MPI in ROMS

Kate Hedstrom

Dan Schaffer, NOAA

Tom Henderson, NOAA

January 2011

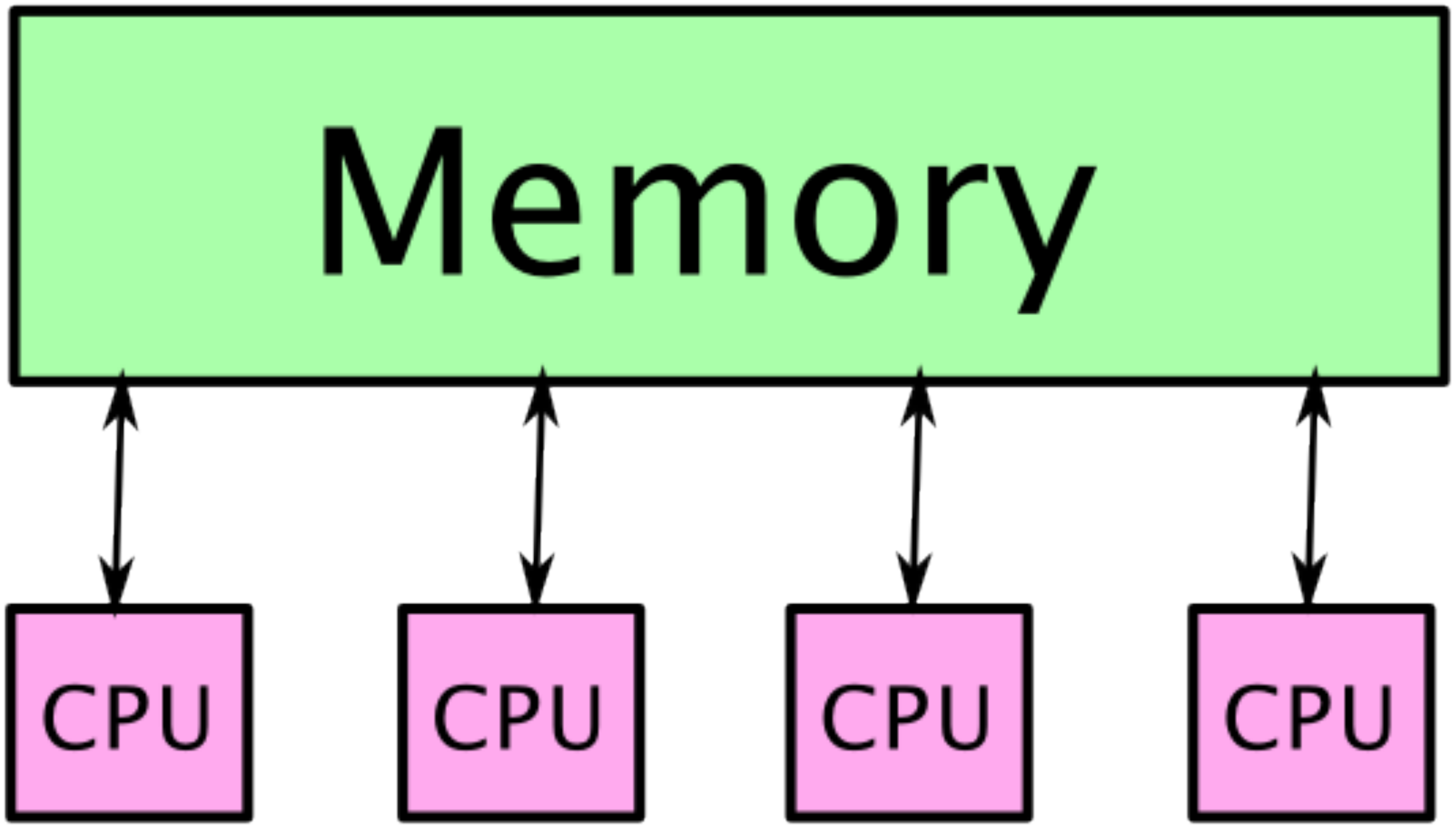
Outline

- Introduction to parallel computing
- ROMS grids
- Domain decomposition
- Picky details
- Debugging story

Parallel Processing

- Computation/communication – if you spend more time communicating, then you should be running on fewer processors
- It’s all about the memory – shared memory vs. distributed memory. What does your process know?
- Threads are for shared memory

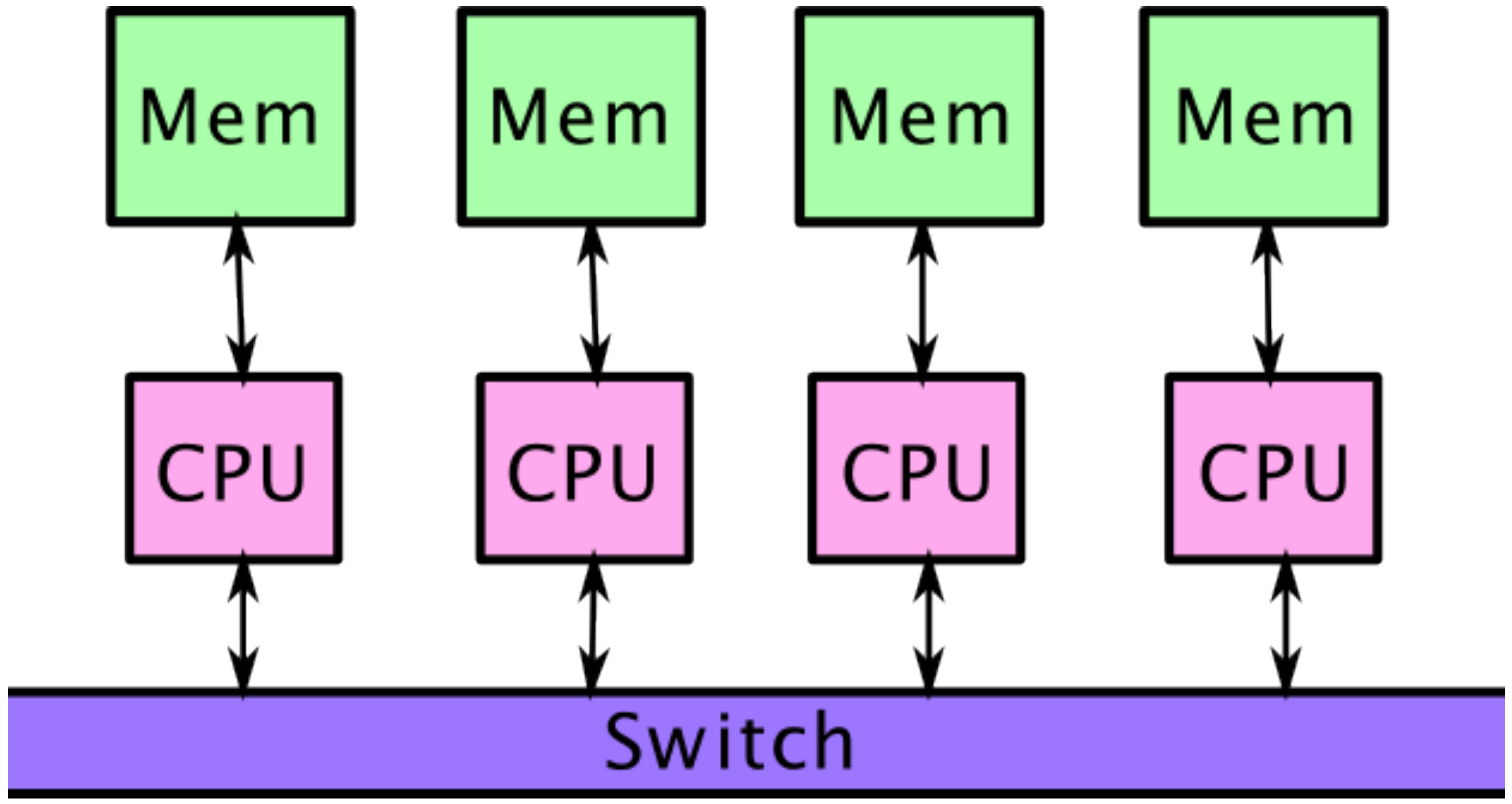
Shared Memory



Shared Memory

- **All can see all memory, but some is closer than others**
- **System does communication if needed – handled automatically**
- **“First touch” means first process to write to an address becomes owner**
  - Each process initializes own tile to become “owner”
  - Read from other tiles, but never write to them

Distributed Memory



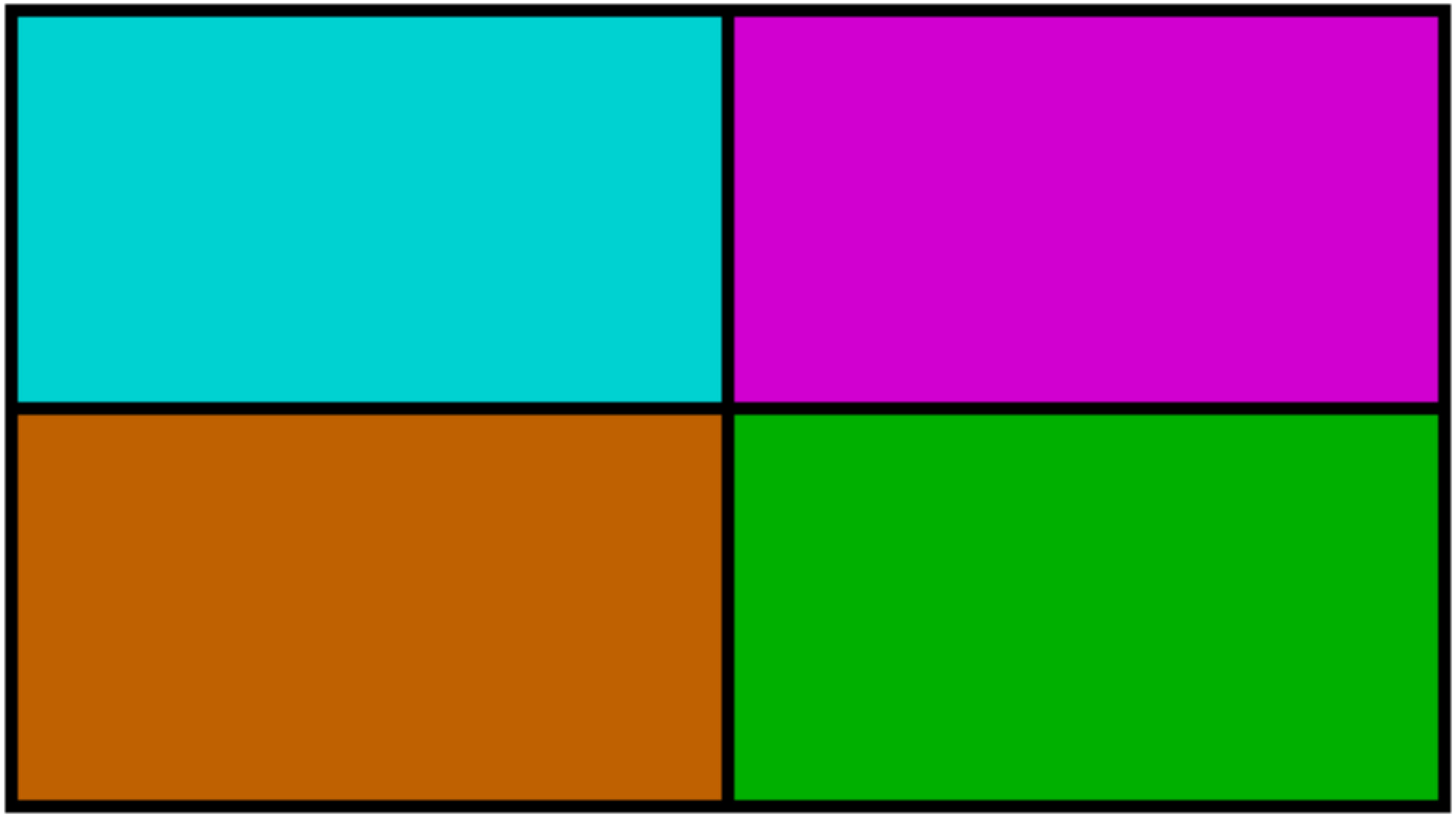
Distributed Memory

- Each processor has its own memory, perhaps own disk
- Need a method to communicate between processes
- Need to structure your code to work on these systems
- Computers with multiple “nodes” are usually shared within a node, distributed across nodes

