CS 5220: Locality and parallelism in simulations I

David Bindel

2017-09-12

Parallelism and locality

- Real world exhibits *parallelism* and *locality*
  - Particles, people, etc function independently
  - Nearby objects interact more strongly than distant ones
  - Can often simplify dependence on distant objects
- Can get more parallelism / locality through model
  - Limited range of dependency between adjacent time steps
  - Can neglect or approximate far-field effects
- Often get parallism at multiple levels
  - Hierarchical circuit simulation
  - Interacting models for climate
  - Parallelizing individual experiments in MC or optimization

Basic styles of simulation

- Discrete event systems (continuous or discrete time)
  - Game of life, logic-level circuit simulation
  - Network simulation
- Particle systems
  - Billiards, electrons, galaxies, ...
  - Ants, cars, ...?
- Lumped parameter models (ODEs)
  - Circuits (SPICE), structures, chemical kinetics
- Distributed parameter models (PDEs / integral equations)
  - Heat, elasticity, electrostatics, ...

Often more than one type of simulation appropriate.

Sometimes more than one at a time!

Discrete events

Basic setup:

- Finite set of variables, updated via transition function
- *Synchronous* case: finite state machine
- *Asynchronous* case: event-driven simulation
- Synchronous example: Game of Life

Nice starting point — no discretization concerns!

Game of Life

Lonely (Dead next step)

Crowded

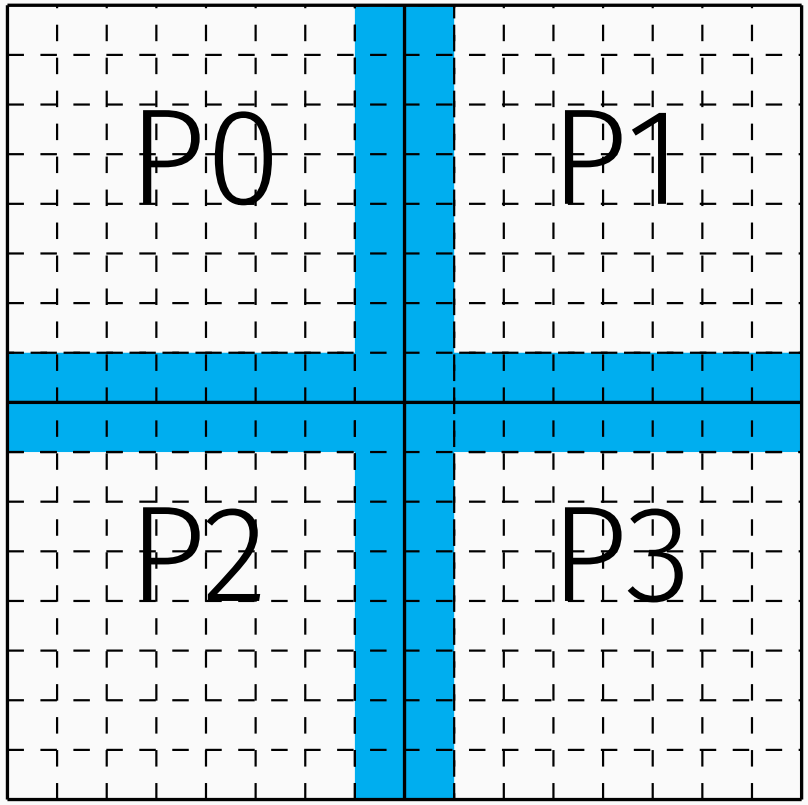
OK (Live next step)

Born

Game of Life (John Conway):

1. Live cell dies with < 2 live neighbors
2. Live cell dies with > 3 live neighbors
3. Live cell lives with 2–3 live neighbors
4. Dead cell becomes live with exactly 3 live neighbors

Game of Life



Easy to parallelize by *domain decomposition*.

- Update work involves *volume* of subdomains
- Communication per step on *surface* (cyan)

Game of Life: Pioneers and Settlers

What if pattern is “dilute”?

- Few or no live cells at surface at each step
- Think of live cell at a surface as an “event”
- Only communicate events!
  - This is *asynchronous*
  - Harder with message passing — when do you receive?

