MPI parallelization

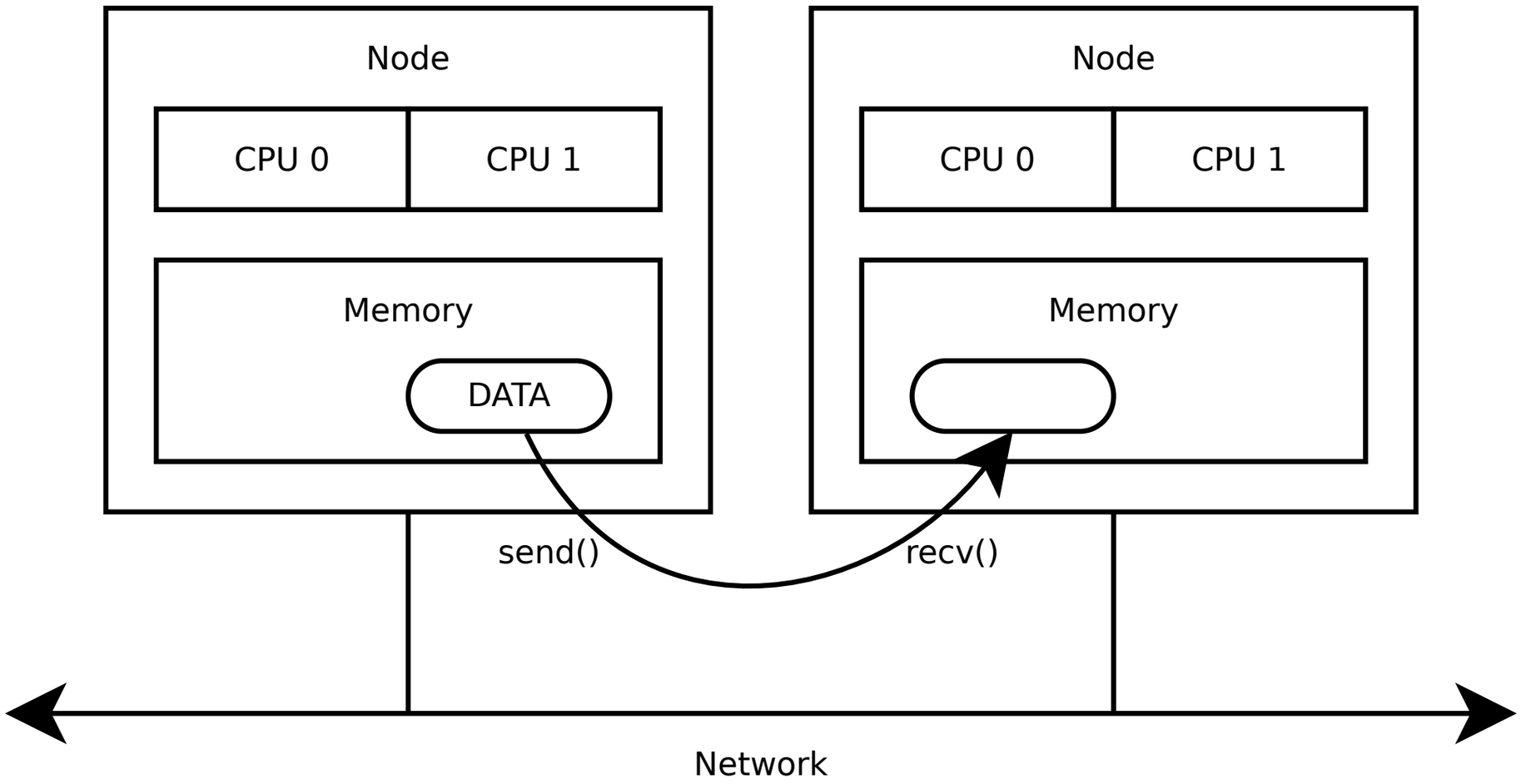
Aims for this module

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

Reference material

- Tutorials
  - step-17
  - step-18
  - Lecture 39: Parallelization: Introduction
  - Lecture 41: Parallelization on a cluster of distributed memory machines — Part 1: Introduction to MPI
  - Lecture 41.25: Parallelization on a cluster of distributed memory machines — Part 2: Debugging with MPI
  - Lecture 41.5: Parallelization on a cluster of distributed memory machines — Part 3: Distributed computing in deal.II and step-40
  - Lecture 41.75: Parallelization on a cluster of distributed memory machines — Part 4: Parallel solvers and preconditioners
- Documentation:
  - Parallel computing with multiple processors using distributed memory

Parallel computing model: MPI



- data is distributed
- each MPI process has access only to its local data
- there is communication between MPI processes (over the network)

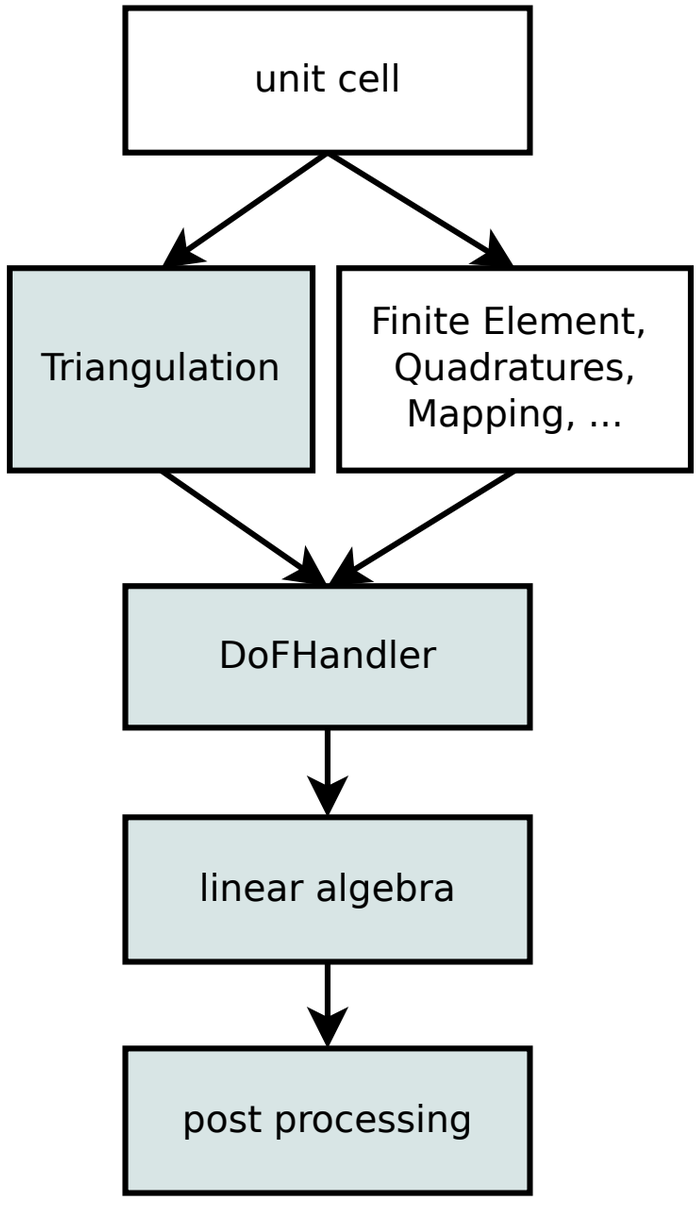
General Considerations

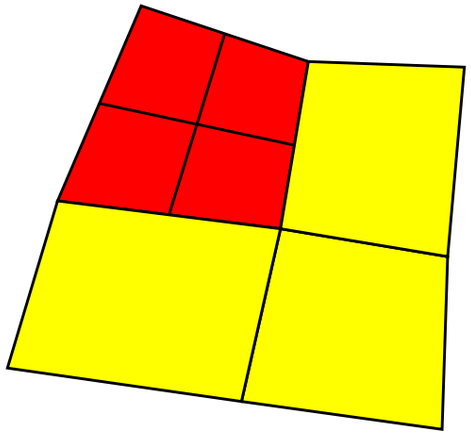
- Goal: get the solution faster!
- If FEM with <500k DoFs, and 2d, use direct solver!

