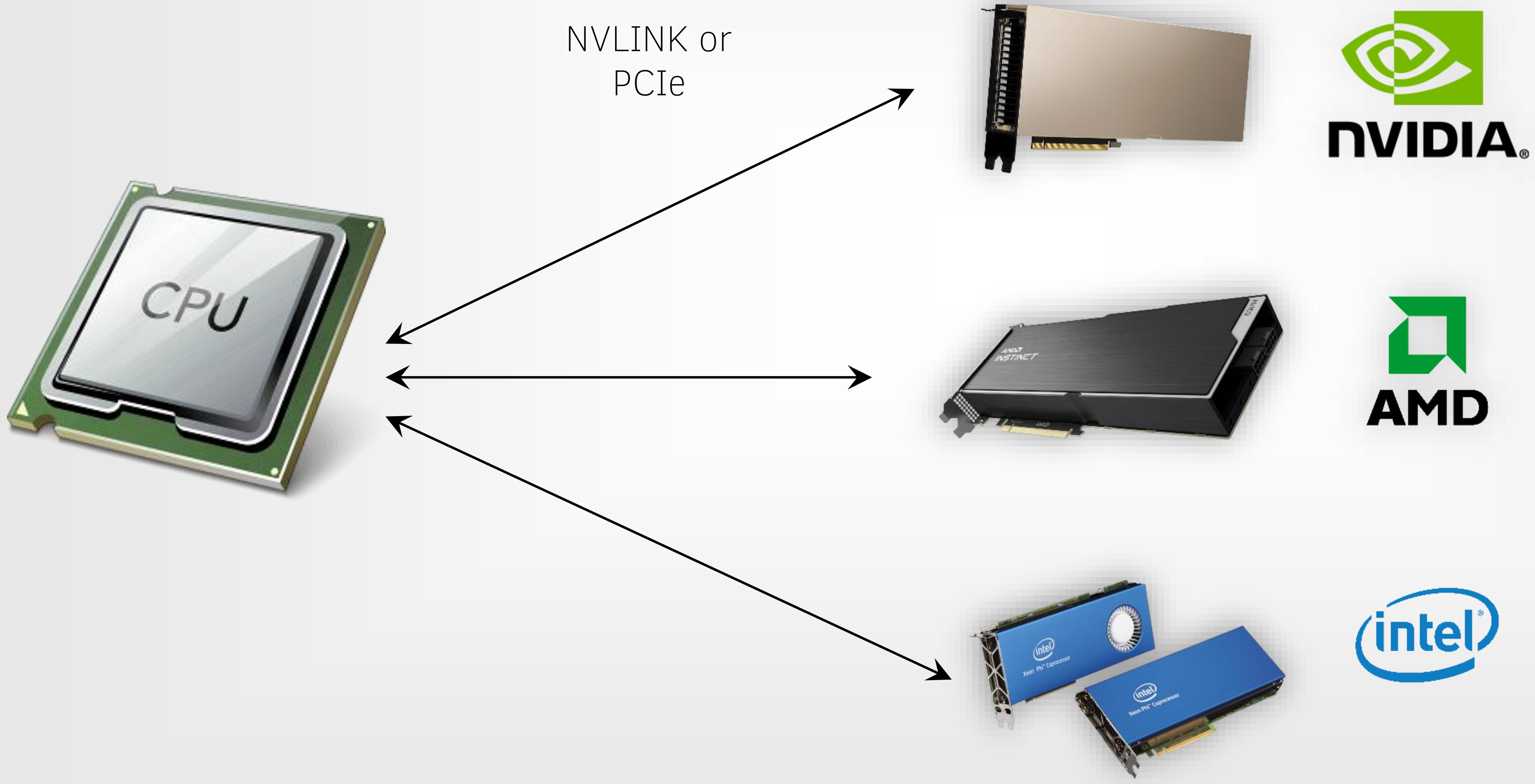
CUDA, OpenCL, OpenACC, OpenMP, SYCL, Kokkos, RAJA