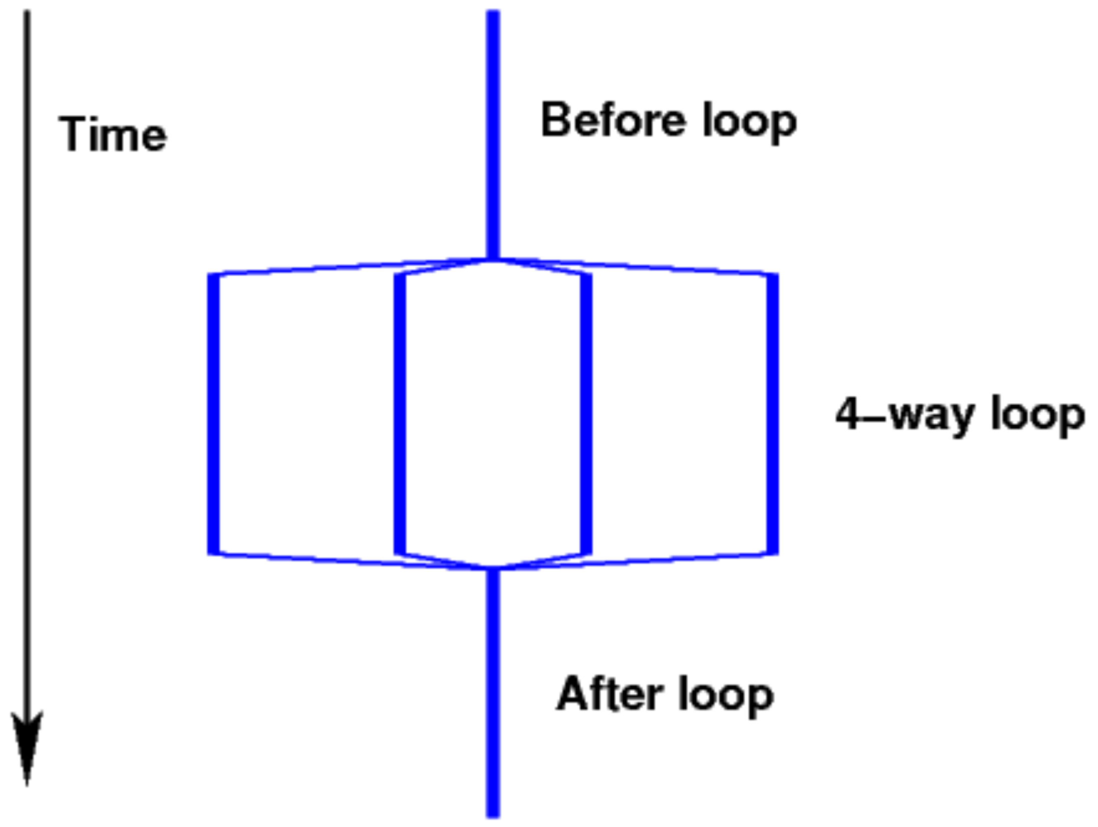
MPI

- Message Passing Interface (MPI) is industry standard for how to code for modern computers
- Works on many kinds of clusters, even those with shared memory
- Library, callable from C, Fortran, C++, etc.
- Need to know how to access on your system

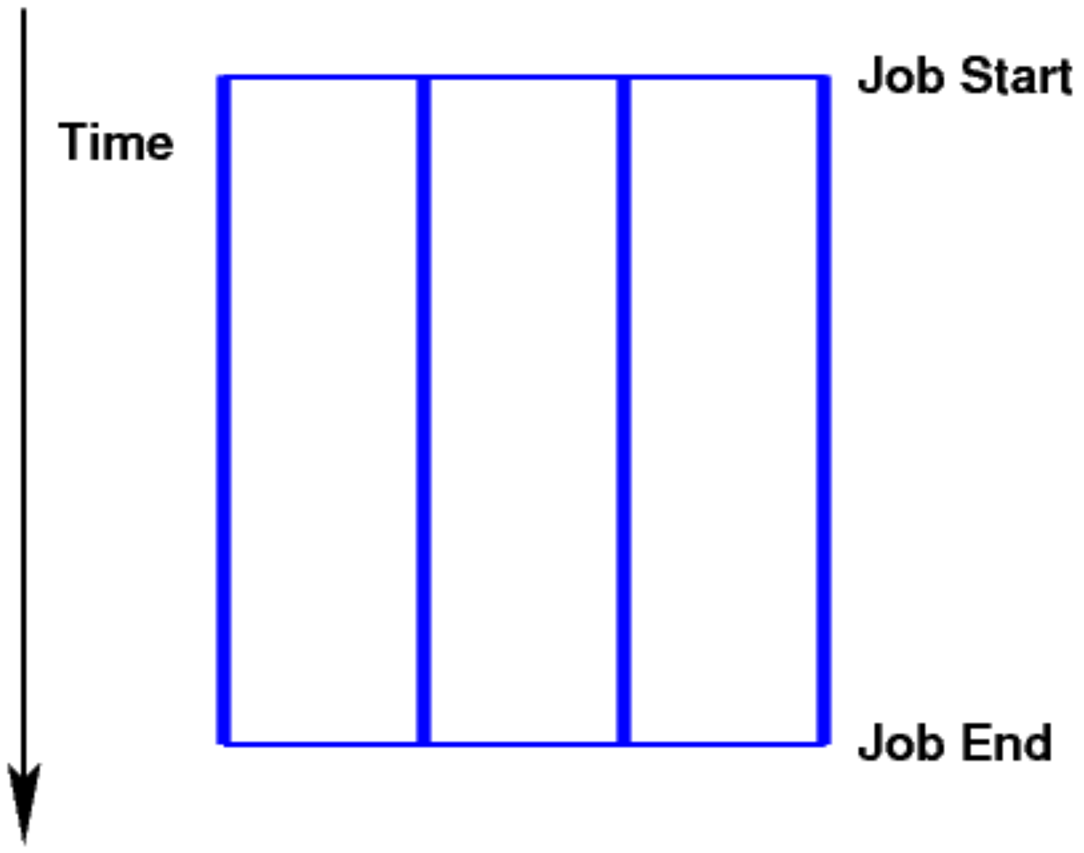
Domain Decomposition



Multi-threaded Job



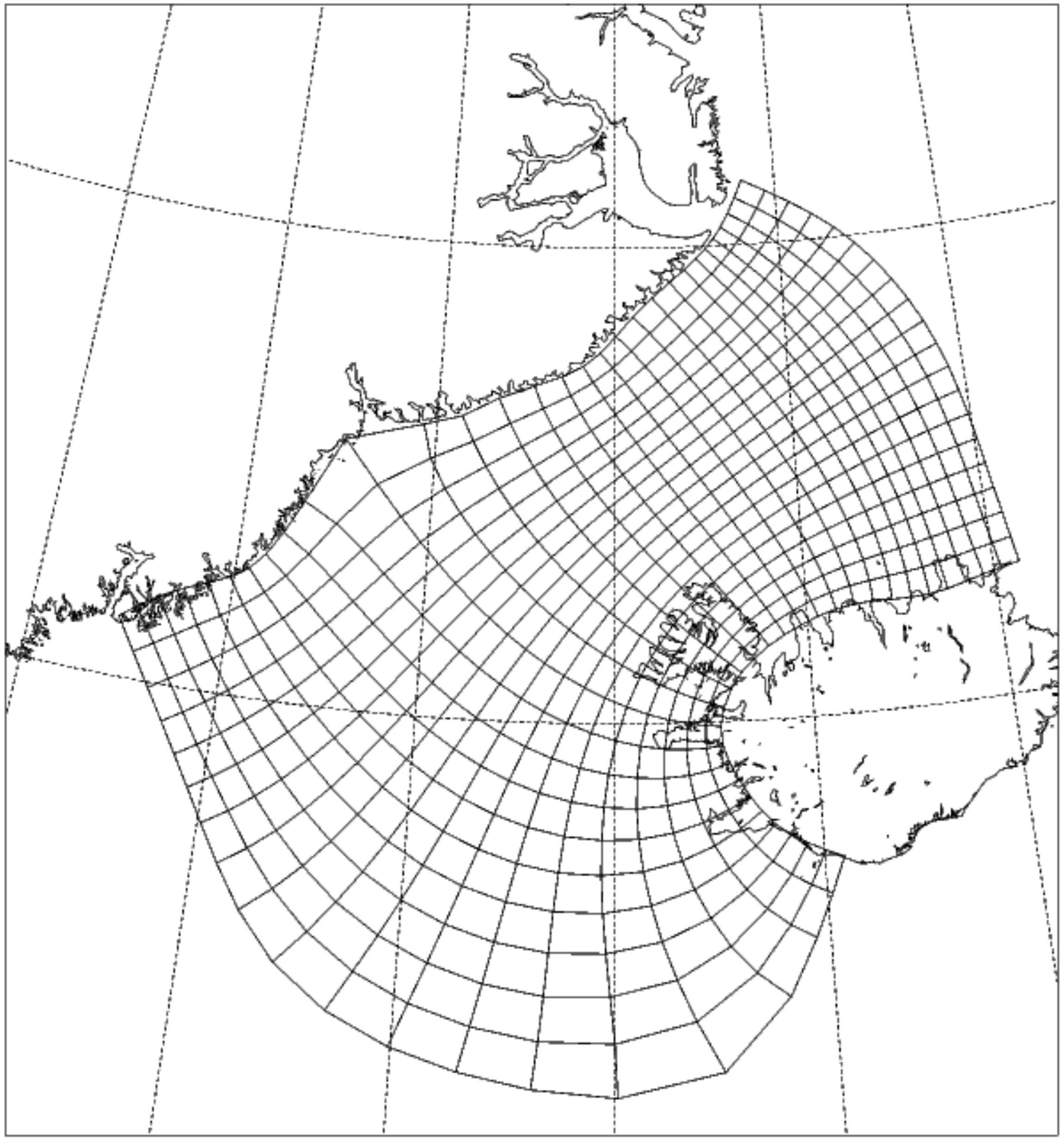
MPI Job



ROMS

- Regional Ocean Modeling System
- Ocean model designed for limited areas, I also have ice in it
- Grid is structured, orthogonal, possibly curvilinear
- Islands and peninsulas can be masked out, but are computed
- Horizontal operations are explicit
- Vertical operations have an implicit tridiagonal solve

