Video Game Physics

Miloš Beočanin, MSc
https://github.com/mbeocanin

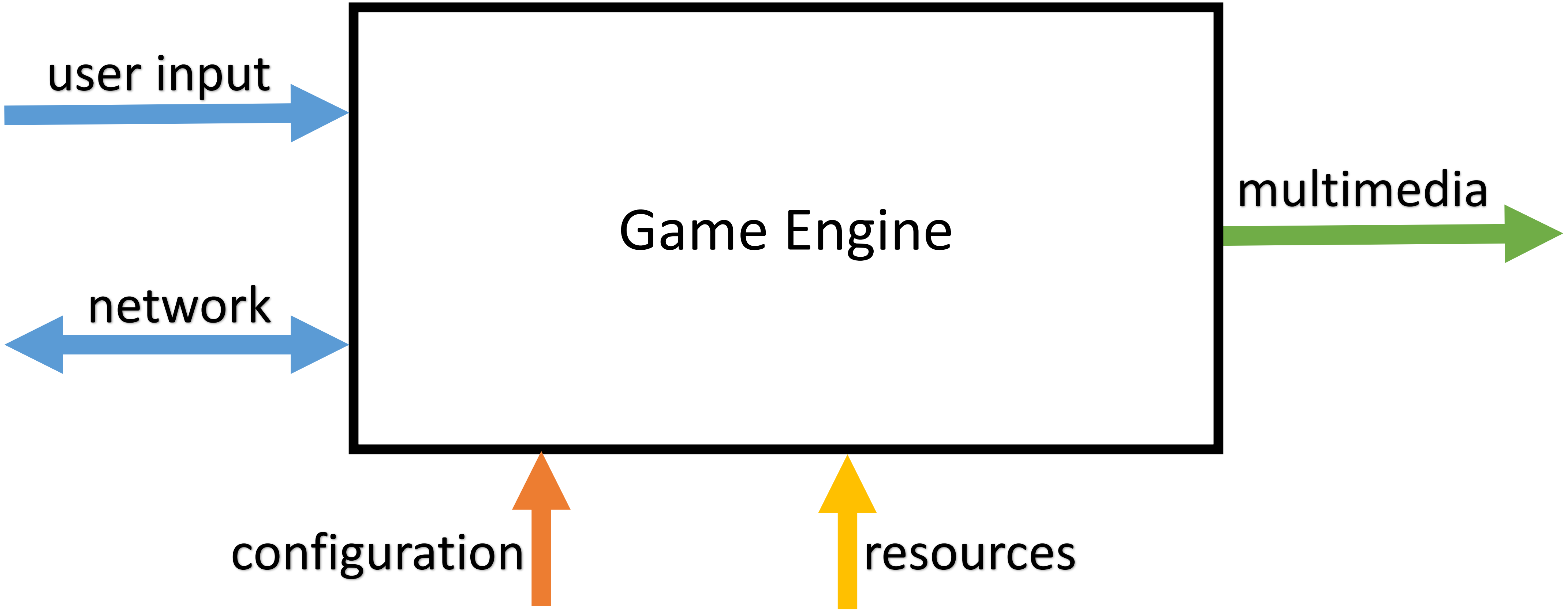
Contents

1. Game engine
2. Phyisics Engine
   - Model
   - Kinematics
   - Constrained motion
3. Demo
4. Q&A

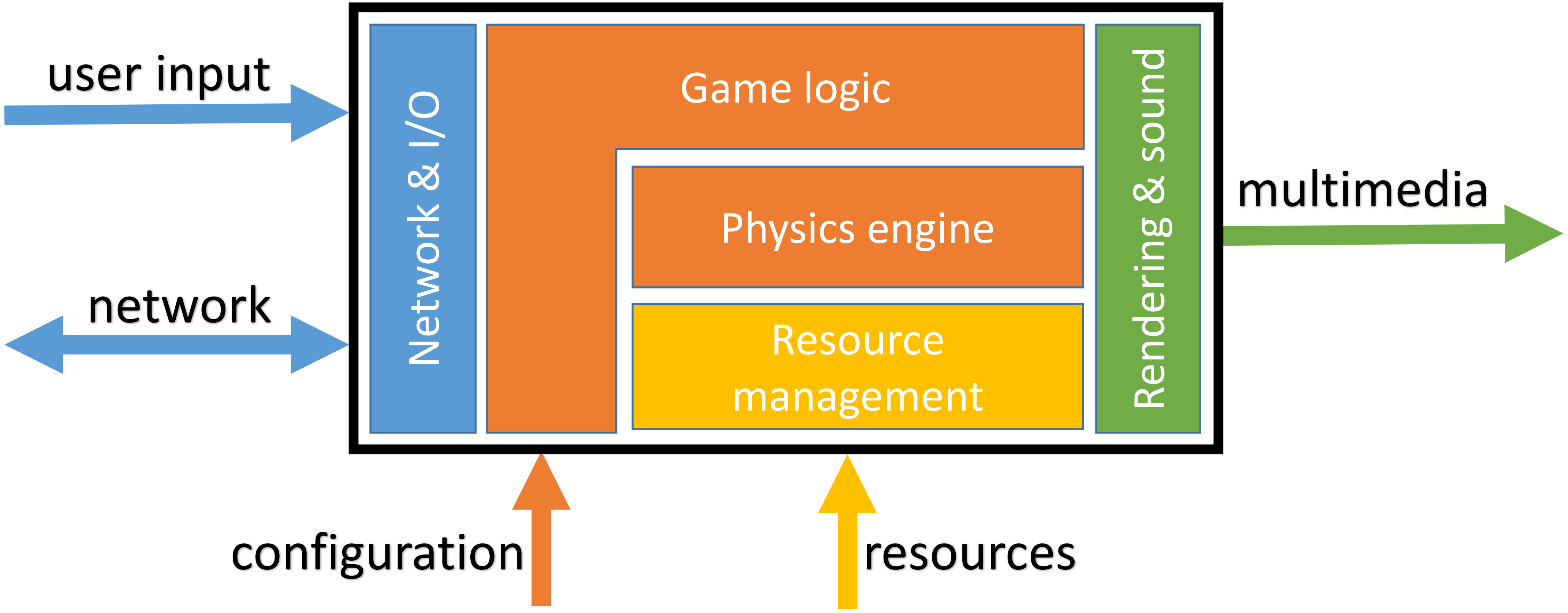
References

- Game Physics – Second Edition, David Eberly
- Erin Catto (*https://box2d.org/publications/*)

Game Engine


user input
network
Game Engine
multimedia
configuration
resources

Game Engine


user input
network
Network & I/O
Game logic
Physics engine
Resource management
Rendering & sound
multimedia
configuration
resources

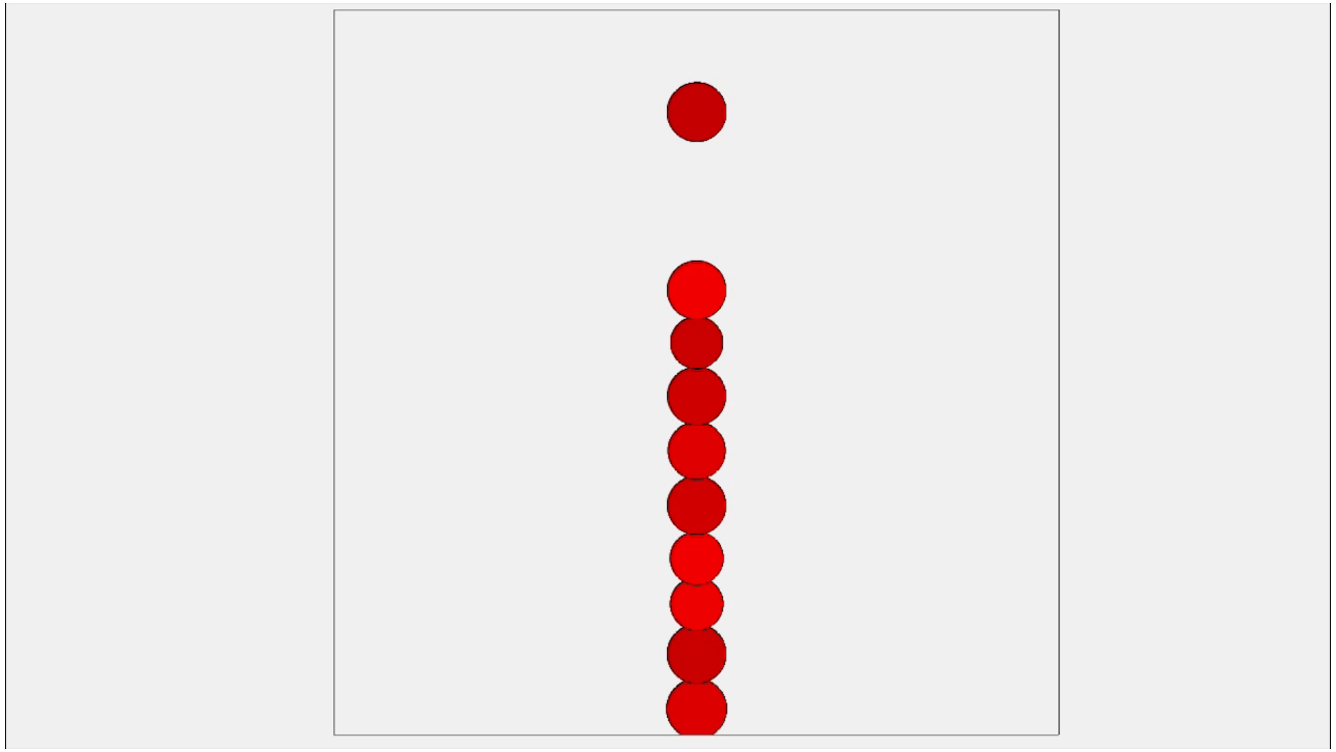
Physics Engine

Fields of study:

1. Physics (classical mechanics)
2. Control theory
3. Numerical algorithms
4. Graphics
5. Mathematical optimization

Physics Engine

1. Model
   I. Space model
   II. Physics body model
2. Kinematics
   1. Newton’s laws of motion
   2. Numerical integration
   3. External forces
3. Constrained motion:
   I. Collision detection
   II. Constraint modeling
   III. Constraint solving



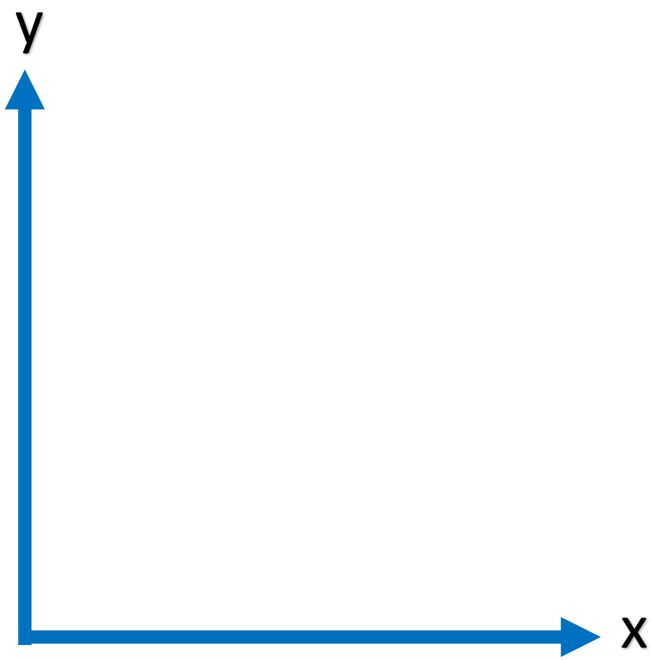
Model

The way to describe the state of a physical system in any given moment?

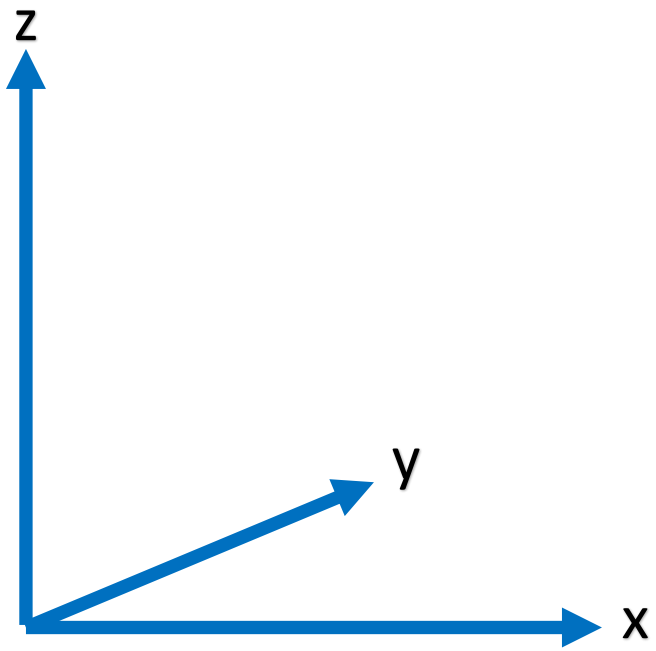
Model

1. Space model

2D



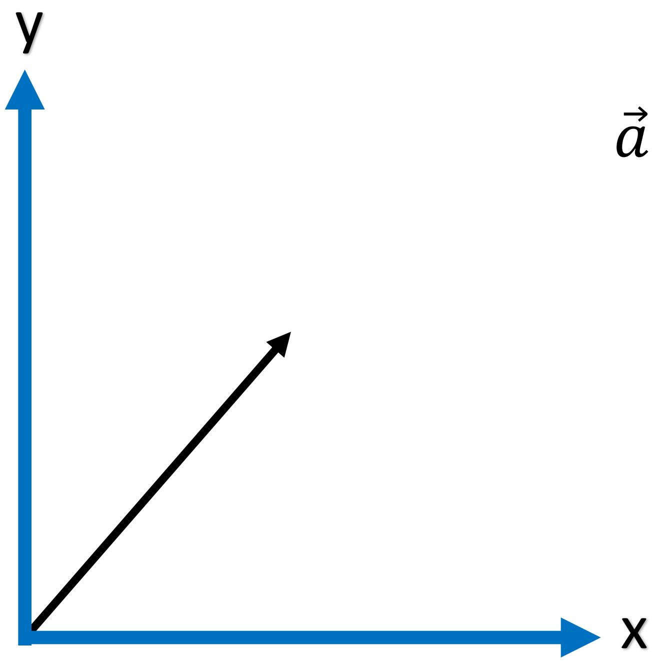
3D



Model

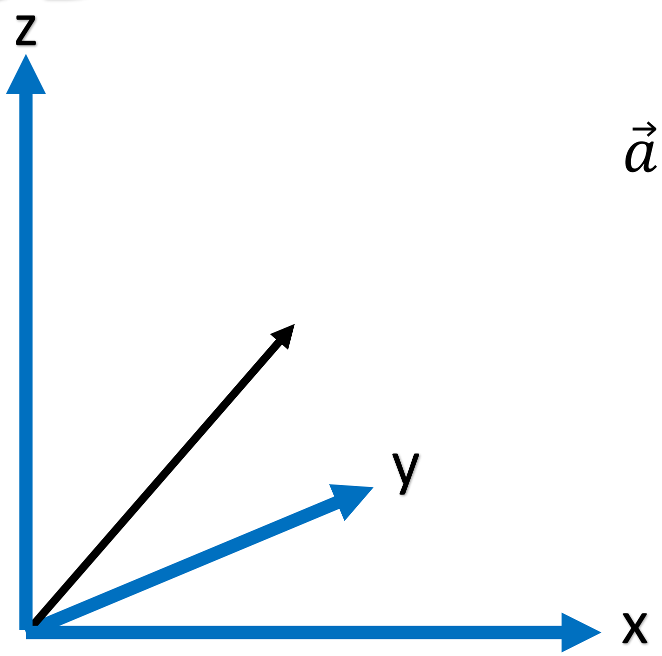
1. Space model
    - Vectors

2D



$$\vec{a} = \begin{bmatrix} x \\ y \end{bmatrix}$$

3D



$$\vec{a} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Model

1. Space model
   - Vectors

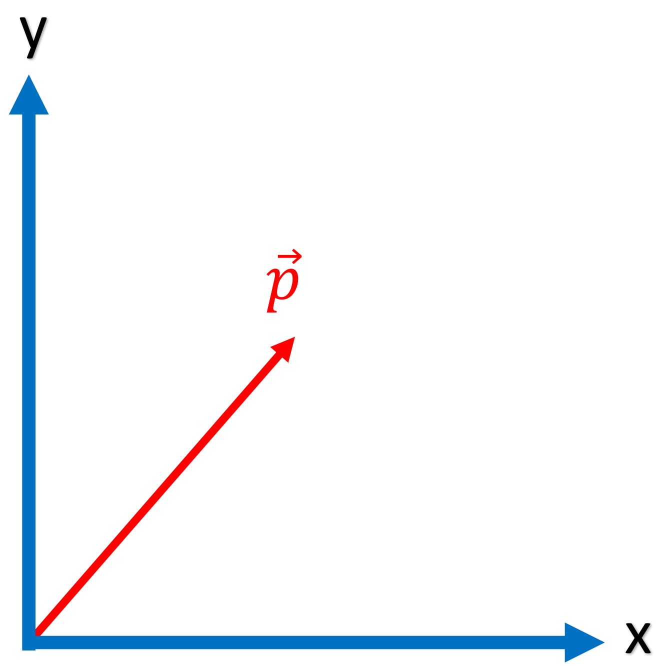
+ vectors generalize the problem (same solutions apply to 2D and 3D)
+ vectors simplify the equations (operations are written simultaneously across all dimensions of space)
+ vectors faster execution (operations are performed simultaneously across all dimensions of space)
+ vectors are suitable for modern hardware
+ most graphics APIs provide implementations of vector algebra

- vectors require familiarity with vector algebra (takes some time to get used to it)

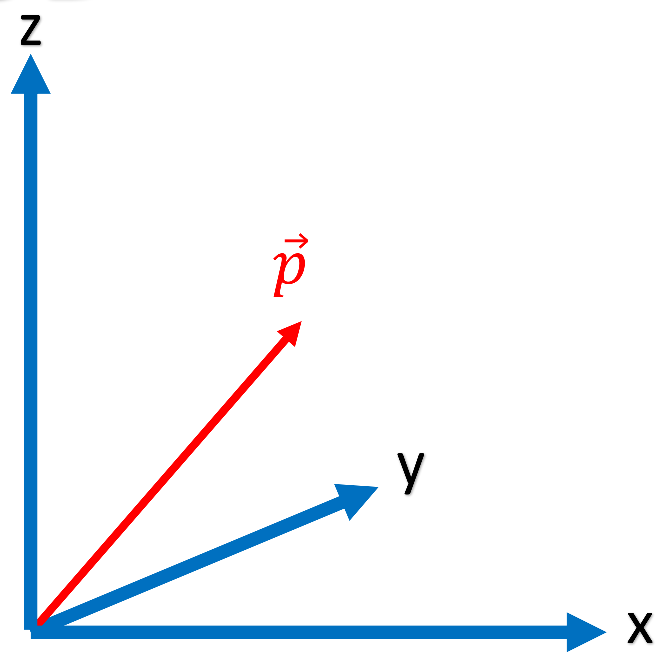
Model

2. Physics body model
   I. position

2D



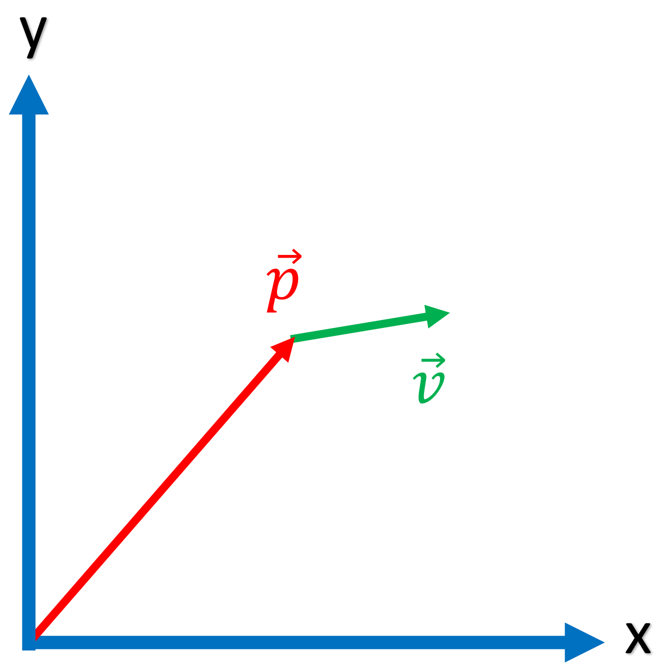
3D



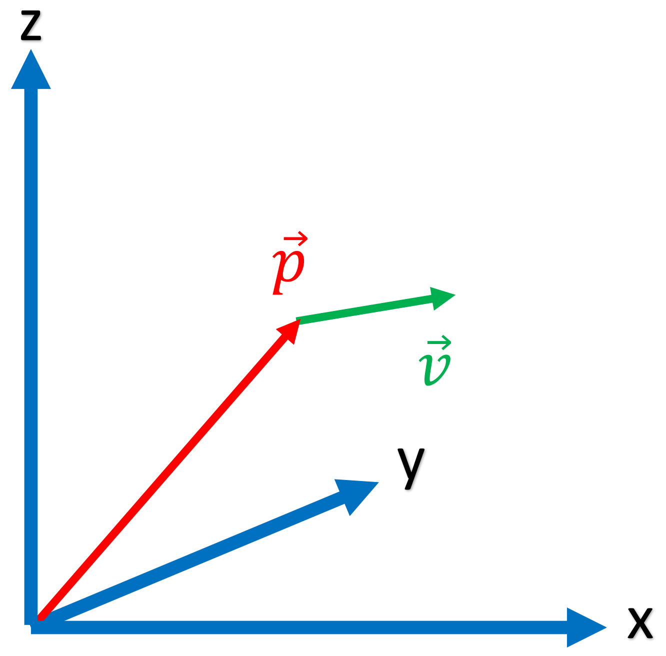
Model

2. Physics body model
   I. position
   II. current velocity

2D



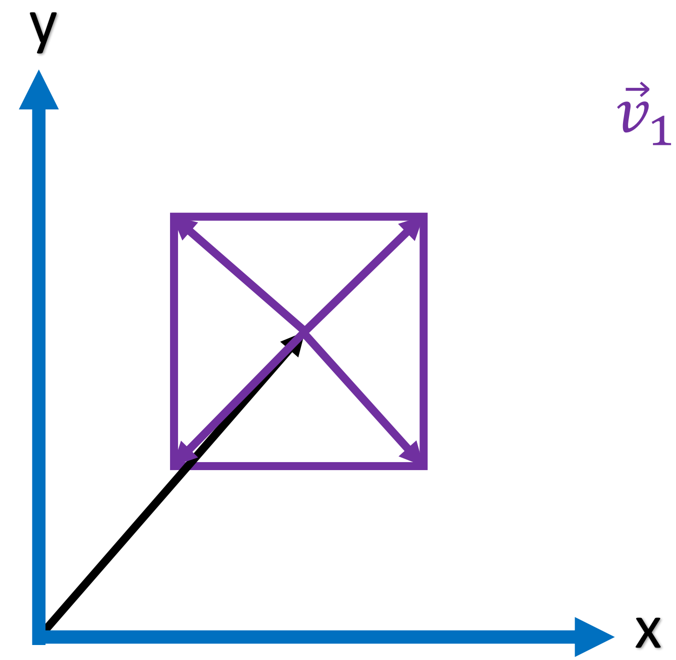
3D



Model

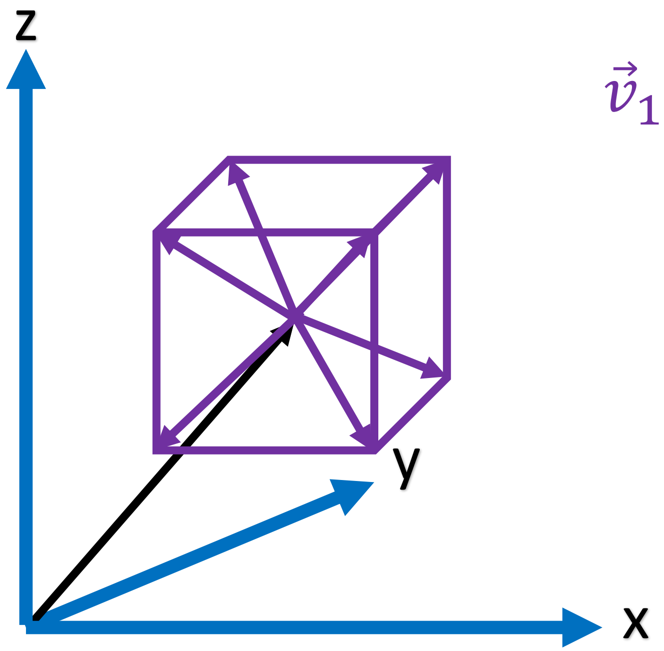
2. Physics body model
   IV. geometry

2D



$\vec{v}_1, \vec{v}_2, \vec{v}_3, \ldots$

3D



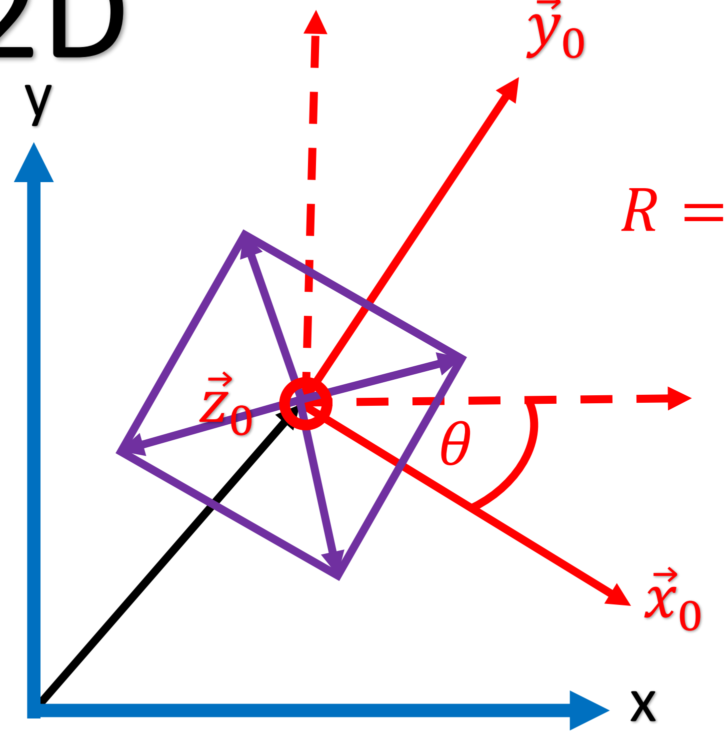
$\vec{v}_1, \vec{v}_2, \vec{v}_3, \ldots$

Model

2. Physics body model

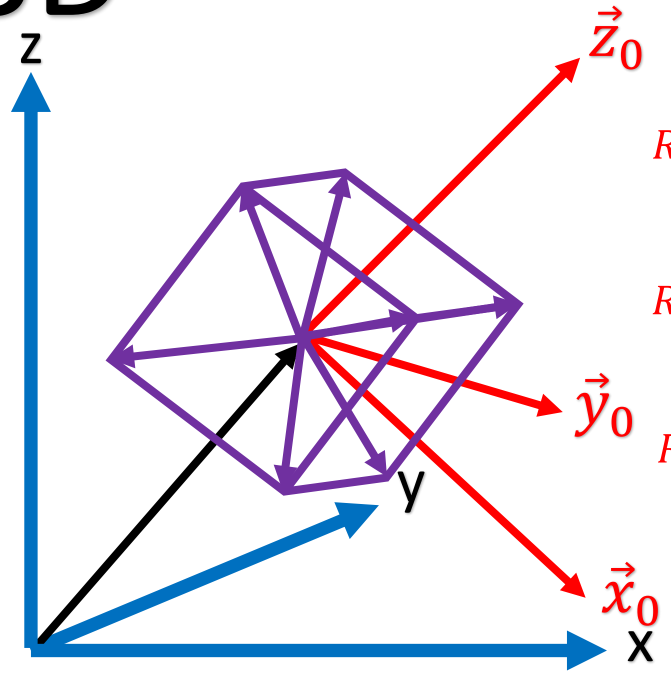
   V. orientation

2D



$$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

3D



$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_x & -\sin\theta_x \\ 0 & \sin\theta_x & \cos\theta_x \end{bmatrix}$$

$$R_y = \begin{bmatrix} \cos\theta_y & 0 & \sin\theta_y \\ 0 & 1 & 0 \\ -\sin\theta_y & 0 & \cos\theta_y \end{bmatrix}$$

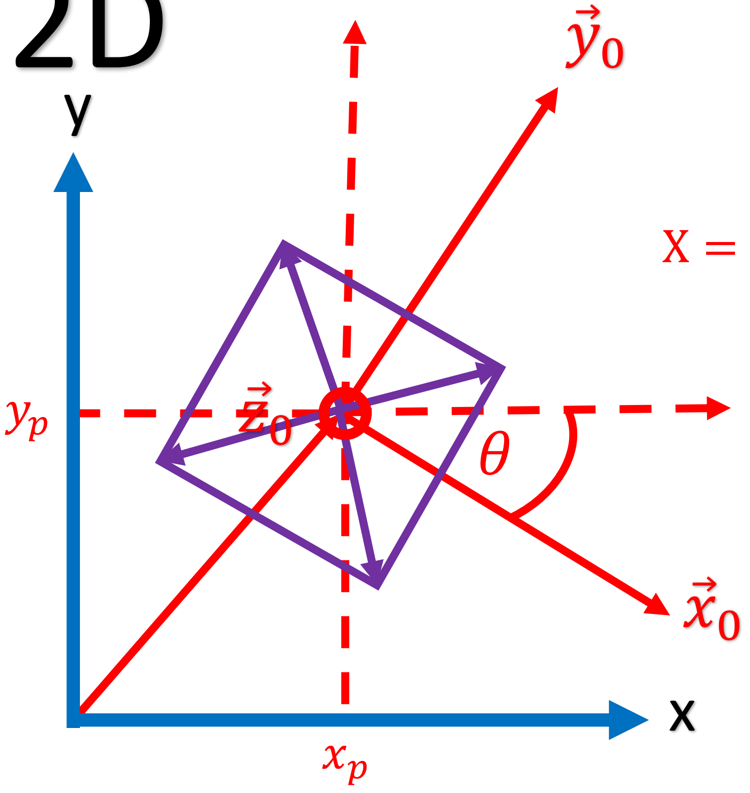
$$R_z = \begin{bmatrix} \cos\theta_z & -\sin\theta_z & 0 \\ \sin\theta_z & \cos\theta_z & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R = R_x R_y R_z$$

Model

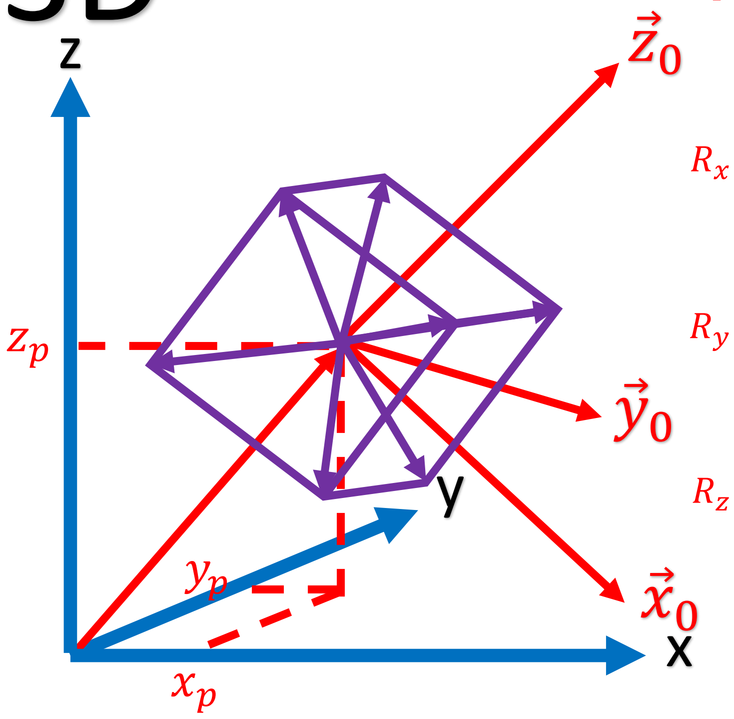
2. Physics body model
   - transformation matrix: position + orientation (homogeneous coordinates)

2D



$$X = \begin{bmatrix} \cos\theta & -\sin\theta & x_p \\ \sin\theta & \cos\theta & y_p \\ 0 & 0 & 1 \end{bmatrix}$$

3D



$$T = \begin{bmatrix} 1 & 0 & 0 & x_p \\ 0 & 1 & 0 & y_p \\ 0 & 0 & 1 & z_p \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta_x & -\sin\theta_x & 0 \\ 0 & \sin\theta_x & \cos\theta_x & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_y = \begin{bmatrix} \cos\theta_y & 0 & \sin\theta_y & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta_y & 0 & \cos\theta_y & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

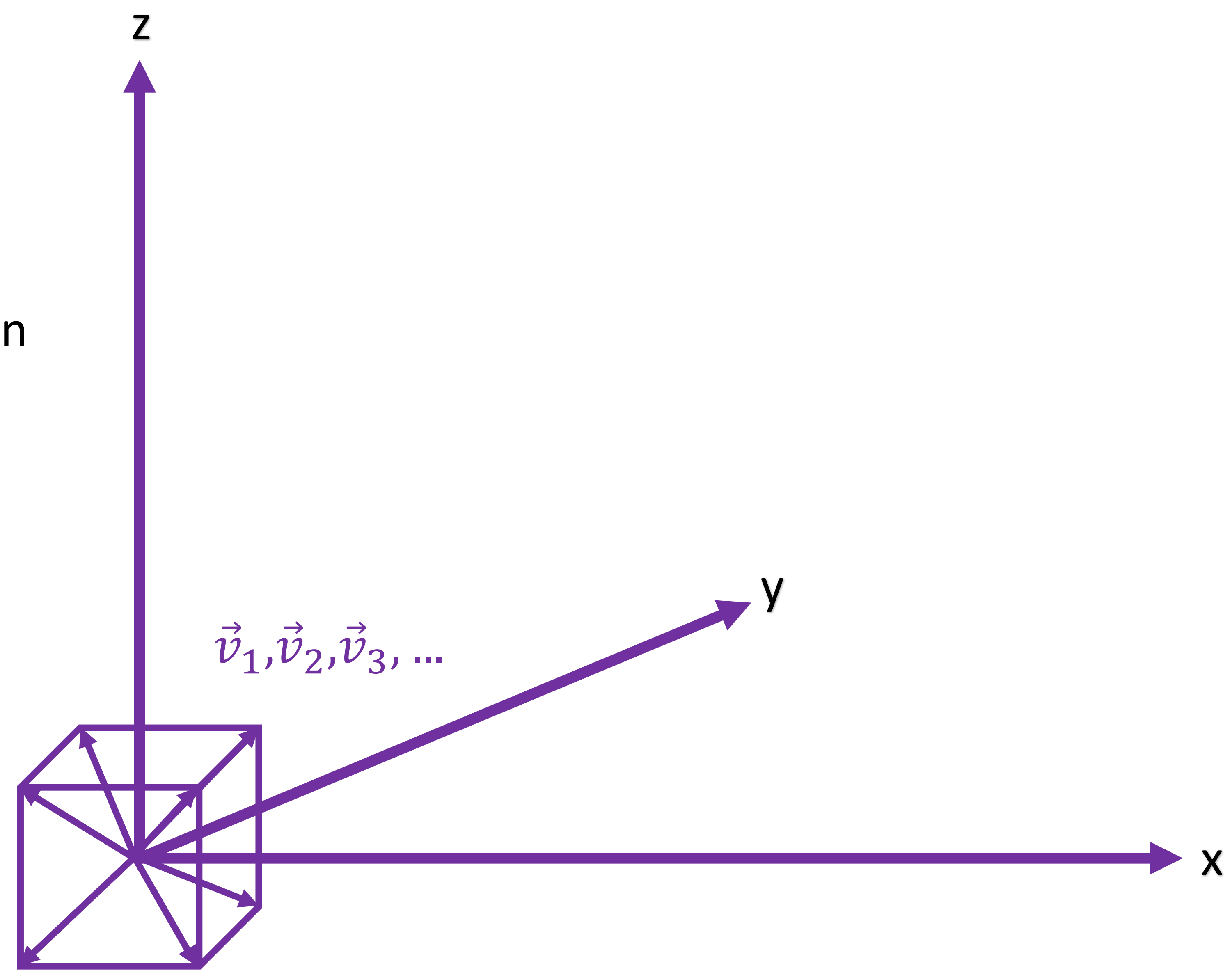
$$R_z = \begin{bmatrix} \cos\theta_z & -\sin\theta_z & 0 & 0 \\ \sin\theta_z & \cos\theta_z & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$X = R_z R_y R_x T$$

Model

2. Physics body model
   - transformation matrix

   I. local coordinate. system

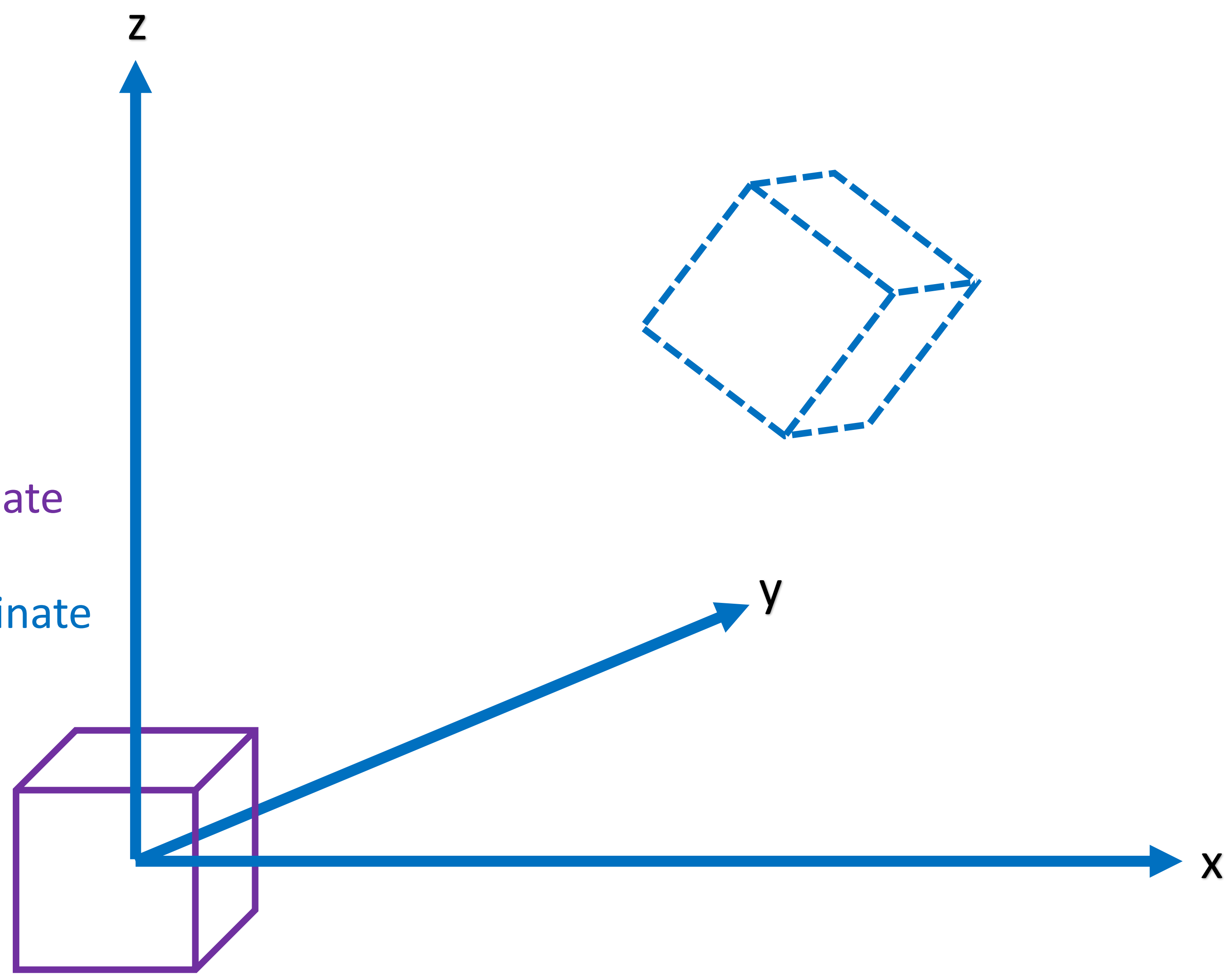


Model

2. Physics body model
   - transformation matrix

I. local coordinate system

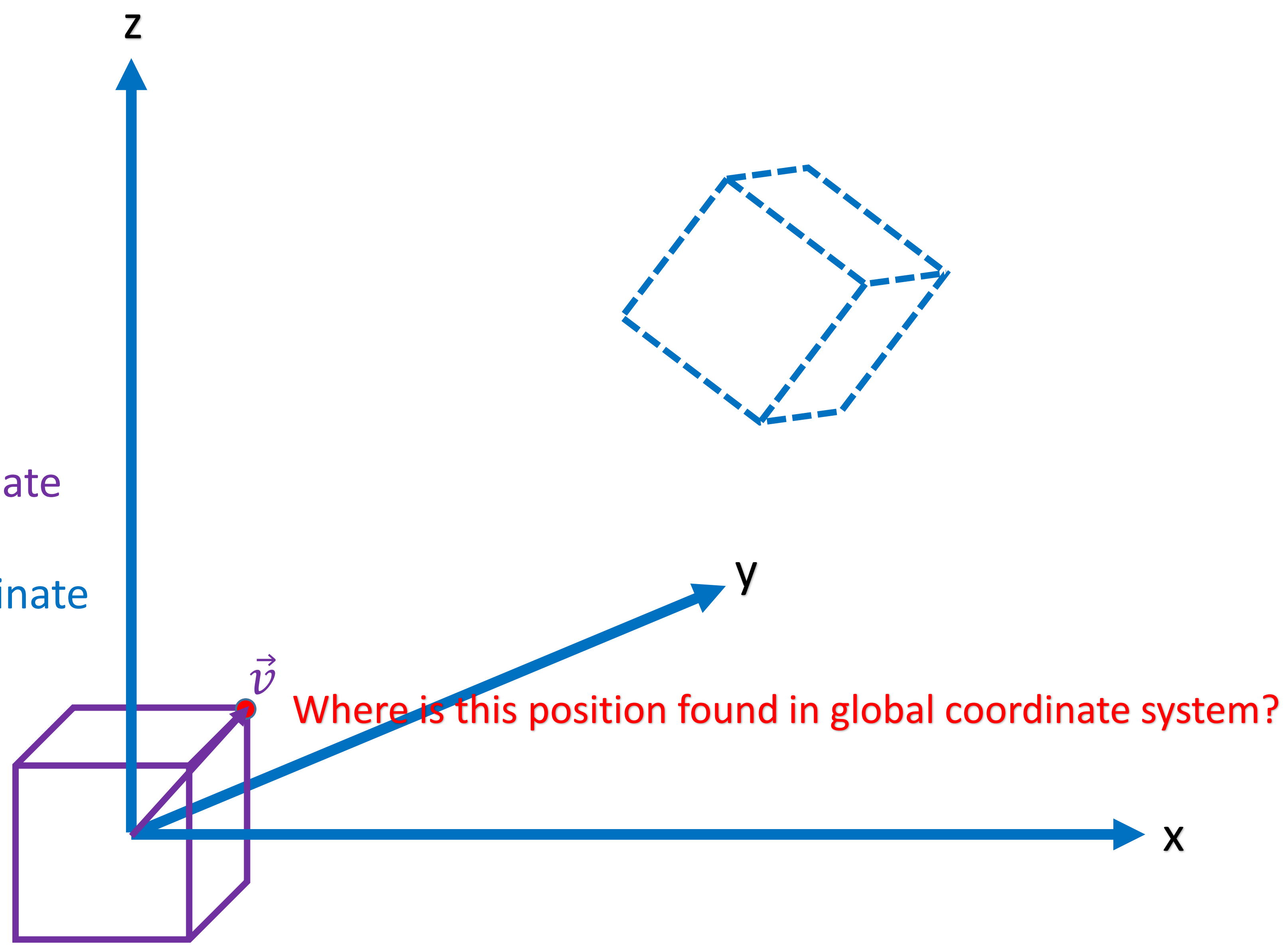
II. global coordinate system



Model

2. Physics body model
   - transformation matrix

I. local coordinate system
II. global coordinate system

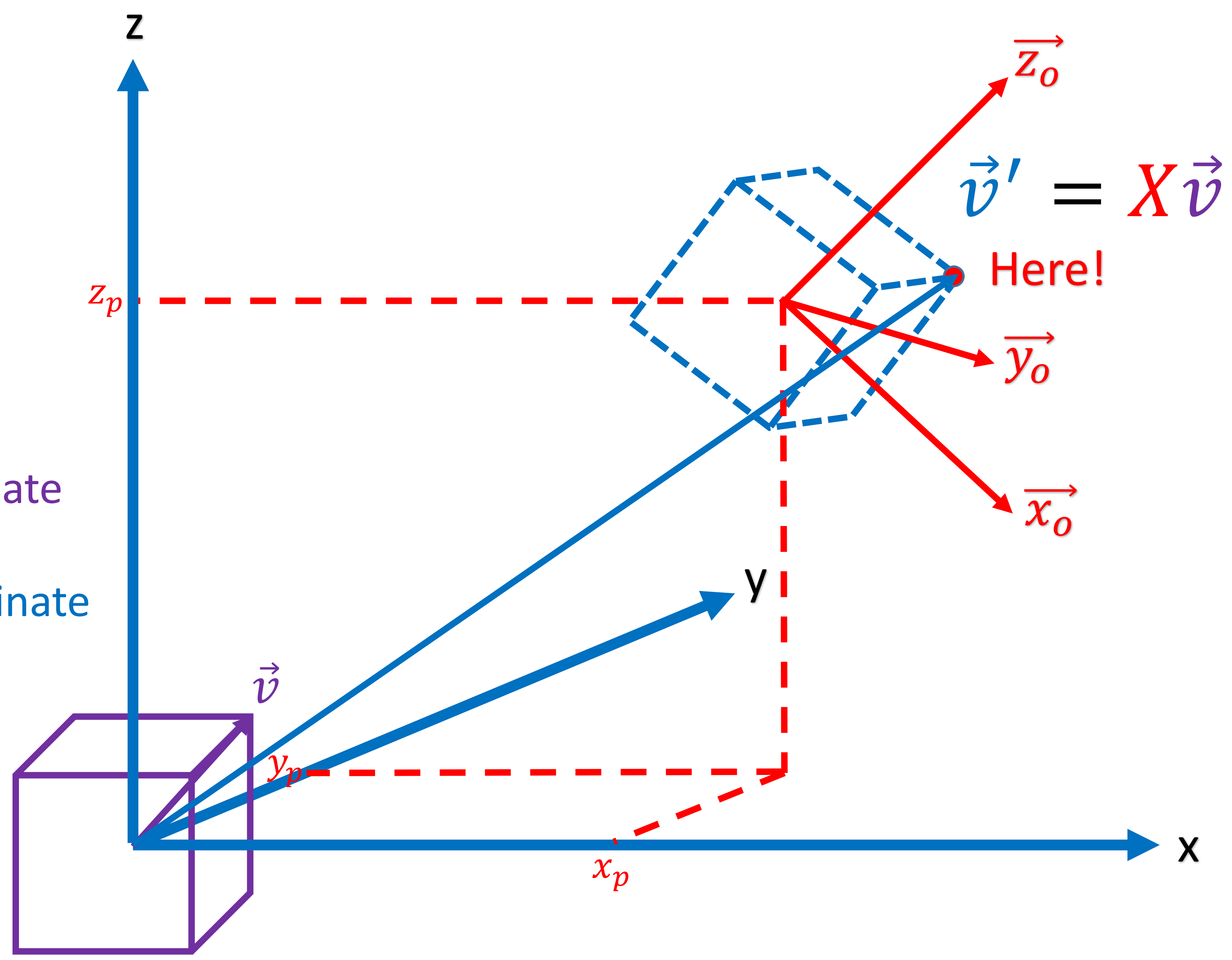


Model

2. Physics body model
   - transformation matrix

I. local coordinate system

II. global coordinate system

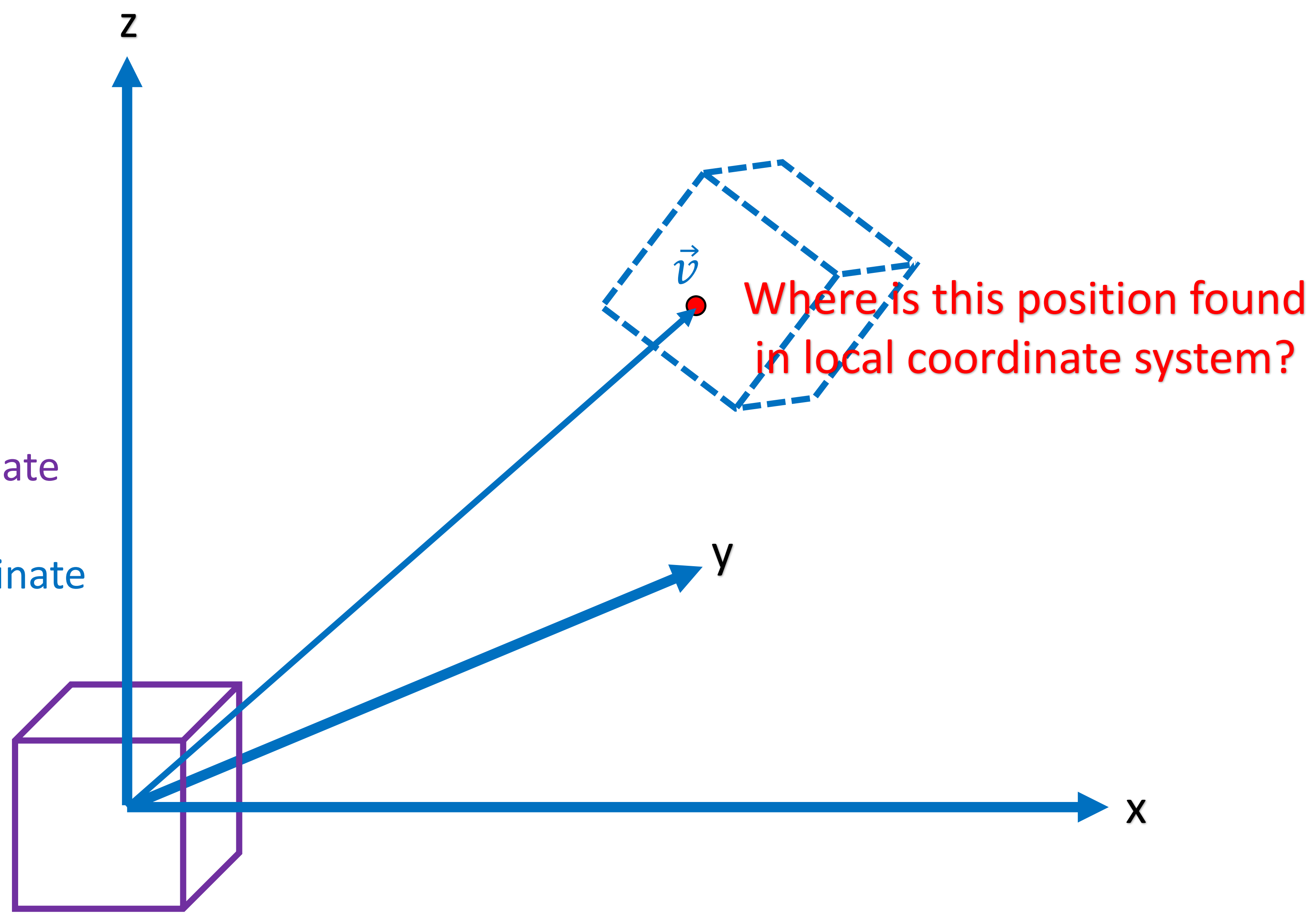


Model

2. Physics body model
   - transformation matrix

I. local coordinate system

II. global coordinate system

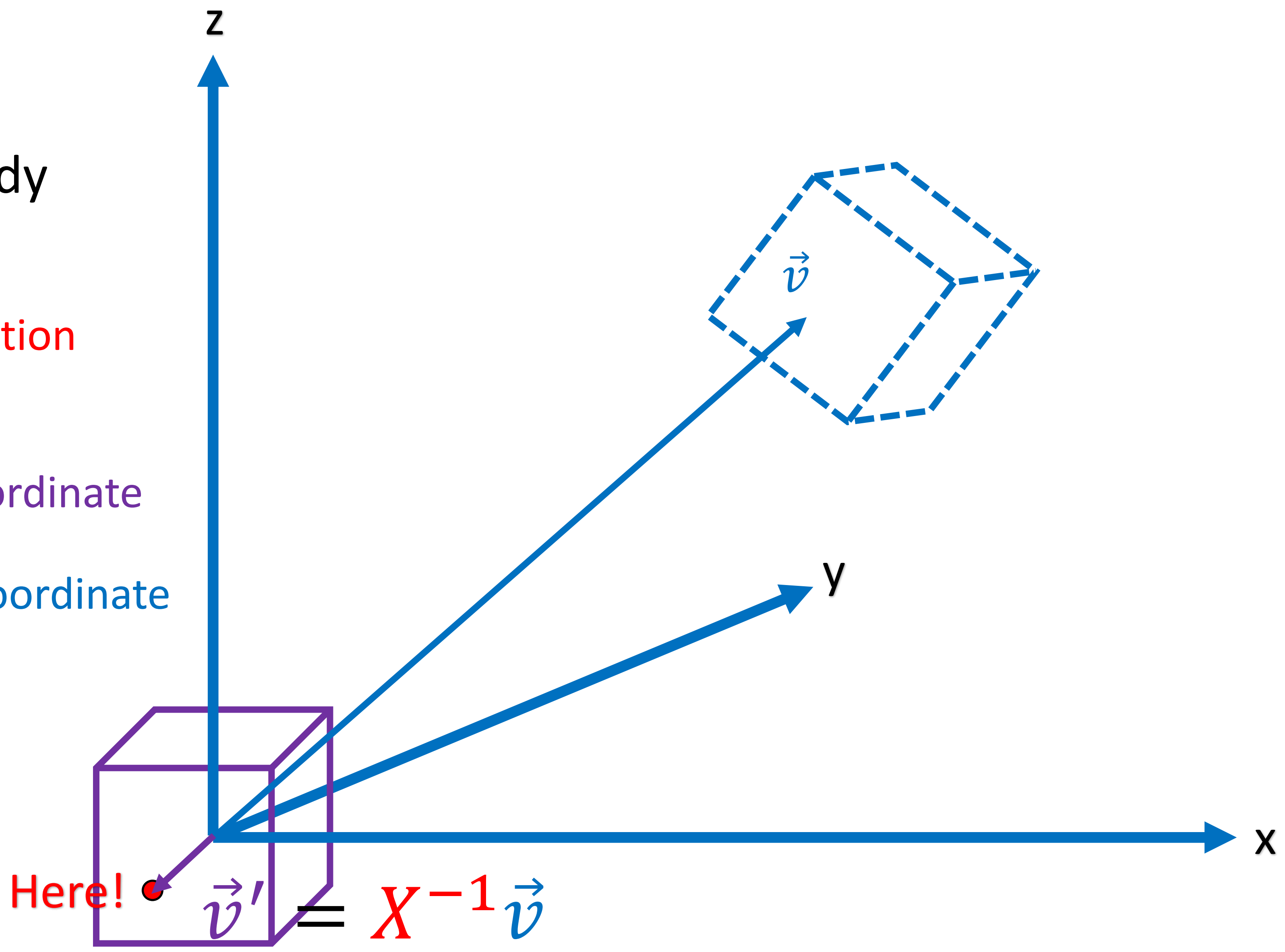


Model

2. Physics body model
   - (inverse) transformation matrix

   I. local coordinate system

   II. global coordinate system



Model

2. Physics body model
    - Transformation matrix

+ body geometry is precomputed in local coordinate system and remains constant
+ as the body moves, only it's position and orientation are updated, but not the entire geometry (much faster)
+ graphics APIs use precomputed body geometry and transformation matrices to render the result, thus the graphics hardware does all the work
+ most graphics APIs provide transformation matrices implementation

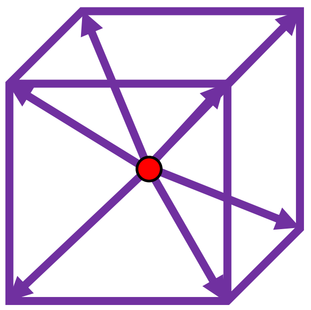
- in case of calculations which require interaction of multiple bodies (e.g. collision detection), geometries of both bodies must be transformed to the same coordinate system (global coordinate system or local coordinate system of either of those bodies); these calculations are usually done on CPU, but are much less frequent than rendering

Model

2. Physics body model
   - geometric center (centroid)

$$\vec{C}_g = \frac{\sum_{i=1}^{n} \vec{v}_i}{n}$$

$\vec{v}_1, \vec{v}_2, \vec{v}_3, \dots$



Model

2. Physics body model
   - geometric center (centroid)
   - center of mass

$$\vec{C}_g = \frac{\sum_{i=1}^{n} \vec{v}_i}{n}$$

$$\vec{C}_m = \frac{\sum_{i=1}^{n} \vec{v}_i m_i}{\sum_{i=1}^{n} m_i}$$

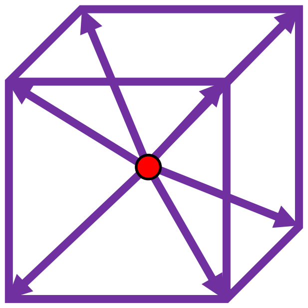
$\vec{v}_1, \vec{v}_2, \vec{v}_3, \ldots$

for homogenous bodies:

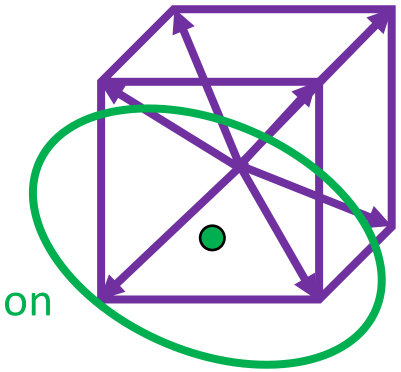
$$\vec{C}_m = \vec{C}_g$$

$\vec{v}_1, \vec{v}_2, \vec{v}_3, \ldots$

$m_1, m_2, m_3 \ldots$



greater mass concentration



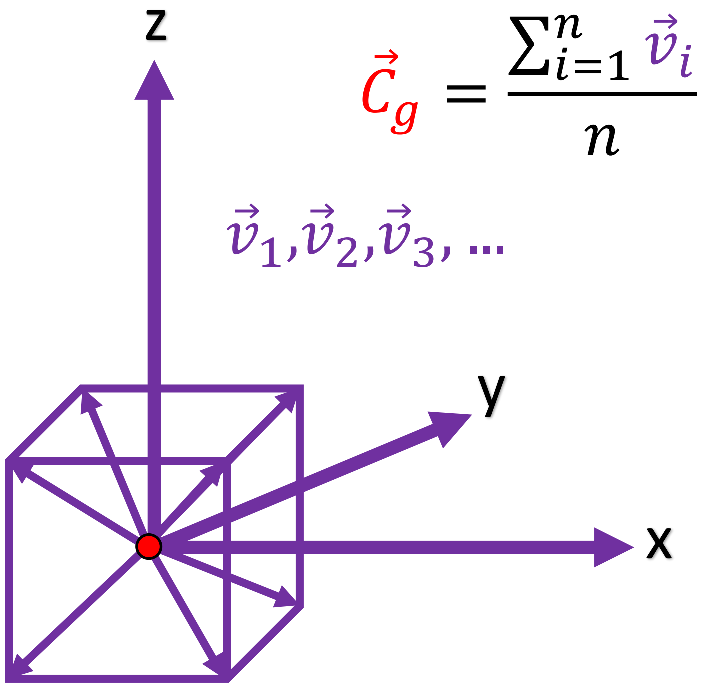
Model

2. Physics body model
    - geometric center (centroid)
    - center of mass

Origin of a local coordinate system should always be precomputed against $\vec{C}_g$ (or $\vec{C}_m$ if exists) so that it becomes the pivot point for all the transformations!

$$\vec{C}_g = \frac{\sum_{i=1}^{n} \vec{v}_i}{n}$$

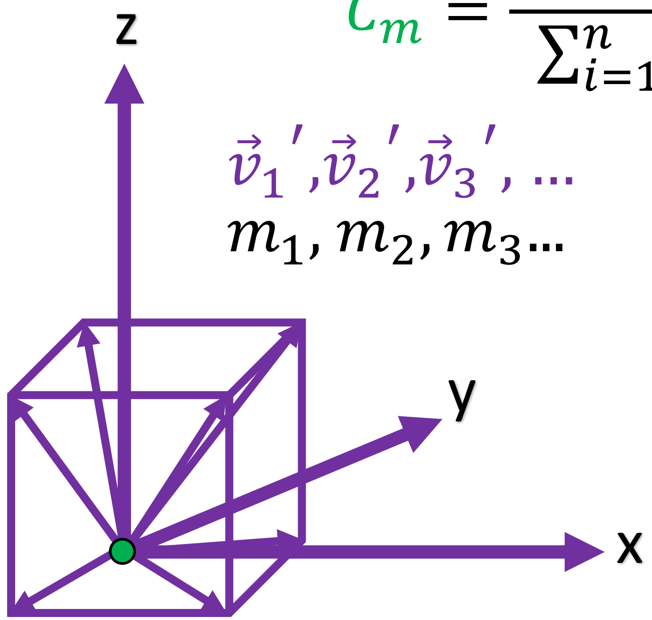
$\vec{v}_1, \vec{v}_2, \vec{v}_3, \ldots$



$$\vec{C}_m = \frac{\sum_{i=1}^{n} \vec{v}_i m_i}{\sum_{i=1}^{n} m_i}$$

$\vec{v}_1{}', \vec{v}_2{}', \vec{v}_3{}', \ldots$

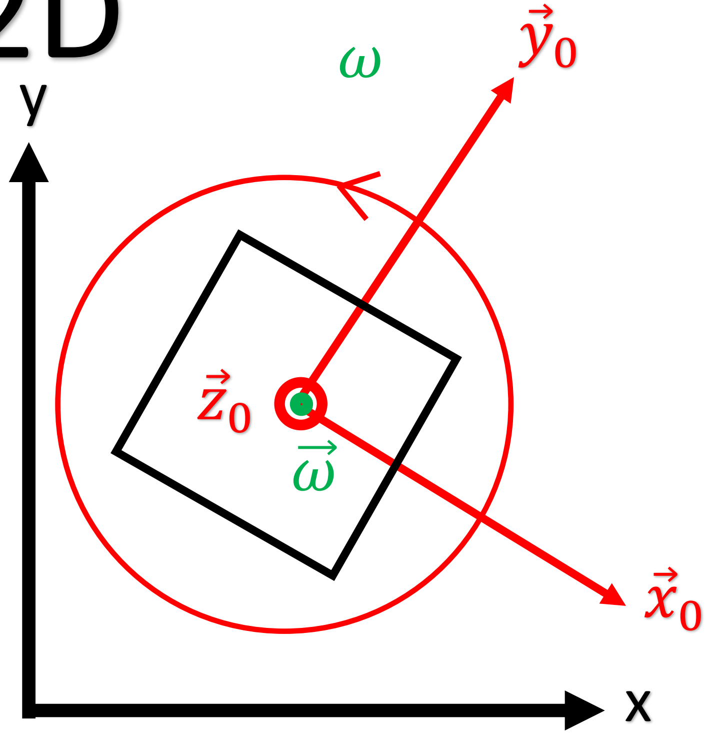
$m_1, m_2, m_3 \ldots$



Model

2. Physics body model
    VI. current angular velocity

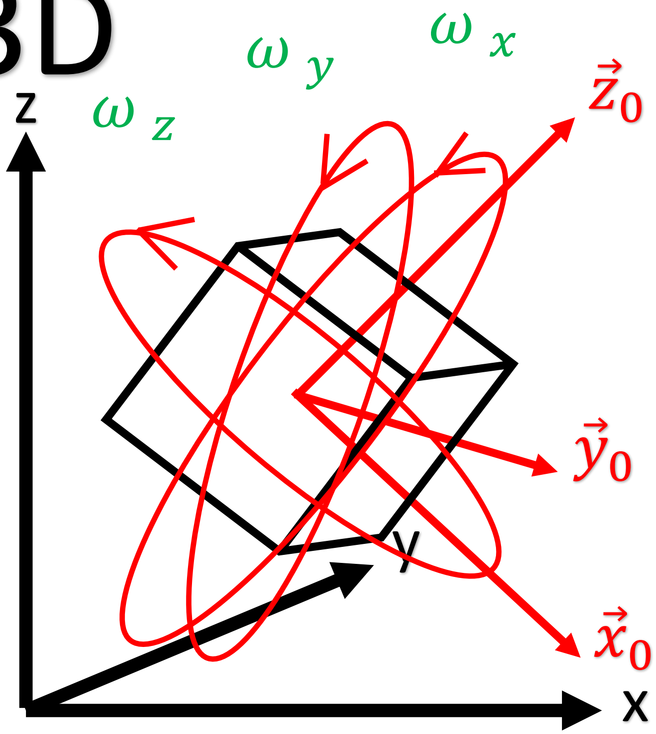
2D



$$\vec{\omega} = \begin{bmatrix} 0 \\ 0 \\ \omega_z \end{bmatrix} = \omega$$

scalar!

