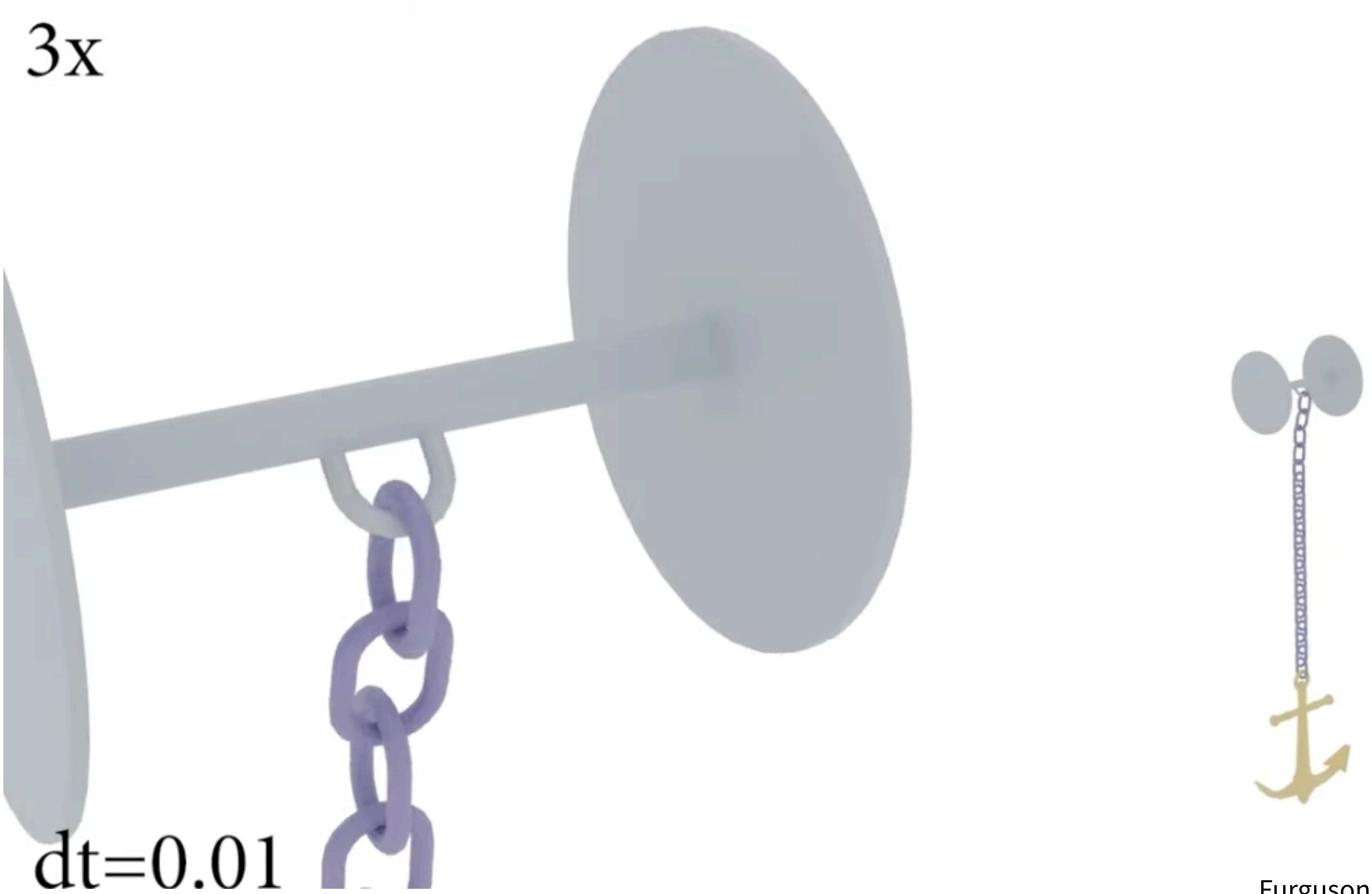
Lecture 13:

Constrained Rigid Body Systems

FUNDAMENTALS OF COMPUTER GRAPHICS

Animation & Simulation

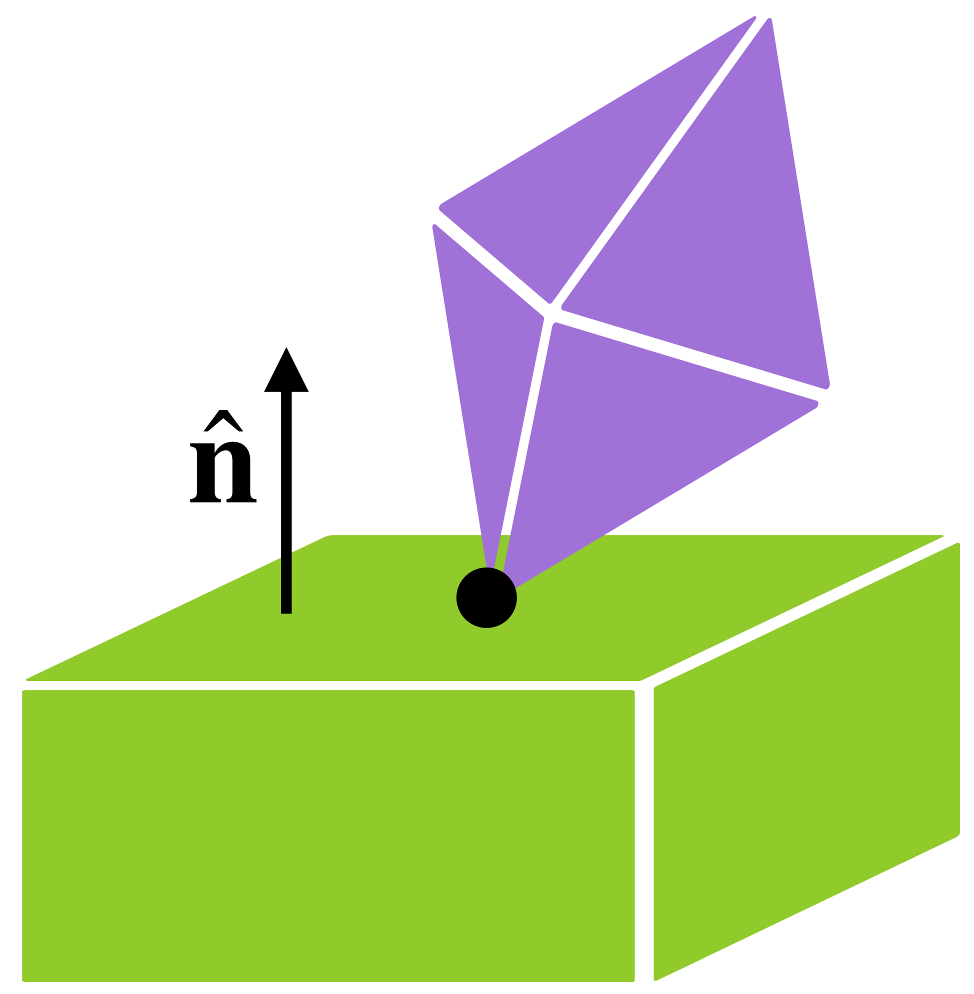
Stanford CS248B, Fall 2023



Furguson et al SIGGRAPH 2021

Collision Detector

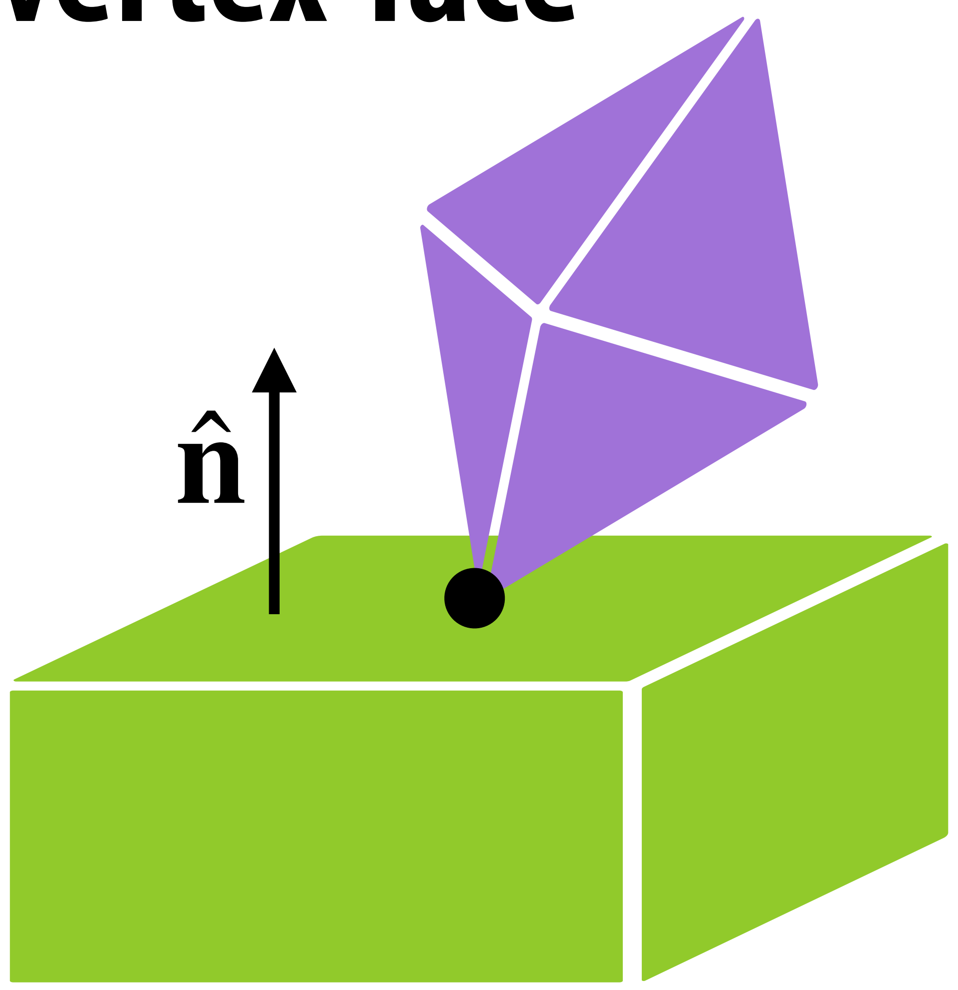
- For rigid bodies, not much different from previous lectures
  - Bounding boxes, Separating planes, Broad / Narrow phases …
  - Not the focus for today
- For each collision on the list, it should contain
  - IDs of a pair of rigid bodies in collision
  - Coordinate of the contact point
  - Normal vector at the contact point
- Today's focus: resolve the collisions, when the list is not empty



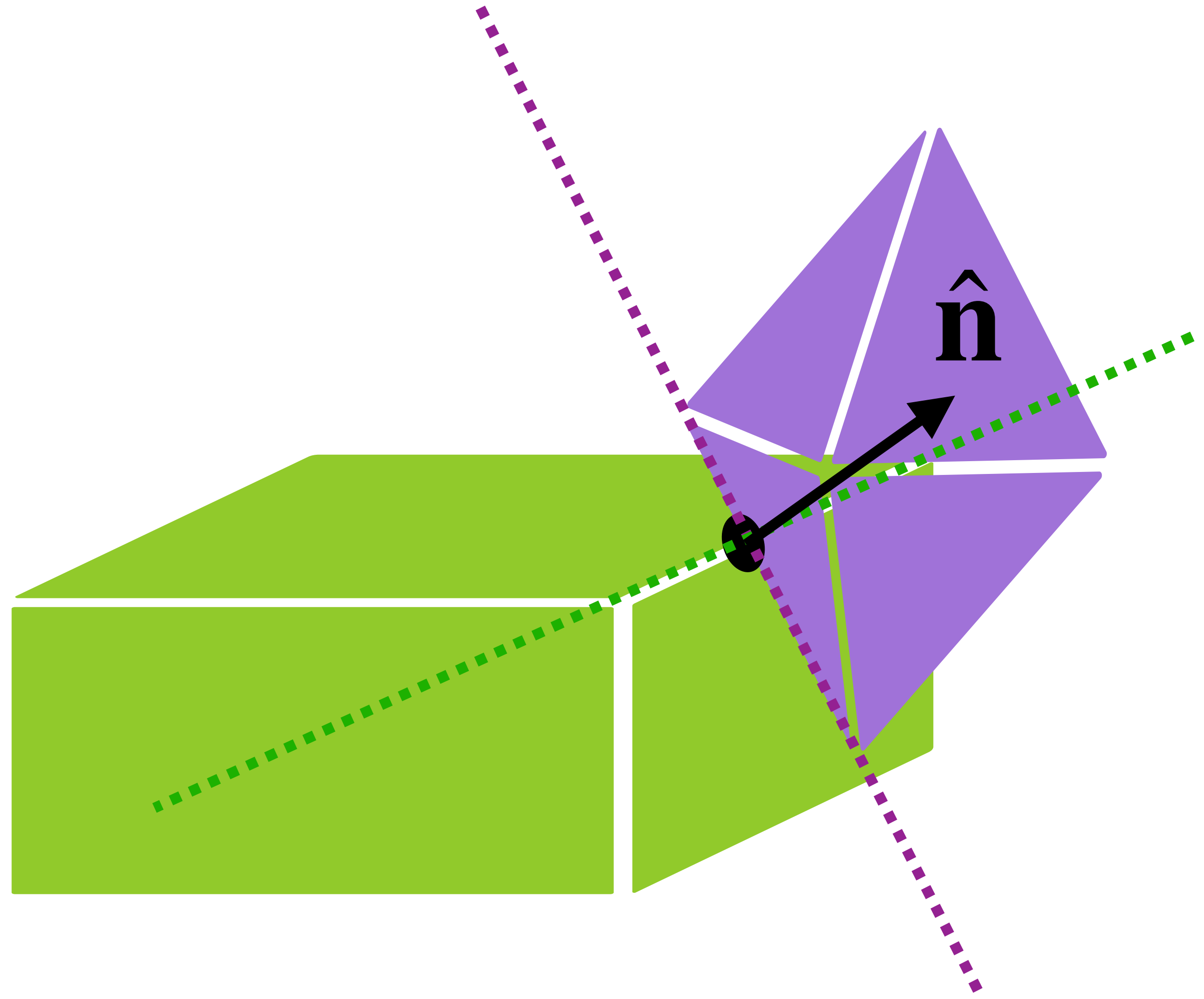
Collision is detected! What now?

- Today's focus: resolve the collisions, when the list is not empty
- Two cases in general in 3D: vertex-face & edge-edge
  - Vertex-vertex & vertex-edge are degenerate
  - What about edge-face & face-face?
- How to obtain the normal vector in each case?

vertex-face

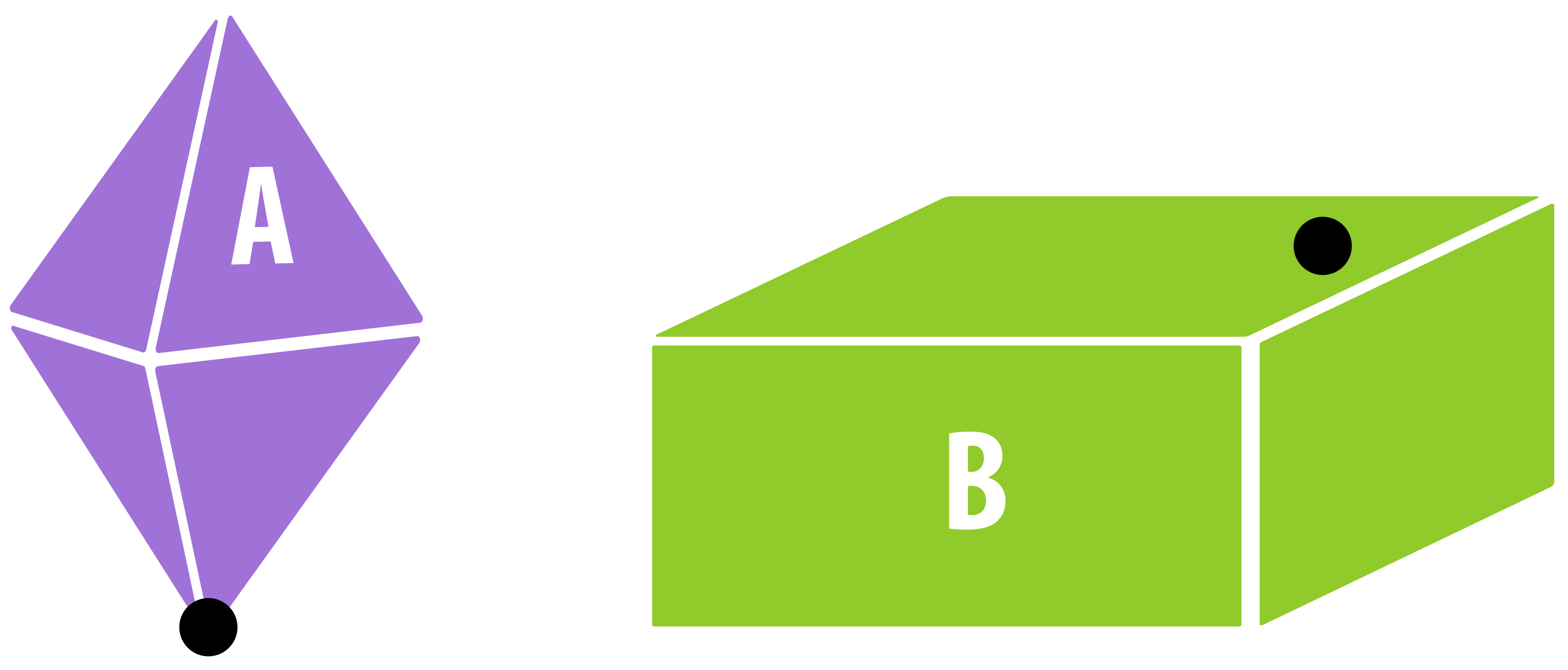


edge-edge

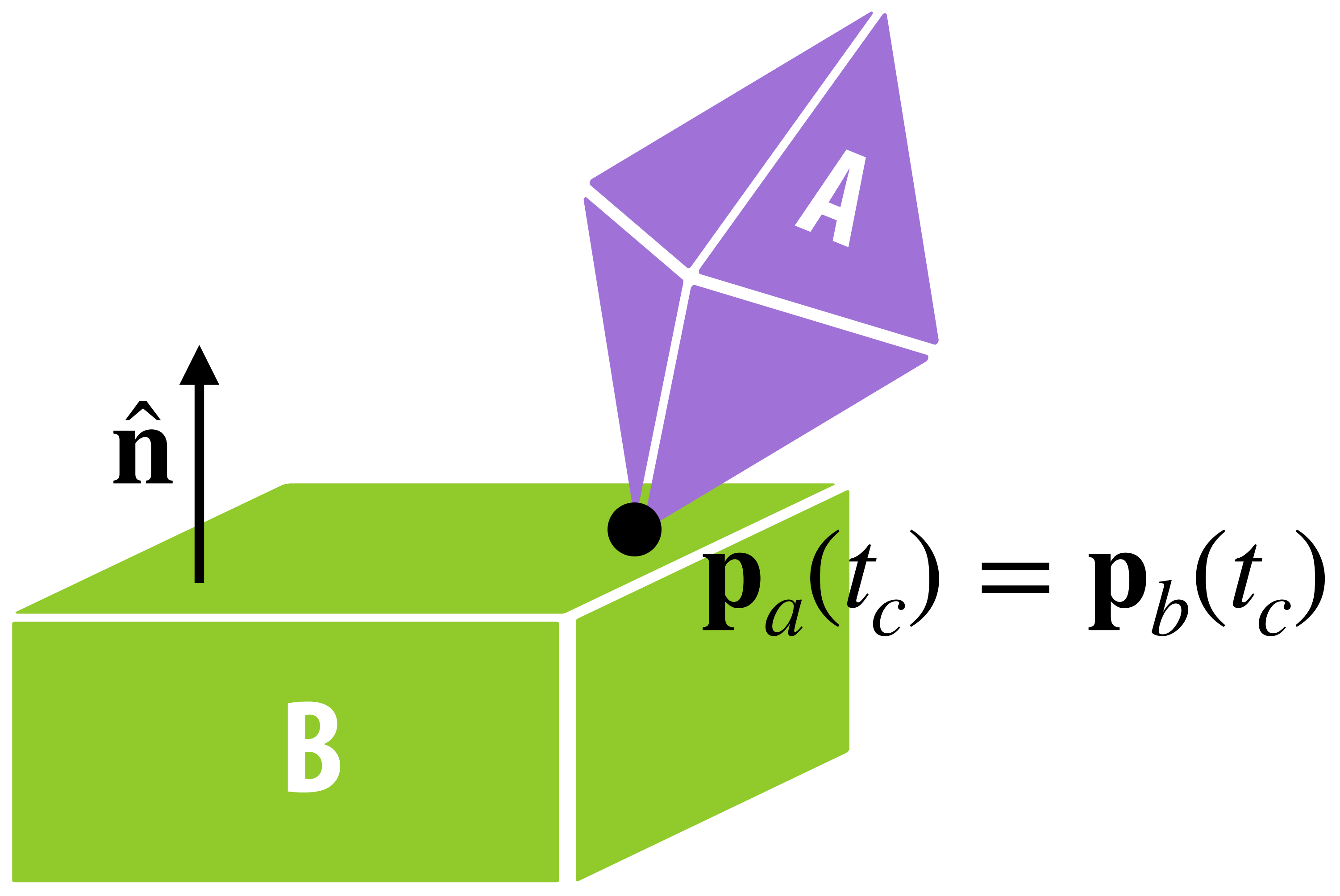


Contact Points

Collision detector tells us that a point on A and a point on B are in collision



Put in the world space…

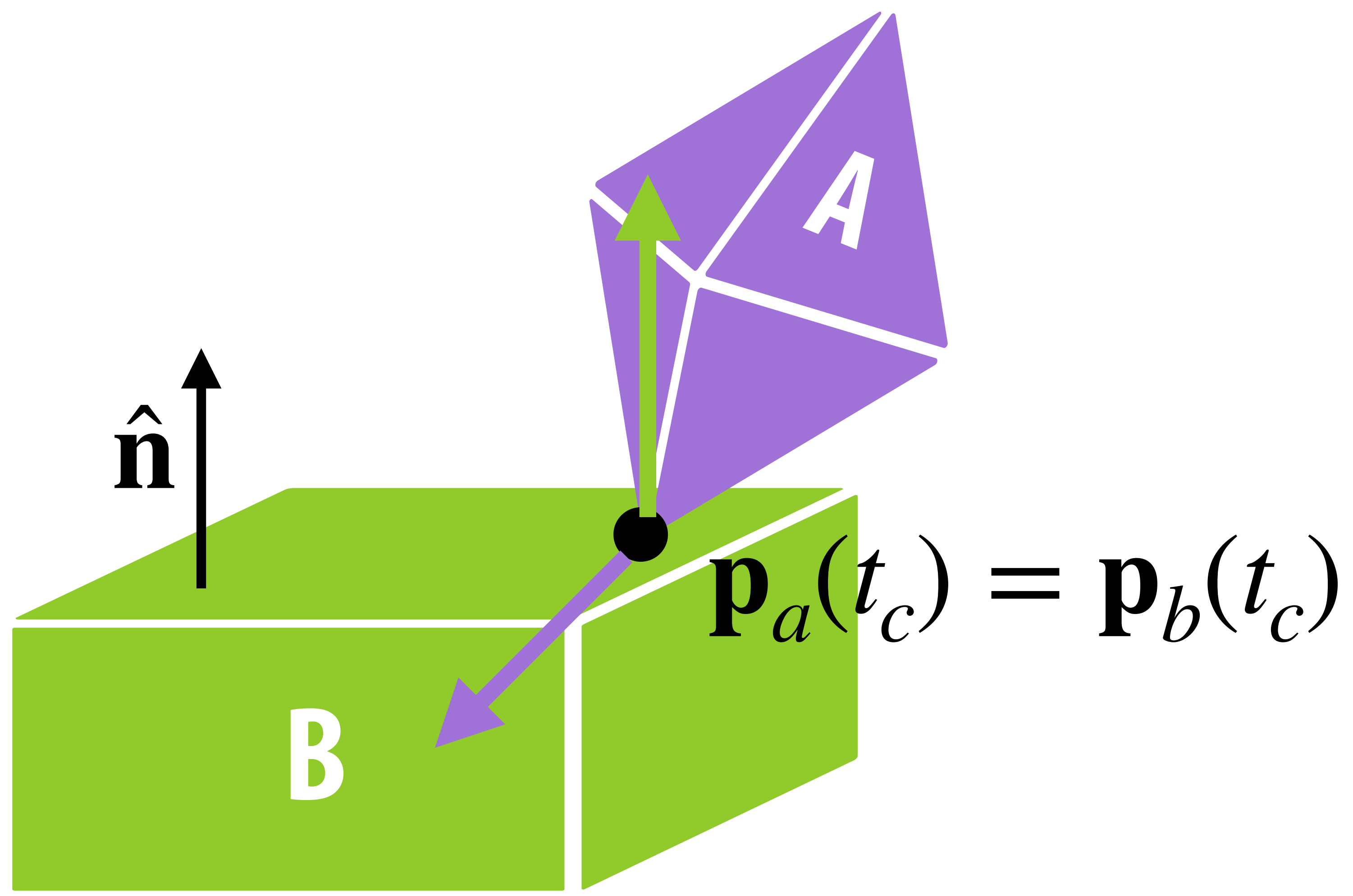


Although $\mathbf{p}_a$ and $\mathbf{p}_b$ are coincident at time t_c, the velocity of the two points may be different!

