

State-Based Scripting in

UNCHARTED 2

AMONG THIEVES

Jason Gregory
Generalist Programmer
Naughty Dog, Inc.



Agenda

- Introduction to **game scripting**
- Extending the **game object model**
- **State script** syntax
- Case studies from
Uncharted 2: Among Thieves
- **Implementation** discussion
- Summary and some **tips**



Introduction to State Scripts



Brief History of State Scripts

- Script system on *Uncharted: Drake's Fortune* originally developed for **in-game cinematics** (IGCs)
- Evolved into general **gameplay scripting system**



Brief History of State Scripts

- UDF IGC system revamped for heavy use on *Uncharted 2: Among Thieves*
- Inspirations:
 - GOAL language used on *Crash* and *Jak* series
 - State objects from *God of War* engine



GOD OF WAR™



NAUGHTY DOG

Why Script?

- Scripting languages used in games since **Quake C**
- Primary benefits of script:
 - Takes pressure off **engineering team**
 - Code becomes data—**rapid iteration**
 - Empowers **content creators**
 - Key enabler of **mod community**



Why Script?

- Scripting languages used in games since **Quake C**
- Primary benefits of script:
 - Takes pressure off **engineering team**
 - Code becomes data—**rapid iteration**
 - Empowers **content creators**
 - Key enabler of **mod community**

I feel empowered!



Scripting Language Characteristics

- Two kinds of game scripting languages:
 - **data definition** languages
 - **runtime** languages
- **Runtime** scripting languages typically:
 - **interpreted** by virtual machine (VM)
 - **simple and small**—low overhead
 - **accessible** to designers and other “non-programmers”
 - **powerful**—one line of code = big impact

Choice of Language

- At Naughty Dog, we make heavy use of both **data definition** and **runtime** script
 - Both based on **PLT Scheme** (a Lisp variant)
- Key benefits of Lisp-like languages:
 - Easy to **parse**
 - Data def and runtime code can be freely **intermixed**
 - Powerful **macro** system—easy to define **custom syntax**
 - Naughty Dog has a rich Lisp heritage—comfortable

Choice of Language

- Of course you don't have to use Lisp!
- Data definition languages:
 - custom text format,
 - Excel comma-separated values (.csv),
 - XML, ...
- Runtime languages:
 - Python, Lua, Pawn (Small C), OCaml, F#, ...
- Many popular engines already provide a scripting language:
 - Quake C, UnrealScript, C# (XNA), ...

Extending the Game Object Model

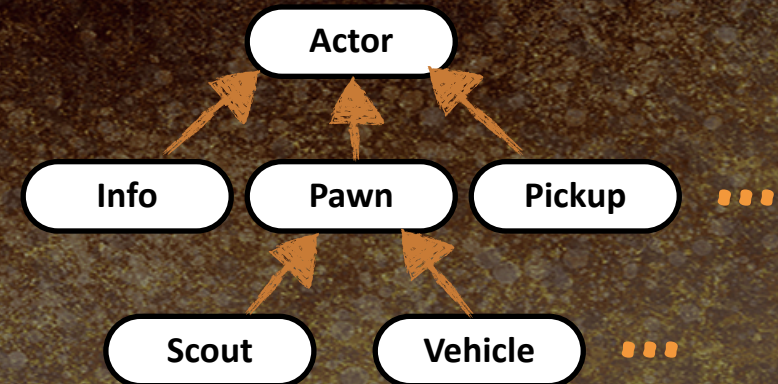
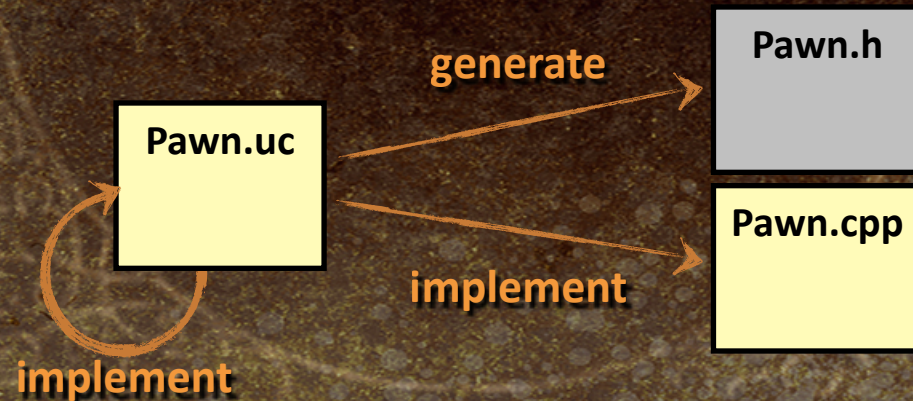
- Every game engine has some kind of **game object model**
 - Defines all **object types** in game world
 - Often (but not always) written in an **object-oriented language**
- Scripting language often used to **extend** the native object model
- Many ways to accomplish this...

Game Object Model References

- Rob Fermier, “**Creating a Data Driven Engine**,” GDC, 2002
- Scott Bilas, “**A Data-Driven Game Object System**,” GDC, 2002
(<http://www.drizzle.com/~scottb/gdc/game-objects.ppt>)
- Alex Duran, “**Building Object Systems: Features, Tradeoffs and Pitfalls**,” GDC, 2003
- Doug Church, “**Object Systems**,” presented at a game development conference in Seoul, Korea, 2003
(<http://chrishecker.com/images/6/6f/ObjSys.ppt>)
- Jason Gregory, “**Game Engine Architecture**,” AK Peters, 2009
(<http://gameenginebook.com>)

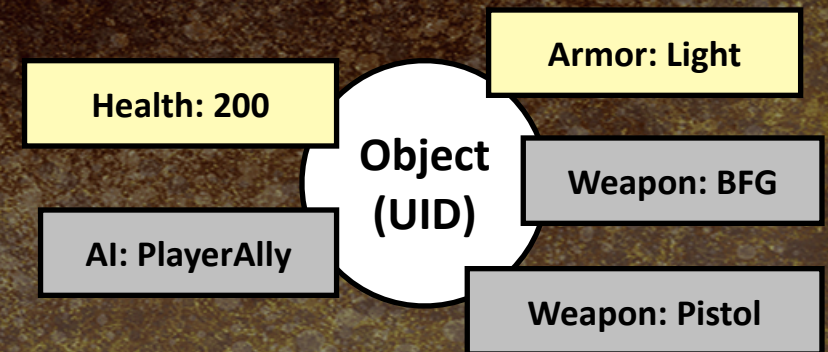
Unreal's Approach

- UnrealScript tightly integrated with C++ object model
 - Single-root class hierarchy with some add-on components
 - Classes defined in UnrealScript (.uc)
 - C++ header file (.h) automatically generated
 - Implementation in C++ or entirely in UnrealScript



