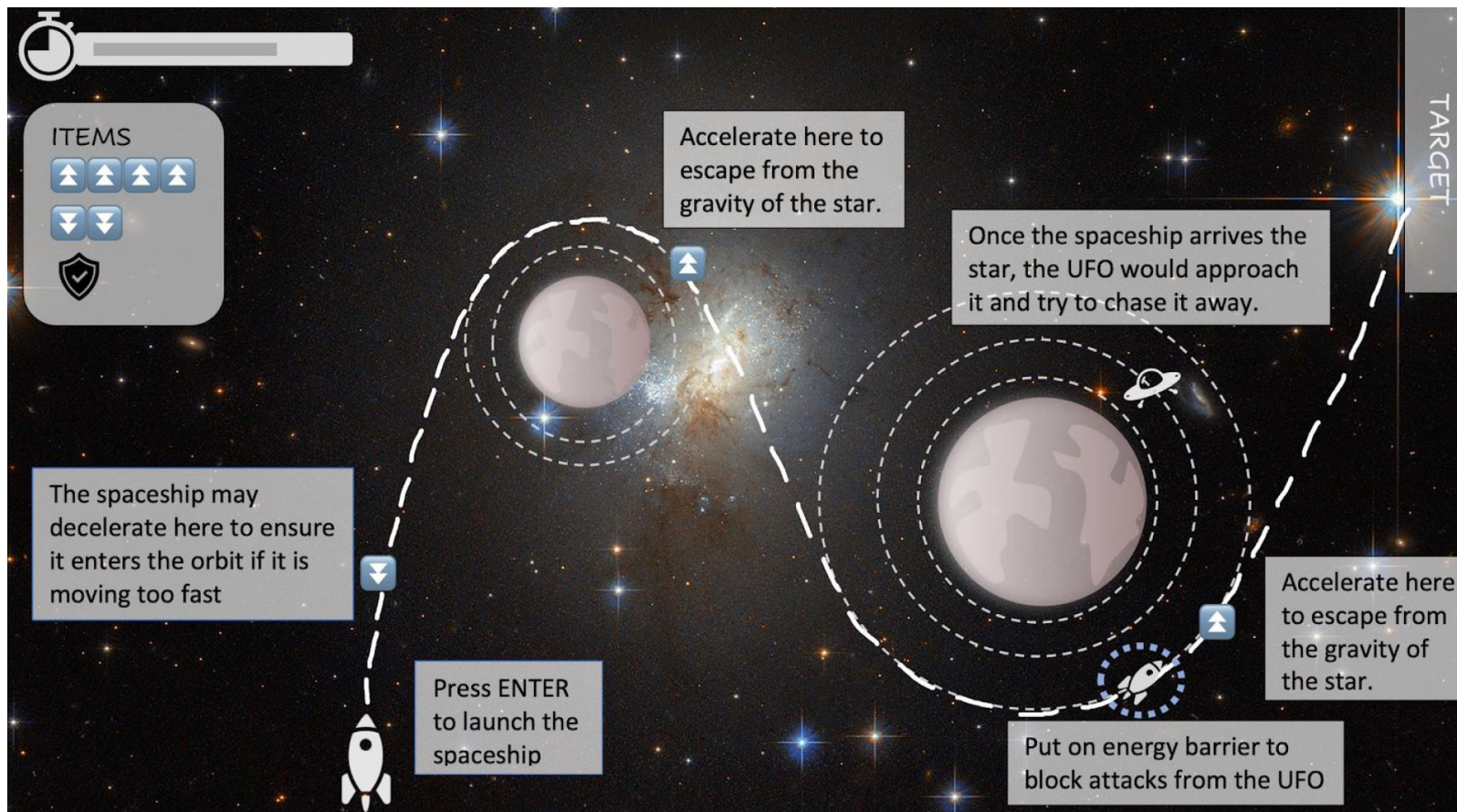


Physical Simulation



Overview

1. Equation of Motion

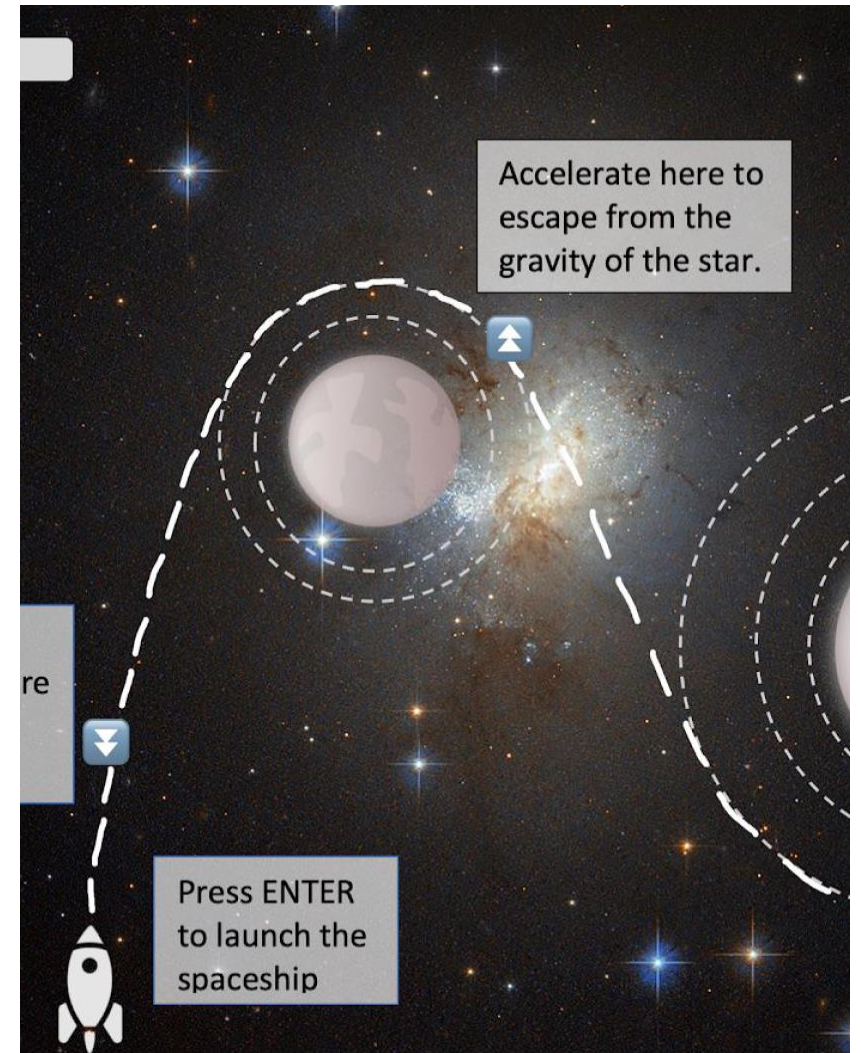
- Examples
- Ordinary Differentiable Equations (ODE)
- Solving ODEs

2. Collision and Reaction Forces

Physics

Learning goals:

- ***Connect your theoretical math knowledge to applications***
- ***Properly simulate object motion and their interaction in your game***



Basic Particle Simulation (first try)

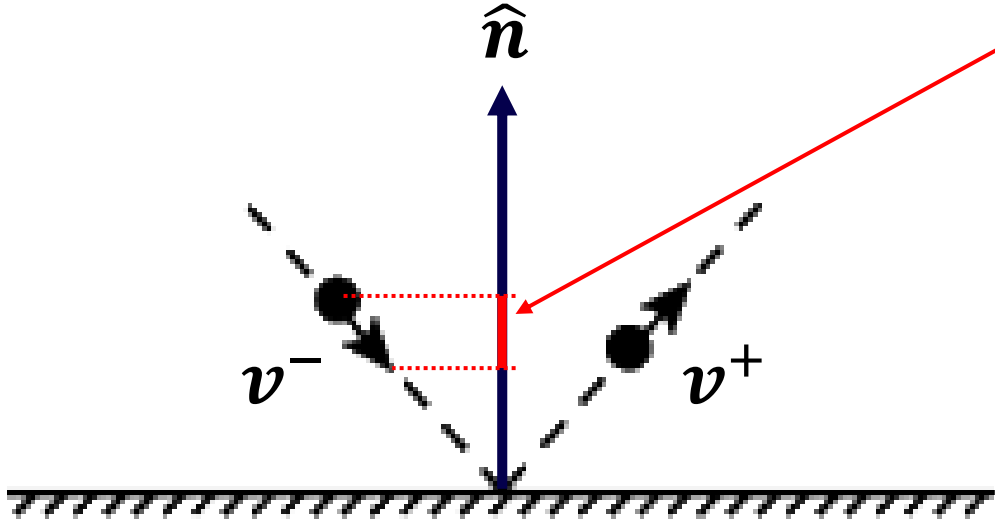
How to compute the change in velocity?

$$\begin{aligned}d_t &= t_{i+1} - t_i \\ \vec{v}_{i+1} &= \vec{v}_i + \Delta \mathbf{v} \\ \vec{p}_{i+1} &= \vec{p}(t_i) + \vec{v}_i d_t\end{aligned}$$



Particle-Plane Collision

- *In direction of normal*



Velocity along normal
(v projected on normal
by the dot product)

Frictionless

$$\Delta v = 2(v^- \cdot \hat{n})\hat{n}$$

Apply change
along normal
(magnitude
times direction)

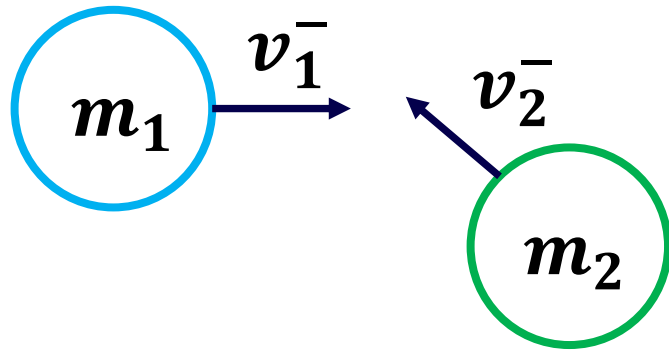
$$v^+ = v^- + \Delta v$$

Loss of energy

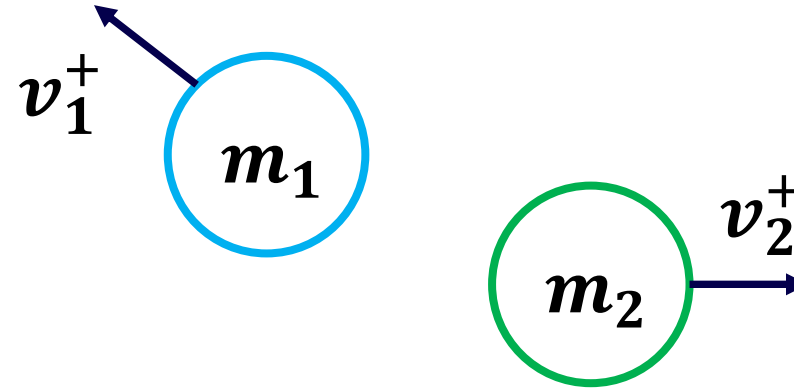
$$\Delta v = (1 + \epsilon)(v^- \cdot \hat{n})\hat{n}$$

Particle-Particle Collisions (spherical objects)

Before collision



After



Response:

$$v_1^+ = v_1^- - \frac{2m_2}{m_1 + m_2} \frac{\langle v_1^- - v_2^- \rangle \cdot \langle p_1 - p_2 \rangle}{\|p_1 - p_2\|^2} \langle p_1 - p_2 \rangle$$

$$v_2^+ = v_2^- - \frac{2m_1}{m_1 + m_2} \frac{\langle v_2^- - v_1^- \rangle \cdot \langle p_2 - p_1 \rangle}{\|p_2 - p_1\|^2} \langle p_2 - p_1 \rangle$$

- This is in terms of velocity
- next: derivation via impulse and forces

From Velocities (Δv) to Forces (F) and back

Force relates to mass and acceleration

$$\mathbf{F} = ma$$

A change in velocity related to acceleration over time

$$\Delta \mathbf{v} = \Delta t a$$

In terms of forces

$$\Delta \mathbf{v} = \Delta t \frac{F}{m}$$

Basic Particle Simulation (first try)

How to compute the change in velocity?

$$\begin{aligned}d_t &= t_{i+1} - t_i \\ \vec{v}_{i+1} &= \vec{v}_i + \Delta \mathbf{v} \\ \vec{p}_{i+1} &= \vec{p}(t_i) + \vec{v}_i d_t\end{aligned}$$



Forces are omnipresent

- **Gravity**

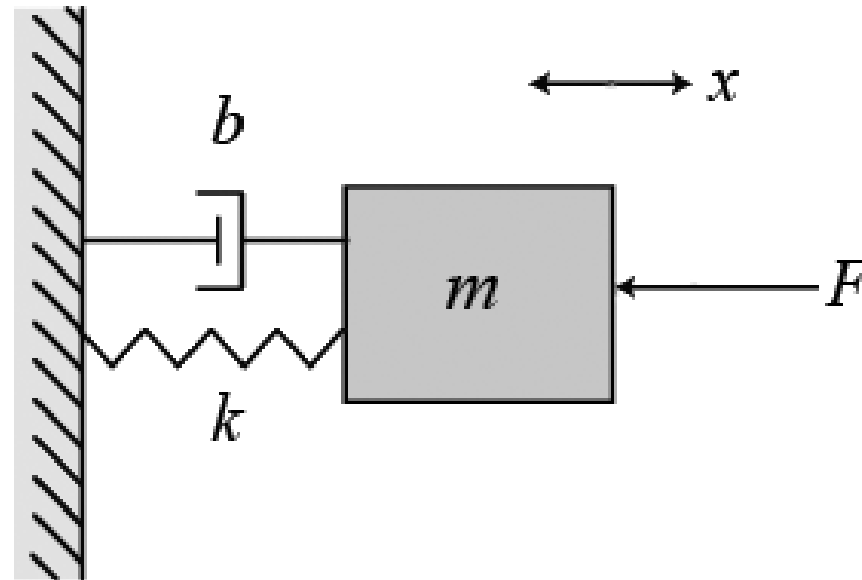
$$F = \begin{bmatrix} 0 \\ -mg \end{bmatrix}$$

- **Viscous damping**

$$F = -bv$$

- **Spring & dampers**

$$F = -kx - bv$$



Gravity direction?