Velocity of a Contact Point

$$\dot{\mathbf{p}}_a(t_c) = \mathbf{v}_a(t_c) + \boldsymbol{\omega}_a(t_c) \times \left(\mathbf{p}_a(t_c) - \mathbf{x}_a(t_c)\right)$$

$$\dot{\mathbf{p}}_b(t_c) = \mathbf{v}_b(t_c) + \boldsymbol{\omega}_b(t_c) \times \left(\mathbf{p}_b(t_c) - \mathbf{x}_b(t_c)\right)$$



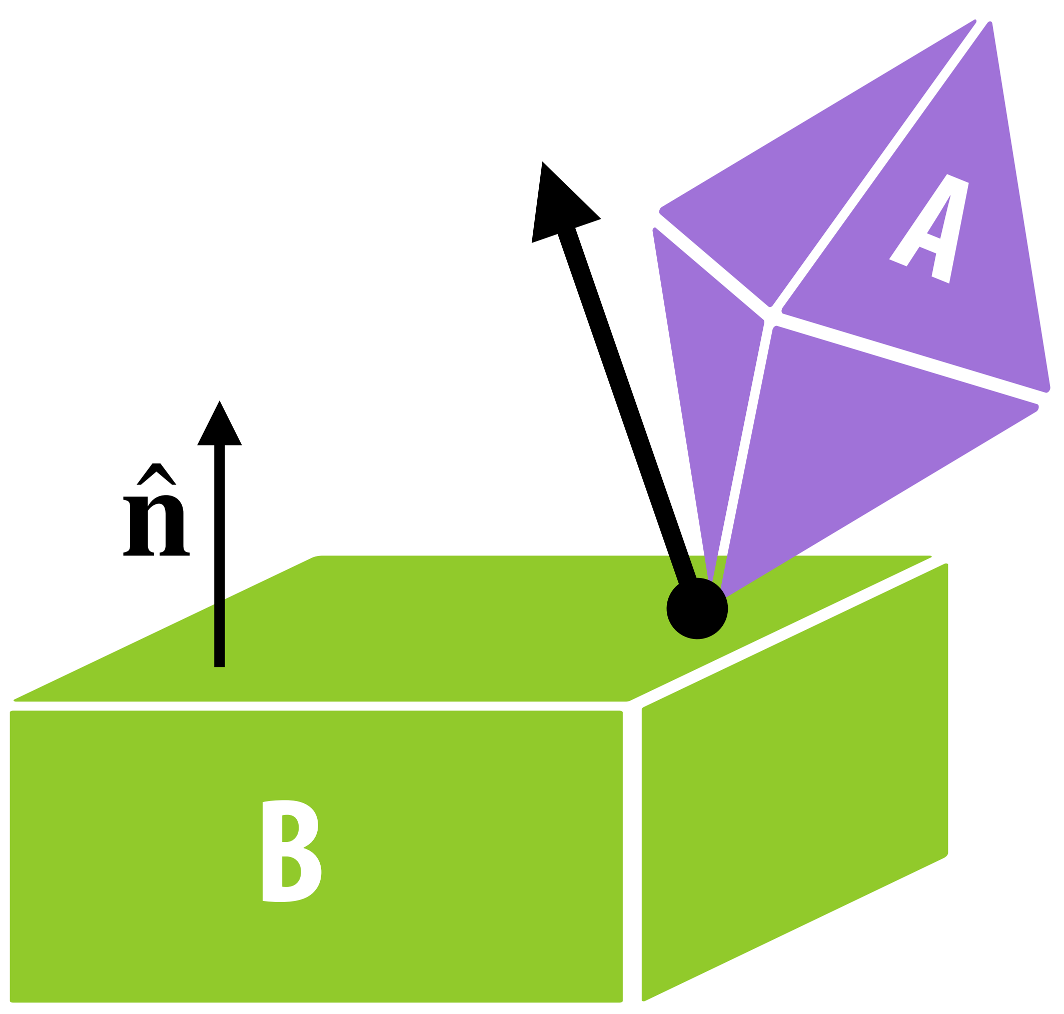
$$v_r = \hat{\mathbf{n}} \cdot \left(\dot{\mathbf{p}}_a(t_c) - \dot{\mathbf{p}}_b(t_c)\right)$$

