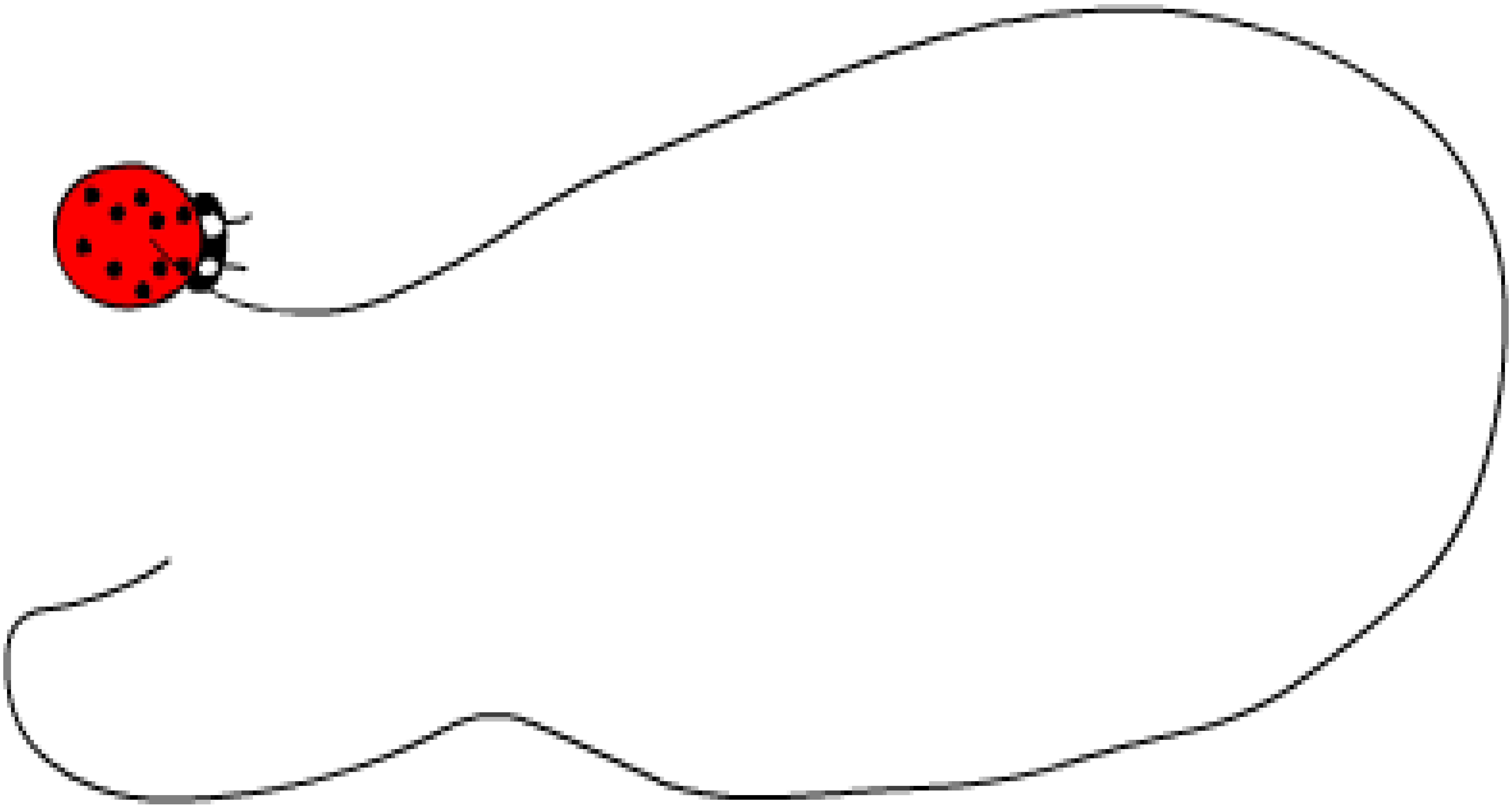
Animation Curves and Splines 1



Animation Homework

- Set up a simple avatar
  - E.g. cube/sphere (or square/circle if 2D)
- Specify some key frames (positions/orientations)
- Associate a time with each key frame
- Create <u>smooth animations</u> for the avatar movement
  - does it pass through or just near the key frames and positions?
- Save the canned animations for later use
- Provide a couple of examples
  - A crawling bug, a bouncing ball, etc…
- Use unity…

Smooth Animation

- Suppose the user pushes down on a button, and holds it down, intending for their avatar to carry out some motion. E.g. move forward, backward, turn, etc.
- Should the avatar be allowed to turn on a dime? Should it be allowed to instantly accelerate to full speed? Should it be allowed to instantly go from full speed to full stop?
- Jerky motion can be rather annoying to the user, and even cause headaches (V.R. is making things even worse)
- Having the right physical feel to the motion helps both immersive-ness and believability (suspension of disbelief)
- E.g. it's more realistic for a car in a racing game to take a wider turn if going too fast, or even spin-out if the user tries to take too tight of a turn at too high of a speed

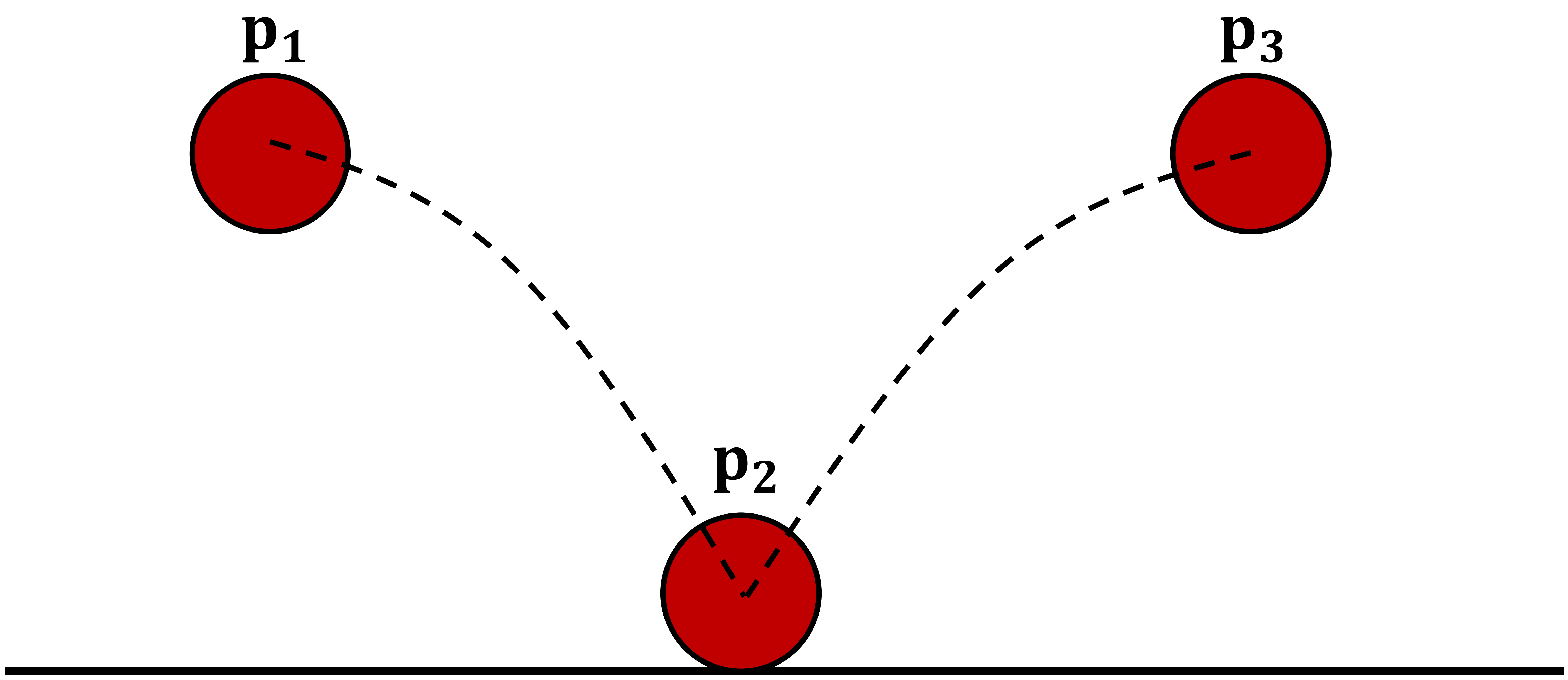
Smooth Animation

- On the other hand, “hard core” users may prefer complete control over their avatars
  - smoothing of animations can be perceived as input delays
- However, even if the user is given tight instantaneous control over their avatar, it is typically desirable to smooth out the camera motion in order to avoid jerkiness of the entire screen
  - especially if the camera is attached in some way to the unrealistically animated avatar
- It is also desirable to smooth out the motion of all the non-player avatars the player interacts with so that they have a more pleasing appearance
- One will also want smooth walk, run, and other motion cycles for the avatar as it carries out its actions

Keyframing

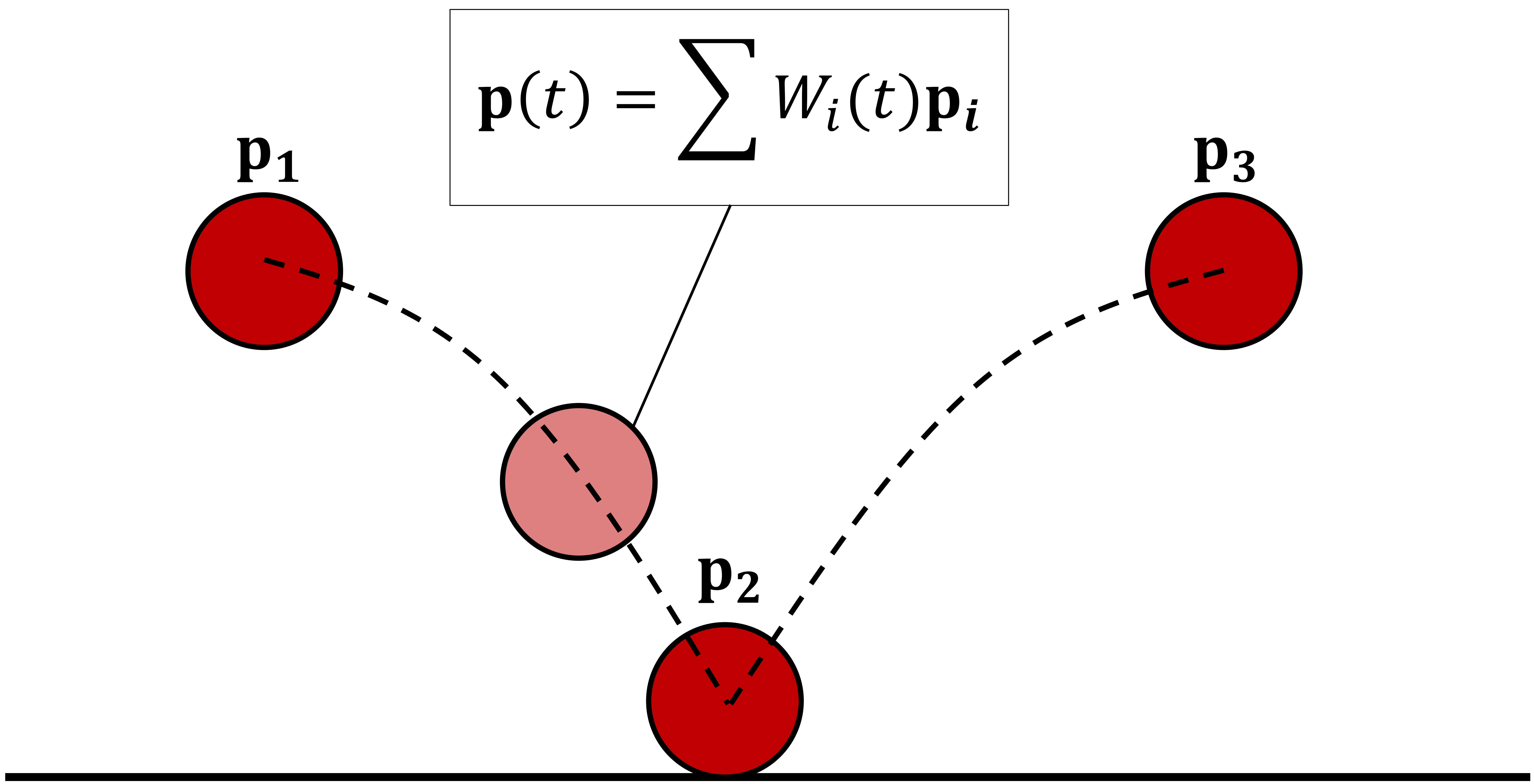
Keyframing

- Keyframing is a method of animating an object by defining starting and ending points of a smooth transition
- These starting and ending points of transitions are key frames
- A sequence of key frames defines what movement the viewer will see



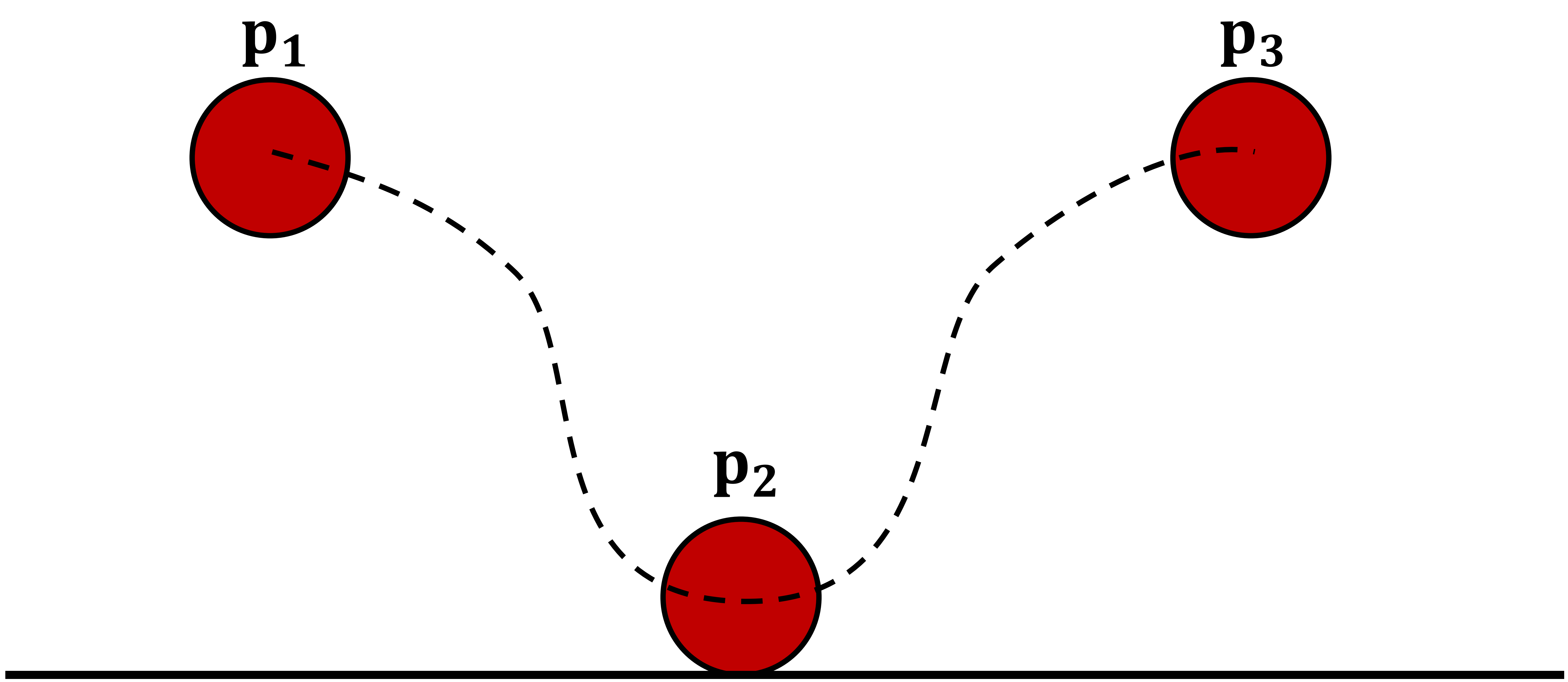
Path Interpolation

- We need enough frames between key frames to give the viewer the illusion of continuous movement
- The frames between key frames are interpolated from key frames



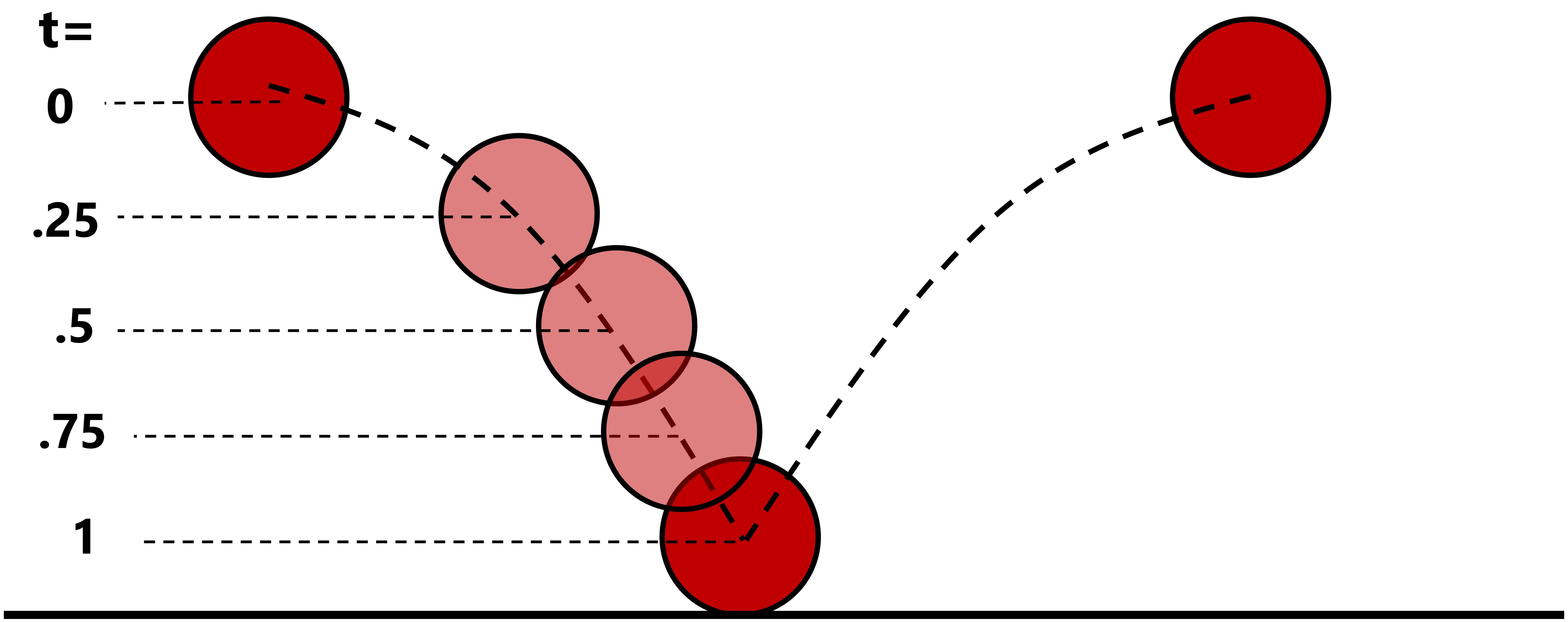
Path Interpolation

- Interpolation is not foolproof. There may be overshoots, undershoots or other side effects. The animator should be careful choosing interpolation methods
- The figure shows a reasonable way of using interpolation to obtain a smooth path connecting key frames
- But this would be an unrealistic path for a bouncing ball



