# MPI, Part 2

Scientific Computing Course, Part 3

SciNet

compute • calcul CANADA

# Something new: Sendrecv

A blocking send and receive built in together

Lets them happen simultaneously

Can automatically pair the sends/recvs!

dest, source does not have to be same; nor do types or size.

```c
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <mpi.h>

int main(int argc, char **argv) {

    int rank, size;          /* the usual MPI stuff */
    int ierr;
    char hearmessage[6];     /* we receive into here, and */
    char sendmessage[]="Hello"; /* send from here.*/
    int leftneighbour, rightneighbour;
    const int OURTAG=1;      /* shared tag to label messages */
    MPI_Status status;       /* receive status info */

    ierr = MPI_Init(&argc, &argv);
    ierr = MPI_Comm_size(MPI_COMM_WORLD, &size);
    ierr = MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    if (size < 2) {          /* need at least a sender, receiver */
        fprintf(stderr,"FAIL: only one task\n");
        MPI_Abort(MPI_COMM_WORLD,1);
    }

    leftneighbour = (rank-1 + size) % size;
    rightneighbour = (rank + 1) % size;

    ierr = MPI_Sendrecv(sendmessage, 6, MPI_CHAR, rightneighbour, OURTAG,
                        hearmessage, 6, MPI_CHAR, leftneighbour, OURTAG,
                        MPI_COMM_WORLD, &status);

    printf("%d: Sent message <%s> to %d \n", rank, sendmessage, rightneighbour);
    printf("%d: Recieved message <%s> from %d\n", rank, hearmessage, leftneighbour);

    MPI_Finalize();

    return 0;
}
```

fourthmessage.c

SciNet compute • calcul CANADA

# Sendrecv = Send + Recv

## C syntax

Send Args

```
MPI_Status status;

ierr = MPI_Sendrecv(sendptr, count, MPI_TYPE, destination,tag,
                    recvptr, count, MPI_TYPE, source, tag,
                    Communicator, &status);
```

Recv Args

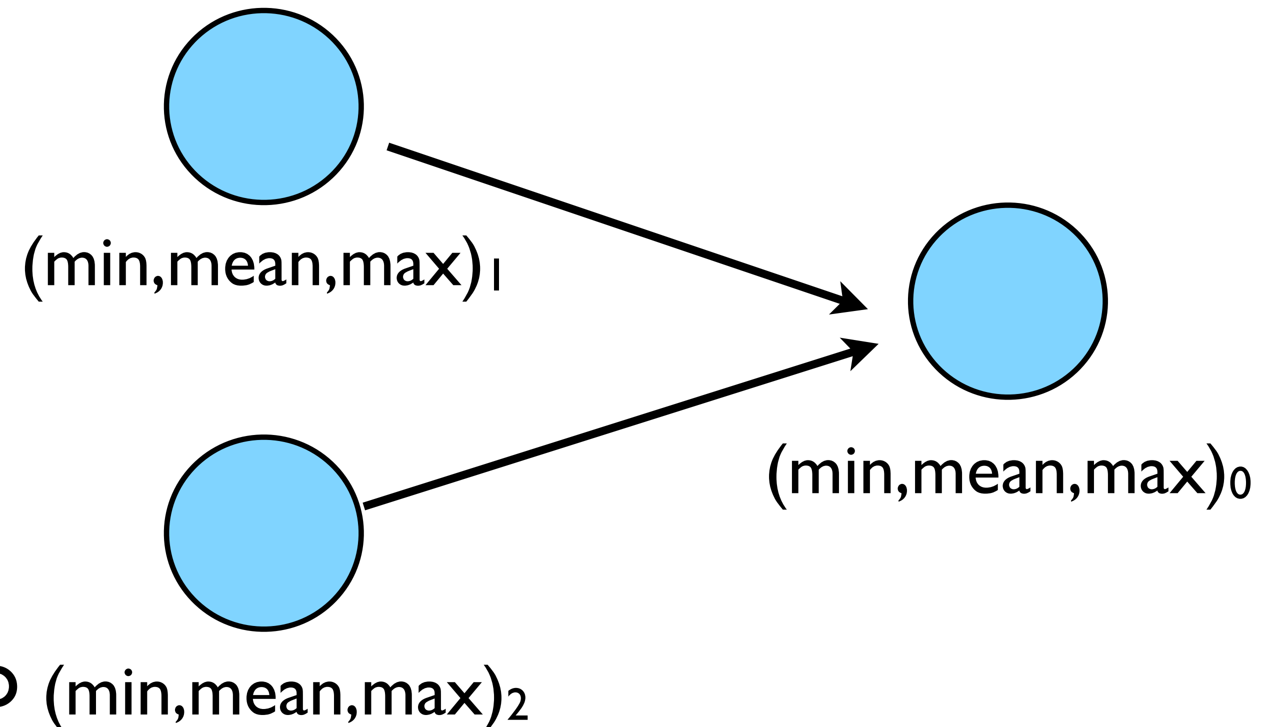
## FORTRAN syntax

```
integer status(MPI_STATUS_SIZE)

call MPI_SENDRECV(sendptr, count, MPI_TYPE, destination,tag,
                    recvptr, count, MPI_TYPE, source, tag,
                    Communicator, status, ierr)
```

Why are there two different tags/types/counts?

SciNet compute • calcul CANADA

# Min, Mean, Max of numbers




Lets try some code that calculates the min/mean/max of a bunch of random numbers -1..1. Should go to -1,0,+1 for large N.

Each gets their partial results and sends it to some node, say node 0 (why node 0?)

~ljdursi/ss2010/mpi-intro/minmeanmax.{c,f90}

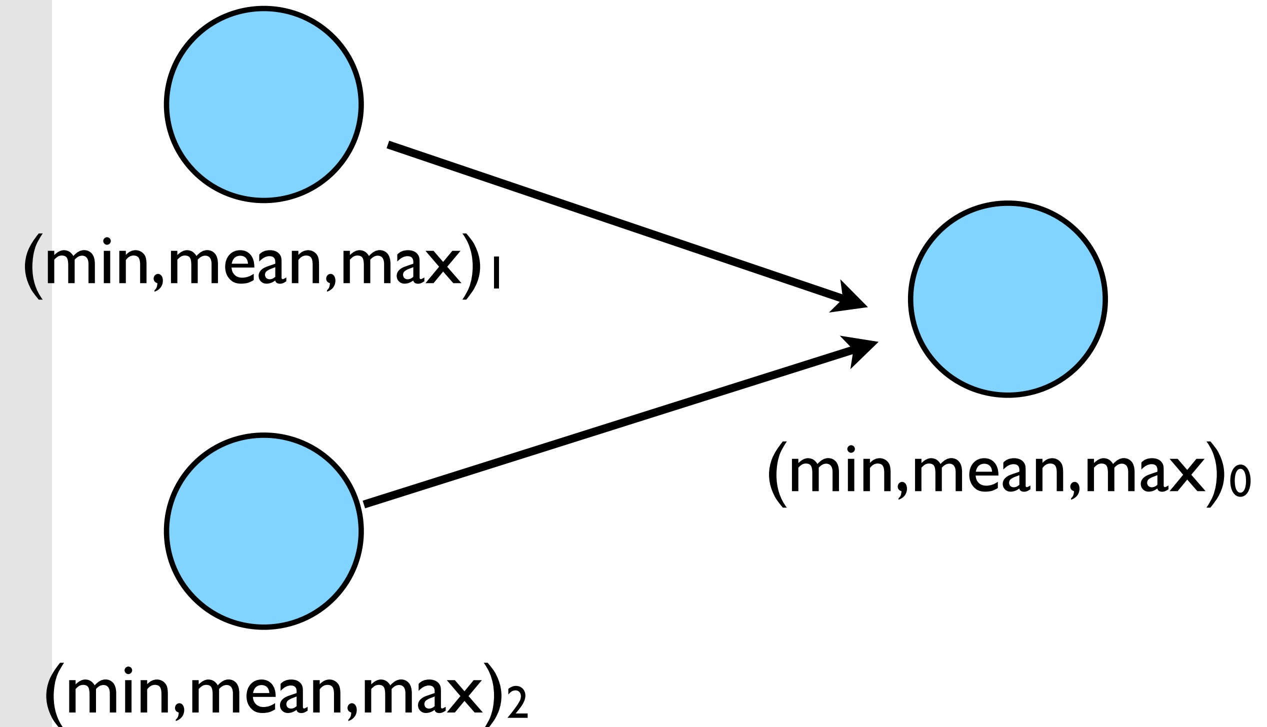
How to MPI it?

SciNet compute • calcul CANADA

```fortran
      program randomdata
      implicit none
      integer,parameter :: nx=1500
      real, allocatable :: dat(:)

      integer :: i
      real    :: datamin, datamax, datamean

!
! random data
!
      allocate(dat(nx))
      call srand(0)
      do i=1,nx
         dat(i) = 2*rand(0)-1.
      enddo

!
! find min/mean/max
!
      datamin = 1e+19
      datamax =-1e+19
      datamean = 0.

      do i=1,nx
         if (dat(i) .lt. datamin) datamin = dat(i)
         if (dat(i) .ge. datamax) datamax = dat(i)
         datamean = datamean + dat(i)
      enddo
      datamean = datamean/(1.*nx)
      deallocate(dat)

      print *,'min/mean/max = ', datamin, datamean, datamax

      return
      end
```

SciNet compute • calcul CANADA

```
33 c
34 c find min/mean/max
35 c
36       datamin = 1e+19
37       datamax =-1e+19
38       datamean = 0
39
40       do i=1,nx
41          do j=1,ny
42              if (dat(i,j) .lt. datamin) datamin = dat(i,j)
43              if (dat(i,j) .gt. datamax) datamax = dat(i,j)
44              datamean = datamean + dat(i,j)
45          enddo
46       enddo
47       datamean = datamean/(1.*nx*ny)
48
49       print *,myid,': min/mean/max = ', datamin, datamean, datamax
50 c
51 c combine data
52 c
53       if (myid .ne. 0) then
54           datapack(1) = datamin
55           datapack(2) = datamean
56           datapack(3) = datamax
57           call MPI_SSEND(datapack,3,MPI_REAL,0,1,MPI_COMM_WORLD,ierr)
58       else
59           globmin  = datamin
60           globmax  = datamax
61           globmean = datamean
62           do proc=1,nprocs-1
63               call MPI_RECV(datapack, 3, MPI_REAL, MPI_ANY_SOURCE, 1,
64      &                      MPI_COMM_WORLD, status, ierr)
65               if (datapack(1) .lt. globmin) globmin=datapack(1)
66               globmean = globmean + datapack(2)
67               if (datapack(3) .gt. globmax) globmax=datapack(3)
68           enddo
69           globmean = globmean/nprocs
70           print *,'Global min/mean/max=',globmin,globmean,globmax
71       endif
72
73       call MPI_FINALIZE(ierr)
74       return
75       end
76
77
```

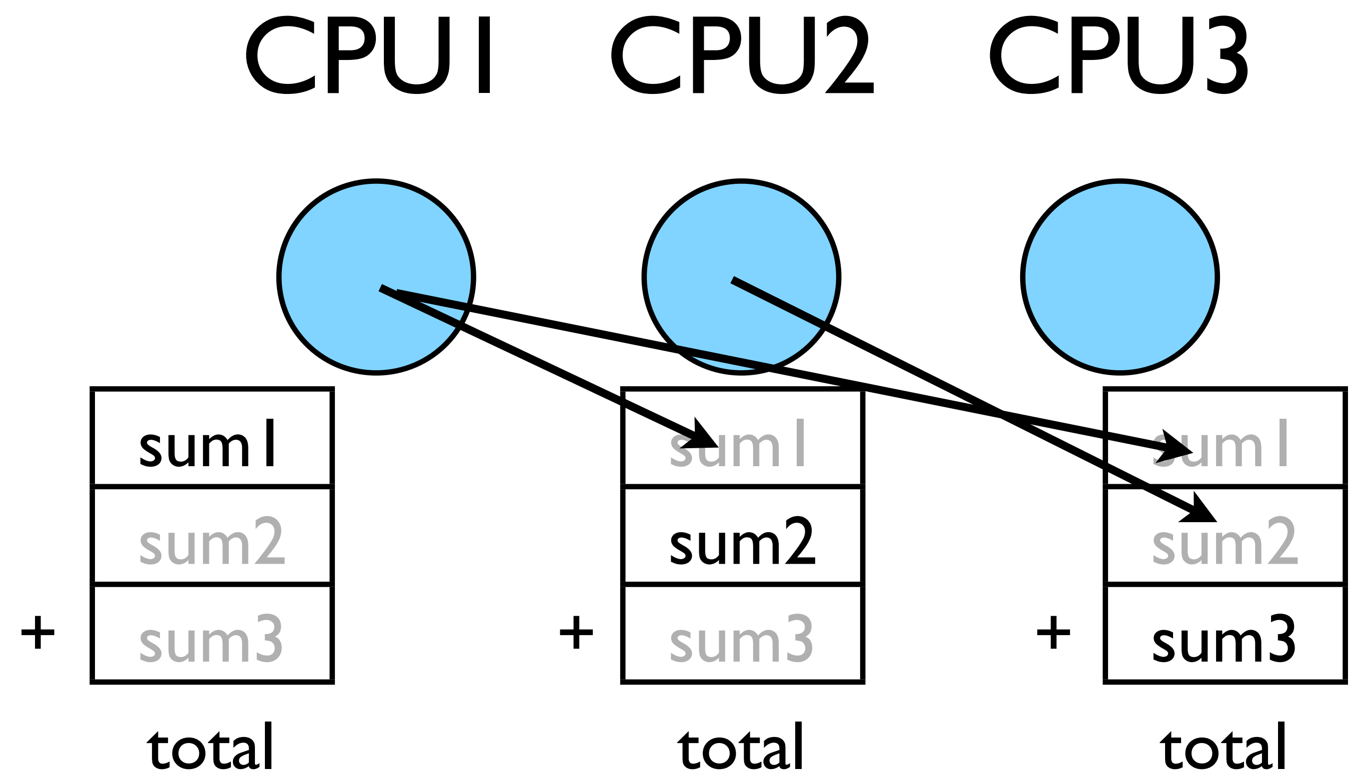



Q: are these sends/recvd adequately paired?

minmeanmax-mpi.f

SciNet compute • calcul CANADA

# Inefficient!




Requires (P-1) messages, 2(P-1) if everyone then needs to get the answer.

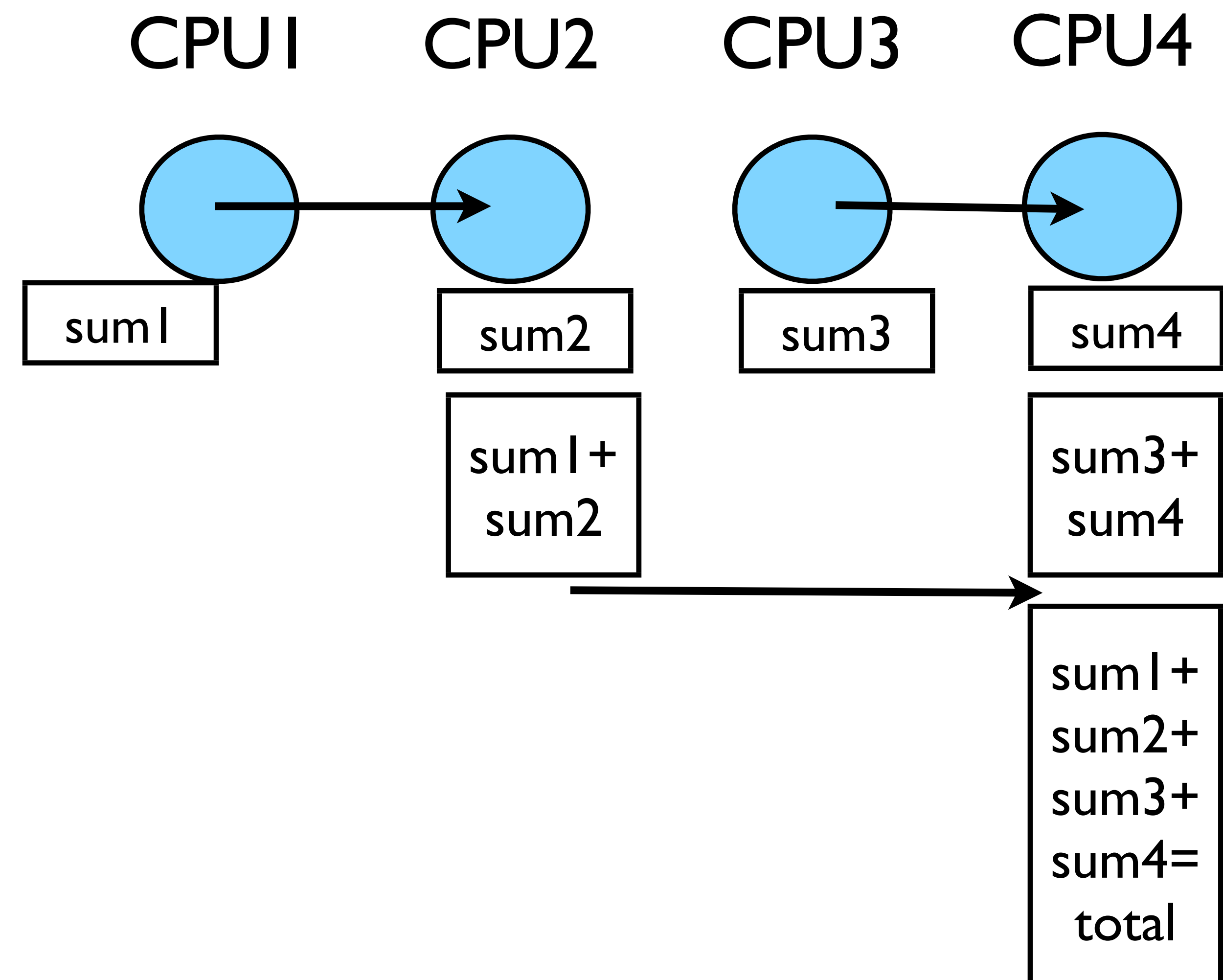
# Better Summing

Pairs of processors; send partial sums

Max messages received $\log_2(P)$

Can repeat to send total back

$$T_{\text{comm}} = 2\log_2(P)C_{\text{comm}}$$




Reduction; works for a variety of operators (+,*,min,max...)

