

MPI3 Hybrid

Proposal Description

PARALLEL@ILLINOIS

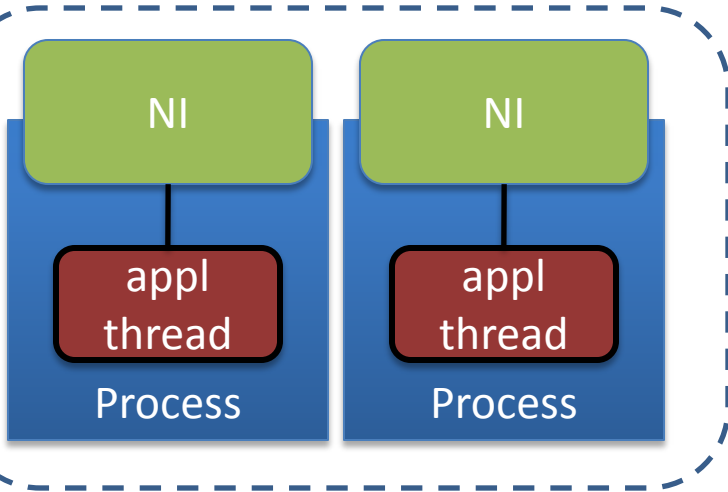
www.parallel.illinois.edu

Goals

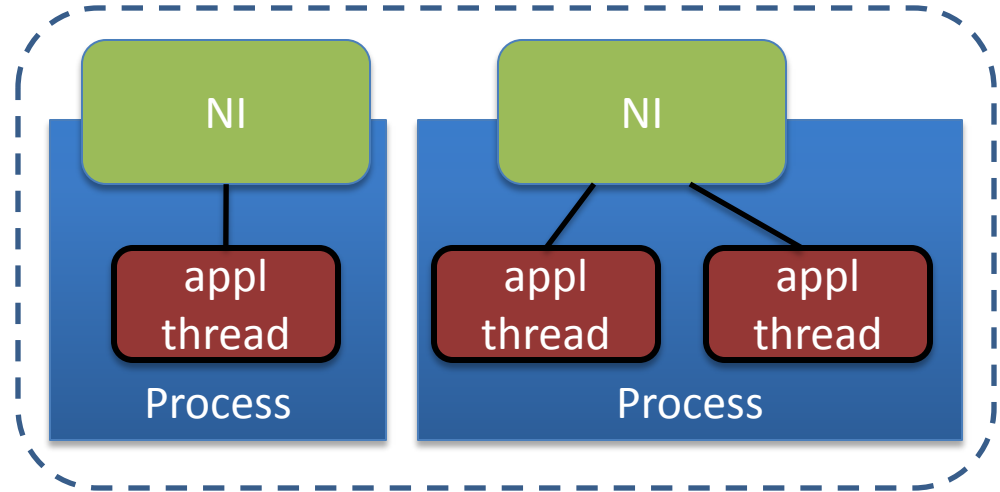
- Allowing more than one “MPI process” per node, without requiring multiple address spaces
- Having a clean binding to OpenMP and other thread-based parallel programming models
- Having a clean binding to UPC, CAF and similar PGAS languages

Current Design

Node



Node



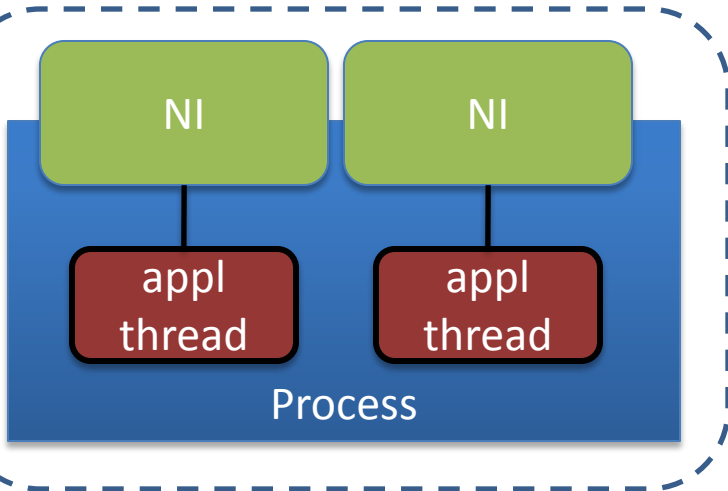
- Single-threaded

- Multi-threaded
 - Requires reentrant MPI library (hard) or mutual exclusion

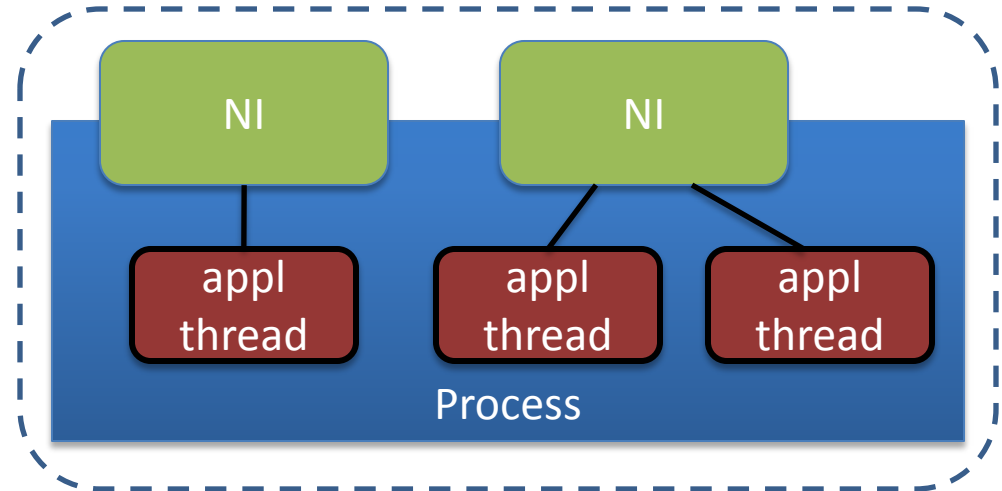
Network Interfaces (NI): physical or virtual, disjoint interfaces