v_r is the magnitude of the *relative* velocity in the normal direction

Relative Normal Velocity

$v_r > 0$

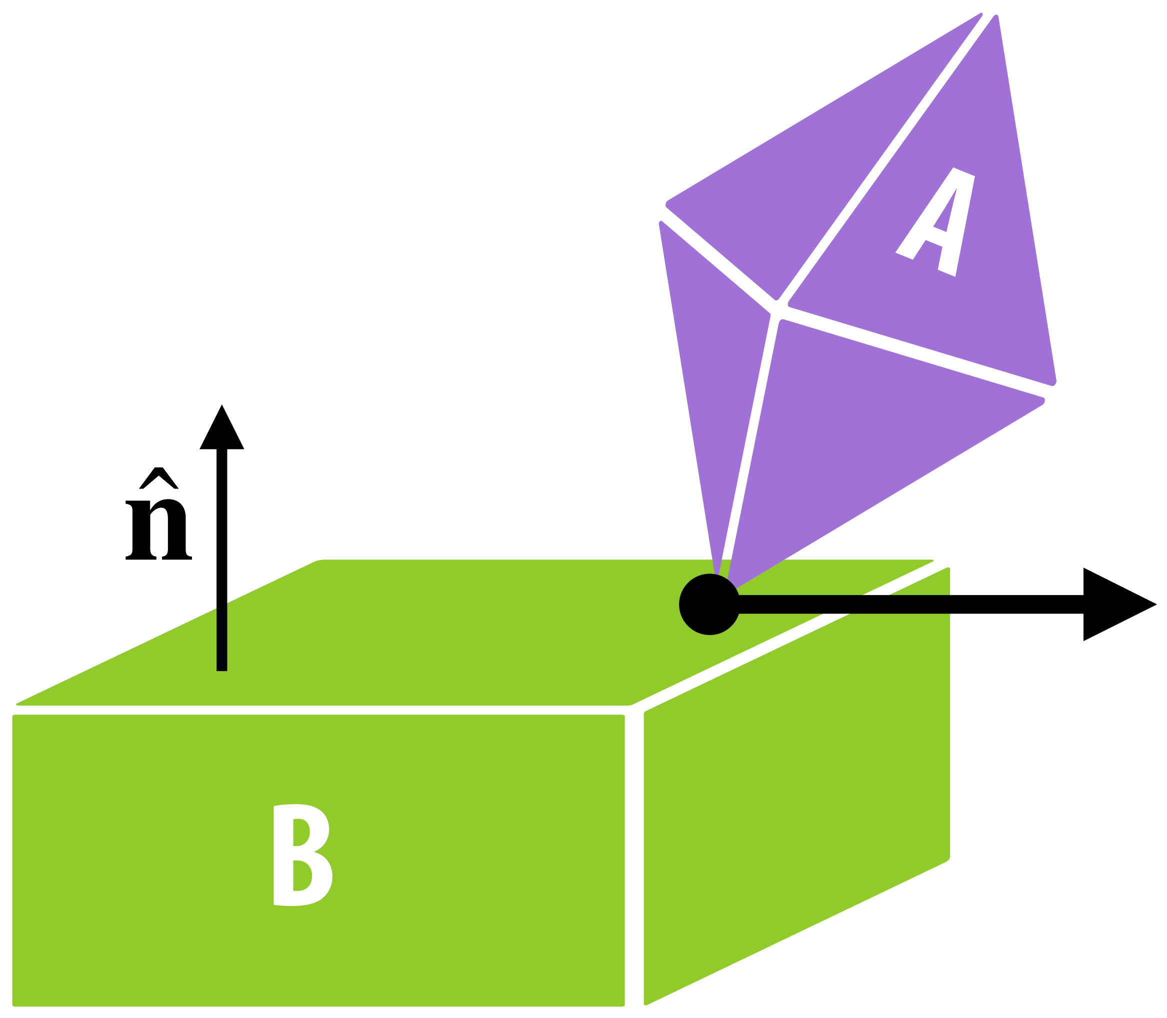
separation



Ignore & Proceed

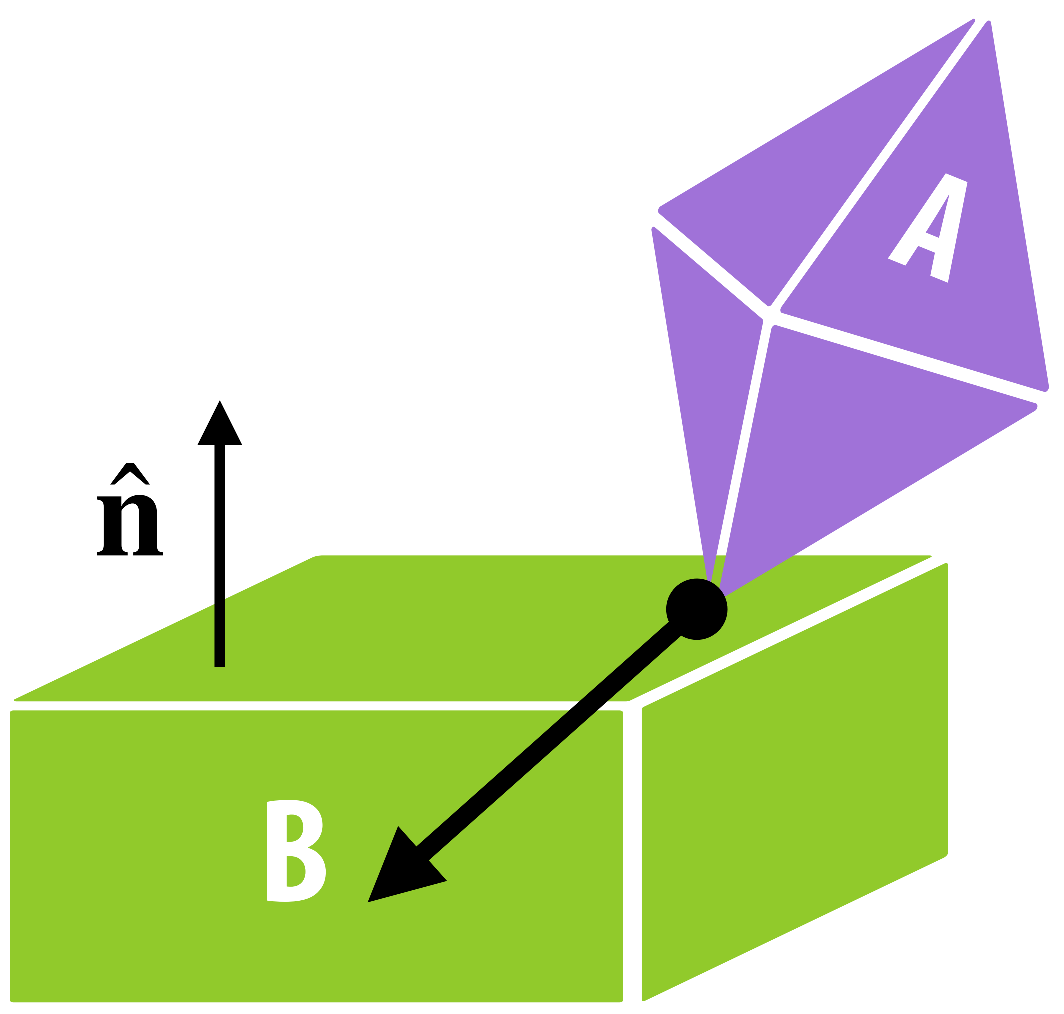
$v_r \approx 0$

resting contact

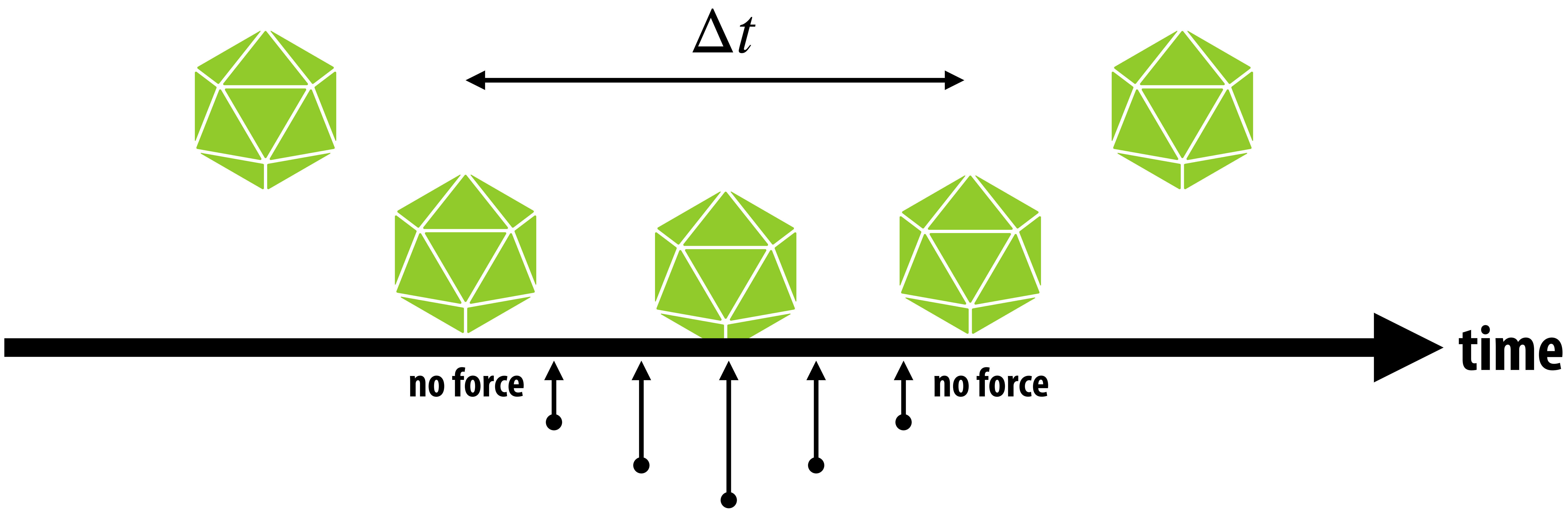


$v_r < 0$

colliding contact

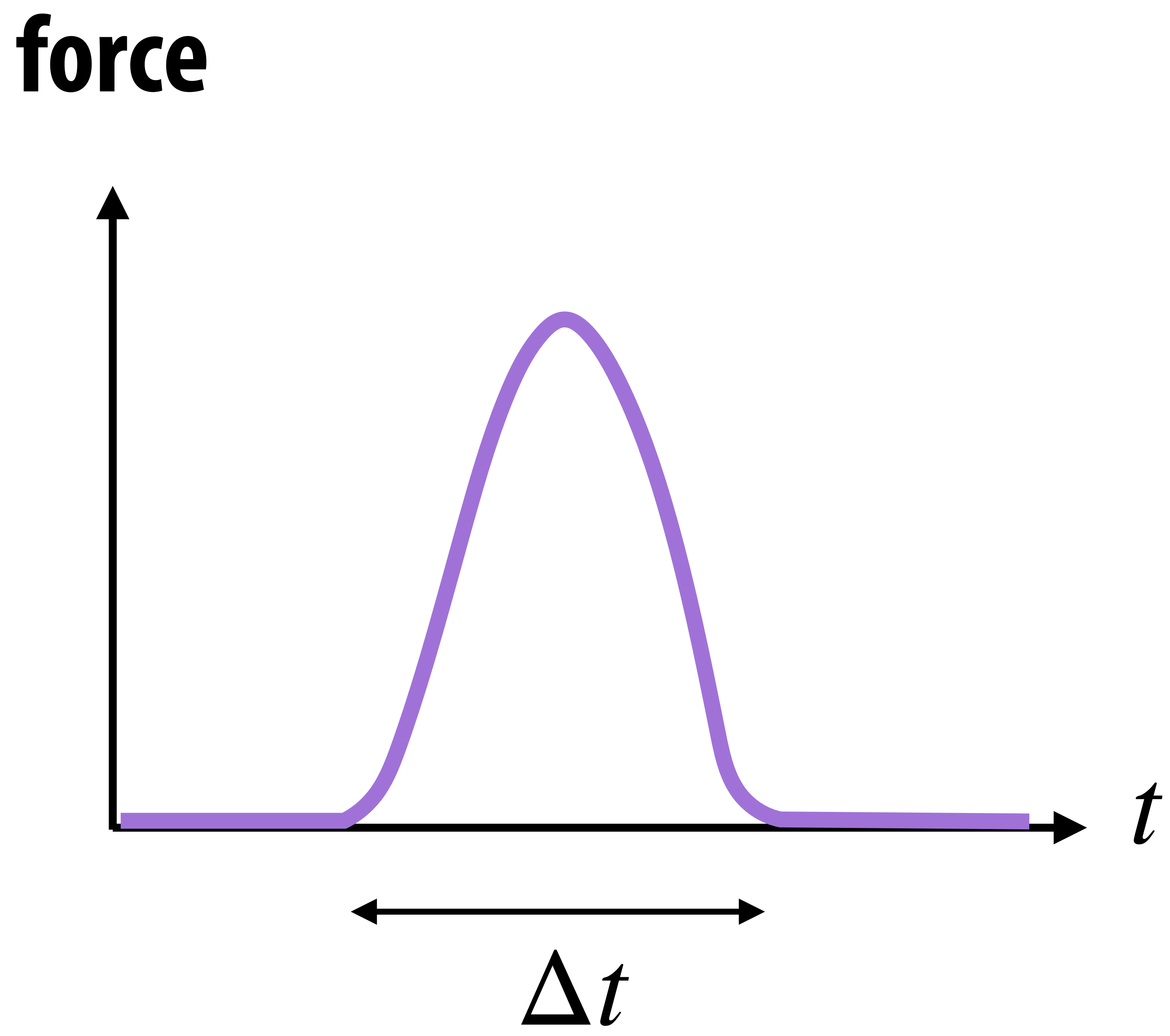


Collision Process

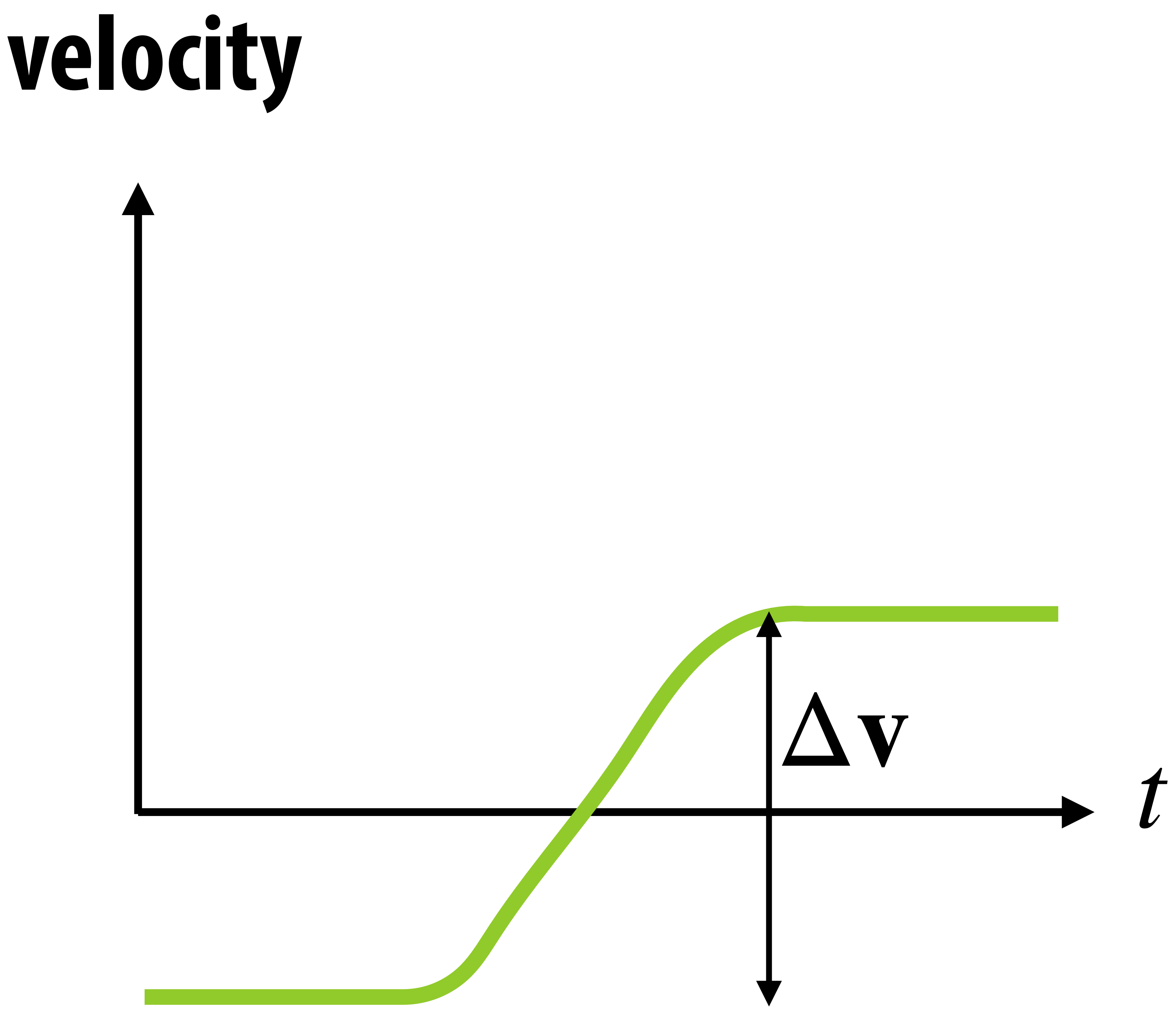


$$\mathbf{J} \equiv \int_0^{\Delta t} \mathbf{f}_t \, dt = m\Delta \mathbf{v}$$

A Soft Collision



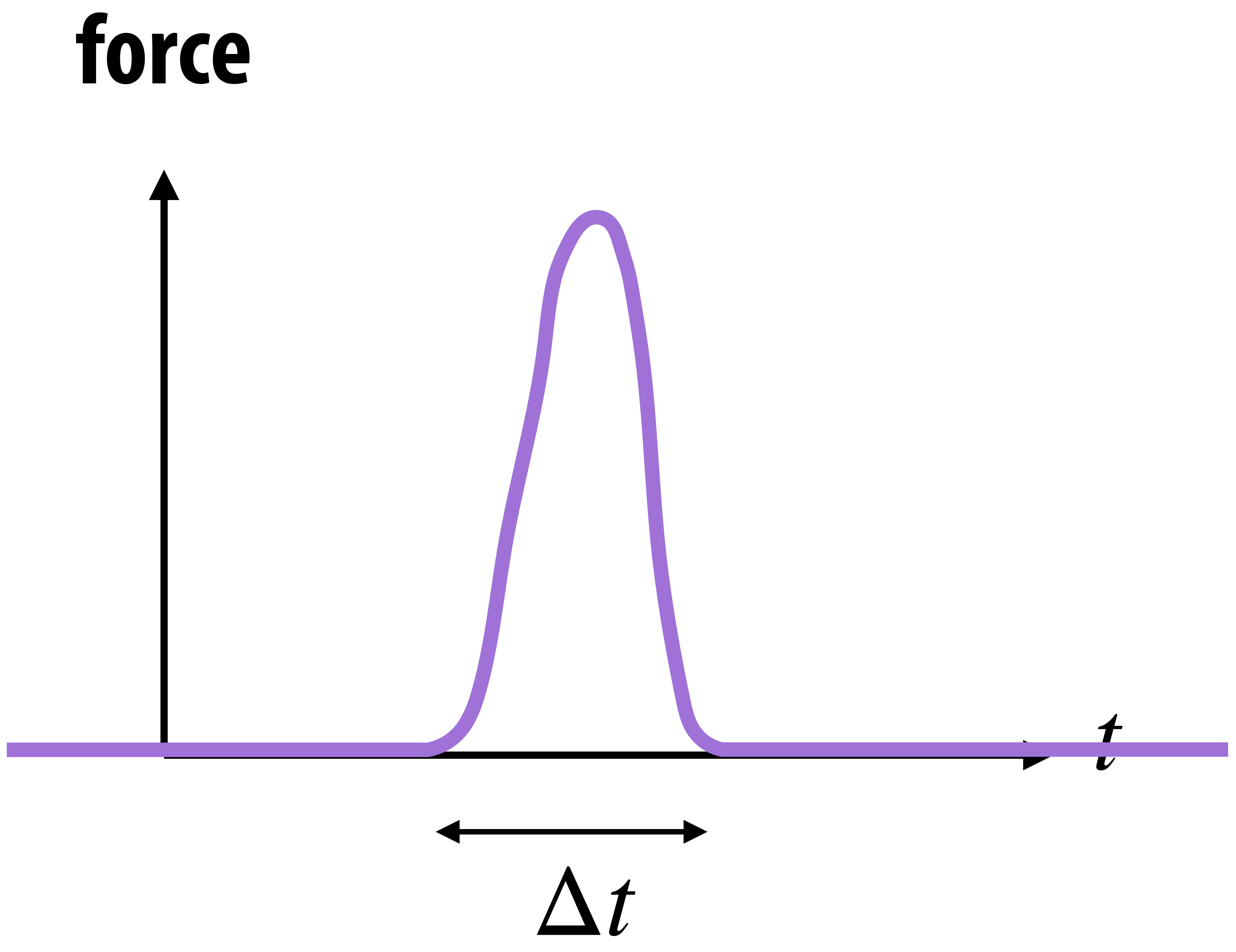
$$\mathbf{J} = \int_0^{\Delta t} \mathbf{f}_t \, dt$$

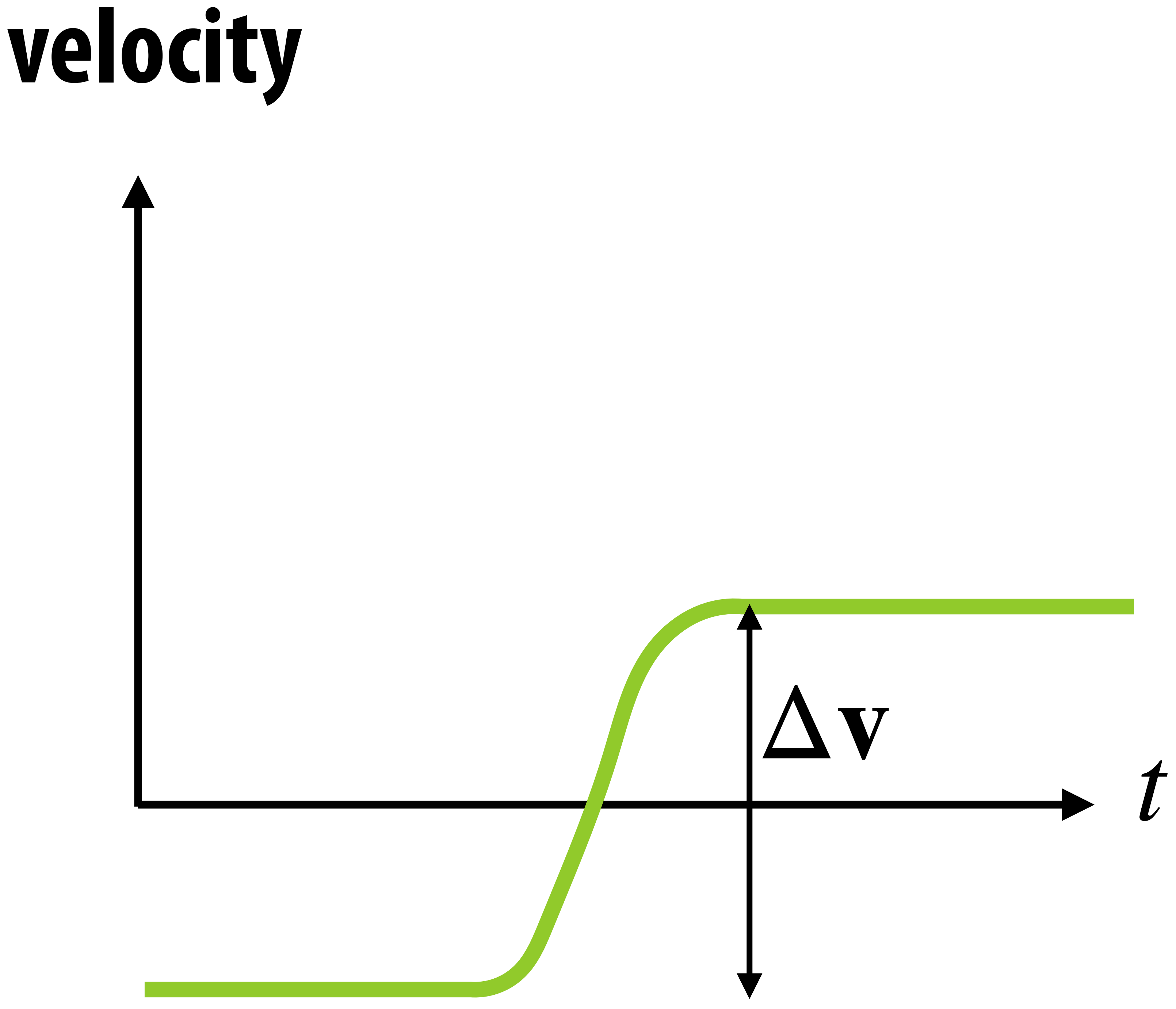


$$\mathbf{J} = m\Delta\mathbf{v}$$

What does this mean when dropping a box to floor?

A Hard Collision

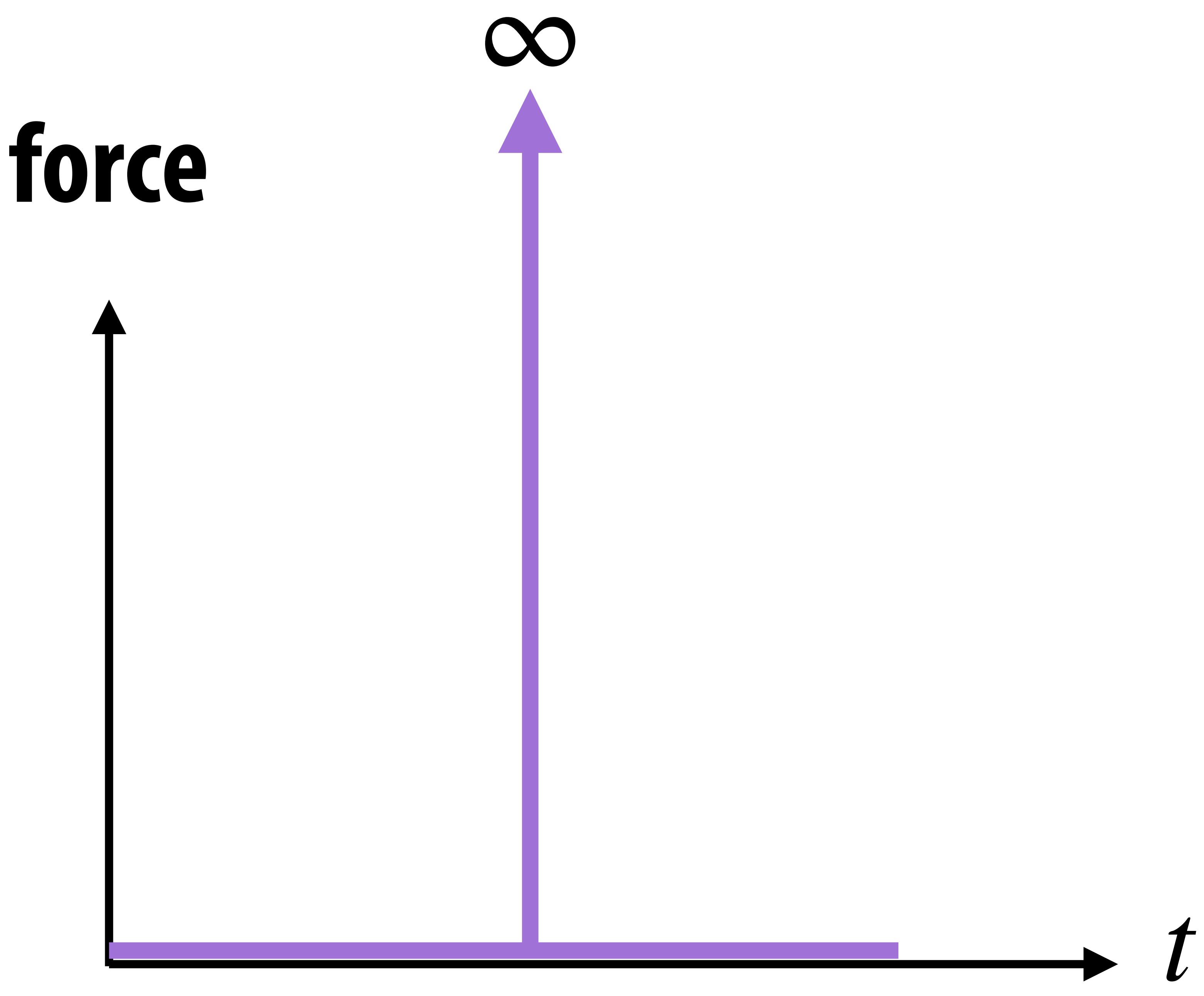




$$\mathbf{J} = \int_0^{\Delta t} \mathbf{f}_t \, dt$$

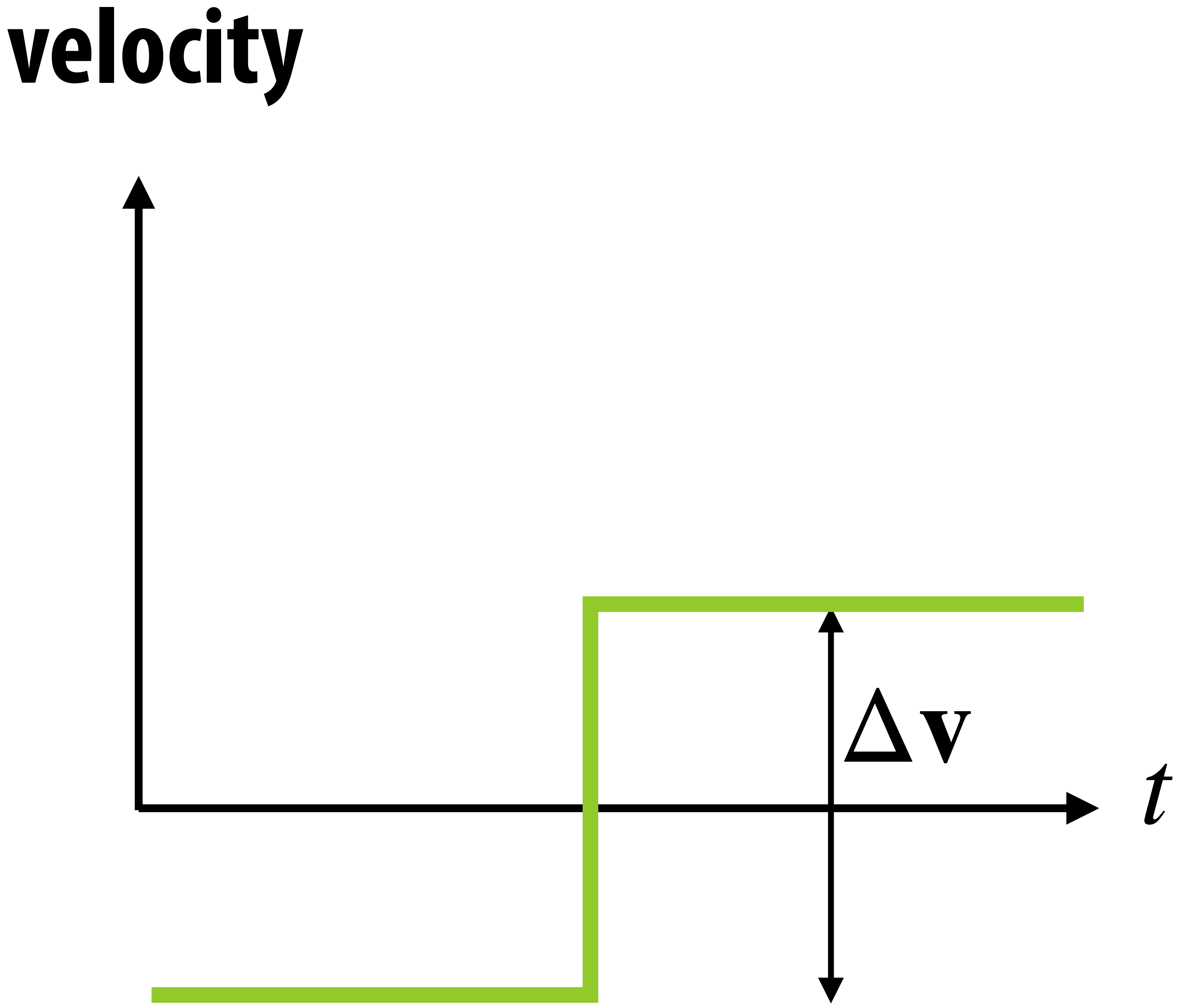
$$\mathbf{J} = m\Delta\mathbf{v}$$

An Infinitely Hard Collision



$$\Delta t = 0$$

$$\mathbf{J} = ?$$



$$\mathbf{J} = m\Delta\mathbf{v}$$

Impulse

- In the rigid body world, we want the velocity to change instantaneously if there is a collision contact.
- Use finite impulse to change velocity instead of infinite force: $\mathbf{J} = \Delta \mathbf{P} = m\Delta \mathbf{v}$

Unit of J in terms of Newton?

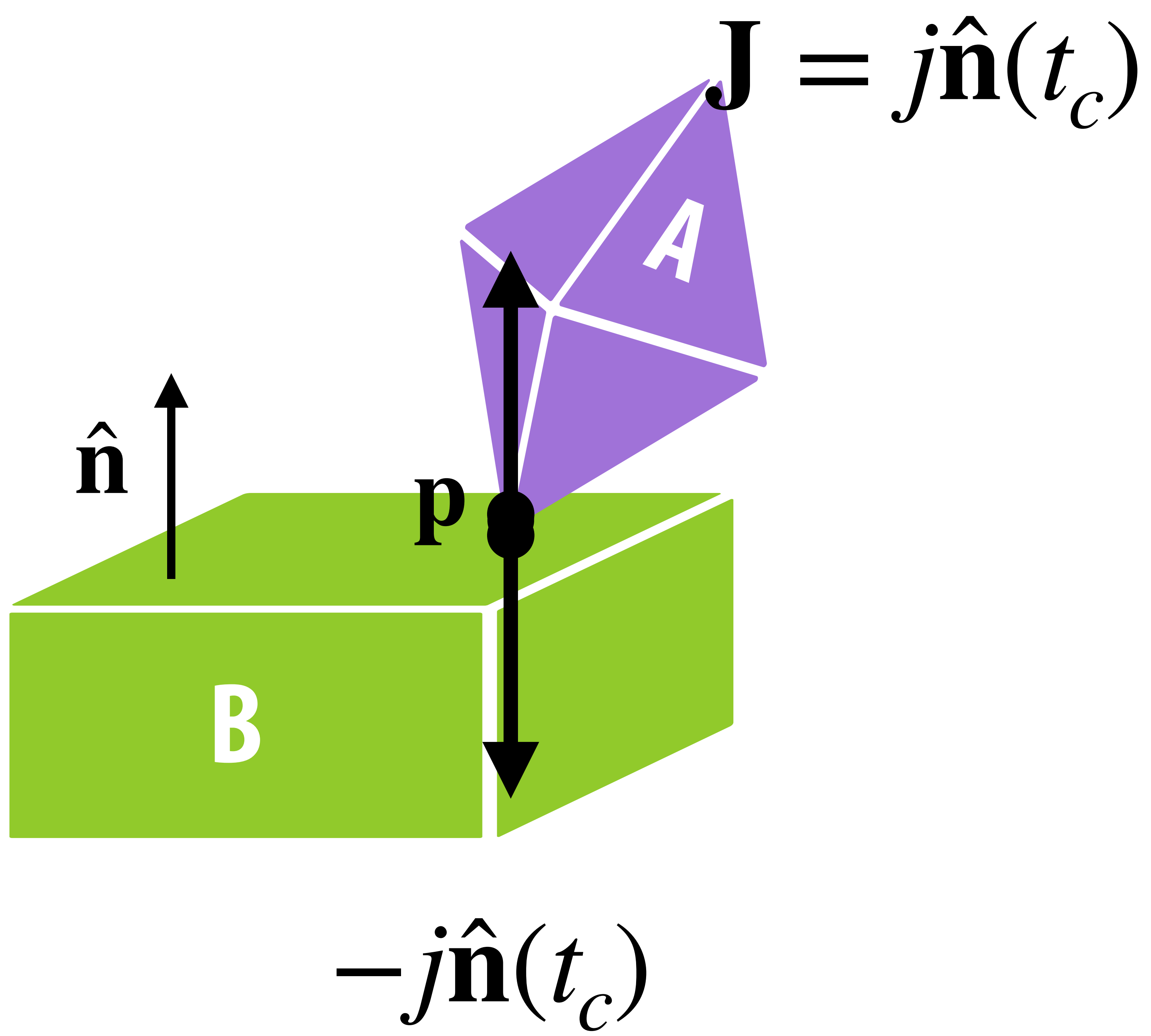
Impulse

- In the rigid body world, we want the velocity to change instantaneously if there is a collision contact.
- Use finite impulse to change velocity instead of infinite force: $\mathbf{J} = \Delta \mathbf{P} = m \Delta \mathbf{v}$
- If the impulse acts on a point $\mathbf{p}$, the impulse produces an impulsive torque
  - $\boldsymbol{\tau}_{imp} = \big(\mathbf{p} - \mathbf{x}(t)\big) \times \mathbf{J}$
  - Impulsive torque results in a change in angular momentum: $\boldsymbol{\tau}_{imp} = \Delta \mathbf{L}$

Unit of J in terms of Newton? N * s

Colliding Contact

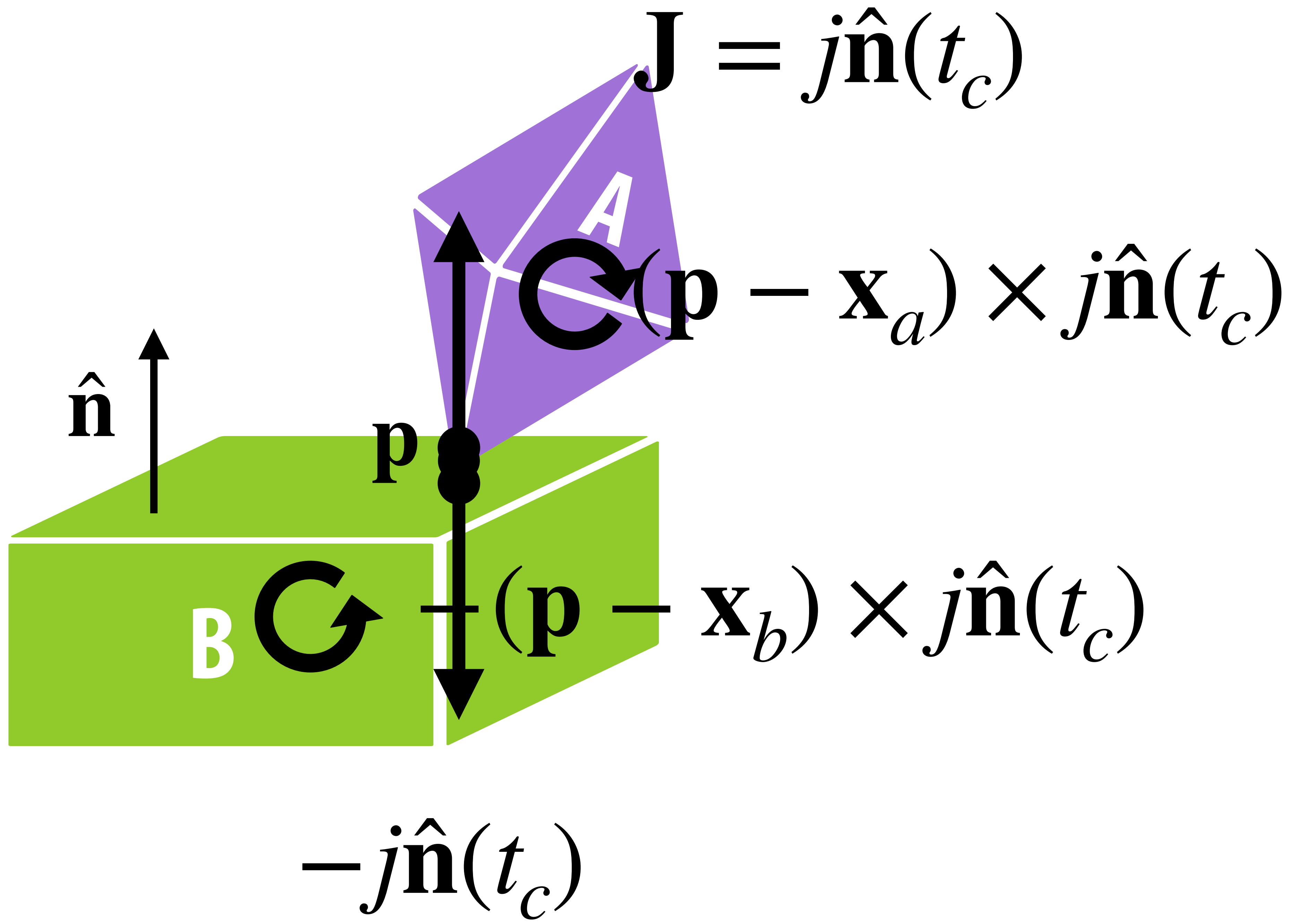
- For frictionless bodies, the direction of the impulse will be in the normal direction $\hat{\mathbf{n}}(t_c)$.



