MPI II: Nonblocking, Datatypes, and Hybrid

Scientific Computing III
High Performance Scientific Computing
Feb 2012

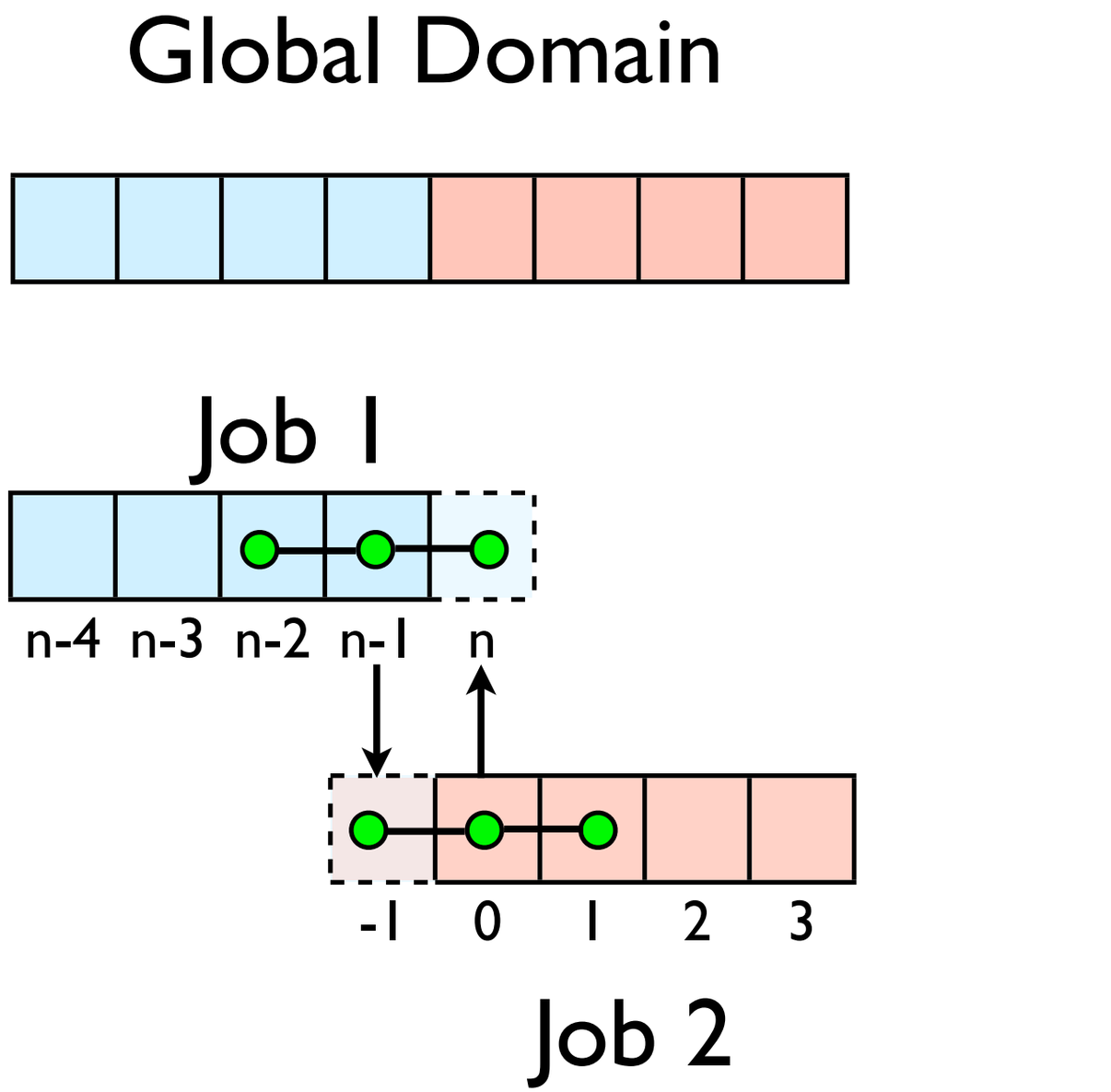
SciNet
compute • calcul
CANADA

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

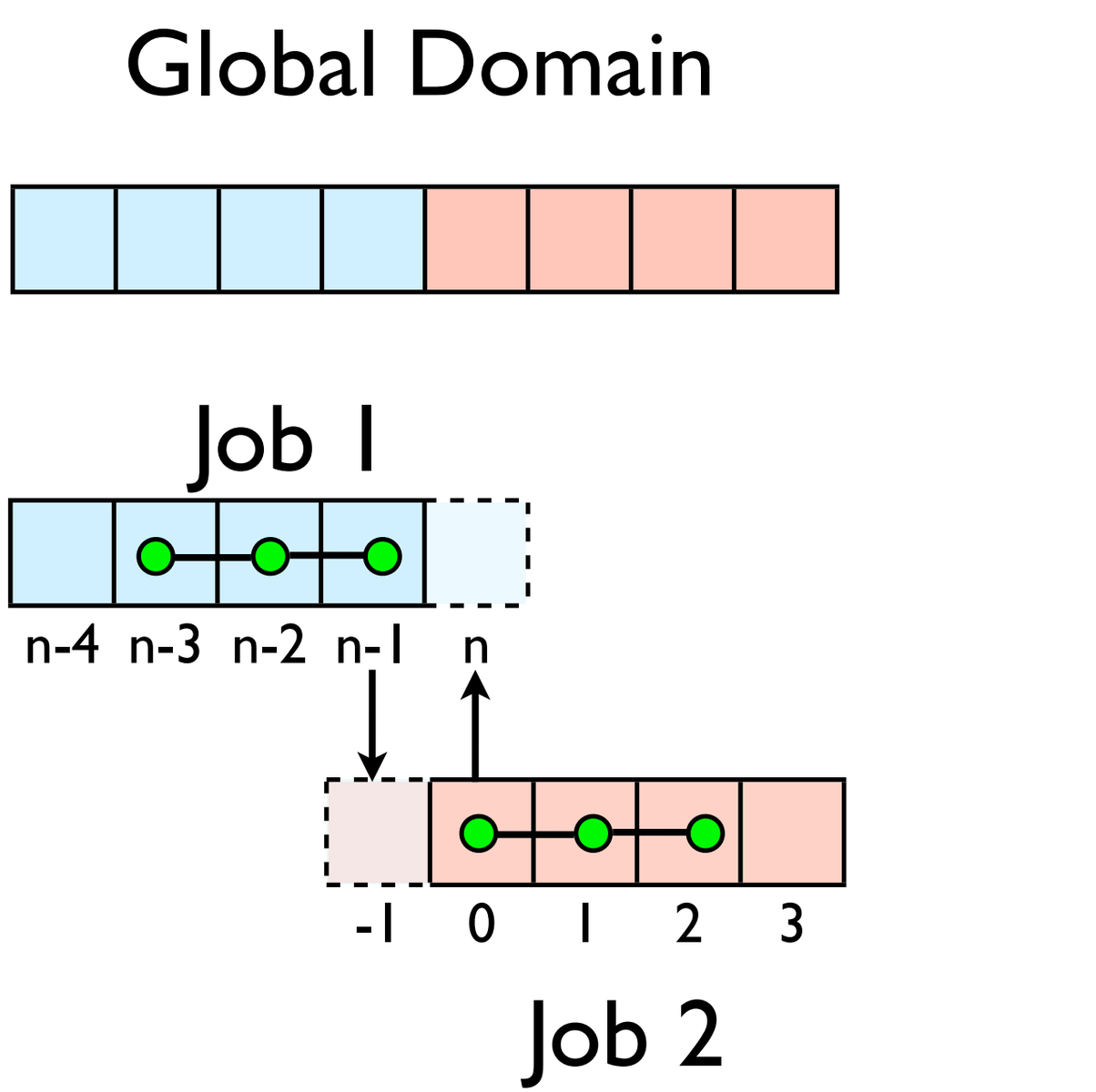


SciNet compute • calcul CANADA

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.



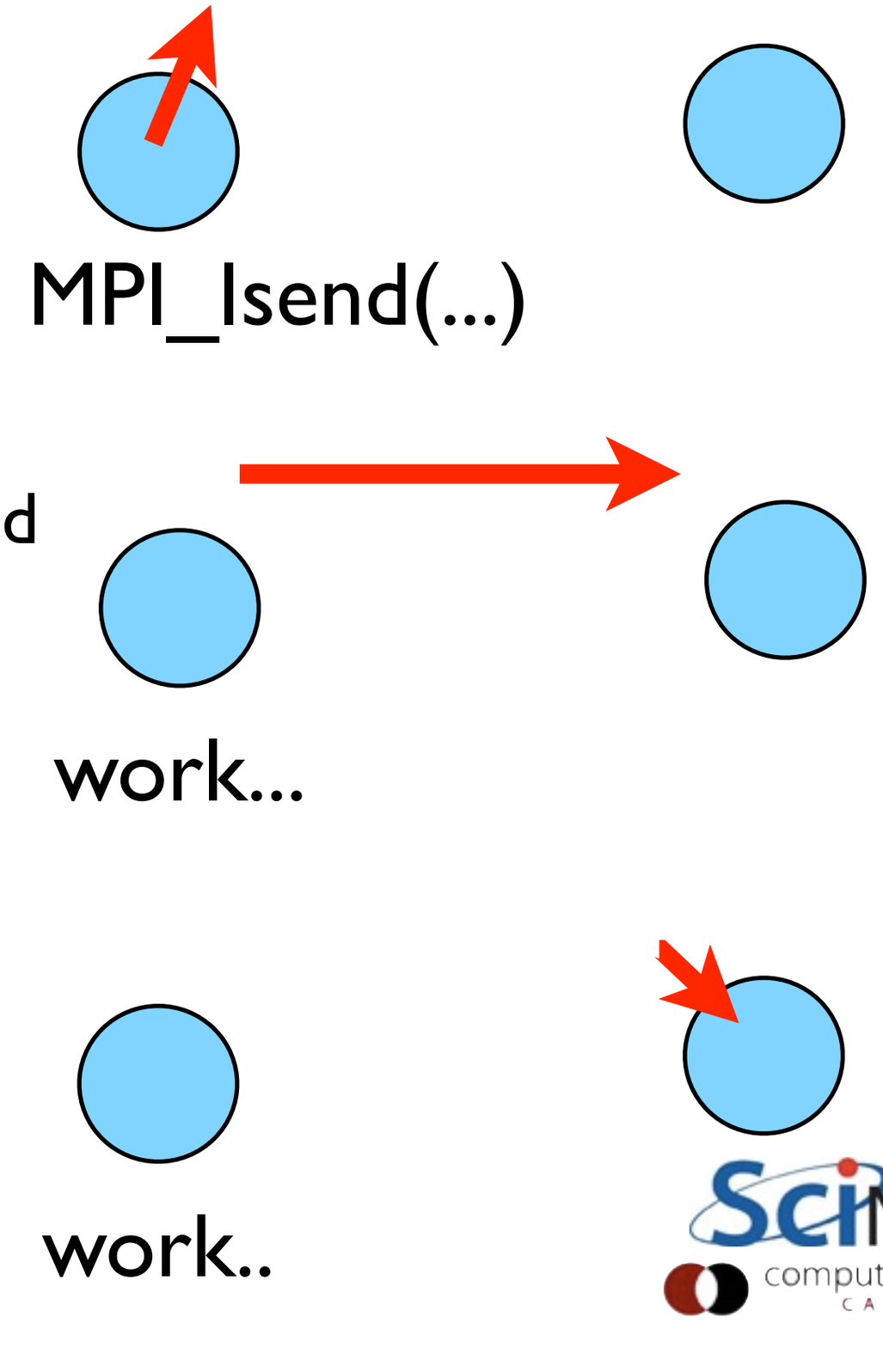
SciNet compute • calcul CANADA

Nonblocking Sends

- Allows you to get work done while message is 'in flight'

- Must **not** alter send buffer until send has completed.

