EPFL

LFMI

Parallel computation

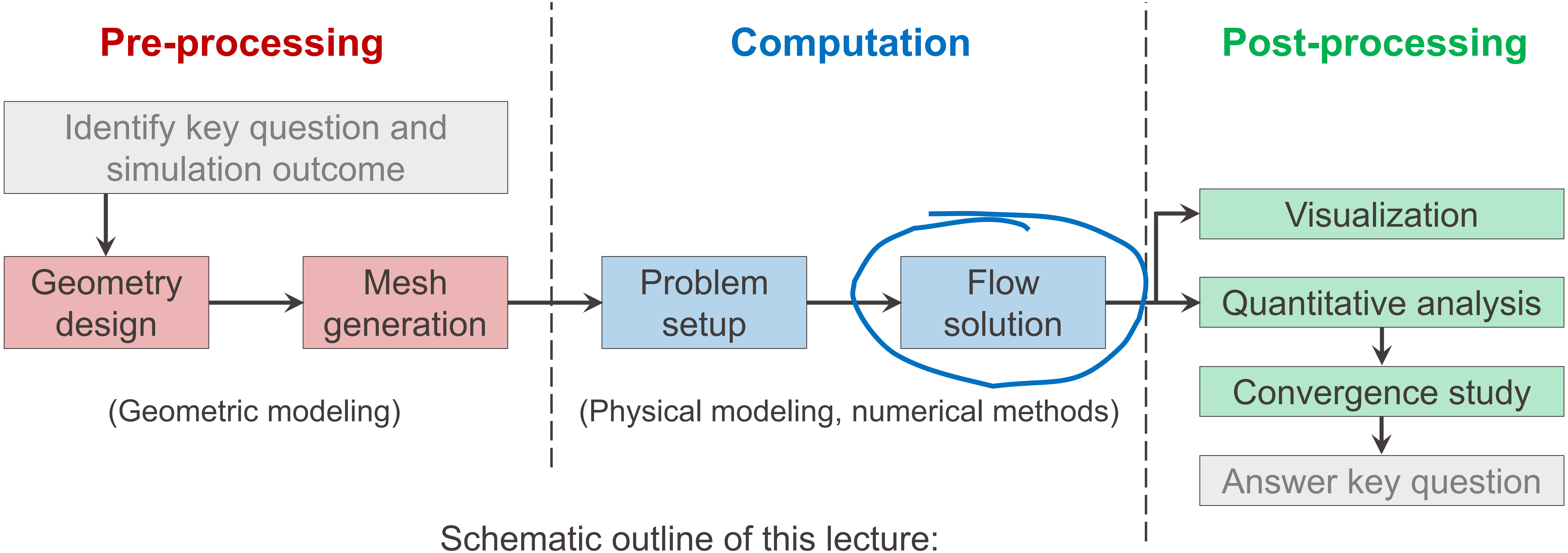
Numerical Flow Simulation

Edouard Boujo

Fall 2022

École
polytechnique
fédérale
de Lausanne

Numerical simulation workflow



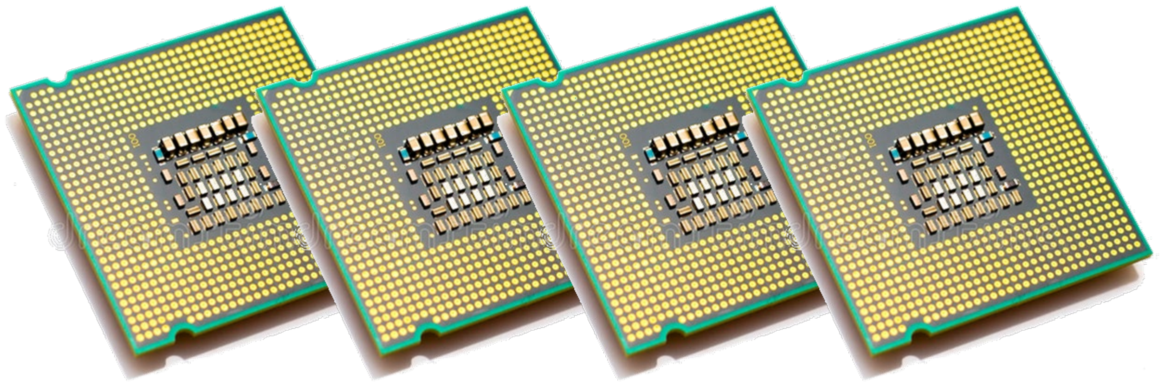
Schematic outline of this lecture:

1. Why parallel CFD?
2. Implementation in Fluent
3. Mesh partitioning
4. Parallel performance

Why parallel CFD?

- Basic idea: distribute the calculations among several processors that work simultaneously → more memory available & shorter calculation time.

- Perform larger simulations
  - Larger meshes
  - More complex physics

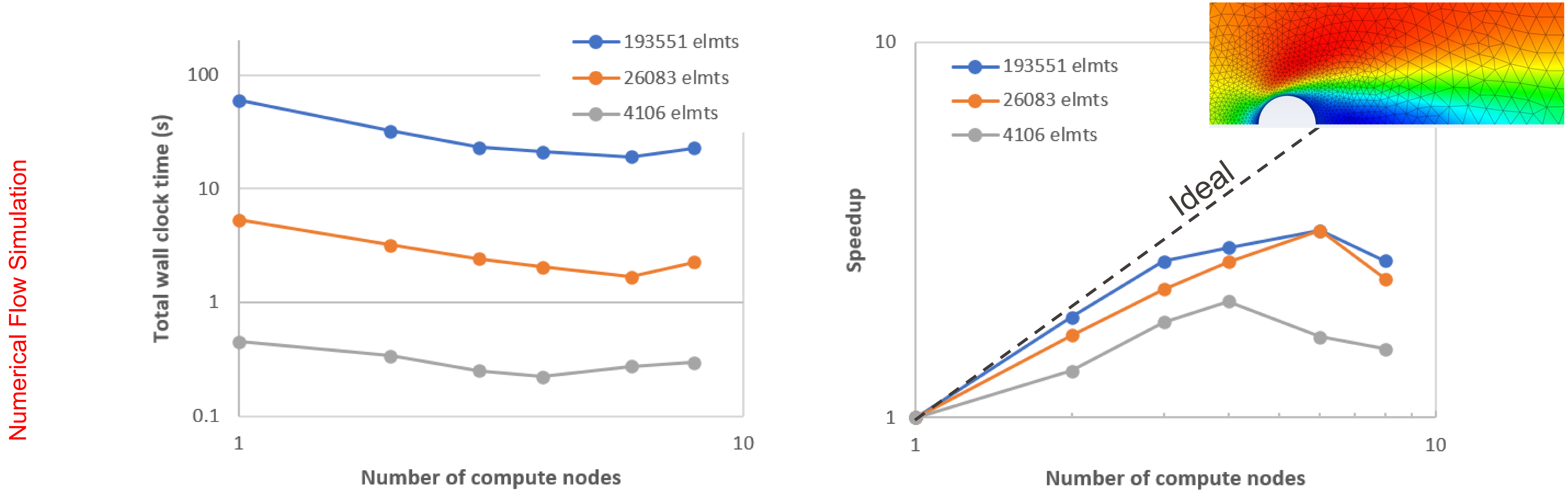


- Perform simulations faster
  - Many designs (virtual prototyping).
  - Shorter turn-around time (e.g. from days to hours, for several-million cell meshes)

- Take advantage of state-of-the-art high-performance computers
  - Fast computation
  - Large memory