SciNet compute • calcul CANADA

# MPI_Reduce and MPI_Allreduce

```
c
c find min/mean/max
c
      datamin = 1e+19
      datamax =-1e+19
      datamean = 0

      do i=1,nx
         do j=1,ny
            if (dat(i,j) .lt. datamin) datamin = dat(i,j)
            if (dat(i,j) .gt. datamax) datamax = dat(i,j)
            datamean = datamean + dat(i,j)
         enddo
      enddo
      datamean = datamean/(1.*nx*ny)

      print *,myid,': min/mean/max = ', datamin, datamean, datamax
c
c combine data
c
      call MPI_ALLREDUCE(datamin, globmin, 1, MPI_REAL, MPI_MIN,
     &                   MPI_COMM_WORLD, ierr)
c
c to just send to task 0:
c       call MPI_REDUCE(datamin, globmin, 1, MPI_REAL, MPI_MIN,
c     &                   0, MPI_COMM_WORLD, ierr)
c       etc.
c
      call MPI_ALLREDUCE(datamax, globmax, 1, MPI_REAL, MPI_MAX,
     &                   MPI_COMM_WORLD, ierr)
      call MPI_ALLREDUCE(datamean, globmean, 1, MPI_REAL, MPI_SUM,
     &                   MPI_COMM_WORLD, ierr)
      globmean = globmean/nprocs
      print *, myid,': Global min/mean/max=',globmin,globmean,globmax

      call MPI_FINALIZE(ierr)
      return
      end
```

Performs a reduction and sends answer to one PE (Reduce) or all PEs (Allreduce)

minmeanmax-allreduce.f

SciNet compute • calcul CANADA

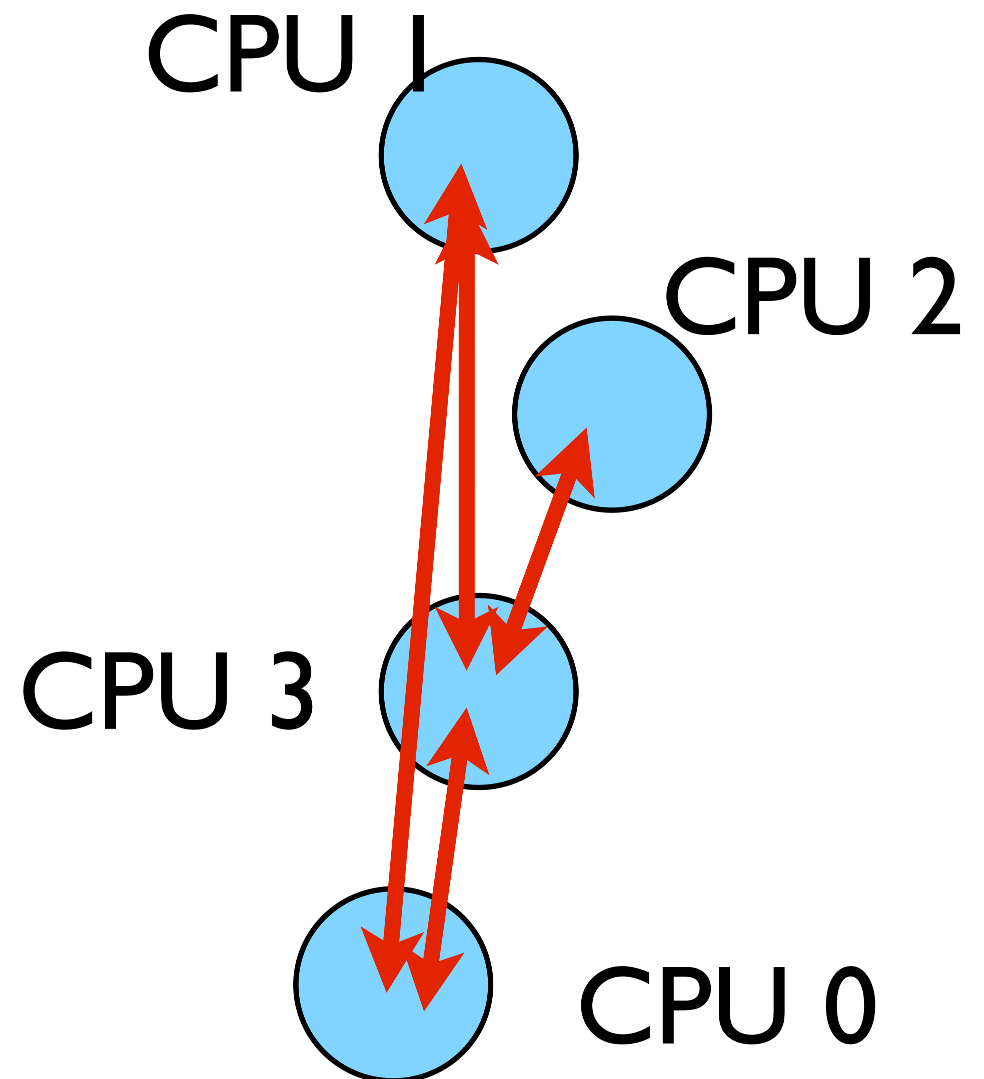
# **Collective** Operations

As opposed to the pairwise messages we've seen

**All** processes in the communicator must participate

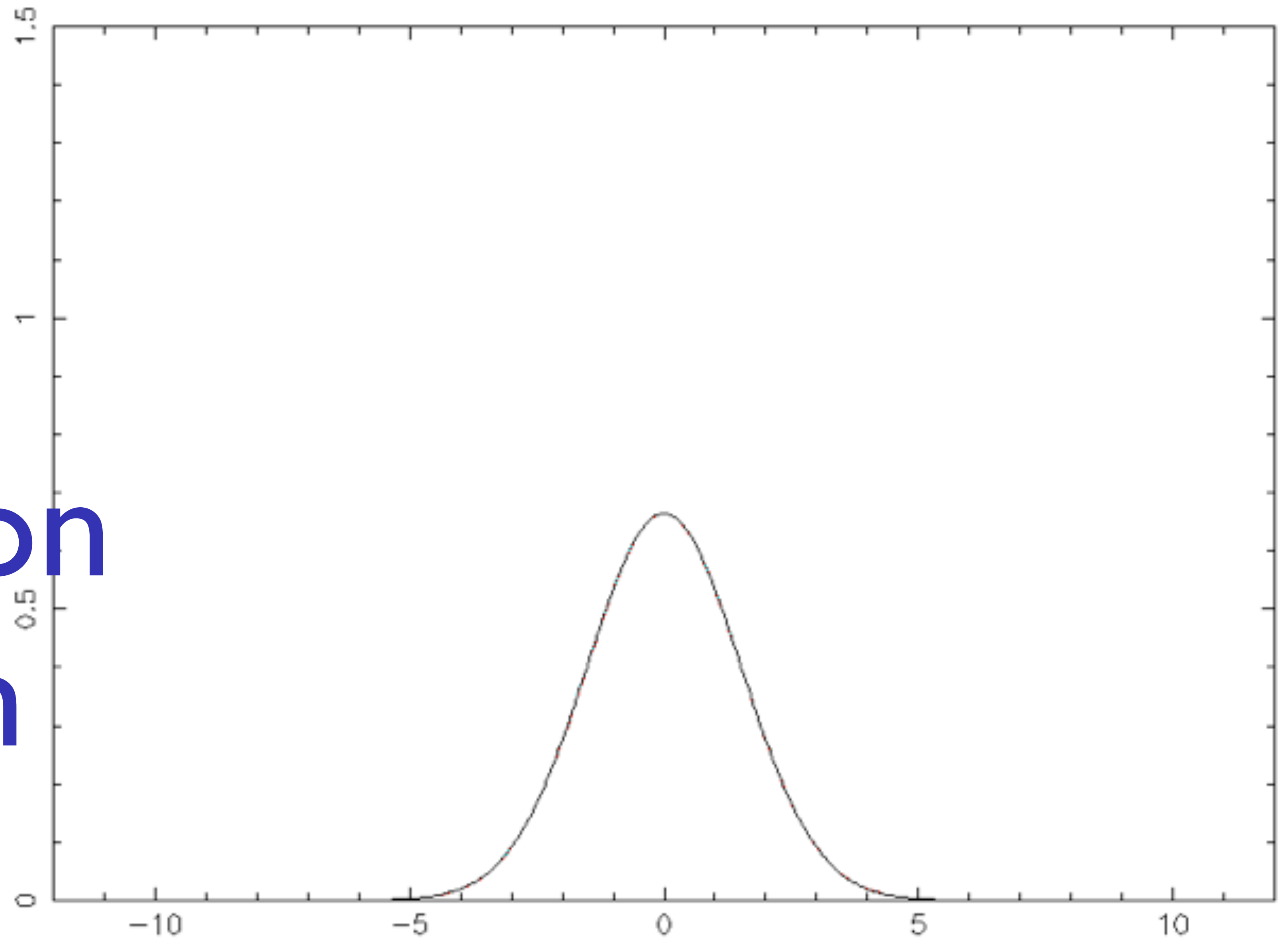
Cannot proceed until all have participated

Don't necessarily know what goes on 'under the hood'




SciNet compute • calcul CANADA

# 1d diffusion equation




```
cp -R ~ljdursi/ss2010/diffusion .
cd diffusion
make diffusionf or make diffusionc
./diffusionf or ./diffusionc
```

SciNet
compute • calcul
CANADA

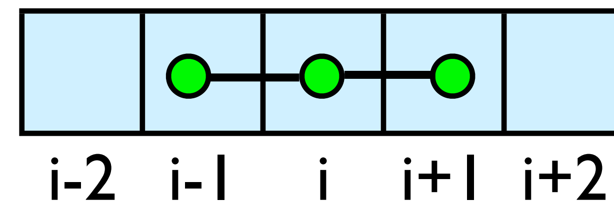
# Discretizing Derivatives

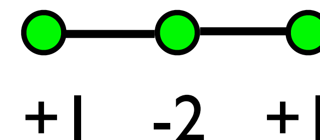
$$\left.\frac{d^2Q}{dx^2}\right|_i \approx \frac{Q_{i+1} - 2Q_i + Q_{i-1}}{\Delta x^2}$$

Done by finite differencing the discretized values

Implicitly or explicitly involves interpolating data and taking derivative of the interpolant

More accuracy - larger ‘stencils’



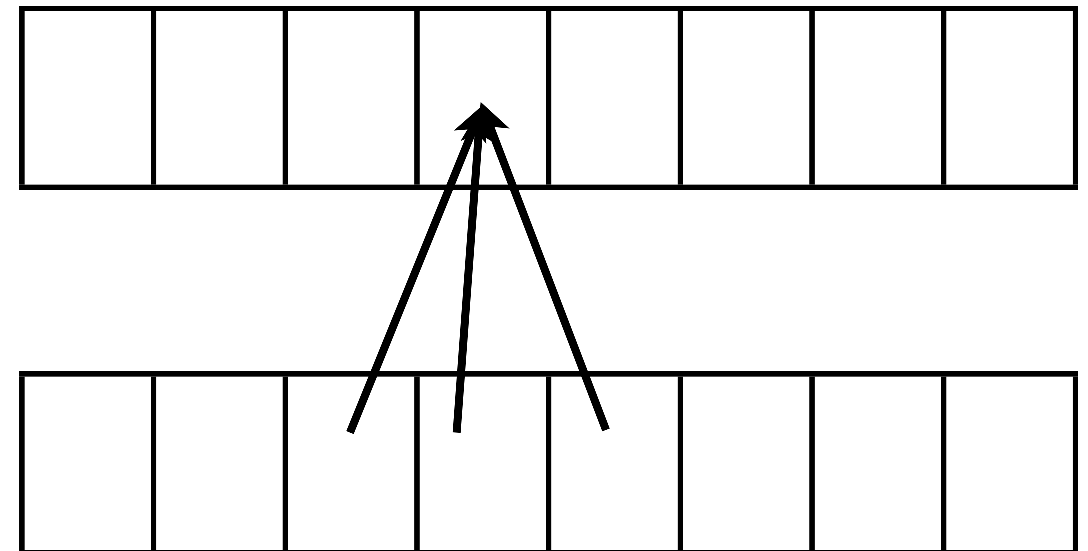

SciNet compute • calcul CANADA

# Diffusion Equation

$$
\begin{aligned}
\frac{\partial T}{\partial t} &= D\frac{\partial^2 T}{\partial x^2} \\
\frac{\partial T_i^{(n)}}{\partial t} &\approx \frac{T_i^{(n)} + T_i^{(n-1)}}{\Delta t} \\
\frac{\partial T_i^{(n)}}{\partial x} &\approx \frac{T_{i+1}^{(n)} - 2T_i^{(n)} + T_{i-1}^{(n)}}{\Delta x^2} \\
T_i^{(n+1)} &\approx T_i^{(n)} + \frac{D\Delta t}{\Delta x^2}\left(T_{i+1}^{(n)} - 2T_i^{(n)} + T_{i-1}^{(n)}\right)
\end{aligned}
$$

Simple 1d PDE

Each timestep, new data for T[i] requires old data for T[i+1],T[i],T[i-1]



SciNet compute • calcul CANADA

# Guardcells

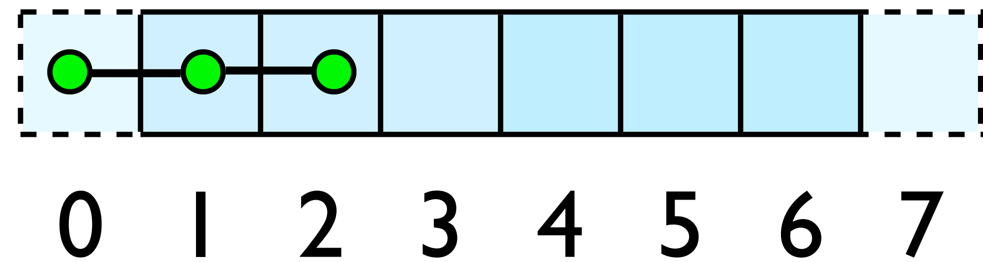
How to deal with boundaries?

Because stencil juts out, need information on cells beyond those you are updating

Pad domain with ‘guard cells’ so that stencil works even for the first point in domain

