

Lecture 18:

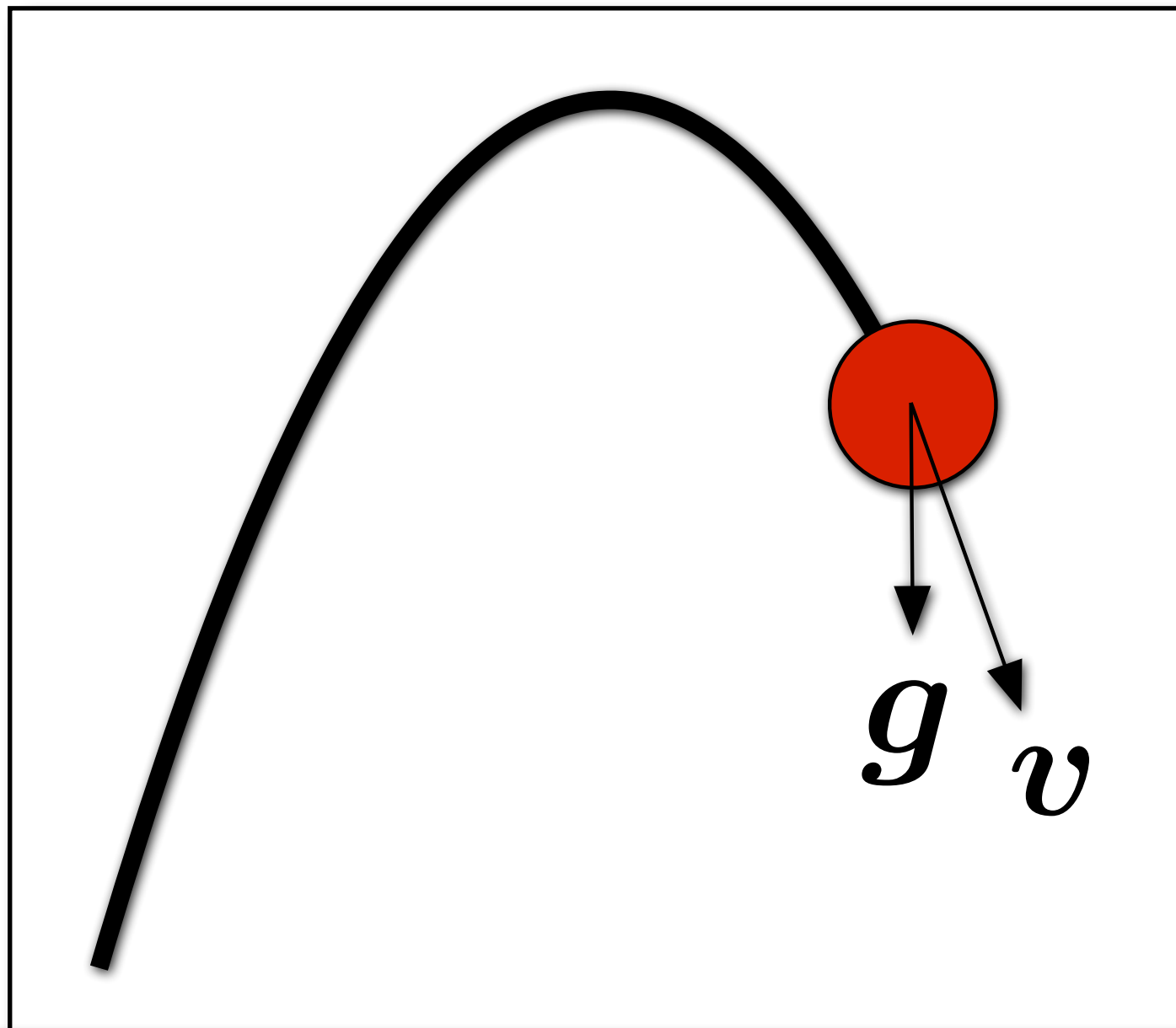
Introduction to Physical Simulation

**Computer Graphics and Imaging
UC Berkeley CS184/284A**

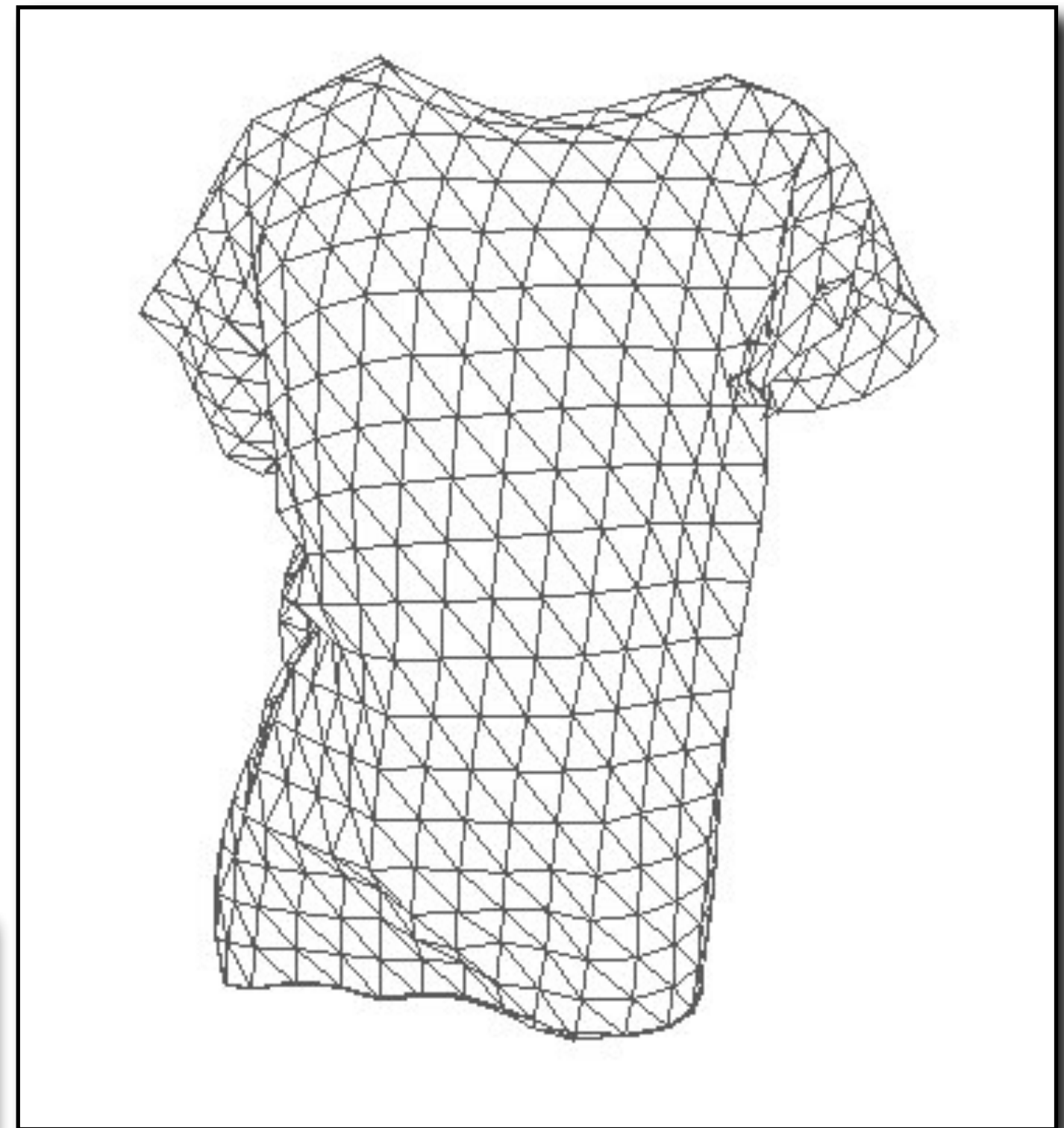
The majority of these slides courtesy of James O'Brien and Keenan Crane.

Physically Based Animation

Generate motion of objects using numerical simulation



$$\mathbf{x}^{t+\Delta t} = \mathbf{x}^t + \Delta t \mathbf{v}^t + \frac{1}{2} (\Delta t)^2 \mathbf{a}^t$$



Example: Cloth Simulation



Example: Fluids



Macklin and Müller, Position Based Fluids

Particle Systems

Single particles are very simple

Large groups can produce interesting effects

Supplement basic ballistic rules

- Collisions
- Interactions
- Force fields
- Springs
- Others...



Karl Sims, SIGGRAPH 1990

PARTICLE DREAMS

Karl Sims

Optomystic

Mass + Spring Systems:

Example of Modeling a Dynamical System

Example: Mass Spring Rope

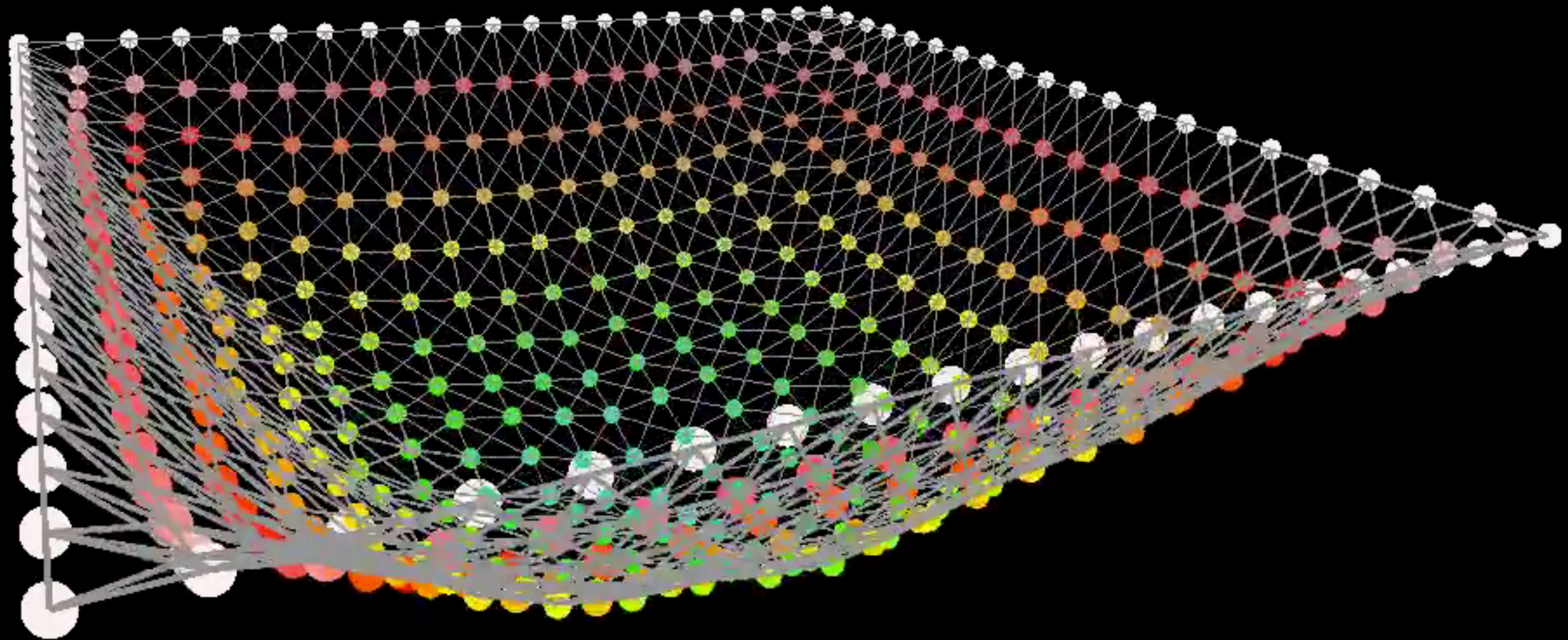


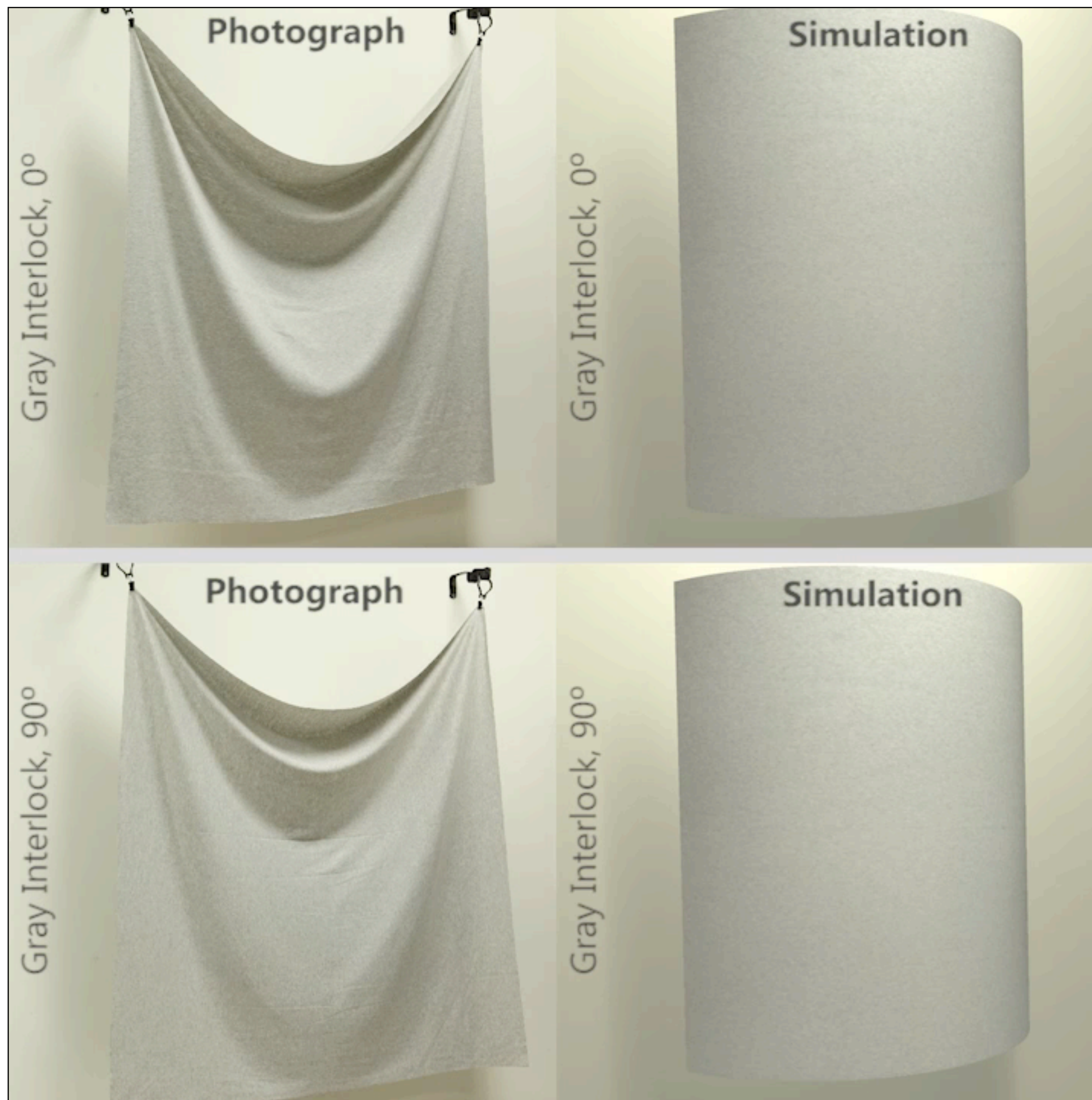
Credit: Elizabeth Labelle, <https://youtu.be/Co8enp8CH34>

Example: Hair



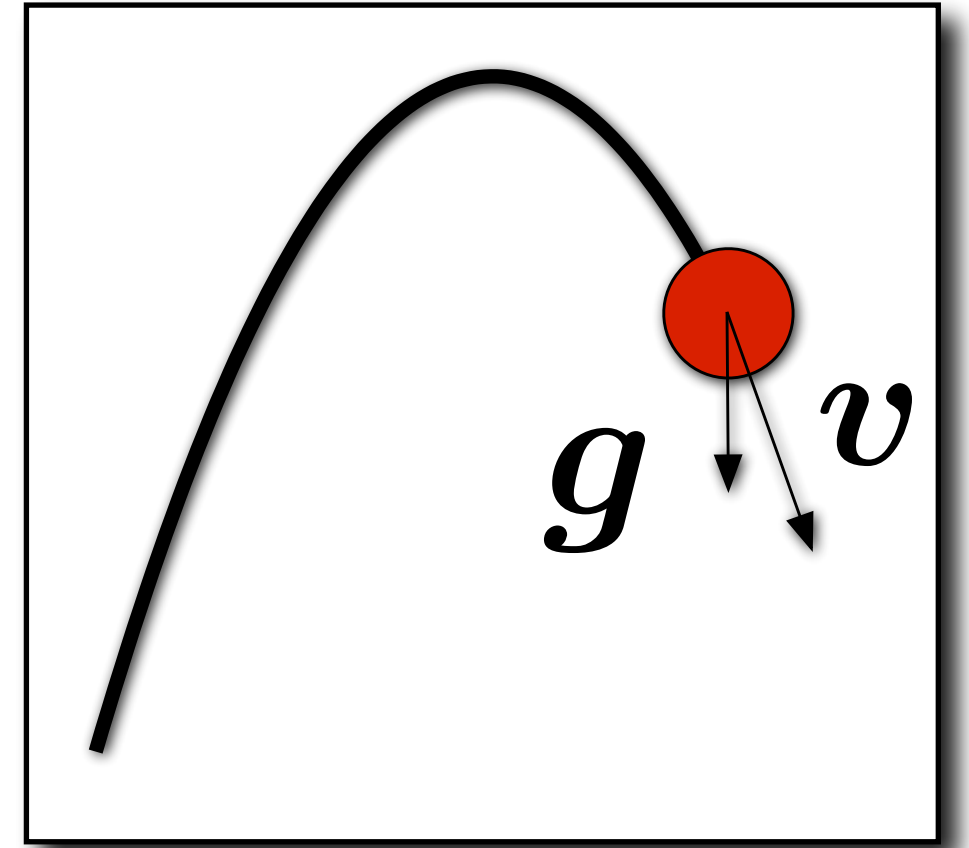
Example: Mass Spring Mesh





Huamin Wang, Ravi Ramamoorthi, and James F. O'Brien. "Data-Driven Elastic Models for Cloth: Modeling and Measurement". *ACM Transactions on Graphics*, 30(4):71:1–11, July 2011. Proceedings of ACM SIGGRAPH 2011, Vancouver, BC Canada.

