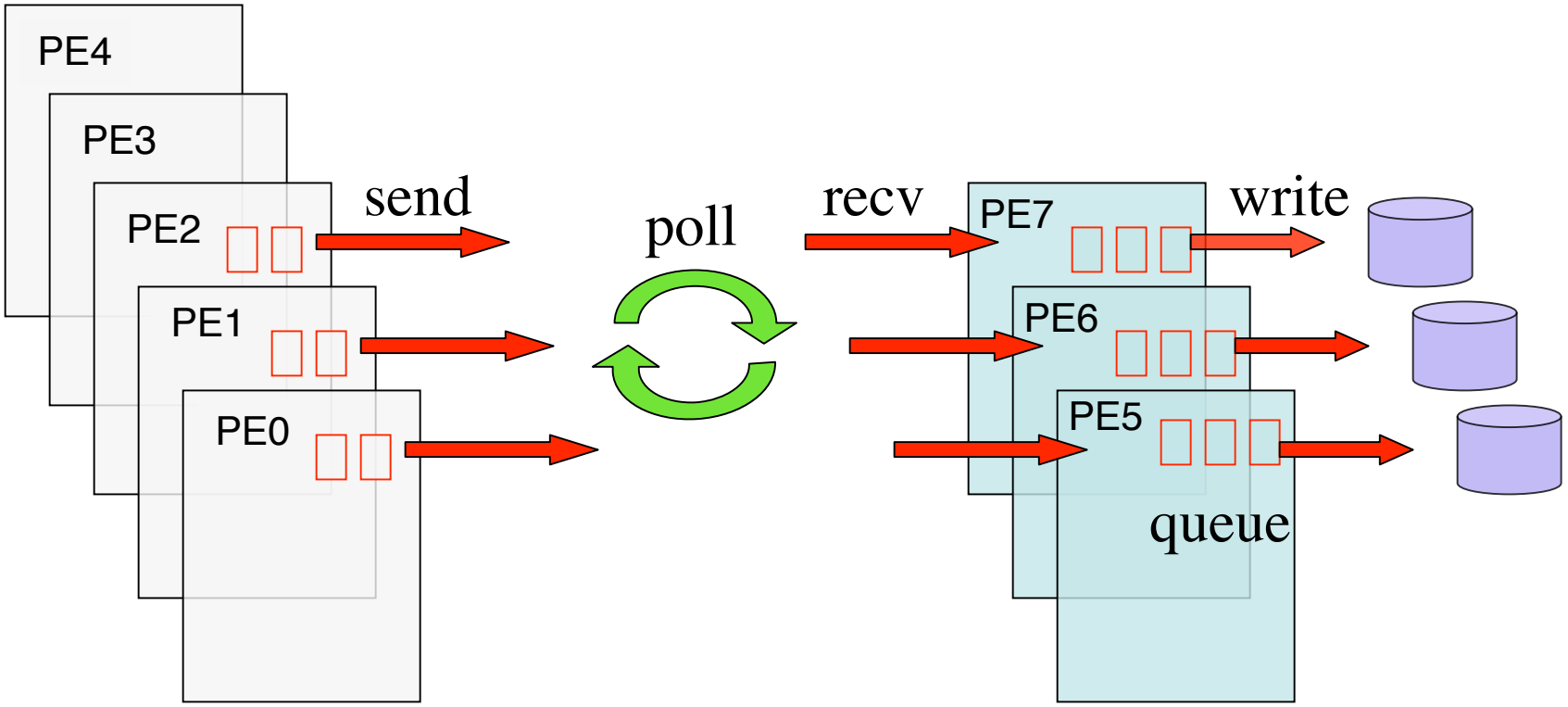
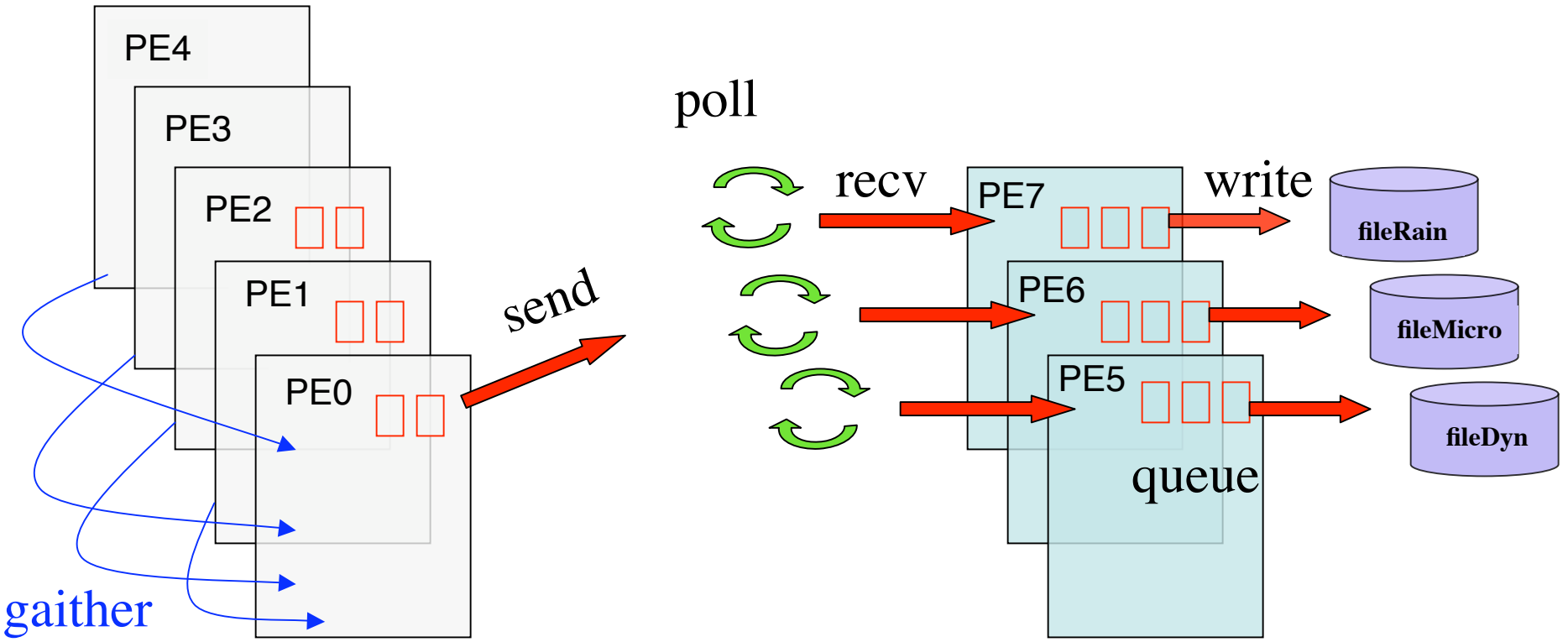
