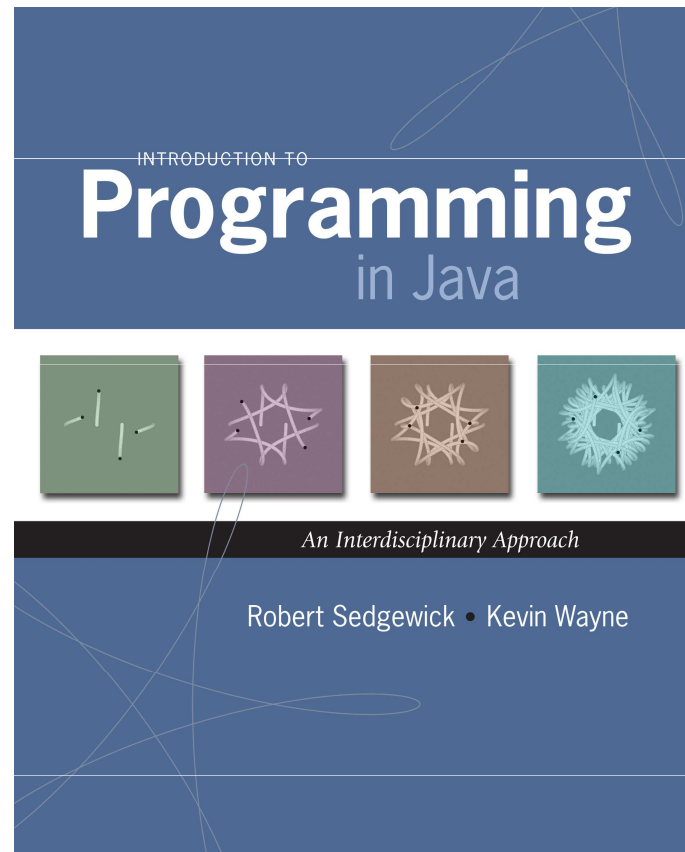


## 3.4 N-body Simulation

---



# N-Body Problem

**Goal.** Determine the motion of  $N$  particles, moving under their mutual Newtonian gravitational forces.

**Ex.** Planets orbit the sun.

QuickTime™ and a  
H.264 decompressor  
are needed to see this picture.

# N-Body: Applications

## Applications to astrophysics.

- Orbits of solar system bodies.
- Stellar dynamics at the galactic center.
- Stellar dynamics in a globular cluster.
- Stellar dynamics during the collision of two galaxies.
- Formation of structure in the universe.
- Dynamics of galaxies during cluster formation.



# N-Body Problem

**Goal.** Determine the motion of  $N$  particles, moving under their mutual Newtonian gravitational forces.

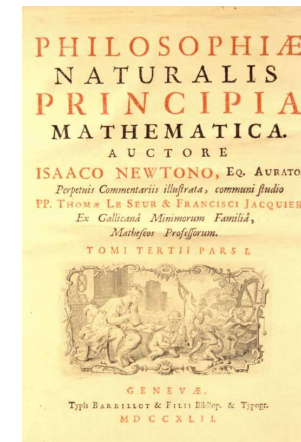
**Context.** Newton formulated the physical principles in Principia.

$$F = m a$$

*Newton's second law of motion*

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

*Newton's law of universal gravitation*



*Kepler*



*Bernoulli*



*Newton*



*Euler*



*Lagrange*



*Delaunay*



*Poincaré*

## 2-Body Problem

### 2 body problem.

- Can be solved analytically via Kepler's 3<sup>rd</sup> law.
- Bodies move around a common barycenter (center-of-mass) with elliptical orbits.

QuickTime™ and a  
H.264 decompressor  
are needed to see this picture.

## 3-Body Problem

**3-body problem.** No solution possible in terms of elementary functions; moreover, orbits may not be stable or periodic!

**Consequence.** Must resort to computational methods.

QuickTime™ and a  
H.264 decompressor  
are needed to see this picture.

# N-Body Simulation

N-body simulation. The ultimate object-oriented program:  
simulate the universe.

