the gamedesigninitiative at cornell university

Lecture 7

2D Animation

Animation Basics: Sprite Sheets

- 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
  - Programmer is actually the one doing animation

CUGL Has Two Options

SpriteNode

- Scene graph node
  - Fits into JSON specification
  - Draw automatically
- Does require a scene graph
  - Limits drawing flexibility
  - Bad for some perspectives (e.g. isometric)

SpriteSheet

- A stand-alone class
  - But familiar drawing API
  - Pass SpriteBatch & transform
- Not part of a scene graph
  - Has a lot more flexibility
  - But less input from designer

CUGL Has Two Options

SpriteNode

- Scene graph node
  - Fits into JSON specification
  - Draw automatically
- Does require a scene graph
  - Limits drawing flexibility
  - Bad for some perspectives (e.g. isometric)

SpriteSheet

- A stand-alone class
  - But familiar drawing API
  - Pass SpriteBatch & transform
- Not part of a scene graph
  - Has a lot more flexibility
  - But less input from designer

Both are stateful!
These are not assets.

Adjusting your Speed

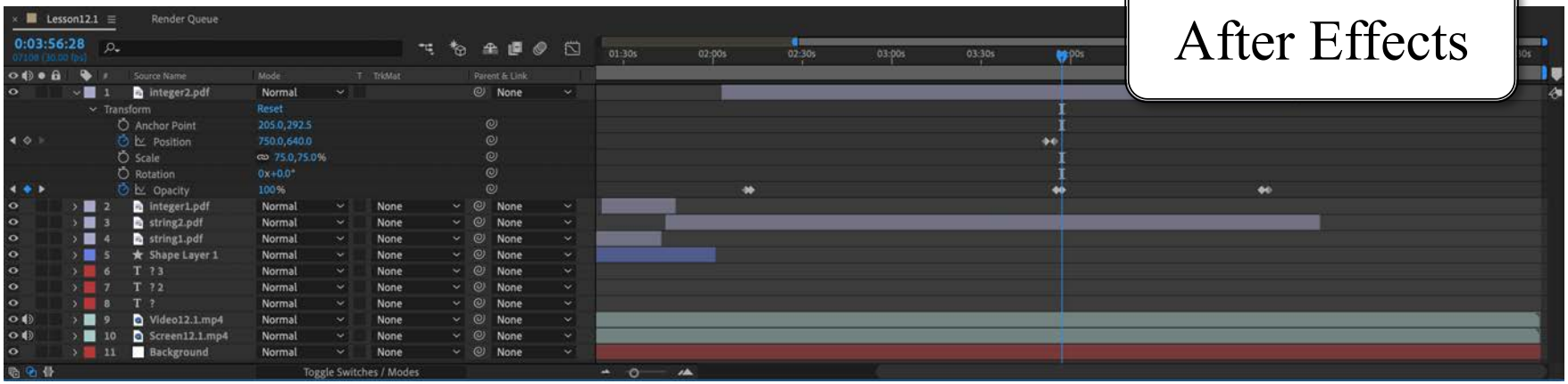
- Do not want to go too fast
  - 1 animation frame = 16 ms
  - Walk cycle = 8/12 frames
  - Completed in 133-200 ms
- General solution: *cooldowns*
  - Add an int timer to your object
  - Go to next frame when it is 0
  - Reset it to > 0 at new frame
- Simple but tedious
  - Have to do for each object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

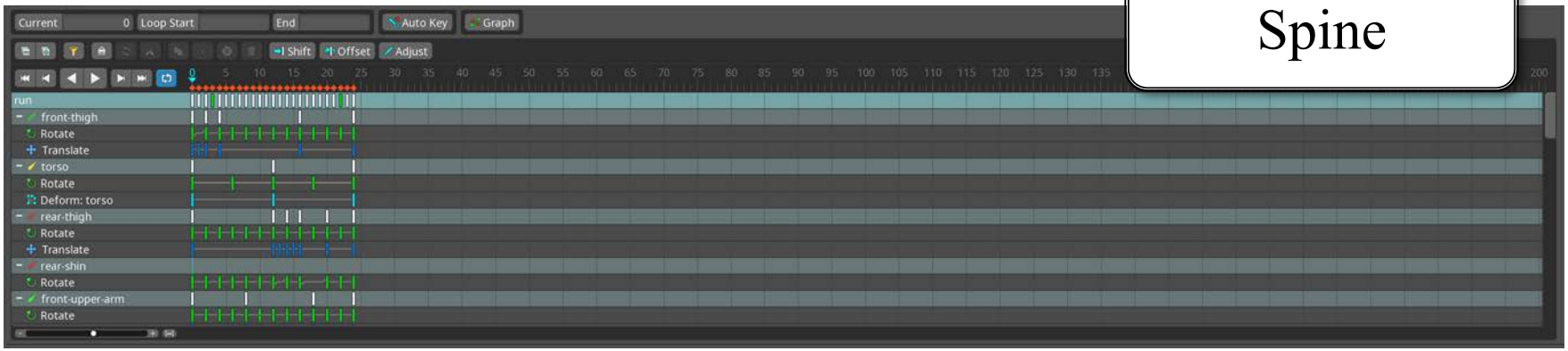
Adjusting your Speed

- Do not want to go too fast
  - 1 animation frame = 16 ms
  - Walk cycle = 8/12 frames
  - Completed in 133-200 ms
- General solut
  - Add an int tir
  - Go to next fra
  - Reset it to > 0 at new frame
- Simple but tedious
  - Have to do for each object
  - Assumes animation is in a loop

Too programmer focused.
How is designer involved?

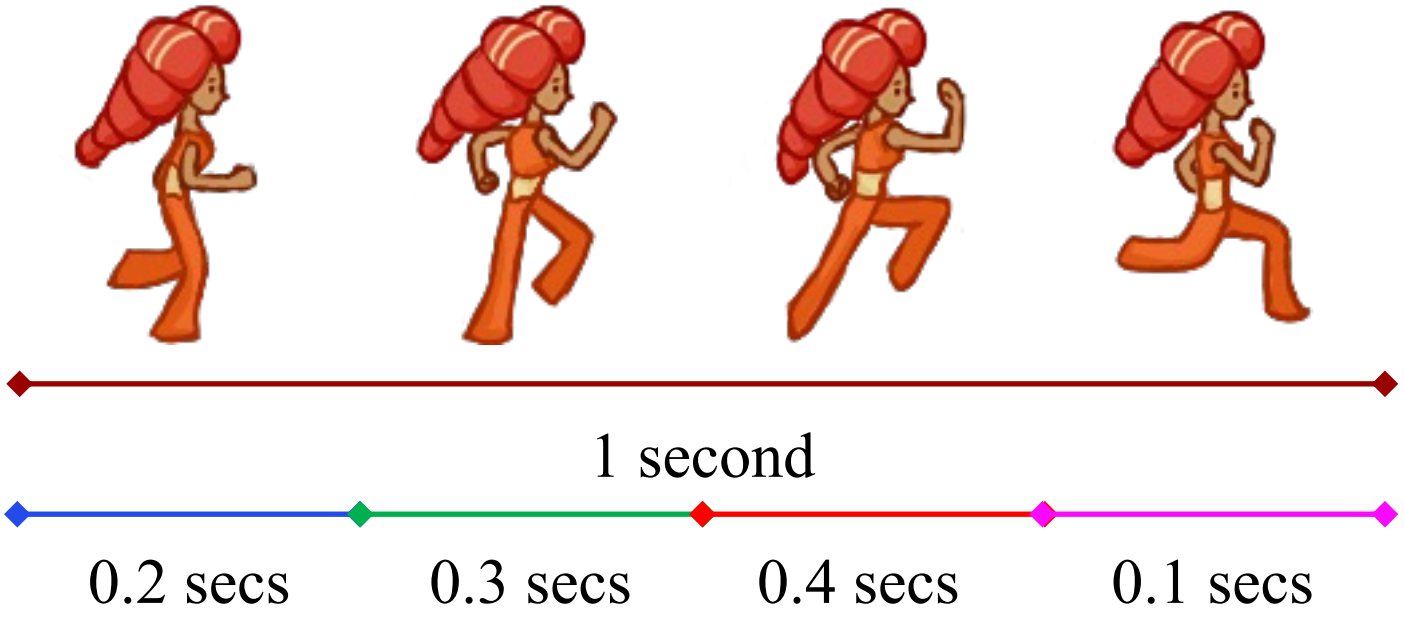
Animation Concept: Keyframes





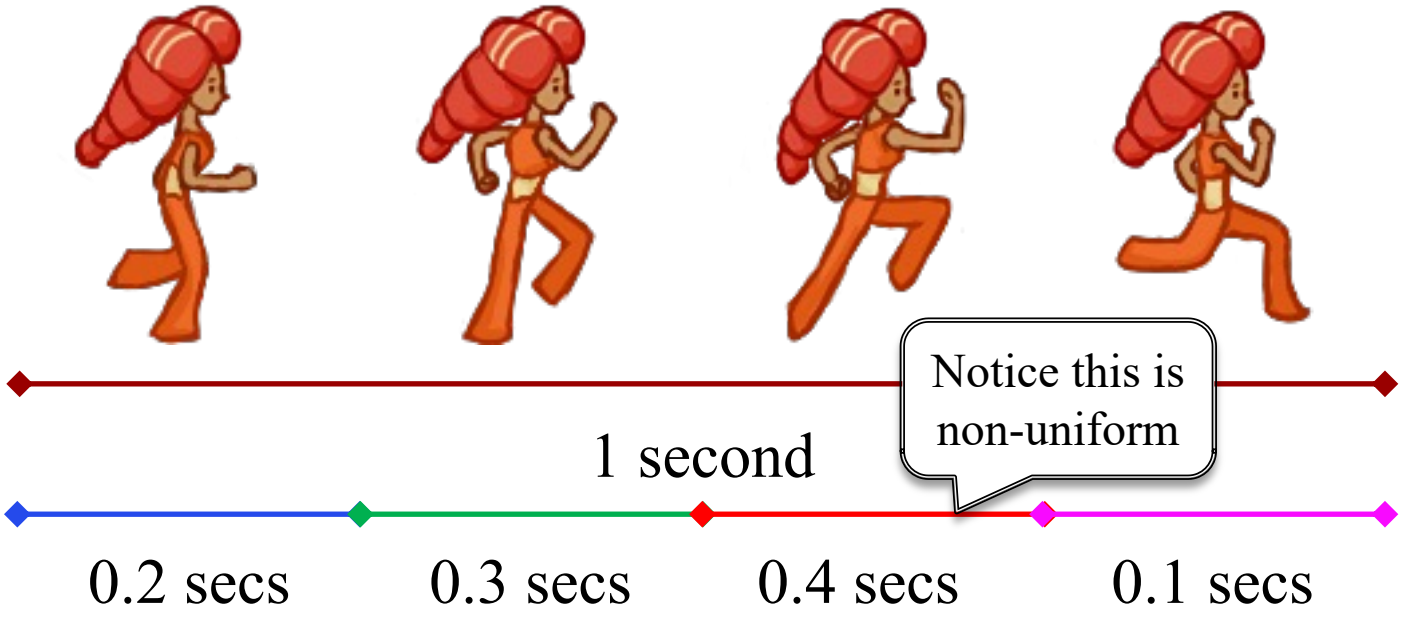
KeyFrames

- Animator creates a timeline
  - Specifies how long animation will last
  - Specifies when a change happens (key frames)
- Ideally, want this in game engine too.



KeyFrames

- Animator creates a timeline
  - Specifies how long animation will last
  - Specifies when a change happens (key frames)
- Ideally, want this in game engine too.



KeyFrames in CUGL

ActionFunction

- Represents an animation
  - User-defined function
  - No return; just animates
- Function takes
  - A value [0,1]
  - A state (begin/update/end)
- CUGL has several **factories**
  - Create common animations
  - Ideal with scene graphs

ActionTimeline

- Manages active animations
  - Assigns them a duration
  - Tracks their current 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

KeyFrames in CUGL

ActionTimeline

- Manages active animations
  - Assigns them a duration
  - Tracks their current 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: Sprite Sheets

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

std::vector<int> frames;
frames.push_back(f1);
 ...
frames.push_back(f8);

auto seq = AnimateSprite::alloc(frames);

auto action = seq->attach(sprite);

mgr->activate(key,action,duration);
while (mgr->isActive(key)) {
    mgr->update(TIMESTEP);
}

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

sprite

Executing Actions: Sprite Sheets

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

std::vector<int> frames;
frames.push_back(f1);
 ...
frames.push_back(f8);

auto seq = AnimateSprite::alloc(frames);

auto action = seq->attach(sprite);

mgr->activate(key,action,duration);
while (mgr->isActive(key)) {
    mgr->update(TIMESTEP);
}

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

Allocate timeline

Progress the loop

sprite

Executing Actions: Sprite Sheets

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

std::vector<int> frames;
frames.push_back(f1);
 ...
frames.push_back(f8);

auto seq = AnimateSprite::alloc(frames);

auto action = seq->attach(sprite);

mgr->activate(key,action,duration);
while (mgr->isActive(key)) {
    mgr->update(TIMESTEP);
}

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

Sequence the frames

Create a **factory**

sprite

Executing Actions: Sprite Sheets

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

std::vector<int> frames;
frames.push_back(f1);
 ...
frames.push_back(f8);

auto seq = AnimateSprite::alloc(frames);

auto action = seq->attach(sprite);

mgr->activate(key,action,duration);
while (mgr->isActive(key)) {
    mgr->update(TIMESTEP);
}

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

Create an **action**

Run for a given duration

sprite

Executing Actions: Sprite Sheets

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

std::vector<int> frames;
frames.push_back(f1);
 ...
frames.push_back(f8);

auto seq = AnimateSprite::alloc(frames);

auto action = seq->attach(sprite);

mgr->activate(key,action,duration);
while (mgr->isActive(key)) {
    mgr->update(TIMESTEP);
}

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

sprite

Can give non-uniform weights

Change is abrupt

What About the Abrupt Change?