Asynchronous Game of Life

How do we manage events?

- Could be *speculative* — assume no communication across boundary for many steps, back up if needed
- Or *conservative* — wait whenever communication possible
  - possible $\not\equiv$ guaranteed!
  - Deadlock: everyone waits for everyone else to send data
  - Can get around this with NULL messages

How do we manage load balance?

- No need to simulate quiescent parts of the game!
- Maybe dynamically assign smaller blocks to processors?

Particle simulation

Particles move via Newton ($F = ma$), with

- External forces: ambient gravity, currents, etc.
- Local forces: collisions, Van der Waals ($1/r^6$), etc.
- Far-field forces: gravity and electrostatics ($1/r^2$), etc.
    - Simple approximations often apply (Saint-Venant)

A forced example

Example force:

$$f_i = \sum_j G m_i m_j \frac{(x_j - x_i)}{r_{ij}^3} \left(1 - \left(\frac{a}{r_{ij}}\right)^4\right), \qquad r_{ij} = \|x_i - x_j\|$$

- Long-range attractive force (r^{-2})
- Short-range repulsive force (r^{-6})
- Go from attraction to repulsion at radius a

A simple serial simulation

In MATLAB, we can write

```
npts = 100;
t = linspace(0, tfinal, npts);
[tout, xyv] = ode113(@fnbody, ...
                     t, [x; v], [], m, g);
xout = xyv(:,1:length(x))';
```

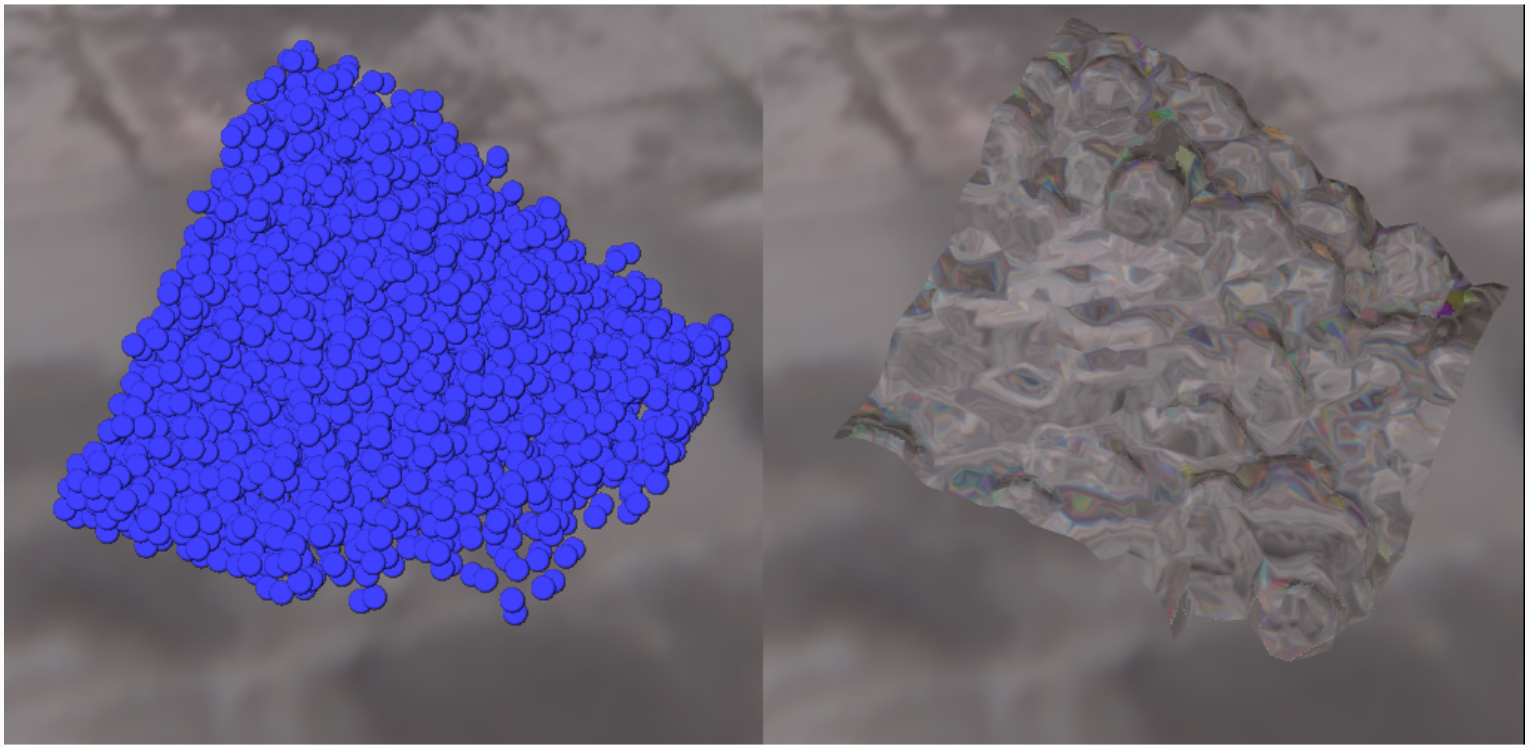
… but I can't call `ode113` in C in parallel (or can I?)

