Bundesanstalt für Wasserbau
Federal Waterways Engineering and Research Institute

UnTRIM-related activities at BAW Karlsruhe

www.baw.de

Regina Patzwahl, Jacek A. Jankowski, Thomas Lege

Bundesanstalt für Wasserbau
Federal Waterways Engineering and Research Institute

Improving and optimising UnTRIM MPI library

www.baw.de

Jacek A. Jankowski

UnTRIM version applied

- **UnTRIM „compiled June 2009“**– the last version before sub-grids
- computational core MPI-parallelised with minimal changes in the code
- almost generic *User Library* adapted for typical river flows, waterways...

- Applied daily for larger, high resolution river models
  - 2-3 millions of base polygons, 64-128-256 partitions

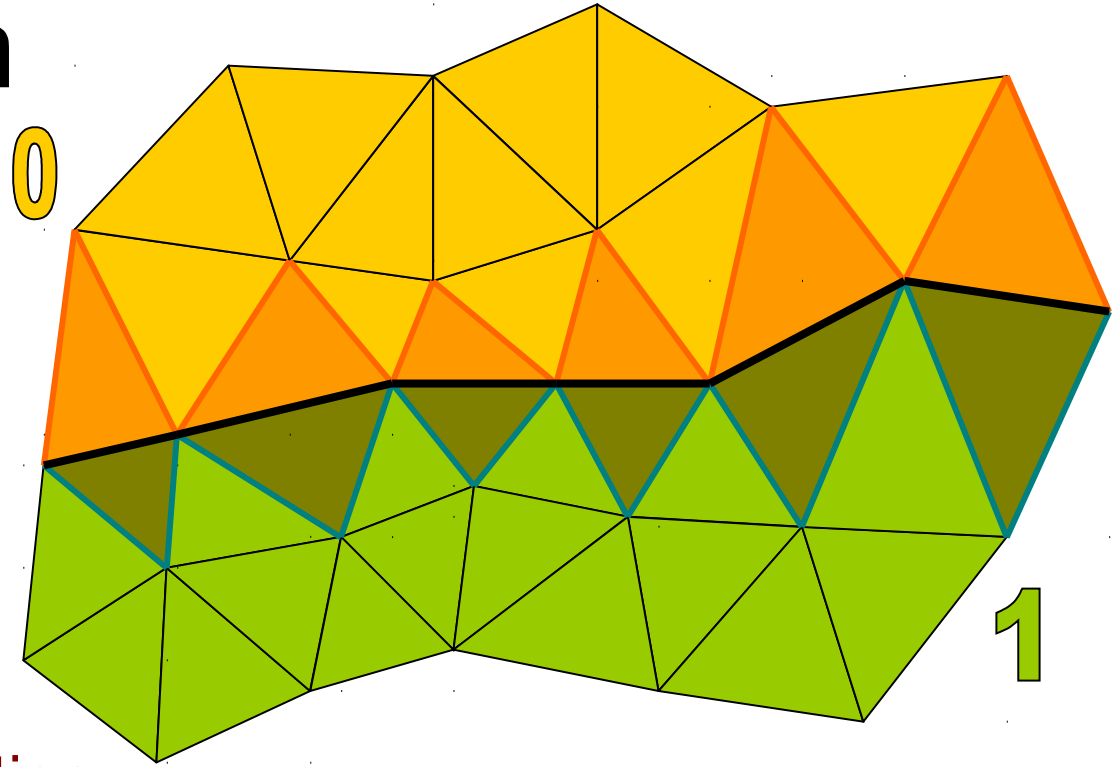
Parallel library: work aims 2009-10

- Performance gain – try to make it fit for a large number of processors
- Easy workflows – simplify steps in pre- and post-processing
- Suspected improvements – just trying

Addressed topics

- Asynchronous message passing
- Partitioning:
  - heterogeneous meshes
  - weighting
- Parallel I/O:
  - removing merging and restart partitioning
  - optimisations via input mesh sorting

