

# MPI Shared Memory Model

---

MPI processes behaving as threads

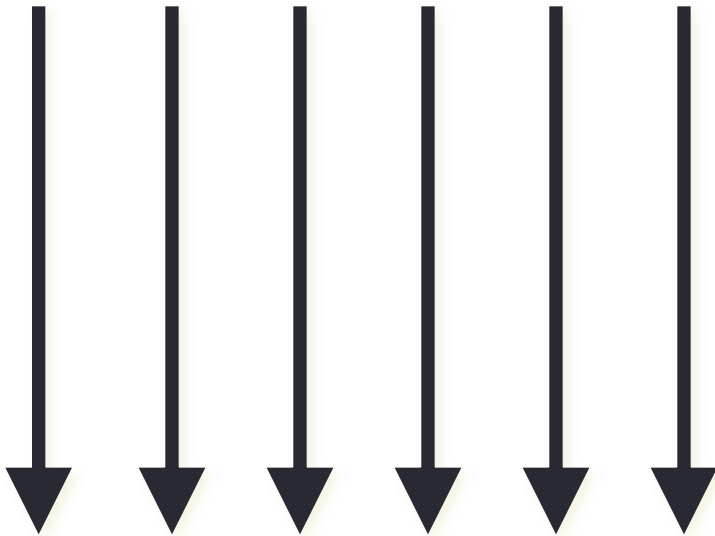
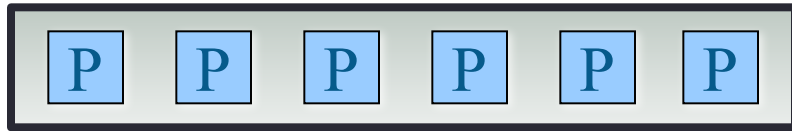
# Overview

- Motivation
- Node-local communicators
- Shared window allocation
- Synchronisation

# MPI + OpenMP

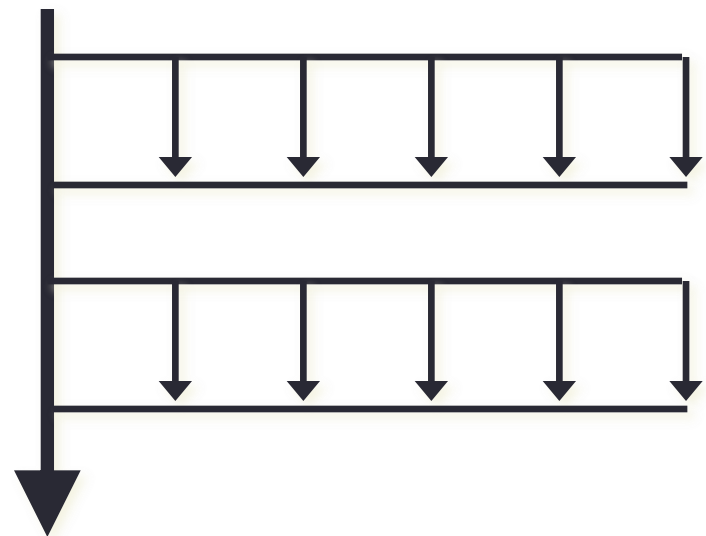
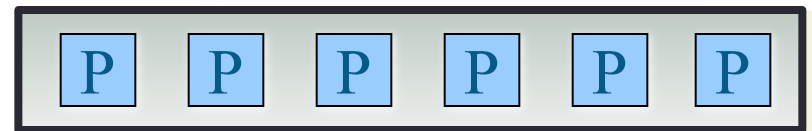
- In OMP parallel regions, all threads access shared arrays
  - why can't we do this with MPI processes?