HIP, OpenCL, OpenACC, OpenMP, SYCL, Kokkos, RAJA

OneAPI, OpenCL, SYCL

How to program on GPUs

- ▶ So far, A GPU always works as an accelerator and needs a CPU for system tasks (IO, network communication…)



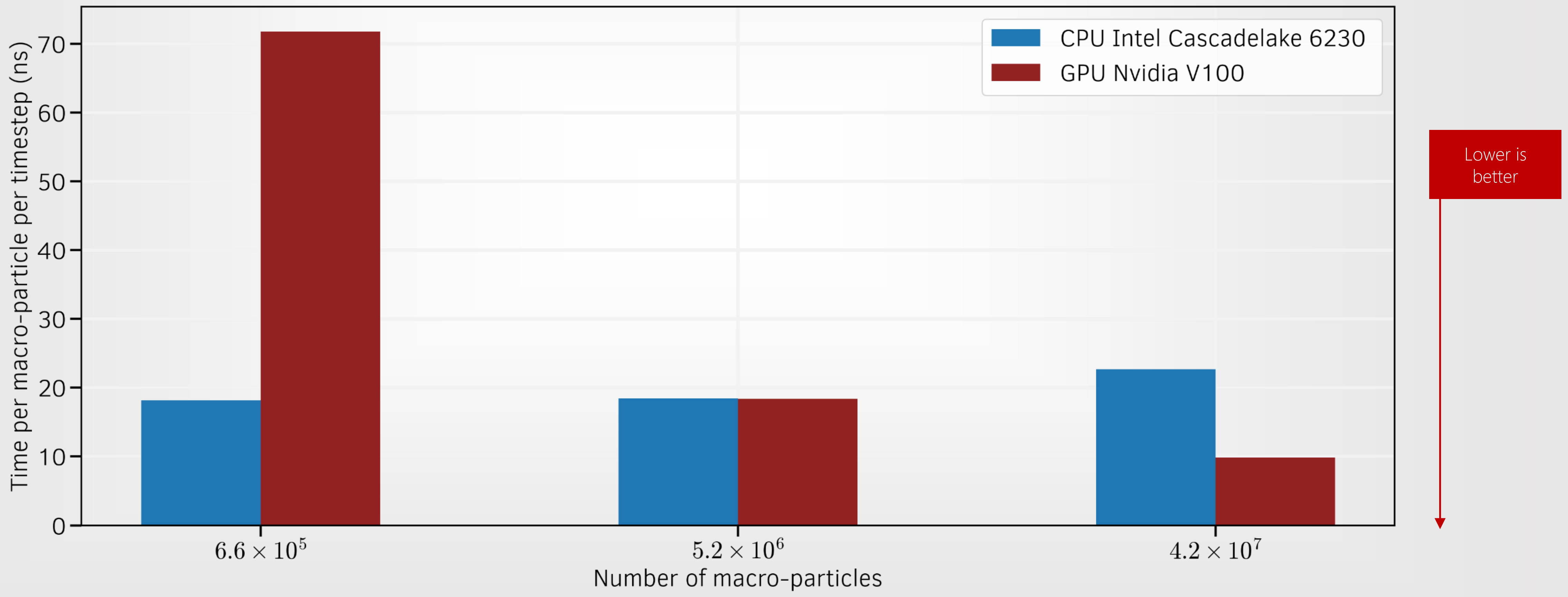
CUDA, OpenCL, OpenACC, OpenMP, SYCL, Kokkos, RAJA

HIP, OpenCL, OpenACC, OpenMP, SYCL, Kokkos, RAJA

OneAPI, OpenCL, SYCL

How to program on GPUs

- GPU version under construction, preliminary results



Conclusion

- Constant under-going efforts to adapt the code to most recent architectures
- Next priority developments in term of HPC are GPU porting (working in 3D with basic operators) and ARM-based processor optimization
- Node-level efficiency is crucial for Exascale computing

Acknowledgement

Thank you for your attention