- Not an issue if animation is fast
  - Frames change too fast to notice
  - Movies are fine with 24 fps
- But what if animation is slower?
  - Could make more frames
  - But much more work on artist
- Want a way to make new frames
  - Ideally done by the computer
  - Called **tweening** (in-betweening)
- **Aside**: generative AI?

What About the Abrupt Change?

- Not an issue if animation is fast
  - Frames change too fast to notice
  - Movies are fine with 24 fps
- But what if animation is slower?
  - Could make more frames
  - But much more work on artist
- Want a way to make new frames
  - Ideally done by the computer
  - Called **tweening** (in-betweening)
- **Aside**: generative AI?

Simple Approach: 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

Simple Approach: 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

Simple Approach: 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

Simple Approach: 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

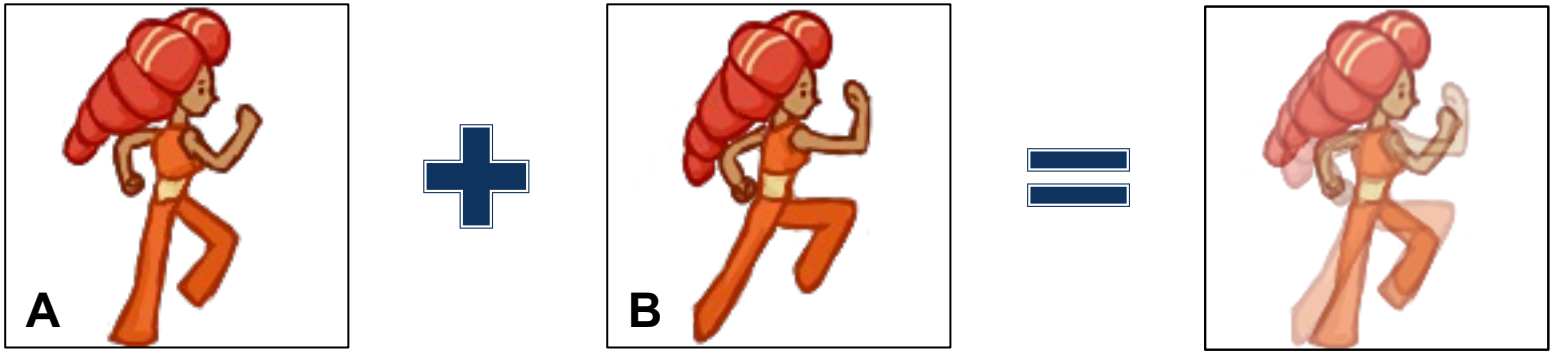
Simple Approach: Crossfade Blending

- Linear interpolation on colors

$$r_c = tr_a + (1-t)r_b$$
$$g_c = tg_a + (1-t)g_b$$
$$b_c = tb_a + (1-t)b_b$$

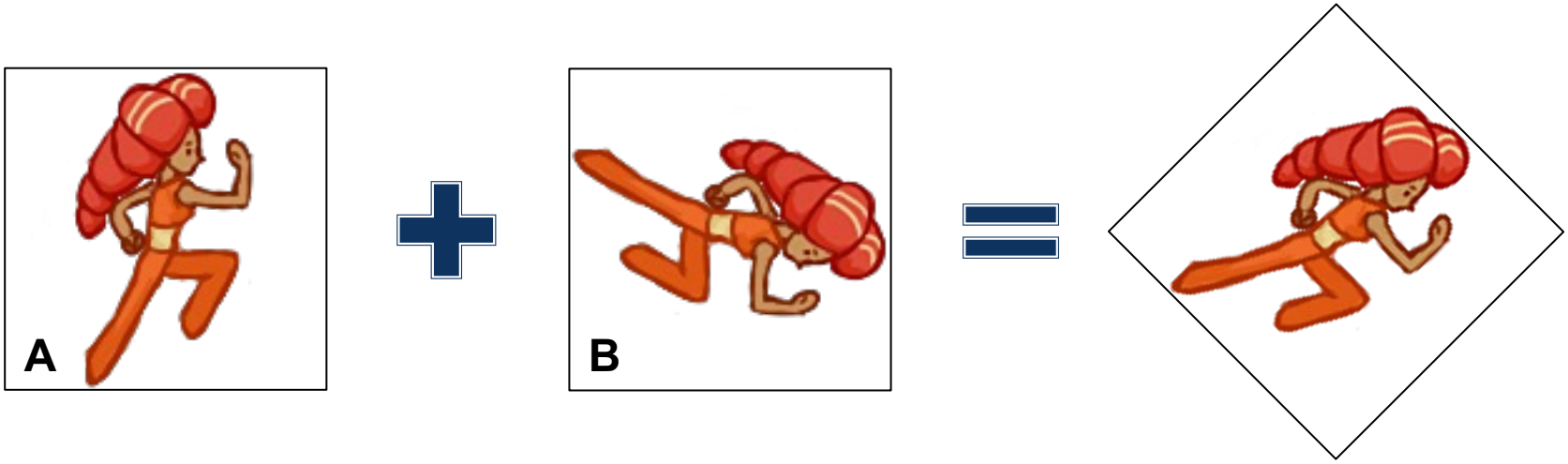
Note weights sum to 1.0

Tweening: Interpolating In-Betweens



- 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 and Transforms



- 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

Application to Sprite Animation

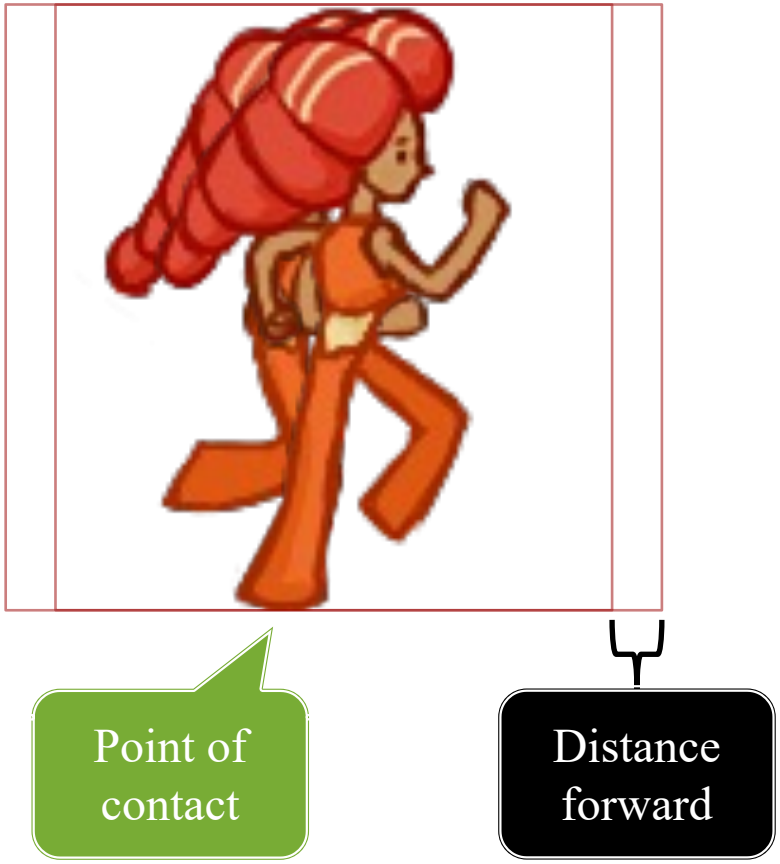
- Movement is *two* things
  - **Animation** of the filmstrip
  - **Translation** of the image
  - These two must align
- **Example:** Walking
  - Foot is point of contact
  - "Stays in place" as move
  - This constrains translation
- Make movement regular
  - Measure distance per frame
  - Keep same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

Application to Sprite Animation

- 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

Application to Sprite Animation

- 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



Executing Actions: **Movement**

```
// Create the factories
auto seq1 = AnimateSprite::alloc(frames);
auto seq2 = MoveBy::alloc(Vec2(PACE,0));

// Create the actions
auto flip = seq1->attach(sprite);
auto shift = seq2->attach(sprite);

// Activate and animate
mgr->activate(key1, flip, duration);
mgr->activate(key2, shift, duration);

while (mgr->isActive(key)) {
    mgr->update(TIMESTEP);
}
```

Executing Actions: **Movement**

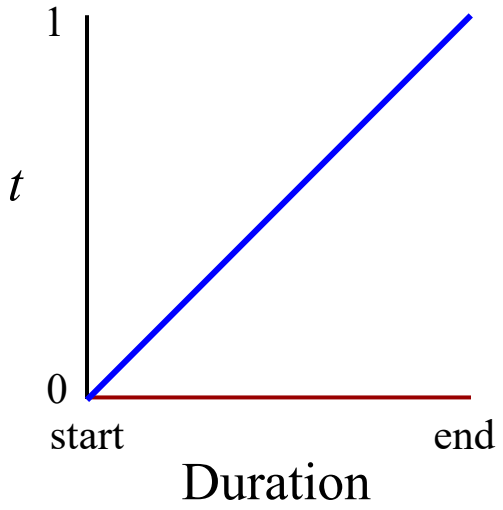
```
// Create the factories
auto seq1 = AnimateSprite::alloc(frames);
auto seq2 = MoveBy::alloc(Vec2(PACE,0));

// Create the act
auto flip = seq1->
auto shift = seq2->attach(sprite);

// Activate and animate
mgr->activate(key1, flip, duration);
mgr->activate(key2, shift, duration);

while (mgr->isActive(key)) {
    mgr->update(TIMESTEP);
}
```

Currently only supported on scene graphs

Ensures animations in sync

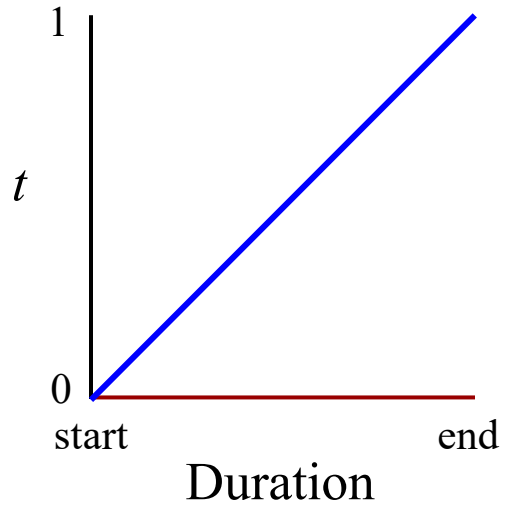
Tweening and Easing Functions

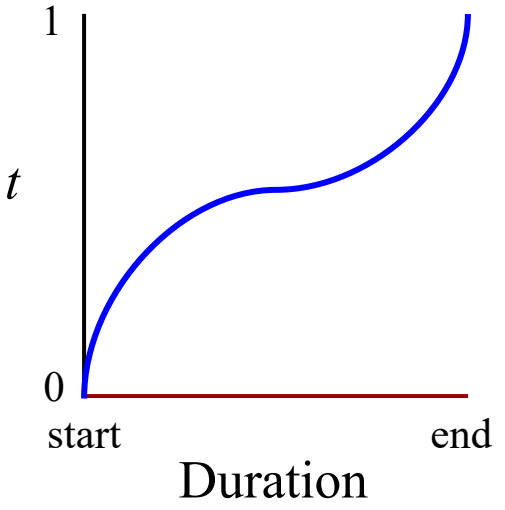
- Basic approach to tweening
  - Specify duration to animate
  - Set $t = 0$ at beginning
  - Normalize $t = 1$ at end
  - Interpolate value with t
- How does *t* change?
  - Usually done *linearly*
  - Could be some other way
- **Easing**: how to change *t*
  - Used for bouncing effects
  - Best used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