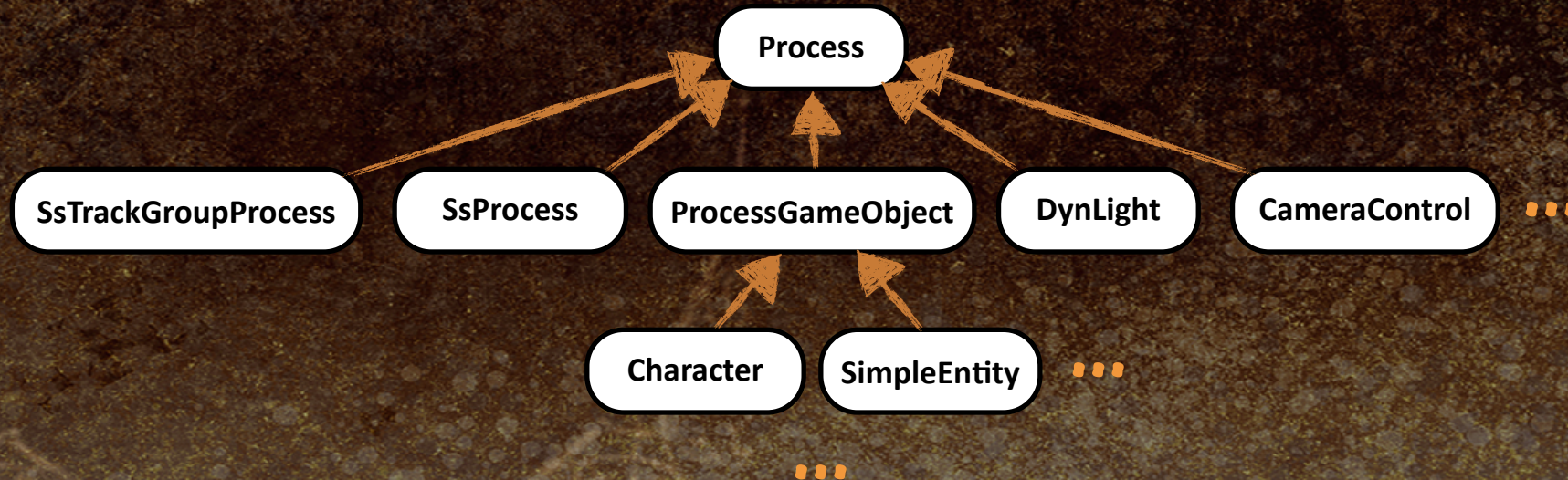
Property-Centric / Componentized Designs

- Property-centric design used on *Thief*, *Dungeon Siege*, *Age of Mythology*, *Deus Ex 2* and others
 - Game object just a **unique id** (UID)
 - “Decorated” with various **properties** (health, armor, weaponry, etc.)
 - Property encapsulates **data + behavior**



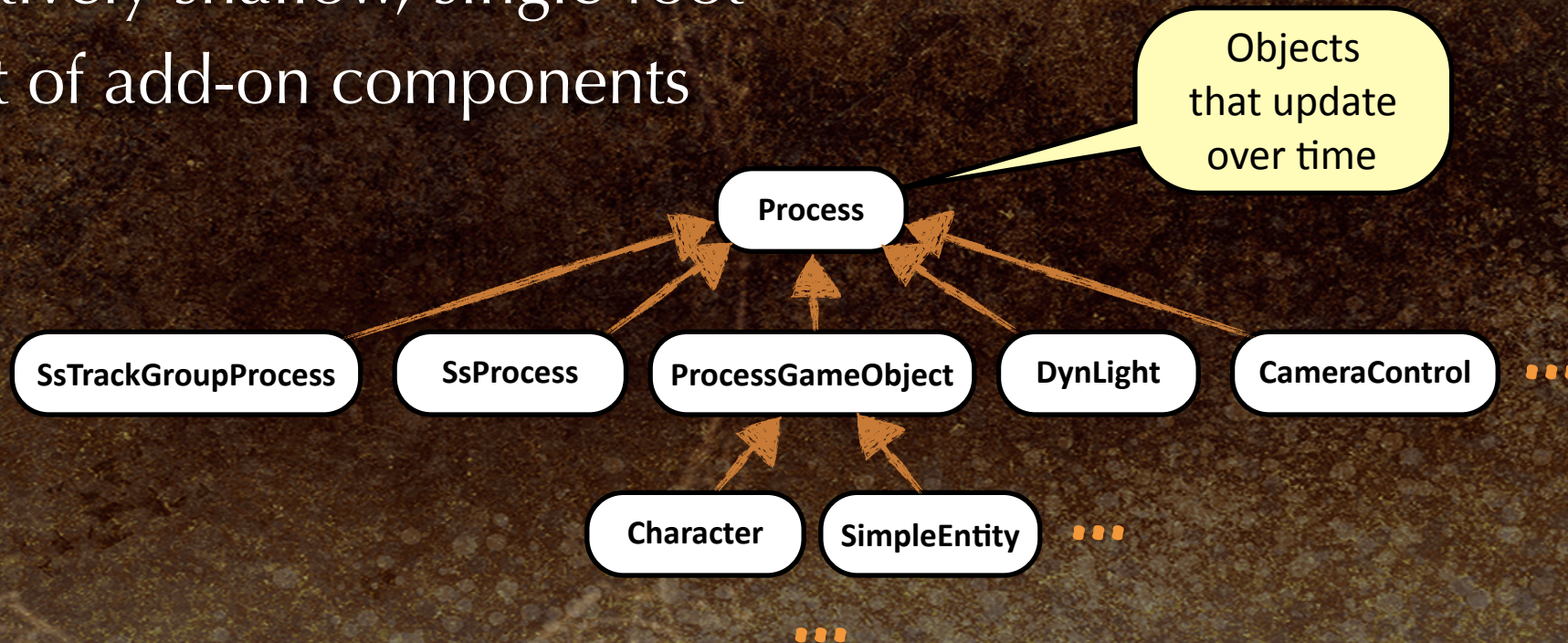
Uncharted Engine's Object Model

- Uncharted class hierarchy:
 - relatively shallow, single-root
 - host of add-on components



