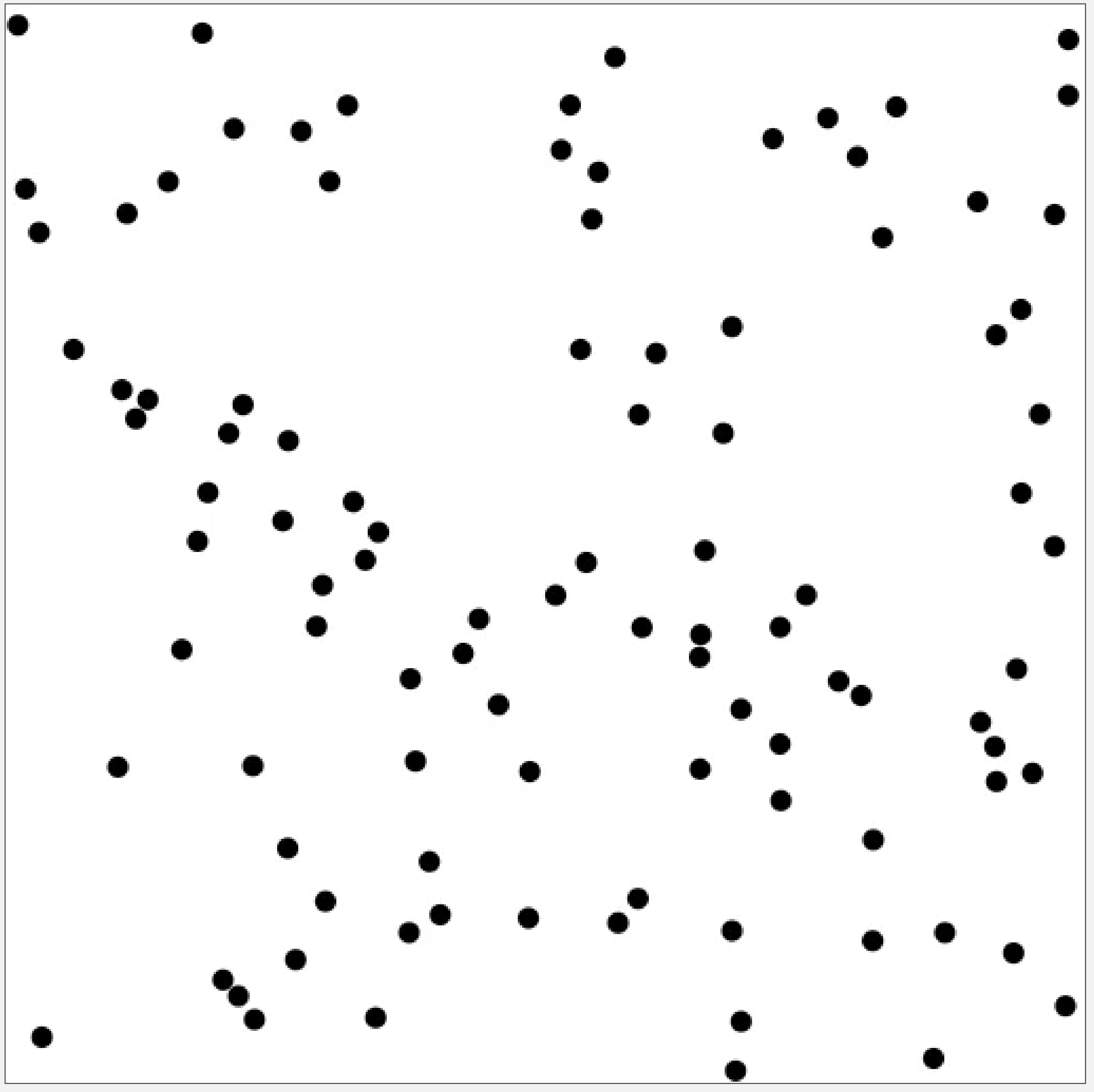
http://algs4.cs.princeton.edu

2.4 PRIORITY QUEUES

- *API and elementary implementations*
- *binary heaps*
- *heapsort*
- *event-driven simulation*

Molecular dynamics simulation of hard discs

Goal. Simulate the motion of N moving particles that behave according to the laws of elastic collision.



Molecular dynamics simulation of hard discs

Goal. Simulate the motion of N moving particles that behave according to the laws of elastic collision.

Hard disc model.

- Moving particles interact via elastic collisions with each other and walls.
- Each particle is a disc with known position, velocity, mass, and radius.
- No other forces.

temperature, pressure, diffusion constant

motion of individual atoms and molecules

Significance. Relates macroscopic observables to microscopic dynamics.

- Maxwell-Boltzmann: distribution of speeds as a function of temperature.
- Einstein: explain Brownian motion of pollen grains.

