# Simulation (next two lectures)

How used in games?

Dynamics

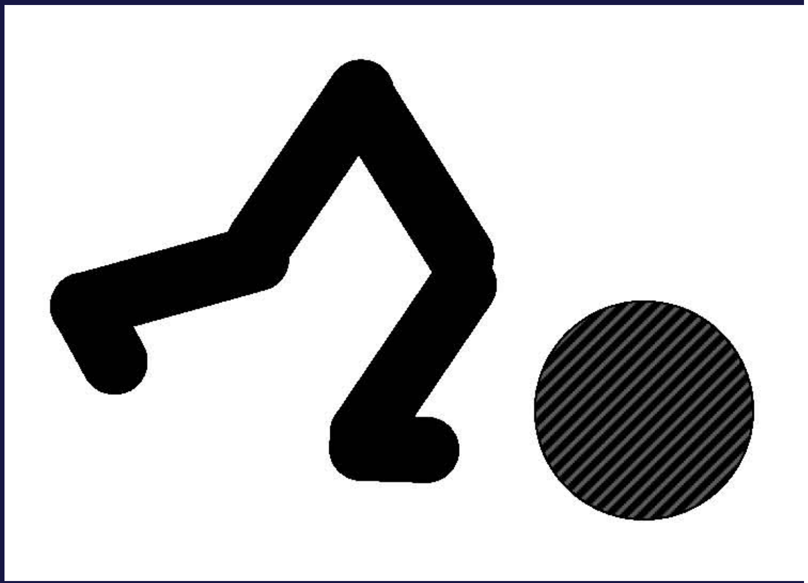
Collisions—simple

Controllers

Collisions—harder

Mocap + simulation

What is the future?



# Credits

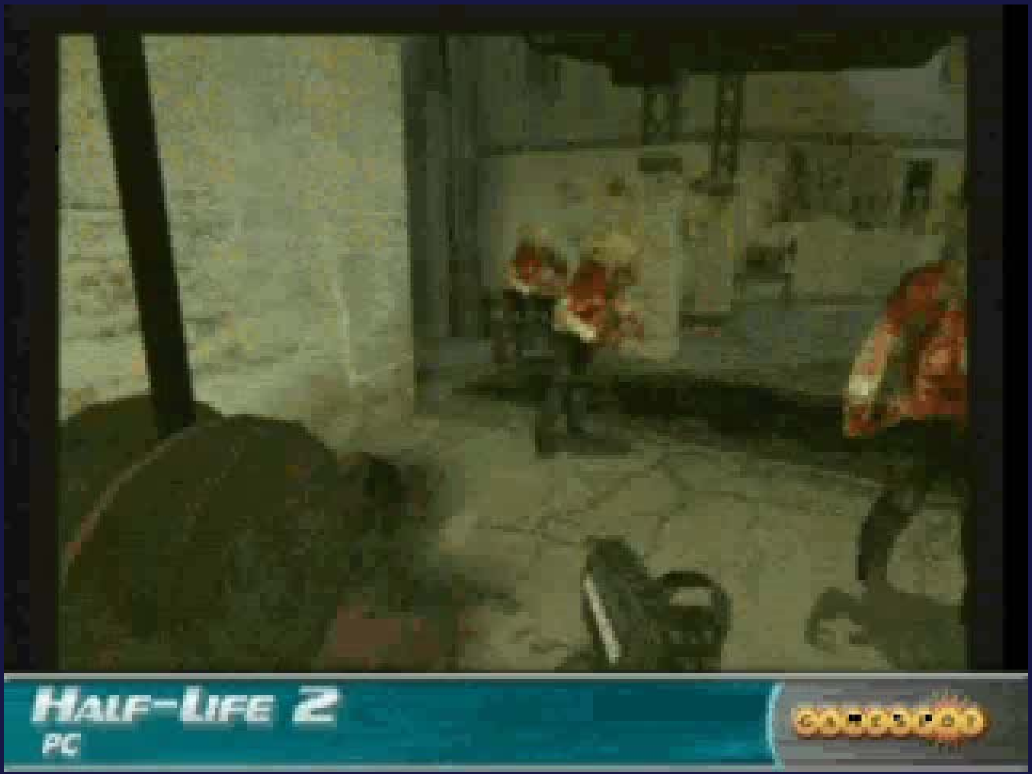
Many slides from Witkin and Baraff SIGGRAPH course (ptr on class page)

Examples and demos from

- Michiel van de Panne (UBC)
- Michael Mandel's talk at GDC (CMU alum)
- Victor Zordan (UC Riverside)

# How is simulation used in games?

- Vehicle dynamics
- Ponytails
- Simple bouncing objects
- Ragdoll physics
- What else?



# How is simulation used in games?

- Vehicle dynamics
- Ponytails
- Simple bouncing objects
- Ragdoll physics
- What else?

# Demo of Ragdoll Physics in ODE


40.00 FPS

# What do you need?

Path from model -> dynamic parameters

Dynamic equations

Control (internal forces/torques)?

Collisions (external forces/torques)

User control

# Dynamic System

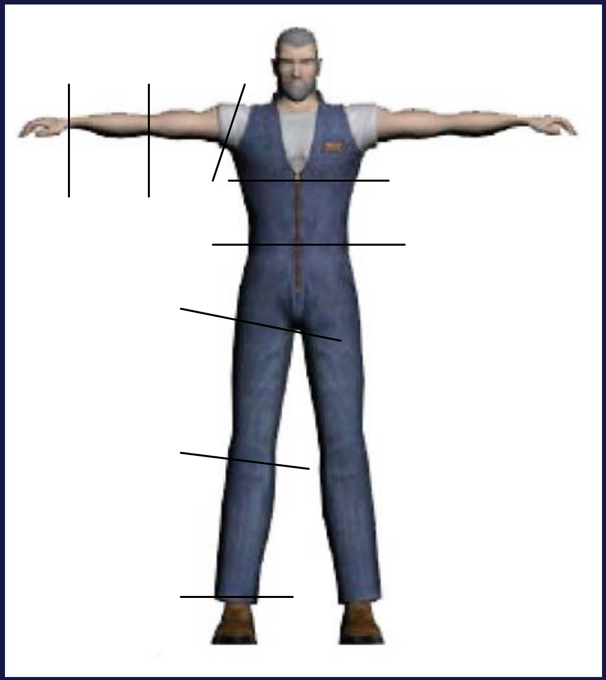
- Mass
- Moment of Inertia
- Location of Joints

From there it is just a compile step…

# Mass

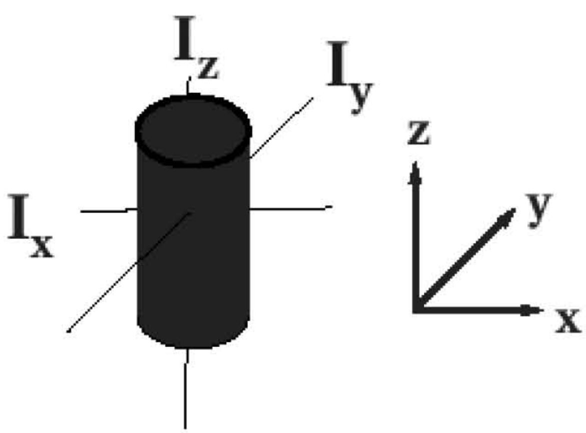
- Need volume of shape
- Assumption about density

High accuracy may not matter here?



# Moment of Inertia

## Inertia Tensor for Simple Shapes




$$I_x = I_y = 1/12\ m\ (3r^2 + L^2)$$

$$I_z = \frac{mr^2}{2}$$

# Moment of Inertia

- *Brian Mirtich*
  - Fast and accurate computation of polyhedral mass properties, JGT 1996

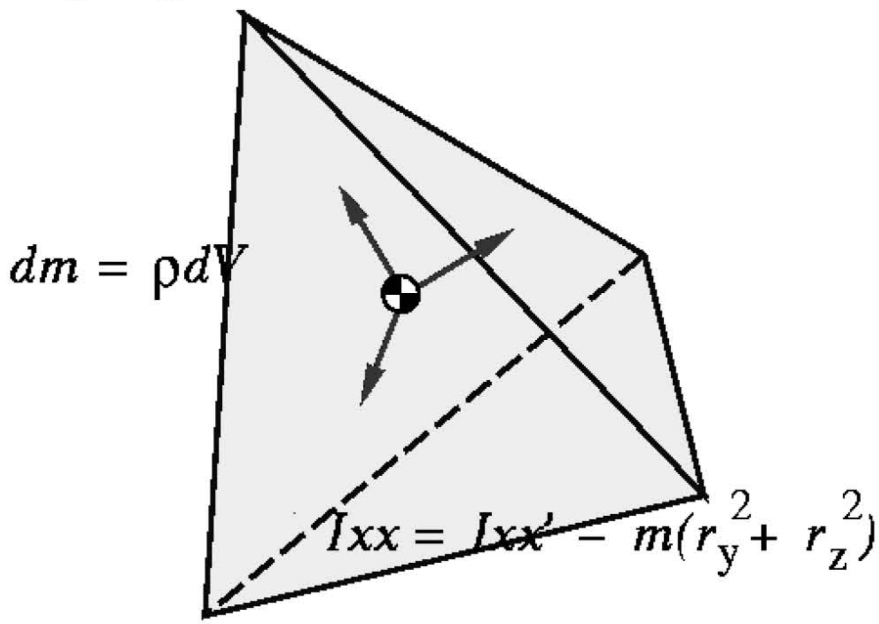
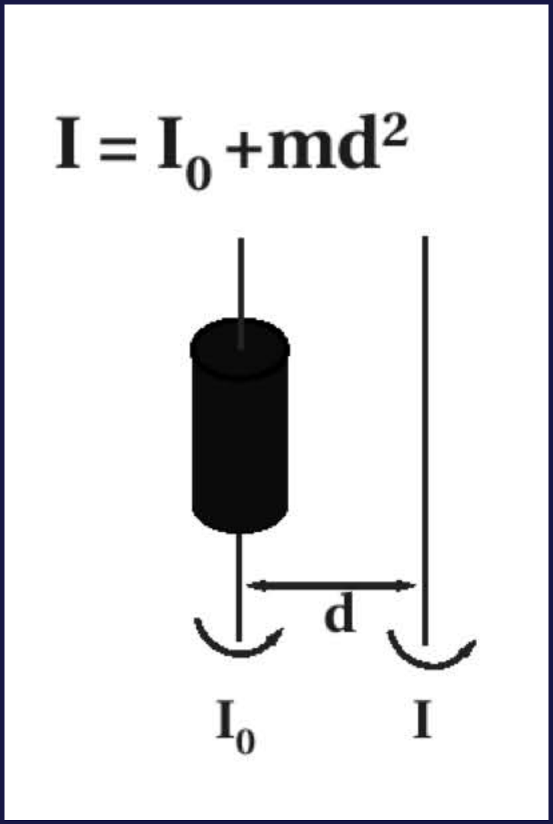
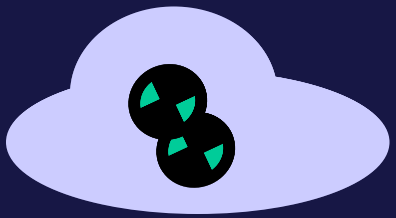



