Theory and Practice of Finite Element Methods

MPI Parallelisation

Luca Heltai <luca.heltai@sissa.it>

International School for Advanced Studies (www.sissa.it)
Mathematical Analysis, Modeling, and Applications (math.sissa.it)
Master in High Performance Computing (www.mhpc.it)
SISSA mathLab (mathlab.sissa.it)

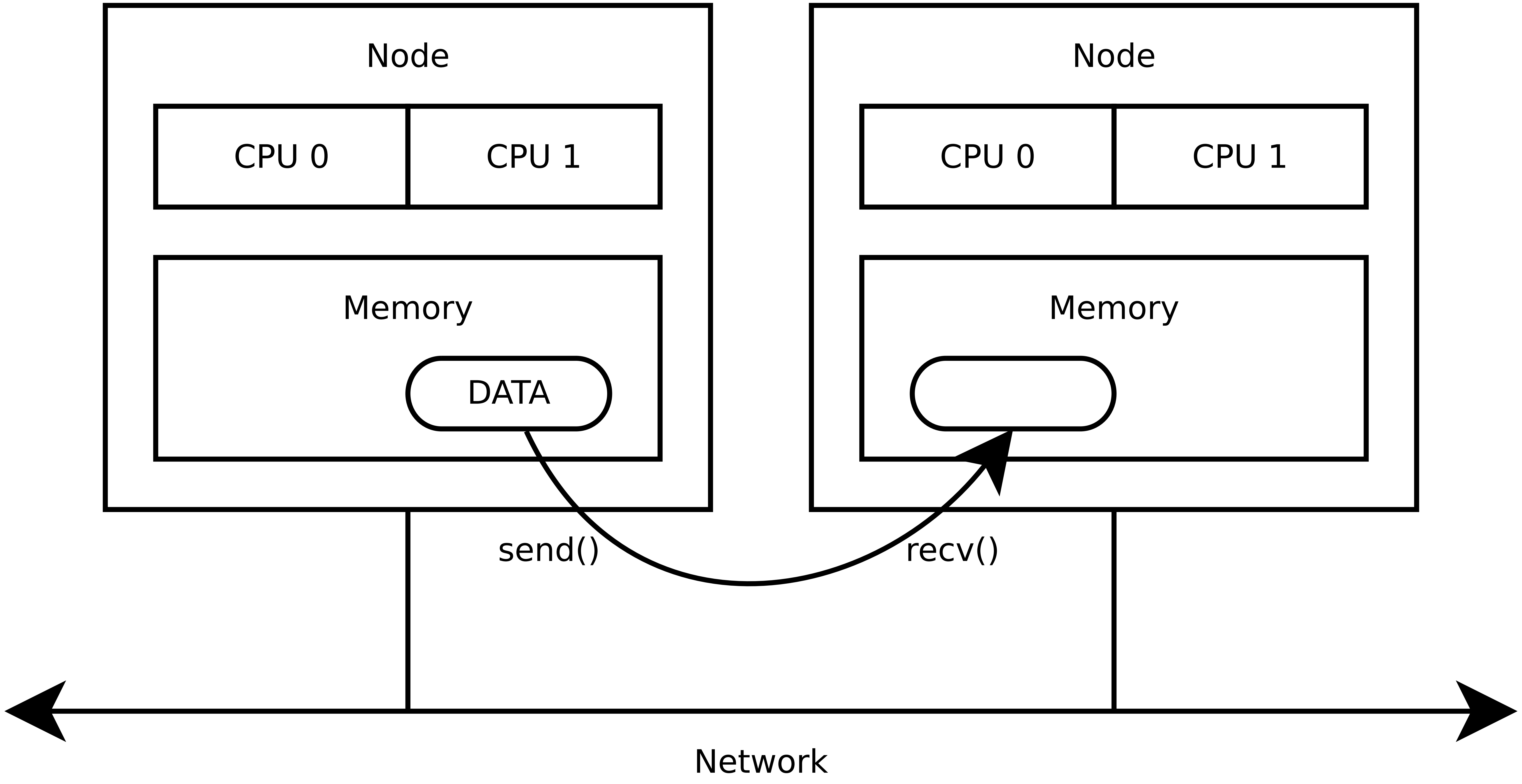
Aims for this module

- First introduction into parallel computing with deal.II using MPI
- Parallel distribution of degrees-of-freedom
  - Ownership concepts
- Setup data structures for parallel processing
- Assembly in parallel
- Synchronisation of distributed data
- Visualisation of distributed solutions

Reference material

- Tutorials
  - https://dealii.org/current/doxygen/deal.II/step_17.html
  - https://dealii.org/current/doxygen/deal.II/step_18.html
  - http://www.math.colostate.edu/~bangerth/videos.676.39.html
  - http://www.math.colostate.edu/~bangerth/videos.676.41.html
  - http://www.math.colostate.edu/~bangerth/videos.676.41.25.html
  - http://www.math.colostate.edu/~bangerth/videos.676.41.5.html
  - http://www.math.colostate.edu/~bangerth/videos.676.41.75.html
- Documentation:
  - https://www.dealii.org/current/doxygen/deal.II/group__distributed.html

Parallel computing model: MPI



General Considerations

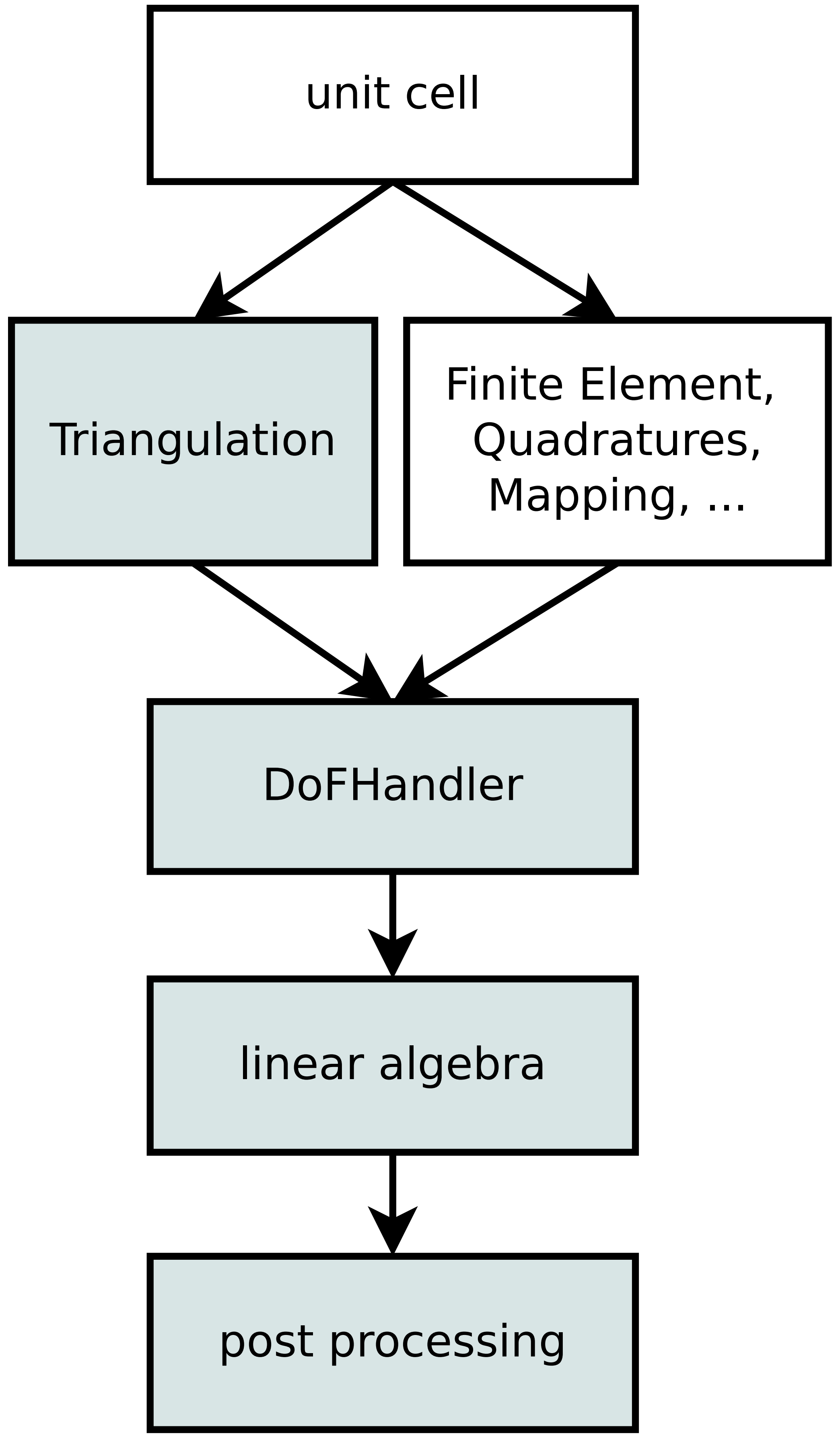
- Goal: get the solution faster!
- If FEM with <500.000 dofs, and 2d, use direct solver!

