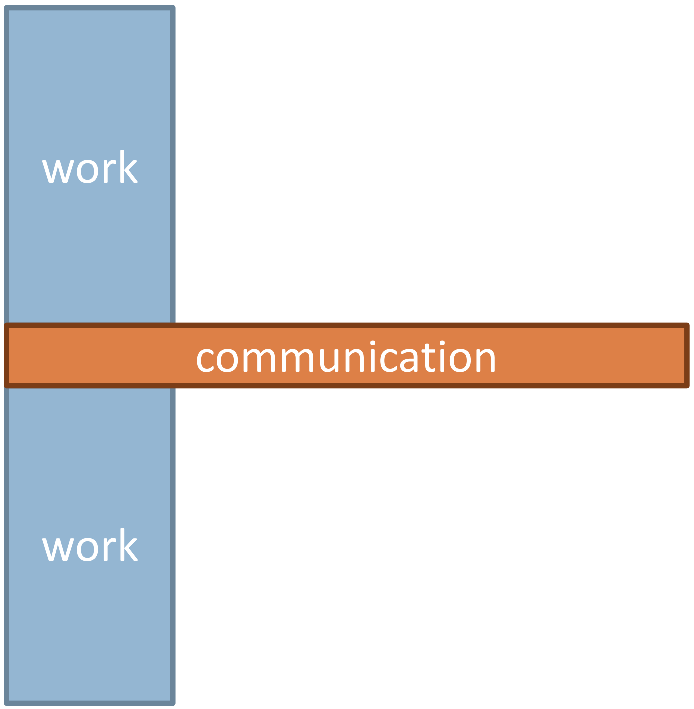
Introduction to
High Performance Computing

MPI (2)

one sided communication

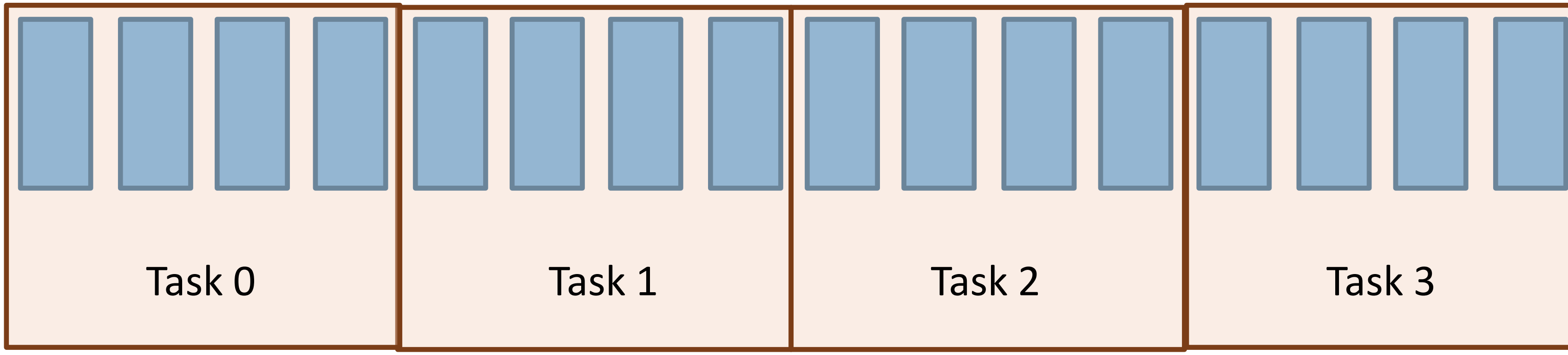
Point-to-point and collective communications require active communication