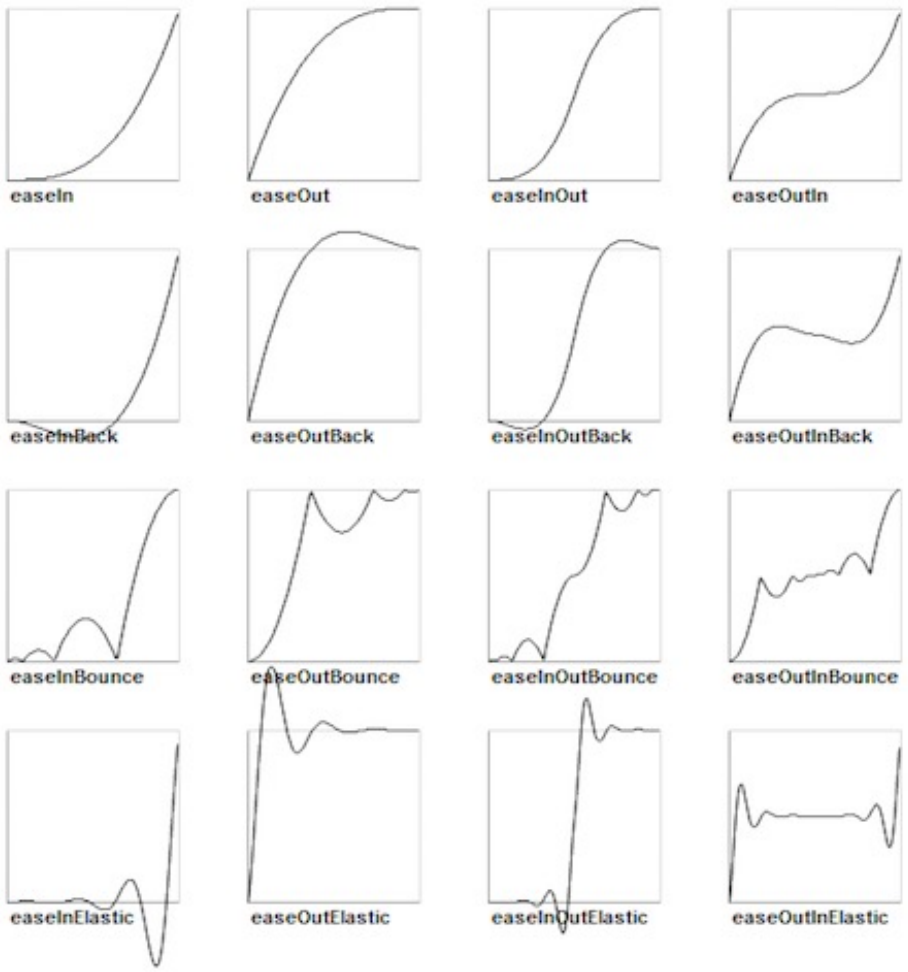
Tweening and Easing Functions

- 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*





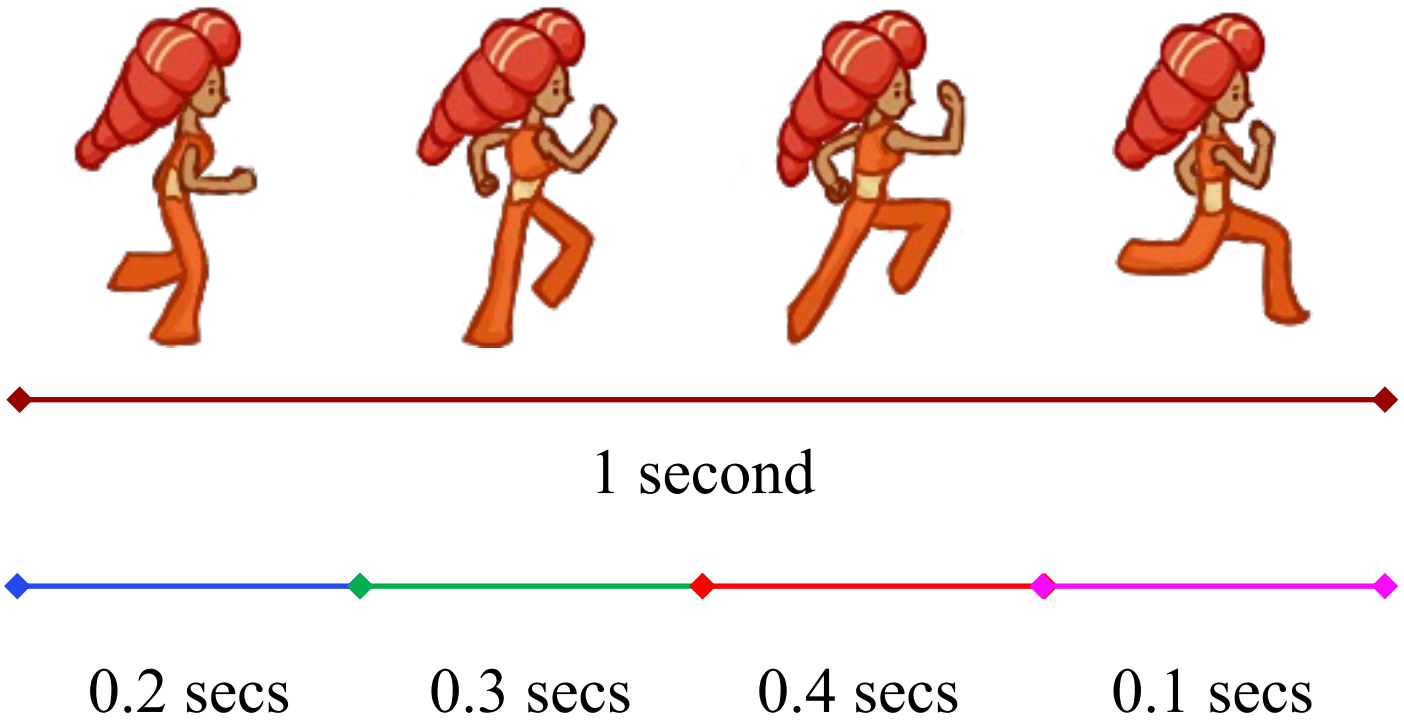
Classic Easing Functions



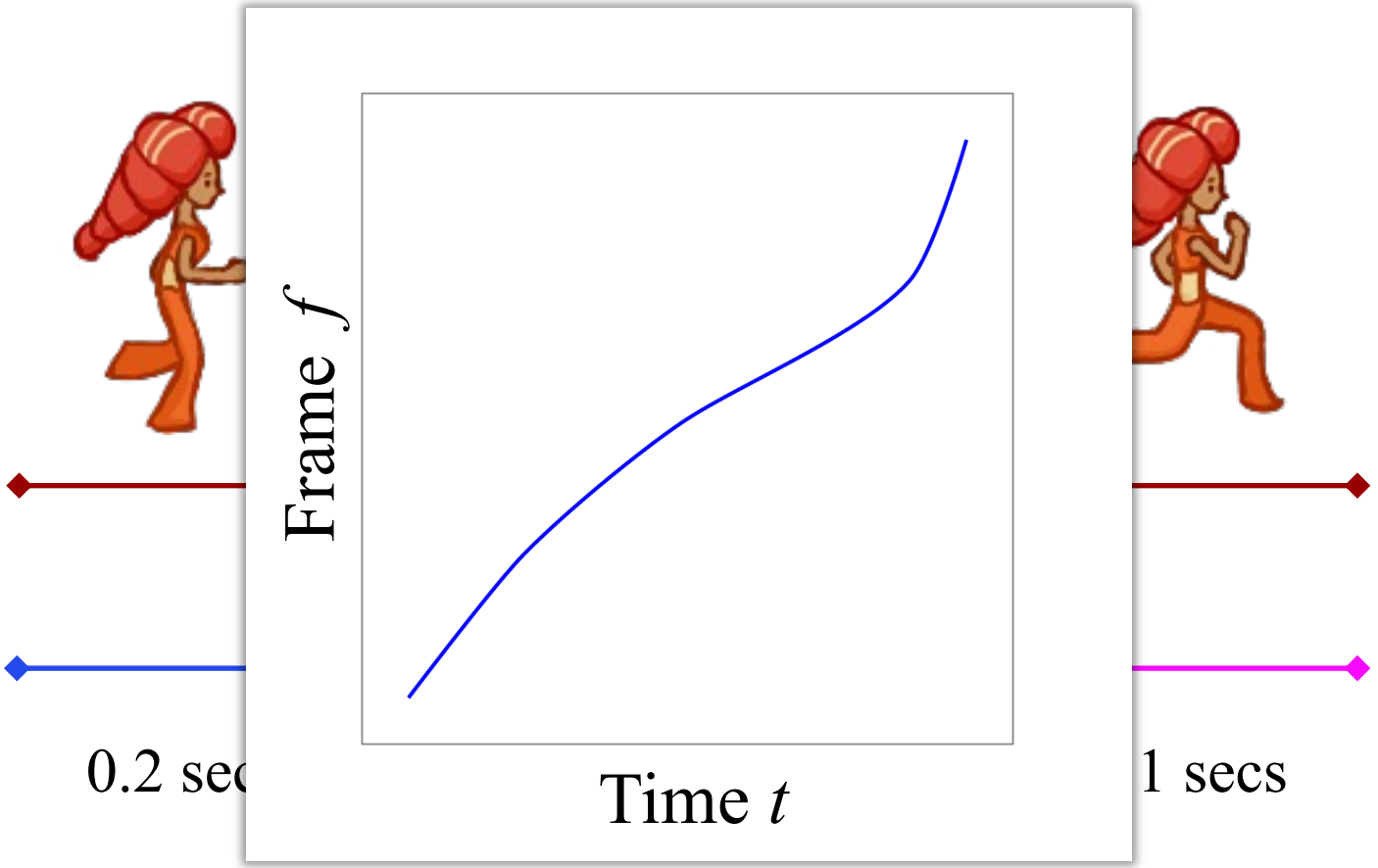
Classic Easing Functions

http://easings.net

Application to Sprite Animation

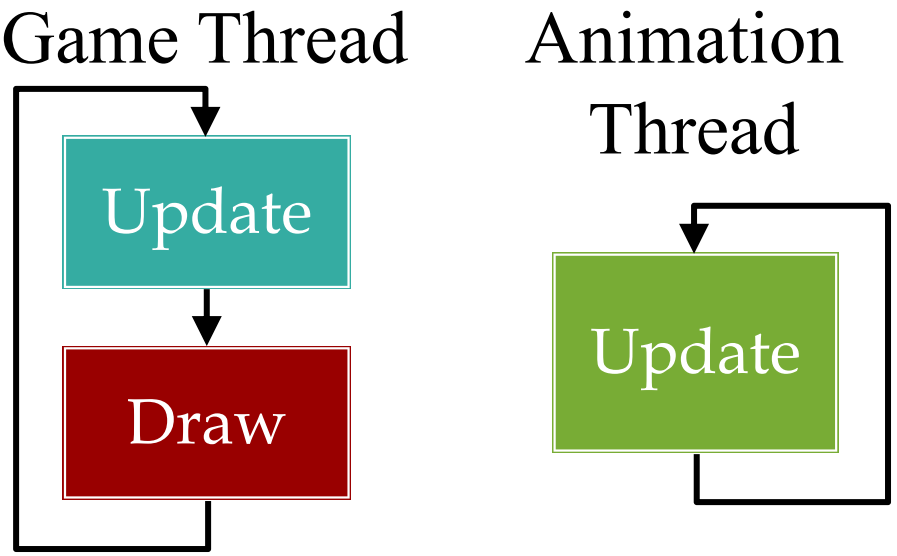


Application to Sprite Animation



Aside: Animating vs Rendering

- You do not have to animate in the main thread
  - Main thread is for rendering (drawing on screen)
  - But animation is simply "posing" your models
- Allows for smoother animation (VSync problem)



Example: *Hi Fi Rush*

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
  - **E.g.** *After Effects, Spine*
  - Great for visualizing design

Modular Animation

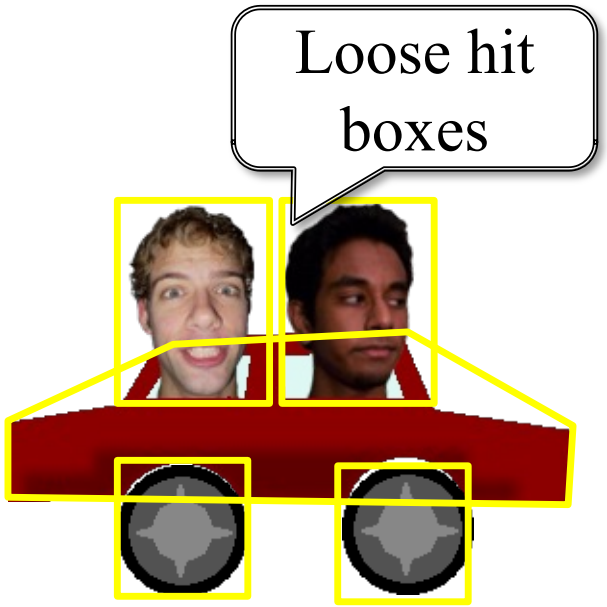
- Break asset into parts
  - Natural for joints/bodies
  - Animate each separately
- Cuts down on filmstrips
  - Most steps are transforms
  - Very natural for tweening
  - Also better for physics
- Several tools to help you
  - **E.g.** *After Effects, Spine*
  - Great for visualizing design

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
  - **E.g.** *After Effects, Spine*
  - Great for visualizing design

