Conservative finite difference schemes and adaptive mesh refinement techniques for hyperbolic systems of conservation laws

Pep Mulet, Universitat de València

Antonio Baeza, IMDEA-Mathematics

Outline

- Finite-difference Shu-Osher schemes
- Adaptive mesh refinement
- A look at the complete algorithm
- Some issues

Problem statement

Hyperbolic system of conservation laws (1D case):

$$\begin{cases} u_t + f(u)_x = 0 & \text{in } \mathbb{R} \times \mathbb{R}^+ \\ u(x,0) = u_0(x) & \text{in } \mathbb{R} \end{cases}$$

Problem statement

Hyperbolic system of conservation laws (1D case):

$$\begin{cases} u_t + f(u)_x = 0 & \text{in } \mathbb{R} \times \mathbb{R}^+ \\ u(x,0) = u_0(x) & \text{in } \mathbb{R} \end{cases}$$

Semi-discrete conservative schemes: Sustitute the term $f(u)_x$ by a discrete approximation

$$f(u)_x \approx \frac{\hat{f}_{j+\frac{1}{2}} - \hat{f}_{j-\frac{1}{2}}}{\Delta x} \quad \text{at } c_j = [j\Delta x, (j+1)\Delta x]$$

Problem statement

Hyperbolic system of conservation laws (1D case):

$$\begin{cases} u_t + f(u)_x = 0 & \text{in } \mathbb{R} \times \mathbb{R}^+ \\ u(x,0) = u_0(x) & \text{in } \mathbb{R} \end{cases}$$

Semi-discrete conservative schemes: Sustitute the term $f(u)_x$ by a discrete approximation

$$f(u)_x \approx \frac{\hat{f}_{j+\frac{1}{2}} - \hat{f}_{j-\frac{1}{2}}}{\Delta x} \quad \text{at } c_j = [j\Delta x, (j+1)\Delta x]$$

Finally solve the ODE system

$$u_t + \frac{\hat{f}_{j+\frac{1}{2}} - \hat{f}_{j-\frac{1}{2}}}{\Delta x} = 0$$

Finite-difference Shu-Osher schemes

- Finite volume schemes evolve cell-averages of the solution according to the integral form of the conservation law

Finite-difference Shu-Osher schemes

- Finite volume schemes evolve cell-averages of the solution according to the integral form of the conservation law
- **[Shu & Osher]** Finite-difference scheme based on evolution of point-values.

Finite-difference Shu-Osher schemes

- Finite volume schemes evolve cell-averages of the solution according to the integral form of the conservation law
- **[Shu & Osher]** Finite-difference scheme based on evolution of point-values.
- Key idea: express the space derivative $f(u)_x$ as a finite difference:

$$
\begin{aligned}
f(u(x,t)) &= \frac{1}{\Delta x}\int_{x-\frac{\Delta x}{2}}^{x+\frac{\Delta x}{2}} \phi(s)ds \\
f(u(x,t))_x &= \frac{\phi(x+\frac{\Delta x}{2}) - \phi(x-\frac{\Delta x}{2})}{\Delta x}
\end{aligned}
$$

for unknown ϕ, whose average on cell $[x_{i-\frac{1}{2}}, x_{i+\frac{1}{2}}]$ is $f(u(x_i,t))$ ($x_i = i\Delta x$).

Finite-difference Shu-Osher schemes

- Finite volume schemes evolve cell-averages of the solution according to the integral form of the conservation law
- **[Shu & Osher]** Finite-difference scheme based on evolution of point-values.
- Key idea: express the space derivative $f(u)_x$ as a finite difference:

$$\begin{aligned} f(u(x,t)) &= \frac{1}{\Delta x}\int_{x-\frac{\Delta x}{2}}^{x+\frac{\Delta x}{2}} \phi(s)ds \\ f(u(x,t))_x &= \frac{\phi(x+\frac{\Delta x}{2})-\phi(x-\frac{\Delta x}{2})}{\Delta x} \end{aligned}$$

  for unknown ϕ, whose average on cell $[x_{i-\frac{1}{2}}, x_{i+\frac{1}{2}}]$ is $f(u(x_i,t))$ ($x_i = i\Delta x$).
- Goal: to approximate point-values $\phi(x_{i+\frac{1}{2}})$ of ϕ from its cell-averaves, i.e. the fluxes $f_i = f(u(x_i,t))$

Finite-difference Shu-Osher schemes

- Finite volume schemes evolve cell-averages of the solution according to the integral form of the conservation law
- **[Shu & Osher]** Finite-difference scheme based on evolution of point-values.
- Key idea: express the space derivative $f(u)_x$ as a finite difference:

$$
\begin{aligned}
f(u(x,t)) &= \frac{1}{\Delta x}\int_{x-\frac{\Delta x}{2}}^{x+\frac{\Delta x}{2}} \phi(s)ds \\
f(u(x,t))_x &= \frac{\phi(x+\frac{\Delta x}{2}) - \phi(x-\frac{\Delta x}{2})}{\Delta x}
\end{aligned}
$$

  for unknown ϕ, whose average on cell $[x_{i-\frac{1}{2}}, x_{i+\frac{1}{2}}]$ is $f(u(x_i,t))$ ($x_i = i\Delta x$).
- Goal: to approximate point-values $\phi(x_{i+\frac{1}{2}})$ of ϕ from its cell-averaves, i.e. the fluxes $f_i = f(u(x_i,t))$
- High order methods (PHM, ENO, WENO ...) can be used for that purpose.

Finite-difference Shu-Osher schemes

- Finite volume schemes evolve cell-averages of the solution according to the integral form of the conservation law
- **[Shu & Osher]** Finite-difference scheme based on evolution of point-values.
- Key idea: express the space derivative $f(u)_x$ as a finite difference:

$$
\begin{aligned}
f(u(x,t)) &= \frac{1}{\Delta x}\int_{x-\frac{\Delta x}{2}}^{x+\frac{\Delta x}{2}} \phi(s)ds \\
f(u(x,t))_x &= \frac{\phi(x+\frac{\Delta x}{2}) - \phi(x-\frac{\Delta x}{2})}{\Delta x}
\end{aligned}
$$

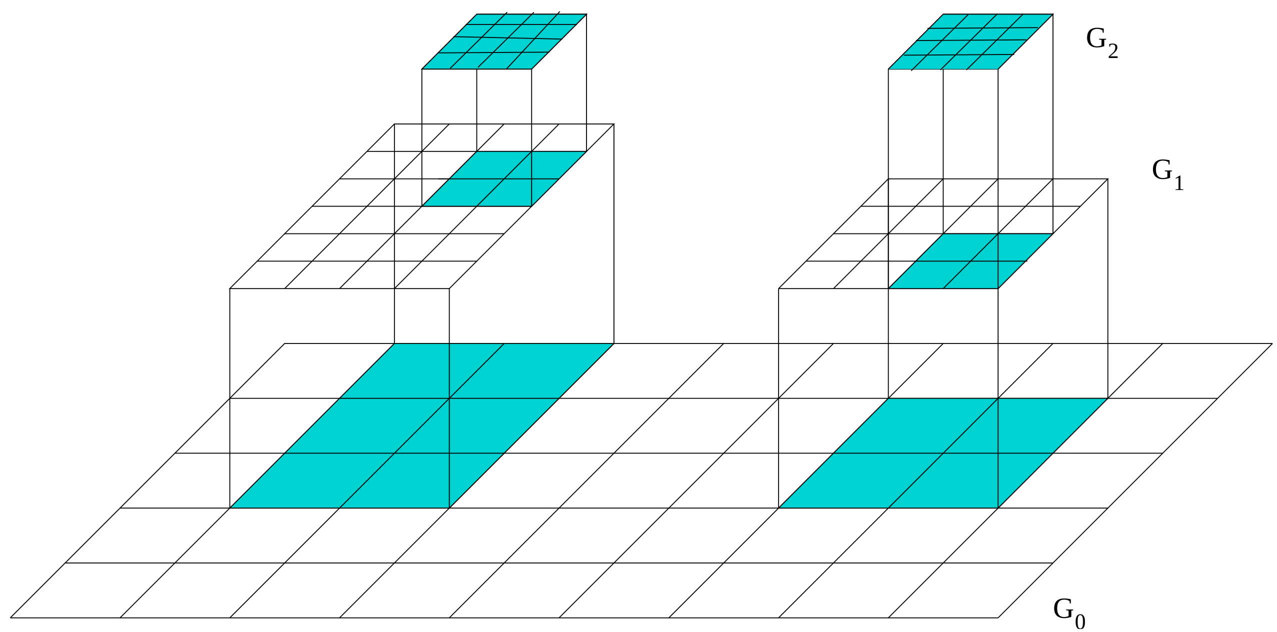
  for unknown ϕ, whose average on cell $[x_{i-\frac{1}{2}}, x_{i+\frac{1}{2}}]$ is $f(u(x_i,t))$ ($x_i = i\Delta x$).
- Goal: to approximate point-values $\phi(x_{i+\frac{1}{2}})$ of ϕ from its cell-averaves, i.e. the fluxes $f_i = f(u(x_i,t))$
- High order methods (PHM, ENO, WENO ...) can be used for that purpose.
- TVD Runge-Kutta methods are used for time evolution

