RWTH AACHEN UNIVERSITY

Advanced MPI

Hybrid programming, profiling and debugging of MPI applications

Hristo Iliev
RZ

Rechen- und Kommunikationszentrum (RZ)

Agenda

- Halos (ghost cells)
- Hybrid programming
- Profiling of MPI applications
- Vampir
- Debugging of MPI applications
- TotalView
- ---
- Lab session

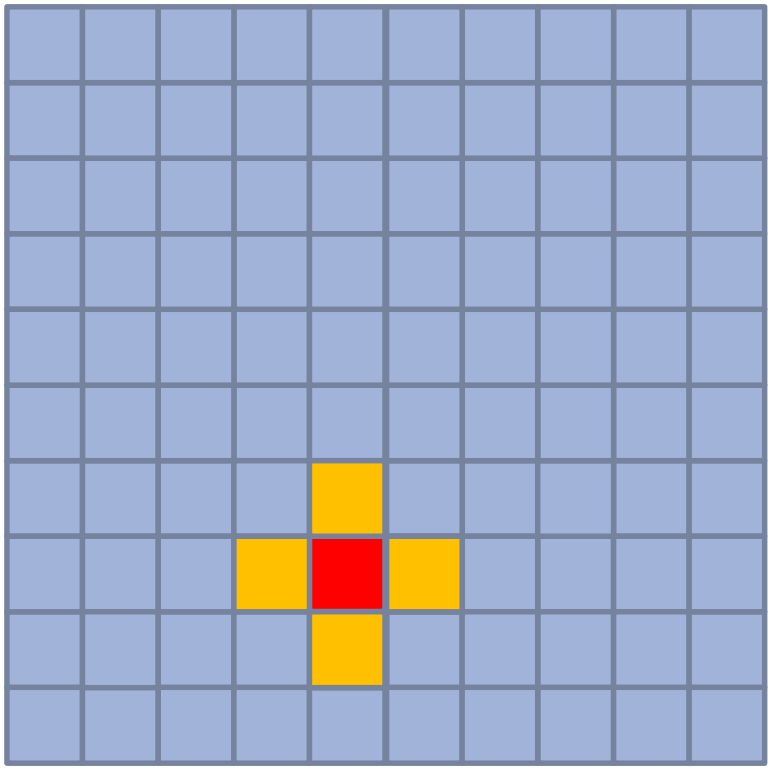
RWTH AACHEN UNIVERSITY

Halos (ghost cells)

Halos (ghost cells)

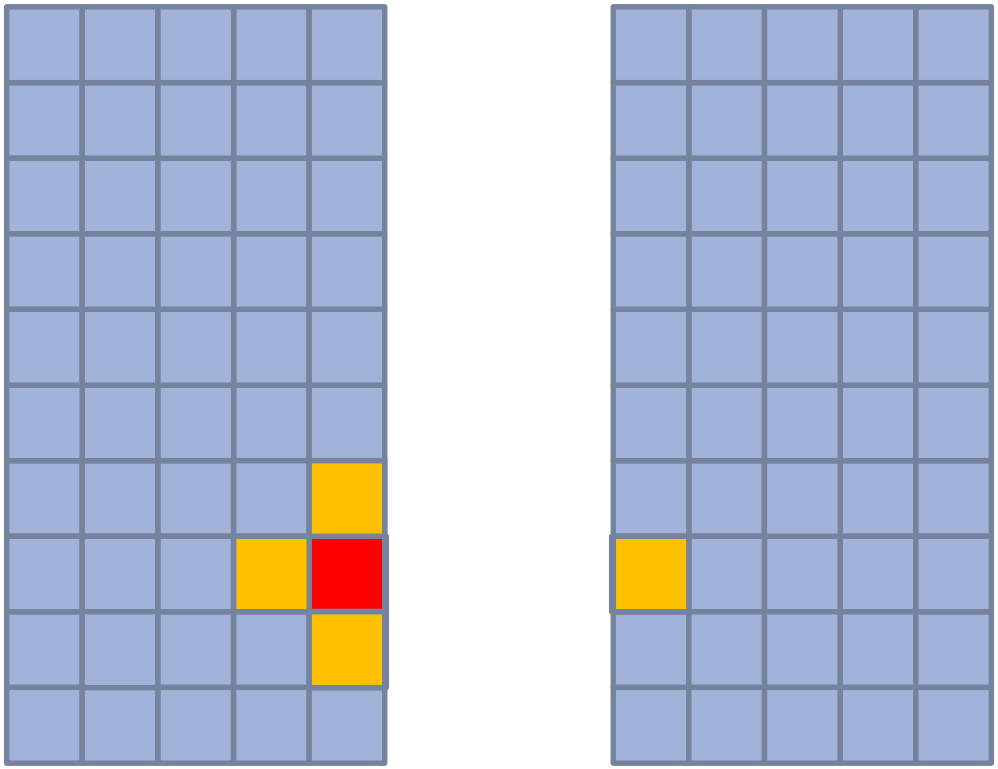
- **Often as a result of domain decomposition interdependent data end in different processes**

$$cell_{i,j} = f\left(cell_{i,j}; cell_{i-1,j}, cell_{i+1,j}, cell_{i,j-1}, cell_{i,j+1}\right)$$



Halos (ghost cells)

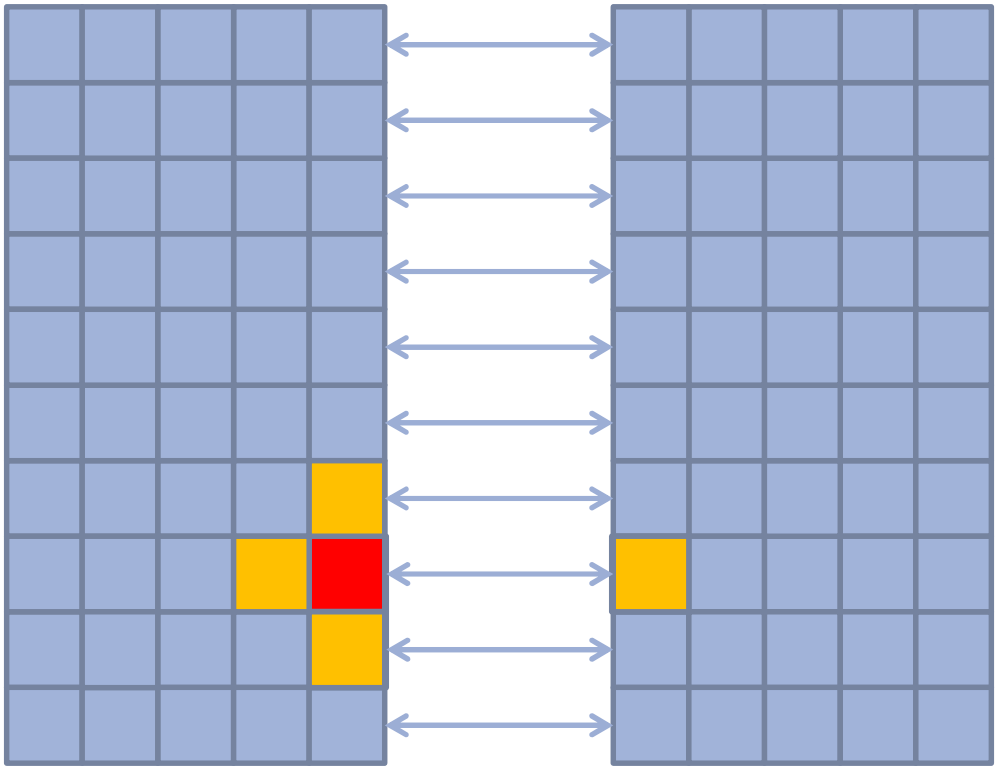
- Often as a result of domain decomposition interdependent data end in different processes



Halos (ghost cells)

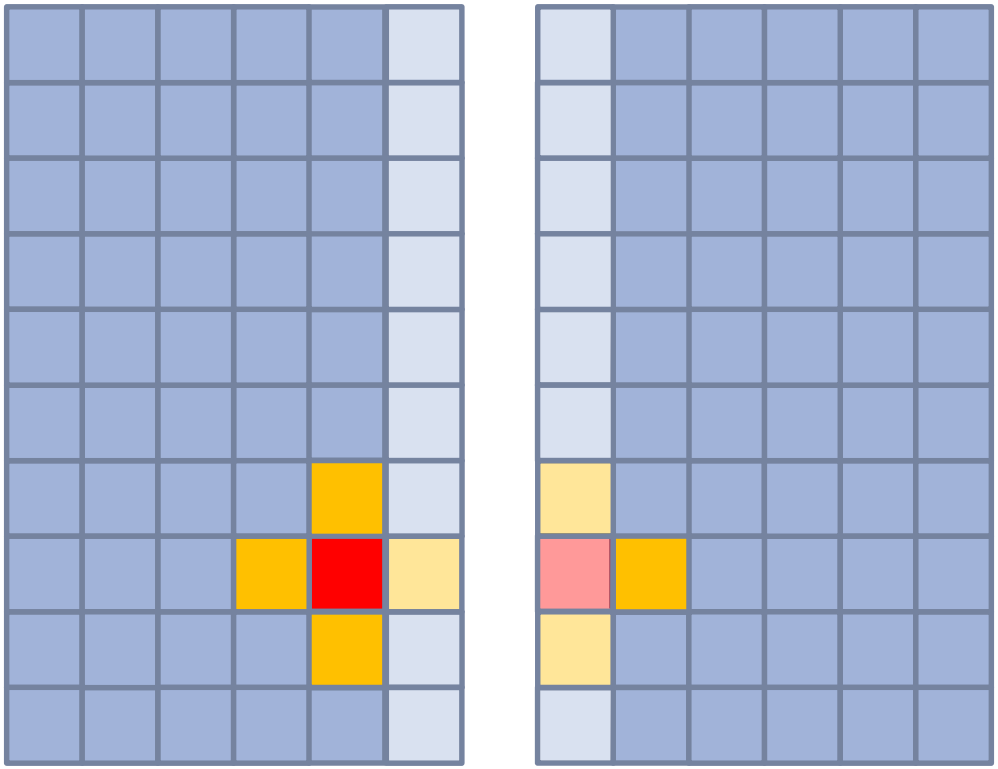
- **It makes no sense to request individually each neighbouring cell that one process needs to access**

  → Better to transfer all cells in bulk, but where to store them?



