Argonne NATIONAL LABORATORY

Systems Software for Scalable Applications (or) Super Faster Stronger MPI (and friends) for Blue Waters Applications

William Gropp
University of Illinois, Urbana-Champaign

Pavan Balaji, Rajeev Thakur
Argonne National Laboratory

U.S. DEPARTMENT OF ENERGY

The Multi- and Many-core Era

- Increasing number of core counts on modern processors
  - Cray XE6 processors have 16 cores per node
  - BG/Q has 16 cores (64 hardware threads) per "node"
- Two important trends are driving systems software
  - Per-core resources are not scaling at the same rate as the number of cores
    - E.g., memory, TLB entries, network endpoints
    - Hybrid programming models (such as MPI+threads) are becoming common
  - "System cores" are becoming an accepted fact of large systems
    - BG/Q already provides an additional 17th core for system tasks
    - Cray does not, but one could envision a similar model in the future

Argonne NATIONAL LABORATORY

MPI+Threads Hybrid Programming

U.S. DEPARTMENT OF ENERGY

Thread Safety for MPI+Threads Programming

- MPI_THREAD_SINGLE
  - MPI only, no threads
- MPI_THREAD_FUNNELED
  - Outside OpenMP parallel region, or OpenMP master region
- MPI_THREAD_SERIALIZED
  - Outside OpenMP parallel region, or OpenMP single region, or critical region
- MPI_THREAD_MULTIPLE
  - Any thread is allowed to make MPI calls at any time

```
#pragma omp parallel for
for (i = 0; i < N; i++) {
      uu[i] = (u[i] + u[i - 1] +  u[i + 1])/5.0;
}

MPI_Function ( );
```

```
#pragma omp parallel
{
      /* user computation */
      #pragma omp single
      MPI_Function ();
}
```

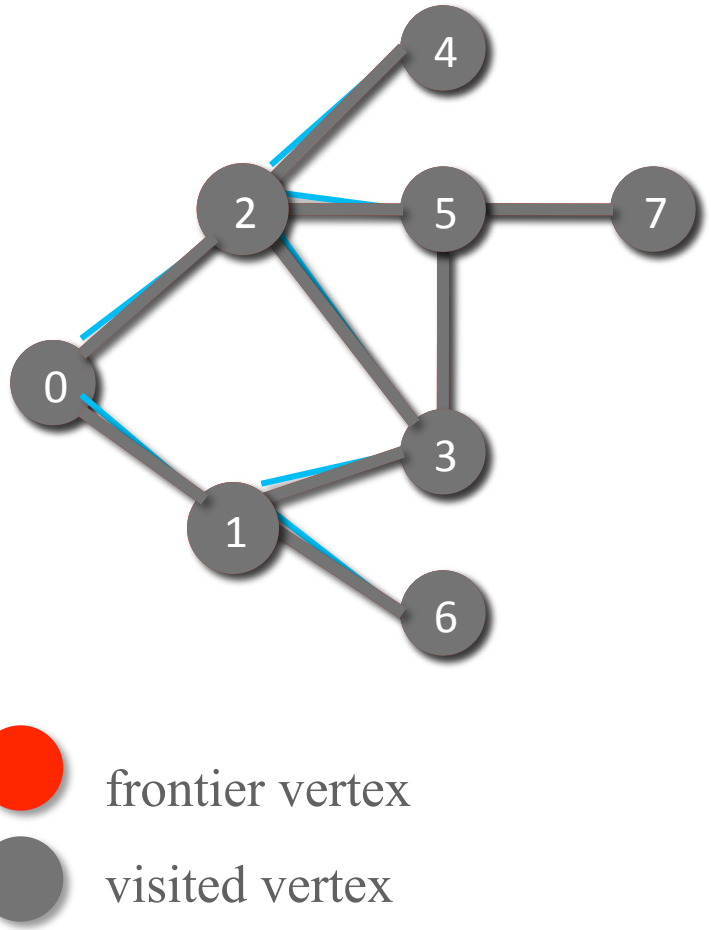
```
#pragma omp parallel
{
      /* user computation */
      #pragma omp critical
      MPI_Function ();
}
```

Increasing push towards THREAD_MULTIPLE

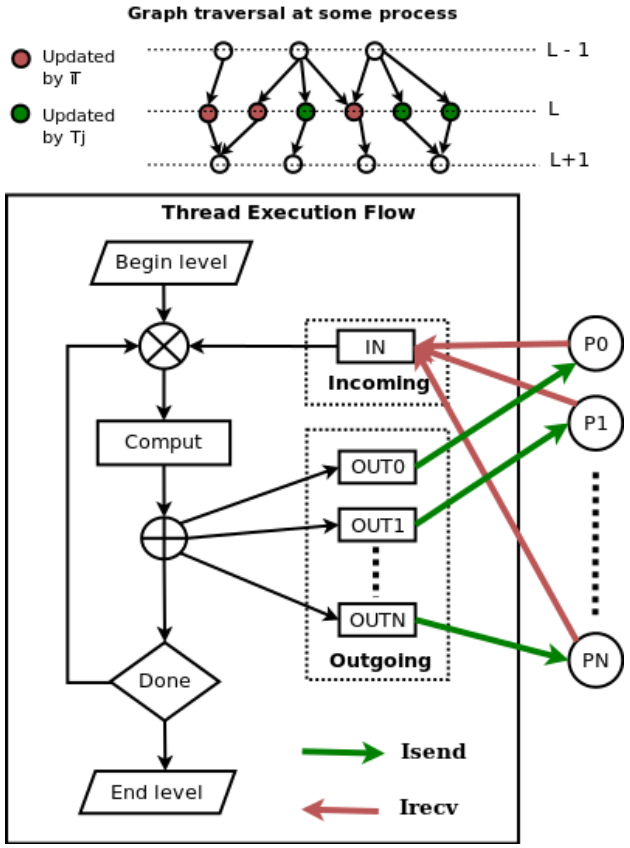
- Several applications are trying to take advantage of THREAD_MULTIPLE capabilities
- Many reasons:
  - Uniformity of computational breakout (no special "MPI thread")
  - Better load balancing and progress (the first available thread can send/receive data)
  - Better network and communication performance (multiple threads driving the network can improve performance substantially)
- We wanted to study what we should expect for such applications on Blue Waters
  - Used Graph500 as a case-study, primarily because of (1) it's irregularity and (2) it's communication intensive nature

