MPI, Part 3

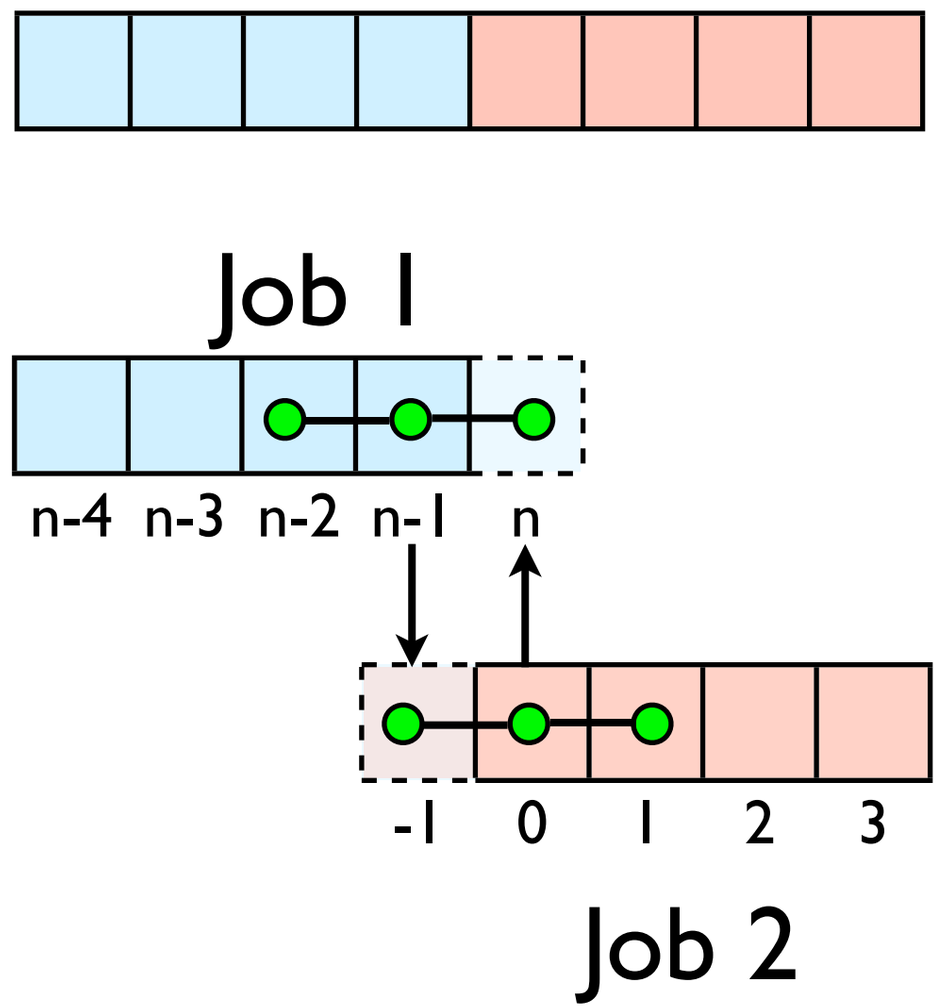
Scientific Computing Course, Part 3

SciNet compute • calcul CANADA

Non-blocking communications

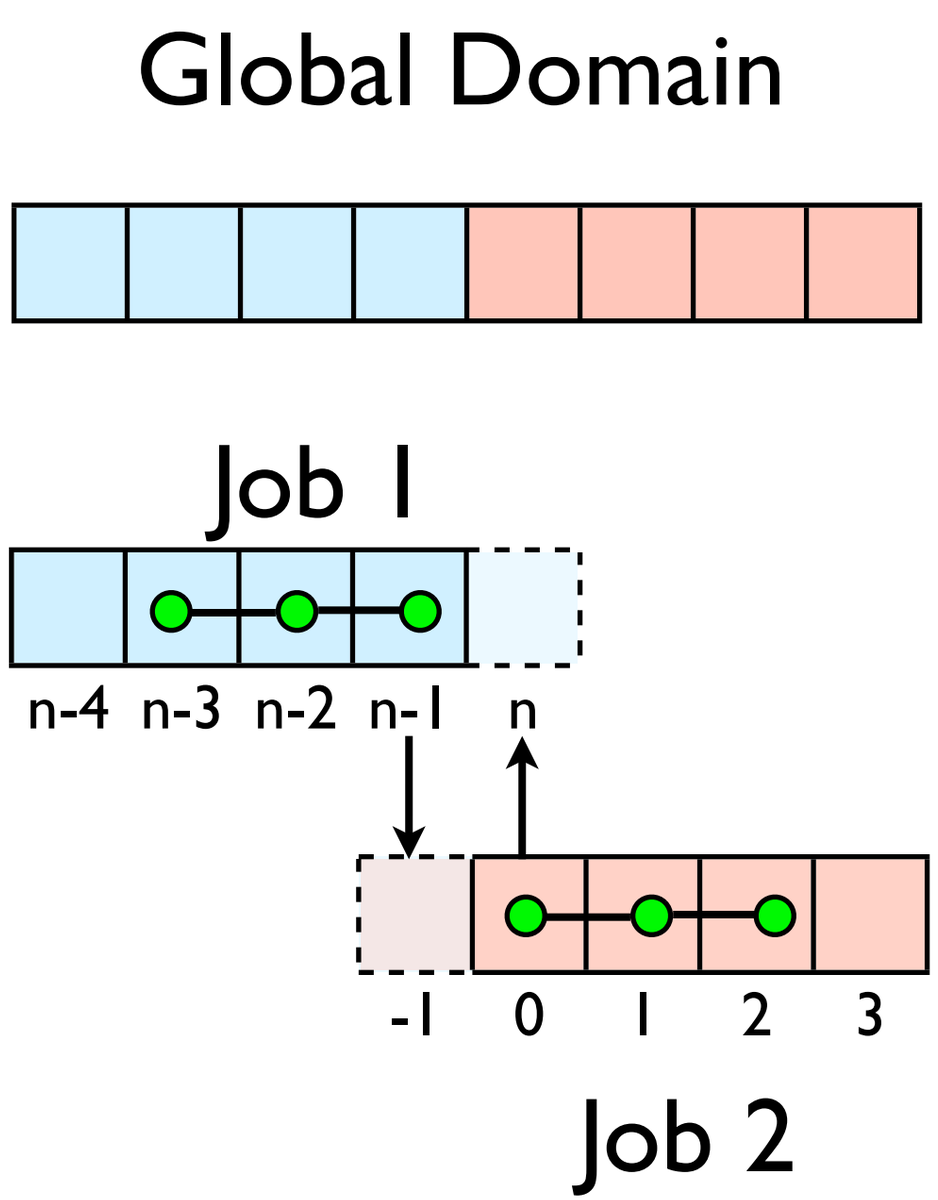
Diffusion: Had to wait for communications to compute

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



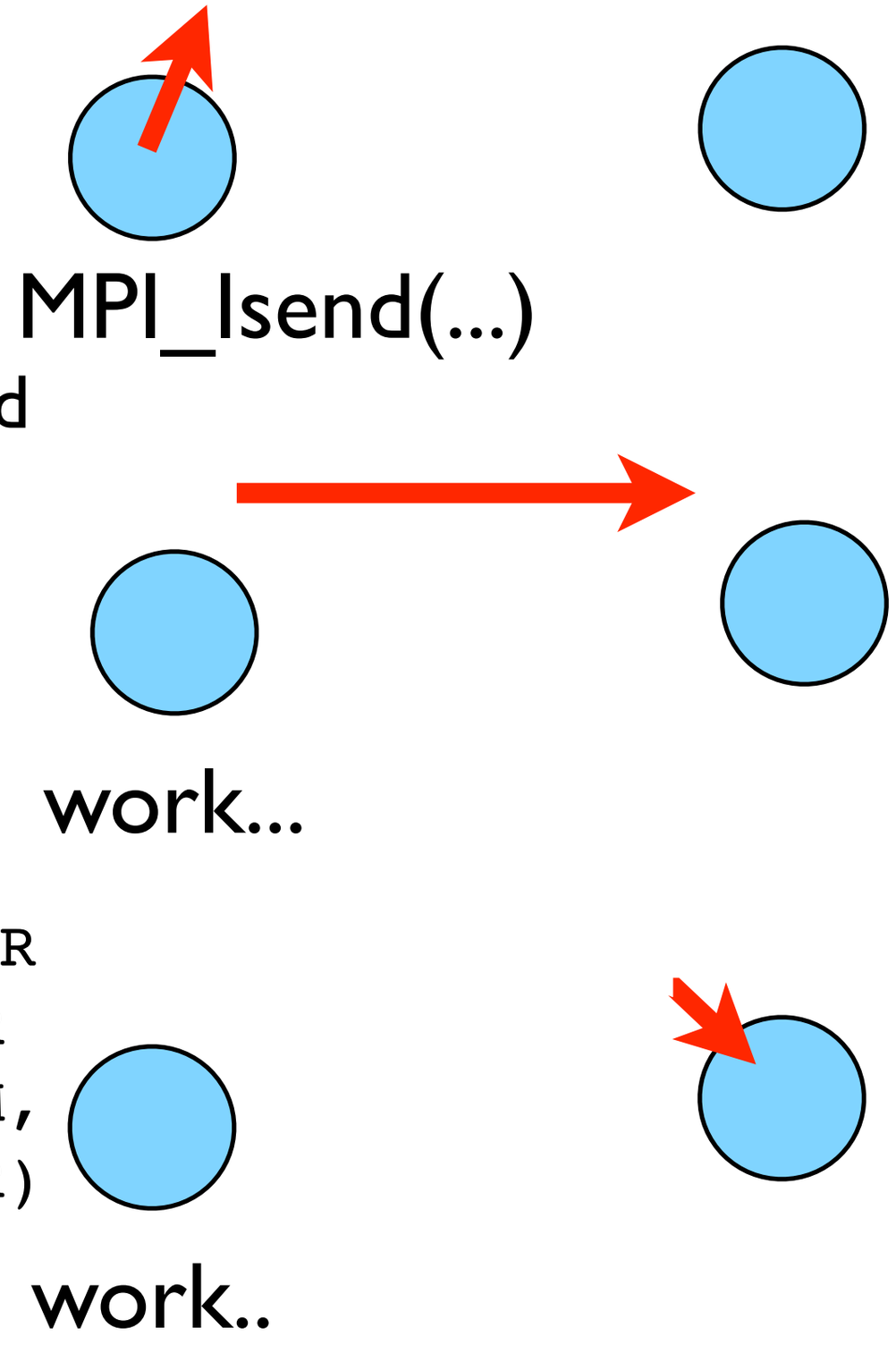
Diffusion: *Had* to wait?

