FLASH3 Boundary Conditions

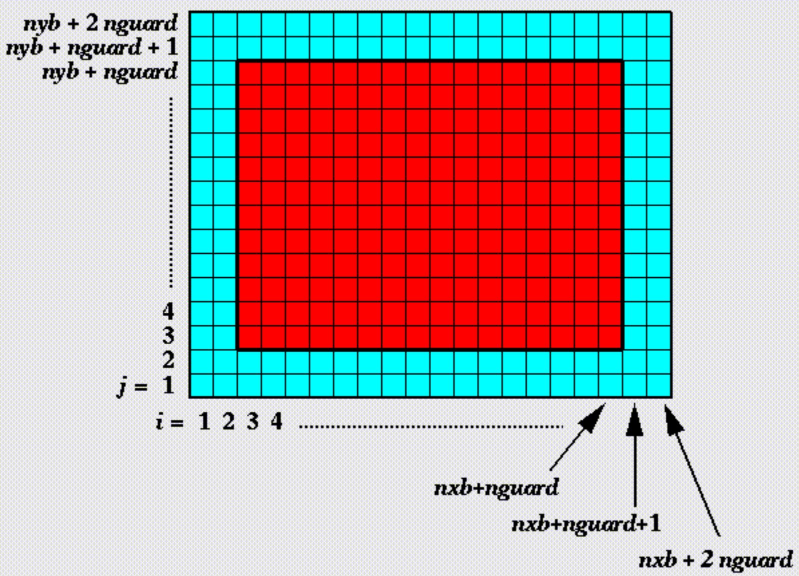
Flash Tutorial
June 23, 2009
Dr. Klaus Weide

An Advanced Simulation & Computing (ASC)
Academic Strategic Alliance Program (ASAP) Center
at The University of Chicago

Grid Boundary Conditions

- Grid Boundary Conditions
  - A subunit by Grid, included by default when Grid is used
  - Implements Grid Boundary Conditions == Fluid Boundary Conditions
    Runtime parameters xl_boundary_type, xr_boundary_type, ...
  - Gravity boundary Conditions are different:
    Runtime parameter grav_boundary_type
- Grid Boundary Conditions are implemented (only!) as part of Guard Cell Filling.
  - No separate high-level call to fill boundary cells.
  - FLASH provides implementation that gets called by PARAMESH4 for each block (and each guard cell region).

Repeat overview: blocks and cells

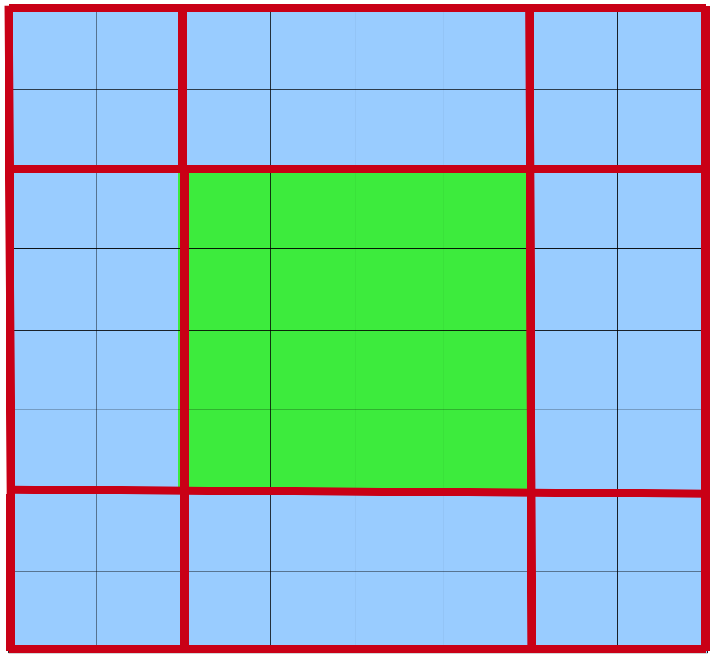


Indexing scheme for a 2D grid block with 2 guard cells (nguard=2) at each block boundary. Interior cells are colored red, guard cells are colored blue.

- The grid is composed of blocks
- FLASH3: In current practice, all blocks are of same size.
- May cover different fraction of the physical domain, depending on a block's resolution.
- Each, block reserves space for some layers of guard cells.