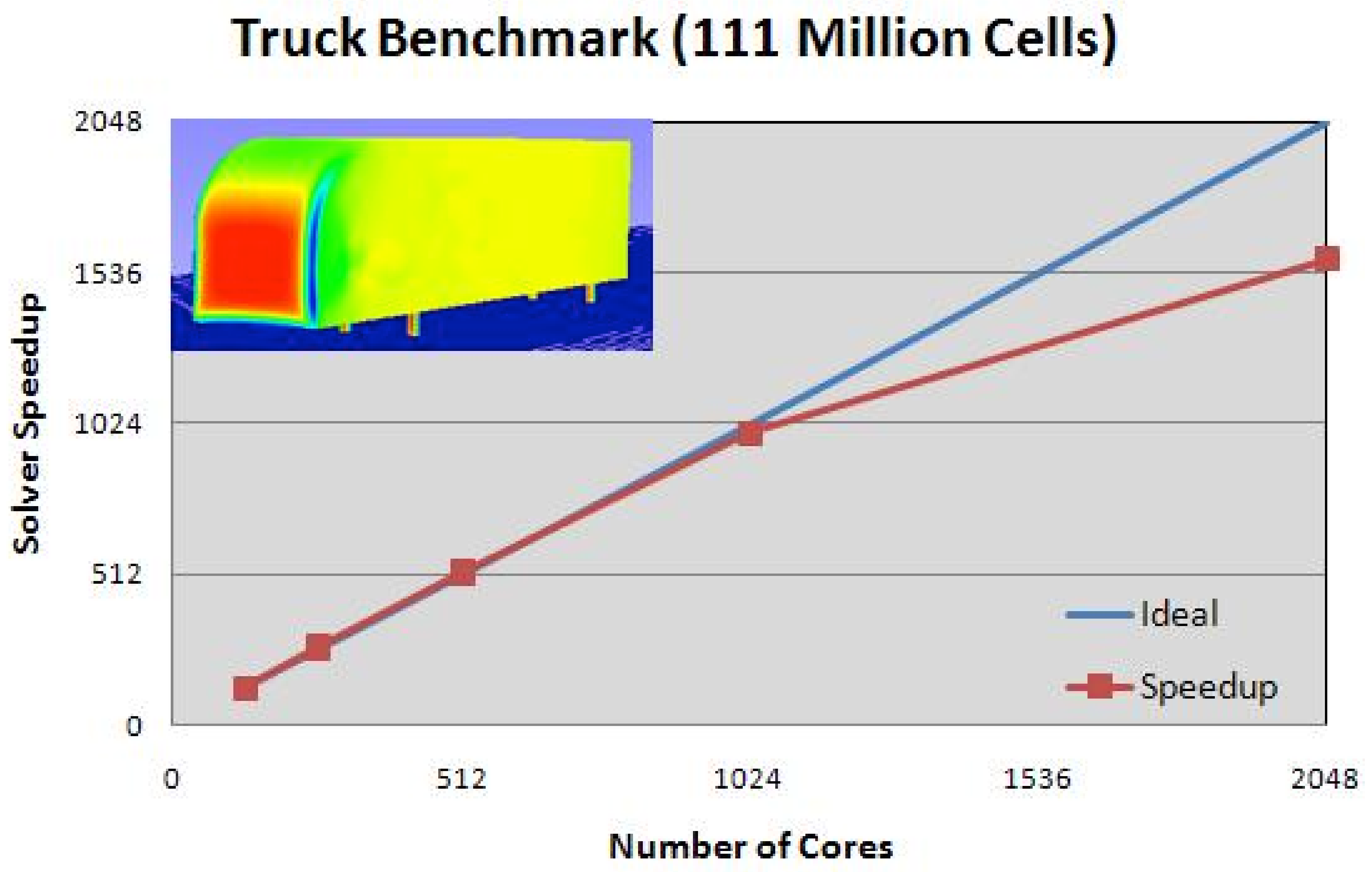
Why parallel CFD?

- When should parallel computation NOT be used?
  - When the problem is too small: too little work per processor and too much communication between processors → inefficient (or even slower!).



Why parallel CFD?

- When should parallel computation NOT be used?
  - When the problem is too small: too little work per processor and too much communication between processors → inefficient (or even slower!).

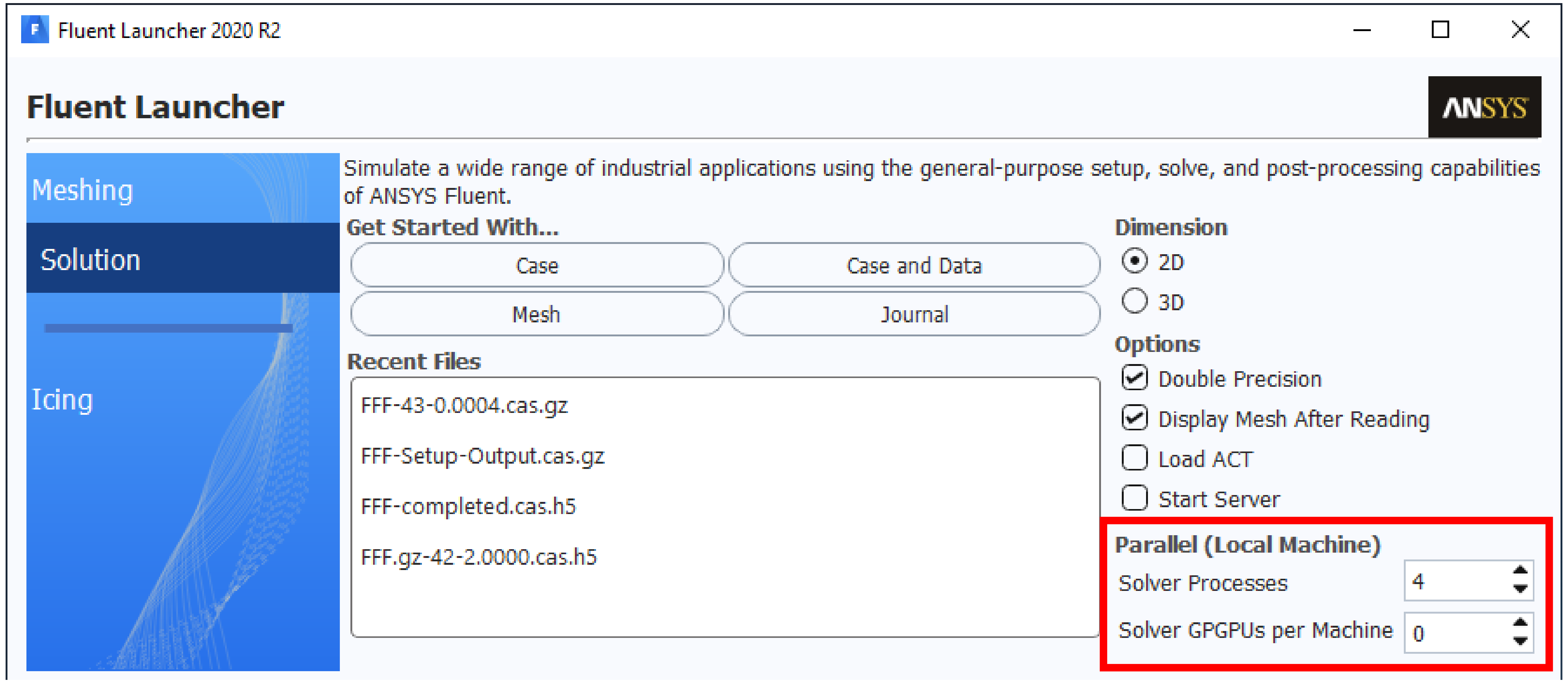


Why parallel CFD?

- When should parallel computation NOT be used?
  - When the problem is too small: too little work per processor and too much communication between processors → inefficient (or even slower!).
  - When the computer resources are not appropriate (e.g. insufficient communication bandwidth) → must check parallel performance (see later).
  - When the user doesn't know what they are doing...

Implementation in Fluent

- Assumption:
  - Users of commercial software are interested in shortening turn-around time, not in parallel computing itself.
  - May want to use software on different serial/parallel computer systems.
- Implementation:
  - As “transparent” as possible: detailed knowledge of parallel computing not required.
  - The code should be highly portable.



Basically, the user only has to choose the number of processors.

Implementation in Fluent

- Memory model: supports both shared-memory and distributed-memory parallel computers (see later).