Communication patterns

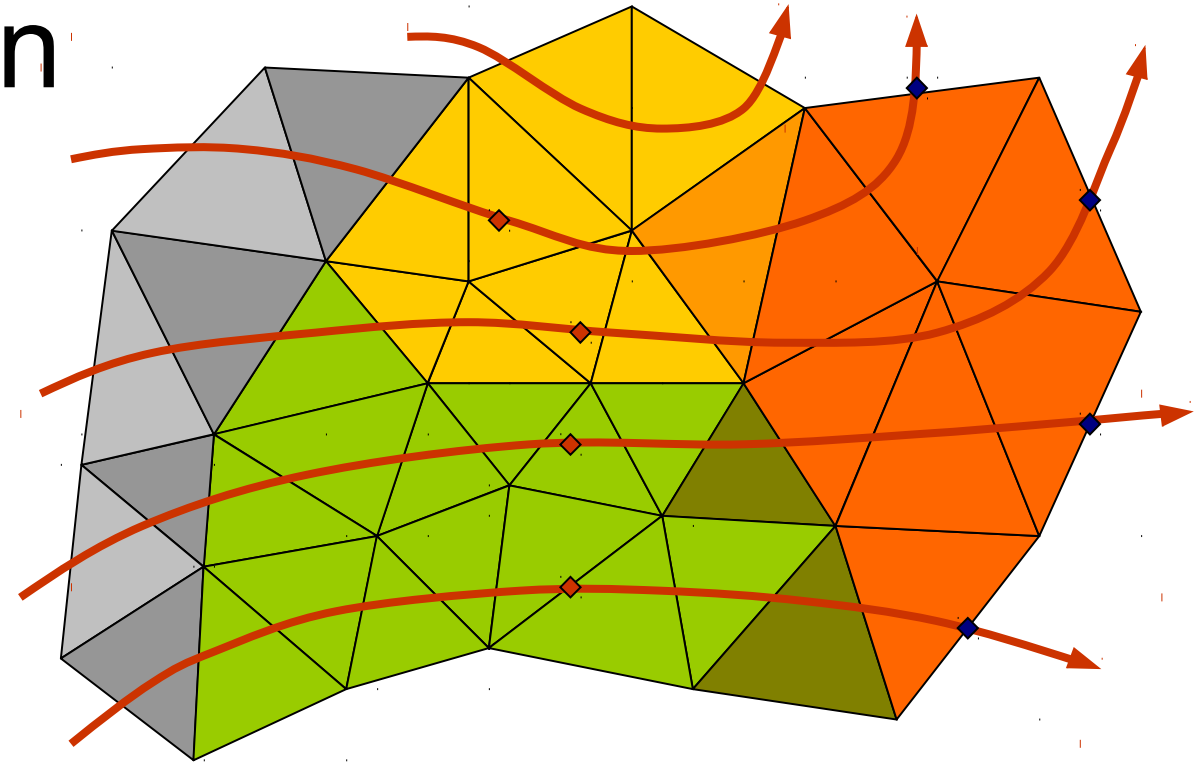


FV/FD – Eulerian

- swapping halo values via point-to-point communication
- two pairs of buffered MPI_Send and MPI_Recv joined to **MPI_SendRecv** for fields of elementary types

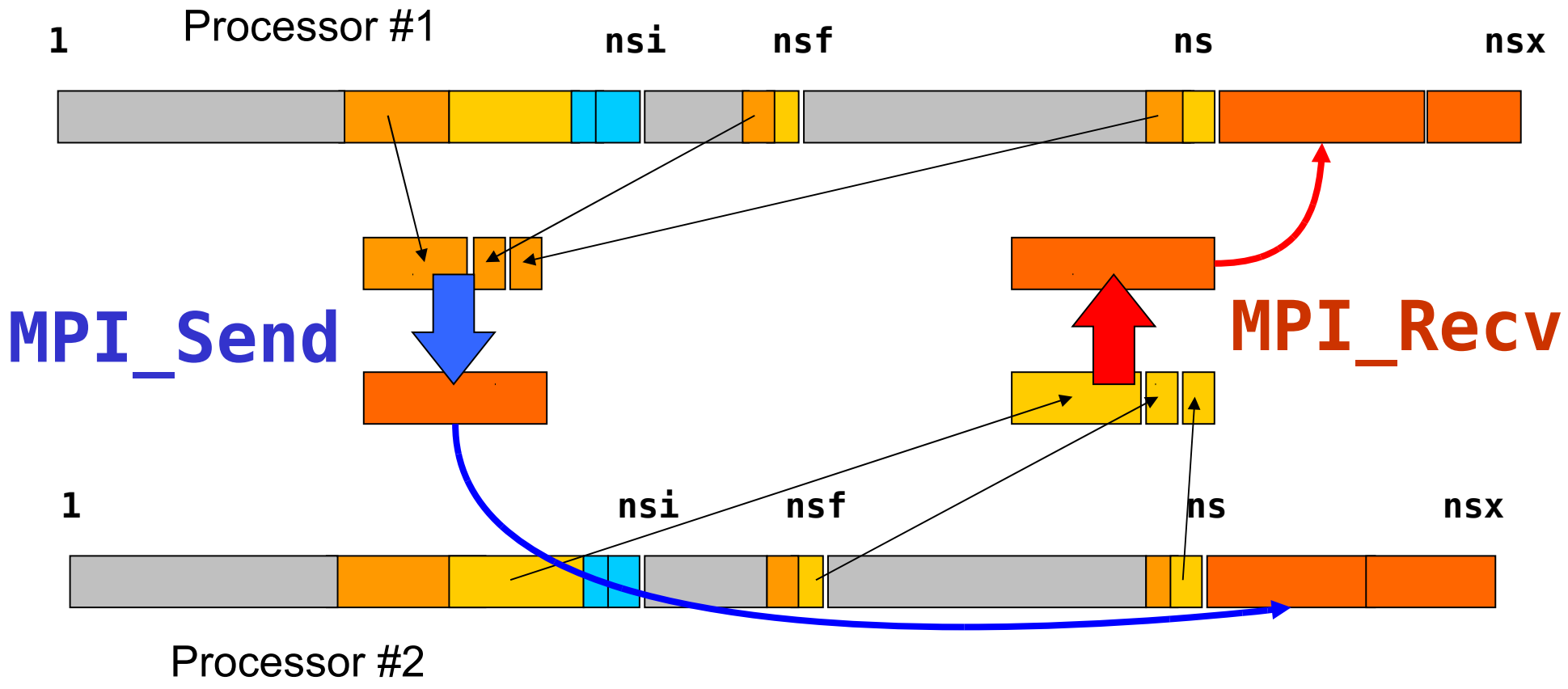
Communication patterns

Advection – Lagrangian



- streamline tracking treating tracebacks as autonomous objects – global communication
- Using **MPI_AllToAll** for objects being **MPI_Type**-s