Sample Grid

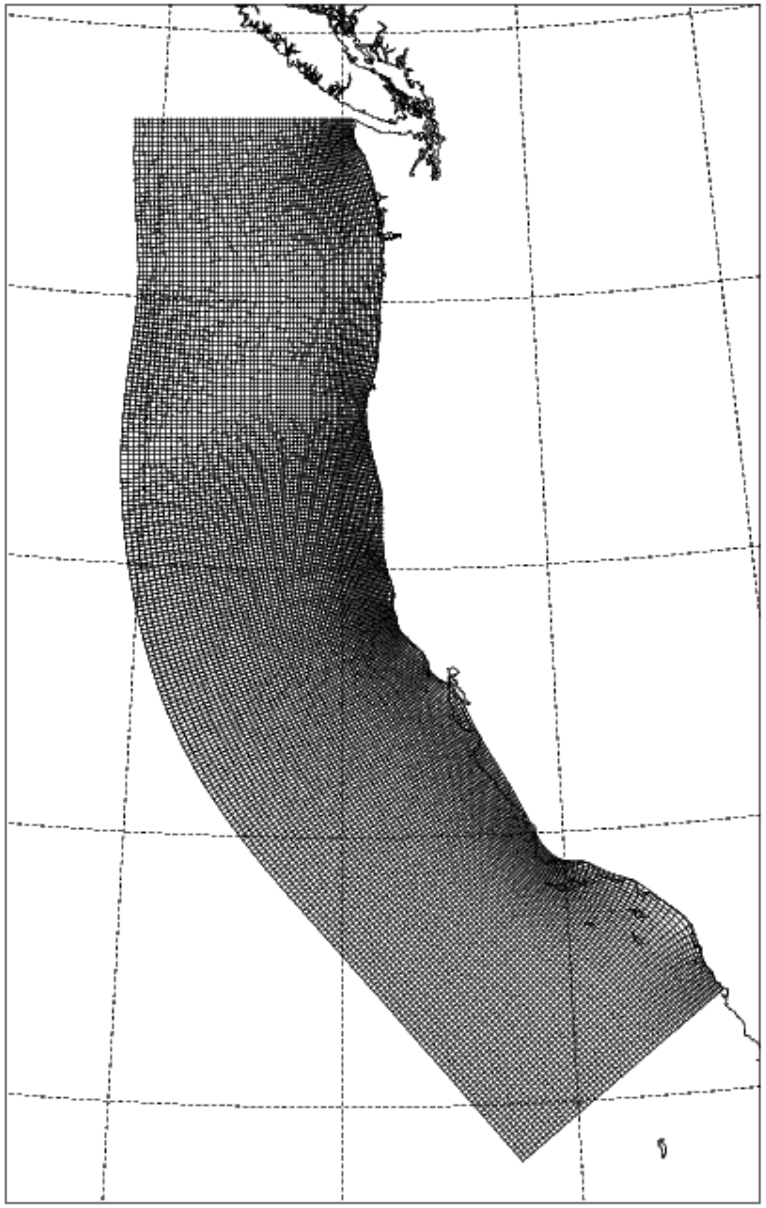


Some History

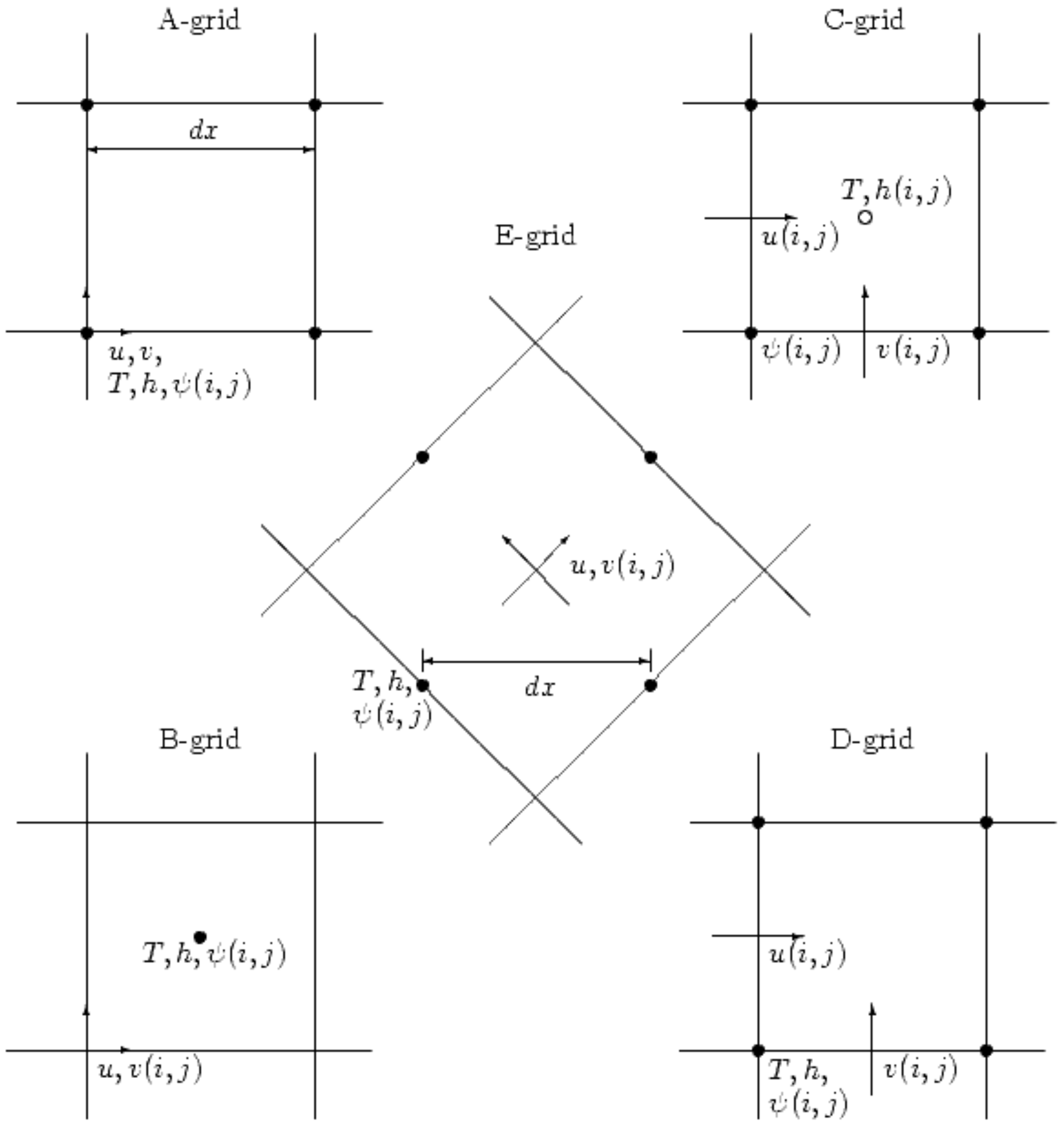
- Started as serial, vector f77 code
- Sasha Shchepetkin was given the job of making it parallel - he chose SGI precursor to OpenMP (late 1990's)
- Set up tile structure, minimize number of thread creation/destruction events
- NOAA people converted it to SMS parallel library (2001)
- Finally went to a native MPI parallel version (2002) - and f90!
- Sasha independently added MPI

Computational Grids

- **Logically rectangular**
- **Best parallelism is domain decomposition**
- **Well understood, should be easy to parallelize**

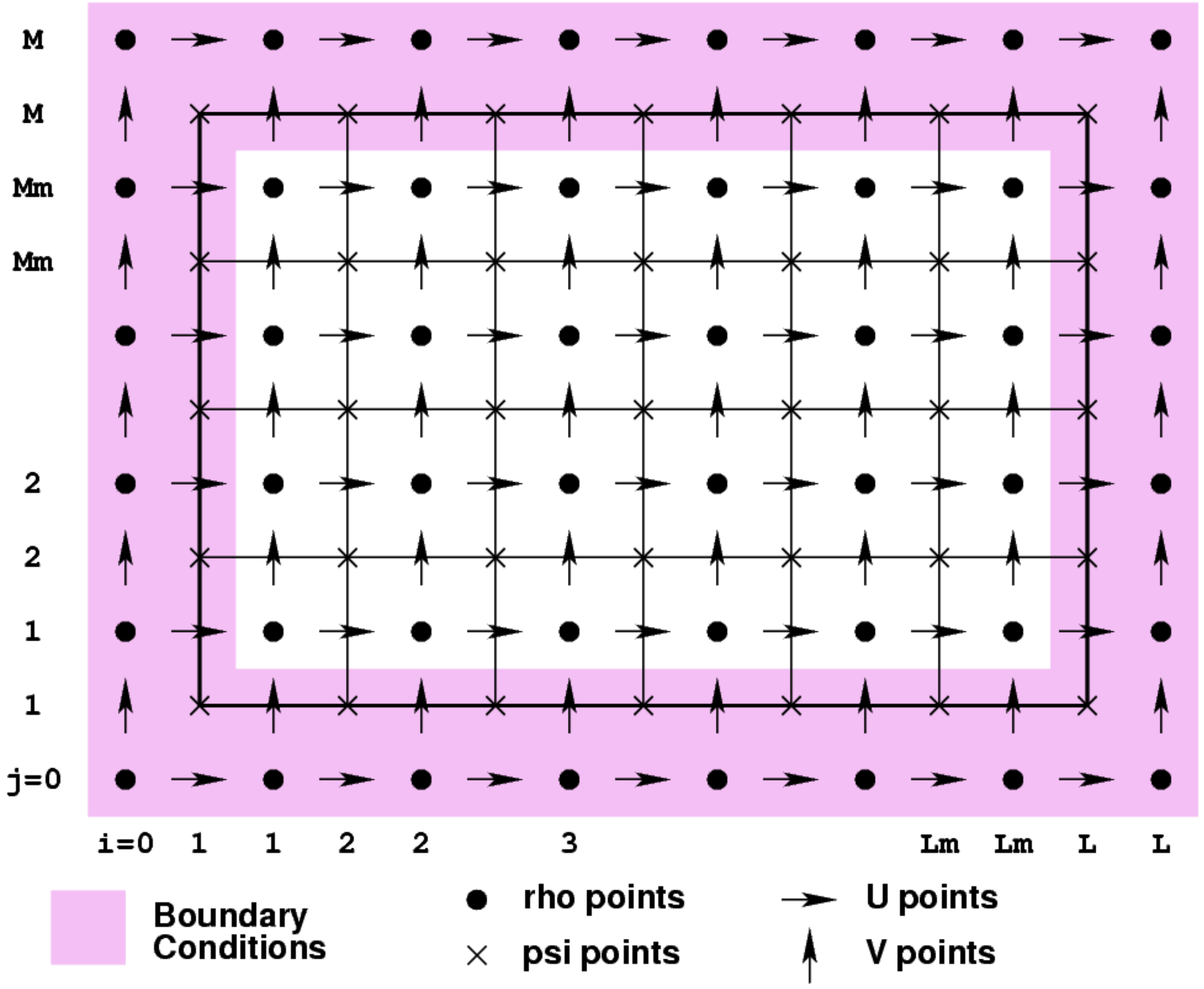


Arakawa Numerical Grids



The Whole Grid

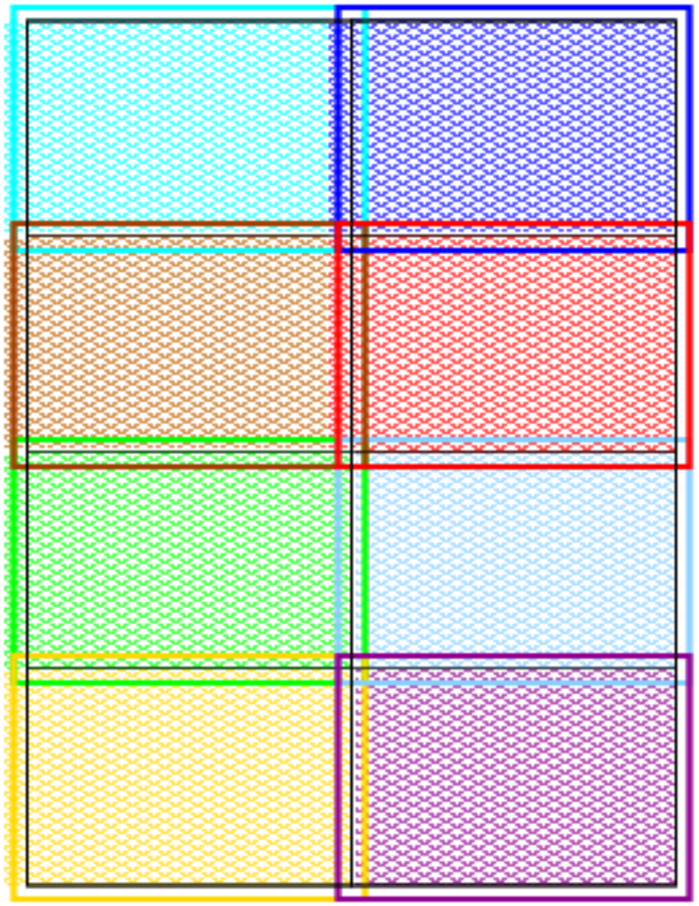
- Arakawa C-grid, but all variables are dimen-sioned the same
- Computa-tional domain is Lm by Mm



