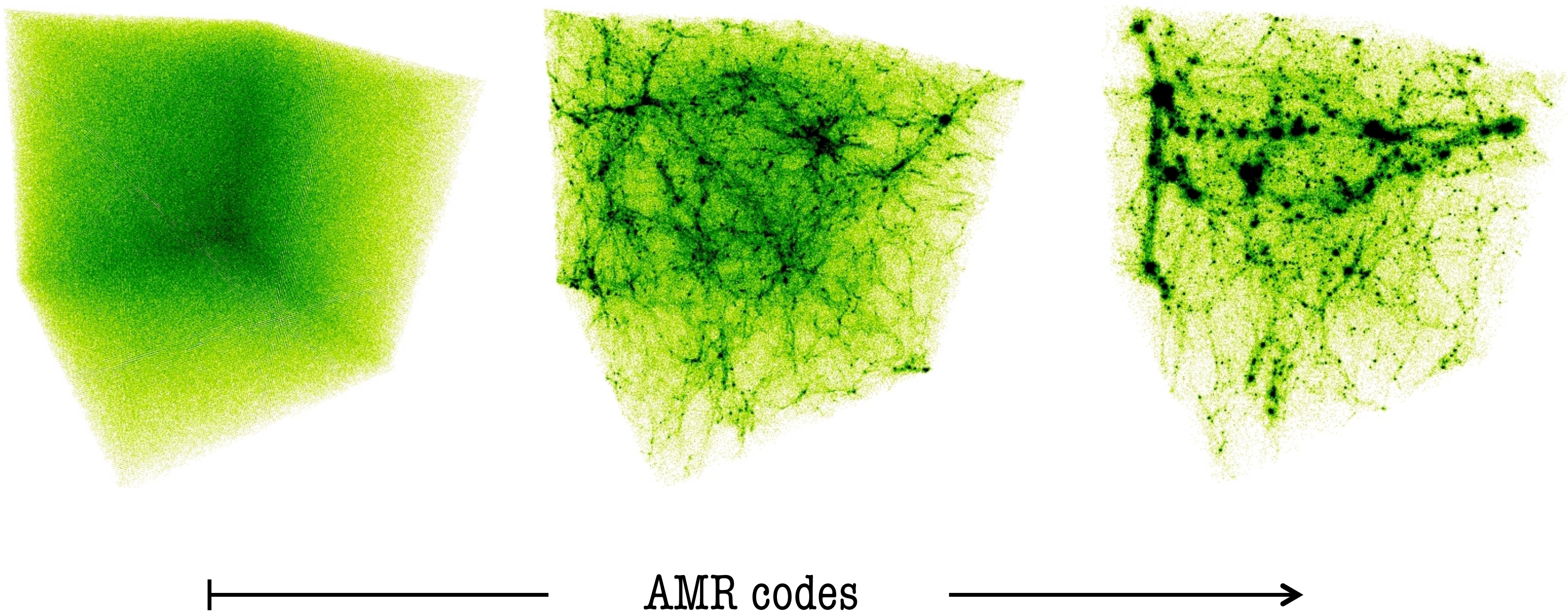
Alexander Knebe, *Universidad Autonoma de Madrid*

Adaptive Mesh Refinement

AMR Codes



- Poisson's equation

$$\Delta\Phi(\vec{x}) = 4\pi G\rho(\vec{x})$$

- Poisson's equation

$$\vec{F}(\vec{x}) = -m\nabla\Phi(\vec{x})$$

$$\Delta\Phi(\vec{x}) = 4\pi G\rho(\vec{x})$$

particle approach

$$\vec{F}(\vec{x}_i) = -\sum_{i\neq j}\frac{Gm_i m_j}{(x_i - x_j)^3}(\vec{x}_i - \vec{x}_j)$$

grid approach ($\vec{x}_{i,j,k}$ =position of centre of grid cell (i,j,k))

$$\Delta\Phi(\vec{x}_{i,j,k}) = 4\pi G\rho(\vec{x}_{i,j,k})$$

$$\vec{F}(\vec{x}_{i,j,k}) = -m\nabla\Phi(\vec{x}_{i,j,k})$$

- Poisson's equation

$$\vec{F}(\vec{x}) = -m\nabla\Phi(\vec{x})$$

$$\Delta\Phi(\vec{x}) = 4\pi G\rho(\vec{x})$$

weapon of choice: tree codes

particle approach

$$\vec{F}(\vec{x}_i) = -\sum_{i\neq j} \frac{Gm_i m_j}{(x_i - x_j)^3}(\vec{x}_i - \vec{x}_j)$$

grid approach ($\vec{x}_{i,j,k}$ =position of centre of grid cell (i,j,k))

$$\Delta\Phi(\vec{x}_{i,j,k}) = 4\pi G\rho(\vec{x}_{i,j,k})$$

$$\vec{F}(\vec{x}_{i,j,k}) = -m\nabla\Phi(\vec{x}_{i,j,k})$$

- Poisson's equation

$$\vec{F}(\vec{x}) = -m\nabla\Phi(\vec{x})$$

$$\Delta\Phi(\vec{x}) = 4\pi G\rho(\vec{x})$$

particle approach

$$\vec{F}(\vec{x}_i) = -\sum_{i\neq j} \frac{Gm_i m_j}{(x_i - x_j)^3}(\vec{x}_i - \vec{x}_j)$$

grid approach ($\vec{x}_{i,j,k}$ =position of centre of grid cell (i,j,k))

$$\Delta\Phi(\vec{x}_{i,j,k}) = 4\pi G\rho(\vec{x}_{i,j,k})$$

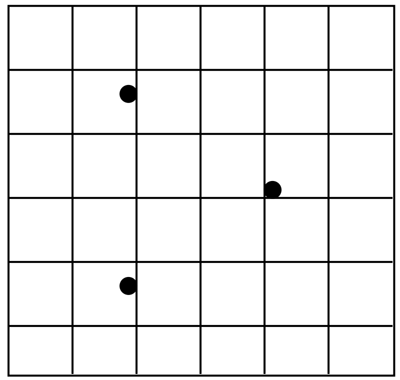
$$\vec{F}(\vec{x}_{i,j,k}) = -m\nabla\Phi(\vec{x}_{i,j,k})$$

weapon of choice: AMR codes

- Particle-Mesh (PM) method

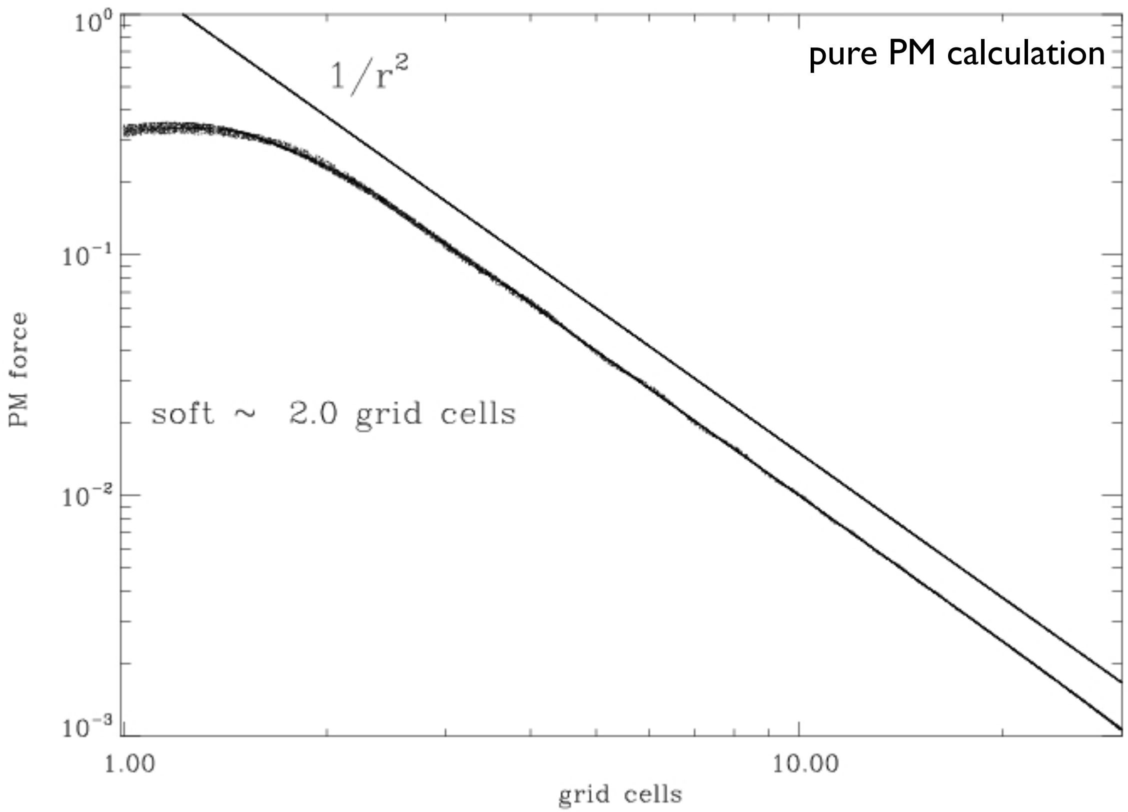
$$\Delta\Phi(\vec{g}_{k,l,m}) = 4\pi G\rho(\vec{g}_{k,l,m})$$

$$\vec{F}(\vec{g}_{k,l,m}) = -m\nabla\Phi(\vec{g}_{k,l,m})$$



1. calculate mass density on grid $\quad \vec{x}_i \rightarrow \rho(\vec{g}_{k,l,m})$
2. solve Poisson's equation on grid $\quad \Phi(\vec{g}_{k,l,m})$
3. differentiate potential to get forces $\quad \vec{F}(\vec{g}_{k,l,m})$
4. interpolate forces back to particles $\quad \vec{F}(\vec{g}_{k,l,m}) \rightarrow \vec{F}(\vec{x}_i)$

- numerically integrate Poisson's equation



- numerically integrate Poisson's equation

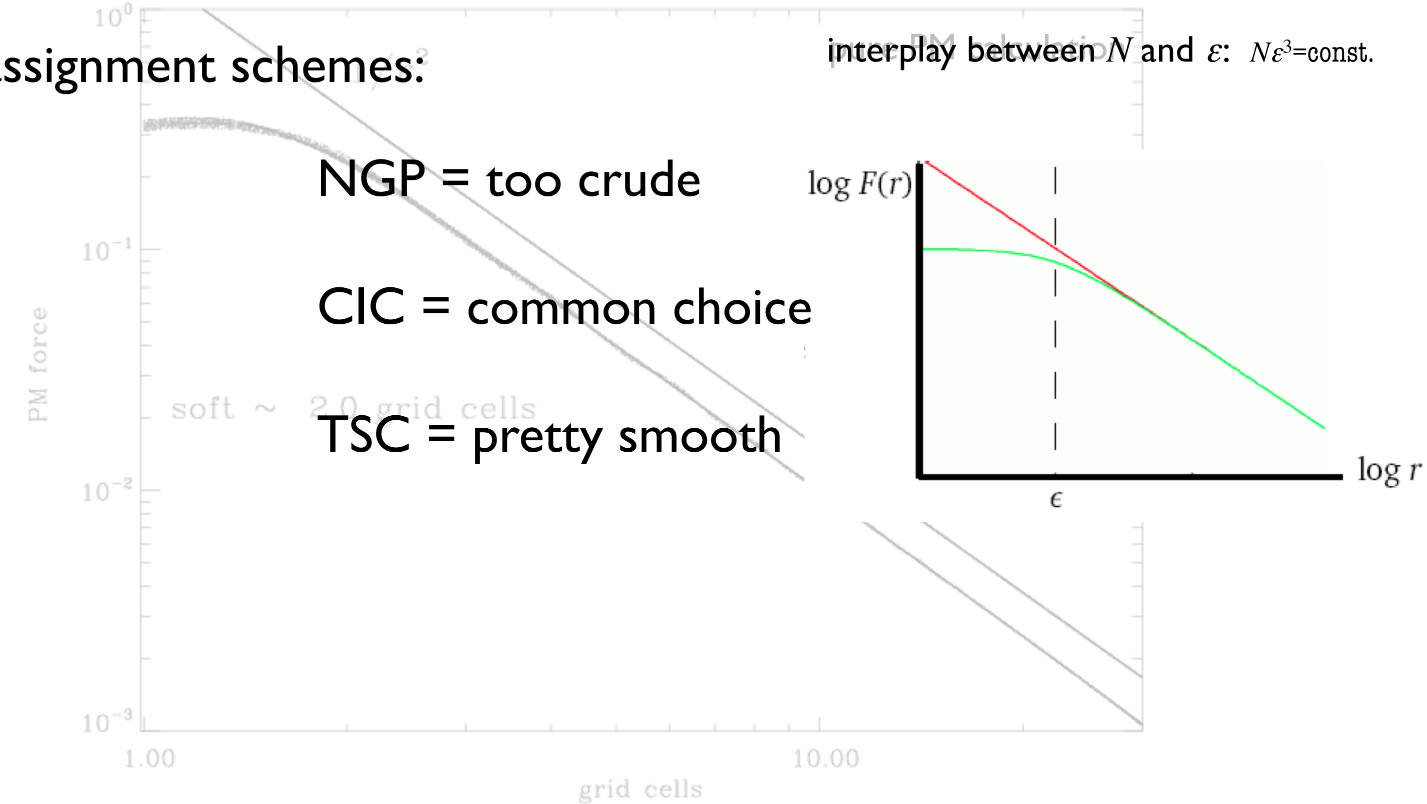
- density assignment schemes:

interplay between N and ε: $N\varepsilon^3$=const.

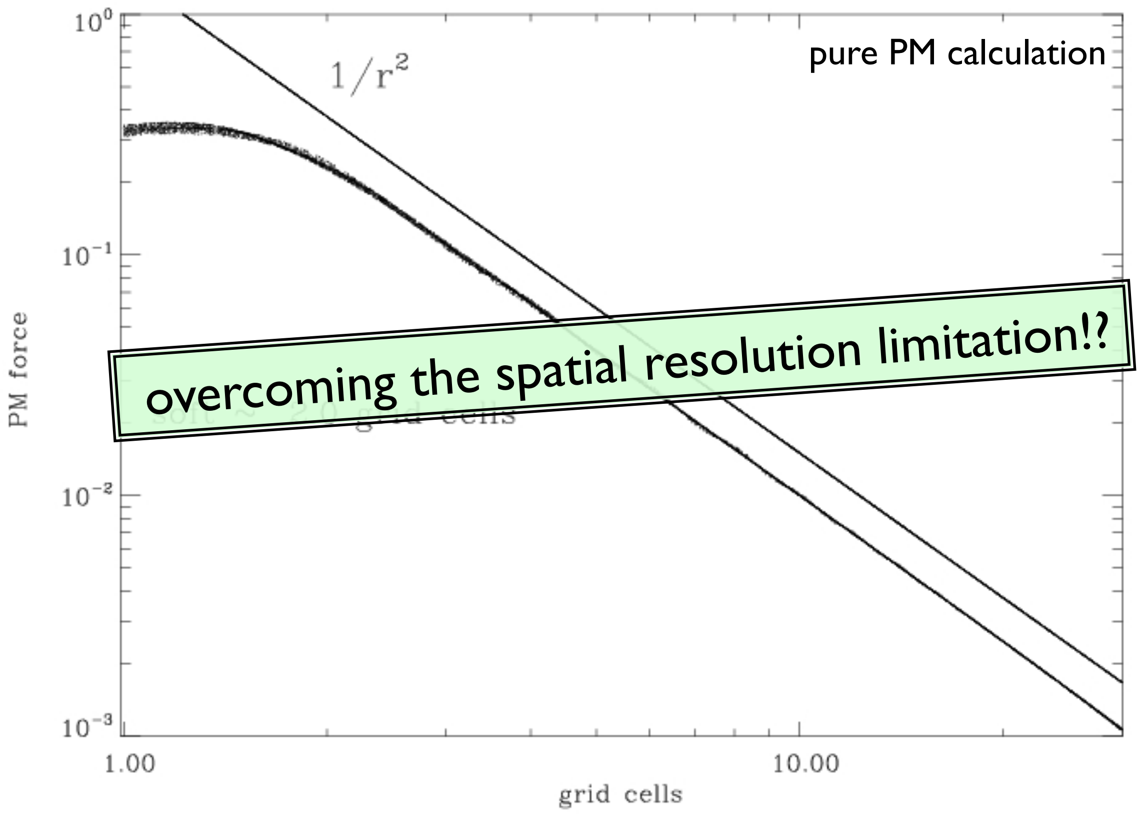
NGP = too crude

CIC = common choice

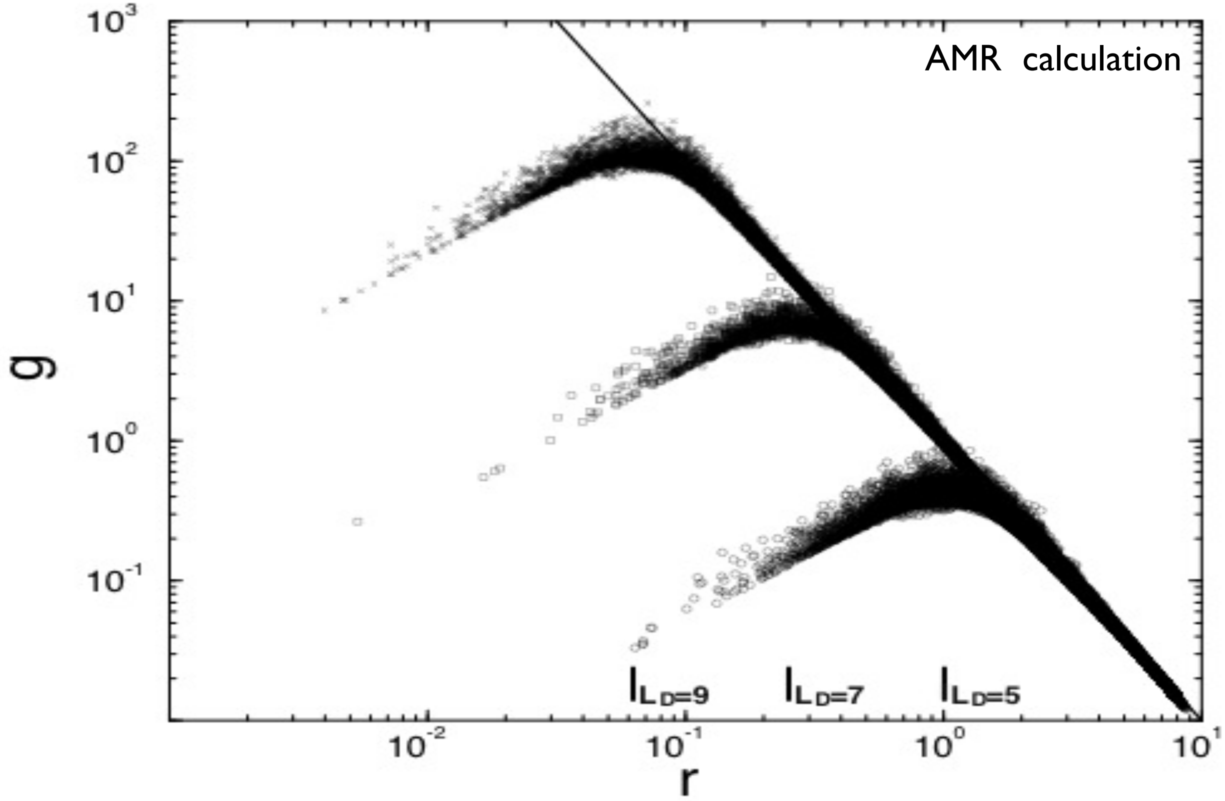
TSC = pretty smooth



- numerically integrate Poisson's equation

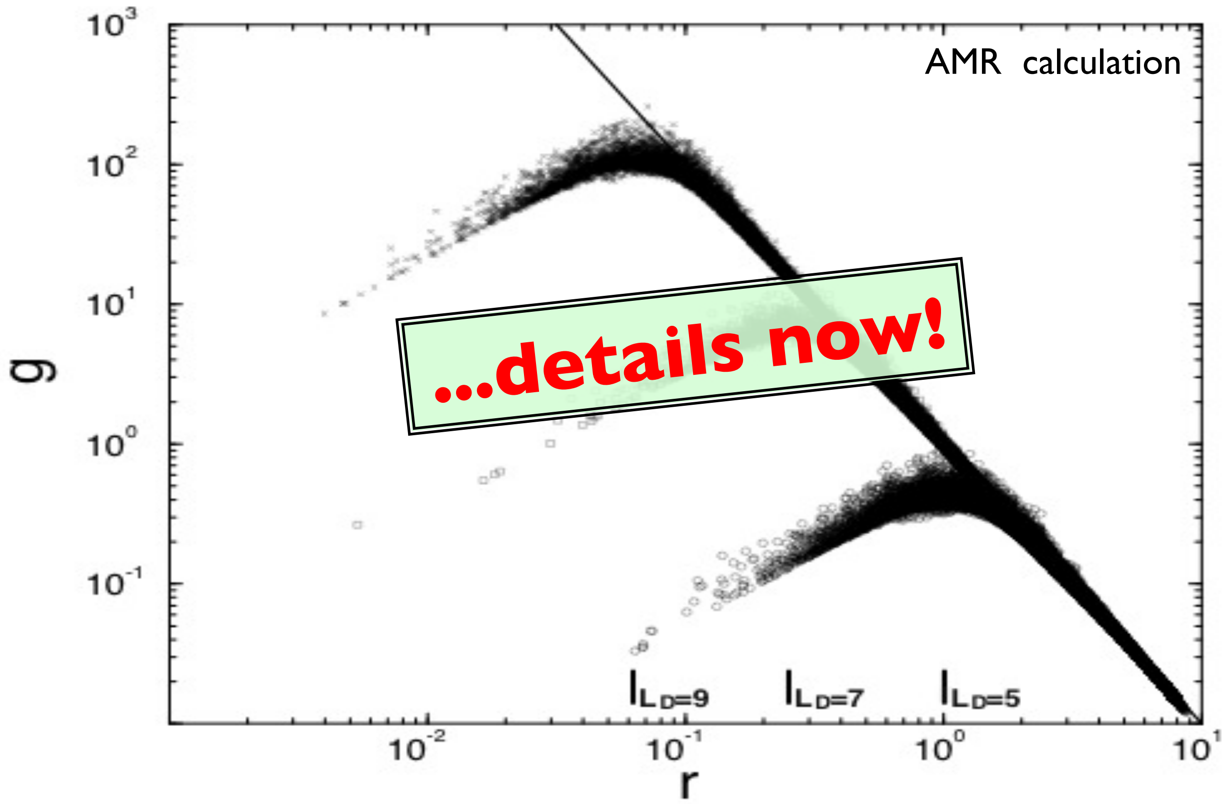


- numerically integrate Poisson's equation



Yahagi & Yoshi (2001)

- numerically integrate Poisson's equation



Yahagi & Yoshi (2001)

- mesh refinements
- adaptive mesh refinement
- adaptive mesh refinement for N-body codes
- handling irregular grids
- adaptive leap-frog integration

- **mesh refinements**
- adaptive mesh refinement
- adaptive mesh refinement for N-body codes
- handling irregular grids
- adaptive leap-frog integration

- types of mesh refinement

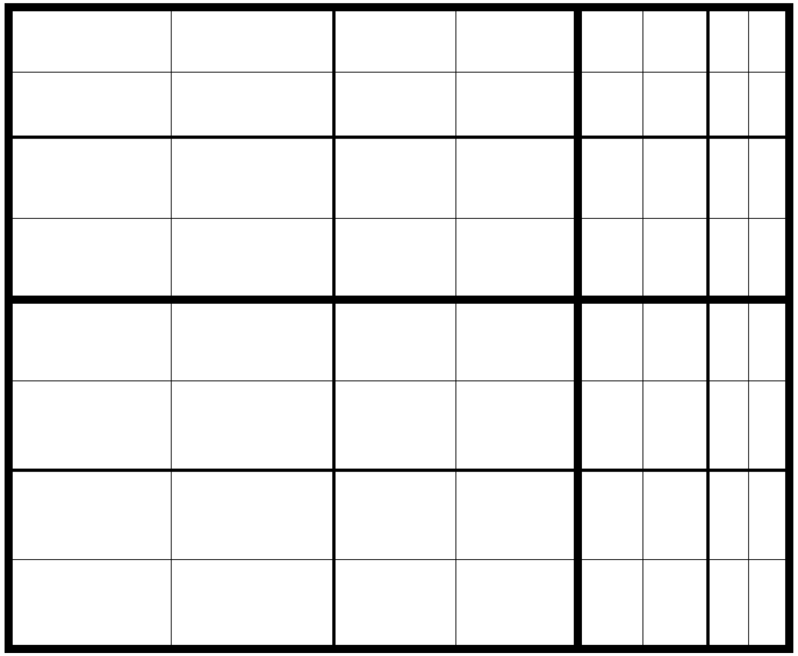
  - *r* refinement: move or stretch the mesh
  - *p* refinement: adjust the order of the method
  - *h* refinement: change the mesh spacing

- types of mesh refinement – *r* refinement

  - non-uniform mesh *(refined region is known)*

    = advantages:
    − simple to implement

    = disadvantages:
    − difference expression for non-constant zone spacing

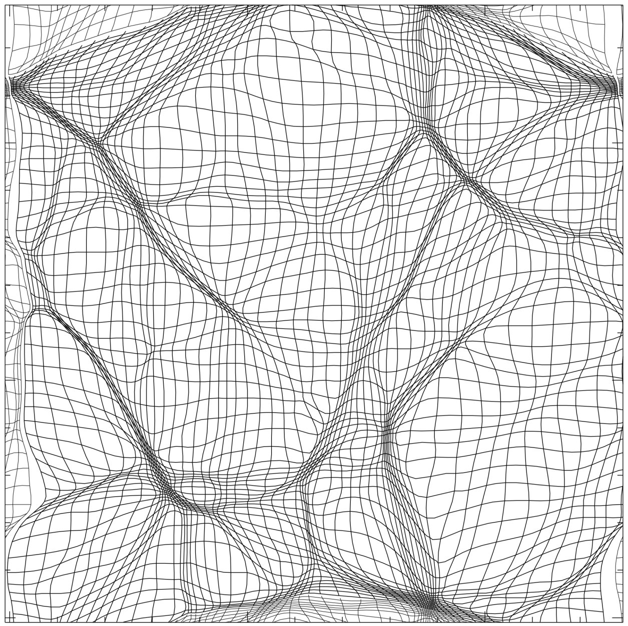


COSMOS code (Ricker 2000)

- types of mesh refinement – *r* refinement
  - Lagragian mesh *(mesh is tied to fluid)*

    = advantages:
      - constant mass resolution
      - sharp resolution of contacts

    = disadvantages:
      - grid stretching causes numerical dissipation
      - grid tangling in rotational flows



MMH code (Pen 1998)

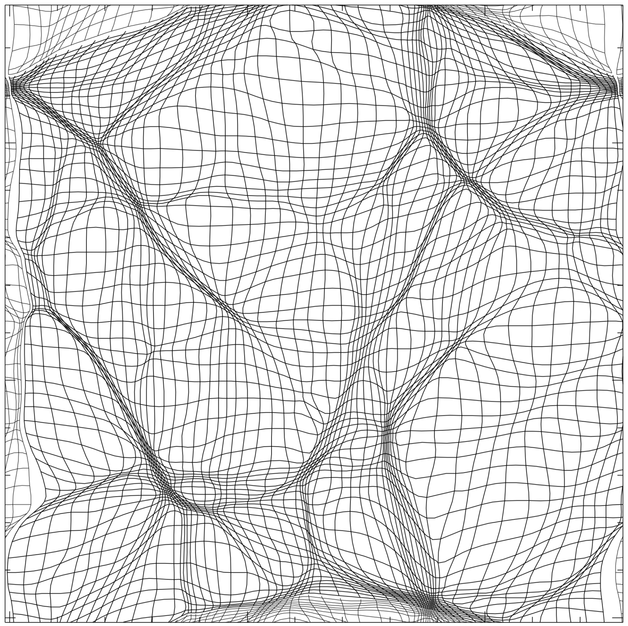
- types of mesh refinement – *r* refinement

  - Lagragian mesh *(mesh is tied to fluid)*

    = advantages:
      - constant mass resolution
      - sharp resolution of contacts

    = disadvantages:
      - grid stretching causes numerical dissipation
      - grid tangling in rotational flows

usually used only in 1D (e.g. stellar evolution codes)



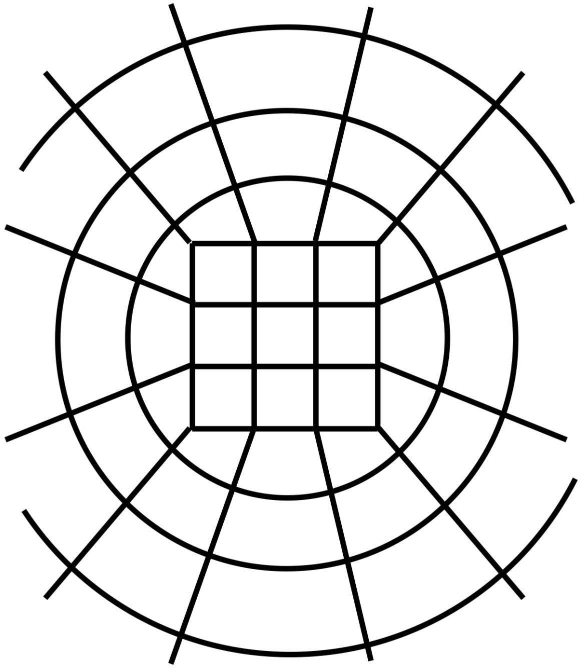
MMH code (Pen 1998)

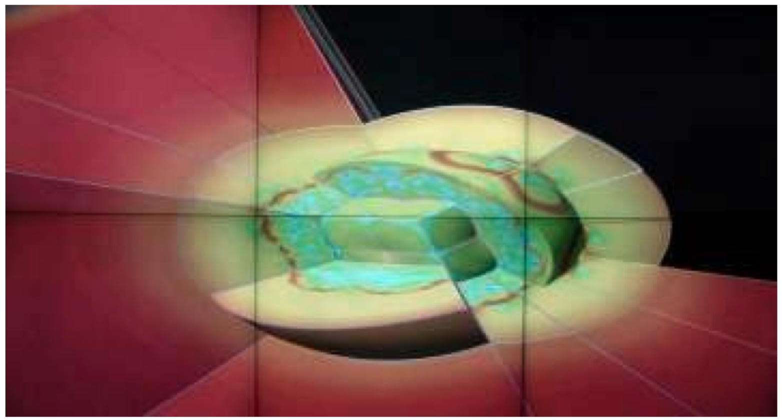
- types of mesh refinement – *r* refinement

  - arbitrary Lagrangian-Eulerion mesh *(mesh moves arbitrarily fluid)*

    = advantages:
    - Lagrangian mesh where flow is irrotational
    - Eulerian where mesh distortion is problematic

    = disadvantages:
    - difficult to handle...





DJEHUTY code (Dearborn et al. 2002)

- types of mesh refinement – *p* refinement

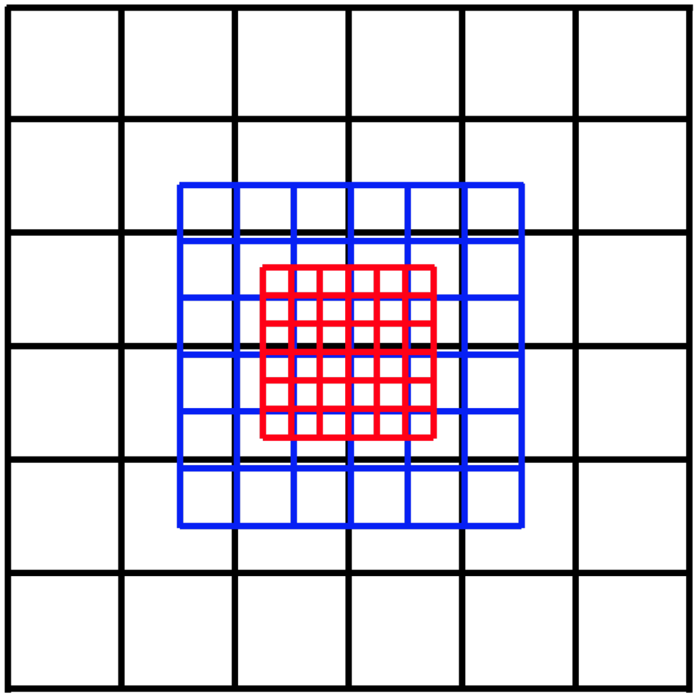
not in this course...

