
the
gamedesigninitiative
at cornell university

Animation (2D)

Animation Basics: The FilmStrip



- Animation is a sequence of **hand-drawn frames**
 - Smoothly displays action when change quickly
 - Also called flipbook animation
- Arrange animation in a **sprite sheet** (one texture)
 - Software chooses which frame to use at any time
 - So programmer is actually the one doing animation

Anatomy of AnimationNode Class

```
/**
 * Sets the active frame as the given index.
 *
 * @param frame the index to make the active frame
 */
void AnimationNode::setFrame(int frame) {
    this->frame = frame;
    int x = (frame % cols)*bounds.size.width;
    int y = (frame / cols)*bounds.size.height;
    bounds.origin.set(x,y);
    setPolygon(bounds);
}
```

Anatomy of AnimationNode Class

```
/**
 * Sets the active frame as the given index.
 *
 * @param frame the index to make the active frame
 */
void AnimationNode::setFrame(int frame) {
    this->frame = frame;
    int x = (frame % cols)*bounds.size.width;
    int y = (frame / cols)*bounds.size.height;
    bounds.origin.set(x,y);
    setPolygon(bounds);
}
```

Actual code has
some minor
optimizations

Adjusting your Speed

- Do not want to go too fast
 - 1 animation frame = 16 ms
 - Walk cycle = 8/12 frames
 - Completed in 133-200 ms
- General solution: *cooldowns*
 - Add int timer to your object
 - Go to next frame when = 0
 - Reset it to > 0 at new frame
- Simple but tedious
 - Have to do for each object
 - **And** for each animation



Matching Your Translation

- Movement is *two* things
 - **Animation** of the filmstrip
 - **Translation** of the image
 - These two must align
- **Example:** Walking
 - Foot is point of contact
 - “Stays in place” as move
 - This constrains translation
- Make movement regular
 - Measure distance per frame
 - Keep same across frames



Matching Your Translation

- Movement is *two* things
 - **Animation** of the filmstrip
 - **Translation** of the image
 - These two must align
- **Example:** Walking
 - Foot is point of contact
 - “Stays in place” as move
 - This constrains translation
- Make movement regular
 - Measure distance per frame
 - Keep same across frames



Matching Your Translation

- Movement is *two* things
 - **Animation** of the filmstrip
 - **Translation** of the image
 - These two must align
- **Example:** Walking
 - Foot is point of contact
 - “Stays in place” as move
 - This constrains translation
- Make movement regular
 - Measure distance per frame
 - Keep same across frames



Point of
contact

Distance
forward

Combining Animations

Landing Animation

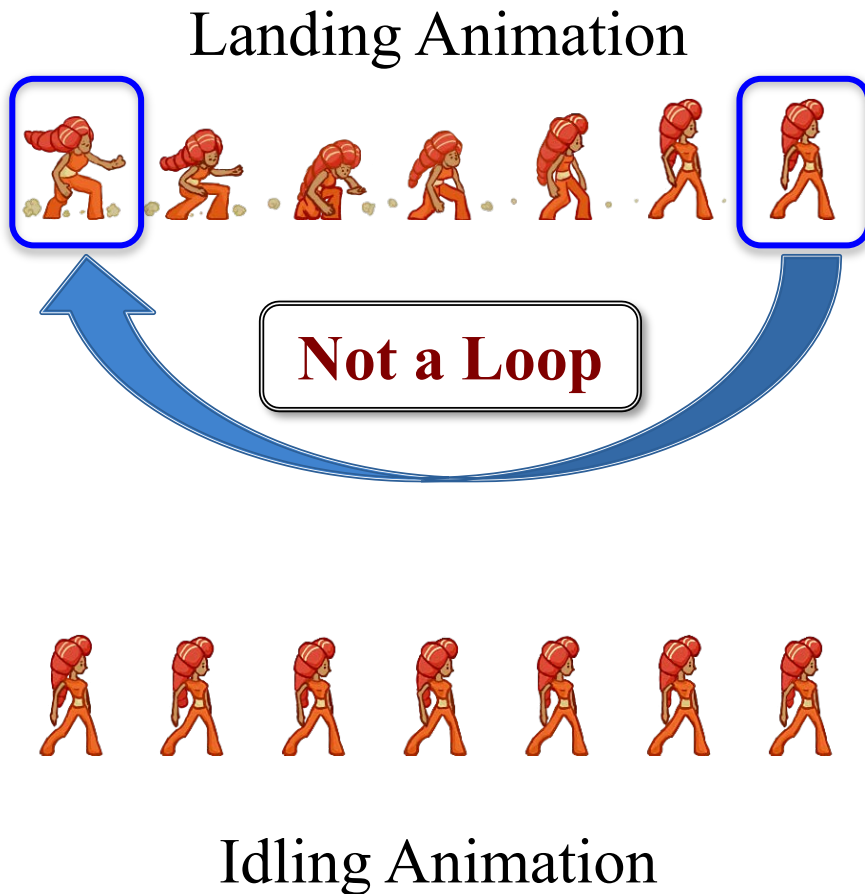


- Characters to a lot of things
 - Run, jump, duck, slide
 - Fire weapons, cast spells
 - Fidget while player AFK
- Want animations for all
 - Is loop appropriate for each?
 - How do we transition?
- **Idea:** shared boundaries
 - End of loop = start of another
 - Treat like advancing a frame



Idling Animation

Combining Animations



- Characters to a lot of things
 - Run, jump, duck, slide
 - Fire weapons, cast spells
 - Fidget while player AFK
- Want animations for all
 - Is loop appropriate for each?
 - How do we transition?
- **Idea:** shared boundaries
 - End of loop = start of another
 - Treat like advancing a frame

Combining Animations

Landing Animation



Transition



Idling Animation

- Characters to a lot of things
 - Run, jump, duck, slide
 - Fire weapons, cast spells
 - Fidget while player AFK
- Want animations for all
 - Is loop appropriate for each?
 - How do we transition?
- **Idea:** shared boundaries
 - End of loop = start of another
 - Treat like advancing a frame

Combining Animations

Landing Animation



Transition

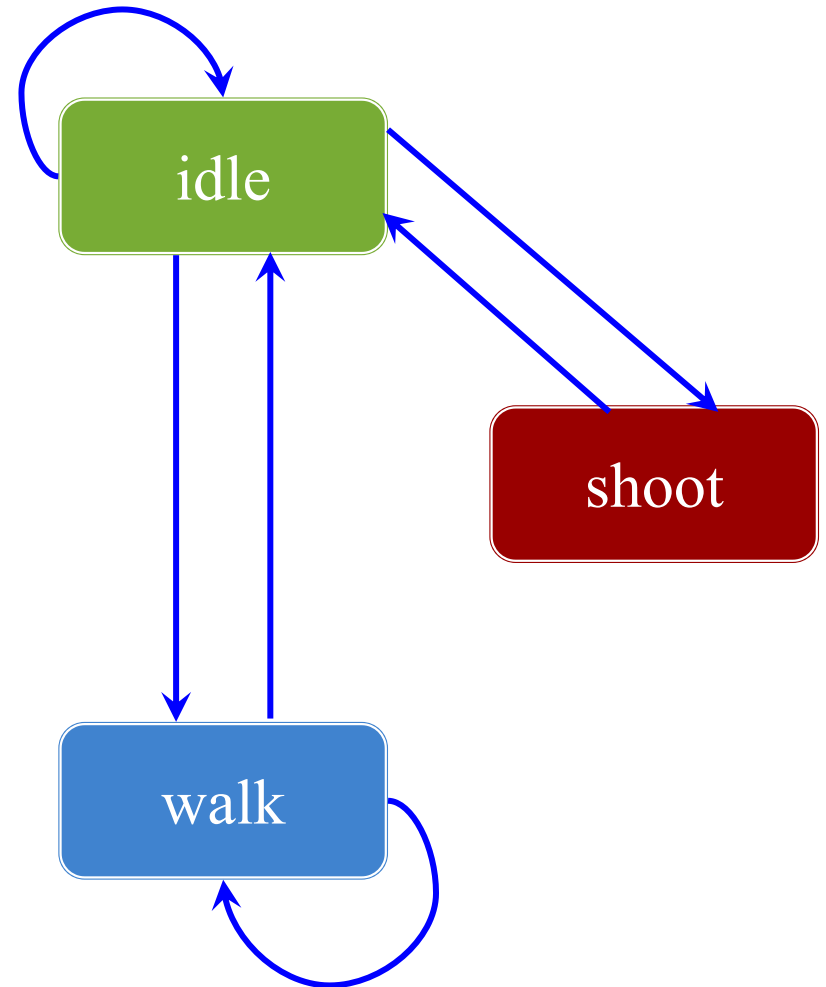


Idling Animation

- Characters do a lot of things
 - Run, jump, duck, slide
 - Fire weapons, cast spells
 - Fidget while player AFK
- Want animations for all
 - Is loop appropriate for each?
 - How do we transition?
- **Idea:** shared boundaries
 - **But do not draw ends twice!**

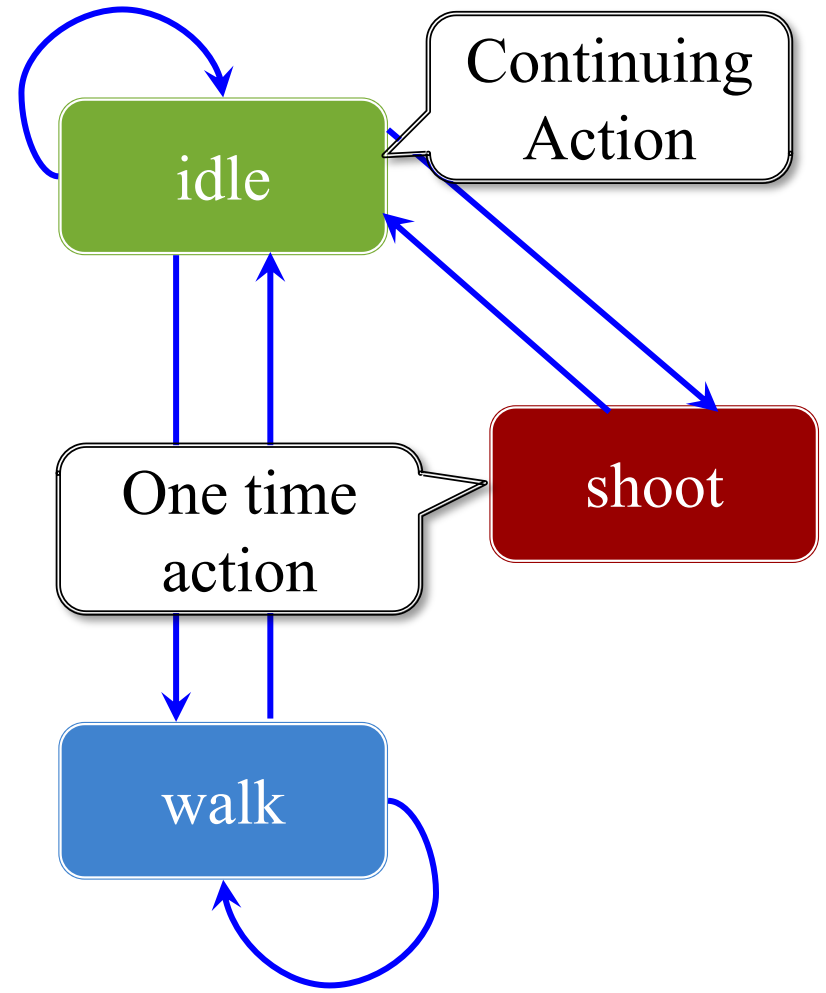
Animation and State Machines

- Idea: Each sequence a state
 - Do sequence while in state
 - Transition when at end
 - Only loop if loop in graph
- A graph edge means...
 - Boundaries match up
 - Transition is allowable
- Similar to data driven AI
 - Created by the designer
 - Implemented by programmer
 - Modern engines have tools

