Introduction to High Performance Computing

MPI (1)

Motivation

OpenMP is only available on systems with shared memory.
The parallelism is limited to the number of cores in ONE system.

Parallelism among different nodes requires explicit communication between tasks.
Messages have to be exchanged.

Message Passing Interface

- standardized parallelization library since 1994
- very performant (almost Hardware latency/bandwidth)
- many different implementations (different focus, but conforming to standard)
- still under development to incorporate multicore architecture (hybrid with OpenMP)
- development needed to incorporate manycore architectures

MPI-versions

MPI-versions

Intel-MPI

module add intel/latest

mpicc for compilation (calls icc (option –cc=icc), binds appropriate libraries, header, …)

```
bsub mpiexec.hydra ./exe
```

to execute in Batch-mode – additional options automatically added by LSF

more Information: Software page on Elwetritsch

MPI-versions

Platform-MPI

module add platform-mpi/latest

mpicc for compilation (calls gcc, binds appropriate libraries, header, …)

`bsub -a platformmpi mpirun -lsb_mcpu_hosts ./exe`

to execute – additional options automatically added by LSF

more Information: Software page on Elwetritsch

MPI-versions

OpenMPI (http://www.open-mpi.org)

module add intel-2015
module add openmpi/1.8.6-intel-2015

mpicc for compilation (calls icc (option –cc=icc), binds appropriate libraries, header, …)

`bsub –a openmpi mpirun ./exe`
to execute – additional options automatically added by LSF

more Information: Software page on Elwetritsch

Interfaces

```
Simple script to compile and/or link MPI programs.
Usage: mpigcc [options] <files>
----------------------------------------------------------------------------
The following options are supported:
   -cc=<name>       specify a C compiler name: i.e. -cc=gcc
   -echo            print the scripts during their execution
   -show            show command lines without real calling
   -config=<name>   specify a configuration file: i.e. -config=gcc for mpicc-gcc.conf file
   -v               print version info of mpigcc and its native compiler
   -profile=<name>  specify a profile configuration file (an MPI profiling
                    library): i.e. -profile=myprofile for the myprofile.cfg file.
                    As a special case, lib<name>.so or lib<name>.a may be used
                    if the library is found
   -check_mpi       link against the Intel(R) Trace Collector (-profile=vtmc).
   -static_mpi      link the Intel(R) MPI Library statically
   -mt_mpi          link the thread safe version of the Intel(R) MPI Library
   -ilp64           link the ILP64 support of the Intel(R) MPI Library
   -t or -trace
                    link against the Intel(R) Trace Collector
   -dynamic_log     link against the Intel(R) Trace Collector dynamically
   -static          use static linkage method
   -nostrip         turn off the debug information stripping during static linking
   -O               enable optimization
   -link_mpi=<name>
                    link against the specified version of the Intel(R) MPI Library
All other options will be passed to the compiler without changing.
----------------------------------------------------------------------------
The following environment variables are used:
   I_MPI_ROOT       the Intel(R) MPI Library installation directory path
   I_MPI_CC or MPICH_CC
                    the path/name of the underlying compiler to be used
   I_MPI_CC_PROFILE or MPICC_PROFILE
                    the name of profile file (without extension)
   I_MPI_COMPILER_CONFIG_DIR
                    the folder which contains configuration files *.conf
   I_MPI_TRACE_PROFILE
                    specify a default profile for the -trace option
   I_MPI_CHECK_PROFILE
                    specify a default profile for the -check_mpi option
   I_MPI_CHECK_COMPILER
                    enable compiler setup checks
   I_MPI_LINK       specify the version of the Intel(R) MPI Library
   I_MPI_DEBUG_INFO_STRIP
                    turn on/off the debug information stripping during static linking
----------------------------------------------------------------------------
```