Newton's Law



$$\begin{array}{c} \text{Force} \uparrow \\ F \end{array} = \begin{array}{c} \text{Mass} \uparrow \\ m \end{array} \begin{array}{c} \text{Acceleration} \uparrow \\ a \end{array}$$

Dot Notation for Derivatives

If x is a vector for the position of a point of interest, we will use dot notation for velocity and acceleration:

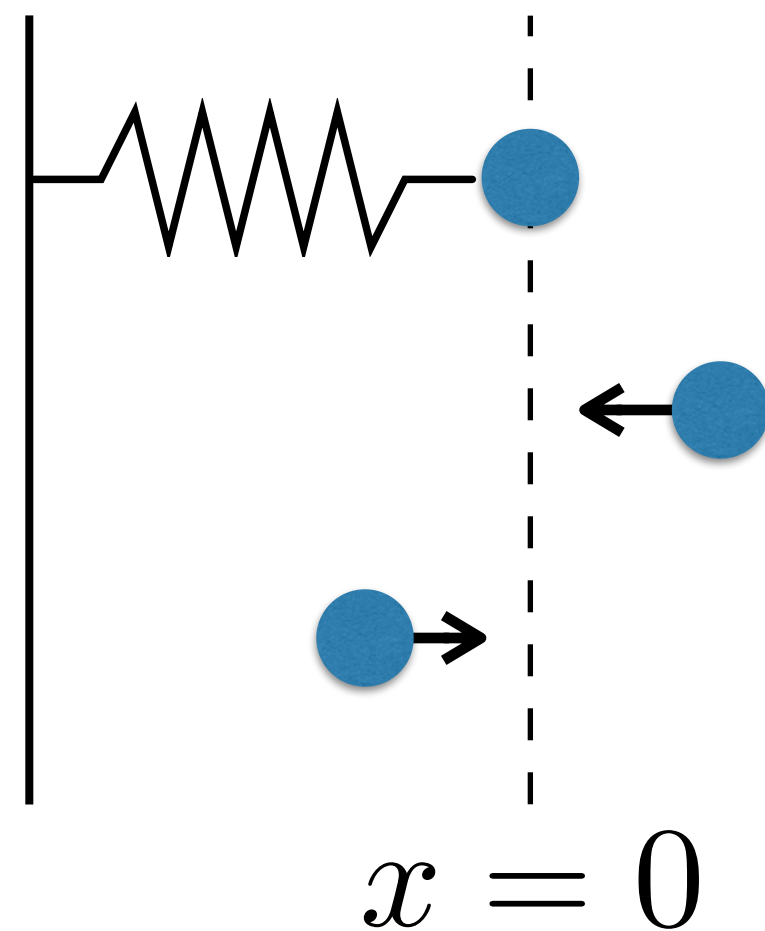
$$x$$

$$\dot{x} = v$$

$$\ddot{x} = a$$

Hooke's Law

Idealized spring



$$f = -k_s x$$

Force pulls point-mass back to rest position

Strength proportional to displacement (Hooke's Law)

k_s is a spring coefficient: stiffness

Generalizing Hooke's Law

Spring connects two particles



The diagram shows two blue circular particles, labeled a and b , connected by a zigzag line representing a spring. Particle a is on the left and particle b is on the right.

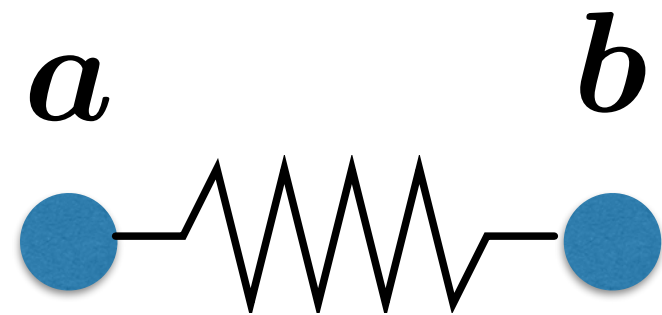
$$f_{a \rightarrow b} = k_s(b - a)$$
$$f_{b \rightarrow a} = -f_{a \rightarrow b}$$

Force pulls points together

Problem: this spring wants to have zero length

Non-Zero Length Spring

Spring with non-zero rest length



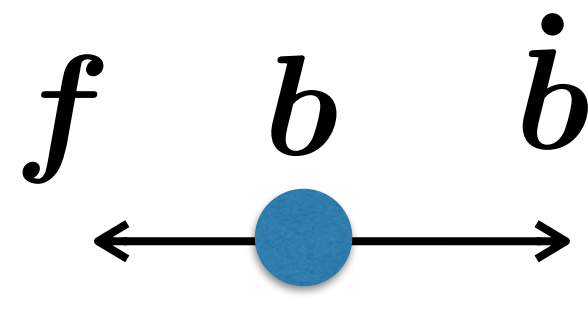
$$\mathbf{f}_{a \rightarrow b} = k_s \frac{\mathbf{b} - \mathbf{a}}{||\mathbf{b} - \mathbf{a}||} (||\mathbf{b} - \mathbf{a}|| - l)$$

Rest length

Problem: oscillates forever

Simple Motion Damping

Simple motion damping



A diagram showing a blue circular mass. Above the mass are three labels: f , b , and $\dot{b}$. Below the mass is a horizontal double-headed arrow, with the left arrow pointing towards the f label and the right arrow pointing towards the $\dot{b}$ label.

$$f = -k_d \dot{b}$$

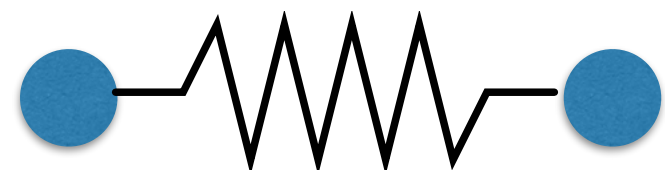
- Behaves like viscous drag on motion
- Acts opposite to the direction of motion
- k_d is a damping coefficient

Problem: slows down *all* motion

- Want a rusty spring's oscillations to slow down, but should it also fall to the ground more slowly?

Internal Damping for Spring

Damp only the internal, spring-driven motion

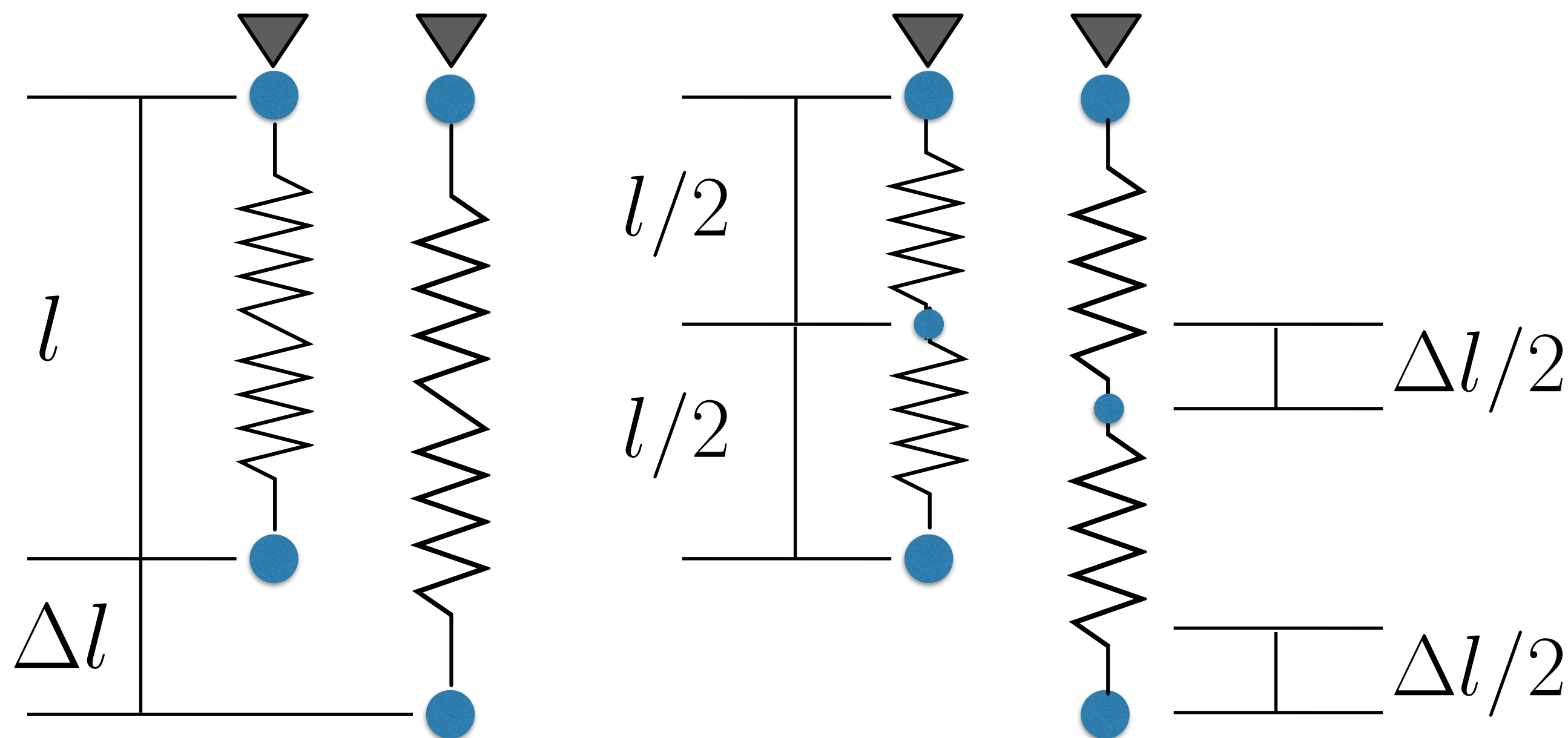


$$f_a = -k_d \frac{b - a}{||b - a||} (\dot{b} - \dot{a}) \cdot \frac{b - a}{||b - a||}$$

- Viscous drag only on change in spring length
 - Won't slow group motion for the spring system (e.g. global translation or rotation of the group)

Spring Constants

Consider two “resolutions” to model a single spring



Problem: constant k_s produces different force on bottom spring for these two different discretizations

Spring Constants

Problem: constant k_s gives inconsistent results with different discretizations of our spring/mass structures

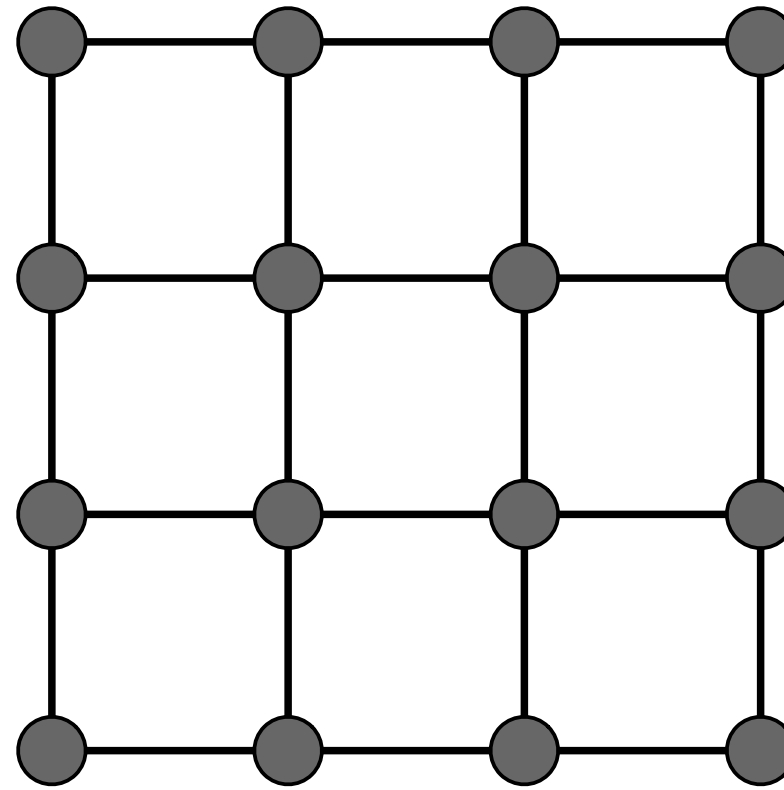
- E.g. 10x10 vs 20x20 mesh for cloth simulation would give different results, and we want them to be the same, just higher level of detail

Solution:

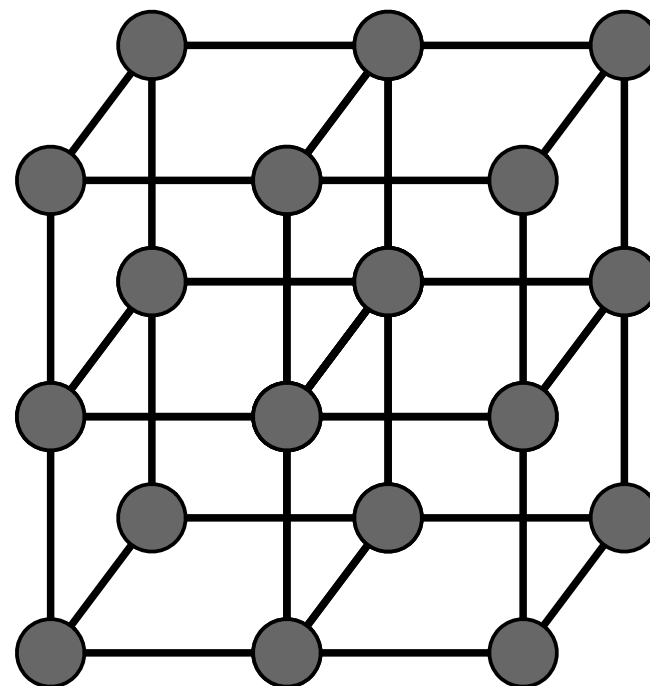
- Change in length is not what we want to measure
- We want to consider the strain = change in length as fraction of original length
$$\epsilon = \frac{\Delta l}{l_0}$$
- Implementation 1: divide spring force by spring length
- Implementation 2: normalize k_s by spring length

