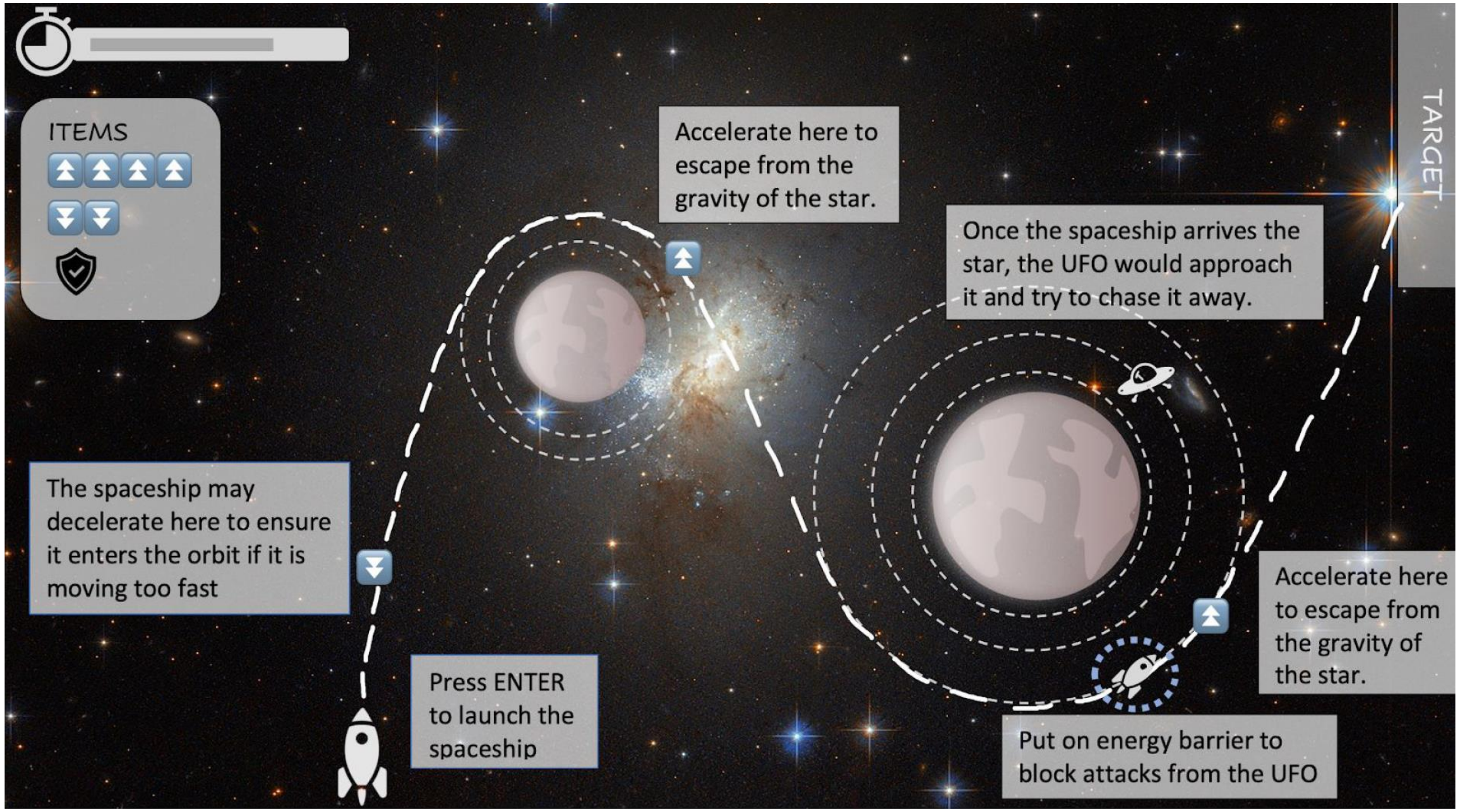
Helge Rhodin

CPSC 427
Video Game Programming

Physical Simulation



Overview

1. ***Recap AI & Debugging***

2. ***Equation of Motion***
   - Ordinary Differentiable Equations (ODE)
   - Solving ODEs