Filling guard cells I

- For purposes of guard cell filling, guard cells are organized into guard cell regions.



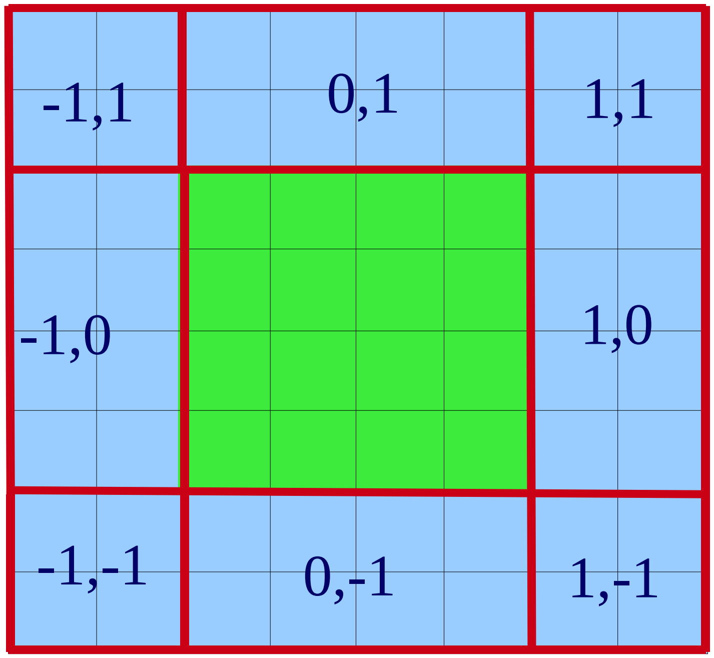
- During guard cell filling, each guard cell region may get filled from a different data source:
  - A local neighbor block
  - A remote neighbor block
  - A boundary condition
    - using data from adjacent interior cells
    - Using fixed or coordinate-based data
  - Interpolation from parent (if the block touches a fine/coarse boundary)

Filling guard cells Ia

- For purposes of guard cell filling, guard cells are organized into guard cell regions.

In 2D, a block has 8 guard cell regions.

In 3D, a block has 26 guard cell regions!



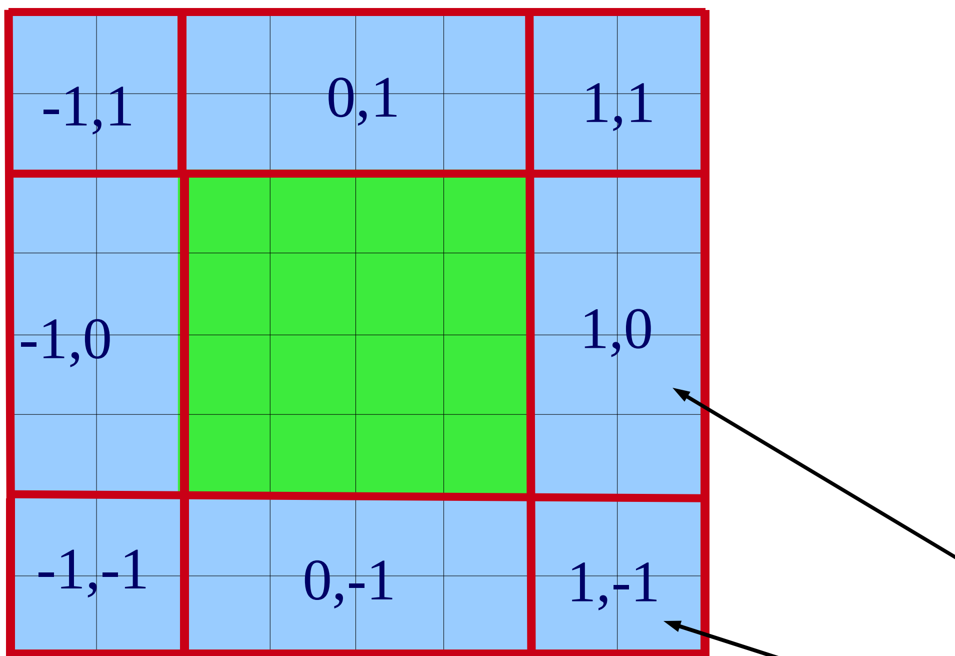
- During guard cell filling, each guard cell region may get filled from a different data source:
  - A local neighbor block
  - A remote neighbor block
  - A boundary condition
    - using data from adjacent interior cells
    - Using fixed or coordinate-based data
  - Interpolation from parent (if the block touches a fine/coarse boundary)

Filling guard cells Ib

- For purposes of guard cell filling, guard cells are organized into guard cell regions.