A simple serial simulation

Maybe a fixed step leapfrog will do?

```
npts = 100;
steps_per_pt = 10;
dt = tfinal/(steps_per_pt*(npts-1));
xout = zeros(2*n, npts);
xout(:,1) = x;
for i = 1:npts-1
  for ii = 1:steps_per_pt
    x = x + v*dt;
    a = fnbody(x, m, g);
    v = v + a*dt;
  end
  xout(:,i+1) = x;
end
```

Plotting particles



Pondering particles

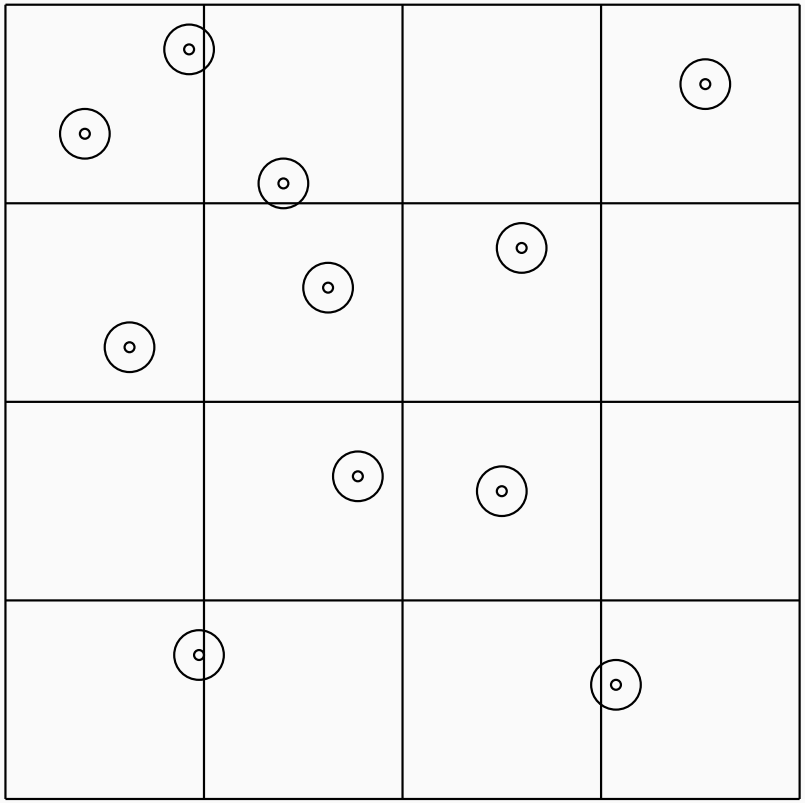
- Where do particles “live” (esp. in distributed memory)?
  - Decompose in space? By particle number?
  - What about clumping?
- How are long-range force computations organized?
- How are short-range force computations organized?
- How is force computation load balanced?
- What are the boundary conditions?
- How are potential singularities handled?
- What integrator is used? What step control?

External forces

Simplest case: no particle interactions.