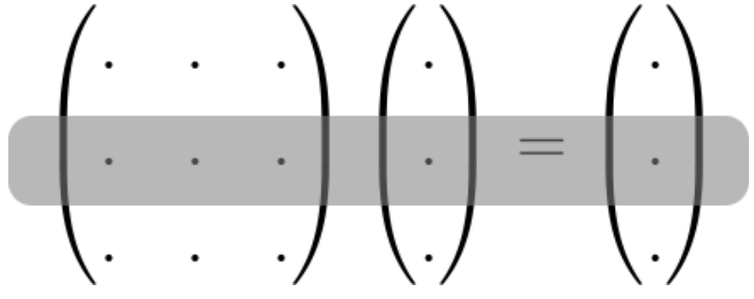
- If you need more, then you have to **SPLIT** the work
  - **Distributed data** storage everywhere
    - need special data structures
  - **Efficient solvers and algorithms**
    - not depending on total problem size
  - “Localize” and “hide” communication
    - point-to-point communication, nonblocking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

How to Parallelize?

- Partition FE mesh between MPI processes
- Introduce 1D row-wise partitioning of the global sparse stiffness matrix
- Assemble locally owned parts of stiffness matrix on each MPI process
- Solve using linear solvers in parallel
  - with good (multi-grid) preconditioner number of CG iterations will NOT depend on the problem size (i.e. number of DoFs)
  - this will give linear scaling w.r.t. the problem size
  - iterative solvers (e.g. conjugate gradient) only need matrix-vector products and inner products

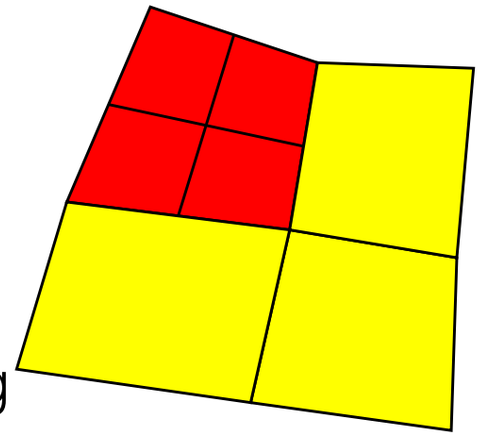






Partitioning

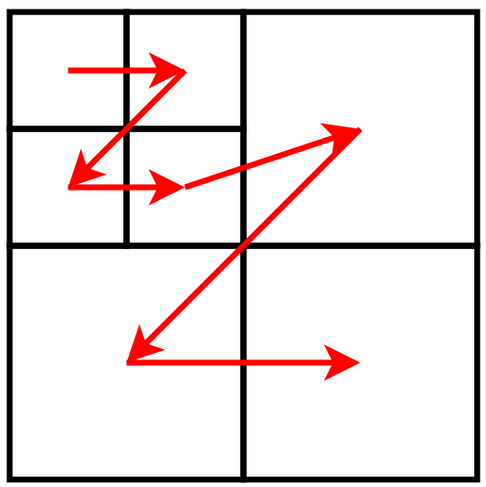
Optimal partitioning of cells between MPI processes is an NP-hard **graph partitioning** problem:

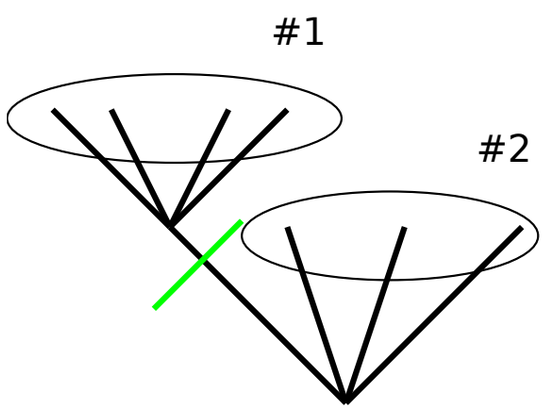


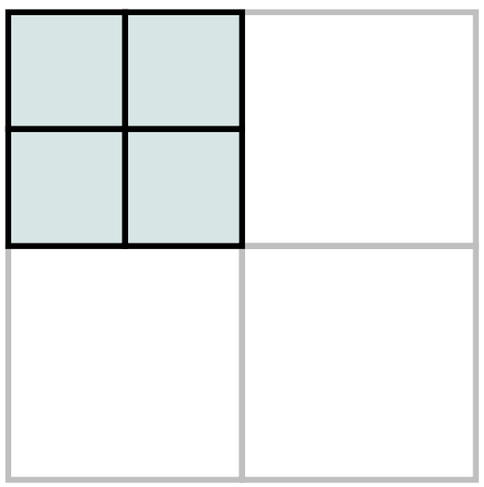
- same size per region (even distribution of work)
- minimize interface between regions (reduce communication)

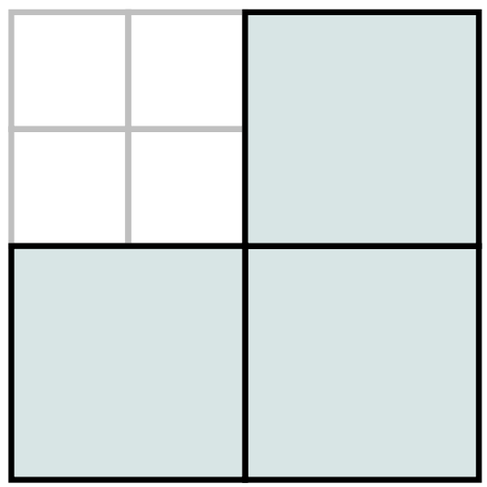
For large meshes its too time consuming and at best can be done using heuristics (e.g. METIS library)

Alternative is to use **space filling curves** for partitioning (e.g. p4est library):