- 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.



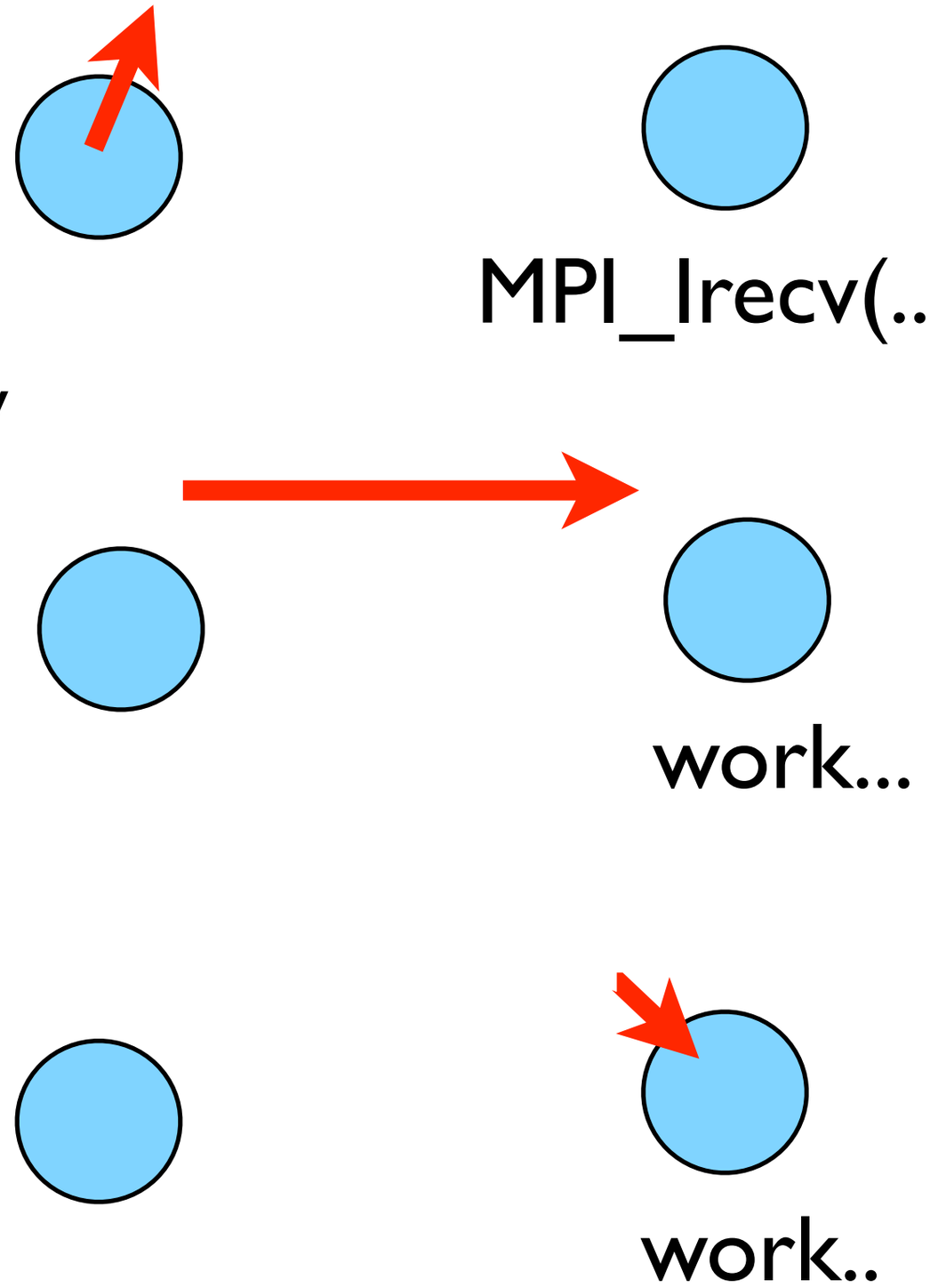
Nonblocking Sends

- 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?

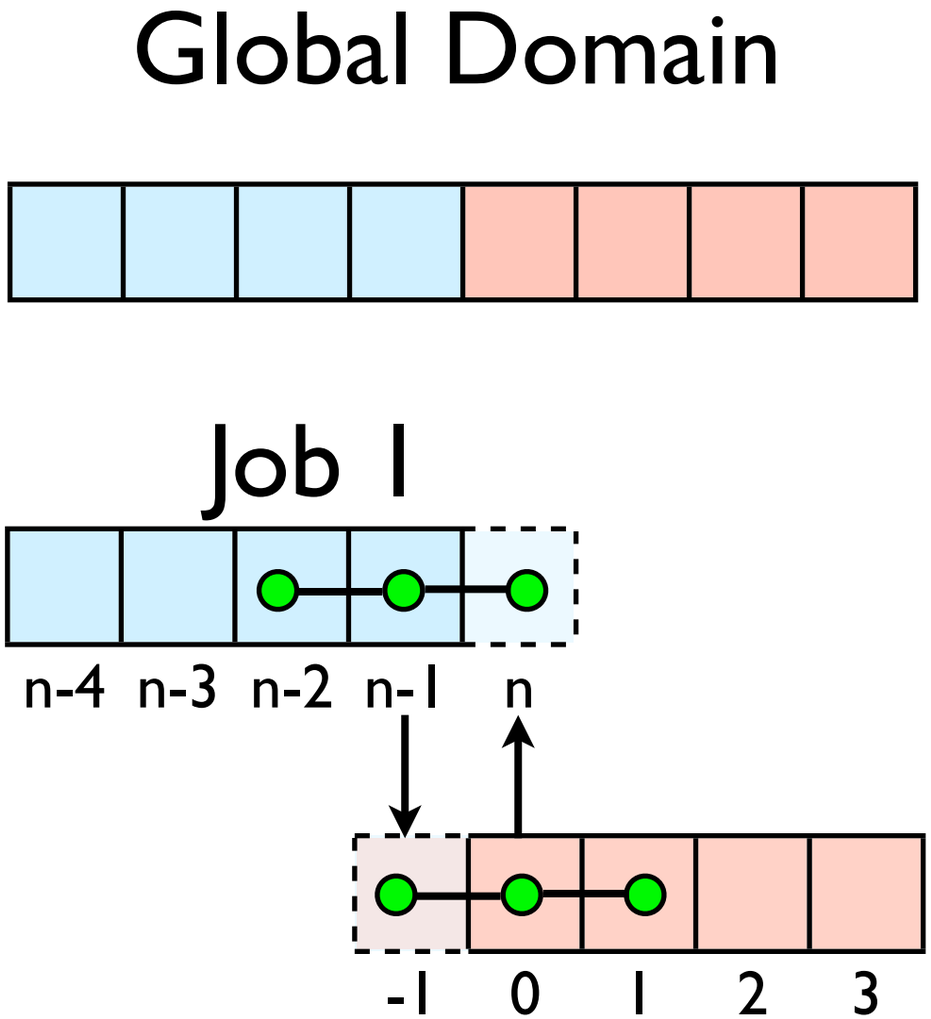
- `int MPI_Wait(MPI_Request *request,MPI_Status *status);`
- `MPI_WAIT(INTEGER REQUEST,INTEGER STATUS(MPI_STATUS_SIZE),INTEGER IERROR)`

- `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

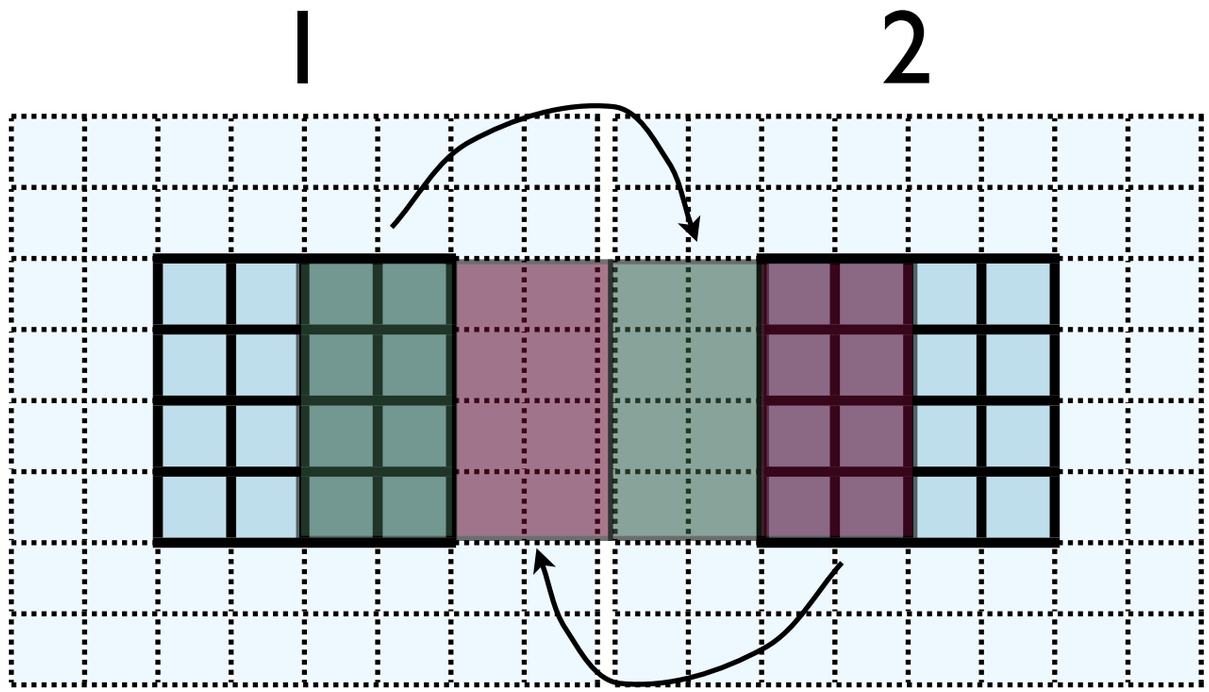
- Works for parallel decomposition!
- Job 1 needs info on Job 2s 0th zone, Job 2 needs info on Job 1s last zone
- Pad array with ‘guardcells’ and fill them with the info from the appropriate node by message passing or shared memory
- Hydro code: need guardcells 2 deep



