MHPC
Master in High Performance Computing

Lecture 9: MPI Parallelization, part II (code structure and changes)

Luca Heltai (luca.heltai@sissa.it)

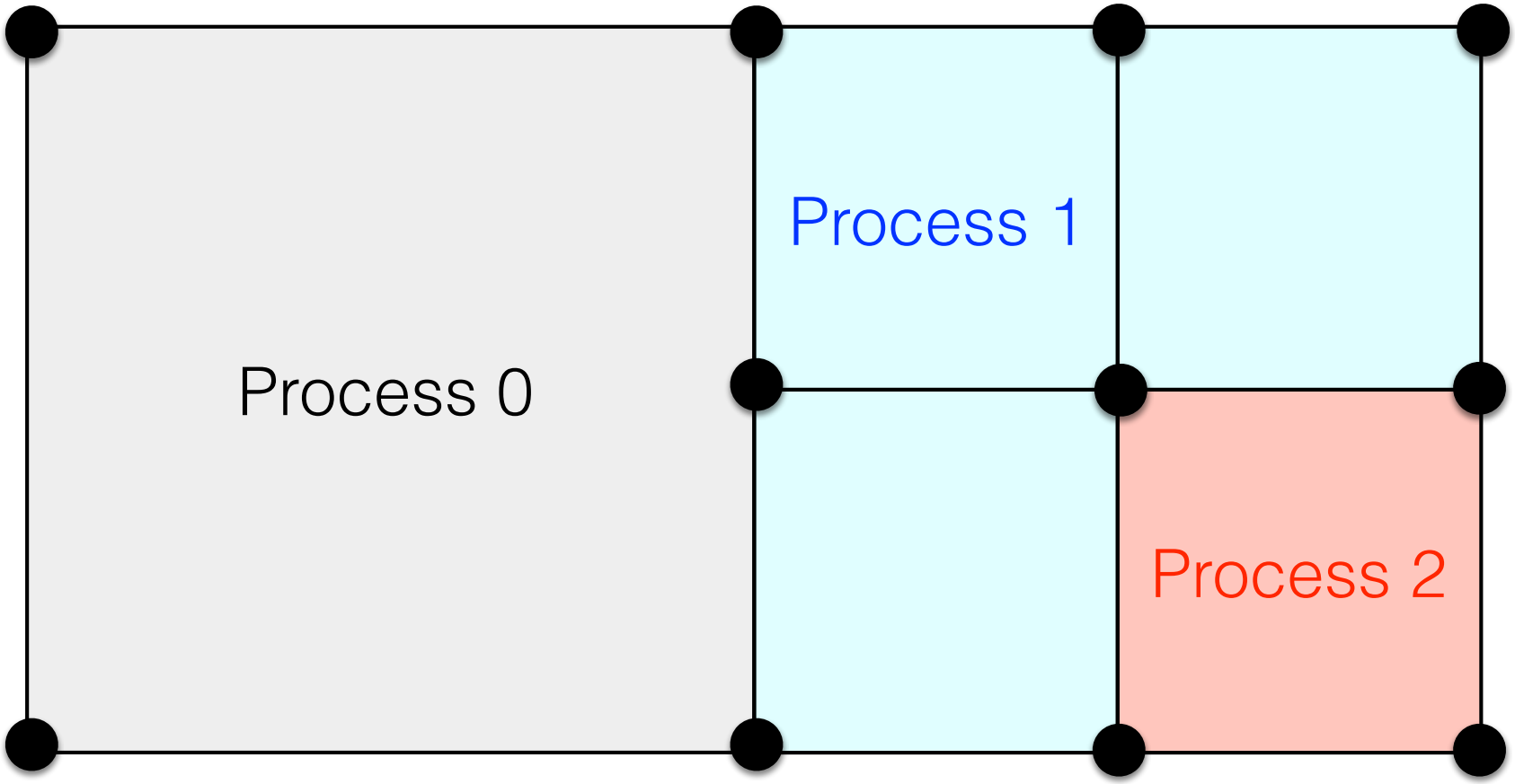
Aims for this module