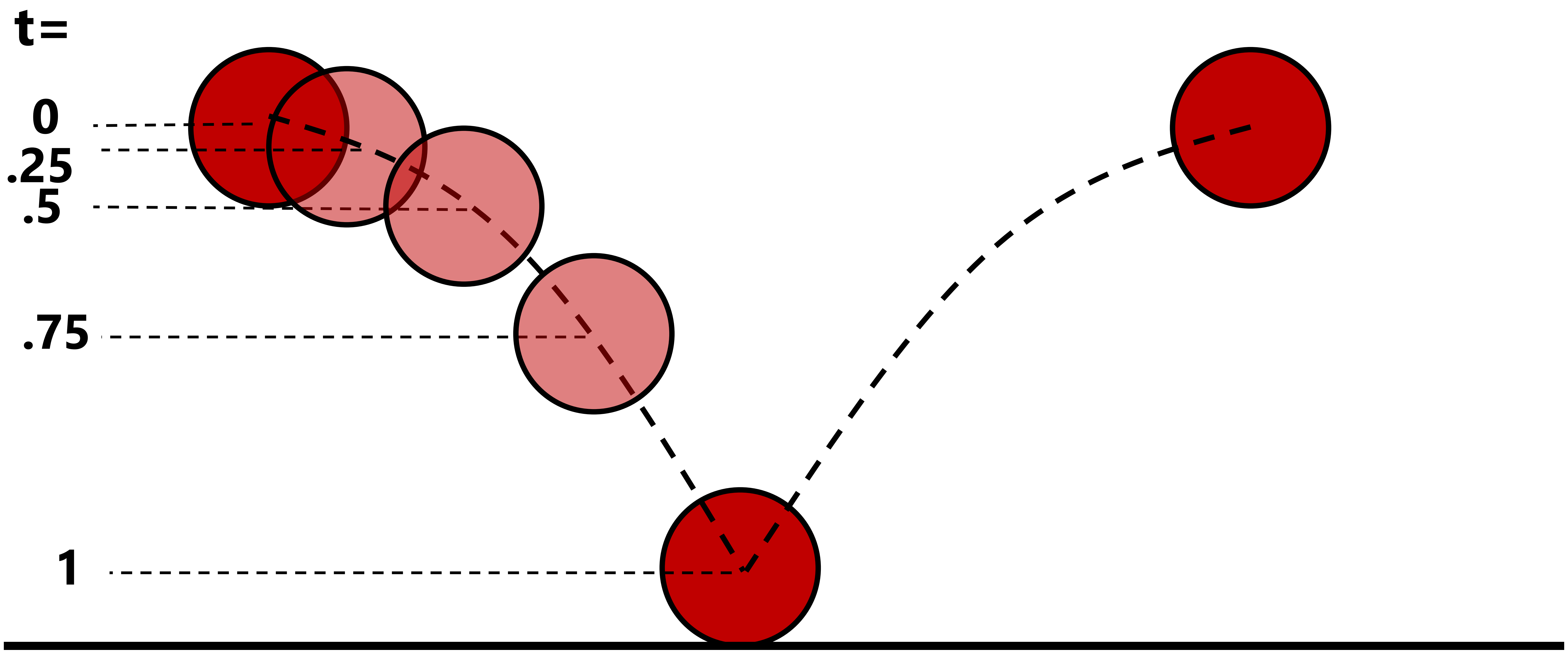
Temporal Parameterization

- Changing the way of parameterizing the curve as a function of time changes the object's motion between key frames
- The figure below shows a plausible path for a bouncing ball, but does not shows a plausible motion



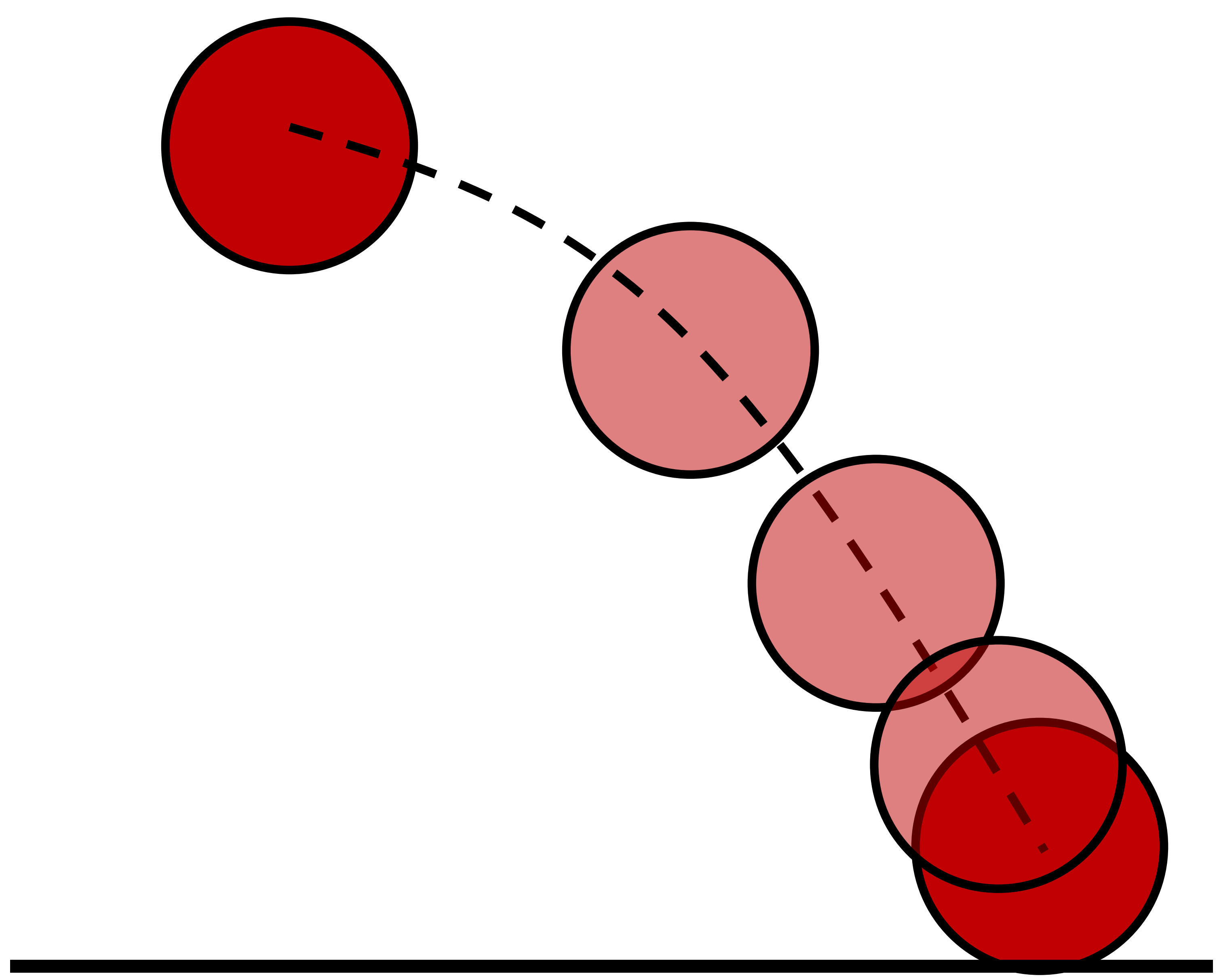
Temporal Parameterization

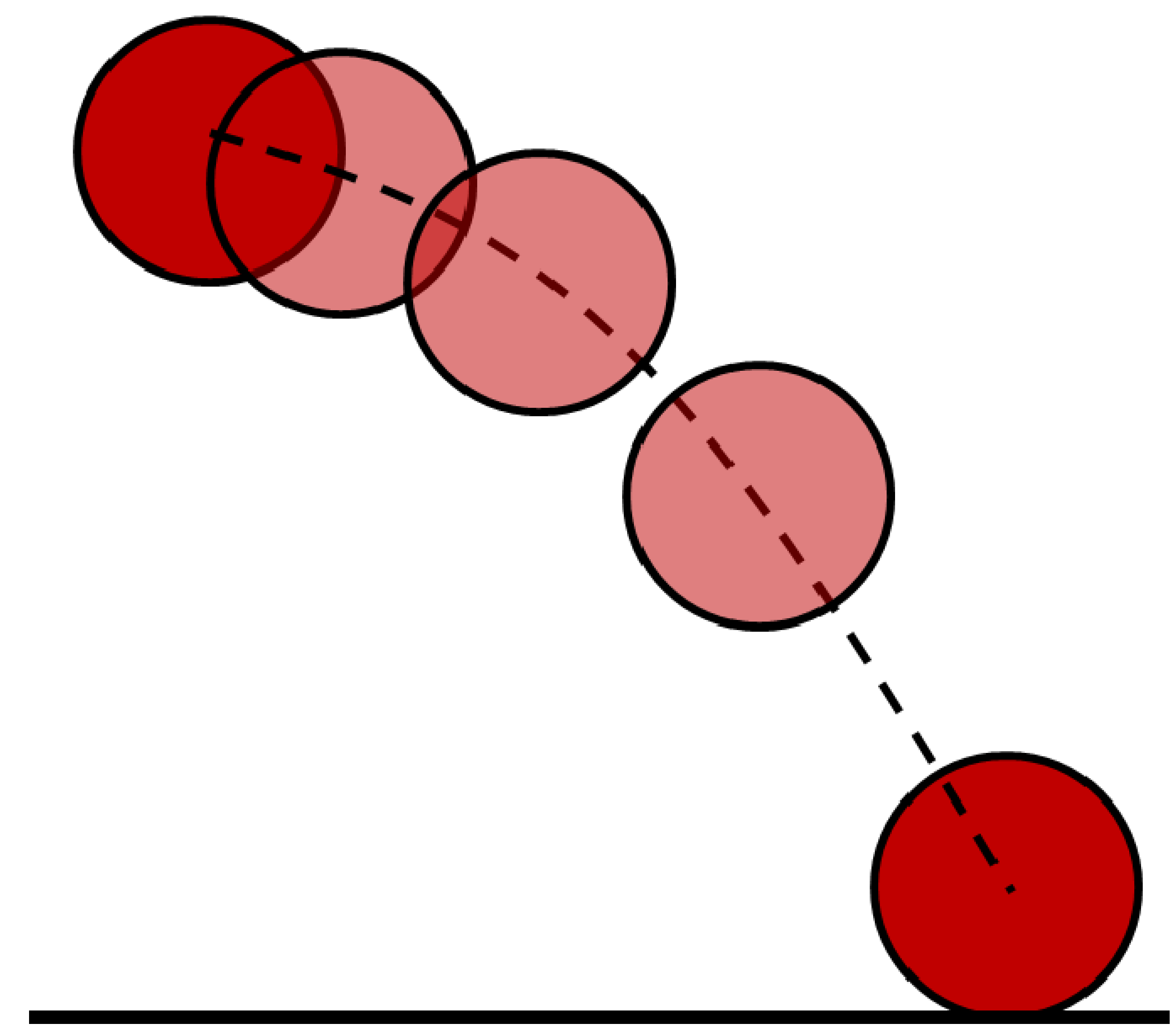
- Changing the parameterization of the curve as a function of time makes the motion more realistic
- The ball accelerates as it falls



Both Space and Time...

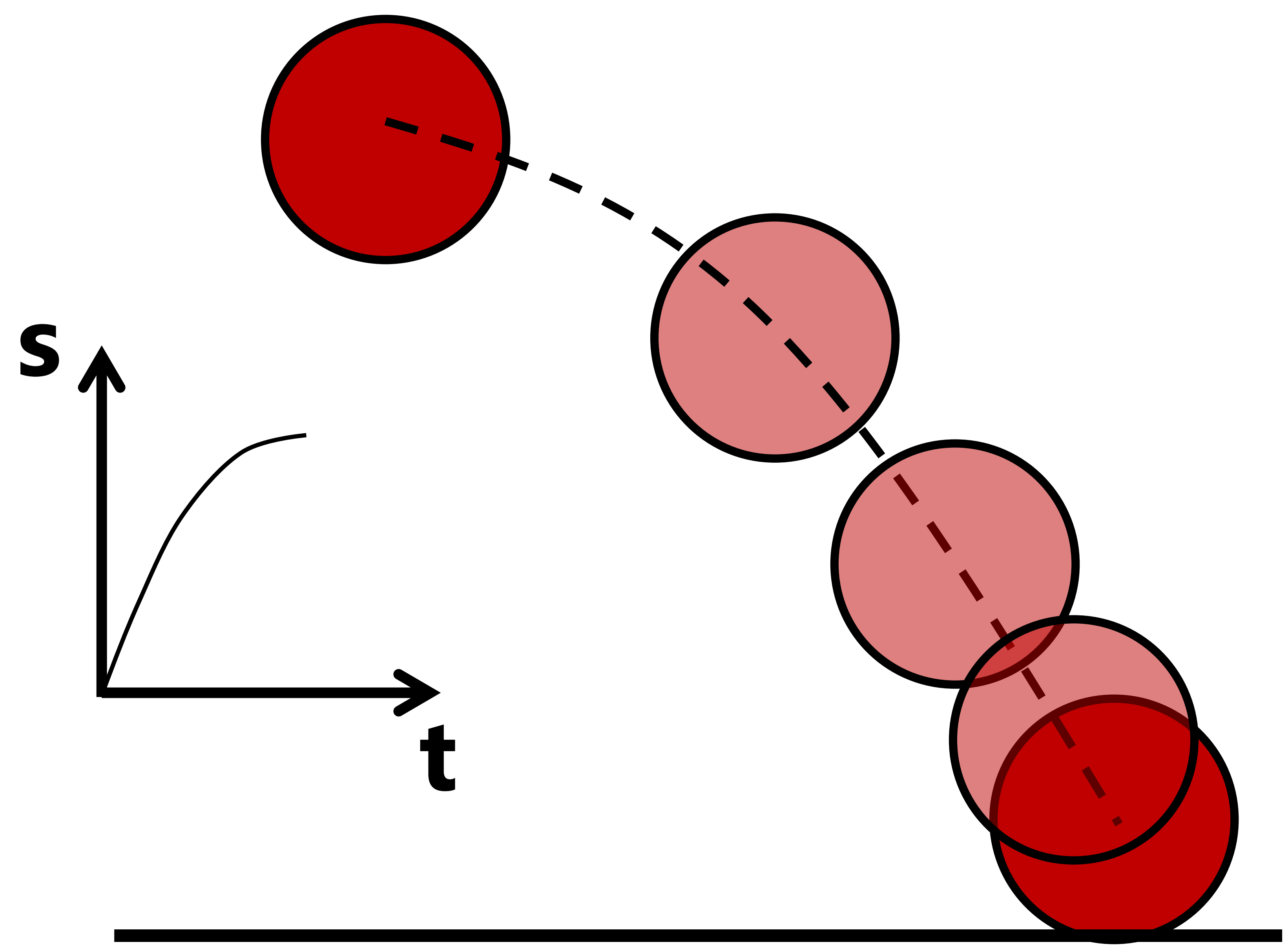
- Spatial curves determine the path for the motion
- Temporal parameterization along that path determines how fast the object moves

