Introduction to Parallel Processing

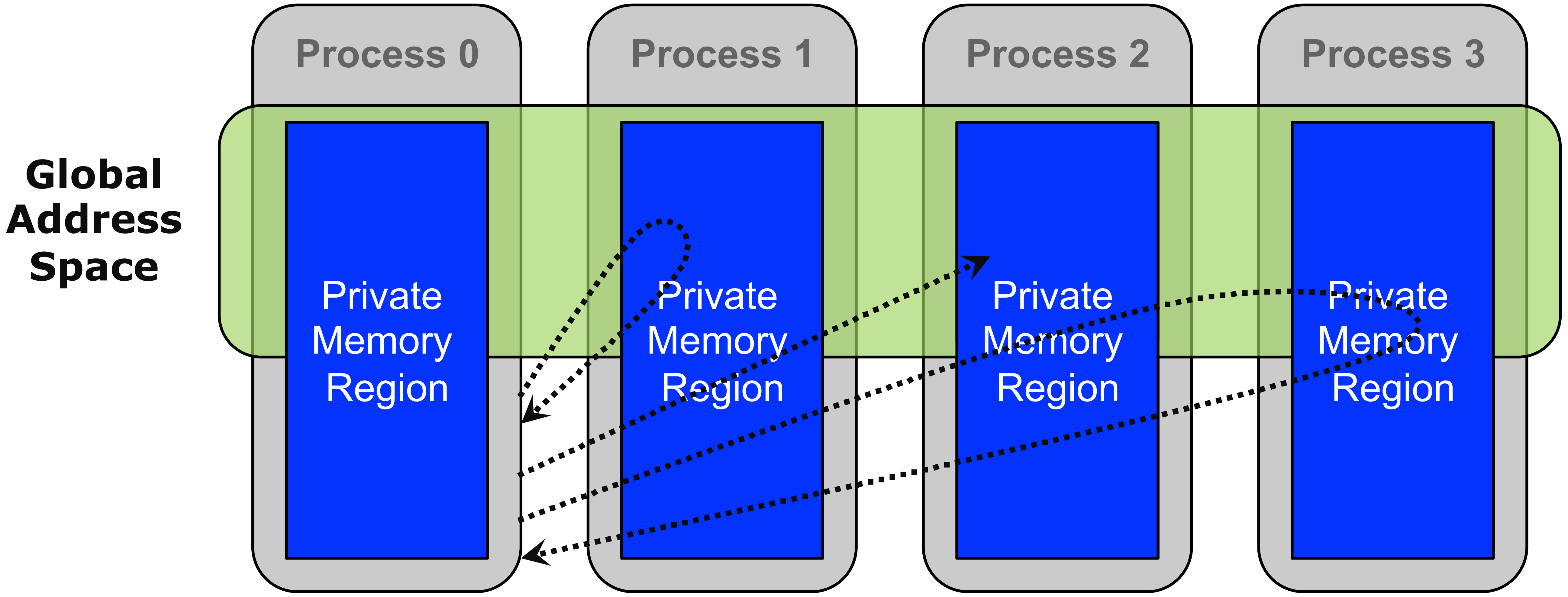
Lecture 21 : One-Sided Communication

Professor Amanda Bienz

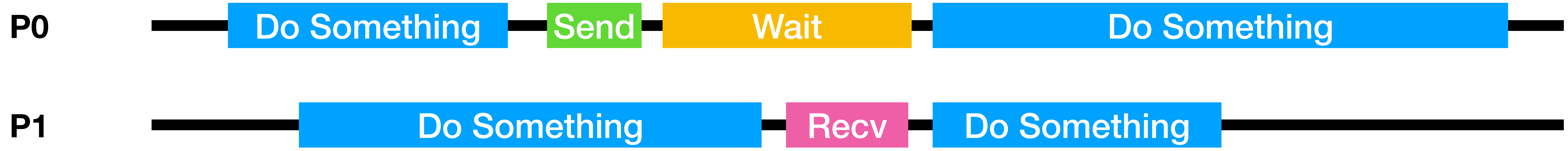
https://wgropp.cs.illinois.edu/courses/cs598-s15/lectures/lecture34.pdf
https://wgropp.cs.illinois.edu/courses/cs598-s15/lectures/lecture35.pdf

Basic Idea

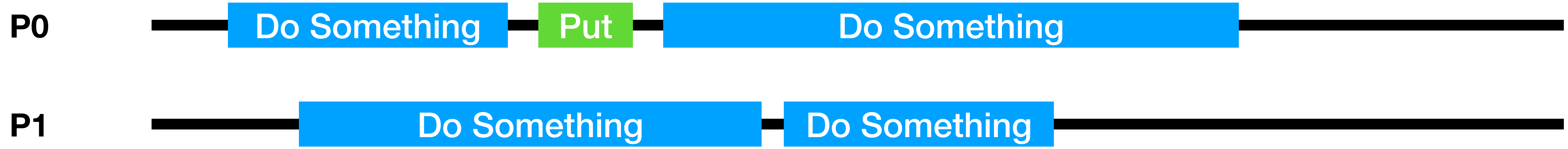
- In standard communication, the process of origin sends data to the destination process, and the receiving process posts a receive.
- One-sided communication decouples data movement with process synchronization
  - Each process exposes part of its memory to other processes
  - Other processes can read from or write to this exposed section of memory