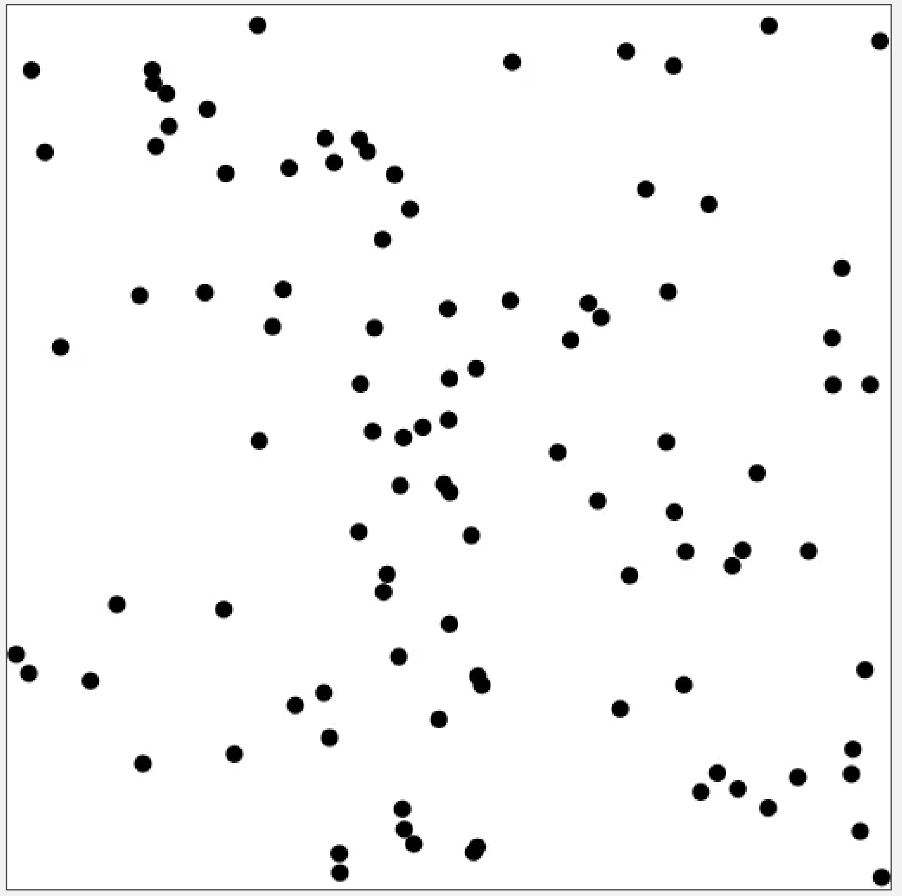
Warmup: bouncing balls

Time-driven simulation. N bouncing balls in the unit square.

```
public class BouncingBalls
{
   public static void main(String[] args)
   {
      int N = Integer.parseInt(args[0]);
      Ball[] balls = new Ball[N];
      for (int i = 0; i < N; i++)
         balls[i] = new Ball();
      while(true)
      {
         StdDraw.clear();
         for (int i = 0; i < N; i++)
         {
            balls[i].move(0.5);
            balls[i].draw();
         }
         StdDraw.show(50);
      }
   }
}
```

main simulation loop

```
% java BouncingBalls 100
```



Warmup: bouncing balls

```
public class Ball
{
    private double rx, ry;        // position
    private double vx, vy;        // velocity
    private final double radius;  // radius
    public Ball(...)
    {  /* initialize position and velocity */  }

    public void move(double dt)
    {
        if ((rx + vx*dt < radius) || (rx + vx*dt > 1.0 - radius)) { vx = -vx; }
        if ((ry + vy*dt < radius) || (ry + vy*dt > 1.0 - radius)) { vy = -vy; }
        rx = rx + vx*dt;
        ry = ry + vy*dt;
    }
    public void draw()
    {  StdDraw.filledCircle(rx, ry, radius);  }
}
```

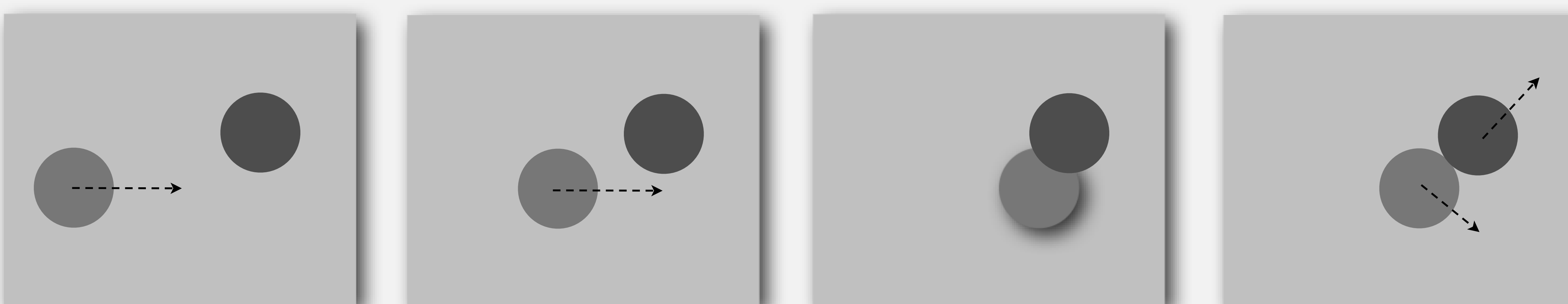
check for collision with walls

Missing. Check for balls colliding with each other.

- Physics problems: when? what effect?
- CS problems: which object does the check? too many checks?