- 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

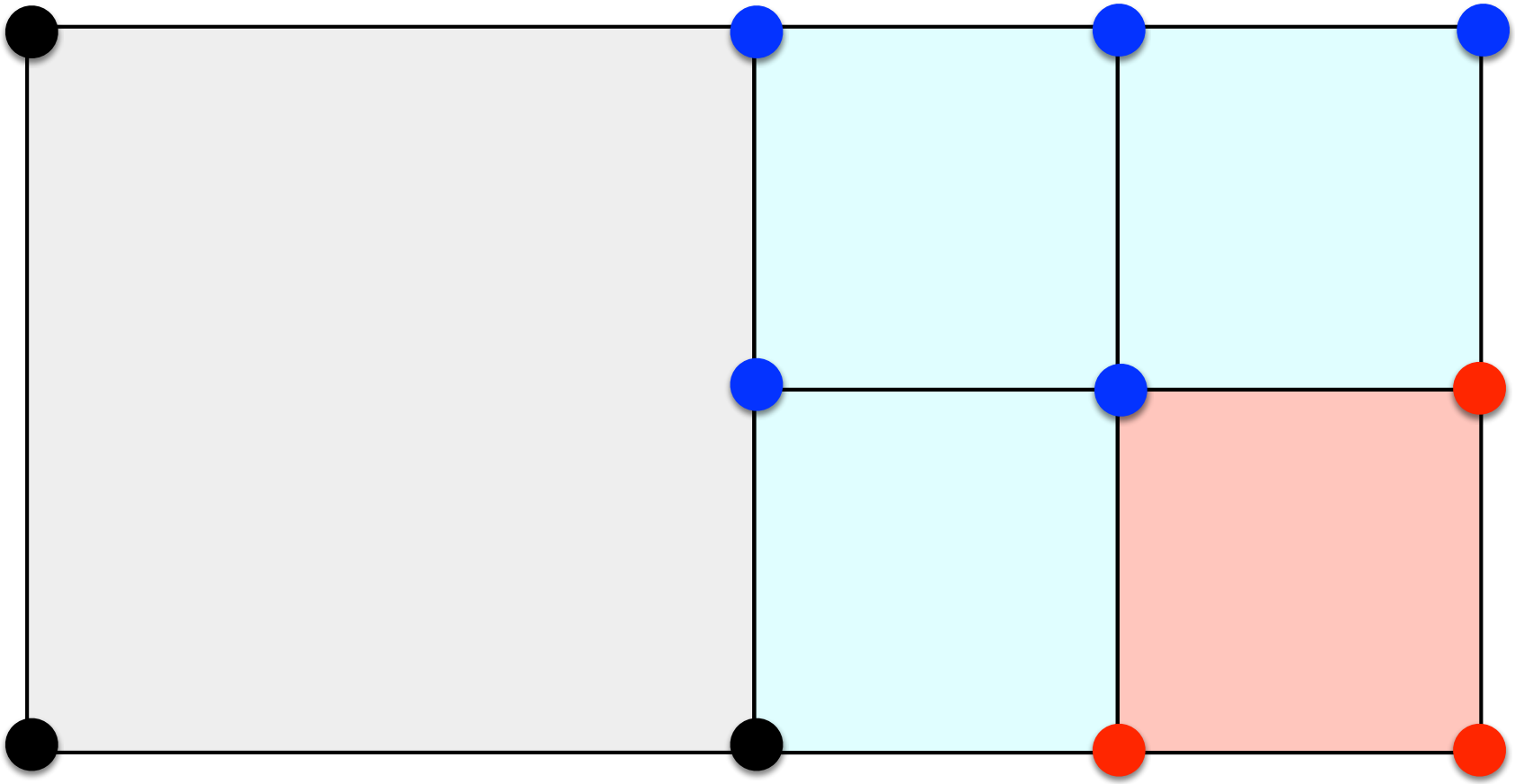
Reference material

- Tutorials
  - 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/developer/doxygen/deal.II/group__distributed.html