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)

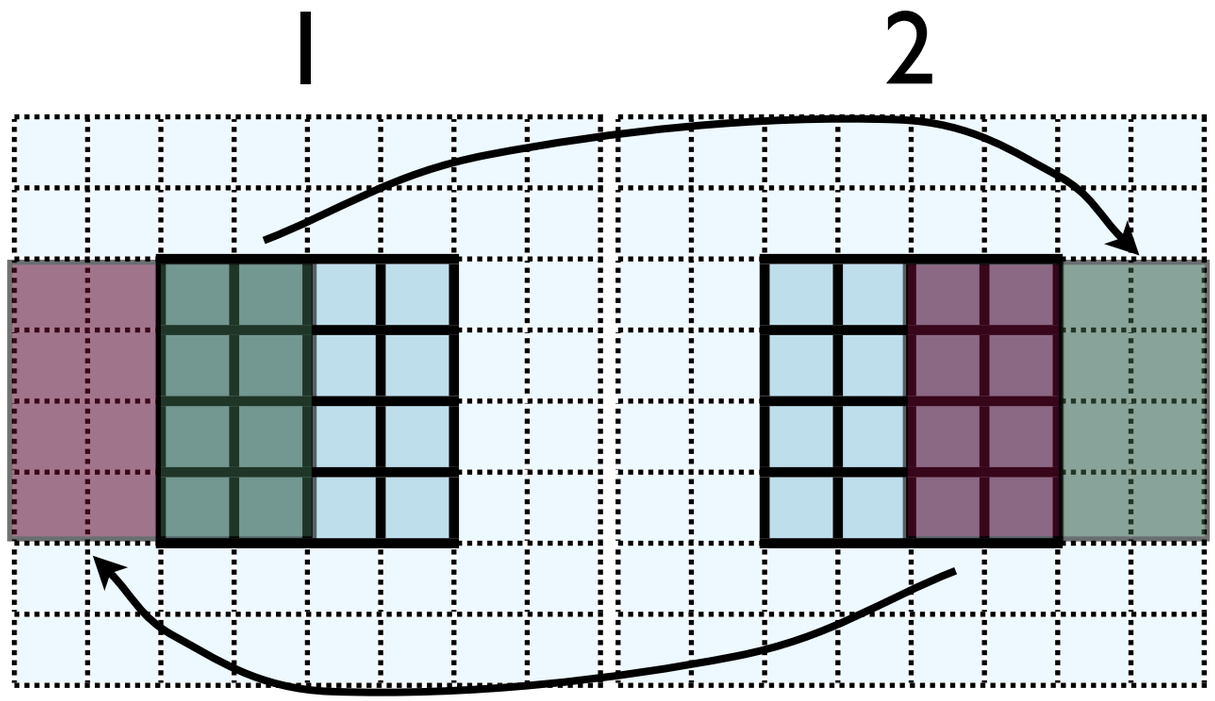
2: u(:, ng+1:2*ng, ng:ny-ng)
→ 1: u(:, nx+ng+1:nx+2*ng, ng:ny-ng)

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

SciNet

Cute way for Periodic BCs

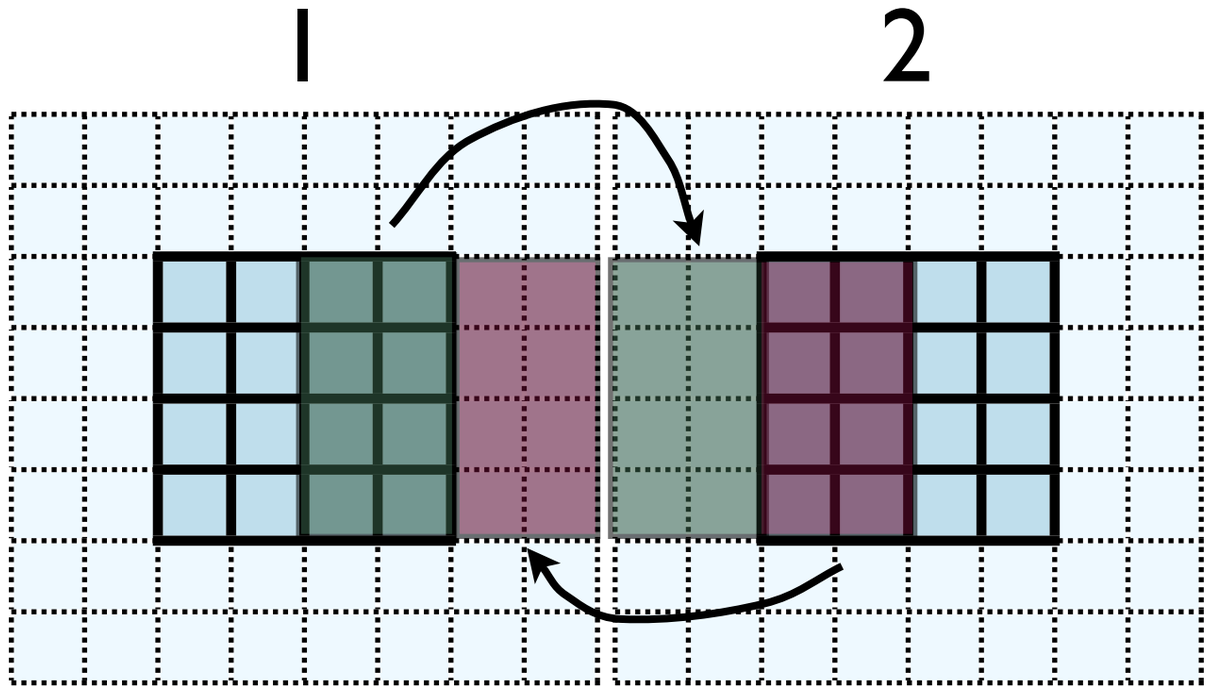
- Actually make the decomposed mesh periodic;
- Make the far ends of the mesh neighbors
- Don't know the difference between that and any other neighboring grid
- Cart_create sets this up for us automatically upon request.



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)

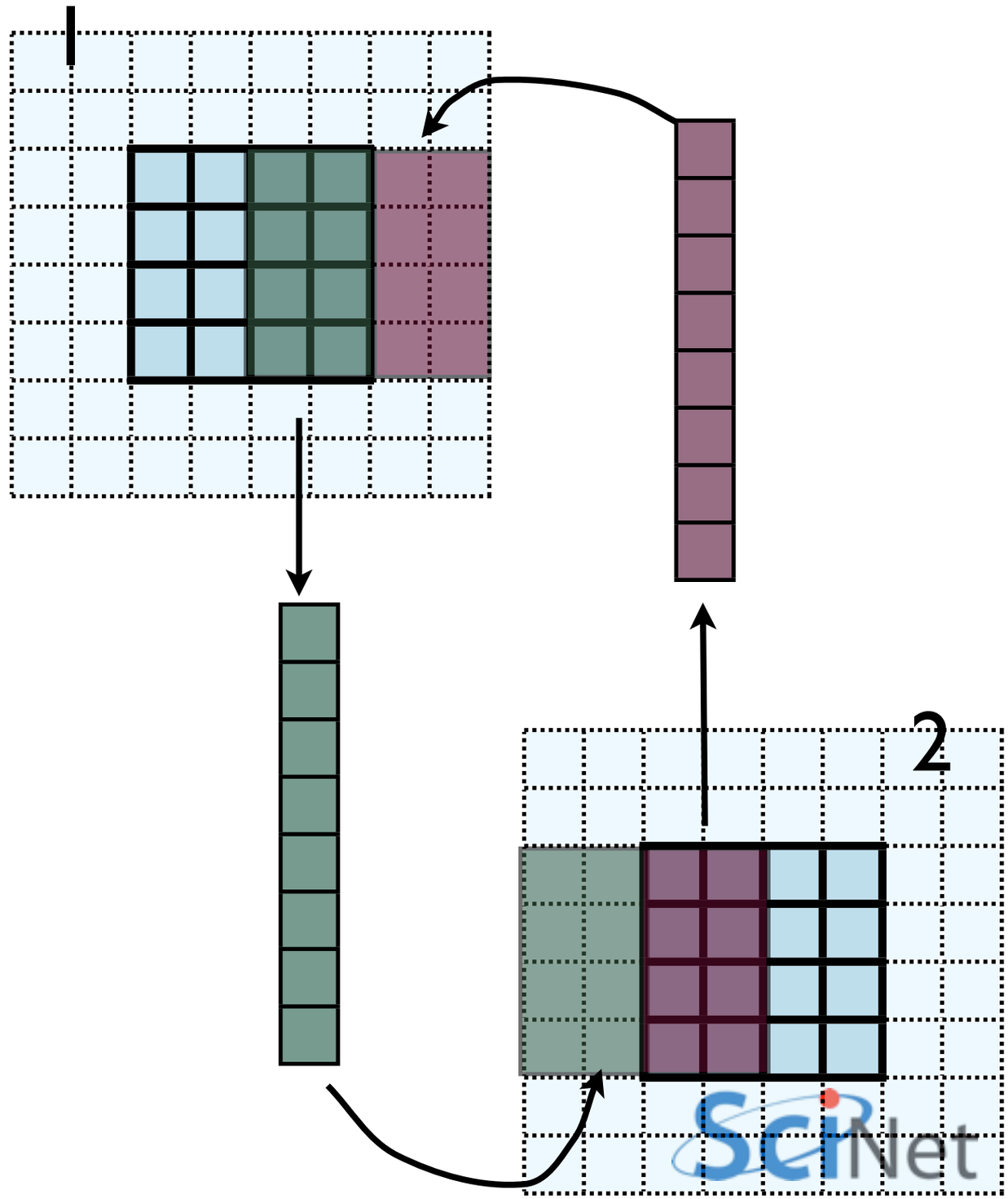
→ 1: u(:, nx+ng+1:nx+2*ng, ng:ny-ng)

