Animation Basics

Giovanni Dicanio
giovanni.dicanio@gmail.com

pluralsight
hardcore dev and IT training

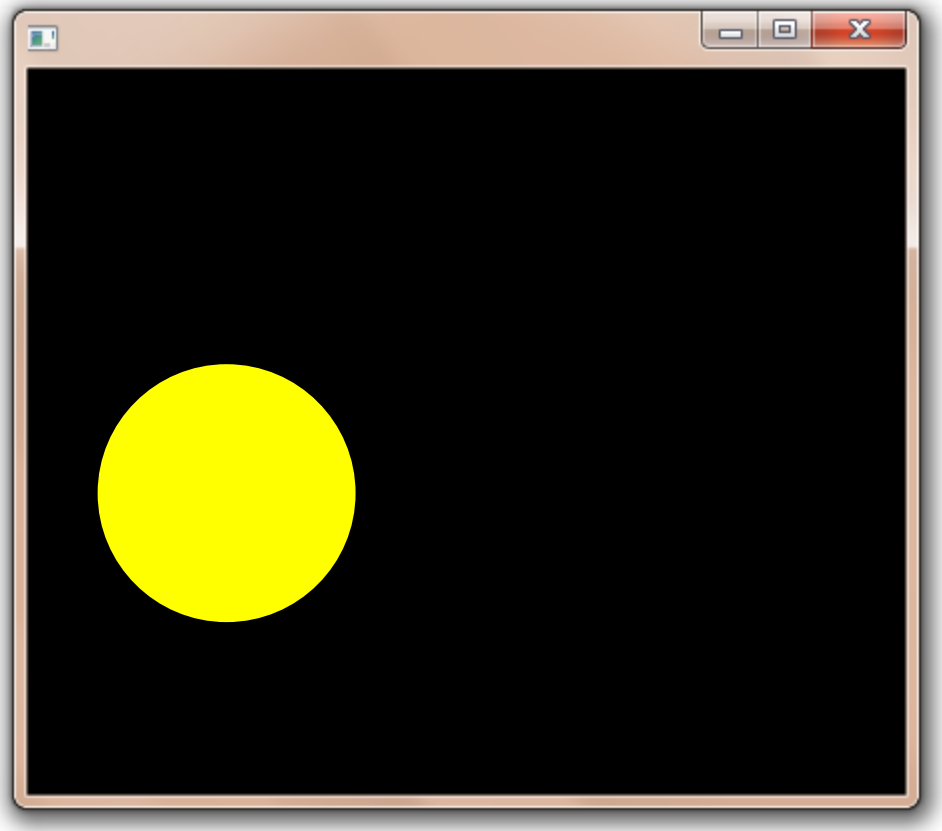
Topics

- **Animation mechanics**
  - Building simple animations
  - Animating along a path
- **Animating from a starting point to a target position**
  - Easing functions
- **Animating multiple particles**
- **Timeline (brief introduction)**
- **Demos**

