# From MPI to OpenSHMEM: Porting LAMMPS

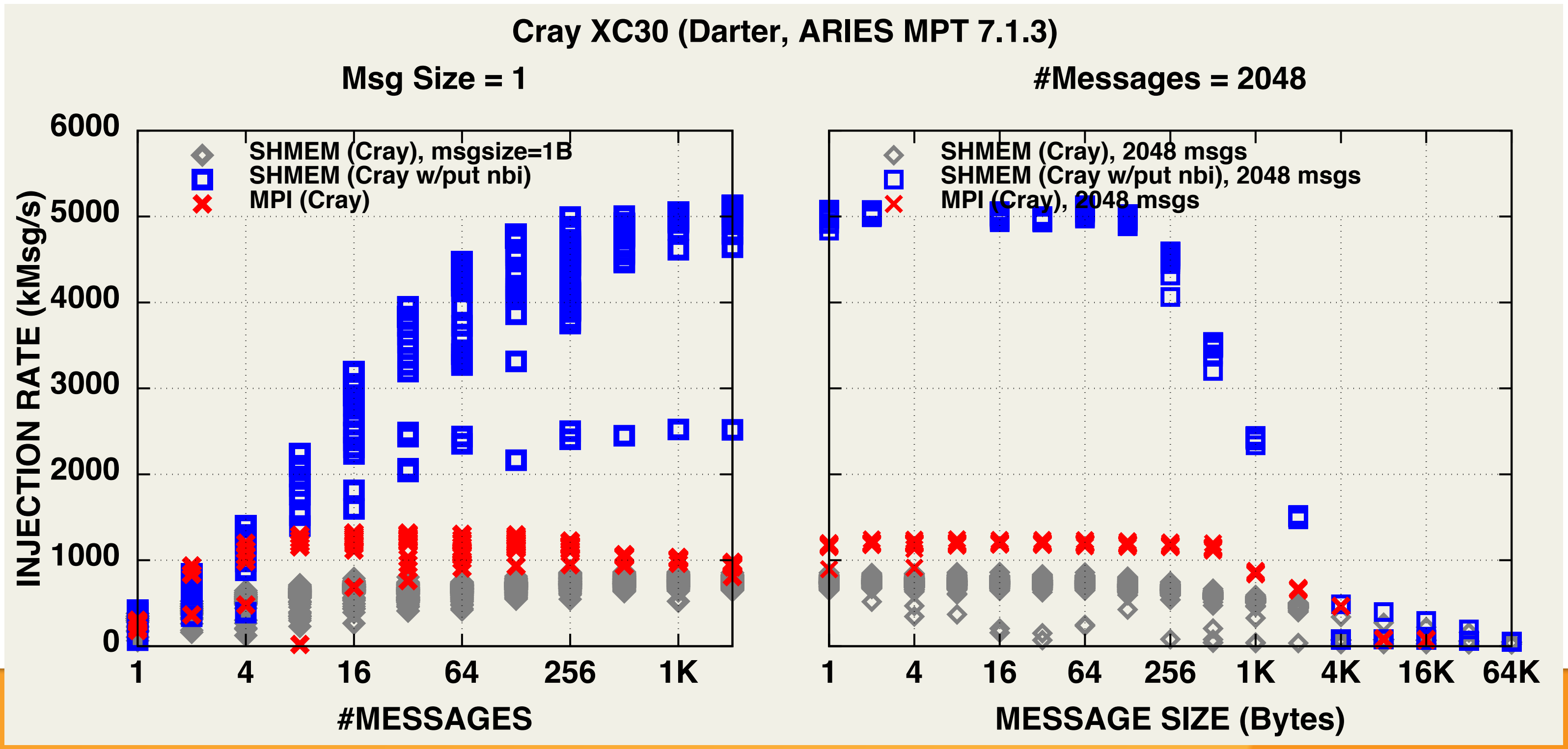
C. Tang, *A. Bouteiller*, T. Herault, M.G. Venkata, G. Bosilca

OpenSHMEM Workshop 2015, Aug. 5, Annapolis, MD

ICL UT INNOVATIVE COMPUTING LABORATORY THE UNIVERSITY of TENNESSEE

# Motivation

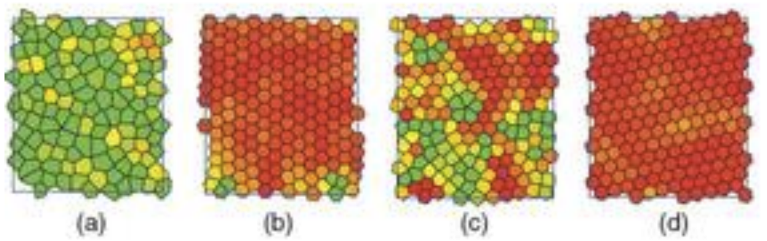
- 1 sided model: potential for large gains (major on injection rate, minor in latency)
- How hard is it to deploy, in practice ?
- The community needs some recipes and guidelines