Fill guard cells with values such that the required boundary conditions are met

Global Domain




ng = 1

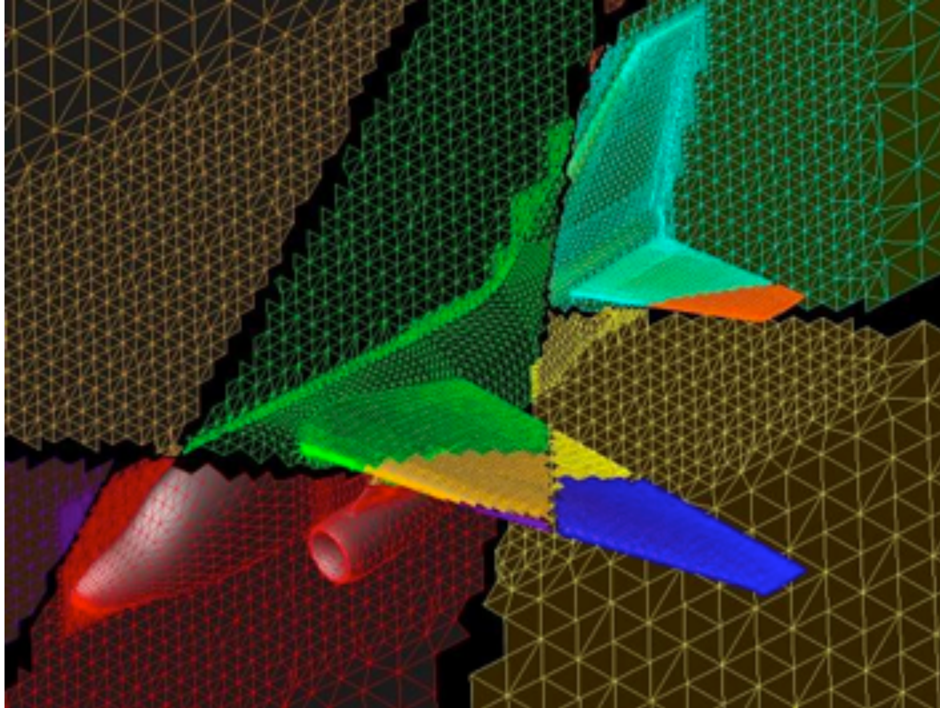
loop from ng, N - 2 ng

SciNet compute • calcul CANADA

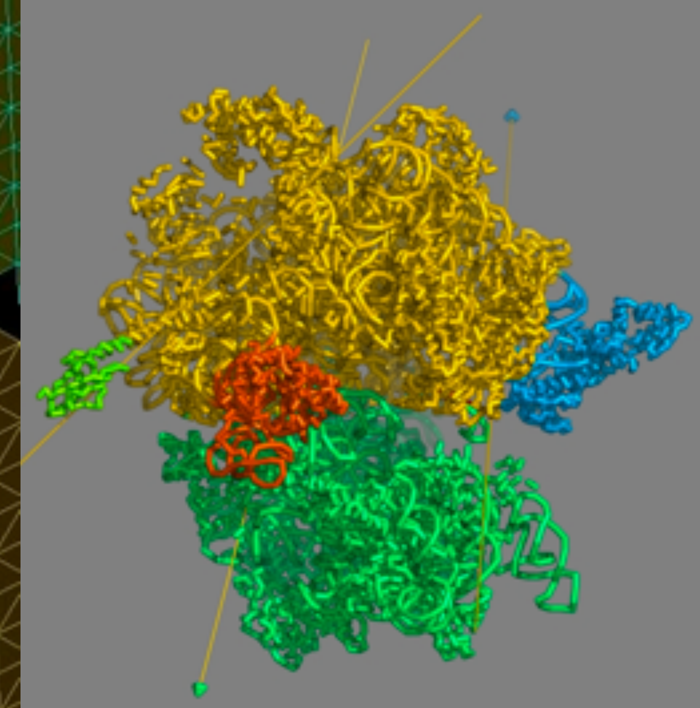
# Domain Decomposition

A very common approach to parallelizing on distributed memory computers

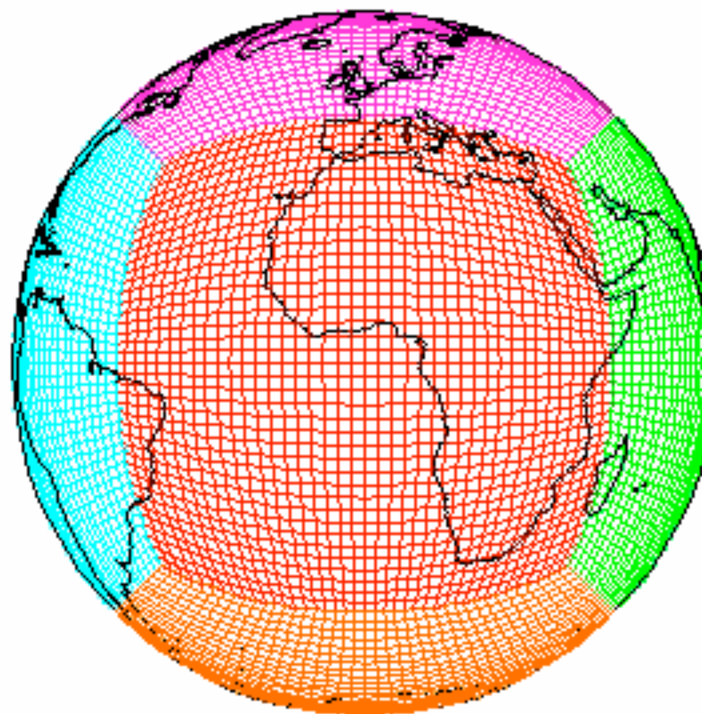
Maintain Locality; need local data mostly, this means only surface data needs to be sent between processes.



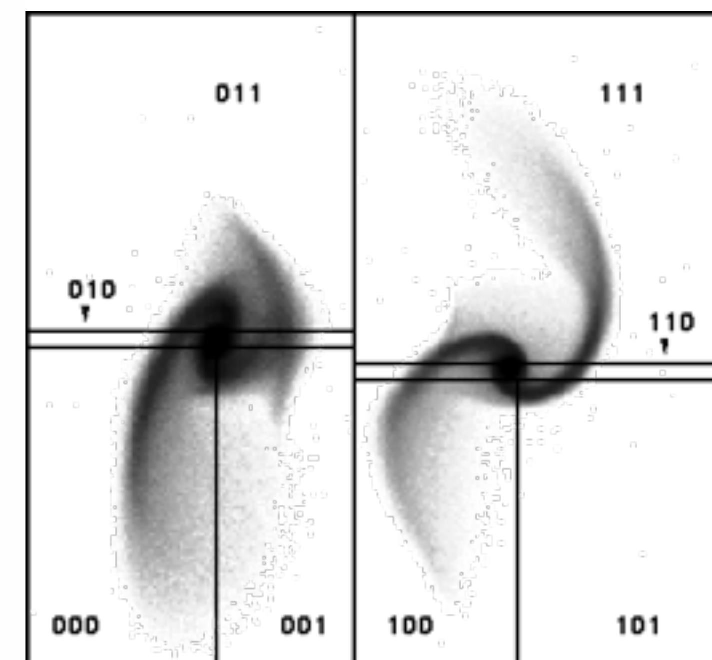
http://adg.stanford.edu/aa241/design/compaero.html



http://www.uea.ac.uk/cmp/research/cmpbio/Protein+Dynamics,+Structure+and+Function



http://sivo.gsfc.nasa.gov/cubedsphere_comp.html



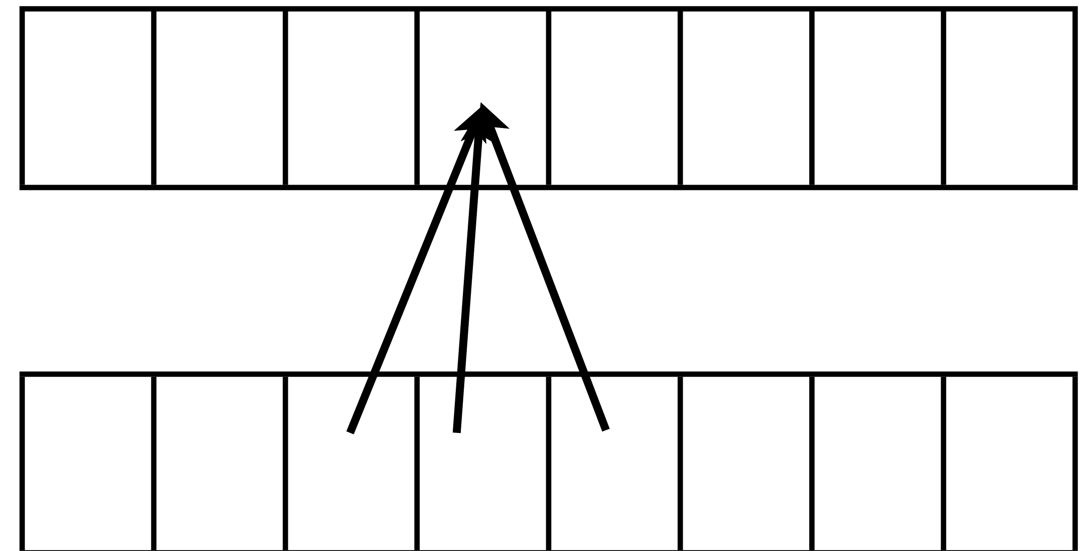

http://www.cita.utoronto.ca/~dubinski/treecode/node8.html

SciNet compute • calcul CANADA

# Implement a diffusion equation in MPI

$$
\begin{aligned}
\frac{dT}{dt} &= D\frac{d^2T}{dx^2} \\
T_i^{n+1} &= T_i^n + \frac{D\Delta t}{\Delta x^2}\left(T_{i+1}^n - 2T_i^n + T_{i-1}^n\right)
\end{aligned}
$$

Need one neighboring number per neighbor per timestep



SciNet
compute • calcul
CANADA

# Guardcells

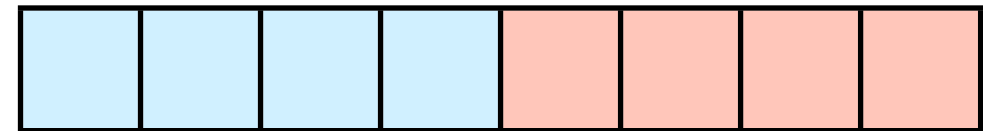
Works for parallel decomposition!

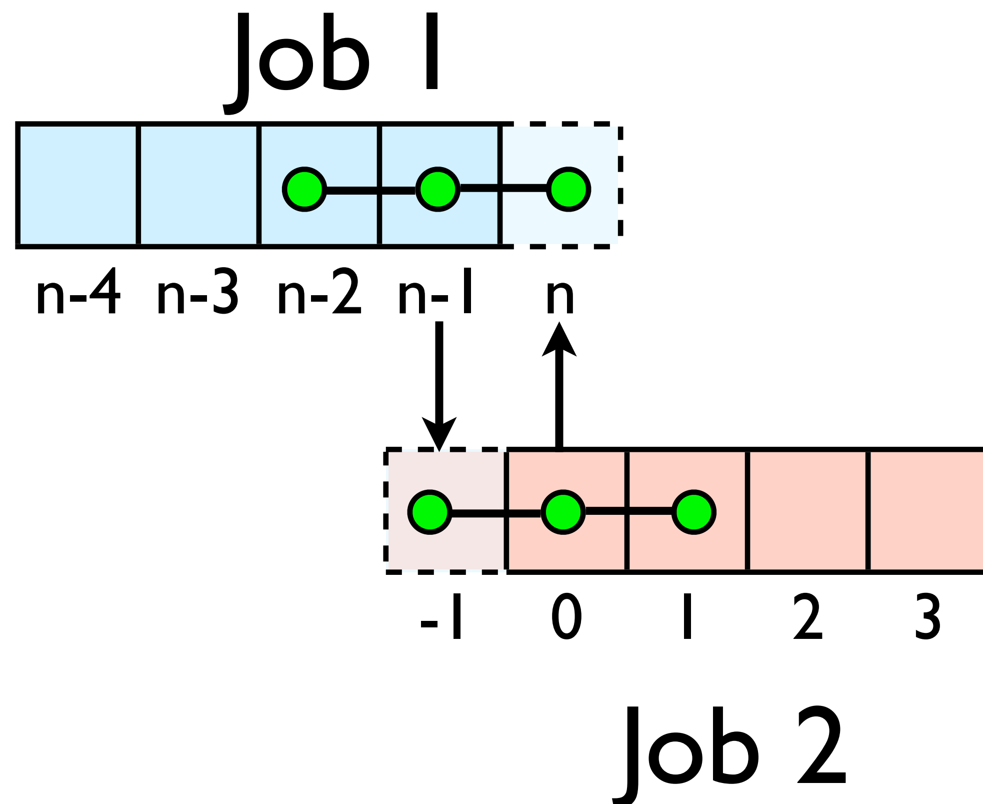
Job 1 needs info on Job 2s 0th zone, Job 2 needs info on Job 1s last zone

Pad array with 'guardcells' and fill them with the info from the appropriate node by message passing or shared memory

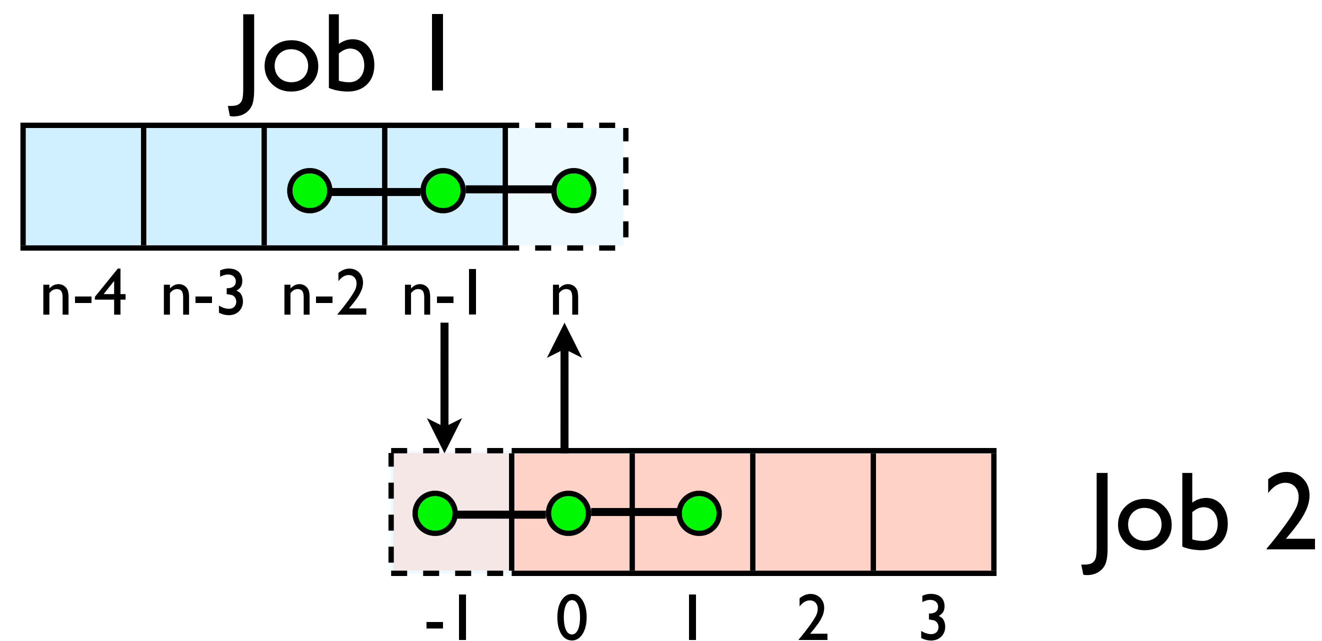
Hydro code: need guardcells 2 deep

## Global Domain






SciNet compute • calcul CANADA




Do computation
guardcell exchange: each cell has to do 2 sendrecvs
its rightmost cell with neighbors leftmost
its leftmost cell with neighbors rightmost
Use even/odd trick from before; if even do rightmost
For simplicity, fixed-temperature BCs; temperature in first, last zones are fixed

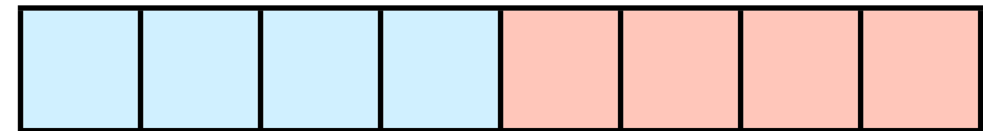
SciNet
compute • calcul
CANADA

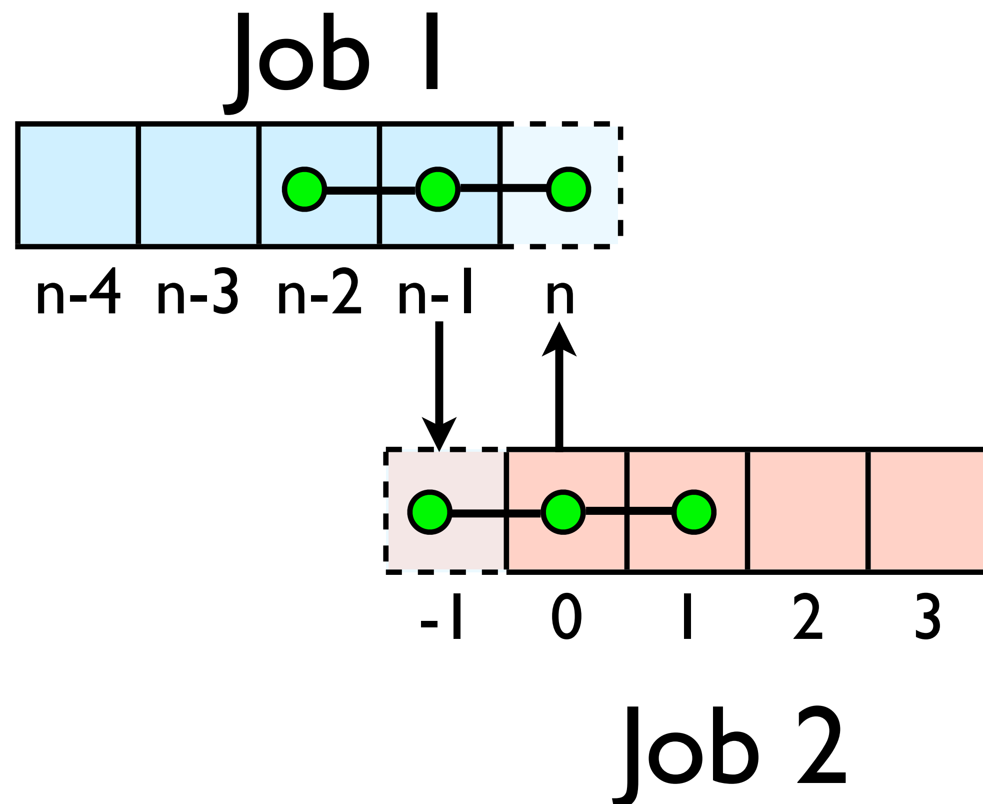
# Non-blocking communications

# Diffusion: Had to wait for communications to compute

- Could not compute end points without guardcell data
- All work halted while all communications occurred
- Significant parallel overhead