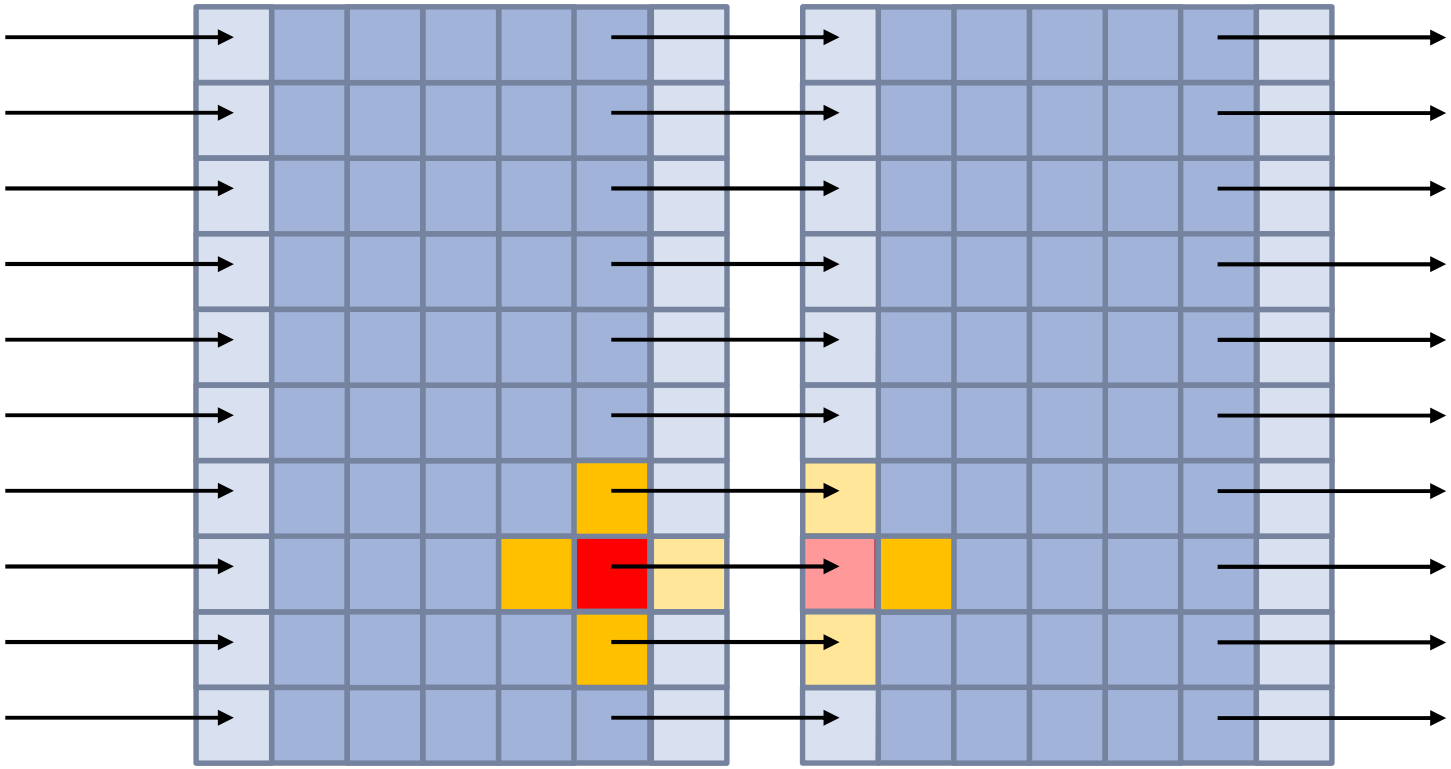
Halos (ghost cells)

- **Halos (ghost cells) are an additional layer of cells that duplicate the subset of neighbour cells that each iteration depends on**
  - → Not the real cells but temporary copies, hence “ghost” cells



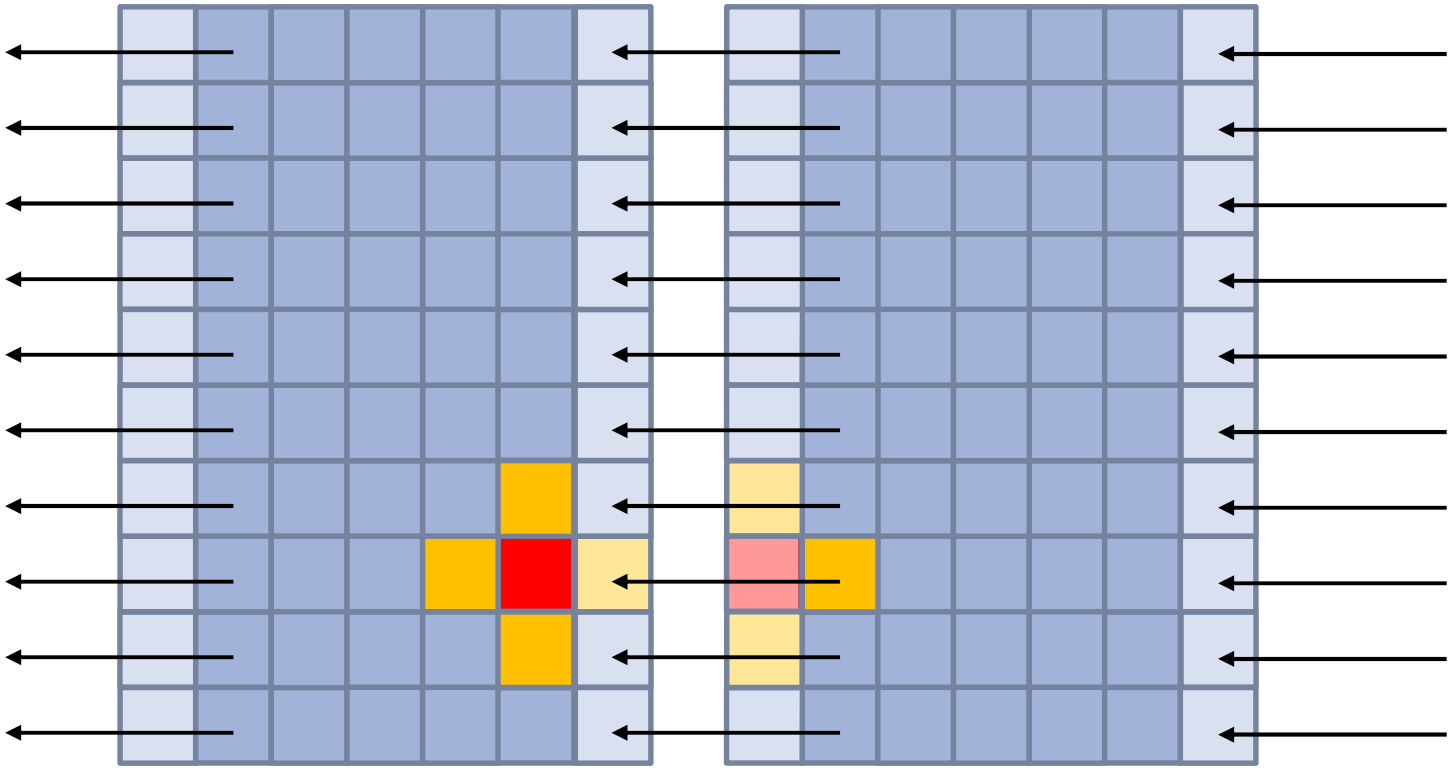
Halos (ghost cells)

- **Halos are updated regularly using halo swap operations**
  - → Conveniently performed using two `MPI_SENDRECV` per dimension



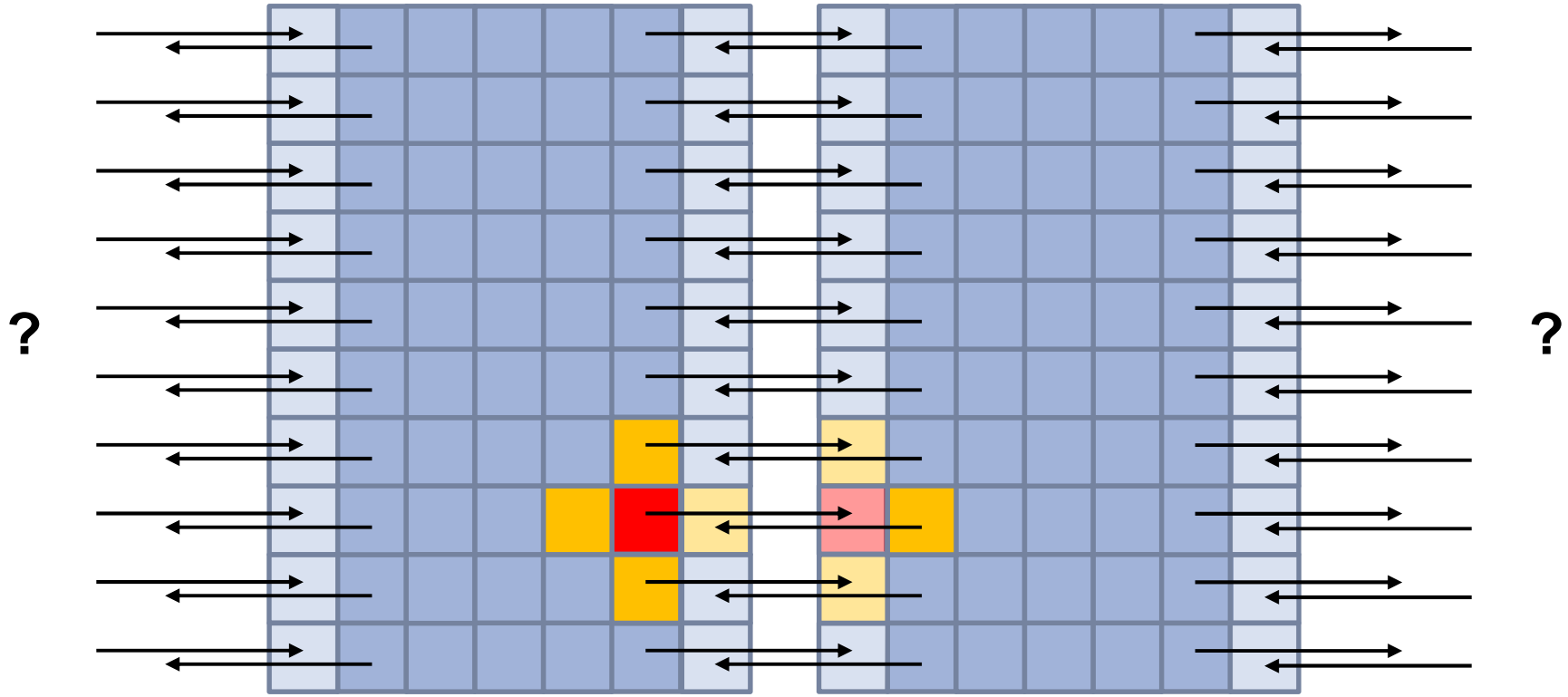
Halos (ghost cells)