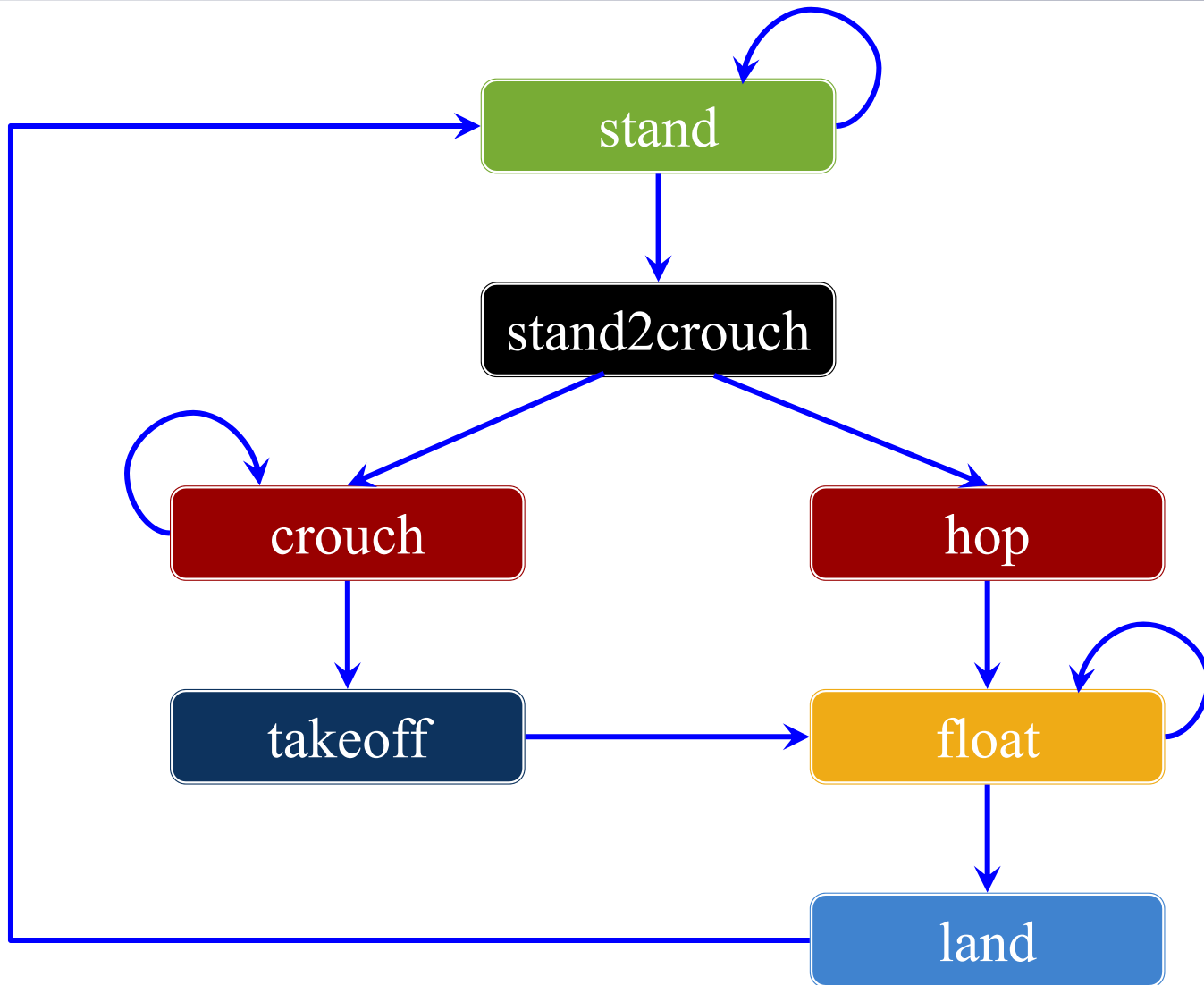


Animation and State Machines

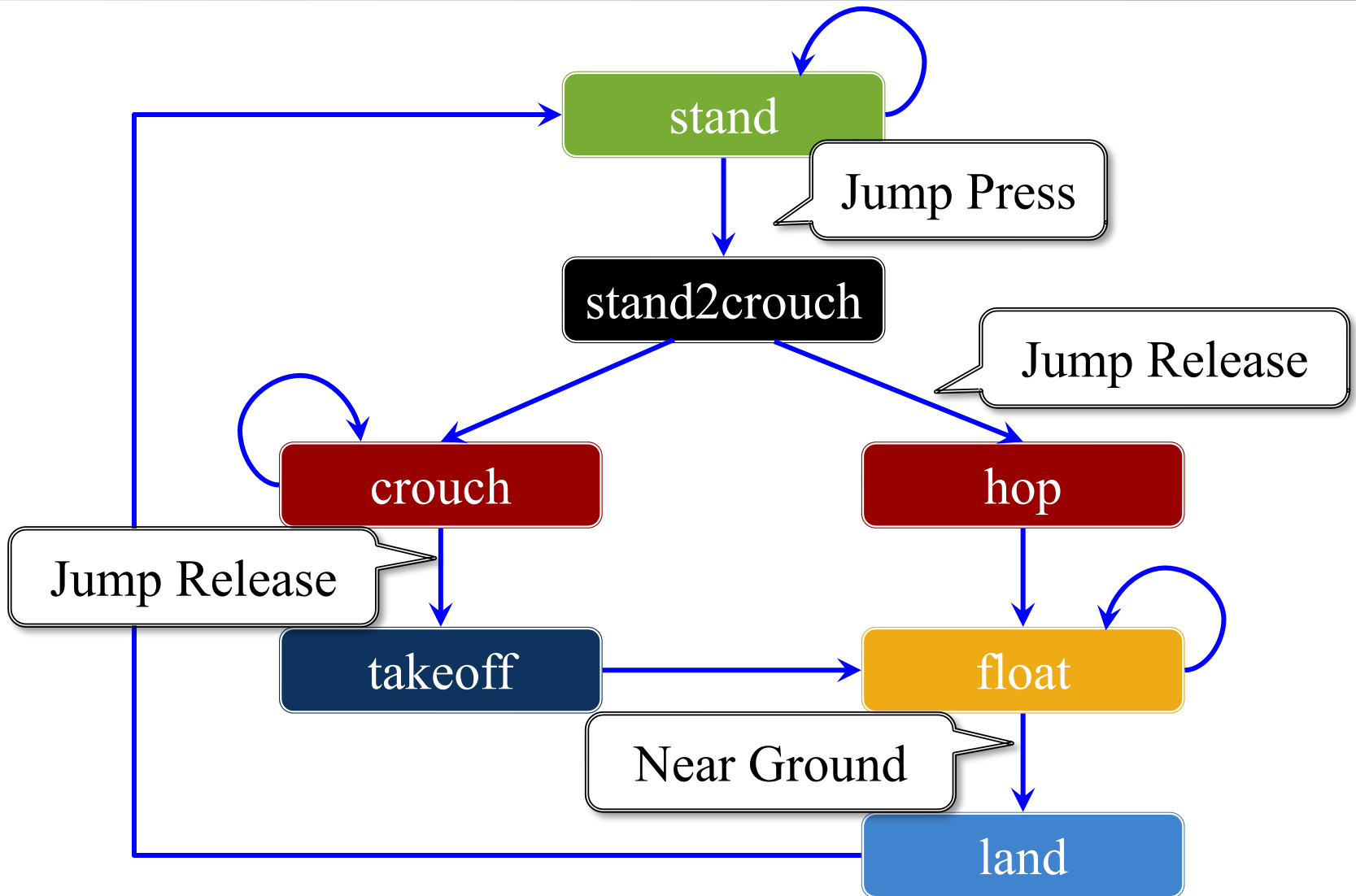
- Idea: Each sequence a state
 - Do sequence while in state
 - Transition when at end
 - Only loop if loop in graph
- A graph edge means...
 - Boundaries match up
 - Transition is allowable
- Similar to data driven AI
 - Created by the designer
 - Implemented by programmer
 - Modern engines have tools