Comparison with Send/Recv

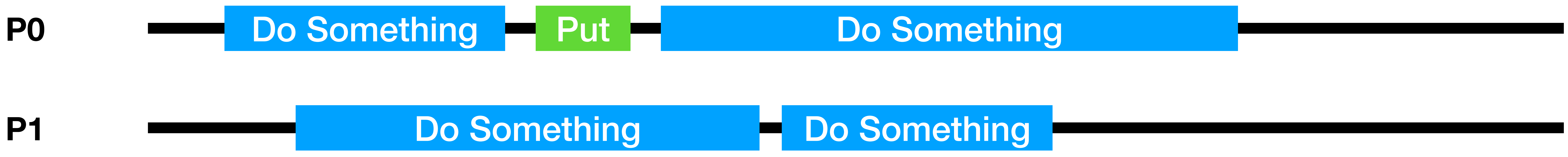


MPI Send/Recv Timeline Example



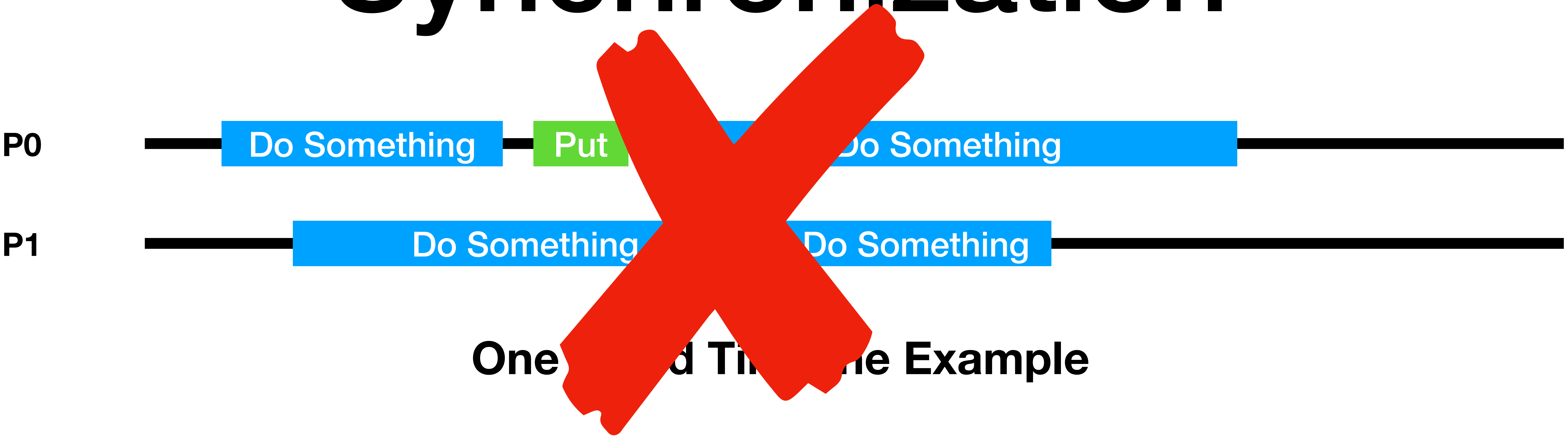
One Sided Timeline Example

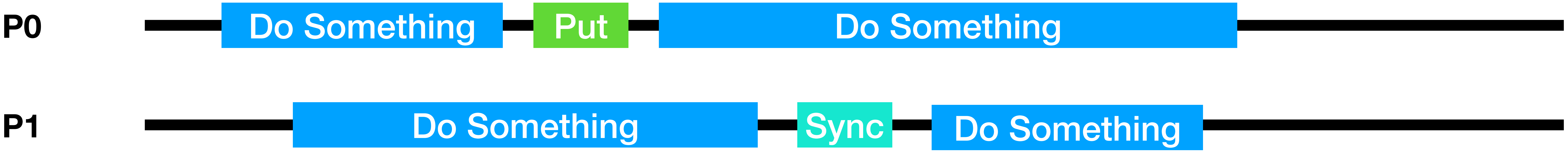
Synchronization



One Sided Timeline Example

Synchronization





One Sided Timeline Example

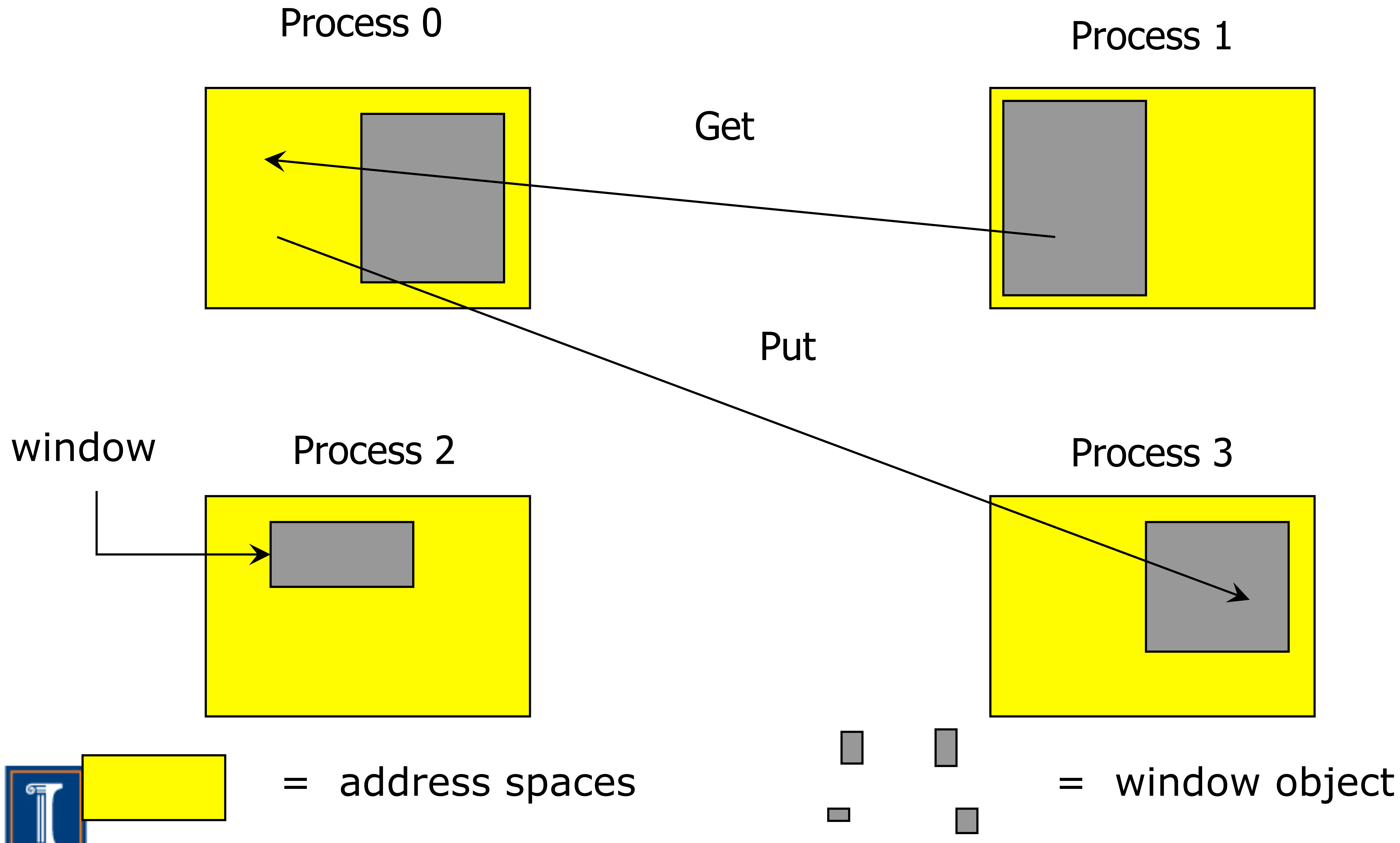
Synchronization

- Can do multiple data transfers and then a single instance of synchronization
- Example : irregular communication where pattern is not known before hand but data locations are known (i.e. SpMV) can just get data instead of telling sending process that I need it
- Additional speedup on shared memory systems (hardware support for remote memory accesses)

Public Memory

- Data allocated by a process is, by default, only accessible by this process
- Once memory is allocated, user can declare memory as remotely accessible
  - MPI window : remotely accessible memory
- MPI_Win_create : collective operation that creates window object (specifies which memory on each process is remotely accessible)
- MPI_Win_free : deallocates window object

Remote Memory Access
Windows and Window Objects



Window Create Modes

- MPI_WIN_CREATE : have already allocated buffer that would like to be remotely accessible
- MPI_WIN_ALLOCATE : want to create a buffer and directly make it remotely accessible
- MPI_WIN_CREATE_DYNAMIC : don't have a buffer yet but will in the future
- MPI_WIN_ALLOCATE_SHARED : want multiple processes on same node to share a buffer

MPI_WIN_CREATE

```
int MPI_Win_create(void *base, MPI_Aint size,
       int disp_unit, MPI_Info info,