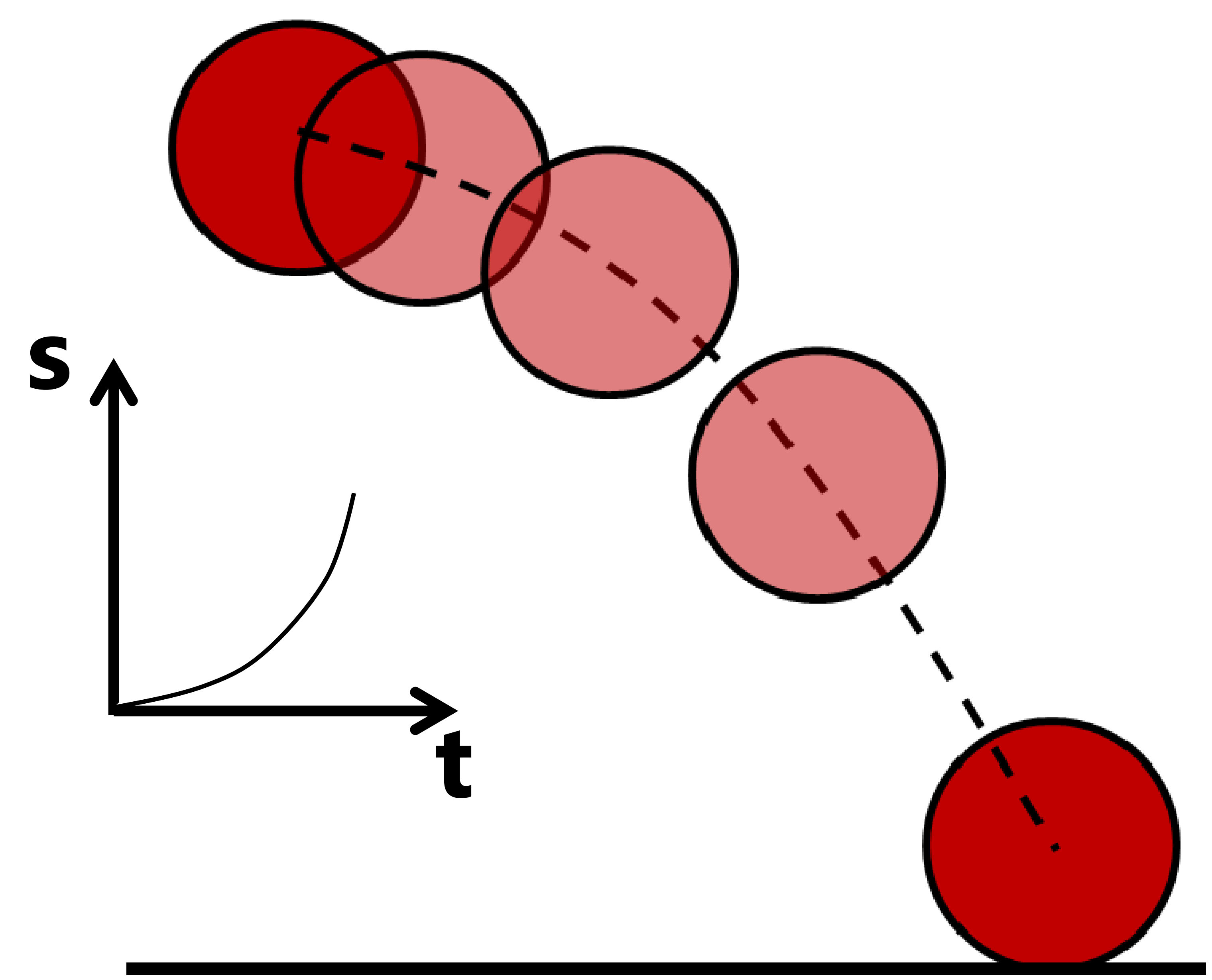




Temporal Parameterization

- First, parameterize the 3D curved path with respect to arc length s to obtain a function **p**(s)
- Then create a graph relating arc length s versus time t to specify how fast the object moves along the curved path through space
- Plug s(t) into **p**(s) to get **p**(t)
- Adjusting the temporal curve s(t) controls how fast the object moves along the path **p**(s) without changing the path

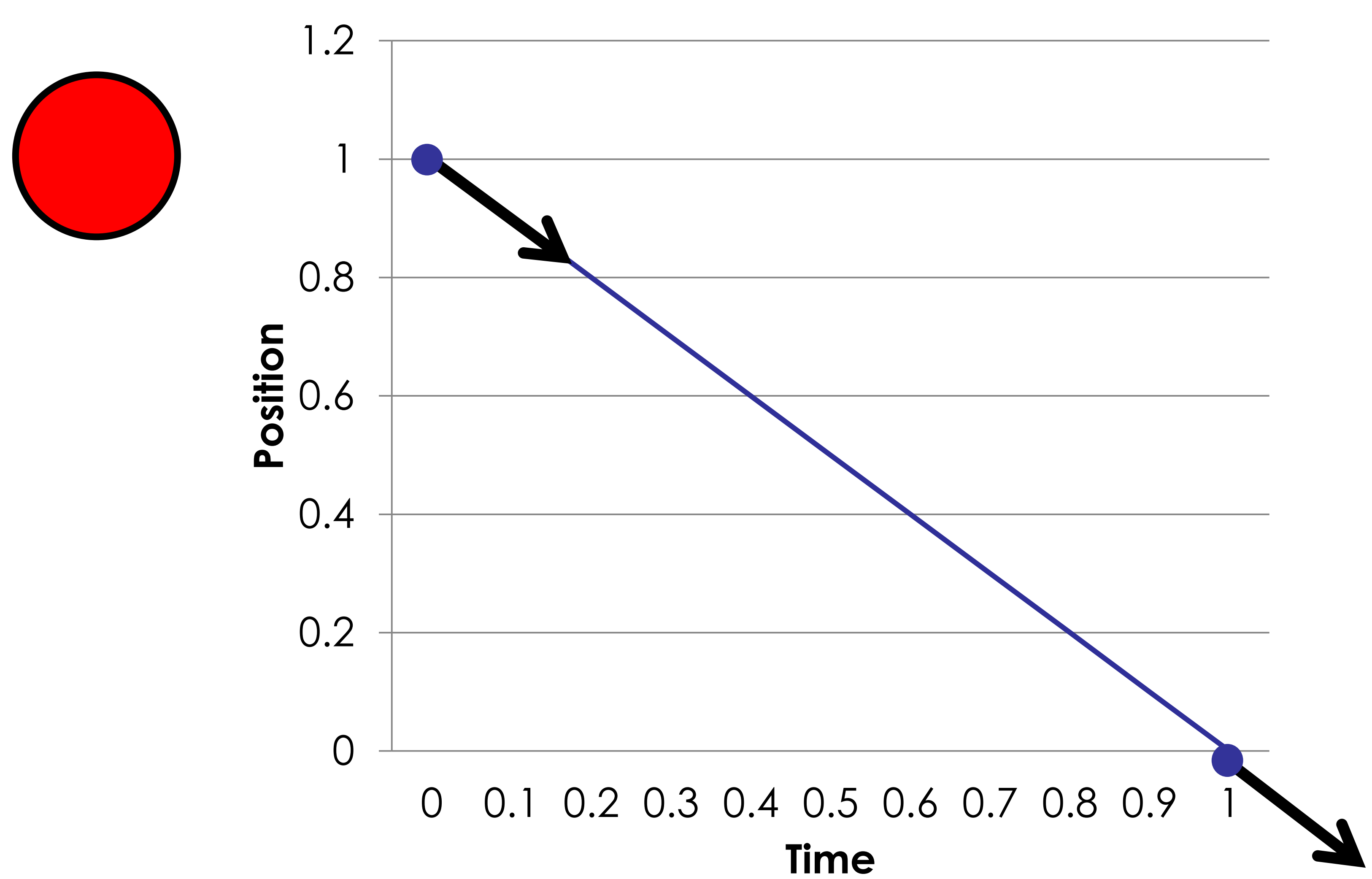


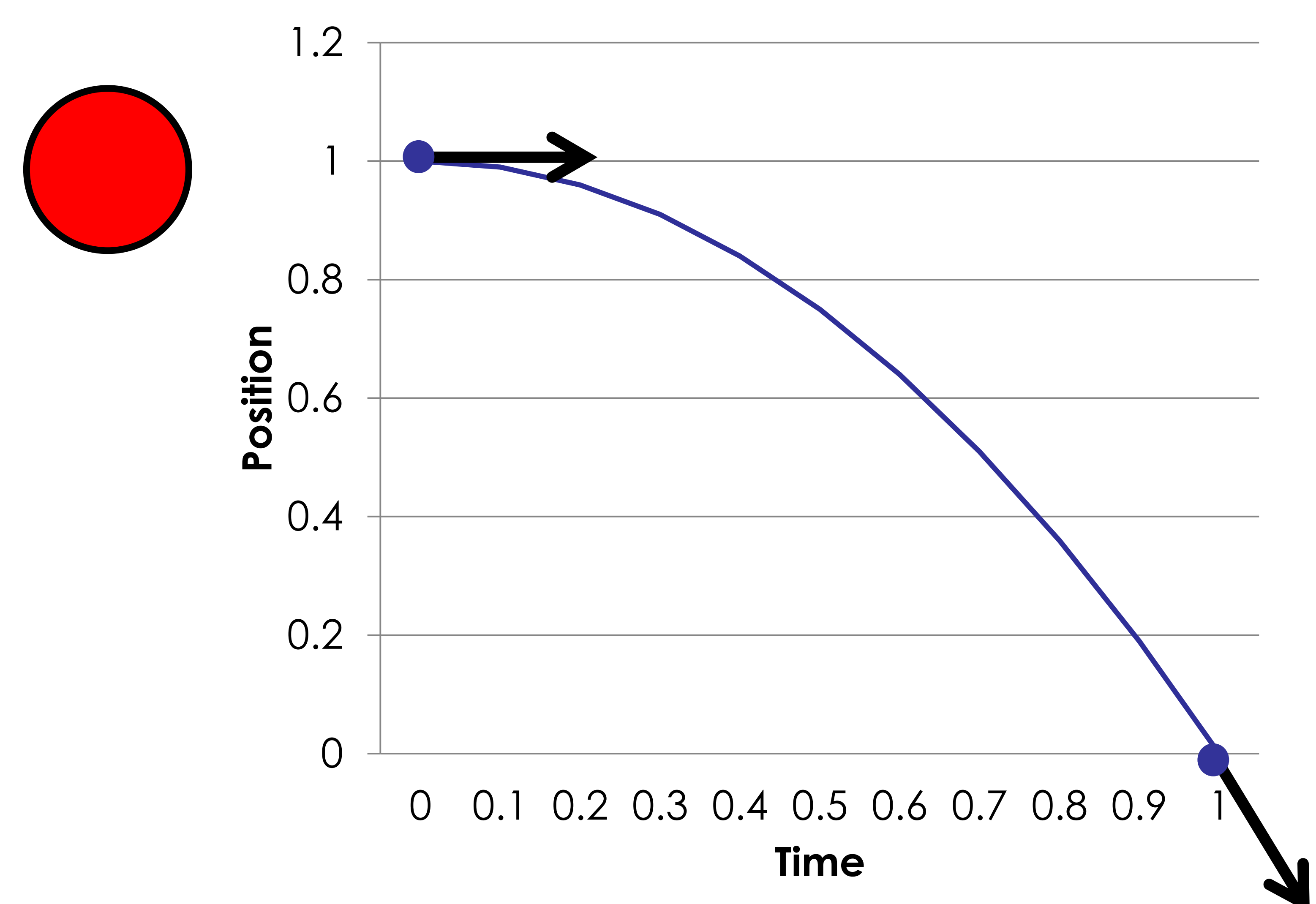


Animation Curves

Animation Curves

- Artists use animation curves to parameterize spatial positions as a function of time
- For a 3D motion, each of x, y, z coordinates can have its own animation curve
- Artist workflow:
  - Manipulate key points (blue dots)
  - Set tangent directions and lengths (arrows) at key points

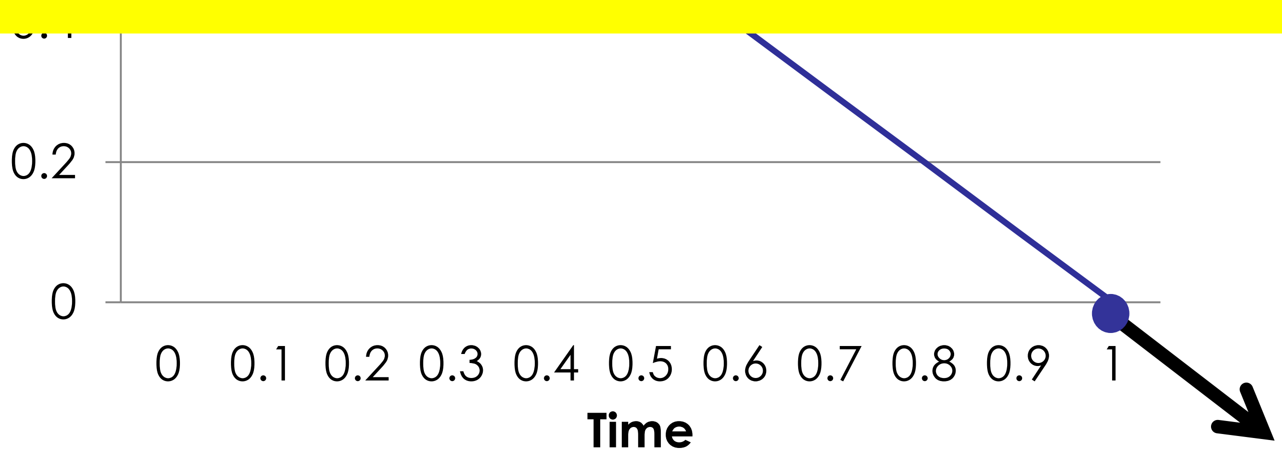


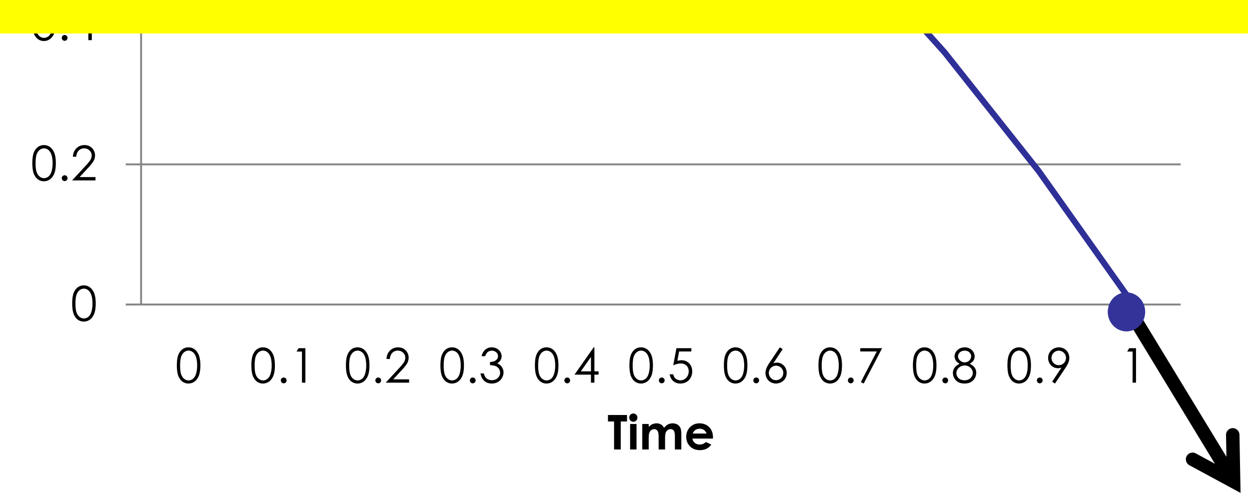


Animation Curves

- Art
  pos
- For s
  ow
- Art
  - M
  - S nts

Wait a minute, arrows?





Let's go watch an artist first…

Question #1

LONG FORM:

- Summarize how one would go about animating a bouncing block using an animation system. What are the important considerations?
- List 5 ideas you have for potential aspects of your game (One sentence for each idea).

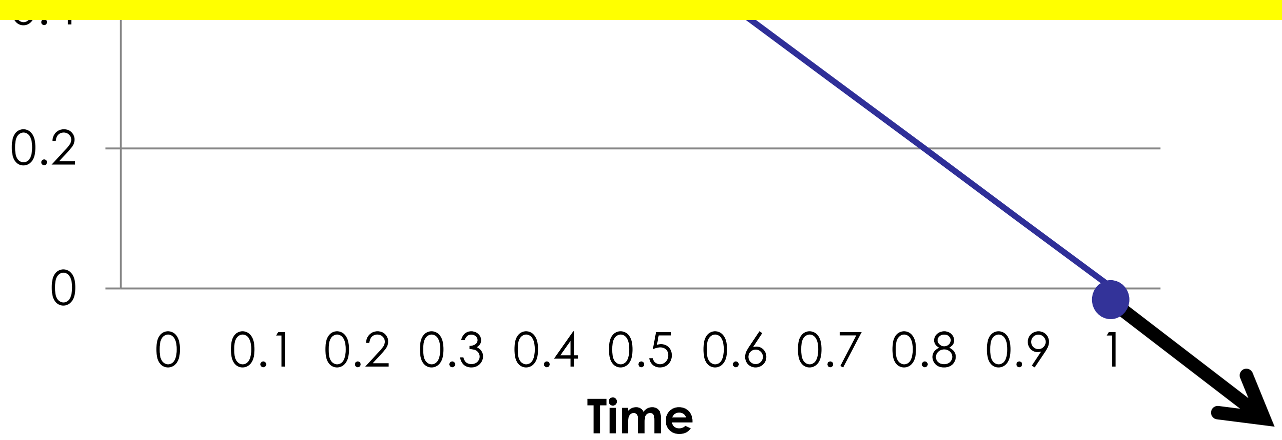
SHORT FORM

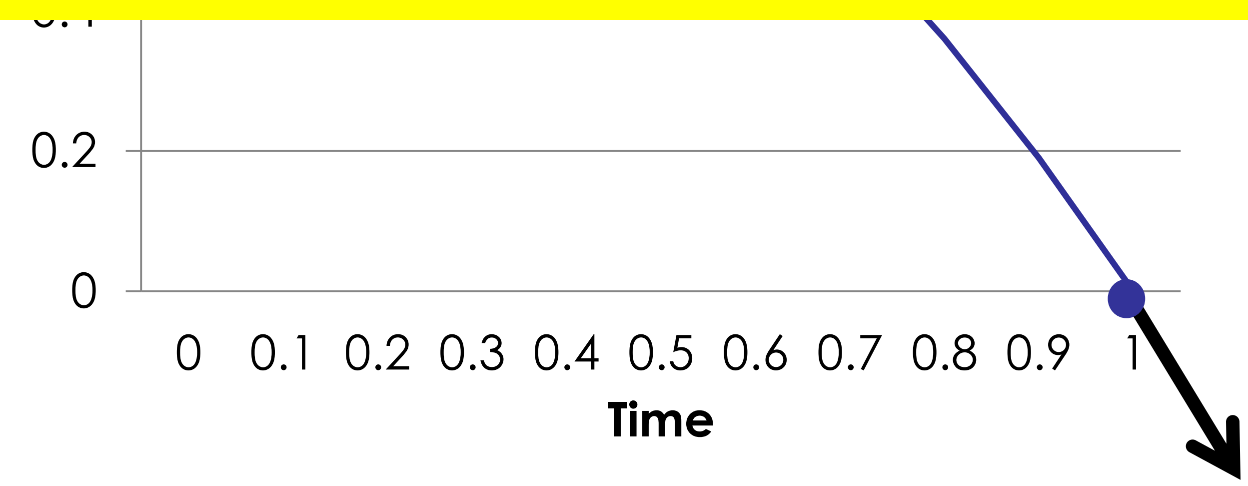
- Form clusters (of about size 5-ish?) with at least one experienced gamer in each cluster.
- Get advice on what sorts of games might be good/feasible/etc. from the <u>experienced</u> gamer ---> name/email too ;)
- Write down the best piece of advice you heard.