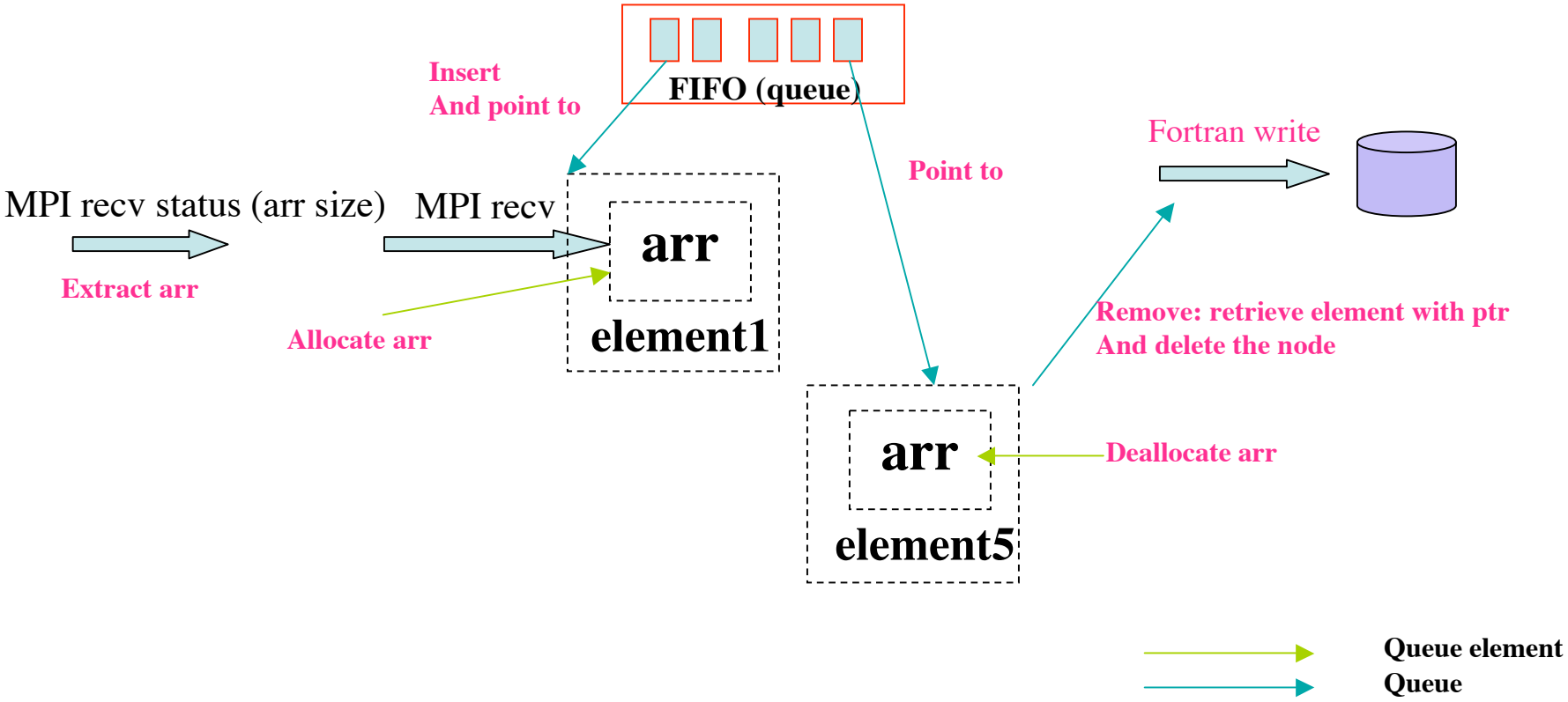
System Architecture