       MPI_Comm comm, MPI_Win *win)
```

- Expose a region of memory in an RMA window
  - Only data exposed in a window can be accessed with RMA ops.
- Arguments:
  - base - pointer to local data to expose
  - size - size of local data in bytes (nonnegative integer)
  - disp_unit - local unit size for displacements, in bytes (positive integer)
  - info - info argument (handle)
  - comm - communicator (handle)
  - win – window object (handle)

MPI_WIN_ALLOCATE

```
int MPI_Win_allocate(MPI_Aint size, int disp_unit,
      MPI_Info info,
      MPI_Comm comm, void *baseptr, MPI_Win *win)
```

- Create a remotely accessible memory region in an RMA window
  - Only data exposed in a window can be accessed with RMA ops.
- Arguments:
  - size - size of local data in bytes (nonnegative integer)
  - disp_unit- local unit size for displacements, in bytes (positive integer)
  - info - info argument (handle)
  - comm - communicator (handle)
  - baseptr - pointer to exposed local data
  - win – window object (handle)

MPI_WIN_CREATE_DYNAMIC

```
int MPI_Win_create_dynamic(MPI_Info info, MPI_Comm comm,
                           MPI_Win *win)
```

- Create an RMA window, to which data can later be attached
  - Only data exposed in a window can be accessed with RMA ops
- Initially "empty"
  - Application can dynamically attach/detach memory to this window by calling MPI_Win_attach/detach
  - Application can access data on this window only after a memory region has been attached
- Window origin is MPI_BOTTOM
  - Displacements are segment addresses relative to MPI_BOTTOM
  - Must tell others the displacement after calling attach

