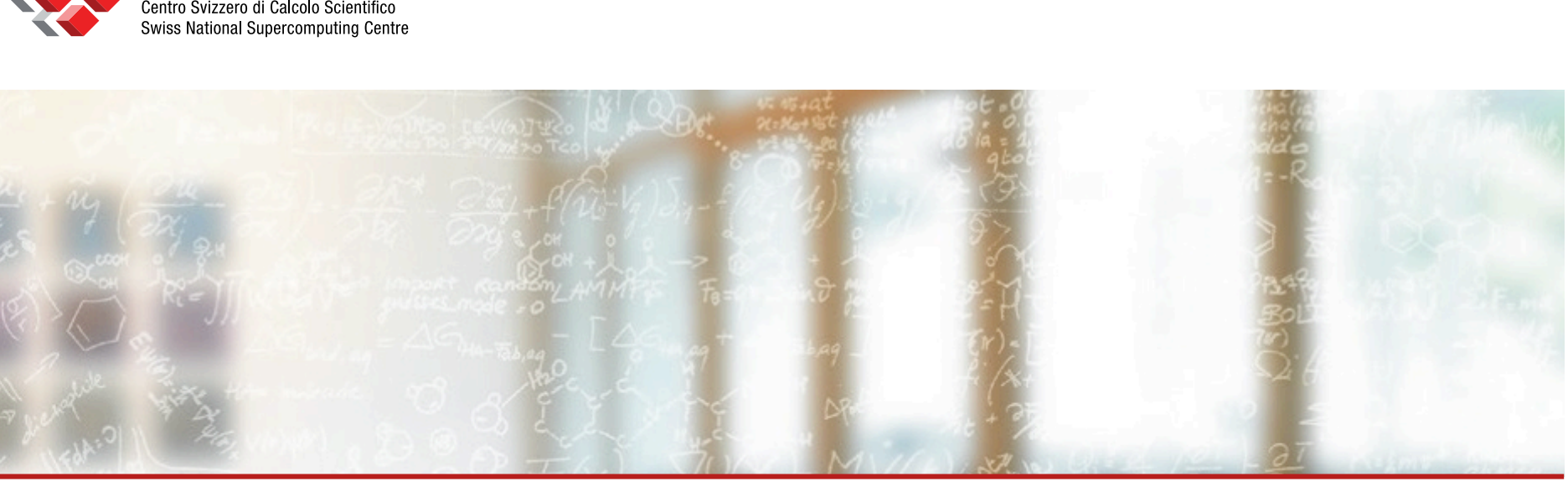
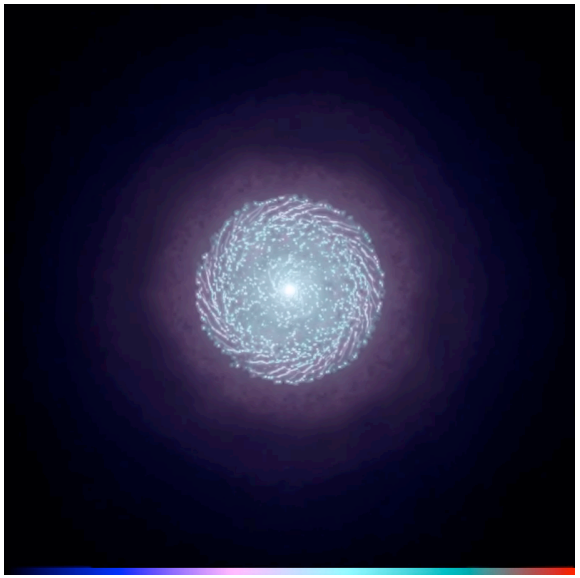


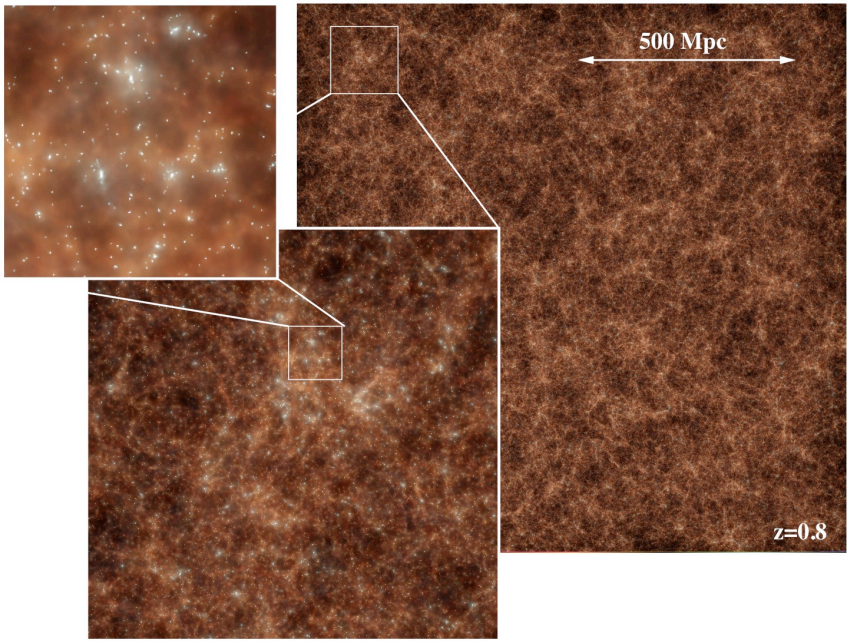
ENZO and RAMSES Codes for Computational Astrophysics

CFD & Astrophysics School
Claudio Gheller, CSCS, cgheller@cscs.ch
November 13-15, 2017



Introduction

- I will focus on a couple of codes addressing problems in **astrophysics** and **cosmology**
- These applications:
  - Need high **RESOLUTION** (space, mass), solving a broad variety of processes developing on very different scales
  - Need **ACCURACY** for properly solving complex physics
  - Need to properly treat **GRAVITATIONAL FORCES**



ENZO & RAMSES

- **3D MPI-parallel Eulerian Adaptive Mesh Refinement (AMR) codes.**
- **Similar codes in terms of applications and functionalities.**
- **Main difference: AMR approach and associated data structures**

Both codes solve:

- Dark Matter dynamics
- Gravity
- Baryonic Matter hydrodynamics
- MHD
- Radiative Transfer
- Many other physical processes…

RAMSES in some more details

(Teyssier, A&A, 385, 2002):

- **various components (dark energy, dark matter, baryonic matter, photons) treated**
- **Includes a variety of physical processes (gravity, MHD, chemistry, star formation, supernova and AGN feedback, etc.)**
- **Adaptive Mesh Refinement adopted to provide high spatial resolution ONLY where this is strictly necessary: Fully Threaded Tree**

- **Open Source**
- **Fortran 90**
- **Code "size": about 70000 lines**
- **MPI parallel (public version)**
- **OpenMP support (not really there…)**
- **OpenACC/CUDA under development**
- **How to get the code: https://bitbucket.org/rteyssie/ramses**

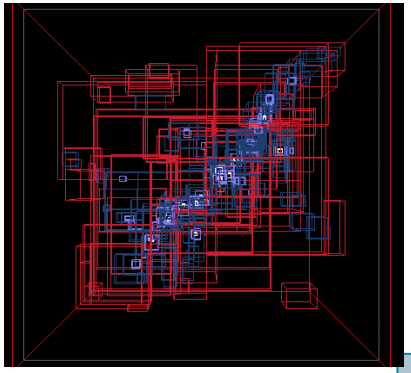
ENZO in some more details

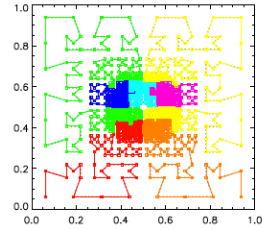
(Bryan et al. ApJS. 211, 2014):

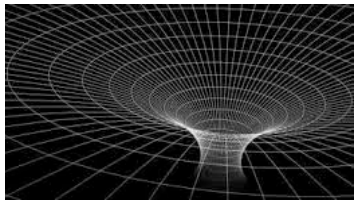
- **various components (dark energy, dark matter, baryonic matter, photons) treated**
- **Includes a variety of physical processes (gravity, MHD, chemistry, star formation, supernova and AGN feedback, etc.)**
- **Adaptive Mesh Refinement adopted to provide high spatial resolution using Structured AMR (SAMR).**

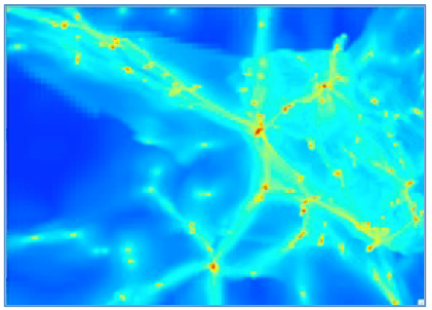
- **Open Source**
- **Fortran 90 + C++**
- **MPI parallel**
- **CUDA support for Hydro and MHD**
- **How to get the code: https://enzo.readthedocs.io/en/latest/**

What do they do?