- Once we solve for j, we then can update the linear momentum of the rigid body after the collision.
- Body A is subject to impulse $\mathbf{J}$, while B is subject to an equal but opposite impulse $-\mathbf{J}$

Colliding Contact

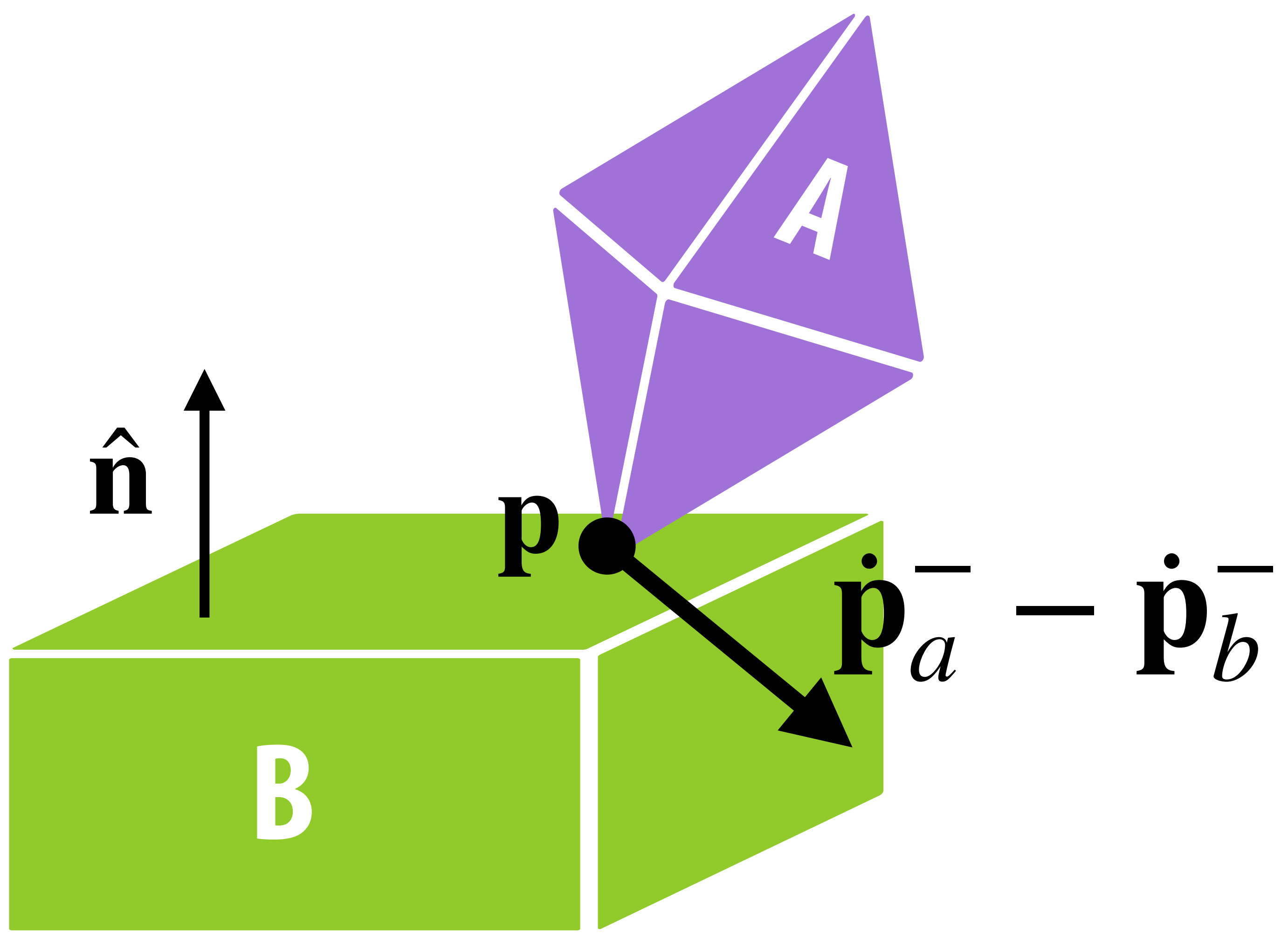
- Similarly, we use impulsive torque to update the angular moment of the rigid bodies



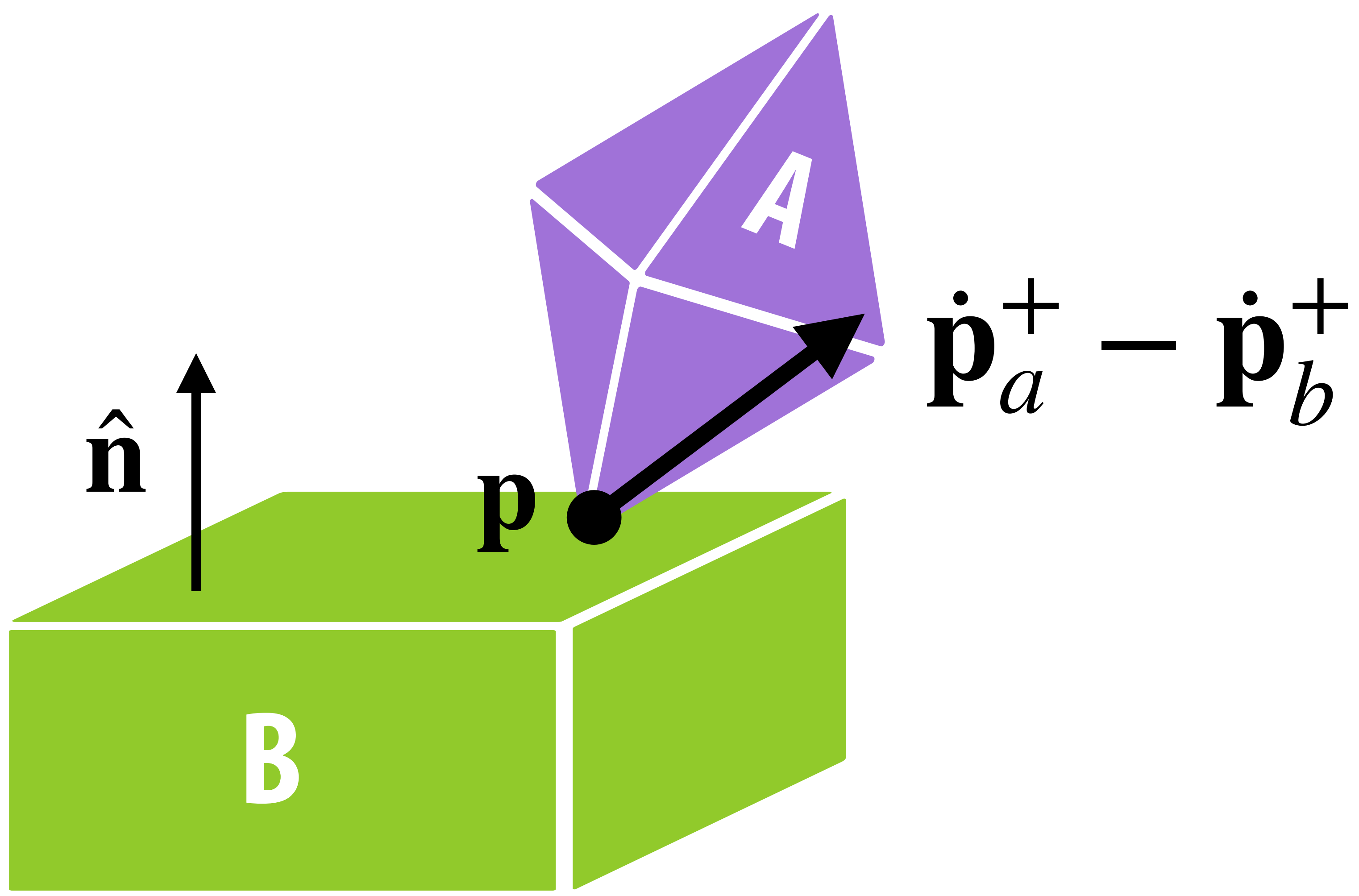
How to solve j?

Recall definition of v_r

before collision



after collision



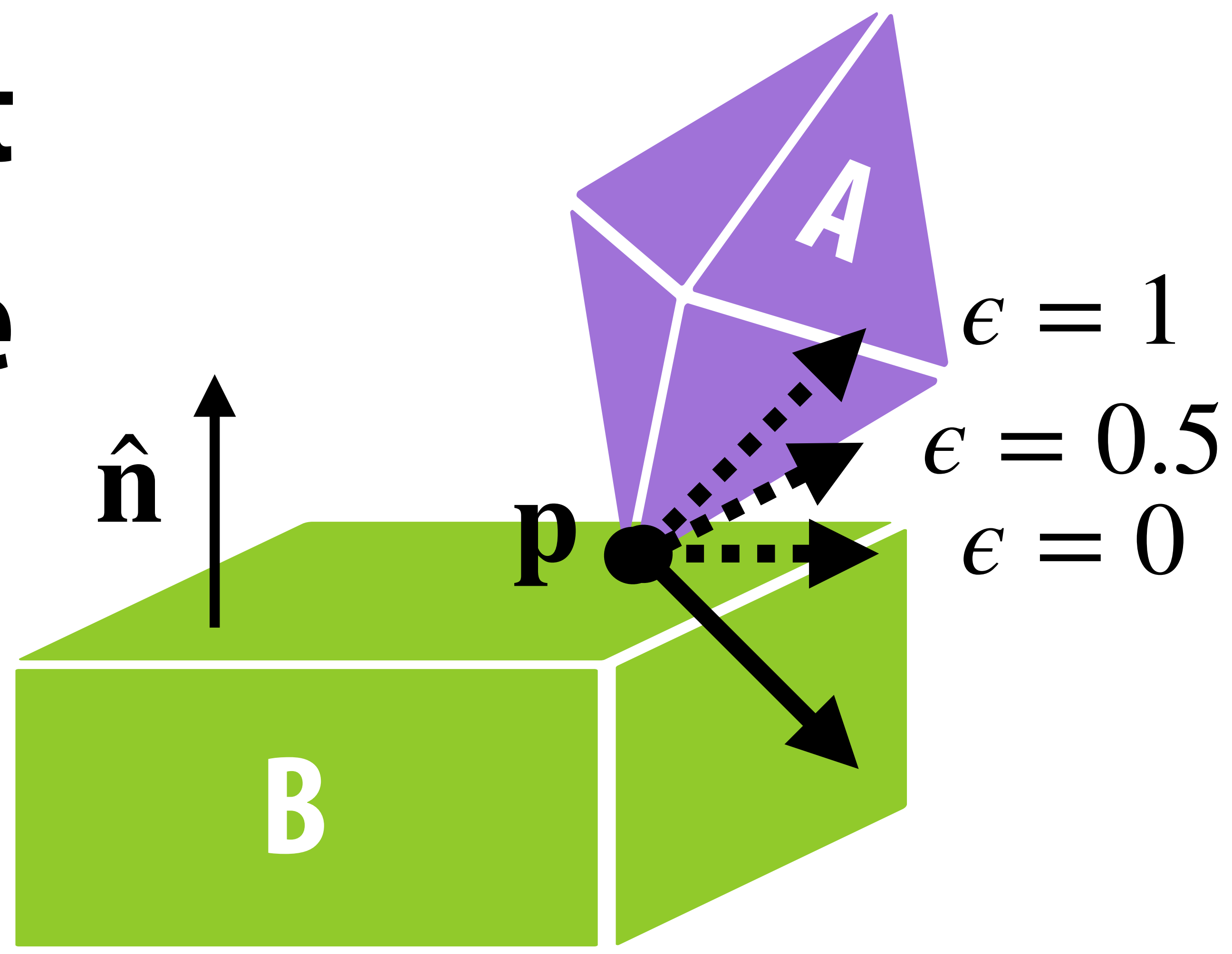
$$v_r^- = \hat{\mathbf{n}}(t_c) \cdot (\dot{\mathbf{p}}_a^- - \dot{\mathbf{p}}_b^-)$$

$$v_r^+ = \hat{\mathbf{n}}(t_c) \cdot (\dot{\mathbf{p}}_a^+ - \dot{\mathbf{p}}_b^+)$$

Scalar!

Colliding Contact

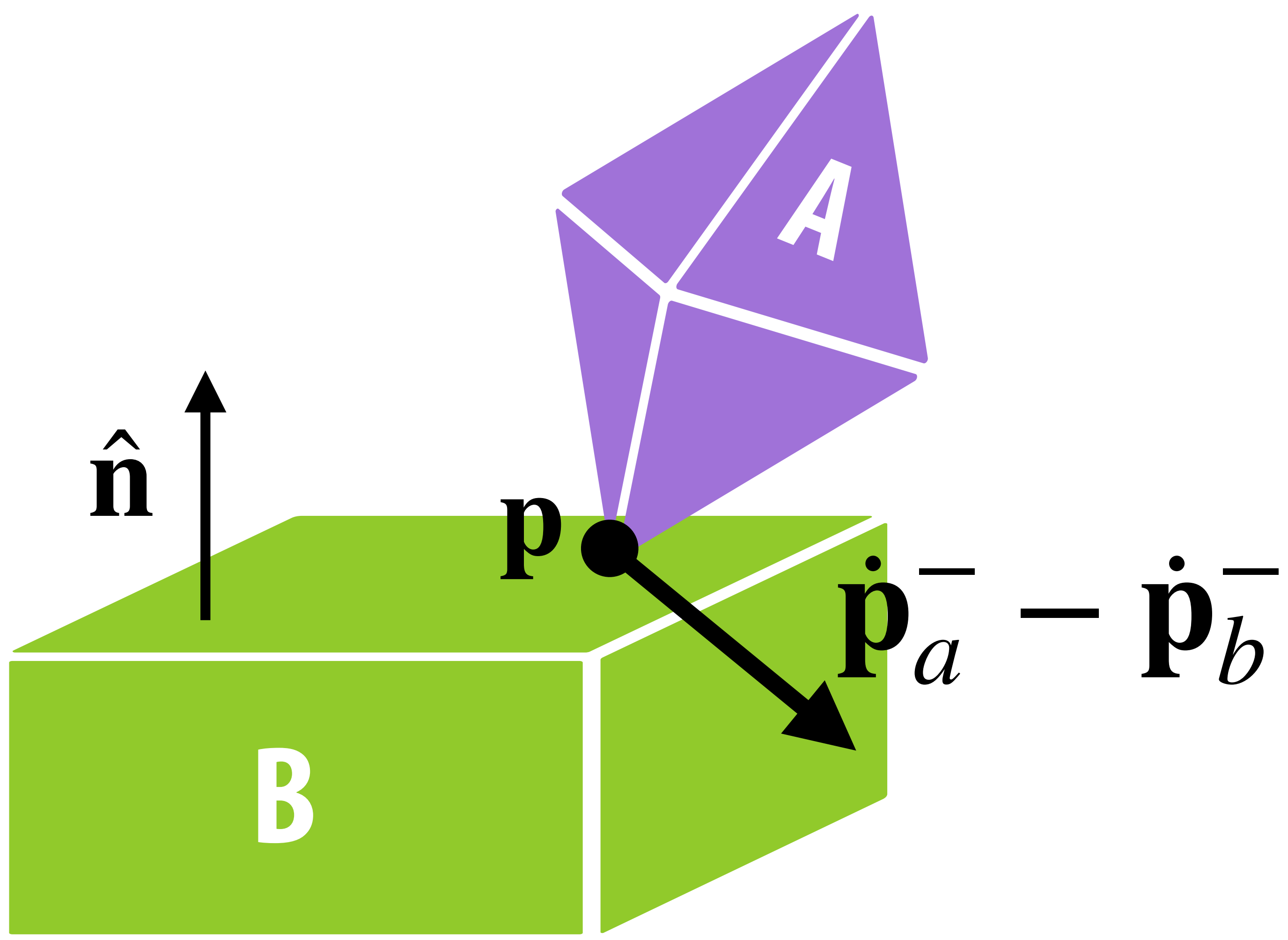
- The change of velocity at the contact point follows the empirical law: $\boxed{v_r^+ = -\epsilon v_r^-}$
- Coefficient of restitution
  - $\epsilon = 0$, resting contact
  - $\epsilon = 1$, perfect bounce



We need to solve for j such that $v_r^+ = -\epsilon v_r^-$

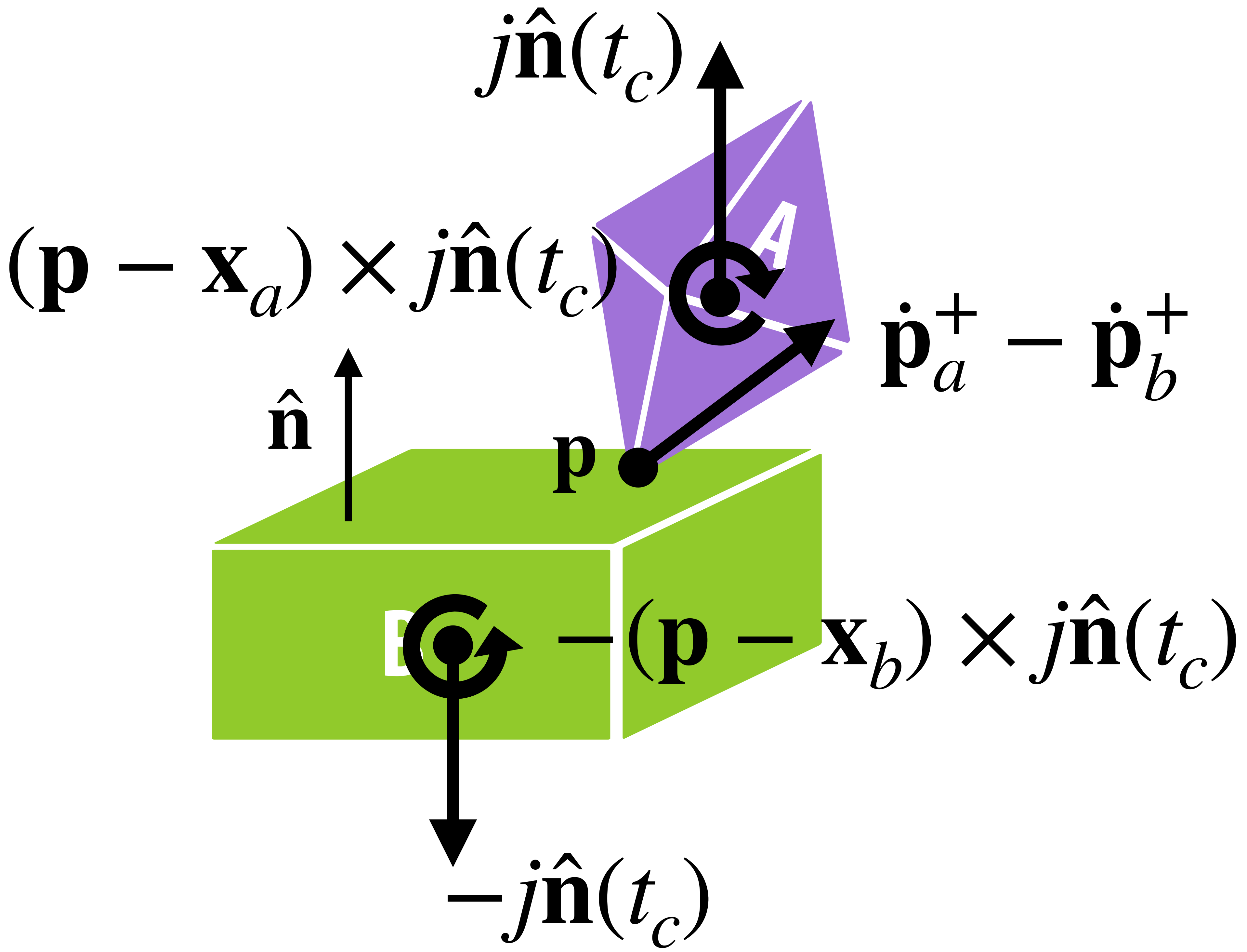
Colliding Contact

before collision



$$v_r^- = \hat{\mathbf{n}}(t_c) \cdot (\dot{\mathbf{p}}_a^- - \dot{\mathbf{p}}_b^-)$$

after collision



$$v_r^+ = \hat{\mathbf{n}}(t_c) \cdot (\dot{\mathbf{p}}_a^+ - \dot{\mathbf{p}}_b^+)$$

Compute the Impulse

- Define the displacement from center of mass
  - $\mathbf{r}_a = \mathbf{p}_a - \mathbf{x}_a$
  - $\mathbf{r}_b = \mathbf{p}_b - \mathbf{x}_b$
- Express contact point velocity in rigid body velocity
  - $\dot{\mathbf{p}}_a^- = \mathbf{v}_a^- + \boldsymbol{\omega}_a^- \times \mathbf{r}_a$, similar for $\dot{\mathbf{p}}_b^-$
  - $\dot{\mathbf{p}}_a^+ = \mathbf{v}_a^+ + \boldsymbol{\omega}_a^+ \times \mathbf{r}_a$, similar for $\dot{\mathbf{p}}_b^+$
- Express post-collision velocity in unknown impulse
  - $\mathbf{v}_a^+ = \boxed{\mathbf{v}_a^- + \frac{j\hat{\mathbf{n}}}{m_a}}$, similar for $\mathbf{v}_b^+$
  - $\boldsymbol{\omega}_a^+ = \boxed{\boldsymbol{\omega}_a^- + \mathbf{I}_a^{-1}(\mathbf{r}_a \times j\hat{\mathbf{n}})}$, similar for $\boldsymbol{\omega}_b^+$