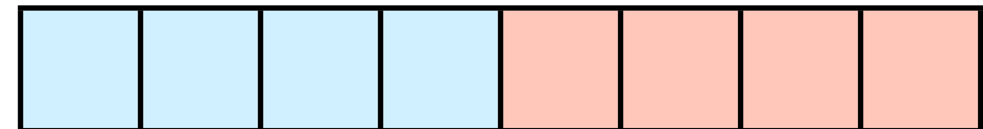
Global Domain

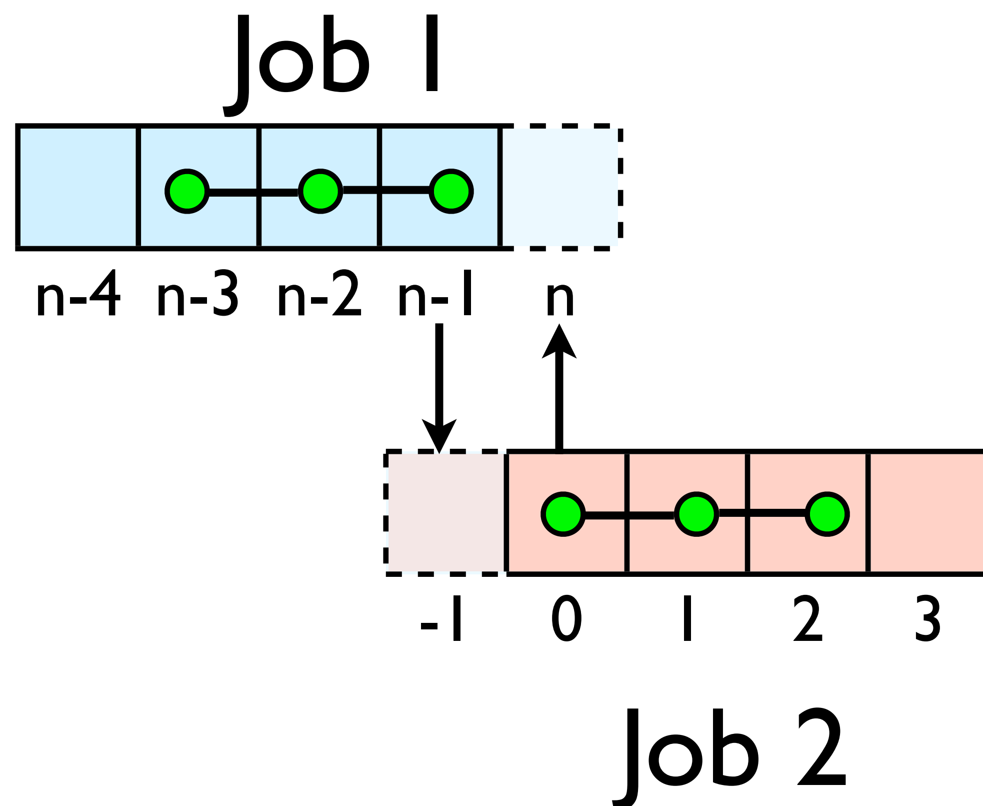




# Diffusion: *Had* to wait?

Global Domain

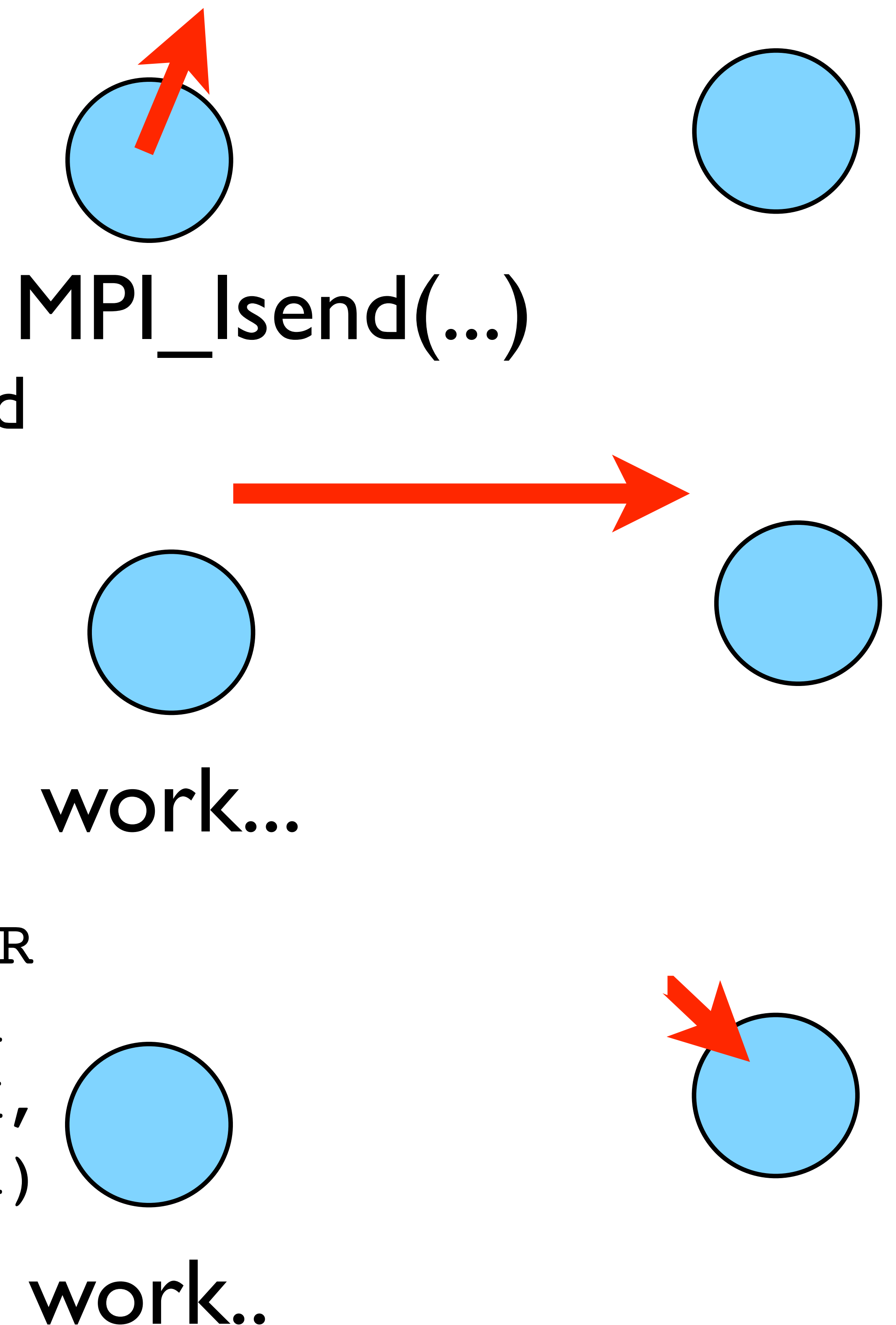





- But inner zones could have been computed just fine
- Ideally, would do inner zones work while communications is being done; then go back and do end points.

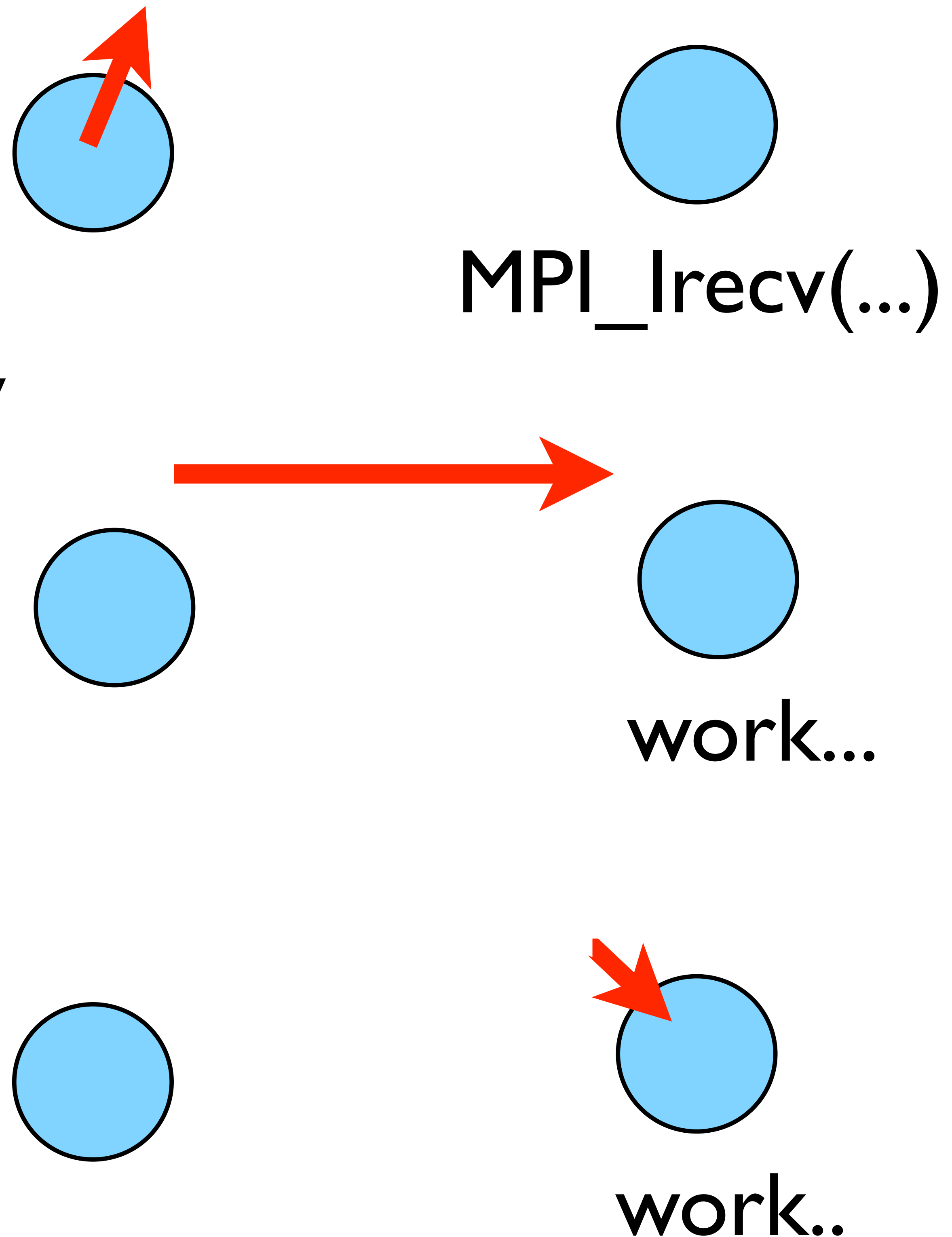
# Nonblocking Sends

- Allows you to get work done while message is 'in flight'
- Must **not** alter send buffer until send has completed.
- C: `MPI_Isend( void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm, MPI_Request *request )`
- FORTRAN: `MPI_ISEND(BUF,INTEGER COUNT,INTEGER DATATYPE,INTEGER DEST,INTEGER TAG, INTEGER COMM, INTEGER REQUEST,INTEGER IERROR)`



# Nonblocking Recv

- Allows you to get work done while message is 'in flight'
- Must **not** access recv buffer until recv has completed.
- C: `MPI_Irecv( void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm,` **`MPI_Request *request`** `)`
- FORTRAN: `MPI_IREV(BUF,INTEGER COUNT,INTEGER DATATYPE,INTEGER SOURCE,INTEGER TAG, INTEGER COMM,` **`INTEGER REQUEST`**`,INTEGER IERROR)`



# How to tell if message is completed?

- `int MPI_Wait(MPI_Request *request,MPI_Status *status);`
- `MPI_WAIT(INTEGER REQUEST,INTEGER STATUS(MPI_STATUS_SIZE),INTEGER IERROR)`

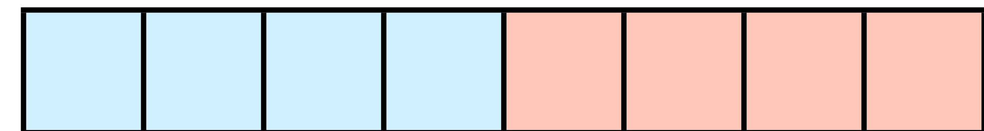
- `int MPI_Waitall(int count,MPI_Request *array_of_requests, MPI_Status *array_of_statuses);`
- `MPI_WAITALL(INTEGER COUNT,INTEGER ARRAY_OF_REQUESTS(*),INTEGER ARRAY_OF_STATUSES(MPI_STATUS_SIZE,*),INTEGER`

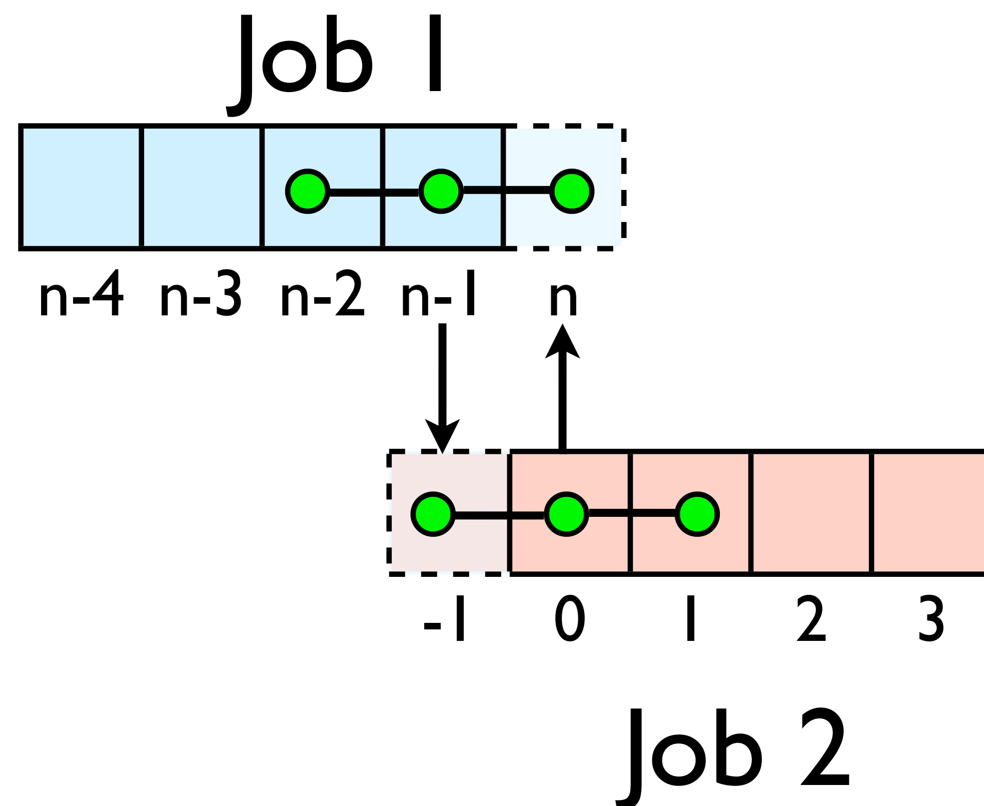
Also: MPI_Waitany, MPI_Test...

# Guardcells

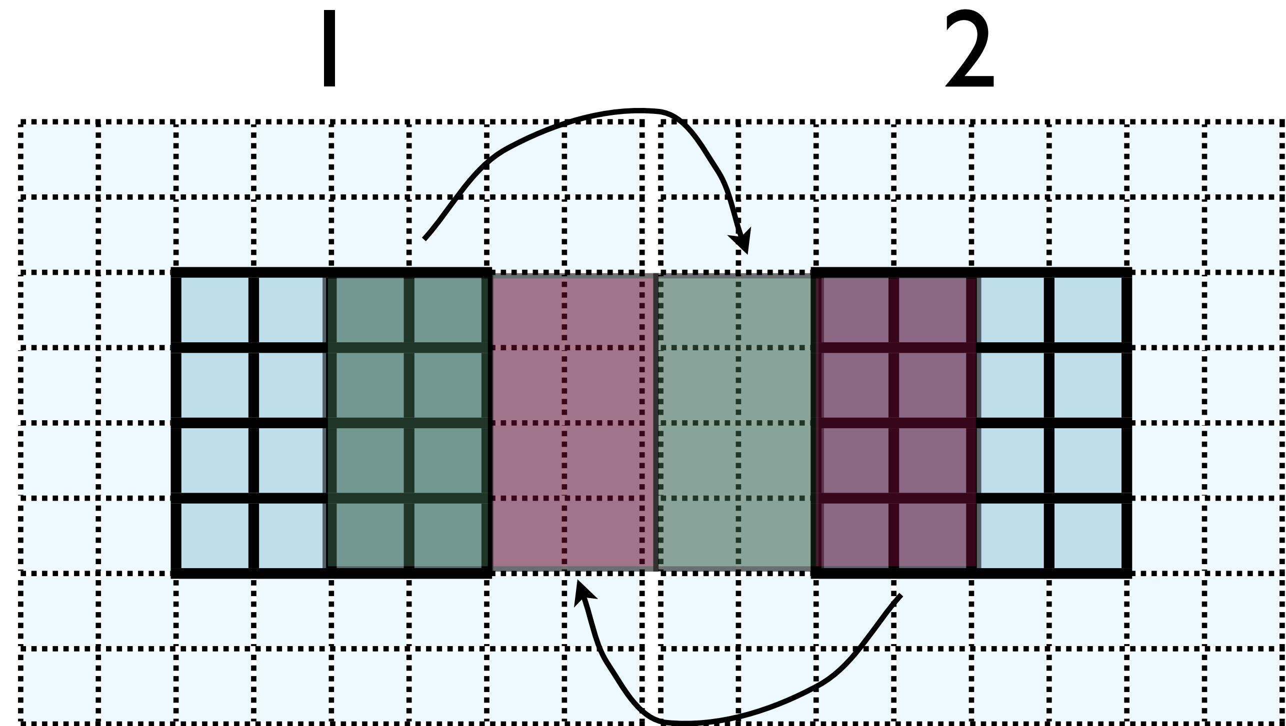
- Works for parallel decomposition!
- Job 1 needs info on Job 2s 0th zone, Job 2 needs info on Job 1s last zone
- Pad array with 'guardcells' and fill them with the info from the appropriate node by message passing or shared memory
- Hydro code: need guardcells 2 deep

## Global Domain






SciNet




# Guard cell fill

- When we're doing boundary conditions.
- Swap guardcells with neighbour.