Time-driven simulation

- Discretize time in quanta of size dt.
- Update the position of each particle after every dt units of time, and check for overlaps.
- If overlap, roll back the clock to the time of the collision, update the velocities of the colliding particles, and continue the simulation.



t

t + dt

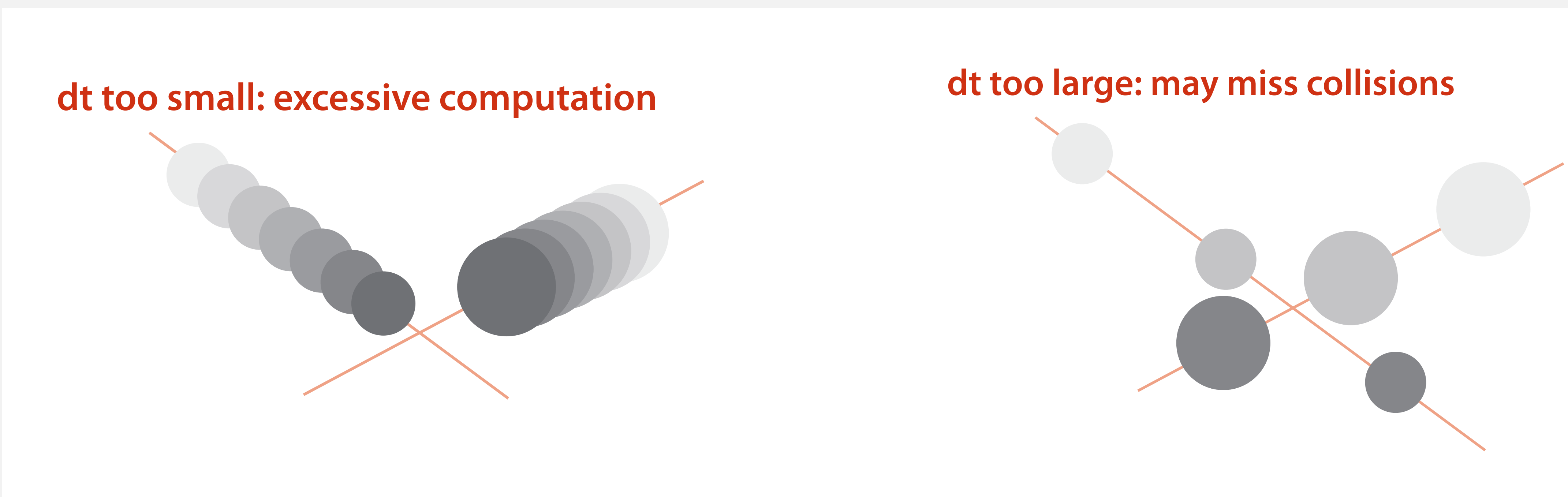
t + 2 dt
(collision detected)

t + Δt
(roll back clock)

Time-driven simulation

Main drawbacks.

- $\sim N^2 / 2$ overlap checks per time quantum.
- Simulation is too slow if dt is very small.
- May miss collisions if dt is too large.
  (if colliding particles fail to overlap when we are looking)



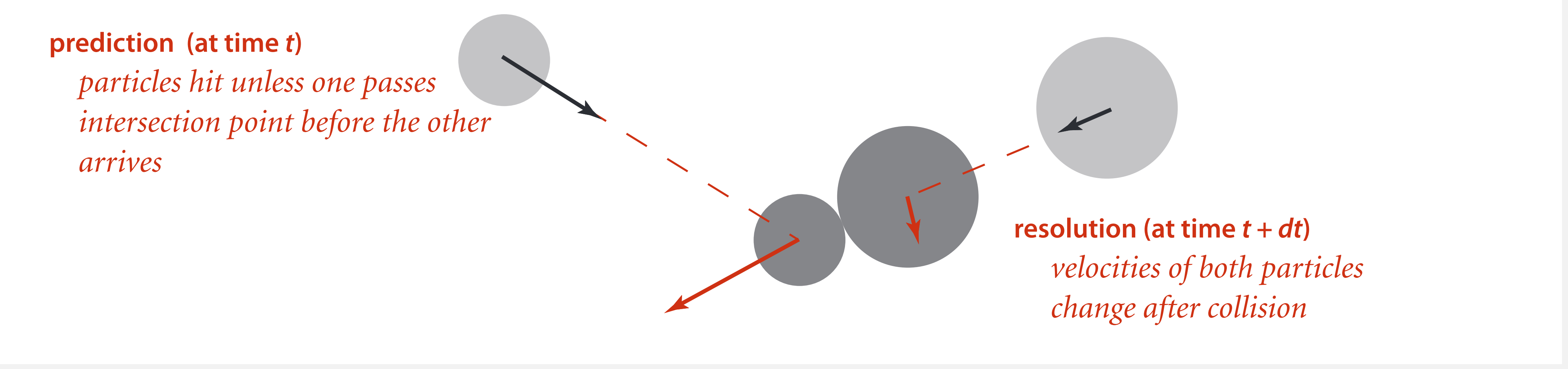
Event-driven simulation

Change state only when something happens.

- Between collisions, particles move in straight-line trajectories.
- Focus only on times when collisions occur.
- Maintain PQ of collision events, prioritized by time.
- Remove the min = get next collision.

Collision prediction. Given position, velocity, and radius of a particle, when will it collide next with a wall or another particle?

