CS354R

DR SARAH ABRAHAM

PHYSICS OVERVIEW

GAME PHYSICS – BASIC AREAS

- Point Masses
  - Particle simulation
  - Collision response
- Rigid Bodies
  - Extensions to non-points
- Soft Body Dynamic Systems
- Articulated Systems and Constraints
- Collision Detection

GAME PHYSICS – BASIC AREAS

- **Point Masses**
  - **Particle simulation**
  - **Collision response**
- Rigid Bodies
  - Extensions to non-points
- Soft Body Dynamic Systems
- Articulated Systems and Constraints
- Collision Detection

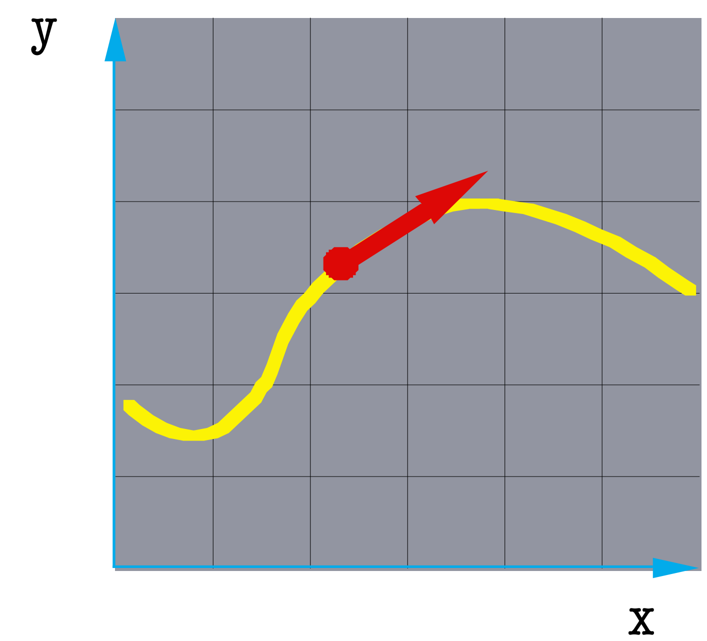
PHYSICS ENGINES

- API for collision detection
- API for kinematics (motion but no forces)
- API for dynamics
- Examples:
  - Box2d
  - Bullet
  - ODE (Open Dynamics Engine)
  - PhysX
  - Havoc
  - Many others!

PARTICLE DYNAMICS AND PARTICLE SYSTEMS

- A particle system is a collection of point masses that obeys some physical laws (e.g, gravity, heat convection, spring behaviors, etc)
- Particle systems can be used to simulate all sorts of physical phenomena:
  - Fluids
  - Cloth
  - Galaxies
  - Other stuff
- So let's consider a single particle…

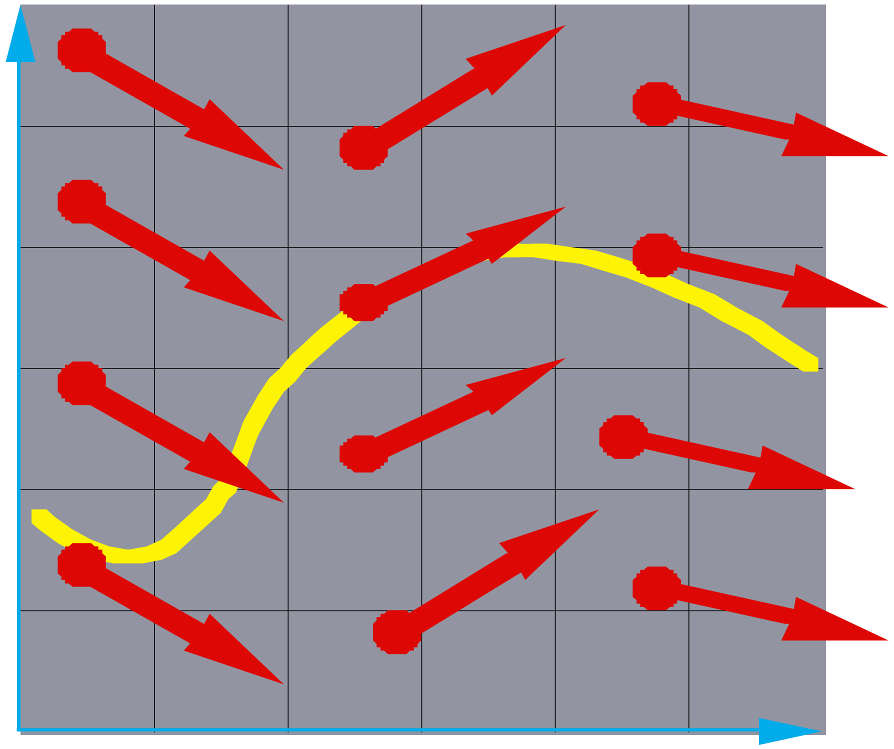
PARTICLE IN A FLOW FIELD



- Consider a single particle that has:
  - Position: $\vec{x} = \begin{bmatrix} x \\ y \end{bmatrix}$
  - Velocity: $\vec{v} = \dot{x} = \dfrac{d\vec{x}}{dt} = \begin{bmatrix} \dfrac{dx}{dt} \\ \dfrac{dy}{dt} \end{bmatrix}$
- Suppose the velocity is actually dictated by some driving function **g**: $\dot{x} = g(\vec{x}, t)$