Proposed Design

Node



Node



- Single-threaded

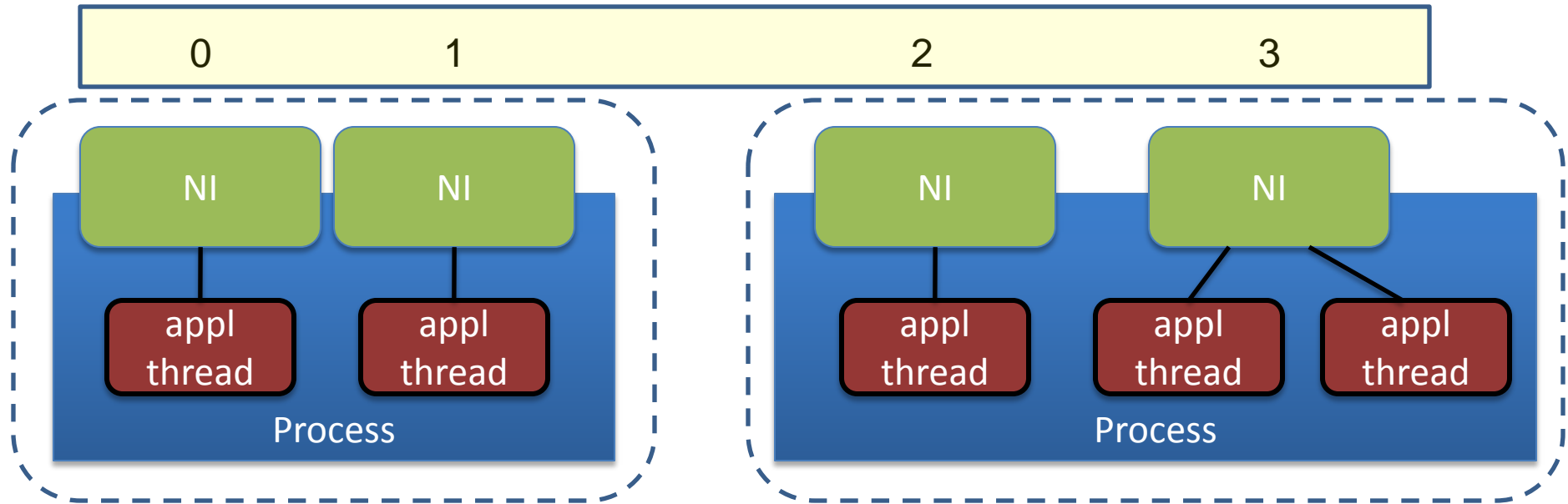
- Each NI (or virtual NI) is accessed by only one thread

- Multi-threaded

- Some NIs are accessed by multiple threads

Same as before, except multiple NIs are mapped in same address space

MPI_COMM_EWORLD



- Each *MPI Endpoint* has unique rank in MPI_COMM_EWORLD
 - rank in derived communicators computed using MPI rules
- MPI code executed by thread(s) *attached* to endpoint
 - Including collectives
 - thread is attached to at most one endpoint

Binding Time

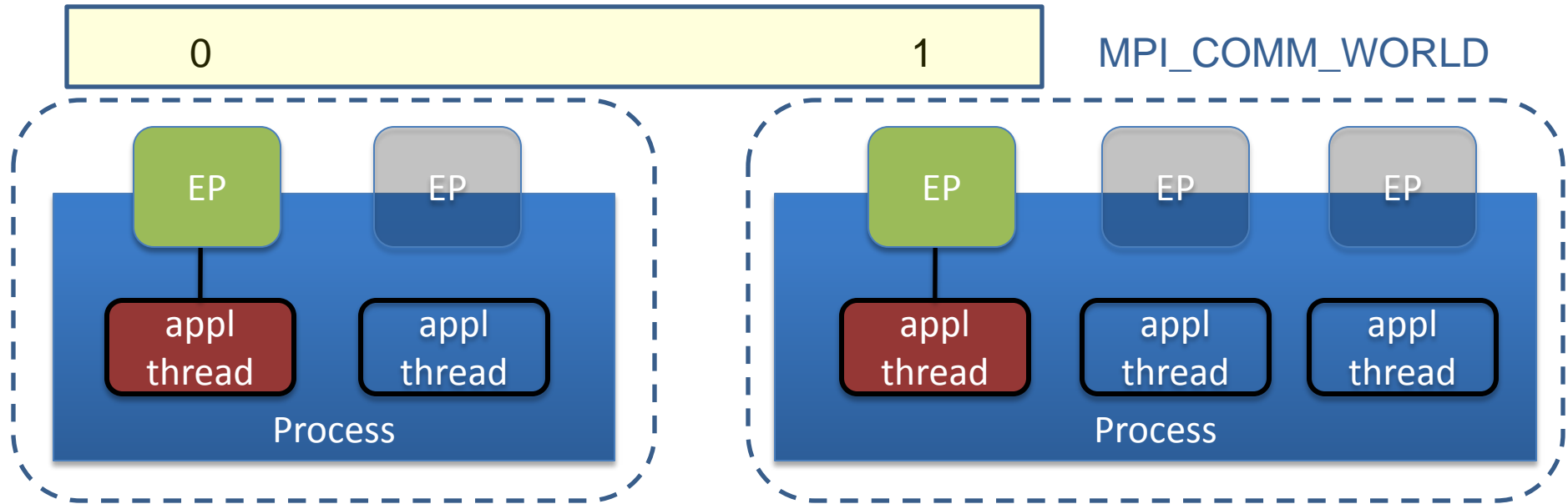
- Binding of MPI Endpoints to processes
 - done at MPI initialization time, or before
- Attachment of threads to MPI endpoints
 - done during execution; can change but expected to rarely change
 - threads can be created after endpoints