3D



$$\vec{\omega} = \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}$$

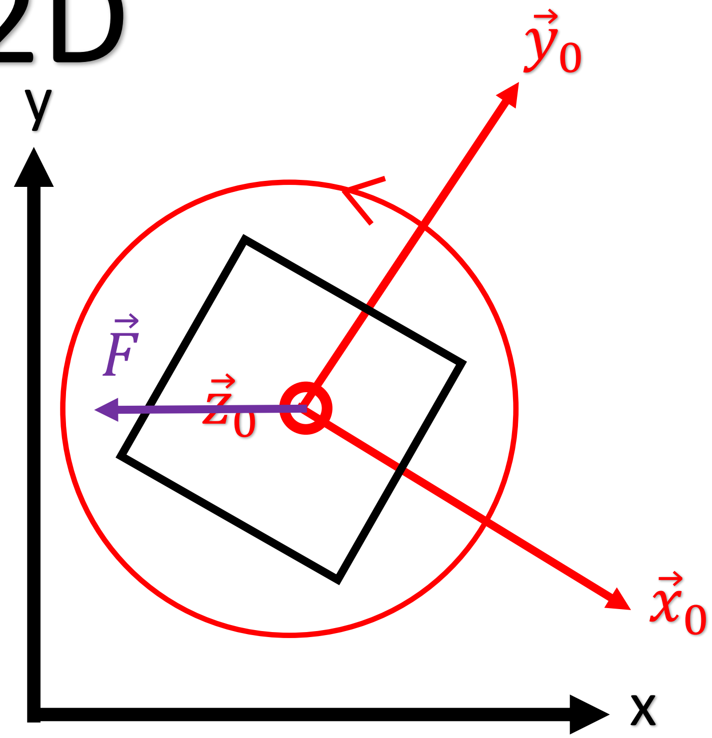
vector!

Model

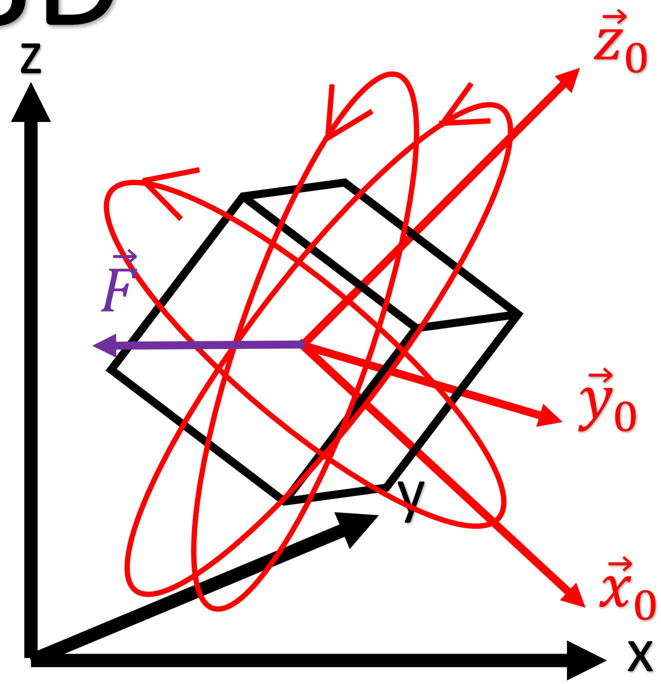
2. Physics body model

   VIII. current total linear force

2D



3D



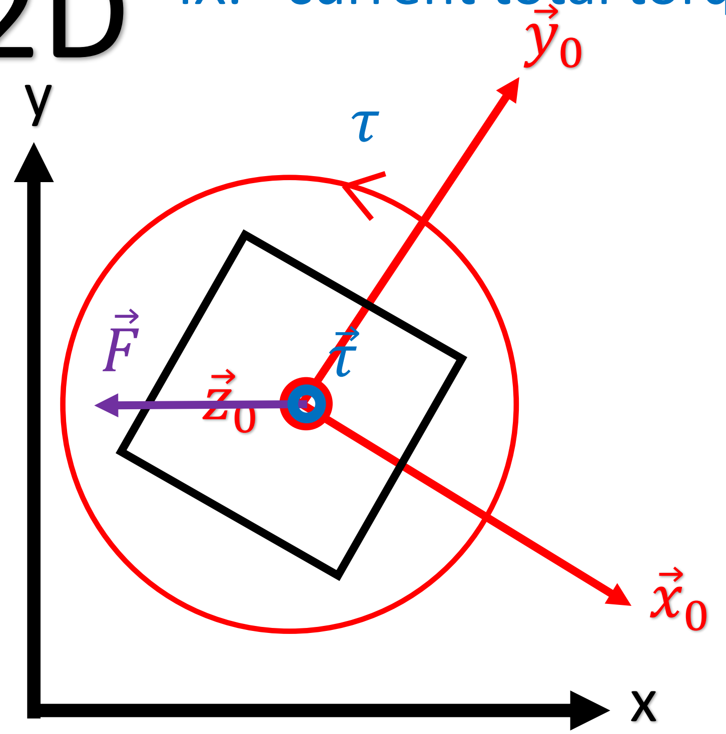
Model

2. Physics body model

    VIII. current total linear force

    IX. current total torque

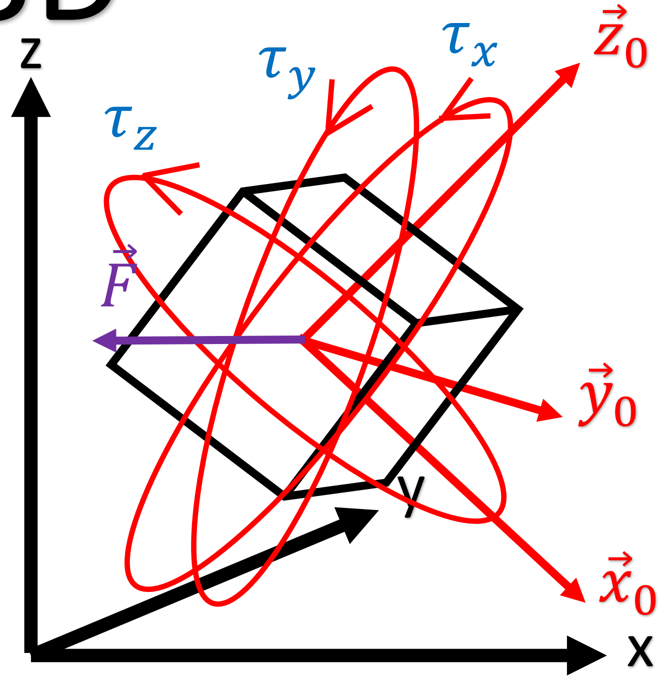
2D



$$\vec{\tau} = \begin{bmatrix} 0 \\ 0 \\ \tau_z \end{bmatrix} = \tau$$

scalar!

3D



$$\vec{\tau} = \begin{bmatrix} \tau_x \\ \tau_y \\ \tau_z \end{bmatrix}$$

vector!

Model

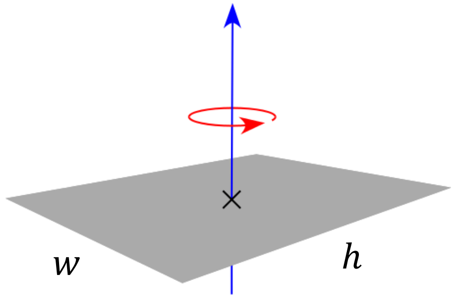
2. Physics body model
   X. mass

Linear motion	Rotational motion
2D/3D	
mass	
m	

Model

2. Physics body model
   X. mass
   XI. moment of inertia

Moments of inertia in the table are given for rotation around $\vec{C}_g$

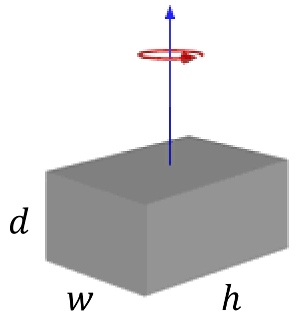


Linear motion	Rotational motion	
2D/3D	2D	
mass	moment of inertia (scalar)	
m	circle: $I = \frac{mr^2}{2}$ rectangle: $I = \frac{m}{12}(w^2 + h^2)$ ⋮	

Model

2. Physics body model
    X. mass
    XI. tensor of inertia

Tensors of inertia in the table are given for rotation around $\vec{C}_g$



Linear motion	Rotational motion	
2D/3D	**2D**	**3D**
mass	moment of inertia (scalar)	tensor of inertia (matrix)
m	circle: $I = \frac{mr^2}{2}$	sphere: $I = \begin{bmatrix} \frac{2}{5}mr^2 & 0 & 0 \\ 0 & \frac{2}{5}mr^2 & 0 \\ 0 & 0 & \frac{2}{5}mr^2 \end{bmatrix}$
	rectangle: $I = \frac{m}{12}(w^2 + h^2)$	prism: $I = \begin{bmatrix} \frac{\mathrm{m}}{12}(h^2 + d^2) & 0 & 0 \\ 0 & \frac{\mathrm{m}}{12}(w^2 + d^2) & 0 \\ 0 & 0 & \frac{\mathrm{m}}{12}(w^2 + h^2) \end{bmatrix}$
	⋮	⋮

Model

2. Physics body model

Linear motion	Rotational motion	
2D/3D	**2D**	**3D**
geometry: $\vec{v}_1, \vec{v}_2, \vec{v}_3, \ldots$ $[m]$	**geometry:** $\vec{v}_1, \vec{v}_2, \vec{v}_3, \ldots$ $[m]$	
position: $\vec{p}$ $[m]$	**angle:** θ $[rad]$	**orientation:** $\vec{\theta} = \begin{bmatrix} \theta_x \\ \theta_y \\ \theta_z \end{bmatrix}$ $[rad]$
velocity: $\vec{v}$ $\left[\frac{m}{s}\right]$	**angular velocity:** ω $\left[\frac{rad}{s}\right]$	**angular velocity:** $\vec{\omega} = \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}$ $\left[\frac{rad}{s}\right]$
force: $\vec{F}$ $[N]$	**torque:** τ $[Nm]$	**torque:** $\vec{\tau} = \begin{bmatrix} \tau_x \\ \tau_y \\ \tau_z \end{bmatrix}$ $[Nm]$
mass: m $[kg]$	**moment of inertia:** I $\left[\frac{kg}{m^2}\right]$	**tensor of inertia:** $I = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{bmatrix}$ $\left[\frac{kg}{m^2}\right]$

Kinematics

How to update the state of a physical system in discrete time steps?

Kinematics

Given initial conditions:

Linear motion	Rotational motion	
2D/3D	**2D**	**3D**
initial position: $\vec{p}_0$	initial angle: θ_0	initial orientation: $\vec{\theta}_0$
initial velocity $\vec{v}_0$	initial angular velocity ω_0	initial angular velocity: $\vec{\omega}_0$

Find:

Linear motion	Rotational motion	
2D/3D	**2D**	**3D**
position: $\vec{p}(t) = ?$	angle: $\theta(t) = ?$	orientation: $\overrightarrow{\theta}(t) = ?$

Kinematics

- exact solution (assuming constant $\vec{F}$):

$$\vec{p}(t) = \vec{p}_0 + \vec{v}_0 t + \frac{1}{2}\vec{a}t^2$$

$$\vec{p}(t) = \vec{p}_0 + \vec{v}_0 t + \frac{1}{2}\frac{\vec{F}}{m}t^2$$

- this form can’t describe complex motion
- this form can’t describe interactive motion (affected by user interaction)

Back to square #1! Where does this solution come from?

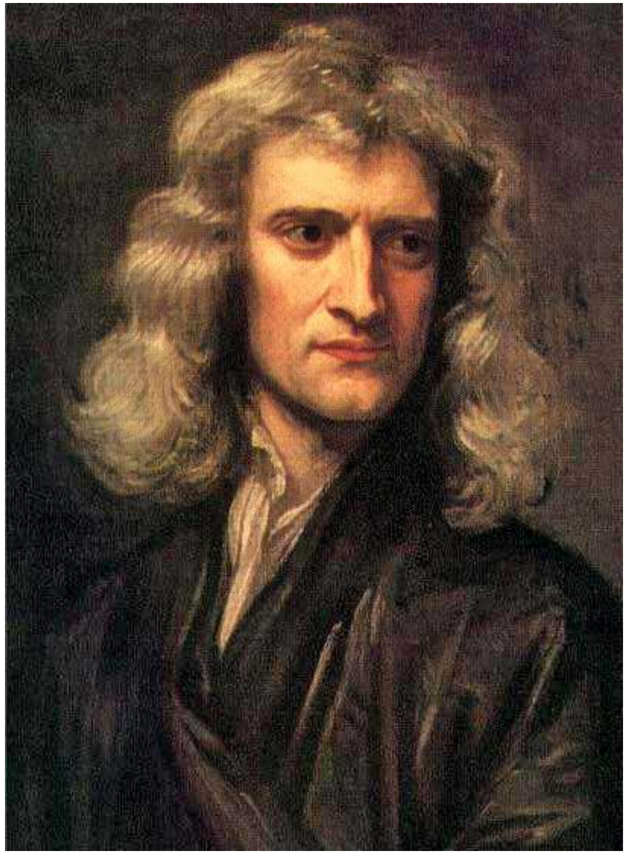
Kinematics

- 2nd Newton's Law of Motion:

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

$$\vec{a} = \frac{\vec{F}}{m}$$

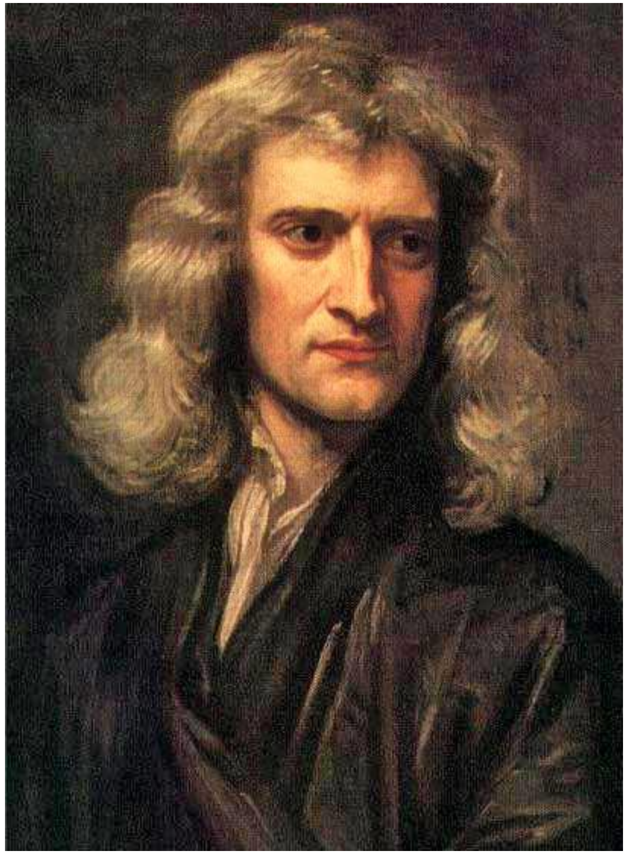
$$\frac{d^2\vec{p}(t)}{dt^2} = \frac{\vec{F}}{m}$$



2nd order differential equation!

Kinematics

- 2nd Newton's Law of Motion:



$$\frac{d^2\vec{p}(t)}{dt^2} = \frac{\vec{F}(t)}{m}$$

What is the solution assuming variable $\vec{F}(t)$?

It depends on $\vec{F}(t)$, and it further depends on user interaction!

Kinematics

At any given moment in time $(t + \ \Delta t)$ a function can be expanded into Taylor series:

$$f(t+\Delta t) = f(t) + \Delta t \frac{df(t)}{dt} + \frac{\Delta t^2}{2}\frac{d^2 f(t)}{dt^2} + \cdots + \frac{\Delta t^n}{n!}\frac{d^n f(t)}{dt^n}$$

This form is suitable for finding an approximate numerical solution!

Kinematics

- Euler's Method

$$f(t + \Delta t) = f(t) + \Delta t \frac{df(t)}{dt} \cancel{+ \frac{\Delta t^2}{2} \frac{d^2 f(t)}{dt^2} + \cdots + \frac{\Delta t^n}{n!} \frac{d^n f(t)}{dt^n}}$$

$$f(t + \Delta t) \approx f(t) + \Delta t \frac{df(t)}{dt}$$

Kinematics

- Euler's Method

Position function:

$$\vec{p}(t+\Delta t)=\vec{p}(t)+\Delta t\,\frac{d\vec{p}(t)}{dt}$$

Kinematics

- Euler's Method

Differentiate both sides once:

$$\vec{p}(t + \Delta t) = \vec{p}(t) + \Delta t \frac{d\vec{p}(t)}{dt}$$

$$\frac{d\vec{p}(t + \Delta t)}{dt} = \frac{d\vec{p}(t)}{dt} + \Delta t \frac{d^2\vec{p}(t)}{dt^2}$$

Kinematics

- Euler's Method

substitution: $\frac{d\vec{p}(t)}{dt} = \vec{v}(t)$

$$\vec{p}(t + \Delta t) = \vec{p}(t) + \Delta t \vec{v}(t)$$

$$\vec{v}(t + \Delta t) = \vec{v}(t) + \Delta t \frac{\vec{F}(t)}{m}$$

Kinematics

- Euler's Method

Iterative form:

$$\vec{p}_i = \vec{p}_{i-1} + \Delta t \vec{v}_{i-1}$$

$$\vec{v}_i = \vec{v}_{i-1} + \Delta t \frac{\vec{F}_{i-1}}{m}$$

Kinematics

- Euler's Method

$$\vec{p}_i = \vec{p}_{i-1} + \Delta t \vec{v}_{i-1}$$

$$\vec{v}_i = \vec{v}_{i-1} + \Delta t \frac{\vec{F}_{i-1}}{m}$$

- Plain Euler's method uses old velocity for finding new position.

Kinematics

- Semi-implicit Euler's Method

$$\vec{v}_i = \vec{v}_{i-1} + \Delta t \frac{\vec{F}_{i-1}}{m}$$

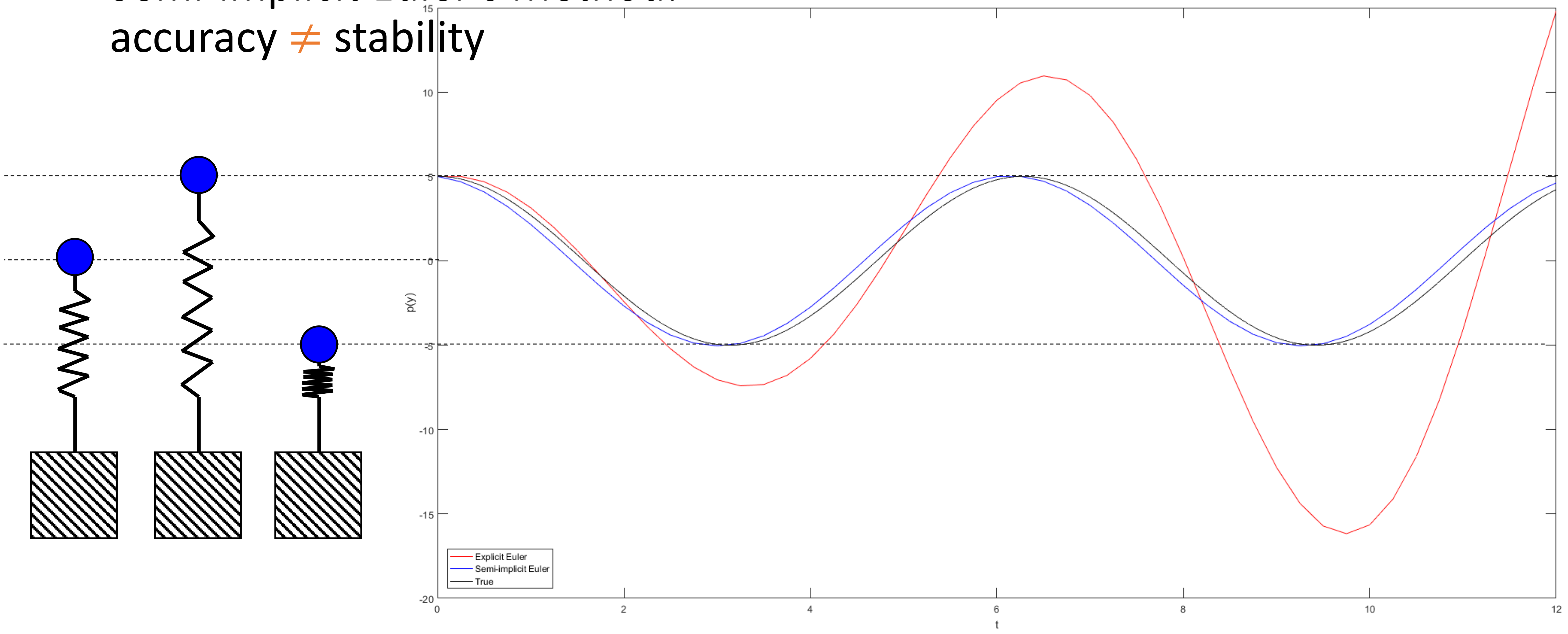
$$\vec{p}_i = \vec{p}_{i-1} + \Delta t \vec{v}_i$$

+ Calculates new velocity first, and then uses it to find new position.
+ Offers improved method stability without additional calculations!

Kinematics

Neptune

- Semi-implicit Euler's Method:
  accuracy ≠ stability



Kinematics

- Semi-implicit Euler's Method (integration)

0
$\vec{v}_0$
$\vec{p}_0$

0
$\vec{\omega}_0$
$\vec{\theta}_0$

Kinematics

- Semi-implicit Euler's Method (integration)

$\mathbf{0}$	$\mathbf{0} + \Delta t$
$\vec{v}_0$	$\vec{v}_1 = \vec{v}_0 + \Delta t \frac{\vec{F}_0}{m}$
$\vec{p}_0$	$\vec{p}_1 = \vec{p}_0 + \Delta t \vec{v}_1$

$\mathbf{0}$	$\mathbf{0} + \Delta t$
$\vec{\omega}_0$	$\vec{\omega}_1 = \vec{\omega}_0 + \Delta t \frac{\vec{\tau}_0}{I}$
$\vec{\theta}_0$	$\vec{\theta}_1 = \vec{\theta}_0 + \Delta t \vec{\omega}_1$

Kinematics

- Semi-implicit Euler's Method (integration)

$\mathbf{0}$	$\mathbf{0}+\Delta t$	$\mathbf{0}+\mathbf{2}\Delta t$
$\vec{v}_0$	$\vec{v}_1 = \vec{v}_0 + \Delta t \frac{\vec{F}_0}{m}$	$\vec{v}_2 = \vec{v}_1 + \Delta t \frac{\vec{F}_1}{m}$
$\vec{p}_0$	$\vec{p}_1 = \vec{p}_0 + \Delta t \vec{v}_1$	$\vec{p}_2 = \vec{p}_1 + \Delta t \vec{v}_2$

$\mathbf{0}$	$\mathbf{0}+\Delta t$	$\mathbf{0}+\mathbf{2}\Delta t$
$\vec{\omega}_0$	$\vec{\omega}_1 = \vec{\omega}_0 + \Delta t \frac{\vec{\tau}_0}{I}$	$\vec{\omega}_2 = \vec{\omega}_1 + \Delta t \frac{\vec{\tau}_1}{I}$
$\vec{\theta}_0$	$\vec{\theta}_1 = \vec{\theta}_0 + \Delta t \vec{\omega}_1$	$\vec{\theta}_2 = \vec{\theta}_1 + \Delta t \vec{\omega}_2$

Kinematics

- Semi-implicit Euler's Method (integration)

$\mathbf{0}$	$\mathbf{0 + \Delta}t$	$\mathbf{0 + 2\Delta}t$	$\mathbf{0 + i\Delta}t$
$\vec{v}_0$	$\vec{v}_1 = \vec{v}_0 + \Delta t \frac{\vec{F}_0}{m}$	$\vec{v}_2 = \vec{v}_1 + \Delta t \frac{\vec{F}_1}{m}$	$\vec{v}_i = \vec{v}_{i-1} + \Delta t \frac{\vec{F}_{i-1}}{m}$
$\vec{p}_0$	$\vec{p}_1 = \vec{p}_0 + \Delta t \vec{v}_1$	$\vec{p}_2 = \vec{p}_1 + \Delta t \vec{v}_2$	$\vec{p}_i = \vec{p}_{i-1} + \Delta t \vec{v}_i$

$\mathbf{0}$	$\mathbf{0 + \Delta}t$	$\mathbf{0 + 2\Delta}t$	$\mathbf{0 + i\Delta}t$
$\vec{\omega}_0$	$\vec{\omega}_1 = \vec{\omega}_0 + \Delta t \frac{\vec{\tau}_0}{I}$	$\vec{\omega}_2 = \vec{\omega}_1 + \Delta t \frac{\vec{\tau}_1}{I}$	$\vec{\omega}_i = \vec{\omega}_{i-1} + \Delta t \frac{\vec{\tau}_{i-1}}{I}$
$\vec{\theta}_0$	$\vec{\theta}_1 = \vec{\theta}_0 + \Delta t \vec{\omega}_1$	$\vec{\theta}_2 = \vec{\theta}_1 + \Delta t \vec{\omega}_2$	$\vec{\theta}_i = \vec{\theta}_{i-1} + \Delta t \vec{\omega}_i$

Kinematics

- Engine pipeline:

Apply external forces

Integrate

Render

Sleep

Kinematics

$$\vec{v}_i = \vec{v}_{i-1} + \Delta t \frac{\vec{F}_{i-1}}{m}$$

$$\vec{p}_i = \vec{p}_{i-1} + \Delta t \vec{v}_i$$

$$\vec{\omega}_i = \vec{\omega}_{i-1} + \Delta t \frac{\vec{\tau}_{i-1}}{I}$$

$$\vec{\theta}_i = \vec{\theta}_{i-1} + \Delta t \vec{\omega}_i$$

Which are known and which are unknown variables?

Kinematics

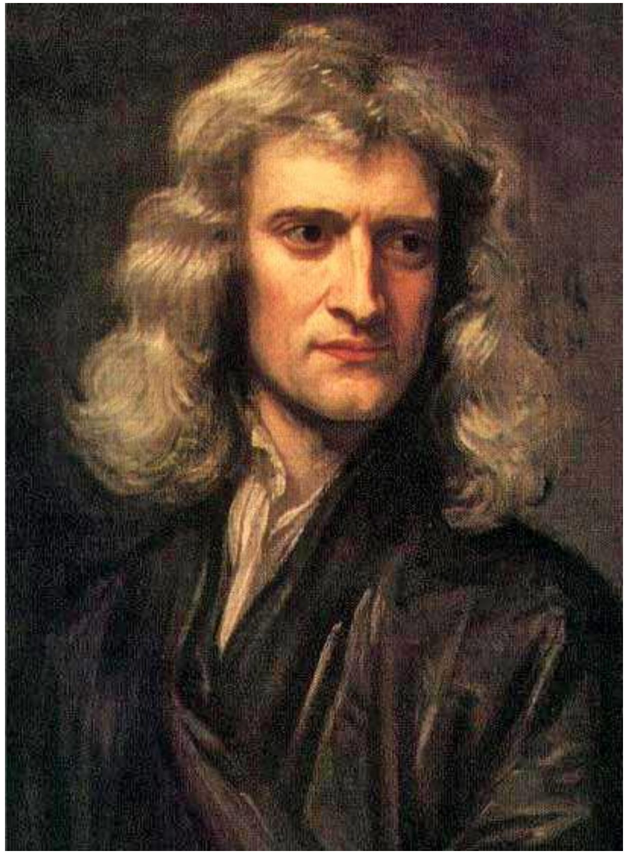
- 1st Newton's Law of Motion:

assuming $\vec{F}_{i-1} = 0$ and $\vec{\tau}_{i-1} = 0$:

a) assuming $\vec{v}_{i-1} \neq 0$ and $\vec{\omega}_{i-1} \neq 0$:

$$\vec{v}_i = \vec{v}_{i-1} + \Delta t \frac{0}{m} = \vec{v}_{i-1} = const.$$

$$\vec{\omega}_i = \vec{\omega}_{i-1} + \Delta t \frac{0}{I} = \vec{\omega}_{i-1} = const.$$



b) assuming $\vec{v}_{i-1} = 0$ and $\vec{\omega}_{i-1} = 0$:

$$\vec{v}_i = 0 + \Delta t \frac{0}{m} = 0 = const.$$

$$\vec{\omega}_i = 0 + \Delta t \frac{0}{I} = 0 = const.$$

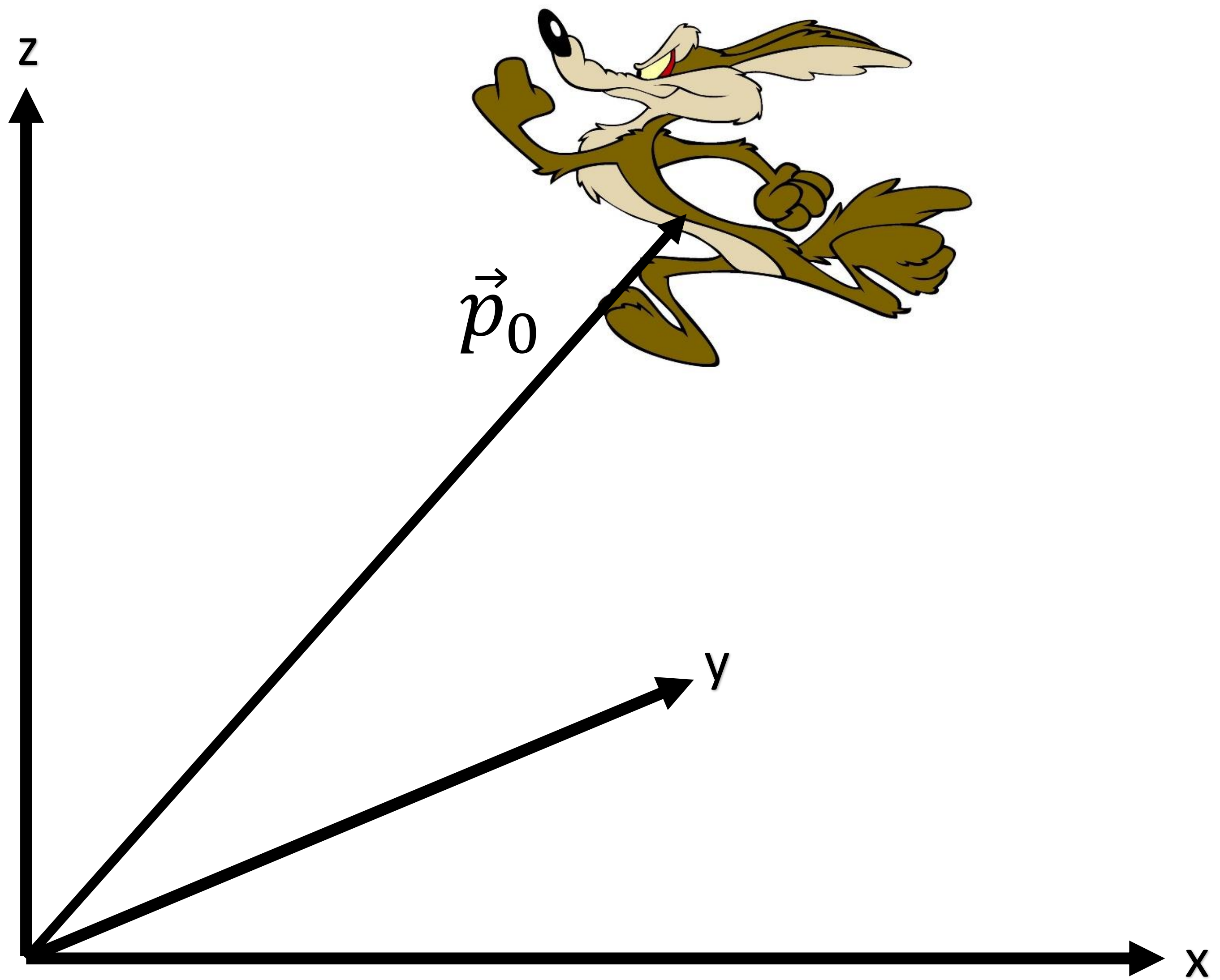
Kinematics

$$\vec{F}_0 = \begin{bmatrix} 0.00 \\ 0.00 \\ 0.00 \end{bmatrix}$$

$$\vec{v}_0 = \begin{bmatrix} 0.00 \\ 0.00 \\ 0.00 \end{bmatrix}$$

$$\vec{p}_0 = \begin{bmatrix} 5.00 \\ 5.00 \\ 5.00 \end{bmatrix}$$

$$t = 0$$



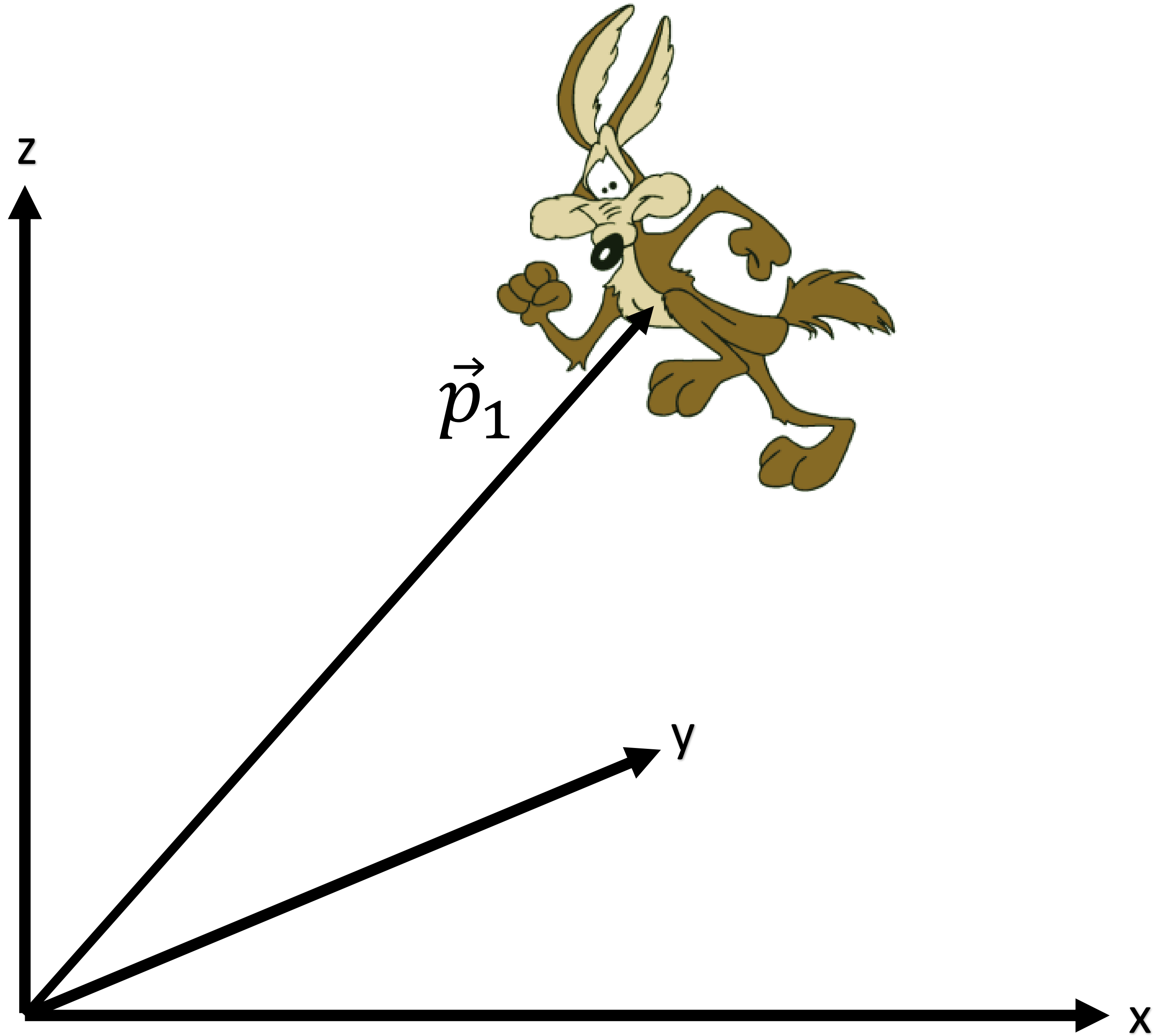
Kinematics

$$\vec{F}_0 = \begin{bmatrix} 0.00 \\ 0.00 \\ 0.00 \end{bmatrix}$$

$$\vec{v}_1 = \begin{bmatrix} 0.00 \\ 0.00 \\ 0.00 \end{bmatrix}$$

$$\vec{p}_1 = \begin{bmatrix} 5.00 \\ 5.00 \\ 5.00 \end{bmatrix}$$

$$t = \Delta t$$



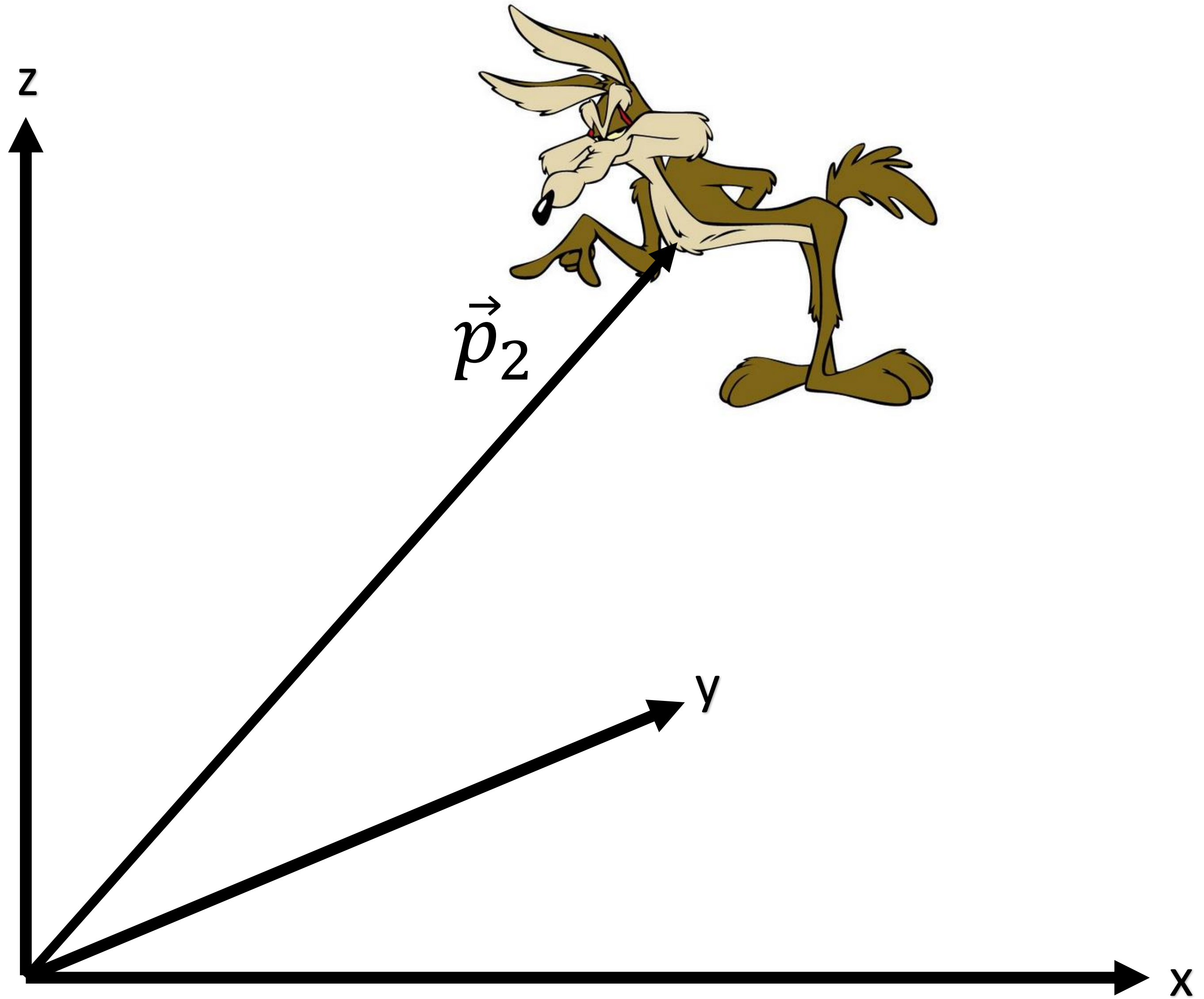
Kinematics

$$\vec{F}_1 = \begin{bmatrix} 0.00 \\ 0.00 \\ 0.00 \end{bmatrix}$$

$$\vec{v}_2 = \begin{bmatrix} 0.00 \\ 0.00 \\ 0.00 \end{bmatrix}$$

$$\vec{p}_2 = \begin{bmatrix} 5.00 \\ 5.00 \\ 5.00 \end{bmatrix}$$

$$t = 2\Delta t$$



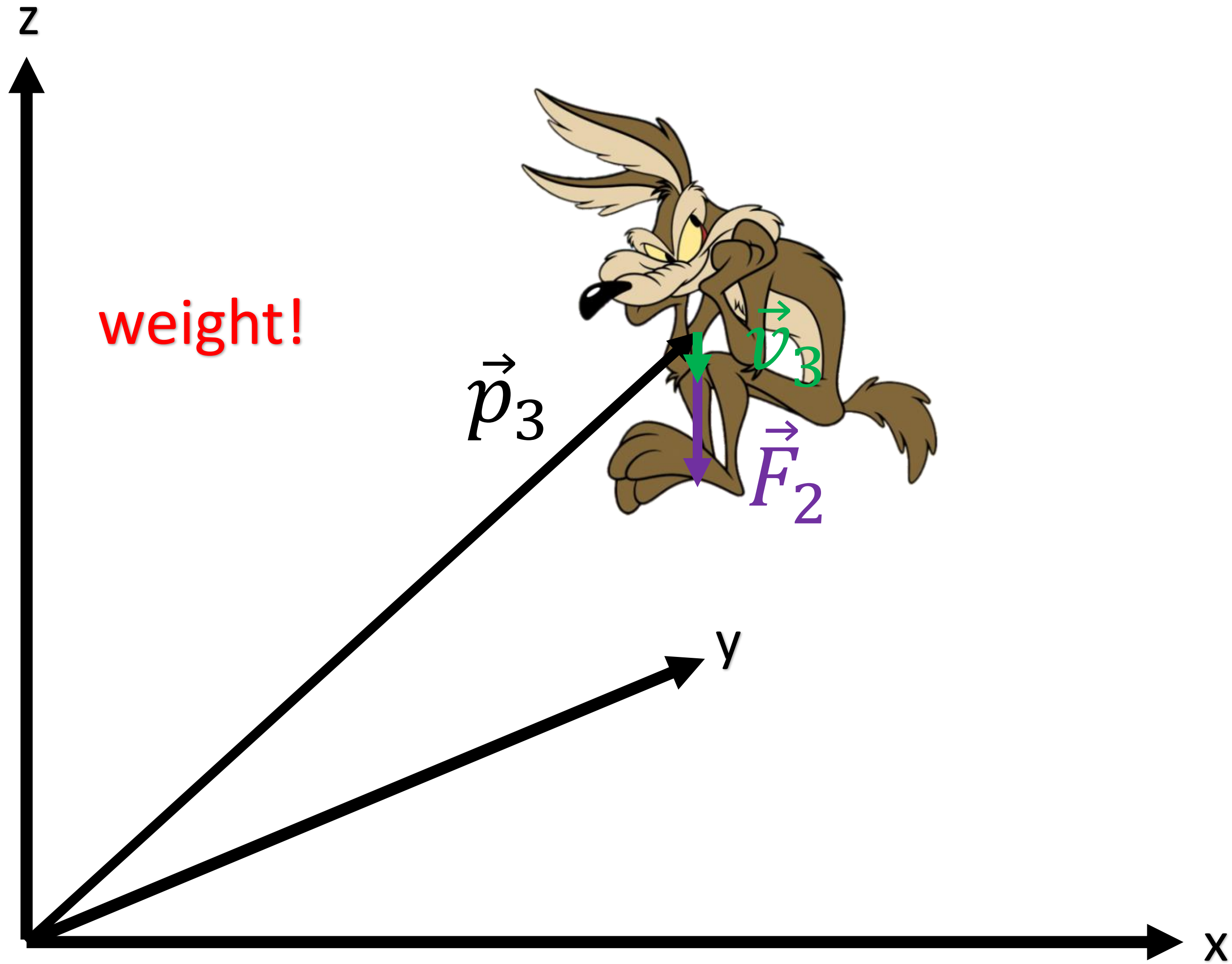
Kinematics

$$\vec{F}_2 = \begin{bmatrix} 0.00 \\ 0.00 \\ -mg \end{bmatrix}$$

$$\vec{v}_3 = \begin{bmatrix} 0.00 \\ 0.00 \\ -2.45 \end{bmatrix}$$

$$\vec{p}_3 = \begin{bmatrix} 5.00 \\ 5.00 \\ 4.38 \end{bmatrix}$$

$$t = 3\Delta t$$



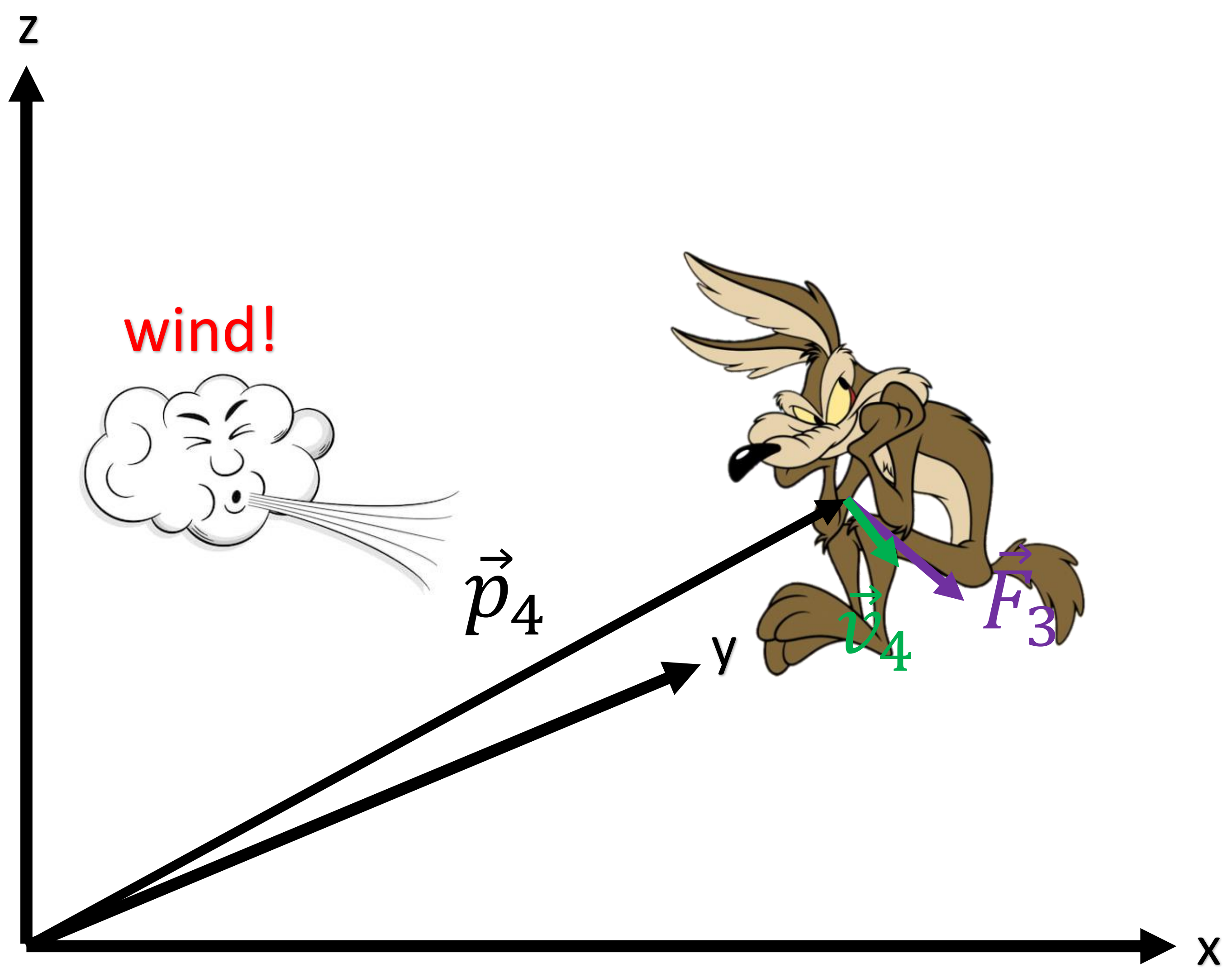
Kinematics

$$\vec{F}_3 = \begin{bmatrix} 10.0 \\ 0.00 \\ -mg \end{bmatrix}$$

$$\vec{v}_4 = \begin{bmatrix} 3.33 \\ 0.00 \\ -4.90 \end{bmatrix}$$

$$\vec{p}_4 = \begin{bmatrix} 6.11 \\ 5.00 \\ 3.15 \end{bmatrix}$$

$$t = 4\Delta t$$



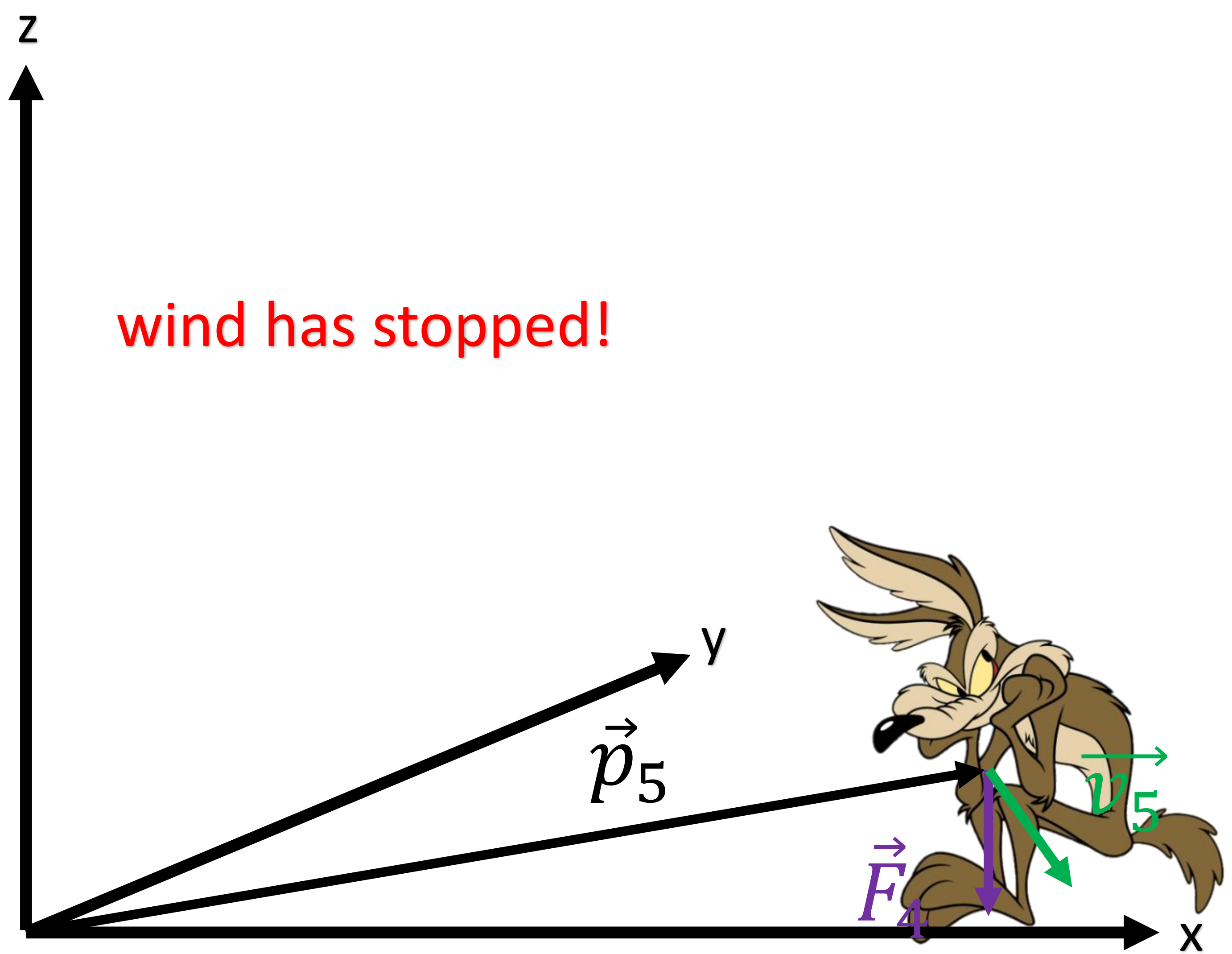
Kinematics

$$\vec{F}_4 = \begin{bmatrix} \color{red}{0.00} \\ 0.00 \\ -mg \end{bmatrix}$$

$$\vec{v}_5 = \begin{bmatrix} 3.33 \\ 0.00 \\ -9.81 \end{bmatrix}$$

$$\vec{p}_5 = \begin{bmatrix} 7.22 \\ 5.00 \\ 1.31 \end{bmatrix}$$

$$t = 5\Delta t$$



Kinematics

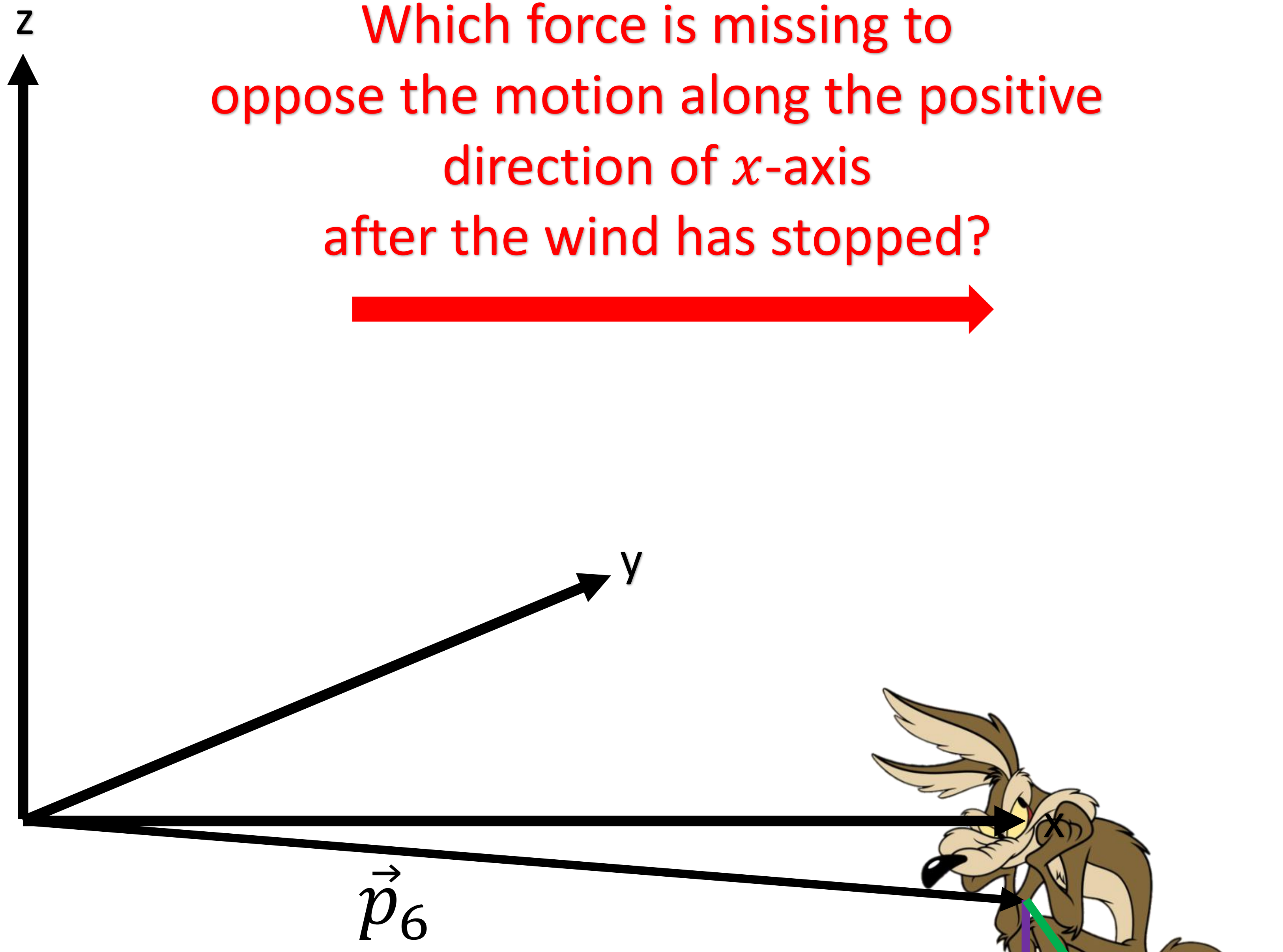
$$\vec{F}_5 = \begin{bmatrix} 0.00 \\ 0.00 \\ -mg \end{bmatrix}$$

$$\vec{v}_6 = \begin{bmatrix} \color{red}{3.33} \\ 0.00 \\ -7.35 \end{bmatrix}$$

$$\vec{p}_6 = \begin{bmatrix} 8.33 \\ 5.00 \\ -1.14 \end{bmatrix}$$

$$t = 6\Delta t$$

Which force is missing to oppose the motion along the positive direction of x-axis after the wind has stopped?