Animation Curves

- Art
pos
- For s
ow
- Art
  - M
  - S nts

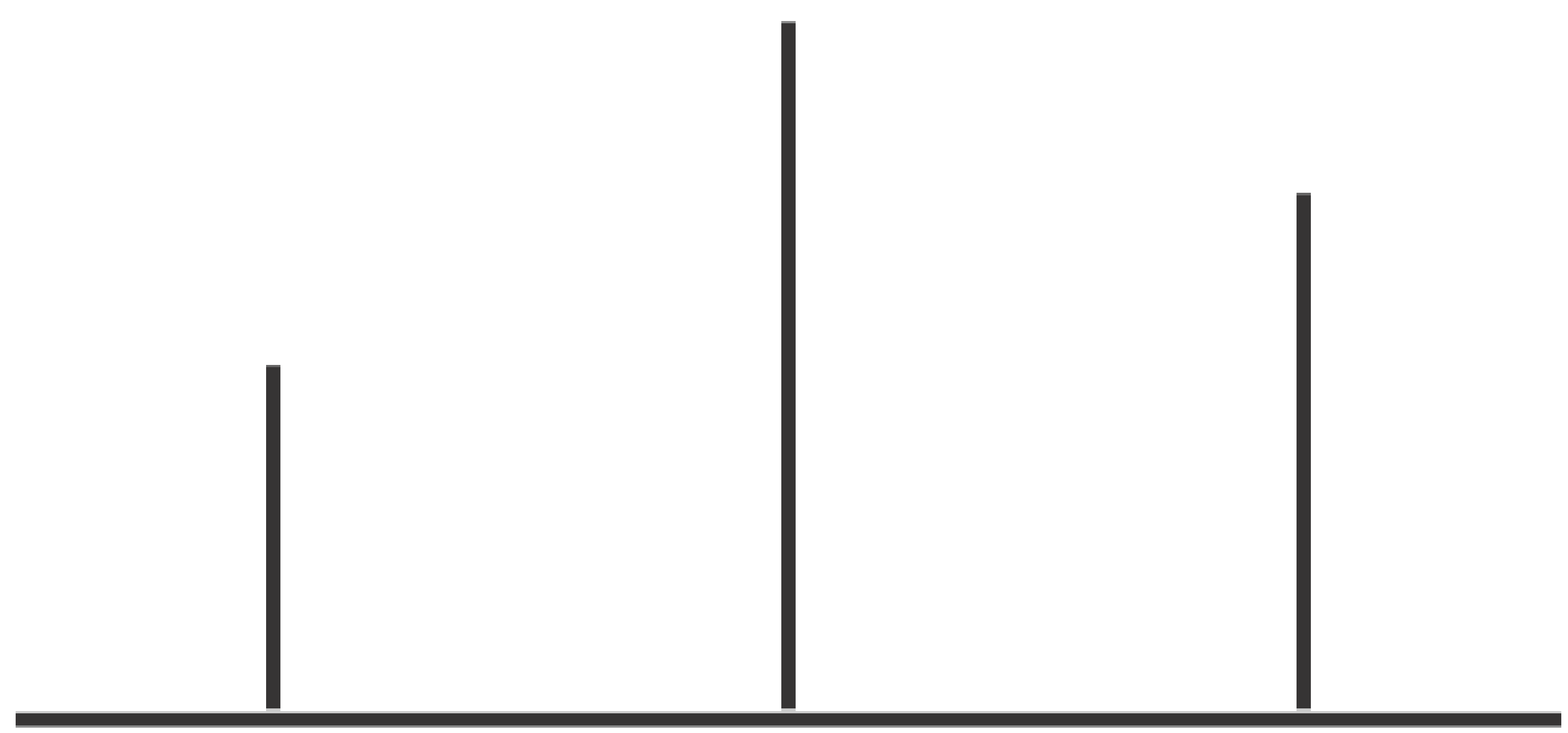
Wait a minute, arrows?



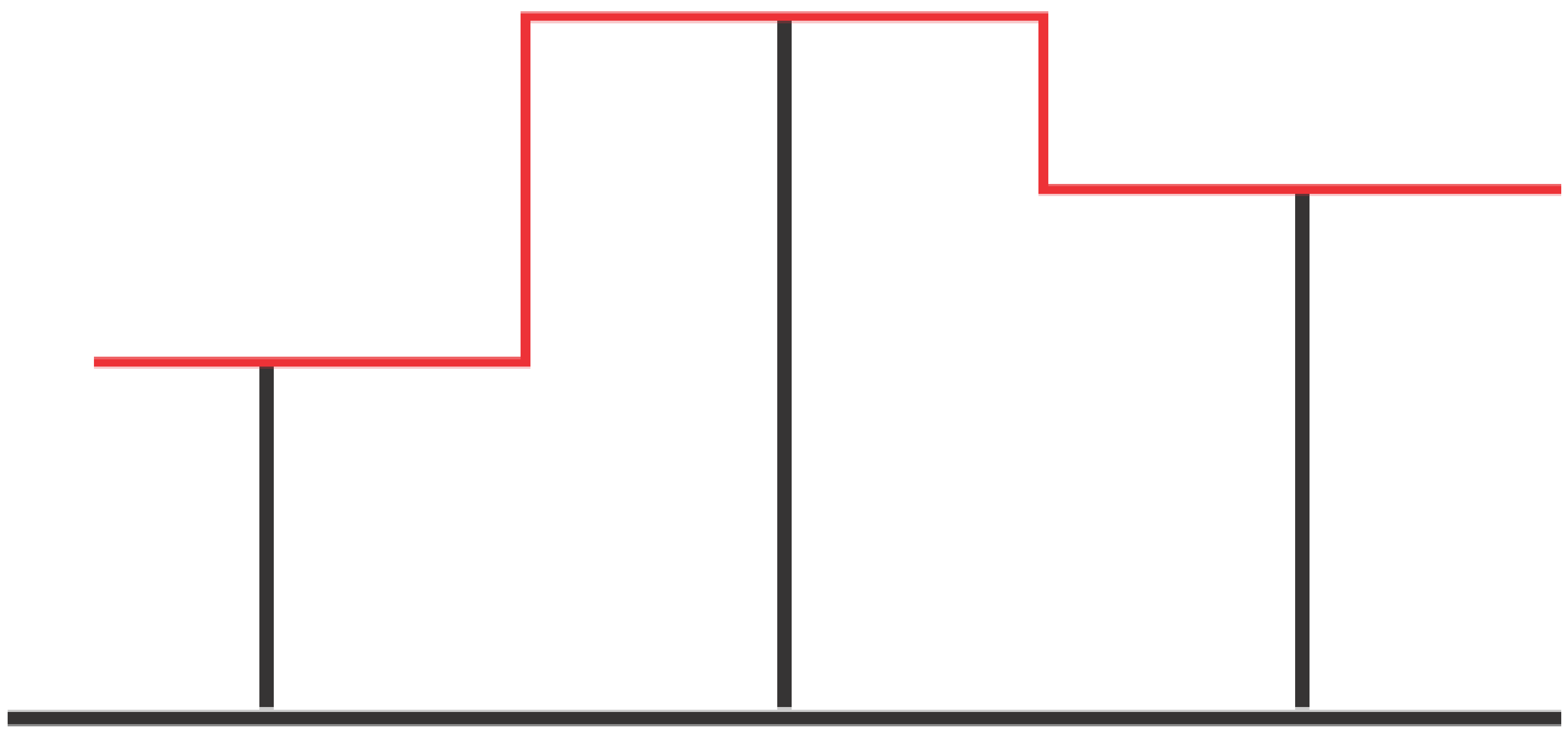


Curves & Splines

Goal: Interpolate Values

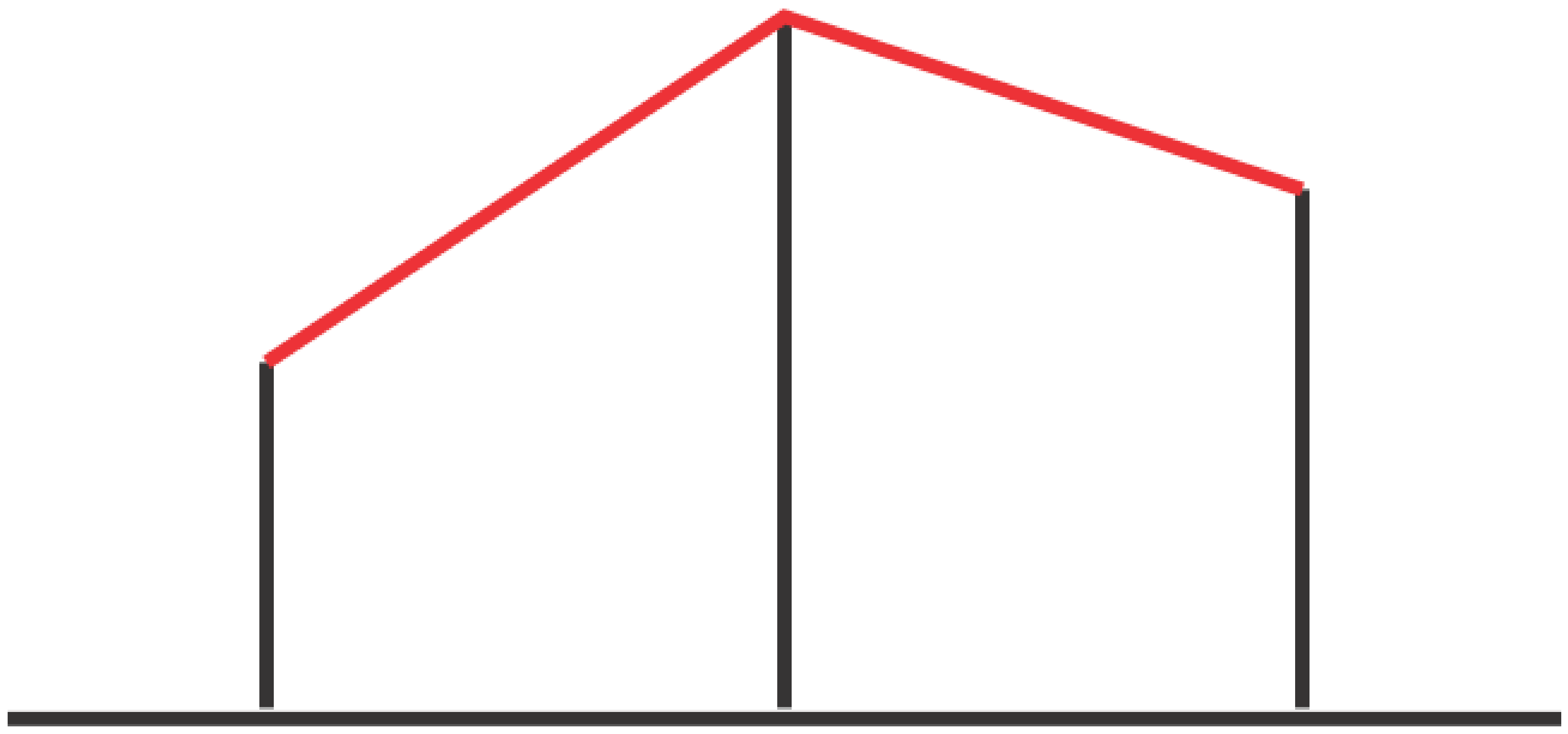


Nearest Neighbor Interpolation



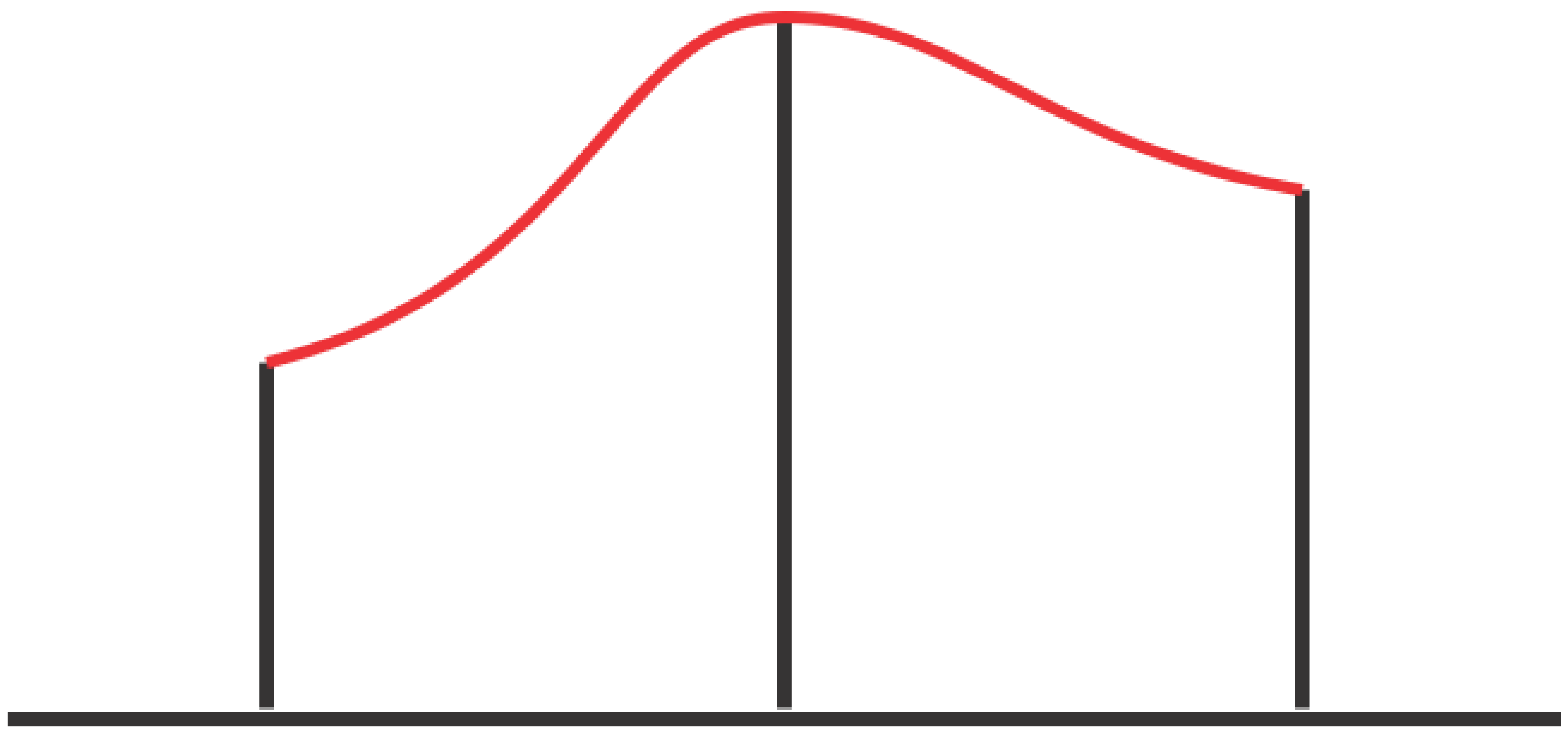
Problem: not continuous

Linear Interpolation

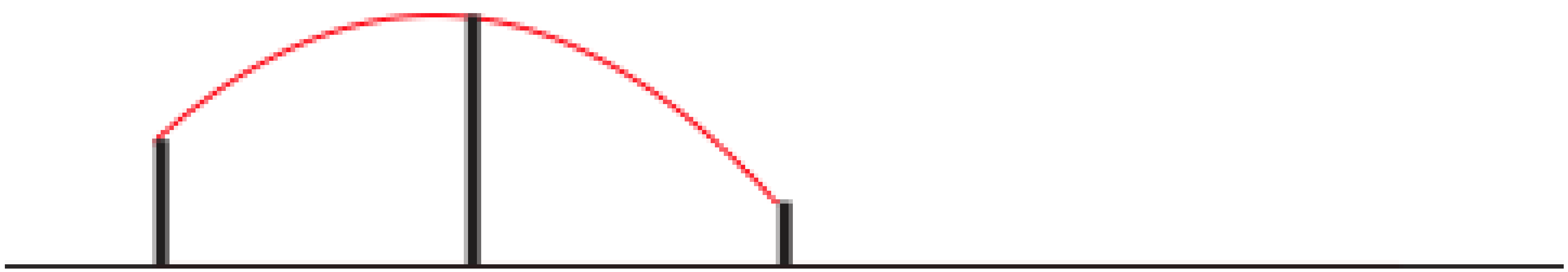


Problem: derivative not continuous

Smooth Interpolation?