- `MPI_Isend( void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm, MPI_Request *request )`

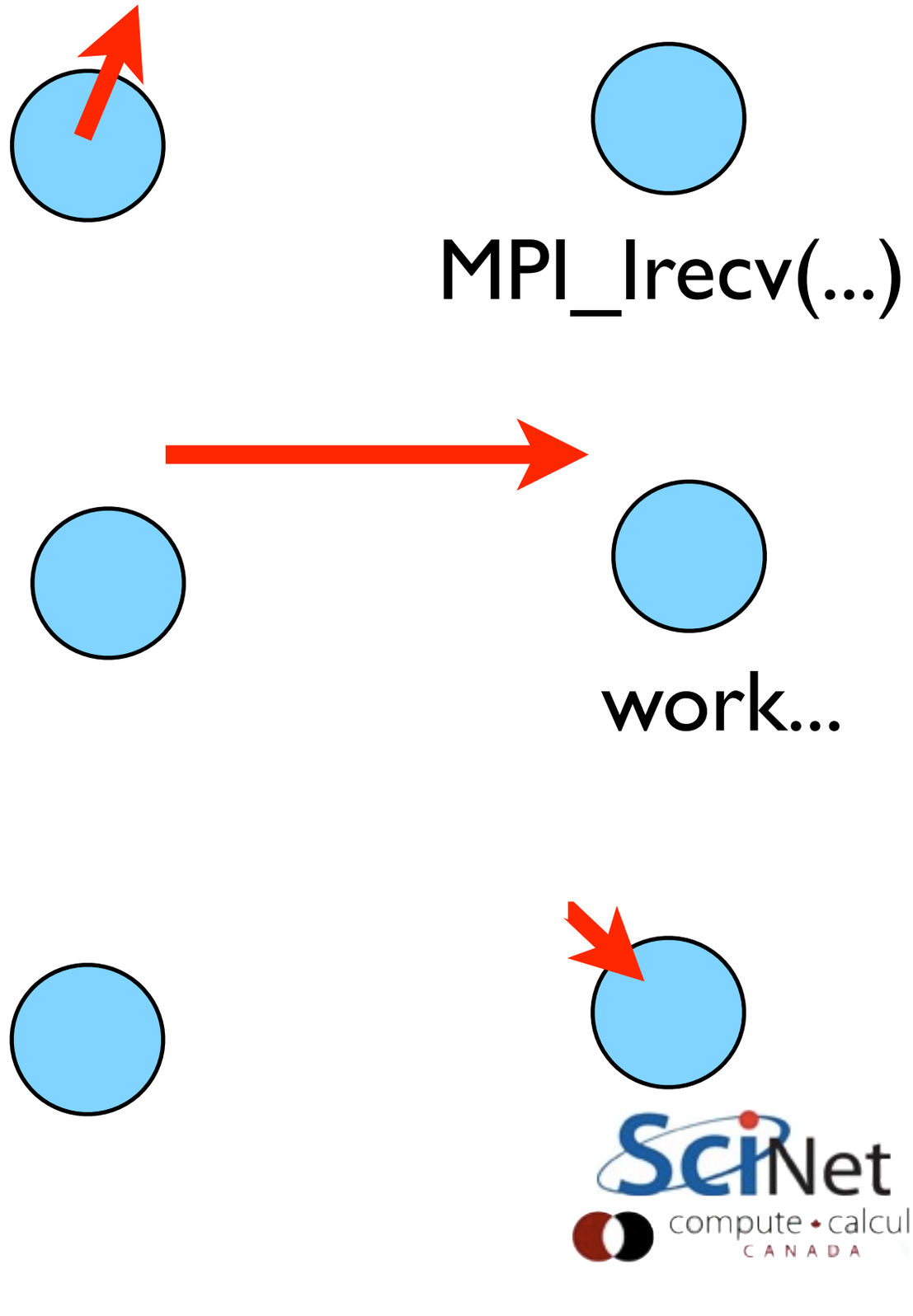


Nonblocking Recv

- Allows you to get work done while message is 'in flight'

- Must **not** access recv buffer until recv has completed.

- `MPI_Irecv( void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Request *request )`



How to tell if message is completed?

- `int MPI_Wait(MPI_Request *request,MPI_Status *status);`
- `int MPI_Waitall(int count,MPI_Request *array_of_requests, MPI_Status *array_of_statuses);`

Also: MPI_Waitany, MPI_Waitsome,MPI_Test...

Example

- git clone /scinet/course/sc3/hw4
- make nonblocking

```
left = rank-1;
if (left < 0) left = size-1;

right = rank+1;
if (right >= size) right = 0;

msgsent = 'A'+rank;
msgrcvd = '-';

/* launch isend/irecv */
ierr = MPI_Isend(&msgsent, 1, MPI_CHAR, right,
                 tag, MPI_COMM_WORLD, &reqs[0]);
ierr = MPI_Irecv(&msgrcvd, 1, MPI_DOUBLE, left,
                 tag, MPI_COMM_WORLD, &reqs[1]);

/* do some work */

/* now wait for the messages to send */
MPI_Waitall(2, reqs, stats);

printf("%d: Got %c, Sent %c\n",
            rank, msgrcvd, msgsent);

MPI_Finalize();
```

Example

- git clone /scinet/course/sc3/hw4
- make nonblocking

```
gpc-f103n084-$ make nonblocking
mpicc -o nonblocking nonblocking.c -std=c99 -Wall -O2 -g
gpc-f103n084-$ mpirun -np 7 nonblocking
1: Got A, Sent B
2: Got B, Sent C
5: Got E, Sent F
0: Got G, Sent A
3: Got C, Sent D
4: Got D, Sent E
6: Got F, Sent G
gpc-f103n084-$
```

SciNet compute • calcul CANADA

Notes

- This was a cycle of sends/recvs. Why does that matter?

- A blocking send can be thought of as an Isend immediately followed by a Wait.

```
left = rank-1;
if (left < 0) left = size-1;

right = rank+1;
if (right >= size) right = 0;

msgsent = 'A'+rank;
msgrcvd = '-';

/* launch isend/irecv */
ierr = MPI_Isend(&msgsent, 1, MPI_CHAR, right,
                 tag, MPI_COMM_WORLD, &reqs[0]);
ierr = MPI_Irecv(&msgrcvd, 1, MPI_DOUBLE, left,
                 tag, MPI_COMM_WORLD, &reqs[1]);

/* do some work */

/* now wait for the messages to send */
MPI_Waitall(2, reqs, stats);

printf("%d: Got %c, Sent %c\n",
            rank, msgrcvd, msgsent);

MPI_Finalize();
```

SciNet compute • calcul CANADA

Reasons to use Nonblocking

- Avoid deadlock
- Overlap communications and computation.
- (Note: most MPI implementations won't do much overlapping with ethernet/tcp; but IB, or maybe shared memory messages.)

```
left = rank-1;
if (left < 0) left = size-1;

right = rank+1;
if (right >= size) right = 0;

msgsent = 'A'+rank;
msgrcvd = '-';

/* launch isend/irecv */
ierr = MPI_Isend(&msgsent, 1, MPI_CHAR, right,
                 tag, MPI_COMM_WORLD, &reqs[0]);
ierr = MPI_Irecv(&msgrcvd, 1, MPI_DOUBLE, left,
                 tag, MPI_COMM_WORLD, &reqs[1]);

/* do some work */

/* now wait for the messages to send */
MPI_Waitall(2, reqs, stats);

printf("%d: Got %c, Sent %c\n",
            rank, msgrcvd, msgsent);

MPI_Finalize();
```

SciNet compute • calcul CANADA

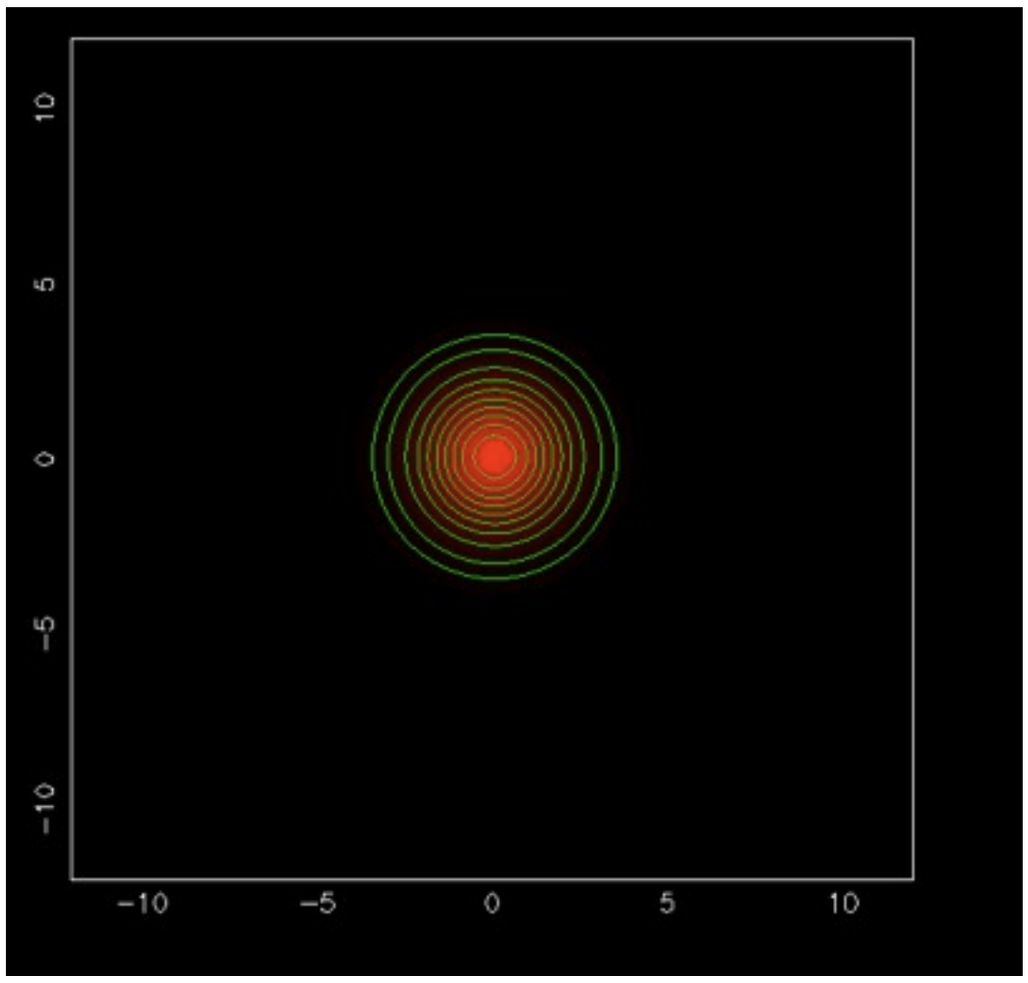
Nonblocking in diffusion-mpi.c

- Diffusion-mpi.c is the answer to hw2.
- How would we use nonblocking sends and receives here to overlap communication and computation?

```
/* set these points; internal boundarie
temperature[old][0] = fixedlefttemp;
temperature[old][locpoints+1] = fixedri

/* do internal boundaries */
/* send last real data point rightward,
const int righttag = 1;
MPI_Status status;

MPI_Sendrecv(&(temperature[old][locpoin
             &(temperature[old][0]),
             MPI_COMM_WORLD, &status);

/* send first real data point leftward,
const int lefttag = 2;
MPI_Sendrecv(&(temperature[old][1]),
             &(temperature[old][locpoin
             MPI_COMM_WORLD, &status);

for (i=1; i<locpoints+1; i++) {
    temperature[new][i] = temperature[o
        (temperature[old][i+1] - 2.*tem
         temperature[old][i-1]) ;
}
```

Datatypes for more complex messages

- Diffusion2d is the generalization of the 1d problem from hw2.
- Calculates diffusion equation in two dimensions.