Kinematics

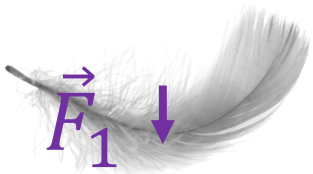
vacuum:

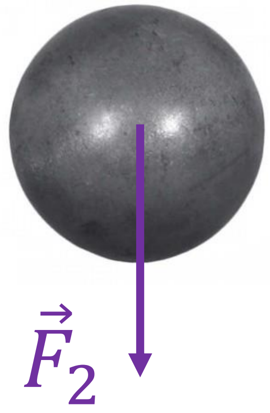
$$\vec{F} = \vec{F}_{weight}$$

(only the force of weight acts on the body)

$$m_1 \ll m_2$$

y





$$v_1 = 0 + \Delta t \frac{m_1 g}{m_1} \qquad v_2 = 0 + \Delta t \frac{m_2 g}{m_2}$$

x

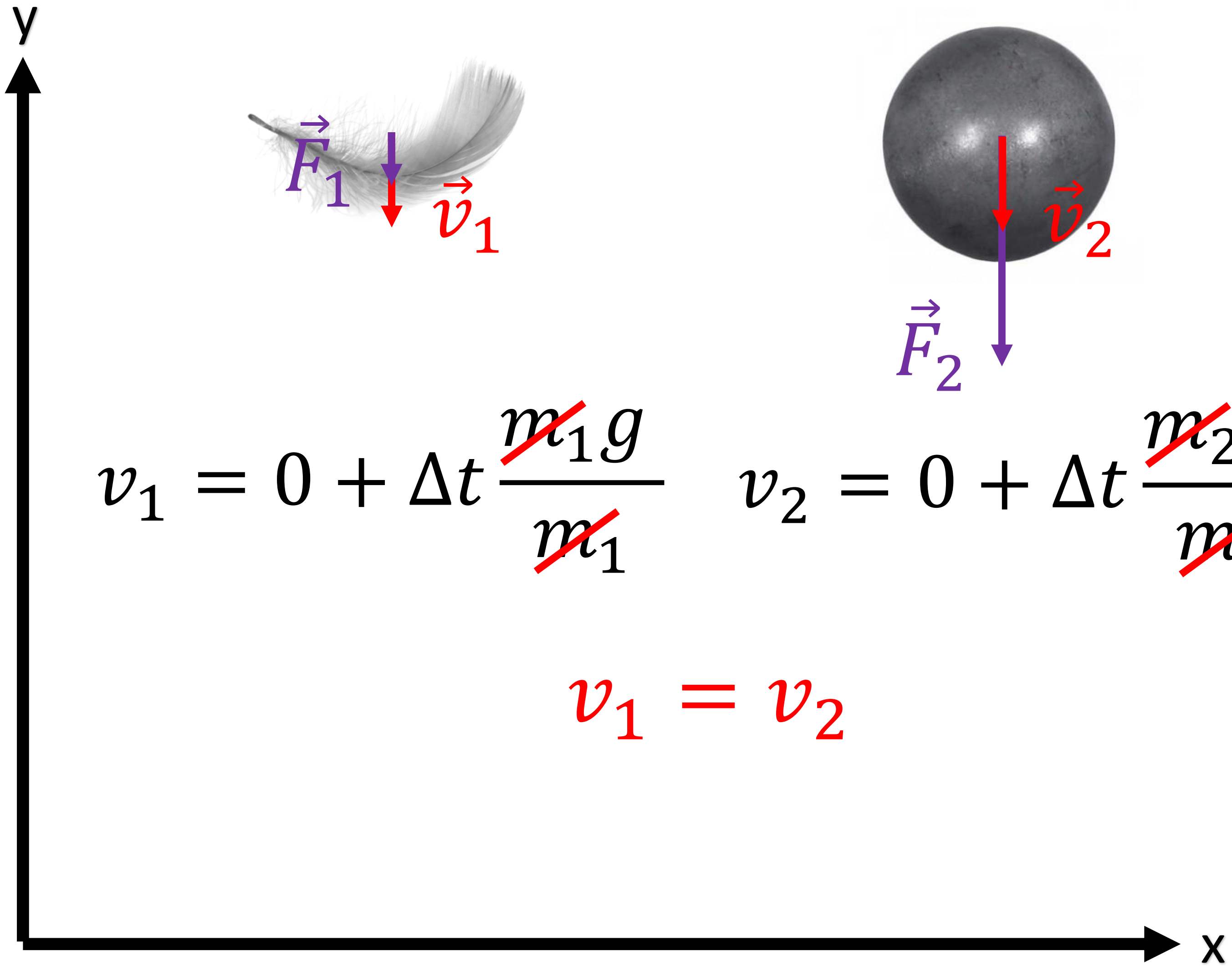
Kinematics

$$m_1 \ll m_2$$

vacuum:

$$\vec{F} = \vec{F}_{weight}$$

(only the force of weight acts on the body)



$$v_1 = 0 + \Delta t \frac{\cancel{m_1} g}{\cancel{m_1}} \qquad v_2 = 0 + \Delta t \frac{\cancel{m_2} g}{\cancel{m_2}}$$

$$v_1 = v_2$$

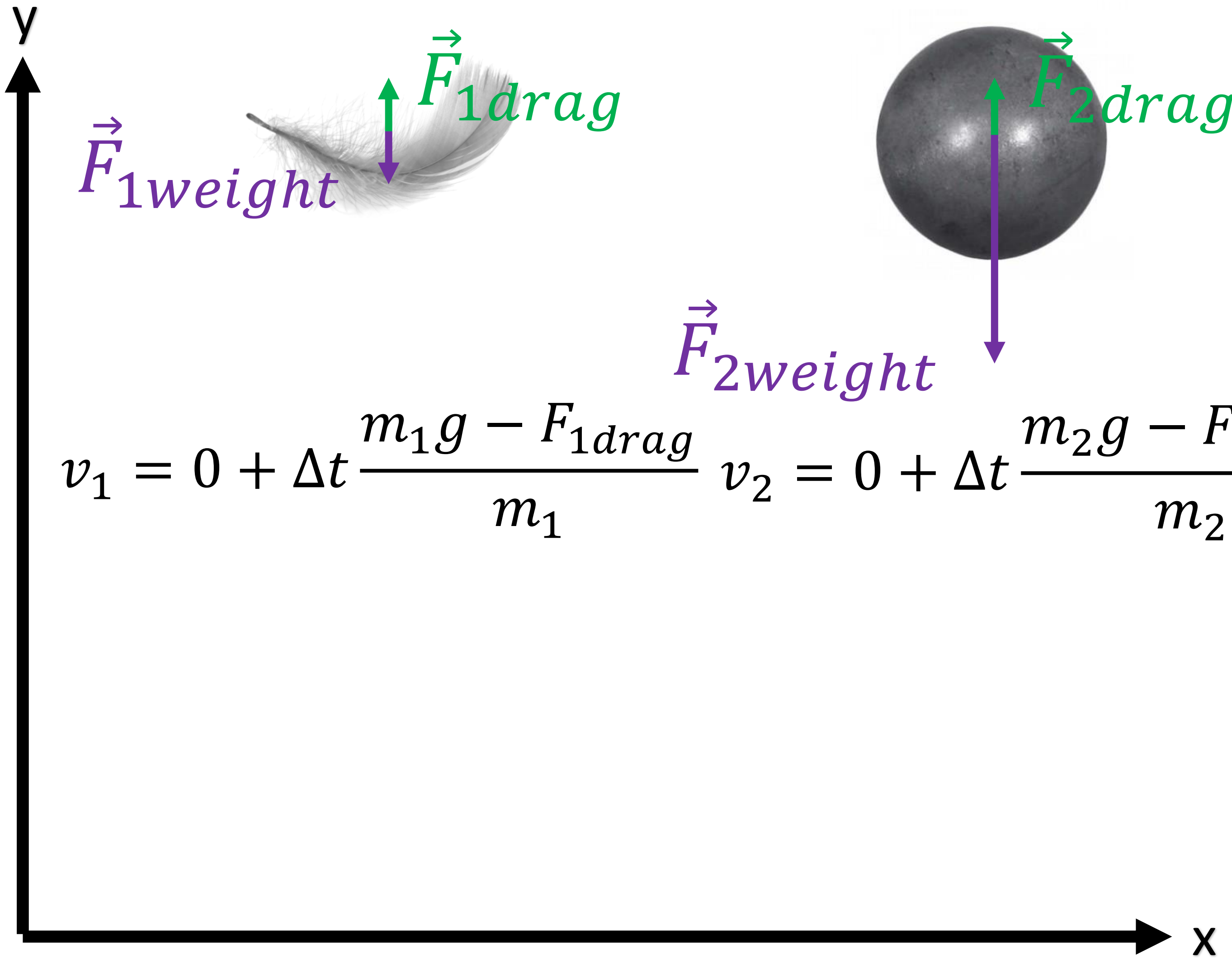
Kinematics

$$m_1 \ll m_2$$

fluid:

$$\vec{F} = \vec{F}_{weight} + \vec{F}_{drag}$$

(weight + fluid drag force)



$$v_1 = 0 + \Delta t \frac{m_1 g - F_{1drag}}{m_1} \quad v_2 = 0 + \Delta t \frac{m_2 g - F_{2drag}}{m_2}$$

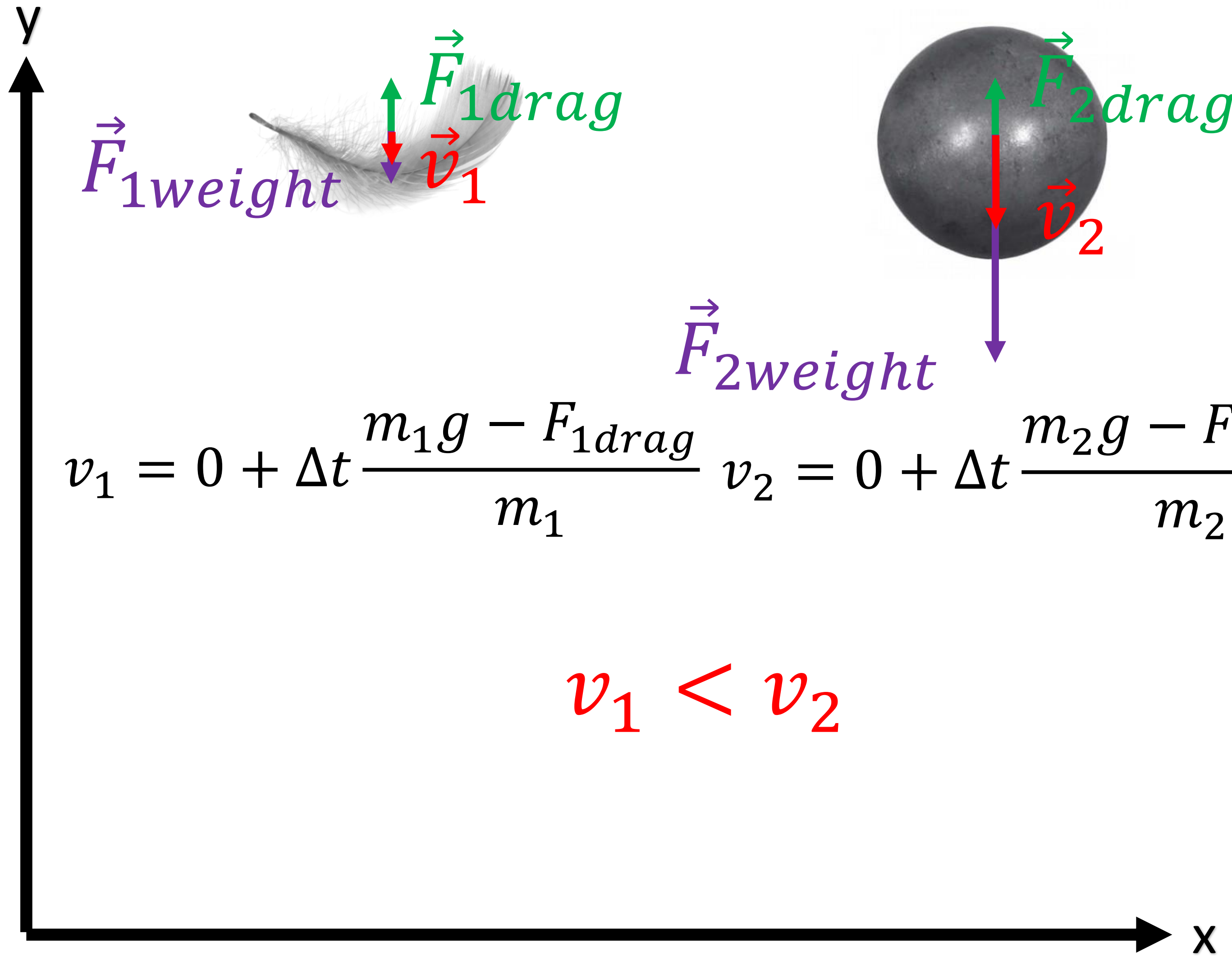
Kinematics

fluid:

$$\vec{F} = \vec{F}_{weight} + \vec{F}_{drag}$$

(weight + fluid drag force)

$$m_1 \ll m_2$$



$$v_1 = 0 + \Delta t \frac{m_1 g - F_{1drag}}{m_1} \qquad v_2 = 0 + \Delta t \frac{m_2 g - F_{2drag}}{m_2}$$

$$v_1 < v_2$$

Kinematics

Two basic types of bodies:

1. Particles
2. Rigid bodies

Complex types of bodies (various combinations of basic types):

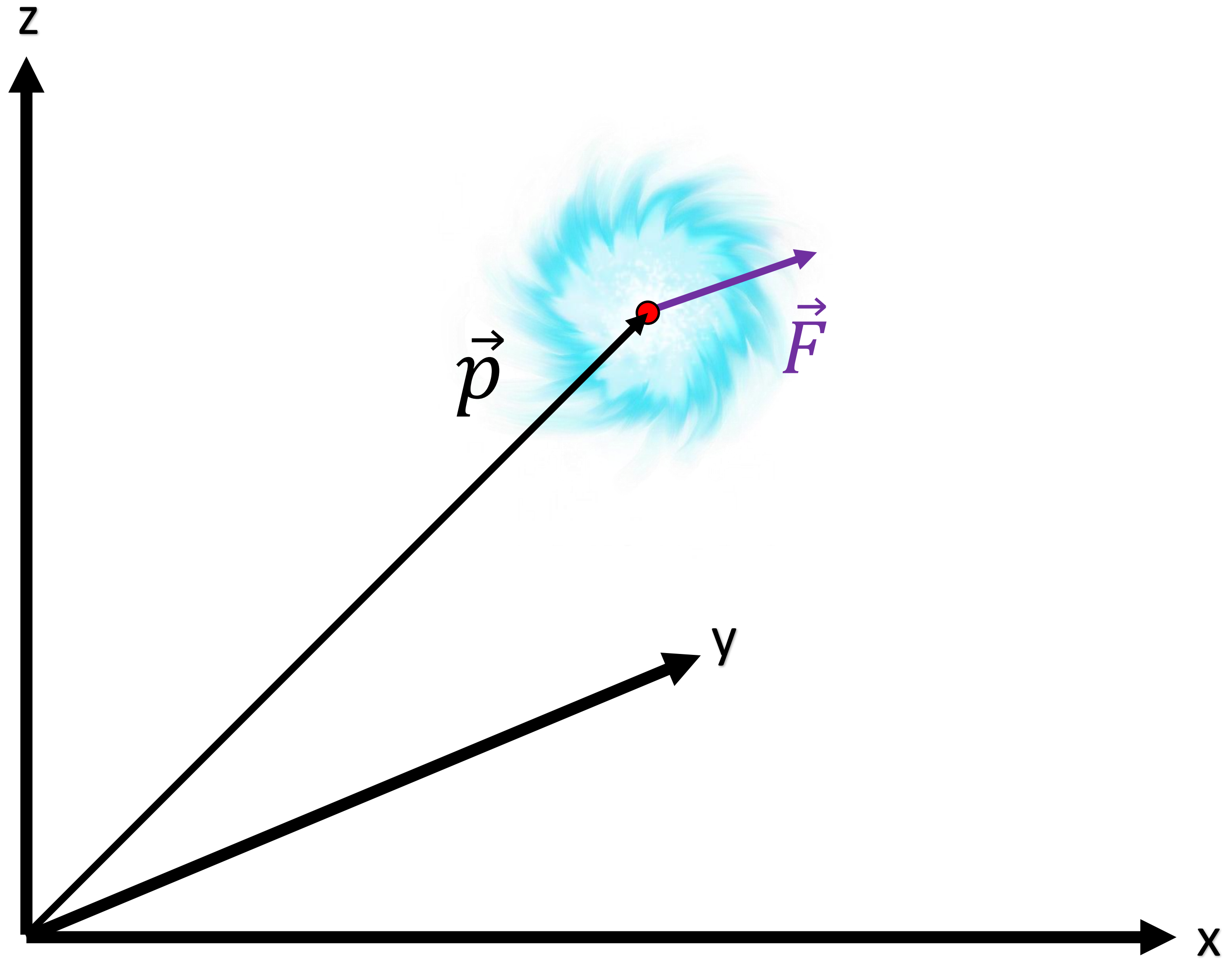
1. Soft bodies
2. Fluids
3. Cloth
4. Ragdolls
5. etc.

Kinematics

- Particle:

+ has no orientation or rotation

+ all external forces act on $\overrightarrow{C_g}$

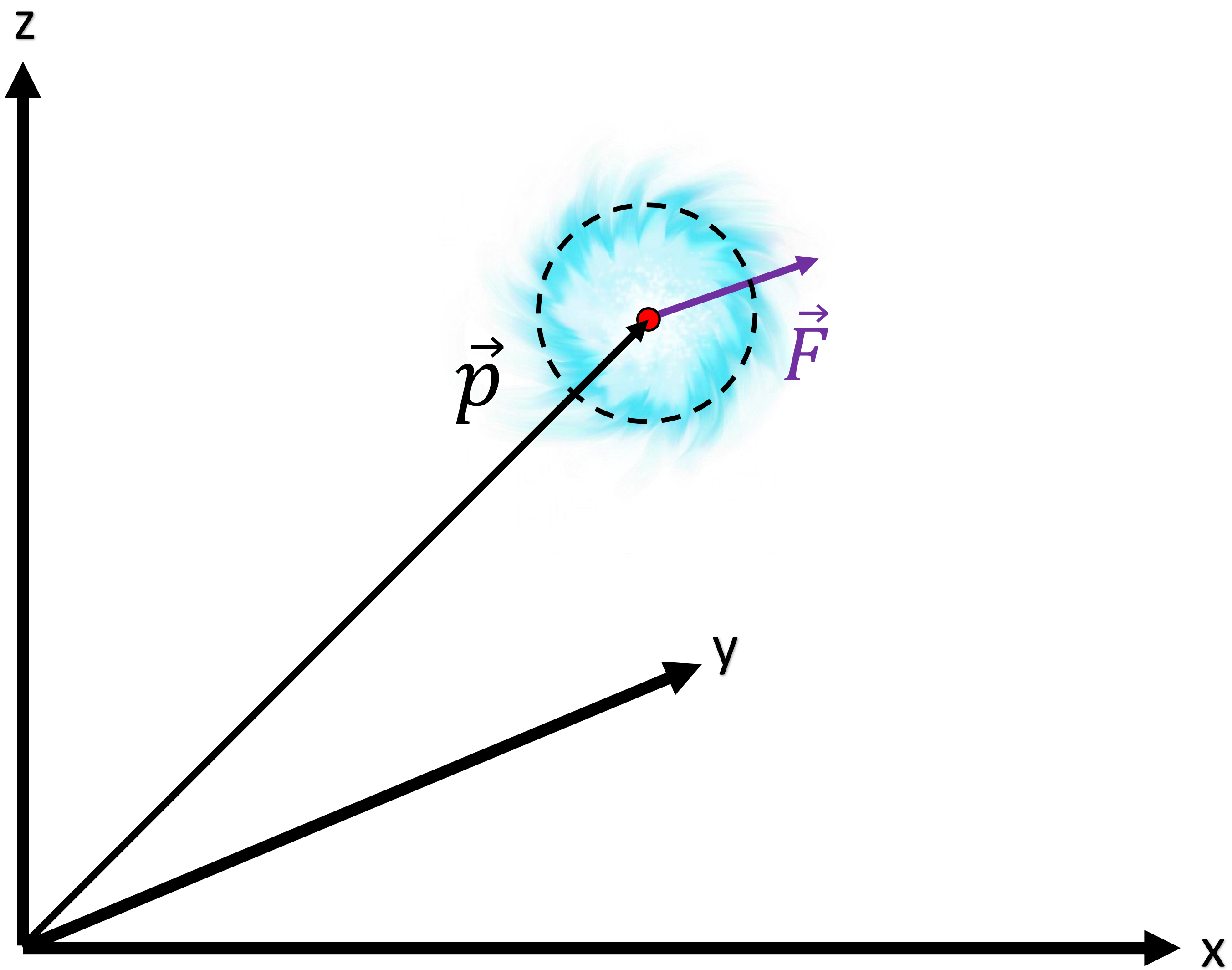


Kinematics

- Particle:

+ has no orientation or rotation
+ all external forces act on $\overrightarrow{C_g}$

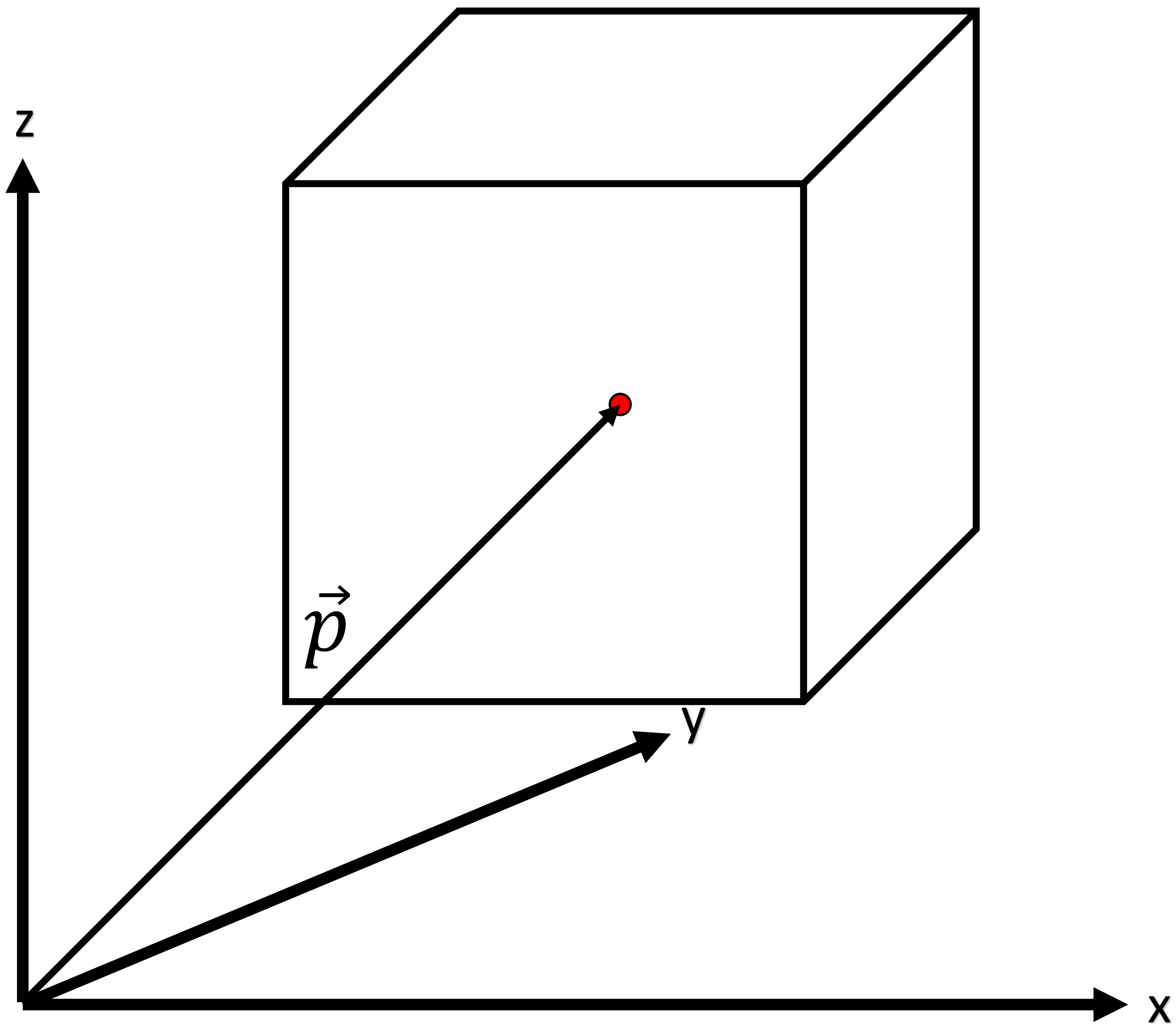
(a symmetrical geometry is usually used for collision detection)



Kinematics

- Rigid body:

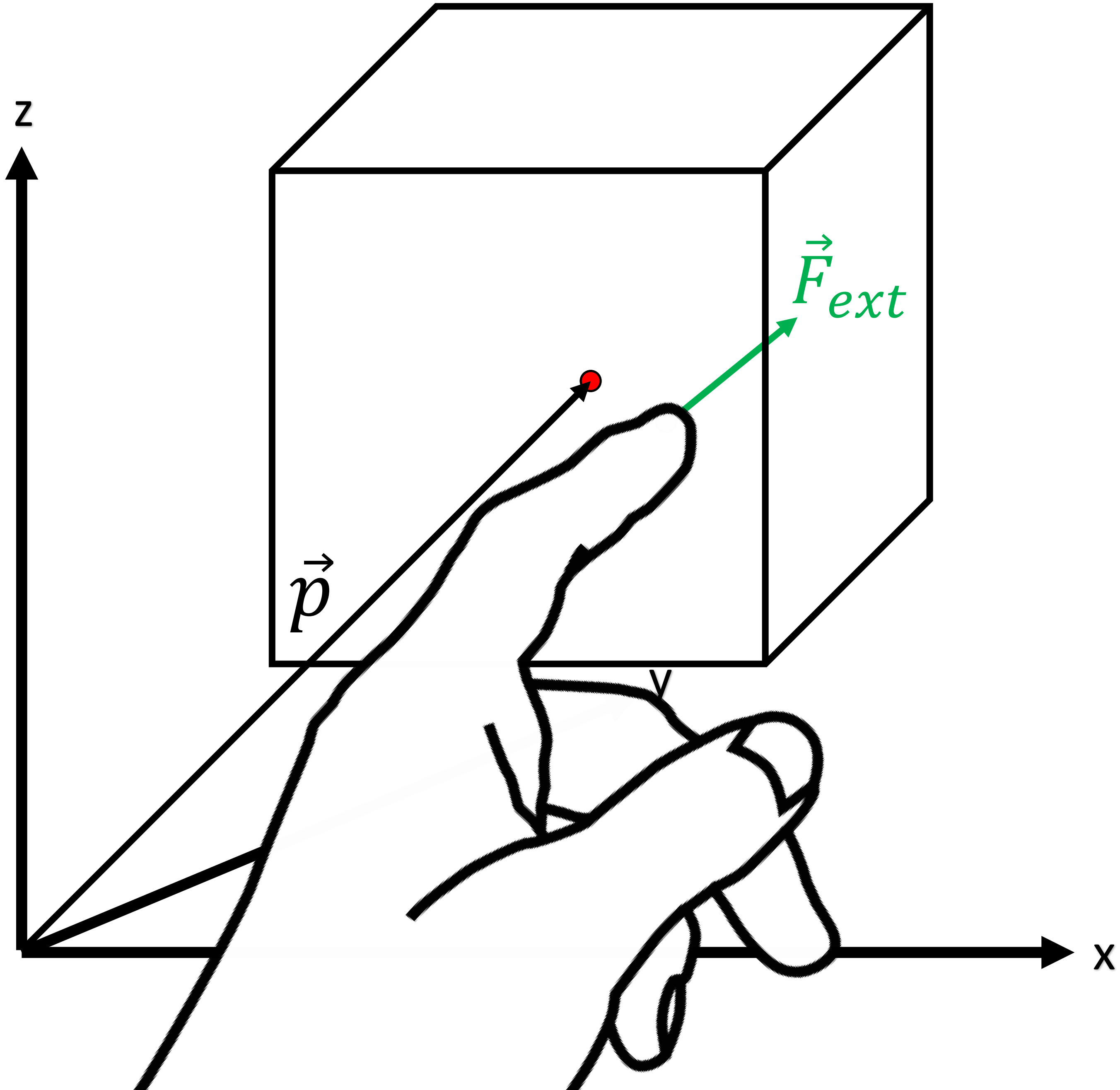
- has orientation and rotation



Kinematics

- Rigid body:

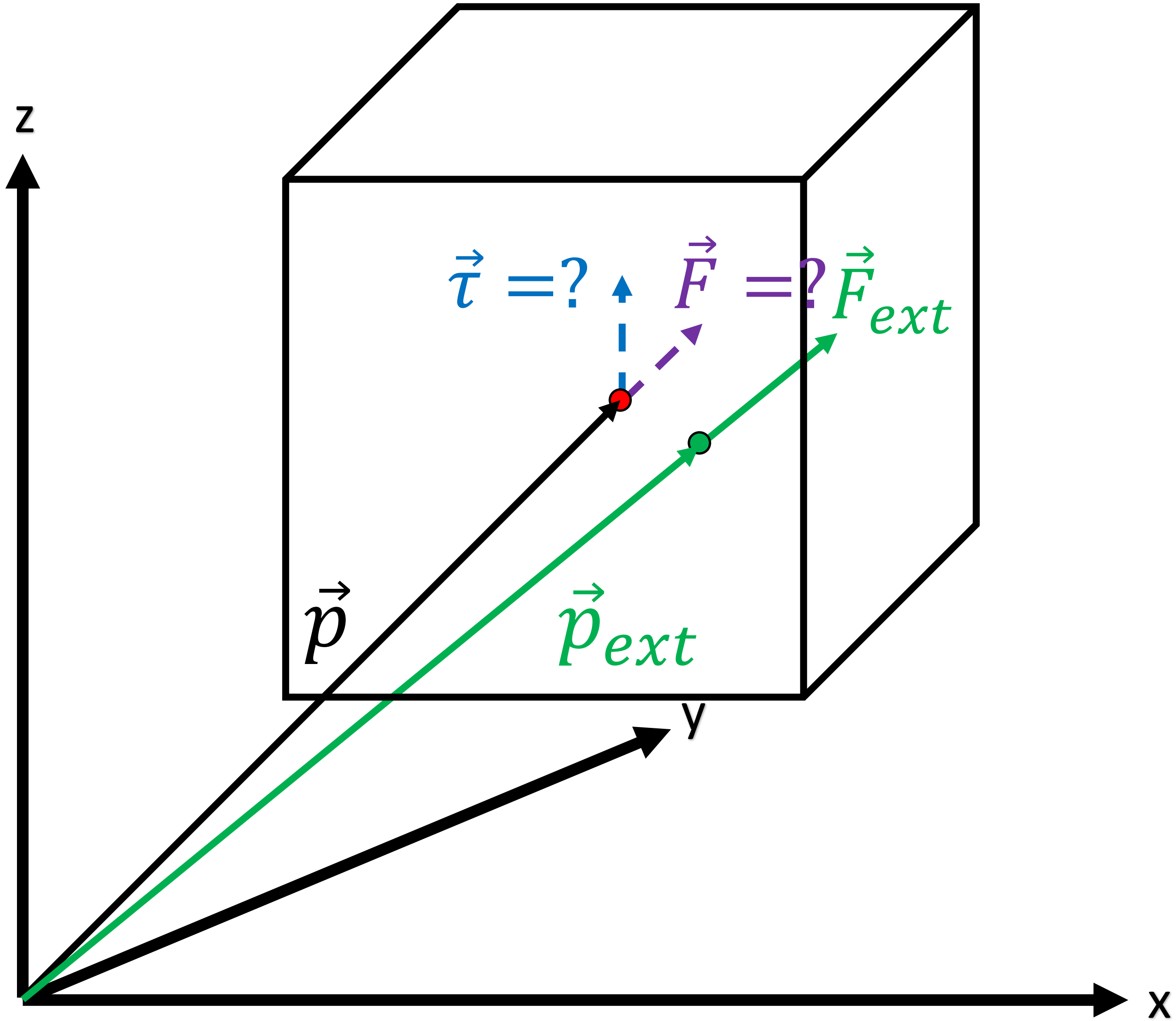
- has orientation and rotation



Kinematics

- Rigid body:

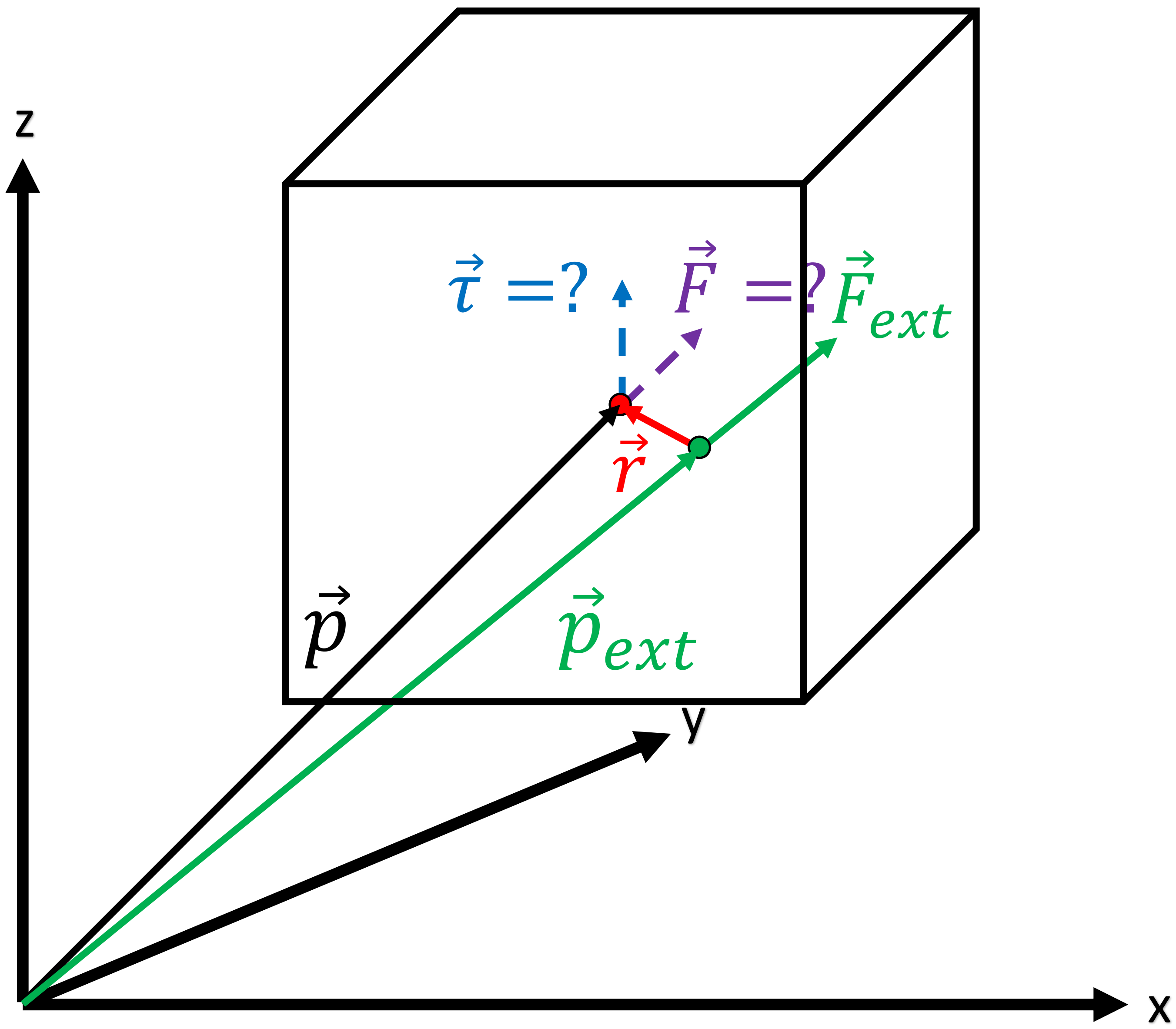
- has orientation and rotation
- Linear force $\vec{F}$ and torque $\vec{\tau}$ must be computed from an applied external force $\vec{F}_{ext}$



Kinematics

- Rigid body:

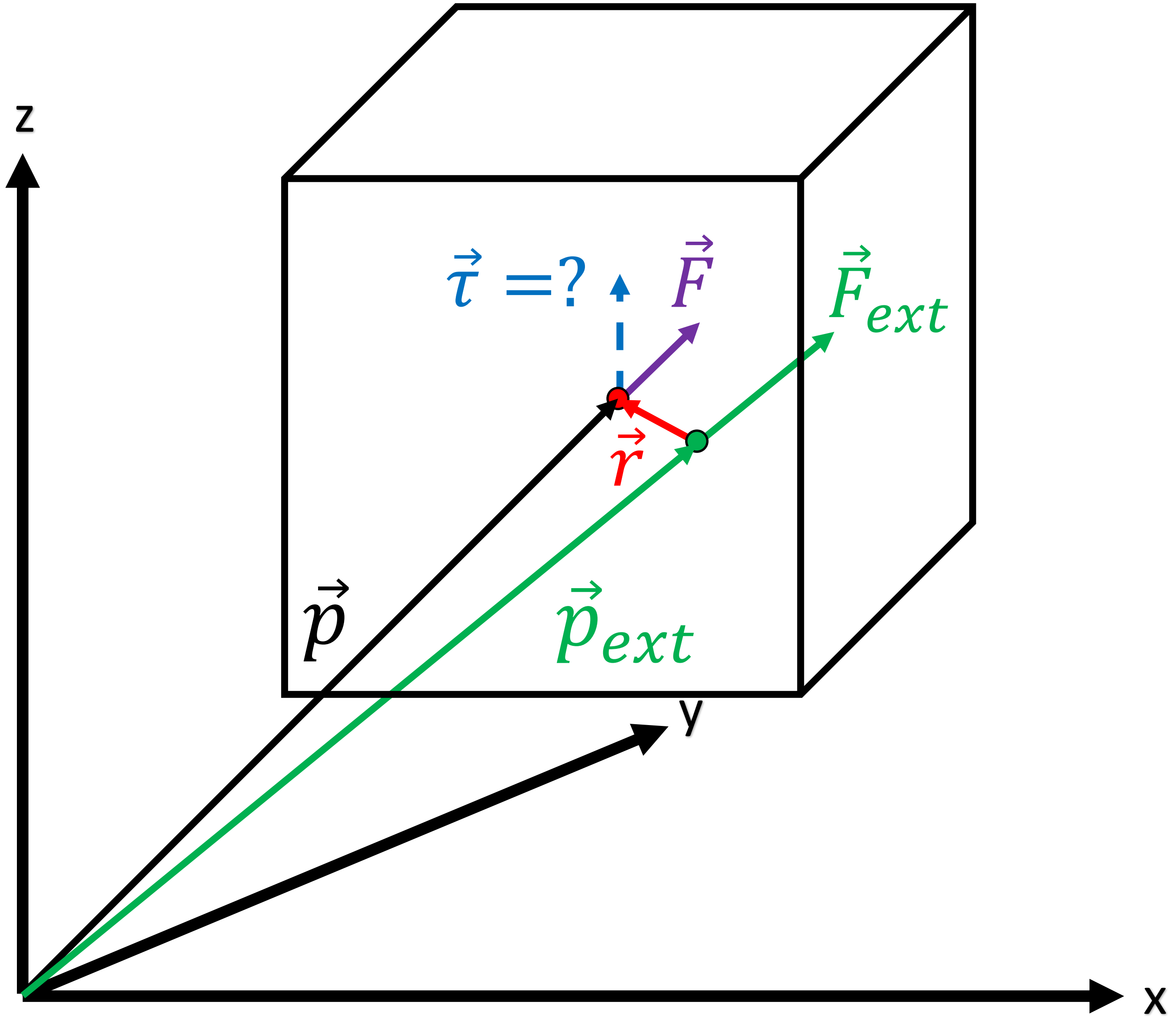
1. $\vec{r} = \vec{p} - \vec{p}_{ext}$



Kinematics

- Rigid body:

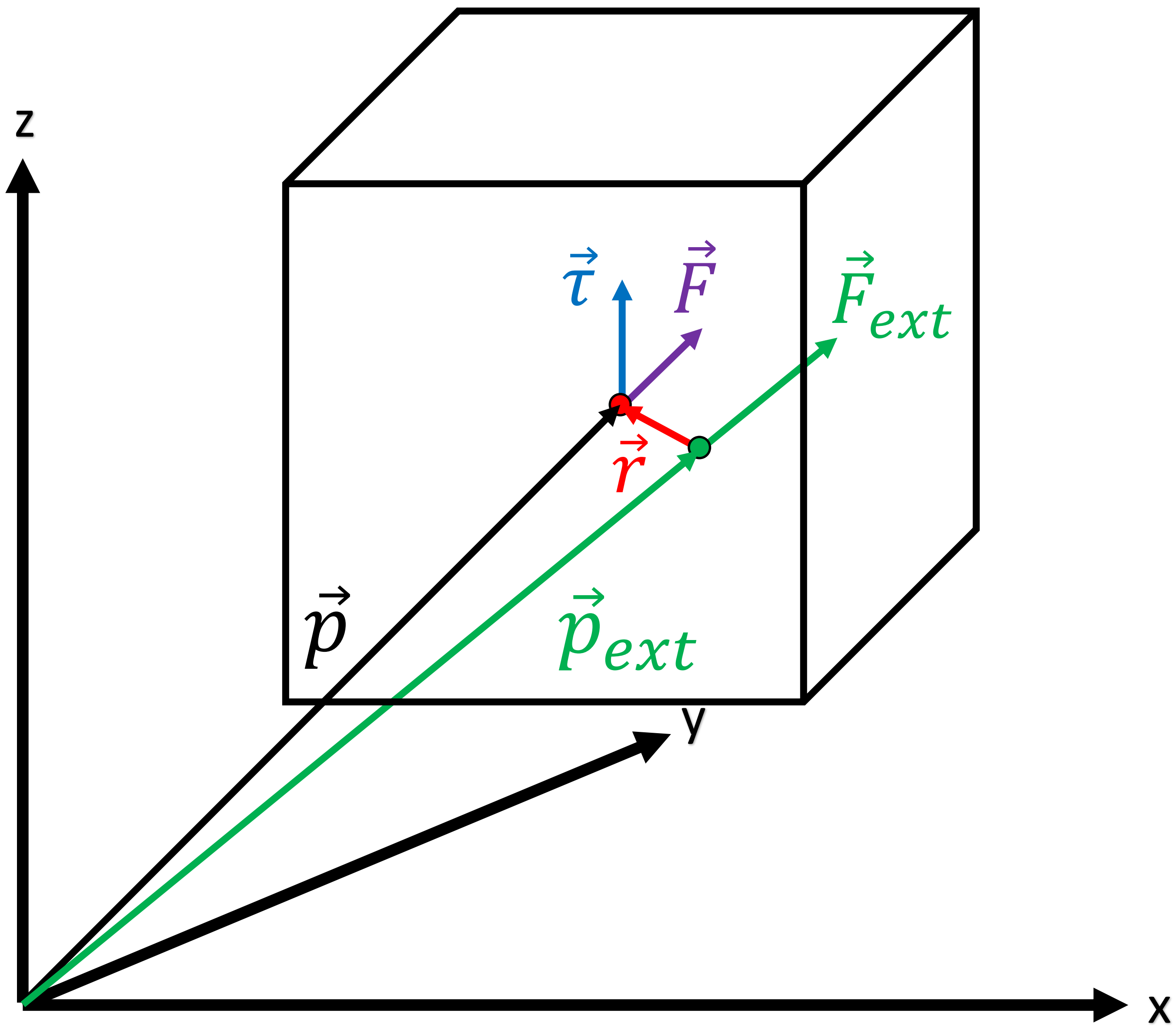
1. $\vec{r} = \vec{p} - \vec{p}_{ext}$
2. $\vec{F} = \vec{F}_{ext} \cdot \vec{r} \, \frac{\vec{F}_{ext}}{|\vec{F}_{ext}|}$



Kinematics

- Rigid body:

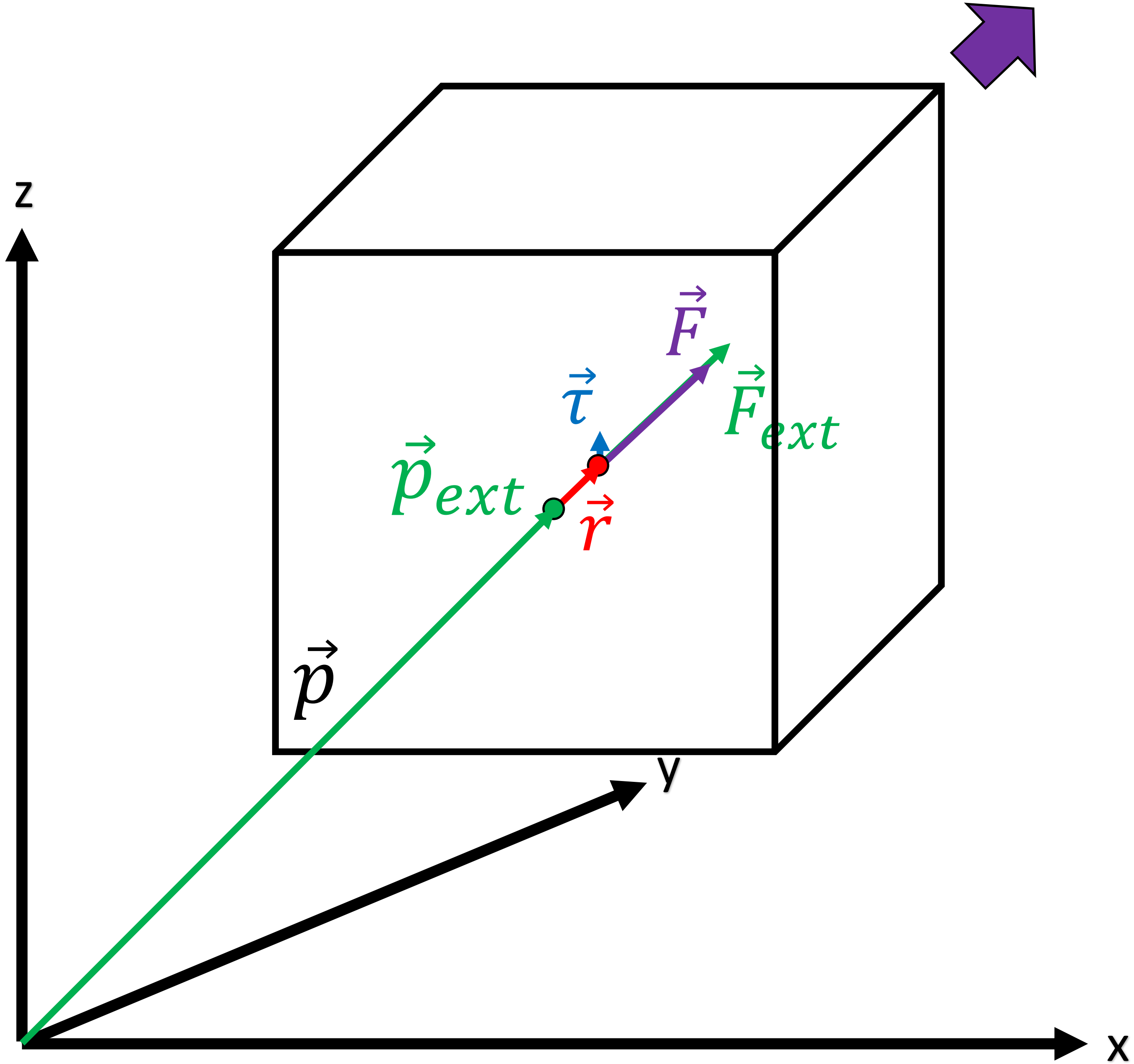
1. $\vec{r} = \vec{p} - \vec{p}_{ext}$
2. $\vec{F} = \vec{F}_{ext} \cdot \vec{r} \, \frac{\vec{F}_{ext}}{|\vec{F}_{ext}|}$
3. $\vec{\tau} = \vec{F}_{ext} \times \vec{r}$



Kinematics

- Rigid body:

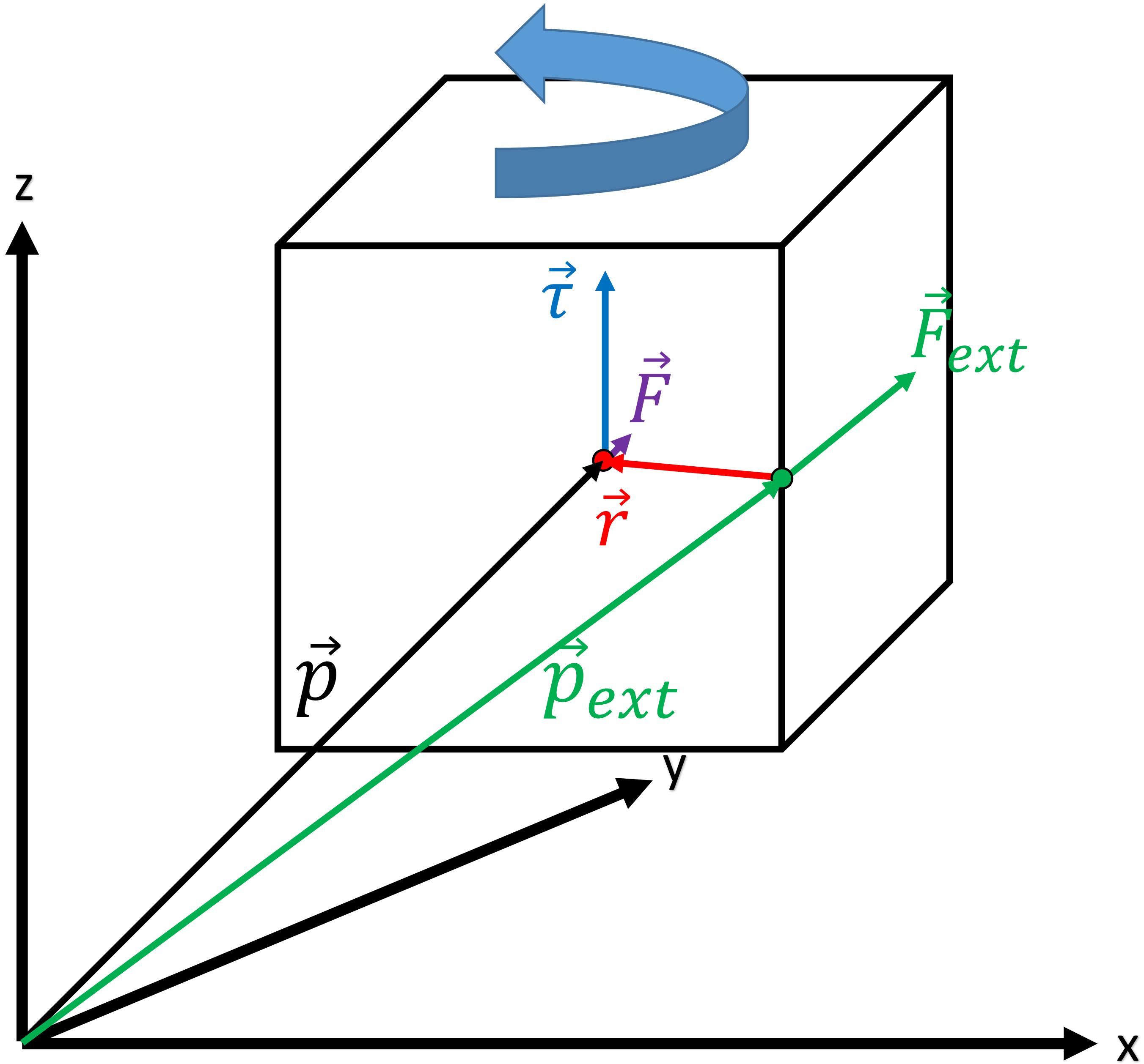
The more the $\vec{r}$ and $\vec{F}_{ext}$ are parallel to each other, the linear motion becomes more dominant



Kinematics

- Rigid body:

The more the $\vec{r}$ and $\vec{F}_{ext}$ are perpendicular to each other, the rotational motion becomes more dominant

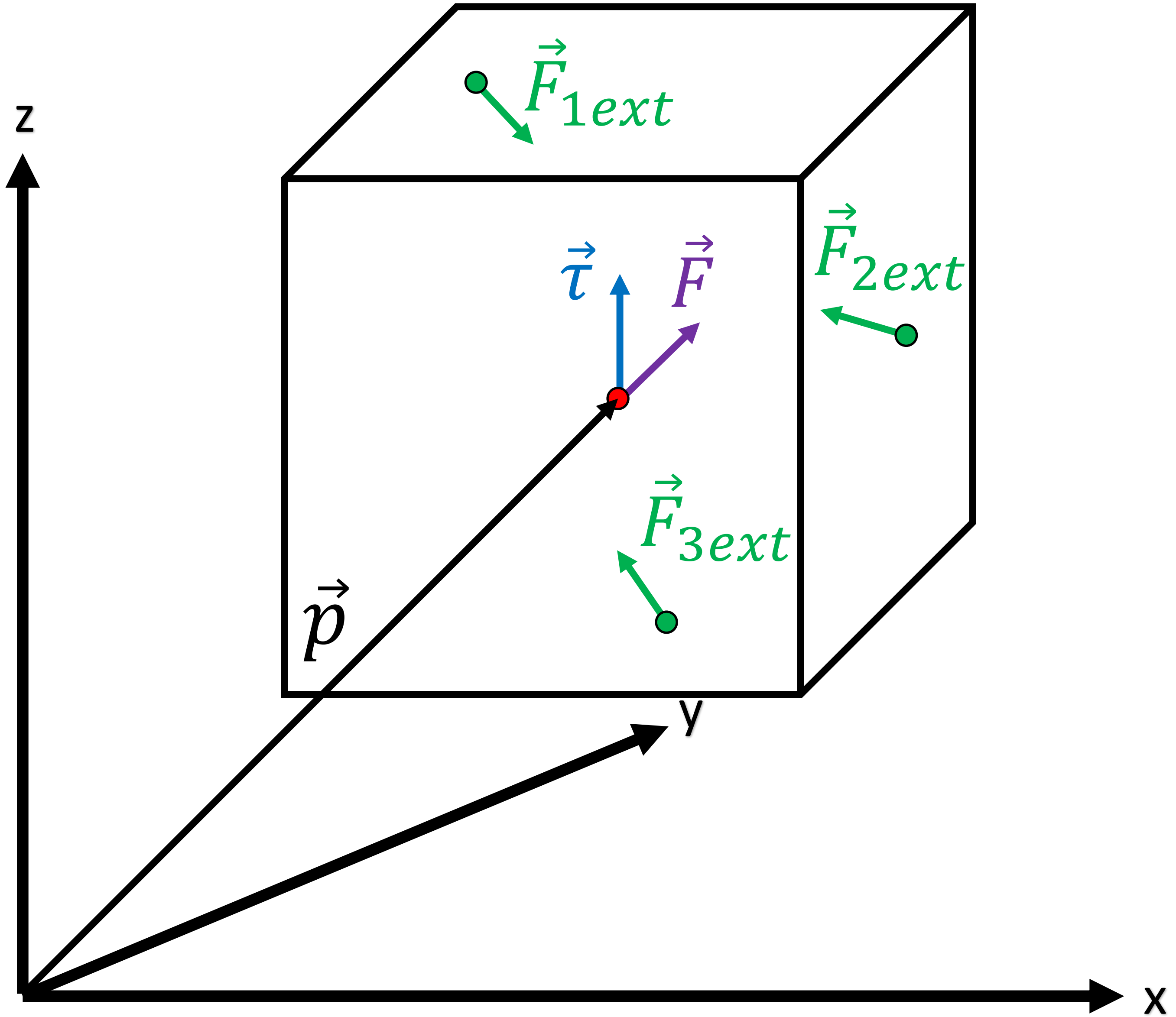


Kinematics

- Rigid body:

$$\vec{F} = \vec{F}_1 + \vec{F}_2 + \cdots$$

$$\vec{\tau} = \vec{\tau}_1 + \vec{\tau}_2 + \cdots$$



Kinematics

- Scaling:

If, for example, a body should travel:

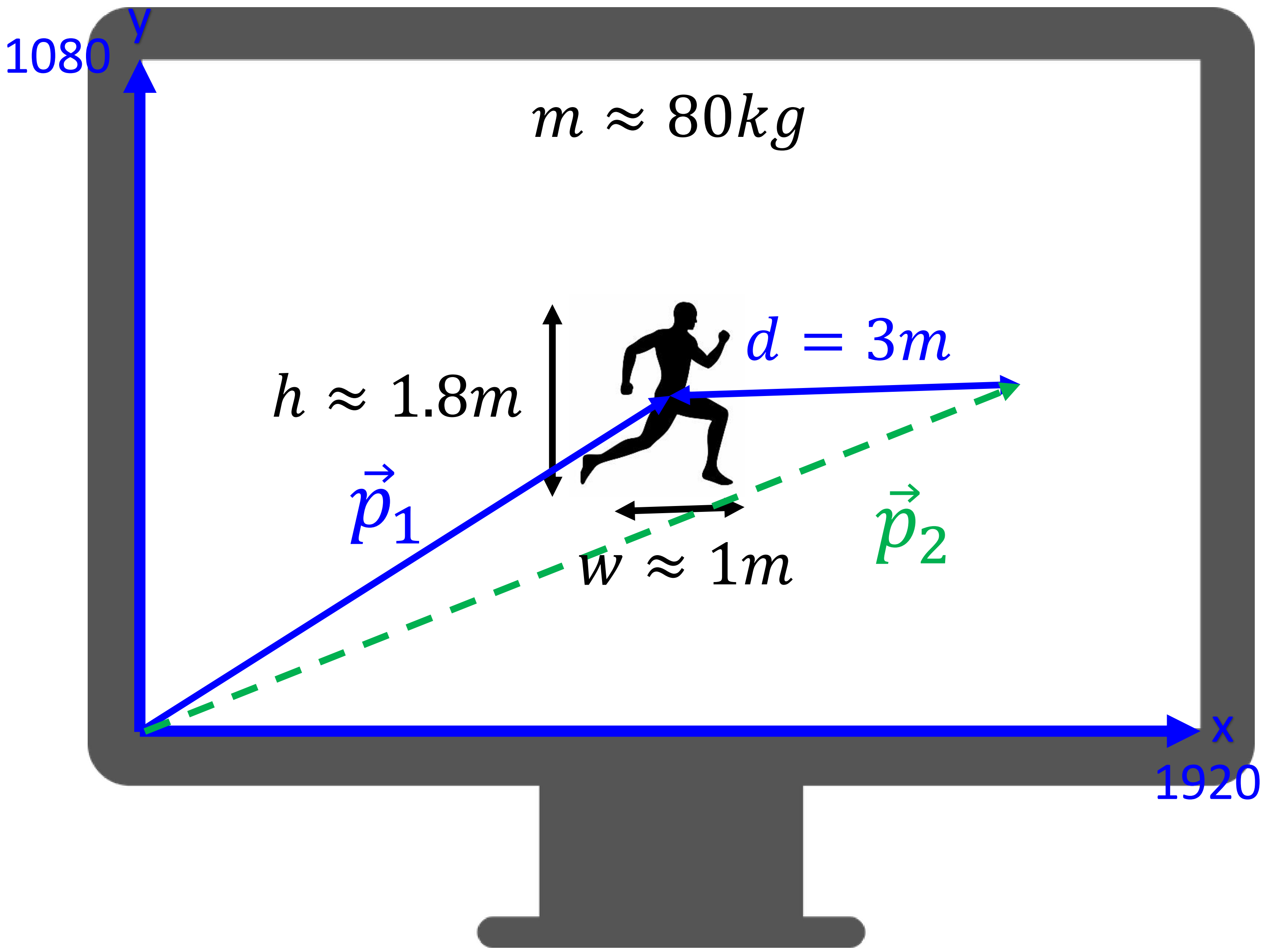
$$t = 1s$$

from position $\vec{p}_1$ to position $\vec{p}_2$ over a distance of:

$$d = 3m$$

it's velocity should be:

$$\vec{v} = \begin{bmatrix} 3 \\ 0 \end{bmatrix} \frac{m}{s}$$



Kinematics

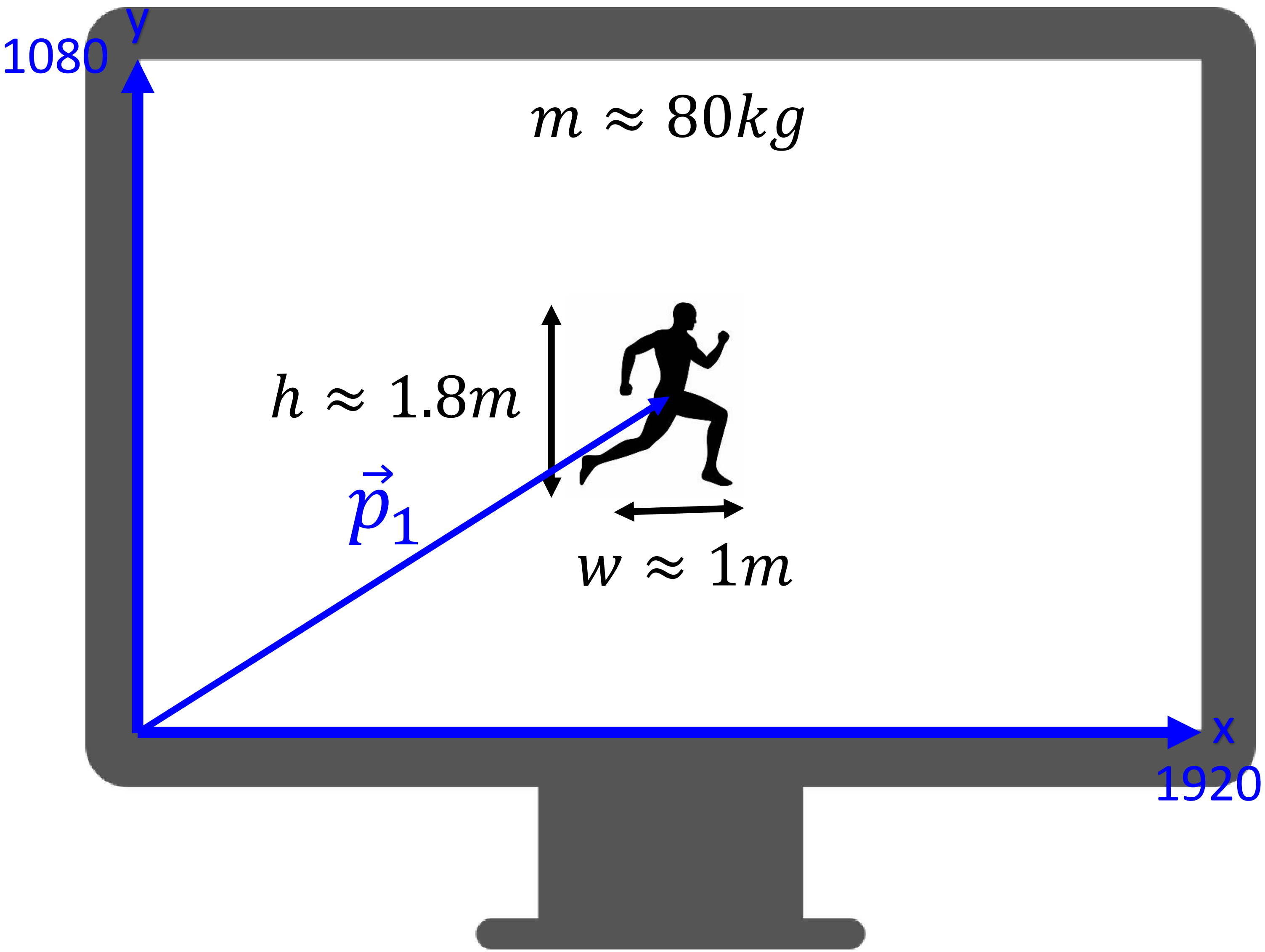
- Scaling:

without the scaling:

$$1m = 1px$$

$$\vec{p}_1 = \begin{bmatrix} 960 \\ 540 \end{bmatrix} px$$

$$\vec{v} = \begin{bmatrix} 3 \\ 0 \end{bmatrix} \frac{px}{s}$$



Kinematics

- scaling:

without the scaling:

$$1m = 1px$$

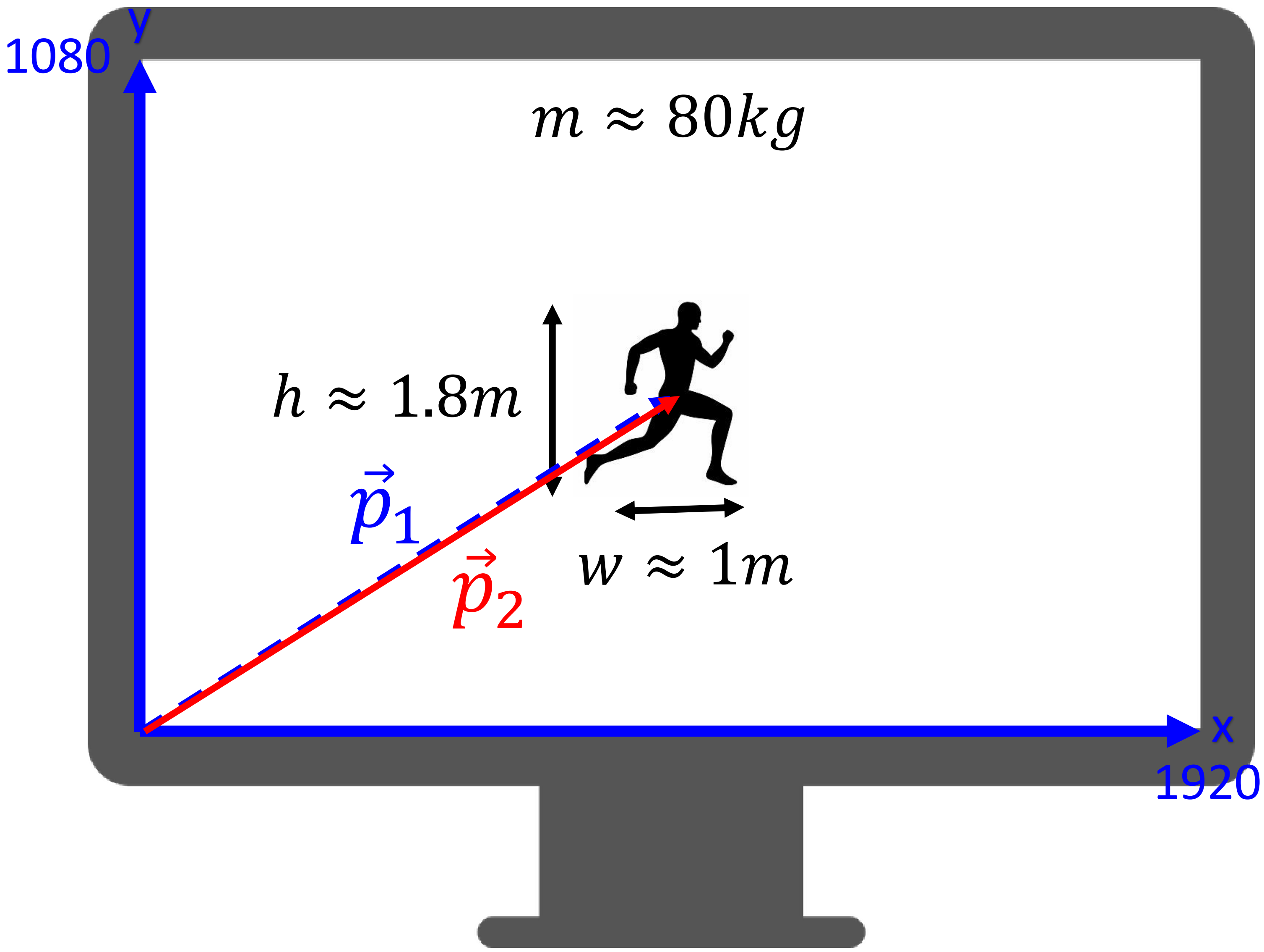
$$\vec{p}_1 = \begin{bmatrix} 960 \\ 540 \end{bmatrix} px$$

$$\vec{v} = \begin{bmatrix} 3 \\ 0 \end{bmatrix} \frac{px}{s}$$

$$t = 1s$$

$$\vec{p}_2 = \begin{bmatrix} 963 \\ 540 \end{bmatrix} px$$

Too slow?