interrupt

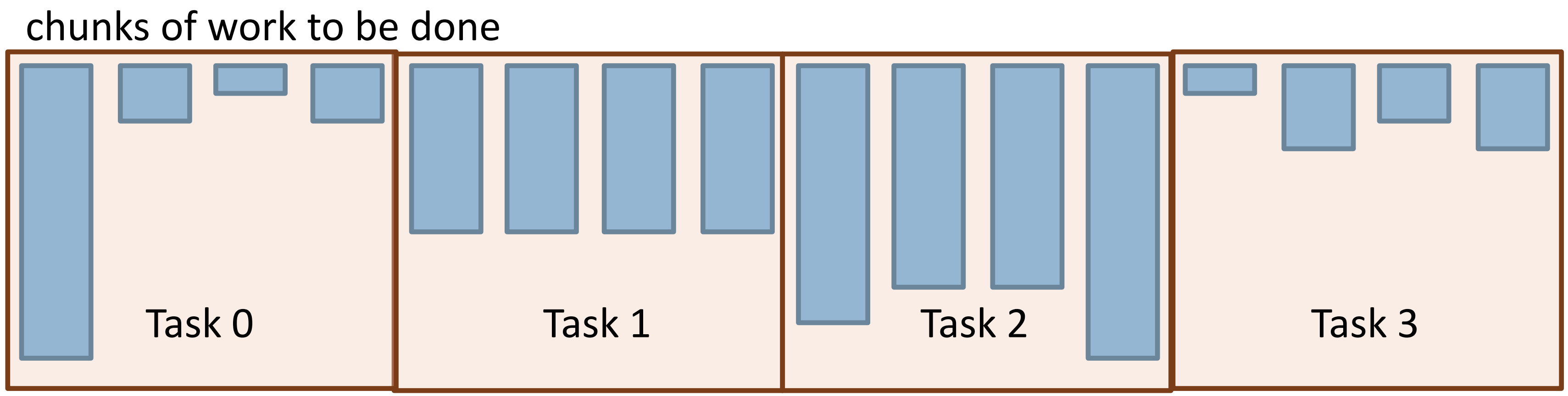
one sided communication

chunks of work to be done



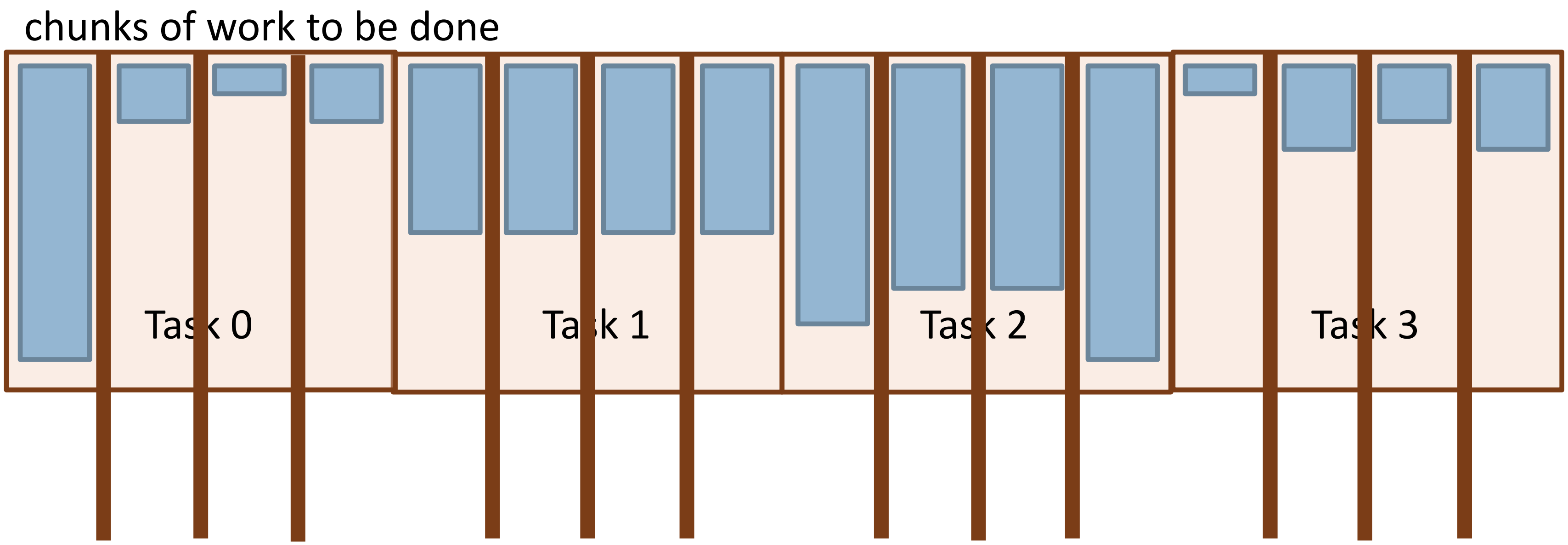
- static scheduling
- data resides at tasks