- types of mesh refinement – *h* refinement

  • nested grids *(static meshes with different resolutions)*

  = advantages:
  – easy to handle boundaries between meshes

  = disadvantages:
  – refined region should not move



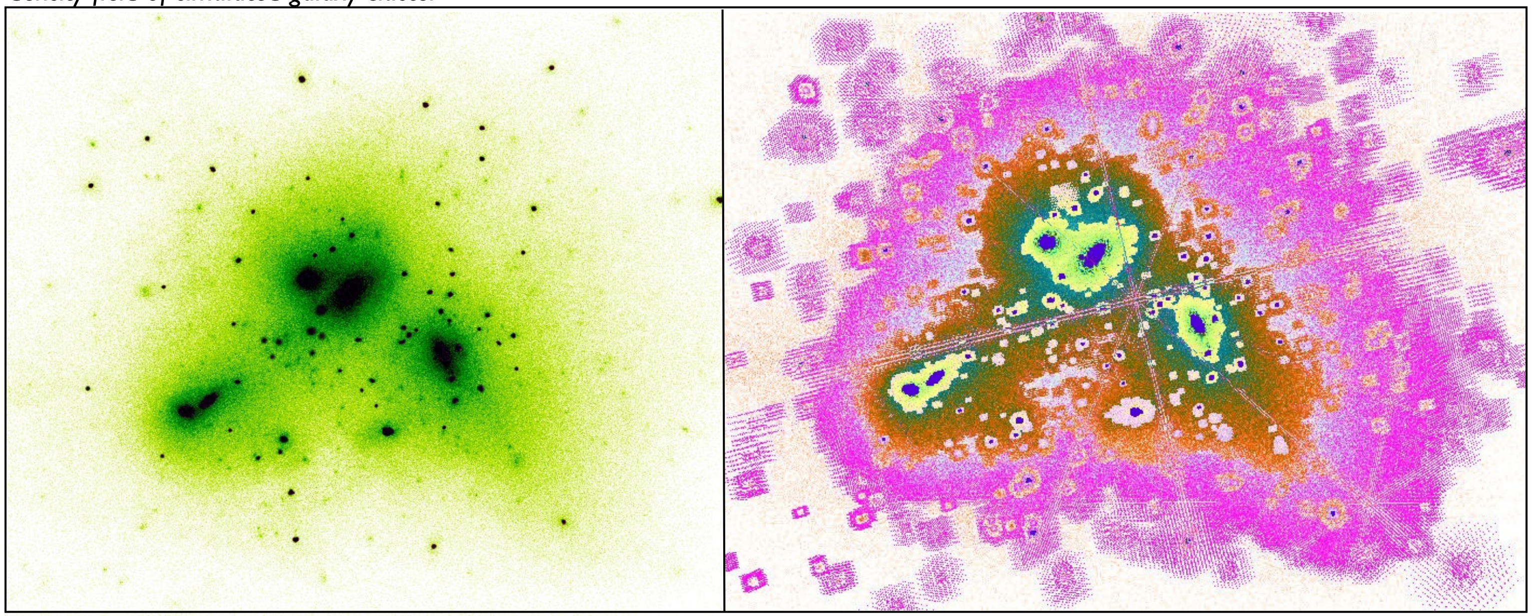
- types of mesh refinement – *h* refinement

  - adaptive mesh refinement *(refined patches are created and destroyed as needed)*

    - advantages:
      - fully flexible to problem

    - disadvantages:
      - serious book-keeping for grid hierarchy

density field of simulated galaxy cluster

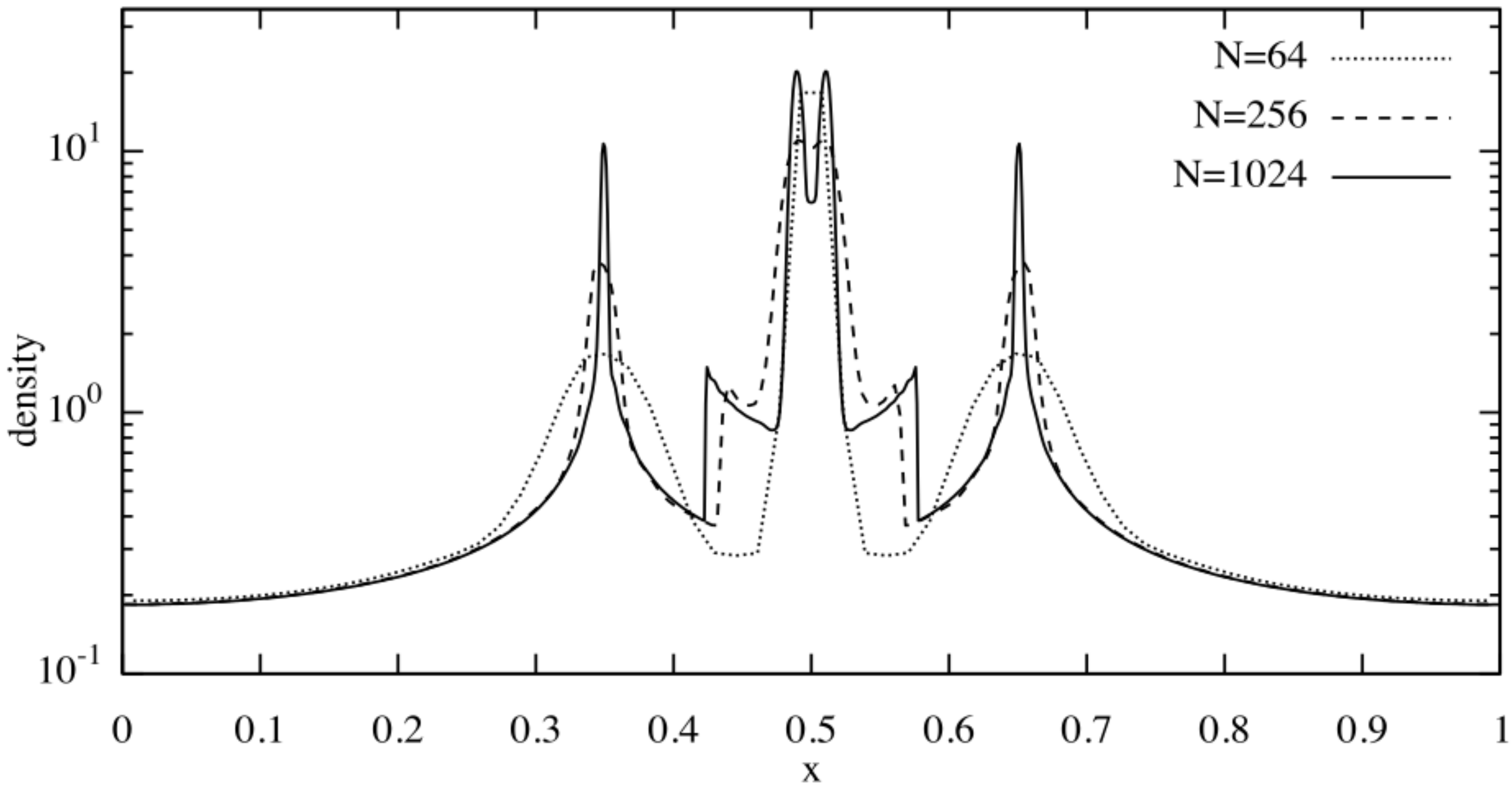


AMIGA code (Doumler & Knebe 2010)

adaptive grid hierarchy

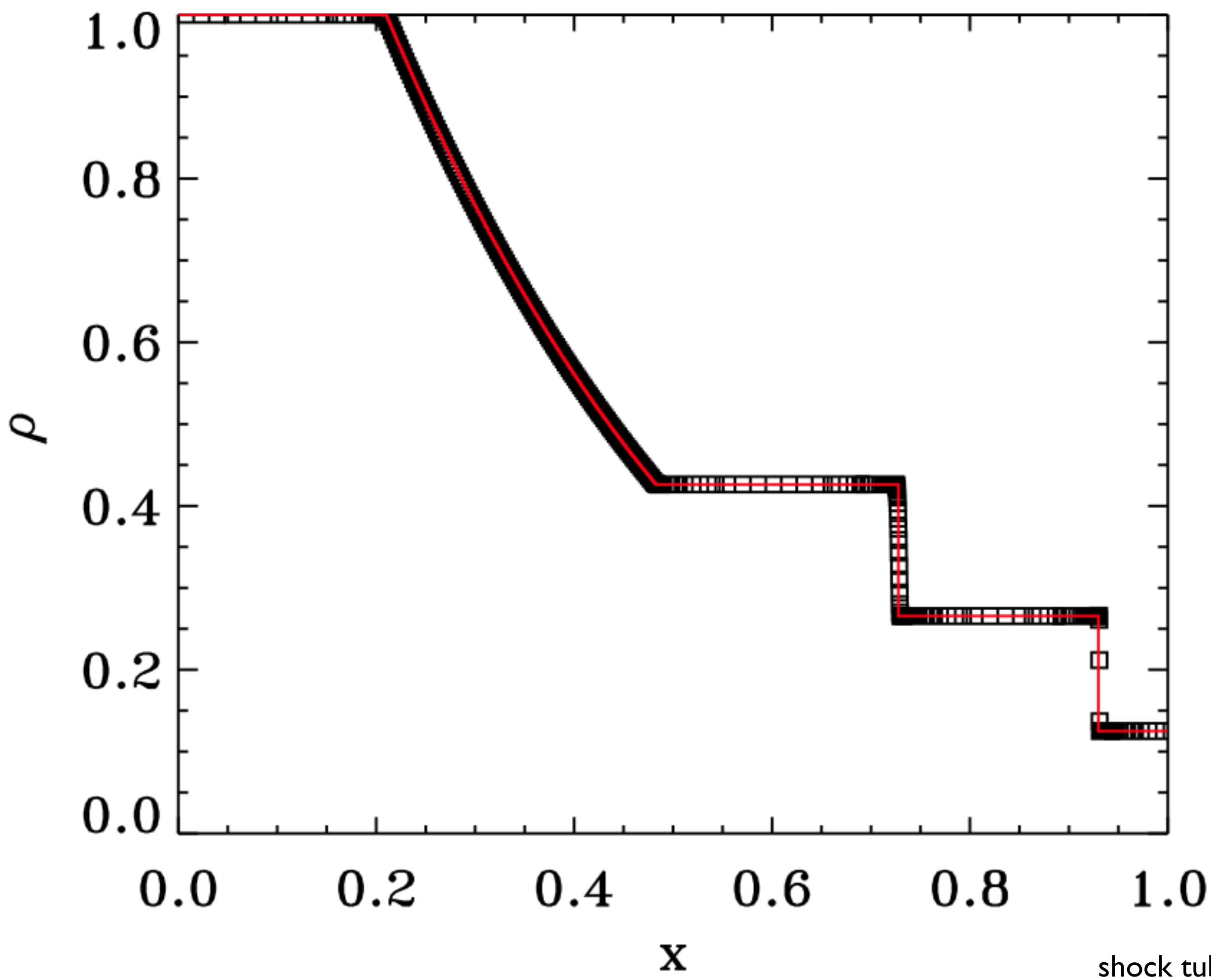
- mesh refinements
- **adaptive mesh refinement**
- adaptive mesh refinement for N-body codes
- handling irregular grids
- adaptive leap-frog integration

- adaptive mesh refinement – improvements using finer grids



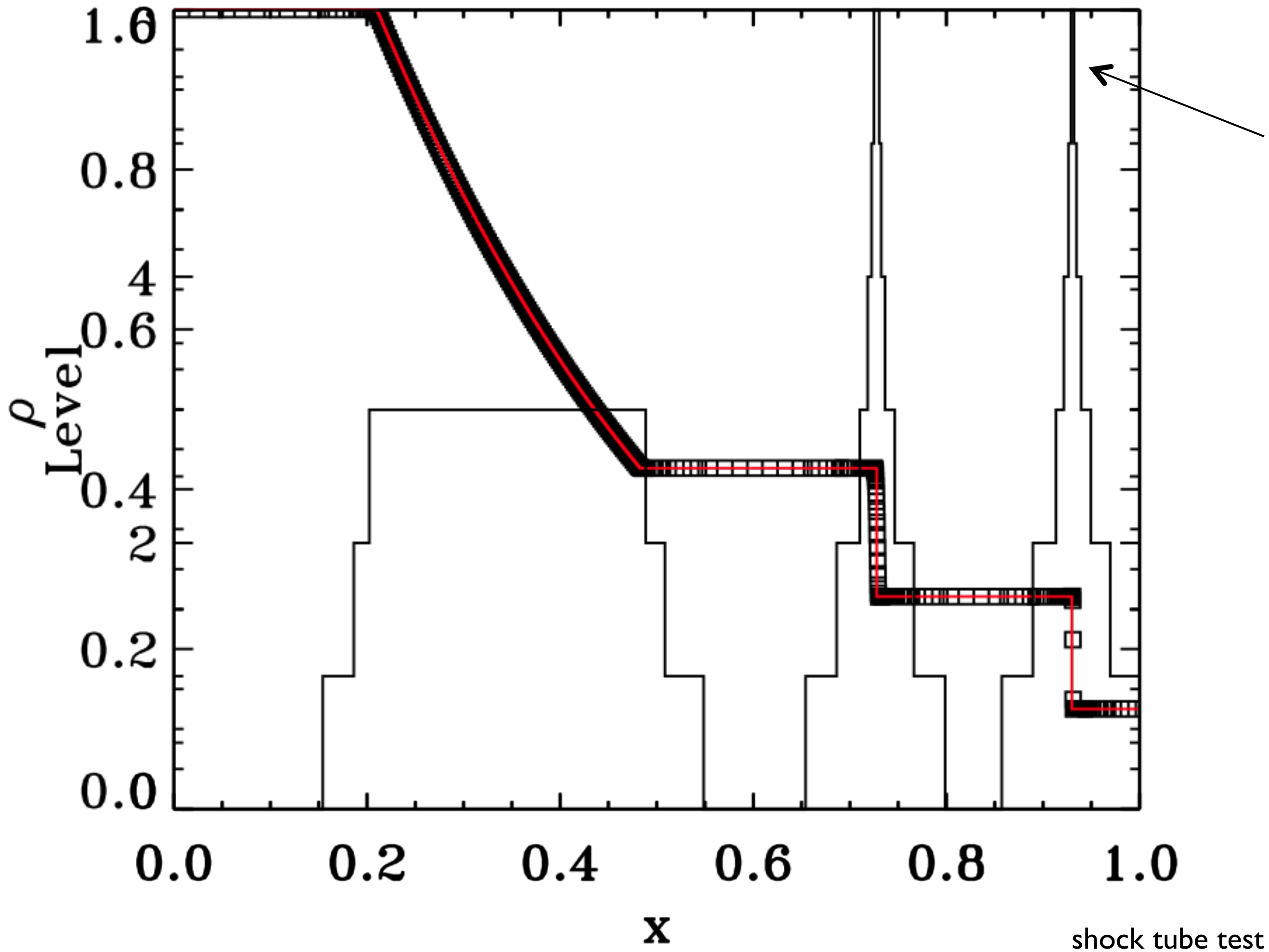
double pancake test (Doumler & Knebe 2010)

- adaptive mesh refinement – improvements using finer grids



shock tube test (Teyssier 2002)

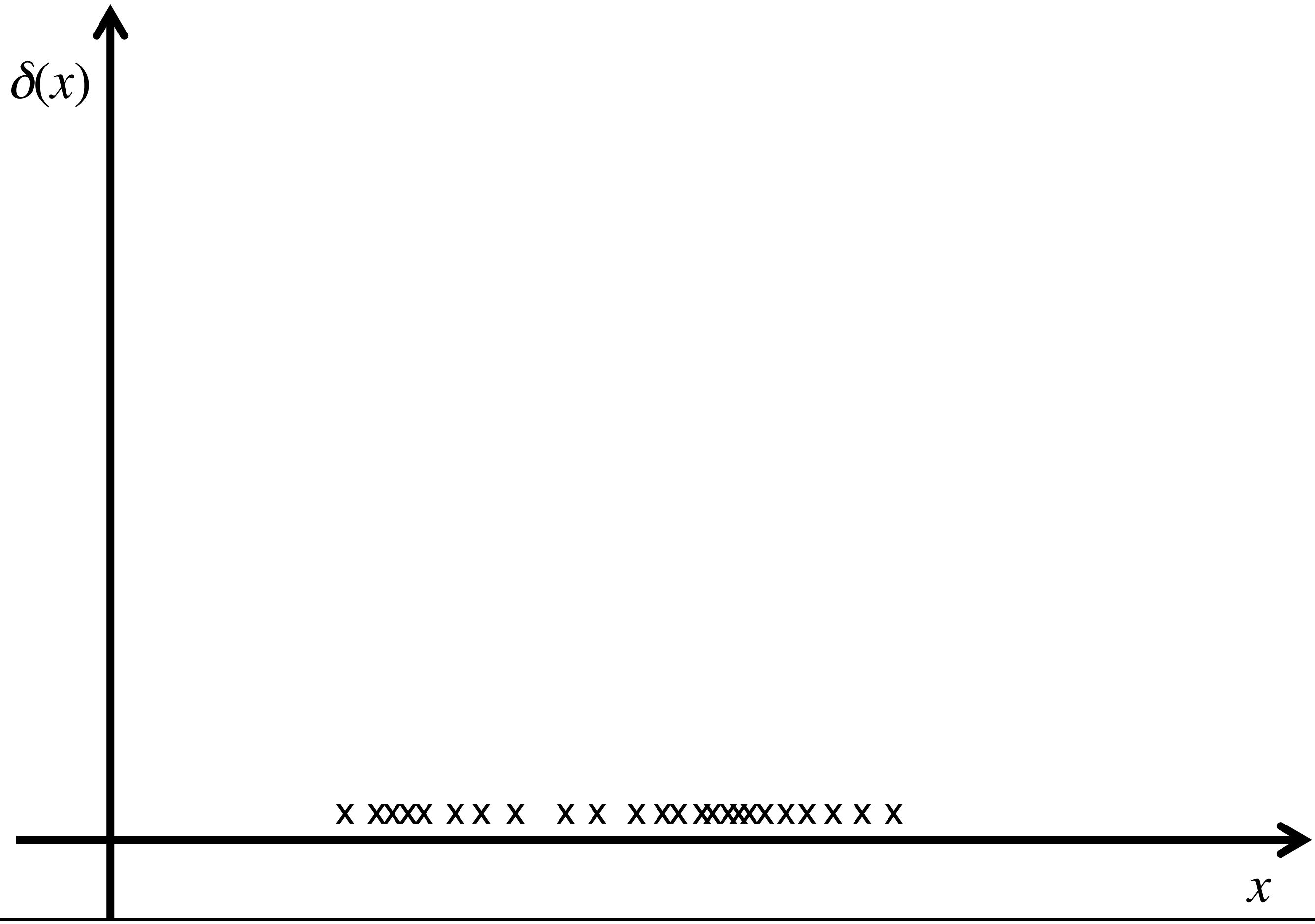
- adaptive mesh refinement – improvements using finer grids



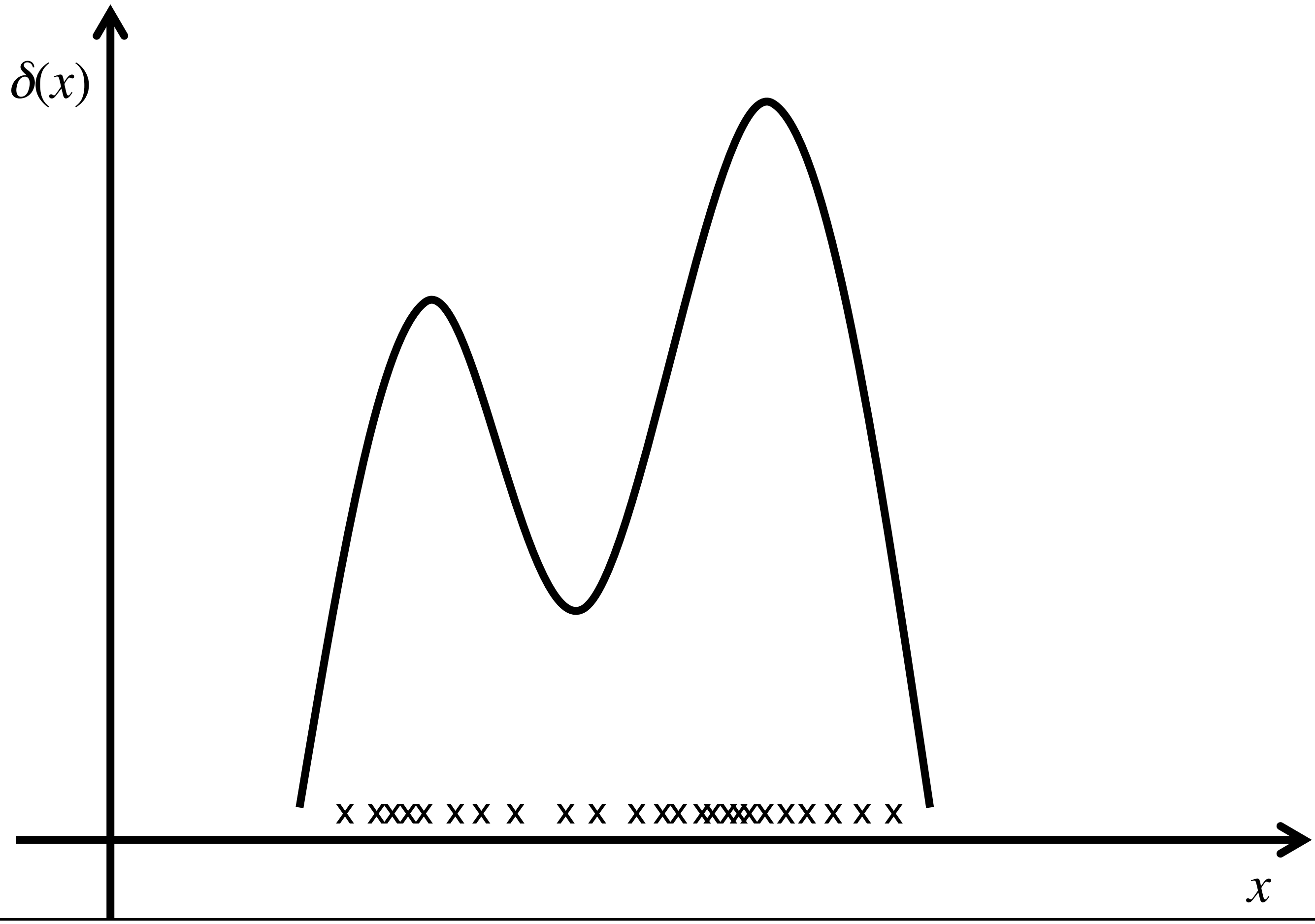
shock tube test (Teyssier 2002)

- adaptive mesh refinement – refinement criterion
    - density
    - truncation error
    - physics

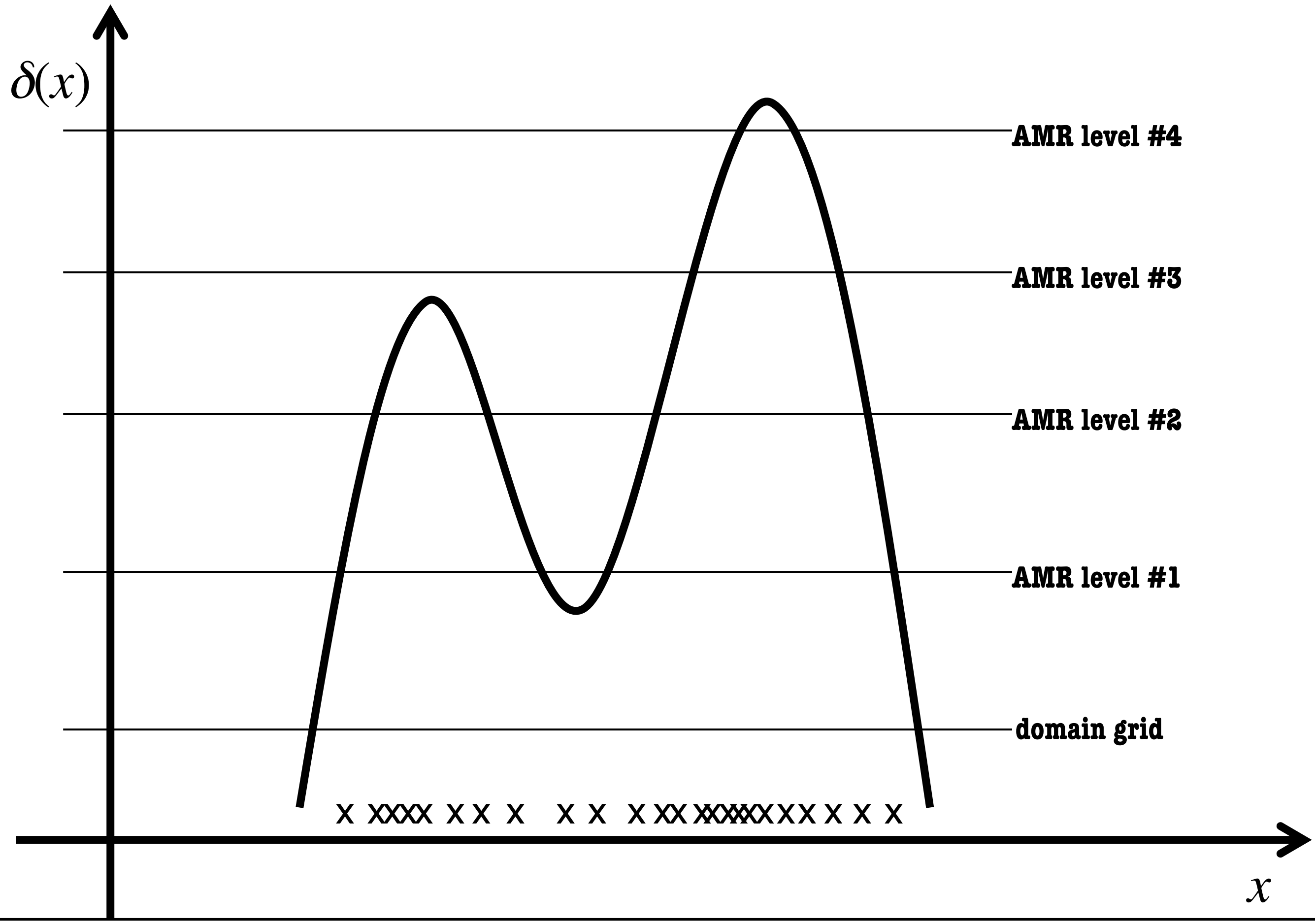
- adaptive mesh refinement – refinement criterion
    - density – 1D density distribution



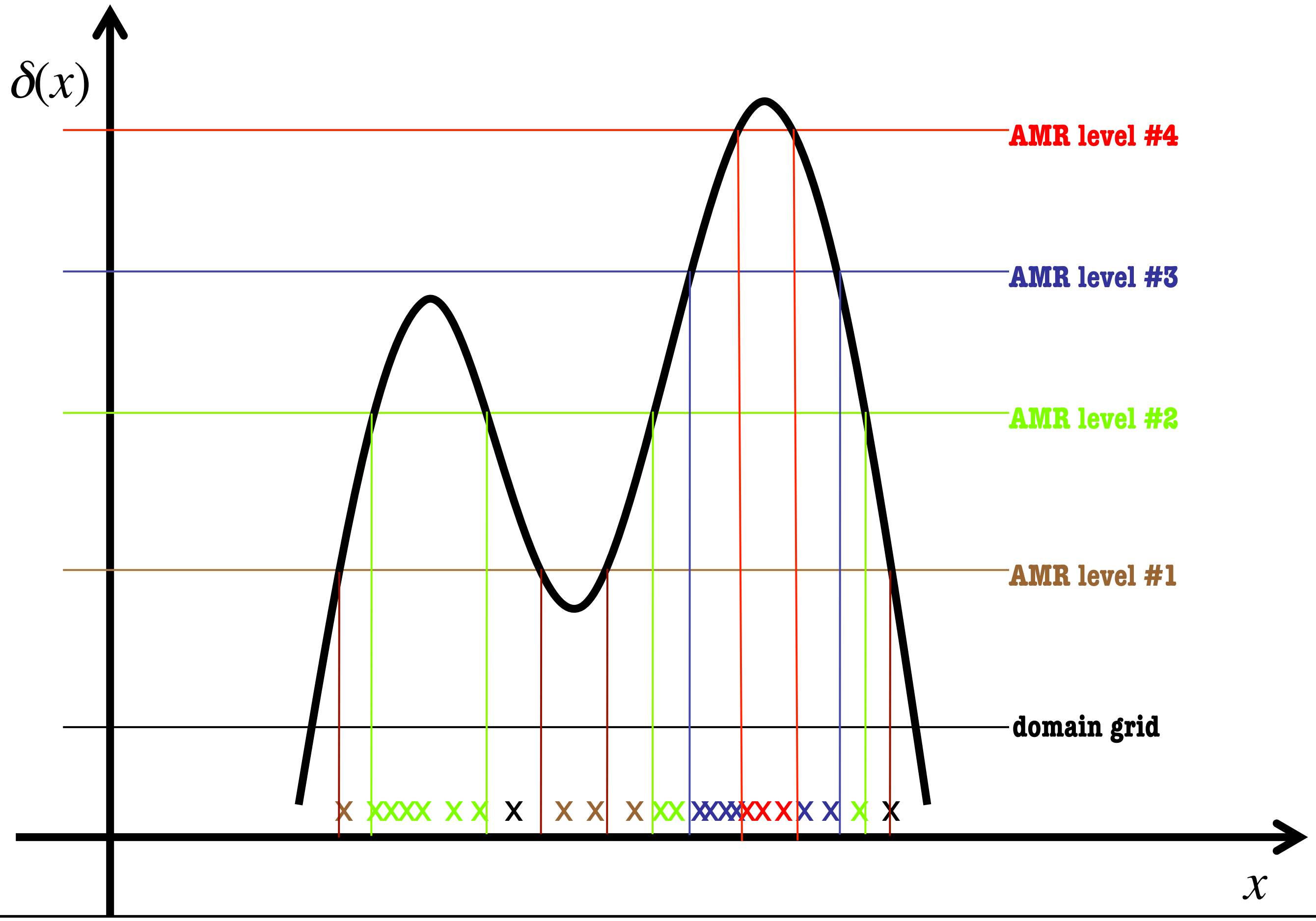
- adaptive mesh refinement – refinement criterion
    - density – 1D density distribution



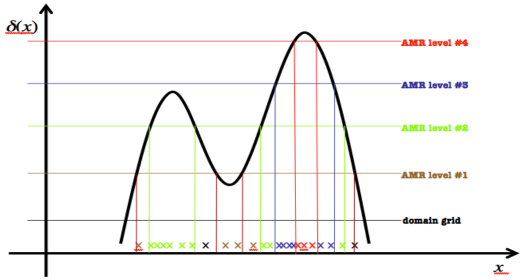
- adaptive mesh refinement – refinement criterion
  - density – 1D density distribution



- adaptive mesh refinement – refinement criterion
  - density – 1D density distribution



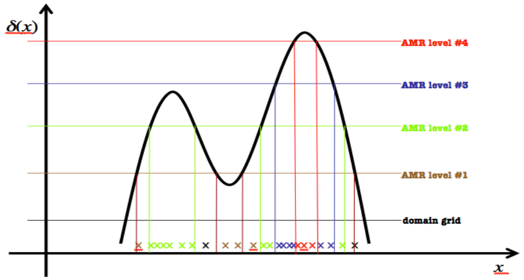
- adaptive mesh refinement – refinement criterion
  - density:
    - refine regions of high density



  - truncation error:
  - physics:

- adaptive mesh refinement – refinement criterion

  - density:

    – refine regions of high density

  - truncation error:

    – refine regions of large truncation errors

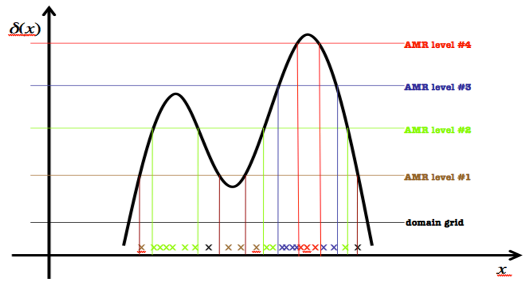
$$R^i_{k,l,m} = \Delta\Phi^i_{k,l,m} - \rho_{k,l,m} \leq \varepsilon \mathrm{T}_{k,l,m} \quad \text{with} \quad \mathrm{T}_{k,l,m} = \mathcal{P}\left[\Delta\left(\mathcal{R}\Phi^i_{k,l,m}\right)\right] - \left(\Delta\Phi^i_{k,l,m}\right)$$

  - physics:

- adaptive mesh refinement – refinement criterion

  - density:

    - refine regions of high density



  - truncation error:

    - refine regions of large truncation errors

$$R^i_{k,l,m} = \Delta\Phi^i_{k,l,m} - \rho_{k,l,m} \leq \varepsilon \mathrm{T}_{k,l,m} \qquad \text{with} \qquad \mathrm{T}_{k,l,m} = \mathcal{P}\left[\Delta\left(\mathcal{R}\Phi^i_{k,l,m}\right)\right] - \left(\Delta\Phi^i_{k,l,m}\right)$$

  - physics:
    - compare grid spacing against local critical wavelength

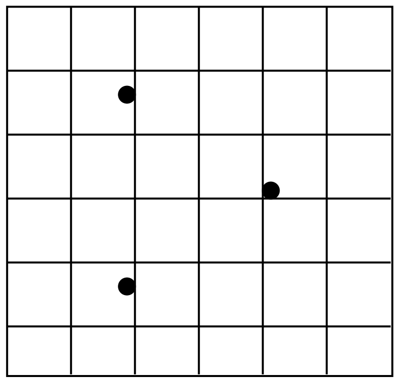
$$\Delta x < \varepsilon\,\lambda \qquad \text{with} \qquad \lambda = c_s\sqrt{\frac{\pi}{G\rho}}$$

- mesh refinements
- adaptive mesh refinement
- **adaptive mesh refinement for N-body codes**
- handling irregular grids
- adaptive leap-frog integration

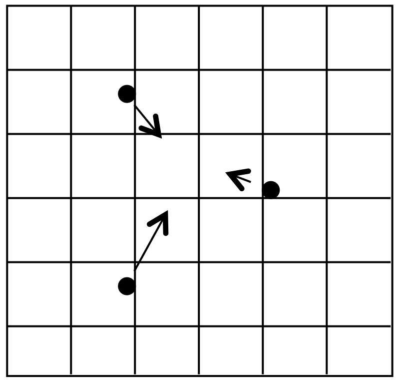
- mesh refinements
- adaptive mesh refinement
- **adaptive mesh refinement for N-body codes**
  - gravity
  - generating refinements
  - density assignment
  - solving Poisson's equation
- handling irregular grids
- adaptive leap-frog integration

- mesh refinements
- adaptive mesh refinement
- **adaptive mesh refinement for N-body codes**
    - ***gravity***
    - generating refinements
    - density assignment
    - solving Poisson's equation
- handling irregular grids
- adaptive leap-frog integration

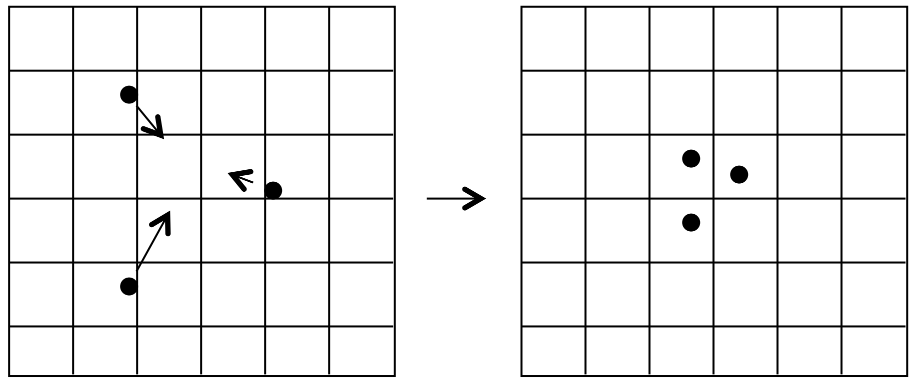
- gravity tends to clump matter together...



- gravity tends to clump matter together...

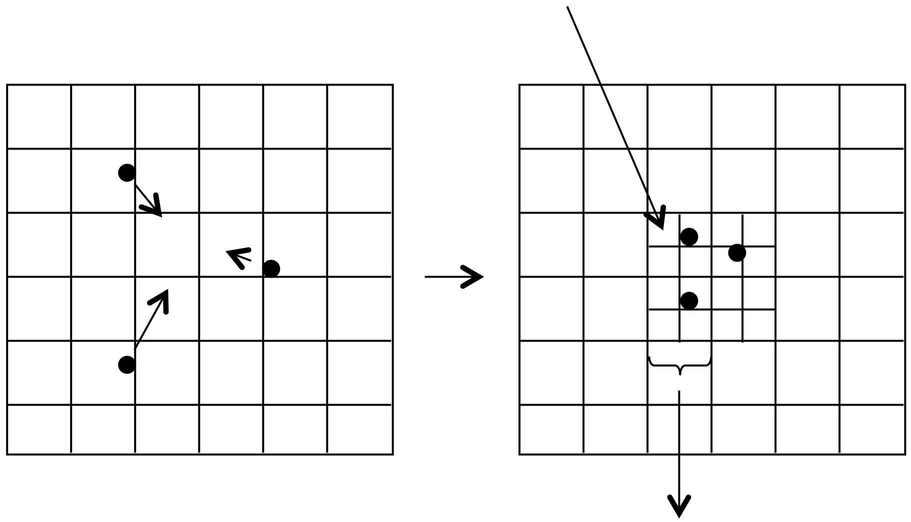


- gravity tends to clump matter together...



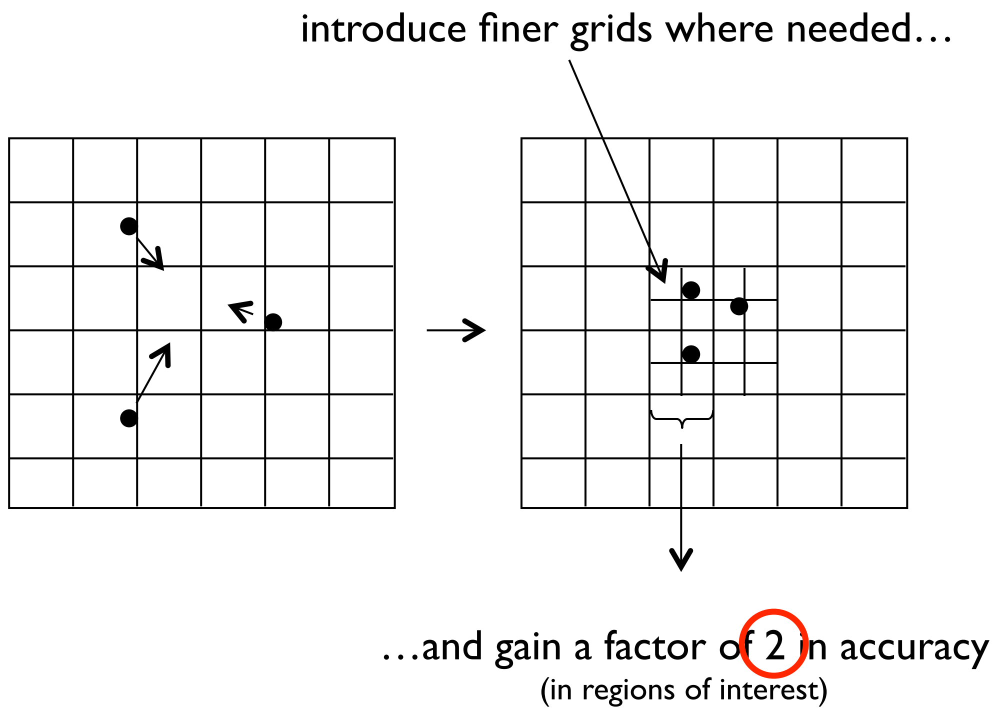
- gravity tends to clump matter together...

introduce finer grids where needed…



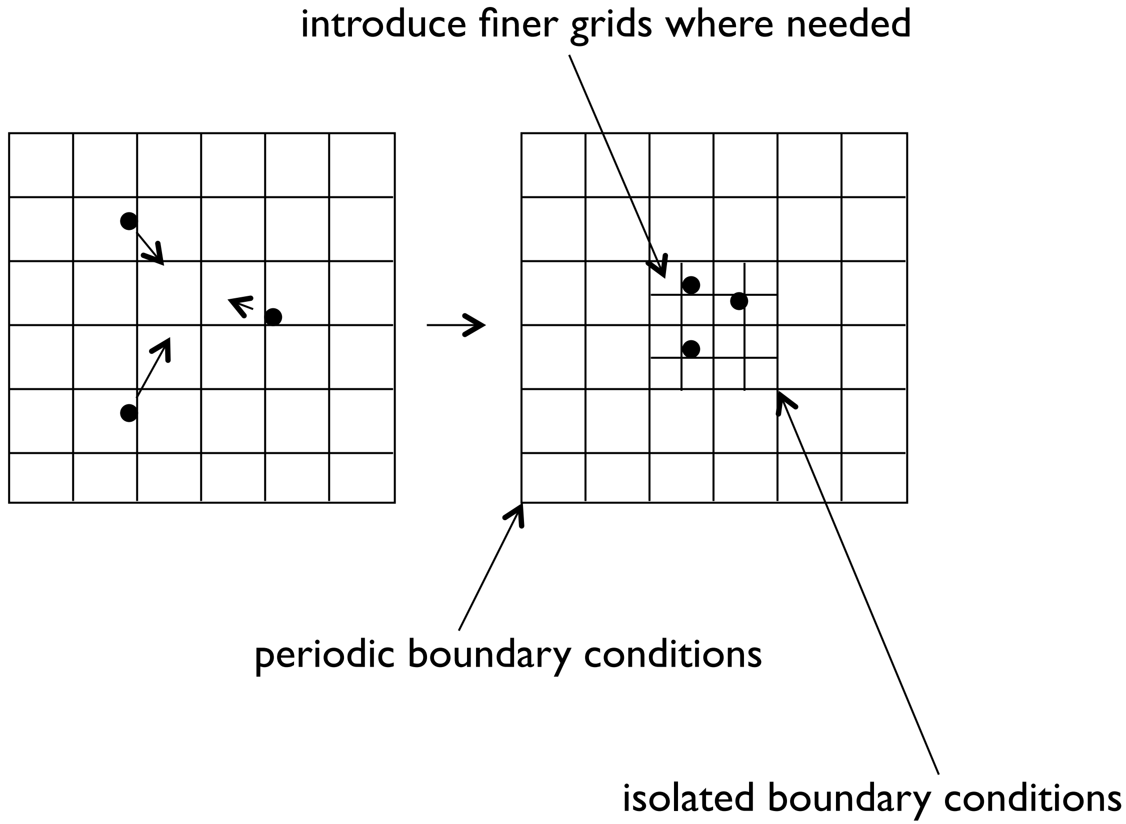
…and gain a factor of 2 in accuracy
(in regions of interest)

- gravity tends to clump matter together...