- If you need more, then you have to **SPLIT** the work
  - **Distributed data** storage everywhere
    - need special data structures
  - **Efficient algorithms**
    - not depending on total problem size
  - **“Localize” and “hide” communication**
    - point-to-point communication, nonblocking sends and receives

Data Structures

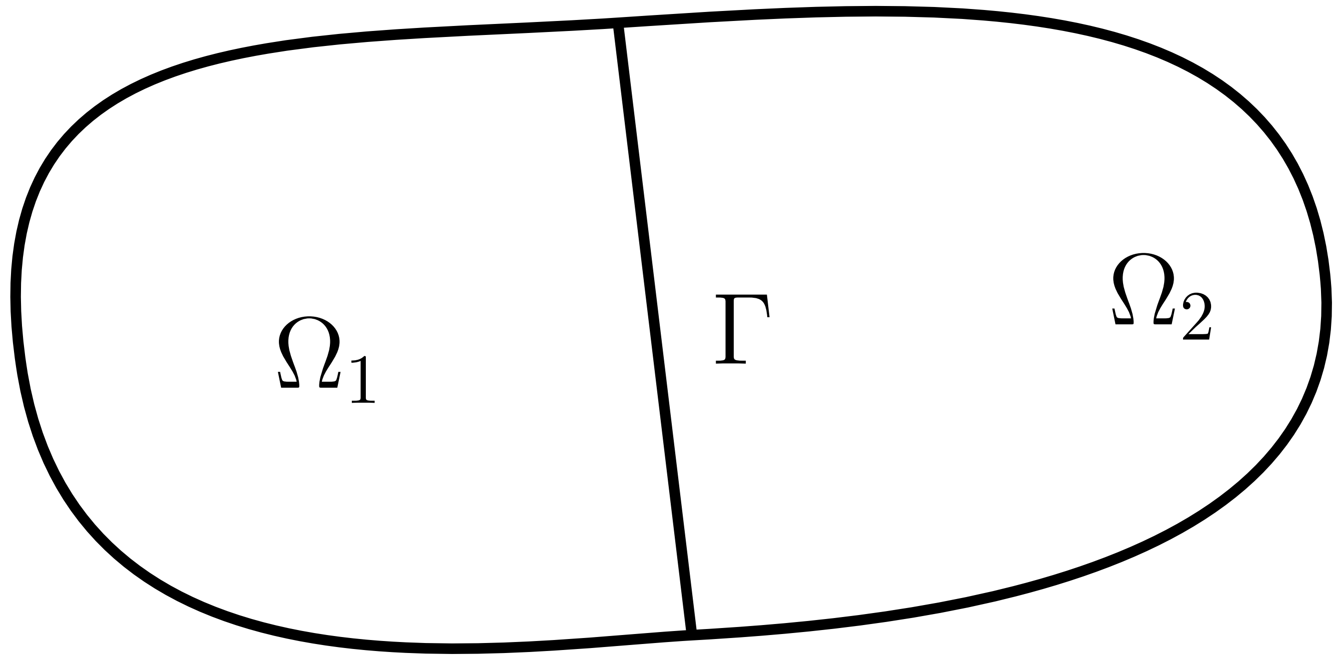
Needs to be parallelized:

1. Triangulation (mesh with associated data)
   — hard: distributed storage, new algorithms
2. DoFHandler (manages degrees of freedom)
   — hard: find global numbering of DoFs
3. Linear Algebra (matrices, vectors, solvers)
   — use existing library
4. Postprocessing (error estimation, solution transfer, output, . . .)
   — do work on local mesh, communicate



How to Parallelize?

Option 1: Domain Decomposition

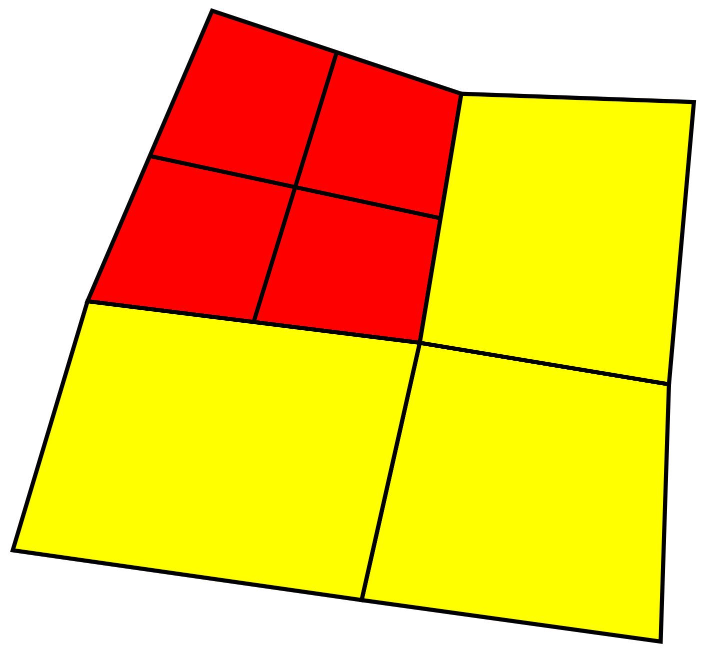


- Split up problem on PDE level
- Solve subproblems independently
- Converges against global solution
- Problems:
  - Boundary conditions are problem dependent:
    $\rightsquigarrow$ sometimes difficult!
    $\rightsquigarrow$ no black box approach!
  - Without coarse grid solver:
    condition number grows with # subdomains
    $\rightsquigarrow$ no linear scaling with number of CPUs!

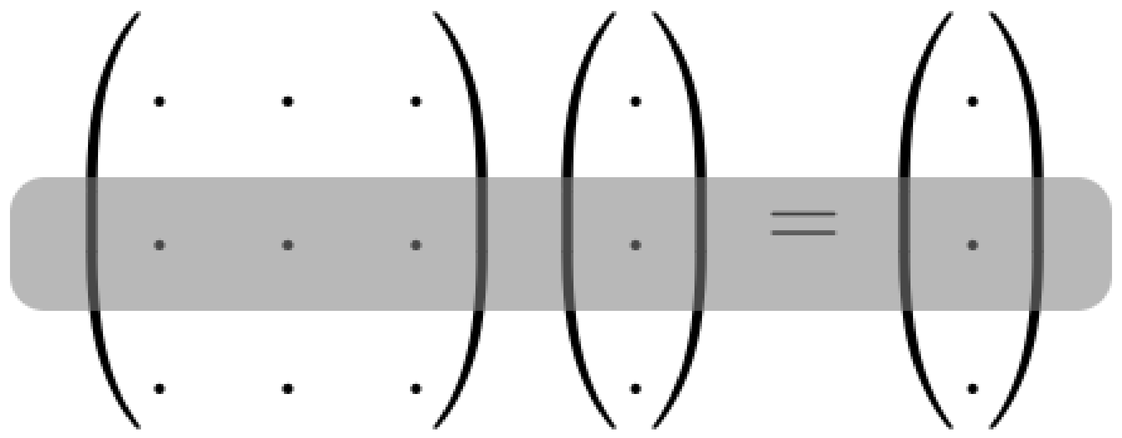
How to Parallelize?

Option 2: Algebraic Splitting

- Split up mesh between processors:



- Assemble logically global linear system (distributed storage):

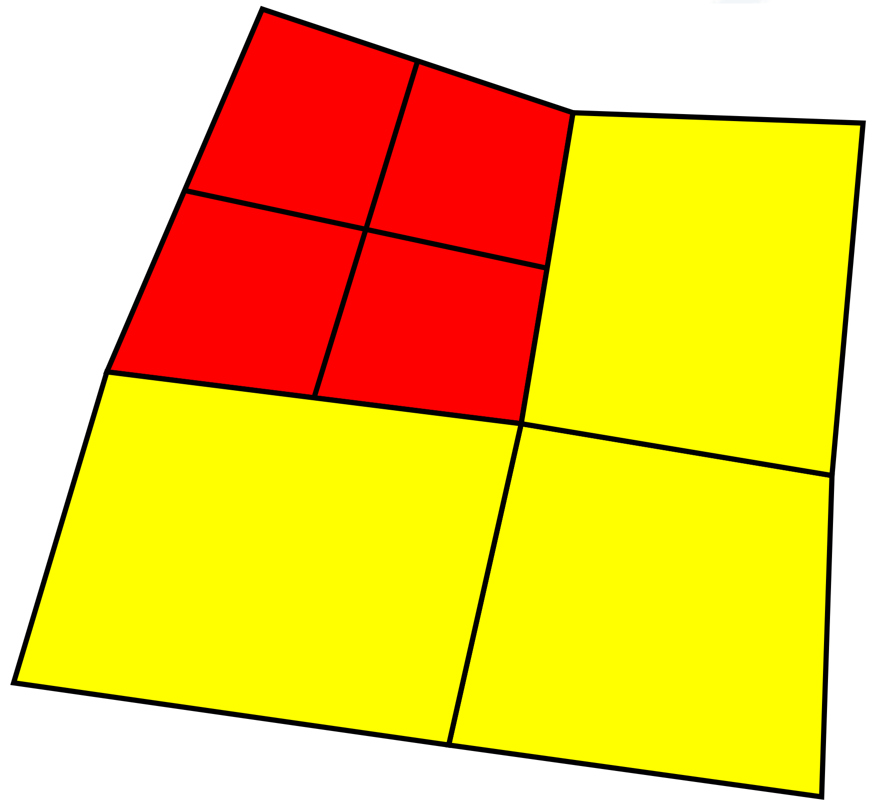


- Solve using iterative linear solvers in parallel
- Advantages:
  - Looks like serial program to the user
  - Linear scaling possible (with good preconditioner)

Partitioning

Optimal partitioning (coloring of cells):

- same size per region
  $\rightsquigarrow$ even distribution of work
- minimize interface between region
  $\rightsquigarrow$ reduce communication



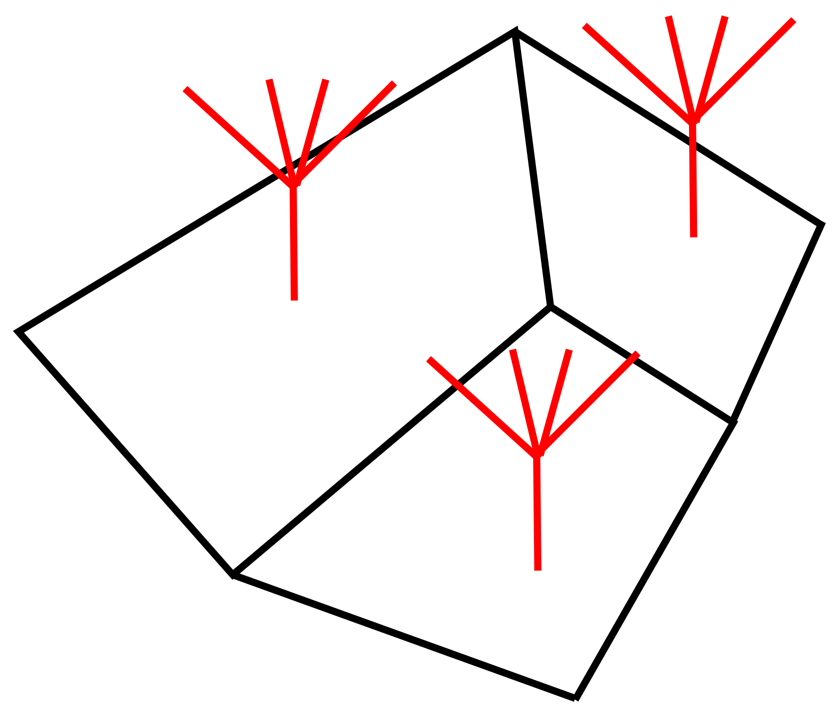
Optimal partitioning is an NP-hard graph partitioning problem.