Structures from Springs

Sheets



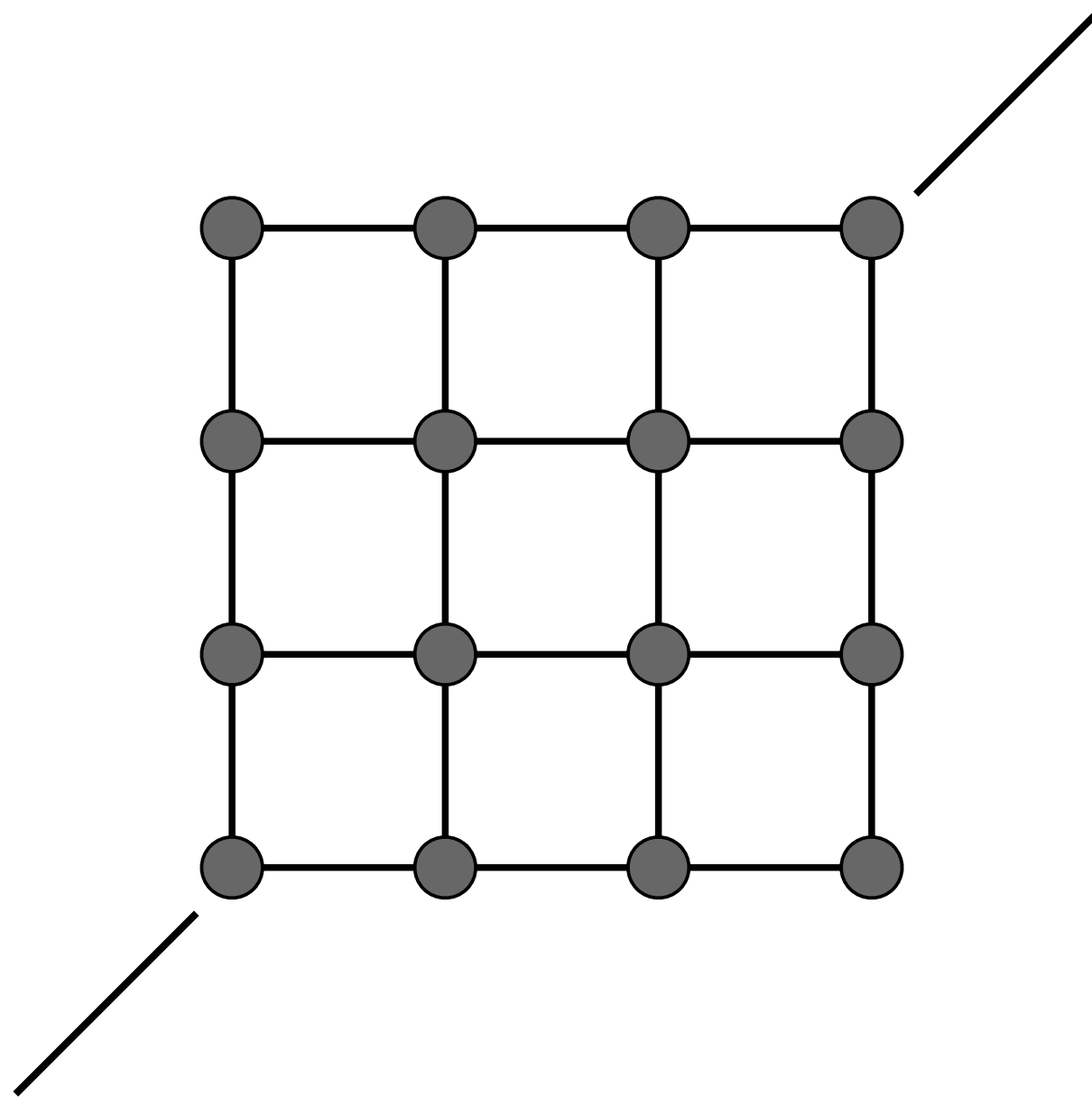
Blocks



Others

Structures from Springs

Behavior is determined by structure linkages

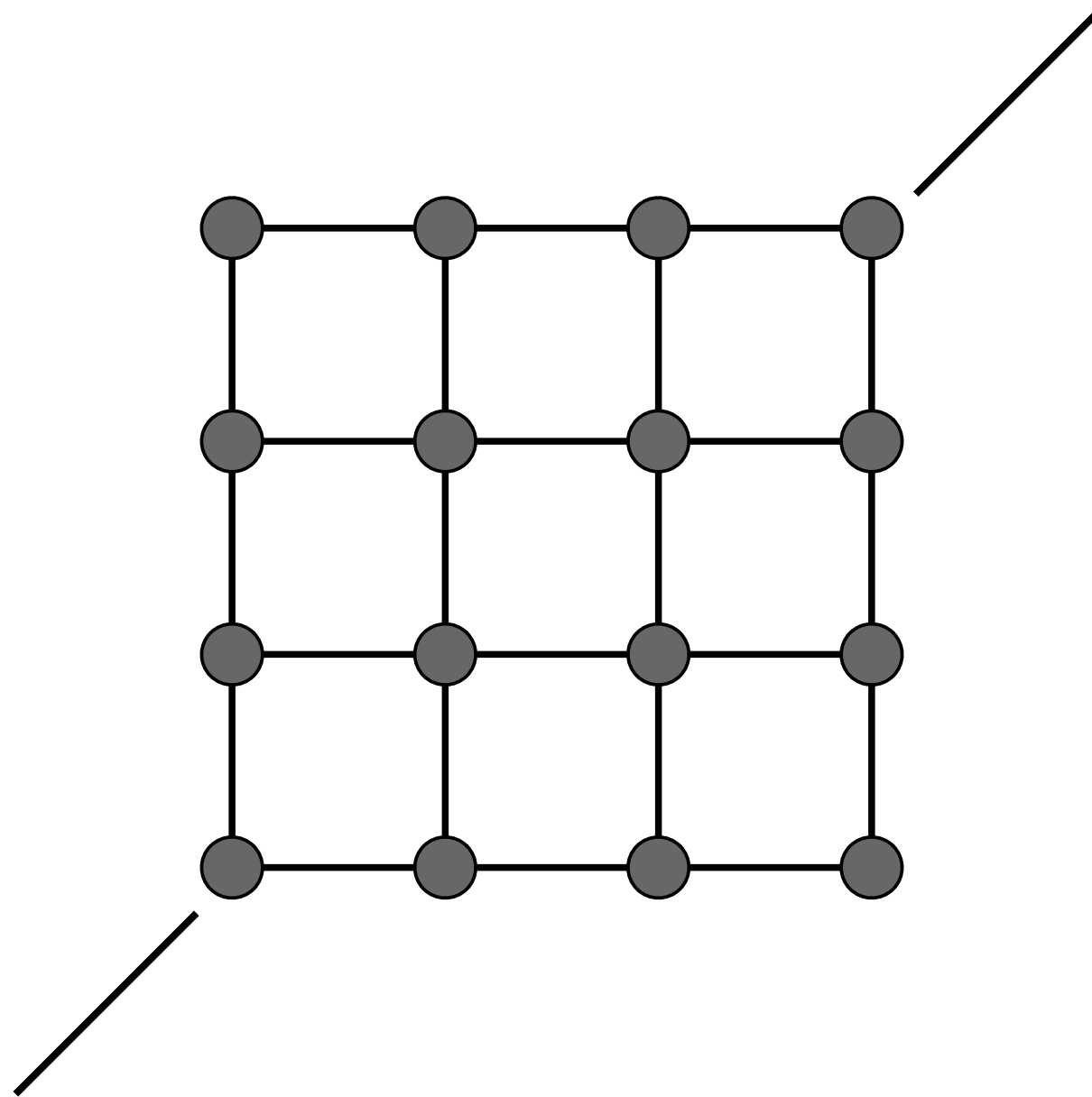


This structure will not resist shearing

This structure will not resist out-of-plane bending...

Structures from Springs

Behavior is determined by structure linkages

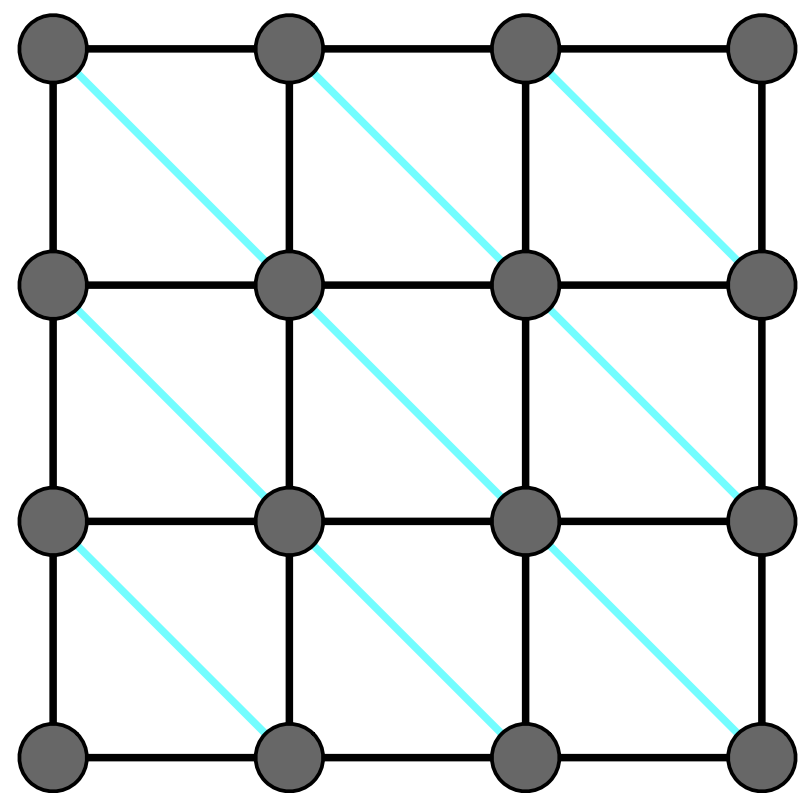


This structure will not resist shearing

This structure will not resist out-of-plane bending...

Structures from Springs

Behavior is determined by structure linkages

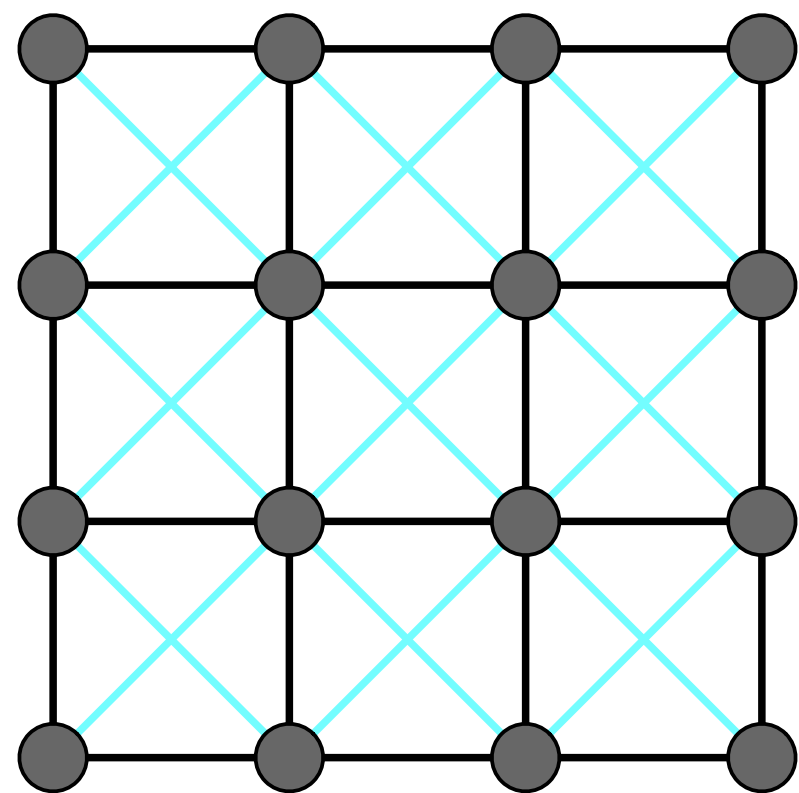


This structure will resist shearing but has anisotropic bias

This structure will not resist out-of-plane bending either...

Structures from Springs

Behavior is determined by structure linkages

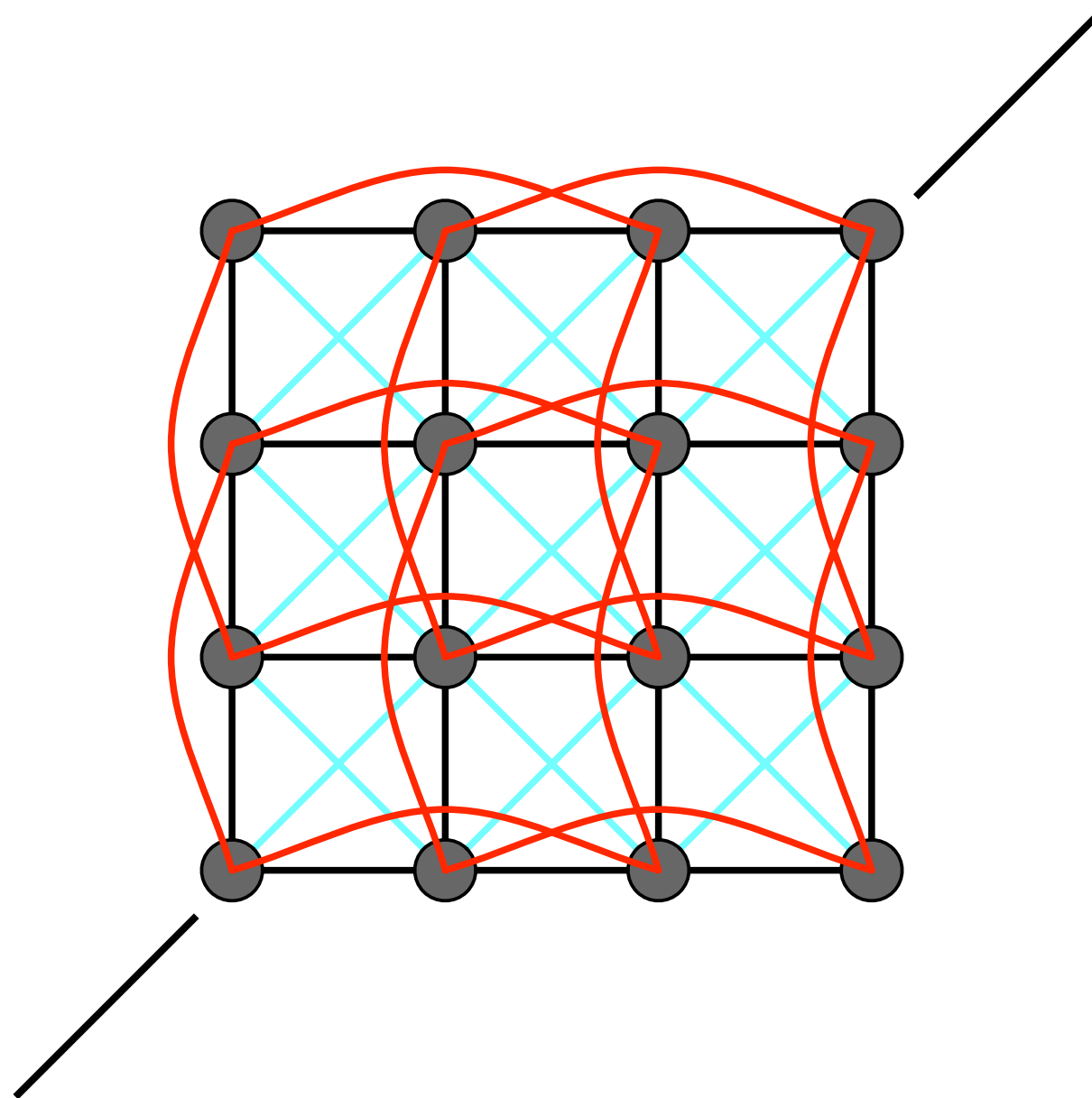


This structure will resist shearing.
Less directional bias.

This structure will not resist out-of-plane
bending either...

Structures from Springs

They behave like what they are (obviously!)



This structure will not resist shearing.
Less directional bias.