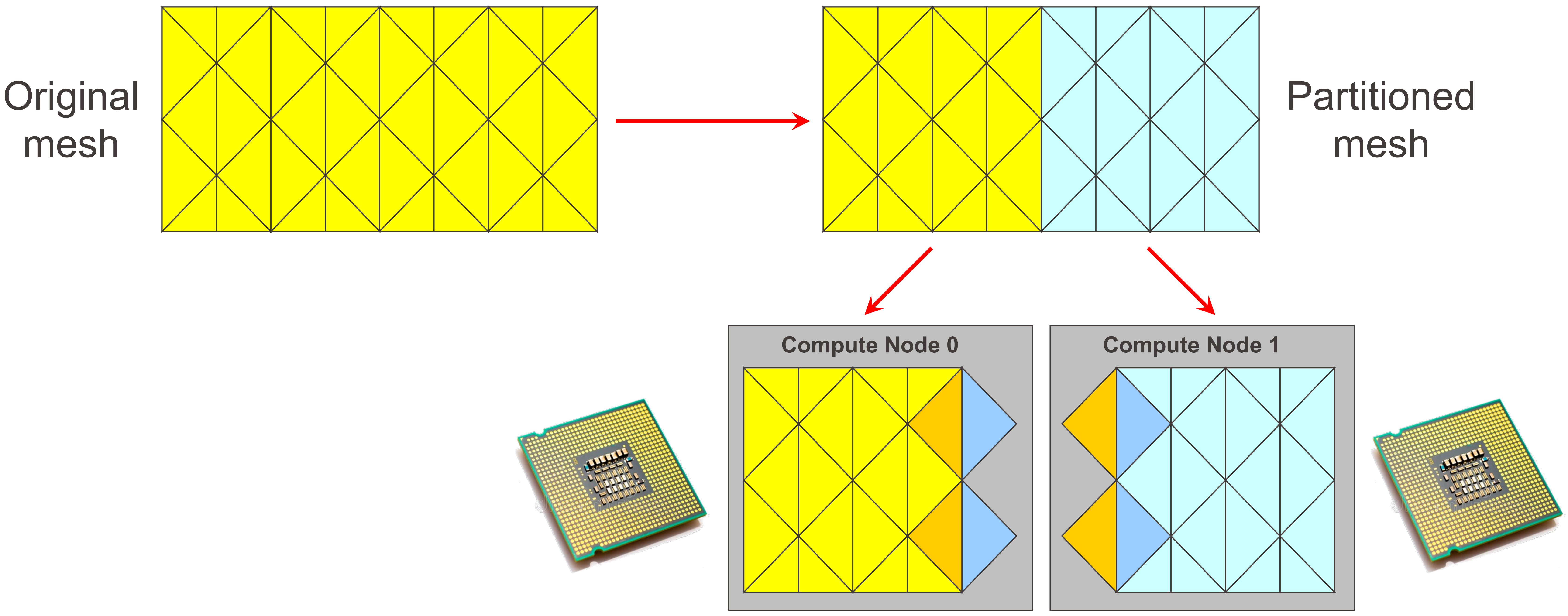
- Based on domain decomposition: computational mesh partitioned into sub-meshes, each partition assigned to a different compute node (see later).

- Communication between processors: message passing
  - Scalable
  - Complete control over communication for better performance optimization
  - MPI standard has emerged with wide support

- Client/server model: separation of front-end tasks (GUI, visualization) and solver.

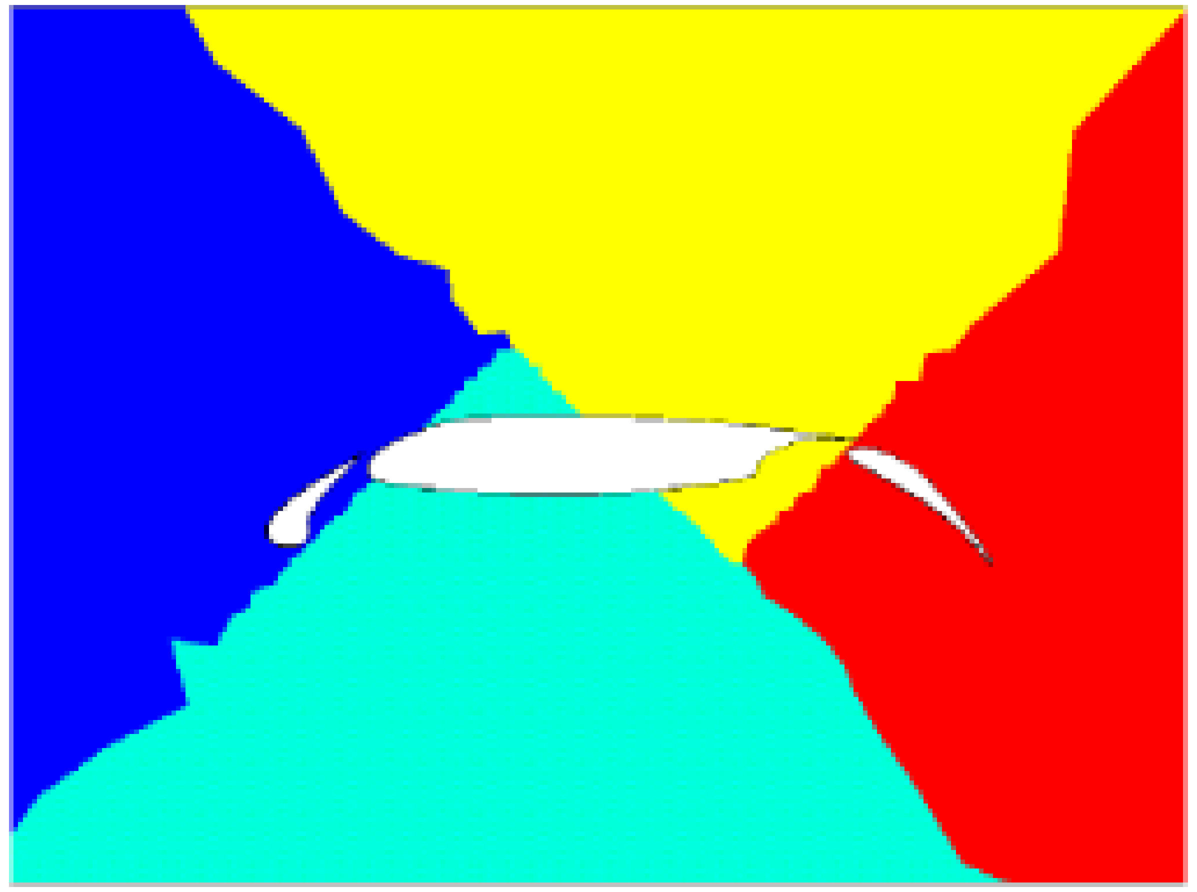
Mesh partitioning

- Basic concept: computational mesh automatically partitioned into several sub-meshes (partitions), each partition assigned to a different processor/node that solves the equations only in its own sub-mesh.



Distributed sub-meshes (with "ghost cells" to exchange information about neighboring cells)