Assuming a flat earth:

$$F = \begin{bmatrix} 0 \\ -mg \end{bmatrix}$$

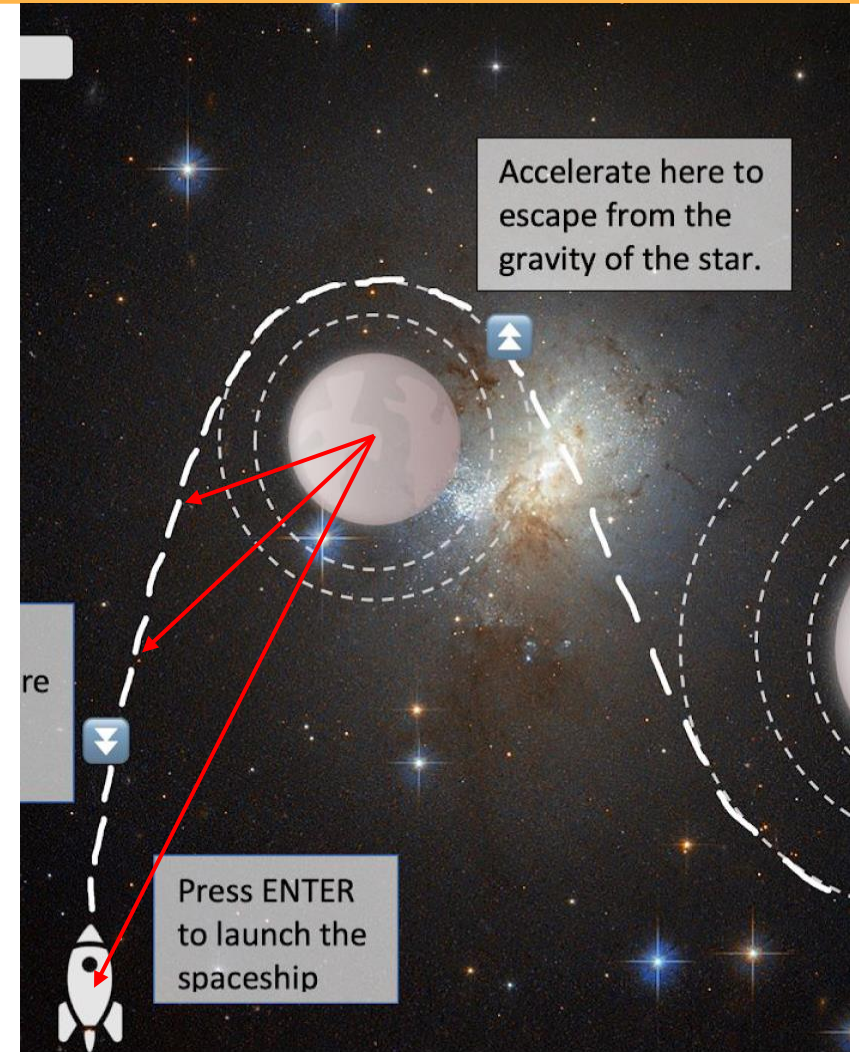
Assuming a spherical earth:

$$F = -mg \begin{bmatrix} a \\ b \end{bmatrix}$$

How to compute the vector (a,b) and g ?

Newton's law of universal gravitation

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



Multiple forces?

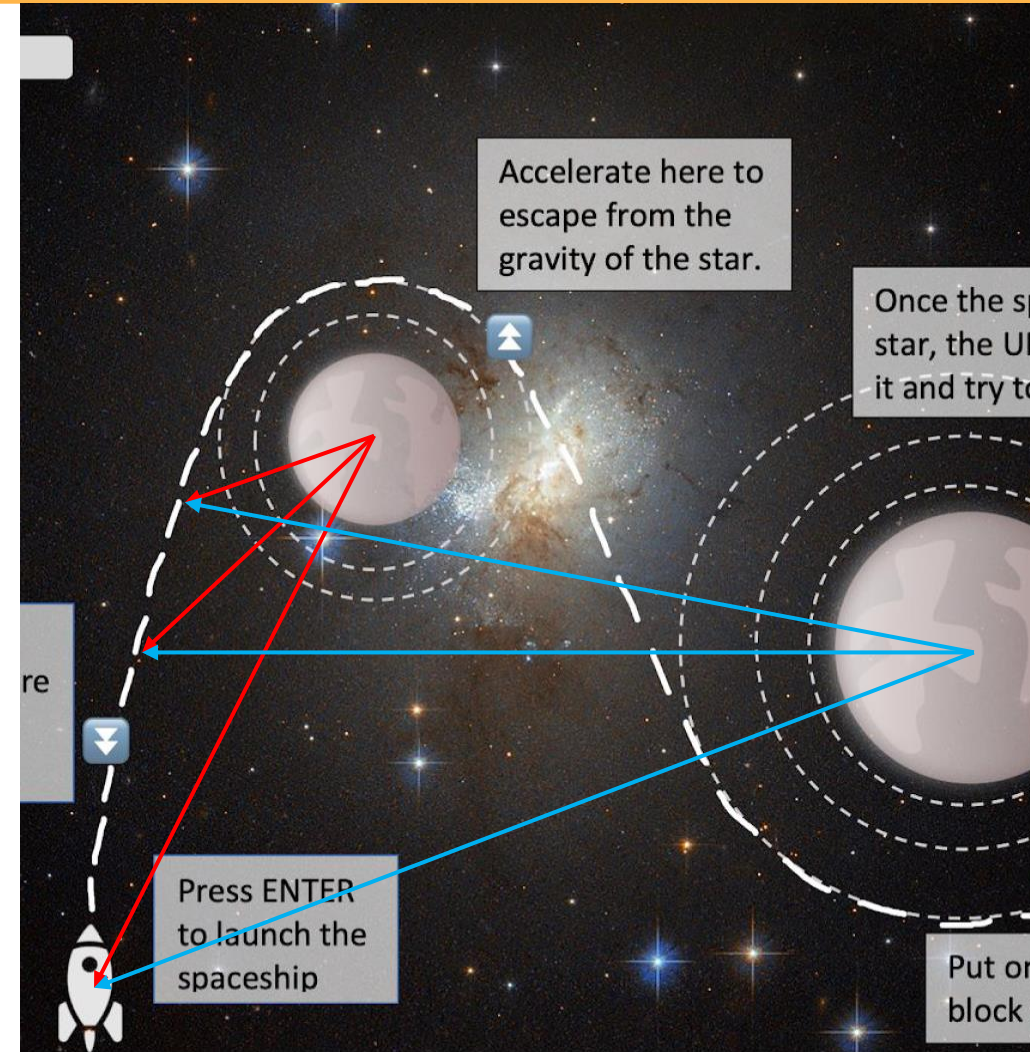
Forces add up (and cancel):

$$F = -m g_1 \begin{bmatrix} a_1 \\ b_1 \end{bmatrix} - m g_2 \begin{bmatrix} a_2 \\ b_2 \end{bmatrix}$$

- ***This holds for all types of forces!***
- ***Notation you might see:***

$$F = \sum_i F_i = \sum F_i = \sum F$$

$$\vec{F} = F$$



Your game idea does not need forces?

Are you sure?

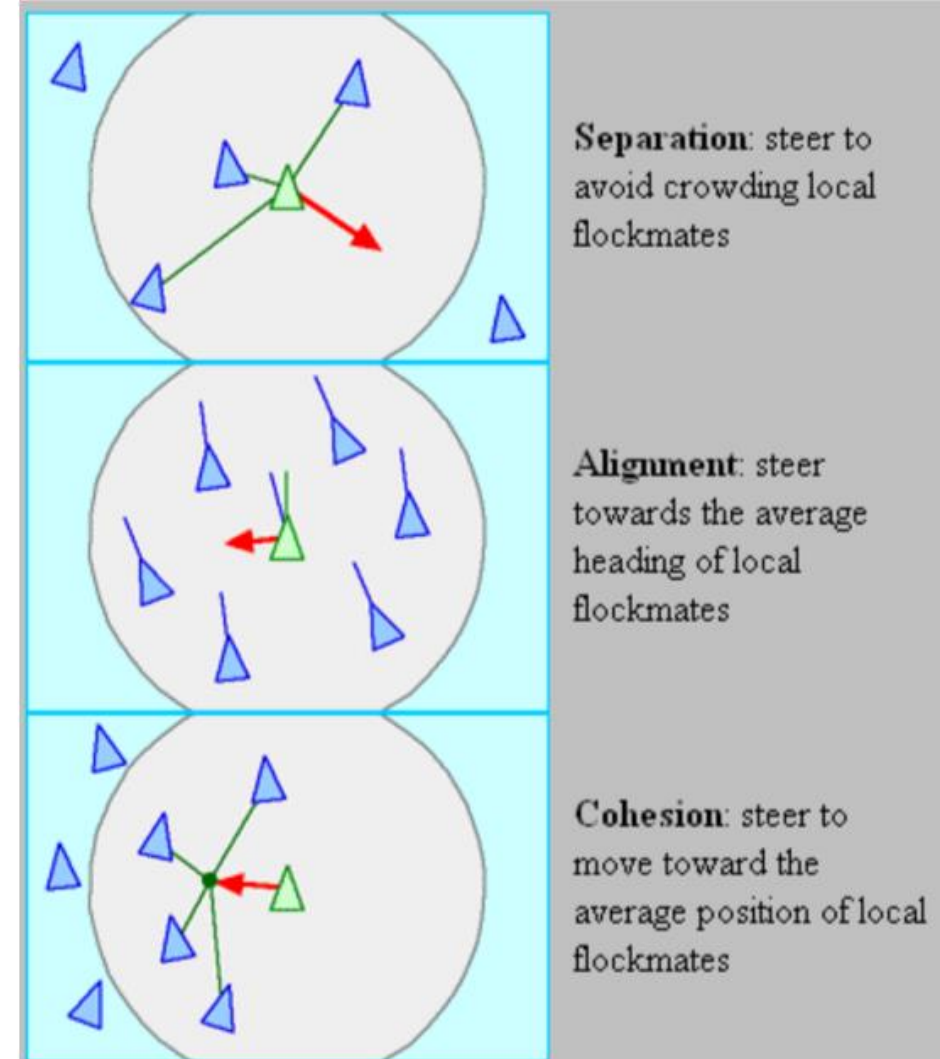
- *Particle effects*
- *Fake forces*
- *Proxy forces*
 - Simulate crowd behaviour



Take it as a chance to connect dry math with a practical application!

Proxy Forces (= fake forces)

- Behavior forces: [“Boids”, Craig Reynolds, SIGGRAPH 1987]
- flocking birds, schooling fish, etc.
- Attract to goal location (like gravity)
 - *E.g., waypoint determined by shortest path search*
- Repulsion if close
- Align orientation to neighbors
- Center to neighbors
- Forces add up!



Simulation Basics

Simulation loop...