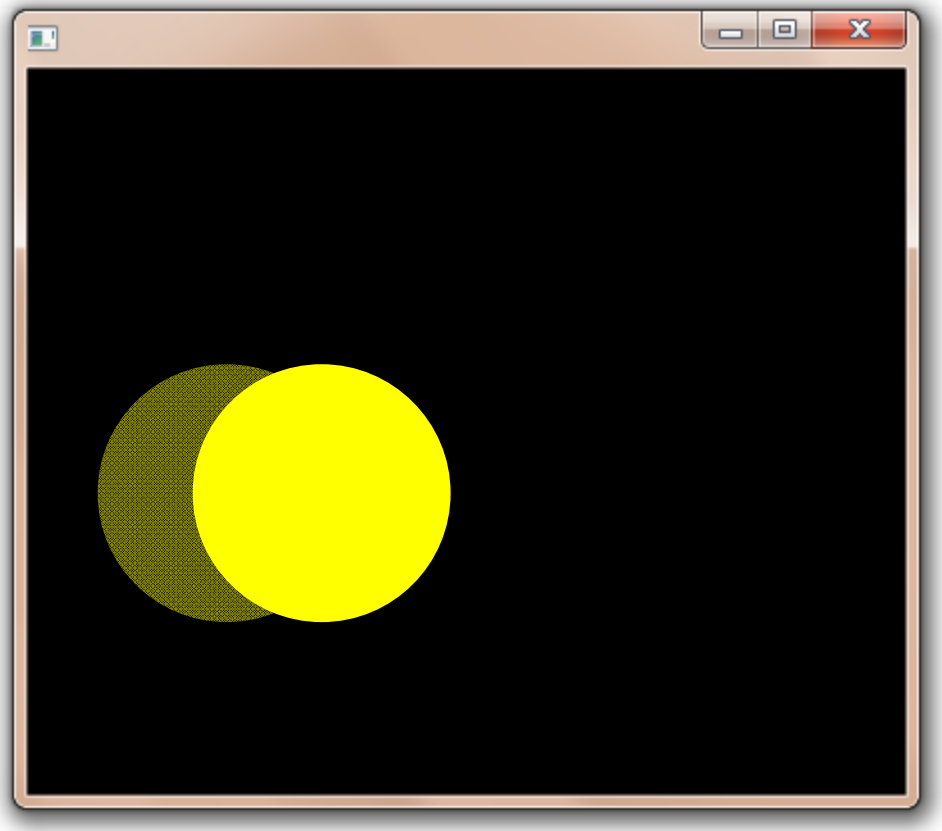
Animation Mechanics

Drawing Slightly Different Frames

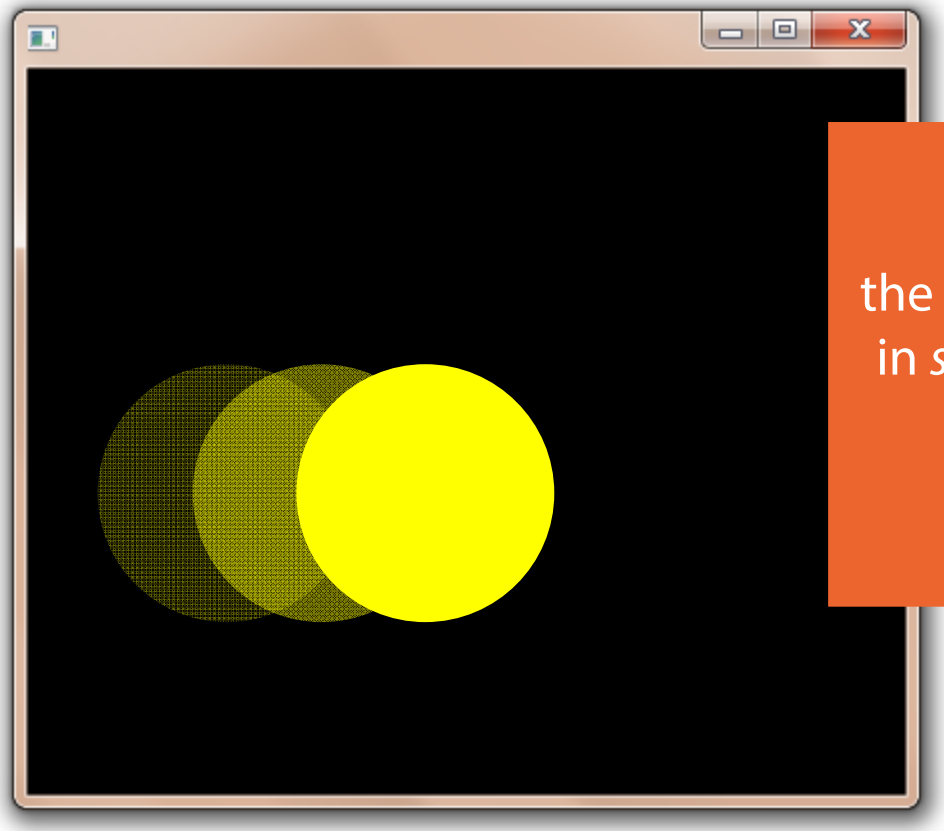
Drawing Slightly Different Frames



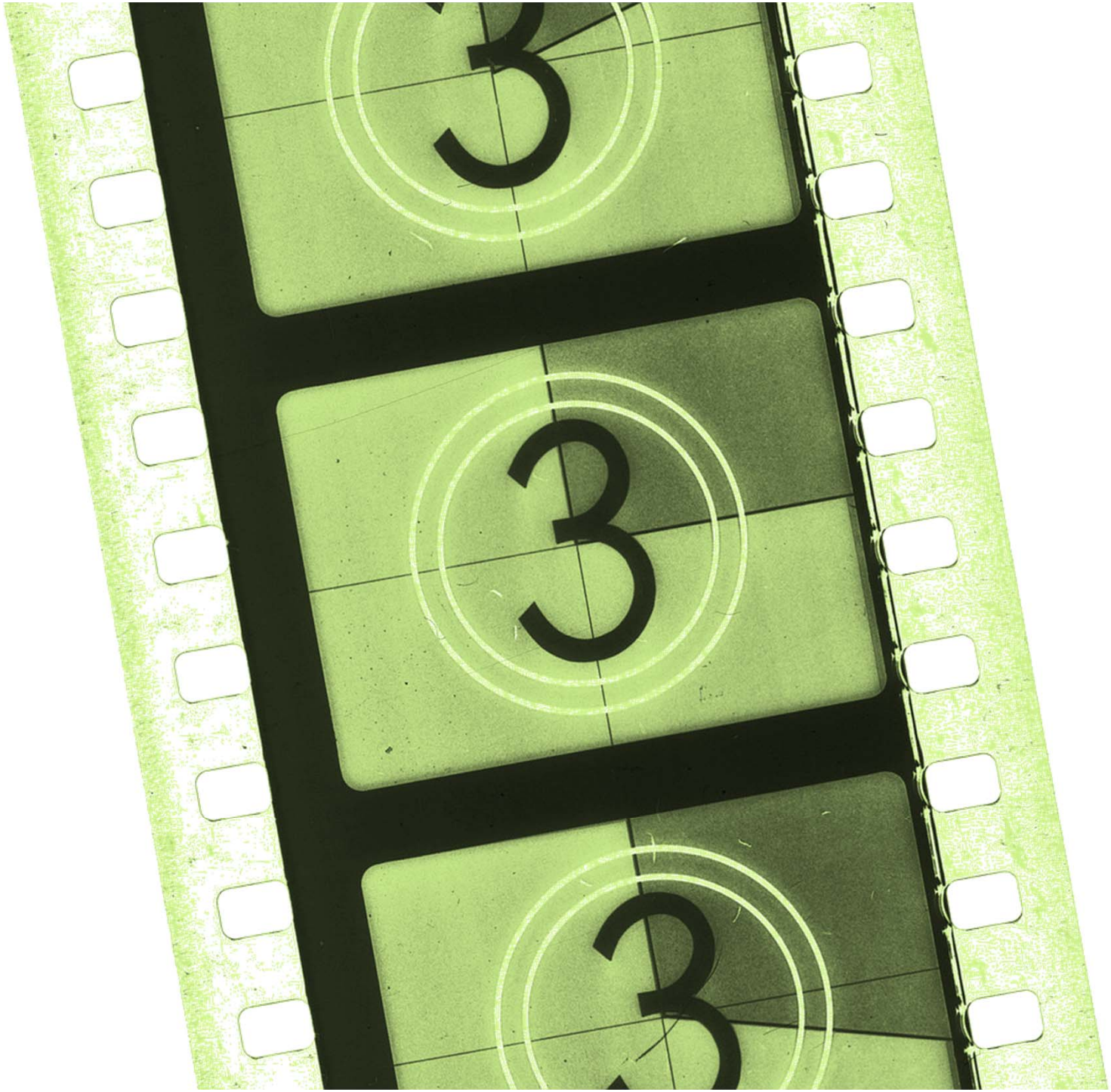
Drawing Slightly Different Frames

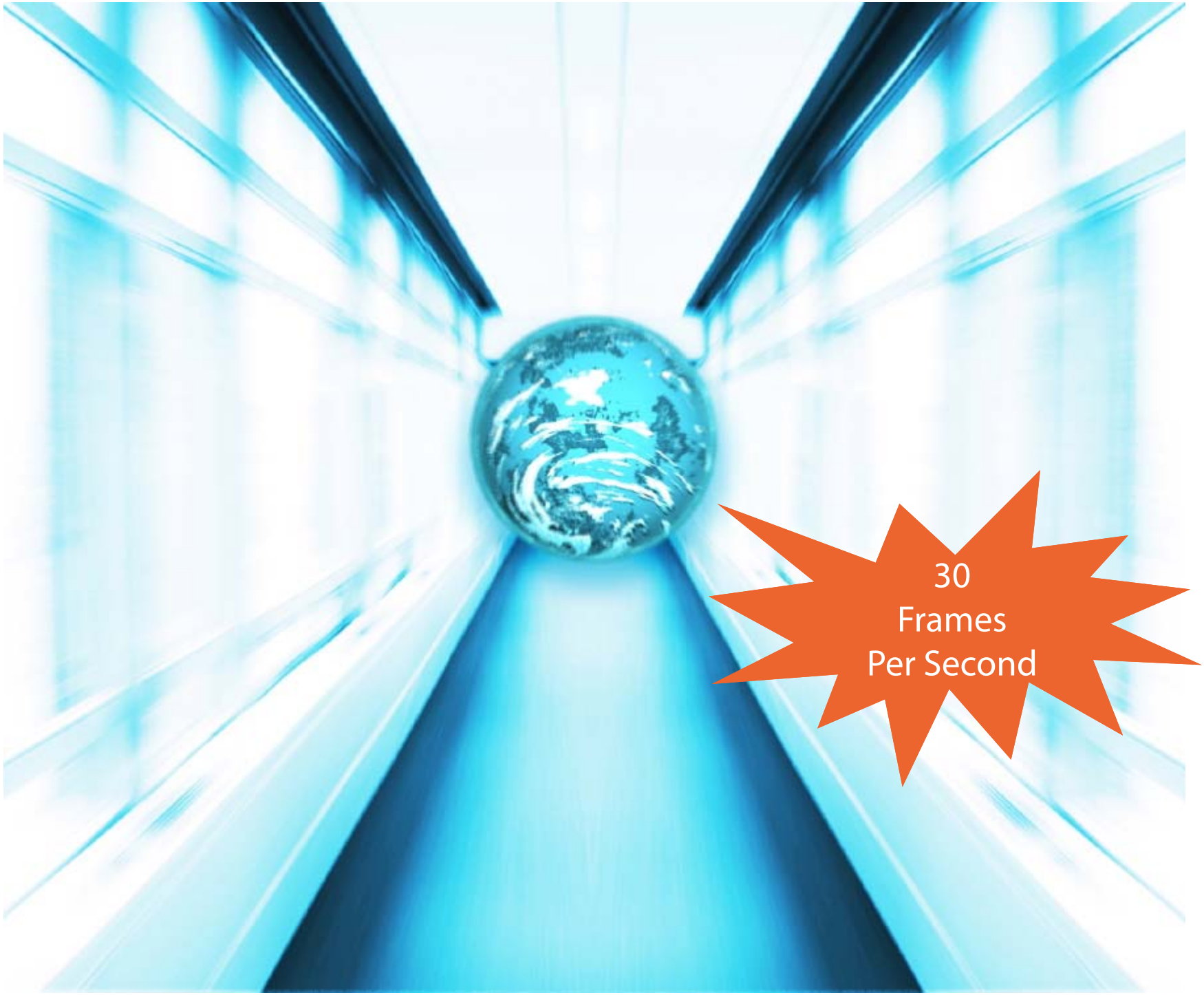


Drawing Slightly Different Frames



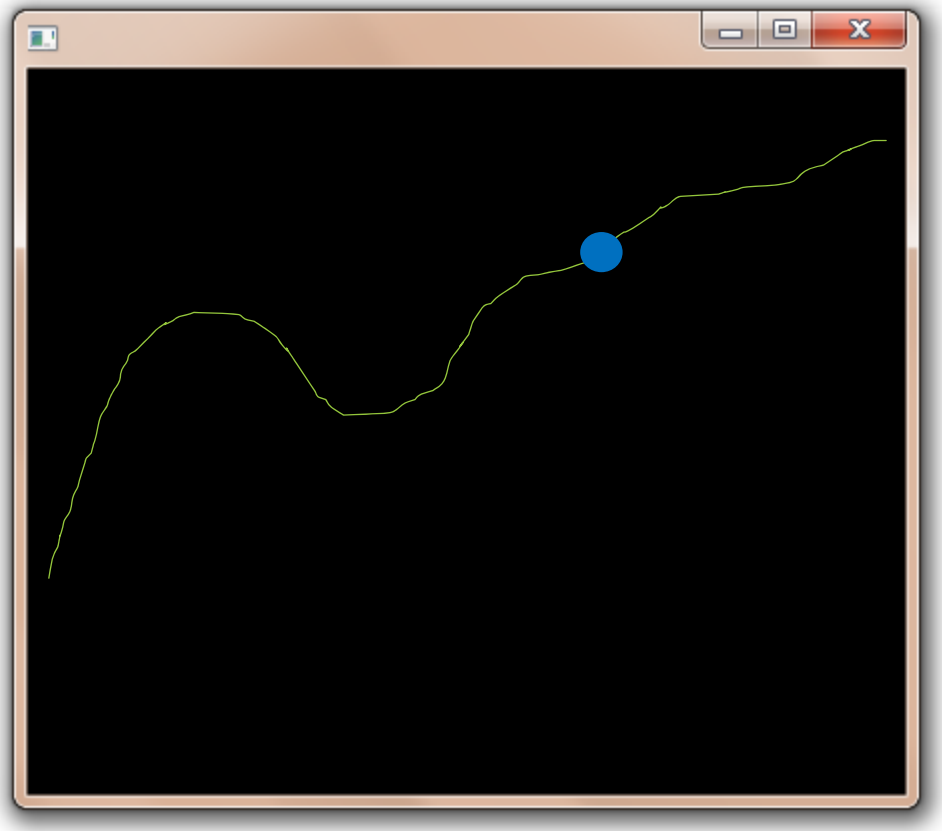
Draw
the moving objects
in *slightly different*
positions


