the gamedesigninitiative at cornell university

Lecture 20

Character Behavior

Classical AI vs. Game AI

- **Classical**: Design of *intelligent agents*
  - Perceives environment, maximizes its success
  - Established area of computer science
  - Subtopics: planning, machine learning
- **Game**: Design of *rational behavior*
  - Does not need to optimize (and often will not)
  - Often about "scripting" a personality
  - More akin to cognitive science

Take Away for This Lecture

- Review the **sense-think-act** cycle
  - How do we separate actions and thinking?
  - Delay the sensing problem to next time
- What is **rule-based** character AI?
  - How does it relate to sense-think-act?
  - What are its advantages and disadvantages?
- What **alternatives** are there to rule-based AI?
  - What is our motivation for using them?
  - How do they affect the game architecture?

Role of AI in Games

- **Autonomous Characters** (NPCs)
  - Mimics the "personality" of the character
  - May be opponent or support character
- **Strategic Opponents**
  - AI at the "player level"
  - Closest to classical AI
- **Character Dialog**
  - Intelligent commentary
  - Narrative management (e.g. Façade)

Role of AI in Games

- **Autonomous Characters** (NPCs)
  - Mimics the "personality" of the character
  - May be opponent or support character
- **Strategic Opponents**
  - AI at the "player level"
  - Closest to classical AI
- **Character Dialog**
  - Intelligent commentary
  - Narrative management (e.g. Façade)

Review: Sense-Think-Act

- **Sense**:
  - Perceive the world
  - Reading the game state
  - **Example**: enemy near?
- **Think**:
  - Choose an action
  - Often merged with sense
  - **Example**: fight or flee
- **Act**:
  - Update the state
  - Simple and fast
  - **Example**: reduce health

S-T-A: Separation of Logic

- **Loops** = sensing
  - Read other objects
  - *Aggregate* for thinking
  - **Example**: nearest enemy
- **Conditionals** = thinking
  - Use results of sensing
  - Switch between possibilities
  - **Example**: attack or flee
- **Assignments** = actions
  - Rarely need loops
  - Avoid conditionals

```
move(int direction) {
  switch (direction) {
  case NORTH:
    y -= 1;
    break;
  case EAST:
    x += 1;
    break;
  case SOUTH:
    y += 1;
    break;
  case WEST:
    x -= 1;
    break;
  }
}
```

S-T-A: Separation of Logic

- **Loops** = sensing
  - Read other objects
  - *Aggregate* for thinking
  - **Example**: nearest enemy
- **Conditionals** = thinking
  - Use results of sensing
  - Switch between possibilities
  - **Example**: attack or flee
- **Assignments** = actions
  - Rarely need loops
  - Avoid conditionals

```
move(int direction) {
  switch (direction) {
  case NORTH:
    y -= 1;
    break;
  case EAST:
    x += 1;
    break;
  case SOUTH:
    y += 1;
    break;
  case WEST:
    x -= 1;
    break;
  }
}
```

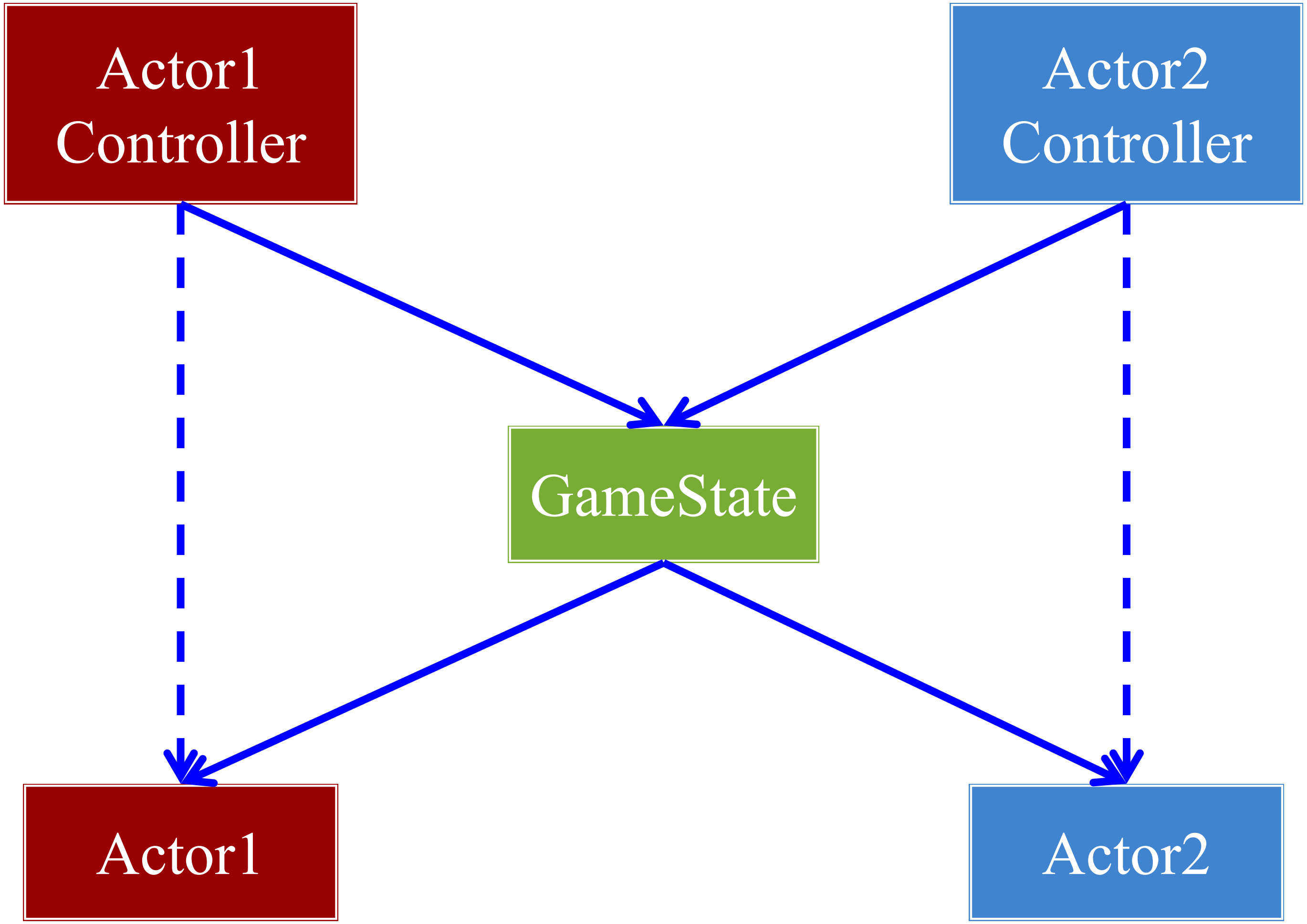
S-T-A: Separation of Logic

- **Loops** = sensing
  - Read other objects
  - *Aggregate* for thinking
  - **Example**: nearest enemy
- **Conditionals** = thinking
  - Use results of sensing
  - Switch between possibilities
  - **Example**: attack or flee
- **Assignments** = actions
  - Rarely need loops
  - Avoid conditionals

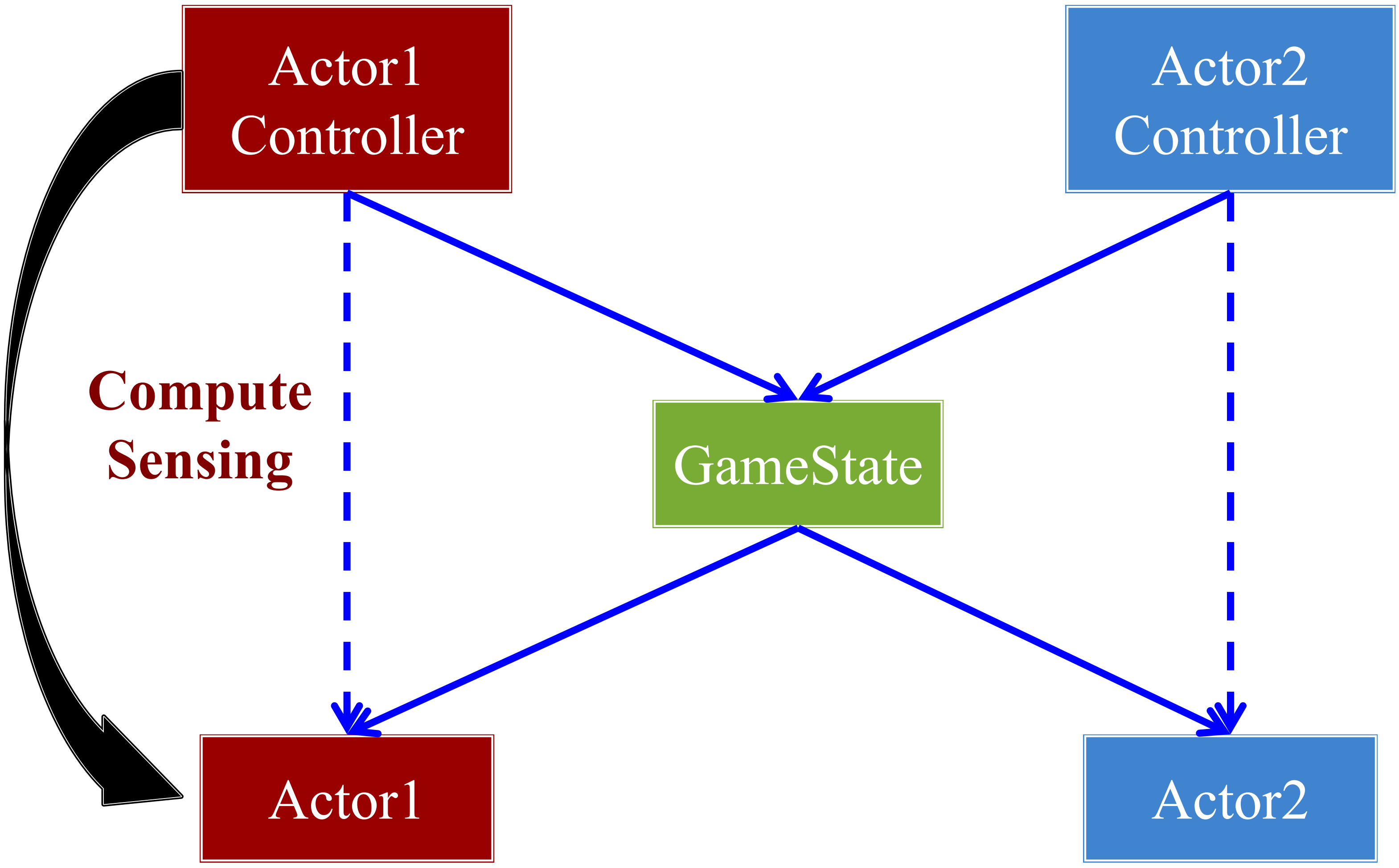
```
move(int direction) {
  switch (direction) {
  case NORTH:
    y -= 1;
    break;
  case EAST:
  ...
  case WEST:
    x -= 1;
    break;
  }
}
```

```
move(int dx, int dy) {
    x += dx;
    y += dy;
}
```