1. *Equations of Motion*

- sum forces & torques
- solve for accelerations: $\vec{F} = m\vec{a}$

2. *Numerical integration*

- update positions, velocities

3. *Collision detection*

4. *Collision resolution*

What we did so far: Forward Euler

Forces only $\vec{F} = m\vec{a}$

$$d_t = t_{i+1} - t_i$$

acceleration = $\frac{\partial v}{\partial t}$

$$\vec{v}_{i+1} = \vec{v}_i + (\vec{F}(t_i)/m)d_t$$

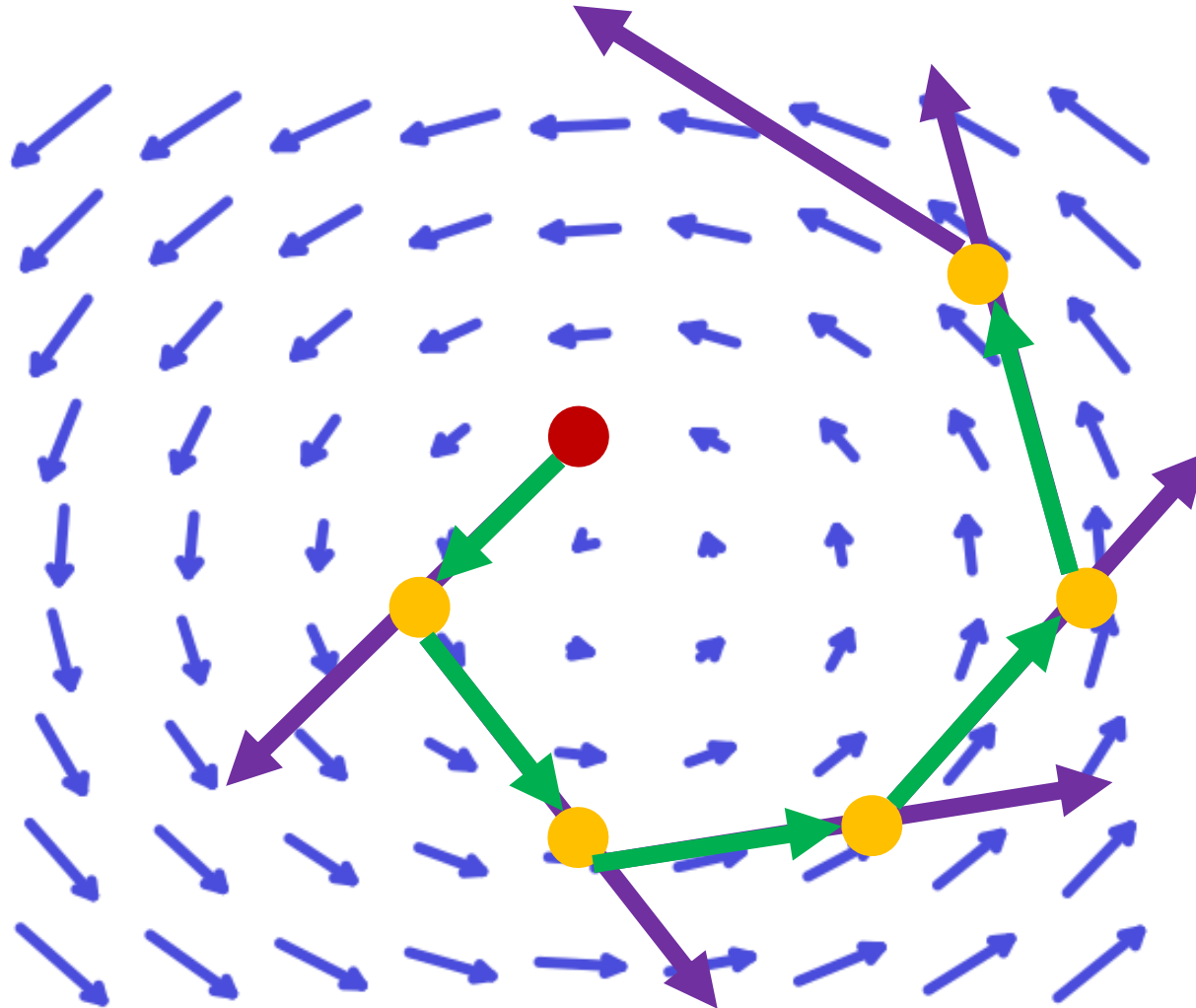
$$\vec{p}_{i+1} = \vec{p}(t_i) + \vec{v}_{i+1}d_t$$

get values at time t_{i+1} from values at time t_i

Issues? Alternatives?

How can we discretize this?

Issue: extrapolation

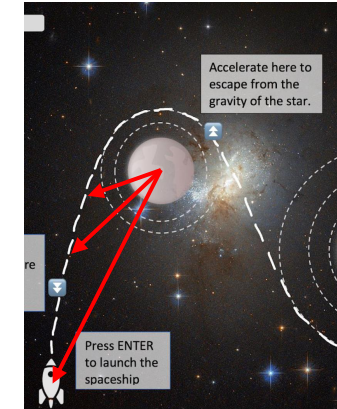


Which forces depend on t?

- Gravity

$$F = \begin{bmatrix} 0 \\ -mg \end{bmatrix}$$

$$F = -mg \begin{bmatrix} a \\ b \end{bmatrix}$$

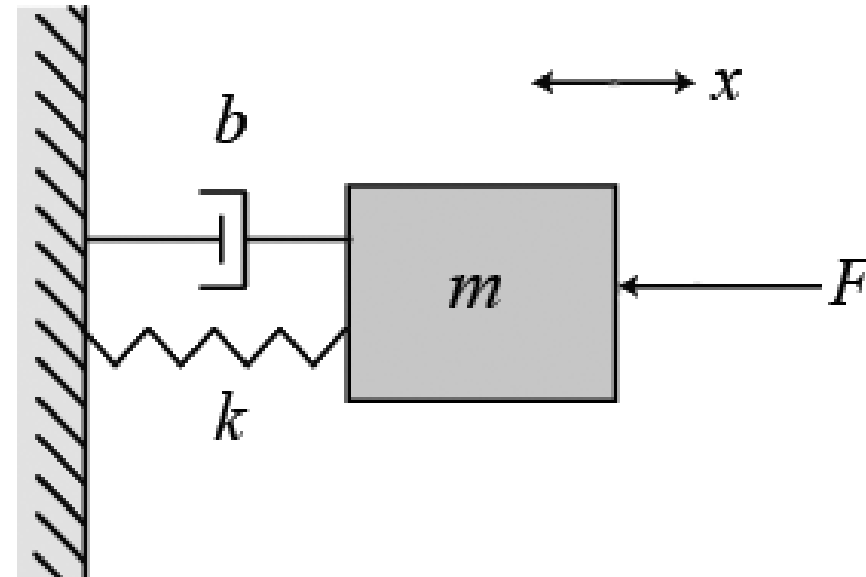


- Viscous damping

$$F = -bv$$

- Spring & dampers

$$F = -kx - bv$$



Basic Particle Simulation: Small Problem...

$$\begin{aligned}d_t &= t_{i+1} - t_i \\ \vec{v}_{i+1} &= \vec{v}_i + (\vec{F}(t_{???})/m)d_t \\ \vec{p}_{i+1} &= \vec{p}(t_i) + \vec{v}_{i+1}d_t\end{aligned}$$

Equations of motion describe state (equilibrium)

- **Involves quantities and their derivatives**
 - **-> we need to solve differential equations**

Lets start from scratch

Given:

$$\vec{F} = m \frac{\partial^2 x}{\partial t^2}$$

Wait!

**There is no position x in this equation?!
Only contains acceleration a !**

How to solve such differential equation?

Desired: the position x at time t


x

Newtonian Physics as First-Order Diff. Eq. (DE)

Second-order DE

$$\vec{F} = m \frac{\partial^2 x}{\partial t^2}$$

acceleration
 $= \frac{\partial v}{\partial t}$




Now we have an x!

First-order DE

$$\frac{\partial}{\partial t} \begin{bmatrix} \vec{x} \\ \vec{v} \end{bmatrix} = \begin{bmatrix} \vec{v} \\ \vec{F}/m \end{bmatrix}$$

velocity
 $= \frac{\partial x}{\partial t}$



Higher-order DEs can be turned into a first-order DE with additional variables and equations!

Newtonian Physics as First-Order DE

- Motion of **one** particle

Second-order DE

$$\vec{F} = m \frac{\partial^2 \vec{x}}{\partial t^2}$$

First-order DE

$$\frac{\partial}{\partial t} \begin{bmatrix} \vec{x} \\ \vec{v} \end{bmatrix} = \begin{bmatrix} \vec{v} \\ \Sigma \vec{F}/m \end{bmatrix}$$

- Motion of **many** particles

$$\frac{\partial}{\partial t} \begin{bmatrix} \vec{x}_1 \\ \vec{v}_1 \\ \vec{x}_2 \\ \vec{v}_2 \\ \vdots \\ \vec{x}_n \\ \vec{v}_n \end{bmatrix} = \begin{bmatrix} \vec{v}_1 \\ \vec{F}_1/m_1 \\ \vec{v}_2 \\ \vec{F}_2/m_2 \\ \vdots \\ \vec{v}_n \\ \vec{F}_n/m_n \end{bmatrix}$$

Overview

Different DE solvers

- ***Forward Euler***
(take current accel. to update vel., current vel. to update pos.)
- ***Midpoint Method & Trapezoid Method***
(mix current and approximations of future vel. & acc. Estimates)
- ***Backwards Euler***
(solve for future pos., vel., and accel. jointly)
 - *May require an iterative solver*

Recap: Forward Euler

Forces only $\vec{F} = m\vec{a}$

$$d_t = t_{i+1} - t_i$$

acceleration = $\frac{\partial v}{\partial t}$

$$\vec{v}_{i+1} = \vec{v}_i + (\vec{F}(t_i)/m)d_t$$

$$\vec{p}_{i+1} = \vec{p}(t_i) + \vec{v}_{i+1}d_t$$

get values at time t_{i+1} from values at time t_i

Issues? Alternatives?



Idea: Backwards Euler

$$\begin{aligned}d_t &= t_{i+1} - t_i \\ \vec{v}_{i+1} &= \vec{v}_i + (\vec{F}(t_{i+1})/m)d_t \\ \vec{p}_{i+1} &= \vec{p}(t_i) + \vec{v}_{i+1}d_t\end{aligned}$$

get values at time t_{i+1} from states at time t_i and forces at t_{i+1}

Issues?

- Viscous damping

$$F = -bv$$

- Spring & dampers

$$F = -kx - bv$$



Differential Equations

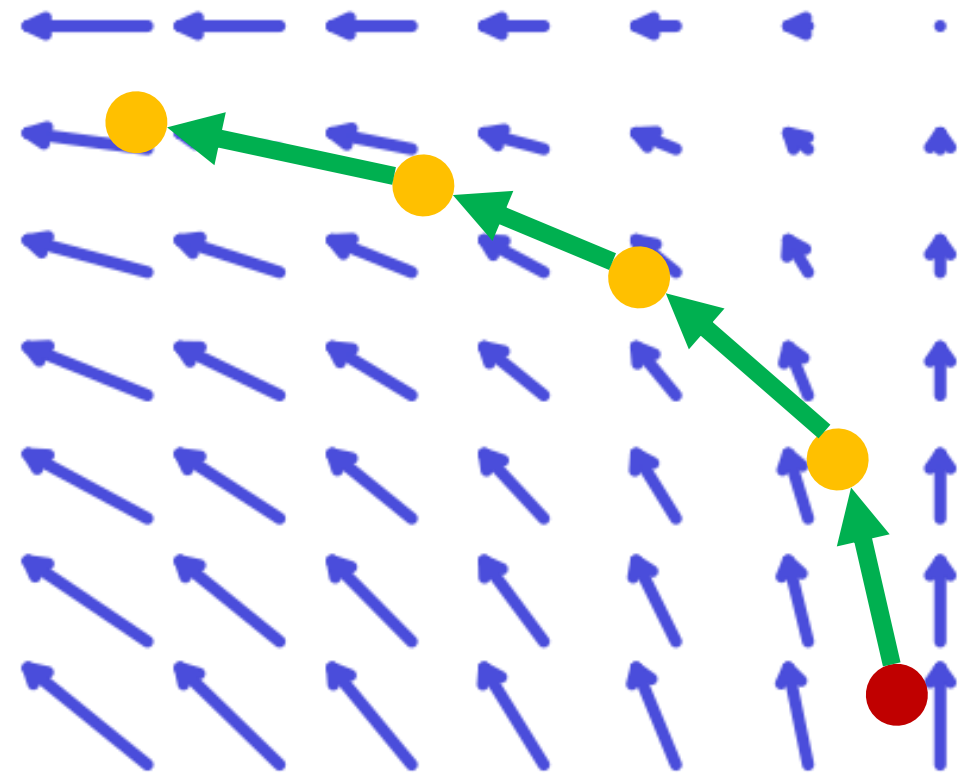
$$\frac{\partial}{\partial t} \vec{X}(t) = f(\vec{X}(t), t)$$