3
3
3


30
Frames
Per Second

Demo: Our First Simple Animation With Cinder

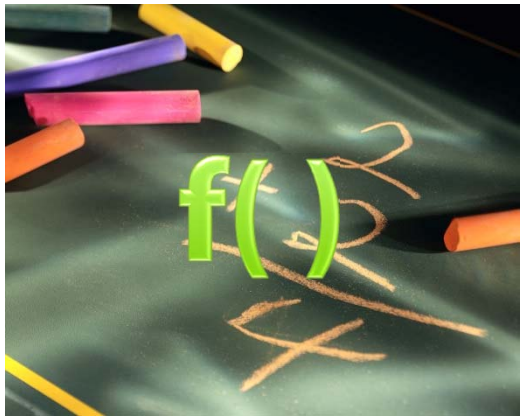
Animating Along a Path



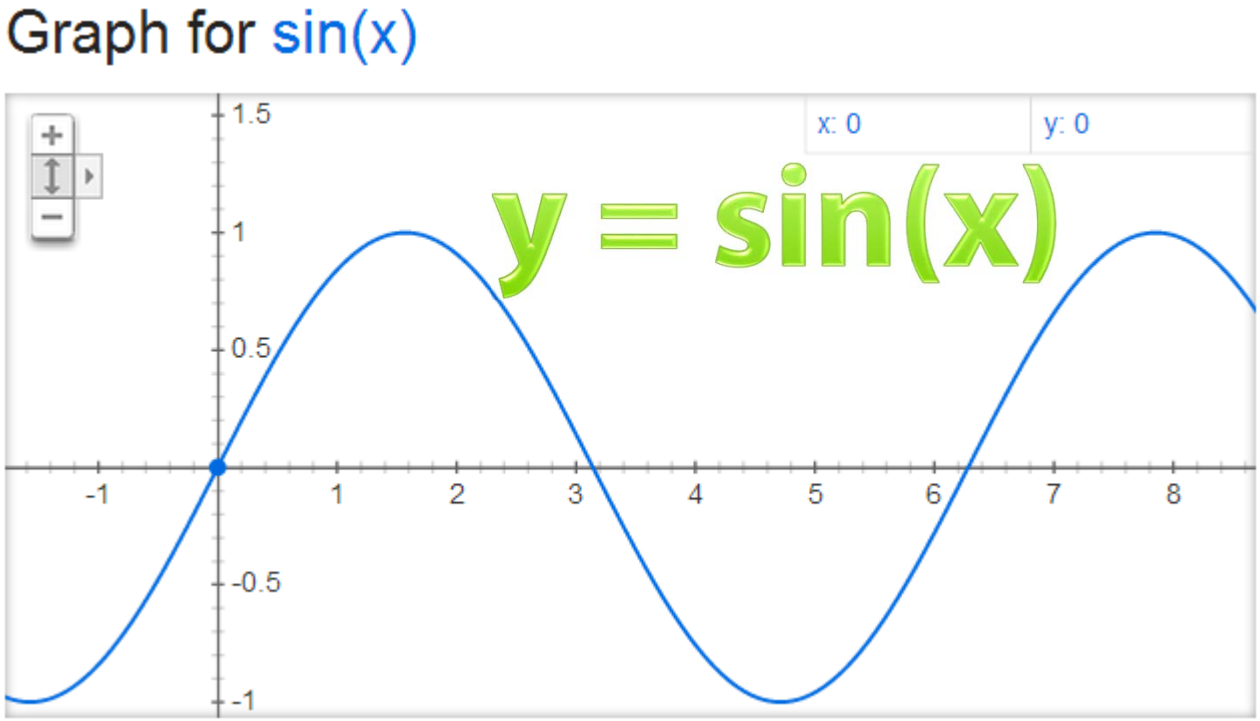
Defining the Animation Path

$$y = f(x)$$

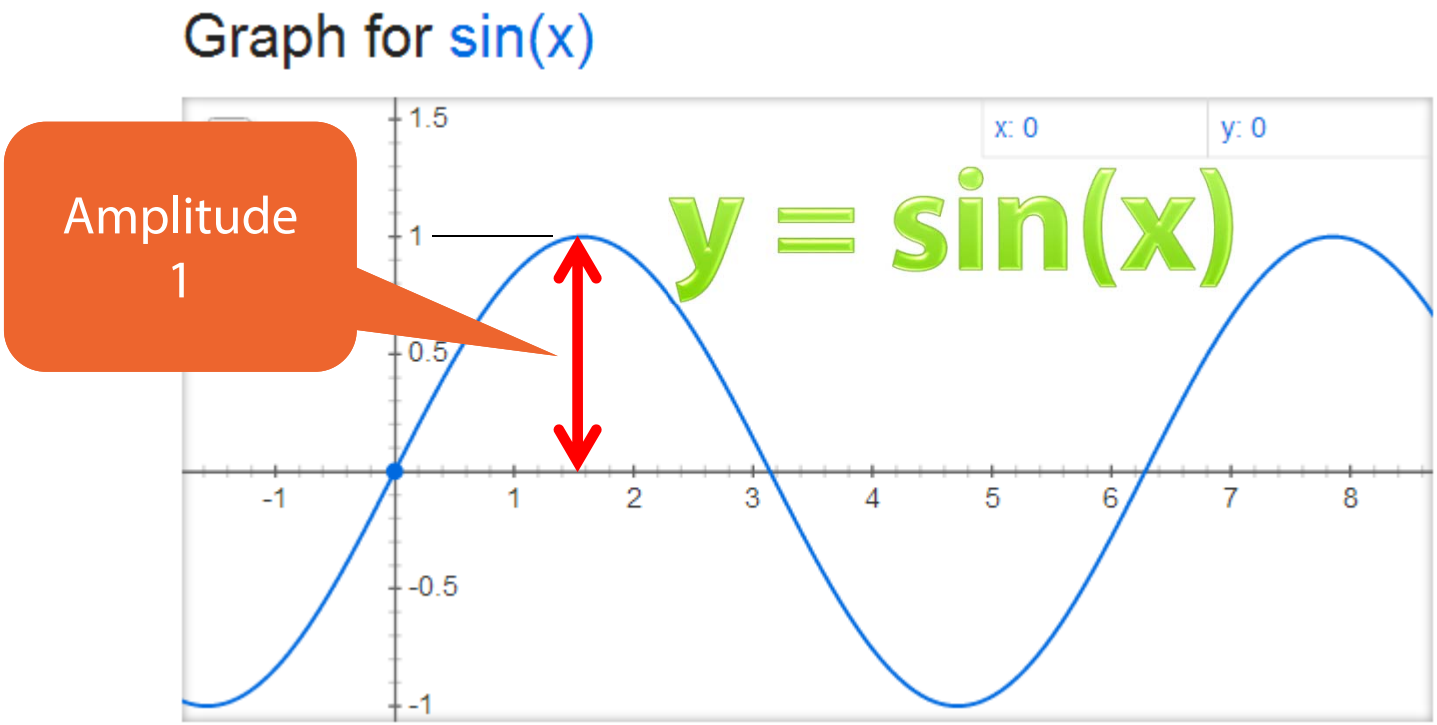
x and *y*
define
a 2D position

x →  → y