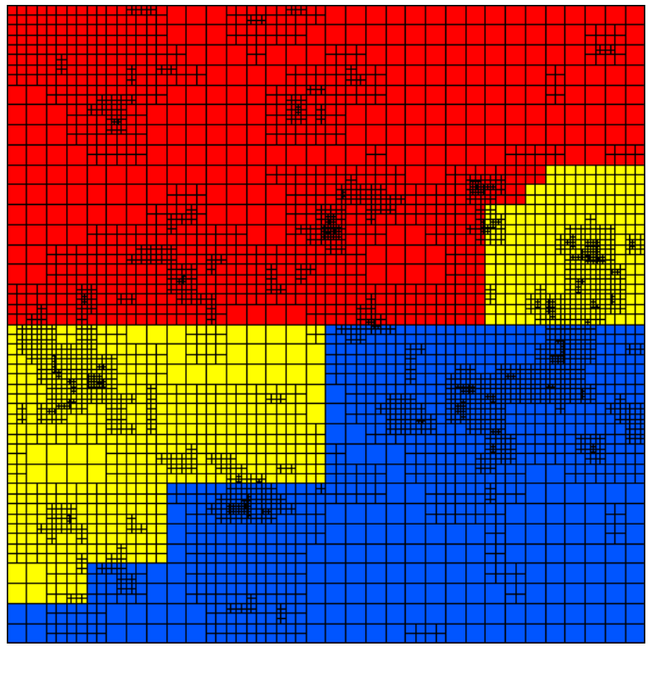




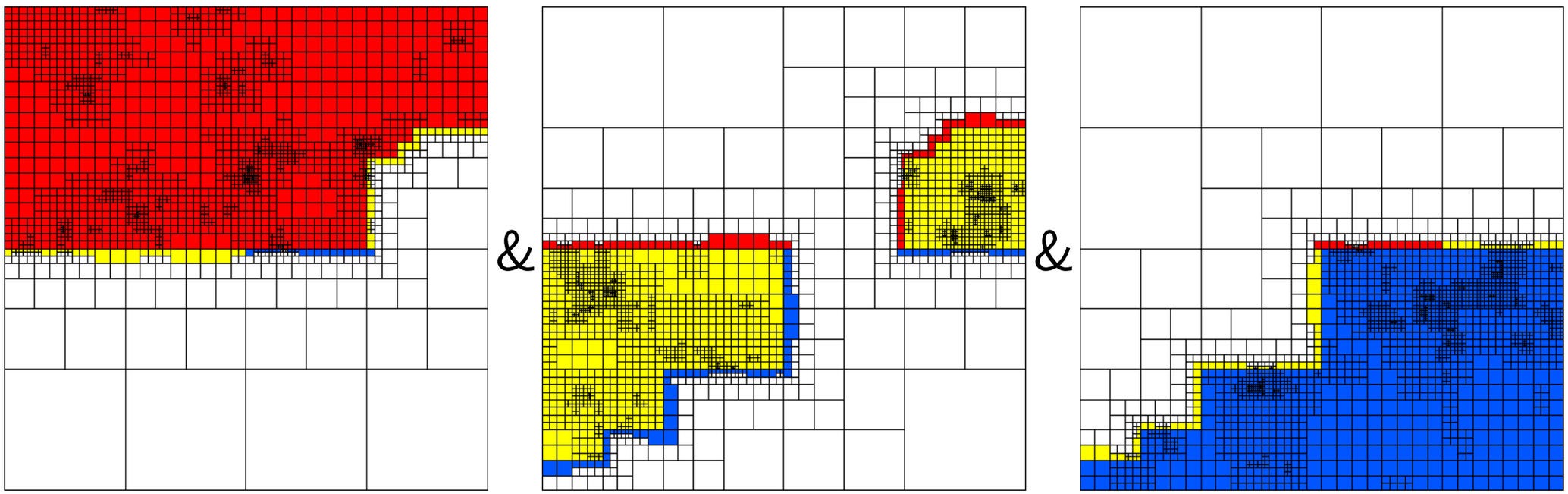


- good scalability, cheap, simple
- parallel quad- and oct-trees
- each process stores only locally owned cells* and a ghost layer around

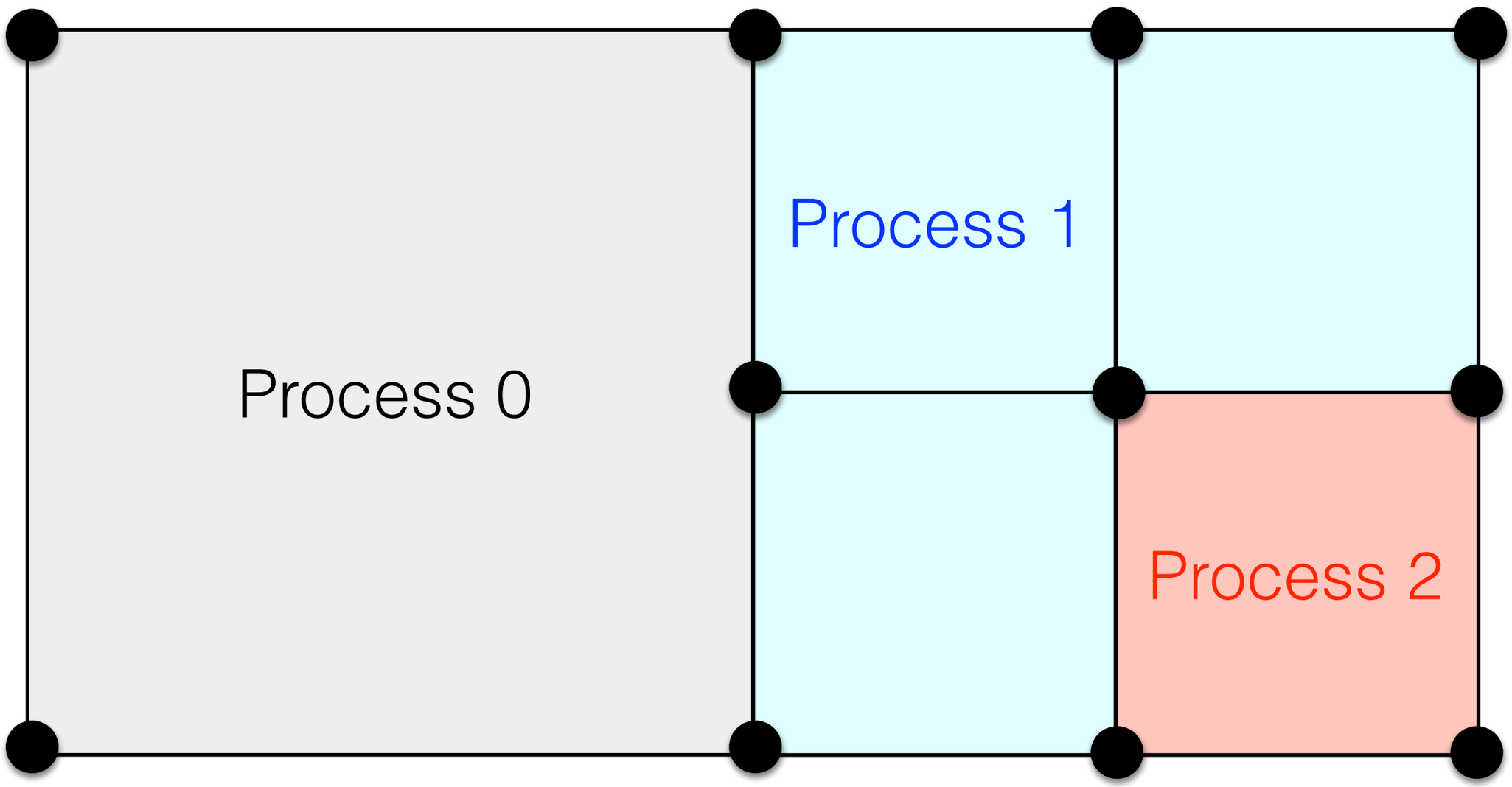
Example (color by CPU ID)



