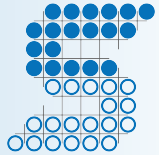
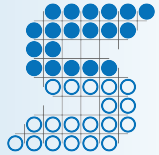
PIC codes in the HPC environment

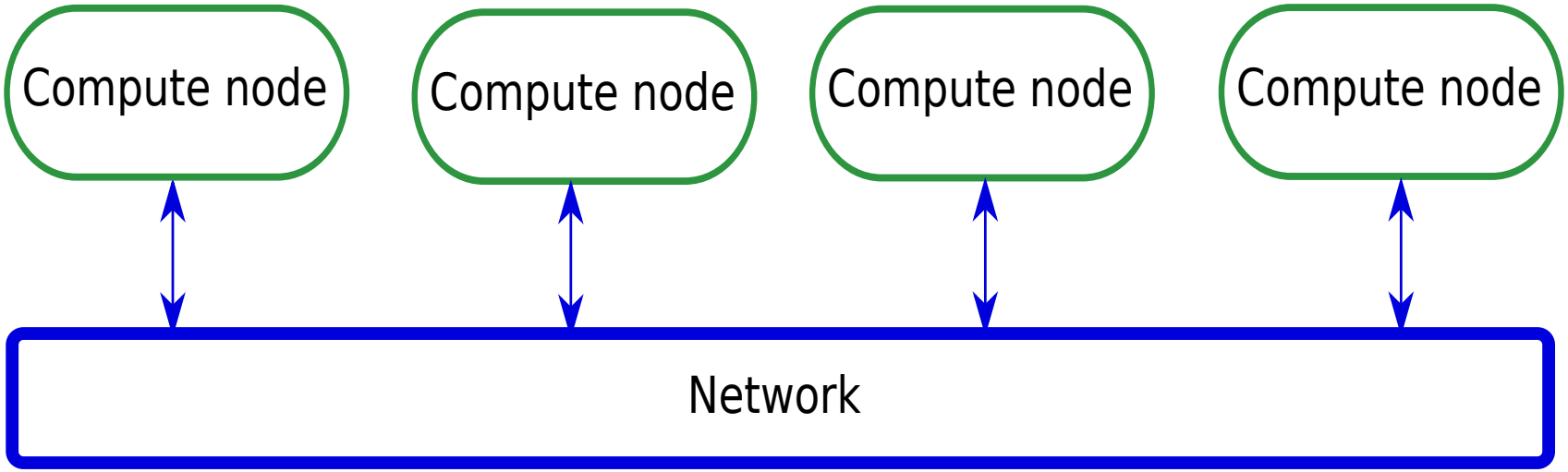
Structure



Structure

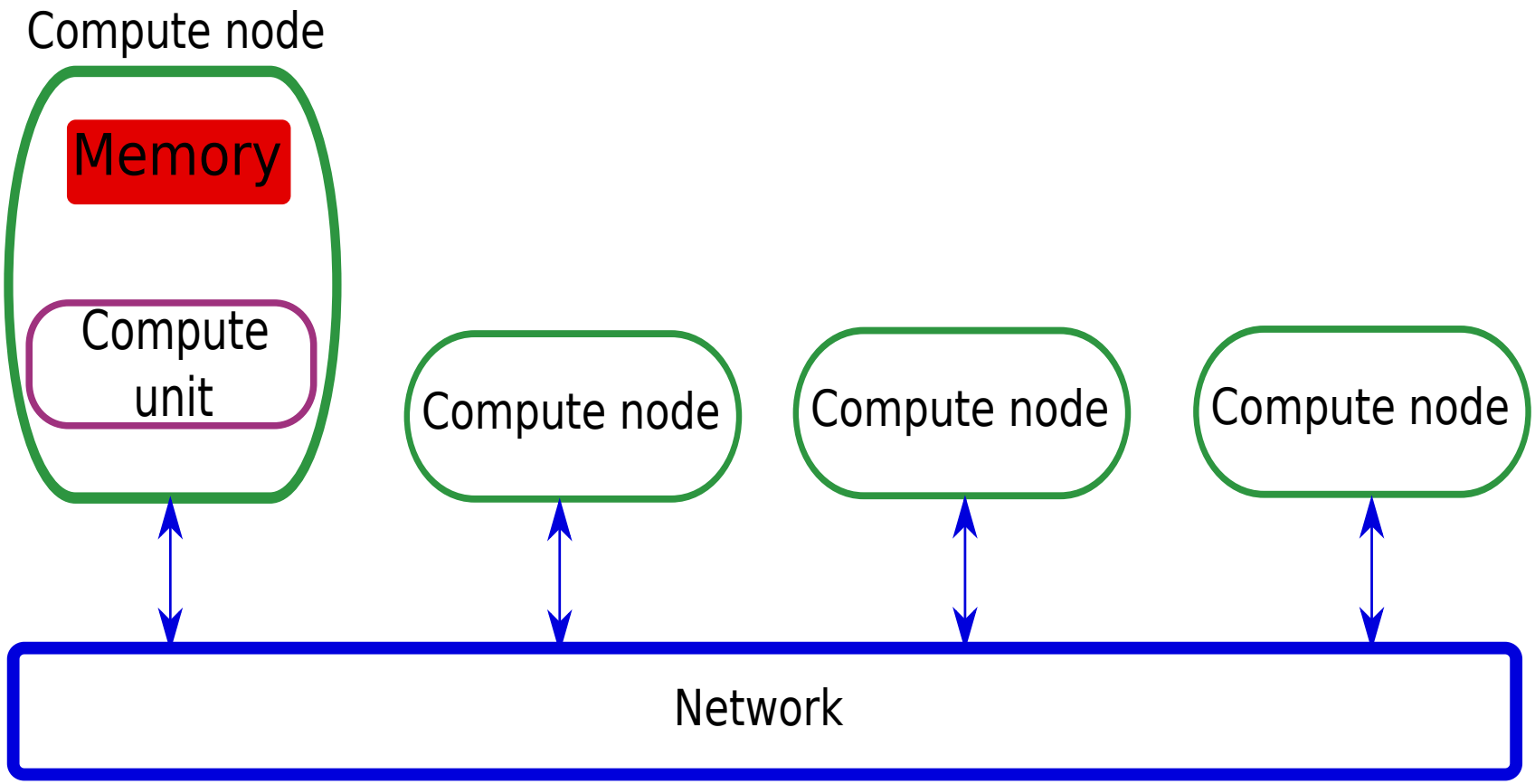


What is a super computer ?



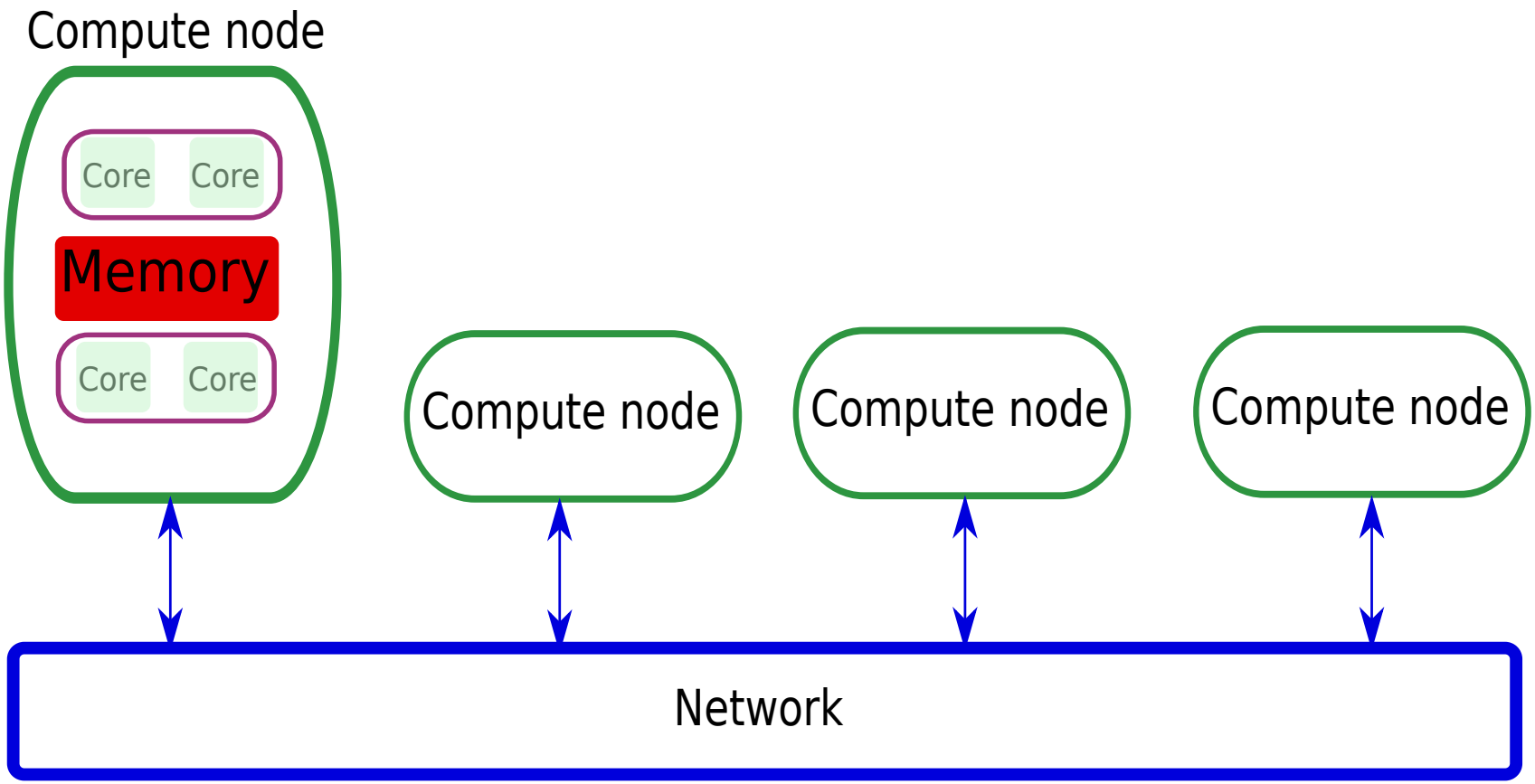
Distributed computing

What is a super computer ?



Distributed memory system

What is a super computer ?



Distributed {shared memory} system

Objective exaflop/s

Tianhe (China, June 2013) : 31 PFLOPS for 17 MW gives 1.85 GFLOPS/W.