Adaptive mesh refinement

AMR aims to locally refine the mesh by using a grid hierarchy composed by mesh patches having different levels of resolution

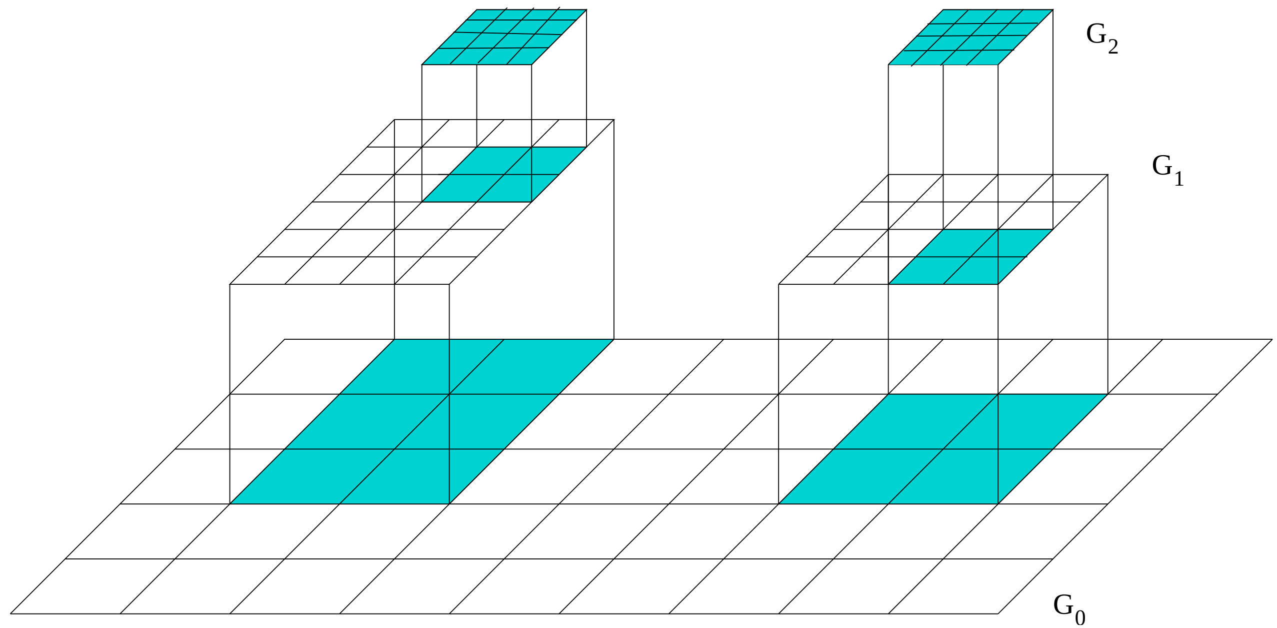
Adaptive mesh refinement

AMR aims to locally refine the mesh by using a grid hierarchy composed by mesh patches having different levels of resolution



Adaptive mesh refinement

AMR aims to locally refine the mesh by using a grid hierarchy composed by mesh patches having different levels of resolution



Key idea: To reduce the total number of cell updates (flux computations).

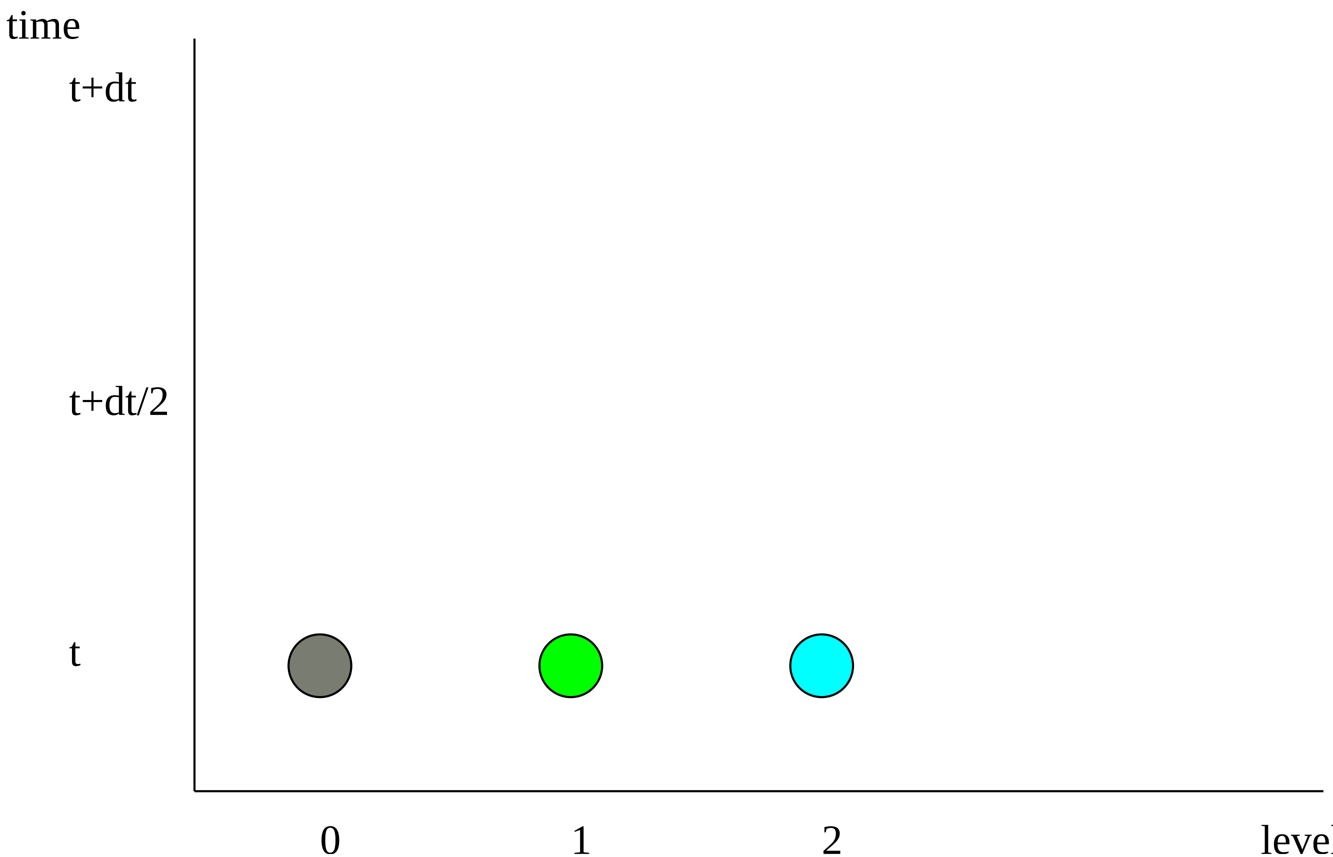
A look at the complete algorithm

We use a grid hierarchy $G_0, \ldots, G_L$:

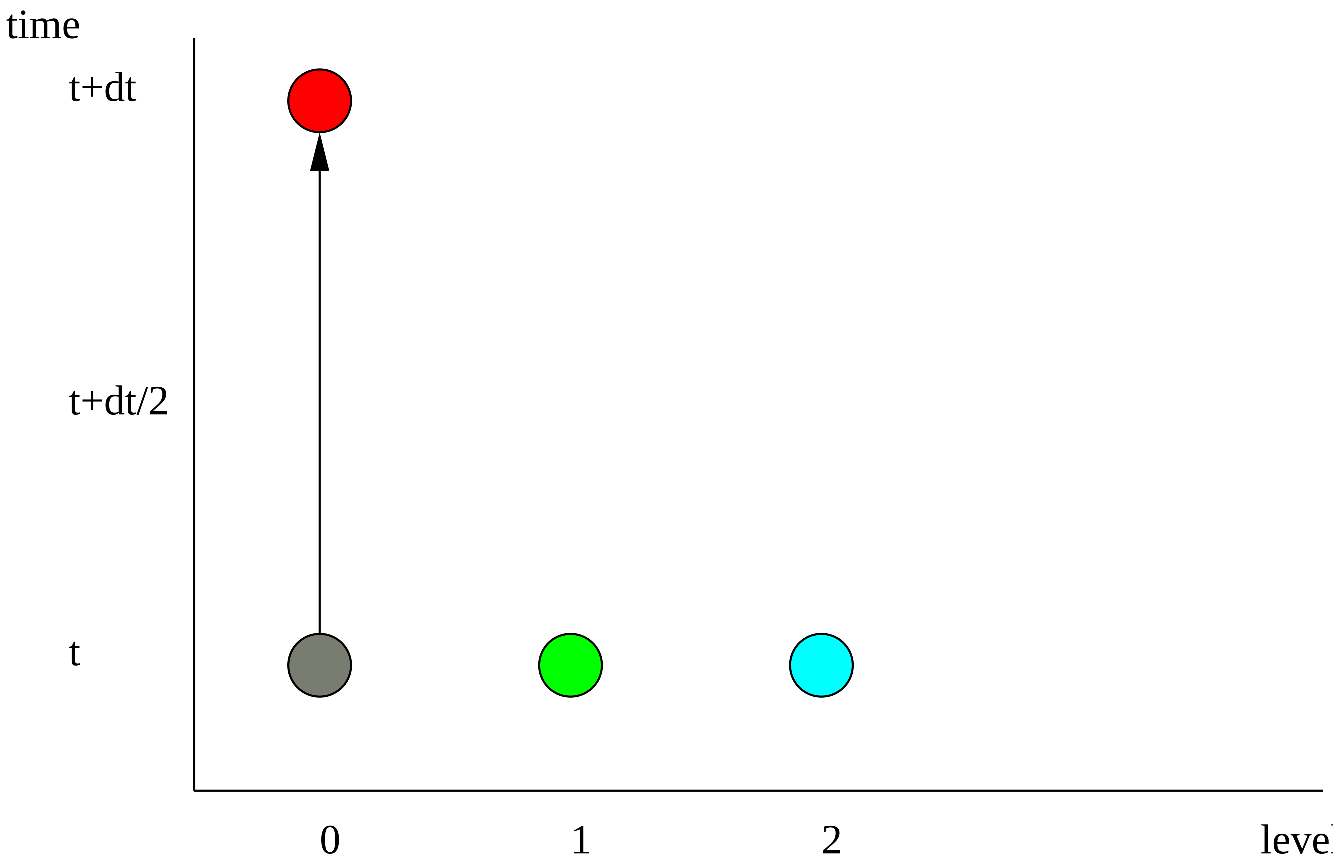
- $G_l \equiv$ union of Cartesian patches of uniform mesh size
- G_l is finer than G_{l-1} and $G_l \subseteq G_{l-1}$ (nestedness)
- Singularities **never** cross a fine mesh boundary (moving grids)

$\Rightarrow$ **Adaptive mesh refinement(AMR) [Berger, Oliger]**.