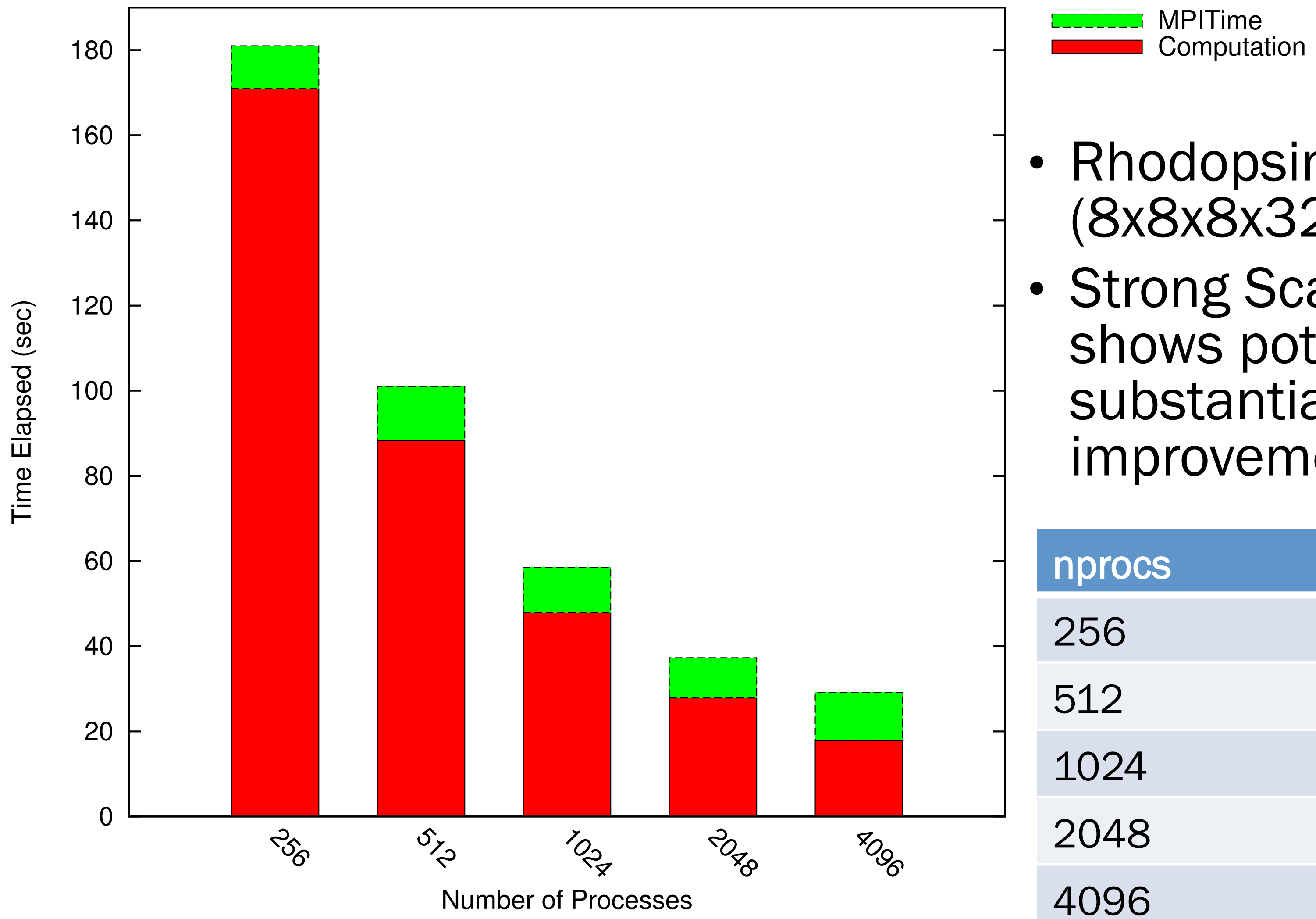
by Sandia National Lab

# Why LAMMPS

- Large-scale Atomic/Molecular Massively Parallel Simulator
- Widely used in production
- Solid-state materials (metals, semiconductors)
- Soft matter (biomolecules, polymers)
- Etc.
- MPI based application



# Why LAMMPS

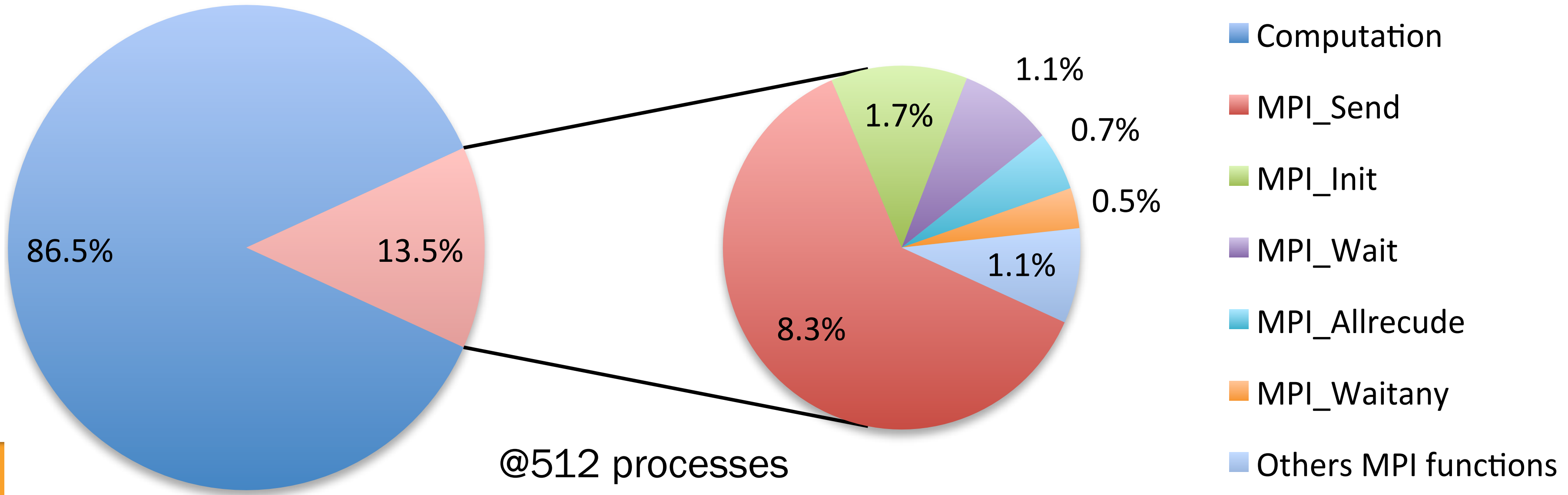


- Rhodopsin (8x8x8x32k atoms)
- Strong Scaling shows potential for substantial improvement

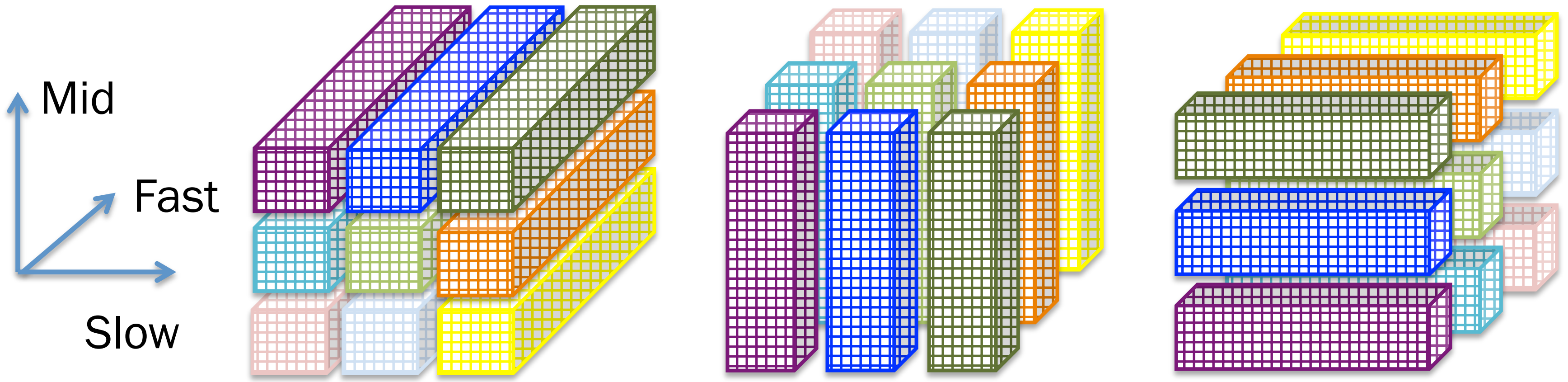
| nprocs | MPI% |
|---|---|
| 256 | 5.5 |
| 512 | 12.6 |
| 1024 | 18.1 |
| 2048 | 25.4 |
| 4096 | 38.5 |

# MPI Usage in LAMMPS

- Goal: hybrid MPI/Shmem application, upgrade to high return routines first
- Profiling with mpiP: more than 167 MPI call sites
  - MPI_Send dominates the MPI wait time.
  - Remap_3d() sends 32.45% of the data
    - Initial effort on this operation