Parallelization Goals

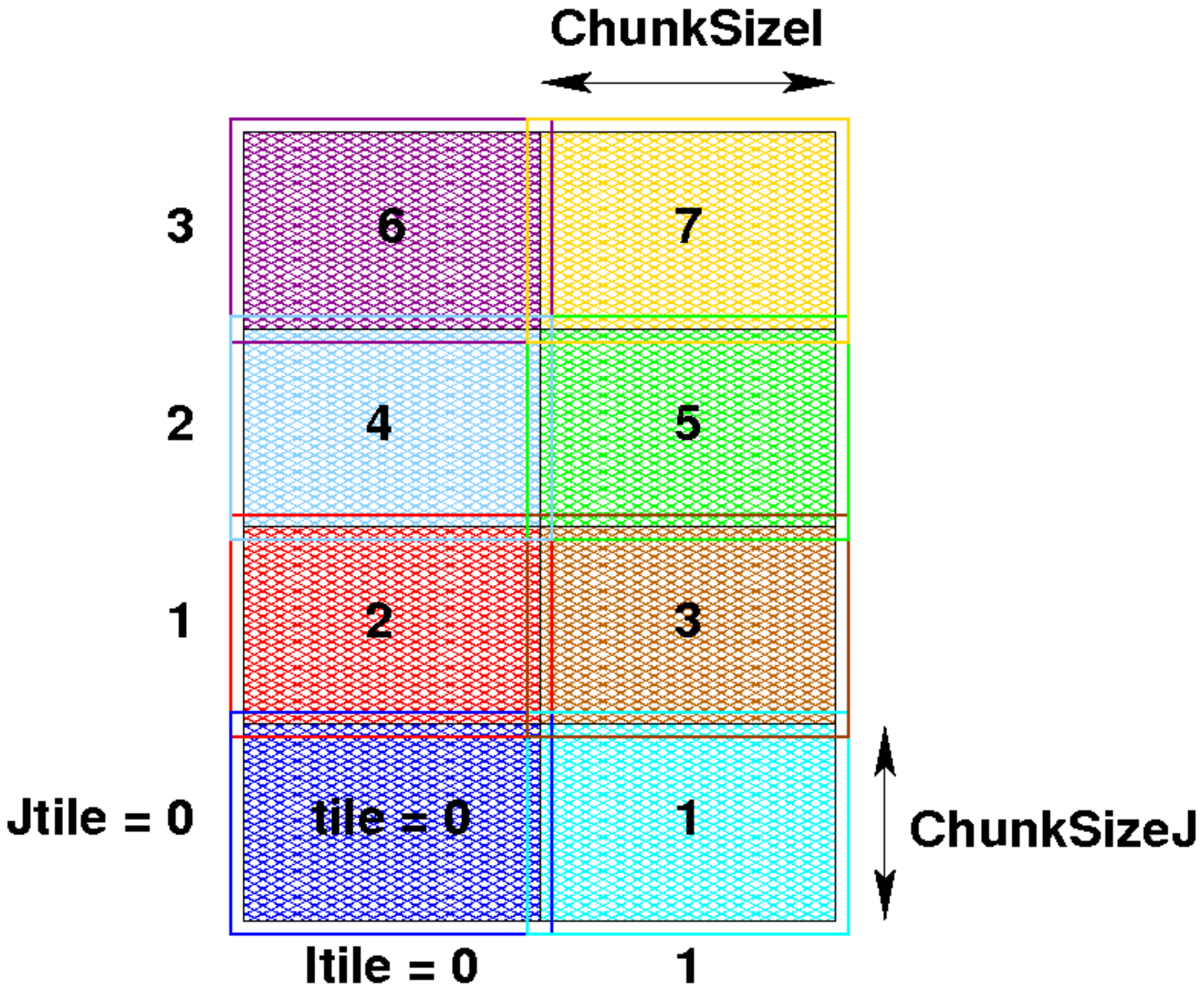
- **Ease of use**
  - Minimize code changes
  - Don't hard-code number of processes
  - Same structure as OpenMP code
- **High performance**
  - Don't break serial optimizations
- **Correctness**
  - Same result as serial code for any number of processes
- **Portability**
  - Able to run on anything (Unix)

Domain Decomposition



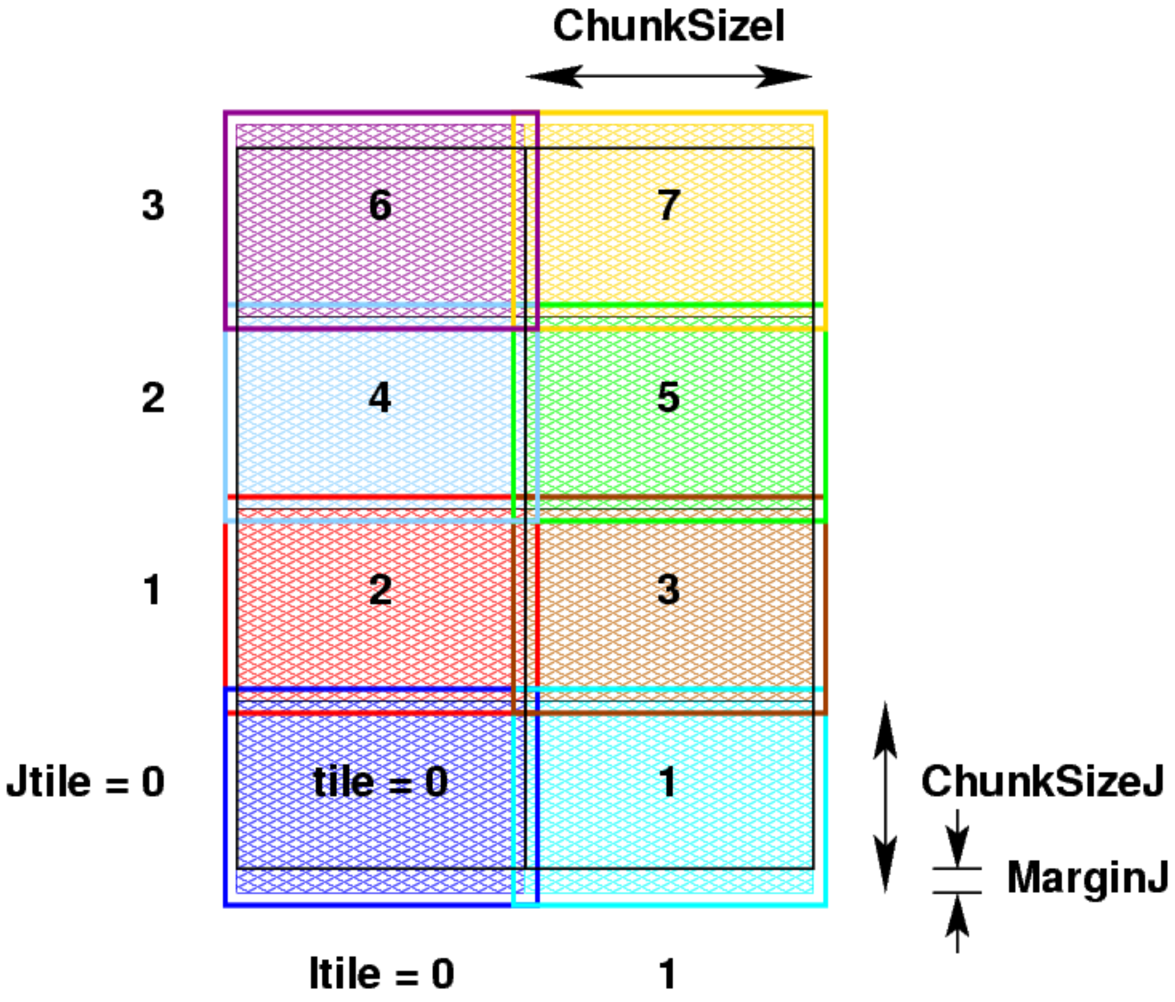
- **Overlap areas are known as ghost points**

Some Numbering Schemes

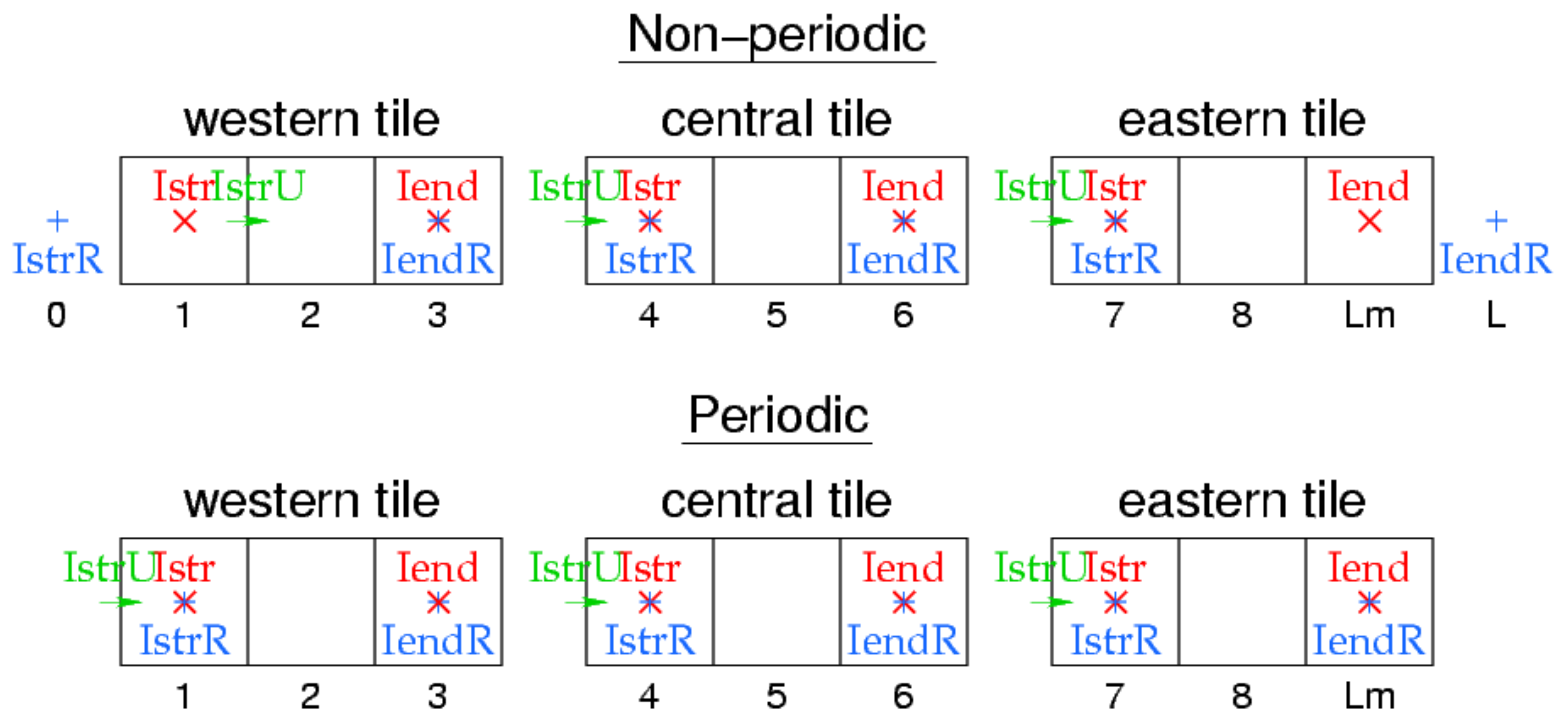


Mm Not Divisible by 4

- These numbers are in structure BOUNDS in mod_param.F
- ROMS should run with any Mm, may be unbalanced