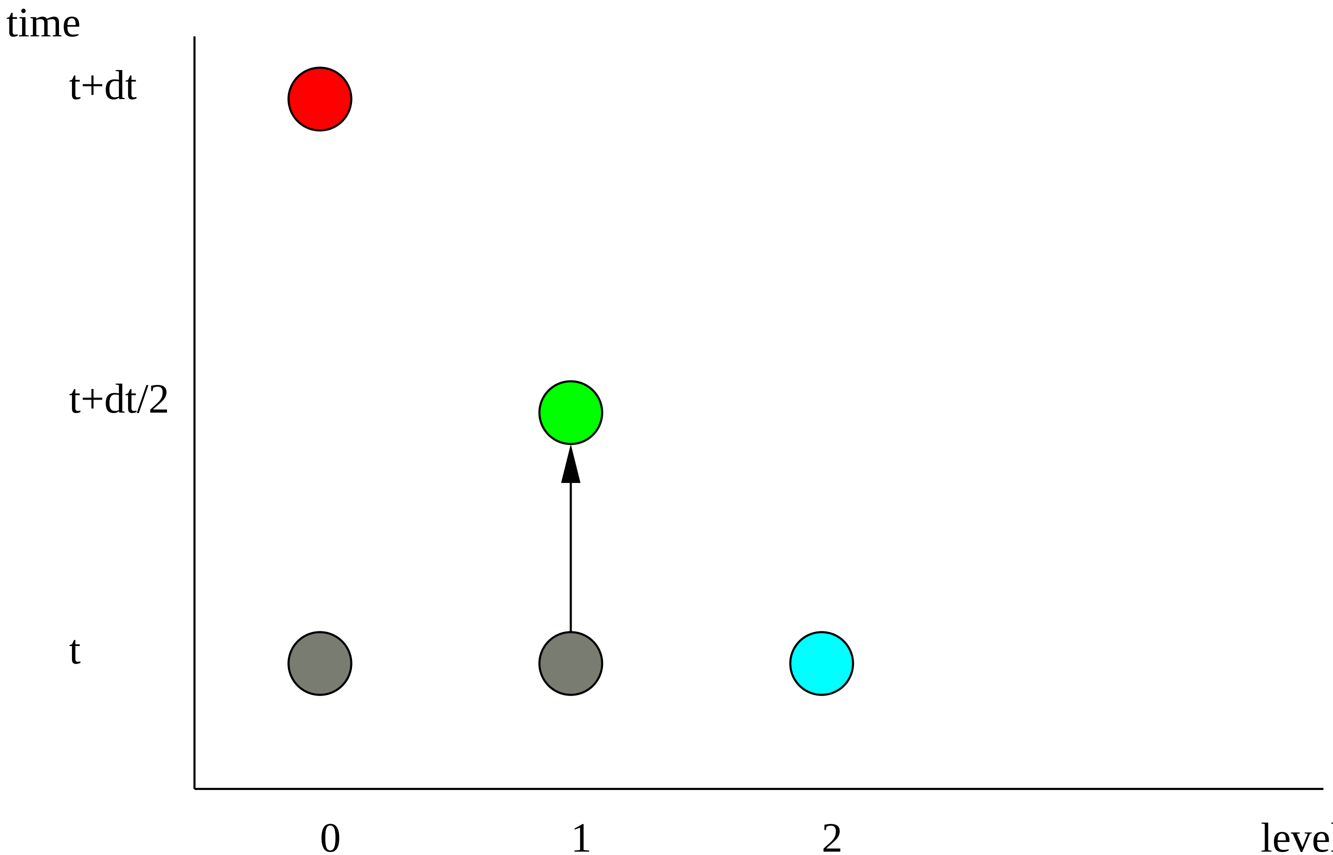
A look at the complete algorithm



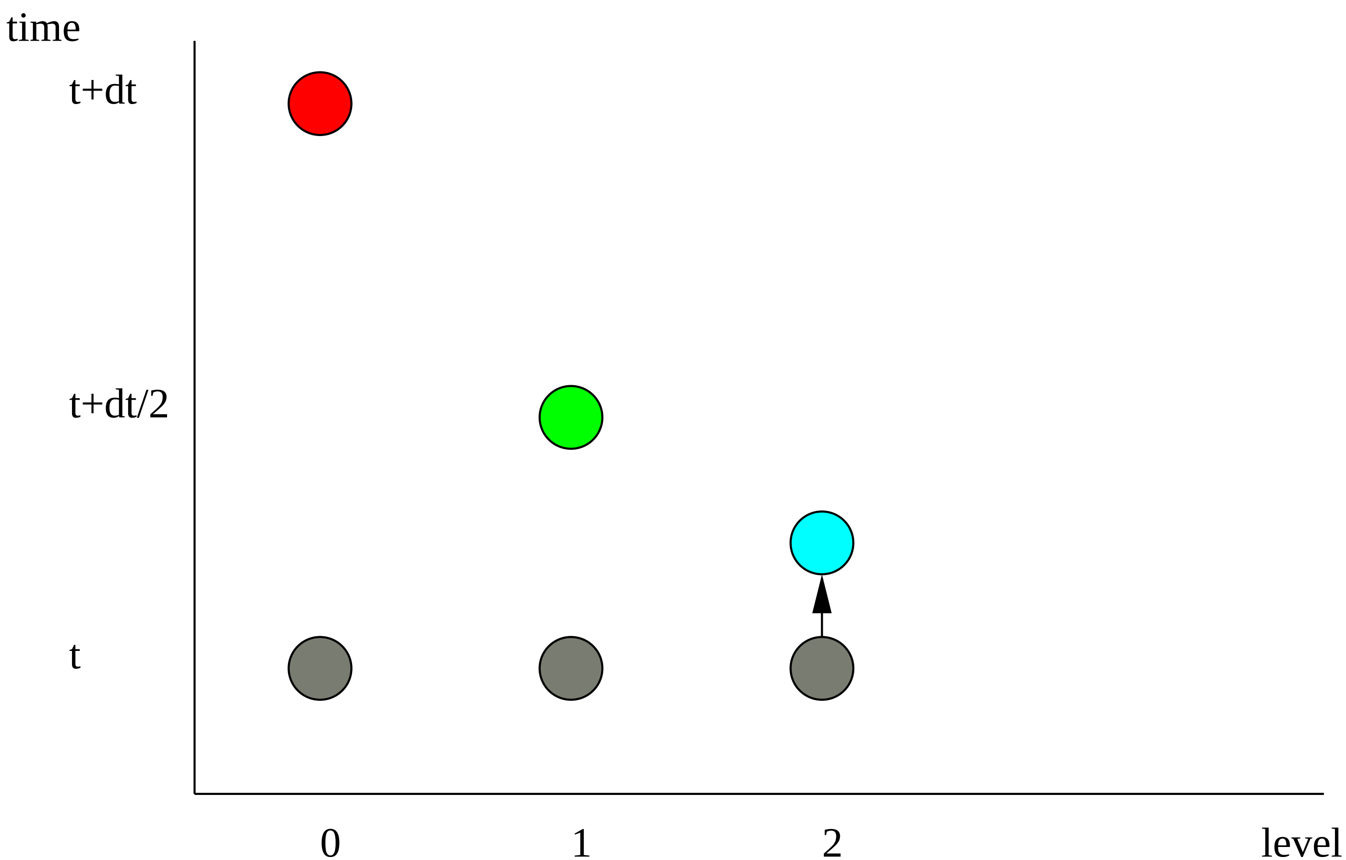
A look at the complete algorithm



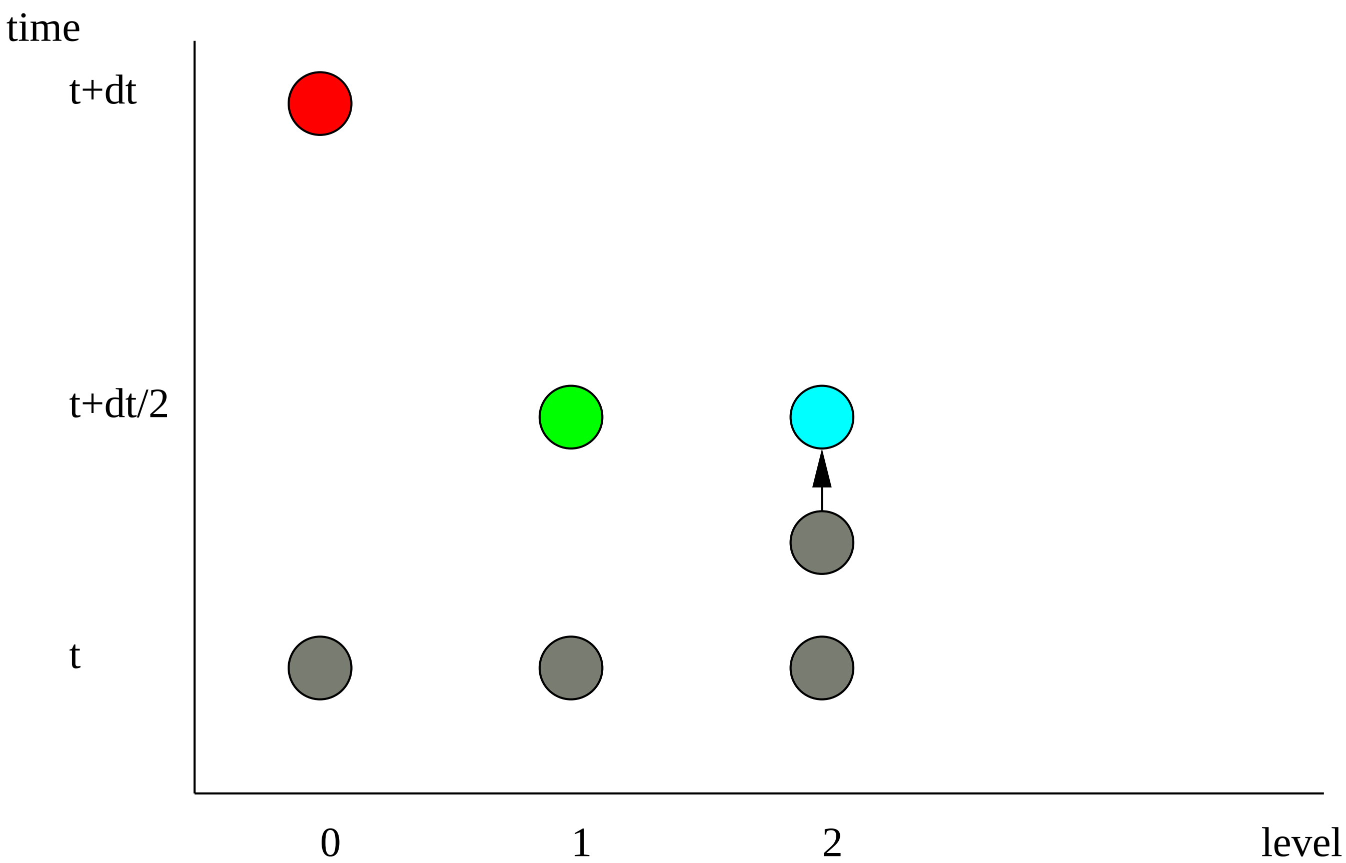
A look at the complete algorithm



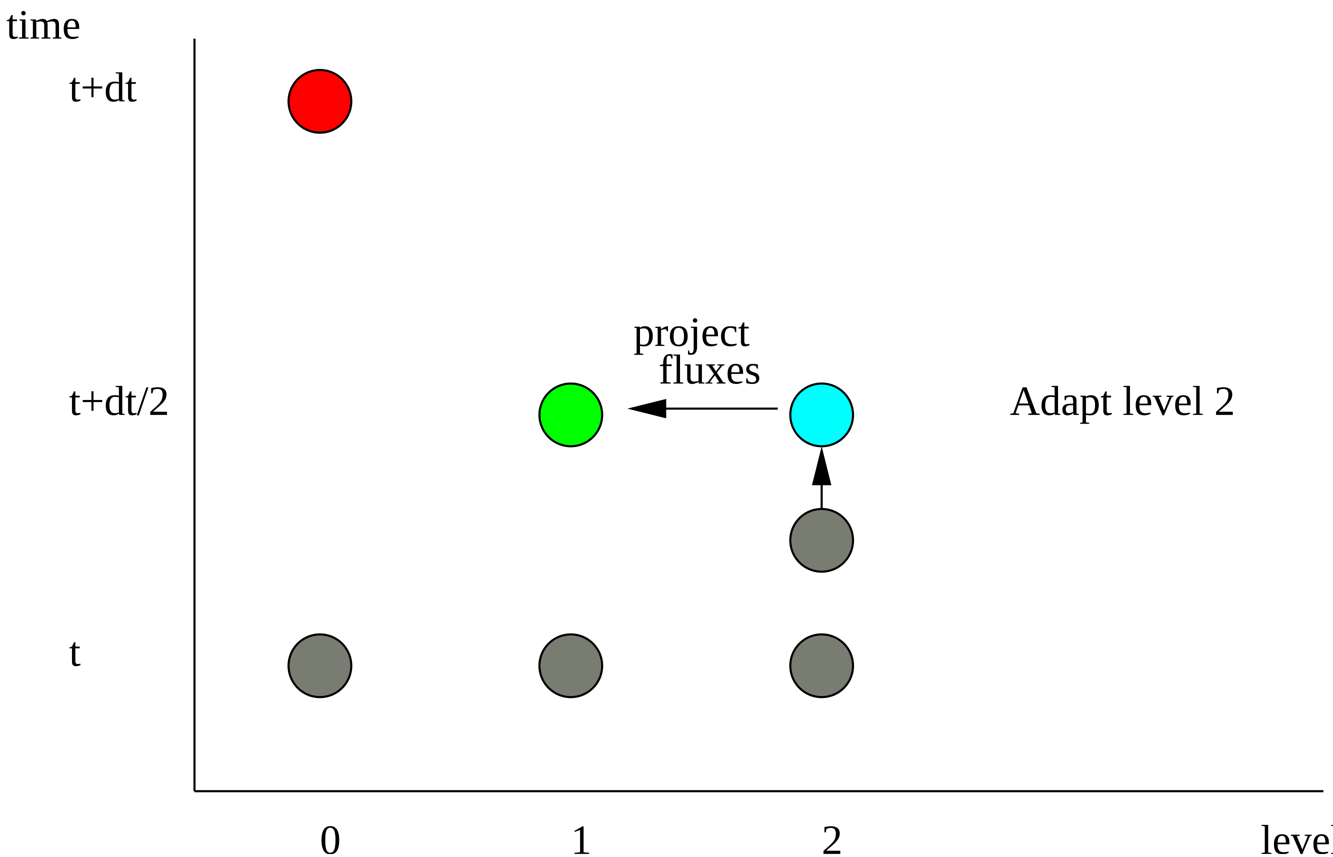