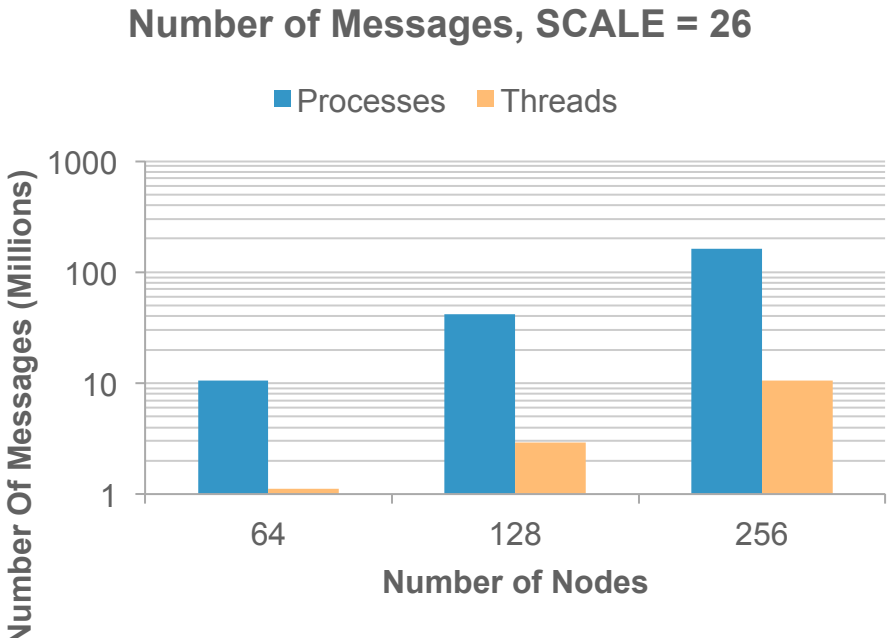
Breadth-First Search in Graph500

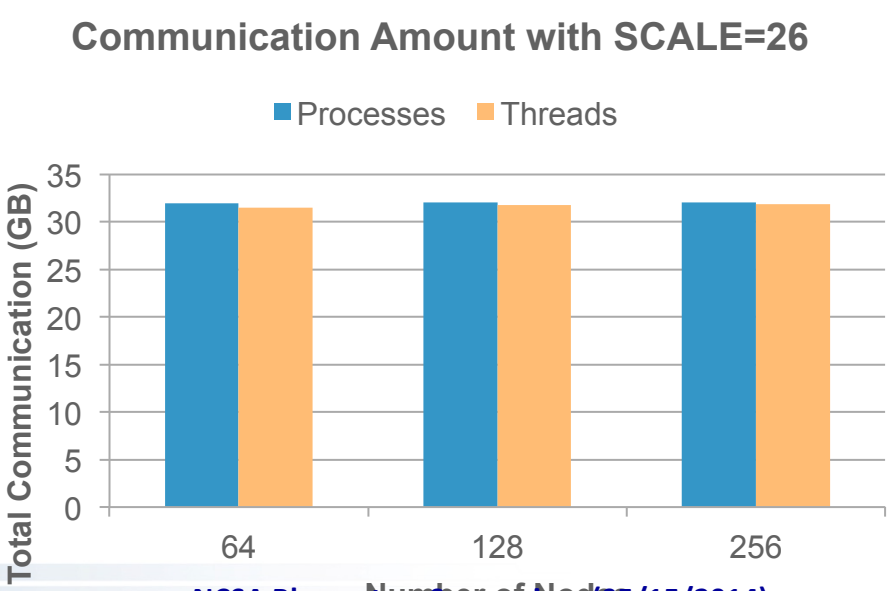
- BFS is a subroutine for many algorithms
  - Betweenness centrality
  - Maximum flows
  - Connected components
  - Spanning forests
- Characteristics of BFS
  - irregular
  - low-arithmetic
  - abundant parallelism



Multithreaded Graph500 Benchmark



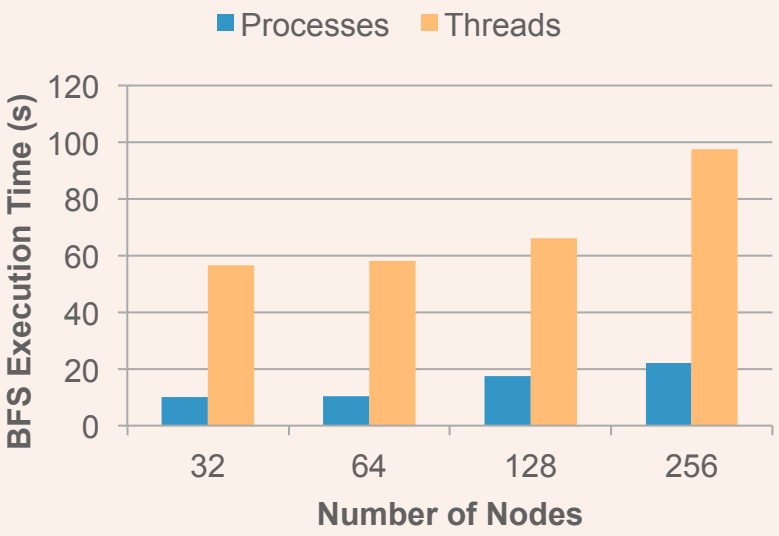


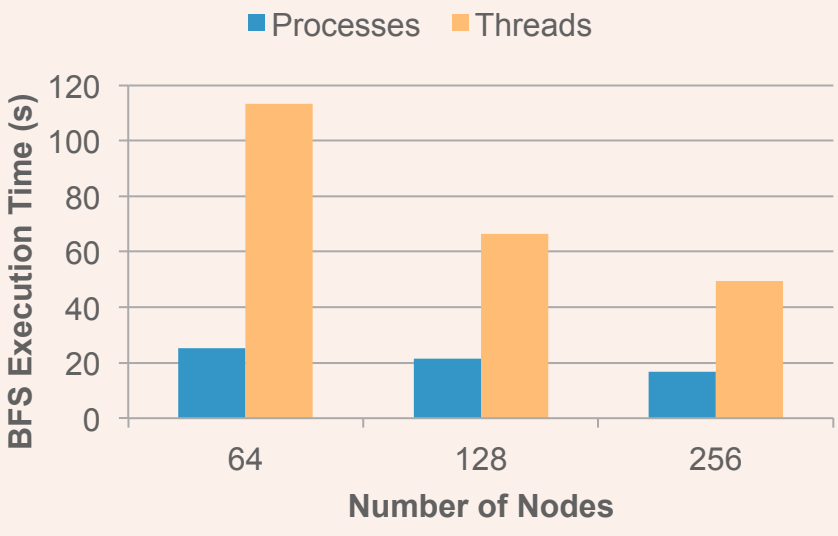


- Point-to-Point asynchronous communication
- Threads Implementation exhibits less communication

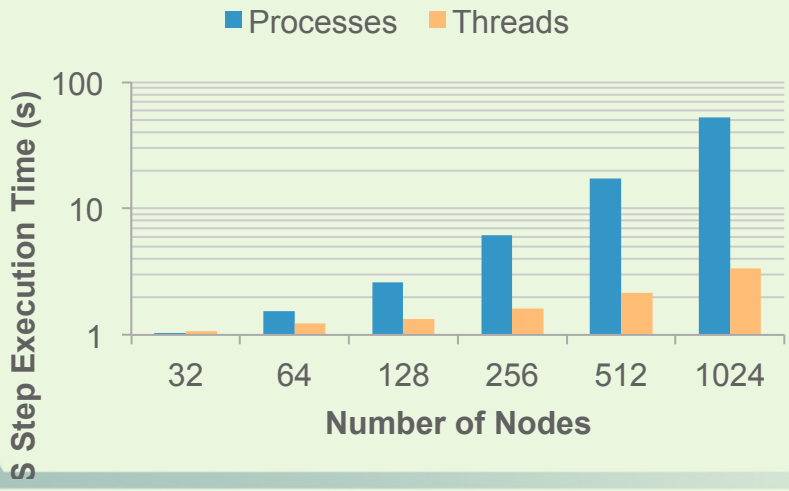
Graph500 Benchmark: Processes vs. Threads