In 2D, a block has 8 guard cell regions.

In 3D, a block has 26 guard cell regions!



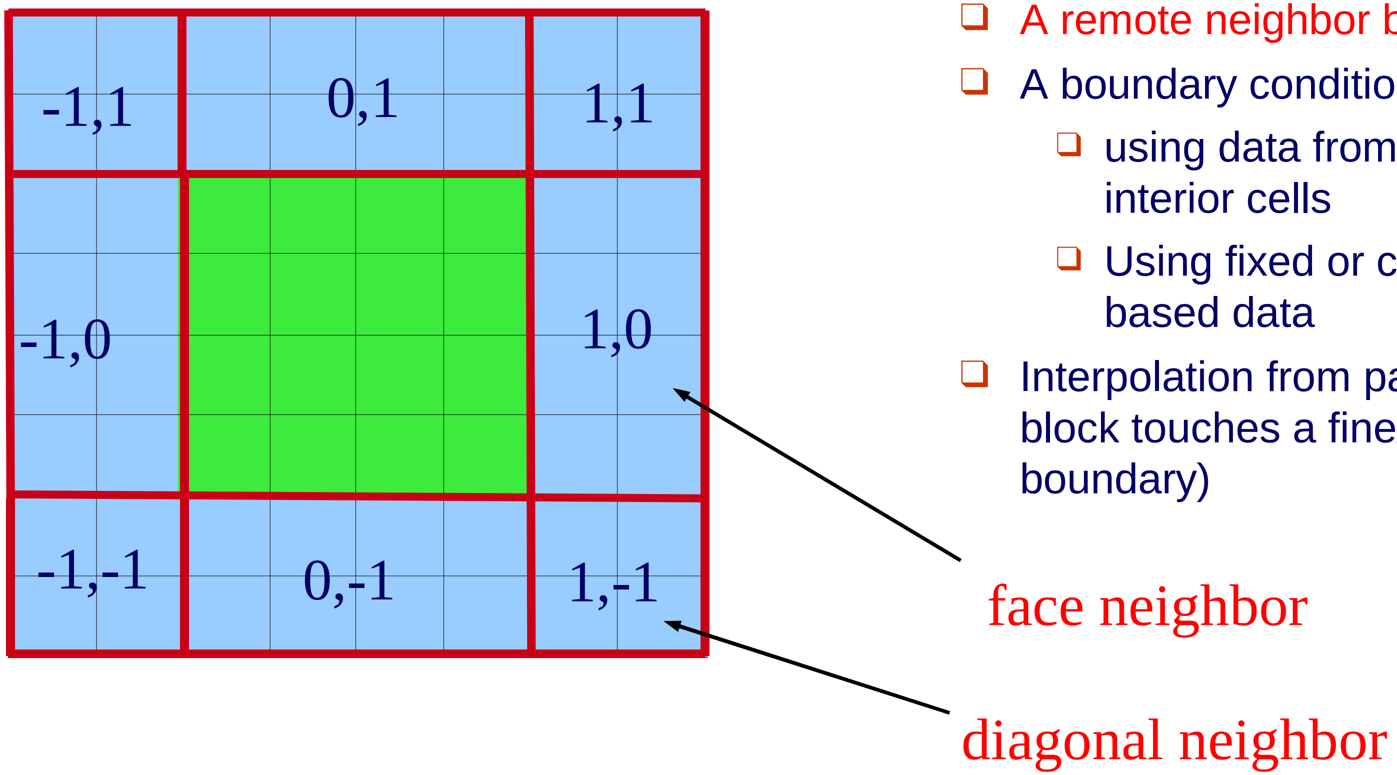
- During guard cell filling, each guard cell region may get filled from a different data source:
  - A local neighbor block
  - A remote neighbor block
  - A boundary condition
    - using data from adjacent interior cells
    - Using fixed or coordinate-based data
  - Interpolation from parent (if the block touches a fine/coarse boundary)

Filling guard cells Ic

- For purposes of guard cell filling, guard cells are organized into guard cell regions.