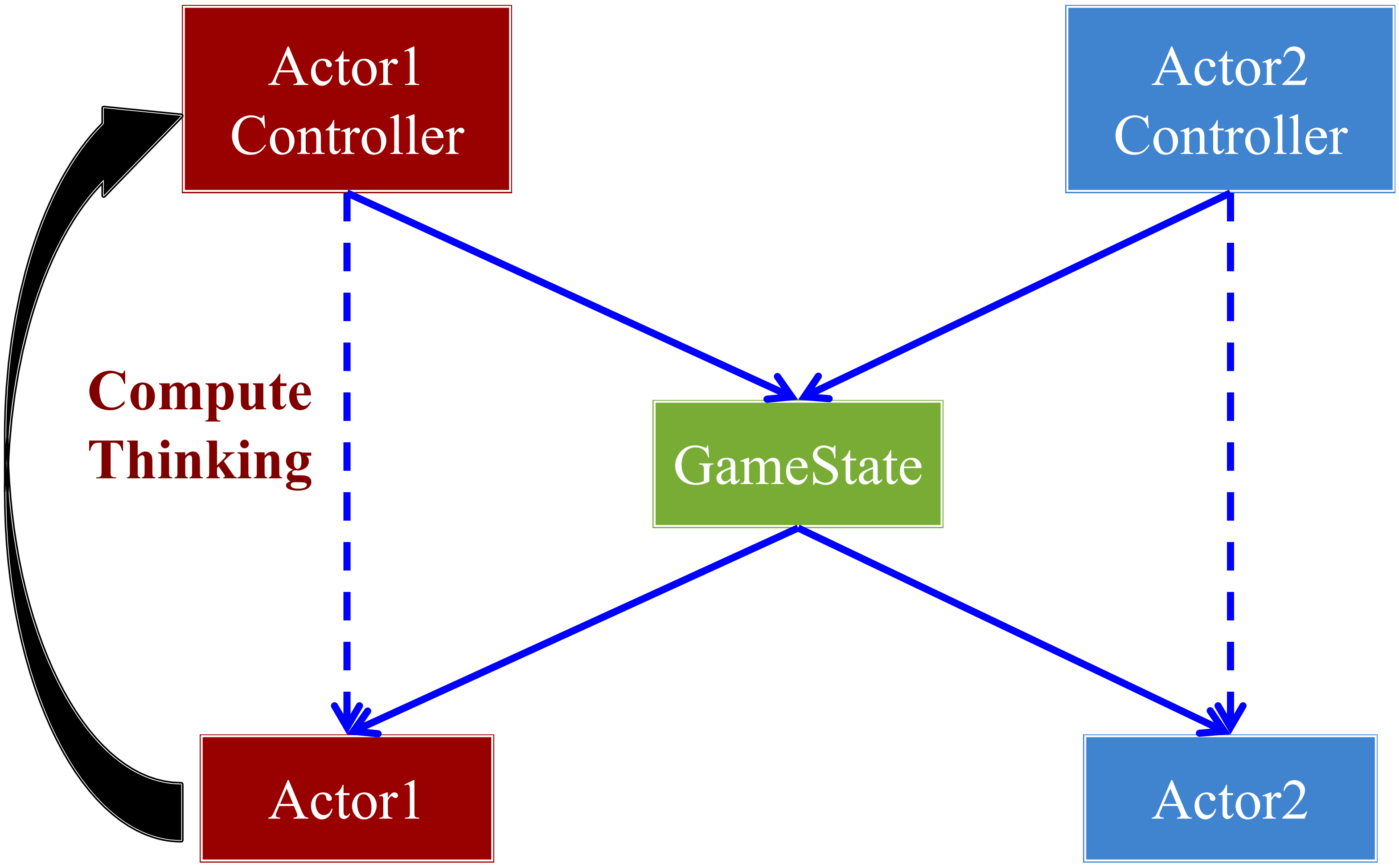
S-T-A: Reducing Dependencies



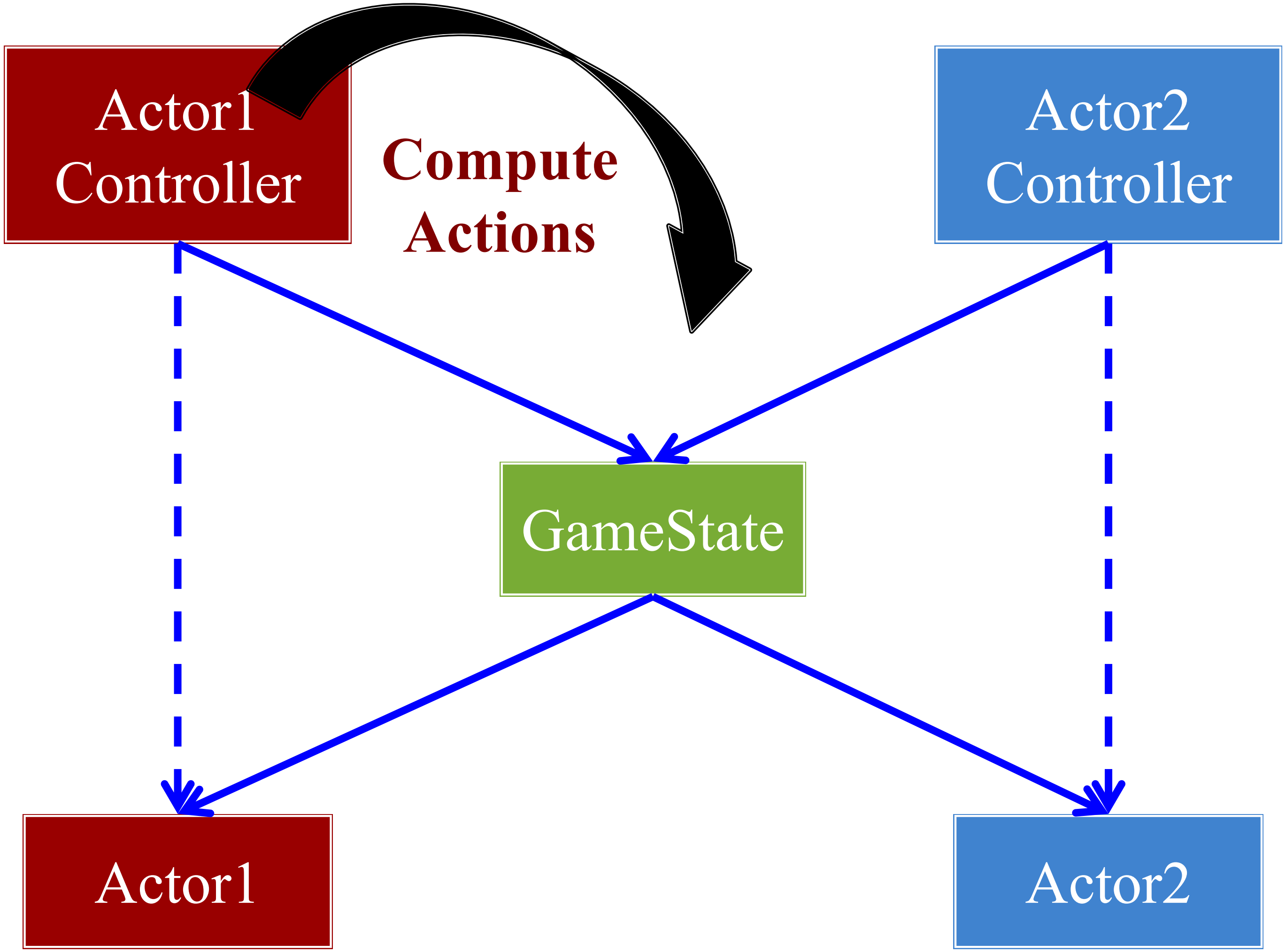
S-T-A: Reducing Dependencies



S-T-A: Reducing Dependencies



S-T-A: Reducing Dependencies



Review: Sense-Think-Act

- **Sense**:
  - Perceive the world
  - Reading the game state
  - **Example**: enemy near?
- **Think**:
  - Choose an action
  - Often merged with sense
  - **Example**: fight or flee
- **Act**:
  - Update the state
  - Simple and fast
  - **Example**: reduce health

Actions: Short and Simple

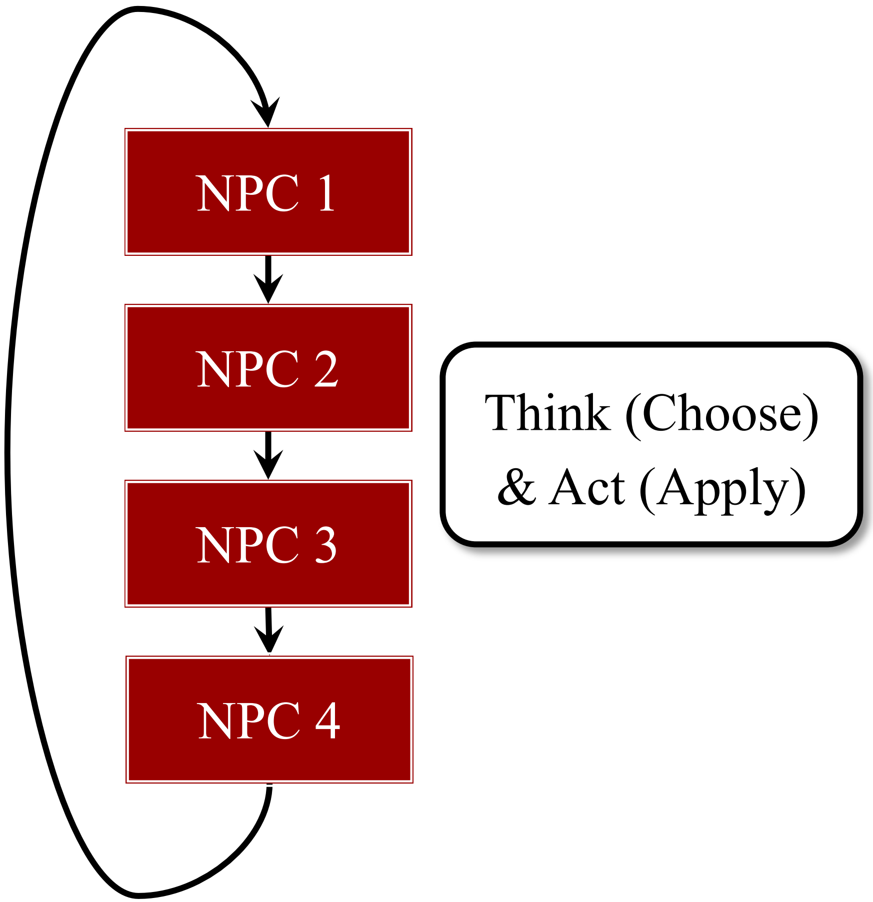
- Mainly use **assignments**
  - Avoid loops, conditionals
  - Similar to getters/setters
  - Complex code in thinking
- Helps with **serializability**
  - Record and undo actions
- Helps with **networking**
  - Keep doing last action
  - Recall: *dead reckoning*

```
move(int direction) {
  switch (direction) {
  case NORTH:
    y -= 1;
    break;
  case EAST:
  ...
  case WEST:
    x -= 1;
    break;
  }
}
```

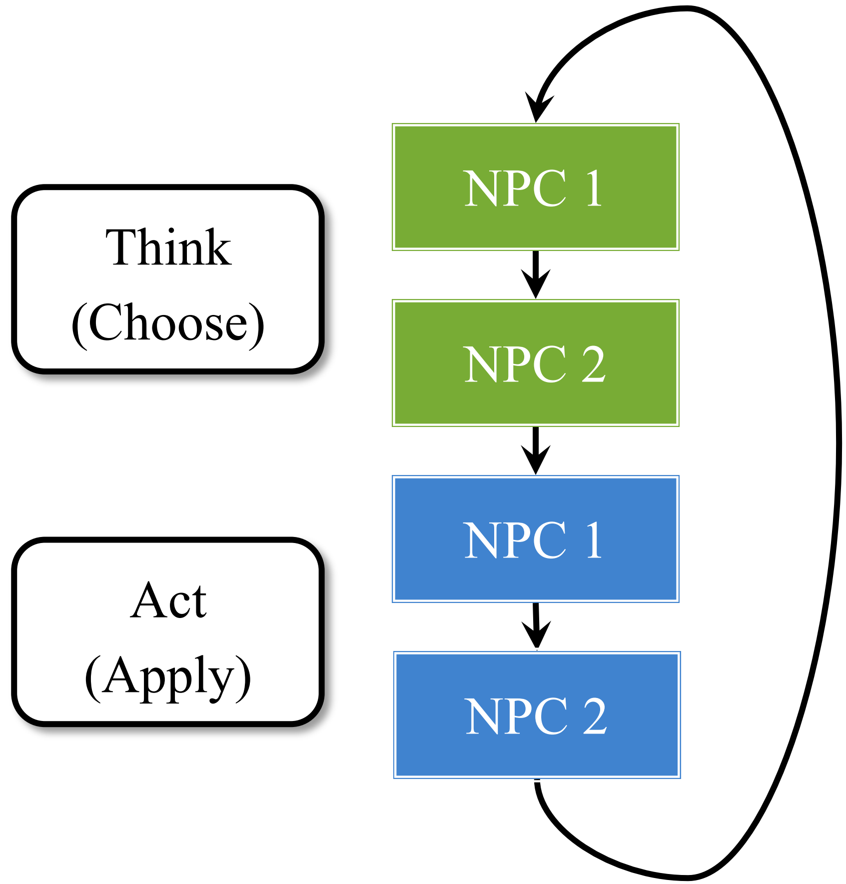
```
move(int dx, int dy) {
    x += dx;
    y += dy;
}
```

Delaying Actions

Sequential Actions are Bad



Choose Action; Apply Later



Thinking: Primary Challenge

- A mess of conditionals
  - "Spaghetti" code
  - Difficult to modify
- Abstraction requirements:
  - Easy to visualize models
  - Mirror "cognitive thought"
- Want to separate talent
  - **Sensing**: Programmers
  - **Thinking**: *Designers*
  - **Actions**: Programmers

```
if (sense1) {
   if (sense11) { …
   } else if (sense12){ …
   } else if (sense13){ …
   } else {…
   }
} else if (sense2) {
   if (sense21) { …
   } else if (sense22){ …
   } else {…
   }
} else if (sense3) { …
}
```

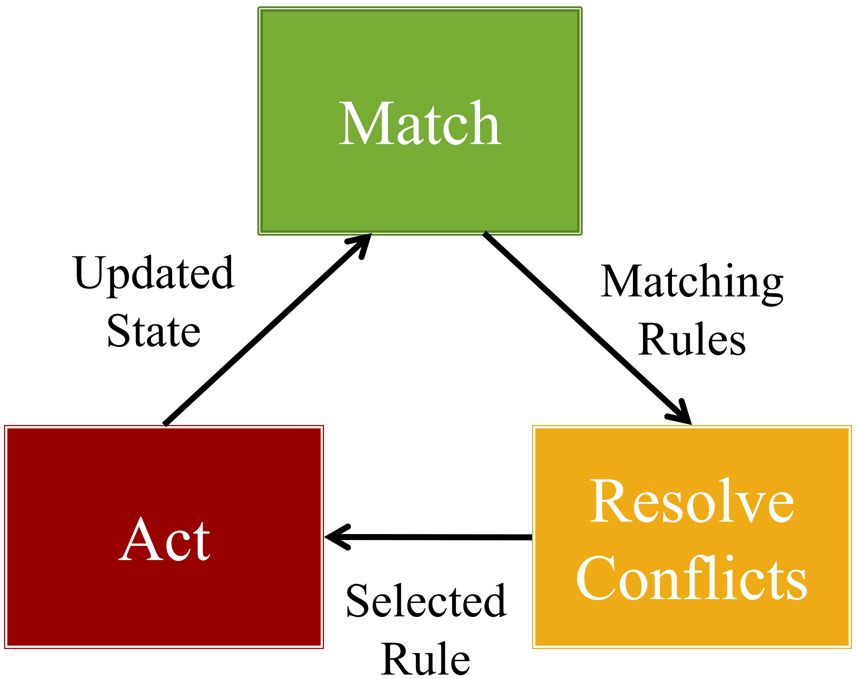
Thinking: Primary Challenge

- A mess of conditionals
  - "Spaghetti" code
  - Difficult to modify
- Abstraction requirements:
  - Easy to visualize models
  - Mirror "cognitive thought"
- Want to separate talent
  - **Sensing**: Programmers
  - **Thinking**: *Designers*
  - **Actions**: Programmers

```
if (sense1) {
   if (sense11) { …
   } else if (sense12){ …
   } else if (sense13){ …
   } else {…
   }
} else if (sense2) {
   if (sense21) { …
   } else if (sense22){ …
   } else {…
   }
} else if (sense3) { …
}
```

Rule-Based AI

If *X* is true, Then do *Y*



Three-Step Process

- **Match**
  - For each rule, check **if**
  - Return *all* matches
- **Resolve**
  - Can only use one rule
  - Use metarule to pick one
- **Act**
  - Do **then**-part

Rule-Based AI

If ***X*** is true, Then do ***Y***

- **Thinking**: Providing a list of several rules
  - But what happens if there is more than one rule?
  - Which rule do we choose?

Rule-Based AI

Sensing / Acting

If ***X*** is true, Then do ***Y***

- **Thinking**: Providing a list of several rules
  - But what happens if there is more than one rule?
  - Which rule do we choose?

Simplicity of Rule-Based AI

Conflict Resolution

- Often **resolve by order**
  - Each rule has a priority
  - Higher priorities go first
  - "Flattening" conditionals
- **Problems**:
  - Predictable
    Same events = same rules
  - Total order
    Sometimes no preference
  - Performance
    On average, go far down list

R_1: `if` event_1 `then` act_1

R_2: `if` event_2 `then` act_2

R_3: `if` event_3 `then` act_3

R_4: `if` event_4 `then` act_4

R_5: `if` event_5 `then` act_5

R_6: `if` event_6 `then` act_6

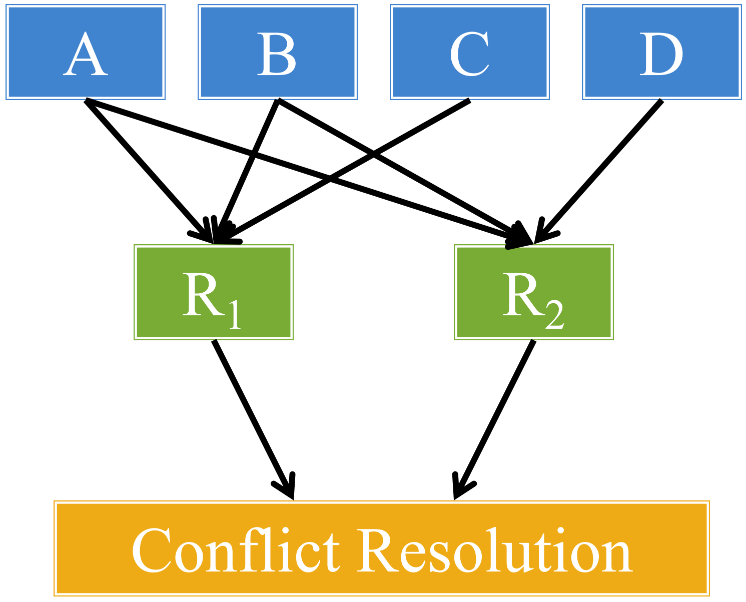
R_7: `if` event_7 `then` act_7

Conflict Resolution

- **Specificity**:
  - Rule w/ most "components"
- **Random**:
  - Select randomly from list
  - May "weight" probabilities
- **Refractory Inhibition**:
  - Do not repeat recent rule
  - Can combine with ordering
- **Data Recency**:
  - Select most recent update

R_1: `if` A, B, C, `then`

R_2: `if` A, B, D, `then`



Impulses

- Correspond to certain events
  - **Global**: not tied to NPC
  - Must also have duration
- Used to **reorder** rules
  - Event makes rule important
  - Temporarily up the priority
  - Restore when event is over
- Preferred conflict resolution
  - Simple but flexible
  - Used in *Halo* 3 AI.