Interfaces

```
schuele@head3 [~] mpirun -help

Usage: ./mpiexec [global opts] [exec1 local opts] : [exec2 local opts] : ...

Global options (passed to all executables):

  Global environment options:

  Other global options:
    -f {name} | -hostfile {name}     file containing the host names
    -hosts {host list}               comma separated host list
    -configfile {name}               config file containing MPMD launch options
    -machine {name} | -machinefile {name}
                                     file mapping procs to machines

Local options (passed to individual executables):

  Local environment options:

  Other local options:

Hydra specific options (treated as global):

  Bootstrap options:

  Resource management kernel options:

  Processor topology options:
    -binding                         process-to-core binding mode

  Checkpoint/Restart options:

  Demux engine options:

  Debugger support options:
    -tv                              run processes under TotalView

  Other Hydra options:
    -perhost <n>                     place consecutive <n> processes on each host
    -ppn <n>                         stand for "process per node"; an alias to -perhost <n>
```

green: usually not used

automatically set by batch system

Example

```
#include "mpi.h"
/* Simple Example for two Tasks */

main(int argc,char **argv)
{
   char text[20];
   int  myrank, size, sender=0, adressat=1, tag=99;
   MPI_Status status;

   MPI_Init(&argc,&argv);              /* Initialization */
    /* get unique ID*/
   MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
   /* get number of tasks involved */
   MPI_Comm_size(MPI_COMM_WORLD, &size);
```

Example

```
if(size != 2) {
      printf("Example for just 2 Tasks\n");
      MPI_Finalize();      exit(1);   }
if(myrank == 0) { /* different tasks do different things*/
    strcpy(text,"Hallo there");
     MPI_Send(text,strlen(text),MPI_CHAR,adressat,
                tag,MPI_COMM_WORLD);               /* Send */

} else {
   MPI_Recv(text,20,MPI_CHAR,sender,
               tag,MPI_COMM_WORLD,&status);/* Receive */
    printf("Task %d received:%s:\n",myrank,text);
}

  MPI_Finalize();                                /* End*/
}
```

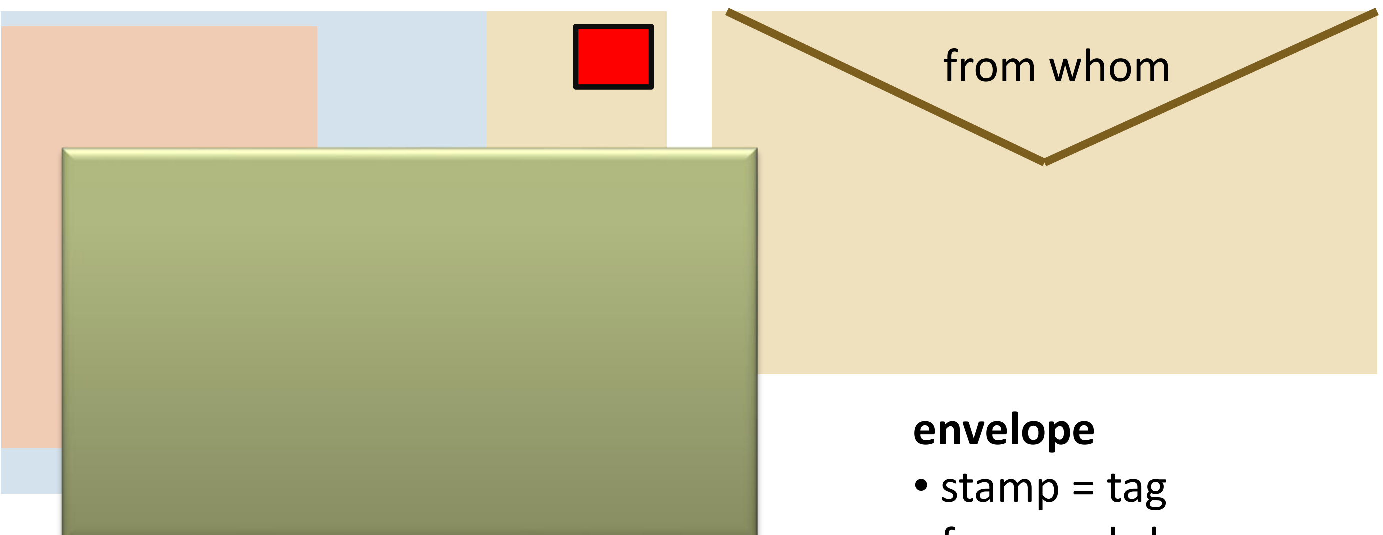
Example

- All functions use MPI_X (capital), rest small
- Tasks are available after MPI_Init
- Communication requires pair-wise action – one sender, one receiver
- No syntax control
- All data has to be exchanged – explicitely
- SPMD (single program multiple data) style
- Different work to different tasks organized by their unique ID
- No automatic work sharing
- Tasks work completely independent of each other
- Explicit synchronization required

