DiffEQ 2

treuille@cs.cmu.edu

Particle Dynamics

from Zoran Popović

Overview

- One lousy particle
- Particle systems
- Forces: gravity, springs
- Implementation

Newtonian particle

- Differential equations: f=ma
- Forces depend on:
- Position, velocity, time

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

Second order equations

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

Has 2nd derivatives

$$\begin{aligned} \dot{x} &= v \\ \dot{v} &= \frac{f(x, \dot{x})}{m} \end{aligned}$$

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

Phase space

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

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

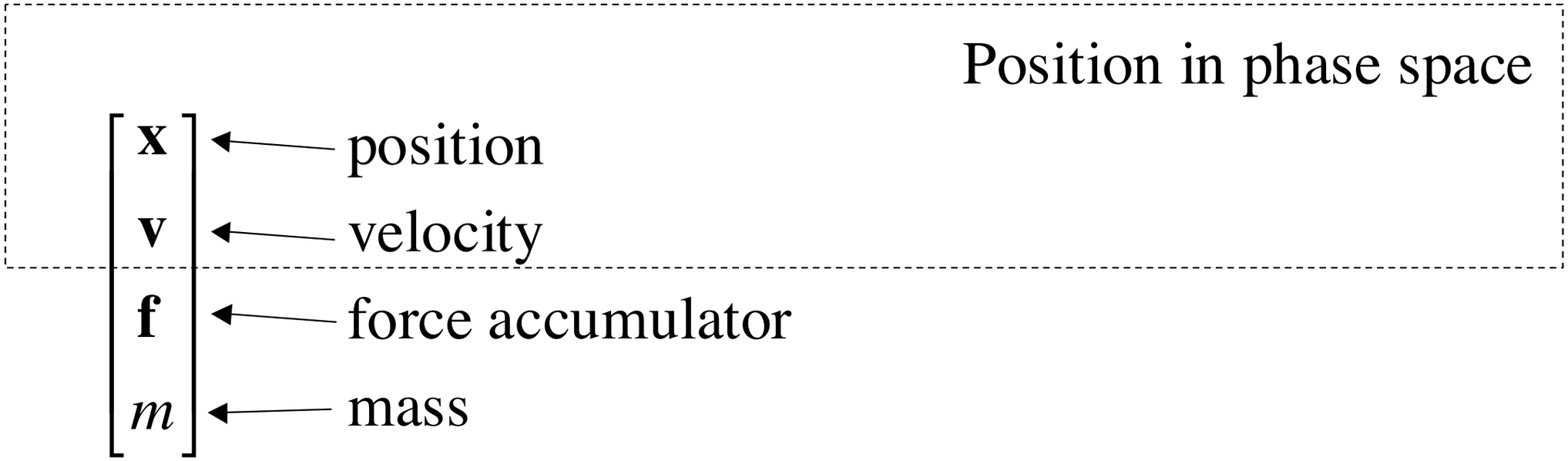
$$\begin{bmatrix} \dot{x} \\ \dot{v} \end{bmatrix}$$

Velocity on Phase space:
Another 6-vector

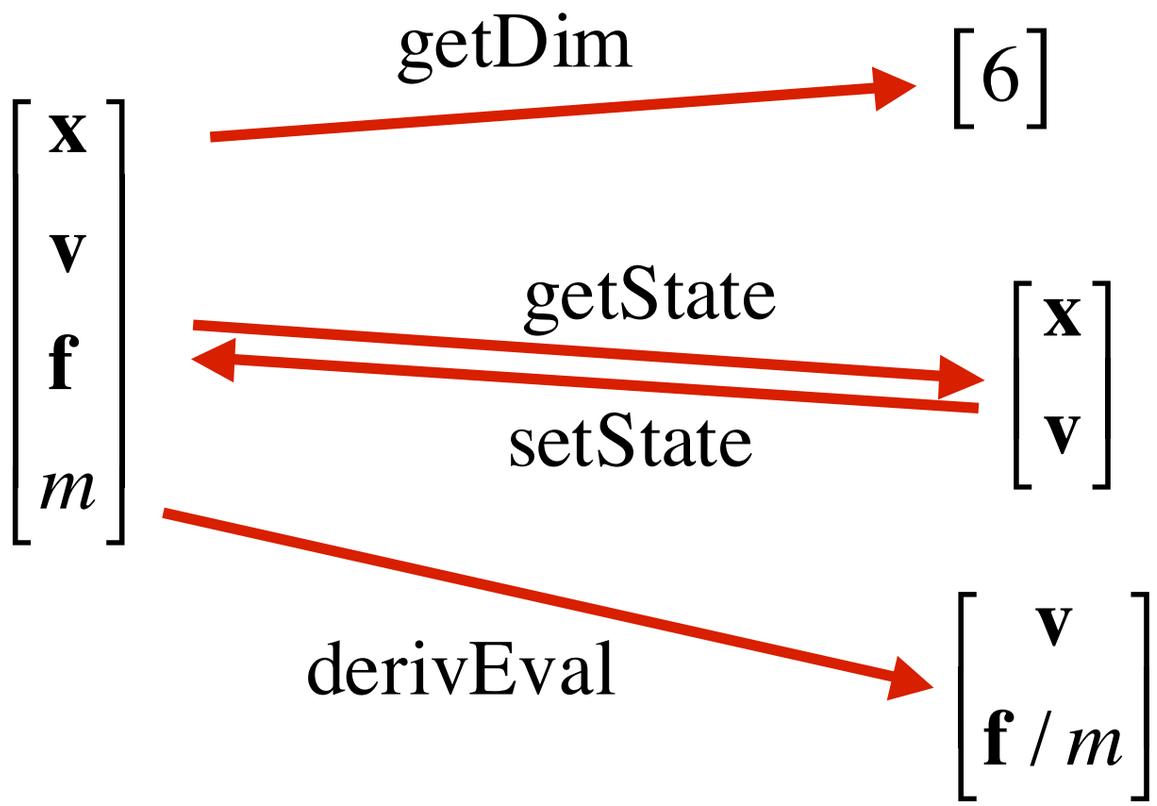
$$\begin{bmatrix} \dot{x} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} v \\ f / m \end{bmatrix}$$