QuickTime™ and a  
H.264 decompressor  
are needed to see this picture.

## Body Data Type

Body data type. Represent a particle.

```
public class Body
```

---

```
    Body(Vector r, Vector v, double mass)
```

```
    void move(Vector f, double dt) apply force f, move body for dt seconds
```

```
    void draw() draw the ball
```

```
    Vector forceFrom(Body b) force vector between this body and b
```

Vector notation. Represent position, velocity, and force using `Vector`.

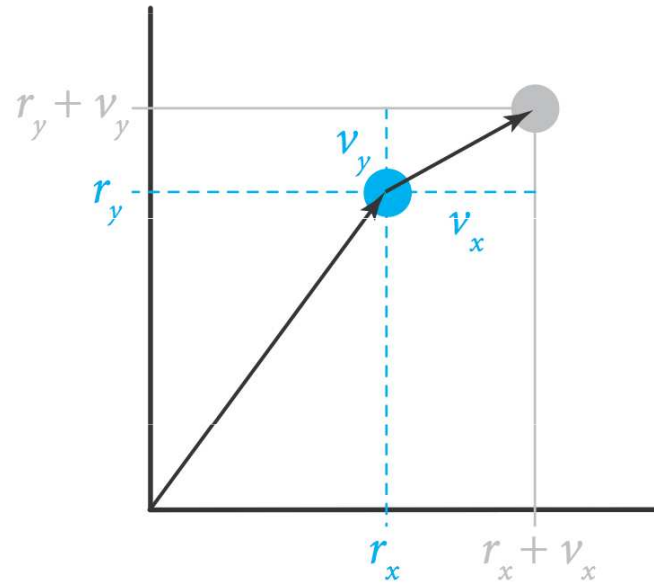
```
public class Body {  
    private Vector r;      // position  
    private Vector v;      // velocity  
    private double mass;   // mass
```

*instance variables*



## Moving a Body

Moving a body. Assuming no other forces, body moves in straight line.



```
r = r.plus(v.times(dt));
```

$$r_x = r_x + dt \cdot v_x$$

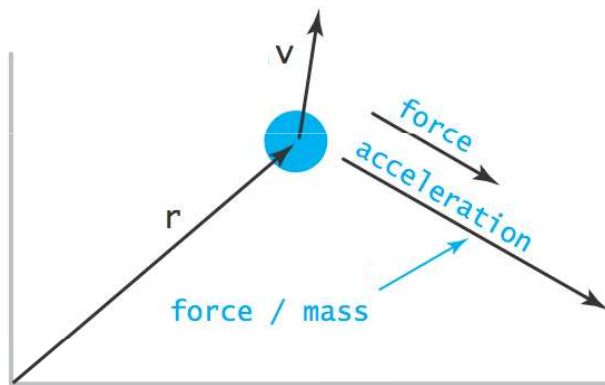
$$r_y = r_y + dt \cdot v_y$$

# Moving a Body

## Moving a body.

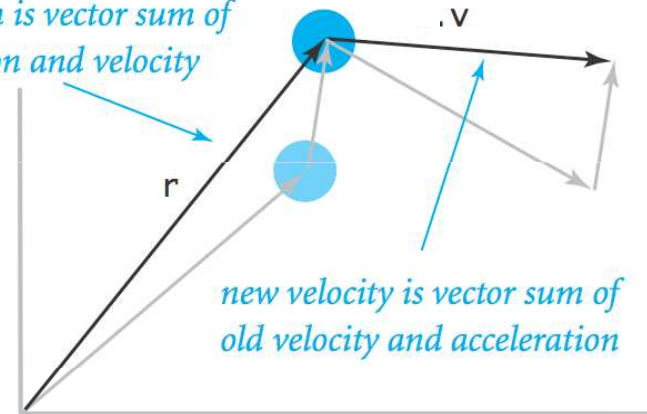
- Given external force  $F$ , acceleration  $a = F/m$ .
- Use acceleration (assume fixed) to compute change in velocity.
- Use velocity to compute change in position.