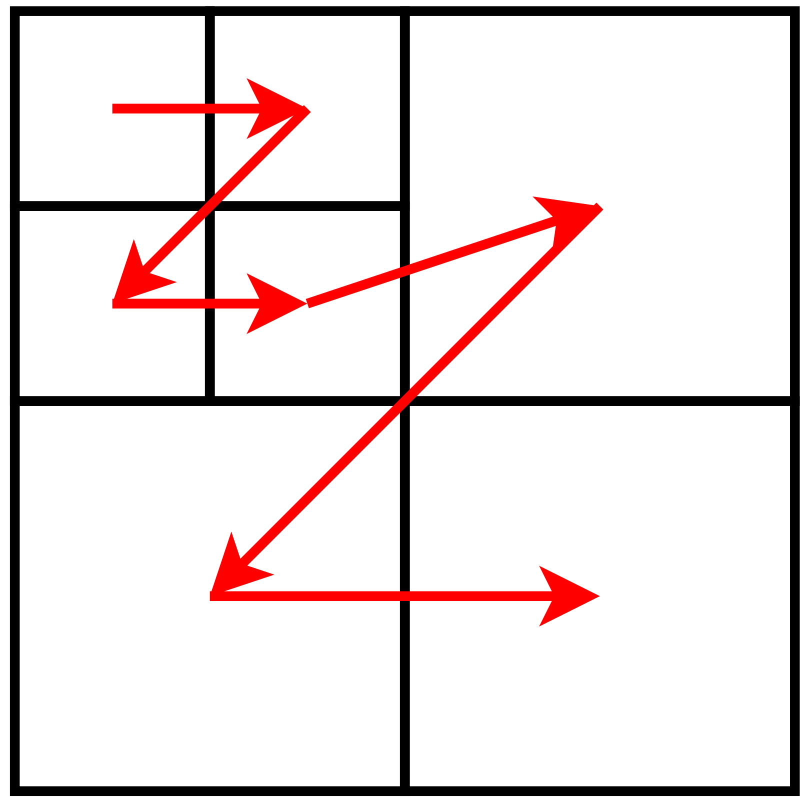
- Typically done: heuristics (existing tools: METIS)
- Problem: worse than linear runtime
- Large graphs: several minutes, memory restrictions

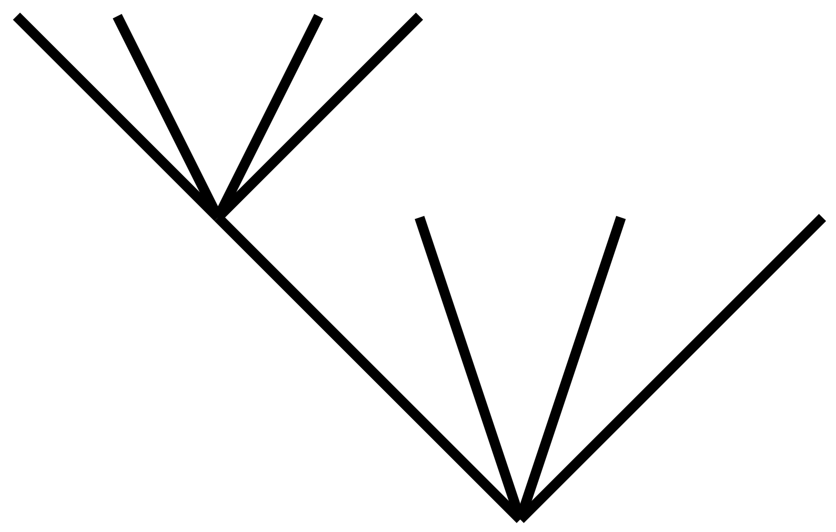
$\rightsquigarrow$ Alternative: avoid graph partitioning

Partitioning with "Space filling curves"

- *p4est* library: parallel quad-/octrees
- Store refinement flags from a base mesh
- Based on space-filling curves
- Very good scalability



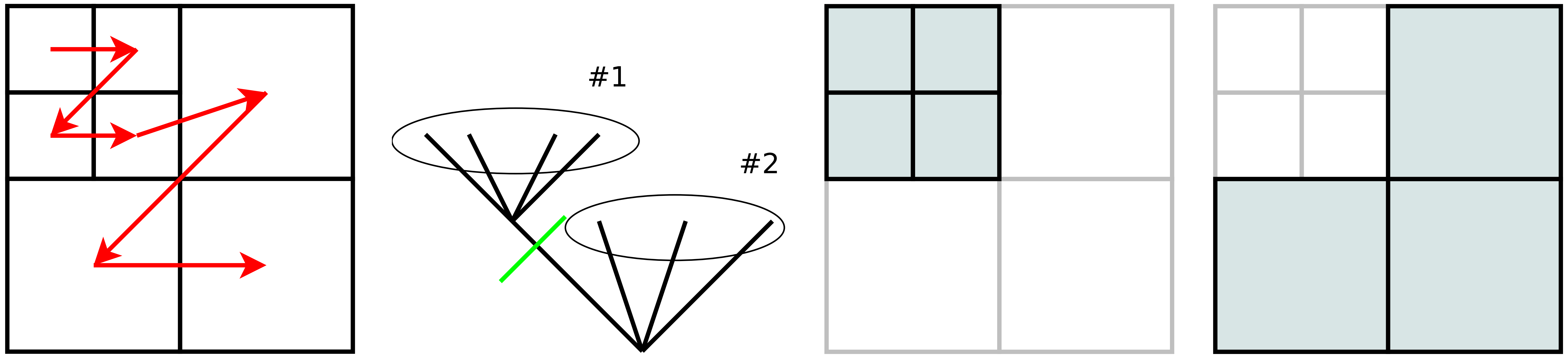




Burstedde, Wilcox, and Ghattas.
p4est: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees.
SIAM J. Sci. Comput., 33 no. 3 (2011), pages 1103-1133.

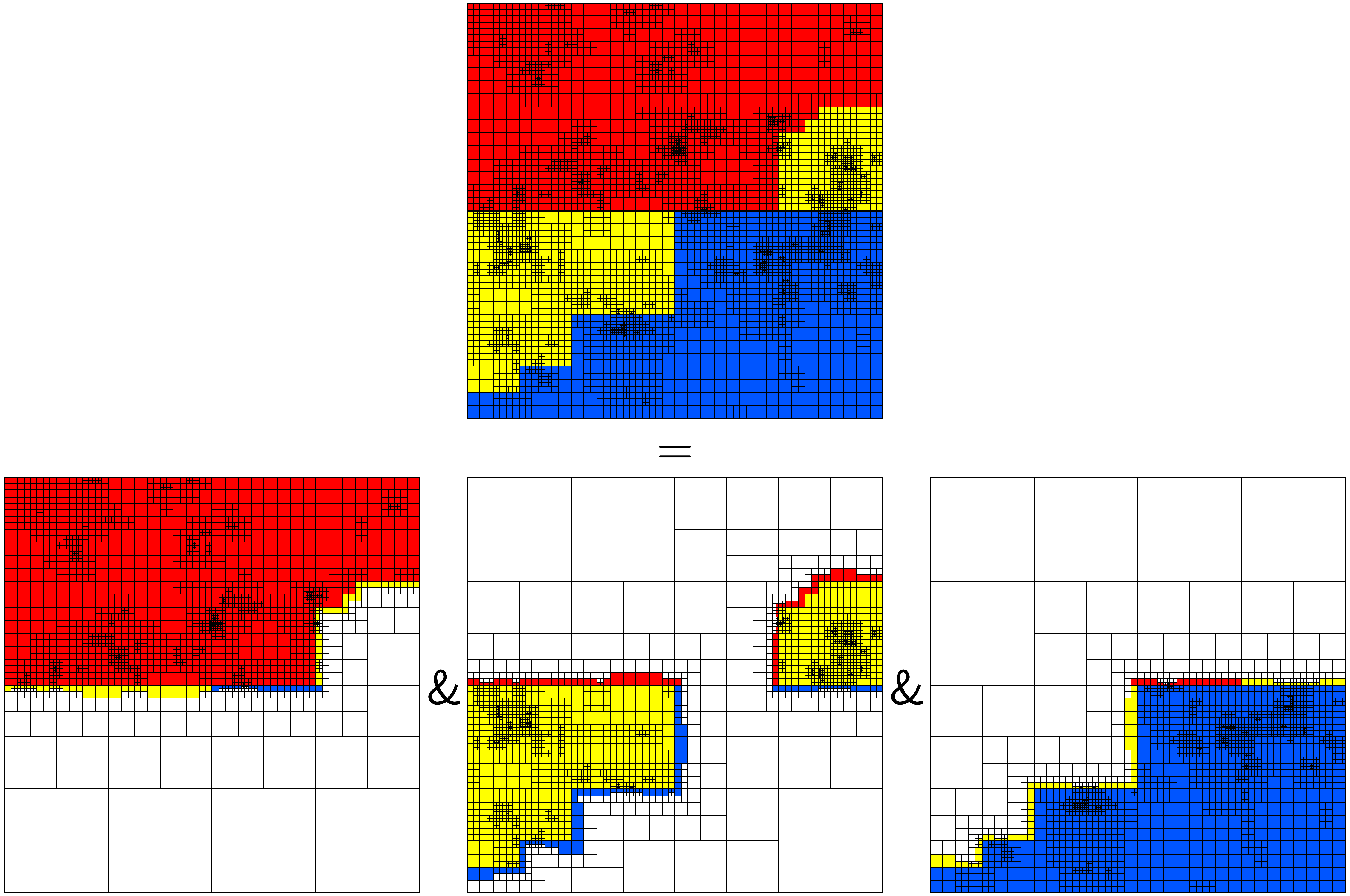
Triangulation

- Partitioning is cheap and simple:



- Then: take *p4est* refinement information
- Recreate rich *deal.II* Triangulation only for local cells (stores coordinates, connectivity, faces, materials, . . .)
- How? recursive queries to *p4est*
- Also create ghost layer (one layer of cells around own ones)

Example (color by CPU ID)

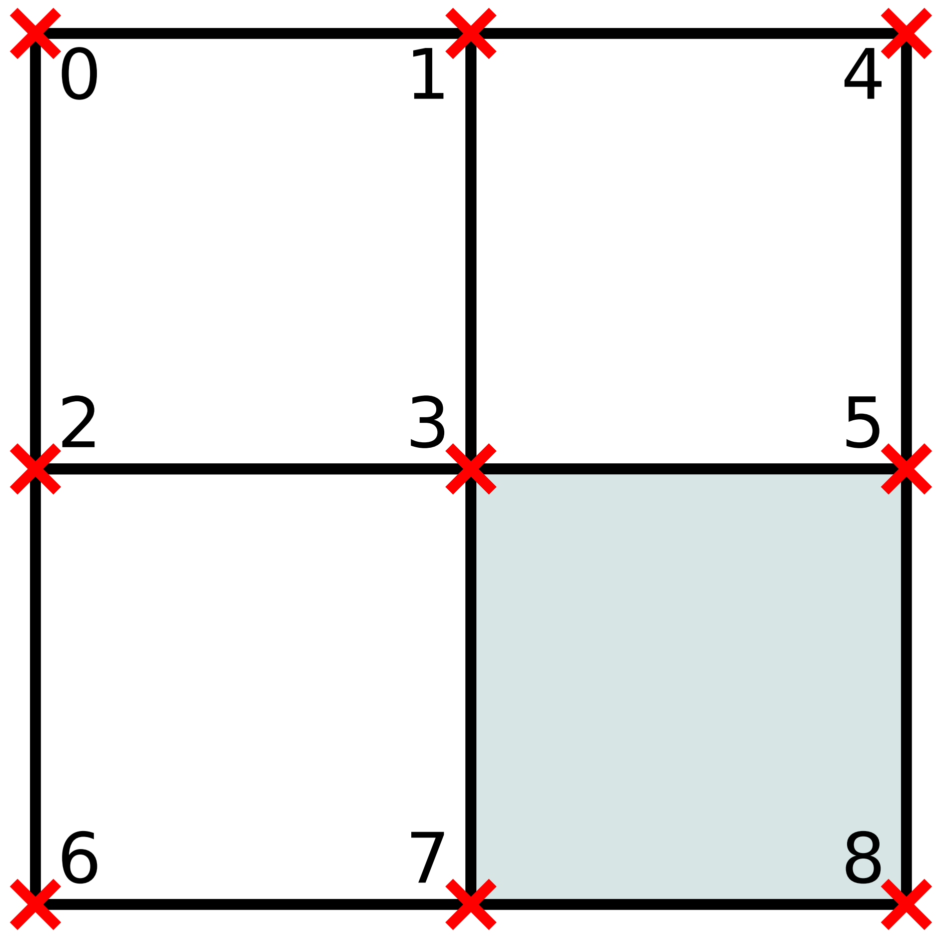


What's needed?

How to use?

- Replace `Triangulation` by `parallel::distributed::Triangulation`
- Continue to load or create meshes as usual
- Adapt with `GridRefinement::refine_and_coarsen*` and `tr.execute_coarsening_and_refinement()`, etc.
- You can only look at own cells and ghost cells: `cell->is_locally_owned()`, `cell->is_ghost()`, or `cell->is_artificial()`
- Of course: dealing with DoFs and linear algebra changes!

Sketch...



- Create global numbering for all DoFs
- Reason: identify shared ones
- Problem: no knowledge about the whole mesh

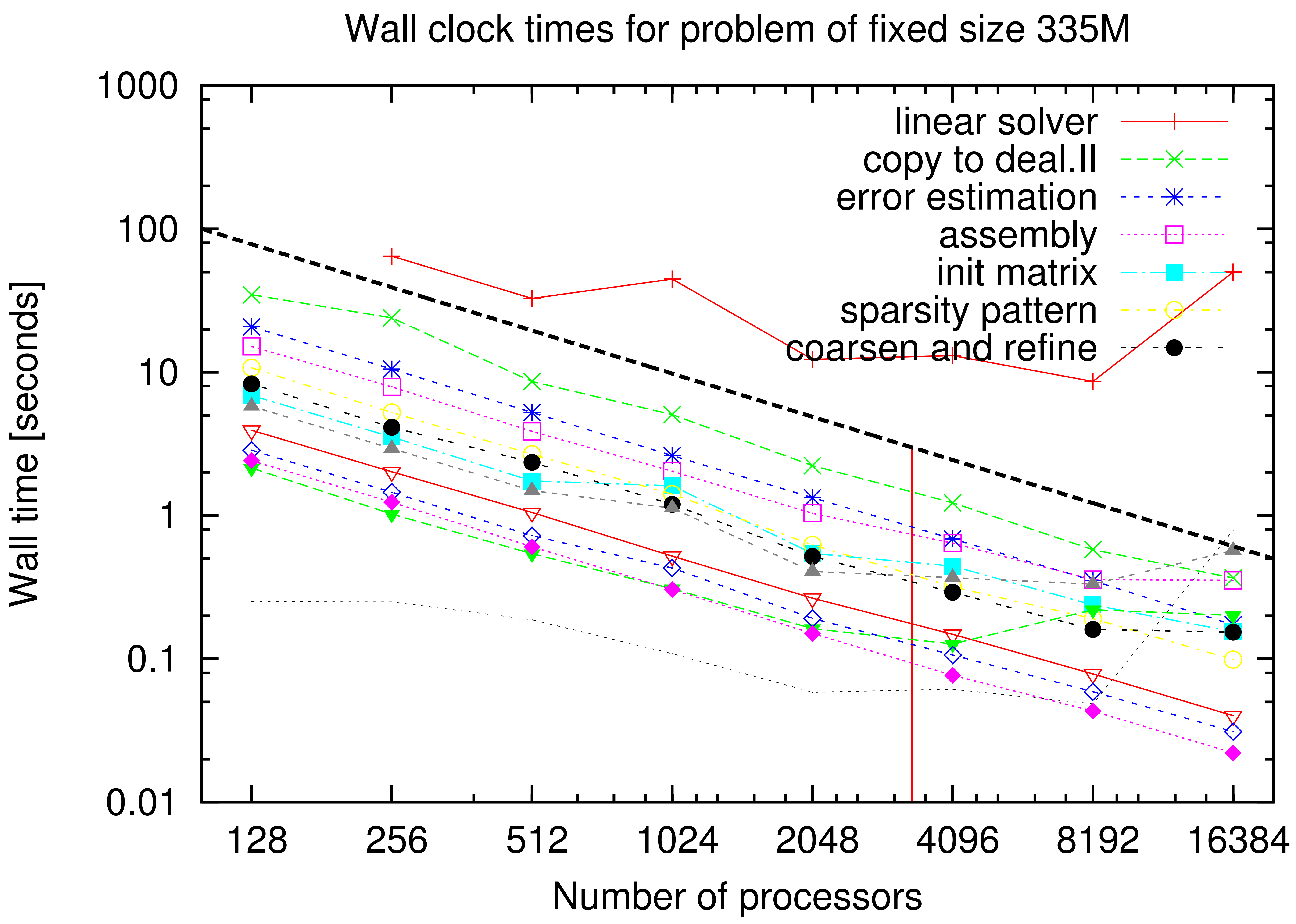
Sketch:

1. Decide on ownership of DoFs on interface (no communication!)
2. Enumerate locally (only own DoFs)
3. Shift indices to make them globally unique (only communicate local quantities)
4. Exchange indices to ghost neighbors