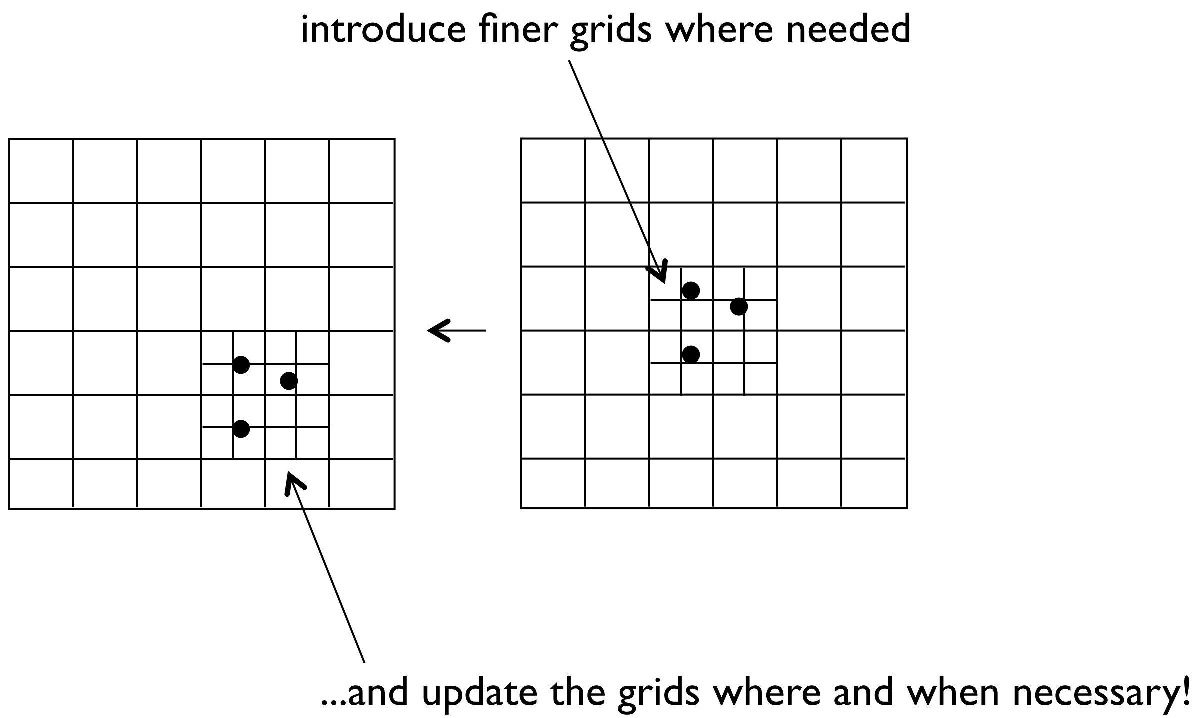


factor 2 not mandatory, but most common choice...

- gravity tends to clump matter together...



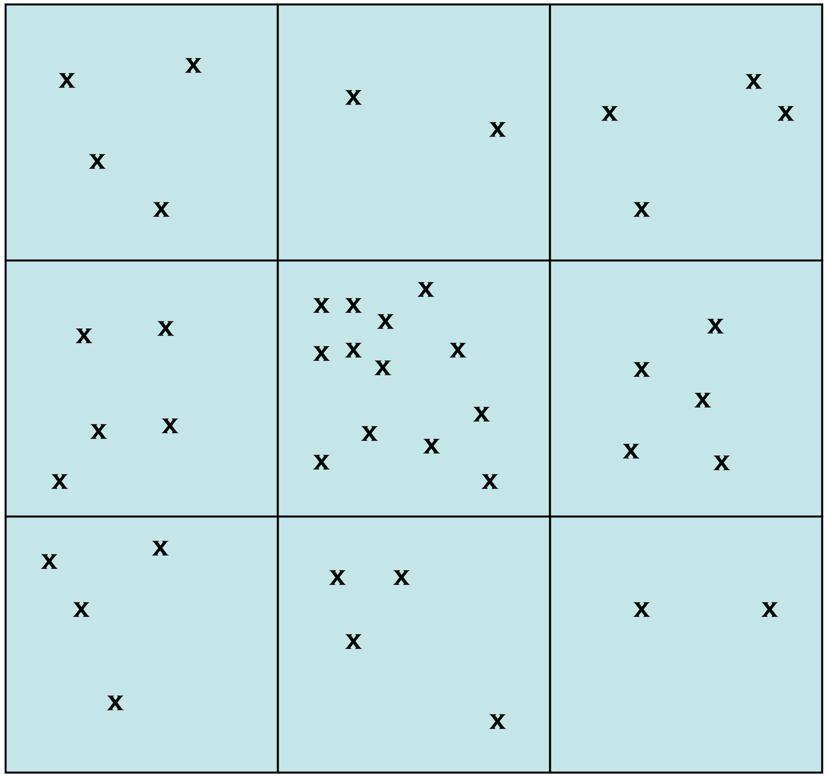
- gravity tends to clump matter together...



- mesh refinements
- adaptive mesh refinement
- **adaptive mesh refinement for N-body codes**
  - gravity
  - ***generating refinements***
  - density assignment
  - solving Poisson's equation
- handling irregular grids
- adaptive leap-frog integration

- generating refinements
  - N-body simulations:

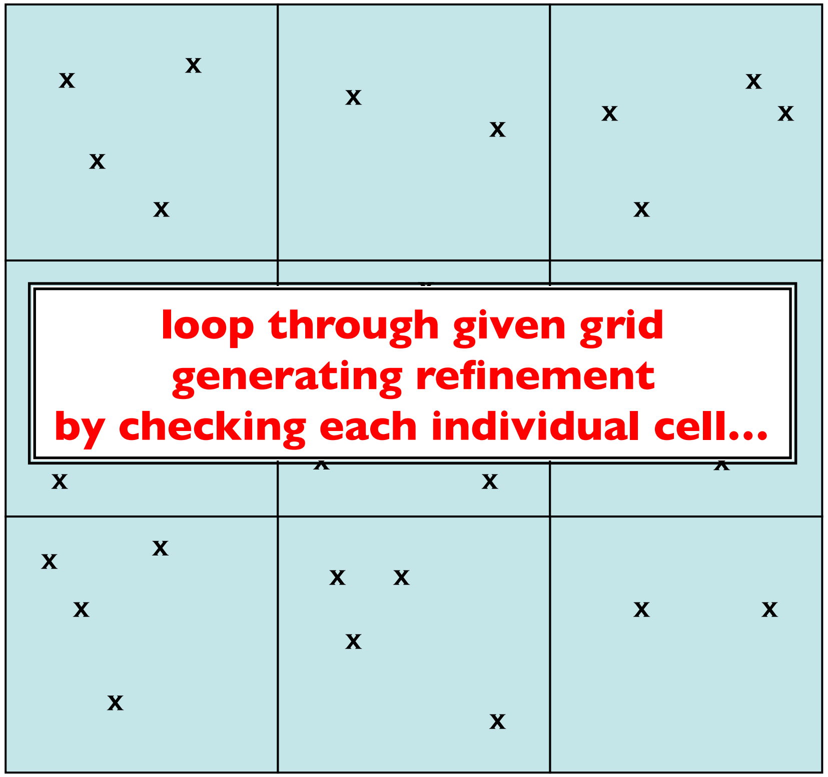
number of particles per cell



refinement criterion: 6 particles/cell

- generating refinements
  - N-body simulations:

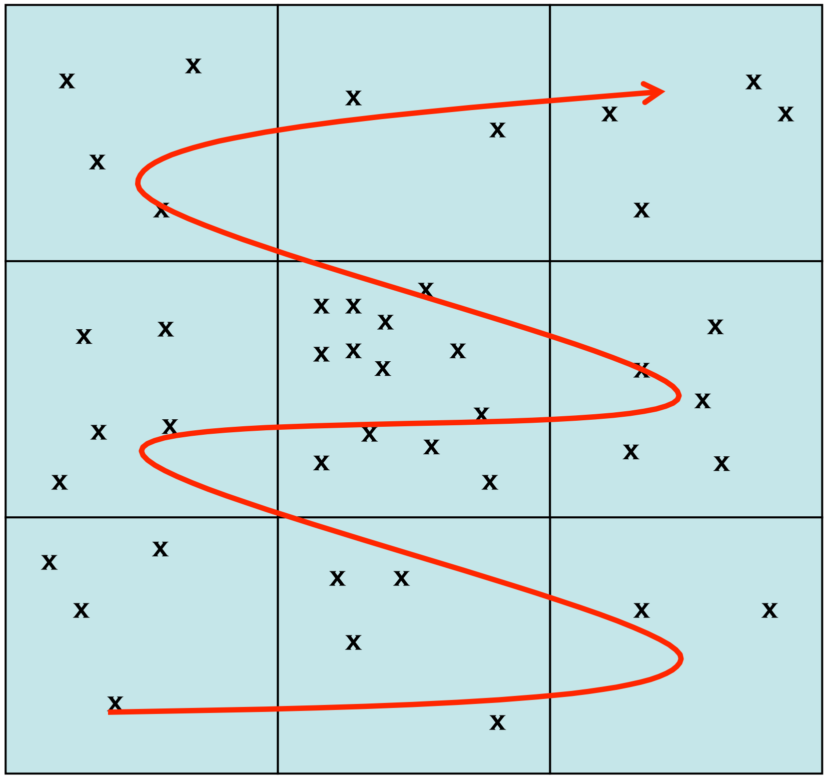
number of particles per cell



refinement criterion: 6 particles/cell

- generating refinements
  - N-body simulations:

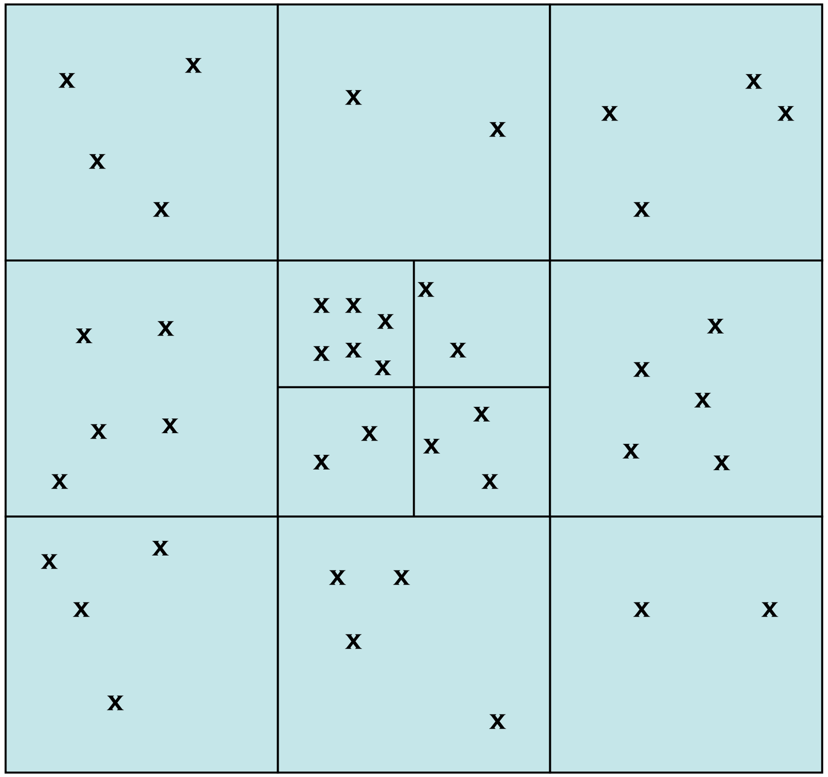
number of particles per cell



refinement criterion: 6 particles/cell

- generating refinements
  - *N*-body simulations:

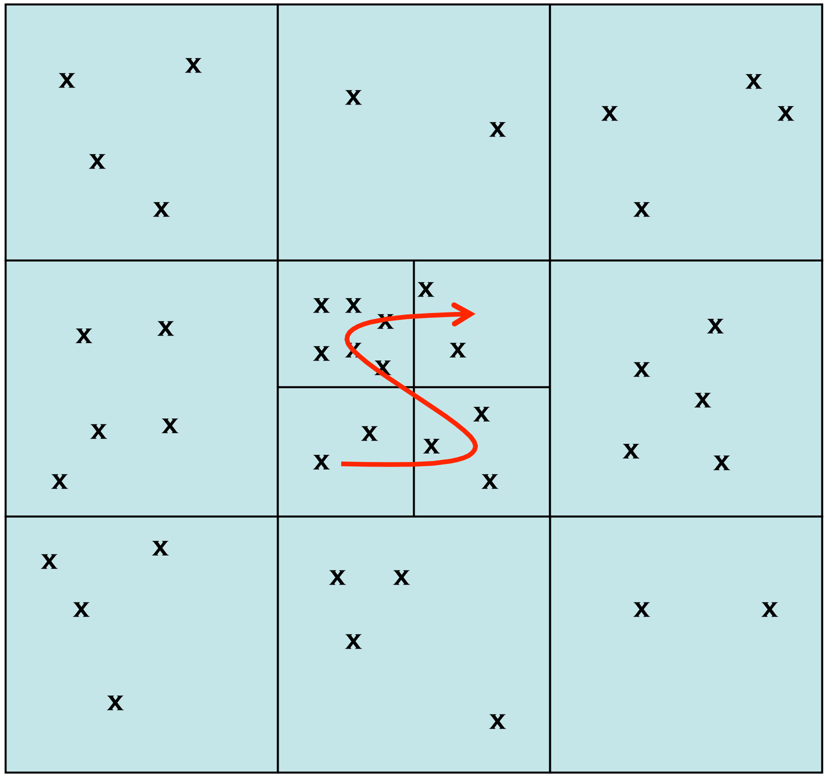
number of particles per cell



refinement criterion: 6 particles/cell

- generating refinements
  - N-body simulations:

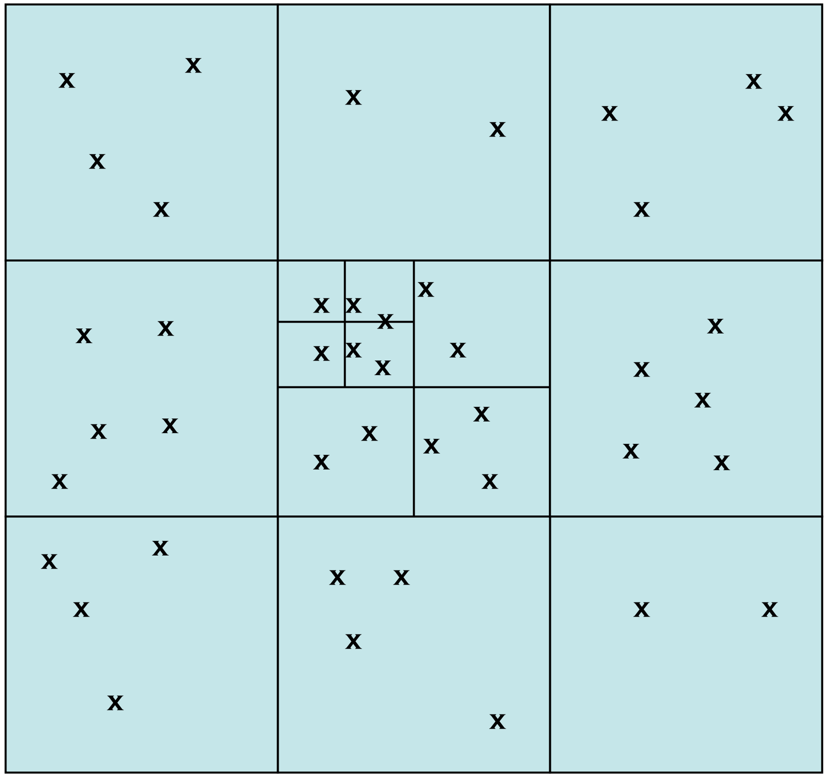
number of particles per cell



refinement criterion: 6 particles/cell

- generating refinements
  - N-body simulations:

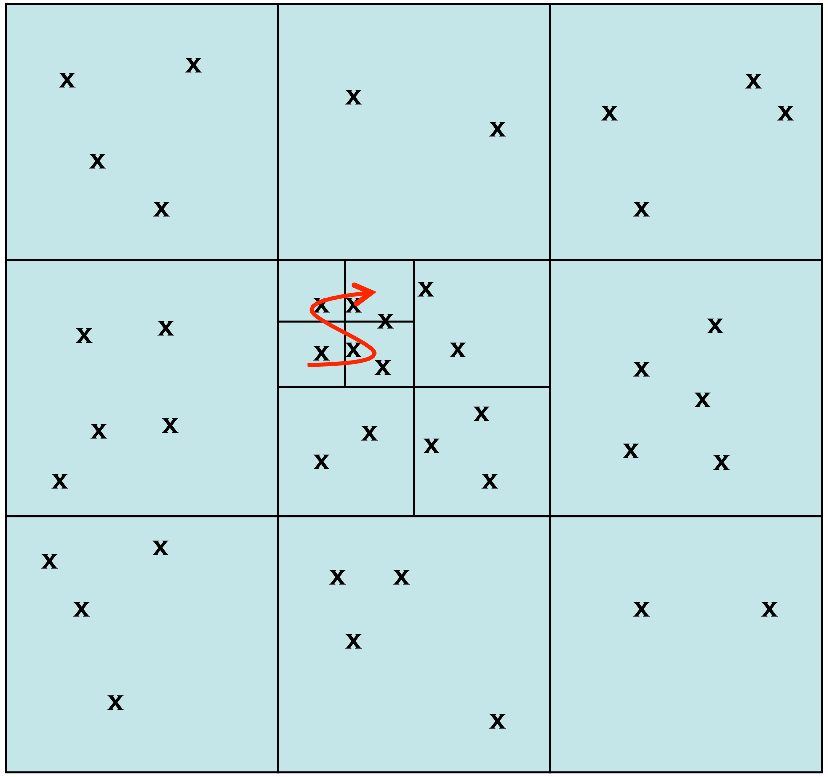
number of particles per cell



refinement criterion: 6 particles/cell

- generating refinements
  - N-body simulations:

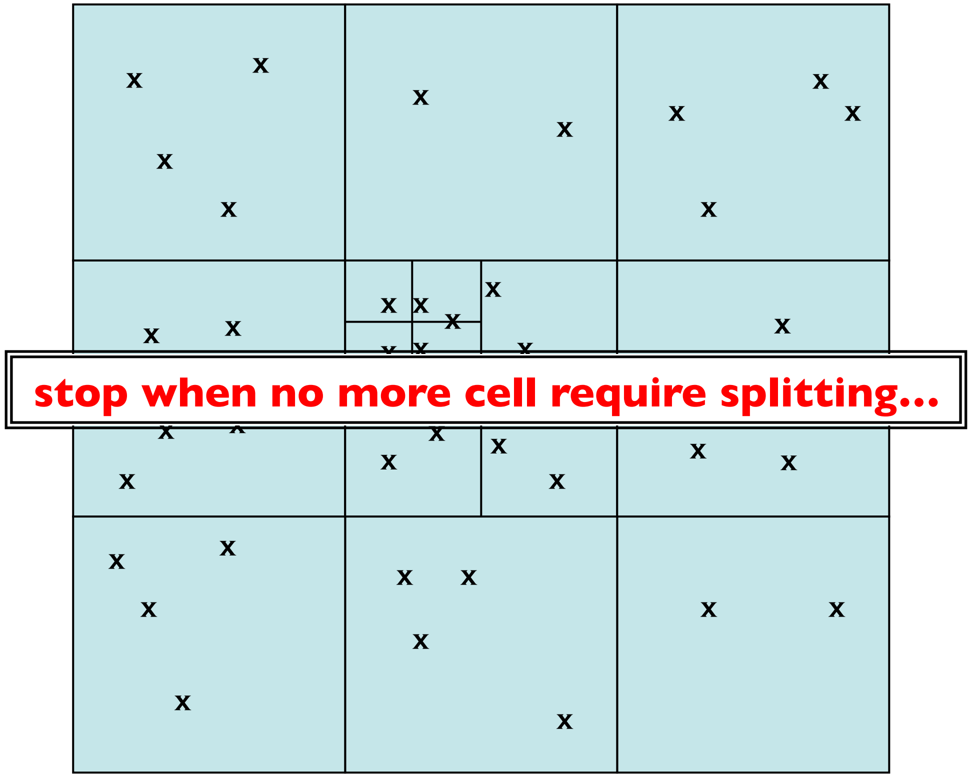
number of particles per cell



refinement criterion: 6 particles/cell

- generating refinements
  - N-body simulations:

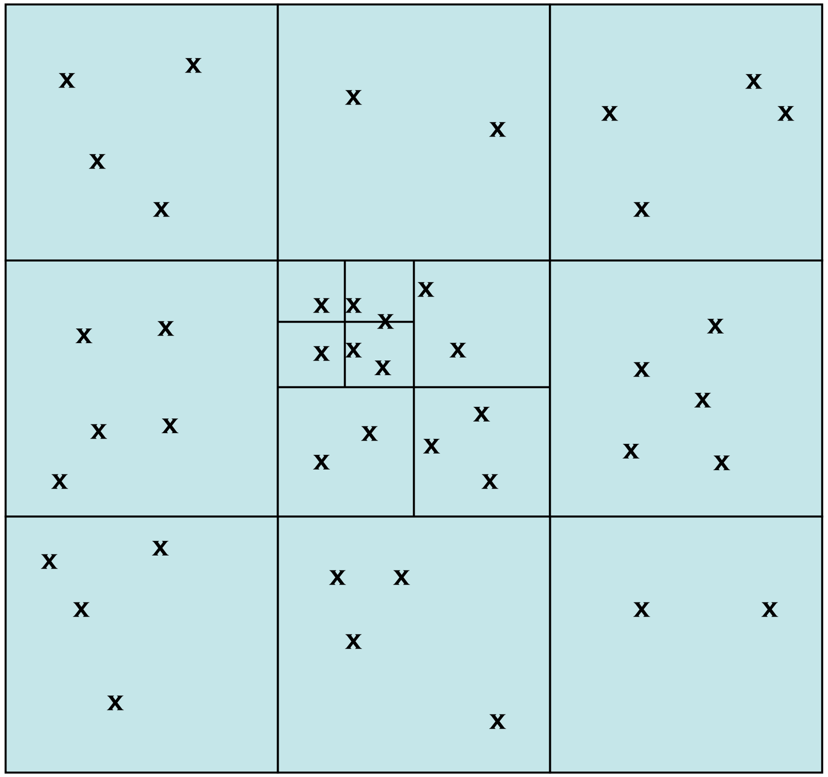
number of particles per cell



refinement criterion: 6 particles/cell

- generating refinements
  - N-body simulations:

number of particles per cell



refinement criterion: 6 particles/cell

Note:
in this scheme we split
the volume of a coarse cell
into eight equal sub-cells…

=> non-cospatial scheme!

- generating refinements
    - interpolation between grids:

$$f(x_i) = F(x_i) + F'(x_i)\Delta x$$

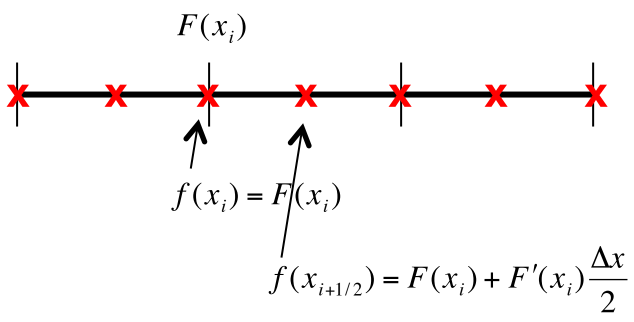
F = value on coarse grid
f = value on fine grid

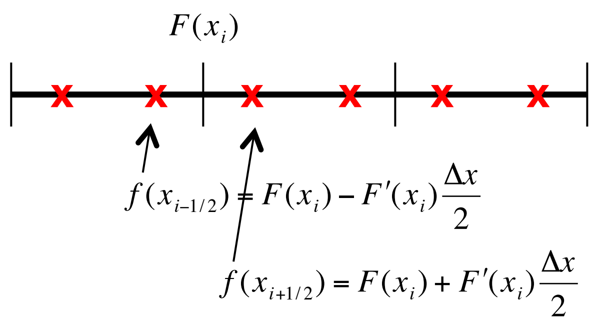
- generating refinements

  - interpolation between grids:

$$f(x_i) = F(x_i) + F'(x_i)\Delta x$$

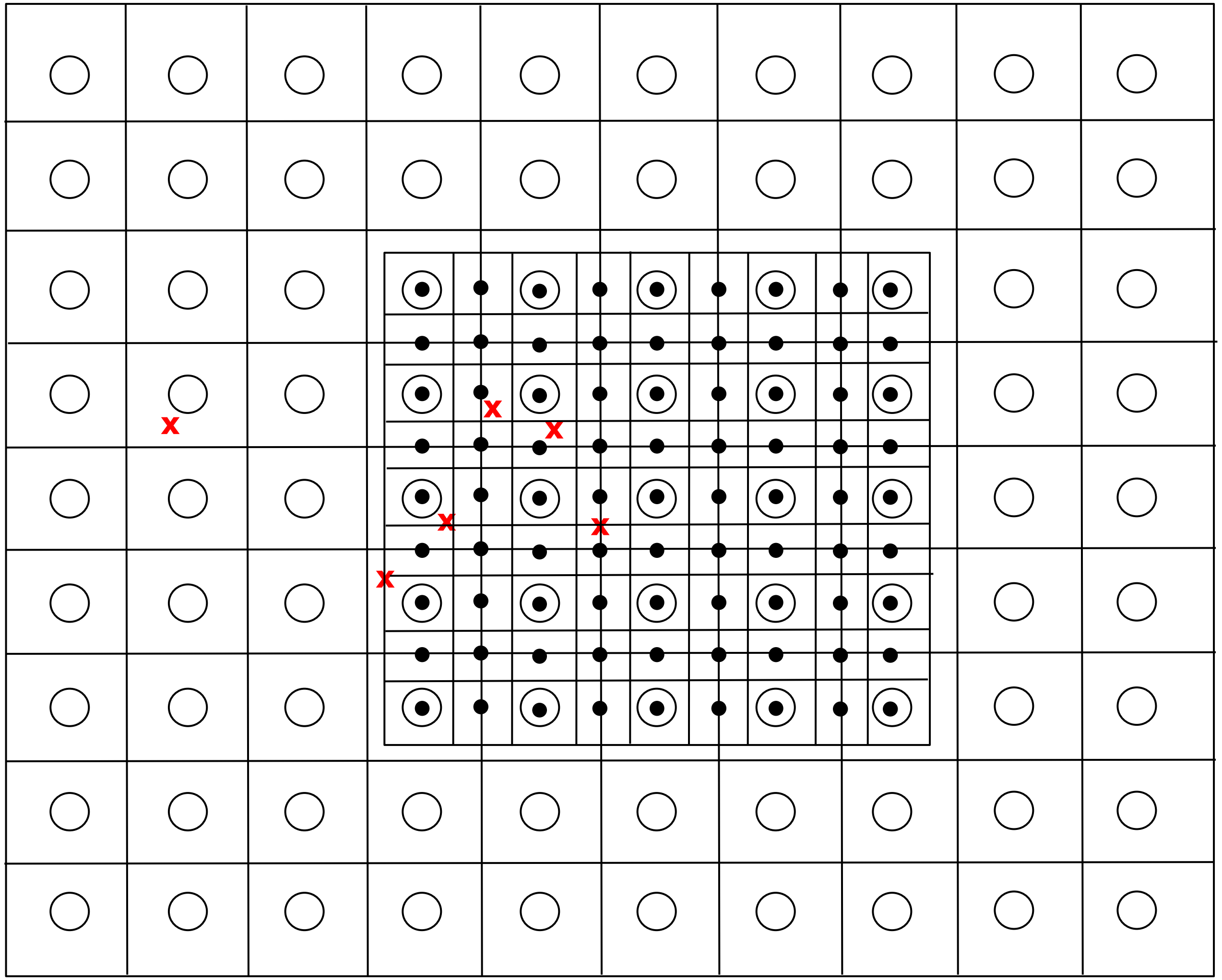
co-spatial vs. non-cospatial



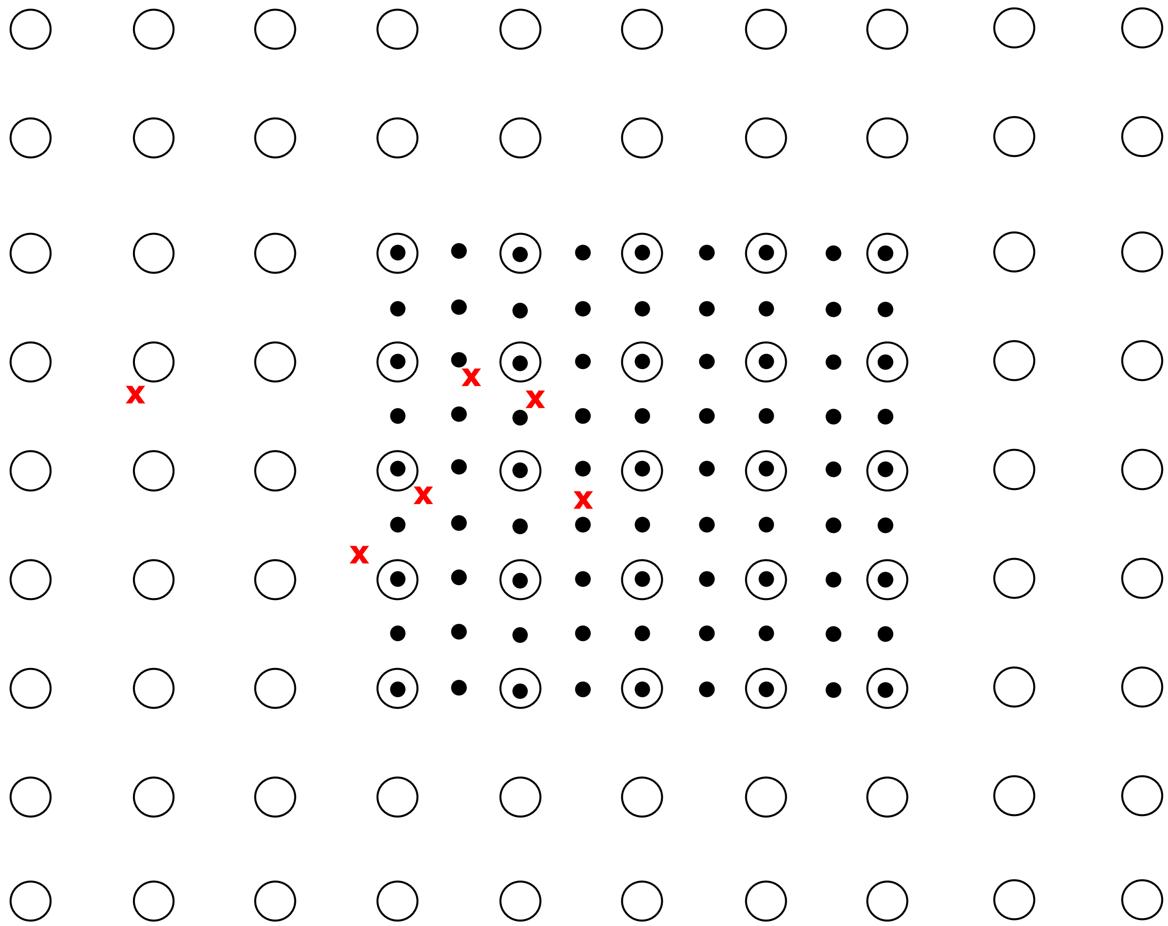


- mesh refinements
- adaptive mesh refinement
- **adaptive mesh refinement for N-body codes**
  - gravity
  - generating refinements
  - ***density assignment***
  - solving Poisson's equation
- handling irregular grids
- adaptive leap-frog integration

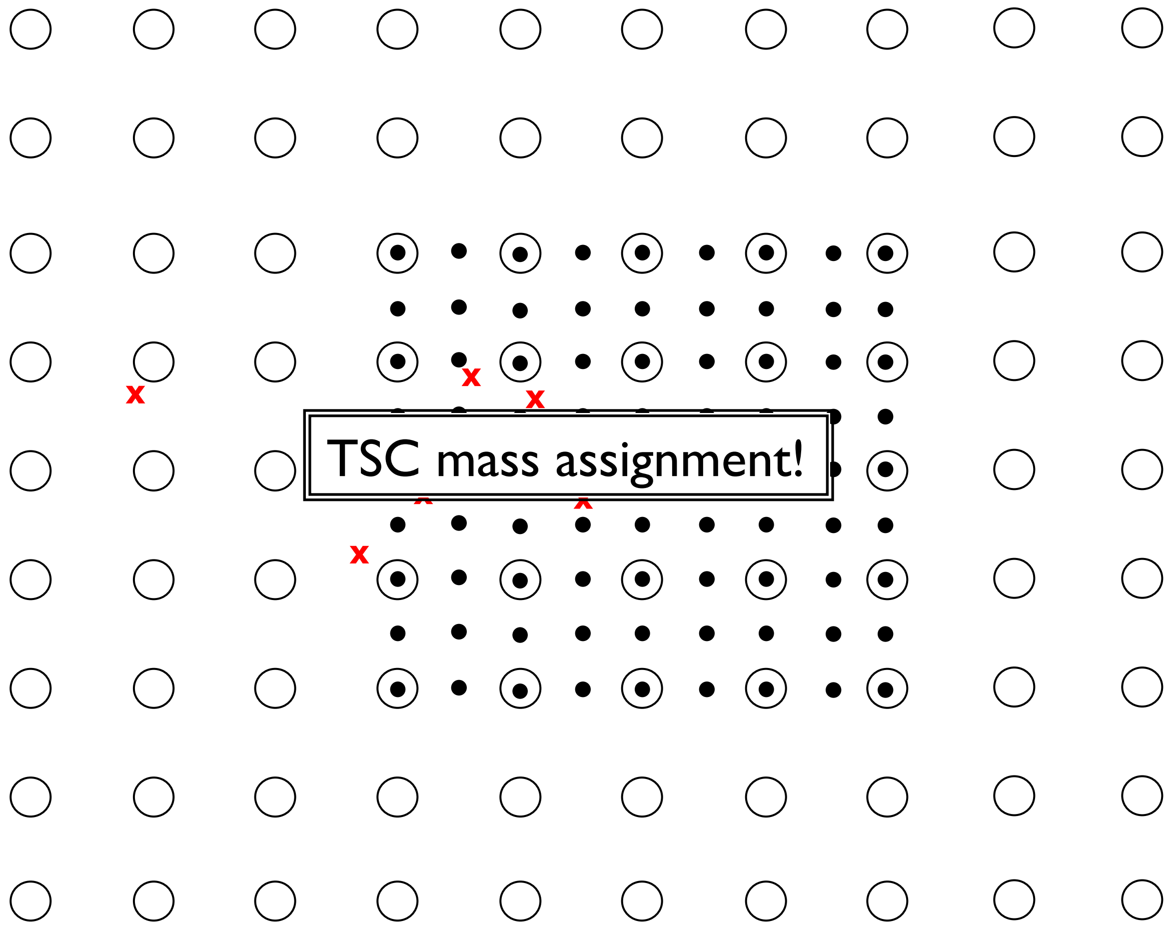
- density assignment (co-spatial scheme)



- density assignment (co-spatial scheme)