This structure will resist out-of-plane
bending
Red springs should be much weaker

Example: Mass Spring Dress + Character

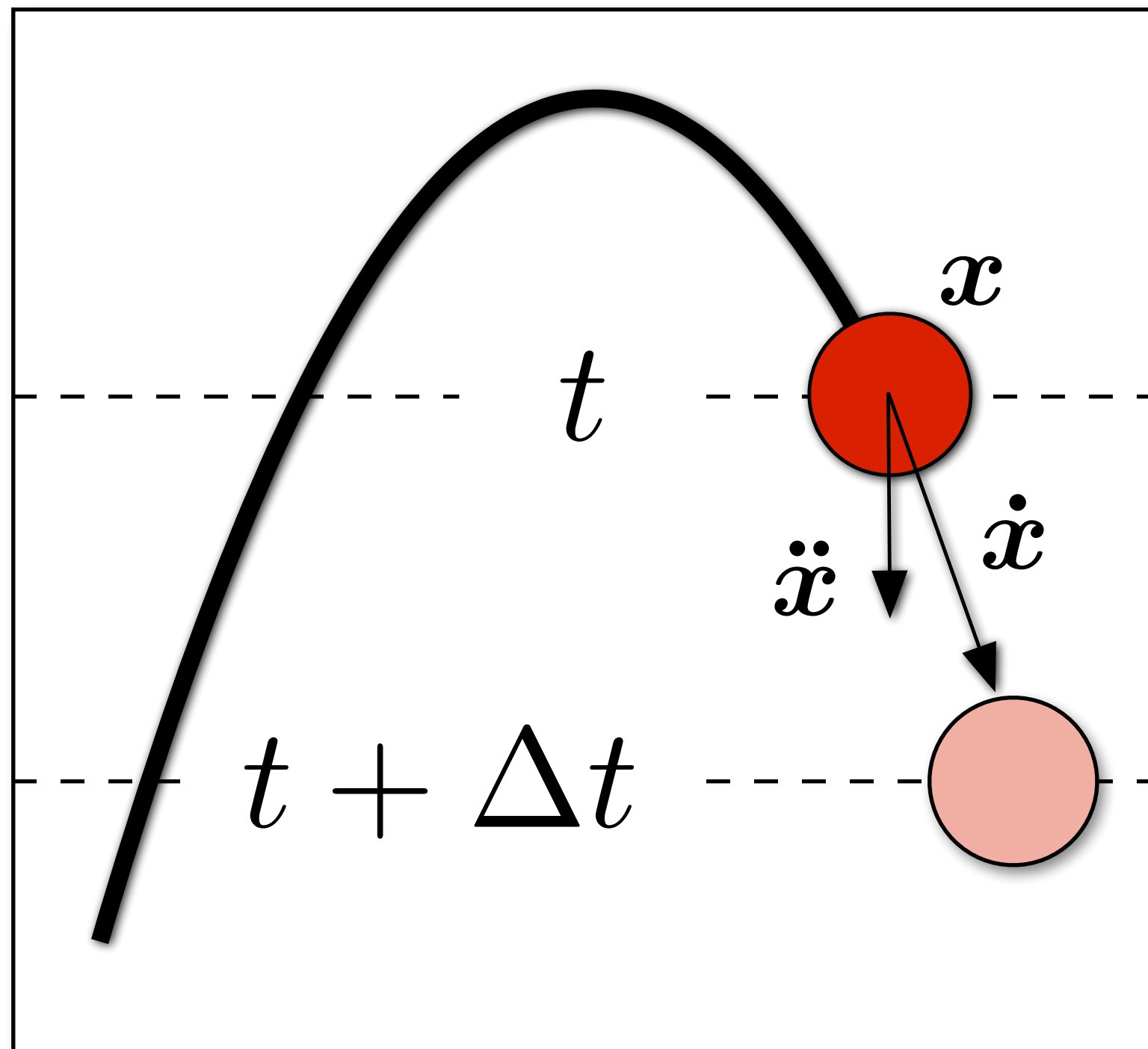


FEM (Finite Element Method) Instead of Springs



Particle Simulation ("Integration")

A First Guess



$$\ddot{x} = \frac{F}{m}$$

$$\ddot{x} = \frac{\Delta \dot{x}}{\Delta t} \quad \dot{x} = \frac{\Delta x}{\Delta t}$$

$$x^{t+\Delta t} = x^t + \Delta t \dot{x}^t$$

$$\dot{x}^{t+\Delta t} = \dot{x}^t + \Delta t \ddot{x}^t$$

Euler's Method

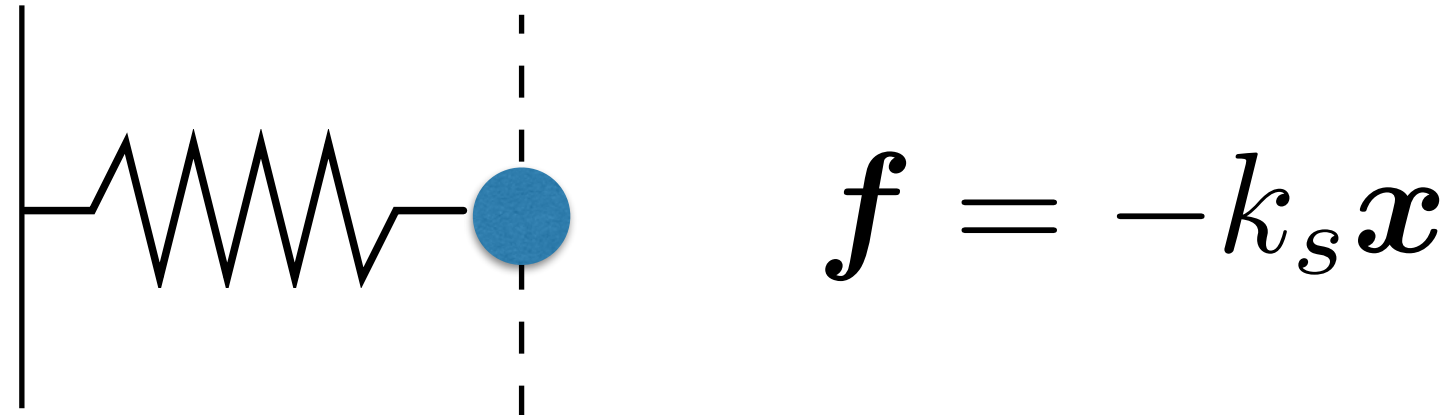
Euler's Method (a.k.a. Forward Euler, Explicit)

- Simple iterative method
- Commonly used
- Very inaccurate
- **Most often goes unstable**

$$x^{t+\Delta t} = x^t + \Delta t \dot{x}^t$$

$$\dot{x}^{t+\Delta t} = \dot{x}^t + \Delta t \ddot{x}^t$$

Instability Example (Spring)



When mass is moving inward:

- Force is decreasing
- Each time-step overestimates the velocity change (increases energy)

When mass gets to origin

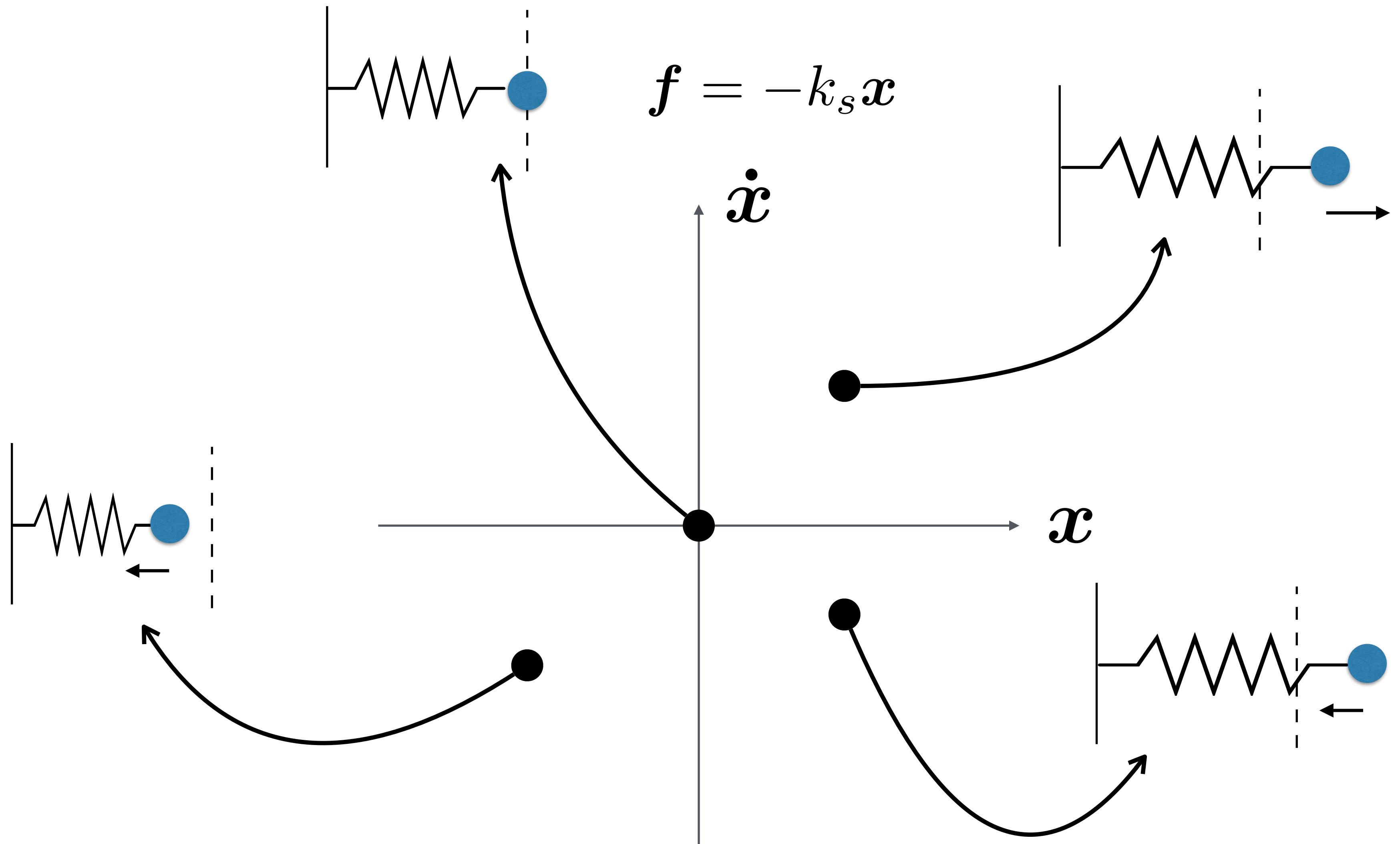
- Has velocity that is too high, now traveling outward

When mass is moving outward

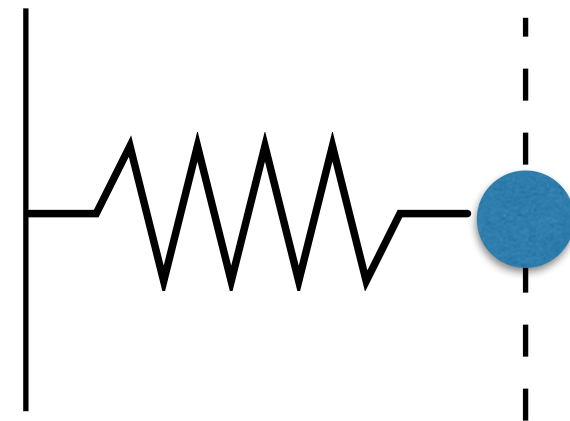
- Force is increasing
- Each time-step underestimates the velocity change (increases energy)

At each motion cycle, mass gains energy exponentially

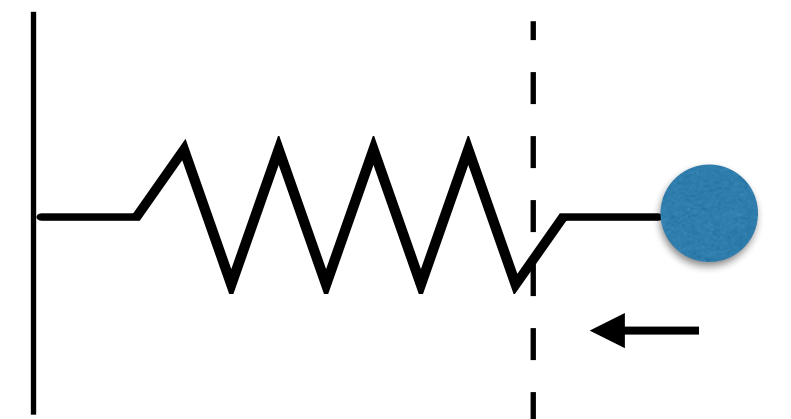
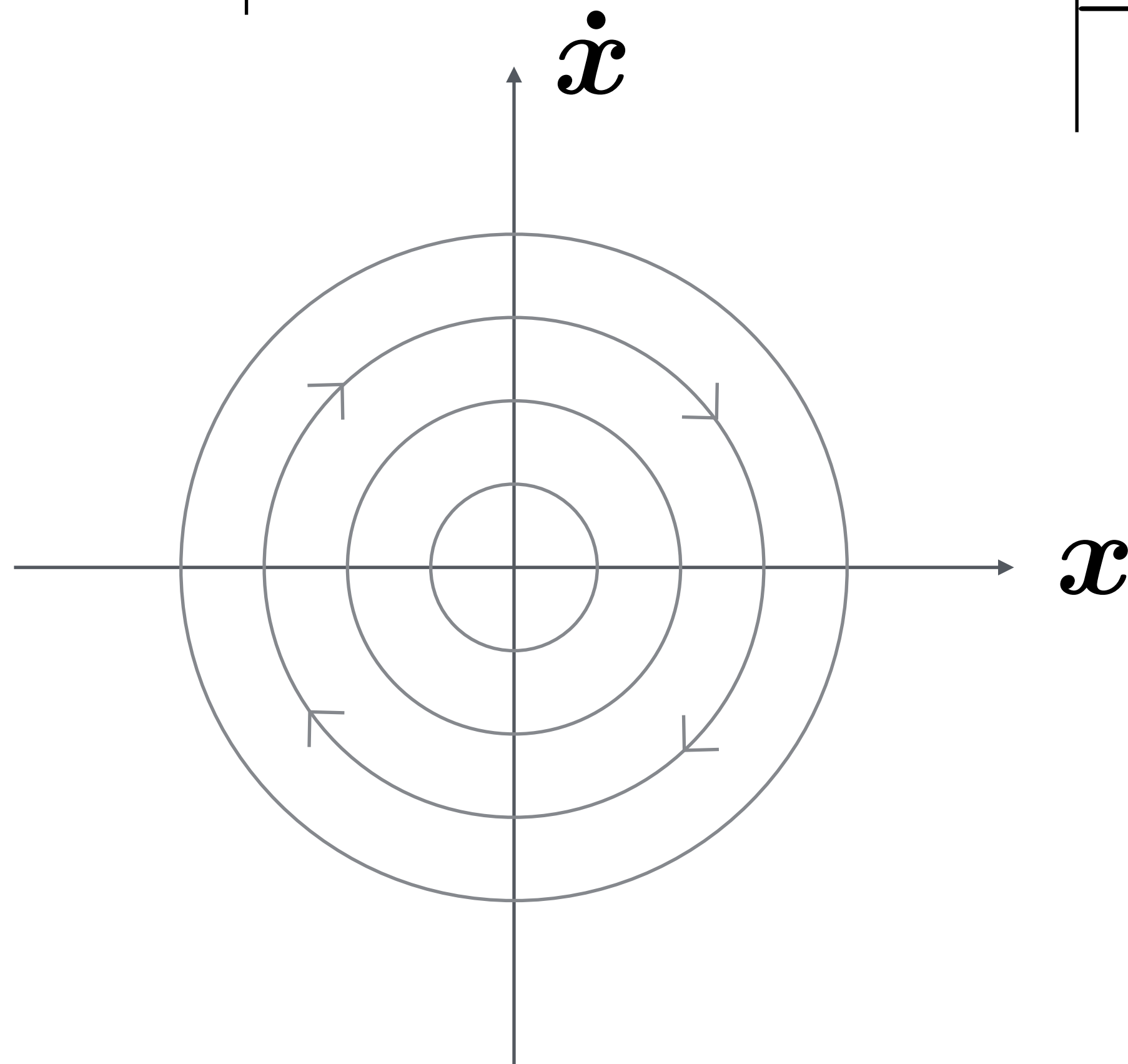
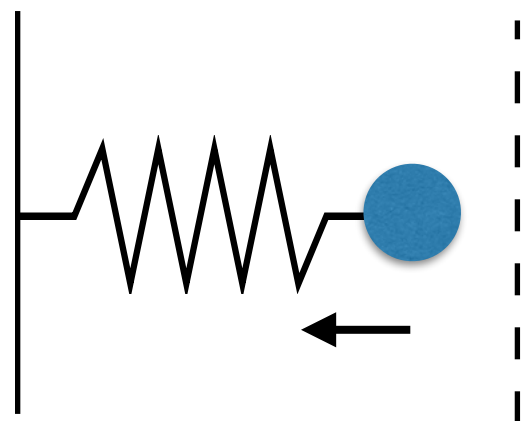
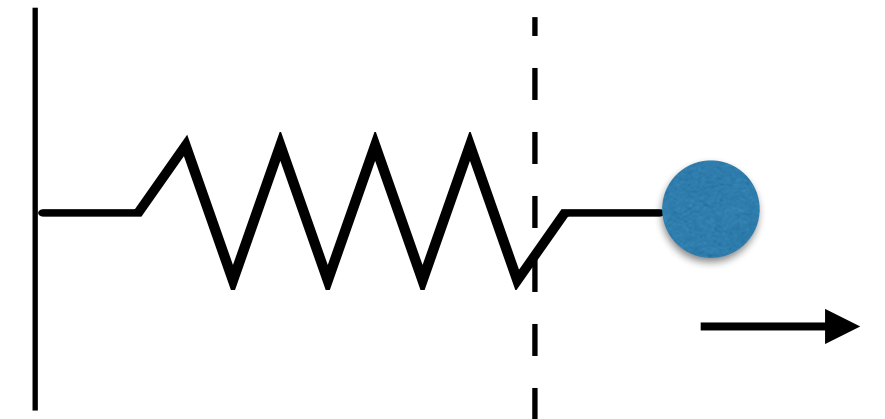
Another View: Phase Space