one sided communication



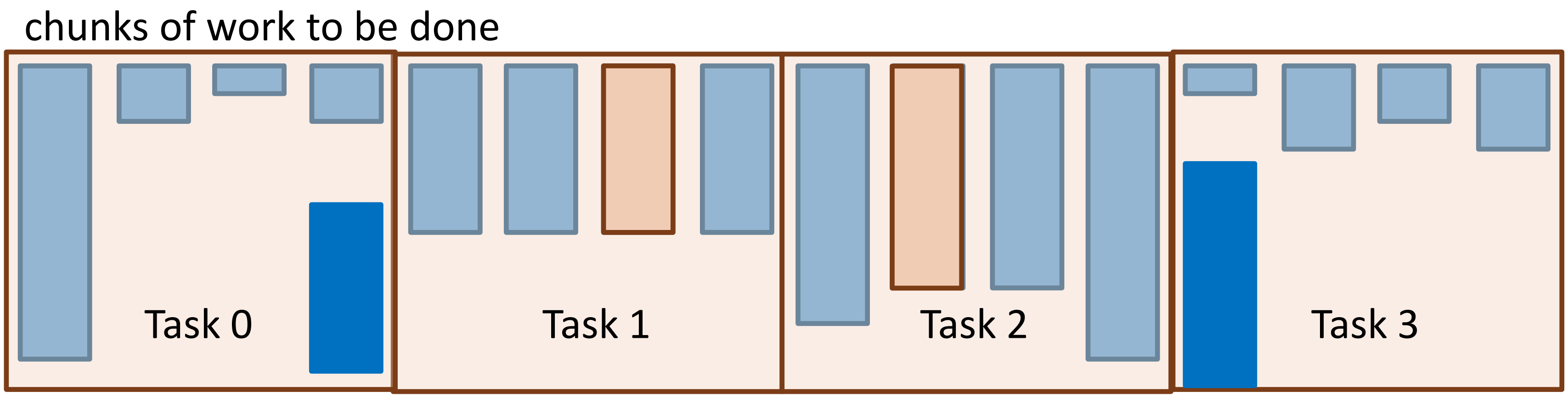
- dynamic scheduling
- data resides at tasks
- data needs to be moved from busy to idle task

one sided communication



- interrupt after every chunk required
- sending data occupies busy task furthermore

one sided communication



- idle task fetches data
- owner is undisturbed

one sided communication

```
...
 MPI_Comm_rank(comm,&myrank);
 MPI_Comm_size(comm,&anz);
 MPI_Comm_group(comm,&group);

 /* Mastertask */
 if(myrank==0) {
   for(i=1;i<anz;i++)

MPI_Win_create(&puffer[i],MAX*sizeof(double),sizeof(double),
                   MPI_INFO_NULL,comm,&win[i]);

    MPI_Win_post(group,0,win[i]);
   }
```