time  $t$



time  $t+1$

*new position is vector sum of  
old position and velocity*

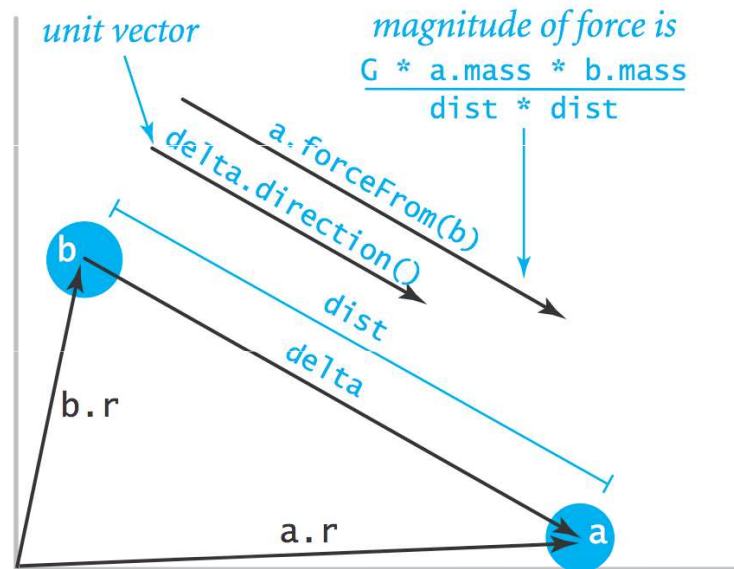


```
Vector a = f.times(1/mass);  
v = v.plus(a.times(dt));  
r = r.plus(v.times(dt));
```

## Force Between Two Bodies

Newton's law of universal gravitation.

- $F = G m_1 m_2 / r^2$ .
- Direction of force is line between two particles.



```
double G = 6.67e-11;
Vector delta = a.r.minus(b.r);
double dist = delta.magnitude();
double F = (G * a.mass * b.mass) / (dist * dist);
Vector force = delta.direction().times(F);
```

## Body Data Type: Java Implementation

```
public class Body {
    private Vector r;      // position
    private Vector v;      // velocity
    private double mass;   // mass

    public Body(Vector r, Vector v, double mass) {
        this.r = r;
        this.v = v;
        this.mass = mass;
    }

    public void move(Vector f, double dt) {
        Vector a = f.times(1/mass);
        v = v.plus(a.times(dt));
        r = r.plus(v.times(dt));
    }

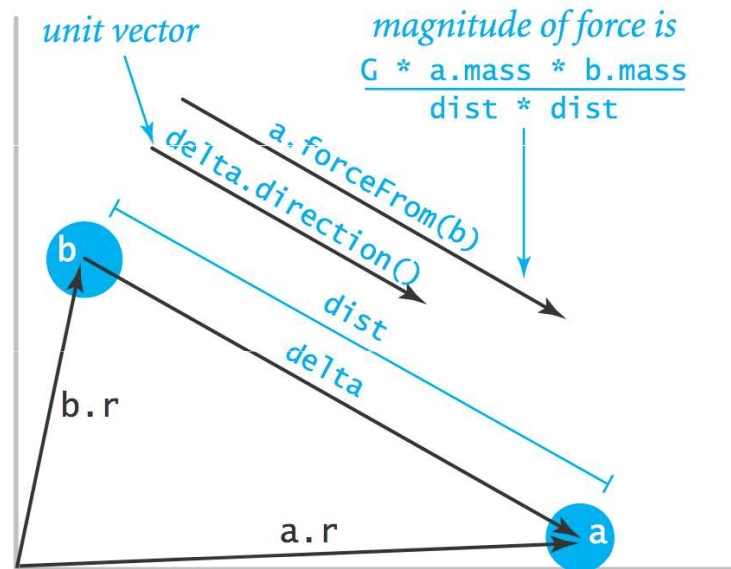
    public Vector forceTo(Body that) {
        double G = 6.67e-11;
        Vector delta = this.r.minus(that.r);
        double dist = delta.magnitude();
        double F = (G * this.mass * that.mass) / (dist * dist);
        return delta.direction().times(F);
    }

    public void draw() {
        StdDraw.setPenRadius(0.025);
        StdDraw.point(r.cartesian(0), r.cartesian(1));
    }
}
```

## Force Between Two Bodies

Newton's law of universal gravitation.