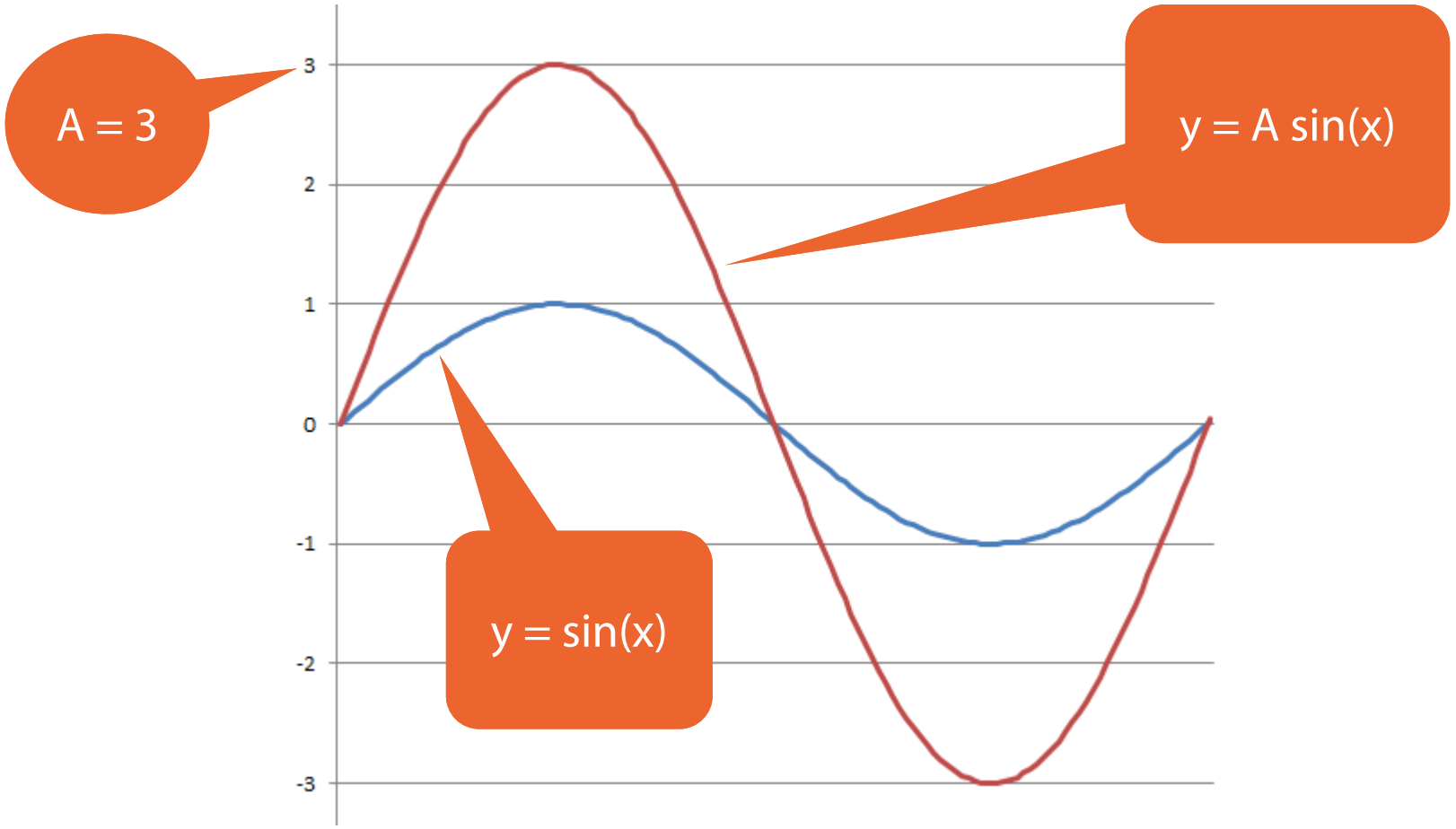
Sinusoidal Path



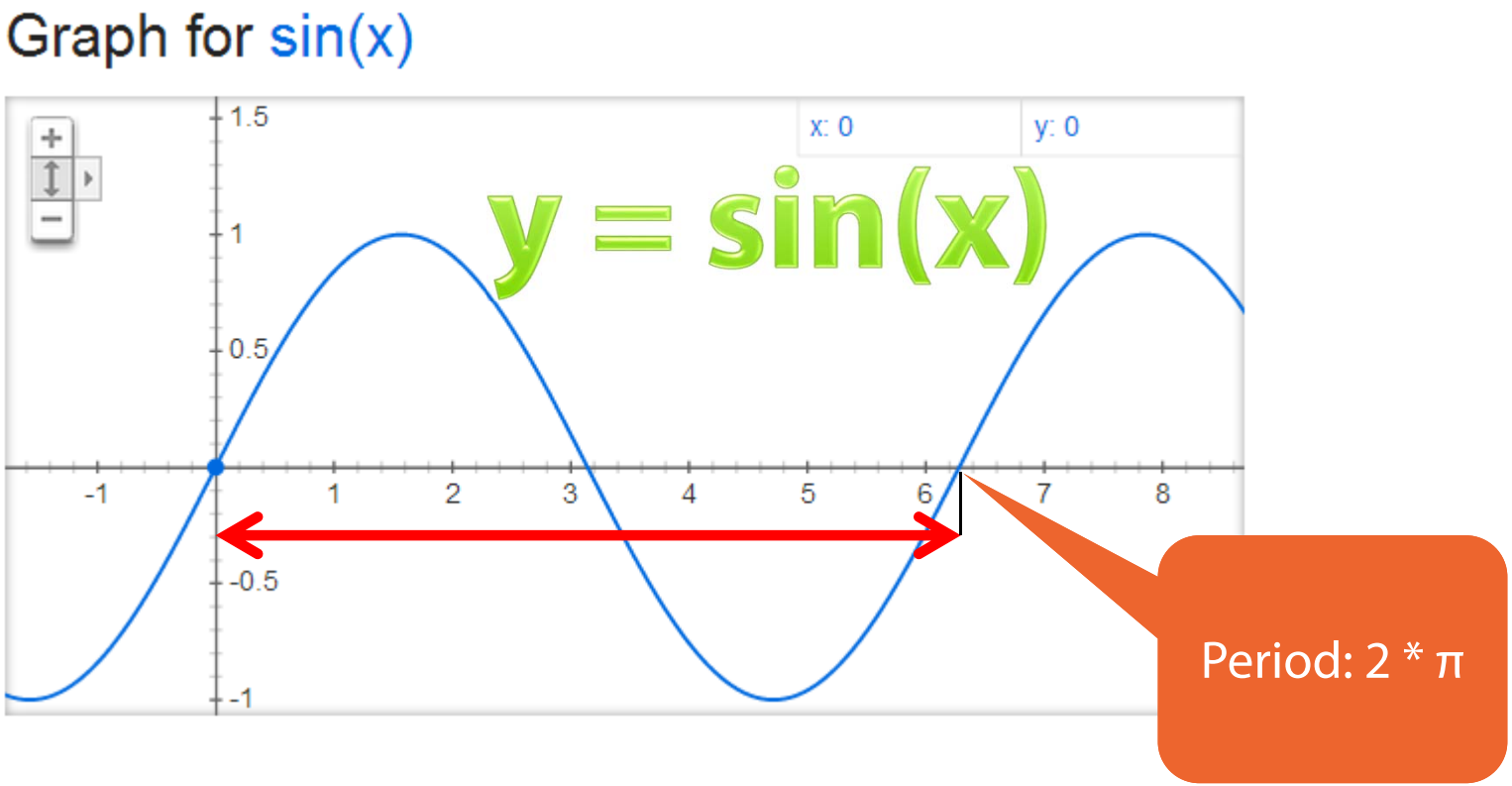
The Amplitude of the Sine Function



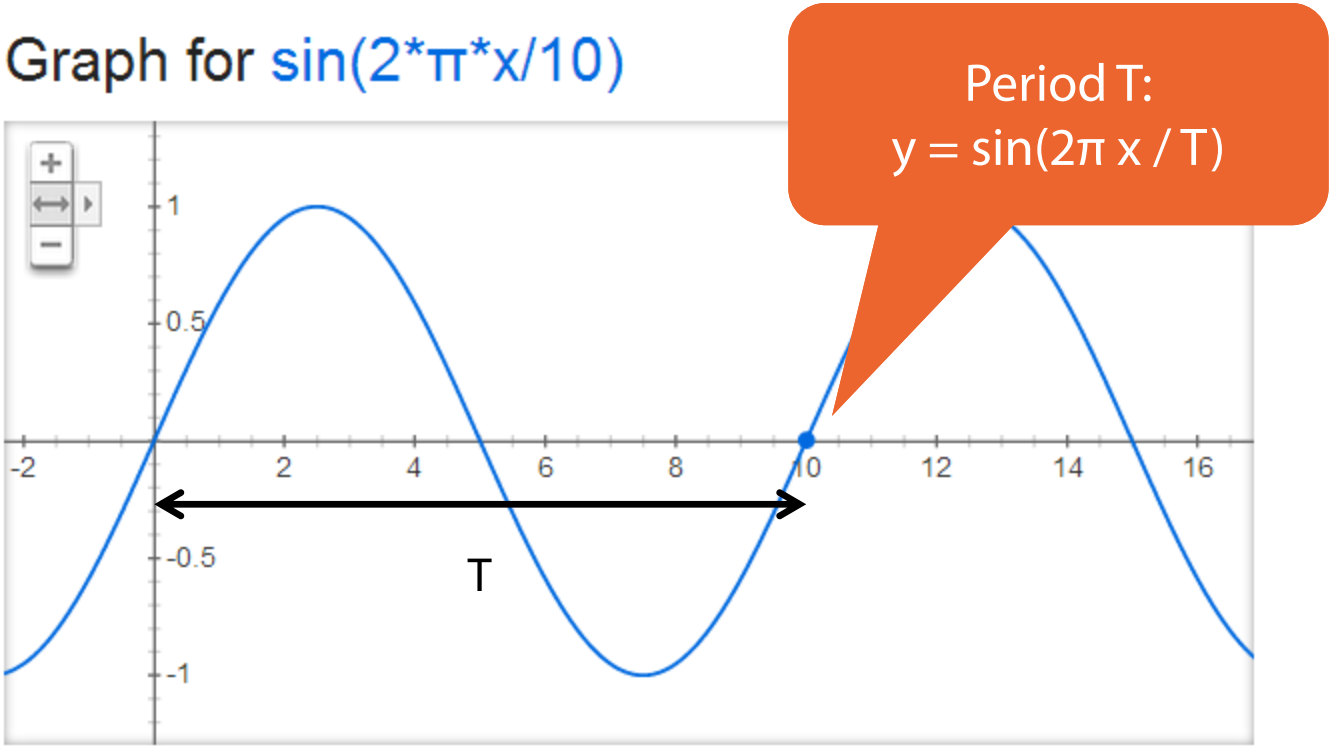
Changing the Amplitude



The Period of the Sine Function



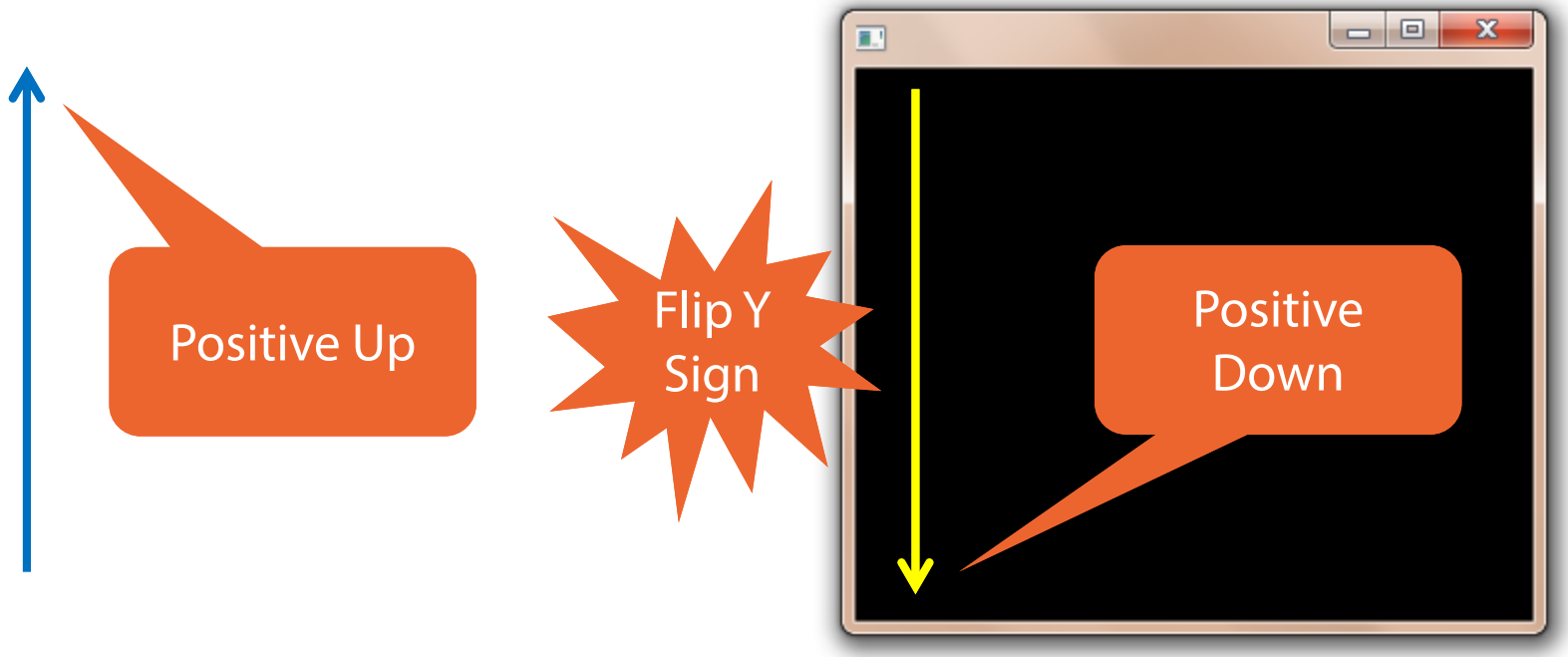
Changing the Period of the Sine Function