Kinematics

- scaling:

without the scaling:

$$1m = 1px$$

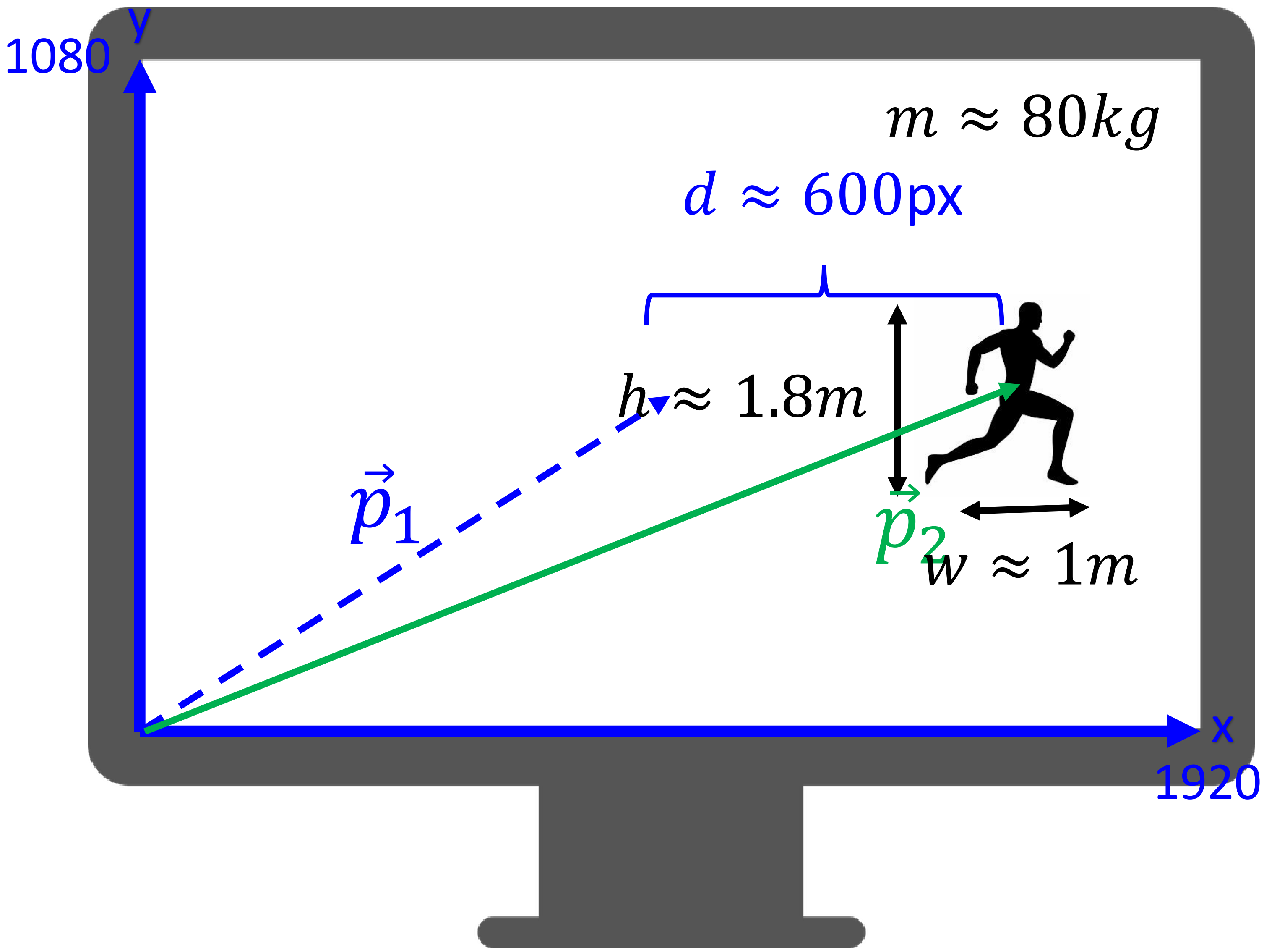
$$\vec{p}_1 = \begin{bmatrix} 960 \\ 540 \end{bmatrix} px$$

$$\vec{v} = \begin{bmatrix} 600 \\ 0 \end{bmatrix} \frac{px}{s}$$

$$t = 1s$$

$$\vec{p}_2 = \begin{bmatrix} 1560 \\ 540 \end{bmatrix} px$$

Solution?



Kinematics

- scaling:

without the scaling:

$$1m = 1px$$

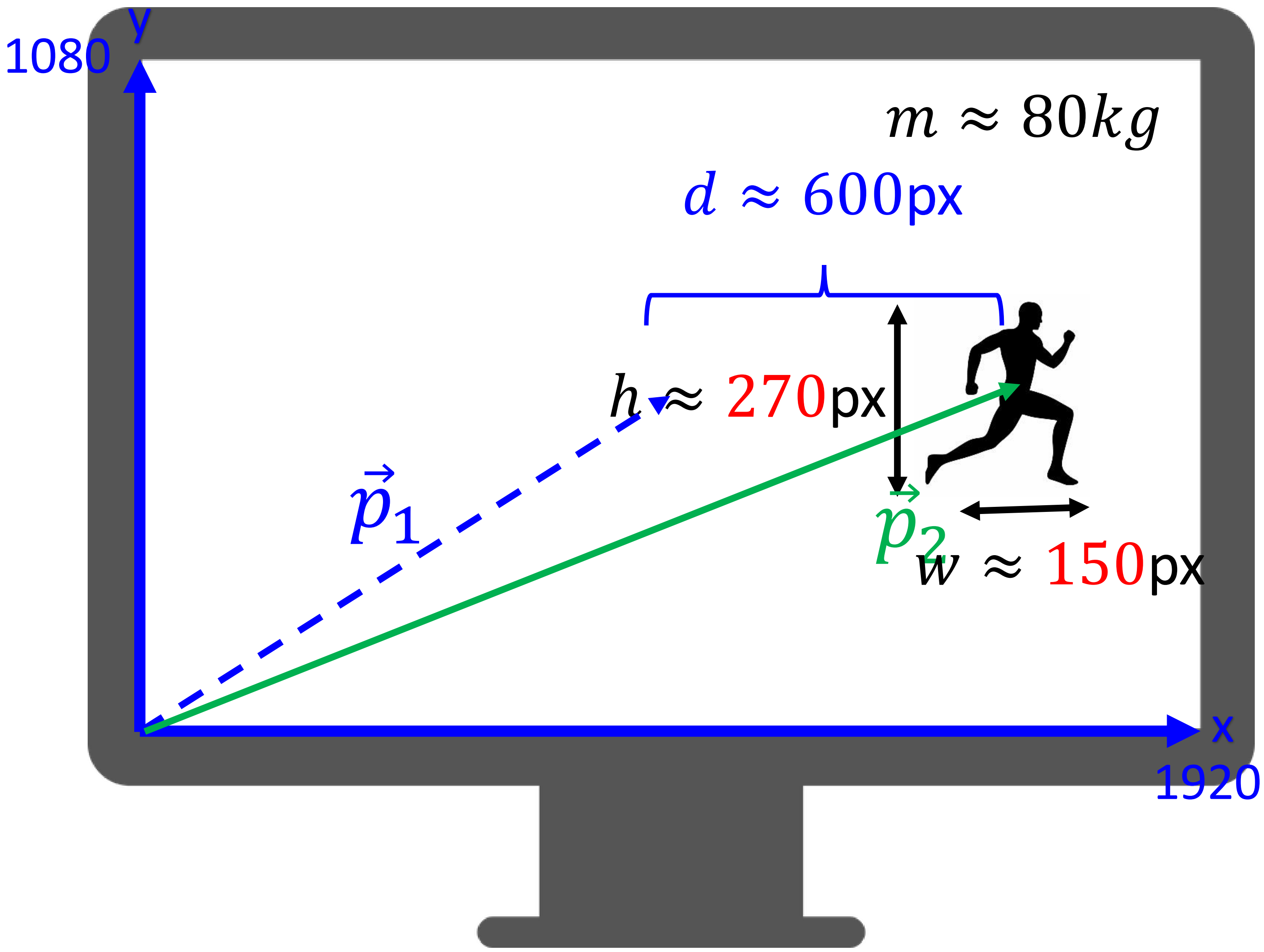
$$\vec{p}_1 = \begin{bmatrix} 960 \\ 540 \end{bmatrix} px$$

$$\vec{v} = \begin{bmatrix} 600 \\ 0 \end{bmatrix} \frac{px}{s}$$

$$t = 1s$$

$$\vec{p}_2 = \begin{bmatrix} 1560 \\ 540 \end{bmatrix} px$$

Solution?



Kinematics

- scaling:

without scaling:

$$1m = 1px$$

$$\vec{p}_1 = \begin{bmatrix} 960 \\ 540 \end{bmatrix} m$$

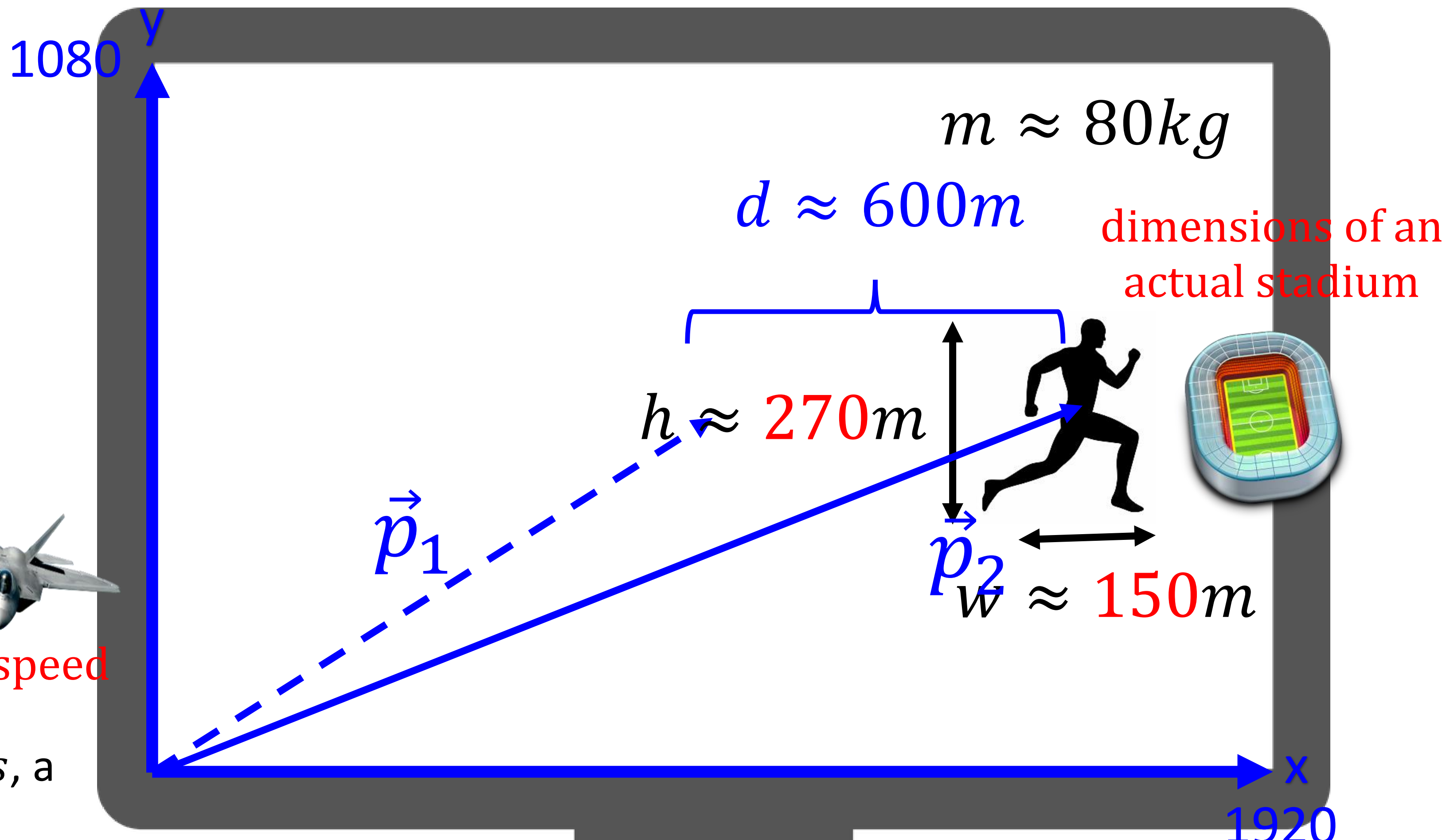
$$\vec{v} = \begin{bmatrix} 600 \\ 0 \end{bmatrix} \frac{m}{s}$$

F22′s speed

To achieve such speed during $1s$, a force is required of:

$$F = 80kg \frac{\begin{bmatrix} 600 \\ 0 \end{bmatrix} \frac{m}{s}}{1s} = \begin{bmatrix} 48000 \\ 0 \end{bmatrix} N$$

thrust force of a jet engine



Kinematics

- scaling:

with the scaling:

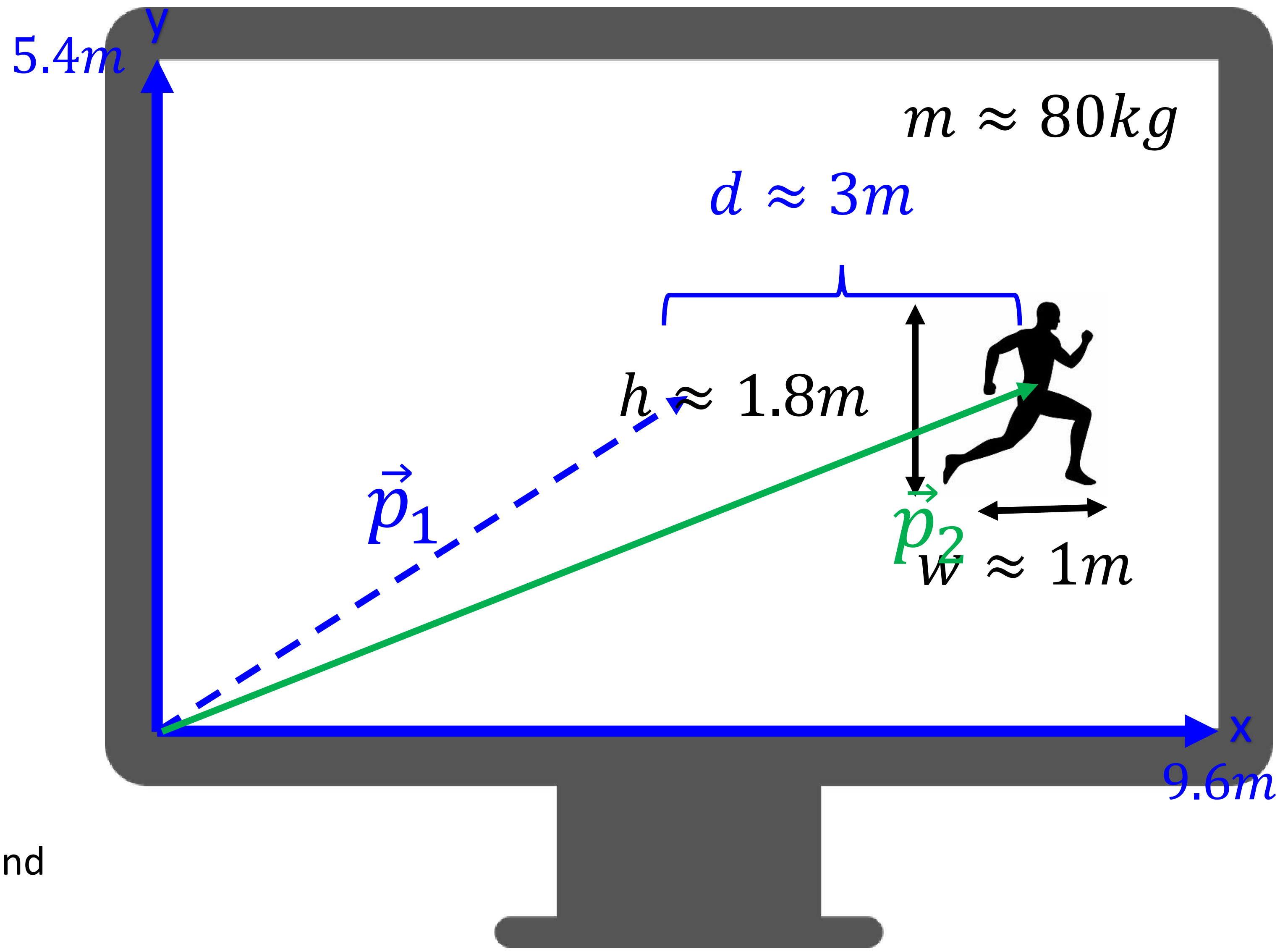
$$1m = 200px$$

$$\vec{p}_1 = \begin{bmatrix} 4.8 \\ 2.8 \end{bmatrix} m$$

$$\vec{v} = \begin{bmatrix} 3 \\ 0 \end{bmatrix} \frac{m}{s}$$

$$t = 1s$$

$$\vec{p}_2 = \begin{bmatrix} 7.8 \\ 2.8 \end{bmatrix} m$$



Ratio is determined empirically and should be parametrized (*zoom*).

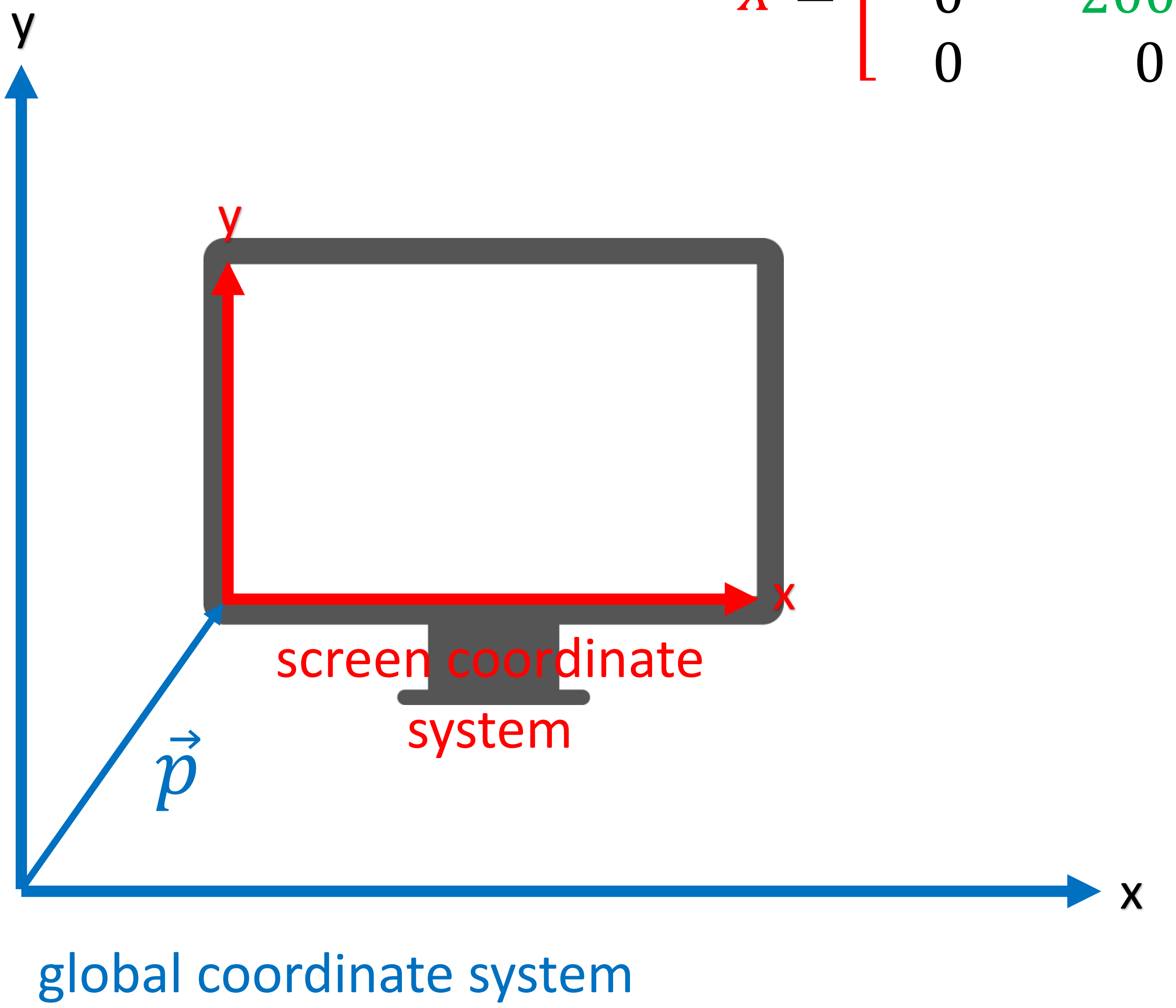
Kinematics

- screen coordinate system:

$$X = \begin{bmatrix} s_x & 0 & p_x \\ 0 & s_y & p_y \\ 0 & 0 & 1 \end{bmatrix}$$

The greater the scaling factor, the greater portion of global space can fit onto screen!

$$X = \begin{bmatrix} 200.0 & 0 & 1.0 \\ 0 & 200.0 & 2.0 \\ 0 & 0 & 1 \end{bmatrix}$$



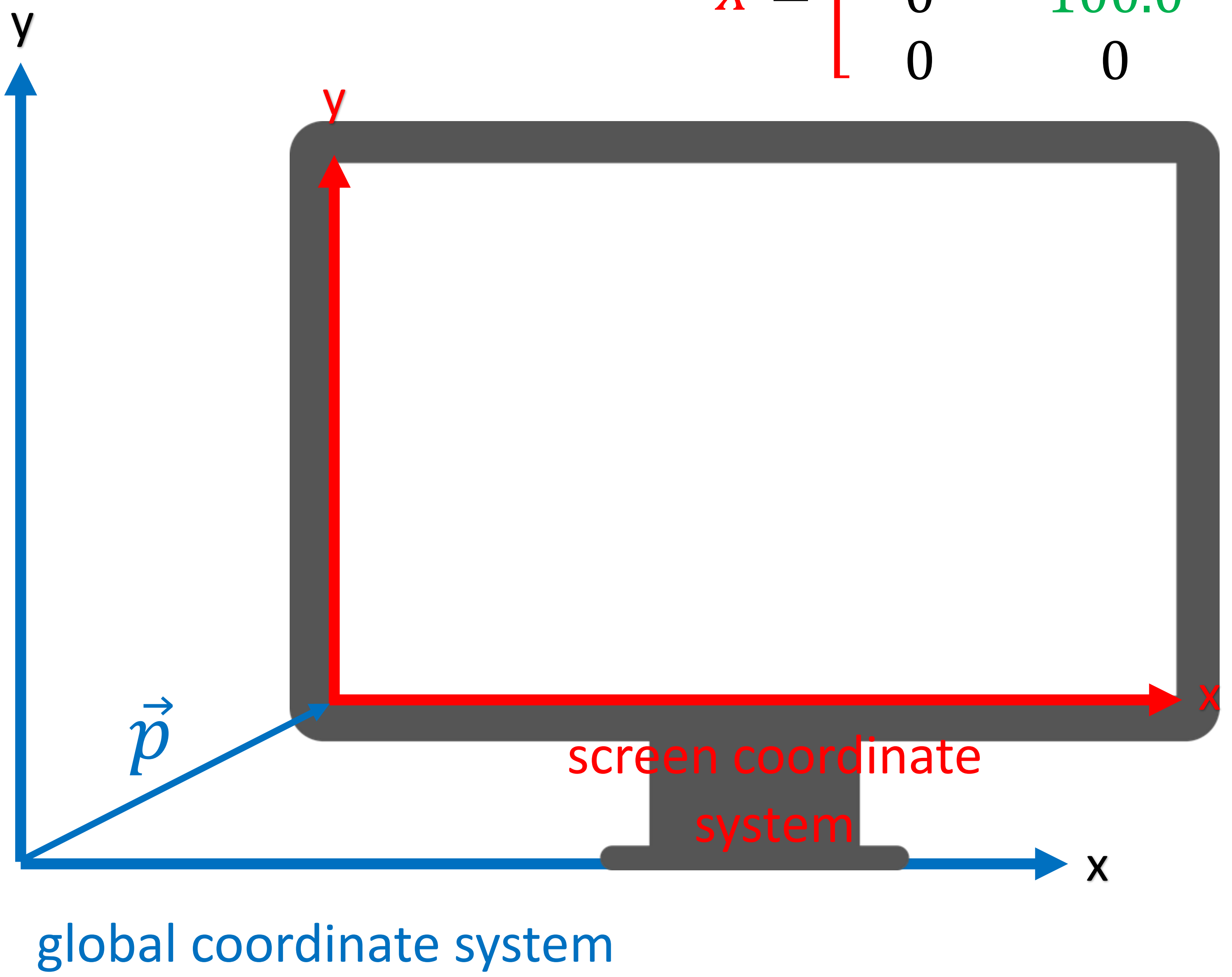
Kinematics

- screen coordinate system:

$$X = \begin{bmatrix} s_x & 0 & p_x \\ 0 & s_y & p_y \\ 0 & 0 & 1 \end{bmatrix}$$

$$X = \begin{bmatrix} 100.0 & 0 & 2.0 \\ 0 & 100.0 & 1.0 \\ 0 & 0 & 1 \end{bmatrix}$$

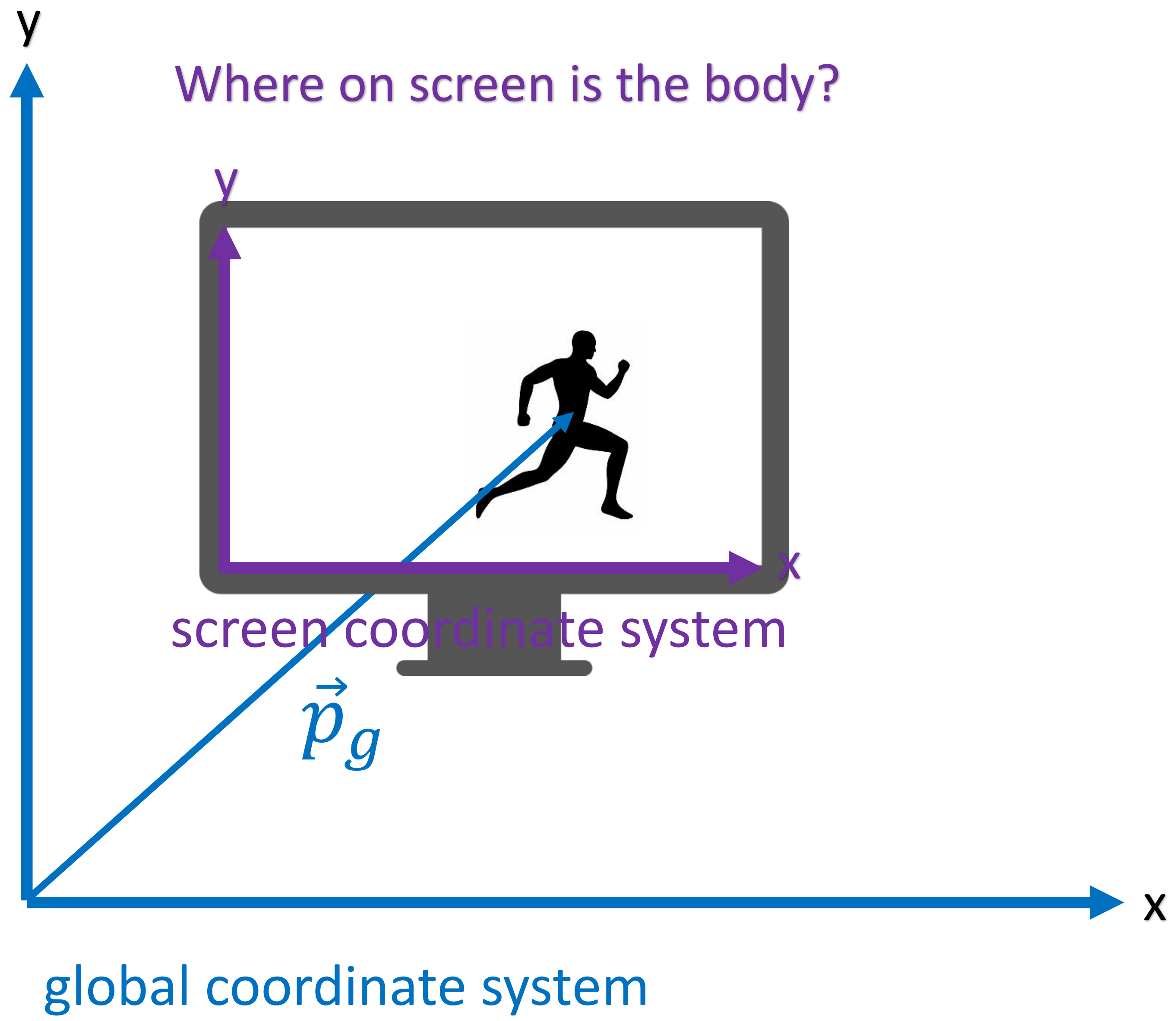
The smaller the scaling factor, the lesser portion of global space can fit onto screen!



Kinematics

- screen coordinate system:

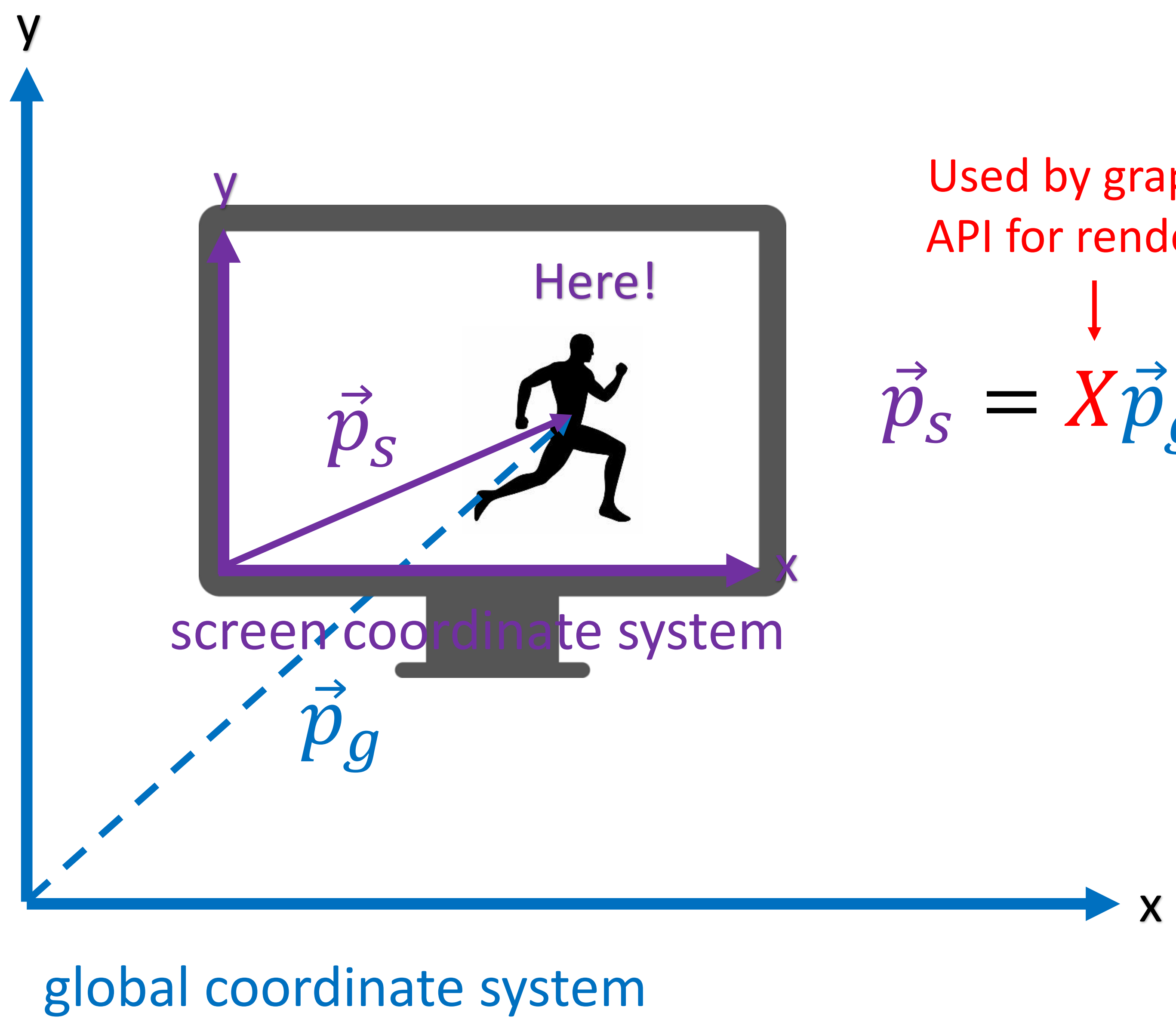
$$X = \begin{bmatrix} s_x & 0 & p_x \\ 0 & s_y & p_y \\ 0 & 0 & 1 \end{bmatrix}$$



Kinematics

- screen coordinate system:

$$X = \begin{bmatrix} s_x & 0 & p_x \\ 0 & s_y & p_y \\ 0 & 0 & 1 \end{bmatrix}$$



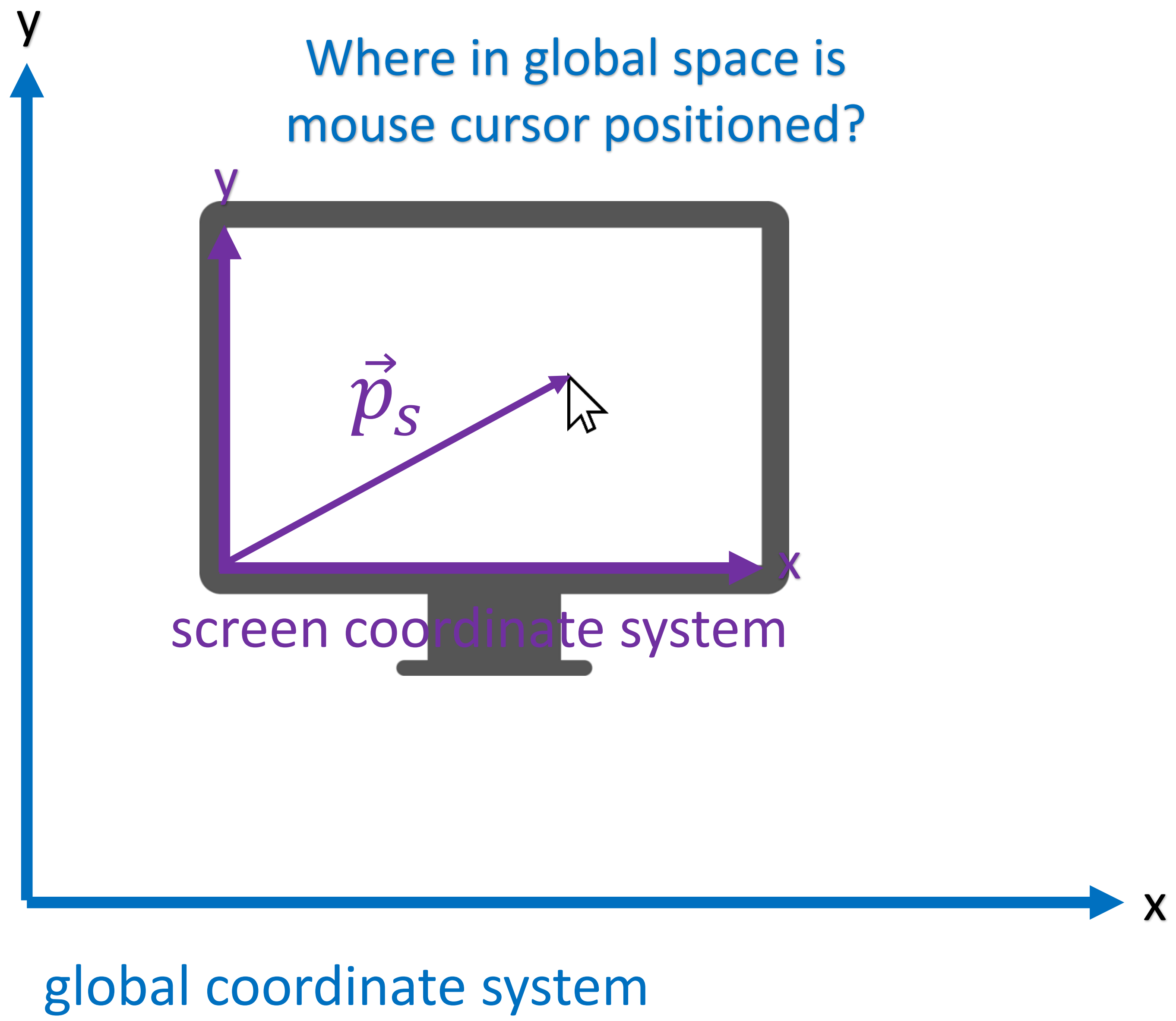
Used by graphics API for rendering

$$\vec{p}_s = X\vec{p}_g$$

Kinematics

- screen coordinate system:

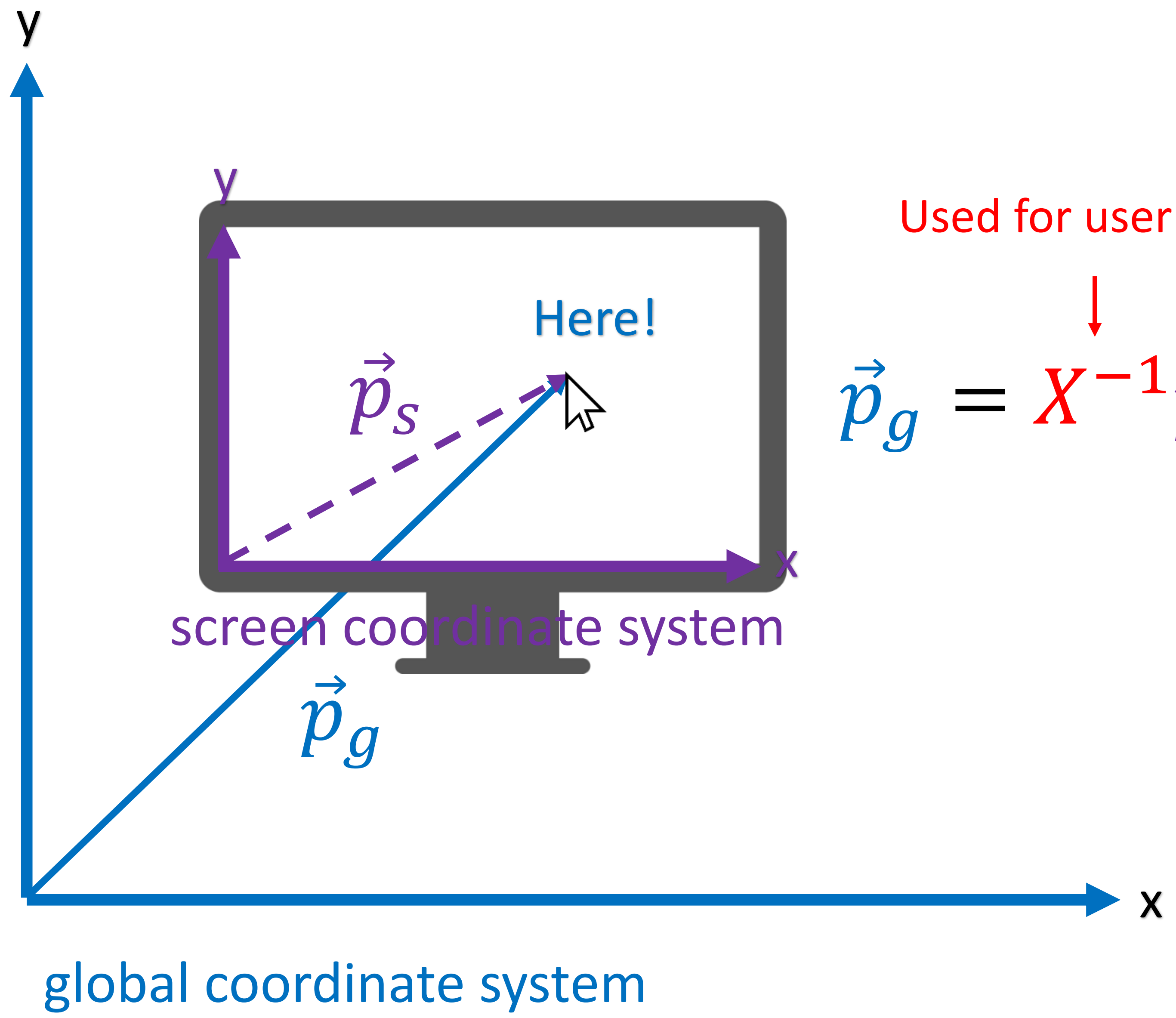
$$X = \begin{bmatrix} s_x & 0 & p_x \\ 0 & s_y & p_y \\ 0 & 0 & 1 \end{bmatrix}$$



Kinematics

- screen coordinate system:

$$X = \begin{bmatrix} s_x & 0 & p_x \\ 0 & s_y & p_y \\ 0 & 0 & 1 \end{bmatrix}$$



Used for user input

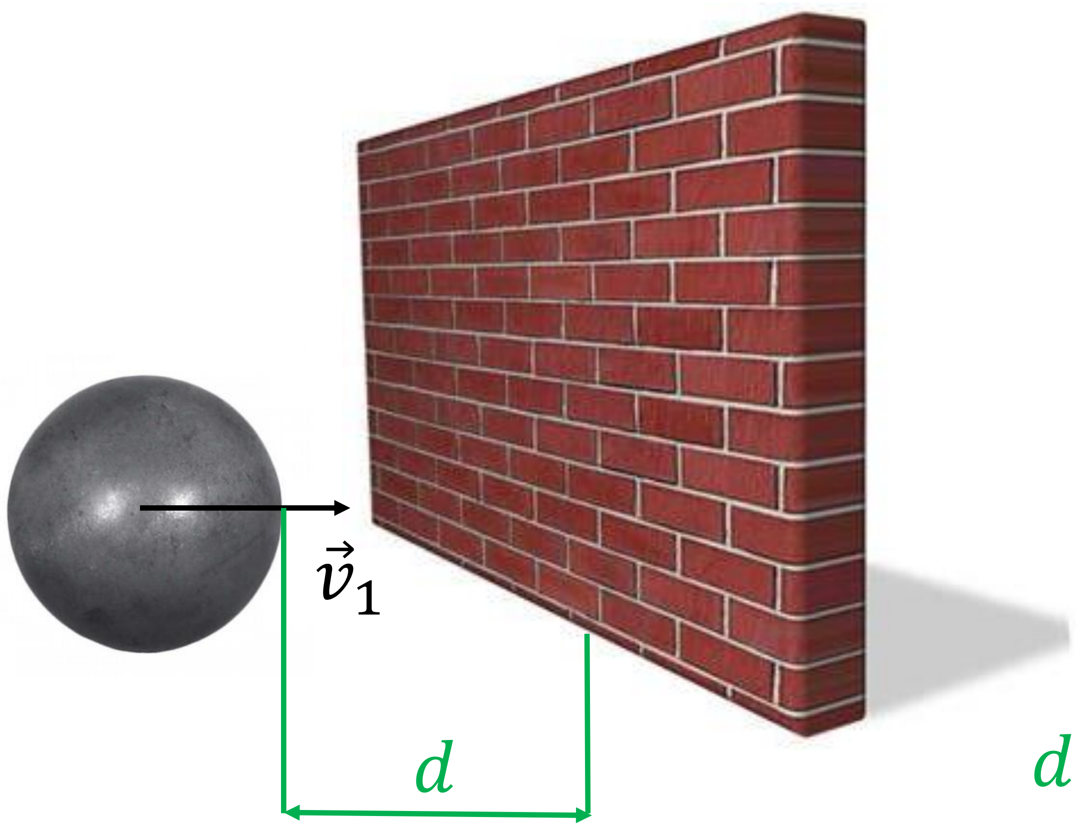
$$\vec{p}_g = X^{-1}\vec{p}_s$$

Constrained motion

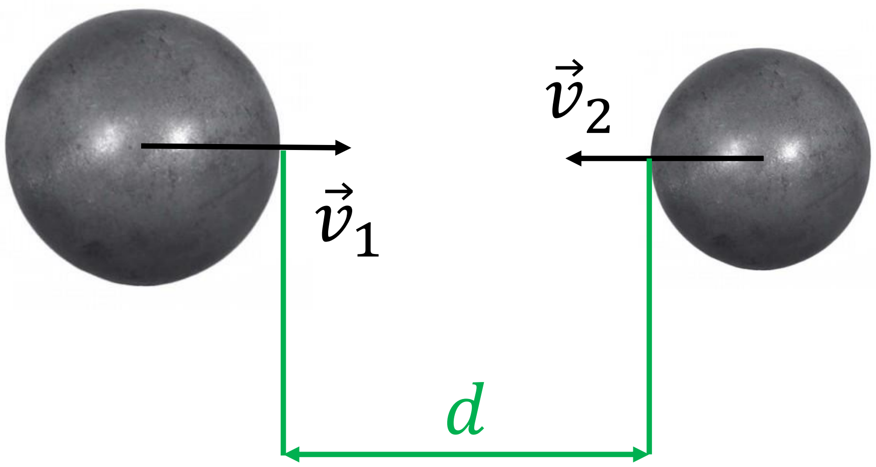
How to limit the motion of a physical system given predetermined conditions?

Constrained motion

- Contact constraints: admissible state

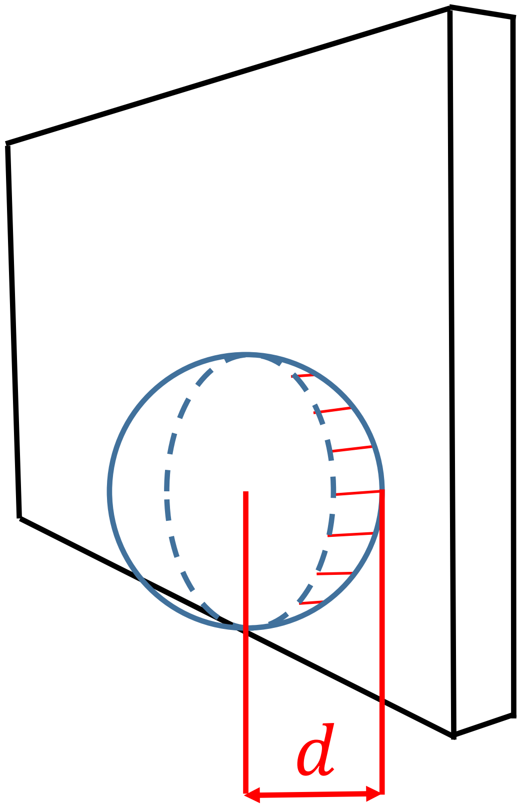


$d \geq 0$

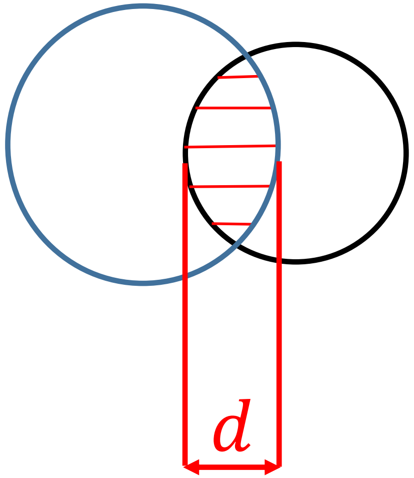


Constrained motion

- Contact constraints: inadmissible state

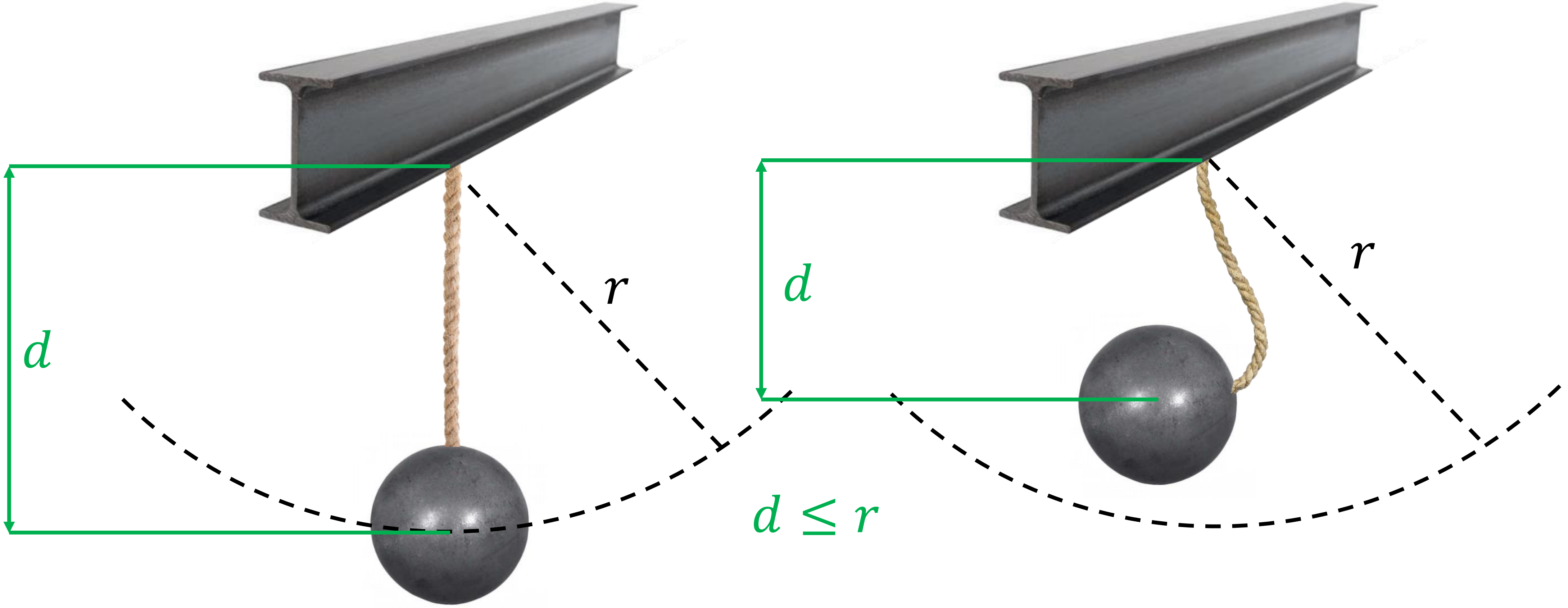


$d < 0$



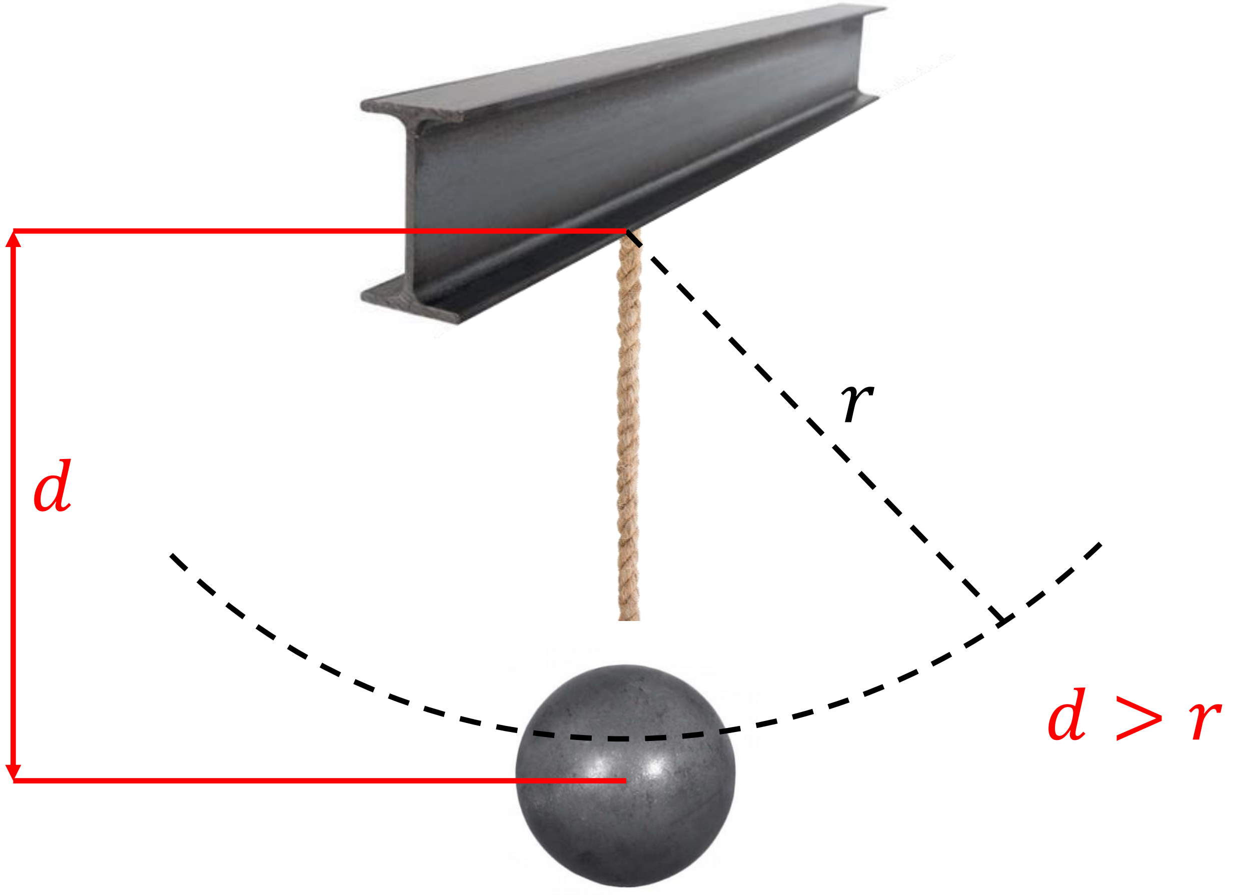
Constrained motion

- Pendulums: admissible state



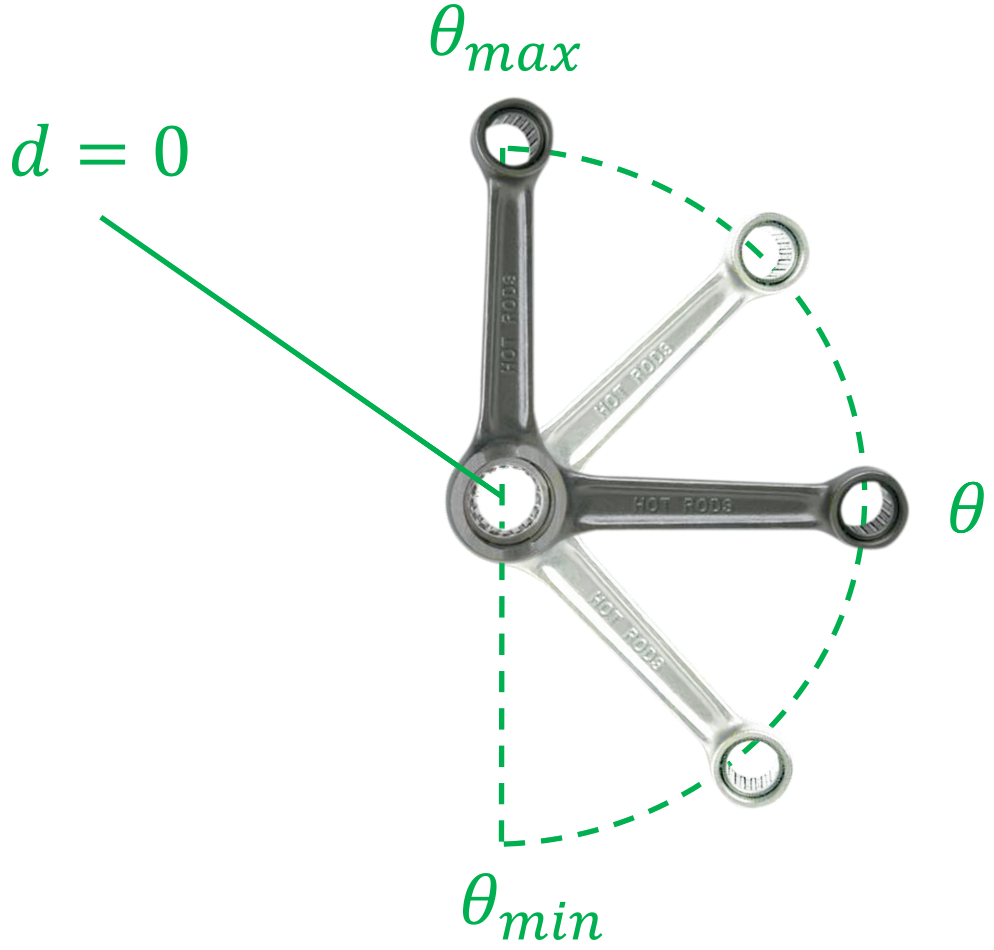
Constrained motion

- Pendulums: inadmissible state



Constrained motion

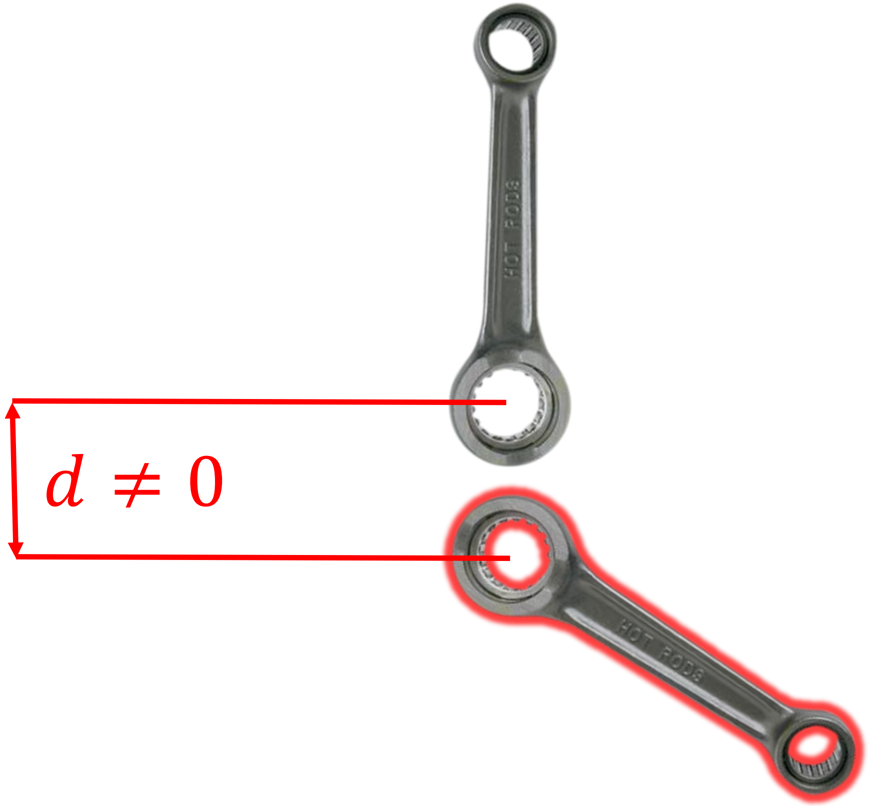
- Joints: admissible state

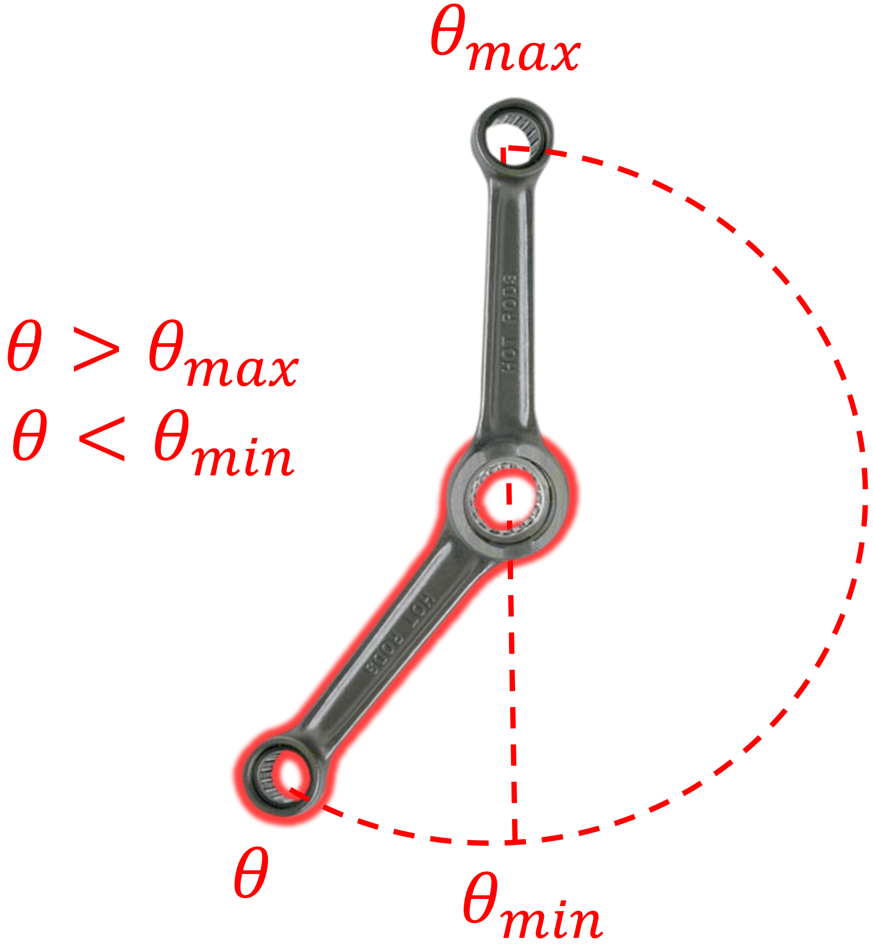


$$\theta_{min} \leq \theta \leq \theta_{max}$$

Constrained motion

- Joints: inadmissible state





Constrained motion

- springs
- fluids
- cloth
- ragdolls
- etc.

Constrained motion

- Engine pipeline:

Apply external forces

Integrate

Detect constraint violations

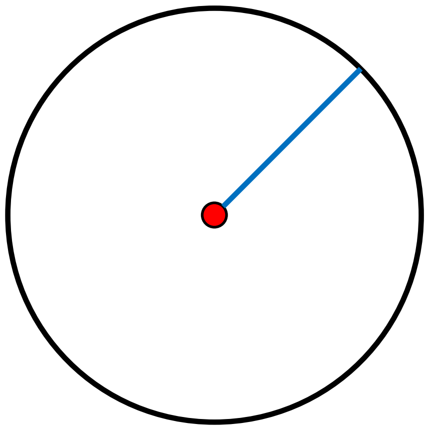
Solve constraint violations

Render

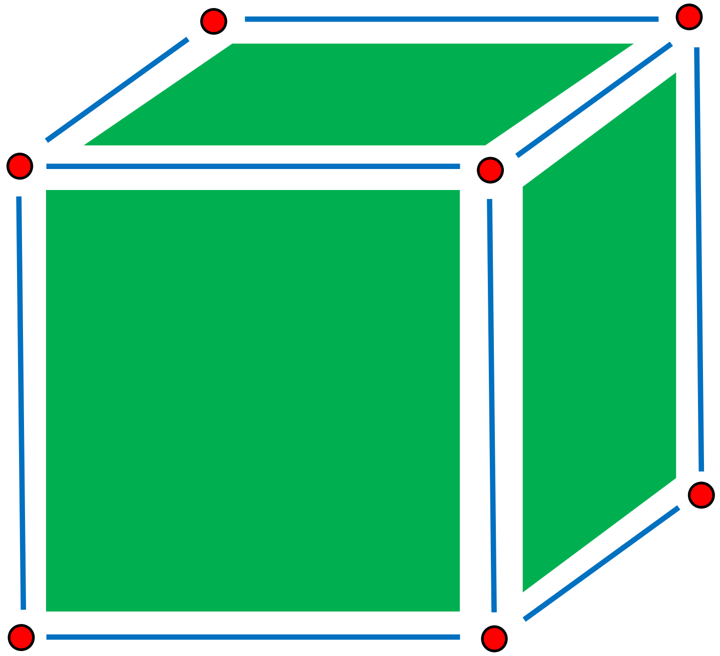
Sleep

Constrained motion

1. Collision detection
   - geometry features



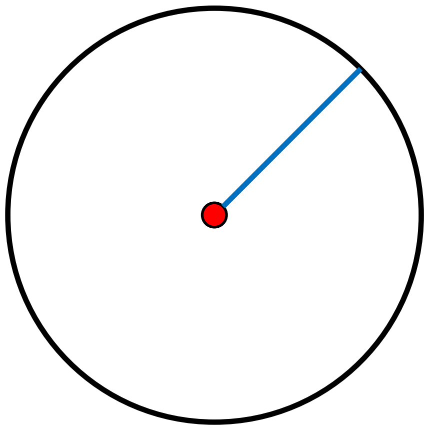
- center
- radius



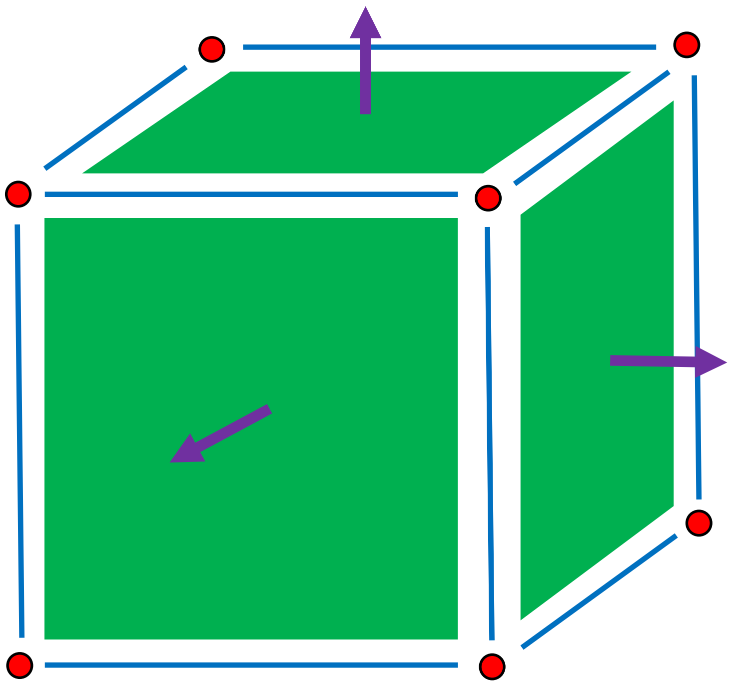
- veritces
- edges
- faces
- where is outside or inside?