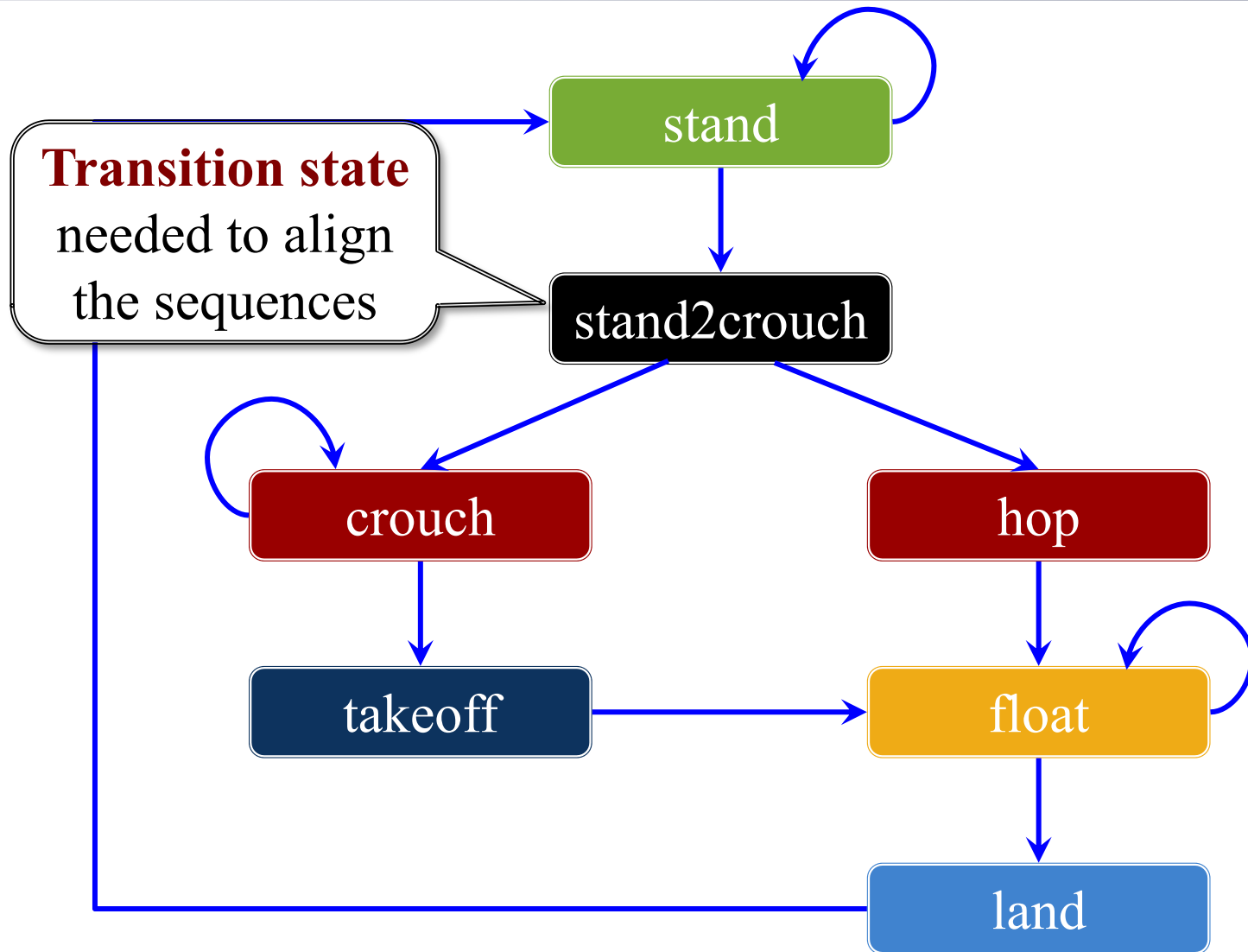
Complex Example: Jumping



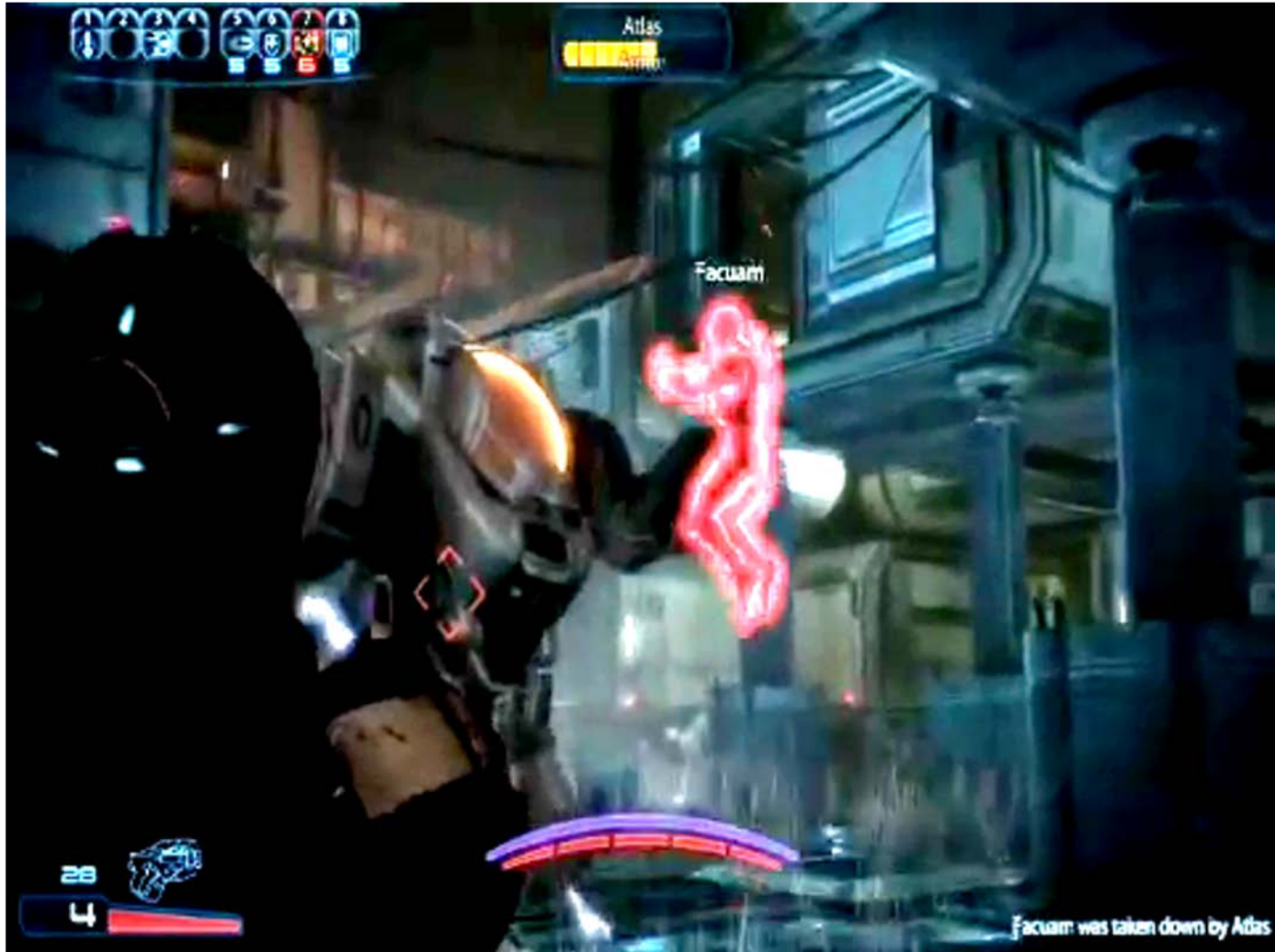
Complex Example: Jumping



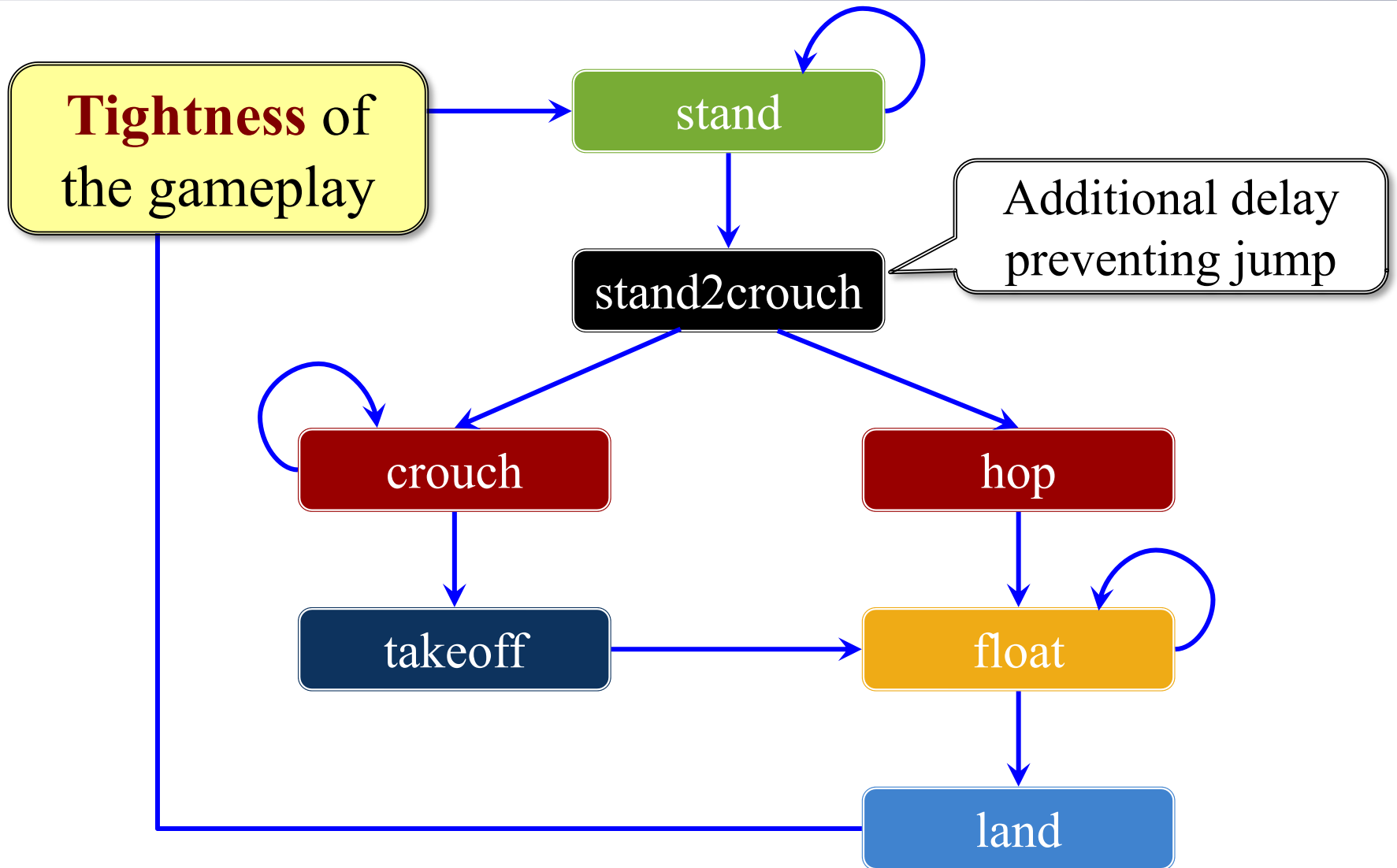
Complex Example: Jumping



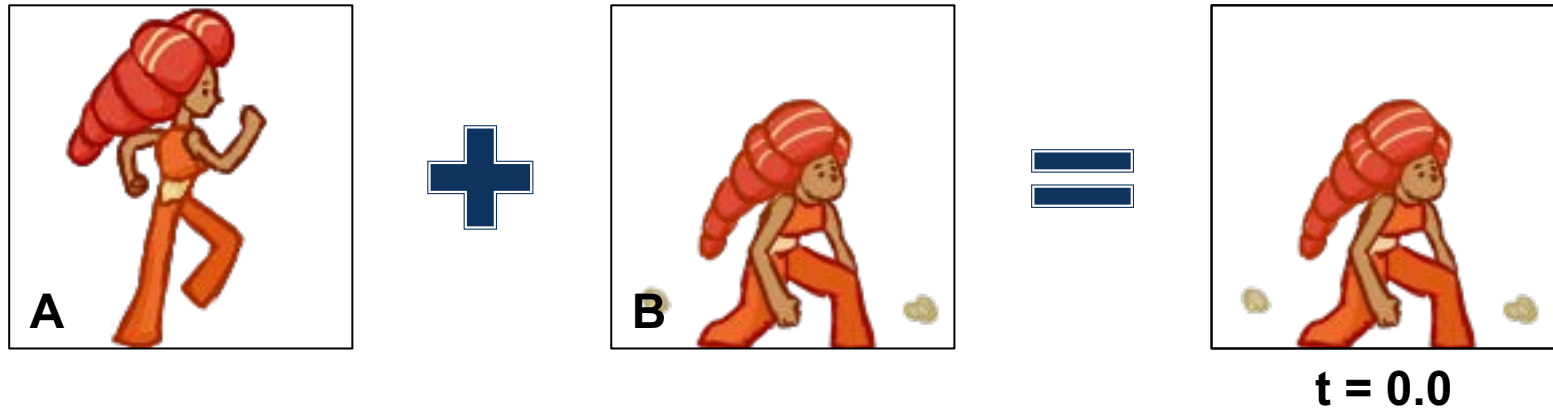
Aside: Sync Kills



The Responsiveness Issue



Fast Transitions: Crossfade Blending



- Linear interpolation on colors

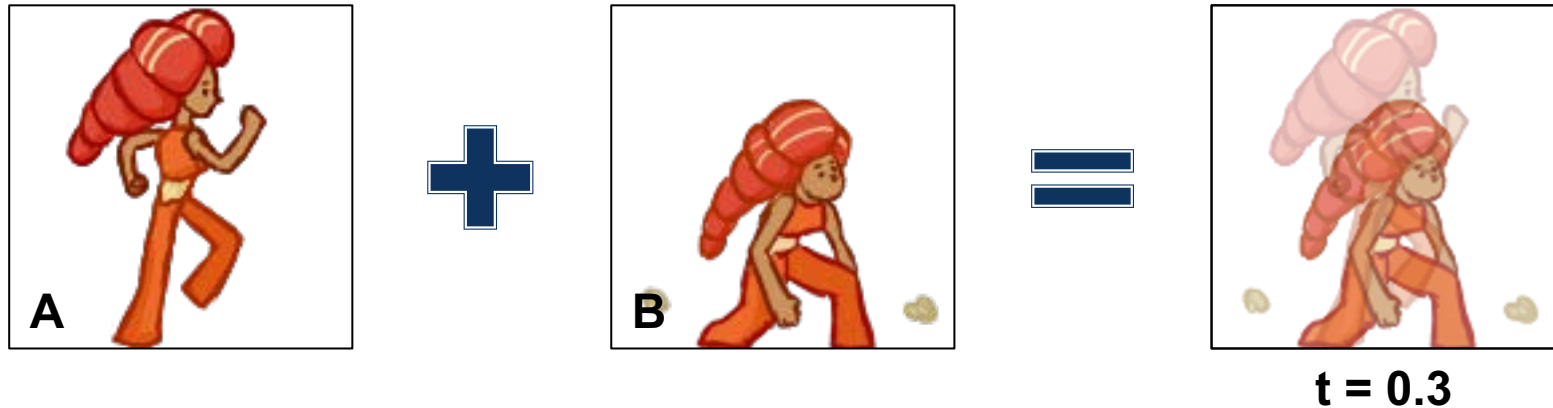
$$r_c = tr_a + (1 - t)r_b$$

$$g_c = tg_a + (1 - t)g_b$$

$$b_c = tb_a + (1 - t)b_b$$

Note weights sum to 1.0

Fast Transitions: Crossfade Blending



- Linear interpolation on colors

$$r_c = tr_a + (1 - t)r_b$$

$$g_c = tg_a + (1 - t)g_b$$

$$b_c = tb_a + (1 - t)b_b$$

Note weights sum to 1.0

Fast Transitions: Crossfade Blending



- Linear interpolation on colors

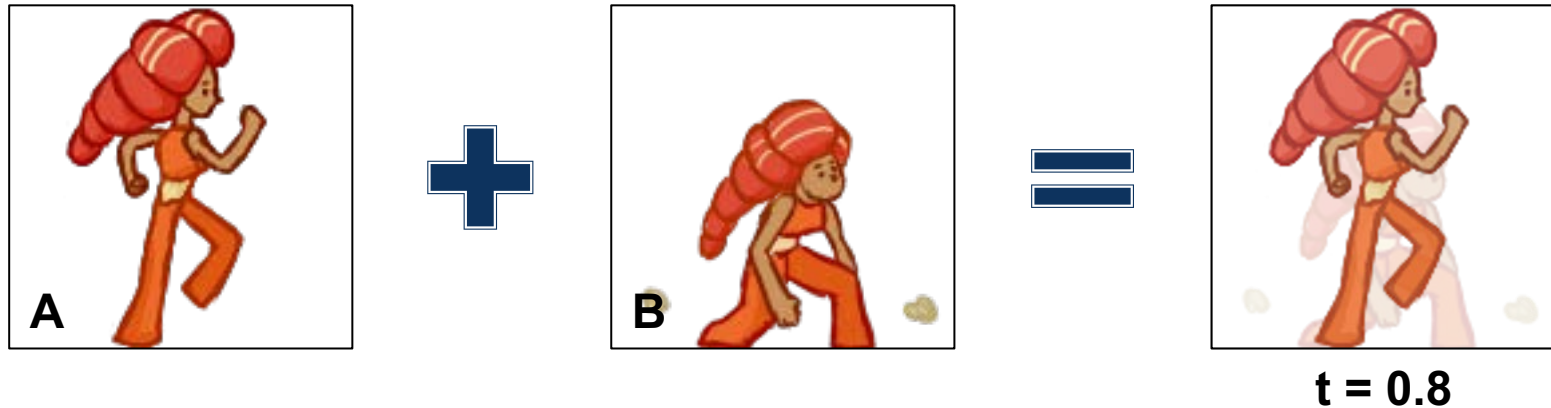
$$r_c = tr_a + (1 - t)r_b$$

$$g_c = tg_a + (1 - t)g_b$$

$$b_c = tb_a + (1 - t)b_b$$

Note weights sum to 1.0

Fast Transitions: Crossfade Blending



- Linear interpolation on colors

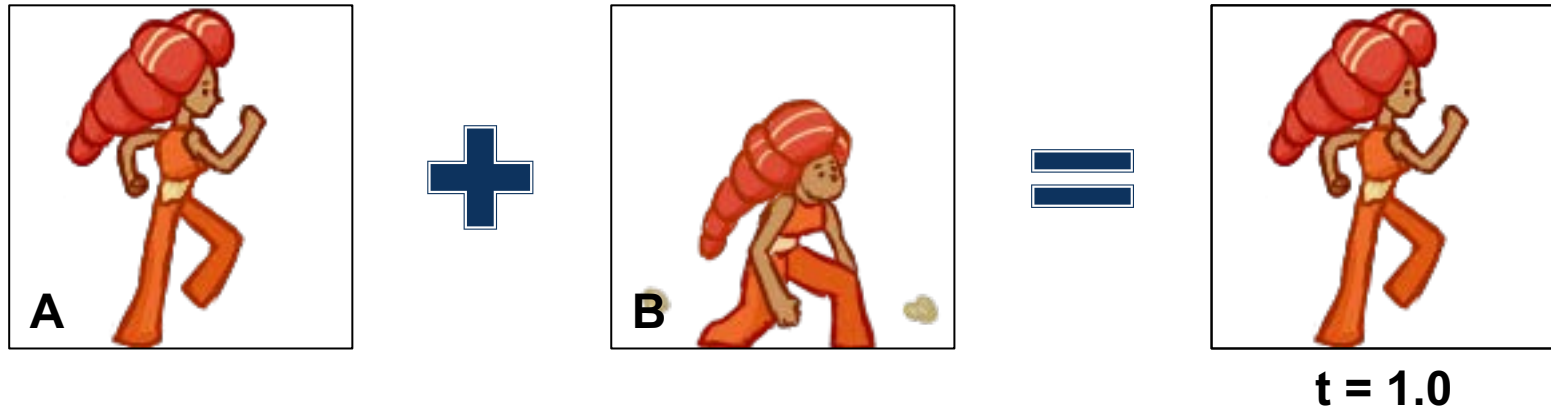
$$r_c = tr_a + (1 - t)r_b$$

$$g_c = tg_a + (1 - t)g_b$$

$$b_c = tb_a + (1 - t)b_b$$

Note weights sum to 1.0

Fast Transitions: Crossfade Blending



- Linear interpolation on colors

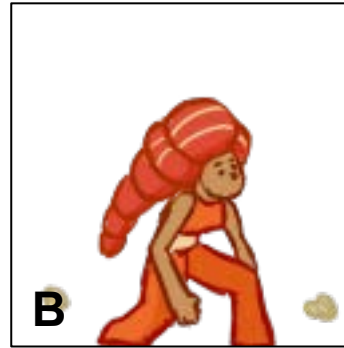
$$r_c = tr_a + (1 - t)r_b$$

$$g_c = tg_a + (1 - t)g_b$$

$$b_c = tb_a + (1 - t)b_b$$

Note weights sum to 1.0

Combining With Animation

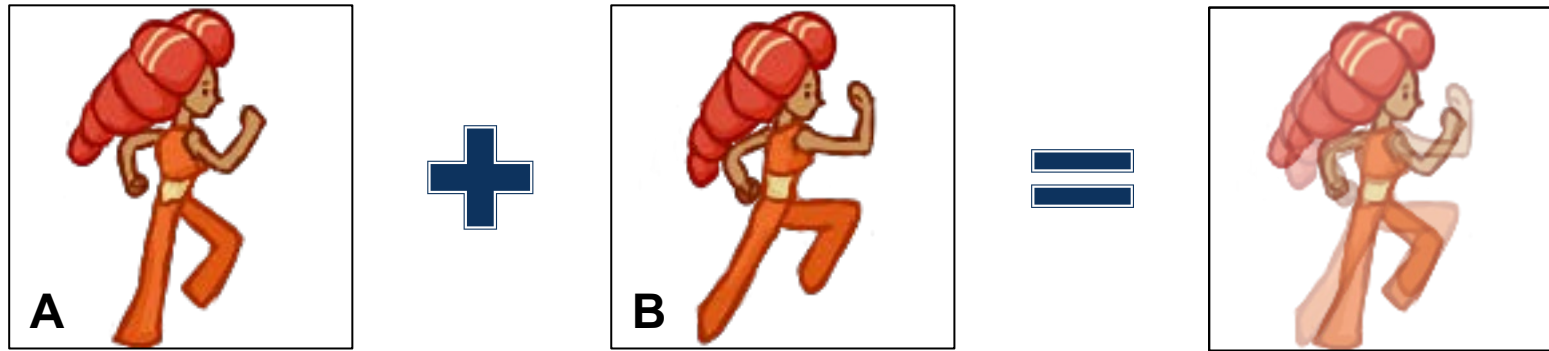


Cycle the
filmstrip
normally

Cycle the
filmstrip
normally

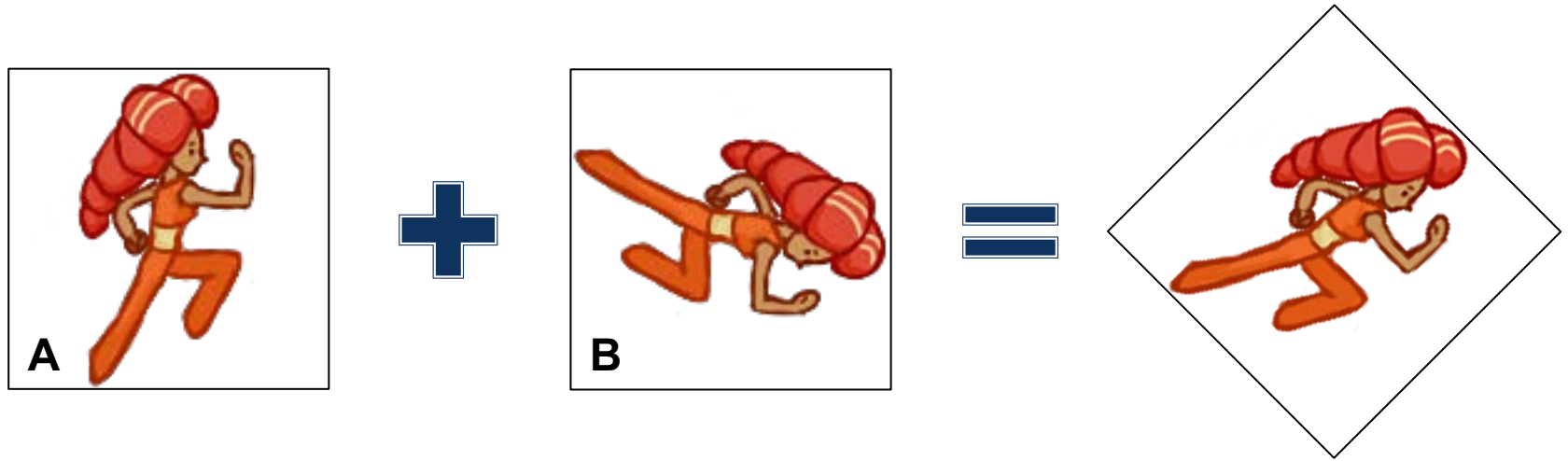
Combine
with alpha
blending

Related Concept: **Tweening**



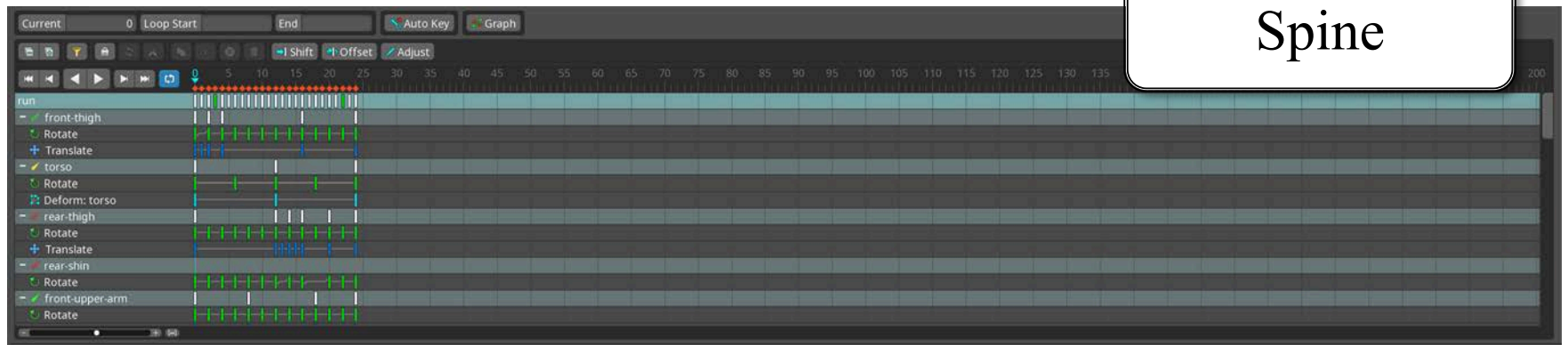
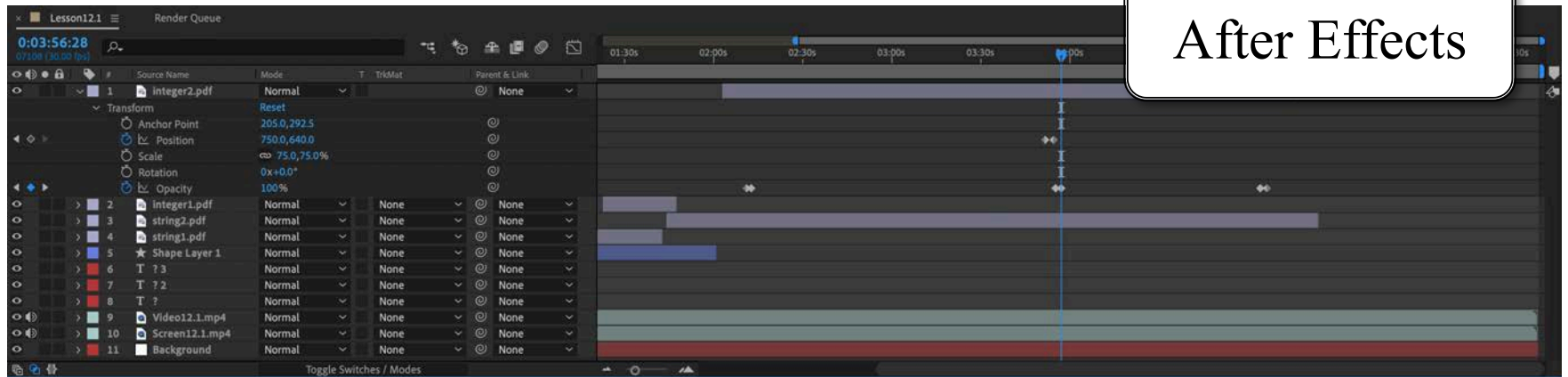
- Act of linear interpolating between animation frames
 - Because we cycle filmstrip slower than framerate
 - Implements a form of motion blur
- If animation **designed right**, makes it smoother

Tweening Works for Transforms Too



- Any transform is represented by a **matrix**
 - Can linearly interpolate matrix components
 - Gives a reasonable transform “in-between”
- **Aside:** This is a motivation for **quaternions**
 - Gives smoother interpolation for rotation

Tweening vs Keyframes



Tweening vs Keyframes

Tweening

- Specify the action
 - Give an action and a time
 - Frames are interpolations
- **Programmer** centric

Keyframes

- Specify the result
 - Give start and end points
 - Middle is interpolated
- **Designer** centric

Essentially the same concept.
Difference is the specification.