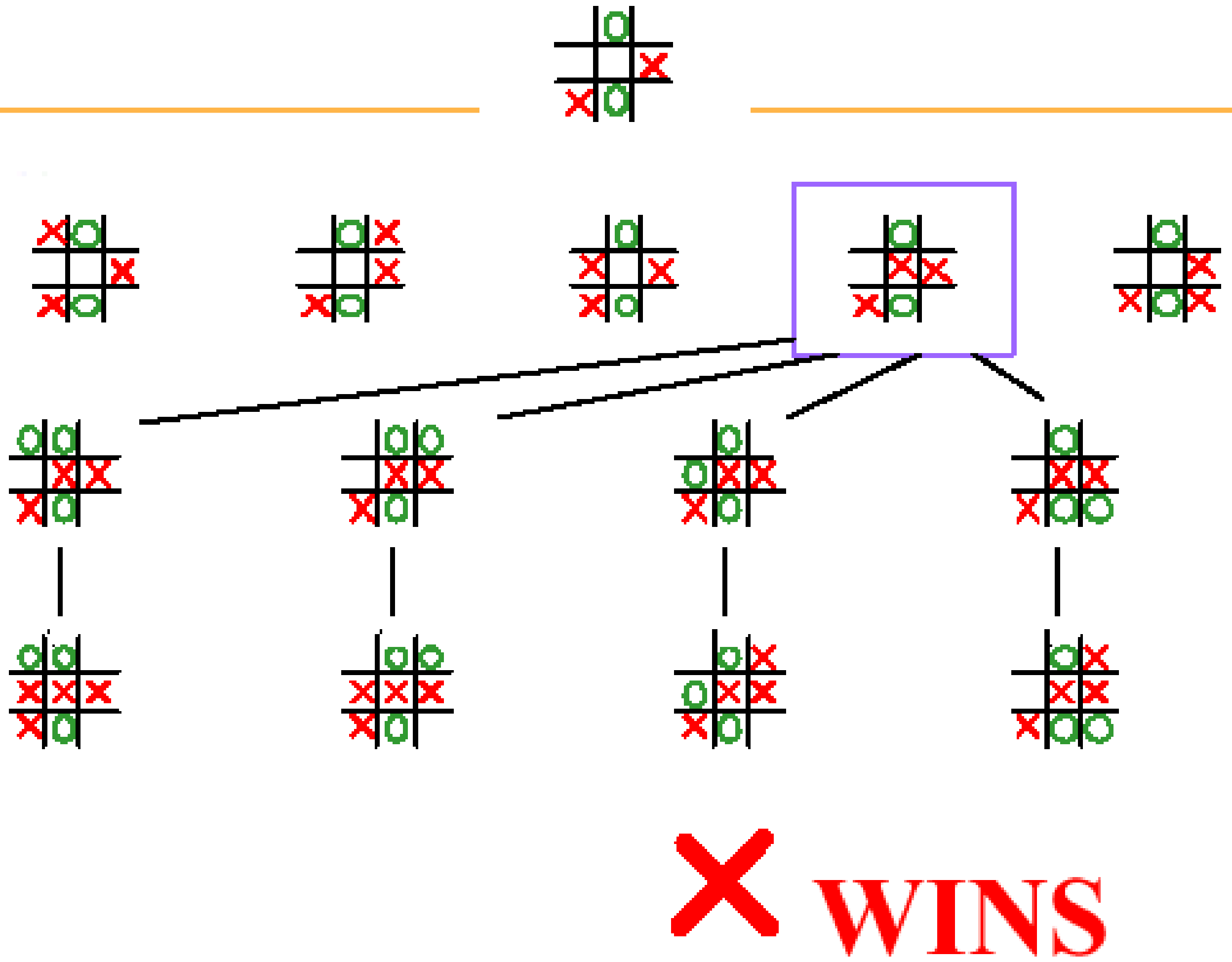
3. ***Collision and Reaction Forces***

Recap: AI

Two-player games

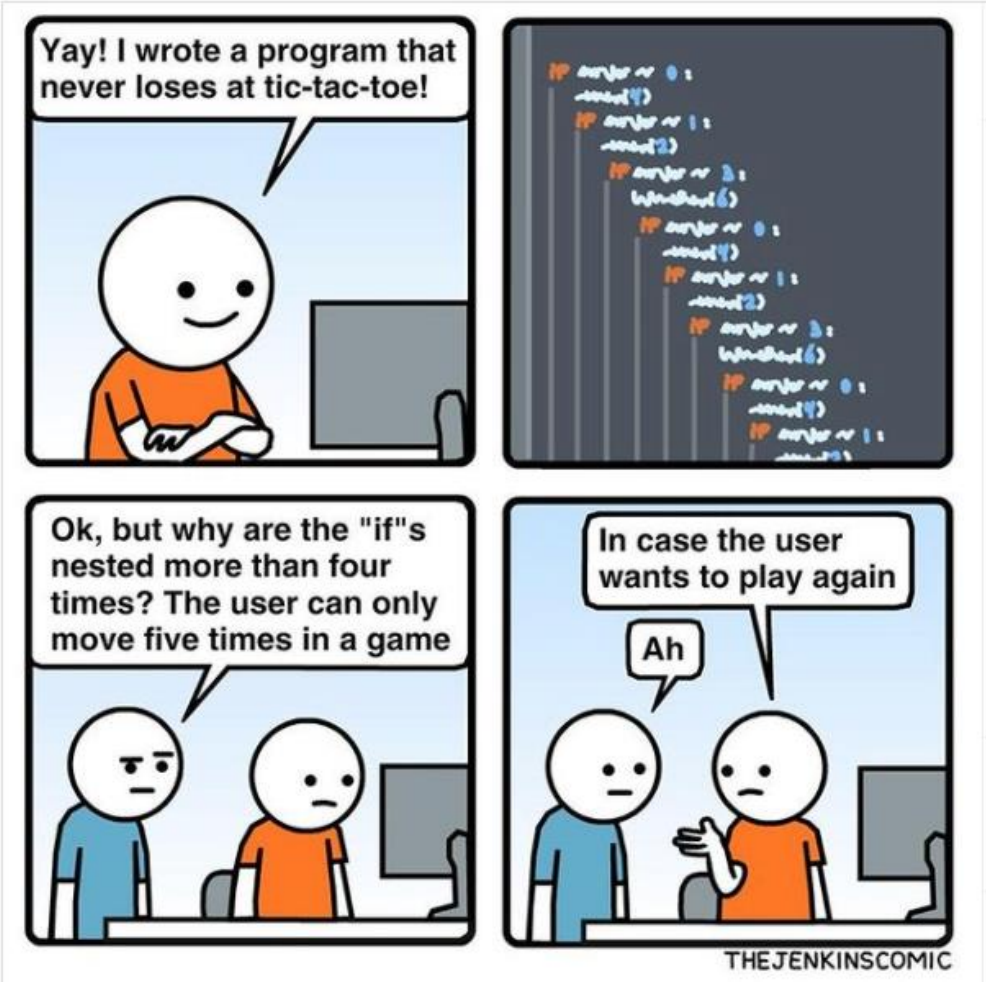
www.npr.org

Our options

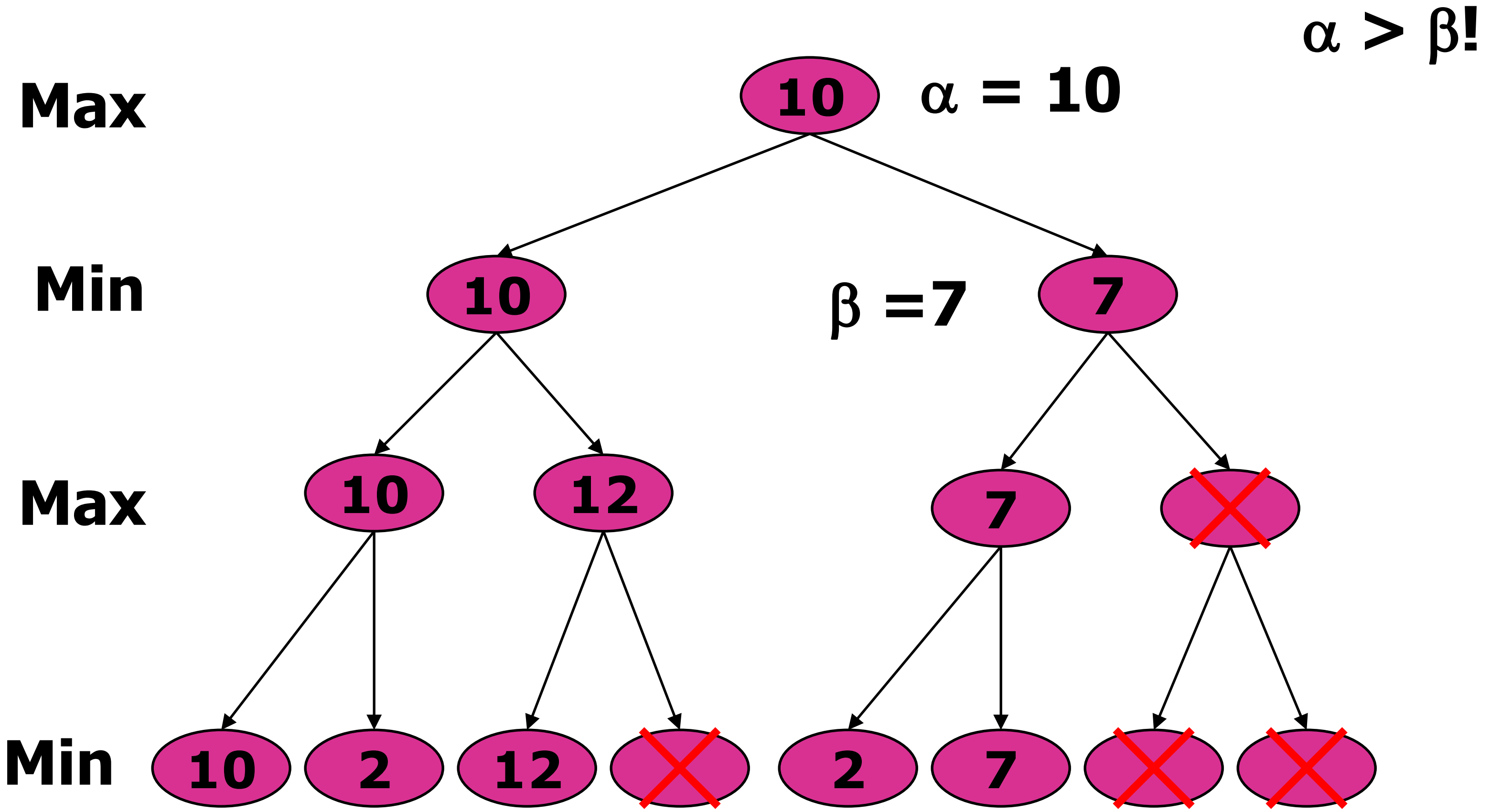


We have a win for any move they make.
Original position in purple is an X win.

Implementation?



Alpha Beta Tree



Debugging

- ***There will be bugs…***

- ***Strategies for Fixing?***
  - Anticipate
  - Reproduce
    - *Things get terribly difficult if randomness is involved!*
  - Localize
  - Use proper debugging tools

Logistics: New TA-Team assignment for M2

Tim: 1, 6, 8

Grace: 4, 5, 7

Dave: 3, 11, 12

Andrew: 2, 9, 10

Logistics: Team Project Presentation

- *Quick summary of game idea*

- *Showcase early results*

- *What was easy?*
- *What was more difficult than imagined?*

- *We will have this on the Thursday after every milestone*
  This Thursday 5 pm!
  4 minutes per team

Logistics: Cross-play

- ***Test other team's games***
- ***Give feedback***
- ***Have fun***

- ***After M2 and subsequent milestones***
- ***Trial run on Thursday, ~6pm (after team presentations)***

Logistics: Guest lectures

- ***1h lecture by a domain expert***
- ***Every Tuesday 5-6 pm during March***
- ***Attendance mandatory (counts to course participation)***

Optional one:

- ***Raytracing & RTX***
- vote for time (morning time slot) on piazza

Logistics: Exam slot?

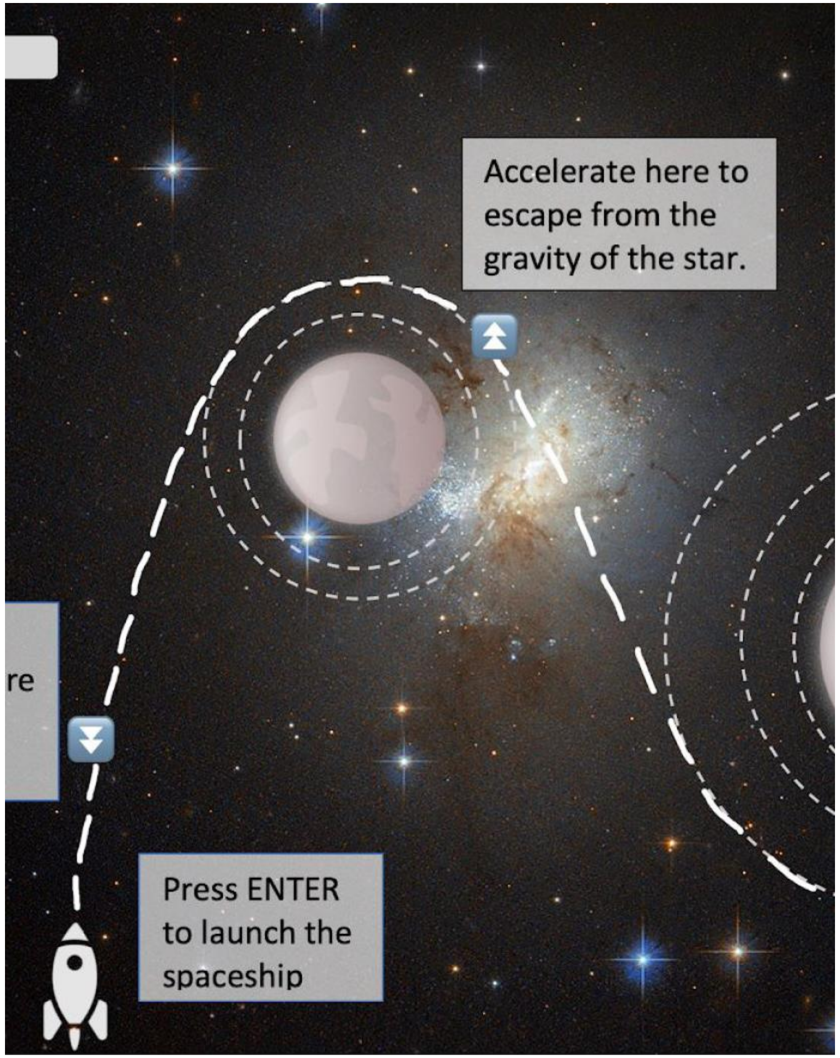
- *Final cross-play session*
- *Industry jury*
- *Awards*

- *19th, 7pm, Attendance mandatory*

Physics

Learning goals:

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



Simulation Basics

Simulation loop…

1. ***Equations of Motion***
   - sum forces & torques
   - solve for accelerations: $\vec{\boldsymbol{F}} = \boldsymbol{ma}$
2. ***Numerical integration***
   - update positions, velocities
3. ***Collision detection***
4. ***Collision resolution***

Basic Particle Simulation (first try)

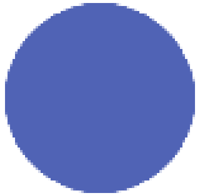
Forces only $\vec{F} = ma$

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

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

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

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



Basic Particle Forces

- **Gravity**

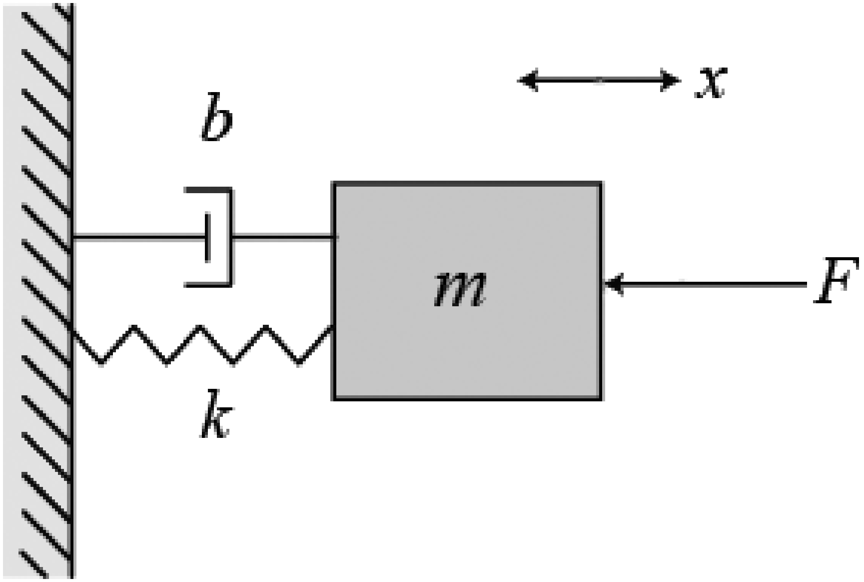
$$F = \begin{bmatrix} 0 \\ -mg \end{bmatrix}$$

- **Viscous damping**

$$F = -bv$$

- **Spring & dampers**

$$F = -kx - bv$$



Gravity direction?

Assuming a flat earth:

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

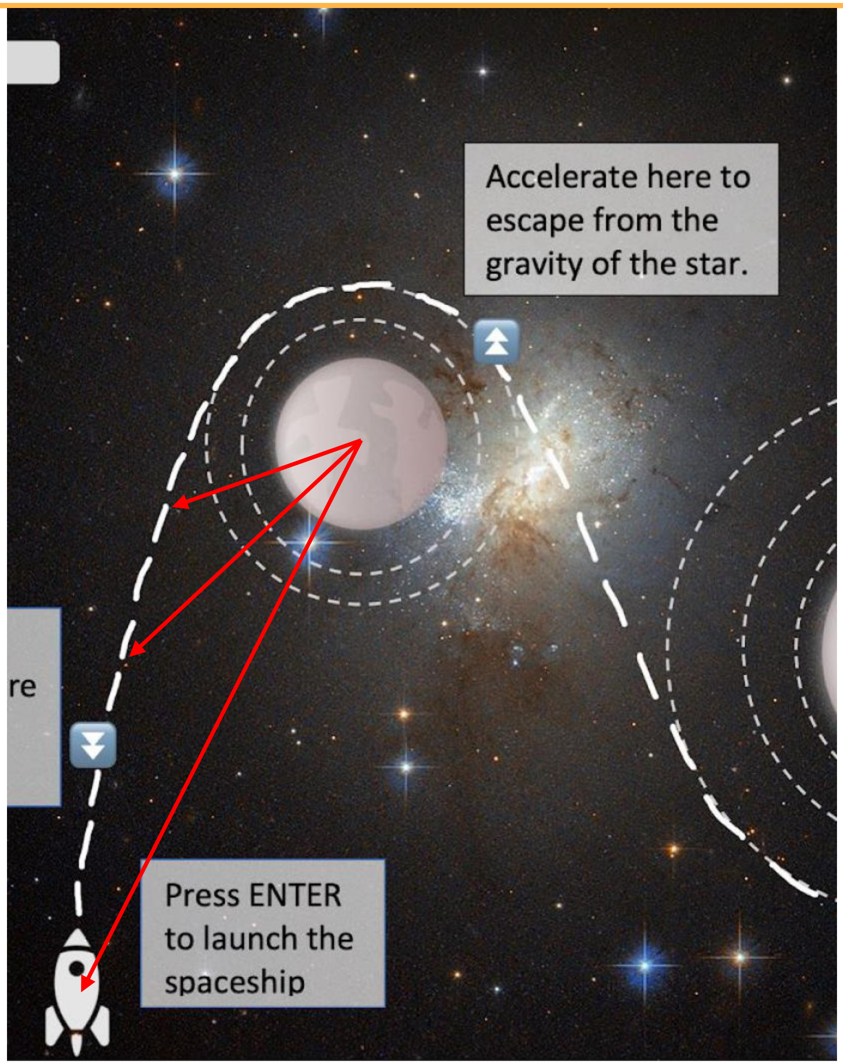
Assuming a spherical earth:

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

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):

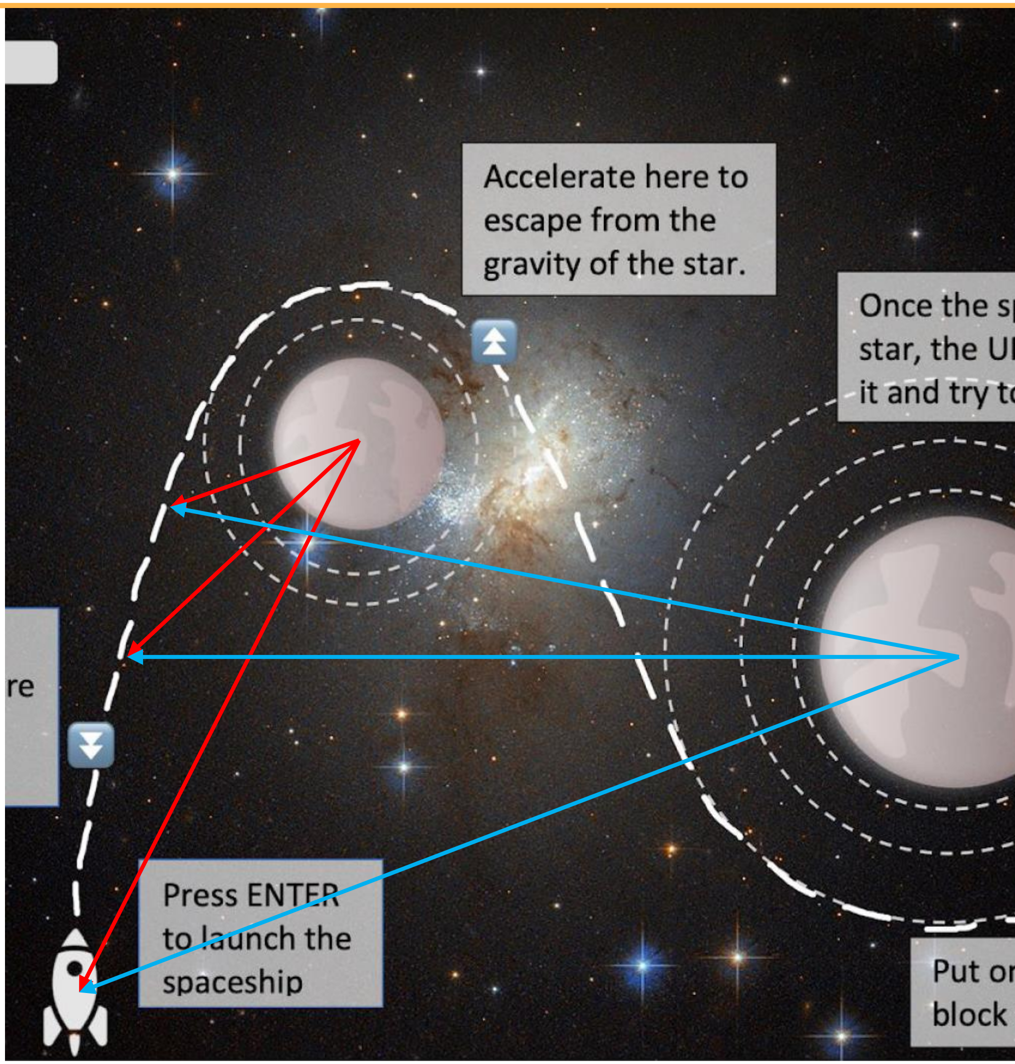
$$F = -mg_1 \begin{bmatrix} a_1 \\ b_1 \end{bmatrix} - mg_2 \begin{bmatrix} a_2 \\ b_2 \end{bmatrix}$$

- ***This holds for all types of forces!***

- ***Notation you might see:***

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

$$\vec{F} = F$$



Newtonian Physics as First-Order ODE

- **Motion of one particle**

Second-order ODE

acceleration

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

First-order ODE

velocity

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

- **General form**
  - ***Higher-order ODE:***

a function of t, f, and it's derivatives

$$f^{(k)}(t) = G[t, f(t), f'(t), f''(t), \cdots, f^{(k-1)}]$$

  - ***Equivalent first-order ODE:***

$$\frac{\partial}{\partial t}\begin{pmatrix} f_0(t) \\ f_1(t) \\ \vdots \\ f_{k-1}(t) \end{pmatrix} = \begin{pmatrix} f_0(t) \\ f_1(t) \\ \vdots \\ G[t, f_0(t), f_1(t), \cdots, f_{(k-1)}(t)] \end{pmatrix}$$

Higher-order ODEs can be turned into a first-order ODE with additional variables and equations!

Context: ODE vs. PDE

A differential equation is an equation that relates one or more functions and their derivatives.

An ordinary differential equation (ODE) is a differential equation containing one or more functions of one variable and the derivatives of those functions.

Equations coupling together derivatives of functions in more than one variable are known as partial differential equations (PDEs)

Basic Particle Simulation: Small Problem…

Forces only $\vec{F} = ma$

$$d_t = t_{i+1} - t_i$$
$$\vec{v}_{i+1} = \vec{v}(t_i) + (\vec{f}(t_i)/m)d_t$$
$$\vec{p}_{i+1} = \vec{p}(t_i) + \vec{v}(t_{i+1})d_t$$

Equations of motion describe state (equilibrium)

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

Ordinary Differential Equations

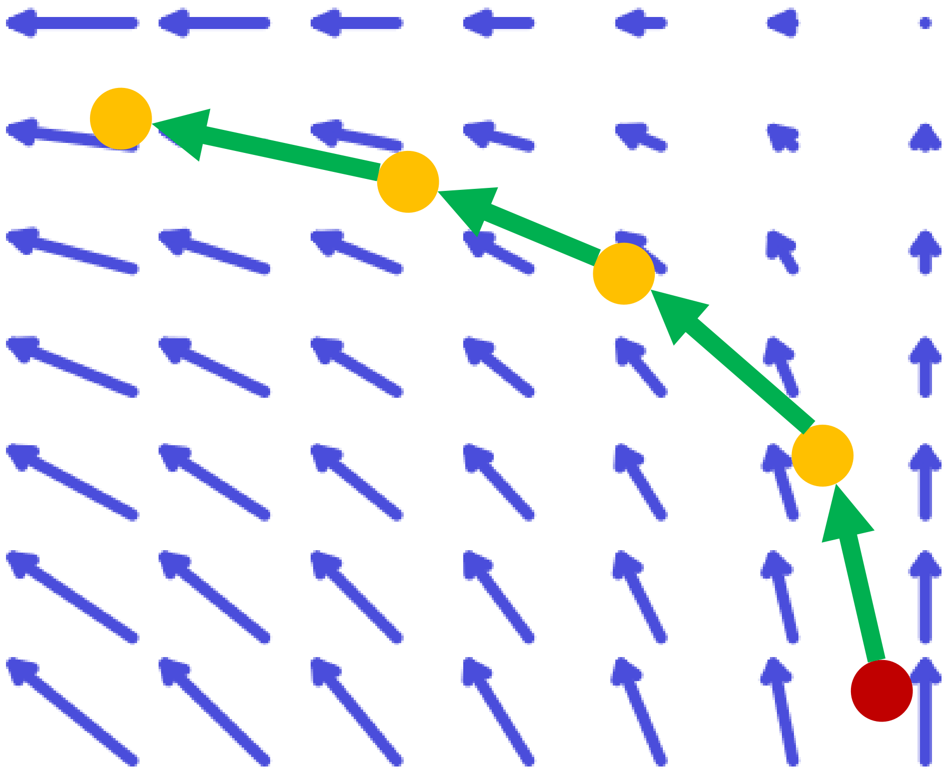
$$\frac{\partial}{\partial t}\vec{X}(t) = f(\vec{X}(t), t)$$

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

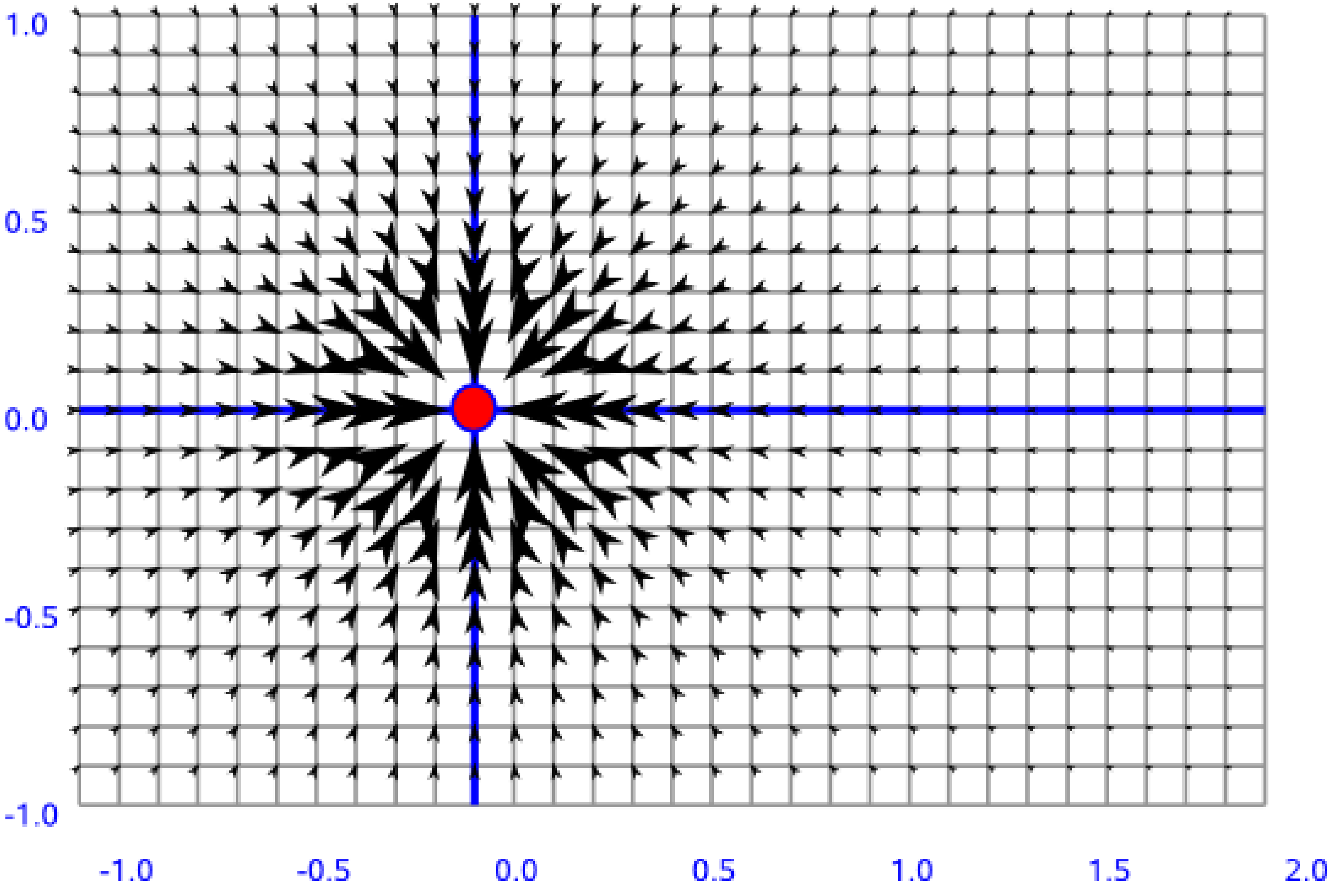
Compute $\vec{X}(t)$ for $\mathrm{t} > t_0$