Point-to-Point

```
MPI_Send(&what, length, type, adressat, tag,comm);
MPI_Recv(&what,length, type, sender, tag, comm, &status);
```

2 types of information are required to send/receive data:
- data specific (memory location, length, type)
- package information (adress, identification possibilities)



envelope
- stamp = tag
- form and shape = communicator

Point-to-Point

data specific information:

- Adress in memory (&what)
- Length (counted in words – not bytes)
- Type
  - pre-defined like MPI_Int, MPI_Double
  - user-defined (not covered in this lecture)
- neither length nor type do have to match
- receive buffer has to be large enough to keep all data sent

send 2 doubles (2x64)
receive 4 float (4x32)

ok

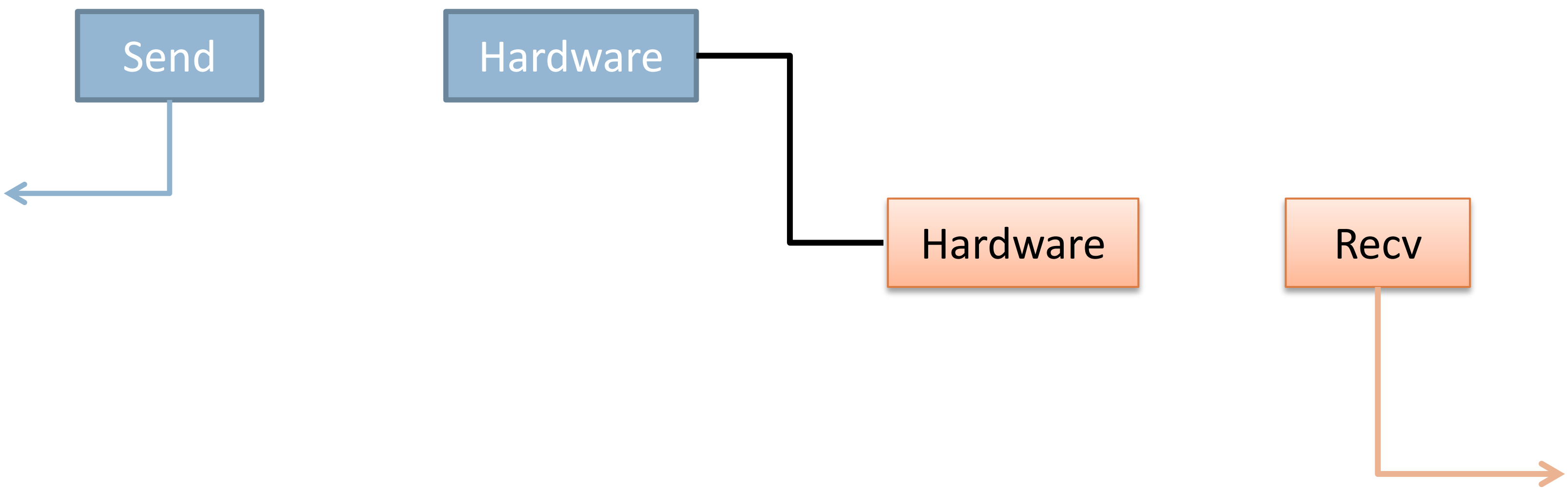
send … 2x64)
receive 2 integers (2x32)

not ok

Send / Recv

What actually happens at MPI_Send nad MPI_Recv?

Players:



blocking communication

Actual implementation of MPI_Send is undefined, except that it is **blocking**.