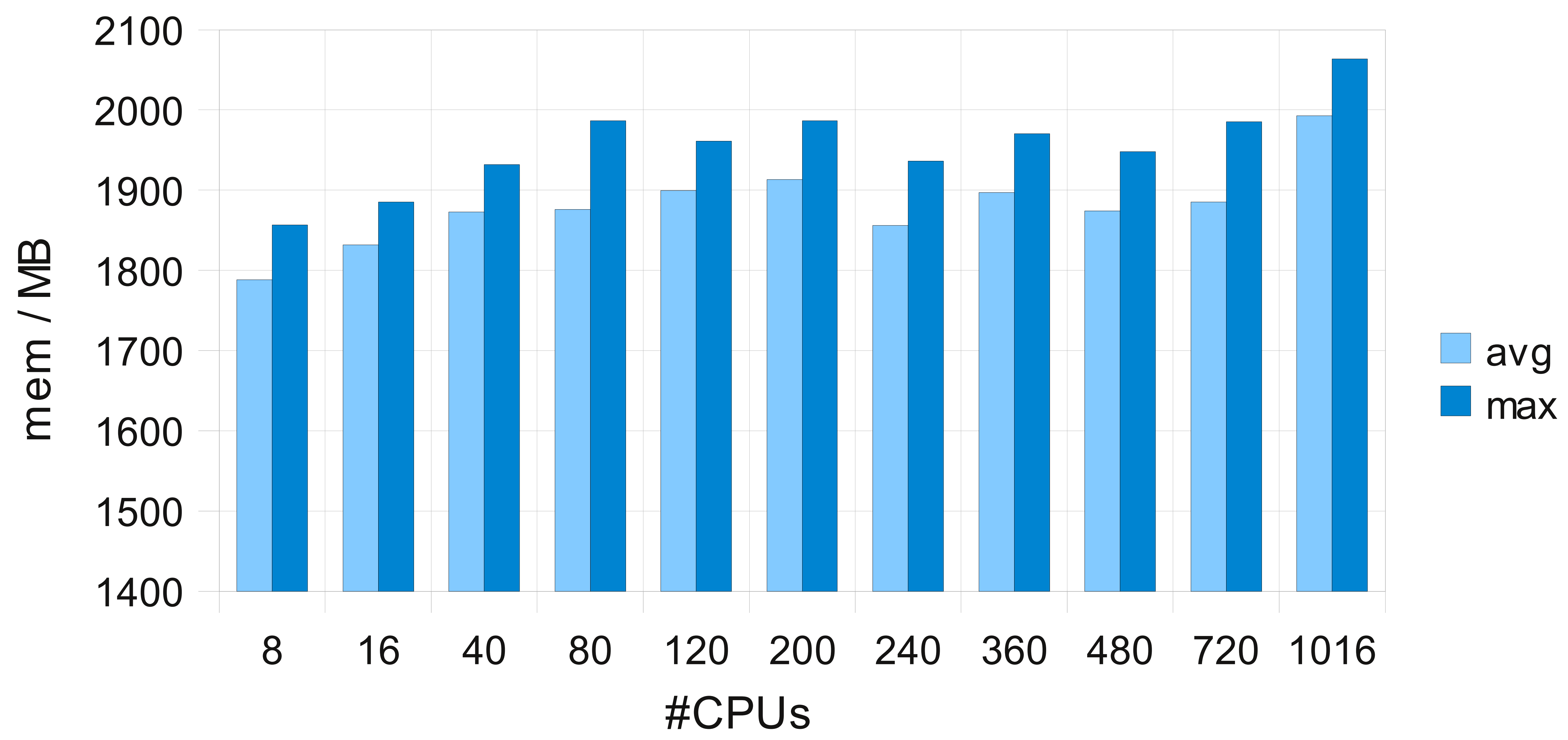
Solvers

- Iterative solvers only need Mat-Vec products and scalar products
  $\rightsquigarrow$ equivalent to serial code
- Can use templated deal.II solvers like GMRES!
- Better: use tuned parallel iterative solvers that hide/minimize communication
- Preconditioners: more work, just operating on local blocks not enough

Strong Scaling: 2d Poisson



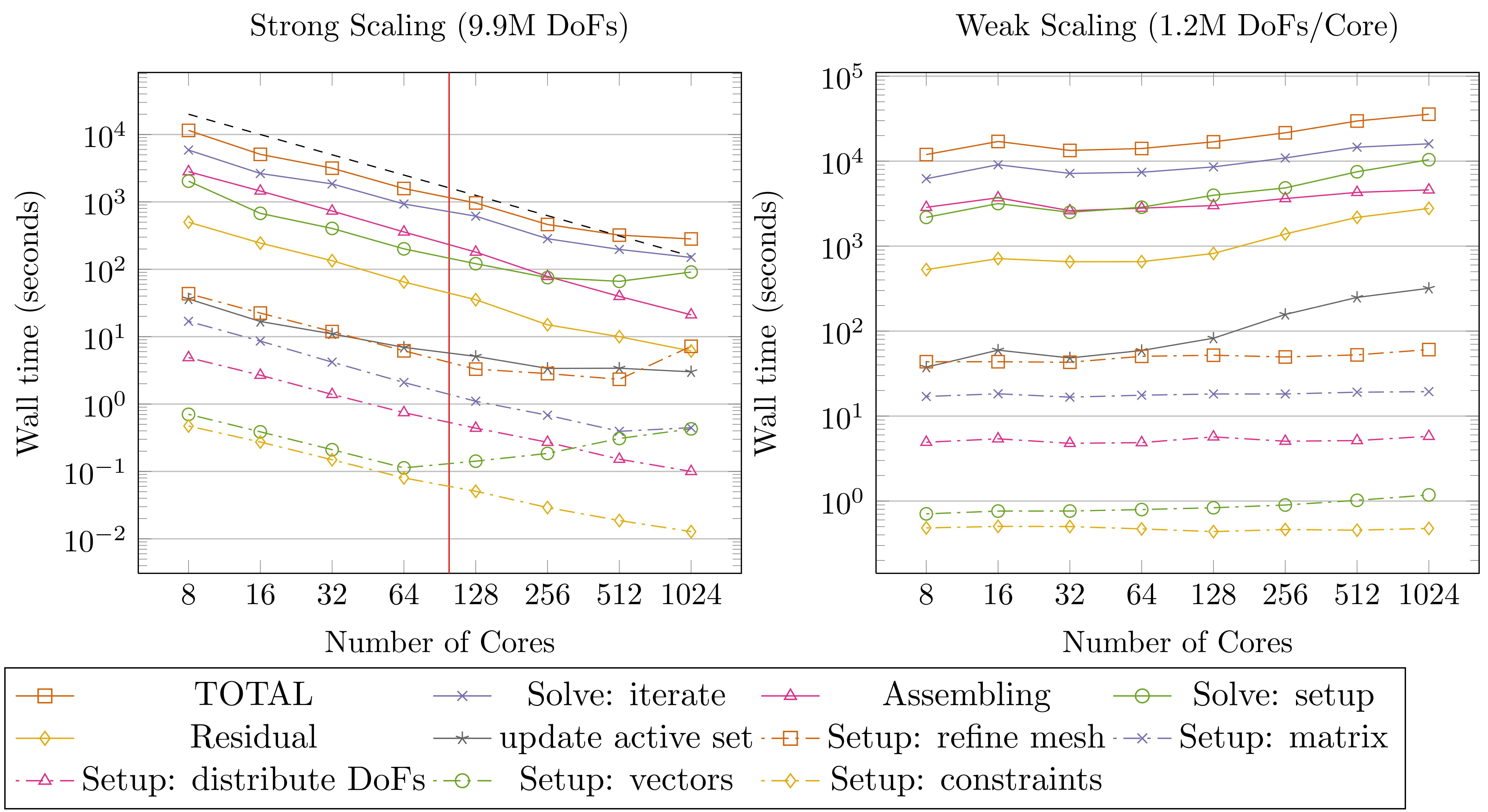
Memory Consumption



average and maximum memory consumption (VmPeak)
3D, weak scalability from 8 to 1000 processors with about 500.000 DoFs per processor (4 million up to 500 million total)

$\rightsquigarrow$ **Constant memory usage with increasing # CPUs & problem size**

Step 40



Strong Scaling (9.9M DoFs)
Weak Scaling (1.2M DoFs/Core)
Wall time (seconds)
Number of Cores
8
16
32
64
128
256
512
1024
10^{-2}
10^{-1}
10^0
10^1
10^2
10^3
10^4
10^5
TOTAL
Solve: iterate
Assembling
Solve: setup
Residual
update active set
Setup: refine mesh
Setup: matrix
Setup: distribute DoFs
Setup: vectors
Setup: constraints

Trilinos VS PETSc

What should I use?

- Similar features and performance
- Pro Trilinos: more development, some more features (automatic differentation, . . .), cooperation with deal.II
- Pro PETSc: stable, easier to compile on older clusters
- But: being flexible would be better! – “why not both?”
  - you can! Example: new `step-40`
  - can switch at compile time
  - need #ifdef in a few places (different solver parameters TrilinosML vs BoomerAMG)
  - some limitations, somewhat work in progress