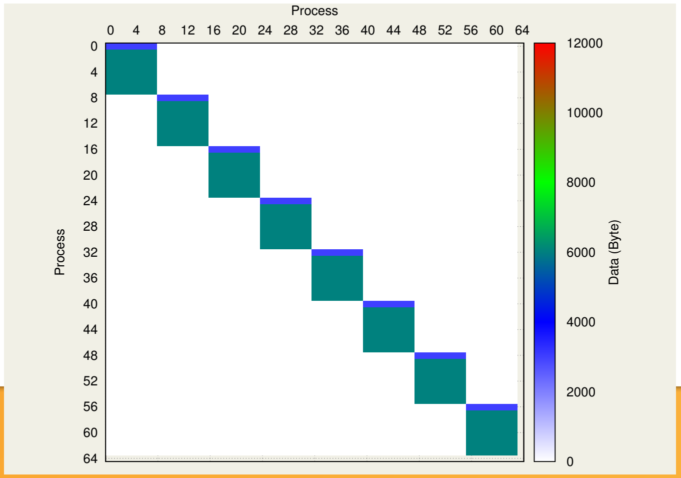


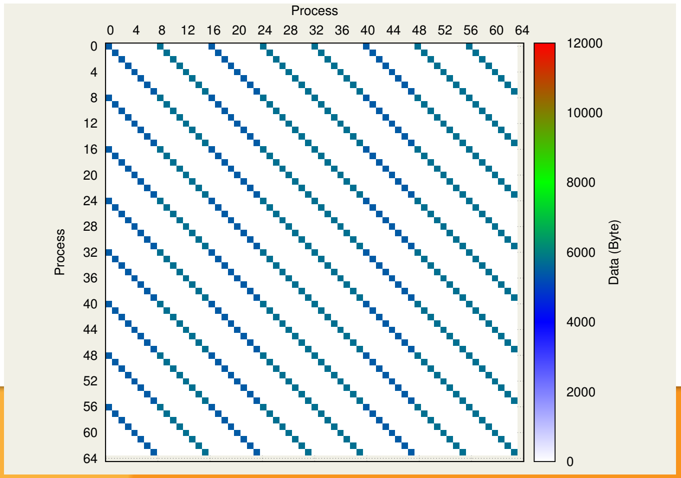
# 2-D decomposition of 3D-FFTs



1d-fft fast axis → Remap1 → 1d- fft mid axis → Remap2 → 1d-fft slow axis

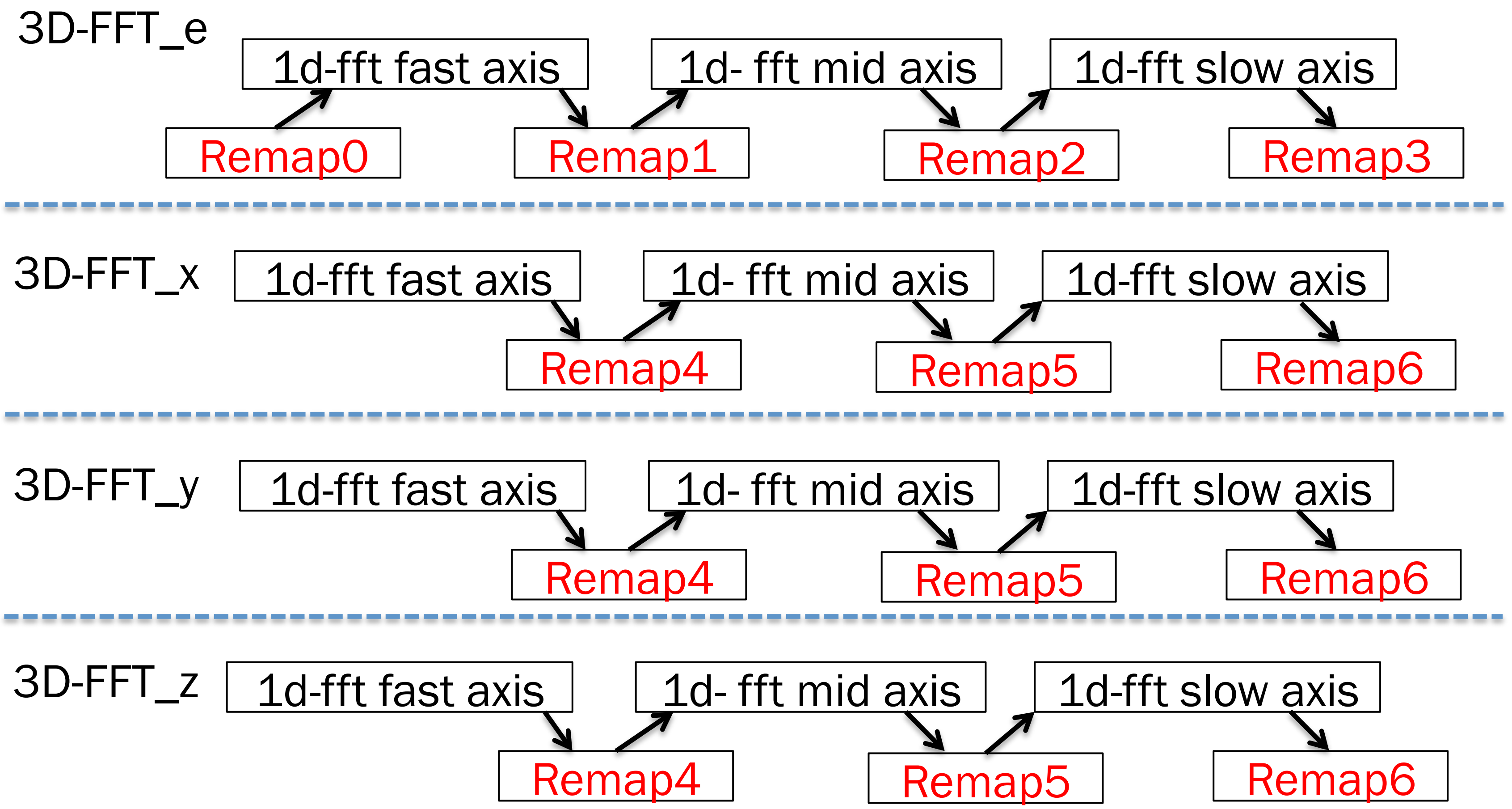
8x8 process grid



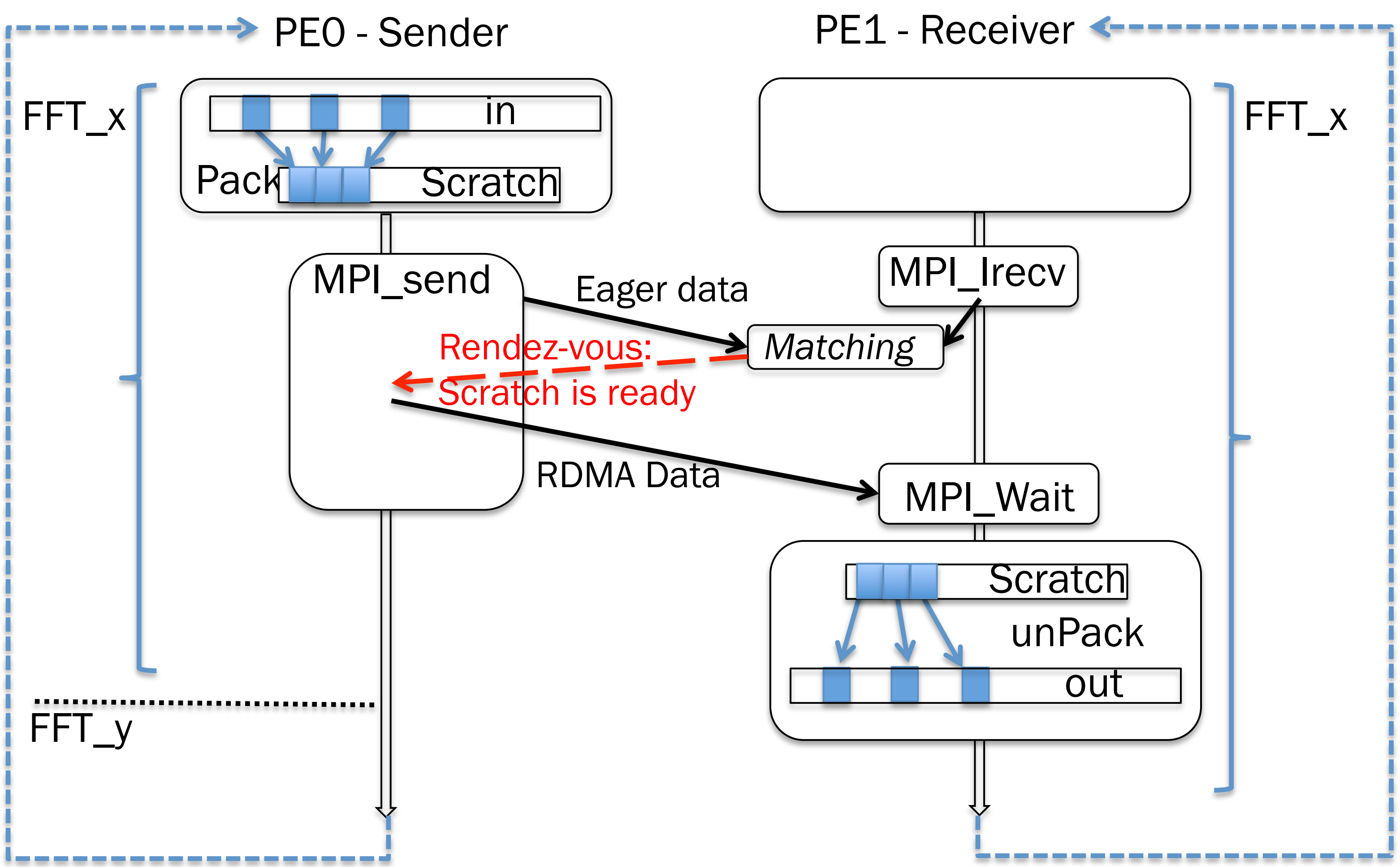


## 3D-FFT in LAMMPS – P(x,y,z,e)

- Multiple 3D FFTs per iteration, work on separate data structures

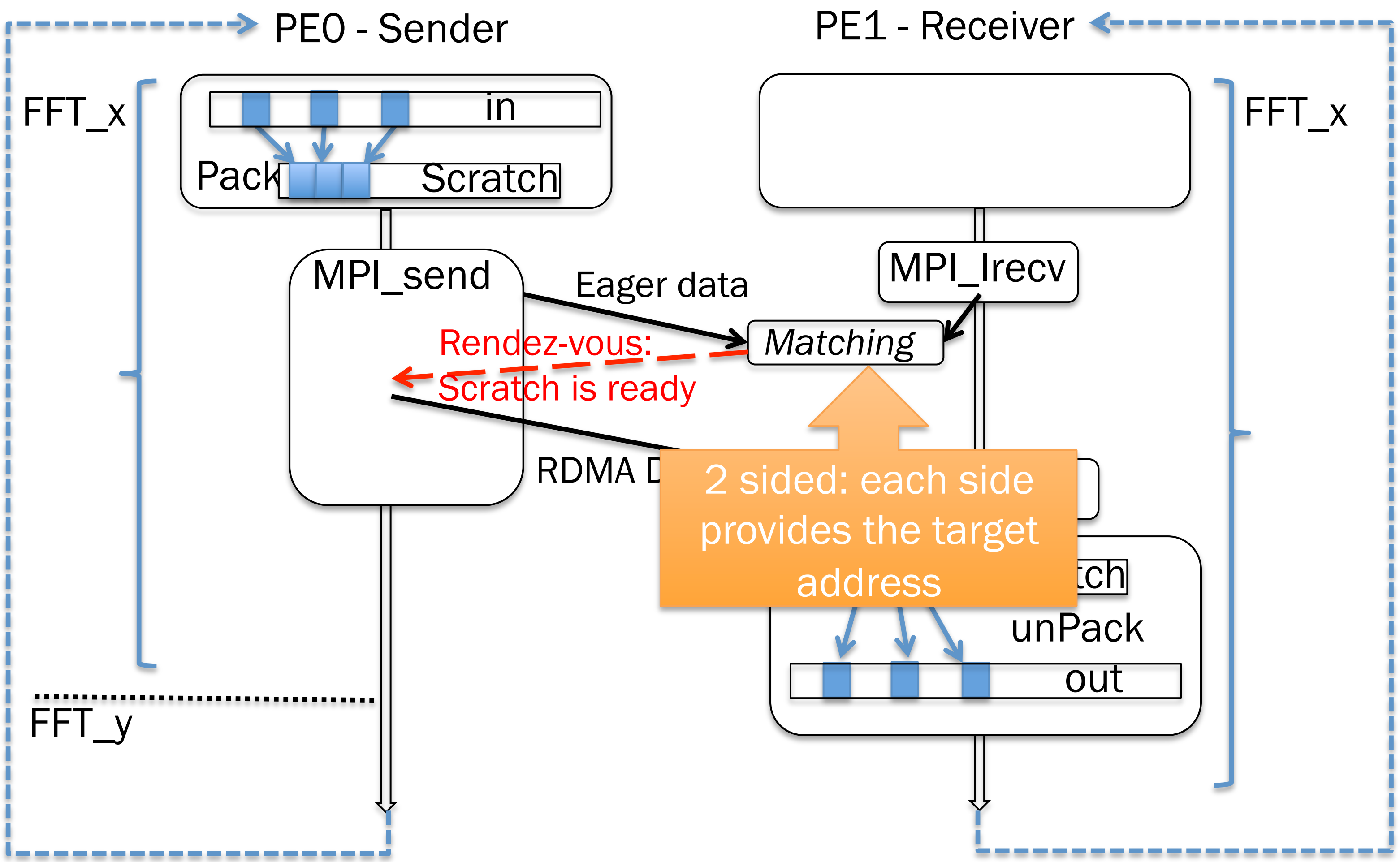


## MPI and its implicit synchros

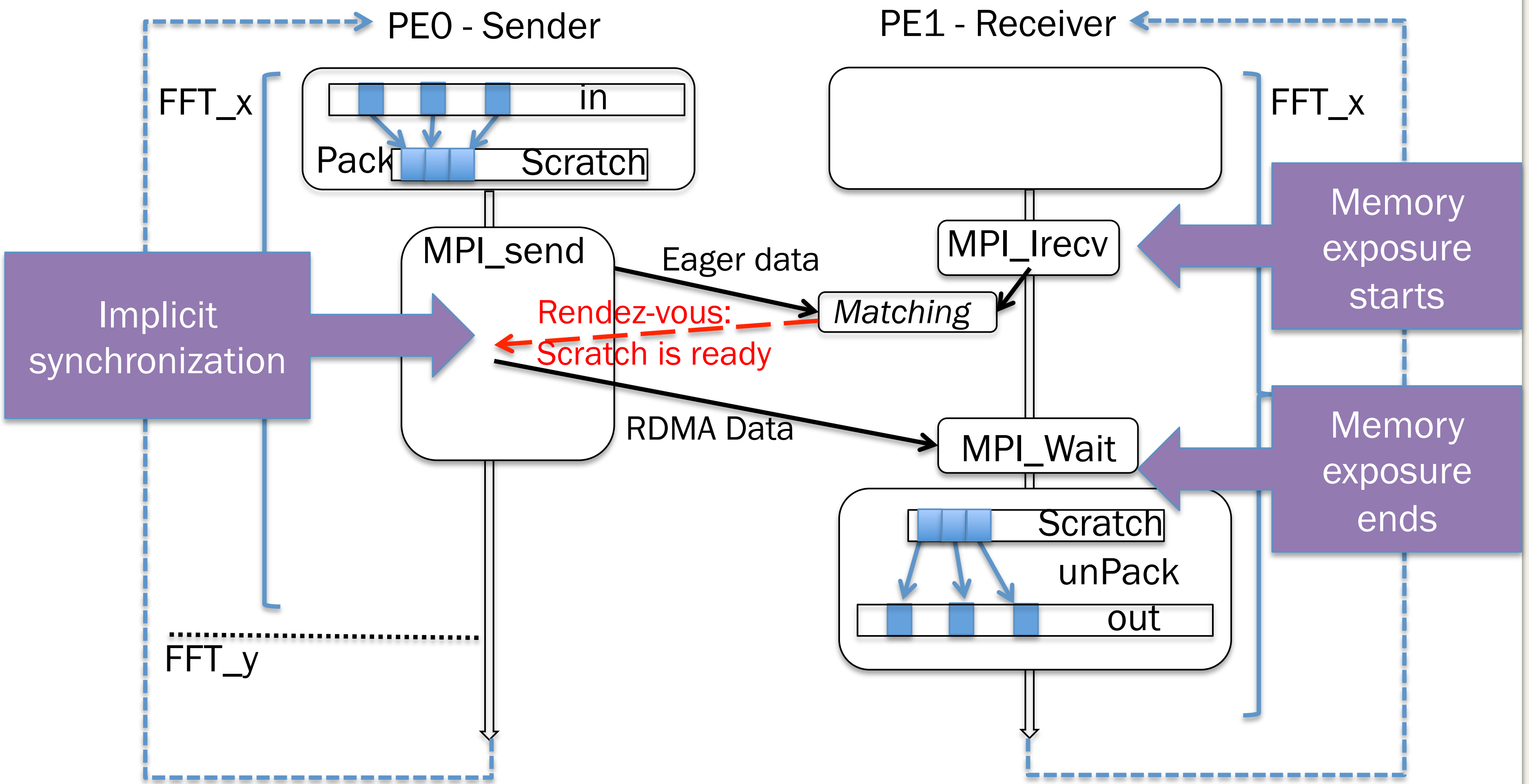