- $F = G m_1 m_2 / r^2$ .
- Direction of force is line between two particles.



```
double G = 6.67e-11;
Vector delta = a.r.minus(b.r);
double dist = delta.magnitude();
double F = (G * a.mass * b.mass) / (dist * dist);
Vector force = delta.direction().times(F);
```

# Universe Data Type

Universe data type. Represent a universe of  $N$  particles.

```
public class Universe
```

---

```
    Universe()
```

```
    void increaseTime(double dt)    simulate the passing of dt seconds
```

```
    void draw()                    draw the universe
```

```
public static void main(String[] args) {  
    Universe newton = new Universe();  
    double dt = Double.parseDouble(args[0]);  
    while (true) {  
        StdDraw.clear();  
        newton.increaseTime(dt);  
        newton.draw();  
        StdDraw.show(10);  
    }  
}
```

*main simulation loop*

# Universe Data Type

Universe data type. Represent a universe of N particles.

```
public class Universe
```

---

```
    Universe()
```

```
    void increaseTime(double dt)    simulate the passing of dt seconds
```

```
    void draw()                    draw the universe
```

```
public class Universe {  
    private double radius; // radius of universe  
    private int N          // number of particles  
    private Body[] orbs;   // the bodies
```

*instance variables*

# Data-Driven Design

## File format.

```
% more 4body.txt
4
5.0e10
-3.5e10 0.0e00 0.0e00 1.4e03 3.0e28
-1.0e10 0.0e00 0.0e00 1.4e04 3.0e28
1.0e10 0.0e00 0.0e00 -1.4e04 3.0e28
3.5e10 0.0e00 0.0e00 -1.4e03 3.0e28
```

Annotations:

- $N$  points to the first line (4).
- $radius$  points to the second line (5.0e10).
- $velocity$  points to the fourth column of the data rows (1.4e03, 1.4e04, -1.4e04, -1.4e03).
- $mass$  points to the fifth column of the data rows (3.0e28, 3.0e28, 3.0e28, 3.0e28).
- $position$  points to the first column of the data rows (-3.5e10, -1.0e10, 1.0e10, 3.5e10).

## Constructor.

```
public Universe() {
    N = StdIn.readInt();
    radius = StdIn.readDouble();
    StdDraw.setXscale(-radius, +radius);
    StdDraw.setYscale(-radius, +radius);

    // read in the N bodies
    orbs = new Body[N];
    for (int i = 0; i < N; i++) {
        double rx    = StdIn.readDouble();
        double ry    = StdIn.readDouble();
        double vx    = StdIn.readDouble();
        double vy    = StdIn.readDouble();
        double mass   = StdIn.readDouble();
        double[] position = { rx, ry };
        double[] velocity = { vx, vy };
        Vector r = new Vector(position);
        Vector v = new Vector(velocity);
        orbs[i] = new Body(r, v, mass);
    }
}
```



## Principle of Superposition

**Principle of superposition.** Net gravitational force acting on a body is the sum of the individual forces.

```
// compute the forces
for (int i = 0; i < N; i++) {
    for (int j = 0; j < N; j++) {
        if (i != j) {
            f[i] = f[i].plus(orbs[j].forceTo(orbs[i]));
        }
    }
}
```

$$F_i = \sum_{i \neq j} \frac{G m_i m_j}{|r_i - r_j|^2}$$

# Universe Data Type: Java Implementation

```
public class Universe {  
    private final double radius;      // radius of universe  
    private final int N;              // number of bodies  
    private final Body[] orbs;       // array of N bodies
```

```
    public Universe() { /* see previous slide */ }
```