Supporting Tweened Animations

Actions

- Represents animation type
 - Moving, rotating, scaling
 - Filmstrip sequences
- But not active animation
 - Can be reused and replayed
 - Can be copied safely
- Think of as a “template”
 - Defines the tweening
 - But has no internal state

ActionManager

- Manages active animations
 - Maps actions to scene graph
 - Allocates animation state
- Has a separate update loop
 - Initialization step at start
 - Update step to increment
- Similar to **asset manager**
 - Animations have key id
 - Run update() to fit budget

Supporting Tweened Animations



ActionManager

- Manages active animations
 - Maps actions to scene graph
 - Allocates animation state
- Has a separate update loop
 - Initialization step at start
 - Update step to increment
- Similar to **asset manager**
 - Animations have key id
 - Run update() to fit budget

Executing Actions: **Transforms**

```
auto mgr = ActionManager::alloc();
```

```
auto action = RotateBy::alloc(90.0f, 2.0f);
```

```
mgr->activate(key, action, sprite);
```

```
while (mgr->isActive(key)) {
```

```
    mgr->update(TIMESTEP);
```

```
}
```

```
// No clean-up. Done automatically
```



Executing Actions: Transforms

ActionManager

```
auto mgr = ActionManager::alloc();
```

```
auto action = RotateBy::alloc(90.0f, 2.0f);
```

```
mgr->activate(key, action, sprite);
```

Map Action to
key and start

```
while (mgr->isRunning(key))
```

```
mgr->update(TIMESTEP);
```

```
}
```

```
// No clean-up
```

Increments
animation state

Action



Tweens
rotation



Executing Actions: Transforms

```
auto mgr = ActionManager::alloc();
```

```
auto action = RotateBy::alloc(90.0f, 2.0f);
```

```
mgr->activate(key, action, sprite);
```

```
while (mgr->isActive(key)) {
```

```
    mgr->update(TIMESTEP);
```

```
}
```

```
// No clean-up. Done at
```

How long
to spend

Maps to
framerate



Tweens
rotation



Executing Actions: **FilmStrips**

```
auto mgr = ActionManager::alloc();
```

```
std::vector<int> frames;
```

```
frames.push_back(f1);
```

```
...
```

```
frames.push_back(f8);
```

```
auto action = Animate::alloc(frames, 2.0f);
```

```
mgr->activate(key, action, sprite);
```

```
while (mgr->isActive(key)) {
```

```
    mgr->update(TIMESTEP);
```

```
}
```

```
// No clean-up. Done automatically
```



Executing Actions: FilmStrips

```
auto mgr = ActionManager::alloc();
```

```
std::vector<int> frames;
```

```
frames.push_back(f1);
```

```
...
```

```
frames.push_back(f8);
```

