An Introduction to AstroBEAR

Baowei Liu

University of Rochester

August 28, 2013

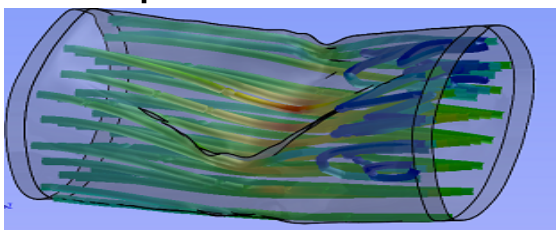
HD/MHD Equations Solver

Fluid Dynamics Equations Solver

HD/MHD Equations Solver

Fluid Dynamics Equations Solver

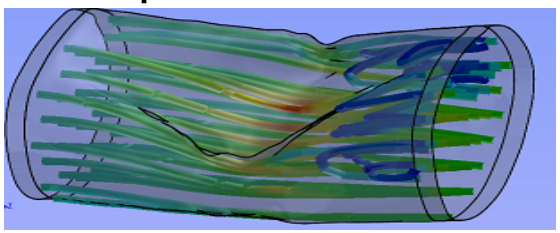
Petroleum Pipelines



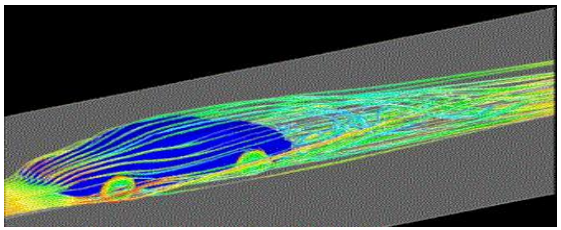
HD/MHD Equations Solver

Fluid Dynamics Equations Solver

Petroleum Pipelines



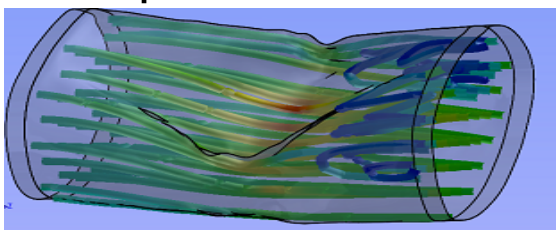
Aircraft/Vehicle



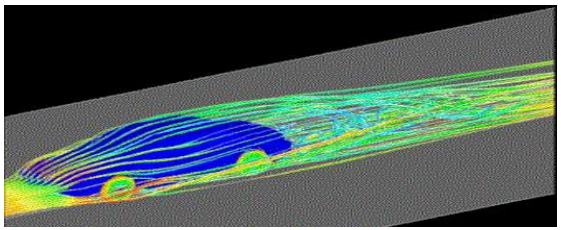
HD/MHD Equations Solver

Fluid Dynamics Equations Solver

Petroleum Pipelines



Aircraft/Vehicle

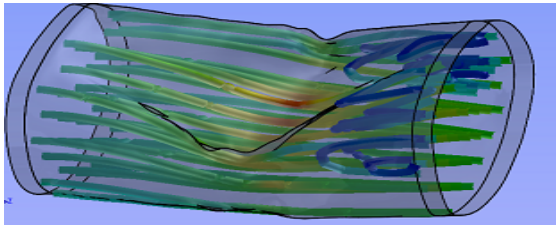


Planetary Nebulae

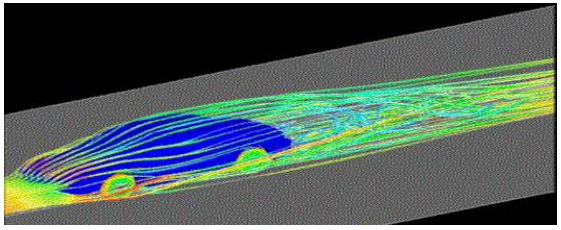
HD/MHD Equations Solver

Fluid Dynamics Equations Solver

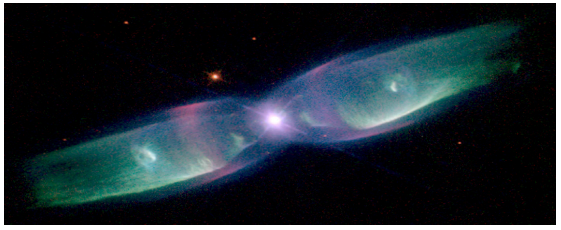
Petroleum Pipelines



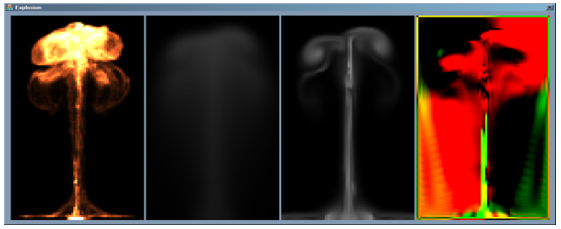
Aircraft/Vehicle



Planetary Nebulae



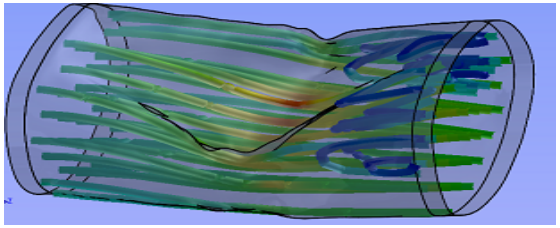
Explosion Simulation



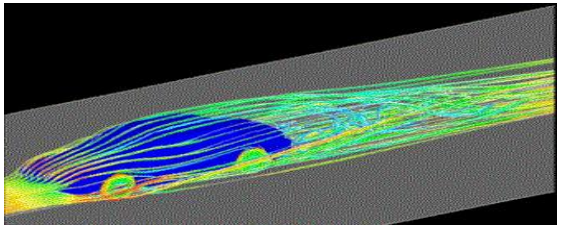
HD/MHD Equations Solver

Fluid Dynamics Equations Solver

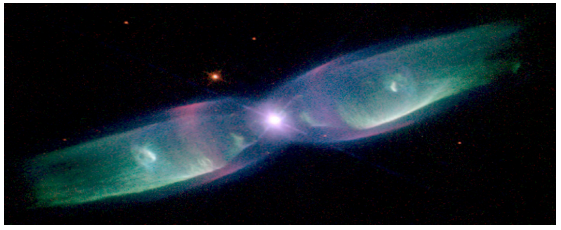
Petroleum Pipelines



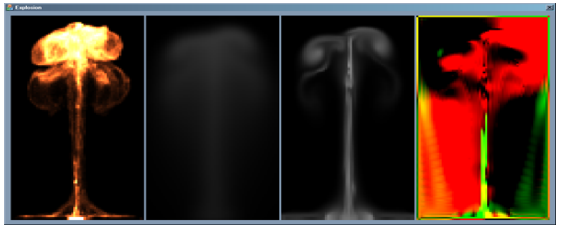
Aircraft/Vehicle



Planetary Nebulae



Explosion Simulation

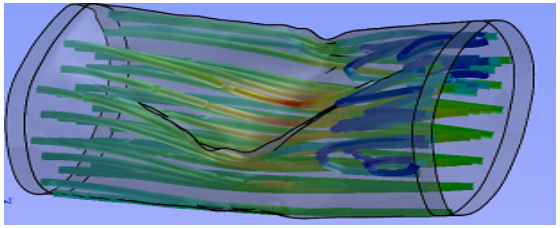


Results from AstroBEAR:

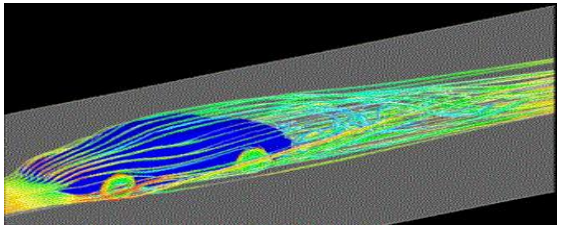
HD/MHD Equations Solver

Fluid Dynamics Equations Solver

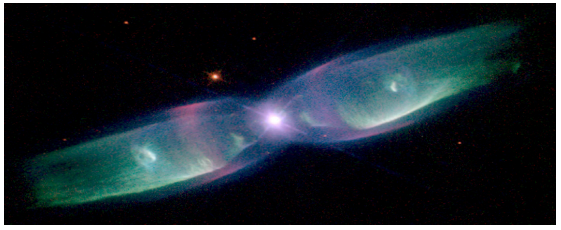
Petroleum Pipelines



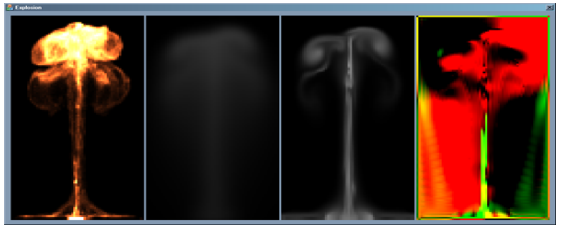
Aircraft/Vehicle



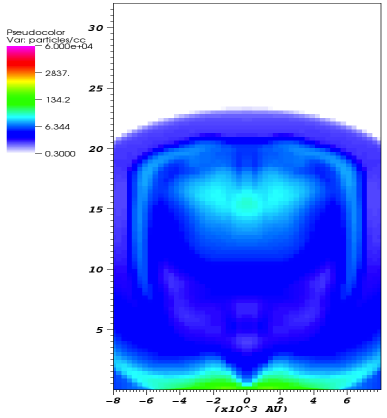
Planetary Nebulae



Explosion Simulation



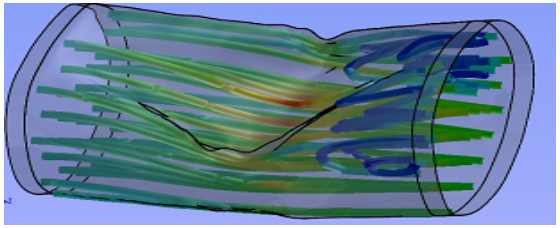
Results from AstroBEAR:



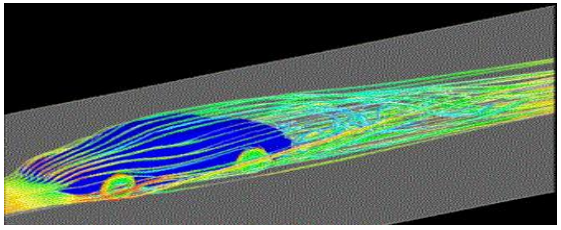
HD/MHD Equations Solver

Fluid Dynamics Equations Solver

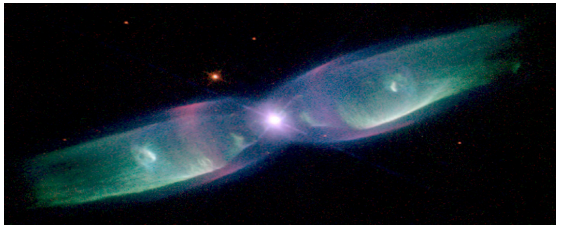
Petroleum Pipelines



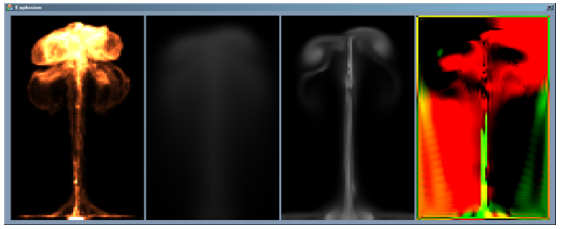
Aircraft/Vehicle



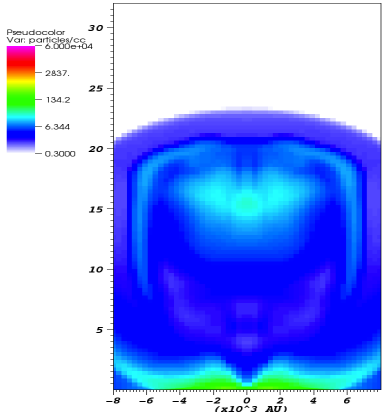
Planetary Nebulae



Explosion Simulation

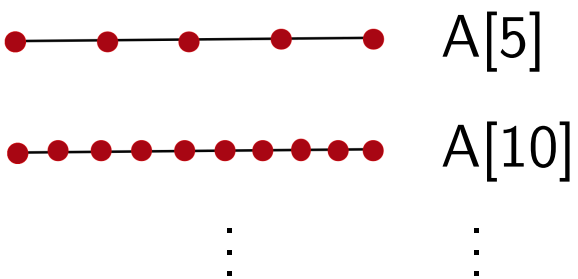


Results from AstroBEAR:

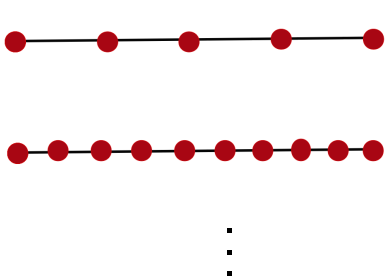


Adaptive Mesh Refinement

Adaptive Mesh Refinement



Adaptive Mesh Refinement



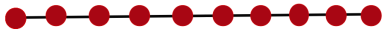
finer grid

higher accuracy

more resources

Adaptive Mesh Refinement

A[5]



A[10]

finer grid

higher accuracy

more resources

AMR: Only refine critical zones
keep a balance between
resources and accuracy.