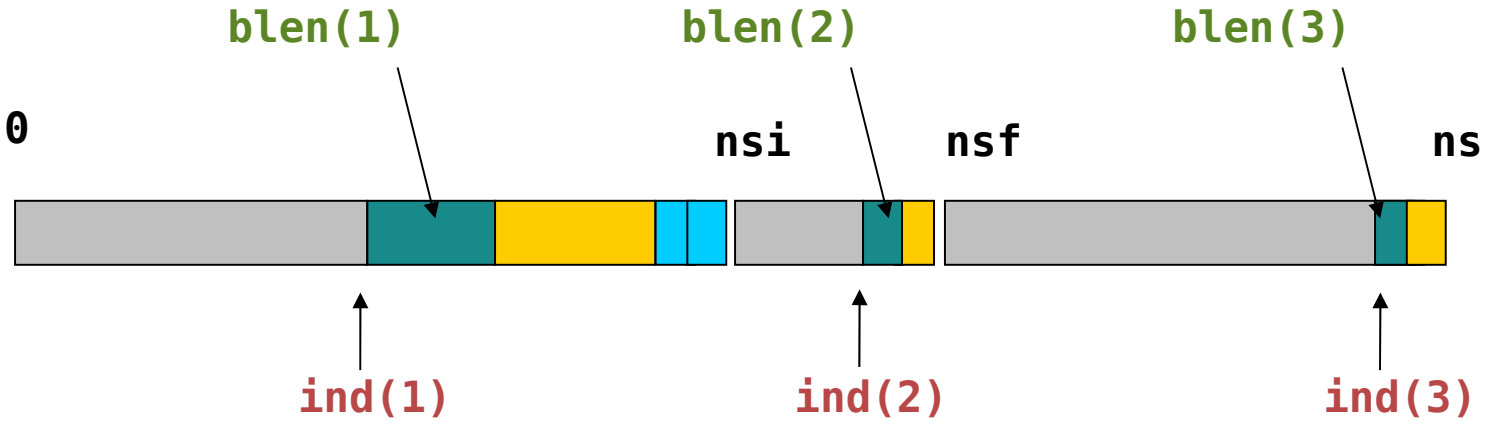
MPI_Send, MPI_Recv with buffers



Swapping data with buffers

- Work in a loop over all communication partners
- Pack data to be sent into the **send buffer**
- Exchange data with **`MPI_SendRecv`**
- Copy the data from the **receive buffer** to the field
- Return...

MPI_Type_Indexed




```
CALL MPI_Type_Indexed (3,blen,ind,MPI_INTEGER,newtype,ier)
CALL MPI_Commit (newtype, ier)
CALL MPI_Send (field,1,newtype,...)
CALL MPI_Free (newtype,ier)
```

Asynchronous communication

- Synchronous – **`MPI_Send, MPI_Recv`**
  - *no computing is being done until the message passing completes*

- Asynchronous – **`MPI_ISend,MPI_IRecv,MPI_Wait`**
  - *sending and receiving messages can be overlapped with computing „in the meanwhile“*

Hiding the buffering

- Work in separate loops over all communication partners
- Start receiving data to the receive buffer with **`MPI_IRecv`**
- Pack field data to be sent into send buffers
- Start sending data with **`MPI_ISend`**
- Test with **`MPI_Wait`** `if` *receiving* messages is completed
- Move the data from the receive buffers to the field
- Test with **`MPI_Wait`** `if` *sending* messages is completed
- Return...

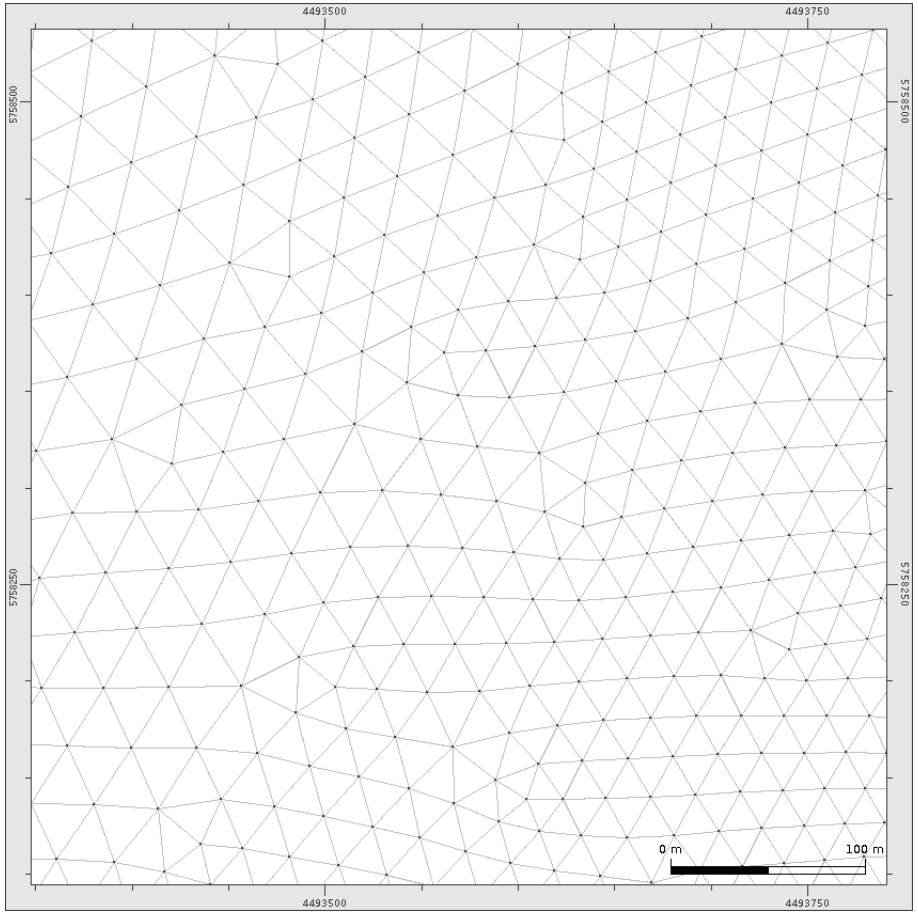
Asynchronous communication

- Results of „hiding the buffering“, e.g. the model of the Elbe River by Coswig:
  - the global swapping communication time (sum for all ranks) reduces by 13%-23%
  - better results for larger rank numbers (range 32-128)
  - but the global computation time reduces only by 1%!