Given that $\vec{X}_0 = \vec{X}(t_0)$

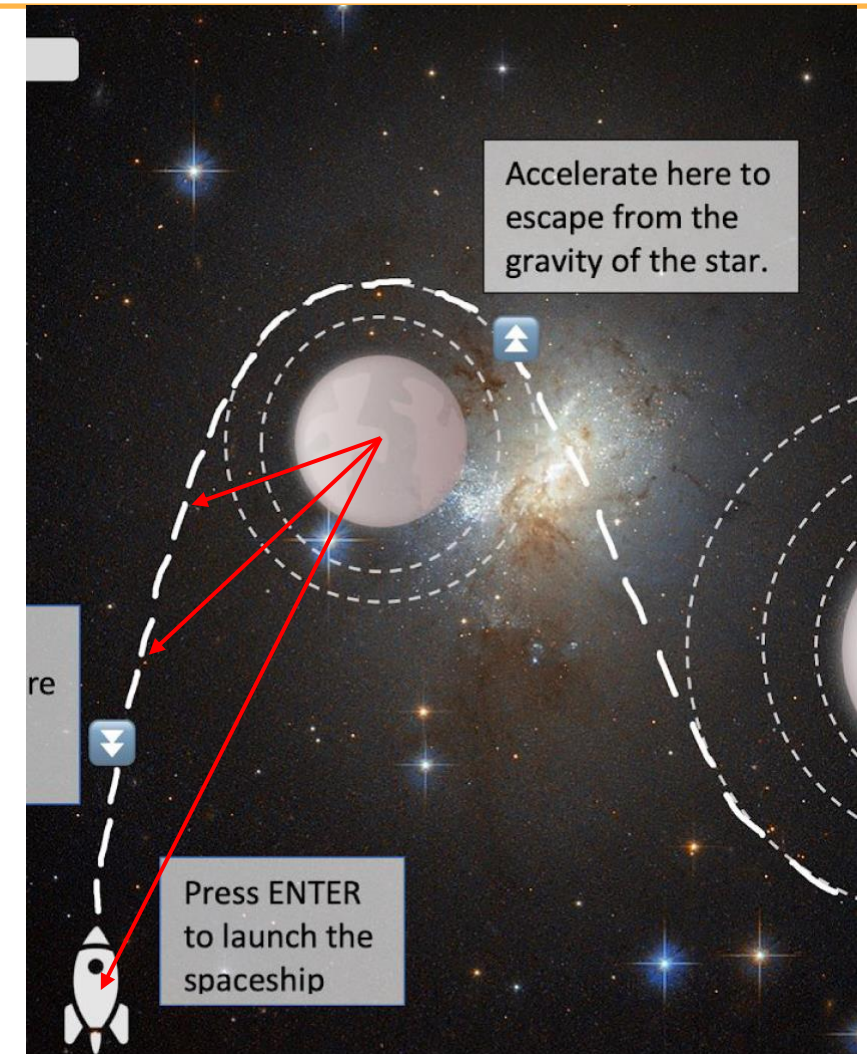
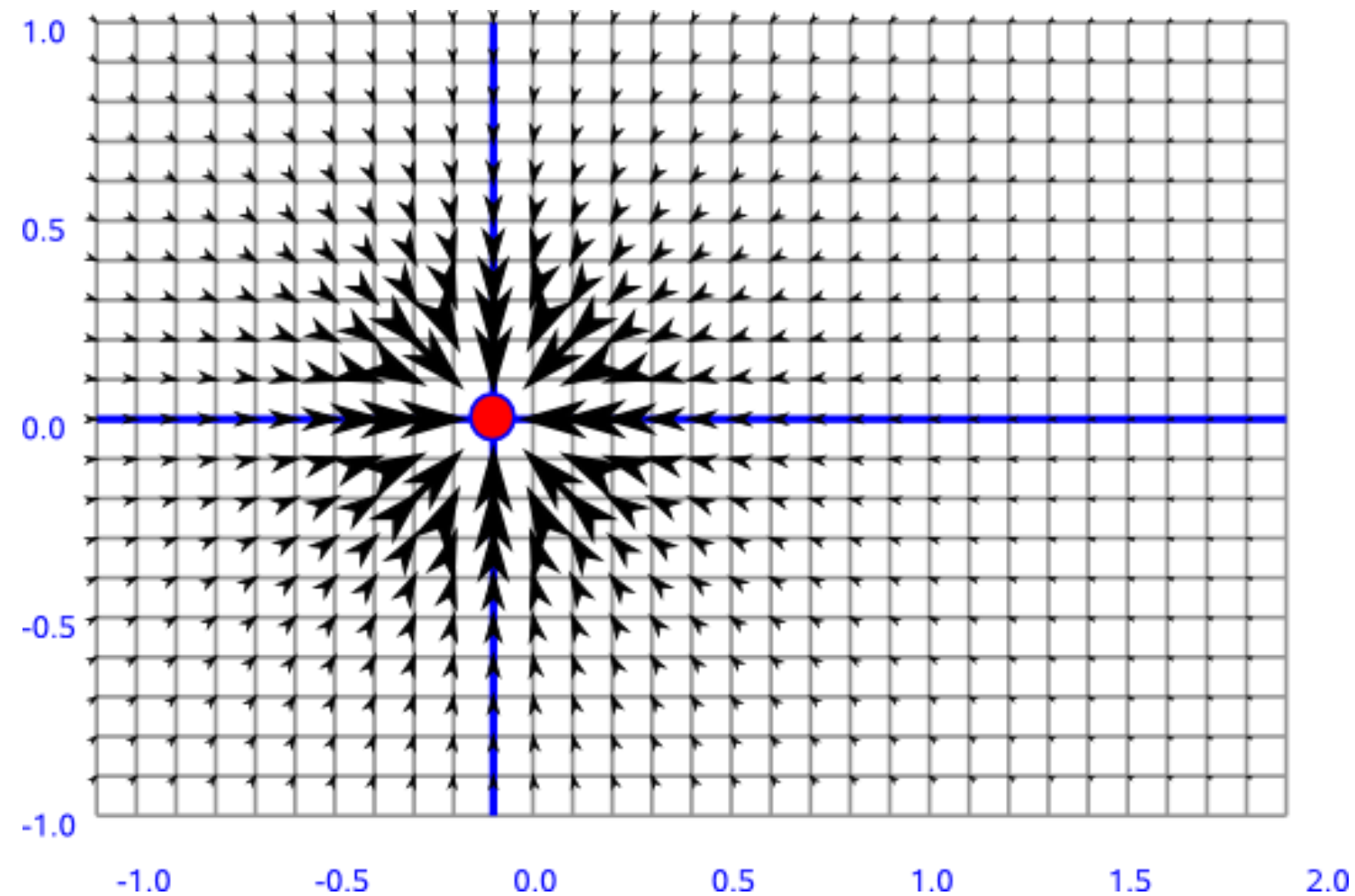
Compute $\vec{X}(t)$ for $t > t_0$

$$\Delta \vec{X}(t) = f(\vec{X}(t), t) \Delta t$$

- **Simulation:**
 - *path through state-space*
 - *driven by vector field*



Gravitational field



DE Numerical Integration: Explicit (Forward) Euler

$$\frac{\partial}{\partial t} \vec{X}(t) = f(\vec{X}(t), t)$$

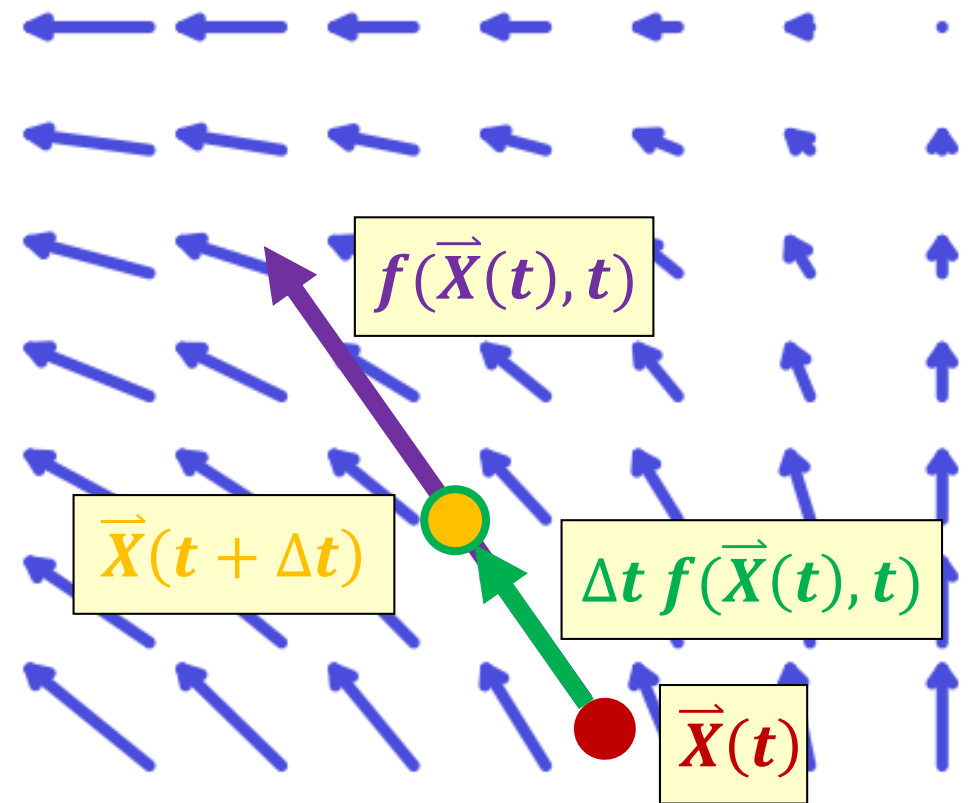
Given that $\vec{X}_0 = \vec{X}(t_0)$

Compute $\vec{X}(t)$ **for** $t > t_0$

$$\Delta t = t_i - t_{i-1}$$

$$\Delta \vec{X}(t_{i-1}) = \Delta t f(\vec{X}(t_{i-1}), t_{i-1})$$

$$\vec{X}_i = \vec{X}_{i-1} + \Delta t f(\vec{X}_{i-1}, t_{i-1})$$

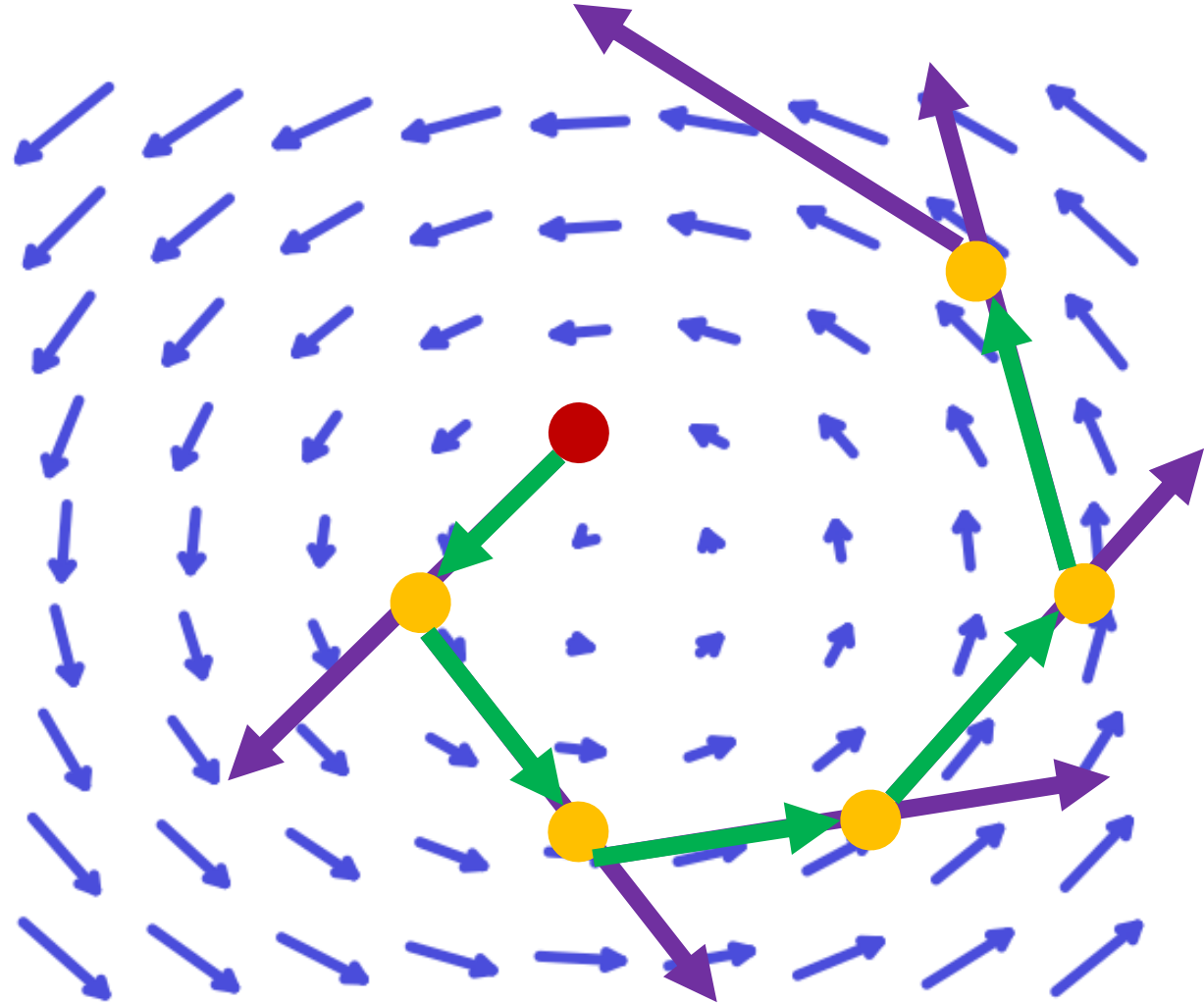


Explicit Euler Problems

- Solution **spirals** out
 - *Even with **small time steps***
 - *Although smaller time steps are still **better***