Illustration on blackboard

# Parallel Axis Theorem




Allows assembly of parts that will always move together



# Software Requirements

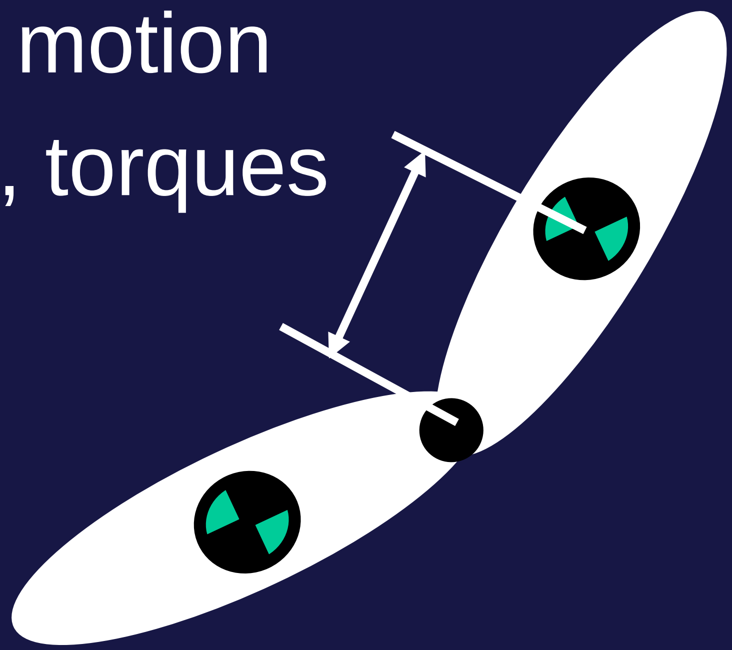
Link: mass, moment of inertia

Joints: DOF, distance from COM of links

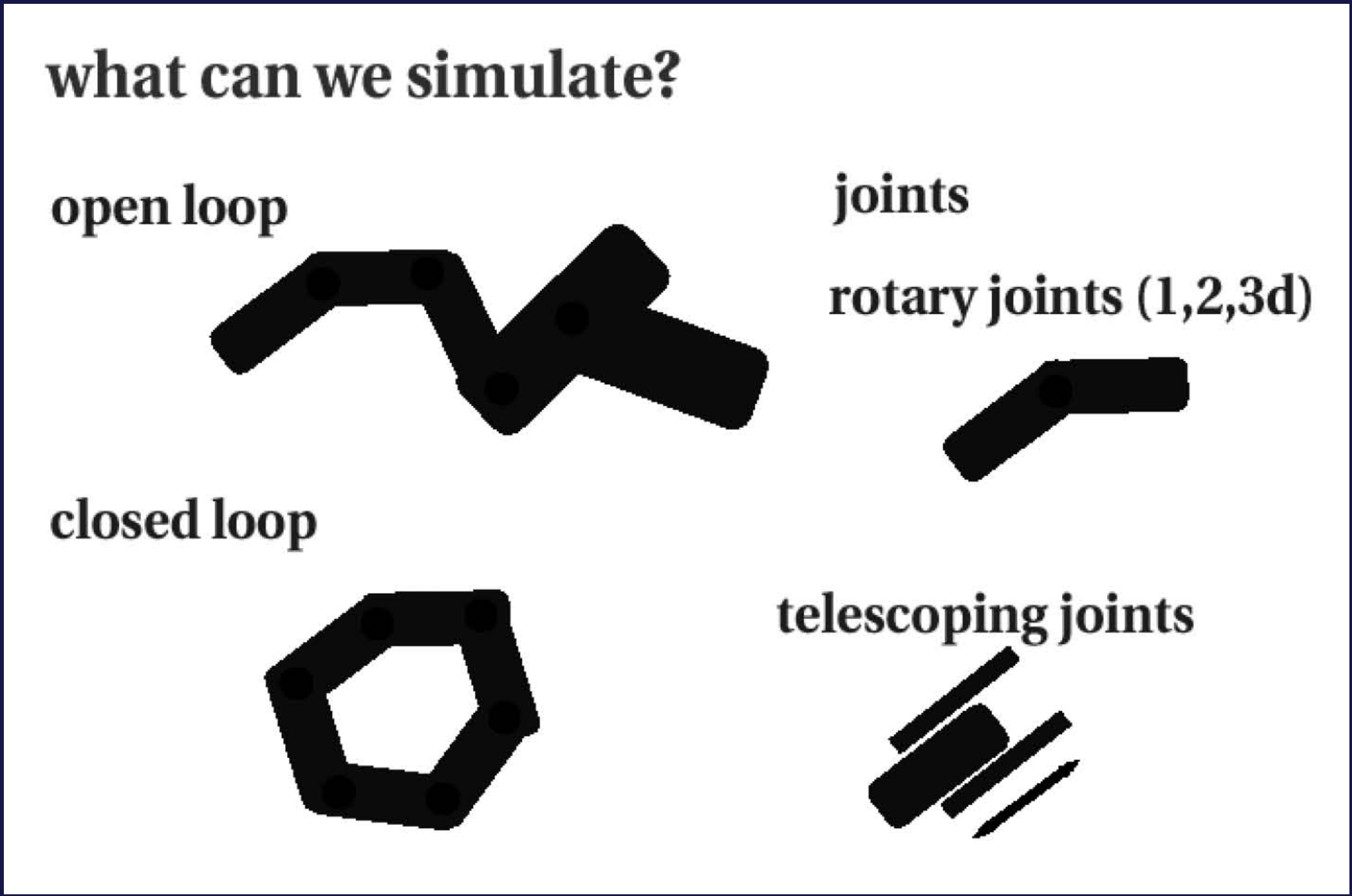
Code for the equations of motion

Hooks for applying forces, torques

Joint limits



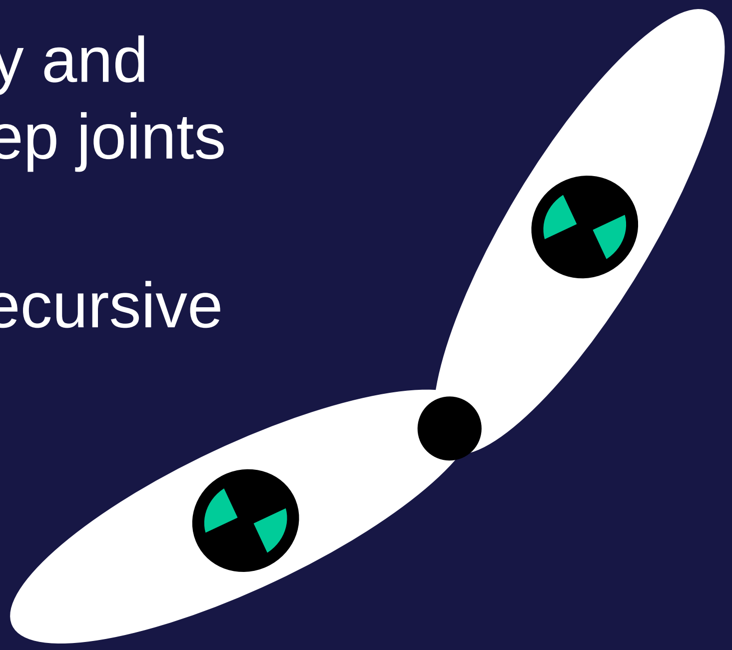
# Linked Rigid Bodies



# Linked Rigid Bodies

Two approaches:

- Treat each link separately and apply constraints to keep joints together
- Only allow legal DOFs (recursive forward algorithms)



# Software Options

- SDFast
- ODE
- Novodex
- Others??