R_1: `if` event_1 `then` act_1

R_2: `if` event_2 `then` act_2

R_3: `if` event_3 `then` act_3

R_4: `if` event_4 `then` act_4

R_5: `if` event_5 `then` act_5

R_6: `if` event_6 `then` act_6

R_7: `if` event_7 `then` act_7

Impulses

- Correspond to certain events
  - **Global**: not tied to NPC
  - Must also have duration
- Used to **reorder** rules
  - Event makes rule important
  - Temporarily up the priority
  - Restore when event is over
- Preferred conflict resolution
  - Simple but flexible
  - Used in *Halo* 3 AI.

R_1: `if` event_1 `then` act_1

R_2: `if` event_2 `then` act_2

R_5: **`if`** $\textbf{event}_5$ **`then`** $\textbf{act}_5$

R_3: `if` event_3 `then` act_3

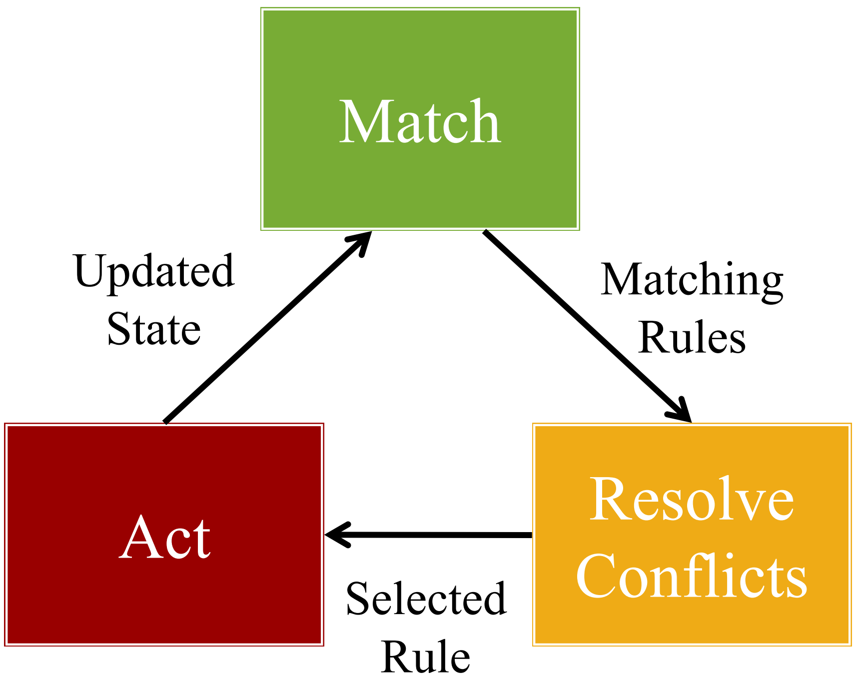
R_4: `if` event_4 `then` act_4

R_6: `if` event_6 `then` act_6

R_7: `if` event_7 `then` act_7

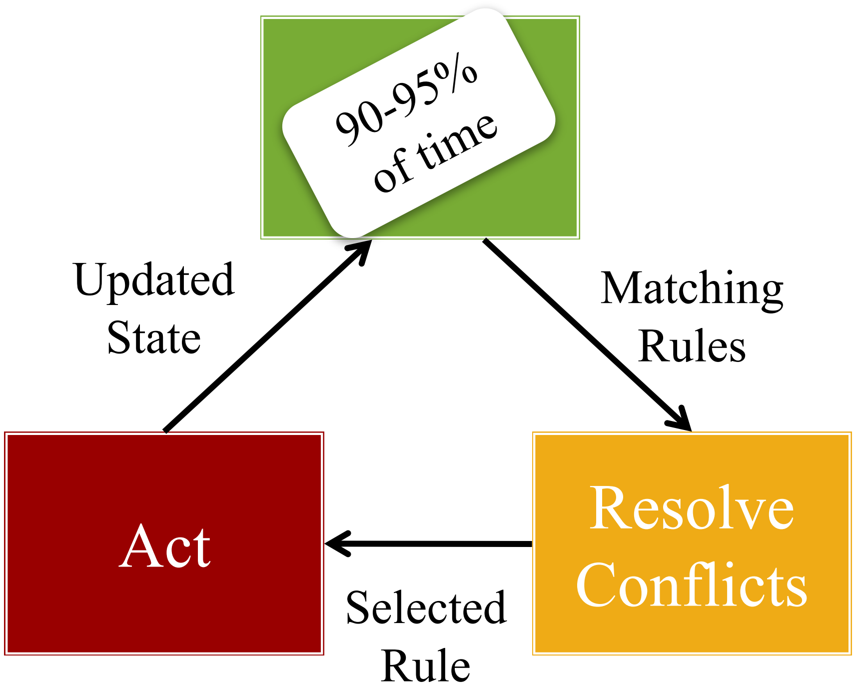
Rule-Based AI: Performance

- Matching = **sensing**
  - If-part is expensive
  - Test *every* condition
  - Many unmatched rules
- Improving performance
  - Optimize sensing (make if-part cheap)
  - Limit number of rules
  - Other solutions?
- Most games limit rules
  - Reason for *state machines*



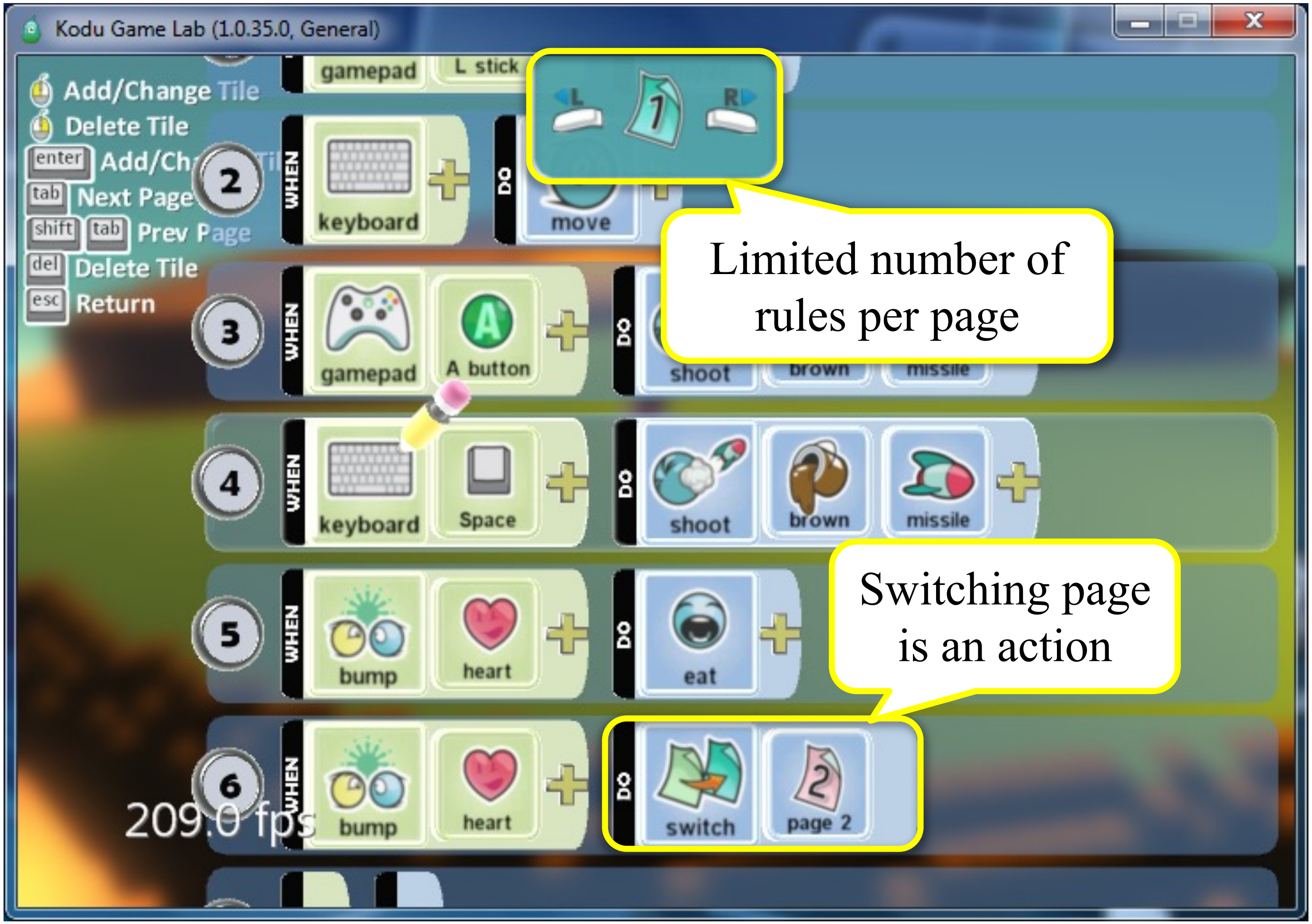
Rule-Based AI: Performance

- Matching = **sensing**
  - If-part is expensive
  - Test *every* condition
  - Many unmatched rules
- Improving performance
  - Optimize sensing (make if-part cheap)
  - Limit number of rules
  - Other solutions?
- Most games limit rules
  - Reason for *state machines*