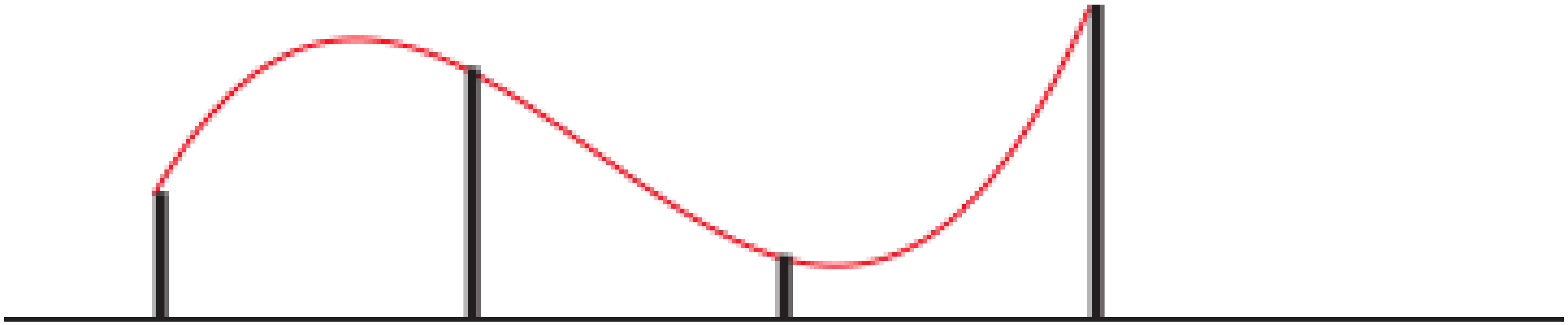


Polynomial Interpolation



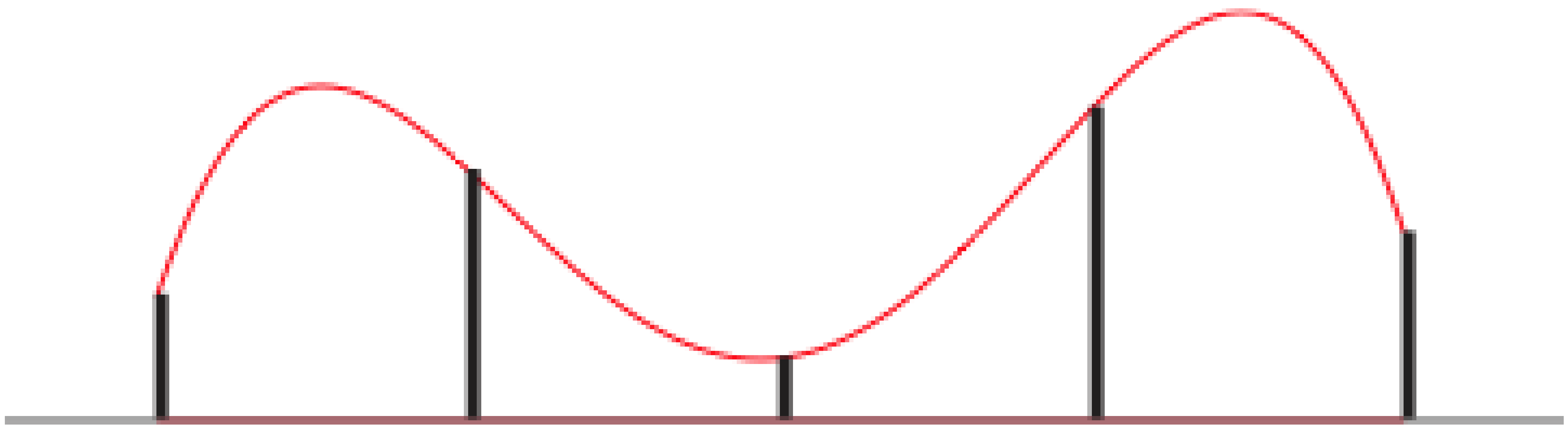
# of constraints = 3
polynomial degree = 2

Polynomial Interpolation



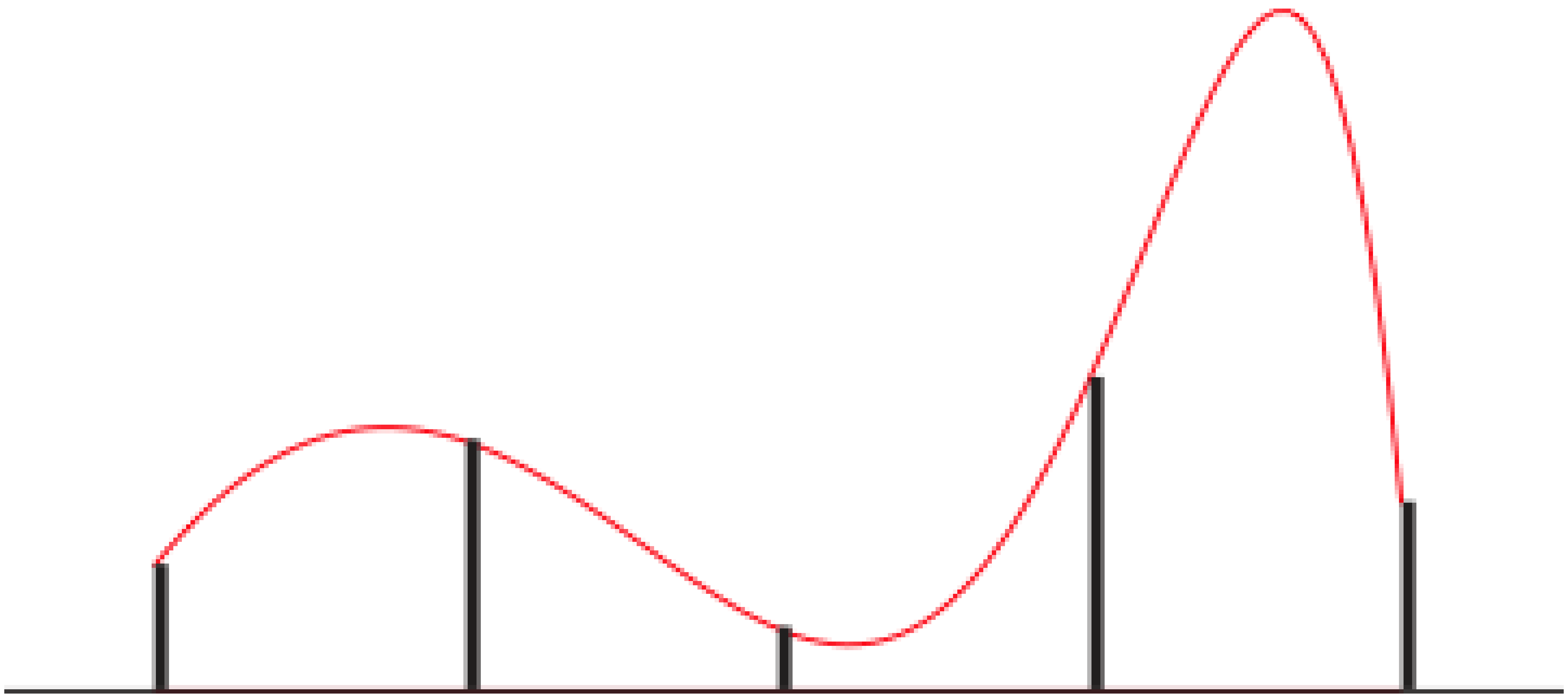
# of constraints = 4
polynomial degree = 3

Polynomial Interpolation



# of constraints = 5
polynomial degree = 4

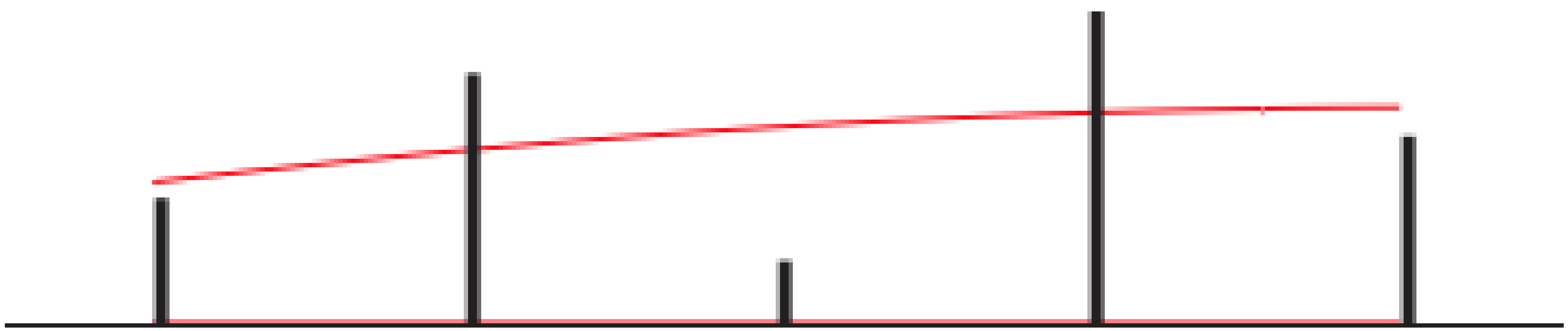
Higher Order Polynomials



# of constraints = 5
polynomial degree = 4

- **Curve may oscillate unexpectedly**

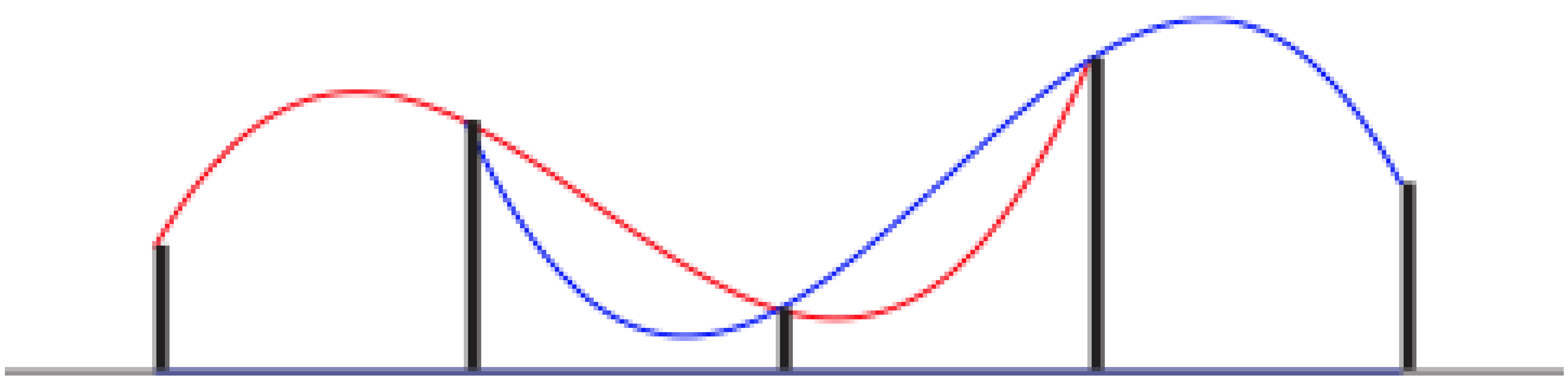
Overconstrained / Least-Squares



# of constraints = 5
polynomial degree = 2

- Curve does not interpolate points
- Instead it approximates the points

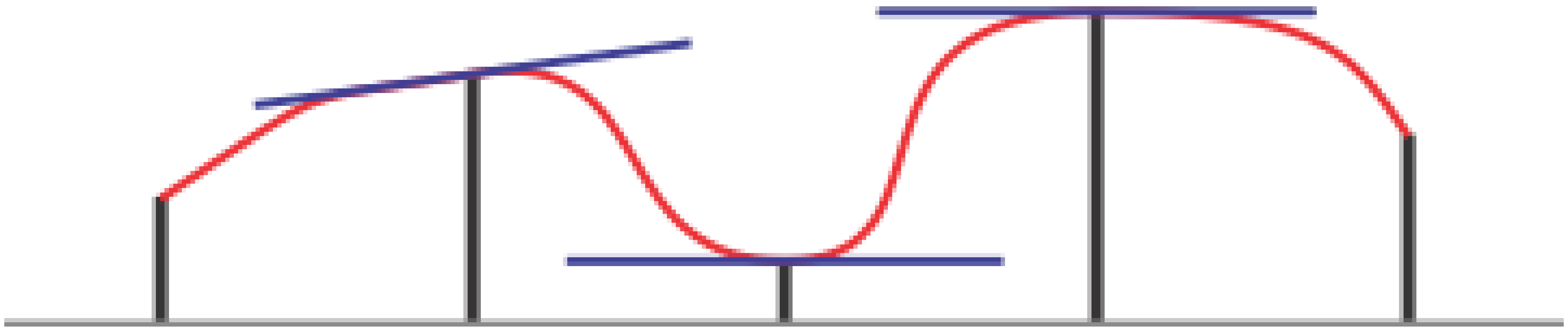
Multiple Lower Order Polynomials



two 4-point interpolations
two degree 3 polynomials

- **Curves don’t agree in region of overlap**

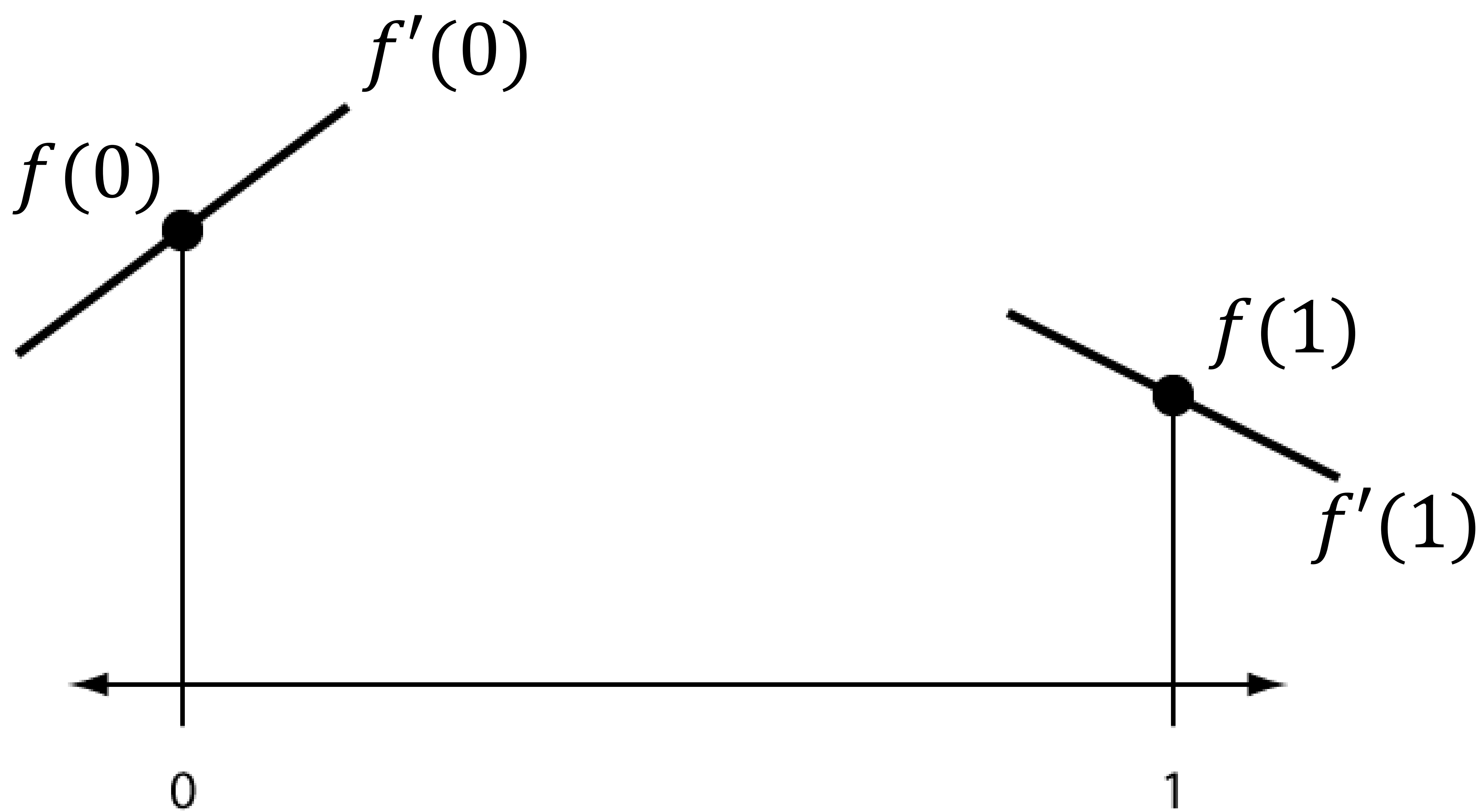
Piecewise Polynomial Interpolation



Different curves in each interval

Match values and slopes at endpoints

Cubic Hermite Interpolation



Given: values and derivatives at 2 points

- 4 constraints → need 4 degrees of freedom
- use a degree 3 cubic polynomial

Cubic Hermite Interpolation

- Cubic polynomial

$$f(t) = at^3 + bt^2 + ct + d$$
$$f'(t) = 3at^2 + 2bt + c$$

- Solve for coefficients:

$$f(0) = h_0 = d$$
$$f(1) = h_1 = a + b + c + d$$
$$f'(0) = h_2 = c$$
$$f'(1) = h_3 = 3a + 2b + c$$

Matrix Representation

$$h_0 = d$$
$$h_1 = a + b + c + d$$
$$h_2 = c$$
$$h_3 = 3a + 2b + c$$

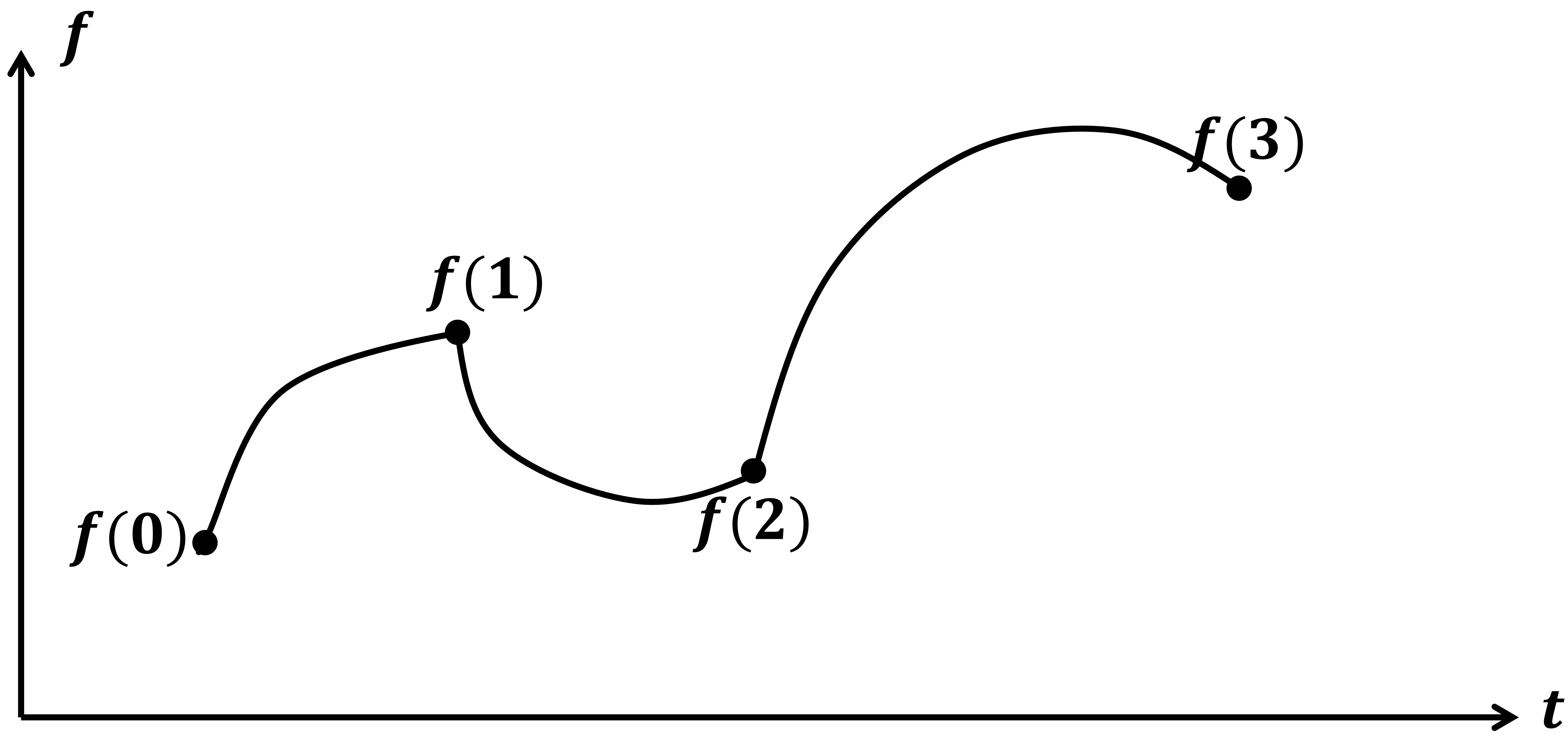
$$\begin{bmatrix} h_0 \\ h_1 \\ h_2 \\ h_3 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$$