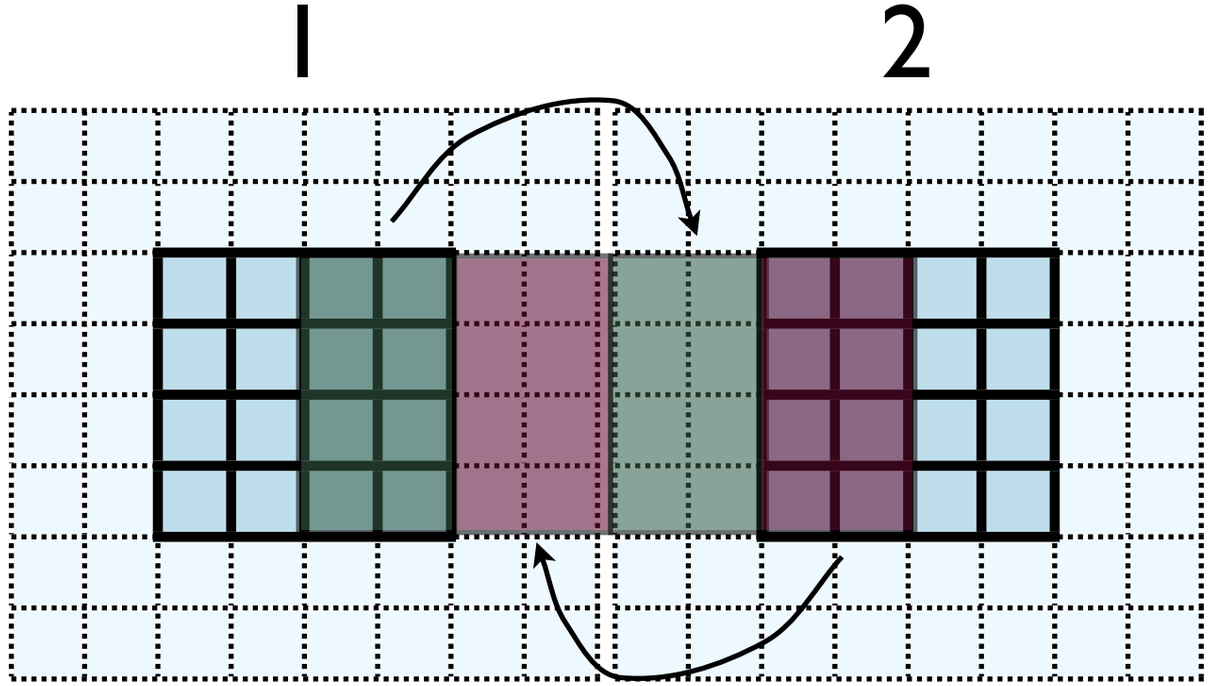



```
gpc-f103n084-$ make diffusion2d
gpc-f103n084-$ ./diffusion2d
```

SciNet compute • calcul CANADA

Guard cell fill

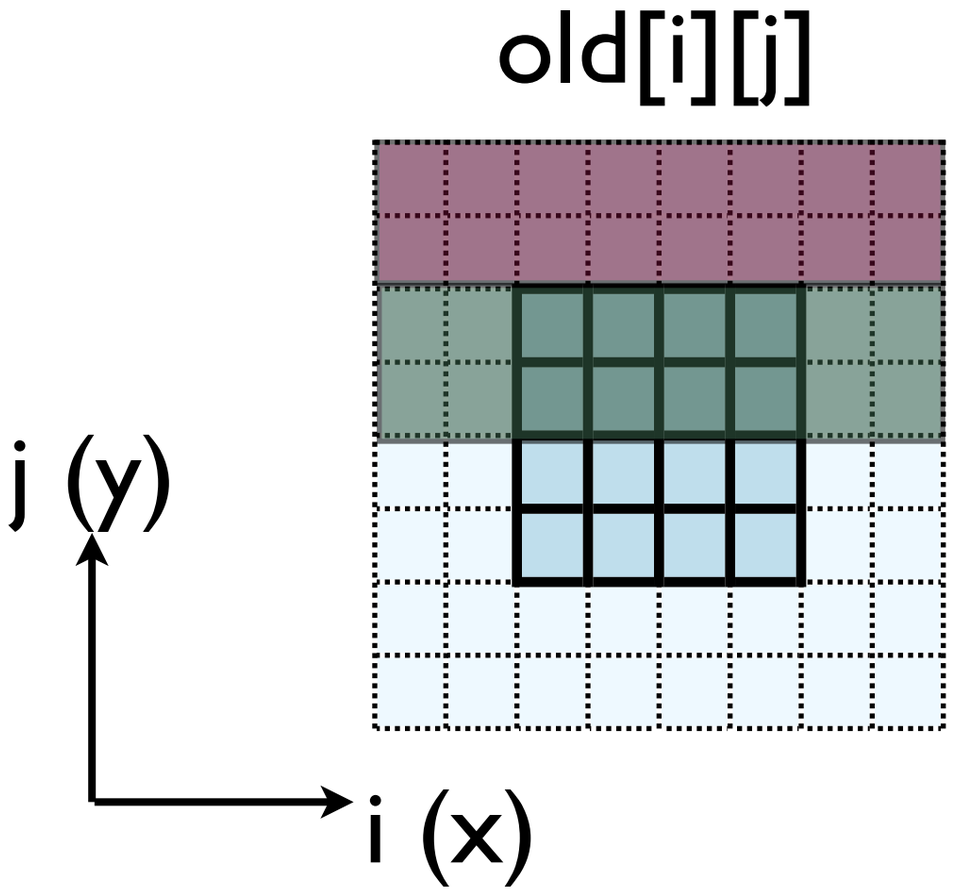
- Basic idea is same as in 1d
- Copy data into guardcells in boundary-condition phase.
- For generality, have more than 1 level of guardcells illustrated here (say, ng) but only need 1 in this code.



SciNet
compute • calcul
CANADA

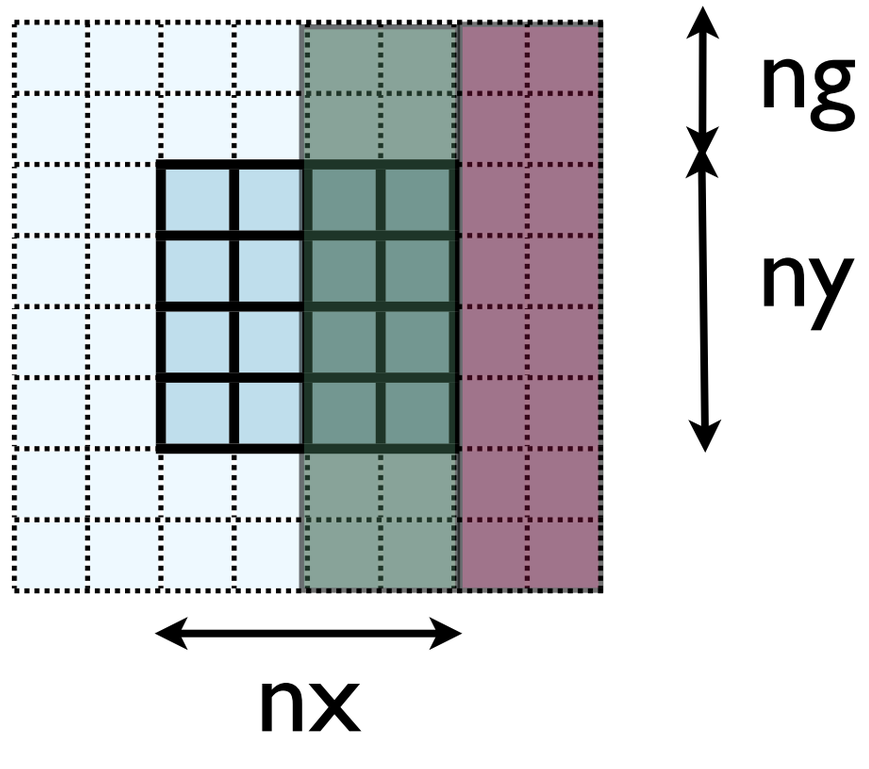
2d Guardcells

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



&old[0][0]

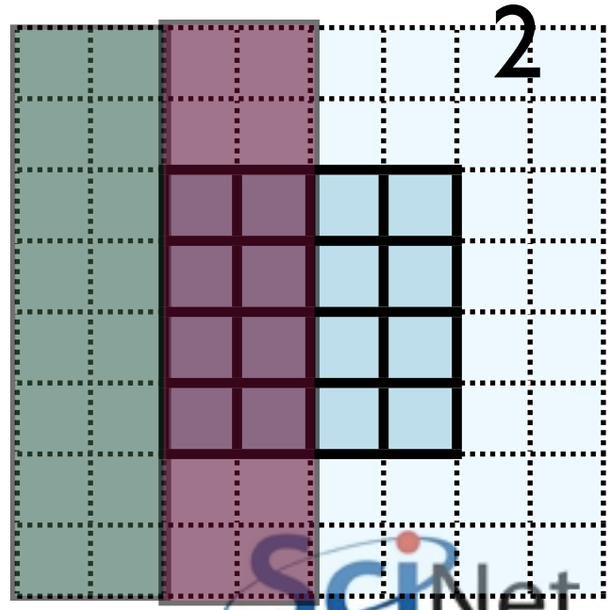
2D Guardcells



- If we're sending our left/ right data to our neighbour, pretty easy

- Send count = ng*(ny+2*ng) contiguous values, recv same.

- MPI_Send(&(old[nx][0]), count, MPI_FLOAT,....)

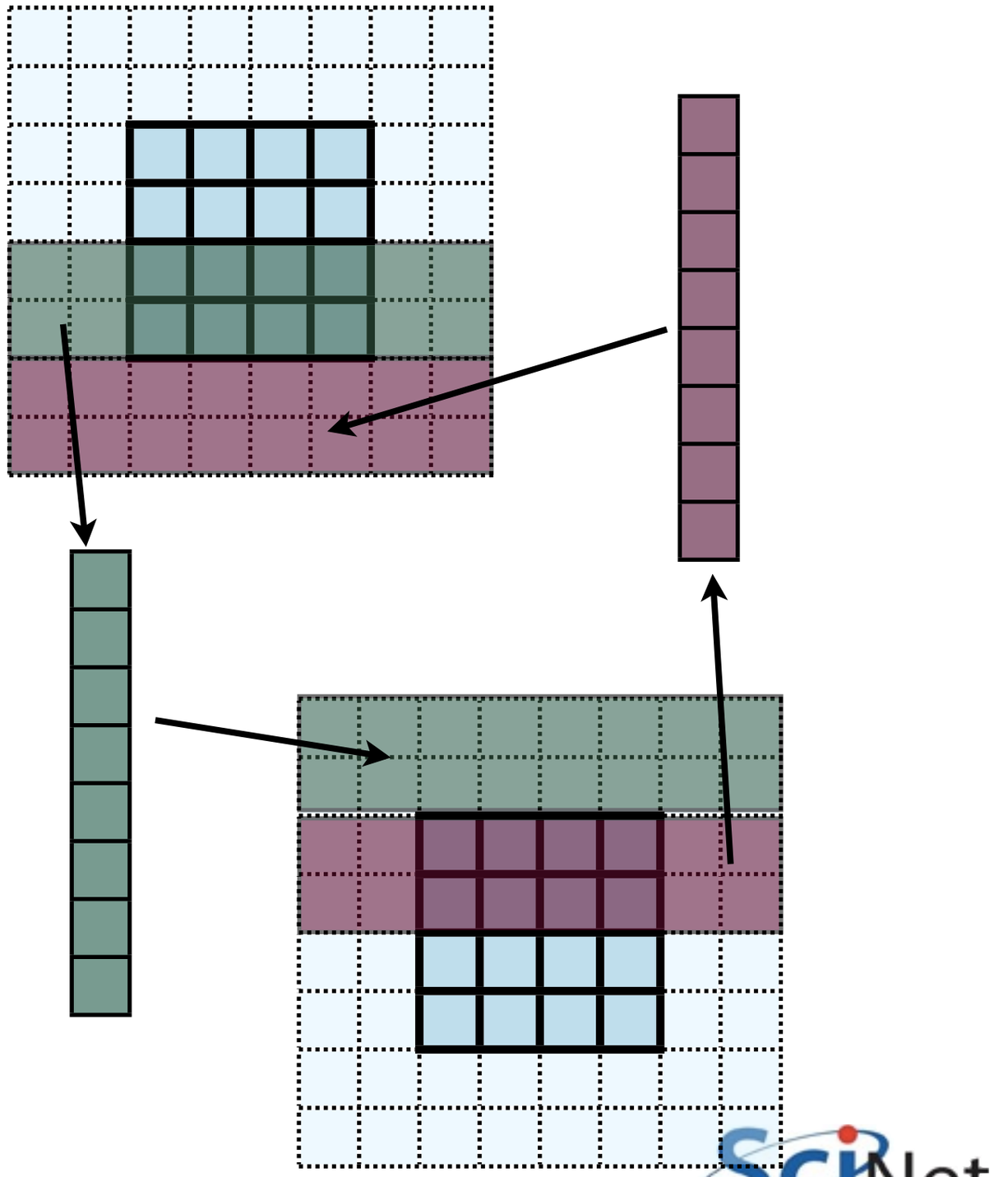


SciNet
compute • calcul
CANADA

What if non-contiguous?