AMR build

Load Balance

Gravity

Hydro

MHD

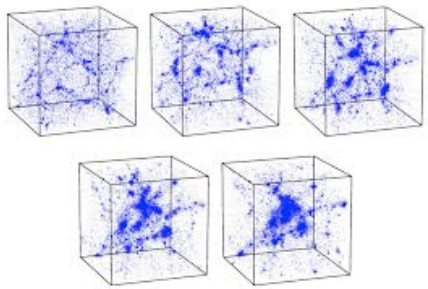
Time loop

More Physics

RT

Cooling

N-Body



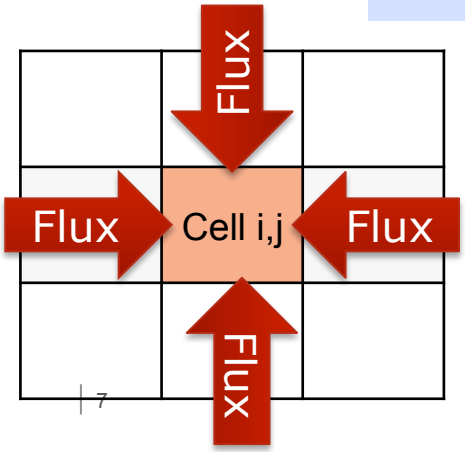
Solving fluid dynamics

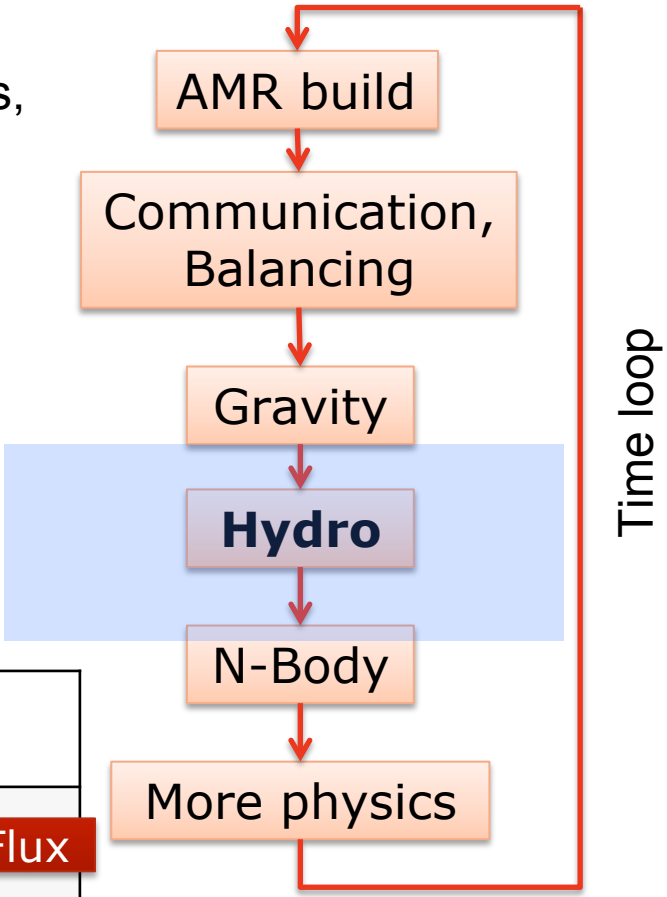
- Fluid dynamics is one of the key and most computational demanding kernels
- In both codes it is solved using a **finite volumes eulerian approach**: conservation equations of mass, momentum and energy are solved on a rectangular (**adaptive**) mesh. Equation of state of a perfect fluid closes the system
- **Shock capturing methods** are used
- Notice the **source (gravity) term**

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0$$

$$\frac{\partial}{\partial t}(\rho \mathbf{u}) + \nabla \cdot (\rho \mathbf{u} \otimes \mathbf{u}) + \nabla p = -\rho \nabla \phi$$

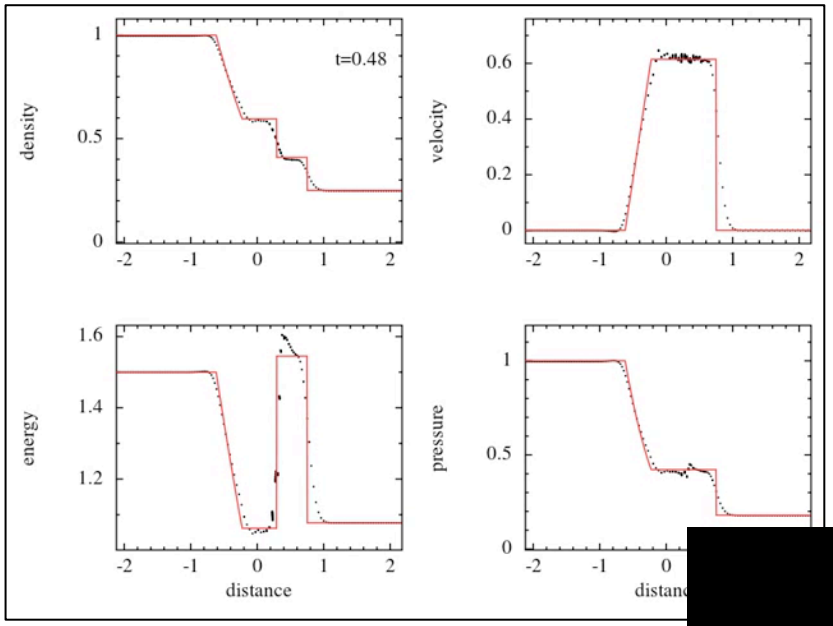
$$\frac{\partial}{\partial t}(\rho e) + \nabla \cdot [\rho \mathbf{u}\,(e + p/\rho)] = -\rho \mathbf{u} \cdot \nabla \phi$$