A vanilla 1st-order differential equation

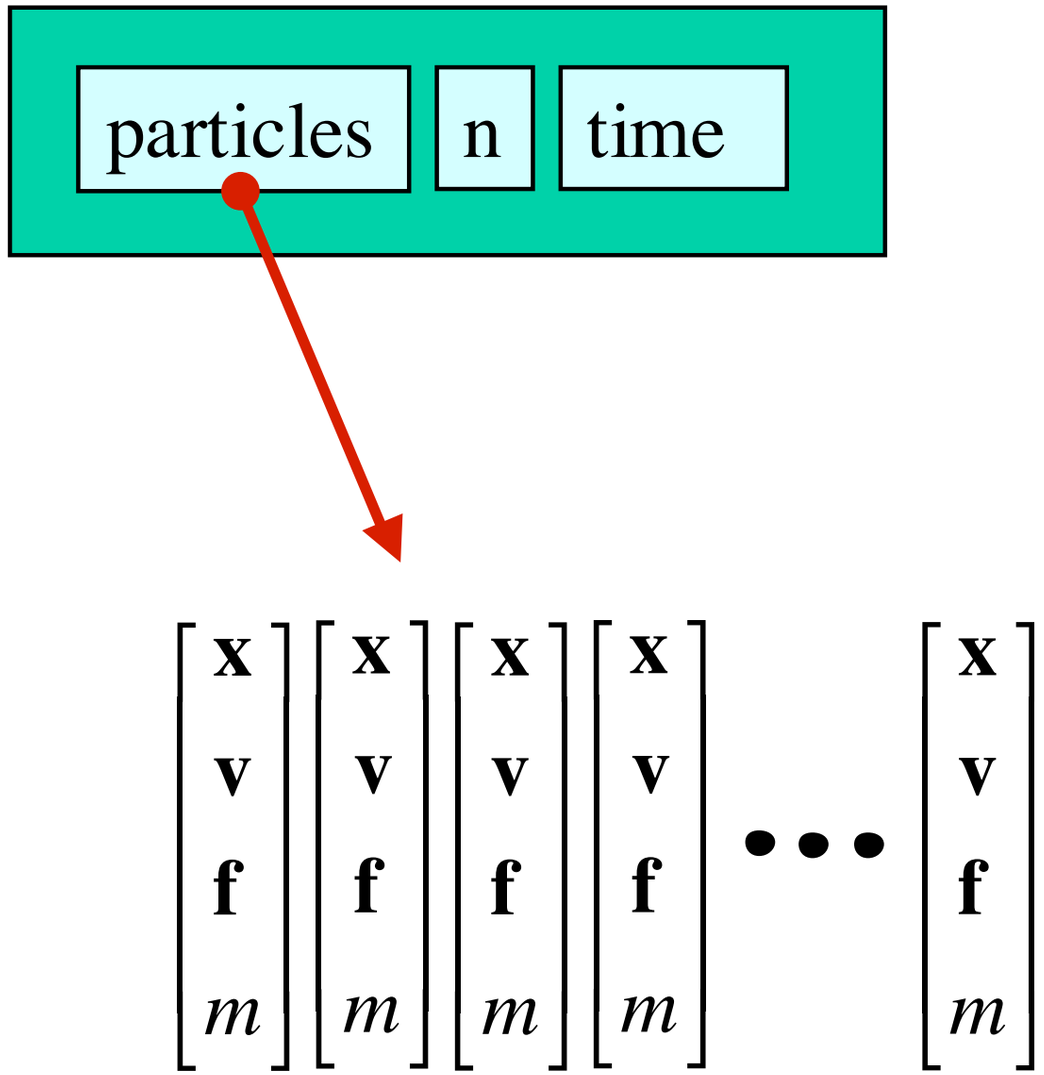
Particle structure



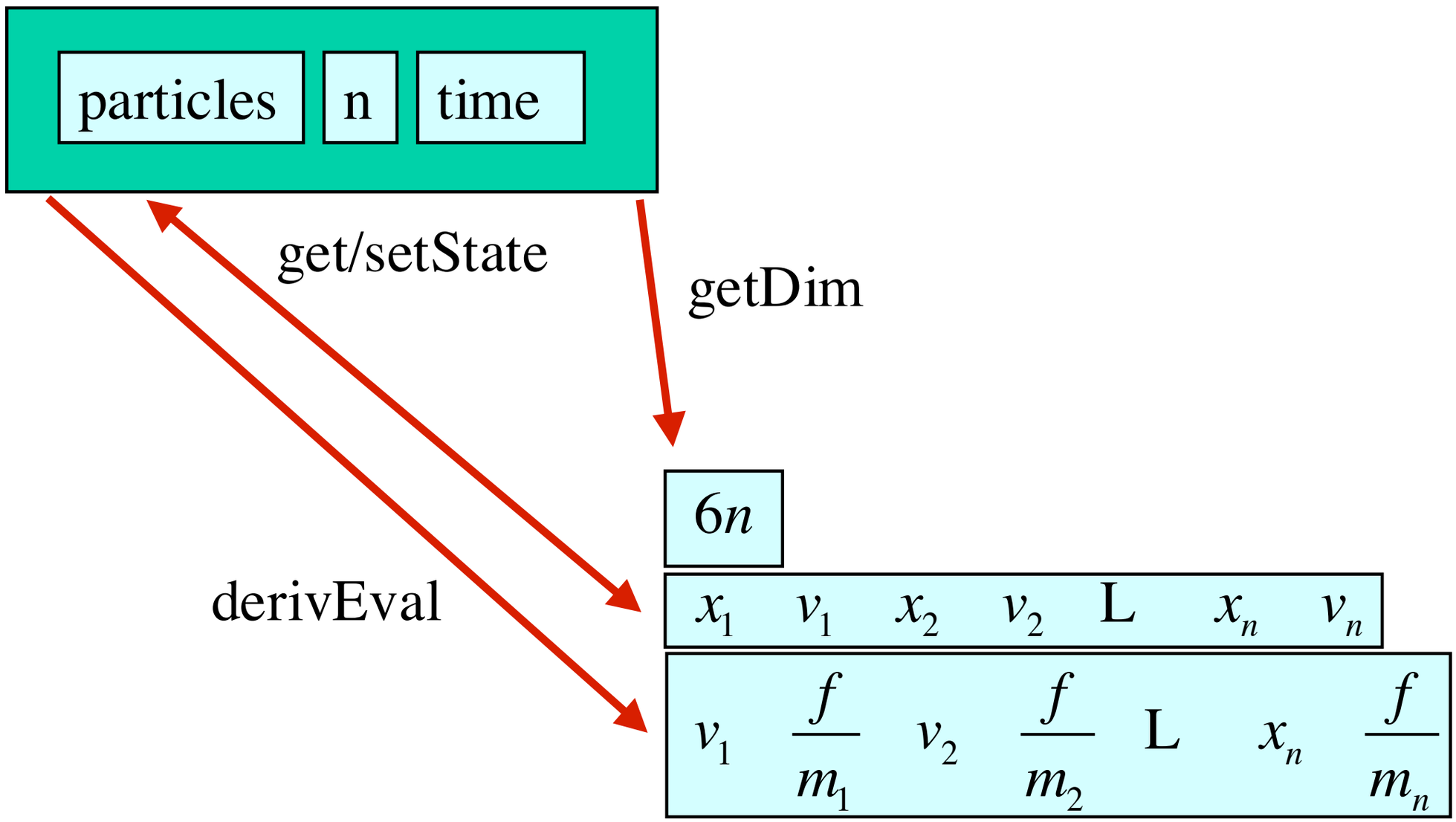
Solver interface



Particle systems



Solver interface



Differential equation solver

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

Euler method: $x(t+h) = x(t) + h \cdot \dot{x}(t)$

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \mathrm{V}t \cdot \dot{x}$$

$$\mathbf{v}_{i+1} = \mathbf{v}_i + \mathrm{V}t \cdot \dot{v}$$

Gets very unstable for large $\mathrm{V}t$

Higher order solvers perform better: (e.g. Runge-Kutta)

derivEval loop

1. Clear forces
   - Loop over particles, zero force accumulators
2. Calculate forces
   - Sum all forces into accumulators
3. Gather
   - Loop over particles, copying v and f/m into destination array

Forces

- Constant (gravity)
- Position/time dependent (force fields)
- Velocity-dependent (drag)
- N-ary (springs)

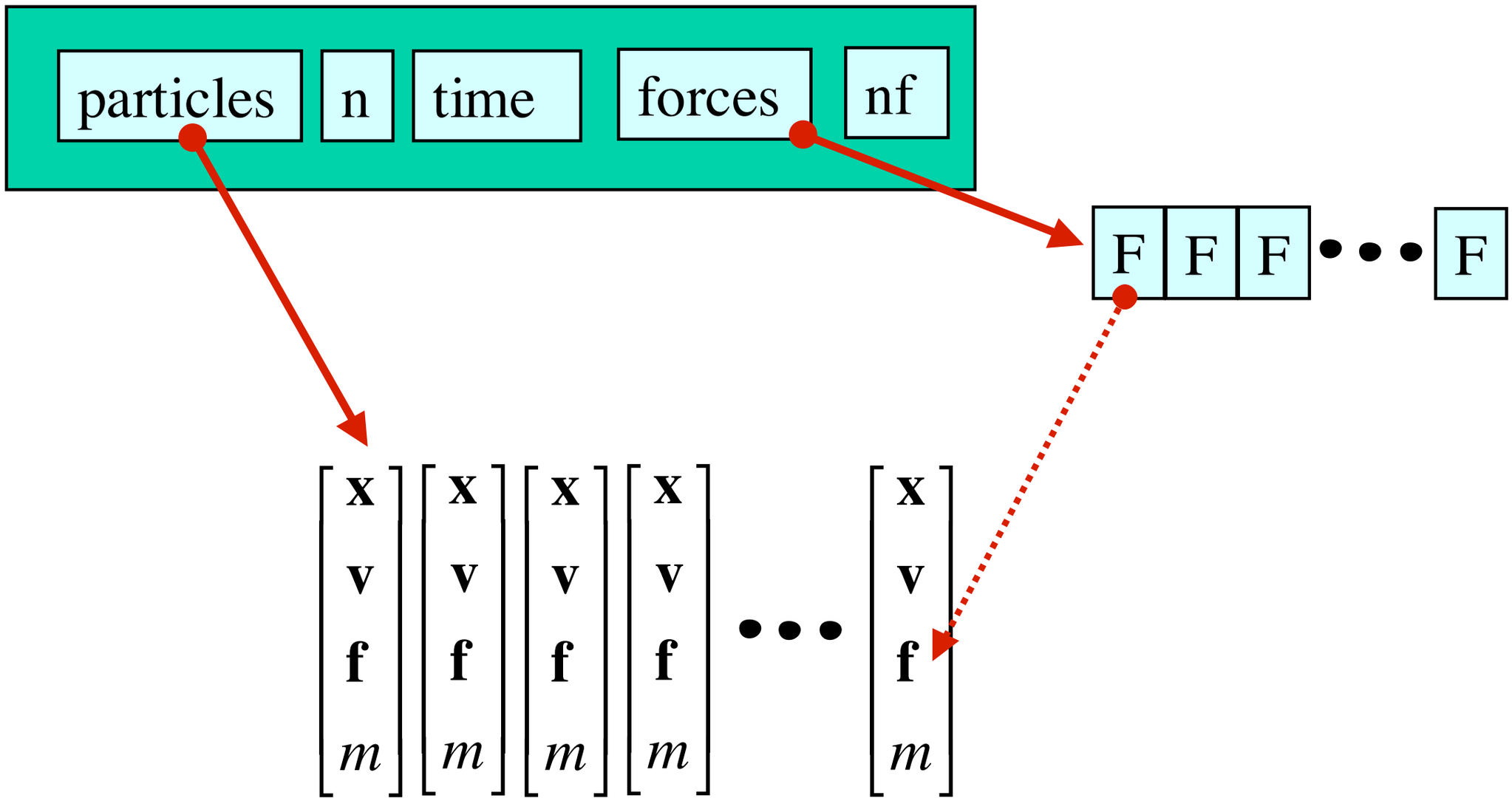
Force structures

Force objects are black boxes that point to the particles they influence, and add in their contribution into the force accumulator.

Global force calculation:

- Loop, invoking force objects

Particle systems with forces



Gravity

Force law:

$$\mathbf{f}_{grav} = m\mathbf{G}$$

```
p->f += p->m * F->G
```

Viscous drag

Force law:

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

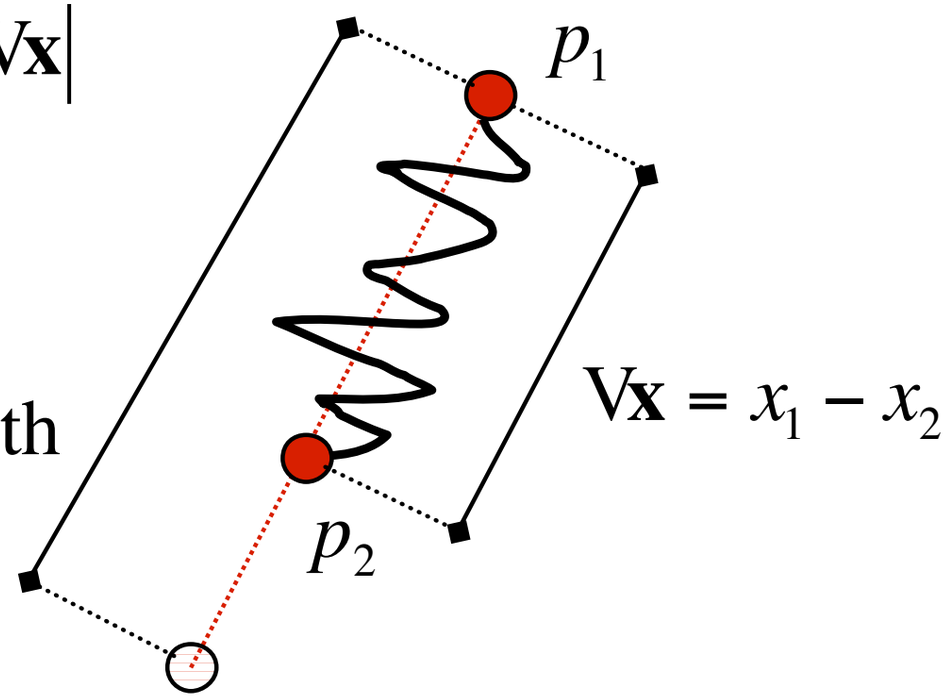
```
p->f -= F->k * p->v
```

Damped spring

Force law:

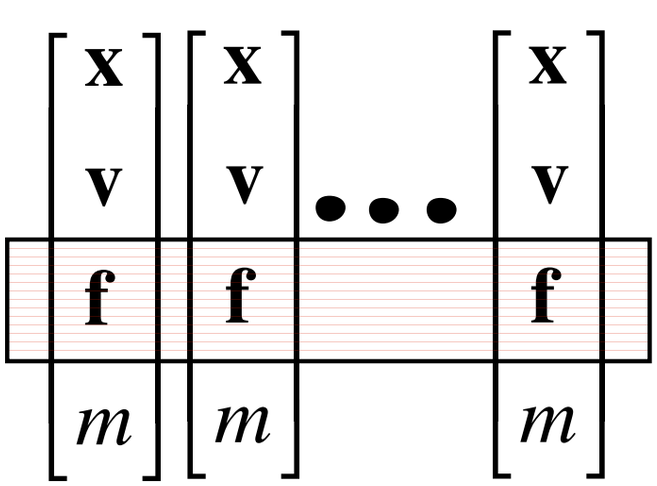
$$\mathbf{f}_1 = -\left[k_s(|\mathbf{Vx}| - \mathbf{r}) + k_d\left(\frac{\mathbf{VvVx}}{|\mathbf{Vx}|}\right)\right]\frac{\mathbf{Vx}}{|\mathbf{Vx}|}$$