Sequence
indices

```
auto action = Animate::alloc(frames, 2.0f);
```

```
mgr->activate(key, action, sprite);
```

```
while (mgr->isActive(key)) {
```

```
    mgr->update(TIMESTEP);
```

```
}
```

```
// No clean-up. Done a
```

Frames
displayed
uniformly



Does not
tween



Executing Actions: **FilmStrips**

```
auto mgr = ActionManager::alloc();
```

```
std::vector<int> frames;  
frames.push_back(f1);
```

```
...  
frames.push_back(f2);
```

```
auto action
```

```
mgr->activate
```

```
while (mgr->is
```

```
    mgr->update(TIMESTEP);
```

```
}
```

```
// No clean-up. Done automatically
```

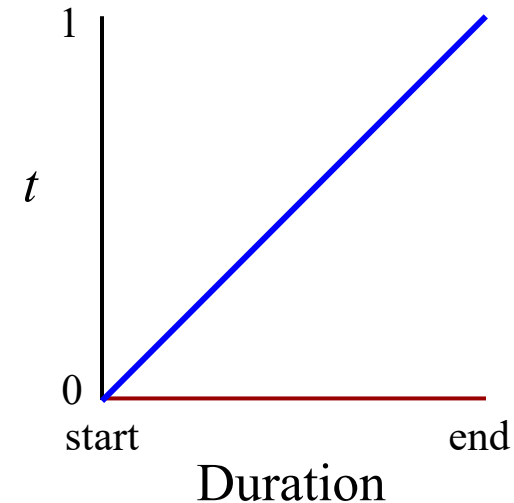


Alternatively, could
specify time per frame



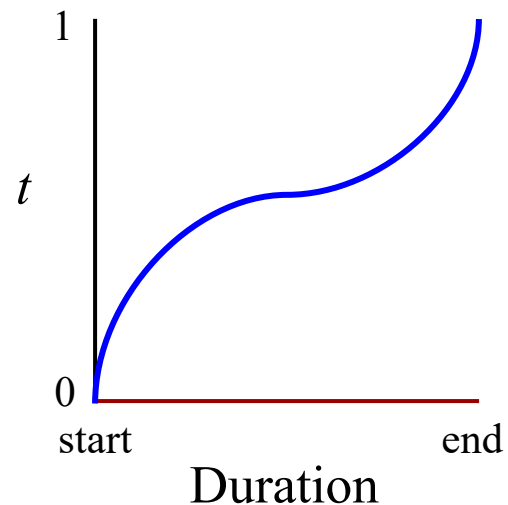
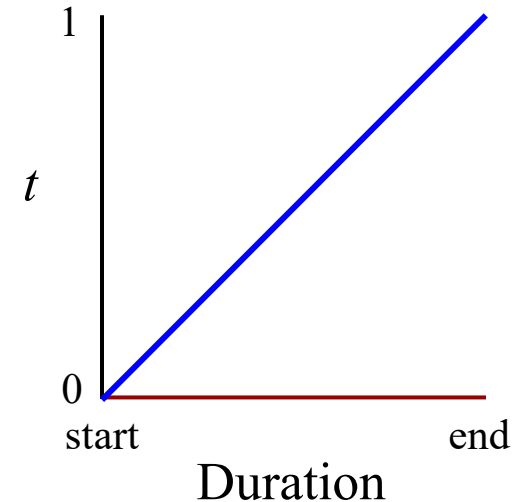
Easing Function

- Basic approach to tweening
 - Specify duration to animate
 - Set $t = 0$ at beginning
 - Normalize $t = 1$ at end
 - Interpolate value with t
- How does t change?
 - Usually done *linearly*
 - Could be some other way
- **Easing**: how to change t
 - Used for bouncing effects
 - Best used for *transforms*

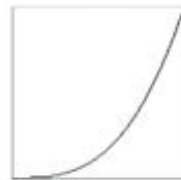


Easing Function

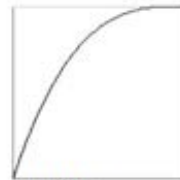
- Basic approach to tweening
 - Specify duration to animate
 - Set $t = 0$ at beginning
 - Normalize $t = 1$ at end
 - Interpolate value with t
- How does t change?
 - Usually done *linearly*
 - Could be some other way
- **Easing**: how to change t
 - Used for bouncing effects
 - Best used for *transforms*



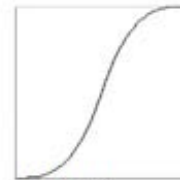
Classic Easing Functions



easeIn



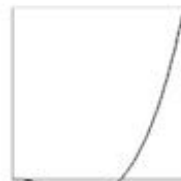
easeOut



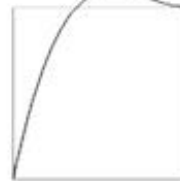
easeInOut



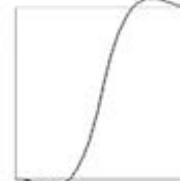
easeOutIn



easeInBack



easeOutBack



easeInOutBack



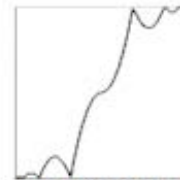
easeOutInBack



easeInBounce



easeOutBounce



easeInOutBounce



easeOutInBounce



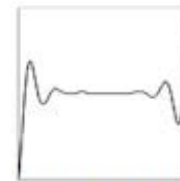
easeInElastic



easeOutElastic

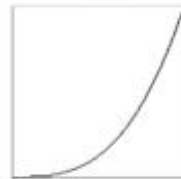


easeInOutElastic



easeOutInElastic

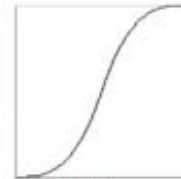
Classic Easing Functions



easeIn



easeOut



easeInOut



easeOutIn



easeInBounce



easeOutBounce



easeInOutBounce



easeOutInBounce



easeInElastic



easeOutElastic



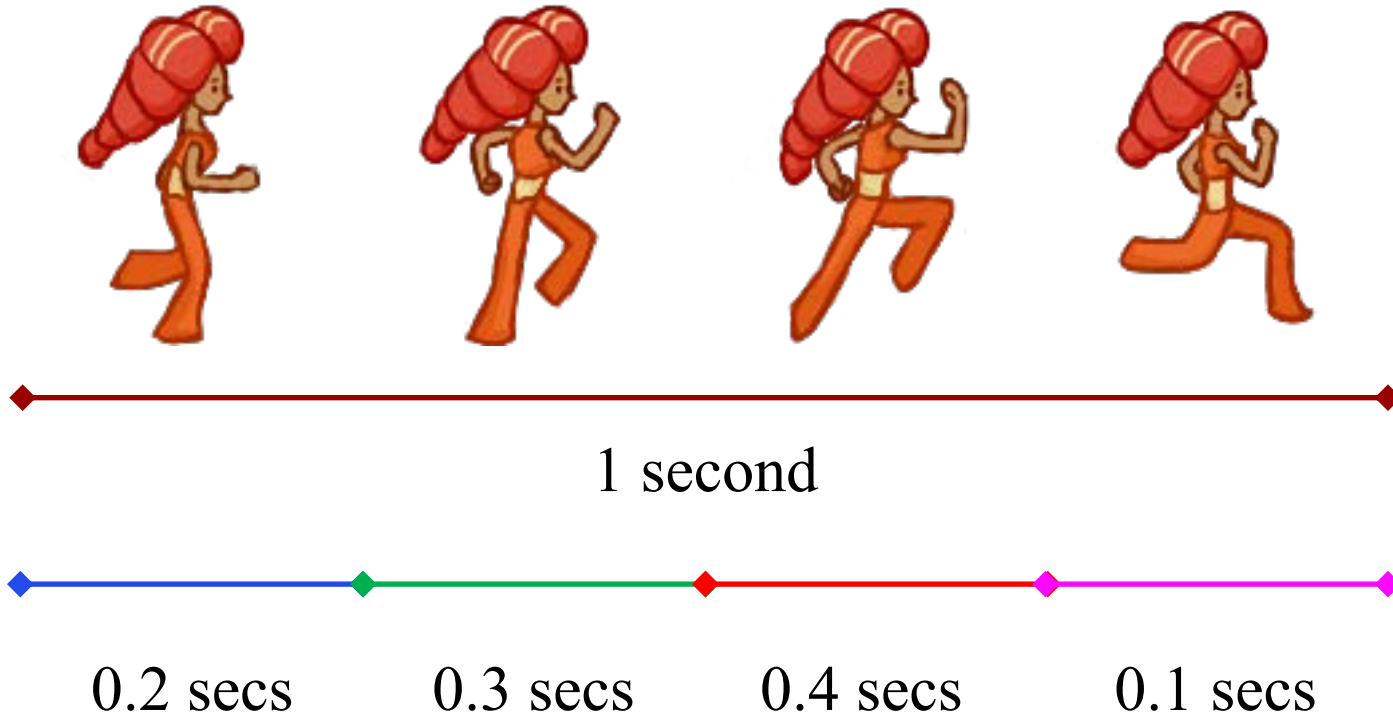
easeInOutElastic



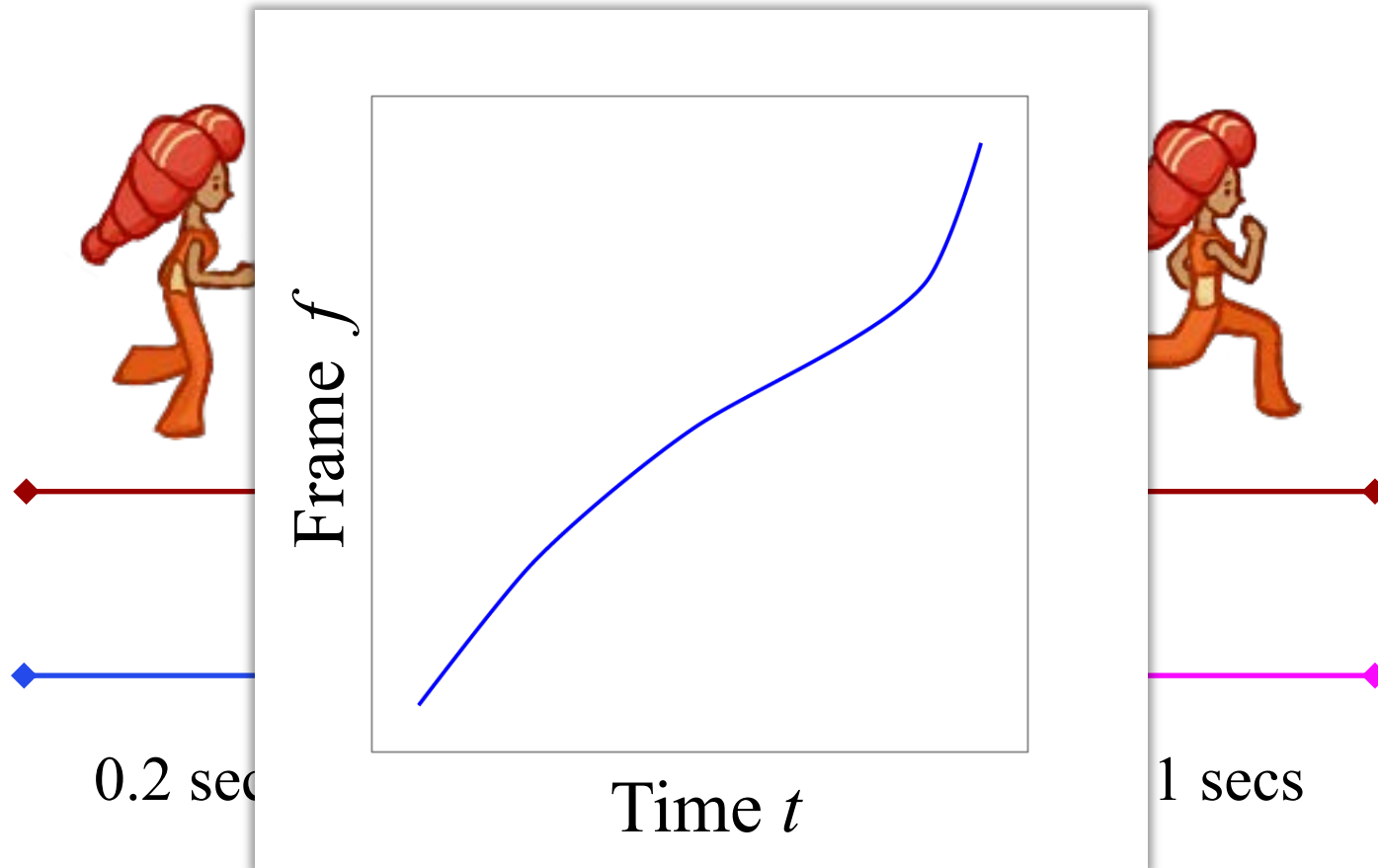
easeOutInElastic

<http://easings.net>

Application to Sprite Animation



Application to Sprite Animation



Problem With Decoupled Animation

```
auto mgr = ActionManager::alloc();
```

```
auto action = RotateBy::alloc(90.0f, 2.0f);
```

```
mgr->activate(key, action, sprite);
```

What if we change our mind before 2 seconds?