Generate memory windows
collective

Open memory window
All tasks may read and write the memory window independently

one sided communication

```
}else{

  for(i=1;i<anz;i++)

  MPI_Win_create(&puffer[i],0,1,MPI_INFO_NULL,comm,&win[i]);
   viel_Arbeit(Ergebnis,&ndat);

   MPI_Win_start(group,0,win[myrank]);
   MPI_Put(Ergebnis,ndat,MPI_DOUBLE,0,0,ndat,MPI_DOUBLE,
              &win[myrank]);

   MPI_Win_complete(win[myrank]);
  }
  for(i=1;i<anz;i++)
      MPI_Win_free(win[i]);
```

generate memory windows
collective
size 0 possible

start memory epoche
blocking

writing data to the window

end of memory epoche
blocking

close memory window
collective

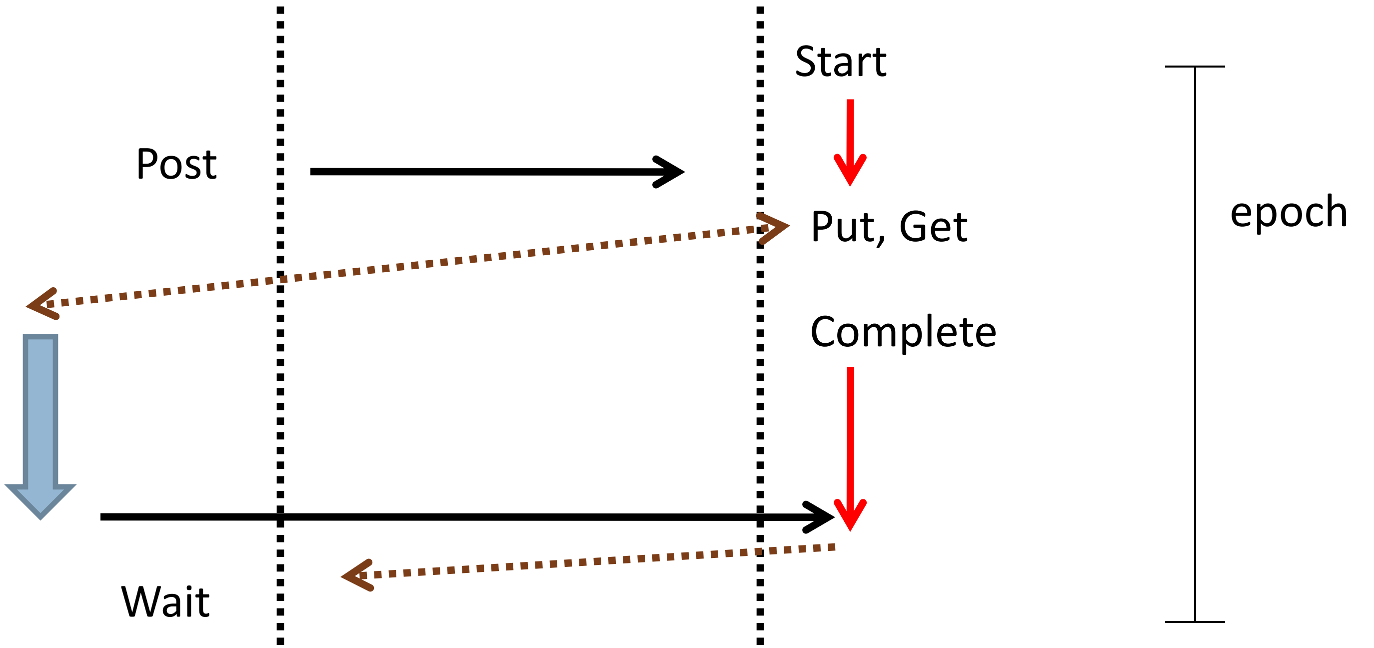
one sided communication

```
 /* Mastertask */
 if(myrank==0) {
   for(i=1;i<anz;i++)
    MPI_Win_wait(win[i]);

//do something with client data

   for(i=1;i<anz;i++)
    MPI_Win_free(win[i]);
}
```

Wait for completion of remote access

close memory window
collective

one sided communication



owner of window
target

remote access of window
origin

one sided communication

```
MPI_Win_create(&baseptr,size,disp_unit,info,comm,&win);
```

Allocates memory of given size (may be 0) at allocated adress (baseptr) and opens a memory window.