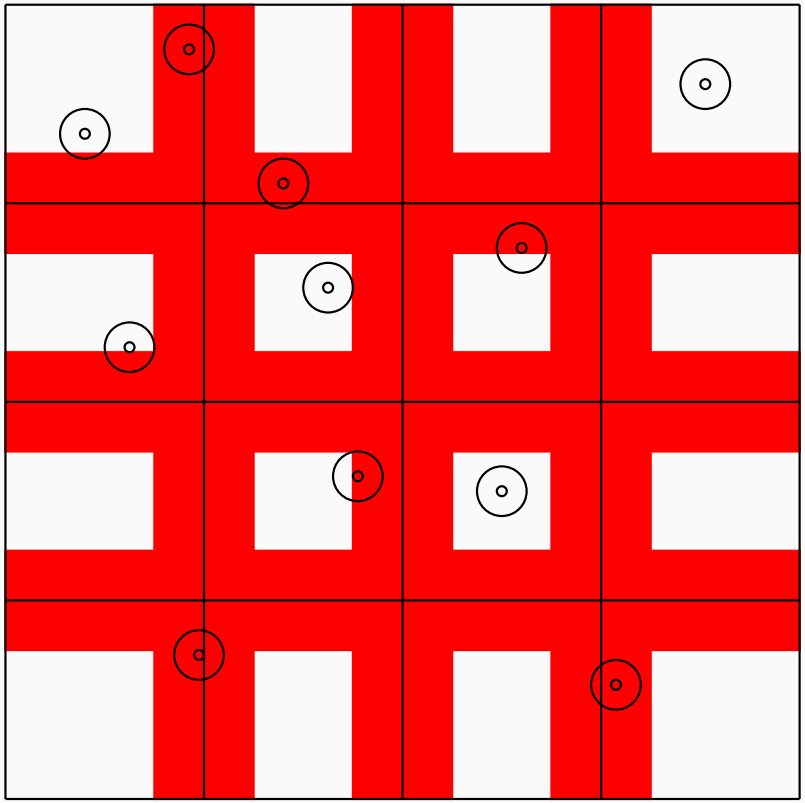
- Embarrassingly parallel (like Monte Carlo)!
- Could just split particles evenly across processors
- Is it that easy?
  - Maybe some trajectories need short time steps?
  - Even with MC, load balance may not be entirely trivial.

Local forces



- Simplest all-pairs check is $O(n^2)$ (expensive)
- Or only check close pairs (via binning, quadtrees?)
- Communication required for pairs checked
- Usual model: domain decomposition

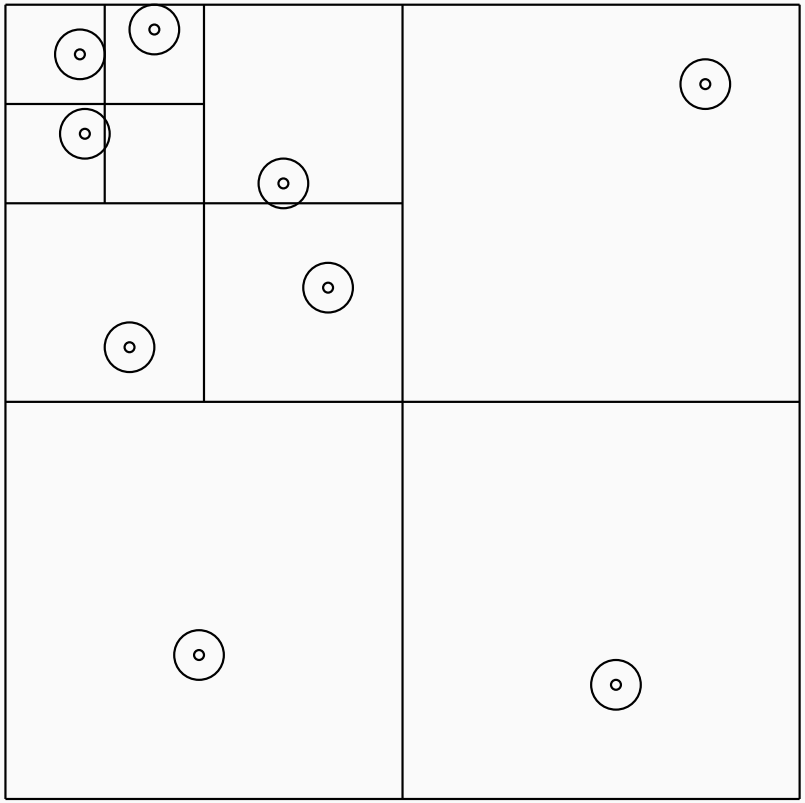
Local forces: Communication



Minimize communication:

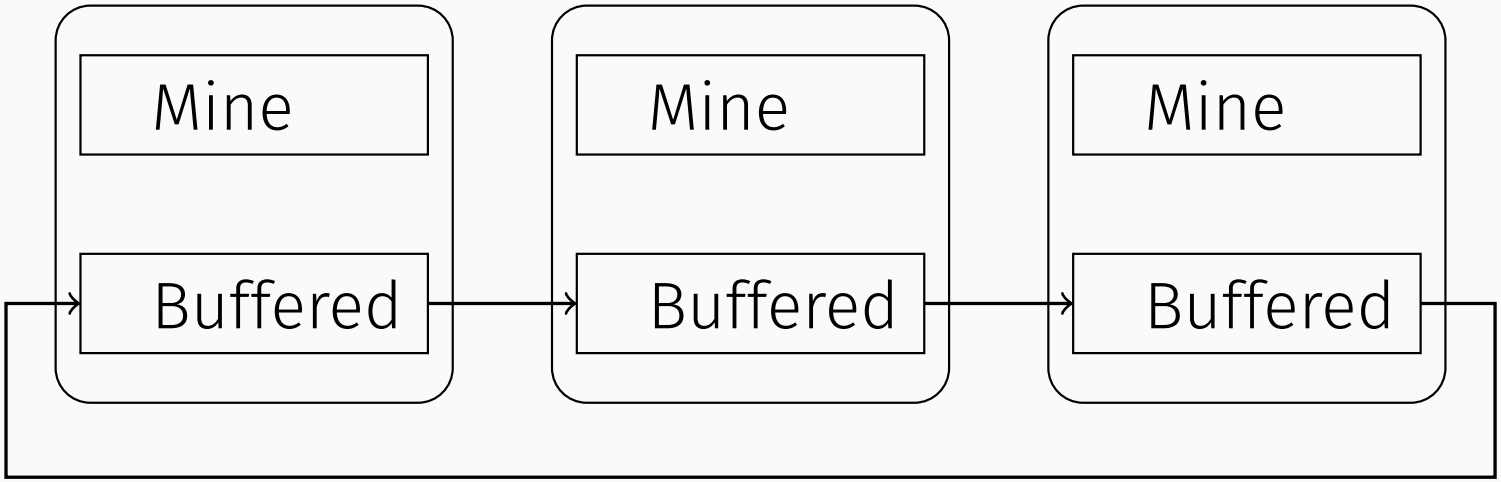
- Send particles that might affect a neighbor “soon”
- Trade extra computation against communication
- Want low surface area-to-volume ratios on domains