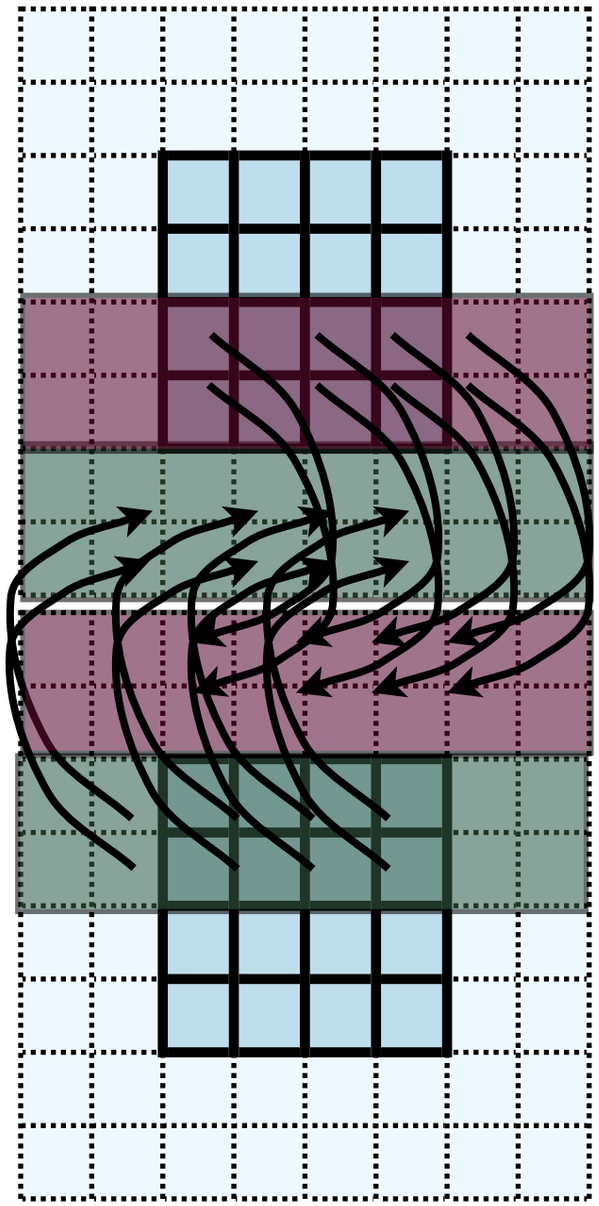
- But how do we do the up/down boundary conditions?

- Data non-contiguous in memory.



What if non-contiguous?

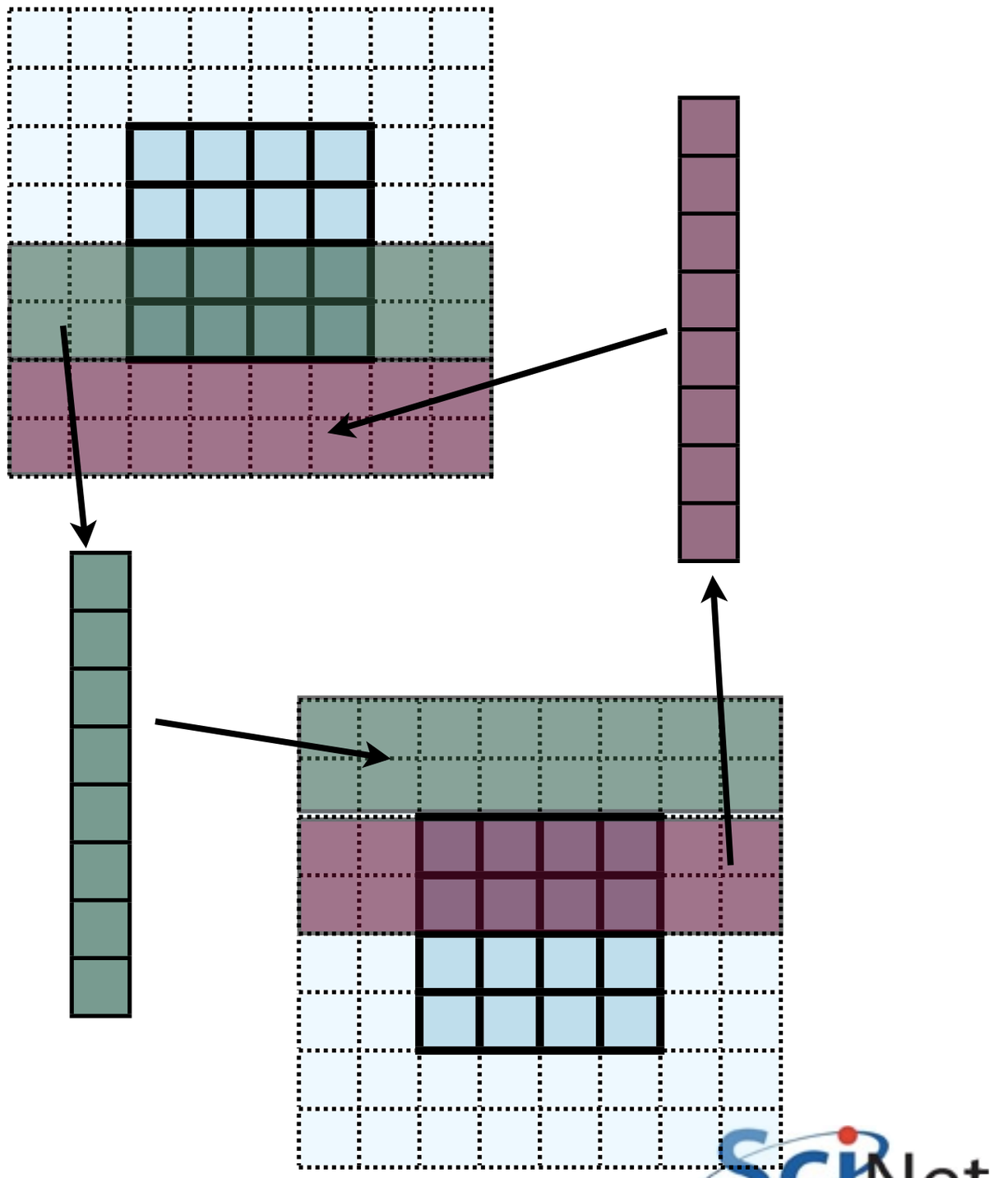
- One way:
- Loop over values, sending each one.
- Latency hit for each message.
- Would completely dominate communications cost.
- Terrible idea.



What if non-contiguous?

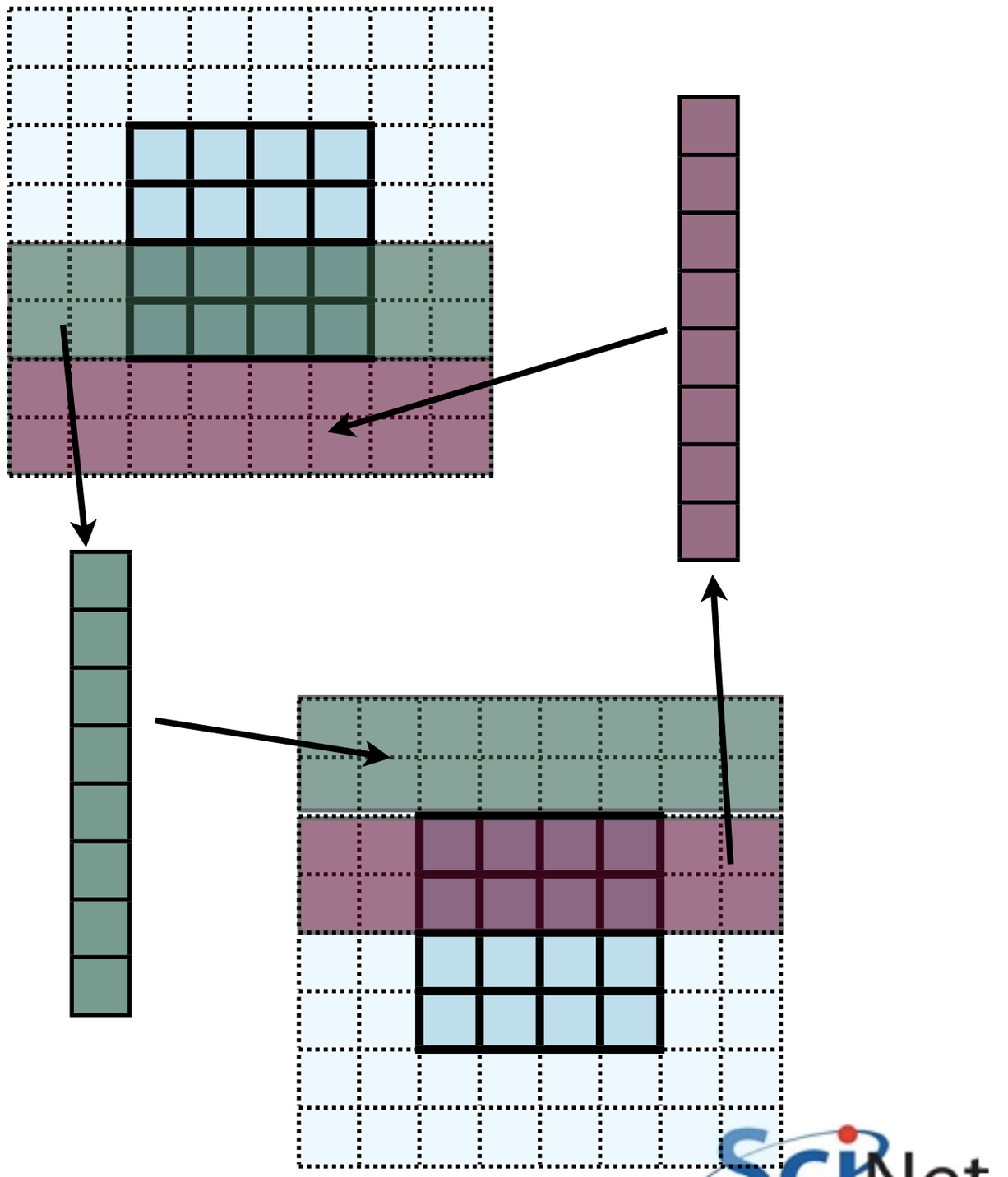
- Another way:

- Copy data into a buffer, send once; receive into a buffer, unpack into array.

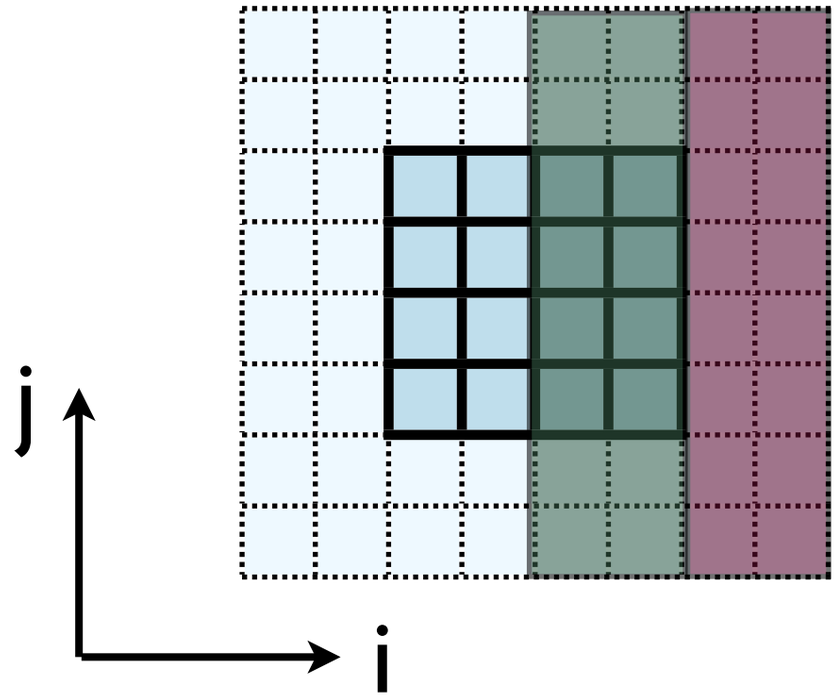


What if non-contiguous?

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

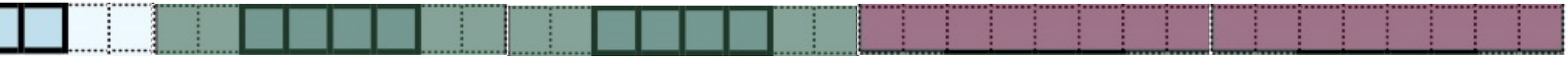


Contiguous case:

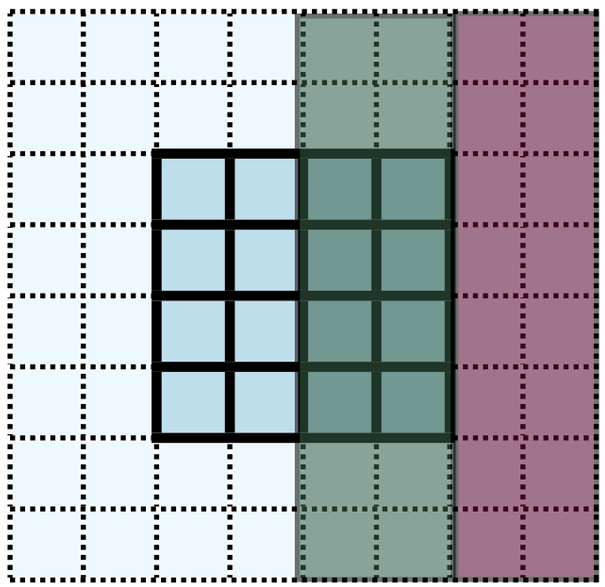


- Let's look back at the left/ right case.
- Can send in one go:

```
count = ng*(ny+2*ng);
MPI_Send(&(old[nx][0]), count, MPI_FLOAT, right, ....)
```



MPI Data Types



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

Count OldType &NewType

```
MPI_Datatype lrgctype;

MPI_Type_contiguous(ng*(ny+2*ng), MPI_REAL, &lrgctype);
ierr = MPI_Type_commit(&lrgctype);

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

ierr = MPI_Type_free(&lrgctype);
```

SciNet compute • calcul CANADA

Type workflow

- Create a type with MPI_Type_... calls

- Commit it when done (you can modify the type as building it, commit only final version)

- Use it as any other type.