Endpoint Creation (1)

```
MPI_ENDPOINT_INIT(required, provided)
  IN required desired level of thread support (integer)
  OUT provided provided level of thread support (integer)
```

- `MPI_THREAD_FUNNELED`
 - each endpoint will have only one attached thread
- `MPI_THREAD_SERIALIZED`
 - multiple threads can attach to same endpoint, but cannot invoke MPI concurrently
- `MPI_THREAD_MULTIPLE`
 - multiple threads can attach to same endpoint and can invoke MPI concurrently
- Executed by one thread on each process; initializes `MPI_COMM_WORLD`

After MPI_ENDPOINT_INIT



- Attribute MPI_ENDPOINTS (on MPI_COMM_WORLD) indicates maximum number of endpoints at each process.
 - Can be different at each process (can depend on mpiexec arguments)

Endpoint Creation (2)

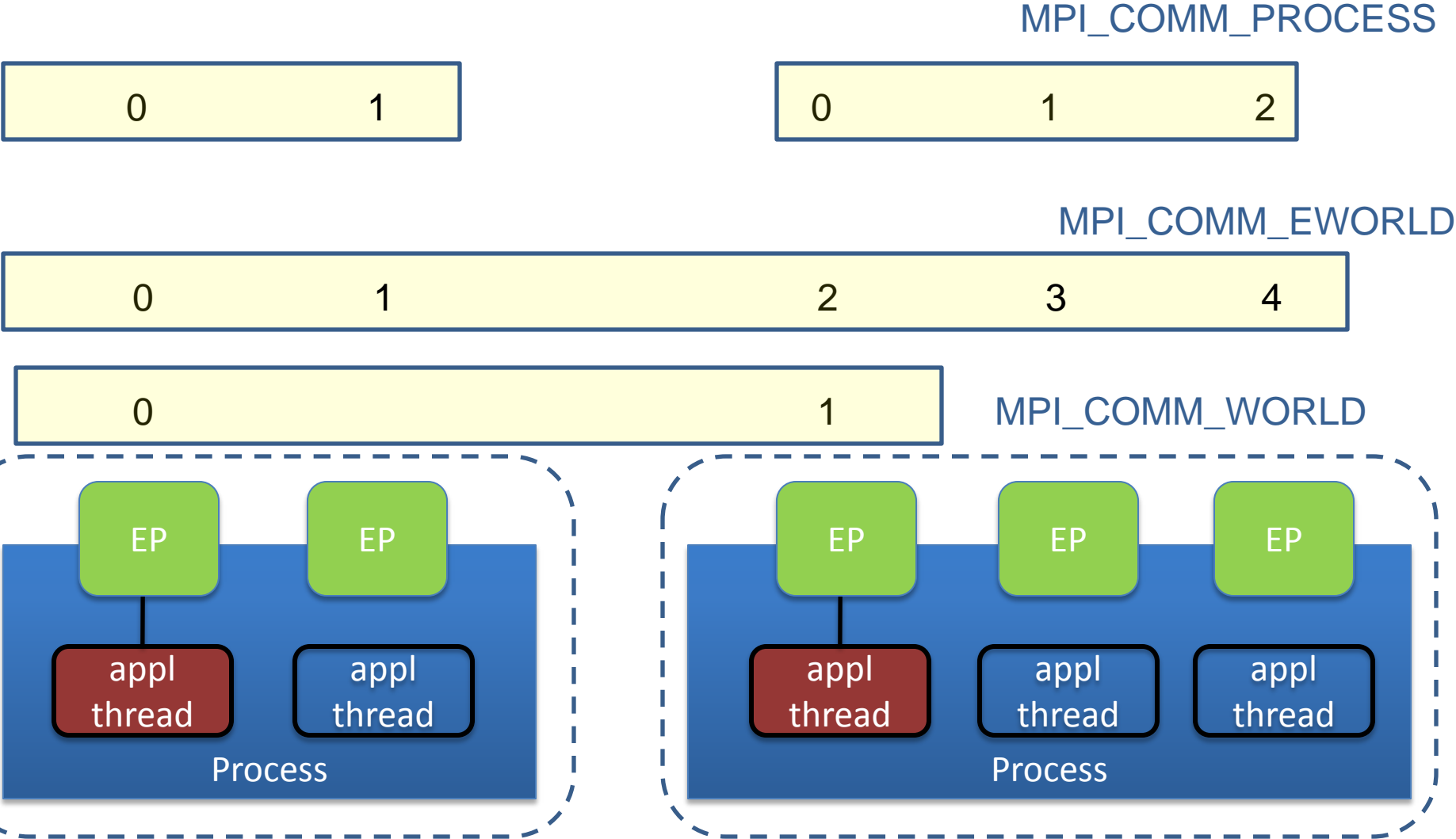
```
MPI_ENDPOINT_CREATE (num_endpoints,  
array_of_endpoints)
```