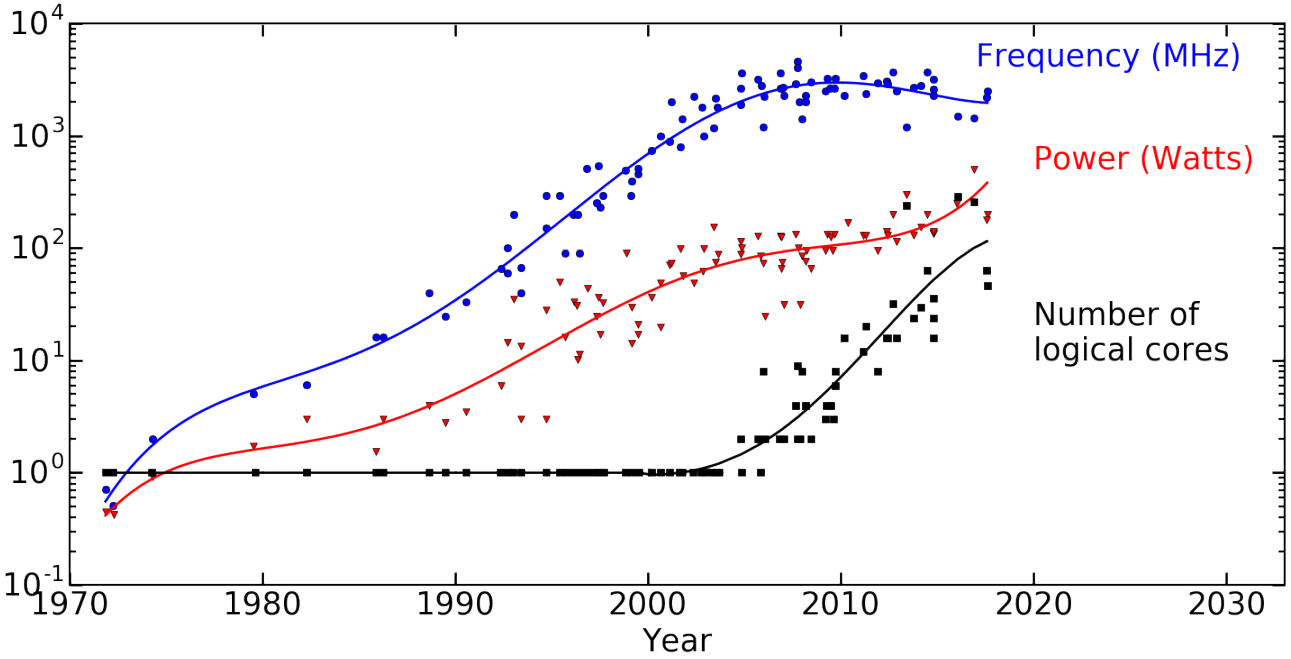
Extrapolation : 1000 PFLOPS ==> 540 MW !

= ? =

The objective is $\boxed{P < 20\ \mathrm{MW}}$

The challenge for constructors is to increase both **total performance** and **energy efficiency** of computing nodes.

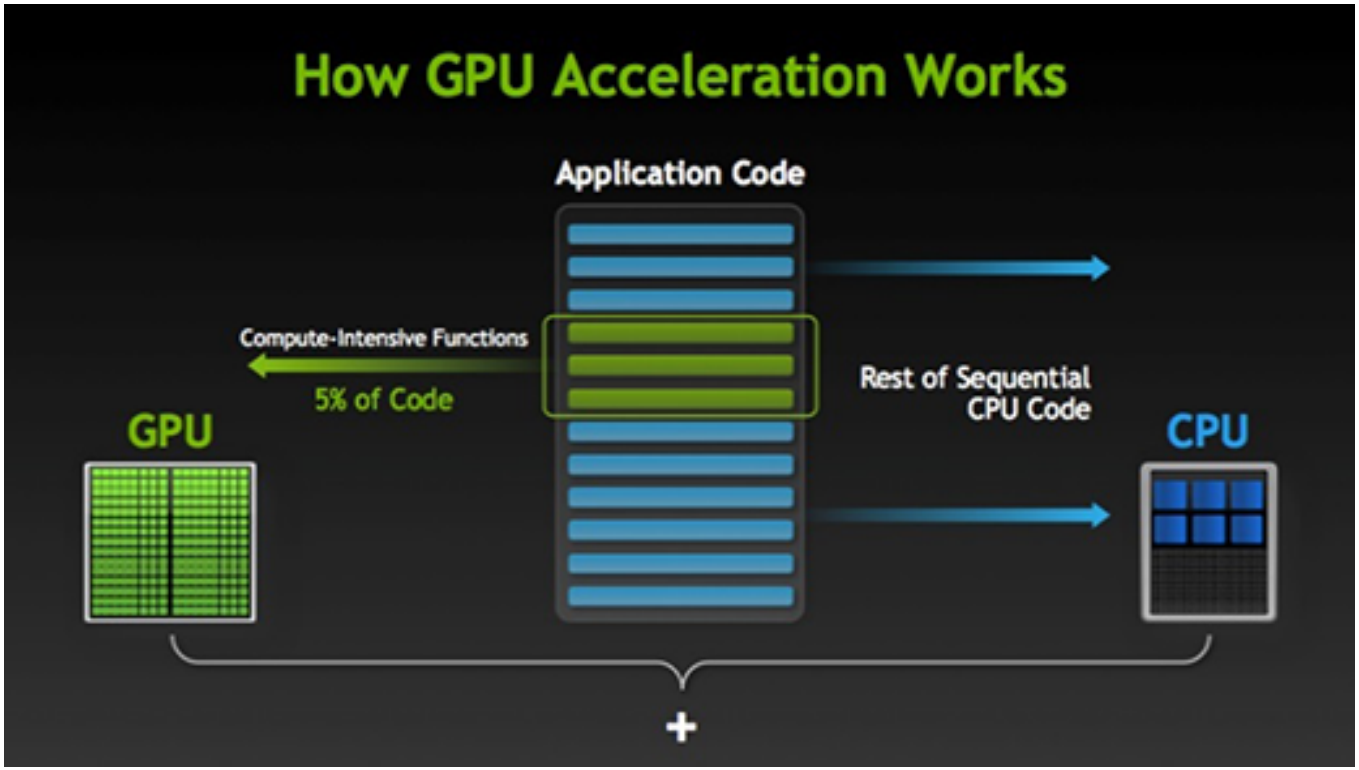
Constructors strategy : 1) Many core



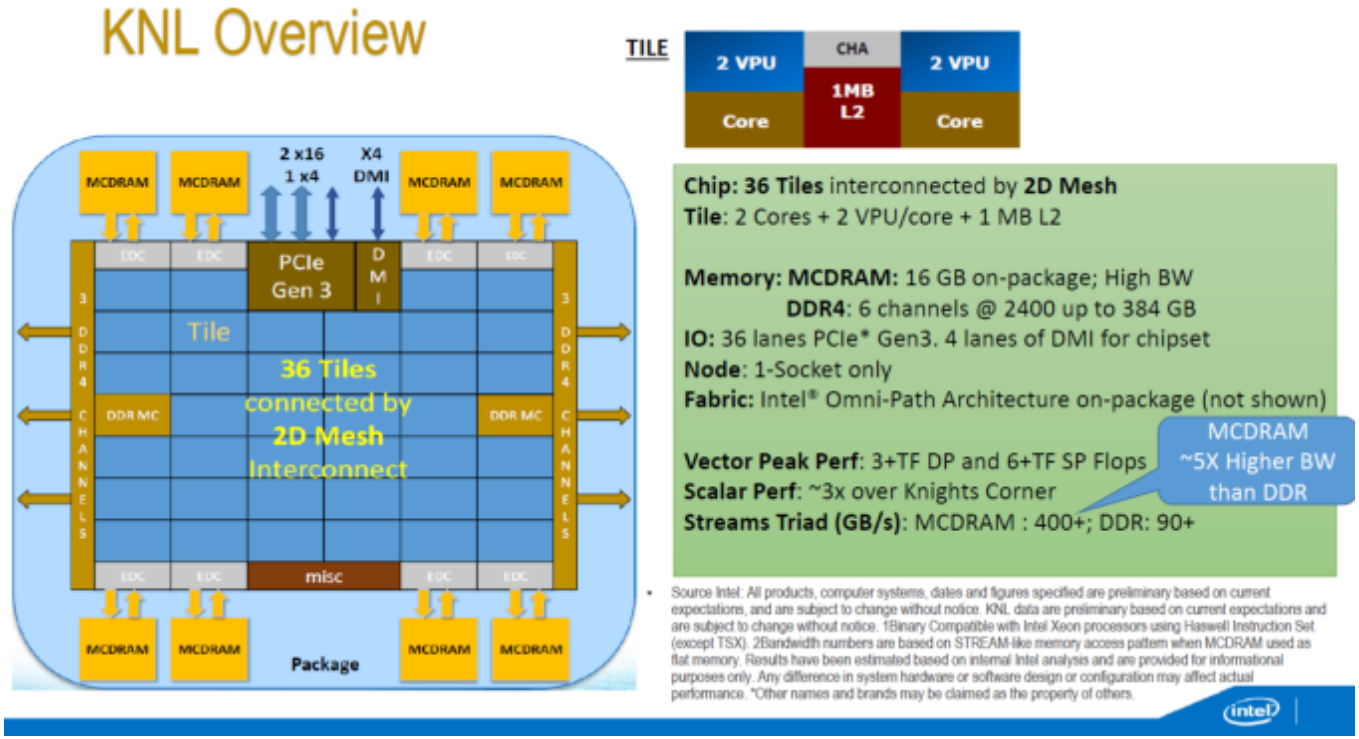
- Increased performances
- Reasonable energy budget