Problems With Decoupled Animation

```
auto mgr = ActionManager::alloc();
```

```
auto action = RotateBy::alloc(90.0f, 2.0f);
```

```
mgr->activate(key, action, sprite);
```

Compatible: Combine
Incompatible: Replace



Modular Animation

- Break asset into parts
 - Natural for joints/bodies
 - Animate each separately
- Cuts down on filmstrips
 - Most steps are transforms
 - Very natural for tweening
 - Also better for physics
- Several tools to help you
 - **Example:** *Spriter, Spine*
 - Great for visualizing design



Modular Animation

- Break asset into parts
 - Natural for joints/bodies
 - Animate each separately
- Cuts down on filmstrips
 - Most steps are transforms
 - Very natural for tweening
 - Also better for physics
- Several tools to help you
 - **Example:** *Spriter, Spine*
 - Great for visualizing design



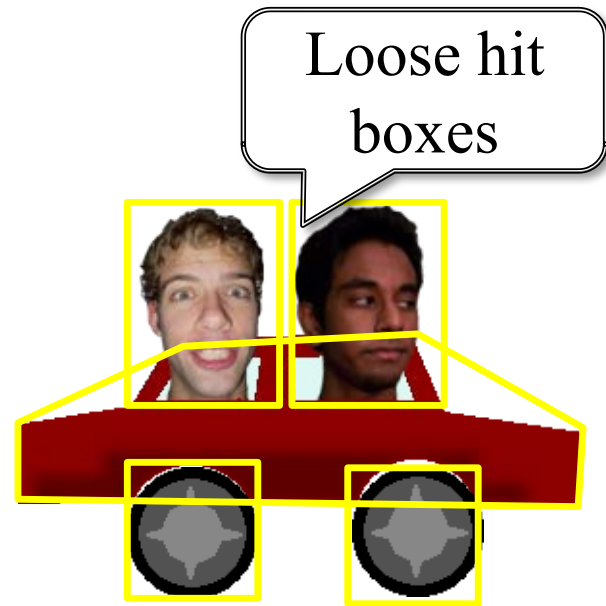
Modular Animation

- Break asset into parts
 - Natural for joints/bodies
 - Animate each separately
- Cuts down on filmstrips
 - Most steps are transforms
 - Very natural for tweening
 - Also better for physics
- Several tools to help you
 - **Example:** *Spriter, Spine*
 - Great for visualizing design



Modular Animation

- Break asset into parts
 - Natural for joints/bodies
 - Animate each separately
- Cuts down on filmstrips
 - Most steps are transforms
 - Very natural for tweening
 - Also better for physics
- Several tools to help you
 - **Example:** *Spriter, Spine*
 - Great for visualizing design



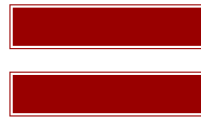
- Inside hit box can safely
 - Transform with duration
 - Tween animations
 - Manage multiple actions