Mesh partitioning

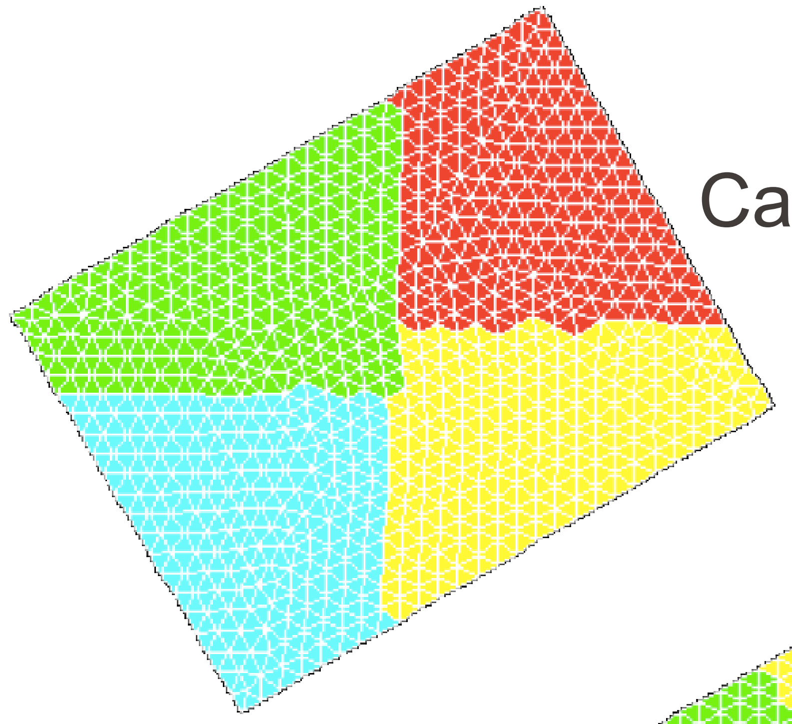


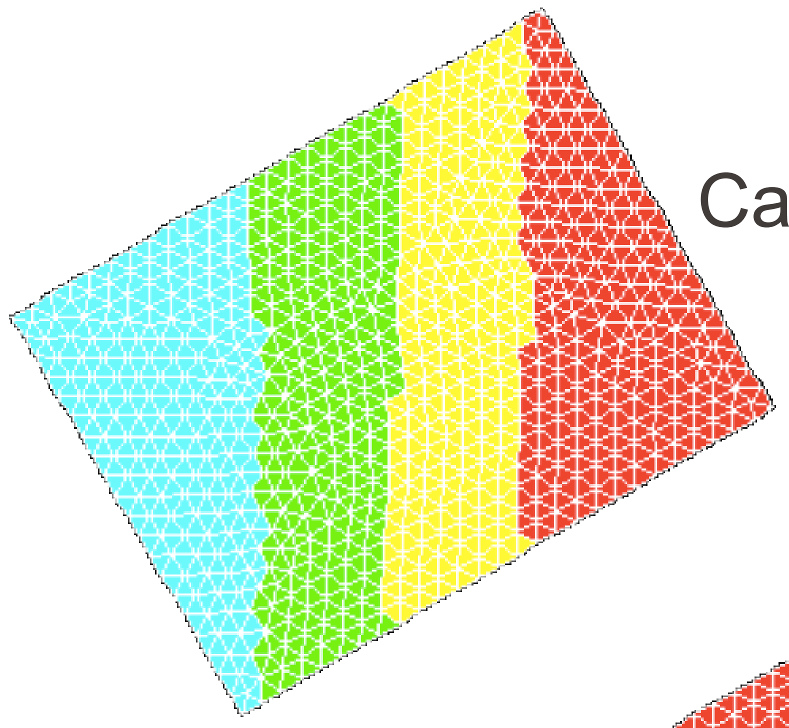
- Wide range of algorithms:
  - Coordinate bisection,
  - Recursive spectral bisection,
  - Multi-level graph partitioning…

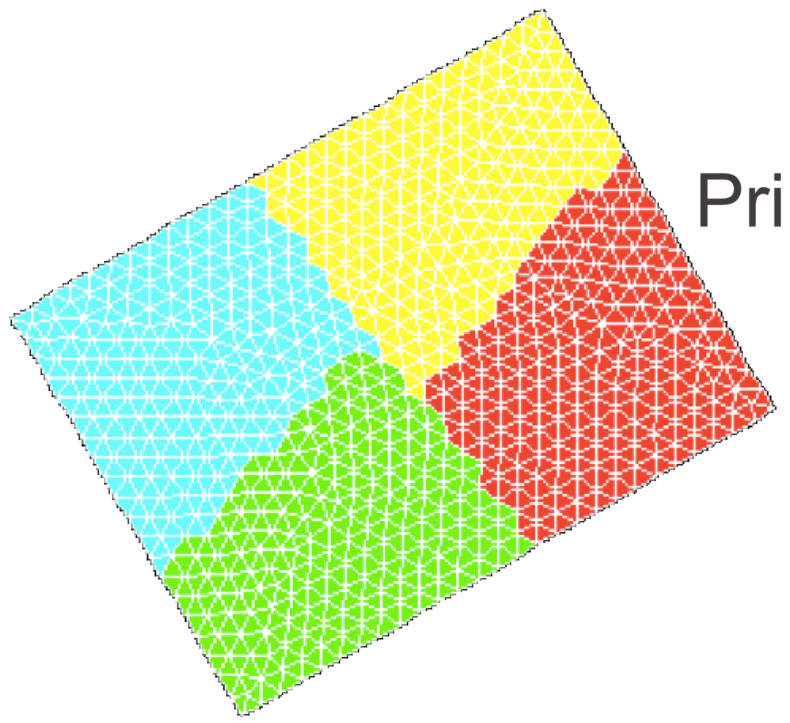
- Goals for optimal algorithm:
  - Partitions with similar number of cells (similar load on each processor)
  - Simply-connected partitions (no holes)
  - Minimize data transfer (interface boundaries with short length and small number of faces)

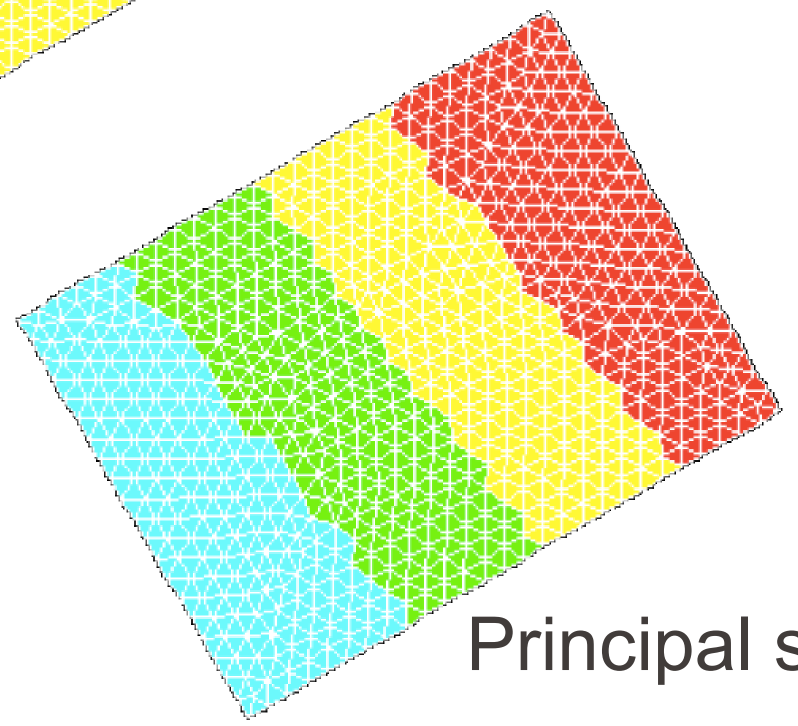
Mesh partitioning

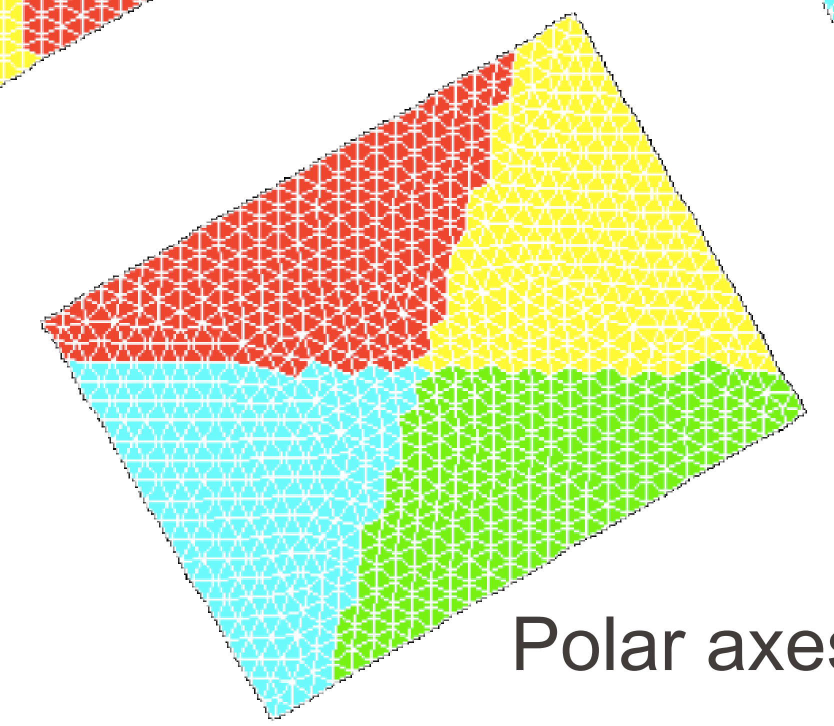
- Fluent can use the following algorithms:
  - Cartesian axes; Cartesian *x,y,z;* Cartesian strip
  - Principal axes (default); principal *x,y,z;* principal strip
  - Polar axes; polar r,θ (only in 2D)
  - Spherical axes; spherical ρ,θ,φ (only in 3D)
  - METIS (multi-level graph partitioning)