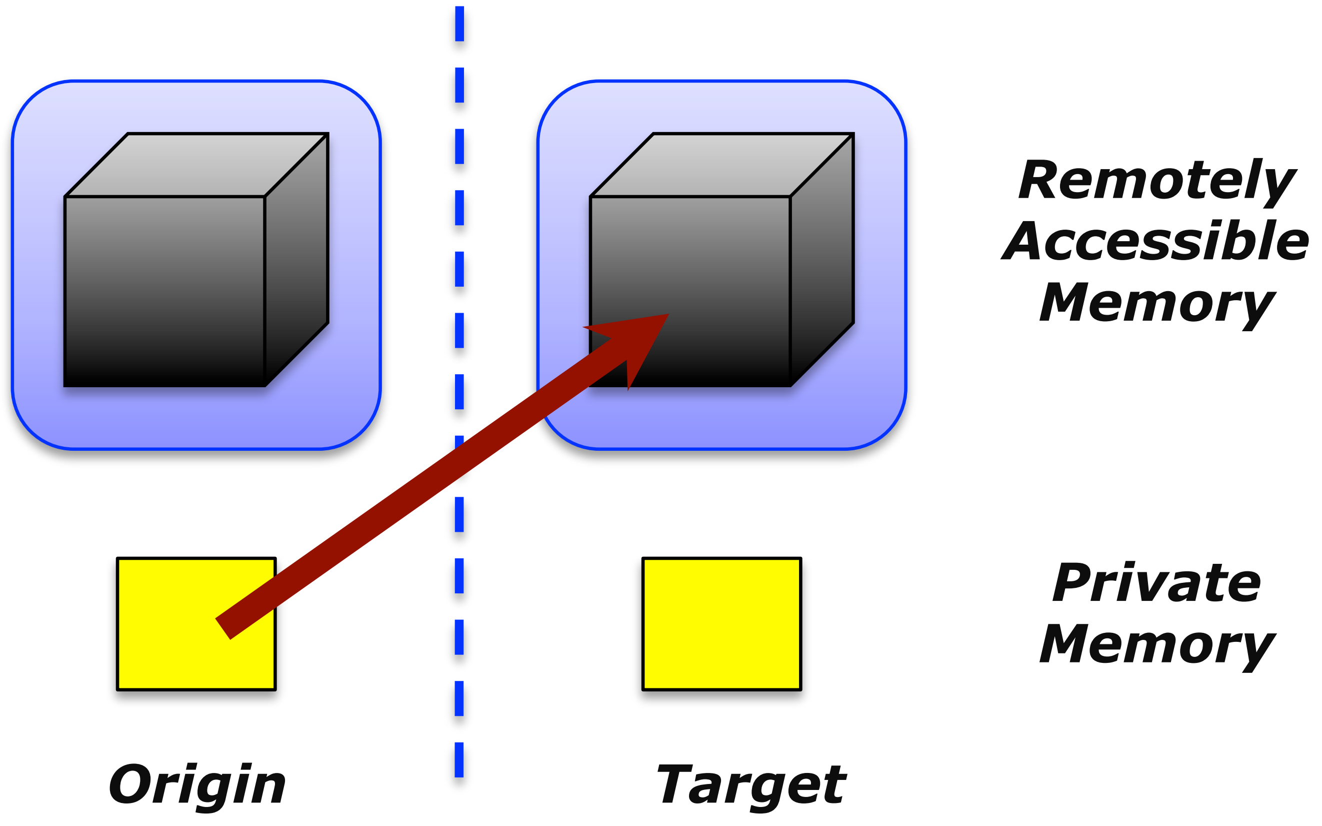
MPI RMA

- After creating MPI window:
  - MPI_Put moves data from local memory to a remote memory
  - MPI_Get retrieves data from a remote memory to local memory
  - MPI_Accumulate updates remote memory using local values
- All data movement operations are non-blocking
- **We need synchronization on window object to tell when operation is complete**

Data movement: *Put*

```
MPI_Put(void *origin_addr, int origin_count,
    MPI_Datatype origin_dtype, int target_rank,
    MPI_Aint target_disp, int target_count,
    MPI_Datatype target_dtype, MPI_Win win)
```

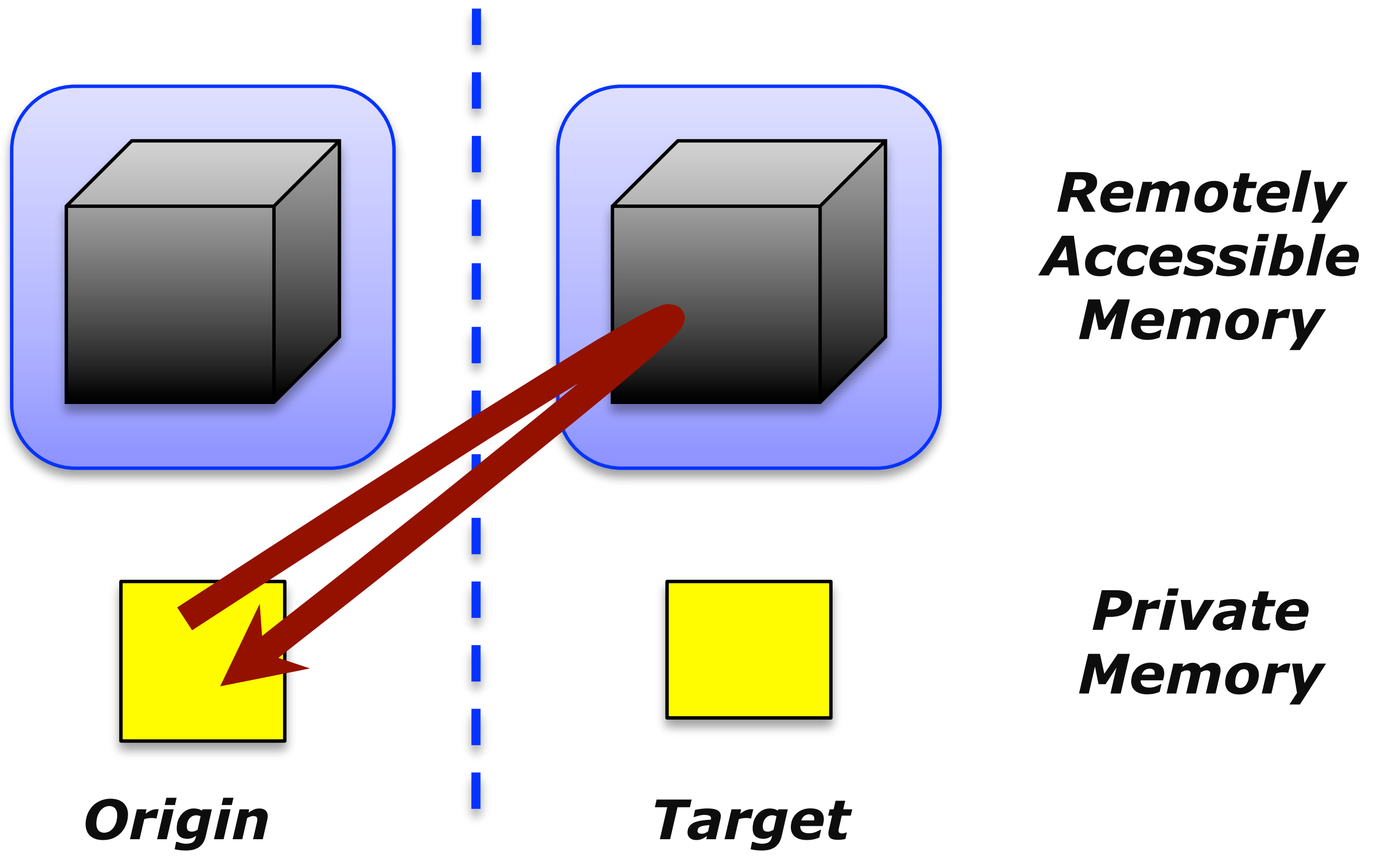
- Move data from origin, to target
- Separate data description triples for origin and target