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



$\mathbf{r}$ = rest length

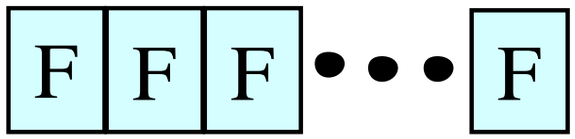
derivEval Loop



Clear force accumulators

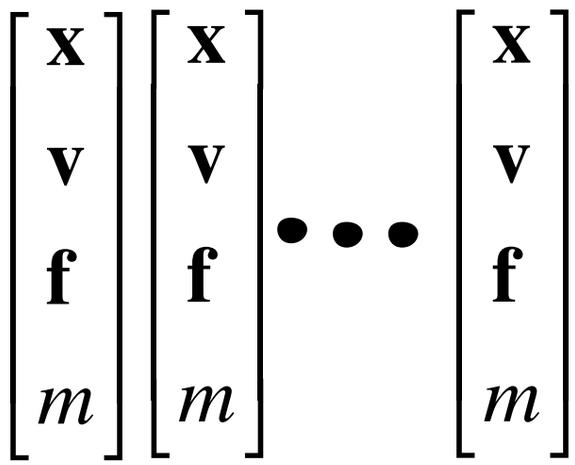
1

2



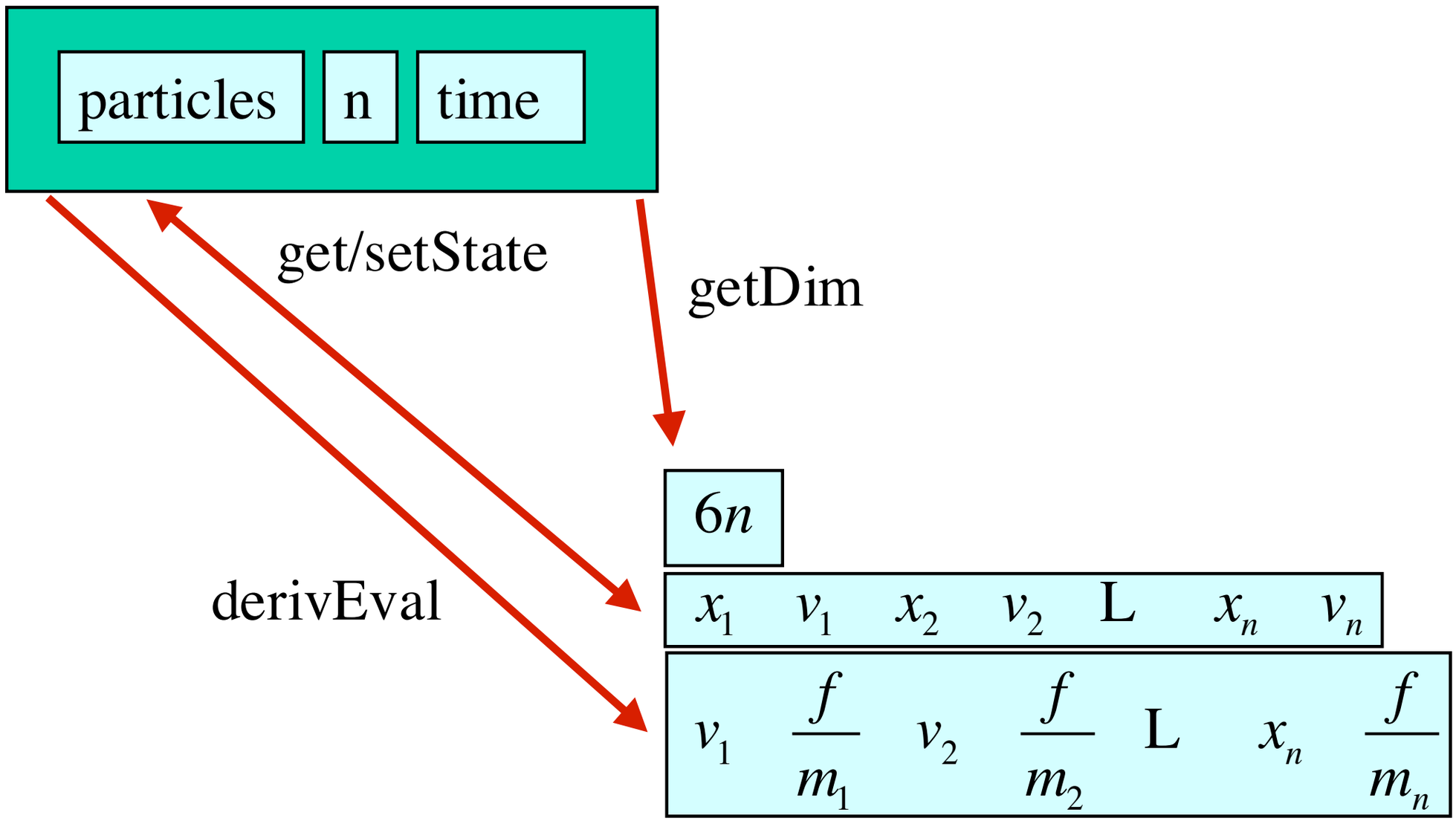
Apply forces
to particles

3



Return [v,f/m,…]
to solver

Solver interface



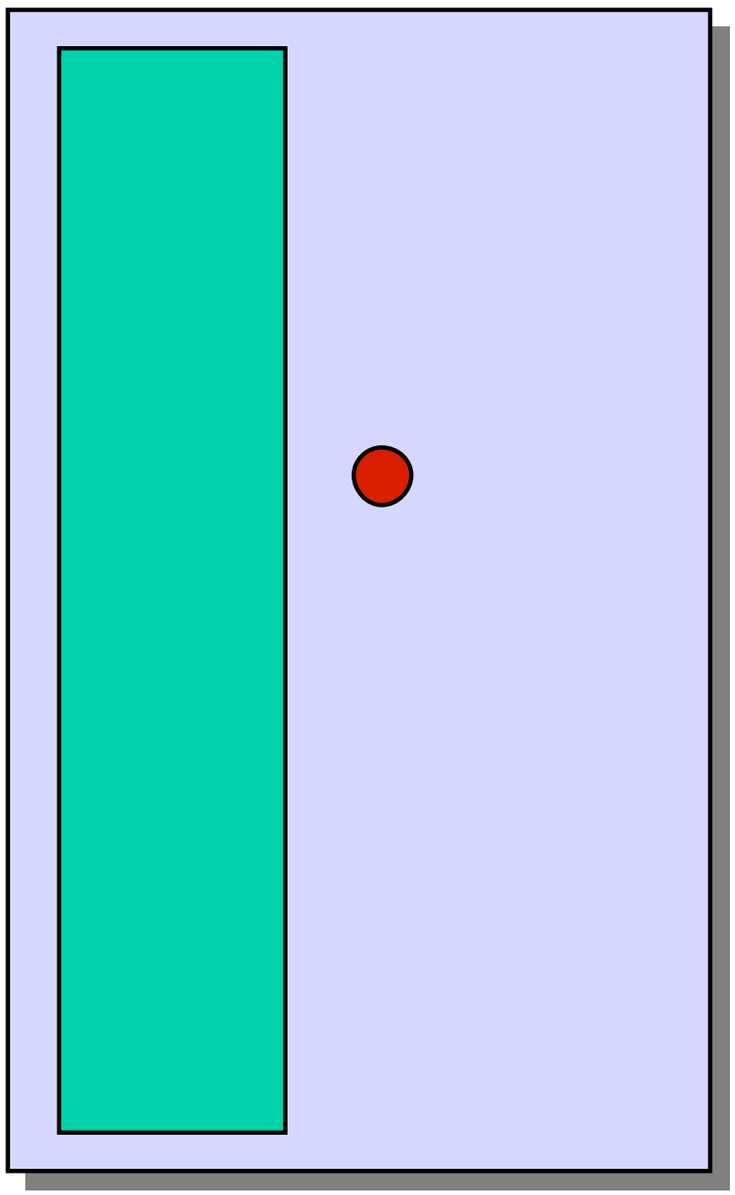
Differential equation solver

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

Euler method:

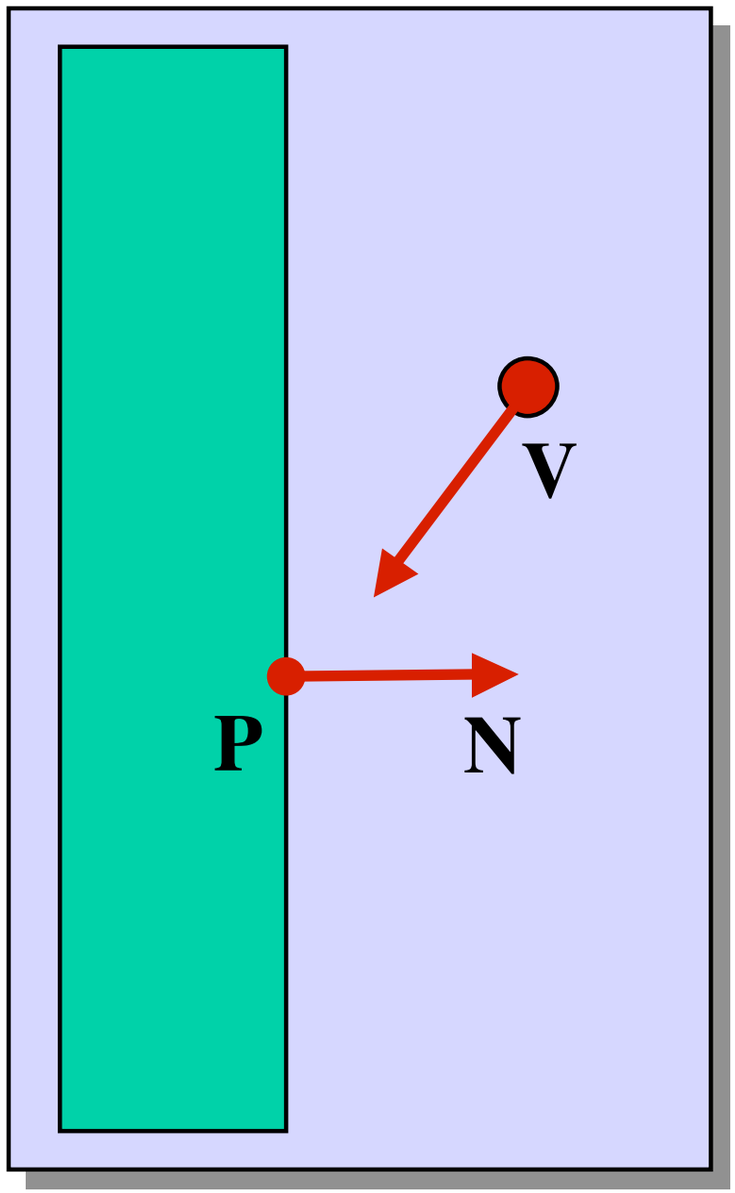
$$\begin{bmatrix} x_1^{\,i+1} \\ v_1^{\,i+1} \\ \mathrm{M} \\ x_n^{\,i+1} \\ v_n^{\,i+1} \end{bmatrix} = \begin{bmatrix} x_1^{\,i} \\ v_1^{\,i} \\ \mathrm{M} \\ x_n^{\,i} \\ v_n^{\,i} \end{bmatrix} + \mathrm{V}t \begin{bmatrix} v_1^{\,i} \\ f_1^{\,i} / m_1 \\ \mathrm{M} \\ v_n^{\,i} \\ f_n^{\,i} / m_n \end{bmatrix}$$