Collective

disp_unit may be 1 (byte) or sizeof(datatype), like 4 for MPI_FLOAT.

```
MPI_Win_allocate(size, disp_unit,info,comm, &baseptr,&win);
```

Allocates memory of given size (may be 0) and opens a memory window.

Collective

May allocate memory especially suited for RMA like pinned memory.

one sided communication

```
MPI_Win_allocate(size, disp_unit,info,comm, &baseptr,&win);
```

Allocates memory of given size (may be 0) and opens a memory window.

Collective

May allocate memory especially suited for RMA like pinned memory.

```
MPI_Win_create_dynamic(info, comm, &win);
```

Opens an empty window, there later memory (several times) may be attached.

Collective

Intented for linked lists there each leaf (new or malloc) would require a new window.

```
MPI_Win_attach(win, &base, size);
```

```
MPI_Win_detach(win, &base);
```

```
MPI_Win_free(&win);
```

one sided communication

```
MPI_Win_allocate_shared(
                  size,disp_unit,info,comm, &baseptr,&win);
```

Allocates shared memory of given size (may be 0) and opens a memory window.

Collective

Users responsibility to use it for hardware shared memory. Each task uses its own shared memory range.

```
MPI_Win_shared_query(win,rank,&size, &disp_unit, &baseptr);
```

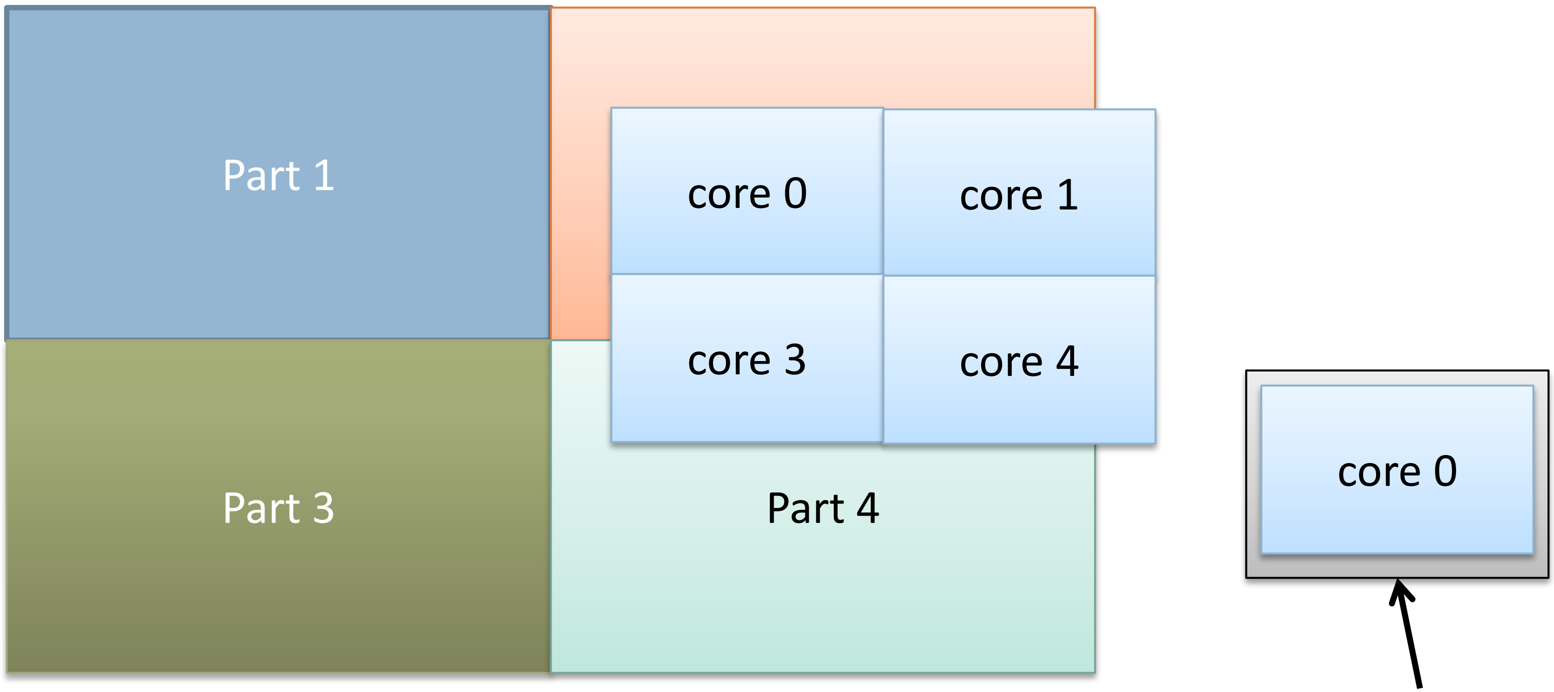
Local function.