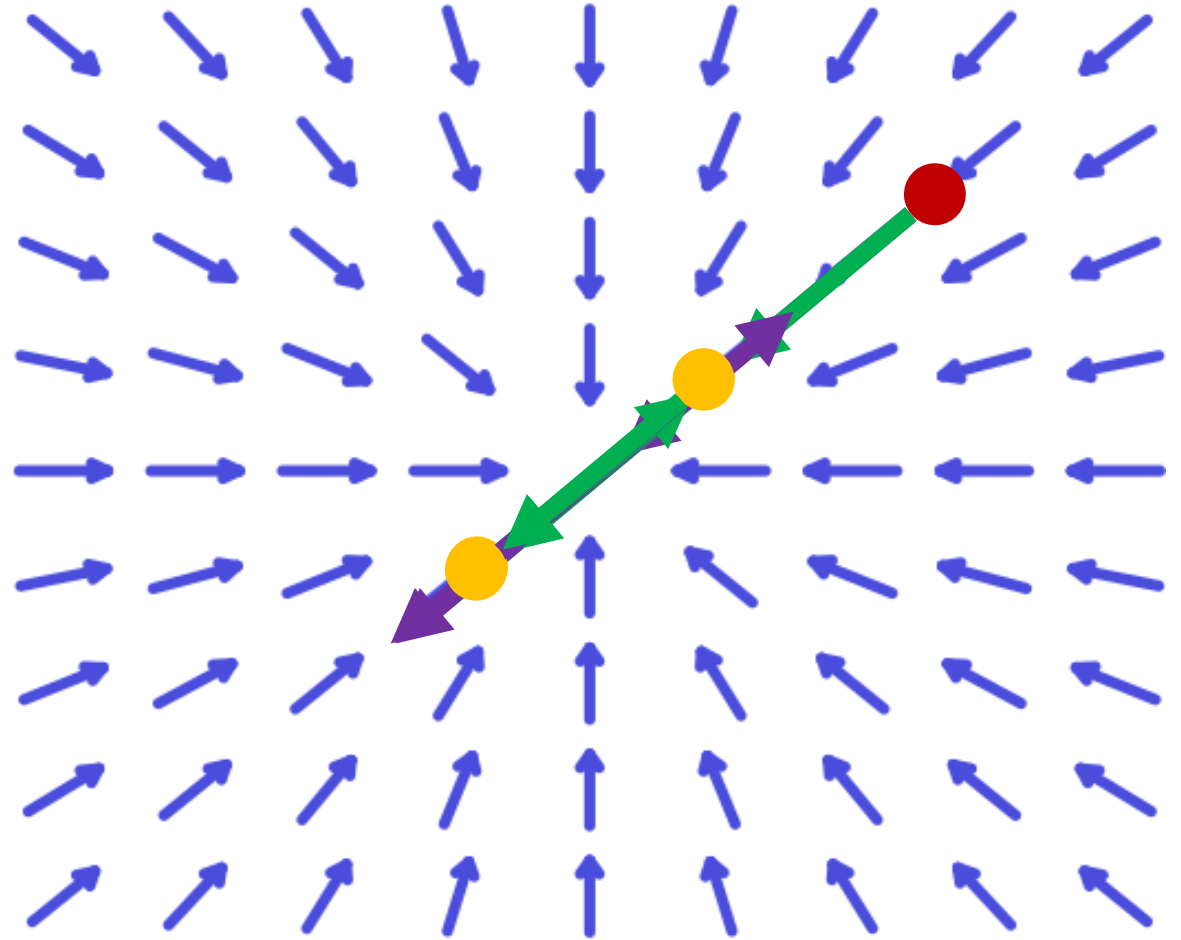
Definition: Explicit

- **Closed-form/analytic solution**
- **no iterative solve required**



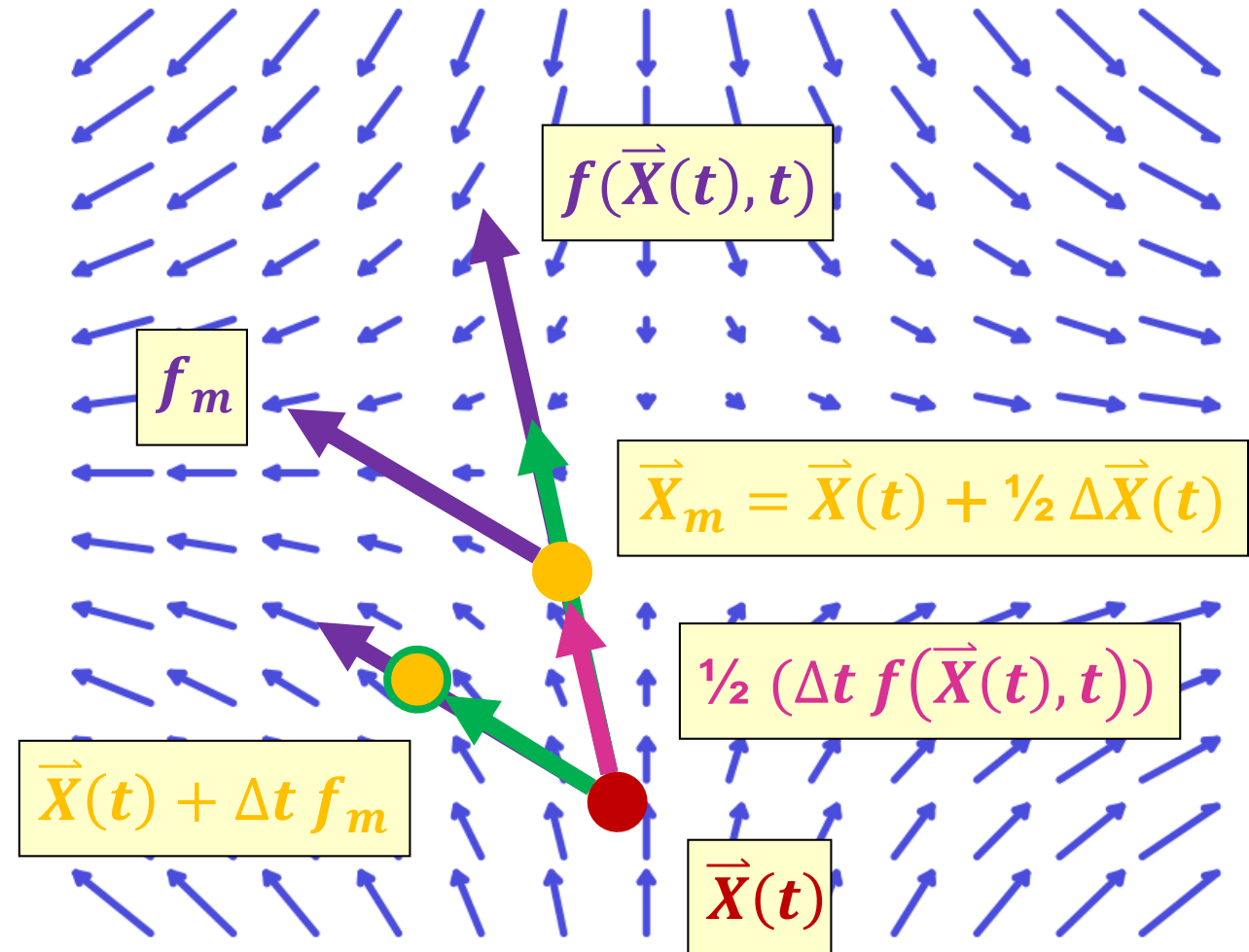
Explicit Euler Problems

- Can lead to **instabilities**



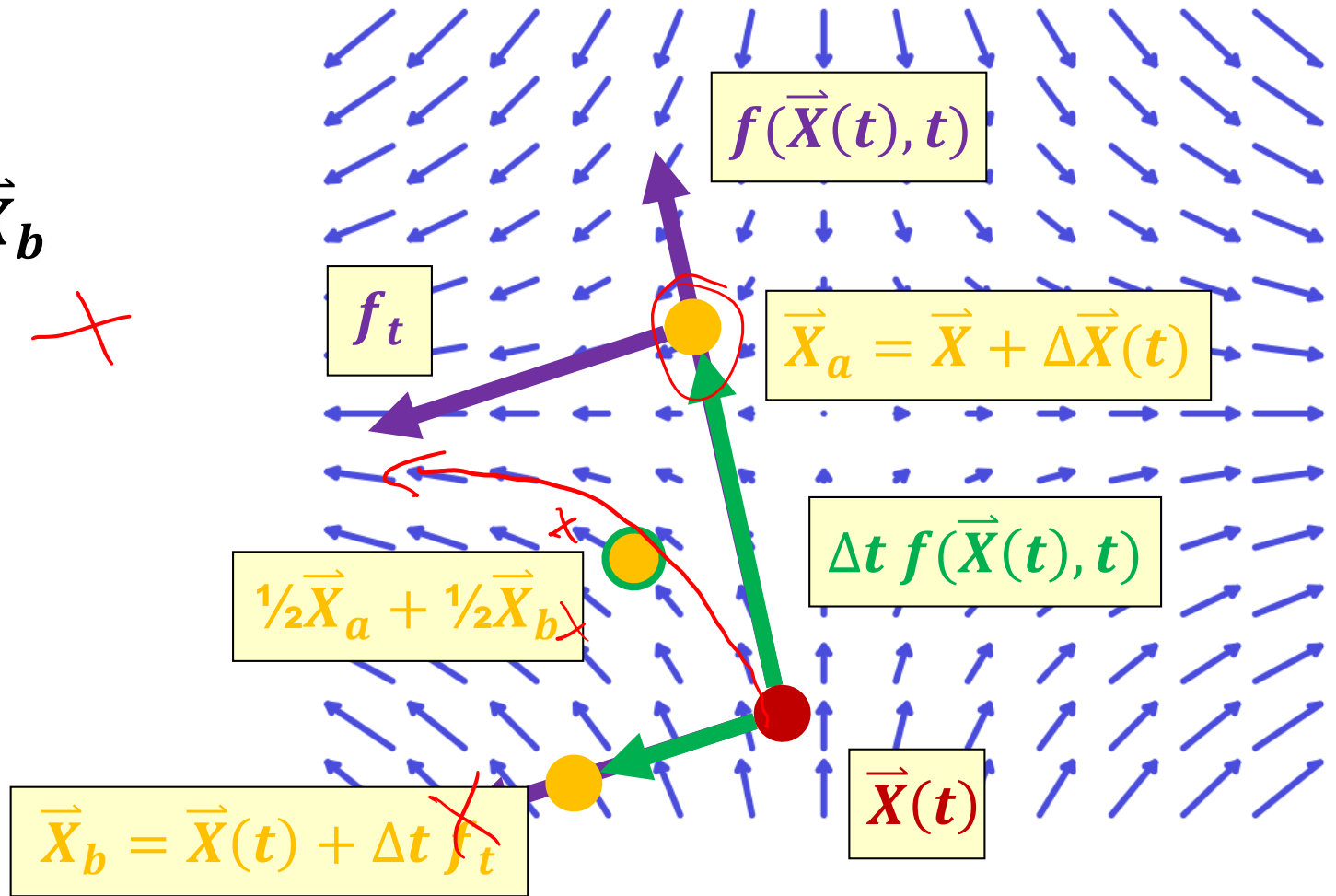
Midpoint Method

1. $\frac{1}{2}$ Euler step
2. evaluate \mathbf{f}_m at $\bar{\mathbf{X}}_m$
3. full step using \mathbf{f}_m