Constructors strategy : 2) GPGPU

NVIDIA & AMD :General Purpose Graphical Processor Unit



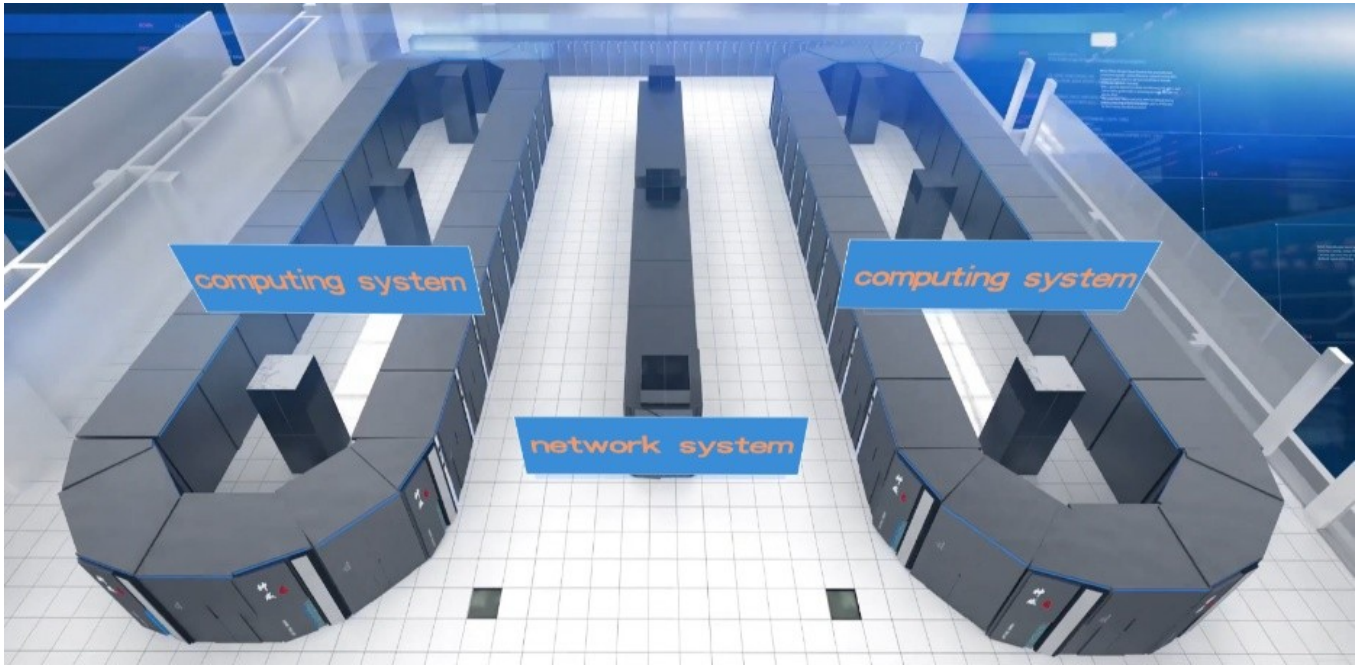
- Most energy efficient architecture today
- Difficult to adress :
  - Libraries : Cuda, OpenCl.
  - Directives programming : OpenMP 4 ou openACC.



- Powers several top HPC systems.
- + Irene (France) - Aurora (U.S)
- Supposedly accessible through “Normal” programming but relies criticaly on the SIMD instruction set.

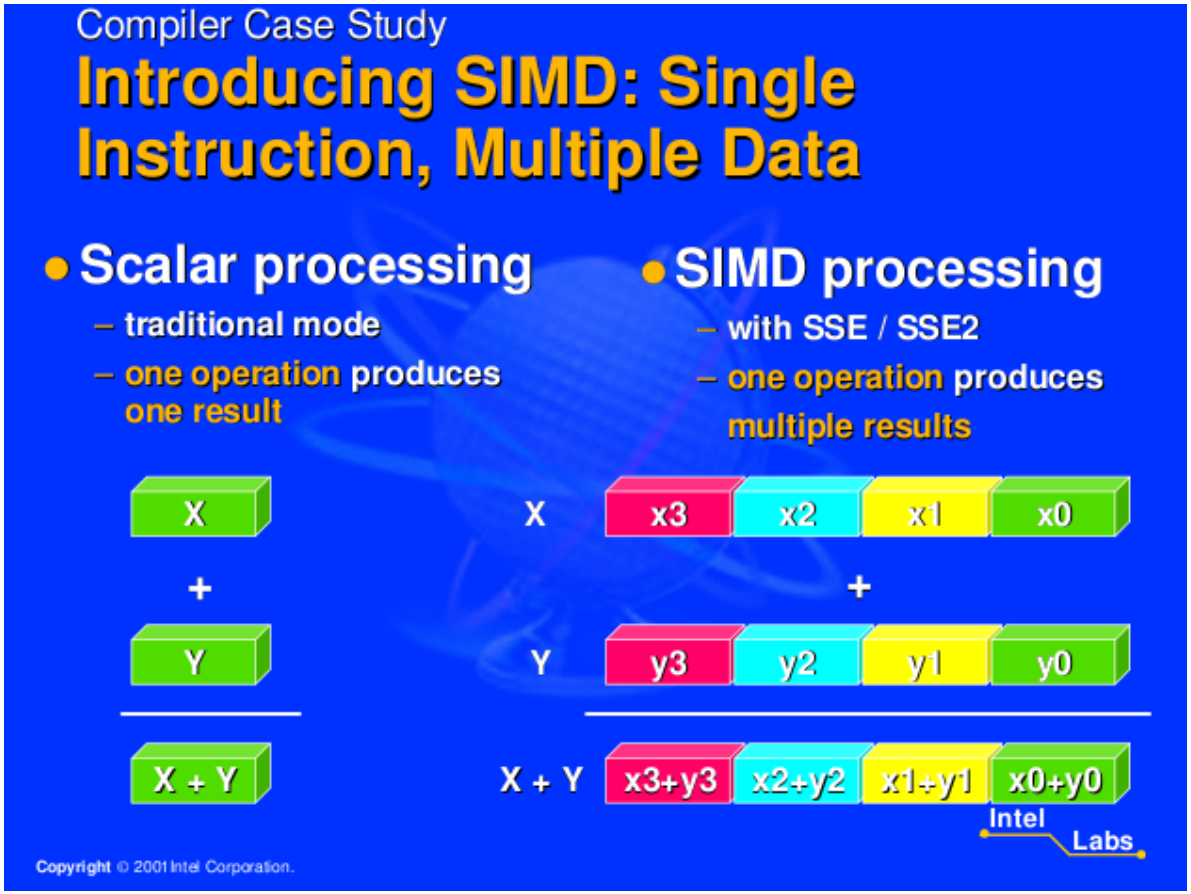
Constructors strategy : 4) China

Architecture SunWay



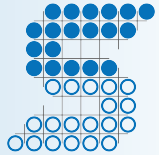
- Most powerful system in the world : 93 PFLOPS.
- 15 MW
- The SunWay architecture mimicks Xeon Phi.

Constructors strategy : 5) Vectorization



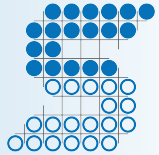
- Excellent potential speed up, very good power budget.
- Heavy constraints on data structure and algorithm.
- Difficult to use at its full extent in a PIC code.

Official announcments for Exascale



- U.S. : Exascale for 2021. No specifications.
- Japan : “Post K Supercomputer”. EFLOPS for 2020. Architecture ARM.
- China : 3 exascale systems for 2020.
- Europe : 2 Exascale systems for 2022. At least 1 powered by European technology (probably ARM).

Why am I concerned ? What should I do ?



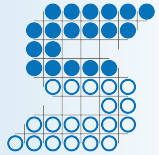
As a developer

1. Expose parallelism. Massive parallelization is key.
2. Focus on the algorithm and data structures. Not on architectures.
3. Reduce data movement : Computation is becoming cheaper, loads and stores not so much.
4. Be aware of the increasing gap between peak power and effective performances. The race to exascale is becoming a race to exaflops.

As a scientist

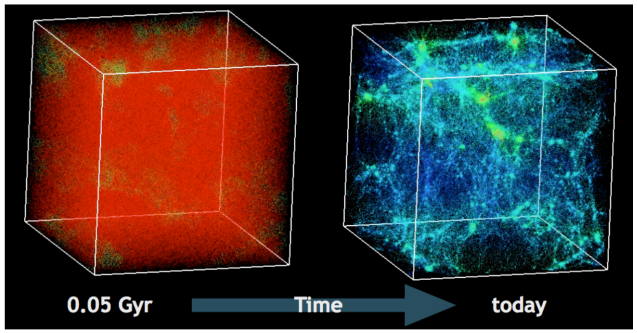
1. Collaborate with experts : complexity of HPC systems increases a lot !

Structure



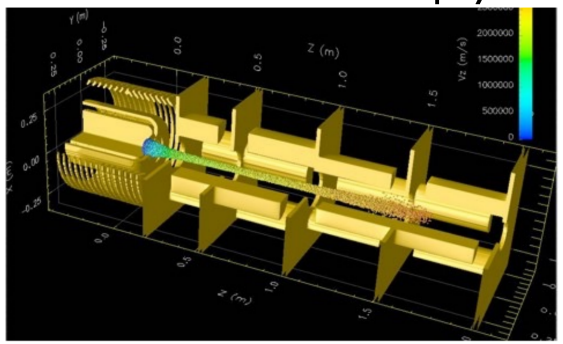
The Particle-In-Cell (PIC) method is a central tool for simulation over a wide range of physics studies

Cosmology



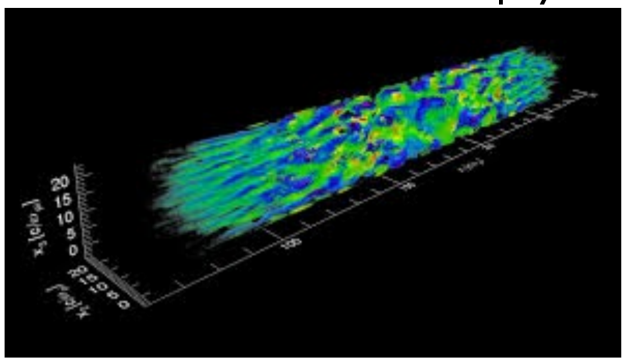
source: K. Heitmann, Argonne National Lab