- Free when done.

```
MPI_Datatype lrgctype;

MPI_Type_contiguous(ng*(ny+2*ng), MPI_REAL, &lrg
ierr = MPI_Type_commit(&lrgctype);

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

ierr = MPI_Type_free(&lrgctype);
```

SciNet compute • calcul CANADA

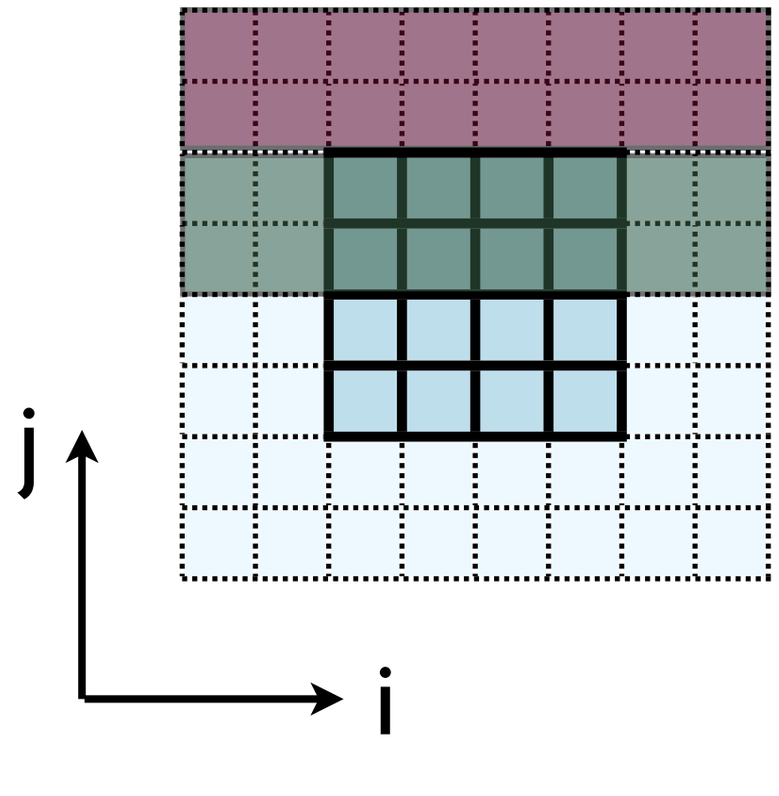
Three Types of MPI Functionality:

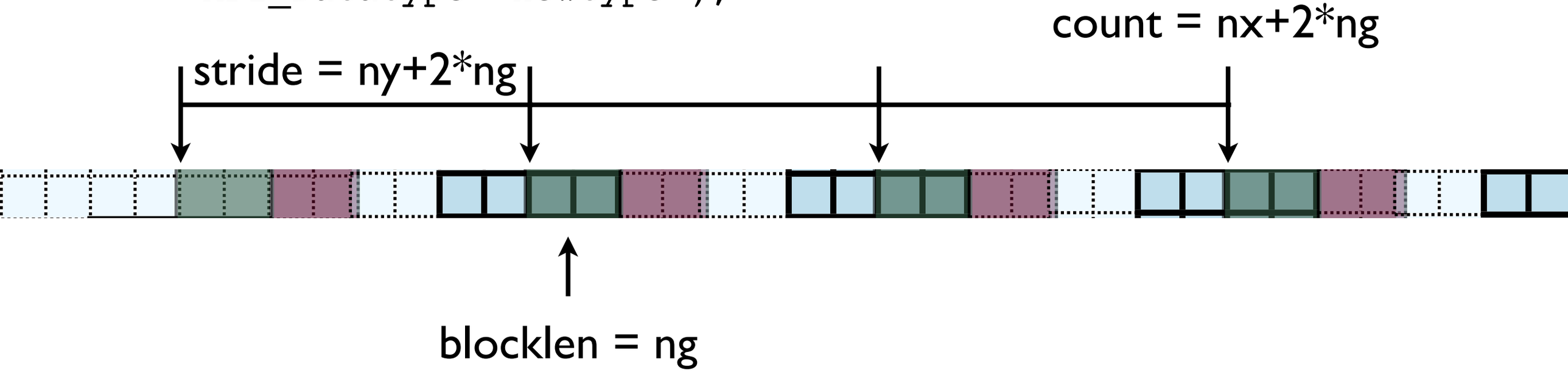
- Point to point
- Collectives
- Routines to allow efficient transfers in, out of memory.

```
MPI_Datatype lrgctype;

MPI_Type_contiguous(ng*(ny+2*ng), MPI_REAL
ierr = MPI_Type_commit(&lrgctype);

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

ierr = MPI_Type_free(&lrgctype);
```

Noncontiguous Case

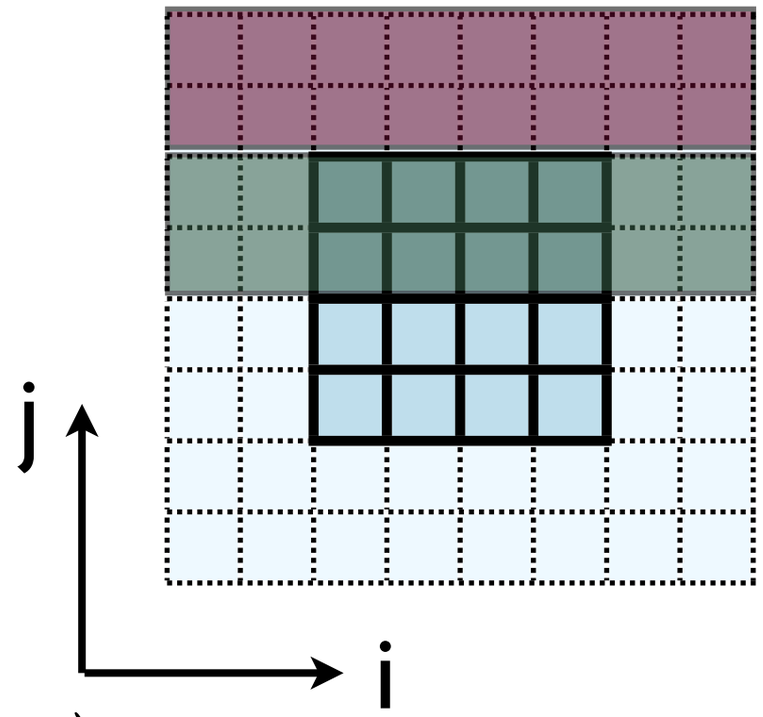
```
int MPI_Type_vector(
        int count,
        int blocklen,
        int stride,
        MPI_Datatype old_type,
        MPI_Datatype *newtype );
```

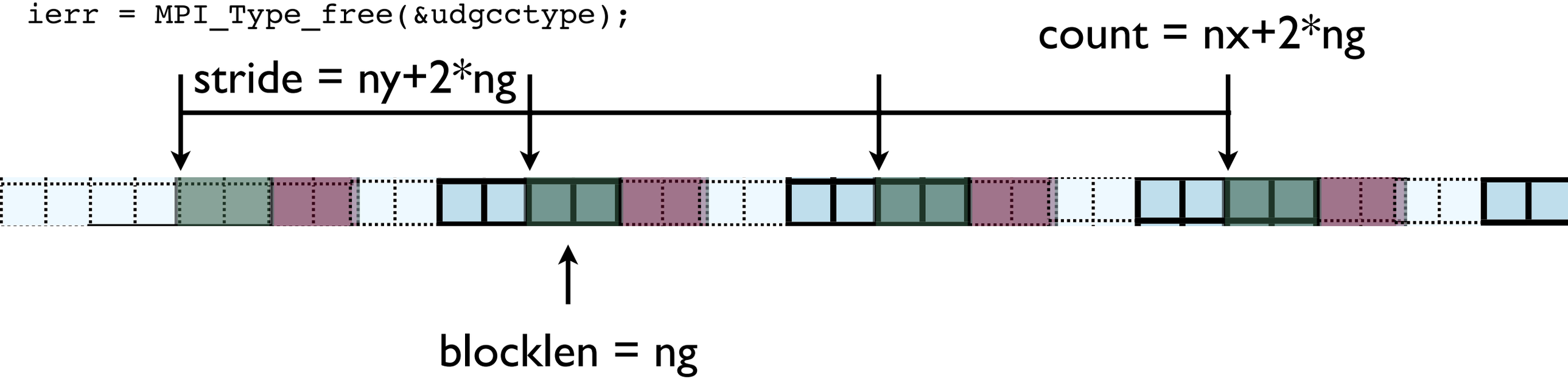




MPI_Type_vector

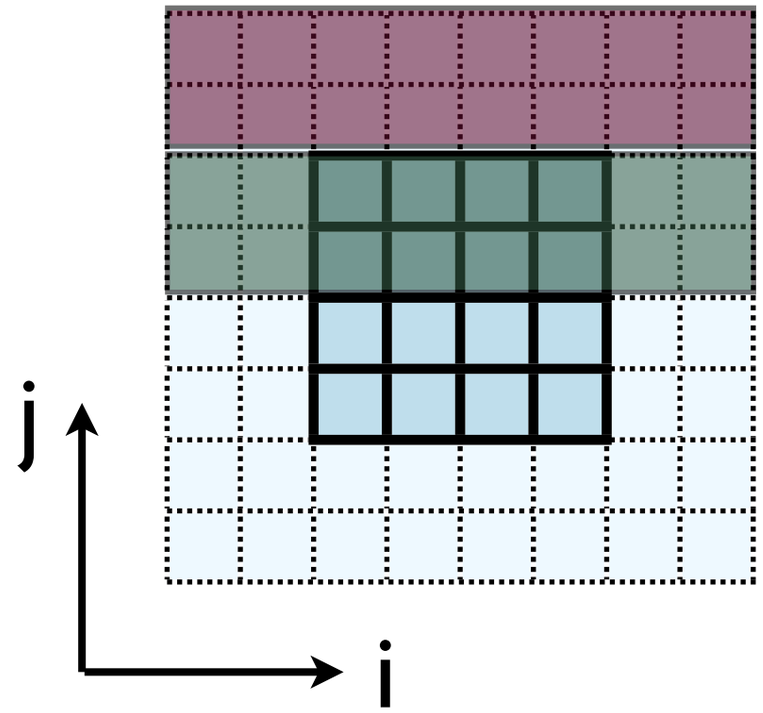
```
ierr = MPI_Type_vector(ny+2*ng, ng,
         nx+2*ng, MPI_FLOAT, &udgctype);

ierr = MPI_Type_commit(&udgctype);

ierr = MPI_Send(&(u[0][ny]), 1, udgctype, ....)

ierr = MPI_Type_free(&udgcctype);
```