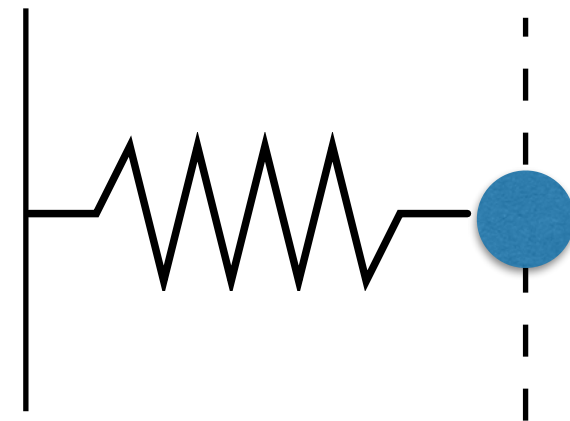
Force is a Vector Field in Phase Space



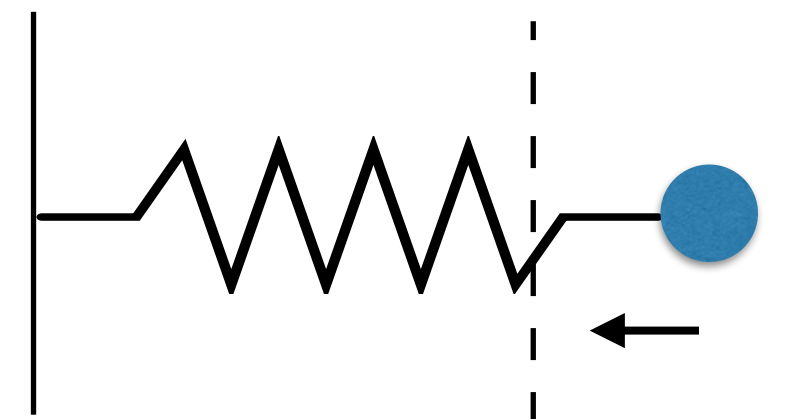
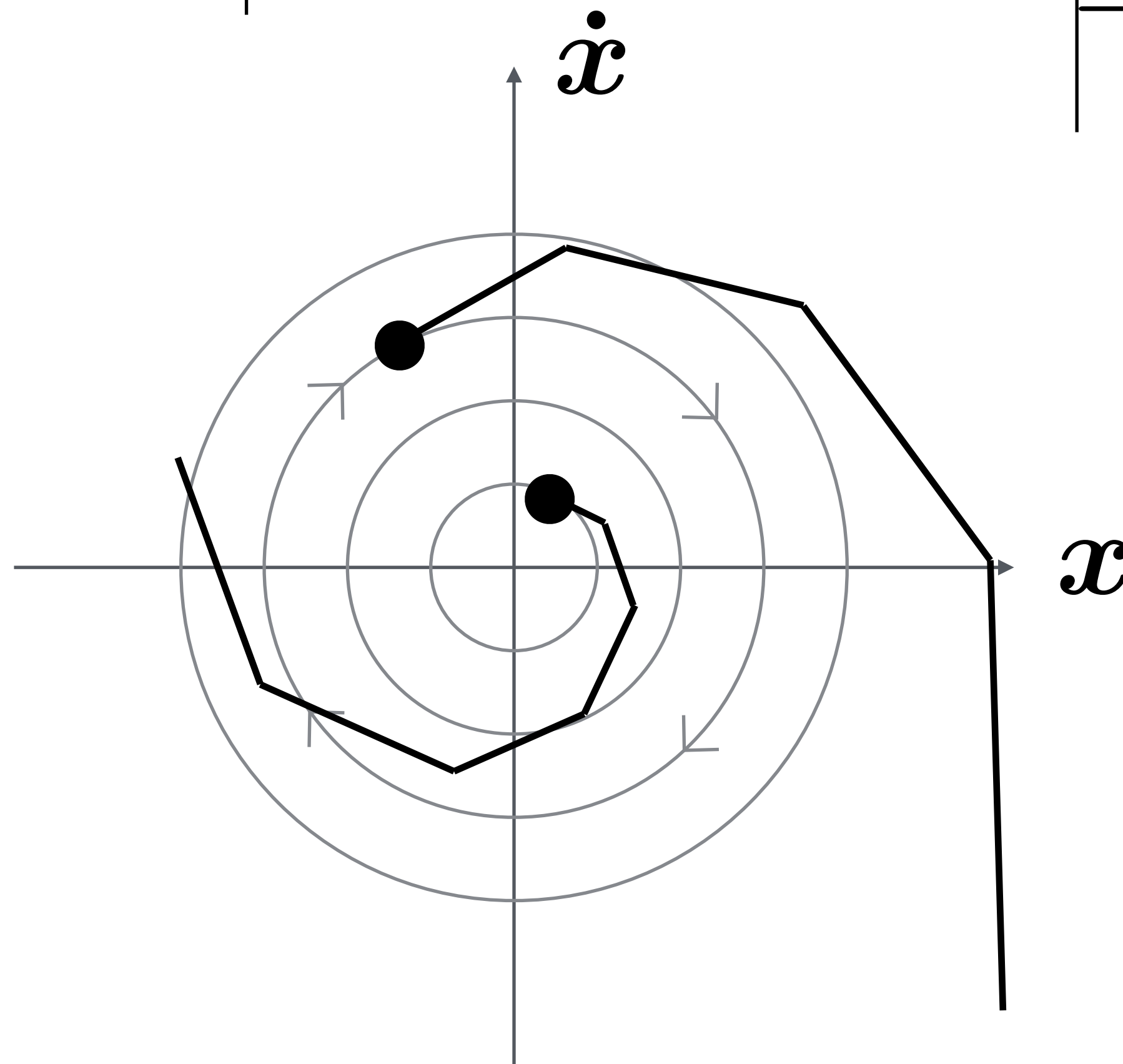
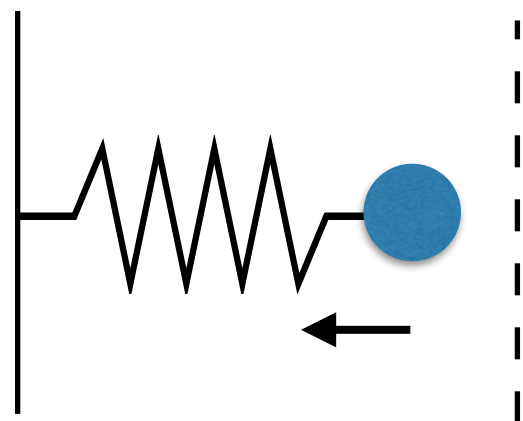
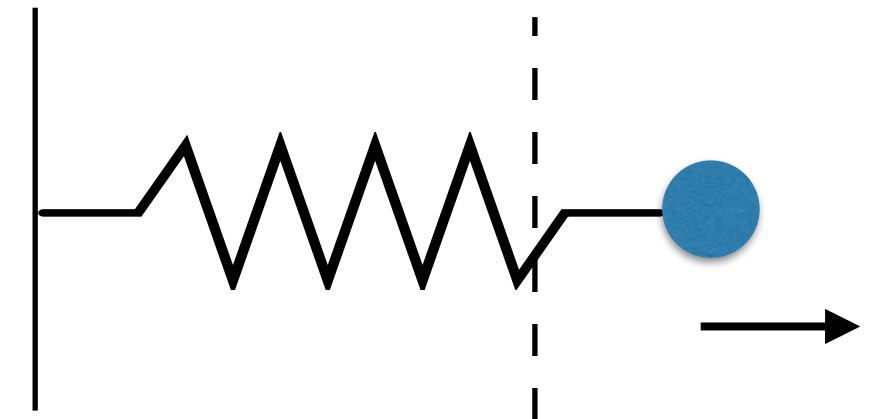
$$f = -k_s x$$



"Spiraling Out of Control"



$$f = -k_s x$$



Euler's Method - Errors

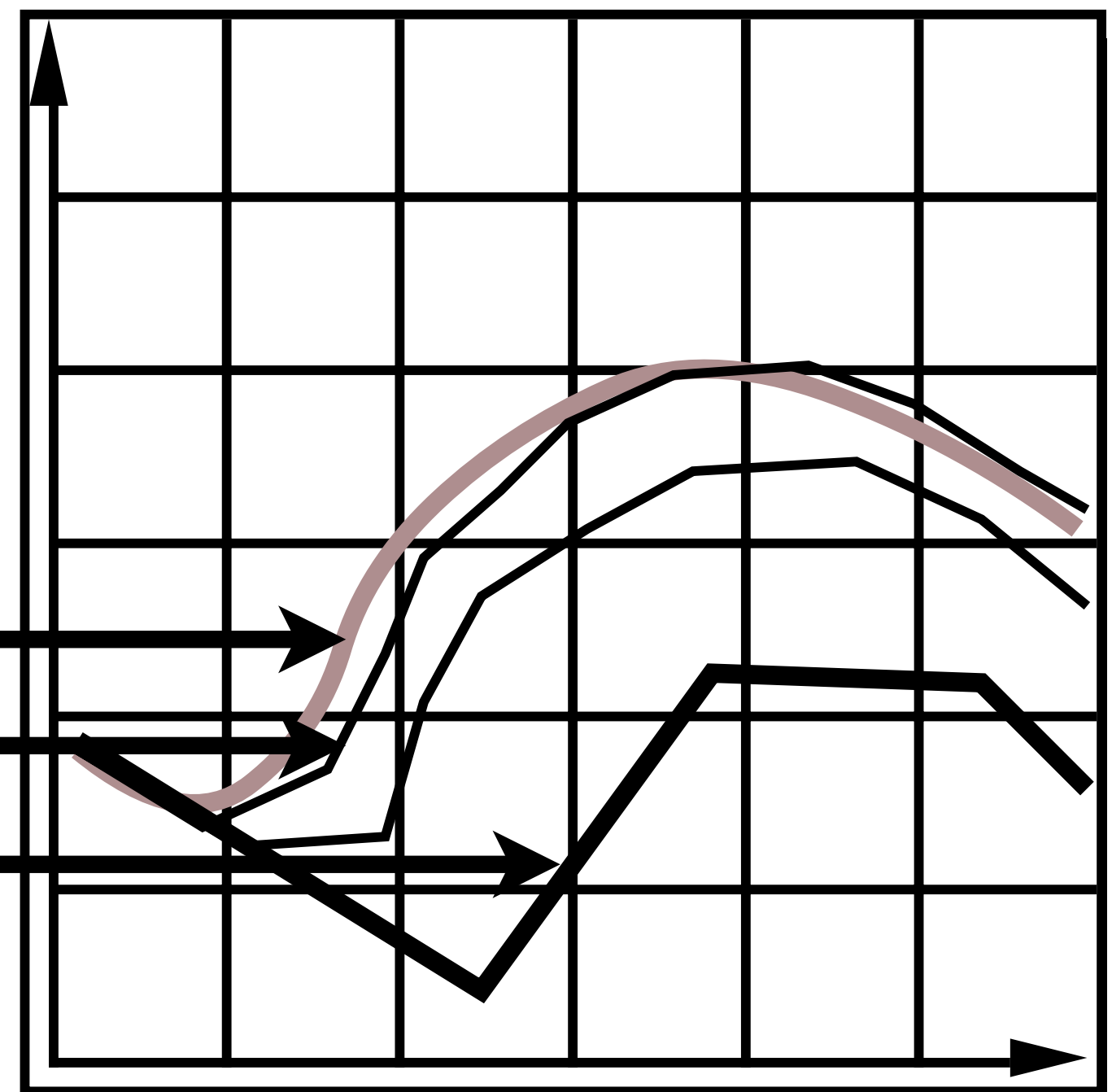
With numerical integration, errors accumulate

Euler integration is particularly bad

Example:

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

Solution path
Euler estimate with small time step
Euler estimate with large time step



Witkin and Baraff

Errors and Instability

Solving by numerical integration with finite differences leads to two problems

Errors

- Errors at each time step accumulate. Accuracy decreases as simulation proceeds
- Accuracy may not be critical in graphics applications

Instability

- Errors can compound, causing the simulation to diverge even when the underlying system does not
- Lack of stability is a fundamental problem in simulation, and cannot be ignored