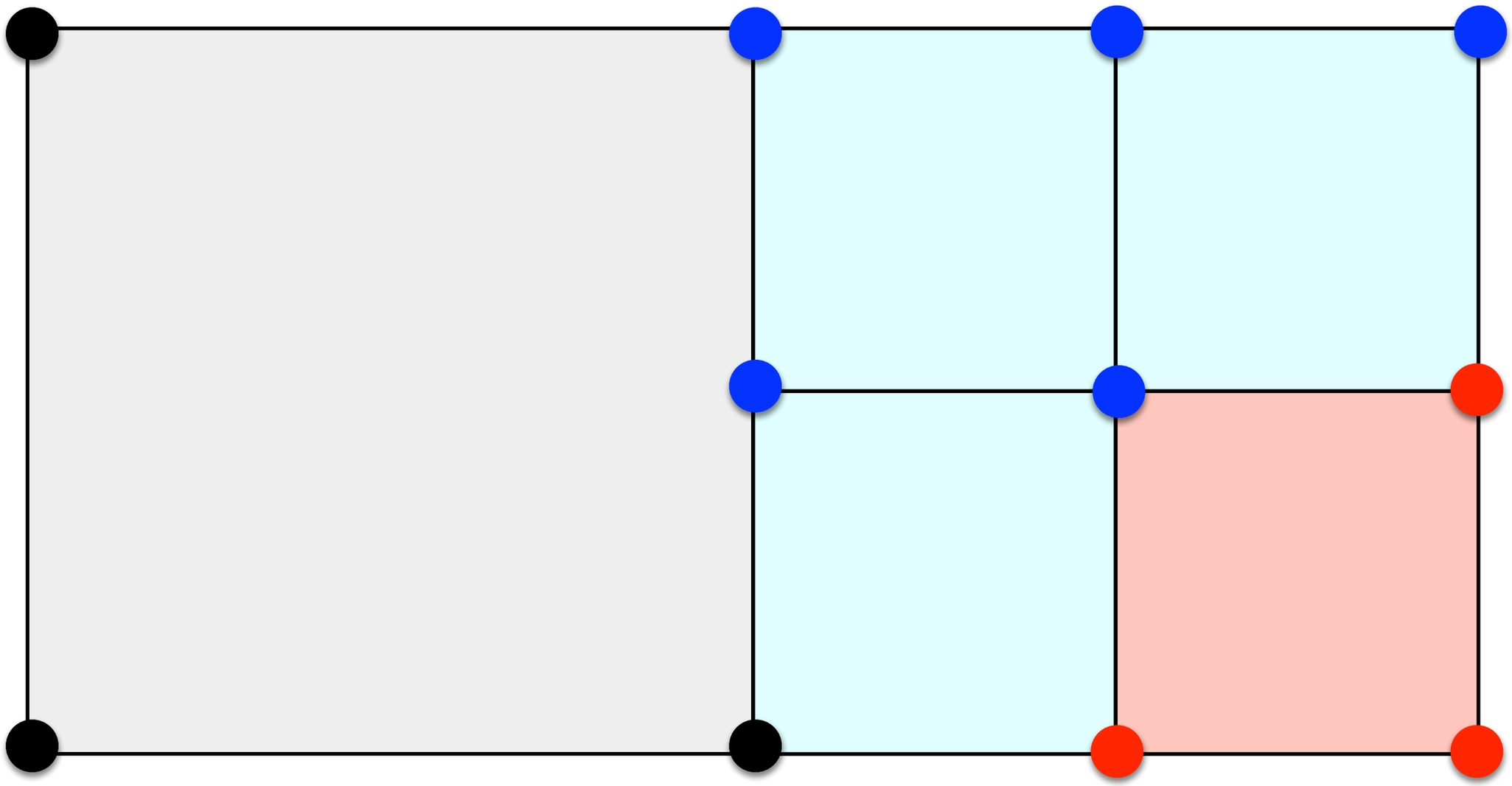
=



Distribution of degrees-of-freedom: Coloring of cell ownership

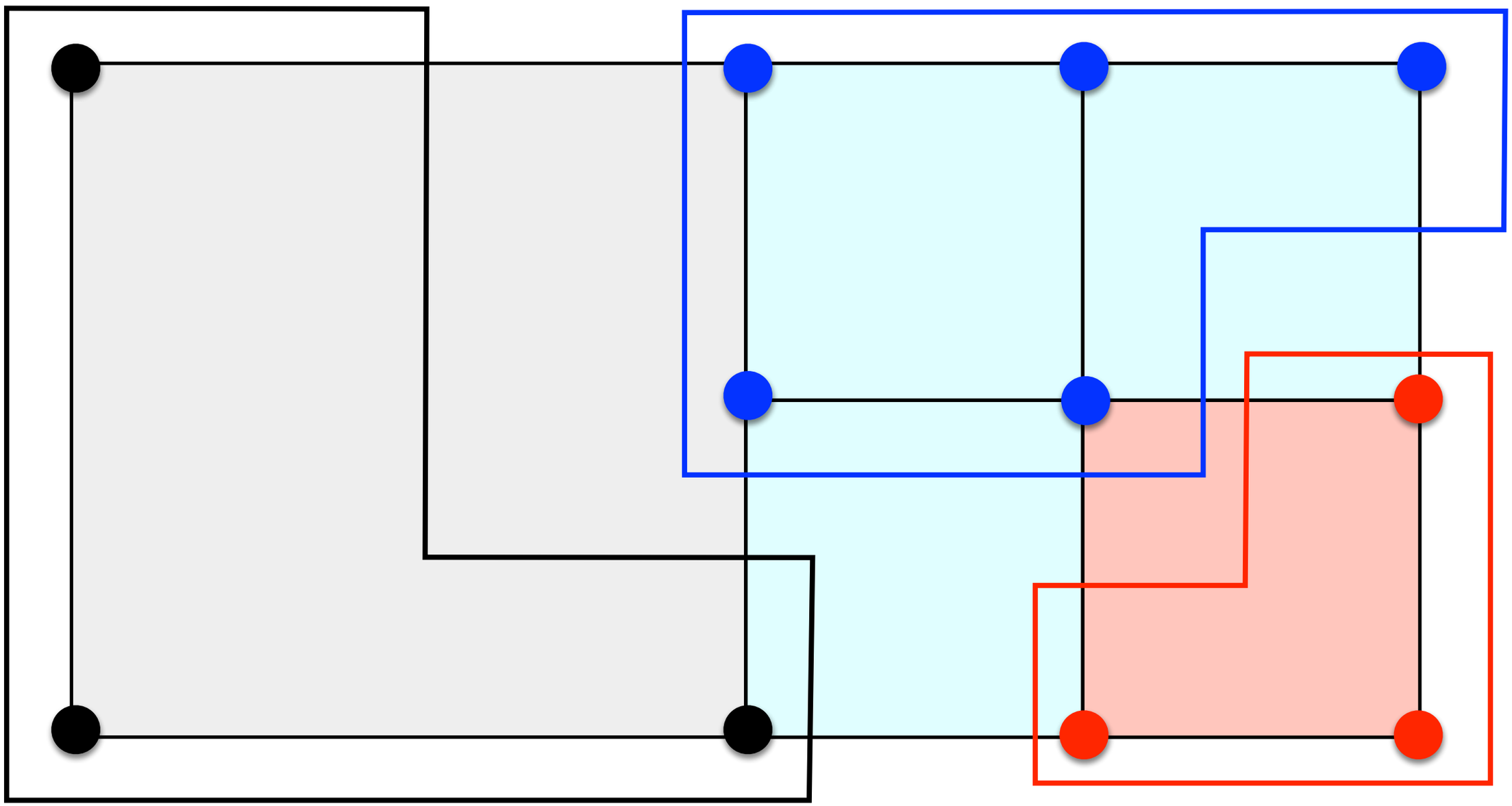


Distribution of degrees-of-freedom: Coloring of DoFs based on cell coloring



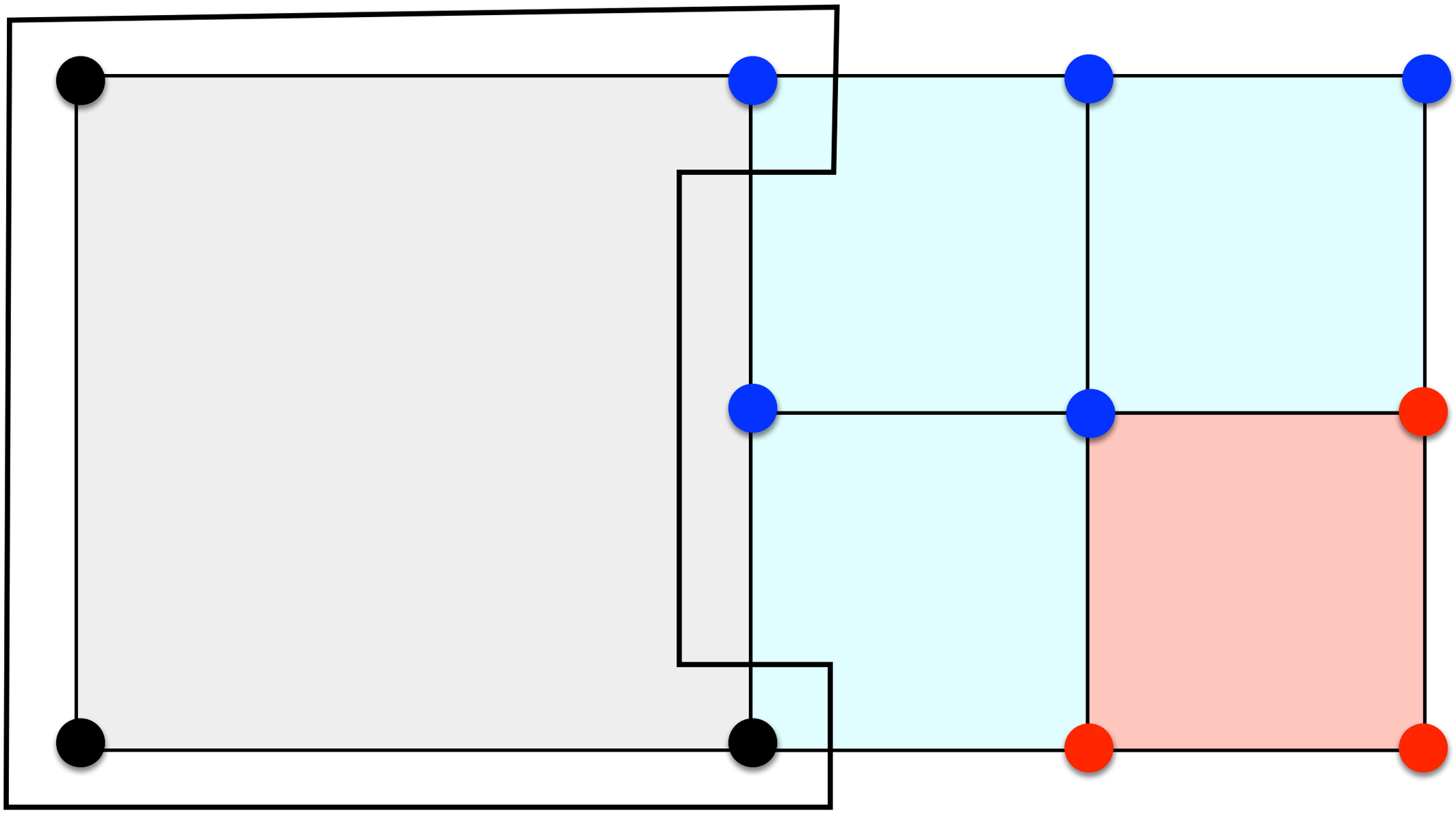
For example, MPI process with lower rank with “own” DoFs at the interface between domains