Local forces: Load balance



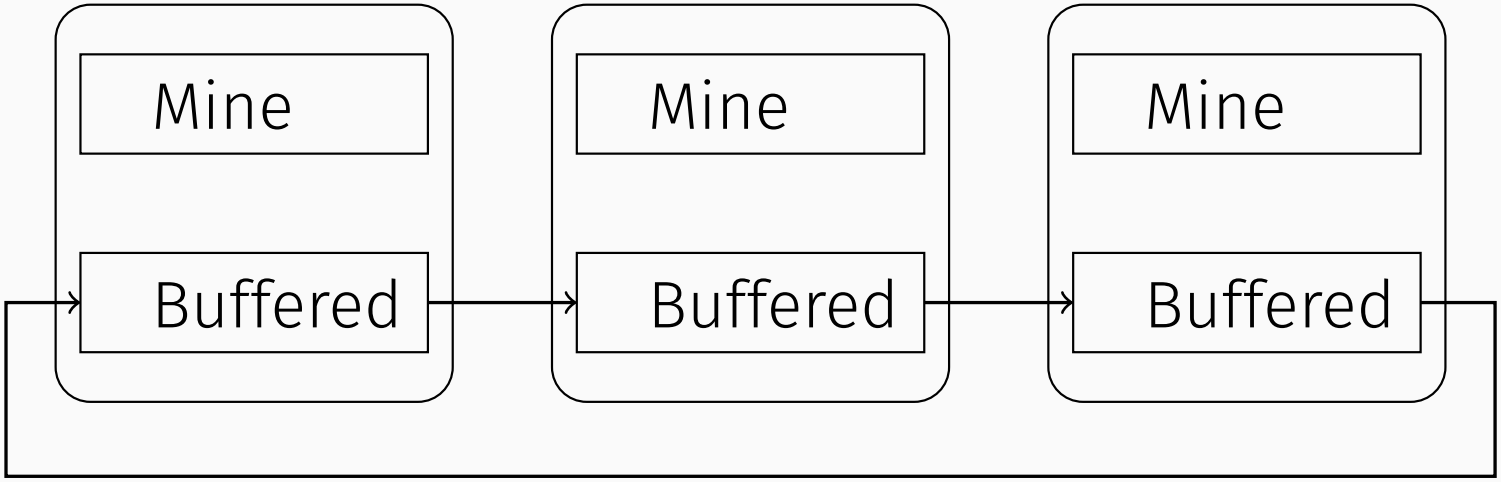
- Are particles evenly distributed?
- Do particles remain evenly distributed?
- Can divide space unevenly (e.g. quadtree/octtree)

Far-field forces



- Every particle affects every other particle
- All-to-all communication required
  - Overlap communication with computation
  - Poor memory scaling if everyone keeps everything!
- Idea: pass particles in a round-robin manner

Passing particles for far-field forces




```
copy local particles to current buf
for phase = 1:p
  send current buf to rank+1 (mod p)
  recv next buf from rank-1 (mod p)
  interact local particles with current buf
  swap current buf with next buf
end
```

Passing particles for far-field forces

Suppose $n = N/p$ particles in buffer. At each phase

$$t_{\text{comm}} \approx \alpha + \beta n$$
$$t_{\text{comp}} \approx \gamma n^2$$

So we can mask communication with computation if

$$n \geq \frac{1}{2\gamma}\left(\beta + \sqrt{\beta^2 + 4\alpha\gamma}\right) > \frac{\beta}{\gamma}$$

More efficient serial code
$\implies$ larger n needed to mask communication!
$\implies$ worse speed-up as p gets larger (fixed N)
but scaled speed-up (n fixed) remains unchanged.

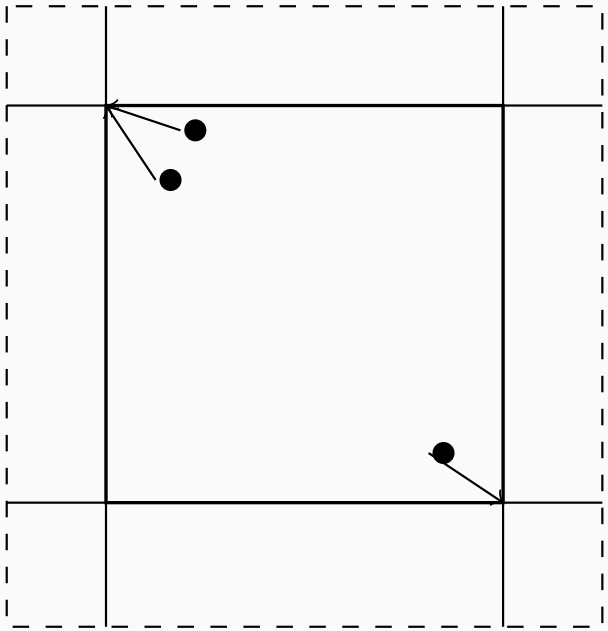
This analysis neglects overhead term in LogP.