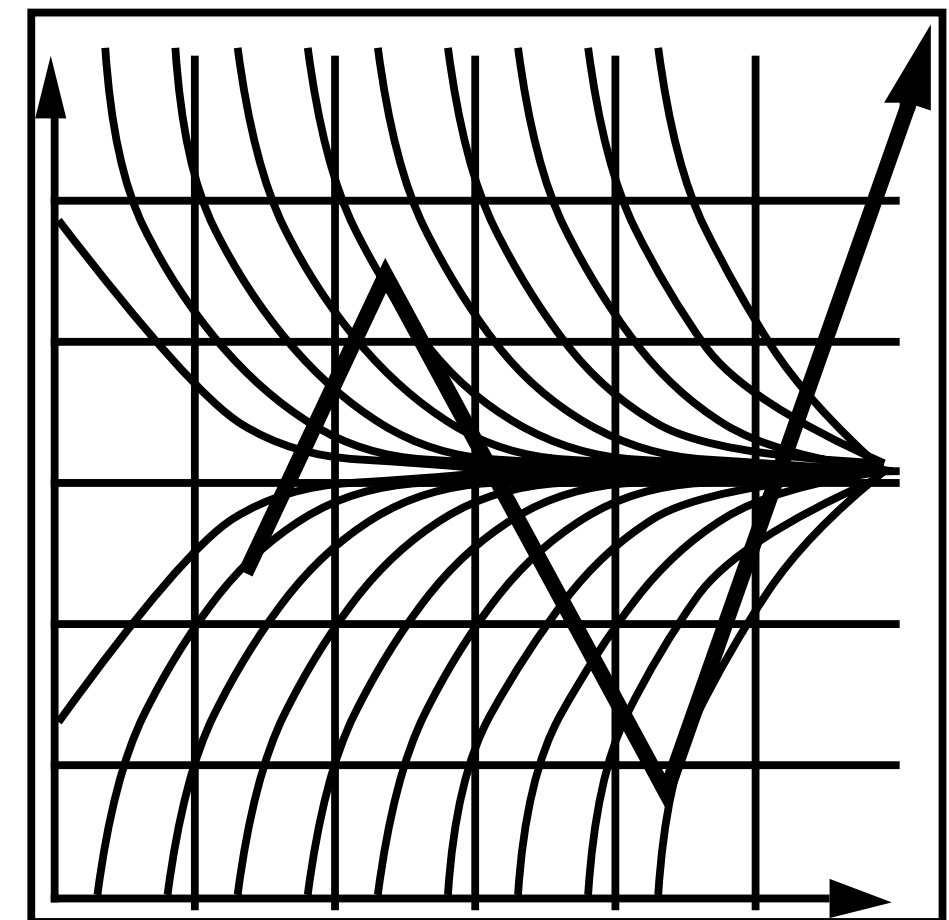
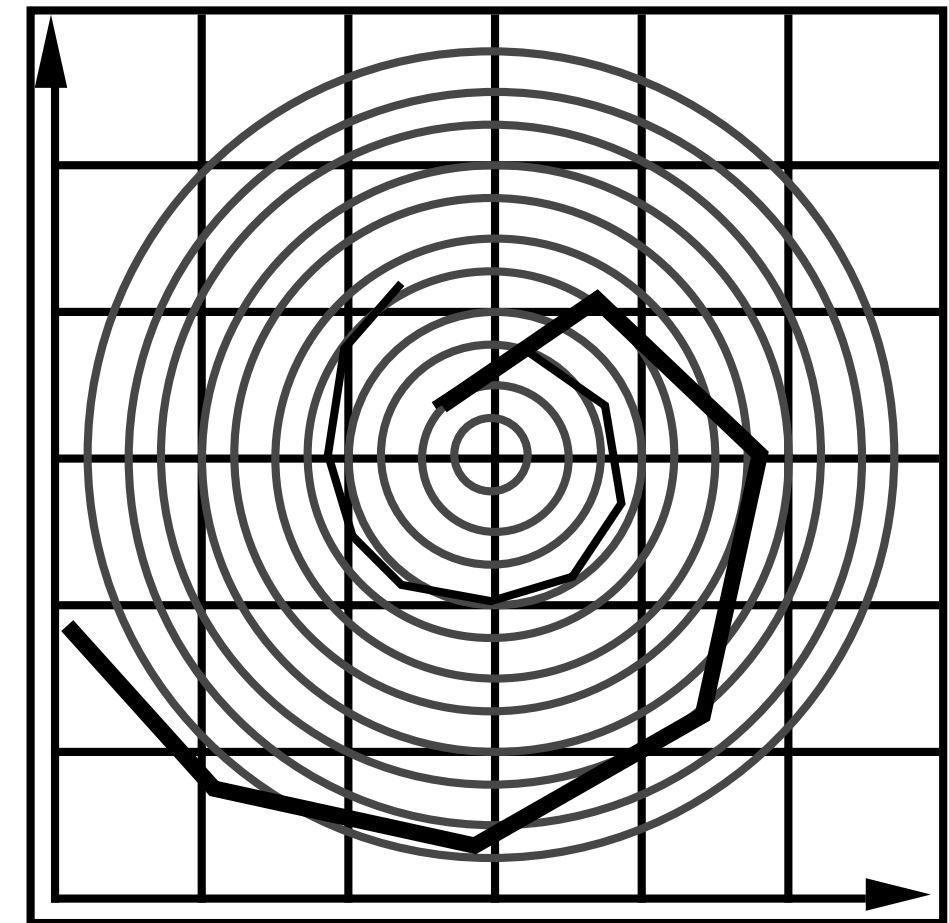
Instability of Forward Euler Method

Forward Euler (explicit)

$$x^{t+\Delta t} = x^t + \Delta t v(x, t)$$

Two key problems:

- Inaccuracies increase as time step Δt increases
- Instability is a common, serious problem that can cause simulation to diverge



Witkin and Baraff

Combating Instability

Some Methods to Combat Instability

Modified Euler

- Average velocities at start and endpoint

Adaptive step size

- Compare one step and two half-steps, recursively, until error is acceptable

Implicit methods

- Use the velocity at the next time step (hard)

Position-based / Verlet integration

- Constrain positions and velocities of particles after time step

Modified Euler

Modified Euler

- Average velocity at start and end of step
- OK if system is not very stiff (k_s small enough)
- But, still unstable

$$\mathbf{x}^{t+\Delta t} = \mathbf{x}^t + \frac{\Delta t}{2} (\dot{\mathbf{x}}^t + \dot{\mathbf{x}}^{t+\Delta t})$$

$$\dot{\mathbf{x}}^{t+\Delta t} = \dot{\mathbf{x}}^t + \Delta t \ddot{\mathbf{x}}^t$$

$$\mathbf{x}^{t+\Delta t} = \mathbf{x}^t + \Delta t \dot{\mathbf{x}}^t + \frac{(\Delta t)^2}{2} \ddot{\mathbf{x}}^t$$

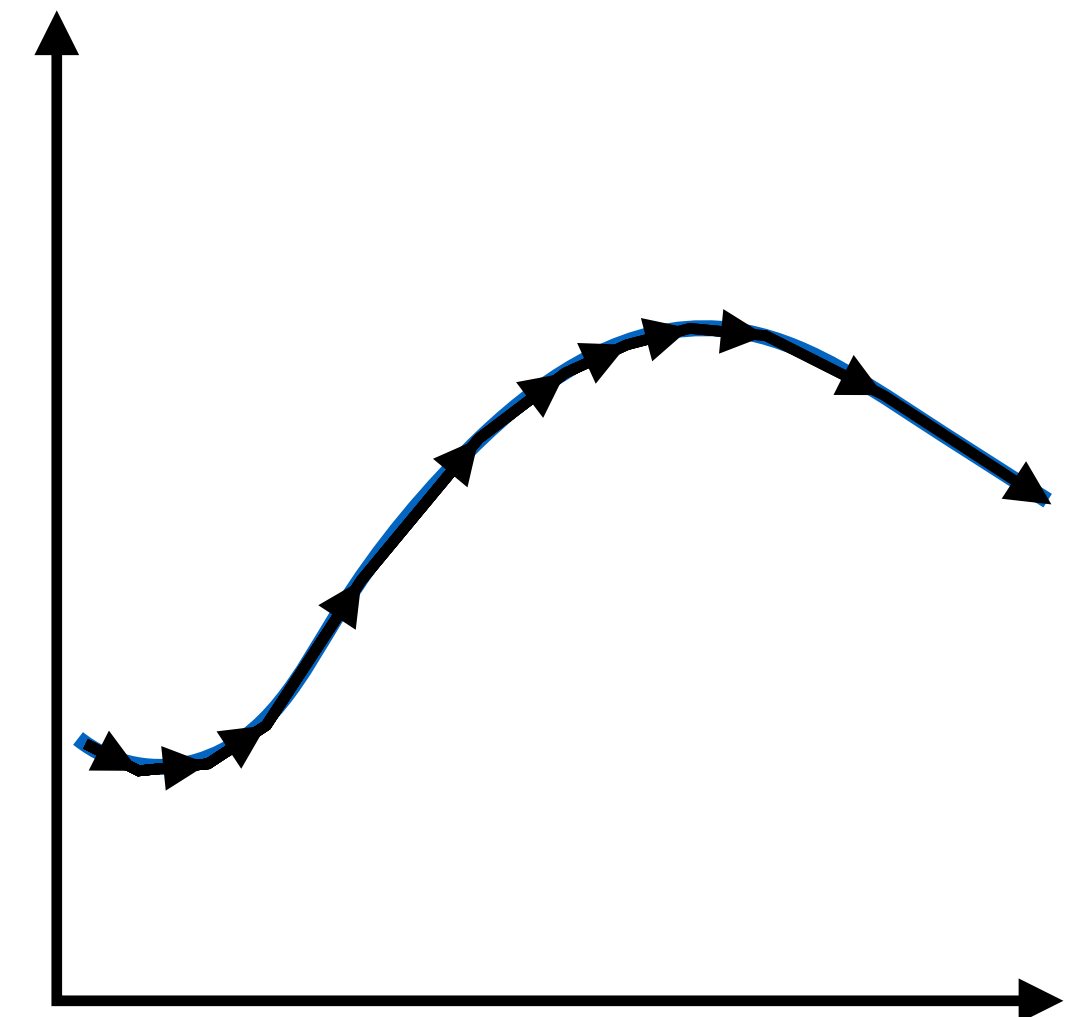
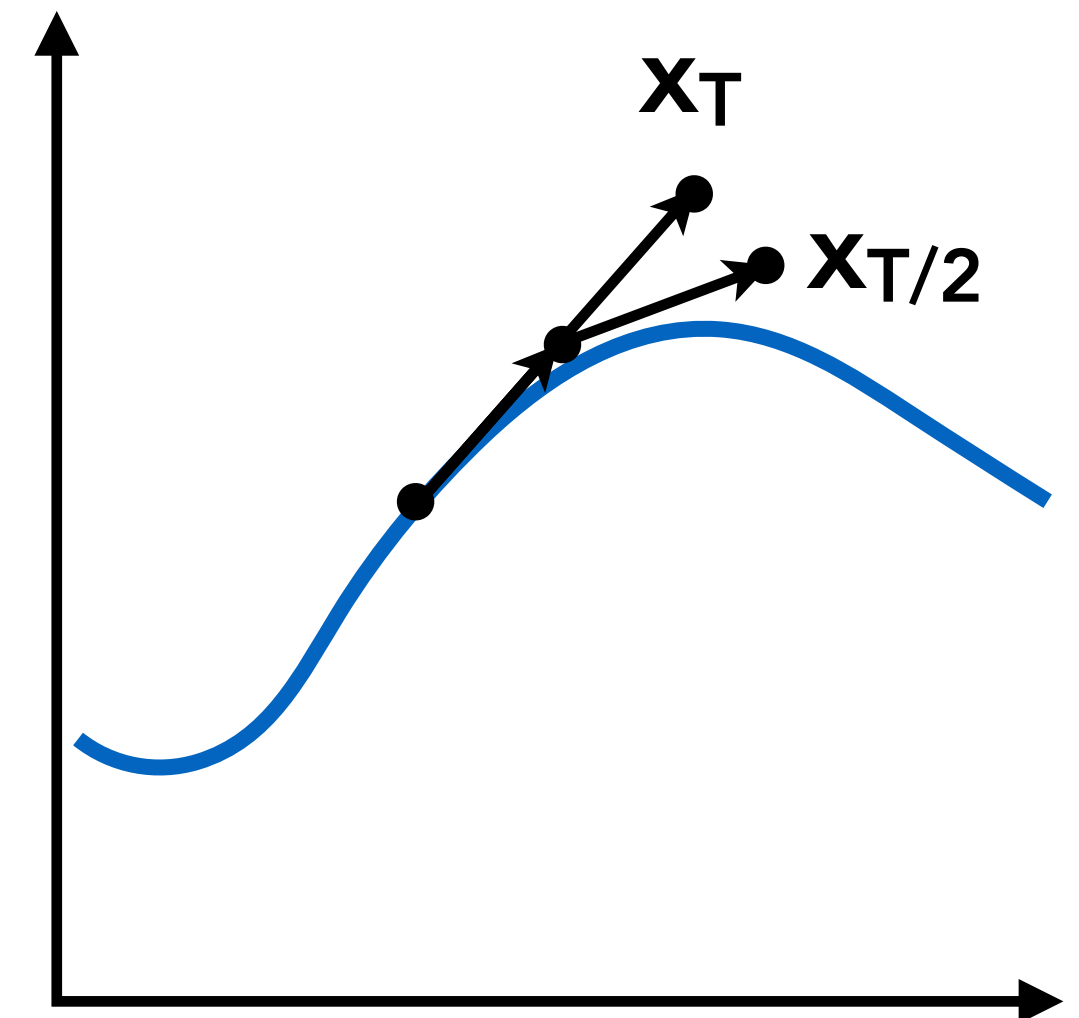
Adaptive Step Size

Adaptive step size

- Technique for choosing step size based on error estimate
- Highly recommended technique
- But may need very small steps!

Repeat until error is below threshold:

- Compute x_T an Euler step, size T
- Compute $x_{T/2}$ two Euler steps, size $T/2$
- Compute error $\|x_T - x_{T/2}\|$
- If (error > threshold) reduce step size and try again



Implicit Euler Method

Implicit methods

- Informally called backward methods
- Use derivatives in the future, for the current step

$$x^{t+\Delta t} = x^t + \Delta t \dot{x}^{t+\Delta t}$$

$$\dot{x}^{t+\Delta t} = \dot{x}^t + \Delta t \ddot{x}^{t+\Delta t}$$

$$\dot{x}^{t+\Delta t} = V(x^{t+\Delta t}, \dot{x}^{t+\Delta t}, t + \Delta t)$$

$$\ddot{x}^{t+\Delta t} = A(x^{t+\Delta t}, \dot{x}^{t+\Delta t}, t + \Delta t)$$

Implicit Euler Method

Implicit methods

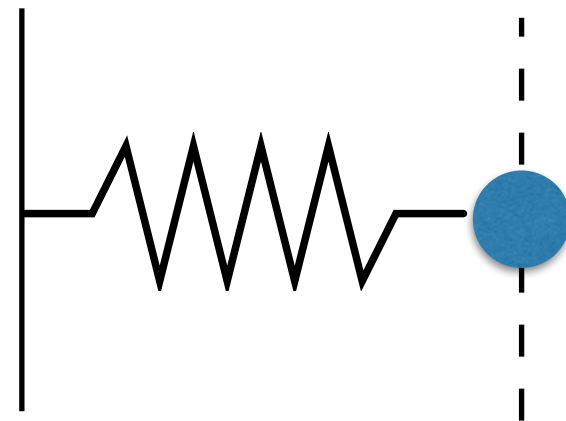
- Informally called backward methods
- Use derivatives in the future, for the current step

$$\dot{\mathbf{x}}^{t+\Delta t} = \dot{\mathbf{x}}^t + \Delta t \mathbf{V}(\mathbf{x}^{t+\Delta t}, \dot{\mathbf{x}}^{t+\Delta t}, t + \Delta t)$$