Solve for a, b, c, d

$$\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} h_0 \\ h_1 \\ h_2 \\ h_3 \end{bmatrix}$$

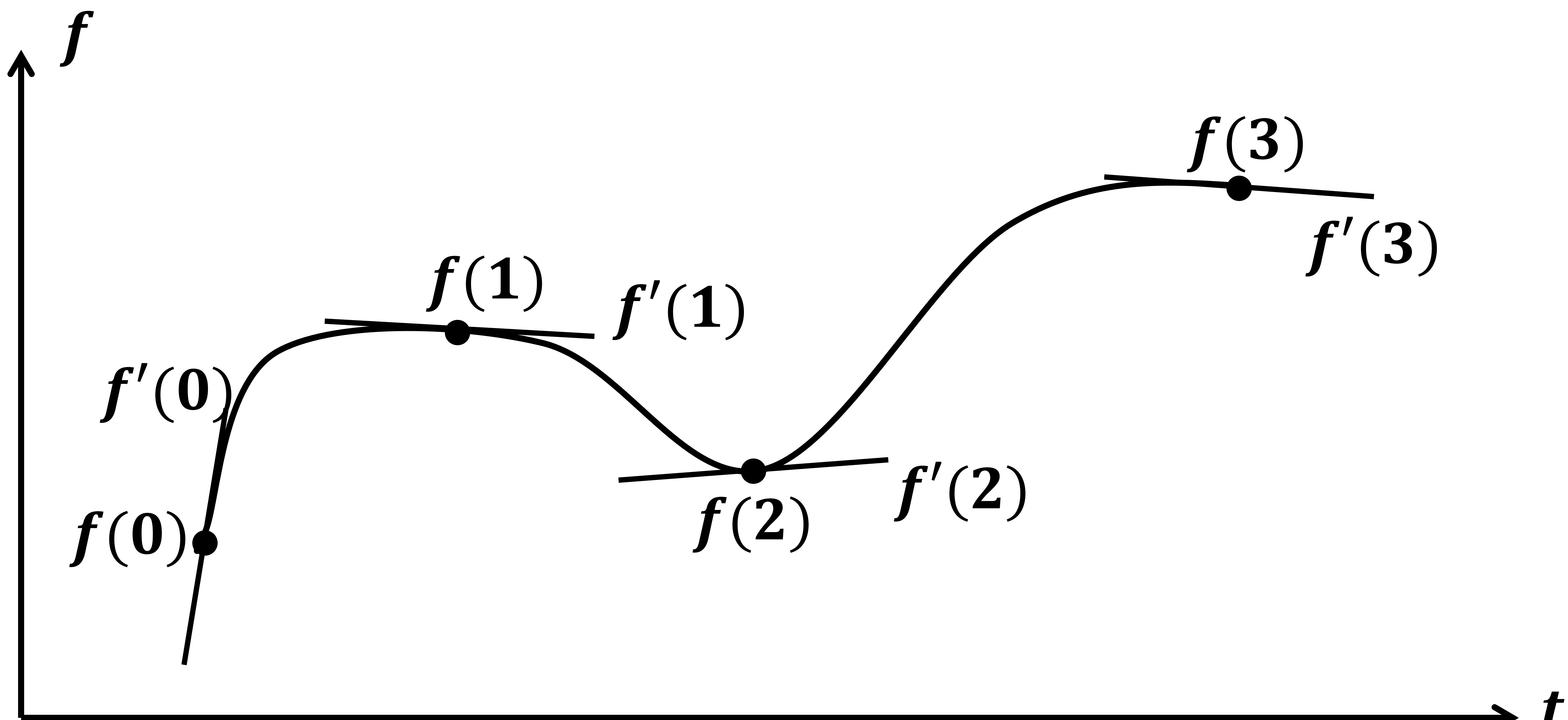
Inverse Matrix

C^0 Continuity for Hermite Splines



- **Points are specified continuously**
- **But derivatives are specified discontinuously**

C^1 Continuity for Hermite Splines



- **Derivatives are specified continuously as well**

Basis Functions

Hermite Basis Functions?

$$f(t) = \sum_{i=0}^{3} h_i H_i(t)$$

$$\begin{bmatrix} a & b & c & d \end{bmatrix} \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix} = \begin{bmatrix} h_0 & h_1 & h_2 & h_3 \end{bmatrix} \begin{bmatrix} H_0(t) \\ H_1(t) \\ H_2(t) \\ H_3(t) \end{bmatrix}$$

monomial basis **Hermite basis**

Insert Identity Matrix

identity

$$\underbrace{\begin{bmatrix} a & b & c & d \end{bmatrix}\begin{bmatrix} 0 & 1 & 0 & 3 \\ 0 & 1 & 0 & 2 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix}}_{\begin{bmatrix} h_0 & h_1 & h_2 & h_3 \end{bmatrix}}\underbrace{\begin{bmatrix} 2 & -3 & 0 & 1 \\ -2 & 3 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 1 & -1 & 0 & 0 \end{bmatrix}\begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix}}_{\begin{bmatrix} H_0(t) \\ H_1(t) \\ H_2(t) \\ H_3(t) \end{bmatrix}}$$

Hermite Basis

Hermite Basis Functions

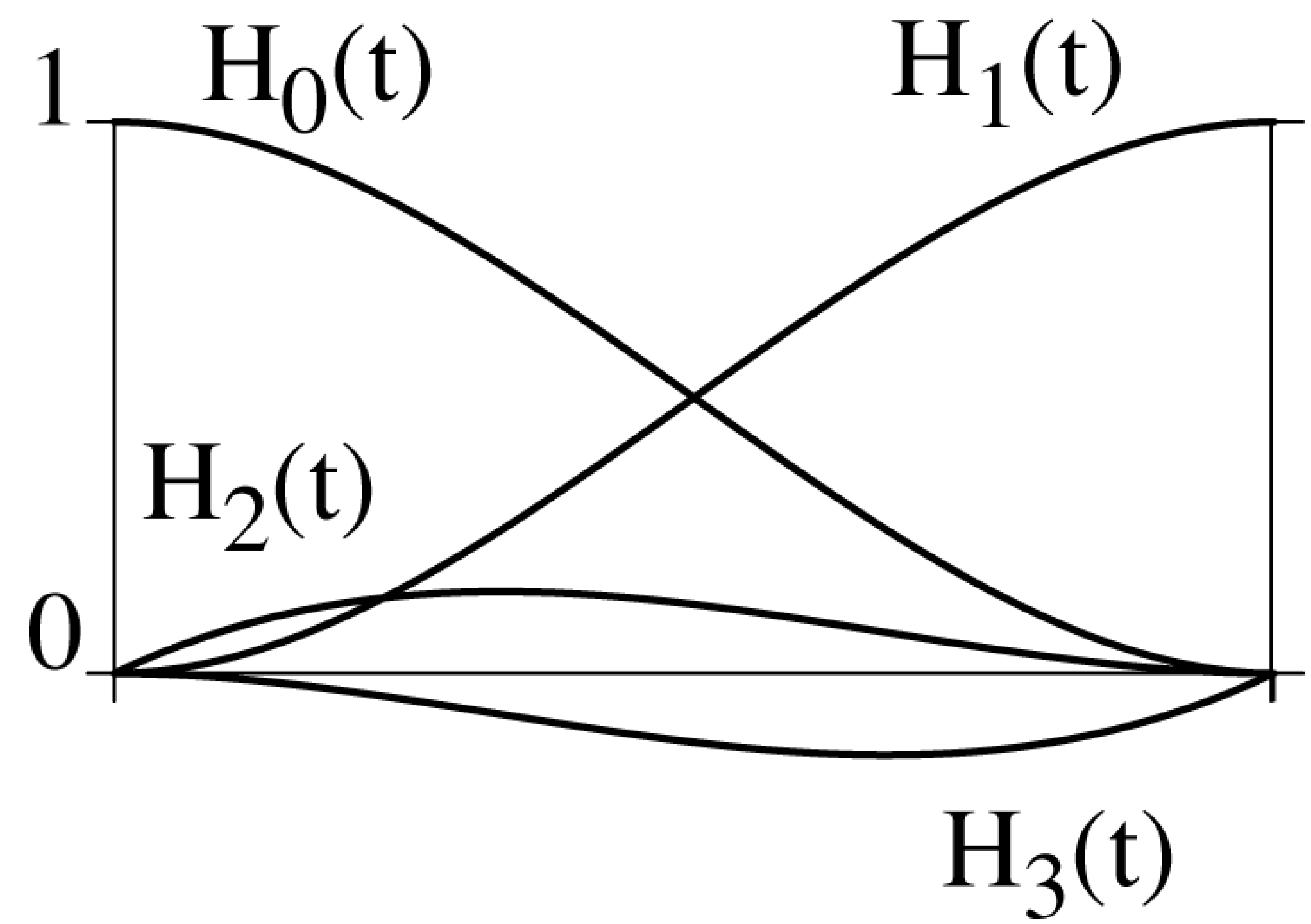
$$\begin{bmatrix} H_0(t) \\ H_1(t) \\ H_2(t) \\ H_3(t) \end{bmatrix} = \begin{bmatrix} 2 & -3 & 0 & 1 \\ -2 & 3 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 1 & -1 & 0 & 0 \end{bmatrix} \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix}$$

$$H_0(t) = 2t^3 - 3t^2 + 1$$

$$H_1(t) = -2t^3 + 3t^2$$

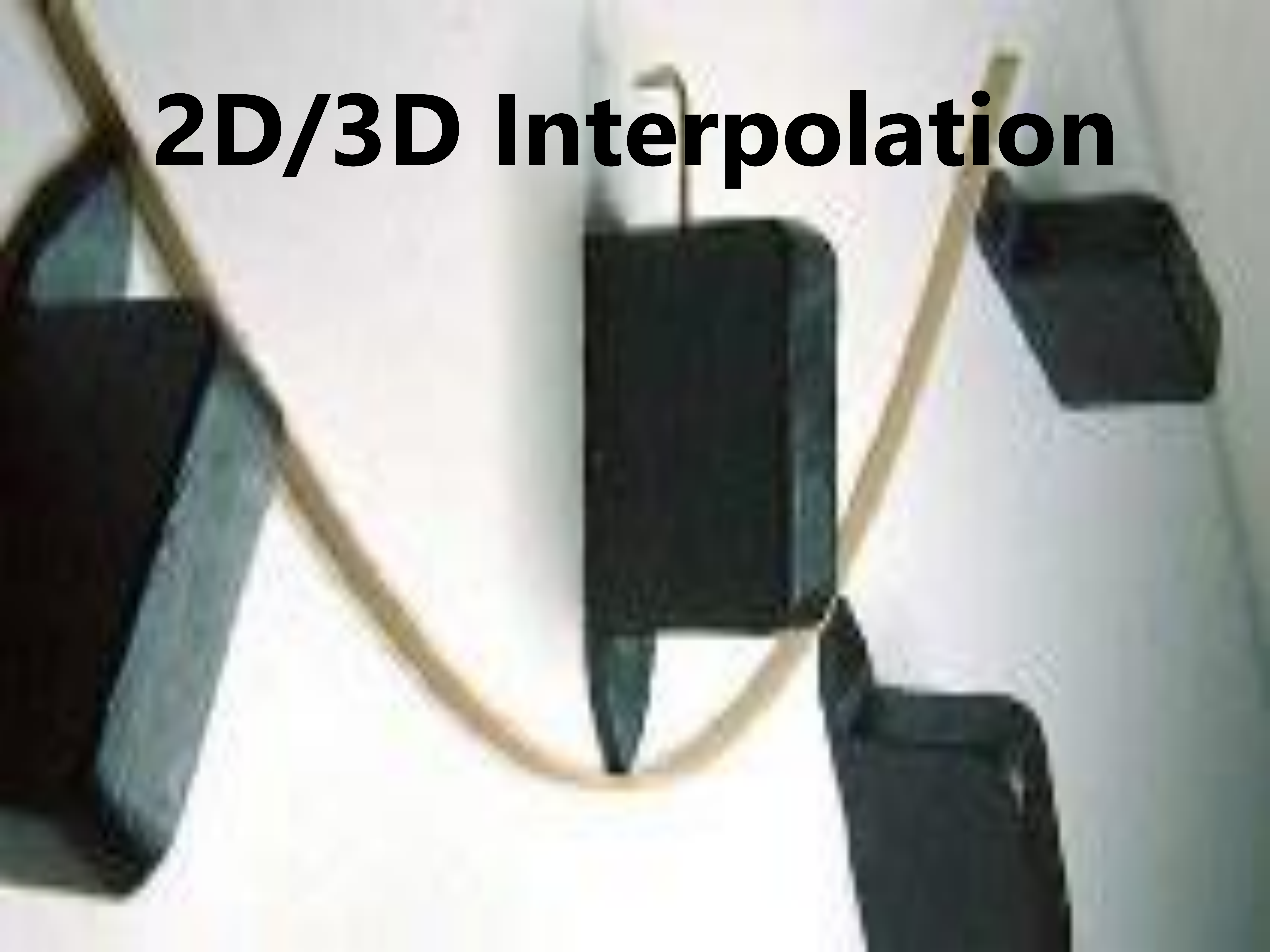
$$H_2(t) = t^3 - 2t^2 + t$$

$$H_3(t) = t^3 - t^2$$



No need for inverse/solve, just add together splines: $\sum_{i=0}^{3} h_i H_i(t)$

2D/3D Interpolation



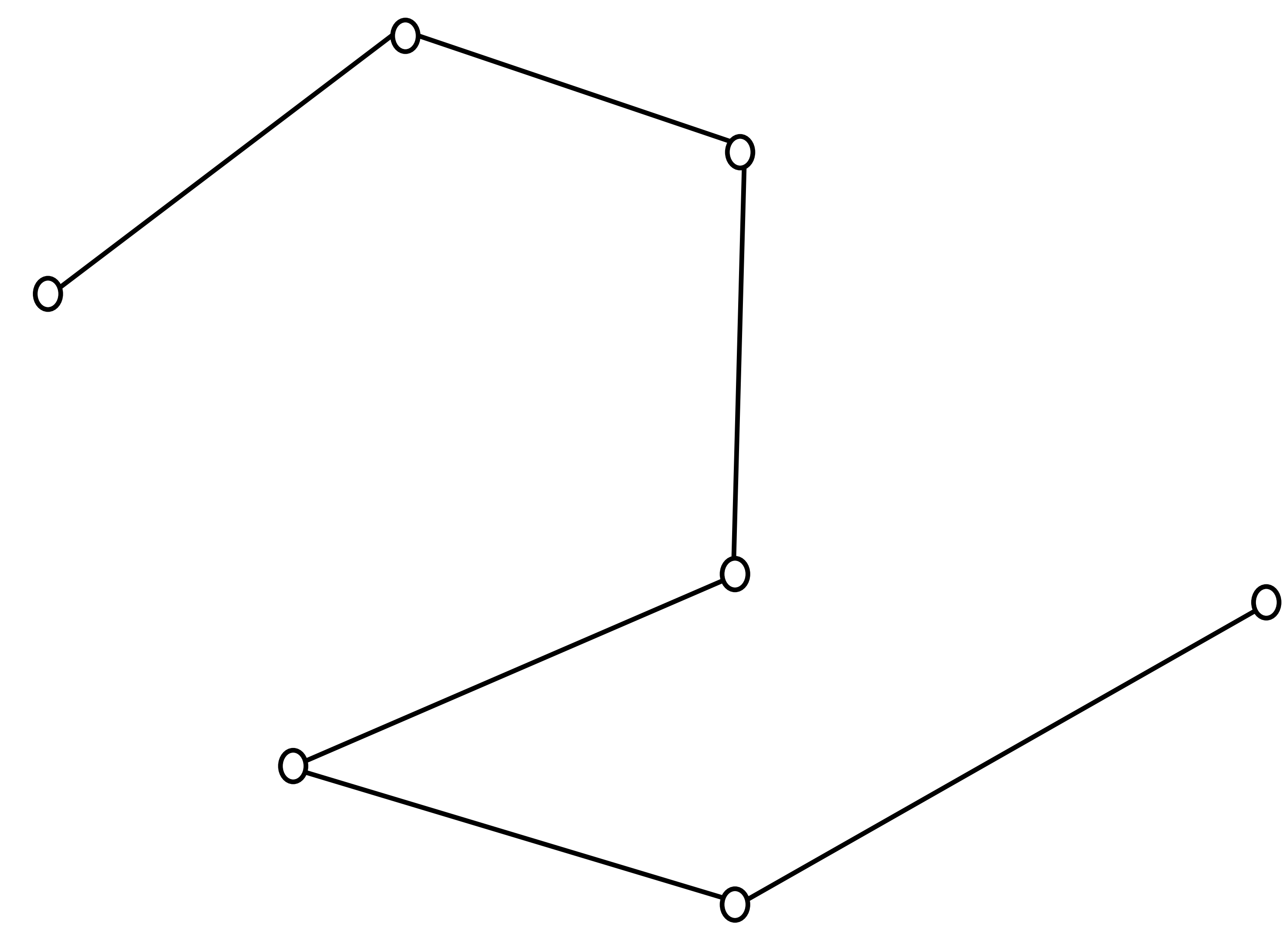
Piecewise Linear 2D Interpolation

- **Draw a line between each pair of points**
- **Consider linear interpolation between points $\mathbf{p_0} = (x_0, y_0)$ and $\mathbf{p_1} = (x_1, y_1)$:**