Trilinos VS PETSc

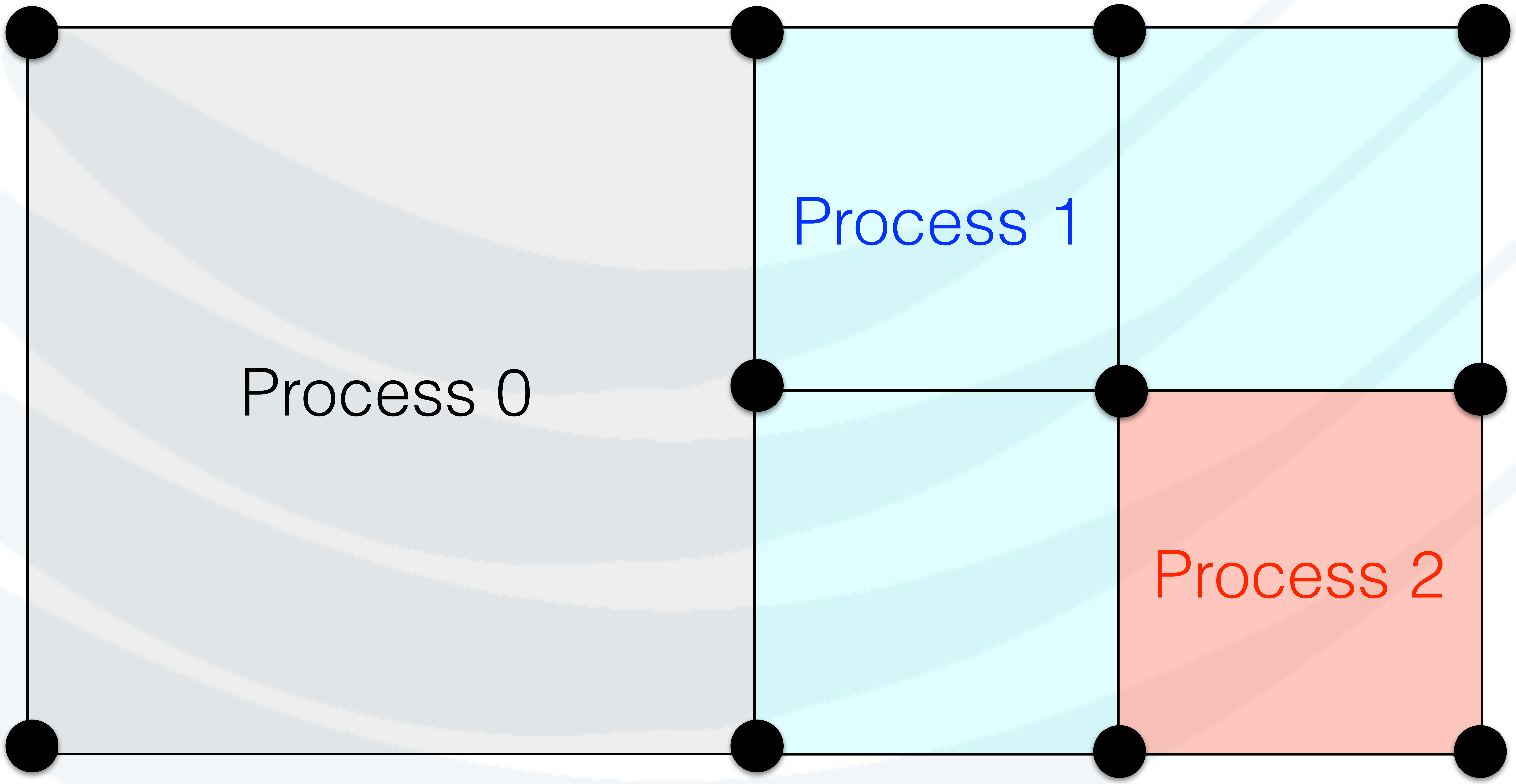
```
#include <deal.II/lac/generic_linear_algebra.h>
#define USE_PETSC_LA // uncomment this to run with Trilinos

namespace LA
{
#ifdef USE_PETSC_LA
  using namespace dealii::LinearAlgebraPETSc;
#else
  using namespace dealii::LinearAlgebraTrilinos;
#endif
}

// ...
    LA::MPI::SparseMatrix system_matrix;
    LA::MPI::Vector solution;

// ...
    LA::SolverCG solver(solver_control, mpi_communicator);
    LA::MPI::PreconditionAMG preconditioner;

    LA::MPI::PreconditionAMG::AdditionalData data;

#ifdef USE_PETSC_LA
    data.symmetric_operator = true;
#else
    //trilinos defaults are good
#endif
    preconditioner.initialize(system_matrix, data);

// ...
```

sissa mathLab innovating with mathematics

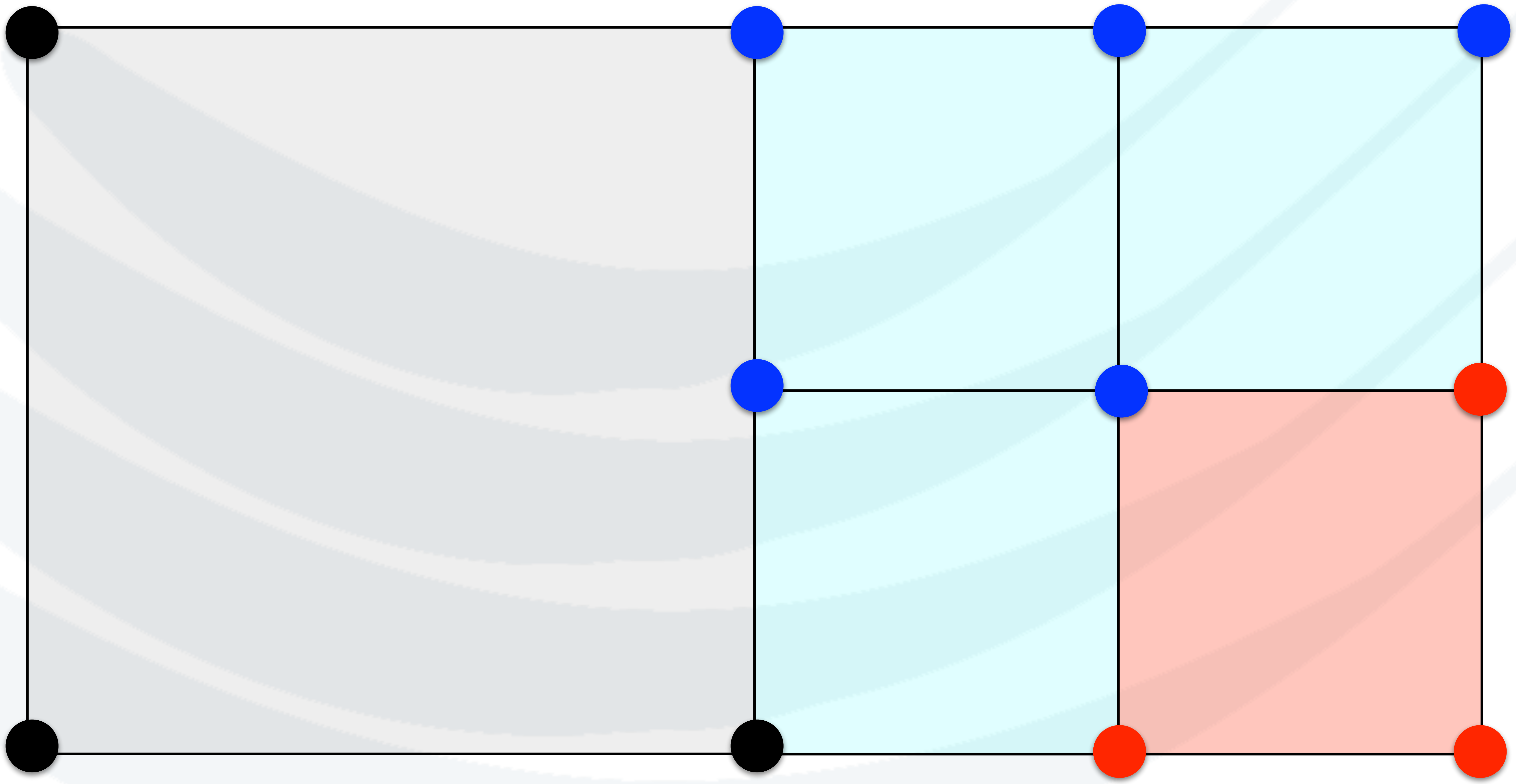
The Abdus Salam International Centre for Theoretical Physics

SISSA

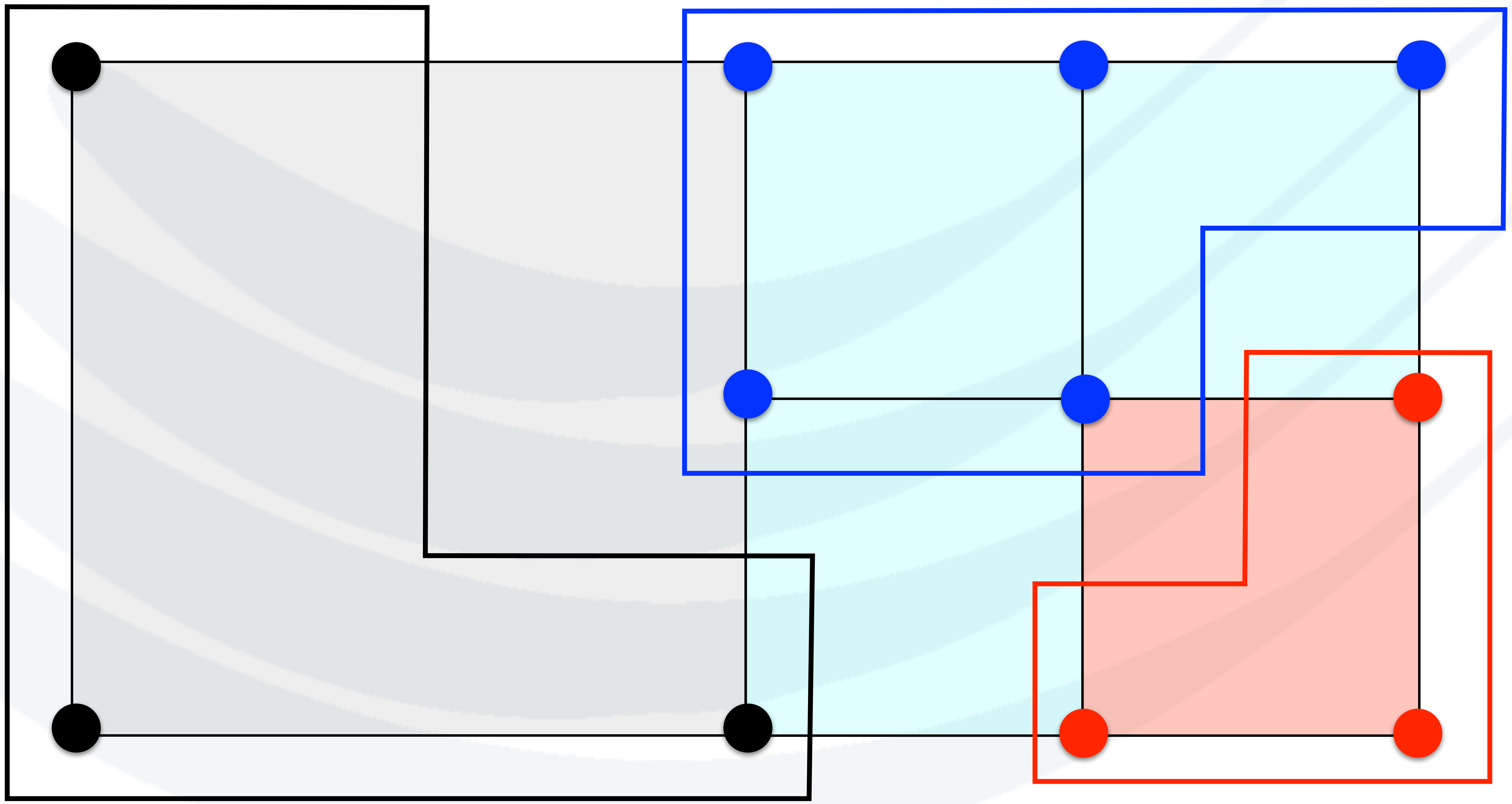
Distribution of degrees-of-freedom: Colourisation of DoFs via graph partitioner