Substitute post-collision rigid body velocity

$$\dot{\mathbf{p}}_a^+ = \boxed{\mathbf{v}_a^-} + \frac{j\hat{\mathbf{n}}}{m_a} + \left(\boxed{\boldsymbol{\omega}_a^-} + \mathbf{I}_a^{-1}(\mathbf{r}_a \times j\hat{\mathbf{n}})\right)\boxed{\times\, \mathbf{r}_a}$$

Recover pre-collision contact velocity, $\dot{\mathbf{p}}_a^-$

$$\dot{\mathbf{p}}_a^+ = \dot{\mathbf{p}}_a^- + j\left(\frac{\hat{\mathbf{n}}}{m_a} + \left(\mathbf{I}_a^{-1}(\mathbf{r}_a \times \hat{\mathbf{n}})\right) \times \mathbf{r}_a\right)$$

Compute the Impulse

- **Express the empirical law in contact velocity**

$$v_r^+ = -\epsilon v_r^-$$

$$\boxed{\dot{\mathbf{p}}_a^+ = \dot{\mathbf{p}}_a^- + j\left(\frac{\hat{\mathbf{n}}}{m_a} + \left(\mathbf{I}_a^{-1}(\mathbf{r}_a \times \hat{\mathbf{n}})\right) \times \mathbf{r}_a\right)}$$

$$v_r^+ = \hat{\mathbf{n}} \cdot (\dot{\mathbf{p}}_a^+ - \dot{\mathbf{p}}_b^+)$$

$$= \boxed{\hat{\mathbf{n}} \cdot (\dot{\mathbf{p}}_a^- - \dot{\mathbf{p}}_b^-)} + j(\frac{1}{m_a} + \frac{1}{m_b} + \hat{\mathbf{n}} \cdot ((\mathbf{I}_a^{-1}(\mathbf{r}_a \times \hat{\mathbf{n}})) \times \mathbf{r}_a) + \hat{\mathbf{n}} \cdot ((\mathbf{I}_b^{-1}(\mathbf{r}_b \times \hat{\mathbf{n}})) \times \mathbf{r}_b))$$

$$\hat{\mathbf{n}} \cdot \hat{\mathbf{n}} = 1$$

$$= v_r^- + j(\frac{1}{m_a} + \frac{1}{m_b} + \hat{\mathbf{n}} \cdot ((\mathbf{I}_a^{-1}(\mathbf{r}_a \times \hat{\mathbf{n}})) \times \mathbf{r}_a) + \hat{\mathbf{n}} \cdot ((\mathbf{I}_b^{-1}(\mathbf{r}_b \times \hat{\mathbf{n}})) \times \mathbf{r}_b))$$

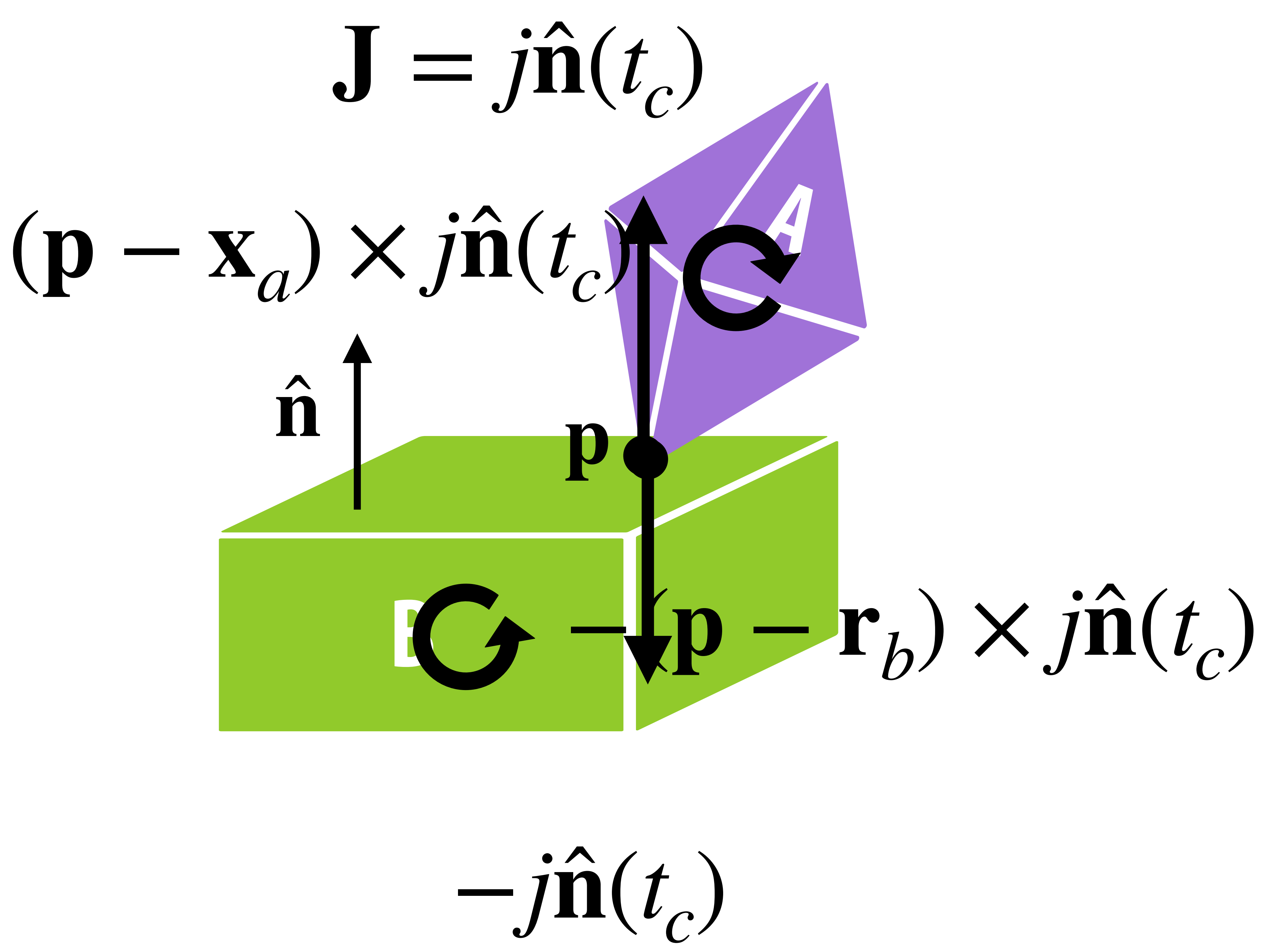
$$-\epsilon v_r^- = v_r^- + j(\frac{1}{m_a} + \frac{1}{m_b} + \hat{\mathbf{n}} \cdot ((\mathbf{I}_a^{-1}(\mathbf{r}_a \times \hat{\mathbf{n}})) \times \mathbf{r}_a) + \hat{\mathbf{n}} \cdot ((\mathbf{I}_b^{-1}(\mathbf{r}_b \times \hat{\mathbf{n}})) \times \mathbf{r}_b))$$

$$\boxed{j = \frac{-(1+\epsilon)v_r^-}{\frac{1}{m_a} + \frac{1}{m_b} + \hat{\mathbf{n}} \cdot ((\mathbf{I}_a^{-1}(\mathbf{r}_a \times \hat{\mathbf{n}})) \times \mathbf{r}_a) + \hat{\mathbf{n}} \cdot ((\mathbf{I}_b^{-1}(\mathbf{r}_b \times \hat{\mathbf{n}})) \times \mathbf{r}_b)}}$$

Colliding Contact

- Apply change in momentum to current state:
  - Body A:
    - $\mathbf{P}(t_c + h) = \mathbf{P}(t_c) + \mathbf{J}$
    - $\mathbf{L}(t_c + h) = \mathbf{L}(t_c) + (\mathbf{p} - \mathbf{x}_a) \times \mathbf{J}$
  - Body B:
    - $\mathbf{P}(t_c + h) = \mathbf{P}(t_c) - \mathbf{J}$
    - $\mathbf{L}(t_c + h) = \mathbf{L}(t_c) + (\mathbf{p} - \mathbf{x}_b) \times (-\mathbf{J})$
- Solve one by one and iteratively until all colliding contact resolved, similar to Project 2

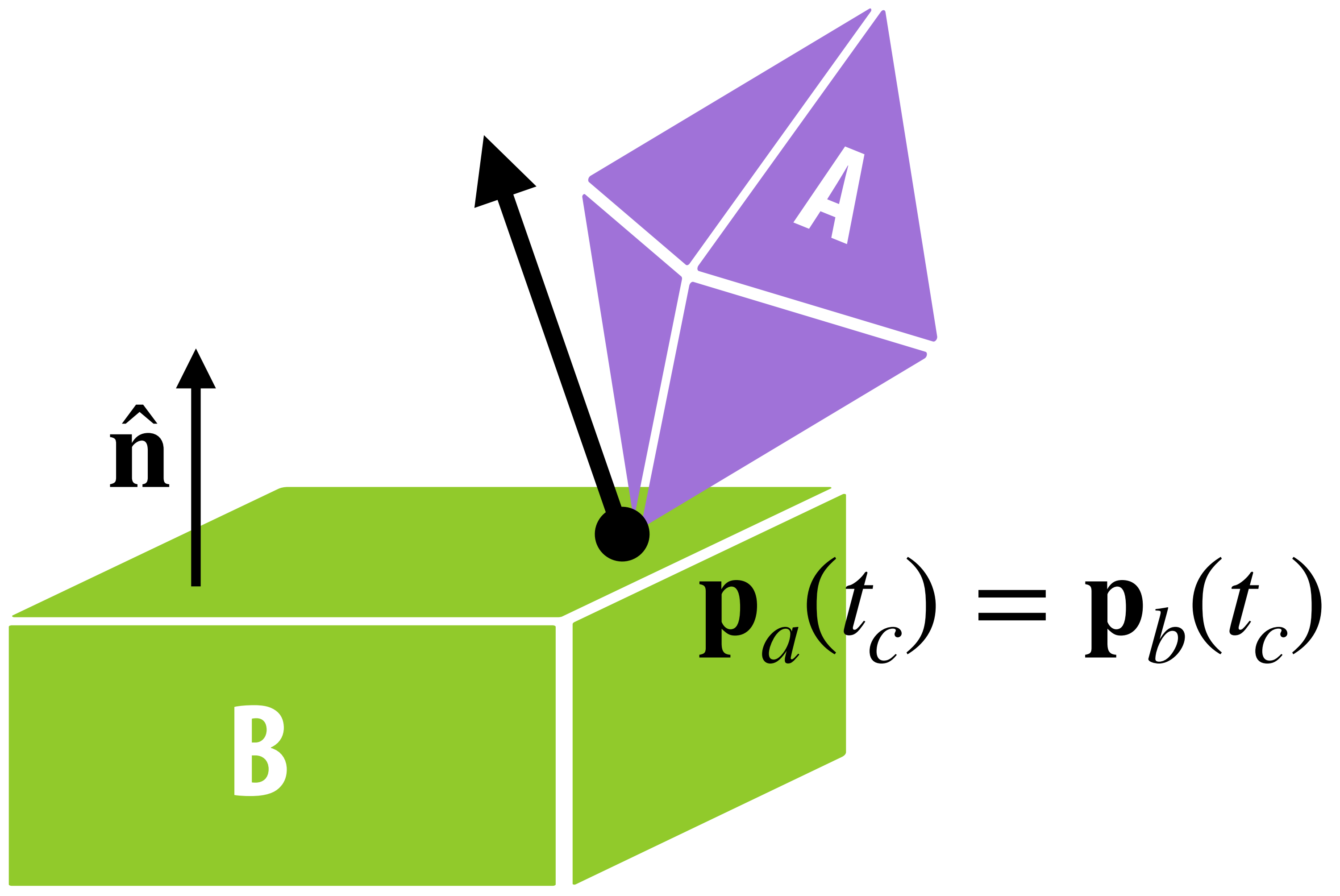
after collision



Relative Normal Velocity

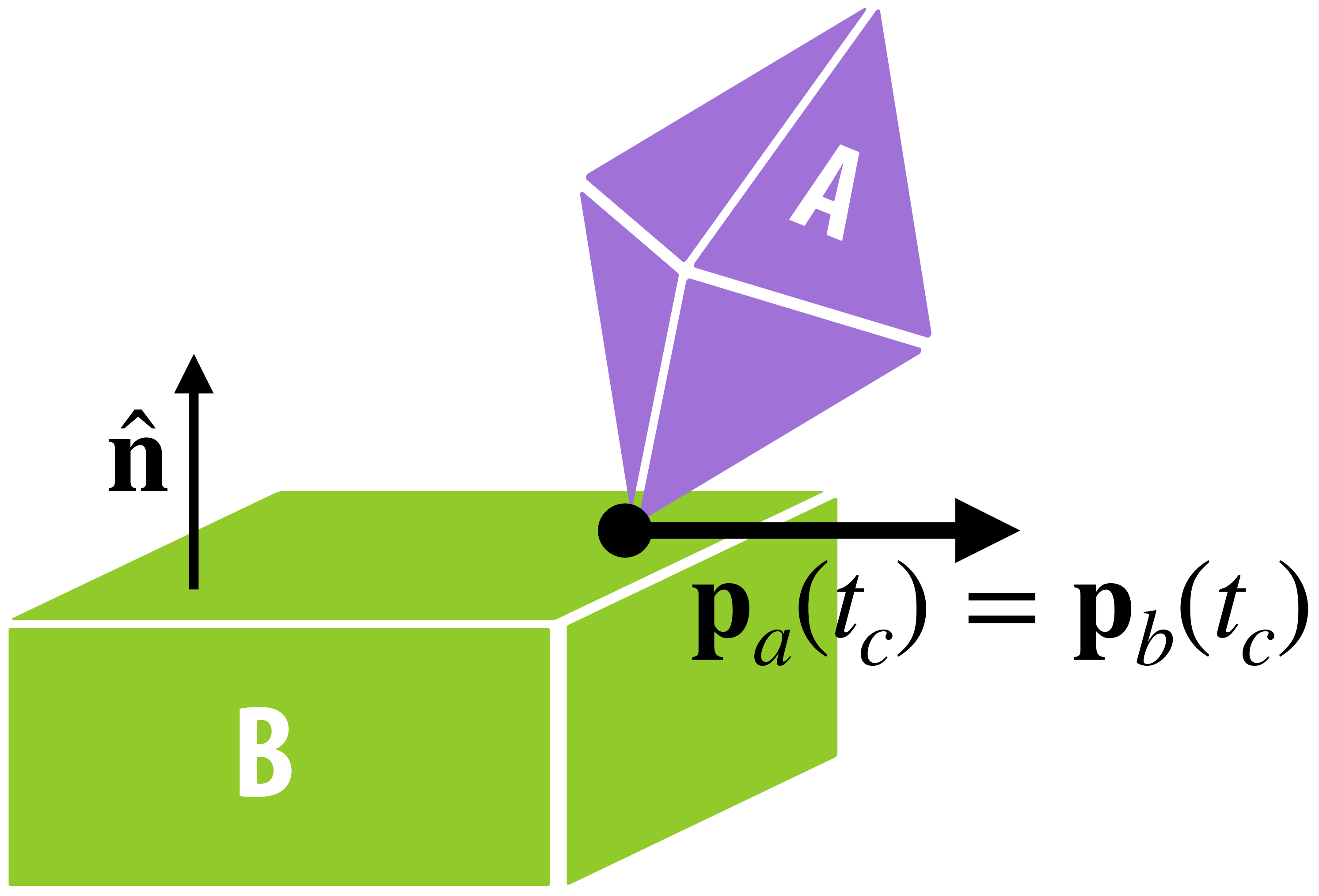
$v_r > 0$

separation



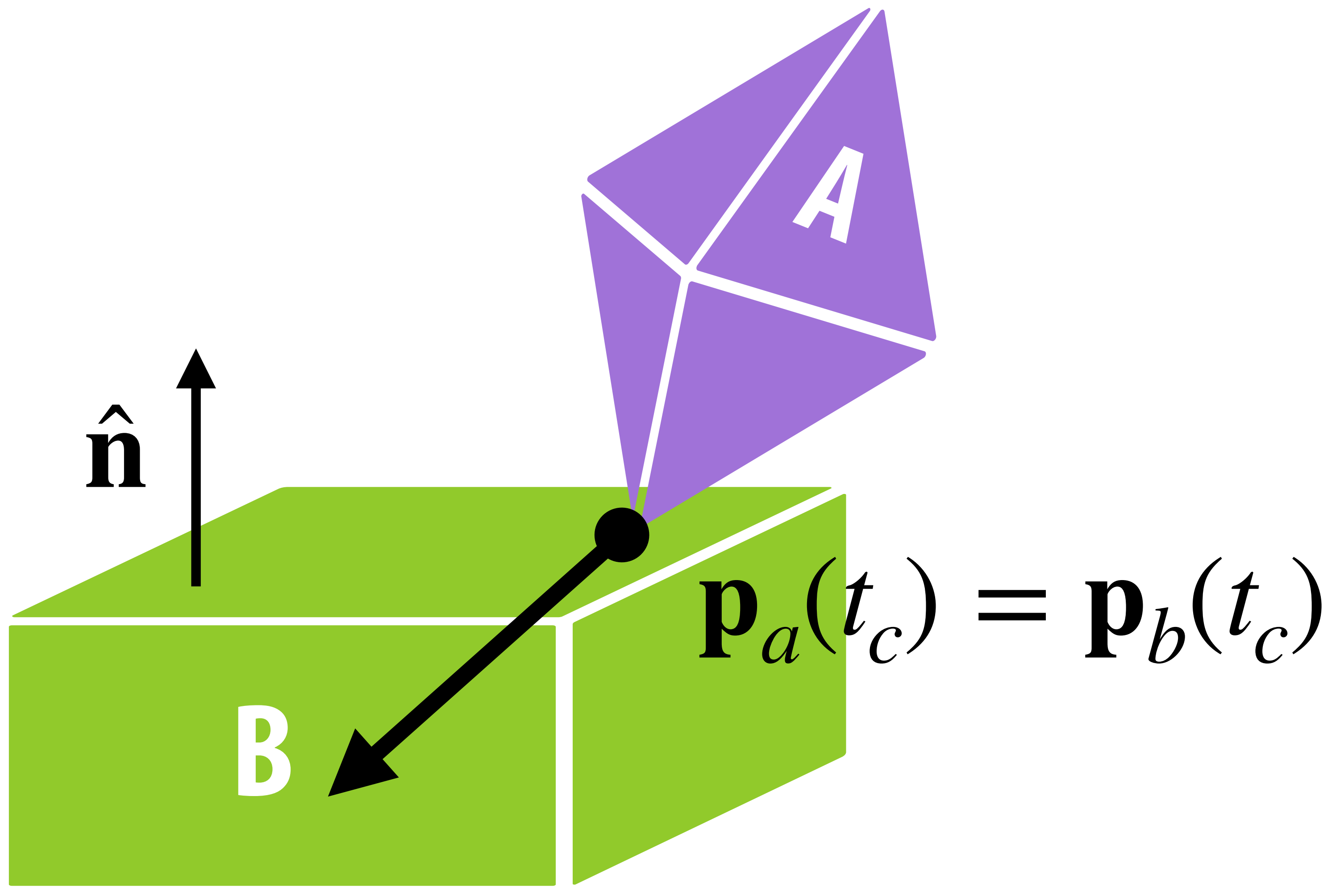
$v_r = 0$

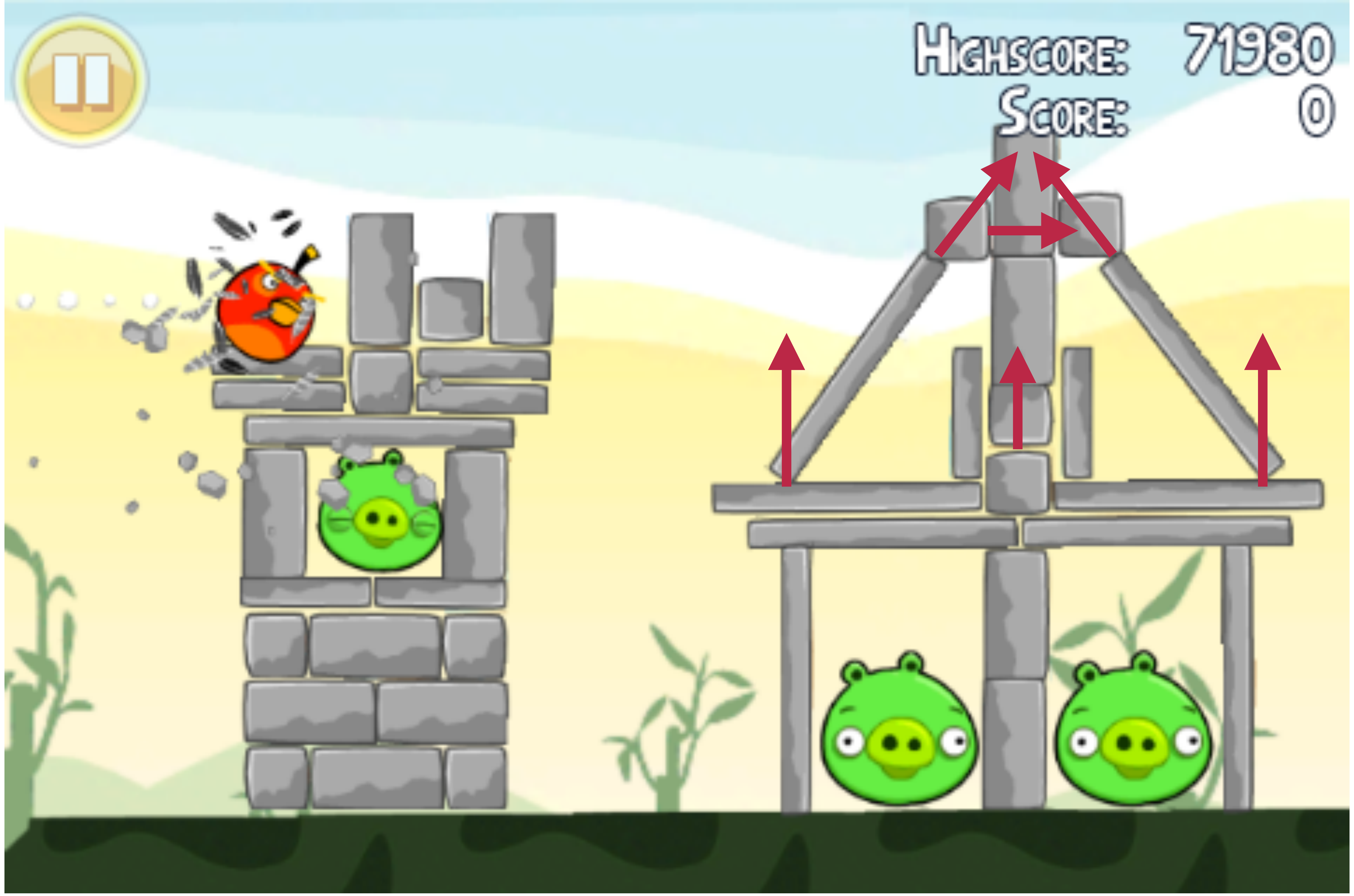
resting contact



$v_r < 0$

colliding contact



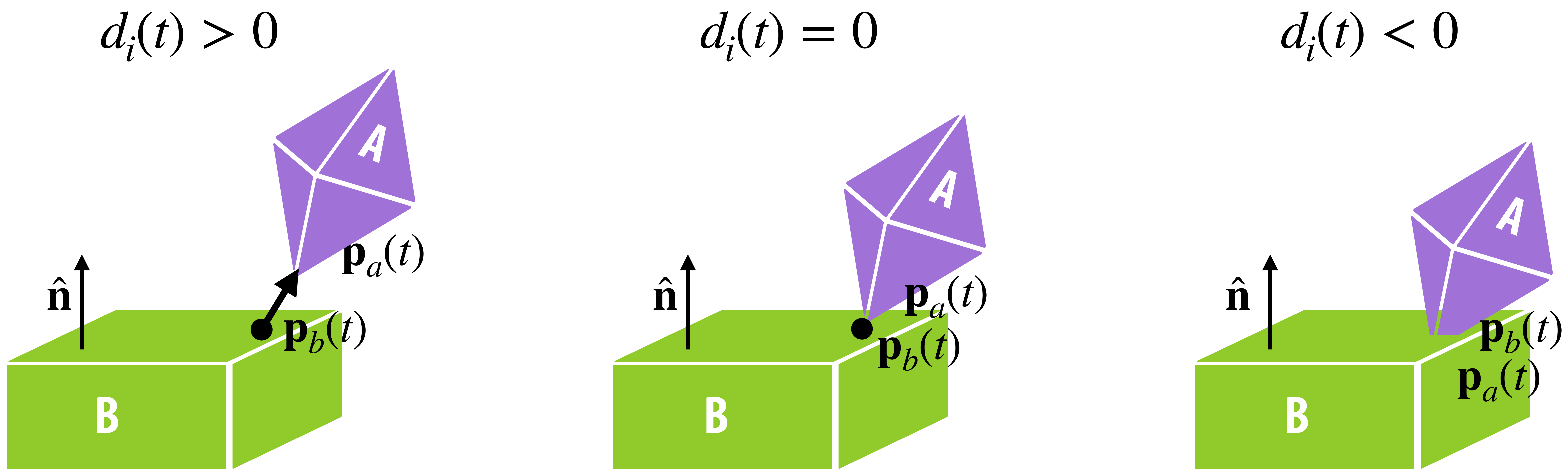

HIGHSCORE: 71980
SCORE: 0

Resting Contact

- **In this case, all n contact points have the zero relative velocity**
- **At each contact point there is some force $f_i\hat{\mathbf{n}}_i$, where f_i is an unknown scalar and $\hat{\mathbf{n}}_i$ is a defined normal at that contact point**
- **Our goal is to determine what each f_i is by solving all of them simultaneously**
- **What are the conditions for f_i?**