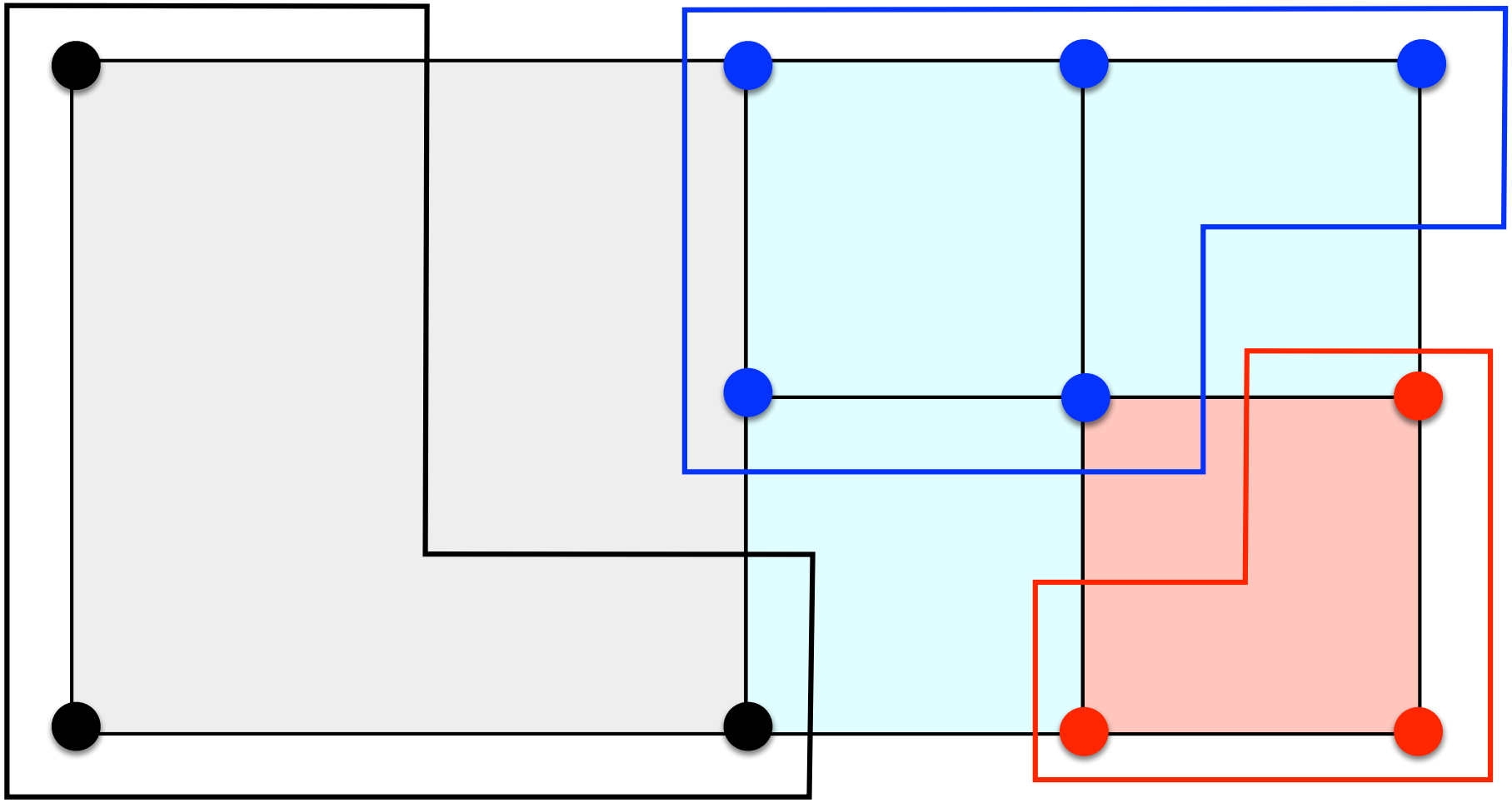
Distribution of degrees-of-freedom: Colouring of cell ownership via graph partitioner