Accelerator physics



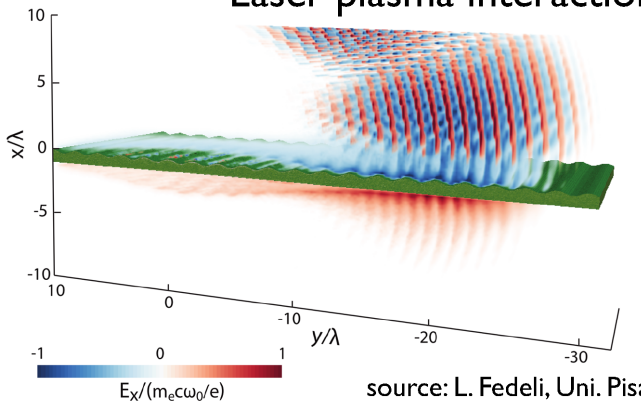
source: WARP , Berkeley Lab

Relativistic astrophysics



source: F. Fiuza, Livermore National Lab

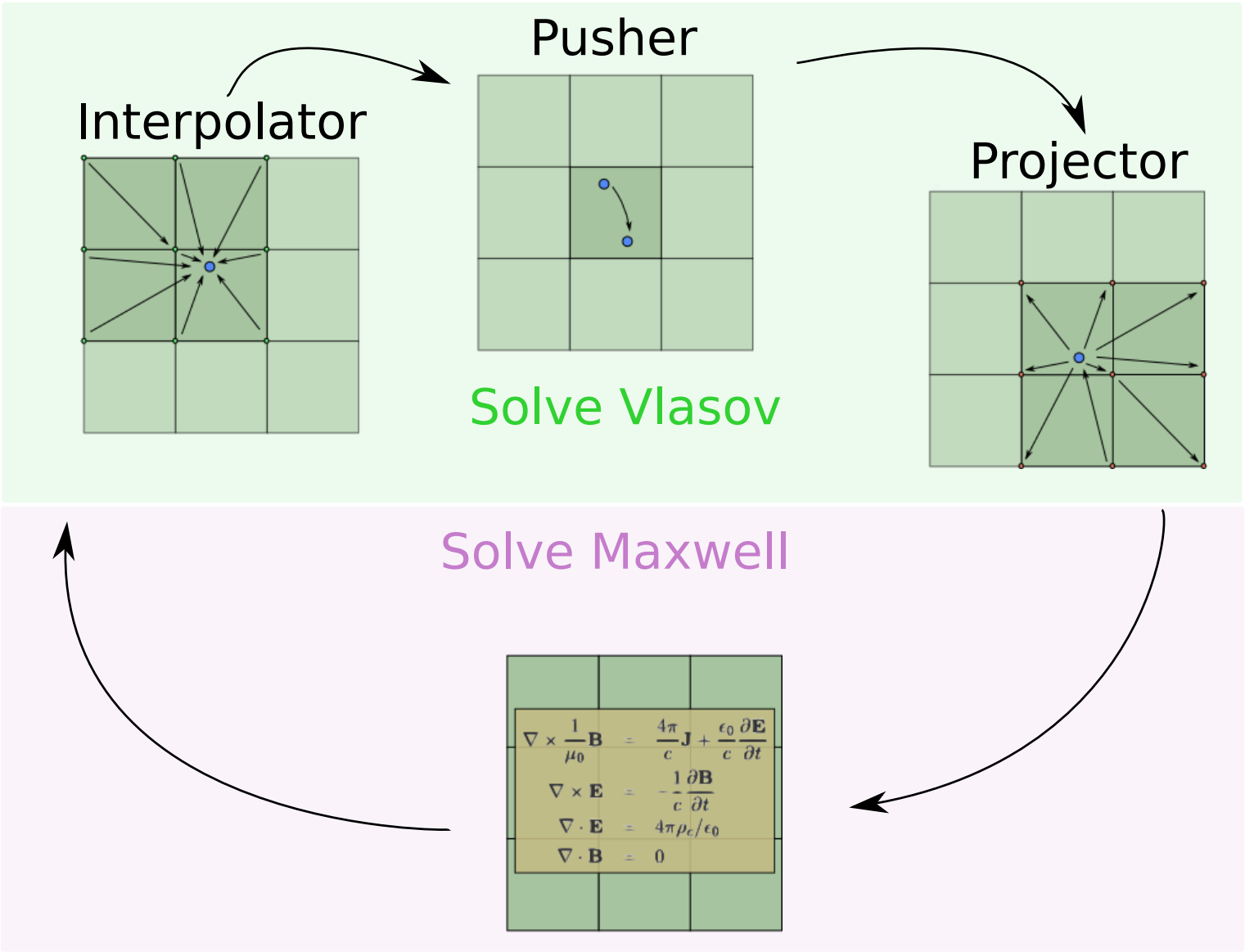
Laser plasma interaction



source: L. Fedeli, Uni. Pisa

- Conceptually simple
- Efficiently implemented on (massively) parallel super-computers