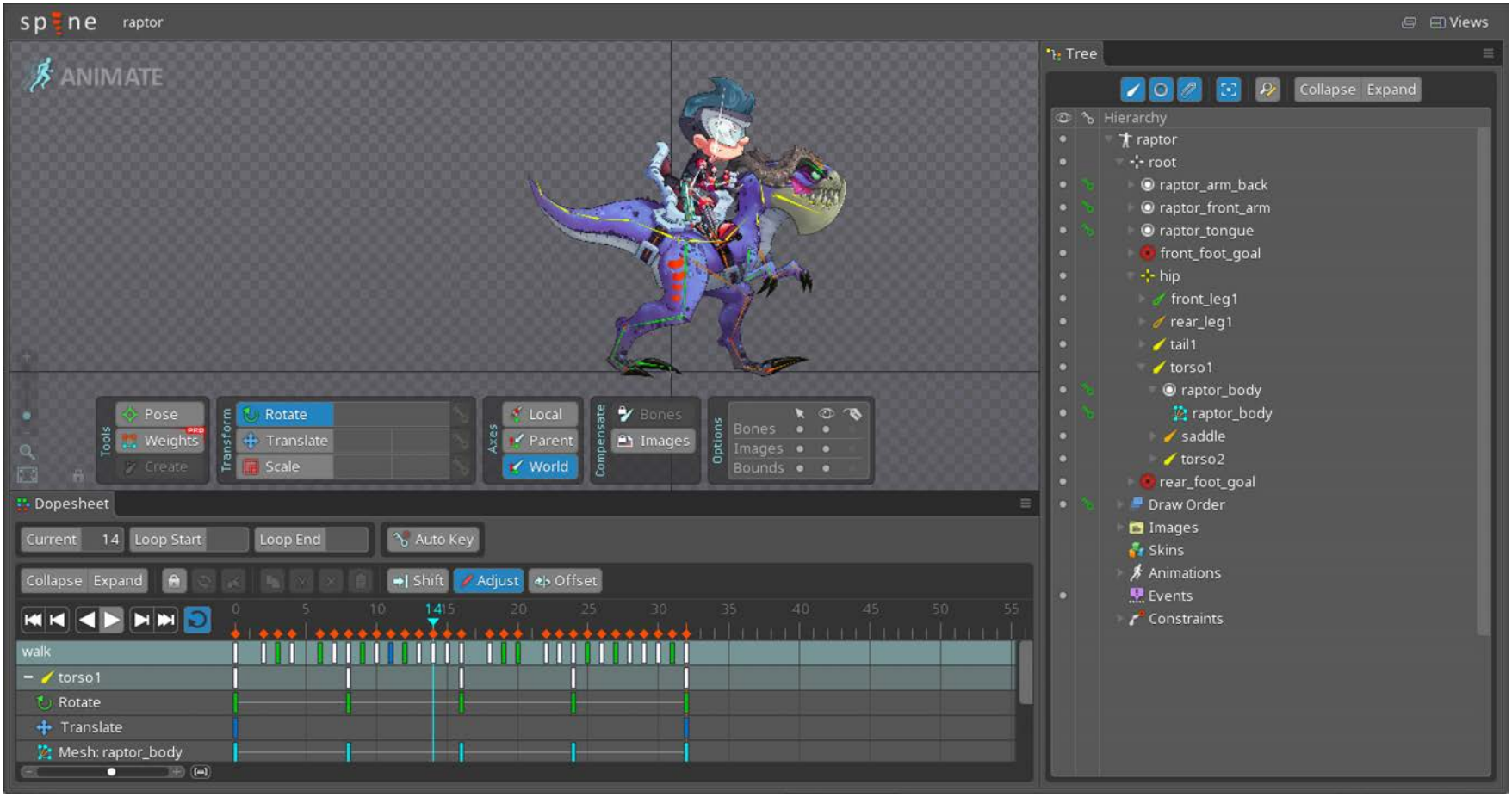
Modular Animation

- 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
  - **E.g.** *After Effects, Spine*
  - Great for visualizing design

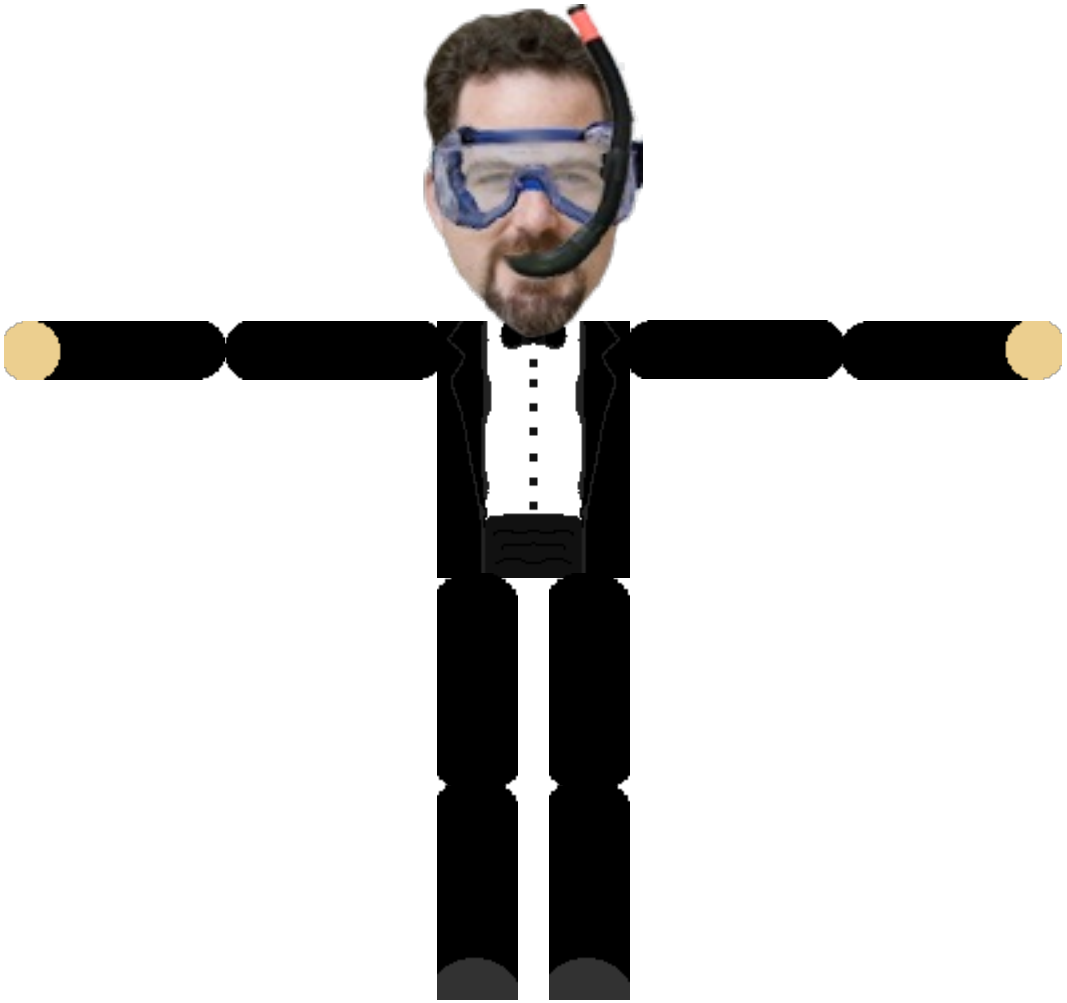


- Inside hit box can safely
  - Transform with duration
  - Tween animations
  - Manage multiple actions

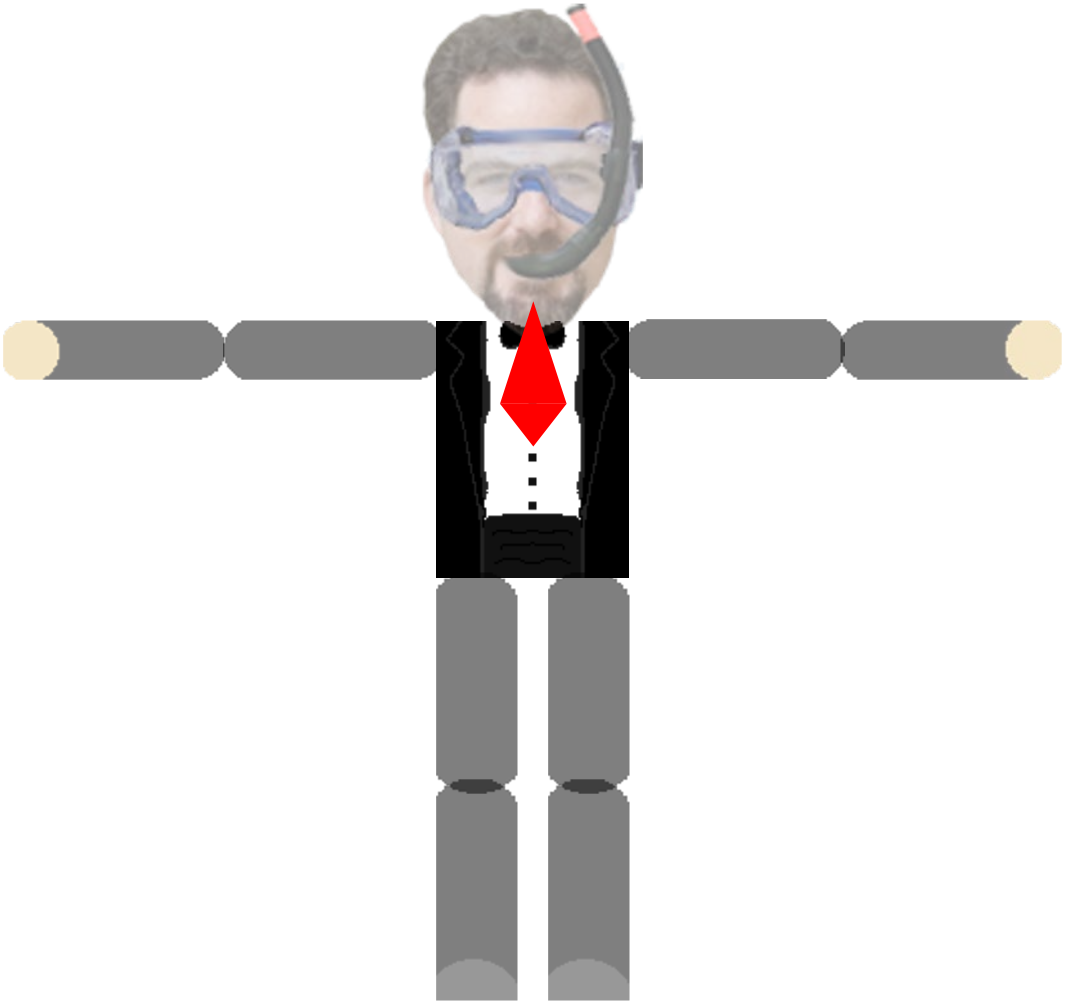
Spine Demo


spine
raptor
Views
ANIMATE
Tree
Collapse
Expand
Hierarchy
raptor
root
raptor_arm_back
raptor_front_arm
raptor_tongue
front_foot_goal
hip
front_leg1
rear_leg1
tail1
torso1
raptor_body
raptor_body
saddle
torso2
rear_foot_goal
Draw Order
Images
Skins
Animations
Events
Constraints
Tools
Pose
Weights
Create
Transform
Rotate
Translate
Scale
Axes
Local
Parent
World
Compensate
Bones
Images
Options
Bones
Images
Bounds
Dopesheet
Current
14
Loop Start
Loop End
Auto Key
Collapse
Expand
Shift
Adjust
Offset
walk
torso1
Rotate
Translate
Mesh: raptor_body