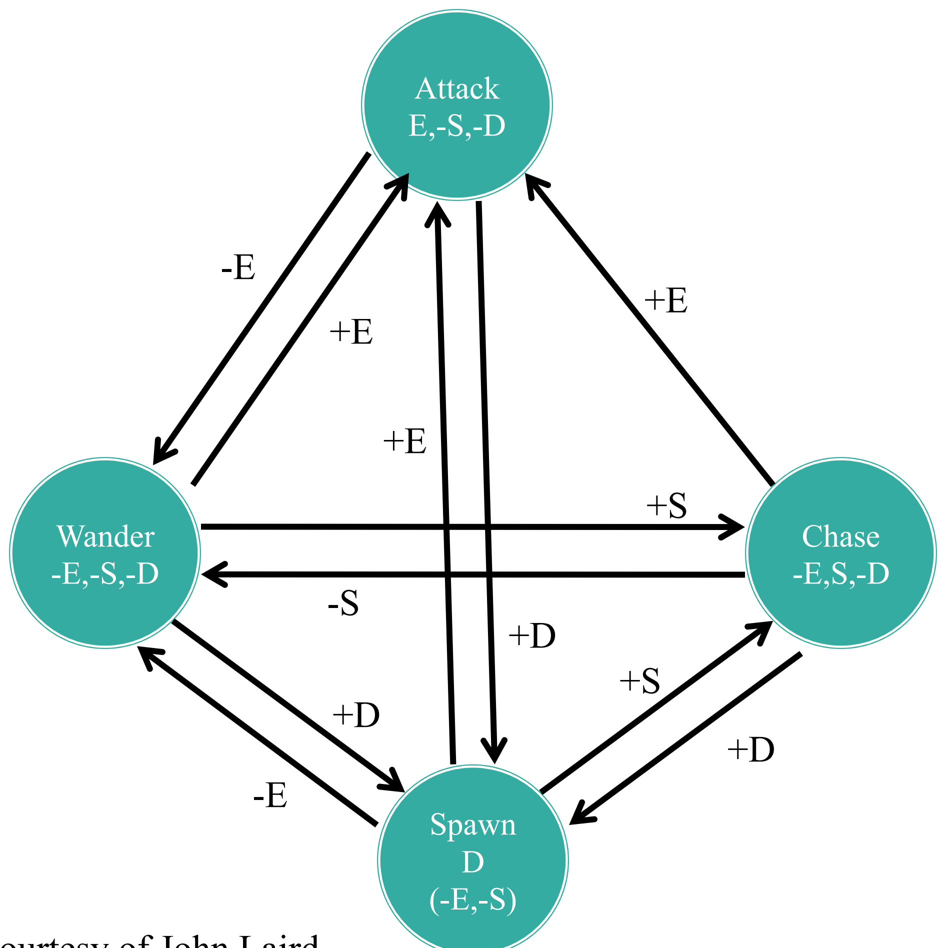


Making the Rules Manageable

Making the Rules Manageable



Finite State Machines



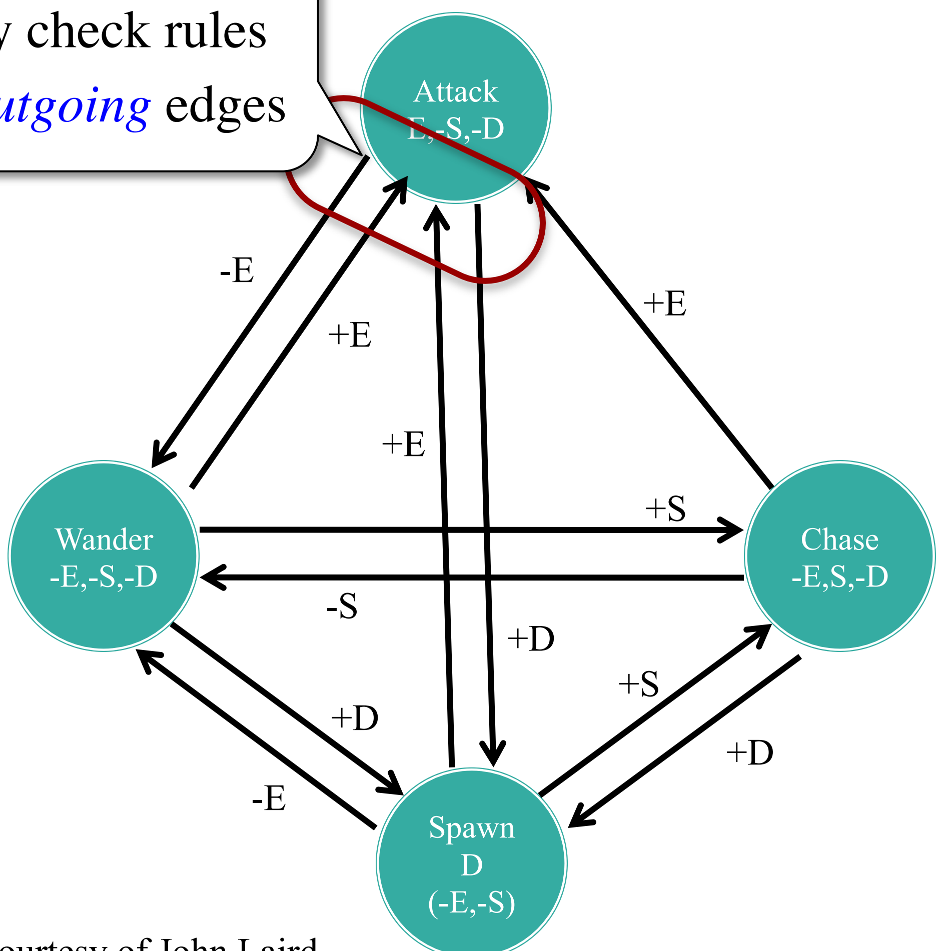
Events

- **E**=Enemy Seen
- **S**=Sound Heard
- **D**=Die

Slide courtesy of John Laird

Finite State Machines

Only check rules for *outgoing* edges



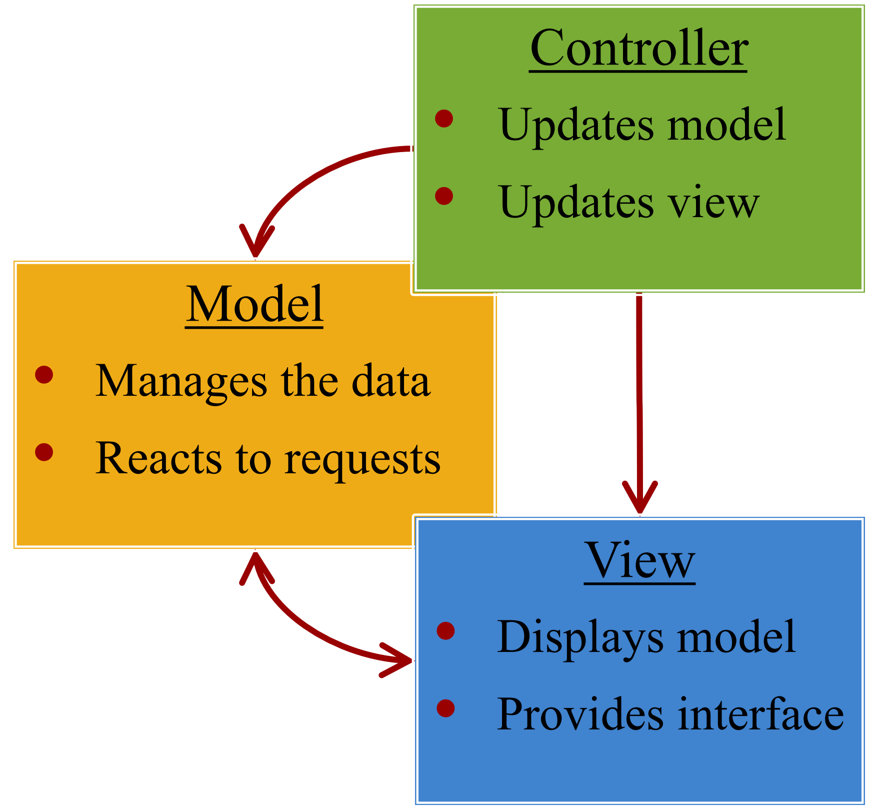
Events

- **E**=Enemy Seen
- **S**=Sound Heard
- **D**=Die

Slide courtesy of John Laird

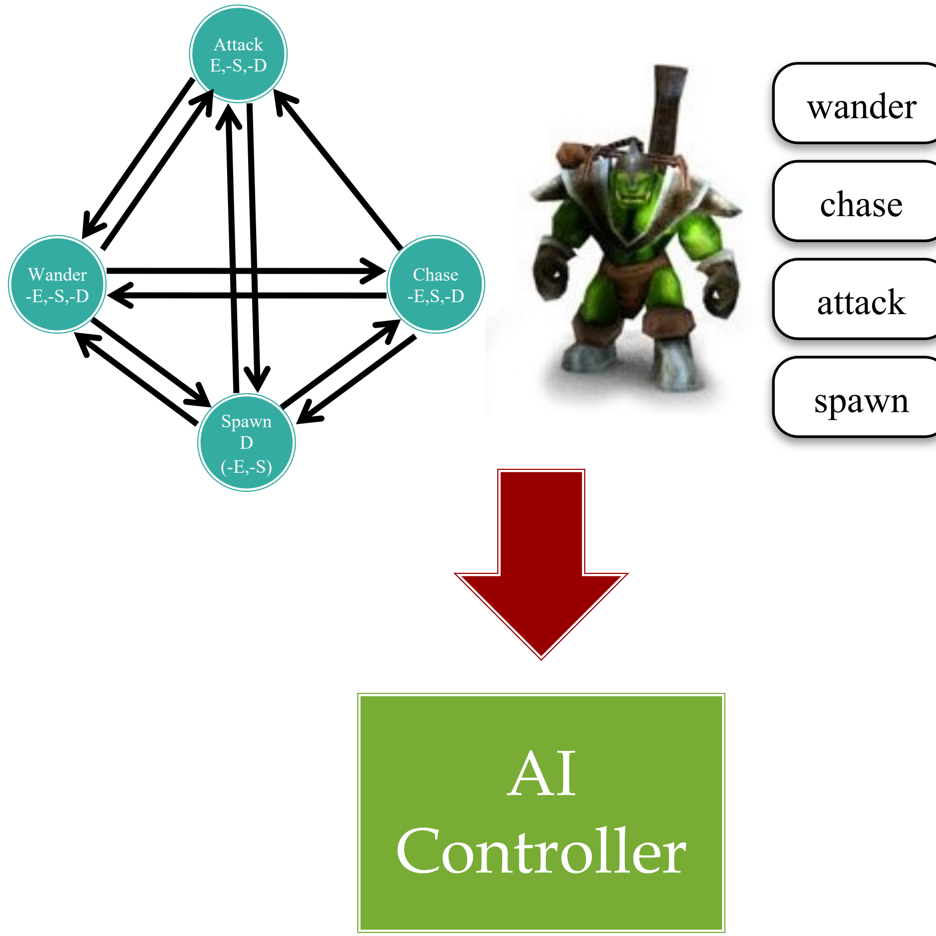
Implementation: Model-View-Controller

- Games have **thin** models
  - Methods = get/set/update
  - Controllers are heavyweight
- AI is a **controller**
  - Uniform process over NPCs
- But behavior is *personal*
  - Diff. NPCs = diff. behavior
  - Do not want unique code
- What can we do?
  - Data-Driven Design



Implementation: Model-View-Controller

- **Actions** go in the model
  - Lightweight updates
  - Specific to model or role
- Controller is framework for general **sensing**, **thinking**
  - Standard FSM engine
  - Or FSM alternatives (later)
- **Process** stored in a model
  - Represent thinking as *graph*
  - Controller processes graph



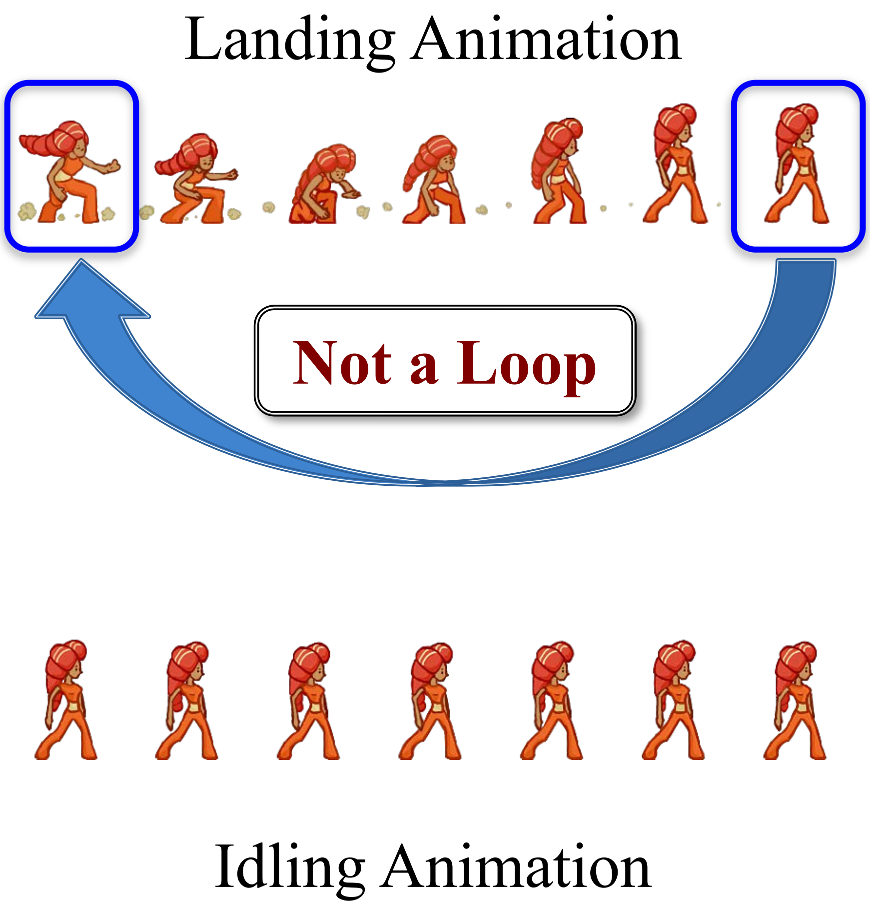
An Aside: Animations

Landing Animation

Idling Animation

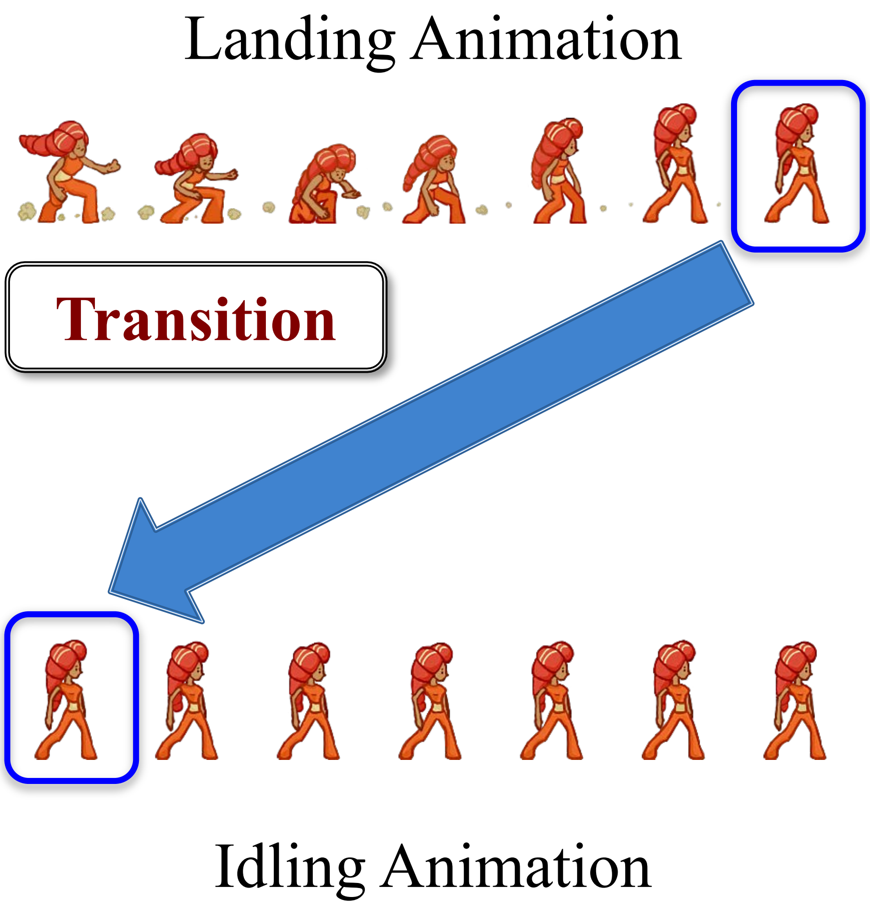
- AI may need many actions
  - Run, jump, duck, slide
  - Fire weapons, cast spells
  - Fidget while idling
- Want animations for all
  - Is loop appropriate for each?
  - How do we transition?
- **Idea**: shared boundaries
  - End of loop = start of another
  - Treat like advancing a frame

An Aside: Animations



- AI may need many actions
  - Run, jump, duck, slide
  - Fire weapons, cast spells
  - Fidget while idling
- 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

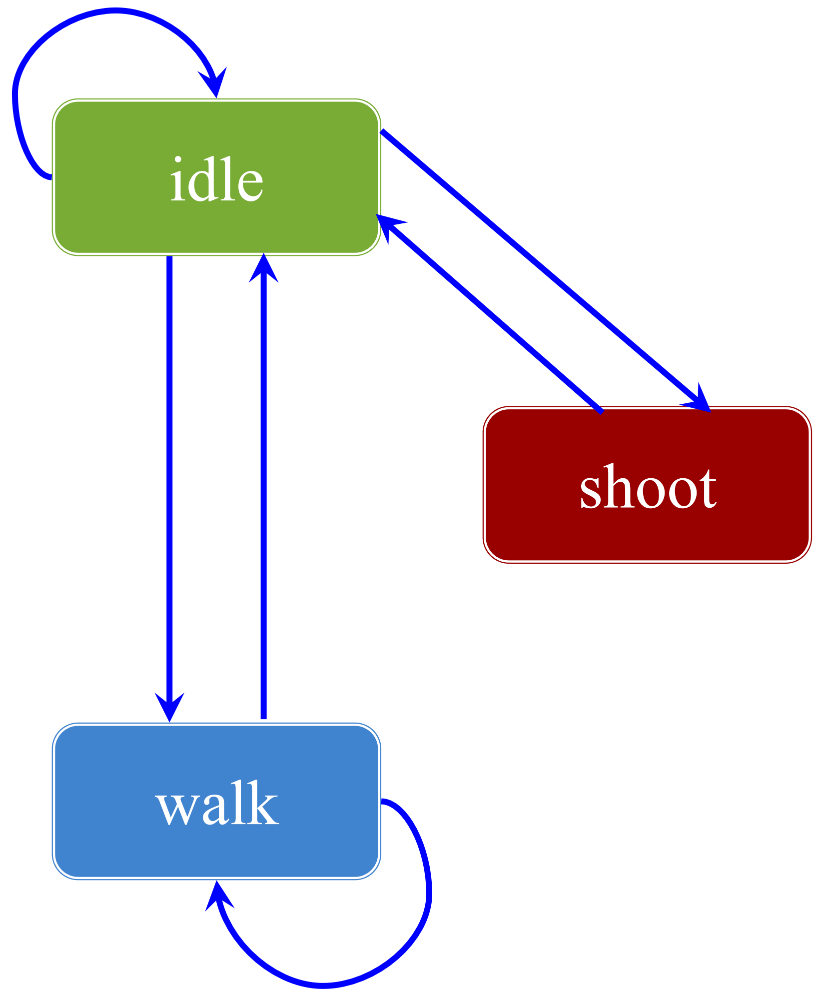
An Aside: Animations



- AI may need many actions
  - Run, jump, duck, slide
  - Fire weapons, cast spells
  - Fidget while idling
- 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