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



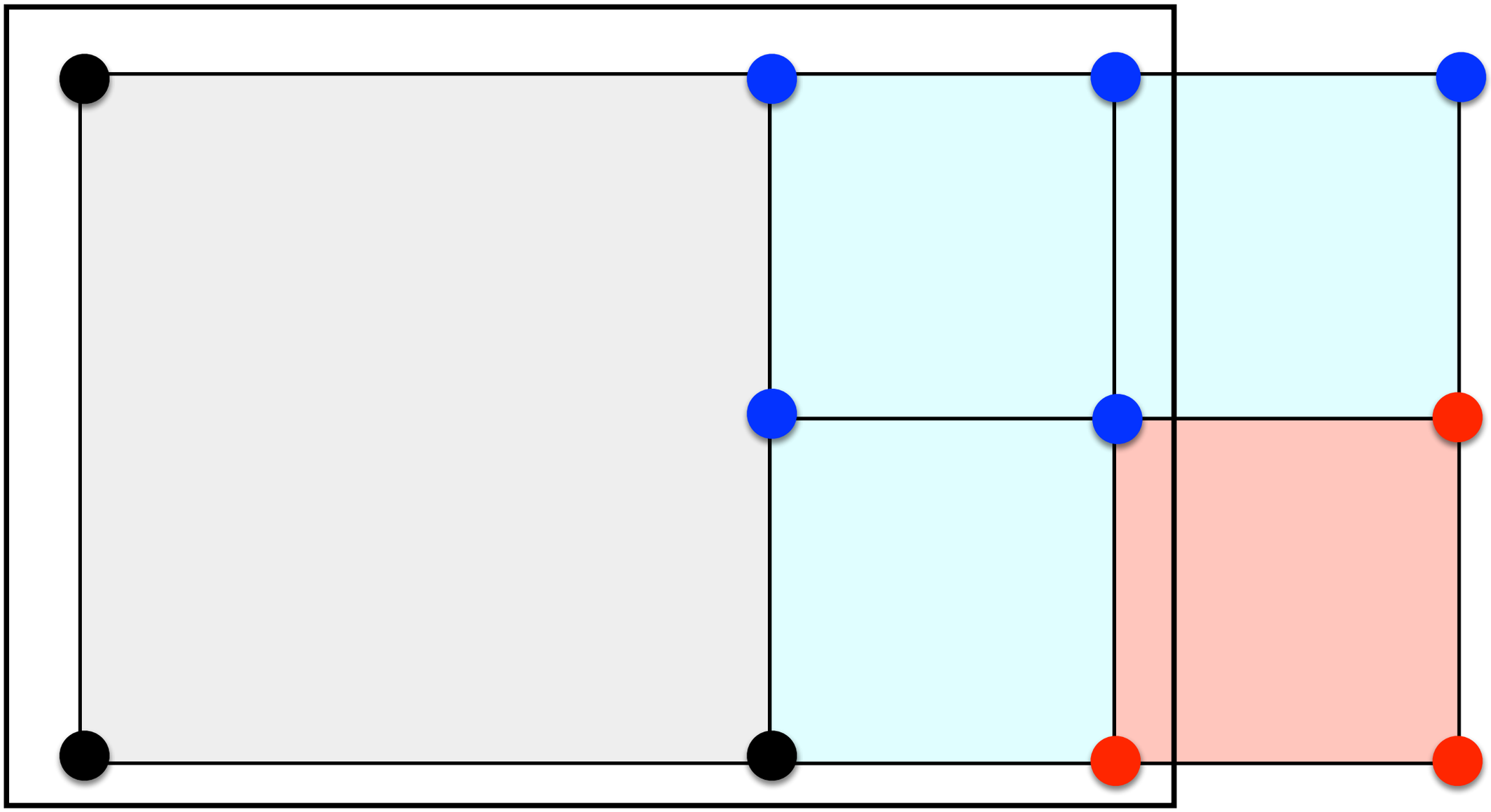
Locally owned DoFs are always one-to-one partition between DoFs and MPI processes (i.e. DoF is attributed to a single MPI process)

Distribution of DoFs:
Locally active degrees-of-freedom (proc=0)