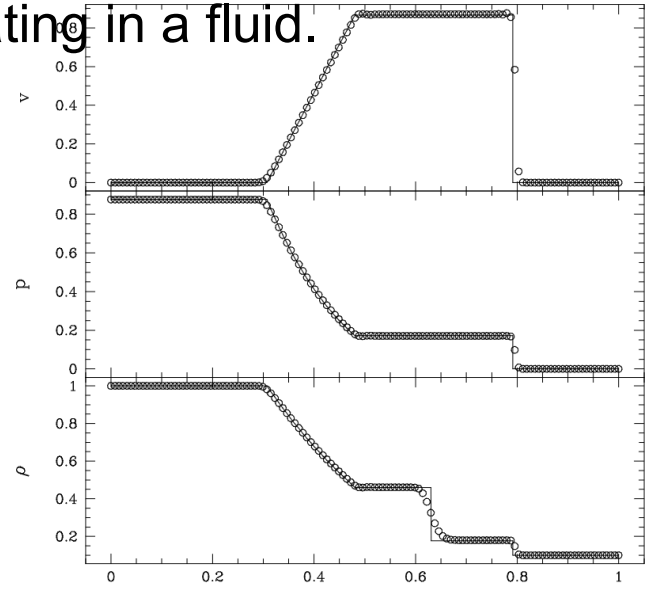


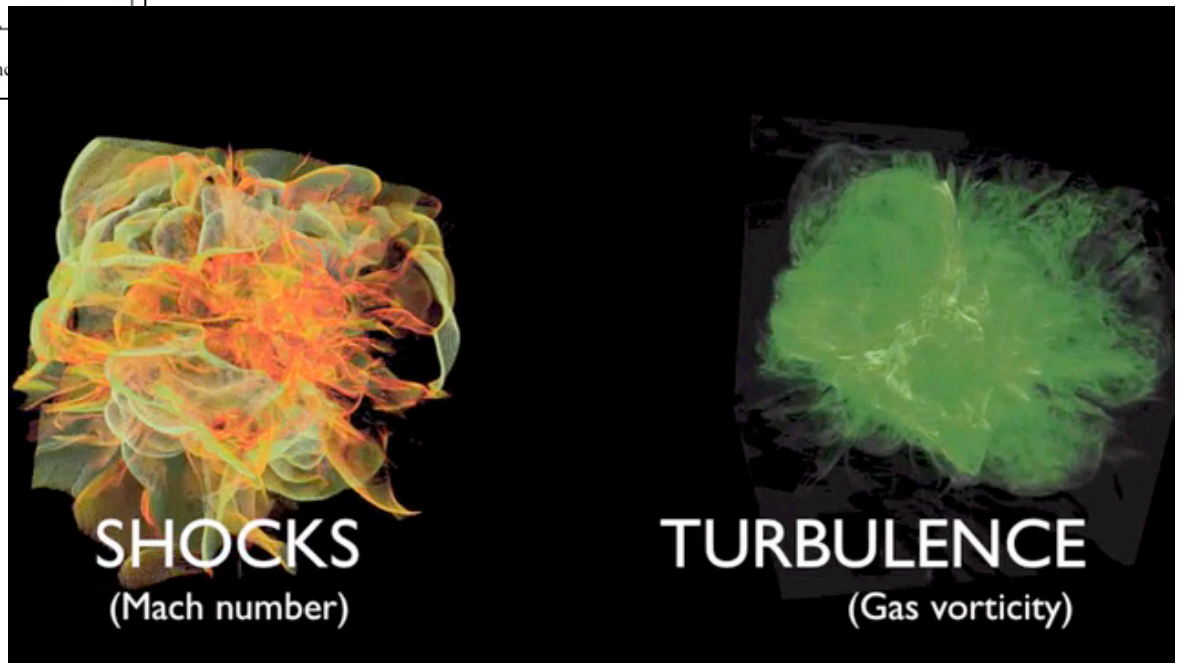


Shock Capturing Methods

Numerical methods specifically designed to accurately describe shock waves, i.e. physical discontinuities, propagating in a fluid.







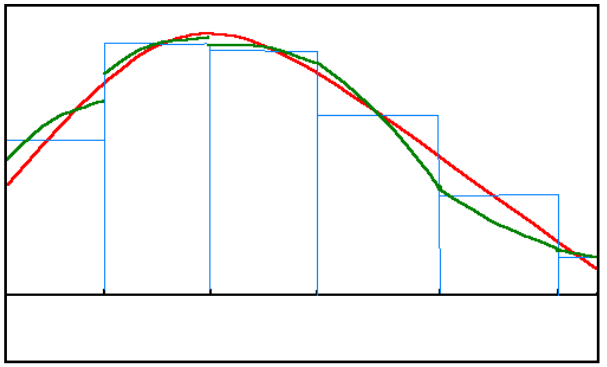
cscs

Shock Capturing Methods: Main Steps

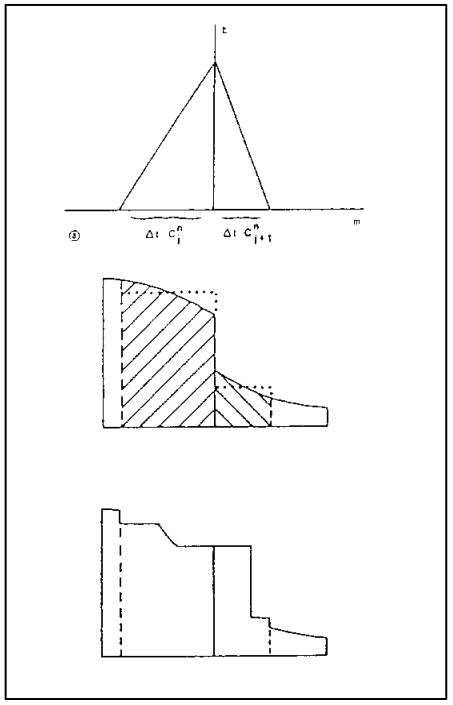
1. Formulate equations as finite difference

$$\varrho_{H,j}^{n+1} = \varrho_j^n + \frac{\Delta t^{n+1/2}}{a^{n+1/2}} \left(\frac{\bar{\varrho}_{j-1/2}\bar{v}_{j-1/2} - \bar{\varrho}_{j+1/2}\bar{v}_{j+1/2}}{\Delta x} \right)$$

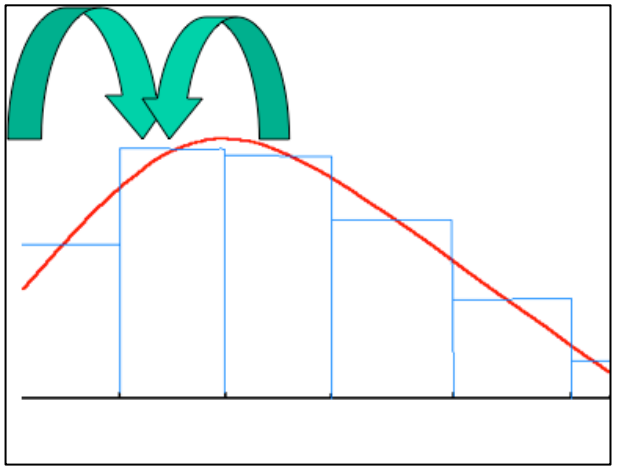
2. Reconstruct fluid profiles in the neighborhood of the cell to be updated (first order or more)



3. Calculate time centered fluxes at the cell boundaries (with different methods)



4. Use the fluxes to solve the equations



Dark Matter

- Non-collisional component
- Usually described as a set of particles
- Each particle is characterized by a mass, a 3D position and a 3D velocity
- Position and velocities are calculated solving the equations of motion:

$$dx/dt = v \; ; \quad dv/dt = -\nabla\Phi$$

- Leapfrog integration
- Density is calculated distributing the mass according to a proper kernel (more on this tomorrow).

Cosmological simulations

Gravity

Gravity has to be accurately calculated since it **drives the dynamics of the system**.

In principle it is a straightforward problem, following Newton's law of gravity (if relativistic effects can be neglected):

$$F = G\frac{m_1 m_2}{d^2}$$

However, gravity is highly **computationally challenging**:

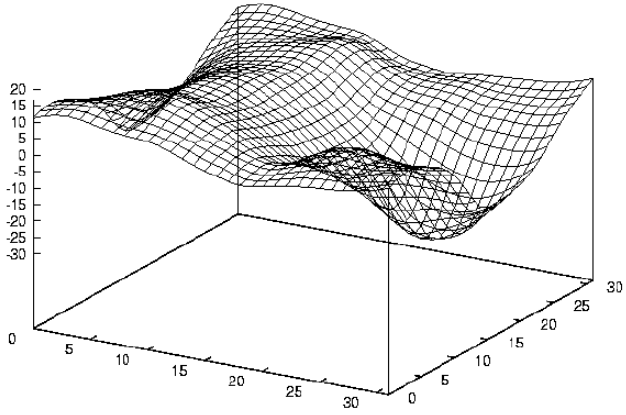
1. Newton's law scales as N^2
2. Gravity is a purely attractive long range force, involving for each point in space all the others

Calculating gravity

Many computational efficient methods have been implemented to calculate gravity.