1: u(:, nx:nx+ng, ng:ny-ng)
→ 2: u(:,1:ng, ng:ny-ng)

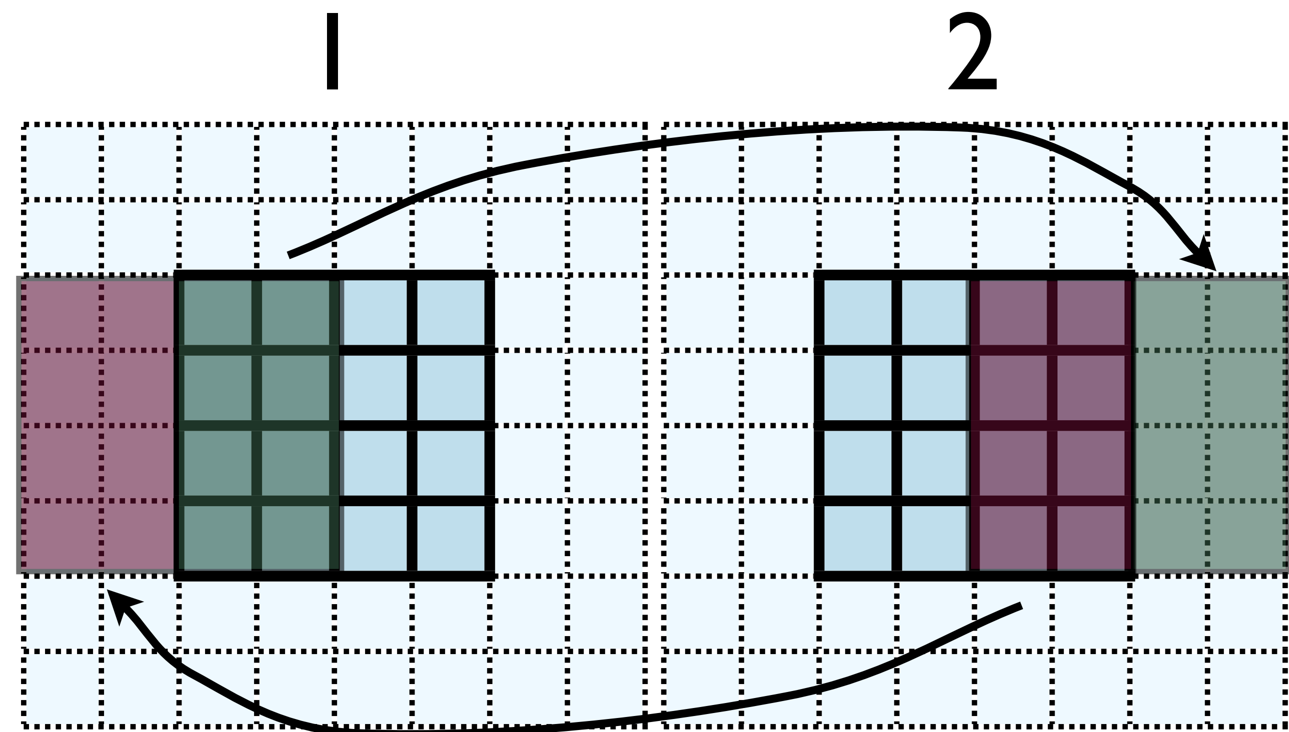
2: u(:, ng+1:2*ng, ng:ny-ng)
→ 1: u(:, nx+ng+1:nx+2*ng, ng:ny-ng)

(ny-2*ng)*ng values to swap

SciNet

# Cute way for Periodic BCs

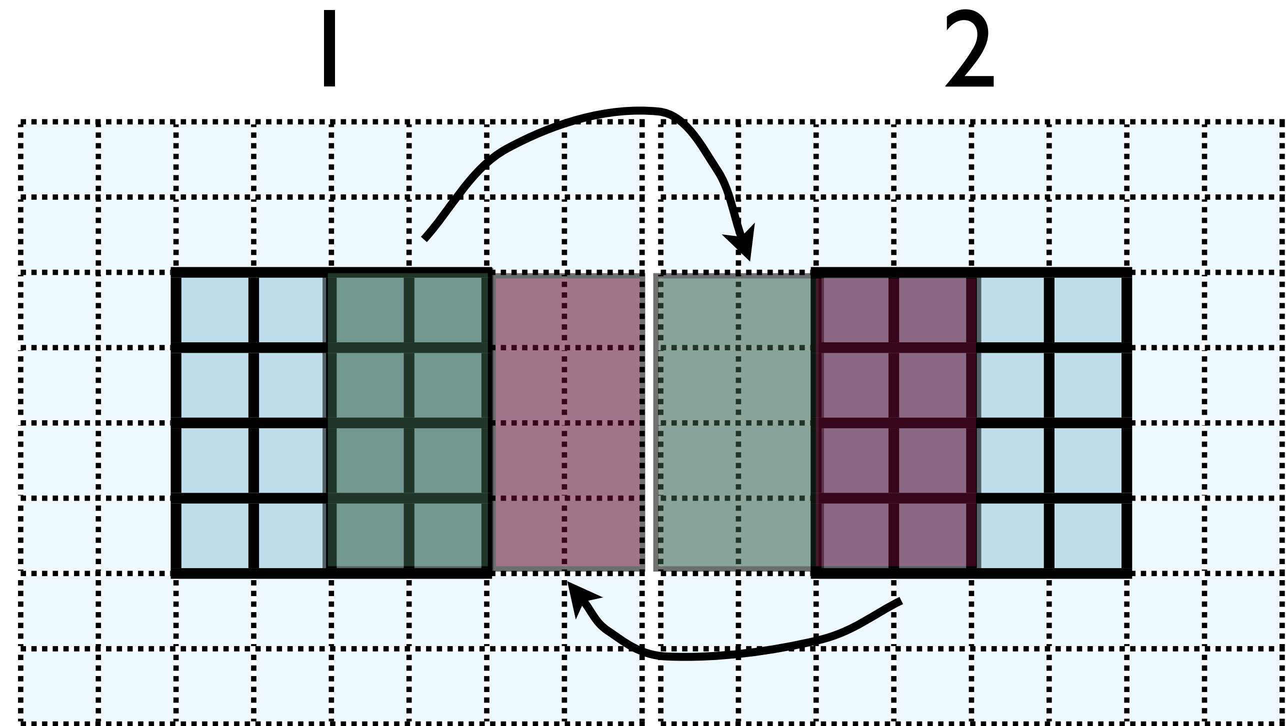
- Actually make the decomposed mesh periodic;
- Make the far ends of the mesh neighbors
- Don’t know the difference between that and any other neighboring grid
- Cart_create sets this up for us automatically upon request.




SciNet

# Implementing in MPI

- No different in principle than diffusion
- Just more values
- And more variables: dens, ener, imomx....
- Simplest way: copy all the variables into an NVARS*(ny-2*ng)*ng sized




1: u(:, nx:nx+ng, ng:ny-ng)
→ 2: u(:,1:ng, ng:ny-ng)

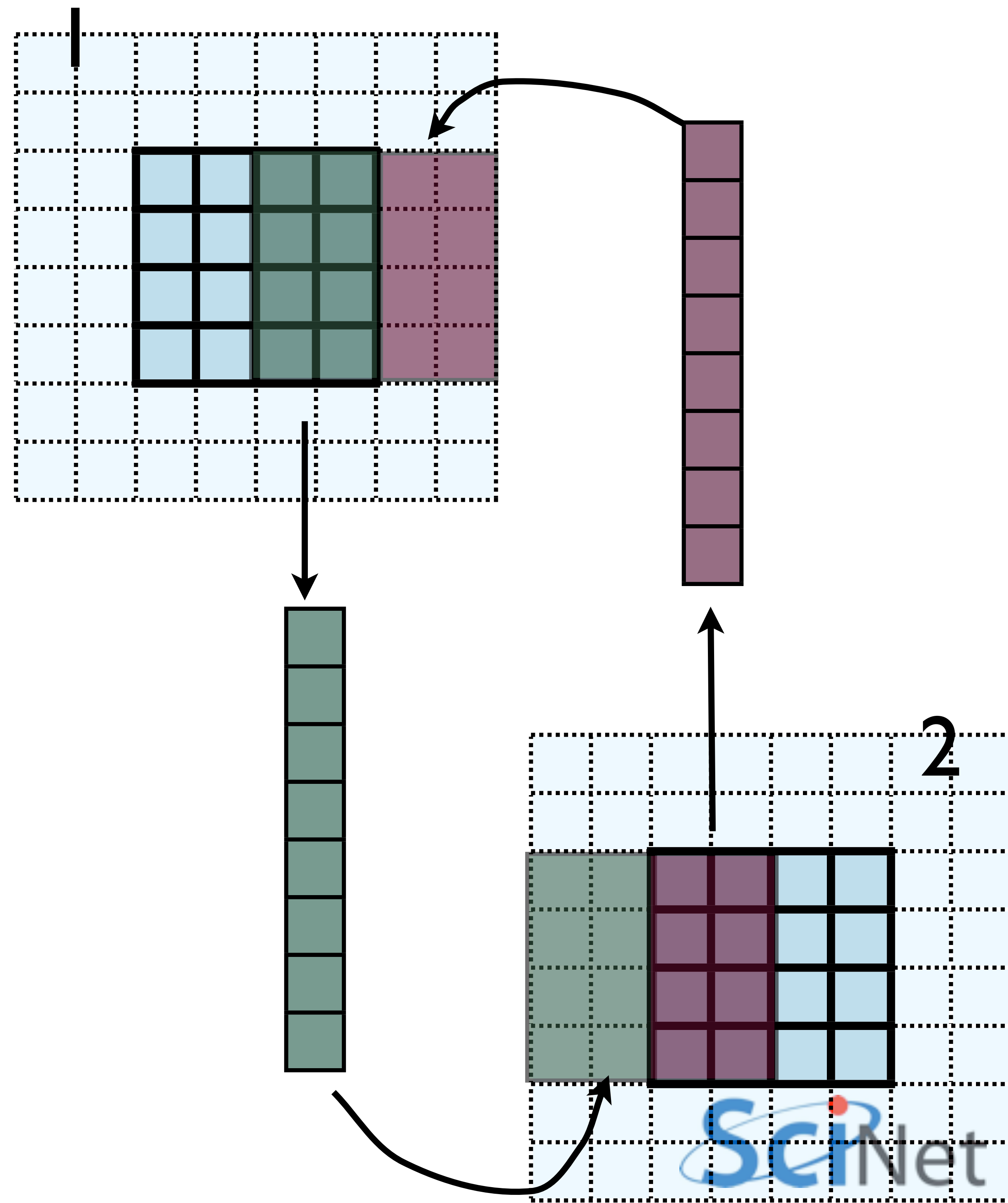
2: u(:, ng+1:2*ng, ng:ny-ng)
→ 1: u(:, nx+ng+1:nx+2*ng, ng:ny-ng)

nvars*(ny-2*ng)*ng values to swap

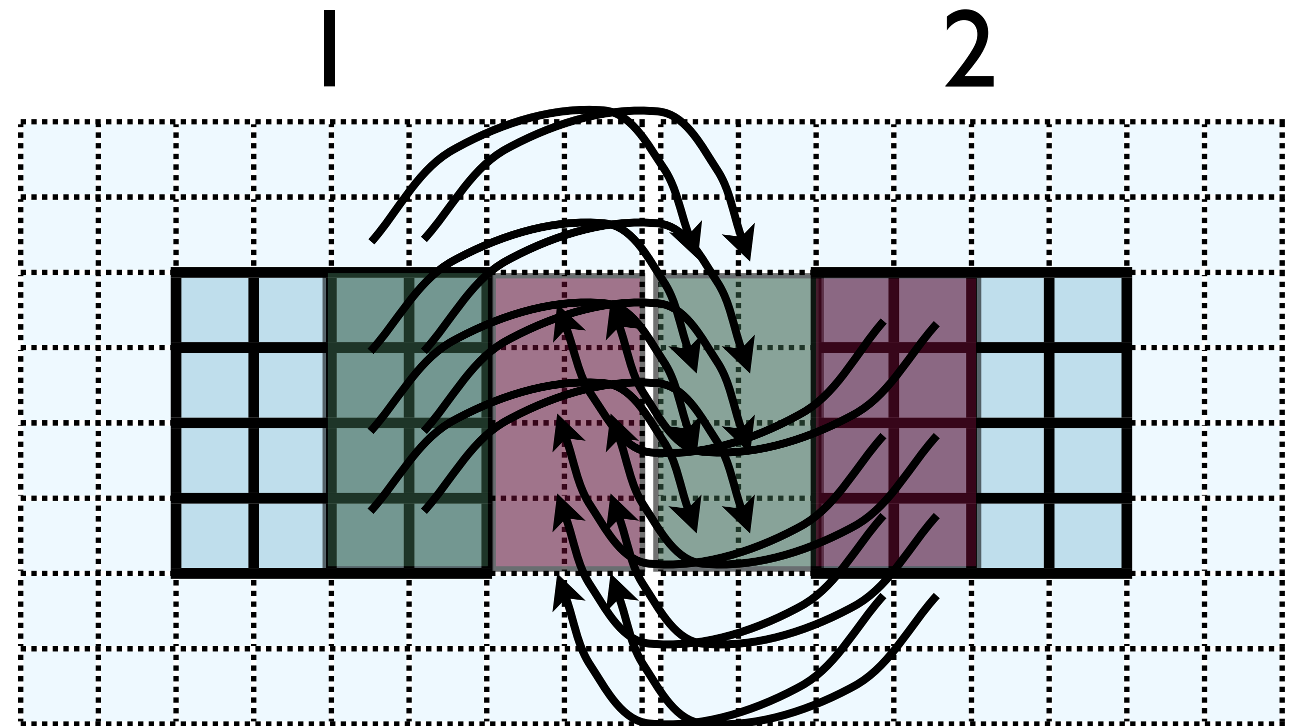
SciNet

# Implementing in MPI

- No different in principle than diffusion
- Just more values
- And more variables: dens, ener, temp....
- Simplest way: copy all the variables into an NVARS*(ny-2*ng)*ng sized



# Implementing in MPI




- Even simpler way:
- Loop over values, sending each one, rather than copying into buffer.
- NVARS*nguard*(ny-2*nguard) latency hit.
- Would completely dominate communications cost.

SciNet

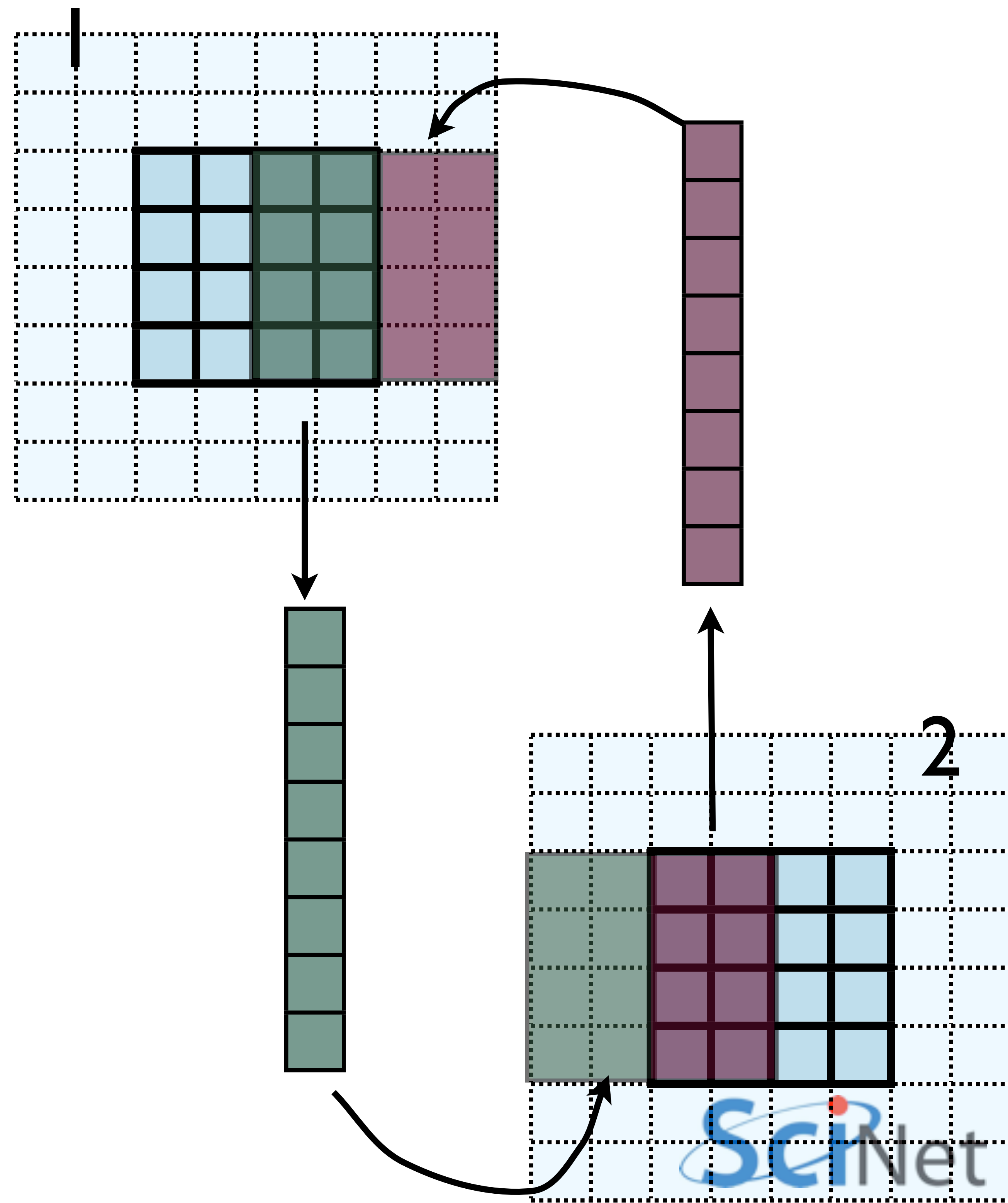
# Implementing in MPI

- Let's do this together
- solver.f90/solver.c; implement to bufferGuardcells
- When do we call this in timestep?