# Particles—Equations of Motion

- Just one particle
- Particle systems
- Forces, gravity, springs
- Digression for integration
- Simple collisions

# A Newtonian Particle

- **Differential equation: f = ma**
- **Forces can depend on:**
  - **Position, Velocity, Time**

$$\ddot{\mathbf{x}} = \frac{\mathbf{f}(x,\dot{x},t)}{m}$$

# Second Order Equations

$$\ddot{x} = \frac{f(x, \dot{x}, t)}{m}$$

$$\begin{cases} \dot{x} = v \\ \dot{v} = f/m \end{cases}$$

Not in our standard form for differential equations because it has 2nd derivatives

Add a new variable to get a pair of coupled 1st order equations

# Phase Space

$$\begin{bmatrix} \mathbf{x} \\ \mathbf{v} \end{bmatrix}$$

Concatenate x and v to make a vector of length 6: position in phase space

$$\begin{bmatrix} \dot{\mathbf{x}} \\ \dot{\mathbf{v}} \end{bmatrix}$$

Velocity in phase space: another vector of length 6

$$\begin{bmatrix} \dot{\mathbf{x}} \\ \dot{\mathbf{v}} \end{bmatrix} = \begin{bmatrix} \mathbf{v} \\ \mathbf{f}/m \end{bmatrix}$$

Vanilla 1st order differential equation

# Particle Structure

**x** — Position

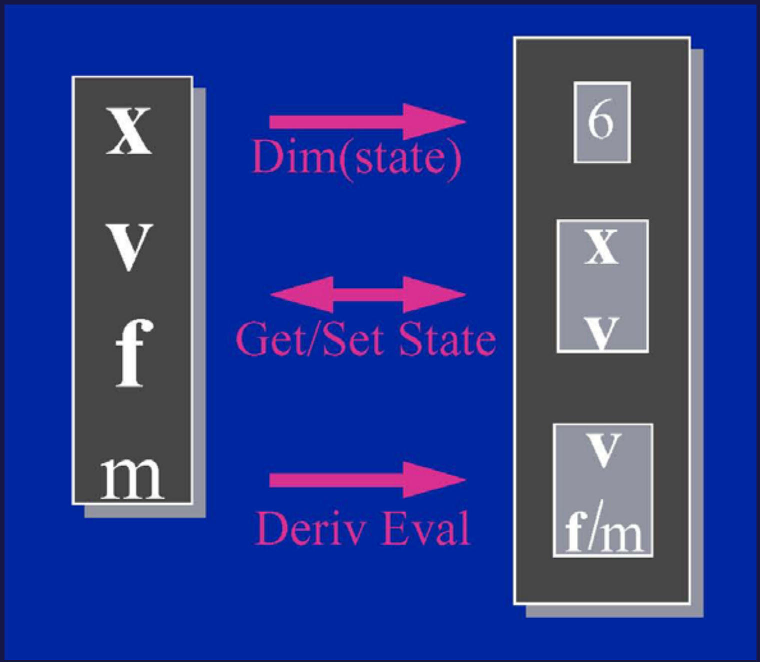
**v** — Velocity

**f** — Force Accumulator

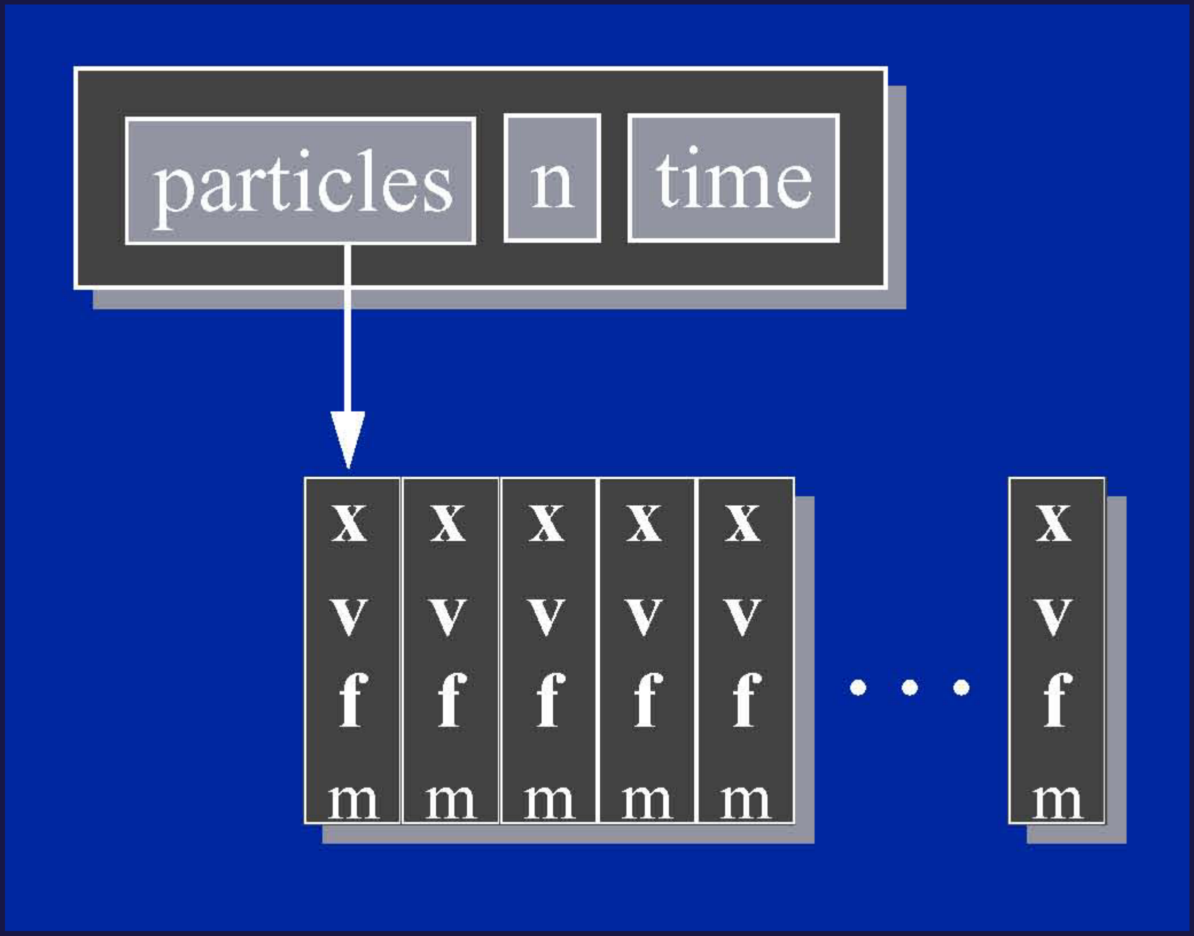
**m** — mass

Position in Phase Space (x, v)