VECTOR FIELDS

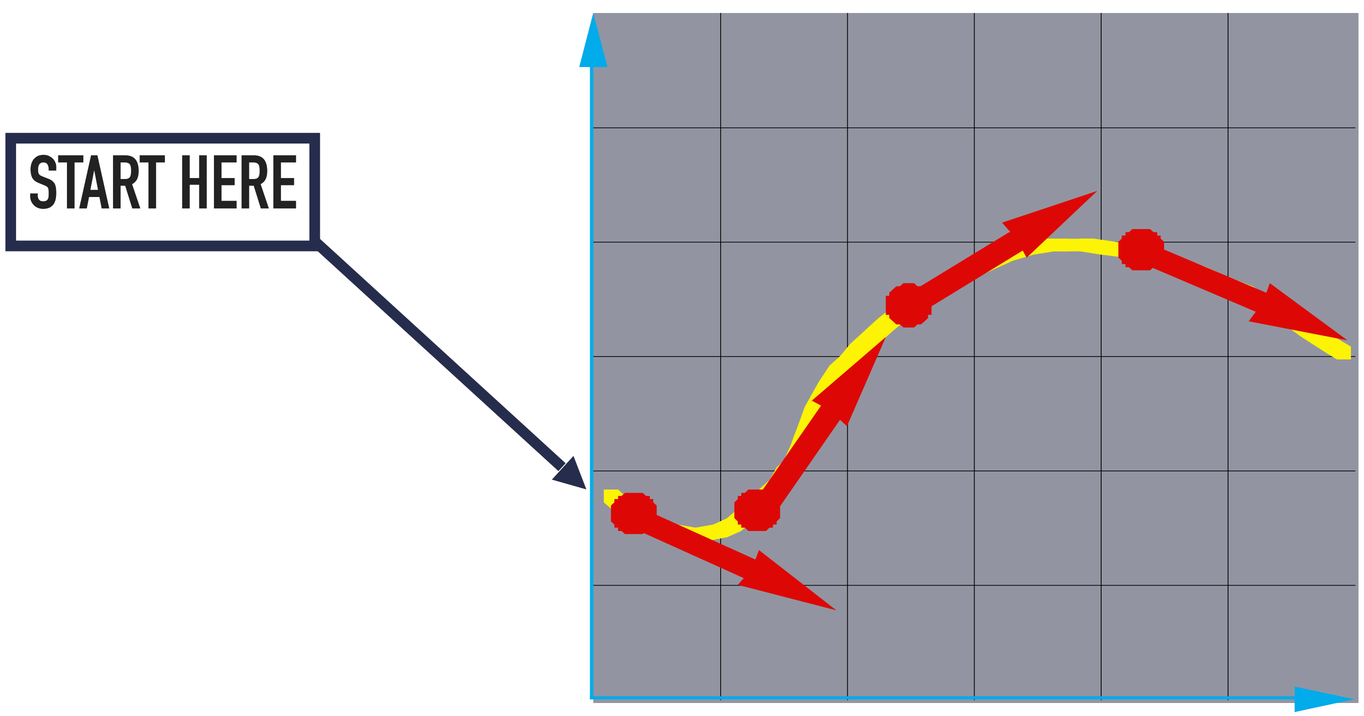
- At any moment in time, the function **g** defines a vector field over **x**:



- How can we use this to determine where we are in the field?

DIFFERENTIAL EQUATIONS AND INTEGRAL CURVES

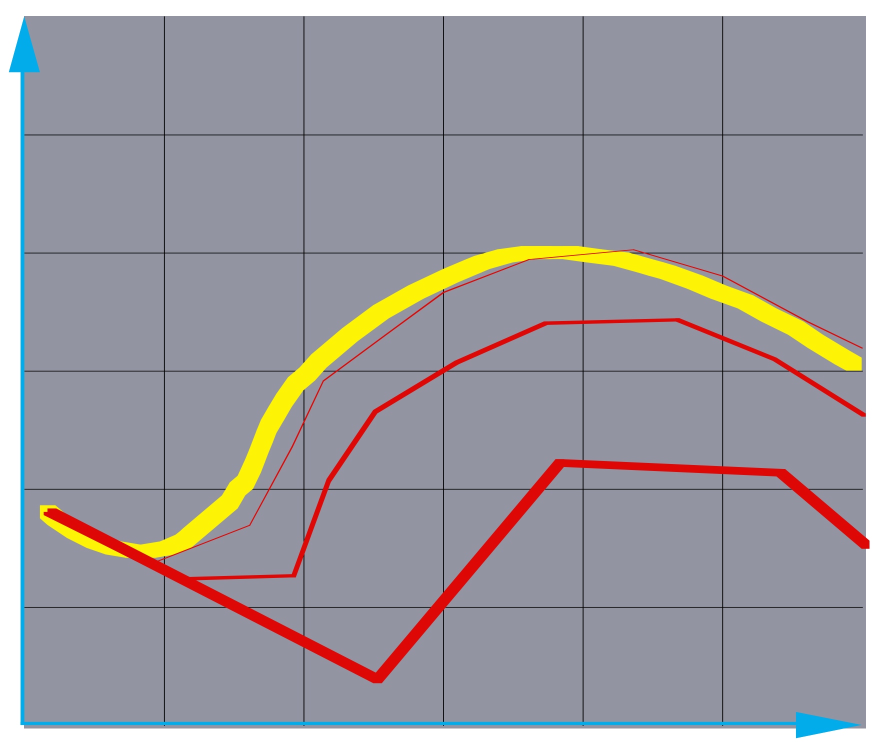
- The equation: $\dot{x} = g(\overrightarrow{x}, t)$ is actually a **first order differential equation**.
- We can solve for **x** through time by starting at an initial point and stepping along the vector field:



- This is called an **initial value problem** and the solution is called an **integral curve**.