Constrained motion

1. Collision detection
   - geometry features



- center
- radius

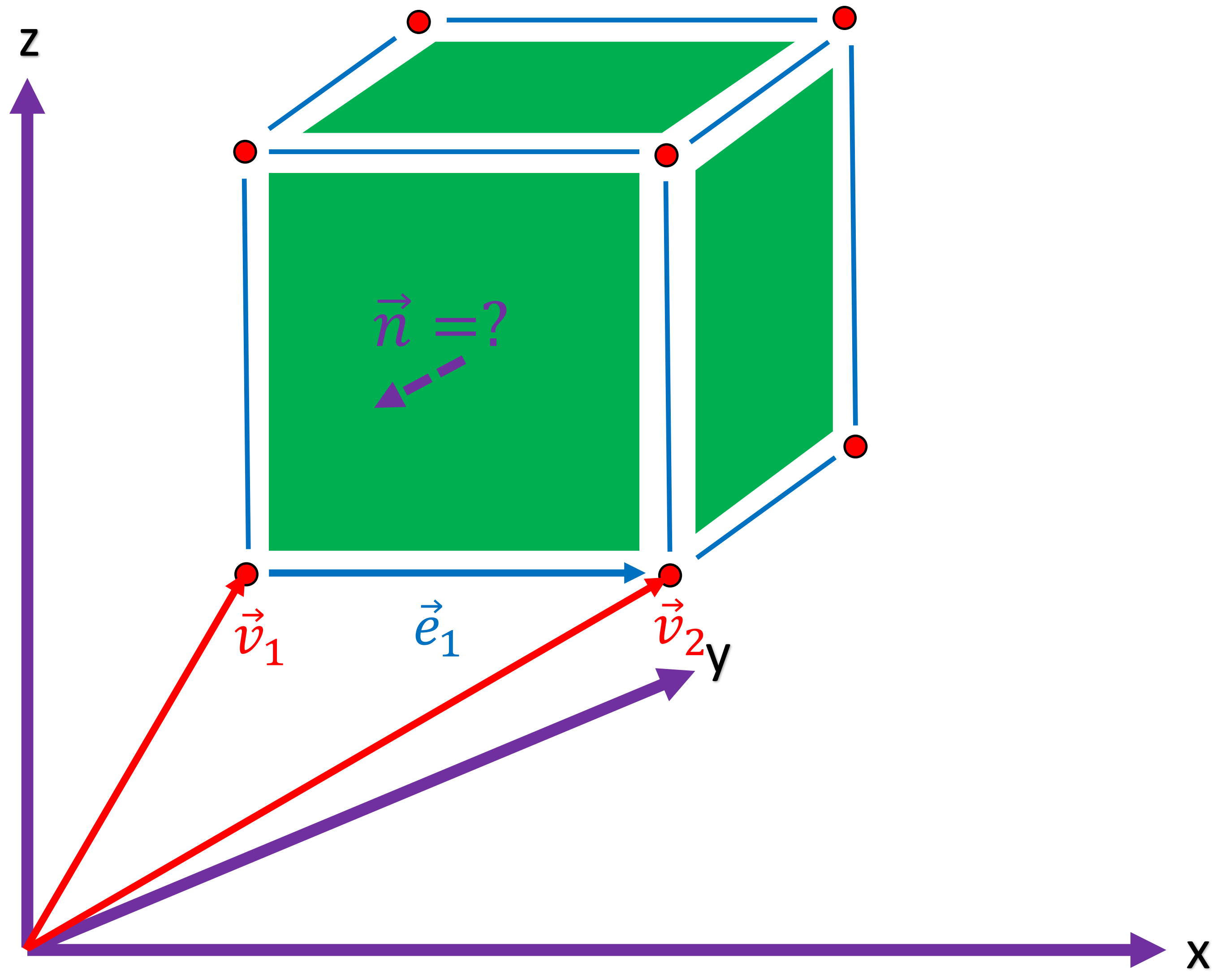


- veritces
- edges
- faces
- normals

Constrained motion

1. Collision detection
   - normals (3D)

$\vec{e}_1 = \vec{v}_2 - \vec{v}_1$

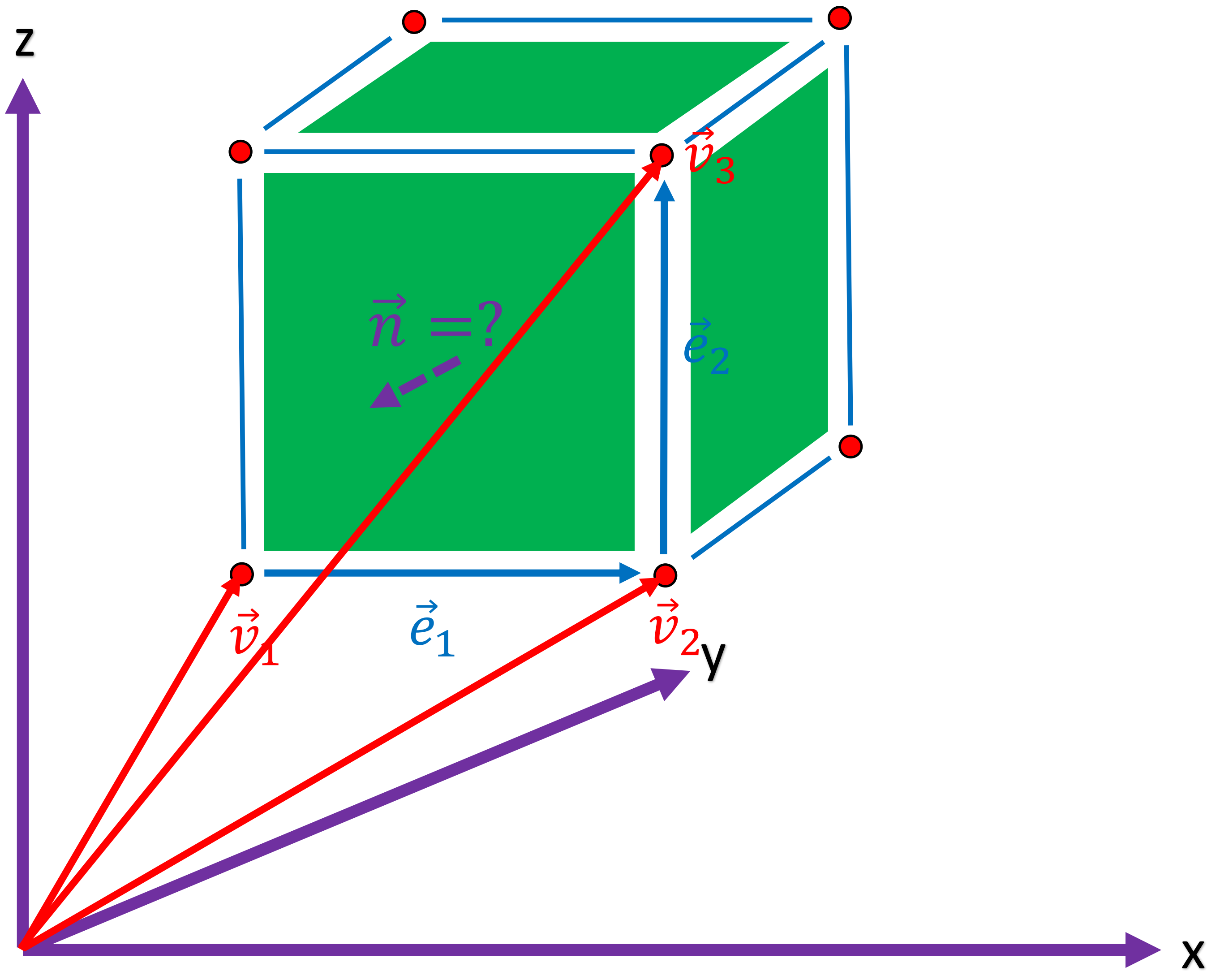


Constrained motion

1. Collision detection
   - normals (3D)

$\vec{e}_1 = \vec{v}_2 - \vec{v}_1$

$\vec{e}_2 = \vec{v}_3 - \vec{v}_2$



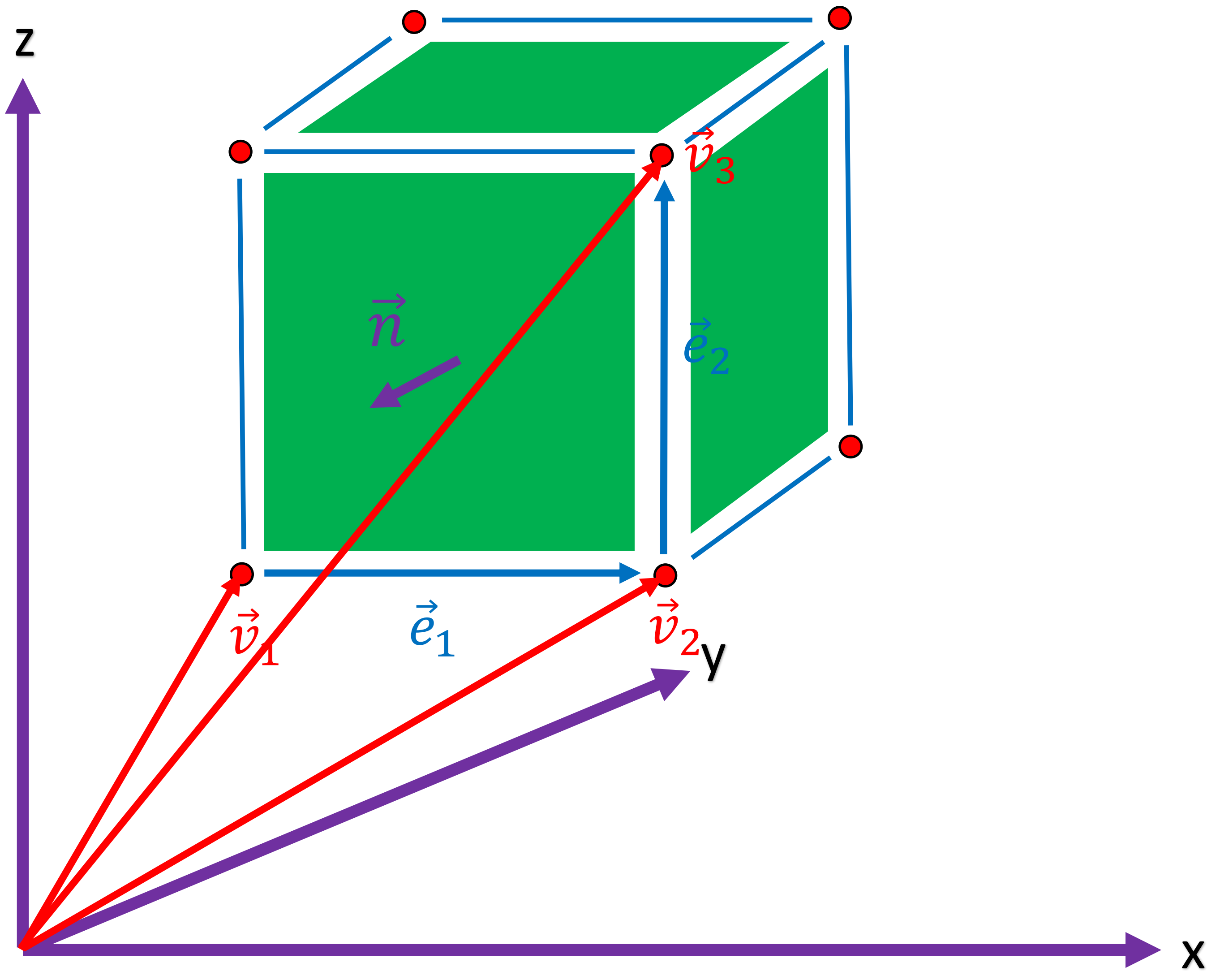
Constrained motion

1. Collision detection
   - normals (3D)

$$\vec{e}_1 = \vec{v}_2 - \vec{v}_1$$

$$\vec{e}_2 = \vec{v}_3 - \vec{v}_2$$

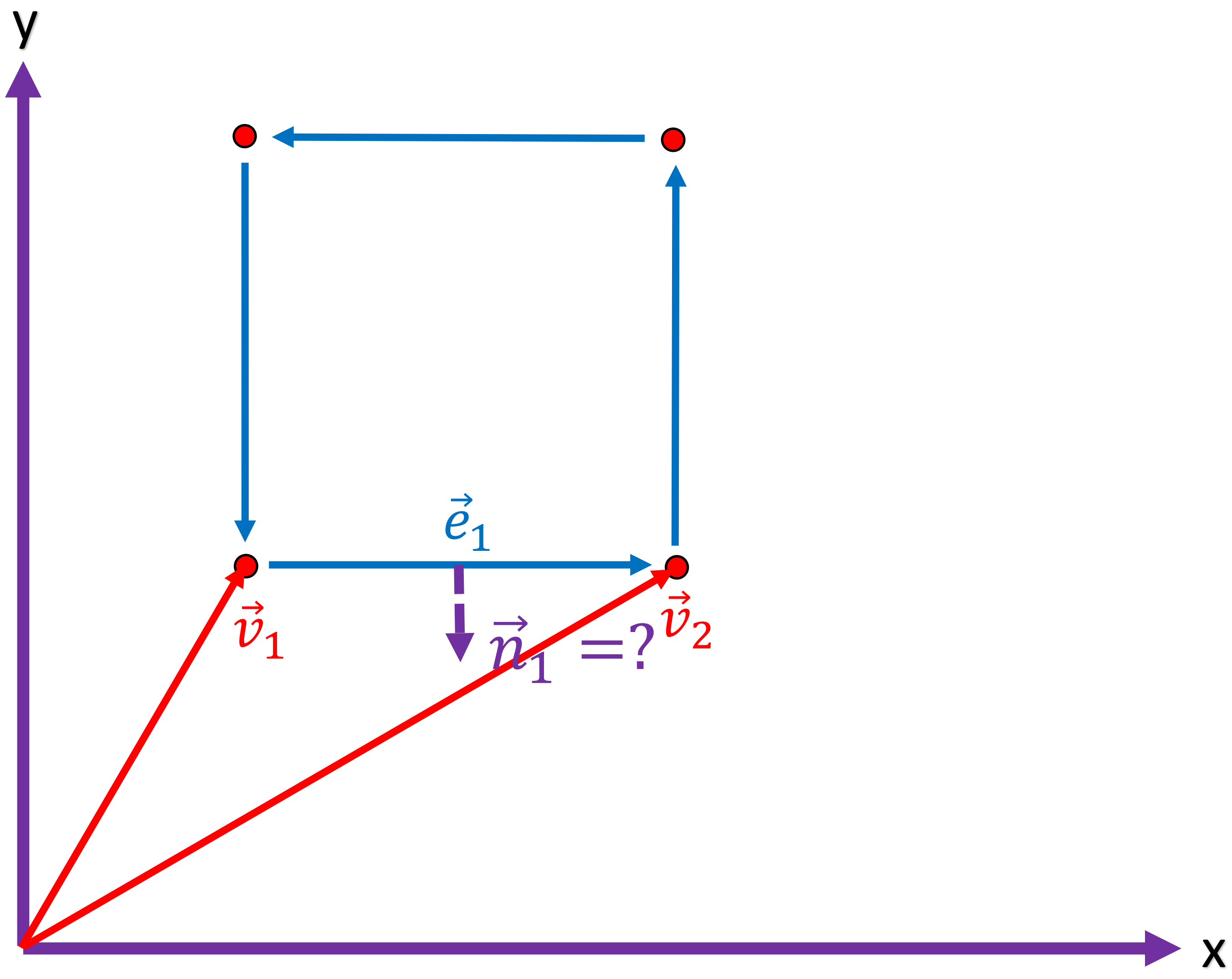
$$\vec{n} = \frac{\vec{e}_2 \times \vec{e}_1}{|\vec{e}_2 \times \vec{e}_1|}$$



Constrained motion

1. Collision detection
   - normals (2D)

$$\vec{e}_1 = \vec{v}_2 - \vec{v}_1 = (x_1, y_1)$$

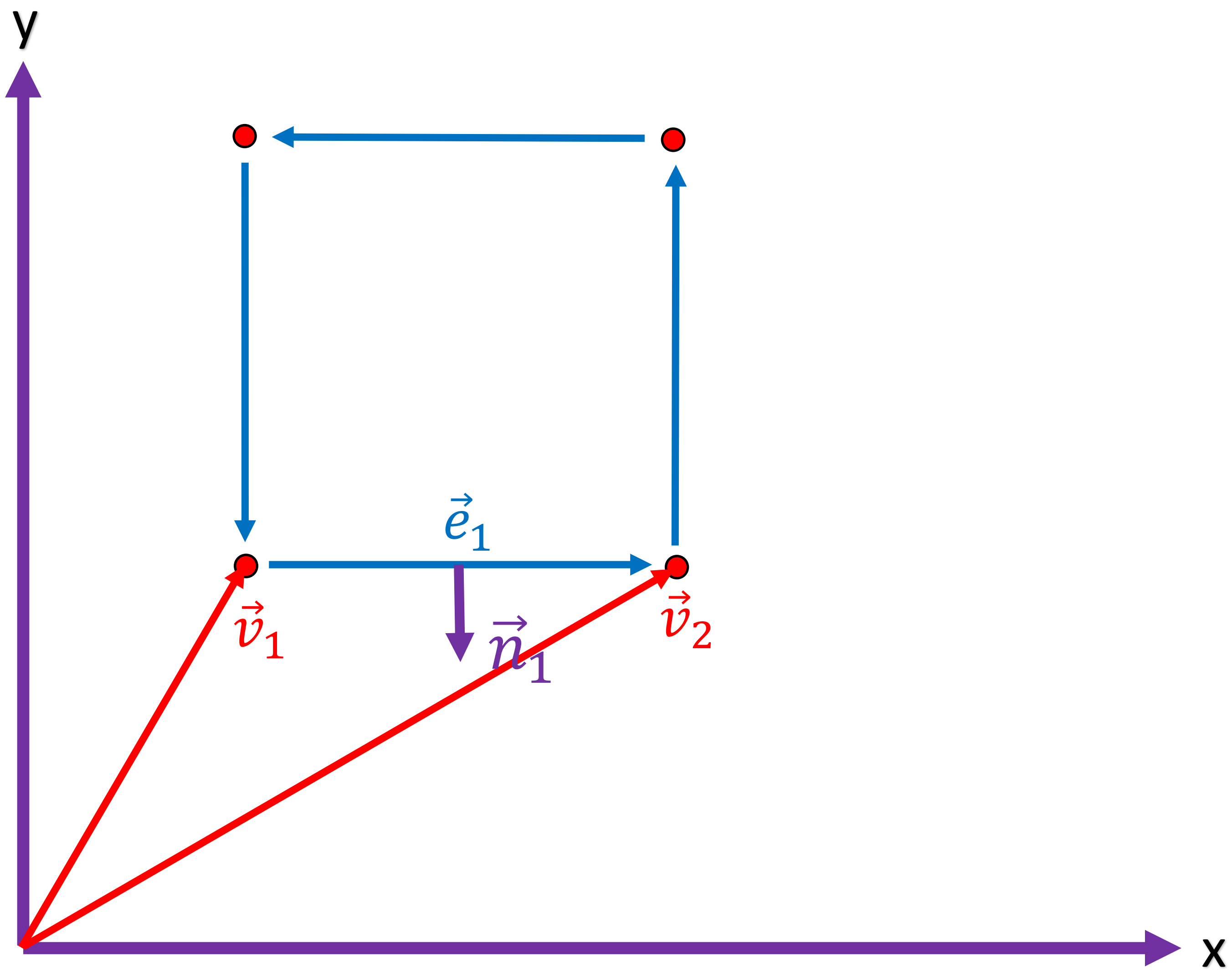


Constrained motion

1. Collision detection
   - normals (2D)

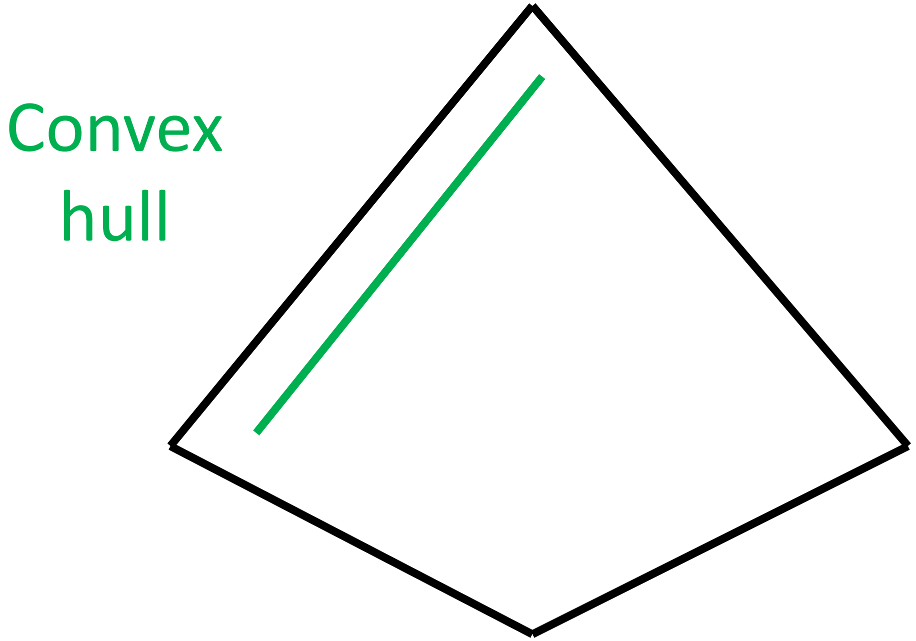
$$\vec{e}_1 = \vec{v}_2 - \vec{v}_1 = (x_1, y_1)$$

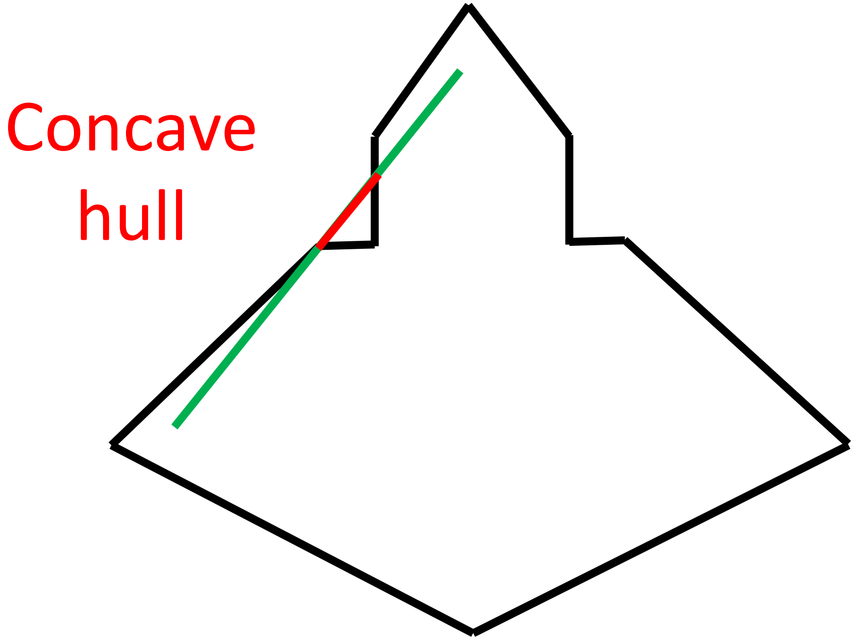
$$\vec{n}_1 = \frac{(y_1, -x_1)}{|(y_1, -x_1)|}$$



Constrained motion

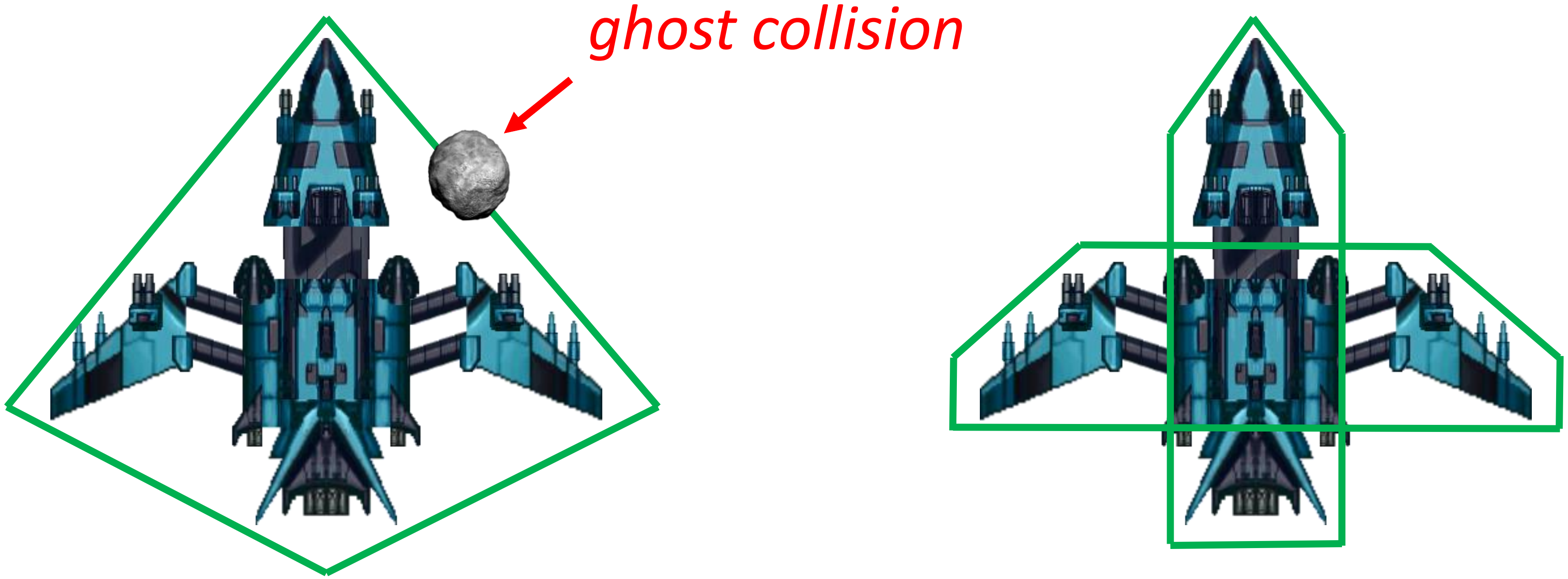
1. Collision detection
    - Convex hull:
        a) polygon (2D)
        b) polyhedron (3D)





Constrained motion

1. Collision detection
   - Convex hull:



monolithic composite

Constrained motion

1. Collision detection

algorithms:

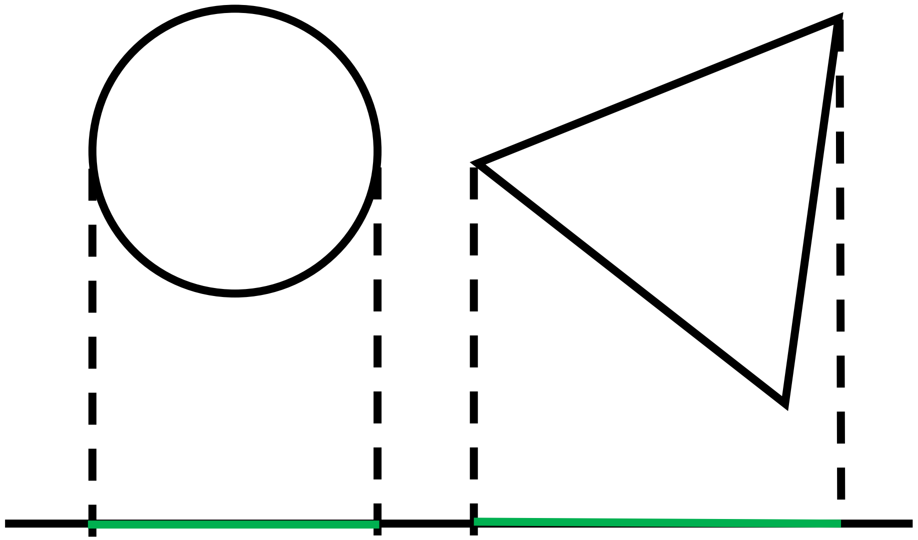
- SAT (Separating Axis Test)
- Gilbert–Johnson–Keerthi Distance Algorithm + Expanding Polytope Algorithm

\- Both use convex hulls!

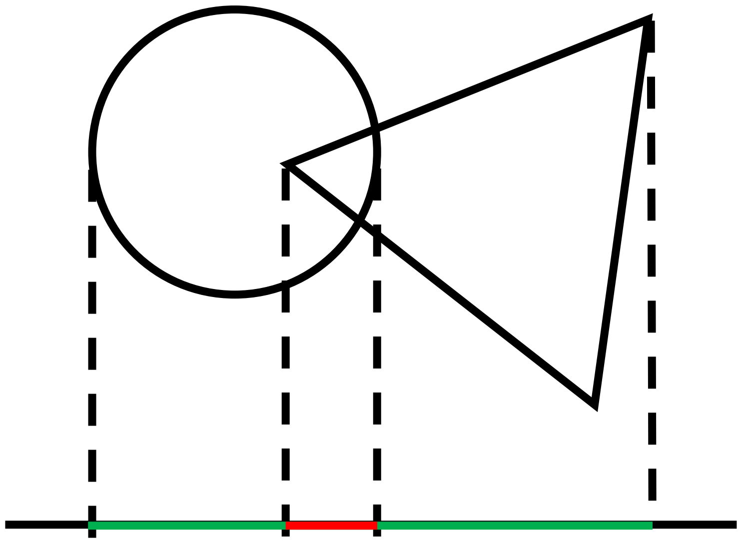
Constrained motion

1. Collision detection
   - Separating Axis Theorem:

“Two convex objects do not overlap if there exists a line (called axis) onto which the two objects' projections do not overlap.”



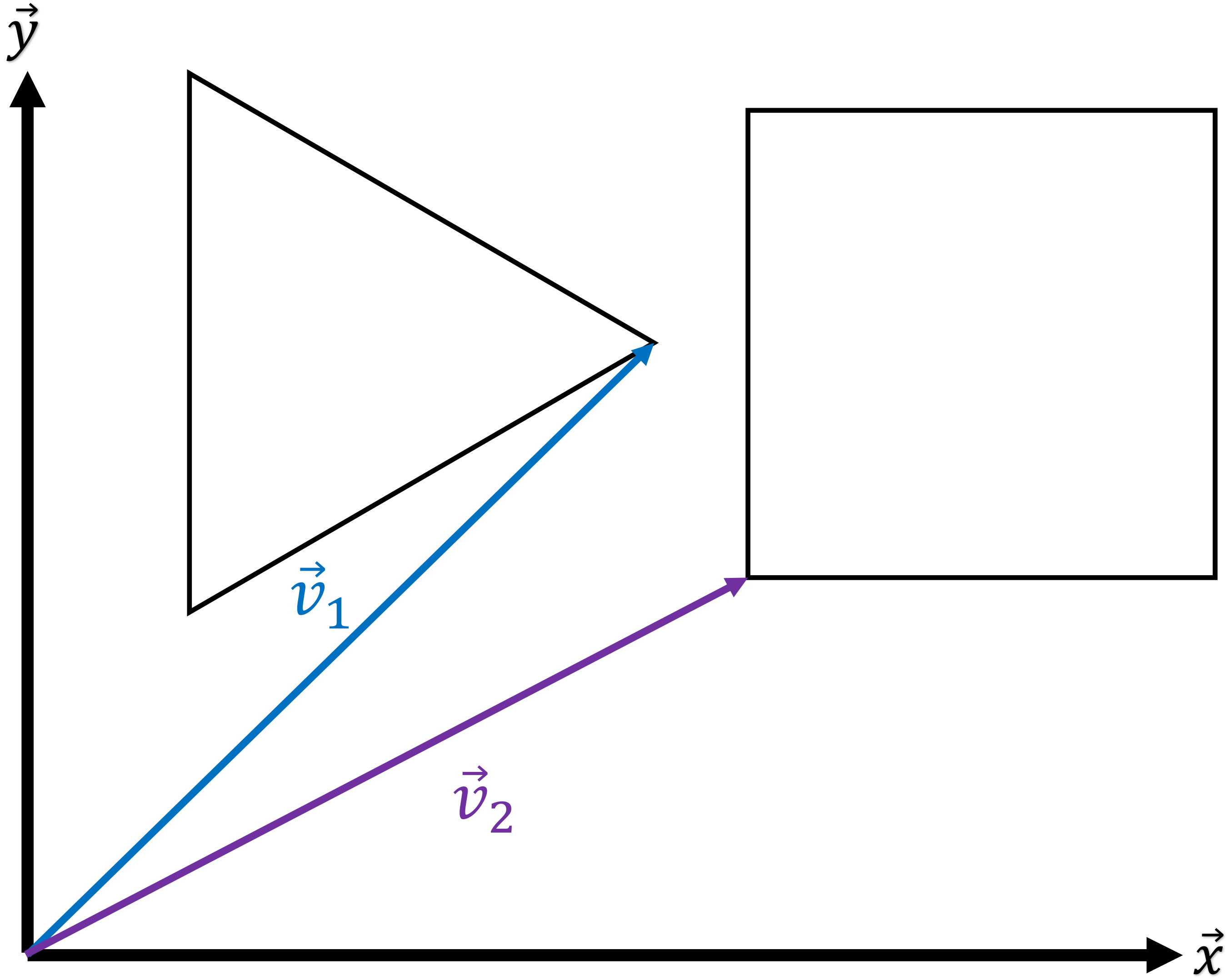
separating axis



this line is not separating axis

Constrained motion

1. Collision detection
   - Separating Axis Test:



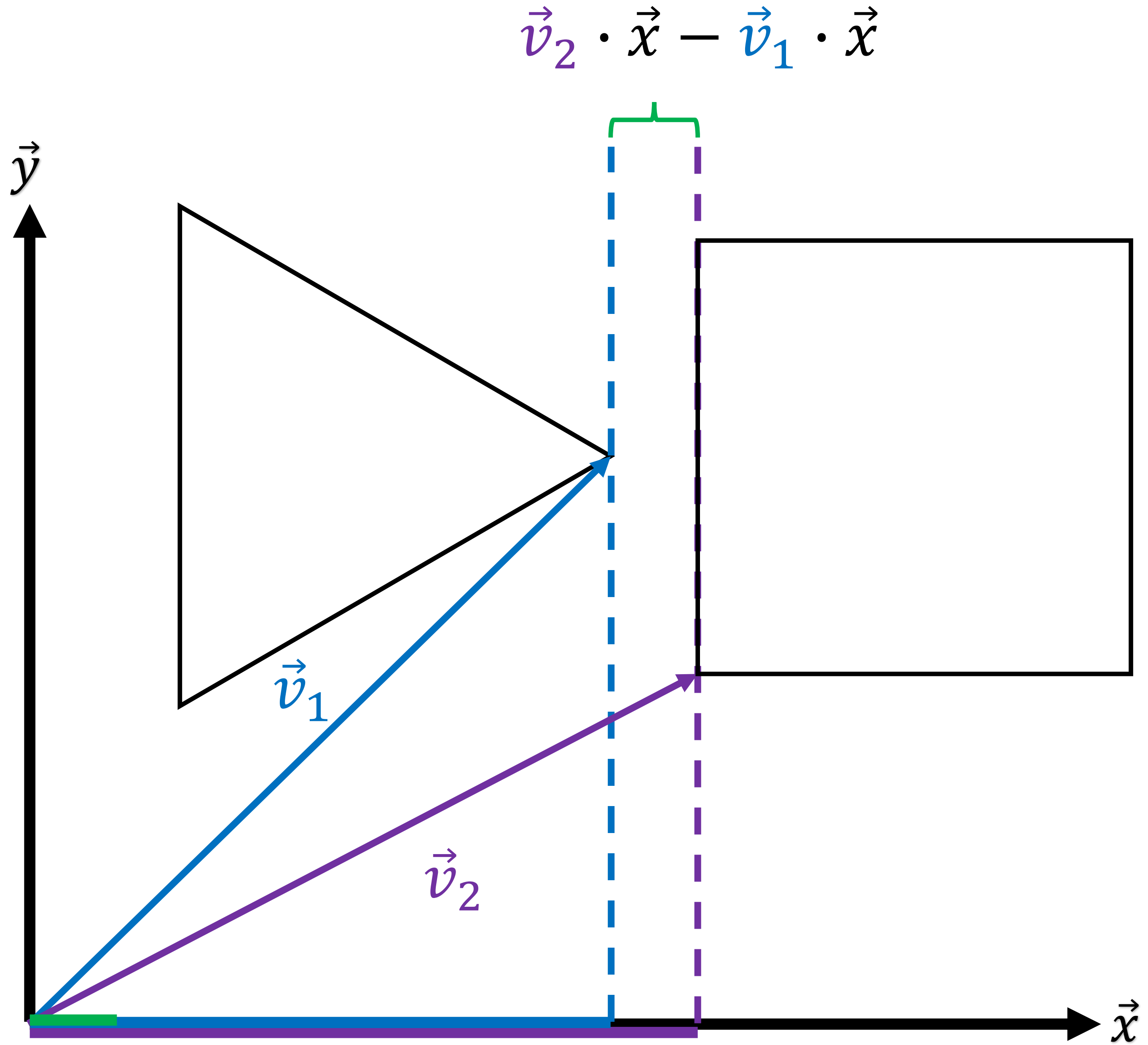
Constrained motion

1. Collision detection
   - Separating Axis Test:

Dot product

$$\vec{v}_2 \cdot \vec{x} - \vec{v}_1 \cdot \vec{x} > 0$$

Bodies do not touch each other!



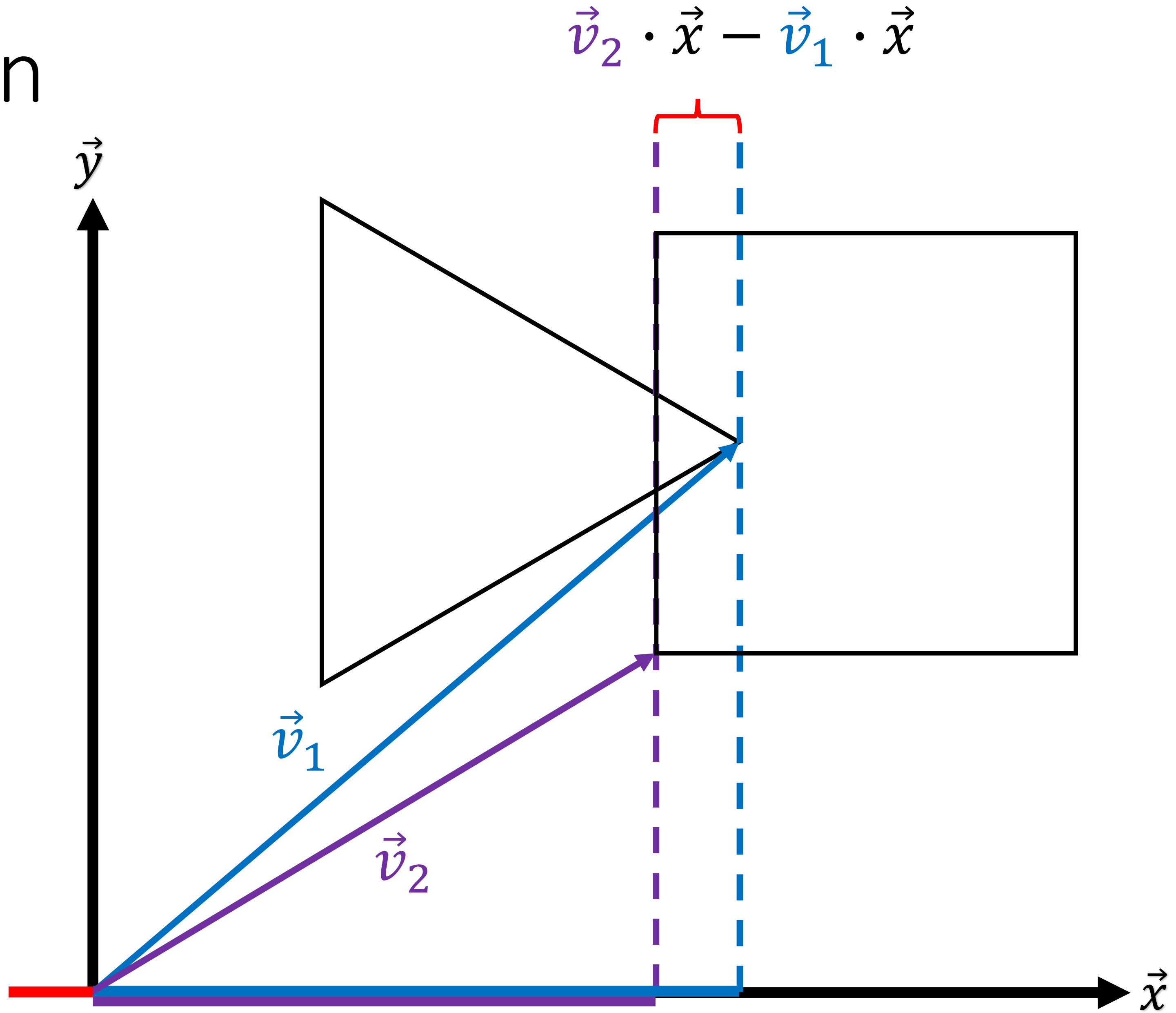
Constrained motion

1. Collision detection
   - Separating Axis Test:

Dot product

$$\vec{v}_2 \cdot \vec{x} - \vec{v}_1 \cdot \vec{x} \leq 0$$

Contact exists!

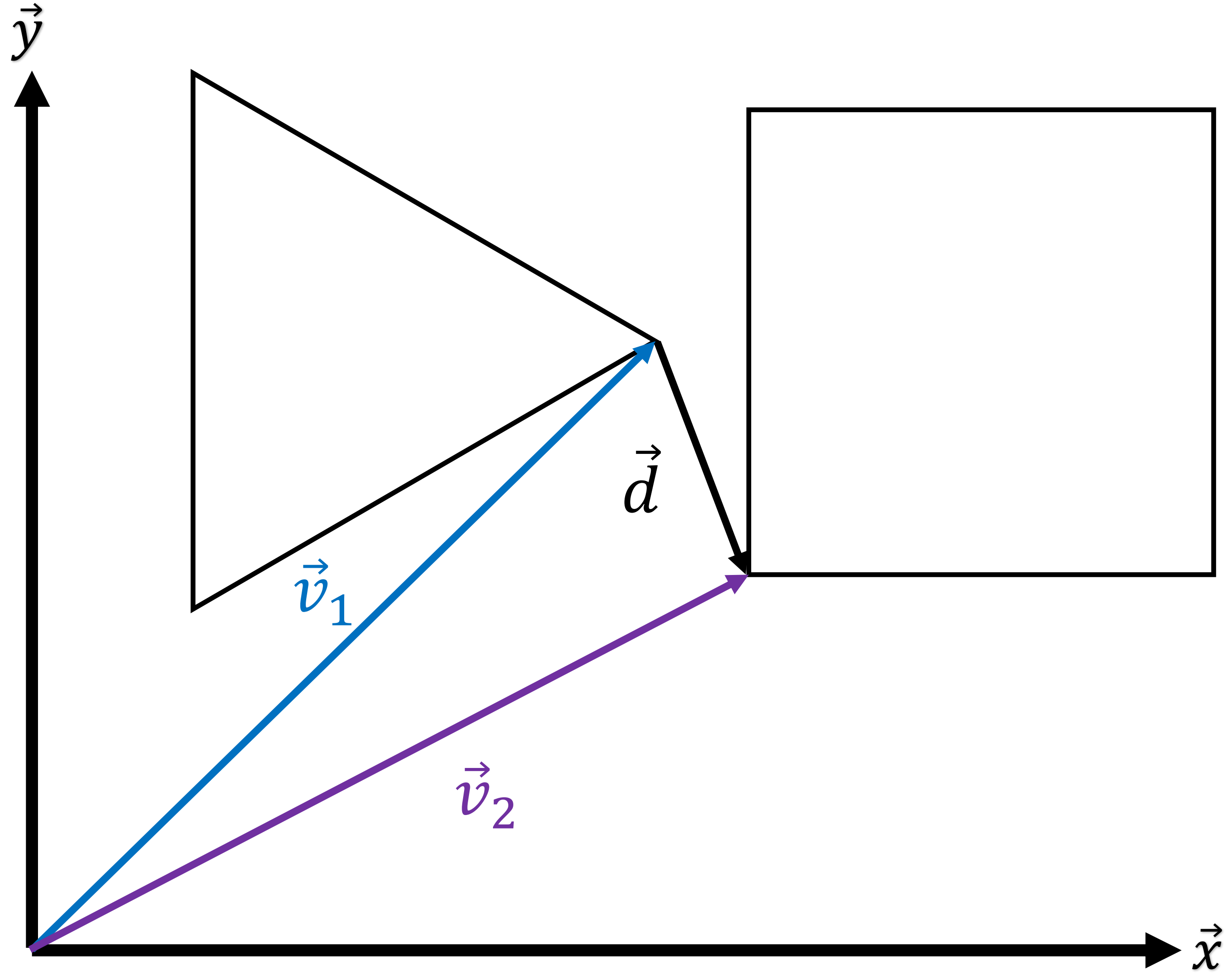


Constrained motion

1. Collision detection
   - Separating Axis Test:

Distance vector:

$$\vec{d} = \vec{v}_2 - \vec{v}_1$$

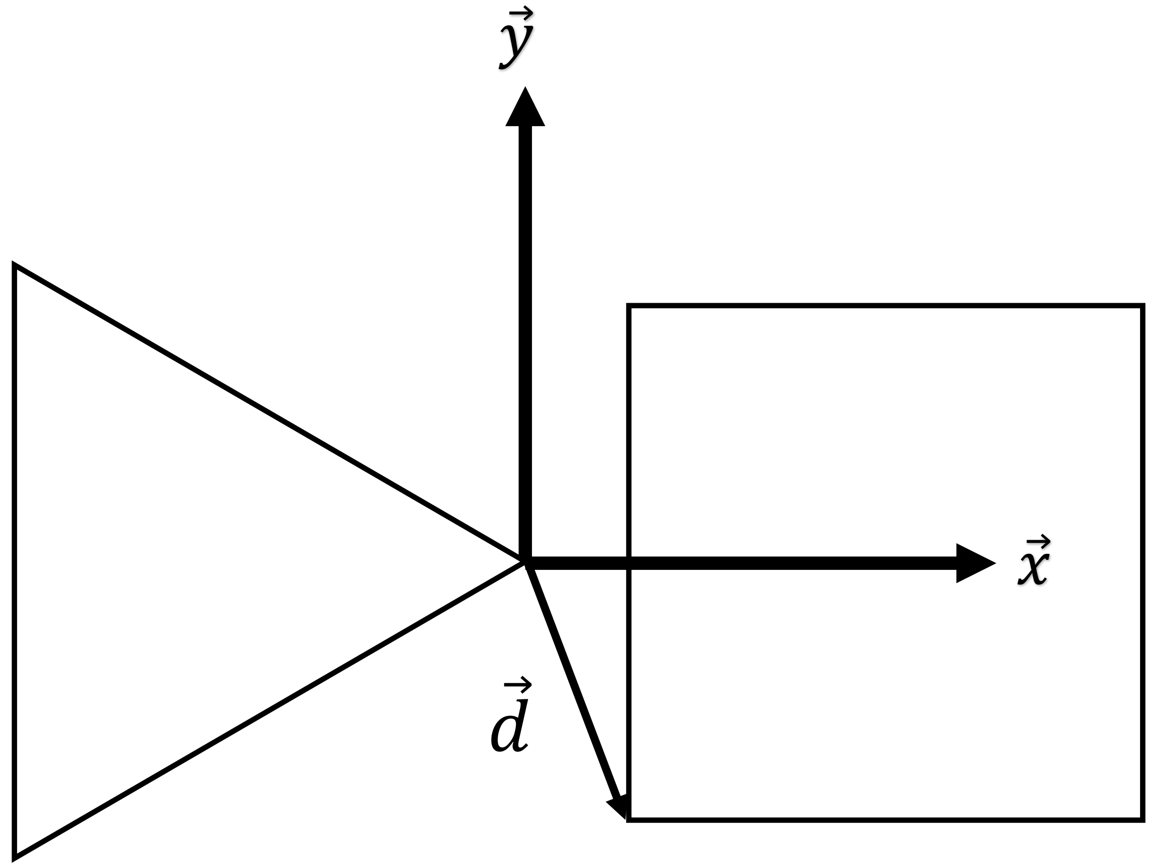


Constrained motion

1. Collision detection
   - Separating Axis Test:

Distance vector:

$$\vec{d} = \vec{v}_2 - \vec{v}_1$$



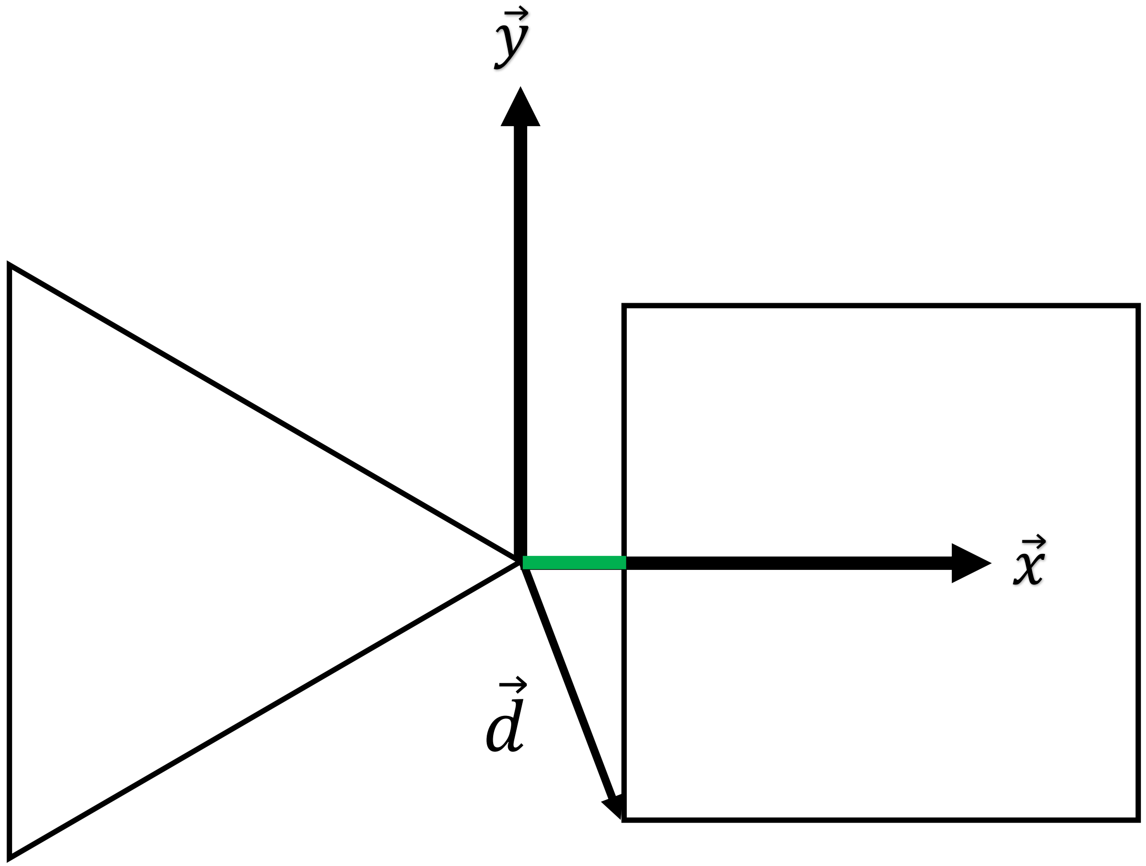
Constrained motion

1. Collision detection
   - Separating Axis Test:

Dot product

$$\vec{d} \cdot \vec{x} > 0$$

Bodies do not touch each other!



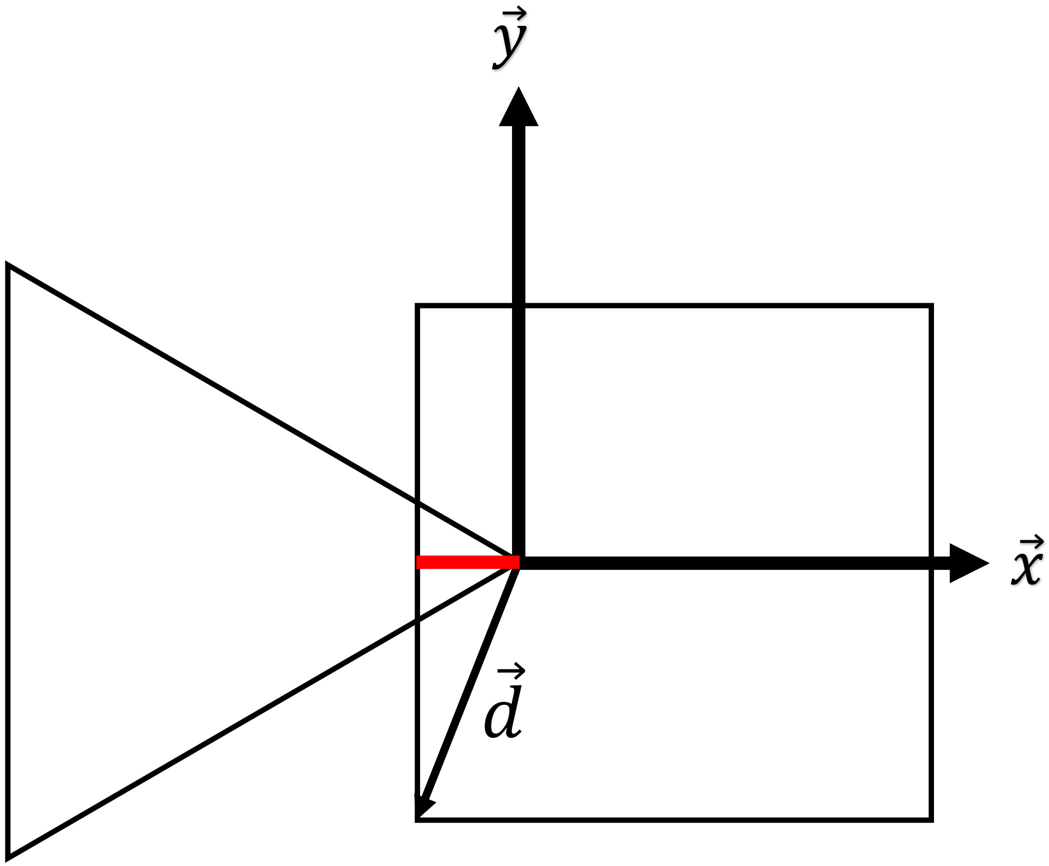
Constrained motion

1. Collision detection
   - Separating Axis Test:

Dot product

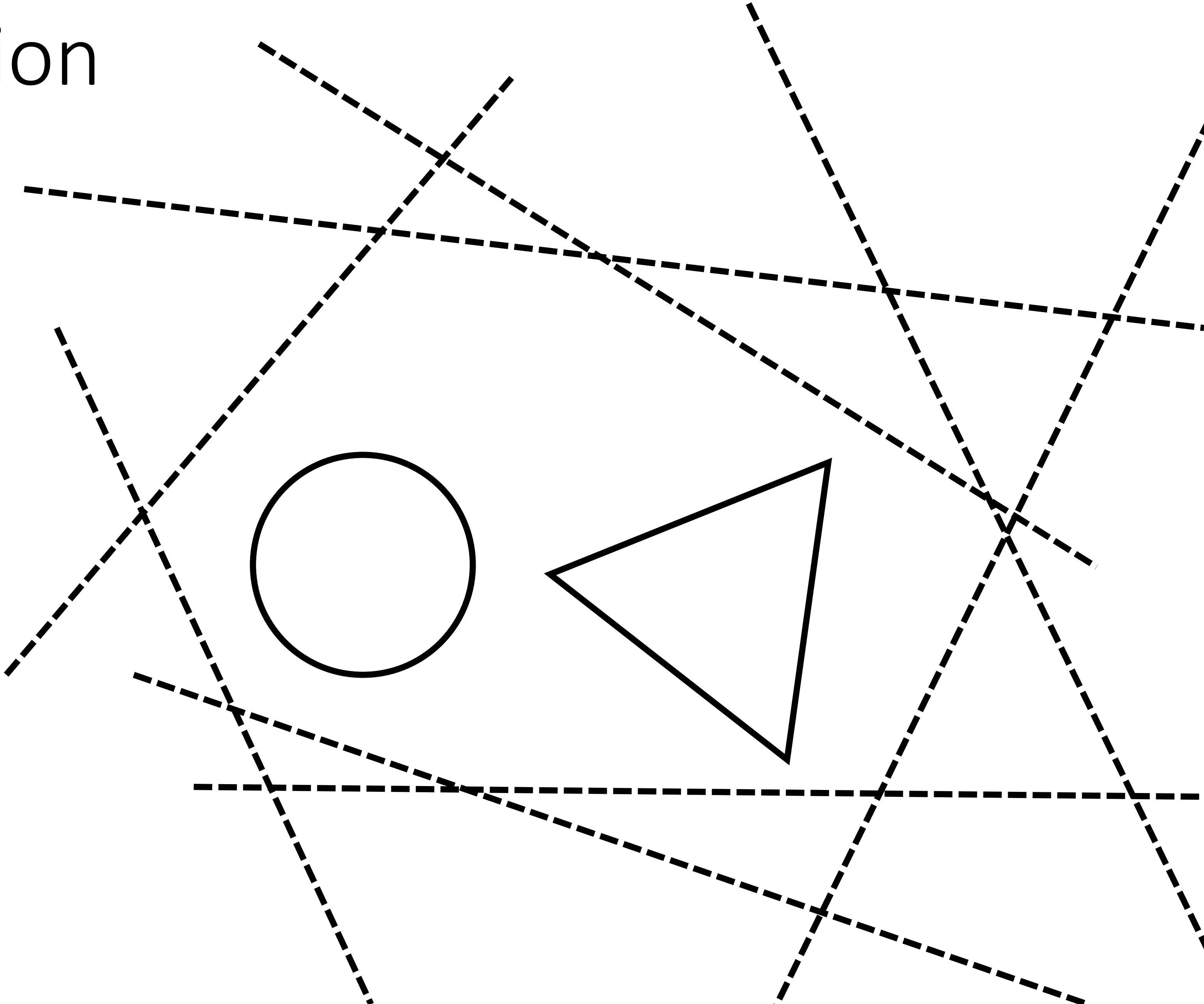
$$\vec{d} \cdot \vec{x} \leq 0$$

Contact exists!



Constrained motion

1. Collision detection
    - Separating Axis Test:

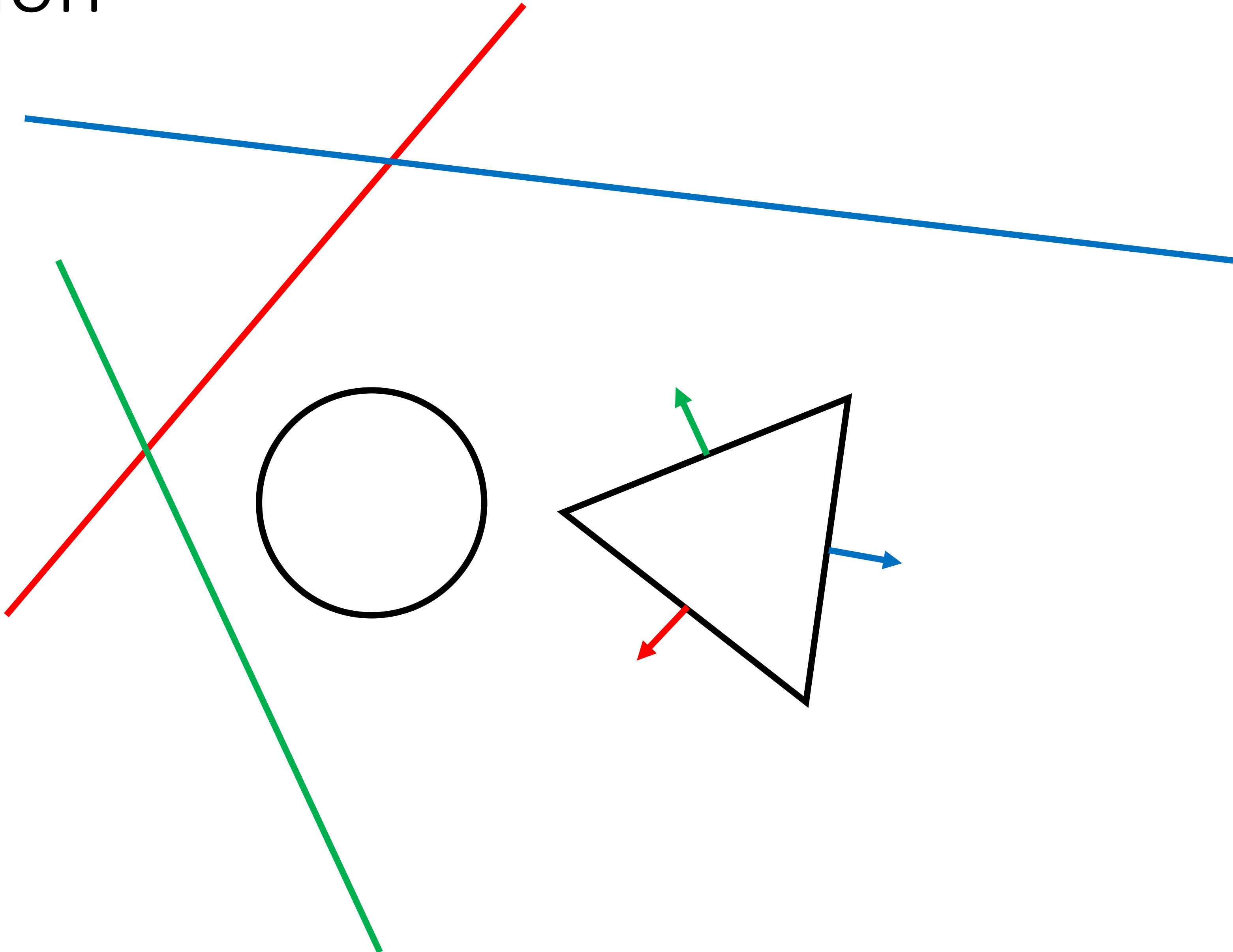


How many axes exist?

Against how many axes should test be performed?

Constrained motion

1. Collision detection
    - Separating Axis Test:



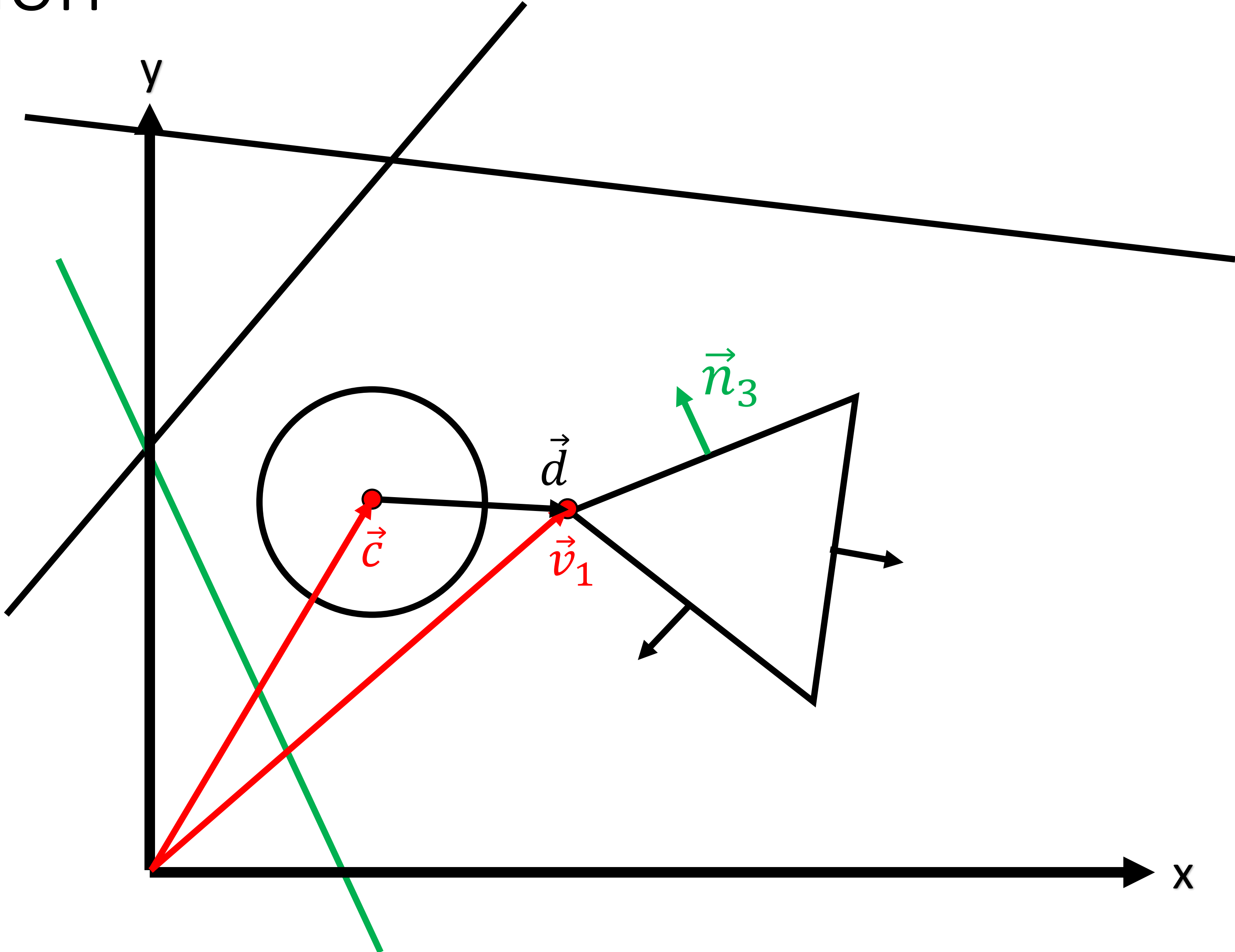
In 2D space it is sufficient to test against convex hull normals!