SciNet

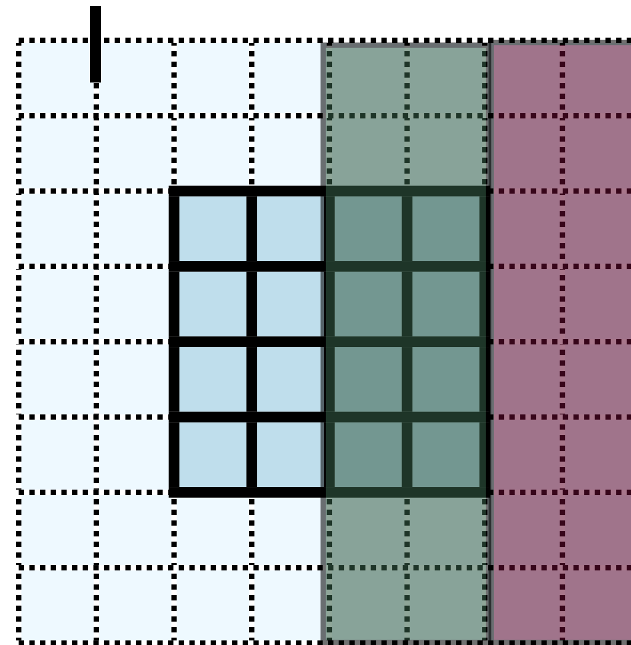
# Implementing in MPI

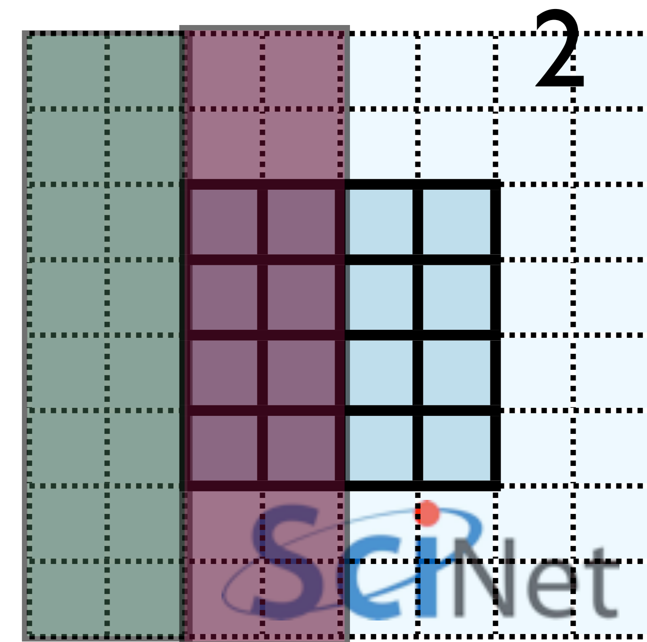
- This approach is simple, but introduces extraneous copies
- Memory bandwidth is already a bottleneck for these codes
- It would be nice to just point at the start of the guardcell data and have MPI read it from there.



# Implementing in MPI

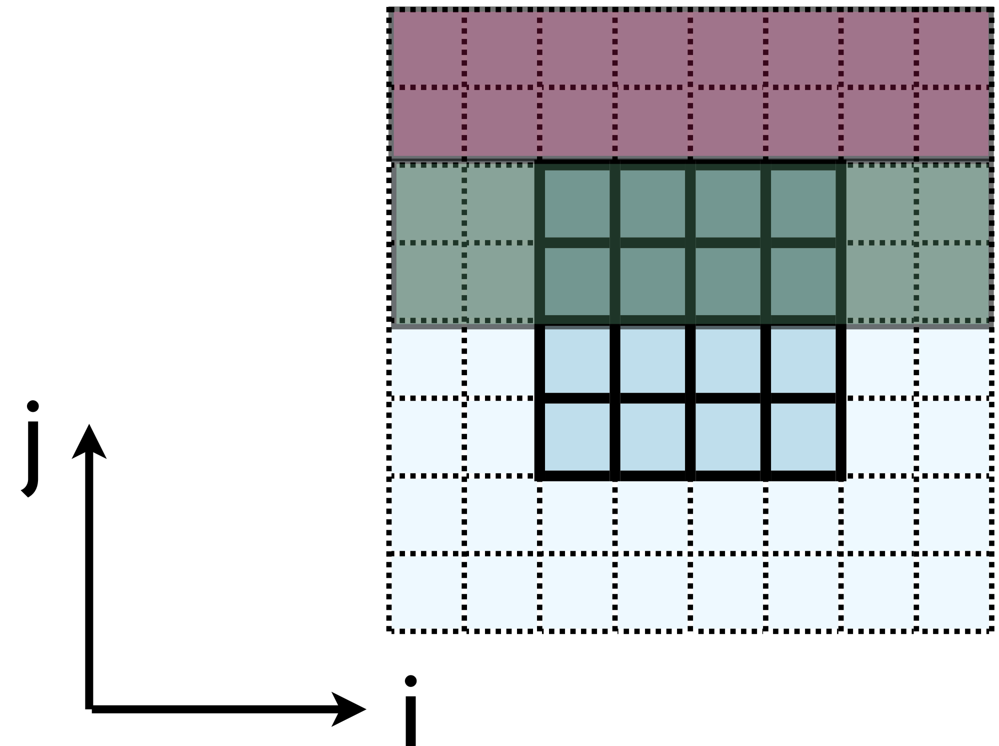
- Let me make one simplification for now; copy whole stripes
- This isn't necessary, but will make stuff simpler at first
- Only a cost of $2 \times Ng^2 = 8$ extra cells (small fraction of ~200-2000 that would normally be copied)



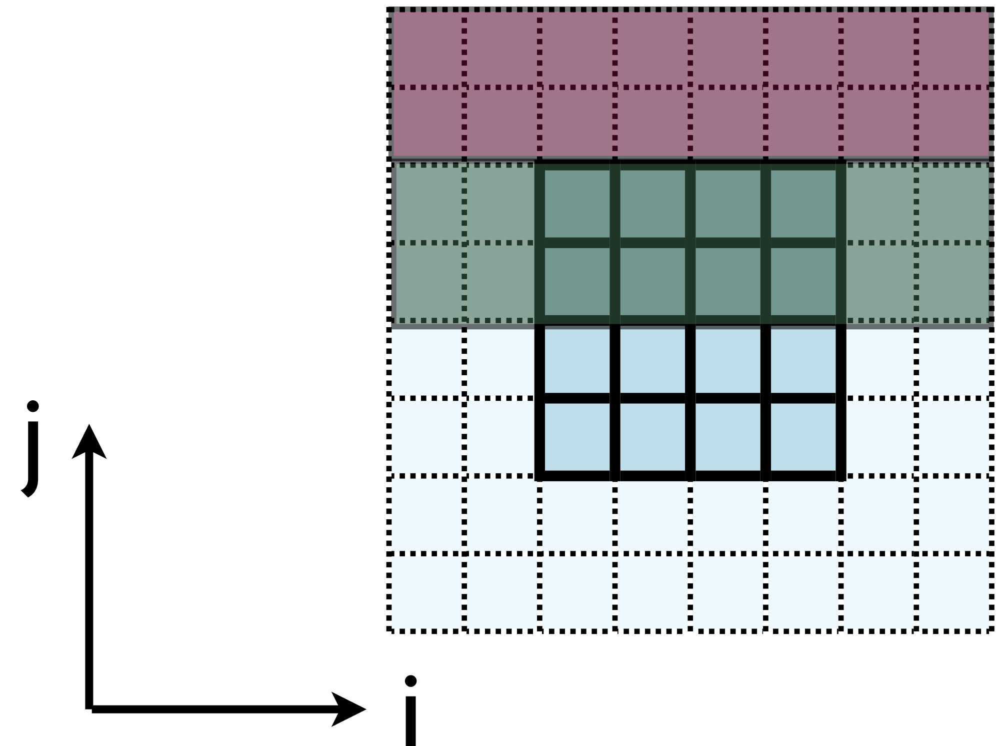
# Implementing in MPI

- Recall how 2d memory is laid out
- y-direction guardcells contiguous



# Implementing in MPI

- Can send in one go:

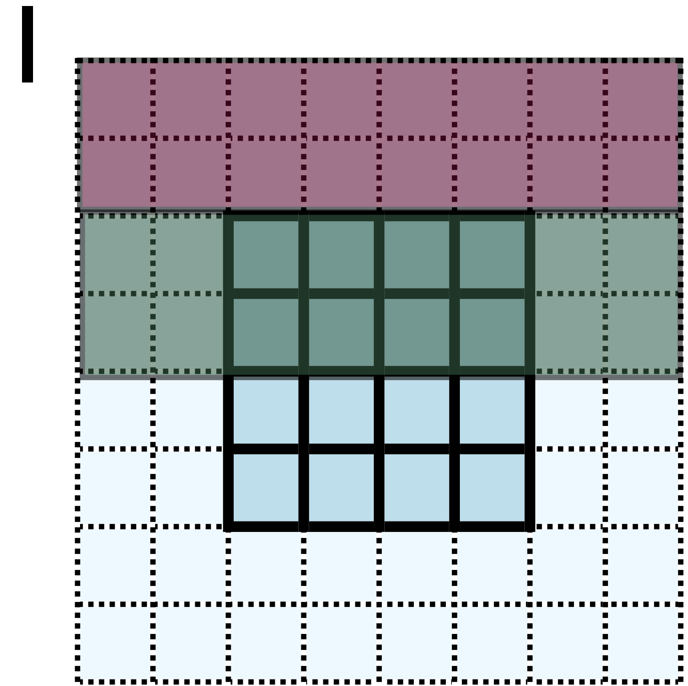



```
call MPI_Send(u(1,1,ny), nvars*nguard*ny, MPI_REAL, ....)
ierr = MPI_Send(&(u[ny][0][0]), nvars*nguard*ny, MPI_FLOAT, ....)
```

SciNet

# Implementing in MPI

- Creating MPI Data types.
- MPI_Type_contiguous: simplest case. Lets you build a string of some other type.



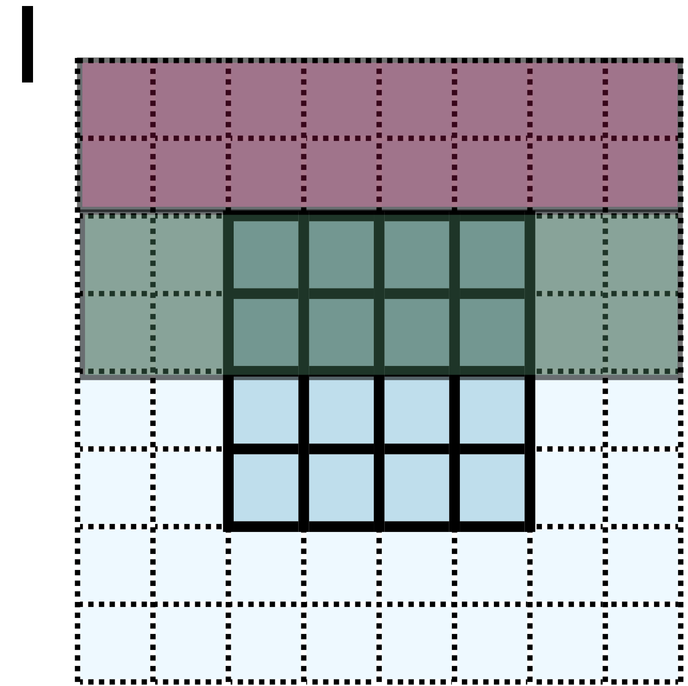
Count OldType &NewType

```
MPI_Datatype ybctype;

ierr = MPI_Type_contiguous(nvals*nguard*(ny), MPI_REAL, &ybctype);
ierr = MPI_Type_commit(&ybctype);

MPI_Send(&(u[ny][0][0]), 1, ybctype, ....)

ierr = MPI_Type_free(&ybctype);
```

SciNet

# Implementing in MPI

- Creating MPI Data types.
- MPI_Type_contiguous: simplest case. Lets you build a string of some other type.



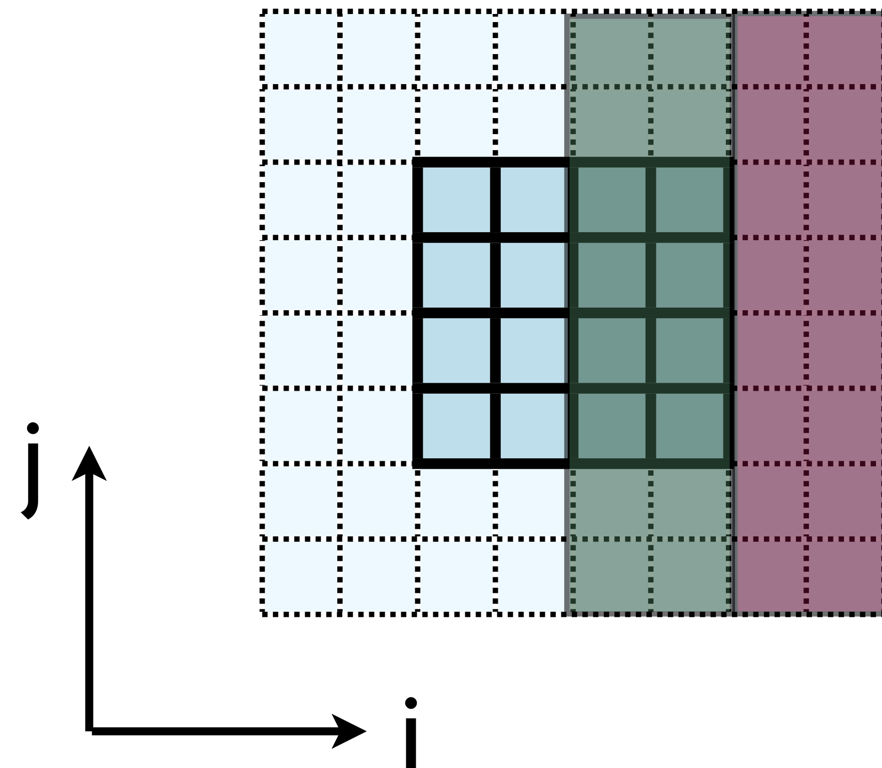

Count OldType NewType

```
integer :: ybctype

call MPI_Type_contiguous(nvals*nguard*(ny), MPI_REAL, ybctype, ierr)
call MPI_Type_commit(ybctype, ierr)

MPI_Send(u(1,1,ny), 1, ybctype, ....)

call MPI_Type_free(ybctype, ierr)
```

SciNet

# Implementing in MPI

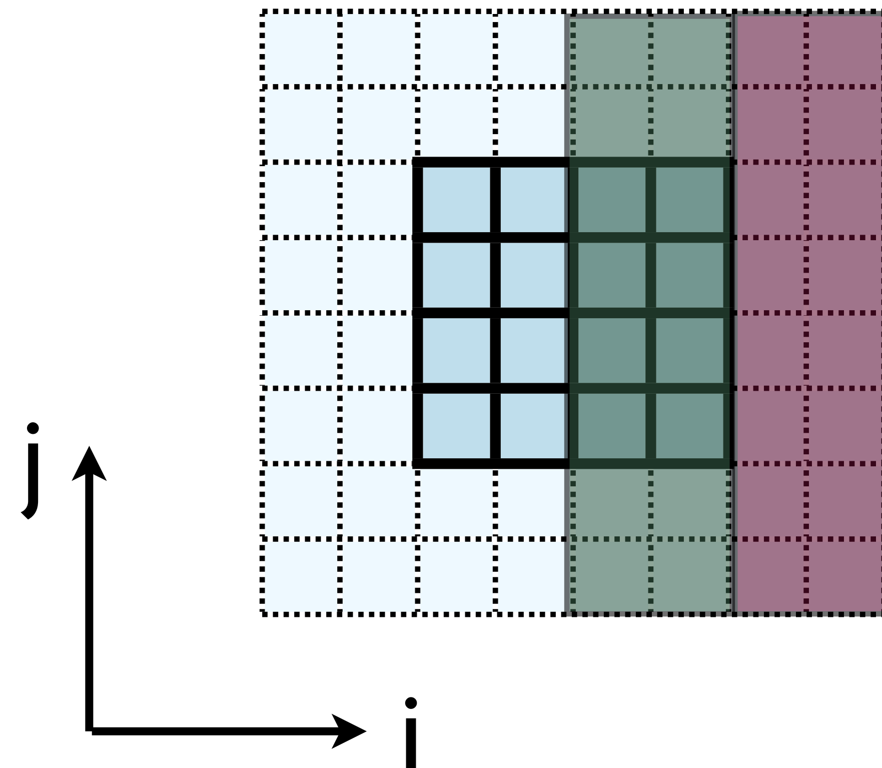
- Recall how 2d memory is laid out
- x gcs or boundary values *not* contiguous
- How do we do something like this for the x-direction?