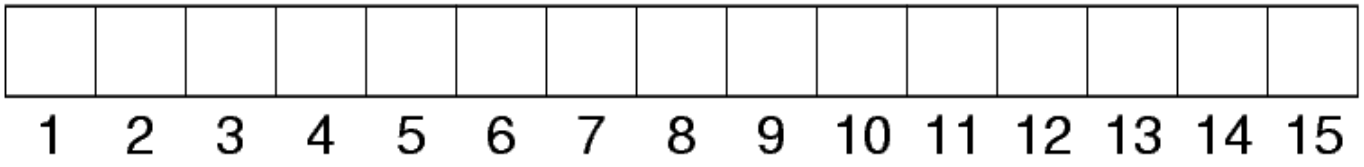
ROMS Tiling Details

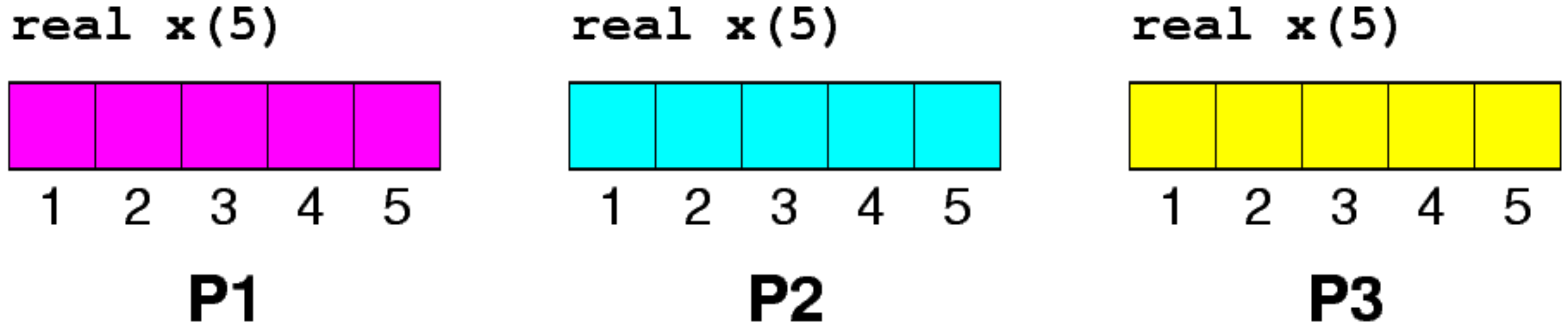


- **Do loop bounds given in terms of Istr, Iend, etc., from BOUNDS**

Simple 1D Decomposition: Static Memory

```
real x(15)
```

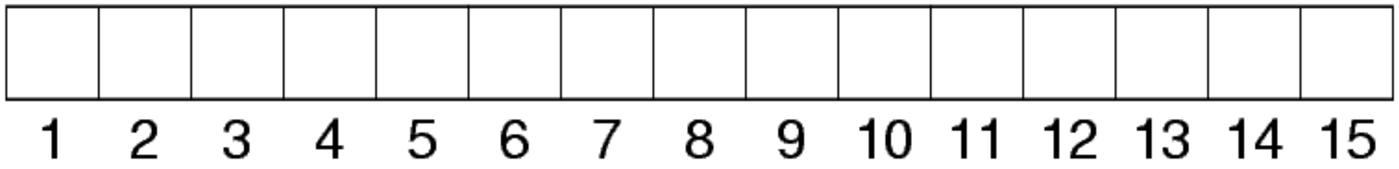


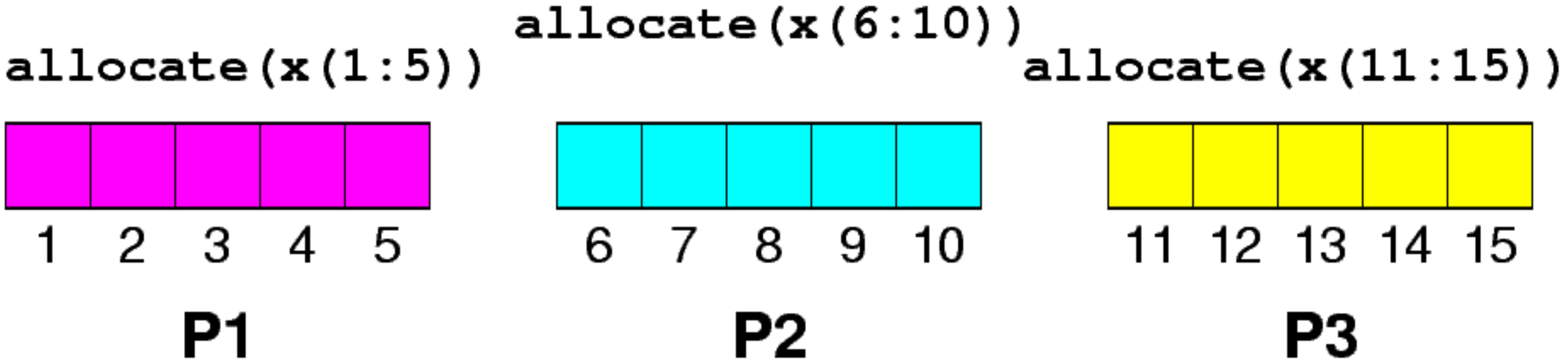


Simple 1D Decomposition: Dynamic Memory

```
real, allocatable :: x(:)

allocate(x(15))
```





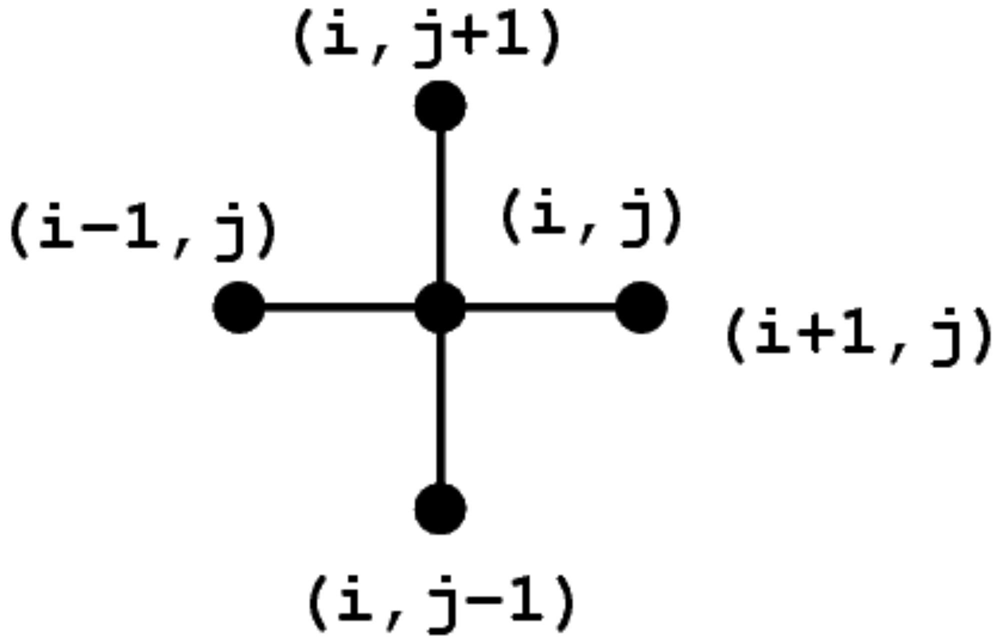
We Chose Dynamic

- More convenient for location of river sources, land mask, etc
- Simpler debugging, even if just with print statements
- If we manage it right, there shouldn't be extra overhead
- Sasha chose static, not trusting new f90 features to be *fast*

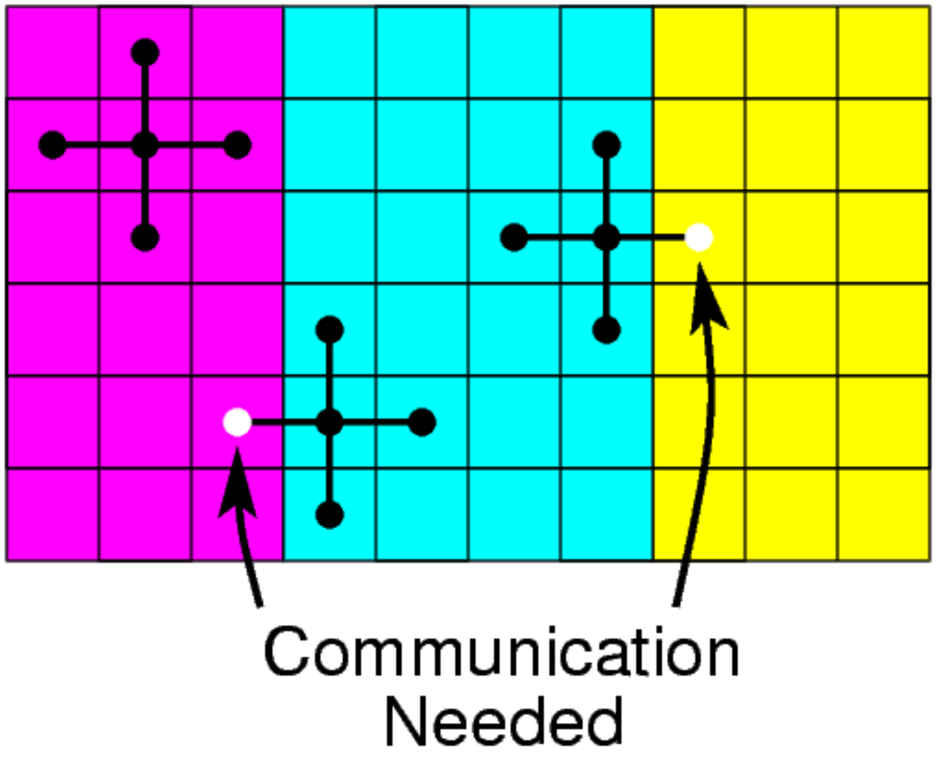
Adjacent Dependencies

```
y(i,j) = x(i,j) + x(i+1,j) + x(i-1,j)
                + x(i,j+1) + x(i,j-1)
```

"Stencil"