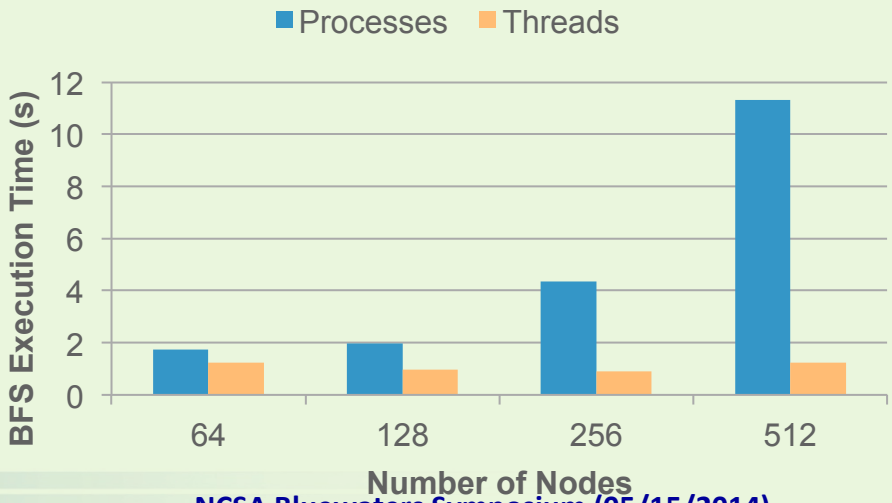
BlueWaters





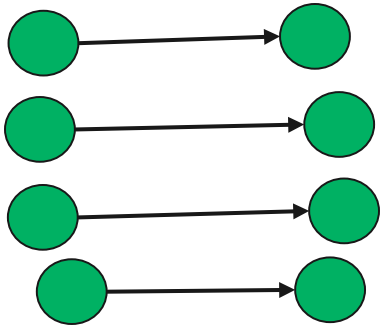
Blue Gene/Q



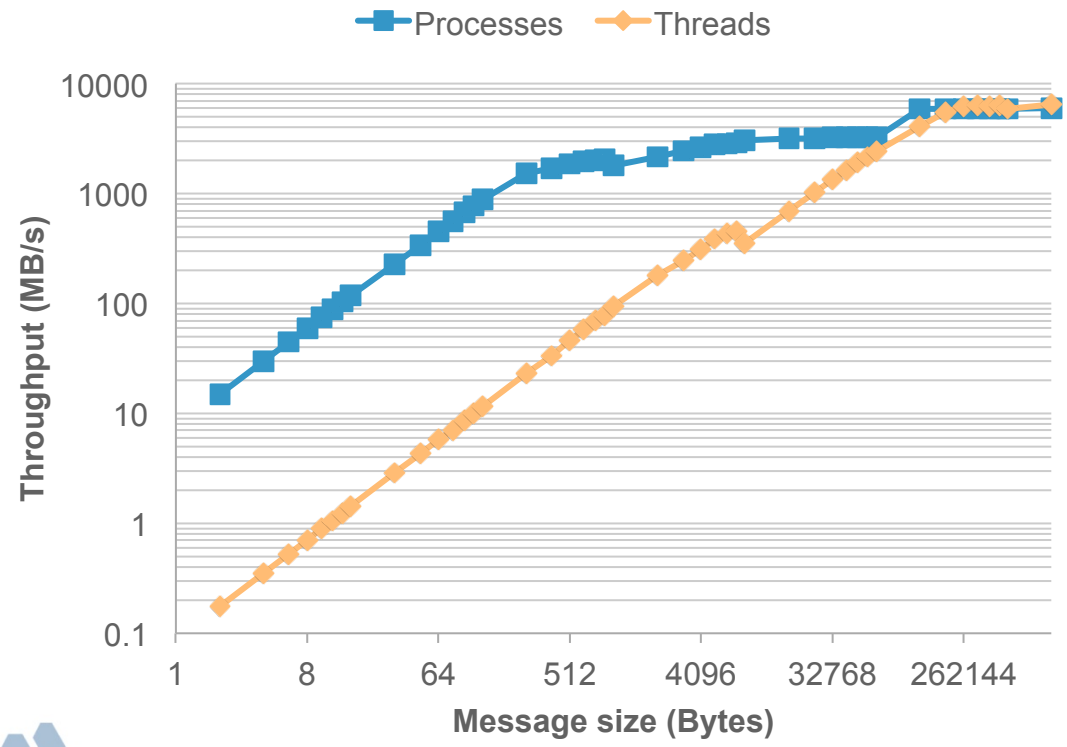


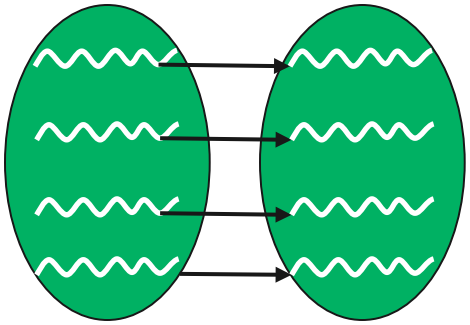
Processes vs. Threads: Throughput Benchmark on BlueWaters

- Throughput between two nodes
- Send data via processes or threads



Processes



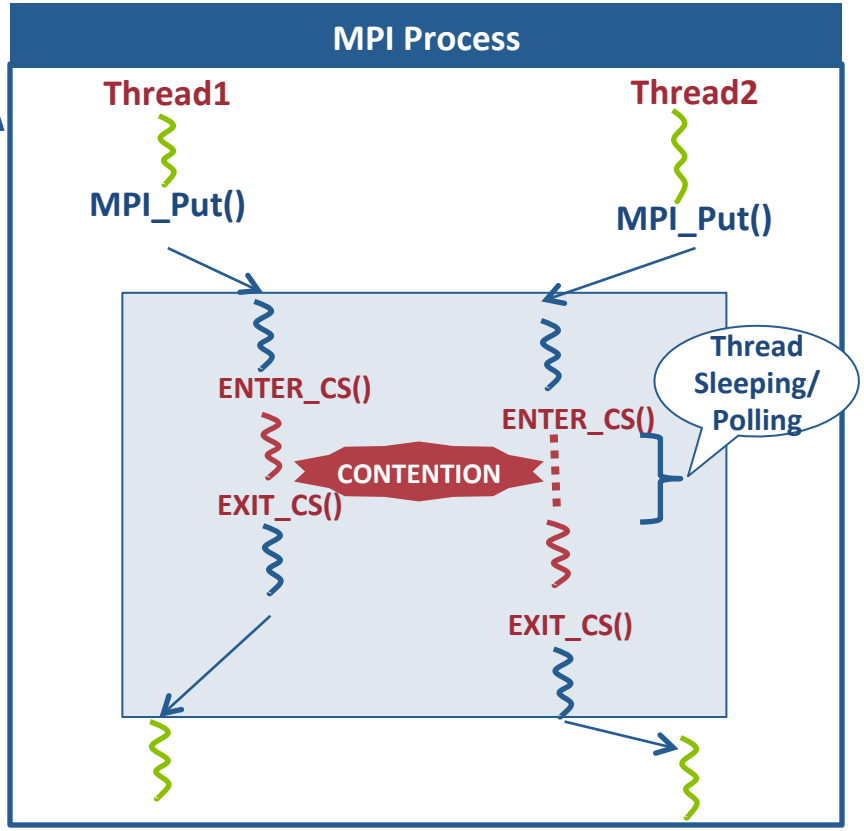


Threads

Contention in a Multithreaded MPI Model

```
MPI_Init_thread(…,MPI_THREAD_MULTIPLE,…);
.
.
#pragma omp parallel
{
    /* Do Work */
    MPI_Put();
    /* Do Work */
}
```

- Multithreaded MPI
  - Threads can make MPI calls concurrently
  - Thread-safety is necessary



Thread-safety can be ensured by:

- **Critical Sections** (Locks)