MPI



|epcc|

MPI + OpenMP



# Exploiting Shared Memory

- With standard RMA
  - publish local memory in a collective shared window
  - can do read and write with `MPI_Get` / `MPI_Put`
  - (plus appropriate synchronisation)
- Seems wasteful on a node
  - why can't we just read and write directly as in OpenMP?
- Requirement
  - technically requires the Unified model
    - where there is no distinction between RMA and local memory
  - can check this calling `MPI_Win_get_attr` with `MPI_WIN_MODEL`
    - model should be `MPI_WIN_UNIFIED`
  - this is not a restriction in practice for standard CPU architectures

# Procedure

- Processes join separate communicators for each node
- Shared array allocation across all processes on a node
  - OS can arrange for it to be a single global array
- Access memory by indexing outside limits of local array
  - e.g. `localarray[-1]` will be last entry on the previous process
- Need appropriate synchronisation for local accesses
- Still need MPI calls for internode communication
  - e.g. standard send and receive

# Splitting the communicator

```
int MPI_Comm_split_type(MPI_Comm comm, int split_type,  
    int key, MPI_Info info, MPI_Comm *newcomm)
```

```
MPI_COMM_SPLIT_TYPE(COMM, SPLIT_TYPE, KEY, INFO,  
    NEWCOMM, IERROR)
```

```
INTEGER COMM, SPLIT_TYPE, KEY, INFO, NEWCOMM, IERROR
```

- comm: parent communicator, e.g. MPI\_COMM\_WORLD
- split\_type: MPI\_COMM\_NODE
- key: controls rank ordering within sub-communicator
- info: can just use default: MPI\_INFO\_NULL

# Example

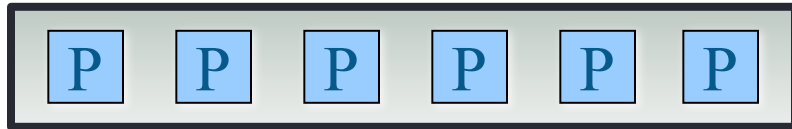
```
MPI_Comm_split_type(MPI_COMM_WORLD, MPI_COMM_TYPE_SHARED,  
rank, MPI_INFO_NULL, &nodecomm);
```

**COMM\_WORLD**

size = 12

rank

0 1 2 3 4 5



6 7 8 9 10 11



0 1 2 3 4 5

rank  
size = 6  
**nodecomm**

0 1 2 3 4 5

rank  
size = 6  
**nodecomm**

# Allocating the array

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

```
MPI_WIN_ALLOCATE_SHARED(SIZE, DISP_UNIT, INFO, COMM, BASEPTR,  
    WIN, IERROR)
```

```
INTEGER(KIND=MPI_ADDRESS_KIND) SIZE, BASEPTR
```