- **Halos are updated regularly using halo swap operations**
  - → Conveniently performed using two `MPI_SENDRECV` per dimension



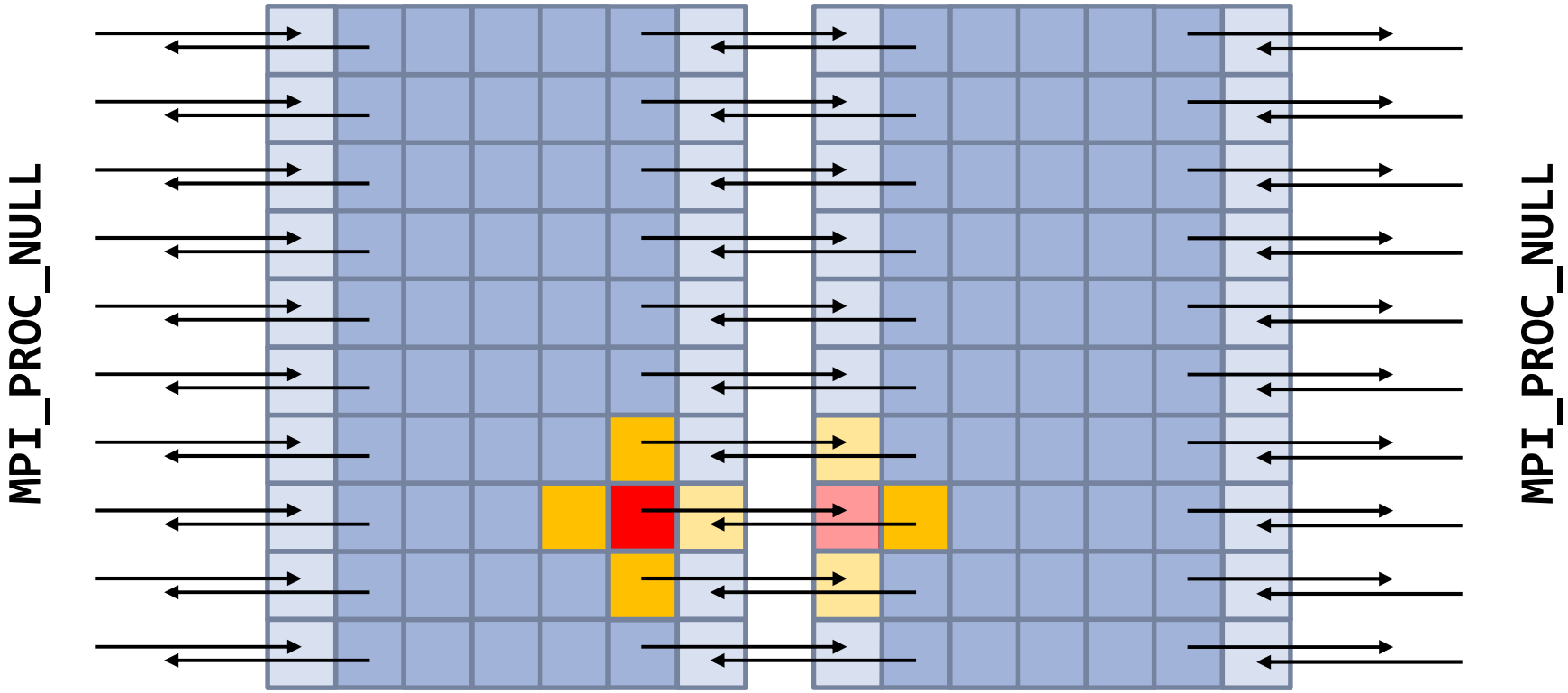
Halos (ghost cells)

- **Halos are updated regularly using halo swap operations**
  - → Conveniently performed using two `MPI_SENDRECV` per dimension



Halos (ghost cells)

- **Halos are updated regularly using halo swap operations**
  - → Conveniently performed using two `MPI_SENDRECV` per dimension



Halos (ghost cells)

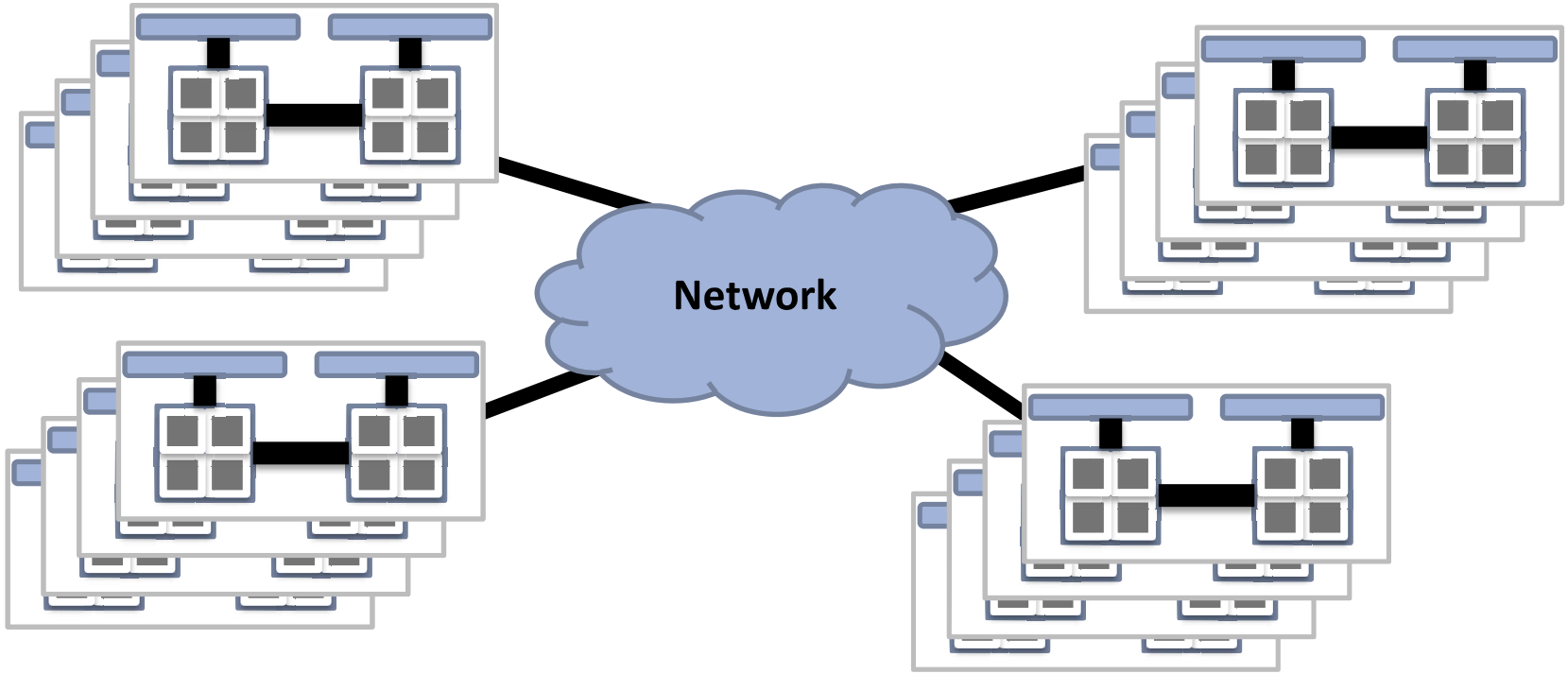
RWTH AACHEN UNIVERSITY

- **Could be performed in multiple dimensions**
  - → Independent swaps on each dimension
  - → Order is not important in most cases
  - → Diagonal elements in 2D-, 3D-, 4D-, etc. cases move automagically to their correct positions in the halos
- **If communication is expensive**
  - → Multilevel halos
  - → More data during halo swaps but less frequent communication
  - → Use halo data from level *n* to compute halo data from level *n*-1 and so on

Hybrid programming

Motivation for hybrid programming

- **Clusters of small supercomputers**
  - → Increasingly complex nodes – many cores, GPUs, Intel MIC, etc.
  - → Green computing will make things even more complex