$$\mathbf{p}(t) = (1-t)\mathbf{p}_0 + t\mathbf{p}_1$$

or

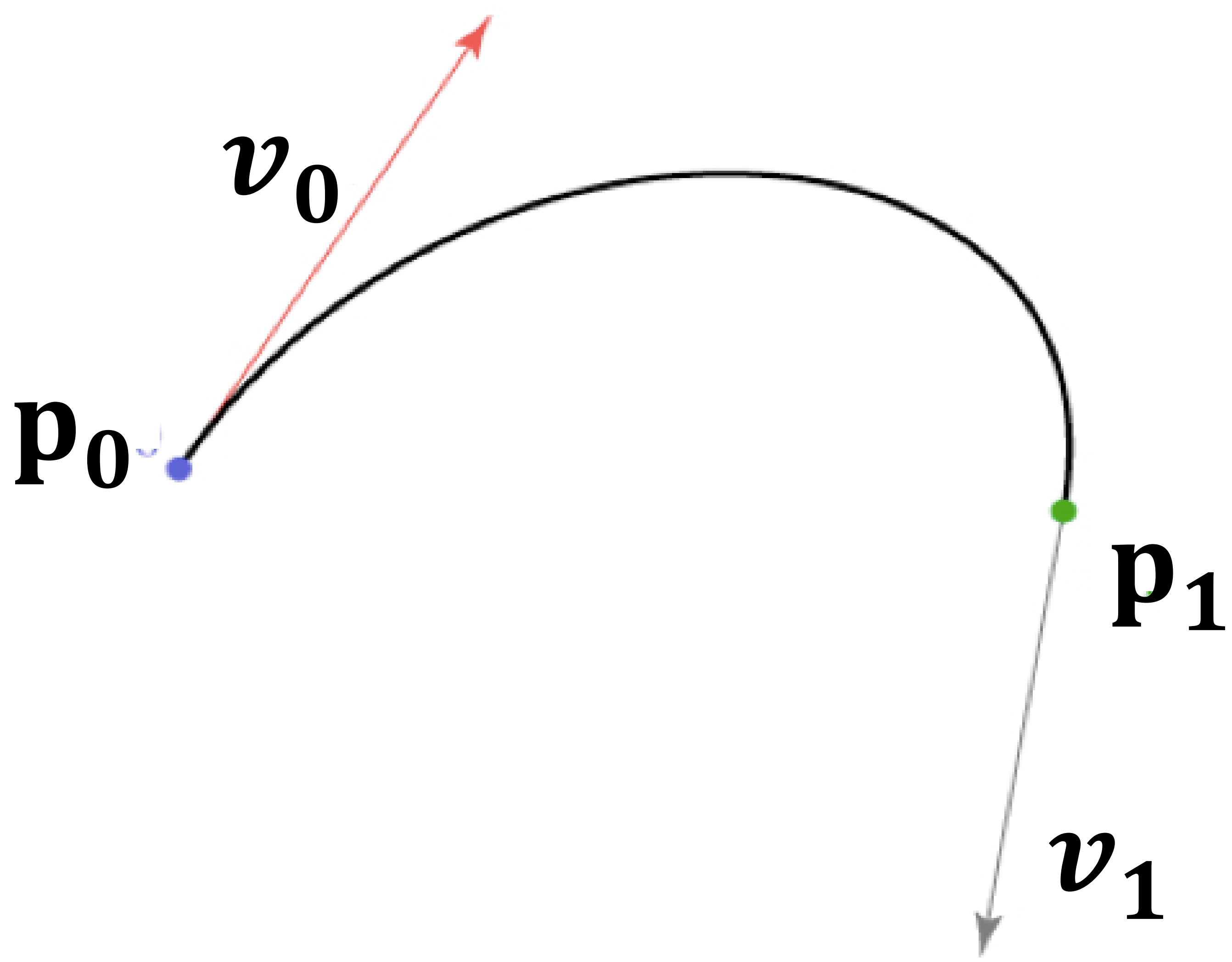
$$x(t) = (1-t)x_0 + tx_1$$
$$y(t) = (1-t)y_0 + ty_1$$



2D/3D Hermite Curves

- Given two points $\mathbf{p}_0 = (x(0), y(0))$ and $\mathbf{p}_1 = (x(1), y(1))$
- To apply cubic Hermite interpolation to each component, we also need derivative constraints: $x'(0), y'(0), x'(1), y'(1)$
- $(x'(0), y'(0))$ is the (un-normalized) direction of the tangent at $\mathbf{p}_0$
  - Different magnitudes of (x', y') at end points lead to different Hermite splines!

$v_0 = (x'(0), y'(0))$



$v_1 = (x'(1), y'(1))$

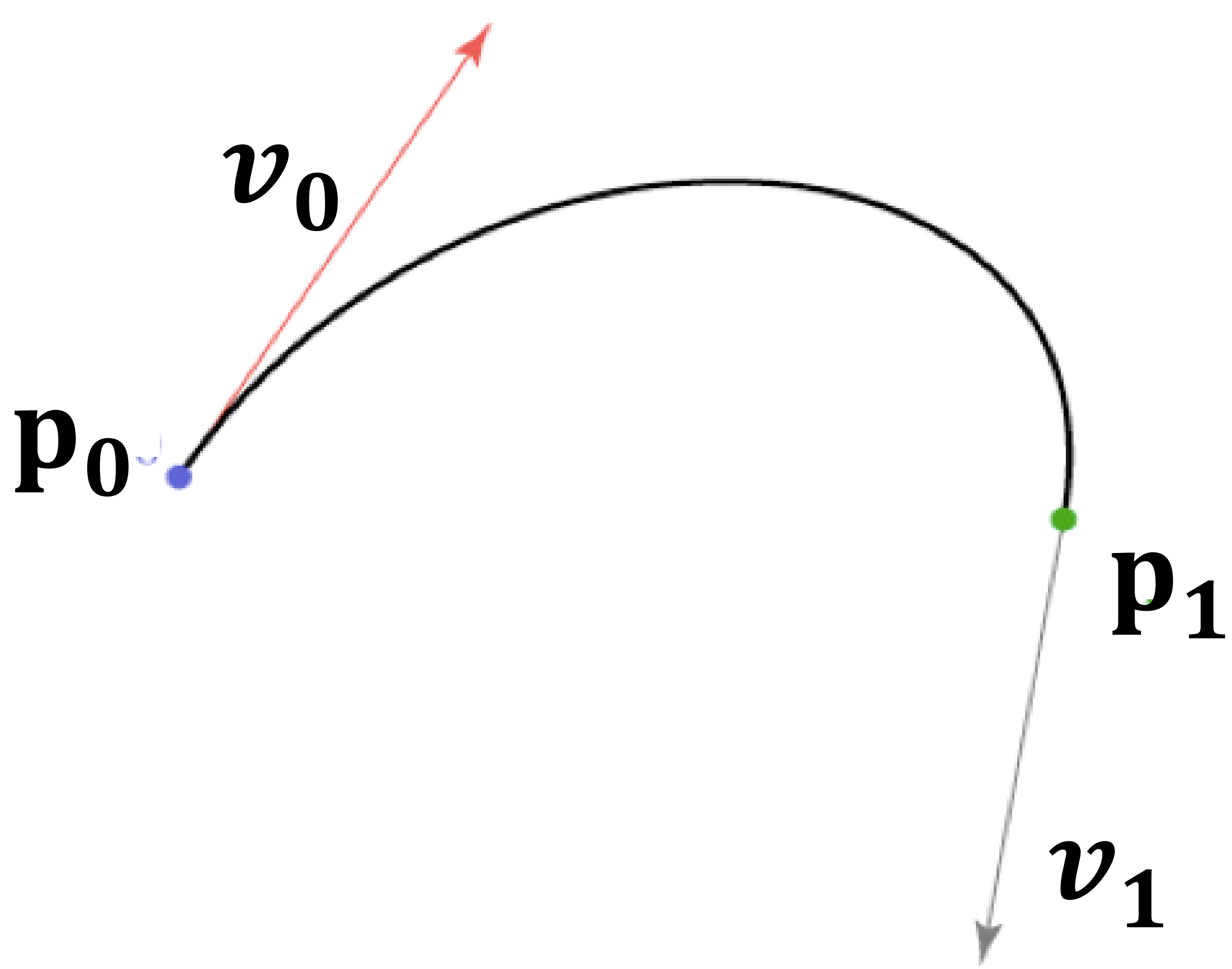
2D/3D Hermite Curves

- Given two points $\mathbf{p}_0 = (x(0), y(0))$ and $\mathbf{p}_1 = (x(1), y(1))$
- To apply cubic Hermite interpolation to each component, we also need derivative constraints: $x'(0), y'(0), x'(1), y'(1)$
-

Arrows!

s!

$v_0 = (x'(0), y'(0))$



$v_1 = (x'(1), y'(1))$

Basis Functions for 2D/3D Curves

- Consider interpolation between points $\mathbf{p}_i = (x_i, y_i)$ for $i = 0,1, \dots, n$.
- Each component of $\mathbf{p}(t)$ is independently interpolated via:

$$x(t) = \sum_{i=0}^{n} x_i W_i(t) \qquad\qquad y(t) = \sum_{i=0}^{n} y_i W_i(t)$$

- Since the basis functions are the same for every component, we get:

$$\mathbf{p}(t) = \sum_{i=0}^{n} \mathbf{p}_i W_i(t)$$

2D/3D Hermite Curves

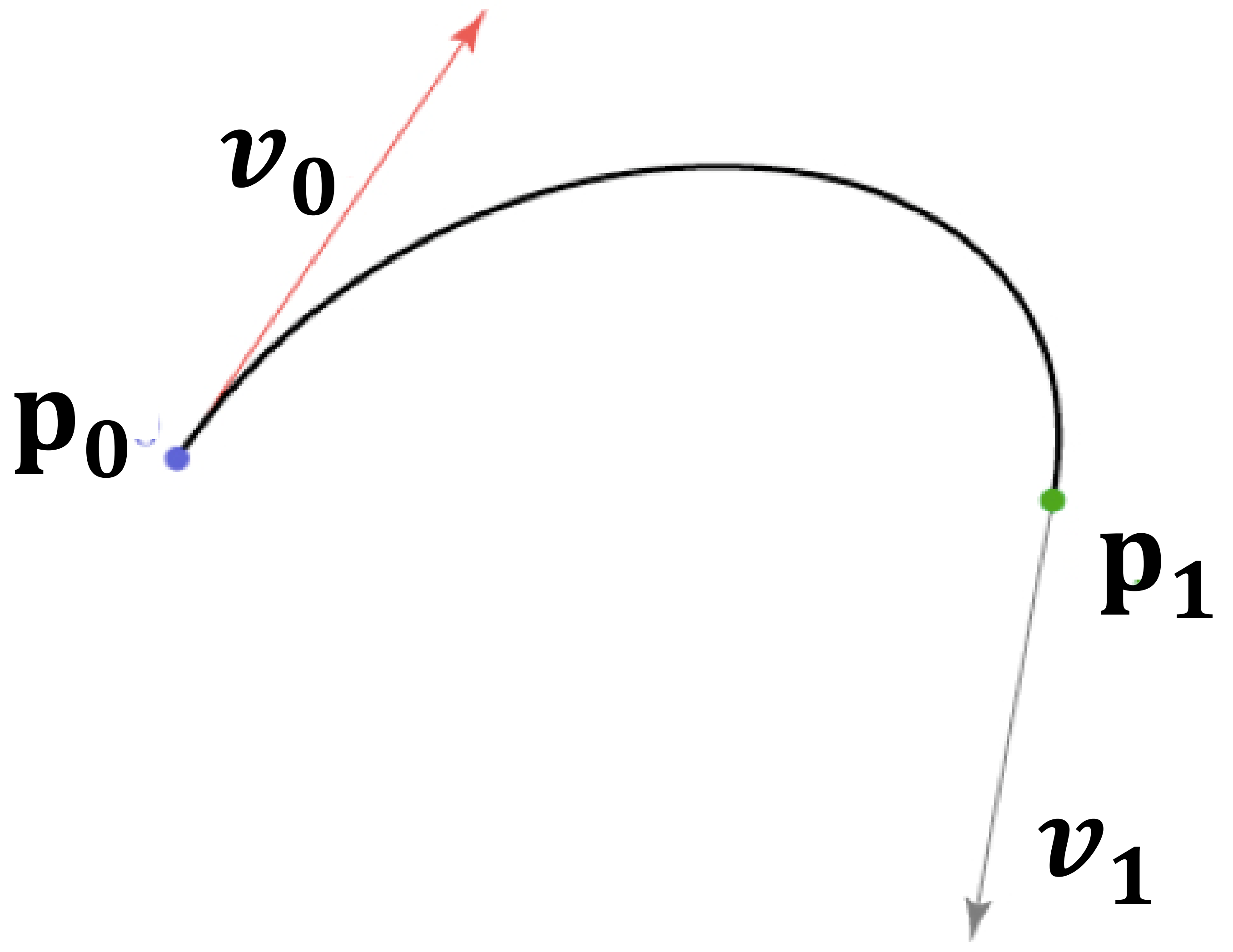
- Cubic Hermite interpolation for each component

$$x(t) = H_0(t)x(0) + H_1(t)x(1) + H_2(t)x'(0) + H_3(t)x'(1)$$

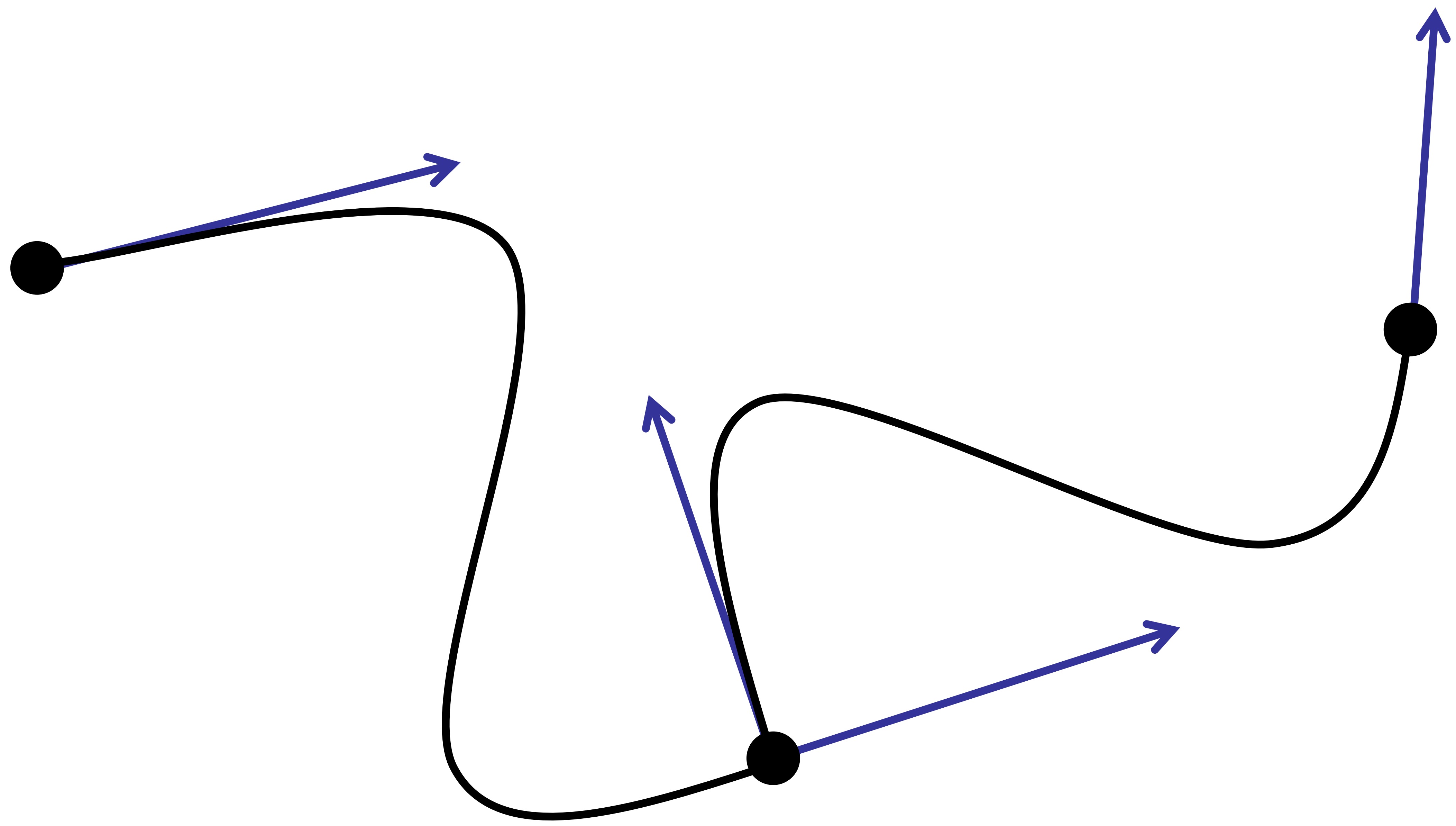
$$y(t) = H_0(t)y(0) + H_1(t)y(1) + H_2(t)y'(0) + H_3(t)y'(1)$$

- Assemble the equations of each component to get the 2D interpolation equation

$$\mathbf{p}(t) = H_0(t)\mathbf{p}_0 + H_1(t)\mathbf{p}_1 + H_2(t)\boldsymbol{v}_0 + H_3(t)\boldsymbol{v}_1$$