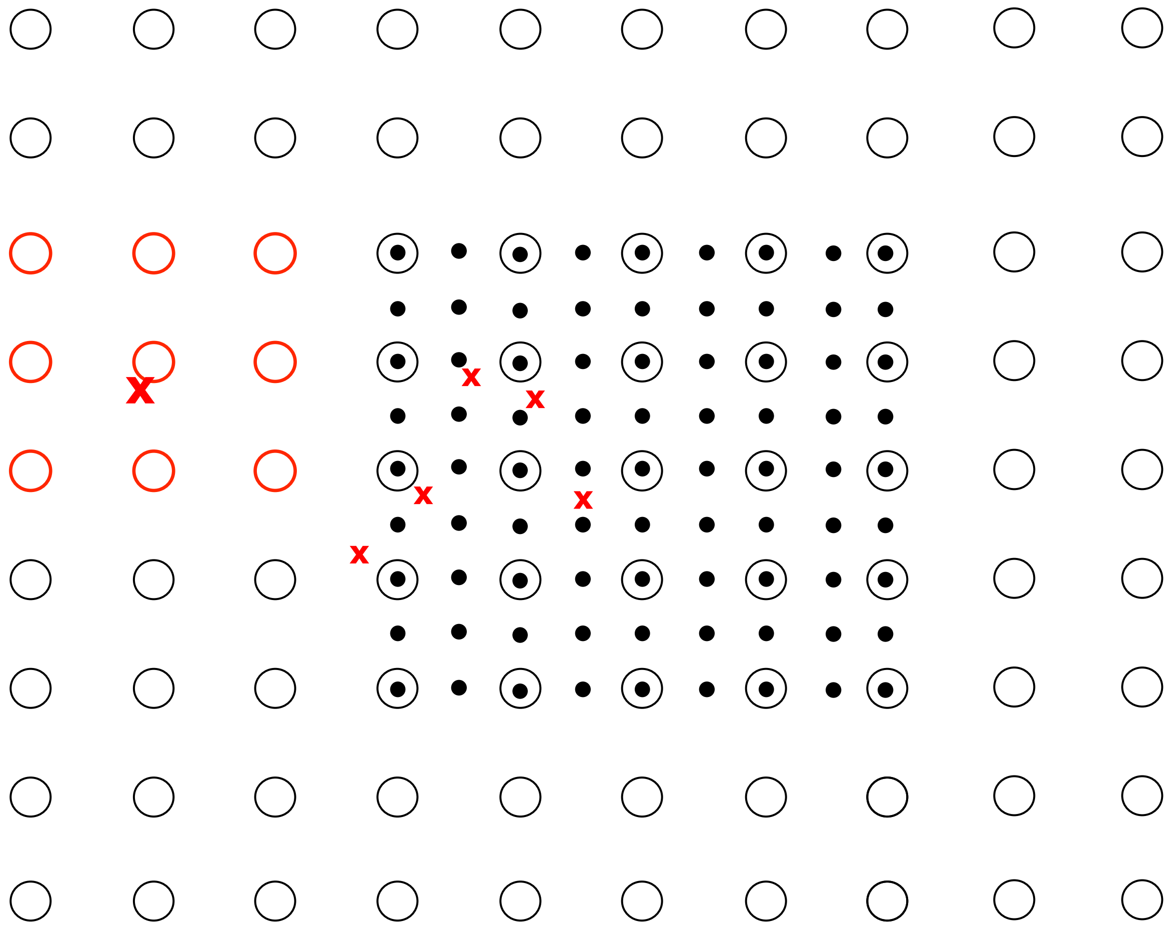


- density assignment (co-spatial scheme)



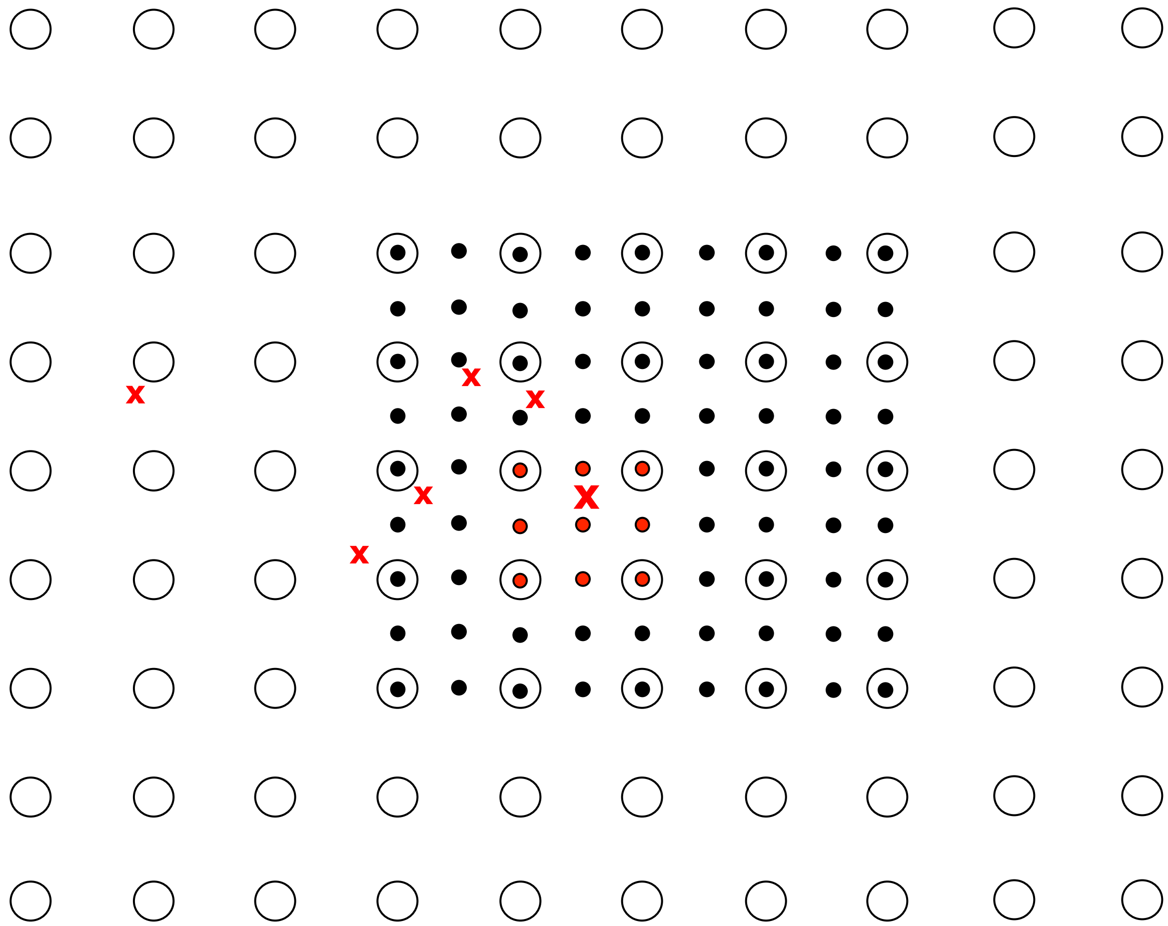
- density assignment (co-spatial scheme)

unproblematic:



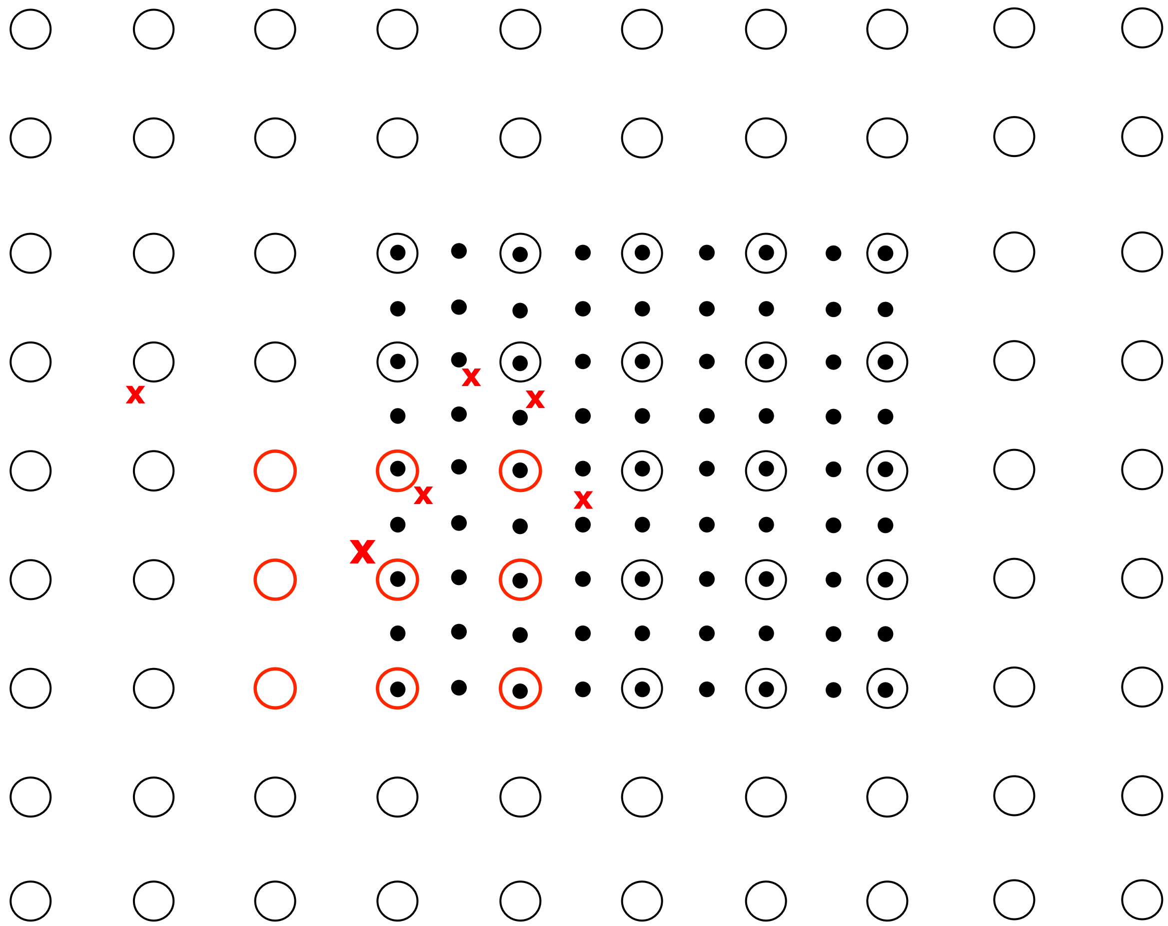
- density assignment (co-spatial scheme)

unproblematic:



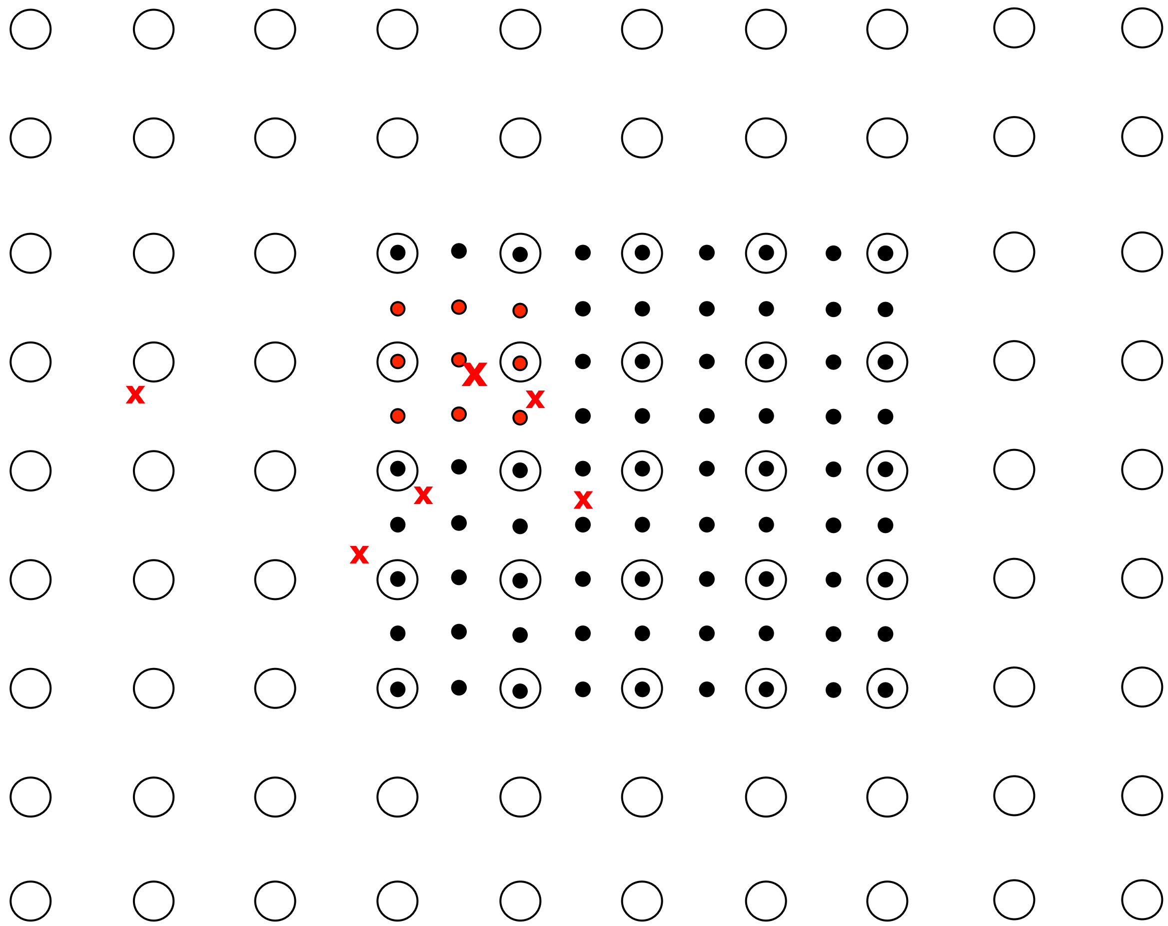
- density assignment (co-spatial scheme)

problematic:



- density assignment (co-spatial scheme)

problematic:



- density assignment (co-spatial scheme)

  - steps required to get density correct on both coarse and fine grid…

1. transfer particles from coarse to fine grid
2. assign "coarse" particles to coarse grid
3. assign "fine" particles to refinement grid
4. temporarily store "borderline" density
5. inject refinement density to coarse grid
6. add "borderline" density to refinement

- density assignment (co-spatial scheme)

  - steps required to get density correct on both coarse and fine grid…

1. transfer particles from coarse to fine grid

2. assign "coarse" particles to coarse grid

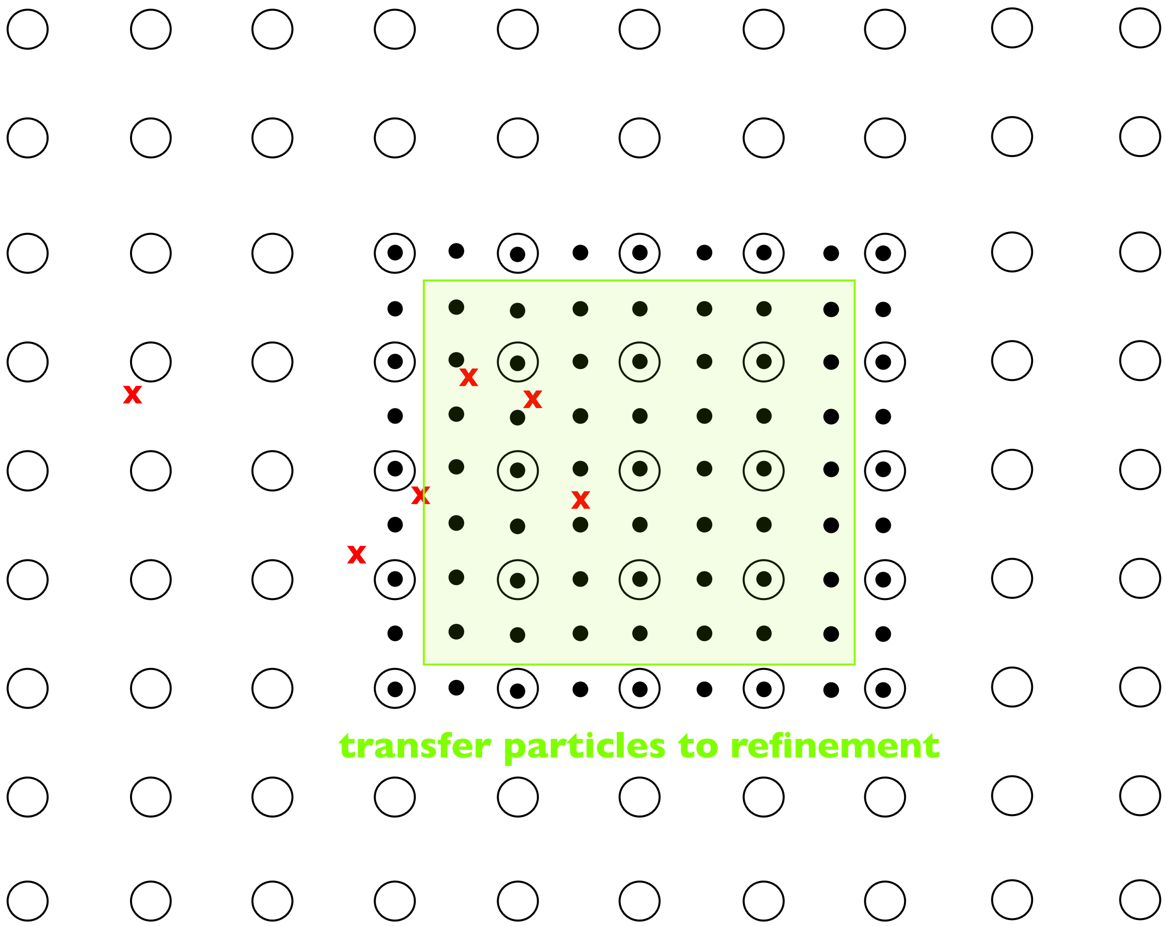
3. assign "fine" particles to refinement grid

4. temporarily store "borderline" density

5. inject refinement density to coarse grid

6. add "borderline" density to refinement

- density assignment (co-spatial scheme)

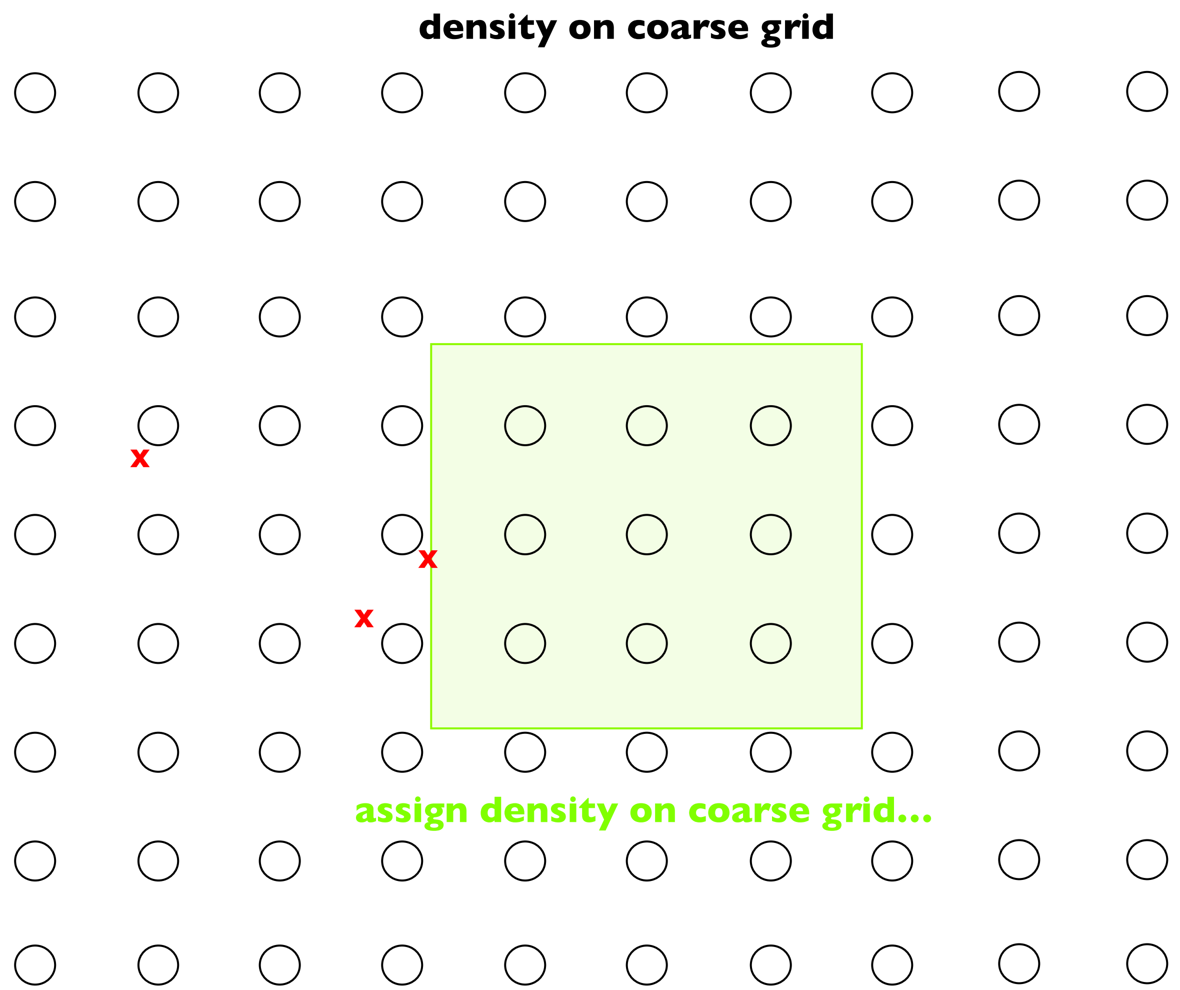


- density assignment (co-spatial scheme)

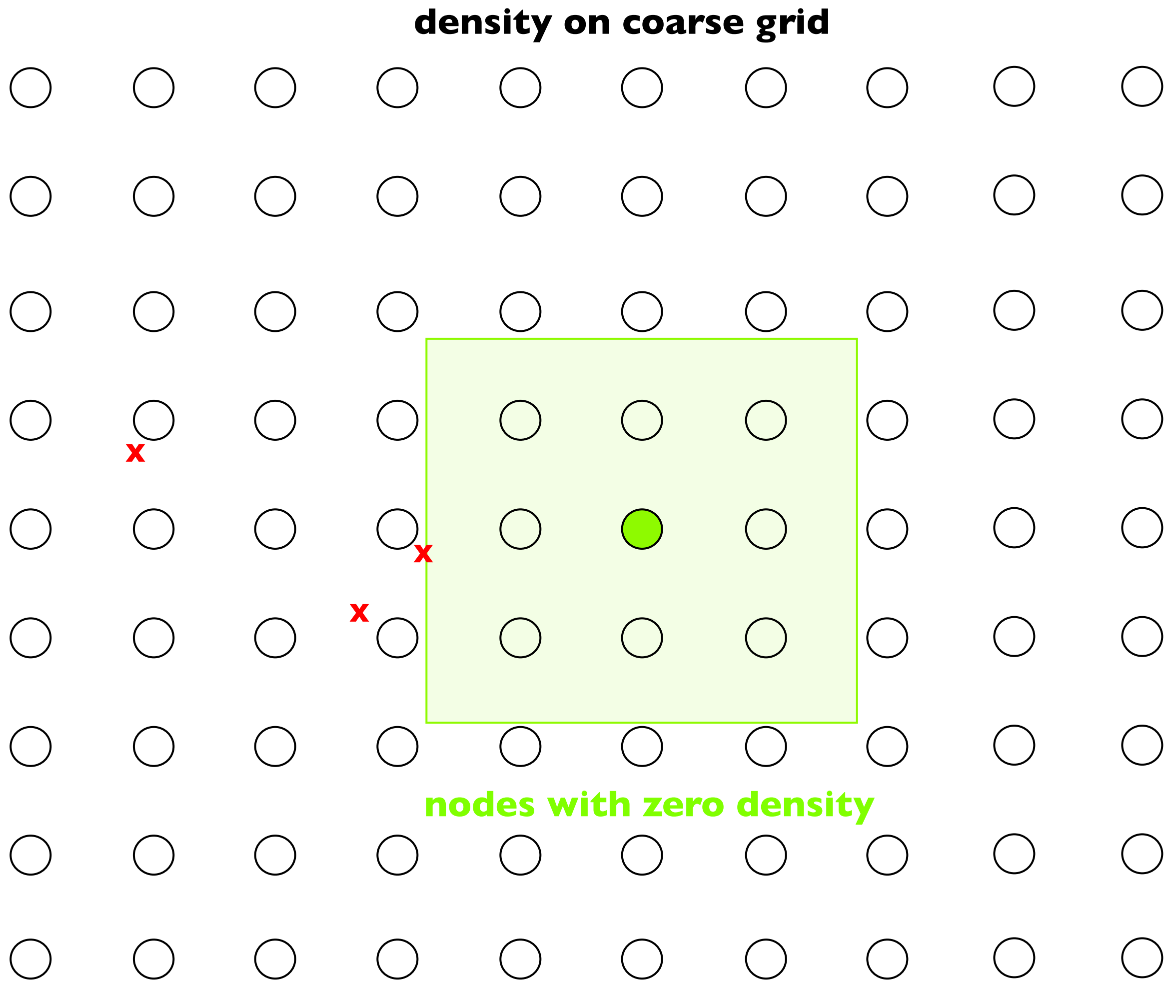
  - steps required to get density correct on both coarse and fine grid…

1. transfer particles from coarse to fine grid
2. **assign "coarse" particles to coarse grid**
3. assign "fine" particles to refinement grid
4. temporarily store "borderline" density
5. inject refinement density to coarse grid
6. add "borderline" density to refinement

- density assignment (co-spatial scheme)

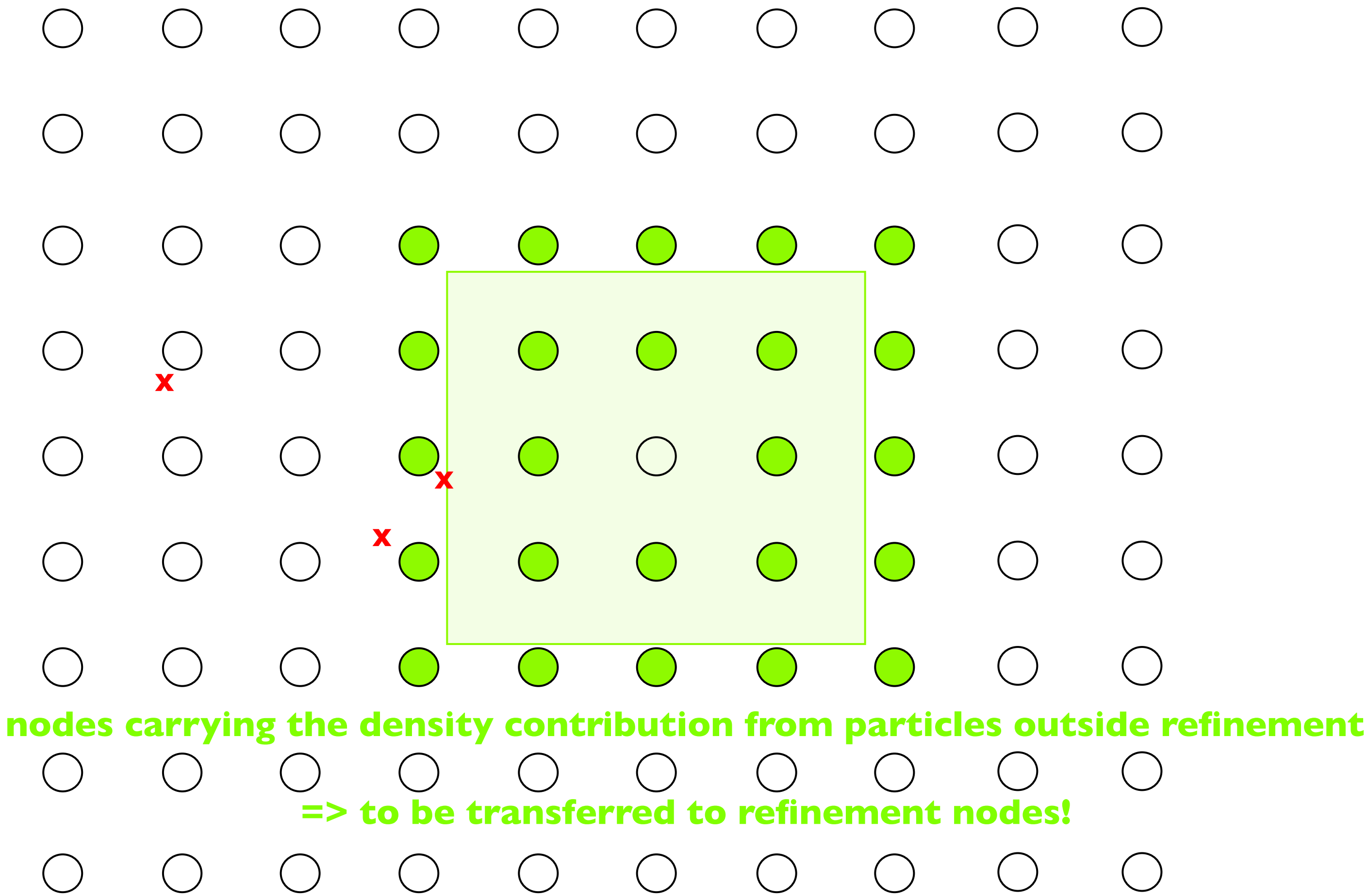


- density assignment (co-spatial scheme)

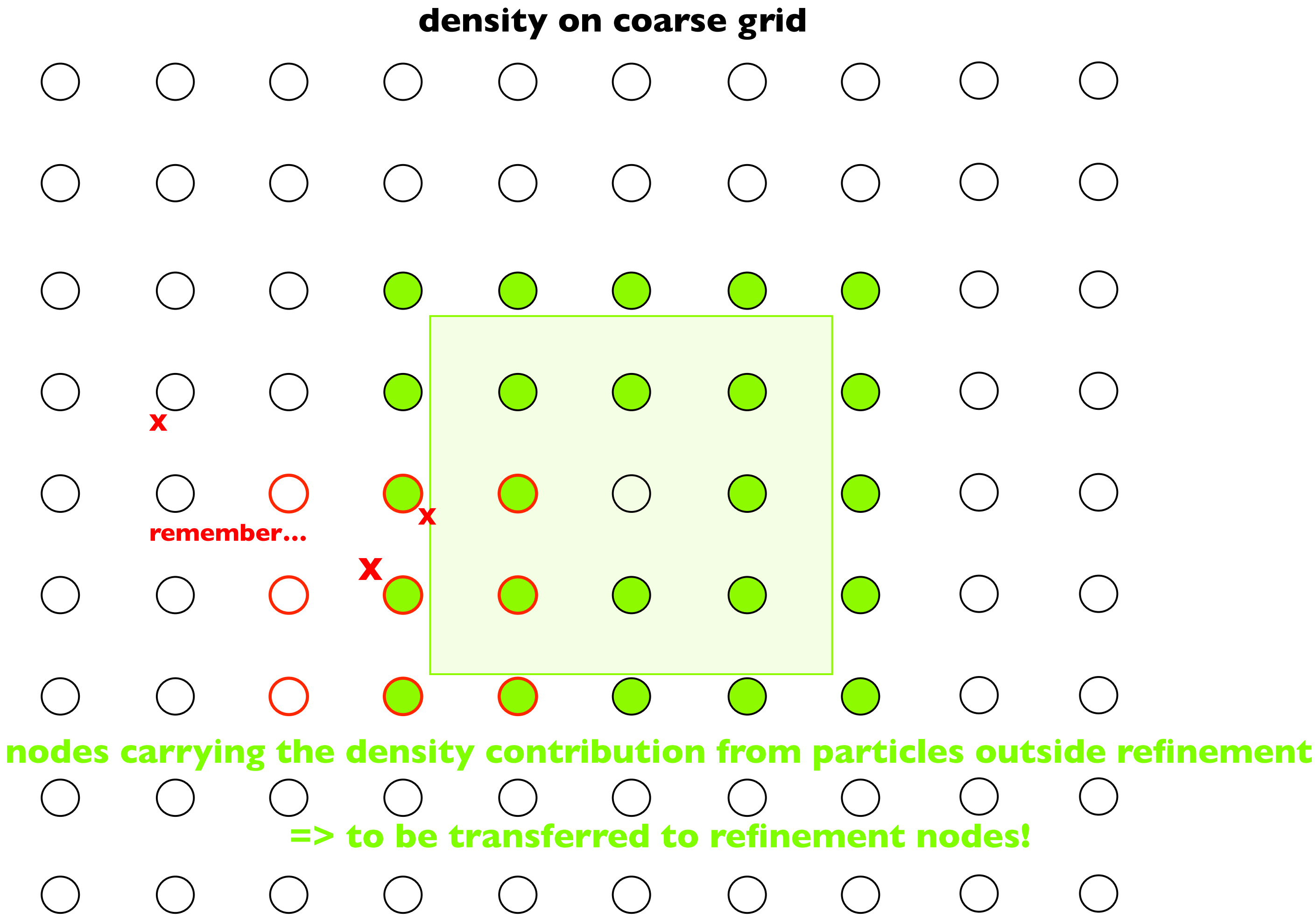


- density assignment (co-spatial scheme)

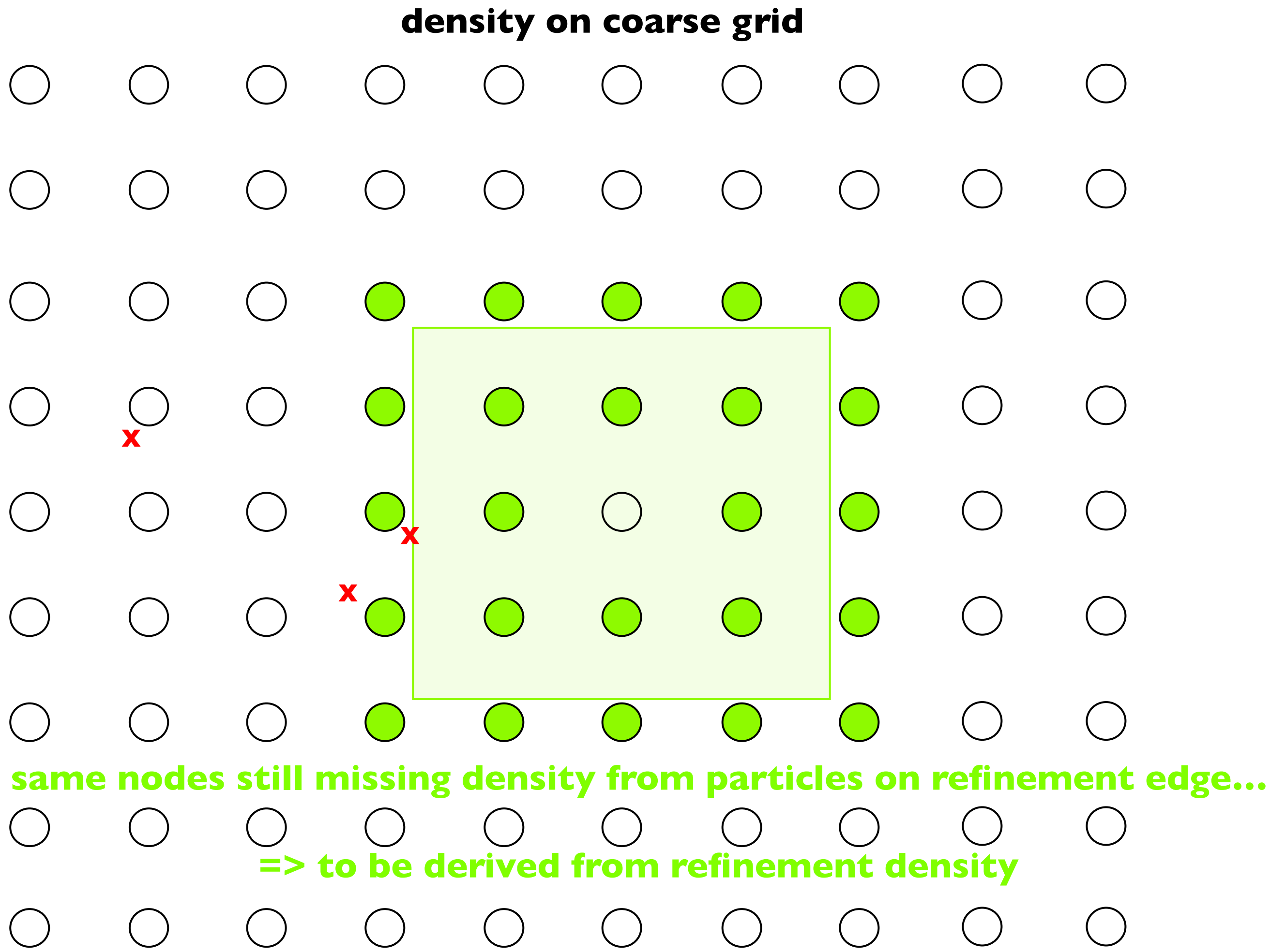
density on coarse grid



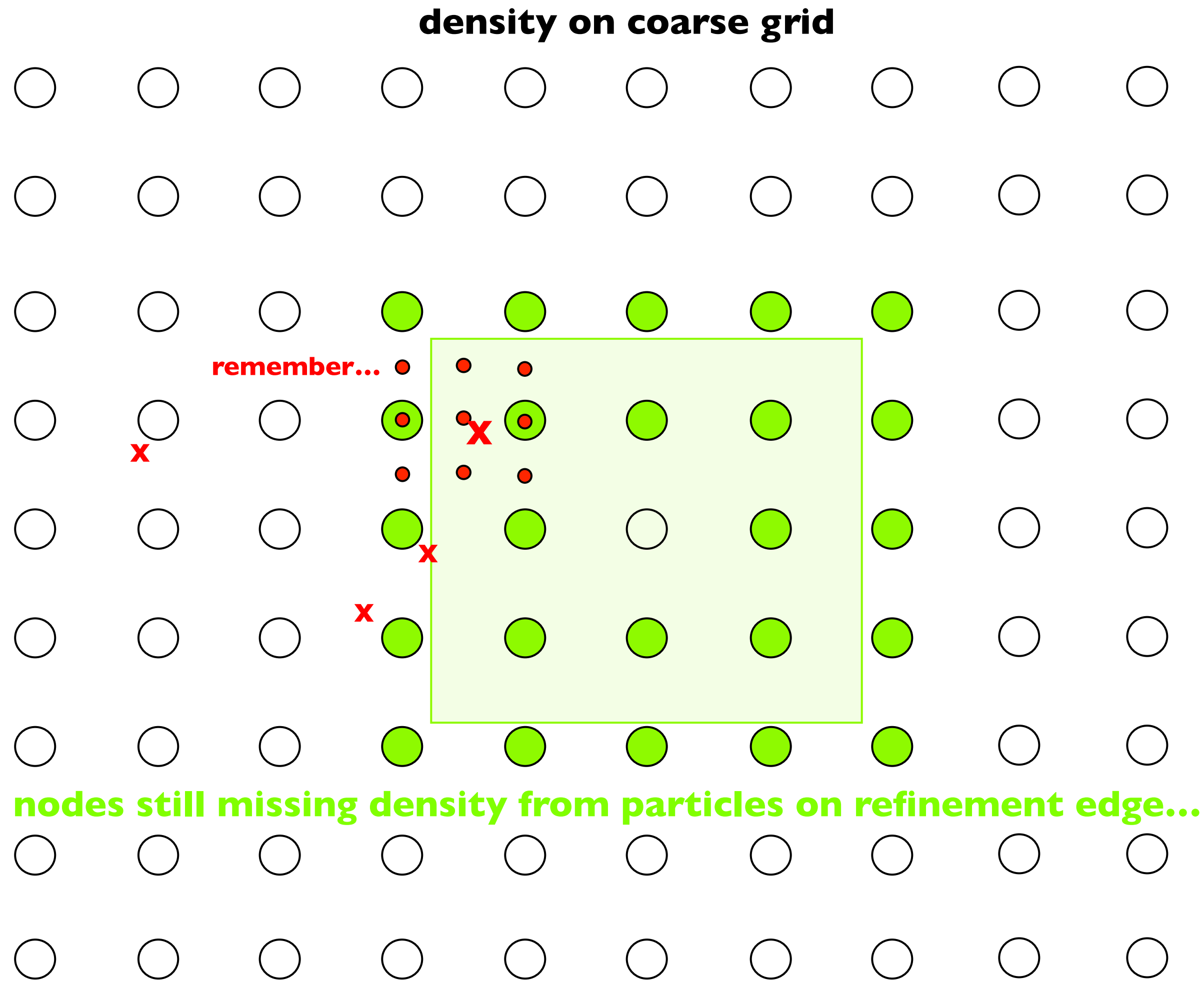
- density assignment (co-spatial scheme)