EULER'S METHOD

- Choose a time step, Δt, and take linear steps along the flow: $\vec{\mathbf{x}}(t+\Delta t) = \vec{\mathbf{x}}(t) + \Delta t \cdot \dot{\vec{\mathbf{x}}}(t) = \vec{\mathbf{x}}(t) + \Delta t \cdot g(\vec{\mathbf{x}}, t)$
- Writing as a time iteration: $\vec{\mathbf{x}}^{i+1} = \vec{x}^{i} + \Delta t \cdot \vec{\mathbf{v}}^{i}$
- This approach is called **Euler's method** and looks like:



ADDING FORCES AND MASS

- Now consider a particle in a force field **f**
- In this case, the particle has:
  - Mass: m
  - Acceleration: $\overrightarrow{a} \equiv \ddot{x} = \dfrac{d\overrightarrow{v}}{dt} = \dfrac{d^2\overrightarrow{x}}{dt^2}$
- The particle obeys Newton's law: $\overrightarrow{f} = m\overrightarrow{a} = m\ddot{x}$
- The force field **f** can in general depend on the position and velocity of the particle as well as time.
- Thus, with some rearrangement, we end up with: $\ddot{x} = \dfrac{\overrightarrow{f}(\overrightarrow{x}, \dot{x}, t)}{m}$

SECOND ORDER EQUATIONS

- This equation: $\ddot{x} = \dfrac{\vec{f}(\vec{x}, \dot{x}, t)}{m}$ is a **second order differential equation**.
- Our solution method works on first order differential equations.
- We can rewrite this as:

$$\begin{bmatrix} \dot{x} = \vec{v} \\ \dot{v} = \dfrac{\vec{f}(\vec{x}, \vec{v}, t)}{m} \end{bmatrix}$$

where we have added a new variable **v** to get a pair of coupled first order equations.

PHASE SPACE

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

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

- Taking the time derivative to make a 6-vector velocity in phase space

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

- A vanilla 1st-order differential equation

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

DIFFERENTIAL EQUATION SOLVER

- Starting with:

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

- Applying Euler's method:

$$\vec{\mathbf{x}}(t+\Delta t) = \vec{\mathbf{x}}(t) + \Delta t \cdot \dot{\mathbf{x}}(t)$$
$$\dot{\mathbf{x}}(t+\Delta t) = \dot{\mathbf{x}}(t) + \Delta t \cdot \ddot{\mathbf{x}}(t)$$

- And making substitutions:

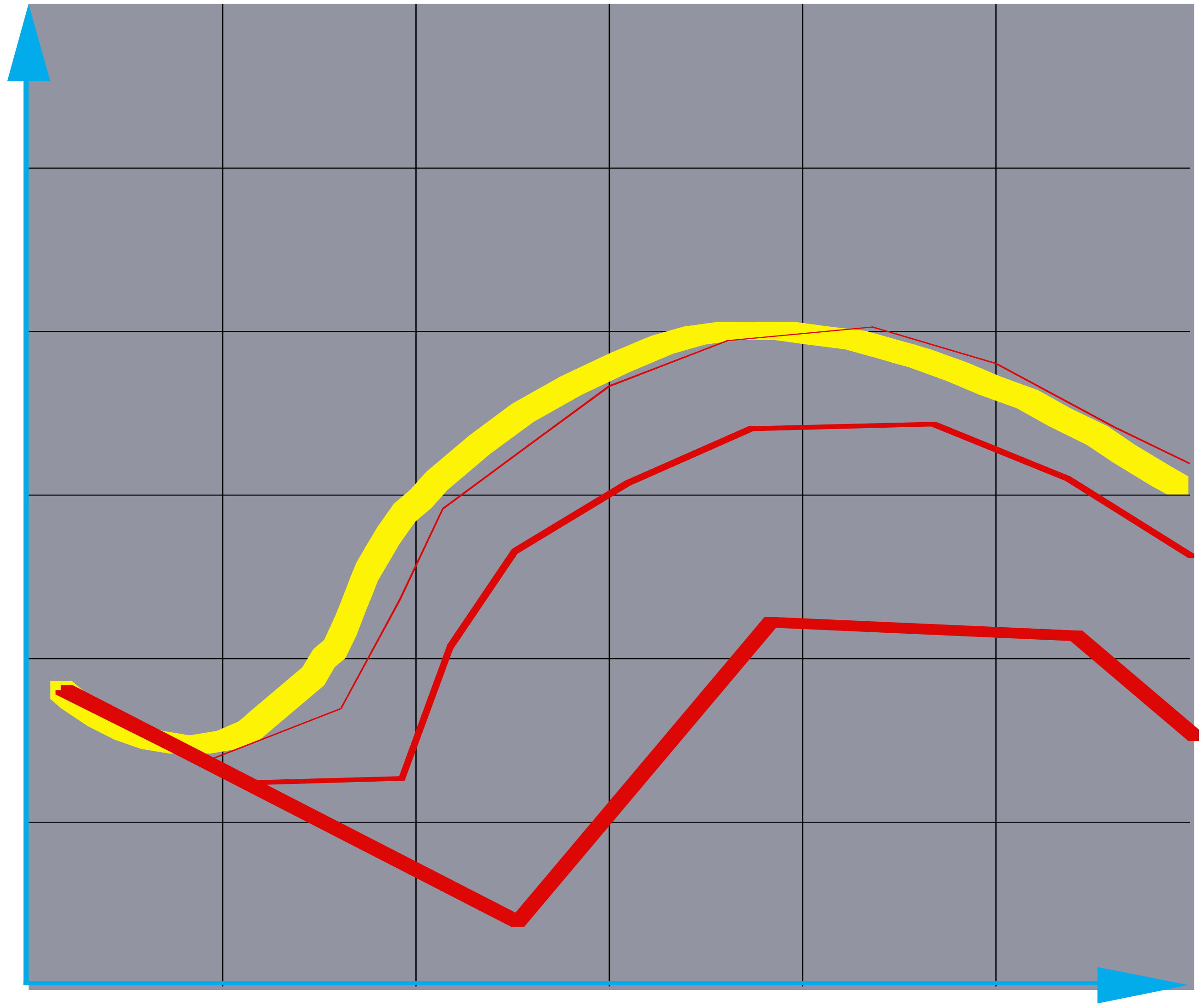
$$\vec{\mathbf{x}}(t+\Delta t) = \vec{\mathbf{x}}(t) + \Delta t \cdot \vec{\mathbf{v}}(t)$$
$$\dot{\mathbf{x}}(t+\Delta t) = \dot{\mathbf{x}}(t) + \Delta t \cdot \vec{\mathbf{f}}(\vec{\mathbf{x}},\dot{\mathbf{x}},t)\big/m$$

- Writing this as an iteration:

$$\vec{\mathbf{x}}^{\,i+1} = \vec{x}^{\,i} + \Delta t \cdot \vec{\mathbf{v}}^{\,i}$$
$$\vec{\mathbf{v}}^{\,i+1} = \vec{\mathbf{v}}^{\,i} + \Delta t \cdot \frac{\vec{\mathbf{f}}^{\,i}}{m}$$

- (Still performs poorly for large Δt)

REMEMBER THIS GRAPH?