A look at the complete algorithm



A look at the complete algorithm

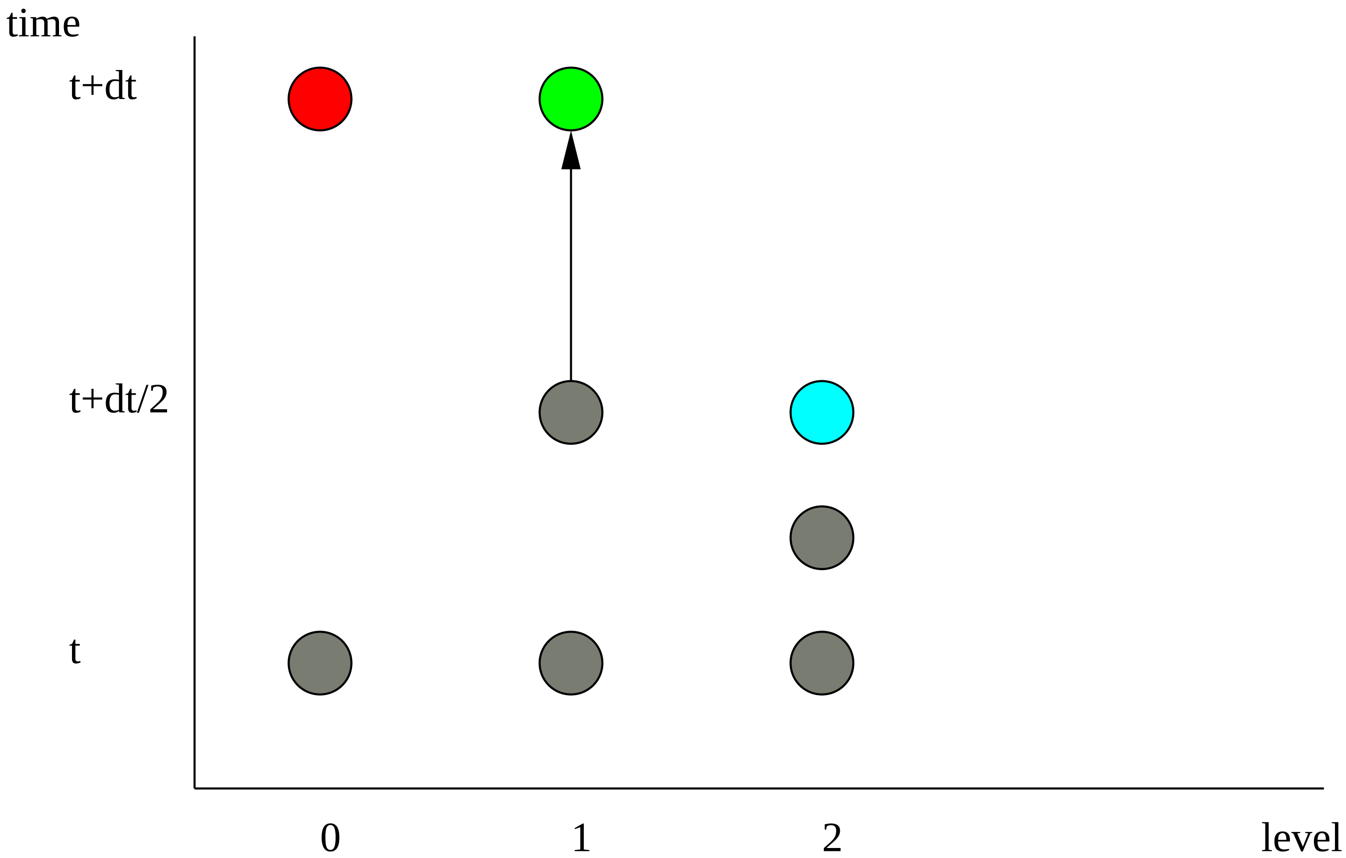


A look at the complete algorithm



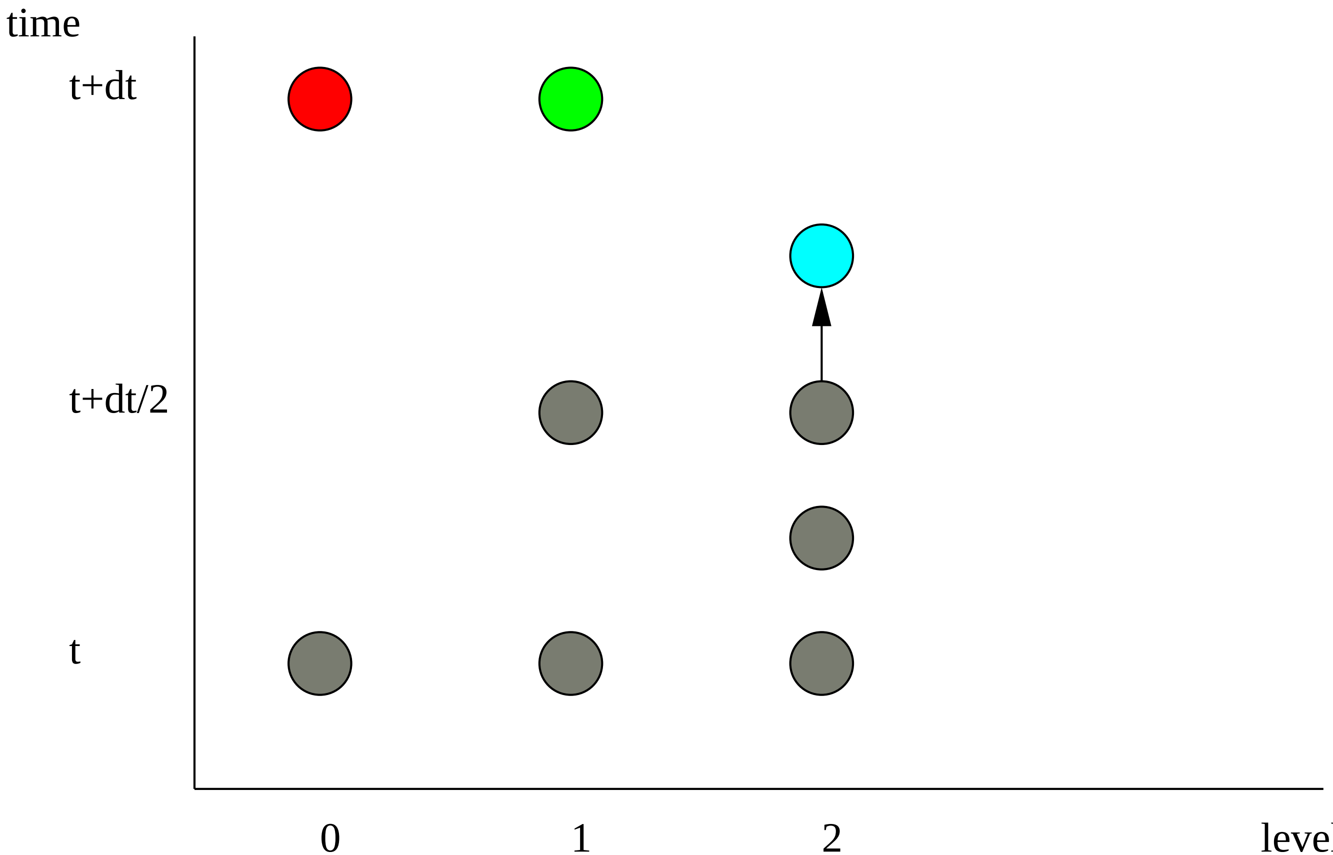
- Adapt level 2 before any singularity escapes.