More Generic Formula for a Sinusoidal Path

$$y = A \sin(2\pi x / T)$$

The Positive Direction of the Y Axis



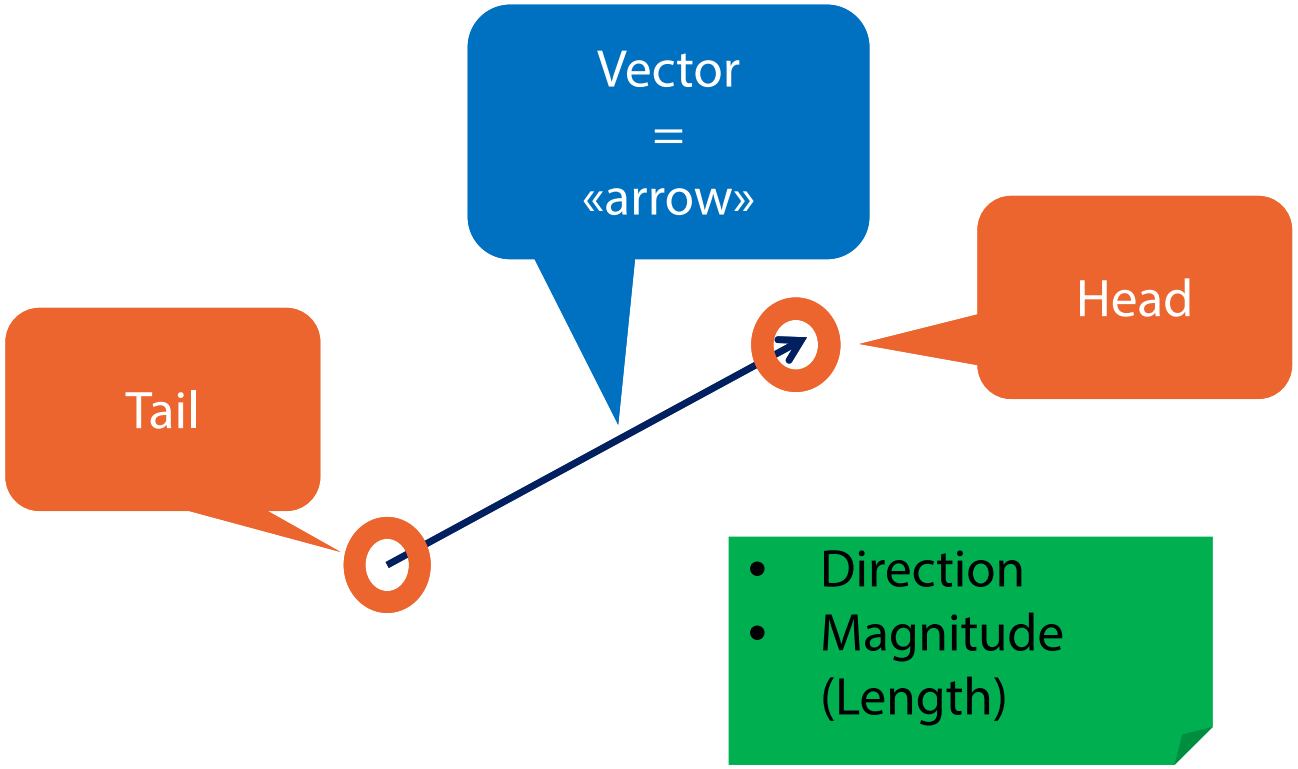
Demo: Animating Along a Sinusoidal Path

Reaching a Target Position

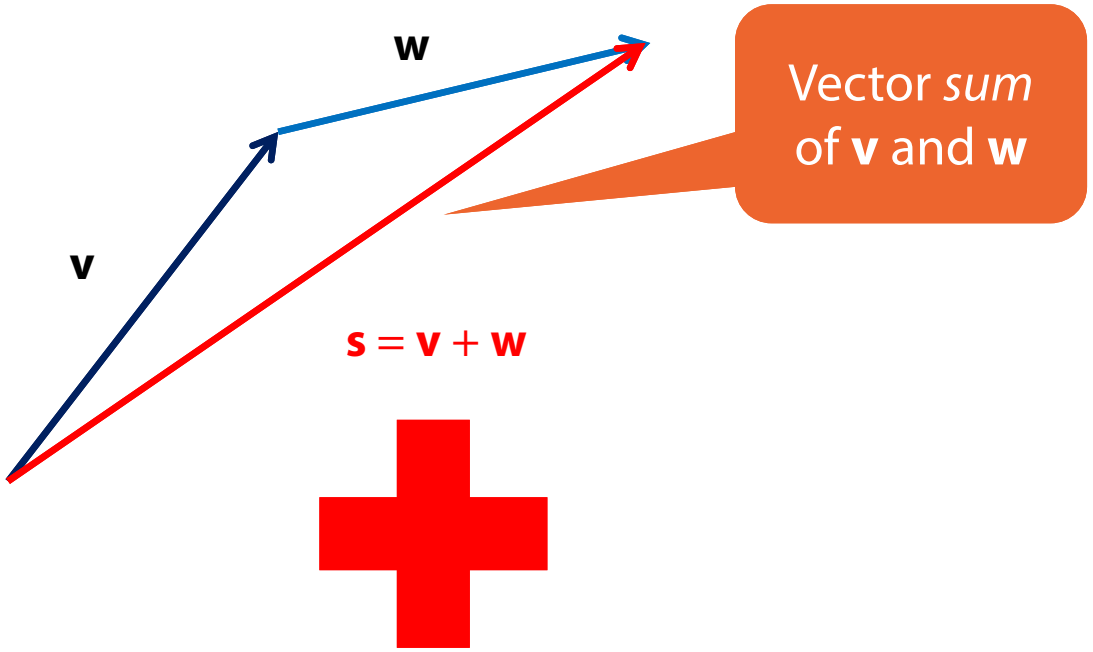
Reaching a Target Position

We need some *math*…

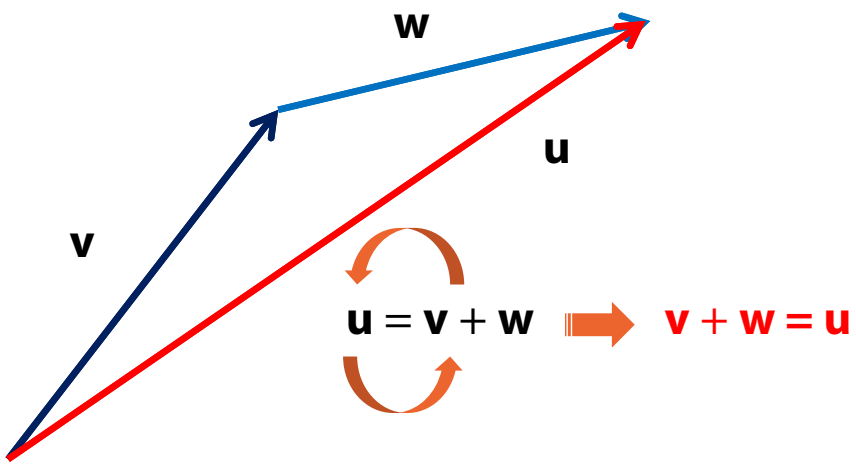
Recalling Some Basic Vector Algebra



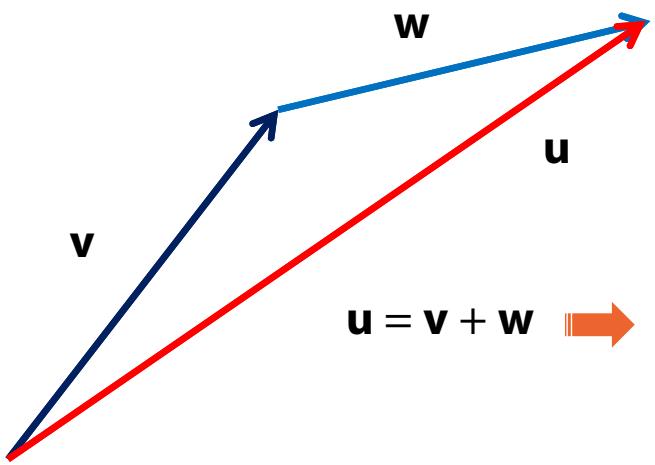
Adding Two Vectors



From Sum to Subtraction

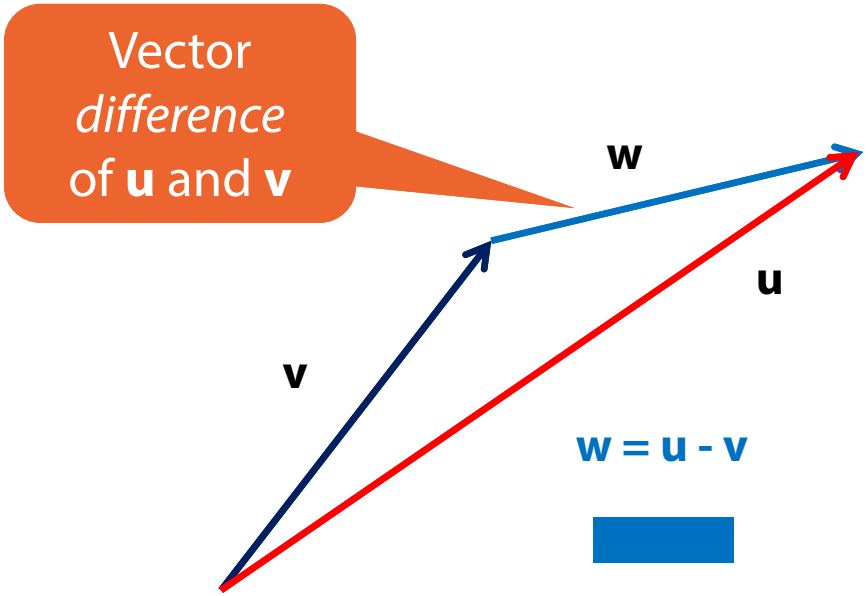


From Sum to Subtraction

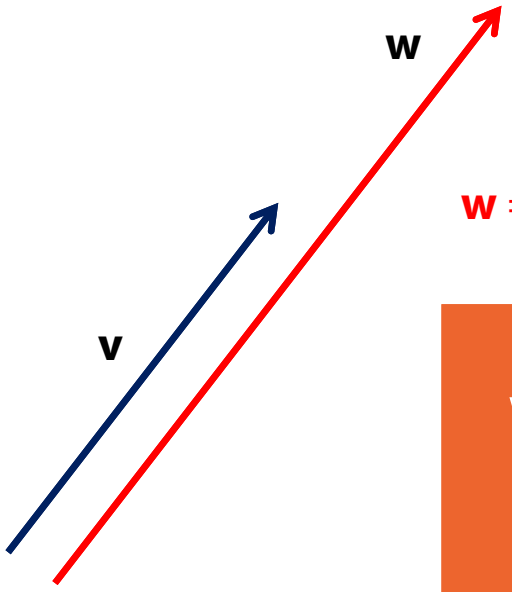


$u = v + w$ → $v + w = u$ → $w = u - v$

Subtracting Two Vectors



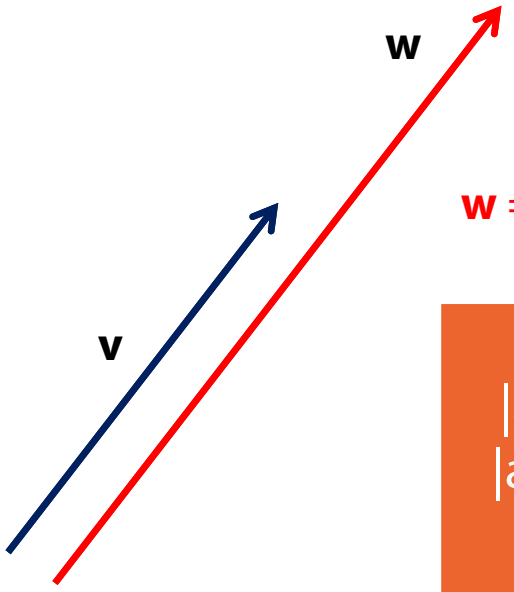
Multiplying a Vector by a Scalar



$\mathbf{w} = a\,\mathbf{v}$

w has the same direction of **v**

Multiplying a Vector by a Scalar

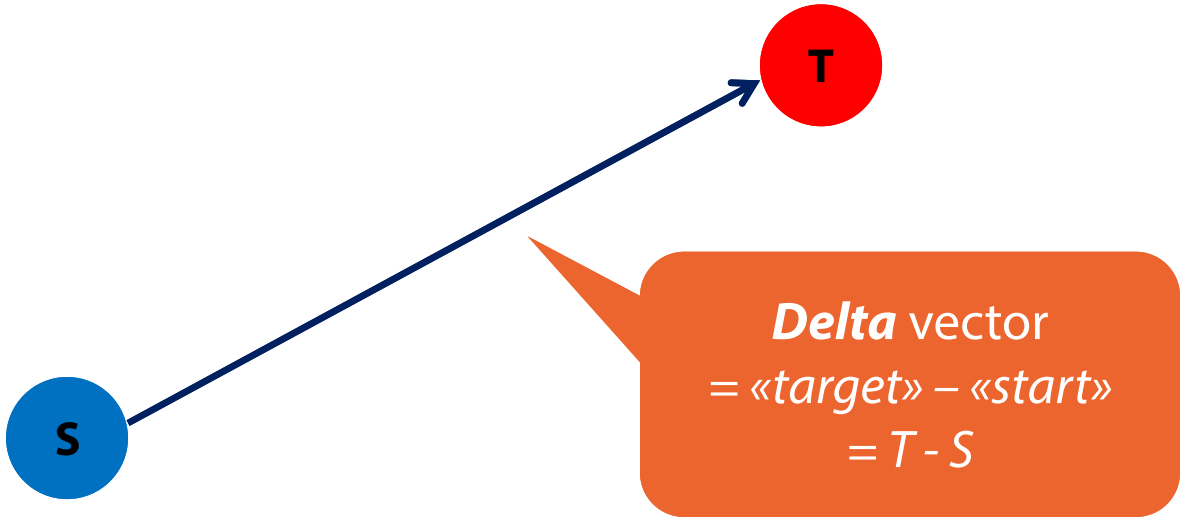


$\mathbf{w} = a\,\mathbf{v}$

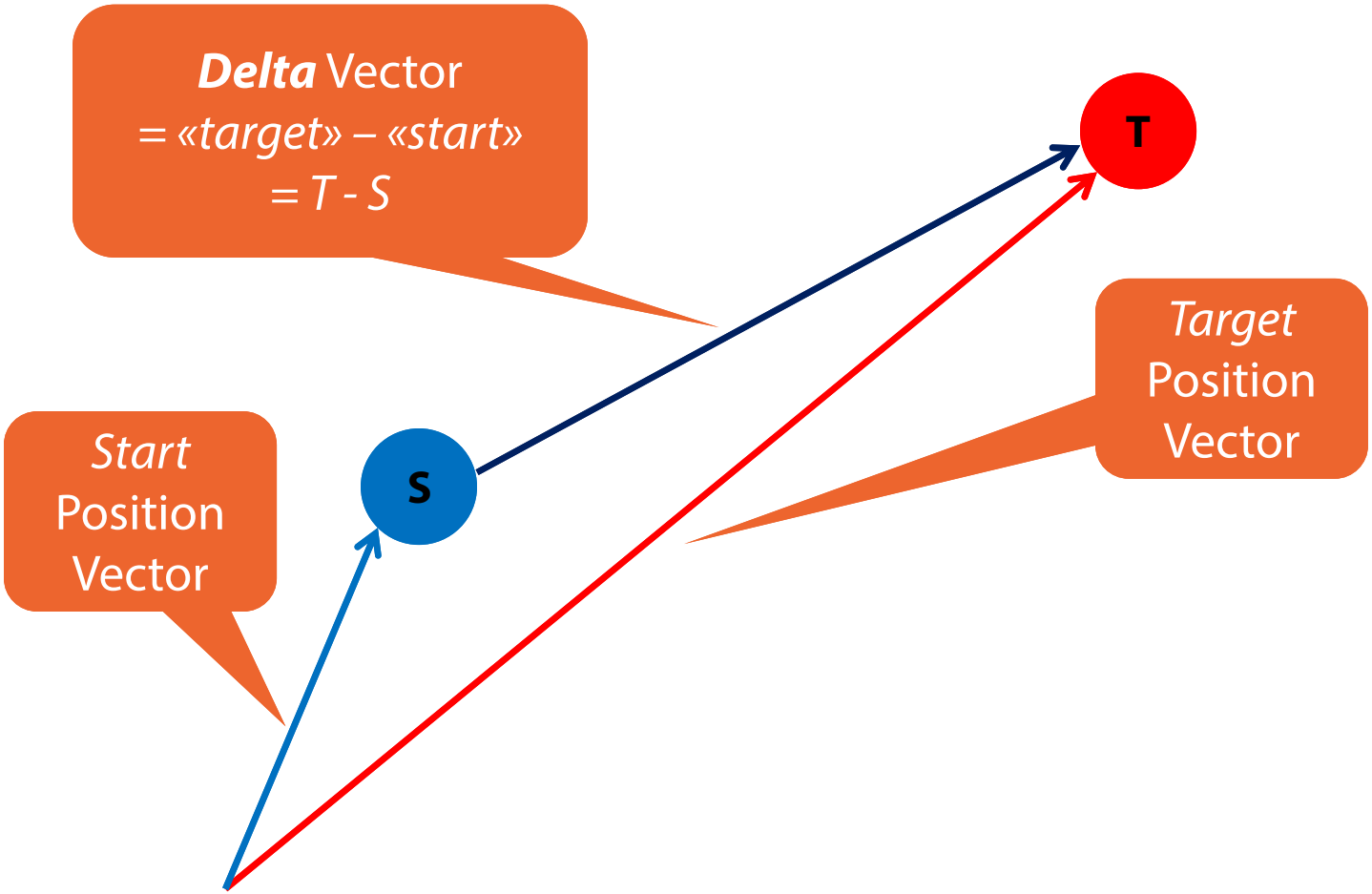
$|a| > 1$: $\mathbf{w}$ *longer* than $\mathbf{v}$
$|a| < 1$: $\mathbf{w}$ *shorter* than $\mathbf{v}$