TIME STEP MATTERS

https://www.reddit.com/r/gaming/comments/9yg41t/nothing_to_see_here_just_some_good_old_bethesda/

EULER'S METHOD PROPERTIES

- Properties:
  - Simplest numerical method
  - Bigger steps, bigger errors. Error ~ $O(\Delta t^2)$.
- Need to take pretty small steps, so not very efficient
- Better methods exist:
  - Runge-Kutta
  - Implicit Integration
  - Semi-implicit Euler
  - Verlet
- These methods range in terms of complexity and computation

SO LET'S TALK VERLET…

- Verlet integration is frequently used in video games
  - Good numerical stability
  - Good booking-keeping properties
  - Good performance (as fast as forward Eulerian!)
- Verlet flavors:
  - Position Verlet
    - Uses 2 previous positions to obtain next position without using a velocity
  - Leapfrog
    - Alternately updates to position and velocity
  - Velocity Verlet
    - Similar to Leapfrog but updates position and velocity in the same timestep

VERLET

- Consider Forward Euler:

$$\vec{v}^{i+1} = \vec{v}^{i} + a^{i}\Delta t$$

$$\vec{x}^{i+1} = \vec{x}^{i} + \vec{v}^{i+1}\Delta t$$

- Substitute velocity calculation into position calculation:

$$\vec{x}^{i+1} = \vec{x}^{i} + (\vec{v}^{i} + a^{i}\Delta t)\Delta t$$

$$\vec{x}^{i+1} = \vec{x}^{i} + \vec{v}^{i}\Delta t + a^{i}\Delta t^{2}$$

POSITION VERLET INTEGRATION

- Does not directly store velocity

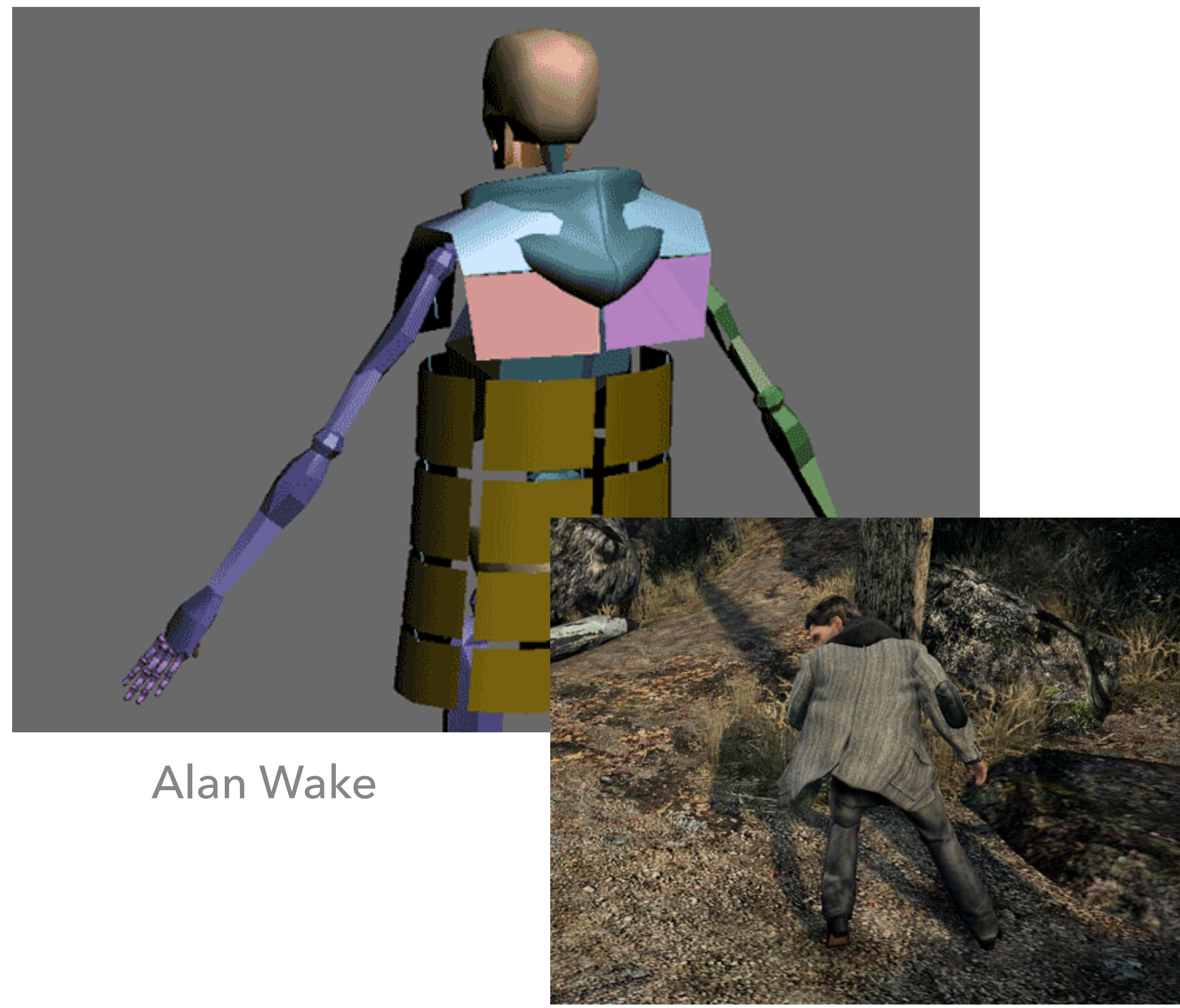
$$\overrightarrow{x}^{i+1} = \overrightarrow{x}^{i} + (\overrightarrow{x}^{i} - \overrightarrow{x}^{i-1}) + a^i \Delta t^2$$

$$\overrightarrow{x}^{i-1} = \overrightarrow{x}^{i}$$

- What do we need when i = 0?