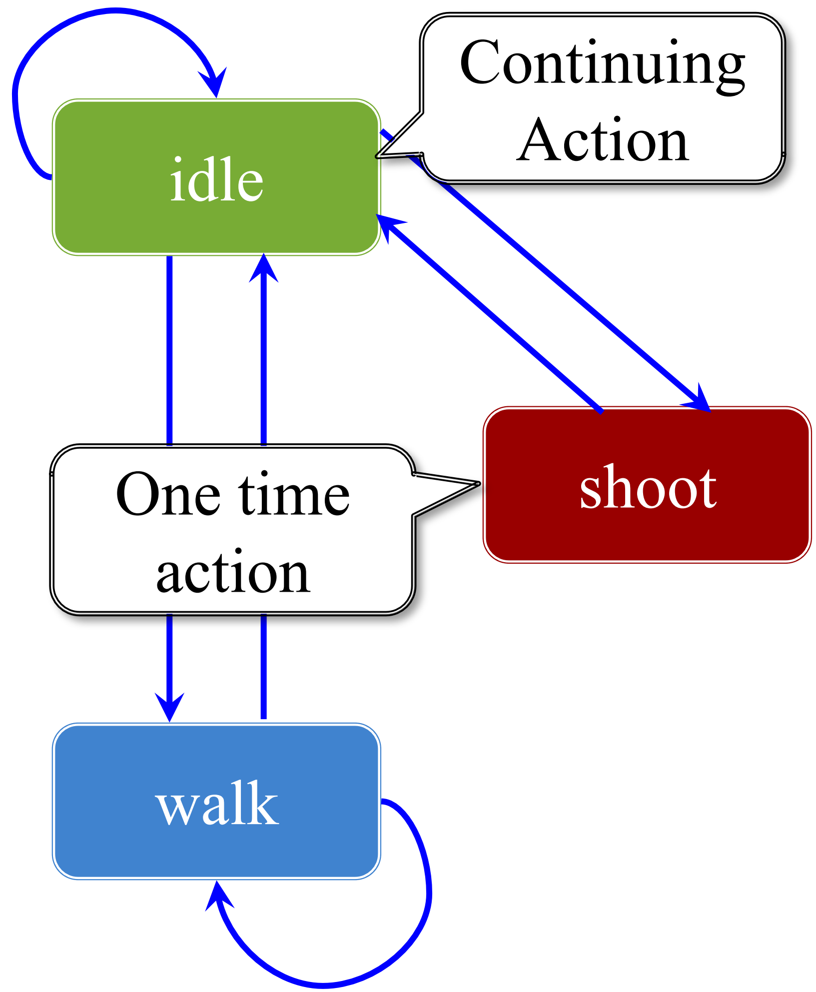
Animation and State Machines

- **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

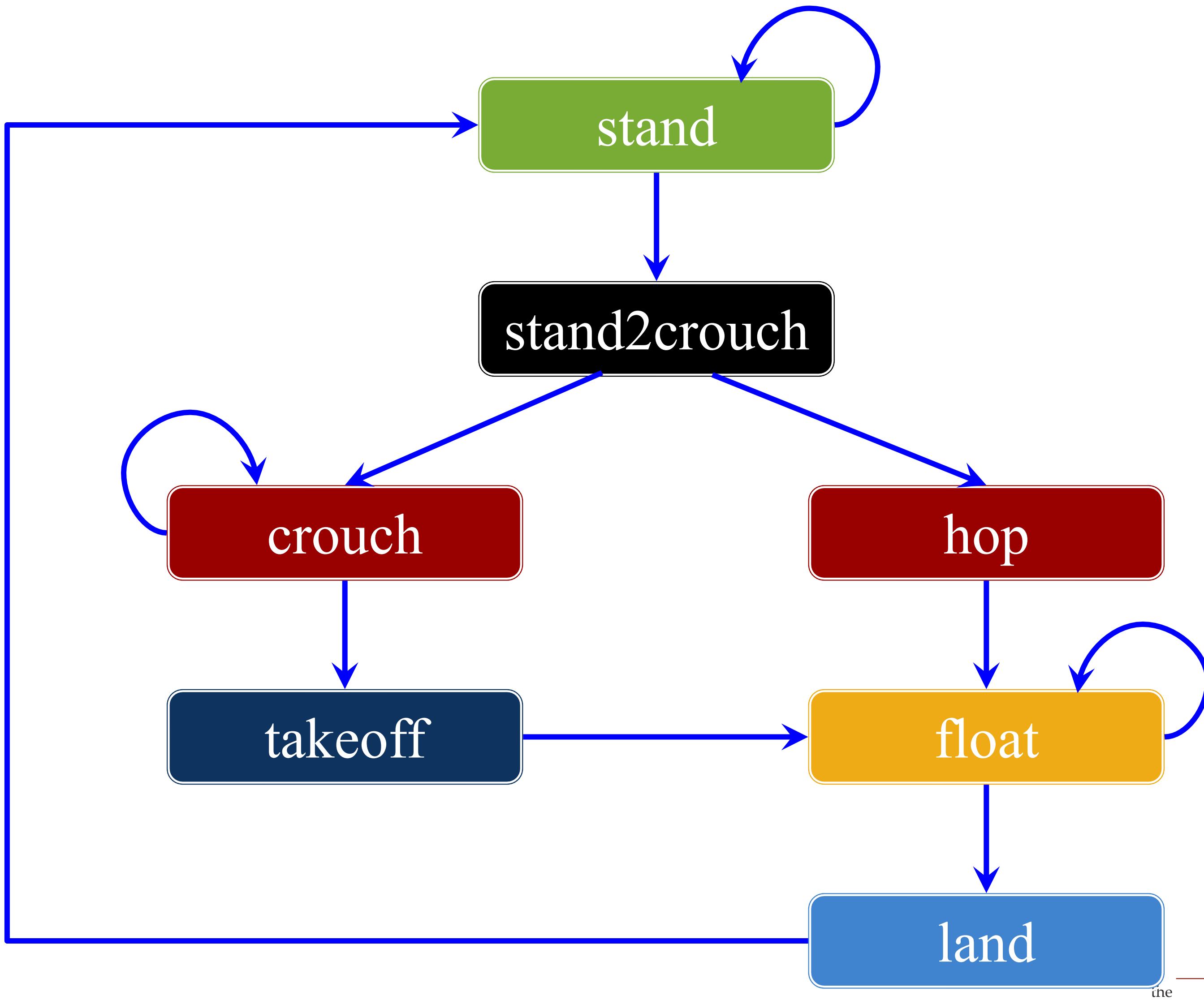


Animation and State Machines

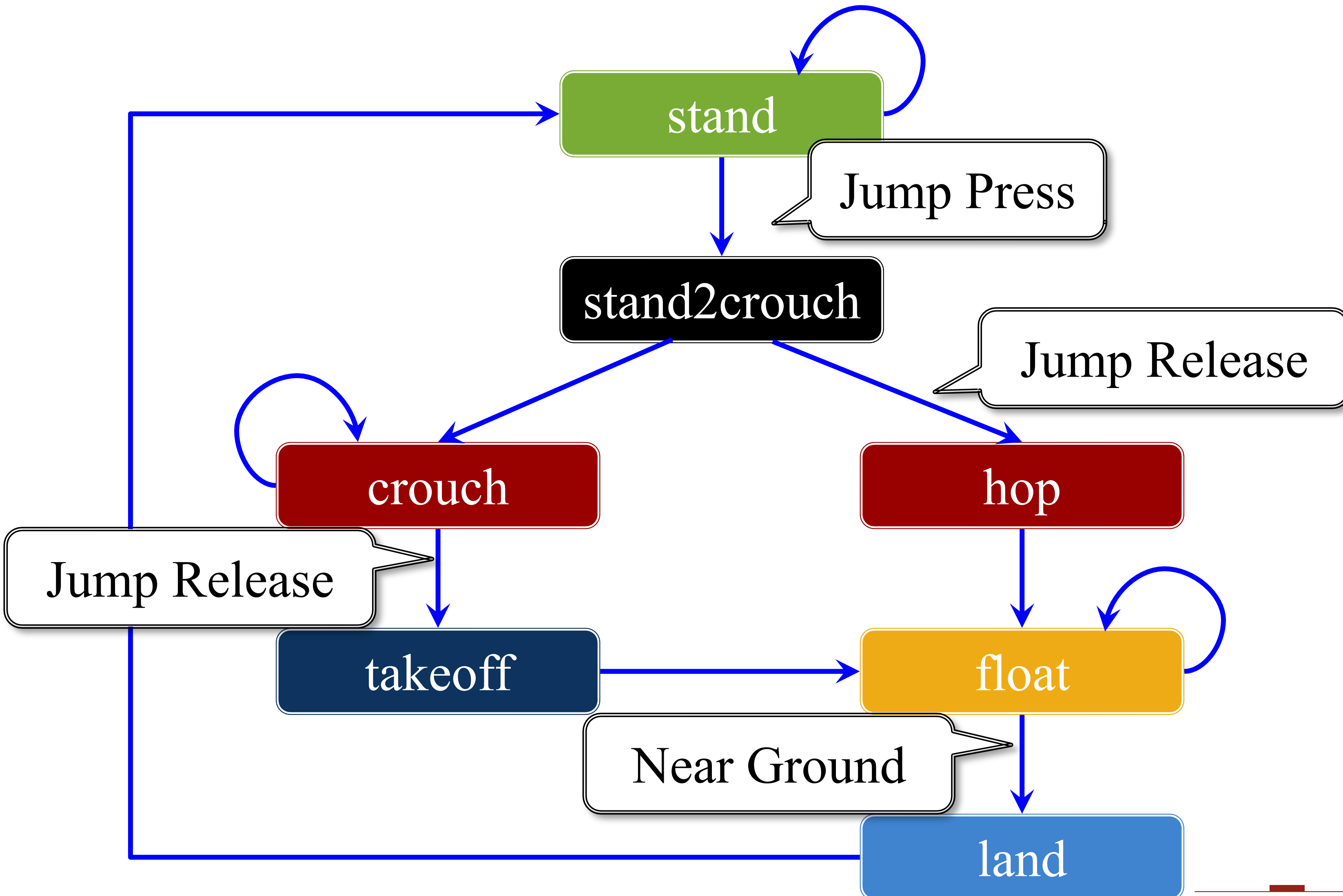
- **Idea**: Each sequence a state
  - Do sequence while in state
  - Transition when at end
  - Only loop if loop in graph
- A graph edge means…
  - Boundaries match up
  - Transition is allowable
- Similar to data driven AI
  - Created by the designer
  - Implemented by programmer
  - Modern engines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



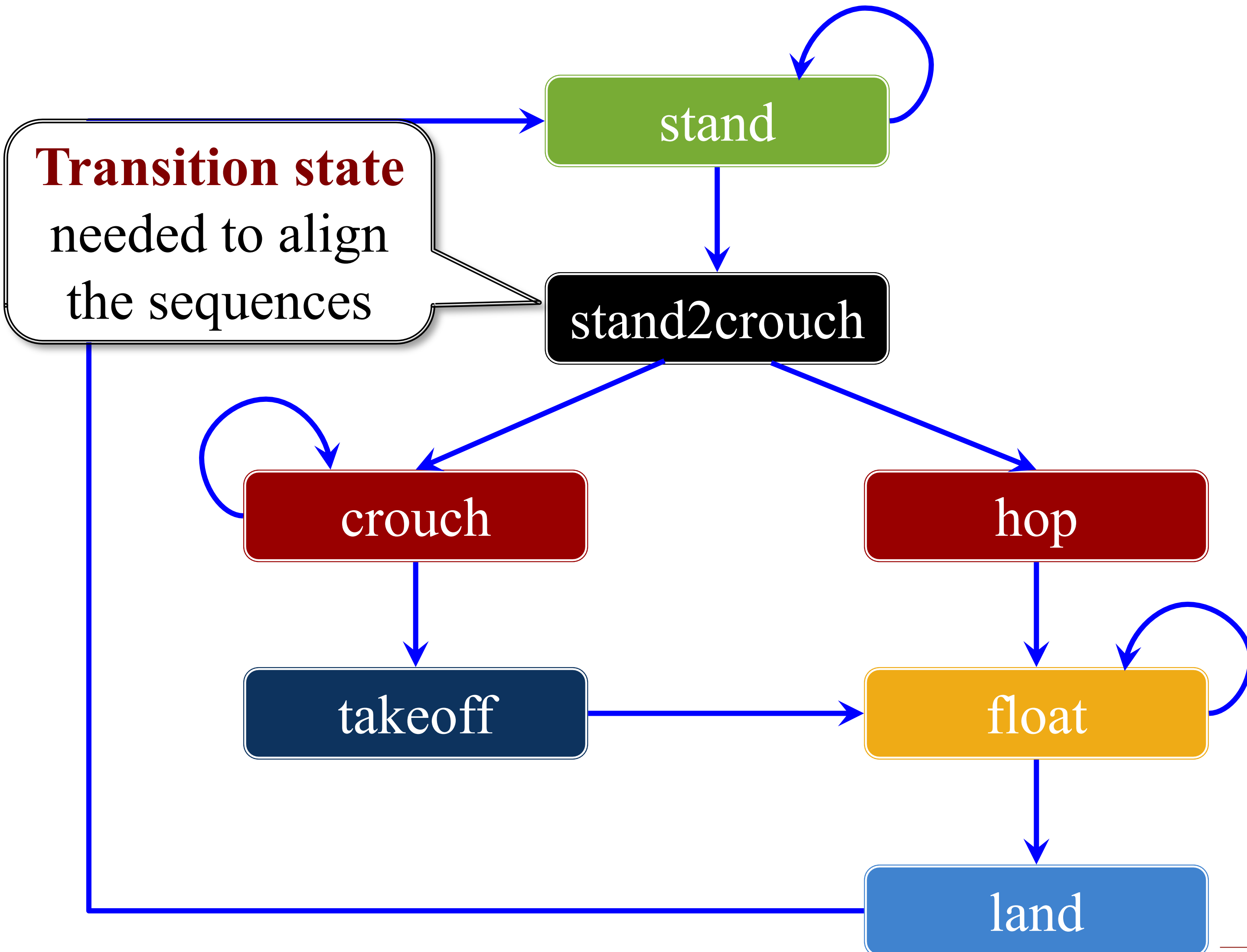
Complex Example: Jumping



Complex Example: Jumping



Complex Example: Jumping



LibGDX Interfaces

StateMachine<E>

- Attached to an entity
  - Set the entity in constructor
  - New entity, new state machine
- Must implement methods
  - `update()`
  - `changeState(State<A> state)`
  - `revertToPreviousState()`
  - `getCurrentState()`
  - `isInState(State<A> state)`
- `DefaultStateMachine` provided

State<E>

- Not attached to an entity
  - `StateMachine` sets state
  - `StateMachine` passes entity
- Must implement methods
  - `enter(E entity)`
    When machine enters state
  - `exit(E entity)`
    When machine enters state
  - `update(E entity)`
    When machine stays in state

LibGDX Interfaces

StateMachine<E>

- Attached to an entity
  - ...ructor
  - ...ma...
- Mu... implement metho...
  - update()
  - changeState(State<A> state)
  - revertToPreviousState()
  - getCurrentState()
  - isInState(State<A> state)
- DefaultStateMachine provided

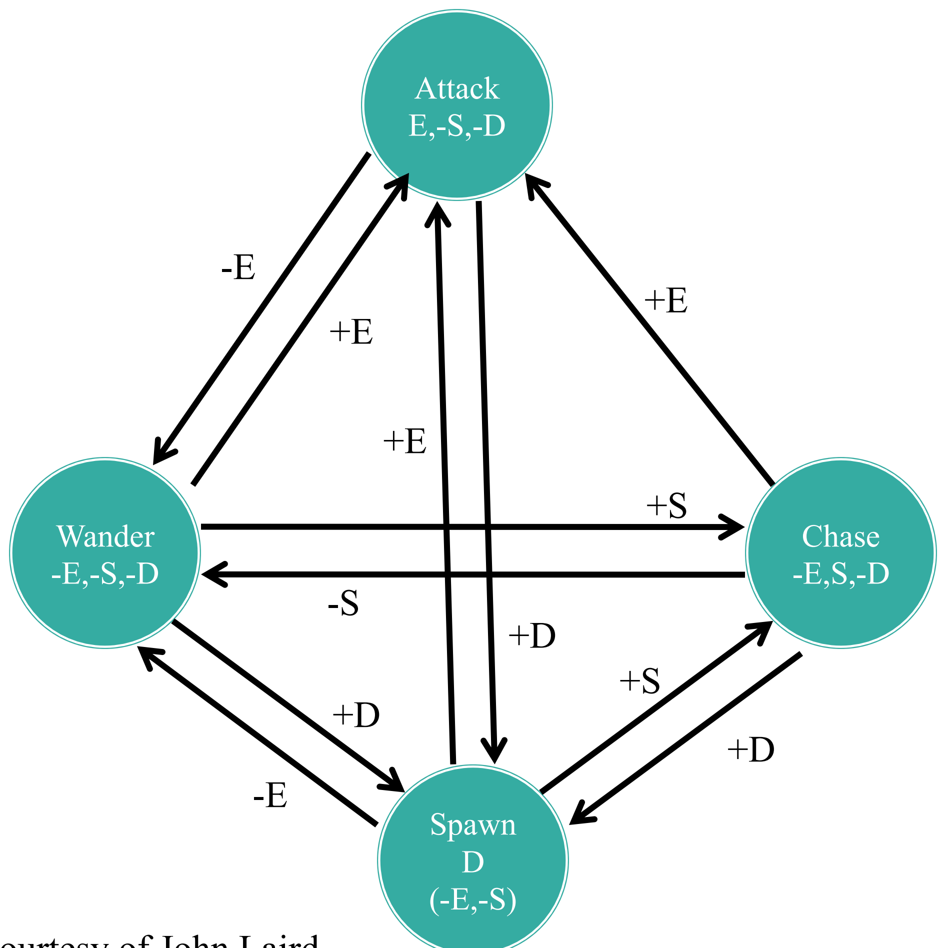
Updates current state.
Does not transition!