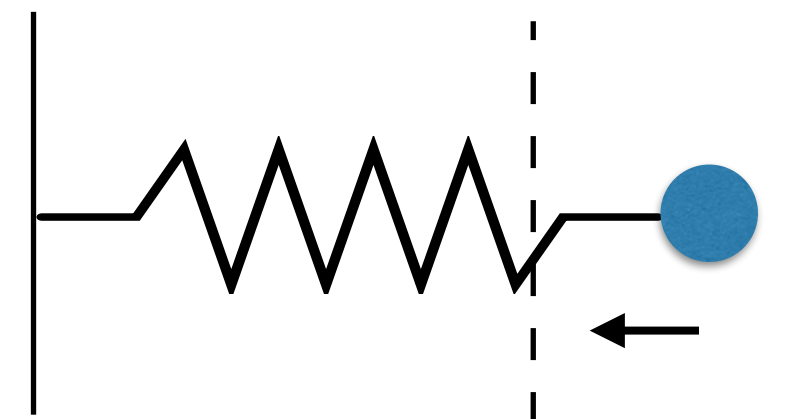
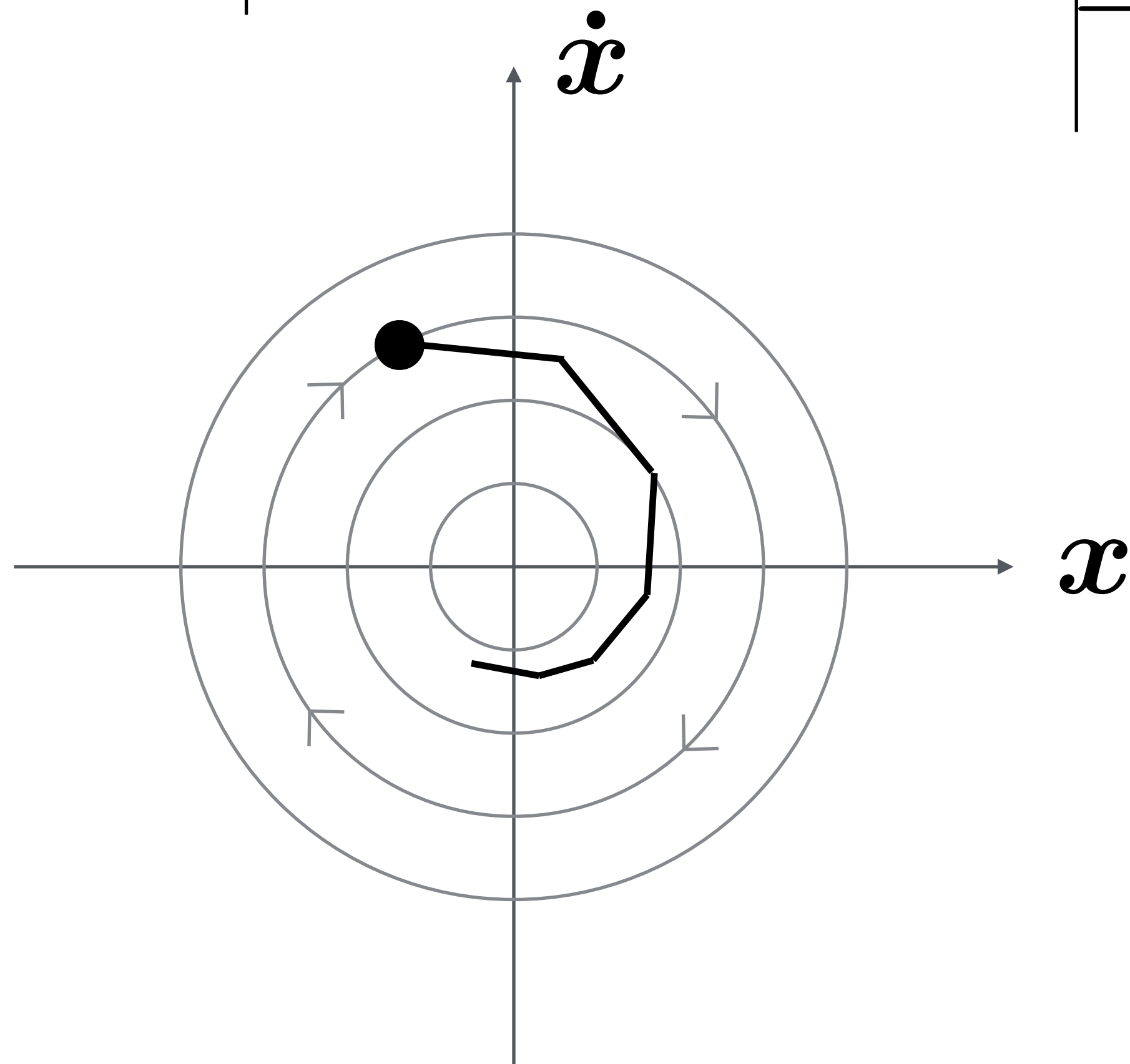
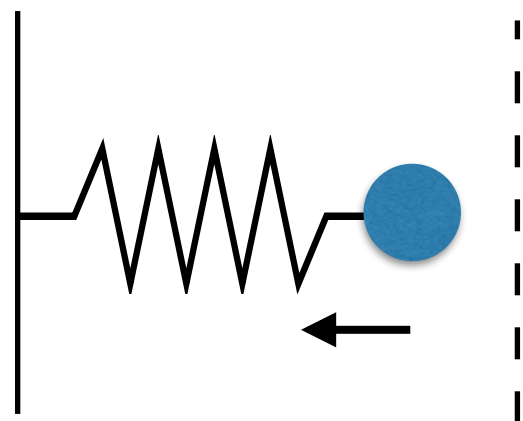
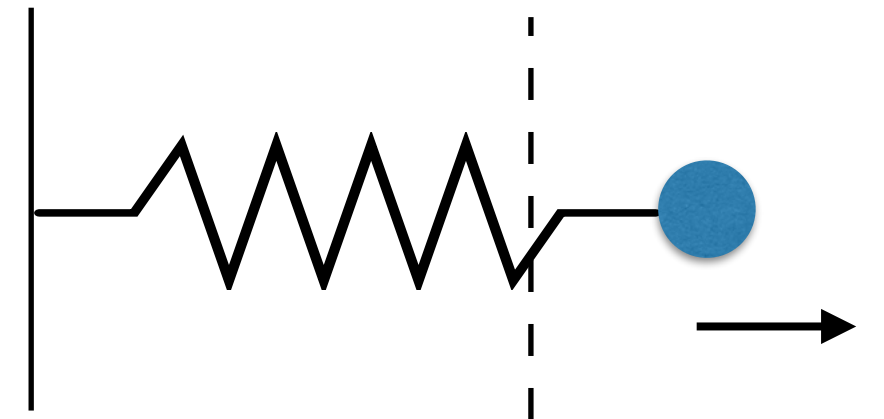
$$\dot{\mathbf{x}}^{t+\Delta t} = \dot{\mathbf{x}}^t + \Delta t \mathbf{A}(\mathbf{x}^{t+\Delta t}, \dot{\mathbf{x}}^{t+\Delta t}, t + \Delta t)$$

- Solve nonlinear problem for $\mathbf{x}^{t+\Delta t}$ and $\dot{\mathbf{x}}^{t+\Delta t}$
- Use root-finding algorithm, e.g. Newton's method
- Can be made unconditionally stable

"Spiraling In"



$$f = -k_s x$$



Position-Based / Verlet Integration

Idea:

- After modified Euler forward-step, constrain positions of particles to prevent divergent, unstable behavior
- Use constrained positions to calculate velocity
- Both of these ideas will dissipate energy, stabilize

Pros / cons

- Fast and simple
- Not physically based, dissipates energy (error)
- Highly recommended (assignment)

Position-Based / Verlet Integration

Algorithm 1 Position-based dynamics

```
1: for all vertices  $i$  do
2:   initialize  $\mathbf{x}_i = \mathbf{x}_i^0$ ,  $\mathbf{v}_i = \mathbf{v}_i^0$ ,  $w_i = 1/m_i$ 
3: end for
4: loop
5:   for all vertices  $i$  do  $\mathbf{v}_i \leftarrow \mathbf{v}_i + \Delta t w_i \mathbf{f}_{\text{ext}}(\mathbf{x}_i)$ 
6:   for all vertices  $i$  do  $\mathbf{p}_i \leftarrow \mathbf{x}_i + \Delta t \mathbf{v}_i$ 
7:   for all vertices  $i$  do  $\text{genCollConstraints}(\mathbf{x}_i \rightarrow \mathbf{p}_i)$ 
8:   loop solverIteration times
9:      $\text{projectConstraints}(C_1, \dots, C_{M+M_{\text{Coll}}}, \mathbf{p}_1, \dots, \mathbf{p}_N)$ 
10:  end loop
11:  for all vertices  $i$  do
12:     $\mathbf{v}_i \leftarrow (\mathbf{p}_i - \mathbf{x}_i) / \Delta t$ 
13:     $\mathbf{x}_i \leftarrow \mathbf{p}_i$ 
14:  end for
15:   $\text{velocityUpdate}(\mathbf{v}_1, \dots, \mathbf{v}_N)$ 
16: end loop
```

Position-Based Simulation Methods in Computer Graphics
Bender, Müller, Macklin, Eurographics 2015

Particle Systems

(More Examples)

Particle Systems

Model dynamical systems as collections of large numbers of particles

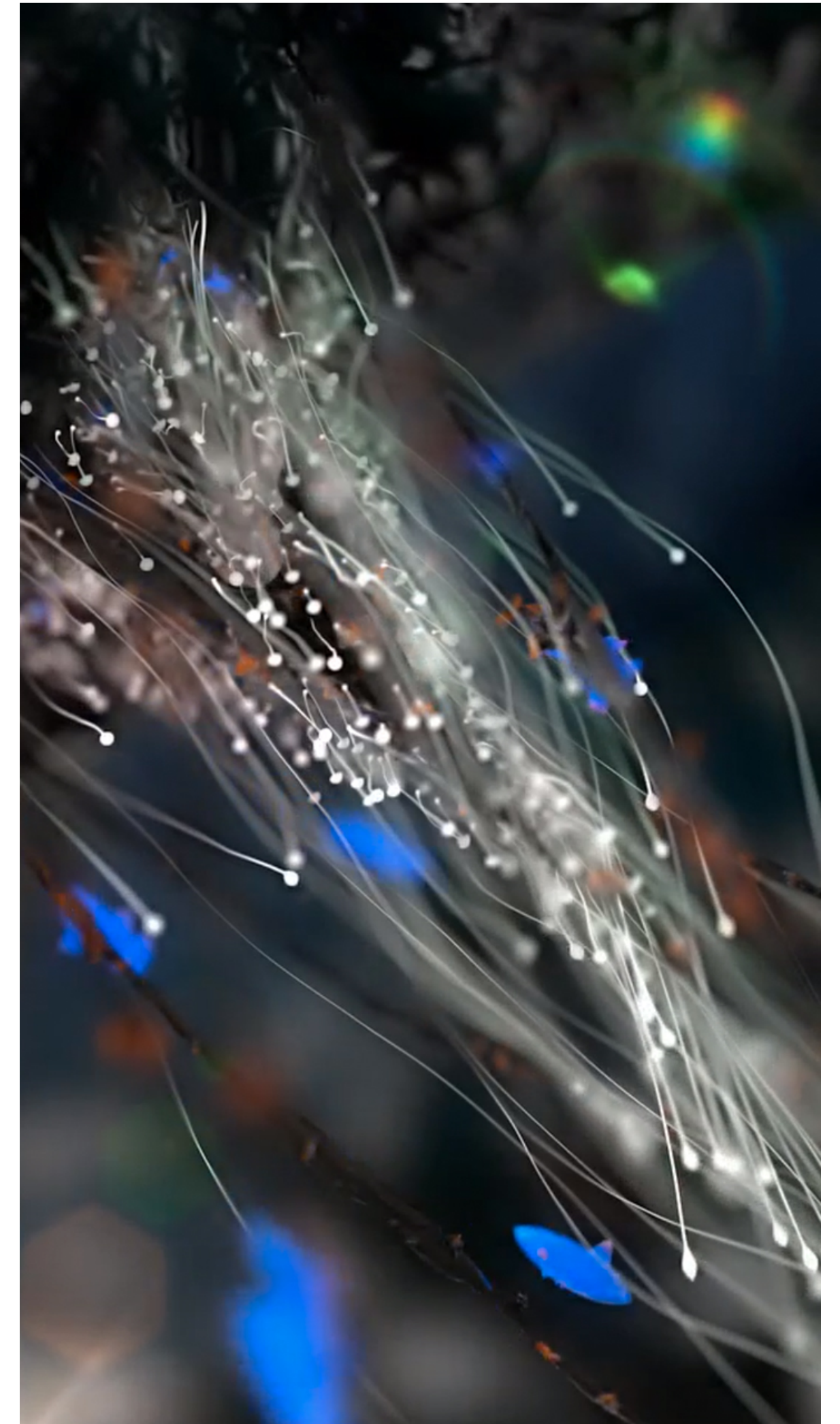
Each particle's motion is defined by a set of physical (or non-physical) forces

Popular technique in graphics and games

- Easy to understand, implement
- Scalable: fewer particles for speed, more for higher complexity

Challenges

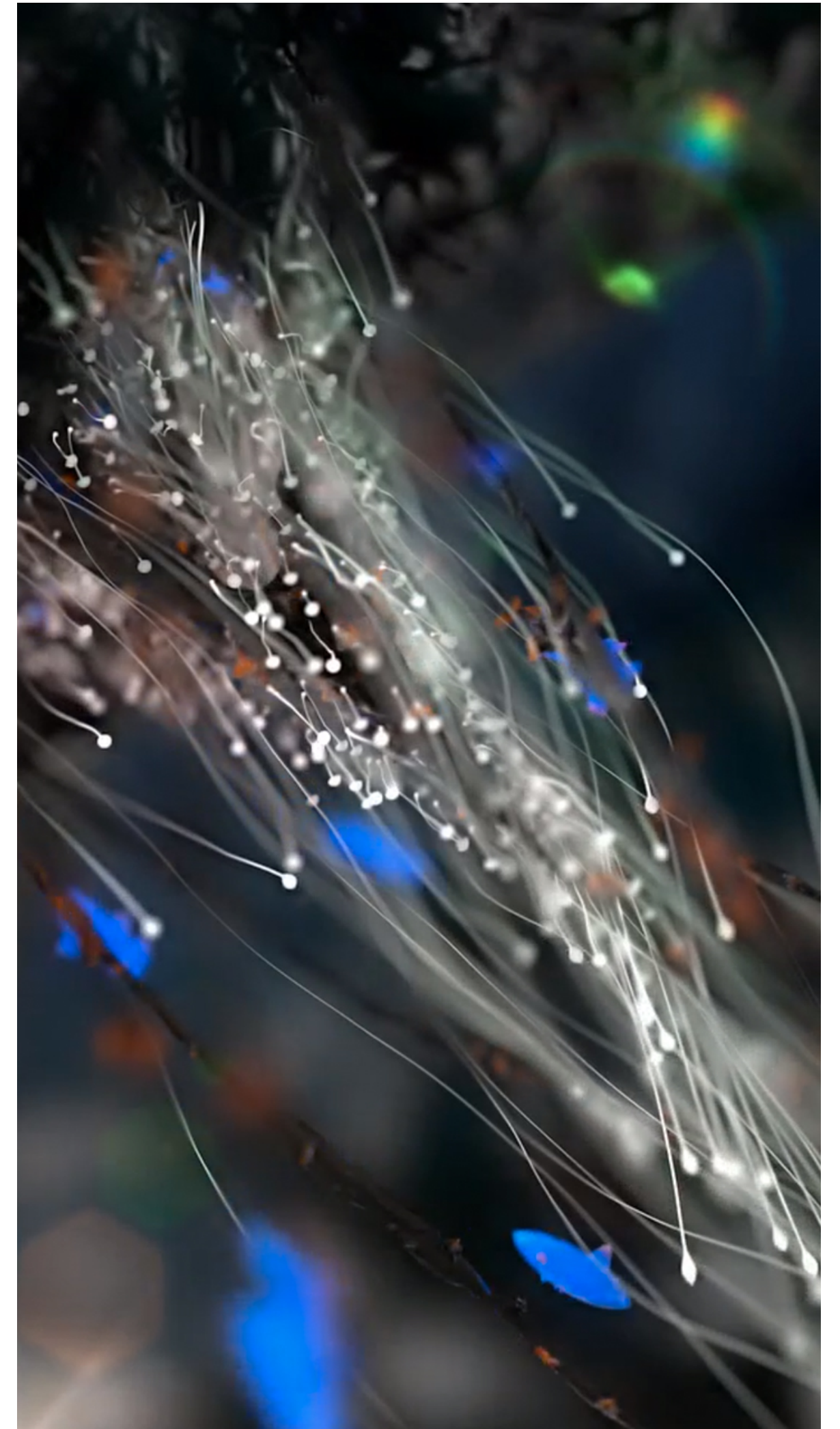
- May need *many* particles (e.g. fluids)
- May need acceleration structures (e.g. to find nearest particles for interactions)



Particle System Animations

For each frame in animation

- [If needed] Create new particles
- Calculate forces on each particle
- Update each particle's position and velocity
- [If needed] Remove dead particles
- Render particles



Particle System Forces

Attraction and repulsion forces

- Gravity, electromagnetism, ...
- Springs, propulsion, ...

Damping forces

- Friction, air drag, viscosity, ...