create  
universe

```
    public void increaseTime(double dt) {  
        Vector[] f = new Vector[N];  
        for (int i = 0; i < N; i++)  
            f[i] = new Vector(new double[2]);  
        for (int i = 0; i < N; i++)  
            for (int j = 0; j < N; j++)  
                if (i != j)  
                    f[i] = f[i].plus(orbs[j].forceTo(orbs[i]));  
  
        for (int i = 0; i < N; i++)  
            orbs[i].move(f[i], dt);  
    }
```

update  
the bodies

```
    public void draw() {  
        for (int i = 0; i < N; i++)  
            orbs[i].draw();  
    }
```

draw the bodies

simulate the universe

```
    public static void main(String[] args) { /* see previous slide */ }
```

```
}
```

## Odds and Ends

**Accuracy.** How small to make  $\Delta t$ ? How to avoid floating-point inaccuracies from accumulating?

**Efficiency.**

- Direct sum: takes time proportional to  $N^2$   
 $\Rightarrow$  not usable for large  $N$ .
- Appel / Barnes-Hut: takes time proportional to  $N \log N$  time  
 $\Rightarrow$  can simulate large universes.

**3D universe.** Use a 3D vector (only drawing code changes!).

**Collisions.**

- Model inelastic collisions.
- Use a softening parameter to avoid collisions.

$$F_i = \sum_{i \neq j} \frac{G m_i m_j}{|r_i - r_j|^2 + \epsilon^2}$$

# Extra Slides

---

# N-Body Simulation

## 1. Setup initial distribution of particles.

- Need accurate data and model of mass distribution.

## 2. Compute forces between particles.

- Direct sum:  $N^2$ .
- Appel / Barnes-Hut"  $N \log N$ .

$$\mathbf{F}_i = \sum_{i \neq j} \frac{Gm_i m_j}{|\mathbf{r}_i - \mathbf{r}_j|^2 + \epsilon^2}$$

$\epsilon$  = softening parameter  
eliminates binary stars with  $r < \epsilon$   
hard binaries can be important  
source of energy

## 3. Evolve particles using ODE solver.

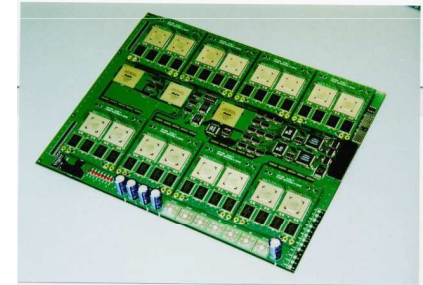
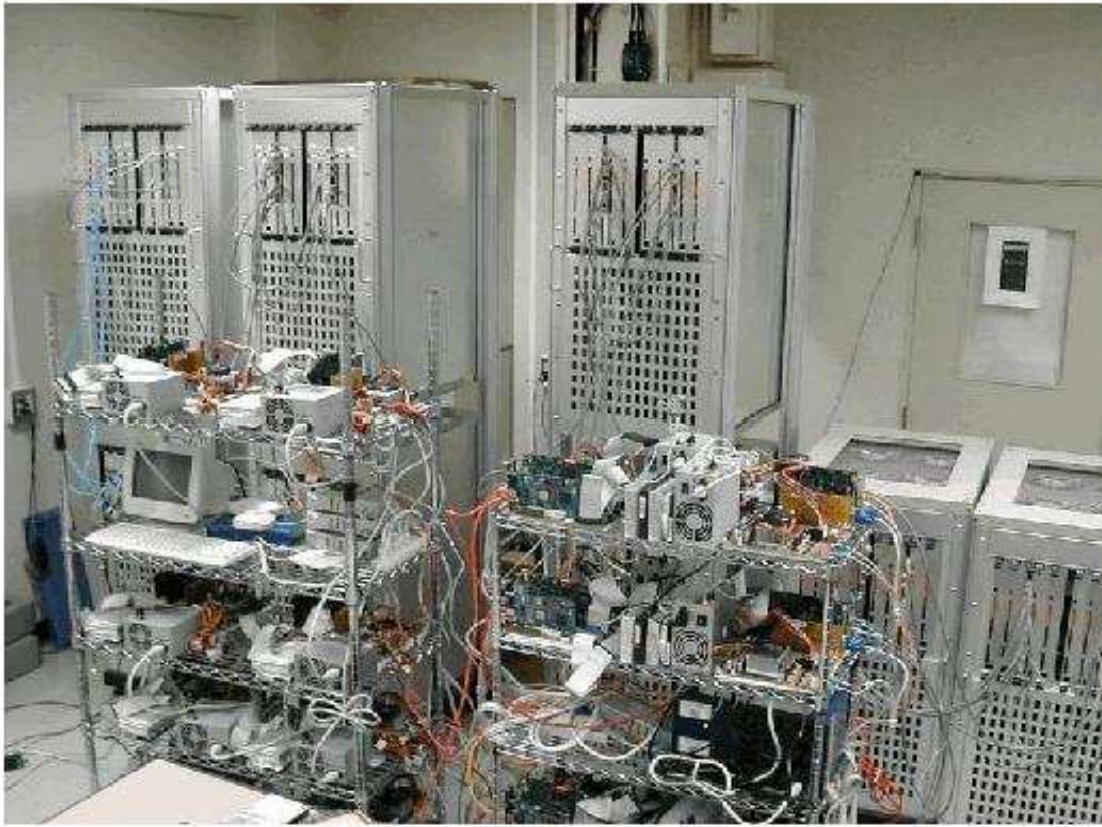
- Leapfrog method balances efficiency and accuracy.
- Truncation error =  $O(\Delta t^2)$ .
- Symplectic.