Reaching a Target Position

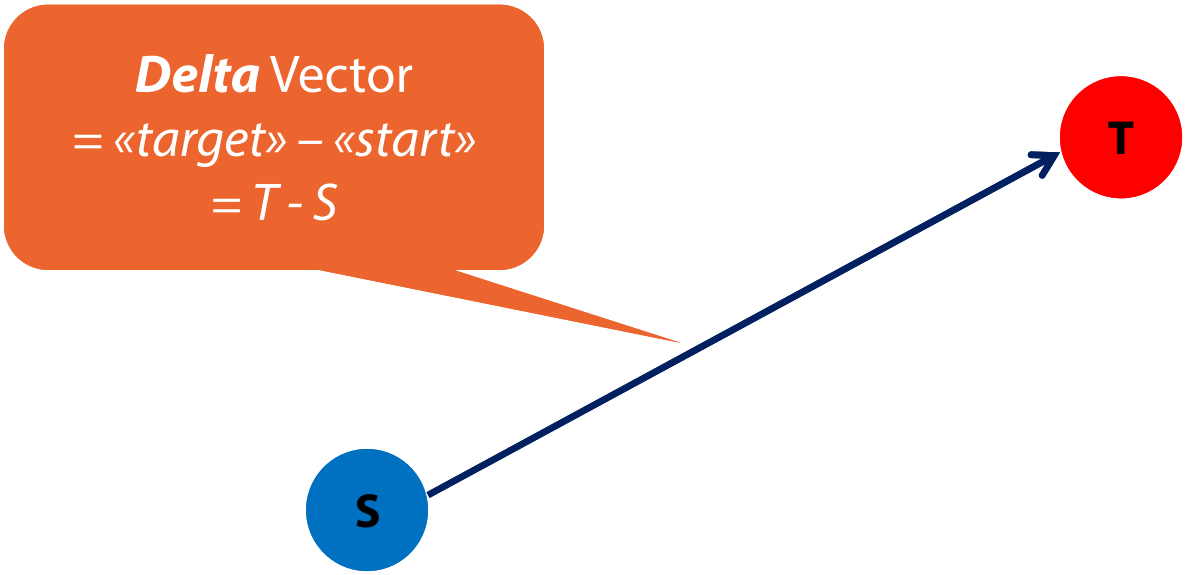
Reaching a Target Position



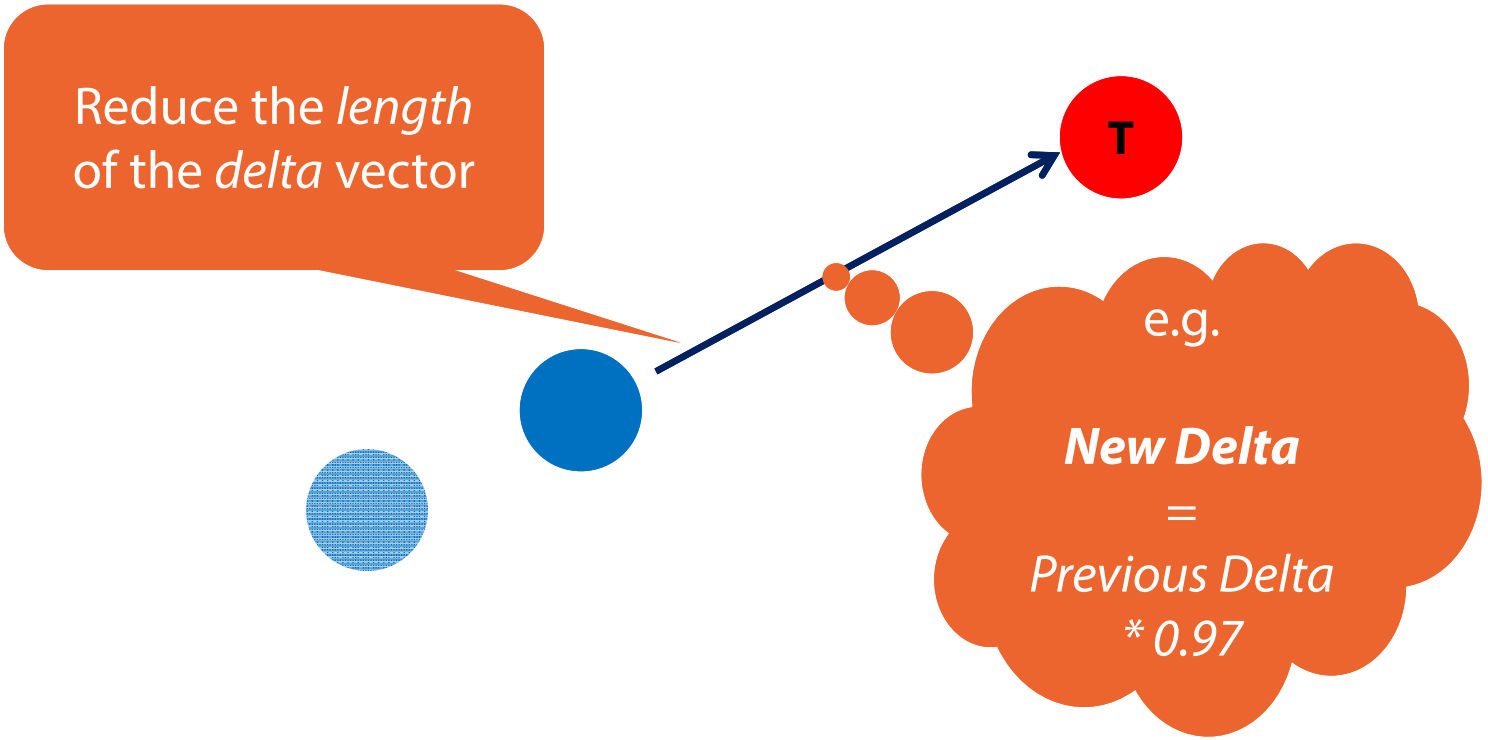
Reaching a Target Position



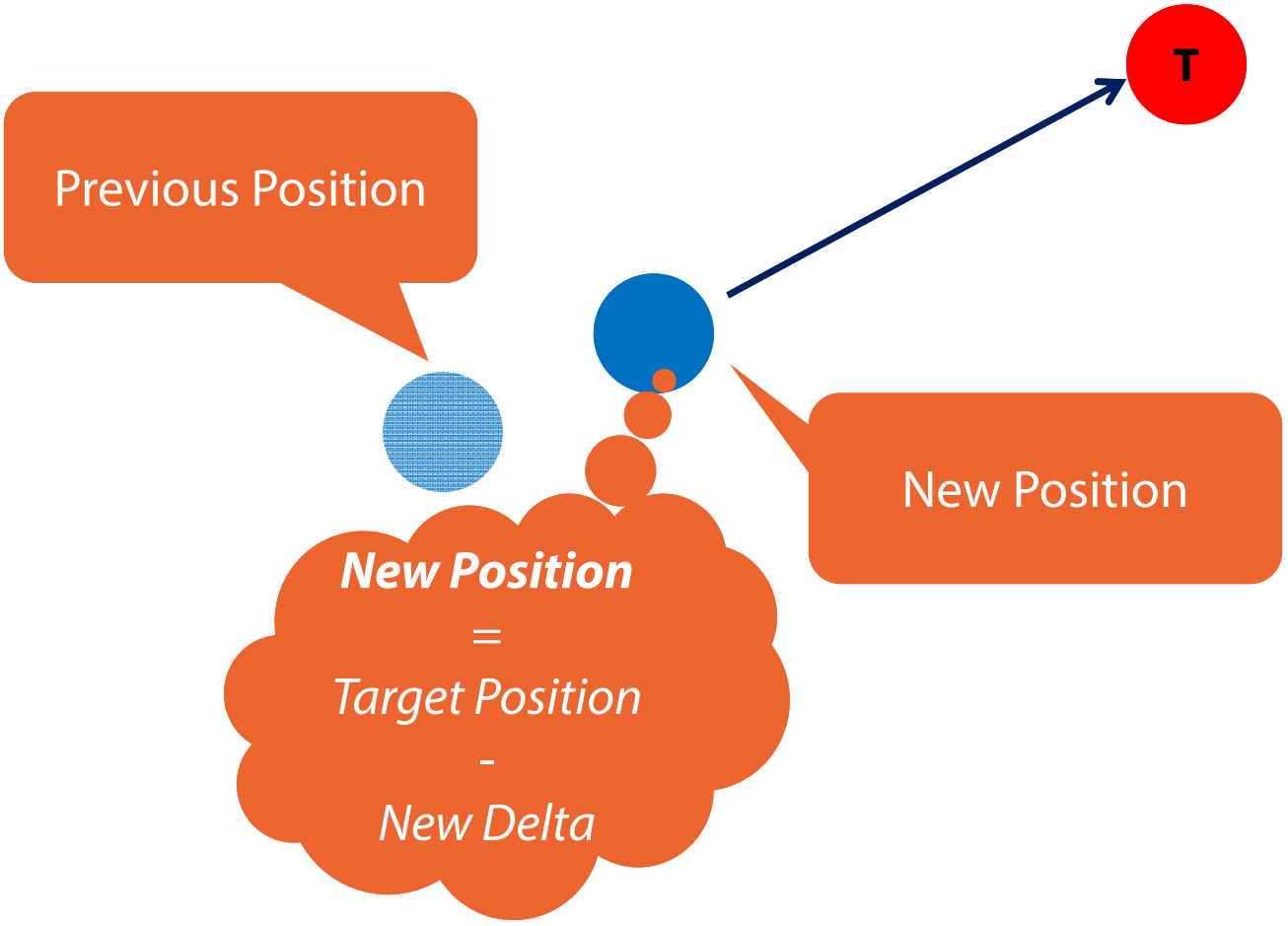
Reaching a Target Position



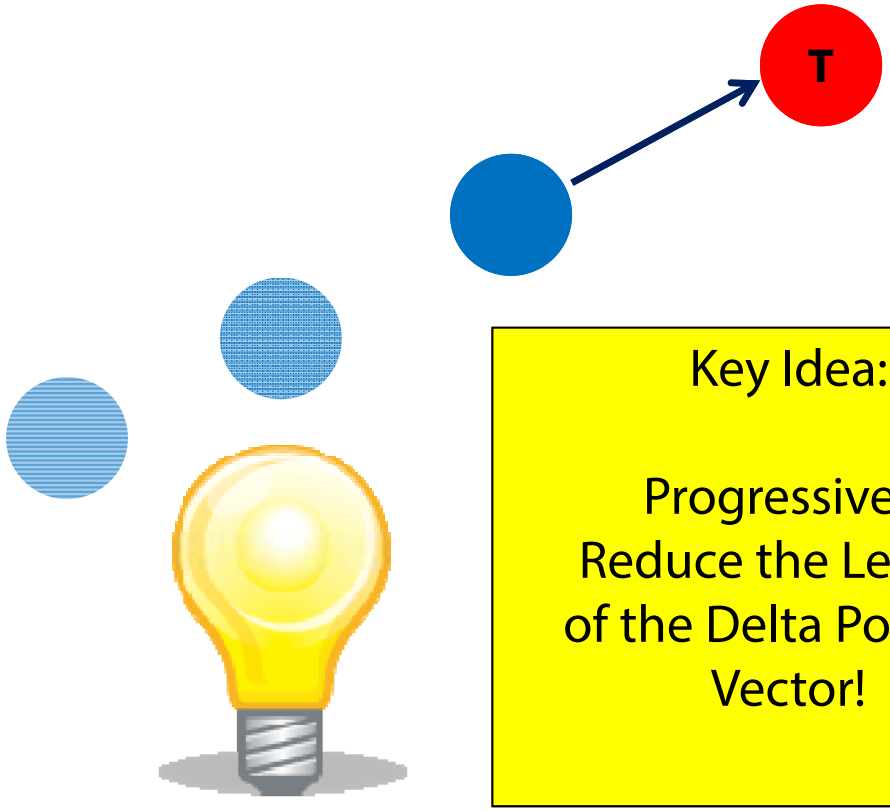
Reaching a Target Position



Reaching a Target Position



Reaching a Target Position



Key Idea:

Progressively
Reduce the Length
of the Delta Position
Vector!

Cinder's Vec2f Class

Vec2f

```
Vec2f  v, w;
Vec2f  u = v – w;
Vec2f  z = 0.97 * u;
```

…will do *The Right Thing* ™

Reaching a Target Position: C++ Code

```
// Data members e.g. in Cinder application C++ class
Vec2f mCurrentPos;
Vec2f mTargetPos;
```

Reaching a Target Position: C++ Code

e.g. inside the *update()* method

```
// 1. Calculate the delta position vector
Vec2f deltaPos = mTargetPos - mCurrentPos;
```

Reaching a Target Position: C++ Code

```
// 2. Progressively reduce
// the length of the delta position vector,
// multiplying the delta vector
// by a positive scalar less than 1
deltaPos *= 0.97f;
```

Reaching a Target Position: C++ Code

```
// 3. Recalculate current position
// (this will be progressively nearer
// to the target position)
mCurrentPos = mTargetPos - deltaPos;
```

Reaching a Target Position: C++ Code

NOTE:
Inside the *draw()* method,
just draw the object
at the *current* position!

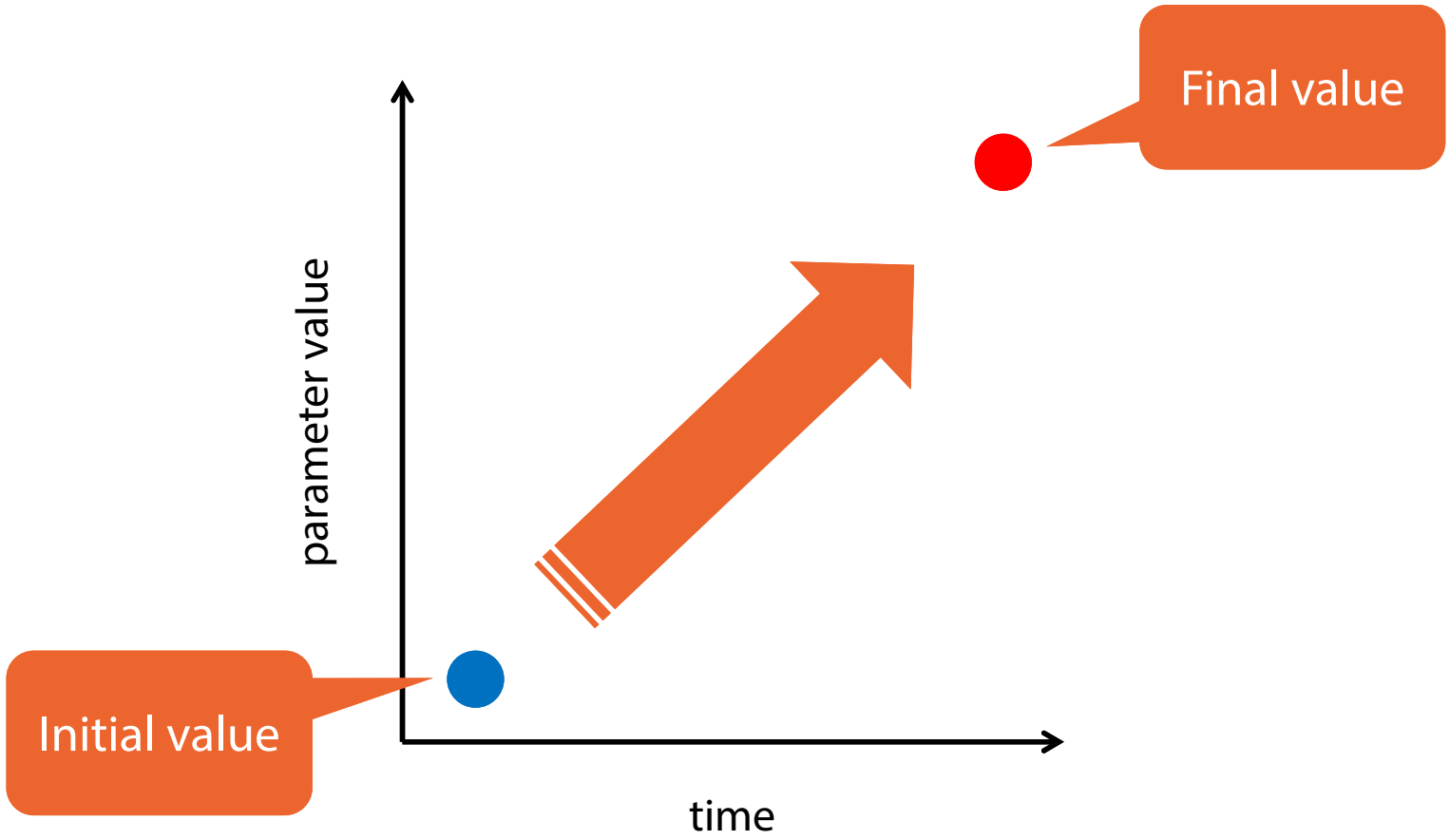
update() & draw()

Calculate
new positions
in *update()*

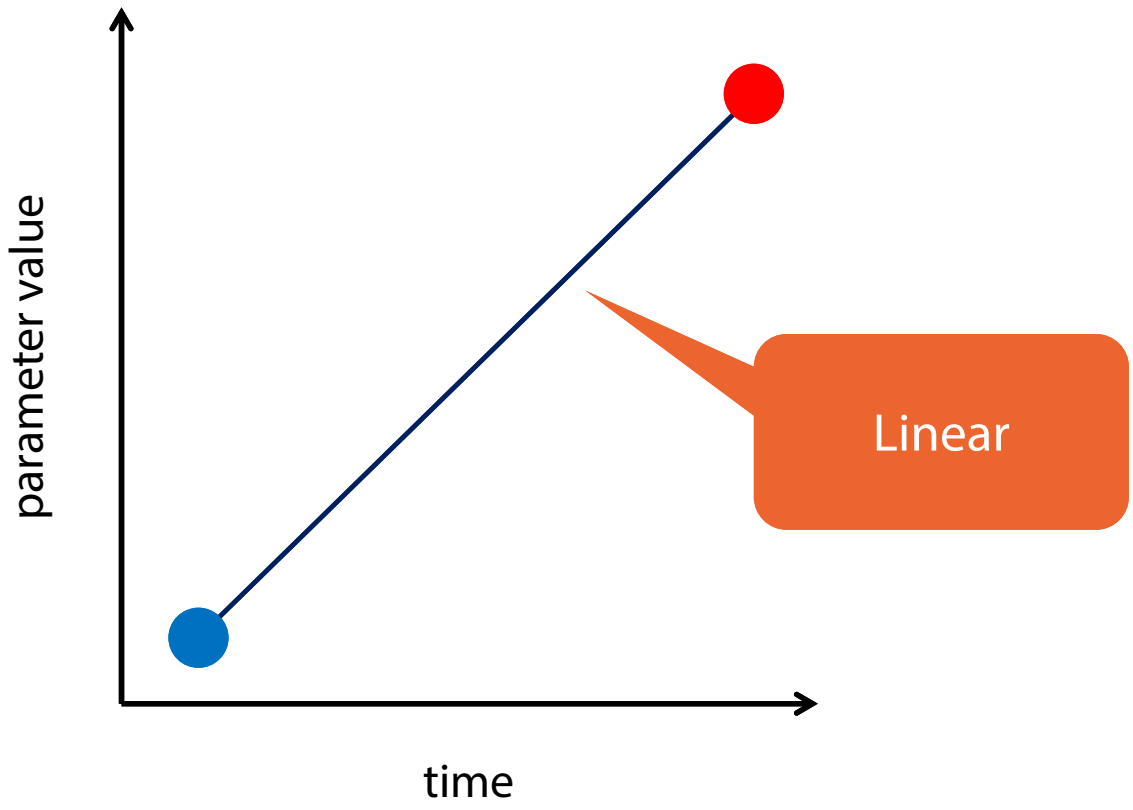
Paint each frame
in *draw()*
(using current positions)