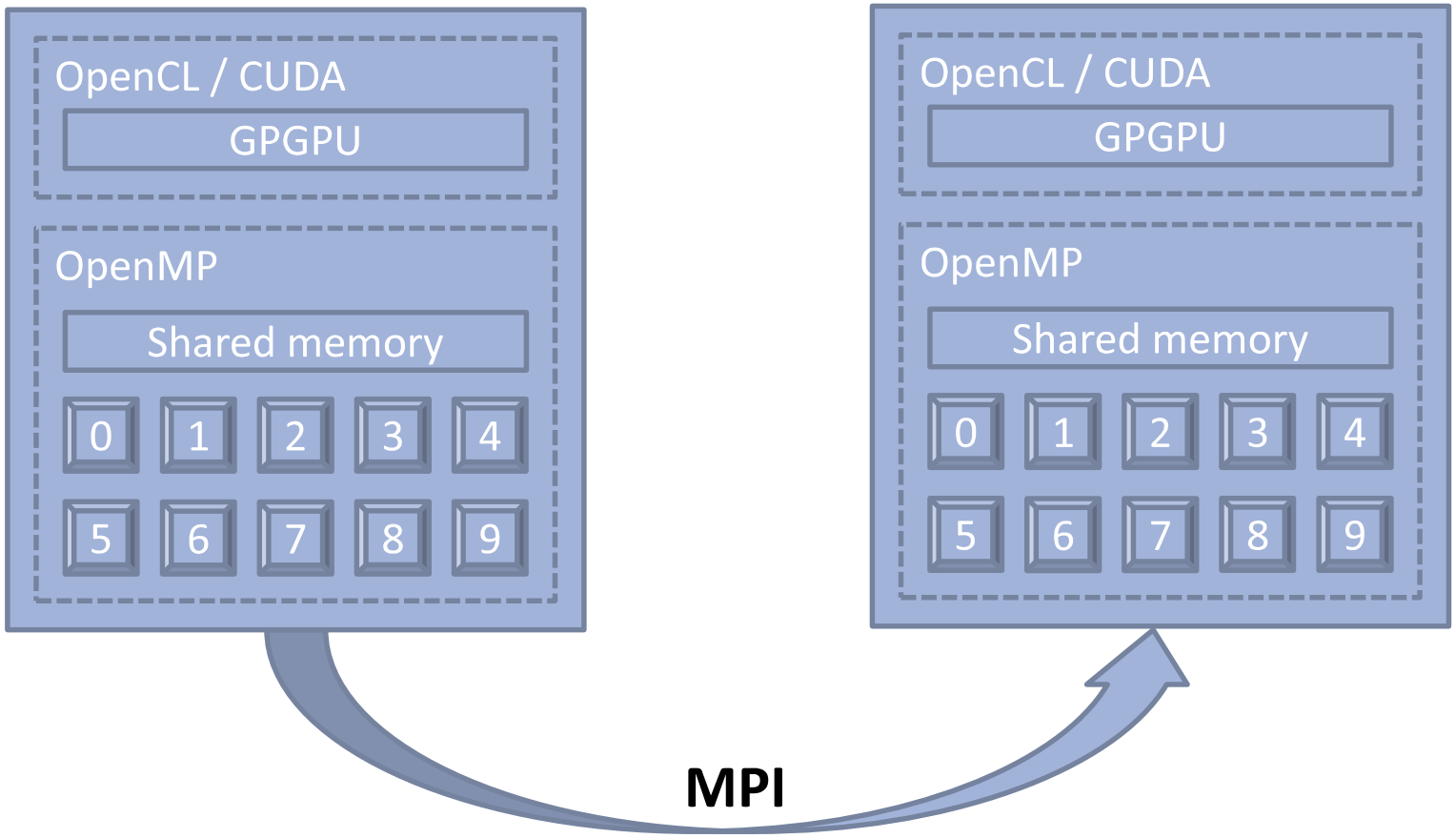


MPI on a single node

- **MPI is sufficiently abstract to run on a single node**
  - → It doesn't care where its processes are located
  - → Message passing implemented using shared memory and IPC
    - →The MPI library takes care
    - →Usually faster than sending messages over the network
  - → but…
- **… it is far from optimal**
  - → MPI processes are implemented as separate OS processes
  - → Portable data sharing is hard to achieve
  - → Lots of control / problem data has to be duplicated
  - → Reusing cached data is practically impossible

Hybrid programming

- **Hierarchical mixing of different programming paradigms**



Best of all worlds

RWTH AACHEN UNIVERSITY

- **OpenMP / pthreads**
  - → Cache reusage, data sharing
  - → Simple programming model (not so simple with POSIX threads)
  - → Threaded libraries (e.g. MKL)
- **OpenCL / CUDA**
  - → Massively parallel GPGPU accelerators
- **MPI**
  - → Fast transparent networking
  - → Scalability
- **Downsides**
  - → Only benefits truly multiscale hierarchical problems
  - → Code could be hard to maintain

MPI – threads interaction

- **Most MPI implementation are threaded (e.g. for non-blocking requests) but not thread-safe**

- **Four levels of threading support**

Level identifier	Description
`MPI_THREAD_SINGLE`	Only one thread may execute
`MPI_THREAD_FUNNELED`	Only the main thread may make MPI calls
`MPI_THREAD_SERIALIZED`	Only one thread may make MPI calls at a time
`MPI_THREAD_MULTIPLE`	Multiple threads may call MPI at once with no restrictions

- **All implementations support `MPI_THREAD_SINGLE`, but some does not support `MPI_THREAD_MULTIPLE`**

MPI and threads – initialisation

RWTH AACHEN UNIVERSITY

- **Initialise MPI with thread support**

```
C/C++:   int MPI_Init_thread (int *argc, char ***argv,
                              int required, int *provided)
Fortran: SUBROUTINE MPI_INIT_THREAD (required, provided,
                                     ierr)
```

→ **`required`** tells MPI what level of thread support is required

→ **`provided`** tells us what level of threading MPI actually supports

→ could be less than the required level

→ **`MPI_INIT`** – same as a call with **`required = MPI_THREAD_SINGLE`**

→ The thread that has called **`MPI_INIT_THREAD`** becomes the *main thread*

→ The level of thread support is fixed by this call and cannot be changed later

MPI and threads – initialisation

RWTH AACHEN UNIVERSITY

- **Obtain current level of thread support**

```
MPI_Init_thread (provided)
```

→ **provided** is filled with the current level of thread support

→ If **MPI_INIT_THREAD** was called, **provided** will equal the value returned by the MPI initialisation call

→ If **MPI_INIT** was called, **provided** will equal an implementation specific value

- **Find out if current thread is the main one**

```
MPI_Is_thread_main (flag)
```

MPI thread support @ BULL cluster

RWTH AACHEN UNIVERSITY

- **Thread support in various MPI libraries**

requested	Open MPI	Open MPI mt	Intel MPI w/o -mt_mpi	Intel MPI w/ -mt_mpi
SINGLE	SINGLE	SINGLE	SINGLE	SINGLE
FUNNELED	SINGLE	FUNNELED	SINGLE	FUNNELED
SERIALIZED	SINGLE	SERIALIZED	SINGLE	SERIALIZED
MULTIPLE	SINGLE	MULTIPLE	SINGLE	MULTIPLE

- **Open MPI**
  - → Use module versions ending with **`mt`** (e.g. **`openmpi/1.6mt`**)
  - → The InfiniBand BTL is not fully thread safe!
- **Intel MPI**
  - → Enabled when compiling with **`-openmp`**, **`-parallel`**, or **`-mt_mpi`**

Interoperation with CUDA (experimental!)

RWTH AACHEN UNIVERSITY

- **Modern MPI libraries provide some degree of CUDA interoperation**
  - → Device pointers as buffer arguments in many MPI calls
  - → Direct device to device RDMA transfers over InfiniBand
  - → Implemented by Open MPI and MVAPICH2

- **Experimental CUDA build of Open MPI 1.7 in the BETA modules category**

```
gpu-cluster$ module load BETA
gpu-cluster$ module switch openmpi openmpi/1.7cuda
```

  - → You would need first to obtain access to the GPU cluster

    (ask RZ ServiceDesk)

Running hybrid code with LSF

RWTH AACHEN UNIVERSITY

- **Sample hybrid job script (Open MPI)**

```
#!/usr/bin/env zsh

# 16 MPI procs x 6 threads = 96 cores
#BSUB -x
#BSUB -a openmpi
#BSUB -n 16
# 12 cores/node / 6 threads = 2 processes per node
#BSUB -R “span[ptile=2]”

# 6 threads per process
export OMP_NUM_THREADS=6
# Pass OMP_NUM_THREADS on to all MPI processes
$MPIEXEC $FLAGS_MPI_BATCH -x OMP_NUM_THREADS \
         program.exe <args>
```

- **For the most up to date information refer to the HPC Primer**

Running hybrid code with LSF

RWTH AACHEN UNIVERSITY

- **Sample hybrid job script (Intel MPI)**

```
#!/usr/bin/env zsh

# 16 MPI procs x 6 threads = 96 cores
#BSUB -x
#BSUB -a intelmpi
#BSUB -n 16
# 12 cores/node / 6 threads = 2 processes per node
#BSUB -R “span[ptile=2]”

module switch openmpi intelmpi
# 6 threads per process
# Pass OMP_NUM_THREADS on to all MPI processes
$MPIEXEC $FLAGS_MPI_BATCH -genv OMP_NUM_THREADS 6 \
         program.exe <args>
```

- **For the most up to date information refer to the HPC Primer**

RWTH AACHEN UNIVERSITY

Profiling of MPI applications

VampirTrace and Vampir

Profiling of parallel applications

RWTH AACHEN UNIVERSITY

- **Profiling – collecting performance data about an application**
  - → Number of calls
  - → Time spent in specific calls
  - → MPI messages – volume, speed, latency, etc.
- **Challenges**
  - → Many interdependent processes running on many nodes
  - → Measurement overhead
  - → Accurate vs. realistic profile
  - → Scalability of performance data collection
  - → Analysis of performance data

The PMPI interface

- **MPI defines a very lightweight "black box" tools interface – PMPI**
  - → `MPI_Some_call` is not a real function but a weak symbol synonym to…
  - → The true MPI call is `PMPI_Some_call`
  - → Weak symbols can be overridden at link time

```
int MPI_Some_call (int arg1, MPI_Comm arg2)
{
   measure_enter("MPI_Some_call", arg1, arg2);
   int ierr = PMPI_Some_call(arg1, arg2);
   measure_exit();
   return ierr;
}
```

- **Can be abused e.g. for testing different collective algorithms**
  - → Link with the rest of the code and all `MPI_Some_call()` calls get redirected

Tracing and instrumentation

RWTH AACHEN UNIVERSITY

- **Instrumentation**
  - → Insertion of data and event collection points into the program code
  - → Automatic (by the compiler) or by hand
- **Trace – collection of timestamped events, gathered through the means of instrumentation**
  - → enter into and exit from subroutines
  - → IO and network events
  - → performance counters
  - → user-defined events
- **Traces are collected separately for each rank (and then merged)**