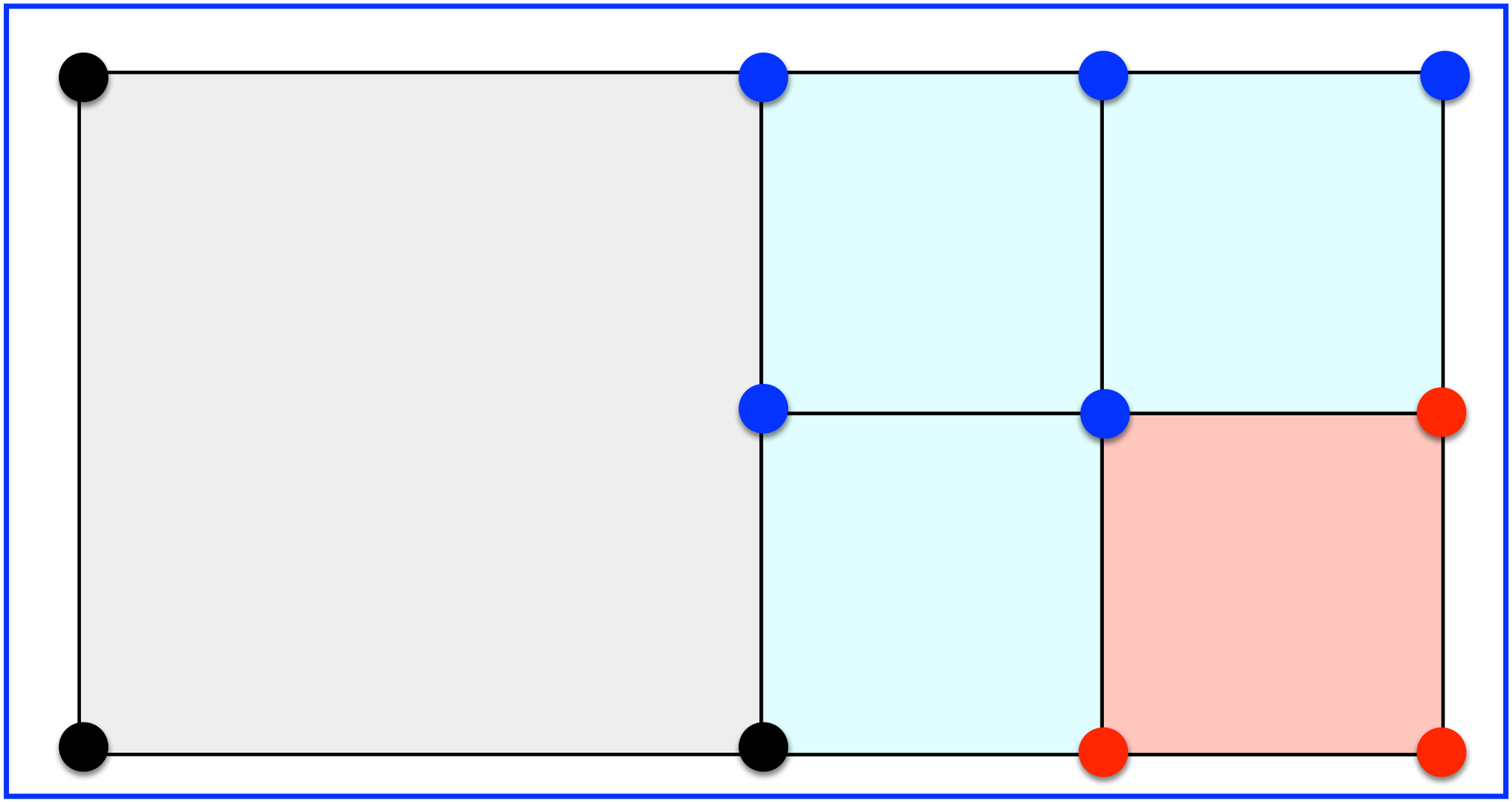
- Required for assembly, data output

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



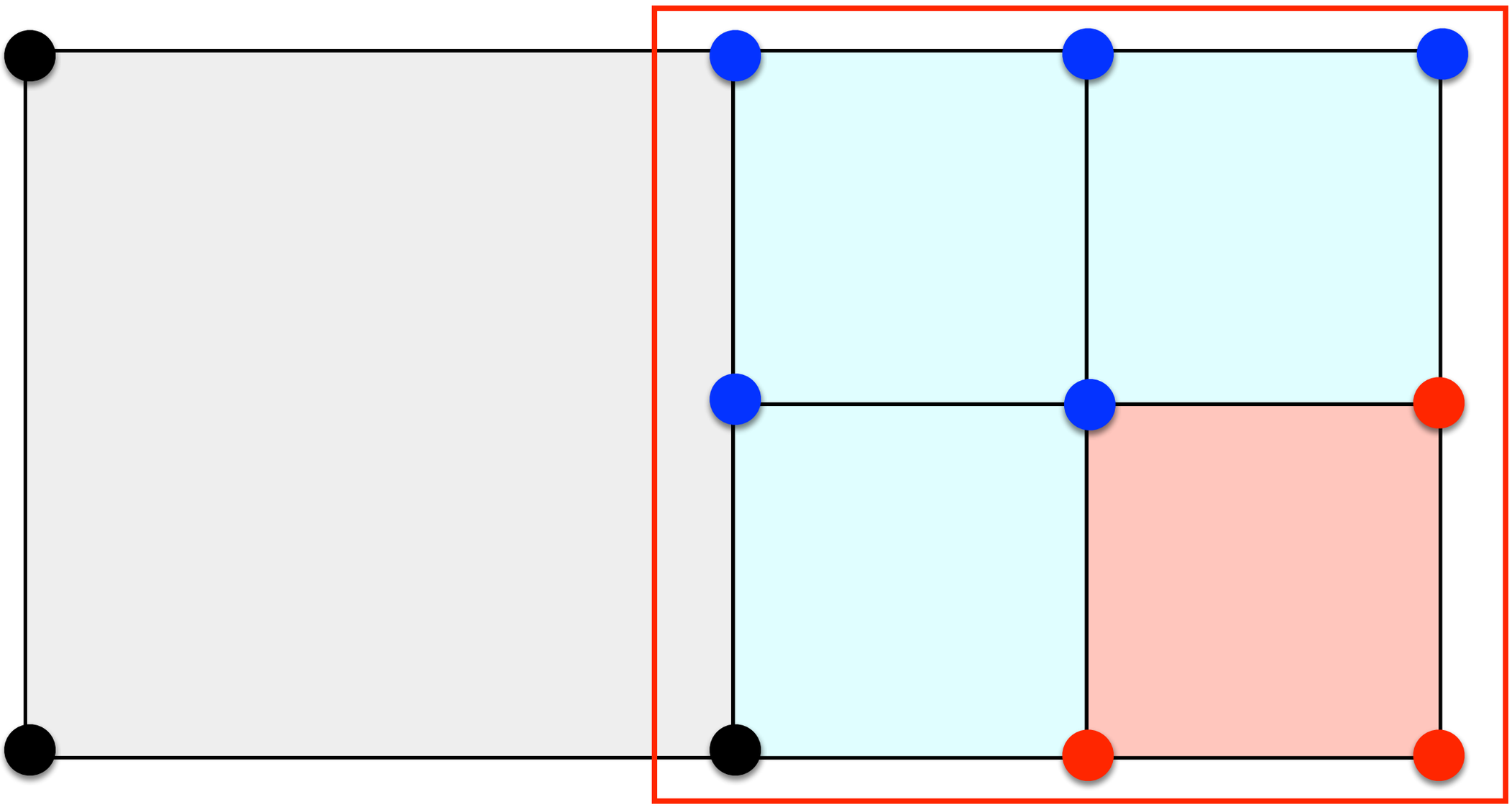
- Required for Kelly error estimator, which evaluates jump in solution gradient across the interface.

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



- Required for Kelly error estimator

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



- Required for Kelly error estimator

Distributed and ghosted vectors

Each MPI process has owned, active and relevant DoFs. Those are subsets of $\{0,\ldots,\mathrm{n\_global\_dofs}-1\}$ and are represented using IndexSet class.

Fully **distributed vectors** (initialized with DoFHandler::locally_owned_dofs()) are used in

- linear algebra solvers (solution vectors, residuals, RHS)

Ghosted vectors using active DoFs are used to

- output/visualization of solution
- computing cell integrals (e.g. error norms)

Ghost vectors with relevant DoFs (DoFTools::extract_locally_relevant_dofs()) are used in

- Kelly error estimator and other place that would require evaluation of jumps across the faces

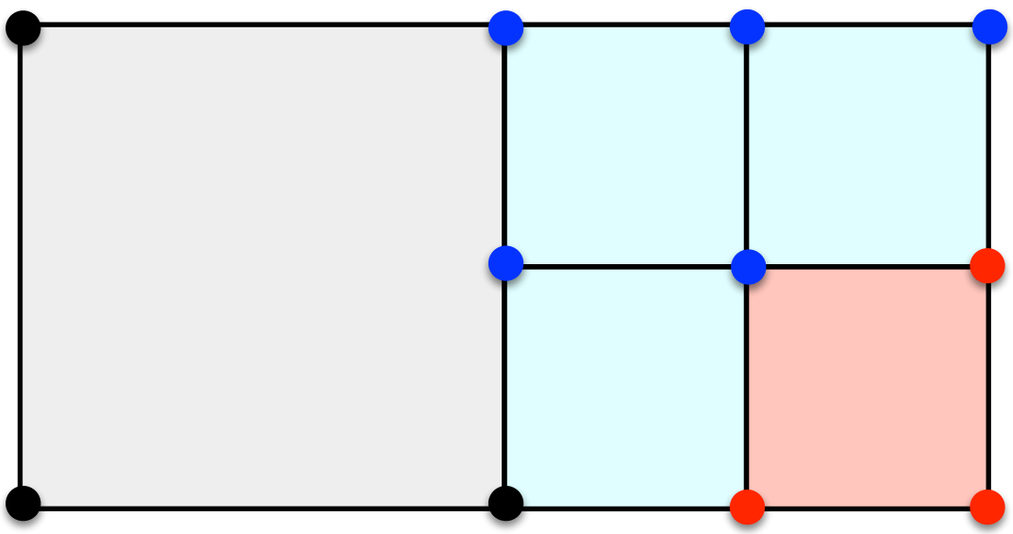
Ghosted vectors and MPI communication

There is no read/write conflict for fully distributed vectors (initialized with locally owned DoFs). This is different for ghost vectors where more than one MPI process can read/write into a shared DoF.