Data movement: *Get*

```
MPI_Get(void *origin_addr, int origin_count,
    MPI_Datatype origin_dtype, int target_rank,
    MPI_Aint target_disp, int target_count,
    MPI_Datatype target_dtype, MPI_Win win)
```

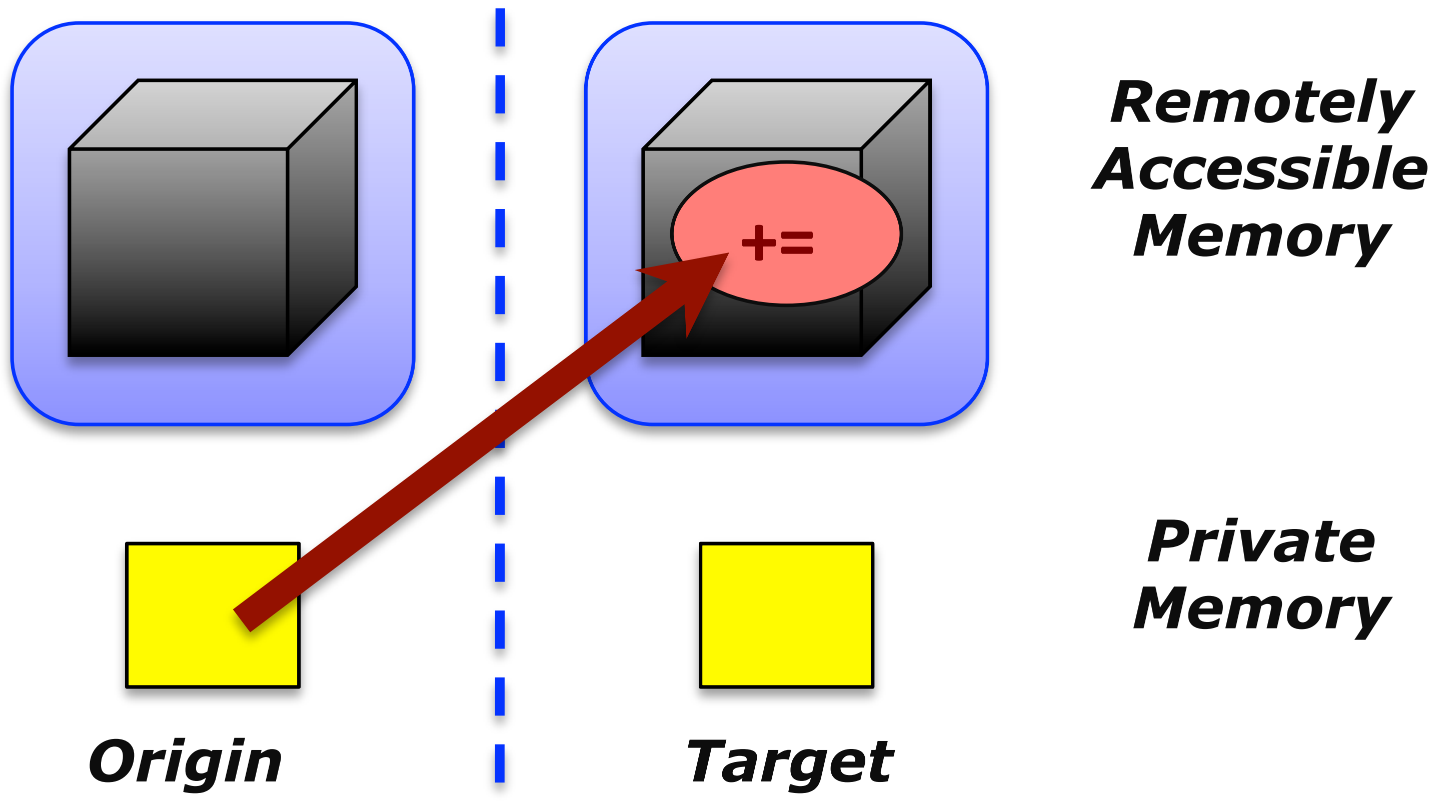
- Move data to origin, from target



Atomic Data Aggregation: *Accumulate*

```
MPI_Accumulate(void *origin_addr, int origin_count,
    MPI_Datatype origin_dtype, int target_rank,
    MPI_Aint target_disp, int target_count,
    MPI_Datatype target_dtype, MPI_Op op, MPI_Win win)
```

- Element-wise atomic update operation, similar to a put
  - Reduces origin and target data into target buffer using op argument as combiner
  - Predefined ops only, no user-defined operations
- Different data layouts between target/origin OK
  - Basic type elements must match
- Op = MPI_REPLACE
  - Implements *f(a,b)=b*
  - Element-wise atomic PUT