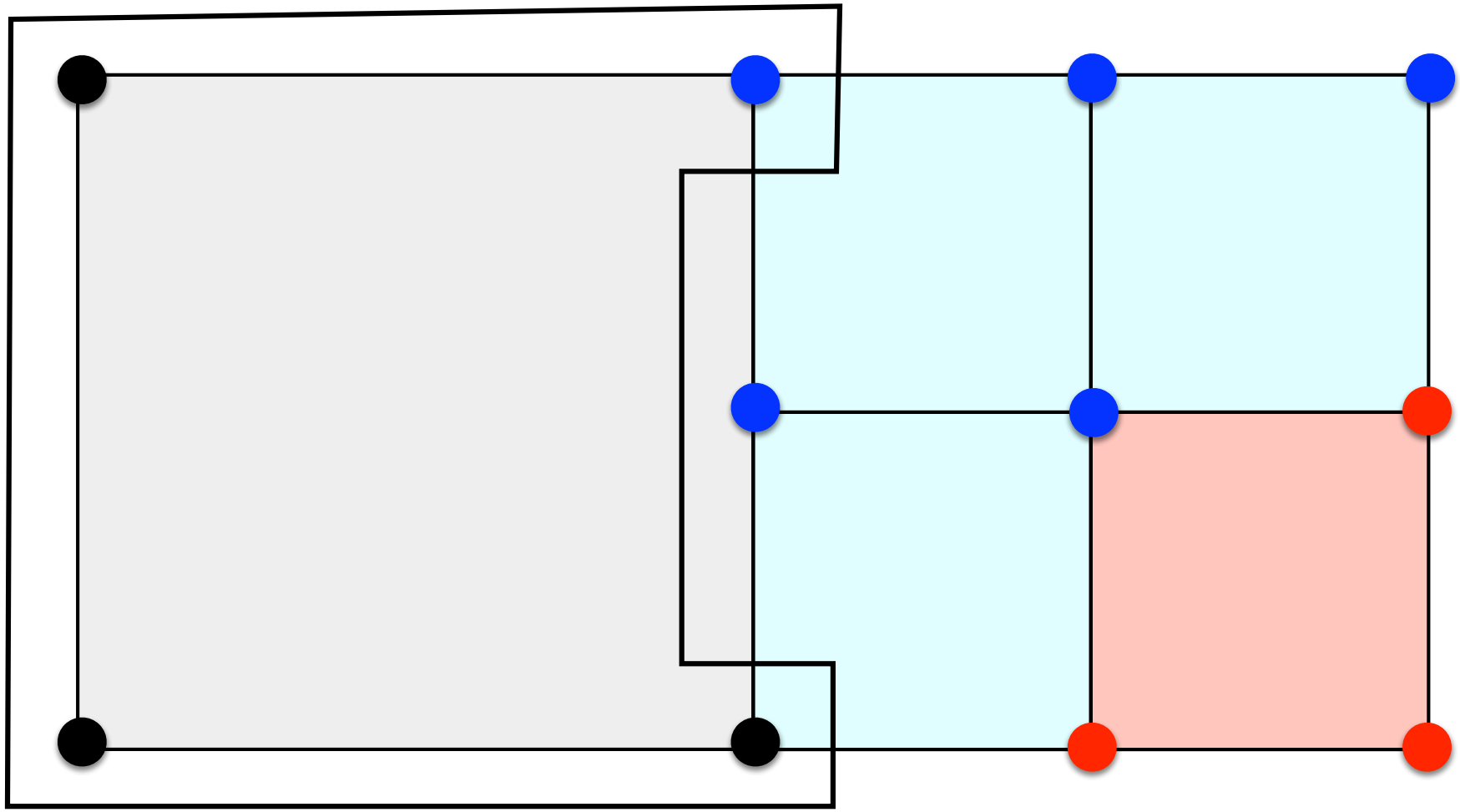
Distribution of degrees-of-freedom: Colouring of DoFs based on cell colouring