Multiple IO Nodes for Multiple Output Files (Current Implementation)



Implementation details

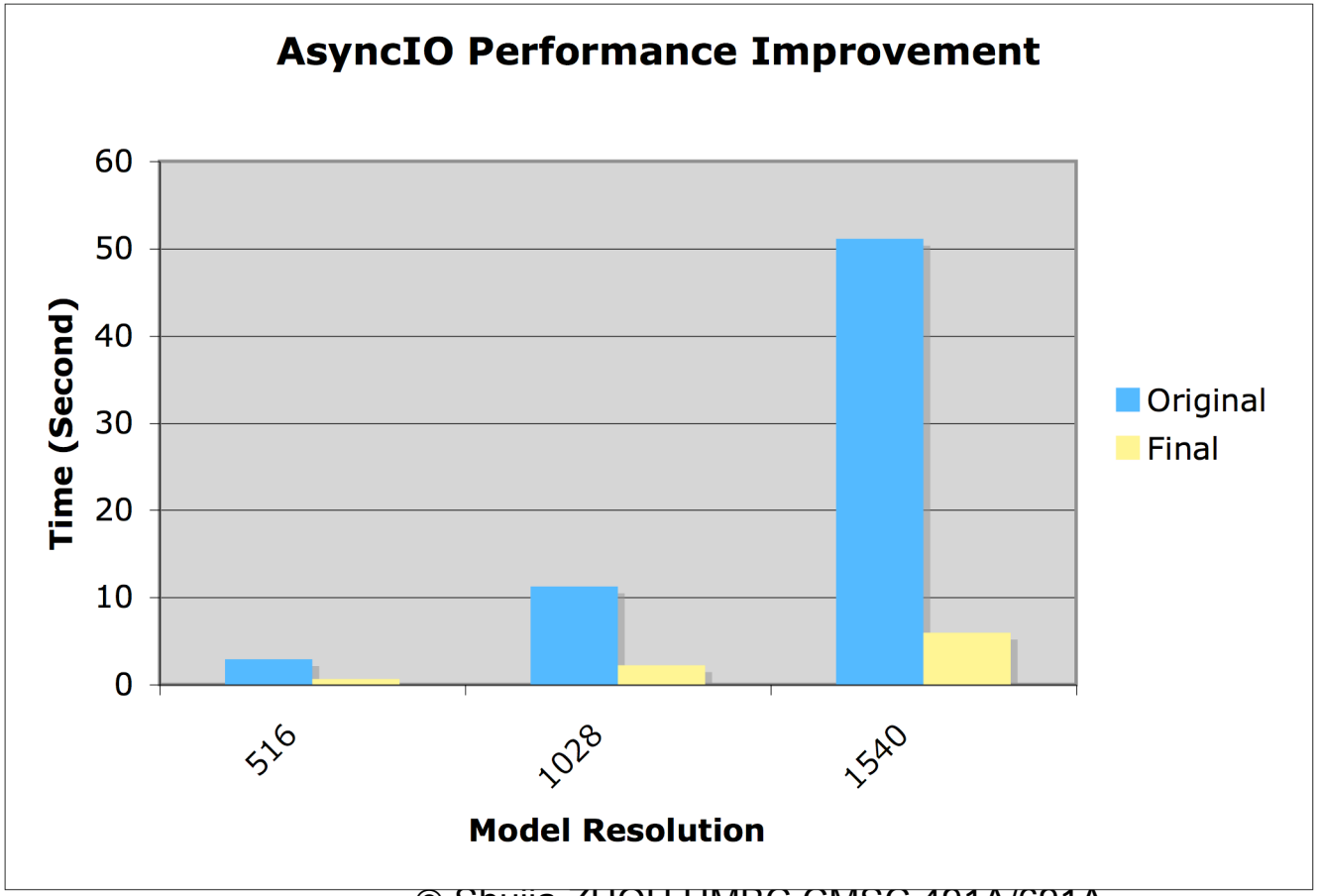