Problems With Decoupled Animation

Transform Tweening



Physical Animation



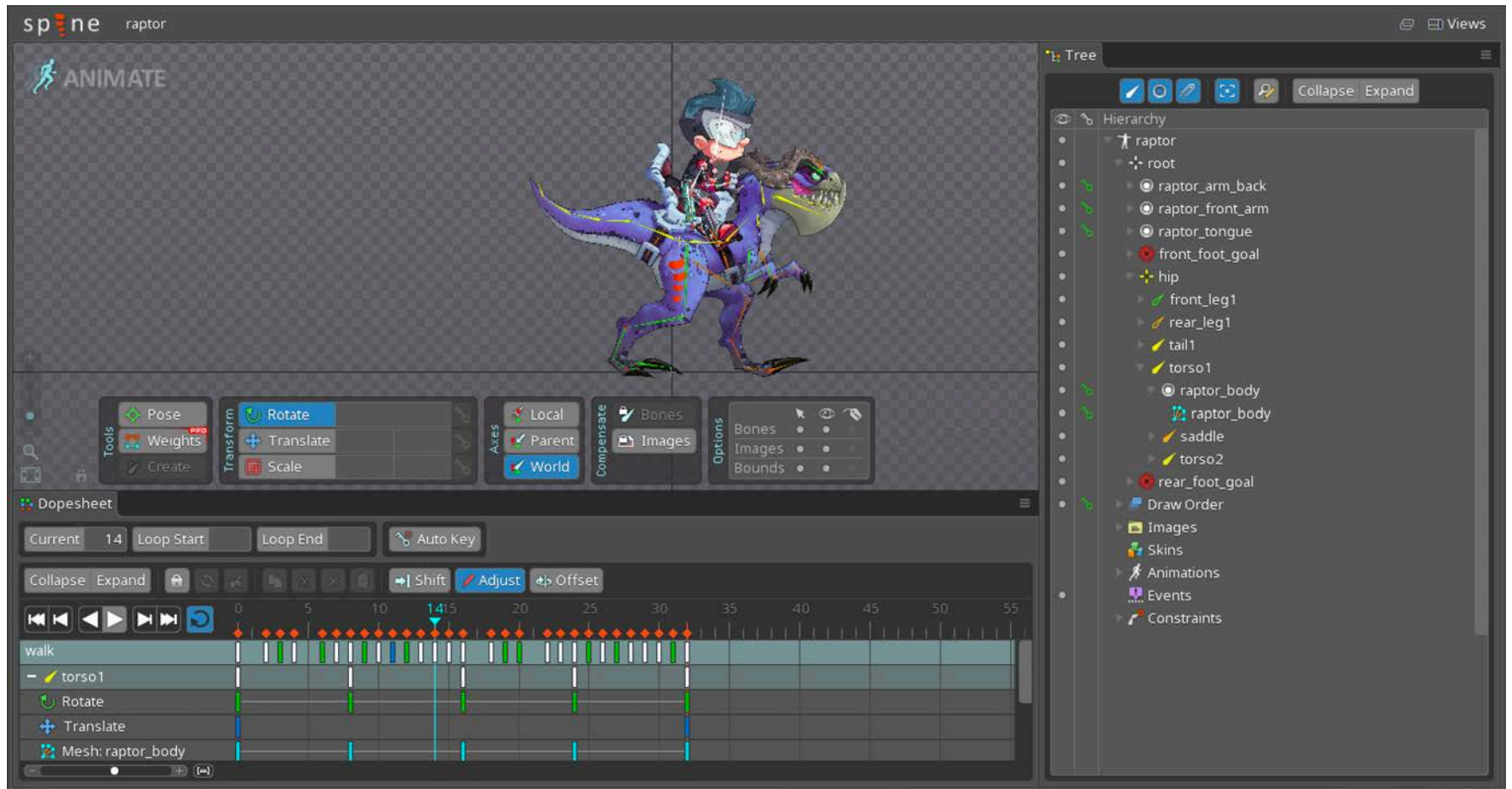
Complete Disaster

Aside: Skinning

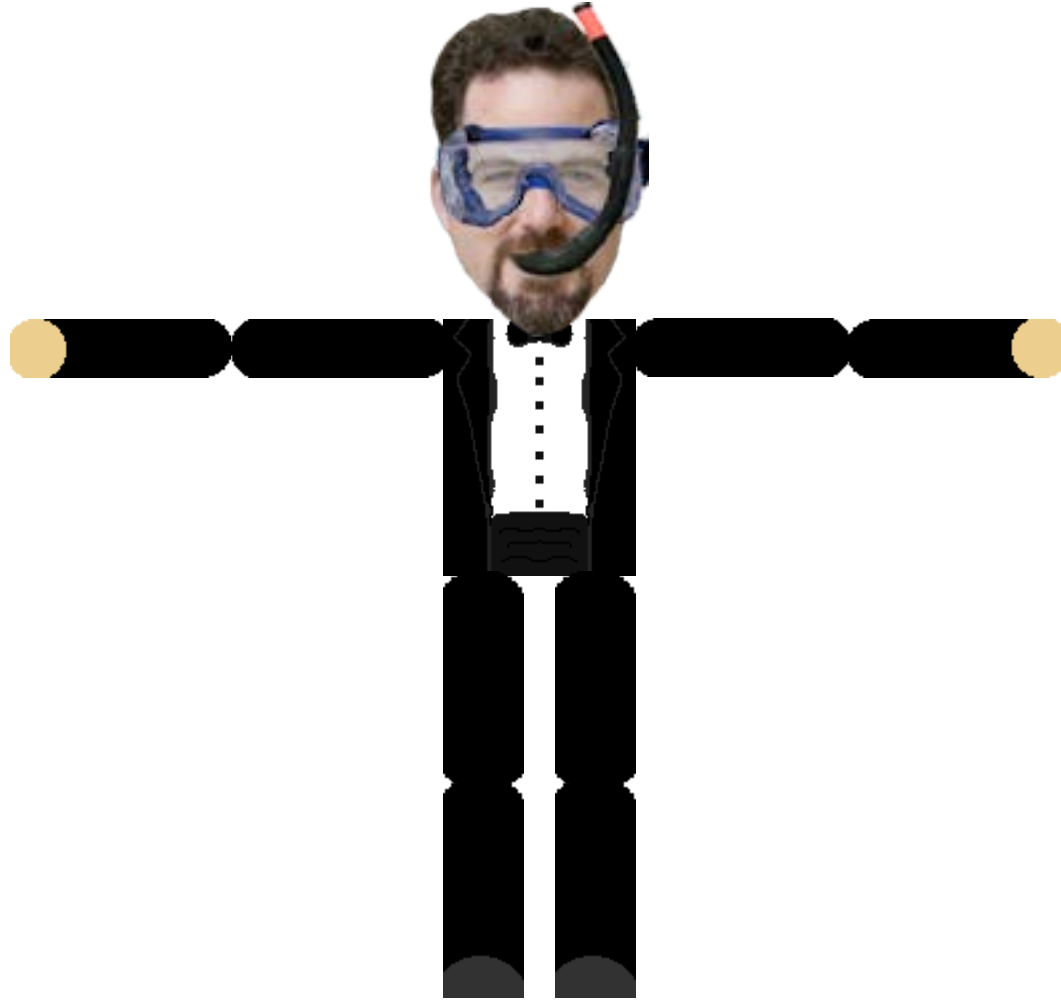


Way to get extra usage
of hand-drawn frames

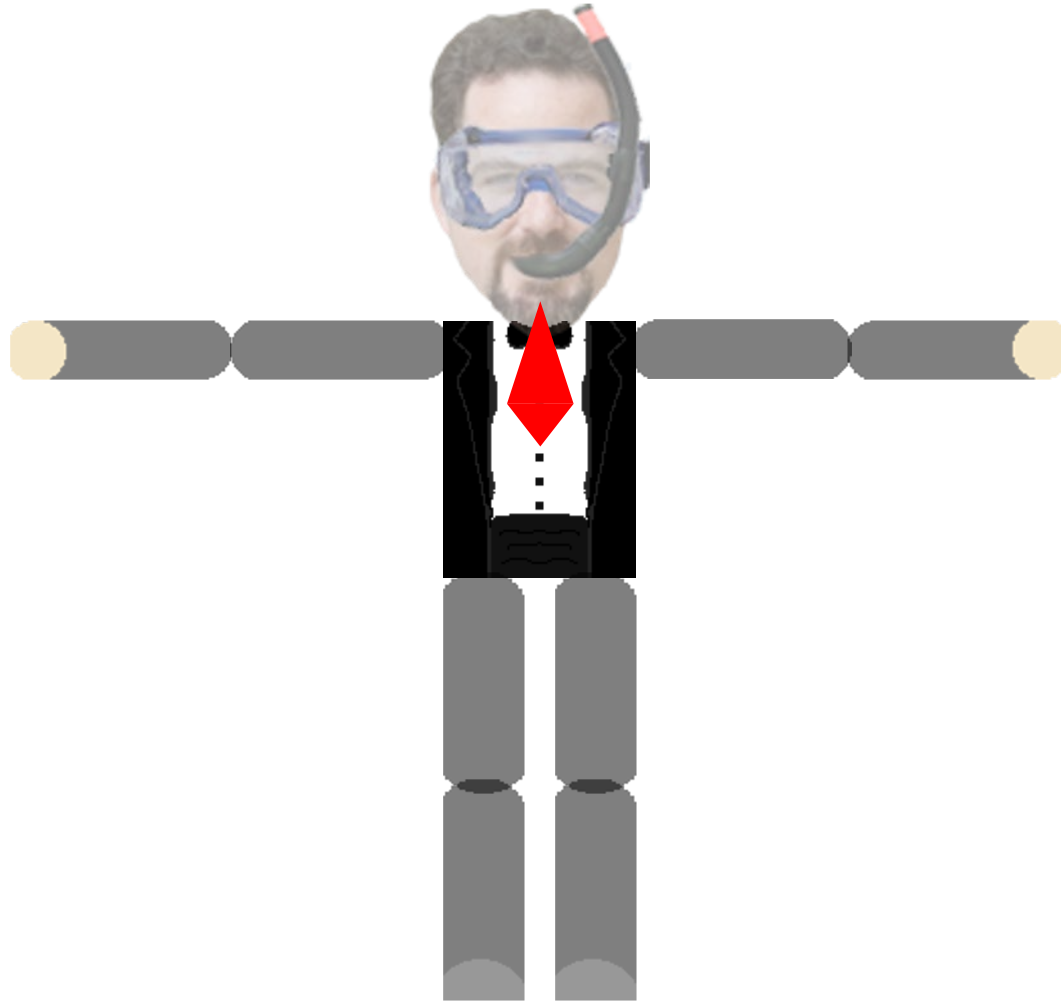
Spine Demo



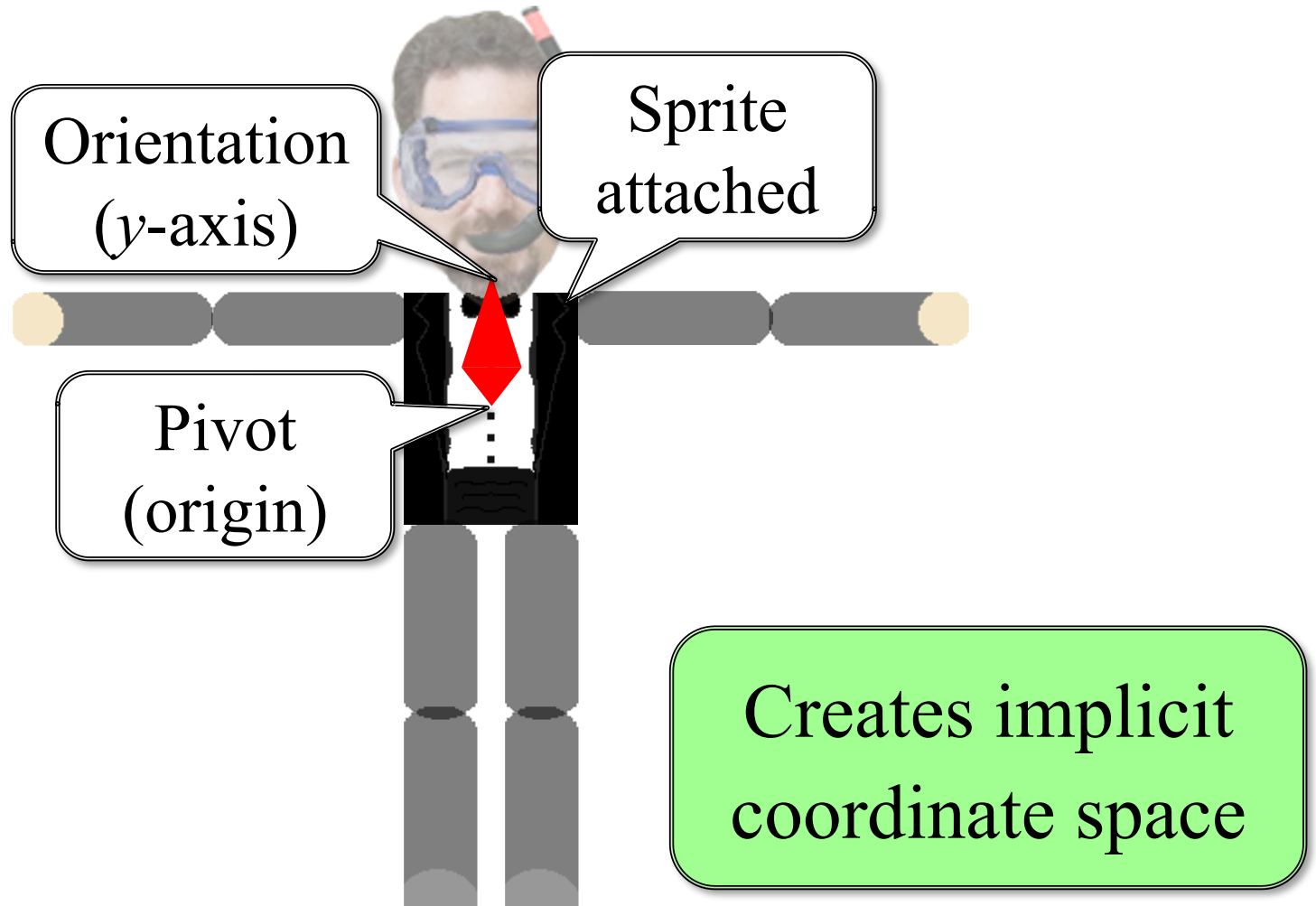
Basic Idea: **Bones**



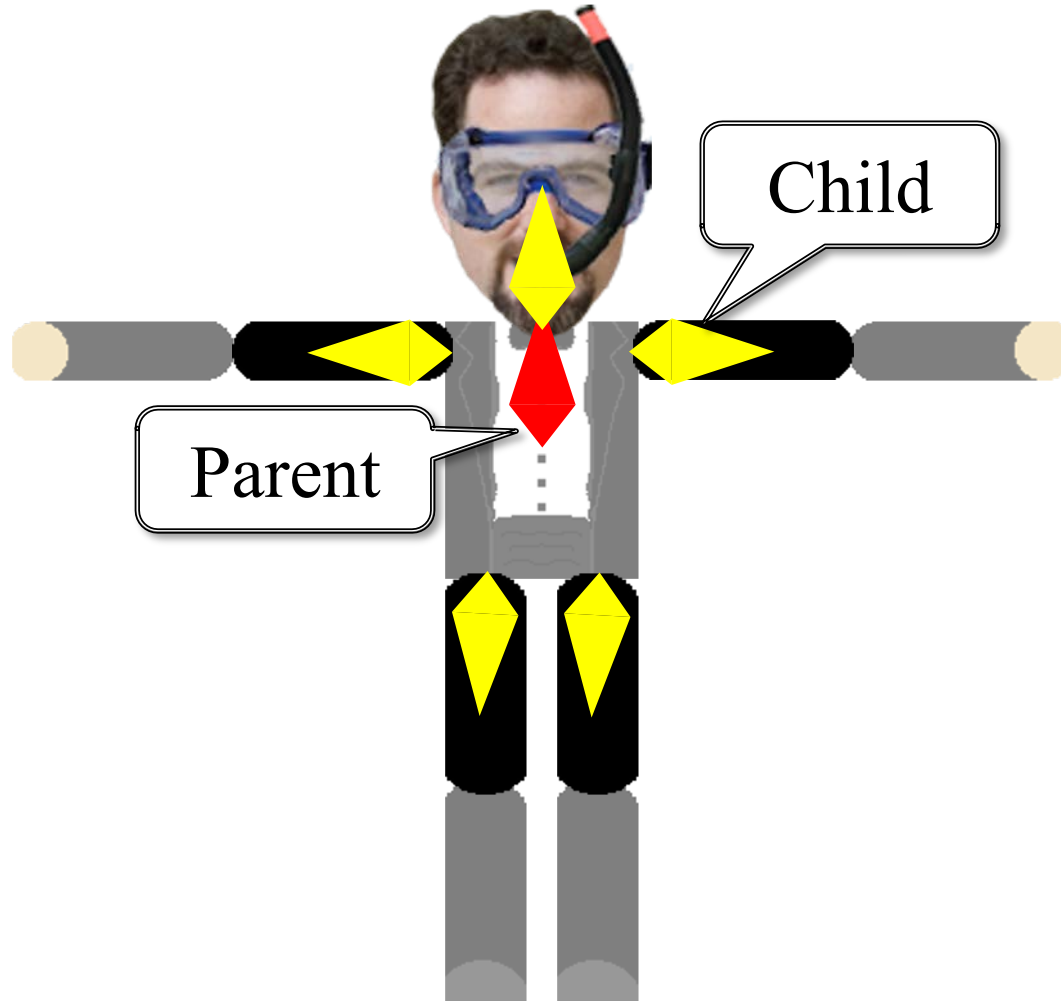
Basic Idea: **Bones**



Basic Idea: **Bones**



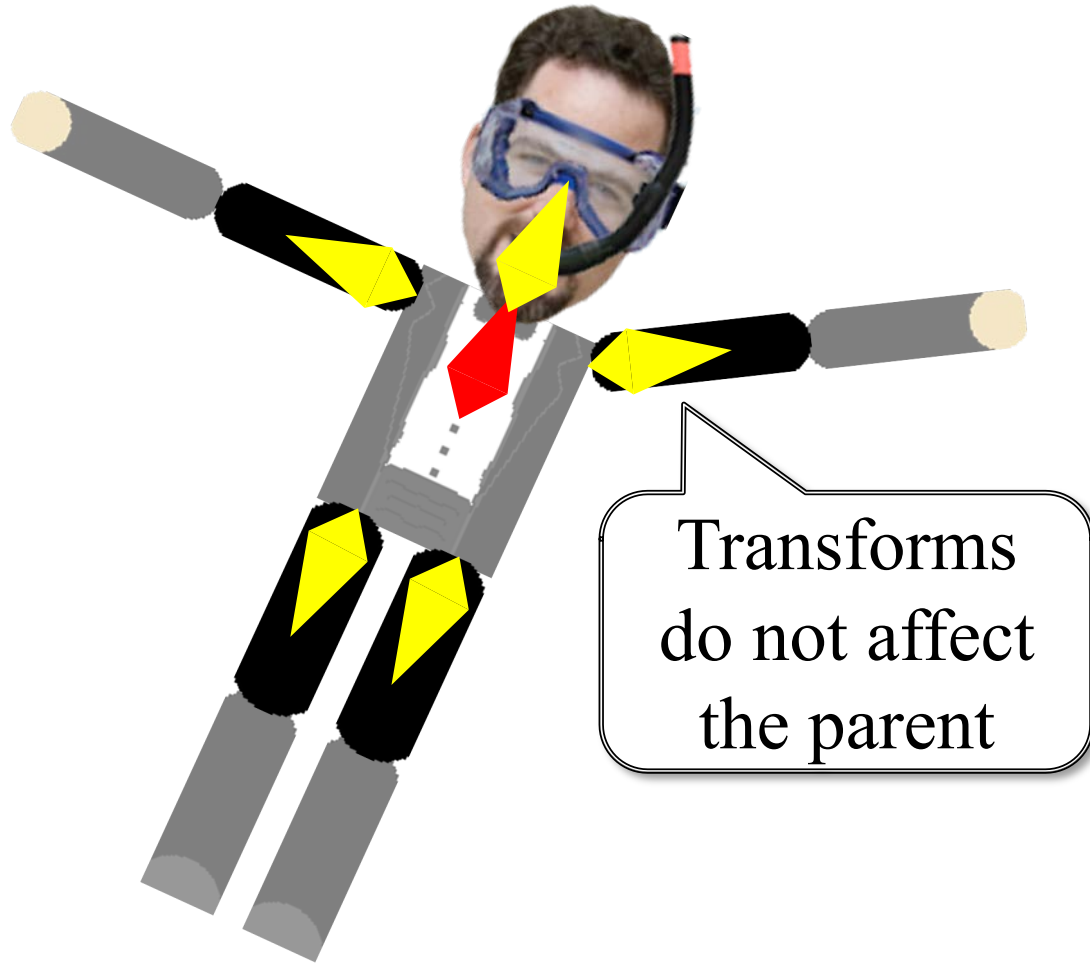
Bones are Heirarchical



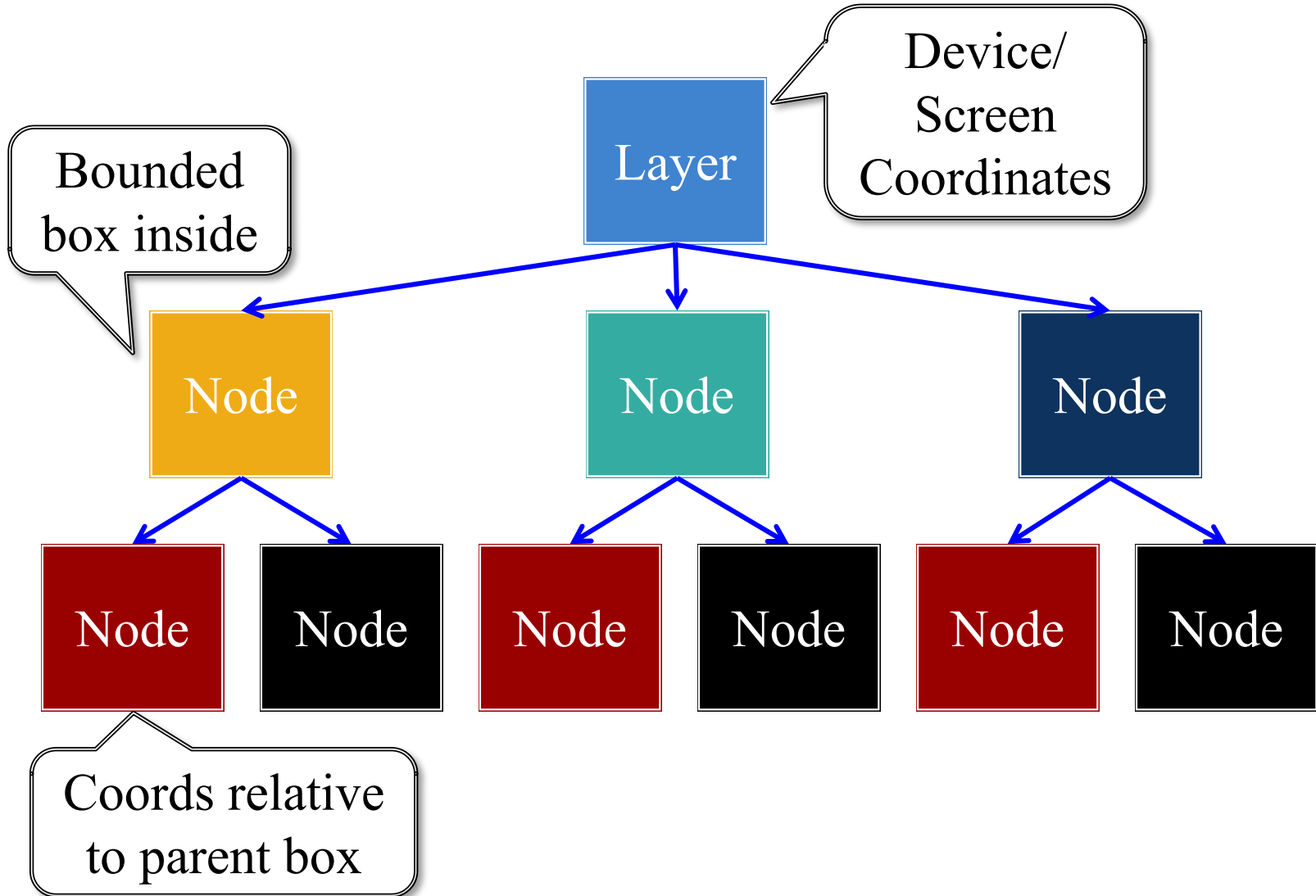
Bones are Heirarchical



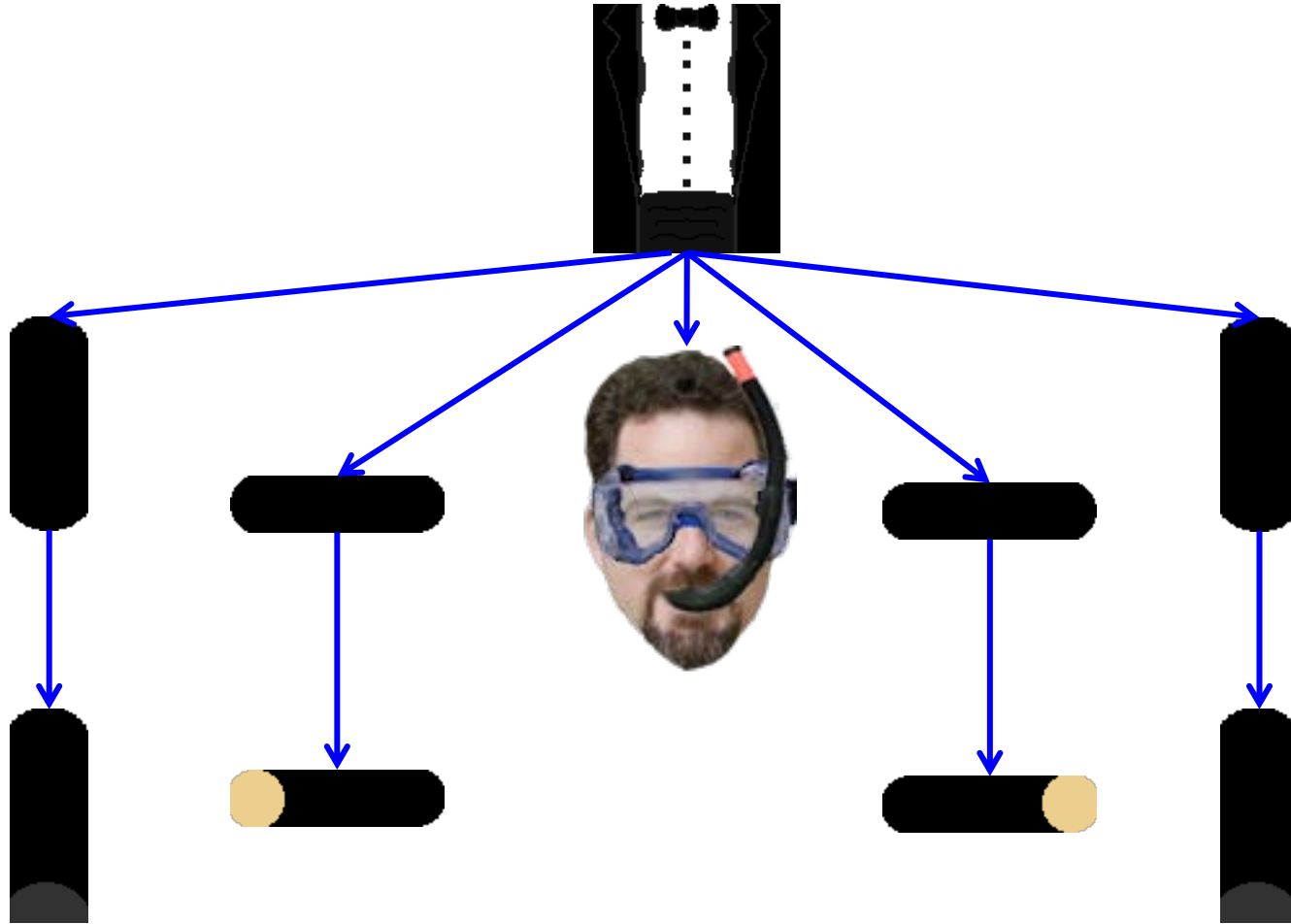
Bones are Heirarchical