Non-penetration

- Let's define penetration:
  - $d_i = \hat{\mathbf{n}} \cdot (\mathbf{p}_a - \mathbf{p}_b)$
- We want to avoid $d_i < 0$

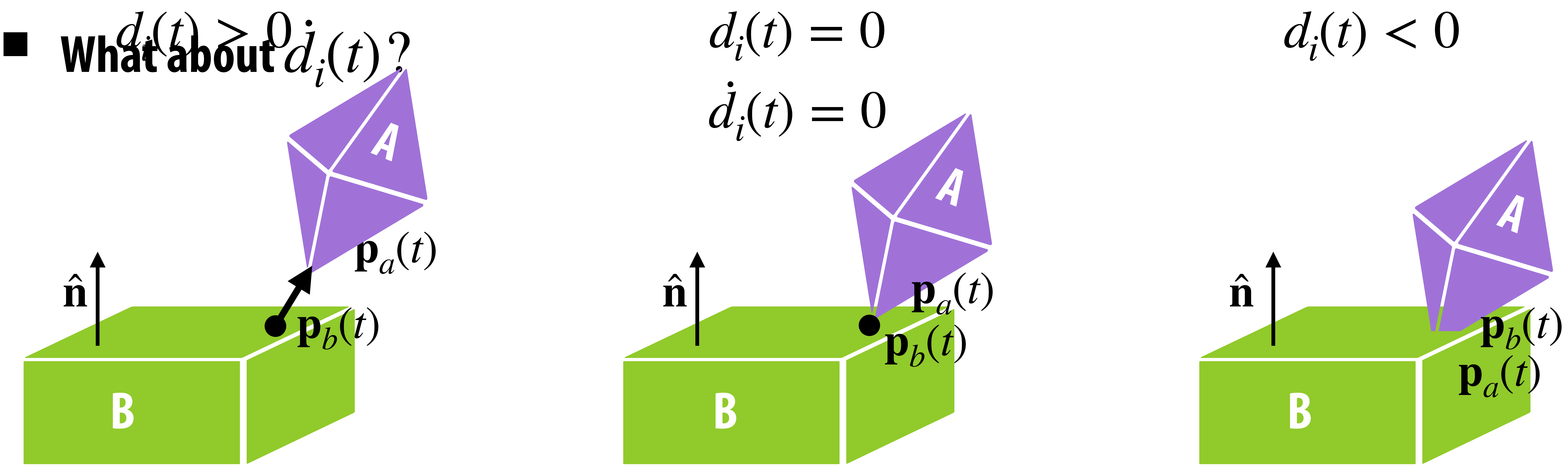


Non-penetration

$$\dot{d}_i(t) = \cancel{\dot{\hat{\mathbf{n}}}_i(t) \cdot \big(\mathbf{p}_a(t) - \mathbf{p}_b(t)\big)} + \hat{\mathbf{n}}_i(t) \cdot \big(\dot{\mathbf{p}}_a(t) - \dot{\mathbf{p}}_b(t)\big)$$

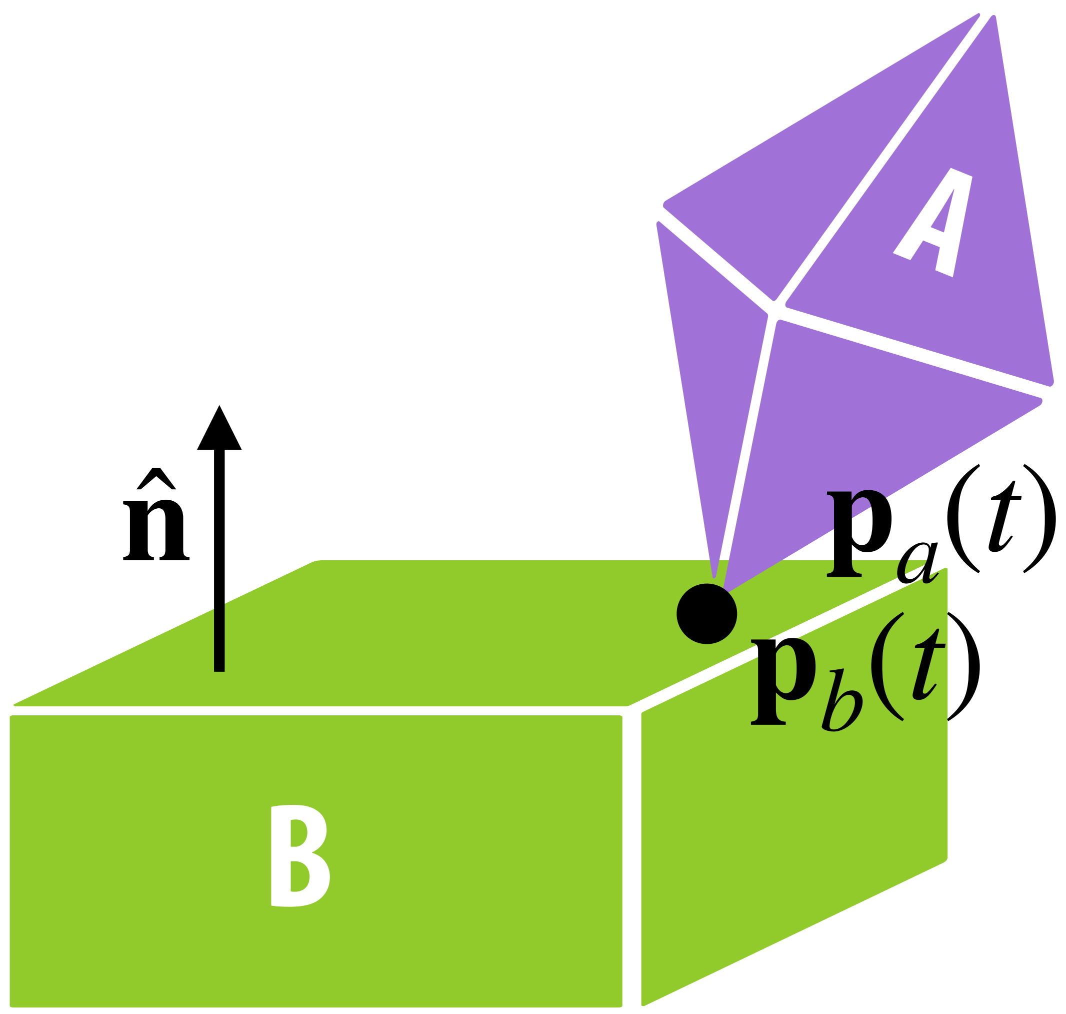
$\dot{d}_i(t) = v_r = 0$ **because it is a resting contact**

- **Let's define penetration:**
  - $d_i = \hat{\mathbf{n}} \cdot (\mathbf{p}_a - \mathbf{p}_b)$
- **We want to avoid** $d_i < 0$
- **Since collision is detected,** $d_i(t) = 0$
- **What about** $\dot{d}_i(t)$?



Non-penetration

- Let's define penetration:
  - $d_i = \hat{\mathbf{n}} \cdot (\mathbf{p}_a - \mathbf{p}_b)$
- We want to avoid $d_i < 0$
- At rest contact, $d_i(t) = 0$ and $\dot{d}_i(t) = 0$
- If $\ddot{d}(t) < 0$, bodies have an acceleration toward each other and the penetration will occur.

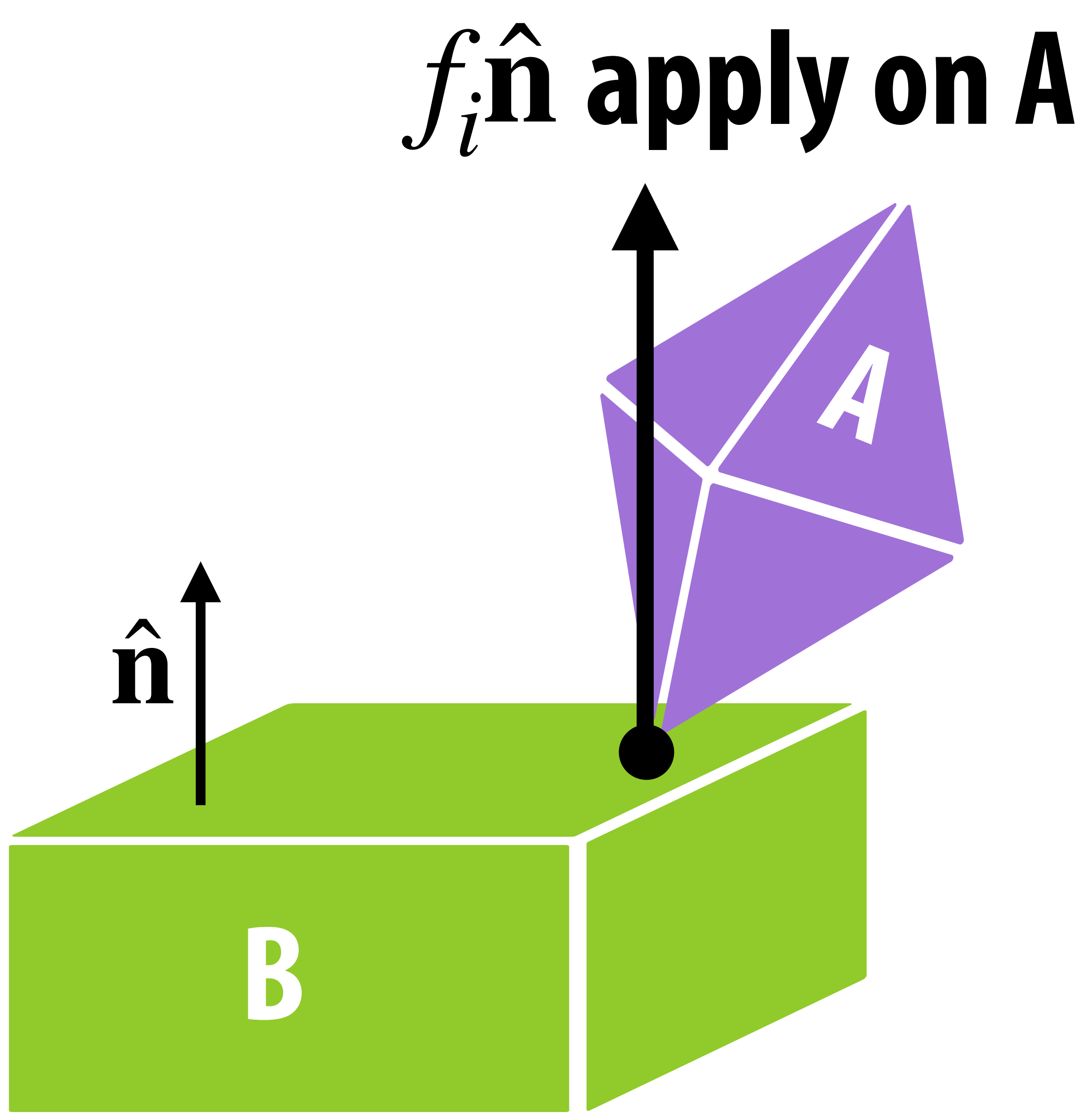


Non-penetration

- Let's define penetration:
  - $d_i = \hat{\mathbf{n}} \cdot (\mathbf{p}_a - \mathbf{p}_b)$
- We want to avoid $d_i < 0$
- At rest contact, $d_i(t) = 0$ and $\dot{d}_i(t) = 0$
- If $\ddot{d}(t) < 0$, bodies have an acceleration toward each other and the penetration will occur.
- Therefore, the first condition is $\boxed{\ddot{d}(t) \geq 0}$

Repulsive force

- The contact forces can push bodies apart, but can never act like "glue" and hold bodies together.
- Therefore, each contact force must act outward: $\boxed{f_i \geq 0}$



Workless force

- The contact force at the a contact point becomes zero if the bodies begin to separate.
- If contact is breaking, that is, $\ddot{d}_i(t) > 0$, then f_i should be zero.
- If f_i is not zero, then the contact is not breaking, that is, $\ddot{d}_i(t) = 0$.
- What is the equation that satisfies these two conditions?

$$\boxed{f_i \ddot{d}_i(t) = 0}$$

Compute contact forces

- **Non- penetration**

$$\ddot{d}_i(t) \geq 0$$

- **Repulsive force**

$$f_i \geq 0$$

- **Workless force**

$$f_i \ddot{d}_i(t) = 0$$

Express $\ddot{d}$'s in terms of f's:

$$\ddot{d}_i = \hat{\mathbf{n}} \cdot (\ddot{\mathbf{p}}_a - \ddot{\mathbf{p}}_b) + 2\dot{\hat{\mathbf{n}}} \cdot (\dot{\mathbf{p}}_a - \dot{\mathbf{p}}_b)$$
$$= a_{i1}f_1 + a_{i2}f_2 + \cdots + a_{in}f_n + b_i$$

Factor out the terms that depend on f_j and assign them to a_{ij}

Assign the rest of terms to b_i

Collect all the a_{ij} to form matrix $\mathbf{A}$ and all the b_i to form vector $\mathbf{b}$

$\ddot{\mathbf{d}} = \mathbf{A}\mathbf{f} + \mathbf{b}$, where $\ddot{\mathbf{d}} = [\ddot{d}_1, \cdots \ddot{d}_n]$ and $\mathbf{f} = f_1, \cdots, f_n]$

See details in Baraff and Witkin's course notes

Linear complementarity program (LCP)

- **Solve for** $\mathbf{f} = [f_i, f_2, \cdots, f_n]$

- **Subject to**

$$\mathbf{Af} + \mathbf{b} \geq 0$$

$$\mathbf{f} \geq \mathbf{0}$$

$$(\mathbf{Af} + \mathbf{b})^T \mathbf{f} = 0$$

Can solve it as a Quadratic Program (for some A)

Velocity-based LCP

- In practice, for physics engines used in industry…
- Unified treatment of colliding & resting contacts, through one LCP problem
- Instead of solving force and acceleration for resting contact, solving Velocity-based LCP for impulse and momentum change, as in colliding

Friction

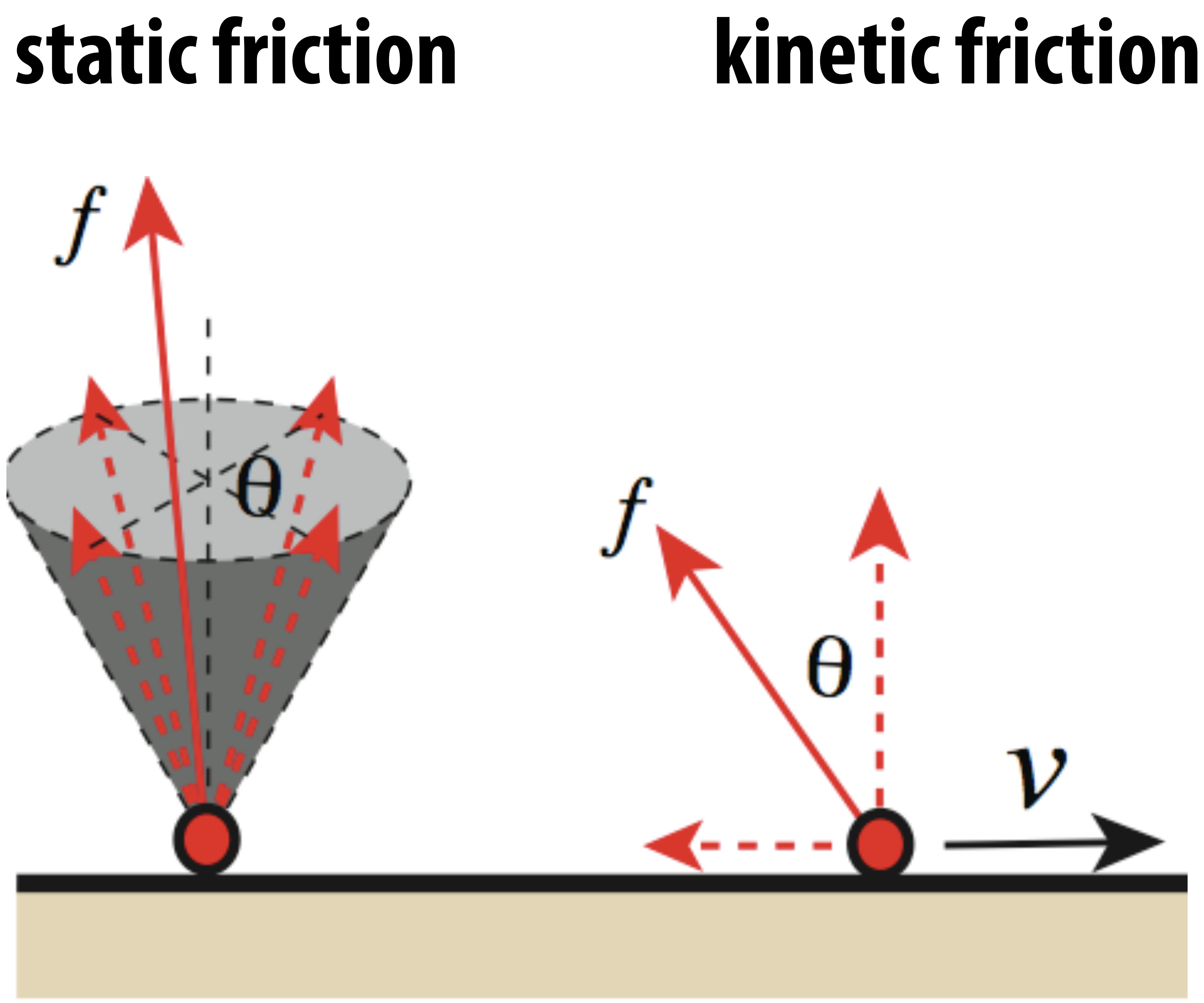
- **Coulomb's Law of Friction**
  - If sliding, the kinetic friction is

$$\mathbf{f}_{\|} = -\mu_k |\mathbf{f}_{\perp}| \frac{\mathbf{v}_{\|}}{|\mathbf{v}_{\|}|}$$