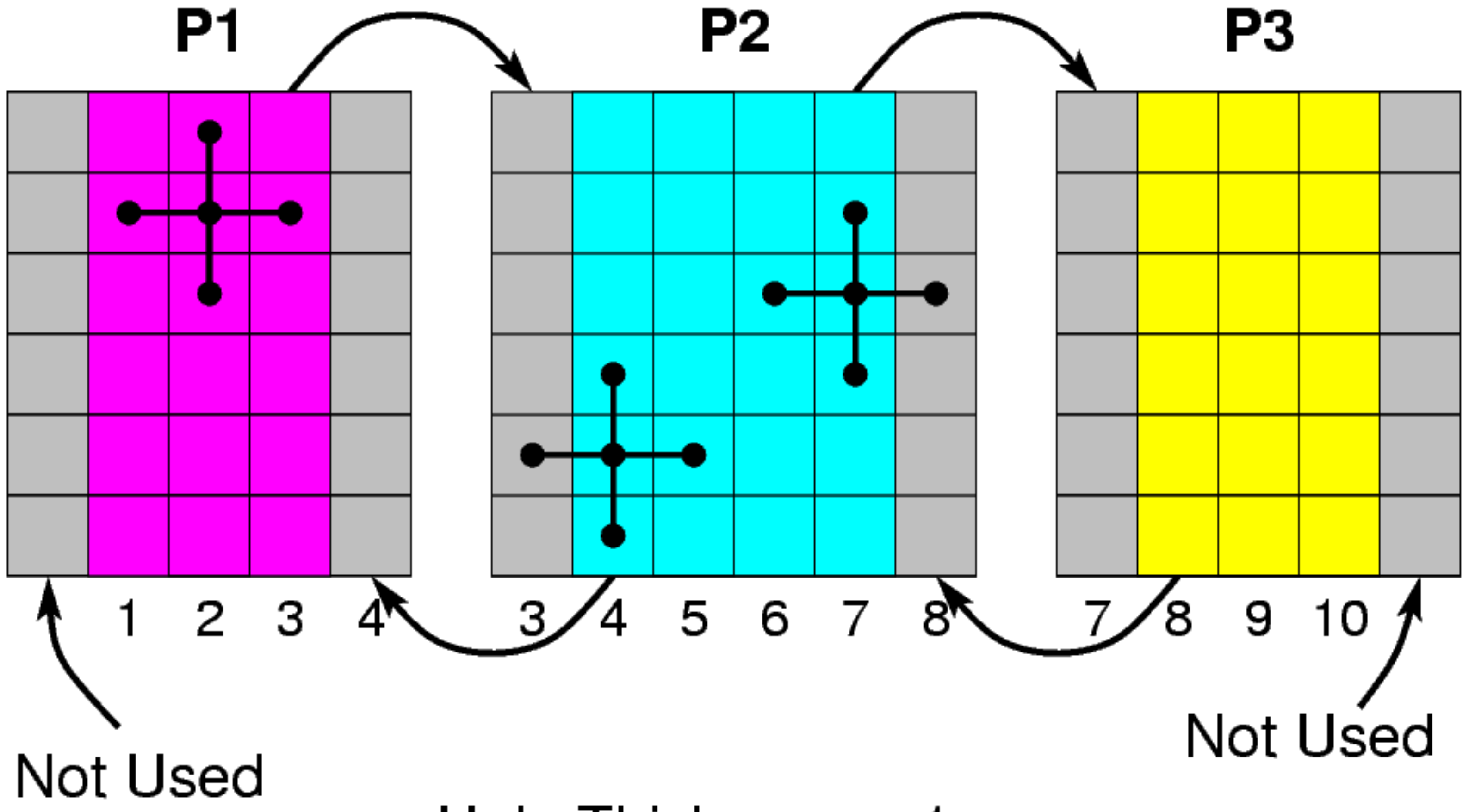


Add "Halo" Regions for Adjacent Dependencies



```
y(i,j) = x(i,j) + x(i+1,j) + x(i-1,j)
                + x(i,j+1) + x(i,j-1)
```

Halo Region Update: Non-Periodic Exchange



Halo Thickness = 1

Some Details

- **Number of ghost/halo points needed depends on numerical algorithm used**
  - 2 for most
  - 3 for MPDATA advection scheme, biharmonic viscosity

More Details

- **Number of tiles NtileI and NtileJ read from a file during initialization**
- **Product NtileI*NtileJ must match number of MPI processes**
- **Size of tiles is computed:**

```
ChunkSizeI=(Lm+NtileI-1)/NtileI
MarginI=(NtileI*ChunkSizeI-Lm)/2
```

- **Each tile has a number, matching the MPI process number**

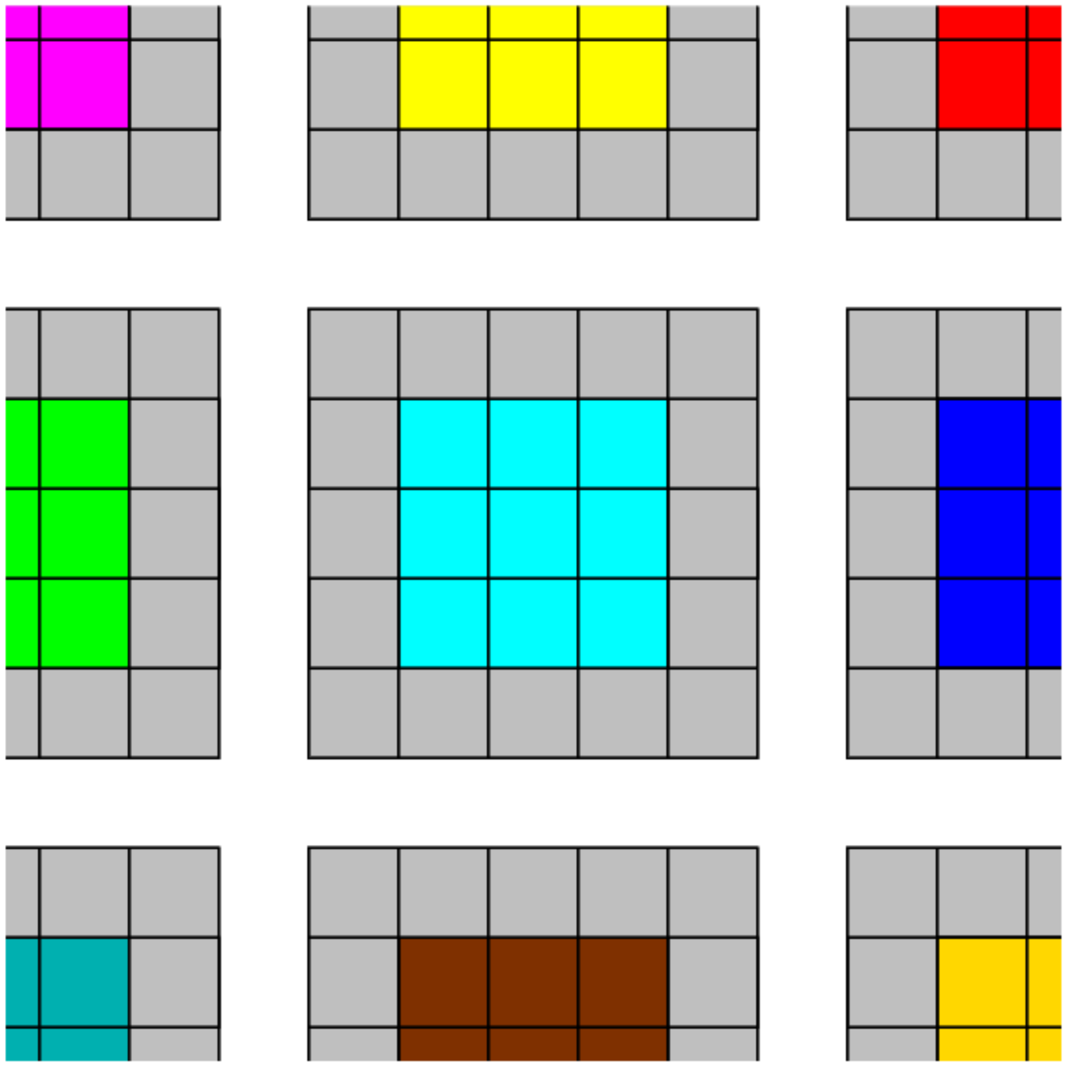
Still More

- **We use the C preprocessor extensively**
- **DISTRIBUTE is cpp tag for the MPI code**
- **There are #defines for EASTERN_EDGE, etc:**

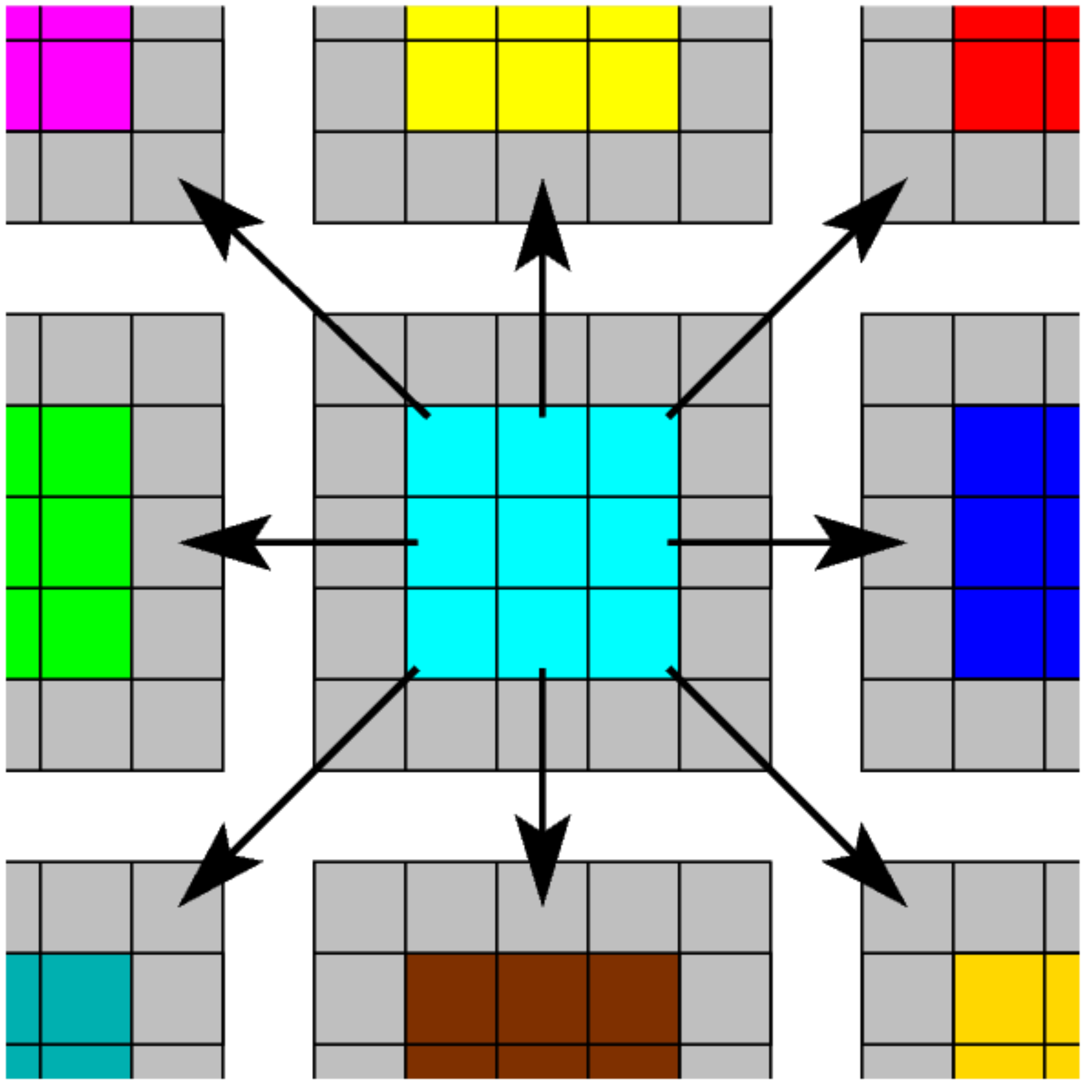
```
#define EASTERN_EDGE Iend.eq.Lm
   if (EASTERN_EDGE) then
           :
#define PRIVATE_1D_SCRATCH_ARRAY
  IminS:ImaxS
```