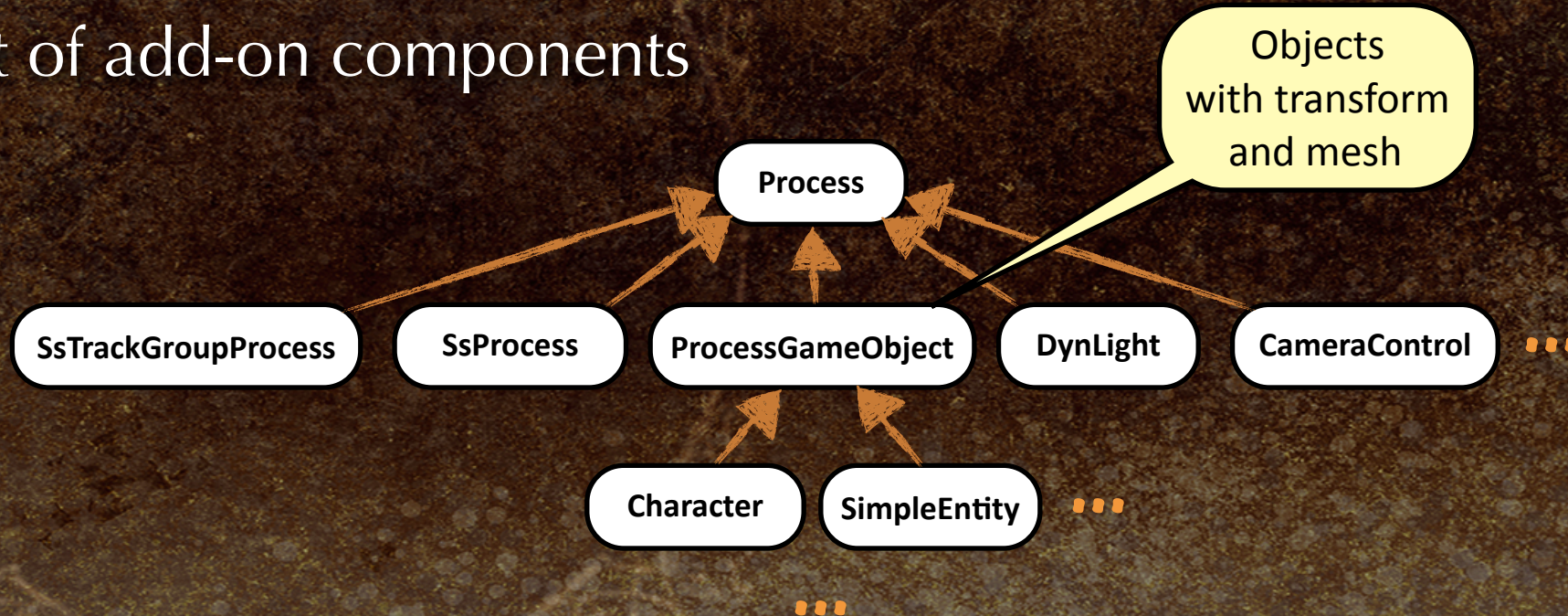
Uncharted Engine's Object Model

- Uncharted class hierarchy:
 - relatively shallow, single-root
 - host of add-on components



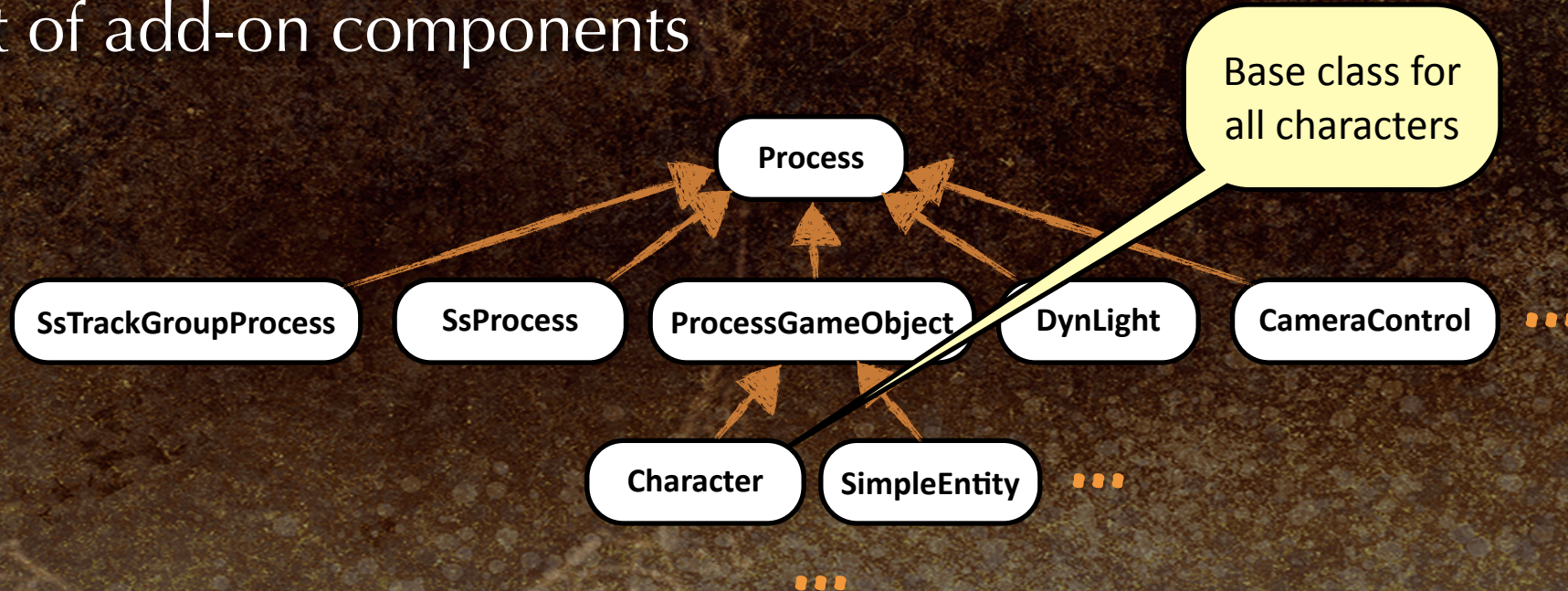
Uncharted Engine's Object Model

- Uncharted class hierarchy:
 - relatively shallow, single-root
 - host of add-on components



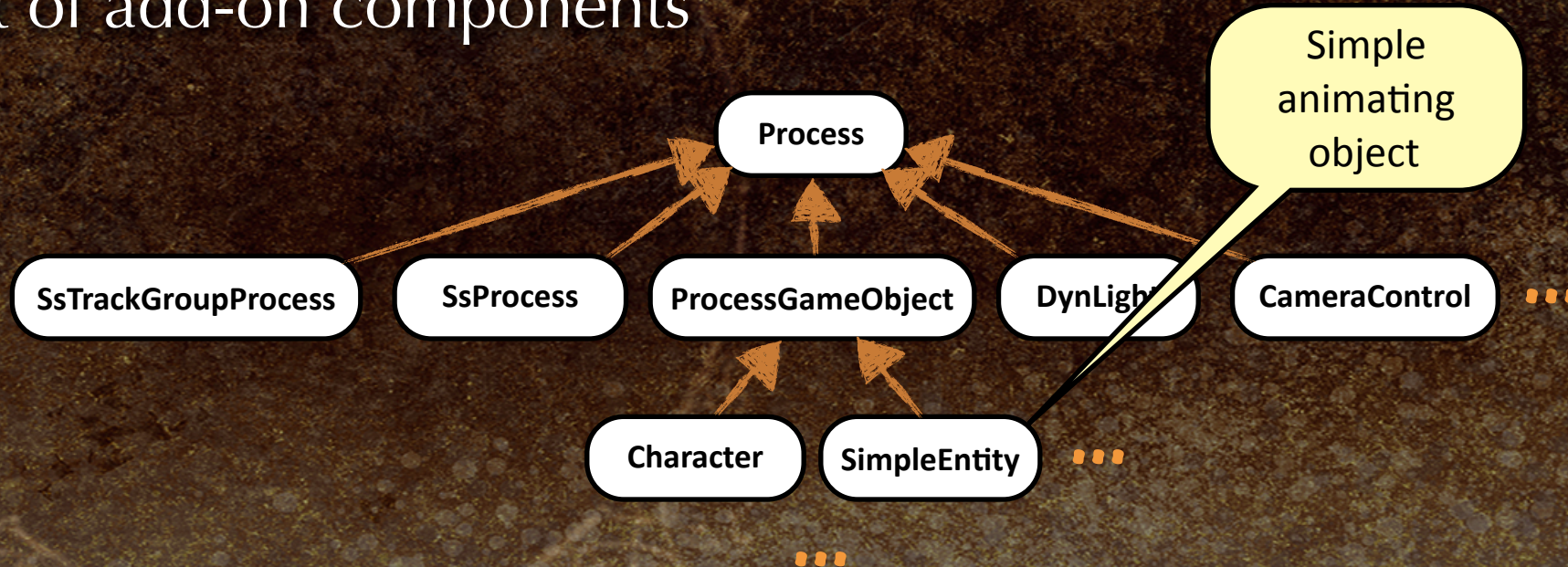
Uncharted Engine's Object Model

- Uncharted class hierarchy:
 - relatively shallow, single-root
 - host of add-on components