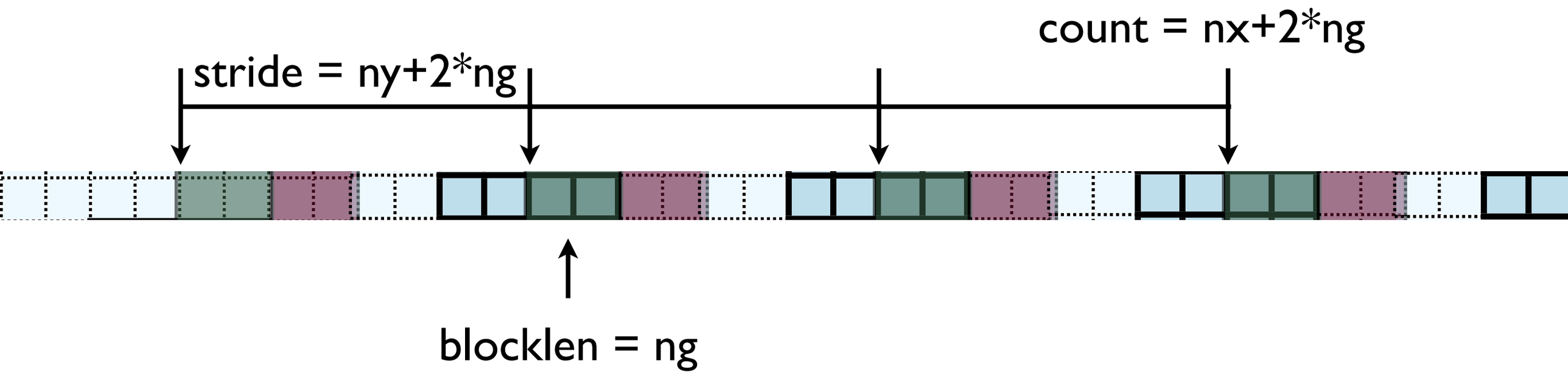




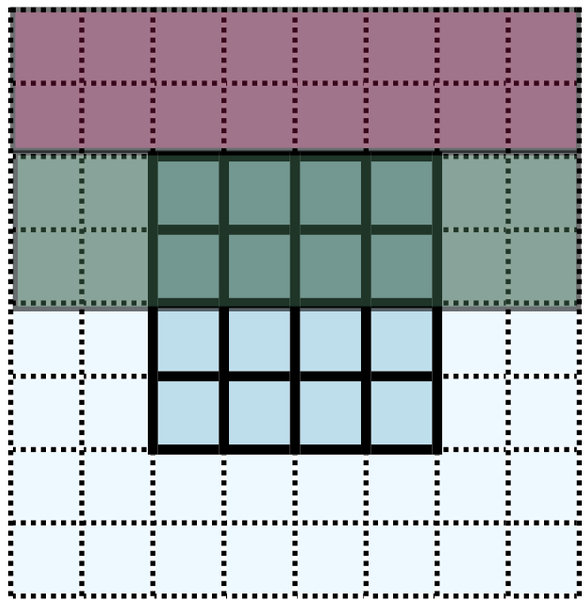
MPI_Type_vector

- Check: total amount of data = blocklen*count = ng*(nx+2*ng)
- Skipped over stride*count = (nx+2*ng)*(ny+2*ng)





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

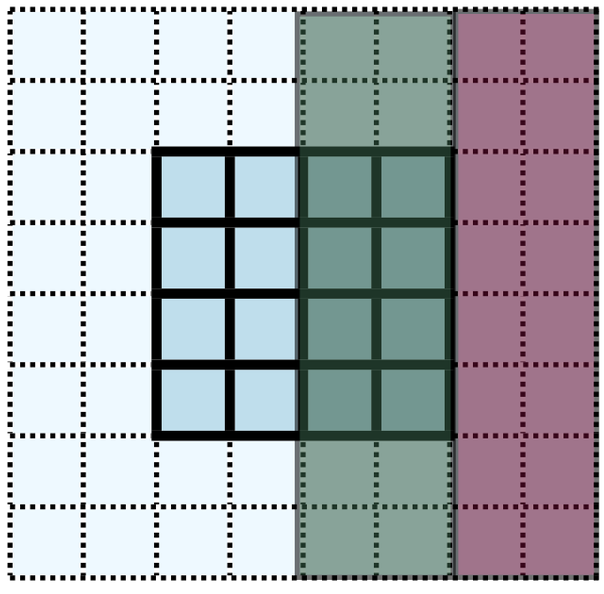


- MPI_Type_create_subarray ; piece of a multi-dimensional array.
- *Much* more convenient for higher-dimensional arrays
- (Otherwise, need vectors of vectors of vectors...)

```
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);
```

SciNet compute • calcul CANADA

```
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);
```

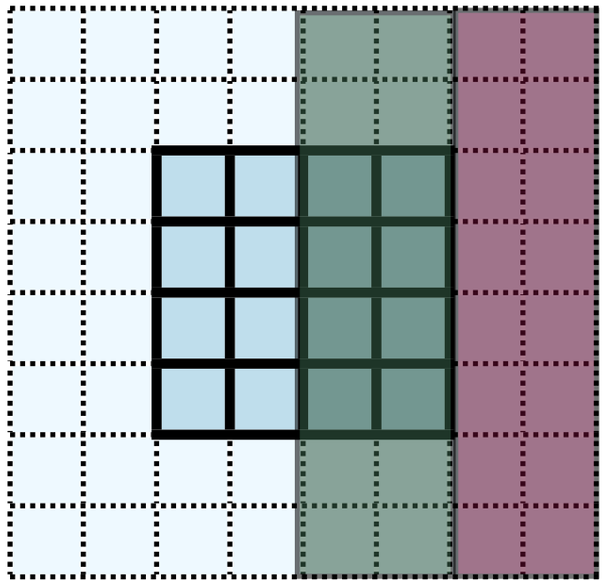


```
sizes[2] = {nx+2*ng, ny+2*ng};
subsizes[2] = {ng, ny+2*ng};
starts[2] = {nx,0};
```

Can also set starts at (say) {0,0}, and just point send buffer to first place to send.

SciNet compute • calcul CANADA

Implementing in MPI



- Hands-On:
- In diffusion2d-mpi, implement left/right guardcellfilling, and up/down filling with types.

- For now, create/free type each cycle through; ideally, we'd create/free these once.

SciNet compute • calcul CANADA

More complicated still?

- Not just a multiple of a single data type
- Contiguous, vector, subarray types won't do it.

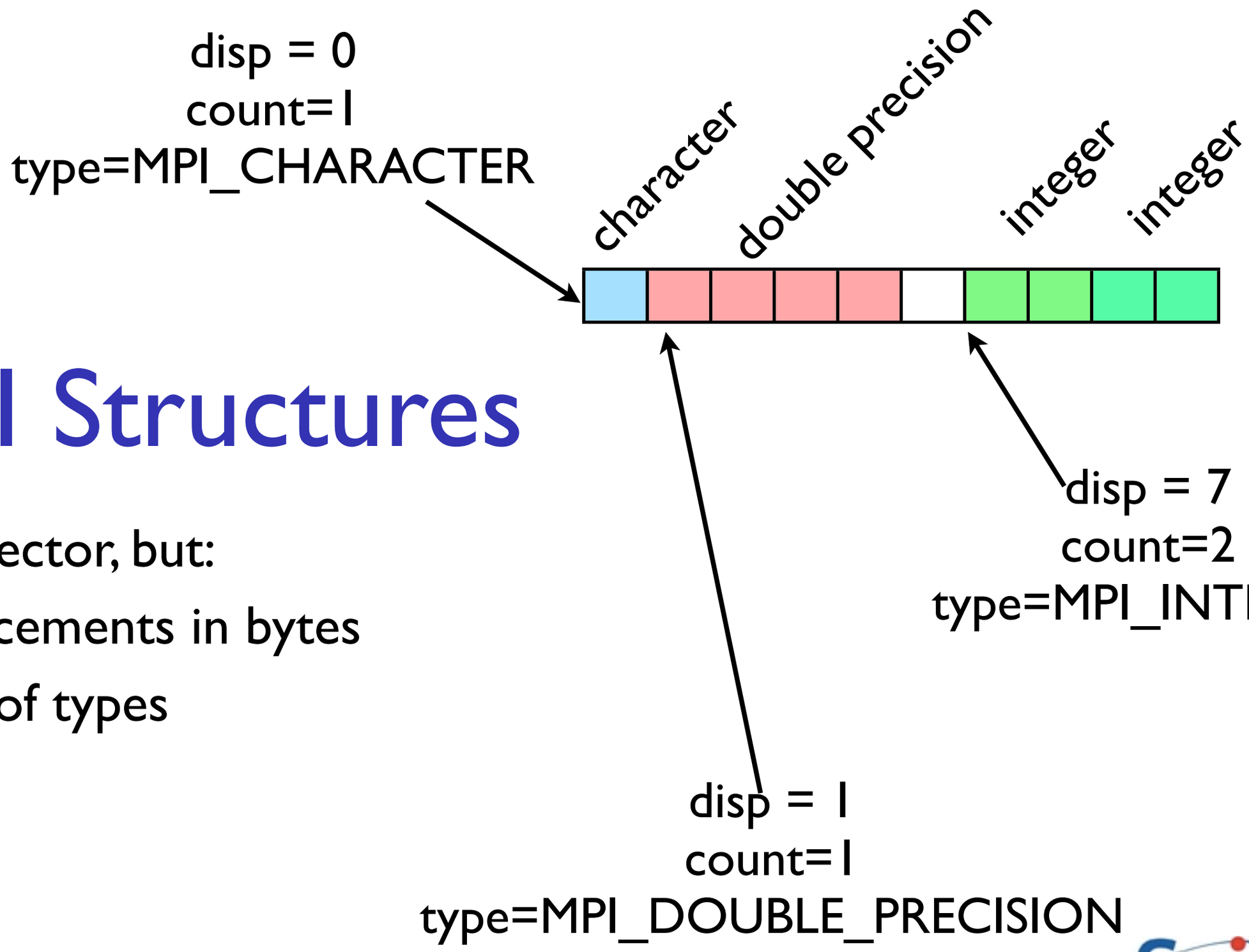
```
typedef struct domain_s {
    float xleft, xright;
    float yleft, yright;
    int nxpts, nypts;
    float dx, dy;
    float **temp1, **temp2;
    float **old, **new;
} domain_t;
```

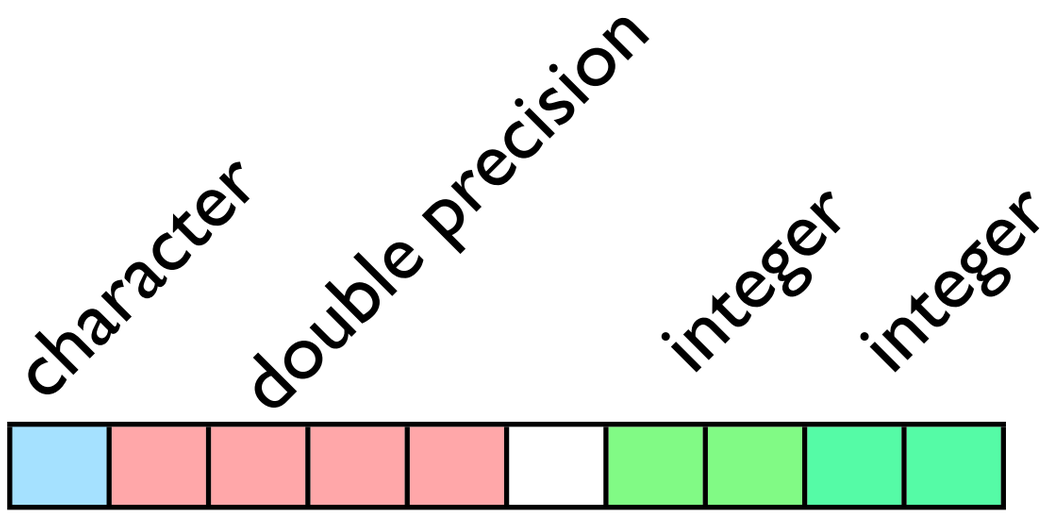
```
int MPI_Type_create_struct(int count, int array_of_blocklengths[],
    MPI_Aint array_of_displacements[], MPI_Datatype array_of_types[],
    MPI_datatype *newtype);
```

SciNet compute • calcul CANADA

MPI Structures

- Like vector, but:
- displacements in bytes
- array of types





MPI Type Maps

- Complete description of this structure looks like:

blocklens = (1,1,2)
displacements = (0,1,6)
types = (MPI_CHARACTER, MPI_DOUBLE_PRECISION, MPI_INTEGER)

- Note typemaps not unique; could write the integers out as two single integers with displacements 6, 8.

MPI Type Maps

- What does type map look like for domain_s?

```
typedef struct domain_s {
    float xleft, xright;
    float yleft, yright;
    int nxpts, nypts;
    float dx, dy;
    float **temp1, **temp2;
    float **old, **new;
} domain_t;
```

SciNet
compute • calcul
CANADA

MPI Type Maps

- Note: *can't* count on guessed locations.
- C compiler is allowed to insert padding between fields for performance or any other reason.
- Use offsetof(domain_s, dx) (eg) to find bytes of offset fro start of structure.