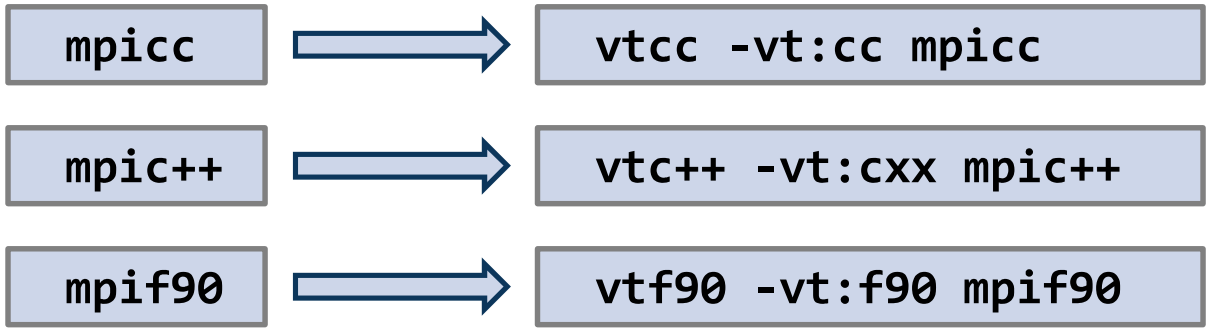
VampirTrace

- **Open-source tracing library, part of the Vampir suite, developed at TU Dresden**
- **Supports many different events**
  - → Full call tracing (very data-intensive, doesn't scale)
  - → MPI events
  - → IO events
  - → Hardware performance counters (via PAPI)
- **Collected traces stored in Open Trace Format (OTF)**
  - → master file `<name>.otf`
  - → global definition file `<name>.0.def`
  - → events for each MPI rank `<name>.<1-#procs>.events`
  - → ZLIB compression

Using VampirTrace – compiling

RWTH AACHEN UNIVERSITY

- **More compiler wrappers**



- **Instrumentation type selected via `-vt:inst <type>`**
  - → `compinst` compiler assisted instrumentation (default)
    all function calls traced; very detailed; huge trace files
  - → `manual` manual tracing using VampirTrace API
    traces only MPI events and user-specified events;
    significantly reduced trace file size

Using VampirTrace – running

RWTH AACHEN UNIVERSITY

- **VampirTrace is controlled by many environment variables**
  - → `VT_BUFFER_SIZE` internal trace buffer size; flushed to the disk when full (default: 32M)
  - → `VT_FILE_PREFIX` OTF file prefix (default: executable name)
  - → `VT_MAX_FLUSHES` number of trace buffer flushes before tracing is disabled (0 – no limit)
  - → `VT_SYNC_FLUSH` synchronised buffer flushes (default: no)

- **Things to be aware of:**
  - → By default buffers are flushed asynchronously and it takes time
  - → Significant skew in program's performance profile possible
  - → No trace written after abnormal program termination

Using VampirTrace – running

RWTH AACHEN UNIVERSITY

- **Open MPI**

```
module load UNITE vampirtrace

export VT_BUFFER_SIZE=128M

$MPIEXEC -x VT_BUFFER_SIZE -x LD_LIBRARY_PATH \
         <program> <args>
```

- **Intel MPI**

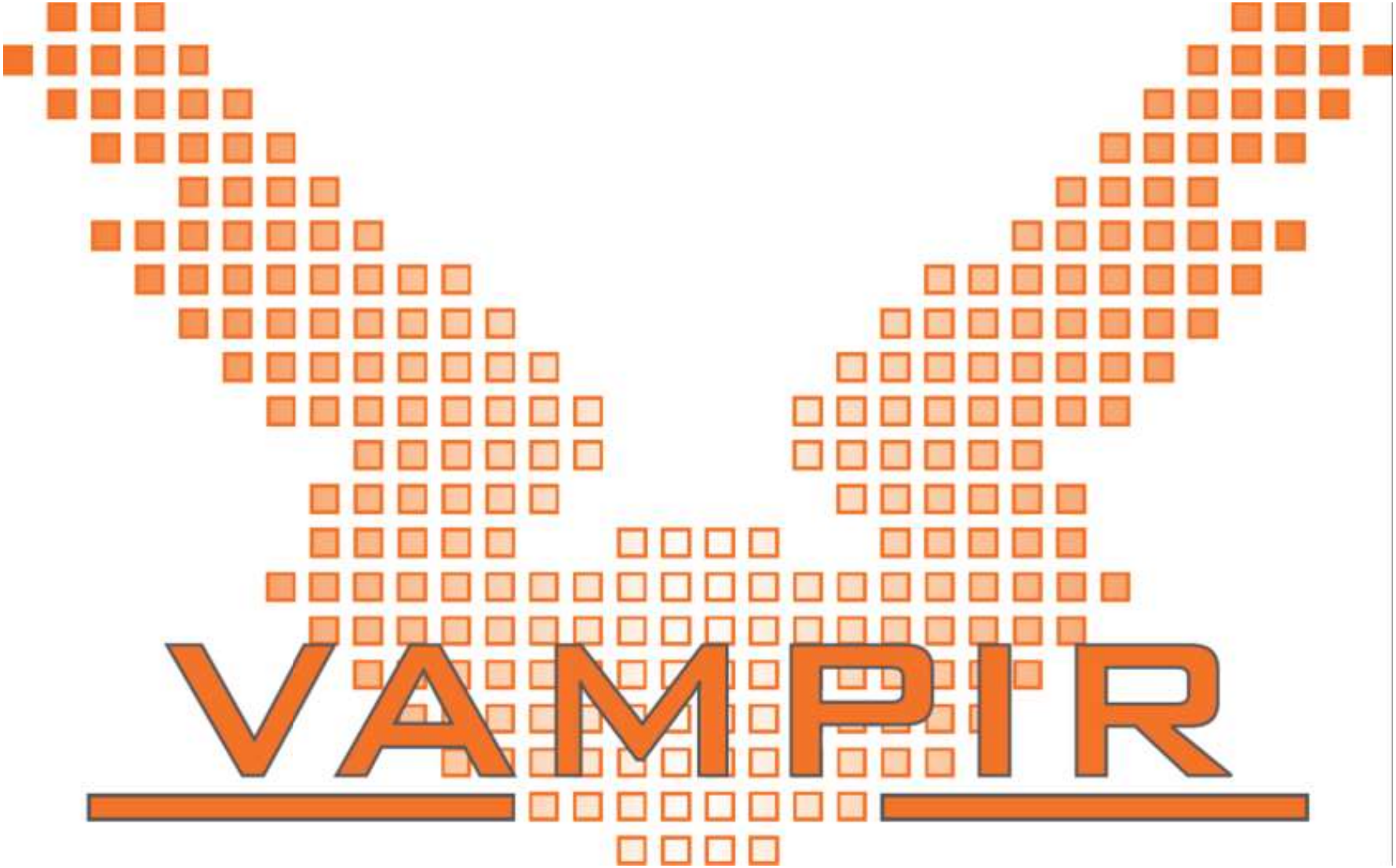
```
module load UNITE vampirtrace

export VT_BUFFER_SIZE=128M

$MPIEXEC -genvlist VT_BUFFER_SIZE,LD_LIBRARY_PATH \
         <program> <args>
```