A look at the complete algorithm



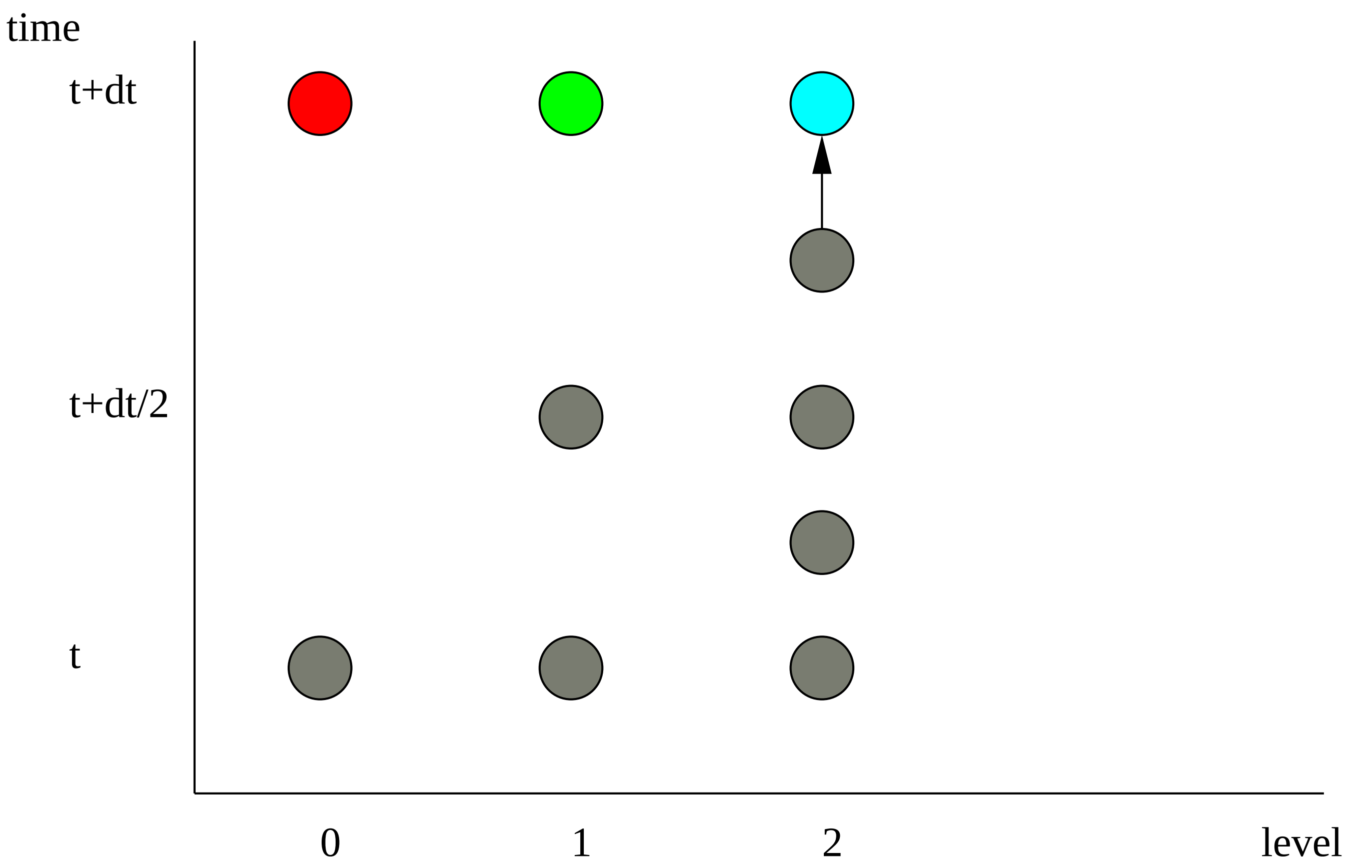
- Adapt level 2 before any singularity escapes.

A look at the complete algorithm



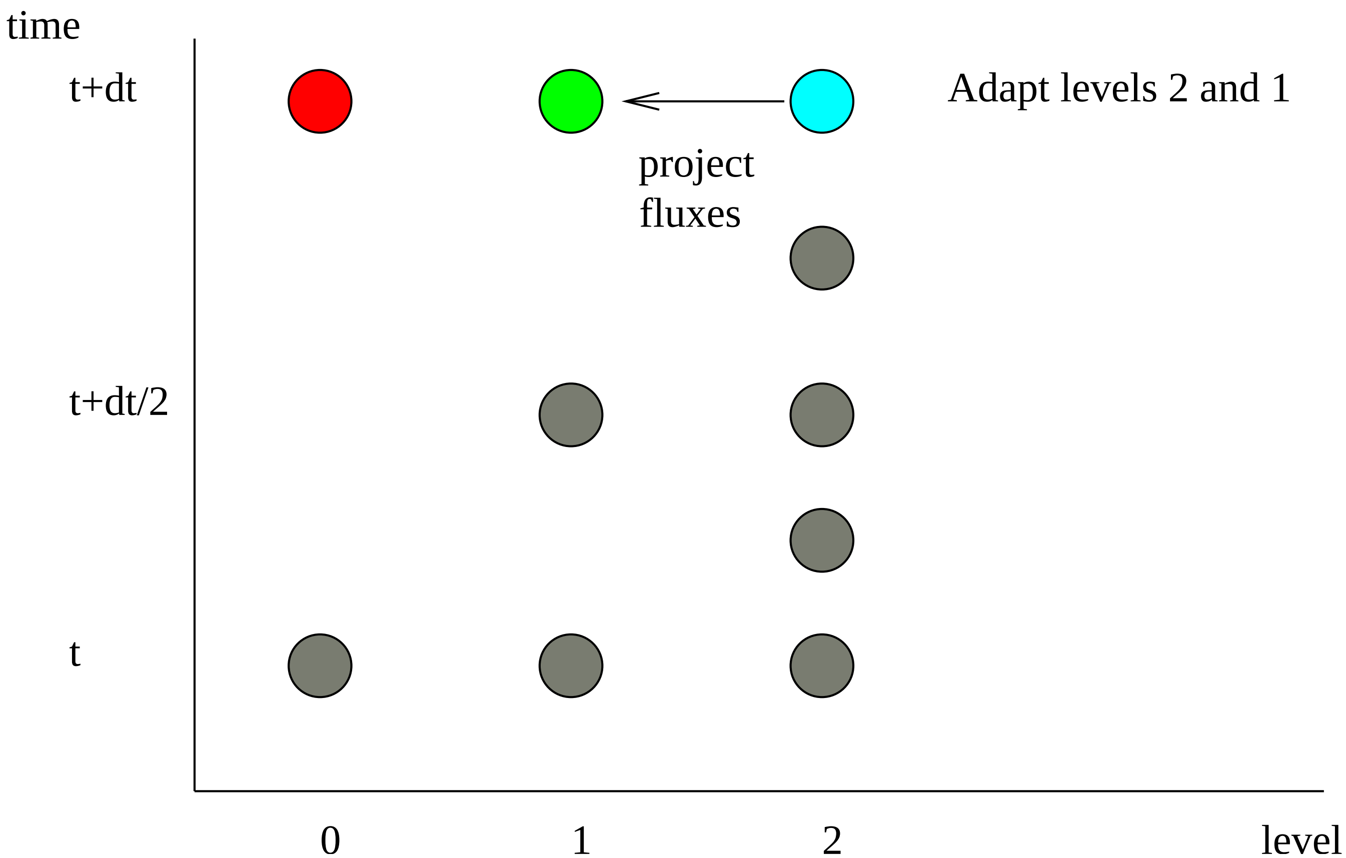
- Adapt level 2 before any singularity escapes.

A look at the complete algorithm



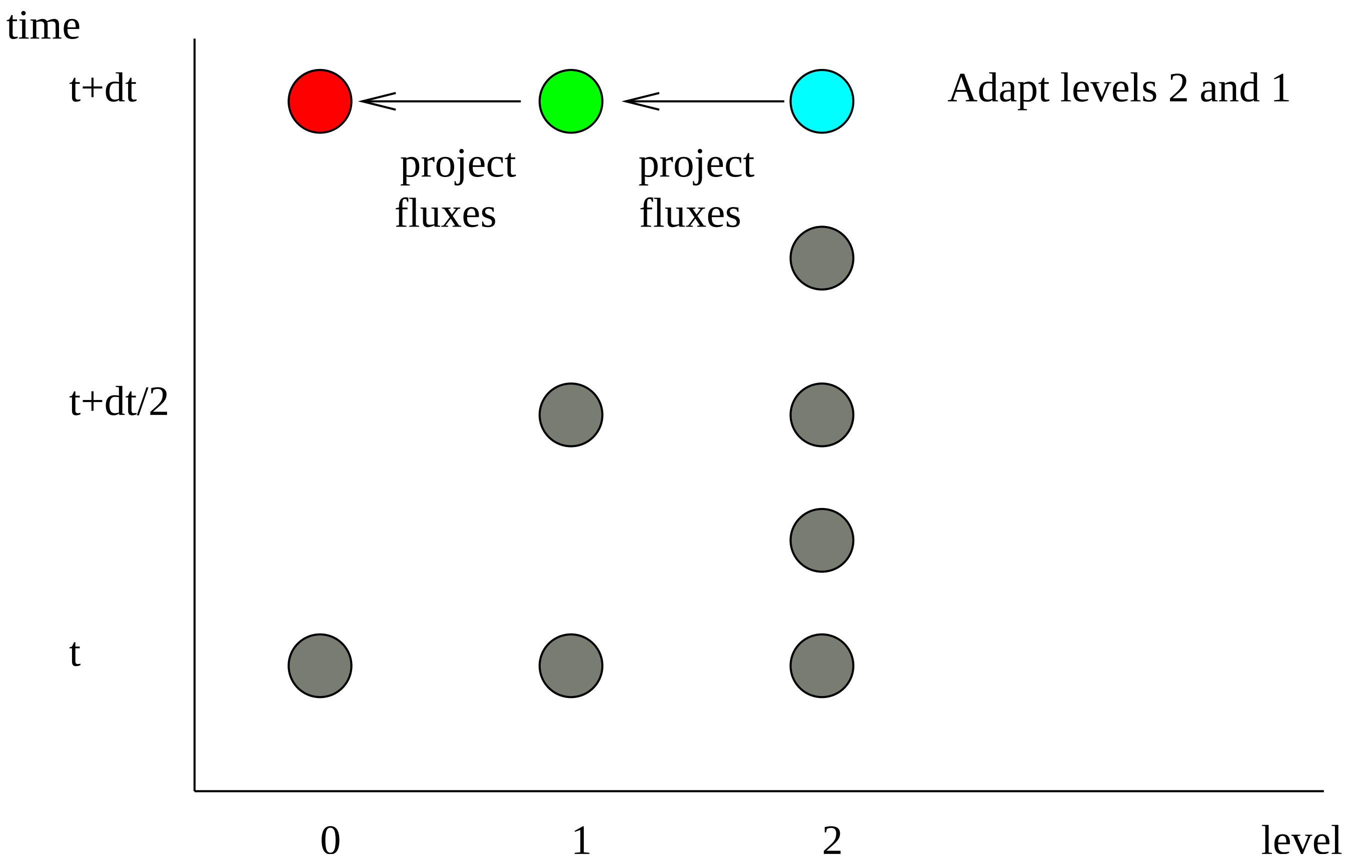
- Adapt level 2 before any singularity escapes.