ENZO and RAMSES solve the Poisson equation: $\nabla^2 \Phi = -4\pi G\rho$

On the computational mesh.



They both combine a FFT based method:

$$\int_{-\infty}^{+\infty} \frac{\partial^2 \Phi}{\partial x^2} e^{ikx} dx = \int_{-\infty}^{+\infty} A\varrho(x) e^{ikx} dx = A\varrho(k) \quad \Rightarrow \quad \Phi(k) = A\frac{\varrho(k)}{k^2}$$

with a Multigrid algorithm:

relaxation iterative method + multiple levels

$$u^{n+1}_{(i,j,k)} = (1/6)(u_{(i+1,j,k)} + u_{(i-1,j,k)} + u_{(i,j+1,k)} + u_{(i,j-1,k)} + u_{(i,j,k+1)} + u_{(i,j,k-1)}) - (dx^2/6)F_{(i,j,k)}$$

+

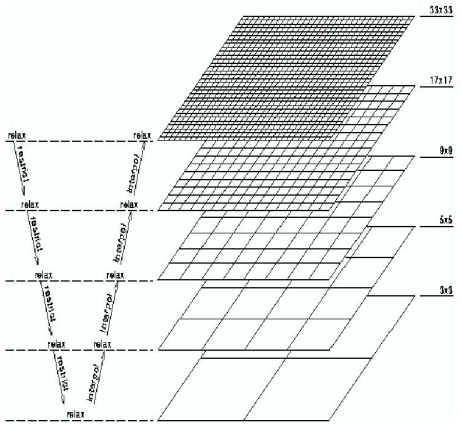
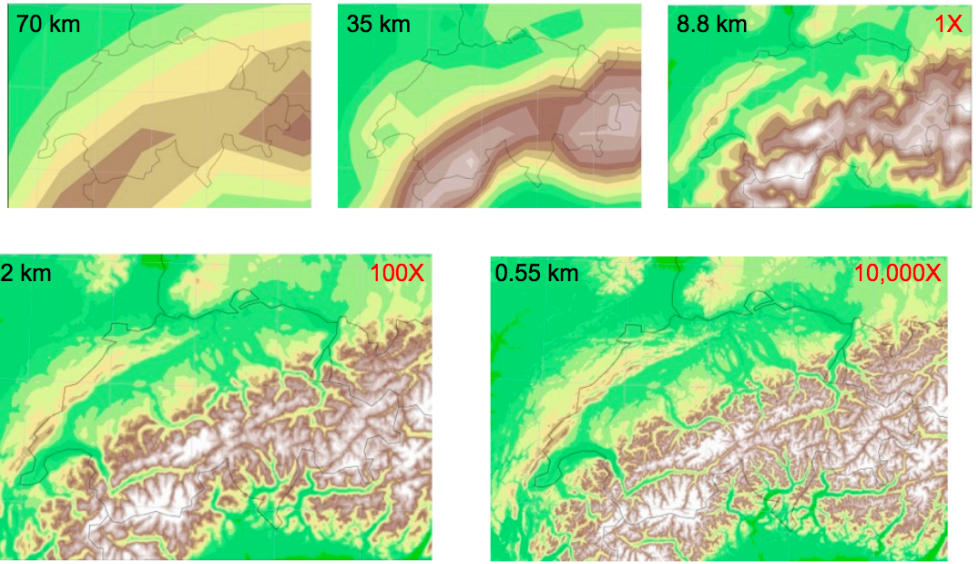


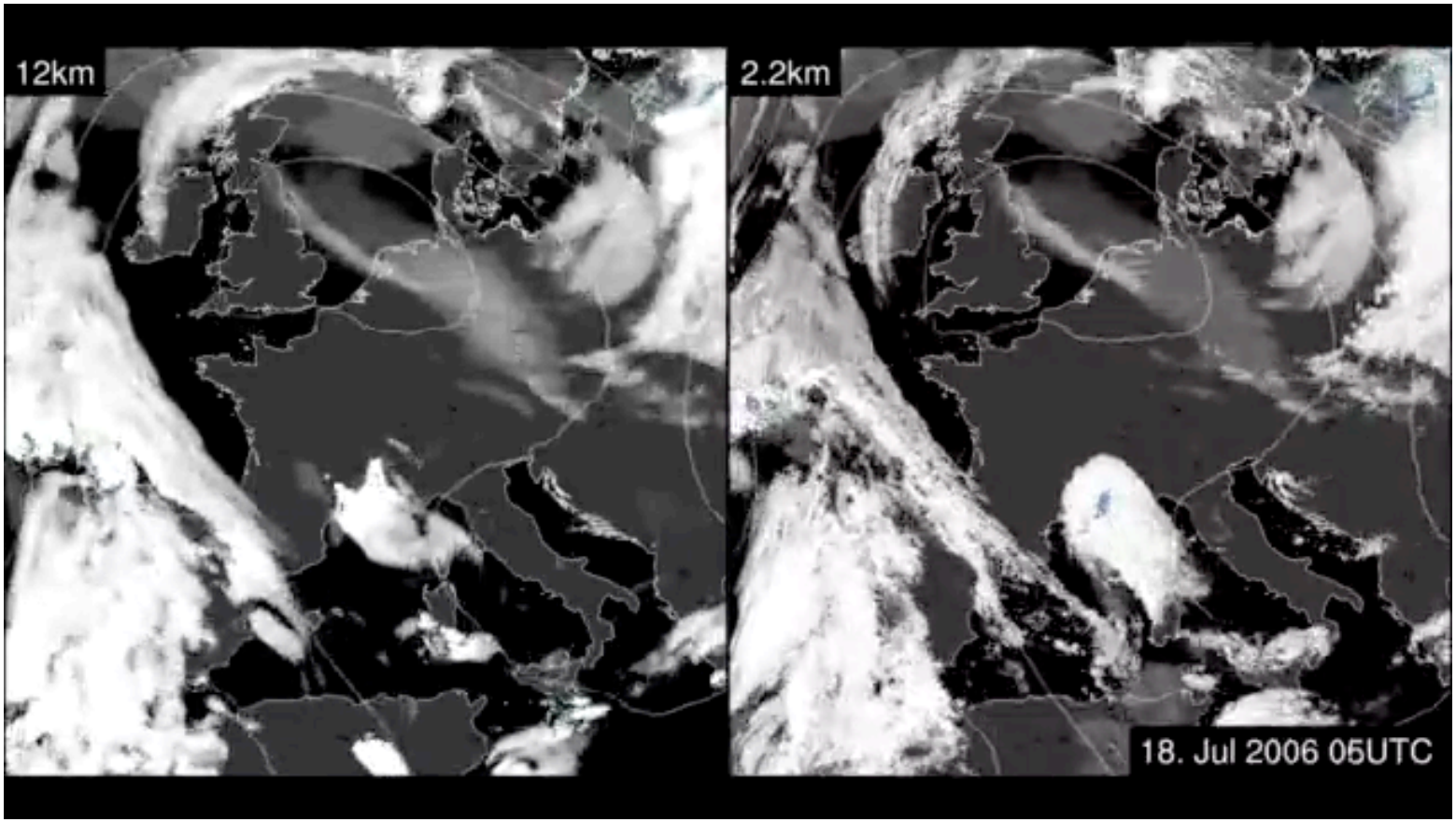
Figura 2.1: Struttura standard dello schema correttivo V-cycle.

The mesh

- So far we have assumed the existence of a computational mesh, on which our equations and quantities are discretized, focusing on the most appropriate numerical method.
- The mesh **defines most of the data structures, so the memory layout and usage**
- We have not addressed yet the **need for resolution**
- With a mesh, the resolution is set by the cell size
- Brute force, one can think of using a super-fine mesh: of course this is computationally impossible (computational needs scale with the CUBE of the resolution)
- **But for many physical processes, having high resolution everywhere is NOT necessary**
- This is particularly true in astrophysics, due to the action of gravity, which tends to cluster evolving objects.