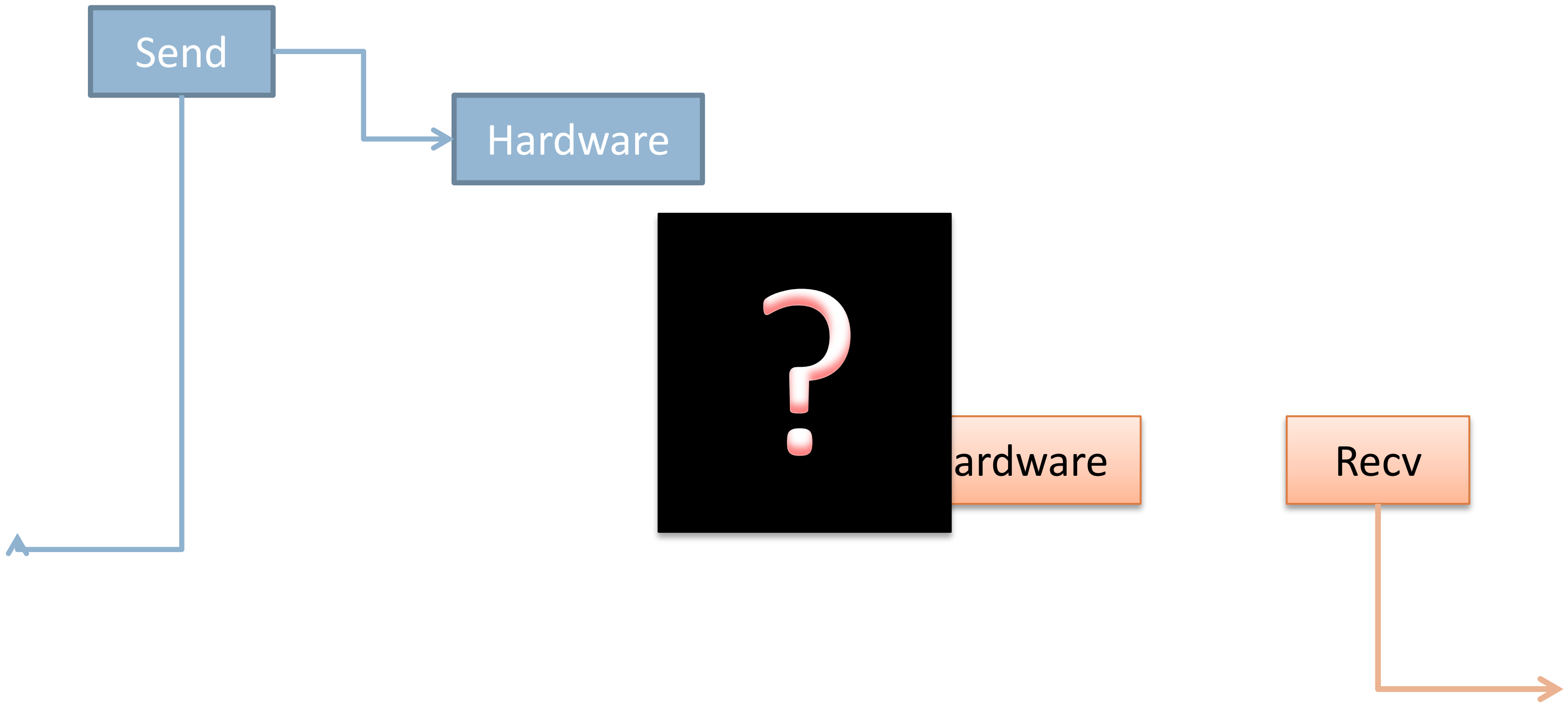
Blocking communication:
memory location employed in communication is after return of call **(re)usable**

```
x=1.;
MPI_Send (&x,1,MPI_DOUBLE,....);
x=4.;
MPI_Receive(&y,1,MPI_DOUBLE, ....);
z=y;
```

will give correct results. x=1 is sent and x=y.

Send / Recv

What actually happens at MPI_Send nad MPI_Recv?