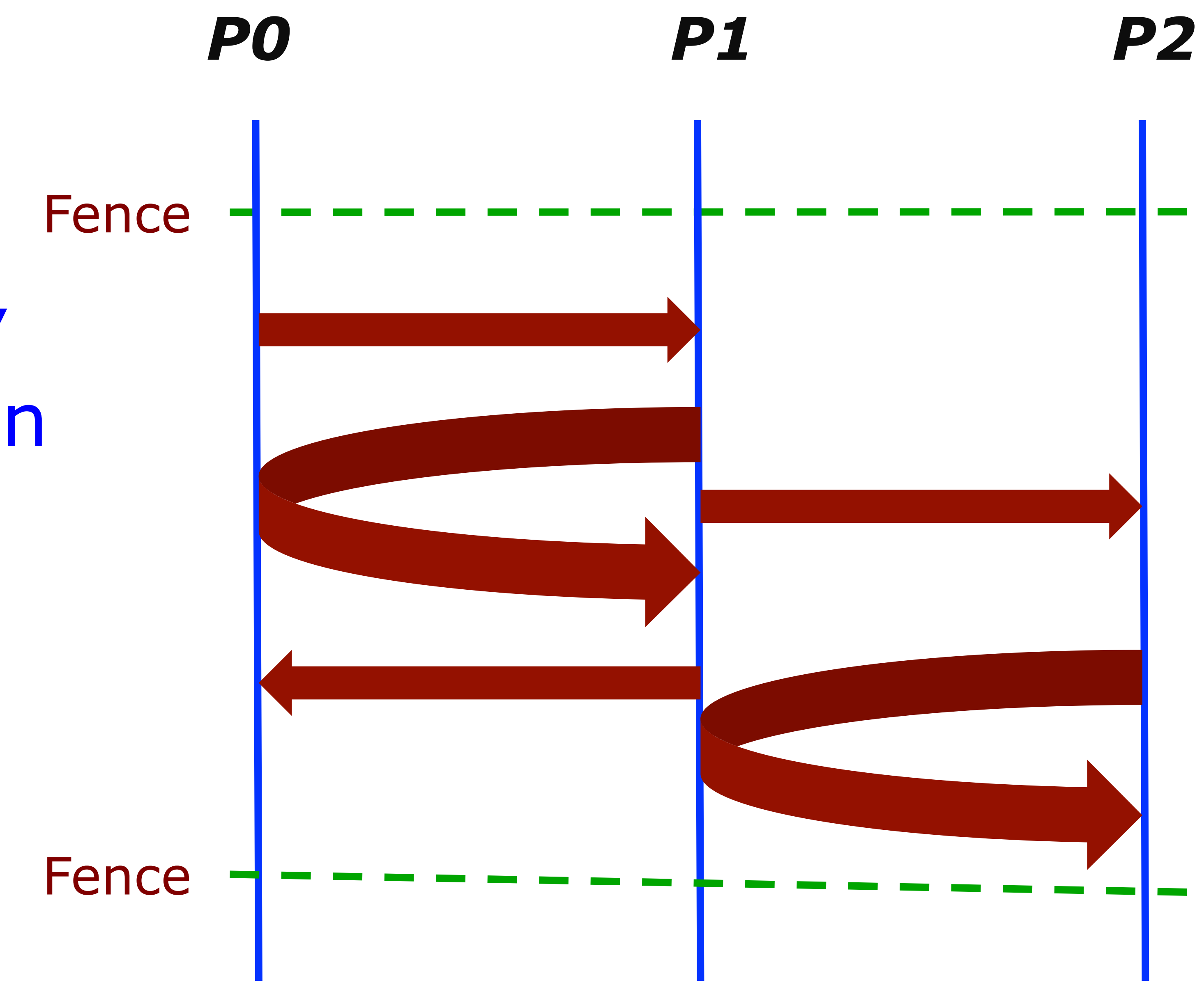
Synchronization

- Three MPI modes of synchronization
  - Fence
  - Post-start-complete-wait
  - Lock / Unlock

Fence: Active Target Synchronization

```
MPI_Win_fence(int assert, MPI_Win win)
```

- Collective synchronization model
- Starts *and* ends access and exposure epochs on all processes in the window
- All processes in group of "win" do an MPI_WIN_FENCE to open an epoch
- Everyone can issue PUT/GET operations to read/write data
- Everyone does an MPI_WIN_FENCE to close the epoch
- All operations complete at the second fence synchronization