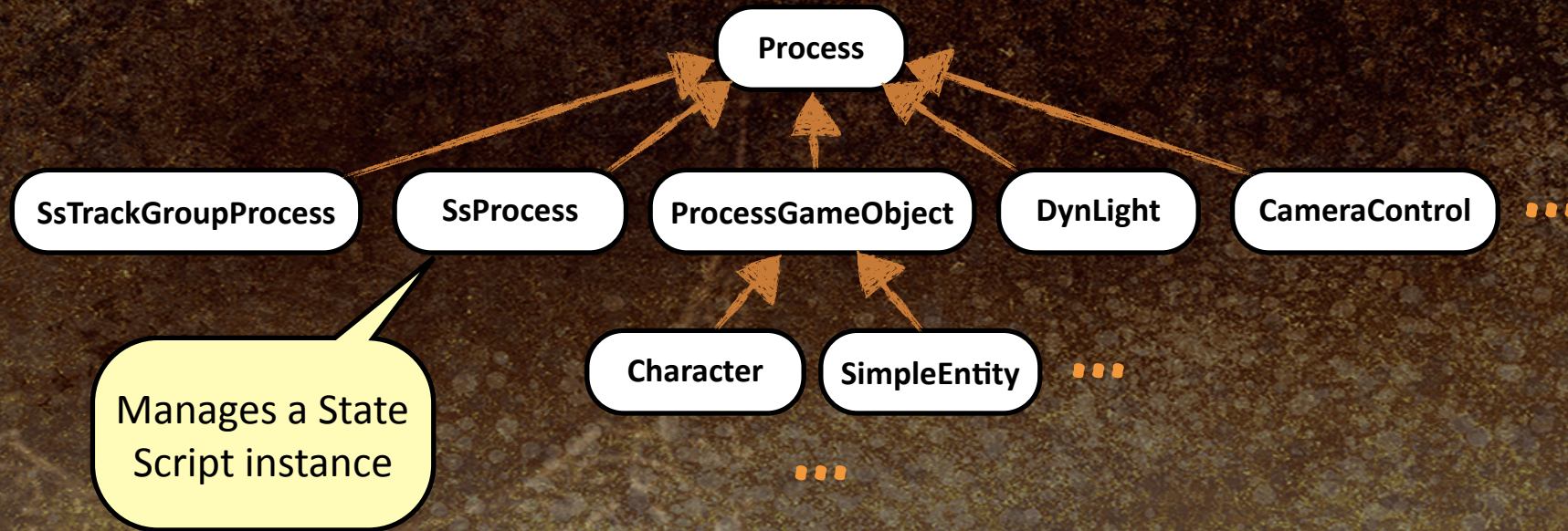
Uncharted Engine's Object Model

- Uncharted class hierarchy:
 - relatively shallow, single-root
 - host of add-on components



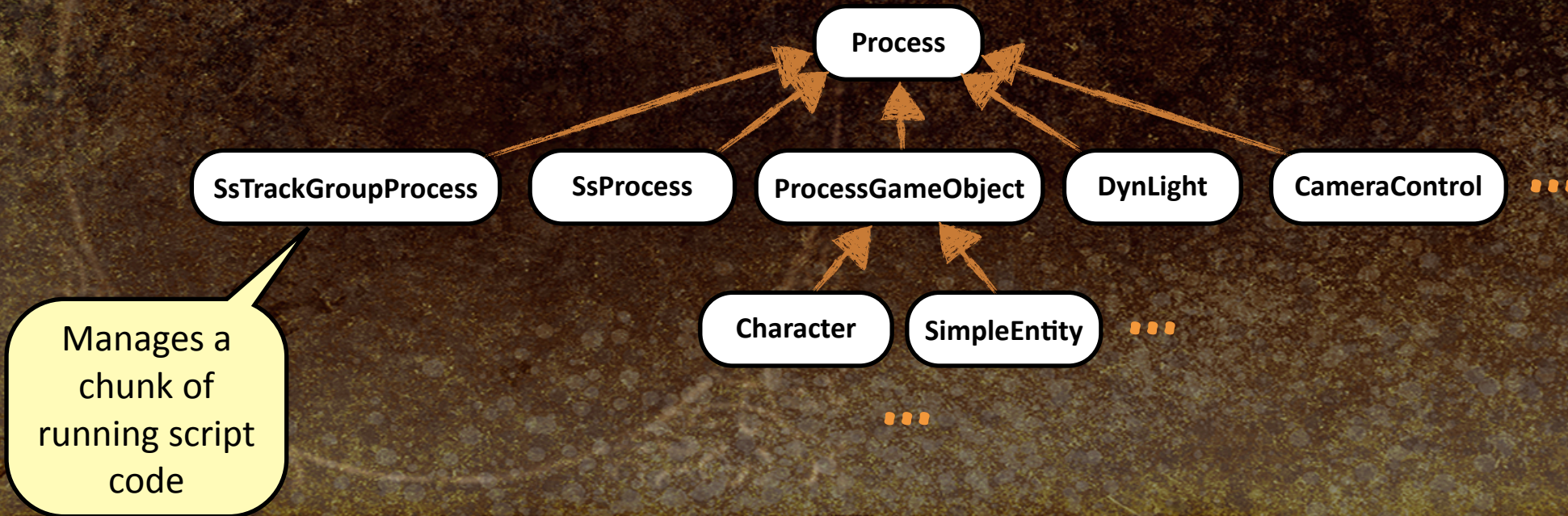
Uncharted Engine's Object Model

- Uncharted class hierarchy:
 - relatively shallow, single-root
 - host of add-on components



Uncharted Engine's Object Model

- Uncharted class hierarchy:
 - relatively shallow, single-root
 - host of add-on components



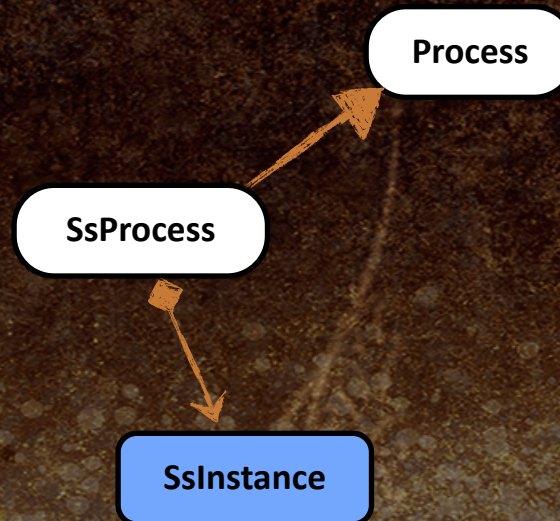
Uncharted Engine's Object Model

- Uncharted class hierarchy:
 - relatively shallow, single-root
 - host of add-on components