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

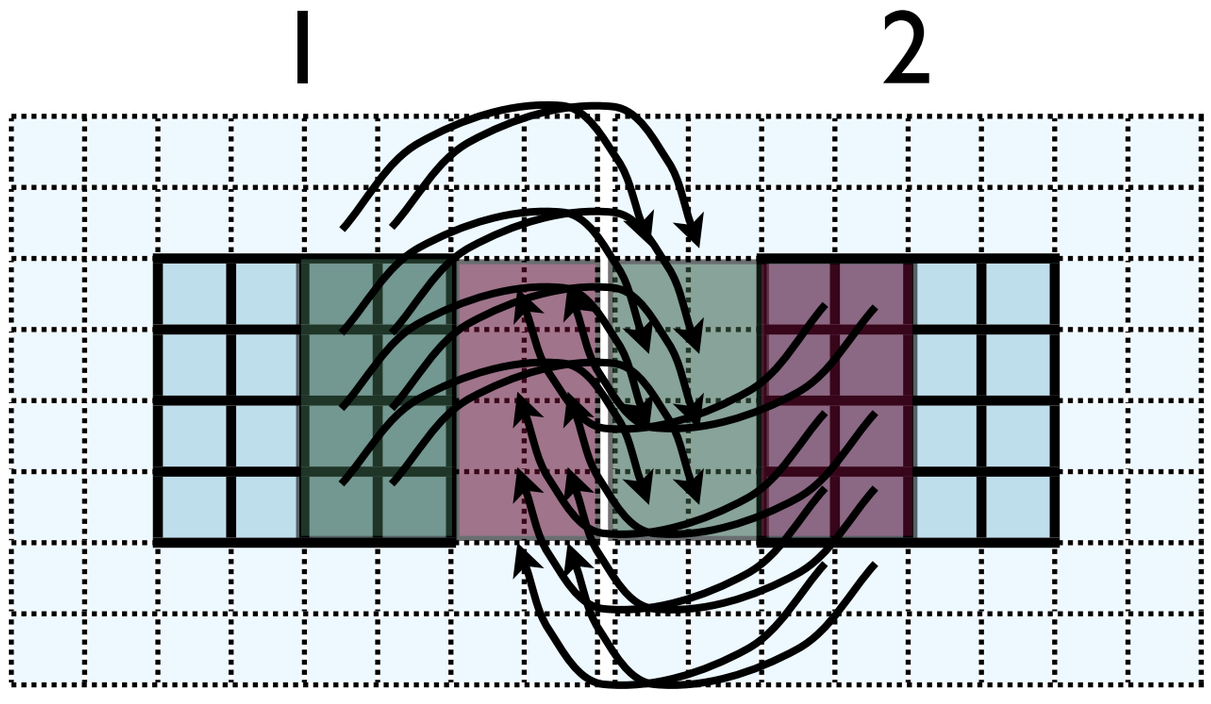
SciNet

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



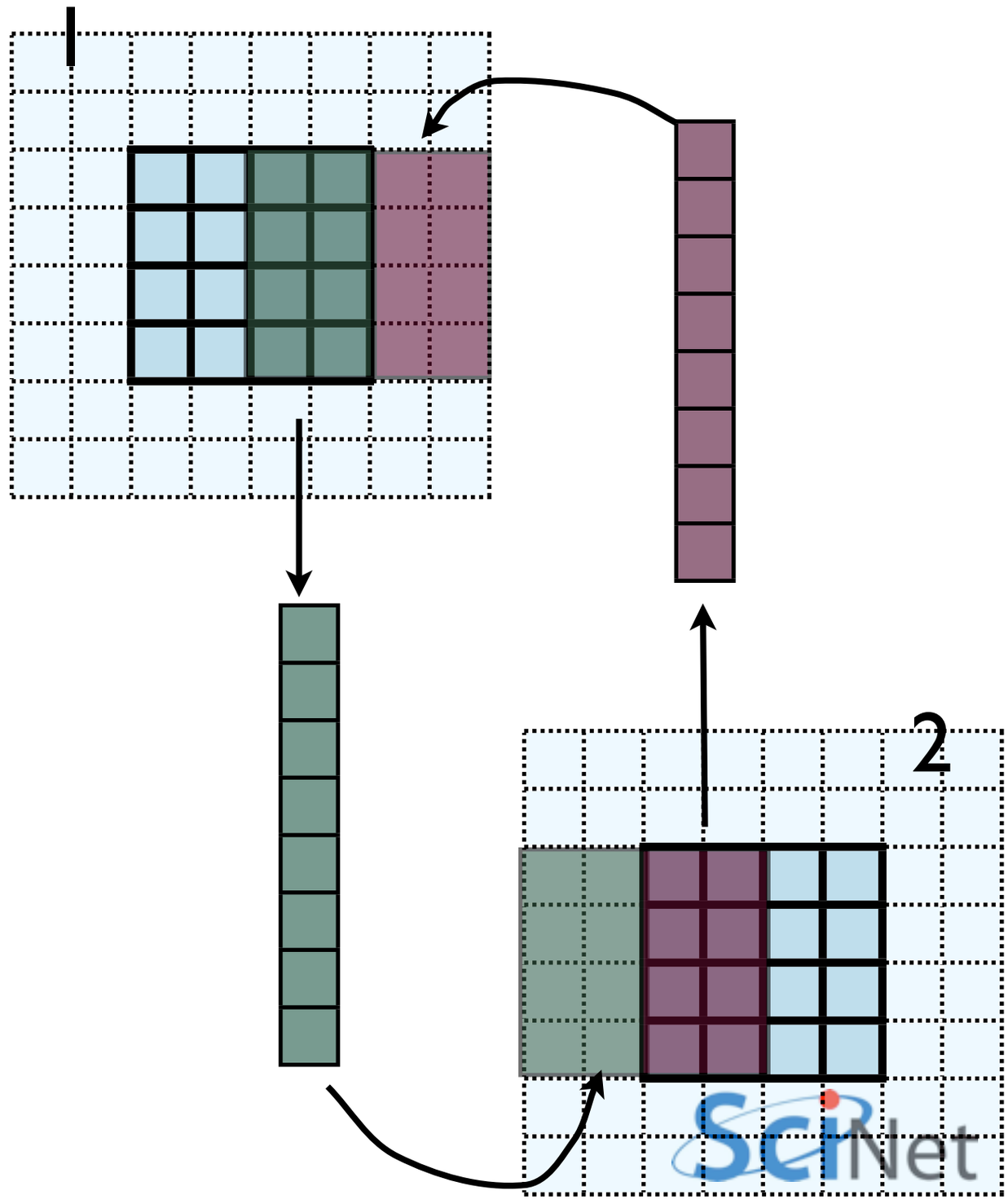
Implementing in MPI



- 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.

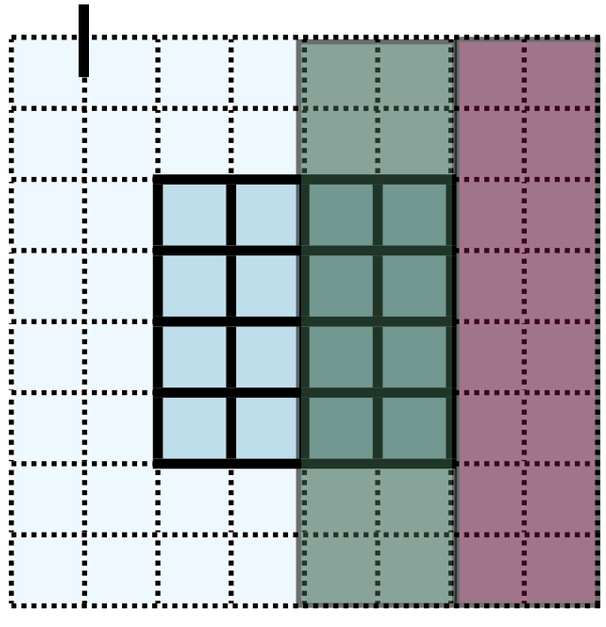
Implementing in MPI

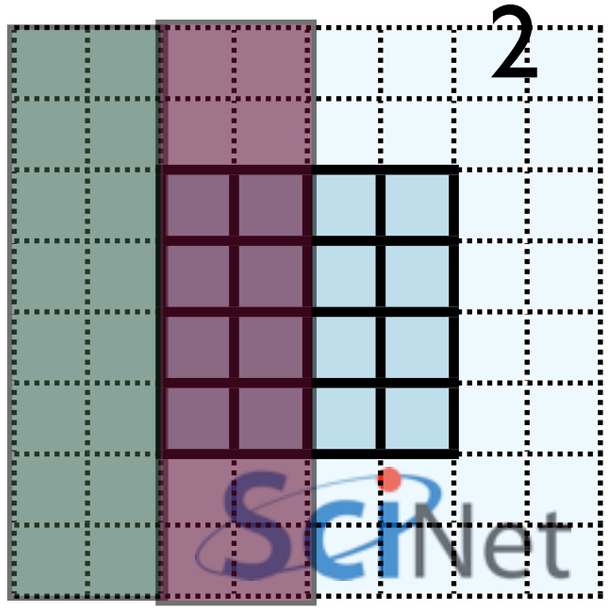
- This approach is simple, but introduces extraneous copies
- Memory bandwidth is already a bottleneck for these codes
- It would be nice to just point at the start of the guardcell data and have MPI read it from there.