Returns pointer, size and disp_unit to shared memory allocated by rank.

- Allocate close memory on NUMA architecture.
- Users responsibility to keep memory consistent.

shared memory

Outline for usage of shared memory:



halo or ghost region
overlapping with neighbours

shared memory

partial differential equation with scheme:

u[i,j] = f(u[i,j], u[i+1,j],u[i-1,j],u[i,j+1],u[i,j-1])

Calculation of u[i,j] is local, except at domain boundaries
Neighbour points are attached to allow update without communication (halo, ghost region)
After one iteration, information in this halo region has changed and an update is required

calculate domain
synchronization
exchange of halo region
synchronization

send data to different nodes
copy shared data on my node

one sided communication

```
MPI_Put(&o_addr, o_count, o_type, target_rank, t_disp,
t_count, t_type, win);
```

nonblocking

writes data in task **rank** of **win** at **baseptr+t_disp*disp_unit**

```
MPI_Rput(&o_addr, o_count, o_type, target_rank, t_disp,
t_count, t_type, win, &request);
```

Same as Put, but the reusability of o_addr may be synchronized with MPI_Wait via the request returned. The end of the communication epoche is not synchronized, e.g. the target process may not have received the data. Extra synchronization is required on the target.

one sided communication

```
MPI_Get(&o_addr, o_count, o_type, target_rank, t_disp,
t_count, t_type, win);
```

nonblocking

same as put but opposite direction

```
MPI_Rget(&o_addr, o_count, o_type, target_rank, t_disp,
t_count, t_type, win, &request);
```

Same as Get, but the usability of o_addr may be synchronized with MPI_Wait via the request returned.

one sided communication

```
MPI_Win_fence(0,win)
```

memory barrier (fence)

Collective

An RMA request started and ended with a fence is completed in every respect.

This is valid for the target as well as the origin of an operation.

Example

Given a large distributed array B:

task 1	task 2	B	task p

sort this array according to a Map (distributed as well) into a distributed array A.

- the sizes of each task are all equal to m
- A, B and Map are local arrays with indices 0:m-1
- Map contains global indices 0:m*p
- the rank of a task holding a global index is given by Map[]/m
- the position of a local element is given by Map[]%m

Example

```
int i, m, *Map, disp_unit, rank, size, t_disp, target_rank;
float *A, *B;
MPI_Win win; MPI_Comm comm;

disp_unit=sizeof(float);      // we count array elements
size=disp_unit*m;             // local size of array in byte

/* All tasks open a window at beginning of local array B */
MPI_Win_create(B,size,disp_unit,MPI_INFO_NULL,comm,&win);
MPI_Win_fence(0,win);     // start a RMA epoche
for(i=0;i<m;i++) {
       t_disp=Map[i]%m;
       target_rank=Map[i]/m;
       MPI_Get(&A[i], 1, MPI_FLOAT, target_rank, t_disp, 1,
               MPI_FLOAT, win);
}
MPI_Win_fence(0,win);     // end this RMA epoche
MPI_Win_free(&win);
```

Example

Previous example has to be improved before using it:

- local copies are fetched with MPI_Get
- each word is copied one by one (think about the communication latency)

one sided communication

MPI_Win_post(group,0,win)

Start of an RMA epoch called by target process (the one whos memory is read or written). non blocking

MPI_Win_start(group,0,win)

Start of an RMA epoch called by origin process. Return is delayed until target process has freed its window (MPI_Win_post).