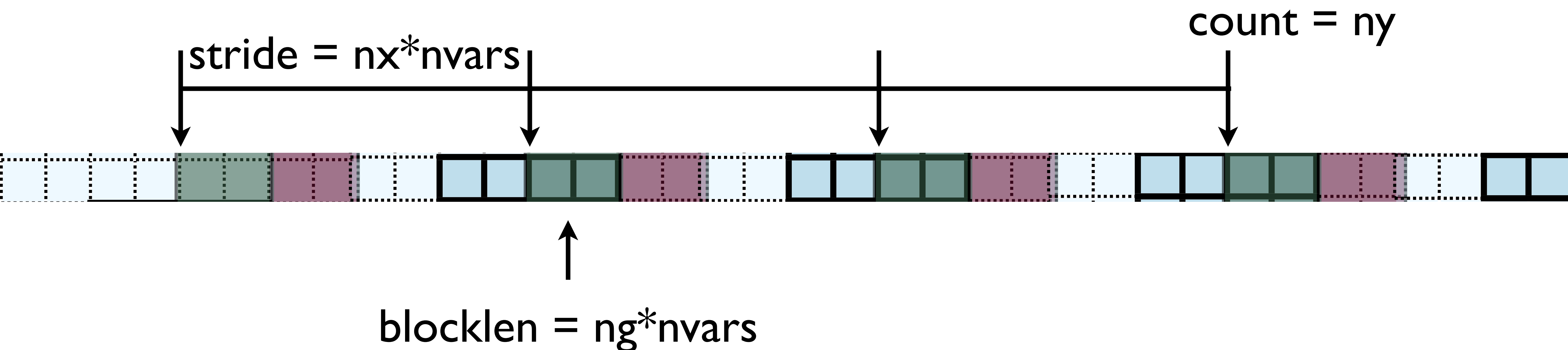


SciNet

# Implementing in MPI

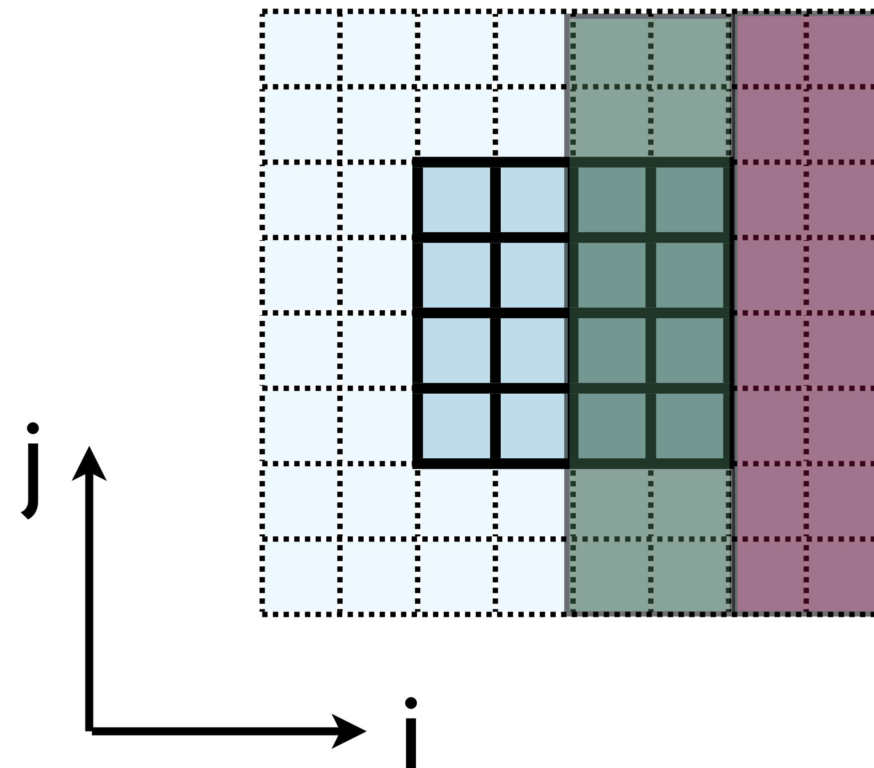
```
int MPI_Type_vector(
        int count,
        int blocklen,
        int stride,
        MPI_Datatype old_type,
        MPI_Datatype *newtype );
```

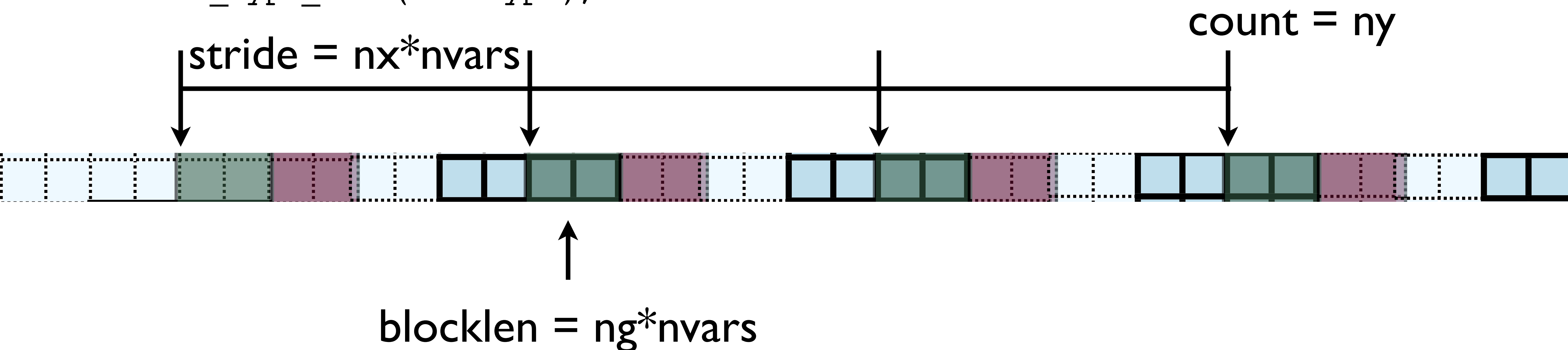


# Implementing in MPI

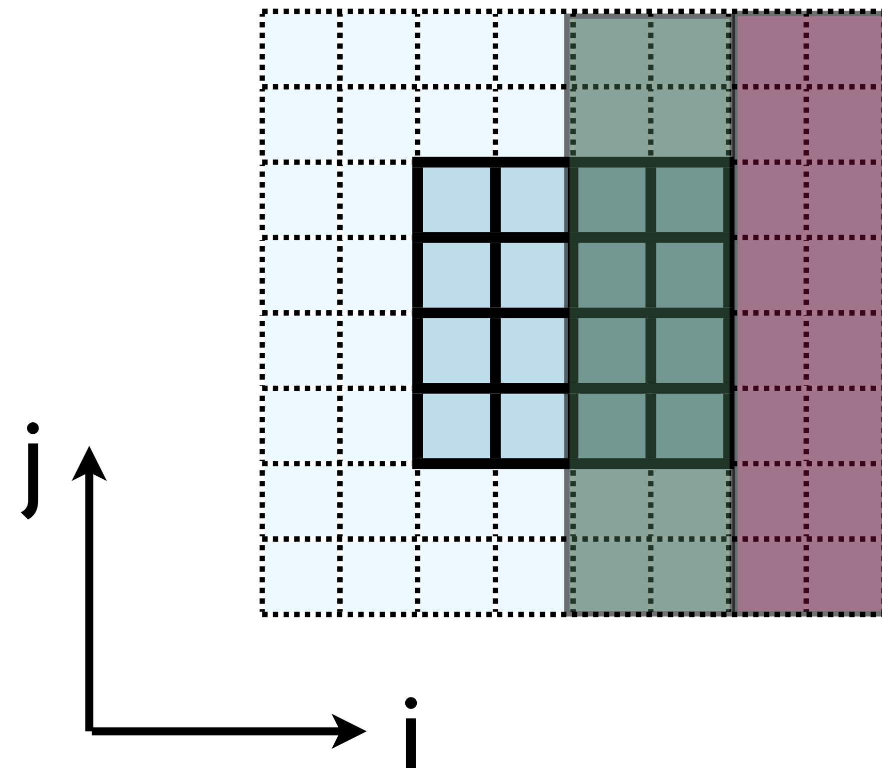
```
ierr = MPI_Type_vector(ny, nguard*nvars,
         nx*nvars, MPI_FLOAT, &xbctype);

ierr = MPI_Type_commit(&xbctype);

ierr = MPI_Send(&(u[0][nx][0]), 1, xbctype, ....)

ierr = MPI_Type_free(&xbctype);
```

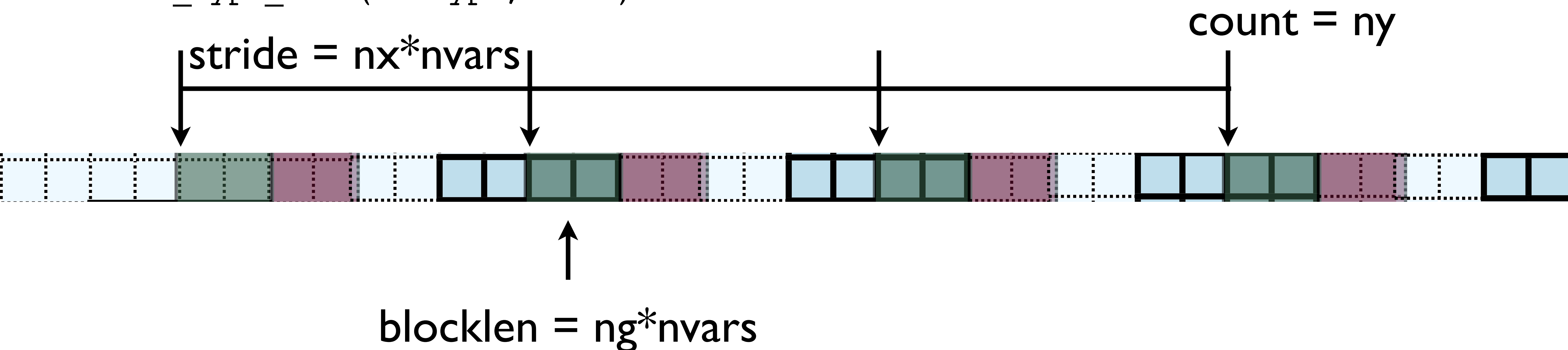


# Implementing in MPI

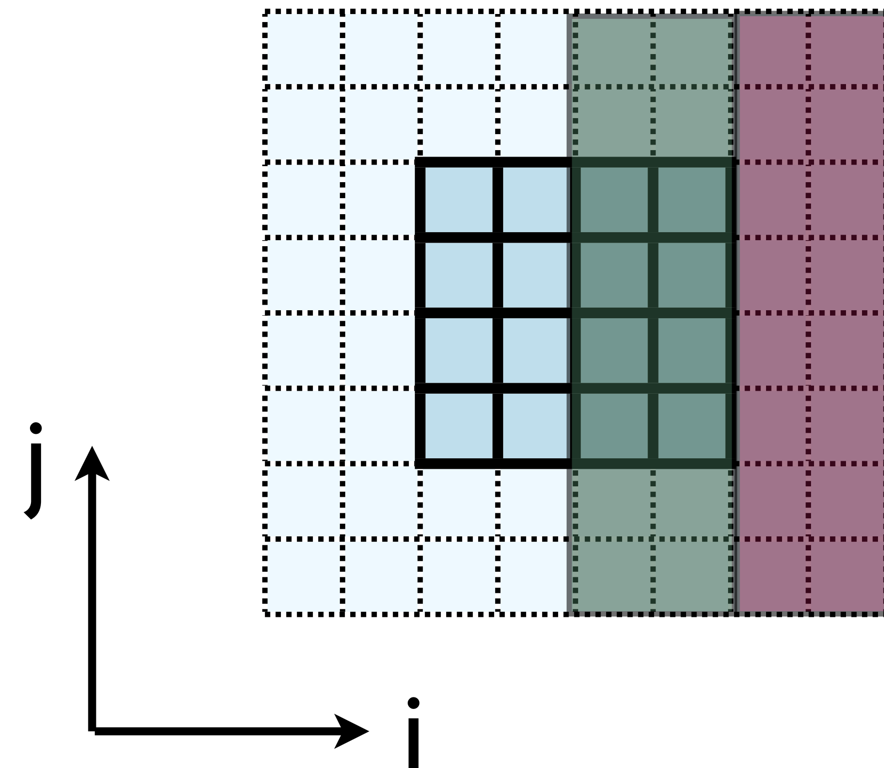
```
call MPI_Type_vector(ny, nguard*nvars,
        nx*nvars, MPI_REAL, xbctype, ierr)

call MPI_Type_commit(xbctype, ierr)

call MPI_Send(u(1,nx,1), 1, ybctype, ....)

call MPI_Type_free(xbctype, ierr)
```

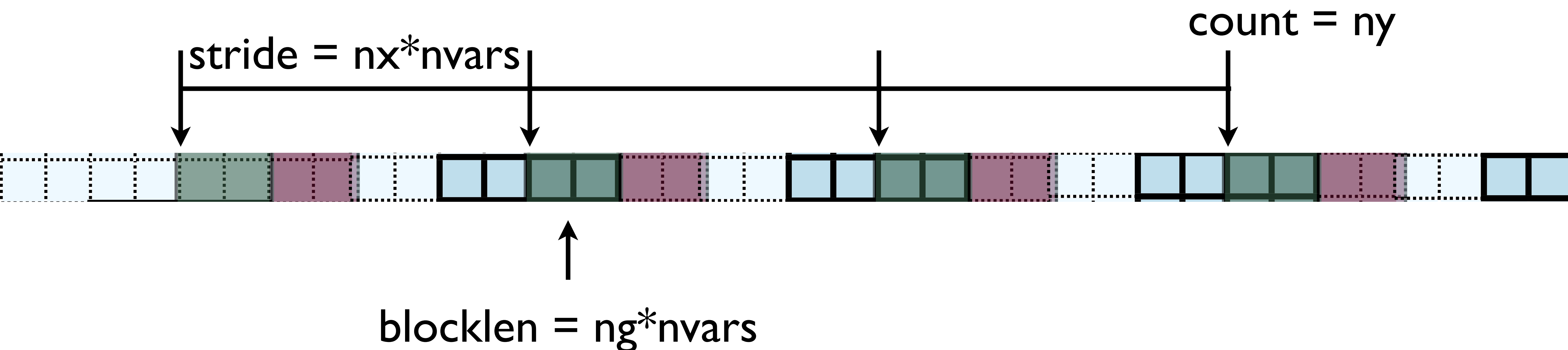


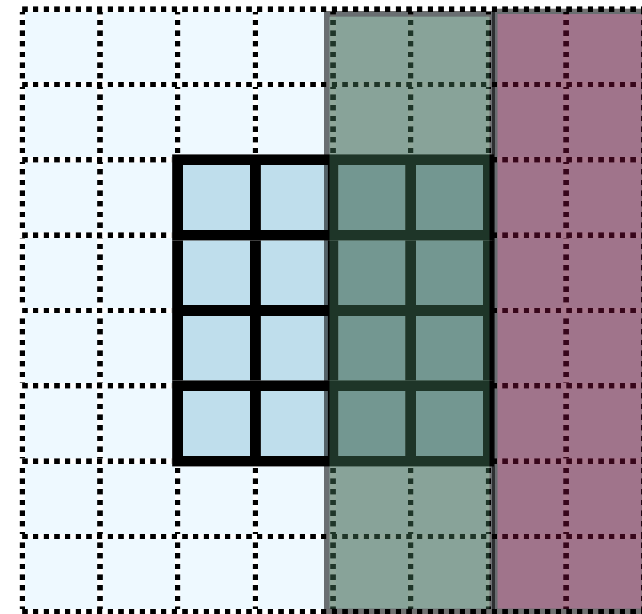
# Implementing in MPI

- Check: total amount of data = blocklen*count = ny*ng*nvars
- Skipped over stride*count = nx*ny*nvars



# Implementing in MPI

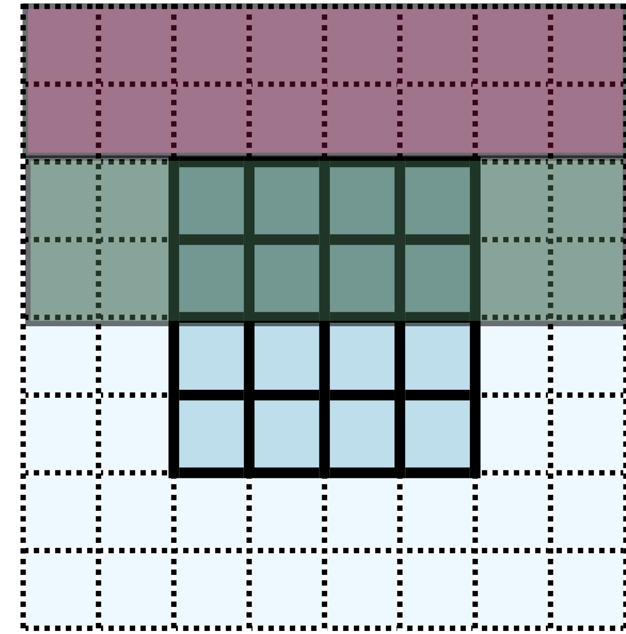


- Hands-On: Implement X guardcell filling with types.
- Implement vectorGuardCells
- For now, create/free type each cycle through; ideally, we'd create/free these once.