A look at the complete algorithm



- Adapt level 2 before any singularity escapes.
- Adapt level 2 and 1 before any singularity escapes.

A look at the complete algorithm



- Adapt level 2 before any singularity escapes.
- Adapt level 2 and 1 before any singularity escapes.

Some issues

- Refinement criteria

Some issues

- Refinement criteria
  - Sensors based on gradients, second derivatives

Some issues

- Refinement criteria
  - Sensors based on gradients, second derivatives
  - Multiresolution analisis, error estimation

Some issues

- Refinement criteria
  - Sensors based on gradients, second derivatives
  - Multiresolution analisis, error estimation
- Projection of solution

Some issues

- Refinement criteria
  - Sensors based on gradients, second derivatives
  - Multiresolution analisis, error estimation
- Projection of solution
  - Once evolved $t_n \rightarrow t_{n+1}$, solution in G_l is more precise than in coarser grid G_{l-1}

Some issues

- Refinement criteria
  - Sensors based on gradients, second derivatives
  - Multiresolution analisis, error estimation
- Projection of solution
  - Once evolved $t_n \to t_{n+1}$, solution in G_l is more precise than in coarser grid G_{l-1} $\Rightarrow$ modify solution (at $t = t_{n+1}$) in G_{l-1} from solution at G_l **conservatively**:
    - first modify numerical fluxes at interfaces of cells in G_{l-1} covered by G_l
    - then modify the solution at t_{n+1} and level $l-1$.

Some issues

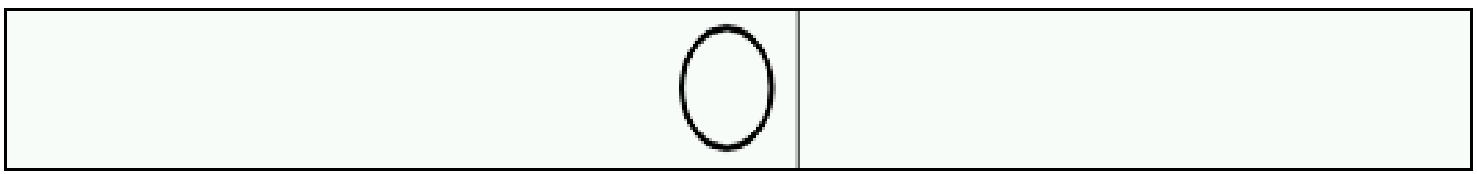
- Refinement criteria
  - Sensors based on gradients, second derivatives
  - Multiresolution analisis, error estimation
- Projection of solution
  - Once evolved $t_n \rightarrow t_{n+1}$, solution in G_l is more precise than in coarser grid $G_{l-1} \;\Rightarrow$ modify solution (at $t = t_{n+1}$) in G_{l-1} from solution at G_l **conservatively**:
    - first modify numerical fluxes at interfaces of cells in G_{l-1} covered by G_l
    - then modify the solution at t_{n+1} and level $l-1$.
  - This **projection** from **fine** fluxes to **coarse** fluxes entails communication from finest to coarsest grids and is fundamental for the efficiency of the algorithm.

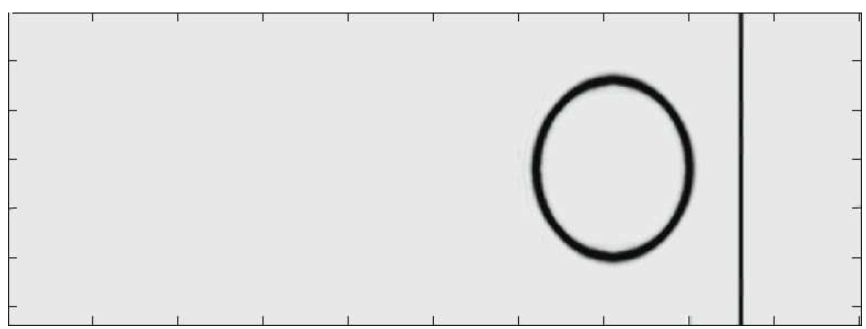
Some issues

- Refinement criteria
  - Sensors based on gradients, second derivatives
  - Multiresolution analisis, error estimation
- Projection of solution
  - Once evolved $t_n \to t_{n+1}$, solution in G_l is more precise than in coarser grid G_{l-1} $\Rightarrow$ modify solution (at $t = t_{n+1}$) in G_{l-1} from solution at G_l **conservatively**:
    - first modify numerical fluxes at interfaces of cells in G_{l-1} covered by G_l
    - then modify the solution at t_{n+1} and level $l-1$.
  - This **projection** from **fine** fluxes to **coarse** fluxes entails communication from finest to coarsest grids and is fundamental for the efficiency of the algorithm.
- Algorithm implementation, Parallelisation

Shock-Helium bubble

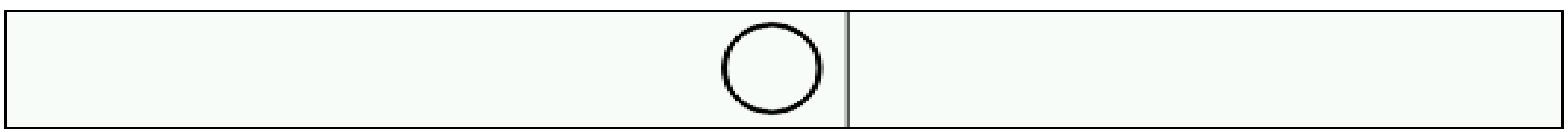
- Mach 1.22 shock interaction with Helium bubble **[Haas & Sturtevant]**, **[Karni & Quirk]**, **[Marquina & Mulet]**.
- Basic scheme: Shu-Osher+Donat-Marquina+WENO 5 reconstruction $\Rightarrow 5^{\text{th}}$ order space accuracy + 3^{rd} order time accuracy.





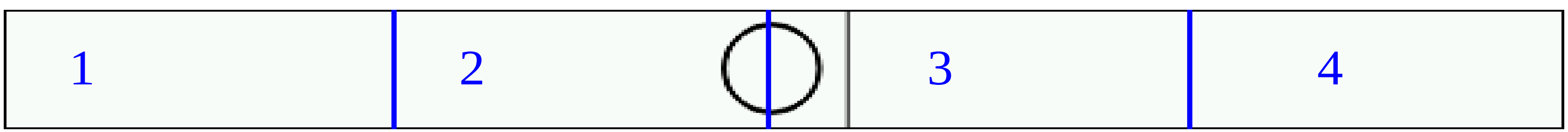
Parallel implementation

- Parallelization by domain decomposition: split G_0 and evenly assign each piece (along with overlying pieces of each G_l) to processors.



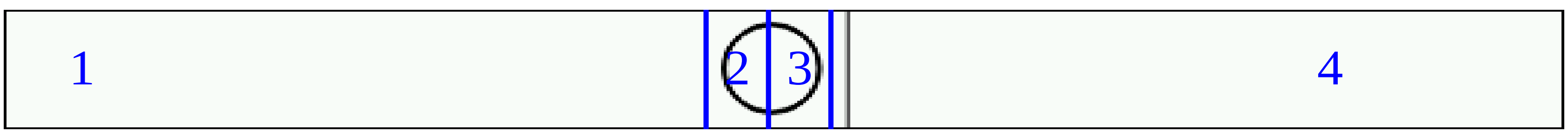
Parallel implementation

- Parallelization by domain decomposition: split G_0 and evenly assign each piece (along with overlying pieces of each G_l) to processors.
- Difficulty with **load balancing** (each processor performs same work):
  - If assignment is spatially symmetric processors 2 and 3 get assigned the heaviest part.