Adaptive Mesh Refinement

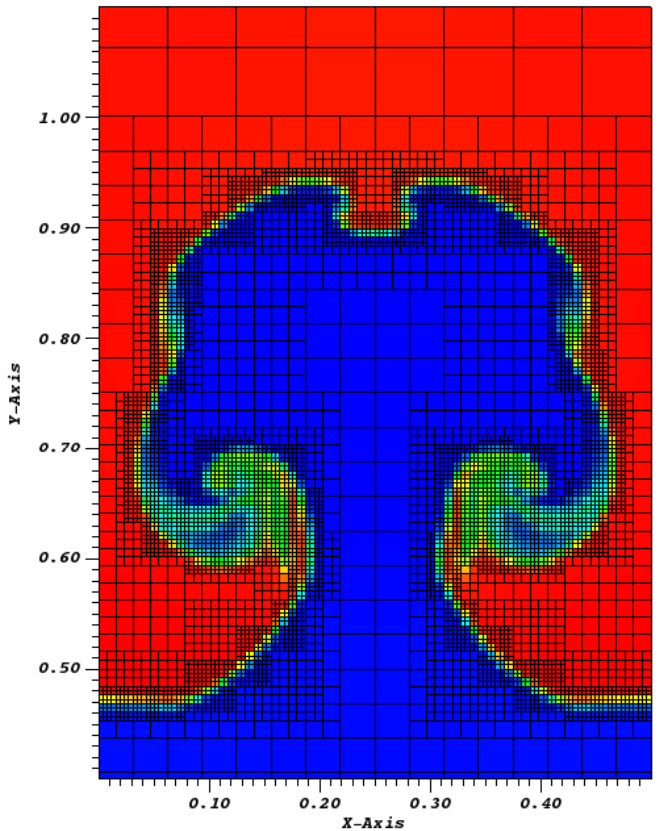

A[5]


A[10]

finer grid

higher accuracy

more resources



AMR: Only refine critical zones
keep a balance between
resources and accuracy.

Adaptive Mesh Refinement

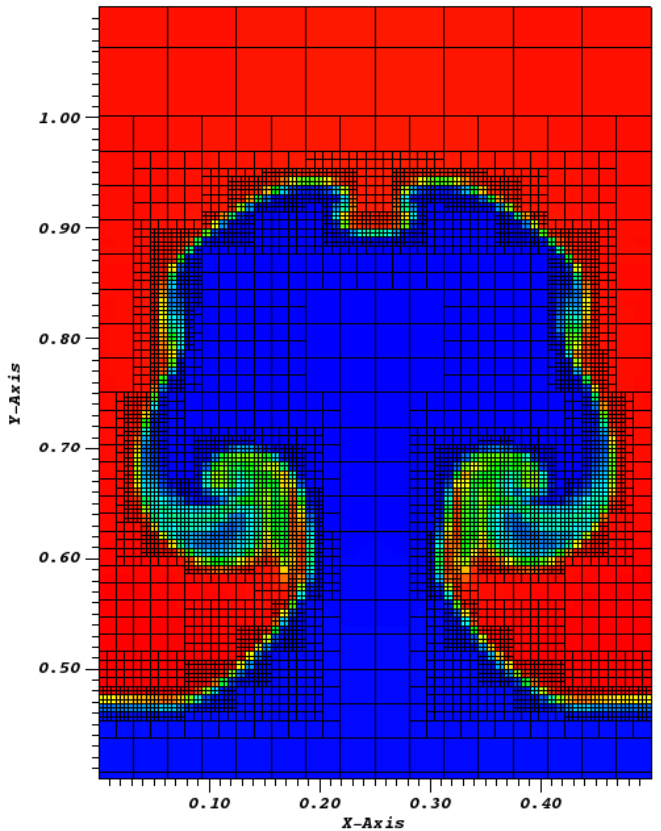

A[5]


A[10]

finer grid

higher accuracy

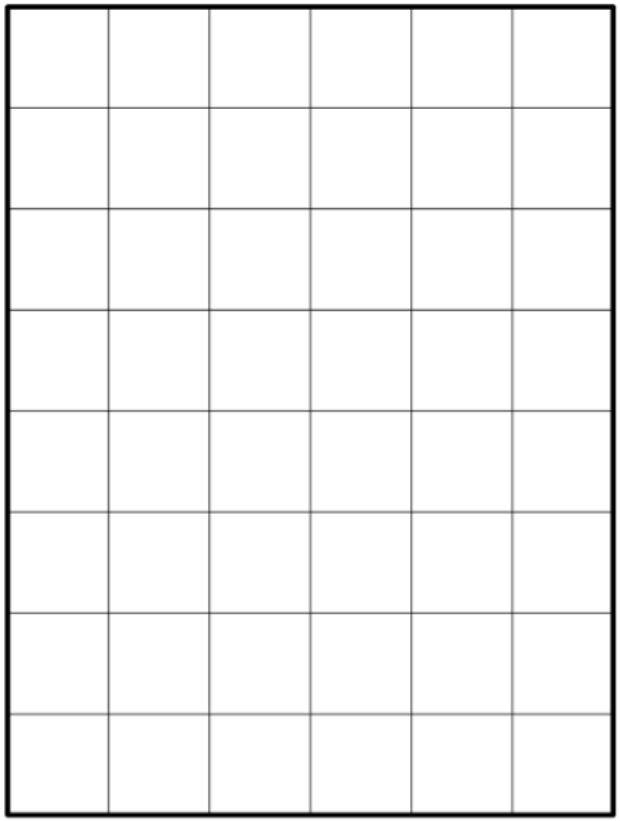
more resources



AMR: Only refine critical zones keep a balance between resources and accuracy.

BEAR:
Boundary Embedded Adaptive Refinement

AMR Tree

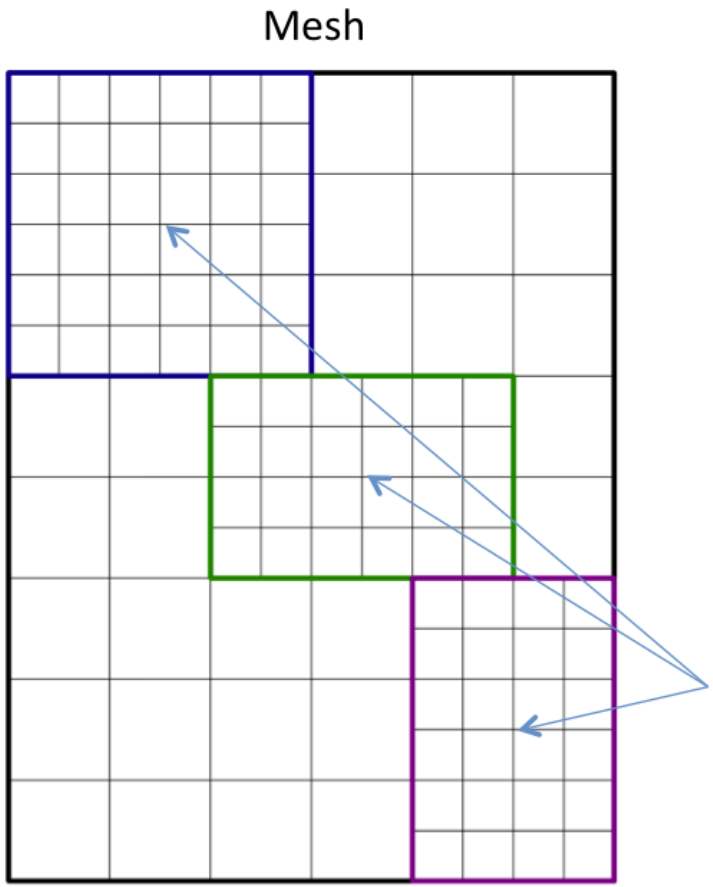


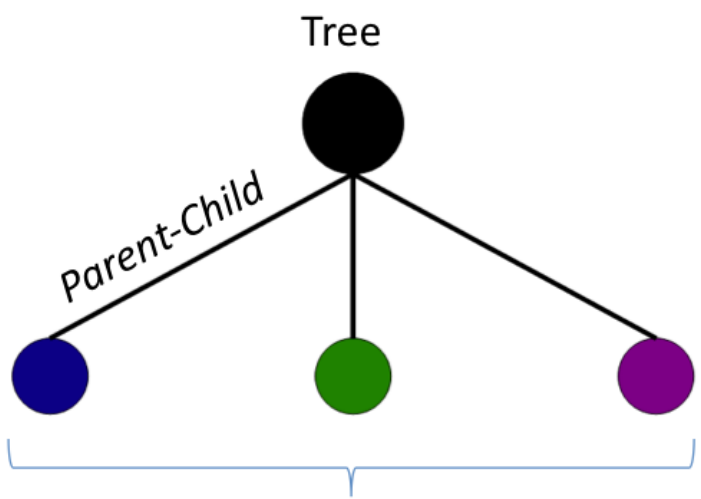
Level 0 base grid

Level 0 node

Grids store fluid quantities in a regular array of cells on a single processor. Nodes contain the meta-data that includes the physical location of the grid, and the processor containing the grids data.

AMR Tree



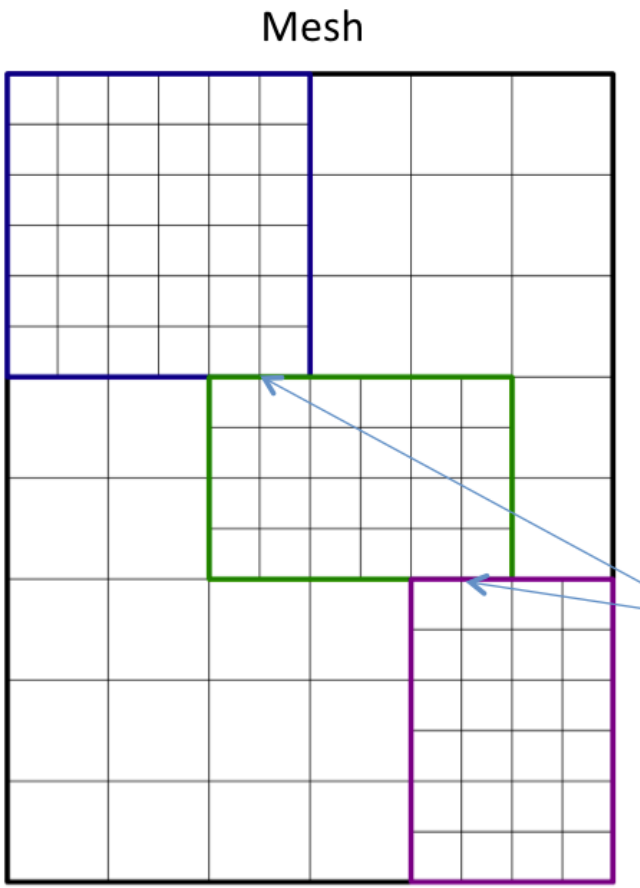


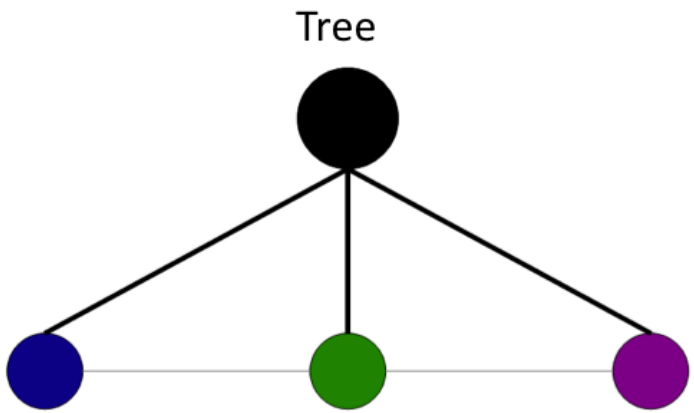
Level 1 nodes are children of level 0 node