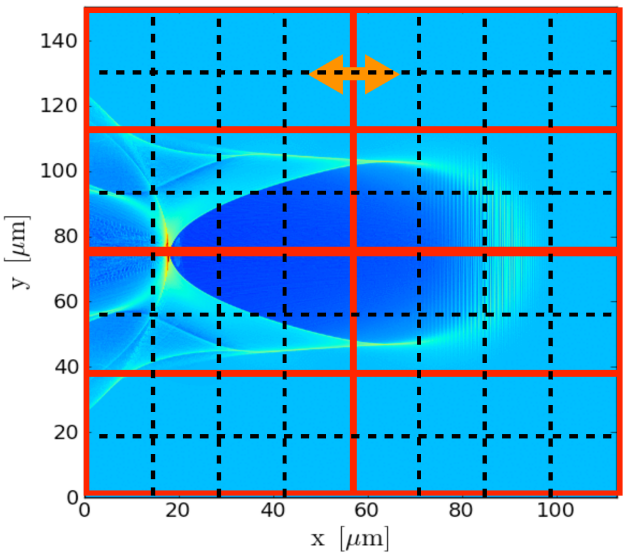
Explicit PIC code principle

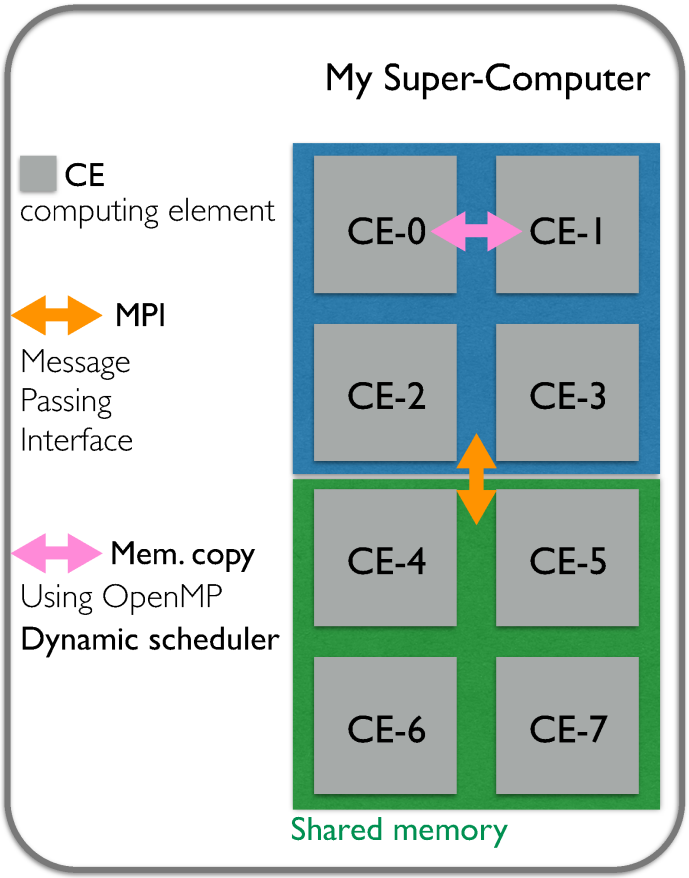


Domain decomposition

PIC code are 'easily' parallelized using domain decomposition

Simulation of laser wakefield acceleration of electrons

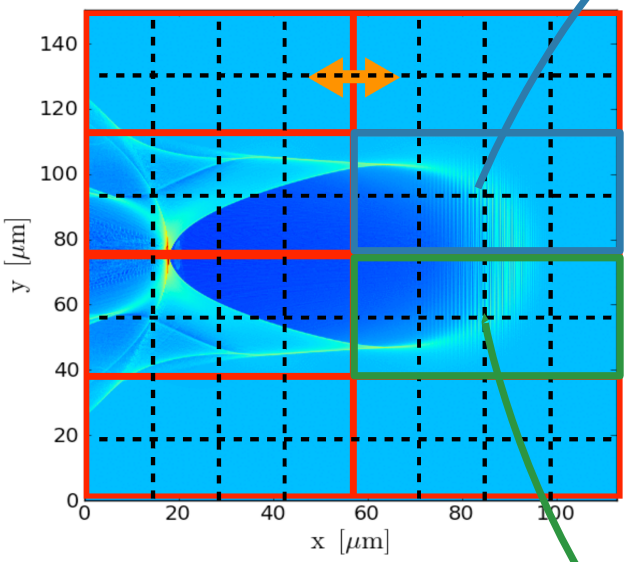




Domain decomposition : MPI

PIC code are 'easily' parallelized using domain decomposition

Simulation of laser wakefield acceleration of electrons



My Super-Computer

CE
computing element

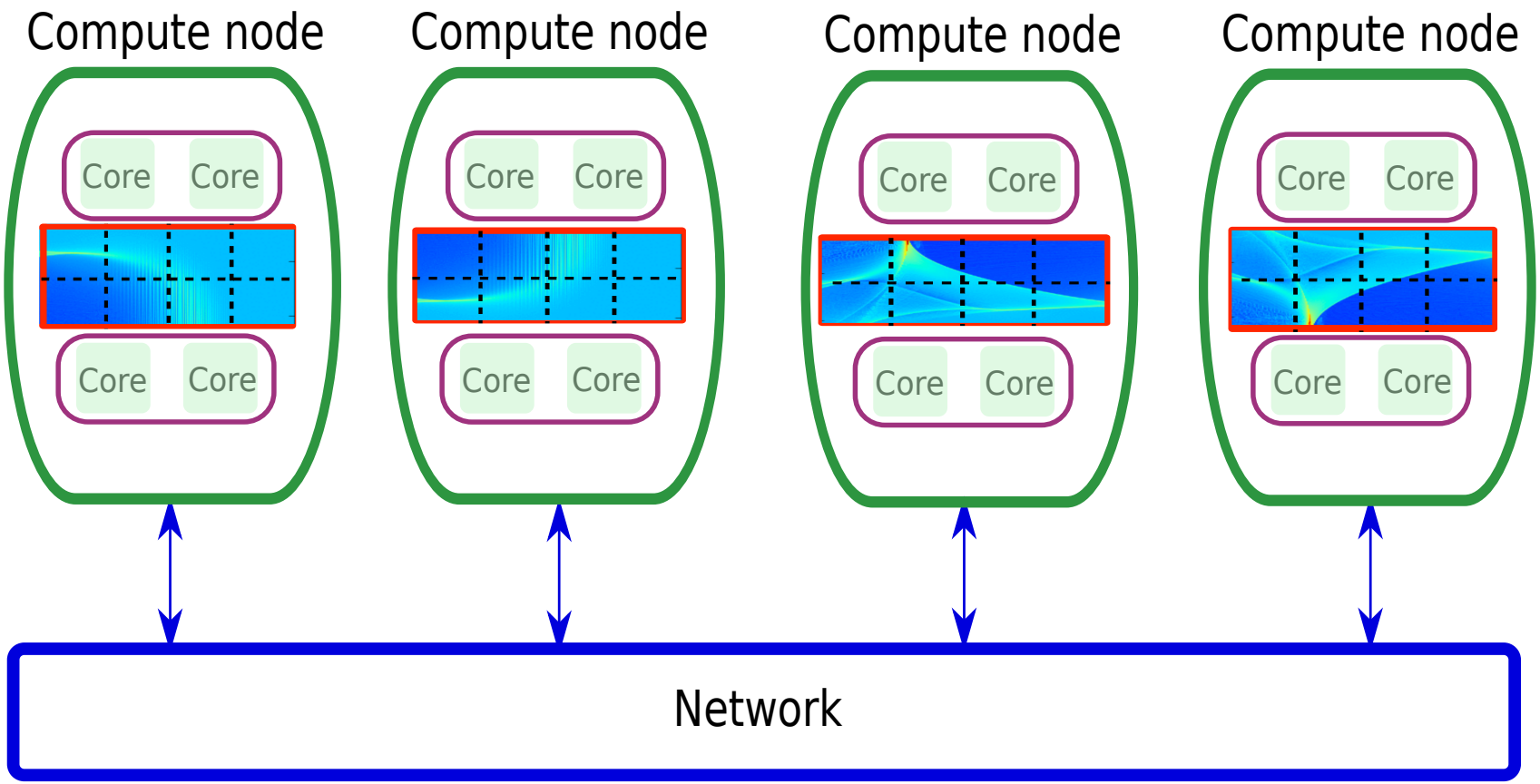
MPI
Message
Passing
Interface

Mem. copy
Using OpenMP
Dynamic scheduler

CE-0 CE-1
CE-2 CE-3
CE-4 CE-5
CE-6 CE-7

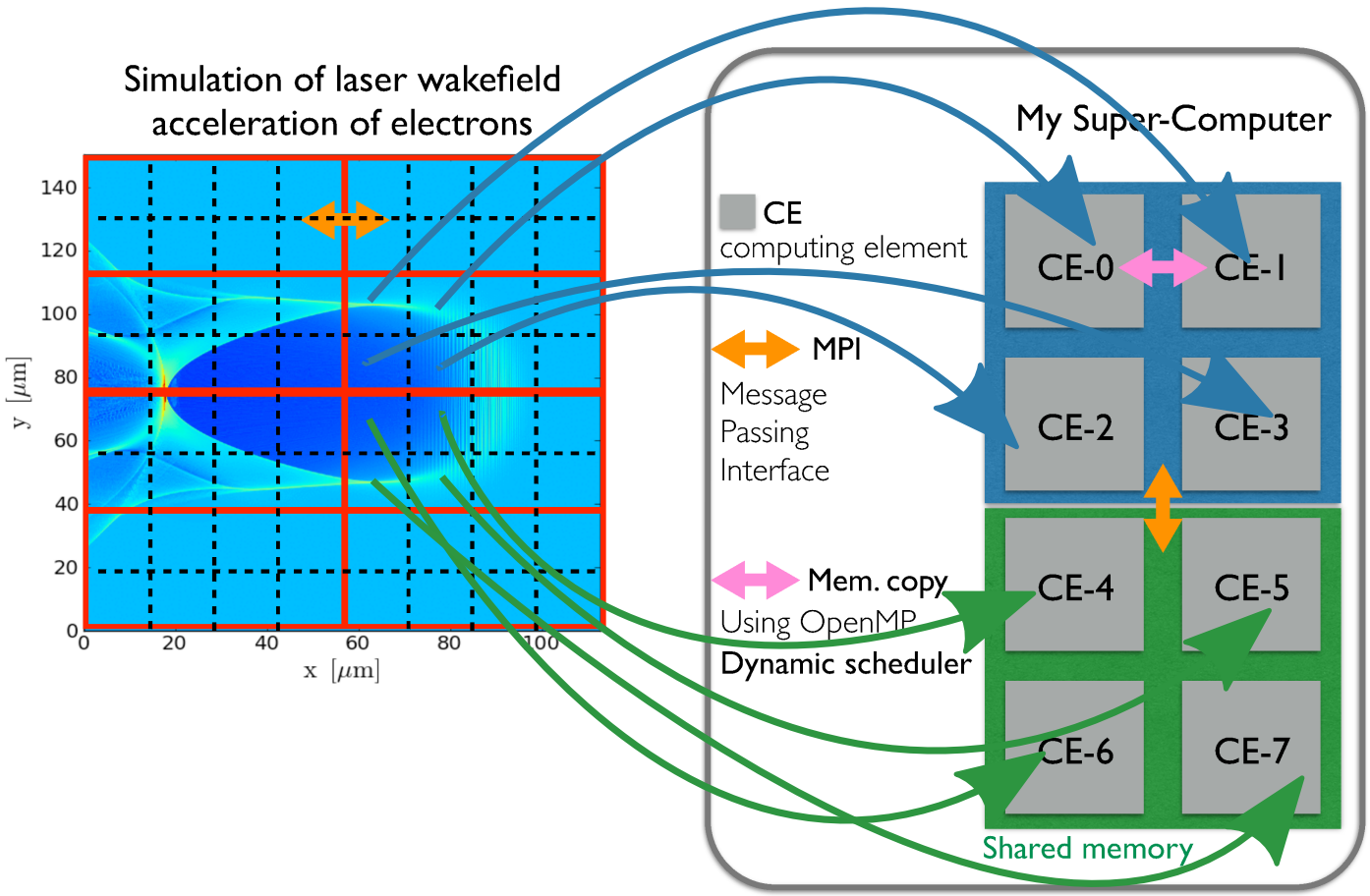
Shared memory

Domain decomposition : MPI

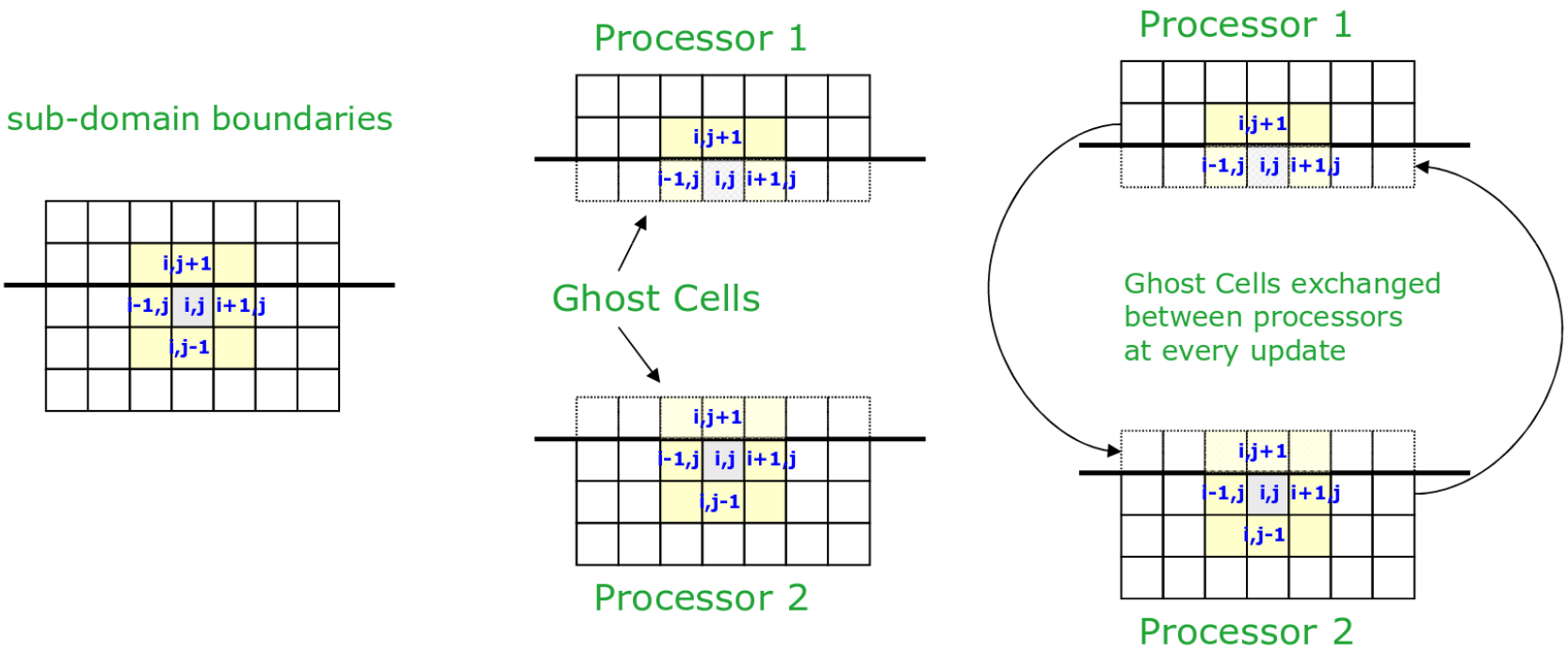


Domain decomposition : MPI + openMP in SMILEI

PIC code are 'easily' parallelized using domain decomposition + Patch

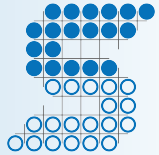


Domain synchronization



- If processors have a shared memory ==> OpenMP
- If processors have ditributed memory ==> MPI
- Same logic for particles

Message Passing Interface (MPI)



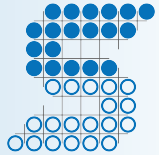
Characteristics

- Library
- Coarse grain
- Inter node
- Distributed memory
- **Almost all HPC codes**

Issues

- Latency
- OS jitter
- Global communication scalability

Open Multi-Threading (openMP)