Implementing in MPI

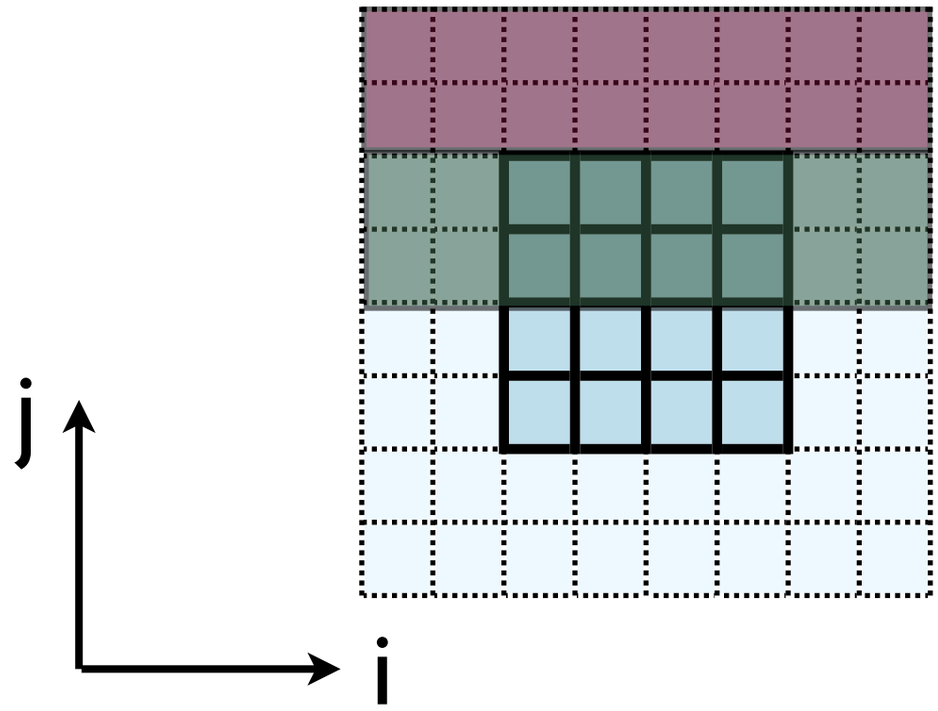
- Let me make one simplification for now; copy whole stripes
- This isn't necessary, but will make stuff simpler at first
- Only a cost of $2xNg^2 = 8$ extra cells (small fraction of ~200-2000 that would normally be copied)

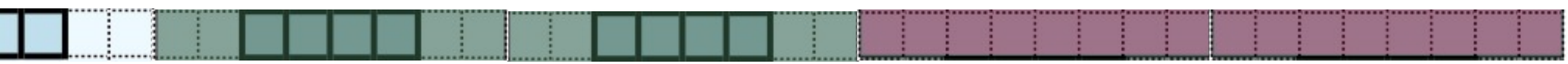




Implementing in MPI

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

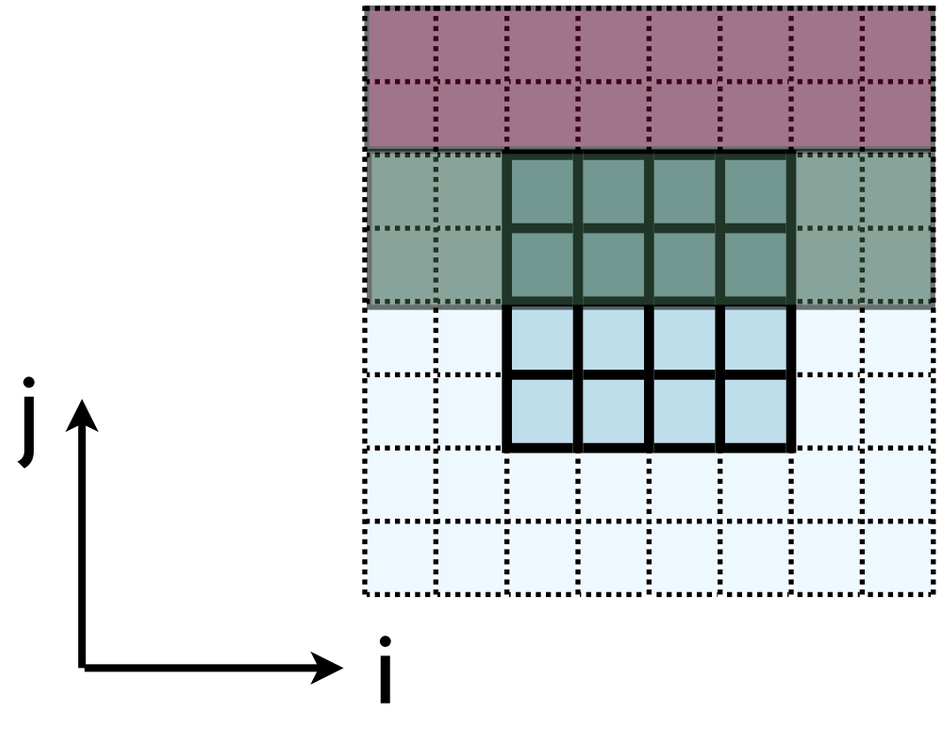




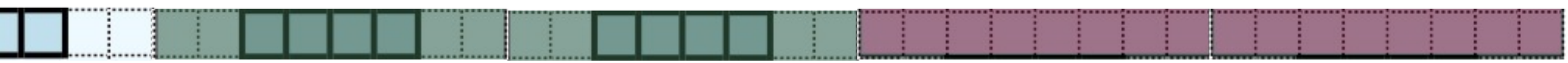
SciNet

Implementing in MPI

- Can send in one go:



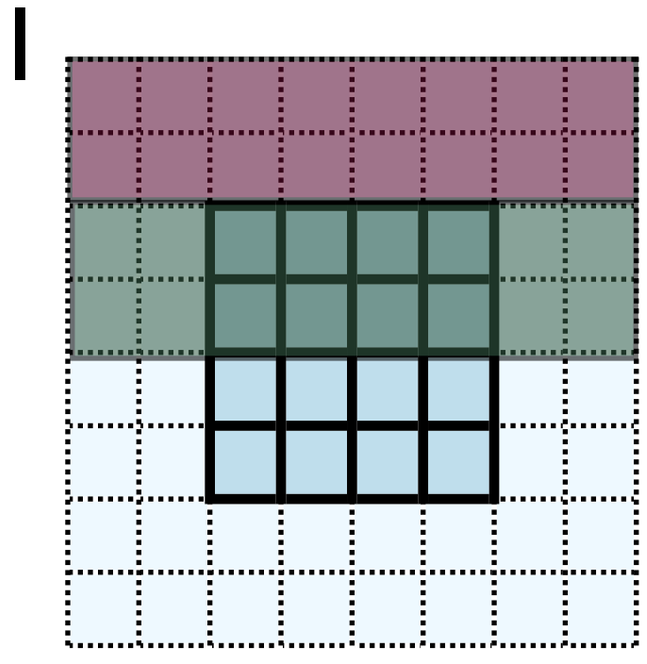

```
call MPI_Send(u(1,1,ny), nvars*nguard*ny, MPI_REAL, ....)
ierr = MPI_Send(&(u[ny][0][0]), nvars*nguard*ny, MPI_FLOAT, ....)
```