$$\frac{d\mathbf{X}_i}{dt} = \mathbf{V}_i$$
$$m_i \frac{d\mathbf{V}_i}{dt} = \mathbf{F}_i$$

## 4. Display and analyze results.

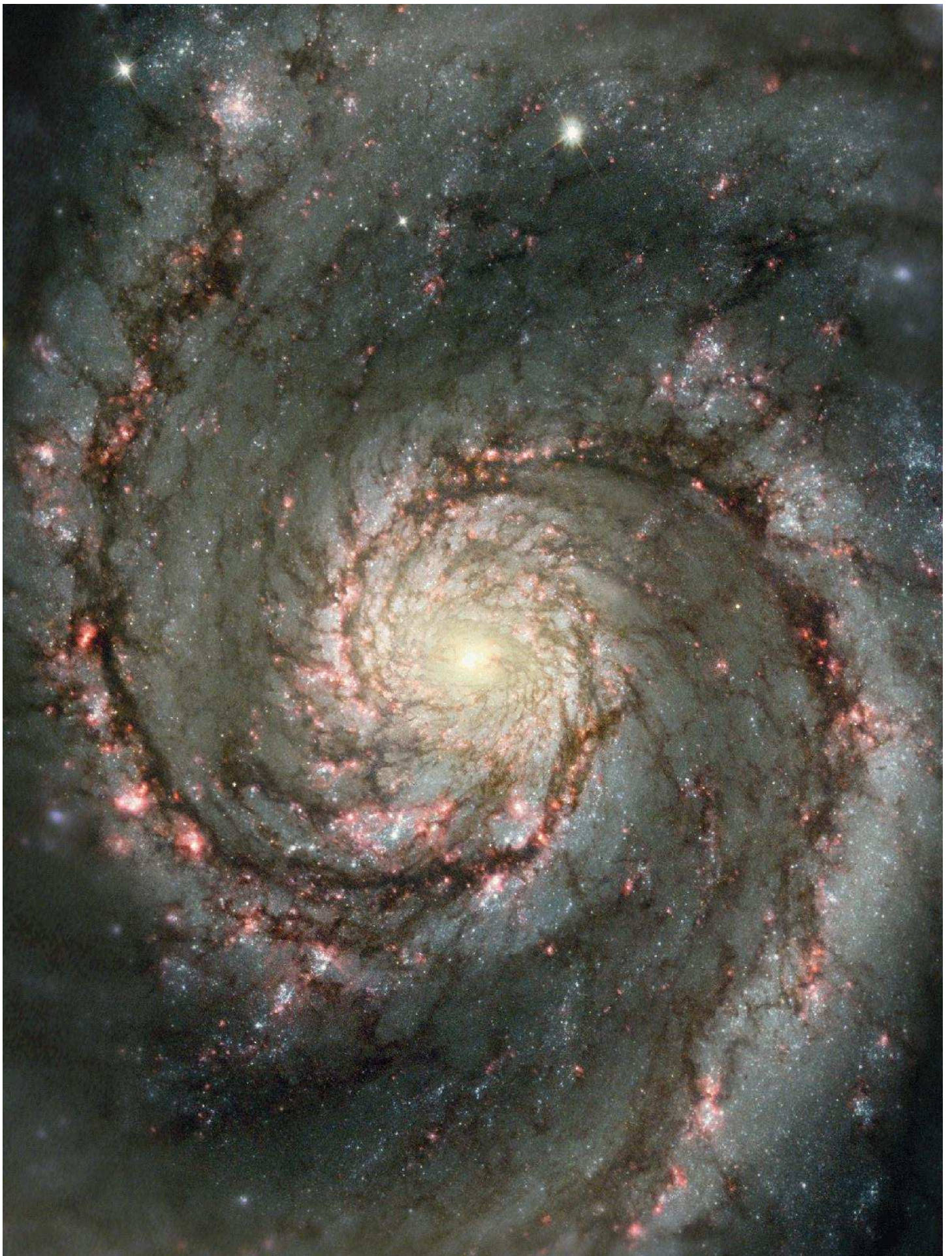
Solving the force problem with hardware.