Level 1 grids nested within parent level 0 grid

Physical relationships between grids in the mesh are reflected in connections between their corresponding nodes in the AMR tree

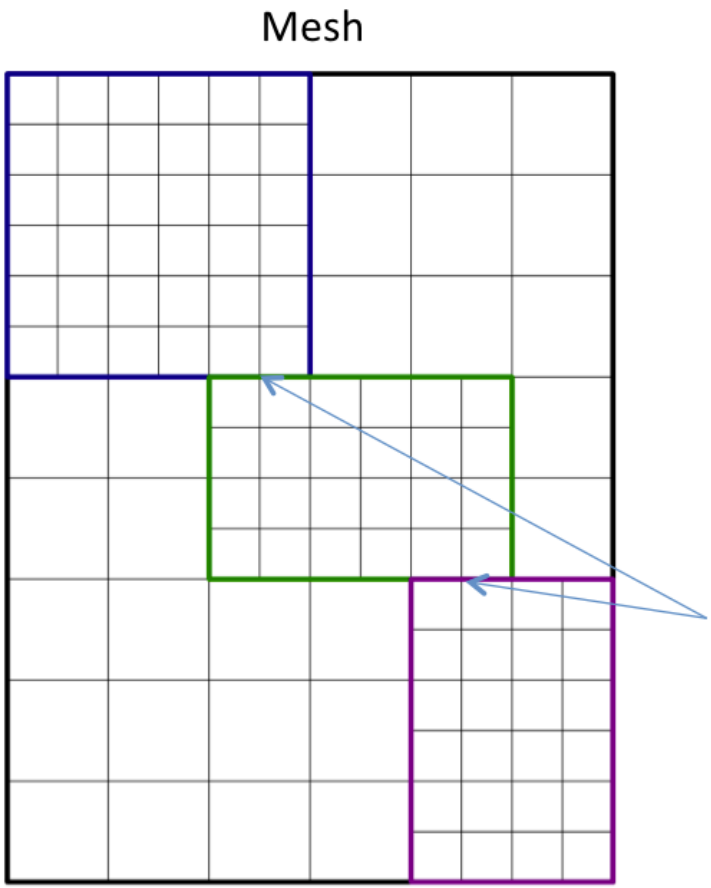
AMR Tree

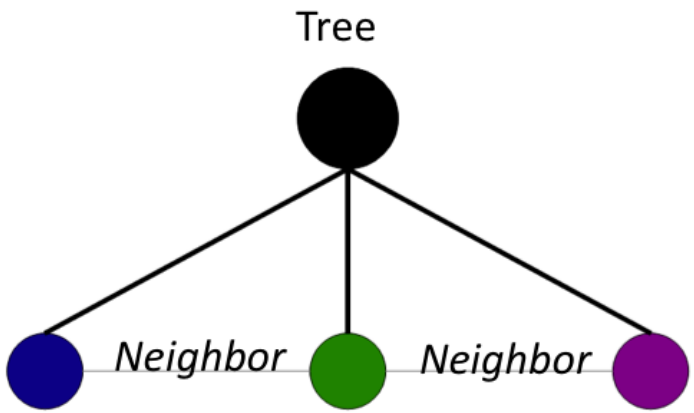




Conservative methods require synchronization of fluxes between neighboring grids as well as the exchanging of ghost zones

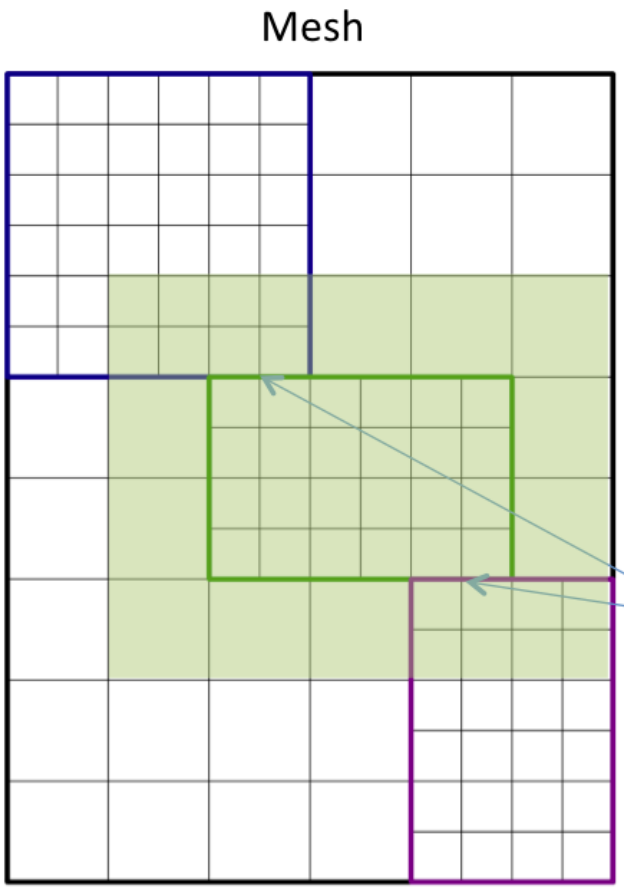
AMR Tree

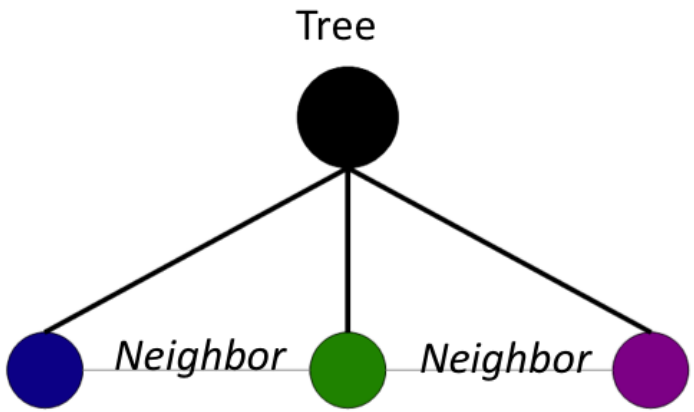




Conservative methods require synchronization of fluxes between neighboring grids as well as the exchanging of ghost zones

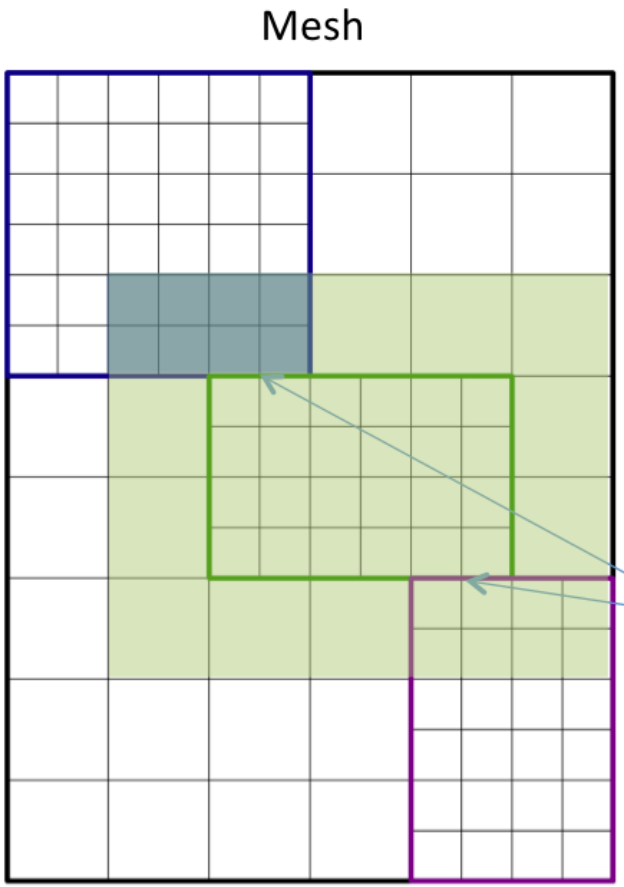
AMR Tree

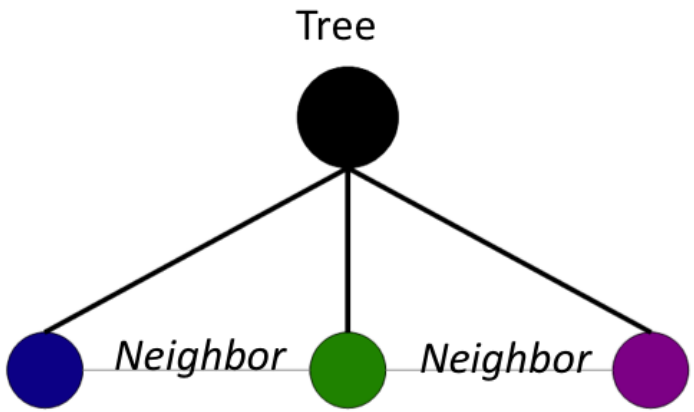




Conservative methods require synchronization of fluxes between neighboring grids as well as the exchanging of ghost zones

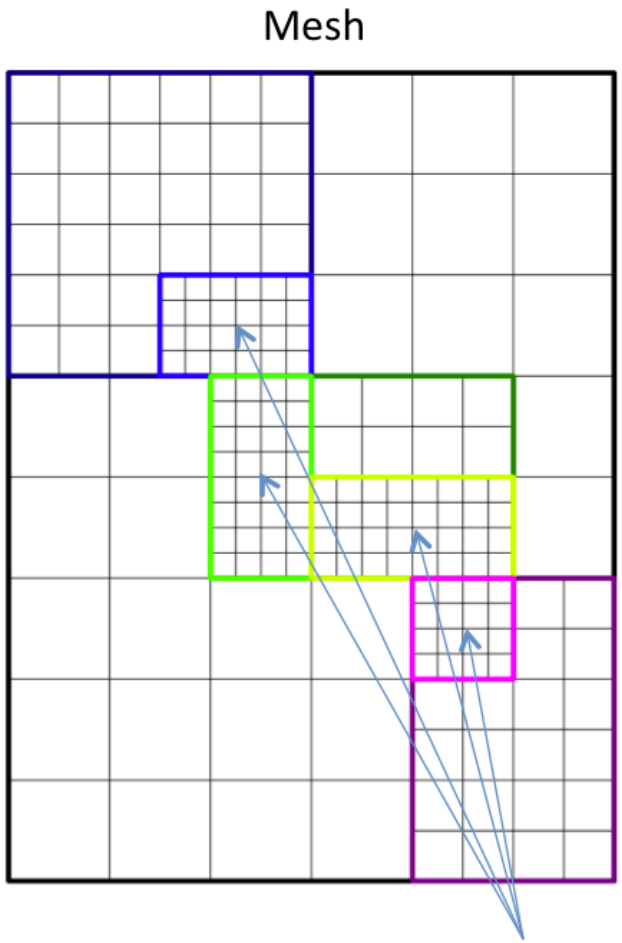
AMR Tree

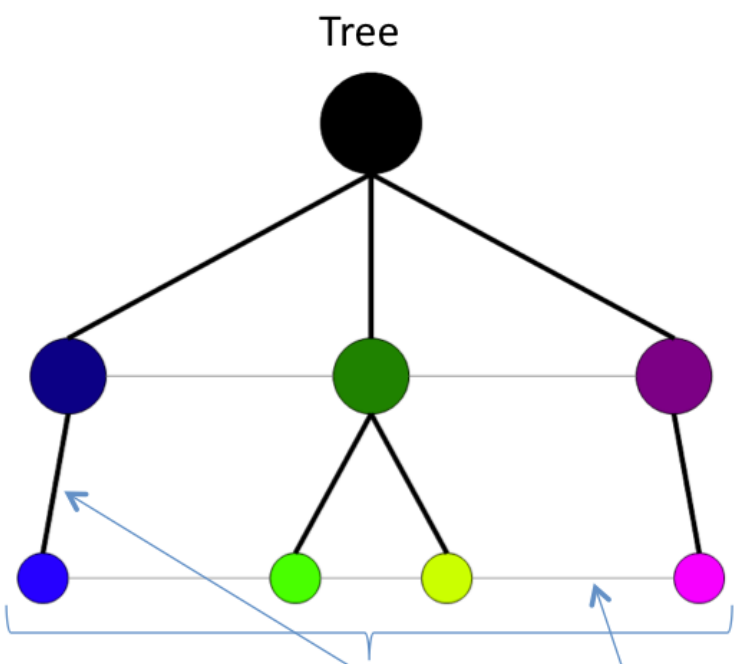




Conservative methods require synchronization of fluxes between neighboring grids as well as the exchanging of ghost zones