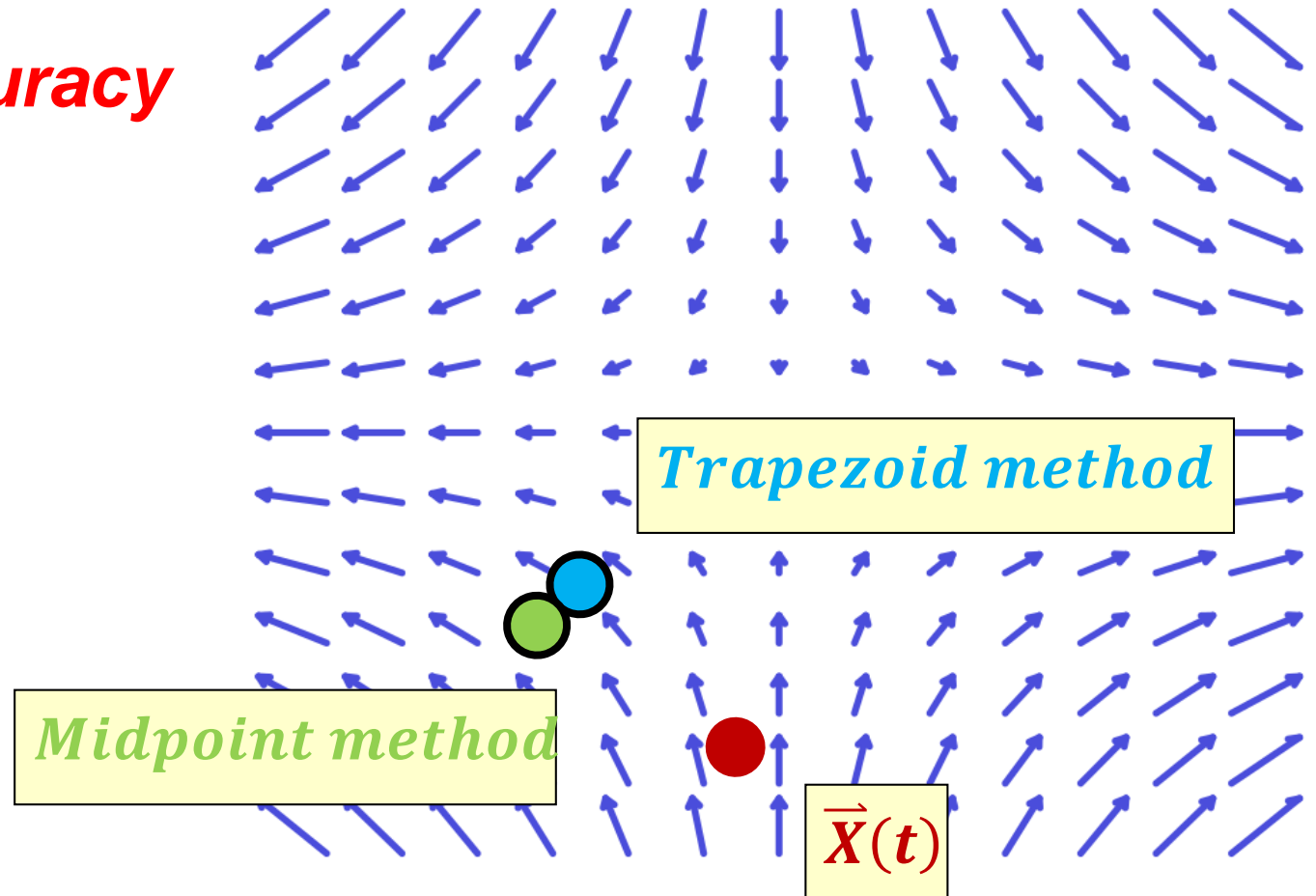
Trapezoid Method

1. full Euler step get \vec{X}_a
2. evaluate f_t at \vec{X}_a
3. full step using f_t get \vec{X}_b
4. average \vec{X}_a and \vec{X}_b

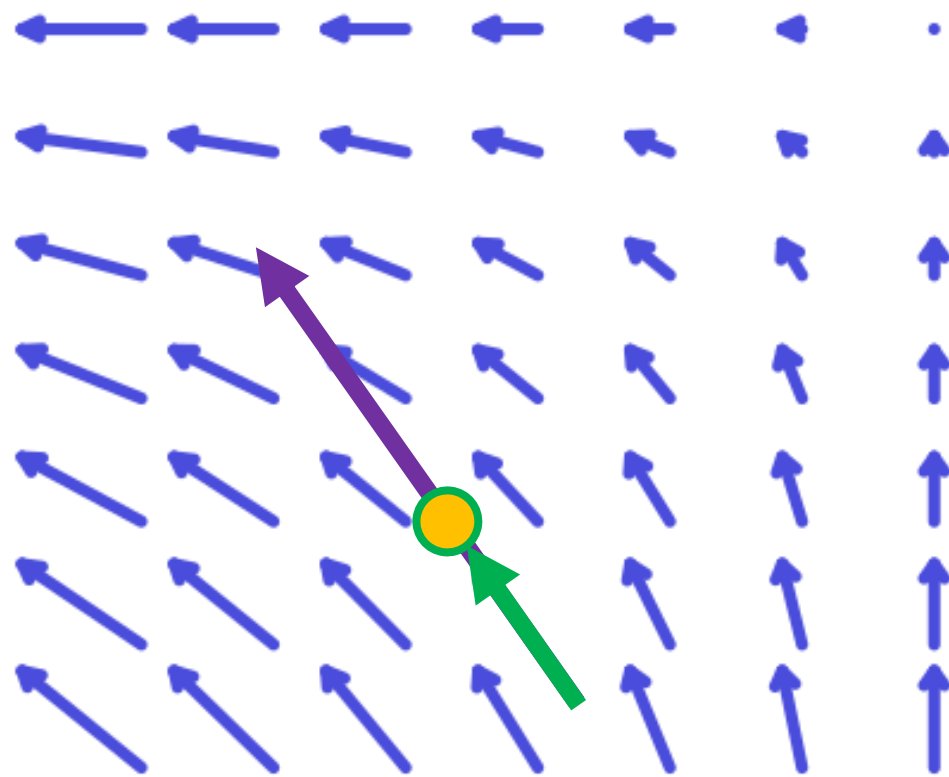


Midpoint & Trapezoid Method

- Not exactly the same
 - *But same order of accuracy*



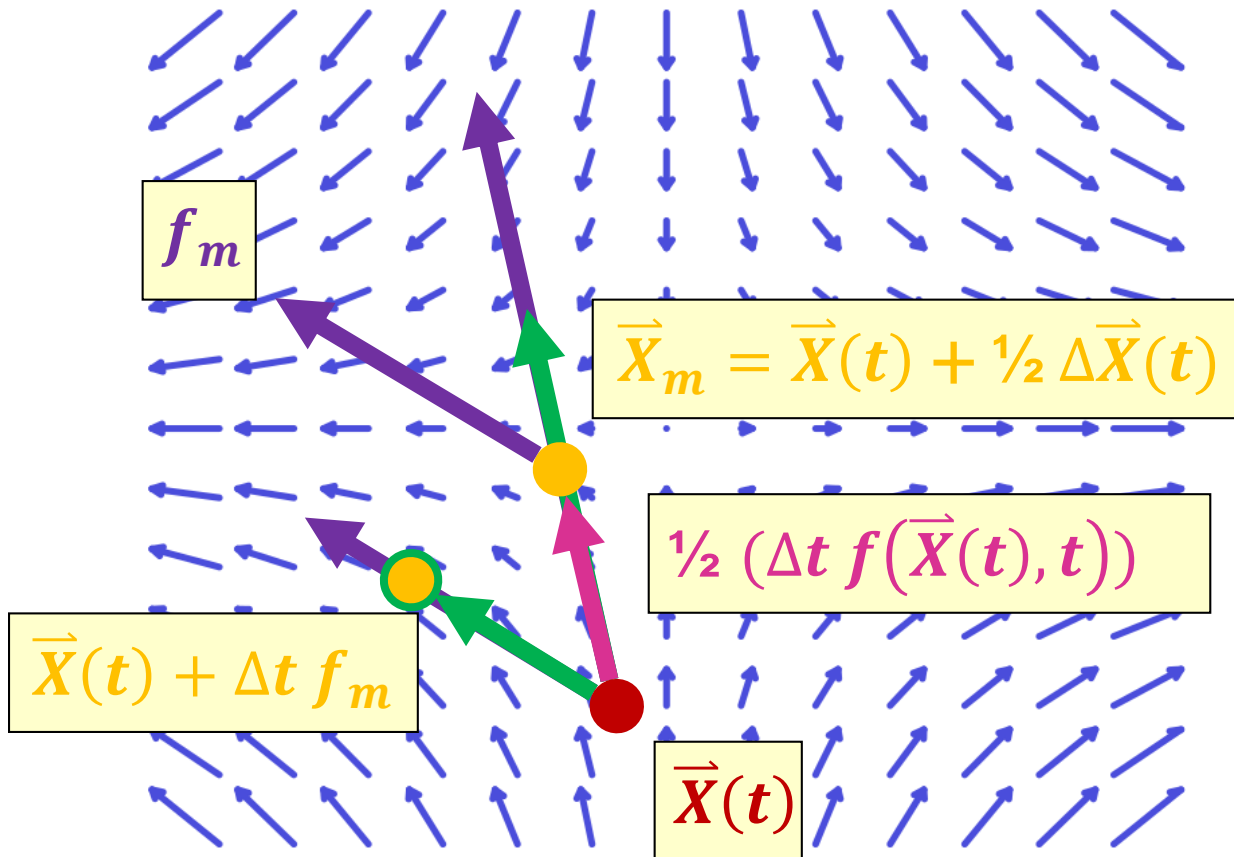
Explicit Euler: Code



```
void takeStep(ParticleSystem* ps, float h)
{
    velocities = ps->getStateVelocities()
    positions = ps->getStatePositions()
    forces = ps->getForces(positions, velocities)
    masses = ps->getMasses()
    accelerations = forces / masses
    newPositions = positions + h*velocities
    newVelocities = velocities + h*accelerations
    ps->setStatePositions(newPositions)
    ps->setStateVelocities(newVelocities)
}
```

Midpoint Method: Code

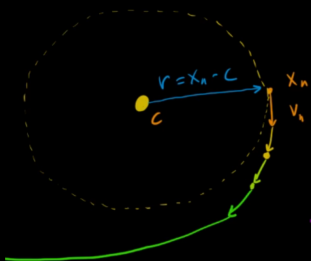
```
void takeStep(ParticleSystem* ps, float h)
{
    velocities = ps->getStateVelocities()
    positions = ps->getStatePositions()
    forces = ps->getForces(positions, velocities)
    masses = ps->getMasses()
    accelerations = forces / masses
    midPositions = positions + 0.5*h*velocities
    midVelocities = velocities + 0.5*h*accelerations
    midForces = ps->getForces(midPositions, midVelocities)
    midAccelerations = midForces / masses
    newPositions = positions + h*midVelocities
    newVelocities = velocities + h*midAccelerations
    ps->setStatePositions(newPositions)
    ps->setStateVelocities(newVelocities)
}
```



NUMERICAL INTEGRATION

$$h = \Delta t$$

$$f(x, t) = \cancel{g(t)} + v_0$$



$$x_{n+1} = x_n + h v_n$$

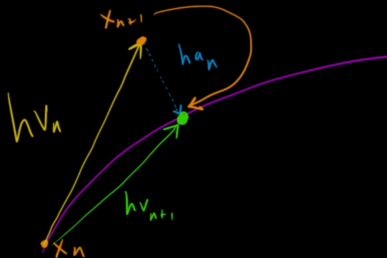
$$v_{n+1} = v_n + h a_n$$

$$a(x_n) = -G \frac{m_{sun}}{|r|^3} \vec{r}$$

SEMI-IMPLICIT EULER

$$v_{n+1} = v_n + h a_n$$

$$x_{n+1} = x_n + h v_{n+1}$$



Implicit (Backward) Euler:

- Use forces at destination

Solve system of equations

$$\frac{\partial}{\partial t} \begin{bmatrix} \vec{x} \\ \vec{v} \end{bmatrix} = \begin{bmatrix} \vec{v} \\ \Sigma \vec{F} / m \end{bmatrix}$$

$$\begin{aligned} x_{n+1} &= x_n + h v_{n+1} \\ v_{n+1} &= v_n + h \left(\frac{F_{n+1}}{m} \right) \end{aligned}$$

- Types of forces:

- **Gravity**

$$F = \begin{bmatrix} 0 \\ -mg \end{bmatrix}$$

- **Viscous damping**

$$F = -bv$$

- **Spring & dampers**

$$F = -kx - bv$$

Implicit (Backward) Euler:

- Use forces at destination + **derivative** at the **destination**

Solve system of equations

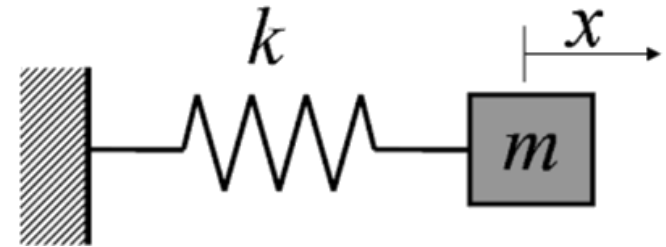
$$\frac{\partial}{\partial t} \begin{bmatrix} \vec{x} \\ \vec{v} \end{bmatrix} = \begin{bmatrix} \vec{v} \\ \Sigma \vec{F} / m \end{bmatrix}$$

$$\begin{aligned} x_{n+1} &= x_n + h v_{n+1} \\ v_{n+1} &= v_n + h \left(\frac{F_{n+1}}{m} \right) \end{aligned}$$

Key idea: use velocity estimated at next step instead of current !