HISTORY OF SIMULATION IN GAMES

Hitman: Codename 47



Alan Wake

SIMULATION IN GAMES

Last Guardian

https://www.youtube.com/watch?v=HacuU5kKae4

REPRESENTING A PARTICLE

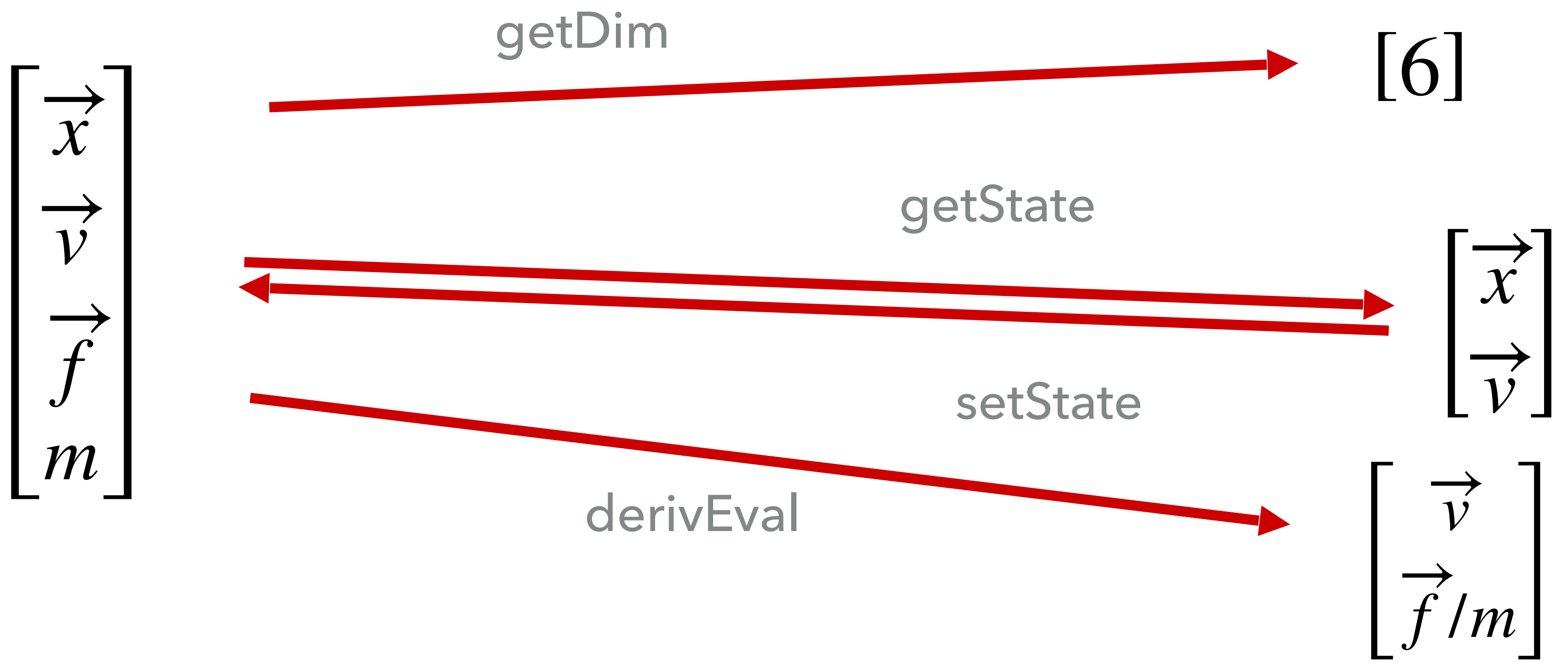
- How do we represent a particle in code?

PARTICLE STRUCTURE

$$\begin{bmatrix}\vec{x}\\ \vec{v}\\ \vec{f}\\ m\end{bmatrix} \begin{array}{l} \leftarrow \text{position}\\ \leftarrow \text{velocity}\\ \leftarrow \text{force accumulator}\\ \leftarrow \text{mass}\end{array}$$