Distribution of degrees-of-freedom: Colourisation of DoFs via graph partitioner

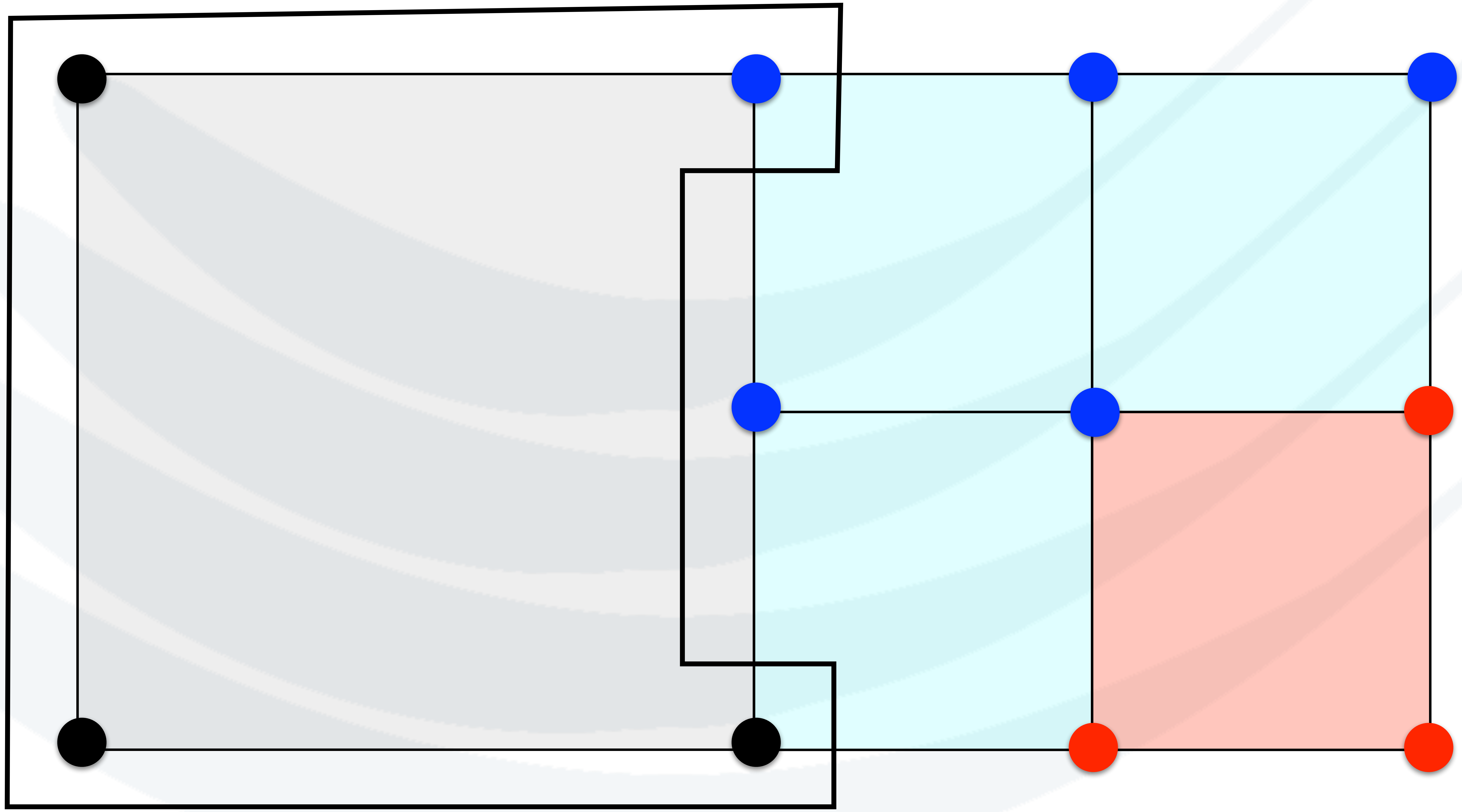


Ownership of distributed DoFs: Locally owned degrees-of-freedom



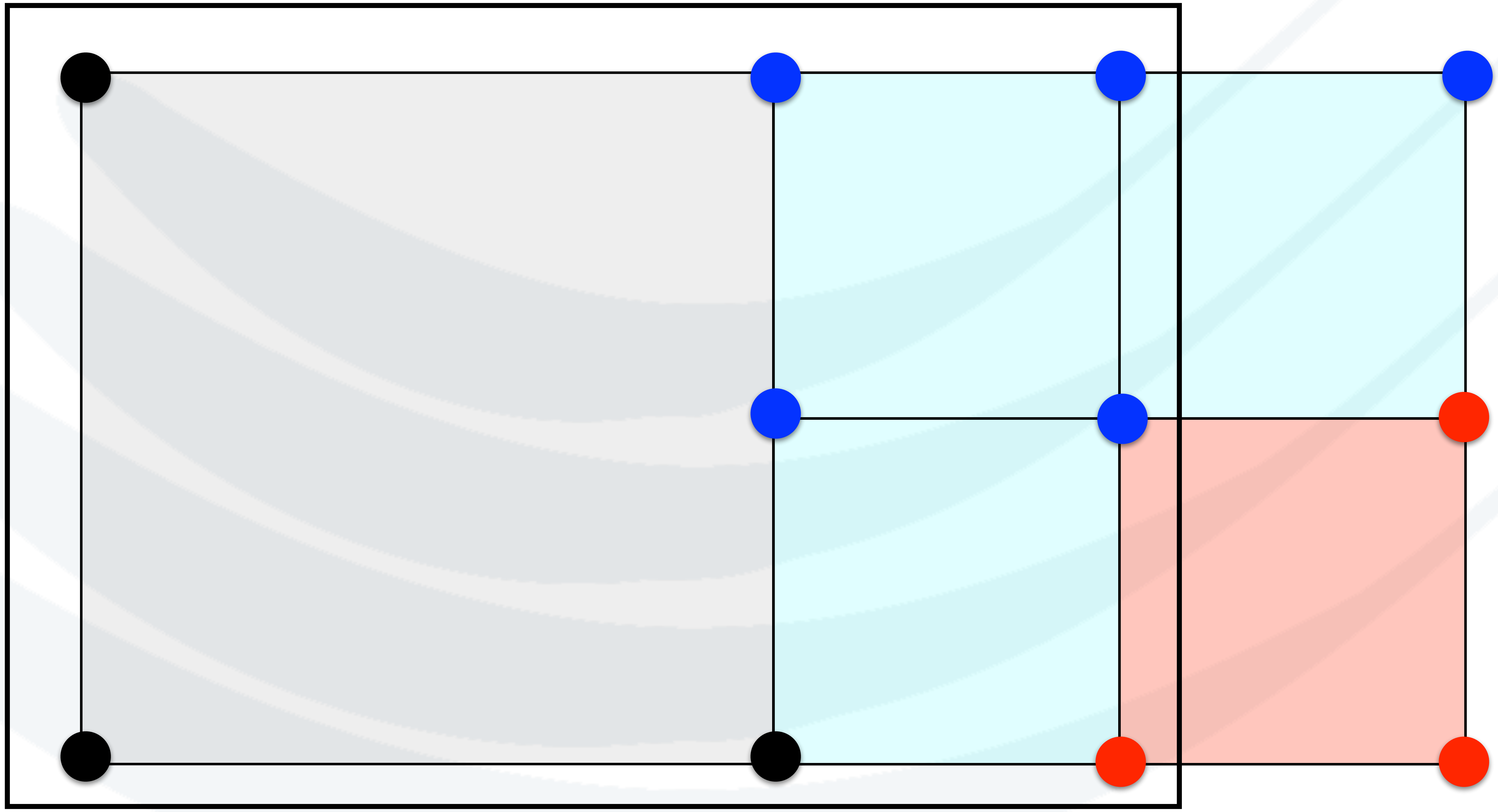
Distribution of DoFs:
Locally relevant degrees-of-freedom (process 0)