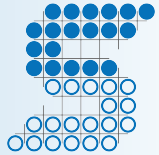
Characteristics

- Compiler Directives
- Medium grain
- Intra node
- Shared memory
- **Many HPC codes**

Issues

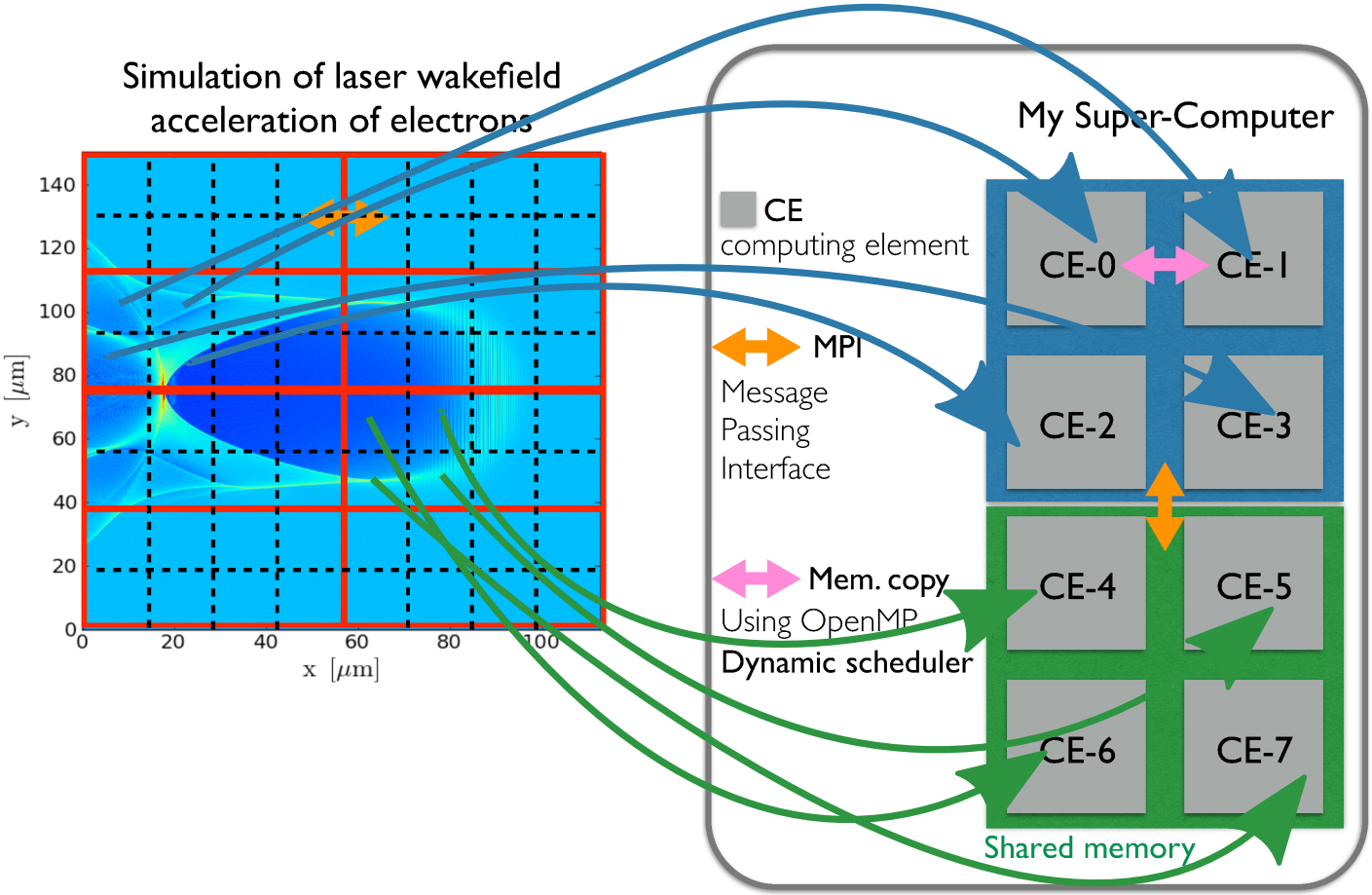
- Thread creation overhead
- Memory/core affinity
- Interface with MPI (*MPI_THREAD_MULTIPLE*)

Structure



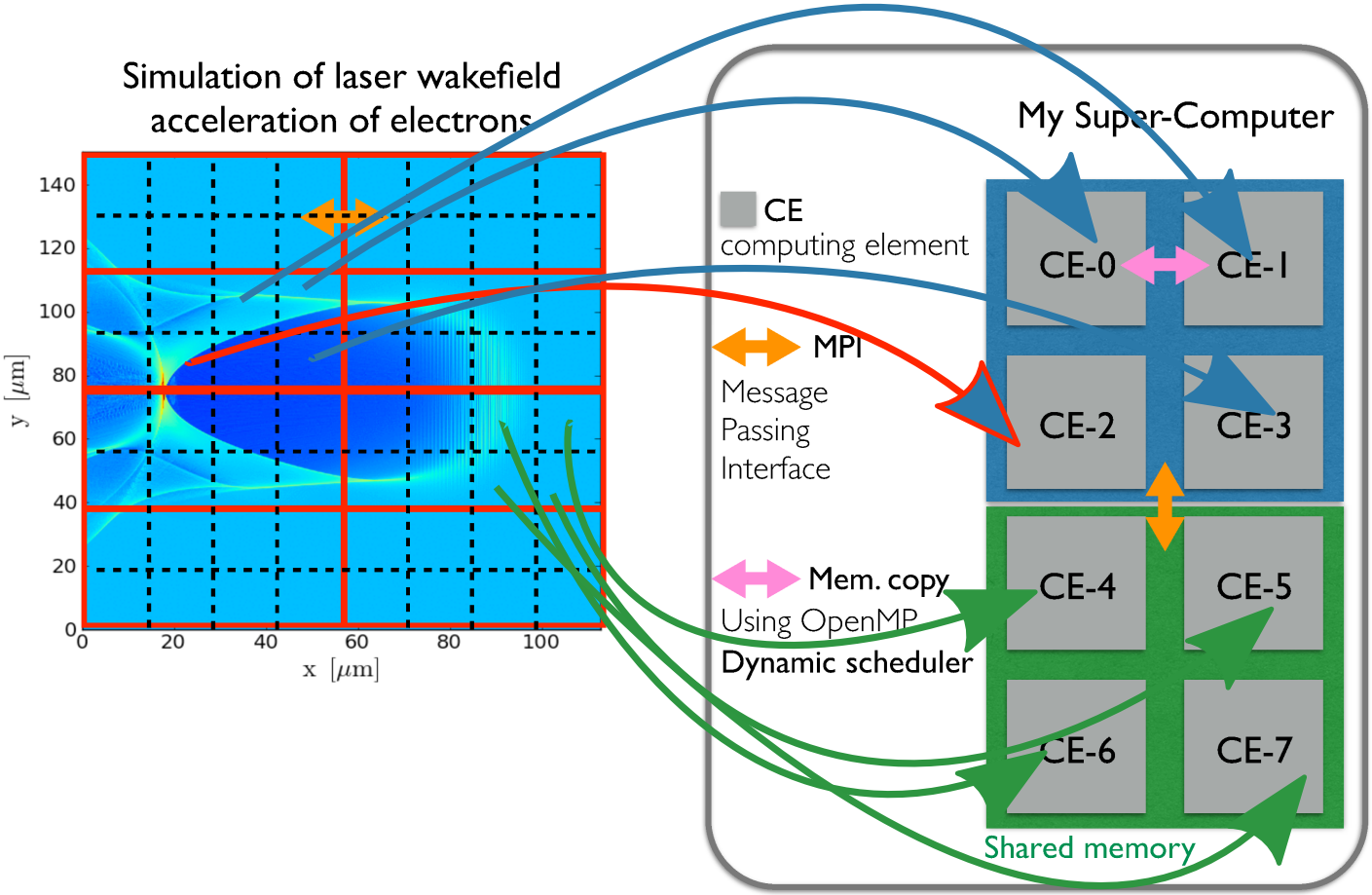
Domain decomposition : MPI + openMP in SMILEI

PIC code are 'easily' parallelized using domain decomposition



Domain decomposition : MPI + openMP in SMILEI

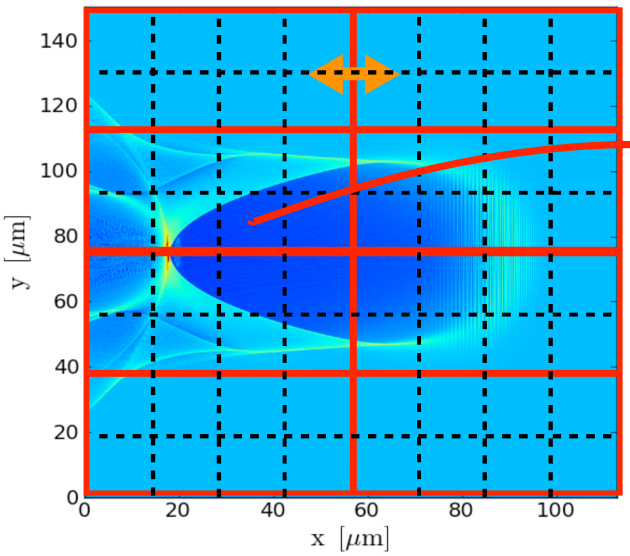
PIC code are 'easily' parallelized using domain decomposition