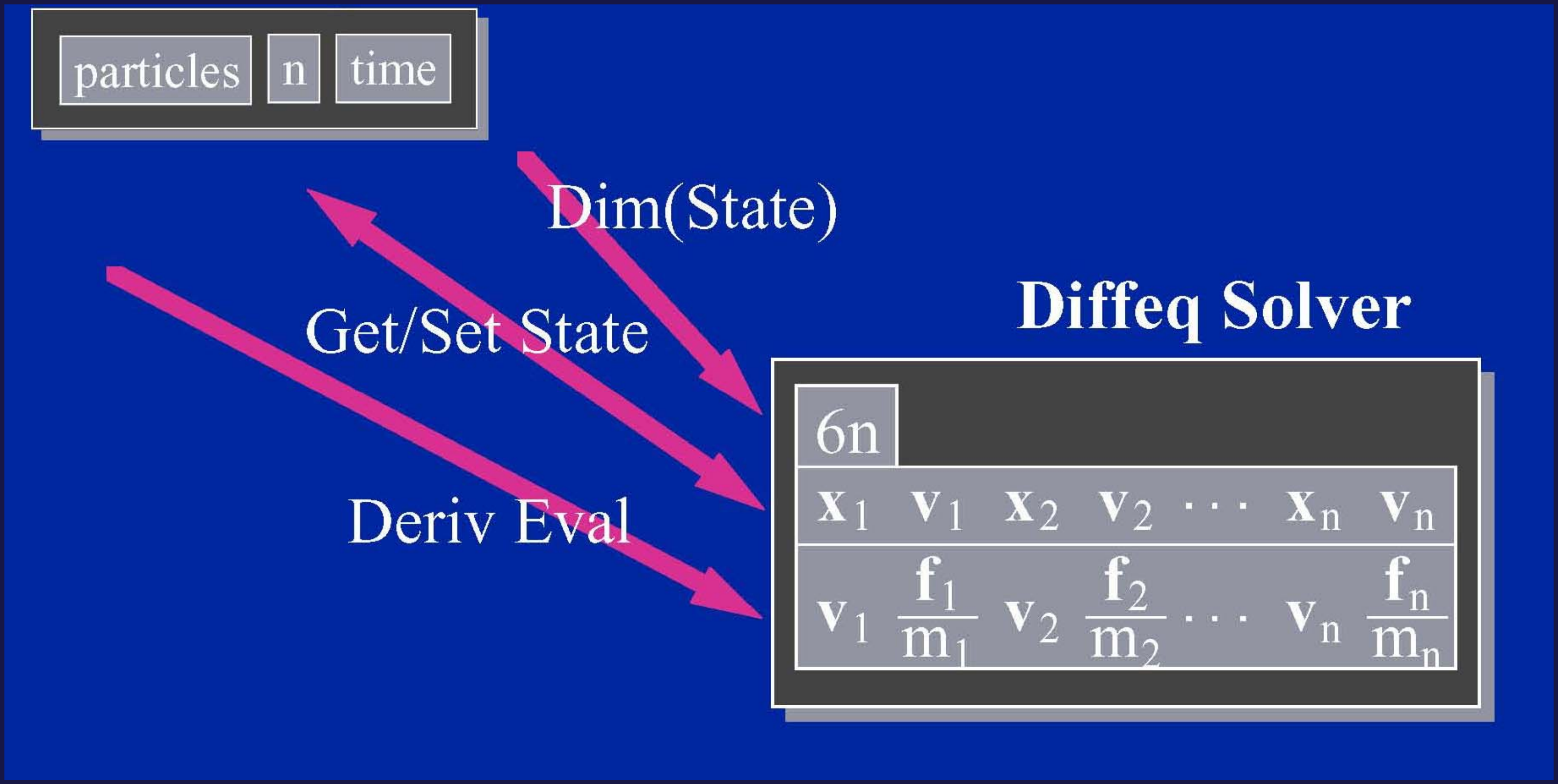
# Solver Interface



# Particle Systems



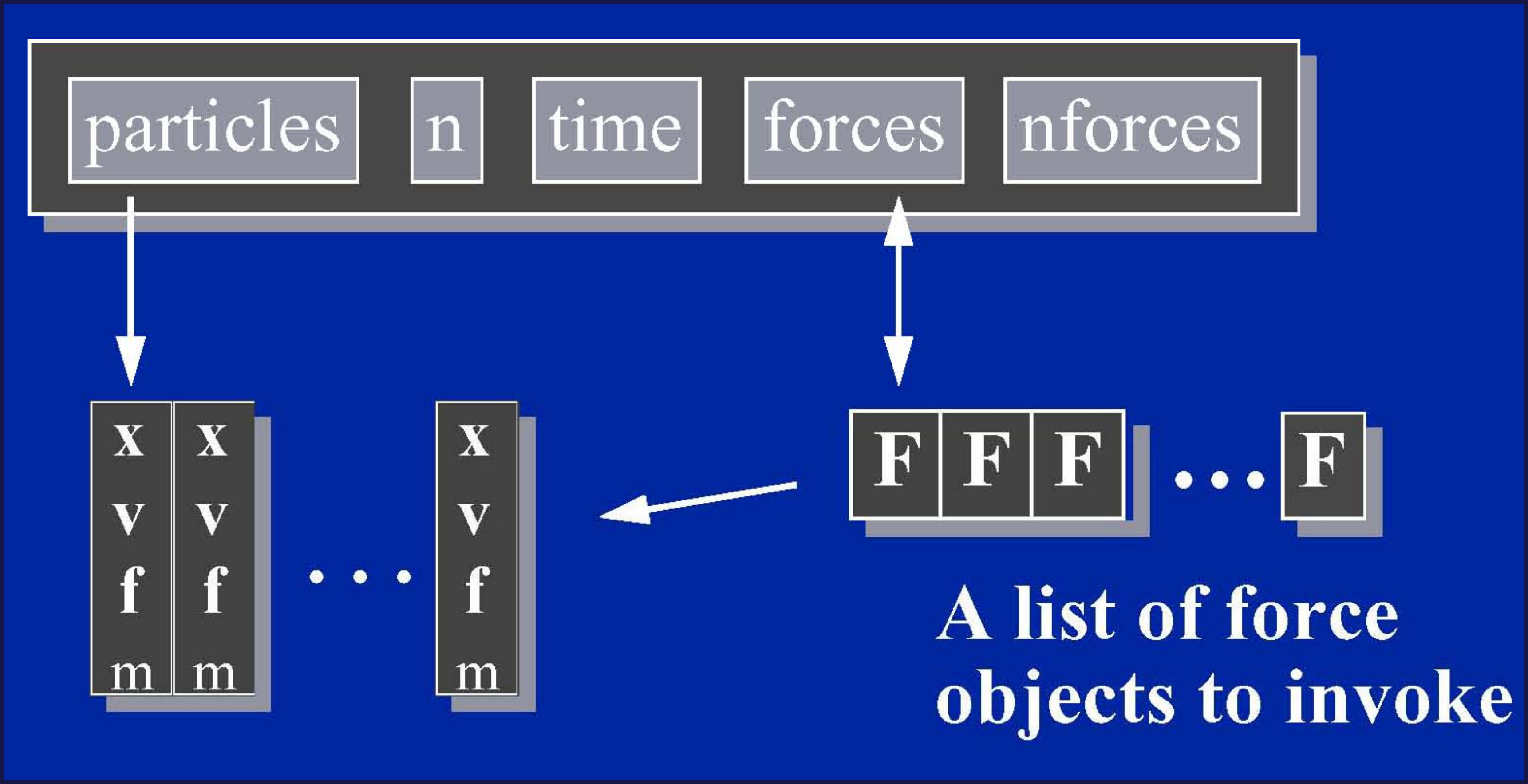
# Solver Interface



# Evaluation Loop

- Clear forces
  - Loop over particles, zero force accumulators
- Calculate forces (haven't talked about these)
  - Sum all forces into accumulators
- Gather
  - Loop over particles, copying v and f/m into destination array

# Particle Systems, with forces



# Forces

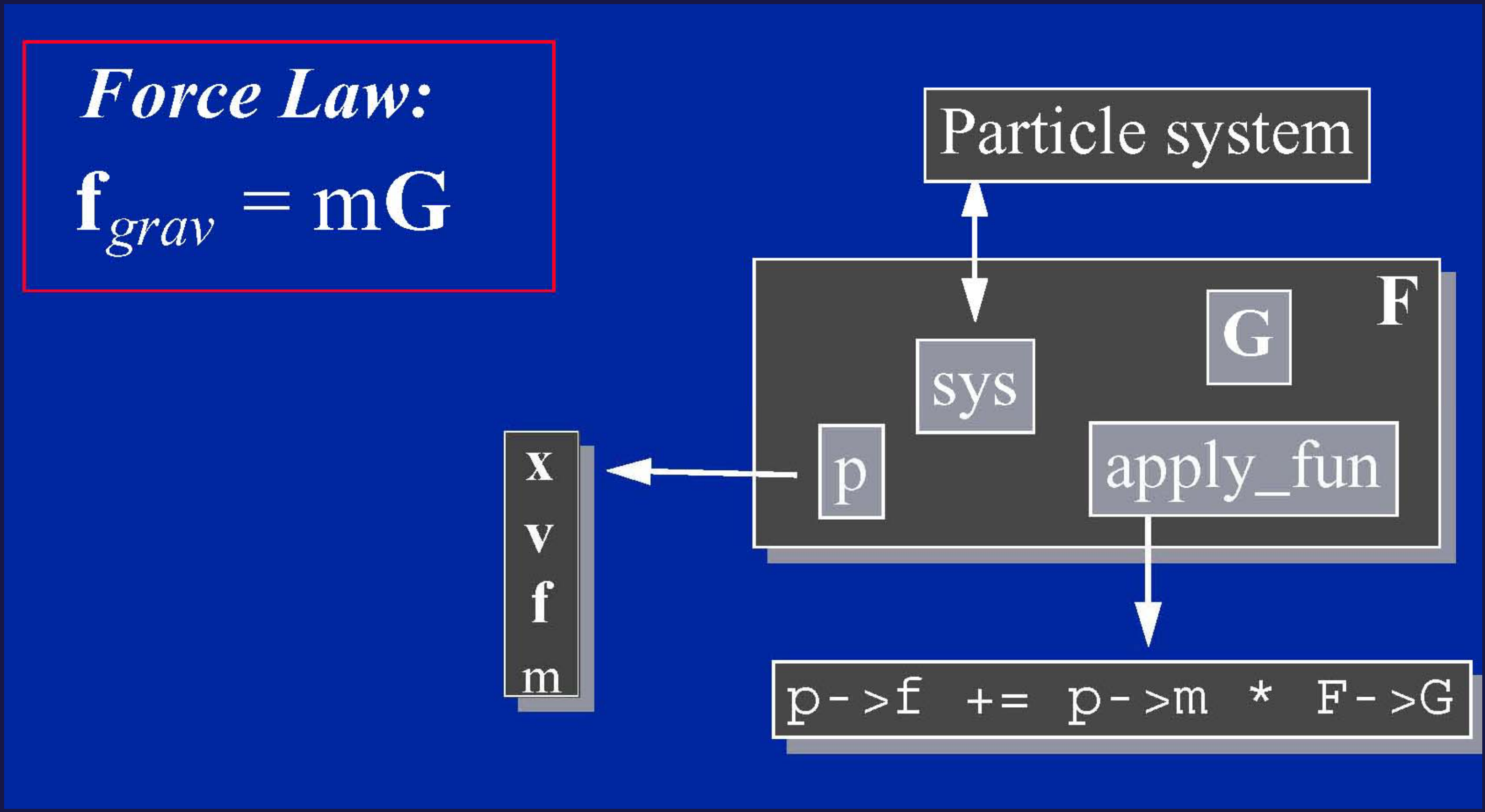
Constant—gravity

Position/Time dependent—wind fields

Velocity dependent—drag

N-ary—springs

# Gravity

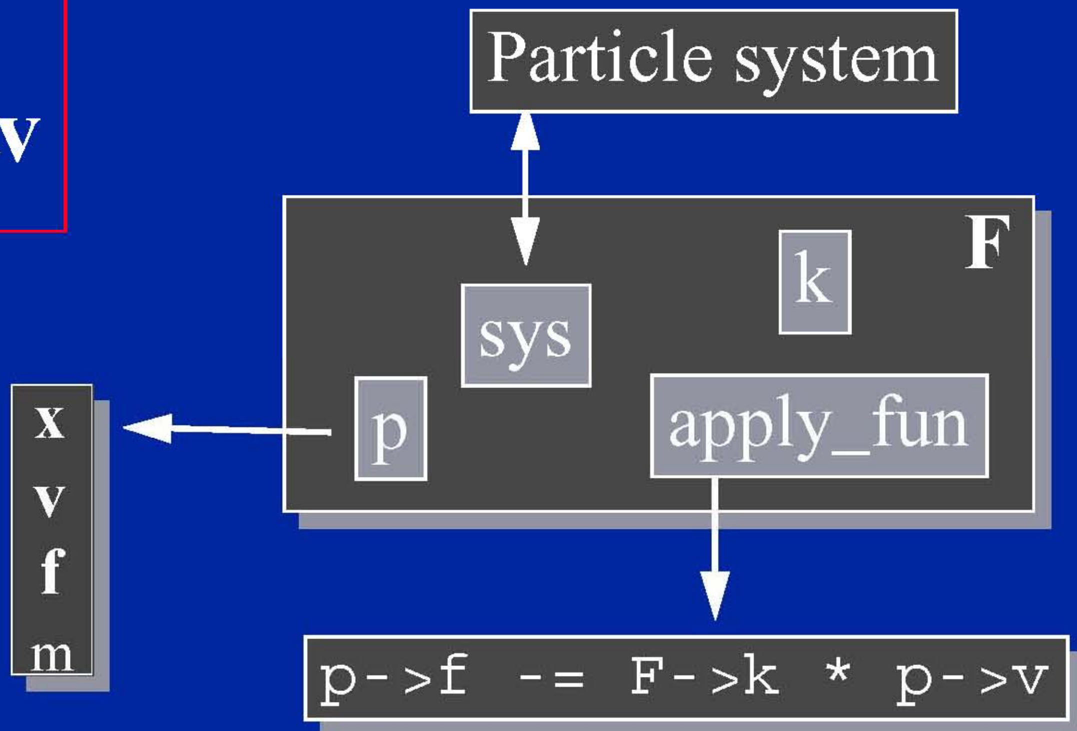


Force Law:
$f_{grav} = mG$
Particle system
F
G
sys
p
apply_fun
x
v
f
m
p->f += p->m * F->G