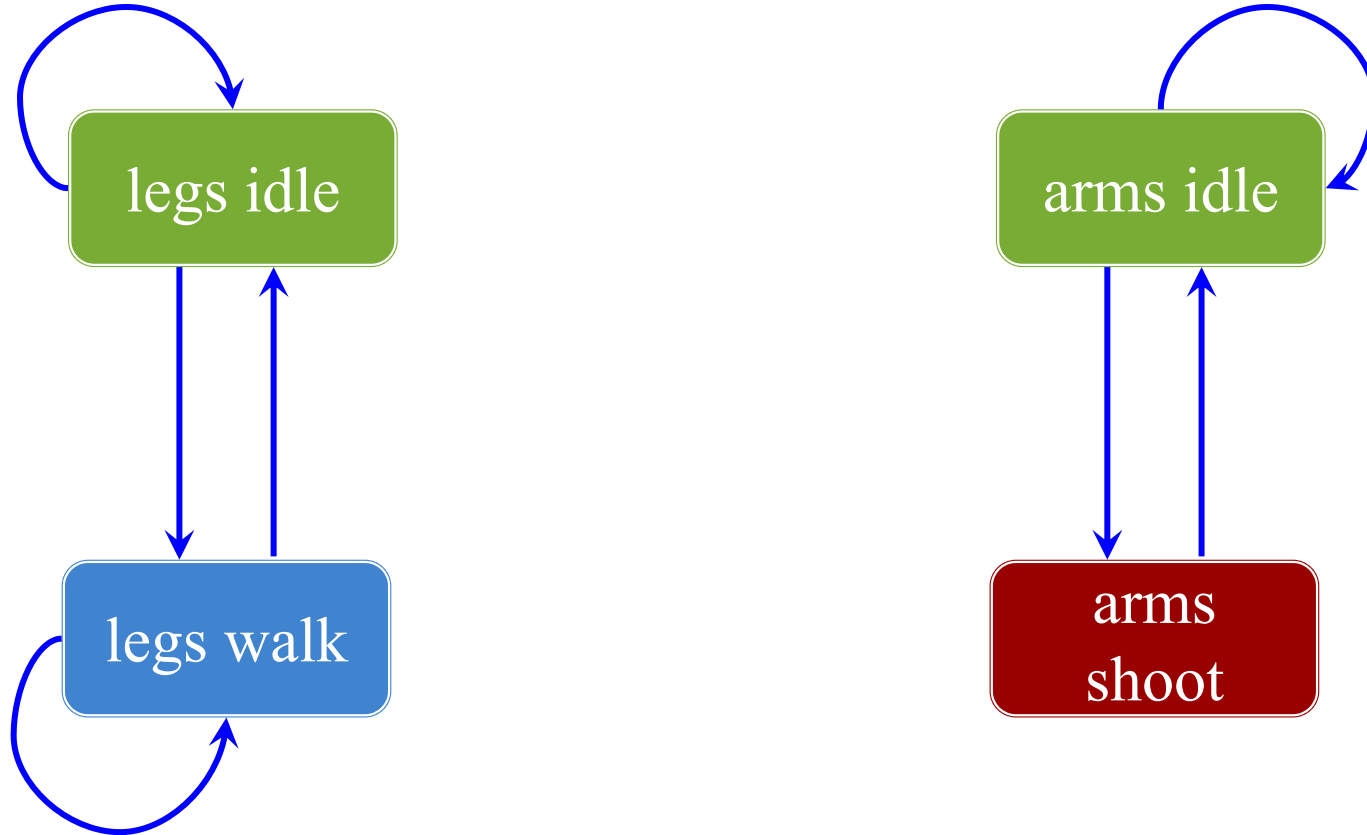
Recall: Scene Graph Hierarchy



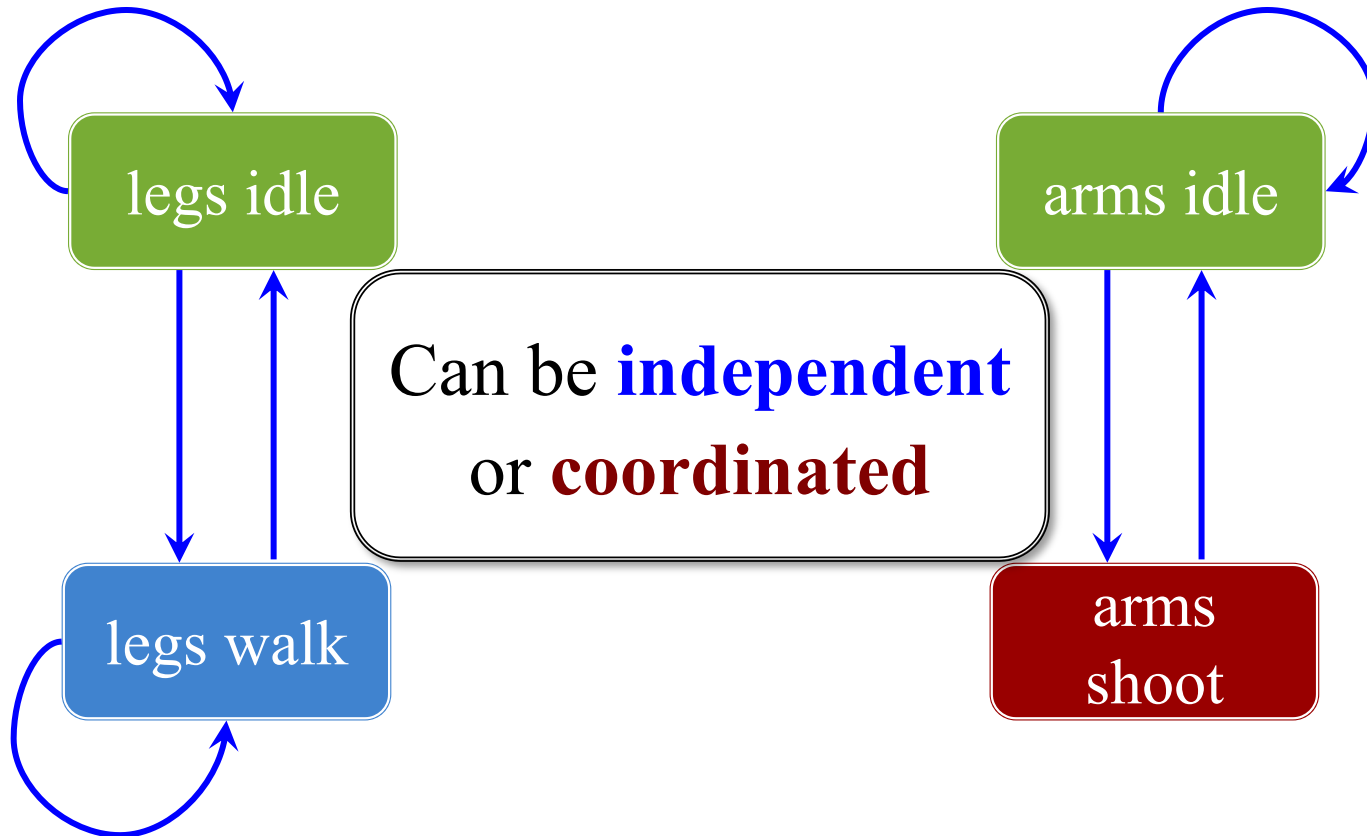
Bones are a Scene Graph Visualization



Manage With Multiple State Machines



Manage With Multiple State Machines



Summary

- Standard 2D animation is **flipbook** style
 - Create a sequence of frames in sprite sheet
 - Switch between sequences with state machines
- **Tweening** supports interpolated transitions
 - Helpful for motion blur, state transitions
 - Transforms can be combined with easing functions
- Professional 2D animation uses **modular sprites**
 - Scene graphs are a simplified form of model rigging
 - State machine coordination can be very advanced