Demo: Reaching a Target Position

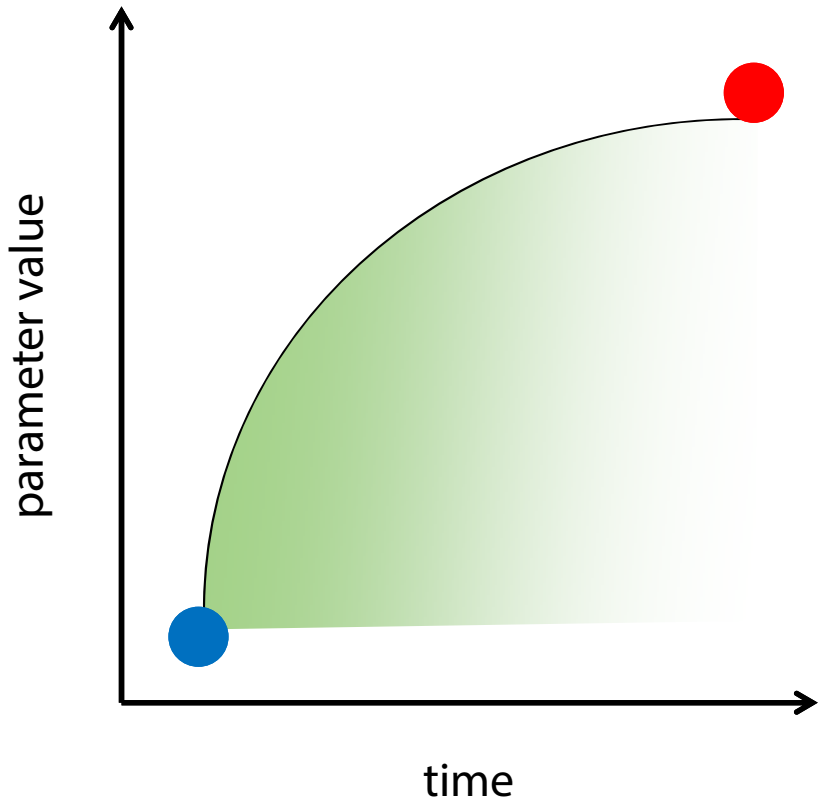
Easing Functions



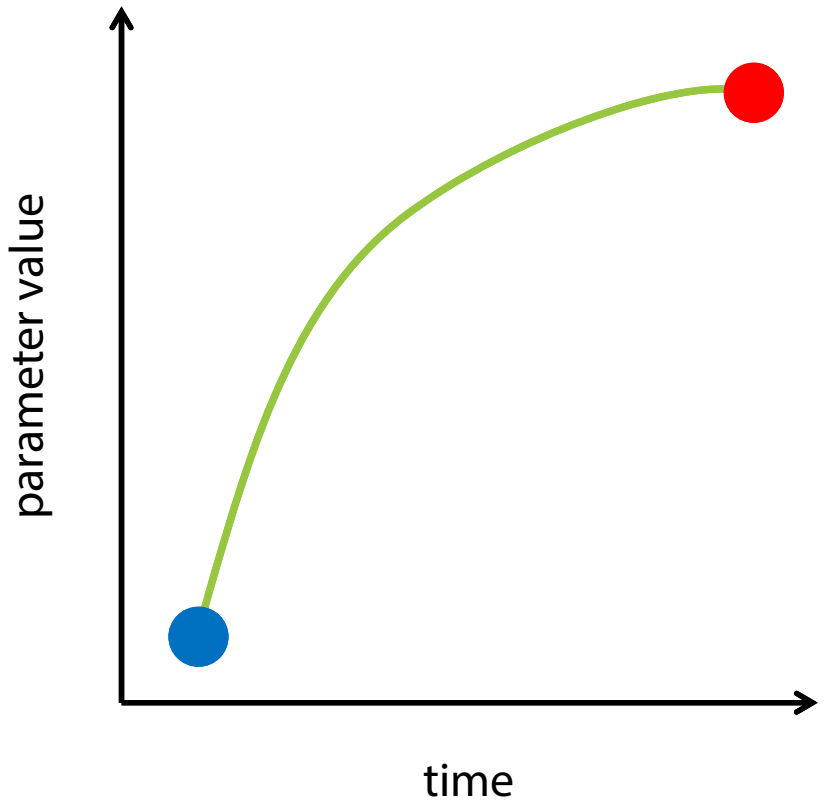
Easing Functions – Linear Evolution



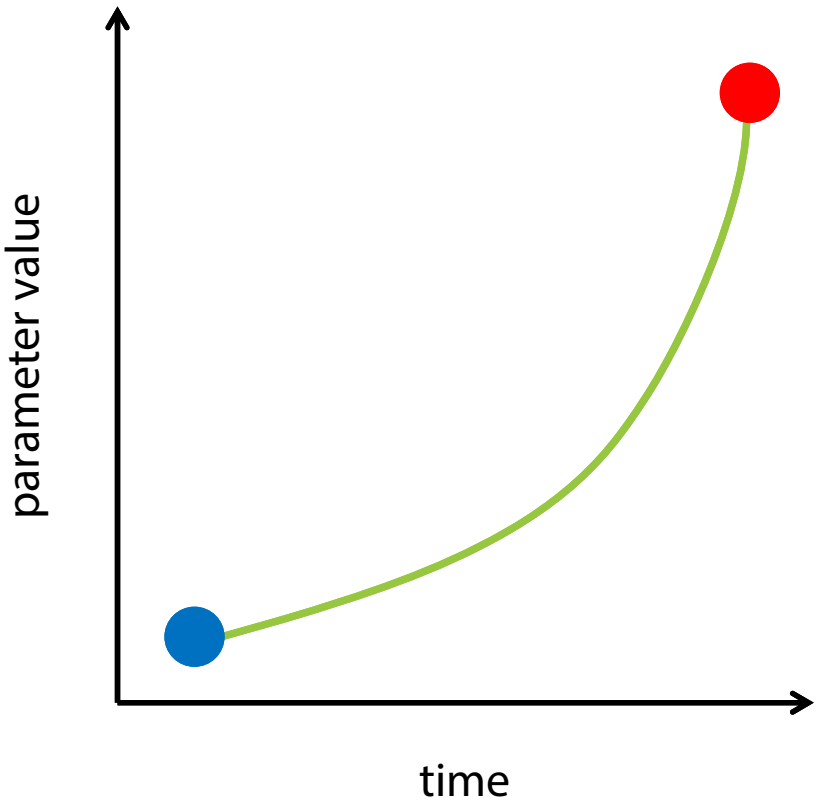
Easing Functions – Non-Linear Evolution



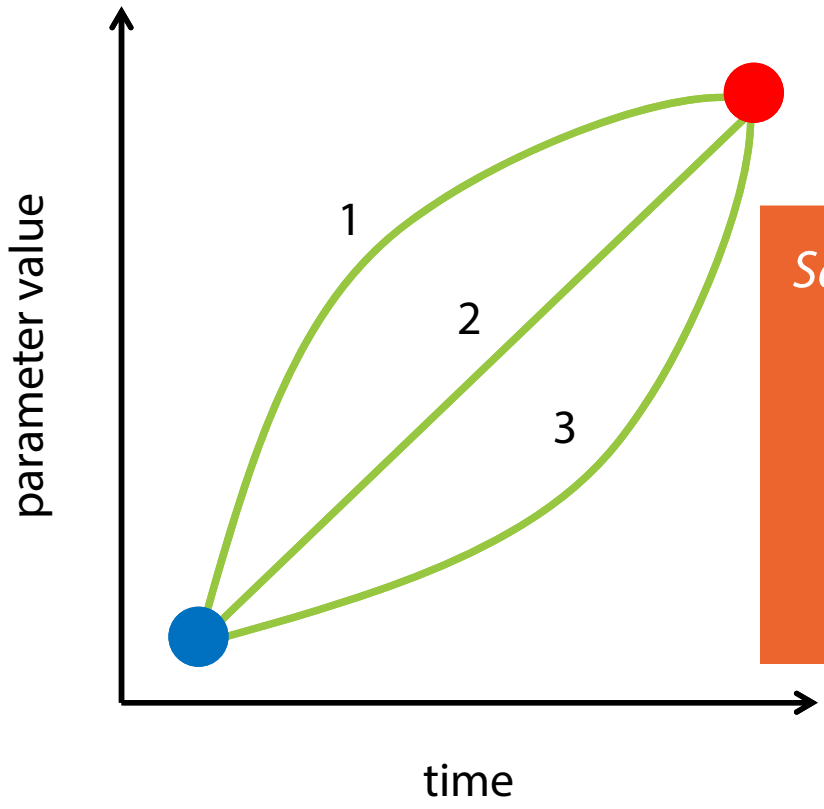
Easing Functions



Easing Functions



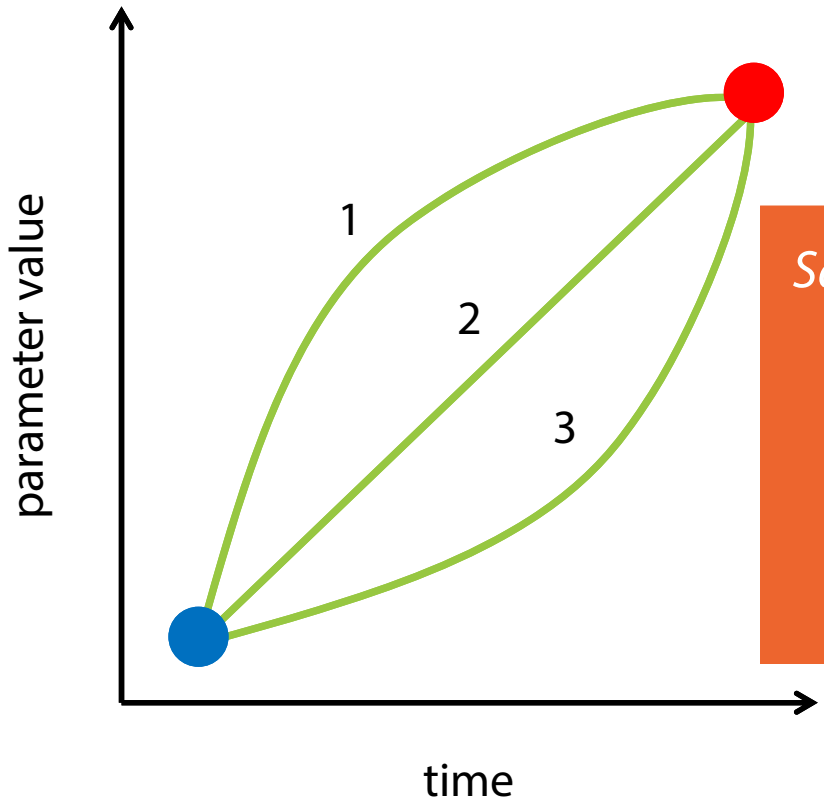
Easing Functions



Same initial and final values.

Different paths of evolution.

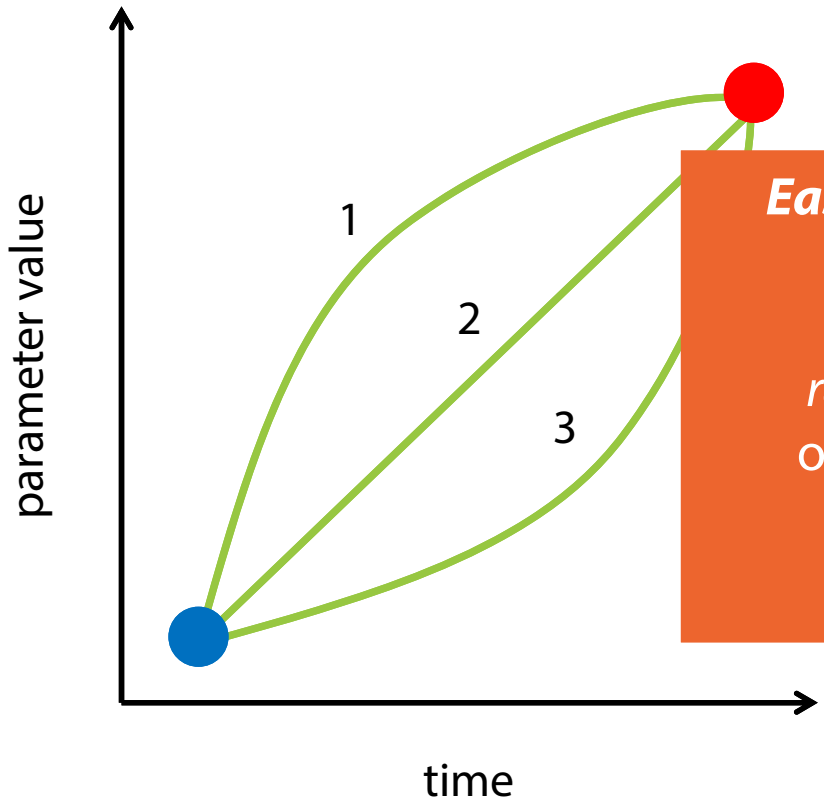
Easing Functions



Same initial and final values.