IN num_endpoints number of endpoints (integer)

OUT array_of_endpoints array of endpoint
handles (array of handles)

- Executed by one thread on each process;
initializes MPI_COMM_WORLD

After MPI_ENDPOINT_CREATE



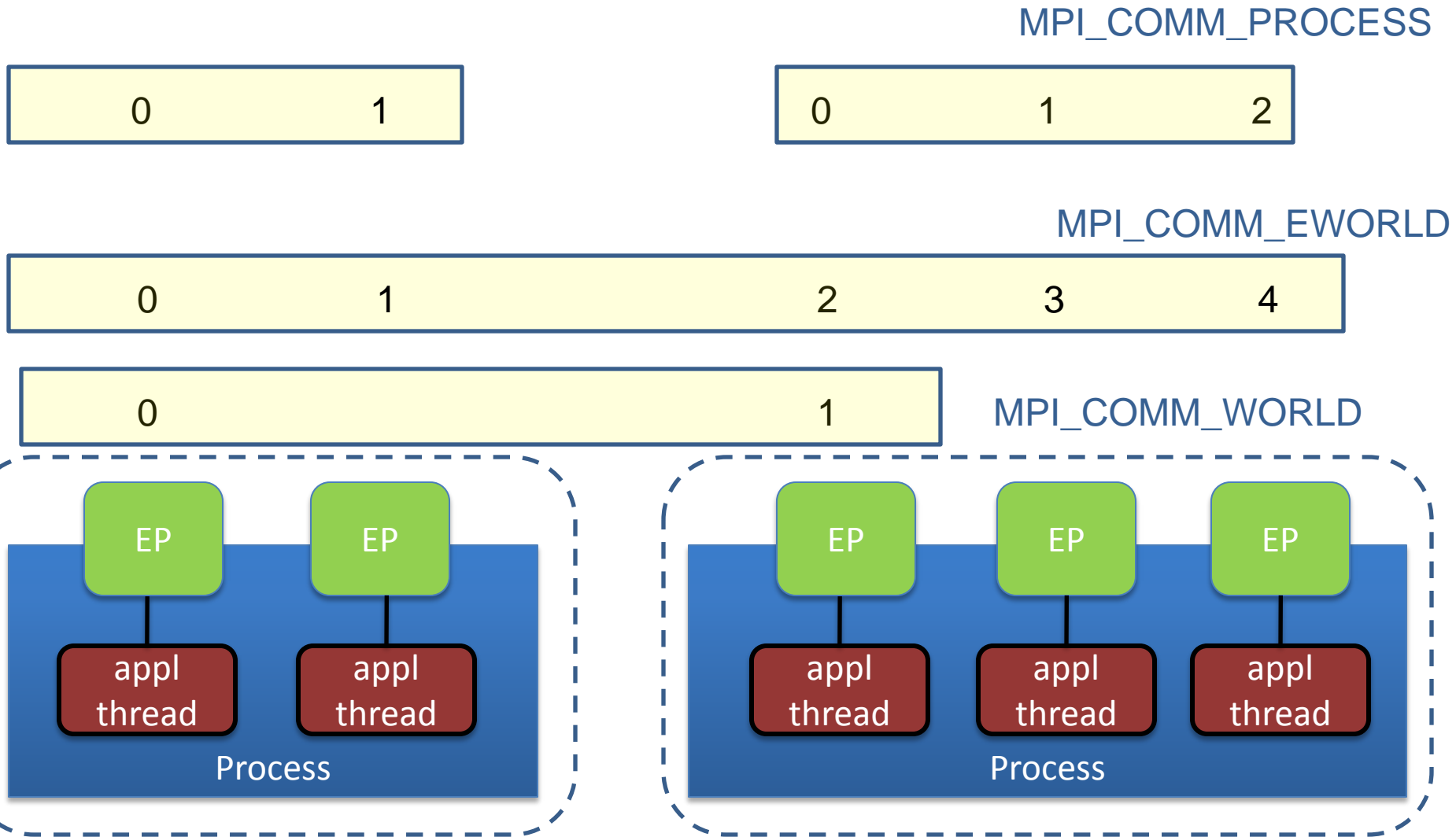
Thread Registration

```
MPI_THREAD_ATTACH (endpoint)  
  IN endpoints endpoint handle (handle)
```

```
MPI_THREAD_DETACH (endpoint)  
  IN endpoints endpoint handle (handle)
```

- Executed by each thread

After All Threads Attached



Remarks

- MPI calls invoked by thread pertain to endpoint the thread is attached to
 - E.g., sender rank is rank of endpoint
- MPI call may block if participation of endpoint is needed and no thread is attached to endpoint
 - E.g., collective or send to “orphan endpoint”
- Normally, endpoint will have attached thread after initialization