# Viscous Drag

***Force Law:***

$$\mathbf{f}_{drag} = -k_{drag}\mathbf{v}$$

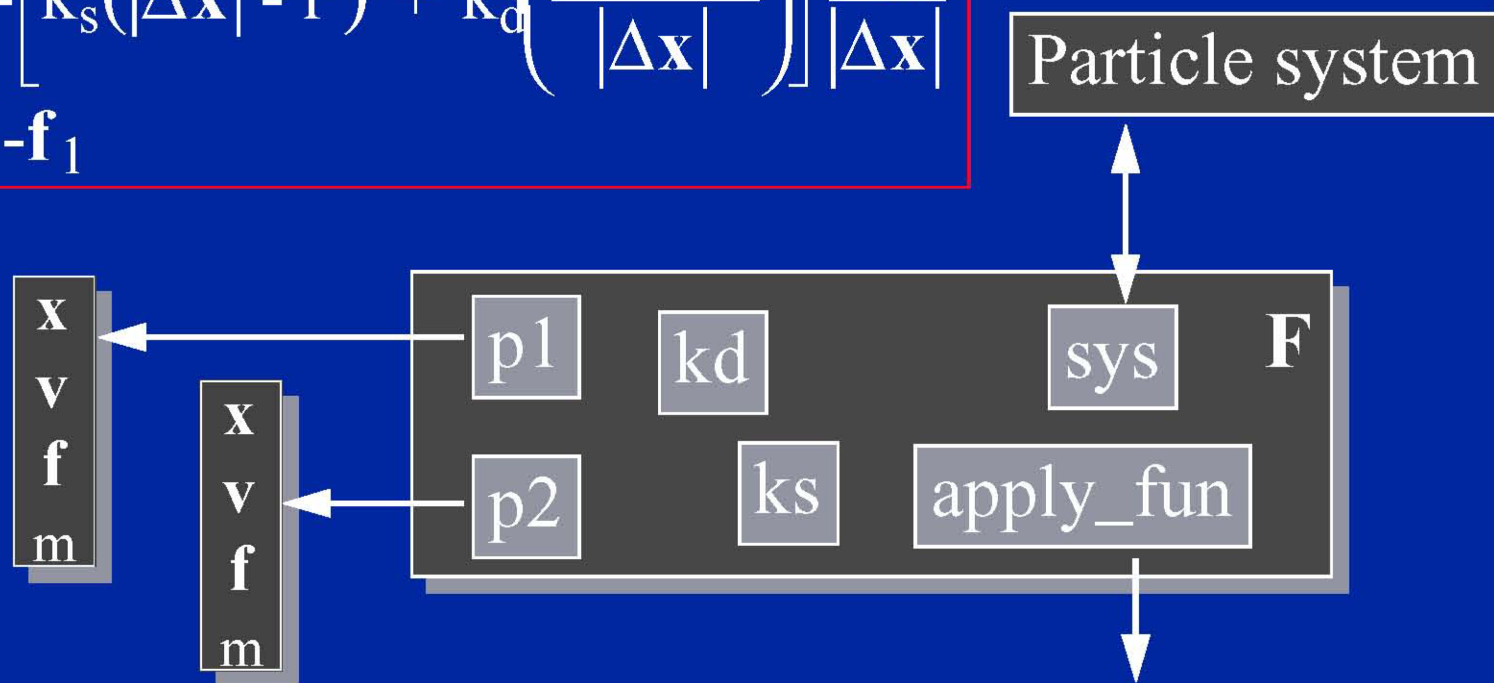


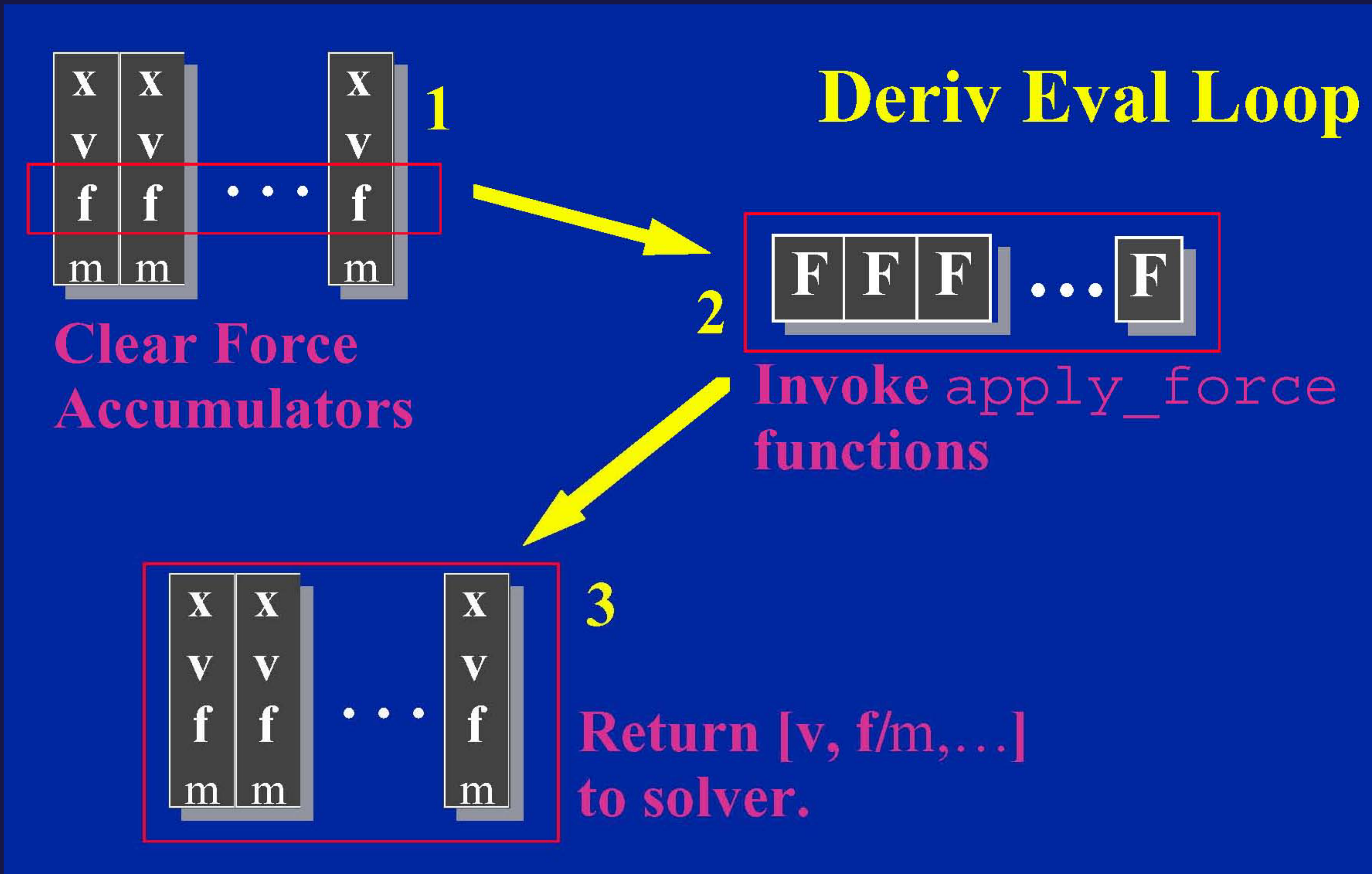
# Damped Spring

***Force Law:***

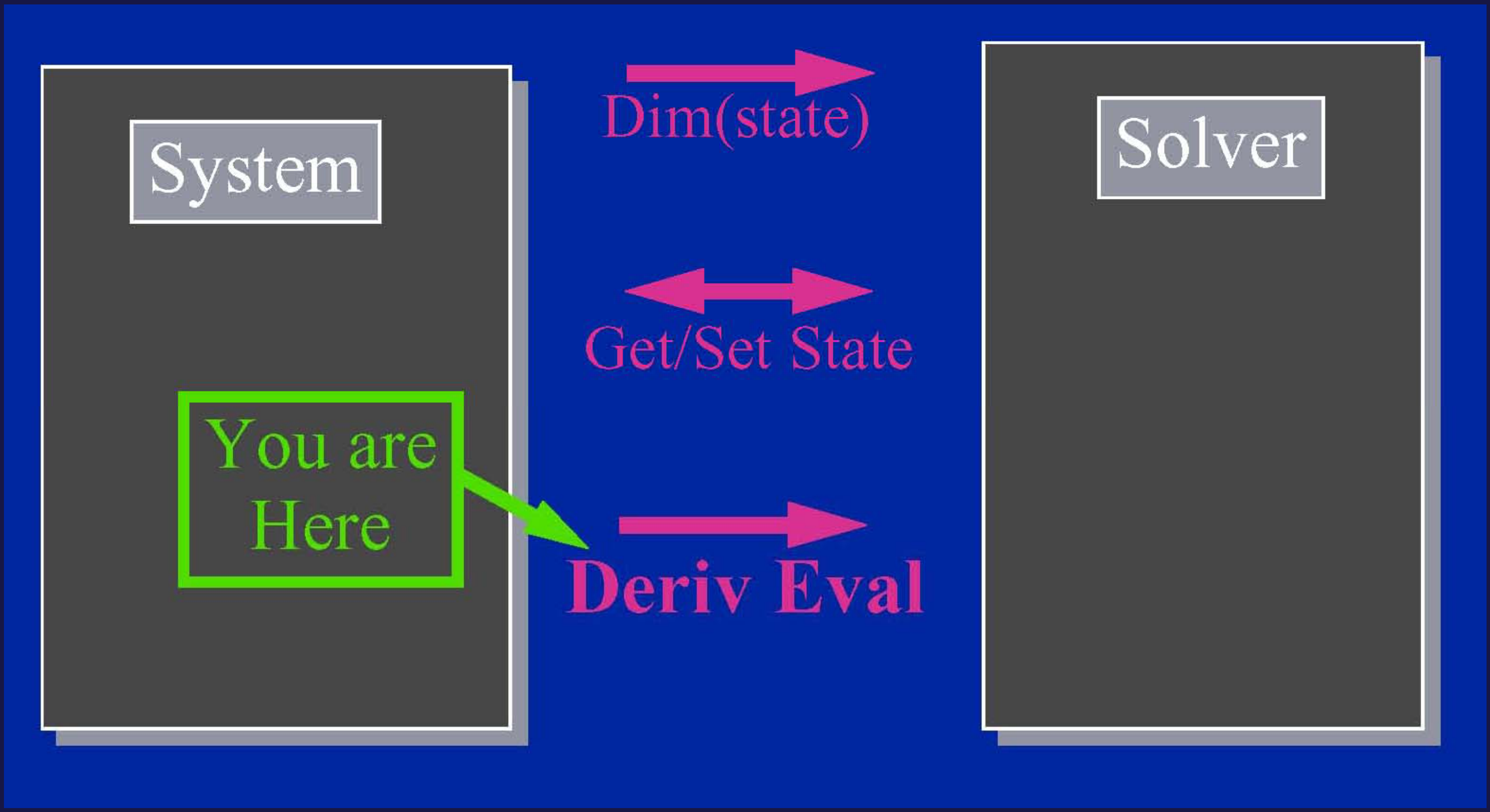
$$\mathbf{f}_1 = -\left[k_s(|\Delta\mathbf{x}| - r) + k_d\left(\frac{\Delta\mathbf{v}\cdot\Delta\mathbf{x}}{|\Delta\mathbf{x}|}\right)\right]\frac{\Delta\mathbf{x}}{|\Delta\mathbf{x}|}$$