Transition logic
external to the
state machine.

State<E>

- Not attached to an entity
  - StateMachine sets state
  - ...achine passes entity
- ...lement methods
  - enter(E entity)
    When machine enters state
  - exit(E entity)
    When machine enters state
  - update(E entity)
    When machine stays in state

Problems with FSMs



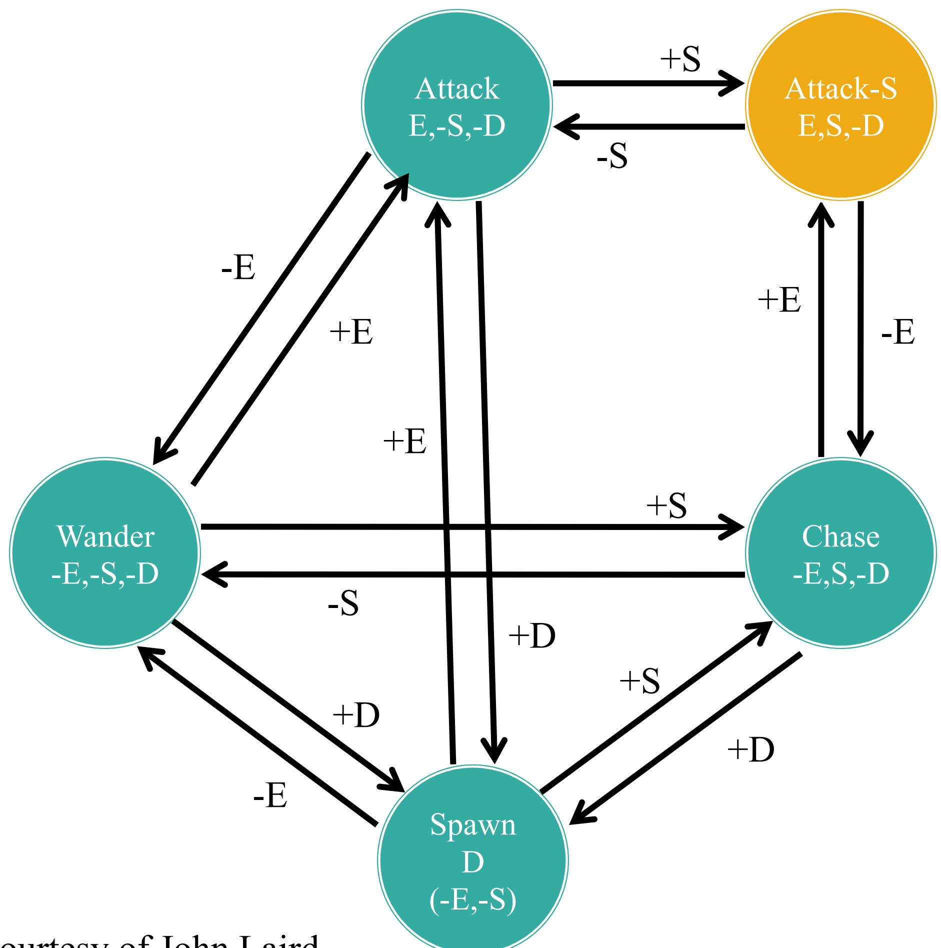
Events

- **E**=Enemy Seen
- **S**=Sound Heard
- **D**=Die

No edge from **Attack** to **Chase**

Slide courtesy of John Laird

Problems with FSMs



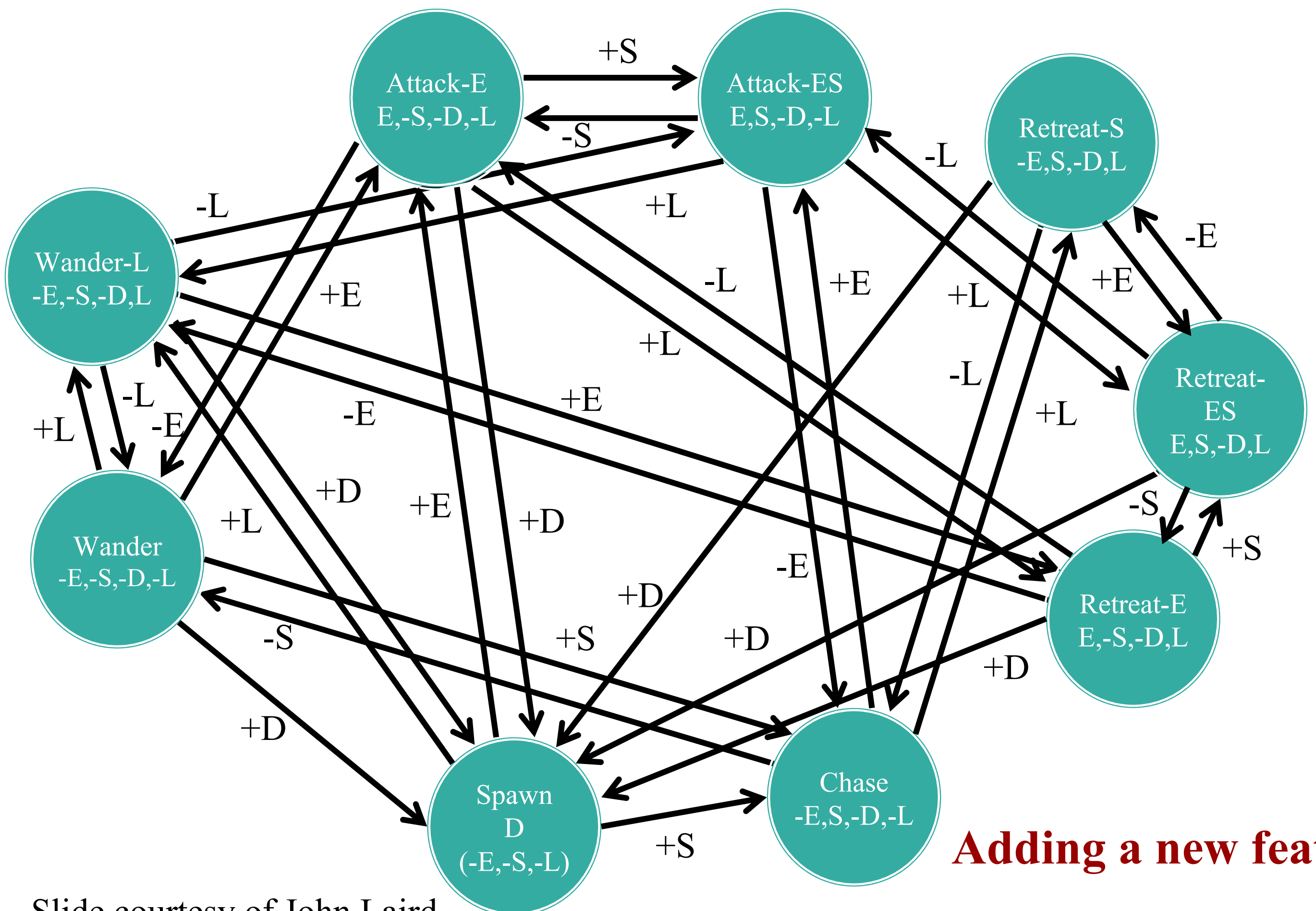
Events

- **E**=Enemy Seen
- **S**=Sound Heard
- **D**=Die

Requires a *redundant* state

Slide courtesy of John Laird

Problems with FSMs



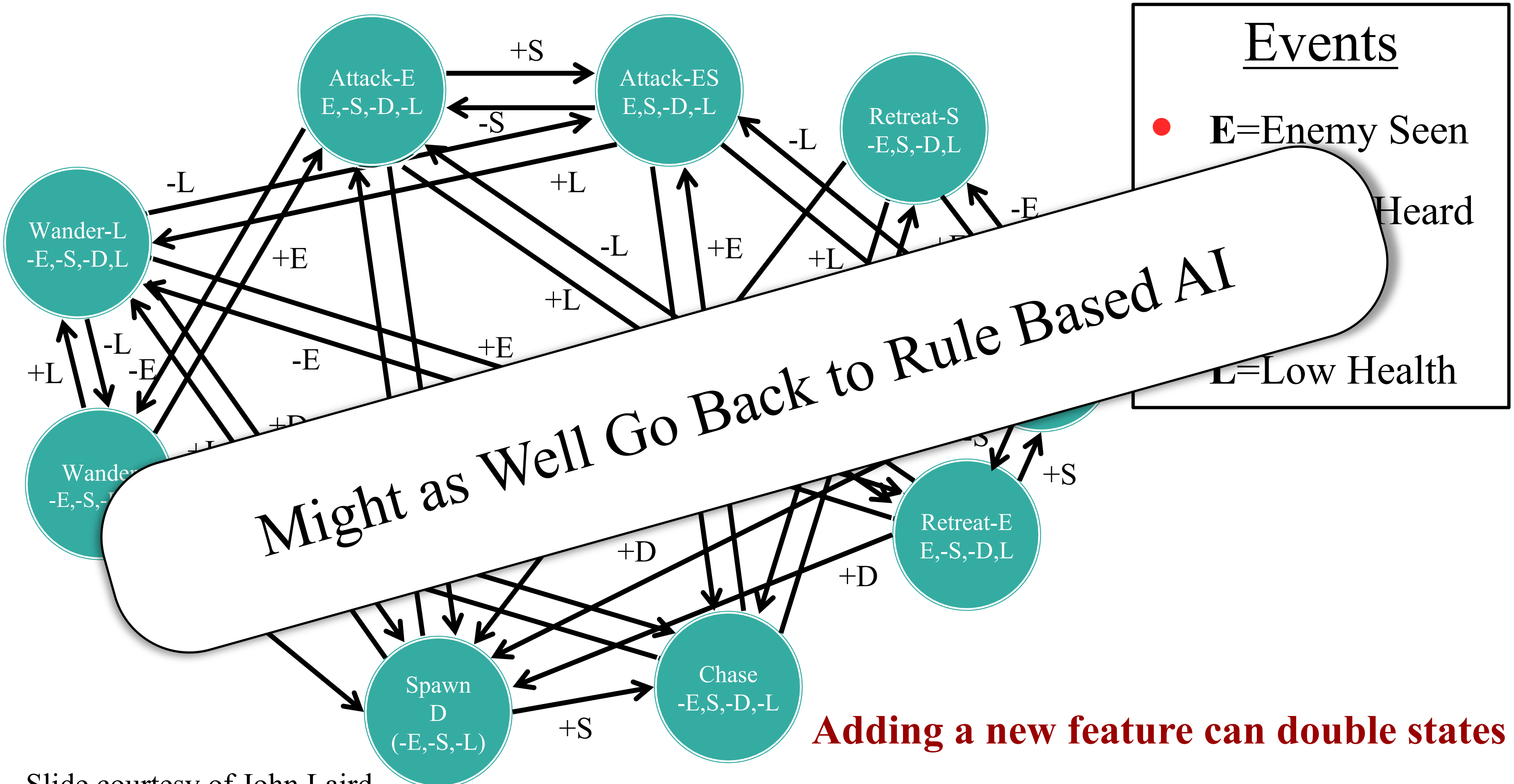
Events

- **E**=Enemy Seen
- **S**=Sound Heard
- **D**=Die
- **L**=Low Health

Adding a new feature can double states

Slide courtesy of John Laird

Problems with FSMs



Adding a new feature can double states

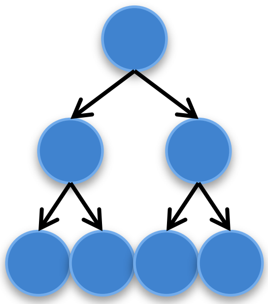
Slide courtesy of John Laird

An Observation

- Each state has a set of **global attributes**
  - Different attributes may have same actions
  - Reason for redundant behavior
- Currently just cared about attributes
  - Not really using the full power of a FSM
  - Why don't we just check attributes directly?
- Attribute-based selection: *decision trees*

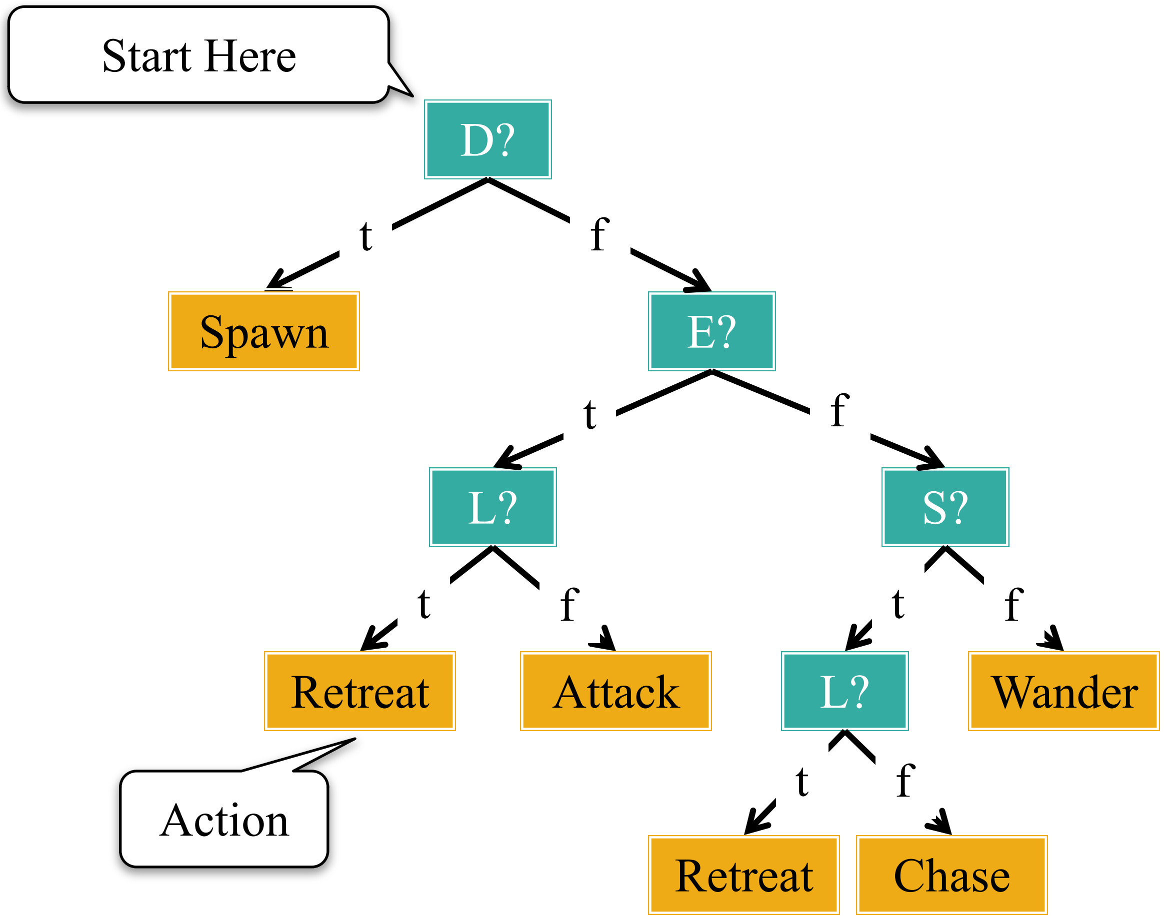
Decision Trees

- Thinking **encoded as a tree**
  - Attributes = tree nodes
  - Left = true, right = false
  - Actions = leaves (reach from the root)