- Required for assembly, data output



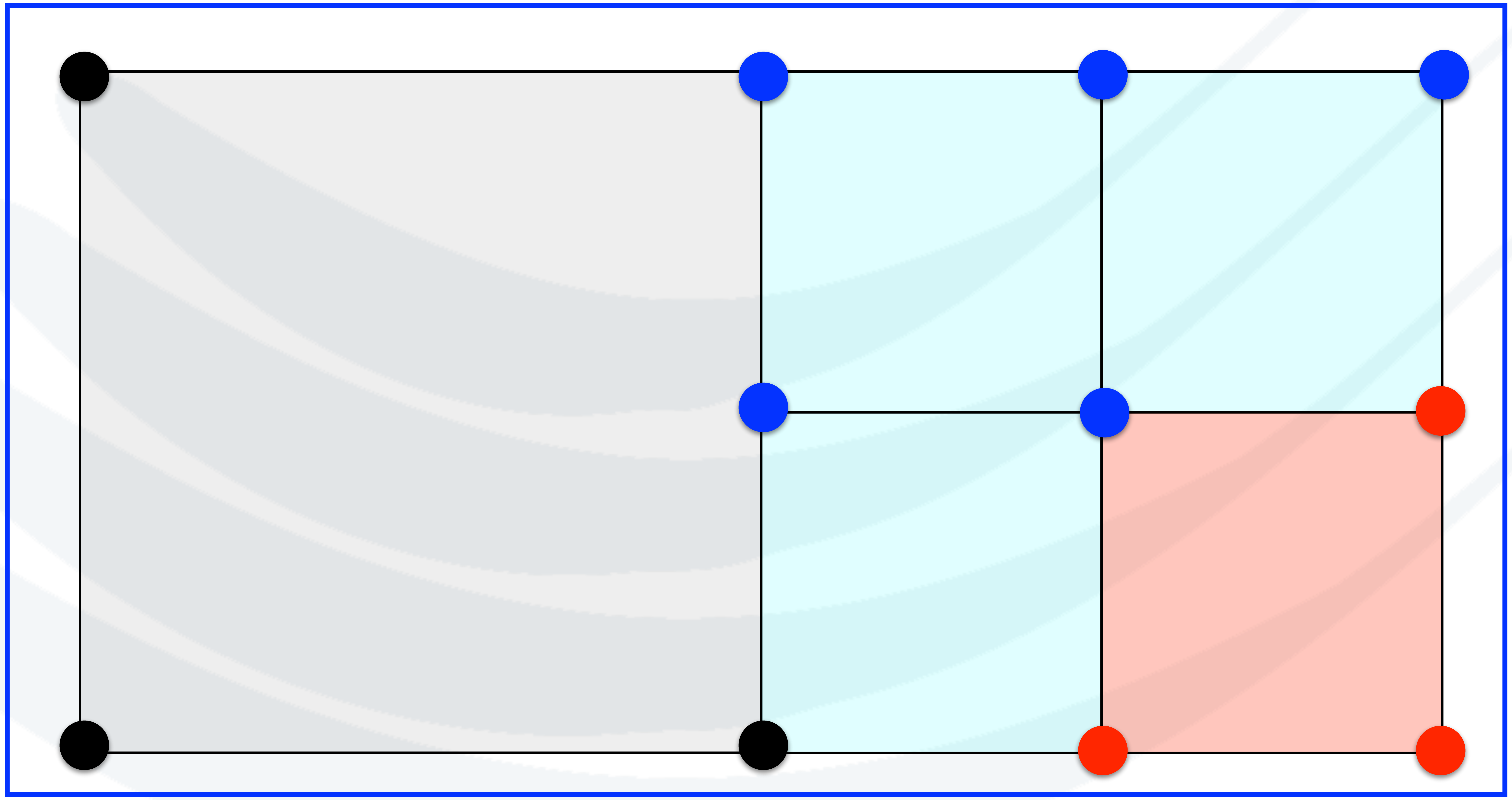
Distribution of DoFs:
Locally relevant degrees-of-freedom (process 0)

- Required for Kelly error estimator



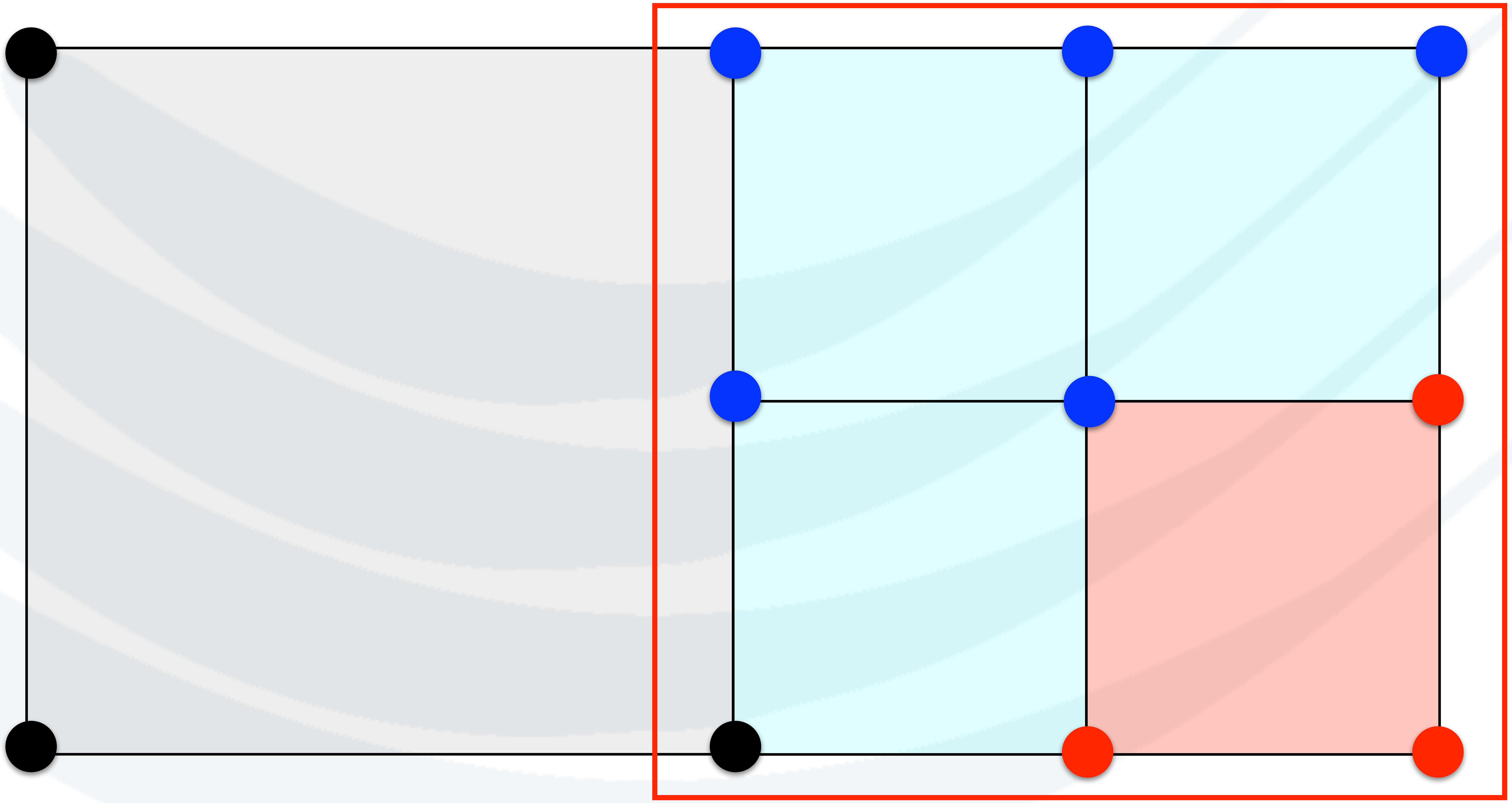
Distribution of DoFs:
Locally relevant degrees-of-freedom (process 1)

- Required for Kelly error estimator



Distribution of DoFs:
Locally relevant degrees-of-freedom (process 2)

- Required for Kelly error estimator



Changes: New headers

- MPI
  ```
  #include <deal.II/base/mpi.h>
  ```
- Parallel shared triangulation
  ```
  #include <deal.II/distributed/shared_tria.h>
  ```
- Filtered iterator
  ```
  #include <deal.II/grid/filtered_iterator.h>
  ```
- IndexSet
  ```
  #include <deal.II/base/index_set.h>
  ```
- Trilinos linear algebra
  ```
  #include <deal.II/lac/trilinos_*>
  ```
- Output filter
  ```
  #include <deal.II/base/conditional_ostream.h>
  ```

Changes: Class definition

- MPI utility objects
  - MPI communicator
  - Number of processes, number of “this” process
  - Stream output assistant (filter)
- Triangulation type
- Sparse linear algebra objects
- IndexSets
  - Locally owned
  - Locally relevant

Changes: System setup

- Must determine the set of locally owned and locally relevant DoFs
  - Locally owned = those assigned to a particular MPI process
  - Locally relevant = those assigned to other processors, but are required to perform some action on the current process
- Distribution of sparsity pattern
  - Tell the locally defined sparsity pattern which entries require data exchange / may be written into by other processes
- Can interrogate information about problem distribution as viewed from other processors

Changes: Assembly

- Cell loop: Only cells owned by the MPI process
  - Can use filtered iterators
  - Can check within the cell loop
    `cell->locally_owned();`
- All assembled data is initially localised to a process
- Final synchronisation of data between MPI processes
  - Accumulation of values on DoFs written into by more than one process
    `[matrix/vector].compress(VectorOperations::add);`
    - Only once (outside of cell loop)

Changes: Linear solver

- Solver templated on Vector type
- Preconditioner type

Changes: Mesh refinement

- Kelly error estimator needs to know solution values on cells that are not owned by this current MPI process
- Need to provide view of solution with values for all locally owned and locally relevant DoFs

Changes: Postprocessing

- Write out portion of solution from each processor
  - deal.II writes these outputs on a per-cell basis
  - Need to provide view of solution with values for all locally owned and locally relevant DoFs

Changes: Main function

- Setup MPI environment

```
Utilities::MPI::MPI_InitFinalize mpi_initialization(argc, argv, 1);
```