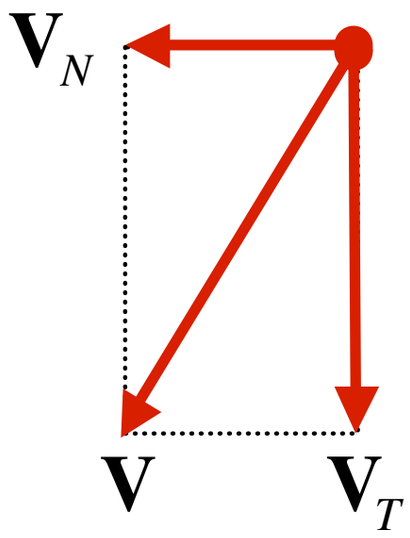
Bouncing off the walls



- Add-on for a particle simulator
- For now, just simple point-plane collisions

Normal and tangential components

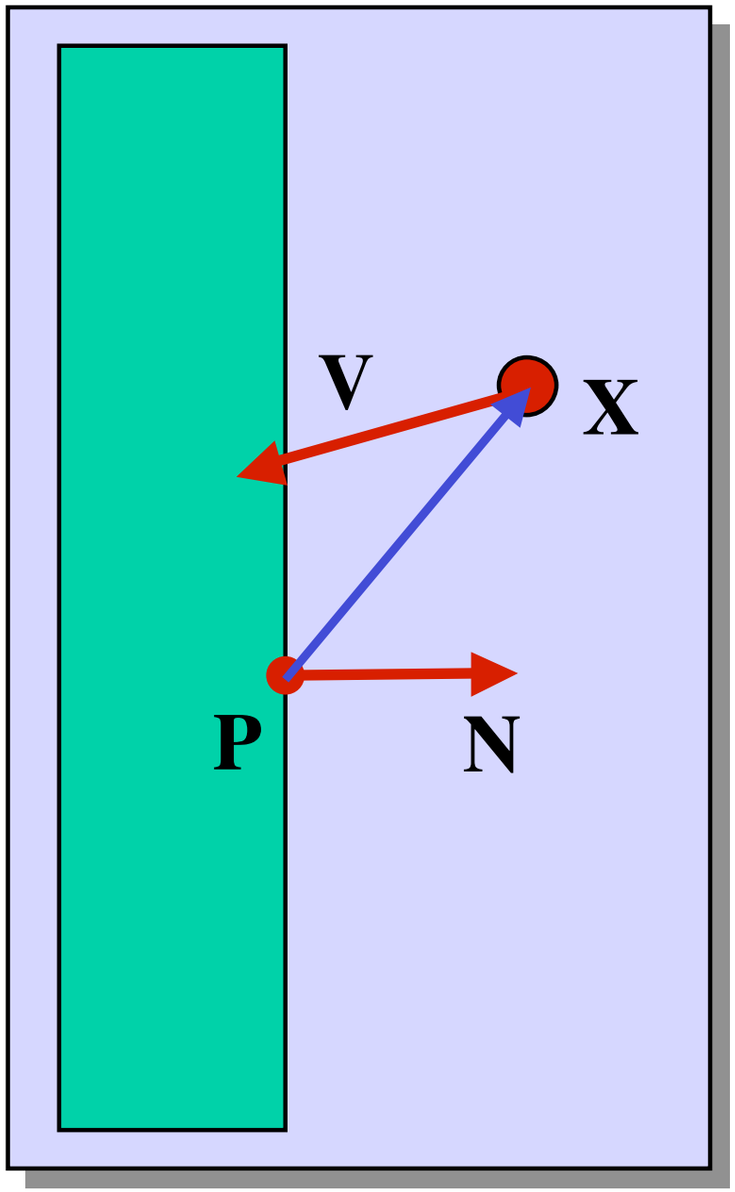




$$V_N = (N \cdot V)N$$

$$V_T = V - V_N$$

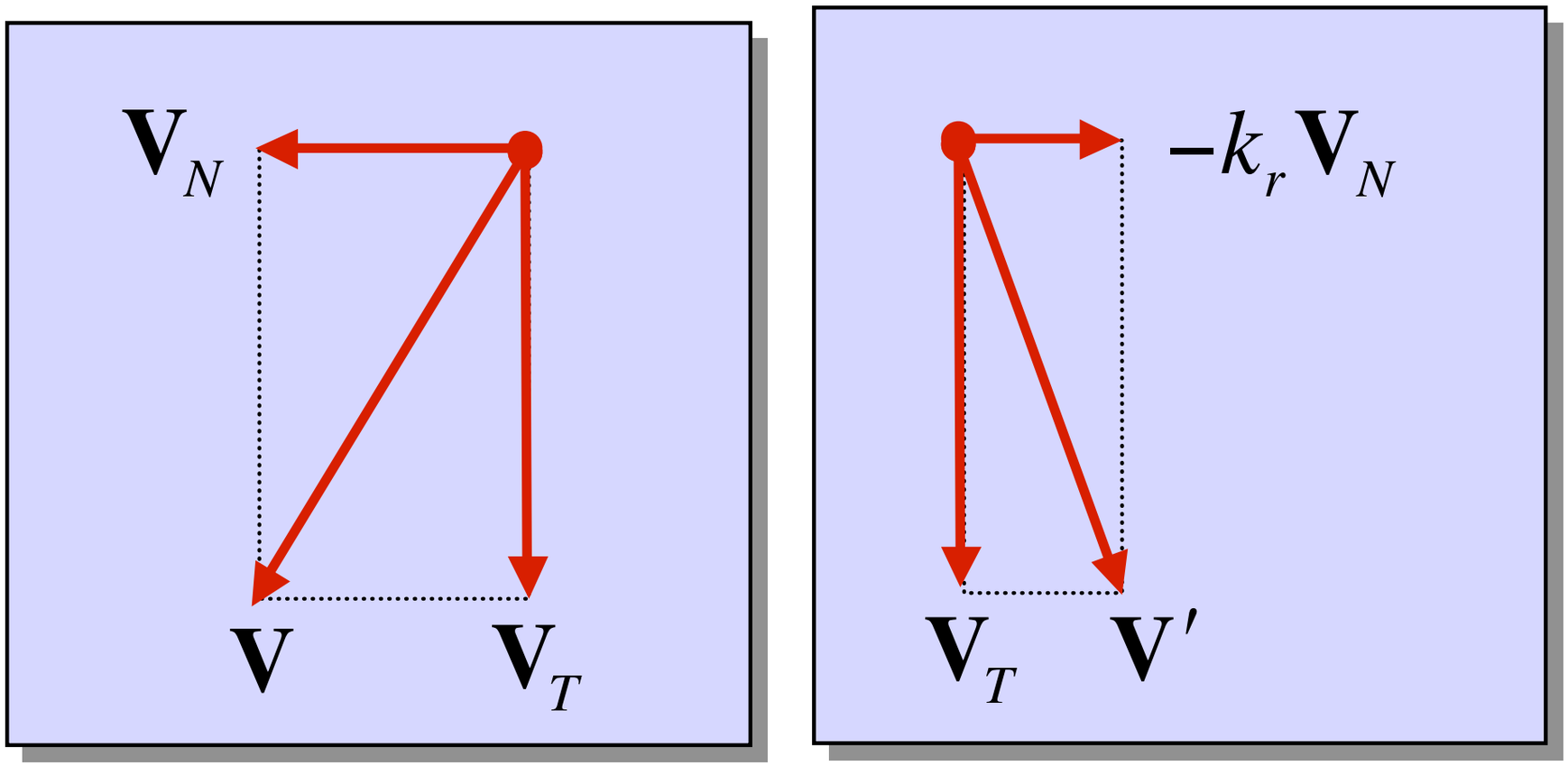
Collision Detection



$(\mathbf{X}-\mathbf{P})\cdot\mathbf{N} < \varepsilon$ Within e of the wall

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

Collision Response



before after

$$\mathbf{V}' = \mathbf{V}_T - k_r\mathbf{V}_N$$

Summary

- Physics of a particle system
- Various forces acting on a particle
- Combining particles into a particle system
- Euler method for solving differential equations

Implicit Integration

Implicit Methods

David Baraff

PIXAR
ANIMATION STUDIOS

“Give me Stability
or
Give me Death”

— Baraff’s other motto

stability is all stability is all stability is all

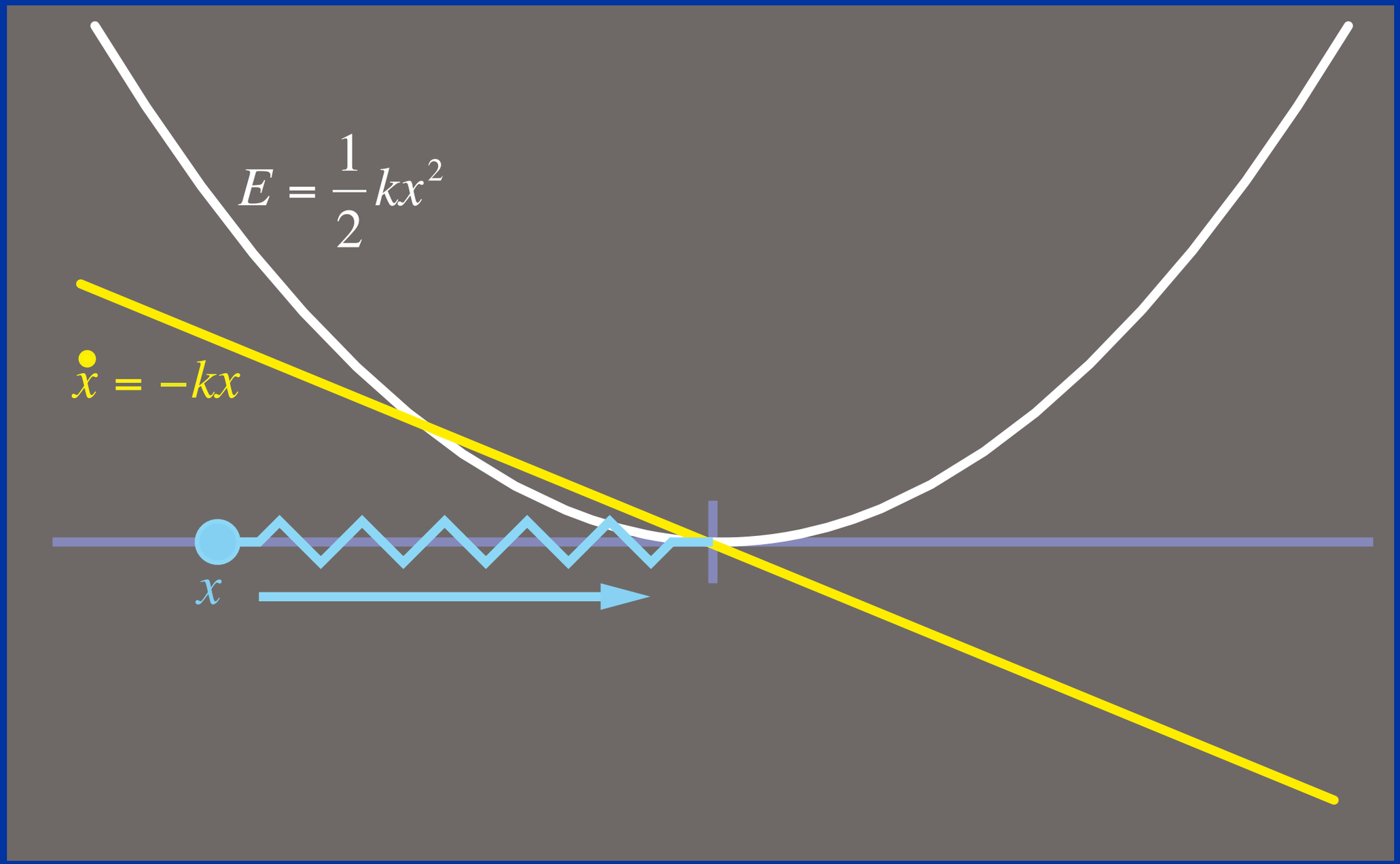
- If your step size is too big, your simulation blows up. It isn't pretty.
- Sometimes you have to make the step size so small that you never get anyplace.
- Nasty cases: cloth, constrained systems.

stability is all stability is all stability is all

- If your step size is too big, your simulation blows up. It isn't pretty.
- Sometimes you have to make the step size so small that you never get anyplace.
- Nasty cases: cloth, constrained systems.
- Solutions:
  - –Now: use explosion-resistant methods.
  - –Later: reformulate the problem.