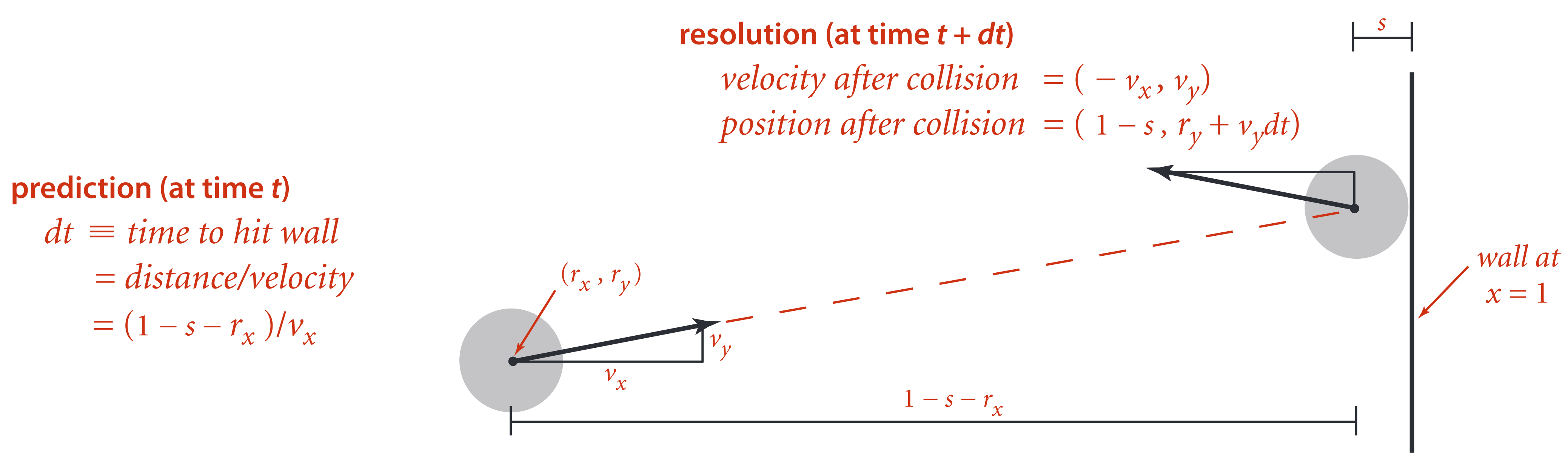
Collision resolution. If collision occurs, update colliding particle(s) according to laws of elastic collisions.



Particle-wall collision

Collision prediction and resolution.

- Particle of radius s at position (rx, ry).
- Particle moving in unit box with velocity (vx, vy).
- Will it collide with a vertical wall? If so, when?