- density assignment (co-spatial scheme)



- density assignment (co-spatial scheme)



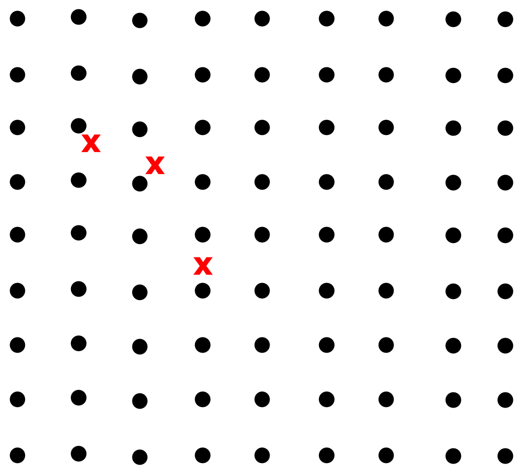
- density assignment (co-spatial scheme)

  • steps required to get density correct on both coarse and fine grid…

1. transfer particles from coarse to fine grid
2. assign "coarse" particles to coarse grid
3. **assign "fine" particles to refinement grid**
4. temporarily store "borderline" density
5. inject refinement density to coarse grid
6. add "borderline" density to refinement

- density assignment (co-spatial scheme)

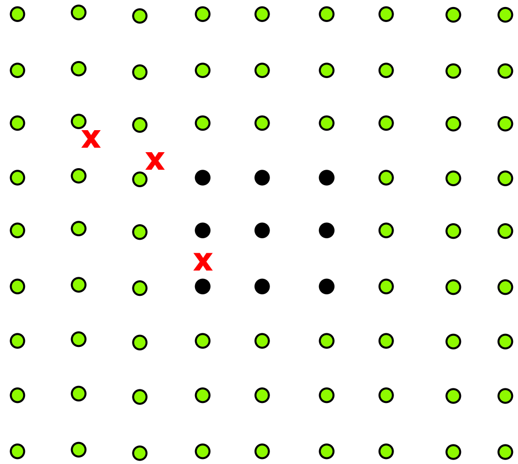
density on refinement grid



assign density on refinement grid…

- density assignment (co-spatial scheme)

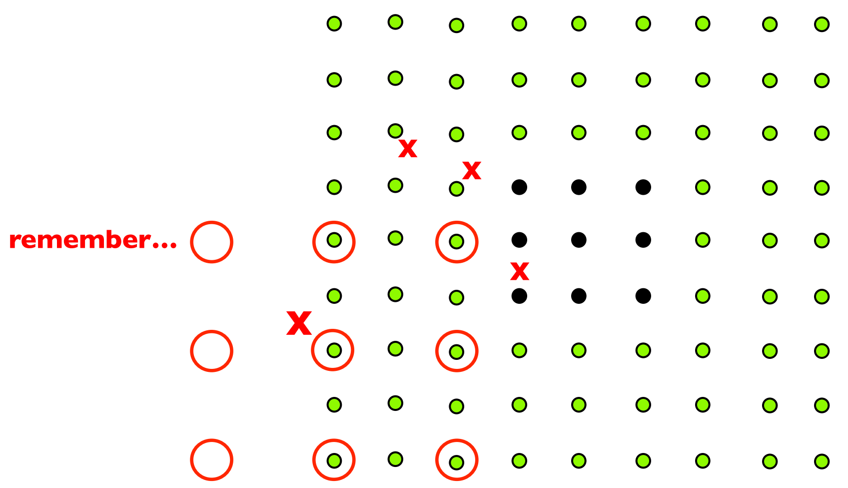
density on refinement grid



refinement nodes still missing the density contribution from particles outside refinement

- density assignment (co-spatial scheme)

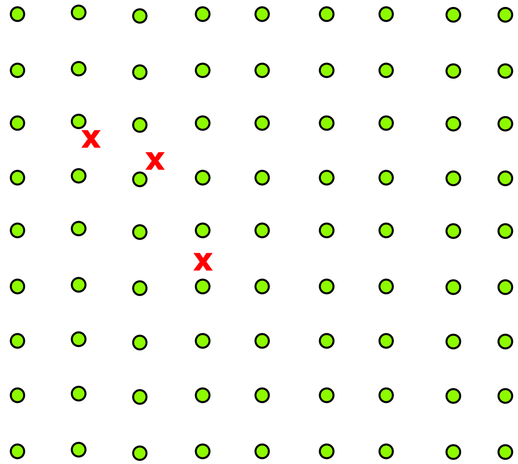
density on refinement grid



refinement nodes still missing the density contribution from particles outside refinement

- density assignment (co-spatial scheme)

density on refinement grid



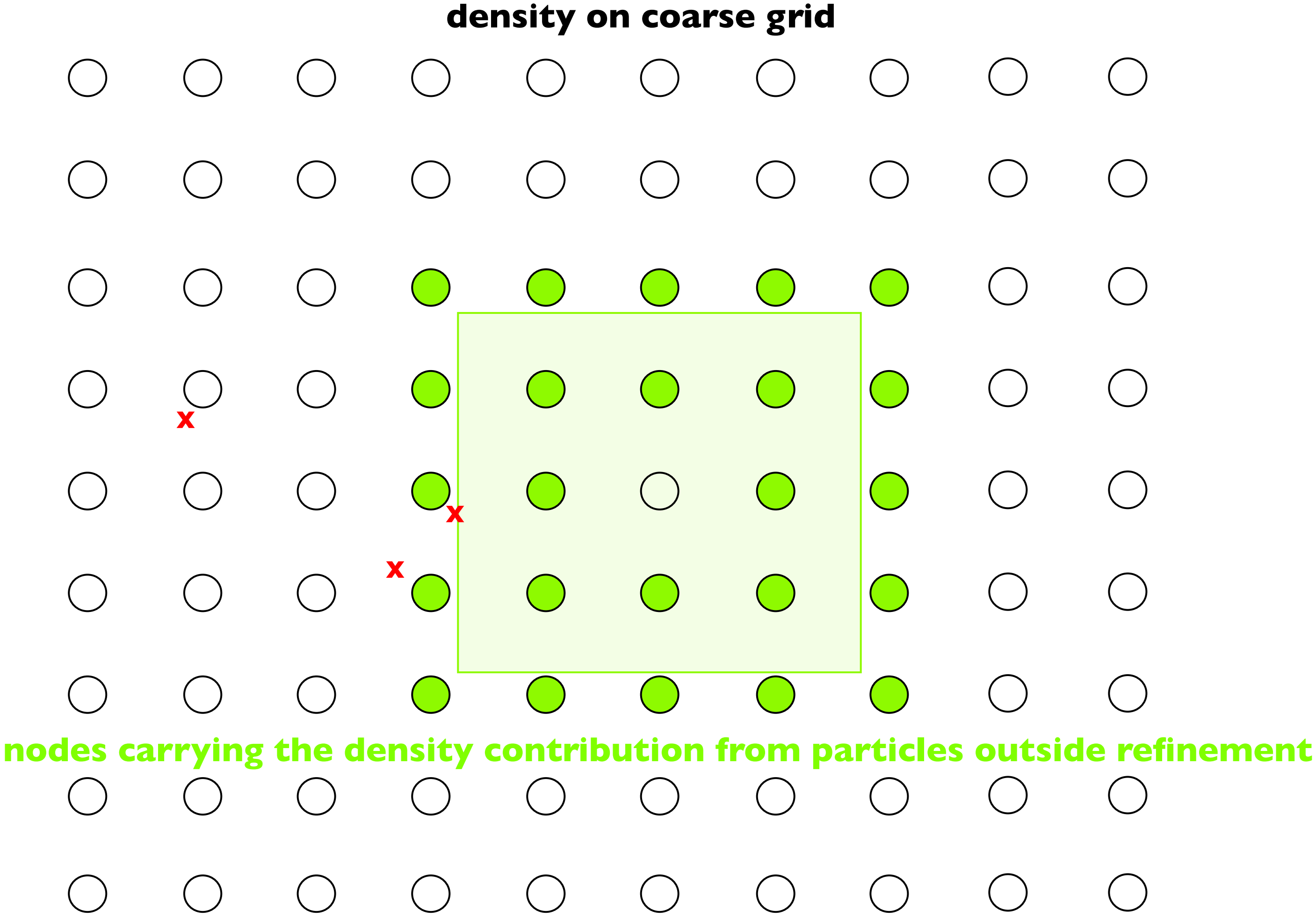
all refinement nodes carry information required by coarse nodes...

- density assignment (co-spatial scheme)

  - steps required to get density correct on both coarse and fine grid…

1. transfer particles from coarse to fine grid
2. assign "coarse" particles to coarse grid
3. assign "fine" particles to refinement grid
4. **temporarily store "borderline" density**
5. inject refinement density to coarse grid
6. add "borderline" density to refinement

- density assignment (co-spatial scheme)

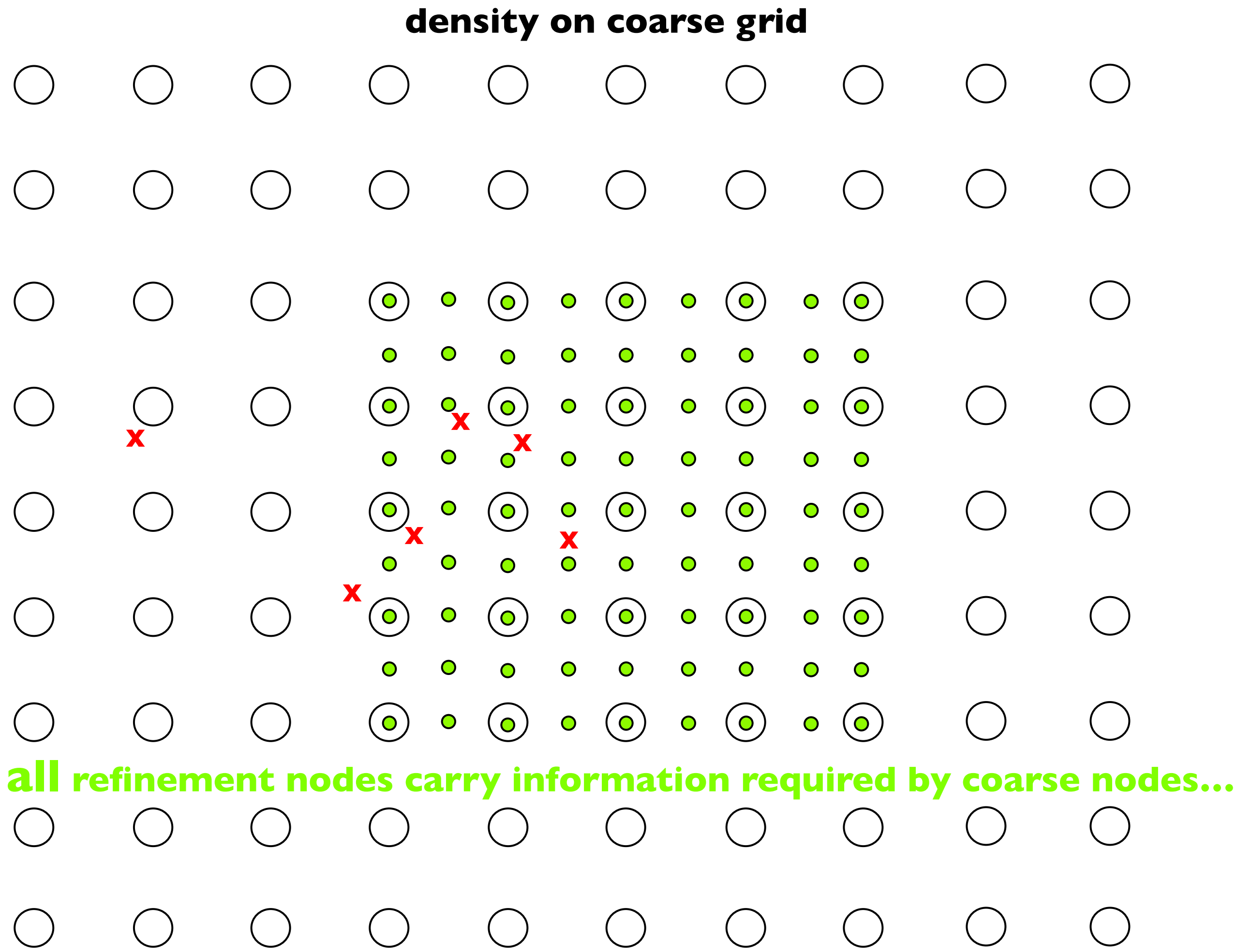


- density assignment (co-spatial scheme)

  - steps required to get density correct on both coarse and fine grid…

1. transfer particles from coarse to fine grid
2. assign "coarse" particles to coarse grid
3. assign "fine" particles to refinement grid
4. temporarily store "borderline" density
5. **inject refinement density to coarse grid**
6. add "borderline" density to refinement

- density assignment (co-spatial scheme)

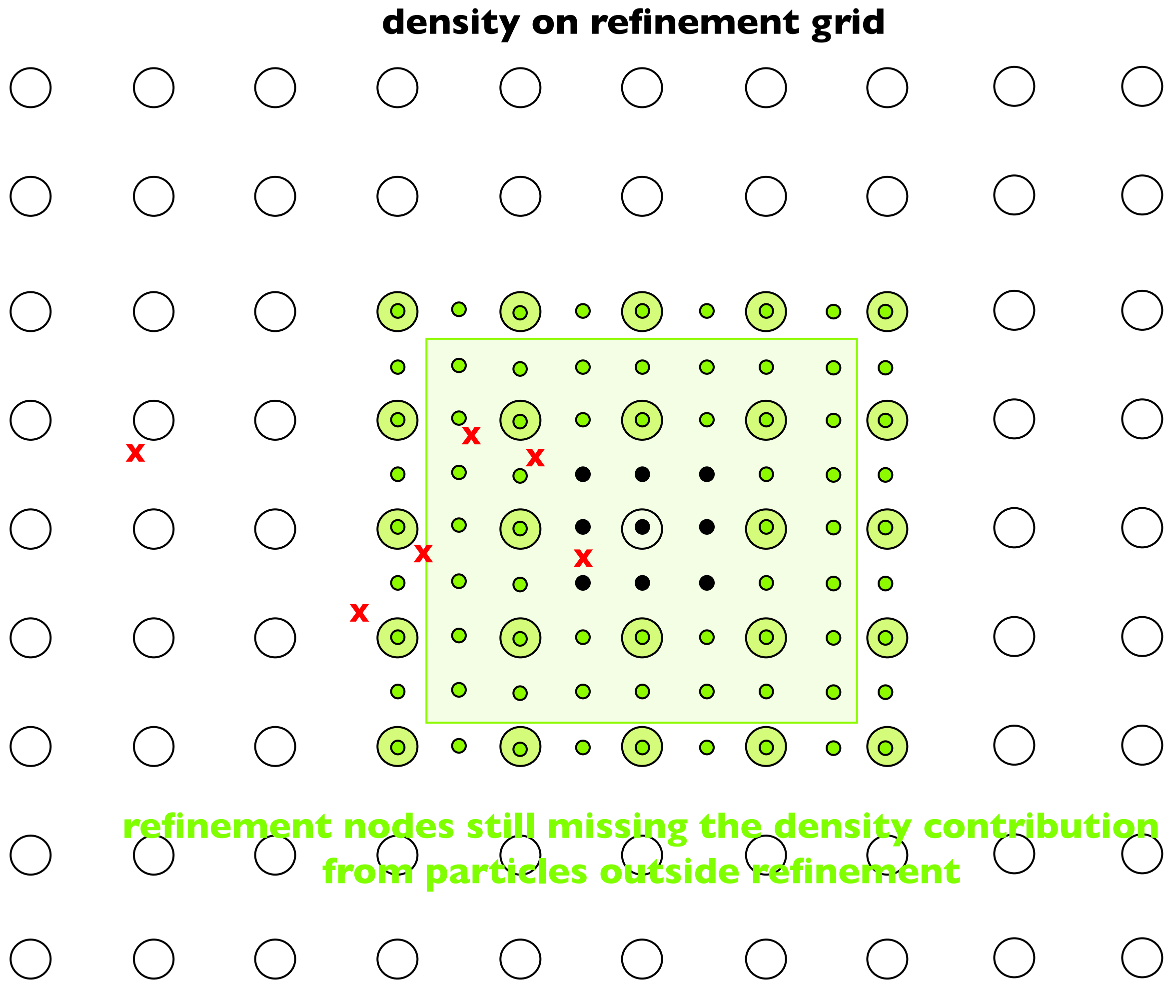


- density assignment (co-spatial scheme)

  - steps required to get density correct on both coarse and fine grid…

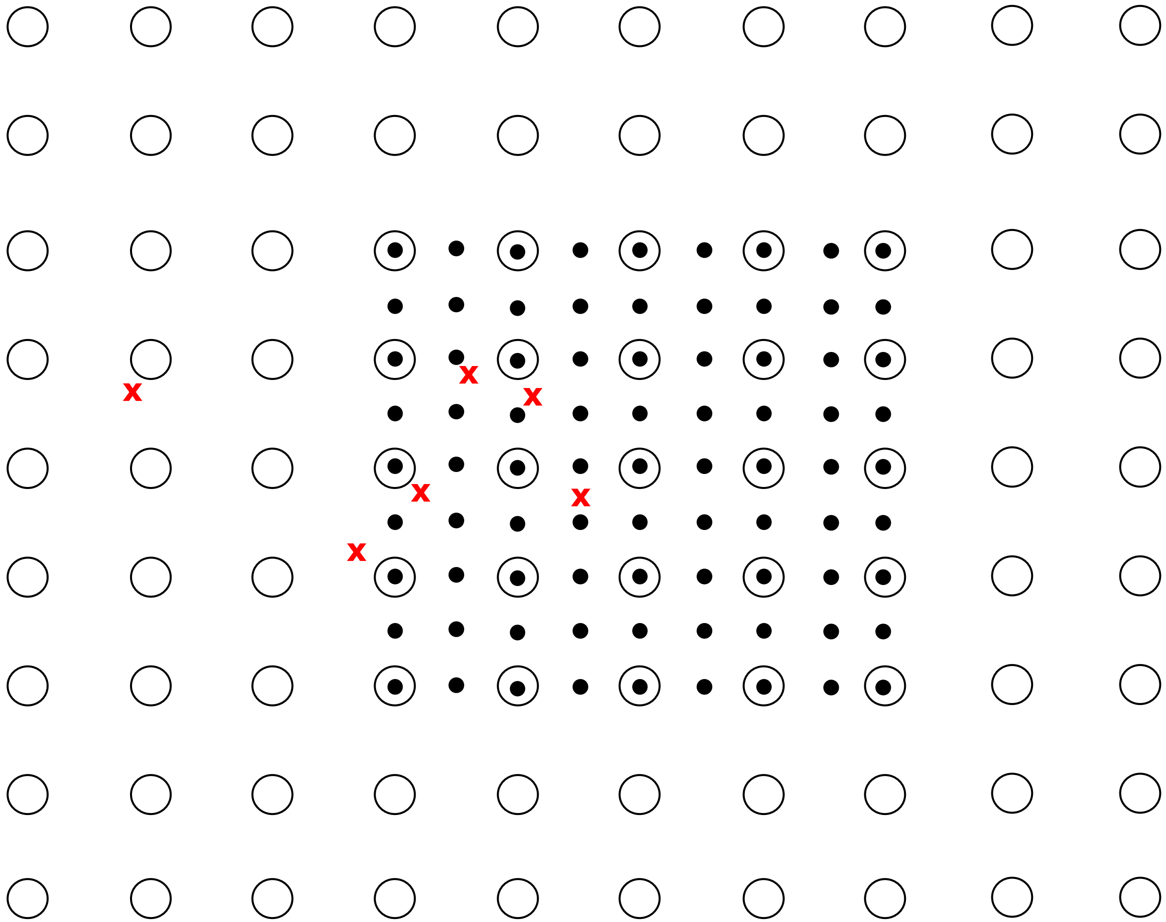
1. transfer particles from coarse to fine grid
2. assign “coarse” particles to coarse grid
3. assign “fine” particles to refinement grid
4. temporarily store “borderline” density
5. inject refinement density to coarse grid
6. **add “borderline” density to refinement**

- density assignment (co-spatial scheme)



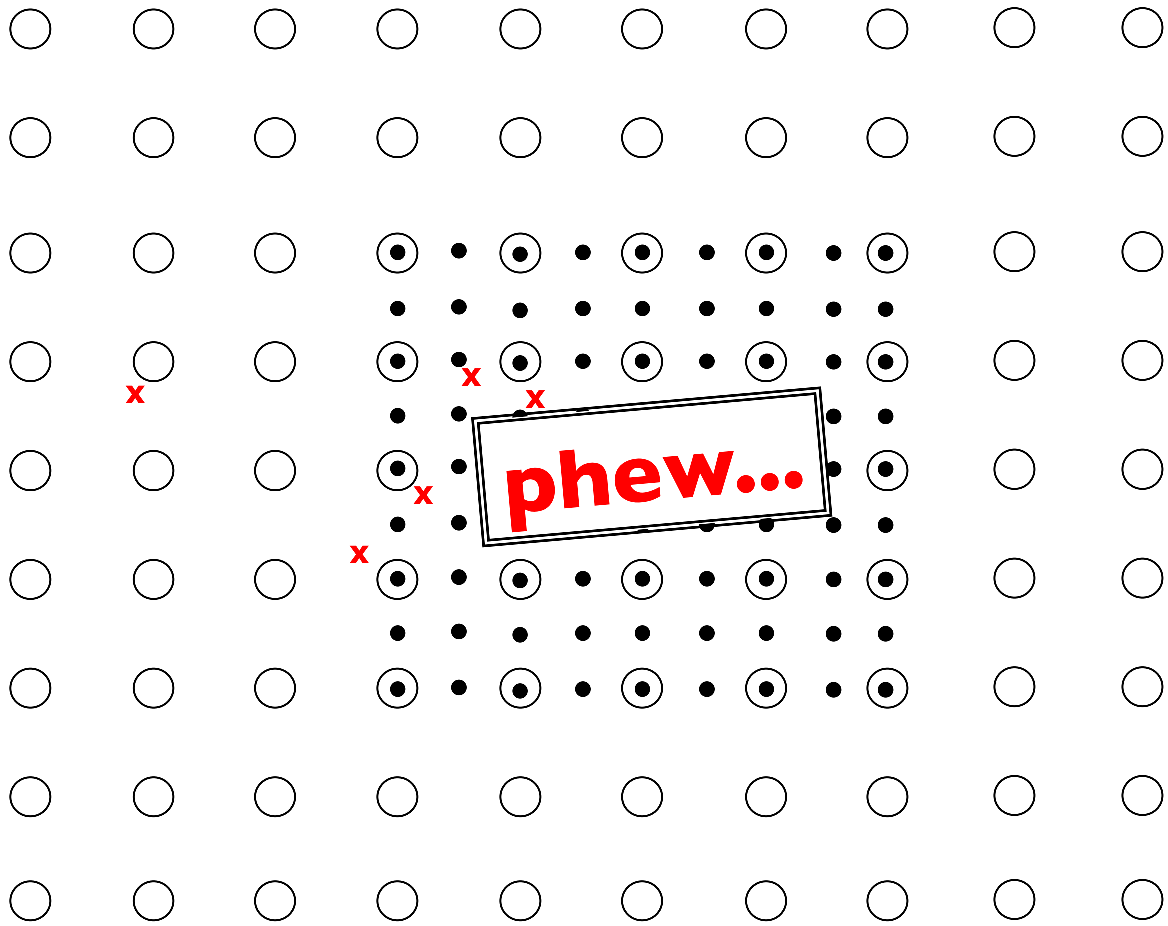
- density assignment (co-spatial scheme)

density finally correct on both levels…



- density assignment (co-spatial scheme)

density finally correct on both levels...

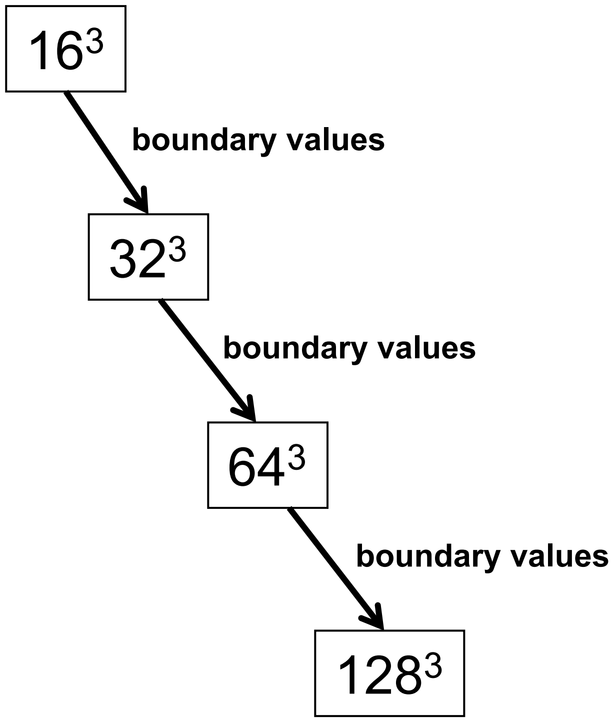


- mesh refinements
- adaptive mesh refinement
- **adaptive mesh refinement for N-body codes**
    - gravity
    - generating refinements
    - density assignment
    - ***solving Poisson's equation***
- handling irregular grids
- adaptive leap-frog integration

- solving Poisson's equation

1. the domain grid:
   - relaxation, FFT, …

2. the refinement grids:
   - brute force relaxation!



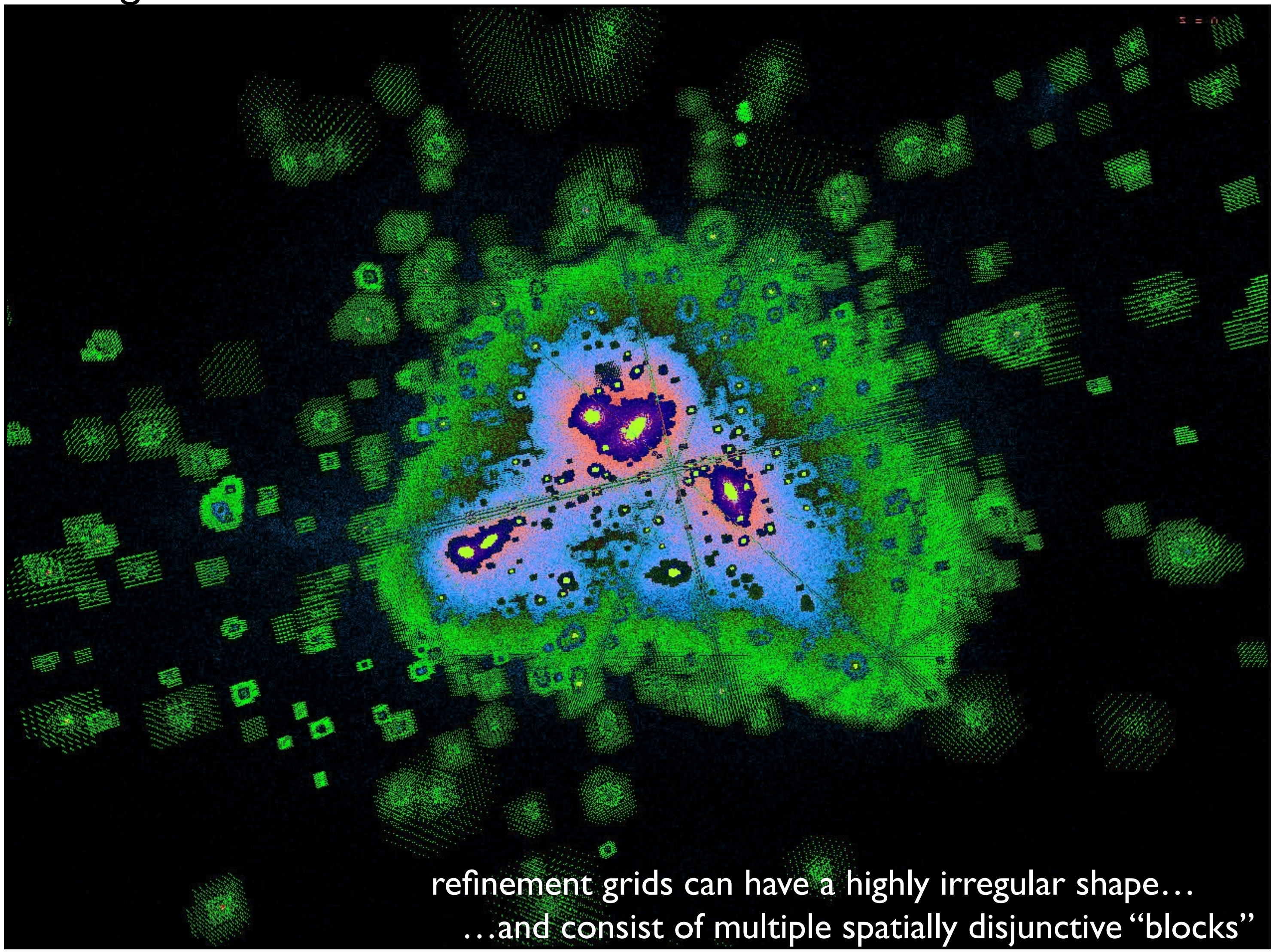
- adaptive mesh refinement
    - cover simulation with regular domain grid
    - create AMR hierarchy:
        - generate fine grid by comparing each node against some refinement criterion…

          ➔ recursive procedure!
    - assign density on all grids
    - solve Poisson's equation on regular domain grid (FFT is fastest…)
    - loop over all refinement levels:
        - interpolate potential down from parent level
        - relax potential until converged (keeping boundary values fixed)

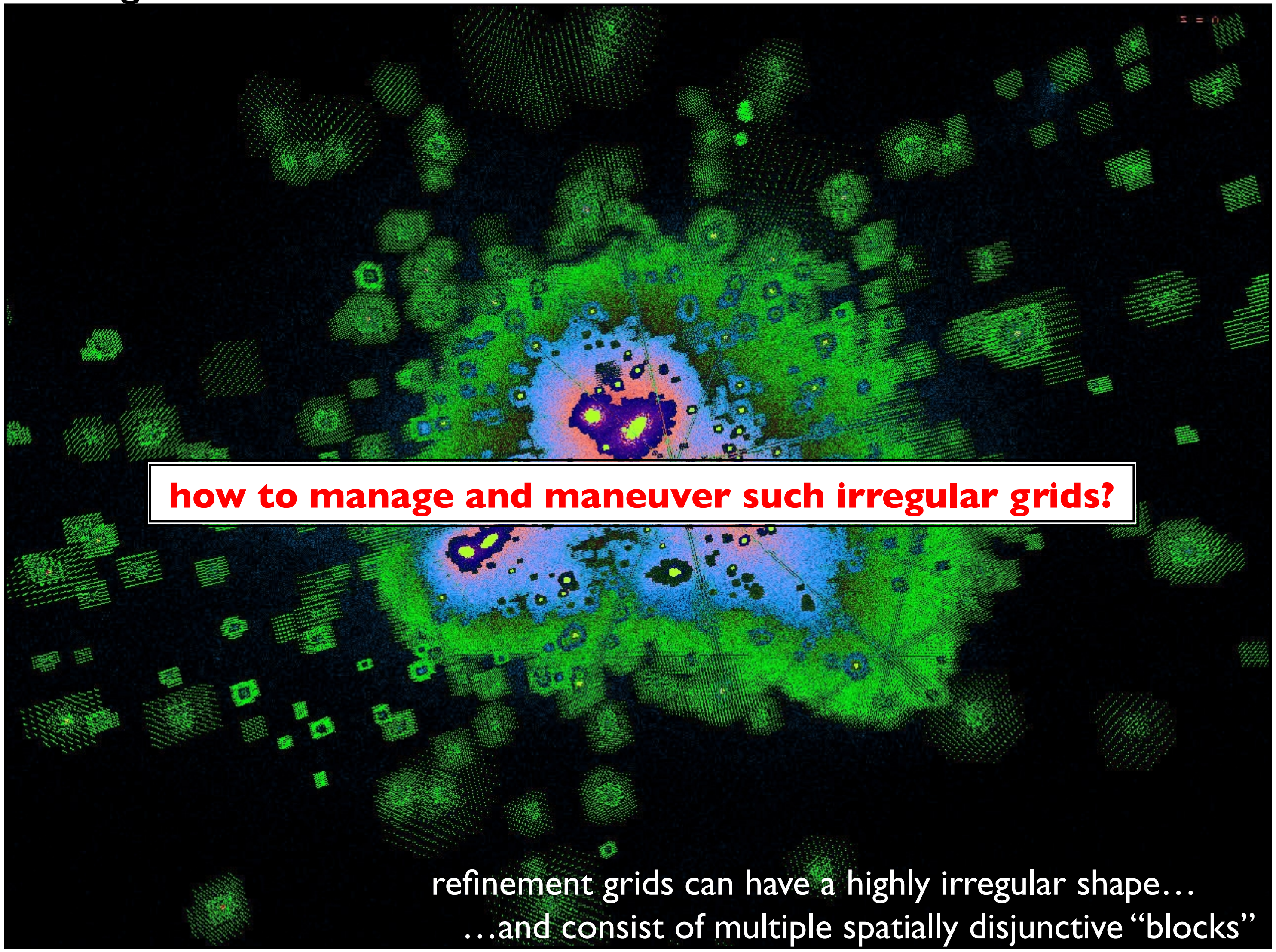
          ➔ this will give the correct potential on all (refinement) grids

- mesh refinements
- adaptive mesh refinement
- adaptive mesh refinement for N-body codes
- **handling irregular grids**
- adaptive leap-frog integration

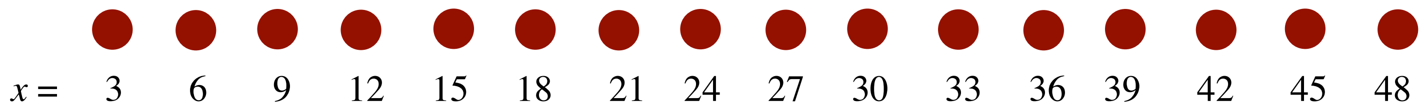
- handling refinements



- handling refinements



- handling regular grids (1D)



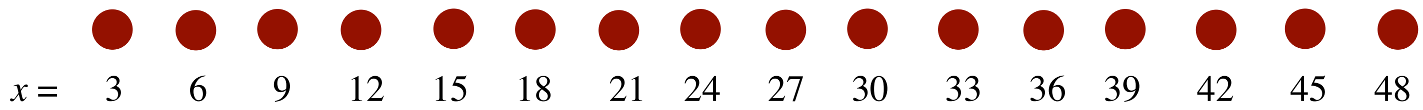
N = 16

node[0].rho, x=3
node[1].rho, x=6
…,
node[N-1].rho, x=48

```
struct {

 float rho;
 …
} node;
```

- handling regular grids (1D)