  - If static, stay static as long as

$$|\mathbf{f}_{\|}| \leq \mu_s |\mathbf{f}_{\perp}|$$

- **These conditions can be merged into LCP as well**



$$\theta = \tan^{-1} \mu_s$$

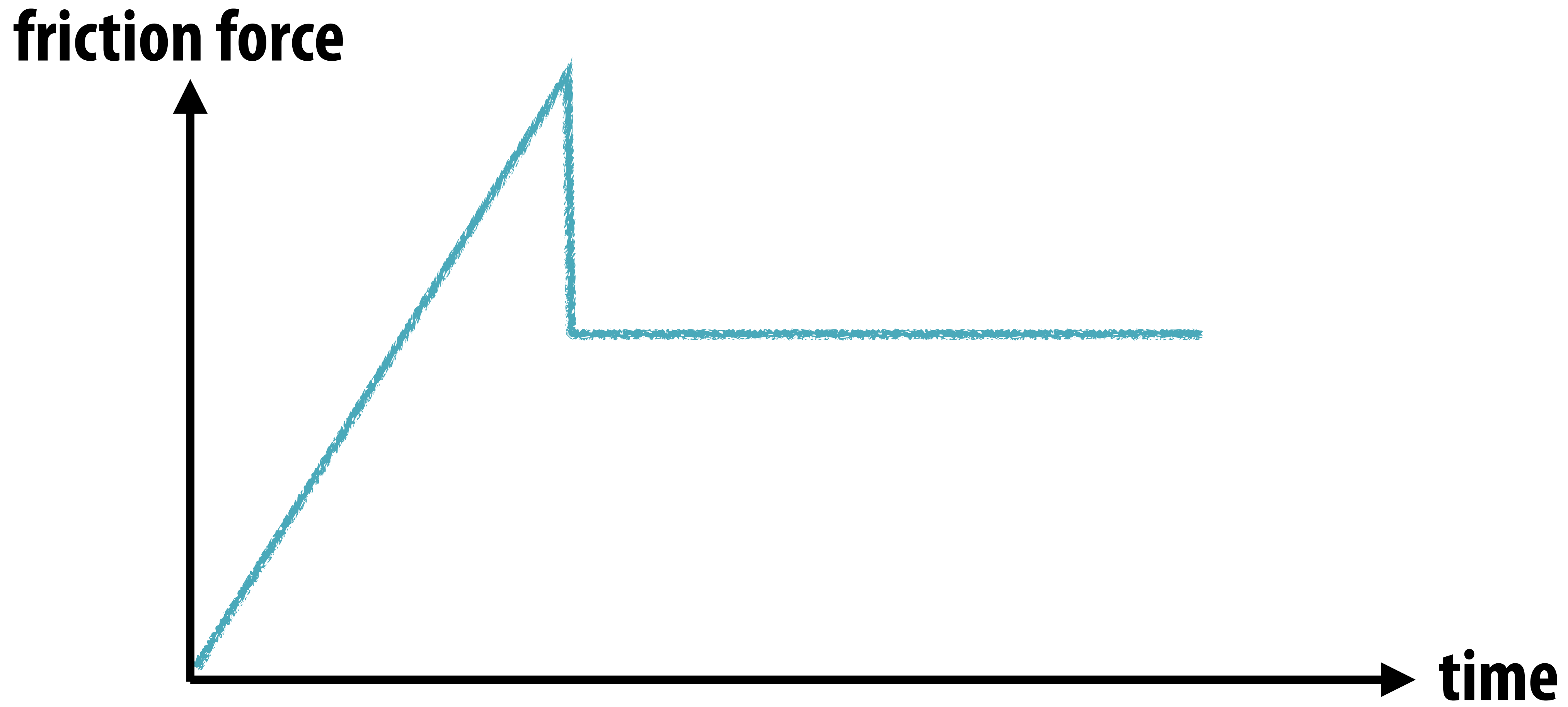
Friction coefficient

Materials		Static Friction, μ_s		Kinetic/Sliding Friction, μ_k	
		Dry and clean	Lubricated	Dry and clean	Lubricated
Aluminium	Steel	0.61[25]		0.47[25]	
Aluminium	Aluminium	1.05–1.35[25]	0.3[25]	1.4[25]–1.5[26]	
Gold	Gold			2.5[26]	
Platinum	Platinum	1.2[25]	0.25[25]	3.0[26]	
Silver	Silver	1.4[25]	0.55[25]	1.5[26]	
Alumina ceramic	Silicon nitride ceramic				0.004 (wet)[27]
BAM (Ceramic alloy $AlMgB_{14}$)	Titanium boride (TiB_2)	0.04–0.05[28]	0.02[29][30]		
Brass	Steel	0.35–0.51[25]	0.19[25]	0.44[25]	
Cast iron	Copper	1.05[25]		0.29[25]	
Cast iron	Zinc	0.85[25]		0.21[25]	
Concrete	Rubber	1.0	0.30 (wet)	0.6–0.85[25]	0.45–0.75 (wet)[25]
Concrete	Wood	0.62[25][31]			
Copper	Glass	0.68[32]		0.53[32]	
Copper	Steel	0.53[32]		0.36[25][32]	0.18[32]
Glass	Glass	0.9–1.0[25][32]	0.005–0.01[32]	0.4[25][32]	0.09–0.116[32]
Human synovial fluid	Human cartilage		0.01[33]		0.003[33]
Ice	Ice	0.02–0.09[34]			
Polyethene	Steel	0.2[25][34]	0.2[25][34]		
PTFE (Teflon)	PTFE (Teflon)	0.04[25][34]	0.04[25][34]		0.04[25]
Steel	Ice	0.03[34]			
Steel	PTFE (Teflon)	0.04[25]–0.2[34]	0.04[25]		0.04[25]
Steel	Steel	0.74[25]–0.80[34]	0.005–0.23[32][34]	0.42–0.62[25][32]	0.029–0.19[32]
Wood	Metal	0.2–0.6[25][31]	0.2 (wet)[25][31]	0.49[32]	0.075[32]
Wood	Wood	0.25–0.62[25][31][32]	0.2 (wet)[25][31]	0.32–0.48[32]	0.067–0.167[32]

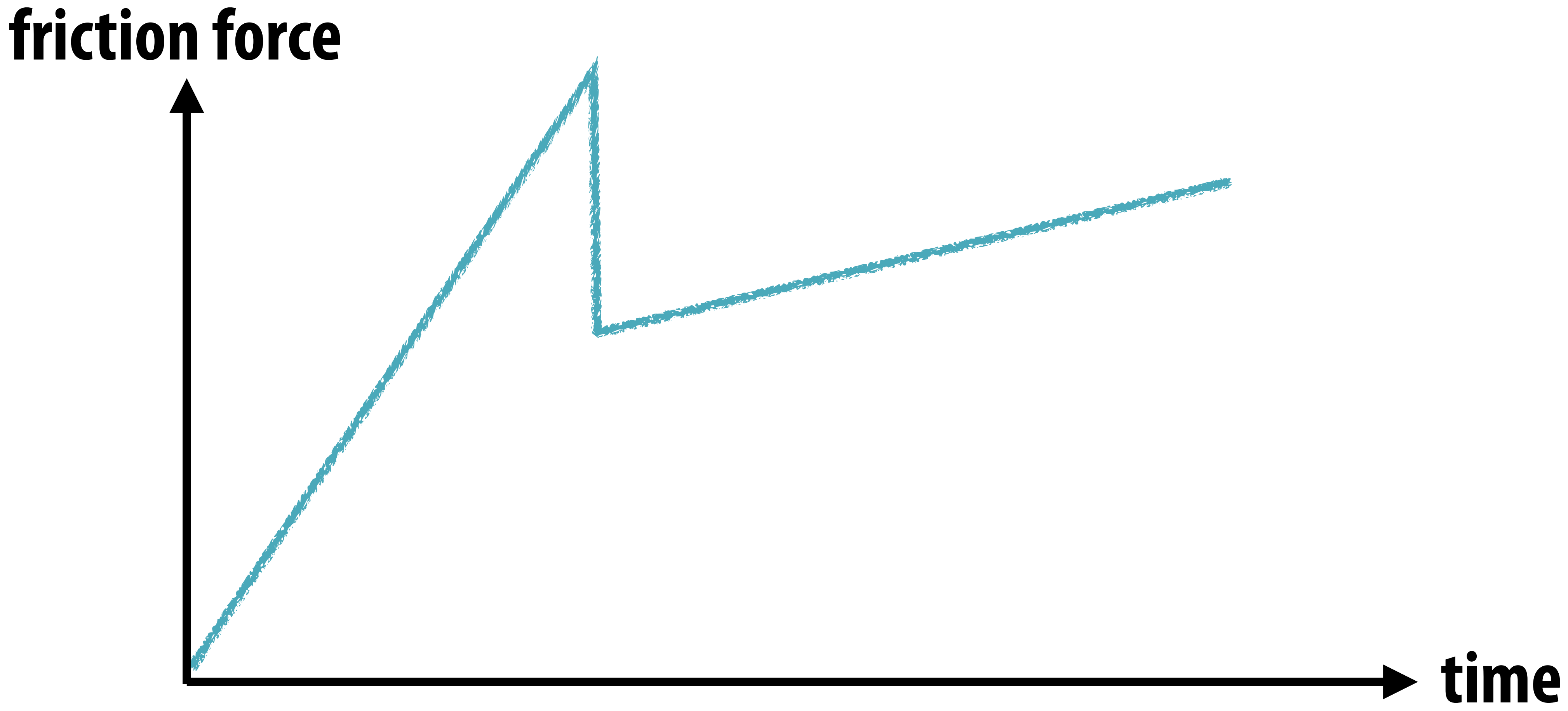
Quiz

- A block is pushed by an increasing horizontal force. The friction force overtime looks like:

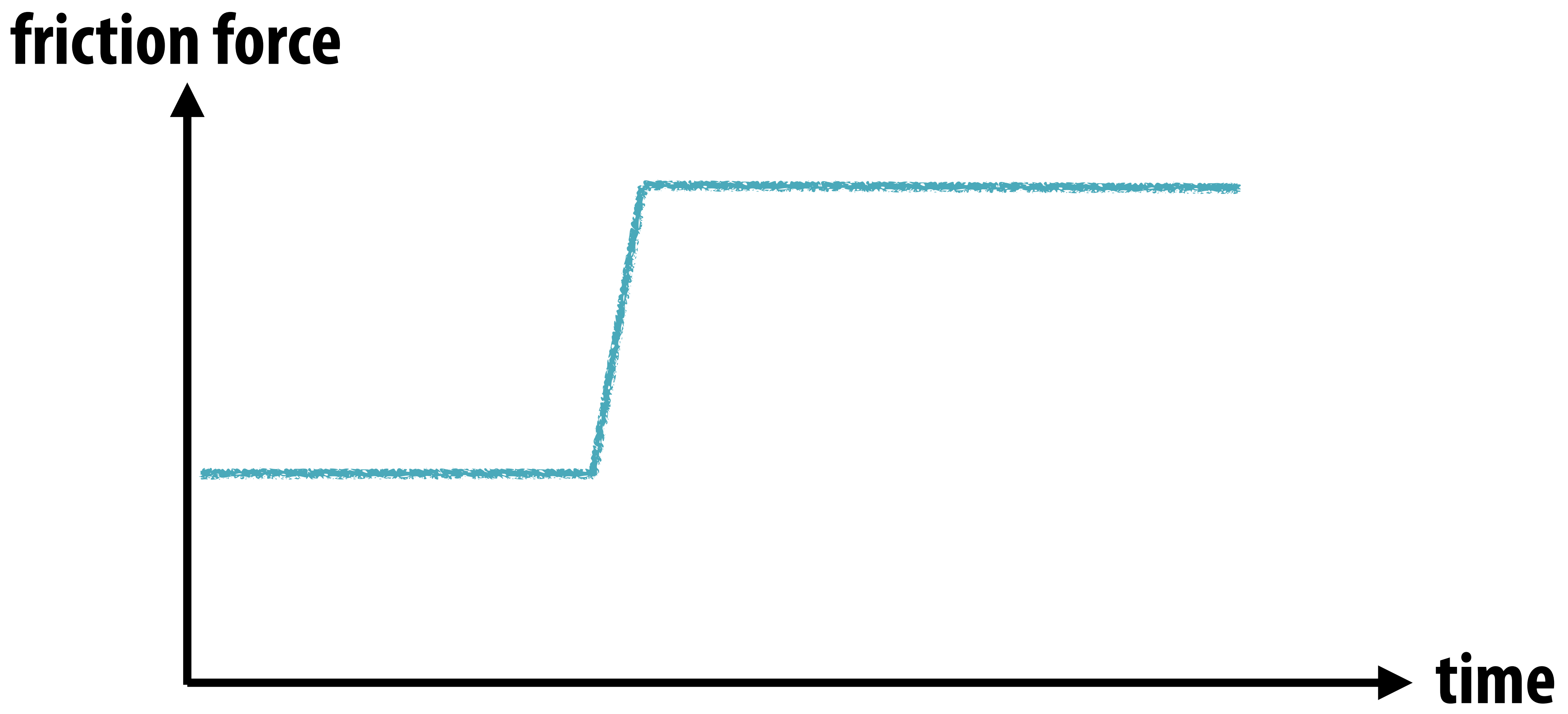
(a)



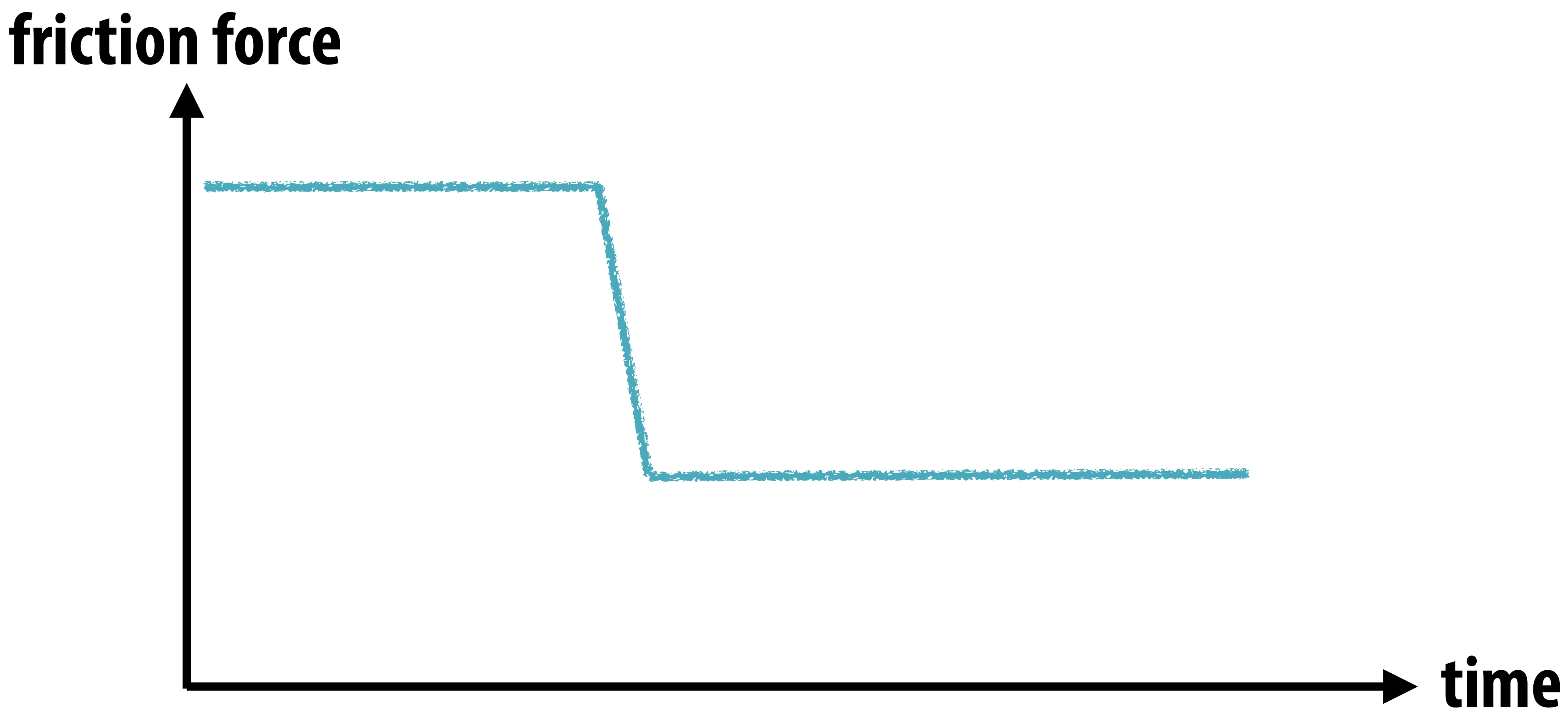
(b)



(c)



(d)



Recitation Session for HW3 Coding Question

aka “things are so much simpler in 2D”

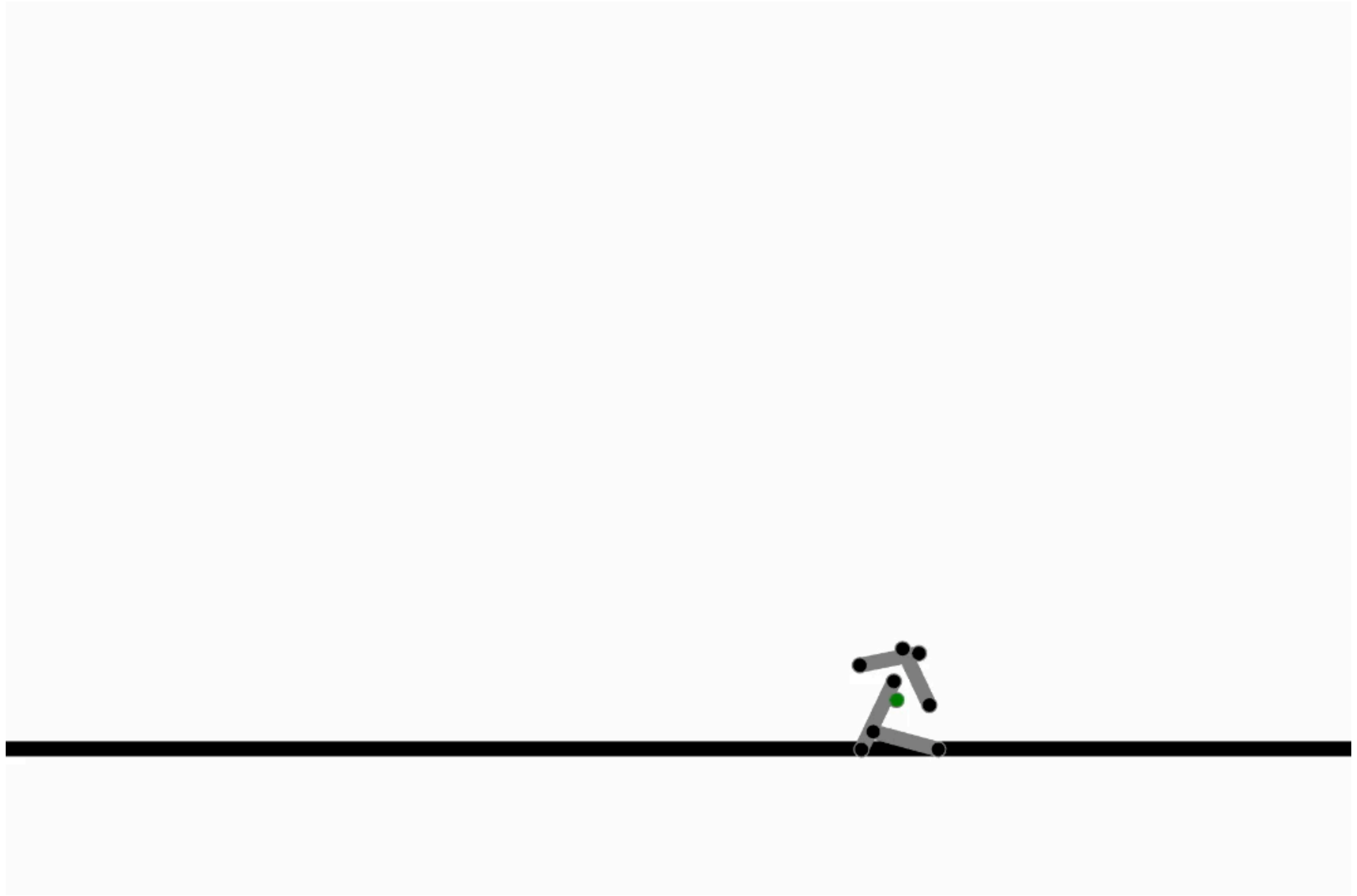
FUNDAMENTALS OF COMPUTER GRAPHICS

Animation & Simulation

Stanford CS248B, Fall 2023

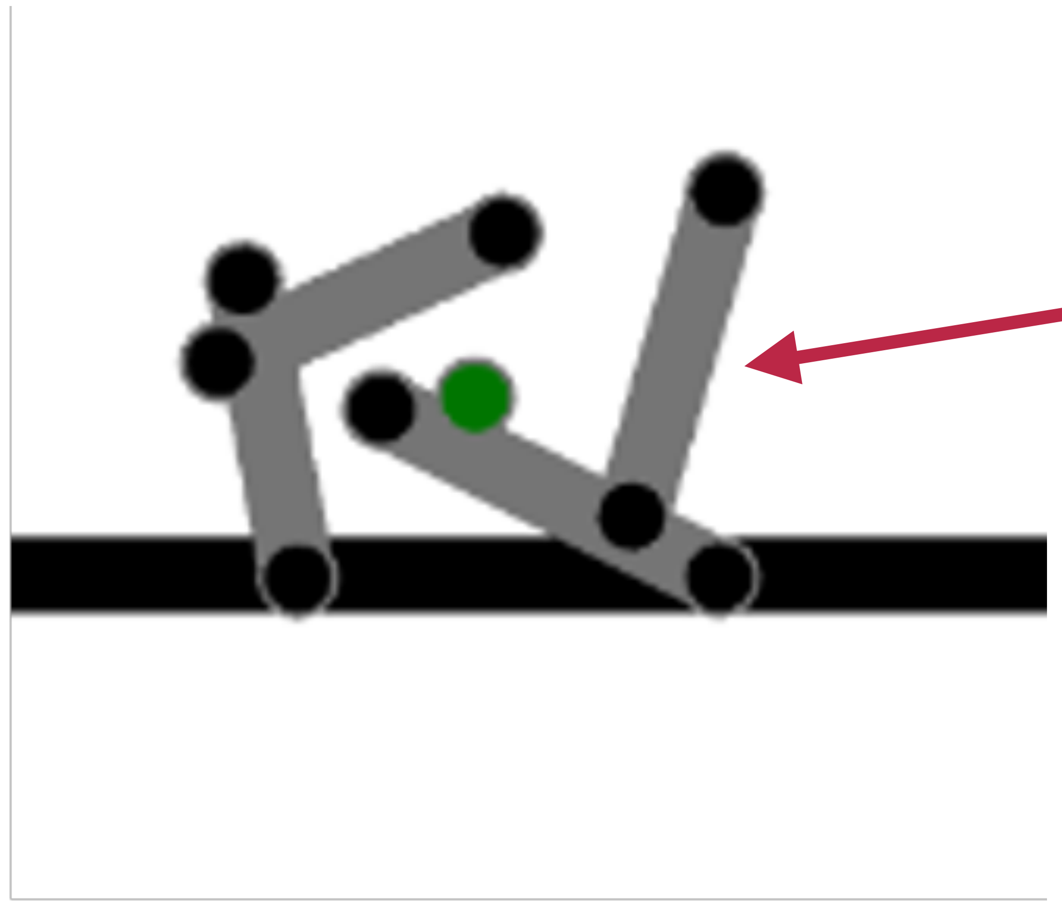
2D Rigid-body Sim for HW3 and Project 3

- P3 requires you to reuse what you will have built in HW3
- Offload the work for P3, and help you get familiar with math.js library
- Written part for HW3 will thus be shorter



2D Rigid-body Sim for HW3 and Project 3

- Rigid body: a bunch of 1D rods rigidly attached to each other
- Rods: all with uniform density
- Assume collisions with floor can only happen at the ends of the rods



the 1D rod

Position, Orientation, Linear/Angular Velocities

- They fully specify the state of the rigid body
- Position: $[x, y]$ since in 2D
- Linear Velocity: $[\dot{x}, \dot{y}]$
- What's the dimension of orientation in 2D?

Position, Orientation, Linear/Angular Velocities

- They fully specify the state of the rigid body

- What's the dimension of orientation in 2D? — 1 !
  - Orientation (angle, in radius): $[a]$
  - Or equivalently, as rotation matrix $\begin{bmatrix} \cos(a) & -\sin(a) \\ \sin(a) & \cos(a) \end{bmatrix}$
  - When to use which representation?
    - 1D angle for easily maintaining simulation state, rot matrix for transforming a vector (e.g. calculate rod pose for visualization)

Position, Orientation, Linear/Angular Velocities

- They fully specify the state of the rigid body

- What's the dimension of orientation in 2D? — 1 !
  - Orientation (angle, in radius) : $[a]$
  - Angular Velocity: $[\dot{a}]$

- The full state to maintain $[x, y, a, \dot{x}, \dot{y}, \dot{a}]$

What's a rod's position in world space?