$$\mathbf{f}_2 = -\mathbf{f}_1$$

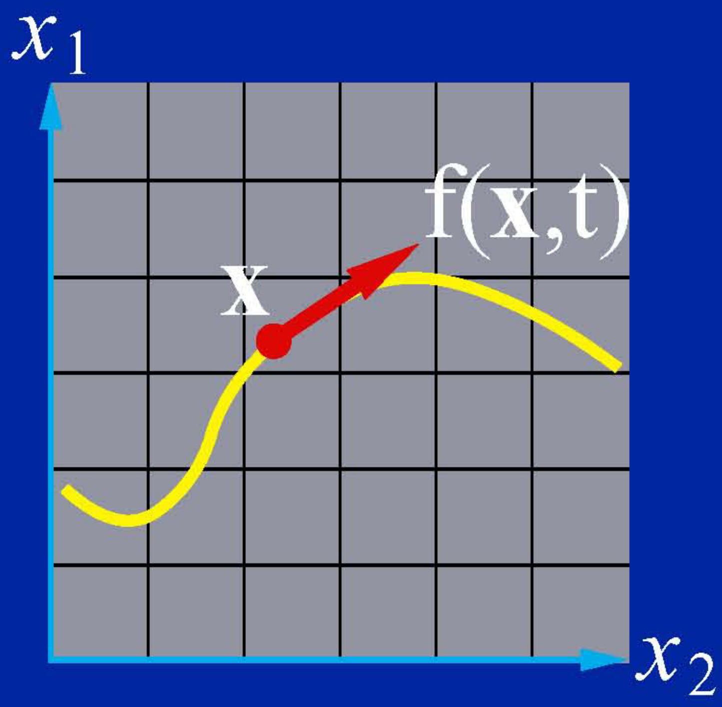



Deriv Eval Loop
x v f m
x v f m
x v f m
1
Clear Force Accumulators
2
F F F F
Invoke apply_force functions
3
x v f m
x v f m
x v f m
Return [v, f/m,…] to solver.

# Solver Interface



# Digression for integration: a differential equation

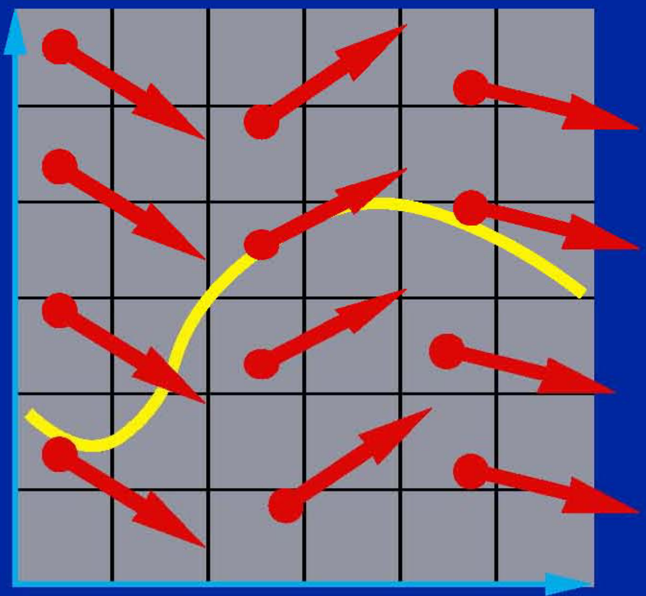



$$\dot{\mathbf{x}} = f(\mathbf{x},t)$$

- **x(t): a moving point.**
- **f(x,t): x's velocity.**

f is function not force here (sorry)

# Vector Field

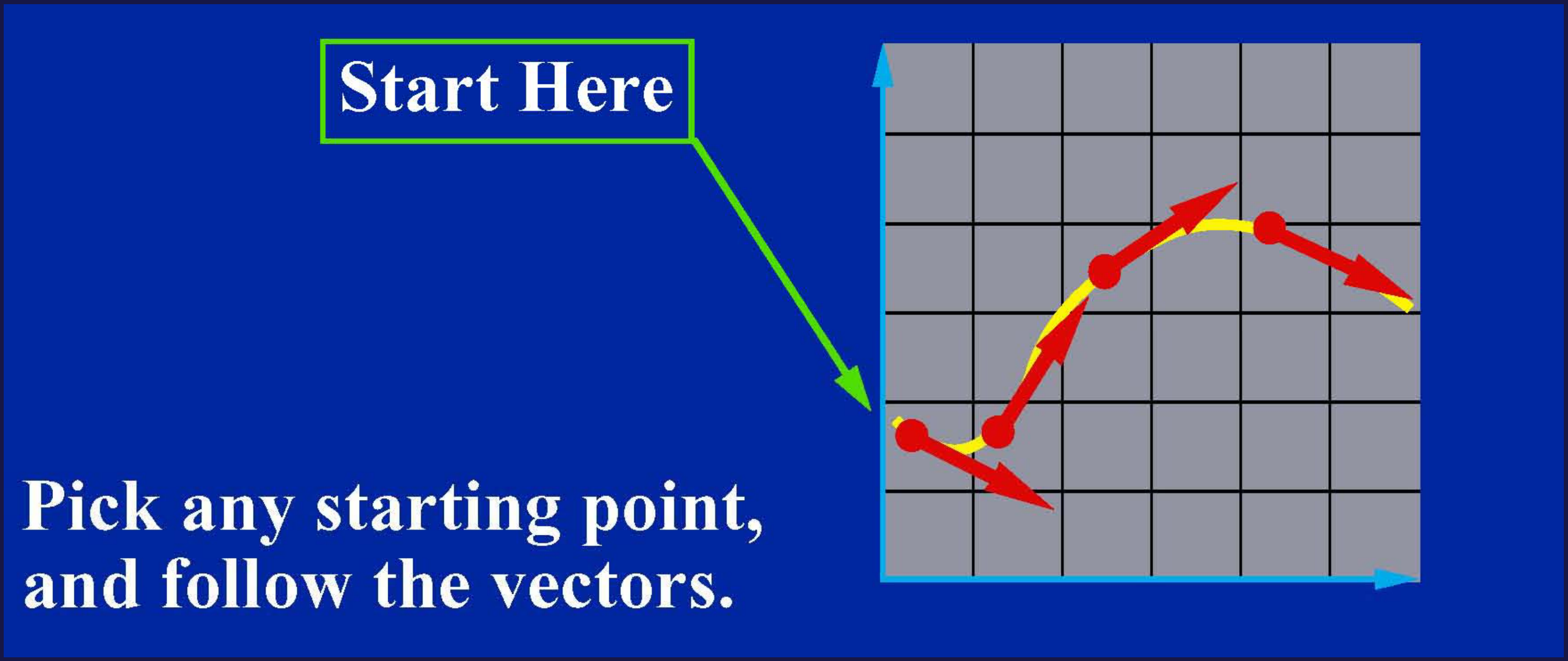


The differential equation

$$\dot{x} = f(x,t)$$

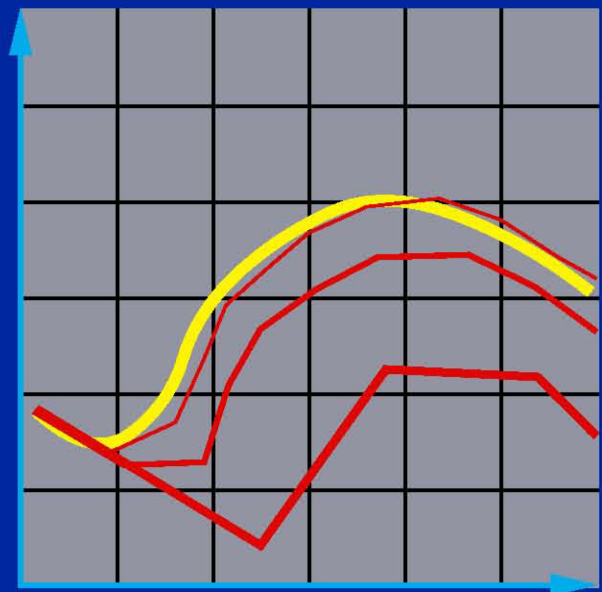
defines a vector field over x.

# Integrating along the curve




But how to use those vectors to follow the curve?