Cartesian axes

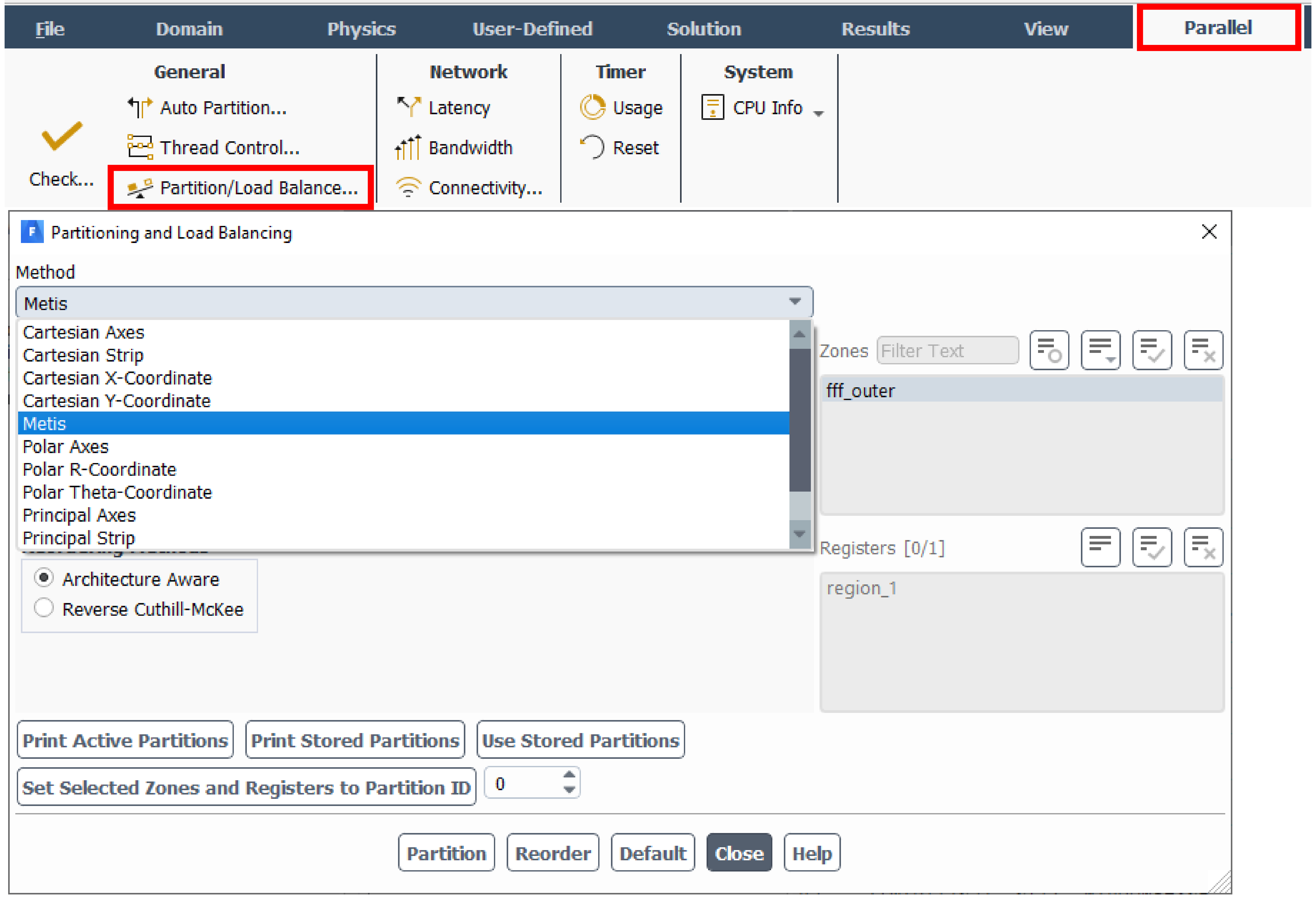

Cartesian *x*


Principal axes


Principal strip


Polar axes

Mesh partitioning

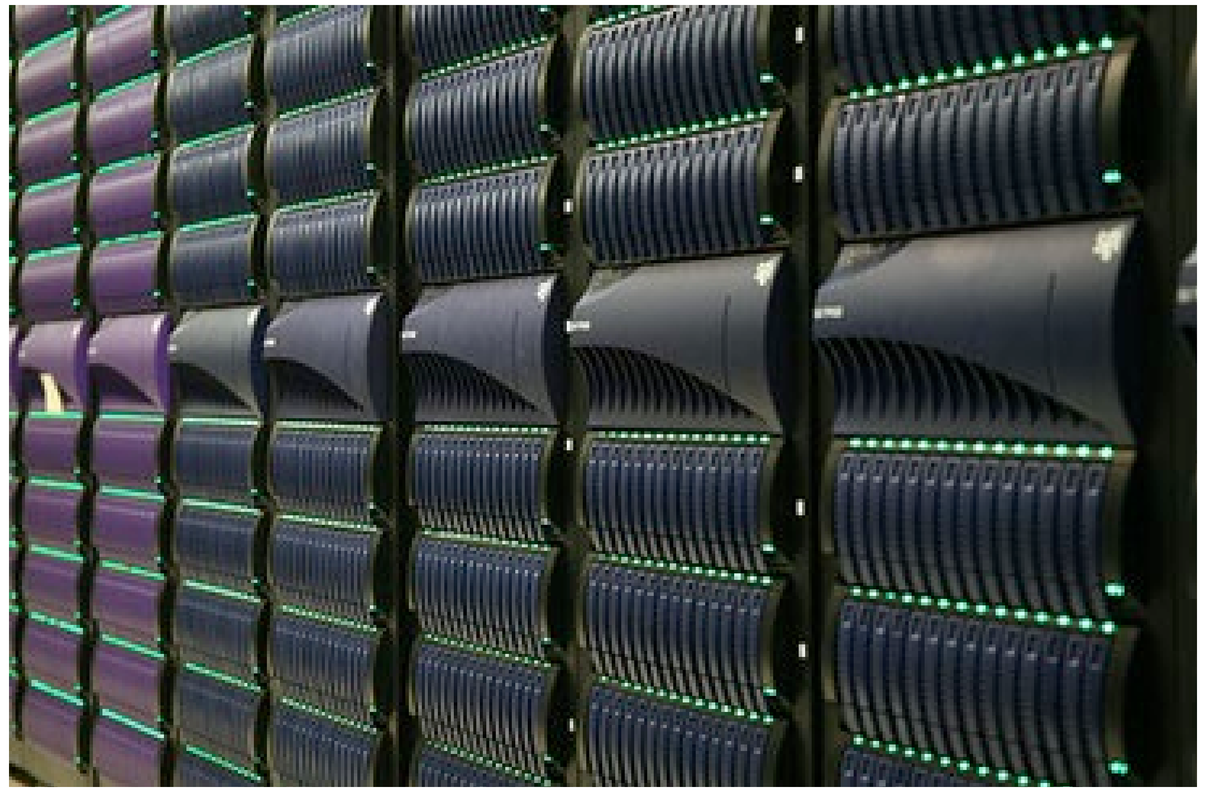


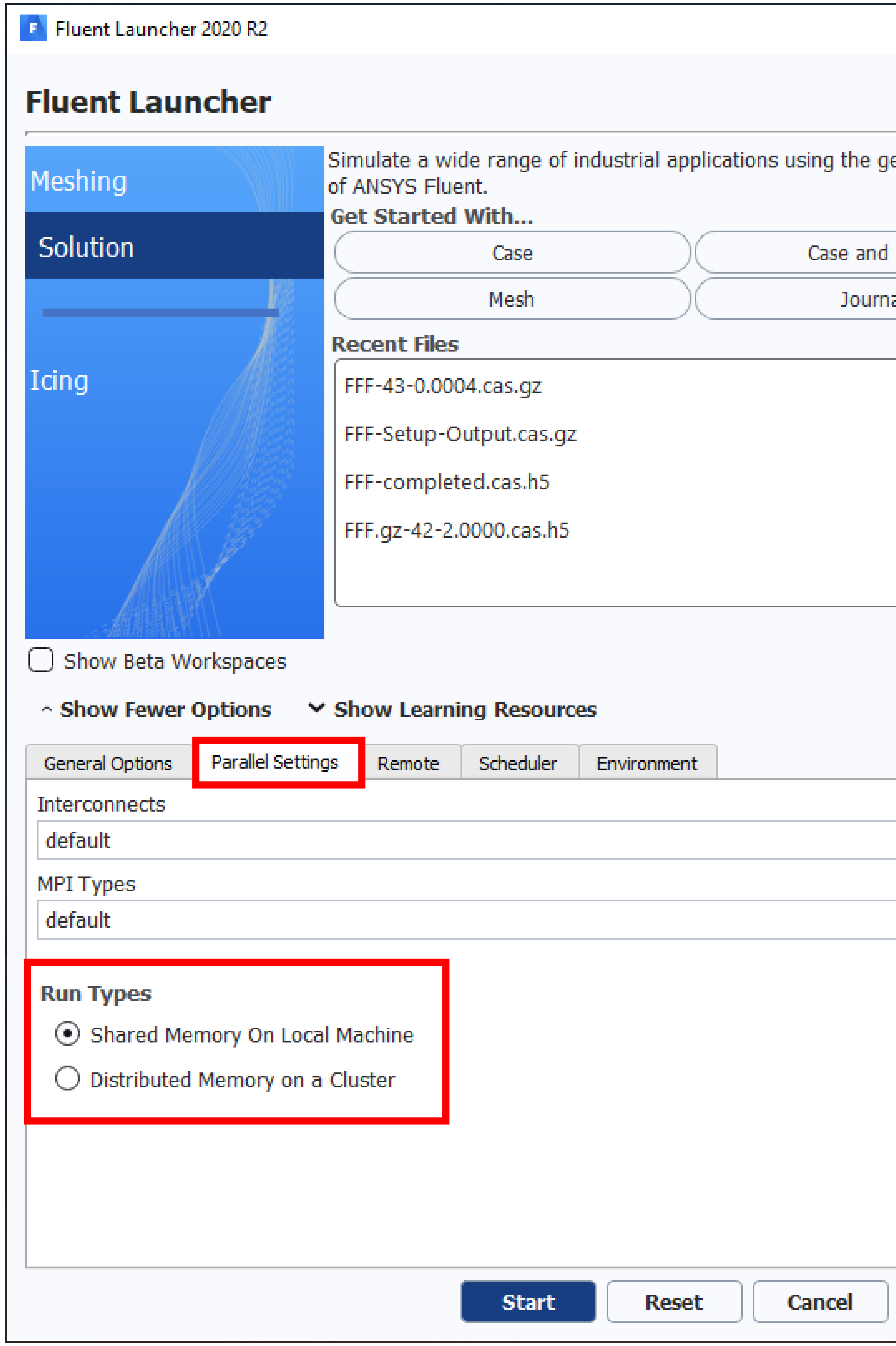
Architecture

- Fluent can run parallel calculations on 2 types of architecture:

  1. **Shared memory**: **one machine** with several processors (from simple PC/laptop to high-performance workstations)

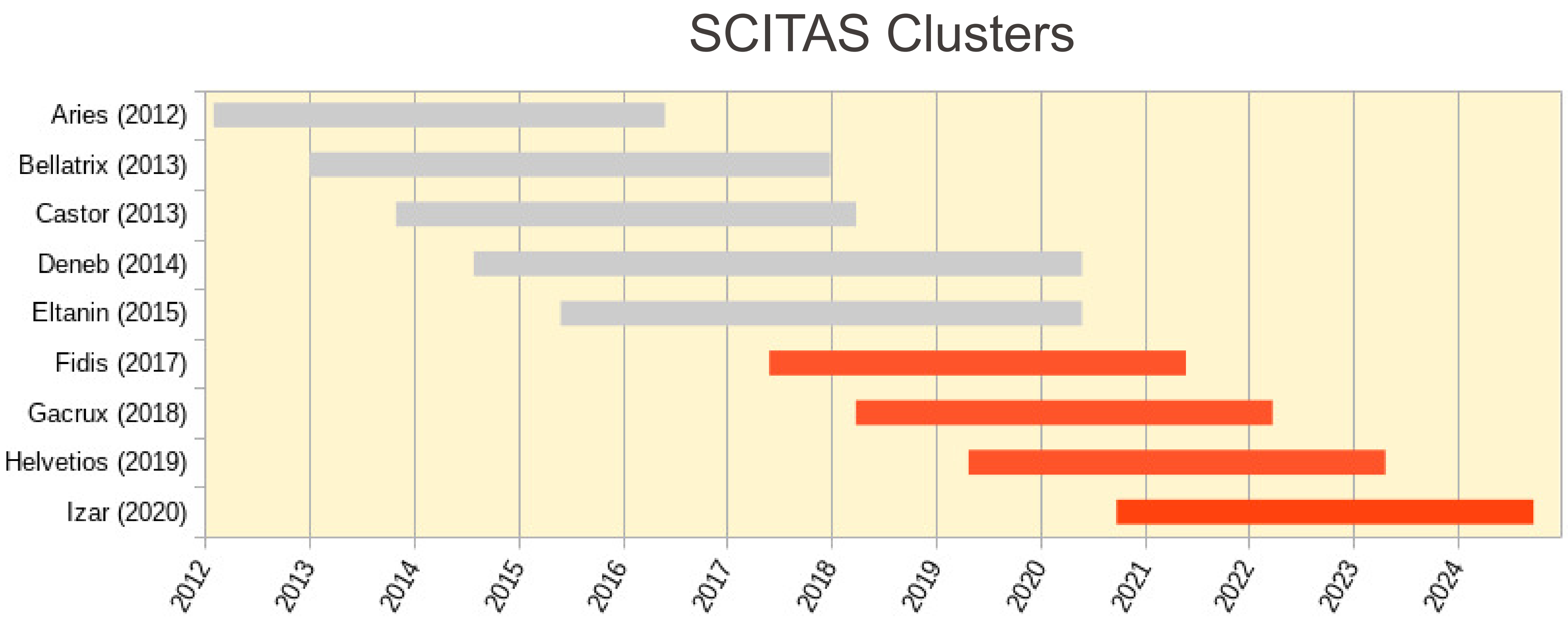
  2. **Distributed memory**: **cluster** of several machines (e.g. EPFL clusters)