N = 16

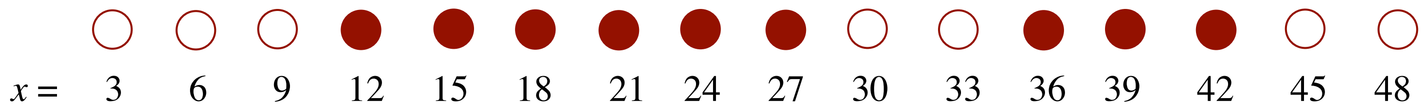
node[0].rho, x=3
node[1].rho, x=6
…,
node[N-1].rho, x=48

```
struct {

 float rho;
 ...
} node;
```

unique mapping between array index i and spatial position x possible

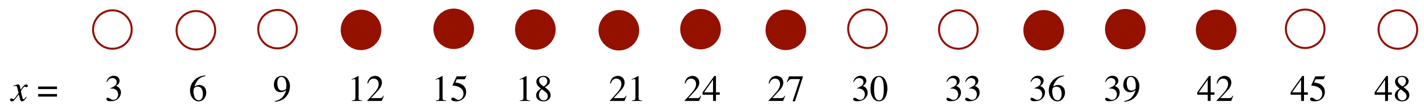
- handling irregular grids (1D)



N = 9

```
struct {
 float rho;
 …
} node;
```

- handling irregular grids (1D)



N = 9

node[0].x = 12
node[1].x = 15
node[2].x = 18
node[3].x = 21
node[4].x = 24
node[5].x = 27

node[6].x = 36
node[7].x = 39
node[8].x = 42

```
struct {
 long  x;
 float rho;
 ...
} node;
```

- handling irregular grids (1D)

$x =$ 3 6 9 12 15 18 21 24 27 30 33 36 39 42 45 48

N = 9

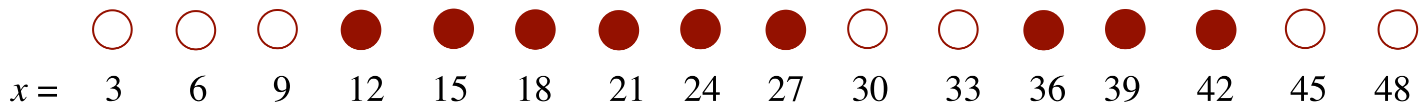
node[0].x = 12
node[1].x = 15
node[2].x = 18
node[3].x = 21
node[4].x = 24
node[5].x = 27

node[6].x = 36
node[7].x = 39
node[8].x = 42

```
struct {
 long  x;
 float rho;
 …
} node;
```

no unique mapping between array index i and spatial position x possible

- handling irregular grids (1D)



N = 9

node[0].x = 12
node[1].x = 15
node[2].x = 18
node[3].x = 21
node[4].x = 24
node[5].x = 27

node[6].x = 36
node[7].x = 39
node[8].x = 42

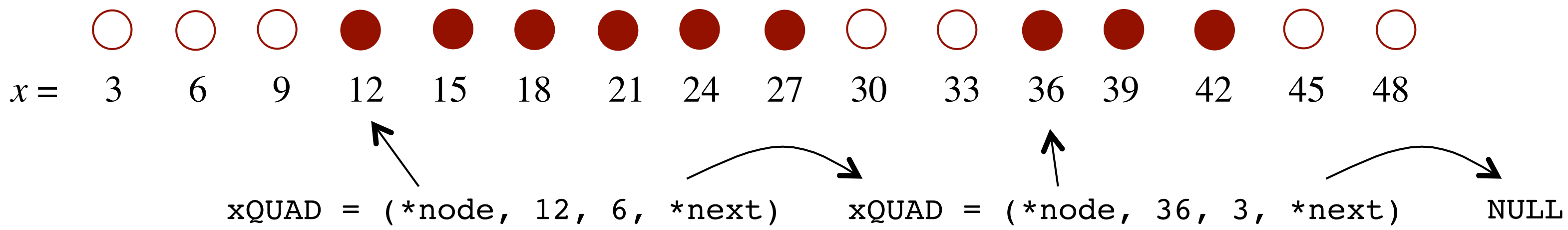
```
struct {
 long  x;
 float rho;
 ...
} node;
```

no … *x* **possible**

generate a meta-structure storing the geometry of the grid

- handling irregular grids (1D)

quad's



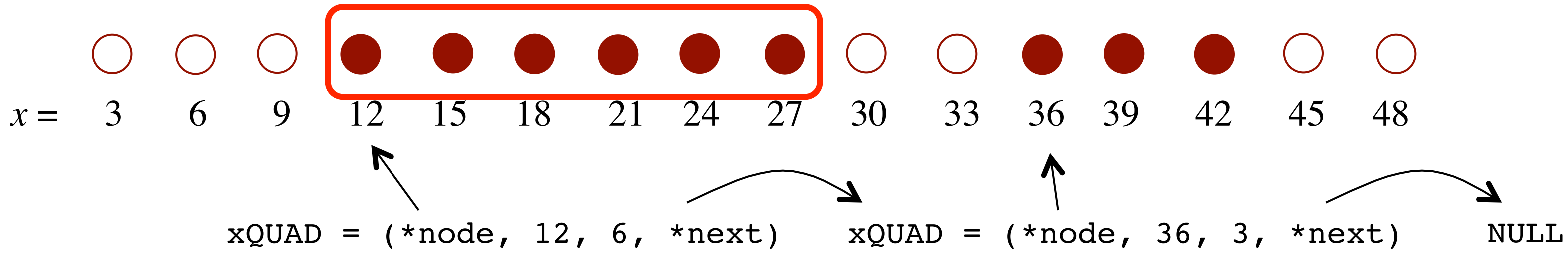

```
struct {

 float rho;
 …
} node;
```

- handling irregular grids (1D)

quad's

the memory for each array of nodes needs to be consecutively malloc()'ed



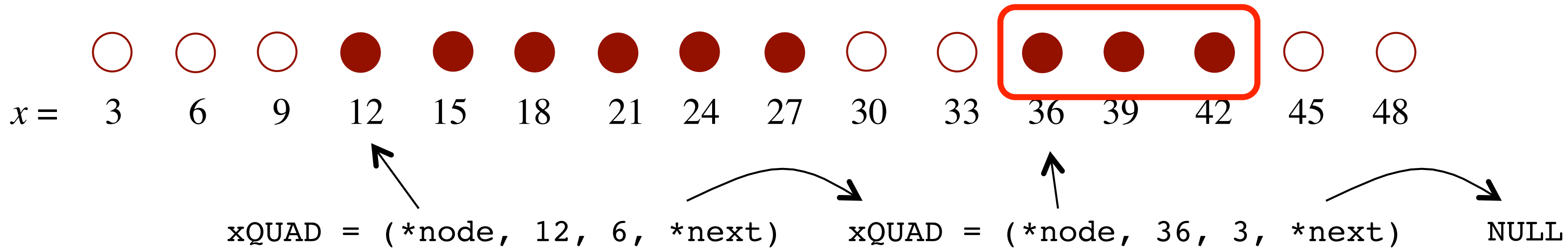

```
struct {

 float rho;
 ...
} node;
```

- handling irregular grids (1D)

quad's

the memory for each array of nodes needs to be consecutively malloc()'ed

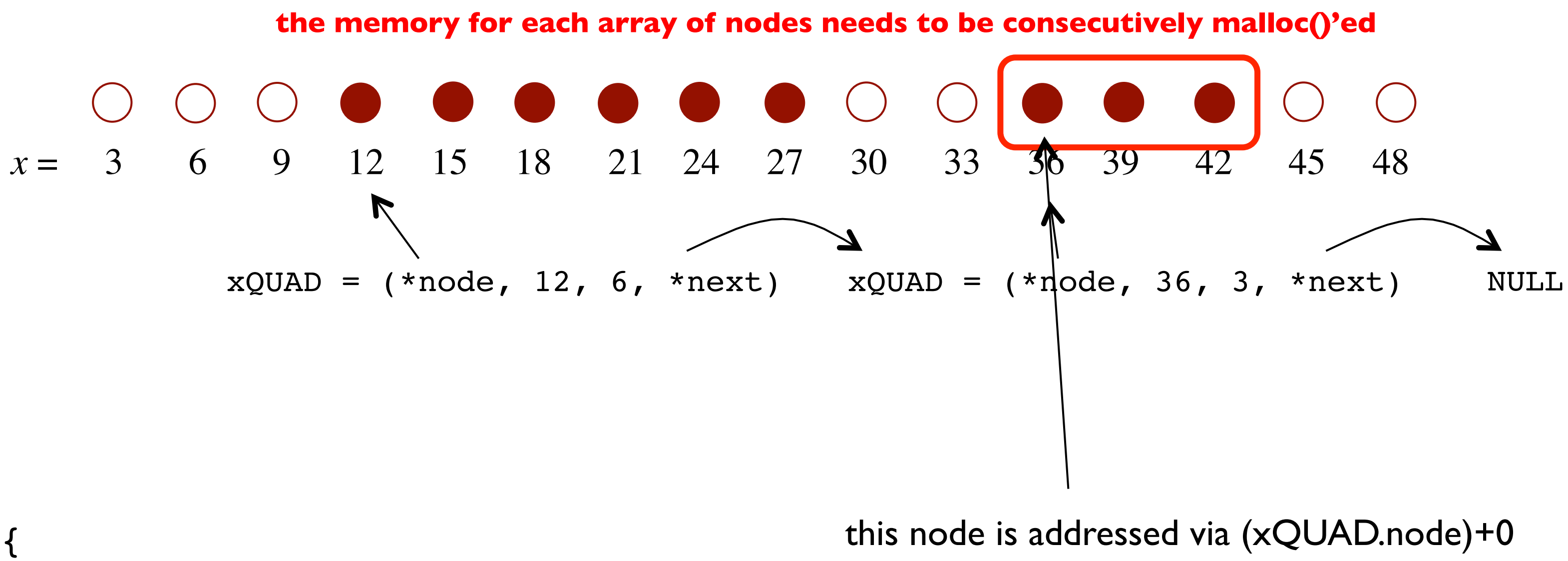



```
struct {

 float rho;
 …
} node;
```

- handling irregular grids (1D)

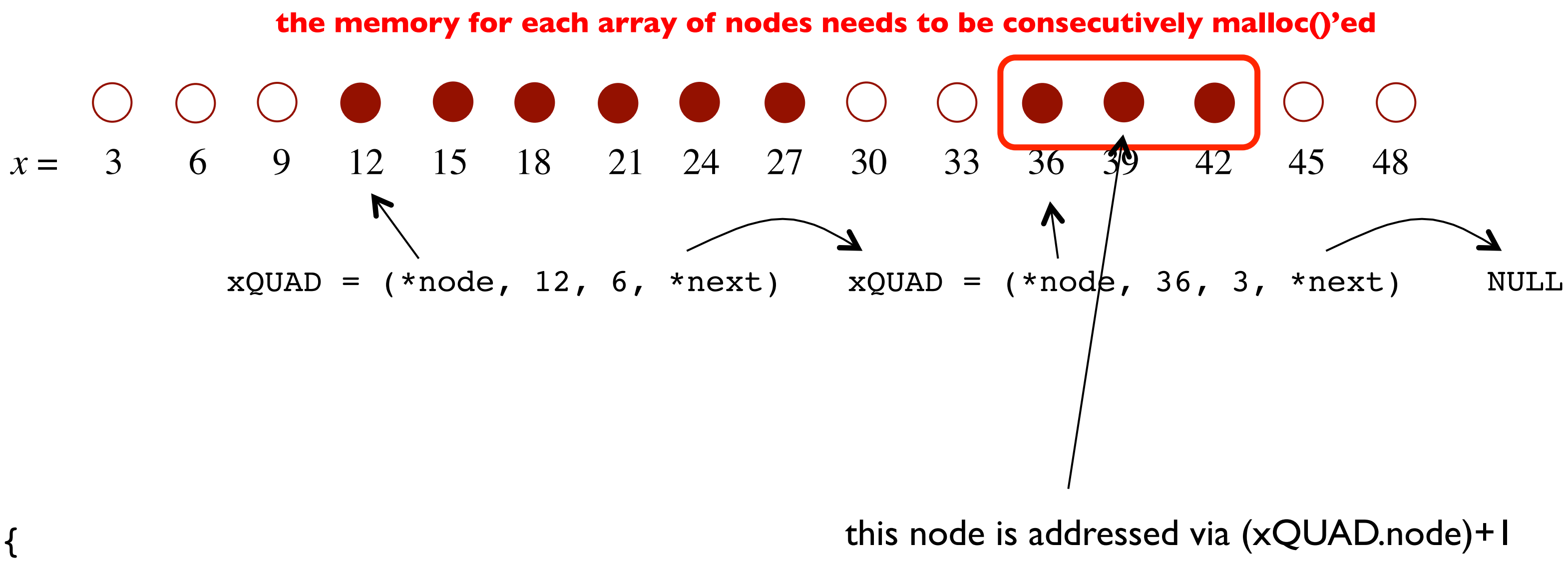
quad's




```
struct {

 float rho;
 …
} node;
```

- handling irregular grids (1D) ***quad's***

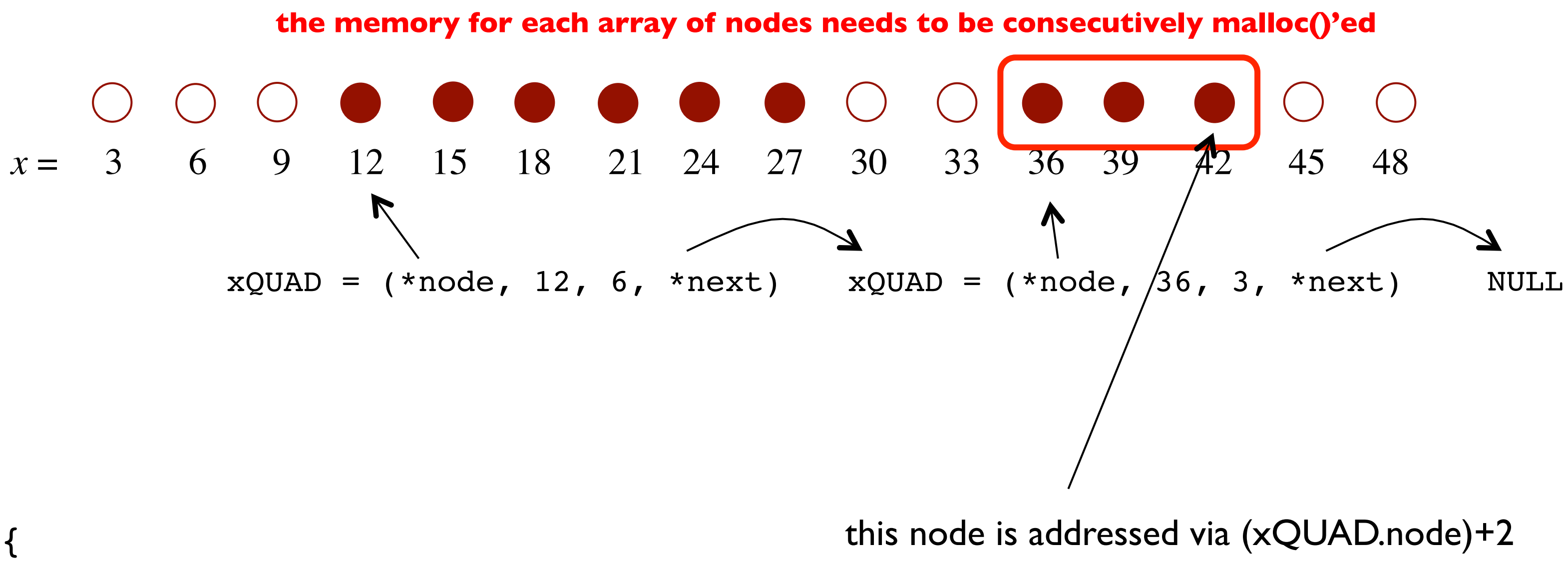



```
struct {

 float rho;
 …
} node;
```

- handling irregular grids (1D)

quad's

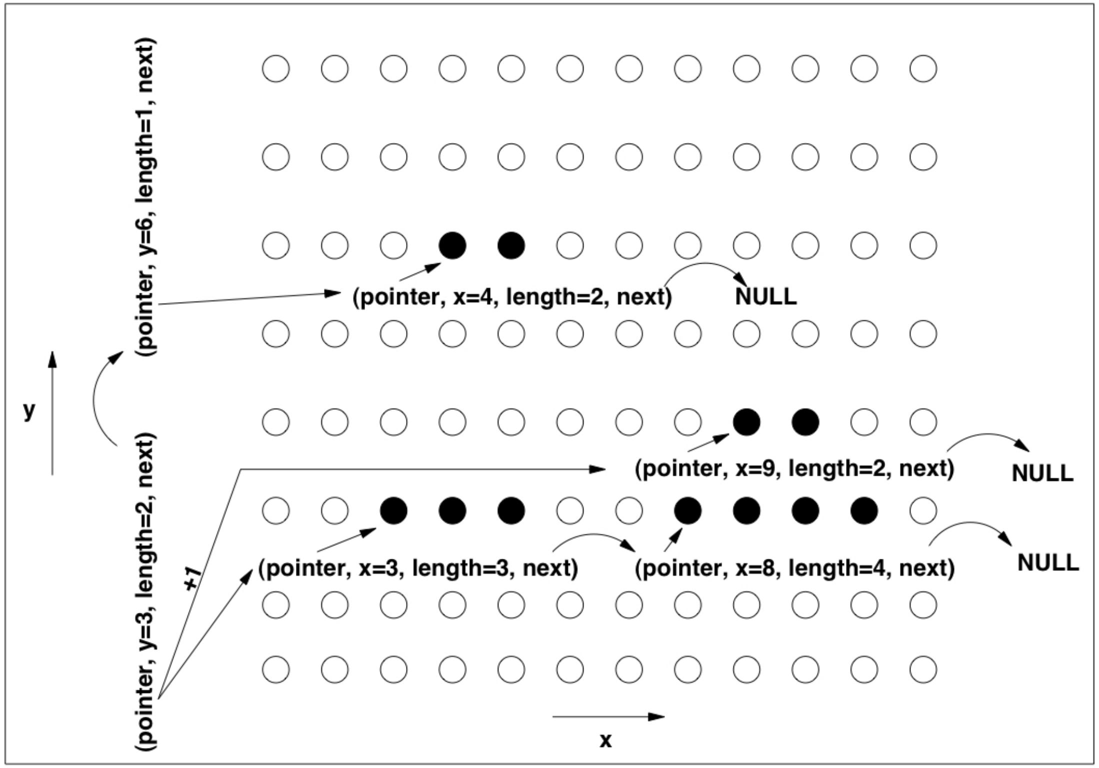



```
struct {

 float rho;
 …
} node;
```

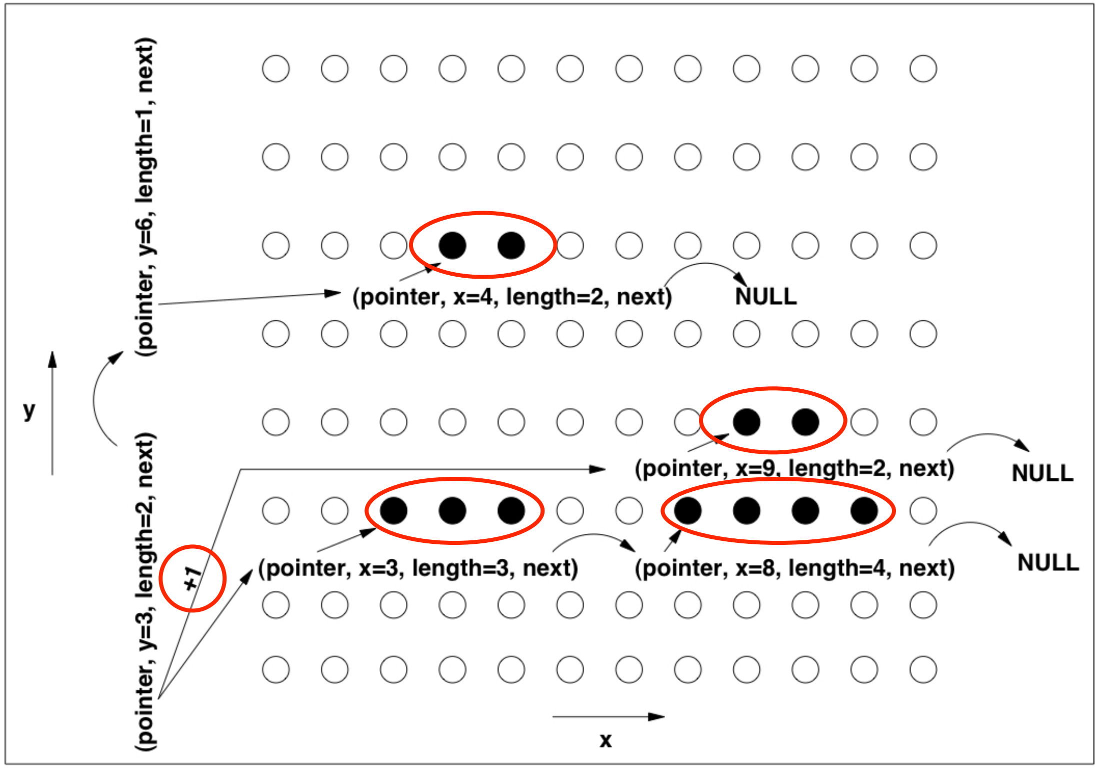
- handling irregular grids (2D)

quad's



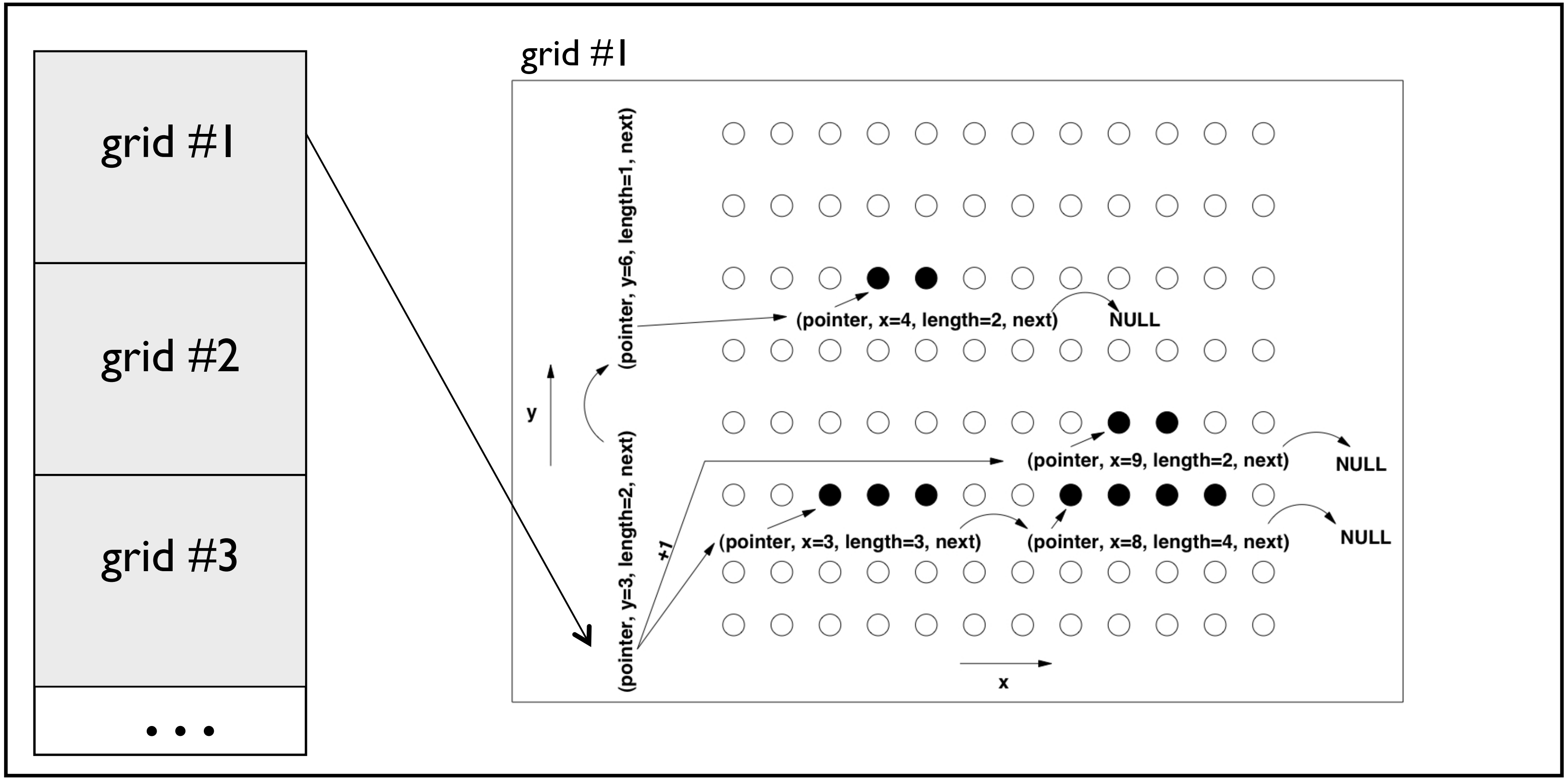
- handling irregular grids (2D)

quad's

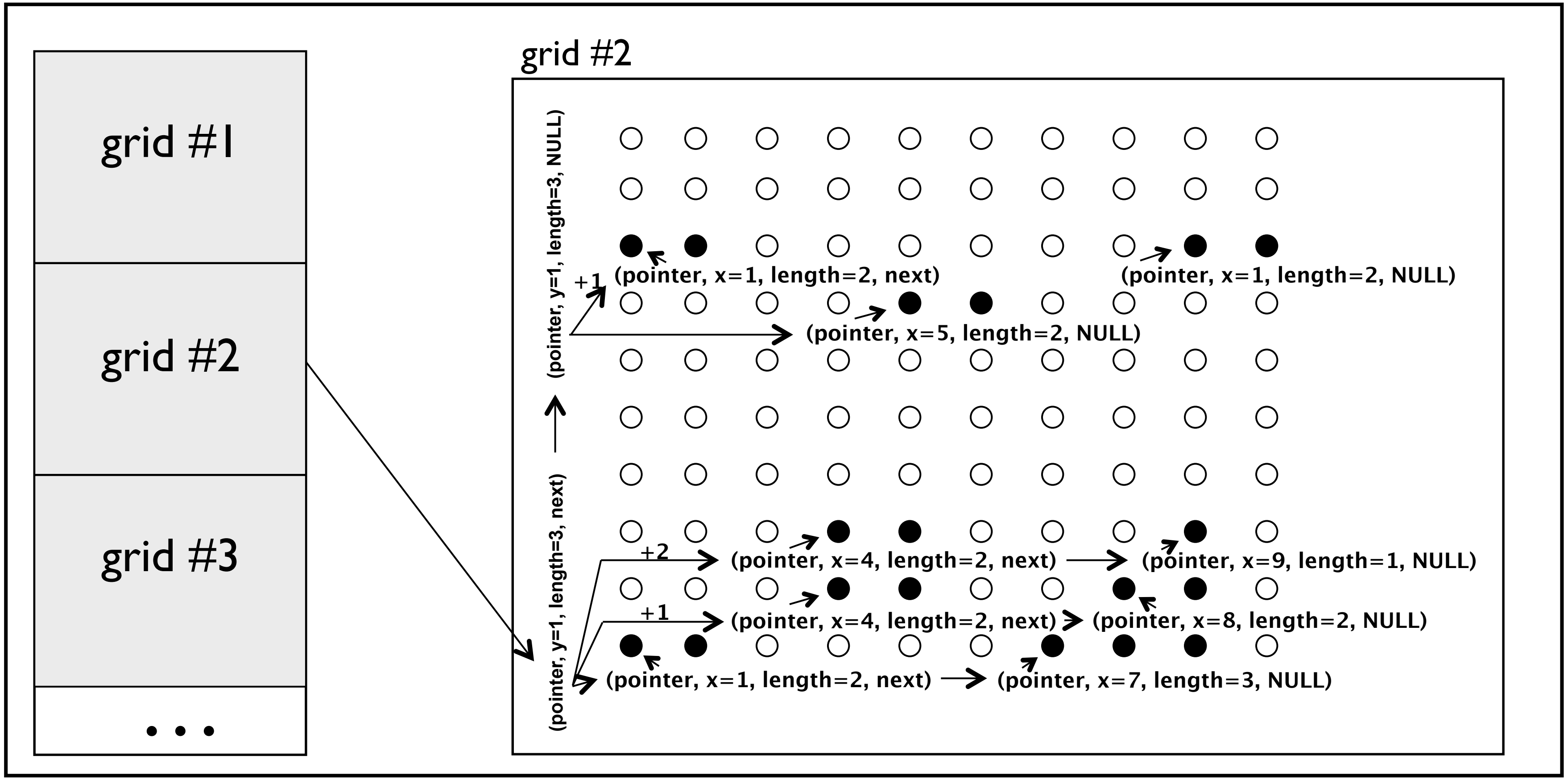


the memory for each quad array needs to be malloc()'ed consecutively!

- handling irregular grids (2D) ***quad's***
  - store "grid structures" as a consecutive memory block
  - each "grid" points to the first yQUAD which in turns gives access to all nodes



- handling irregular grids (2D) ***quad's***
  - store "grid structures" as a consecutive memory block
  - each "grid" points to the first yQUAD which in turns gives access to all nodes



- handling irregular grids (3D)

quad's

too complicated to sketch…

- handling irregular grids (3D) **quad's**
  - C-code example of how to loop over all nodes attached to a "grid"

```
for (zquad=grid.first_zquad; zquad != NULL; zquad=zquad->next) {
  for (yquad=zquad->first_yquad; yquad < yquad->pointer+yquad->length; yquad++)

    for (iyquad=yquad; iyquad != NULL; iyquad=iyquad->next) {
      for (xquad=yquad->first_xquad; xquad < xquad->pointer+xquad->length; xquad++)

        for (ixquad=xquad; ixquad != NULL; ixquad=ixquad->next) {
          for (node=ixquad->pointer; node < ixquad->x+ixquad->length; node++) {

            /* the node is at your disposal */
            density       = node->density;
            potential     = node->potential;
            forceX        = node->force[X];

            for(part=node->first_particle; part != NULL; part=part->next)
            /* use particle structure to access particle position, velocity, etc. */ }}}
```

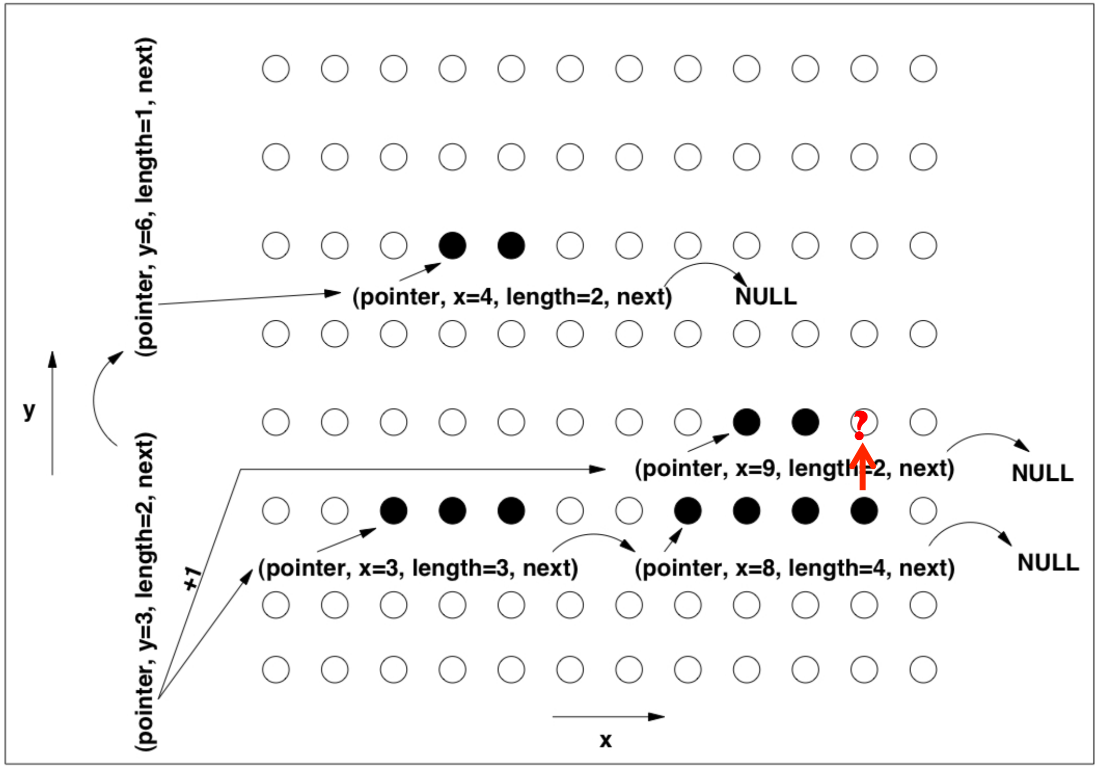
loc = **loc**ation of first quad

- handling irregular grids

quad's

  - drawback:

    no direct access to neighbouring nodes…



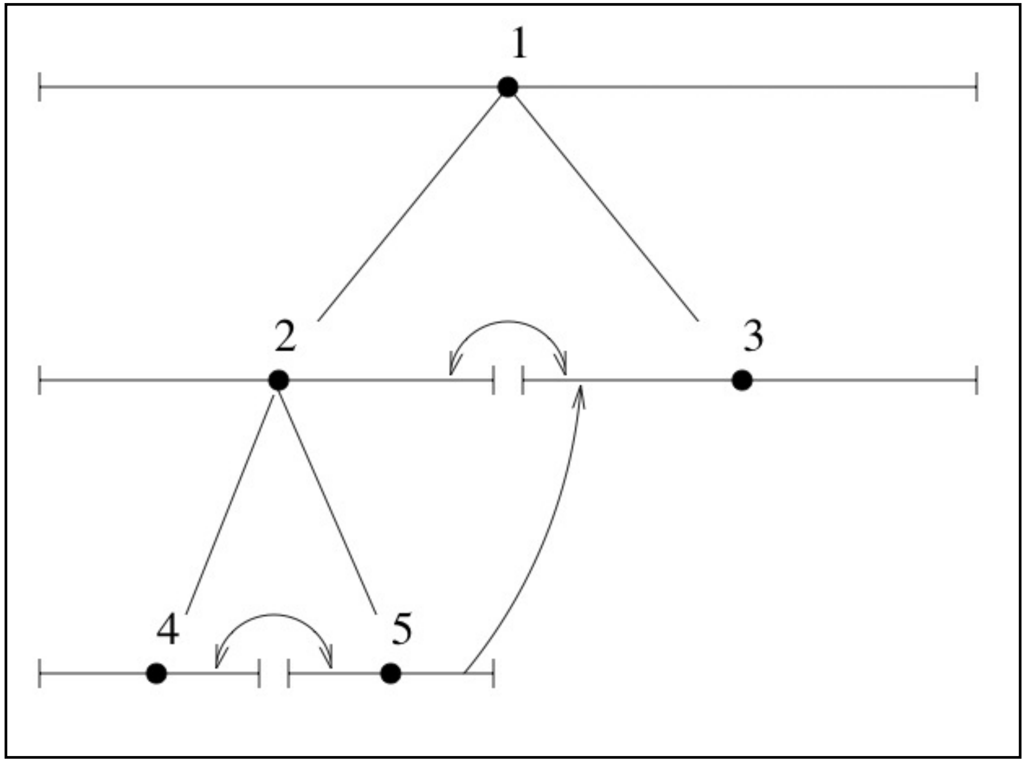
- handling irregular grids

FTT

  - other schemes possible:

```
struct {

 NODE *daughter;
 float rho;
 …
} node;
```



Fully-Threaded-Tree (FTT) by Khokhlov, 1998, J. Comp. Phy. 143, 519

(used with Andrey Kravtsov's ART code…)

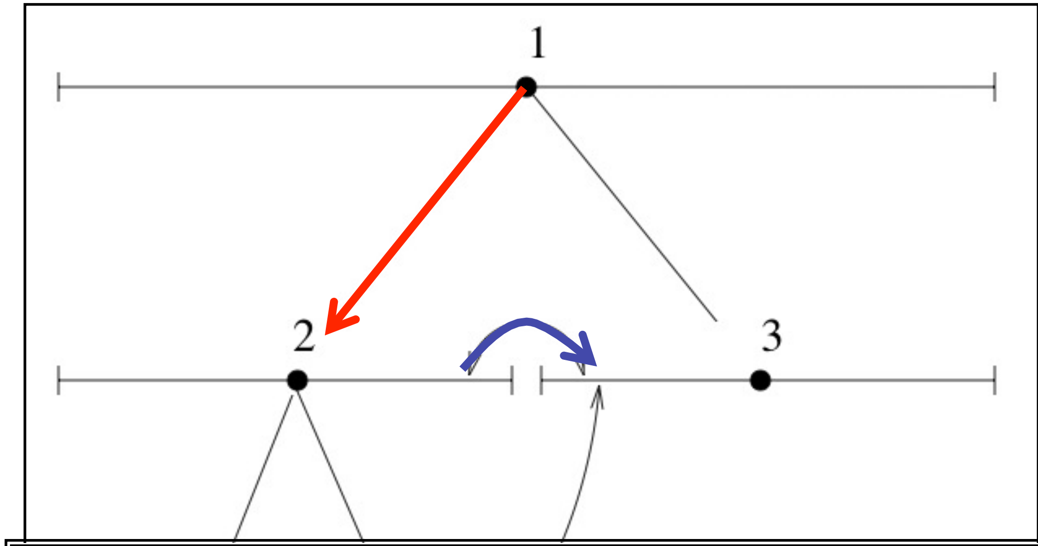
- handling irregular grids

FTT

  - other schemes possible:

```
struct {

 NODE *daughter;
 float rho;
 …
} node;
```