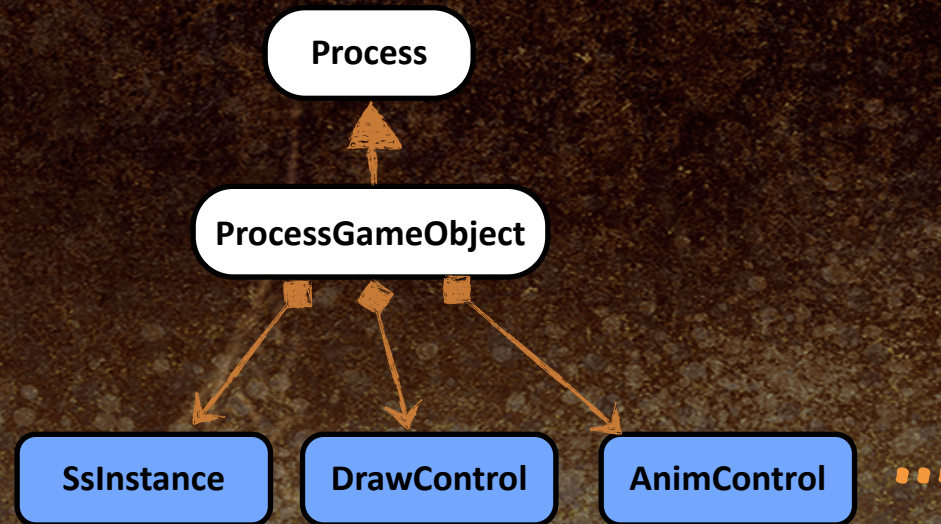
Uncharted Engine's Object Model

- Uncharted class hierarchy:
 - relatively shallow, single-root
 - host of add-on components



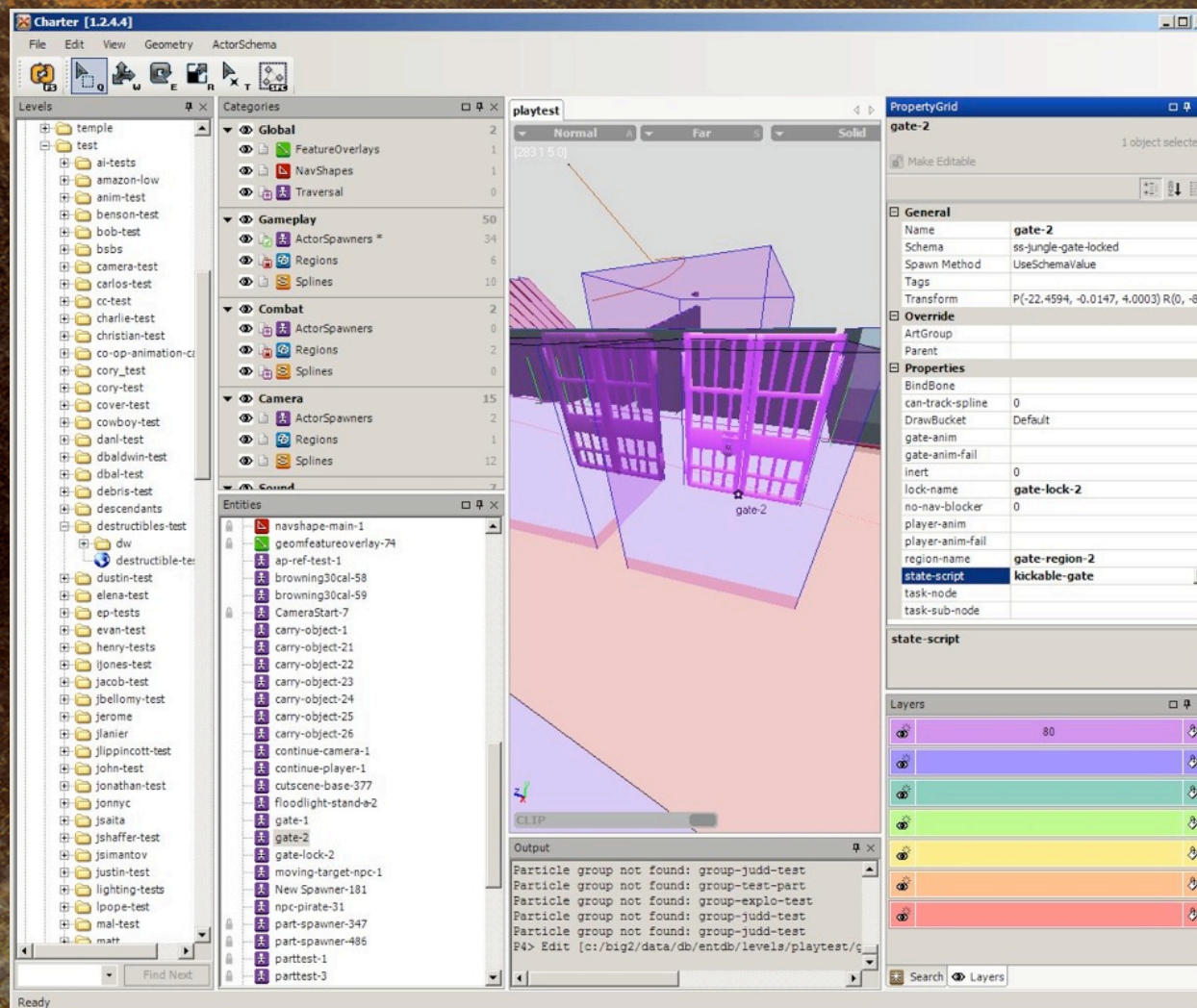
Uncharted Engine's Object Model

- Uncharted class hierarchy:
 - relatively shallow, single-root
 - host of add-on components



Charter

- World editor for *Uncharted* is called **Charter**
 - Place game objects
 - Edit object properties
 - Control level streaming



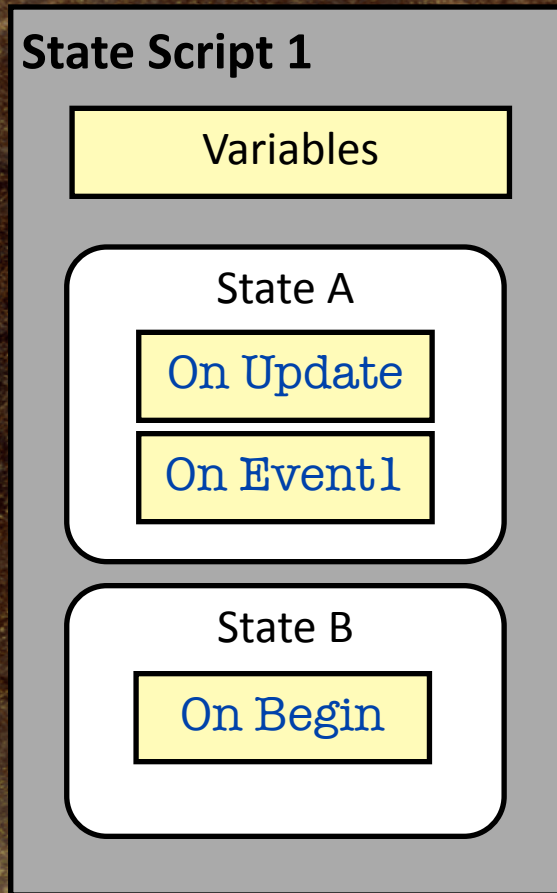
Uncharted State Scripts

- **State scripts** similar in many respects to property-centric model...
 - adds **finite state machine** (FSM) support
 - not specifically tied to “**properties**”
 - coarser-grained (one script per object)
- More like scripted **extension** to existing entity type...
 - ... or a “**director**” that orchestrates actions of other entities