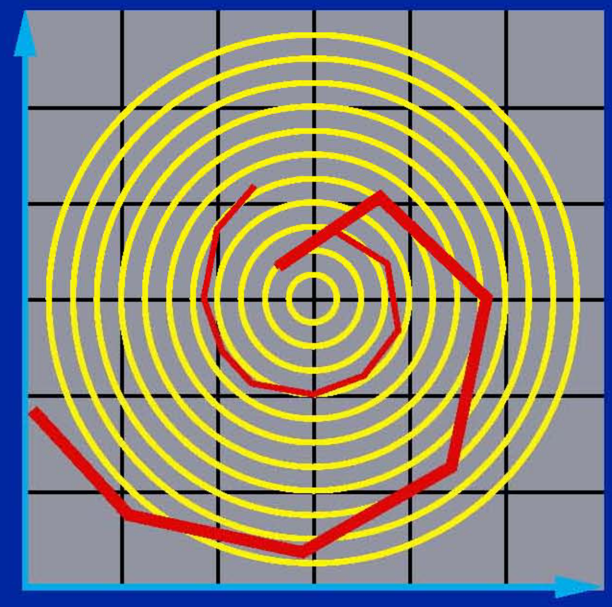
# Euler’s Method



- **Simplest numerical solution method**
- **Discrete time steps**
- **Bigger steps, bigger errors.**

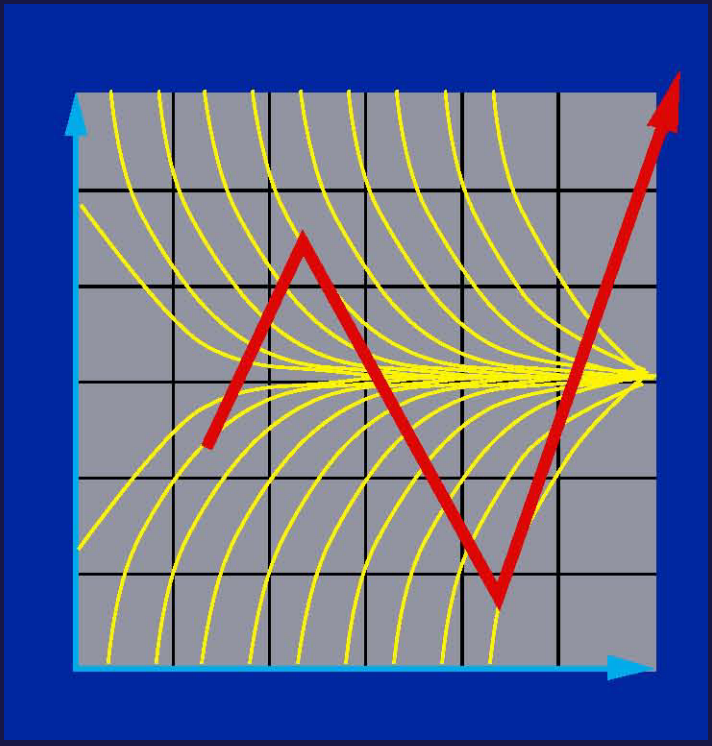
$$\mathbf{x}(t + \Delta t) = \mathbf{x}(t) + \Delta t\, f(\mathbf{x},t)$$

# Problem 1: Inaccuracy

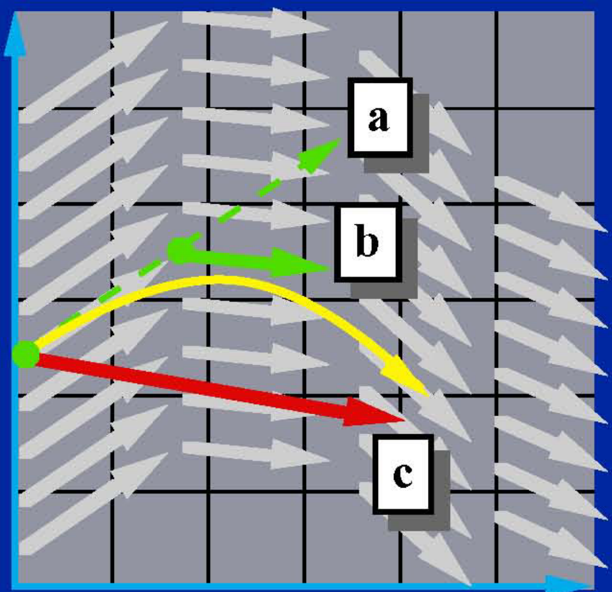


**Error turns x(t) from a circle into the spiral of your choice.**

# Problem 2: Instability



# The Midpoint Method




**a. Compute an Euler step**

$$\Delta \mathbf{x} = \Delta t\ \mathbf{f}(\mathbf{x}, t)$$

**b. Evaluate f at the midpoint**

$$\mathbf{f}_{mid} = \mathbf{f}\left(\frac{x + \Delta x}{2}, \frac{t + \Delta t}{2}\right)$$

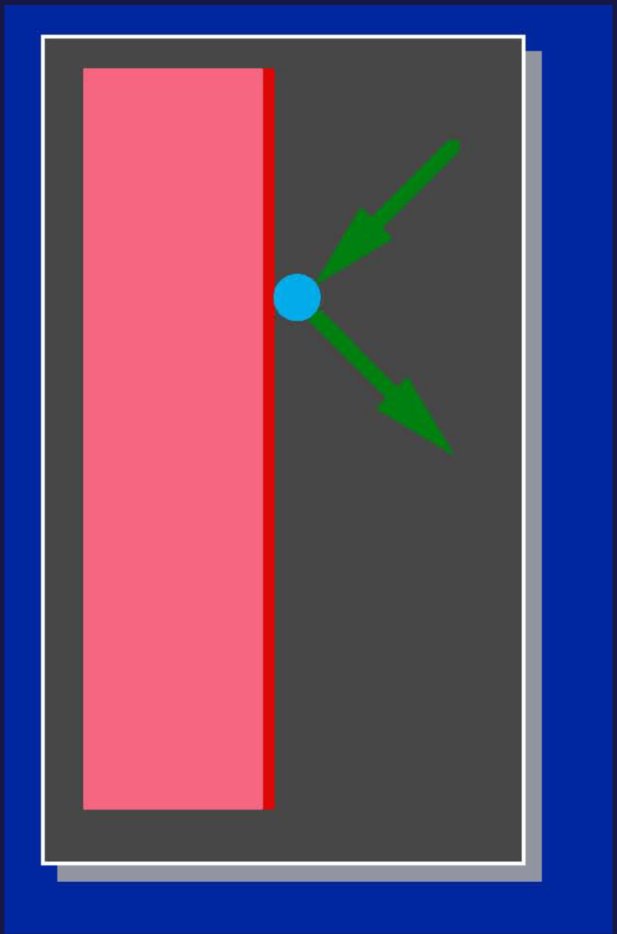
**c. Take a step using the midpoint value**

$$\mathbf{x}(t + \Delta t) = \mathbf{x}(t) + \Delta t\ \mathbf{f}_{mid}$$

# More Methods

- Euler's method is 1st order
- The midpoint method is 2nd order
- Just the tip of the iceberg – see Numerical Recipes for more
- Helpful hints (from Witkin/Baraff course)
  - Don't use Euler's method (you will anyway)
  - Use an adaptive time step