In 2D, a block has 8 guard cell regions.

In 3D, a block has 26 guard cell regions!

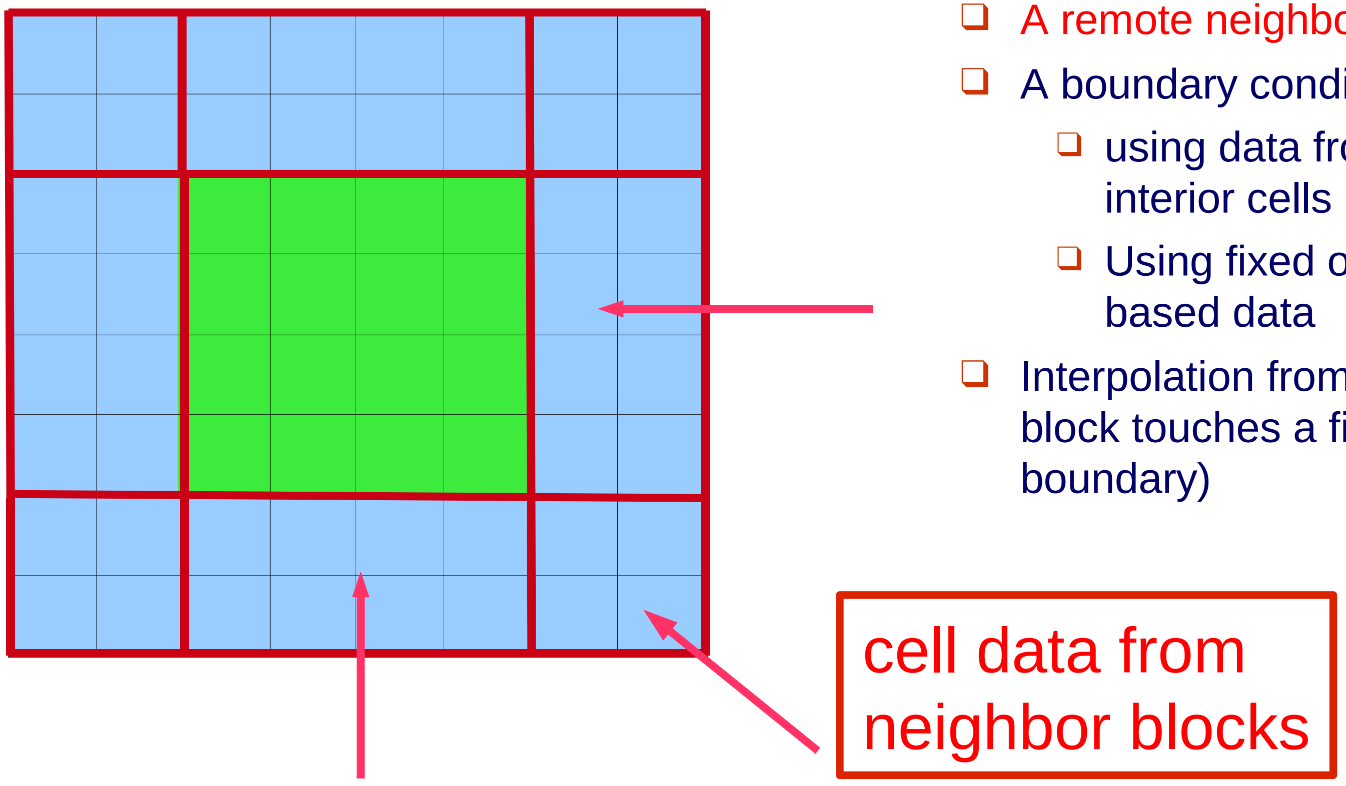


- During guard cell filling, each guard cell region may get filled from a different data source:
  - A local neighbor block
  - A remote neighbor block
  - A boundary condition
    - using data from adjacent interior cells
    - Using fixed or coordinate-based data
  - Interpolation from parent (if the block touches a fine/coarse boundary)

Filling guard cells from neighbors I

- For purposes of guard cell filling, guard cells are organized into guard cell regions.