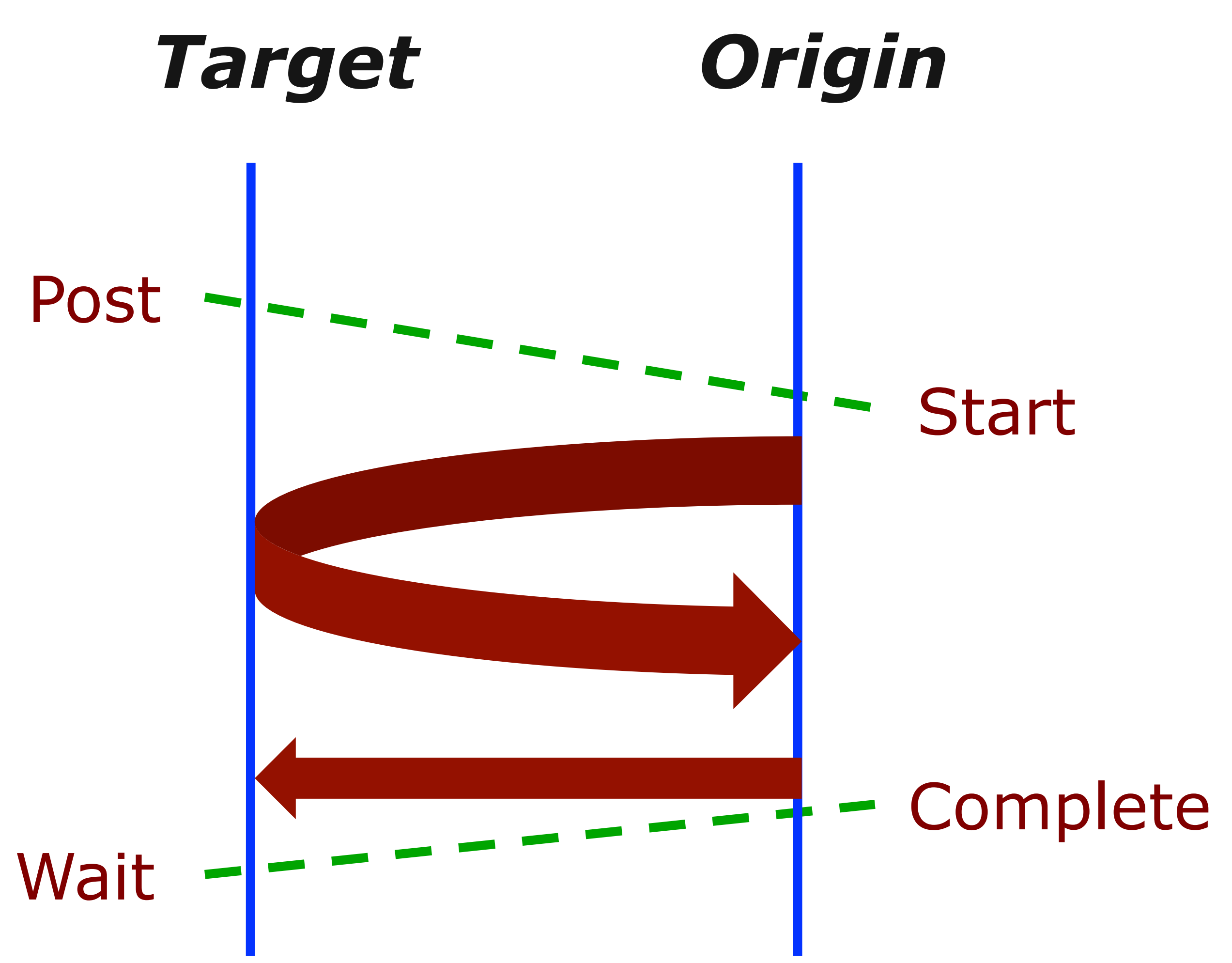


PARALLEL@ILLINOIS

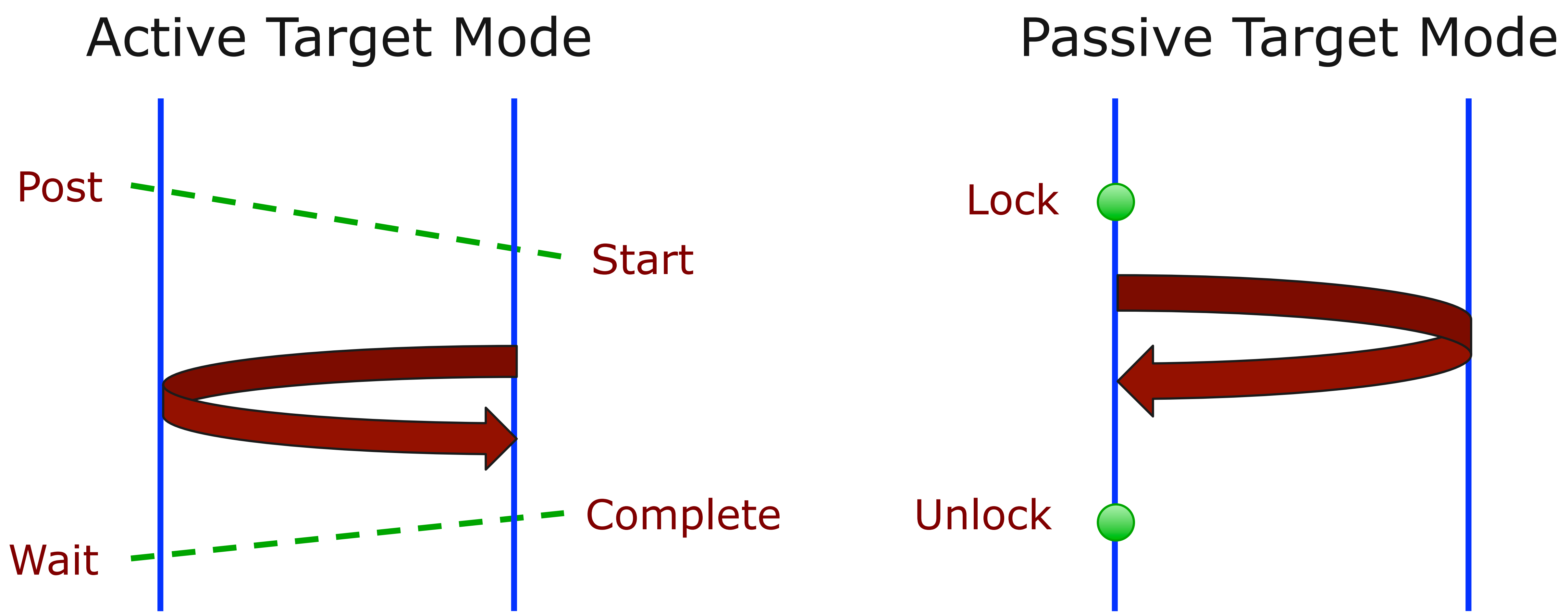
PSCW: Generalized Active Target Synchronization

```
MPI_Win_post/start(MPI_Group grp, int assert, MPI_Win win)
MPI_Win_complete/wait(MPI_Win win)
```