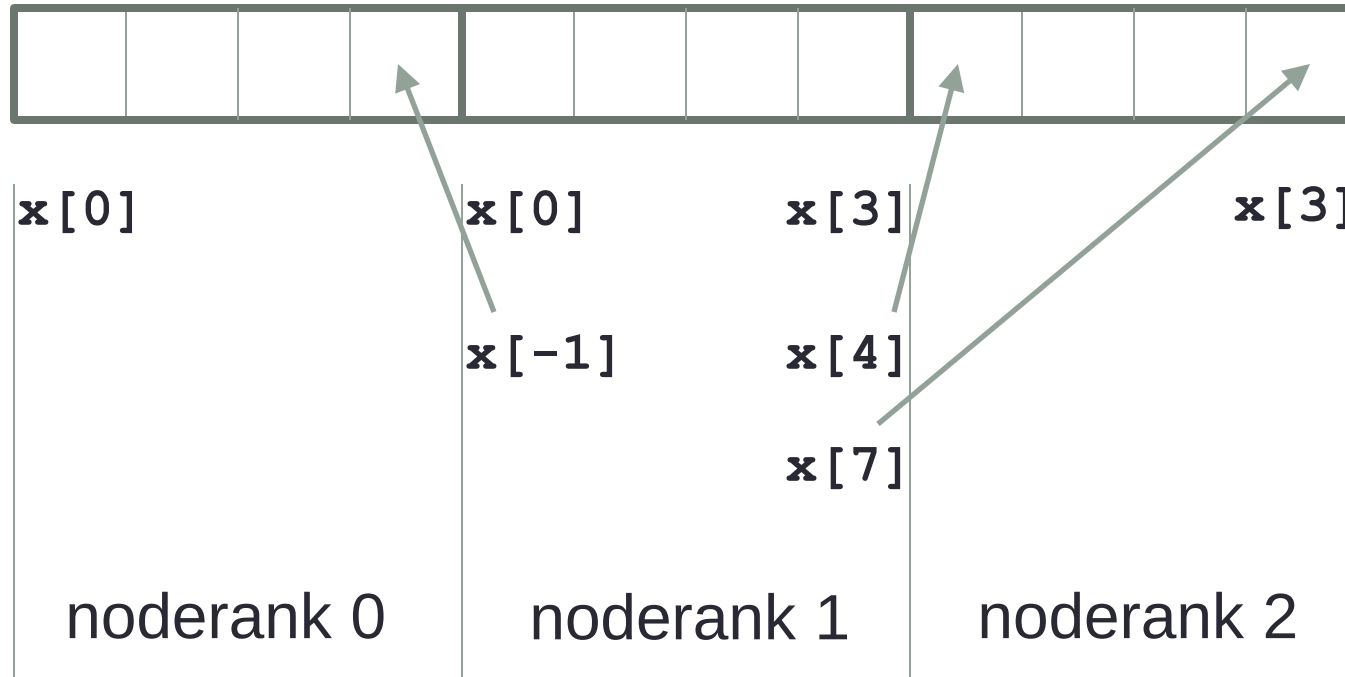
```
INTEGER DISP_UNIT, INFO, COMM, WIN, IERROR
```

- size: window size in bytes
- disp\_unit: basic counting unit in bytes, e.g. sizeof(int)
- info: can just use default: MPI\_INFO\_NULL
- comm: parent comm (must be within a single node)
- baseptr: allocated storage
- win: allocated window

# Traffic Model Example

```
MPI_Comm nodecomm;  
int *oldroad;  
MPI_Win nodewin;  
MPI_Aint winsize;  
int displ_unit;  
  
winsize = (nlocal+2)*sizeof(int);  
  
// displacements counted in units of integers  
disp_unit = sizeof(int);  
  
MPI_Win_allocate_shared(winsize, displ_unit,  
                        MPI_INFO_NULL, nodecomm, &oldroad, &nodewin);
```

# Shared Array with winsize = 4



# Synchronisation

- Can do halo swapping by direct copies
  - need to ensure data is ready beforehand and available afterwards
  - requires synchronisation, e.g.. MPI\_Win\_fence
  - takes hints – can just set to default of 0
- Entirely analogous to OpenMP
  - bracket remote accesses with omp\_barrier or begin / end parallel

```
MPI_Win_fence(0, nodecomm);  
oldroad[nlocal+2] = oldroad[nlocal]  
oldroad[-1]      = oldroad[0];  
MPI_Win_fence(0, nodecomm);
```

# Off-node comms

- Direct read / write only works within node
- Still need MPI calls for inter-node
  - e.g. `noderank = 0` and `noderank = nodesize-1` call `MPI_Send / Recv`
  - could actually use *any* rank to do this ...
- This must take place in `MPI_COMM_WORLD`

# Conclusion

- Relatively simple syntax for shared memory in MPI
  - much better than roll-your-own solutions
- Possible use cases
  - on-node computations without needing MPI
  - one copy of static data per node (not per process)
- Advantages
  - an incremental “plug and play” approach unlike MPI + OpenMP
- Disadvantages
  - no automatic support for splitting up parallel loops
  - global array may have halo data sprinkled inside
  - may not help in some memory-limited cases