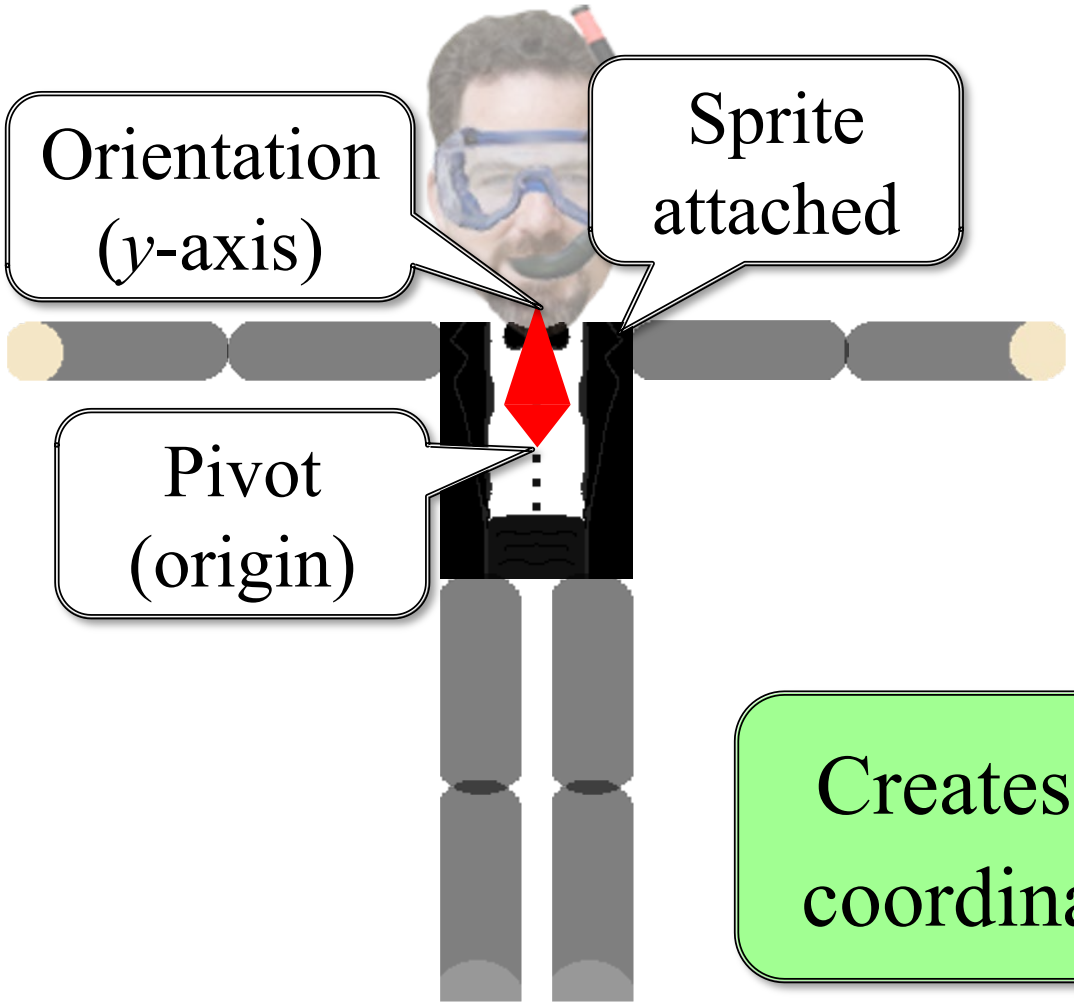
Basic Idea: **Bones**



Basic Idea: **Bones**

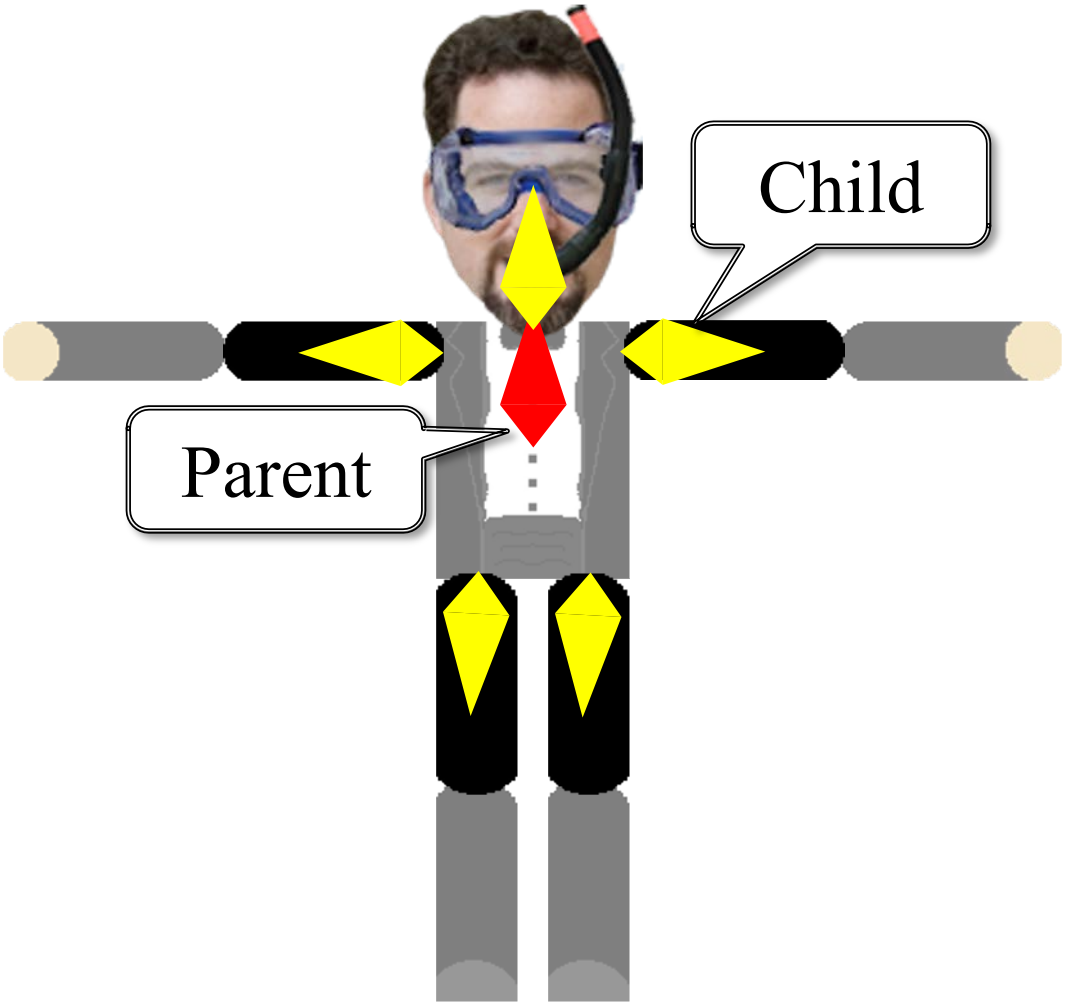


Basic Idea: **Bones**



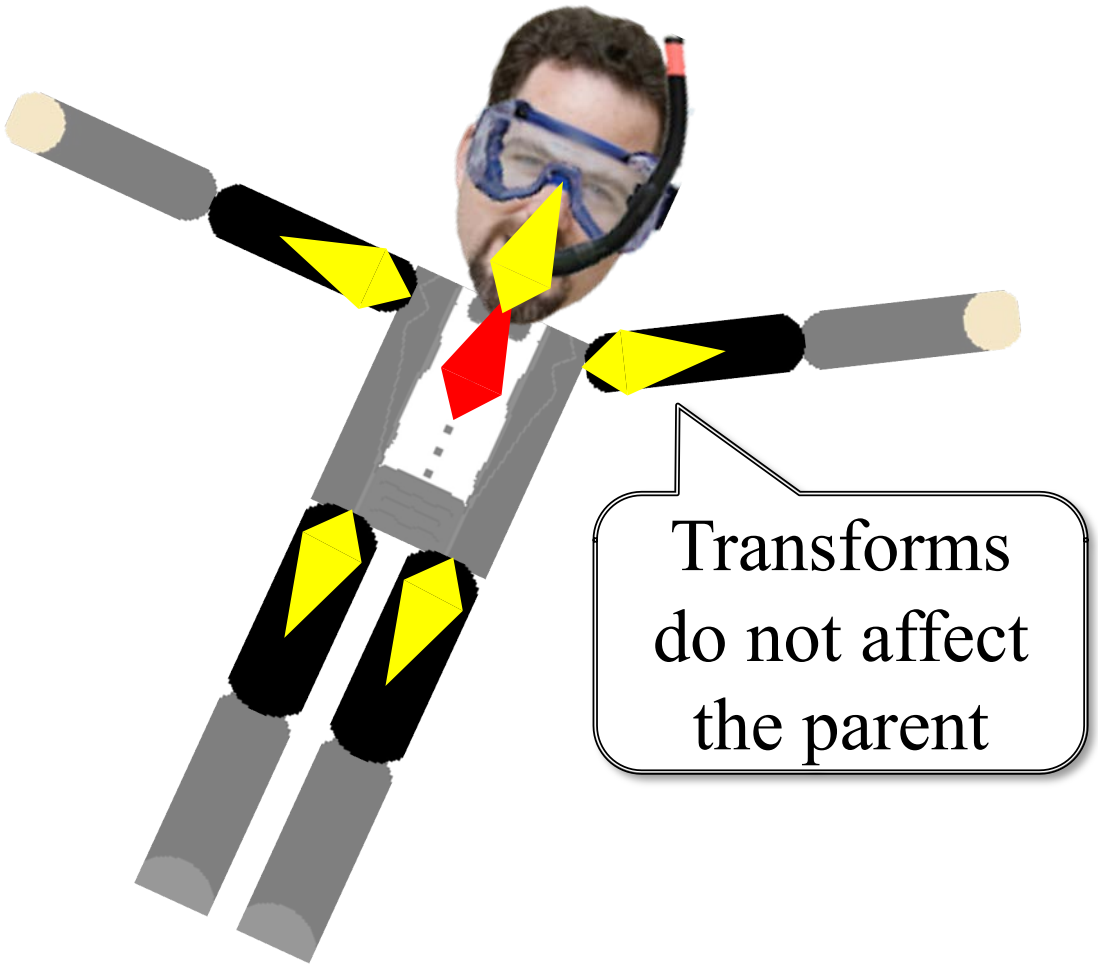
Creates implicit coordinate space

Bones are Heirarchical

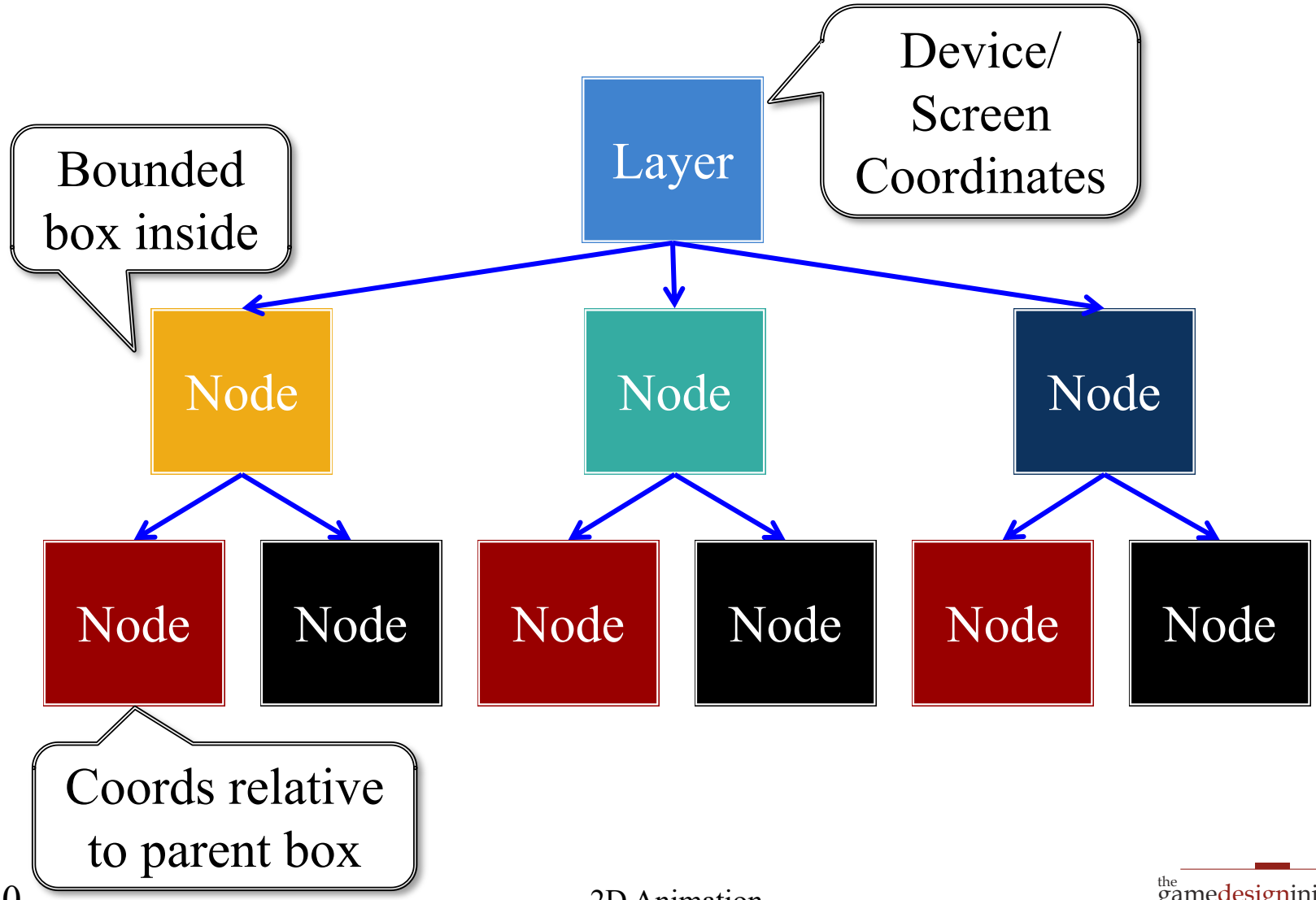


Bones are Heirarchical

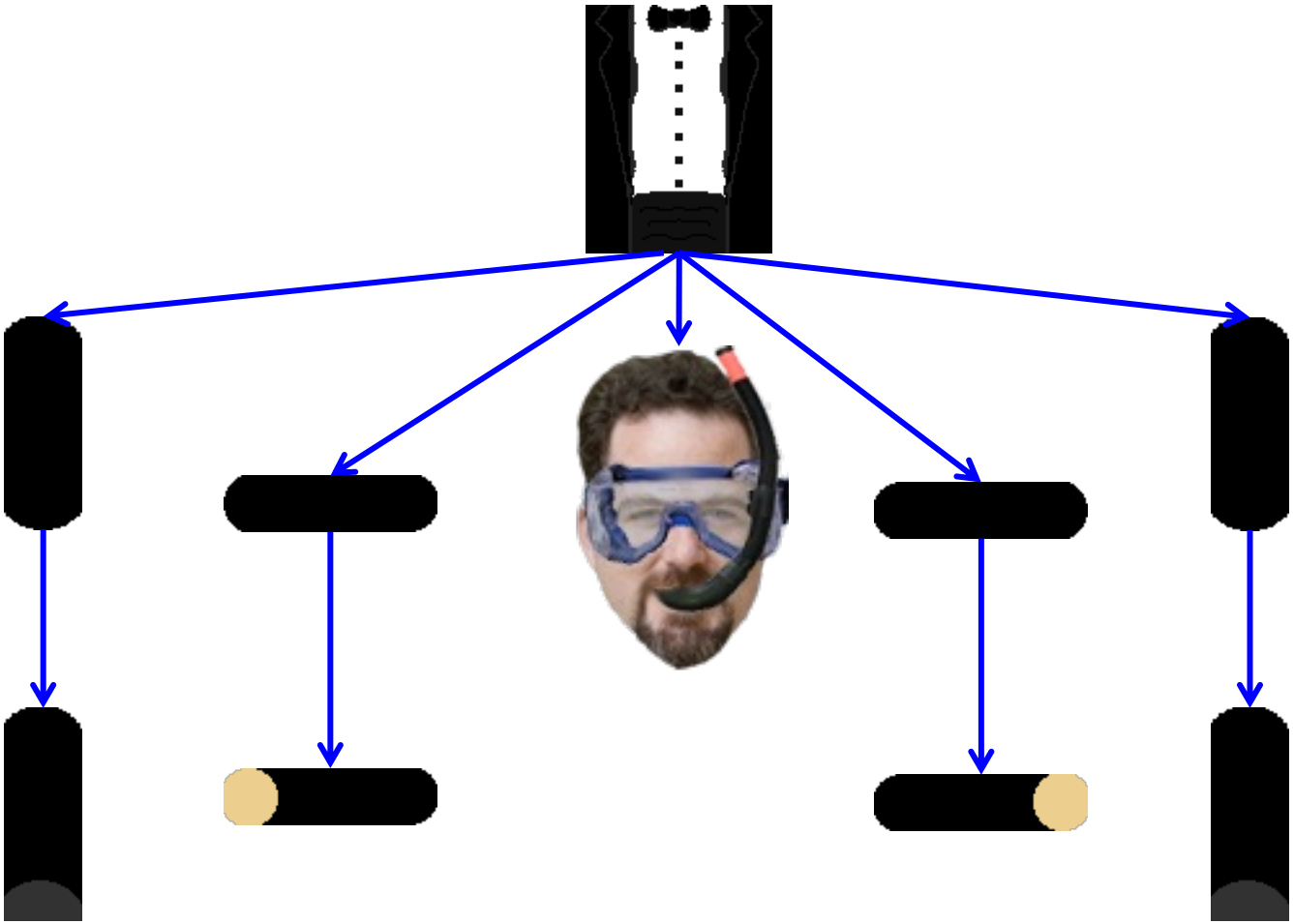
Bones are Heirarchical



Recall: Scene Graph Hierarchy

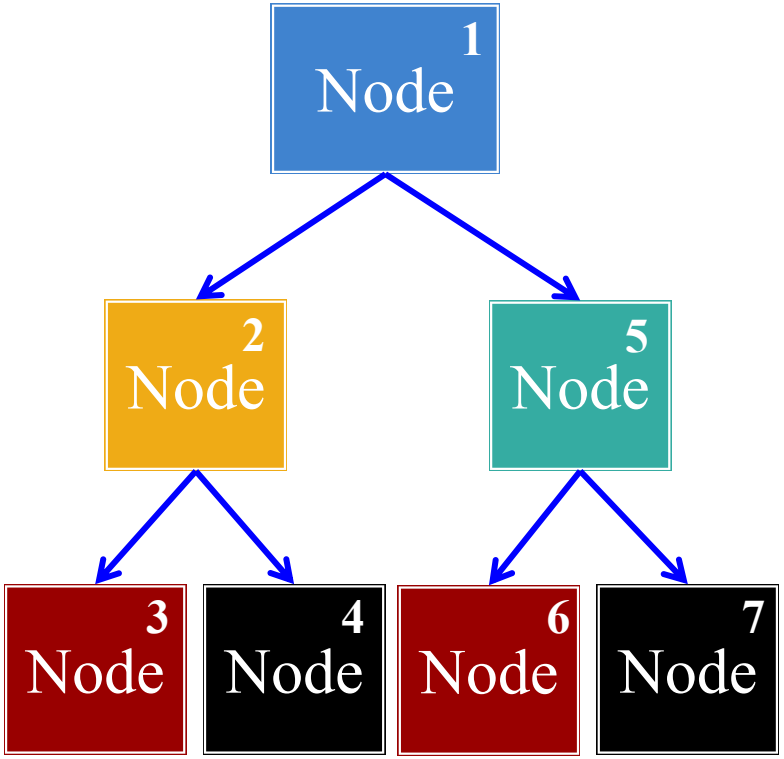


Bones are a Scene Graph Visualization



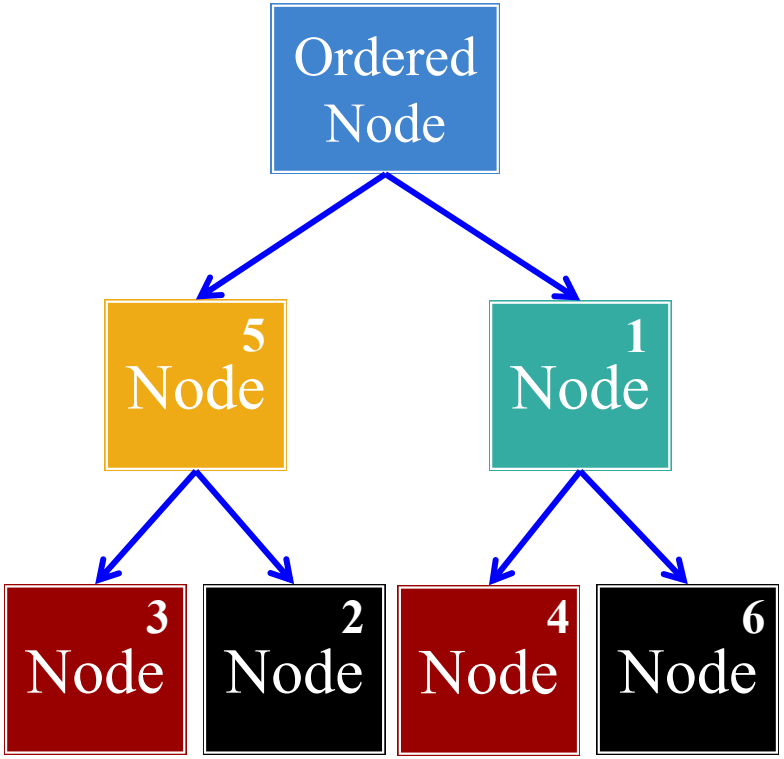
Problem: Scene Graphs are Preorder

- **Parents are drawn first**
  - Children are drawn in front
  - Ideal for UI elements
  - Bad for modular animation
- Solution: `OrderedNode`
  - Puts descendents into a list
  - Sorts based on priority value
  - Draws nodes in that order
- Acts as a render barrier
  - What if nested `OrderedNode`?
  - Each `OrderedNode` is a unit
  - Priorities do not mix



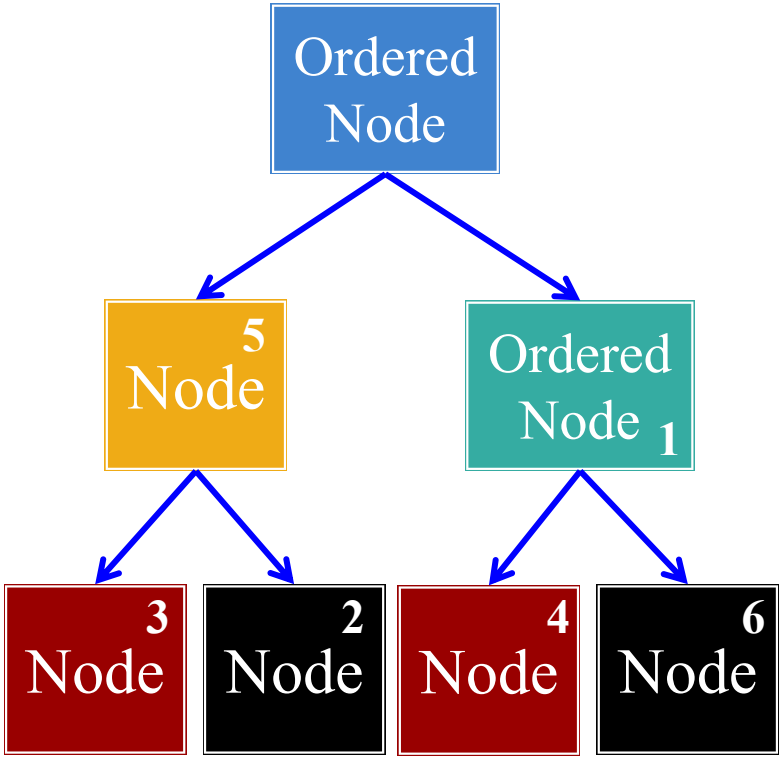
Problem: Scene Graphs are Preorder

- Parents are drawn first
  - Children are drawn in front
  - Ideal for UI elements
  - Bad for modular animation
- **Solution:** `OrderedNode`
  - Puts descendents into a list
  - Sorts based on priority value