- **each cell stores pointers to 1st daughter**
- **daughters are malloc()'ed consecutively**

Fully-Threaded-Tree (FTT) by Khokhlov, 1998, J. Comp. Phy. 143, 519

(used with Andrey Kravtsov's ART code…)

- mesh refinements
- adaptive mesh refinement
- adaptive mesh refinement for N-body codes
- handling irregular grids
- **adaptive leap-frog integration**

- full set of equations

  - collisionless matter (e.g. dark matter)

$$\frac{d\vec{x}_{DM}}{dt} = \vec{v}_{DM}$$

$$\frac{d\vec{v}_{DM}}{dt} = -\nabla\phi$$

leap-frog integration

AMR solver

  - Poisson's equation

$$\Delta\phi = 4\pi G\rho_{tot}$$

  - collisional matter (e.g. gas)

$$\frac{\partial\rho}{\partial t} + \nabla\cdot\left(\rho\vec{v}\right) = 0$$

$$\frac{\partial\left(\rho\vec{v}\right)}{\partial t} + \nabla\cdot\left(\rho\vec{v}\otimes\vec{v} + \left(p + \frac{1}{2\mu}B^2\right)\vec{1} - \frac{1}{\mu}\vec{B}\otimes\vec{B}\right) = \rho\ \left(-\nabla\phi\right)$$

$$\frac{\partial\left(\rho E\right)}{\partial t} + \nabla\cdot\left(\left[\rho E + p + \frac{1}{2\mu}B^2\right]\vec{v} - \frac{1}{\mu}\left[\vec{v}\cdot\vec{B}\right]\vec{B}\right) = \rho\vec{v}\cdot\left(-\nabla\phi\right) + \left(\Gamma - L\right)$$

  - ideal gas equations

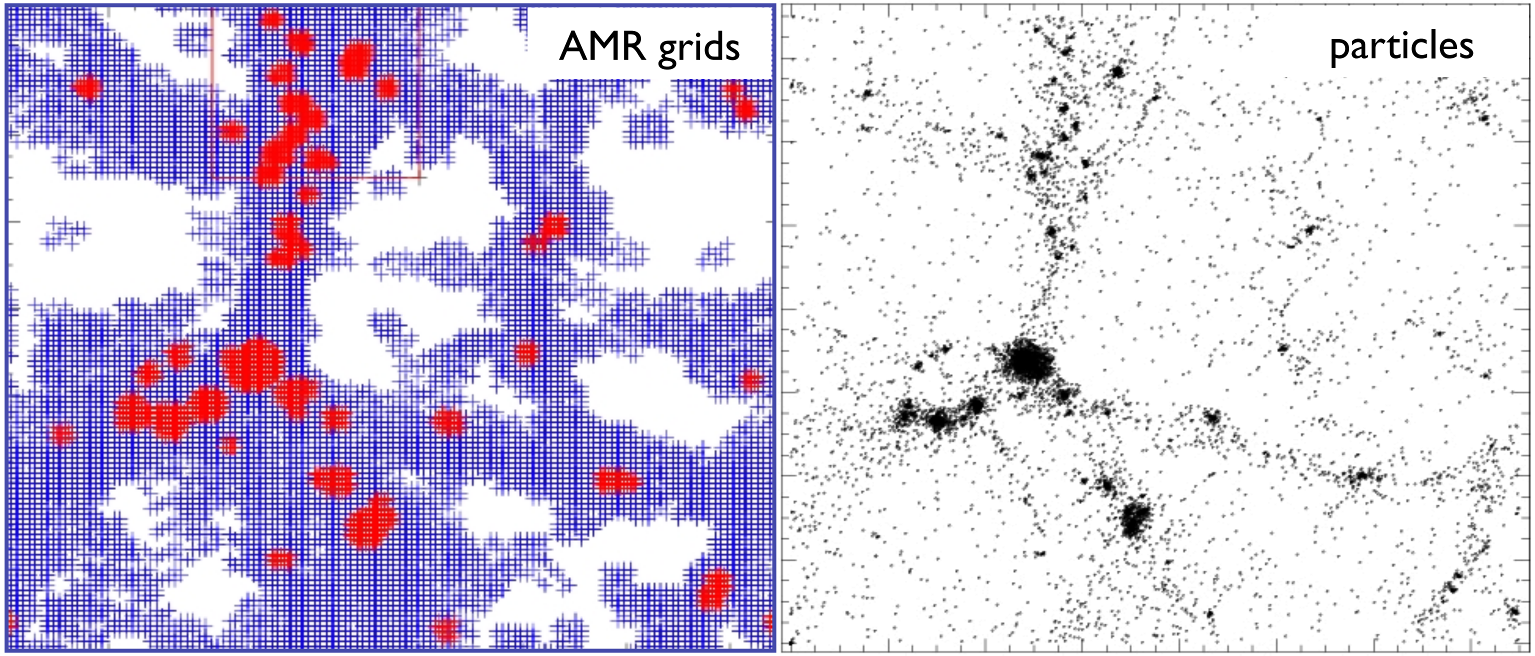
$$p = \left(\gamma - 1\right)\rho\varepsilon$$

$$\rho\varepsilon = \rho E - \frac{1}{2}\rho v^2$$

  - Maxwell's equation

$$\frac{\partial\vec{B}}{\partial t} = -\nabla\times\left(\vec{v}\times\vec{B}\right)$$

- moving particles on the AMR hierarchy

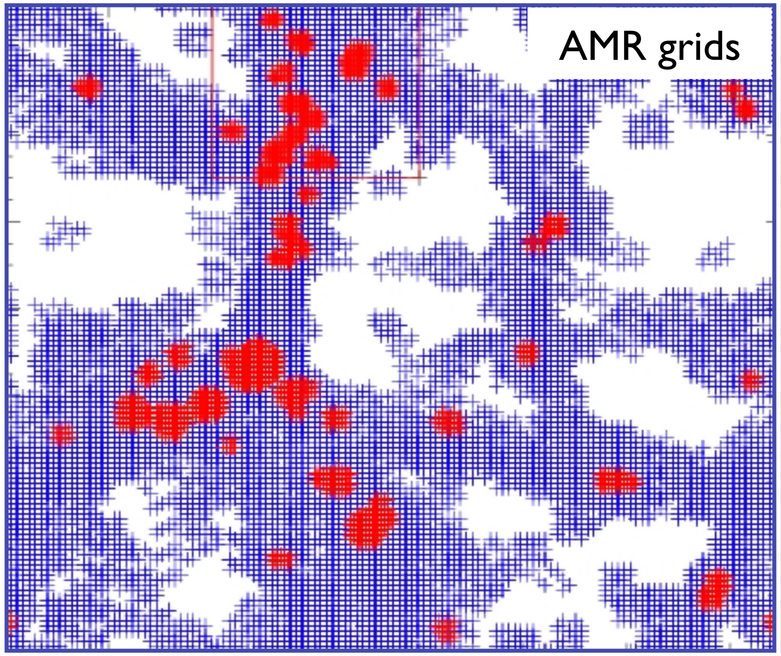


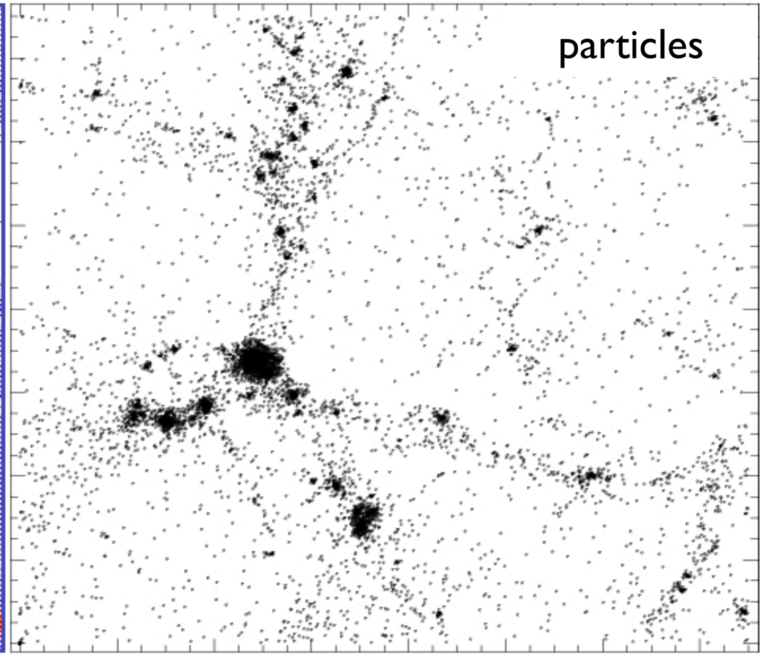
$$\Delta\phi = 4\pi G\rho_{tot}$$

$$\frac{d\vec{x}_{DM}}{dt} = \vec{v}_{DM}$$
$$\frac{d\vec{v}_{DM}}{dt} = -\nabla\phi$$

- moving particles on the AMR hierarchy

move particles on fine grids with smaller time step
to better resolve the dynamics, too!



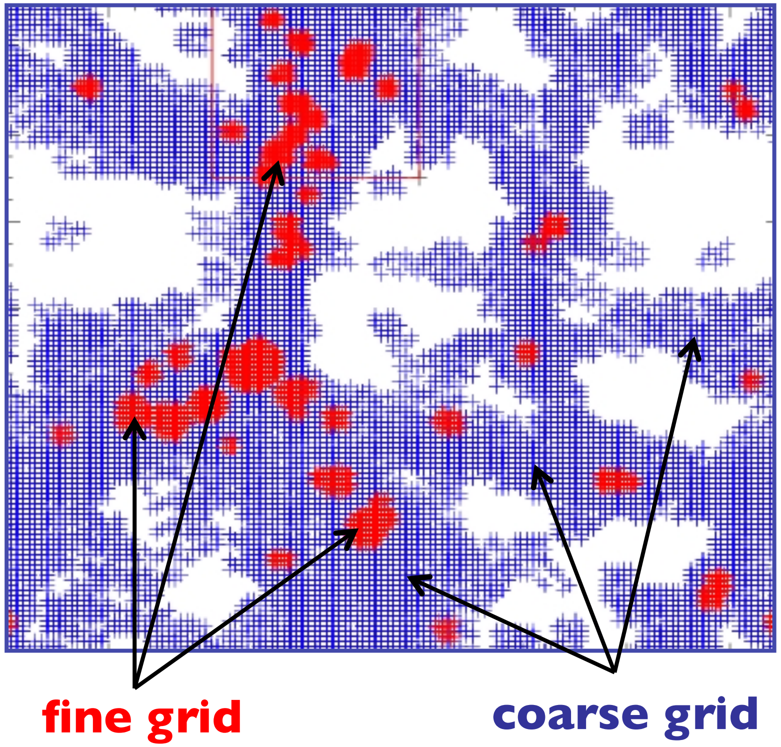


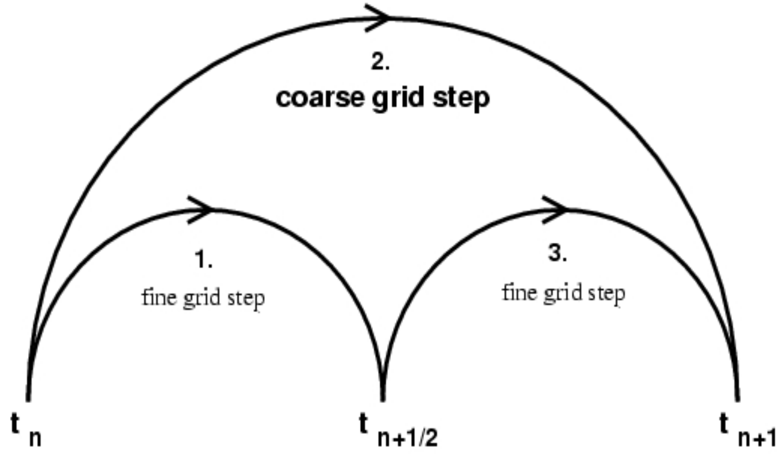
$$\Delta\phi = 4\pi G\rho_{tot}$$

$$\frac{d\vec{x}_{DM}}{dt} = \vec{v}_{DM}$$
$$\frac{d\vec{v}_{DM}}{dt} = -\nabla\phi$$

- moving particles on the AMR hierarchy

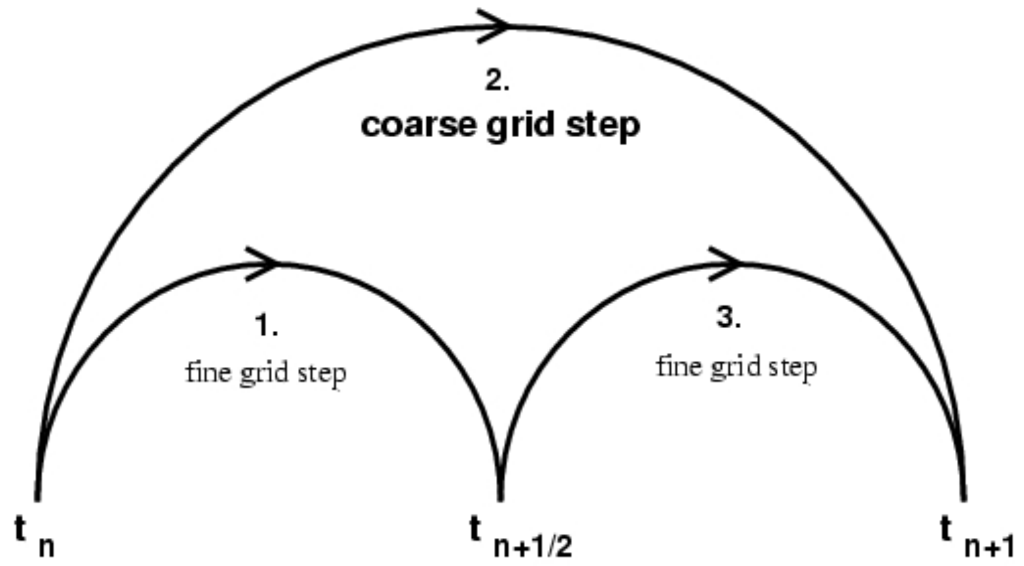
  - fully recursive approach:





- moving particles on the AMR hierarchy

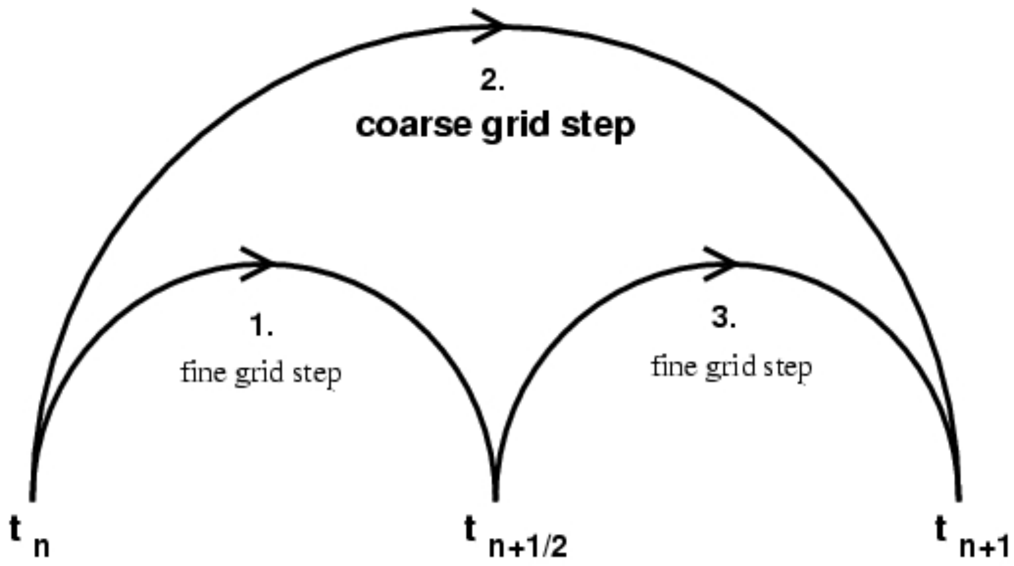
  - fully recursive approach:



Drift-Kick-Drift variant of the leap-frog integrator:

time synchronisation between different grid levels rather than "leap-frogging"!

- moving particles on the AMR hierarchy
  - fully recursive approach:



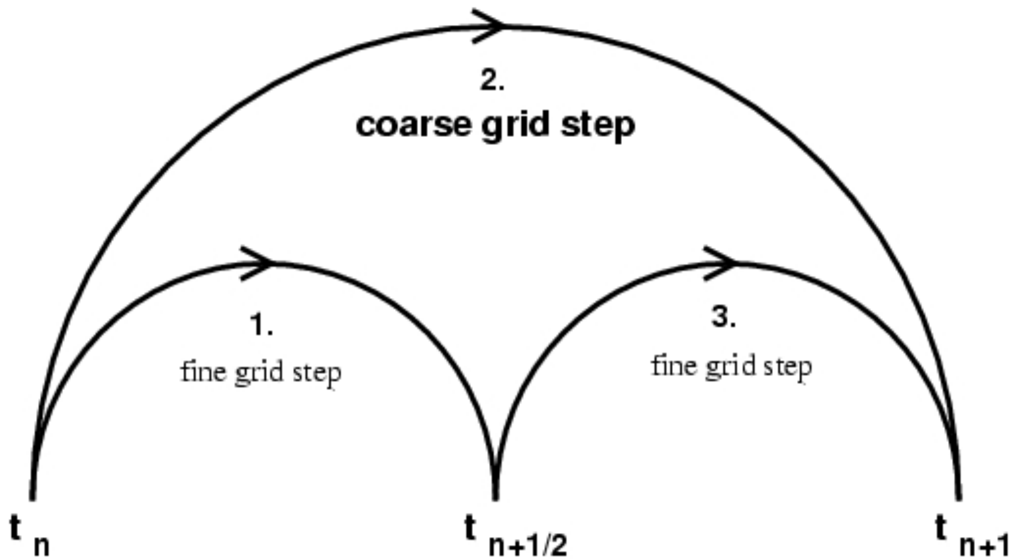
1. fine grid step:

$$\text{Drift:} \quad \vec{x}^{n+1/4} = \vec{x}^{n} + \vec{p}^{n} \int_{t_n}^{t_n+\Delta t/4} dt$$

$$\leftarrow \quad \text{Kick:} \quad \vec{p}^{n+1/2} = \vec{p}^{n} - \vec{\nabla}\Phi^{n+1/4} \int_{t_n}^{t_n+\Delta t/2} dt \quad \rightarrow$$

$$\text{Drift:} \quad \vec{x}^{n+1/2} = \vec{x}^{n+1/4} + \vec{p}^{n+1/2} \int_{t_n+\Delta t/4}^{t_n+\Delta t/2} dt$$

- moving particles on the AMR hierarchy
  - fully recursive approach:



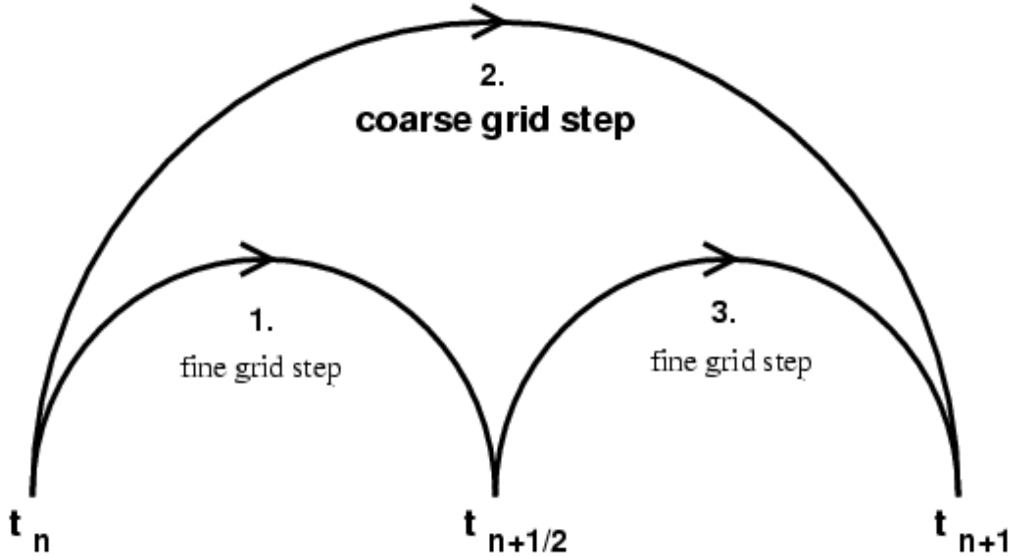
2. coarse grid step:

$$\text{Drift}: \quad \vec{x}^{n+1/2} = \vec{x}^{n} + \vec{p}^{n} \int_{t_n}^{t_n+\Delta t/2} dt$$

$$\text{Kick}: \quad \vec{p}^{n+1} = \vec{p}^{n} - \vec{\nabla}\Phi^{n+1/2} \int_{t_n}^{t_n+\Delta t} dt$$

$$\text{Drift}: \quad \vec{x}^{n+1} = \vec{x}^{n+1/2} + \vec{p}^{n+1} \int_{t_n+\Delta t/2}^{t_n+\Delta t} dt$$

- moving particles on the AMR hierarchy
  - fully recursive approach:



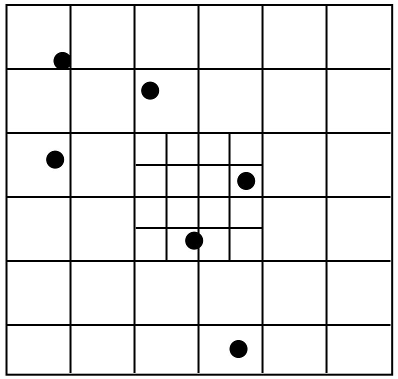
3. fine grid step:

$$\text{Drift}: \quad \vec{x}^{n+3/4} = \vec{x}^{n+1/2} + \vec{p}^{n} \int_{t_n+\Delta t/2}^{t_n+3\Delta t/4} dt$$

$$\leftarrow \quad \text{Kick}: \quad \vec{p}^{n+1} = \vec{p}^{n+1/2} - \vec{\nabla}\Phi^{n+3/4} \int_{t_n+\Delta t/2}^{t_n+\Delta t} dt \quad \rightarrow$$

$$\text{Drift}: \quad \vec{x}^{n+1} = \vec{x}^{n+3/4} + \vec{p}^{n+1} \int_{t_n+3\Delta t/4}^{t_n+\Delta t} dt$$

- moving particles on the AMR hierarchy



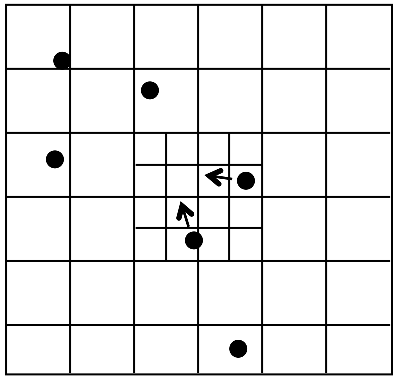
- moving particles on the AMR hierarchy

1. fine grid DKD step:

$$\text{Drift}: \quad \vec{x}^{n+1/4} = \vec{x}^{n} + \vec{p}^{n} \int_{t_n}^{t_n+\Delta t/4} dt$$

$$\text{Kick}: \quad \vec{p}^{n+1/2} = \vec{p}^{n} - \vec{\nabla}\Phi^{n+1/4} \int_{t_n}^{t_n+\Delta t/2} dt$$

$$\text{Drift}: \quad \vec{x}^{n+1/2} = \vec{x}^{n+1/4} + \vec{p}^{n+1/2} \int_{t_n+\Delta t/4}^{t_n+\Delta t/2} dt$$



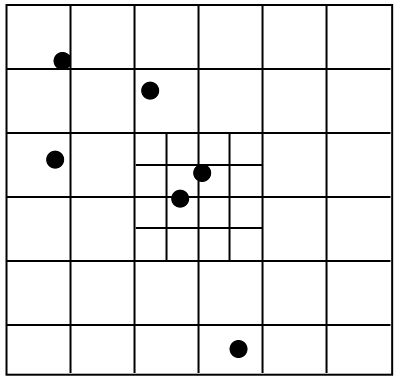
- moving particles on the AMR hierarchy

1. fine grid DKD step:

$$\text{Drift}: \quad \vec{x}^{n+1/4} = \vec{x}^{n} + \vec{p}^{n} \int_{t_n}^{t_n+\Delta t/4} dt$$

$$\text{Kick}: \quad \vec{p}^{n+1/2} = \vec{p}^{n} - \vec{\nabla}\Phi^{n+1/4} \int_{t_n}^{t_n+\Delta t/2} dt$$

$$\text{Drift}: \quad \vec{x}^{n+1/2} = \vec{x}^{n+1/4} + \vec{p}^{n+1/2} \int_{t_n+\Delta t/4}^{t_n+\Delta t/2} dt$$



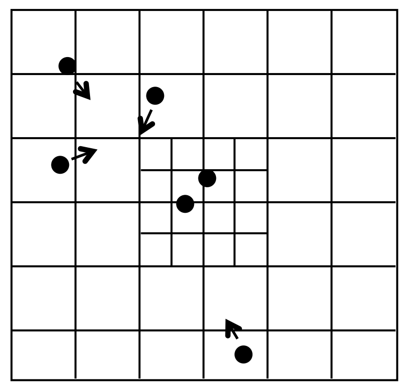
- moving particles on the AMR hierarchy

2. coarse grid DKD step:

$$\text{Drift}: \quad \vec{x}^{n+1/2} = \vec{x}^{n} + \vec{p}^{n} \int_{t_n}^{t_n+\Delta t/2} dt$$

$$\text{Kick}: \quad \vec{p}^{n+1} = \vec{p}^{n} - \vec{\nabla}\Phi^{n+1/2} \int_{t_n}^{t_n+\Delta t} dt$$

$$\text{Drift}: \quad \vec{x}^{n+1} = \vec{x}^{n+1/2} + \vec{p}^{n+1} \int_{t_n+\Delta t/2}^{t_n+\Delta t} dt$$



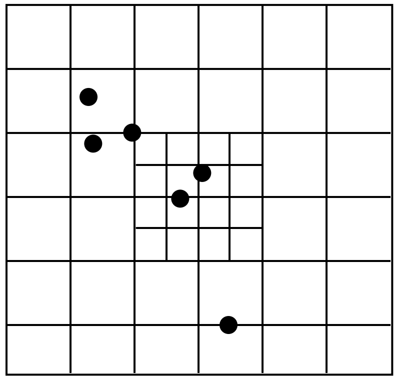
- moving particles on the AMR hierarchy

2. coarse grid DKD step:

$$\text{Drift}: \quad \vec{x}^{n+1/2} = \vec{x}^{n} + \vec{p}^{n} \int_{t_n}^{t_n+\Delta t/2} dt$$

$$\text{Kick}: \quad \vec{p}^{n+1} = \vec{p}^{n} - \vec{\nabla}\Phi^{n+1/2} \int_{t_n}^{t_n+\Delta t} dt$$

$$\text{Drift}: \quad \vec{x}^{n+1} = \vec{x}^{n+1/2} + \vec{p}^{n+1} \int_{t_n+\Delta t/2}^{t_n+\Delta t} dt$$



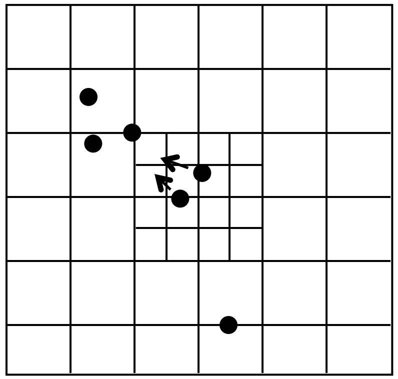
- moving particles on the AMR hierarchy

3. fine grid DKD step:

$$\text{Drift}: \quad \vec{x}^{n+3/4} = \vec{x}^{n+1/2} + \vec{p}^{n} \int_{t_n+\Delta t/2}^{t_n+3\Delta t/4} dt$$

$$\text{Kick}: \quad \vec{p}^{n+1} = \vec{p}^{n+1/2} - \vec{\nabla}\Phi^{n+3/4} \int_{t_n+\Delta t/2}^{t_n+\Delta t} dt$$

$$\text{Drift}: \quad \vec{x}^{n+1} = \vec{x}^{n+3/4} + \vec{p}^{n+1} \int_{t_n+3\Delta t/4}^{t_n+\Delta t} dt$$



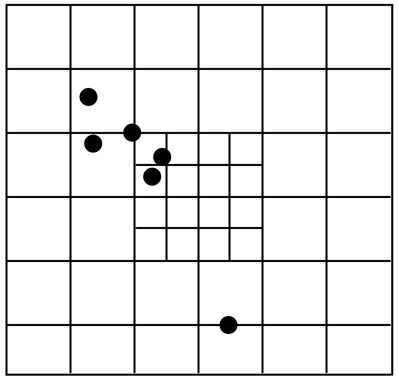
- moving particles on the AMR hierarchy

3. fine grid DKD step:

$$\text{Drift}: \quad \vec{x}^{n+3/4} = \vec{x}^{n+1/2} + \vec{p}^{n} \int_{t_n+\Delta t/2}^{t_n+3\Delta t/4} dt$$

$$\text{Kick}: \quad \vec{p}^{n+1} = \vec{p}^{n+1/2} - \vec{\nabla}\Phi^{n+3/4} \int_{t_n+\Delta t/2}^{t_n+\Delta t} dt$$

$$\text{Drift}: \quad \vec{x}^{n+1} = \vec{x}^{n+3/4} + \vec{p}^{n+1} \int_{t_n+3\Delta t/4}^{t_n+\Delta t} dt$$



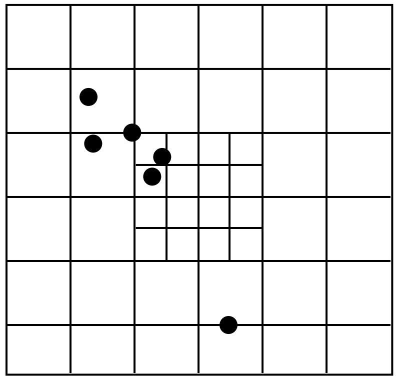
- moving particles on the AMR hierarchy

3. fine grid DKD step:

$$\text{Drift:}\quad \vec{x}^{n+3/4} = \vec{x}^{n+1/2} + \vec{p}^{n} \int_{t_n+\Delta t/2}^{t_n+3\Delta t/4} dt$$

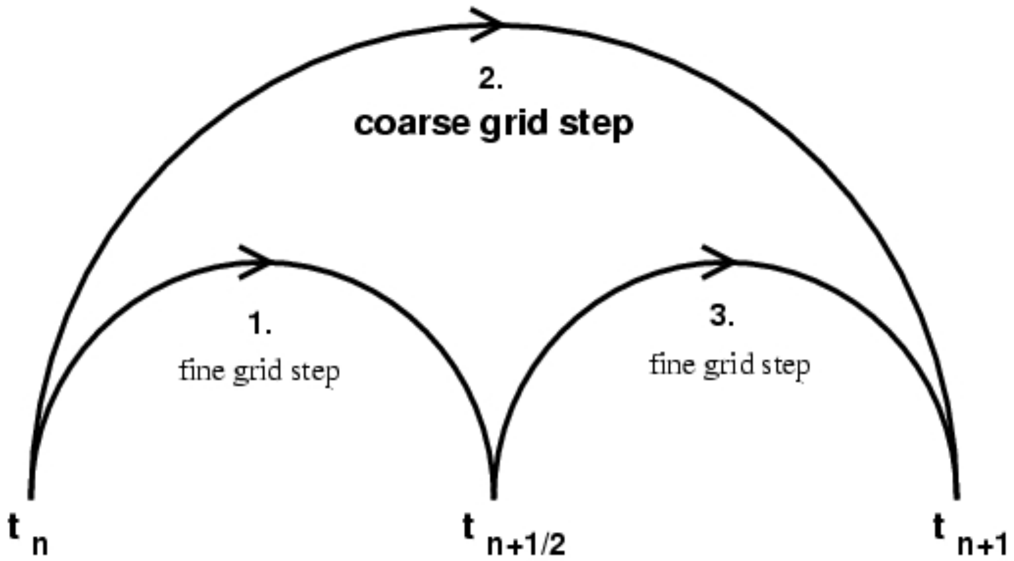
$$\text{Kick:}\quad \vec{p}^{n+1} = \vec{p}^{n+1/2} - \vec{\nabla}\Phi^{n+3/4} \int_{t_n+\Delta t/2}^{t_n+\Delta t} dt$$

$$\text{Drift:}\quad \vec{x}^{n+1} = \vec{x}^{n+3/4} + \vec{p}^{n+1} \int_{t_n+3\Delta t/4}^{t_n+\Delta t} dt$$



what about particles crossing grid boundaries?

- moving particles on the AMR hierarchy
  - particles crossing grid boundaries

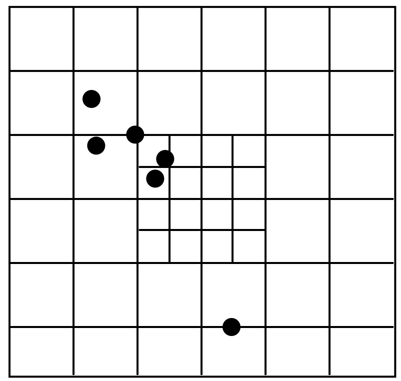


$$1.\ \text{Drift}: \quad \vec{x}^{n+1/4} = \vec{x}^{n} + \vec{p}^{n} \int_{t_n}^{t_n+\Delta t/4} dt$$

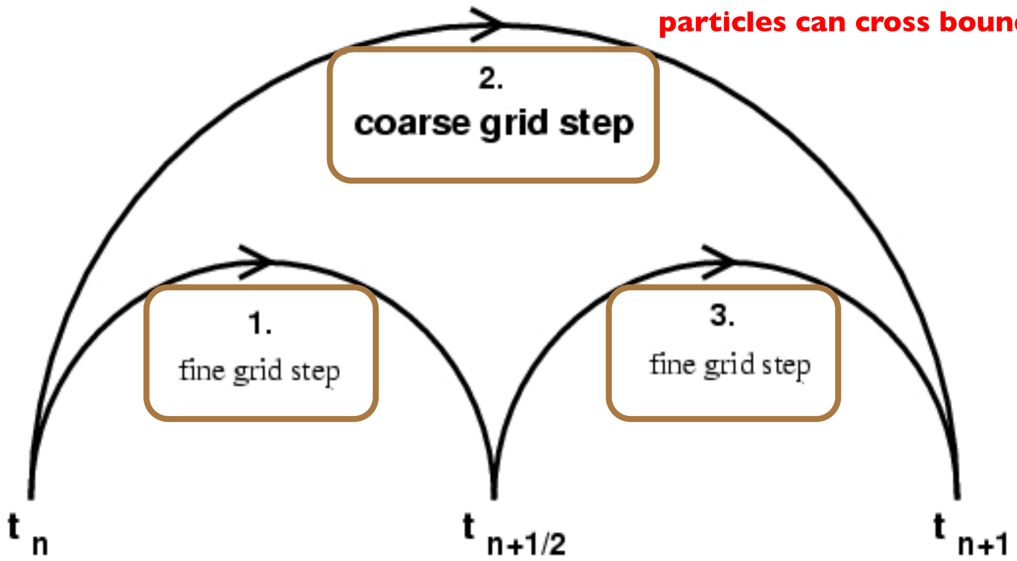
$$2.\ \text{Drift}: \quad \vec{x}^{n+1/2} = \vec{x}^{n+1/4} + \vec{p}^{n+1/2} \int_{t_n+\Delta t/4}^{t_n+\Delta t/2} dt$$

$$3.\ \text{Drift}: \quad \vec{x}^{n+3/4} = \vec{x}^{n+1/2} + \vec{p}^{n} \int_{t_n+\Delta t/2}^{t_n+3\Delta t/4} dt$$

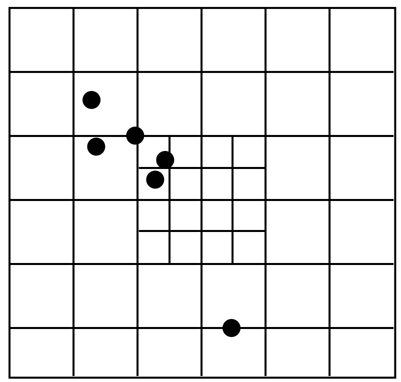
$$4.\ \text{Drift}: \quad \vec{x}^{n+1} = \vec{x}^{n+3/4} + \vec{p}^{n+1} \int_{t_n+3\Delta t/4}^{t_n+\Delta t} dt$$



- moving particles on the AMR hierarchy
    - particles crossing grid boundaries



particles can cross boundary during any of these steps...