Simplified view: every process sends and receive from/to multiple peers

# MPI and its implicit synchros



# MPI and its implicit synchros



# Conversion: *put* target offsets

- 1sided: other processes need to understand each others' memory layout
  - Scratch buffers allocated in the symmetric address space
  - Each PE has a different communication plan
    - PE1 receives 1MB from PE0, PE2's offset in target scratch is then at *base+1MB*
    - PE2 receives 2MB from PE0, PE1's offset in target scratch is then at *base+2MB*
  - The plan is invariant: we exchange all offsets once, during startup

```
...
plan->remote_offset = (int *) shmem_malloc(nprocs*sizeof(int));
for( i = 0; i < plan->nrecv; i++)
  shmem_int_p(&plan->remote_offset[me], plan->recv_bufloc[i],
                    plan->recv_proc[i]);
shmem_fence();
...
```

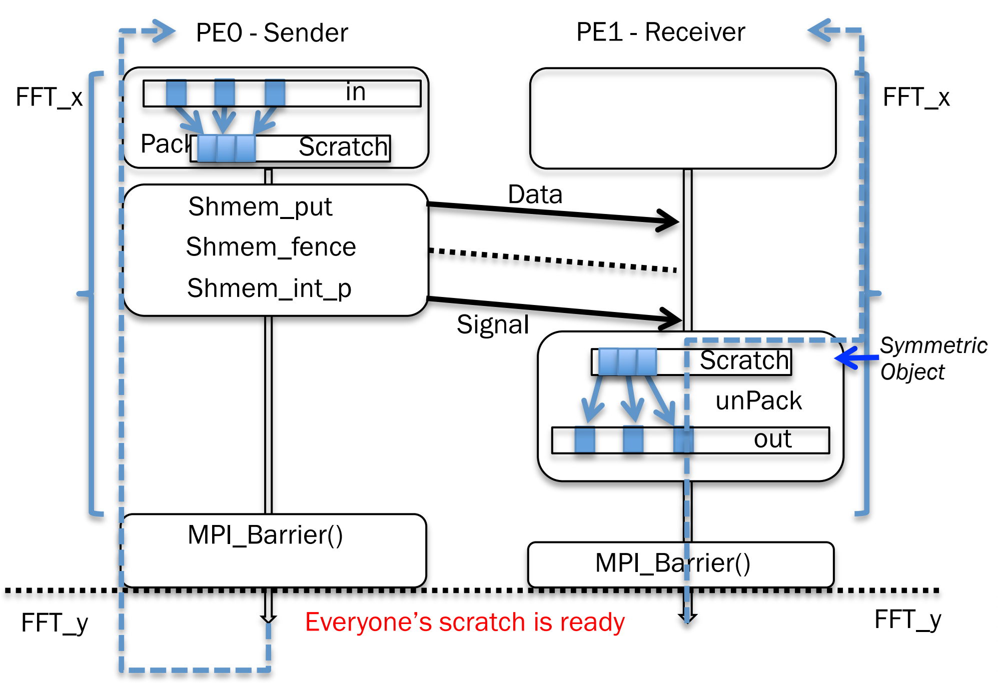
Listing 1.2: Exchanging the offsets in the target scratch buffers; a parameter to `shmem_put` that was not required with `MPI_Send`.