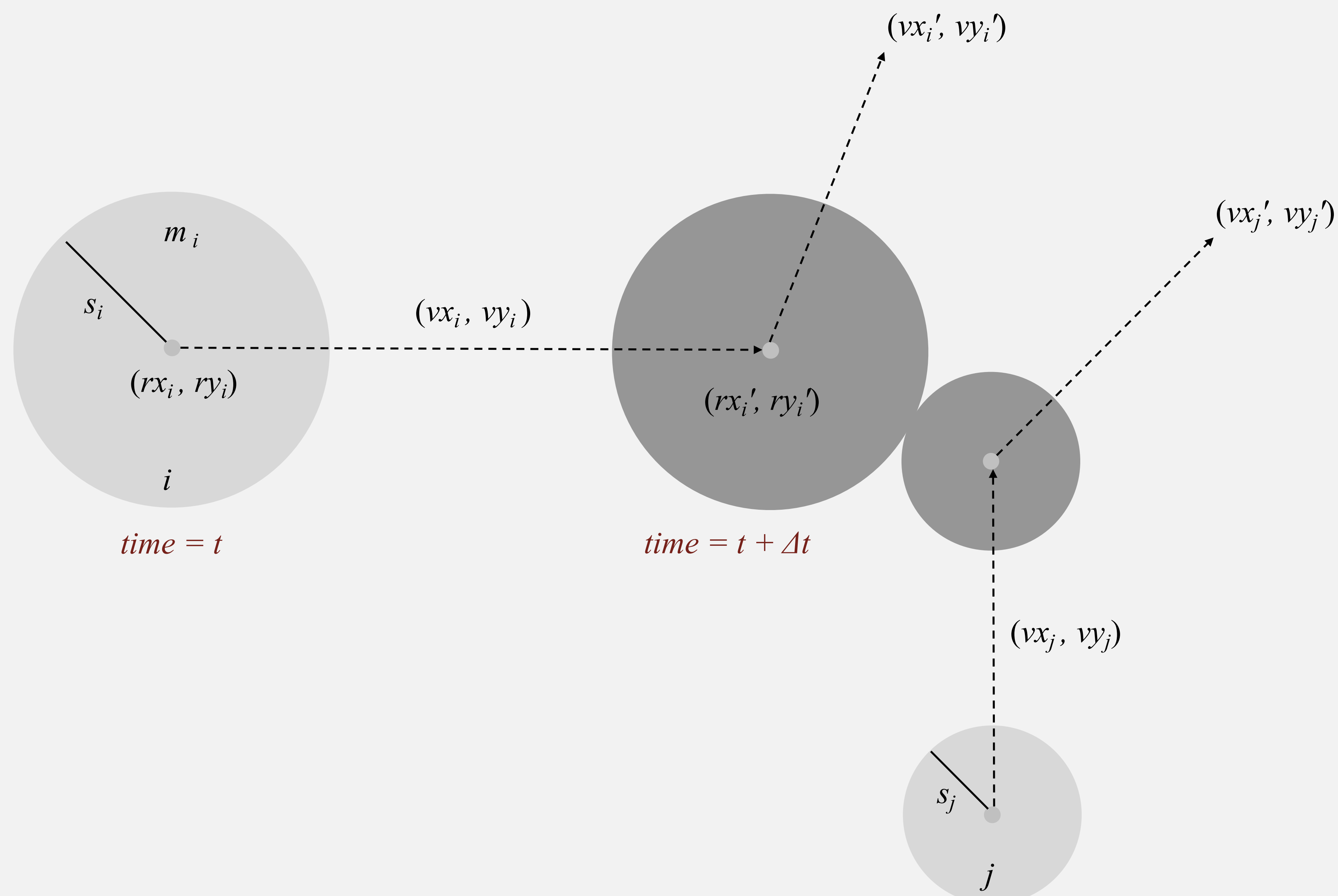


Predicting and resolving a particle-wall collision

Particle-particle collision prediction

Collision prediction.

- Particle i: radius s_i, position (rx_i, ry_i), velocity (vx_i, vy_i).
- Particle j: radius s_j, position (rx_j, ry_j), velocity (vx_j, vy_j).
- Will particles i and j collide? If so, when?



Particle-particle collision prediction

Collision prediction.

- Particle i: radius s_i, position (rx_i, ry_i), velocity (vx_i, vy_i).
- Particle j: radius s_j, position (rx_j, ry_j), velocity (vx_j, vy_j).
- Will particles i and j collide? If so, when?

$$\Delta t = \begin{cases} \infty & \text{if } \Delta v \cdot \Delta r \ge 0 \\ \infty & \text{if } d < 0 \\ -\dfrac{\Delta v \cdot \Delta r + \sqrt{d}}{\Delta v \cdot \Delta v} & \text{otherwise} \end{cases}$$

$$d = (\Delta v \cdot \Delta r)^2 - (\Delta v \cdot \Delta v)\ (\Delta r \cdot \Delta r - \sigma^2) \qquad \sigma = \sigma_i + \sigma_j$$

$$\Delta v = (\Delta vx,\ \Delta vy) = (vx_i - vx_j,\ vy_i - vy_j)$$
$$\Delta r = (\Delta rx,\ \Delta ry) = (rx_i - rx_j,\ ry_i - ry_j)$$

$$\Delta v \cdot \Delta v = (\Delta vx)^2 + (\Delta vy)^2$$
$$\Delta r \cdot \Delta r = (\Delta rx)^2 + (\Delta ry)^2$$
$$\Delta v \cdot \Delta r = (\Delta vx)(\Delta rx) + (\Delta vy)(\Delta ry)$$

Important note: This is physics, so we won't be testing you on it!

Particle-particle collision resolution

Collision resolution. When two particles collide, how does velocity change?

$$
\begin{aligned}
vx_i' &= vx_i + Jx / m_i \\
vy_i' &= vy_i + Jy / m_i \\
vx_j' &= vx_j - Jx / m_j \\
vy_j' &= vy_j - Jy / m_j
\end{aligned}
$$

Newton's second law
(momentum form)

$$
Jx = \frac{J\,\Delta rx}{\sigma}, \quad Jy = \frac{J\,\Delta ry}{\sigma}, \quad J = \frac{2\,m_i\,m_j\,(\Delta v \cdot \Delta r)}{\sigma(m_i + m_j)}
$$

impulse due to normal force
(conservation of energy, conservation of momentum)

Important note: This is physics, so we won't be testing you on it!

Particle data type skeleton

```
public class Particle
{
    private double rx, ry;        // position
    private double vx, vy;        // velocity
    private final double radius;  // radius
    private final double mass;    // mass
    private int count;            // number of collisions

    public Particle(...) { }

    public void move(double dt) { }
    public void draw()          { }

    public double timeToHit(Particle that)  { }
    public double timeToHitVerticalWall()   { }
    public double timeToHitHorizontalWall() { }

    public void bounceOff(Particle that)    { }
    public void bounceOffVerticalWall()     { }
    public void bounceOffHorizontalWall()   { }

}
```

predict collision with particle or wall

resolve collision with particle or wall

Particle-particle collision and resolution implementation

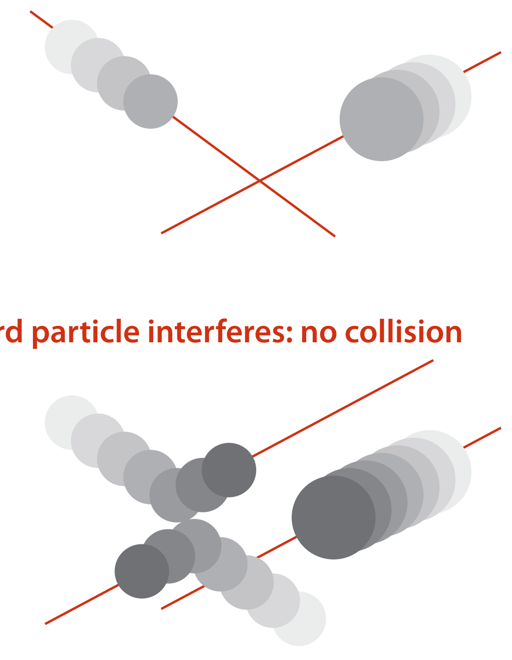
```
public double timeToHit(Particle that)
{
   if (this == that) return INFINITY;
   double dx  = that.rx - this.rx, dy  = that.ry - this.ry;
   double dvx = that.vx - this.vx; dvy = that.vy - this.vy;
   double dvdr = dx*dvx + dy*dvy;
   if( dvdr > 0) return INFINITY;
   double dvdv = dvx*dvx + dvy*dvy;
   double drdr = dx*dx + dy*dy;
   double sigma = this.radius + that.radius;
   double d = (dvdr*dvdr) - dvdv * (drdr - sigma*sigma);
   if (d < 0) return INFINITY;
   return -(dvdr + Math.sqrt(d)) / dvdv;
}
```