Resolution: weather forecast examples





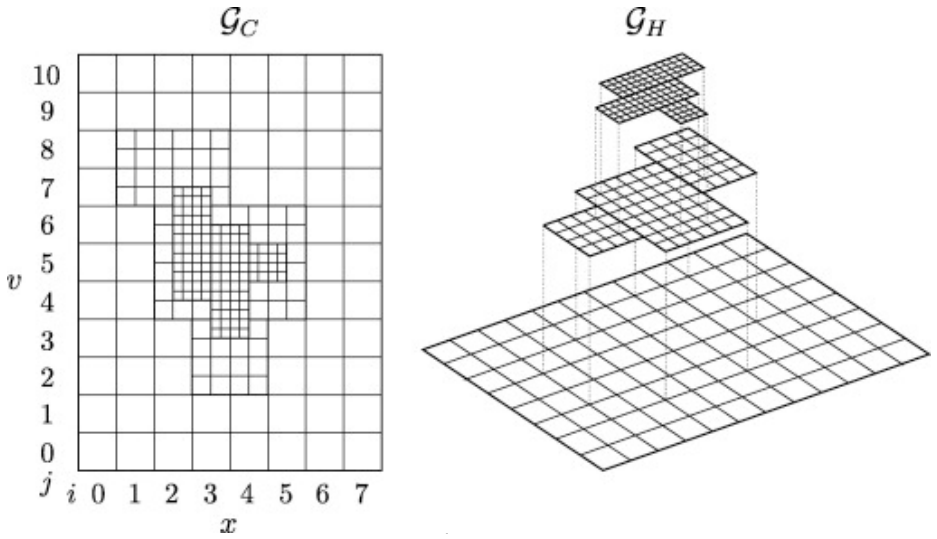
ETH zürich

Adaptive Mesh Refinement (AMR)

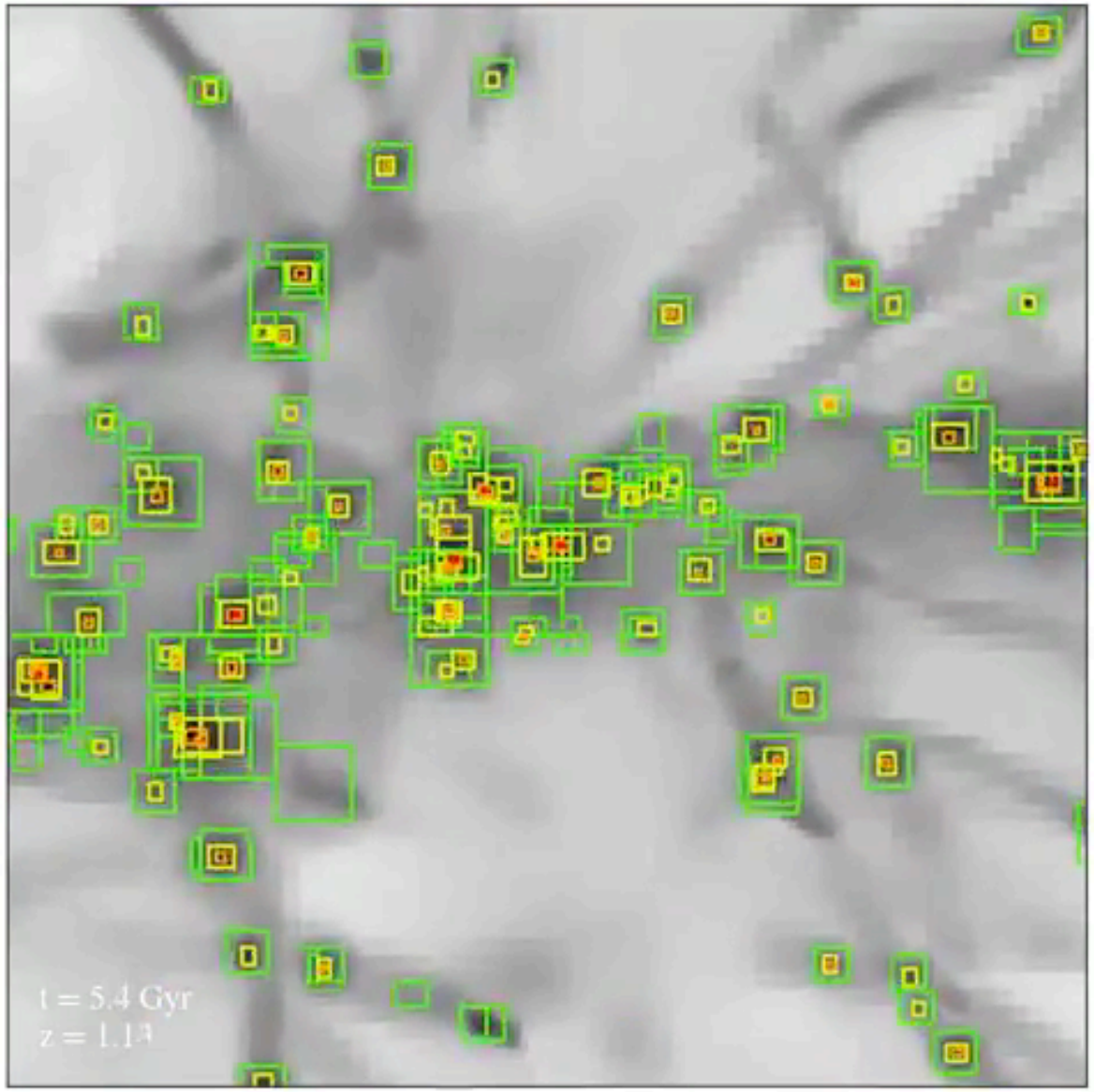
- AMR is a class of meshing algorithms designed to **increase the resolution only where this is necessary**
- Geometry is preserved
- Different solutions can be adopted:
  - **RAMSES → Fully Threaded Tree**
  - **ENZO → Structured AMR**
- Both approaches have advantages and drawbacks. However they dramatically reduce the demand of computational resources

Structured AMR (ENZO)

- Refinement is done on a **“patch”** basis
- Main steps:
  1. Flag cells that have to be refined according to given criteria
  2. Cluster neighboring cells
  3. Create rectangular patches filling the gaps according to a given “efficiency” criterion and “properly nested”
  4. Initialize the variables in the patches
  5. Update the “control tree” with the newly created patches