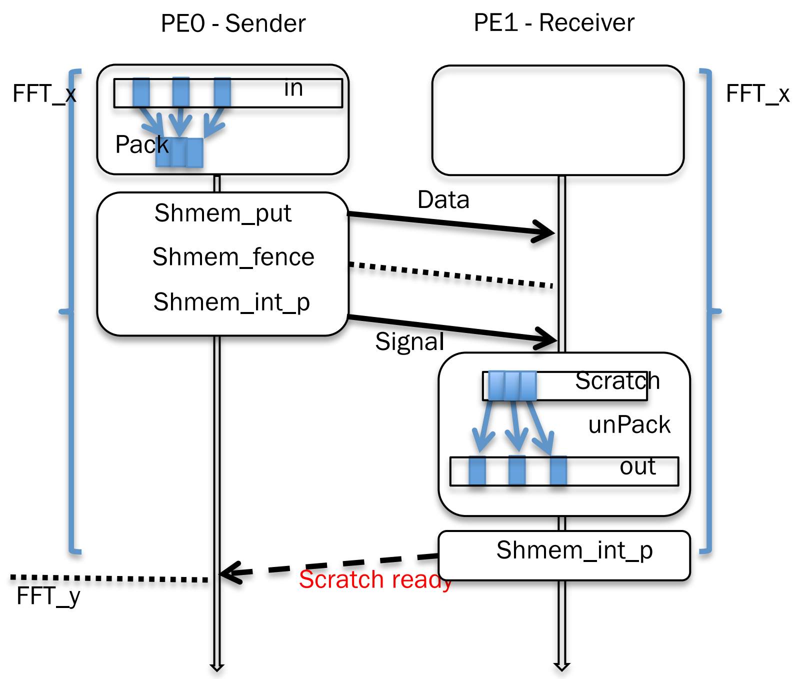
# Conversion: *put* completion signaling

- In two sided, MPI_Waitany tracks the completion of iRecv
- In one sided, the target cannot tell directly when put has completed
  - Plan->remote_status is initialized in status “WAIT”, the target issues
    - `shmem_putmem(plan->offset[me], …, tgt);`
    - `shmem_fence();`
    - `shmem_int_p(plan->remote_status[me], READY, tgt);`
  - Unpack starts when status is read as READY in a while loop over all statuses

## Conversion: signaling exposure

- The same remap plan is reused multiple times
  - Risk of scratch buffer overwrite between different axis
- Multiple possible strategies
  - Bulk synchronization (Barrier)
  - Fine grain synchronization (per-scratch readyness with shmem_int_p)



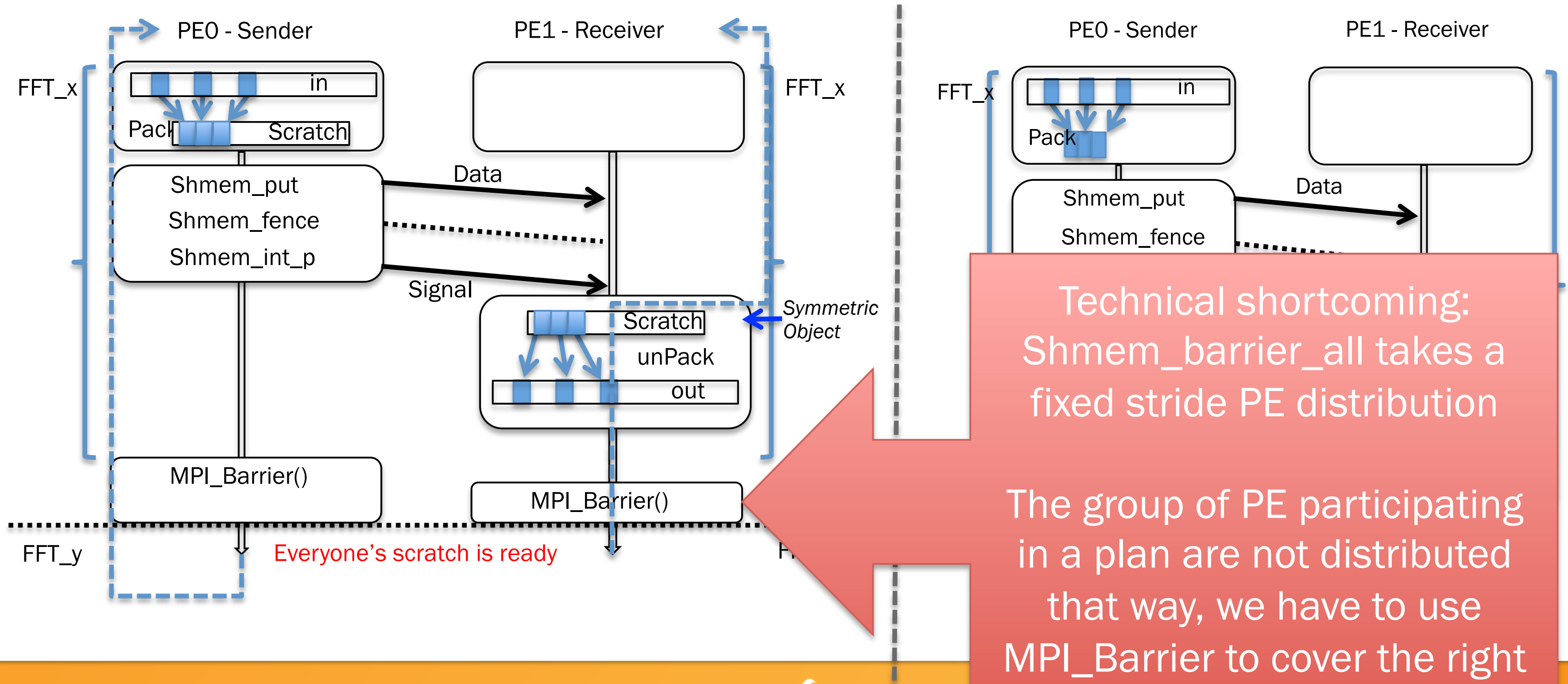


HybridBarrier

ChkBuff/AdvChkBuff

# Conversion: signaling exposure

- The same remap plan is reused multiple times
  - Risk of scratch buffer overwrite between different axis
- Multiple possible strategies
  - Bulk synchronization (Barrier)
  - Fine grain synchronization (per-scratch readyness with shmem_int_p)



Technical shortcoming: Shmem_barrier_all takes a fixed stride PE distribution

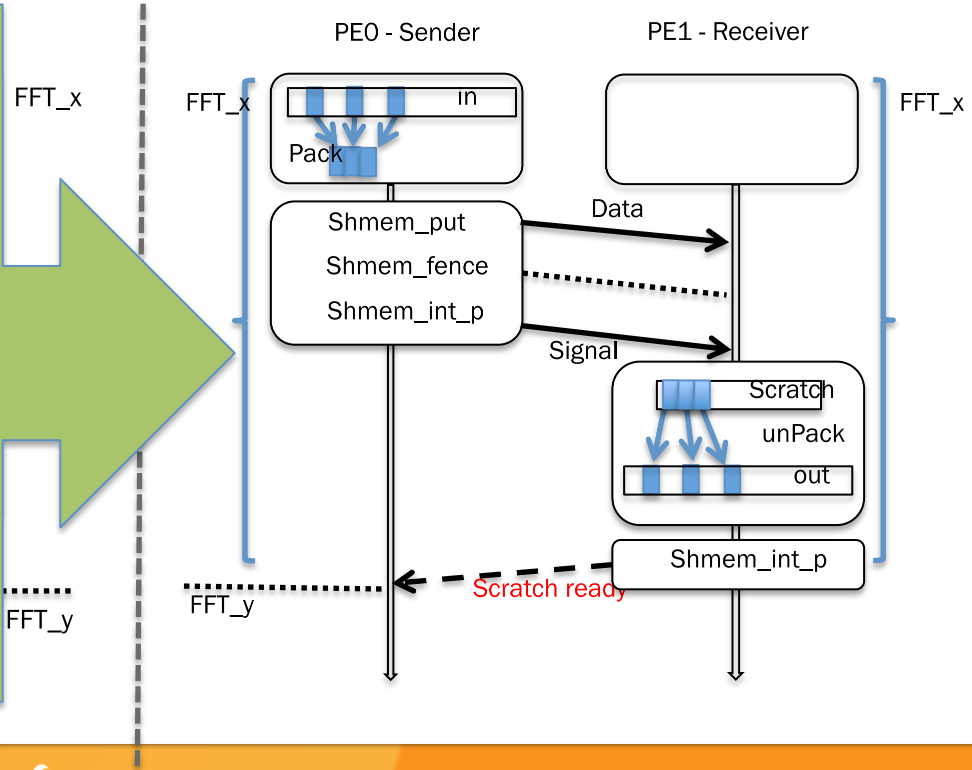
The group of PE participating in a plan are not distributed that way, we have to use MPI_Barrier to cover the right group....