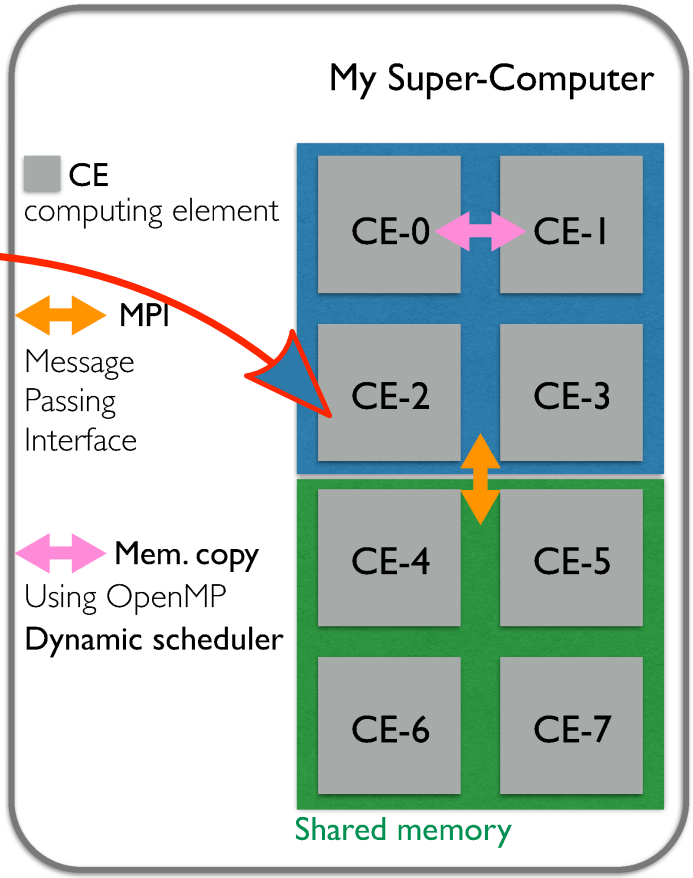


Domain decomposition : MPI + openMP in SMILEI

PIC code are 'easily' parallelized using domain decomposition

Simulation of laser wakefield acceleration of electrons



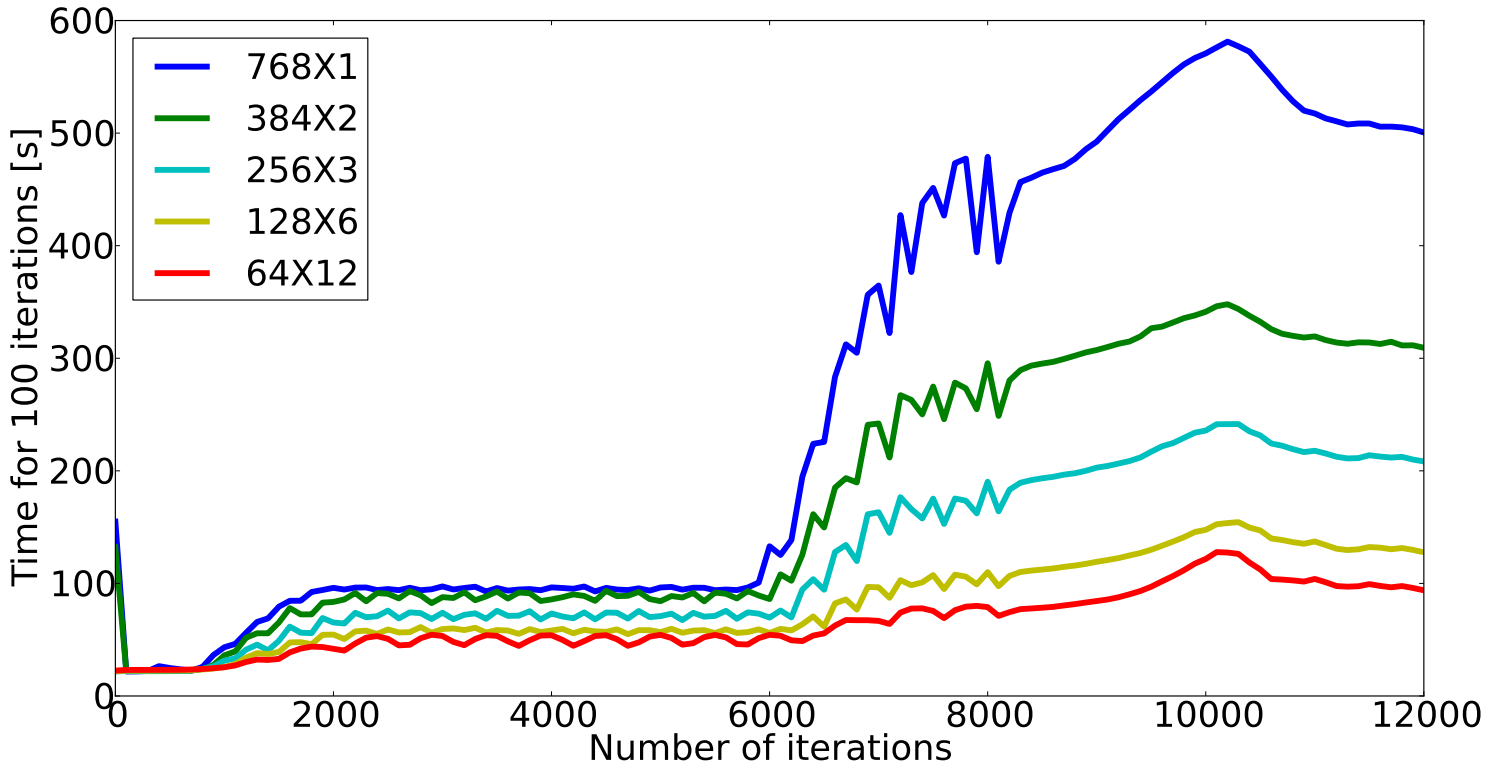


Domain decomp. is not enough!!!

- workload not optimally shared
- not adapted to new architectures!

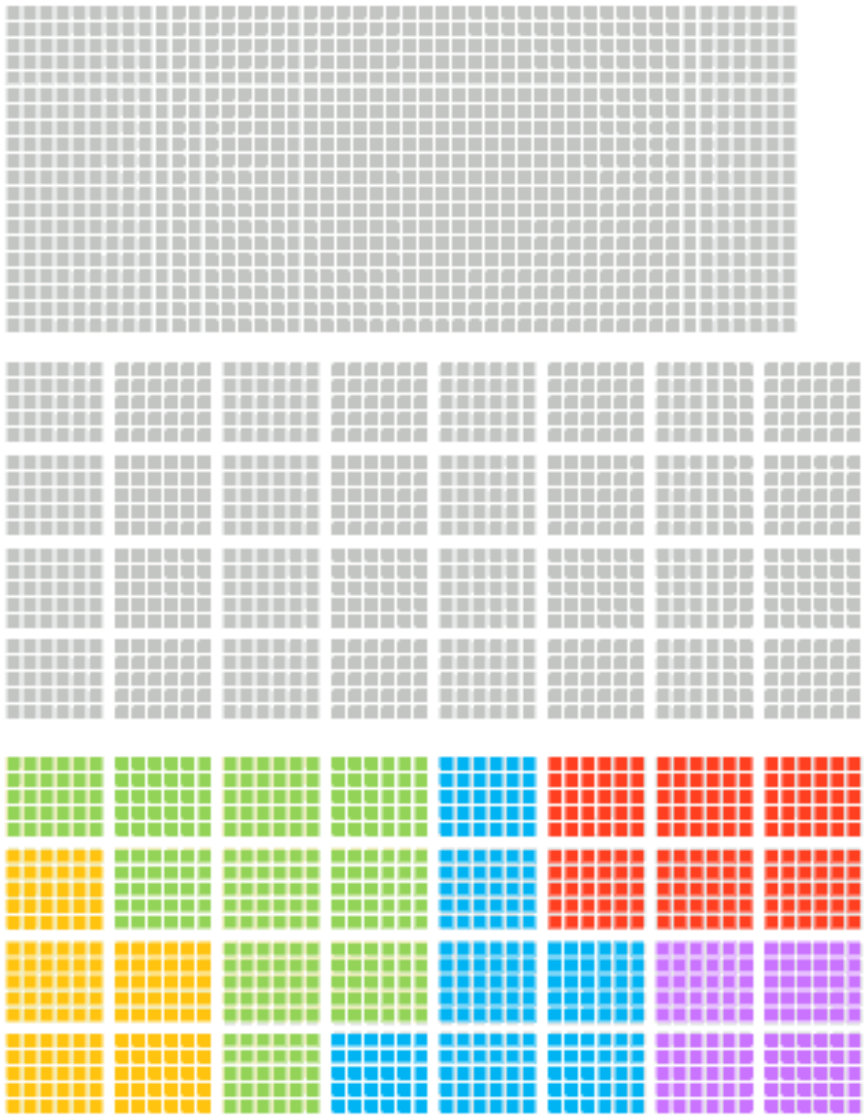
openMP dynamic scheduler benefits

MPI $\times$ *OpenMP*



OpenMP dynamic scheduler is able to smooth the load but only at the node level.

Patched base data structure

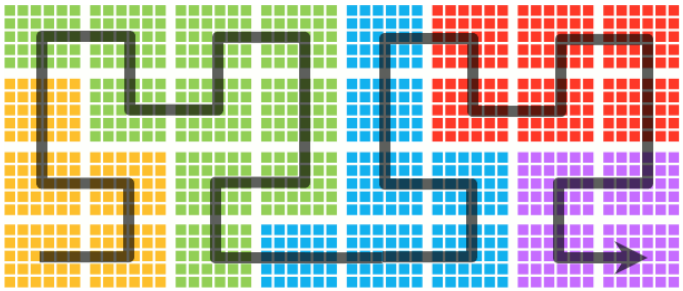


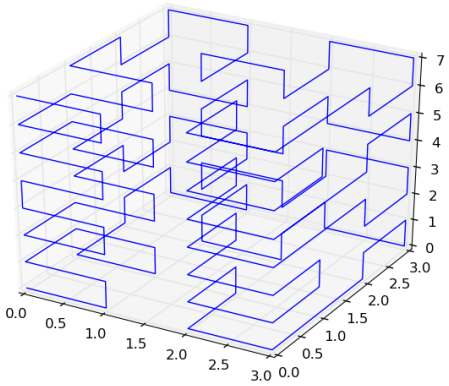
960 cells

32 patches

5 MPI regions

Hilbert ordering

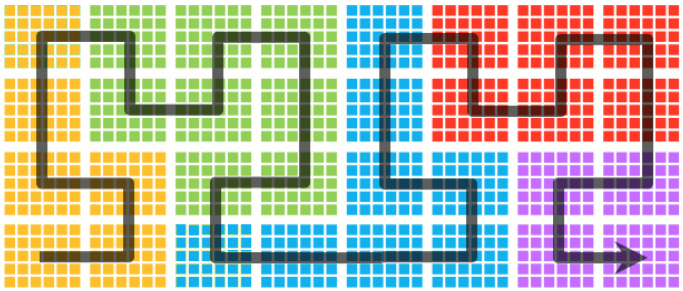


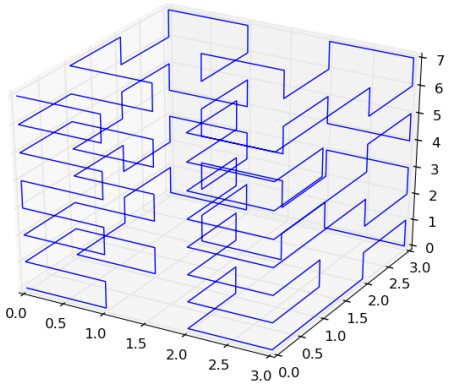


We need a policy to assign patches to MPI processes. To do so, patches are organized along a one dimensional space-filling curve.