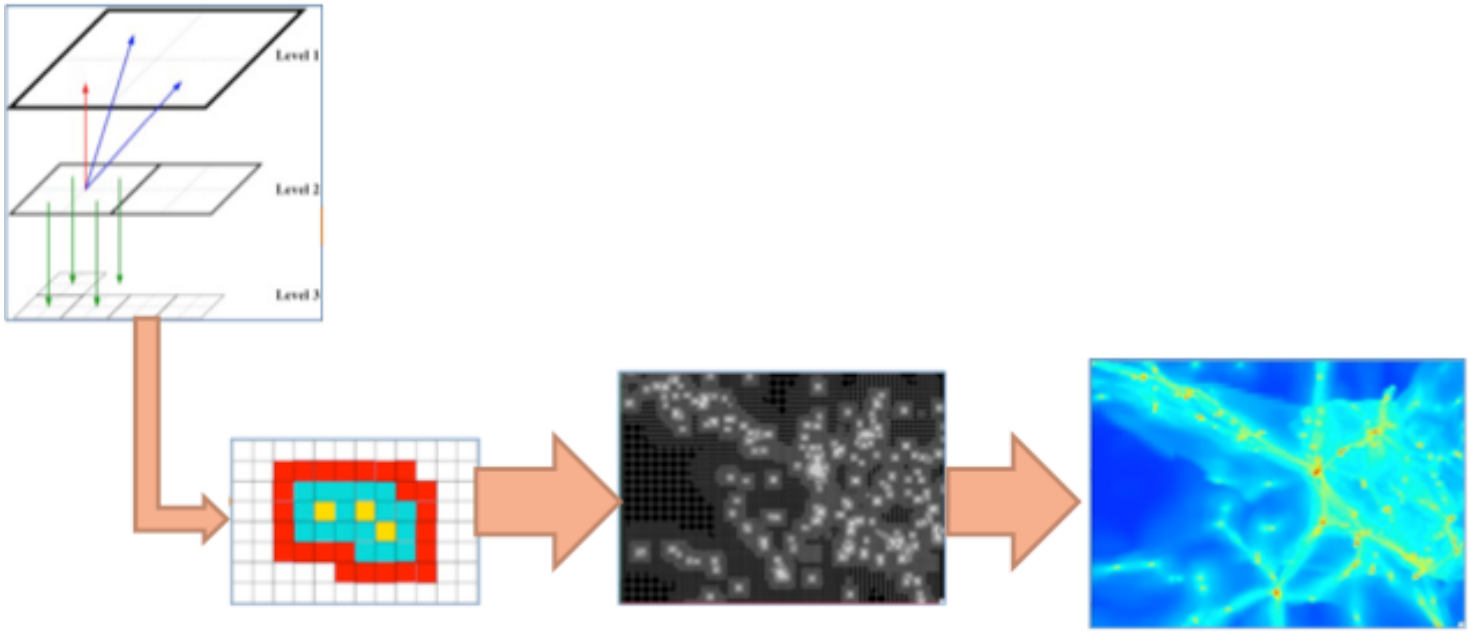


SAMR in action

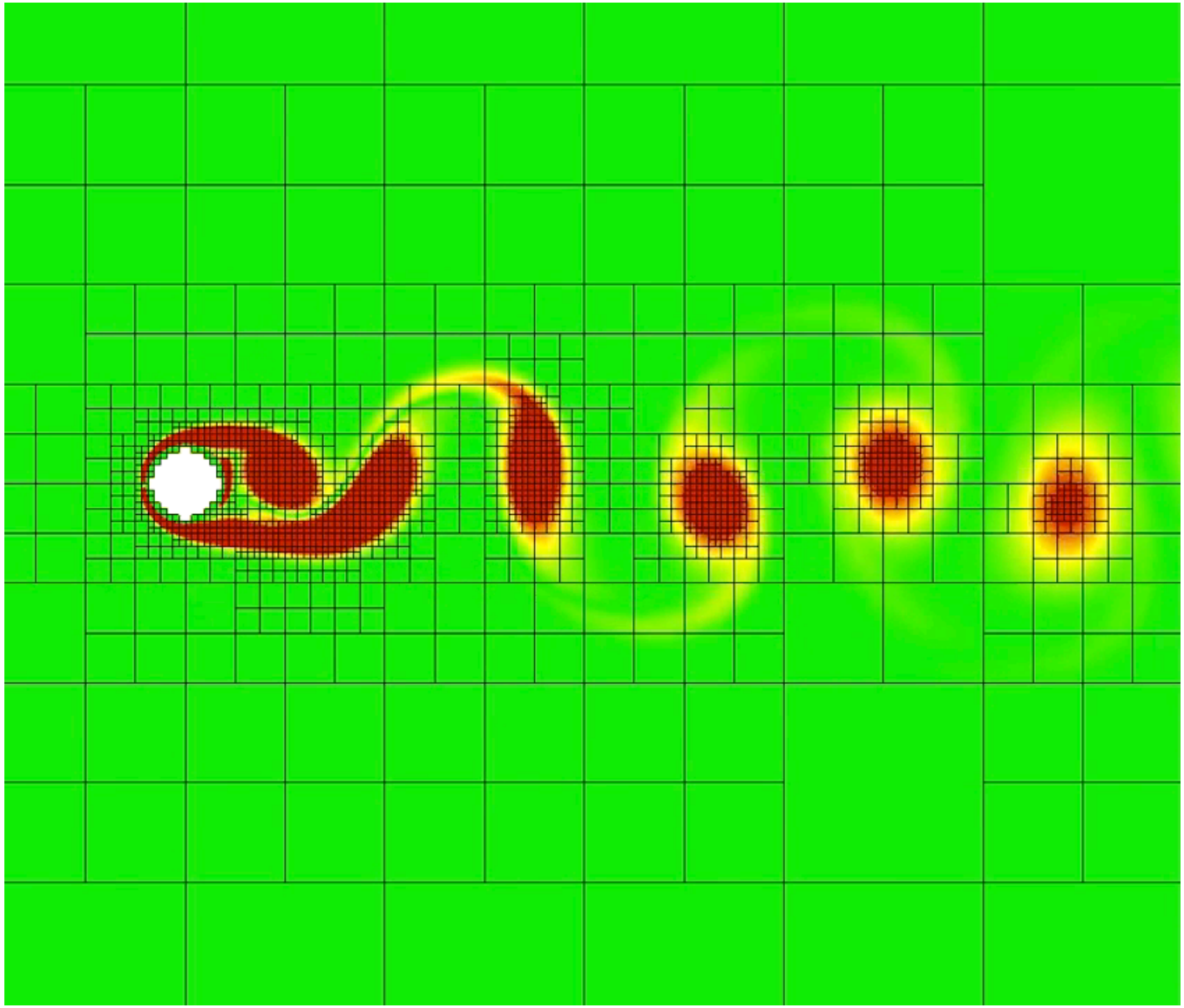


Fully Threaded Tree (RAMSES)

- Refinement is done on a **“per-cell”** basis
- Main steps:
  1. Flag cells that have to be refined according to given criteria
  2. Divide each flagged cell in 8 equal parts (OCTS)
  3. Initialize variables in the Oct’s cell
  4. **Define the pointers to parents, siblings and Oct’s cells**



Fully Threaded Tree in action



SAMR vs Fully Threaded Tree: what's the best?

- **SAMR:**
  - \+ Simple geometry
  - \+ Optimal layout in memory, cache friendly
  - − Refinement of unnecessary cells
  - − Need for a patch manager (two linked lists) which can fill memory

- **Fully Threaded Tree:**
  - \+ Optimal allocation of memory
  - \+ All information stored in the Octs (no need of a tree)
  - − Not so efficient memory layout
  - − Complex geometry

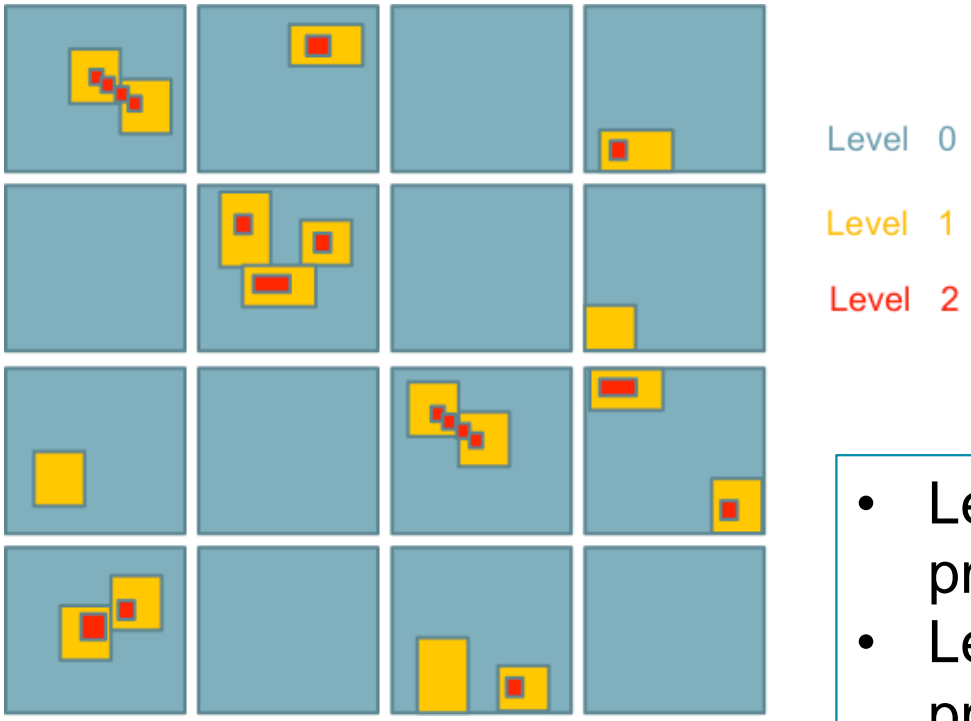
AMR: is it actually so memory-effective?