AMR Tree





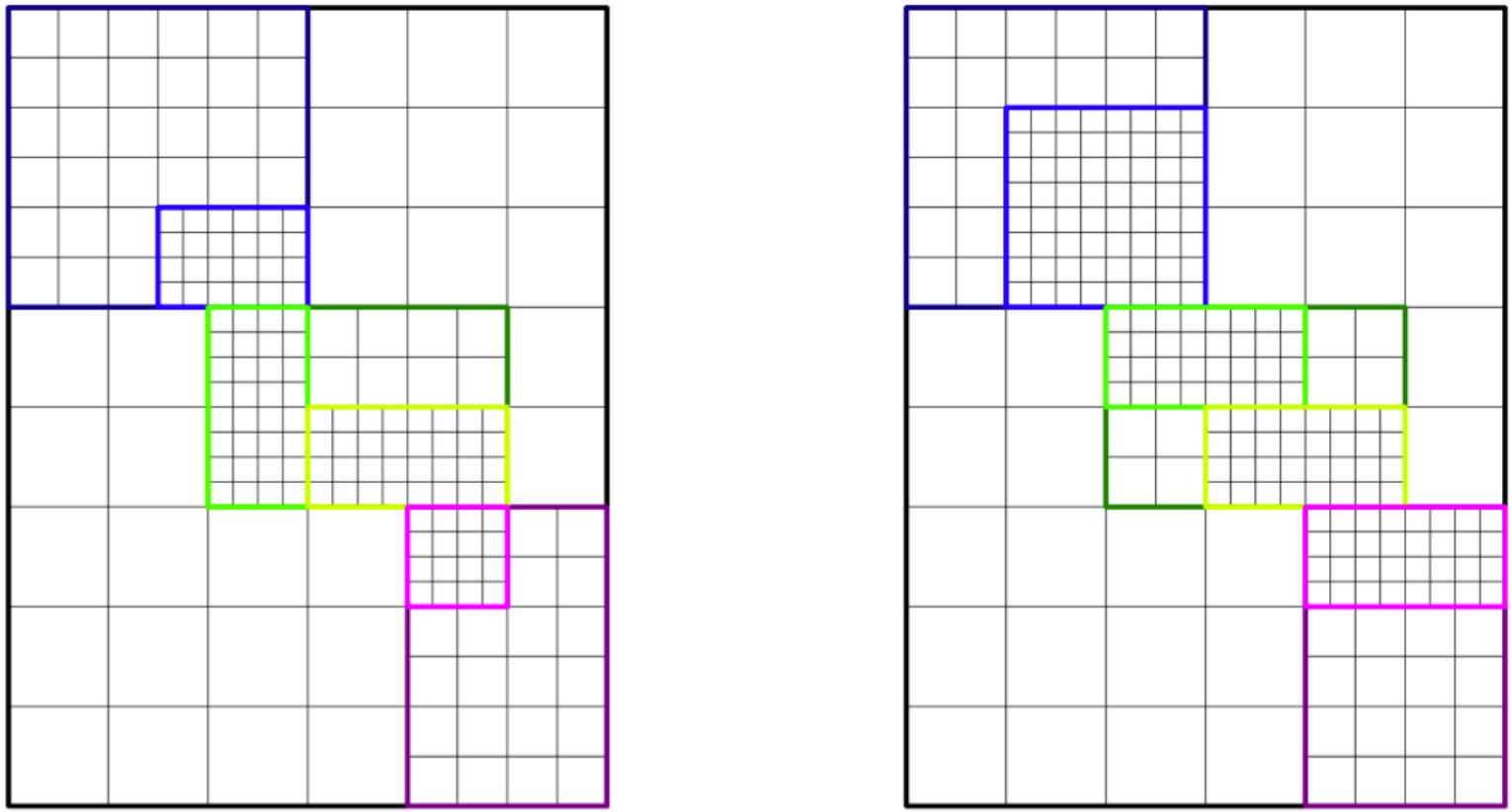
Level 2 nodes are children of level 1 nodes

The AMR tree contains all of the **parent-child** relationships between grids on different levels as well as **neighbor** relationships between grids on the same level.

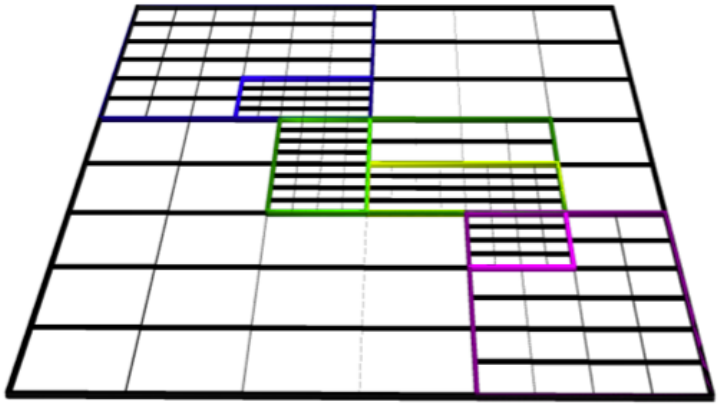
Level 2 grids nested within parent level 1 grids

AMR Tree

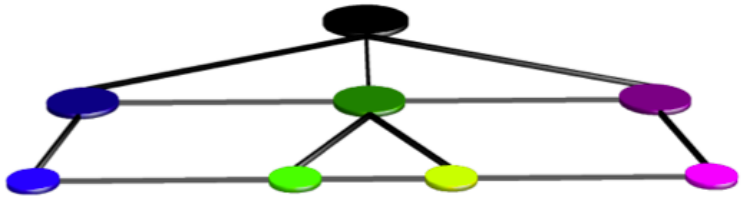
Since the mesh is adaptive there are successive iterations of grids and nodes



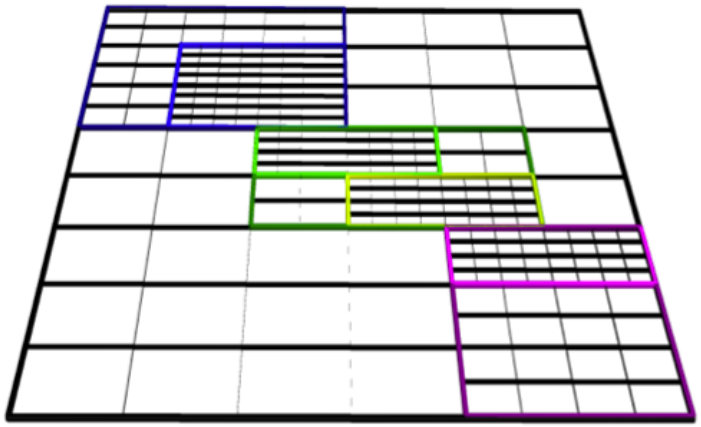
AMR Tree

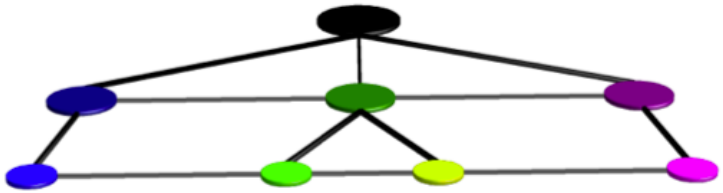


Current iteration



Current grids need access to data from the previous iteration of grids that physically overlap.

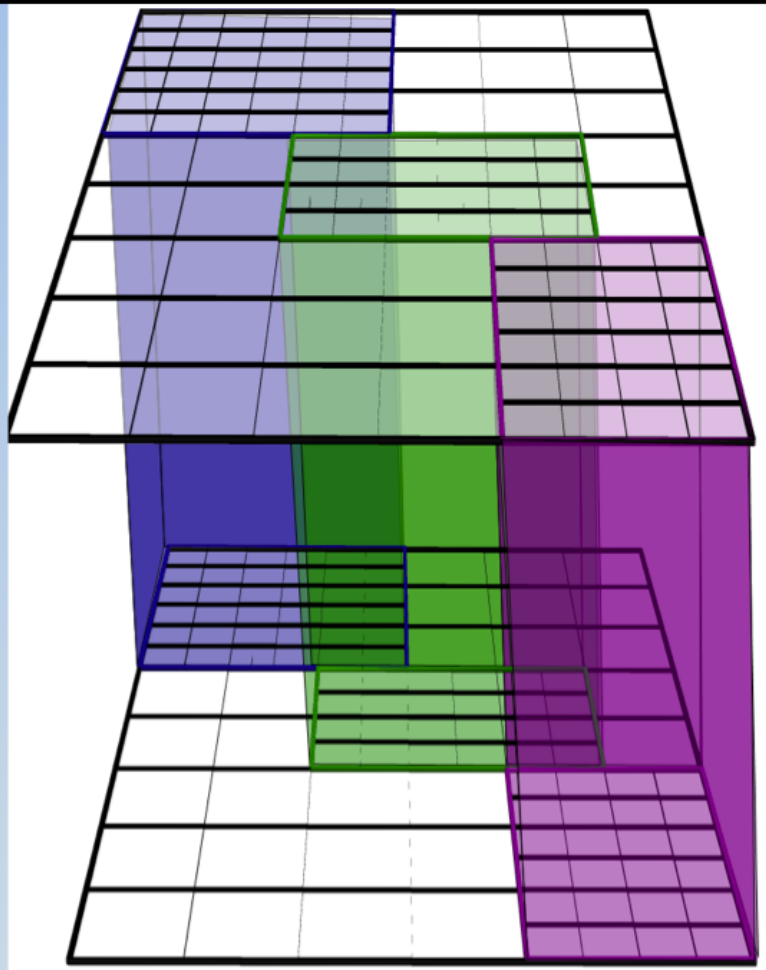




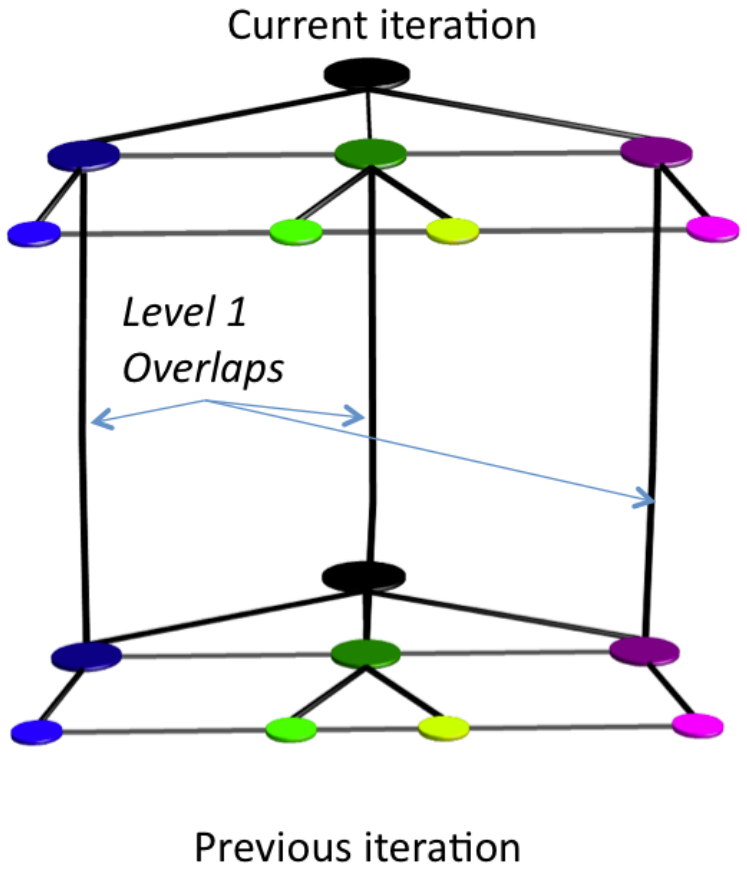
Previous iteration

Note that while the mesh has changed, the tree happens to have the same structure. This is only for simplicity and is not true in general.