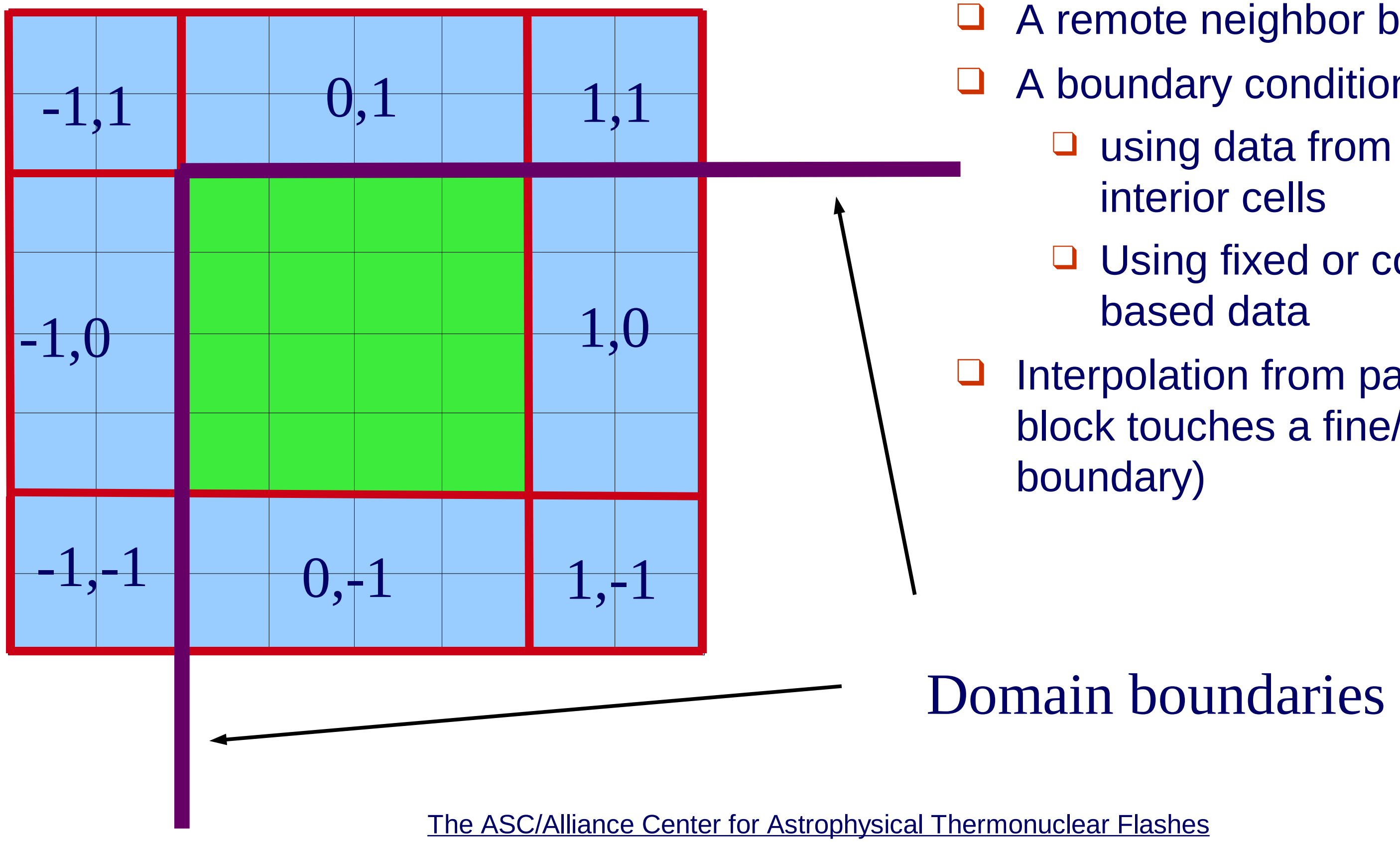
- During guard cell filling, each guard cell region may get filled from a different data source:
  - A local neighbor block
  - A remote neighbor block
  - A boundary condition
    - using data from adjacent interior cells
    - Using fixed or coordinate-based data
  - Interpolation from parent (if the block touches a fine/coarse boundary)



Filling guard cells at Boundary I

- For purposes of guard cell filling, guard cells are organized into guard cell regions.