Parallel implementation

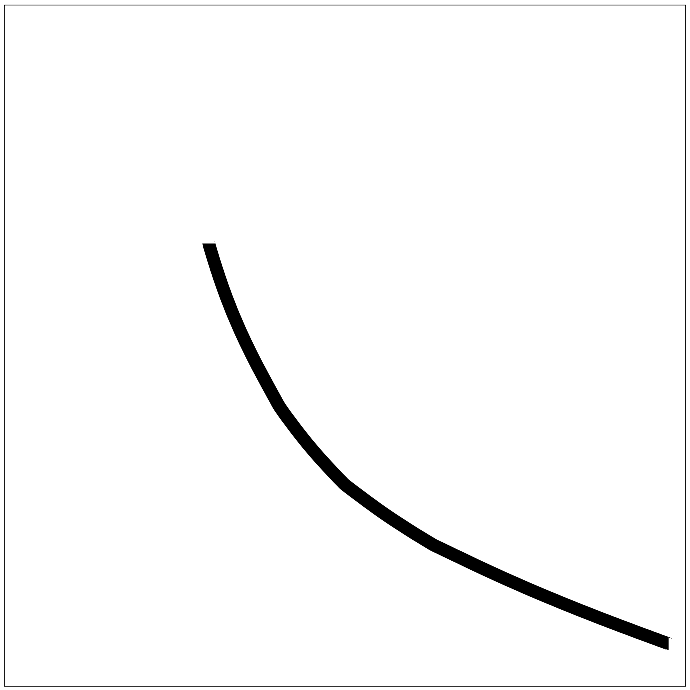
- Parallelization by domain decomposition: split G_0 and evenly assign each piece (along with overlying pieces of each G_l) to processors.
- Difficulty with **load balancing** (each processor performs same work):
  - If assignment is spatially symmetric processors 2 and 3 get assigned the heaviest part.
  - Assignment must be asymmetric $\cdots$



Parallel implementation

- Parallelization by domain decomposition: split G_0 and evenly assign each piece (along with overlying pieces of each G_l) to processors.
- Difficulty with **load balancing** (each processor performs same work):
  - If assignment is spatially symmetric processors 2 and 3 get assigned the heaviest part.
  - Assignment must be asymmetric $\cdots$
  - but can not be static, since now processor 1 does almost all the work.

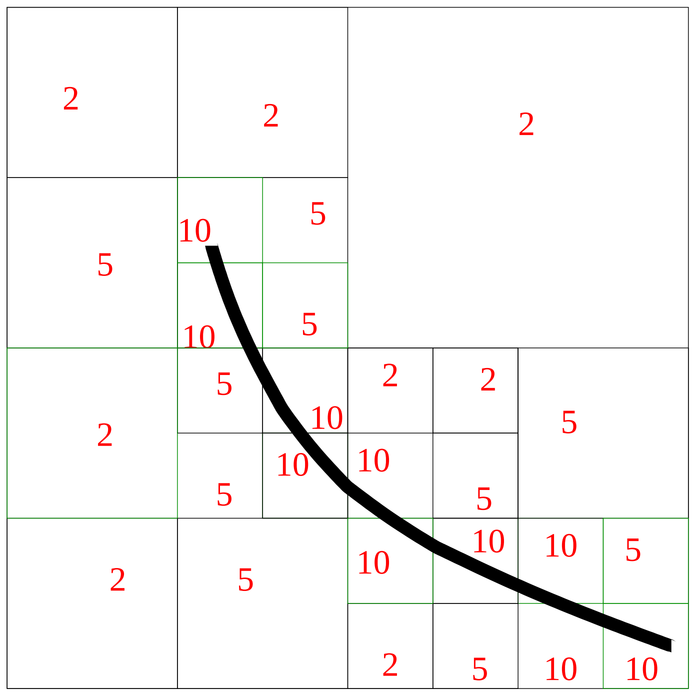
Parallel implementation



discontinuity

- **[Parashar & coauthors]**, **[Devine & coauthors]**
- **[Baeza & Mulet]** work in progress . . .
- Need multidimensional partitioning.

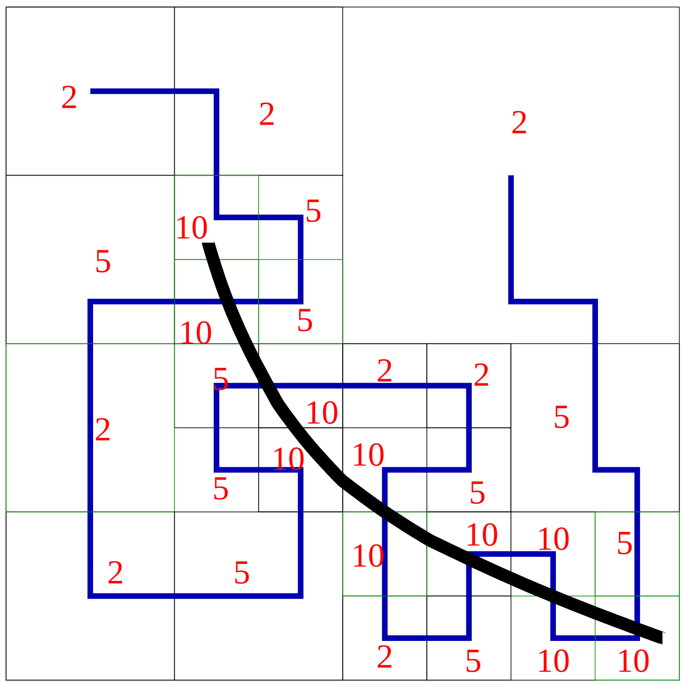
Parallel implementation



computational costs

- **[Parashar & coauthors]**, **[Devine & coauthors]**
- **[Baeza & Mulet]** work in progress . . .
- Need multidimensional partitioning.
- Recursively bisect until some level and estimate computational costs.

Parallel implementation

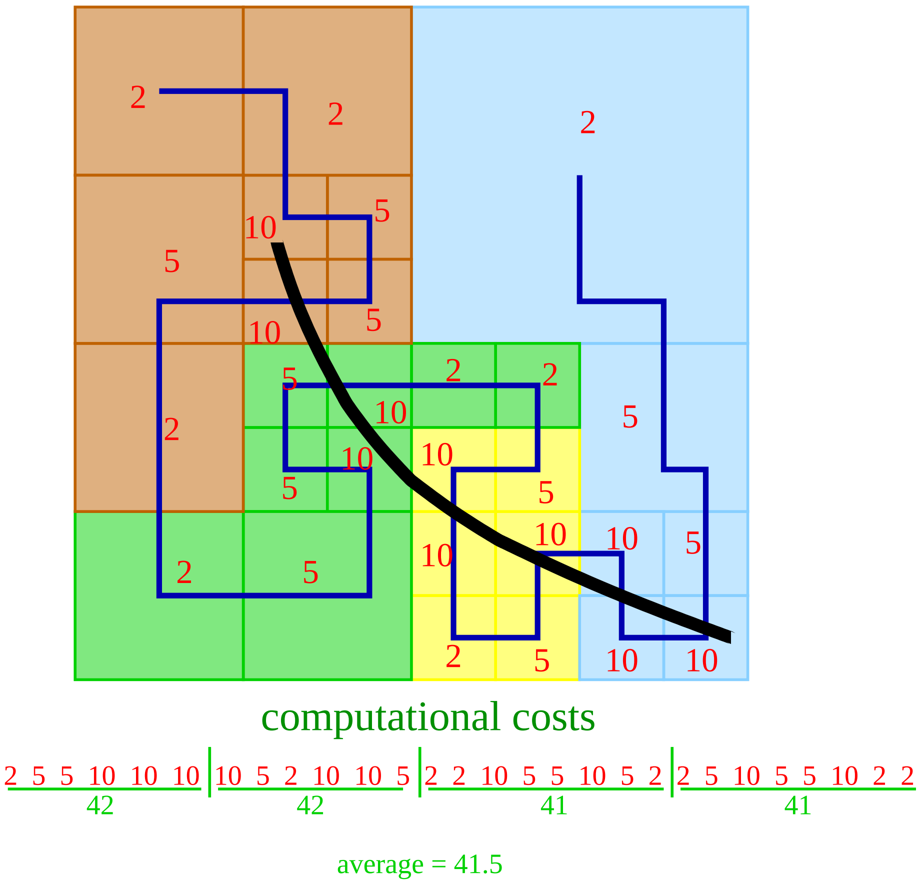


computational costs

2 5 5 10 10 10 10 5 2 10 10 5 2 2 10 5 5 10 5 2 2 5 10 5 5 10 2 2

- **[Parashar & coauthors]**, **[Devine & coauthors]**
- **[Baeza & Mulet]** work in progress . . .
- Need multidimensional partitioning.
- Recursively bisect until some level and estimate computational costs.
- Use Peano-Hilbert curve to uni-dimensionally order by proximity.

Parallel implementation



- **[Parashar & coauthors]**, **[Devine & coauthors]**
- **[Baeza & Mulet]** work in progress . . .
- Need multidimensional partitioning.
- Recursively bisect until some level and estimate computational costs.
- Use Peano-Hilbert curve to uni-dimensionally order by proximity.
- Assign work evenly (trying to minimize communication cost) .