SciNet

Implementing in MPI

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



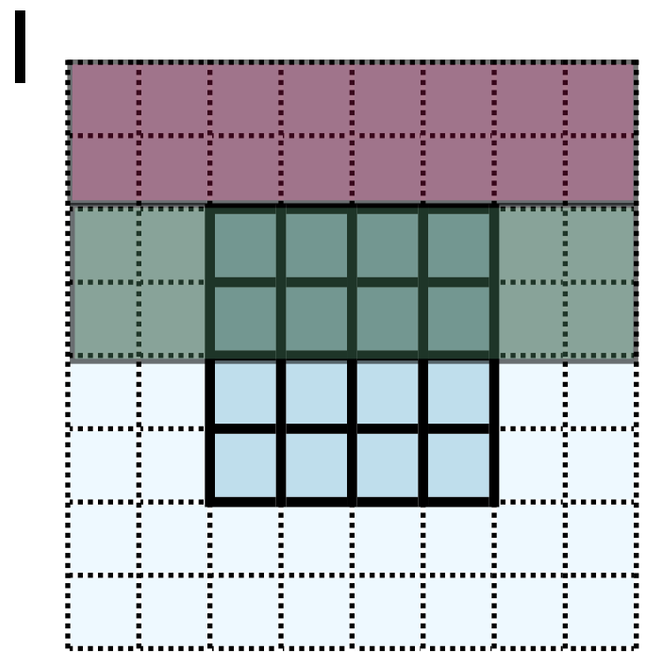
Count OldType &NewType

```
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.



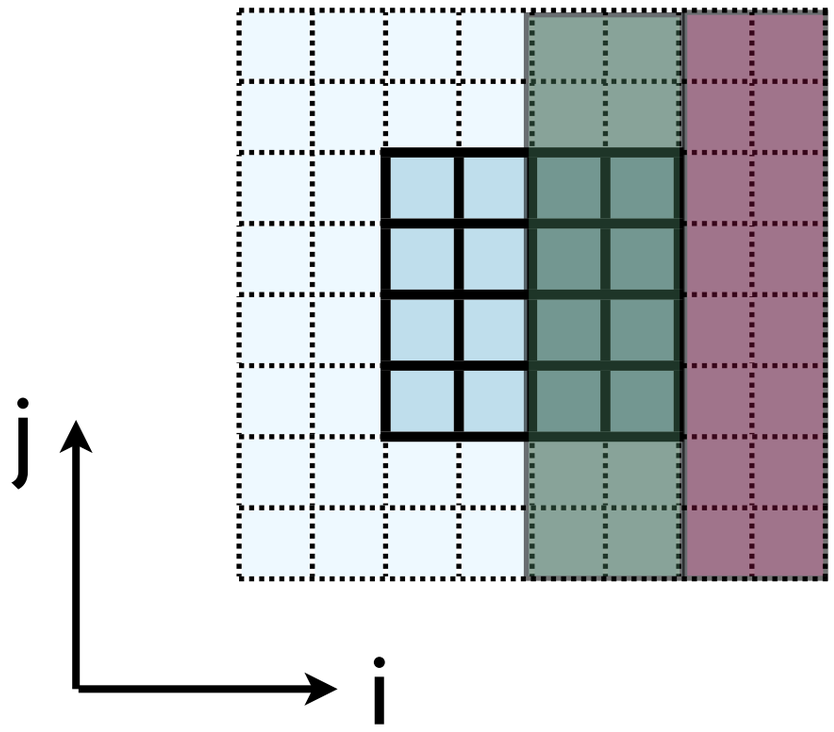
Count OldType NewType

```
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

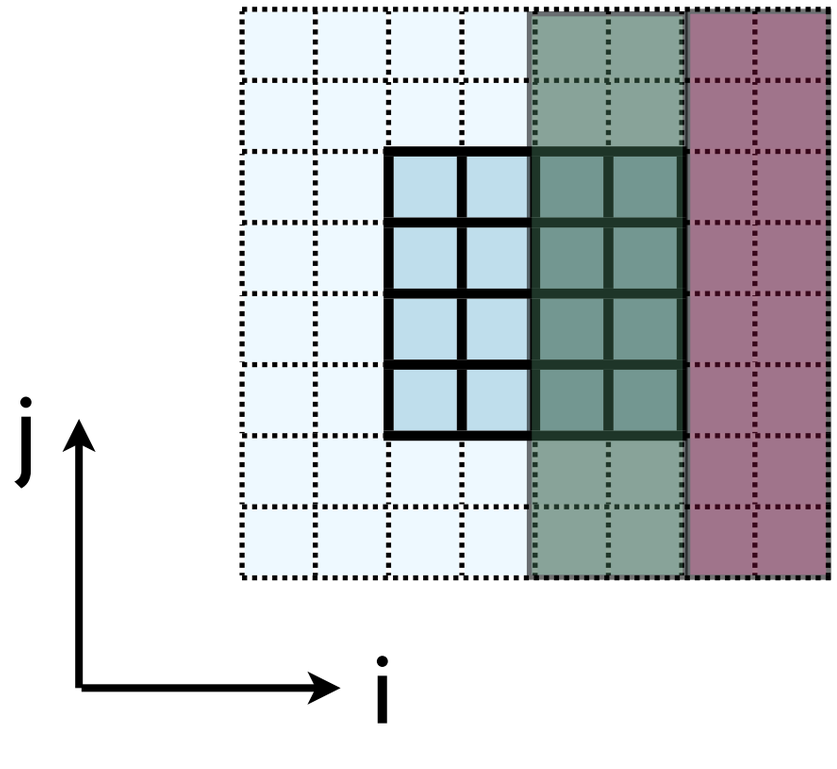
- Recall how 2d memory is laid out
- x gcs or boundary values *not* contiguous
- How do we do something like this for the x-direction?

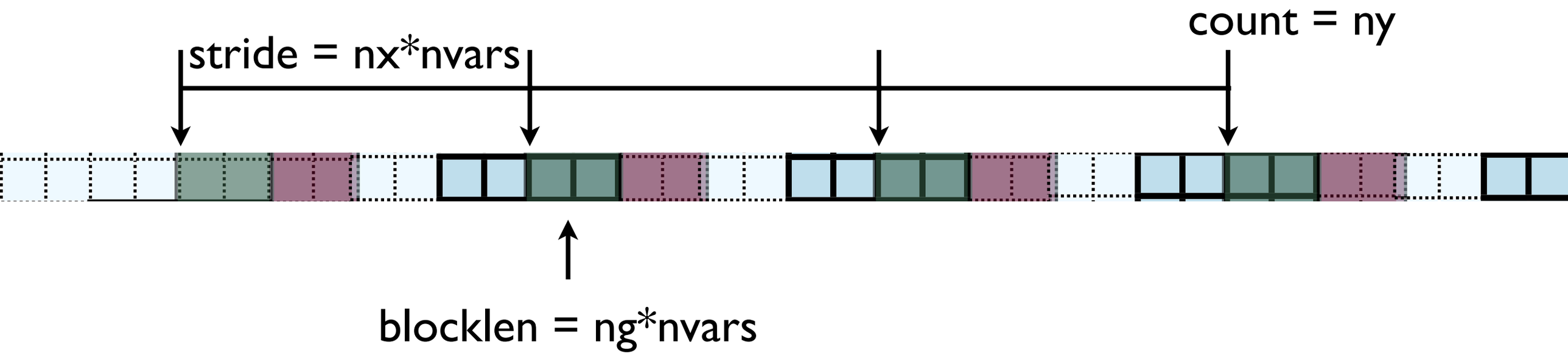


SciNet

Implementing in MPI

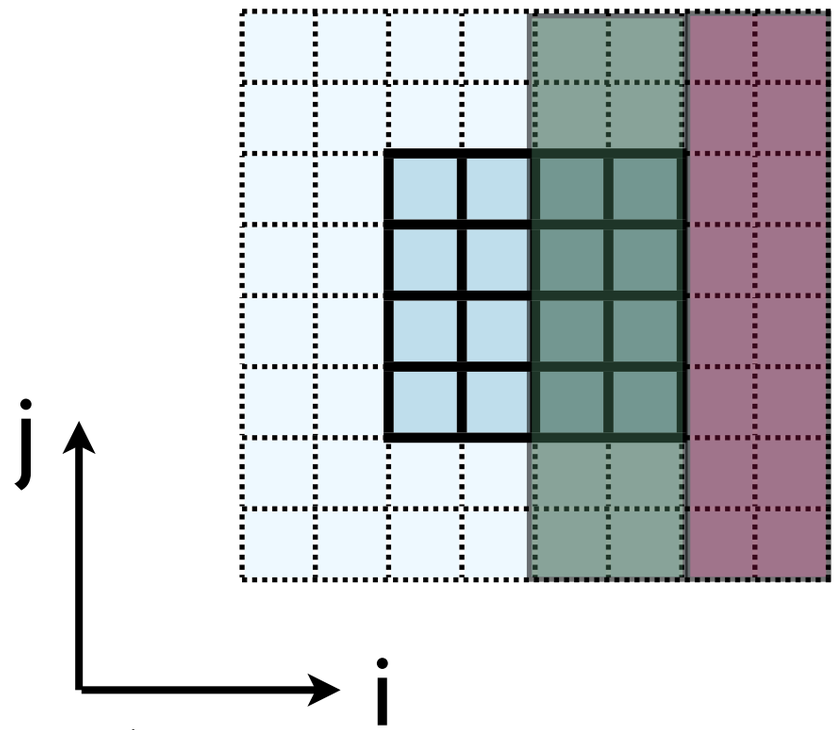
```
int MPI_Type_vector(
        int count,
        int blocklen,
        int stride,
        MPI_Datatype old_type,
        MPI_Datatype *newtype );
```