EPFL SCITAS clusters

- Several high-performance clusters are available at EPFL.
- Some of them may be available for student projects.

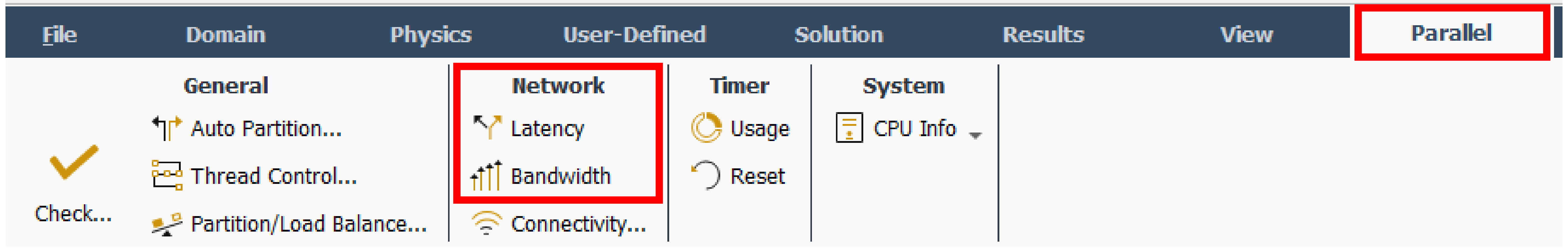


https://www.epfl.ch/research/facilities/scitas/

Parallel performance

- Two important characteristics of parallel communication:
  - **Bandwidth**: transmission speed = max. amount of data sent in given time [MB/s]
  - **Latency**: time to send data = delay [ms] or [μs]

- Fluent can measure bandwidth and latency between processors.