Implementation with MPI and Fortran 95

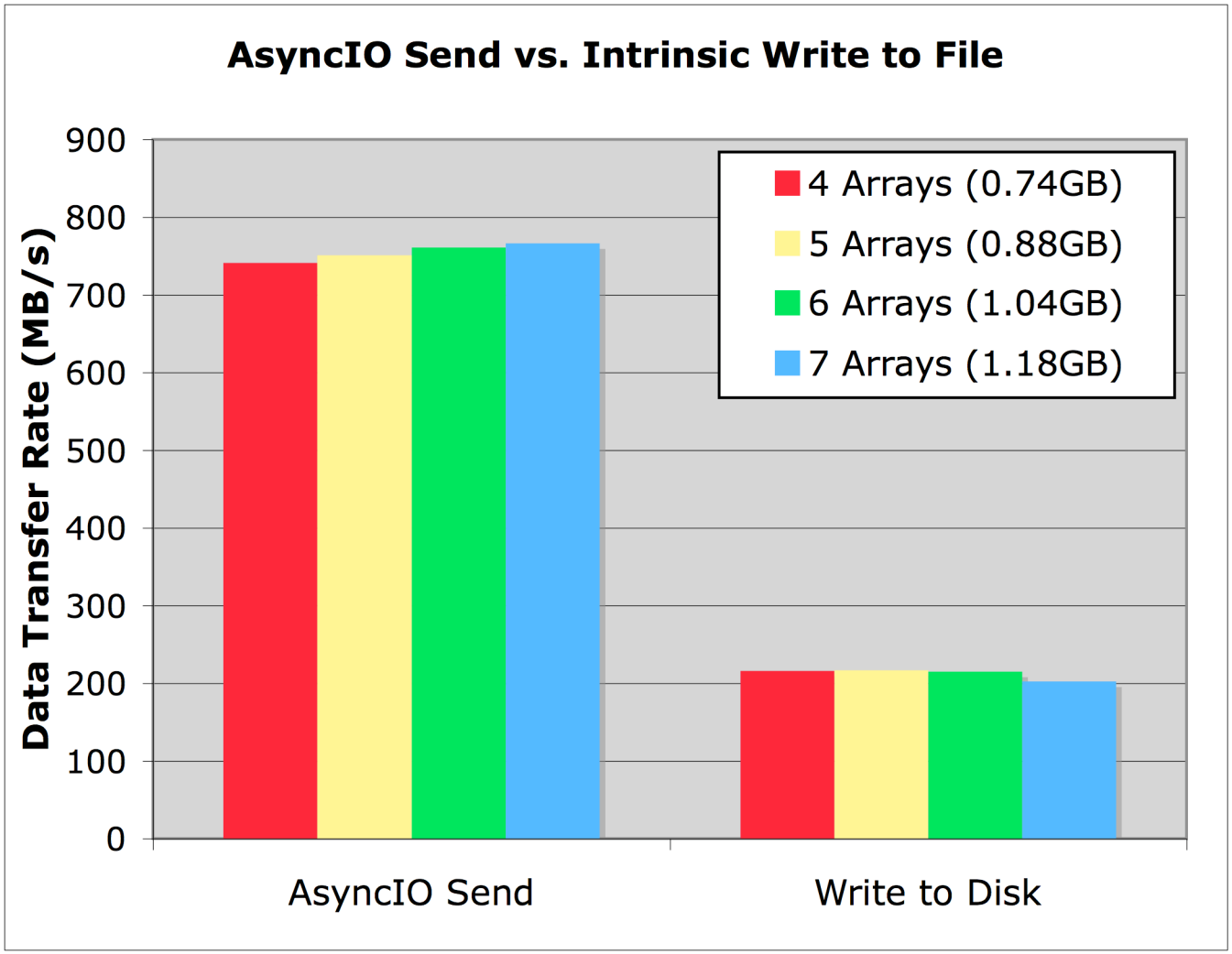
Integration into An Application: NASA GCE code