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

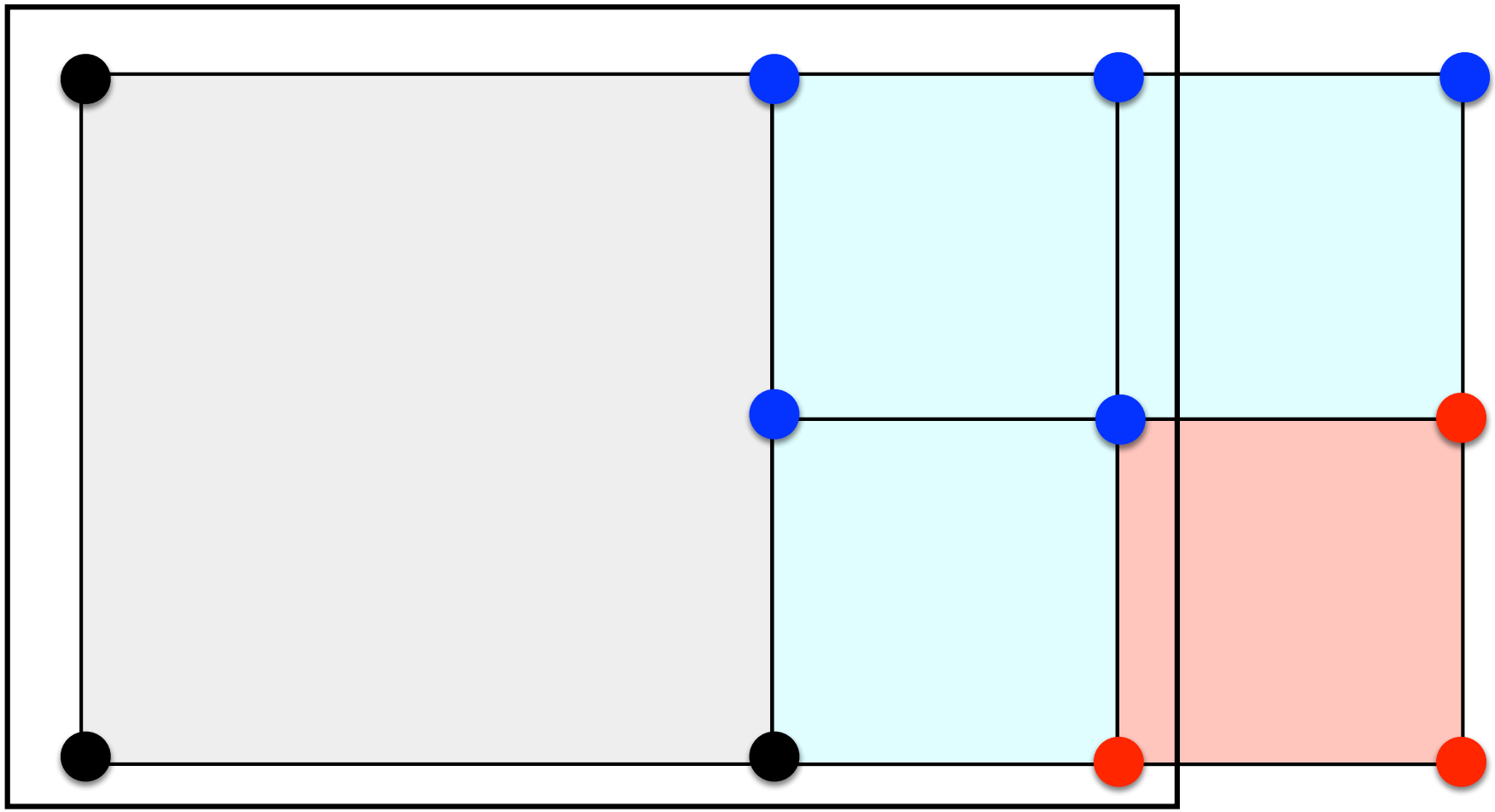


Distribution of DoFs: Locally relevant degrees-of-freedom (process 0)