- Like FENCE, but origin and target specify who they communicate with
- Target: Exposure epoch
  - Opened with MPI_Win_post
  - Closed by MPI_Win_wait
- Origin: Access epoch
  - Opened by MPI_Win_start
  - Closed by MPI_Win_complete
- All synchronization operations may block, to enforce P-S/C-W ordering
  - Processes can be both origins and targets



Lock/Unlock: Passive Target Synchronization



- Passive mode: One-sided, *asynchronous* communication
  - Target does **not** participate in communication operation
- Shared memory-like model

Passive Target Synchronization

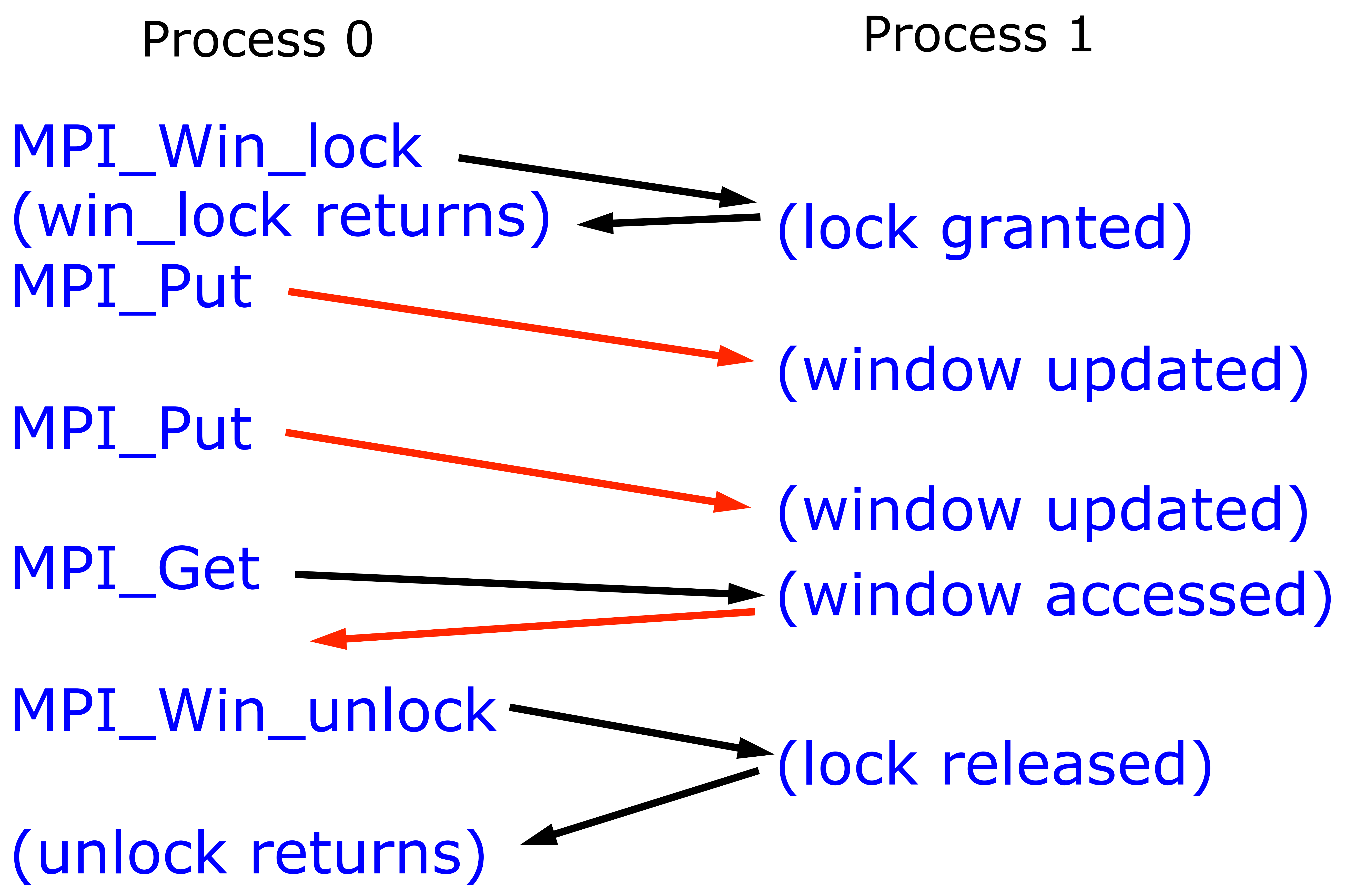
```
MPI_Win_lock(int locktype, int rank, int assert, MPI_Win win)
```

```
MPI_Win_unlock(int rank, MPI_Win win)
```

```
MPI_Win_flush/flush_local(int rank, MPI_Win win)
```

- Lock/Unlock: Begin/end passive mode epoch
  - Target process does not make a corresponding MPI call
  - Can initiate multiple passive target epochs to different processes
  - Concurrent epochs to same process not allowed (affects threads)
- Lock type
  - SHARED: Other processes using shared can access concurrently
  - EXCLUSIVE: No other processes can access concurrently
- Flush: Remotely complete RMA operations to the target process
  - After completion, data can be read by target process or a different process
- Flush_local: Locally complete RMA operations to the target process

Data Moves Early



Data Moves Early

Process 0	Process 1
MPI_Win_lock	
(win_lock returns)	(lock granted)
MPI_Put	
	(window updated)
MPI_Put	
	(window updated)
MPI_Get	(window accessed)
MPI_Win_unlock	(lock released)
(unlock returns)	

To Learn More…

https://wgropp.cs.illinois.edu/courses/cs598-s15/lectures/lecture34.pdf

https://wgropp.cs.illinois.edu/courses/cs598-s15/lectures/lecture35.pdf

https://wgropp.cs.illinois.edu/courses/cs598-s15/examples/ga_mpi_ddt_rma.c