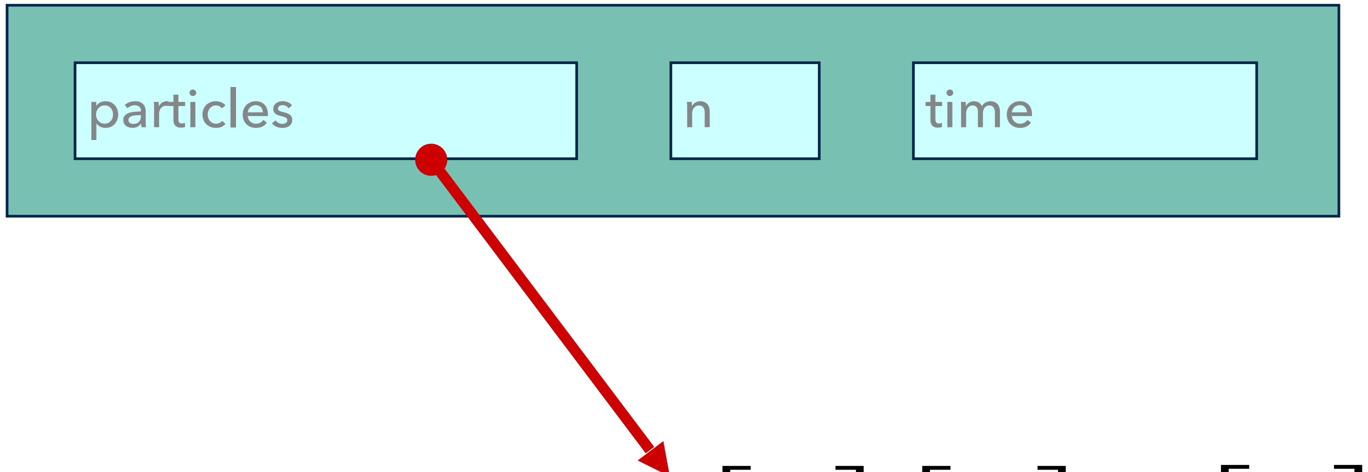
Position in phase space

SINGLE PARTICLE SOLVER INTERFACE



PARTICLE SYSTEMS

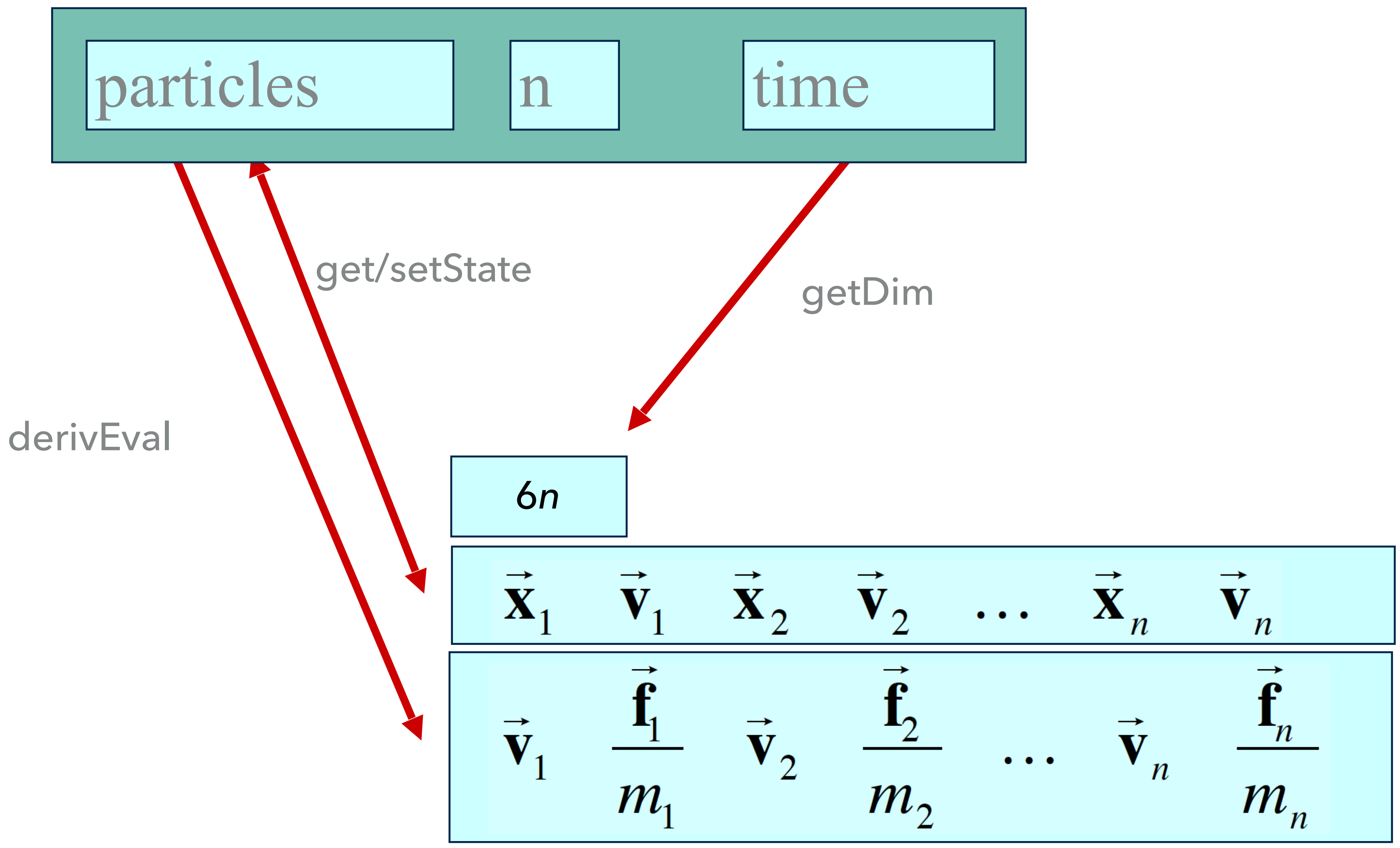
- In general, we have a particle system consisting of n particles to be managed over time:



$$\begin{bmatrix} \vec{x}_1 \\ \vec{v}_1 \\ \vec{f}_1 \\ m_1 \end{bmatrix} \begin{bmatrix} \vec{x}_2 \\ \vec{v}_2 \\ \vec{f}_2 \\ m_2 \end{bmatrix} \cdots \begin{bmatrix} \vec{x}_n \\ \vec{v}_n \\ \vec{f}_n \\ m_n \end{bmatrix}$$

PARTICLE SYSTEM SOLVER INTERFACE

- For n particles, the solver interface now looks like:



PARTICLE SYSTEM DIFF. EQ. SOLVER

- We can determine the evolution of a particle system using the Euler method or another solver of choice:

$$\begin{bmatrix} \vec{\mathbf{x}}_1^{i+1} \\ \vec{\mathbf{v}}_1^{i+1} \\ \vdots \\ \vec{\mathbf{x}}_n^{i+1} \\ \vec{\mathbf{v}}_n^{i+1} \end{bmatrix} = \begin{bmatrix} \vec{\mathbf{x}}_1^{i} \\ \vec{\mathbf{v}}_1^{i} \\ \vdots \\ \vec{\mathbf{x}}_n^{i} \\ \vec{\mathbf{v}}_n^{i} \end{bmatrix} + \Delta t \begin{bmatrix} \vec{\mathbf{v}}_1^{i} \\ \vec{\mathbf{f}}_1^{i} / m_1 \\ \vdots \\ \vec{\mathbf{v}}_n^{i} \\ \vec{\mathbf{f}}_n^{i} / m_n \end{bmatrix}$$