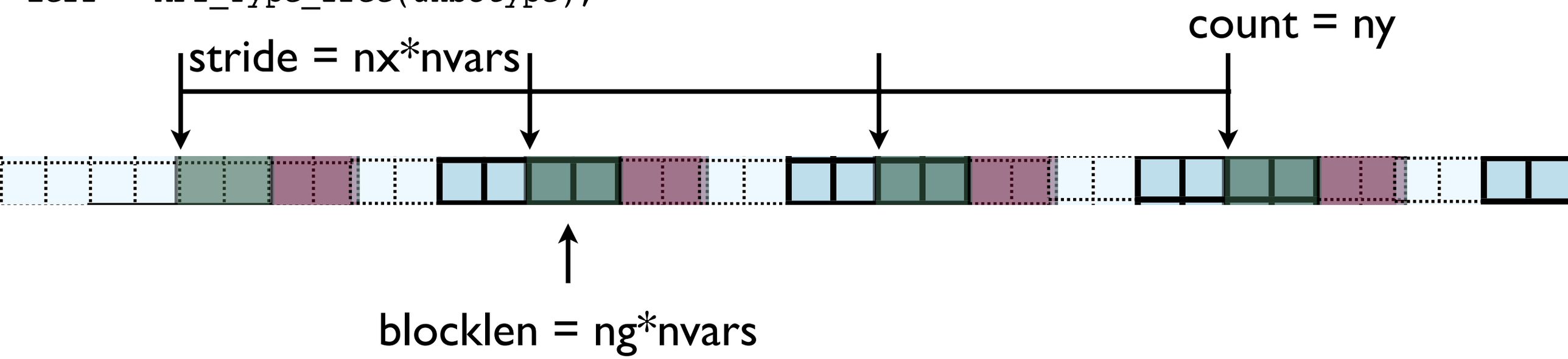




Implementing in MPI

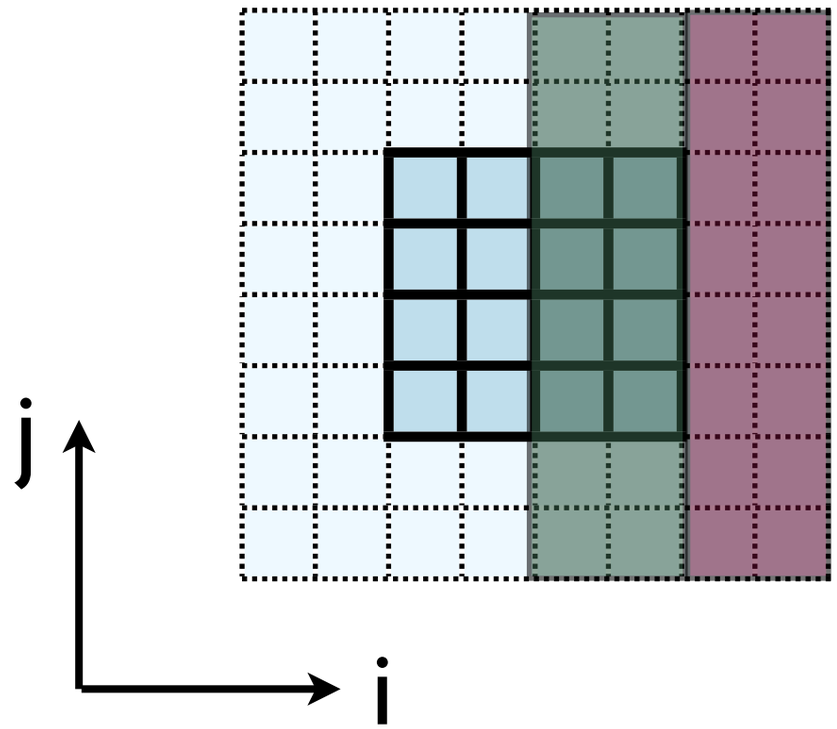
```
ierr = MPI_Type_vector(ny, nguard*nvars,
         nx*nvars, MPI_FLOAT, &xbctype);

ierr = MPI_Type_commit(&xbctype);

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

ierr = MPI_Type_free(&xbctype);
```

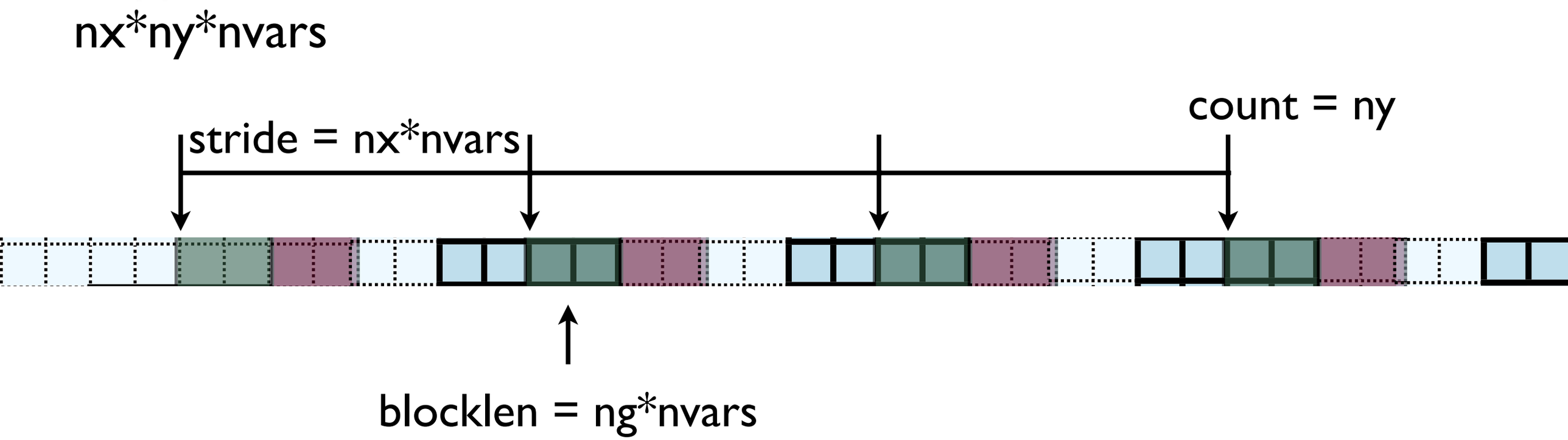




Implementing in MPI

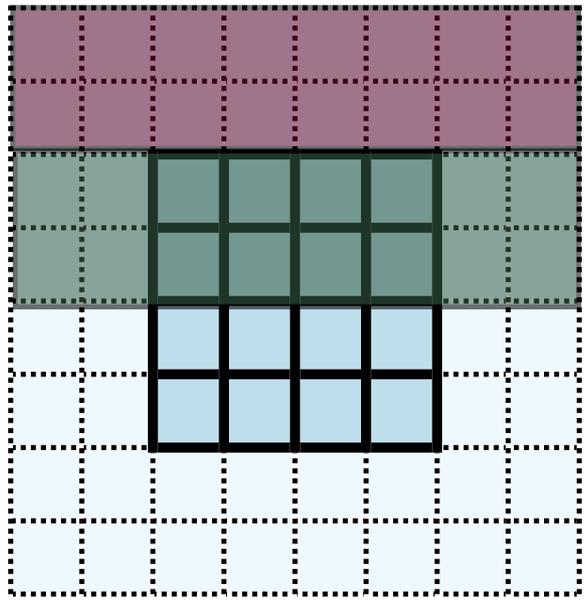
- Check: total amount of data = blocklen*count = ny*ng*nvars
- Skipped over stride*count = nx*ny*nvars





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...)



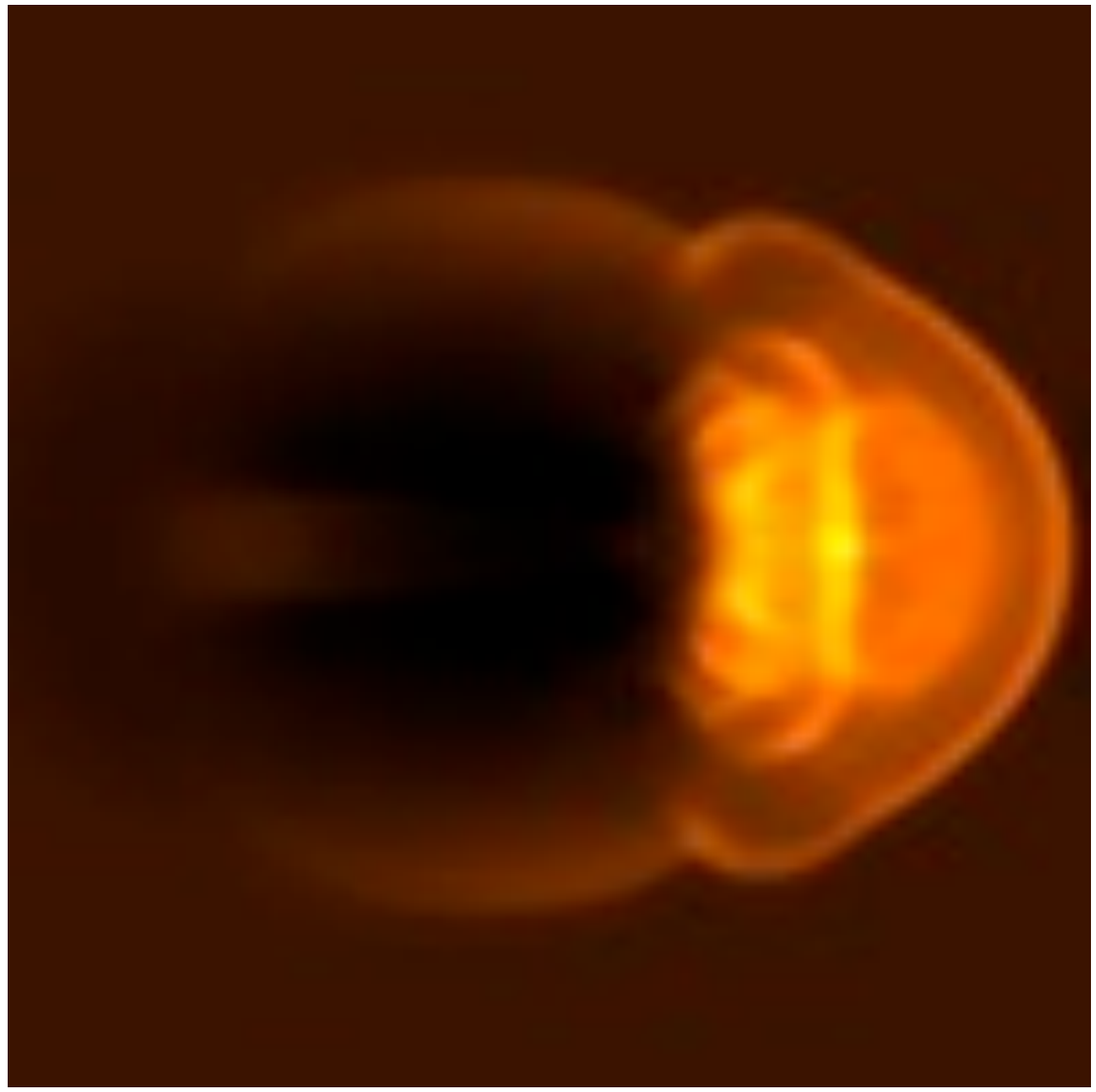
```
int MPI_Type_create_subarray(
      int ndims, int *array_of_sizes,
      int *array_of_subsizes,
      int *array_of_starts,
      int order,
      MPI_Datatype oldtype,
      MPI_Datatype &newtype);
```