- Create a sub MPI communicator for model, ModelComm, in the new driver,
- Replace the original MPI_Comm_World with ModelComm
- Replace all the relevant “write” statements with AsyncIOSend() to send data to IO nodes
- Execute AsyncIOFlush() to flush the grouped data in IO nodes to discs

The I/O Performance Gain of Using AsyncIO in the Microphysics Portion of GCE (on NASA Columbia SGI Altix)

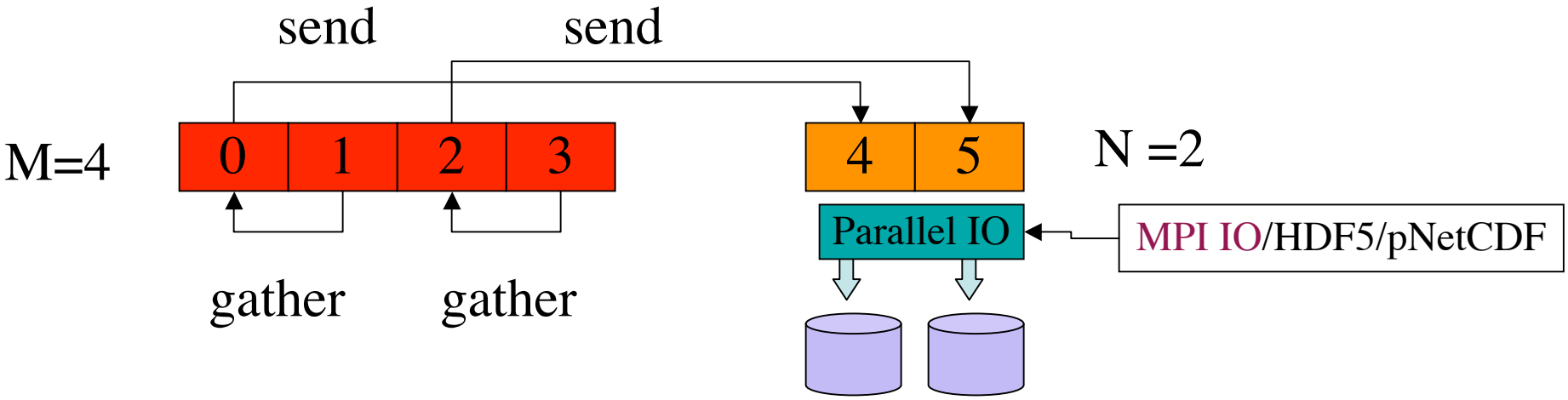


AsyncIO Data Transfer VS. Standard Write Speeds (on a Linux Cluster)