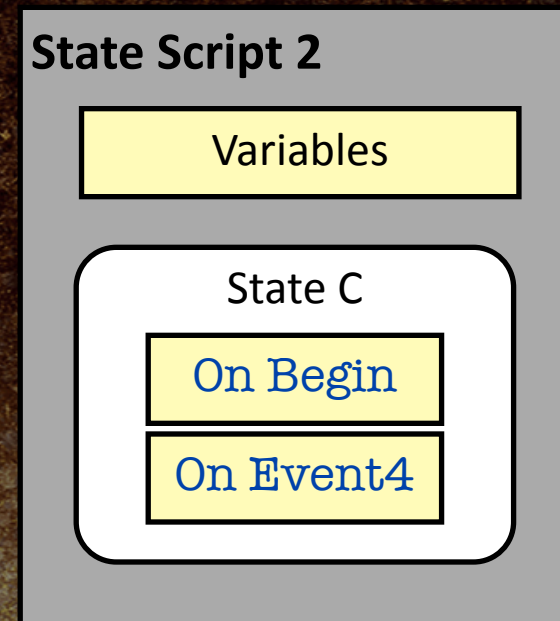
Anatomy of a State Script

- A **state script** is comprised of:
 - **attributes**
 - **states**
- States define object's **behavior** via runtime script code:
 - **response to events**
 - **natural behavior** over time (**update** event)
 - **transitional actions** between states (**begin/end** events)

