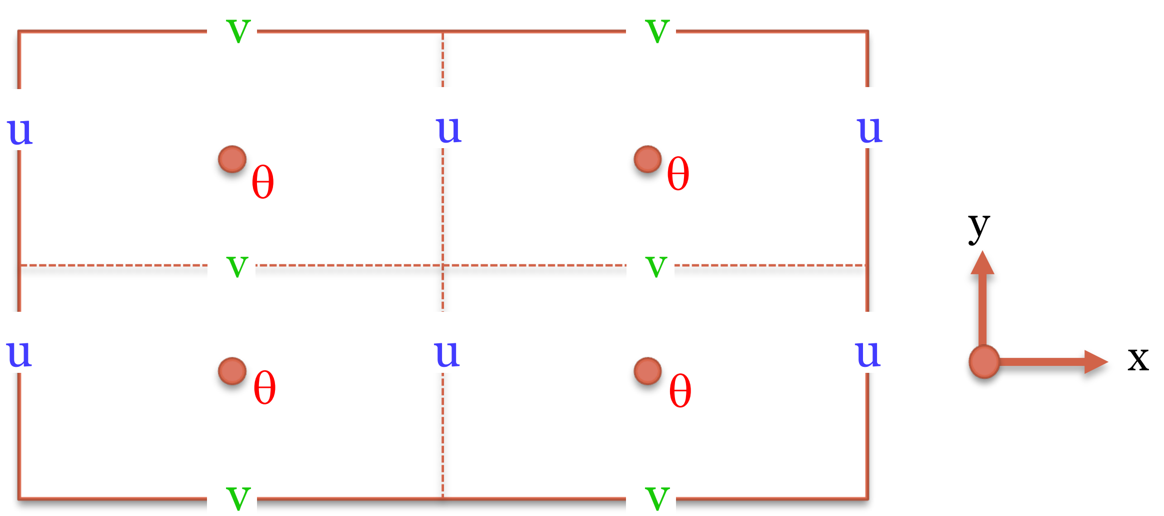
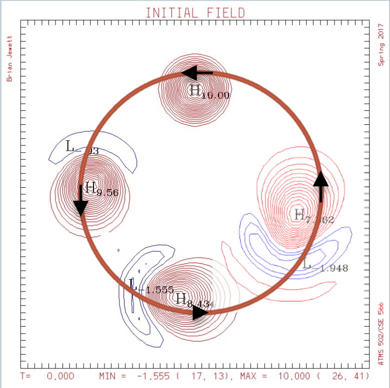
continued:
Computer Program #2

TWO-DIMENSIONAL ADVECTION

2-D Staggered "C-grid"



Program 2 - Review



- We are using *circular flow* …
  - so the easy 'perfect' solution we compare to … = the initial condition (also true in Program 1)
- Arrays etc.
  - Add array: *v* (the y-velocity)
  - Change arrays: to be 2-D, **nx • ny**
  - Set #ghost points to be: 3 (three points each side of domain)
- Initial condition:
  - *s1* "cone" • *u,v*: rotational flow
- Boundary condition: "zero-gradient" *(discuss)*
- Domain: staggered C-grid in 2-D *(discuss)*

Program 2 – IC and physical domain

IC routine for FORTRAN:

- do j = 1,ny
  - do i = 1,nx
    - x = -0.5 + dx*real(i-1)
    - y = -0.5 + dy*real(j-1)
    - *add code for* d = …
    - *add code for* s1(i,j) = …
- do-loops for u, v:
  - u: i=1,nx+1
    - x = (formula above) - dx/2
  - v: j = 1,ny+1
    - y = (formula above) - dy/2

j loop unchanged:
y values unchanged

i loop unchanged:
x values unchanged

IC routine for C:

- for (j=J1; j<=J2; j++) {
  - for (i=I1; i<=I2; i++) {
    - x = -0.5 + dx*(float)(i-I1);
    - y = -0.5 + dy*(float)(j-J1);
    - *add code for* d = …
    - *add code for* s1[i][j] = …
- for-loops for u, v:
  - u: (i=I1; i<=I2+1; i++)
    - x = (formula above) - dx/2;
  - v: (j =J1; j<=J2+1; j++)
    - y = (formula above) - dy/2;