Partitioning

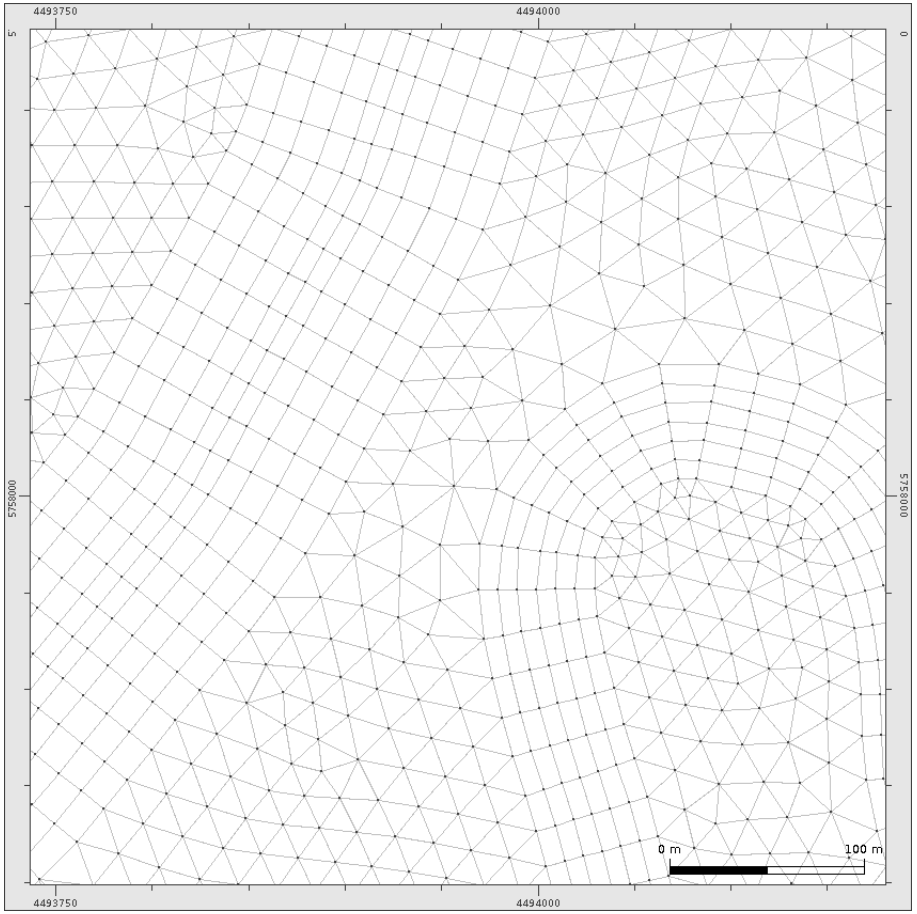
- Moved from METIS 4.0 (1998) to *5.0pre2* (2007) (METIS library – thanks to Karypis et al.)
- Small modifications (C functions interfaces) necessary
- Correct C-binding in Fortran2003 possible

- Allows:
  - Partitioning of heterogeneous meshes
  - Weighting of partitions with weights per polygon



Homogeneous mesh

Direct partitioning



Heterogeneous mesh

Additional pre-processing before partitioning removed

Weighted partitioning

- Metis per default:
  - balances the mesh partition size according to the number of 2D polygons → computation balanced
  - reduces the length of interfaces between partitions to a minimum → communication minimised
- Other criteria – weights per polygon – possible

Weighted partitioning

- Reading and interpolating *a characteristic water level*
- Creating a run-relevant 3D mesh while partitioning

- Tried weighting with the number of:
  - polygons (~***ne***) – default
  - wet cells in the water column (~***n3e***)
  - wet cells in the water column plus one (~***n3e+ne***)
  - ...other criteria straightforward to implement

Weighted partitioning

- Results not encouraging yet...
  - in theory, weighting should take into account the *computational effort* per polygon
  - but be checked with the growth of the length of the interfaces - *communication amount*

  - presently all runs balanced with polygons (**~*ne***) are better than taking wet cells (**~*n3e*, ~*n3e+ne***)
  - balance the communication amount instead...?