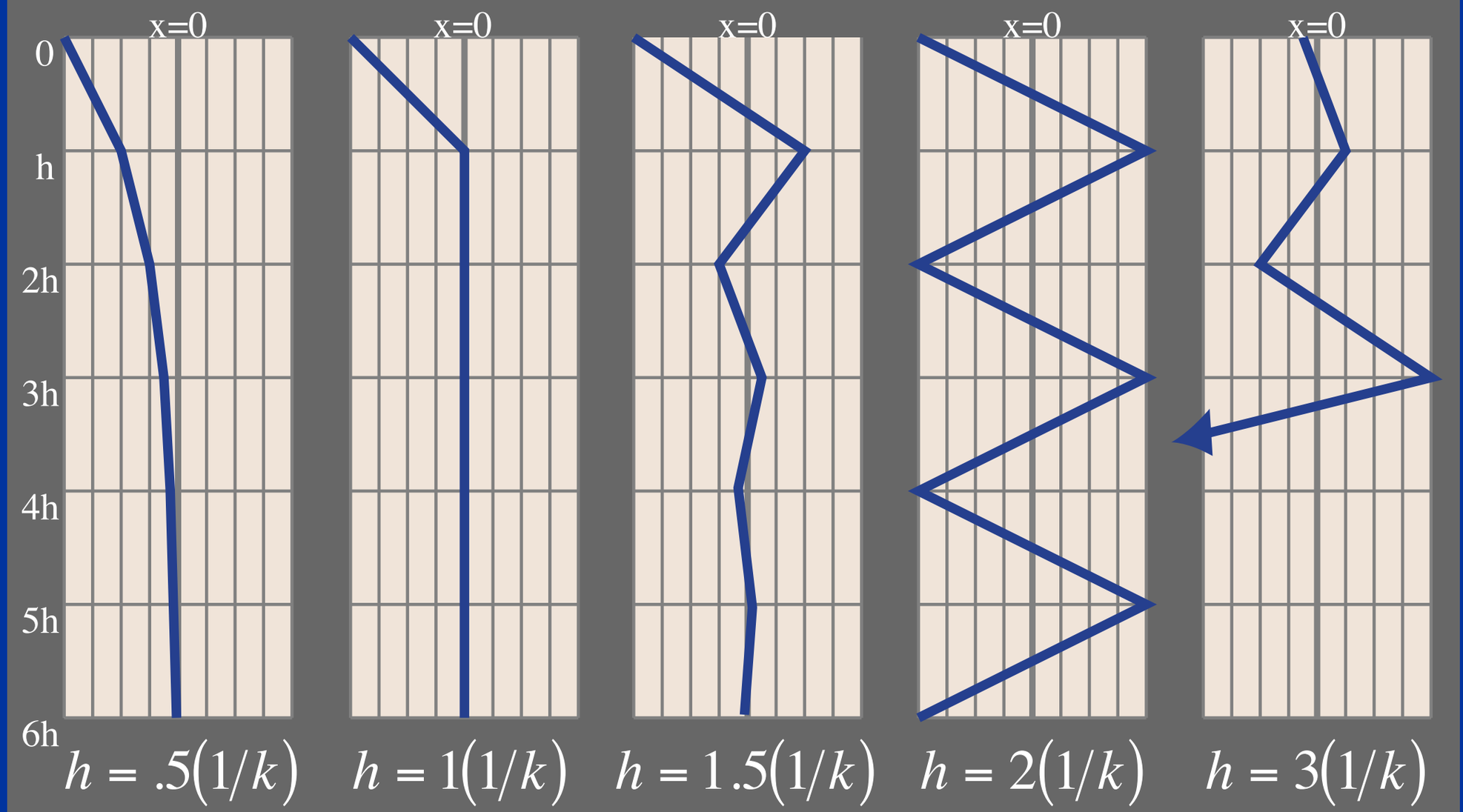
A very simple equation

A 1-D particle governed by $\dot{x} = -kx$ where k is a stiffness constant.



Euler's method has a speed limit

$$x = -kx \qquad \Delta x = -hkx$$

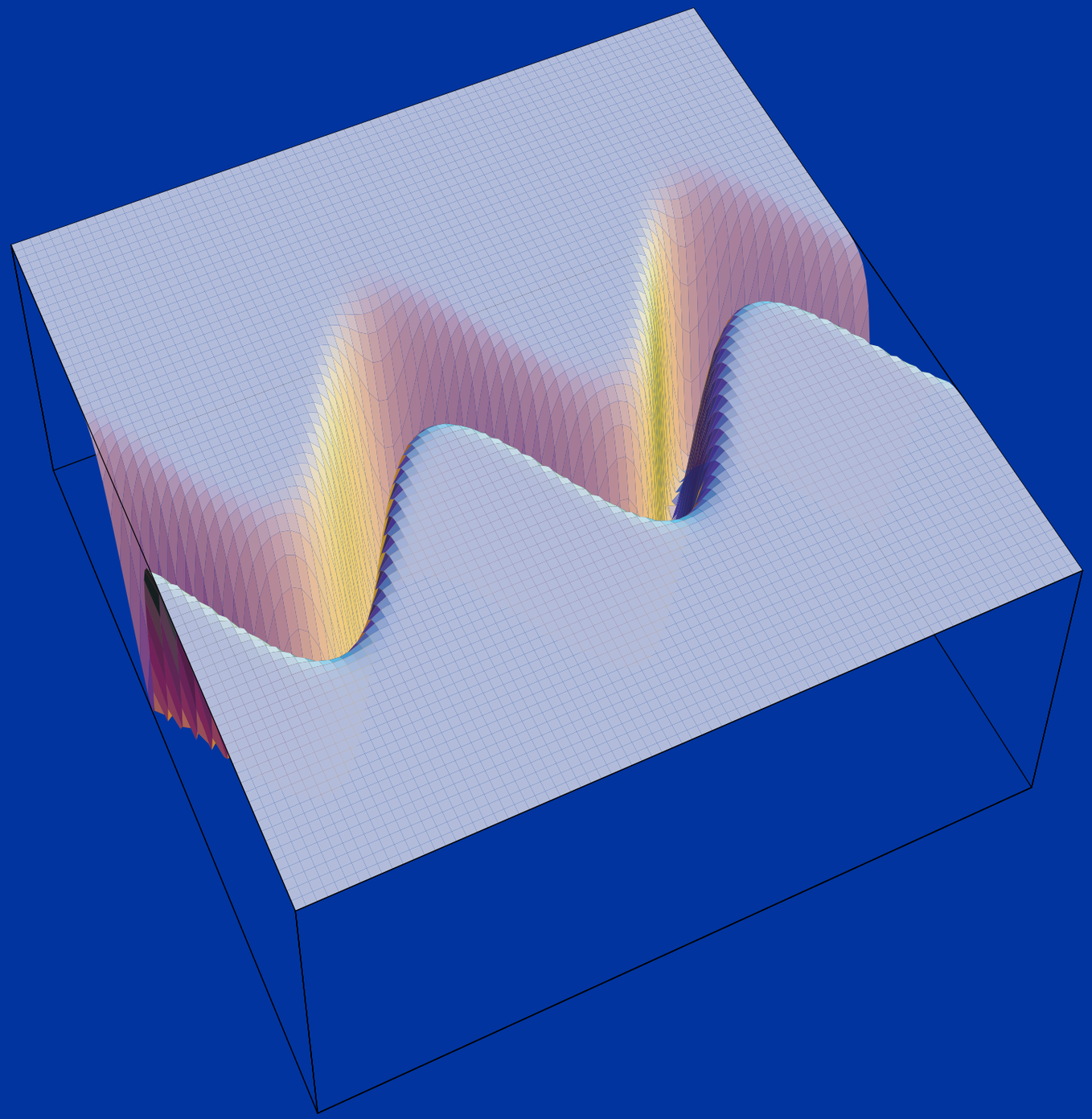


$h > 1/k$: oscillate. $h > 2/k$: explode!

Stiff Equations

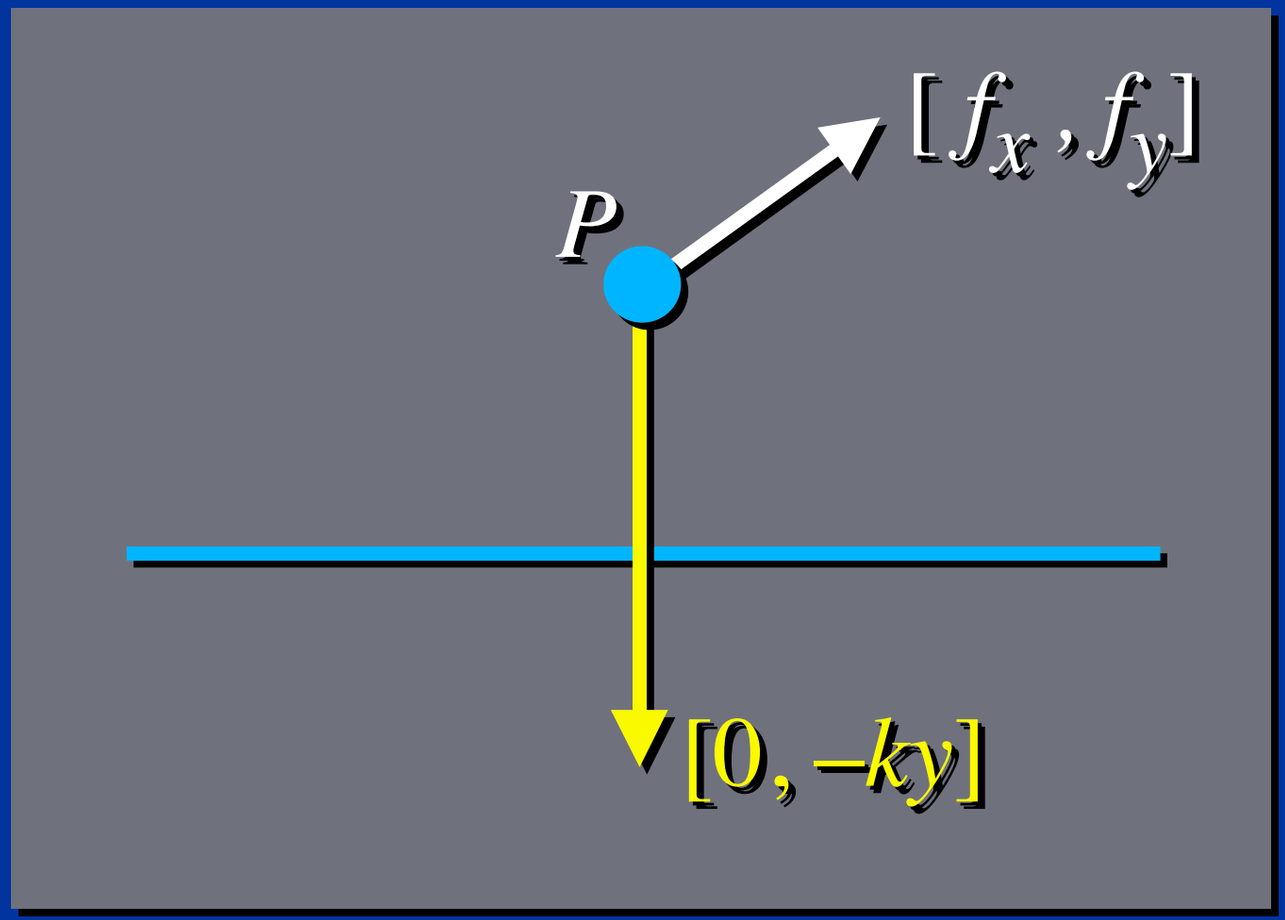
- In more complex systems, step size is limited by the largest k. One stiff spring can screw it up for everyone else.
- Systems that have some big k's mixed in are called stiff systems.