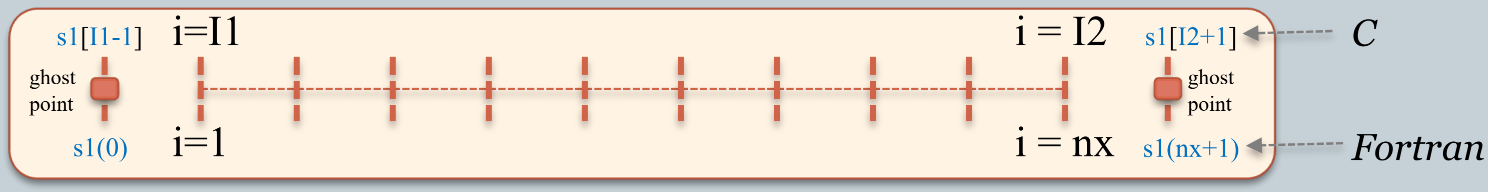
j loop unchanged:
y values unchanged

i loop unchanged:
x values unchanged

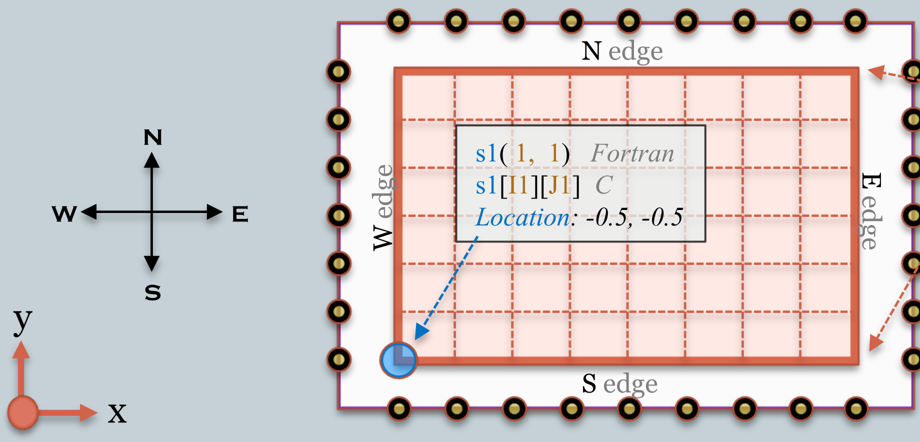
Program 2 – boundary conditions (BCs)

FYI …
halo regions

- In program 1, our domain was 1-D
  - setting BCs required just *two statements* with *1 ghost point*.



- In program 2, for this *2-D* setting …
  - Our BCs now require *two loops*[1] and *3 ghost points*[2]



The **physical** domain is shown in light red. In this illustration, it is surrounded by a 1-point-wide halo (ghost zone) region. You need to set all ghost points shown with **yellow circles**
(note our 1-D advection method never uses corner points!)

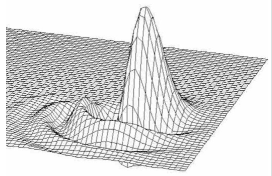
The easiest way to set these ghost point values is with 2 loops:

1. Loop over all **X** columns; set ghost points below S edge and above N edge, e.g. *loop* i=1,nx : *set* s1(i, 0) *and* s1(i, ny+1)
2. Loop over **Y** rows; set ghost points to left of W edge & right of E edge, e.g.: *loop* j=1,ny : *set* s1(0, j) *and* s1(nx+1, j).

This is a 1-ghost-point example. Program #2 needs **3** points!

[1]for *two* spatial dimensions. In 3-D, we'll need triple loops • [2]for 6th-order Crowley

Program 2 – coding Advection in 2-D



- **Advection**
  - I set up temporary 1-D arrays in my 2-D advection routine –
    - s1d(-2:nx+3), vel1d(nx+1) ***no*** *ghost points for U, V !!* until we do nonlinear advection, and u, v are evolved in time, just like s1()

      for the scalar field *for a velocity (u or v) field*

- Advecting rows (X)
  - copy s1(i,j) to s1d
  - copy u(i,j) to vel1d
  - pass s1d, vel1d to *advect1d()*
    - advect1d returns *s1d_out*
  - copy s1d_out(i) to s1(i,j) ***not including*** [2] *ghost points!!*

within outer *j* loop, for y

Each bullet is a 1-D loop; I use index ***i*** **...** for the x-direction. All are within the outer ***j*** loop. Note: in Fortran, can do this w/subscript notation!

- Coding X advection:
  - loop over all *j* rows
    - loop over all *i* columns ***including*** [1] *ghost points!!*
      - copy s1(i,j) to s1d(i)
    - loop over all *i* (nx+1) columns
      - copy u(i,j) to vel1d(i)
    - call *advect1d( )*
    - loop over all *i* (nx) columns [2]
      - copy s1d_out(i) to s1(i,j)

j *i* *i* *i*

[1]so advect1d() doesn't have to worry about BCs [2]advect1d() does **not** set ghost points!