Missing Items

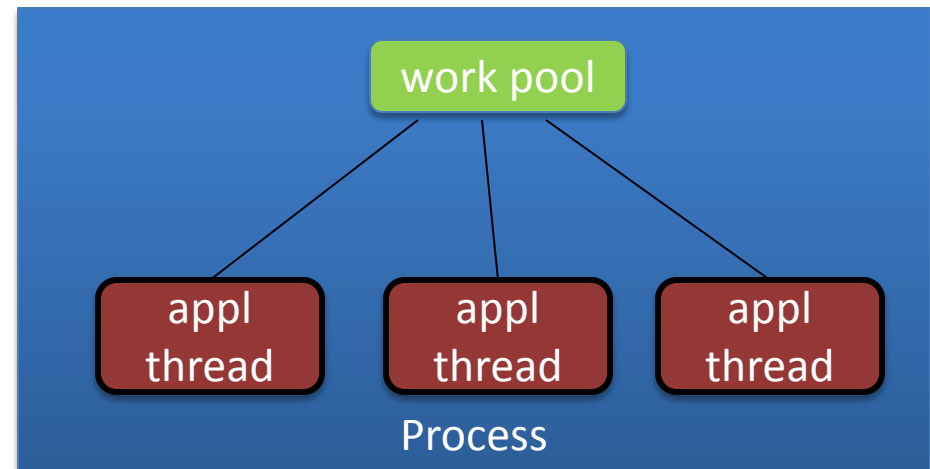
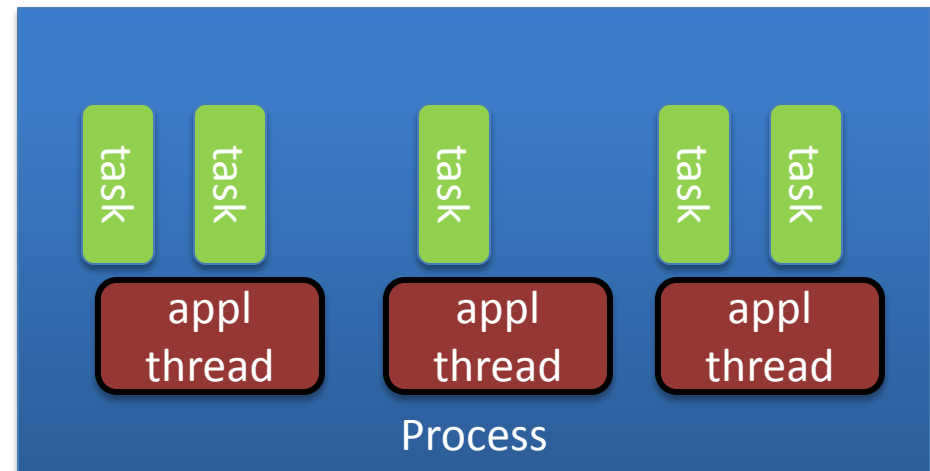
- MPI_FINALIZE()
- mpiexec
- Dynamic processes
- One-sided
- I/O
- Error codes
- ...

MPI + Shared Memory Language/Library

- Goals
 - Allow more than one MPI endpoint per shared memory program
 - Simple use (but not for naïve users!)
 - Simple implementation

Shared Memory Programming Models

- Number of threads can vary
- Tasks can be migrated from one thread to another
- Work can be shared dynamically



General Binding

- MPI calls can be executed anywhere in the shared memory program, with “natural semantics”
 - blocking MPI call blocks task/iterate
 - nonblocking call can be completed in any place in the program execution that causally follows the initial nonblocking call
 - registrations are per task, and are inherited when control forks

```
#pragma omp for
for(i=0; i<N; i++)
{
    MPI_Irecv(...,req[i]);
}
MPI_Waitall(N,req,stat);
```

Issues

- Blocking MPI call blocks thread; this may block work sharing
- OpenMP/TBB/... scheduler does not know about MPI; communication performance may be affected
- OpenMP (TBB, etc.) runtime needs to migrate implicit MPI state (e.g., identity of associated MPI endpoint) when task migrates or forks
- Need MPI_THREAD_MULTIPLE
- **Recommend to specify but not to require**
 - Detailed specification missing in current document

Restricted Binding

- MPI endpoint is associated with thread
- User has to make sure that MPI calls execute on “right” thread

OpenMP Rules

- MPI_ENDPOINT_CREATE is invoked in the initial, sequential part of the program.
- Each endpoint is attached by one OpenMP thread
- Only OpenMP threads attached to an endpoint invoke MPI.
- Each handle should be used in MPI calls by the same thread through program execution.
 - Handle is `threadprivate`
 - Handle is `shared` and is not reused in different parallel regions unless they are consecutive, neither is nested inside another parallel region, both regions use the same number of threads, and *dyn-var = false*
 - An MPI handle should not be used within a work sharing construct.
 - An MPI handle should not be used within an explicit task.

Example (1)

```
... // omit declarations
MPI_Endpoint_Init(argc, argv, MPI_THREAD_SINGLE,
    &provided);
MPI_Comm_get_attr(MPI_COMM_WORLD, MPI_ENDPOINTS, &pnum,
    &flag);
#pragma omp parallel private(myid)
{
    #pragma omp master
    {
        /* find number of threads in current team */
        Nthreads = omp_get_num_threads();
        if (Nthreads != *pnum) abort();
        /* create endpoints */
        MPI_Endpoint_create(Nthreads, *endpoints);
    }
}
```