no collision

```
public void bounceOff(Particle that)
{
   double dx  = that.rx - this.rx, dy  = that.ry - this.ry;
   double dvx = that.vx - this.vx, dvy = that.vy - this.vy;
   double dvdr = dx*dvx + dy*dvy;
   double dist = this.radius + that.radius;
   double J = 2 * this.mass * that.mass * dvdr / ((this.mass + that.mass) * dist);
   double Jx = J * dx / dist;
   double Jy = J * dy / dist;
   this.vx += Jx / this.mass;
   this.vy += Jy / this.mass;
   that.vx -= Jx / that.mass;
   that.vy -= Jy / that.mass;
   this.count++;
   that.count++;
}
```

Important note: This is physics, so we won't be testing you on it!

Collision system: event-driven simulation main loop



An invalidated event

Initialization.

- Fill PQ with all potential particle-wall collisions.
- Fill PQ with all potential particle-particle collisions.

"potential" since collision may not happen if some other collision intervenes

Main loop.

- Delete the impending event from PQ (min priority = t).
- If the event has been invalidated, ignore it.
- Advance all particles to time t, on a straight-line trajectory.
- Update the velocities of the colliding particle(s).
- Predict future particle-wall and particle-particle collisions involving the colliding particle(s) and insert events onto PQ.

Event data type

Conventions.

- Neither particle `null` ⇒ particle-particle collision.
- One particle `null` ⇒ particle-wall collision.
- Both particles `null` ⇒ redraw event.

```
private class Event implements Comparable<Event>
{
    private double time;          // time of event
    private Particle a, b;        // particles involved in event
    private int countA, countB;   // collision counts for a and b

    public Event(double t, Particle a, Particle b) { }

    public int compareTo(Event that)
    {   return this.time - that.time;   }

    public boolean isValid()
    {   }
}
```

create event

ordered by time

invalid if intervening collision

Collision system implementation: skeleton

```
public class CollisionSystem
{
    private MinPQ<Event> pq;          // the priority queue
    private double t  = 0.0;          // simulation clock time
    private Particle[] particles;     // the array of particles

    public CollisionSystem(Particle[] particles) { }

    private void predict(Particle a)
    {
       if (a == null) return;
       for (int i = 0; i < N; i++)
       {
          double dt = a.timeToHit(particles[i]);
          pq.insert(new Event(t + dt, a, particles[i]));
       }
       pq.insert(new Event(t + a.timeToHitVerticalWall()  , a, null));
       pq.insert(new Event(t + a.timeToHitHorizontalWall(), null, a));
 }

    private void redraw()  { }

    public void simulate() {  /* see next slide */  }
}
```

add to PQ all particle-wall and particle-particle collisions involving this particle

Collision system implementation: main event-driven simulation loop

```
public void simulate()
{
   pq = new MinPQ<Event>();
   for(int i = 0; i < N; i++) predict(particles[i]);
   pq.insert(new Event(0, null, null));

   while(!pq.isEmpty())
   {
      Event event = pq.delMin();
      if(!event.isValid()) continue;
      Particle a = event.a;
      Particle b = event.b;

      for(int i = 0; i < N; i++)
         particles[i].move(event.time - t);
      t = event.time;

      if      (a != null && b != null) a.bounceOff(b);
      else if (a != null && b == null) a.bounceOffVerticalWall()
      else if (a == null && b != null) b.bounceOffHorizontalWall();
      else if (a == null && b == null) redraw();

      predict(a);
      predict(b);
   }
}
```