AMR Tree

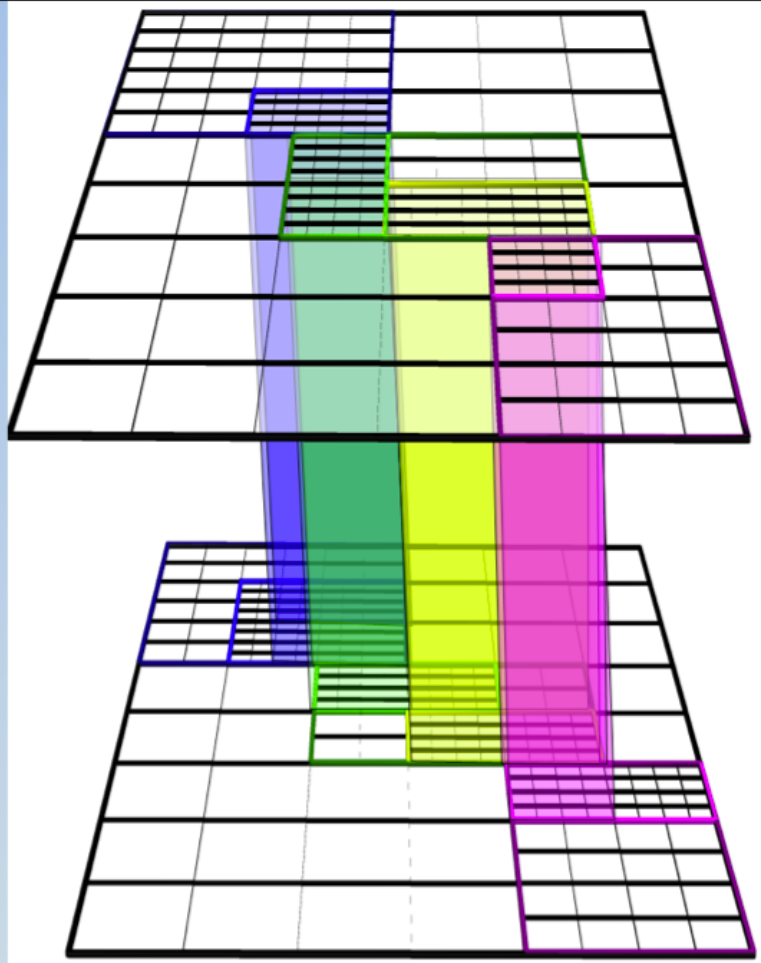


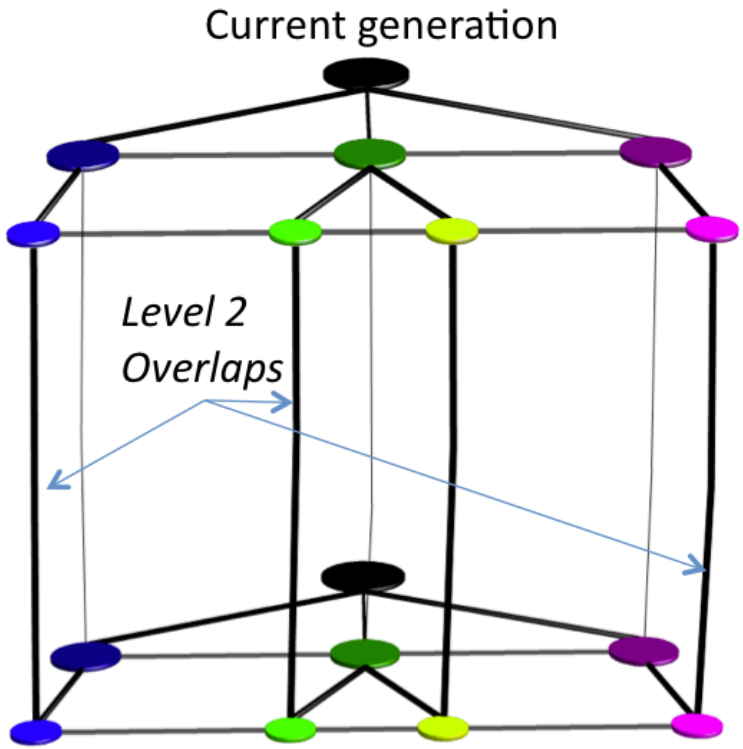
Note that the location of the level 1 grids has not changed between iterations for simplicity, but they still correspond to separate grids and nodes



This overlapping of grids between iterations is also stored in the AMR tree in **overlap** relationships.

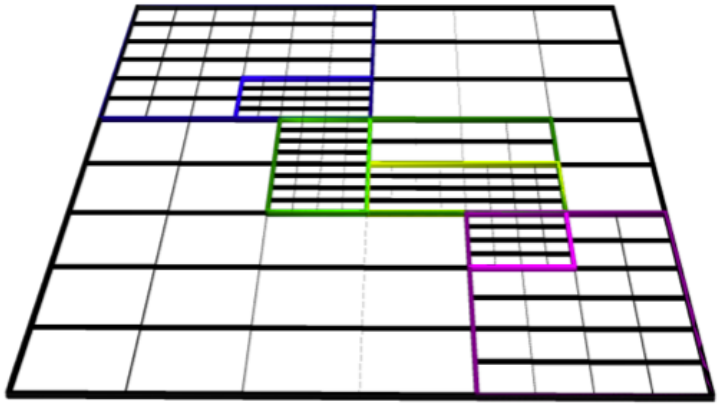
AMR Tree

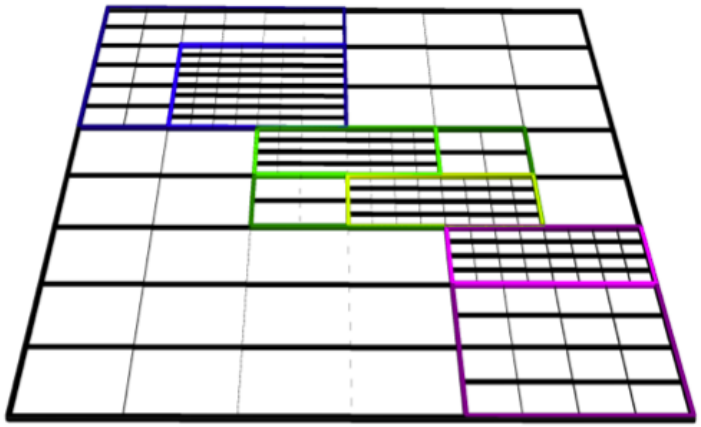


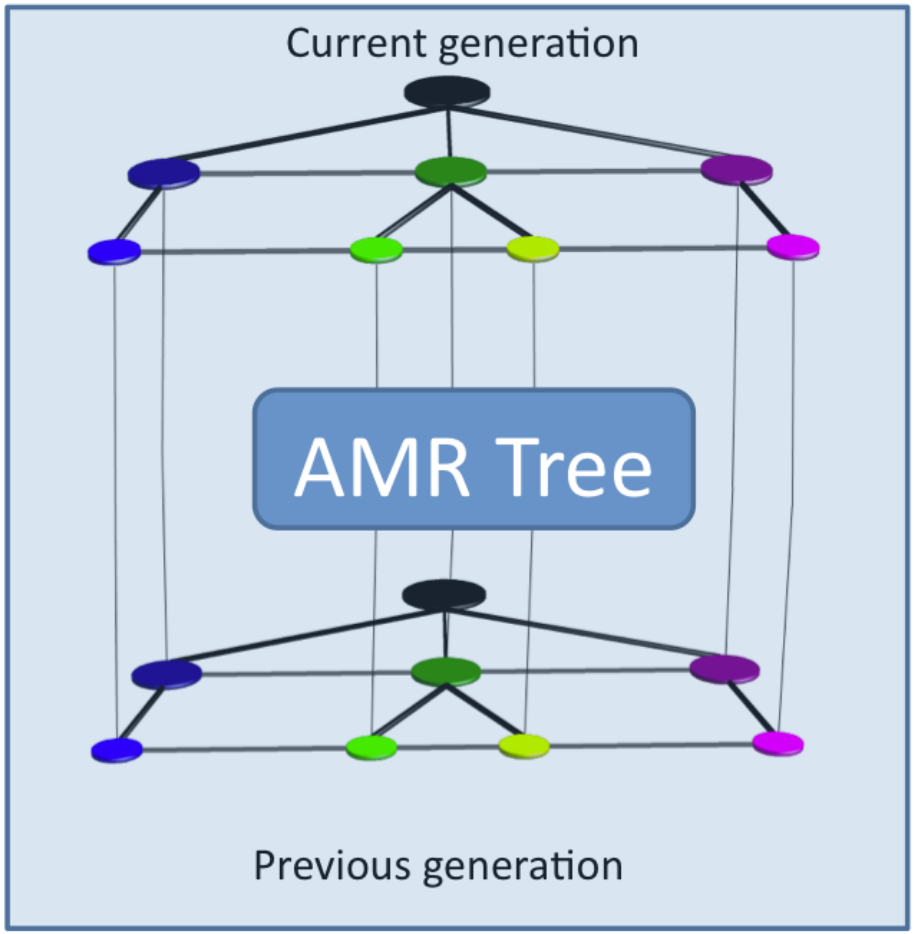


One of the useful features of nested AMR is that neighboring grids will either have the same parent or will have neighboring parents. The same is true for overlaps as well. This can be exploited for establishing neighbor and overlap connections for each iteration of nodes.

AMR Tree

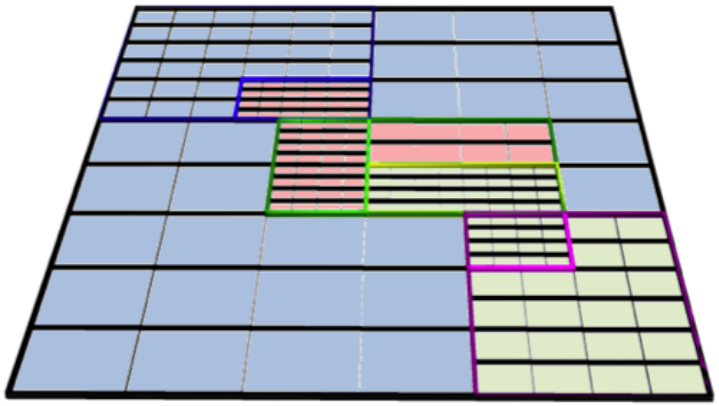


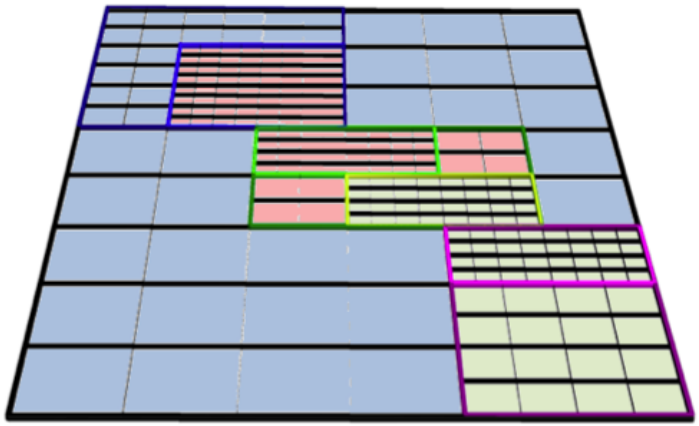




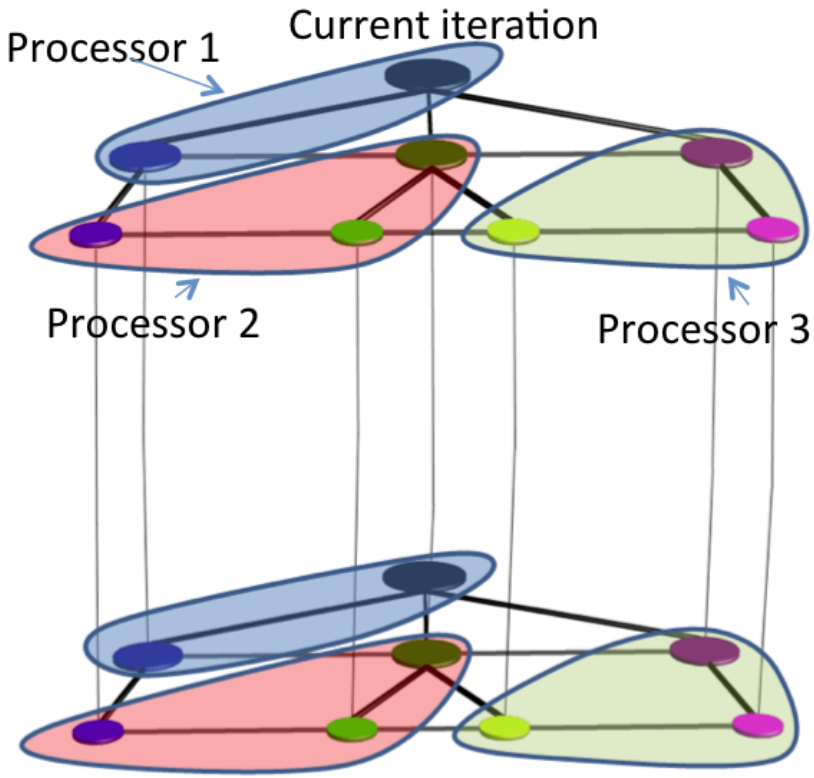
While all patch-based AMR codes distribute the grids that comprise the mesh and the associated computations across multiple processors, few distribute the tree.