- **IminS is Istr-3, ImaxS is Iend+3**

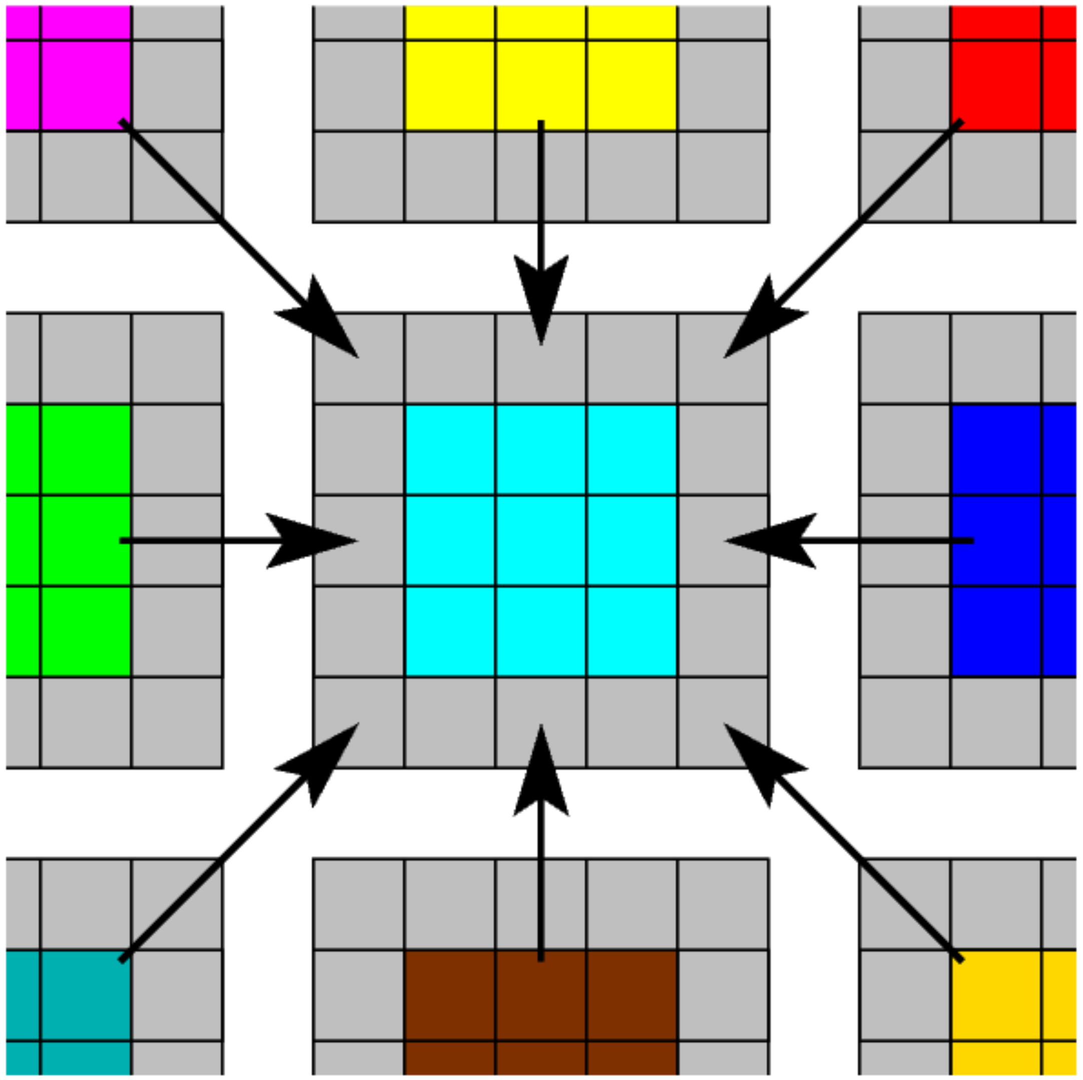
2D Exchange - Before



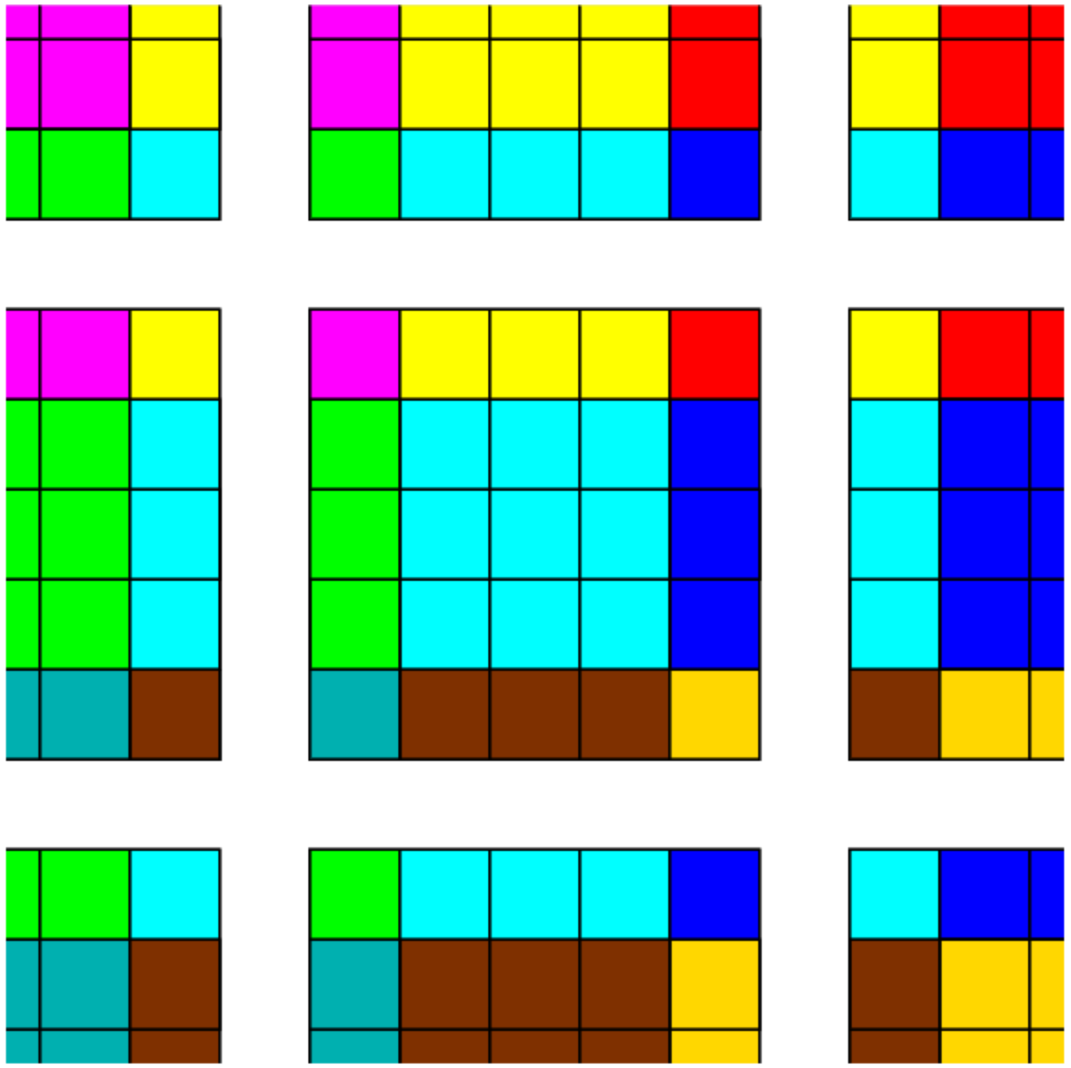
2D Exchange - Sends



2D Exchange - Receives



2D Exchange - After



Notes

- SMS does the 2-D exchanges all in one go
- ROMS does it as a two step process, first east-west, then north-south
- Sasha's code can do either
- Routines for 2-D, 3-D and 4-D fields, mp_exchange2d, etc., exchange up to four variables at a time

mp_exchange

```
call mp_exchange2d(ng, tile,           &
    iNLM, 2, Lbi, Ubi, LBj, Ubj,       &
    Nghost, EWperiodic, NSperiodic,&
    A, B)
```

- **It calls**
  - mpi_irecv
  - mpi_send
  - mpi_wait

Main Program

- **In MPI, numthreads=1, subs=1, tile=0**

```
!$OMP PARALLEL DO PRIVATE…
   DO thread=0,numthreads-1
     subs=NtileX*NtileE/numthreads
     DO tile=subs*thread,subs*(thread+1)-1
       call set_data(ng, TILE)
     END DO
   END DO
!$OMP END PARALLEL DO
```

Sneaky Bit

- **Loop executed once for MPI**
- **globaldefs.h has**

```
#ifdef DISTRIBUTE
#define TILE MyRank
#else
#define TILE tile
#endif
```

- **MyRank is the MPI process number**

set_data

```
Subroutine set_data(ng, tile)
     use mod_param
     implicit none
     integer, intent(in) :: ng, tile
#include tile.h
     call set_data_tile(ng, tile,     &
         LBi, UBi, LBj, Ubj,          &
         IminS, ImaxS, JminS, JmaxS)
     return
End subroutine set_data
```

Array indices

- **There are two sets of array bounds here, the LBi family and the IminS family.**
  - LBi family for bounds of shared global storage (OpenMP) or for MPI task view of the tile – including the halo.
  - IminS family for bounds of local scratch space, always three grids bigger than tile interior on all sides.

set_data_tile

- This is where the real work happens
- It only does the work for its own tile
- Can have the _tile routine use modules for the variables it needs or pass them in as parameters from the non-tile routine

A Word on I/O

- The master process (0) does all the I/O, all in NetCDF
- On input, it sends the tiled fields to the respective processes
- It collects the tiled fields for output
- We now have an option to use NetCDF 4 (and MPI-I/O), but it has so far been slooooowwww

Error checking

- **ROMS now does error checking on all I/O related calls**
  - Master process broadcasts status code
  - Other processes listen for status code
  - All processes check status and exit if trouble, passing status back up the line
- **In the bad old days, you could get processes waiting on the master when the master had trouble**

More Changes

- **MPI communication costs time:**

  latency + size*bandwidth
- **We were passing too many small messages (still are, really)**
- **Combining buffers to pass up to four variables at a time can add up to noticeable savings (10-20%)**