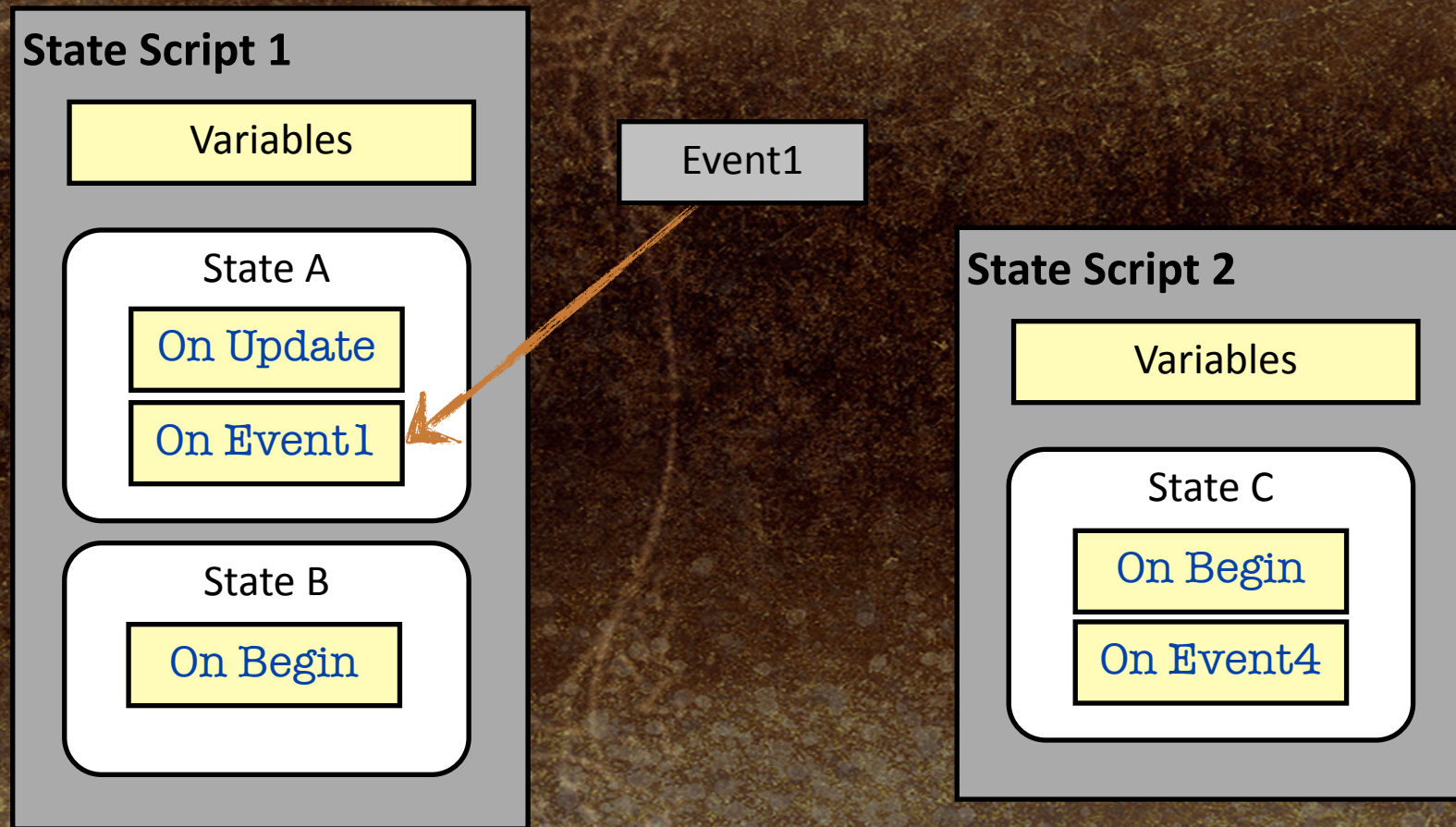
Anatomy of a State Script



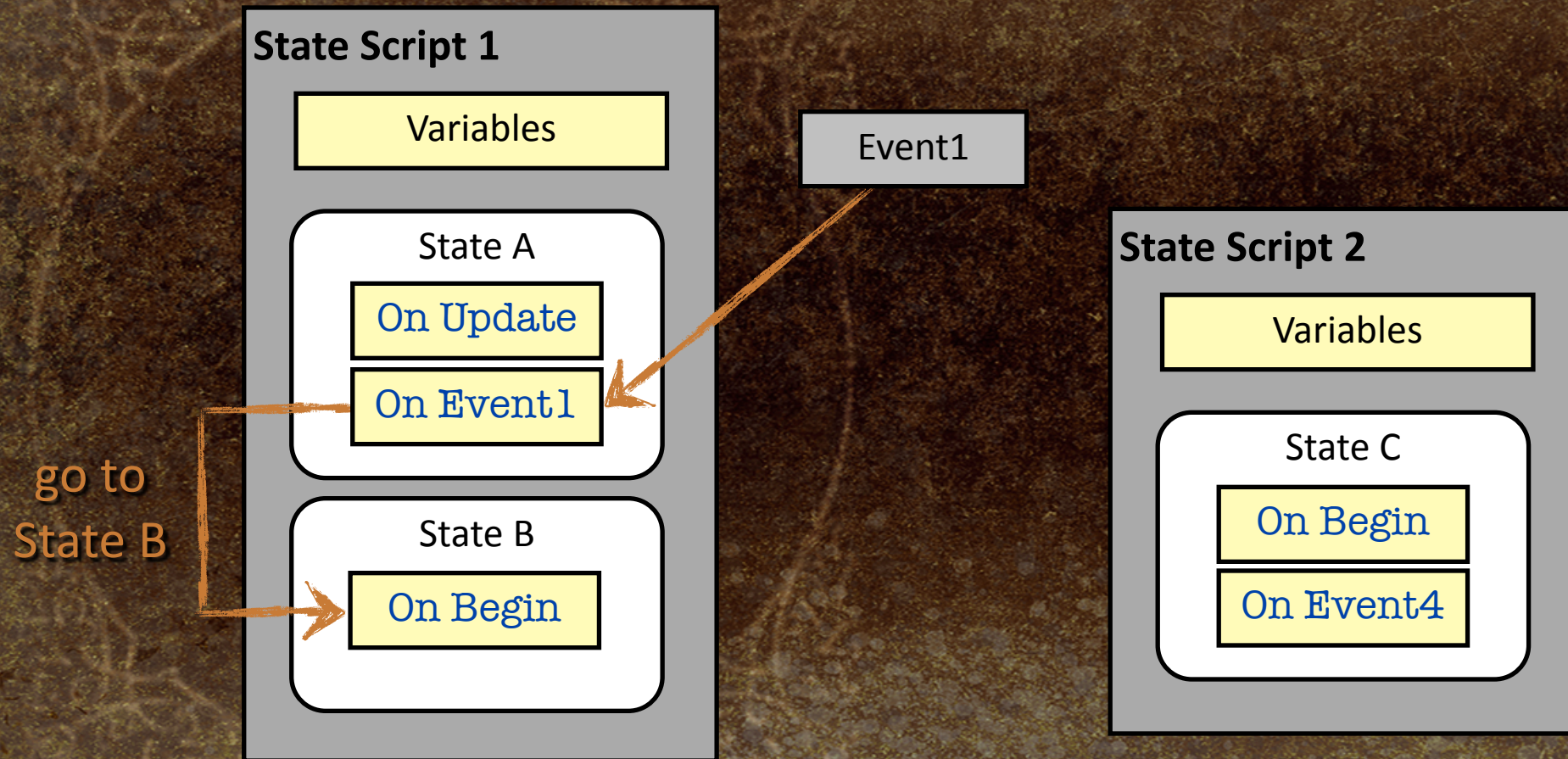
State Script 2



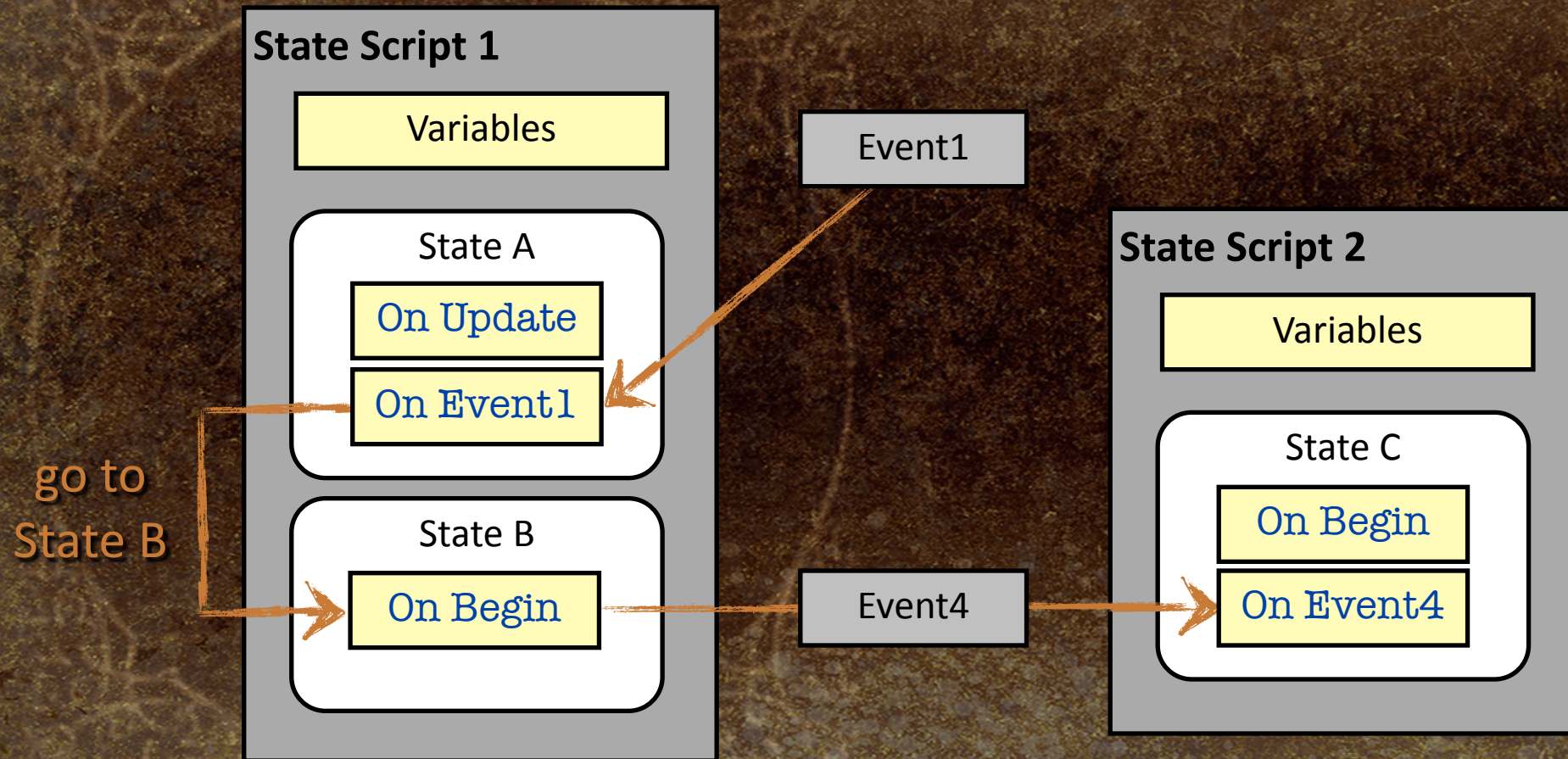
Anatomy of a State Script



Anatomy of a State Script



Anatomy of a State Script



Instantiating a State Script

- Attached to a native (C++) game object:
 - designers extend or modify native C++ object type
 - define entirely new object types

Game
Object

State Script 1

Instantiating a State Script

- Attached to a native (C++) game object:
 - designers extend or modify native C++ object type
 - define entirely new object types
- Attached to a **trigger region**:
 - convex volume
 - detects enter, exit and occupancy