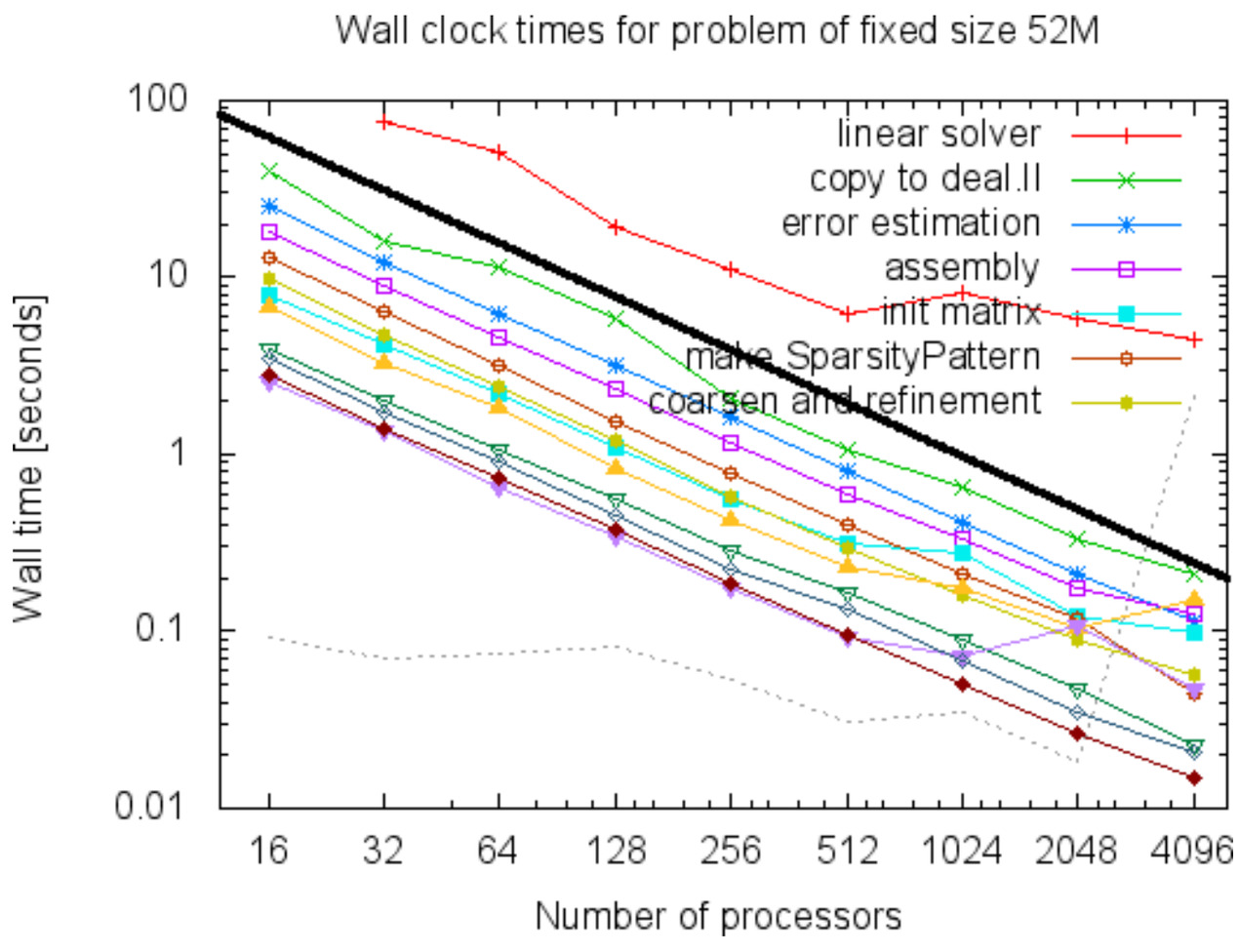
When using Trilinos and PETSc for linear algebra, the RHS vector is initialized as fully distributed vector: Vector(owned_dofs, mpi_communicator):

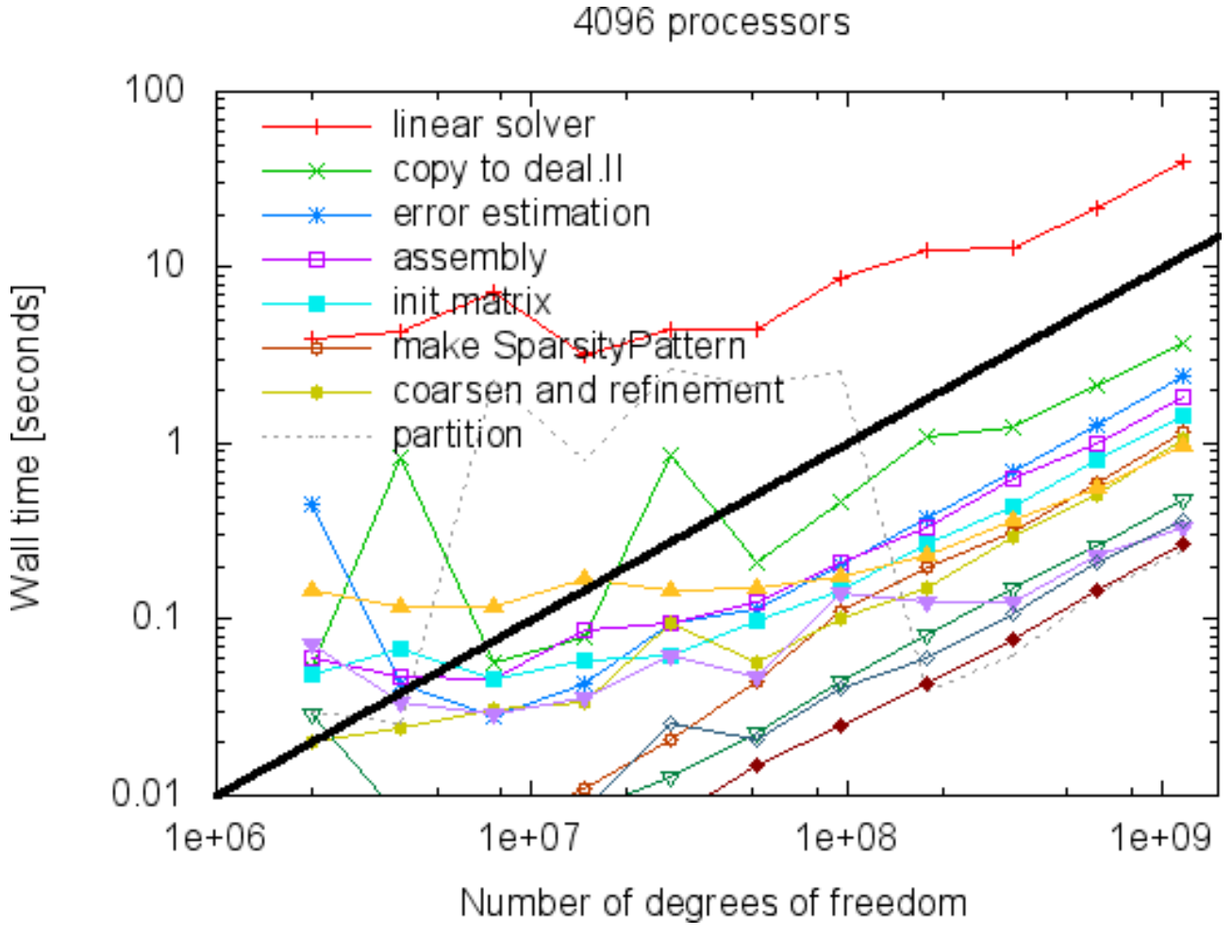
- reading DoF values is allowed for locally owned DoFs only
- writing is allowed everywhere (e.g. during RHS assembly). When all integrals are evaluated, one has to communicate (all at once) contributions to MPI processes that own DoFs on the interface via Vector::compress(VectorOperation::add)

In order to read from ghost entries (e.g. solution), one creates a second vector Vector(owned_dofs, ghost_dofs, mpi_communicator) and uses operator= to copy and communicate data from fully distributed vector.



Step-40


Wall clock times for problem of fixed size 52M
Wall time [seconds]
100
10
1
0.1
0.01
16
32
64
128
256
512
1024
2048
4096
Number of processors
linear solver
copy to deal.II
error estimation
assembly
init matrix
make SparsityPattern
coarsen and refinement


4096 processors
Wall time [seconds]
100
10
1
0.1
0.01
1e+06
1e+07
1e+08
1e+09
Number of degrees of freedom
linear solver
copy to deal.II
error estimation
assembly
init matrix
make SparsityPattern
coarsen and refinement
partition