SciNet

# In MPI, there's always more than one way..

- MPI_Type_create_subarray ; piece of a multi-dimensional array.
- *Much* more convenient for higher-dimensional arrays
- (Otherwise, need vectors of vectors of vectors...)

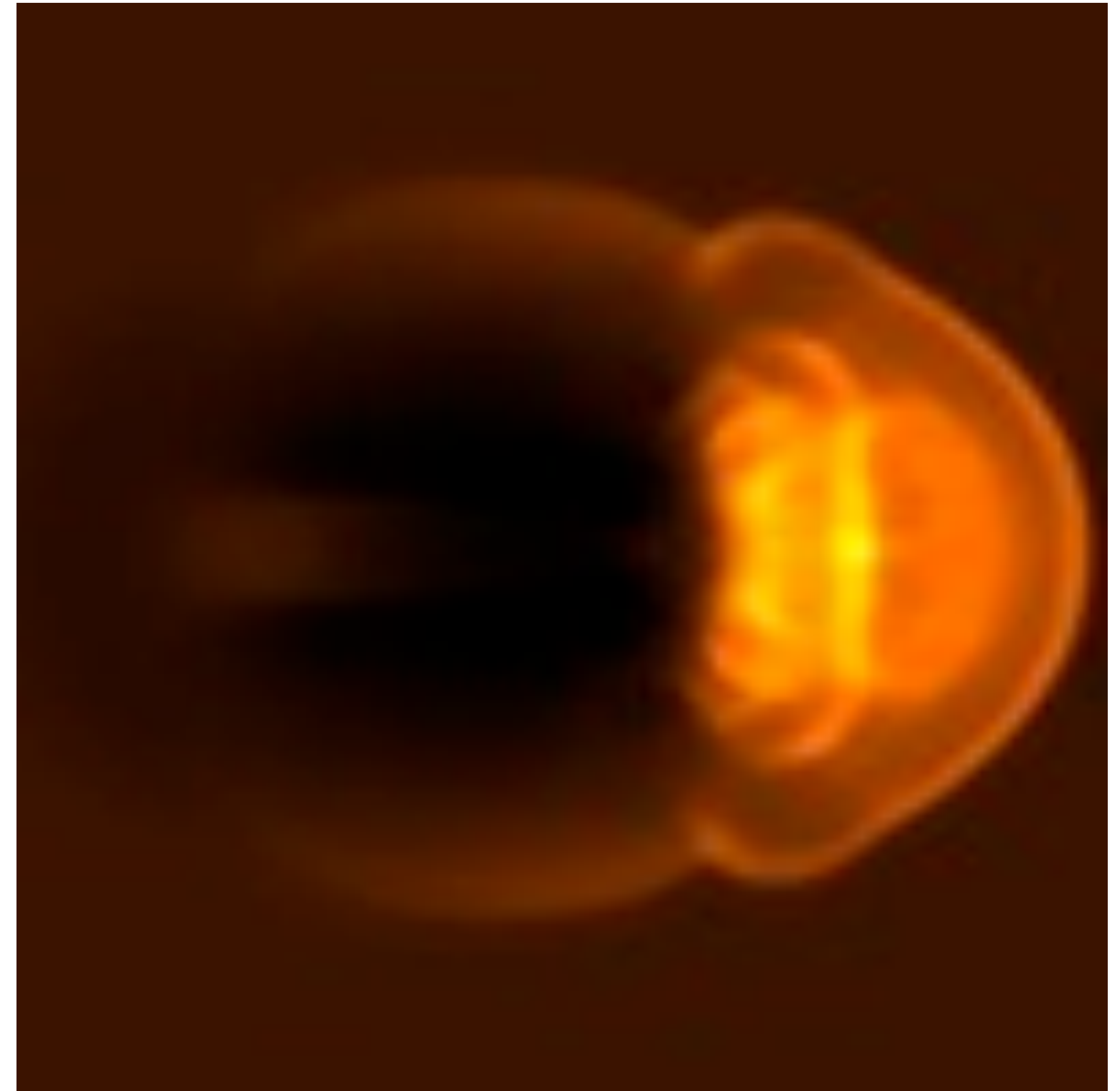


```
int MPI_Type_create_subarray(
      int ndims, int *array_of_sizes,
      int *array_of_subsizes,
      int *array_of_starts,
      int order,
      MPI_Datatype oldtype,
      MPI_Datatype &newtype);

call MPI_Type_create_subarray(
      integer ndims, [array_of_sizes],
      [array_of_subsizes],
      [array_of_starts],
      order, oldtype,
      newtype, ierr)
```

SciNet

# MPI-IO

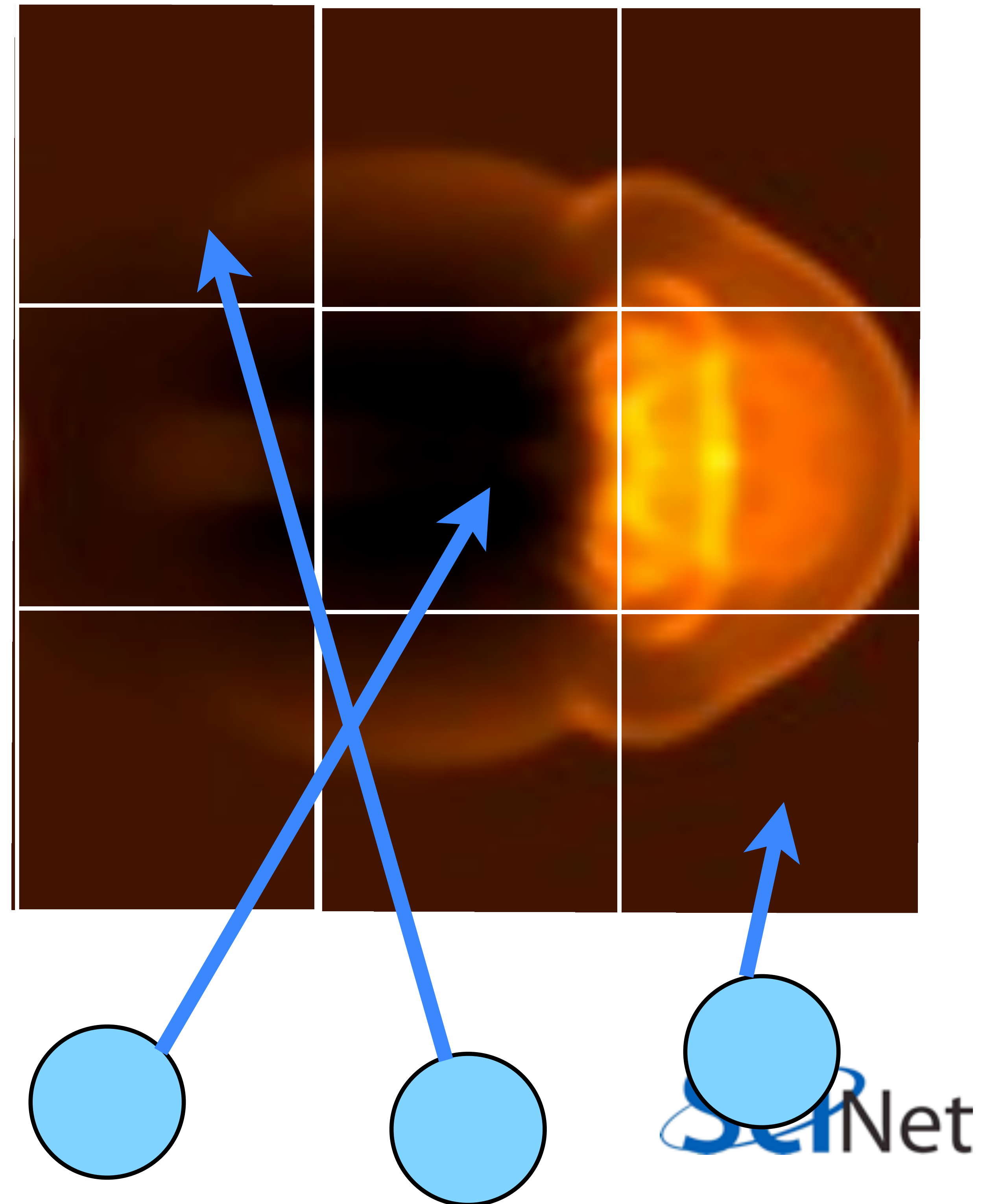
- Would like the new, parallel version to still be able to write out single output files.
- But at no point does a single processor have entire domain...



SciNet

# Parallel I/O

- Each processor has to write its own piece of the domain..
- without overwriting the other.
- Easier if there is global coordination



# MPI-IO

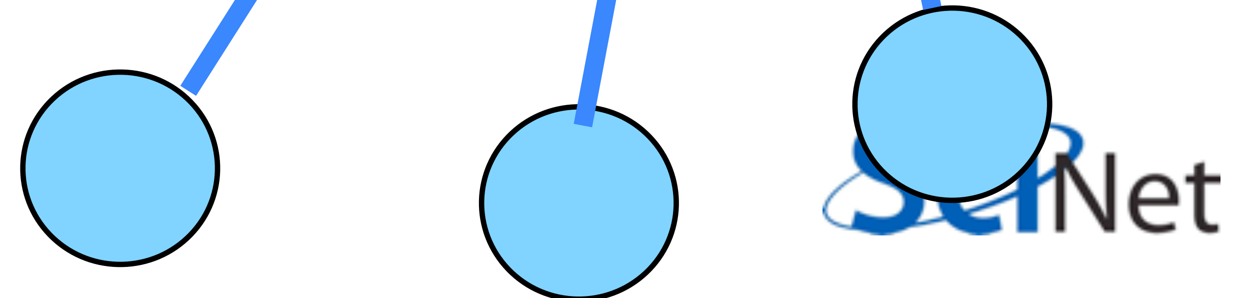
- Uses MPI to coordinate reading/writing to single file

```
ierr = MPI_File_open(MPI_COMM_WORLD,filename, MPI_MODE_WRONLY | MPI_MODE_APPEND , MPI_INFO_NULL, &file);
```

...stuff...

```
ierr = MPI_File_close(&file);
```

- Coordination -- *collective* operations.



# PPM file format

- Simple file format
- Someone has to write a header, then each PE has to output only its 3-bytes pixels skipping everyone elses.

header -- ASCII characters

'P6', comments, height/width, max val

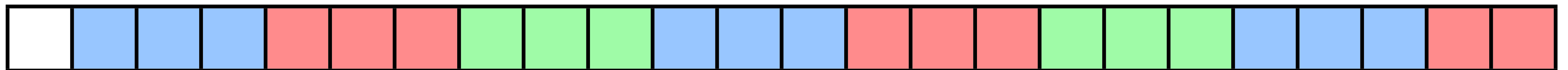
```
P6
# min = 1.000000e+00, max = 4.733462e+01
100 100
255
(rgb)(rgb)(rgb)...
(rgb)(rgb)(rgb)...
```

row by row triples of bytes: each pixel = 3 bytes

SciNet

# MPI-IO File View

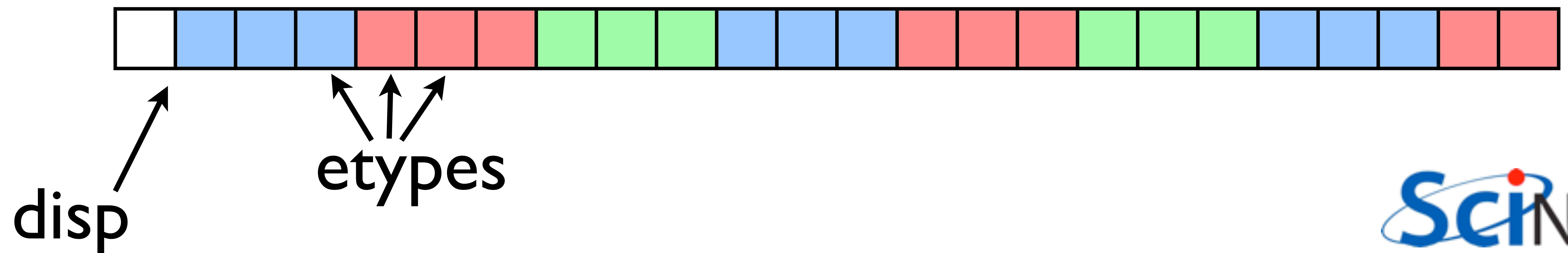
- Each process has a view of the file that consists of only of the parts accessible to it.
- For writing, hopefully non-overlapping!
- Describing this - how data is laid out in a file - is very similar to describing how data is laid out in memory...



SciNet

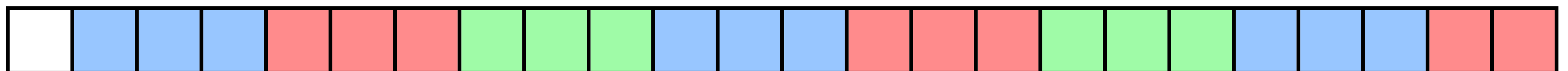
# MPI-IO File View

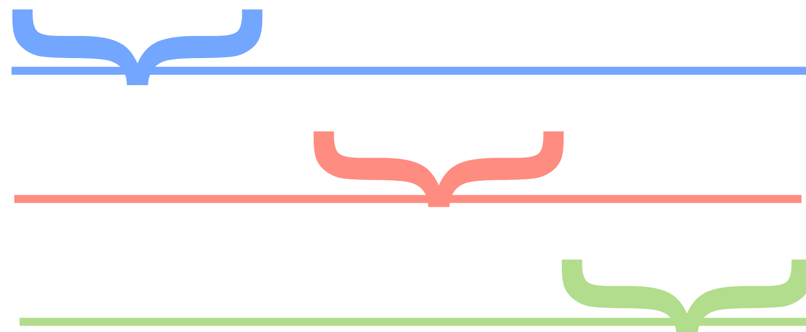
- int MPI_File_set_view(
  MPI_File fh,
  MPI_Offset disp, /* displacement in *bytes* from start */
  MPI_Datatype etype, /* elementary type */
  MPI_Datatype filetype, /* file type; prob different for each proc */
  char *datarep, /* 'native' or 'internal' */
  MPI_Info info) /* MPI_INFO_NULL for today */



# MPI-IO File View

- int MPI_File_set_view(
  MPI_File fh,
  MPI_Offset disp, /* displacement in bytes from start */
  MPI_Datatype etype, /* elementary type */
  MPI_Datatype filetype, /* file type; prob different for each proc */
  char *datarep, /* 'native' or 'internal' */
  MPI_Info info) /* MPI_INFO_NULL */





Filetypes (made up of etypes; repeat as necessary)

SciNet

# MPI-IO File Write

- int MPI_File_write_all(
  MPI_File fh,
  void *buf,
  int count,
  MPI_Datatype datatype,
  MPI_Status *status)

Writes (_all: collectively) to part of file within view.

SciNet

# Homework MPI diffusion

onc-mpi.c or
Make an MPI-ed version of diffusion equation
(Build: `make make diffusionc-mpi`)
Test on 1..8 procs

- add standard MPI calls: init, finalize, comm_size, comm_rank
- Figure out how many points PE is responsible for (~totpoints/size)
- Figure out neighbors
- Start at 1, but end at totpoints/size
- At end of step, exchange guardcells; use sendrecv
- Get total error

SciNet compute • calcul CANADA

# C syntax

```
MPI_Status status;

ierr = MPI_Init(&argc, &argv);
ierr = MPI_Comm_{size,rank}(Communicator, &{size,rank});
ierr = MPI_Send(sendptr, count, MPI_TYPE, destination,
                 tag, Communicator);
ierr = MPI_Recv(rcvptr, count, MPI_TYPE, source, tag,
                 Communicator, &status);
ierr = MPI_Sendrecv(sendptr, count, MPI_TYPE, destination,tag,
                    recvptr, count, MPI_TYPE, source, tag,
                    Communicator, &status);
ierr = MPI_Allreduce(&mydata, &globaldata, count, MPI_TYPE,
                     MPI_OP, Communicator);

Communicator -> MPI_COMM_WORLD
MPI_Type -> MPI_FLOAT, MPI_DOUBLE, MPI_INT, MPI_CHAR...
MPI_OP -> MPI_SUM, MPI_MIN, MPI_MAX,...
```

SciNet compute • calcul CANADA

# FORTRAN syntax

```
integer status(MPI_STATUS_SIZE)

call MPI_INIT(ierr)
call MPI_COMM_{SIZE,RANK}(Communicator, {size,rank},ierr)
call MPI_SSEND(sendarr, count, MPI_TYPE, destination,
                 tag, Communicator)
call MPI_RECV(rcvarr, count, MPI_TYPE, destination,tag,
                 Communicator, status, ierr)
call MPI_SENDRECV(sendptr, count, MPI_TYPE, destination,tag,
                    recvptr, count, MPI_TYPE, source, tag,
                    Communicator, status, ierr)
call MPI_ALLREDUCE(&mydata, &globaldata, count, MPI_TYPE,
                     MPI_OP, Communicator, ierr)


Communicator -> MPI_COMM_WORLD
MPI_Type -> MPI_REAL, MPI_DOUBLE_PRECISION,
           MPI_INTEGER, MPI_CHARACTER
MPI_OP -> MPI_SUM, MPI_MIN, MPI_MAX,...
```

SciNet compute • calcul CANADA