Now assume a block at the **corner of the domain**:

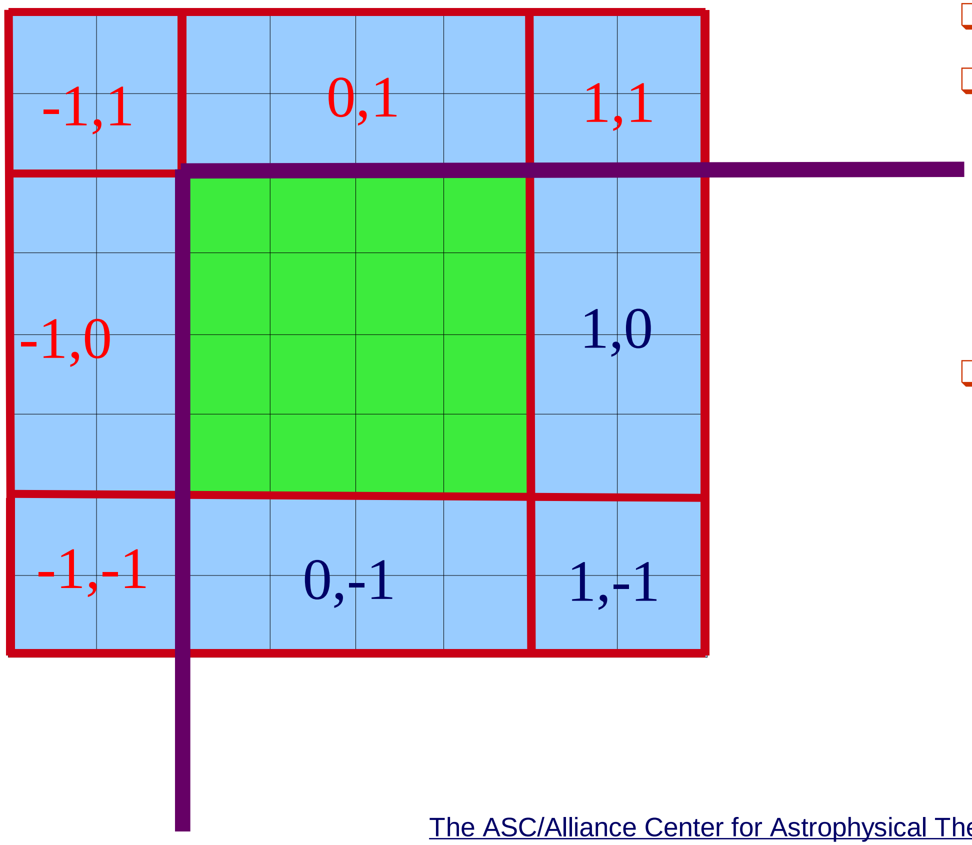


- During guard cell filling, each guard cell region may get filled from a different data source:
  - A local neighbor block
  - A remote neighbor block
  - A boundary condition
    - using data from adjacent interior cells
    - Using fixed or coordinate-based data
  - Interpolation from parent (if the block touches a fine/coarse boundary)

Filling guard cells at Boundary II

- For purposes of guard cell filling, guard cells are organized into guard cell regions.