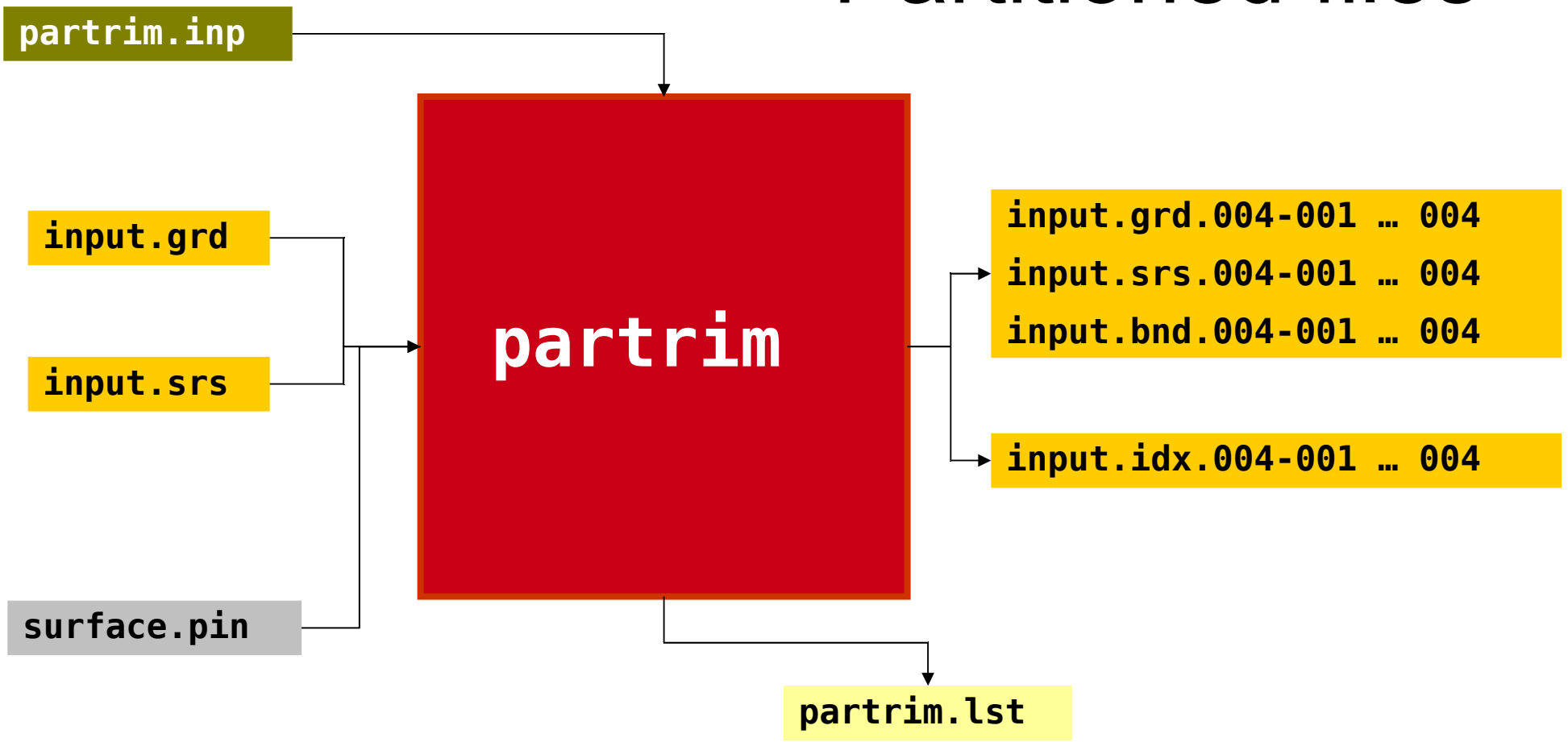
Parallelising I/O

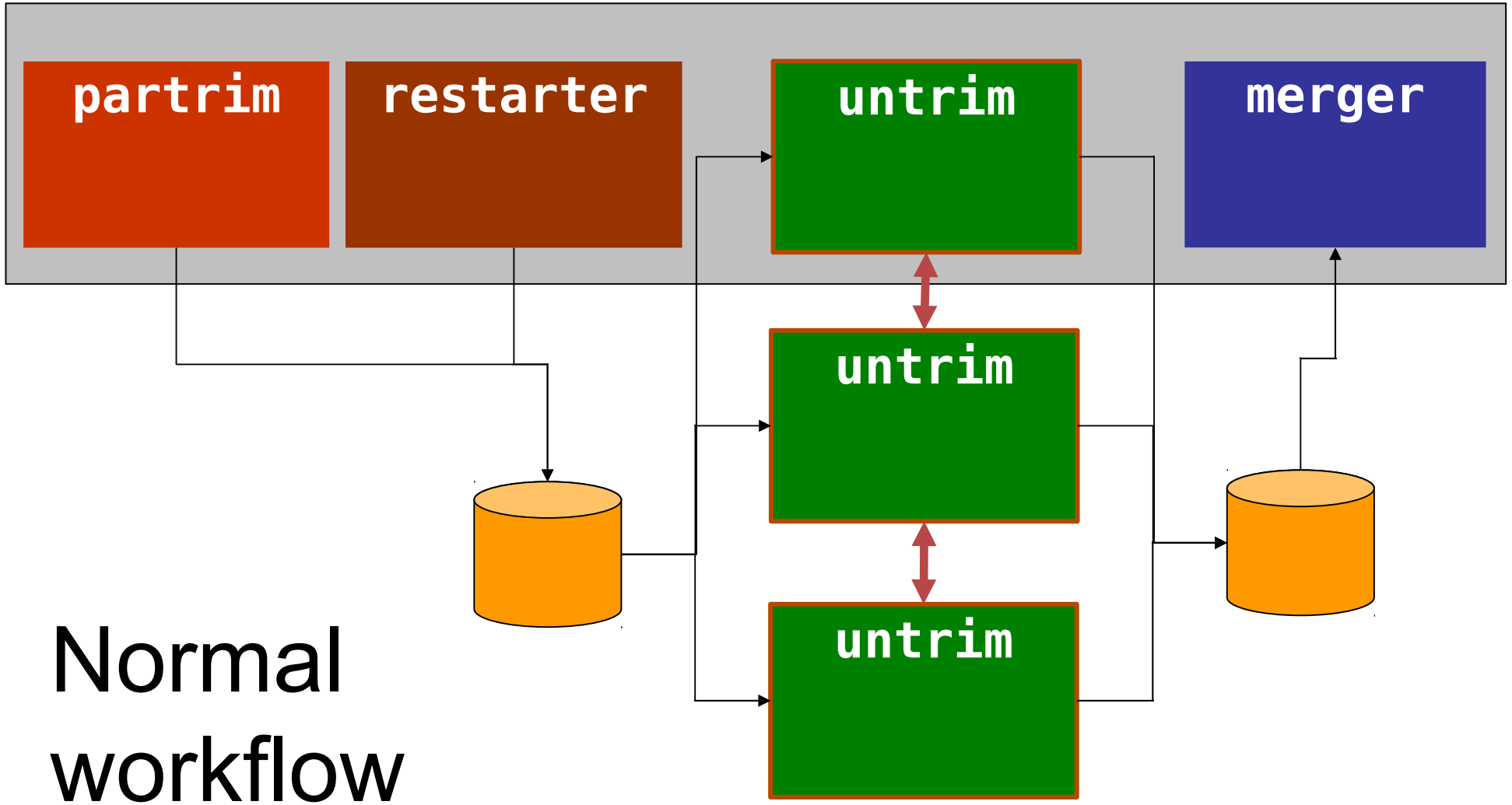
- Operational systems per default assume that only one process accesses a given single file
- The traditional method of treating I/O in message-passing programs:

  → *each rank reads and writes its own set of files*

- MPI 2 allows ordered I/O-operations of a set of ranks accessing a single file – **MPI-I/O**

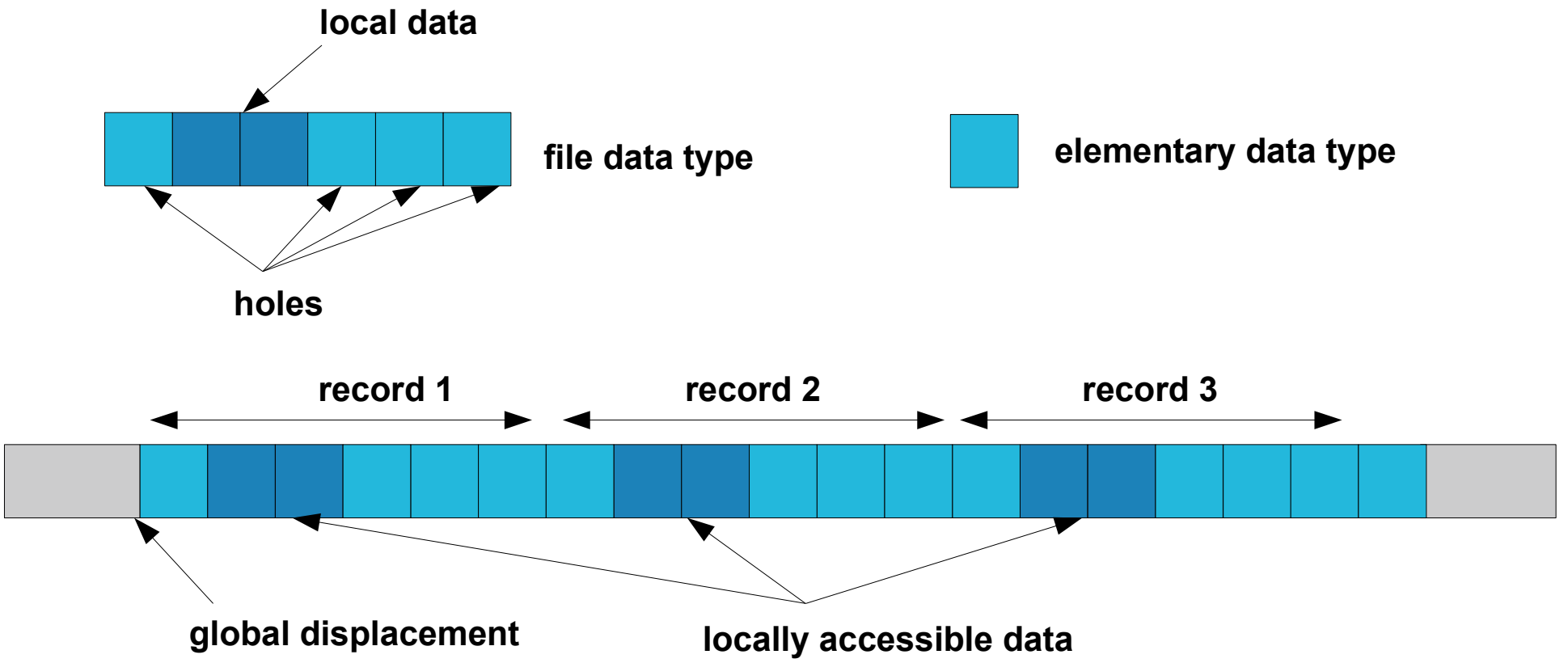
Partitioned files





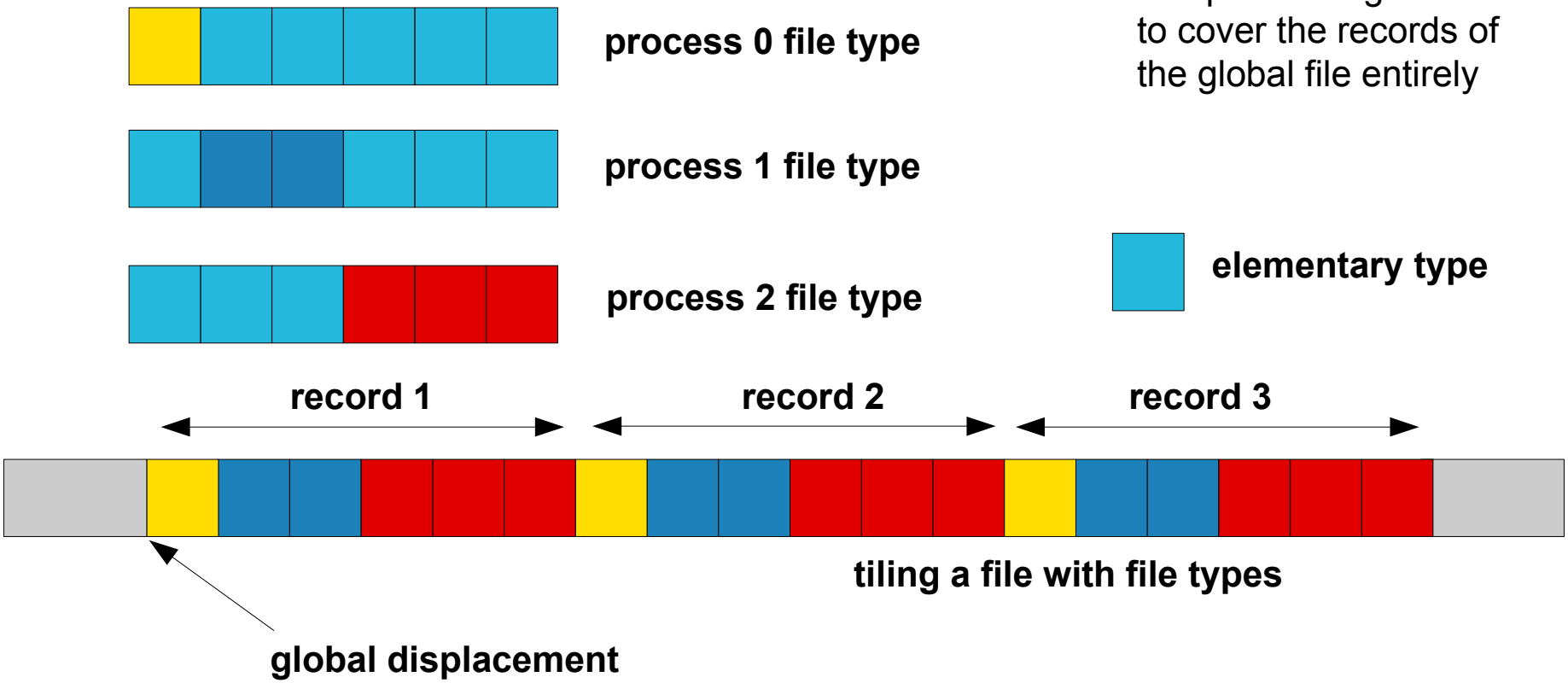
Normal workflow

Single rank view of the file



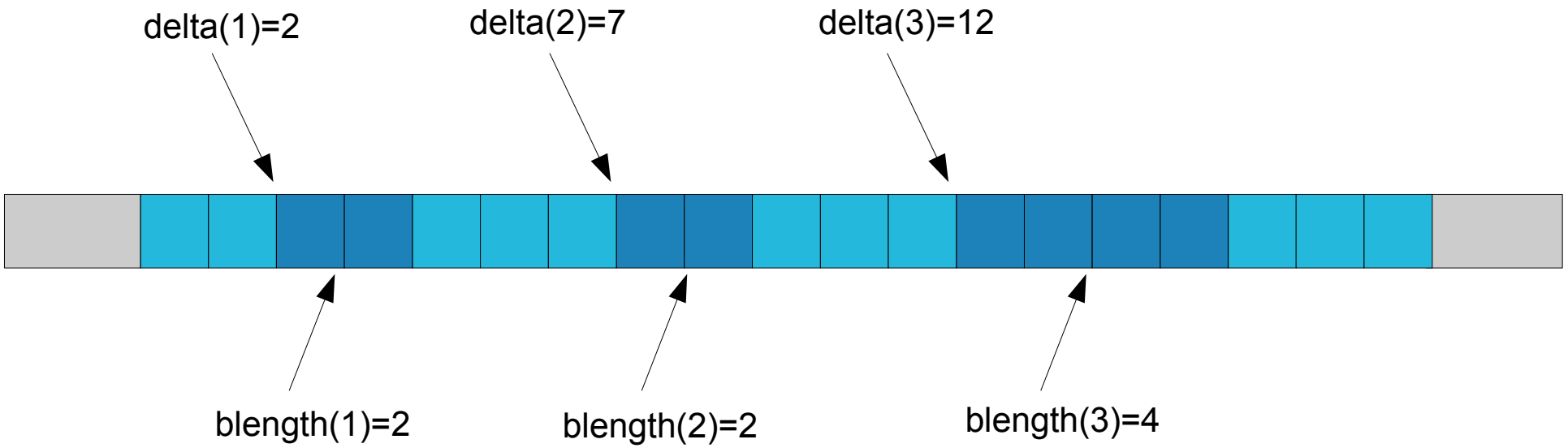
Complementary file types

File types of three ranks complementing each other to cover the records of the global file entirely



Committing a file view

A non-decreasing displacement-blocklength description of the length len=3




```
CALL MPI_Type_Indexed &
   & (len,blength,delta,MPI_REAL8,view,ier)
CALL MPI_Type_Commit(view,ier)
```