→ **Possible Contention !**

- Using **Lock-Free** algorithms

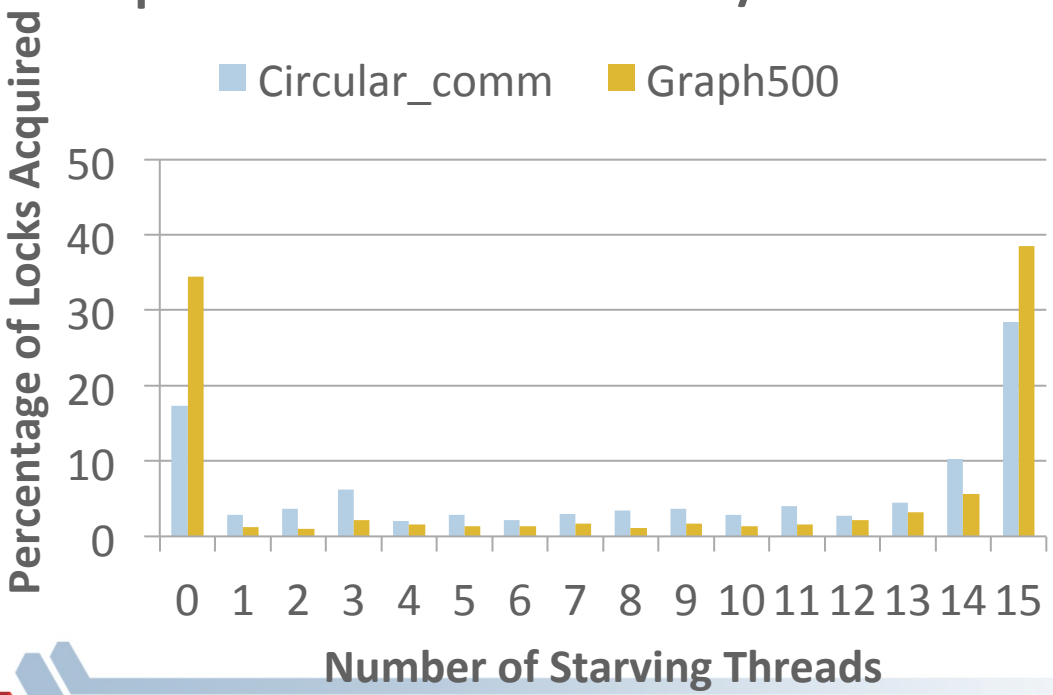
→ **Non trivial !**

→ **Still does memory barriers**

Hidden Evil: Lock Monopolization (Starvation)

- Implementing critical sections with spin-locks or mutexes
- Watch out: **no fairness** guarantee!

Starvation measurement with 16 processes and 16 threads/nodes



Starvation Detection Algorithm

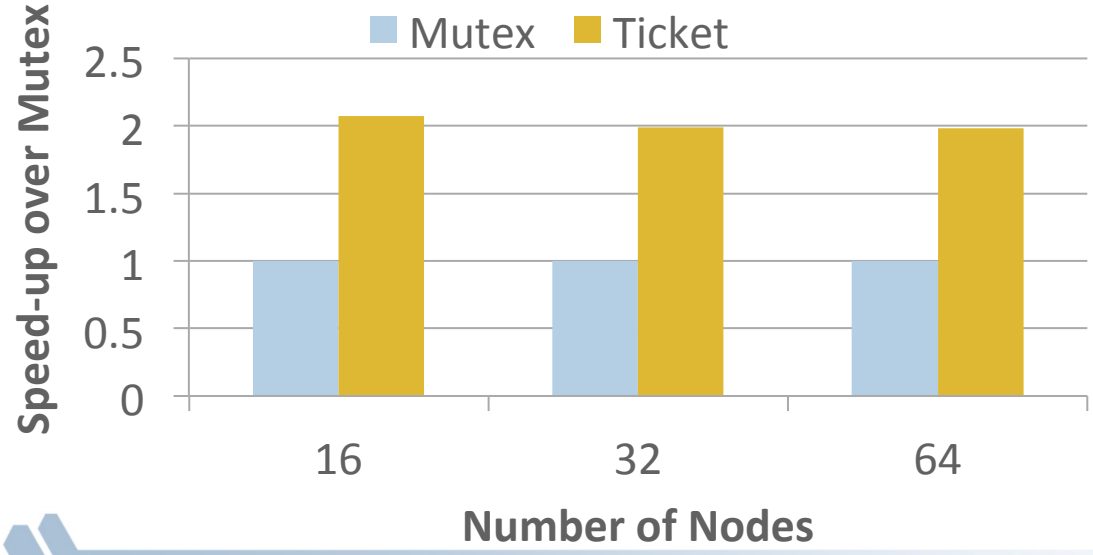
```
int waiting_threads = 0;
int last_holder;

acquire_lock(L)
{
  bool lock_acquired = false;
  try_lock(L, lock_acquired)
  if ((lock_acquired)                    &&
     (my_thread_id == last_holder) &&
     (waiting_threads > 0))
    STARVATION_CASE;
  else if (!lock_acquired)
  {
    atomic_incr(waiting_threads);
     lock(L);
    atomic_decr(waiting_threads);
  }
  last_holder = my_thread_id;
  return;
}
```

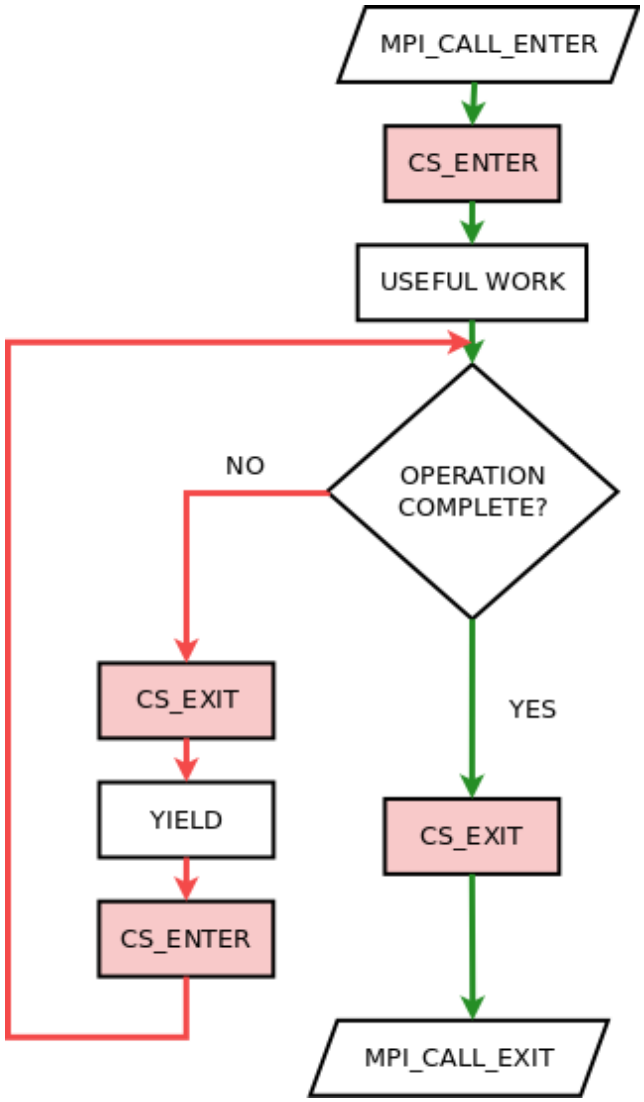
How to fix Lock Monopolization?