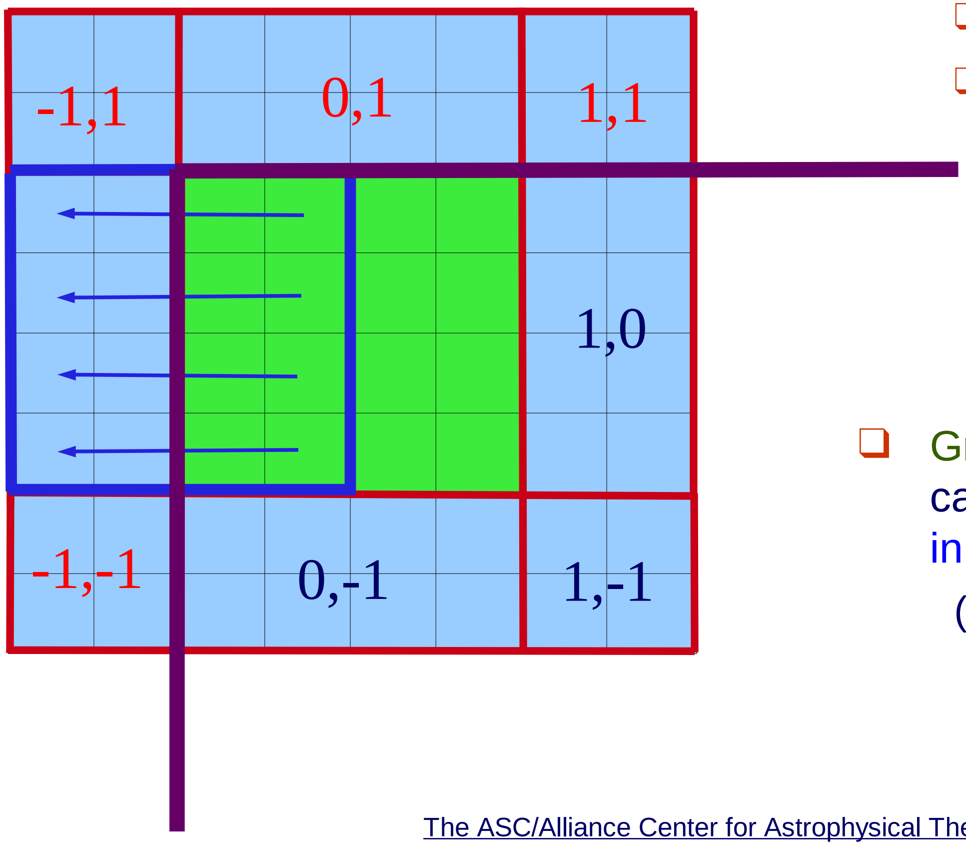
The guard cell regions in red represent locations **outside of the domain**:



- During guard cell filling, each guard cell region may get filled from a different data source:
  - A local neighbor block
  - A remote neighbor block
  - A boundary condition
    - using data from adjacent interior cells
    - Using fixed or coordinate-based data
  - Interpolation from parent (if the block touches a fine/coarse boundary)

Filling guard cells at Boundary III

- For purposes of guard cell filling, guard cells are organized into guard cell regions.



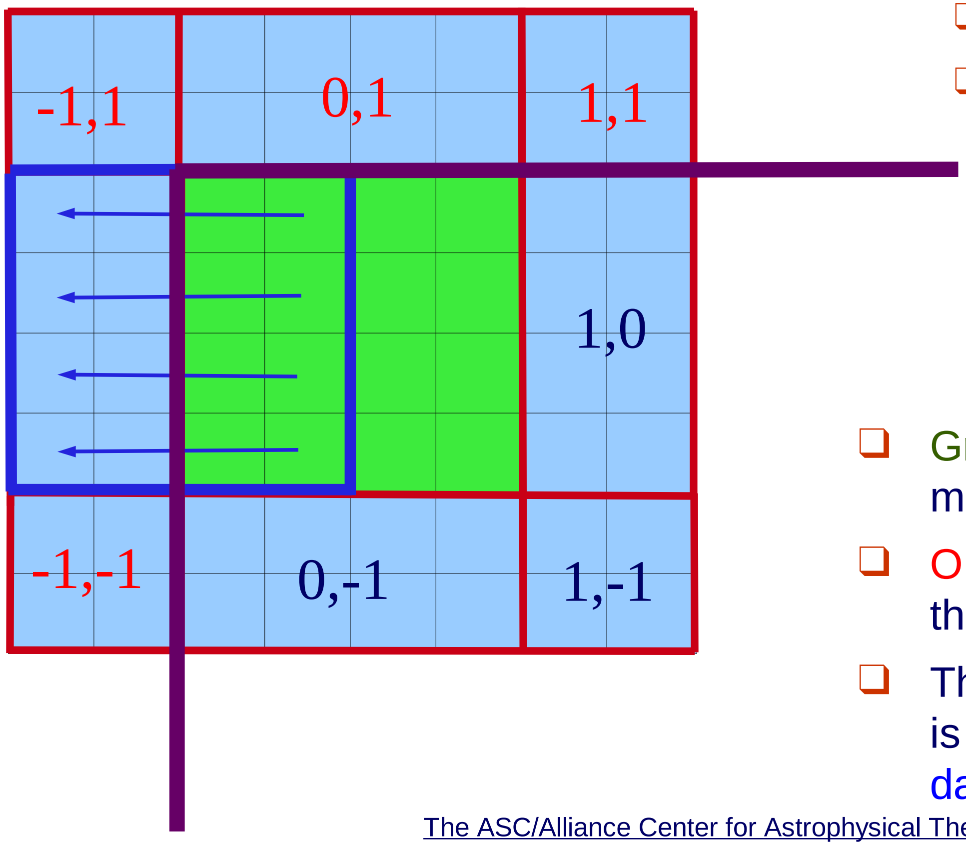
- During guard cell filling, each guard cell region may get filled from a different data source:
  - A local neighbor block
  - A remote neighbor block
  - A boundary condition
    - using data from adjacent interior cells
    - Using fixed or coordinate-based data

- Grid_bcApplyToRegionSpecialized is called and passed a pointer to the data in the blue region.