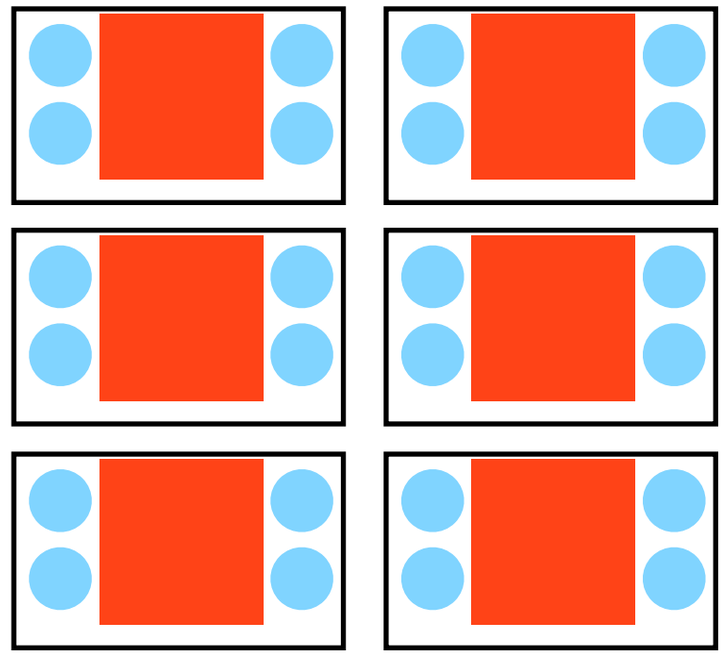
```
typedef struct domain_s {
    float xleft, xright;
    float yleft, yright;
    int nxpts, nypts;
    float dx, dy;
    float **temp1, **temp2;
    float **old, **new;
} domain_t;
```

Hybrid Parallelism: MPI + OpenMP

Two great tastes that go great together

SciNet compute • calcul CANADA

Hybrid programming

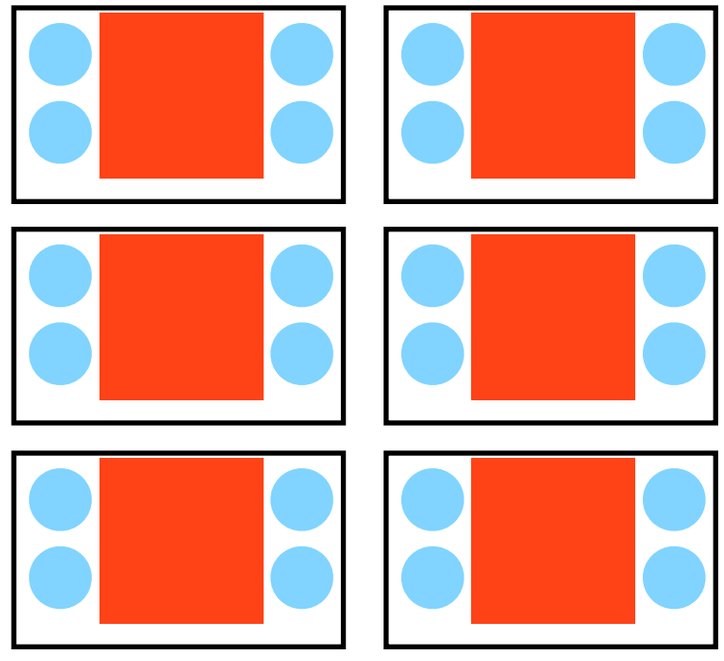


- Most current systems are hybrid:
- Distributed memory clusters of shared-memory nodes.
- Using MPI across nodes and OpenMP within nodes can better match software to hardware.

SciNet compute • calcul CANADA

Advantages: Course + Finer-grained parallelism

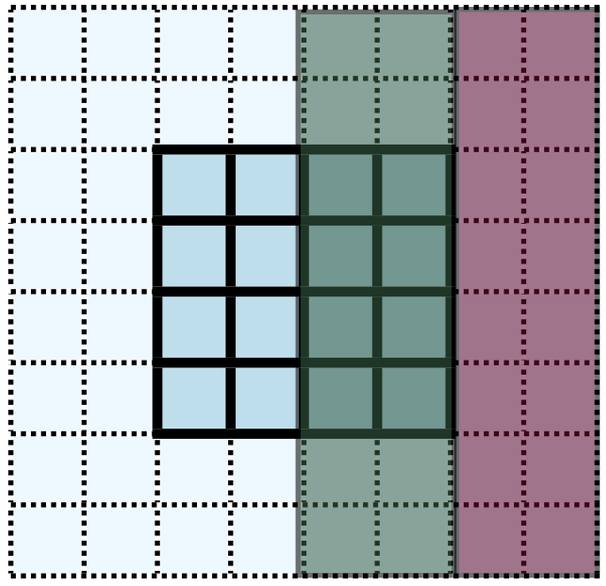
- Sometimes initial MPI parallelism is very coarse (eg, slabs of a domain)
- Requires lots of memory; can only fit 1,2 tasks per node
- With OpenMP, can implement finer grained parallelism within that, use all cores.



SciNet compute • calcul CANADA

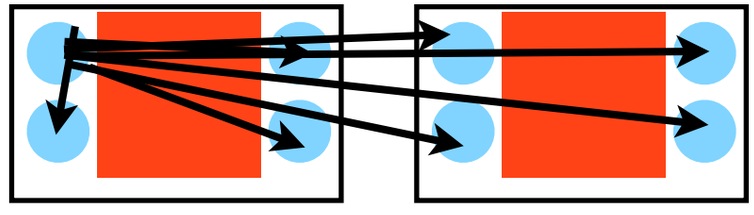
Advantages: Less memory duplication

- Each MPI task needs certain copies of data from remote tasks.
- For (eg) guardcells in 3D, can be sizable fraction of used memory.
- If fewer MPI tasks per node, reduce number of copies - better memory usage.

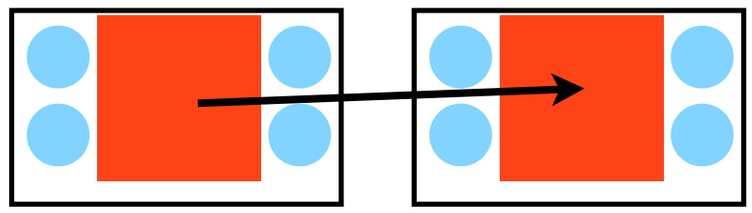


SciNet compute • calcul CANADA

Advantages: Better scaling



vs

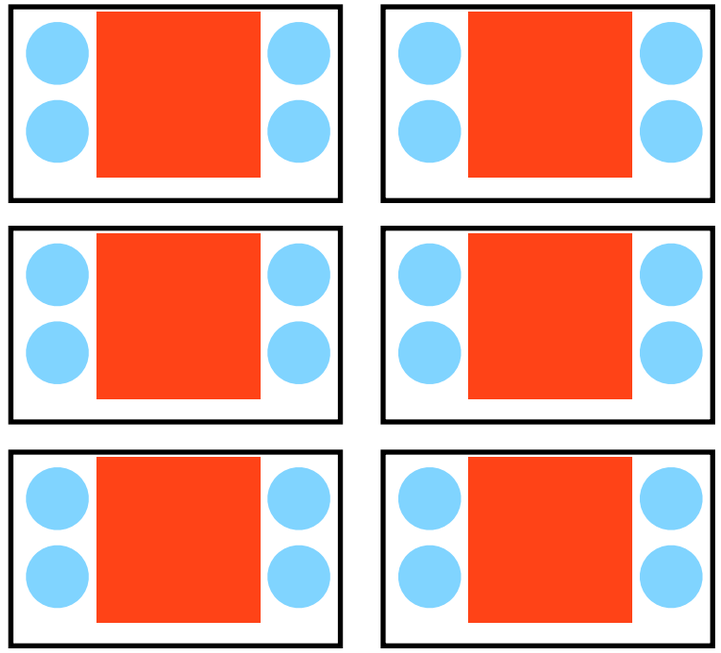


- With fewer MPI tasks,
  - Collectives scale better
  - Messages between nodes are aggregated: fewer, larger
- Can get better scaling at larger processor counts.

SciNet compute • calcul CANADA

Pitfalls

- Like anything with OpenMP, it's pretty easy to start with Hybrid
- But it takes more work to get the performance you want.



MPI_Init_thread

```
int MPI_Init_thread(
   int *argc,
   char ***argv,
   int required,
   int *provided )
```

- Needed to initialize thread-safe version of the MPI library
- There is a (small) performance overhead for this
- Can require different levels of thread safety.

Levels of thread safety

- MPI_THREAD_SINGLE: single-threaded.
- MPI_THREAD_FUNNELED: only the master thread will make MPI calls.
- MPI_THREAD_SERIALIZE: any thread may make MPI calls, but only one at a time.
- MPI_THREAD_MULTIPLE: anything goes.

```
int MPI_Init_thread(
  int *argc,
  char ***argv,
  int required,
  int *provided )
```

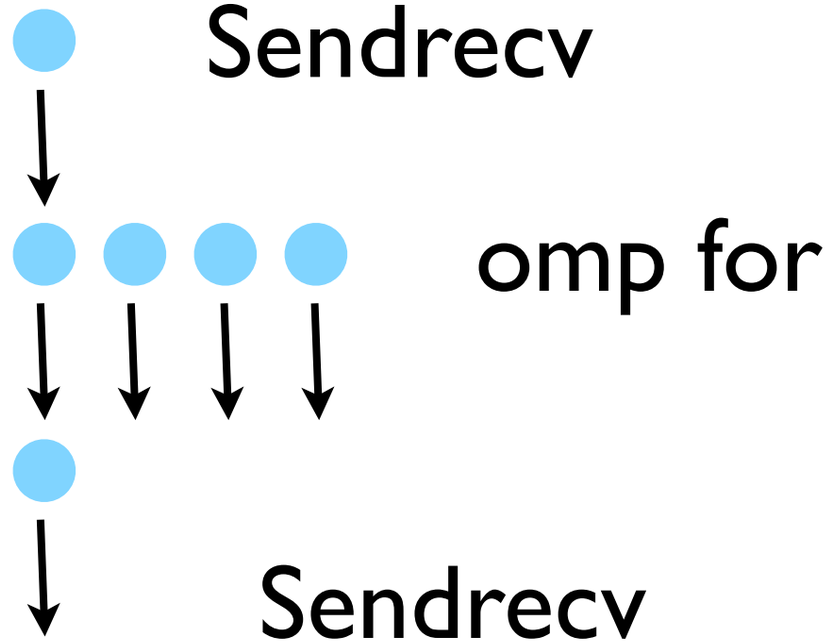
Simple example

- Call MPI Init thread; only master thread uses MPI
- Call MPI, then pragma omp parallel for over work loop
- Run with mpirun, OMP_NUM_THREADS

funneled.c

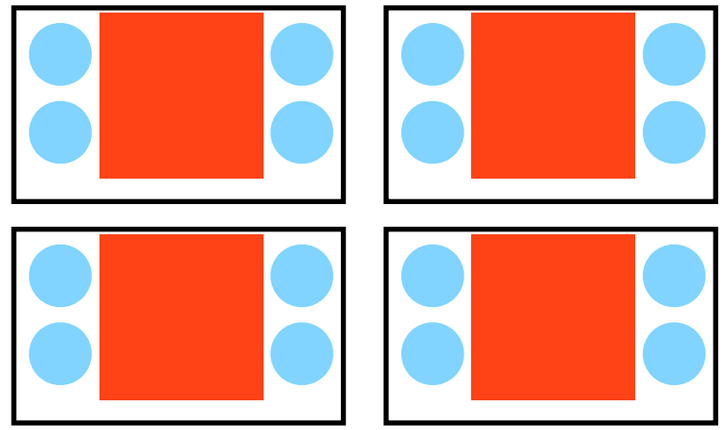
```
MPI_Init_thread(&argc, &argv,
                MPI_THREAD_FUNNELED, &provided);

if (provided < MPI_THREAD_FUNNELED)
    MPI_Abort(MPI_COMM_WORLD, 1);

...

for (int iter=0; iter < niters; iter++) {
    MPI_Sendrecv(&msgsent, 1, MPI_CHAR,
                 right, tag, &msgrcvd, 1,
                 MPI_CHAR, left,  tag,
                  MPI_COMM_WORLD, &status);

    /* do some work */

    #pragma omp parallel for
    for (int i=0; i<8; i++) {
        int tid=omp_get_thread_num();
        printf("(%d:%d) doing work item %d\n",
                rank, tid, i);
    }

}
```

Simple example



- One problem - (P-1) threads are sleeping much of the time
- Another problem: layout.

Dude, where's my thread?

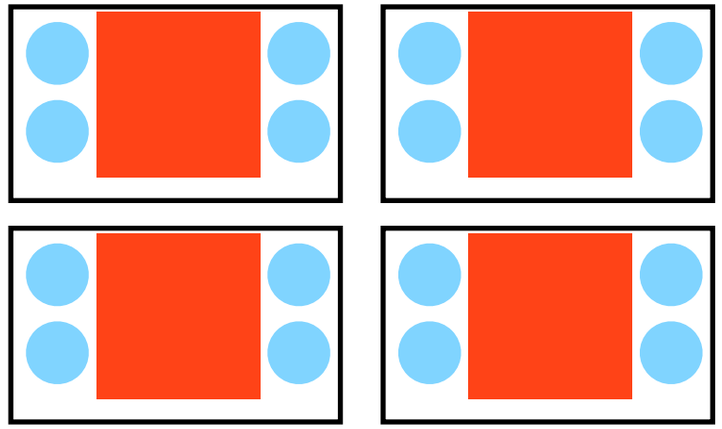


- When you have 4, 4-way nodes and want to run 16 mpi tasks, things are pretty simple.
- When you want to run 4, or 8, tasks, and have each run 4 or 2 threads, it matters a lot where the tasks are.