Example: Spring Force

$$F = -kx$$



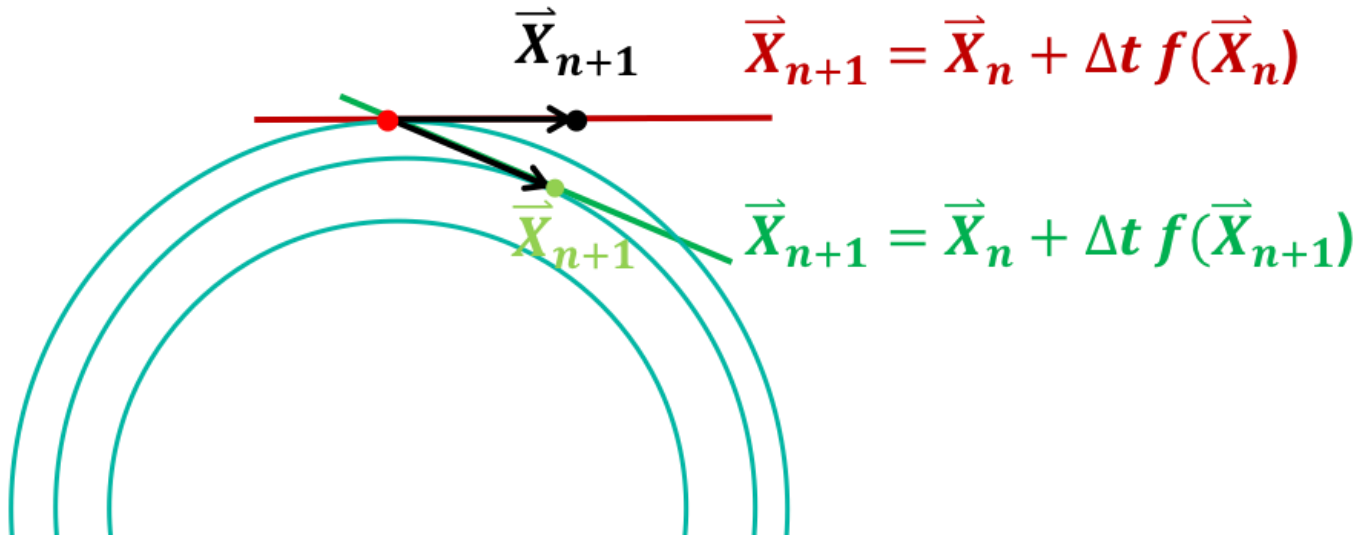
$$\begin{aligned} x_{n+1} &= x_n + h v_{n+1} \\ v_{n+1} &= v_n + h \left(\frac{-k x_{n+1}}{m} \right) \end{aligned}$$

Analytic or iterative solve?

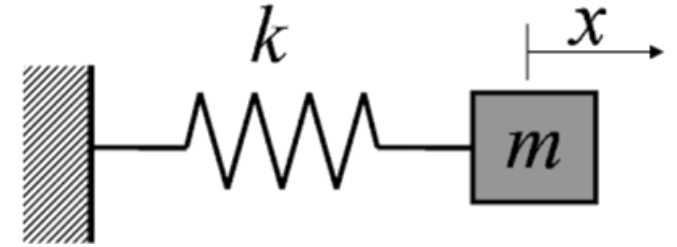
Rung-Kutta Order 4

<https://www.youtube.com/watch?v=hGCP6I2WisM&list=PLW3ZI3wyJwWOpdhYedID-yCB7WQoHf-My&index=110>

Forward vs Backward



Could one apply the Trapezoid Method?



Forward Euler

$$\begin{aligned}x_{n+1} &= x_n + h v_n \\v_{n+1} &= v_n + h \left(\frac{-k x_n}{m} \right)\end{aligned}$$

Backward Euler

$$\begin{aligned}x_{n+1} &= x_n + h v_{n+1} \\v_{n+1} &= v_n + h \left(\frac{-k x_{n+1}}{m} \right)\end{aligned}$$

Particles:

Newtonian Physics as First-Order DE

- Motion of **many** particles?

$$\frac{\partial}{\partial t} \begin{bmatrix} \overrightarrow{x_1} \\ \overrightarrow{v_1} \\ \overrightarrow{x_2} \\ \overrightarrow{v_2} \\ \vdots \\ \overrightarrow{x_n} \\ \overrightarrow{v_n} \end{bmatrix} = \begin{bmatrix} \overrightarrow{v_1} \\ \overrightarrow{F_1}/m_1 \\ \overrightarrow{v_2} \\ \overrightarrow{F_2}/m_2 \\ \vdots \\ \overrightarrow{v_n} \\ \overrightarrow{F_n}/m_n \end{bmatrix}$$

- Interaction of particles?

Multiple-particle collision

- ***naïve implementation is likely unstable***
 - *Objects pushing inside each other*
- ***Further reading:***
 - <https://box2d.org/publications/>
 - *In particular*
https://box2d.org/files/ErinCatto_ModelingAndSolvingConstraints_GDC2009.pdf

Simulation Basics

Simulation loop...

- 1. Equations of Motion***
- 2. Numerical integration***
- 3. Collision detection***
- 4. Collision resolution***

Collisions

- Collision **detection**
 - *Broad phase: AABBs, bounding spheres*
 - *Narrow phase: detailed checks*
- Collision **response**
 - *Collision impulses*
 - *Constraint forces: resting, sliding, hinges,*

Basic Particle Simulation (first try)

Forces only $\vec{F} = m\vec{a}$

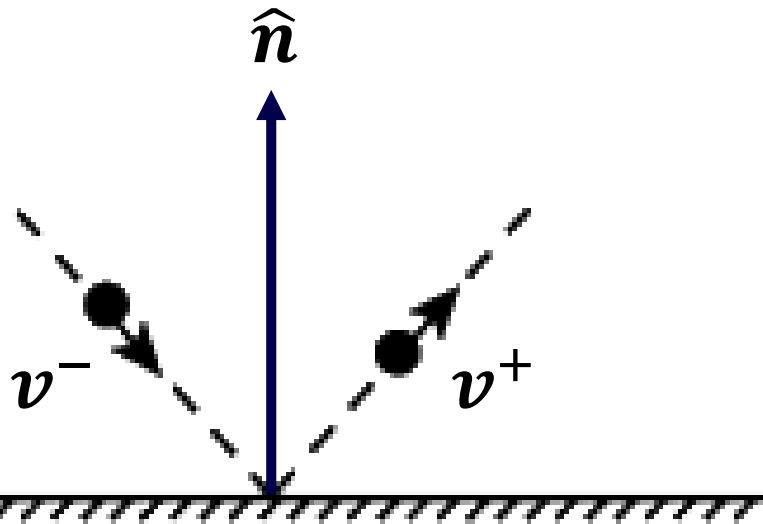
$$\begin{aligned}d_t &= t_{i+1} - t_i \\ \vec{v}_{i+1} &= \vec{v}(t_i) + (\vec{F}(t_i)/m)d_t \\ \vec{p}_{i+1} &= \vec{p}(t_i) + \vec{v}(t_{i+1})d_t\end{aligned}$$



Particle-Plane Collisions

- Apply an **'impulse'** of magnitude j
 - Inversely proportional to mass of particle
- In direction of normal

Impulse in physics: Integral of F over time
In games: an instantaneous step change (not physically possible), i.e., the force applied over one time step of the simulation



$$j = (1 + \epsilon)(v^- \cdot \hat{n})m$$

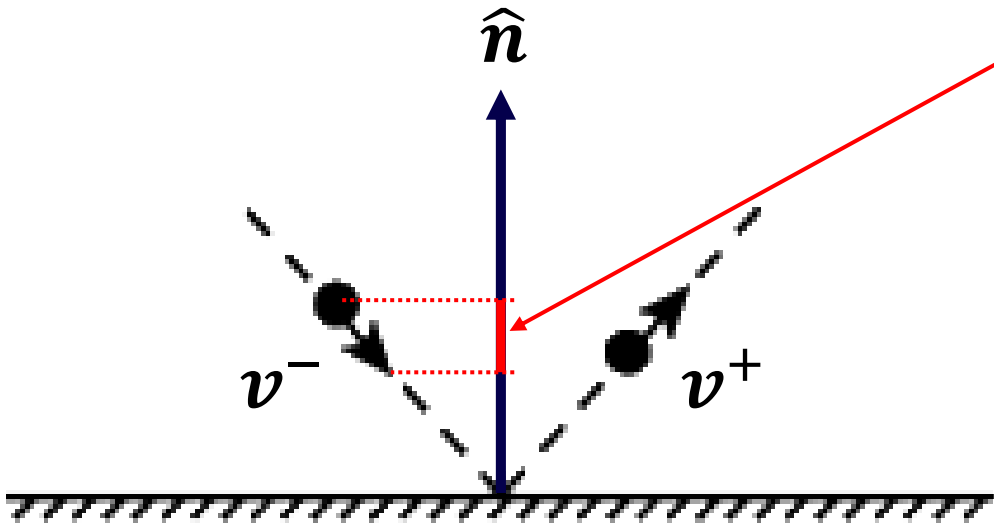
$$\vec{j} = j \hat{n}$$

What is the effect of ϵ ?

$$v^+ = \frac{\vec{j}}{m} + v^-$$

Recap: Particle-Plane Collisions (in terms of vel.)

- ***Change in direction of normal***



Velocity along normal
(v projected on normal
by the dot product)

Frictionless

$$\Delta v = 2(v^- \cdot \hat{n})\hat{n}$$

Apply change
along normal
(magnitude
times direction)