  (actually, a copy of the block data)

Filling guard cells at Boundary IV

- For purposes of guard cell filling, guard cells are organized into guard cell regions.



- During guard cell filling, each guard cell region may get filled from a different data source:
  - A local neighbor block
  - A remote neighbor block
  - A boundary condition
    - using data from adjacent interior cells
    - Using fixed or coordinate-based data
- Grid_bcApplyToRegionSpecialized may fill in the guard cell region.
- OR it may decline to handle this, and then:
- The subroutine Grid_bcApplyToRegion is called and passed a pointer to the data in the blue region.

Implementing Boundary Conditions

- Grid_bcApplyToRegionSpecialized gets called first
  - This is normally a no-op stub
  - This is the preferred place to users to hook in customized implementations.
  - This interface provided more information to an implementation than Grid_bcApplyToRegion, most importantly:
    - A block handle (usually, block ID) identifying the block being filled
    - Location of the data region within the Grid block
  - May decide to handle the call, based on BC type, direction, ...
  - Before returning, sets “`applied`” flag to signal that the BC was handled.

- Grid_bcApplyToRegion gets called if Grid_bcApplyToRegionSpecialized did not handle the case.
  - The standard implementation of Grid_bcApplyToRegion in source/Grid/GridBoundaryConditions provides the standard simple BC types: REFLECTING, OUTFLOW, DIODE, ...
  - It is a good place to start if you need to write your own!

BCs – Complications

- Grid_bcApplyToRegion* may be called on a non-LEAF block.
- Grid_bcApplyToRegion* may be called on a block that is not even local!
  - This can happen if a parent block needs to be filled to provide input data for interpolation, and the parent resides on a different PE from the leaf.
  - Simple BC methods don't have to be aware of this.
  - But if your method depends on coodinate information, or needs to access the block by its ID, beware!
  - See source/Grid/GridBoundaryConditions/README and Users Guide in those cases.
- The data region passed to Grid_bcApplyToRegion* is in transposed form:

  Reference it like regionData(I,J,k,ivar), where
  - I counts cells in the normal direction (NOT always: x direction!),
  - J,K cont cells in the other directions
  - Ivar counts variables

  This is convenient for implementing simple BC where location does not matter, but complicates things if you need to know where a cell is within the block.
  - Use provided examples!

BCs – Simplifications

- If you prefer a simpler interface:
  - Handle one data row at a time (vector of data in normal direction)
  - Powerful enough to implement hydrostatic boundaries
  - REQUIRES Grid/GridBoundaryConditions/OneRow (see source files there!)
  - Implements a version of Grid_bcApplyToRegionSpecialized
  - Provides functions Grid_applyBCEdge, Grid_applyBCEdgeAllUnkVars
  - Too customize, user should provide own implementation of Grid_applyBCEdge.F90 (or Grid_applyBCEdgeAllUnkVars.F90)

Hydrostatic Boundary Conditions

- The ones provided are ported from FLASH2 and probably not the best implementation. You may want to write your own!
- To use: REQUIRES Grid/GridBoundaryConditions/Flash2HSE
- Works by implementing Grid_bcApplyToRegionSpecialized, which calls a function gr_applyFlash2HSEBC.F90 on rows (i.e., vectors) of data

  Grid/GridBoundaryConditions/Flash2HSE/Grid_bcApplyToRegionSpecialized.F90 may be a good template for your own implementation of BCs.
- To use, in flash.par:
  - xl_boundary_type = “hydrostatic-F2+nvout” # etc.
  - xl_boundary_type = “hydrostatic-F2+nvrefl” # etc.
  - xl_boundary_type = “hydrostatic-F2+nvdiode” # etc.
- The three variants differ in the handling of normal velocities.