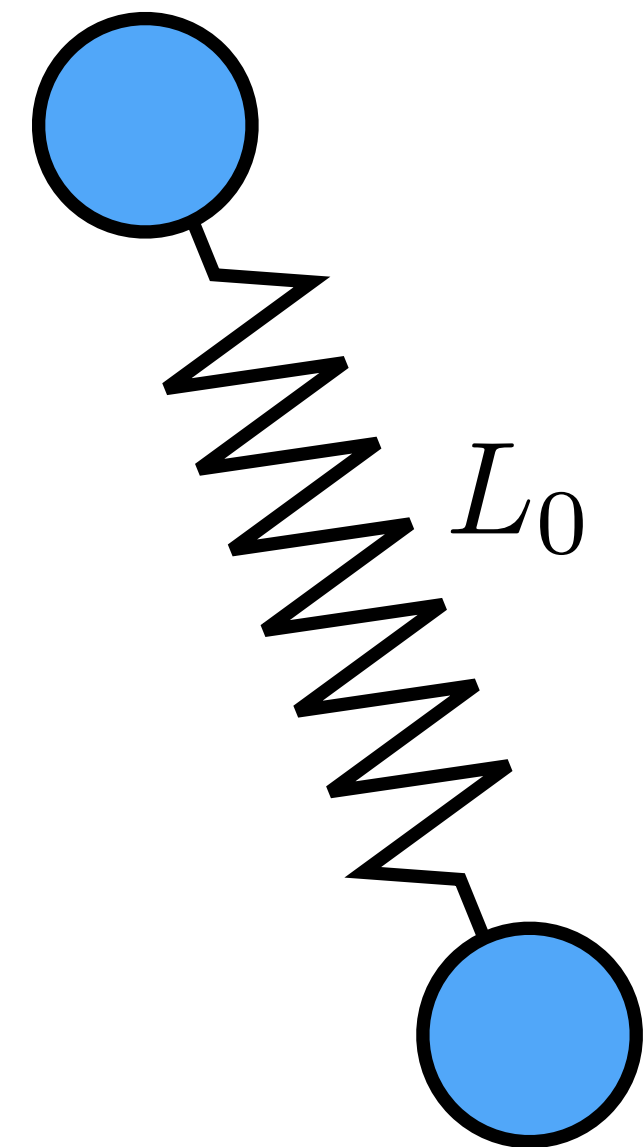
Collisions

- Walls, containers, fixed objects, ...
- Dynamic objects, character body parts, ...

Already Discussed Springs

Internally-damped non-zero length spring

$$\begin{aligned} \mathbf{f}_{a \rightarrow b} = & k_s \frac{\mathbf{b} - \mathbf{a}}{\|\mathbf{b} - \mathbf{a}\|} (\|\mathbf{b} - \mathbf{a}\| - l) \\ & - k_d \frac{\mathbf{b} - \mathbf{a}}{\|\mathbf{b} - \mathbf{a}\|} (\dot{\mathbf{b}} - \dot{\mathbf{a}}) \cdot \frac{\mathbf{b} - \mathbf{a}}{\|\mathbf{b} - \mathbf{a}\|} \end{aligned}$$



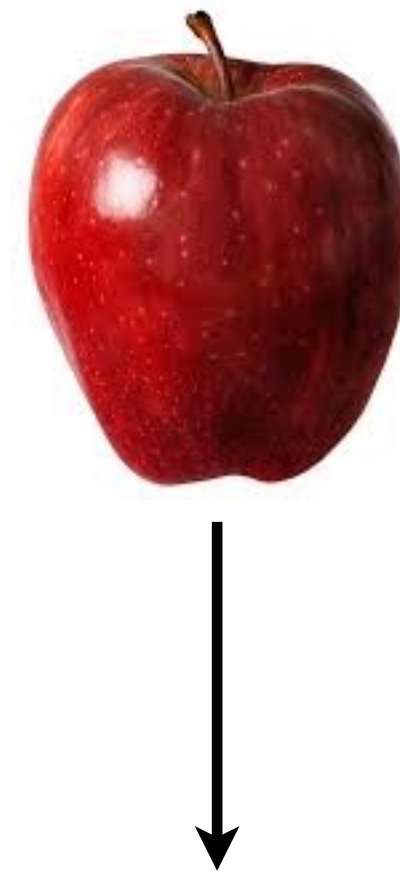
Simple Gravity

Gravity at earth's surface due to earth

- $F = -mg$
- m is mass of object
- g is gravitational acceleration,
 $g = -9.8\text{m/s}^2$

$$F_g = -mg$$

$$g = (0, 0, -9.8) \text{ m/s}^2$$



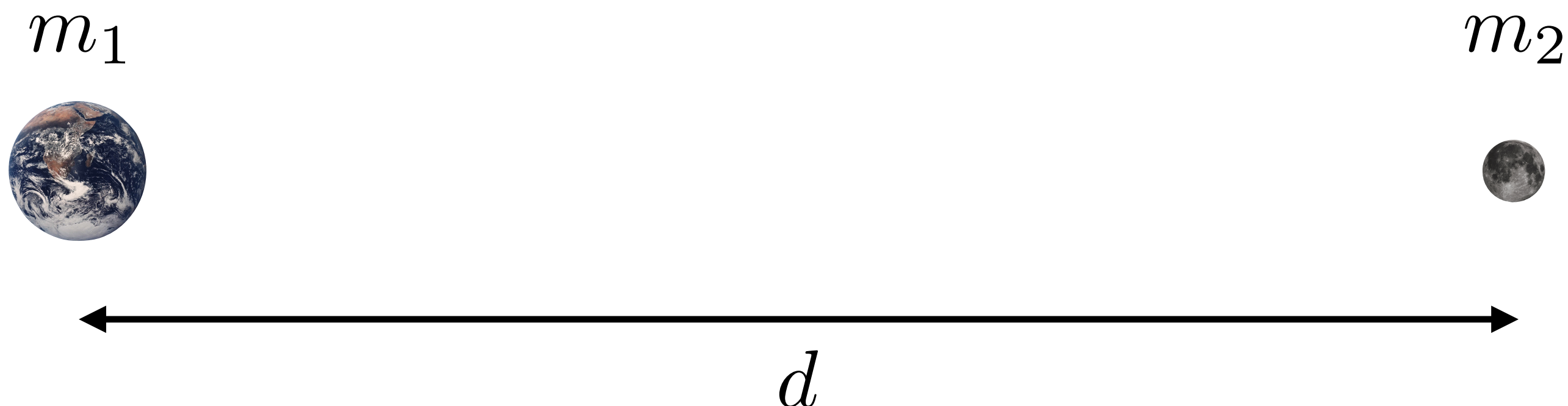
Gravitational Attraction

Newton's universal law of gravitation

- Gravitational pull between particles

$$F_g = G \frac{m_1 m_2}{d^2}$$

$$G = 6.67428 \times 10^{-11} \text{ Nm}^2\text{kg}^{-2}$$

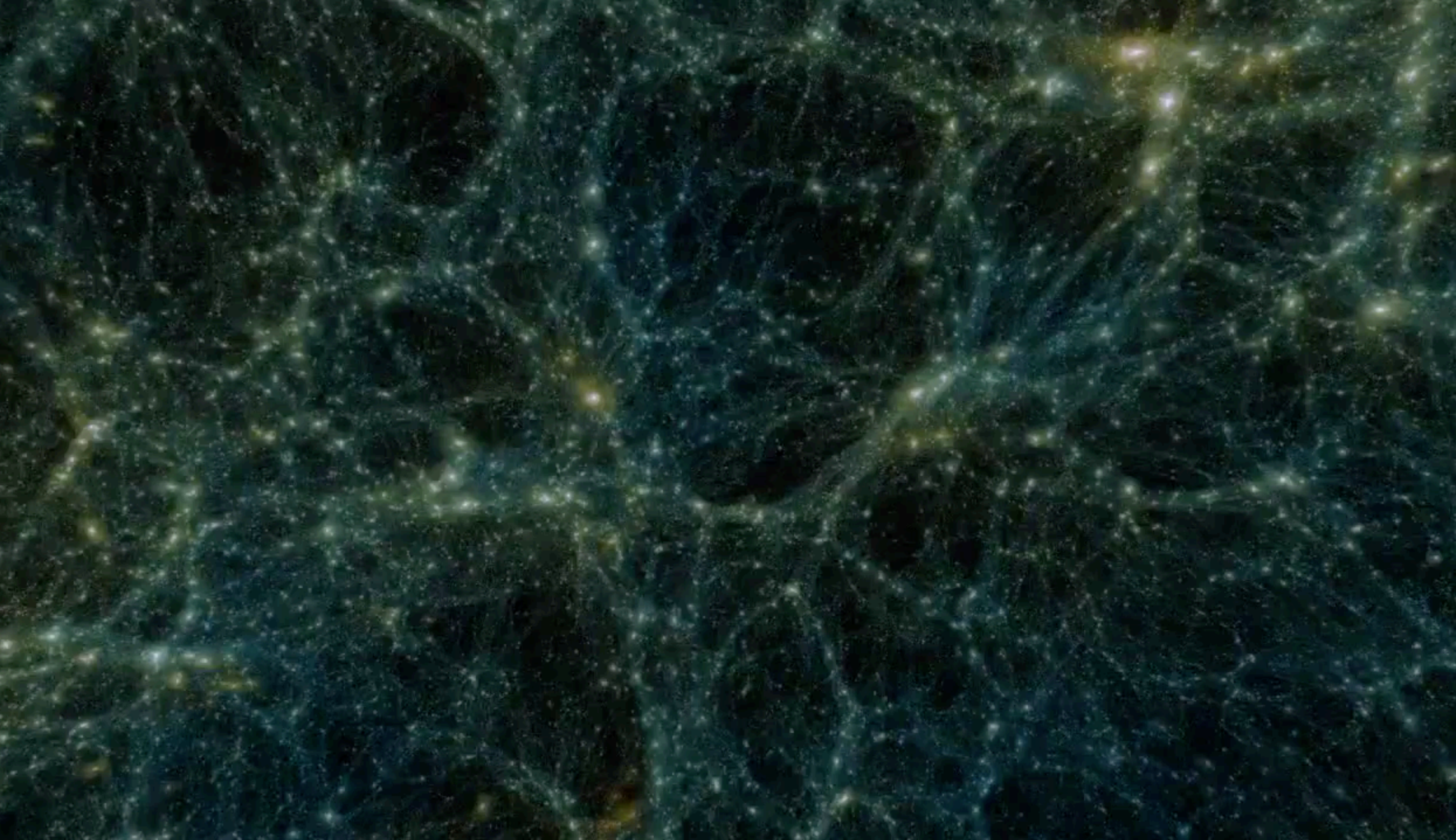


Example: Galaxy Simulation



Disk galaxy simulation, NASA Goddard

Example: Cosmological Simulation



CS184/284A Tomoaki et al - v^2 GC simulation of dark matter (~ 1 trillion particles)

Ren Ng

Example: Particle-Based Fluids



Macklin and Müller, Position Based Fluids

Example: Flocking Birds



Simulated Flocking as an ODE

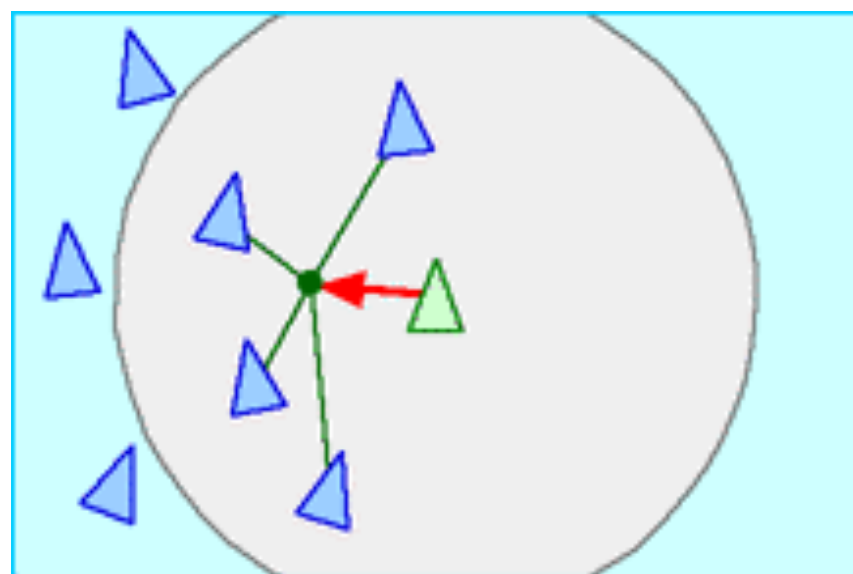
Model each bird as a particle

Subject to very simple forces:

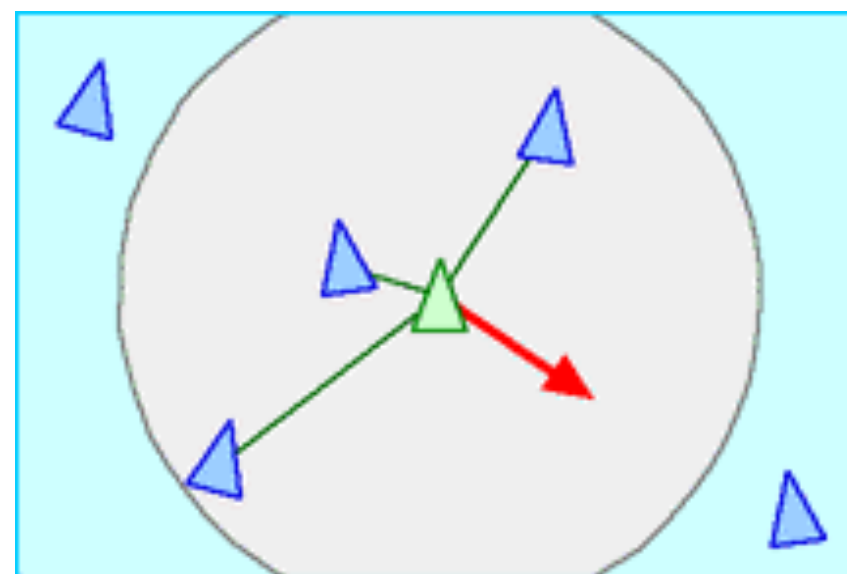
- attraction to center of neighbors
- repulsion from individual neighbors
- alignment toward average trajectory of neighbors

Simulate evolution of large particle system numerically

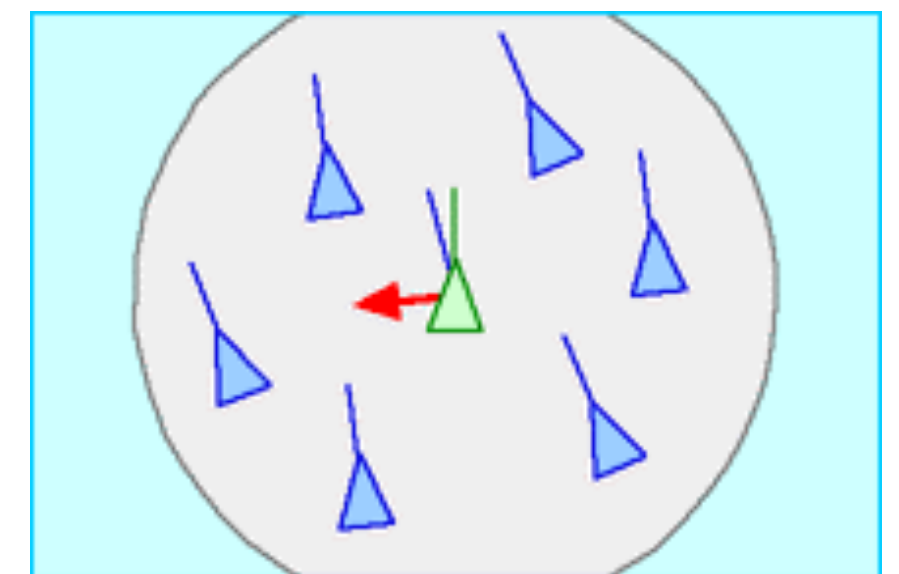
Emergent complex behavior (also seen in fish, bees, ...)



attraction



repulsion



alignment

Example: Crowds



Where are the bottlenecks in a building plan?

Example: Crowds + “Rock” Dynamics



Suggested Reading

Physically Based Modeling: Principles and Practice

- Andy Witkin and David Baraff
- <http://www-2.cs.cmu.edu/~baraff/sigcourse/index.html>

Numerical Recipes in C++

- Chapter 16

Any good text on integrating ODE's

Just Scratching the Surface...

Physical simulation is a huge field in graphics, engineering, science

Today: intro to particle systems, solving ODEs

Partial differential equations

- Diffusion equation, heat equation, ...
- Used in graphics for liquids, smoke, fire, etc.

Rigid body

Simulation of sound

...

Things to Remember

Physical simulation = mathematical modeling of dynamical systems & solution by numerical integration

Particle systems

- **Flexible force modeling, e.g. spring-mass systems, gravitational attraction, fluids, flocking behavior**
- **Newtonian equations of motion = ODEs**
- **Solution by numerical integration of ODEs: Explicit Euler, Implicit Euler, Adaptive, Position-Based / Verlet**
- **Error and instability, methods to combat instability**

Acknowledgments

Many thanks to James O'Brien, Keenan Crane and Tom Funkhouser for lecture resources.