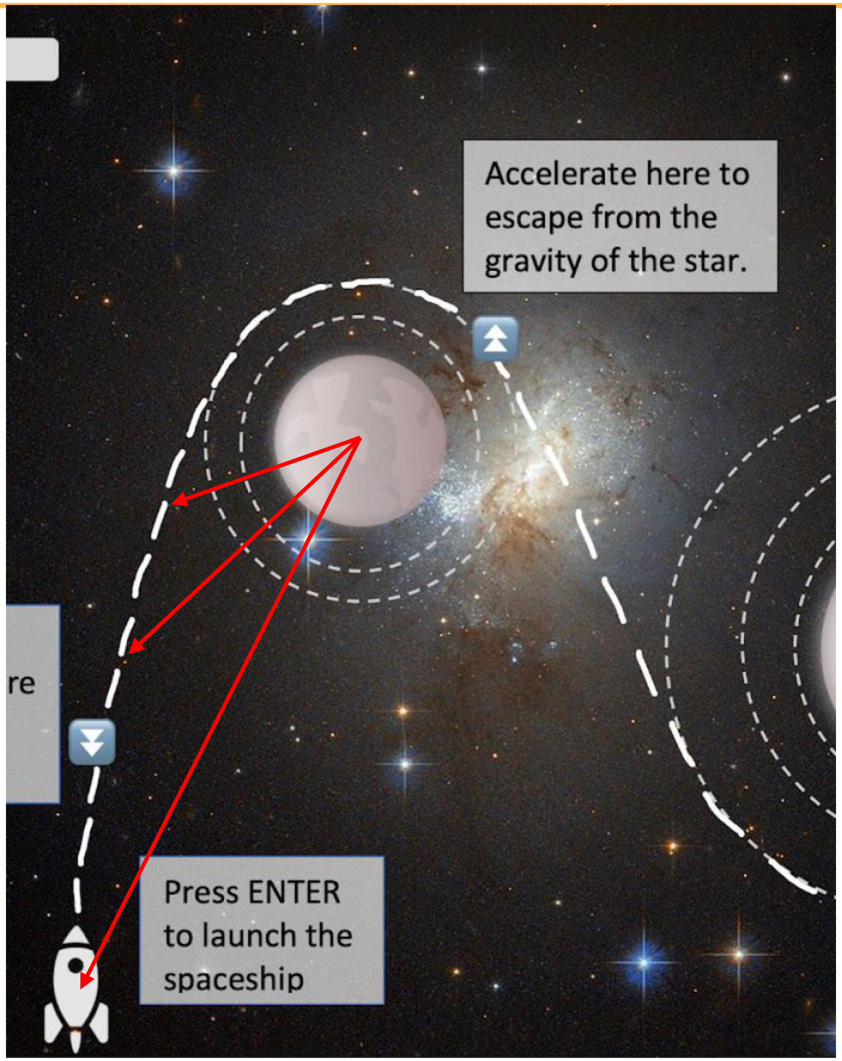
$$\Delta\vec{X}(t) = f(\vec{X}(t), t)\Delta t$$

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



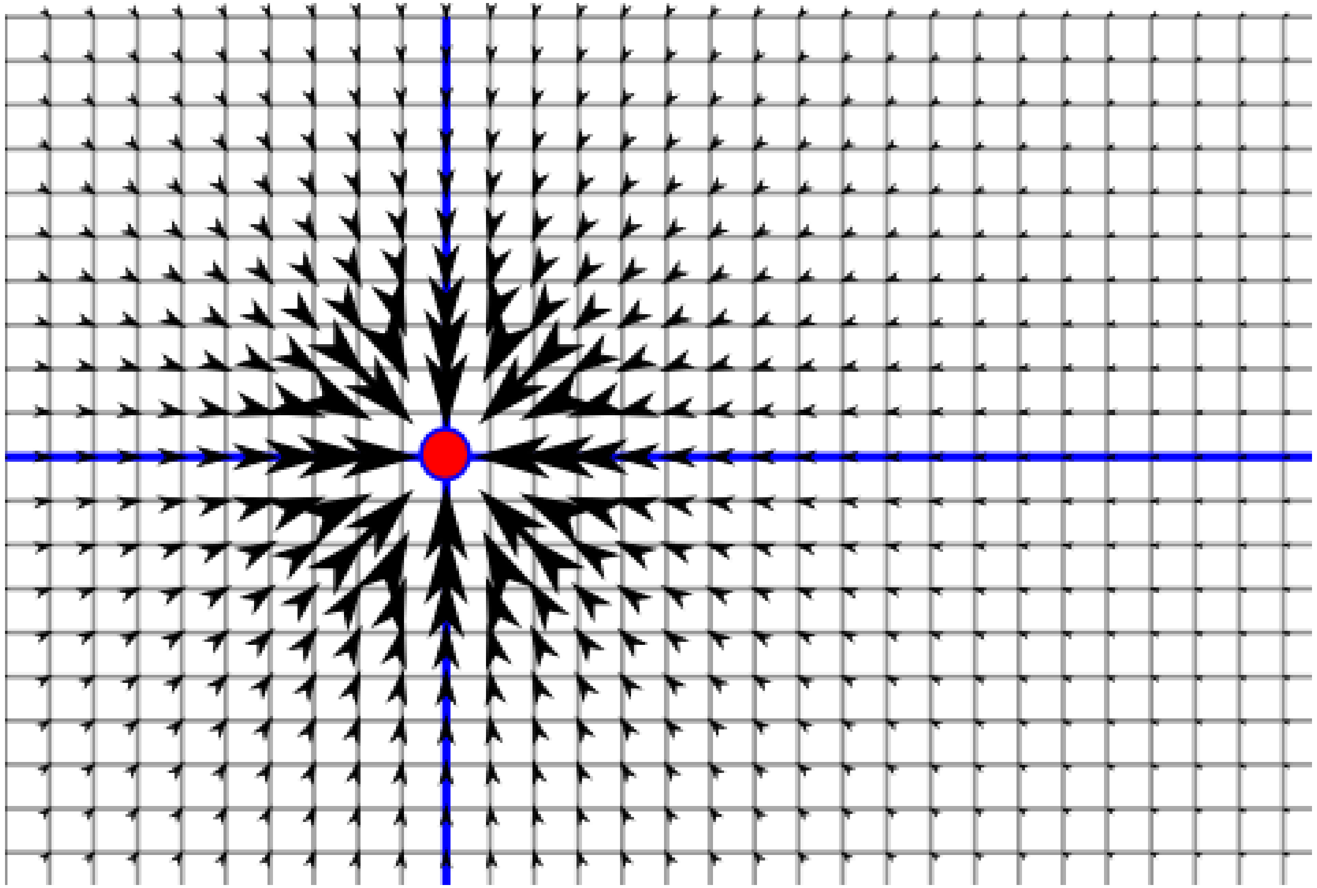
Gravitational field





https://www.euclideanspace.com/maths/geometry/space/fields/index.htm

Water Vortex (assignment?)