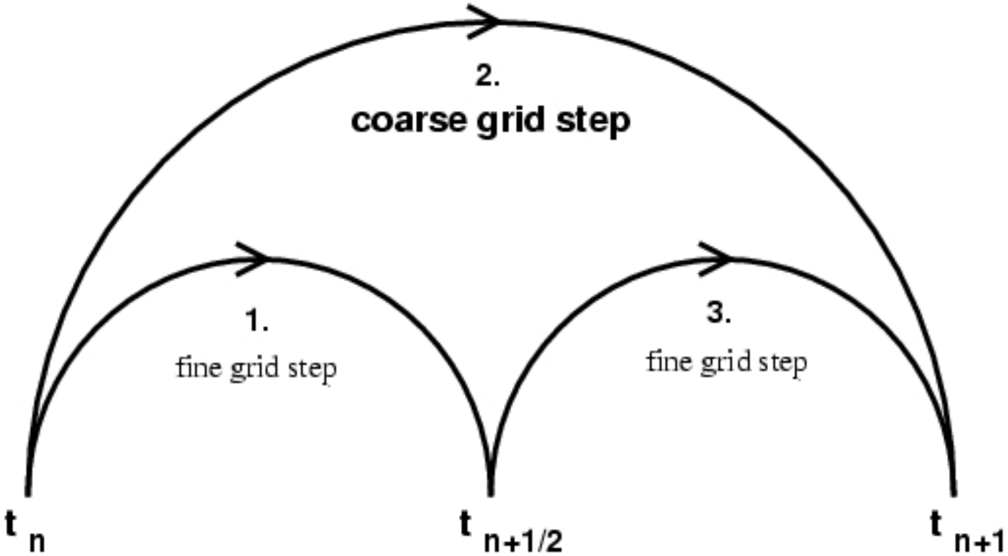
1. Drift: $\vec{x}^{n+1/4} = \vec{x}^{n} + \vec{p}^{n} \int_{t_n}^{t_n+\Delta t/4} dt$

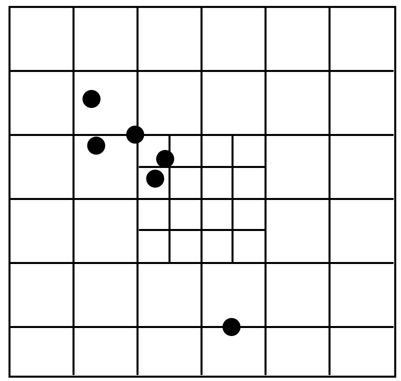
2. Drift: $\vec{x}^{n+1/2} = \vec{x}^{n+1/4} + \vec{p}^{n+1/2} \int_{t_n+\Delta t/4}^{t_n+\Delta t/2} dt$

3. Drift: $\vec{x}^{n+3/4} = \vec{x}^{n+1/2} + \vec{p}^{n} \int_{t_n+\Delta t/2}^{t_n+3\Delta t/4} dt$

4. Drift: $\vec{x}^{n+1} = \vec{x}^{n+3/4} + \vec{p}^{n+1} \int_{t_n+3\Delta t/4}^{t_n+\Delta t} dt$

- moving particles on the AMR hierarchy
  - particles crossing grid boundaries





1. Drift: $\vec{x}^{n+1/4} = \vec{x}^{n} + \vec{p}^{n} \int_{t_n}^{t_n+\Delta t/4} dt$

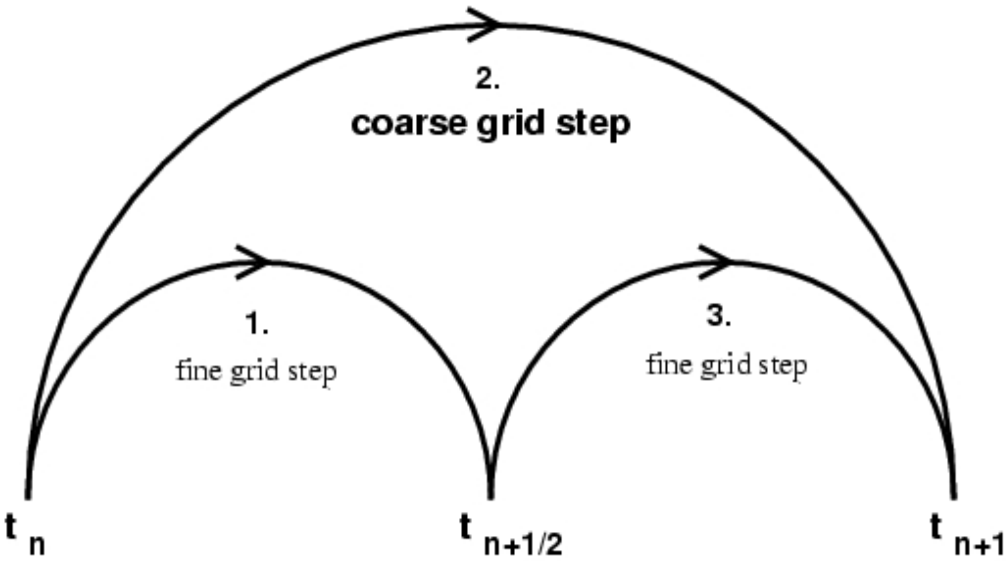
2. Drift: $\vec{x}^{n+1/2} = \vec{x}^{n+1/4} + \vec{p}^{n+1/2} \int_{t_n+\Delta t/4}^{t_n+\Delta t/2} dt$

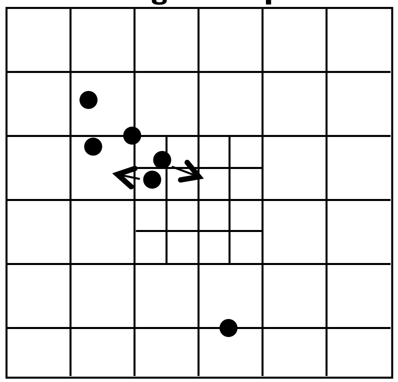
3. Drift: $\vec{x}^{n+3/4} = \vec{x}^{n+1/2} + \vec{p}^{n} \int_{t_n+\Delta t/2}^{t_n+3\Delta t/4} dt$

4. Drift: $\vec{x}^{n+1} = \vec{x}^{n+3/4} + \vec{p}^{n+1} \int_{t_n+3\Delta t/4}^{t_n+\Delta t} dt$

un-drift and move with coarse grid time step to t_{n+1}...

- moving particles on the AMR hierarchy
  - particles crossing grid boundaries





1. Drift: $$\vec{x}^{n+1/4} = \vec{x}^{n} + \vec{p}^{n} \int_{t_n}^{t_n+\Delta t/4} dt$$

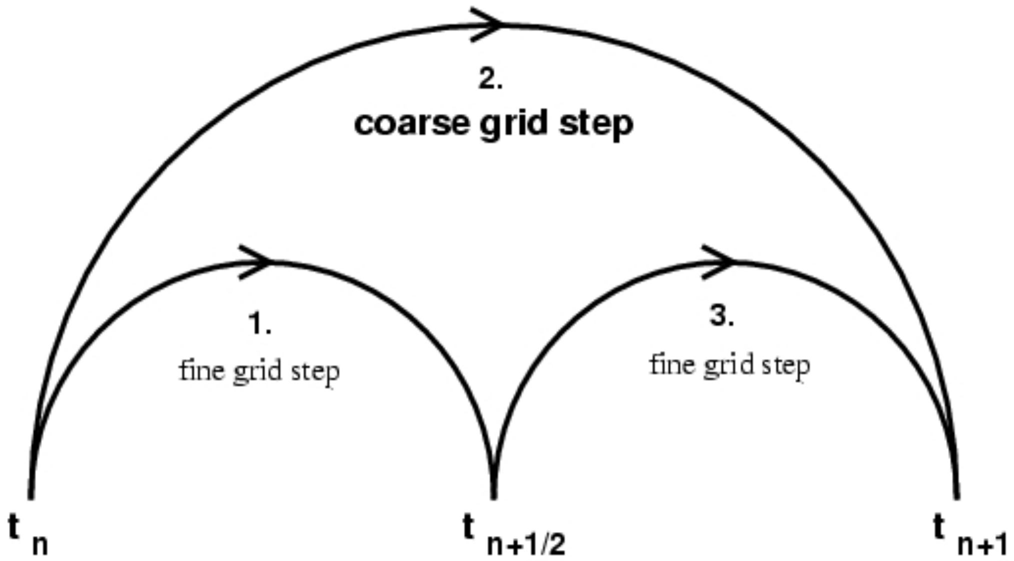
2. Drift: $$\vec{x}^{n+1/2} = \vec{x}^{n+1/4} + \vec{p}^{n+1/2} \int_{t_n+\Delta t/4}^{t_n+\Delta t/2} dt$$

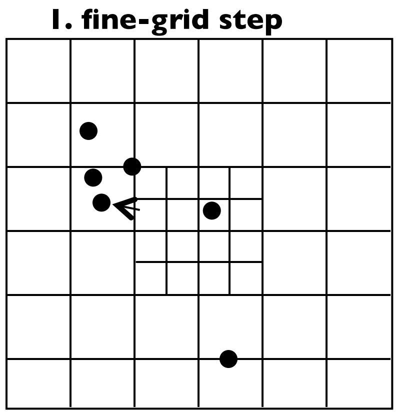
3. Drift: $$\vec{x}^{n+3/4} = \vec{x}^{n+1/2} + \vec{p}^{n} \int_{t_n+\Delta t/2}^{t_n+3\Delta t/4} dt$$

4. Drift: $$\vec{x}^{n+1} = \vec{x}^{n+3/4} + \vec{p}^{n+1} \int_{t_n+3\Delta t/4}^{t_n+\Delta t} dt$$

un-drift and move with coarse grid time step to t_{n+1}...

- moving particles on the AMR hierarchy
  - particles crossing grid boundaries





$$1.\ \text{Drift}: \quad \vec{x}^{n+1/4} = \vec{x}^{n} + \vec{p}^{n} \int_{t_n}^{t_n+\Delta t/4} dt$$

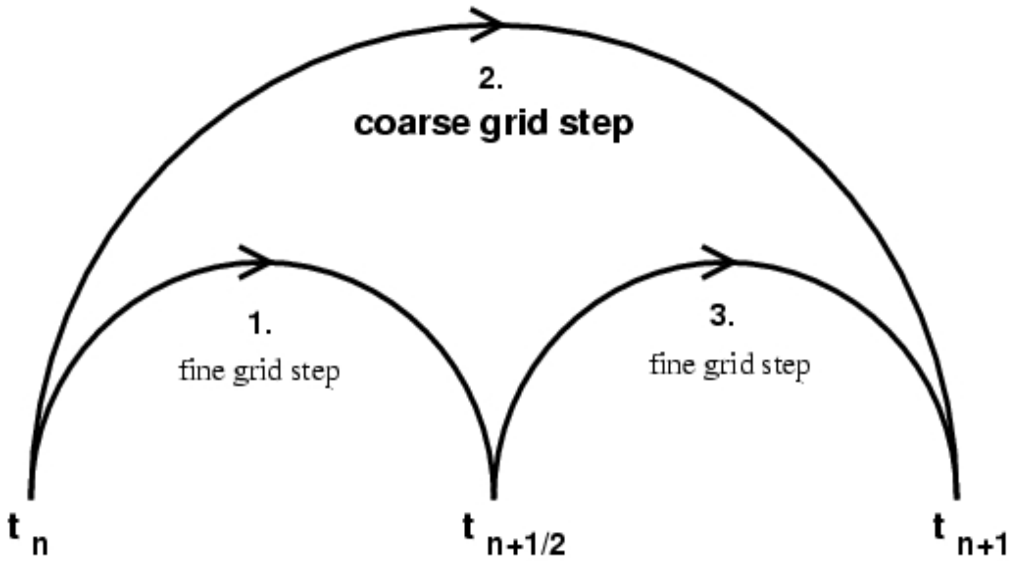
$$2.\ \text{Drift}: \quad \vec{x}^{n+1/2} = \vec{x}^{n+1/4} + \vec{p}^{n+1/2} \int_{t_n+\Delta t/4}^{t_n+\Delta t/2} dt$$

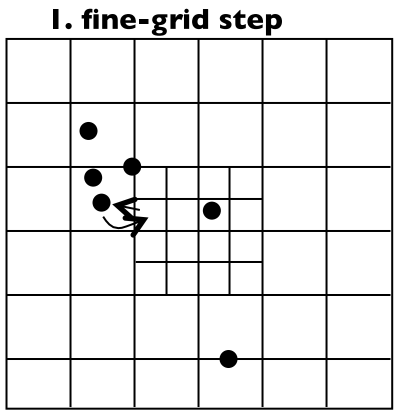
$$3.\ \text{Drift}: \quad \vec{x}^{n+3/4} = \vec{x}^{n+1/2} + \vec{p}^{n} \int_{t_n+\Delta t/2}^{t_n+3\Delta t/4} dt$$

$$4.\ \text{Drift}: \quad \vec{x}^{n+1} = \vec{x}^{n+3/4} + \vec{p}^{n+1} \int_{t_n+3\Delta t/4}^{t_n+\Delta t} dt$$

un-drift and move with coarse grid time step to t_{n+1}...

- moving particles on the AMR hierarchy
    - particles crossing grid boundaries





1. Drift: $\vec{x}^{n+1/4} = \vec{x}^{n} + \vec{p}^{n} \int_{t_n}^{t_n+\Delta t/4} dt$

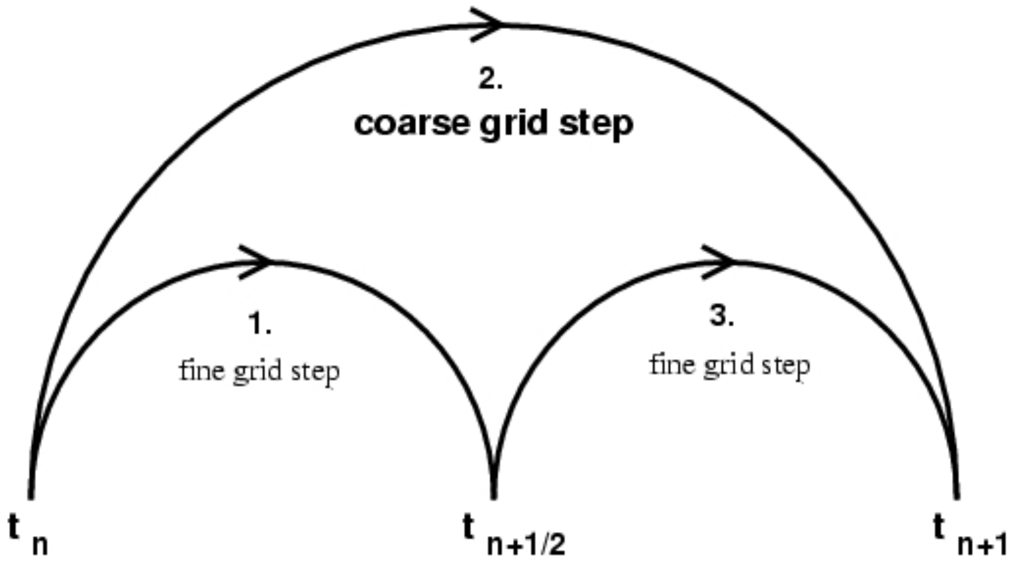
2. Drift: $\vec{x}^{n+1/2} = \vec{x}^{n+1/4} + \vec{p}^{n+1/2} \int_{t_n+\Delta t/4}^{t_n+\Delta t/2} dt$

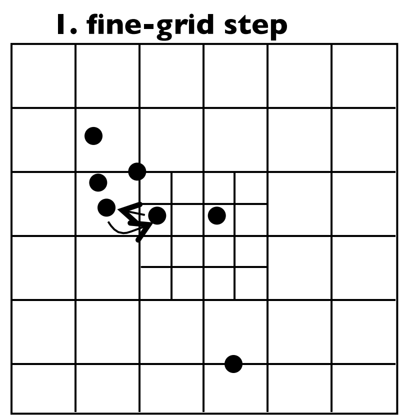
3. Drift: $\vec{x}^{n+3/4} = \vec{x}^{n+1/2} + \vec{p}^{n} \int_{t_n+\Delta t/2}^{t_n+3\Delta t/4} dt$

4. Drift: $\vec{x}^{n+1} = \vec{x}^{n+3/4} + \vec{p}^{n+1} \int_{t_n+3\Delta t/4}^{t_n+\Delta t} dt$

un-drift and move with coarse grid time step to t_{n+1}...

- moving particles on the AMR hierarchy
    - particles crossing grid boundaries





1. Drift: $\vec{x}^{n+1/4} = \vec{x}^{n} + \vec{p}^{n} \int_{t_n}^{t_n+\Delta t/4} dt$

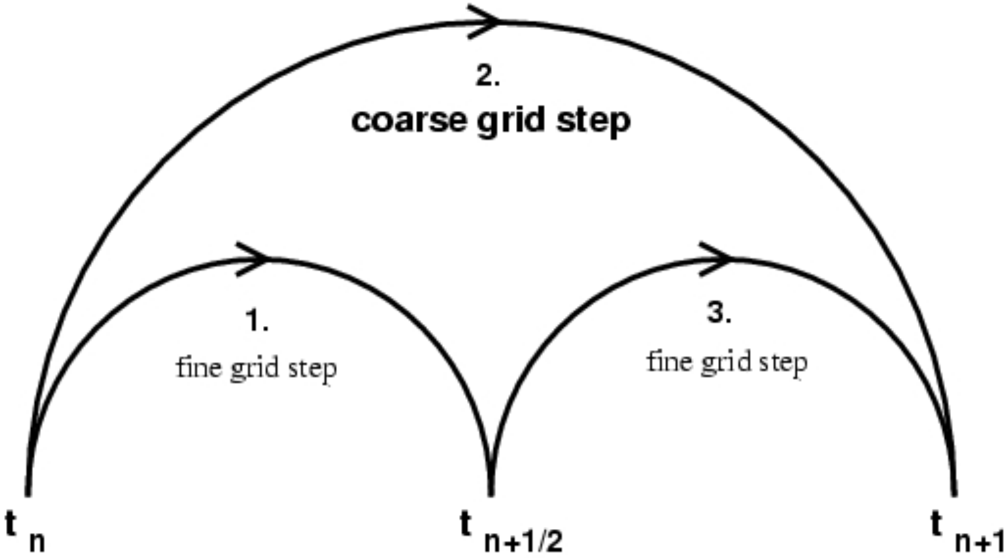
2. Drift: $\vec{x}^{n+1/2} = \vec{x}^{n+1/4} + \vec{p}^{n+1/2} \int_{t_n+\Delta t/4}^{t_n+\Delta t/2} dt$

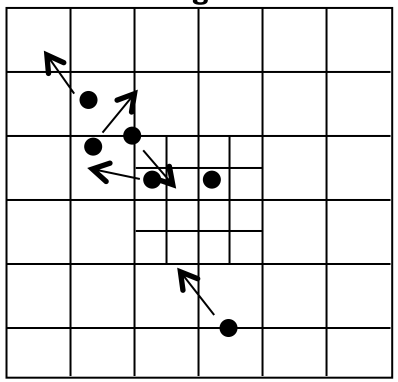
3. Drift: $\vec{x}^{n+3/4} = \vec{x}^{n+1/2} + \vec{p}^{n} \int_{t_n+\Delta t/2}^{t_n+3\Delta t/4} dt$

4. Drift: $\vec{x}^{n+1} = \vec{x}^{n+3/4} + \vec{p}^{n+1} \int_{t_n+3\Delta t/4}^{t_n+\Delta t} dt$

un-drift and move with coarse grid time step to t_{n+1}...

- moving particles on the AMR hierarchy
    - particles crossing grid boundaries





1. Drift: $\vec{x}^{n+1/4} = \vec{x}^{n} + \vec{p}^{n} \int_{t_n}^{t_n+\Delta t/4} dt$

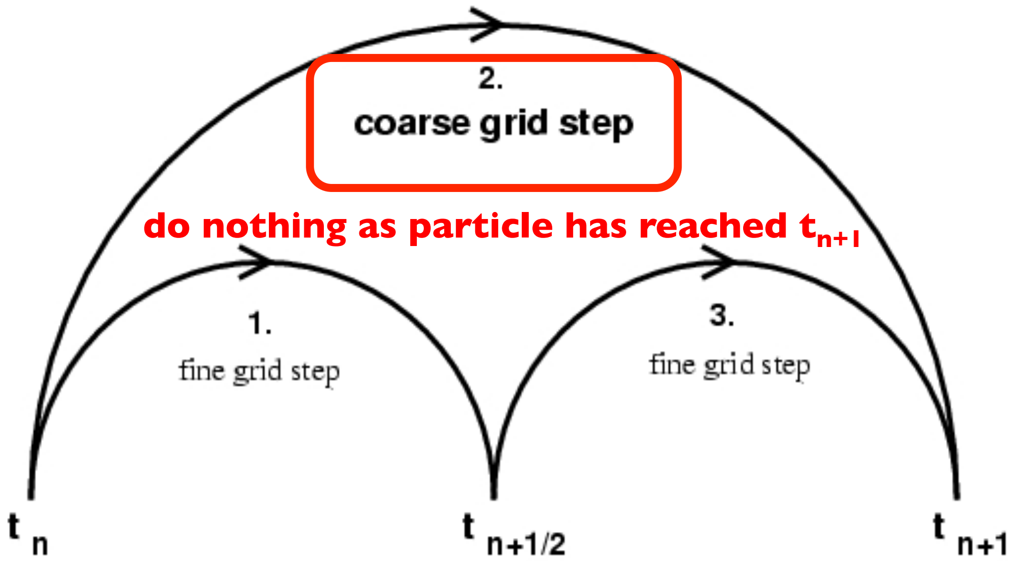
2. Drift: $\vec{x}^{n+1/2} = \vec{x}^{n+1/4} + \vec{p}^{n+1/2} \int_{t_n+\Delta t/4}^{t_n+\Delta t/2} dt$

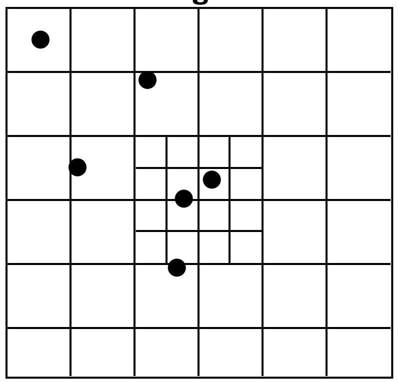
3. Drift: $\vec{x}^{n+3/4} = \vec{x}^{n+1/2} + \vec{p}^{n} \int_{t_n+\Delta t/2}^{t_n+3\Delta t/4} dt$

4. Drift: $\vec{x}^{n+1} = \vec{x}^{n+3/4} + \vec{p}^{n+1} \int_{t_n+3\Delta t/4}^{t_n+\Delta t} dt$

un-drift and move with coarse grid time step to t_{n+1}...

- moving particles on the AMR hierarchy
  - particles crossing grid boundaries





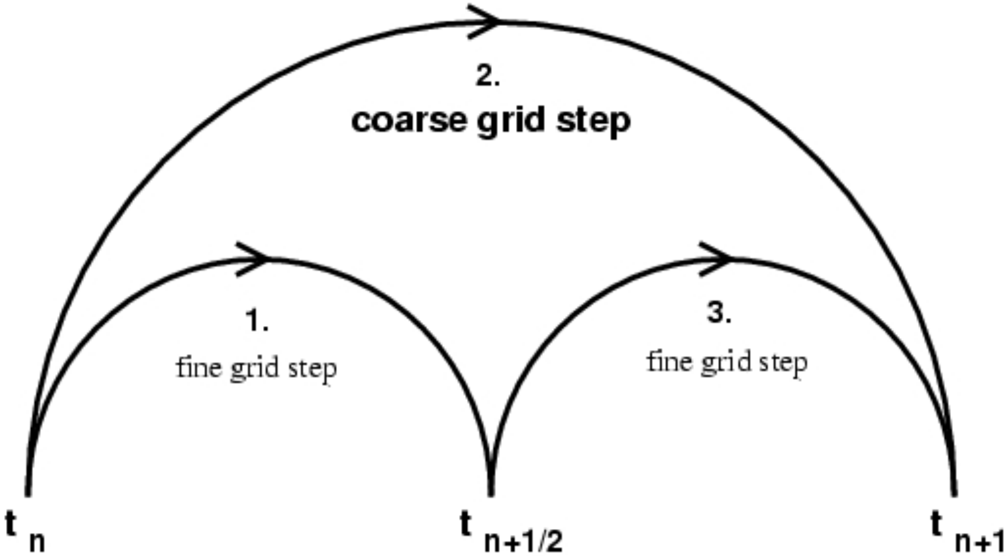
1. Drift: $\vec{x}^{n+1/4} = \vec{x}^n + \vec{p}^n \int_{t_n}^{t_n+\Delta t/4} dt$

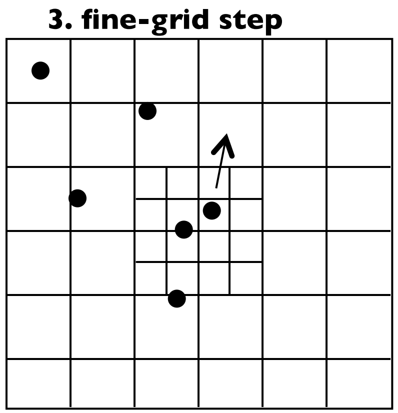
2. Drift: $\vec{x}^{n+1/2} = \vec{x}^{n+1/4} + \vec{p}^{n+1/2} \int_{t_n+\Delta t/4}^{t_n+\Delta t/2} dt$

3. Drift: $\vec{x}^{n+3/4} = \vec{x}^{n+1/2} + \vec{p}^n \int_{t_n+\Delta t/2}^{t_n+3\Delta t/4} dt$

4. Drift: $\vec{x}^{n+1} = \vec{x}^{n+3/4} + \vec{p}^{n+1} \int_{t_n+3\Delta t/4}^{t_n+\Delta t} dt$

- moving particles on the AMR hierarchy
  - particles crossing grid boundaries





1. Drift: $\vec{x}^{n+1/4} = \vec{x}^{n} + \vec{p}^{n} \int_{t_n}^{t_n+\Delta t/4} dt$

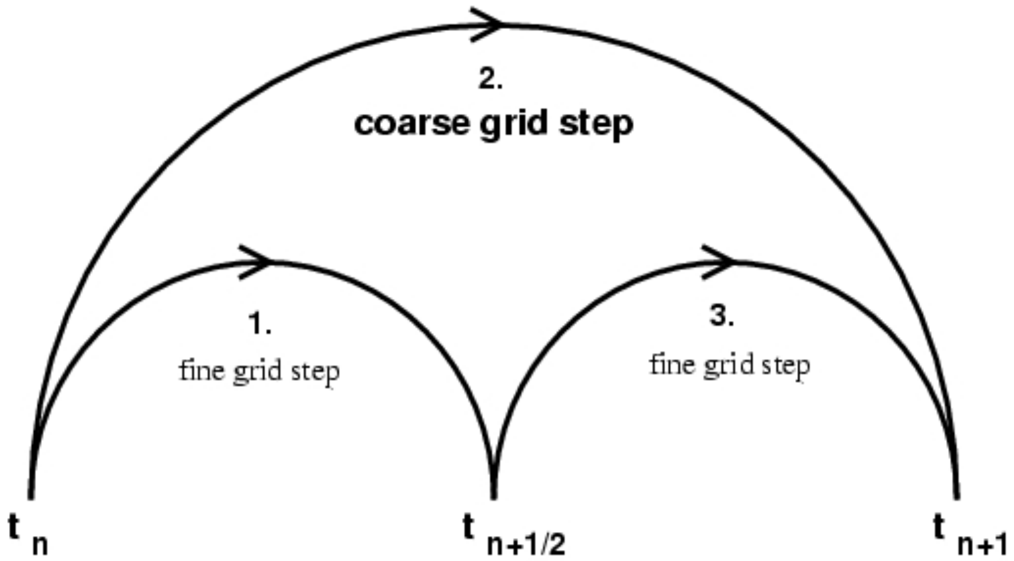
2. Drift: $\vec{x}^{n+1/2} = \vec{x}^{n+1/4} + \vec{p}^{n+1/2} \int_{t_n+\Delta t/4}^{t_n+\Delta t/2} dt$

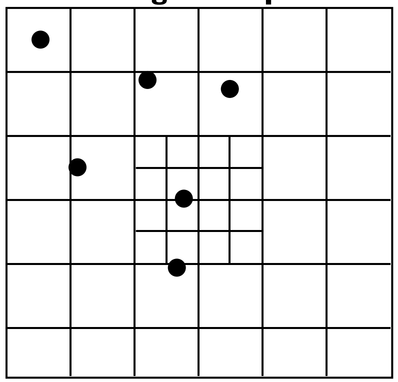
3. Drift: $\vec{x}^{n+3/4} = \vec{x}^{n+1/2} + \vec{p}^{n} \int_{t_n+\Delta t/2}^{t_n+3\Delta t/4} dt$

4. Drift: $\vec{x}^{n+1} = \vec{x}^{n+3/4} + \vec{p}^{n+1} \int_{t_n+3\Delta t/4}^{t_n+\Delta t} dt$

keep on drift'ing as it will bring the particle to t_{n+1}

- moving particles on the AMR hierarchy
  - particles crossing grid boundaries





1. Drift: $\vec{x}^{n+1/4} = \vec{x}^{n} + \vec{p}^{n} \int_{t_n}^{t_n+\Delta t/4} dt$

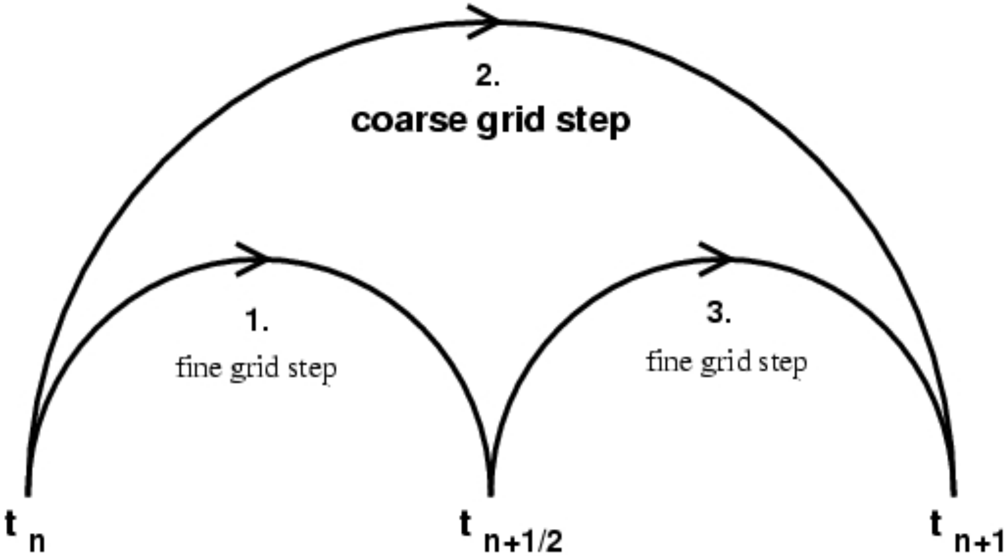
2. Drift: $\vec{x}^{n+1/2} = \vec{x}^{n+1/4} + \vec{p}^{n+1/2} \int_{t_n+\Delta t/4}^{t_n+\Delta t/2} dt$

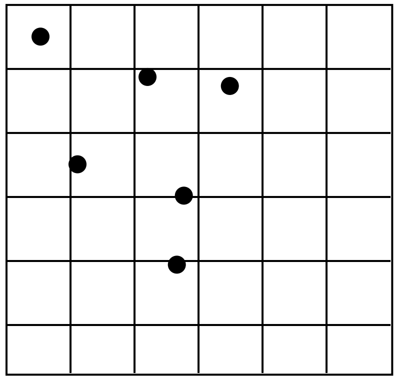
3. Drift: $\vec{x}^{n+3/4} = \vec{x}^{n+1/2} + \vec{p}^{n} \int_{t_n+\Delta t/2}^{t_n+3\Delta t/4} dt$

4. Drift: $\vec{x}^{n+1} = \vec{x}^{n+3/4} + \vec{p}^{n+1} \int_{t_n+3\Delta t/4}^{t_n+\Delta t} dt$

keep on drift'ing as it will bring the particle to t_{n+1}

- moving particles on the AMR hierarchy
  - particles crossing grid boundaries





1. Drift : $\vec{x}^{n+1/4} = \vec{x}$ …

← 2. Drift : $\vec{x}^{n+1/2} = \vec{x}^{n}$ …

all particles have now moved from t_n to t_{n+1} and the refinements will be re-created...

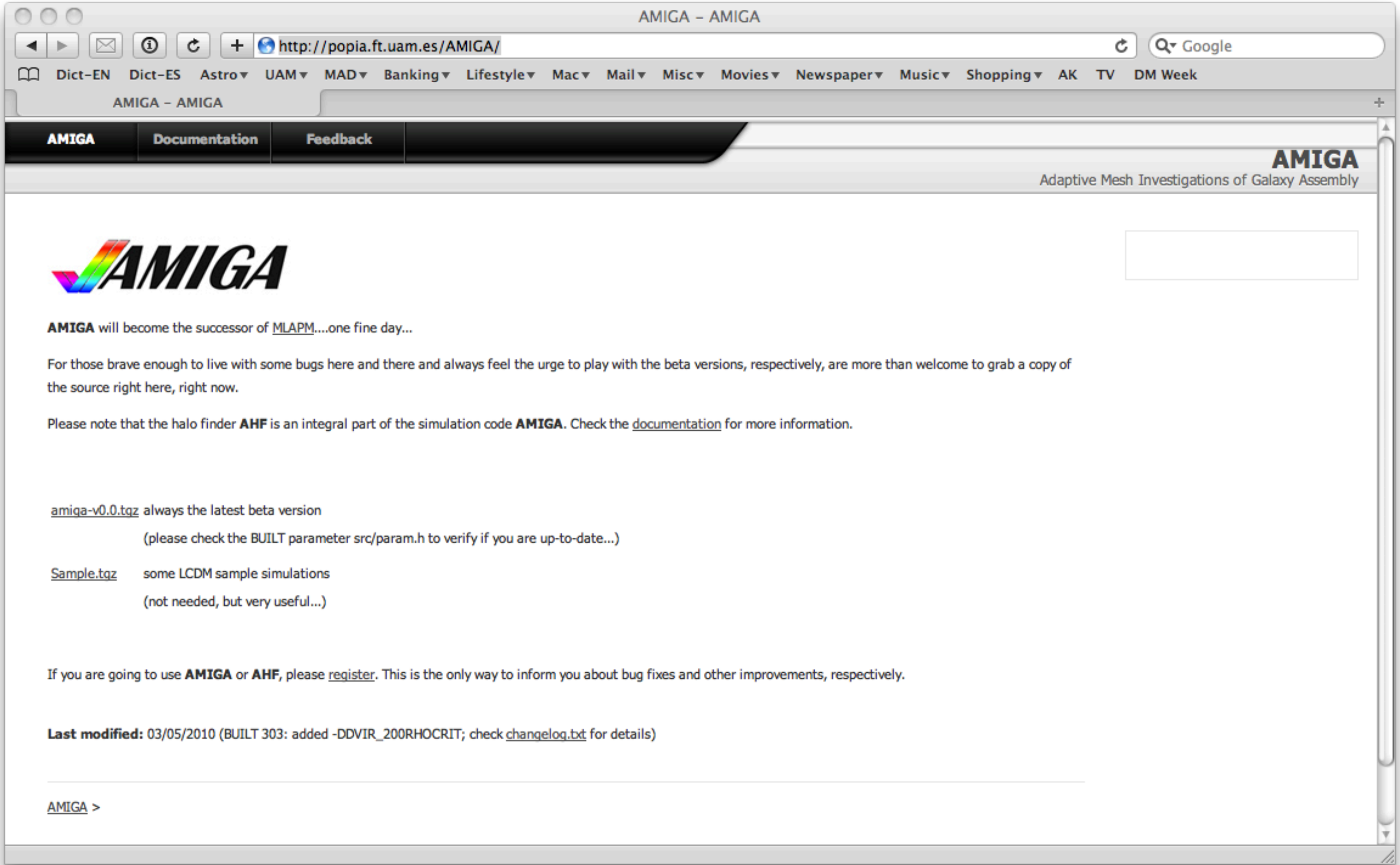
```
Step(dt, CurrentGrid) {

   NewGrid = Refine(CurrentGrid);

   if(NewGrid) {
      Step(dt/2, NewGrid); }

   MoveParticles(dt, CurrentGrid);

   if(NewGrid) {
      Step(dt/2, NewGrid);
      DestroyGrid(NewGrid);}
}
```


AMIGA – AMIGA
http://popia.ft.uam.es/AMIGA/
Google
Dict-EN Dict-ES Astro UAM MAD Banking Lifestyle Mac Mail Misc Movies Newspaper Music Shopping AK TV DM Week
AMIGA – AMIGA
AMIGA Documentation Feedback
AMIGA
Adaptive Mesh Investigations of Galaxy Assembly
AMIGA
AMIGA will become the successor of MLAPM....one fine day...
For those brave enough to live with some bugs here and there and always feel the urge to play with the beta versions, respectively, are more than welcome to grab a copy of the source right here, right now.
Please note that the halo finder AHF is an integral part of the simulation code AMIGA. Check the documentation for more information.
amiga-v0.0.tgz always the latest beta version
(please check the BUILT parameter src/param.h to verify if you are up-to-date...)
Sample.tgz some LCDM sample simulations
(not needed, but very useful...)
If you are going to use AMIGA or AHF, please register. This is the only way to inform you about bug fixes and other improvements, respectively.
Last modified: 03/05/2010 (BUILT 303: added -DDVIR_200RHOCRIT; check changelog.txt for details)
AMIGA >