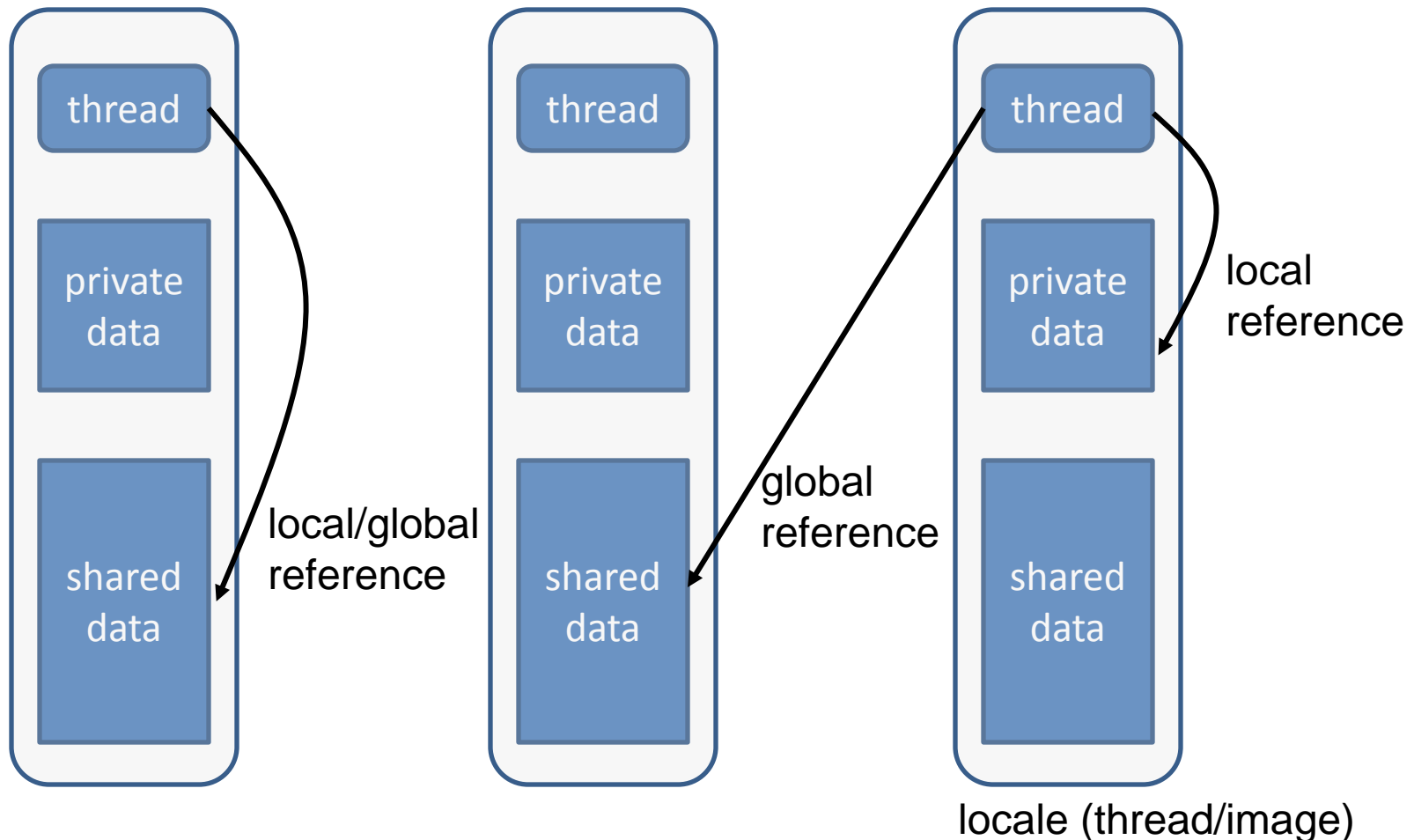
Example (2)

```
/* associate each thread with an endpoint */
myid = omp_get_thread_num();
MPI_Endpoint_register(myid, *endpoints);
/* MPI communication involving all threads */
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Comm_rank(MPI_COMM_PROCESS, &mythreadid);
if (myid != mythreadid) abort();
if (myrank > 0) MPI_Isend(buff, count, MPI_INT,
    myrank-1, 3, MPI_COMM_WORLD, &req[mythreadid]);
if (myrank < size) MPI_Recv(buff1, count, MPI_INT,
    myrank+1, 3, MPI_COMM_WORLD);
MPI_Wait(&req[mythreadid], &status[mythreadid]); }
...
```