ODE Numerical Integration: Explicit (Forward) Euler

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

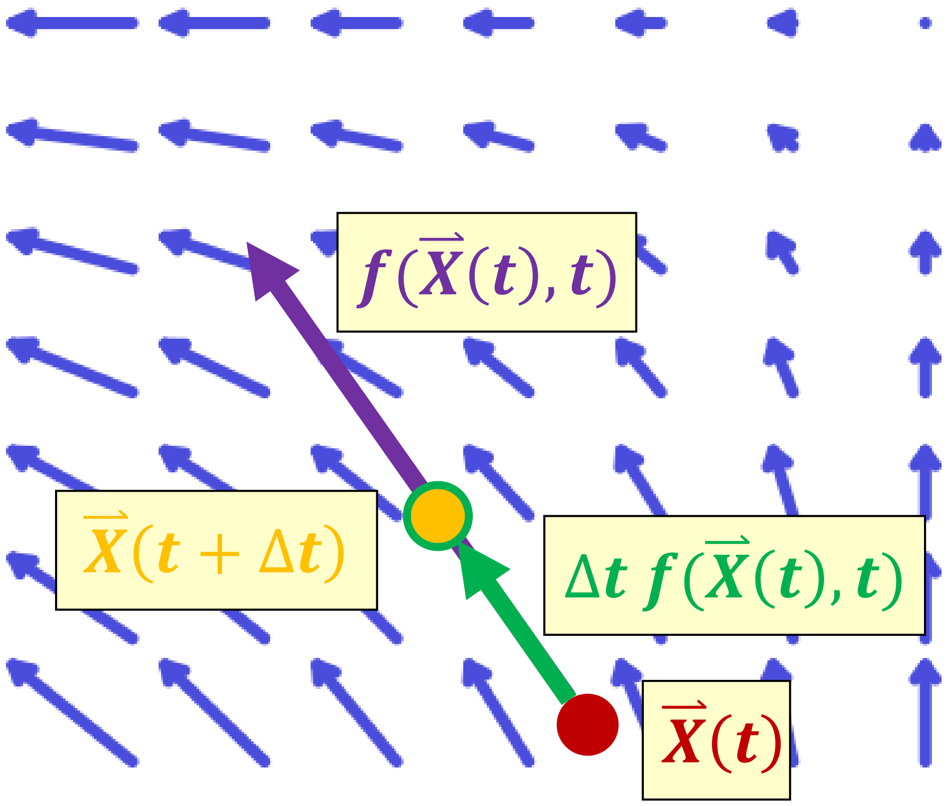
Given that $\vec{X}_0 = \vec{X}(t_0)$

Compute $\vec{X}(t)$ **for** $\mathrm{t} > \mathrm{t}_0$

$$\Delta t = \mathrm{t}_i - \mathrm{t}_{i-1}$$

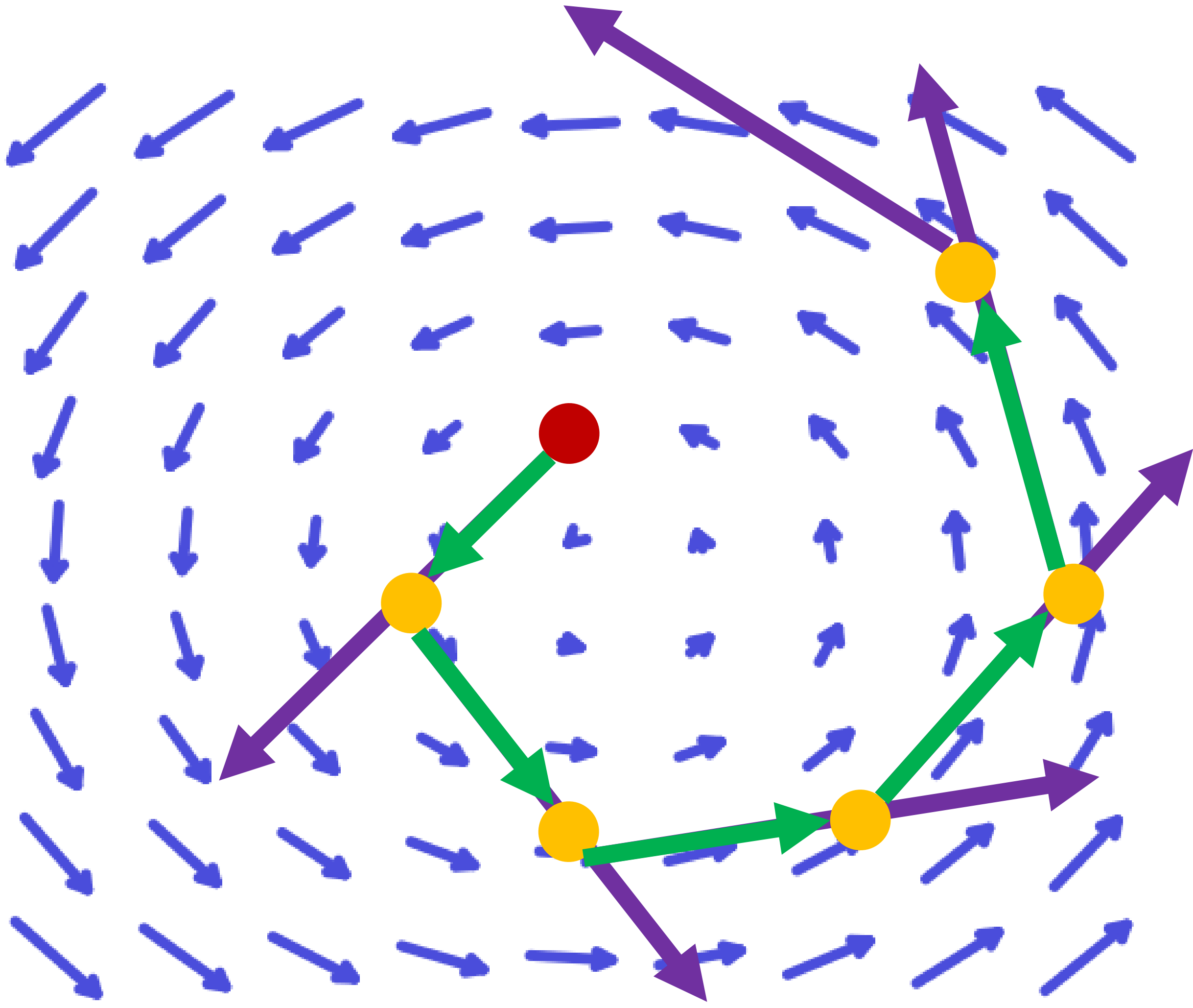
$$\Delta\vec{X}(\mathrm{t}_{i-1}) = \Delta t\, f(\vec{X}(\mathrm{t}_{i-1}), \mathrm{t}_{i-1})$$