Vampir

RWTH AACHEN UNIVERSITY

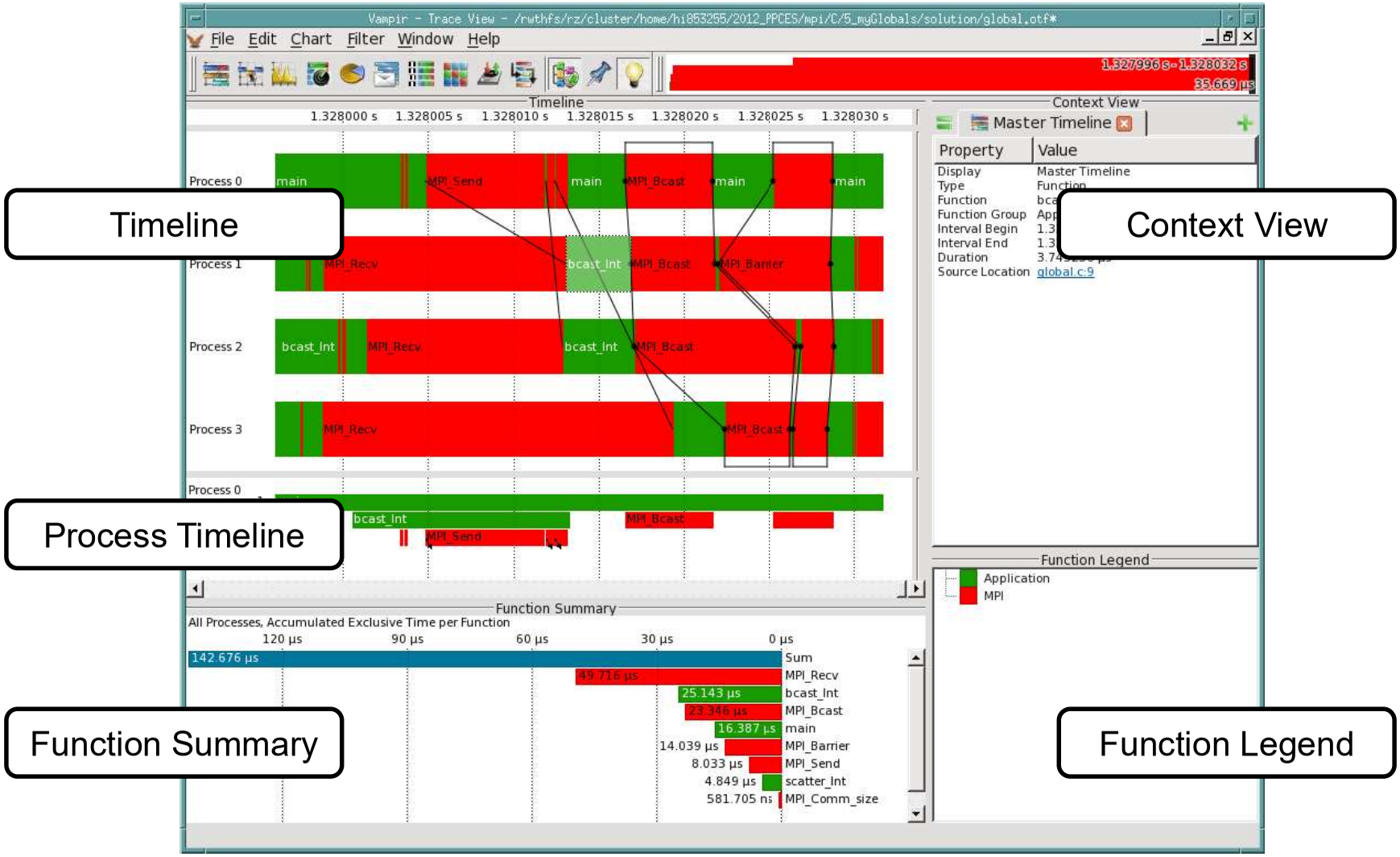


Vampir

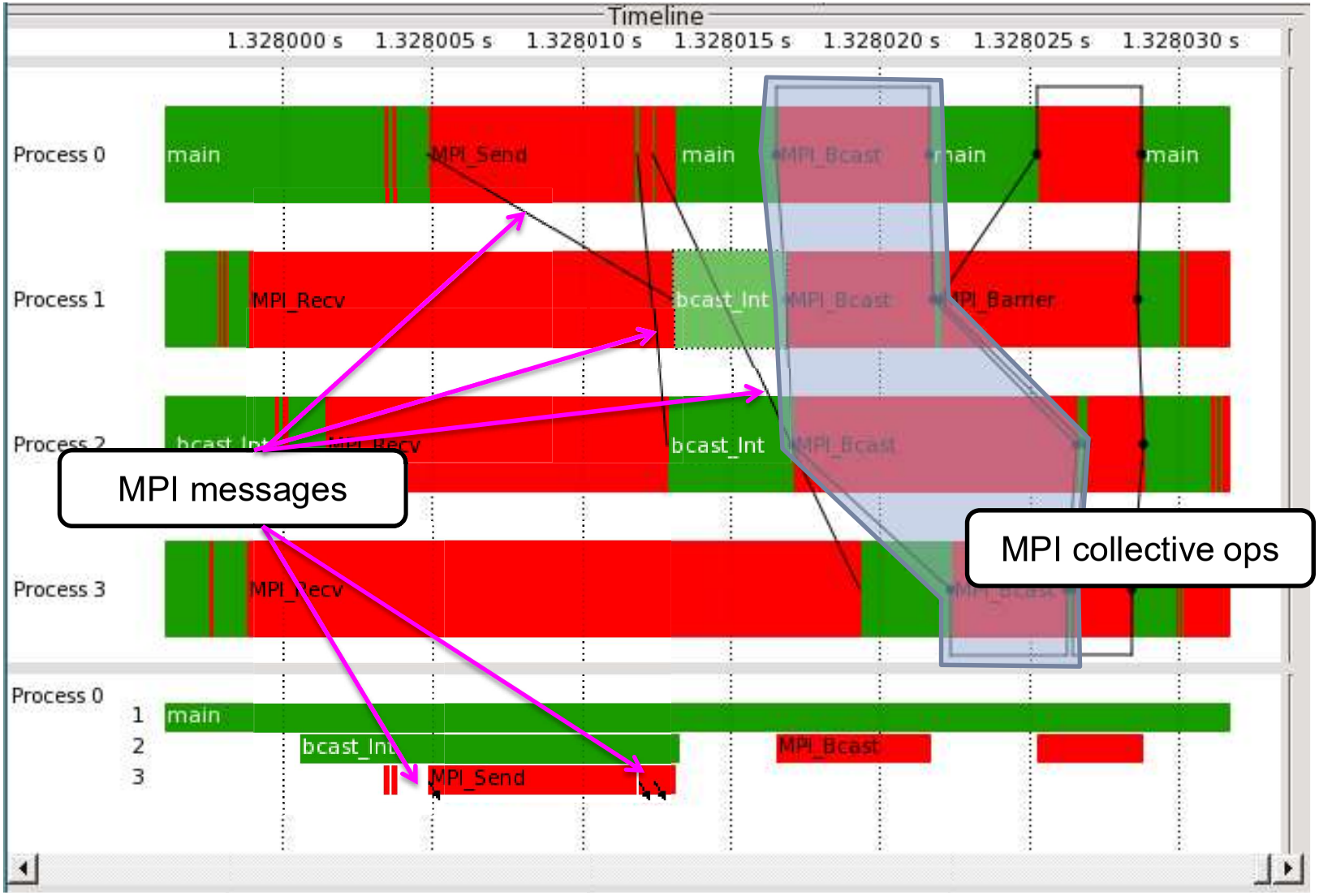
RWTH AACHEN UNIVERSITY

- **Commercial performance analysis tool, developed at TU Dresden**
- **Reads traces in OTF format, written by VampirTrace or any other tracing framework that supports OTF**
  - → MPI events
  - → IO events
  - → Hardware performance counters
- **Could be used simply as a timeline browser**
- **Vampir Server**
  - → Distributed trace analysis
  - → Better scalability – can handle much larger traces