TBB et al

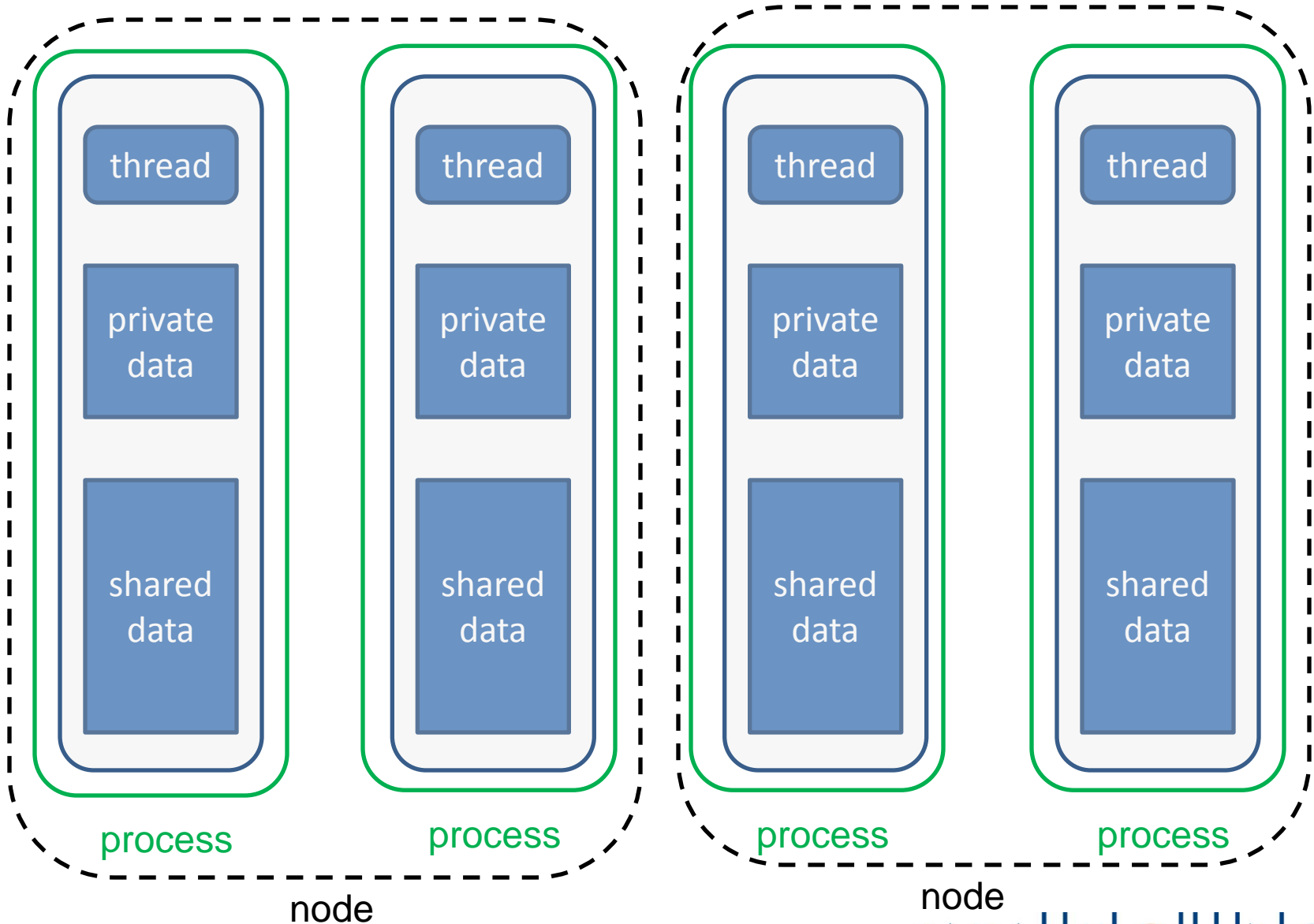
- Need TBB to provide a construct that binds a task to a specific thread.
 - Similar construct may be needed for GPU tasks and other similar messes

PGAS (UPC, Fortran 2008)

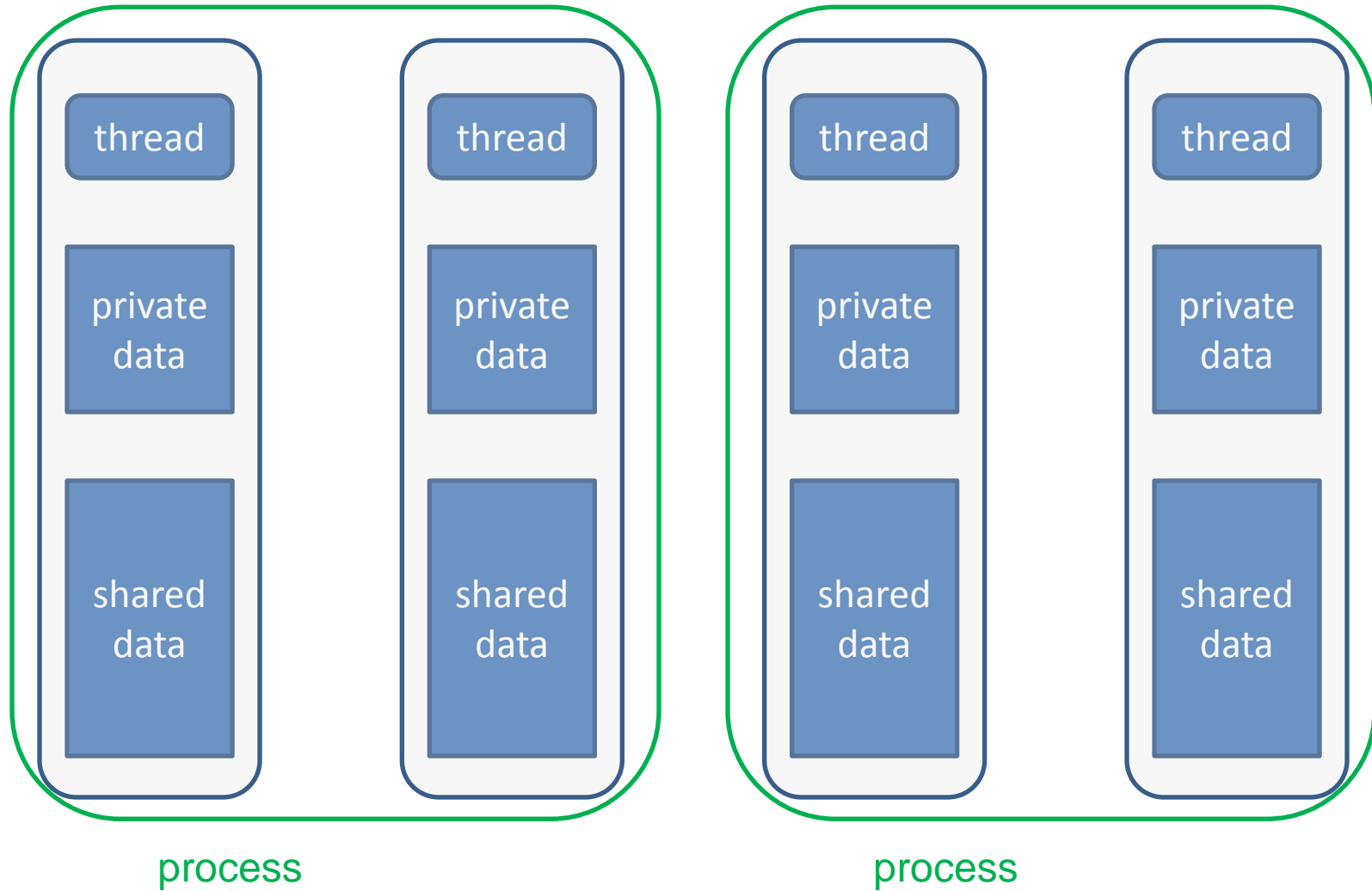


- One thread of control per locale (+ work sharing, in UPC)