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



Explicit Euler Problems

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

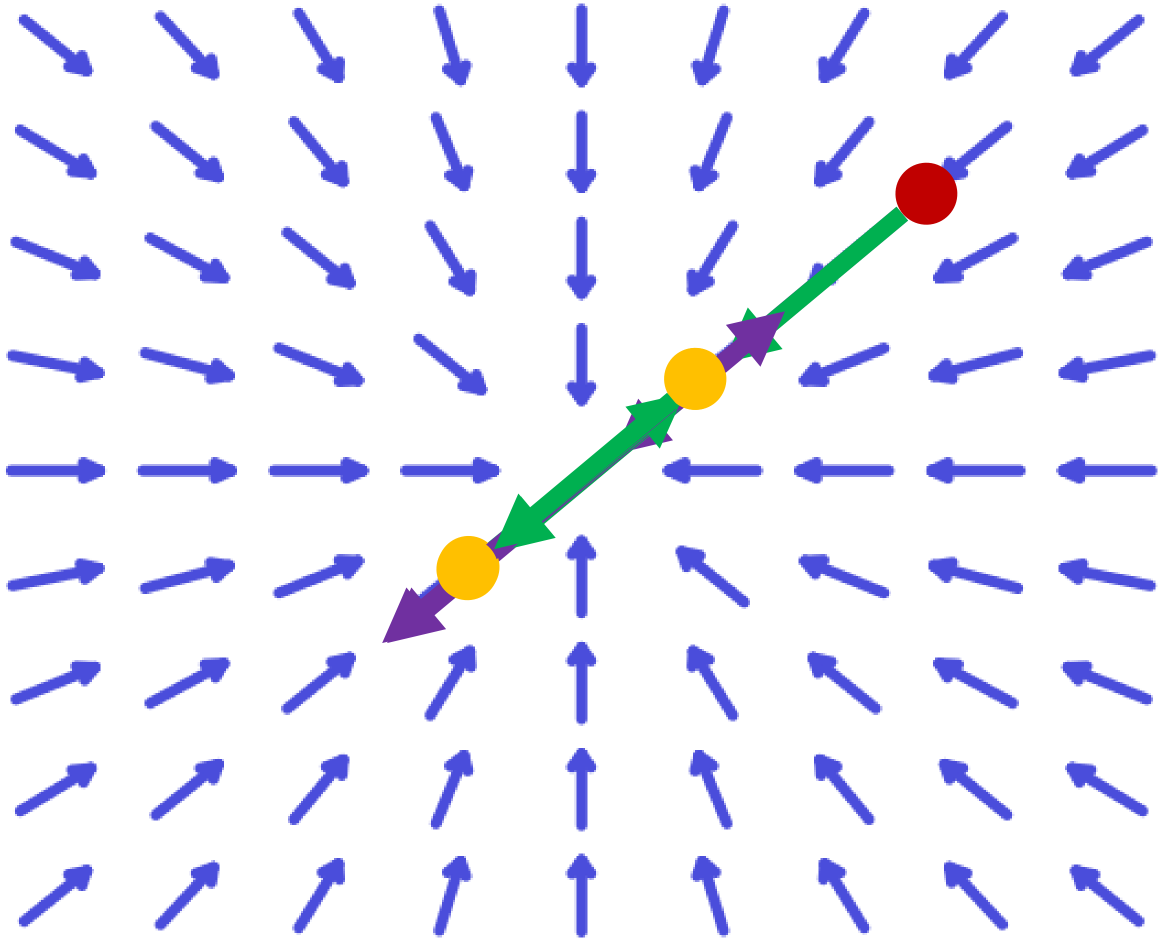


Definition: Explicit

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

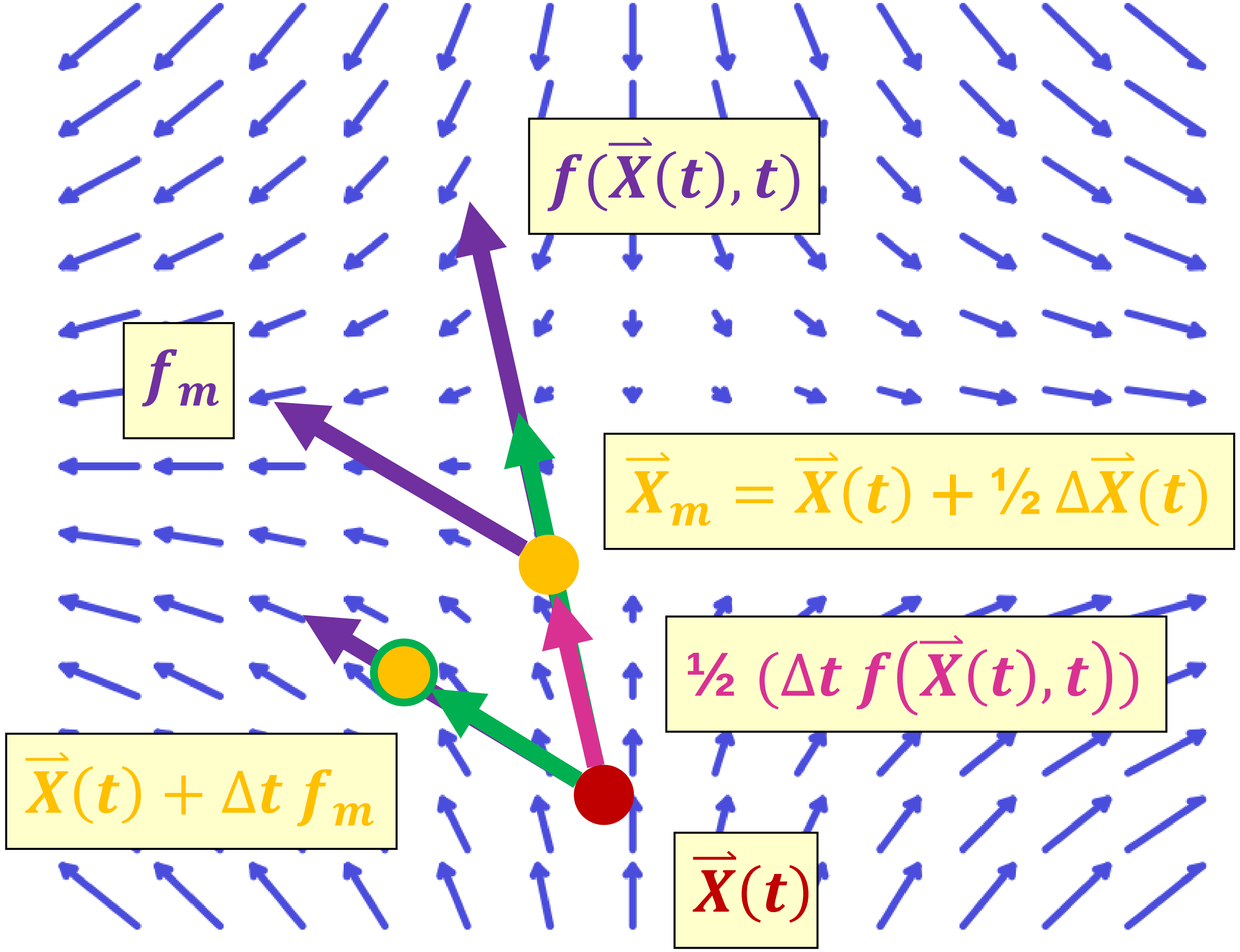
Explicit Euler Problems

- **Can lead to instabilities**



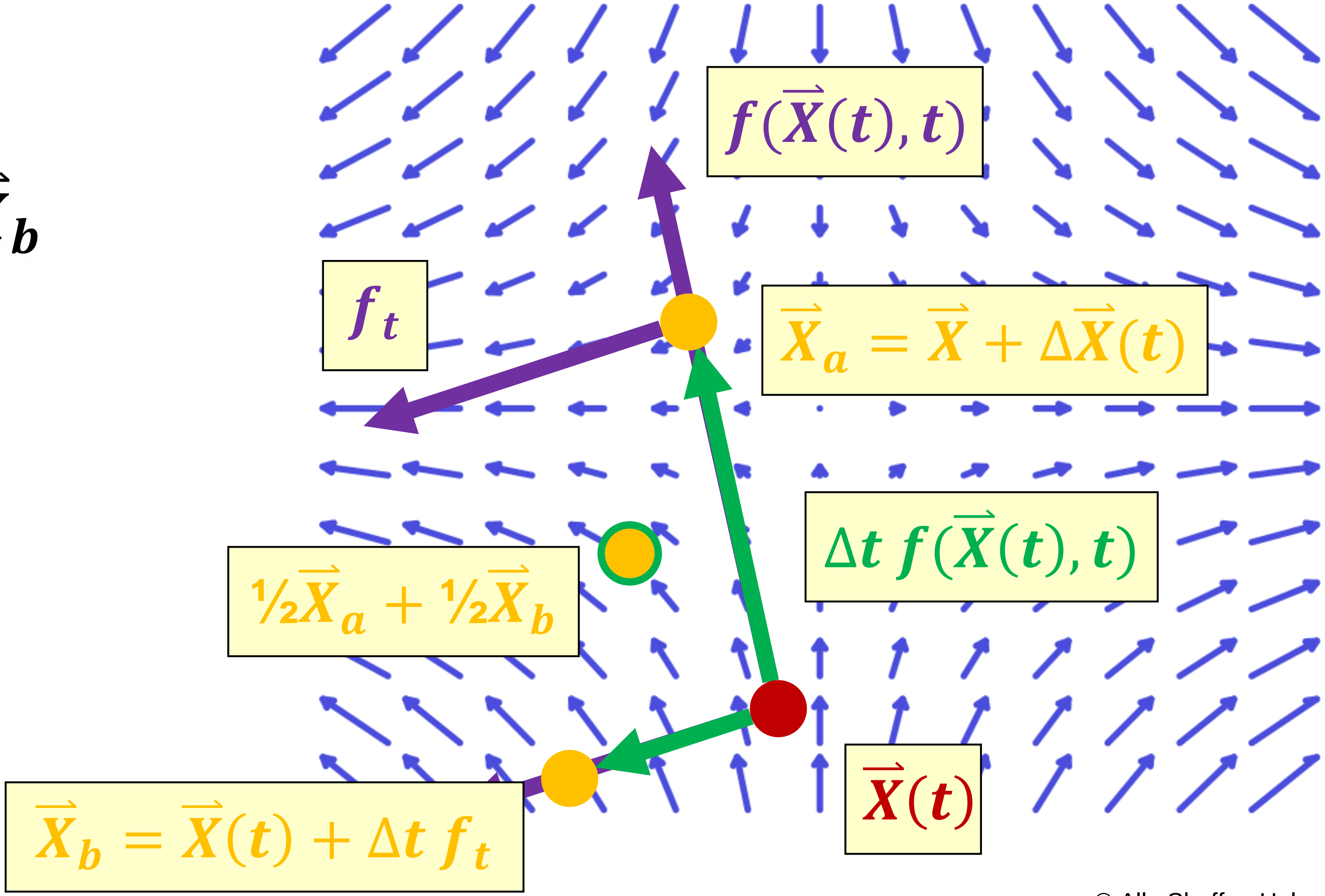
Midpoint Method

1. ½ Euler step
2. evaluate f_m at $\vec{X}_m$
3. full step using f_m



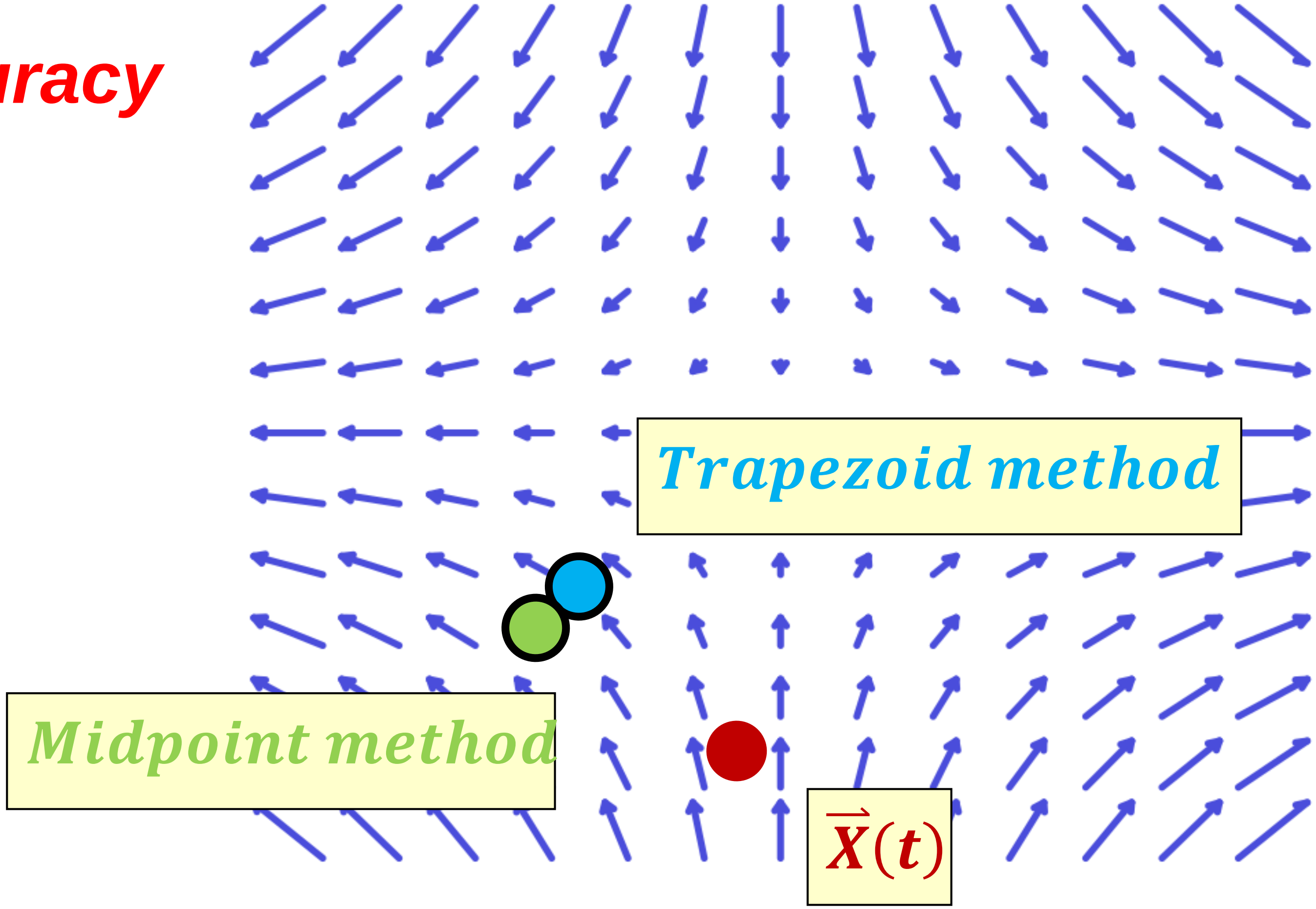
Trapezoid Method

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 $\vec{X}_b$

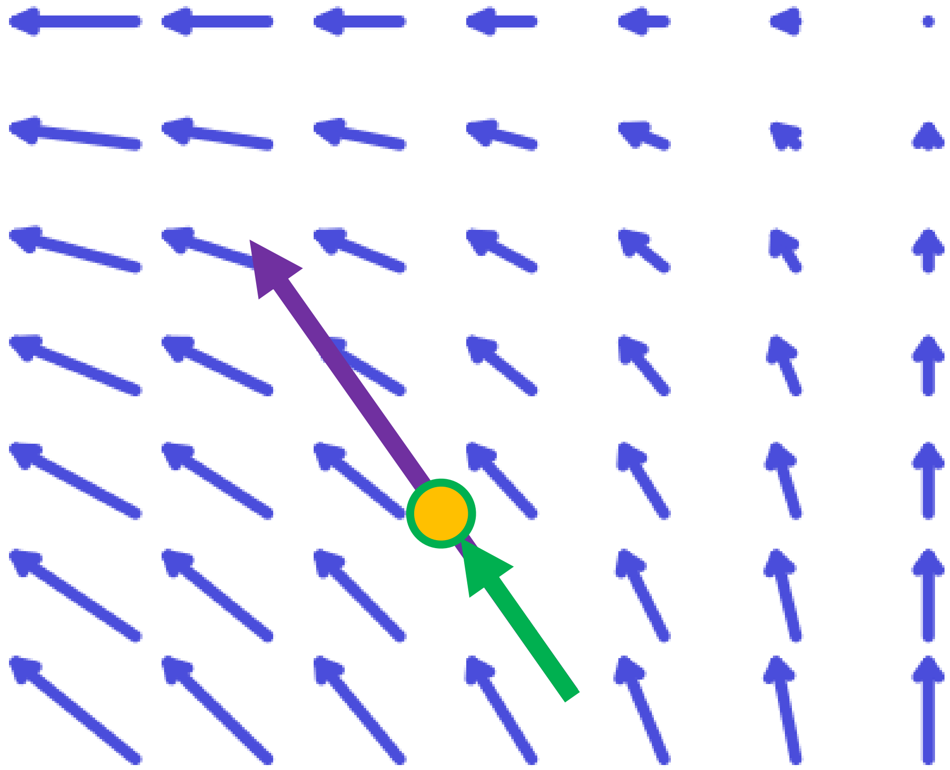


Midpoint & Trapezoid Method

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

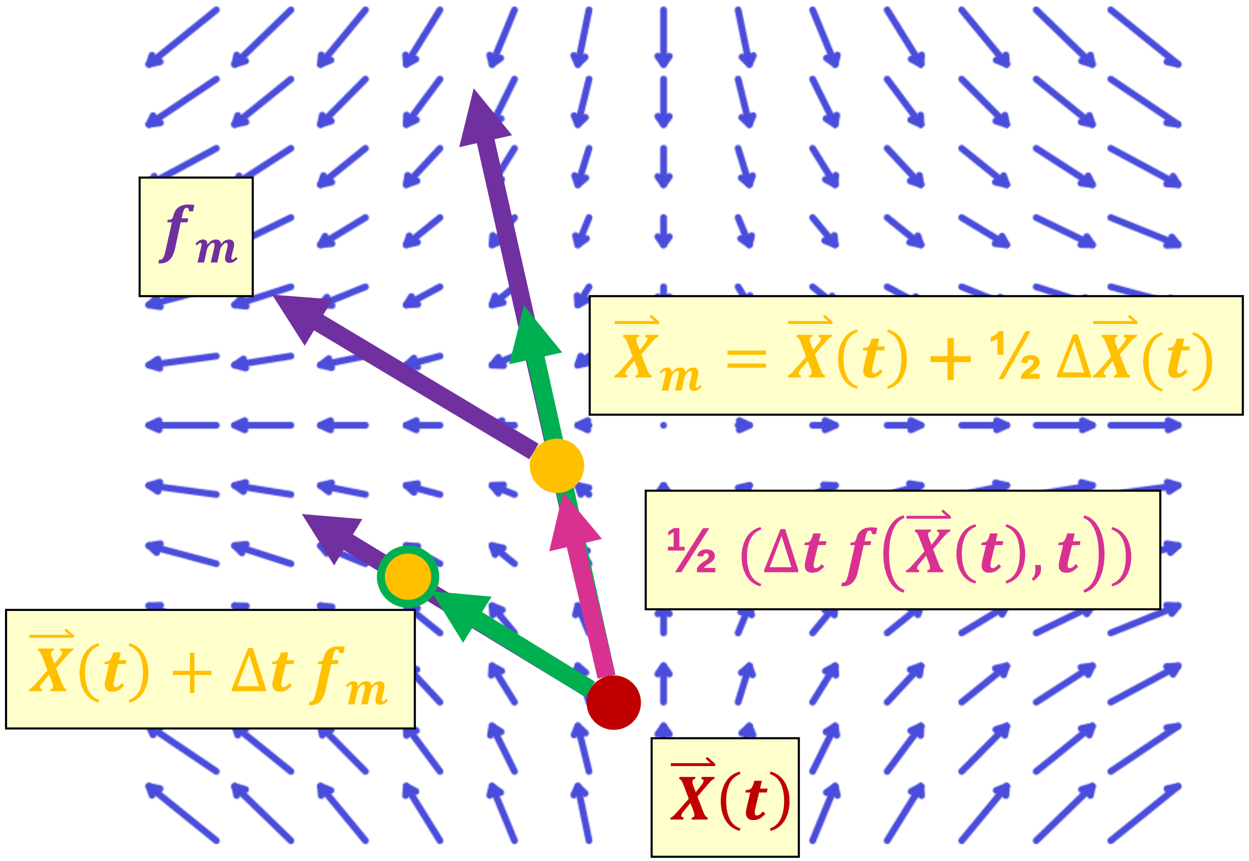


Explicit Euler: Code



```
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)
}
```

Implicit (Backward) Euler:

- **Use forces at destination**

Solve system of equations

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

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

- **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**

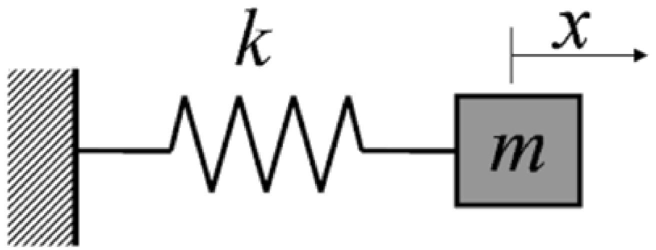
Solve system of equations

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

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

Example: Spring Force

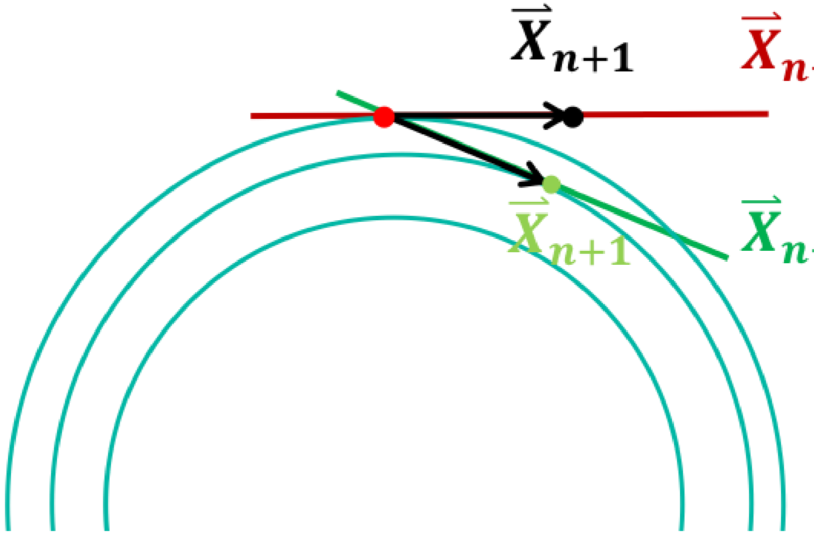
$F = -kx$



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

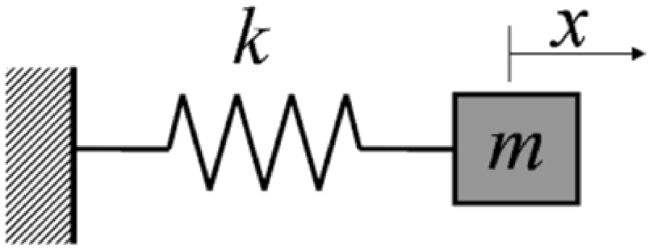
Analytic or iterative solve?

Forward vs Backward



$$\vec{X}_{n+1} = \vec{X}_n + \Delta t\, f(\vec{X}_n)$$

$$\vec{X}_{n+1} = \vec{X}_n + \Delta t\, f(\vec{X}_{n+1})$$



Forward Euler

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

Backward Euler

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

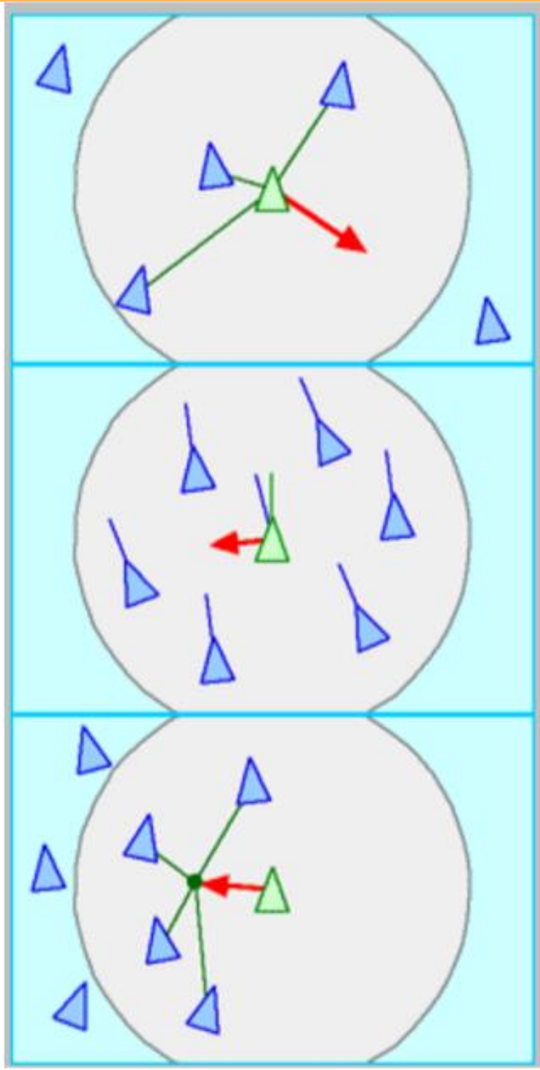
Could one apply the Trapezoid Method?

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!**



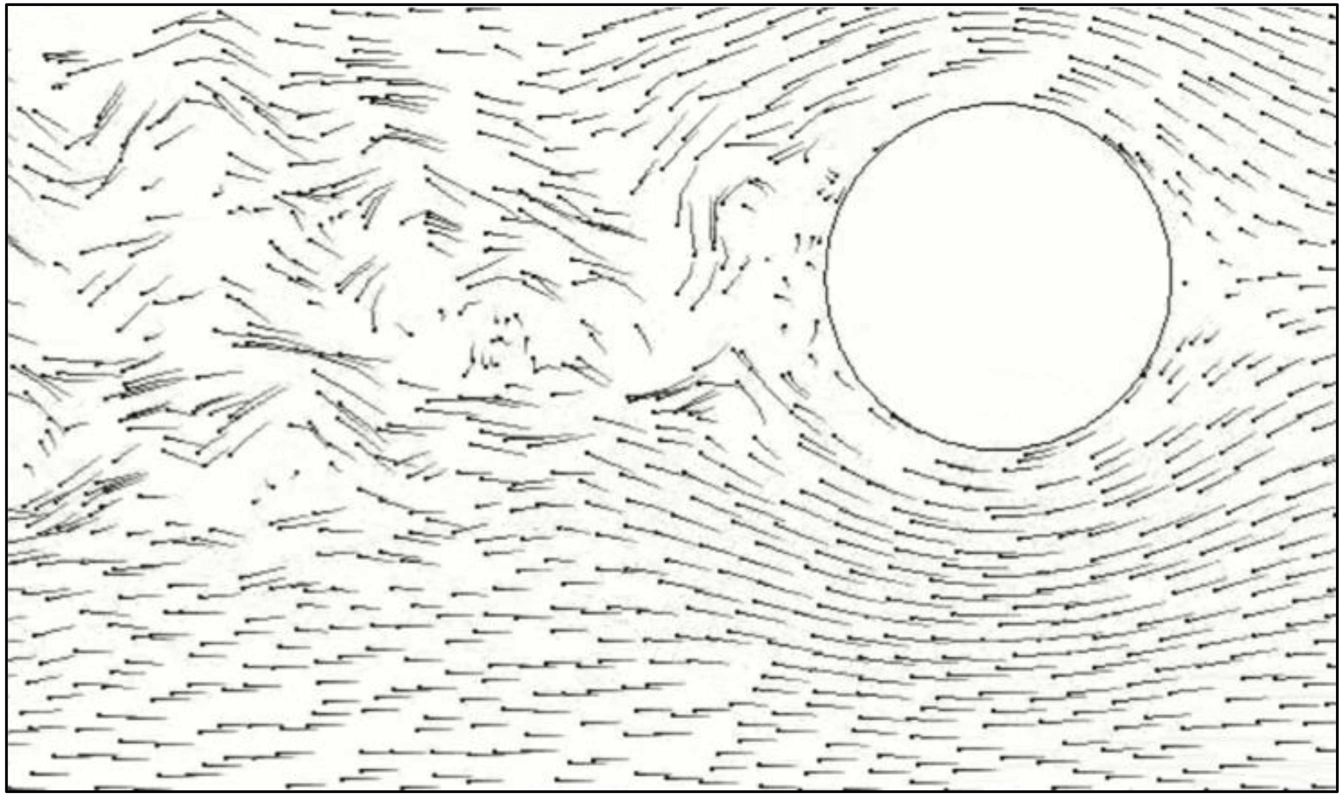
Separation: steer to avoid crowding local flockmates

Alignment: steer towards the average heading of local flockmates

Cohesion: steer to move toward the average position of local flockmates

Proxy Forces

- **Fluids**
  ["Curl Noise for Procedural Fluid Flow" R. Bridson, J. Hourihan, M. Nordenstam, Proc. SIGGRAPH 2007]



- **Many small particles**

- **One can add artificial forces to**
  - ***approximate complex phenomena***
  - ***artistic desires***
  - ***Improve usability (e.g., bias spacecraft to desired trajectory?)***

Particles: Newtonian Physics as First-Order ODE

- **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?**

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, ….*

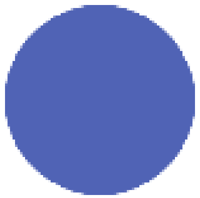
Basic Particle Simulation (first try)

Forces only $\vec{F} = ma$

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

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

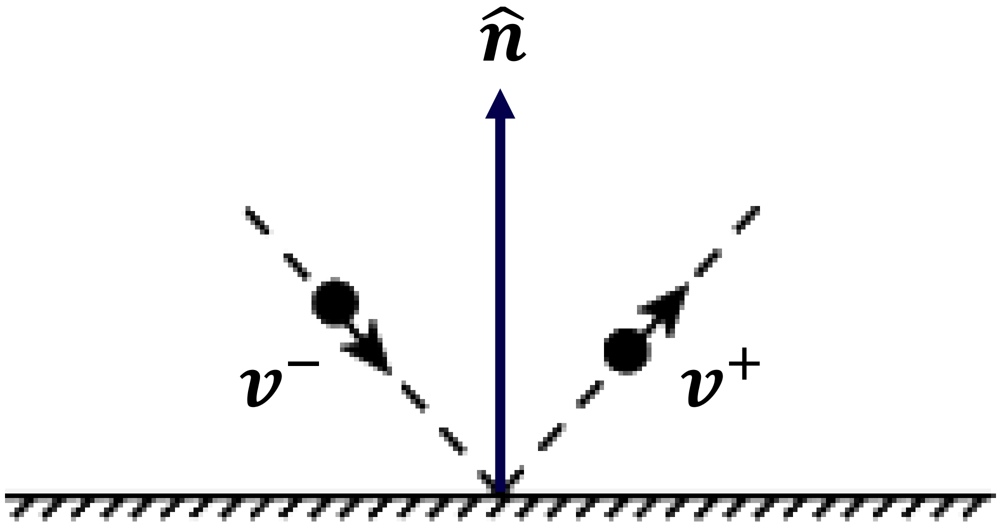
$$\vec{p}_{i+1} = \vec{p}(t_i) + \vec{v}(t_{i+1})d_t$$



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



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

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

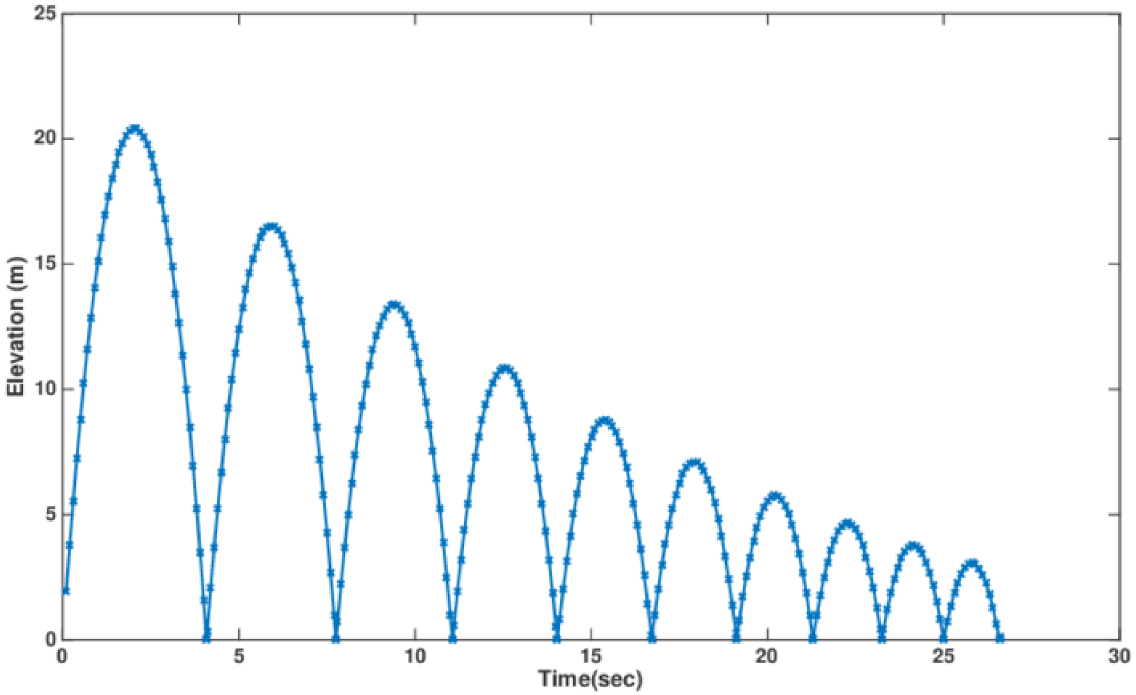
$$v^+ = \frac{\vec{j}}{m} + v^-$$

What is the effect of ϵ ?