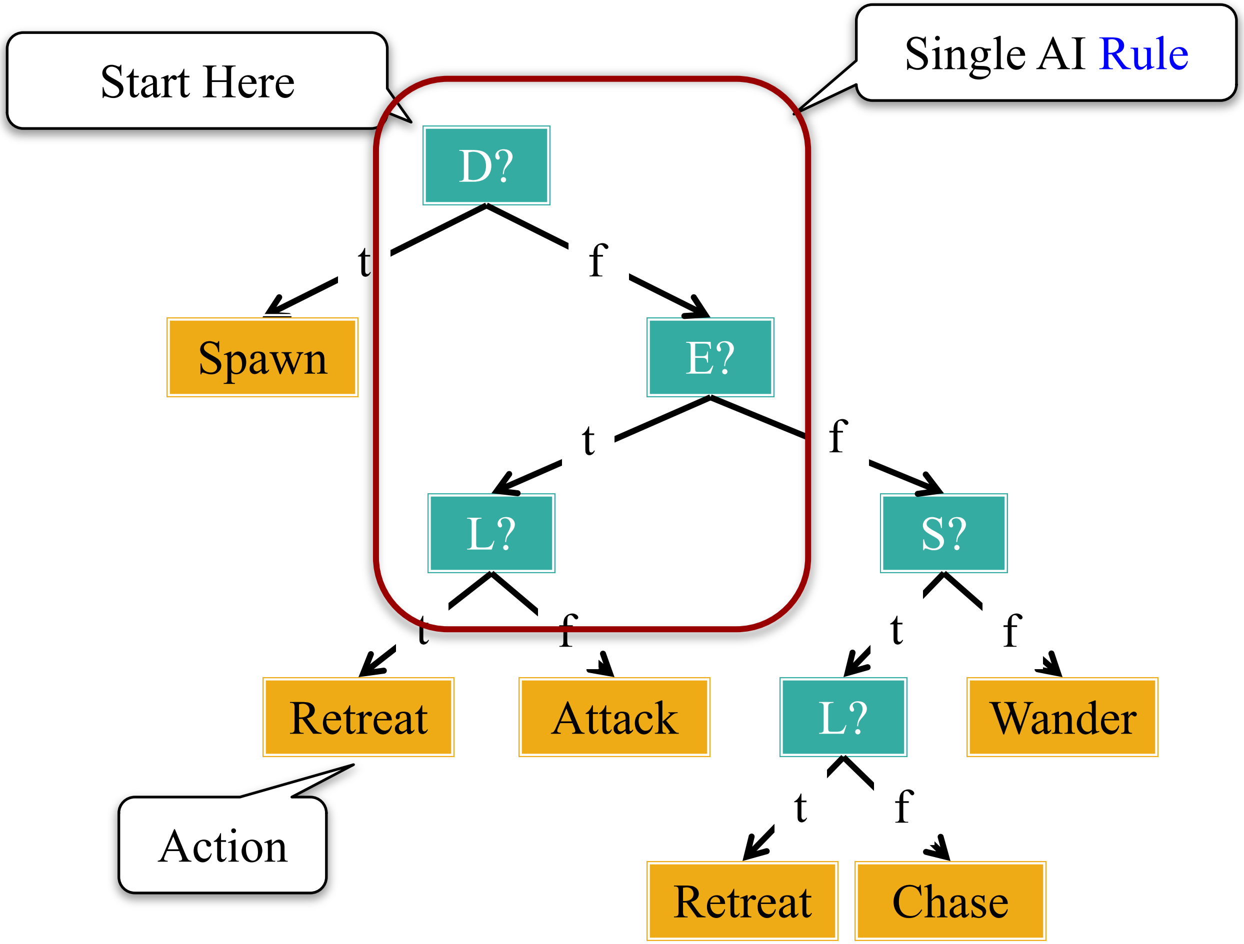
- Classify by **descending** from root to a leaf
  - Start with the test at the root
  - Descend the branch according to the test
  - Repeat until a leaf is reached

Decision Tree Example



Slide courtesy of John Laird

Decision Tree Example

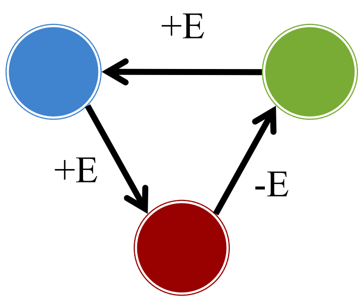


Slide courtesy of John Laird

FSMs vs. Decision Trees

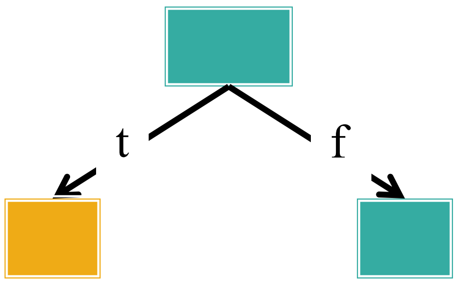
Finite State Machines

- Not limited to attributes
- Allow "arbitrary" behavior
- Explode in size very fast

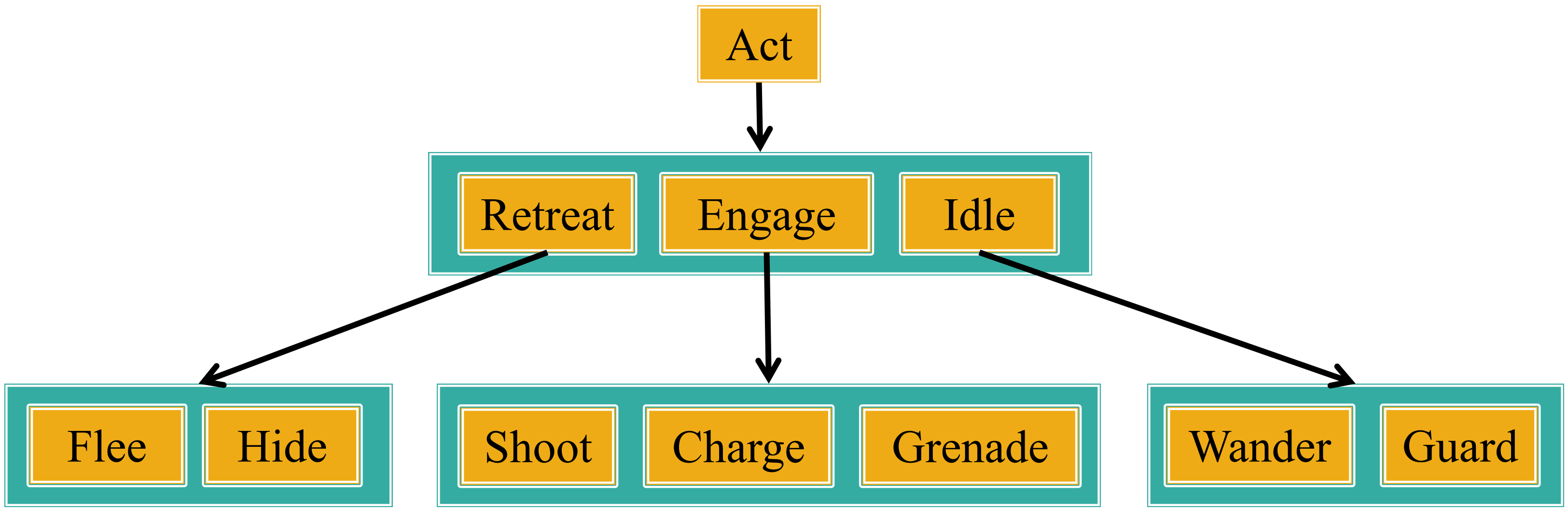


Decision Trees

- Only attribute selection
- Much more manageable
- Mixes w/ machine learning



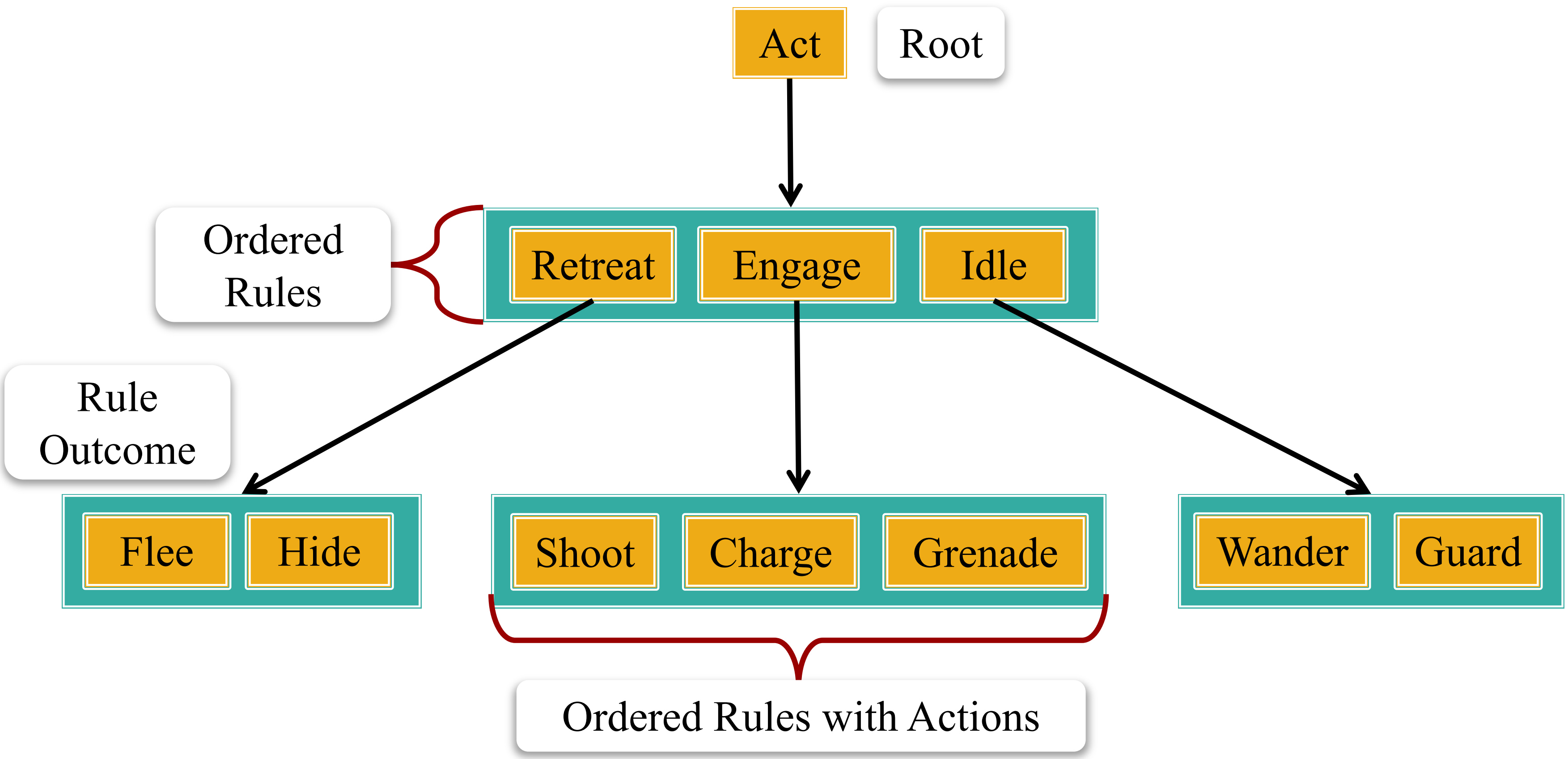
Behavior Trees



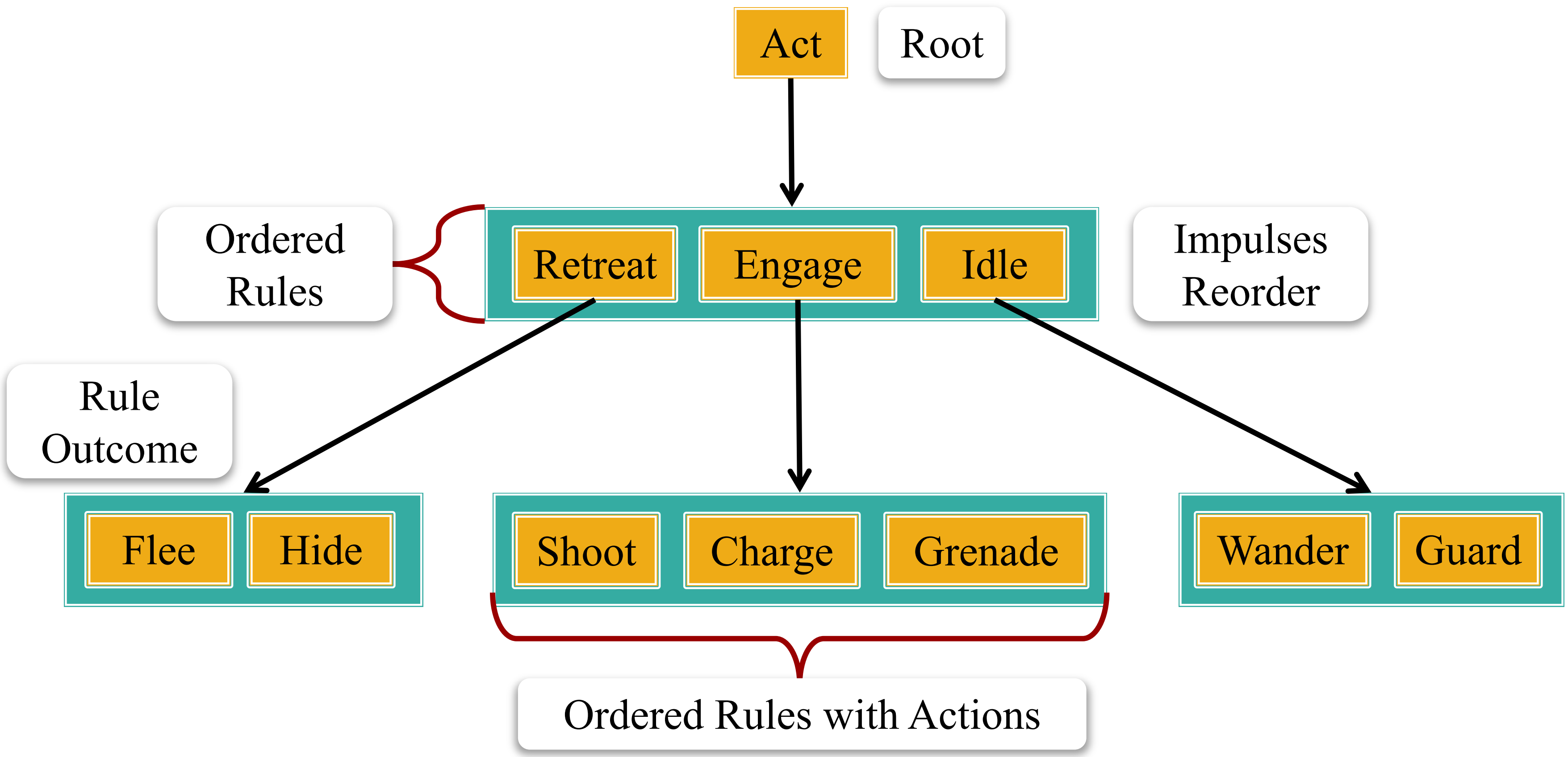
- Part rule-based
- Part decision tree
- Freedom of FSM (almost)