Constrained motion

1. Collision detection
   - Separating Axis Test:

$$\vec{d} = \vec{v}_1 - \vec{c}$$

Geometries of both bodies must be transformed to the same coordinate system!

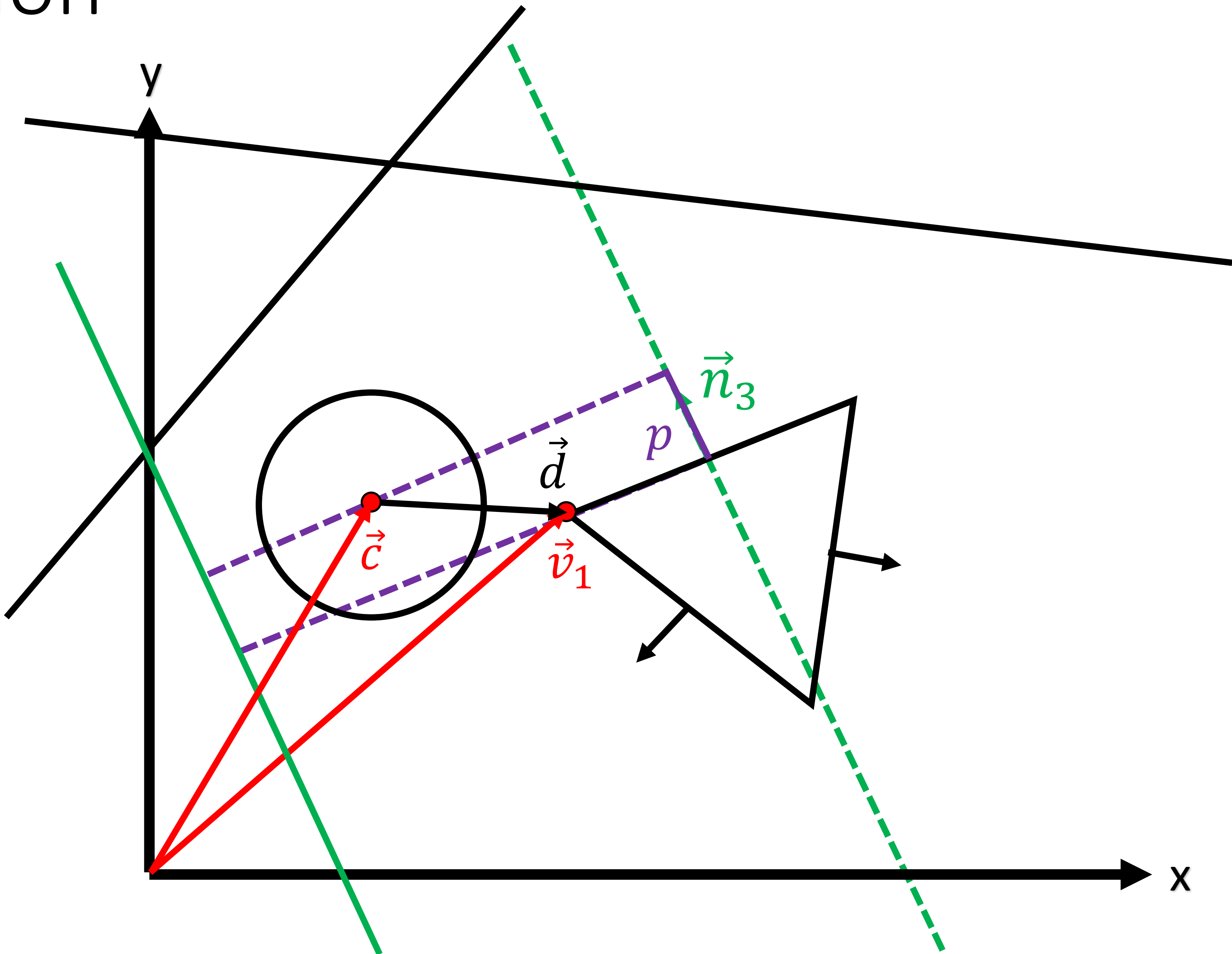


Constrained motion

1. Collision detection
   - Separating Axis Test:

$$\vec{d} = \vec{v}_1 - \vec{c}$$

$$p = \vec{n}_3 \cdot \vec{d}$$



Constrained motion

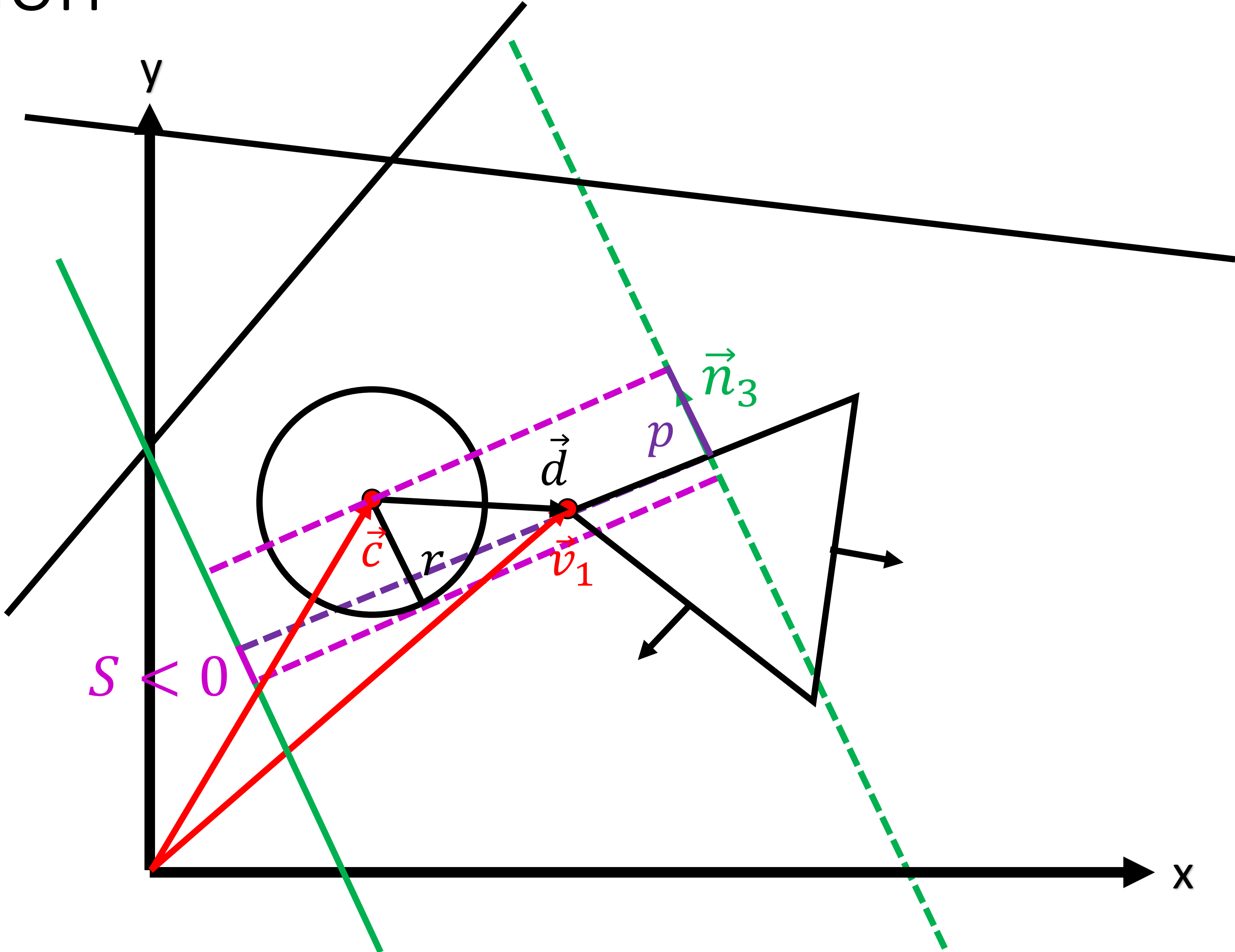
1. Collision detection
   - Separating Axis Test:

$$\vec{d} = \vec{v}_1 - \vec{c}$$

$$p = \vec{n}_3 \cdot \vec{d}$$

$$s = p - \mathrm{r}$$

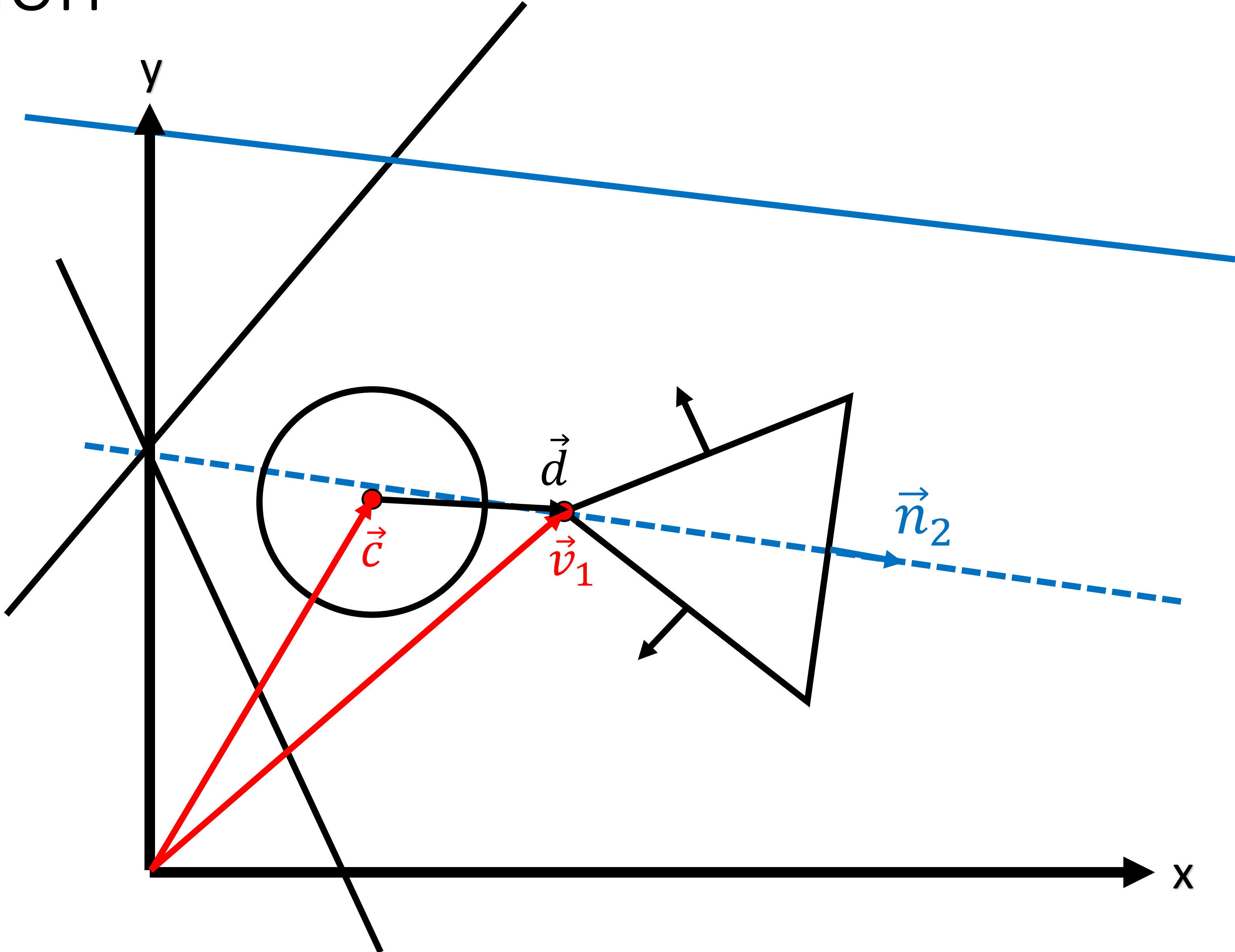
This normal is not a separating axis!



Constrained motion

1. Collision detection
   - Separating Axis Test:

$$\vec{d} = \vec{v}_1 - \vec{c}$$

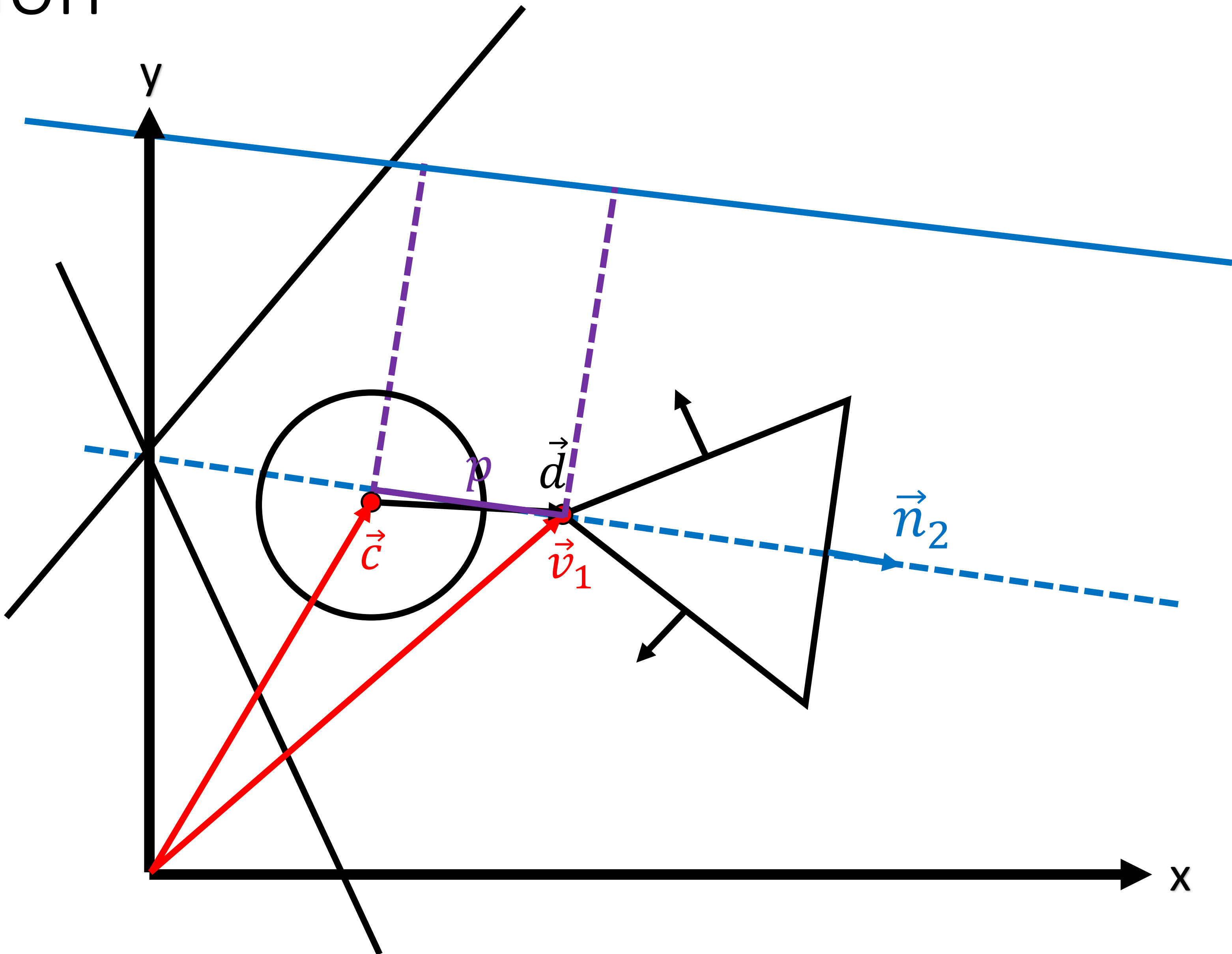


Constrained motion

1. Collision detection
   - Separating Axis Test:

$$\vec{d} = \vec{v}_1 - \vec{c}$$

$$p = \vec{n}_2 \cdot \vec{d}$$



Constrained motion

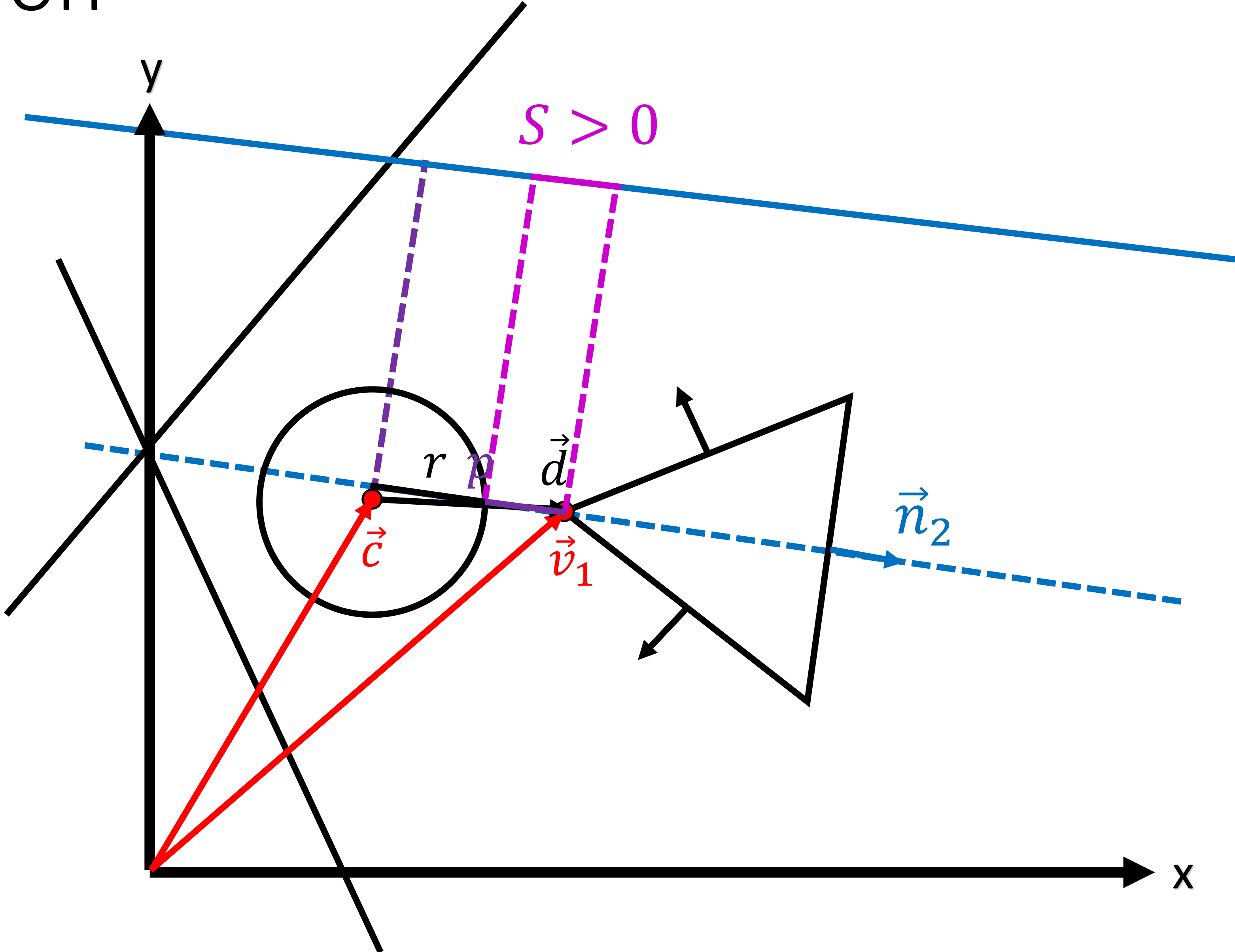
1. Collision detection
   - Separating Axis Test:

$$\vec{d} = \vec{v}_1 - \vec{c}$$

$$\vec{p} = \vec{n}_2 \cdot \vec{d}$$

$$s = p - \mathrm{r}$$

This normal is separating axis!
No further testing is needed!

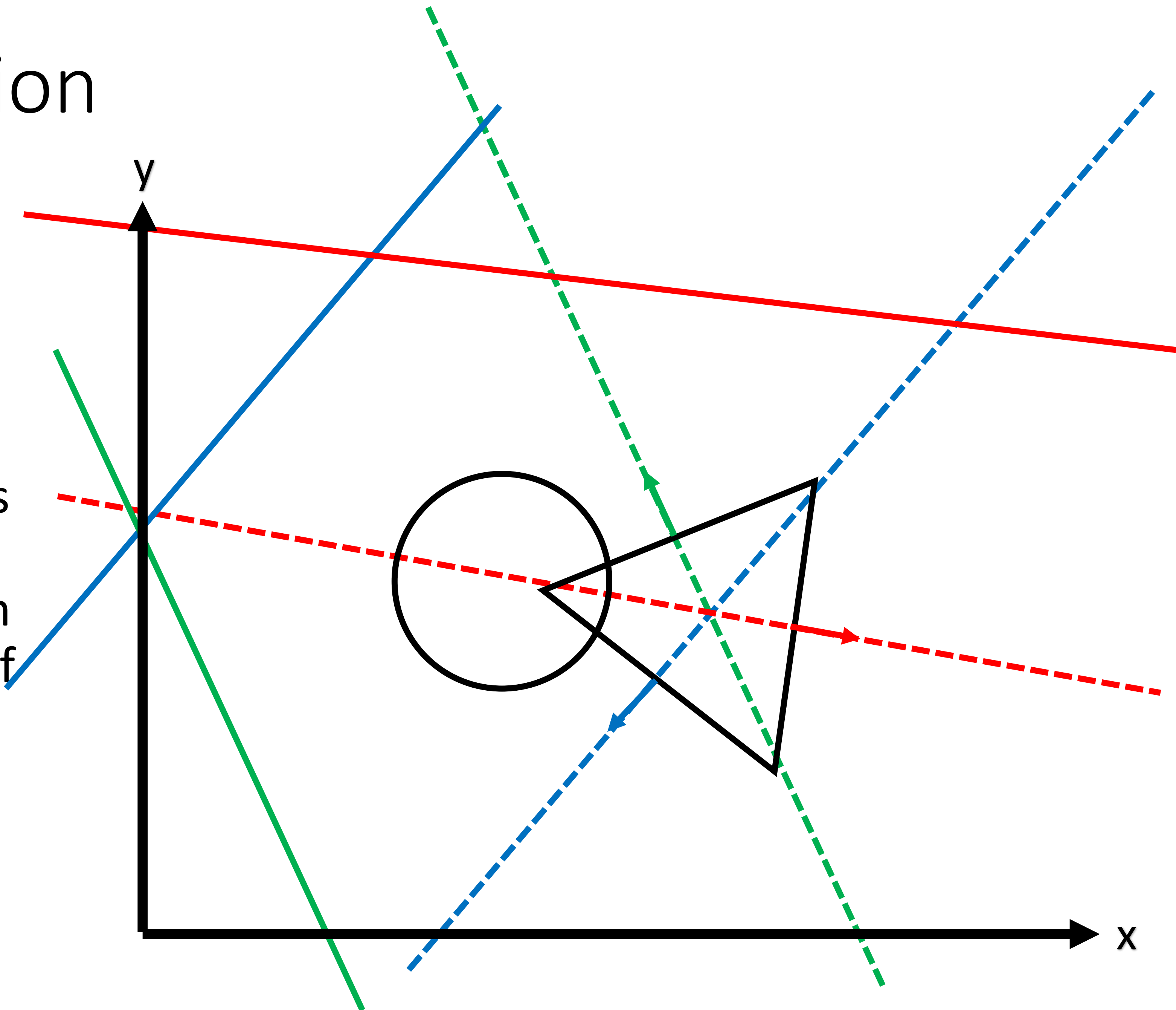


Constrained motion

1. Collision detection
   - Separating Axis Test:

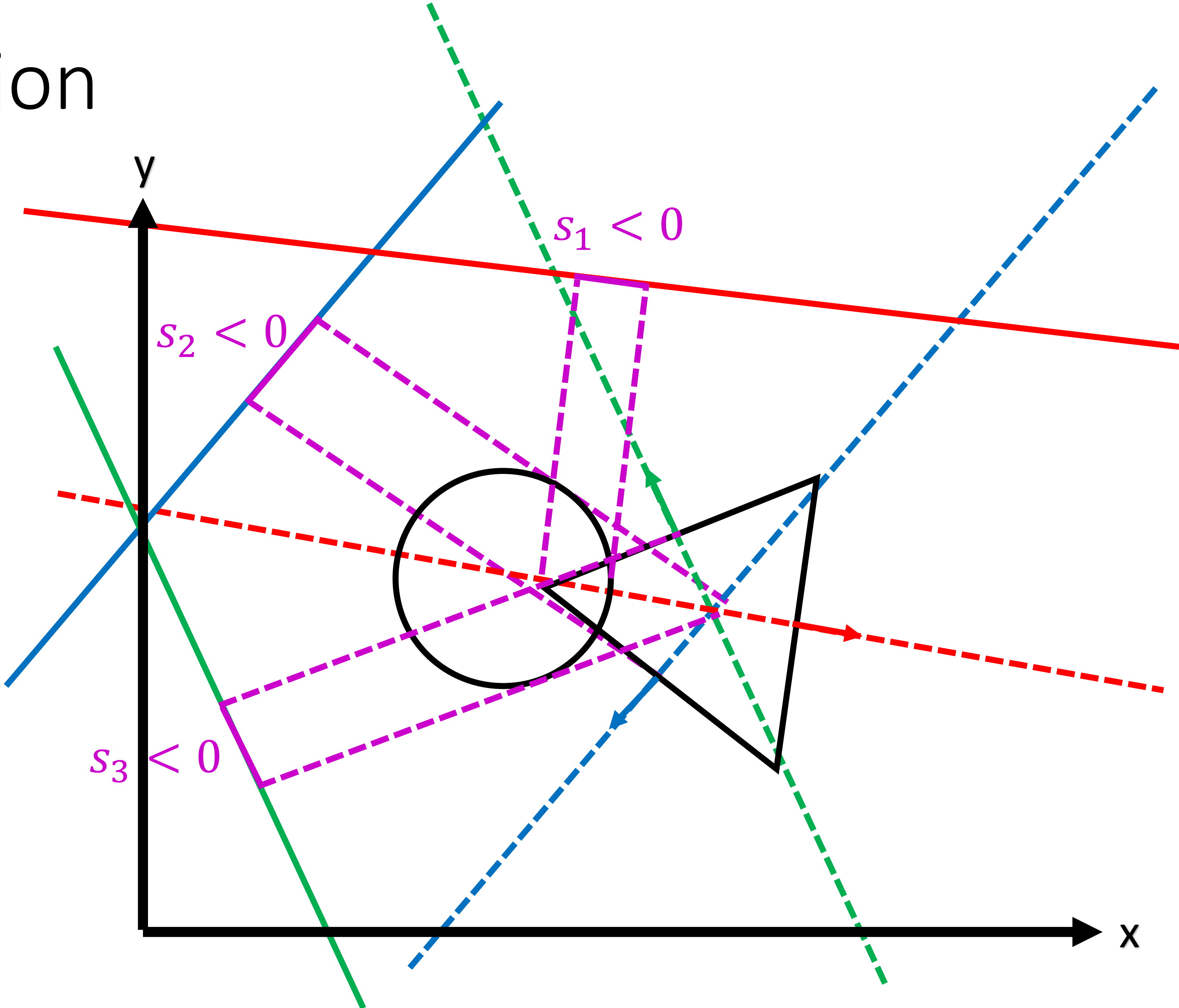
Separating axis test is performed on all the vertices of one body against all the vertices of the other body on all the convex hull normals of both bodies.

What if separating axis isn't found?



Constrained motion

1. Collision detection
   - Separating Axis Test:

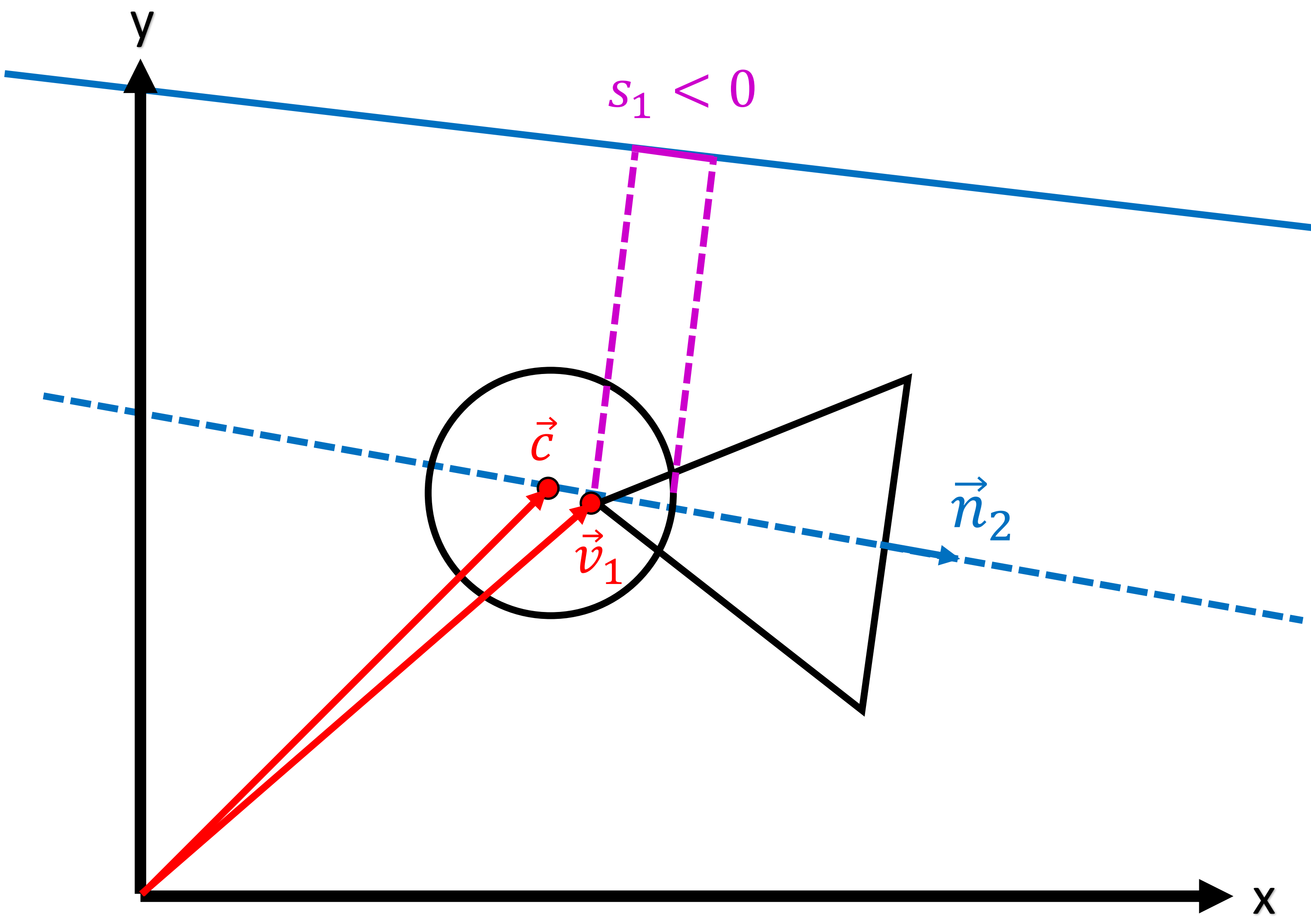


1. Negative separation most closer to 0 is used to describe the contact s (the one whose absolute value is the smallest).

Constrained motion

1. Collision detection
   - Separating Axis Test:

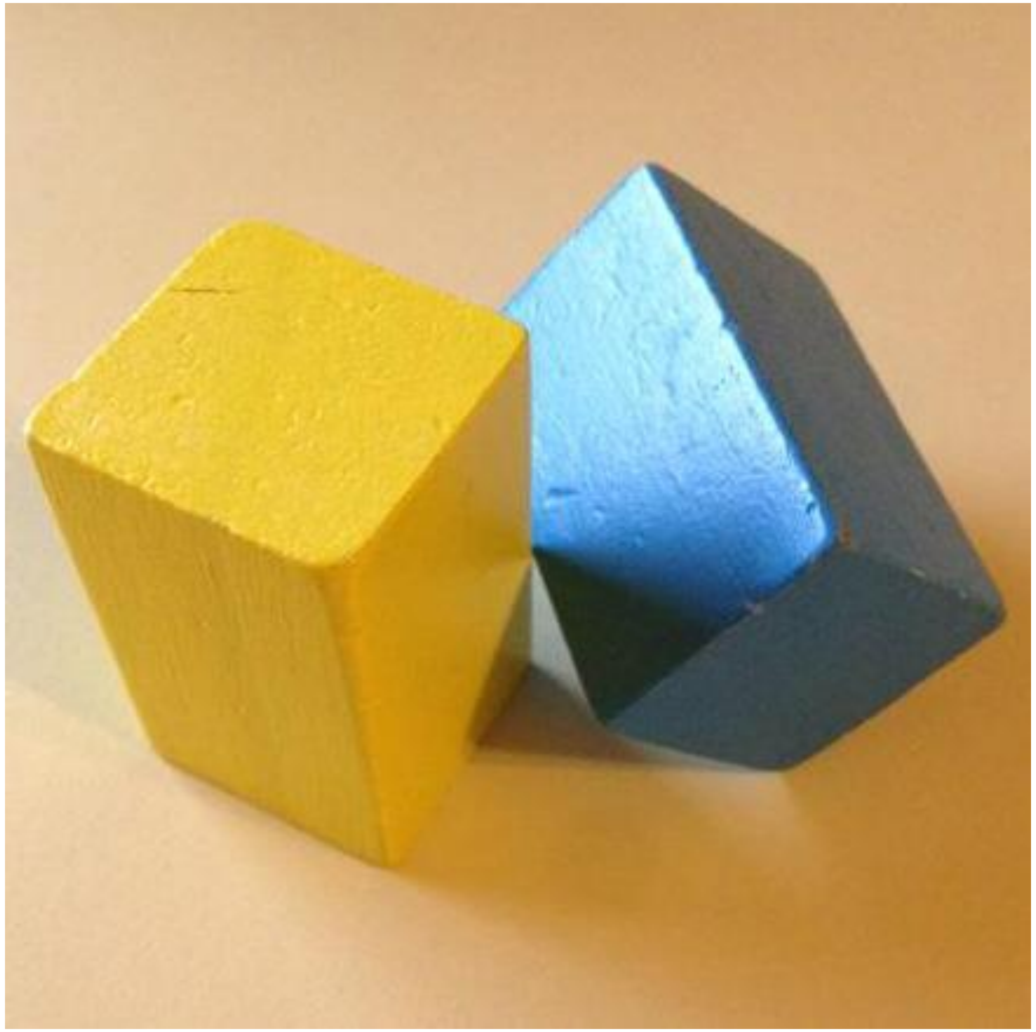
1. Negative separation most closer to 0 is used to describe the contact s (the one whose absolute value is the smallest).
2. Contact is described by:
   - contact normal (separating axis)
   - negative separation (penetration)
   - vertices of both bodies



Constrained motion

1. Collision detection
   - Separating Axis Test:

In 3D space testing is needed against cross products of all the face normals too!



Constrained motion

1. Collision detection
   - Separating Axis Test:

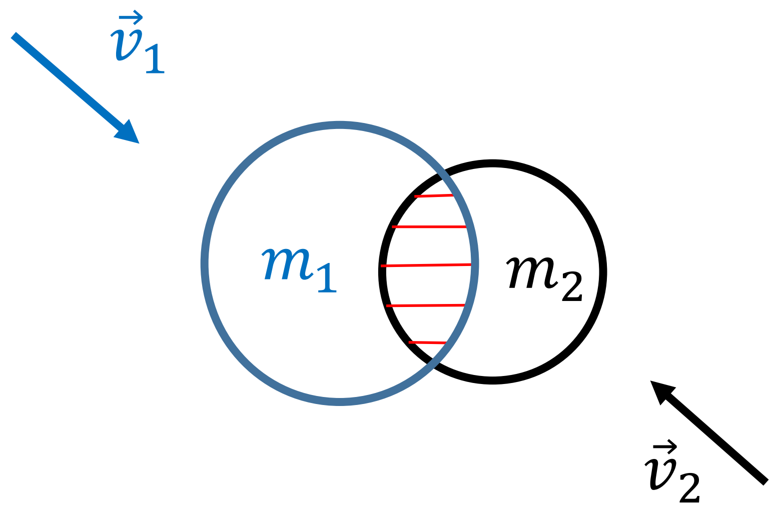
+ explicit geometrical approach to contact detection
+ everything needed to describe the contacts is calculated in the detection phase
+ as soon as the separating axis is found, further testing can be skipped (efficient in many cases).

- many special cases need to be considered, especially in 3D space

Constrained motion

Constraint solving

Naive approach #1 (for simplicity sake, only particles are considered in this example):

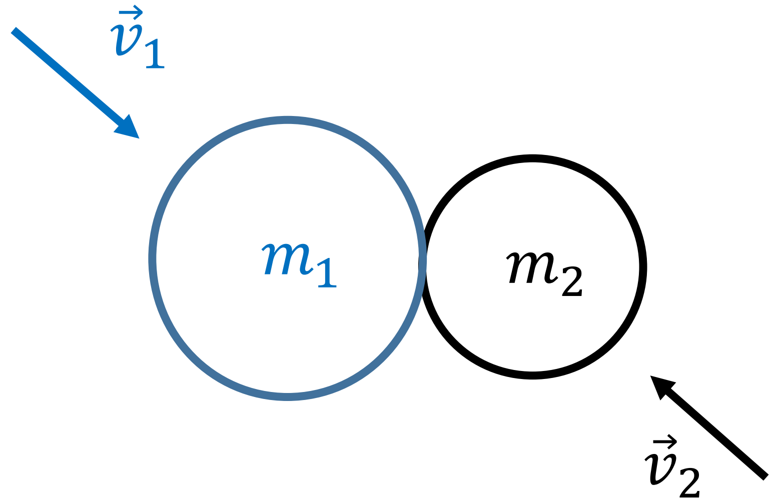


Constrained motion

Constraint solving

Naive approach #1 (for simplicity sake, only particles are considered in this example):

1. position correction against contact normal



Constrained motion

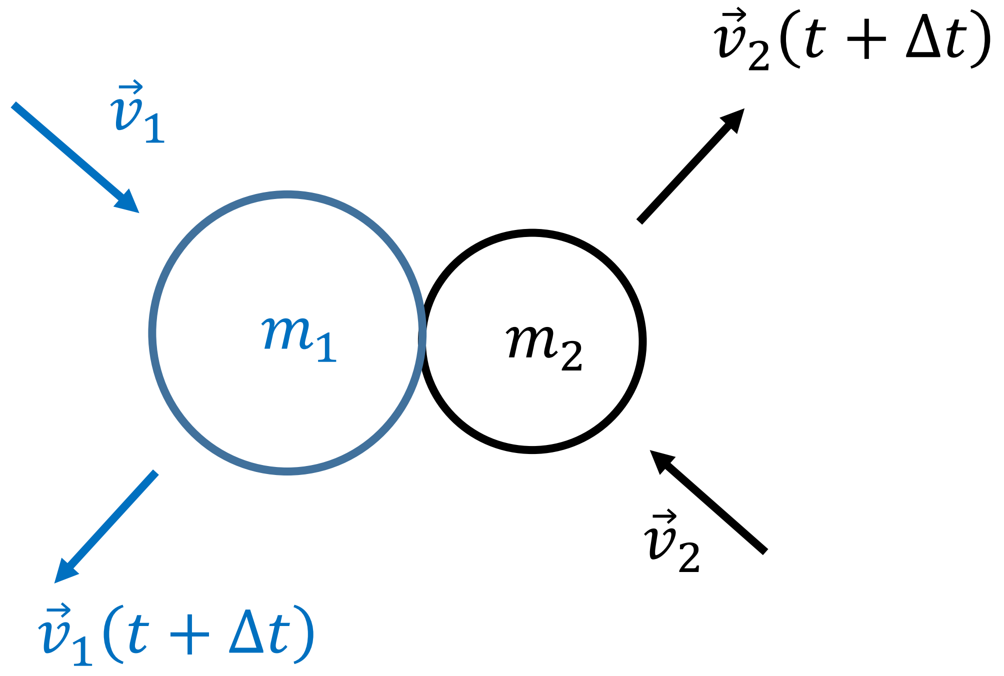
Constraint solving

Naive approach #1 (for simplicity sake, only particles are considered in this example):

1. position correction against contact normal
2. velocity update (conservation of momentum)

$$\vec{v}_1(t+\Delta t) = \frac{\vec{v}_1(m_1 - m_2) + 2m_2\vec{v}_2}{m_1 + m_2}$$

$$\vec{v}_2(t+\Delta t) = \frac{\vec{v}_2(m_2 - m_1) + 2m_1\vec{v}_1}{m_1 + m_2}$$

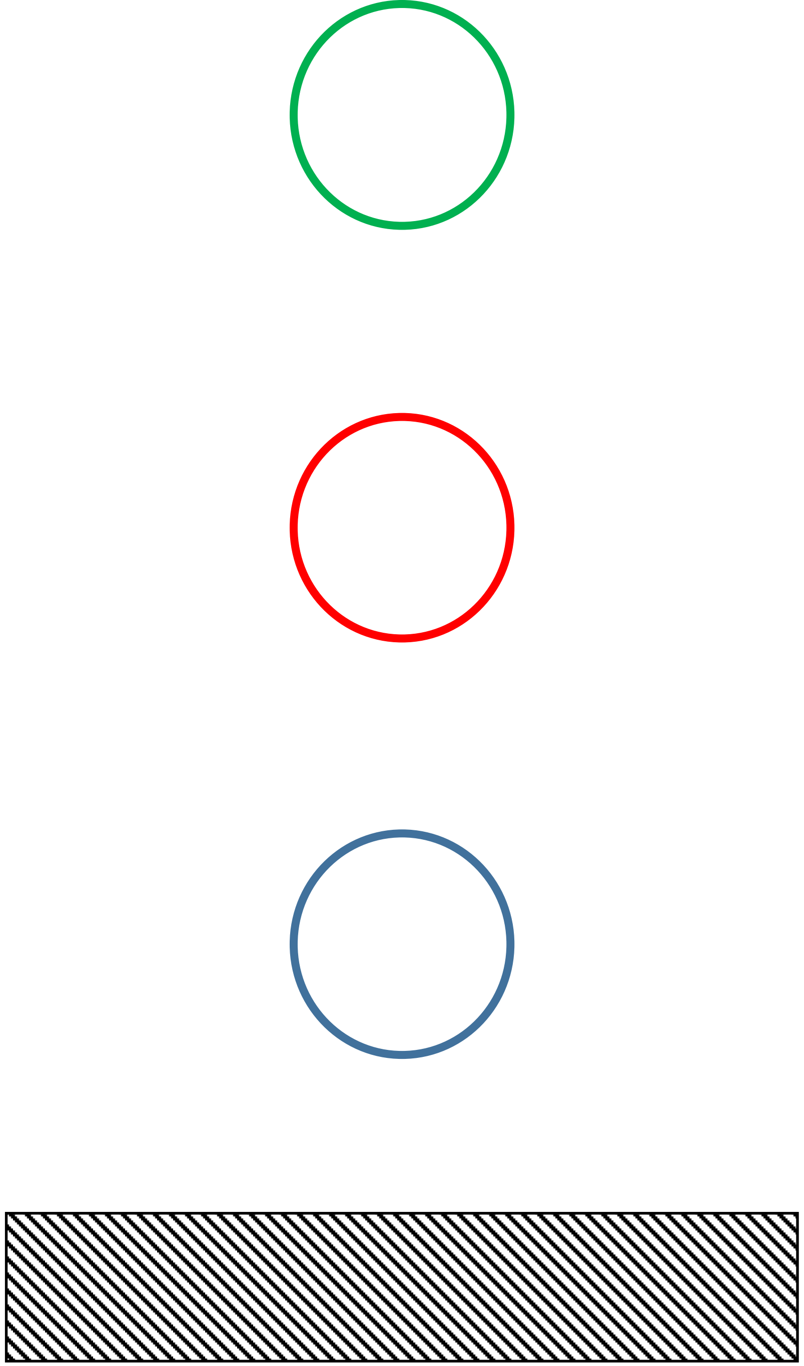


Constrained motion

Constraint solving

Naive approach #1:

```
while true
    applyExternalForces();
    integrate();
    if constraintsViolated()
        solveConstraints();
    end
    render();
    sleep();
end
```

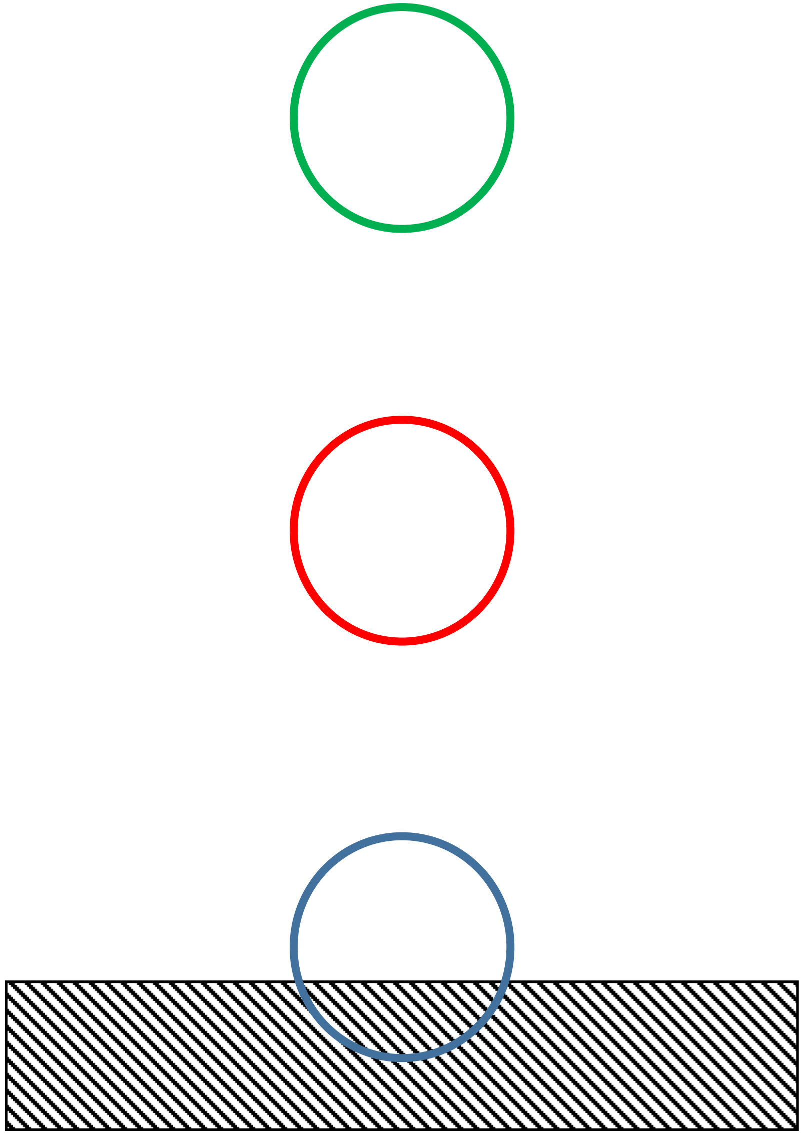


Constrained motion

Constraint solving

Naive approach #1:

```
while true
    applyExternalForces();
    integrate();
    if constraintsViolated()
        solveConstraints();
    end
    render();
    sleep();
end
```

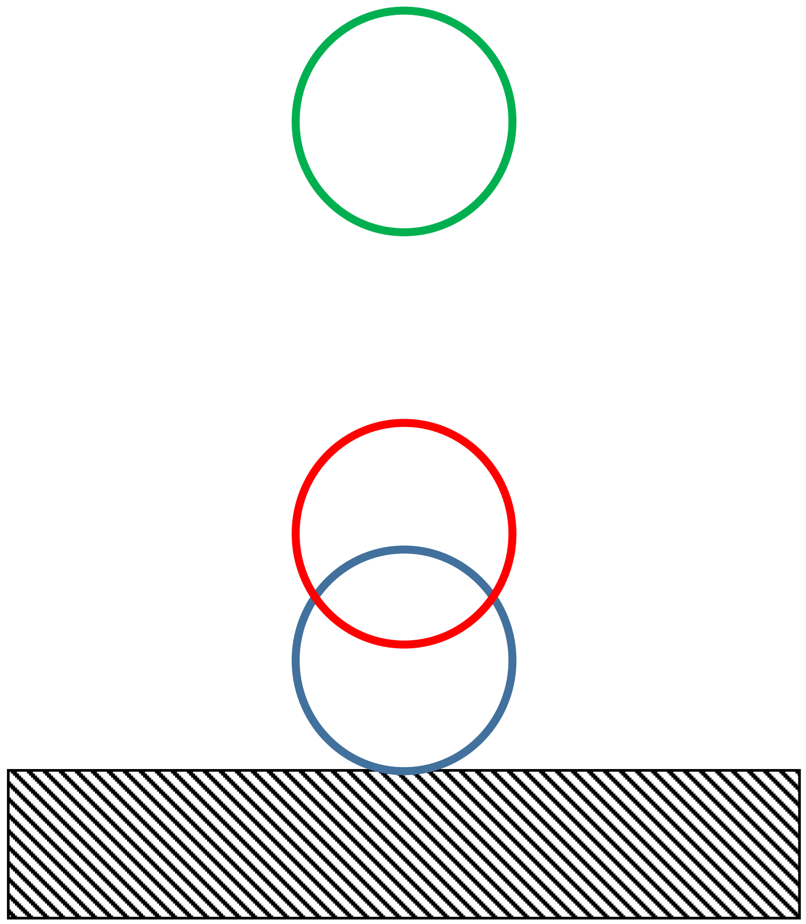


Constrained motion

Constraint solving

Naive approach #1:

```
while true
    applyExternalForces();
    integrate();
    if constraintsViolated()
        solveConstraints();
    end
    render();
    sleep();
end
```

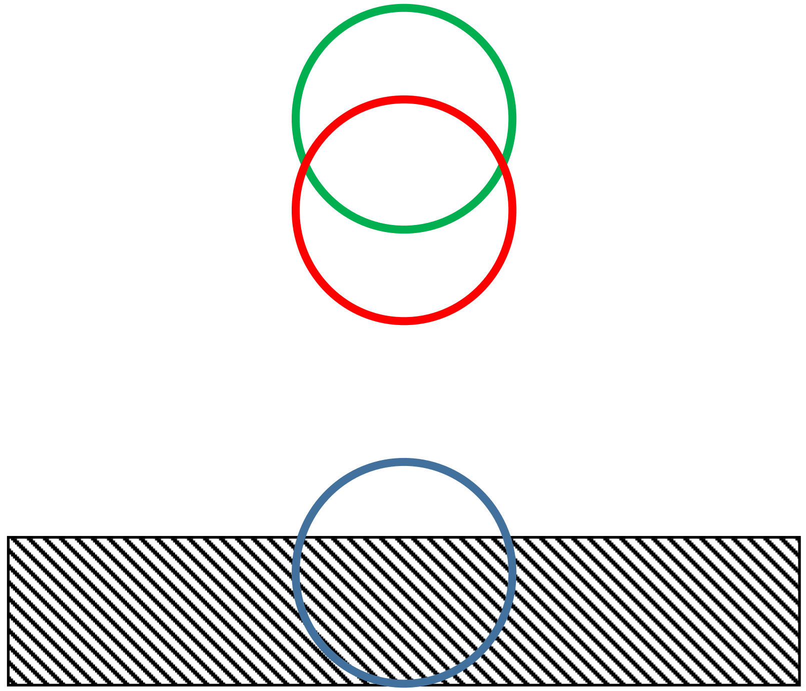


Constrained motion

Constraint solving

Naive approach #1:

```
while true
    applyExternalForces();
    integrate();
    if constraintsViolated()
        solveConstraints();
    end
    render();
    sleep();
end
```

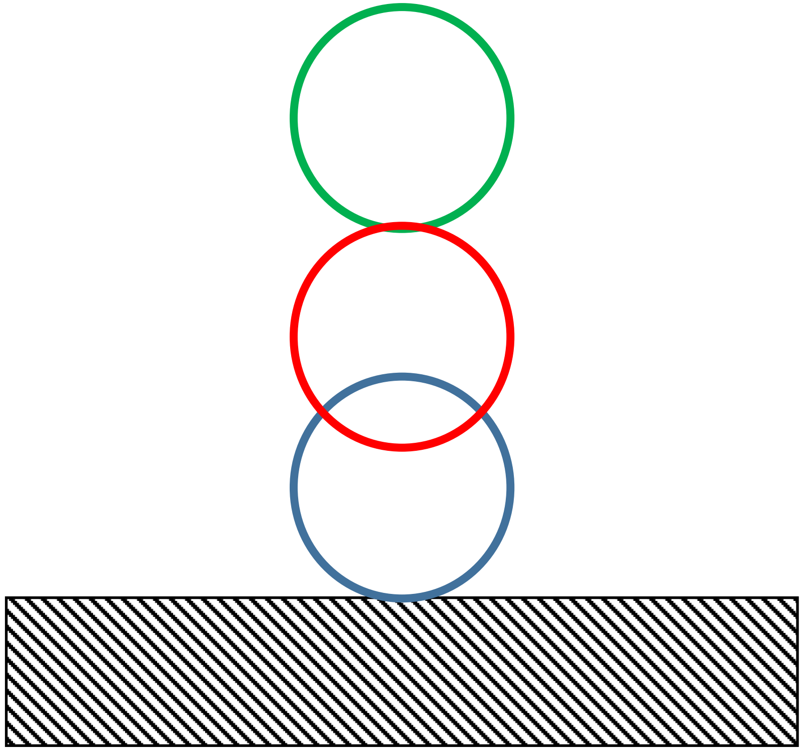


Constrained motion

Constraint solving

Naive approach #1:

```
while true
    applyExternalForces();
    integrate();
    if constraintsViolated()
        solveConstraints();
    end
    render();
    sleep();
end
```

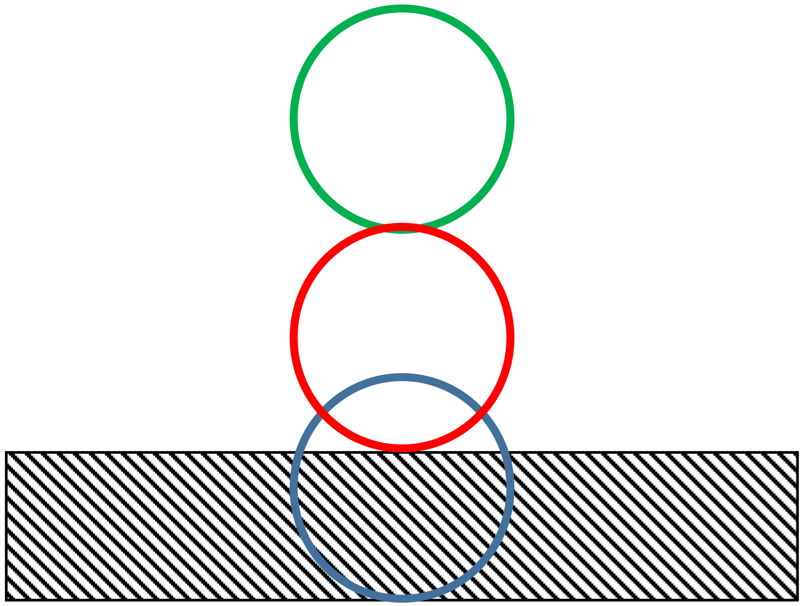


Constrained motion

Constraint solving

Naive approach #1:

```
while true
    applyExternalForces();
    integrate();
    if constraintsViolated()
        solveConstraints();
    end
    render();
    sleep();
end
```



Constrained motion

Constraint solving

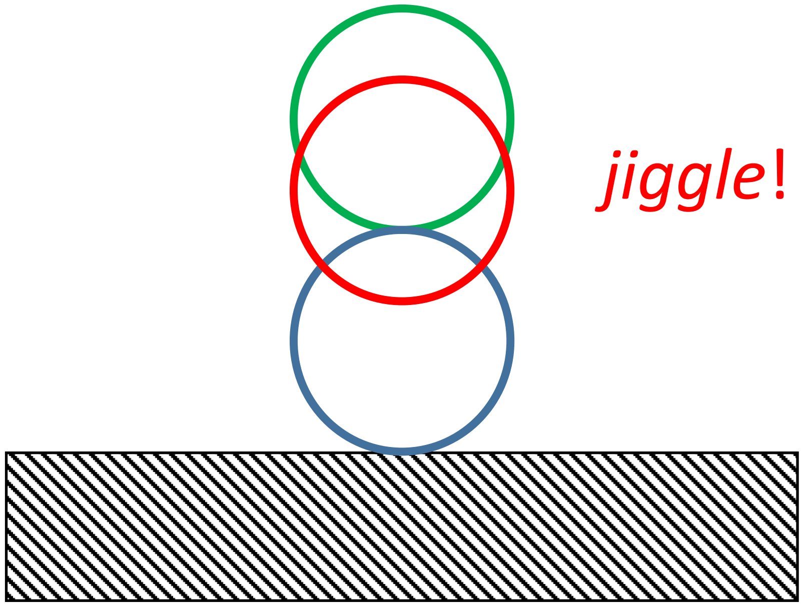
Naive approach #1:

+ easy to implement

+ sufficient for occasional isolated contacts

- each constraint is solved separately so it can't support complex system of constraints

- constraint solving is performed at a position level – inaccurate

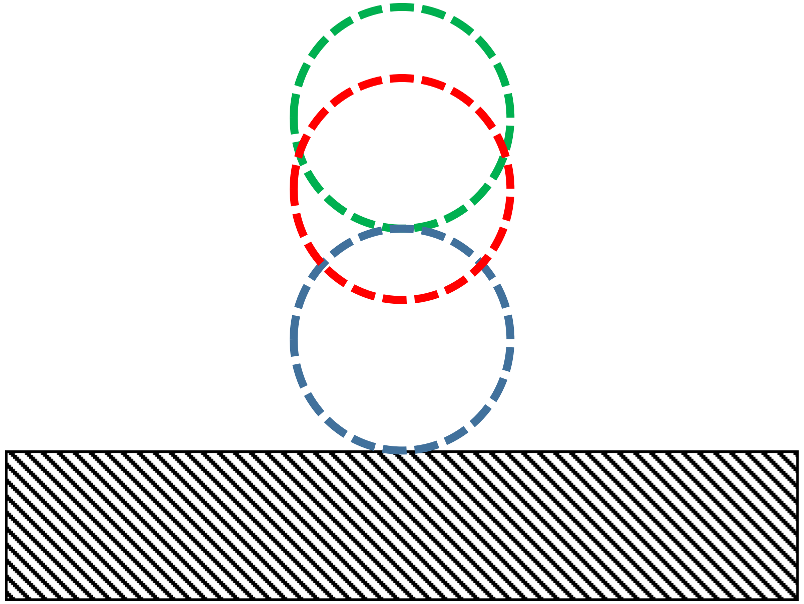


Constrained motion

Constraint solving

Naive approach #2:

```
while true
    applyExternalForces();
    integrate();
    while constraintsViolated()
        solveConstraints();
    end
    render();
    sleep();
end
```

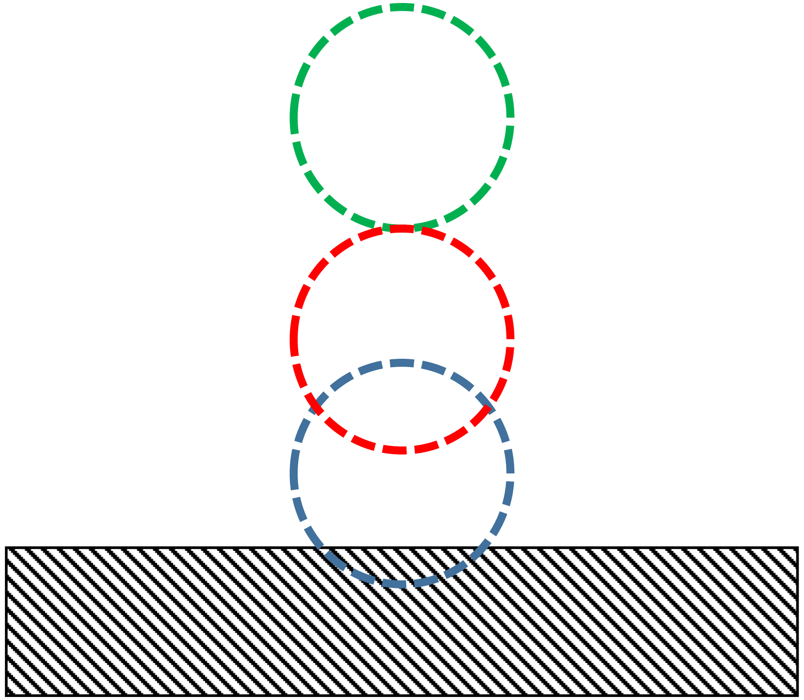


Constrained motion

Constraint solving

Naive approach #2:

```
while true
    applyExternalForces();
    integrate();
    while constraintsViolated()
        solveConstraints();
    end
    render();
    sleep();
end
```

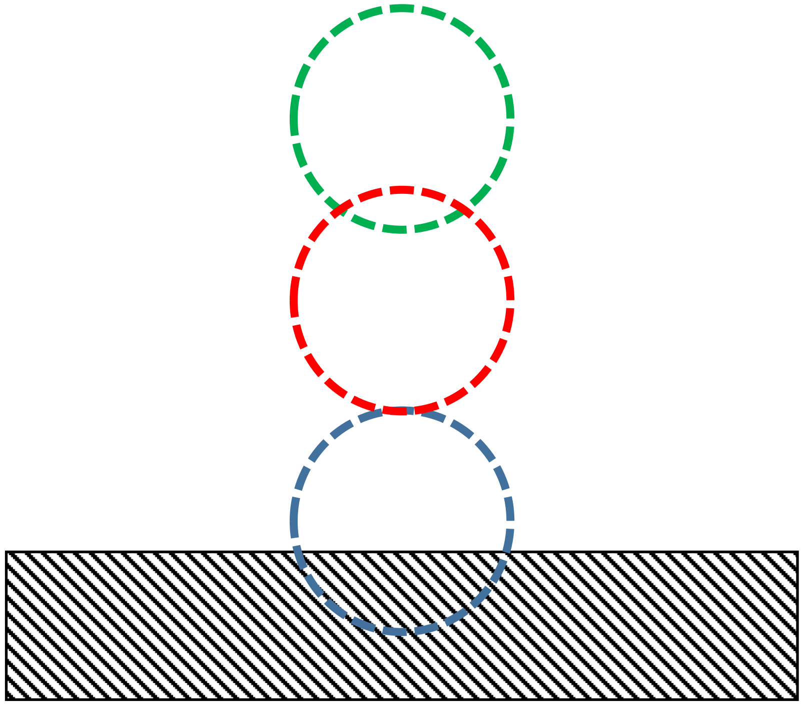


Constrained motion

Constraint solving

Naive approach #2:

```
while true
    applyExternalForces();
    integrate();
    while constraintsViolated()
        solveConstraints();
    end
    render();
    sleep();
end
```