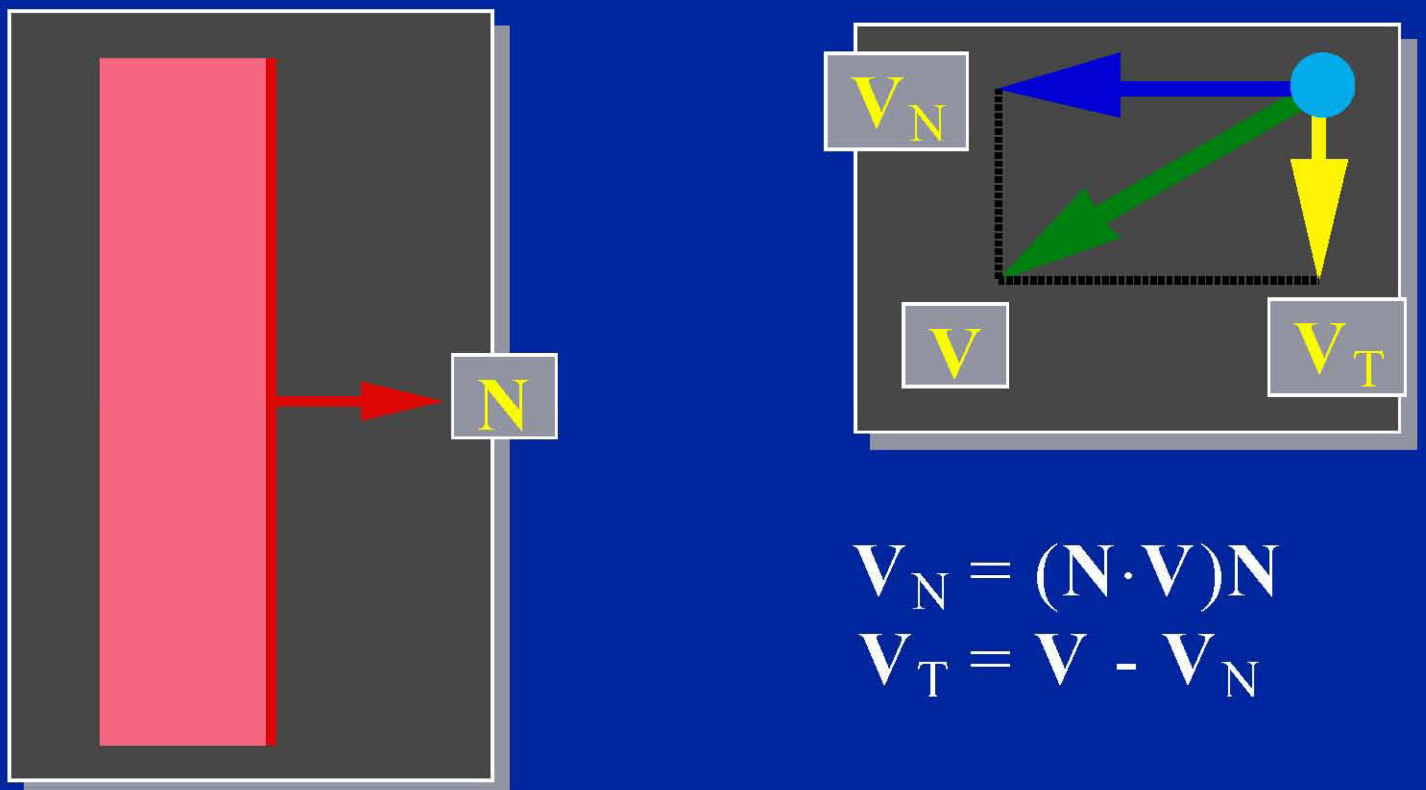
# Simple Collisions



Later: rigid body collision and contact

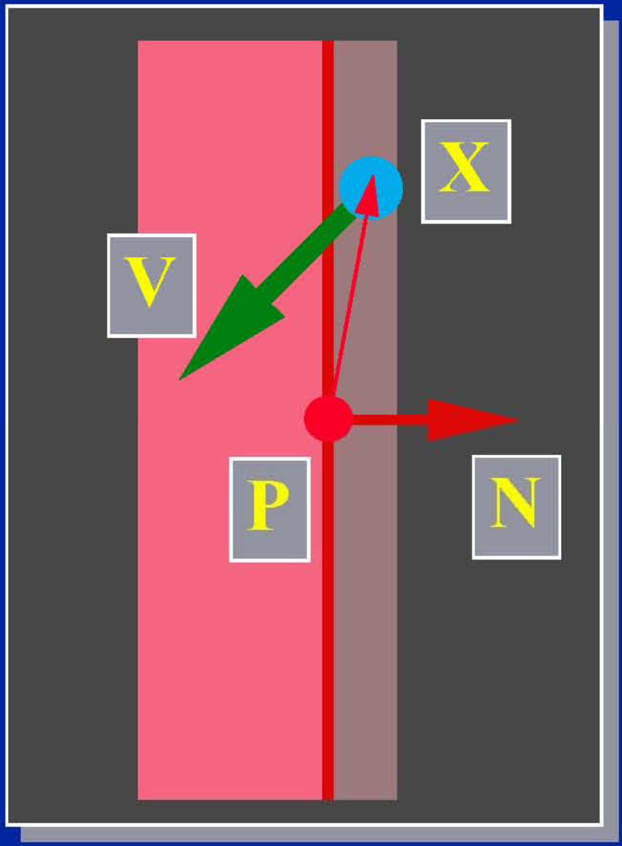
For now, just simple point-plane collisions

# Normal and Tangential Components




$$V_N = (N \cdot V)N$$
$$V_T = V - V_N$$

# Collision Detection

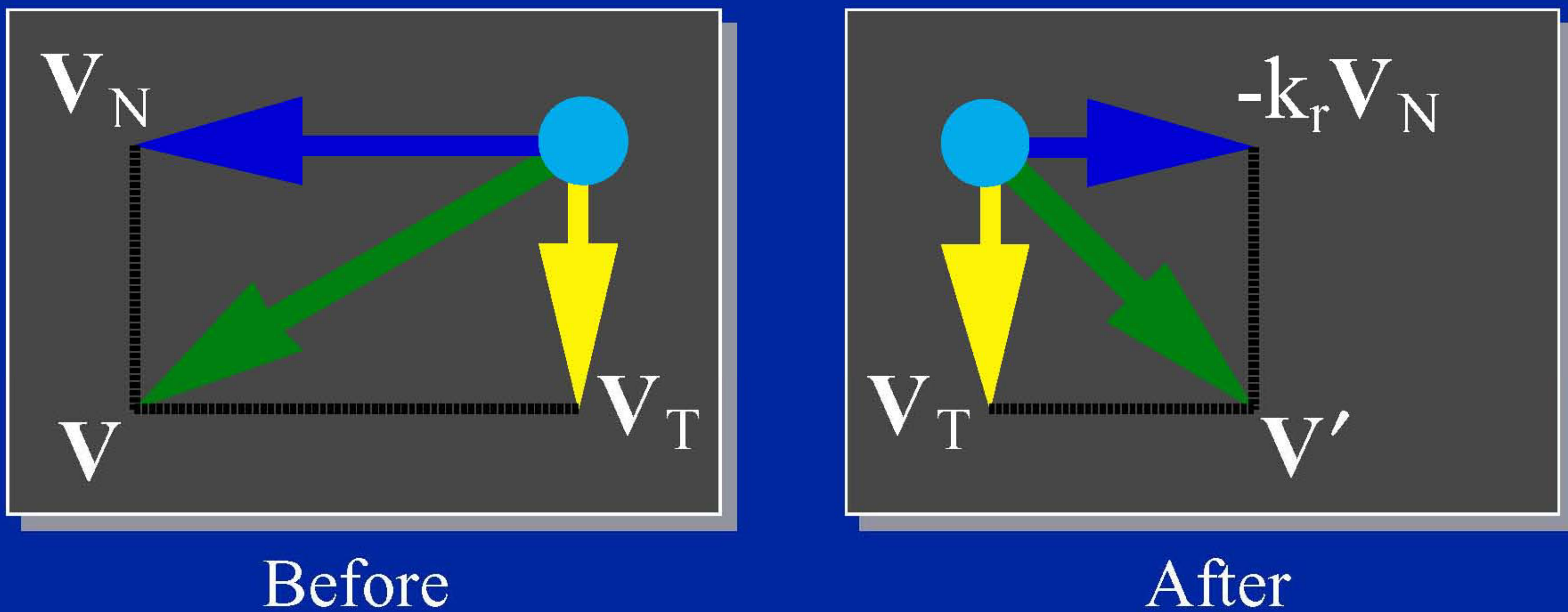



$$(\mathbf{X} - \mathbf{P}) \cdot \mathbf{N} < \varepsilon$$

$$\mathbf{N} \cdot \mathbf{V} < 0$$

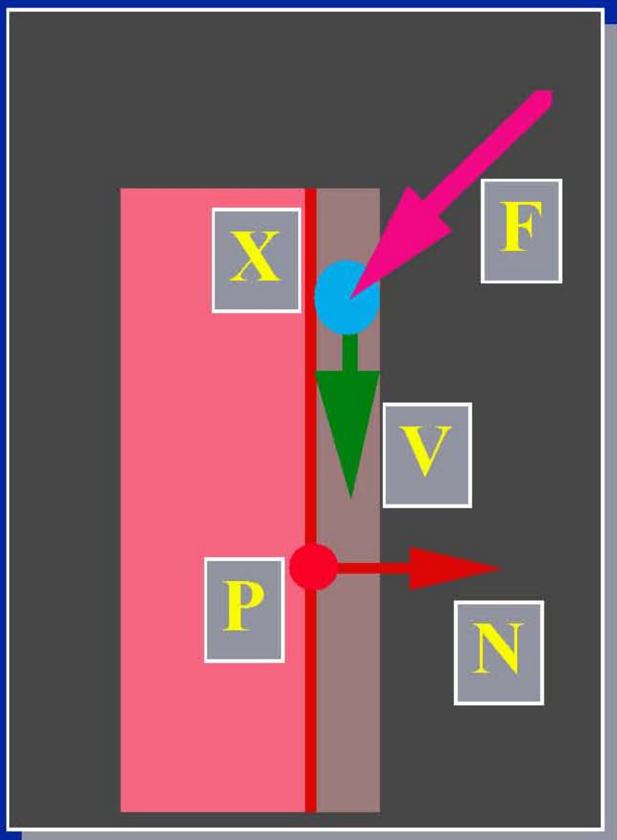
- Within ε of the wall.
- Heading in.

# Collision Response




$$V' = V_T - k_r V_N$$

# Conditions for Contact

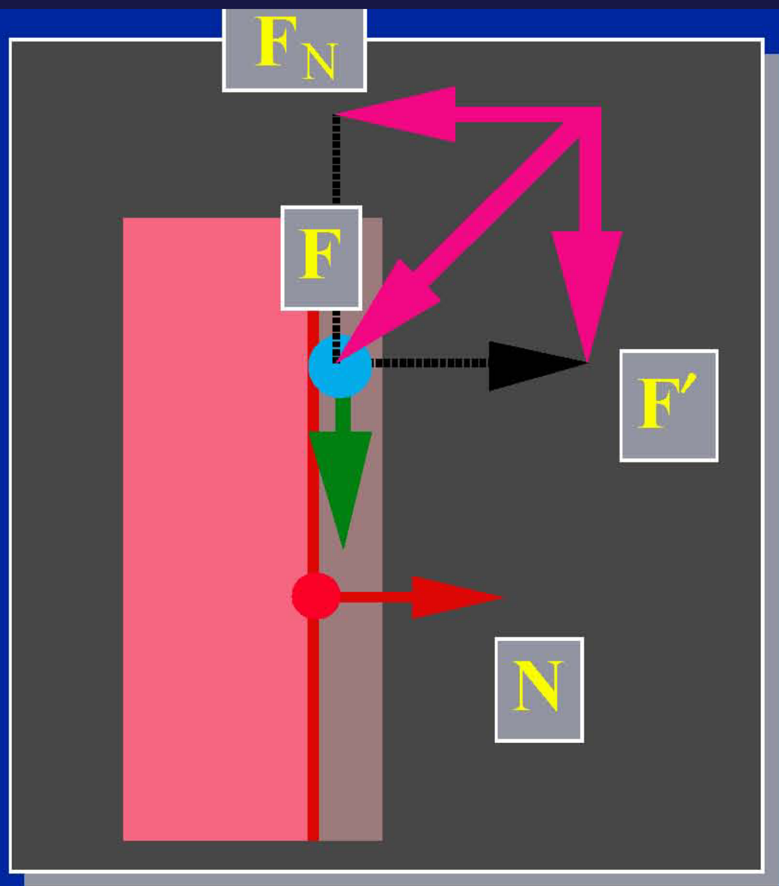



$$\left|(\mathbf{X} - \mathbf{P}) \cdot \mathbf{N}\right| < \varepsilon$$

$$\left|\mathbf{N} \cdot \mathbf{V}\right| < \varepsilon$$

- On the wall
- Moving along the wall
- Pushing against the wall

# Contact Force




**$F' = F_T$**

**The wall pushes back, cancelling the normal component of F.**

**(An example of a *constraint force.*)**

# What did we skip?

- Equations of motion for rigid bodies
- Collision detection of interesting shapes (not just points and planes)
- Controllers
  - Don’t just want ragdolls—not all characters that fall are dead, even in videogames!

# What's coming on Wednesday

- Collision detection
- Controllers
- Combining mocap and simulation
- User control of characters