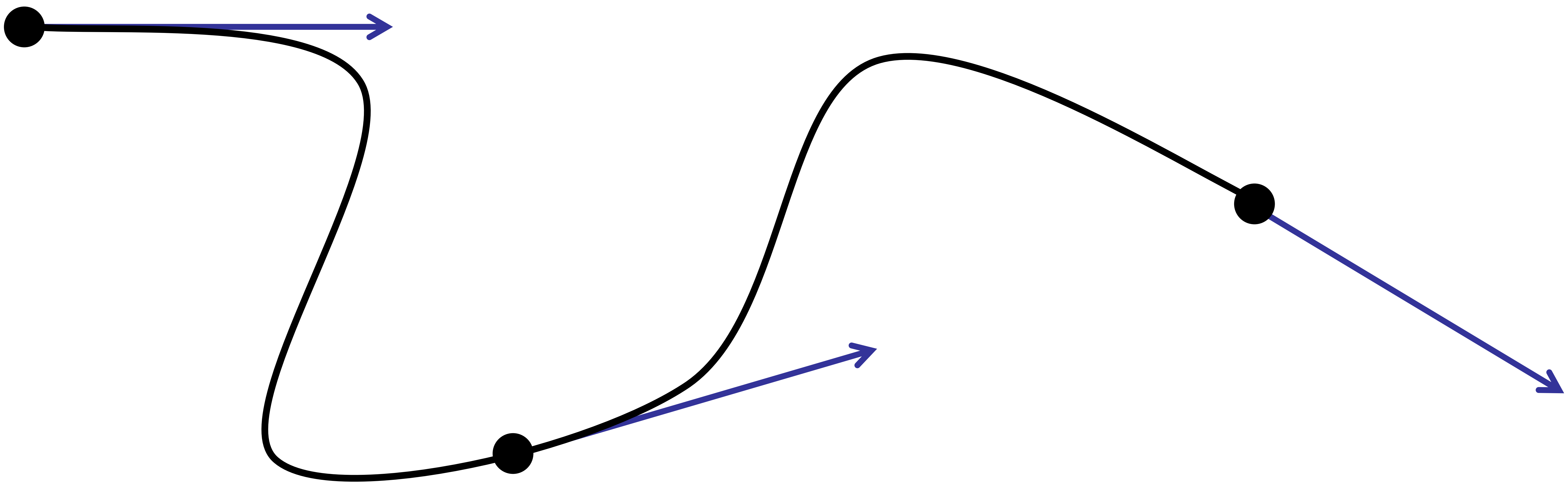


C^0 Continuity for 2D/3D Hermite Curves



- Points are specified continuously
- But tangents are specified discontinuously

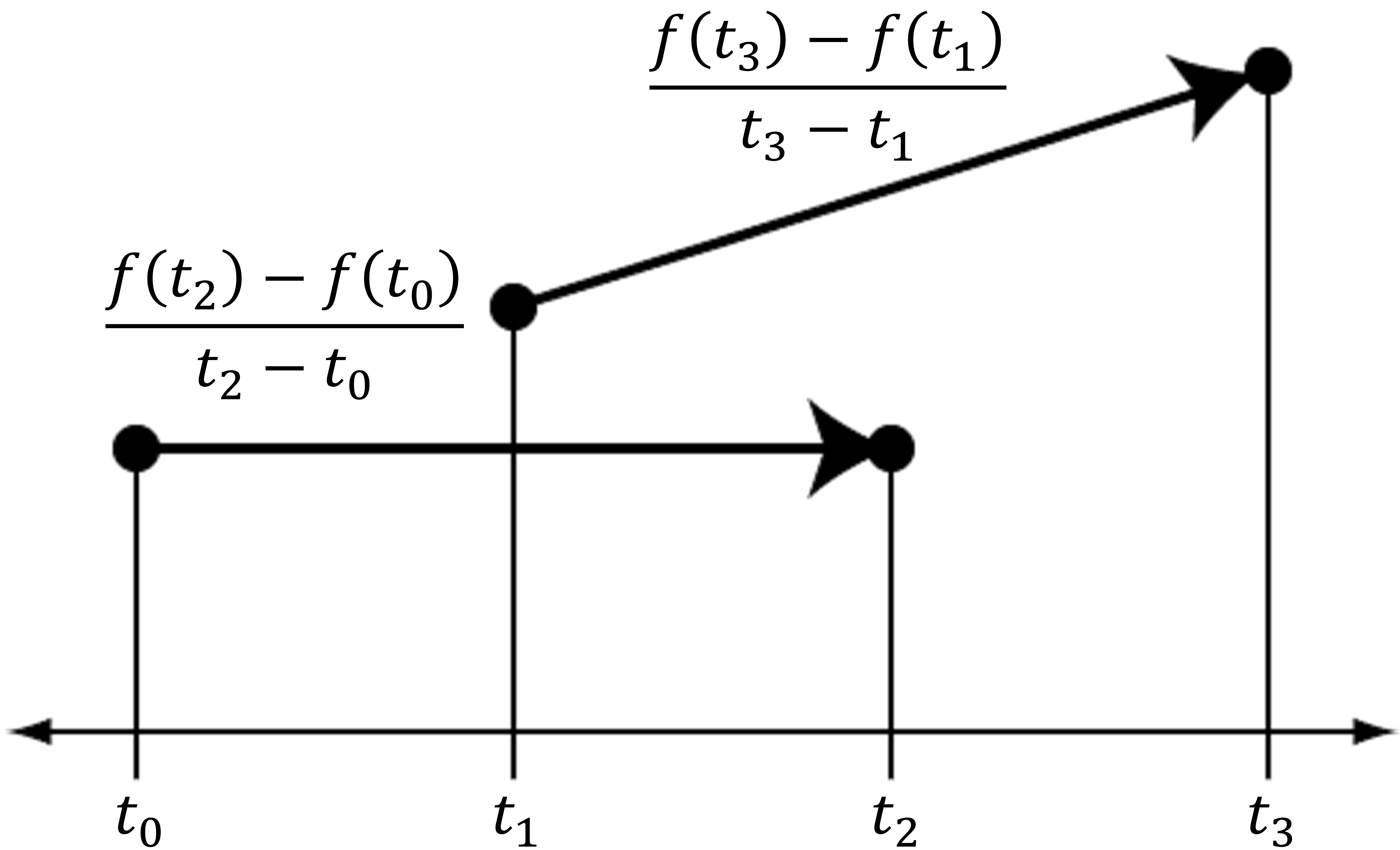
C^1 Continuity for 2D/3D Hermite Curves



- **Tangents are specified continuously as well**

Catmull-Rom

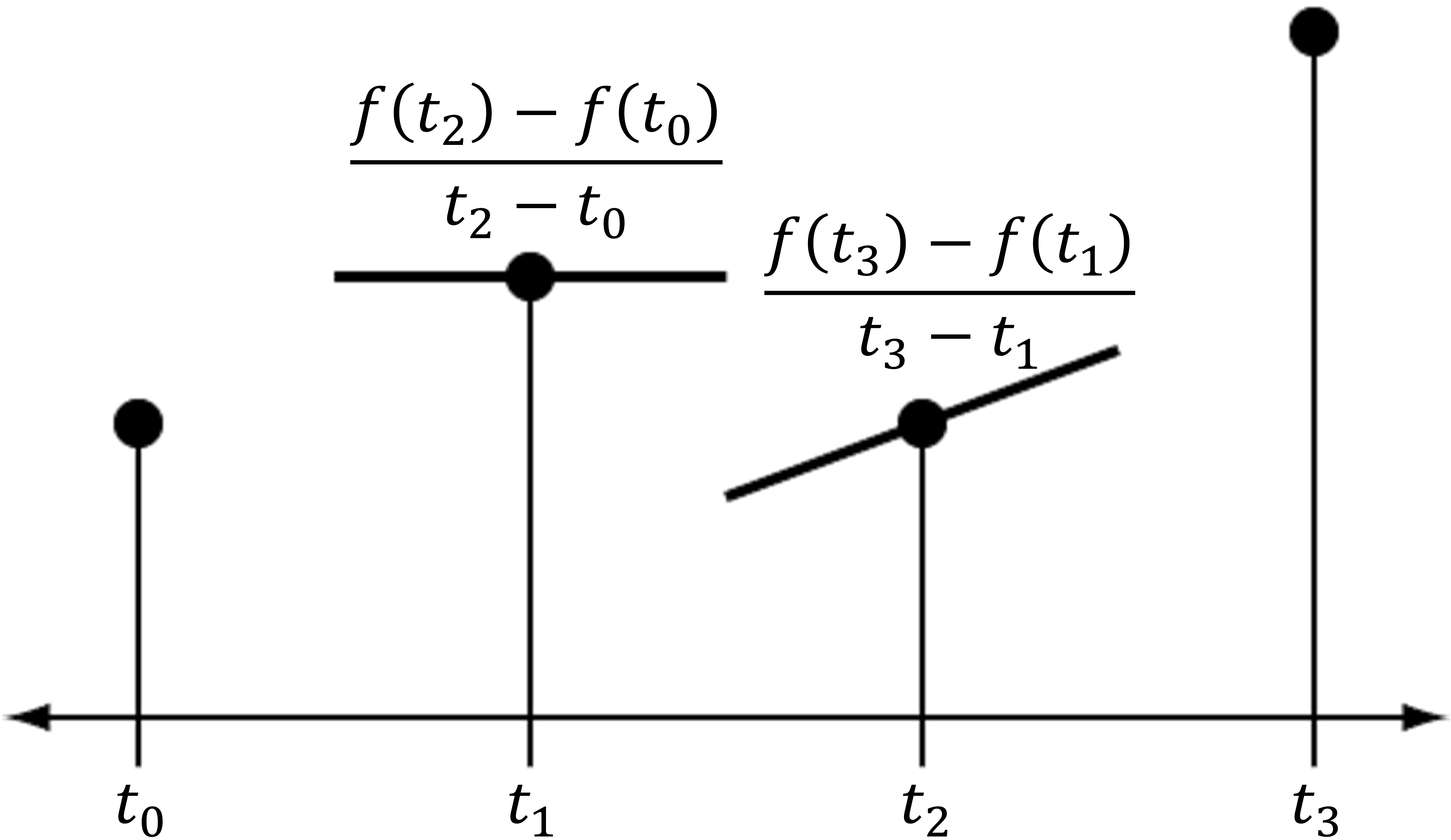
Catmull-Rom (for 1D Hermite) Interpolation



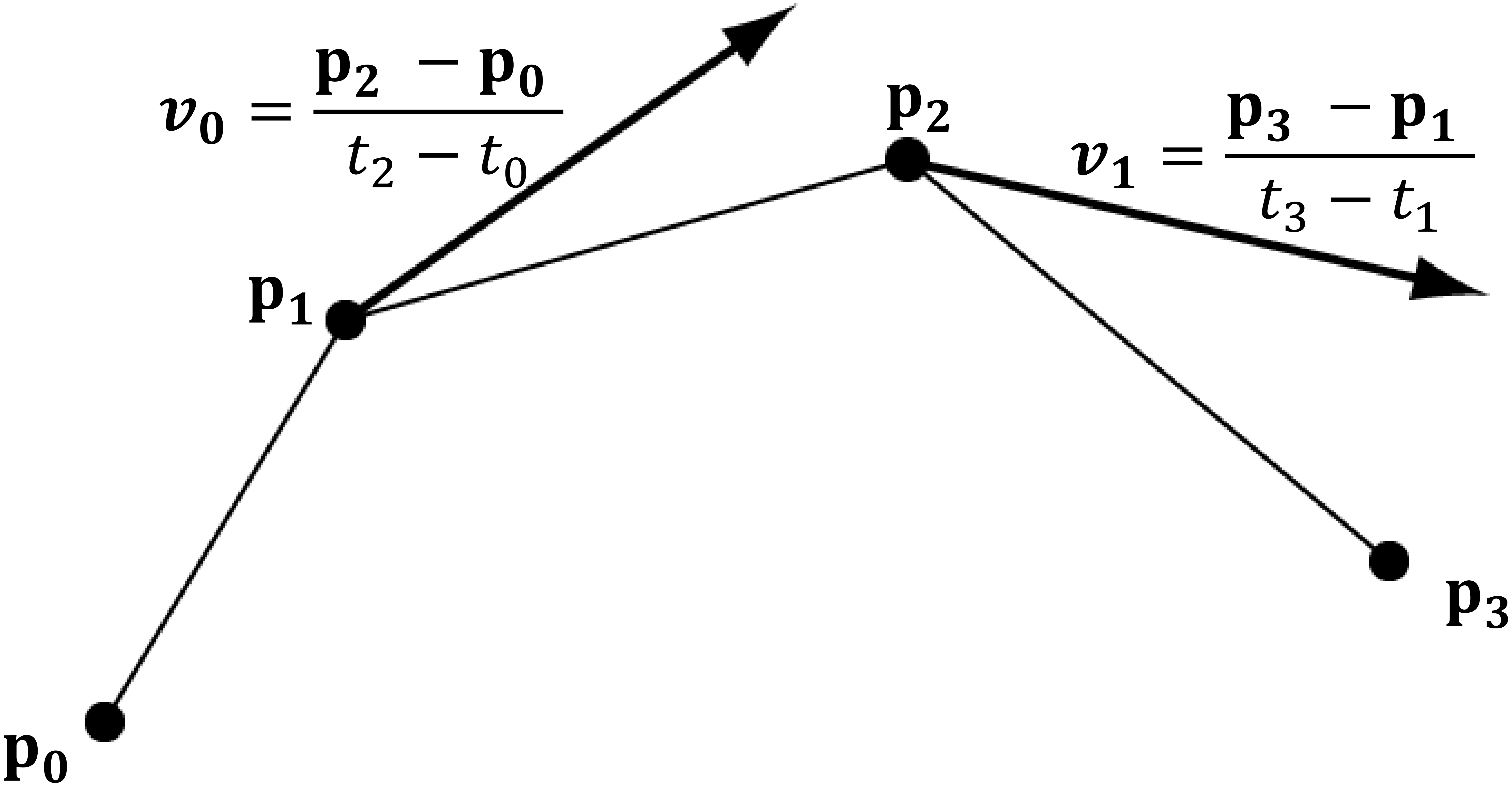
Automatically define derivatives as central differences

Catmull-Rom (for 1D Hermite) Interpolation

Then use Hermite Interpolation

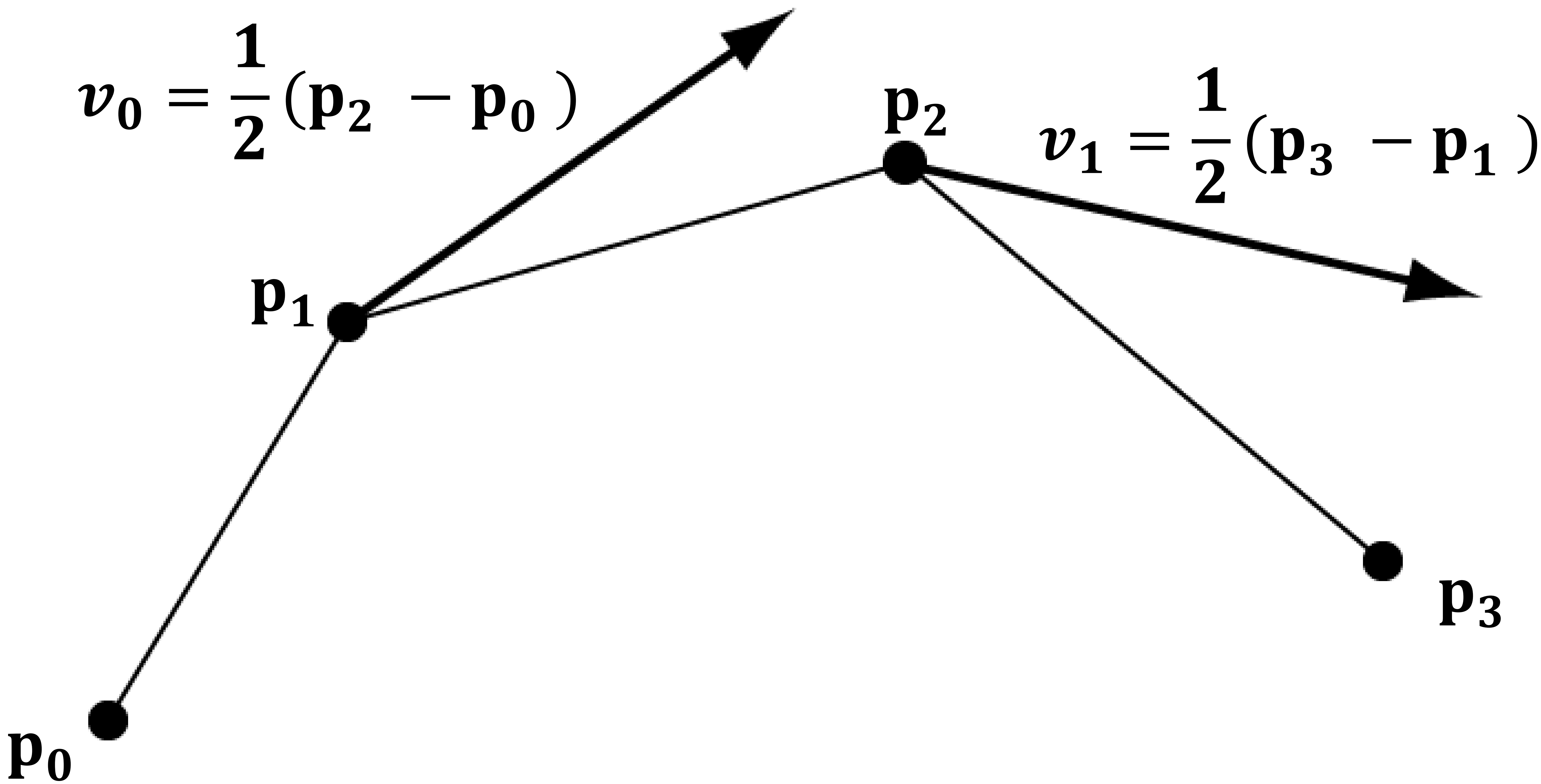


Catmull-Rom (for 2D/3D Hermite) Interpolation



- **We can define the derivatives with respect to a paramater t for each component using central difference**
- **But how do we choose t ?**

Catmull-Rom (for 2D/3D Hermite) Interpolation



- **Common to just set the spacing between points to be** 1**, then** $t_{k+1} - t_{k-1} = 2$