- **Given: in body (CoM) space, a rod starts from (s_1, s_2) and ends in $(e1, e2)$**

$$\begin{bmatrix} s_1 \\ s_2 \end{bmatrix}_w =$$

- **Same for $(e1, e2)$**

What's a rod's position in world space?

- **Given: in body (CoM) space, a rod starts from** (s_1, s_2) **and ends in** $(e1, e2)$

$$\begin{bmatrix} s_1 \\ s_2 \end{bmatrix}_w = \begin{bmatrix} \cos(a) & -\sin(a) \\ \sin(a) & \cos(a) \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \end{bmatrix} + \begin{bmatrix} x \\ y \end{bmatrix}$$

- **Same for** $(e1, e2)$

Mass, Center of Mass

- **Mass:** scalar m
  - For Hw3/P3, sum of all rods $m = \sum_i l_i \rho$, where ρ is the density, kg/m
- **Center of Mass:**
  - Calculate once from initial rod positions before the simulation starts
  - The result CoM is initial $[x, y]$
  - Center of mass of the combined system can be found using the weighted sum of their individual centers of mass
  - For Hw3/P3, $\begin{bmatrix} x \\ y \end{bmatrix} = \frac{\sum_i l_i \rho [x_i, y_i]}{m}$, $[x_i, y_i]$ is the CoM of each rod (which is ?…)

Inertia

- **In 2D, what's the dimension of Inertia I?**

- **(Hint: what's the dimension of angular velocity or acceleration in 2D?)**

Inertia

- Just a scalar since there is only one rotation axis (one angle)
- $I = \sum_j m_j ||\mathbf{r}_j(t) - [x(t), y(t)]||_2^2$
  - Following the inertia definition in rigid-body Lecture by summing "infinite numbers of tiny particles" across the rigid body
  - Is I changing over time?

Inertia

- Just a scalar since there is only one rotation axis (one angle)

- $I = \sum_j m_j \| \mathbf{r}_j(t) - [x(t), y(t)] \|_2^2$
  - Following the inertia definition in rigid-body Lecture by summing "infinite numbers of tiny particles" across the rigid body
  - Is I changing over time? No! Contrast with 3D inertia definition

$$\mathbf{I}(t) = \sum_{i=1}^{N} \begin{bmatrix} m_i(r_{iy}'^2 + r_{iz}'^2) & -m_i r_{ix}' r_{iy}' & m_i r_{ix}' r_{iz}' \\ -m_i r_{iy}' r_{ix}' & m_i(r_{ix}'^2 + r_{iz}'^2) & -m_i r_{iy}' r_{iz}' \\ -m_i r_{iz}' r_{ix}' & -m_i r_{iz}' r_{iy}' & m_i(r_{ix}'^2 + r_{iy}'^2) \end{bmatrix}, \text{where } \mathbf{r}_i' = \mathbf{r}_i(t) - \mathbf{x}(t)$$

All elements being time varying

Inertia

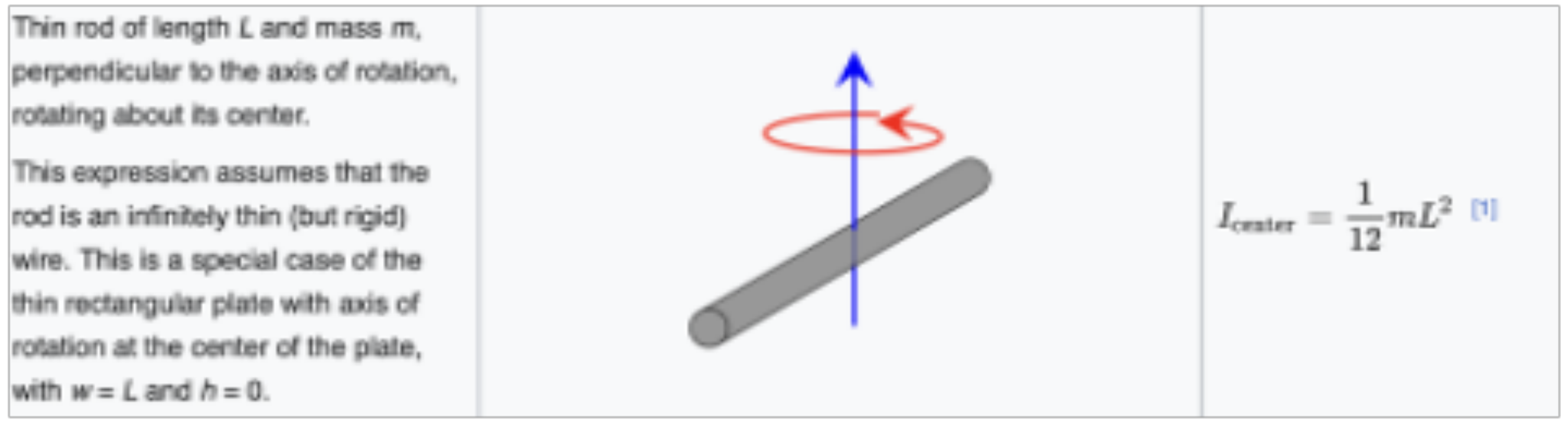
- So we just need to calculate it once before the start of the simulation, and use it throughout.

- How to compute?
  - Don't want to sum up many many particles
  - Instead, add up the inertias from each part of the rigid body

- For HW3/P3:
  - 1. Find inertia for each rod if rotating around the rod's center
  - 2. Apply parallel axis theorem for each rod, depending on their position w.r.t. CoM
  - 3. Add up transformed inertias for all rods

Parallel Axis Theorem

https://en.wikipedia.org/wiki/List_of_moments_of_inertia

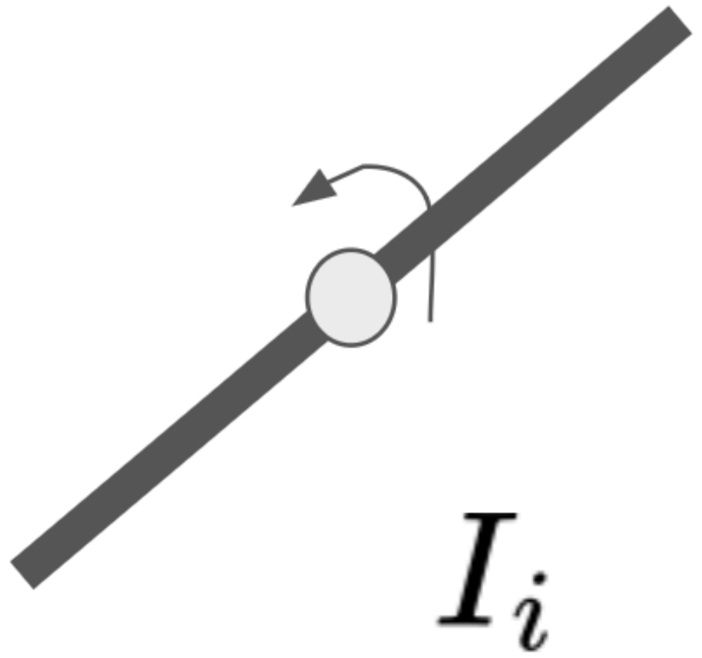


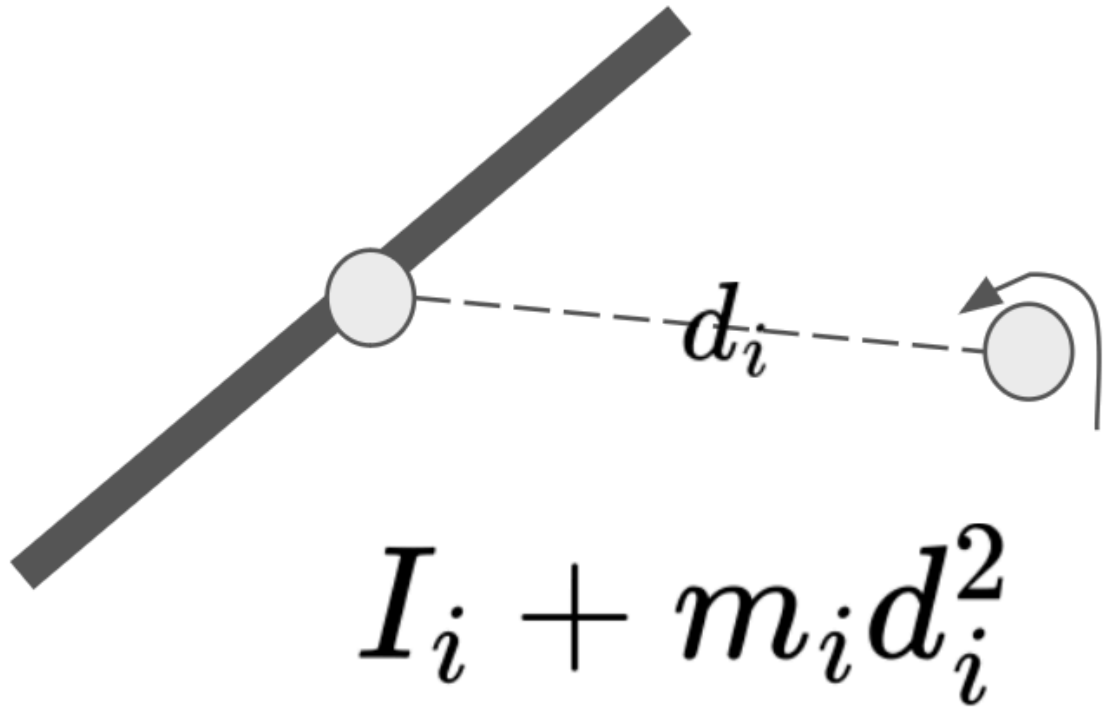
Thin rod of length L and mass m, perpendicular to the axis of rotation, rotating about its center. This expression assumes that the rod is an infinitely thin (but rigid) wire. This is a special case of the thin rectangular plate with axis of rotation at the center of the plate, with $w = L$ and $h = 0$.		$I_{center} = \frac{1}{12}mL^2$ [1]

- What's it for?
- Given inertia around one point (e.g. around center of rod which we have standard formula),
- what's the inertia around any point of interest (e.g. around rigid-body CoM)?

- The theorem:

.





Around axis where inertia is known

Around any given axis

Equations of motions (how to update the state of RB)

$$m \begin{bmatrix} \ddot{x} \\ \ddot{y} \end{bmatrix} = \begin{bmatrix} f_x \\ f_y \end{bmatrix}$$

Newton's second law

$$I\ddot{a} = \tau$$

Euler's second law for ration (2D)

- **For HW3/P3:**
- **Calculate forces & torques from gravity and collision (to be discussed)**
- **Then use your favorite integrator (e.g. symplectic Euler) to simulate in time**

Colliding forces for HW3 / P3

Recall: Symplectic Euler integrator with filters

1. **Accumulate forces:** $\mathbf{f}$ (springs, gravity, drag, etc.)
2. **Evaluate accelerations:** $\mathbf{a} = \mathbf{M}^{-1}\mathbf{f}$
   - ~~Optional: Inverse mass filtering for pinned particles~~
3. **Timestep velocities:** $\mathbf{v} \mathrel{+}= \Delta t\ \mathbf{a}$
   - Optional: Filter velocities for collisions and constraints
4. **Timestep positions:** $\mathbf{p} \mathrel{+}= \Delta t\ \mathbf{v}$
   - ~~Optional: Filter positions~~

Gravity forces, collision forces & torques

divided by m and I

Could solve for collision impulses instead of forces, not required for HW3/P3

An over-simplified collision force model

- You can just add some penalty spring forces — but feel free to improve it!

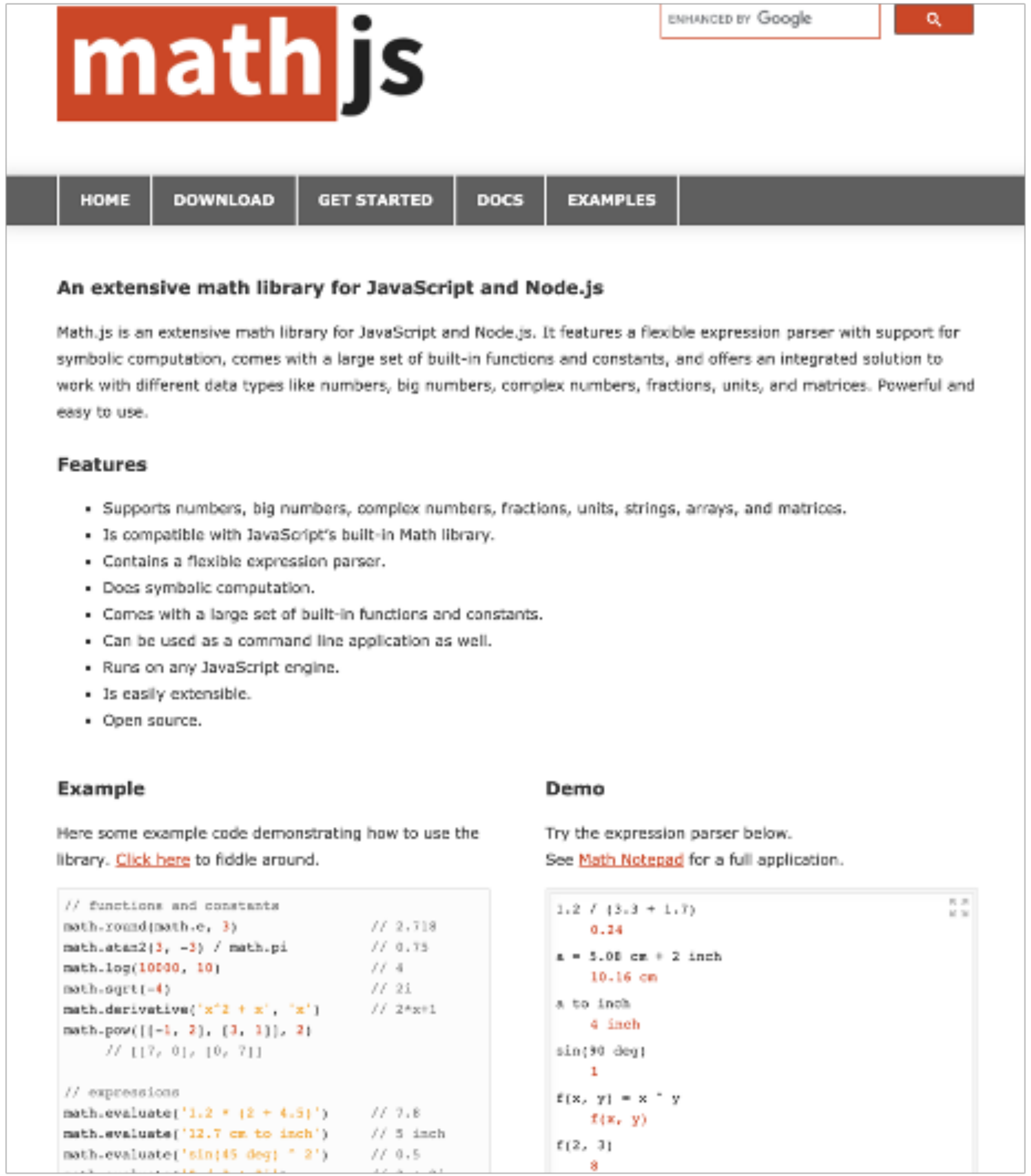
- Iterate through all points (x_k, y_k) on rigid body where
    - We consider collision only at start & end points of the rod,
    - Add force when close to floor (height c) and still going downwards
    - $\mathbf{f}_k = [-\mu \dot{x}_k, -w(y_k - c)]$, where $-\mu \dot{x}_k$ mimics friction effects

    - Don't forget to get τ_k from $\mathbf{f}_k$ and moment arm!

    - How to obtain $\dot{x}_k$? Maintain & store previous-step (x_k, y_k) and do subtracting from current-step (i.e. finite differencing)

math.js

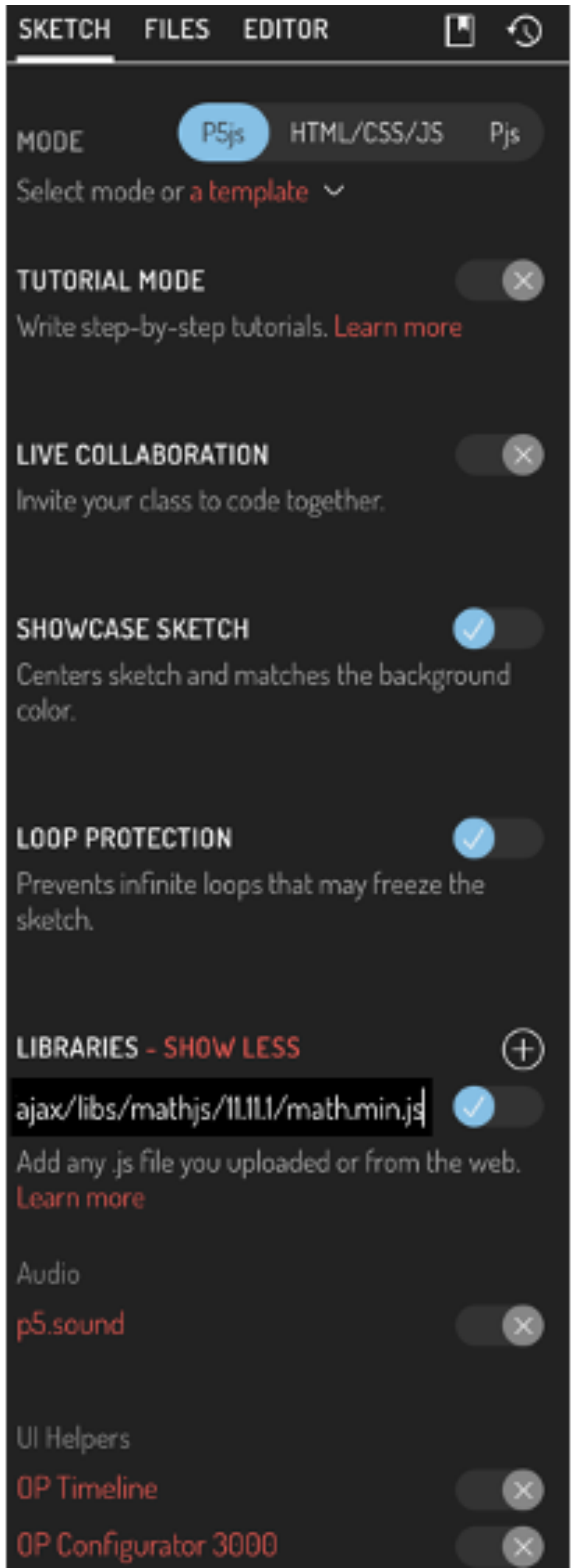
- **Need this for HW3 and P3 since we will be dealing with matrices!**
-



math.js

- Add math.js to your OpenProcessing include list

- Link to the library:
  - https://cdnjs.cloudflare.com/ajax/libs/mathjs/11.11.1/math.min.js



math.js Basics

- **Playground: https://jsbin.com/devacu/edit**

```
// operates on native arrays, no need for CreateVector!
a = [1., 2, 3, 4]

print(math.add(a, 2))
print(a + 2)      // you don't want this except for printing multiple things
// print(a.add(2))   // not allowed

print(math.multiply(0.2, a))

print("a now " + a)  // a is not changed

print(math.add(a.slice(0,2), a.slice(2,4)))

print(math.norm(a))  // returns a scalar

print("")


// Be careful, 2-norm is the default, and for matrix is not element-wise
b = [[1., 3.], [2., 4.]]
print(math.norm(b, 'fro'))
print(math.norm(b, 2))
print(math.norm(math.flatten(b)))


// helper function to output formatted results.
function print(value) {
  var precision = 14;
  document.write(math.format(value, precision) + '<br>');
}
```

```
[3, 4, 5, 6]
"1,2,3,42"
[0.2, 0.4, 0.6, 0.8]
"a now 1,2,3,4"
[4, 6]
5.4772255750517
""
5.4772255750517
5.464985704219
5.4772255750517
```

math.js Basics

- Suggestion? Flatten a column vector when you can, if you are more used to Python convention

```
<!DOCTYPE html>
<html>
<head>
  <meta name="description" content="math.js | basic usage">
  <title>math.js | basic usage</title>
  <script src="https://unpkg.com/mathjs/lib/browser/math.js"></script>
</head>
<body>
  <script>
    // basic usage of math.js
    //
    // website:  http://mathjs.org
    // docs:     http://mathjs.org/docs
    // examples: http://mathjs.org/examples

    // operates on native arrays, no need for CreateVector!
    a = [[1., 2], [3, 4]];
    print(a);

    a_mult_vec = math.multiply(a, [2,3]);
    print(a_mult_vec);

    a_mult_vec = math.multiply(a, [[2],[3]]);   // equivalent
    print(a_mult_vec);                 // but output in different shape

    a_mult_a = math.multiply(a, a);
    print(a_mult_a);

    print("   ");

    c = math.subset(a, math.index(1, [0, 1]))     // get subset
    print(c);

    c = math.subset(a, math.index(1, [0, 1]), 88)    // replace subset
    print(c);
```

Output

```
[[1, 2], [3, 4]]
[8, 18]
[[8], [18]]
[[7, 10], [15, 22]]
" "
[[3, 4]]
[[1, 2], [88, 88]]
```

A note on math.js Matrix class

- **Suggestion? Do not use Matrix for HW3/P3. "Both regular JavaScript arrays as well as the matrix type implemented by math.js can be used interchangeably in all relevant math.js functions"**

```
// operates on native arrays, no need for CreateVector!
a = [1., 2, 3, 4];
print(a);

b = math.matrix(a)
print(b);     // same

// math.matrix bascially is a wrapper around native arrays
print(b.type)
print(b._data)      // you get back the native array
print(a._data)

print("")

c = math.add(a, [2.,3,4,5])
print(c)
print(c._data)    // c is still native array

print("")

d = math.add(b, [2.,3,4,5])
print(d)
print(d._data)    // d is also math.matrix now!

print("")
```

Output

```
[1, 2, 3, 4]
[1, 2, 3, 4]
"DenseMatrix"
[1, 2, 3, 4]
undefined
""
[3, 5, 7, 9]
undefined
""
[3, 5, 7, 9]
[3, 5, 7, 9]
""
```

A note on math.js Matrix class

- Be careful not to involve the Matrix class in unexpected ways…

```
// examples: http://mathjs.org/examples

e = math.zeros(3);     // will return math.matrix object
print(e);
print(e._data);

print("");

f = math.zeros([4]);    // if you don't want to involve math.matrix wrapper
print(f);
print(f._data);

print("");

g = math.identity(3);
print(g);
print(g._data);

print("");

g = math.identity([3]);
print(g);
print(g._data);

print("");
```

Output

```
[0, 0, 0]
[0, 0, 0]
""
[0, 0, 0, 0]
undefined
""
[[1, 0, 0], [0, 1, 0], [0, 0, 1]]
[[1, 0, 0], [0, 1, 0], [0, 0, 1]]
""
[[1, 0, 0], [0, 1, 0], [0, 0, 1]]
undefined
""
```