non-blocking communication

```
x=1.;
MPI_Isend (&x,1,MPI_DOUBLE,….);
x=4.;
```

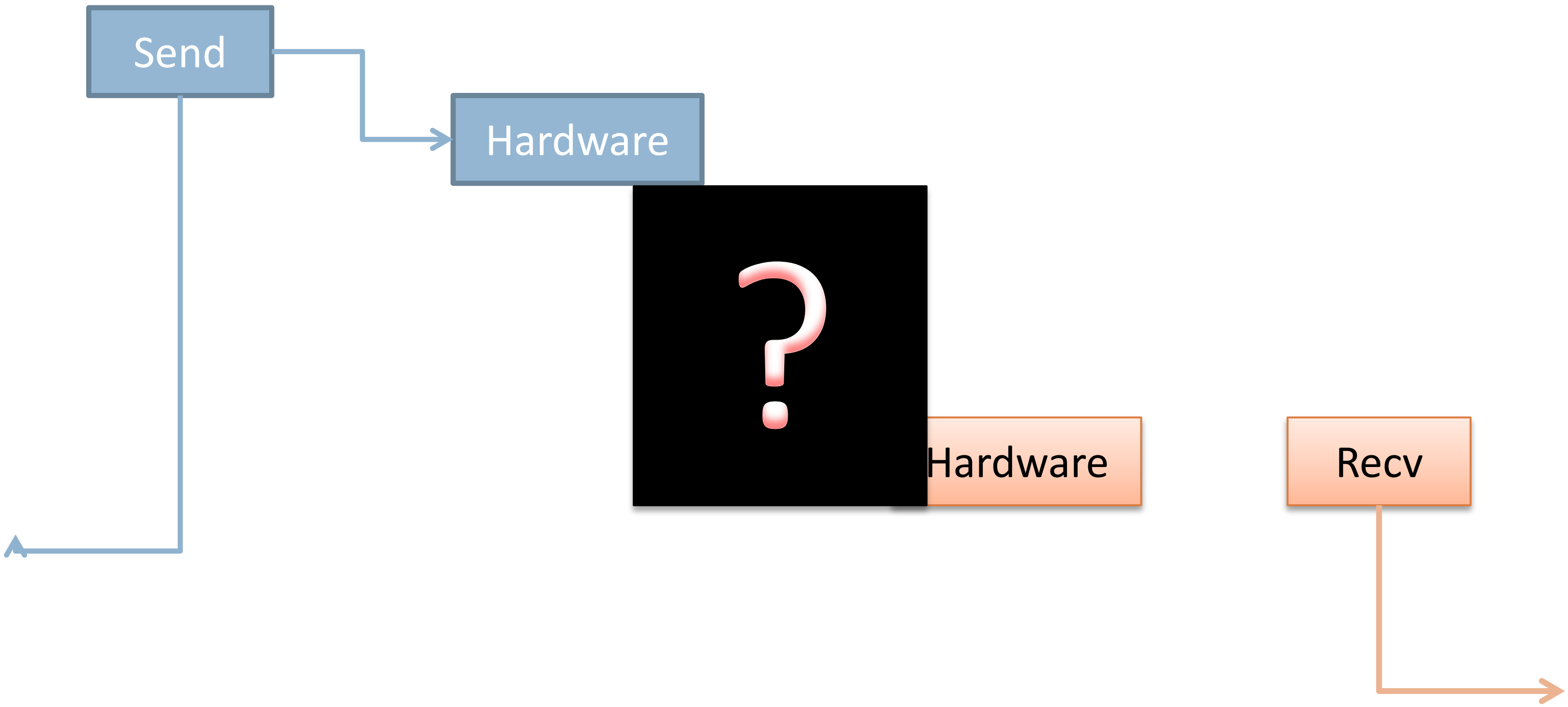
A non-blocking MPI_Isend returns immediately. So the value sent may either be 1. or 4.!

```
MPI_Irecv (&x,1,MPI_DOUBLE,….);
y=x;
```

A nonblocking MPI_Irecv returns immediately. So the value of y may equal x or not.

Isend / Irecv

What actually happens at MPI_Send nad MPI_Recv?



blocking communication

Synchronous
MPI_Ssend(…)
At return, the receiver has (at least) started to receive the message

Buffered
MPI_Bsend(….)
Message is copied before sending. At return, copying is finished.

Rendevouz
MPI_Rsend(…)
Receive has to be issued in advance. At return, message is as good as delivered

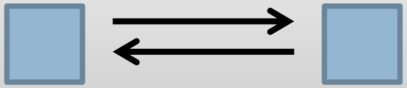
Why that many different Sends?

blocking communication

Situation:

```
if(myrank%2==0)      neighbor=myrank+1;
else                 neighbor=myrank-1;

MPI_Send(…, neighbor, …);
MPI_Receive(…, neighbor, …)
```



Deadlock:
Mutual waiting – forever (until time limit is reached)

```
if(myrank%2==0)      neighbor=myrank+1;
else                 neighbor=myrank-1;

MPI_Bsend(…, neighbor, …);
MPI_Receive(…, neighbor, …)
```