Parallel & Distributed Tree



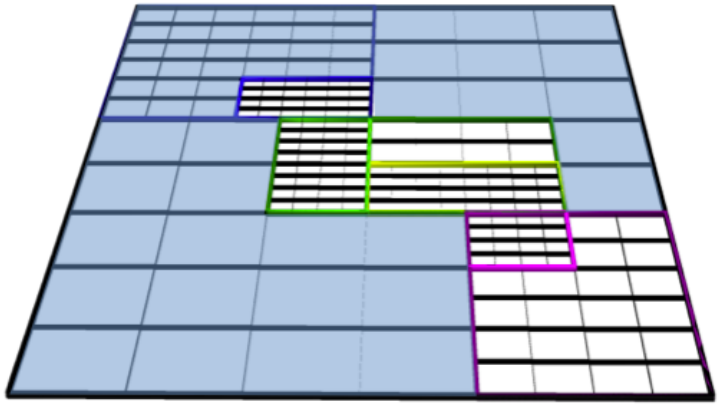


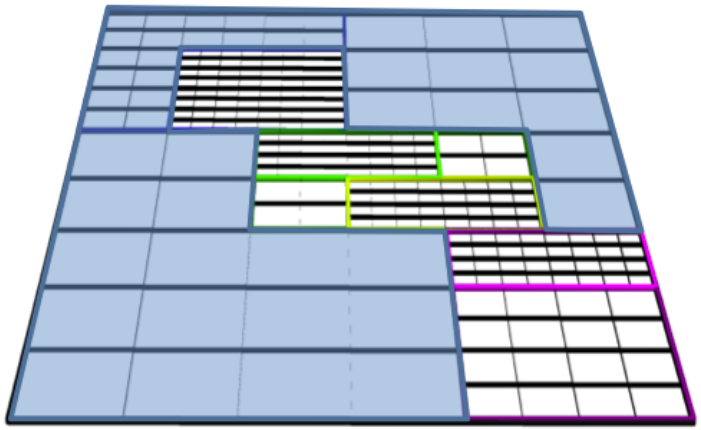
For simplicity we have also kept the distribution of grids similar between the two iterations

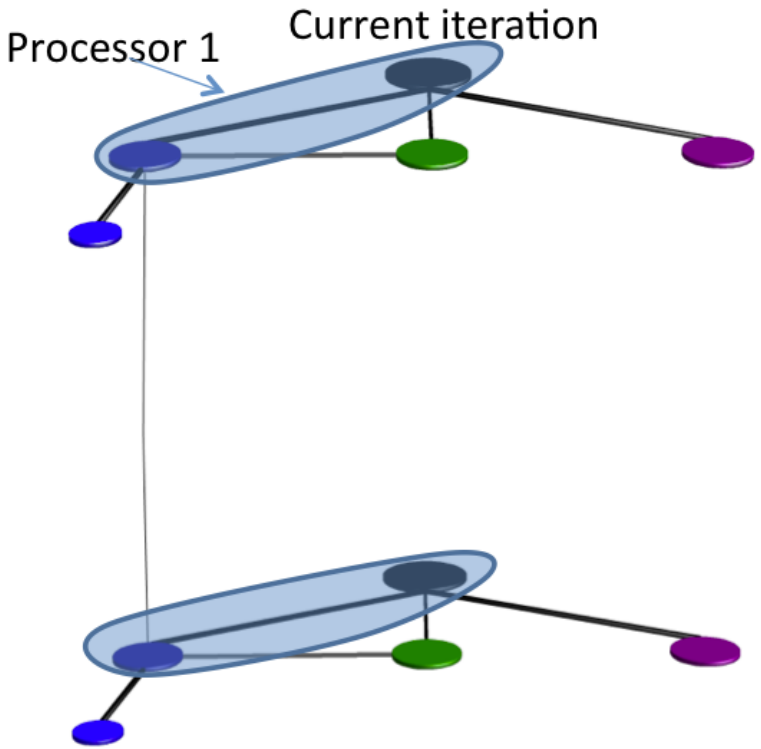


Consider this mesh distributed across 3 processors. Each processor has the data for a set of local grids and needs information about all of the nodes that connect with its local nodes.

Parallel & Distributed Tree

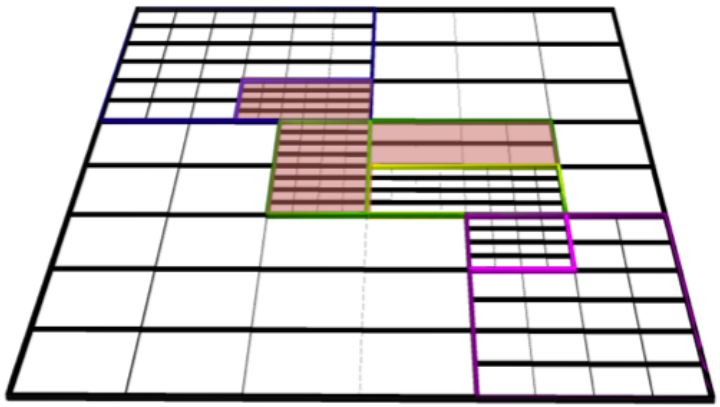


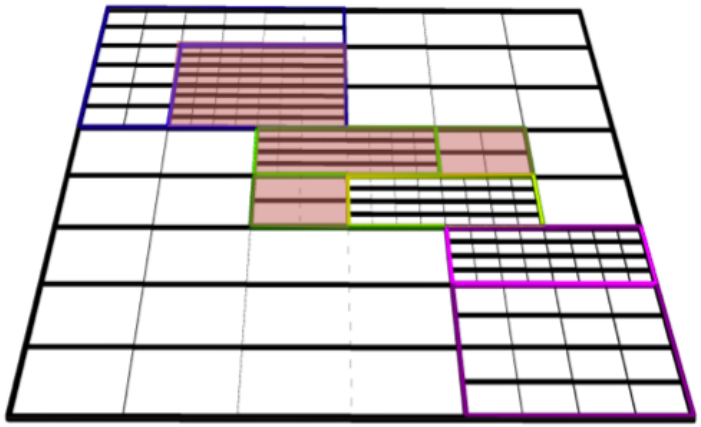




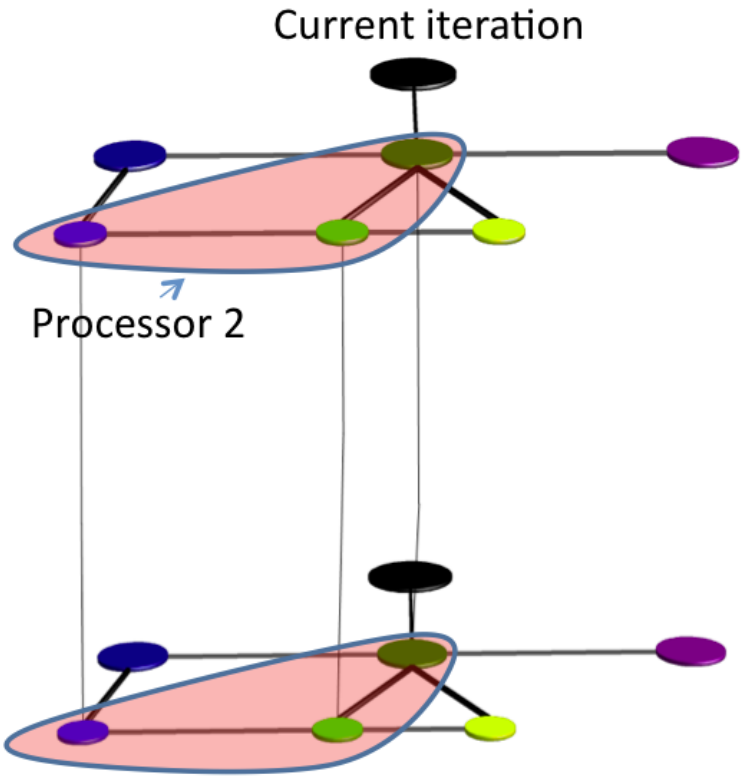
For example processor 1 only needs to know about the following section of the tree or “sub-tree”