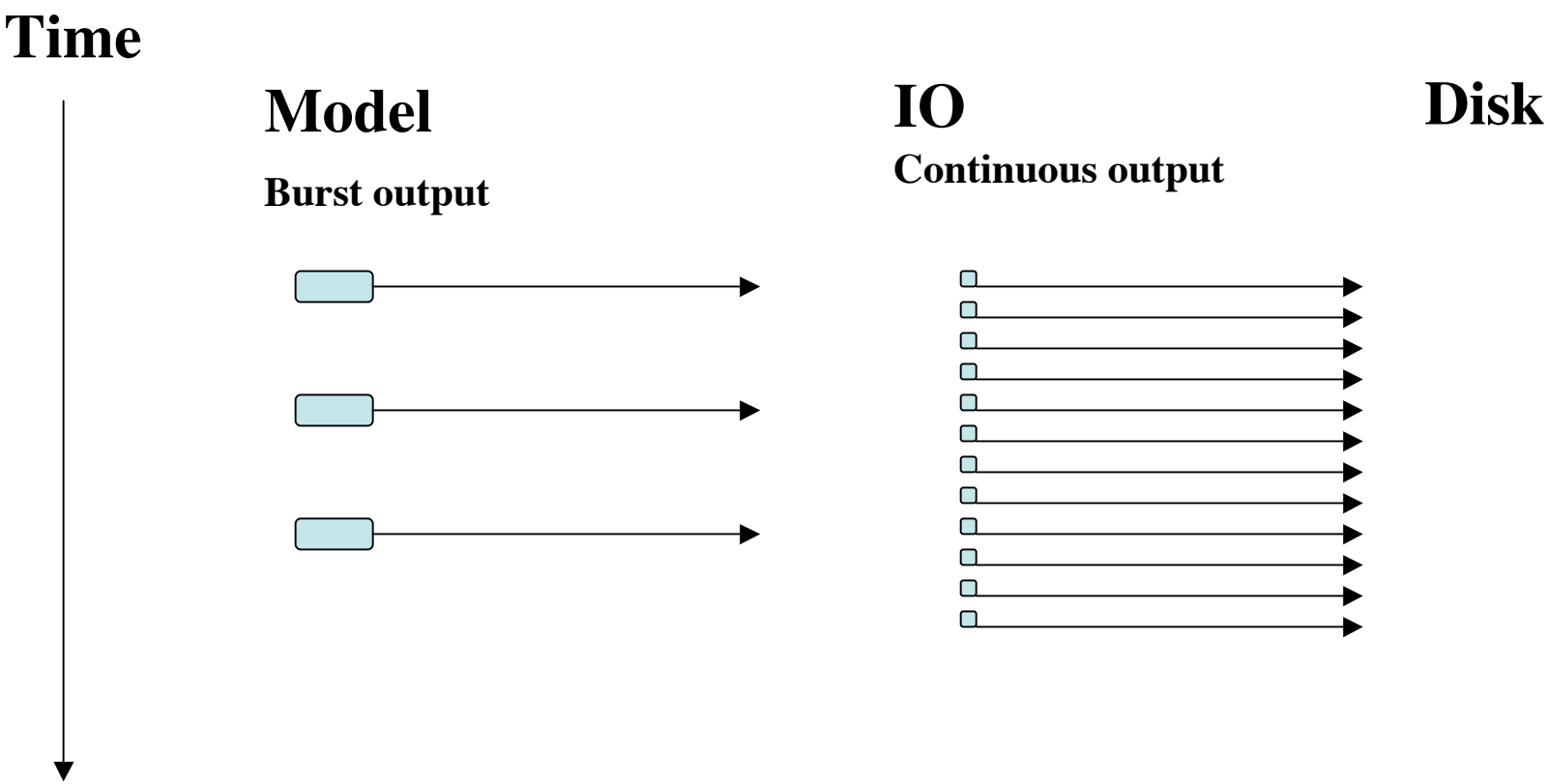


Next Step

- Use M-by-N approach to transfer data
  - Use Round-robin method
    - For M=14 and N=4, then m0,m1,m2,m3 go to n0, m4,m5,m6,m7 go to n1, m8,m9,m10,m11 go to n2, m12,m13 go to n3
- Use MPI-IO, HDF5, pNetCDF for parallel writing to discs
- Use double-buffer scheme to speed up transferring data from compute node to IO node



Optimize



Polling in MPI

- Polling: Periodically probe the system to see whether a request has arrived
  - If a message arrived, the message is received and responded to; otherwise the computation is resumed. That is no resource is wasted in waiting
- Blocking polling:
  - Int MPI_Probe(int source, int tag, MPI_Comm comm, MPI_Status *status)
- Nonblock polling:
  - Int MPI_Iprobe(int source, int tag, MPI_Comm comm, int *flag, MPI_Status *status)

Code for pulling in Fortran

```
do while (iEndPoll /= POLL_DONE)

      ! Use MPI_Tag to separate the type of message
      ! such as integer, real, or array dimension

      call MPI_Probe(MPI_Any_Source, MPI_Any_Tag, MPI_COMM_WORLD, &
      &              status, ierr)

      select case (status(MPI_Tag))
        case(TAG_SEND_REAL)

        ! Find out the length of incoming message
        call MPI_Get_Count(status, MPI_Real, inArraySize, ierr)

        ! Allocate array
        allocate(tempElement)
        tempElement = queueElement(inArraySize, 0, 0)

        call MPI_Recv(tempElement%ptr, inArraySize, MPI_Real, &
        &             0, TAG_SEND_REAL, MPI_COMM_WORLD, status, ierr)

        call insert(q, tempElement)
        nullify(tempElement)
        …
        case default
       write(*,*)' status(MPI_Tag) is ', status(MPI_Tag)
       stop
    end select
    end do
```