A Stiff Energy Landscape



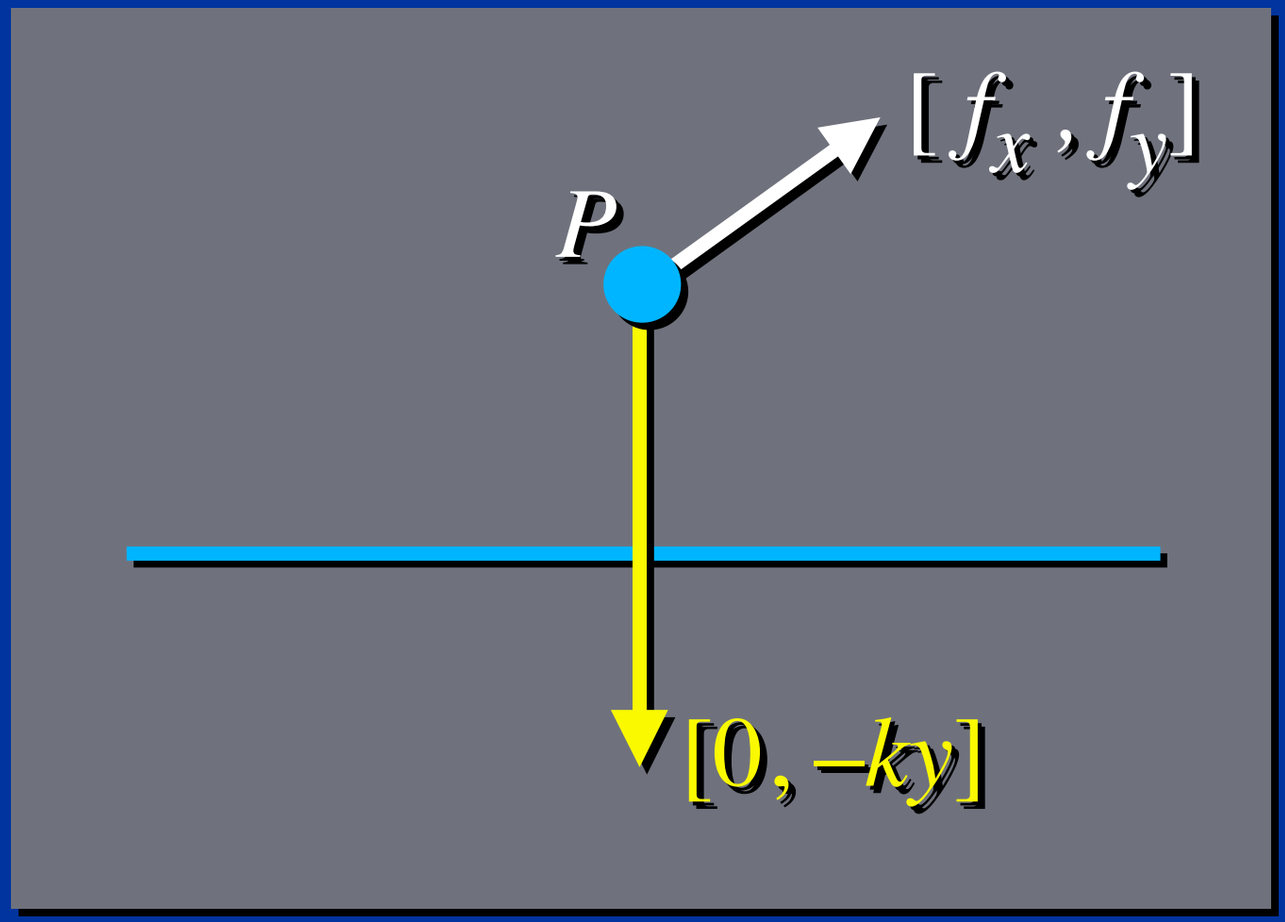
Example: particle-on-line

- A particle P in the plane.
- Interactive "dragging" force $[f_x, f_y]$.
- A penalty force $[0, -ky]$ tries to keep P on the x-axis.



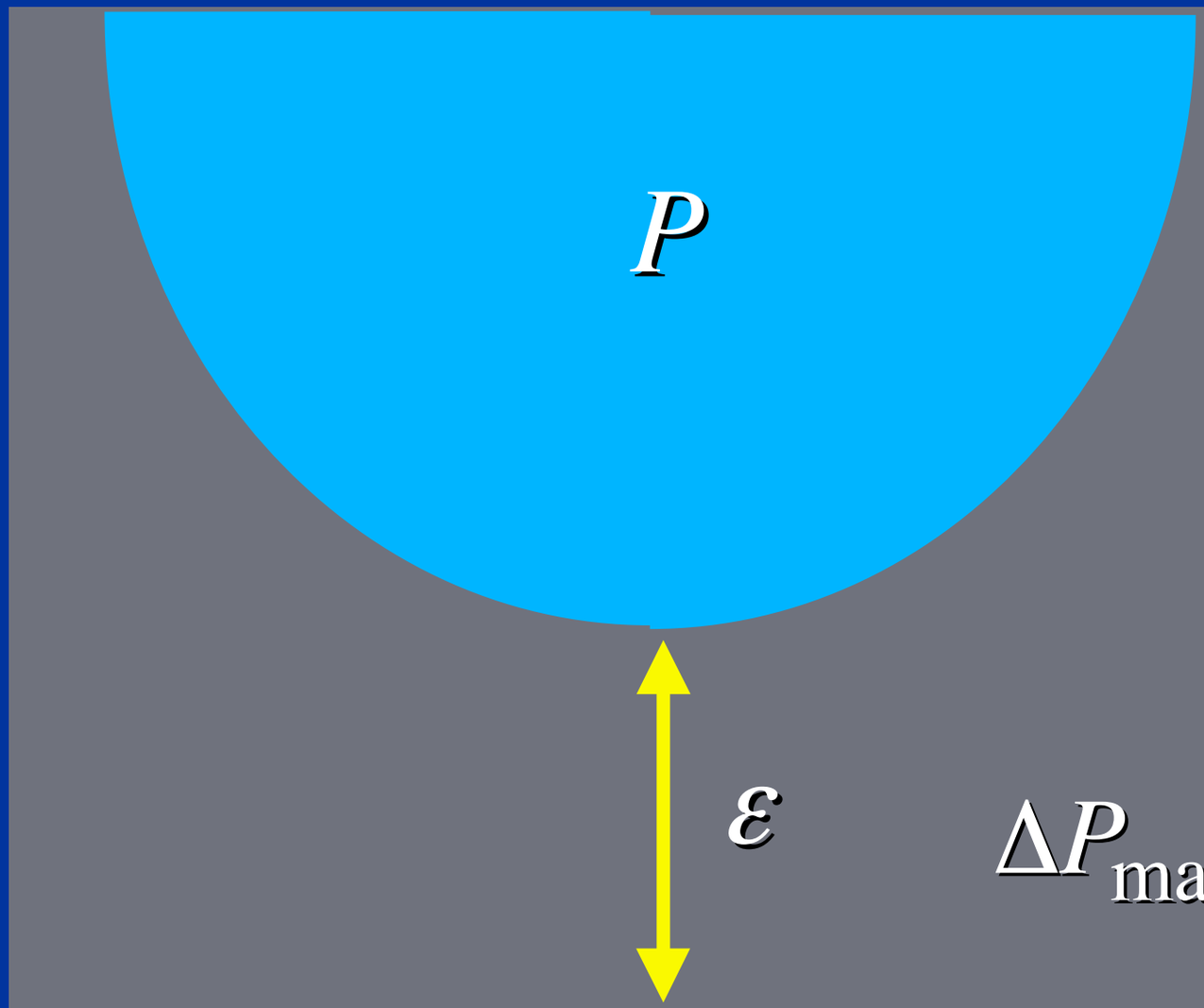
Example: particle-on-line

- A particle P in the plane.
- Interactive "dragging" force $[f_x, f_y]$.
- A penalty force $[0, -ky]$ tries to keep P on the x-axis.



- Suppose you want P to stay within a miniscule ε of the x-axis when you try to pull it off with a huge force $f_{\max}$.
- How big does k have to be? How *small* must h be?

Really big k. Really small h.