SciNet compute • calcul CANADA

Dude, where's my thread?

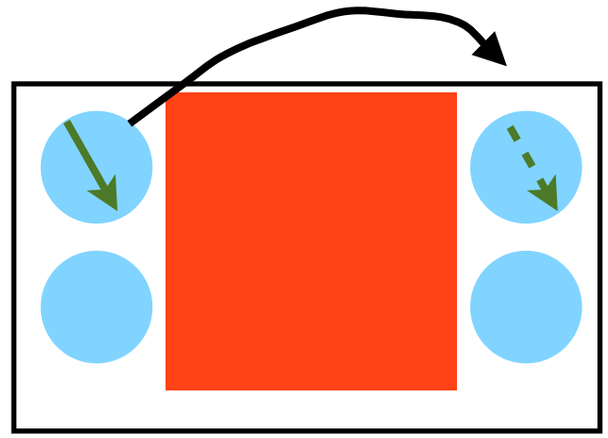
- OpenMPI: -bynode assigns one MPI task per node, then "wraps around" if needed.
- -bycore, -bysocket.
- If you use a nonstardard OMP_NUM_THREADS, you may have to -x OMP_NUM_THREADS to ensure each task sees env variable.



SciNet compute • calcul CANADA

CPU affinity

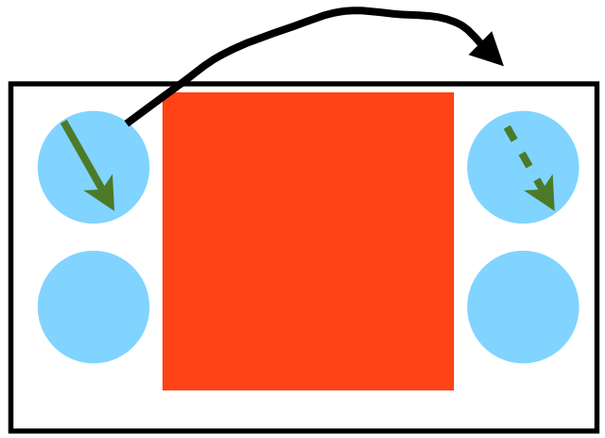
- OS has the flexibility to move processes, threads between cores.
- Generally right thing for, eg, web server, generally not for HPC.
- Want to bind to cores or socket (but _not_ bind everything to same core/ socket!)



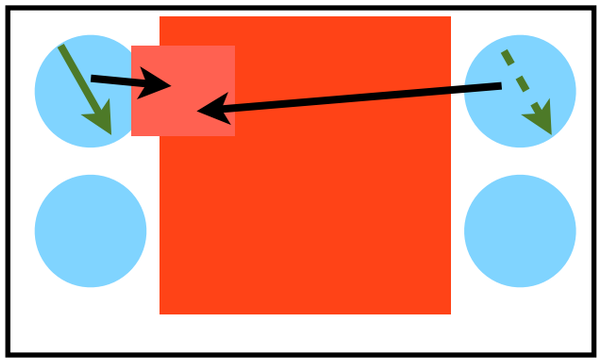
SciNet compute • calcul CANADA

CPU affinity

- OpenMPI: -bind-to-core, -bind-to-socket
- Binds processes (and then all threads) to that compute element. Be careful!
- -display-map, -report-bindings, to see what's going on.



Memory affinity

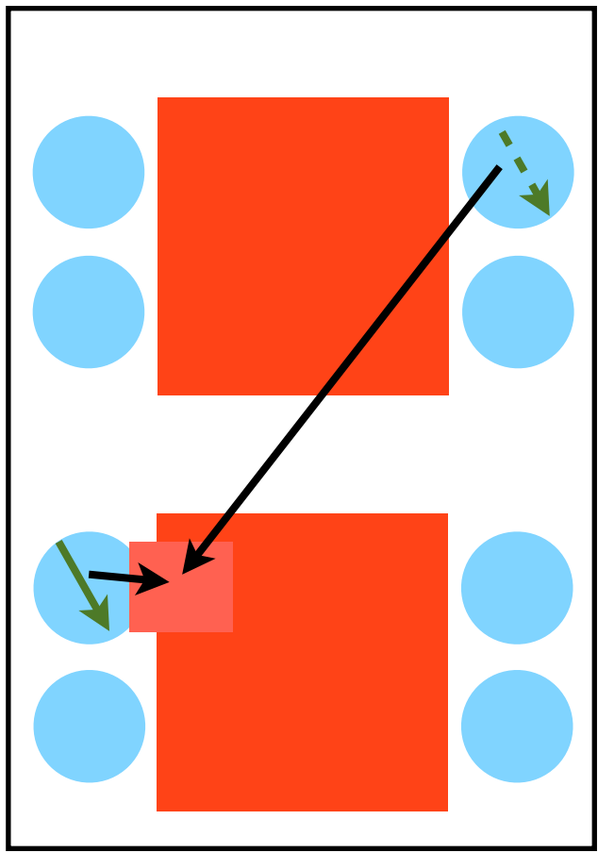


- Memory may be globally accessible, but it isn't *uniform*.
- NUMA - extra ~100ns to access memory in other core's cache on-socket

SciNet compute • calcul CANADA

Memory affinity

- Memory may be globally accessible, but it isn't *uniform*.
- NUMA - extra ~100ns to access memory in other core's cache on-socket
- Even worse if it's off socket.
- An excellent reason to worry about cpu affinity

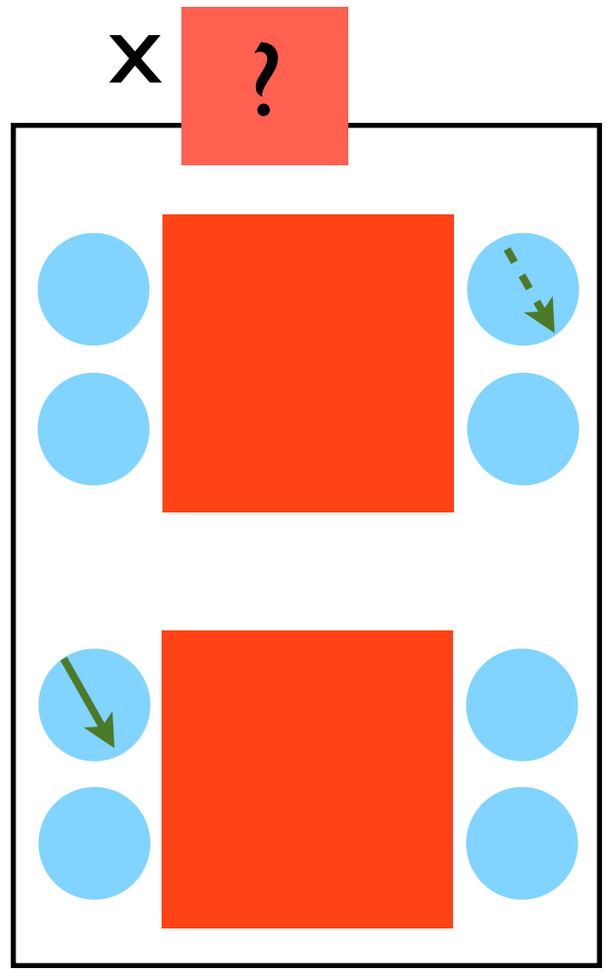


SciNet compute • calcul CANADA

First Touch

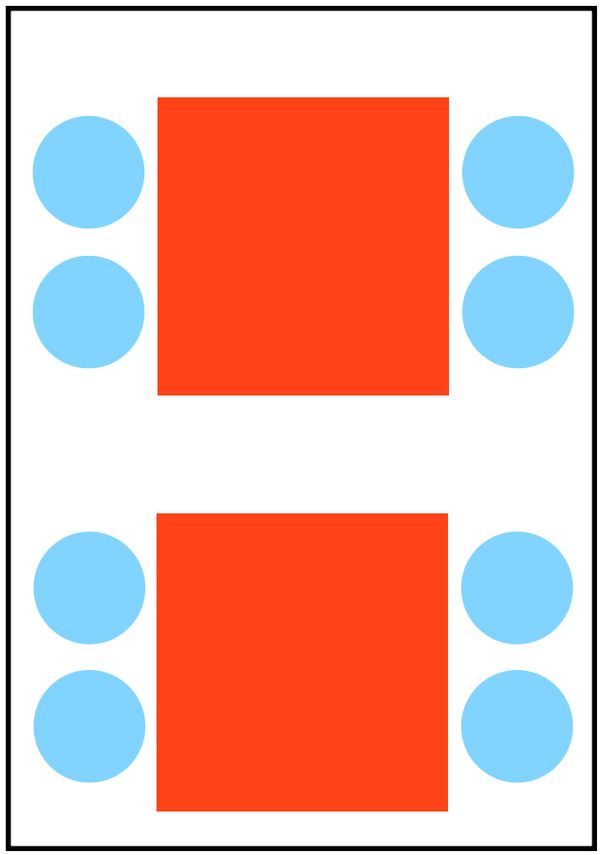
double x[100][100]

- Where does a given array "live"?
- Large arrays broken into *pages*.
- Typically pages associated not with CPU that allocated array, but with CPU that touched it first.
- Makes sense to do even initialization with OpenMP - locks pages to correct CPUs.



How many tasks per node?

- No *a priori* answer - need to experiment.
- Sensible starting points:
- 1 task per socket (ensures good shared mem locality)
- 1 task per network connection (ensures no contention in/out of node).



SciNet compute • calcul CANADA

Can we do better than simple case?

- Ideally, want all threads going at once
- Minimize serialization at communication points.
- Overlap communication with computation via threads.

funneled.c

```
MPI_Init_thread(&argc, &argv,
                MPI_THREAD_FUNNELED, &provided);

if (provided < MPI_THREAD_FUNNELED)
    MPI_Abort(MPI_COMM_WORLD, 1);

...

for (int iter=0; iter < niters; iter++) {
    MPI_Sendrecv(&msgsent, 1, MPI_CHAR,
                 right, tag, &msgrcvd, 1,
                 MPI_CHAR, left,  tag,
                  MPI_COMM_WORLD, &status);

    /* do some work */

    #pragma omp parallel for
    for (int i=0; i<8; i++) {
        int tid=omp_get_thread_num();
        printf("(%d:%d) doing work item %d\n",
                rank, tid, i);
    }

}
```

SciNet compute • calcul CANADA

Can we do better than simple case?

- Have master thread sendrecv, then have rest do dynamic loop.
- NOTE: _all_ omp threads in team _must_ participate in omp loop.

dynamic.c

```
MPI_Init_thread(&argc, &argv,
                MPI_THREAD_FUNNELED, &provided);

if (provided < MPI_THREAD_FUNNELED)
    MPI_Abort(MPI_COMM_WORLD, 1);

...

#pragma omp parallel
{
        #pragma omp master nowait
        {
            sendrecv(rank, sneighbour, rneighbour
            printf("Got data from %d\n", data);
        }

        #pragma omp for schedule(dynamic)
        for (int i=0; i<n; i++) {
            int tid = omp_get_thread_num();
            printf("%d:%d working on item %d\n",
        }
}
```

SciNet compute • calcul CANADA

Can we do better than simple case?

- Have master thread sendrecv, then have rest do dynamic loop.
- NOTE: _all_ omp threads in team _must_ participate in omp loop.

tasks.c

```
omp_set_nested(1);
#pragma omp parallel
{
    #pragma omp single
    {
        #pragma omp task
        {
            sendrecv(rank, sneighbour
            printf("Got data from %d\
        }

        #pragma omp task
        work(rank);
    }
}

                    ...

void work(int rank) {
    const int n=14;
    #pragma omp parallel f
    for (int i=0; i<n; i++
        int tid = omp_get_
        printf("%d:%d work
    }
}
```

Homework: Hybrid diffusion2d

- Three versions of diffusion2d:
- Pure MPI, blocking guardcells
- Pure MPI, nonblocking
- Hybrid: Timings
- Set points to something much larger (10k? 50k?) and reduce number of iterations to few dozen
- Try to get best performance you can on 4 nodes (=32 processors). Is one MPI task per node best, or 2, or?
- Due Mar 22.

SciNet compute • calcul CANADA