# Conversion: signaling exposure

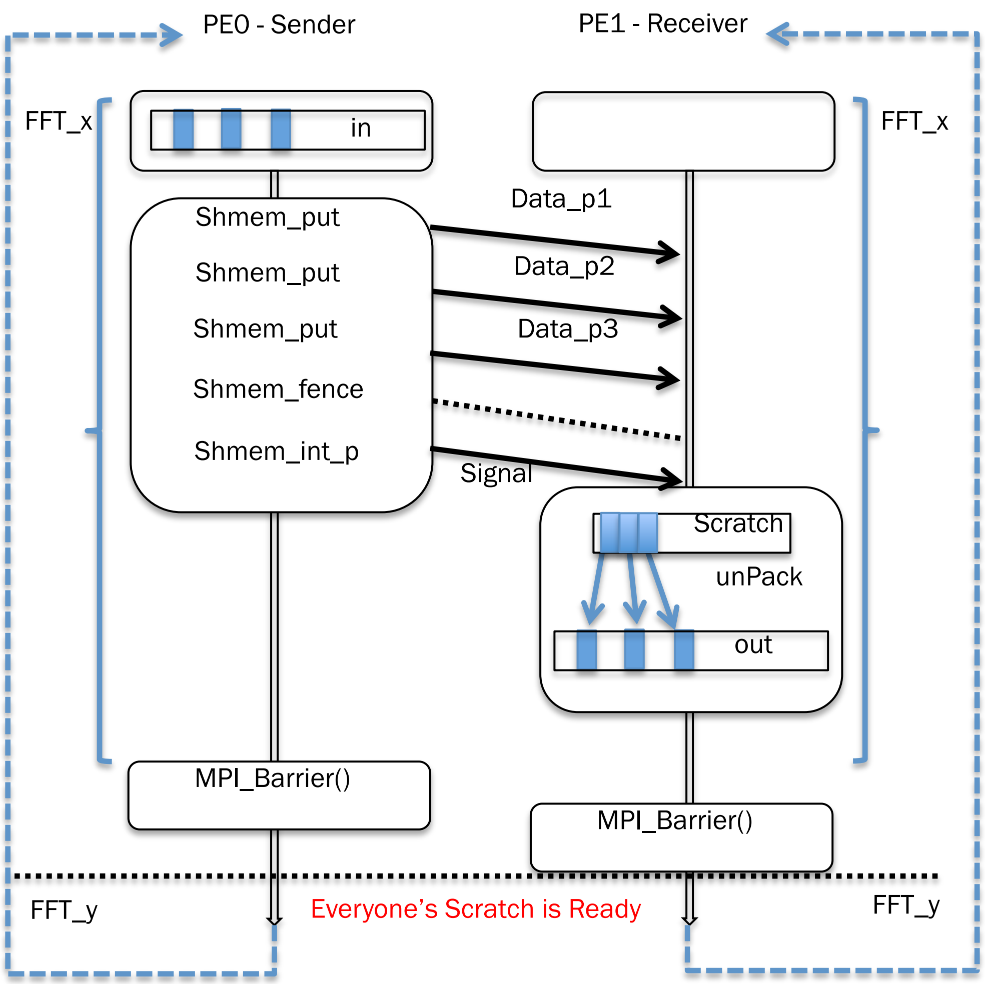
- The same remap plan is reused multiple times
  - Risk of scratch buffer overwrite between different axis
- Multiple possible strategies
  - Bulk synchronization (Barrier)
  - Fine grain synchronization (per-scratch readyness with shmem_int_p)

This sync pattern looks like 2 sided, but:

1. Rendez-vous to notify exposure comes earlier
2. Opportunistic unpacking can overlap "target not ready" wait time with unpack computation



# Taking advantage of injection rate



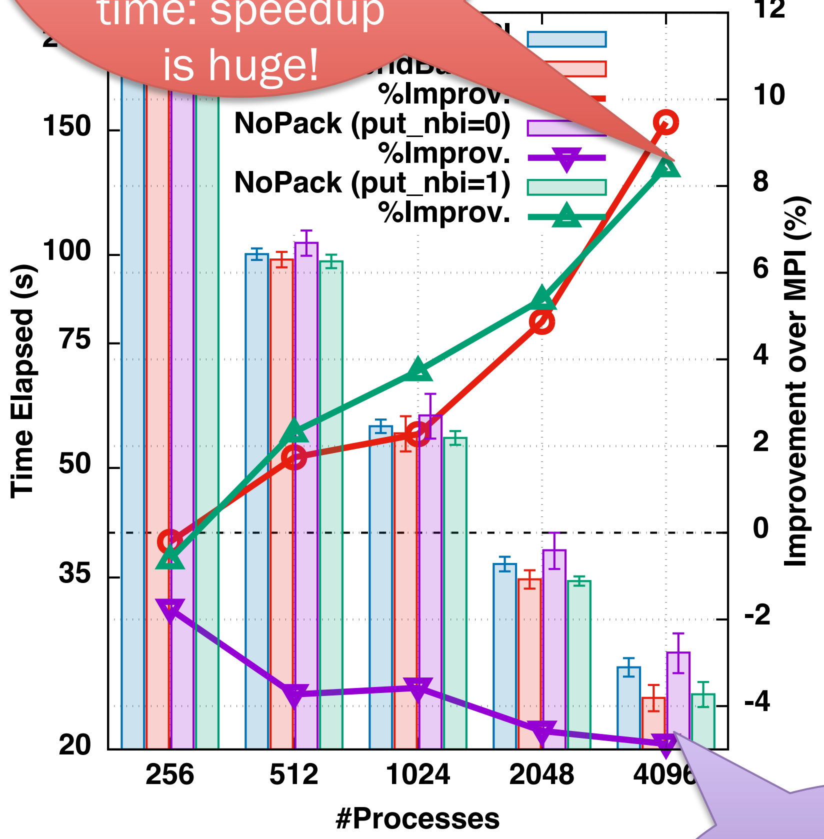
- Remove packing
  - multiple small puts
  - 1 less memory copy
  - Bulk put-completion synchronization with one `fence();` `shmem_int_p();`