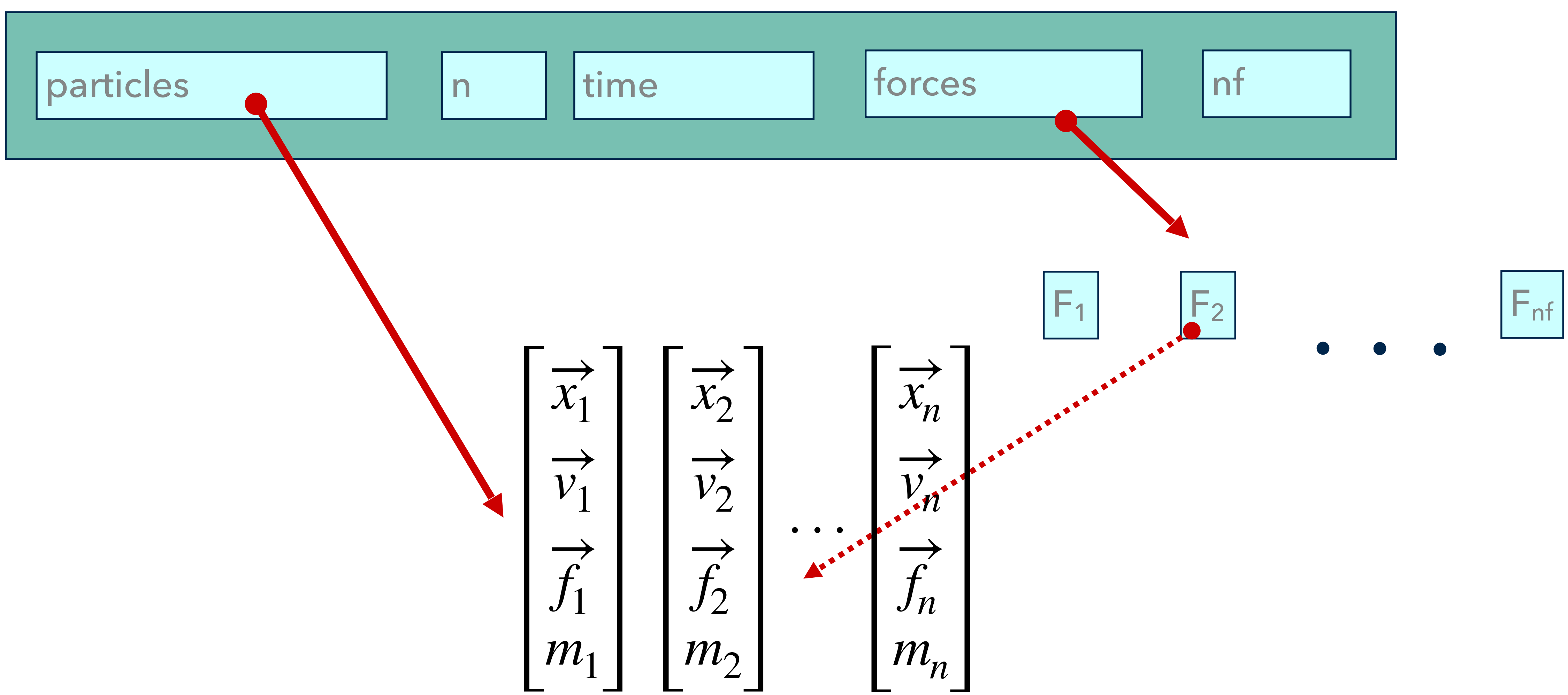
FORCES

- Each particle can experience a force which sends it on its way
- Forces can be:
  - Constant (gravity)
  - Position/time dependent (force fields)
  - Velocity-dependent (drag)
  - Combinations (damped springs)

- How do we compute the net force on a particle?