- Use locks that ensure **fairness**
- Example: Ticket Spin-Lock
- Basics:
  - Get my ticket and Wait my turn
  - Ensures **FIFO** acquisition

2D Stencil, Hallo=2MB/direction, Message size=1KB, 16Threads/Node

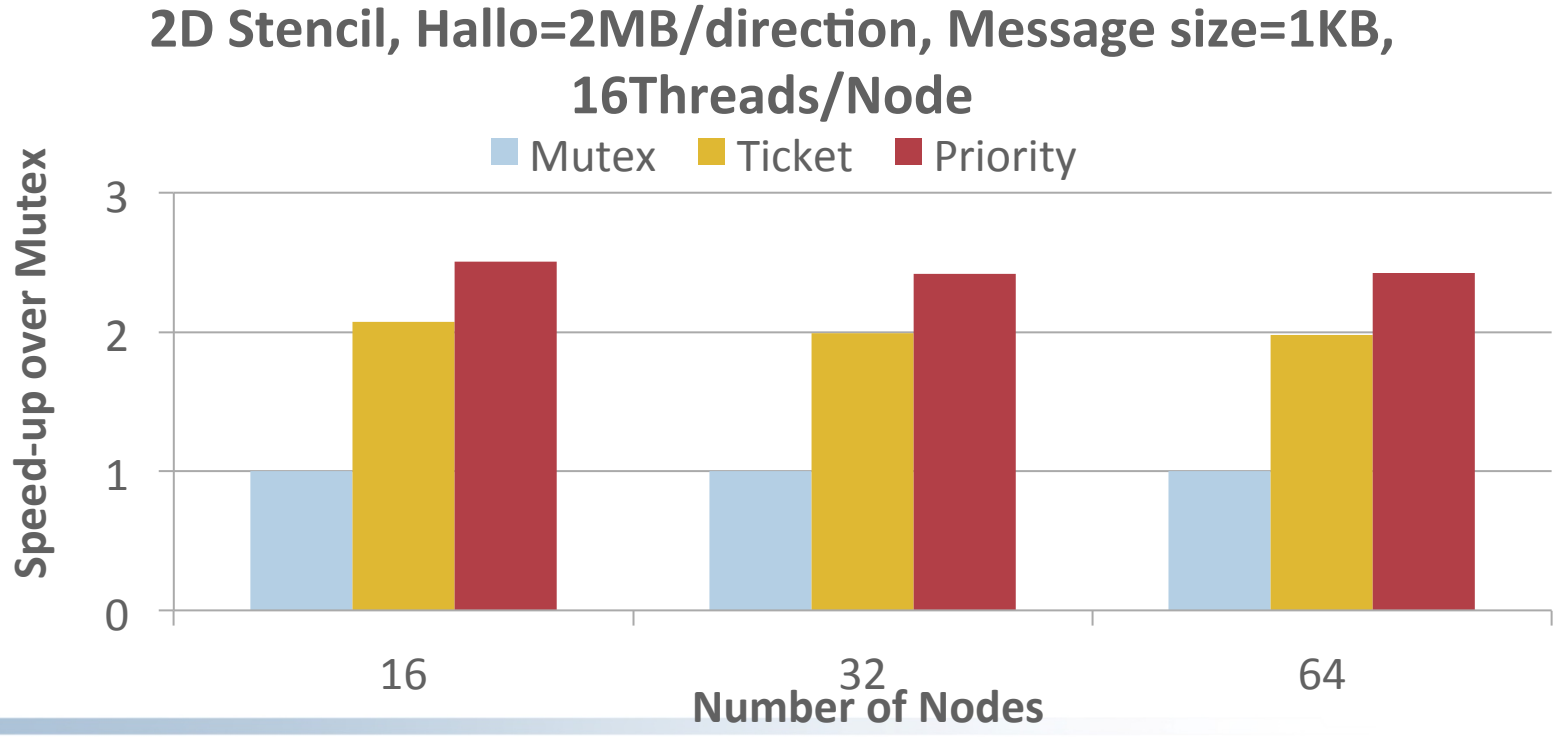


Simplified Execution flow of a Thread-safe MPI implementation with critical sections



Priority Locking Scheme

- 3 basic locks:
  - One for mutual exclusion in each priority level
  - Another for high priority threads to **block** lower ones
- Watch out: do not forget **fairness** in the **same priority level**
  - Use **exclusively** FIFO locks (Ticket)