Parallel & Distributed Tree

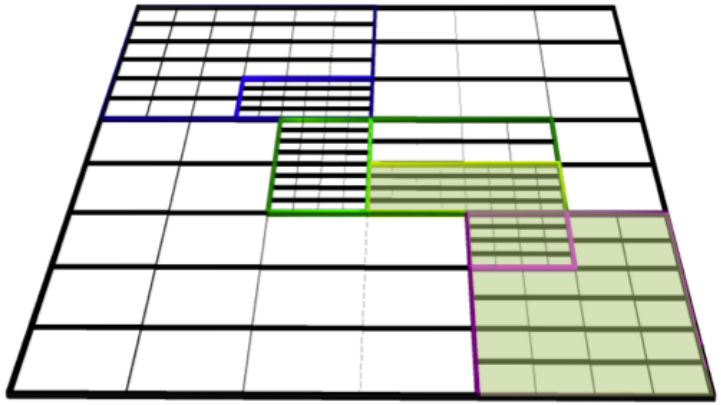


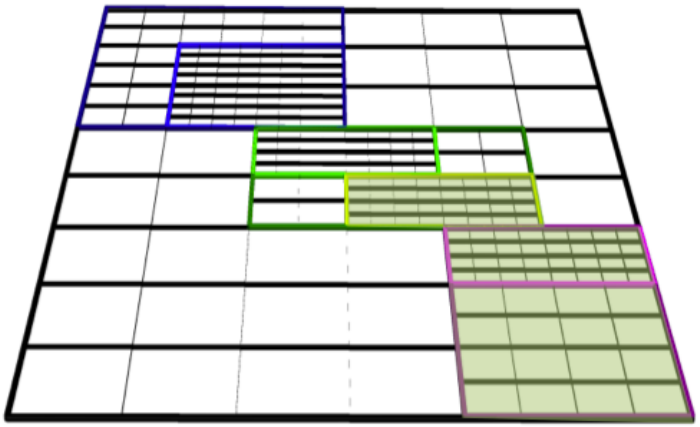


And the sub-tree for processor 2

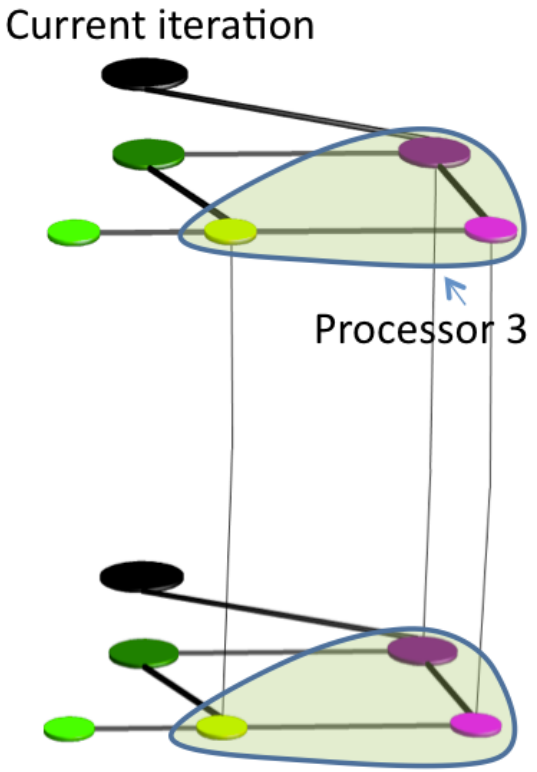


Parallel & Distributed Tree

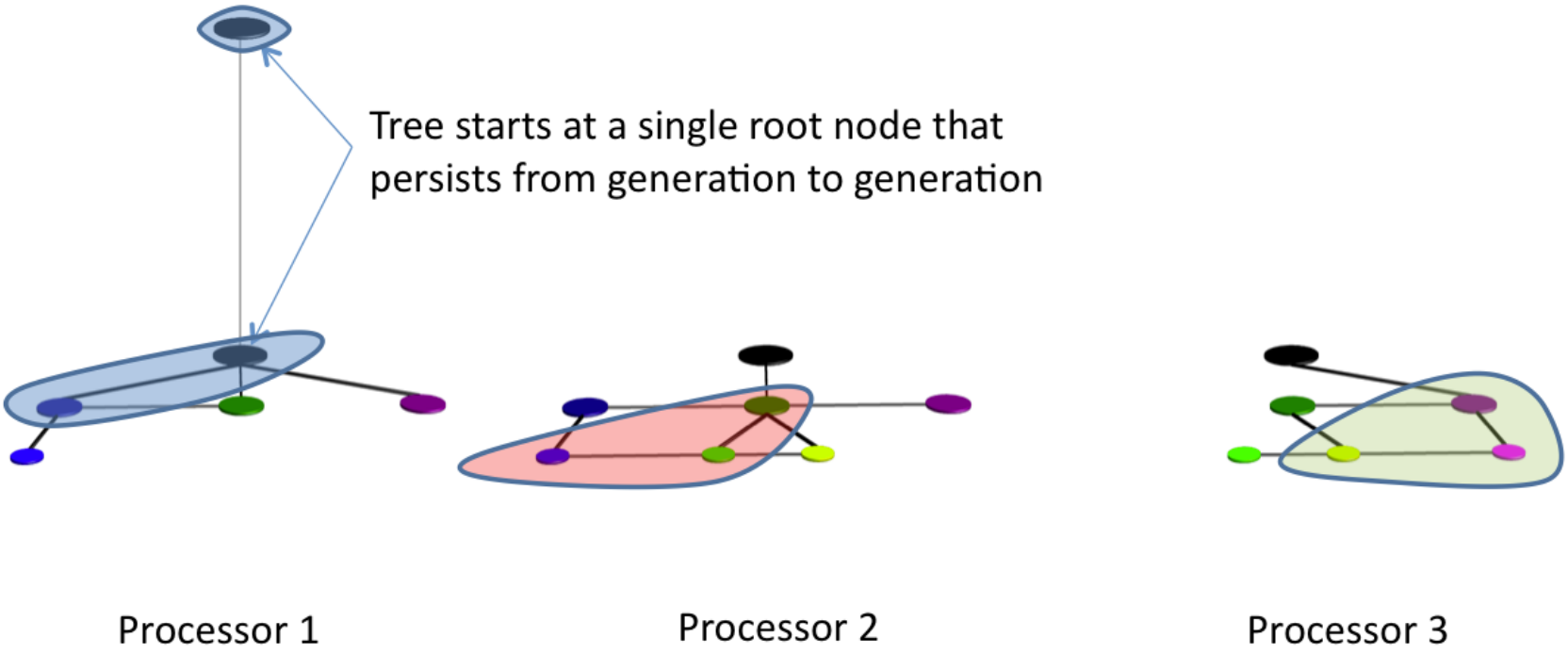




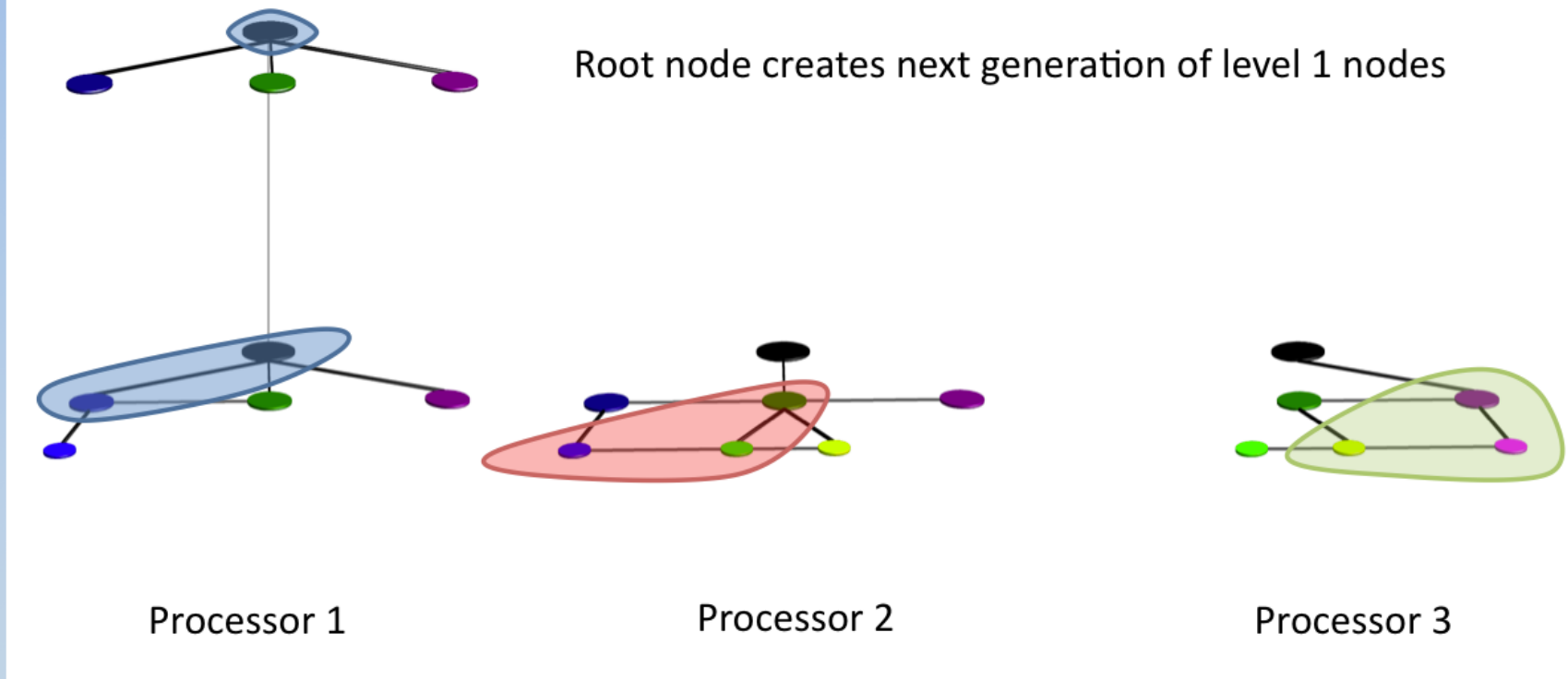
And the sub-tree for processor 3



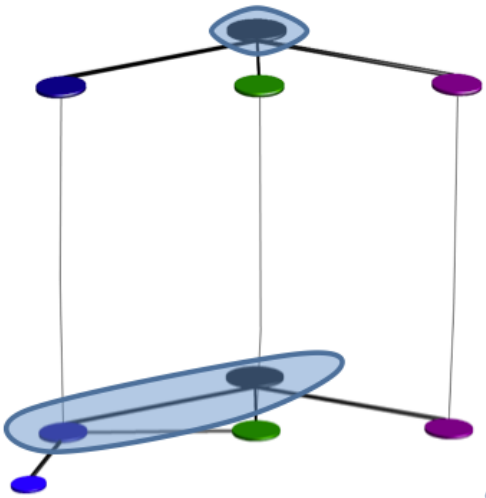
Parallel & Distributed Tree



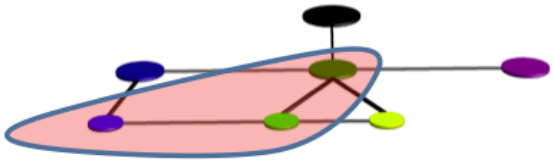
Parallel & Distributed Tree

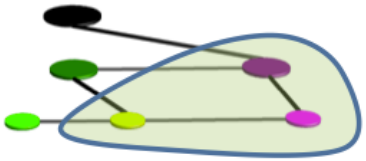


Parallel & Distributed Tree



Root node creates next generation of level 1 nodes
And exchanges child info with its overlap
It next identifies overlaps between new and old children