initialize PQ with collision events and redraw event

get next event

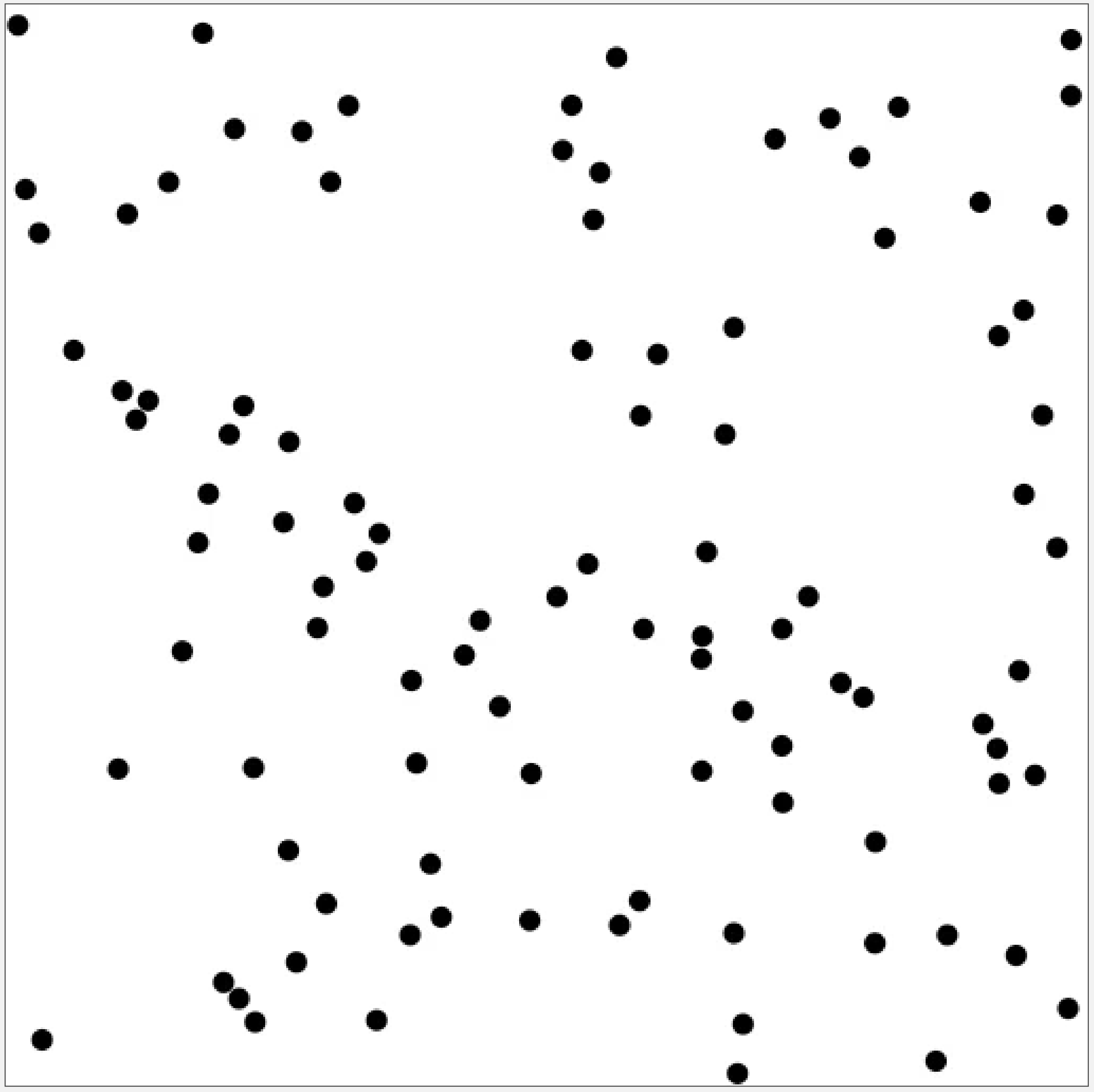
update positions and time

process event

predict new events based on changes

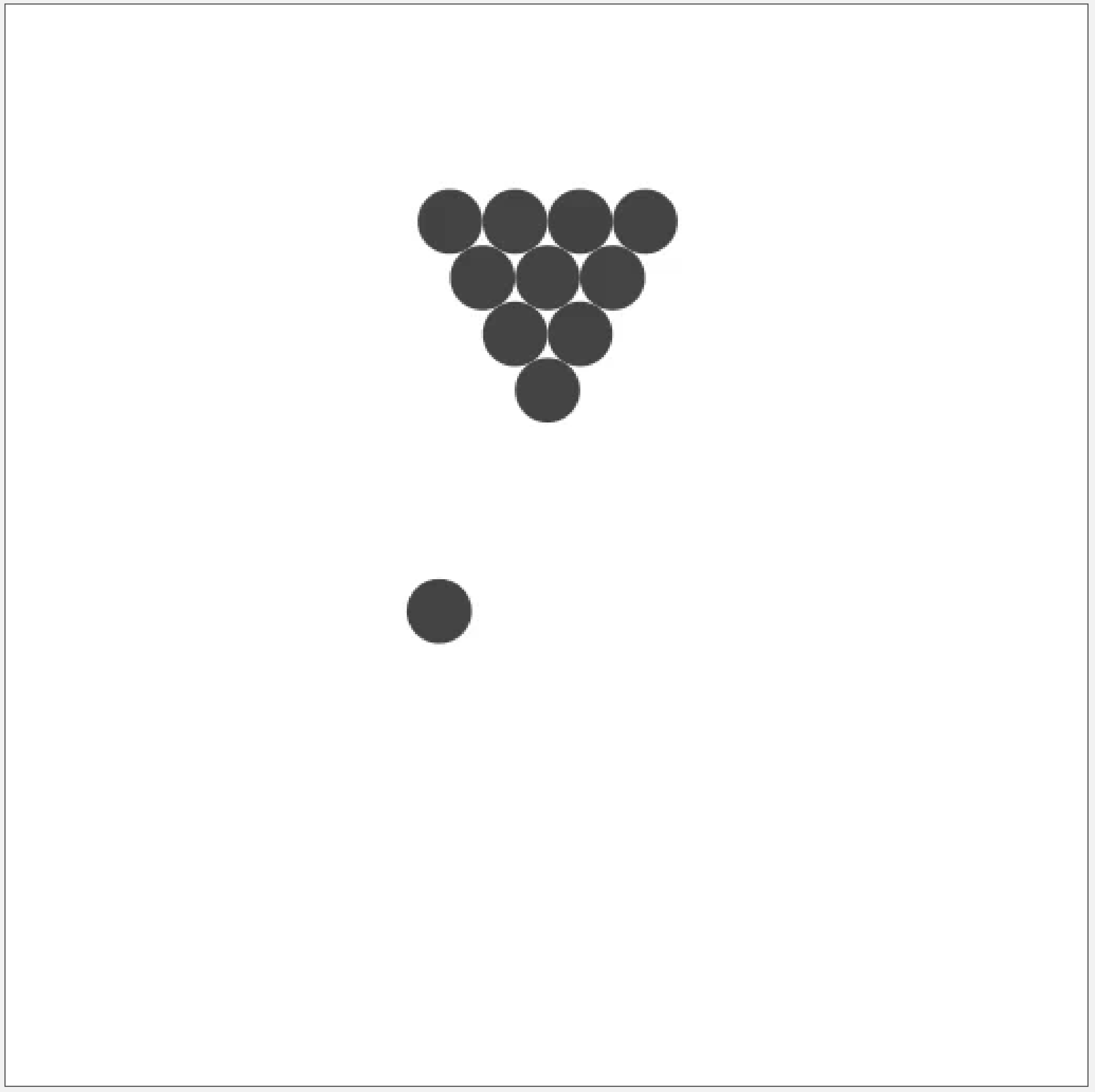
Particle collision simulation example 1

```
% java CollisionSystem 100
```