1. Continuous curve which goes across all patches.
2. Each patch is visited only once.
3. Two consecutive patches are neighbours.
4. In addition we want compactness !

Hilbert ordering



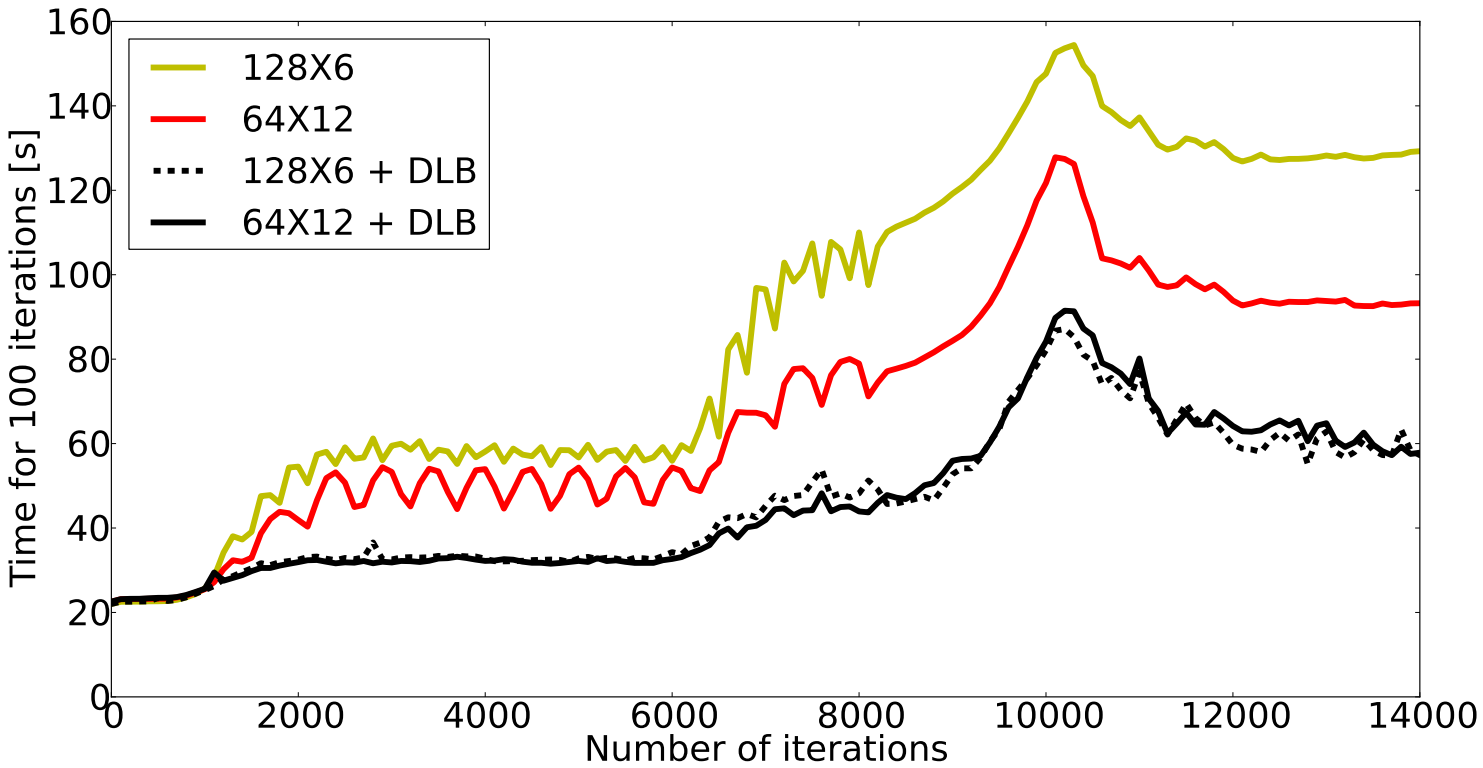


We need a policy to assign patches to MPI processes. To do so, patches are organized along a one dimensional space-filling curve.

1. Continuous curve which goes across all patches.
2. Each patch is visited only once.
3. Two consecutive patches are neighbours.
4. In addition we want compactness !

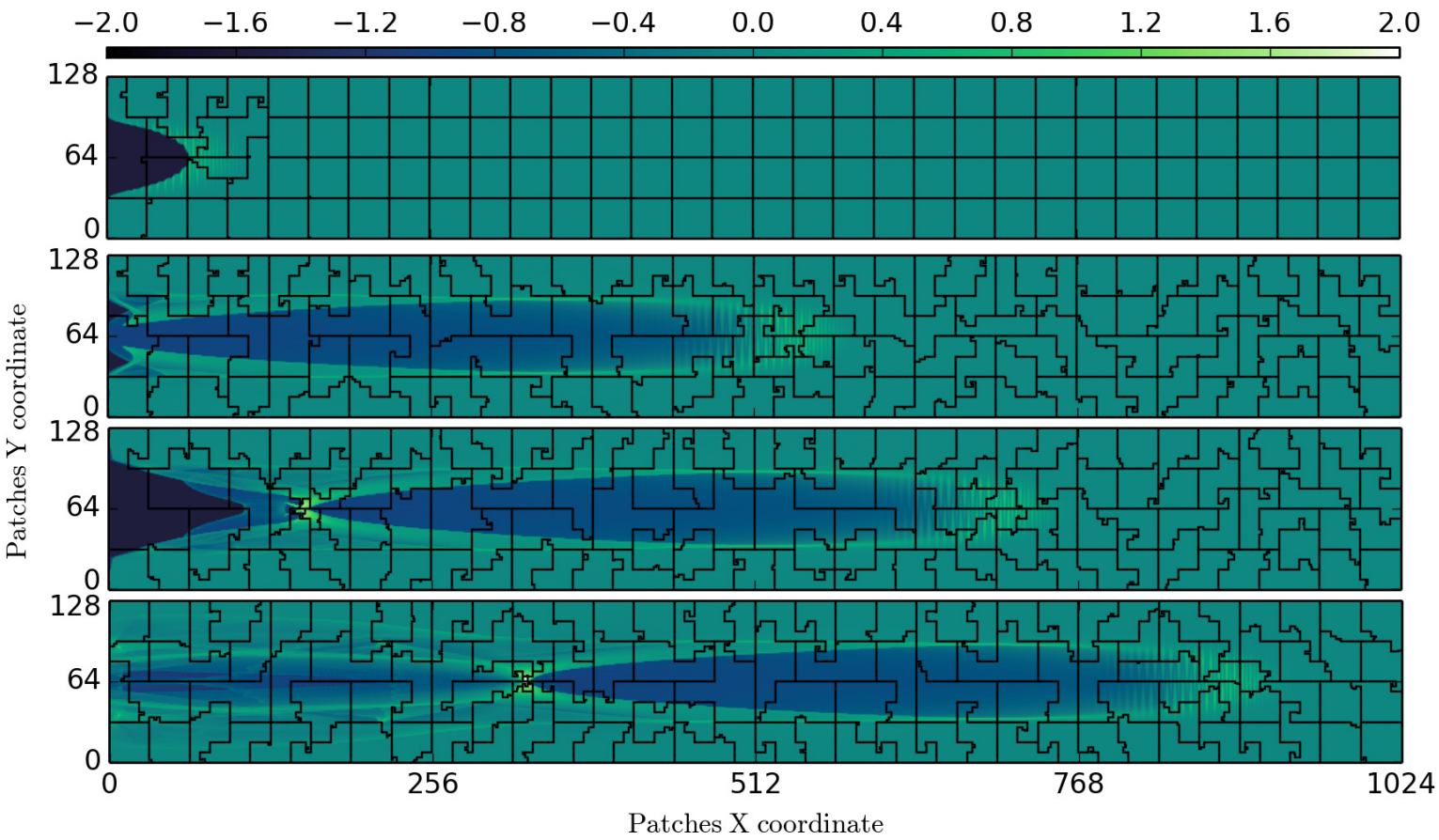
With dynamic load balancing activated

MPI $\times$ *OpenMP*



Yellow and red are copied from previous figure.

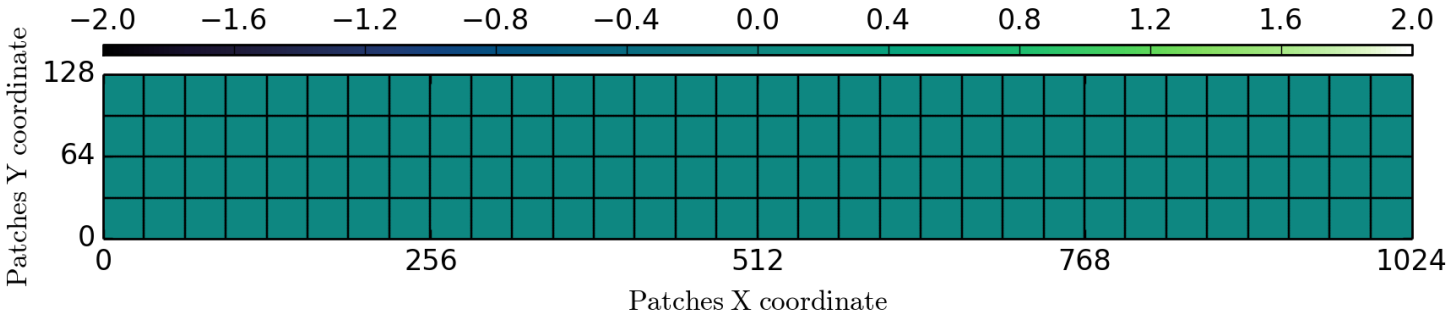
Dynamic evolution of MPI domains



Color represents the local patch computational load imbalance

$$I_{loc} = \log_{10}\left(L_{loc}/L_{av}\right)$$

Dynamic evolution of MPI domains



Color represents the local patch computational load imbalance

$$I_{loc} = \log_{10}\left(L_{loc}/L_{av}\right)$$