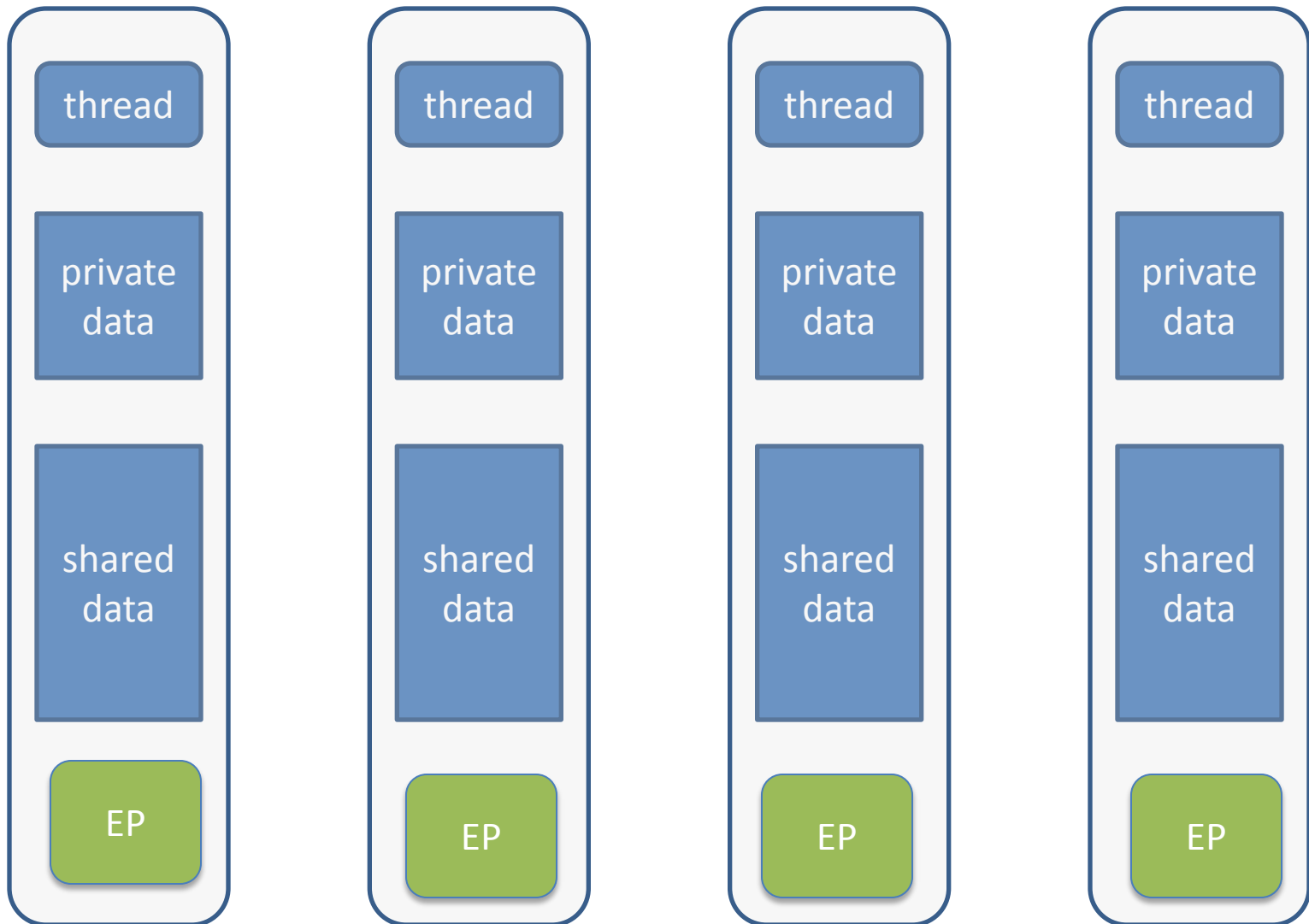
PGAS Implementations (1)



PGAS Implementations (2)



PGAS Invokes MPI



PGAS Invokes MPI (cont.)

- Each locale (thread/image) has MPI_COMM_EWORLD rank corresponding to its PGAS rank
- Each locale can invoke MPI (one endpoint per locale)
 - all arguments must be local
 - cannot be done within work sharing construct
- Model does not depend on chosen PGAS implementation
- Requires joint initialization of PGAS and of MPI
 - Both are initialized before user code starts running

MPI Invokes PGAS

Need “awareness” of PGAS language within MPI code

- Link and invoke UPC routine from (SPMD) C
 - Collective call over all MPI processes (endpoints) in communicator
- Create shared array from (SPMD) C
 - collective malloc; shared array descriptor
- Pass shared array as argument
- Query shared arrays

Problems mostly are on PGAS side; should we address them?