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

SciNet

MPI-IO

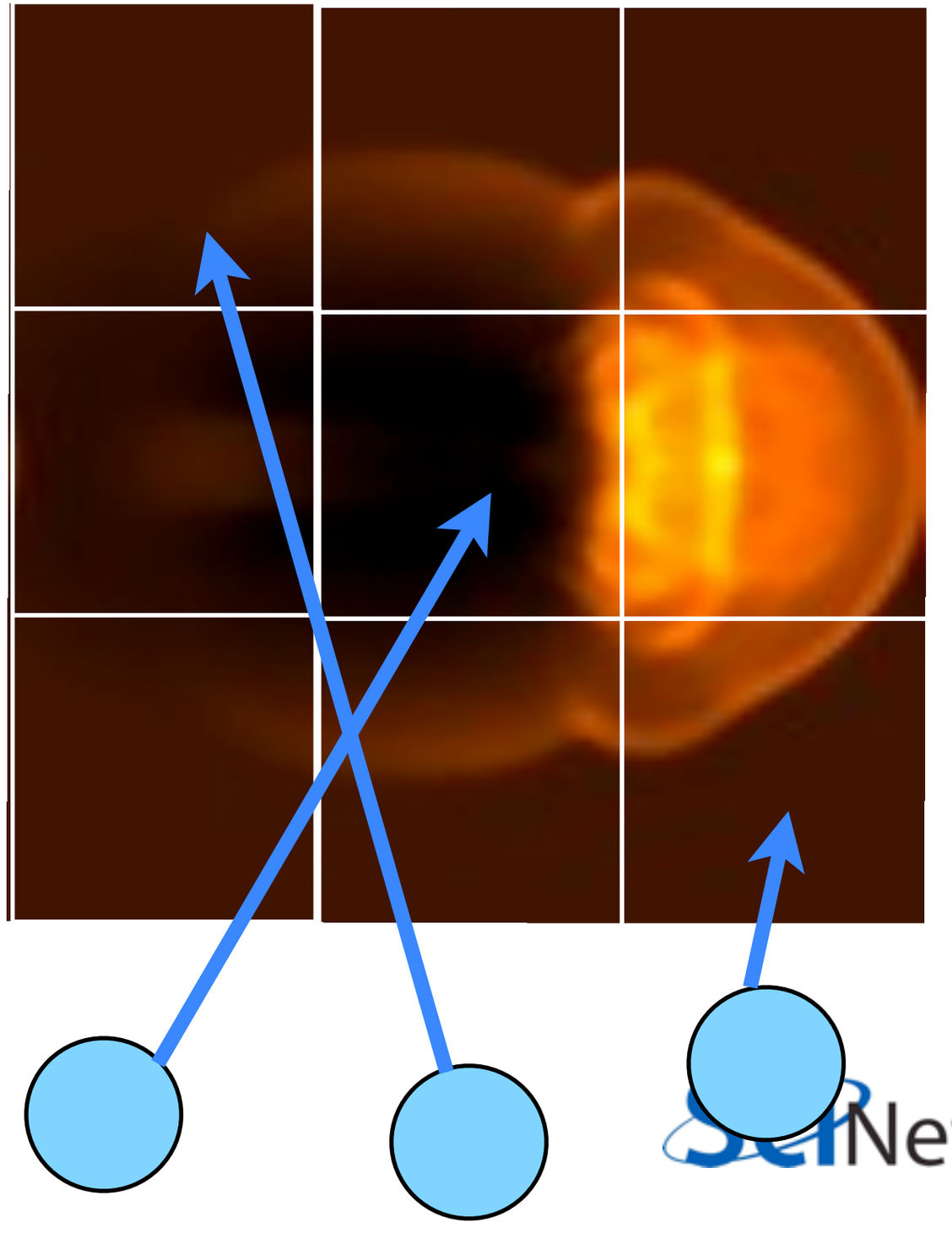
- Would like the new, parallel version to still be able to write out single output files.
- But at no point does a single processor have entire domain...



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

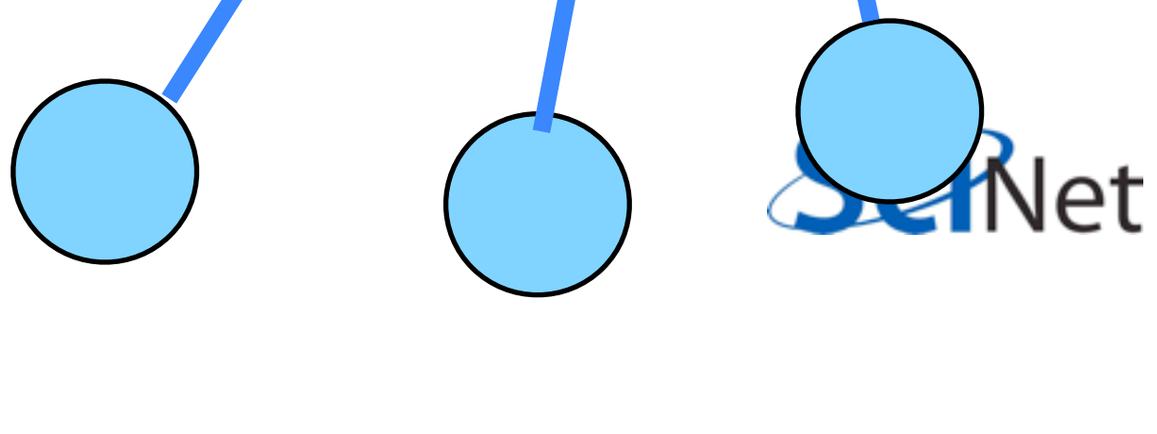
- Uses MPI to coordinate reading/writing to single file


```
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

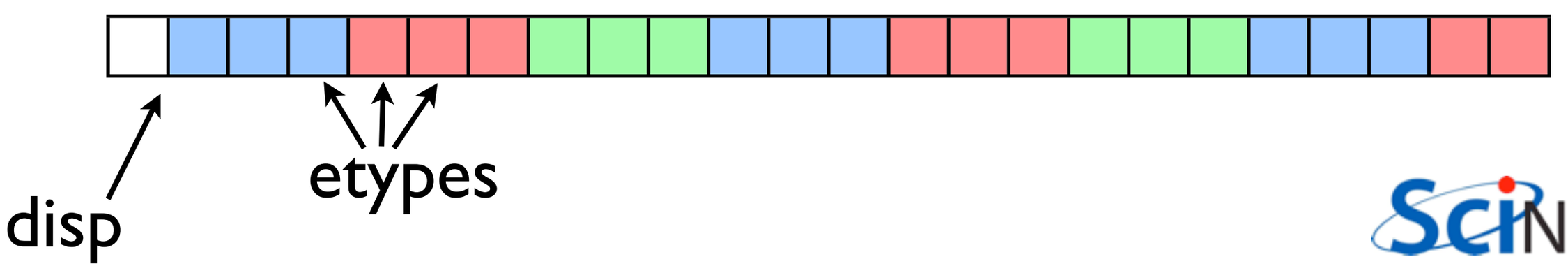
MPI-IO File View

- 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

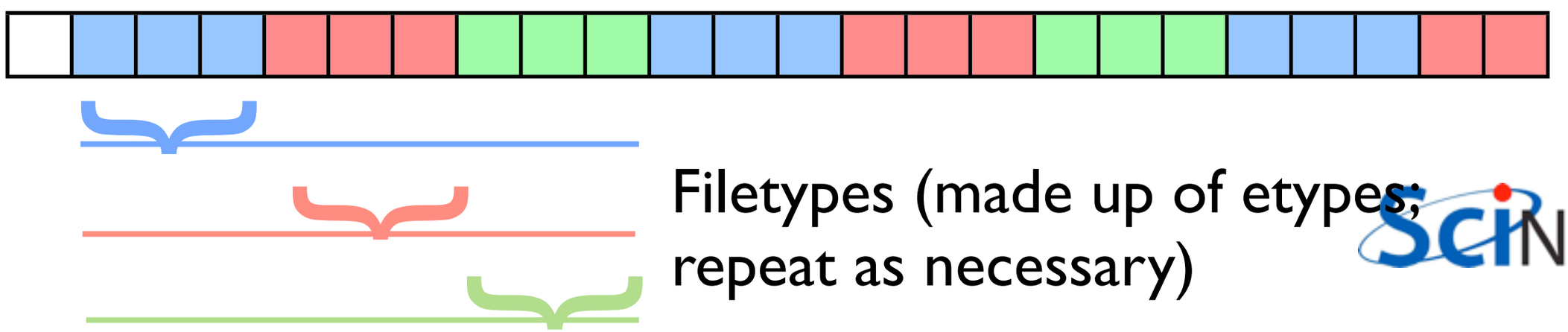
- int MPI_File_set_view(
  MPI_File fh,
  MPI_Offset disp, /* displacement in *bytes* from start */
  MPI_Datatype etype, /* elementary type */
  MPI_Datatype filetype, /* file type; prob different for each proc */
  char *datarep, /* 'native' or 'internal' */
  MPI_Info info) /* MPI_INFO_NULL for today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



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 */



Filetypes (made up of etypes; repeat as necessary)

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

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,...
```

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,...
```