GRAPE-6. Special purpose hardware to compute force.



Jun Makino, U. Tokyo







Do we really need to compute force from every star for distant objects?



Andromeda – 2 million light years away

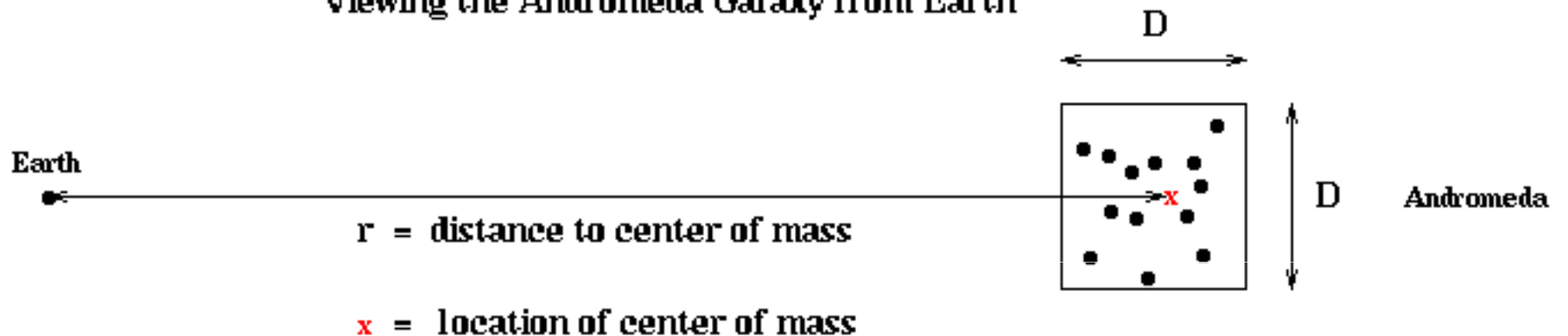


# Solving the force problem with software -- tree codes



← Distance = 25 times size →

Viewing the Andromeda Galaxy from Earth



Organize particles into a tree. In Barnes-Hut algorithm, use a quadtree in 2D

**A Complete Quadtree with 4 Levels**