- This example is for RAMSES, but similar results can be obtained for ENZO:

```
Mesh structure
Level  1 has       1 grids (     0,     1,     0,)
Level  2 has       8 grids (     0,     1,     0,)
Level  3 has      64 grids (     0,     1,     0,)
Level  4 has     512 grids (     1,     3,     2,)
Level  5 has    4096 grids (    10,    19,    16,)
Level  6 has   32768 grids (    80,   149,   128,)
Level  7 has  262144 grids (   648,  1190,  1024,)
Level  8 has 2097152 grids (  5192,  9526,  8192,)
Level  9 has 16777216 grids ( 41541, 76208, 65536,)
Level 10 has 2608087 grids (  1971, 27866, 10187,)
Level 11 has  536556 grids (     0,  8409,  2095,)
Level 12 has  155720 grids (     0,  2717,   608,)
Level 13 has   24607 grids (     0,   675,    96,)
Level 14 has    1137 grids (     0,    90,     4,)
```

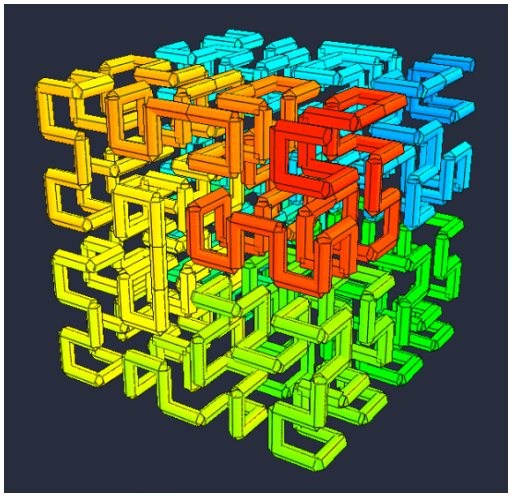
- The mesh has about 23 million Octs, corresponding to 184 million cells, at the resolution level 14.
- A uniform mesh would need 2^{45} cells to get to the same resolution → 35000 billion cells
- **Storing a single float variable requires 1.5GB with AMR, 280TB with the uniform mesh!**

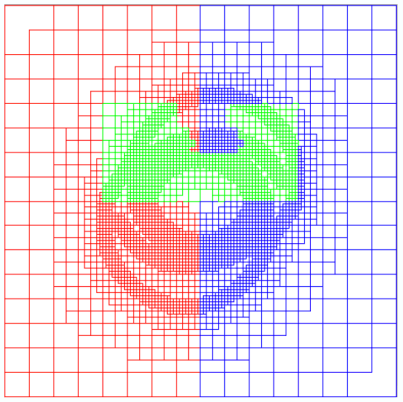
Parallelization Strategy: ENZO



- Level 0 grid partitioned across processors
- Level >0 grids within a processor executed sequentially
- Dynamic load balancing by messaging grids to underloaded processors

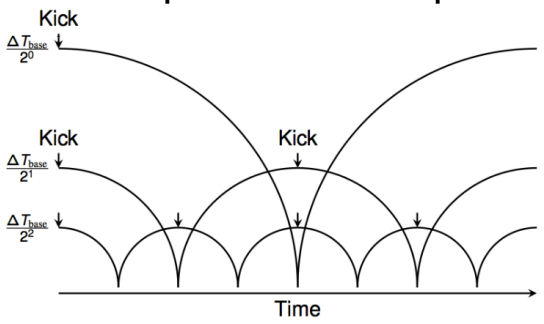
Parallelization Strategy: RAMSES



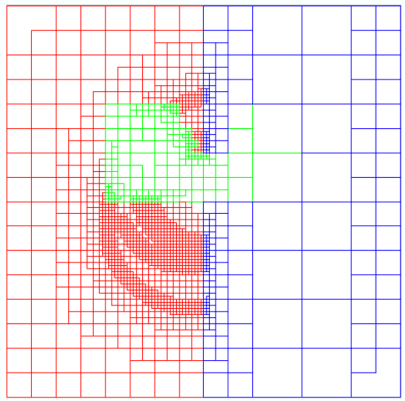


Domain decomposition over 3 processors

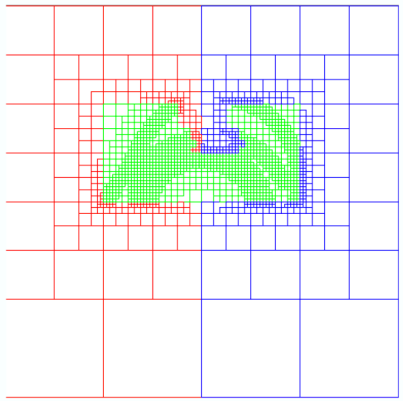
Adaptive Time Step



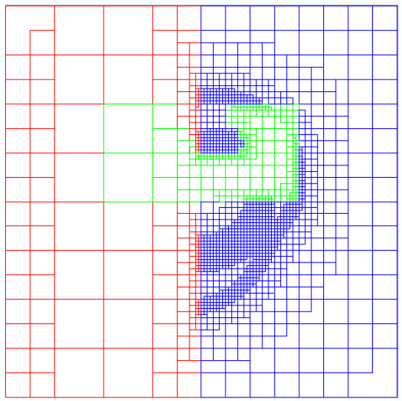
Each processor octree is surrounded by ghost cells (local copy of distant processor octrees) so that the resulting local octree contains all the necessary information.