Why use 'Impulse'?

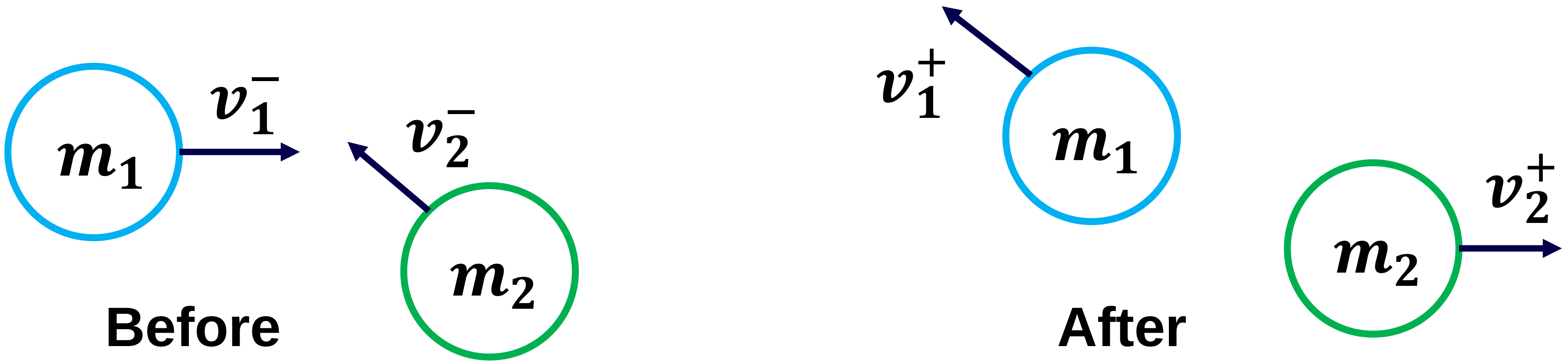
- ***Integrates with the physics solver***

- ***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

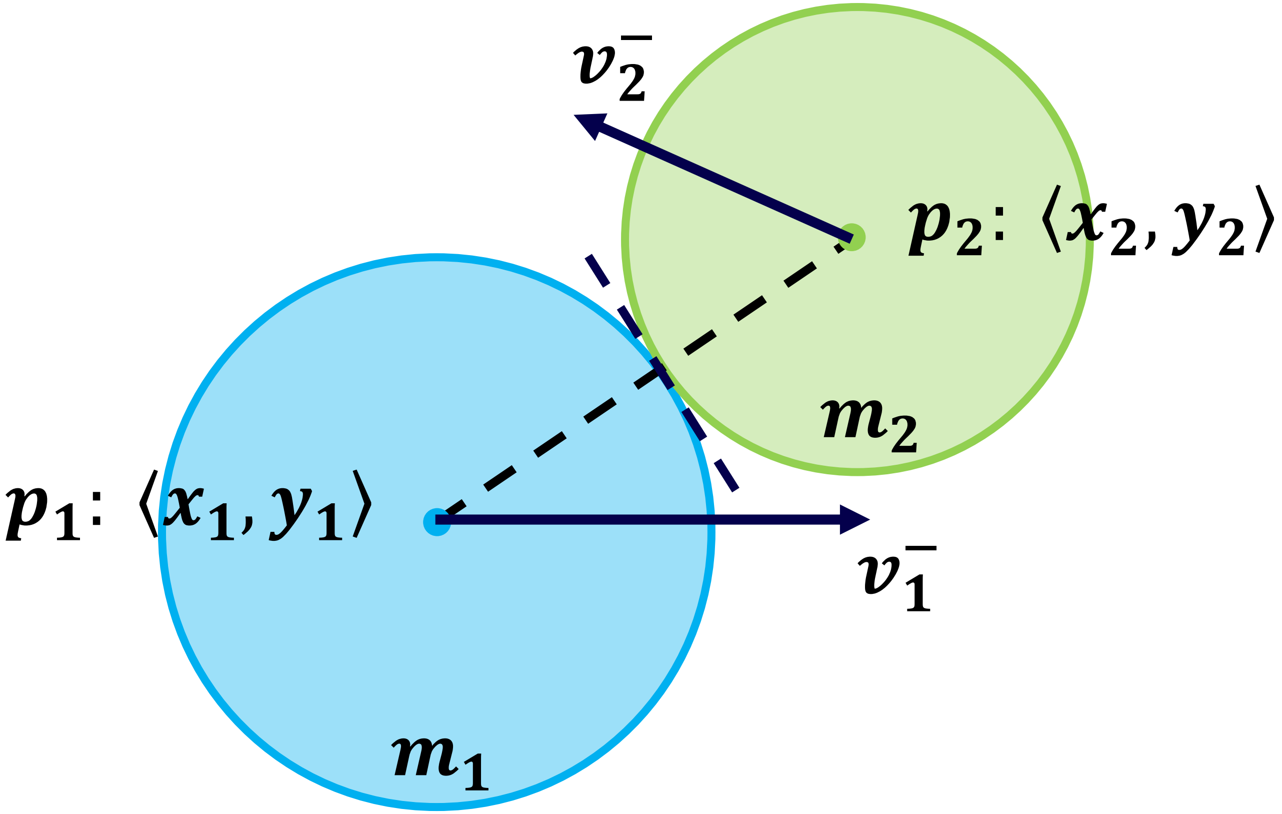
$$½\, m_1 {v_1^-}^2 + ½\, m_2 {v_2^-}^2 = ½\, m_1 {v_1^+}^2 + ½\, m_2 {v_2^+}^2$$

- Velocity is preserved in tangential direction

$$t \circ v_1^- = t \circ v_1^+ , \qquad t \circ v_2^- = t \circ v_2^+$$

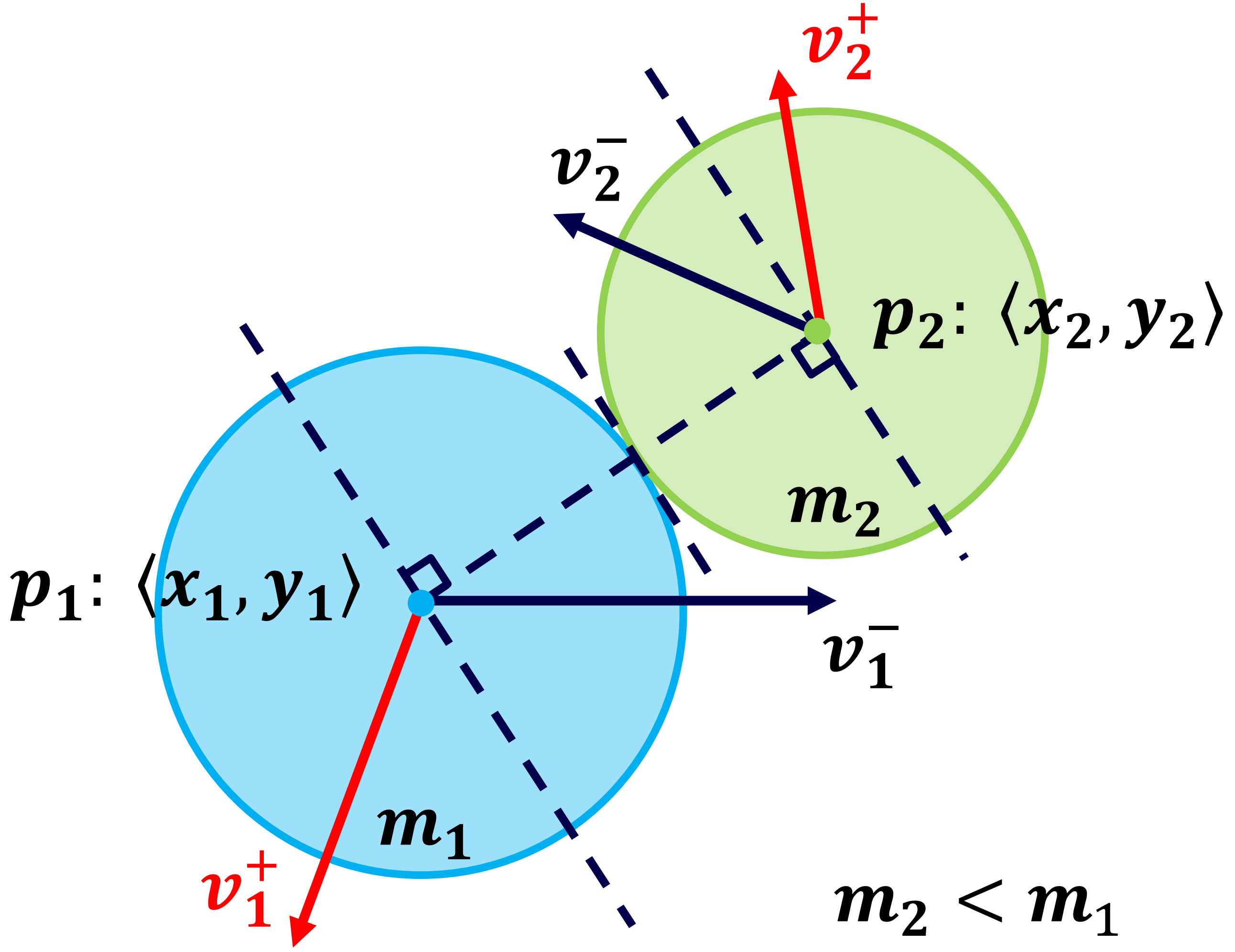
Particle-Particle Collisions (radius >0)

- **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…**

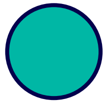
$$\boldsymbol{v}_1^+ = \boldsymbol{v}_1^- - \frac{2\boldsymbol{m}_2}{\boldsymbol{m}_1 + \boldsymbol{m}_2} \frac{\langle \boldsymbol{v}_1^- - \boldsymbol{v}_2^- \rangle \cdot \langle \boldsymbol{p}_1 - \boldsymbol{p}_2 \rangle}{\|\boldsymbol{p}_1 - \boldsymbol{p}_2\|^2} \langle \boldsymbol{p}_1 - \boldsymbol{p}_2 \rangle$$

$$\boldsymbol{v}_2^+ = \boldsymbol{v}_2^- - \frac{2\boldsymbol{m}_1}{\boldsymbol{m}_1 + \boldsymbol{m}_2} \frac{\langle \boldsymbol{v}_2^- - \boldsymbol{v}_1^- \rangle \cdot \langle \boldsymbol{p}_2 - \boldsymbol{p}_1 \rangle}{\|\boldsymbol{p}_2 - \boldsymbol{p}_1\|^2} \langle \boldsymbol{p}_2 - \boldsymbol{p}_1 \rangle$$

- 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

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

$\mathbb{R}^4$ **in 2D**

$\mathbb{R}^6$ **in 3D**

Rigid body

$$state = \begin{cases} \vec{x}\ position \\ \vec{v}\ velocity \\ R\ rotation\ matrix\ 3x3 \\ \vec{w}\ angular\ velocity \end{cases}$$

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