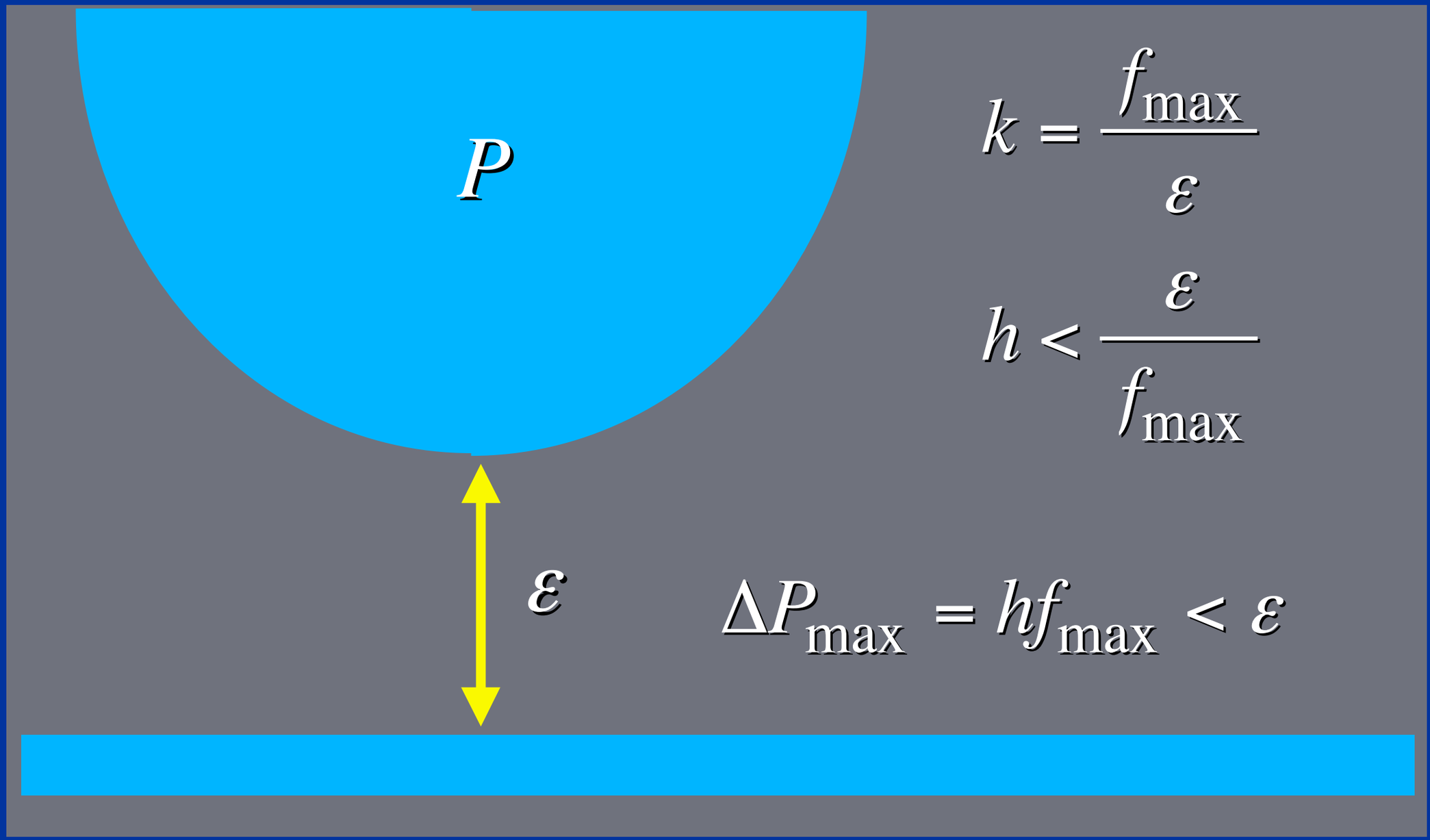


$$k = \frac{f_{\max}}{\varepsilon}$$

$$h < \frac{\varepsilon}{f_{\max}}$$

$$\Delta P_{\max} = h f_{\max} < \varepsilon$$

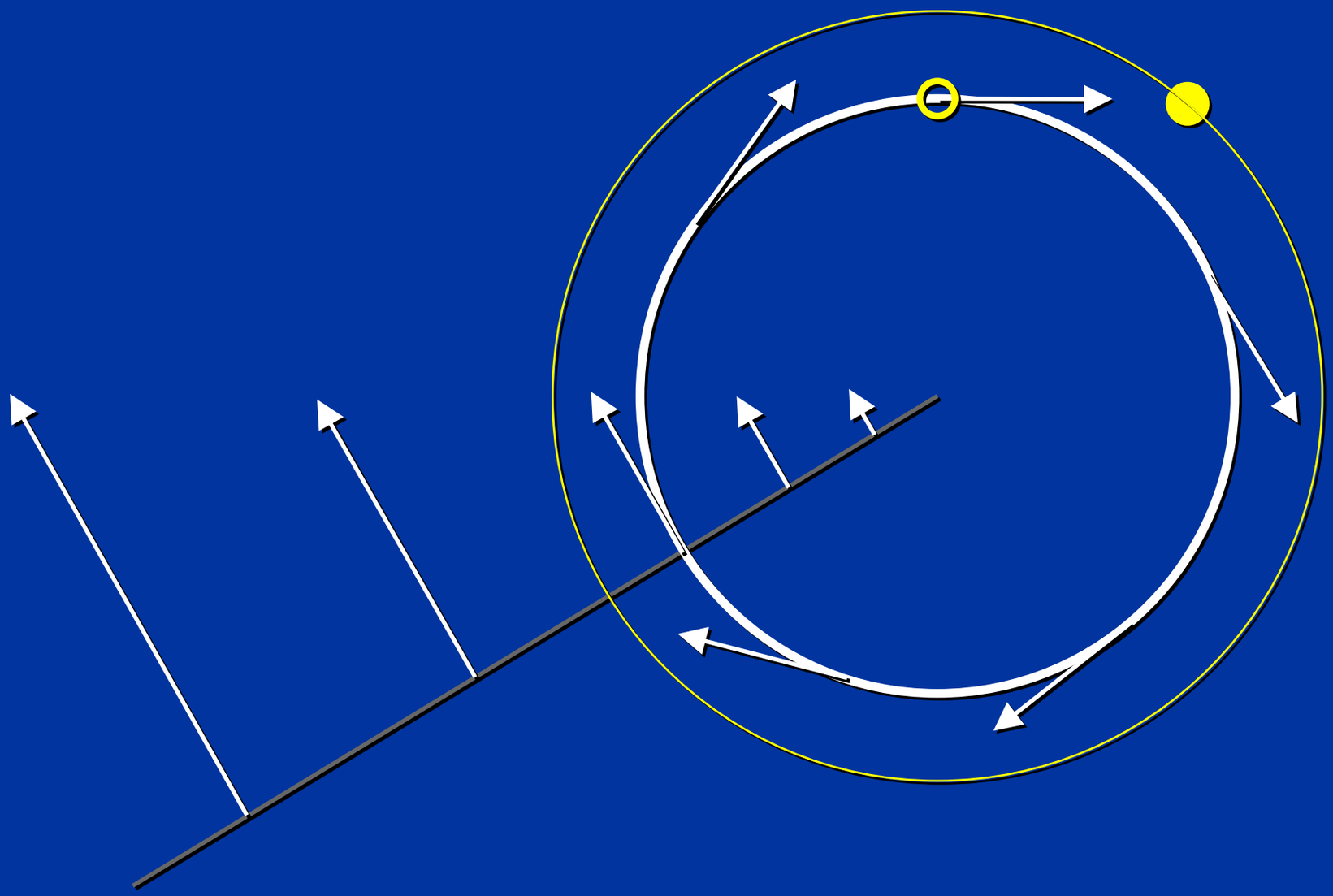
Really big k. Really small h.



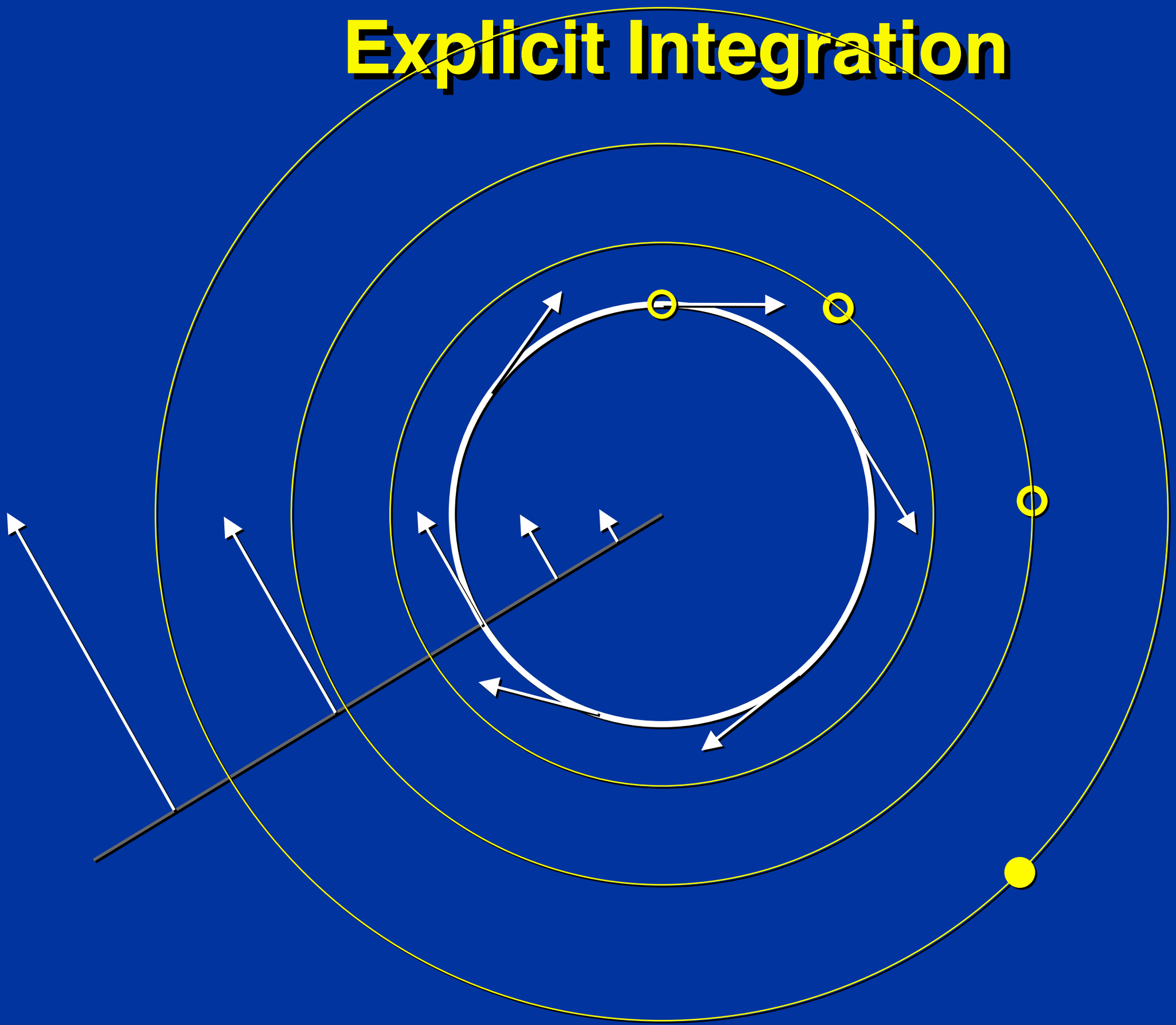
Answer: h has to be so small that P will never move more than ε per step.

Result: Your simulation grinds to a halt.