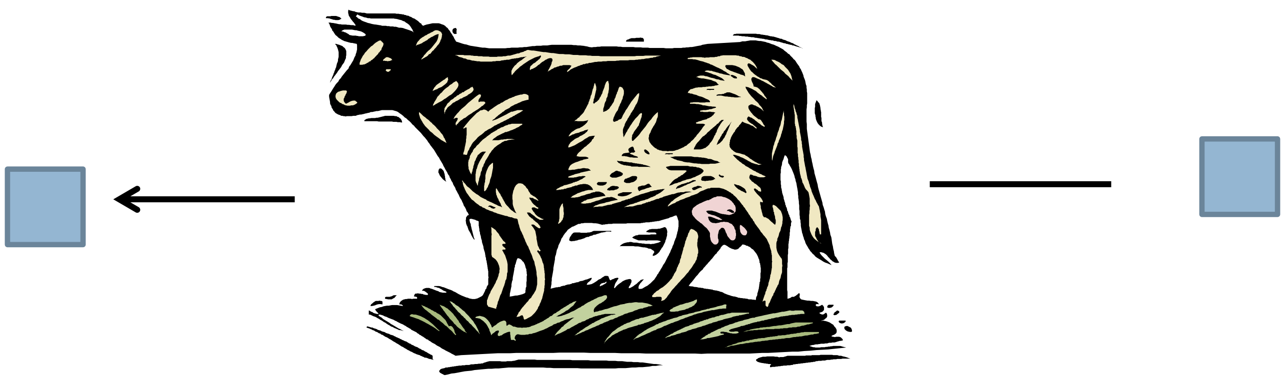
Both copy and proceed to receive

blocking communication

Situation:

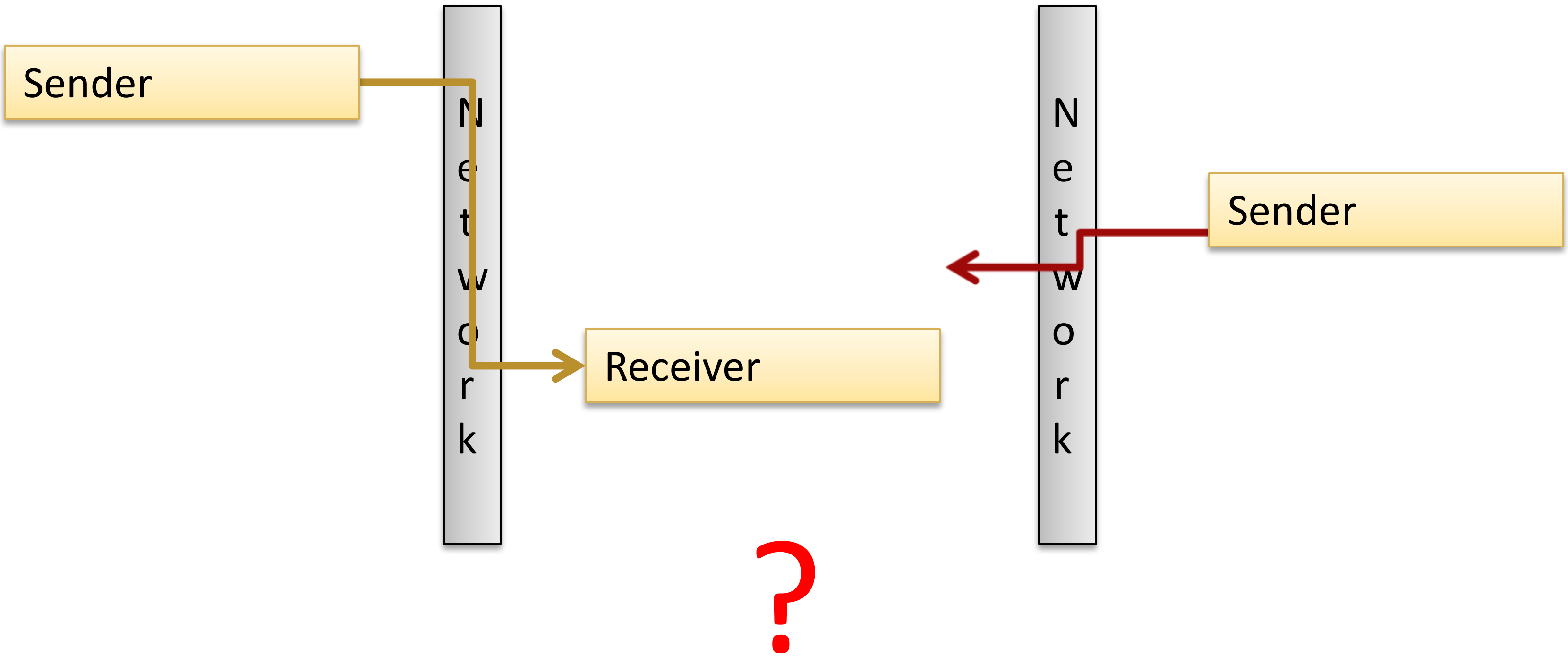
```
N=100 000 000;

MPI_Send(Buffer,N,MPI_INT,neighbor, …);
```



Everything get's stuck, if the huge message is not deliverable.
MPI_Rsend guarantees that receiver is ready.