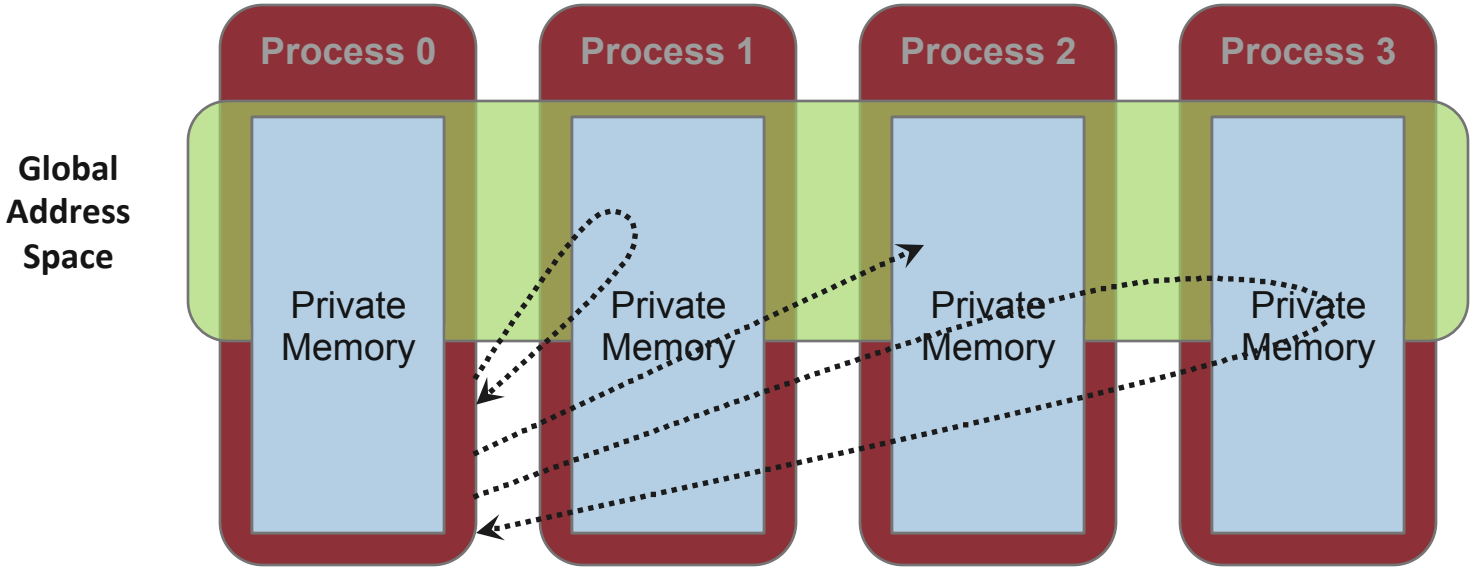
Argonne NATIONAL LABORATORY

MPI-3 One-sided Communication

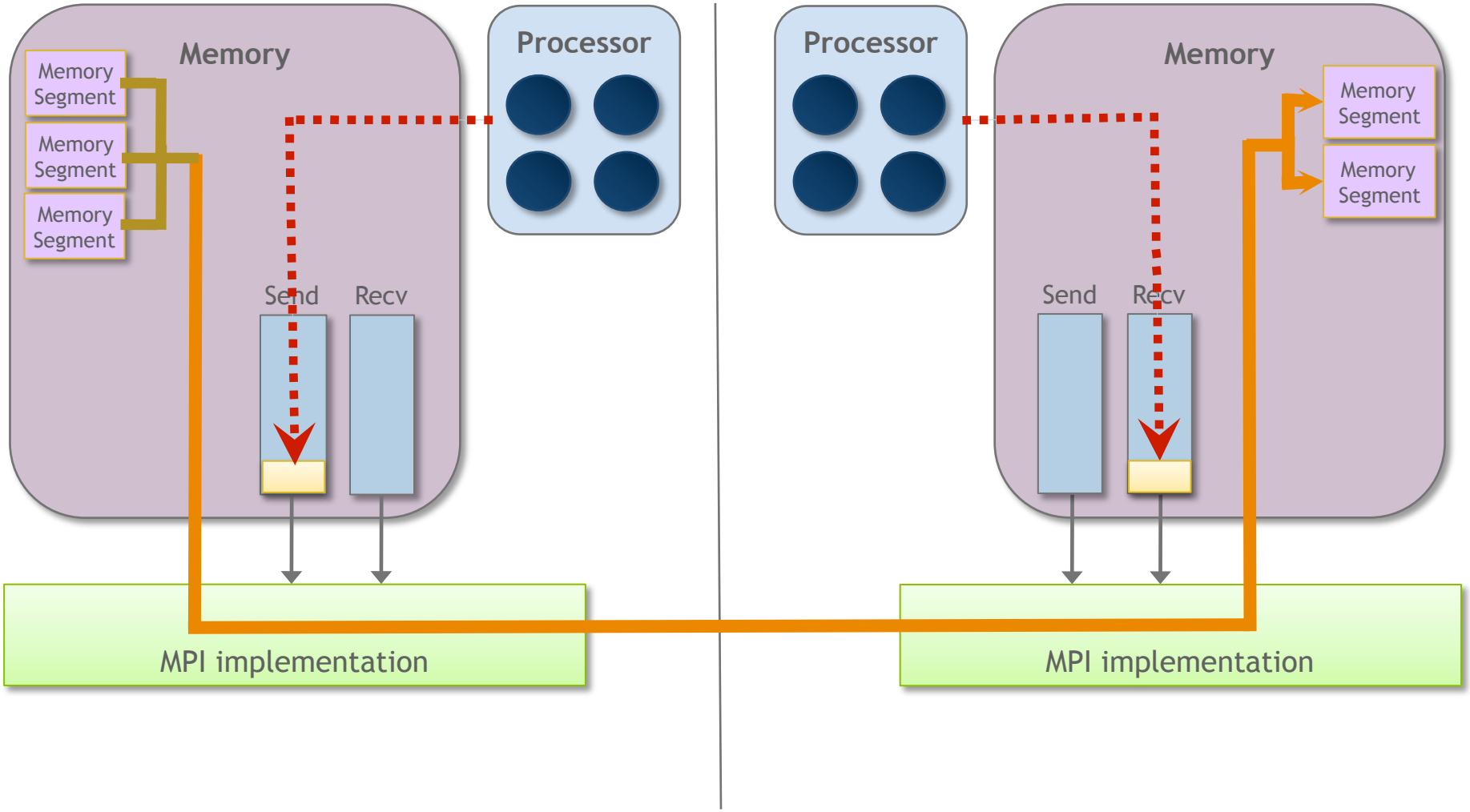
U.S. DEPARTMENT OF ENERGY

MPI-3 One-sided Communication

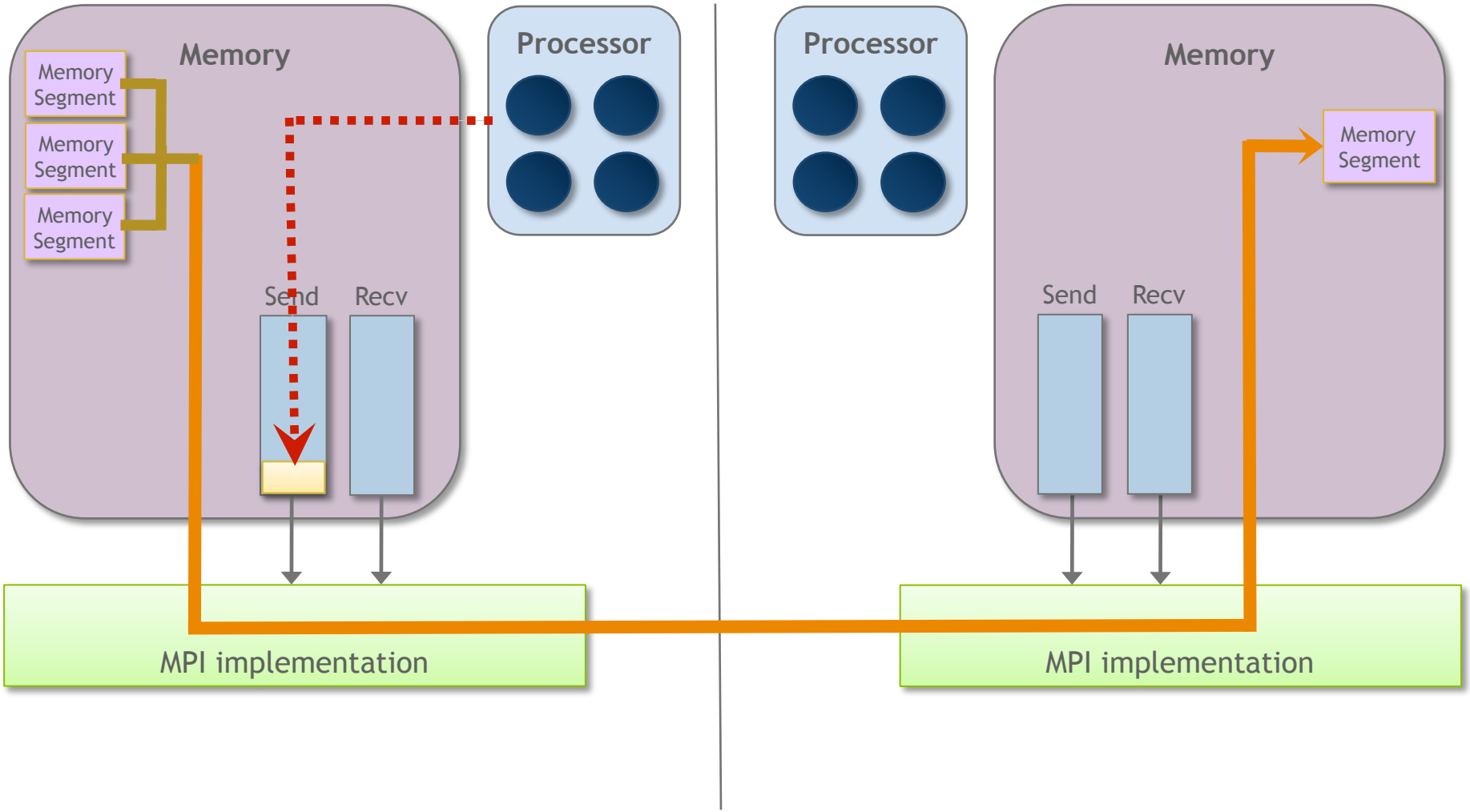
- The basic idea of one-sided communication models is to decouple data movement with process synchronization
  - Should be able move data without requiring that the remote process synchronize
  - Each process exposes a part of its memory to other processes
  - Other processes can directly read from or write to this memory