Different rates of change.

Easing Functions



Easing Functions

Specify the
rate of change
of a parameter
over time.

Cinder's Easing Functions

Demo: Cinder's EaseGallery

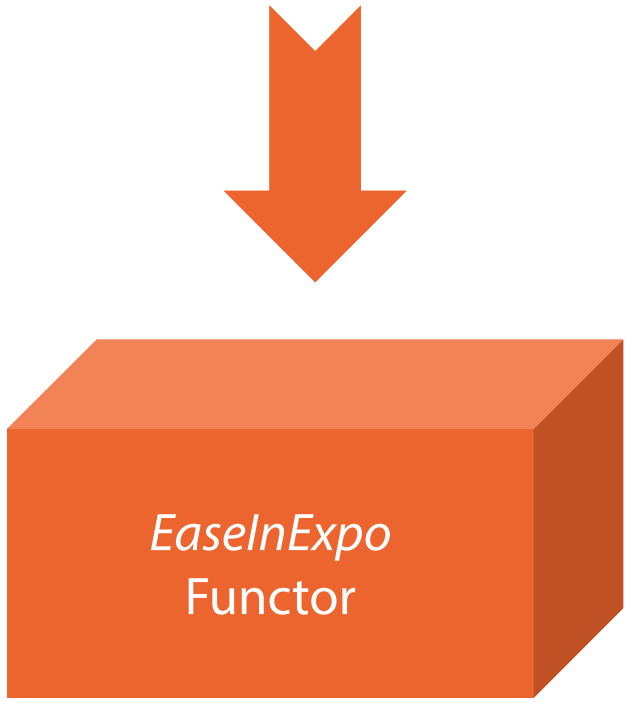
Cinder's Easing Functions

```
//
// Maps [0, 1] domain to another [0, 1] interval.
//
float easeSomething( float t );




float easeInExpo( float t );
```

Example

Easing Functor

```
float easeInExpo( float t );
```



C++ Recap: Functor

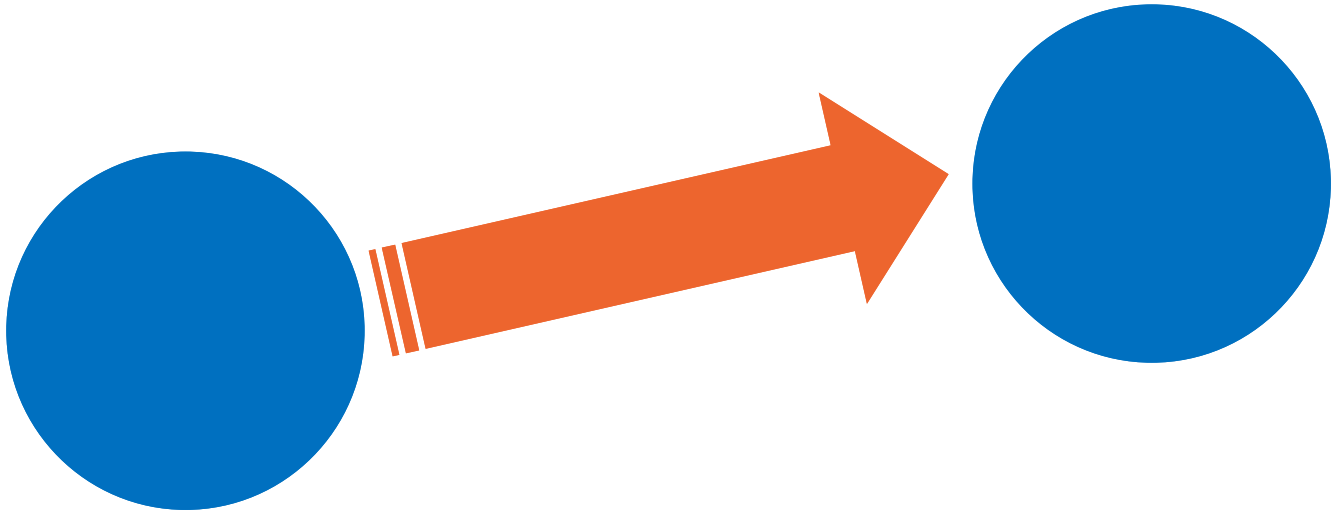
```
class SomeFunctor
{
public:
    SomeFunctor( /* May have some init parameters */ );

    float operator()( float t )
    {
       // Do something...
    }

    // ...
};
```

A ***functor*** is a C++ class that overloads *operator()*.

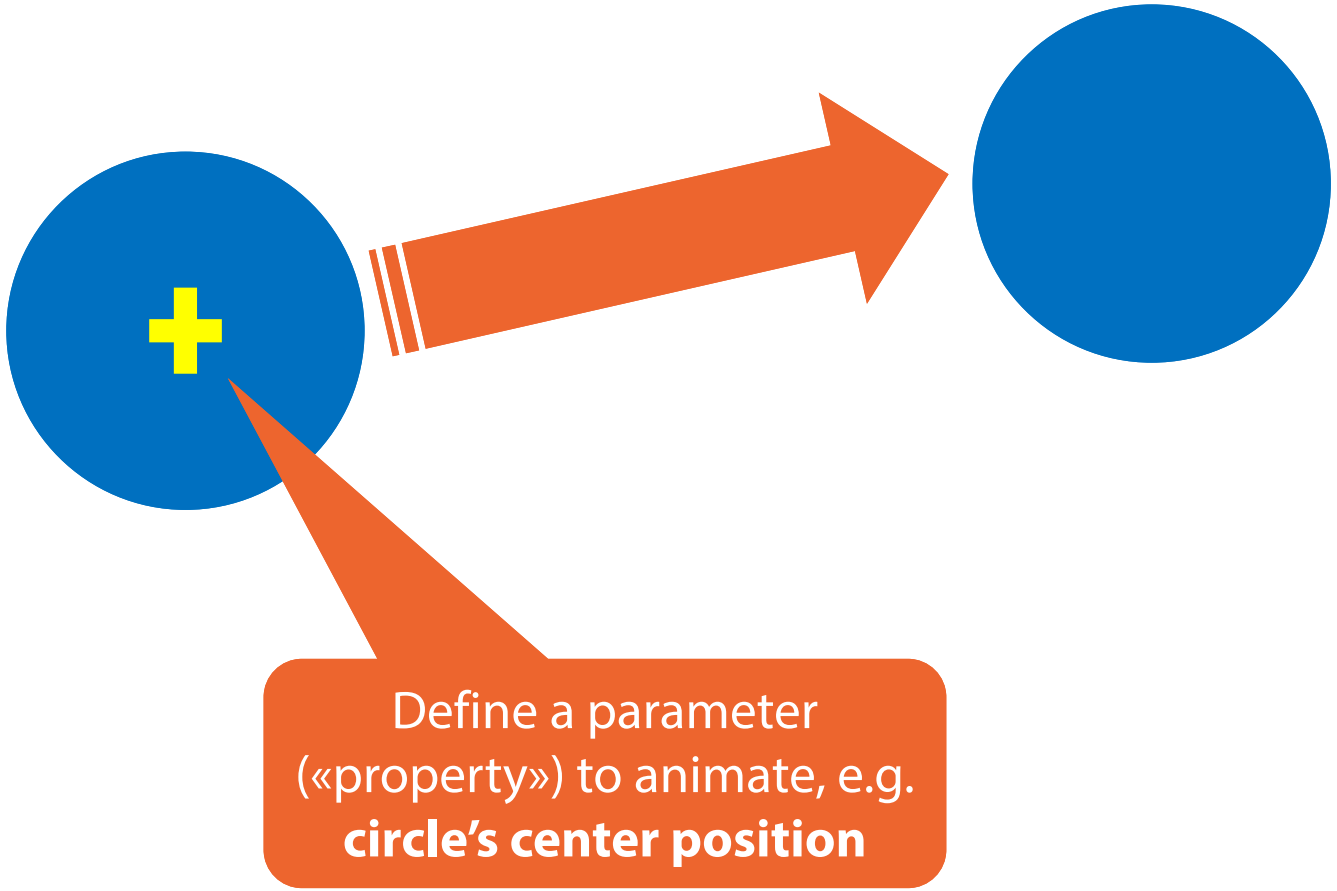
Demo: Animating With Easing Functions

Demo: Multiple Particles Animation

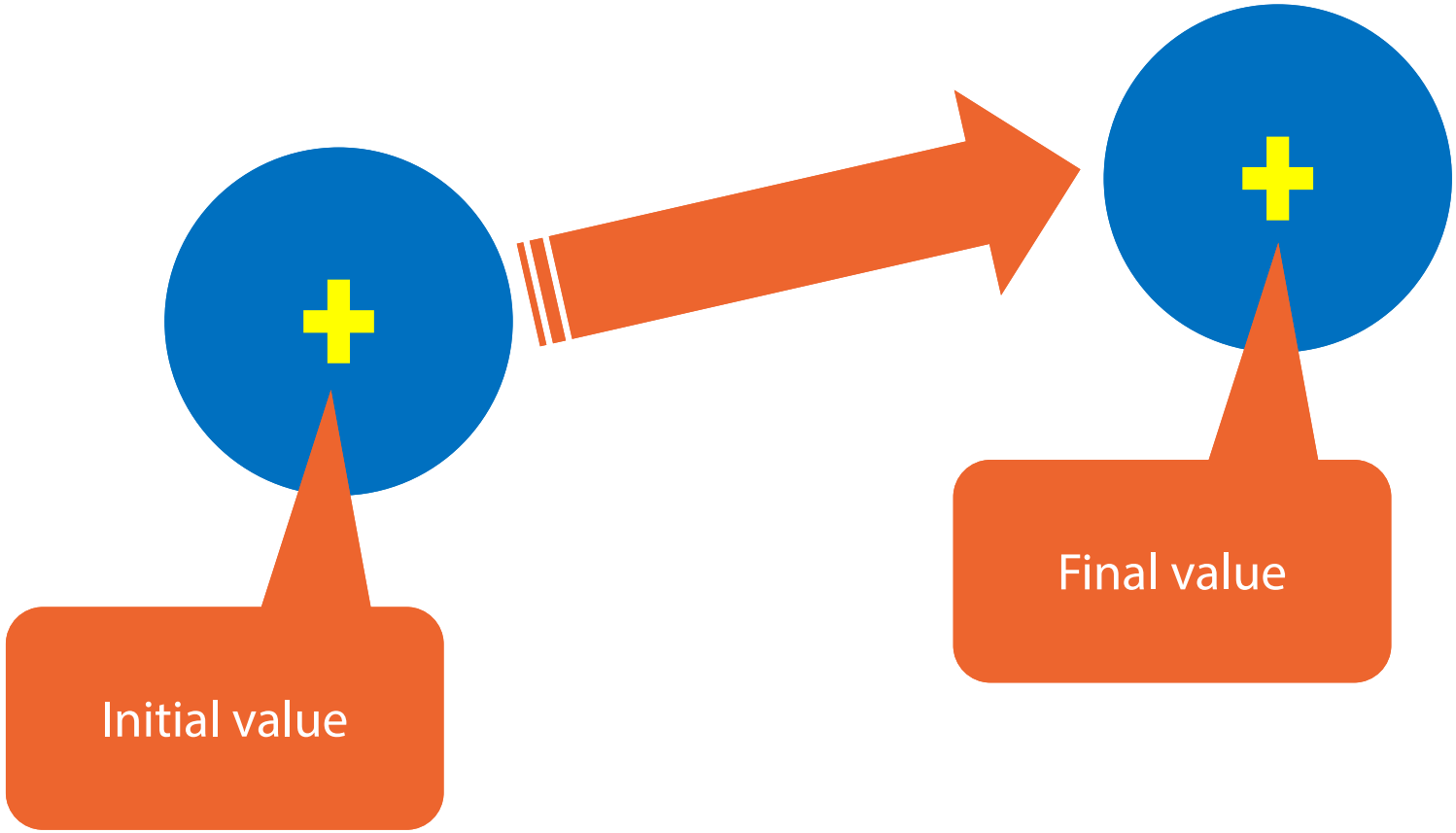
Timeline: A Brief Introduction



Timeline: A Brief Introduction



Timeline: A Brief Introduction



The Anim<T> Template

```
Anim<Vec2f> mCenterPos;
```

Type of the parameter to «animate»

Note:
T can be *float*, or *Color*, etc. !!

The Timeline's apply() Method

```
timeline().apply(
    &mCenterPos,    // Anim<T> parameter to animate
    mTargetPos,     // End value
    1.0f,           // Duration (in seconds)
    EaseInCubic()   // Easing Functor
);
```

Demo: Animating With the Timeline

Summary

- **Animation mechanics**
  - Show slightly different frames at >= 30 FPS
  - Division of labor in Cinder: draw() & update() methods
- **Moving from a starting position to a target position**
- **Easing functions (and functors)**
- **Timeline**
  - Anim<T> wrapper
  - Timeline's apply() method

Wrap Up