Far-field forces: particle-mesh methods

Consider r^{-2} electrostatic potential interaction

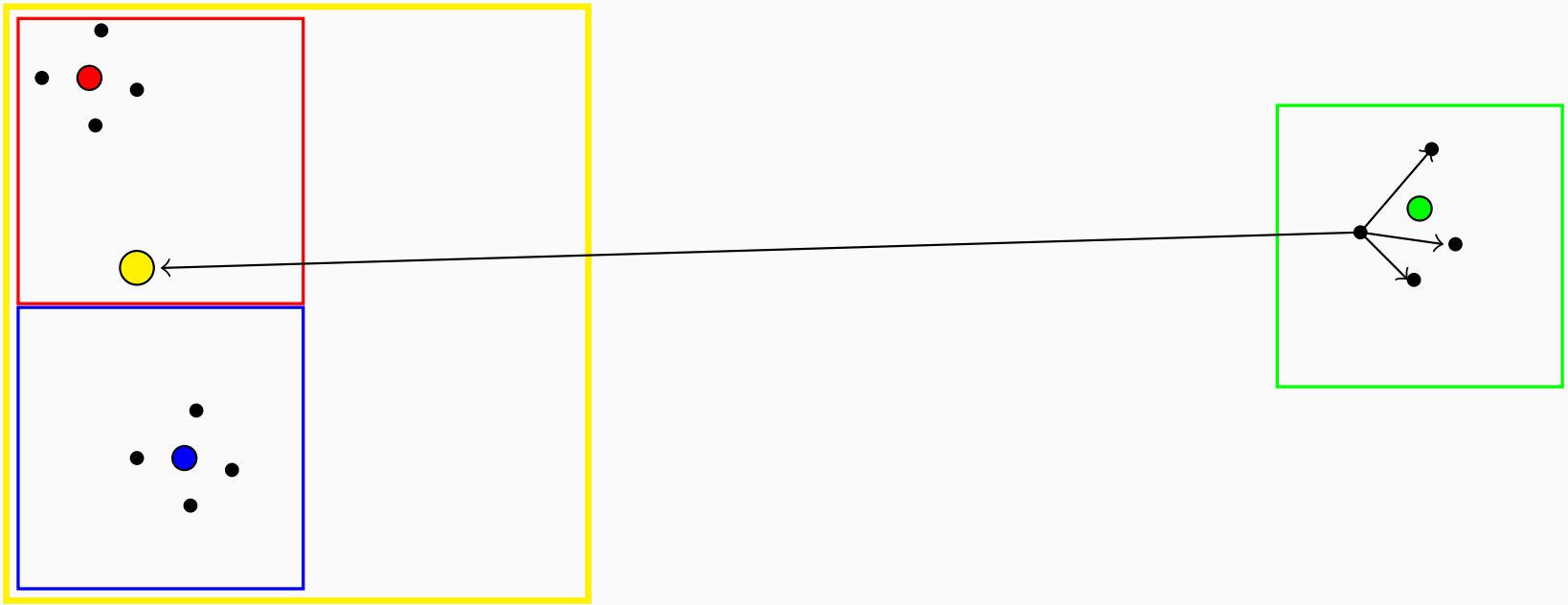
- Enough charges looks like a continuum!
- Poisson equation maps charge distribution to potential
- Use fast Poisson solvers for regular grids (FFT, multigrid)
- Approximation depends on mesh and particle density
- Can clean up leading part of approximation error

Far-field forces: particle-mesh methods



- Map particles to mesh points (multiple strategies)
- Solve potential PDE on mesh
- Interpolate potential to particles
- Add correction term – acts like local force

Far-field forces: tree methods



- Distance simplifies things
  - Andromeda looks like a point mass from here?
- Build a tree, approximating descendants at each node
- Several variants: Barnes-Hut, FMM, Anderson's method
- More on this later in the semester

Summary of particle example

- Model: Continuous motion of particles
  - Could be electrons, cars, whatever...
- Step through discretized time
- Local interactions
  - Relatively cheap
  - Load balance a pain
- All-pairs interactions
  - Obvious algorithm is expensive ($O(n^2)$)
  - Particle-mesh and tree-based algorithms help

An important special case of lumped/ODE models.