NoPack

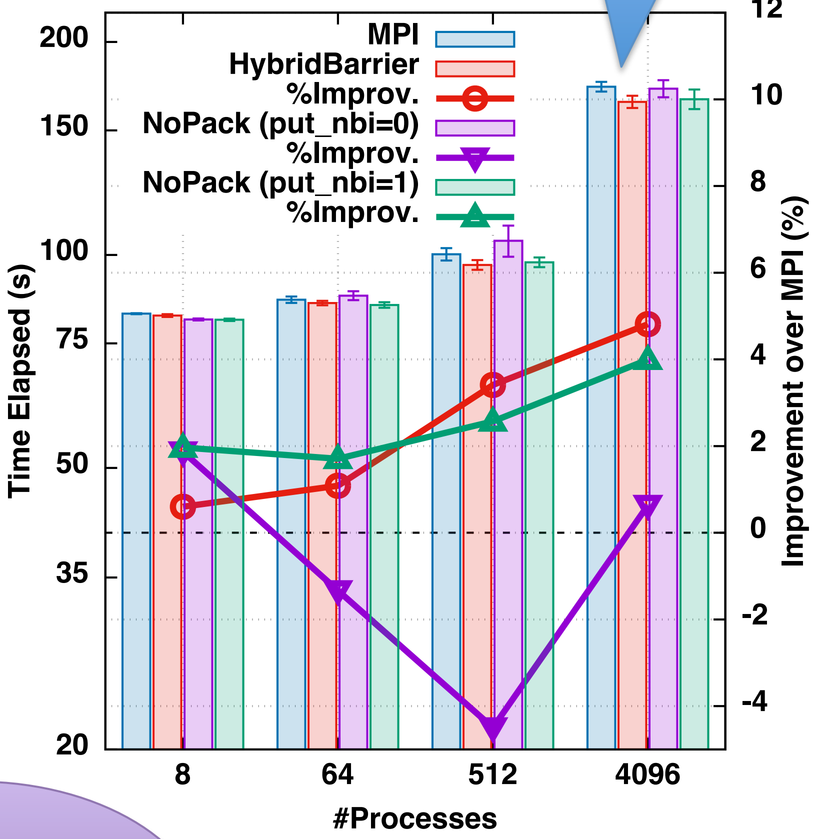
## Performance

- Titan Cray XK7 supercomputer (@ORNL)
  - Cray MPICH 6.3.0, Cray-shmem 6.3.0
  - Rhodopsin protein input problem, 8x8x8x32k atoms (strong scaling) or 32k atoms per PE (weak scaling)

# Pack/NoPack



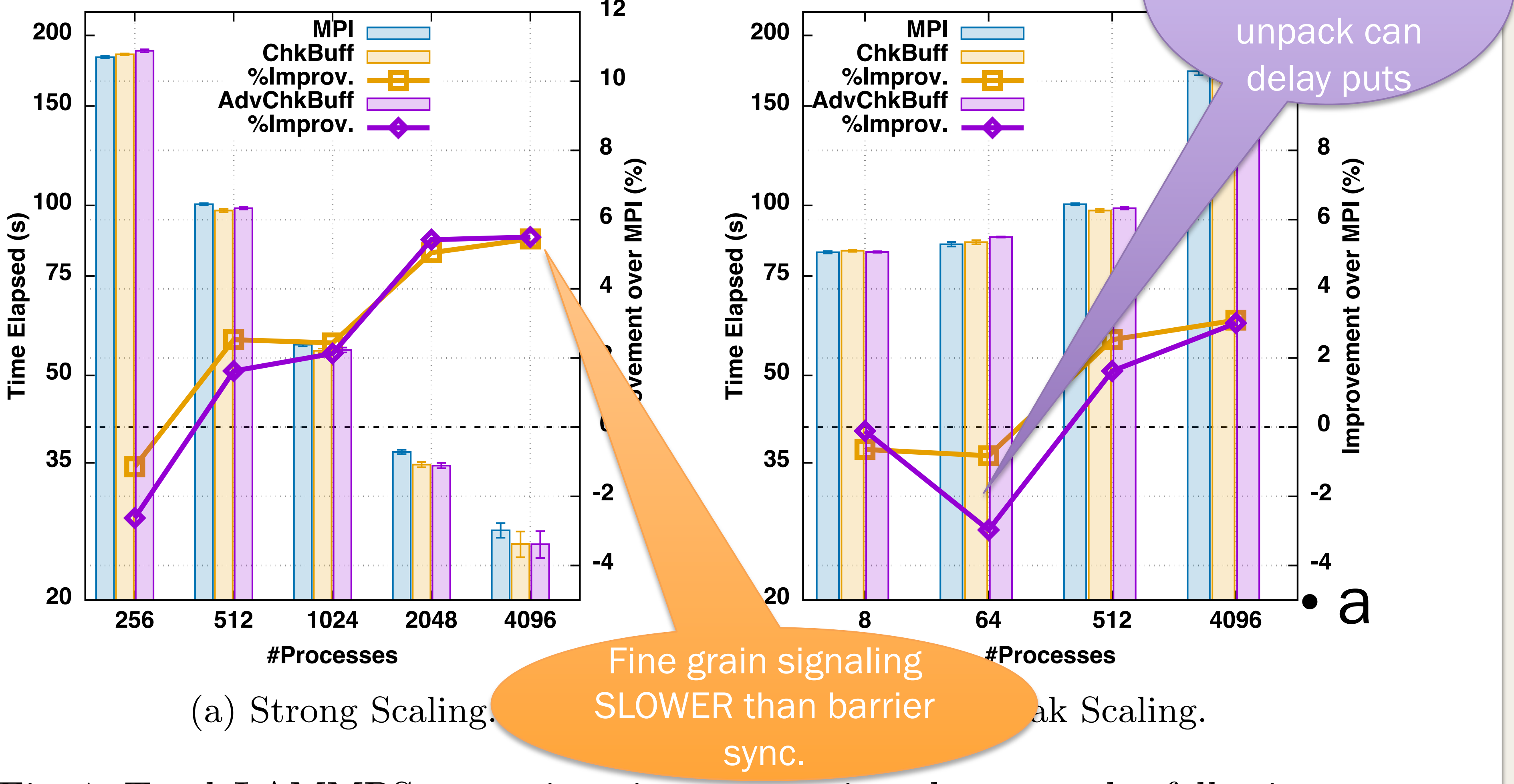
(a) Strong Scaling.



(b) Weak Scaling.

Fig. 3: Total LAMMPS execu… …son between the following versions: original MPI, *HybridBarrie…* …*Pack* (with and w/o non-blocking shmem_put).

# Fine grain signaling



(a) Strong Scaling. ak Scaling.

• a

Fig. 4: Total LAMMPS execution time comparison between the following versions: original MPI, *ChkBuff*, and *AdvChkBuff*.

## Conclusions

- Accelerated the `remap3d` routine in LAMMPS with OpenSHMEM
- ☺ Shmem does improve performance vs MPI
  - No costly handling of unexpected messages, no internal buffering, early exposure of “recv” buffers, bulk synchronization, etc.
- 😐 Missing features (but soon?)
  - Weird structure for groups on which collective operate, porting MPI code with collective on MPI groups is hard!
  - shmem_put too synchronizing by default: if one can relax put-completion semantic at the origin, huge performance gains
  - Missing shmem_put_notify (a proposal is ongoing in the std body)
- ☹ Hard to predict performance
  - Explicit handling of all synchronization can be cumbersome to end-users
  - Default bulk synchronization not available (see missing features) => one has to implement it himself
  - Bad handling of synchronization can get worse performance than implicit sync. in MPI
  - Often hard to predict the best strategy, performance portability diminished