Bandwidth (MB/s) with 5 messages of size 4MB

ID	n0	n1	n2	n3	n4	n5
n0		111.8	*55	111.8	97.5	101.3
n1	111.8		69.2	98.7	111.7	*51
n2	54.7	69.2		72.9	104.8	*45
n3	111.8	98.7	72.9		64.0	*45
n4	97.6	111.7	104.8	*64		76.9
n5	101.2	50.9	45.5	*45	76.9	

Latency (usec) with 1000 samples

ID	n0	n1	n2	n3	n4	n5
n0		48.0	48.2	48.2	48.3	*50
n1	48.0		48.2	48.3	48.3	*48
n2	48.2	48.2		48.8	49.1	*53
n3	48.2	48.3	*49		48.6	48.5
n4	48.3	48.3	49.1	48.6		*50
n5	49.7	48.5	*53	48.5	49.7	

Parallel performance

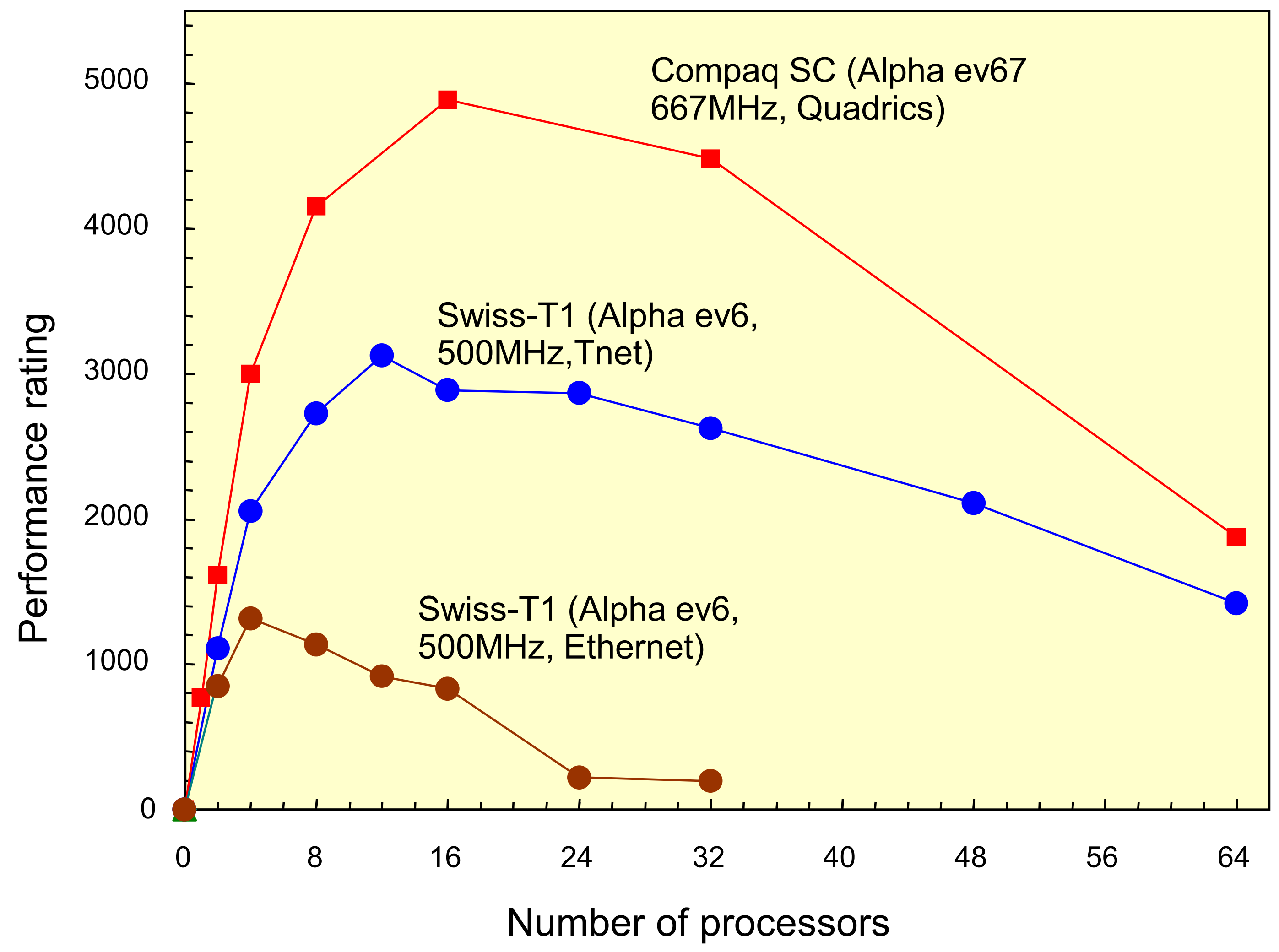
- Both bandwidth and latency influence parallel performance.
- Latency tends to have a greater effect (over a range of common problem sizes and available networks)
  - Scalability of Fluent largely determined by scalability of its linear equation solver. Multigrid limited by latency because requires several (small) message exchanges.
  - A low-latency interconnect can greatly improve performance scalability of Fluent.

- Performance also depends on:
  - Problem size
  - Physical modeling
  - Numerical method

Parallel performance

- Optimal use
  - For a given problem (size, physical modeling, numerical method):
    - Performance scales well for small number of processors
    - Performance will saturate (or decrease) for large number of processors
  - Optimal value of number of processors:
    - Increases with problem size
    - Increases (generally) with problem complexity
  - Choose wisely:
    - Parallel computer system (bandwidth & latency)
    - Number of processors