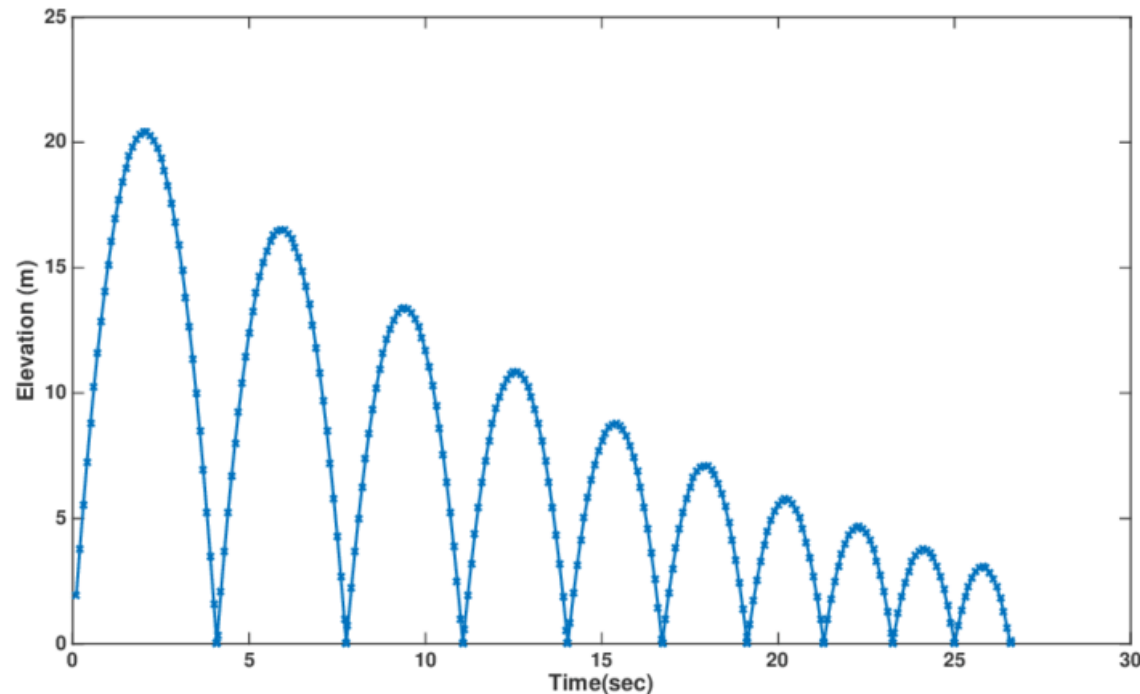
$$v^+ = v^- + \Delta v$$

Loss of energy

$$\Delta v = (1 + \epsilon)(v^- \cdot \hat{n})\hat{n}$$

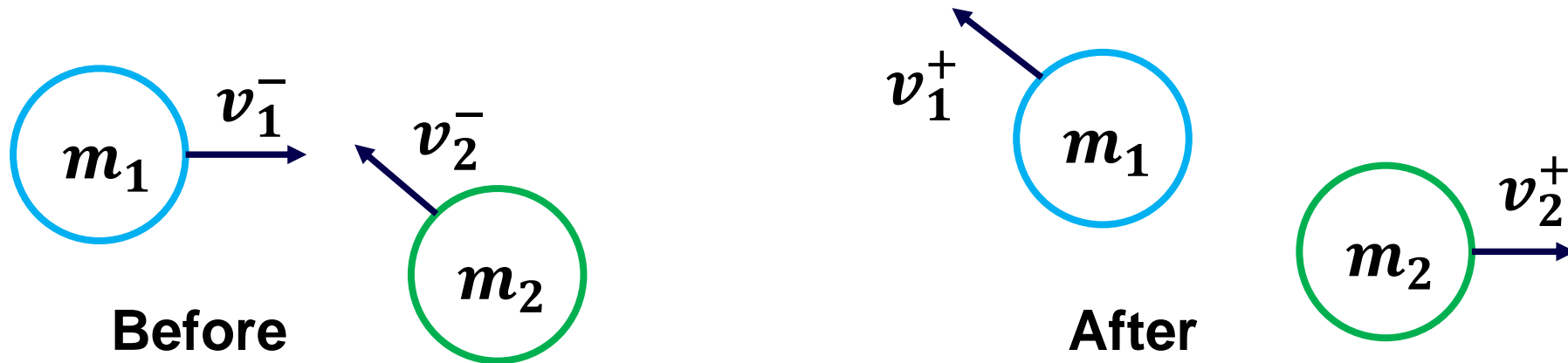
Why use 'Impulse'?

- *Integrates with the physics solver*
- *How to integrate damping?*



Particle-Particle Collisions (radius=0)

- Particle-particle **frictionless elastic impulse response**



- Momentum is **preserved**

$$m_1 v_1^- + m_2 v_2^- = m_1 v_1^+ + m_2 v_2^+$$

- Kinetic energy is **preserved**

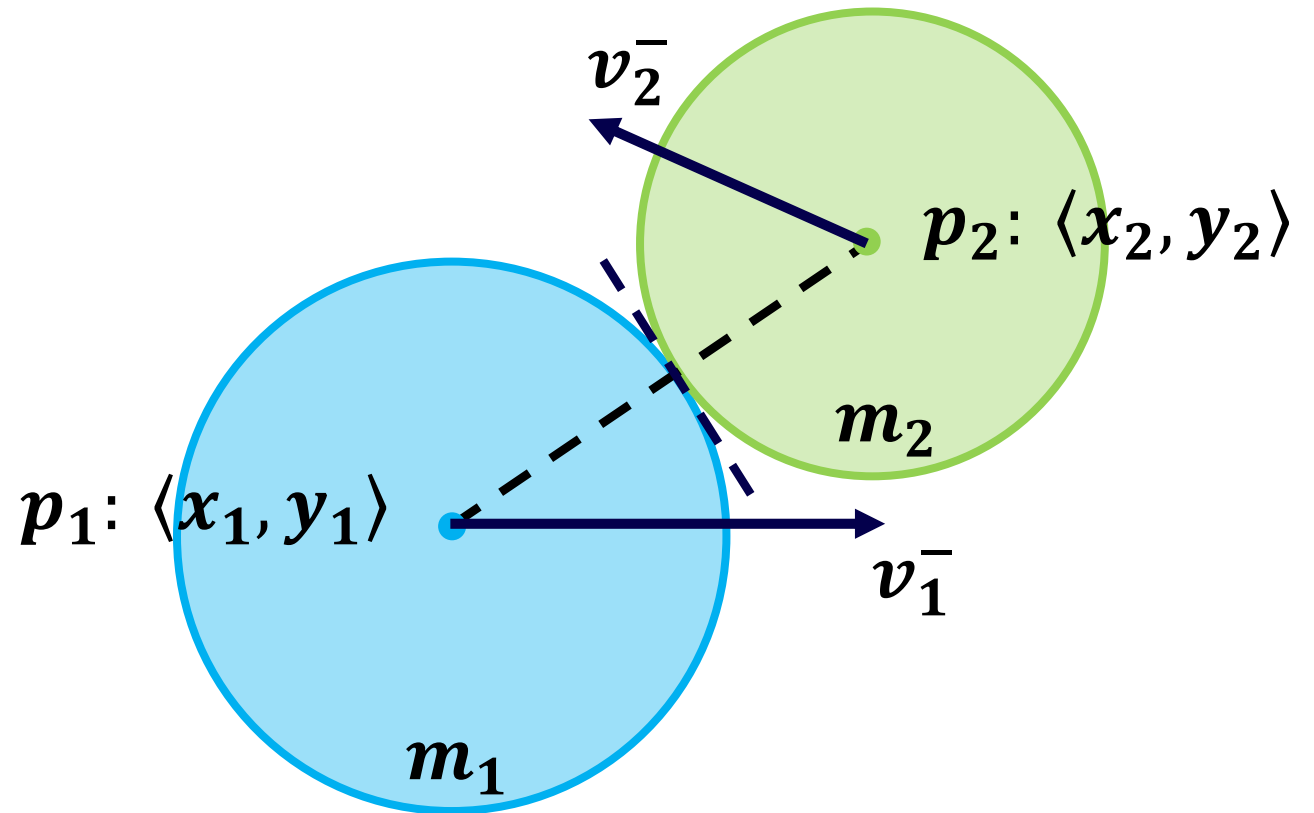
$$\frac{1}{2} m_1 v_1^{-2} + \frac{1}{2} m_2 v_2^{-2} = \frac{1}{2} m_1 v_1^{+2} + \frac{1}{2} m_2 v_2^{+2}$$

- Velocity is **preserved in tangential direction**

$$t \cdot v_1^- = t \cdot v_1^+, \quad t \cdot v_2^- = t \cdot v_2^+$$

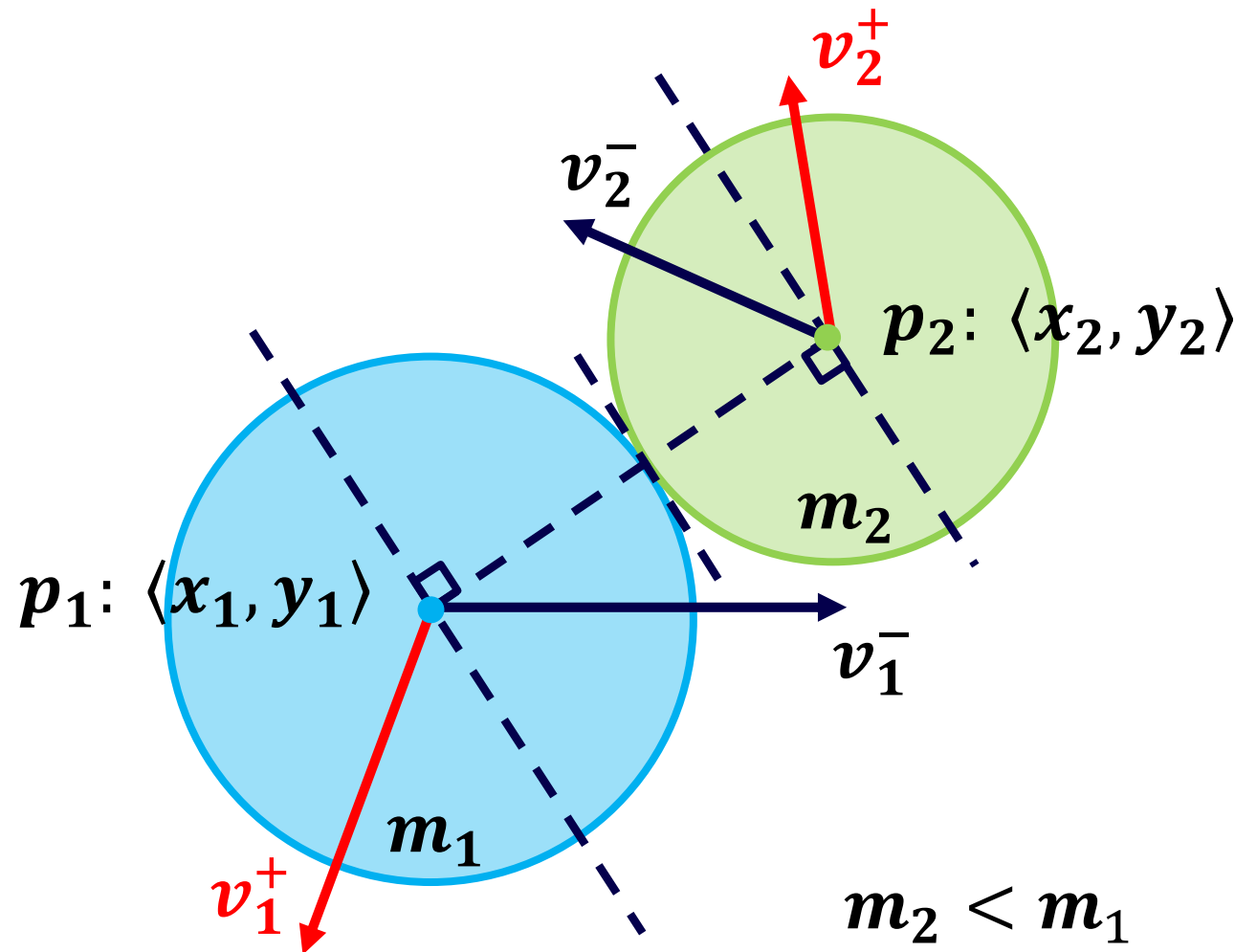
Particle-Particle Collisions (radius >0)

- What we know...
 - *Particle centers*
 - *Initial velocities*
 - *Particle Masses*
- What we can calculate...
 - *Contact normal*
 - *Contact tangent*



Particle-Particle Collisions (radius >0)

- Impulse **direction** reflected across **tangent**
- Impulse **magnitude** proportional to **mass of other particle**



Particle-Particle Collisions (radius >0)

- **More formally...**

$$\mathbf{v}_1^+ = \mathbf{v}_1^- - \frac{2m_2}{m_1 + m_2} \frac{\langle \mathbf{v}_1^- - \mathbf{v}_2^- \rangle \cdot \langle \mathbf{p}_1 - \mathbf{p}_2 \rangle}{\|\mathbf{p}_1 - \mathbf{p}_2\|^2} \langle \mathbf{p}_1 - \mathbf{p}_2 \rangle$$

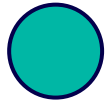
$$\mathbf{v}_2^+ = \mathbf{v}_2^- - \frac{2m_1}{m_1 + m_2} \frac{\langle \mathbf{v}_2^- - \mathbf{v}_1^- \rangle \cdot \langle \mathbf{p}_2 - \mathbf{p}_1 \rangle}{\|\mathbf{p}_2 - \mathbf{p}_1\|^2} \langle \mathbf{p}_2 - \mathbf{p}_1 \rangle$$

- This is in terms of velocity, what would the corresponding impulse be?

Rigid Body Dynamics

(rotational motion of objects?)

- From particles to rigid bodies...



Particle

$$state = \begin{cases} \vec{x} \text{ position} \\ \vec{v} \text{ velocity} \end{cases}$$

\mathbb{R}^4 in 2D

\mathbb{R}^6 in 3D



Rigid body

$$state = \begin{cases} \vec{x} \text{ position} \\ \vec{v} \text{ velocity} \\ R \text{ rotation matrix } 3 \times 3 \\ \vec{\omega} \text{ angular velocity} \end{cases}$$

\mathbb{R}^{12} in 3D

DEMOS

1. Code on Files for particles and integration methods.

Objectives: take a look at the code and forces field, comparison between methods

2. Unreal Particles level: ContentExamples, level Particles_intro.

Objectives: understand how they are simulated, efforts needed for programmers, curves, etc.

3. Unreal Physics tutorial

Objectives:

- Apply impulse on drop objects, check PlayerCharacter blueprint
- radial force
- understanding Skeletal Mesh bones and physical asset
- constraints
- thruster (X axis neg force constant applied)
- angular motors on constraints
- show angular, linear limits for constraints