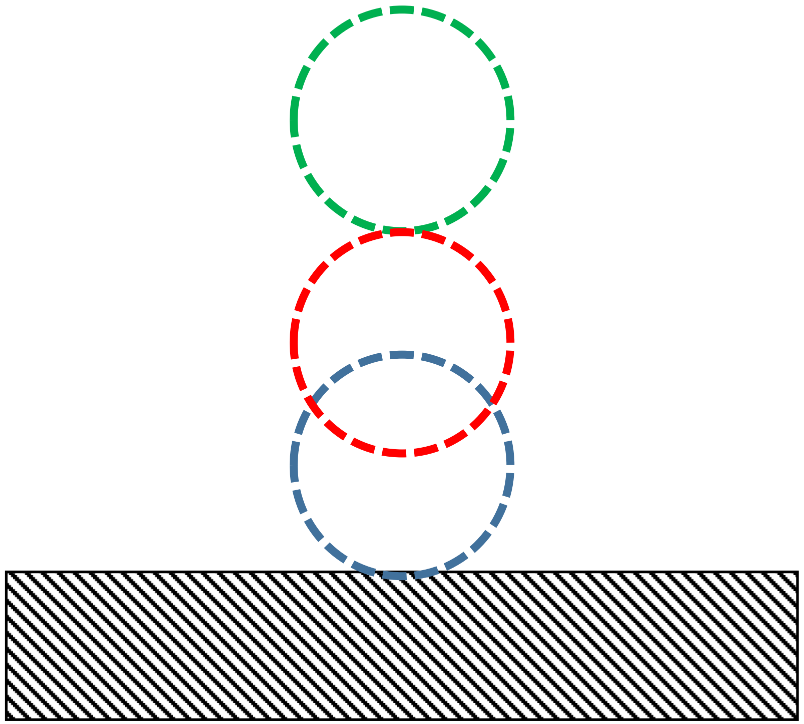


Constrained motion

Constraint solving

Naive approach #2:

```
while true
    applyExternalForces();
    integrate();
    while constraintsViolated()
        solveConstraints();
    end
    render();
    sleep();
end
```

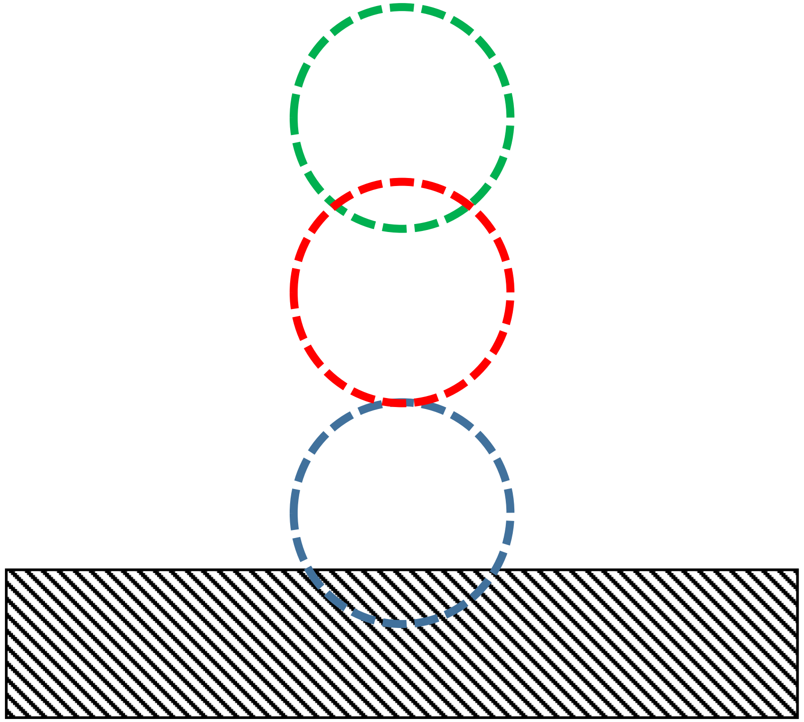


Constrained motion

Constraint solving

Naive approach #2:

```
while true
    applyExternalForces();
    integrate();
    while constraintsViolated()
        solveConstraints();
    end
    render();
    sleep();
end
```

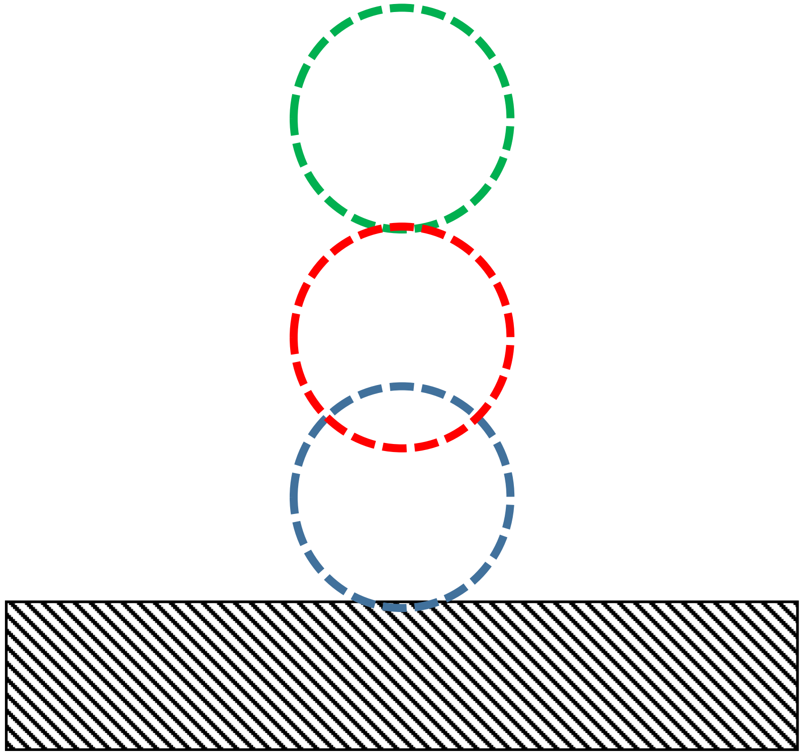


Constrained motion

Constraint solving

Naive approach #2:

```
while true
    applyExternalForces();
    integrate();
    while constraintsViolated()
        solveConstraints();
    end
    render();
    sleep();
end
```

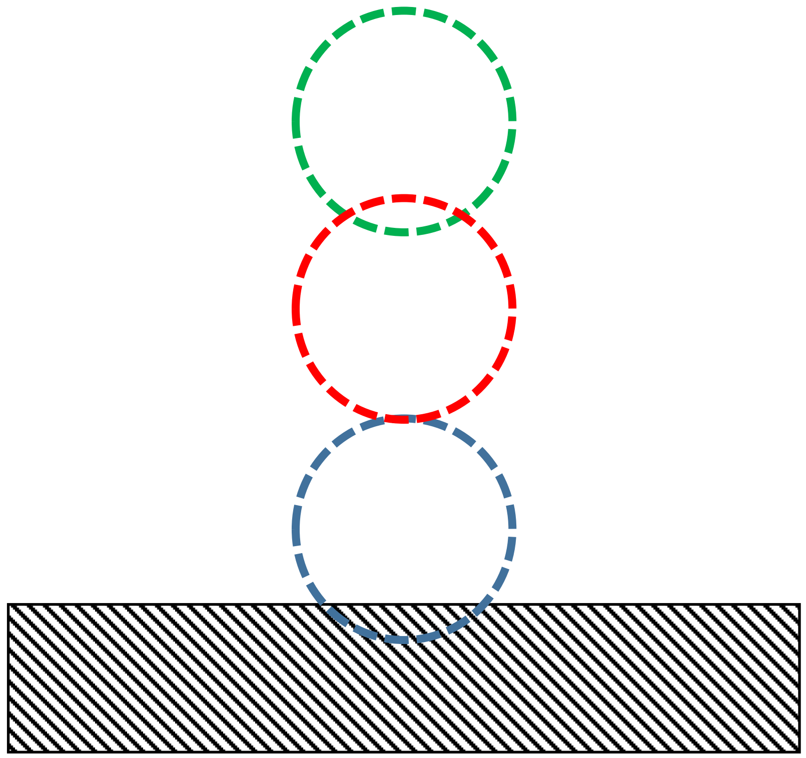


Constrained motion

Constraint solving

Naive approach #2:

```
while true
    applyExternalForces();
    integrate();
    while constraintsViolated()
        solveConstraints();
    end
    render();
    sleep();
end
```

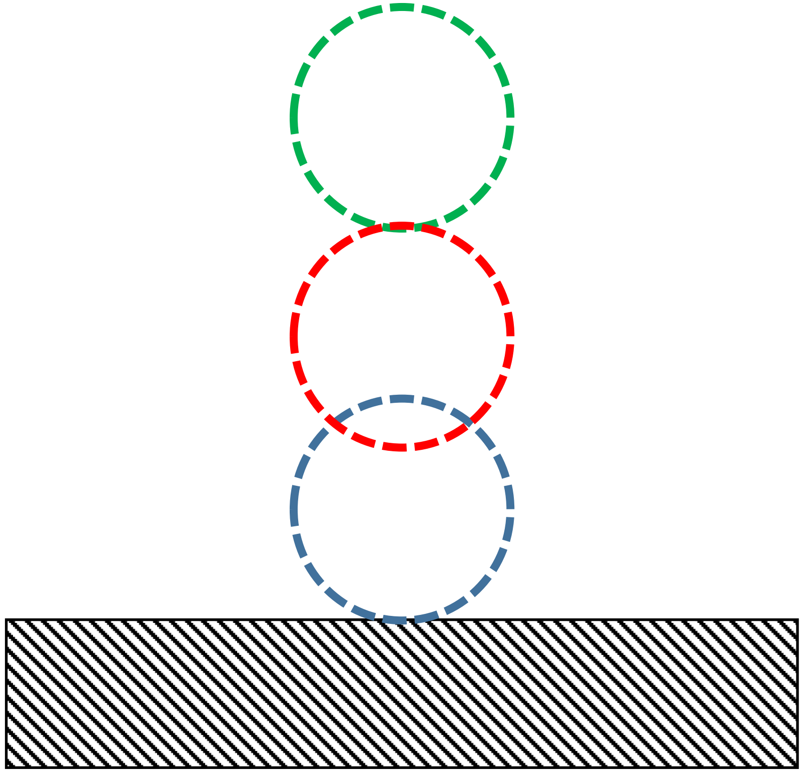


Constrained motion

Constraint solving

Naive approach #2:

```
while true
    applyExternalForces();
    integrate();
    while constraintsViolated()
        solveConstraints();
    end
    render();
    sleep();
end
```

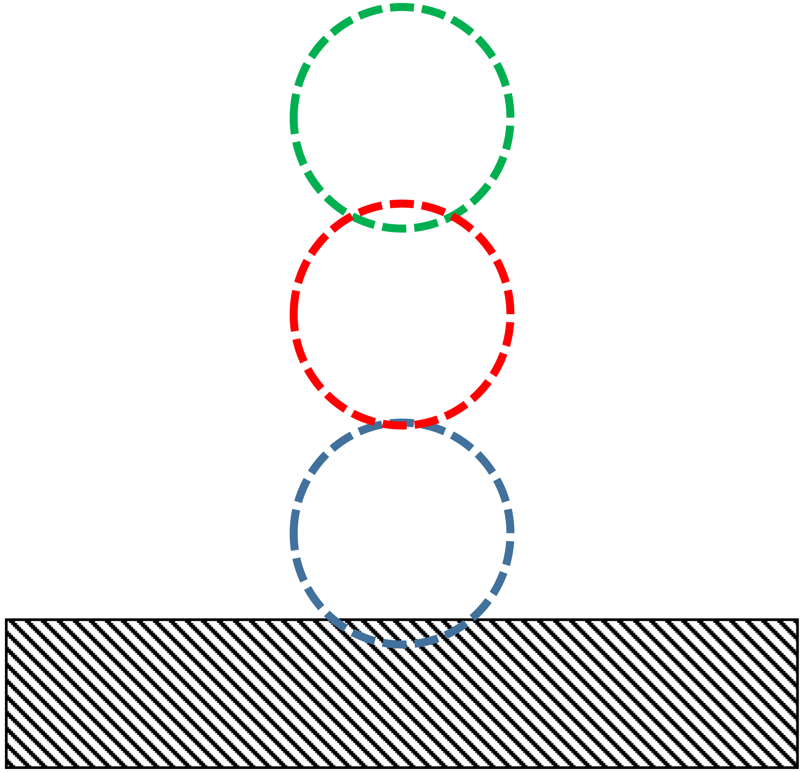


Constrained motion

Constraint solving

Naive approach #2:

```
while true
    applyExternalForces();
    integrate();
    while constraintsViolated()
        solveConstraints();
    end
    render();
    sleep();
end
```

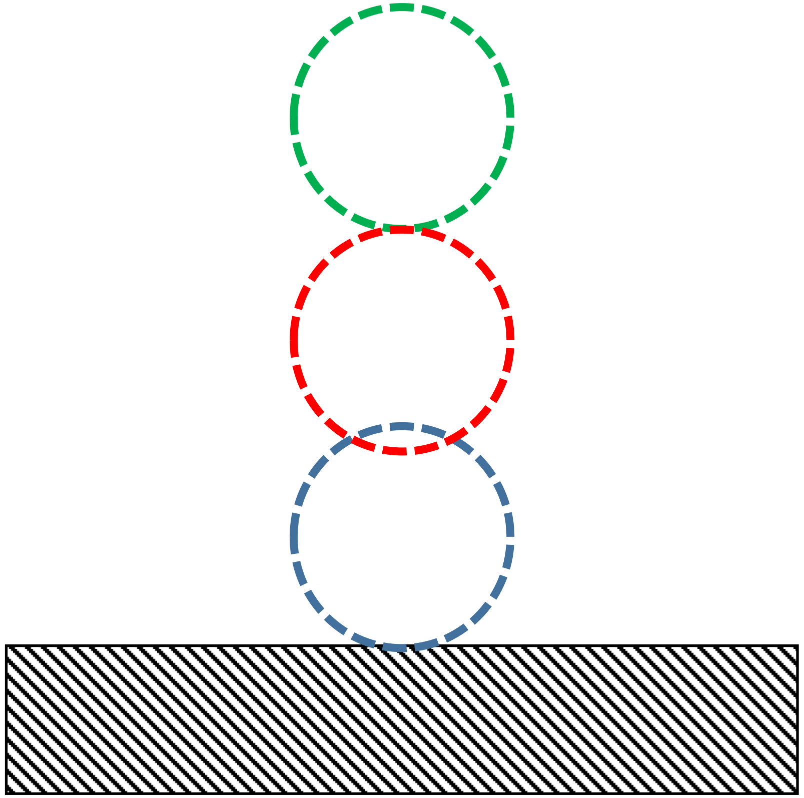


Constrained motion

Constraint solving

Naive approach #2:

```
while true
    applyExternalForces();
    integrate();
    while constraintsViolated()
        solveConstraints();
    end
    render();
    sleep();
end
```

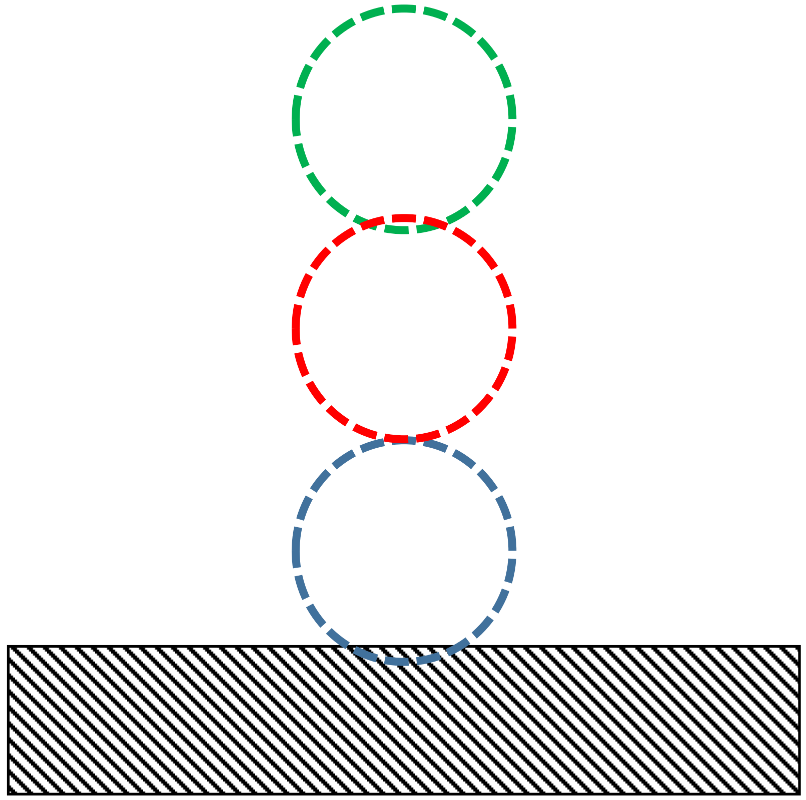


Constrained motion

Constraint solving

Naive approach #2:

```
while true
    applyExternalForces();
    integrate();
    while constraintsViolated()
        solveConstraints();
    end
    render();
    sleep();
end
```



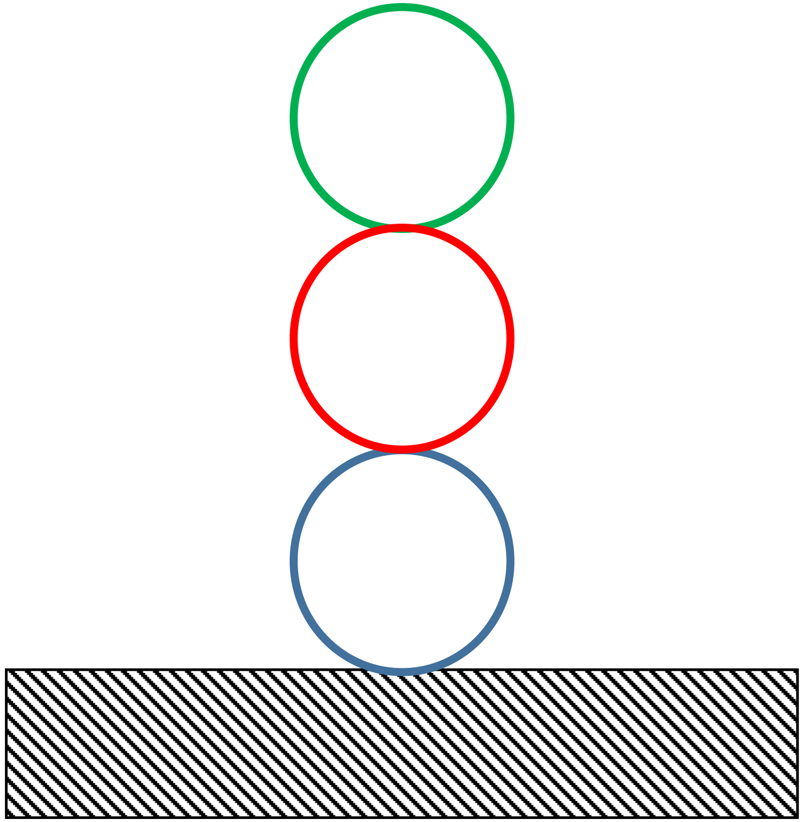
Constrained motion

Constraint solving

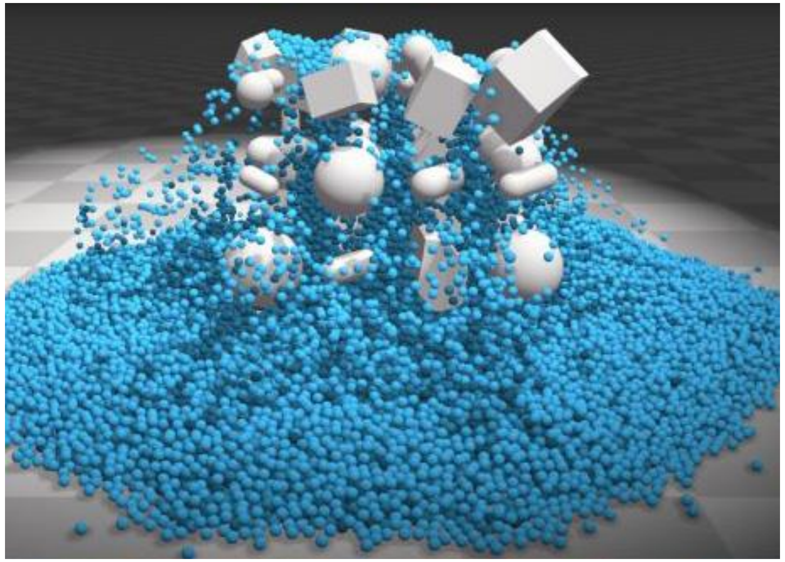
Naive approach #2:

+ easy to implement
+ solves more complex systems of constraints

- too slow (even for a small number of constraints)
- constraint solving is performed at a position level – inaccurate



Constrained motion



Constraint solving

Idea:

1. Define abstract model for all constraint types (so that all constraints are described in the same form)
2. Define system of linear equations to describe all the constraints together
3. Solve the system of linear equations (solve all the constraints together)
4. Use the solution to modify <u>forces and torques</u> which act on the bodies before the integration phase
5. During integration phase positions and rotations of bodies will be updated so that the constraints are satisfied afterwards

Constrained motion

- Engine pipeline:

Apply external forces

Detect constraint violation

Solve constraints

Integrate

Render

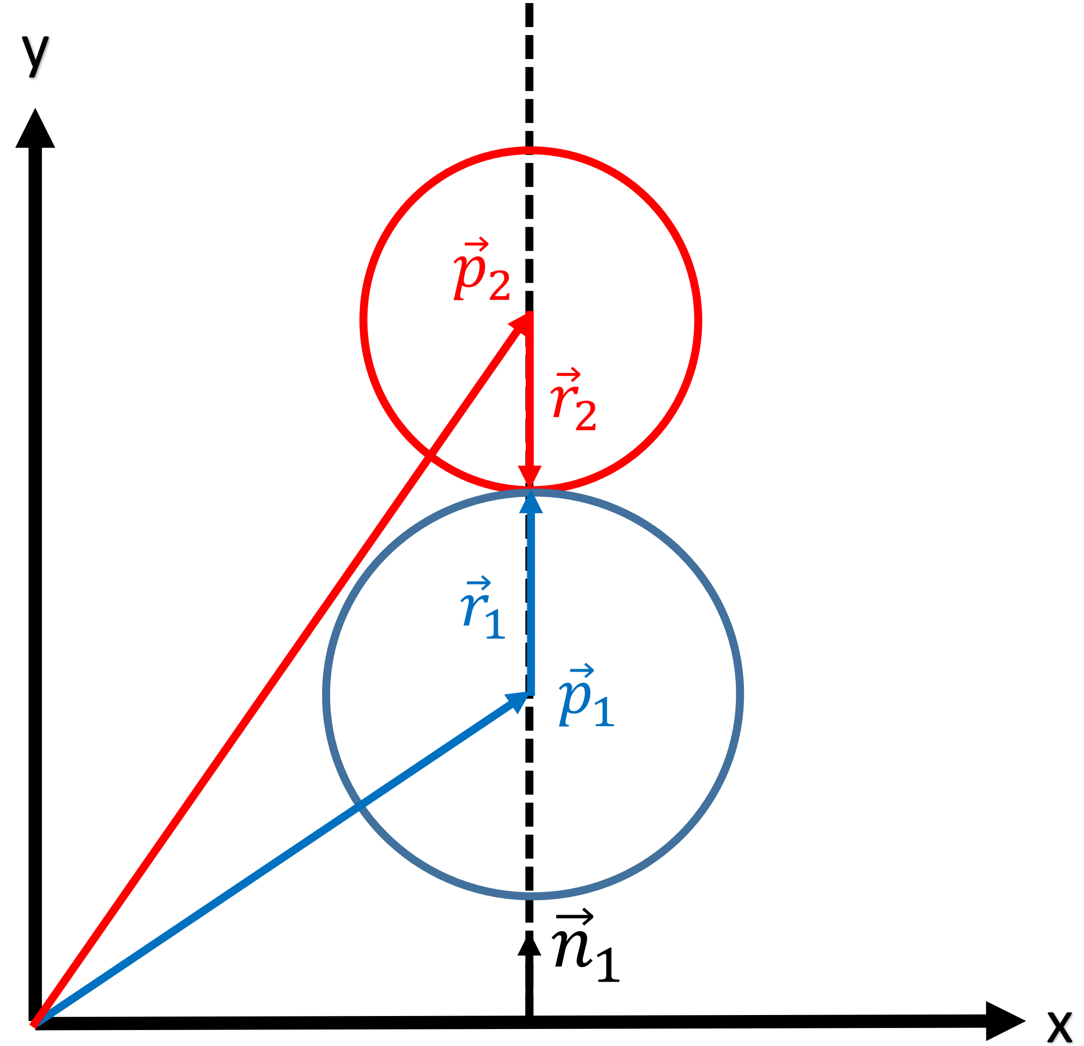
Sleep

Constrained motion

2. Constraint modeling (simple example)

Contact constraint for 2 particles (*pairwise*):

$$c(\vec{p}_1, \vec{p}_2) = 0$$
$$\left(\vec{p}_2 + \vec{r}_2 - (\vec{p}_1 + \vec{r}_1)\right) \cdot \vec{n}_1 = 0$$



Constrained motion

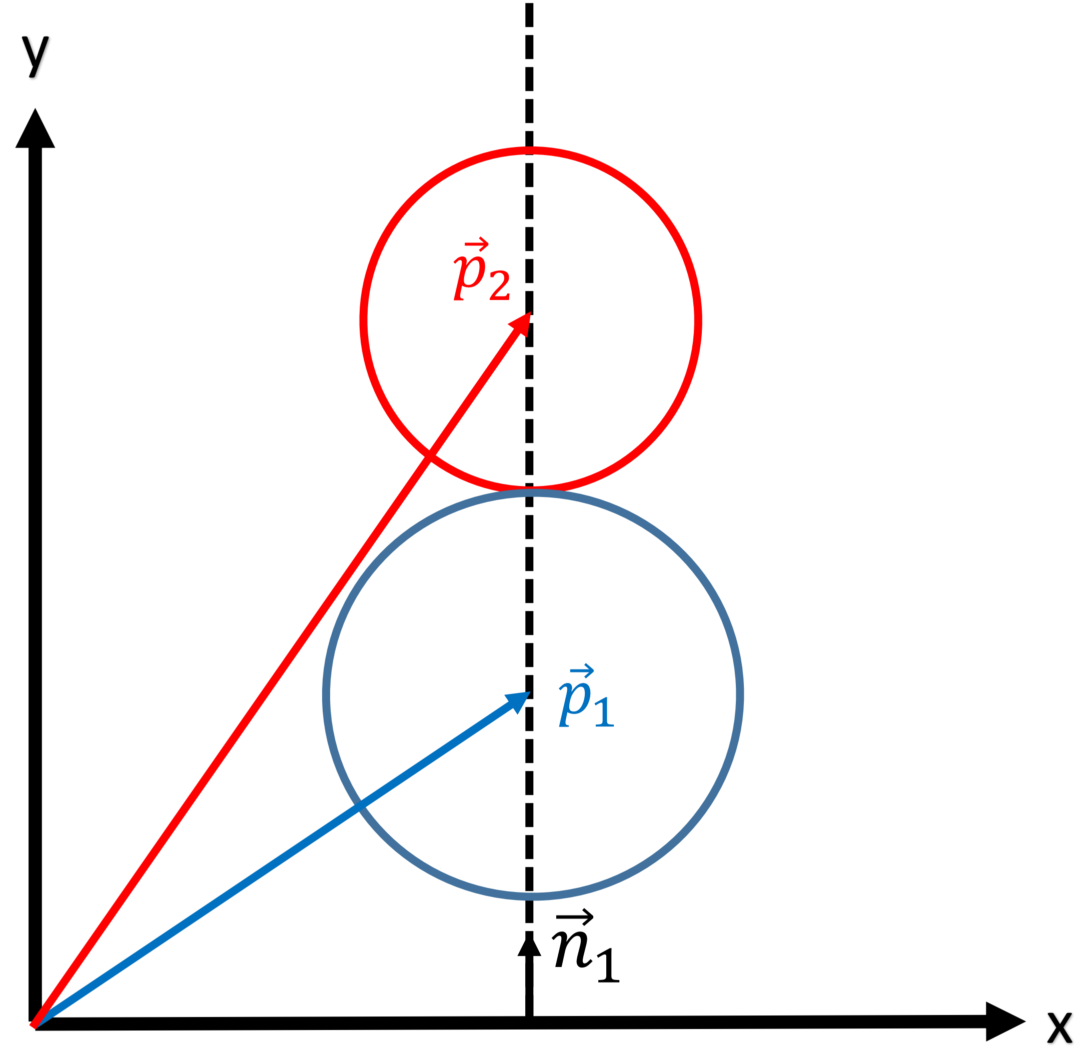
2. Constraint modeling (simple example)

In order for constraint to remain satisfied:

$$c(\vec{p}_1, \vec{p}_2) = 0$$

constraint derivative must be 0:

$$\frac{dc}{dt} = 0$$

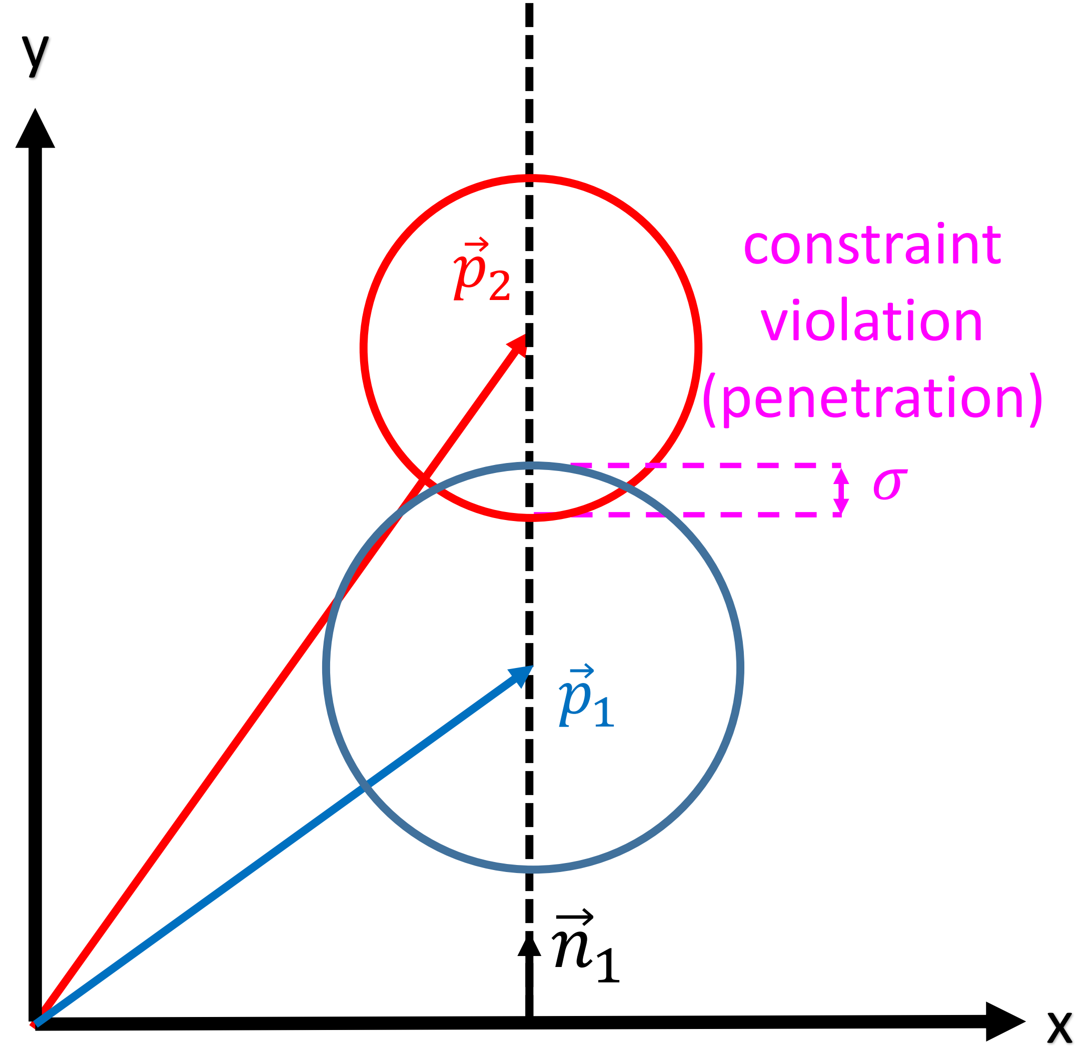


Constrained motion

2. Constraint modeling (simple example)

However, after the detection phase constraints are already violated:

$$c(\vec{p}_1, \vec{p}_2) < 0$$



Constrained motion

2. Constraint modeling (simple example)

wrong assumption:

constraints are satisfied

$$c = 0$$

Solve:

$$\frac{dc}{dt} = 0$$

resulting velocities of bodies will be such that their positions remain unchanged

incorrect positions

$$\xrightarrow{V} \boxed{\int dt} \xrightarrow{P}$$

Constrained motion

2. Constraint modeling (simple example)

Baumgarte stabilization (J. Baumgarte):

correct assumption:

constraints are violated

$$c < 0$$

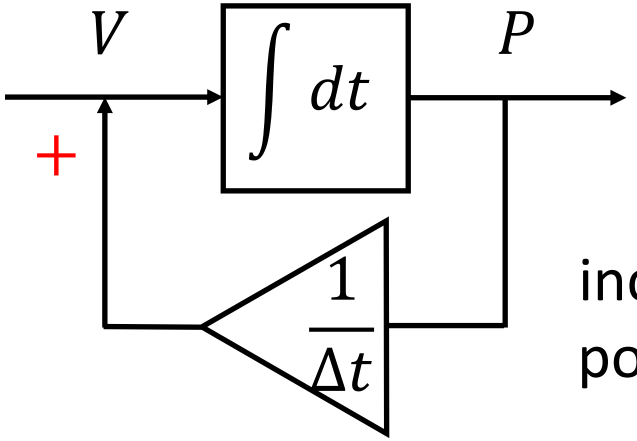
Solve:

$$\frac{dc}{dt} + \frac{c}{\Delta t} = 0$$

positive feedback loop

resulting velocities of bodies will be such that their positions are modified

modified positions



incorrect positions

Constrained motion

2. Constraint modeling (simple example)

Baumgarte stabilization:

correct assumption:

constraints are violated

$$c < 0$$

Solve:

$$\frac{dc}{dt} + \beta \frac{c}{\Delta t} = 0$$

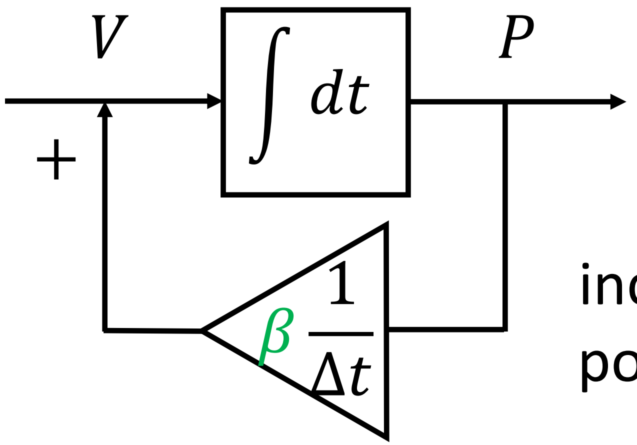
limit introducing the additional energy

$$0 \leq \beta \leq 1$$

predetermined constant

numerical integration inherently causes errors, thus the positions will not be accurately updated from velocities (*numerical drift*), which introduces additional energy into the system

modified positions



incorrect positions

Constrained motion

2. Constraint modeling (simple example)

$$c(\vec{p}_1, \vec{p}_2) = \left(\vec{p}_2 + \vec{r}_2 - (\vec{p}_1 + \vec{r}_1)\right) \cdot \vec{n}_1$$

Solve:

$$\frac{dc}{dt} + \beta \frac{c}{\Delta t} = 0$$

$$\frac{dc}{dt} = -\beta \frac{c}{\Delta t}$$

$$\frac{d}{dt}\left(\left(\vec{p}_2 + \vec{r}_2 - (\vec{p}_1 + \vec{r}_1)\right) \cdot \vec{n}_1\right) = -\beta \frac{c}{\Delta t}$$

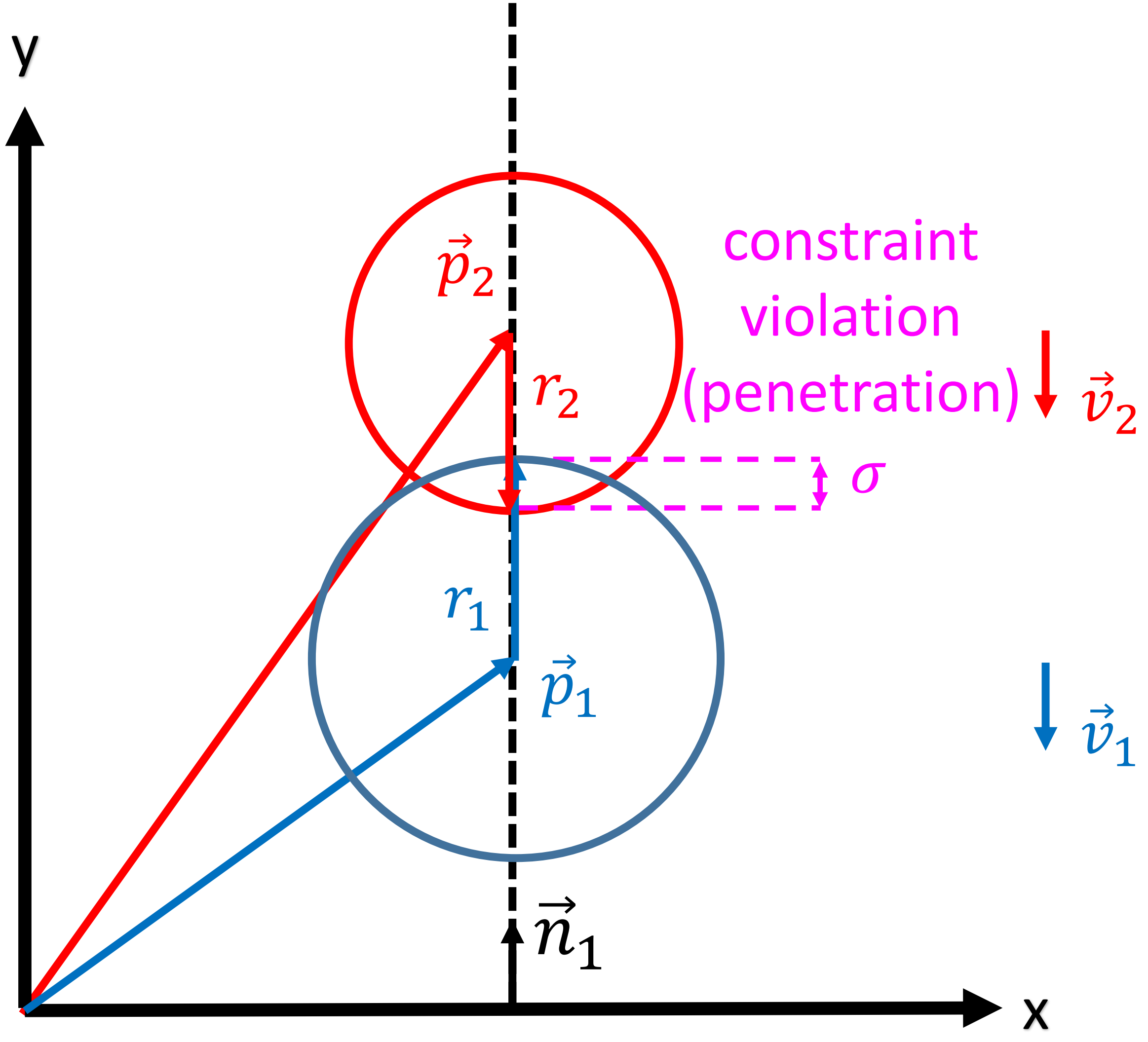
normal is constant

$$\frac{d}{dt}\left(\vec{p}_2 + \vec{r}_2 - (\vec{p}_1 + \vec{r}_1)\right) \cdot \vec{n}_1 + \cancel{\left(\vec{p}_2 + \vec{r}_2 - (\vec{p}_1 + \vec{r}_1)\right) \cdot \frac{d}{dt}\vec{n}_1} = -\beta \frac{c}{\Delta t}$$

$$\left(\frac{d}{dt}\vec{p}_2 + \cancel{\frac{d}{dt}\vec{r}_2} - \frac{d}{dt}\vec{p}_1 - \cancel{\frac{d}{dt}\vec{r}_1}\right) \cdot \vec{n}_1 = -\beta \frac{c}{\Delta t}$$

$$-\vec{v}_1 \cdot \vec{n}_1 + \vec{v}_2 \cdot \vec{n}_1 = -\beta \frac{c}{\Delta t}$$

radii are constant



Constrained motion

2. Constraint modeling (simple example)

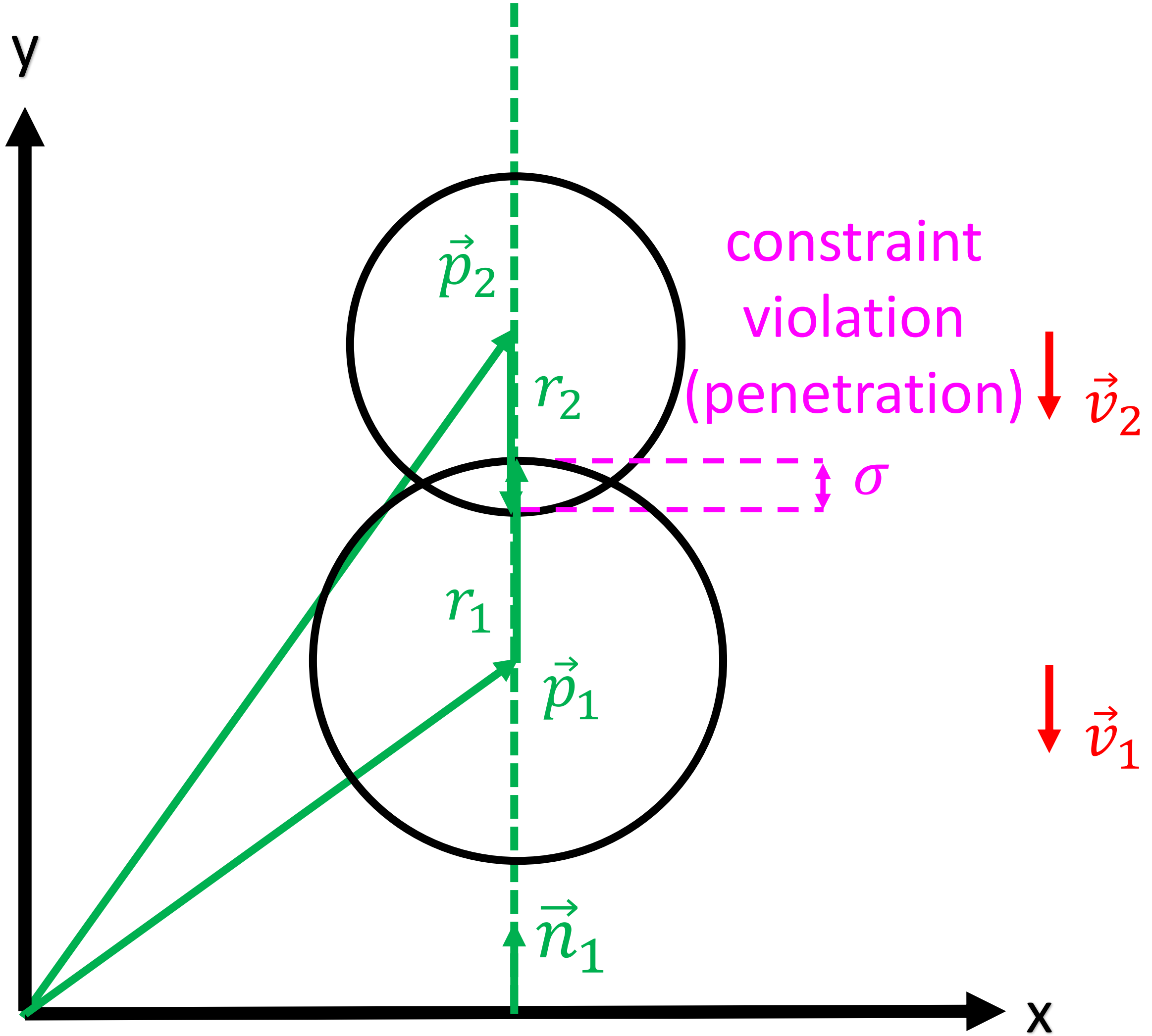
$$c(\vec{p}_1, \vec{p}_2) = \left(\vec{p}_2 + \vec{r}_2 - (\vec{p}_1 + \vec{r}_1)\right) \cdot \vec{n}_1$$

Solve:

?

$$-\vec{v}_1 \cdot \vec{n}_1 + \vec{v}_2 \cdot \vec{n}_1 = -\beta \frac{c}{\Delta t}$$

unknown known



Constrained motion

2. Constraint modeling (simple example)

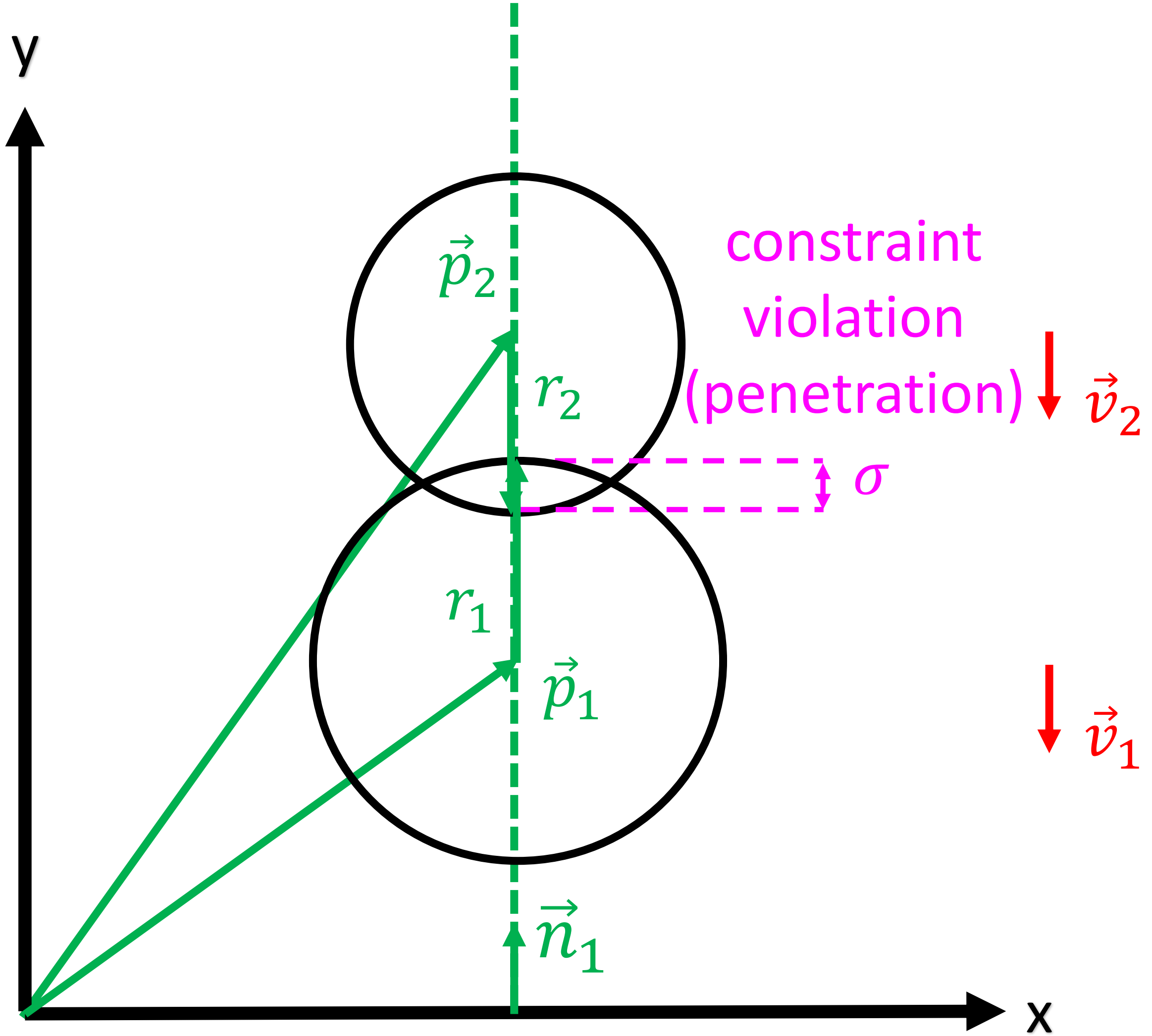
$$\sigma = (\vec{p}_2 + \vec{r}_2 - (\vec{p}_1 + \vec{r}_1)) \cdot \vec{n}_1$$

Solve:

found in detection phase

$$-\vec{v}_1 \cdot \vec{n}_1 + \vec{v}_2 \cdot \vec{n}_1 = -\beta \frac{\sigma}{\Delta t}$$

unknown known



Constrained motion

2. Constraint modeling (simple example)

$$c(\vec{p}_1, \vec{p}_2) = \left(\vec{p}_2 + \vec{r}_2 - (\vec{p}_1 + \vec{r}_1)\right) \cdot \vec{n}_1$$

vector form:

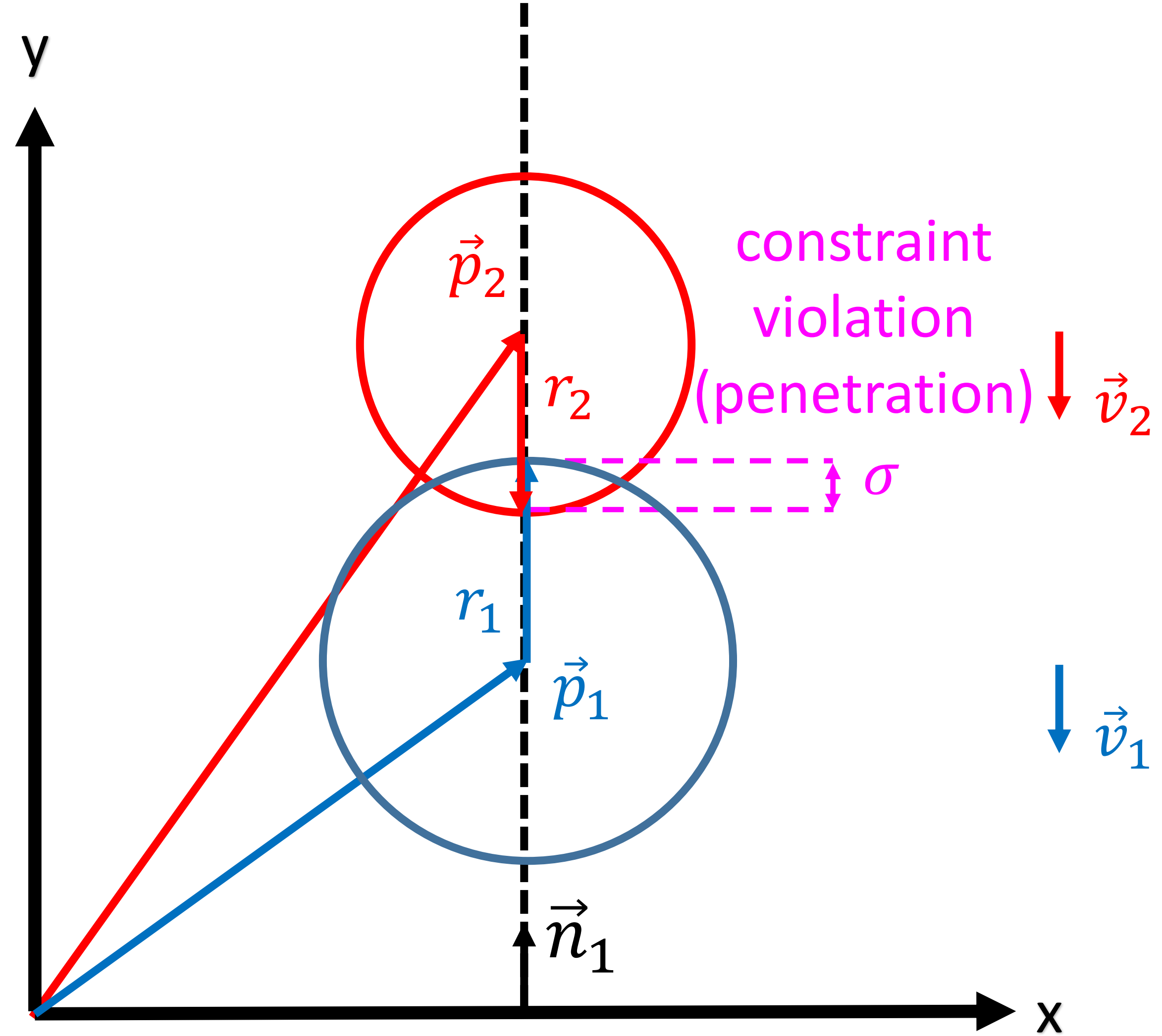
$$-\vec{v}_1 \cdot \vec{n}_1 + \vec{v}_2 \cdot \vec{n}_1 = -\beta \frac{\sigma}{\Delta t}$$

variable form depending on the constraints

$$\begin{bmatrix} -\vec{n}_1 & \vec{n}_1 \end{bmatrix} \begin{bmatrix} \vec{v}_1 \\ \vec{v}_2 \end{bmatrix} = -\beta \frac{\sigma}{\Delta t}$$

same form always

$$jv = -\beta \frac{\sigma}{\Delta t}$$



Constrained motion

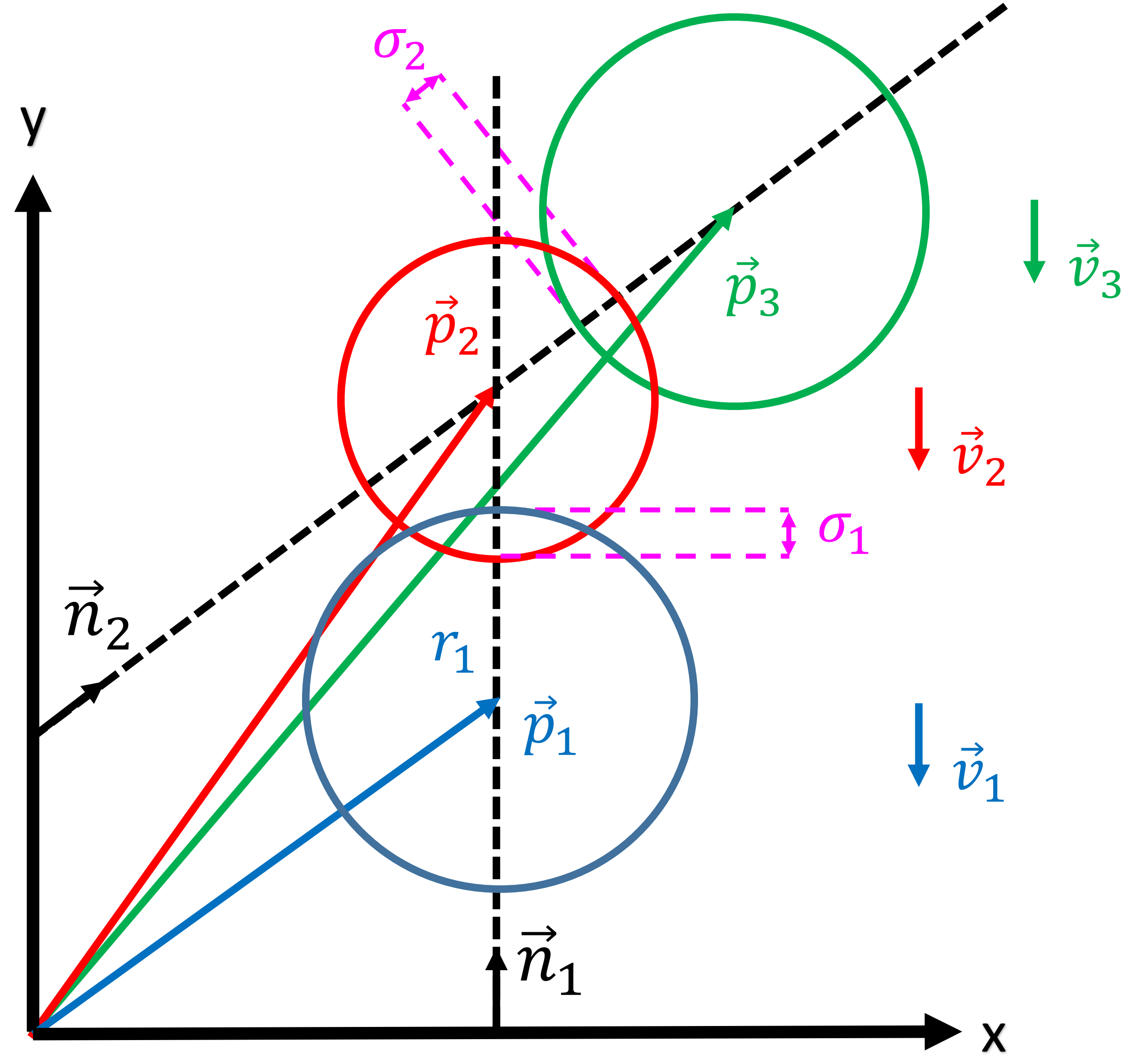
2. Constraint modeling (simple example)

For multiple contacts:

$$\begin{bmatrix} -\vec{n}_1 & \vec{n}_1 \end{bmatrix} \begin{bmatrix} \vec{v}_1 \\ \vec{v}_2 \end{bmatrix} = -\beta \frac{\sigma_1}{\Delta t}$$

$$\begin{bmatrix} -\vec{n}_2 & \vec{n}_2 \end{bmatrix} \begin{bmatrix} \vec{v}_2 \\ \vec{v}_3 \end{bmatrix} = -\beta \frac{\sigma_2}{\Delta t}$$

$$\vdots$$



Constrained motion

2. Constraint modeling (simple example)

System of first order constraint derivatives:

$$\begin{bmatrix} -\vec{n}_1 & \vec{n}_1 \end{bmatrix} \begin{bmatrix} \vec{v}_1 \\ \vec{v}_2 \end{bmatrix} = -\beta \frac{\sigma_1}{\Delta t}$$

$$\begin{bmatrix} -\vec{n}_2 & \vec{n}_2 \end{bmatrix} \begin{bmatrix} \vec{v}_2 \\ \vec{v}_3 \end{bmatrix} = -\beta \frac{\sigma_2}{\Delta t}$$

$$\vdots$$

$$\begin{bmatrix} -\vec{n}_n & \vec{n}_n \end{bmatrix} \begin{bmatrix} \vec{v}_n \\ \vec{v}_{n+1} \end{bmatrix} = -\beta \frac{\sigma_n}{\Delta t}$$

Constrained motion

2. Constraint modeling (simple example)

Expanded to include all the space dimensions:

$$\begin{bmatrix} -n_{1x} & -n_{1y} & n_{1x} & n_{1y} \end{bmatrix} \begin{bmatrix} v_{1x} \\ v_{1y} \\ v_{2x} \\ v_{2y} \end{bmatrix} = -\beta \frac{\sigma_1}{\Delta t}$$

$$\begin{bmatrix} -n_{2x} & -n_{2y} & n_{2x} & n_{2y} \end{bmatrix} \begin{bmatrix} v_{2x} \\ v_{2y} \\ v_{3x} \\ v_{3y} \end{bmatrix} = -\beta \frac{\sigma_2}{\Delta t}$$

$$\vdots$$

$$\begin{bmatrix} -n_{nx} & -n_{ny} & n_{nx} & n_{ny} \end{bmatrix} \begin{bmatrix} v_{nx} \\ v_{ny} \\ v_{(n+1)x} \\ v_{(n+1)y} \end{bmatrix} = -\beta \frac{\sigma_n}{\Delta t}$$

Constrained motion

2. Constraint modeling (simple example)

Jacobian — velocities — constraint violation (penetration)

$$
\begin{array}{c}
\begin{matrix} v_{1x} & v_{1y} & v_{2x} & v_{2y} & \cdots \end{matrix} \\
\begin{bmatrix}
-n_{1x} & -n_{1y} & n_{1x} & n_{1y} & 0 & 0 & \cdots & & & & & & \\
0 & 0 & -n_{2x} & -n_{2y} & n_{2x} & n_{2y} & & & & & & & \\
 & & \vdots & & & & \ddots & & & \vdots & & & \\
 & & & & & & \cdots & -n_{(n-1)x} & -n_{(n-1)y} & n_{(n-1)x} & n_{(n-1)y} & 0 & 0 \\
 & & & & & & & 0 & 0 & -n_{nx} & -n_{ny} & n_{nx} & n_{ny}
\end{bmatrix}
\end{array}
\begin{bmatrix} v_{1x} \\ v_{1y} \\ v_{2x} \\ v_{2y} \\ \vdots \\ v_{nx} \\ v_{ny} \end{bmatrix}
= -\beta \frac{1}{\Delta t} \begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \vdots \\ \sigma_{n-1} \\ \sigma_n \end{bmatrix}
$$

$$
JV = -\beta \frac{1}{\Delta t} S
$$

In order for constraint to <u>be solved</u> velocity along the constraint direction must cancel out constraint violation.

Constrained motion

3. Constraint solving:

known constraint violation directions

known constraint violation measures

$$\mathrm{J}V = -\beta \frac{1}{\Delta t} S$$

required unknown velocity in the next time step

Final solution is the constraint force F_c
which will cancel out the constraint violation!

Constrained motion

$$\frac{d^2\vec{p}(t)}{dt^2} = \frac{\vec{F}(t)}{m}$$

$$\frac{d\vec{v}(t)}{dt} = \frac{\vec{F}(t)}{m}$$

3. Constraint solving:

known constraint violation directions

known constraint violation measures

$$JV = -\beta \frac{1}{\Delta t} S$$

first order position derivative

matrix form:

$$\frac{dV}{dt} = \frac{1}{M} F \leftarrow \text{total force}$$

second order position derivative

2nd Newton’s Law of Motion!

Constrained motion

$$\frac{d^2\vec{p}(t)}{dt^2} = \frac{\vec{F}(t)}{m}$$
$$\frac{d\vec{v}(t)}{dt} = \frac{\vec{F}(t)}{m}$$

3. Constraint solving:

known constraint violation directions

known constraint violation measures

$$JV = -\beta \frac{1}{\Delta t} S$$

first order position derivative

Doesn't fit!

matrix form:

$$\frac{dV}{dt} = \frac{1}{M} F$$
$$\frac{dV}{dt} = \frac{1}{M} (F_{ext} + F_c)$$

required constraint forces

second order position derivative

known masses

known external forces

Constrained motion

$$\frac{d^2\vec{p}(t)}{dt^2} = \frac{\vec{F}(t)}{m}$$
$$\frac{d\vec{v}(t)}{dt} = \frac{\vec{F}(t)}{m}$$

3. Constraint solving:

known constraint violation directions

known constraint violation measures

$$JV = -\beta \frac{1}{\Delta t} S$$

first order position derivative

Solution #1: differentiate both sides once more
Result: complicated!

matrix form:

$$\frac{dV}{dt} = \frac{1}{M} F$$
$$\frac{dV}{dt} = \frac{1}{M} (F_{ext} + F_c)$$

required constraint forces

second order position derivative

known masses

known external forces

Constrained motion

$$\frac{d^2\vec{p}(t)}{dt^2} = \frac{\vec{F}(t)}{m}$$
$$\frac{d\vec{v}(t)}{dt} = \frac{\vec{F}(t)}{m}$$

3. Constraint solving:

known constraint violation directions

known constraint violation measures

$$JV(t+\Delta t) = -\beta\frac{1}{\Delta t}S$$

velocity in next time step

matrix form:

$$\frac{dV}{dt} = \frac{1}{M}F$$
$$\frac{V(t+\Delta t) - V}{\Delta t} = \frac{1}{M}(F_{ext} + F_c)$$

required constraint forces

known velocity in current time step

known masses

known external forces

Solution #2:
Finite difference approximation!