  - Draws nodes in that order
- Acts as a render barrier
  - What if nested `OrderedNode`?
  - Each `OrderedNode` is a unit
  - Priorities do not mix



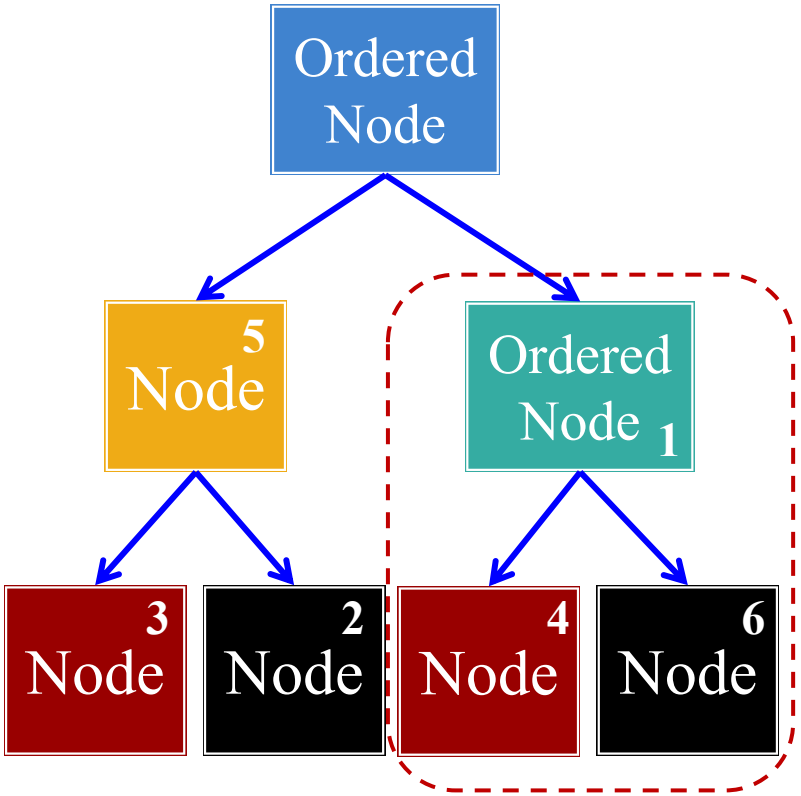
Problem: Scene Graphs are Preorder

- Parents are drawn first
  - Children are drawn in front
  - Ideal for UI elements
  - Bad for modular animation
- Solution: `OrderedNode`
  - Puts descendents into a list
  - Sorts based on priority value
  - Draws nodes in that order
- Acts as a **render barrier**
  - What if nested `OrderedNode`?
  - Each `OrderedNode` is a unit
  - Priorities do not mix



Problem: Scene Graphs are Preorder

- Parents are drawn first
  - Children are drawn in front
  - Ideal for UI elements
  - Bad for modular animation
- Solution: `OrderedNode`
  - Puts descendents into a list
  - Sorts based on priority value
  - Draws nodes in that order
- Acts as a **render barrier**
  - What if nested `OrderedNode`?
  - Each `OrderedNode` is a unit
  - Priorities do not mix



Problems With Tweening

Transform Tweening

Physical Animation

Complete Disaster

Problems With Tweening

```
auto seq = RotateBy::alloc(90.0f);

auto action – seq->attach(sprite);

mgr->activate(key,action,2.0f);
```

What if we change our mind before 2 seconds?