Two-sided Communication Example



One-sided Communication Example

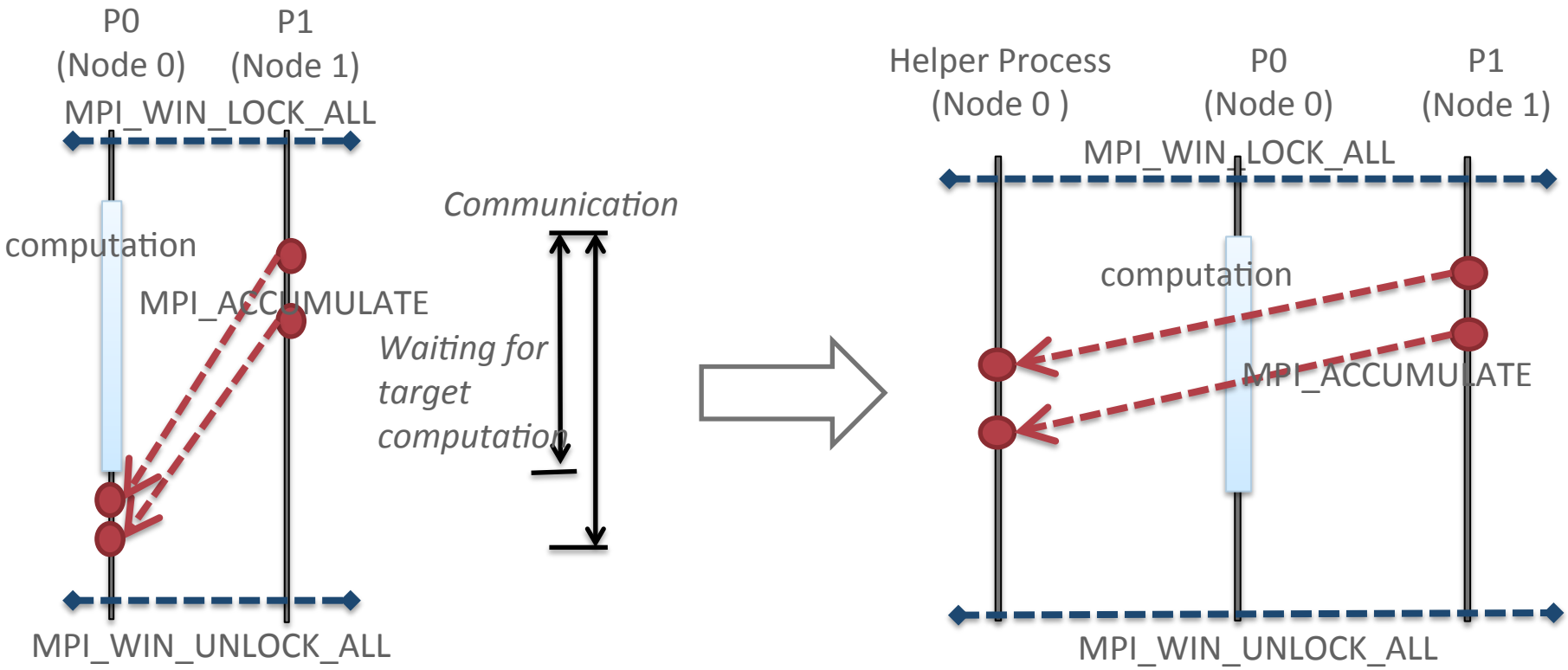


Asynchronous Communication Management on Cray XE6

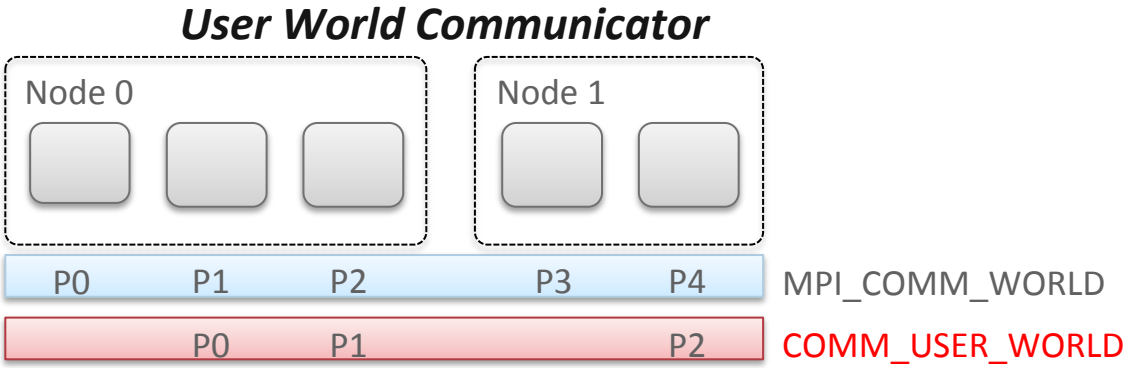
- One-sided communication operations are not always truly one-sided
  - Typically, hardware supported operations (such as contiguous PUT/GET) are done in hardware; everything else is done in software (e.g., 3D accumulates of double precision data)
  - On Blue Waters, most operations are done in software because of some issues in the layering structure
- Software implementation of one-sided operations means that the remote process has to make an MPI call to make progress

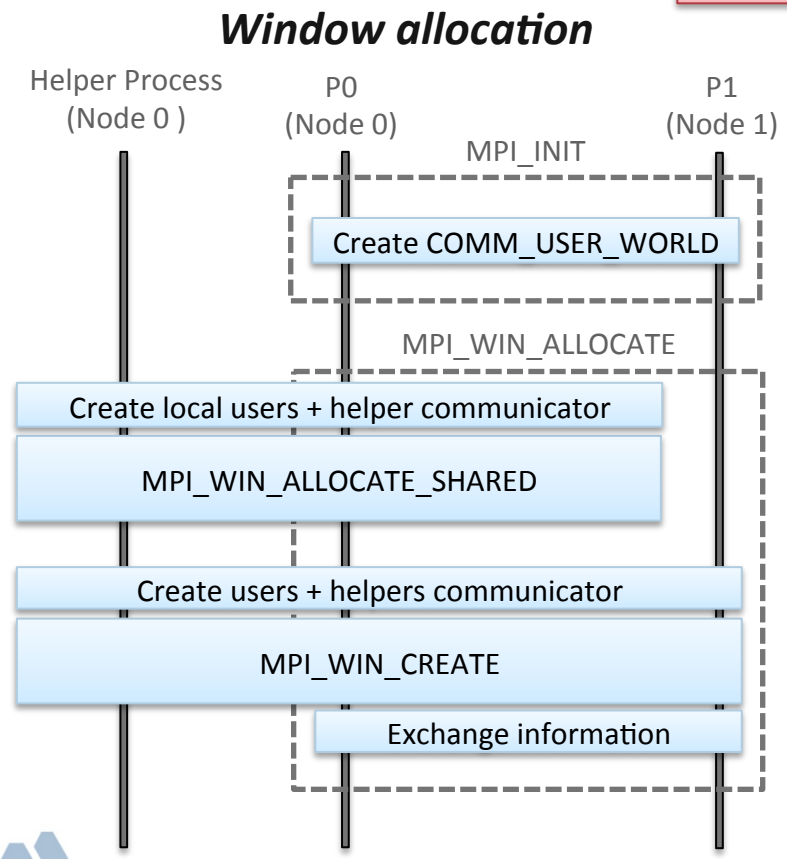
ASP

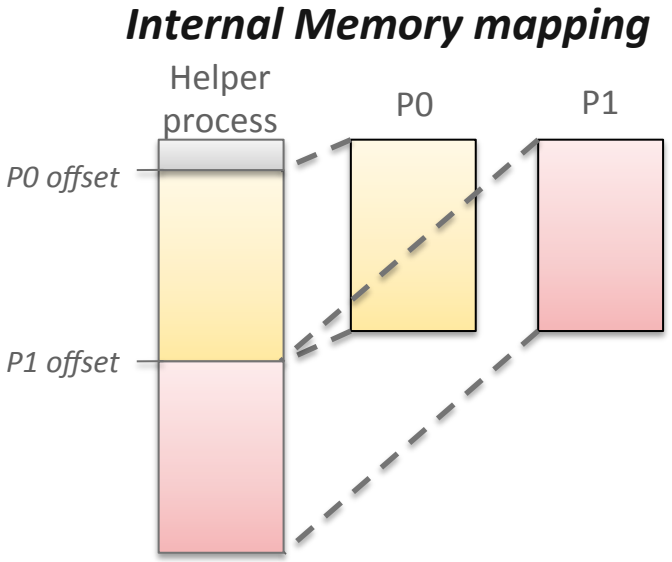
- Communication dedicated Helper processes handle incoming messages instead of original target processes