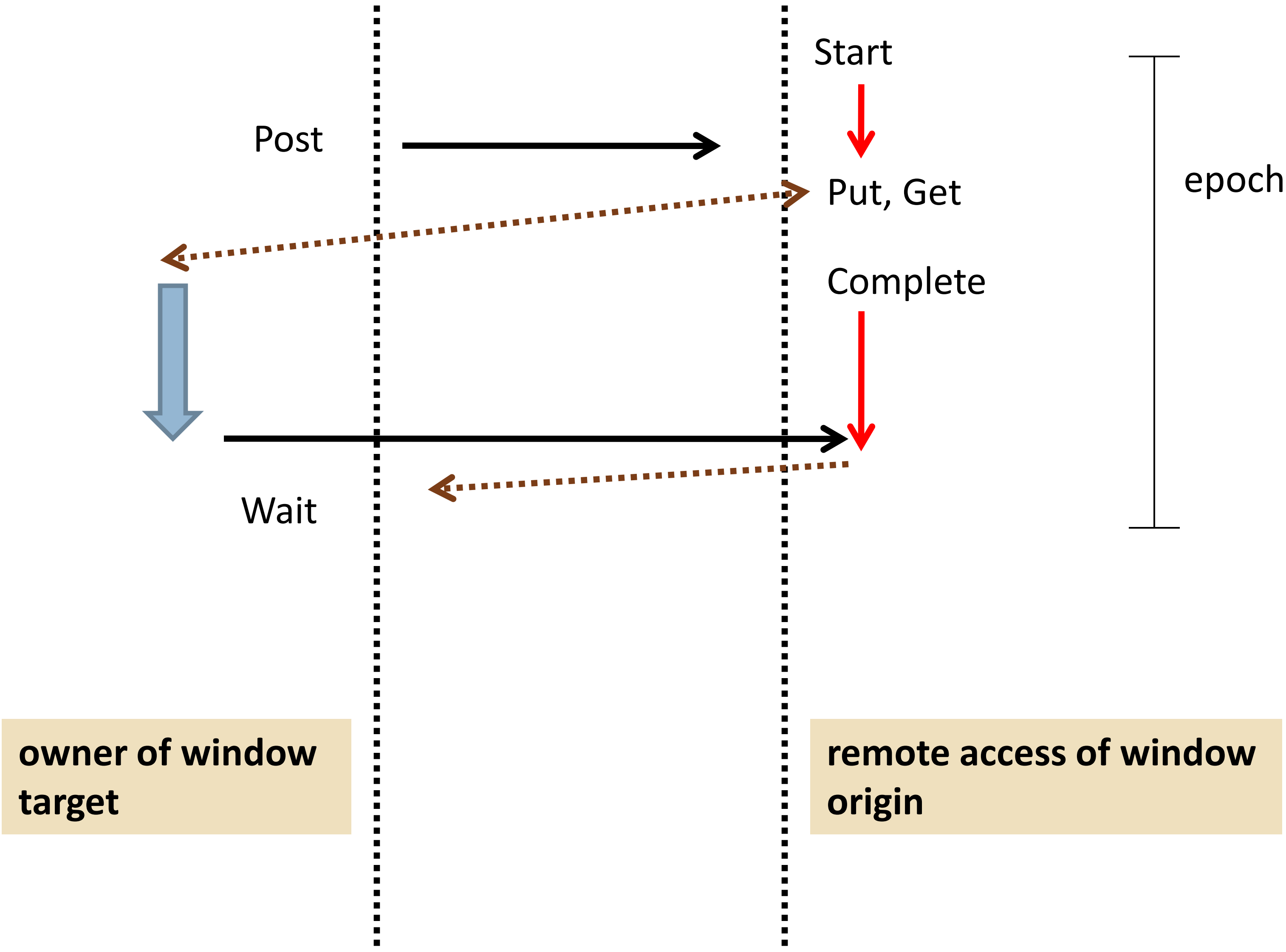
MPI_Win_complete(win)