Constrained motion

$$\frac{d^2\vec{p}(t)}{dt^2} = \frac{\vec{F}(t)}{m}$$

$$\frac{d\vec{v}(t)}{dt} = \frac{\vec{F}(t)}{m}$$

3. Constraint solving:

matrix form:

$$\frac{dV}{dt} = \frac{1}{M}F$$

$$V(t+\Delta t) = -\beta\frac{1}{dt}J^T S$$

$$V(t+\Delta t) - V = \Delta t\frac{1}{M}(F_{ext} + F_c)$$

Constrained motion

3. Constraint solving:

$$-\beta \frac{1}{\Delta t} J^T S - V = \Delta t \frac{1}{M} (F_{ext} + F_c)$$

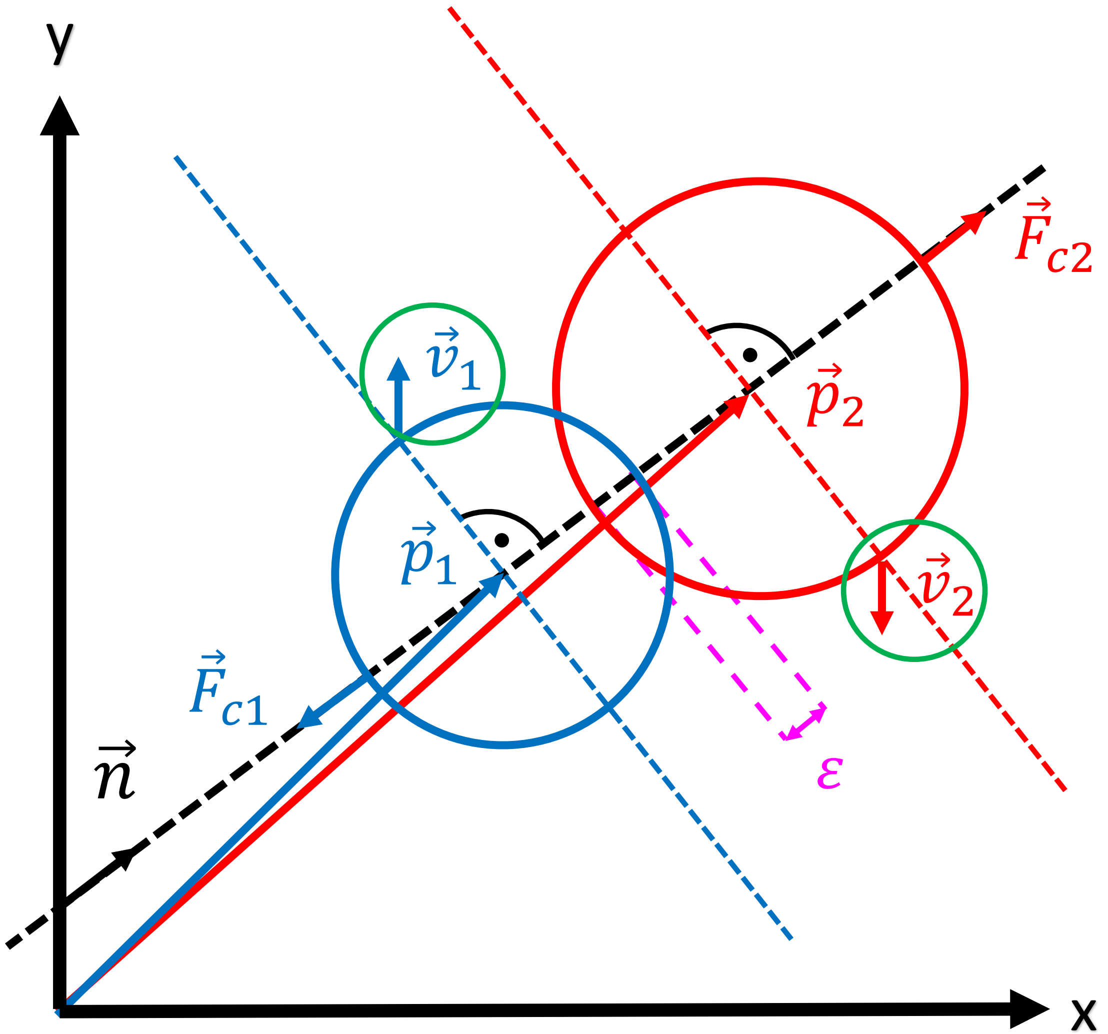
Constrained motion

3. Constraint solving:

Constraint forces:

$$F_c \perp V(t + \Delta t)$$

- they cancel out fragments of current velocities which would further violate constraints
- they do not act along the directions of velocities in next time step (admissible velocities), but perpendicular to them
- they do not introduce additional energy into the system (they perform virtual work)



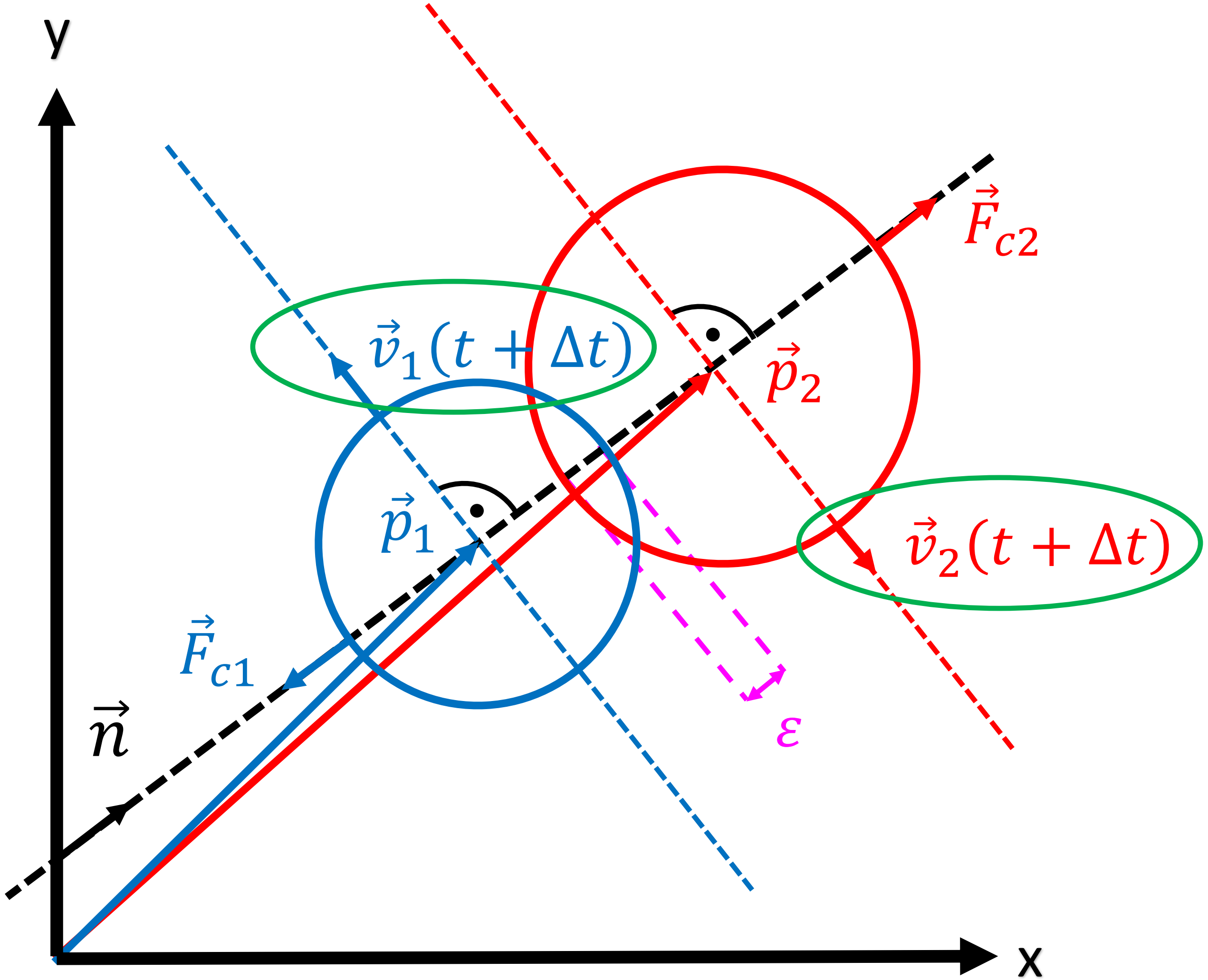
Constrained motion

3. Constraint solving:

Constraint forces:

$$F_c \perp V(t + \Delta t)$$

- they cancel out fragments of current velocities which would further violate constraints
- they do not act along the directions of velocities in next time step (admissible velocities), but perpendicular to them
- they do not introduce additional energy into the system (they perform virtual work)



Constrained motion

3. Constraint solving:

Jacobian rows are perpendicular to corresponding admissible velocities

$$\frac{dc}{dt} = 0$$

$$JV(t + \Delta t) = 0$$

Considering:

$$F_c \perp V(t + \Delta t)$$

constraint forces are perpendicular to corresponding admissible velocities

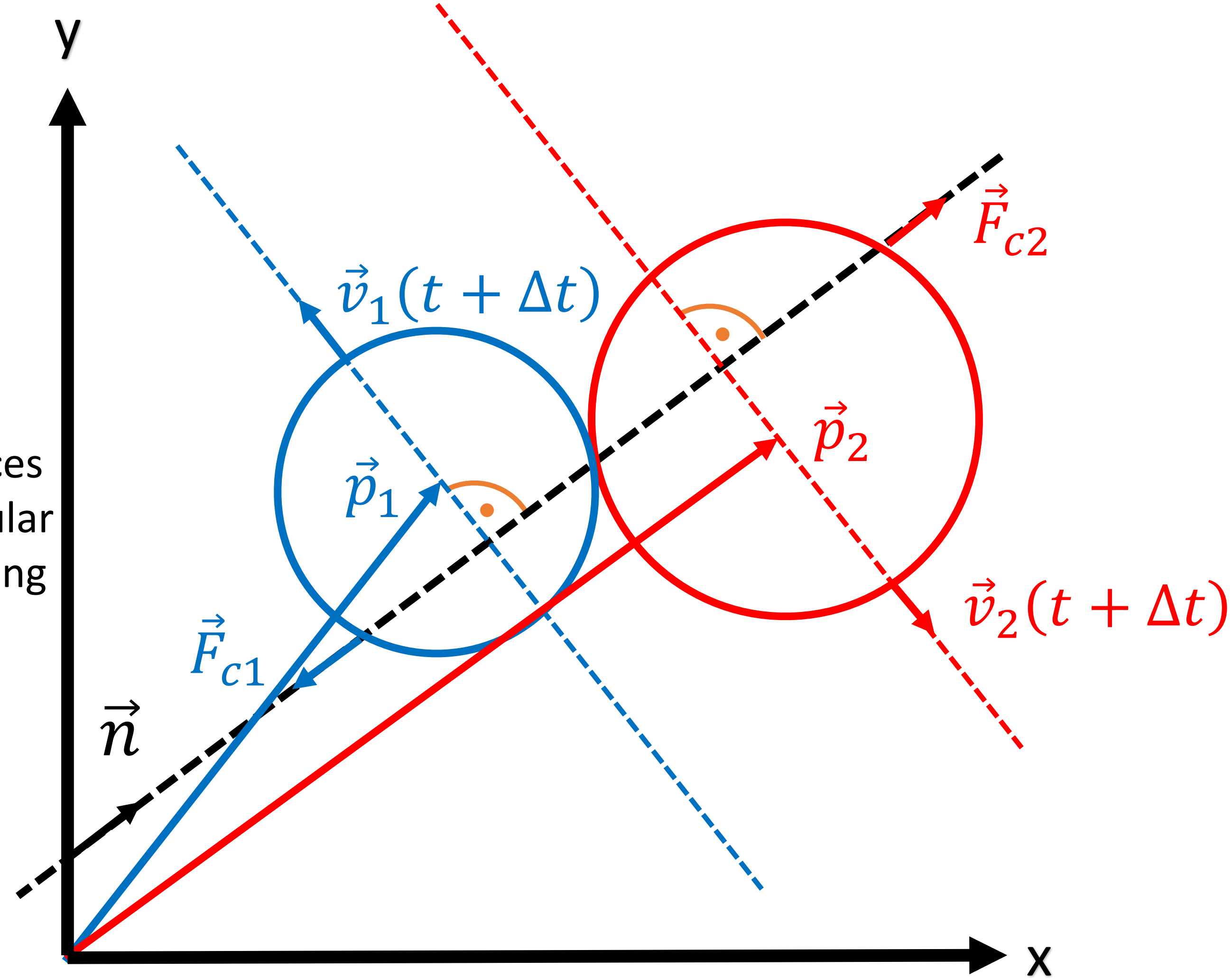
the following is then true:

$$F_c = J^T \lambda$$

constraint forces must be parallel to corresponding rows of J^T (must have same directions)

known direciton

unknown magnitudes



Constrained motion

3. Constraint solving:

$$-\beta \frac{1}{\Delta t} J^T S - V = \Delta t \frac{1}{M} (F_{ext} + F_c)$$

Constrained motion

3. Constraint solving:

$$-\beta \frac{1}{\Delta t} J^T S - V = \Delta t \frac{1}{M} (F_{ext} + J^T \lambda)$$

Constrained motion

3. Constraint solving:

$$-\beta \frac{1}{\Delta t} J^T S - V = \Delta t \frac{1}{M} (F_{ext} + J^T \lambda)$$

$$-\beta \frac{1}{\Delta t} J^T S - V = \Delta t \frac{1}{M} F_{ext} + \Delta t \frac{1}{M} J^T \lambda$$

$$\Delta t \frac{1}{M} J^T \lambda = -\beta \frac{1}{\Delta t} J^T S - V - \Delta t \frac{1}{M} F_{ext}$$

$$J \frac{1}{M} J^T \lambda = -\beta \frac{1}{\Delta t^2} S - \frac{1}{\Delta t} JV - J \frac{1}{M} F_{ext}$$

Constrained motion

3. Constraint solving:

$$J\frac{1}{M}J^T\lambda = -\beta\frac{1}{\Delta t^2}S - \frac{1}{\Delta t}JV - J\frac{1}{M}F_{ext}$$

Constrained motion

3. Constraint solving:

$$J\frac{1}{M}J^T\lambda = -\beta\frac{1}{\Delta t^2}S - \frac{1}{\Delta t}JV - J\frac{1}{M}F_{ext}$$

What is known and what is unknown?

Constrained motion

3. Constraint solving:

$$J\frac{1}{M}J^T\lambda = -\beta\frac{1}{\Delta t^2}S - \frac{1}{\Delta t}JV - J\frac{1}{M}F_{ext}$$

effective inverse mass

constraints resolution accelerations

accelerations needed to prevent further violation of constraints as a consequence of existing momentums

accelerations needed to prevent further violation of constraints as a consequence of applied external forces

$\frac{1}{kg}$

$\frac{m}{s^2}$

$\frac{m}{s^2}$

$\frac{1}{kg}\frac{kg \cdot m}{s^2} = \frac{m}{s^2}$

Constrained motion

3. Constraint solving:

$$J\frac{1}{M}J^T\lambda = -\beta\frac{1}{\Delta t^2}S - \frac{1}{\Delta t}JV - J\frac{1}{M}F_{ext}$$

effective inverse mass	constraints resolution accelerations	accelerations needed to prevent further violation of constraints as a consequence of existing momentums	accelerations needed to prevent further violation of constraints as a consequence of applied external forces
$\frac{1}{kg}$	$\frac{m}{s^2}$	$\frac{m}{s^2}$	$\frac{1}{kg}\frac{kg\cdot m}{s^2} = \frac{m}{s^2}$
matrix $n \times n$!	vector n × 1!	vector n × 1!	vector n × 1!

Constrained motion

3. Constraint solving:

$$J\frac{1}{M}J^T\lambda = -\beta\frac{1}{\Delta t^2}S - \frac{1}{\Delta t}JV - J\frac{1}{M}F_{ext}$$

$$A\lambda = b$$

Constrained motion

3. Constraint solving:

$$J\frac{1}{M}J^T\lambda = -\beta\frac{1}{\Delta t^2}S - \frac{1}{\Delta t}JV - J\frac{1}{M}F_{ext}$$

$$A\lambda = b$$

$$\lambda = \mathrm{A}\backslash\mathrm{b}$$

numerical solution of
system of linear equations

Constrained motion

3. Constraint solving:

$$J\frac{1}{M}J^T\lambda = -\beta\frac{1}{\Delta t^2}S - \frac{1}{\Delta t}JV - J\frac{1}{M}F_{ext}$$

$$A\lambda = b$$

$$\lambda = \mathrm{A}\backslash\mathrm{b}$$

$$F_c = J^T\lambda$$

Constrained motion

3. Constraint solving:

1st constraint force is affected only by 1st Lagrange multiplier!

$$
\begin{bmatrix} F_{c1x} \\ F_{c1y} \\ F_{c2x} \\ F_{c2y} \\ F_{c3x} \\ F_{c3y} \\ \vdots \\ F_{cnx} \\ F_{cny} \end{bmatrix}
=
\begin{array}{c}
\begin{array}{ccccc} \lambda_1 & & & & \end{array} \\
\begin{bmatrix}
-n_{1x} & 0 & & & \\
-n_{1y} & 0 & & & \\
n_{1x} & -n_{2x} & \cdots & & \\
n_{1y} & -n_{2y} & & & \\
0 & n_{2x} & & & \\
0 & n_{2y} & & & \\
\vdots & & \ddots & \vdots & \\
 & & & -n_{(n-1)x} & 0 \\
 & & & -n_{(n-1)y} & 0 \\
 & & \cdots & n_{(n-1)x} & -n_{nx} \\
 & & & n_{(n-1)y} & -n_{ny} \\
 & & & 0 & n_{nx} \\
 & & & 0 & n_{ny}
\end{bmatrix}
\end{array}
\begin{bmatrix} \boxed{\lambda_1} \\ \lambda_2 \\ \vdots \\ \lambda_{n-1} \\ \lambda_n \end{bmatrix}
$$

Lagrange multipliers (constraint forces' magnitudes)

They are the solution of this system!

$$F_c = J^T \lambda$$

Constrained motion

3. Constraint solving:

2nd constraint force is affected by first 2 Lagrange multipliers!

$$\begin{bmatrix} F_{c1x} \\ F_{c1y} \\ F_{c2x} \\ F_{c2y} \\ F_{c3x} \\ F_{c3y} \\ \vdots \\ F_{cnx} \\ F_{cny} \end{bmatrix} = \begin{array}{c} \begin{array}{ccccc} \lambda_1 & \lambda_2 & & \phantom{-n_{(n-1)x}} & \phantom{-n_{nx}} \end{array} \\ \begin{bmatrix} -n_{1x} & 0 & & & \\ -n_{1y} & 0 & & & \\ n_{1x} & -n_{2x} & \cdots & & \\ n_{1y} & -n_{2y} & & & \\ 0 & n_{2x} & & & \\ 0 & n_{2y} & & & \\ & \vdots & \ddots & \vdots & \\ & & & -n_{(n-1)x} & 0 \\ & & & -n_{(n-1)y} & 0 \\ & & \cdots & n_{(n-1)x} & -n_{nx} \\ & & & n_{(n-1)y} & -n_{ny} \\ & & & 0 & n_{nx} \\ & & & 0 & n_{ny} \end{bmatrix} \end{array} \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_{n-1} \\ \lambda_n \end{bmatrix}$$

Lagrange multipliers (constraint forces' magnitudes)

They are the solution of this system!

$$F_c = J^T \lambda$$

Constrained motion

3. Constraint solving:

$$J\frac{1}{M}J^T\lambda = -\beta\frac{1}{\Delta t^2}S - \frac{1}{\Delta t}JV - J\frac{1}{M}F_{ext}$$

$$A\lambda = b$$

$$\lambda = \mathrm{A\backslash b}$$

numerical solution of system of linear equations

$$F_c = J^T\lambda$$

integration:

$$\vec{v}_i = \vec{v}_{i-1} + \Delta t\frac{\vec{F}_{ext} + \vec{F}_c}{m}$$

$$\vec{p}_i = \vec{p}_{i-1} + \Delta t\vec{v}_i$$

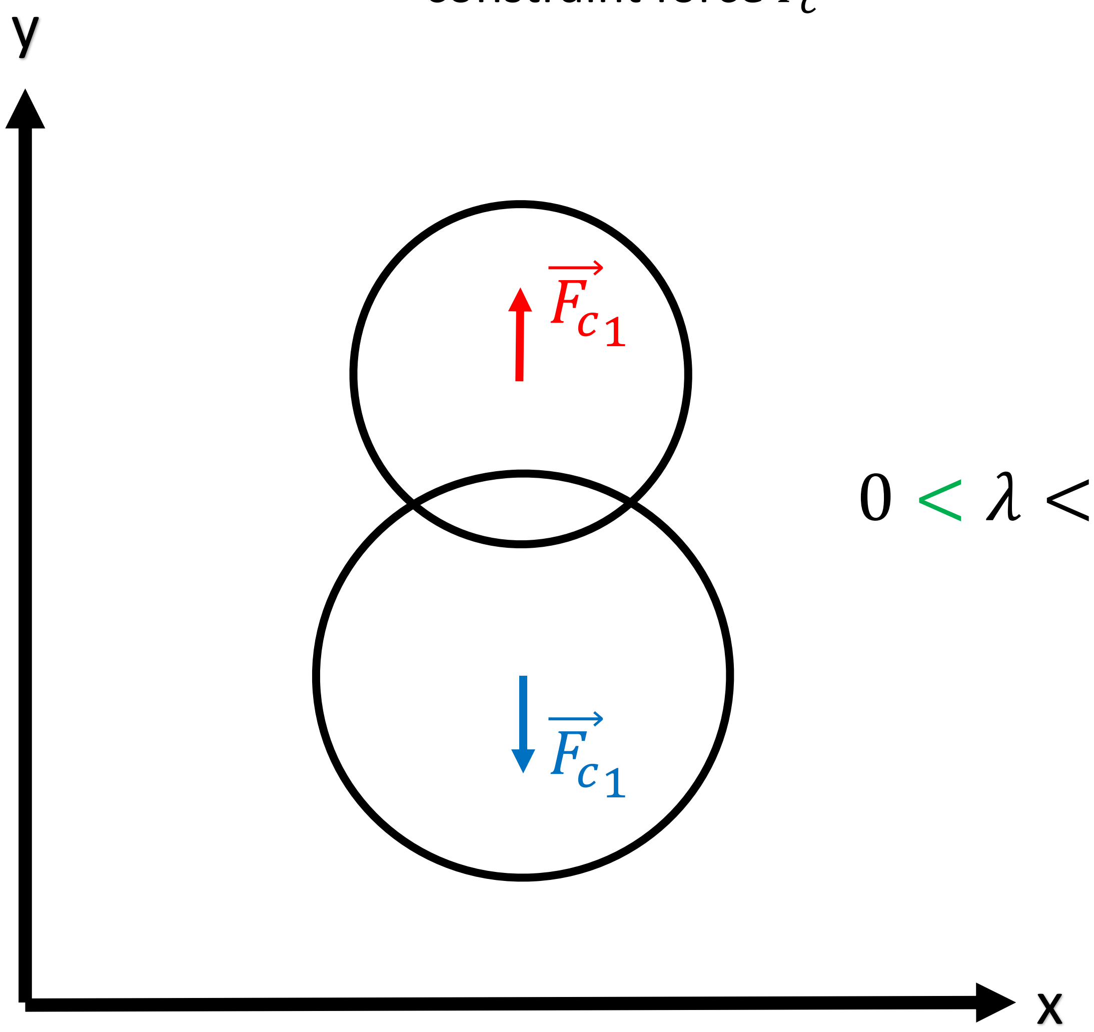
Constrained motion

3. Constraint solving:

Equality constraints:

$$c(\vec{p}_1, \vec{p}_2) = 0$$

resulting positive (pushing) constraint force F_c



$$0 < \lambda < \infty$$

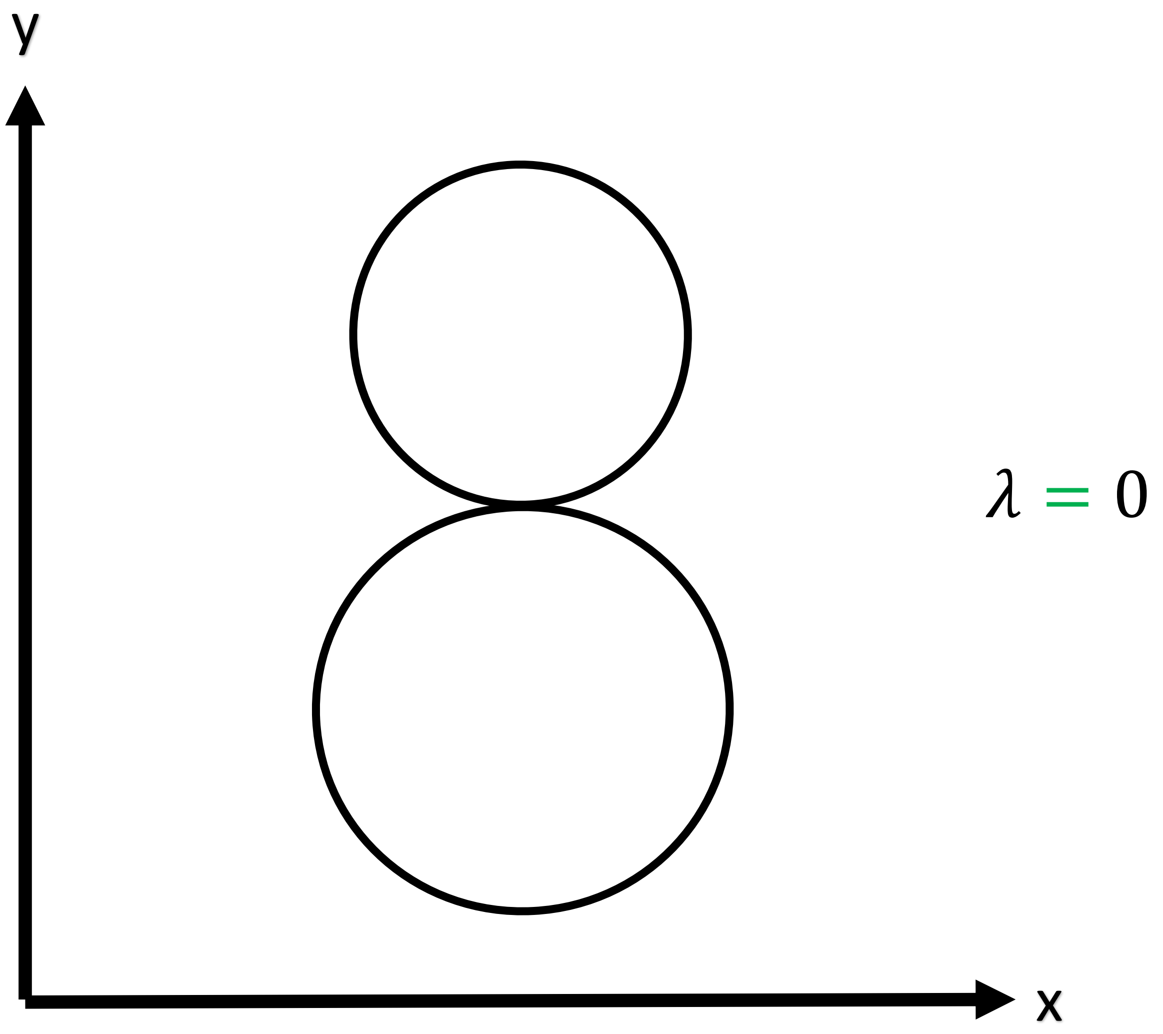
Constrained motion

3. Constraint solving:

Equality constraints:

$$c(\vec{p}_1, \vec{p}_2) = 0$$

Penetration resolved!



Constrained motion

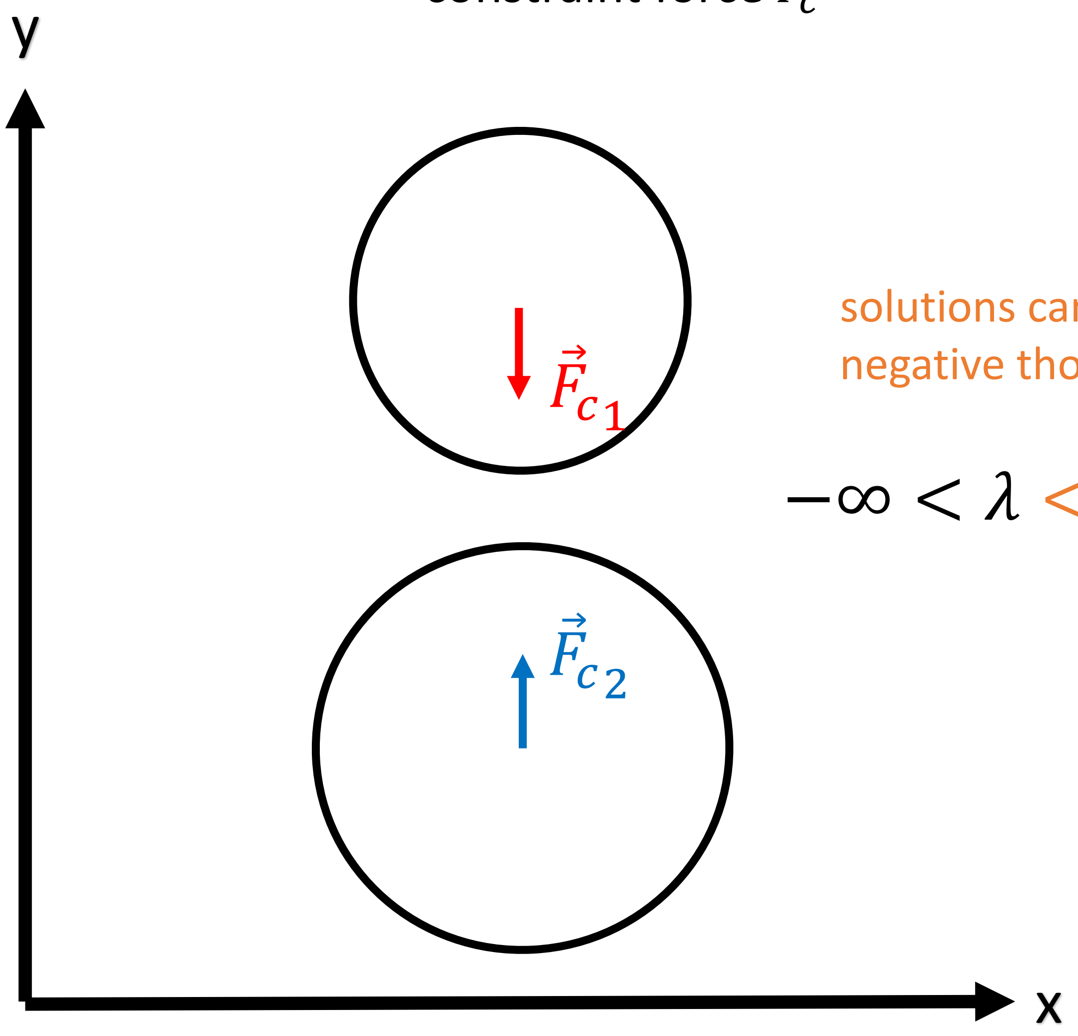
3. Constraint solving:

Equality constraints:

$$c(\vec{p}_1, \vec{p}_2) = 0$$

However separation is prevented too!

resulting negative (pulling) constraint force F_c



solutions can be negative though

$$-\infty < \lambda < 0$$

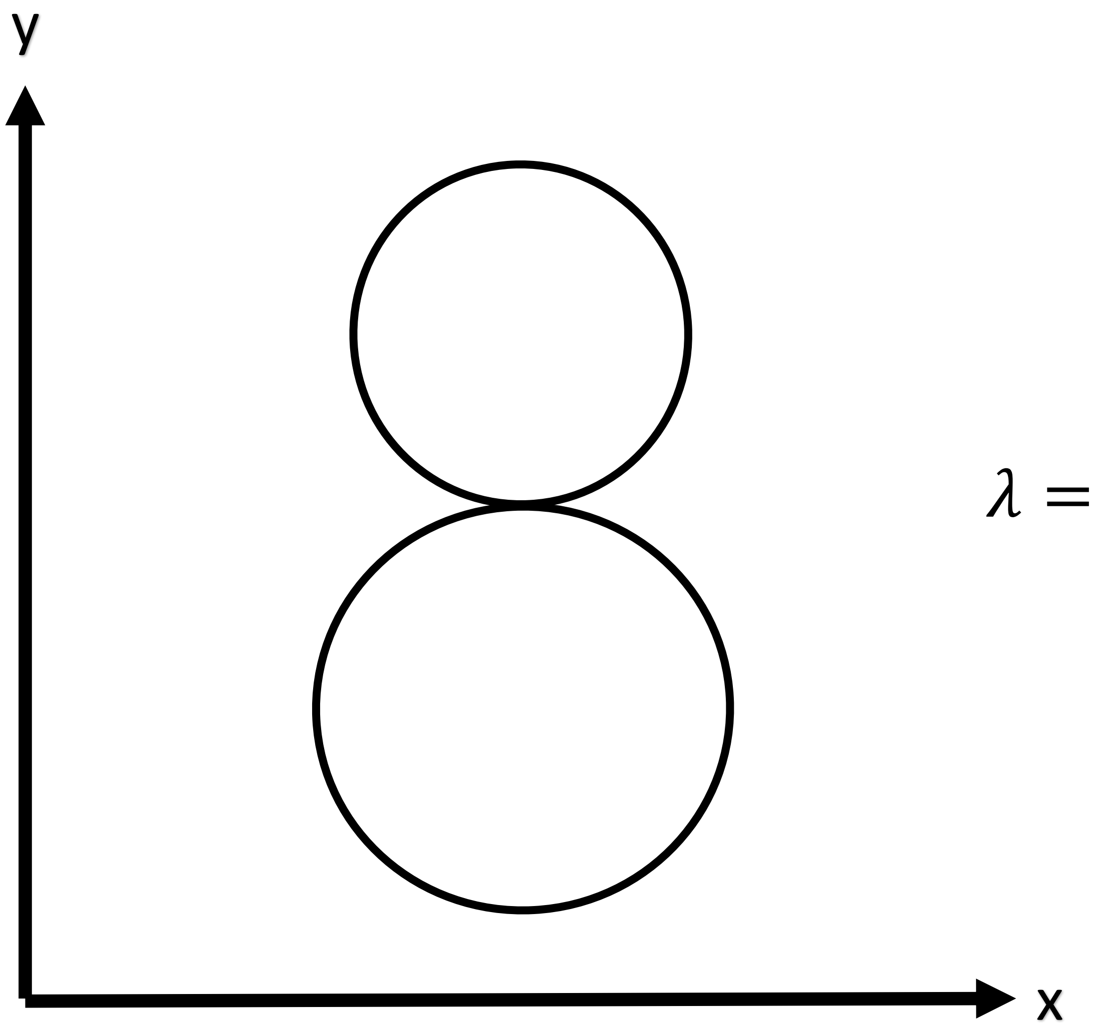
Constrained motion

3. Constraint solving:

Equality constraints:

$$c(\vec{p}_1, \vec{p}_2) = 0$$

However separation is prevented too!



$$\lambda = 0$$

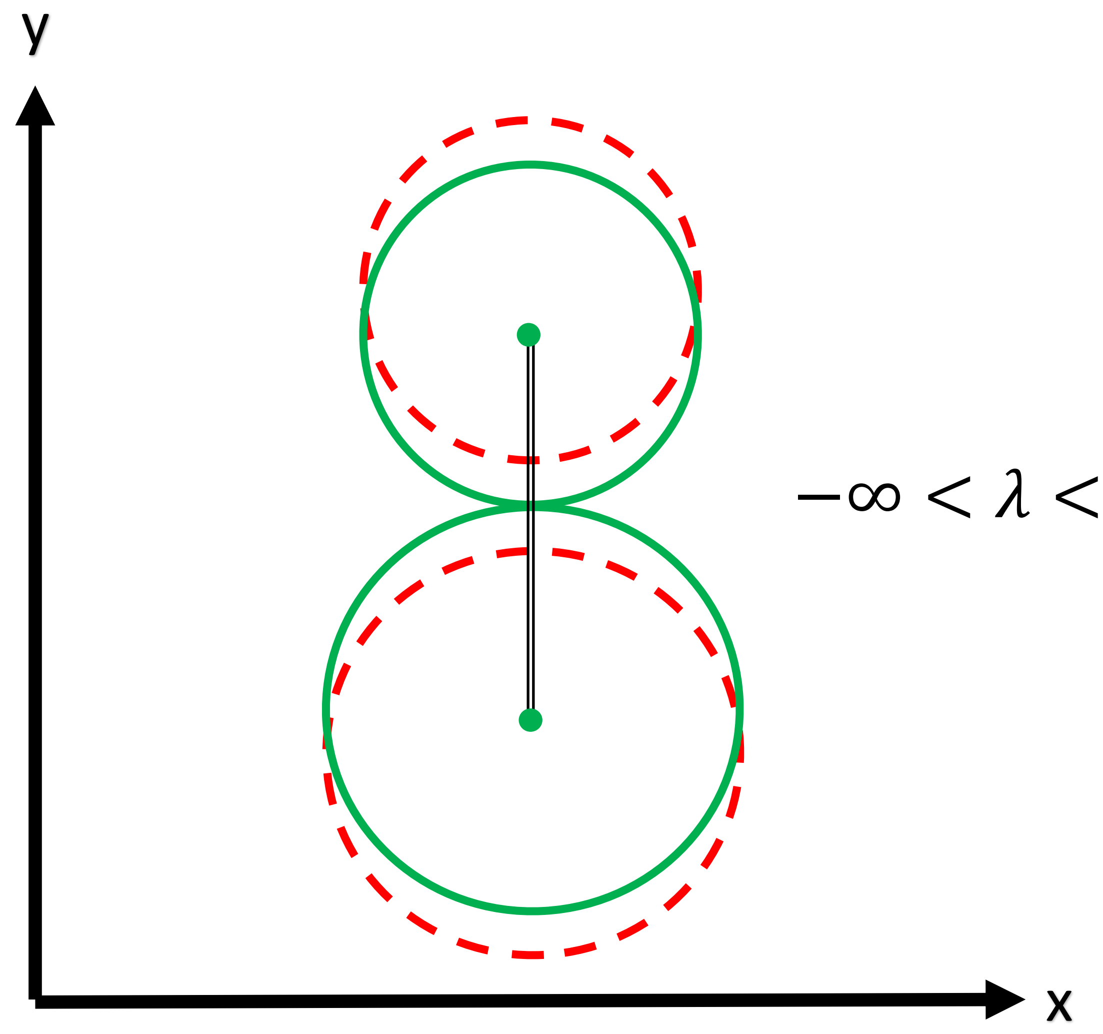
Constrained motion

3. Constraint solving:

Equality constraints:

$$c(\vec{p}_1, \vec{p}_2) = 0$$

e. g. distance joints!



$$-\infty < \lambda < \infty$$

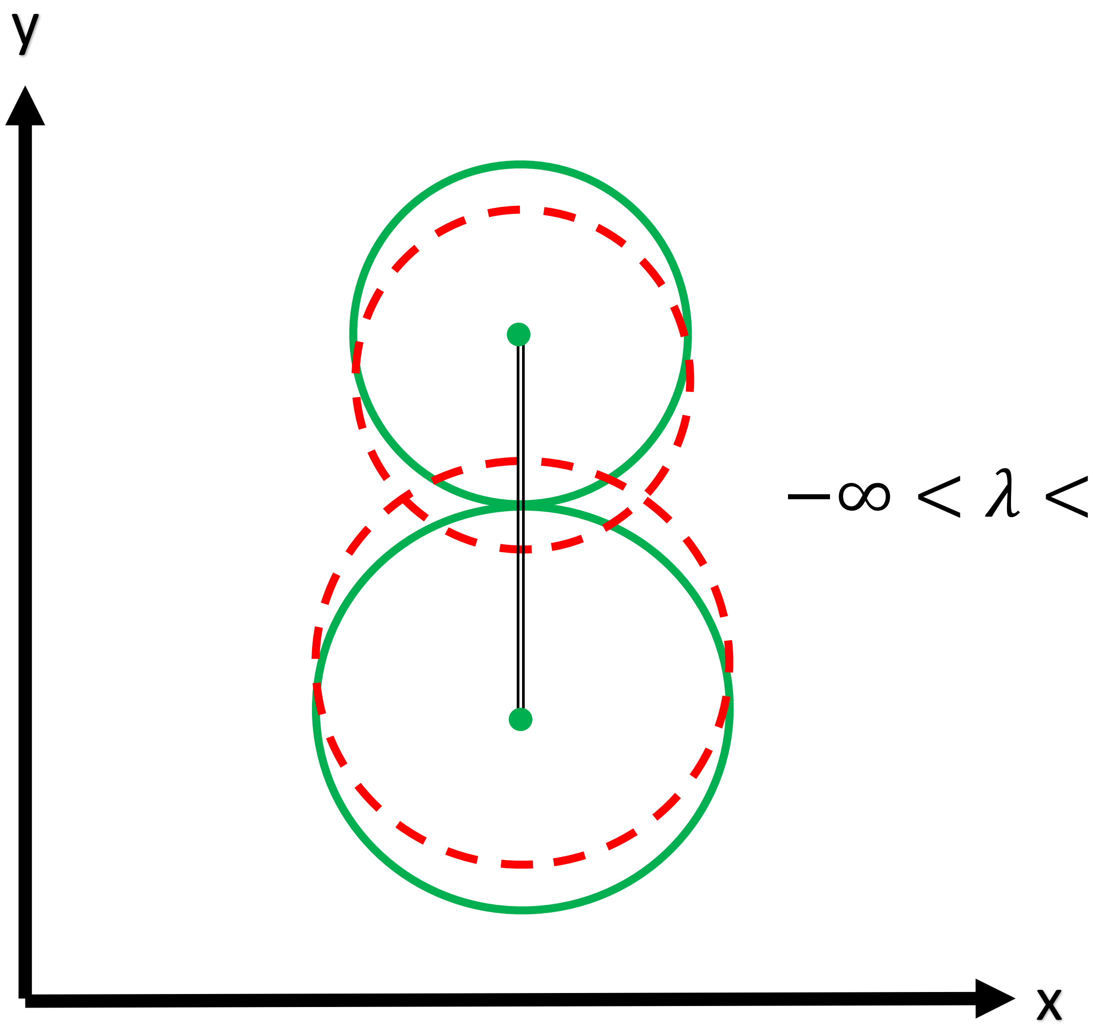
Constrained motion

3. Constraint solving:

Equality constraints:

$$c(\vec{p}_1, \vec{p}_2) = 0$$

e.g. distance joints!



$$-\infty < \lambda < \infty$$

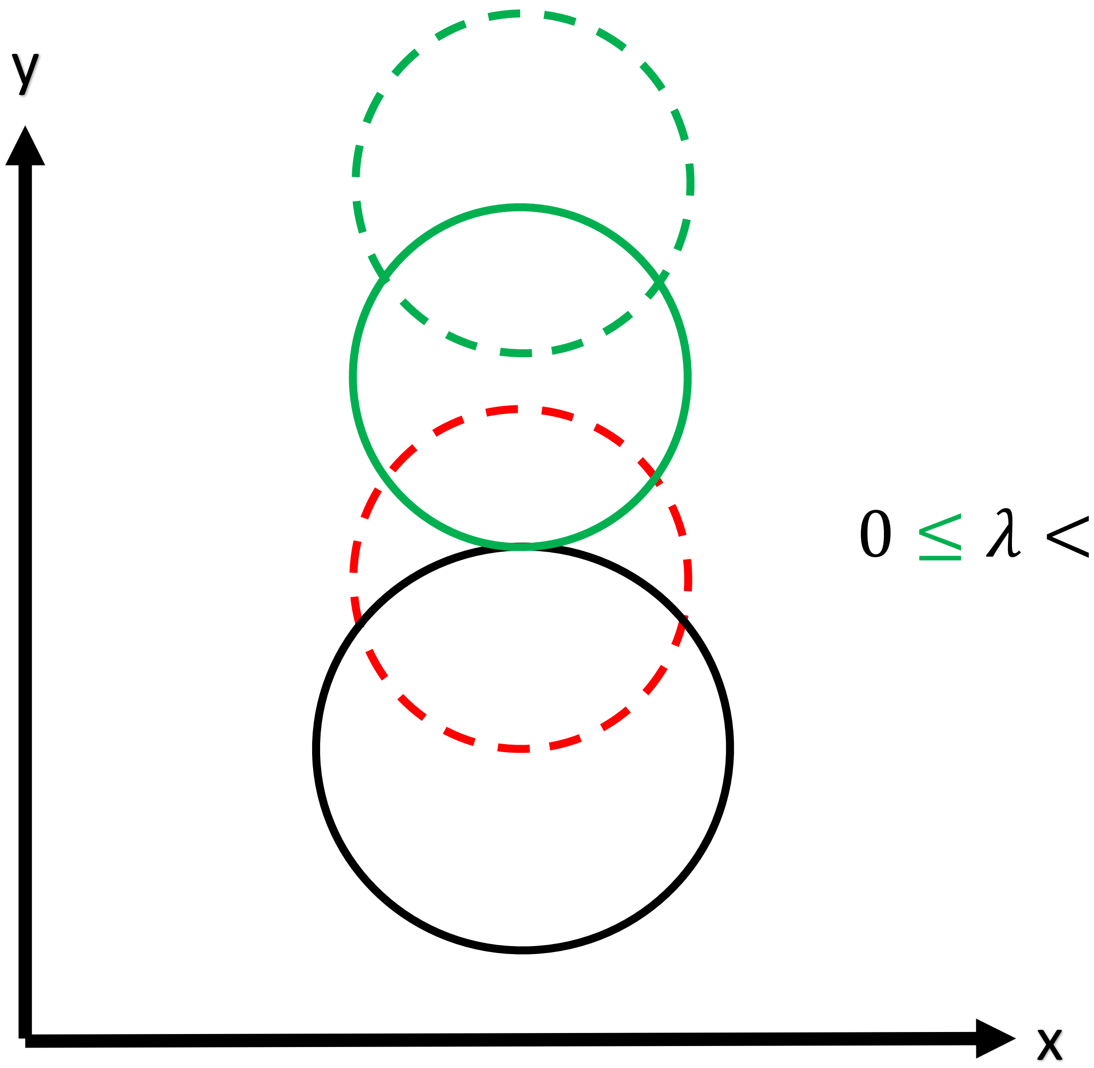
Constrained motion

3. Constraint solving:

Inequality constraints:

$$c(\vec{p}_1, \vec{p}_2) \geq 0$$

e.g. contact constraints!



$$0 \leq \lambda < \infty$$

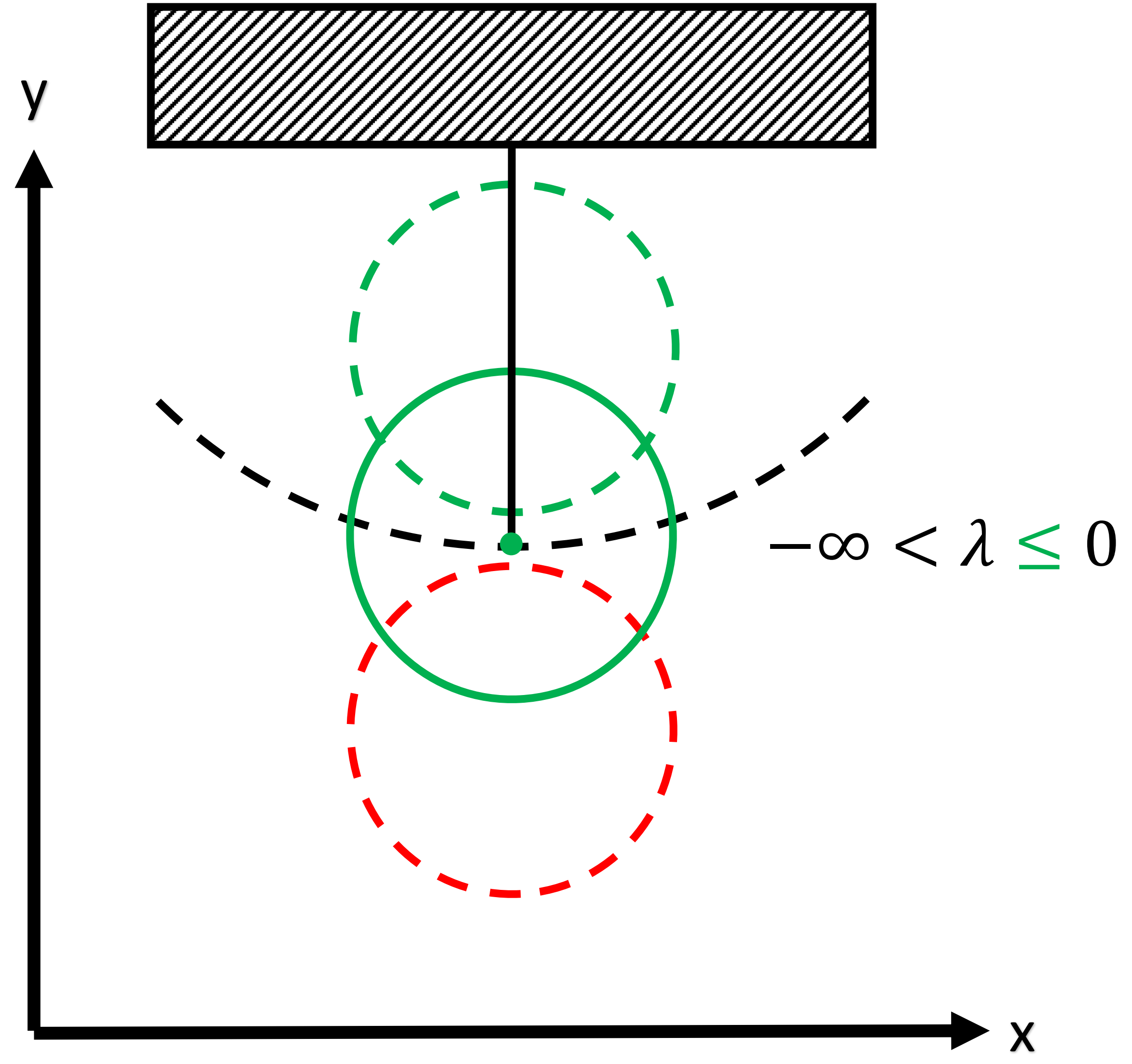
Constrained motion

3. Constraint solving:

Inequality constraints:

$$c(\vec{p}_1, \vec{p}_2) \leq 0$$

e.g. pendulums!



Constrained motion

3. Constraint solving:
   - System of linear equations

$$J\frac{1}{M}J^T\lambda = -\beta\frac{1}{\Delta t^2}S - \frac{1}{\Delta t}JV - J\frac{1}{M}F_{ext}$$

$$A\lambda = b$$

can only solve system of equality constraints!

Constrained motion

3. Constraint solving:

Equality: $c(\ldots) = 0$

Inequality type $\geq$: $c(\ldots) \geq 0$

Inequality type $\leq$: $c(\ldots) \leq 0$

Constrained motion

3. Constraint solving:

Equality: $A\lambda = b$

Inequality type $\geq$: $A\lambda \geq b$

Inequality type $\leq$: $A\lambda \leq b$

Constrained motion

3. Constraint solving:
   I. transform inequalities into equalities

required
(attempts to match F_c)

slack variable (used to describe inequalities as equalities)

Equality: $A\lambda + F_c = b \quad F_c = 0$

Inequality type $\geq$: $A\lambda + F_c = b \quad F_c \leq 0$

Inequality type $\leq$: $A\lambda + F_c = b \quad F_c \geq 0$

Constrained motion

3. Constraint solving:
    I. transform inequalities into equalities

found (matches F_c)

slack variable (is no longer needed)

Equality:	$A\lambda + F_c = b$	$F_c = 0$	$-\infty \leq \lambda \leq \infty$
Inequality type $\geq$:	$A\lambda + F_c = b$	$F_c = 0$	$0 \leq \lambda \leq \infty$
Inequality type $\leq$:	$A\lambda + F_c = b$	$F_c = 0$	$-\infty \leq \lambda \leq 0$

- Constraints are satisfied if and only if λ is within corresponding limits (no additional F_c is needed to satisfy them)

Constrained motion

3. Constraint solving:
   I. transform inequalities into equalities
   II. generalize λ limits to make them parametrized

found
(matches F_c)

slack variable (is no longer needed)

Equality:	$A\lambda + F_c = b$	$F_c = 0$	$\lambda_{min} \leq \lambda \leq \lambda_{max}$
Inequality type $\geq$:	$A\lambda + F_c = b$	$F_c = 0$	$\lambda_{min} \leq \lambda \leq \lambda_{max}$
Inequality type $\leq$:	$A\lambda + F_c = b$	$F_c = 0$	$\lambda_{min} \leq \lambda \leq \lambda_{max}$

- Constraints are satisfied if and only if λ is within corresponding limits (no additional F_c is needed to satisfy them)

Constrained motion

3. Constraint solving:
    I. transform inequalities into equalities
    II. generalize λ limits to make them parametrized
    III. complementarity

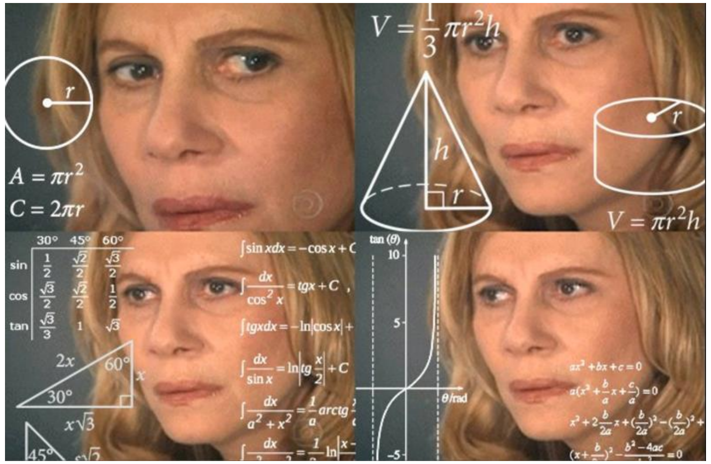
has reached limits
(unable to match F_c)

slack variable (is needed once more)

Equality:	$A\lambda + F_c = b$	$F_c \neq 0$	$\lambda_{min} = \lambda \quad \vee \quad \lambda = \lambda_{max}$
Inequality type $\geq$:	$A\lambda + F_c = b$	$F_c \geq 0$	$\lambda_{min} = \lambda$
Inequality type $\leq$:	$A\lambda + F_c = b$	$F_c \leq 0$	$\lambda = \lambda_{max}$

- Constraints are violated if and only if λ has reached corresponding limits (a constraint force F_c should act to resolve them)

Constrained motion



3. Constraint solving:
   - (mixed) linear complementarity problem (MLCP)

$$J\frac{1}{M}J^T\lambda + F_c = -\beta\frac{1}{\Delta t^2}S - \frac{1}{\Delta t}JV - J\frac{1}{M}F_{ext}$$

$$A\lambda + F_c = b$$

$$F_{c_i} = 0 \Leftrightarrow \lambda_{min_i} \le \lambda_i \le \lambda_{max_i} \quad ,\forall i \in N$$

$$F_{c_i} \ge 0 \Leftrightarrow \lambda_i = \lambda_{min_i} \quad ,\forall i \in N$$

$$F_{c_i} \le 0 \Leftrightarrow \lambda_i = \lambda_{max_i} \quad ,\forall i \in N$$

- Constraints are satisfied if and only if λ is within corresponding limits (no additional F_c is needed to satisfy them)
- Constraints are violated if and only if λ has reached corresponding limits (a constraint force F_c should act to resolve them)

Constrained motion

3. Constraint solving:
   - (mixed) linear complementarity problem (MLCP)

$c(\dots) = 0$	$c(\dots) \geq 0$	$c(\dots) \leq 0$
$-\infty < \lambda < \infty$	$\lambda \geq 0$	$\lambda \leq 0$

In simple terms:

λ should only be found within constraints' corresponding limits!

$$
\begin{gathered}
c_1(\dots) = 0 \\
c_2(\dots) \geq 0 \\
c_3(\dots) \leq 0 \\
\vdots \\
c_n(\dots) = 0
\end{gathered}
\qquad
\lambda_{min} = \begin{bmatrix} -\infty \\ 0 \\ -\infty \\ \vdots \\ -\infty \end{bmatrix}
\lambda_{max} = \begin{bmatrix} \infty \\ \infty \\ 0 \\ \vdots \\ \infty \end{bmatrix}
$$

Constrained motion

3. Constraint solving:
   - *Gauss-Seidel* method (can only find solution to system of linear equations):

```
function [x, it] = gs(A, b, x0, itMax, errMax)
    rows = length(A);

    x = x0;
    for it = 1:itMax
        for row = 1:rows
            x(row) = 1/A(row,row)*(b(row) - A(row,1:row - 1)*x(1:row - 1) - A(row,row + 1:end)*x0(row + 1:end));
        end
        if abs(x - x0) < errMax
            return
        end
        x0 = x;
    end
end
```

Constrained motion

3. Constraint solving:

- *Projected Gauss-Seidel* method (can solve MLCP):

```
function [x, it] = projectedGS(A, b, x0, xMin, xMax, itMax, errMax)
    rows = length(A);

    x = x0;
    for it = 1:itMax
        for row = 1:rows
            x(row) = 1/A(row,row)*(b(row) - A(row,1:row - 1)*x(1:row - 1) - A(row,row + 1:end)*x0(row + 1:end));
            % projection
            if x(row) < xMin(row)
                x(row) = xMin(row);
            end
            if x(row) > xMax(row)
                x(row) = xMax(row);
            end
        end
        if abs(x - x0) < errMax
            return
        end
        x0 = x;
    end
end
```

More algorithms exist, such as Lemke algorithm (*Carlton E. Lemke*)

Constrained motion

3. Constraint solving:

$$J \frac{1}{M} J^T \lambda = -\beta \frac{1}{\Delta t^2} S - \frac{1}{\Delta t} JV - J \frac{1}{M} F_{ext}$$

$$A\lambda = b$$

$$\lambda = \mathrm{A} \backslash \mathrm{b}$$

$$F_c = J^T \lambda$$

numerical solution of MLCP
given λ_{min} i λ_{max}

integration:

$$\vec{v}_i = \vec{v}_{i-1} + \Delta t \frac{\vec{F}_{ext} + \vec{F}_c}{m}$$

$$\vec{p}_i = \vec{p}_{i-1} + \Delta t \vec{v}_i$$

Constrained motion

3. Constraint solving:

- Contact caching:

\+ There is high probability that solution will not change much in between time steps. Caching dramatically reduces number of iterations needed to reach the solution in the next time step.

\+ If system of equations is overdetermined, it could have infinite number of solutions. Without caching new solution would be found in each time step. This would introduce a *jiggle* effect.

\- Constraint violations are evaluated in each time step. Vector λ can differ in size and order in between multiple time steps. Each element λ in current time step must be matched with corresponding element from previous time step. Some kind of search algorithm must be implemented.

Constrained motion

3. Constraint solving:
   - Contact caching:

time step Δt:

$\vdots$

```
lambda1 = projectedGS(A, b, 0, lambdaMin, lambdaMax, iterations, 10^-4);
```

$\vdots$

time step $2\Delta t$:

$\vdots$

```
lambda2 = projectedGS(A, b, lambda1, lambdaMin, lambdaMax, iterations, 10^-4);
```

$\vdots$

timestep $3\Delta t$:

$\vdots$

```
lambda3 = projectedGS(A, b, lambda2, lambdaMin, lambdaMax, iterations, 10^-4);
```

$\vdots$

Constrained motion

3. Constraint solving:

- Fraction of force used to cancel out constraint violation becomes integrated into velocity and carried over to the next time step even though it is going to cancel out constraint violation in the current time step
- "Excess" velocity (which hasn't originated from external forces) is accumulated in between time steps

$$J\frac{1}{M}J^T\lambda = -\beta\frac{1}{\Delta t^2}S - \frac{1}{\Delta t}JV - J\frac{1}{M}F_{ext}$$

"artificially" introduced term to cancel out existing violation of constraints as a consequence of:
1. them being discovered after they have happened
2. numerical errors

$$A\lambda = b$$

$$\lambda = A\backslash b$$

$$F_c = J^T\lambda$$

integration:

$$V(t+\Delta t) = V + \Delta t\frac{F_{ext} + F_c}{m}$$

$$P(t+\Delta t) = P + \Delta t V(t+\Delta t)$$

Constrained motion

3. Constraint solving:

Problem should be separated:

I. Fraction of force which modifies velocities should be integrated normally

II. Fraction of force which cancels out constraints' violations should be integrated directly into positions to avoid introducing additional energy

$$J\frac{1}{M}J^T\lambda_v = -\frac{1}{\Delta t}JV - J\frac{1}{M}F_{ext}$$

$$A\lambda_v = b_v$$

$$\lambda_v = A\backslash b_v$$

$$F_{c_v} = J^T\lambda_v$$

same value!
(can be used in both systems)

$$J\frac{1}{M}J^T\lambda_p = -\beta\frac{1}{\Delta t^2}S$$

$$A\lambda_p = b_p$$

$$\lambda_p = A\backslash b_p$$

$$F_{c_p} = J^T\lambda_p$$

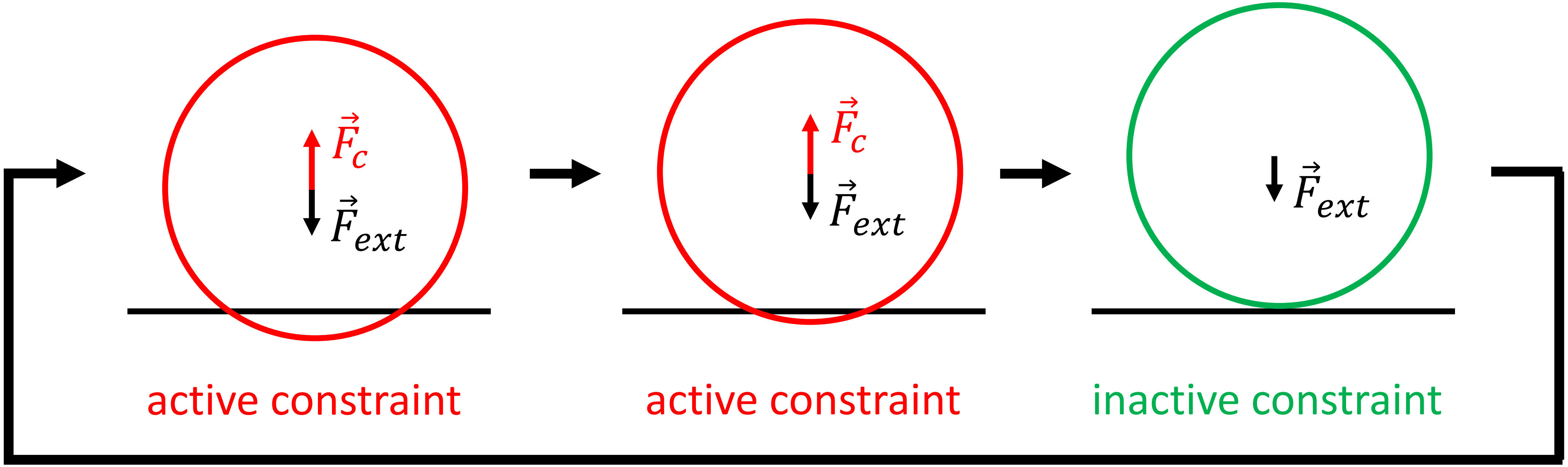
integration:

$$V(t+\Delta t) = V + \Delta t\frac{F_{ext} + F_{c_v}}{m}$$

$$P(t+\Delta t) = P + \Delta t\left(V(t+\Delta t) + \Delta t\frac{F_{c_p}}{m}\right)$$

Constrained motion

3. Constraint solving:
   - Contact constraints:



Constrained motion

3. Constraint solving:
   - Contact constraints:

$$J\frac{1}{M}J^T\lambda_p = -\beta\frac{1}{\Delta t^2}S$$

Prevent canceling out a small fraction of penetration (to keep constraints active as long as the bodies are in contact):

$$S = \sigma + \sigma_{slop}$$

real separation
(negative value)

$$0 < \sigma_{slop} \ll 1$$

Constrained motion

3. Constraint solving:
   - Contact constraints:

$$J\frac{1}{M}J^T\lambda_p = -\beta\frac{1}{\Delta t^2}S$$

Prevent positive separation or else solution will break down:

$$S = min(\sigma + \sigma_{slop}, 0)$$

Demo

https://github.com/mbeocanin/MLCP-Particle-Phyiscs-Sandbox

Q&A

https://github.com/mbeocanin