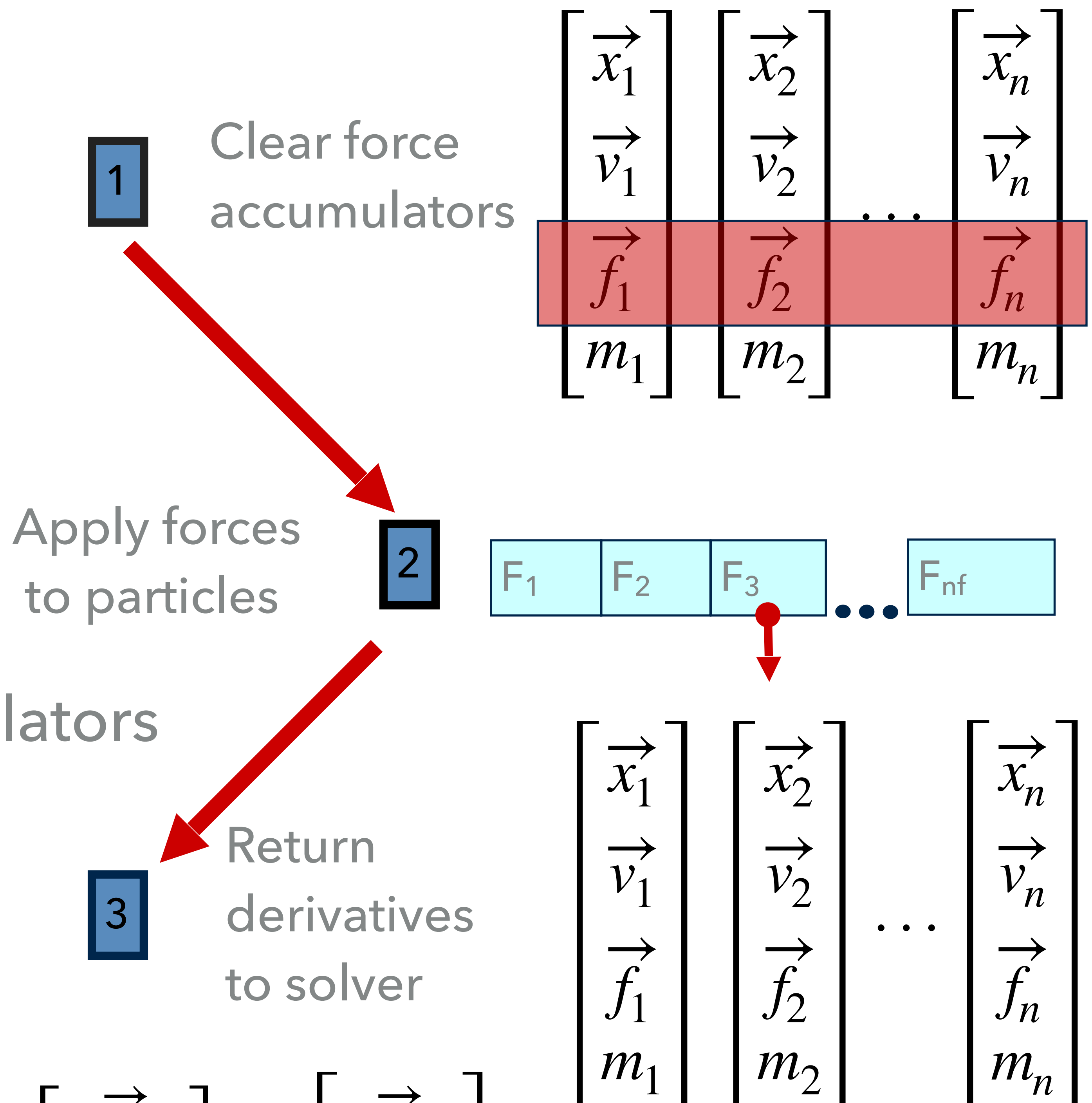
PARTICLE SYSTEMS WITH FORCES

- Force objects are black boxes that point to the particles they influence and add in their contributions.
- We can now visualize the particle system with force objects:



DERIVATIVE EVALUATION

1. Clear forces
   - Loop over particles
   - Zero force accumulators
2. Calculate forces
   - Sum all forces into accumulators
3. Return derivatives
   - Loop over particles,
   - Return **v** and **f**/*m*



$$\begin{bmatrix} \vec{v}_1 \\ \vec{f}_1/m_1 \end{bmatrix} \begin{bmatrix} \vec{v}_2 \\ \vec{f}_2/m_2 \end{bmatrix} \cdots \begin{bmatrix} \vec{v}_n \\ \vec{f}_n/m_n \end{bmatrix}$$

GRAVITY AND VISCOUS DRAG

- The force due to gravity is: $\overrightarrow{f}_{grav} = m\overrightarrow{G}$

```
p→f += p→m * F→G
```

- Often, we want to slow things down with viscous drag:

$$\overrightarrow{f}_{drag} = -k\overrightarrow{v}$$

```
p→f -= F→k * p→v
```

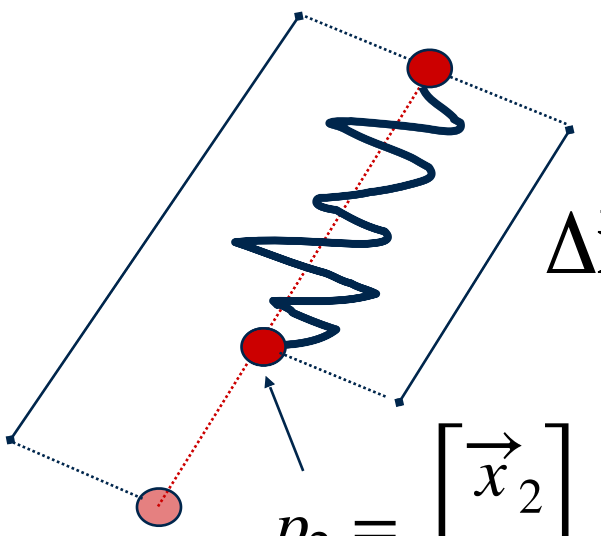
DAMPED SPRING

- Recall the equation for the force due to a spring:

$$f = -k_{spring}\left(\left|\Delta\vec{\mathbf{x}}\right| - r\right)$$

r = rest length

$$p_1 = \begin{bmatrix} \vec{x}_1 \\ \vec{v}_1 \end{bmatrix}$$



$$\Delta\vec{\mathbf{x}} = \vec{\mathbf{x}}_1 - \vec{\mathbf{x}}_2$$

$$p_2 = \begin{bmatrix} \vec{x}_2 \\ \vec{v}_2 \end{bmatrix}$$

DAMPED SPRING

- We can augment this with damping:

$$f = -\left[k_{spring}\left(\left|\Delta\vec{\mathbf{x}}\right| - r\right) + k_{damp}\left|\vec{\mathbf{v}}\right|\right]$$

- The resulting force equations for a spring between two particles become:

$$\vec{\mathbf{f}}_1 = -\left[k_{spring}\left(\left|\Delta\vec{\mathbf{x}}\right| - r\right) + k_{damp}\left(\frac{\Delta\vec{\mathbf{v}} \bullet \Delta\vec{\mathbf{x}}}{\left|\Delta\vec{\mathbf{x}}\right|}\right)\right]\frac{\Delta\vec{\mathbf{x}}}{\left|\Delta\vec{\mathbf{x}}\right|}$$

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

ADDITIONAL RESOURCES

- Verlet Integration:
  - [http://www.saylor.org/site/wp-content/uploads/2011/06/MA221-6.1.pdf]
  - [http://gamedevelopment.tutsplus.com/tutorials/simulate-tearable-cloth-and-ragdolls-with-simple-verlet-integration--gamedev-519]
  - [www.gamasutra.com/view/feature/132771/the_secrets_of_cloth_simulation_in_.php]