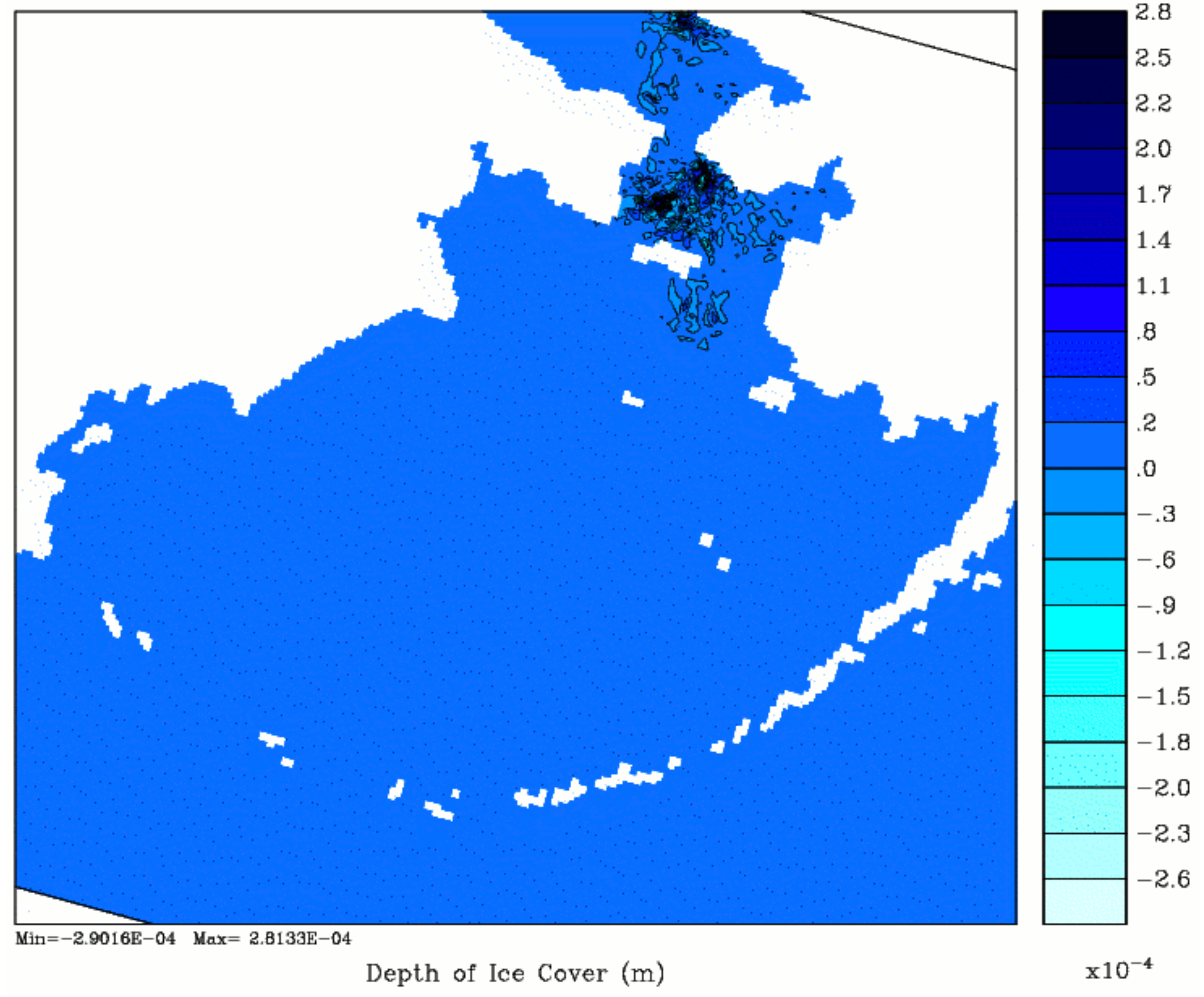
New Version

- Separate mp_exchangeXd for each of 2d, 3d, and 4d arrays
- New tile_neighbors for figuring out neighboring tile numbers (E,W,N,S) and whether or not to send
- Each mp_exchange calls tile_neighbors, then sends up to four variables in the same buffer

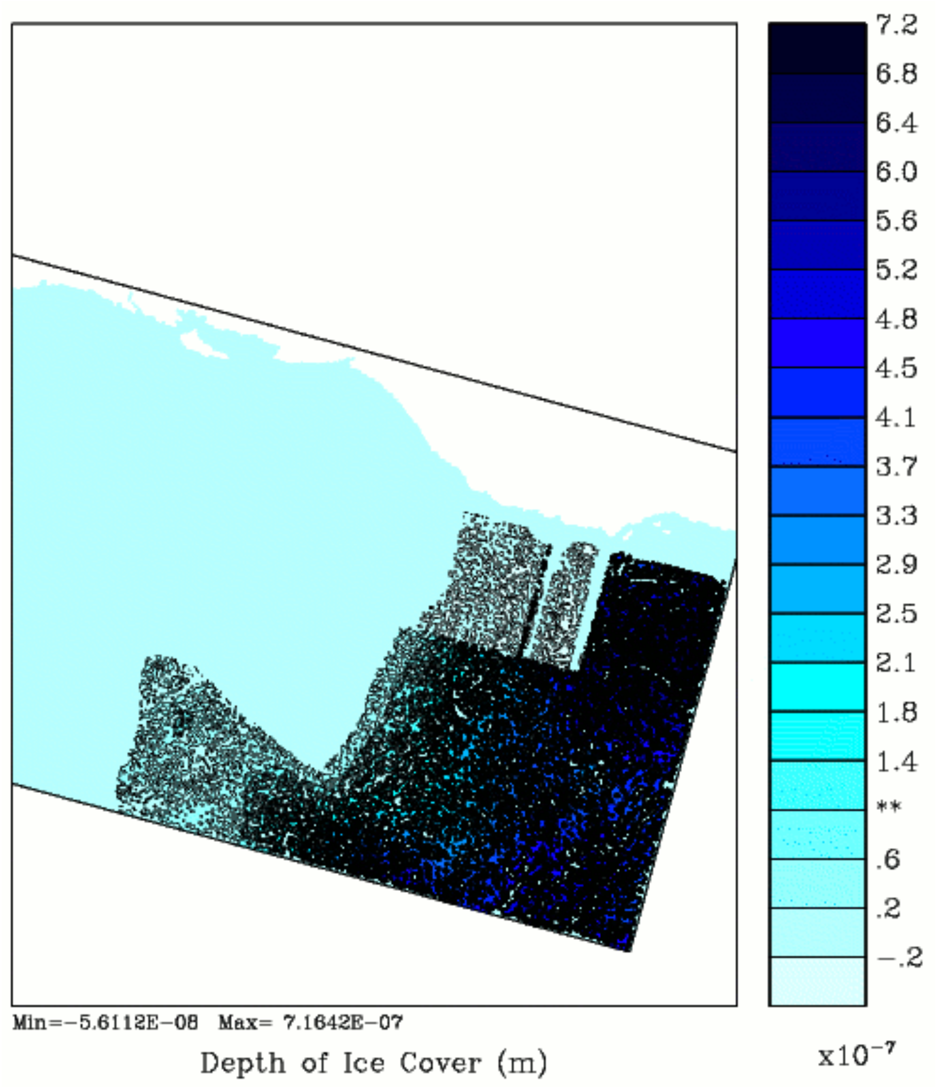
Parallel Bug Story

- It's always a good idea to compare the serial and parallel runs
- I can plot the difference field between the two outputs
- I can create a differences file with ncdiff (part of NCO)

Differences after a Day



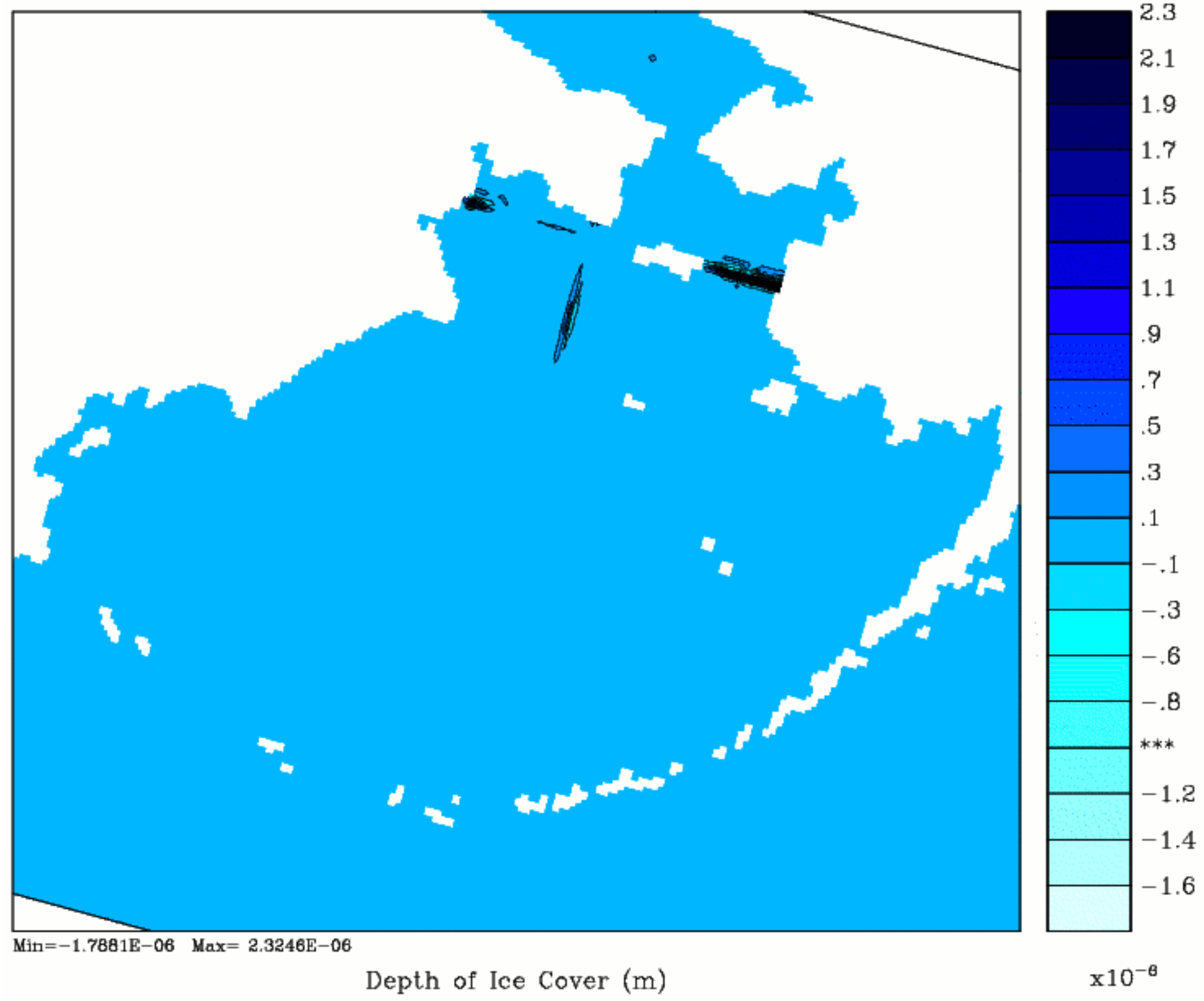
Differences after one step - in a part of the domain without ice



What' s up?

- A variable was not being initialized properly - "if" statement without an "else"
- Both serial and parallel values are random junk
- Fixing this did not fix the one-day plot

Differences after a few steps - guess where the tile boundaries are



What was That?

- The ocean code does a check for water colder than the local freezing point
- It then forms ice and tells the ice model about the new ice
- It adjusts the local temperature and salinity to account for the ice growth (warmer and saltier)
- It failed to then update the salinity and temperature ghost points
- Thinking more, I should have called the frazil ice code from step3d_t before its exchange

More...

- Plotting the differences in surface temperature after one step failed to show this
- The change was very small and the single precision plotting code couldn't catch it
- Differences did show up in timestep two of the ice variables
- Running ncdiff on the first step, then asking for the min/max values in temperature showed a problem

Debugging

- I didn't know how to use totalview in parallel then, plus it's not always available
- Find i,j value of the worst point from the diff file, print just that point - many fields
- Enclosing print statements inside if statements prevents each process from printing, possibly trying to print out-of-range values

Conclusions

- Think before coding - I can't imagine the pain of having picked the static numbering instead
- It is relatively easy for me to modify the code without fear of breaking the MPI parallelism
- Still, be prepared to check for parallel bugs