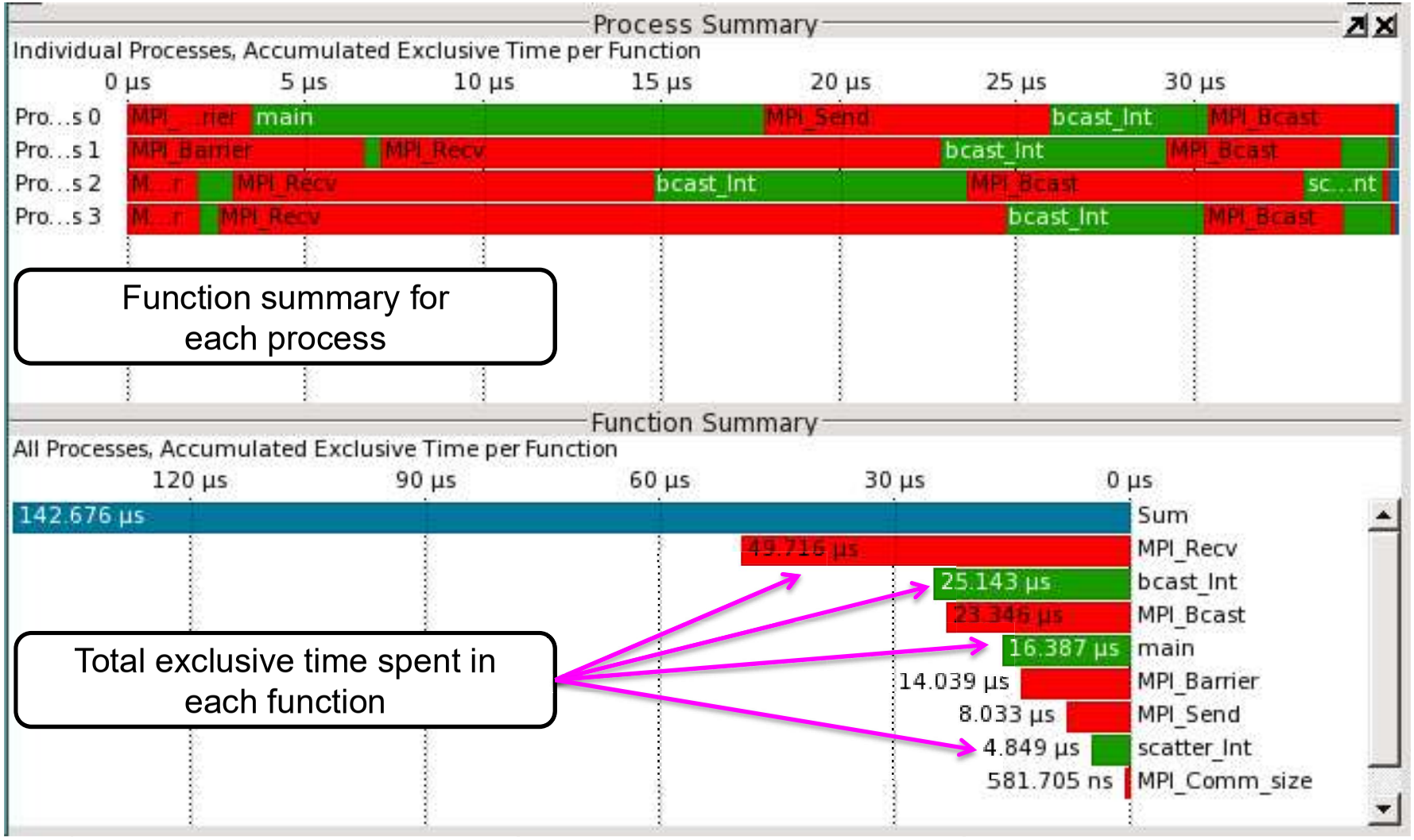
Vampir – trace view



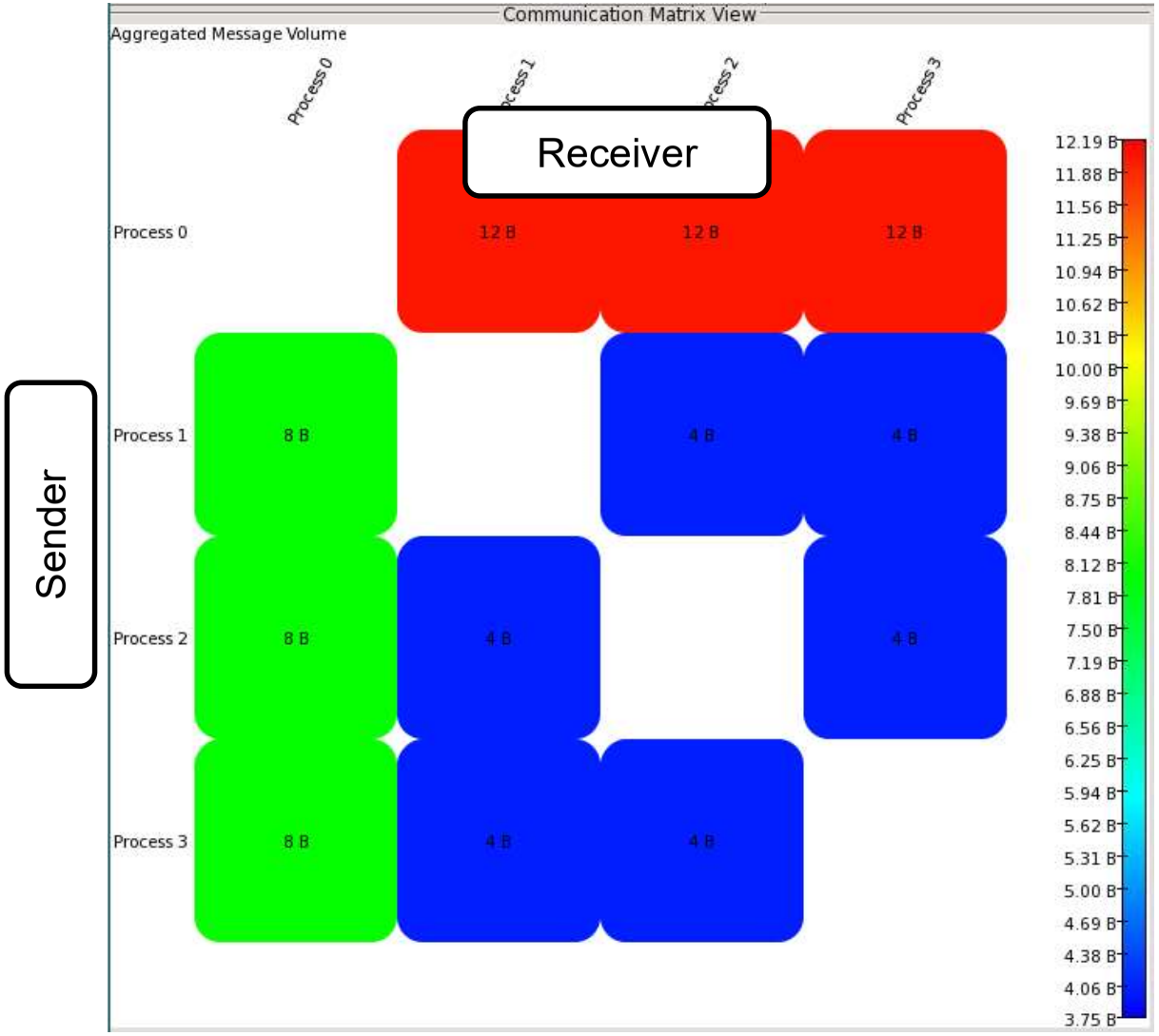
Vampir – timeline and process timeline



Vampir - summaries



Vampir – communication matrix



RWTH AACHEN UNIVERSITY

Vampir demonstration

RWTH AACHEN UNIVERSITY

Debugging of MPI applications

TotalView

Parallel debugging

- **Parallel applications are hard to debug**
  - → Multiple processes running concurrently on different nodes
  - → Communication schemes could be very sophisticated
  - → `print(f)` debugging doesn't scale
- **Parallel debuggers**
  - → Client/server model
  - → Can debug multiple processes over the network
  - → Built-in mechanisms to deal with multiple code/data instances
- **TotalView**
- **Allinea DDT**

TotalView

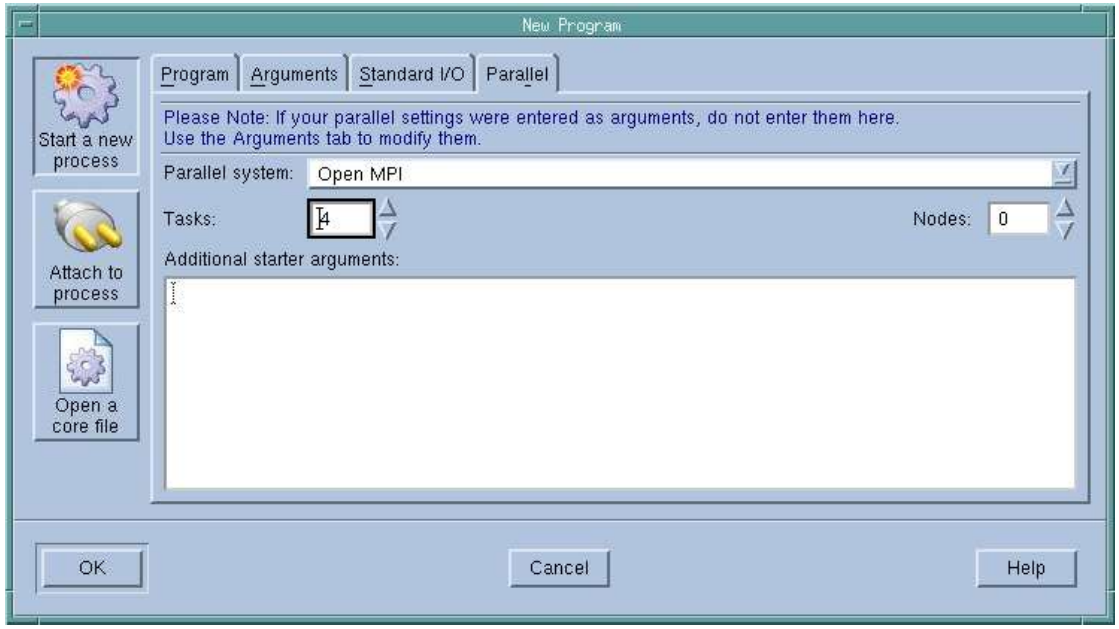
- **Versatile mixed-mode debugger**
  - → MPI
  - → OpenMP / general threading libraries
  - → CUDA
- **Commercial, developed by Rogue Wave Software**
- **All MPI libraries on the RWTH BULL cluster have built-in support to launch programs under TotalView**
  - → Open MPI
  - → Intel MPI
  - → TotalView can break automatically inside the `MPI_INIT` call

Using TotalView

- TotalView is in the `UNITE` module category
- Launch via the MPI program launcher

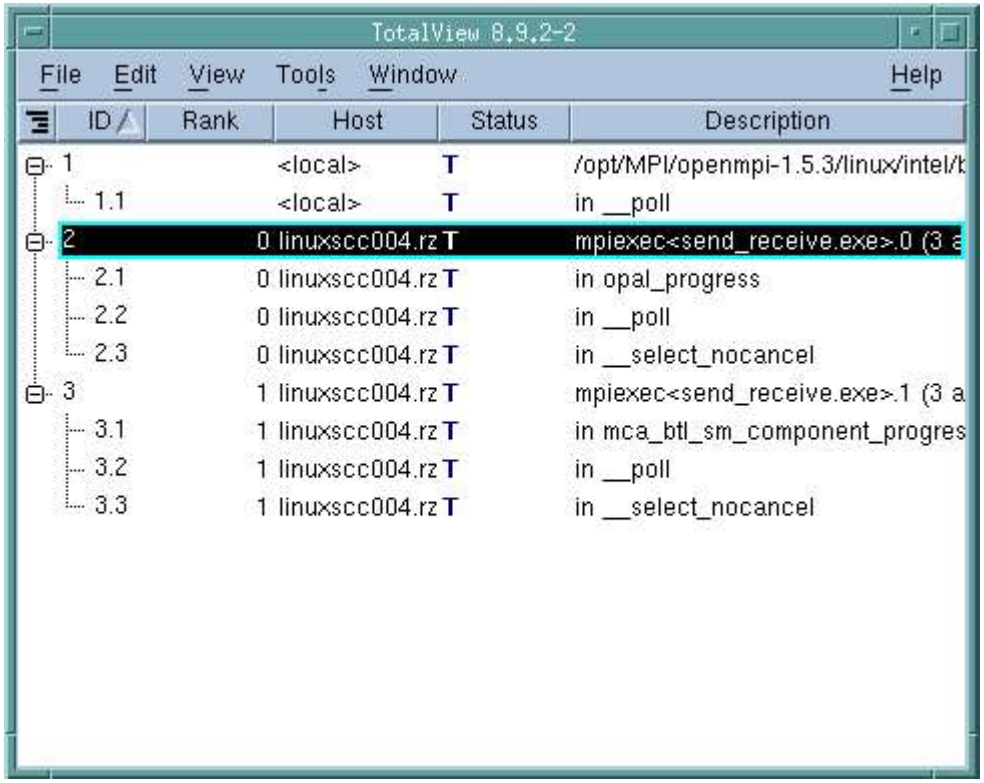
```
cluster$ module load UNITE totalview
cluster$ $MPIEXEC -np 4 -tv <program> <args>
```