Using wildcards



Be careful when using MPI_ANY_TAG, MPI_ANY_SOURCE

nonblocking communication

all blocking communications exist in a nonblocking version

MPI_Isend
MPI_Ibsend
MPI_Irsend
MPI_Issend

I is synonymous for immediate

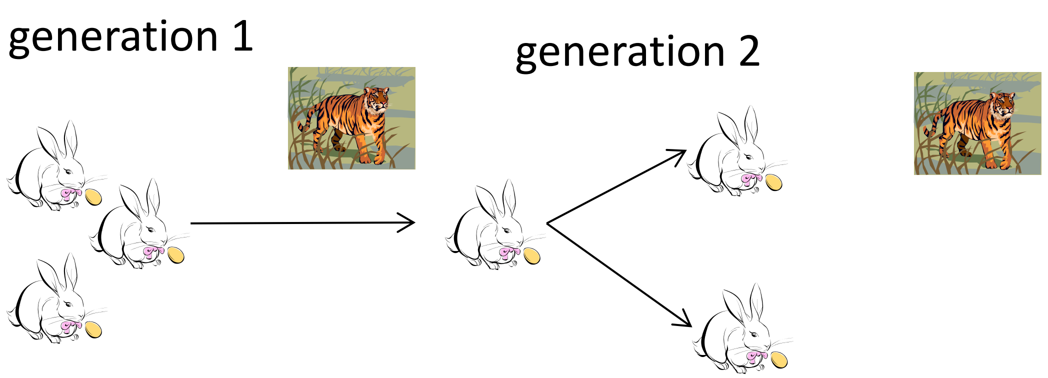
What are nonblocking send operations good for?

Example

Write a client-server application.

Problem: Find best fitting parameters out of a set of 50 parameters.
Solution (genetic algorithm):

1. Try 1000 independent combinations of parameters (called a generation).
2. Select the 10 best fitting combinations (called evolutionary selection).
3. Generate 1000 independent combinations based on the 10 best fitting (called mutation).
4. Go back to step 1 until result is not changing any more.



Example

server – clients type of code.

One master does the bookkeeping, collects results and organizes generations.
Slaves do work and report to master.

Which slave will finish it's work in which order?
How to prevent, that messages from hundreds of slaves block the network card?

One answer – nonblocking receive on the masters side.

nonblocking receive

```
...
 MPI_Comm_rank(comm,&myrank);                                /* get ID */
 MPI_Comm_size(comm,&anz);                          /* number of Tasks */
 if(myrank==0) {                                  /* specify master task */
  /* use as many receive buffers as there are tasks */
  for(i=1;i<anz;i++)
    MPI_Irecv(&puffer[i],MAX,MPI_DOUBLE,i,tag,comm,&req[i]);
  i=1;
  for(j=1;j<anz;j++) {
   for(info=0;!info;i=i%(anz-1)+1)              /* data received? */
      MPI_Test(&req[i],&info,&status);
   MPI_Get_count(&status,MPI_DOUBLE,&ndat);           /* how much */
   do_work(&puffer[i],ndat);       /* evaluate, set up new parameters */
  }
 } else {                                           /* slave tasks */
  evaluate_parameter_combination(quality,&ndat);
  MPI_Send(quality,ndat,MPI_DOUBLE,0,tag,comm);
 }
 ...
```

usage of nonblocking sending

```
MPI_Isend(…&request);
set_timer(&start);
.
get_timer(&jetzt);
MPI_Test(&request,&flag,&status);
if(!flag && jetzt-start>erlaubt) {
 printf(„Receiver is dead – use another one\n“);
  .
}
```

- blocking send operations get stuck if messages are not received – whole application has to be killed.
- nonblocking send operations allow the introduction of a timer and appropriate actions

non blocking completion

Non blocking communication requires synchronization.

`MPI_Wait(&request, &status)`

(blocking) wait until communication is finished

`MPI_Test(&request, &flag, &status)`

look whether communication has finished. Flag is true (1) if finished, else false (0).

Einführung in das Hochleistungsrechnen

Introduction to High Performance Computing

VIELEN DANK
THANK YOU