Completes an RMA epoch on the origin process. Return is delayed until all RMA requests on the origin is finished.
MPI_Get: data is available at origin
MPI_Put: data buffer may be reused

MPI_Win_wait(win)

Ends an RMA epoch at the target. Return is delayed until MPI_Win_complete on origin and all memory operations on target are finished.

one sided communication



one sided communication

MPI_Win_lock(lock_type,rank,0,win)
Locks access to window on task rank and starts an access epoch to that window. lock_type may be MPI_LOCK_EXCLUSIVE or MPI_LOCK_SHARED.

MPI_Win_lock_all(0,win)
Locks access to all windows of the corresponding group (by callee only). Type: MPI_LOCK_SHARED.

MPI_Win_unlock(rank,win)
Unlock frees the window (end of an epoch). All memory accesses on origin and target side are completed after return.

MPI_Win_unlock_all(win)
Unlock_all frees the window (end of an epoch) after a lock_all. All memory accesses on origin and target side are completed after return.

one sided communication

MPI_Accumulate(&o_addr, o_count, o_type, t_rank, t_disp, t_count, t_type, op, win)

Accumulates (t.ex. adds) data in o_addr to data on the target at base_ptr+t_disp*disp_unit.
All predifined operations (s. MPI_Reduce) are possible, but no user defined functions.
Not collective.
MPI_REPLACE as additional operation (= similar to put).

MPI_Raccumulate(&o_addr, o_count, o_type, t_rank, t_disp, t_count, t_type, op, win, &req)

Same functionality as MPI_Accumulate but the end may be controled with help of req (MPI_Test, MPI_Wait).
Completion of this function means that o_addr may be reused, but not necessarily that operation is finished on the target.

Example

```
int i, m, *Map, disp_unit, rank, size, t_disp, target_rank;
float *A, *B;
MPI_Win win; MPI_Comm comm;

disp_unit=sizeof(float);      // we count array elements
size=disp_unit*m;             // local size of array in byte

/* All tasks open a window at beginning of local array B */
MPI_Win_create(B,size,disp_unit,MPI_INFO_NULL,comm,&win);
MPI_Win_fence(0,win);     // start a RMA epoche
for(i=0;i<m;i++) {
       t_disp=Map[i]%m;
       target_rank=Map[i]/m;
       MPI_Accumulate(&A[i], 1, MPI_FLOAT, target_rank,
               t_disp, 1,MPI_FLOAT, win);
}
MPI_Win_fence(0,win);     // end this RMA epoche
MPI_Win_free(&win);
```

Example

Similar example as previous for MPI_Get. This time

$$B(dist) = \sum_{Map(i)} A(i)$$

one sided communication

```
MPI_Get_accumulate(&o_addr, o_count, o_type, &result, r_count, r_type, t_rank, t_disp,
                   t_count, t_type, op, win)
```

Very general function combining an MPI_Accumulate with an MPI_Get operation (as well available as MPI_Rget_accumulate).

```
MPI_Fetch_and_op(&o_addr, &result, type, t_rank, t_disp, op, win)
```

Simplified version of MPI_Get_accumulate valid for just a single word with basic datatype. If op is MPI_SUM it works like:

result = value at target(t_disp)
target(t_disp) = value at o_addr

```
MPI_Compare_and_swap(&o_addr, &compare, &result, type, t_rank, t_disp, win)
```

Works like:
result = target(t_disp);
if(compare==target(t_disp)) target(t_disp)=o_addr;

Einführung in das Hochleistungsrechnen

Introduction to High Performance Computing

VIELEN DANK
THANK YOU