Explicit Integration



Explicit Integration



(Explicit) Euler Method

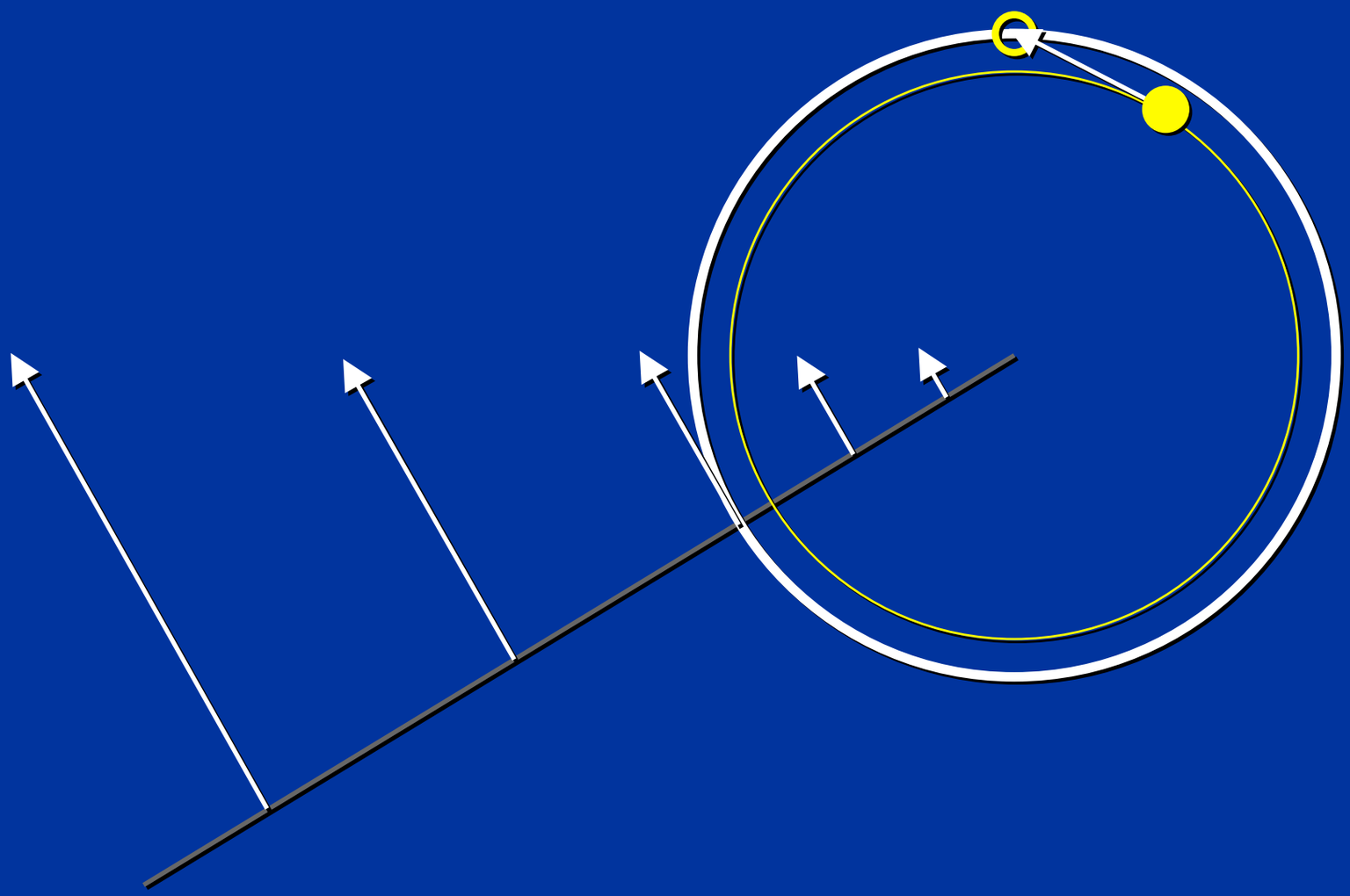
$$x(t_0 + h) = x(t_0) + h\dot{x}(t_0)$$

Implicit Euler Method

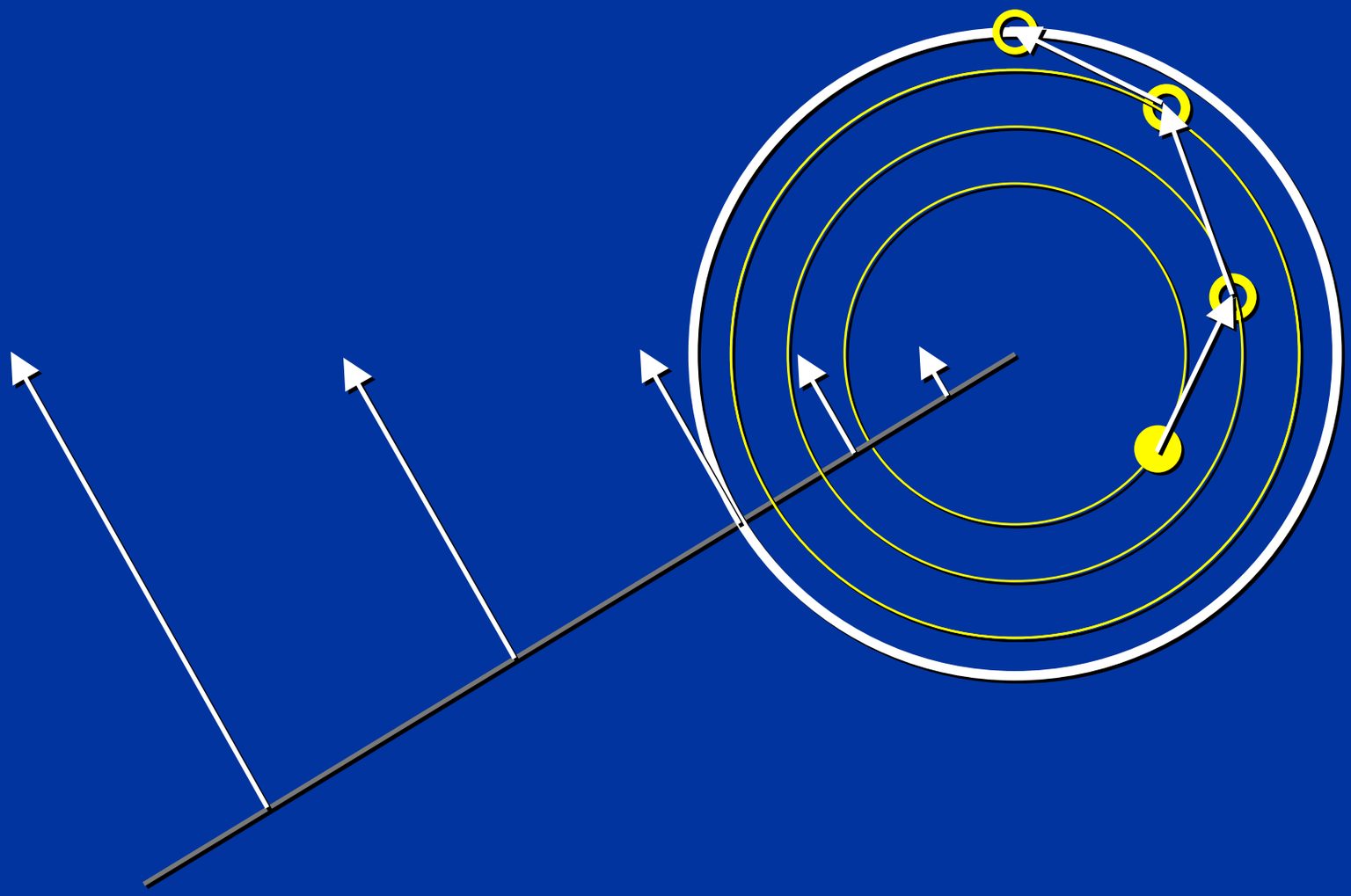
$$x(t_0 + h) = x(t_0) + h\dot{x}(t_0)$$

$$x(t_0 + h) = x(t_0) + h\dot{x}(t_0 + \Delta t)$$

Implicit Integration



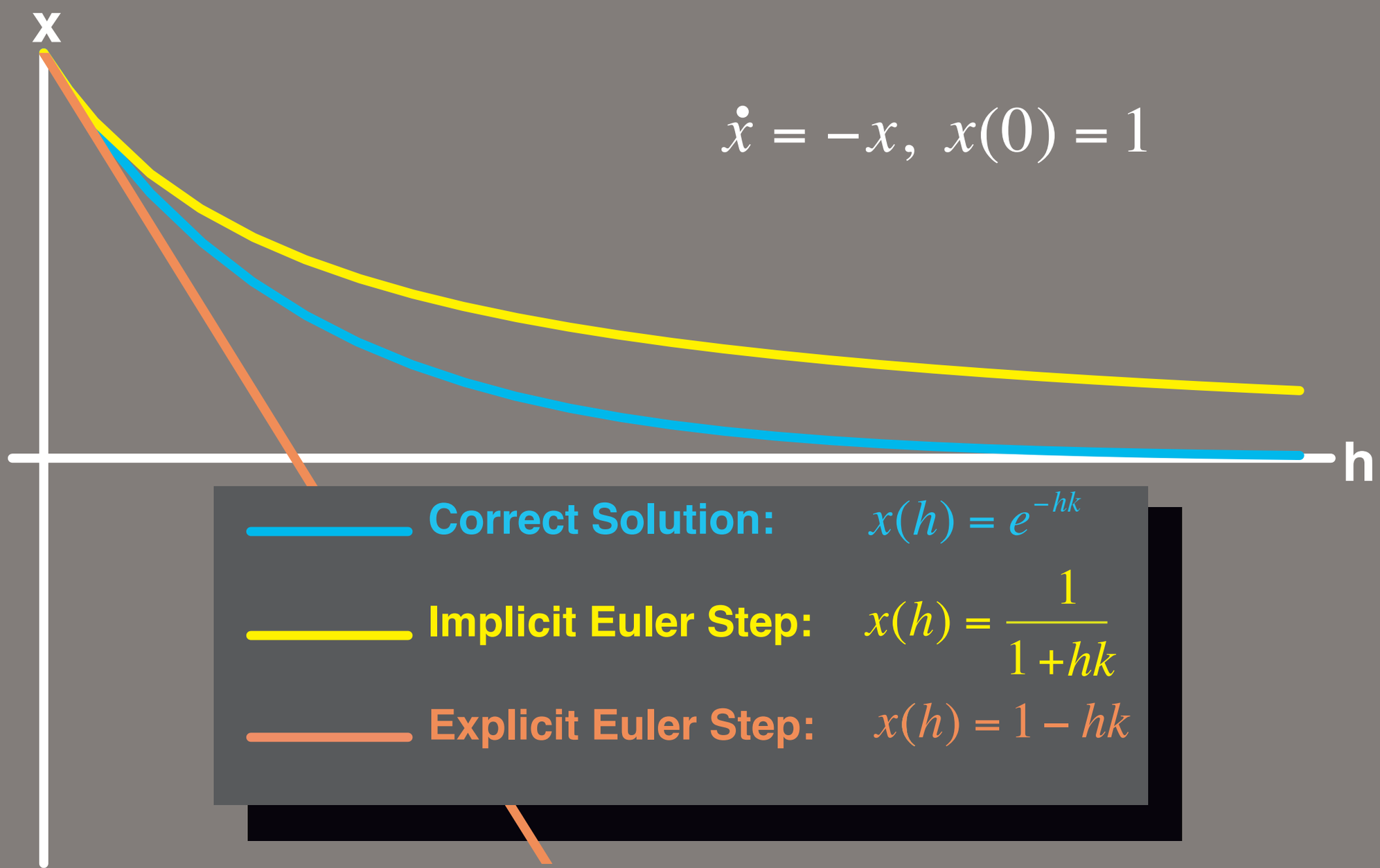
Implicit Integration



Implicit Euler

$$\begin{cases} \dot{x} = -x \\ x(0) = 1 \end{cases}$$

One Step: Implicit vs. Explicit



$$\dot{x} = -x, \; x(0) = 1$$

Correct Solution: $x(h) = e^{-hk}$

Implicit Euler Step: $x(h) = \dfrac{1}{1+hk}$

Explicit Euler Step: $x(h) = 1 - hk$

Large Systems

$$\frac{d}{dt}\mathbf{X}(t) = \dot{\mathbf{X}}(t) = f\left(\mathbf{X}(t)\right)$$

$$\Delta\mathbf{X}(t_0) = h\dot{\mathbf{X}}(t_0 + \Delta t) = h f\left(\mathbf{X}(t_0 + \Delta t)\right)$$

$$= h f\left(\mathbf{X}(t_0) + \Delta\mathbf{X}(t_0)\right)$$

(Linearized) Implicit Integration

$$\dot{\mathbf{X}}(t) = f\left(\mathbf{X}(t)\right)$$

$$\Delta\mathbf{X} = h\, f\left(\mathbf{X}_0 + \Delta\mathbf{X}\right)$$

$$\Delta\mathbf{X} = h\left(f\left(\mathbf{X}_0\right) + \left(\frac{\partial f}{\partial \mathbf{X}}\right)\Delta\mathbf{X}\right)$$

Single-Step Implicit Euler Method

$$\Delta \mathbf{X} = h\left(f(\mathbf{X}_0) + \left(\frac{\partial f}{\partial \mathbf{X}} \right) \Delta \mathbf{X} \right)$$

$$\underbrace{\left(\mathbf{I} - h \frac{\partial}{\partial \mathbf{X}} \left(\dot{\mathbf{X}}(t_0) \right) \right)}_{n \times n \text{ sparse matrix}} \Delta \mathbf{X} = h \dot{\mathbf{X}}(t_0)$$

Solving Large Systems

- Matrix structure reflects force-coupling: (i,j)th entry exists iff f_i depends on $\mathbf{X}_j$
- Conjugate gradient a good first choice
- Is this a lot of work?

Questions

- **Consider the system:**

$$\dot{\mathbf{x}} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \mathbf{x}$$

- **What would happen if you solved x_1 explicitly and x_2 implicitly?**

Reading for Next Monday

- Read *Implicit Methods for Differential Equations*
- (on the website)