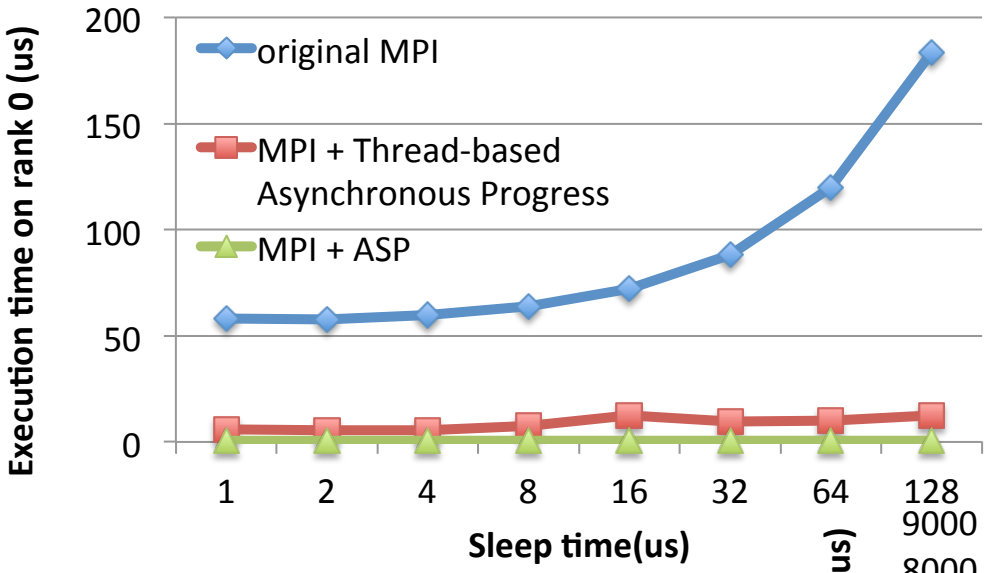
ASP design

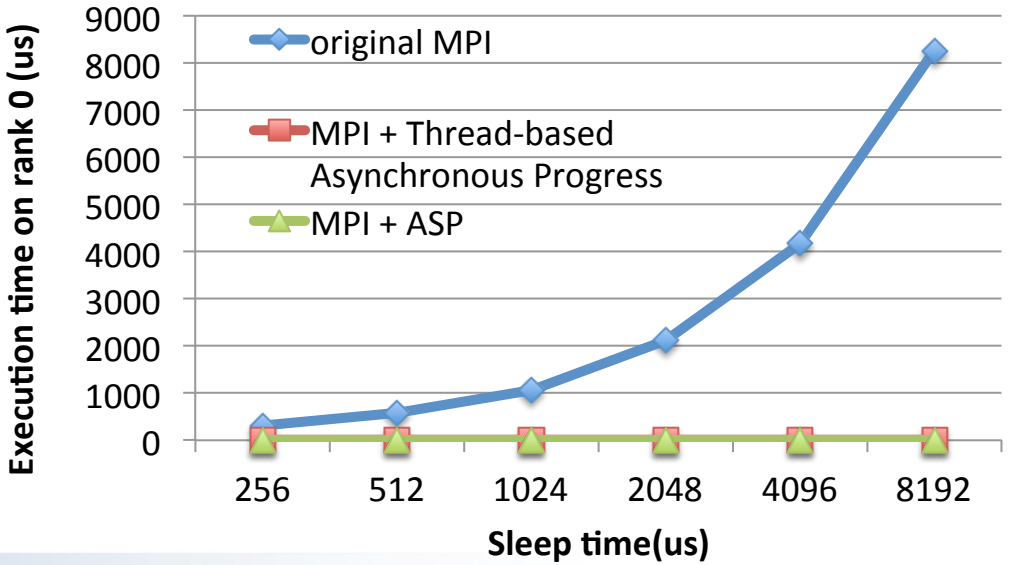

User World Communicator
Node 0
Node 1
P0
P1
P2
P3
P4
MPI_COMM_WORLD
P0
P1
P2
COMM_USER_WORLD


Window allocation
Helper Process (Node 0)
P0 (Node 0)
P1 (Node 1)
MPI_INIT
Create COMM_USER_WORLD
MPI_WIN_ALLOCATE
Create local users + helper communicator
MPI_WIN_ALLOCATE_SHARED
Create users + helpers communicator
MPI_WIN_CREATE
Exchange information

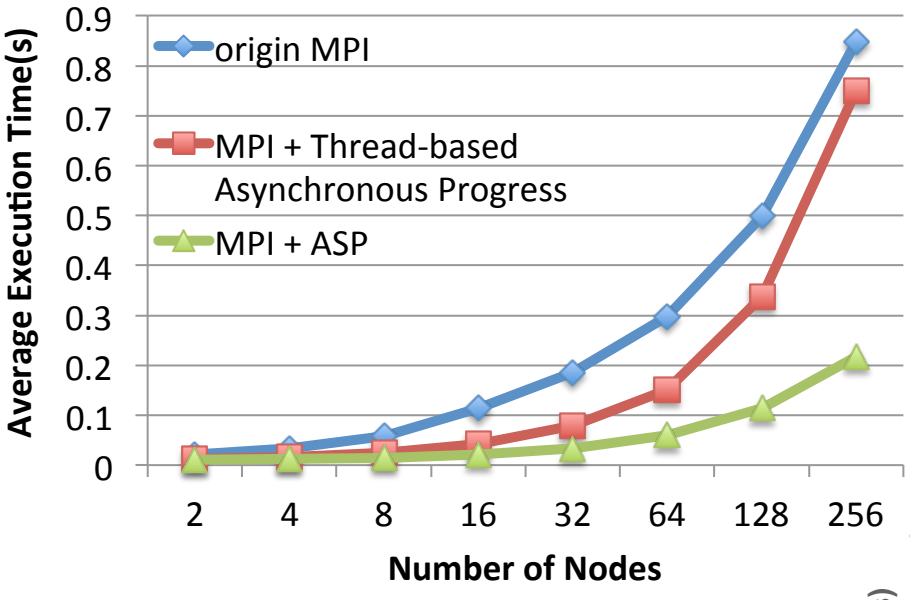

Internal Memory mapping
Helper process
P0
P1
P0 offset
P1 offset

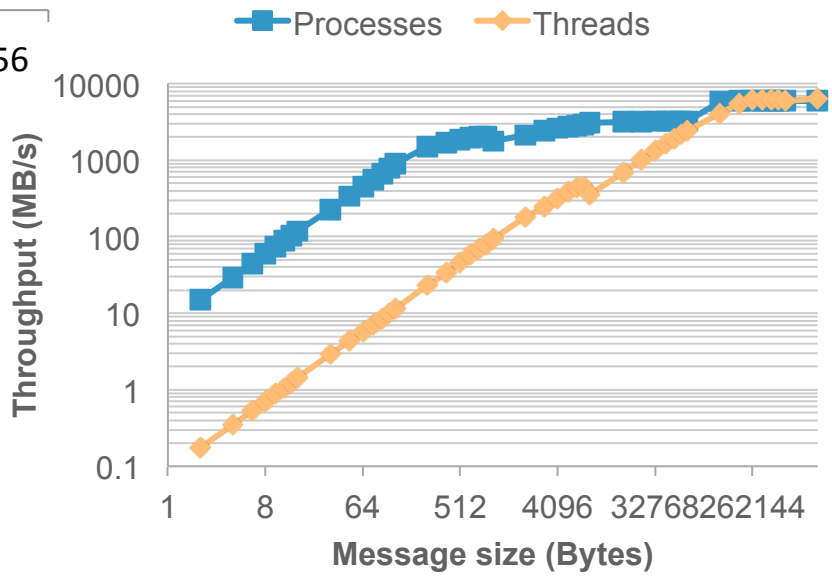
Overlap improvement using 2 interconnected processes





Scalability using increasing number of nodes (1 process per node)





Take Away

- Multi- and Many-core systems are already here
- Users are looking at different ways to utilize them
- Fewer resources means that we need ways of sharing – threading models sound like a good approach, but performance challenges need to be addressed
- Many cores means that some cores can be kept aside as "system cores"
- We have investigated both models