Writing a MPI-I/O file

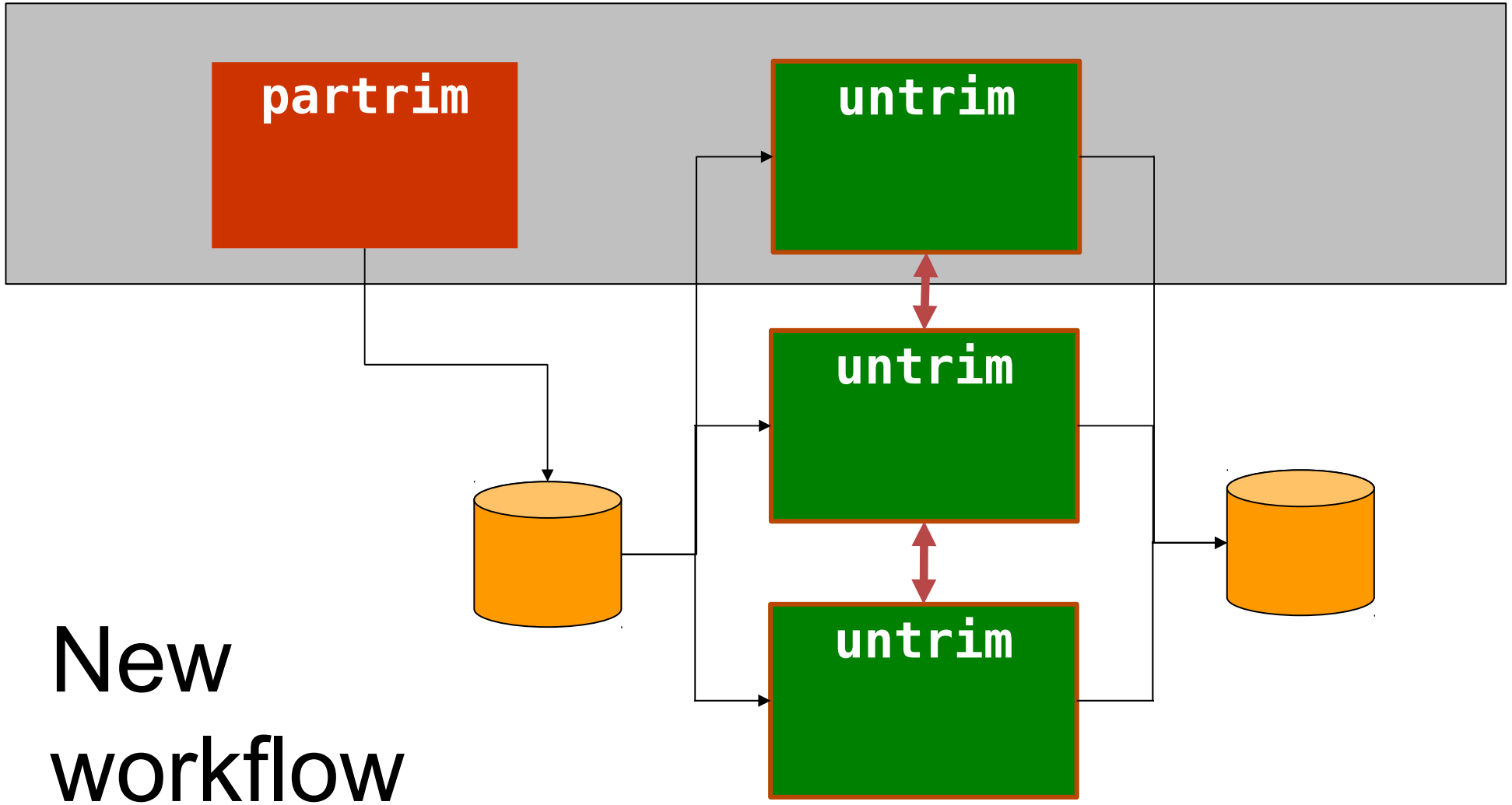
```
CALL MPI_File_Open &
       & (MPI_COMM_WORLD,TRIM(mpiio_restart_file), &
       & IOR(MPI_MODE_CREATE,MPI_MODE_WRONLY), &
       & MPI_INFO_NULL, fhrst, ier)

CALL MPI_File_Set_View &
       & (fhrst, idisp, MPI_REAL8, view, 'native', &
& MPI_INFO_NULL, ier)

CALL MPI_File_Write_All &
       & (fhrst, iobuffer, lenbuf, MPI_REAL8, &
& MPI_STATUS_IGNORE, ier)
```

Fortran pitfalls

Only *direct-access binary files* possible in Fortran

In the *serial* case:

```
OPEN (ini, FILE=TRIM(get_initial_file(),
           STATUS='old',
           FORM='unformatted',
           ACCESS='direct',
           RECL=reclen*dp_length)
READ (ini, REC=1) t, e(1:ne), u(1:n3s)
```



New workflow

Optimising MPI-I/O

- Using MPI-I/O removed the necessity of merging result files and partitioning of restart files
- However – mostly no gain in I/O-performance compared to the usage of partitioned files *during the parallel run*
- Suspected:
  - extermely long displacement-blocklength descriptions of file views in partitions (3D!)
  - overlapping of file views for edges, sides

Re-sorting the global mesh

- The file view descriptions represent relations between the global and local numbering of mesh objects
- These descriptions can be drastically shortened by re-sorting the original global mesh *in the sequence of partitions*
- Two arts of file view descriptions available – with and without overlapping on interfaces

Optimised MPI-I/O

- Test done on „normal“ file systems – i.e. *no hardware acceleration for MPI-I/O*

- Using sorted meshes speeds MPI-I/O operations 2-8 times
- Better gains for smaller number of processors / larger files
- Removing overlapping views – almost(?) no effect

Optimised MPI-I/O overhead (1)

- **The original meshes sorted so that they remain correct UnTRIM meshes**
  - All the index ranges remain meaningful
- We must use nondecreasing displacement-blocklength file view description
- Negative effect by a single file view per field:
  - I/O must be buffered (up to 2-3% loss...)

Optimised MPI-I/O overhead (2)

- **The restart files must be sorted before the run as well**
- **The results are available on sorted meshes**

- Makes comparisons of results difficult
- Solutions:
  - a small serial program to transfer data between the original and sorted meshes – in both directions
  - do not sort vertices and use vertex-oriented visualisation software (Tecplot)

MPI-I/O assessment

- **Advantages**
  - Partitioning and merging large result files obsolete
  - Simplifies workflows

- **Disadvantages**
  - In optimised form requires an awkward mesh sorting
  - Complicates workflows

Summary

- Communication
  - Balance against workload remains an open problem!
  - Asynchronous communication – OK
  - MPI data types for communications – not done

- Partitioning meshes
  - improved, but still using a lot of files

- MPI-I/O
  - to be assessed / decided using parallel data systems

Possible future work directions

- Working on partitioning software
  - parallel partitioning execution
  - simplifying the usage
- Working on the balancing of the parallel runs
  - especially communication
- Parallel I/O:
  - Hardware-specific improvements
  - Using specialised I/O-libraries for data / visualisation

I listen to all questions