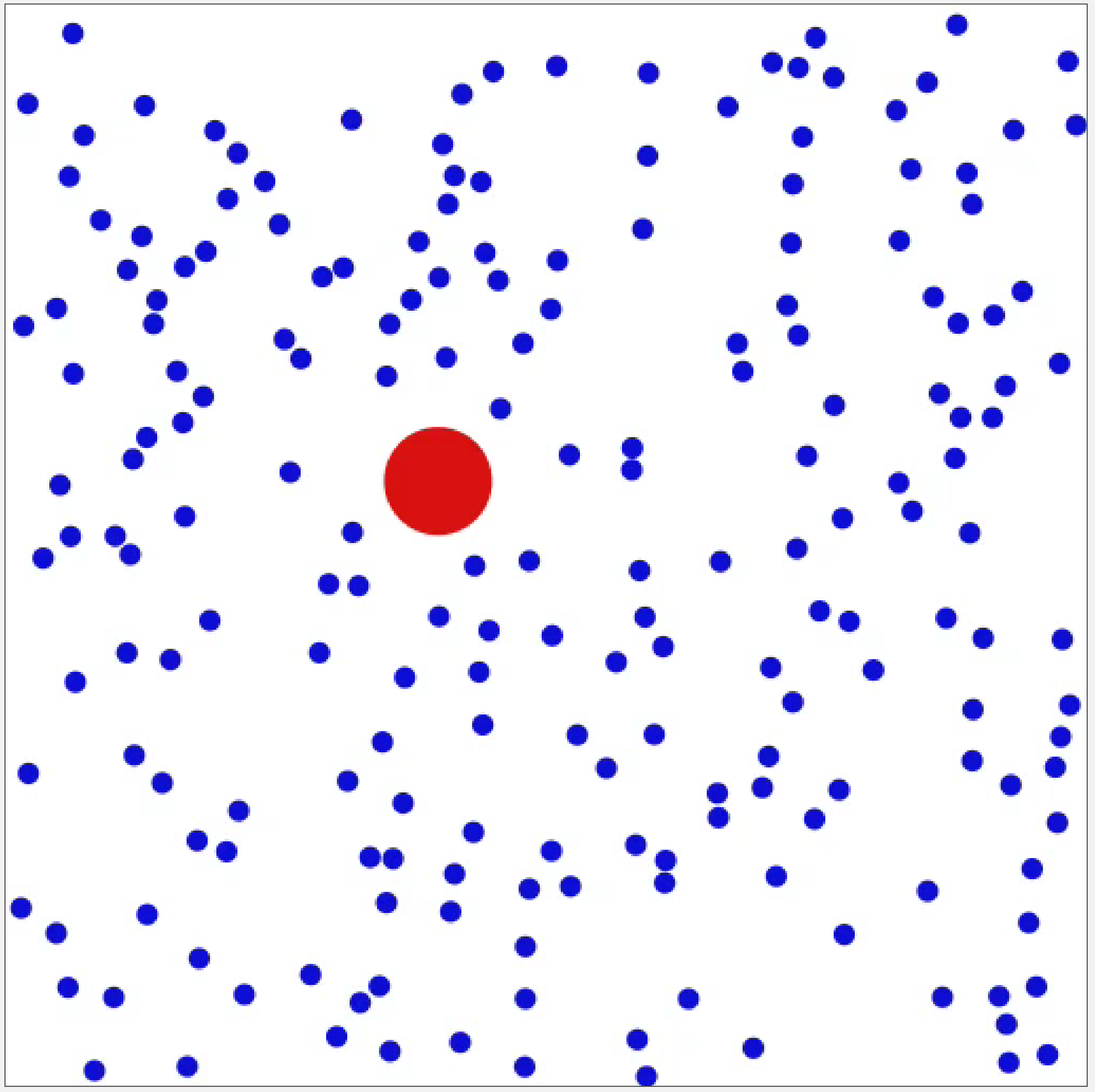
Particle collision simulation example 2

```
% java CollisionSystem < billiards.txt
```



Particle collision simulation example 3

```
% java CollisionSystem < brownian.txt
```



Particle collision simulation example 4

```
% java CollisionSystem < diffusion.txt
```