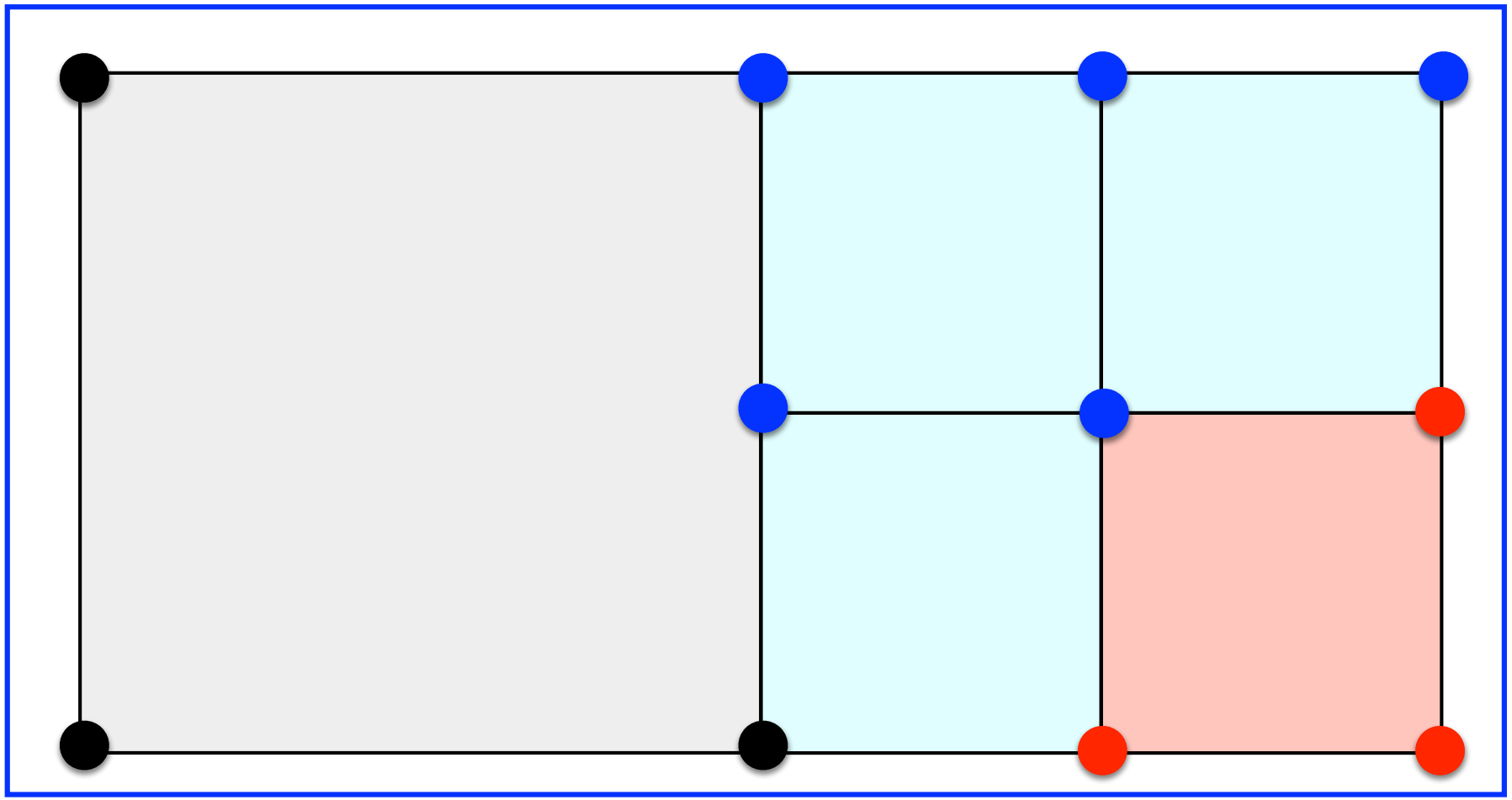
- Required for assembly, data output

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



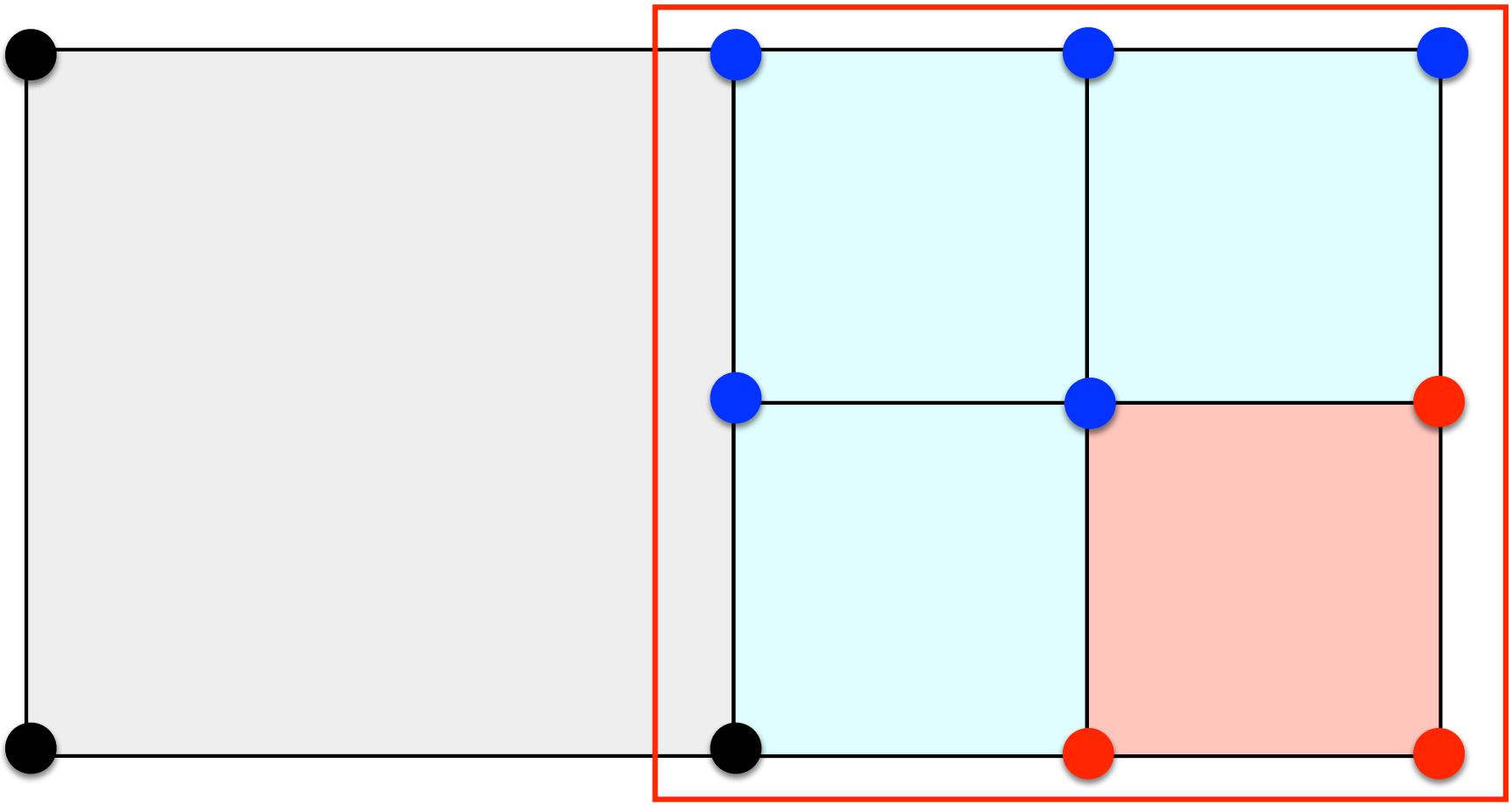
- Required for Kelly error estimator

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

Sketch...

- Each CPU has sets:
  - `owned`: we store vector and matrix entries of these rows
  - `active`: we need those for assembling, computing integrals, output, etc.
  - `relevant`: error estimation
- These set are subsets of $\{0, \ldots, $`n_global_dofs`$\}$
- Represented by objects of type `IndexSet`
- How to get? `DoFHandler::locally_owned_dofs()`, `DoFTools::extract_locally_relevant_dofs()`, `DoFHandler::locally_owned_dofs_per_processor()`, ...

Sketch...

- reading from owned rows only (for both vectors and matrices)
- writing allowed everywhere (more about `compress` later)
- what if you need to read others?
- Never copy a whole vector to each machine!
- instead: ghosted vectors

Sketch...

- read-only
- create using
  `Vector(IndexSet owned, IndexSet ghost, MPI_COMM)`
  where ghost is `relevant` or `active`
- copy values into it by using operator=(Vector)
- then just read entries you need

Compressing

- Why?
  - After writing into foreign entries communication has to happen
  - All in one go for performance reasons
- How?
  - `object.compress (VectorOperation::add);` if you added to entries
  - `object.compress (VectorOperation::insert);` if you set entries
  - This is a collective call
- When?
  - After the assembly loop (with `::add`)
  - After you do `vec(j) = k;` or `vec(j) += k;` (and in between add/insert groups)
  - In no other case (all functions inside deal.II compress if necessary)! (this is new!)

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);
```