Locally essential tree in processor #1



Locally essential tree in processor #2



Locally essential tree in processor #3

Some features compared

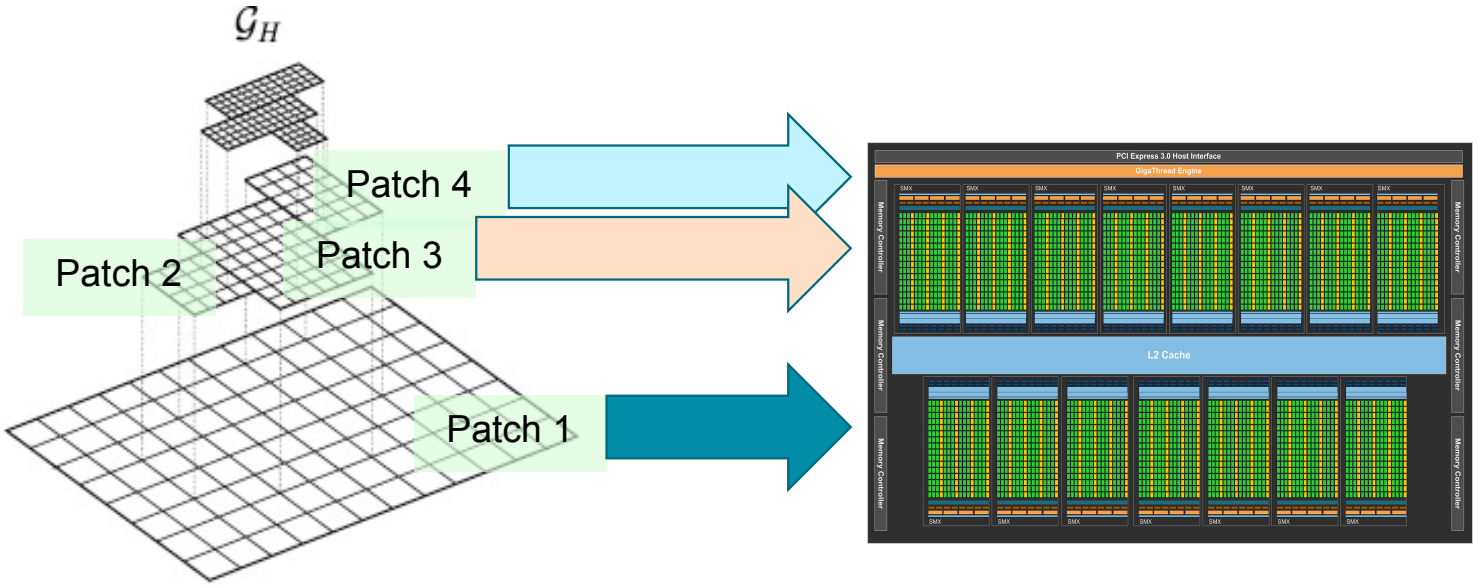
	ENZO	RAMSES
Parallelization	MPI, support of shared memory still missing	
Scalability	Up to some thousands of MPI ranks. Limited by communication.	O(1000) MPI ranks. Limited by load balancing.
Memory “consumption”	Overhead due to SAMR. Tree is replicated over ALL MPI ranks and can fill the memory.	Optimal
Memory usage	Effective, cash friendly, easily vectorizable	Low efficiency due to memory “fragmentation”
Portability	Highly portable. Supports all standard compilers and MPI libraries. Combination of C++ and F90 sometimes tricky. Dependency only from HDF5 library.	Extremely portable. Supports all standard compilers and MPI libraries Straightforward building. No external dependencies

What about support to accelerators?

- No Xeon-PHI porting currently available
- Limited GPU support:
  - ENZO
    - Hydro and MHD solvers efficiently ported on the GPU using CUDA
  - RAMSES
    - Radiative Transfer module available on the GPU based on CUDA
    - On-going implementation of the full code using OpenACC

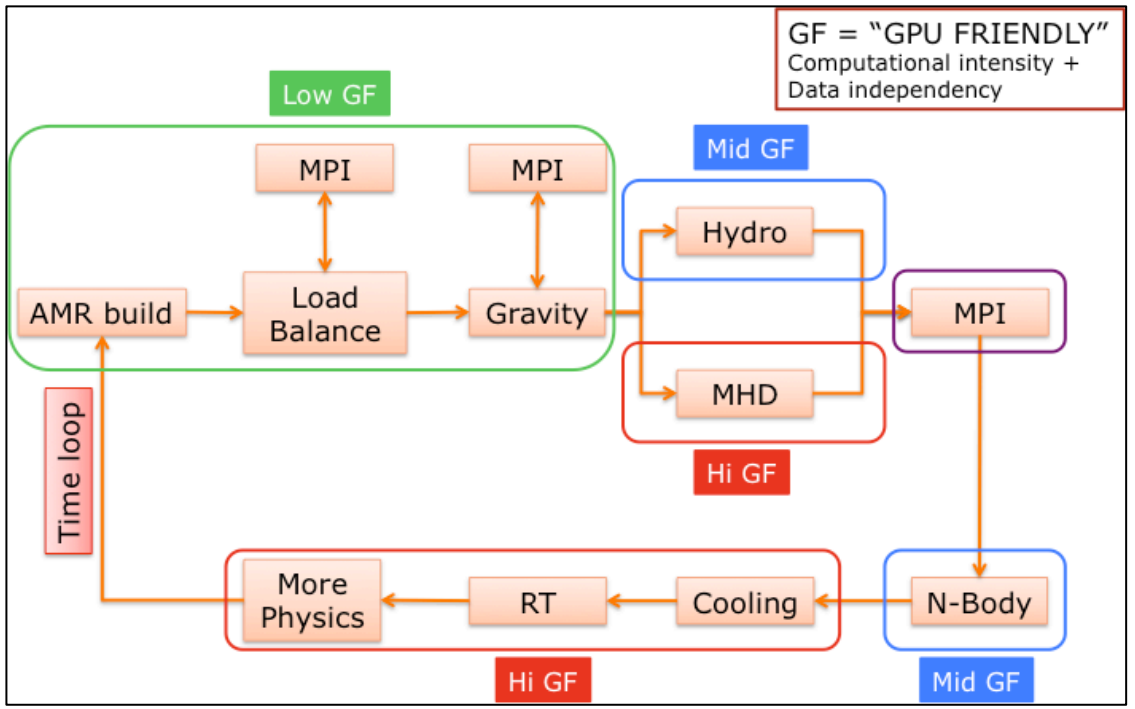
ENZO on the GPU

- Hydro and MHD kernels ported
- Each patch is an independent boundary value problem
- The GPU can solve each patch separately
- Efficient overlap of computation and data transfer
- Low arithmetic intensity for hydro, better for MHD



RAMSES on the GPU

- Radiative Transfer kernel parallelized with CUDA
  - Transport equation solved iteratively
  - Good arithmetic intensity, good efficiency
- Full code under refactoring based on OpenACC
  - Memory layout is the main problem
  - Full redesign of the algorithms required

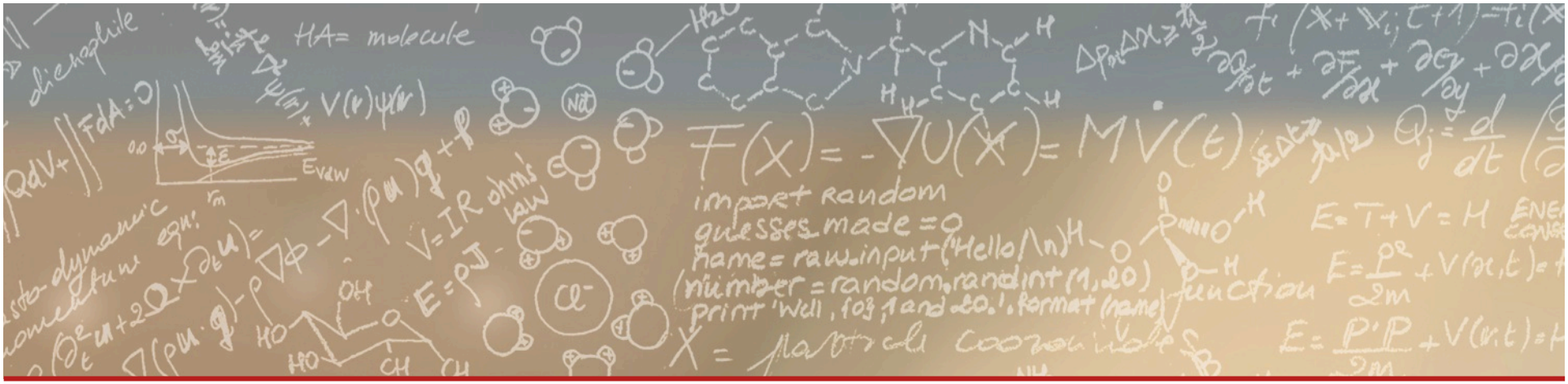


I/O

- Both codes support simple parallel I/O, each MPI task writing a different file
- ENZO writes HDF files
- RAMSES has a “proprietary” binary format (although it supports also HDF5)
- Files can be used also for restart
- Restarting with a different number of MPI tasks is tricky
- The number of files to handle can become “unfriendly”

Final remarks

- Both ENZO and RAMSES are very good tools for numerical astrophysics.
- RAMSES is a little better in terms of accuracy, ENZO is more HPC oriented
- Both have a quite big community of users and developers
- Both are on continuous update and are trying to support new HPC architectures to solve bigger and more complex problems
- From the astronomer point of view, RAMSES is easier to develop. From a computer scientist point of view ENZO is “better”…
- Support is quite limited for both. Documentation is more exaustive for ENZO.



Thank you for your attention.