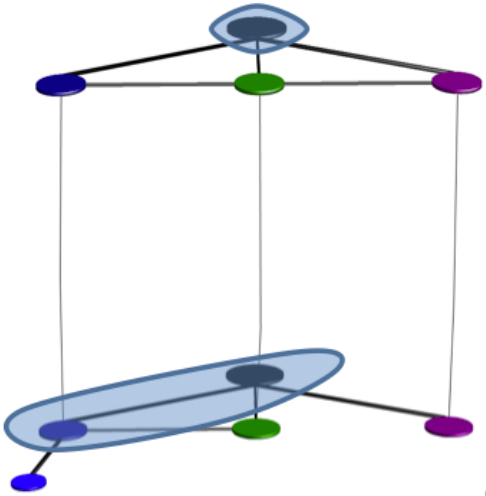


Processor 1

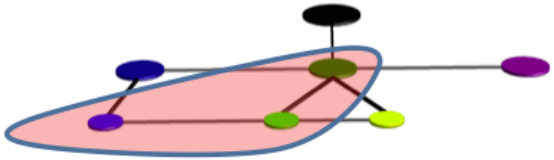
Processor 2

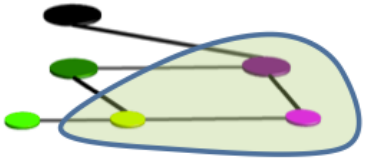
Processor 3

Parallel & Distributed Tree



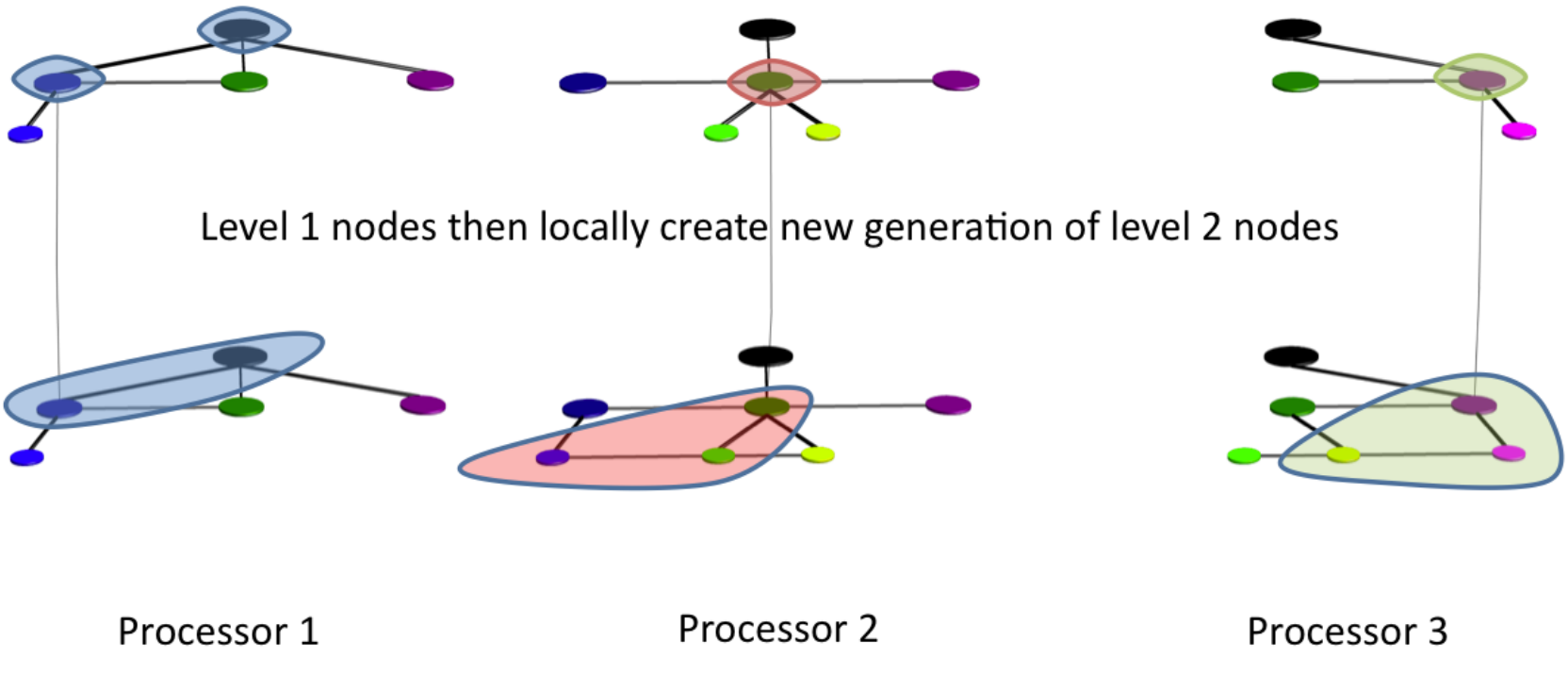
Root node creates next generation of level 1 nodes
And exchanges child info with its overlap
It next identifies overlaps between new and old children
As well as neighbors among new children



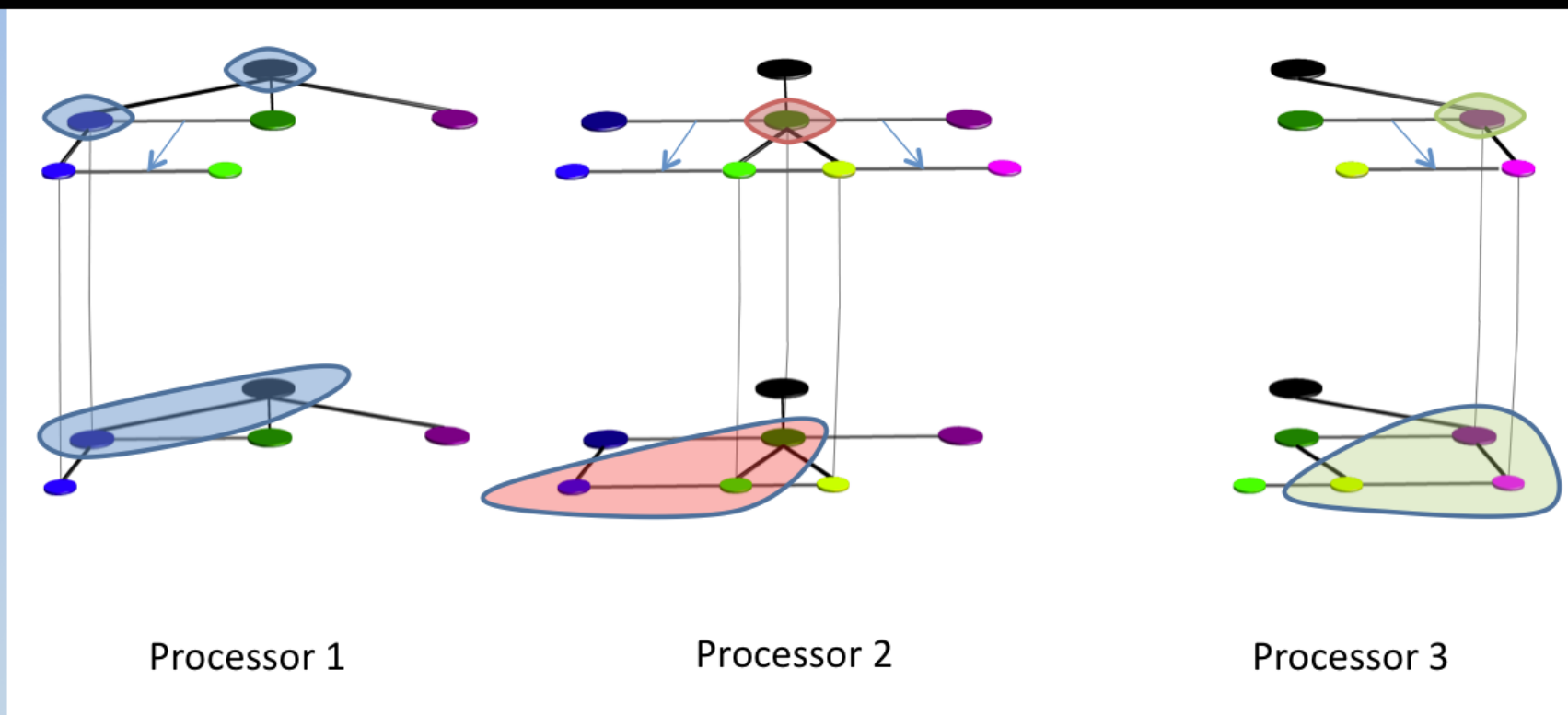


Processor 1 Processor 2 Processor 3

Parallel & Distributed Tree

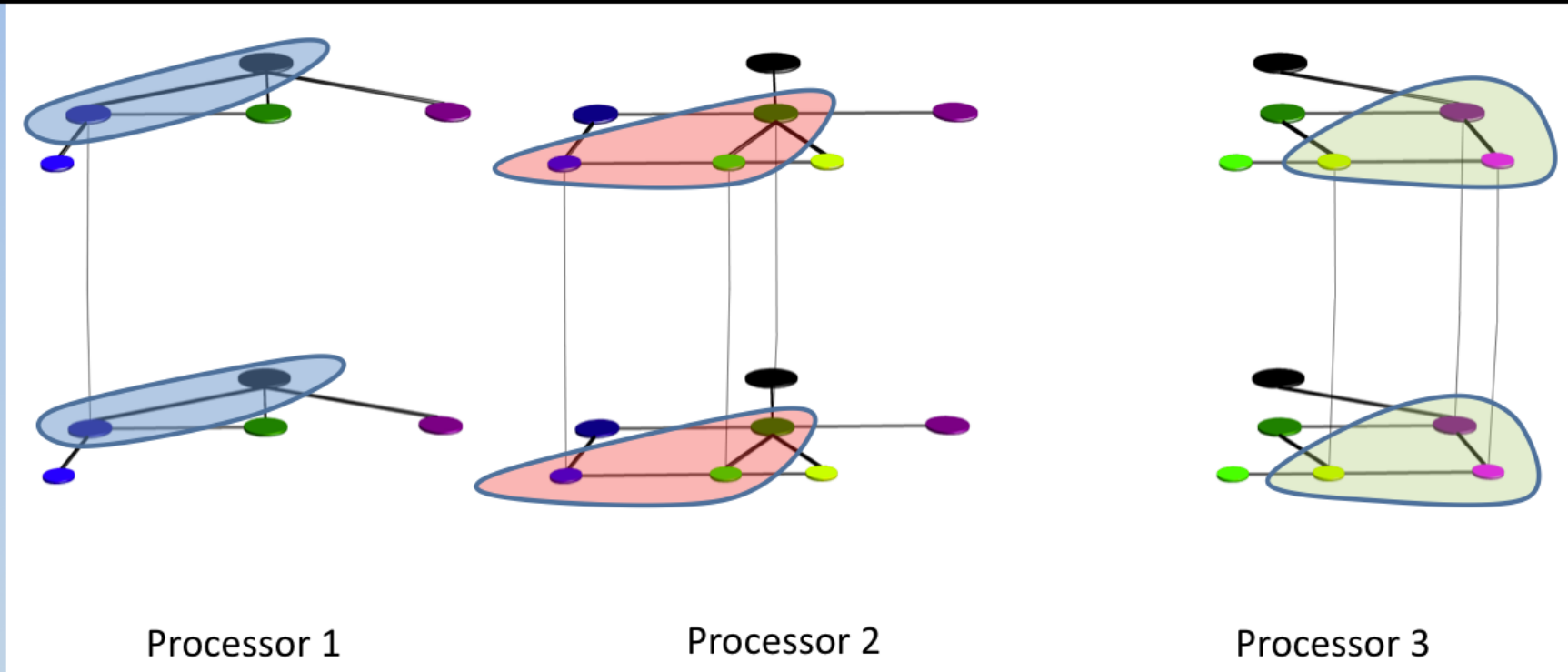


Parallel & Distributed Tree



Level 1 grids communicate new children to their neighbors so that new neighbor connections on level 2 can be determined

Parallel & Distributed Tree



Each processor now has the necessary tree information to update their local grids.

- Problem Module: problem.f90, data files

- Problem Module: problem.f90, data files
- Objects: Clumps, Jets, Winds, Ambient

- Problem Module: problem.f90, data files
- Objects: Clumps, Jets, Winds, Ambient
- HDF5
  - HDF5 is a data model, library, and file format for storing and managing data.
  - It supports an unlimited variety of datatypes, and is designed for flexible and efficient I/O and for high volume and complex data.
  - Can be analyzed with VisIt

- Problem Module: problem.f90, data files
- Objects: Clumps, Jets, Winds, Ambient
- HDF5
  - HDF5 is a data model, library, and file format for storing and managing data.
  - It supports an unlimited variety of datatypes, and is designed for flexible and efficient I/O and for high volume and complex data.
  - Can be analyzed with VisIt

Example Module

```
MODULE Problem
  USE GlobalDeclarations
  USE DataDeclarations
  USE Clumps
  USE Ambients
  USE Winds
  IMPLICIT NONE
  SAVE
  PUBLIC ProblemModuleInit, &
         ProblemGridInit, &
         ProblemBeforeStep, &
         ProblemAfterStep, &
         ProblemSetErrFlag, &
         ProblemBeforeGlobalStep
  PRIVATE
  REAL(KIND=qPREC) :: rho, radius, velocity

CONTAINS

  SUBROUTINE ProblemModuleInit()
    TYPE(AmbientDef), POINTER :: Ambient
    TYPE(ClumpDef), POINTER :: Clump
    TYPE(WindDef), POINTER :: Wind
    NAMELIST/ProblemData/ rho, radius
    OPEN(UNIT=PROBLEM_DATA_HANDLE, FILE= &
          'problem.data', STATUS="OLD")
    READ(PROBLEM_DATA_HANDLE,NML=ProblemData)
    CLOSE(PROBLEM_DATA_HANDLE)
    CALL CreateAmbient(Ambient)
    Ambient%density=rho
    Ambient%radius=radius
    CALL UpdateAmbient(Ambient)
    CALL CreateClump(Clump)
    Clump%density=rho
    Clump%radius=radius
    CALL UpdateClump(Clump)
    CALL CreateWind(Wind)
    Wind%velocity=velocity
    CALL UpdateWind(Wind)
  END SUBROUTINE
  SUBROUTINE ProblemGridInit(Info)
    TYPE(InfoDef) :: Info
  END SUBROUTINE

  SUBROUTINE ProblemBeforeStep(Info)
    TYPE(InfoDef) :: Info
  END SUBROUTINE ProblemBeforeStep

  SUBROUTINE ProblemAfterStep(Info)
    TYPE(InfoDef) :: Info
  END SUBROUTINE ProblemAfterStep

  SUBROUTINE ProblemSetErrFlag(Info)
    TYPE(InfoDef) :: Info
  END SUBROUTINE ProblemSetErrFlag

  SUBROUTINE ProblemBeforeGlobalStep(n)
     INTEGER :: n
  END SUBROUTINE ProblemBeforeGlobalStep

END MODULE
```

- MHD Equations
- More Tech Details
- Dynamic Load balance
- Performance of the code on Supercomputers
- Community Development:
  - Users & Developers
  - Code Management: Mecurial & Testing suite
  - Wiki & Tickets System
  - Youtube Channel: URAstroBEAR