Problems With Tweening

```
auto seq = RotateBy::alloc(90.0f);

auto action – seq->attach(sprite);

mgr->activate(key,action,2.0f);
```

Compatible: Combine
Incompatible: Replace

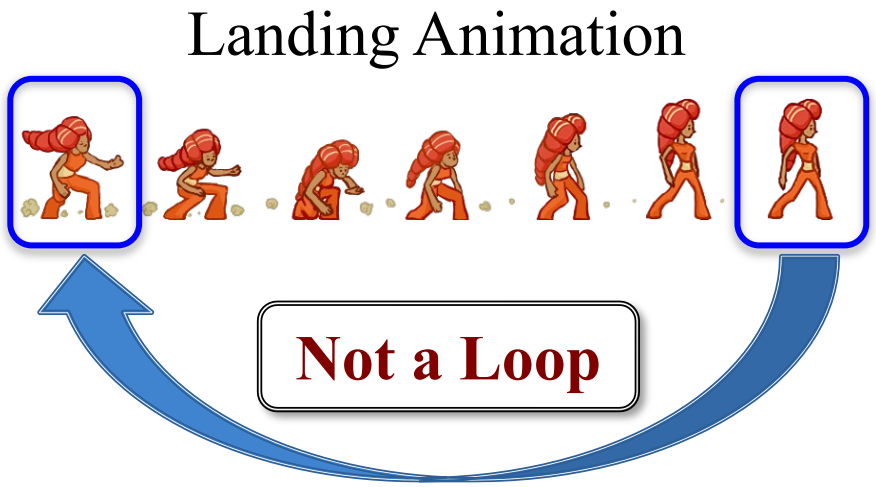
Combining Animations

Landing Animation

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
  - End of loop = start of another
  - Treat like advancing a frame

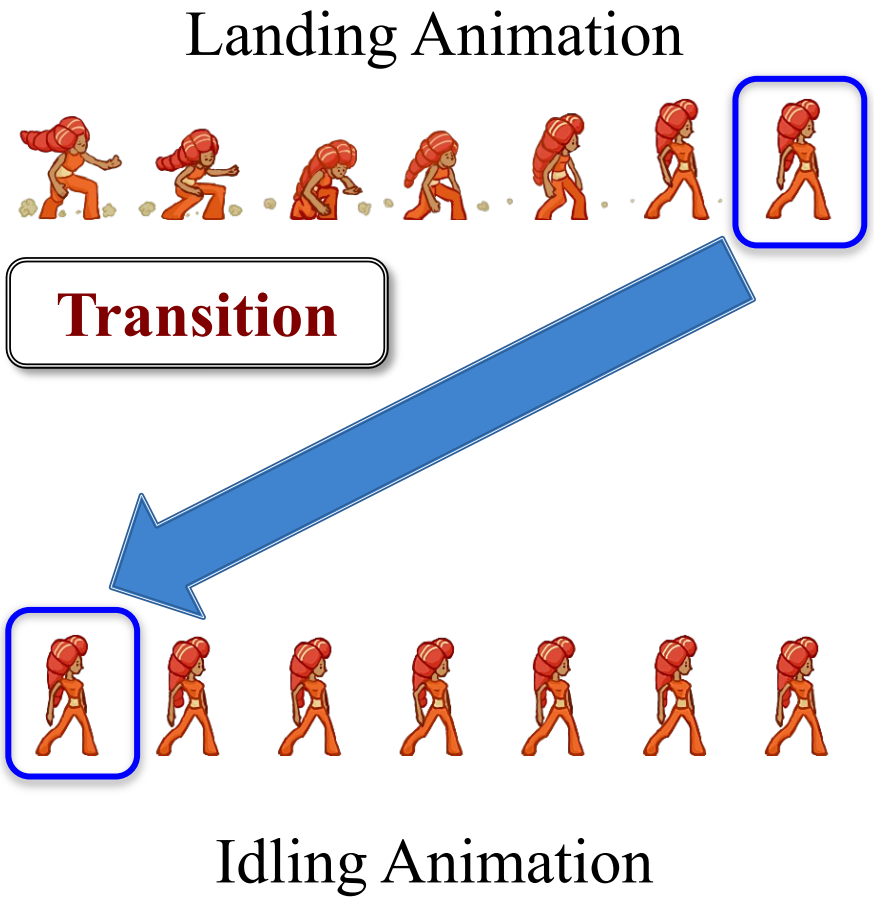
Combining Animations



Idling Animation

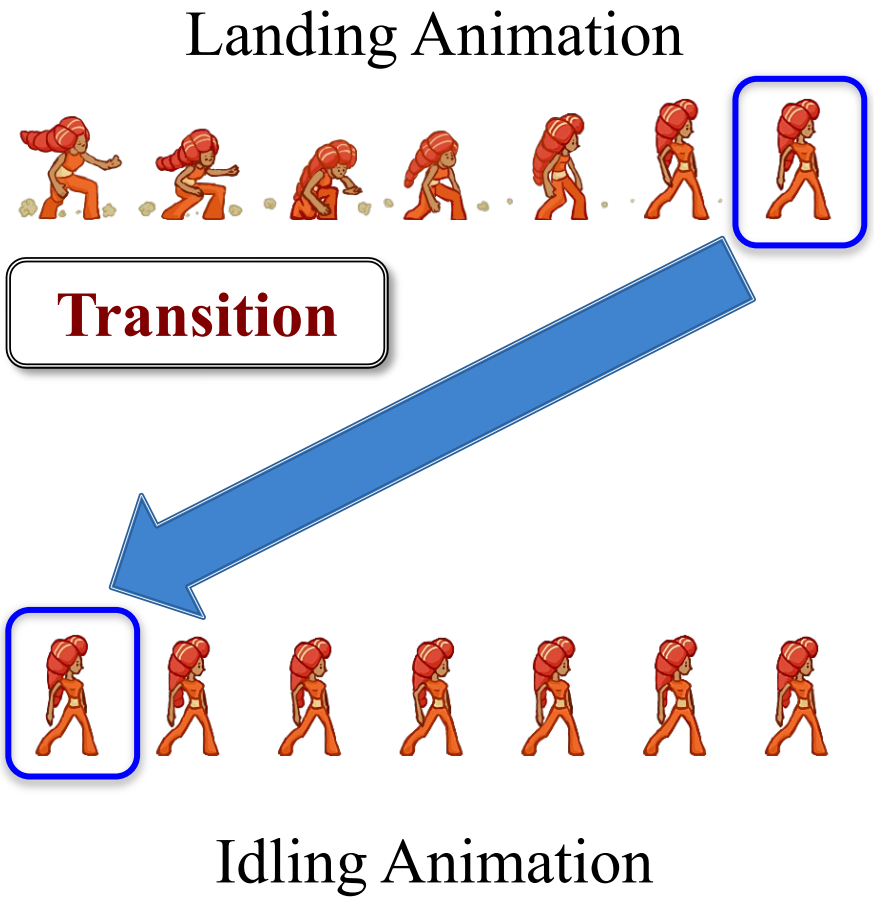
- Characters do a lot of things
  - Run, jump, duck, slide
  - Fire weapons, cast spells
  - Fidget while player AFK
- Want animations for all
  - Is loop appropriate for each?
  - How do we transition?
- **Idea**: shared boundaries
  - End of loop = start of another
  - Treat like advancing a frame

Combining Animations



- 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
  - End of loop = start of another
  - Treat like advancing a frame

Combining Animations

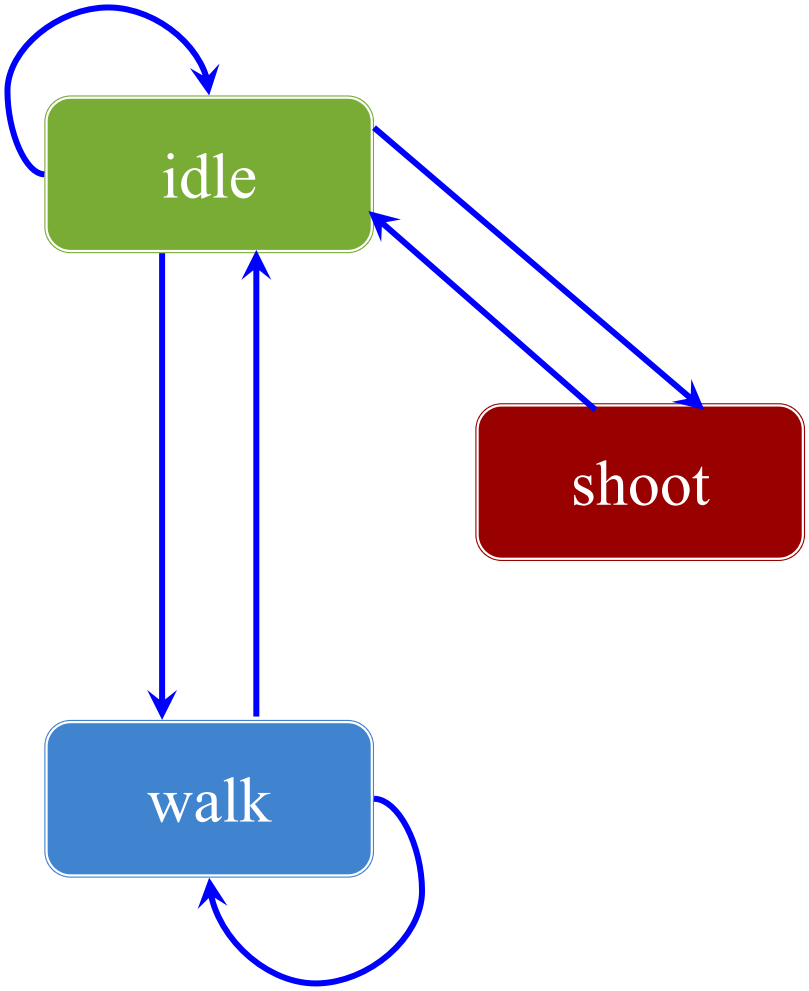


- Characters do a lot of things
  - Run, jump, duck, slide
  - Fire weapons, cast spells
  - Fidget while player AFK
- Want animations for all
  - Is loop appropriate for each?
  - How do we transition?
- **Idea**: shared boundaries
  - H…er
  - T…

But do not draw ends twice!