3D turbulent flow in a bend, 32’000 hex cells



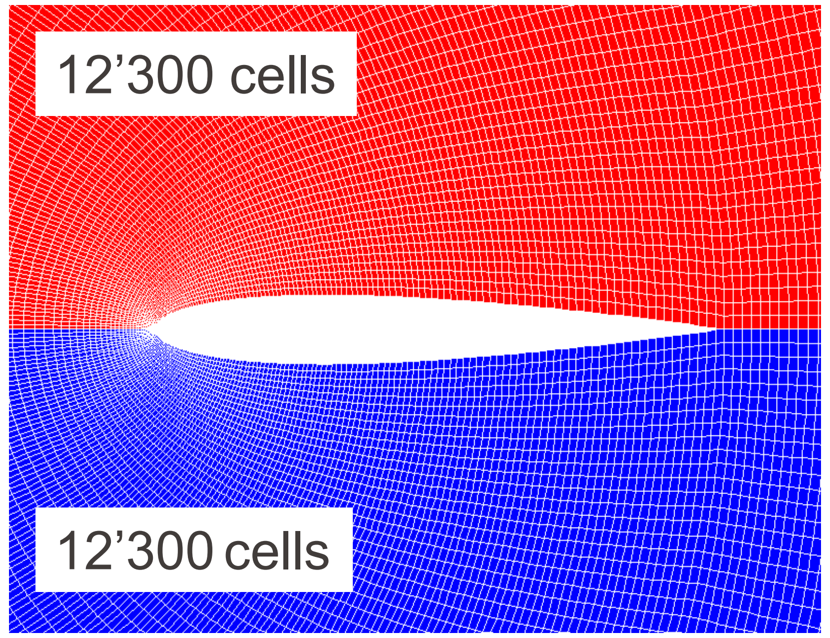
Numerical Flow Simulation

Parallel performance

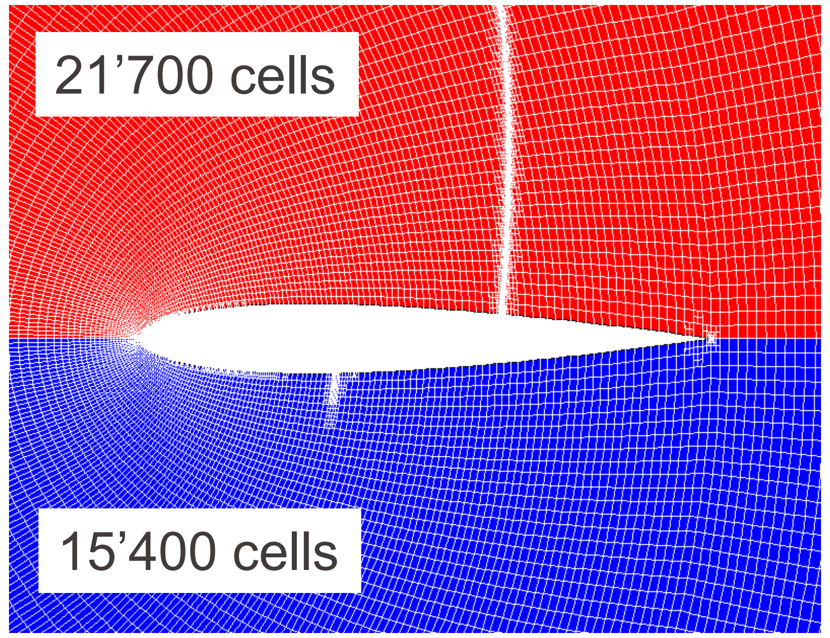
- Dynamic load balancing
  - Automatically redistributes the computational load more evenly among processors (cell migration from one processor's memory to another).
  - Load imbalance can result from:
    - Change in performance of some processors due to other processes
    - Non-uniform network performance
    - Changes in mesh distribution from refinement/coarsening
  - Associated with performance penalty (disabled by default in Fluent, must be explicitly enabled if desired).

Parallel performance

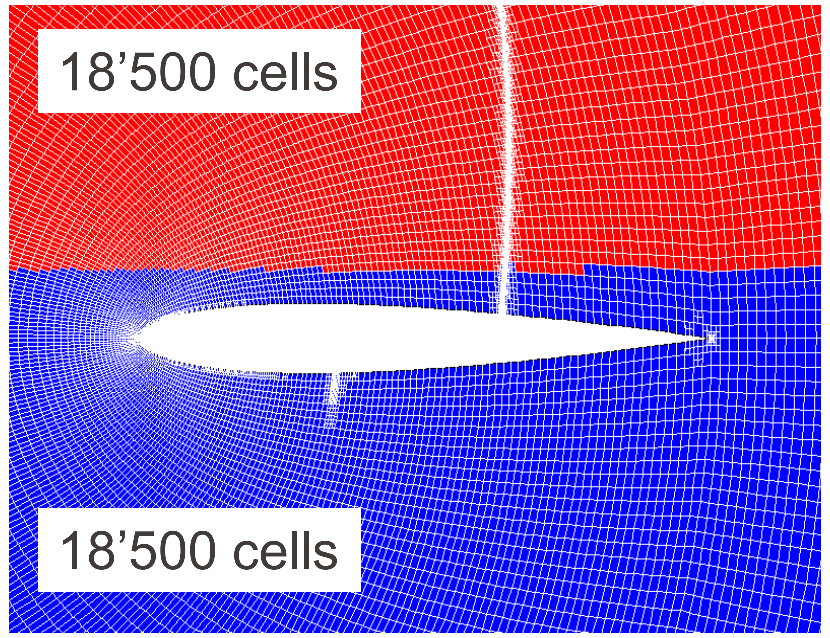
- Dynamic load balancing: example (transonic flow over a NACA 0012 profile)



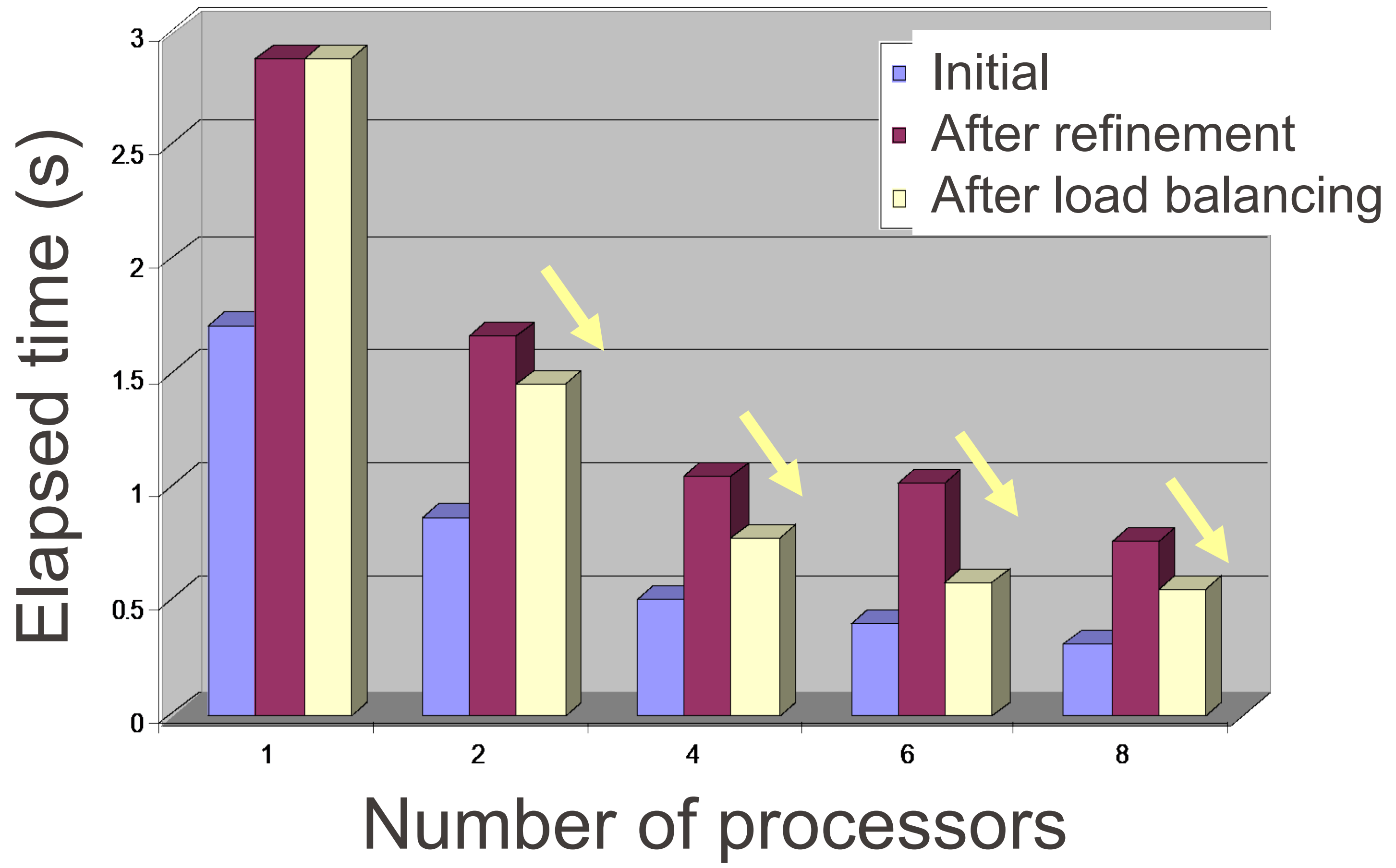
Initial mesh
(here on 2 partitions)



Refined mesh
→ imbalance



Refined mesh
after load balancing



Summary

- Parallel computation allows "large" problems to be solved:
  - Very fine mesh
  - Long-time behavior
  - Complex physics
- Network is characterized by bandwidth (transmission rate) & latency (delay)
- Performance of parallel computation depends on:
  - Numerical problem
  - Network properties
  - Compute node properties

- To learn more: course "Parallel and high-performance computing" (MATH-454)
- Remember: numerical simulation has a cost and an energy footprint
  → only run simulations that are needed!