- Use the built-in parallel launcher



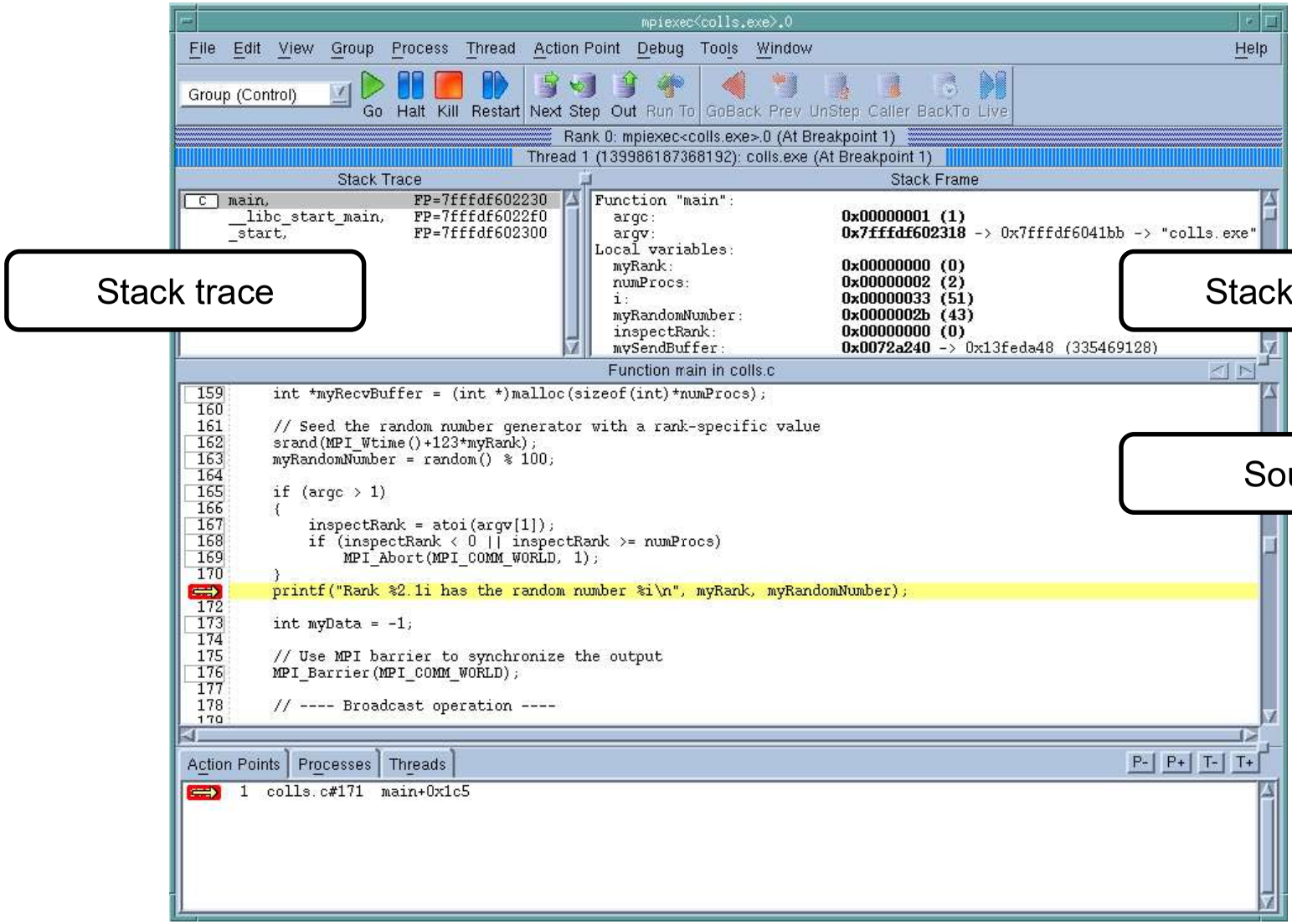
TotalView – Root Window

- **Summary of processes, ranks and top of thread stacks**



- **Also used to quickly switch focus from one process to another**

TotalView – Process Window



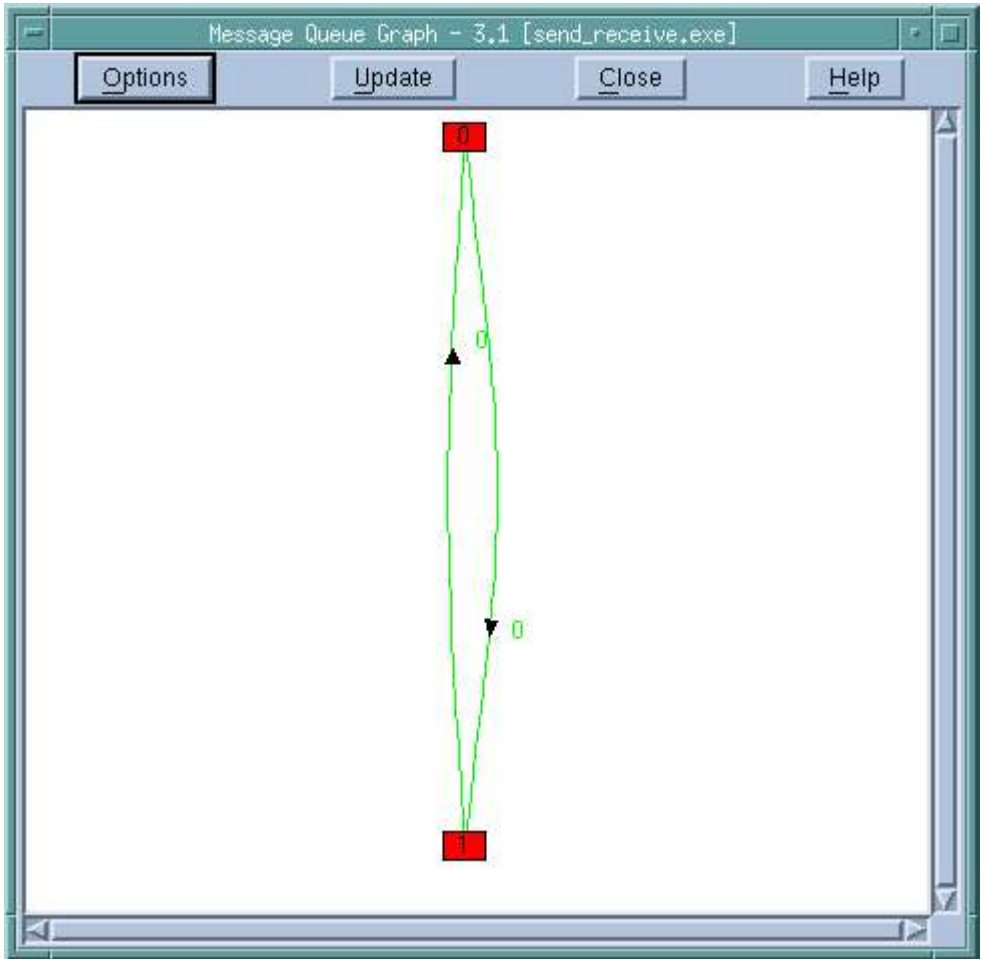
Stack trace

Stack frame

Source

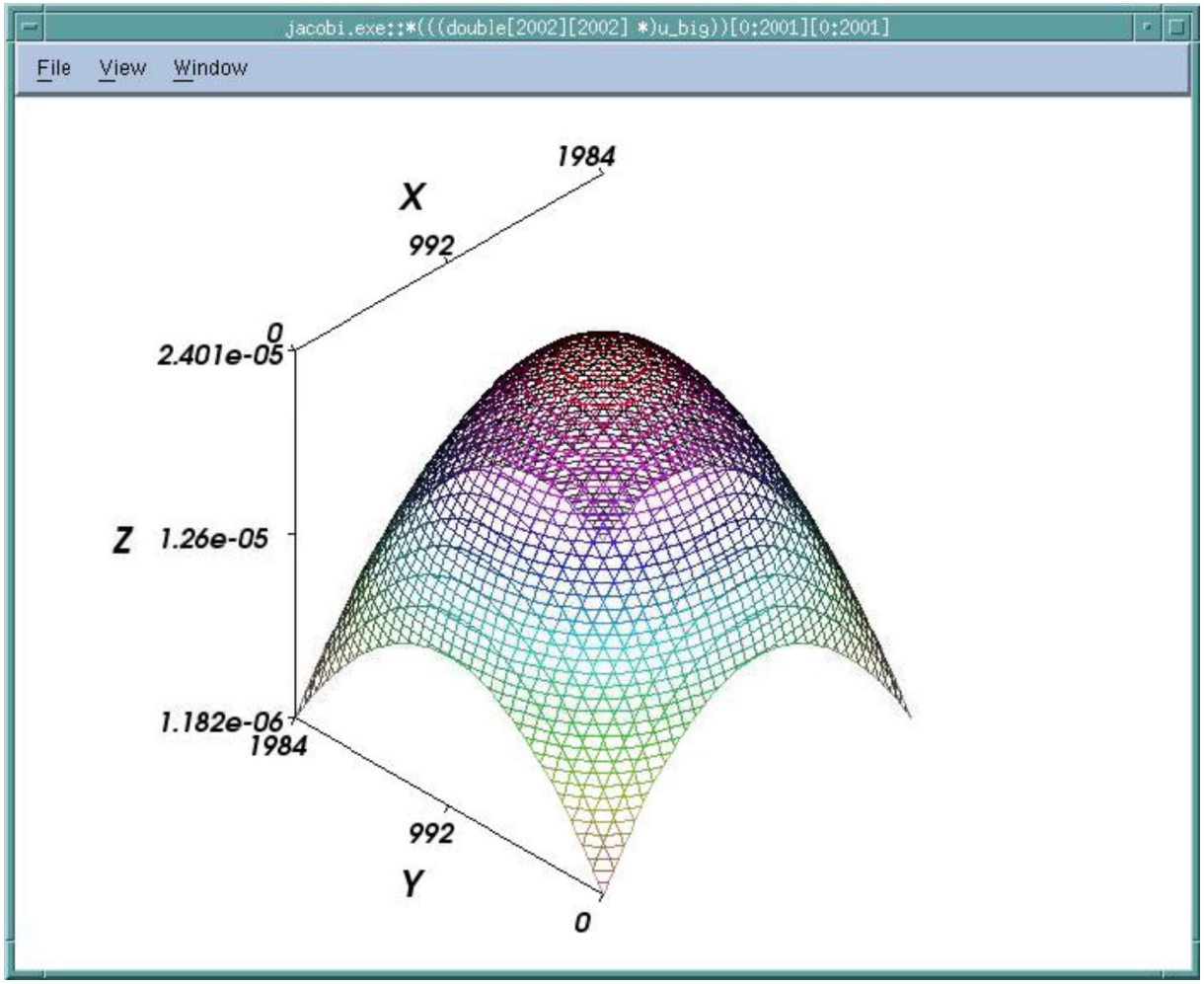
TotalView – Message Queue Graph

- **Cycle (deadlock) detected**



TotalView – Data visualizer

- **Built-in multidimensional data visualizer**



TotalView demonstration

More on the tools

- **More in-depth introduction to TotalView and other correctness tools:**
  - → 05.09.2012 – Correctness Tools tutorial
- **More in-depth introduction to Vampir and other performance tools:**
  - → 12.09.2012 – Performance Tools tutorial

RWTH AACHEN UNIVERSITY

Q & A session

Lab session of your choice

RWTH AACHEN UNIVERSITY

- **Continue working on exercises from the basic MPI tutorial**
  - → Hand-outs are available over there to your right
- **Parallelise from scratch a PDE solver**
  - → Hand-outs are available on your desk
- **Practice tracing with VampireTrace and Vampire**
  - → `make vampir`
- **Practice TotalView debugging**
  - → `make totalview`