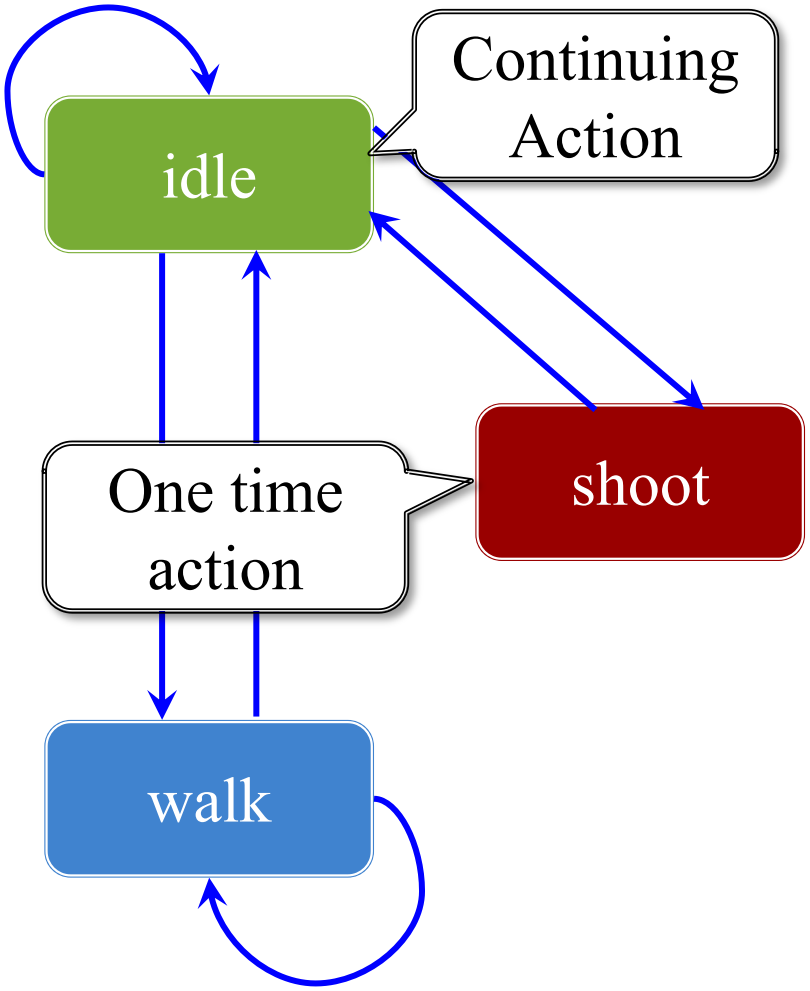
Animation and State Machines

- 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

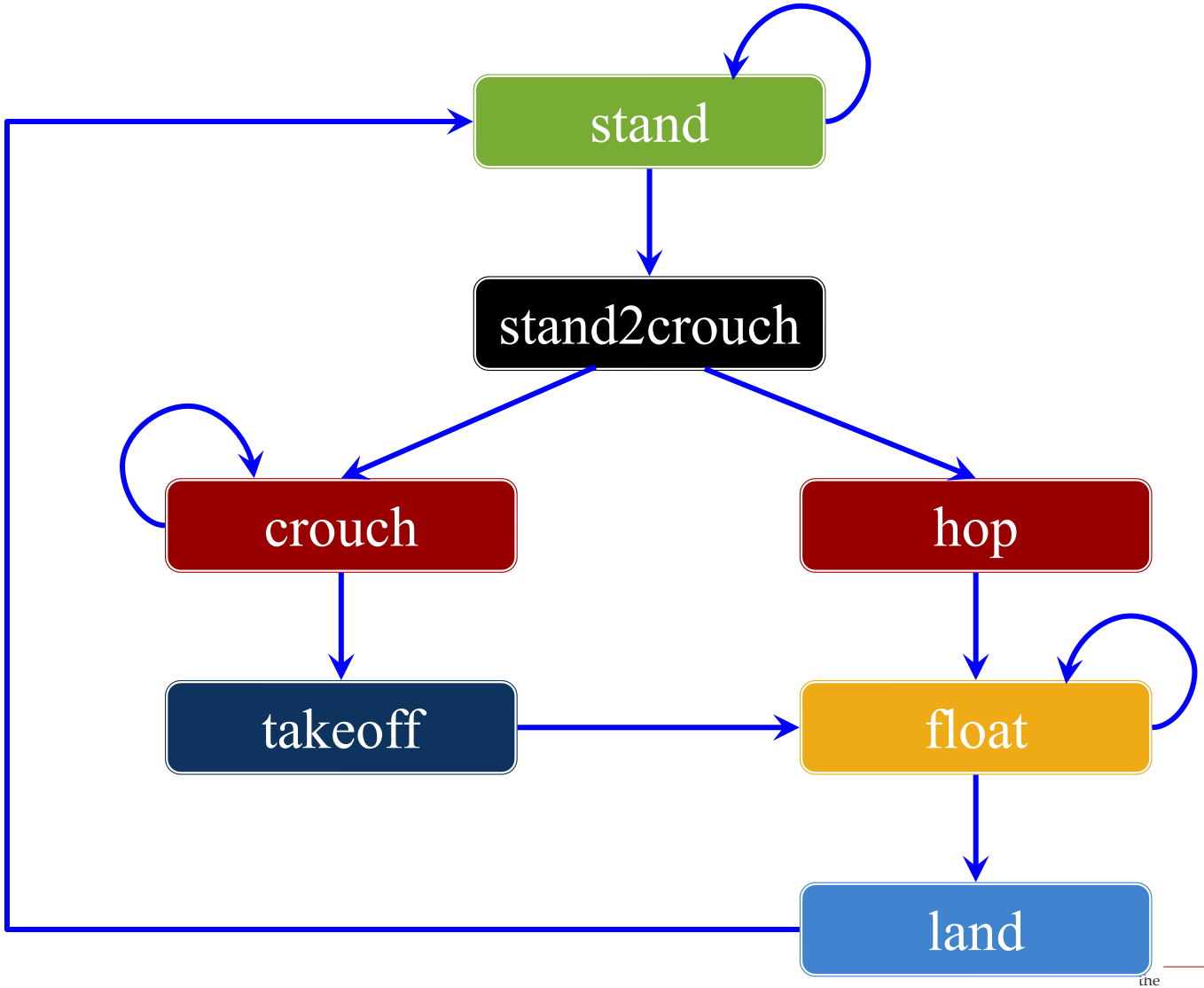


Animation and State Machines

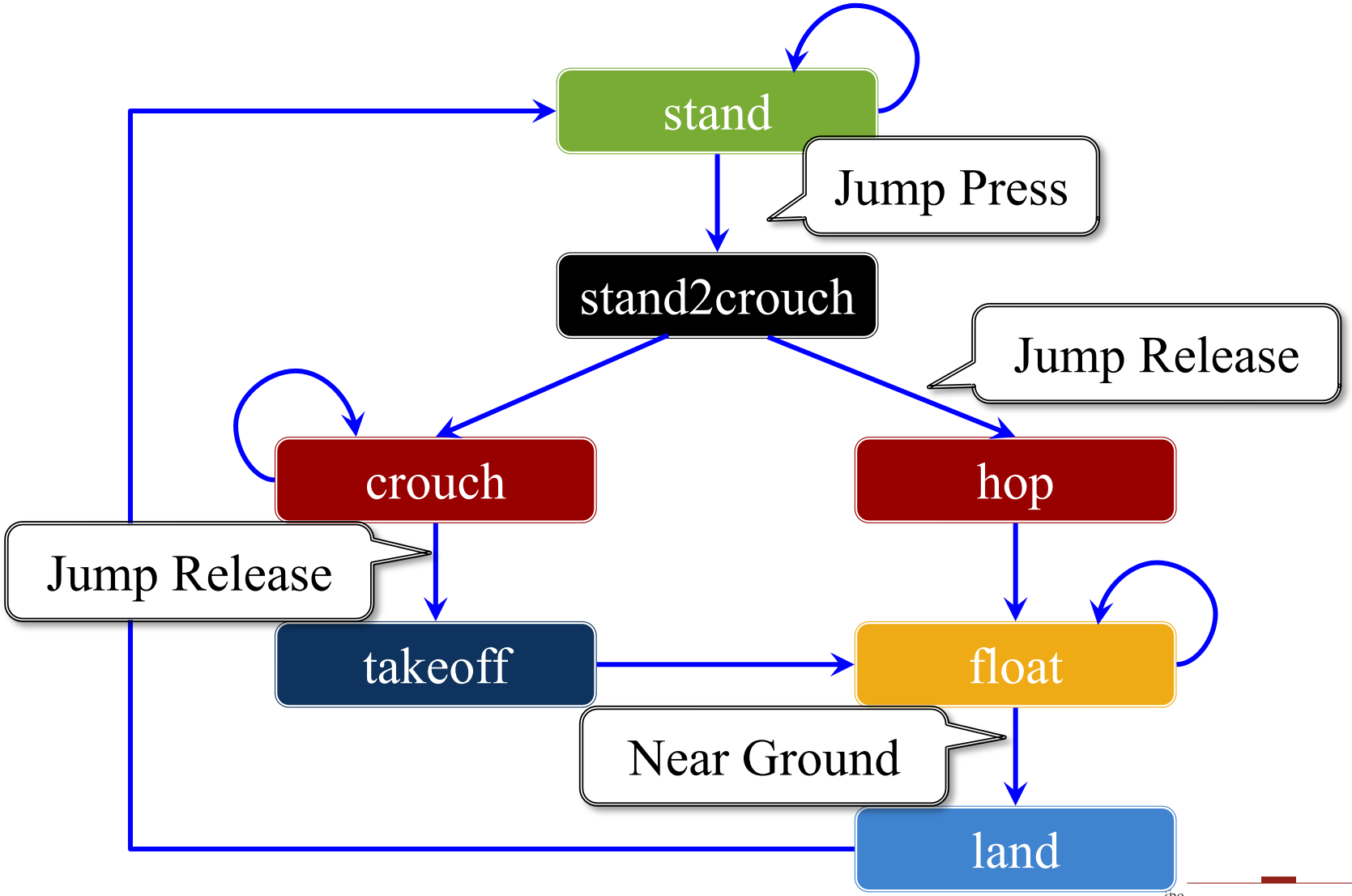
- Idea: Each sequence a state
  - Do sequence while in state
  - Transition when at end
  - Only loop if loop in graph
- A graph edge means…
  - Boundaries match up
  - Transition is allowable
- Similar to data driven AI
  - Created by the designer
  - Implemented by programmer
  - Modern engines have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



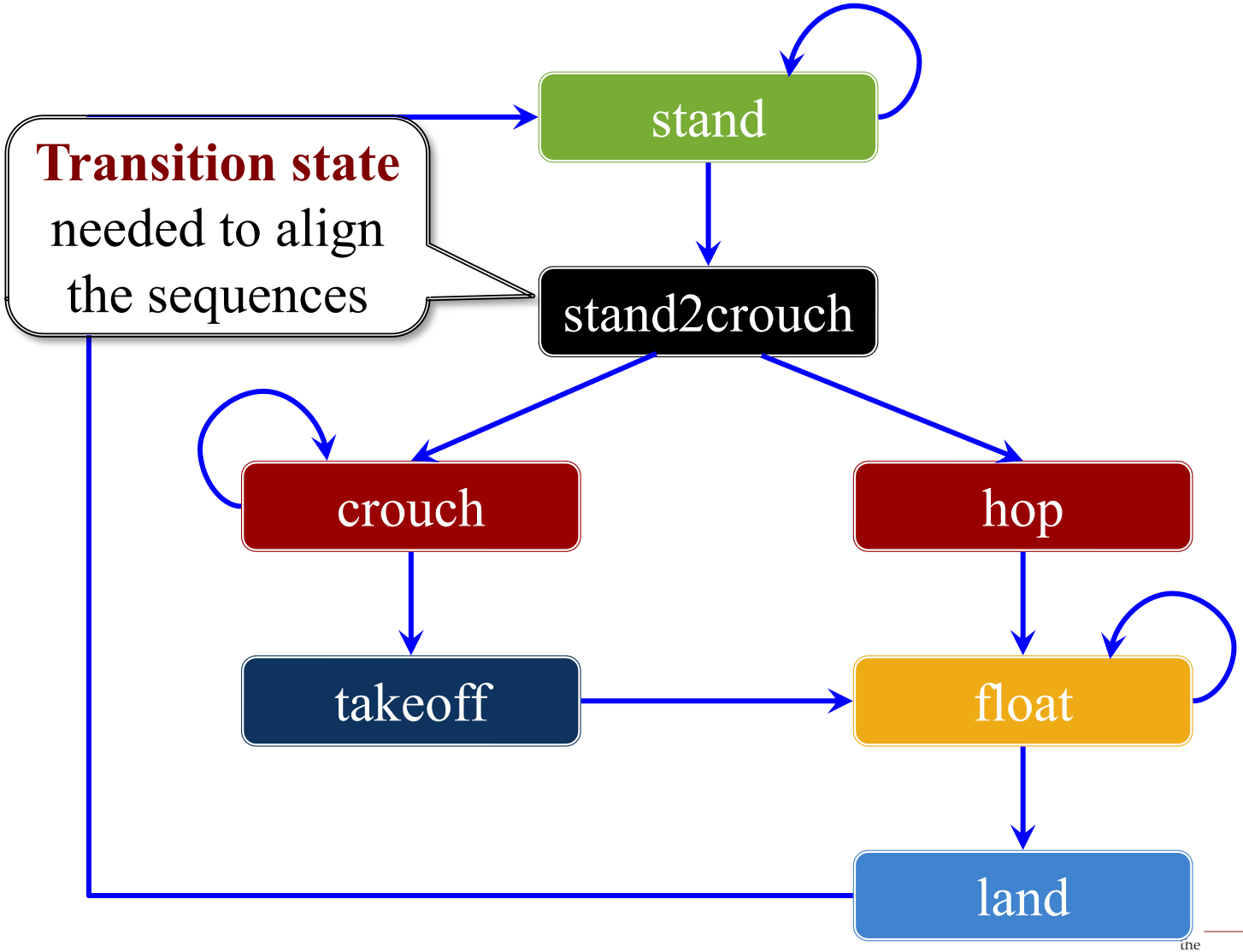
Complex Example: Jumping



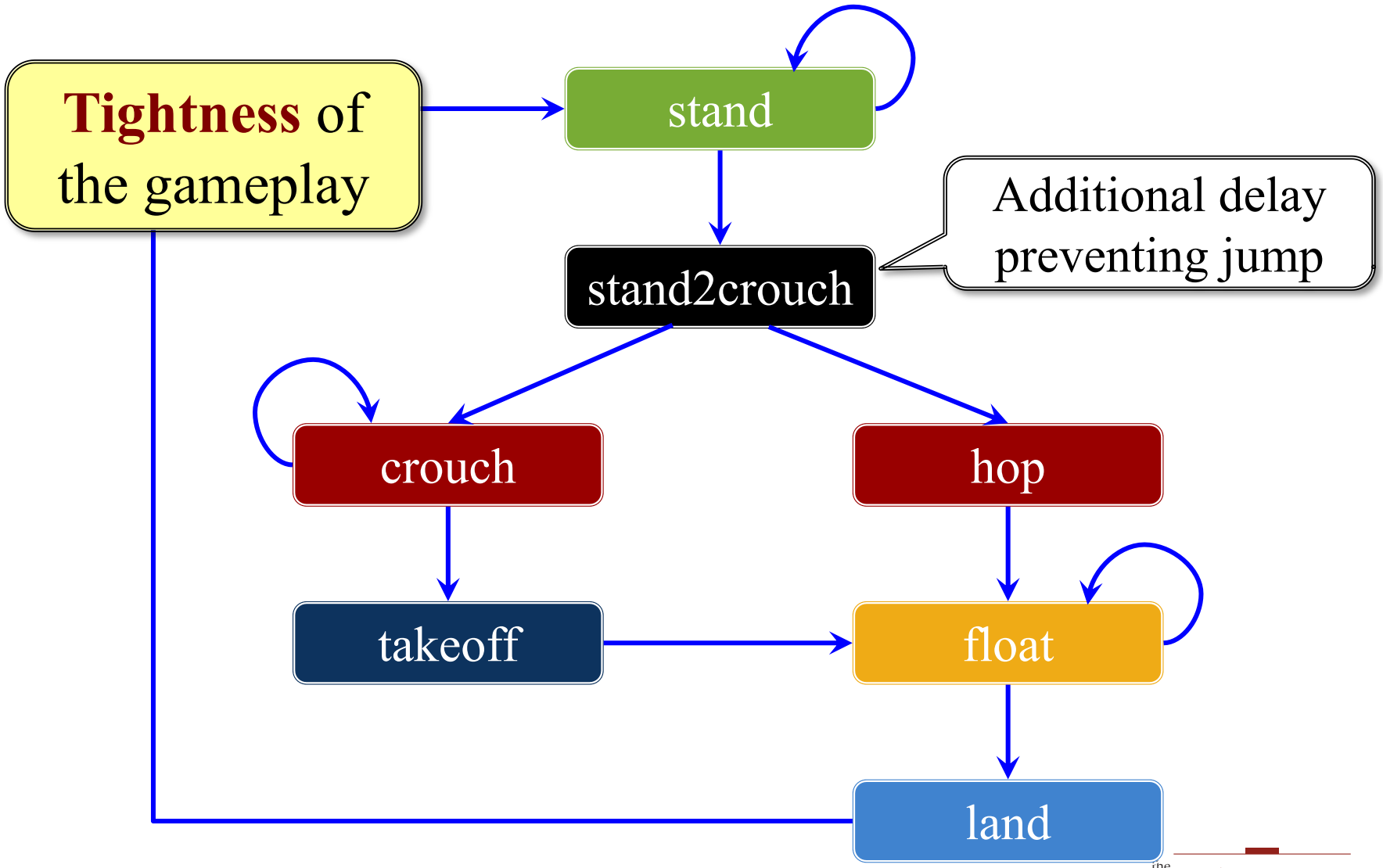
Complex Example: Jumping



Complex Example: Jumping



The Responsiveness Issue



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