- Node is a list of *actions*
- Select action using *rules*
- Action leads to *subactions*

Behavior Trees

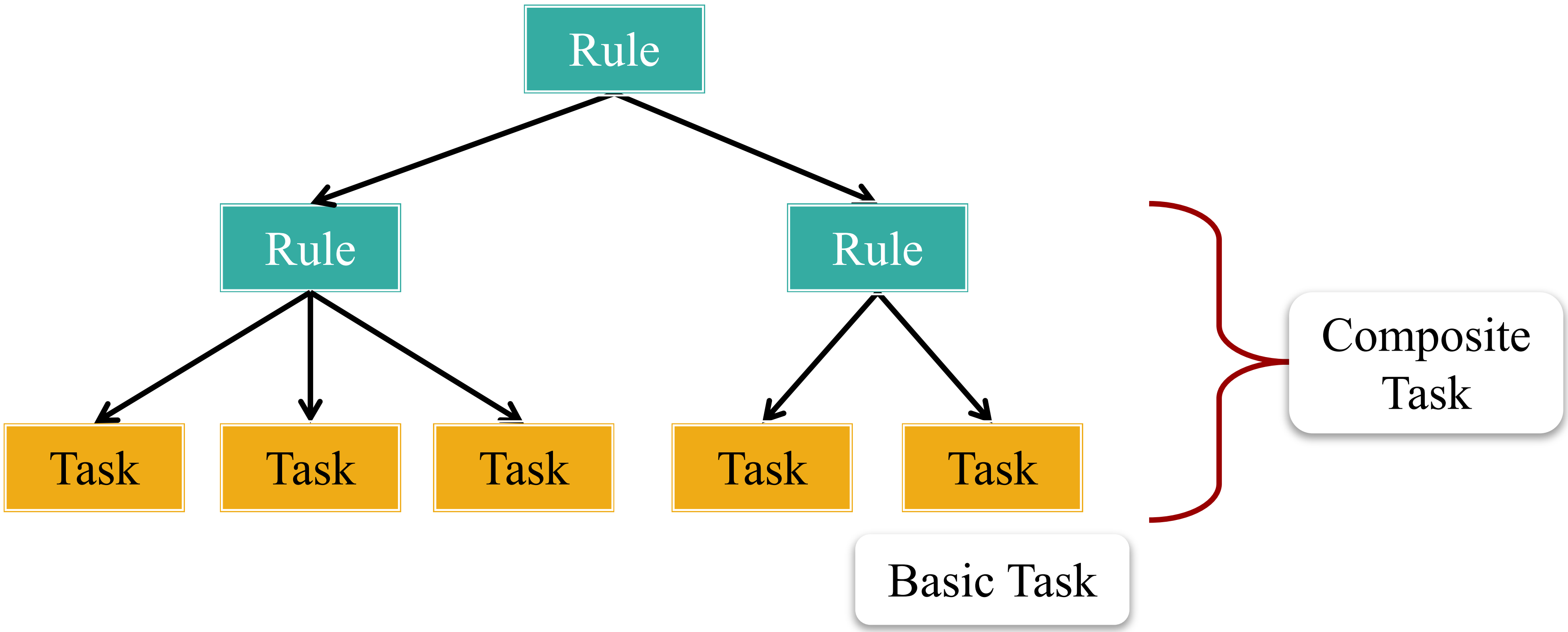


Behavior Trees



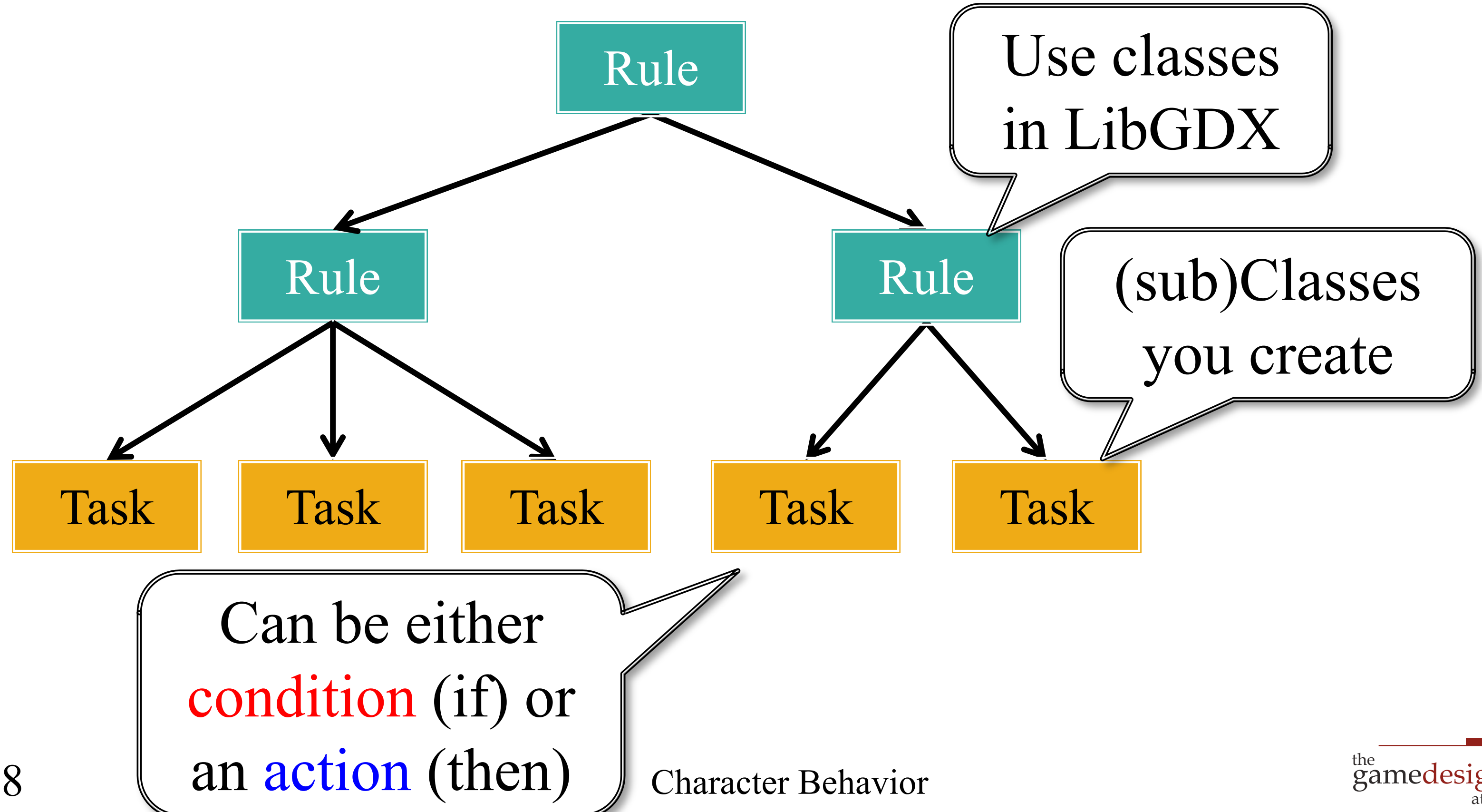
LibGDX Behavior Trees

- Base actions are defined at the leaves
- Internal nodes to **select** or even **combine** tasks

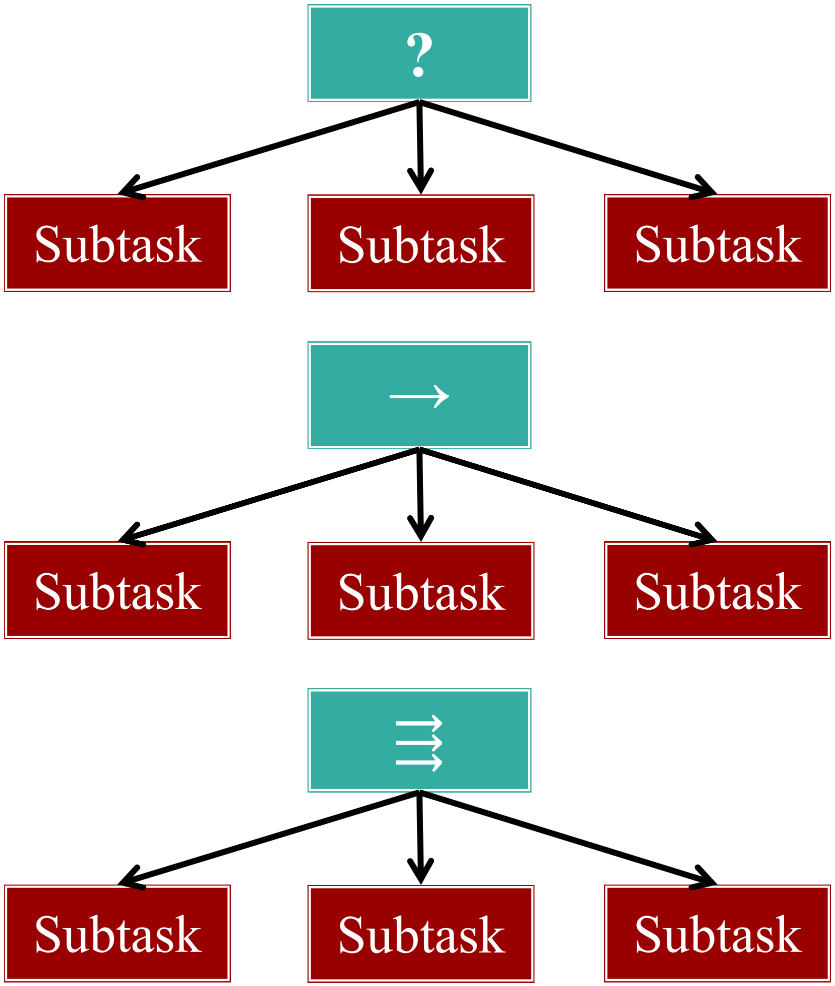


LibGDX Behavior Trees

- Base actions are defined at the leaves
- Internal nodes to **select** or even **combine** tasks

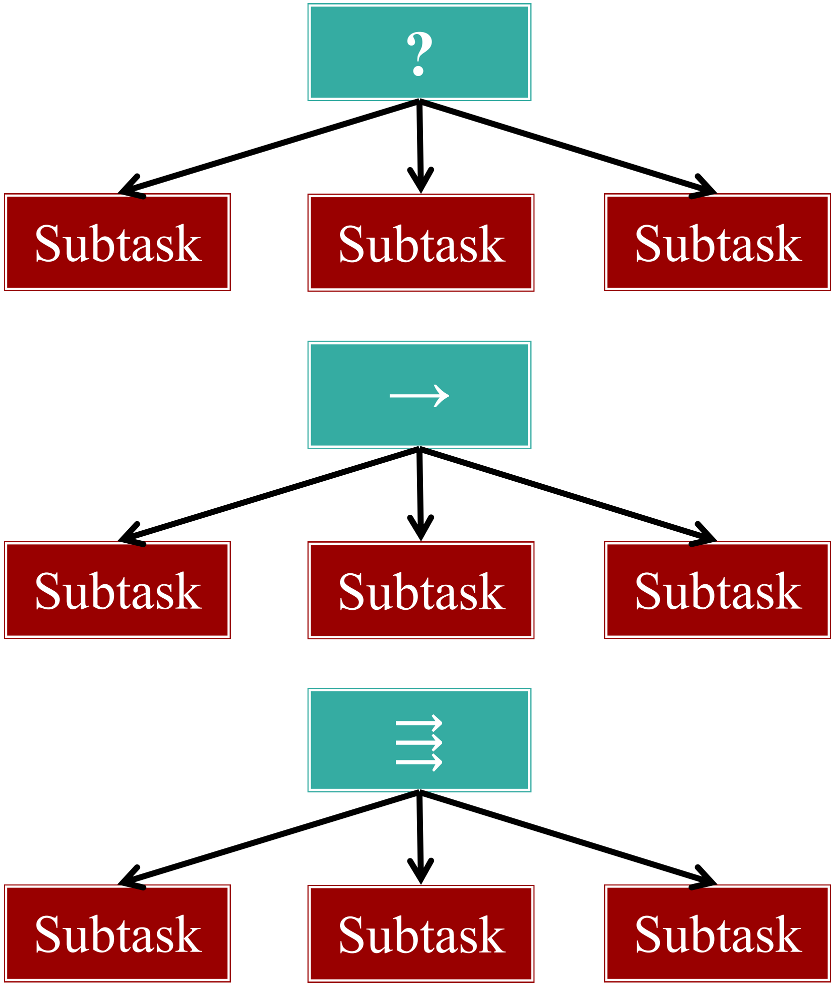


LibGDX Rules



- **Selector** rules
  - Tests each subtask for success
  - Tasks are tried independently
  - Chooses first one to succeed
- **Sequence** rules
  - Tests each subtask for success
  - Tasks are tried in order
  - Does all if succees; else none
- **Parallel** rules
  - Tests each subtask for success
  - Tasks are tried simultaneously
  - Does all if succees; else none

This is the **Wrong** Model



- **Conflates** actions/selection
  - Want way to pick subtask
  - Distinct from performing it
- Actions must be **instant**
  - Can switch each frame
  - Action unaware of switch
  - No way to suspend/recover
- 4152 has **experimental API**
  - Still being tested in class
  - Bring to 3152 eventually

Summary

- Character AI is a **software engineering** problem
  - Sense-think-act aids code reuse and ease of design
  - Least standardized aspect of game architecture
- **Rule-based AI** is the foundation for all character AI
  - Simplified variation of sense-think-act
  - Alternative systems made to limit number of rules
- Games use **graphical models** for data-driven AI
  - Controller outside of NPC model processes AI
  - Graph stored in NPC model tailors AI to individuals