Instantiating State Scripts

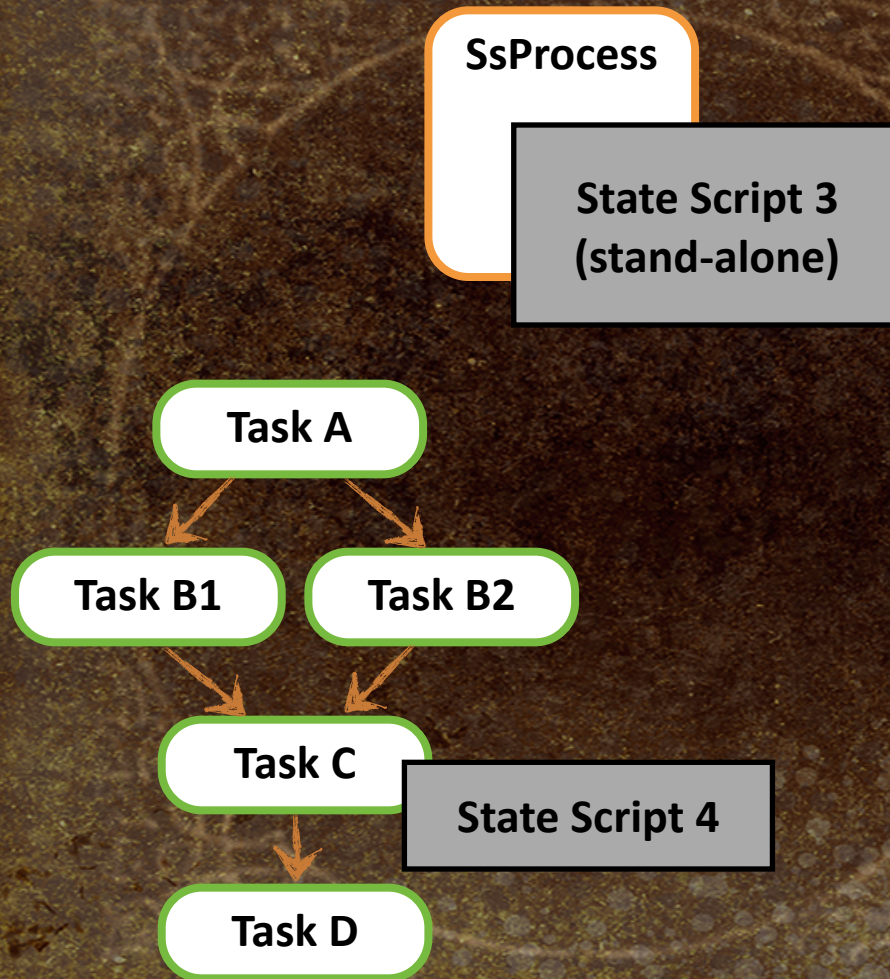
SsProcess

State Script 3
(stand-alone)

- Placed as **stand-alone** object:
 - “director” orchestrates actions of other objects (e.g. IGC)

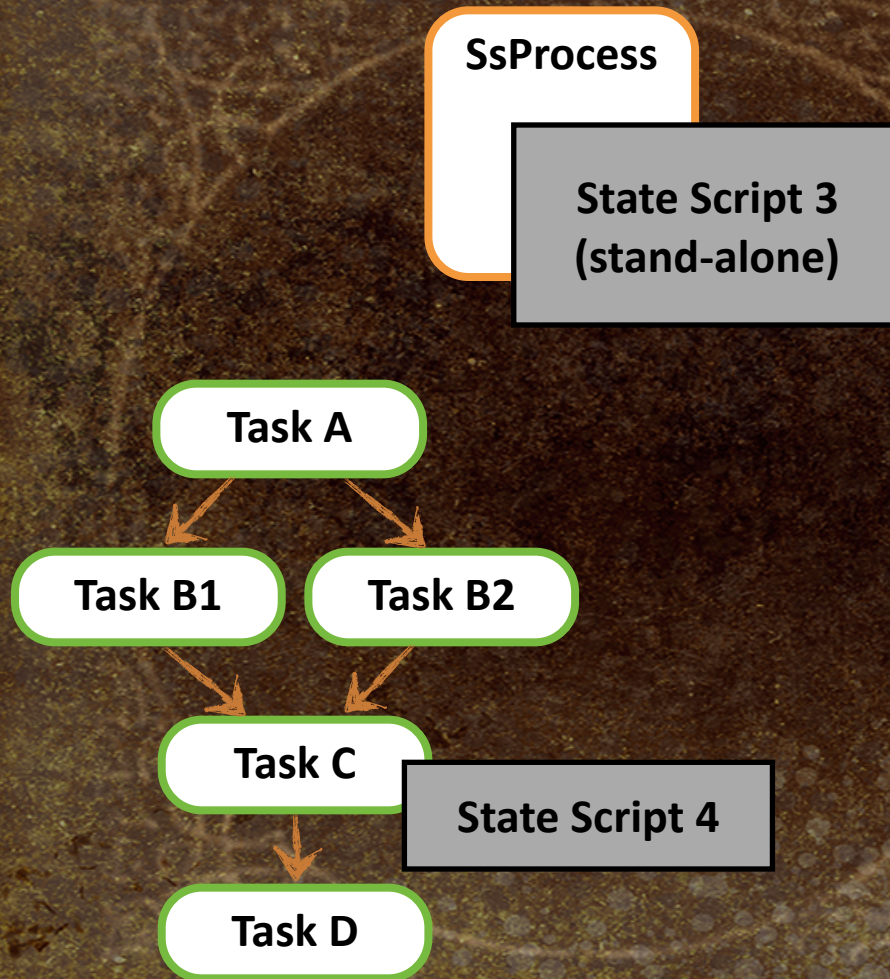


Instantiating State Scripts



- Placed as **stand-alone** object:
 - “director” orchestrates actions of other objects (e.g. IGC)
- Associated with a **task**:
 - task = check point
 - script manages associated task
 - orchestrates AI encounters
 - controls player objectives

Instantiating State Scripts



- Placed as **stand-alone** object:
 - “director” orchestrates actions of other objects (e.g. IGC)
- Associated with a **task**:
 - task = check point
 - script manages associated task
 - orchestrates AI encounters
 - controls player objectives
- **Spawned** by another state script

State Script Syntax





Game Developers Conference 2009

NAUGHTY DOG

State Script Syntax

- State script defined as follows:
 - **Don't let Lisp syntax throw you!**
 - Lisp syntax is all (parentheses)
 - Think C/C++ { } blocks
- Parenthesized blocks highly **context-sensitive** in Lisp/Scheme

```
(define-state-script ("kickable-gate")  
  (state ("locked")  
    ...  
  )  
  (state ("opening")  
    ...  
  )  
  (state ("open")  
    ...  
  )  
)
```


State Script Syntax

- State script defined as follows:
 - **Don't let Lisp syntax throw you!**
 - Lisp syntax is all (parentheses)
 - Think C/C++ { } blocks
- Parenthesized blocks highly **context-sensitive** in Lisp/Scheme

```
(define-state-script ("kickable-gate")  
  (state ("locked")  
    ...  
  )  
  (state ("opening")  
    ...  
  )  
  (state ("open")  
    ...  
  )  
)
```


State Script Syntax

- State script defined as follows:
 - **Don't let Lisp syntax throw you!**
 - Lisp syntax is all (parentheses)
 - Think C/C++ { } blocks
- Parenthesized blocks highly **context-sensitive** in Lisp/Scheme

```
(define-state-script ("kickable-gate")  
  (state ("locked")  
    ...  
  )  
  (state ("opening")  
    ...  
  )  
  (state ("open")  
    ...  
  )  
)
```


State Script Syntax

- State script defined as follows:
 - **Don't let Lisp syntax throw you!**
 - Lisp syntax is all (parentheses)
 - Think C/C++ { } blocks
- Parenthesized blocks highly **context-sensitive** in Lisp/Scheme

```
(define-state-script ("kickable-gate")  
  (state ("locked")  
    ...  
  )  
  (state ("opening")  
    ...  
  )  
  (state ("open")  
    ...  
  )  
)
```


Event Handler Blocks

- Each state contains zero or more **event handler** blocks

```
(define-state-script ("kickable-gate")  
  (state ("locked")  
    (on (event "kick")  
      ... ;; handle "kick" event  
    )  
    (on (begin)  
      ... ;; do when state entered  
    )  
    (on (update)  
      ... ;; do every frame  
    )  
    (on (end)  
      ... ;; do when state exited  
    )  
  )  
)
```


Event Handler Blocks

- Each state contains zero or more **event handler** blocks

```
(define-state-script ("kickable-gate")  
  (state ("locked")  
    (on (event "kick")  
      ... ;; handle "kick" event  
    )  
    (on (begin)  
      ... ;; do when state entered  
    )  
    (on (update)  
      ... ;; do every frame  
    )  
    (on (end)  
      ... ;; do when state exited  
    )  
  )  
)
```


Event Handler Blocks

- Each state contains zero or more **event handler** blocks

```
(define-state-script ("kickable-gate")  
  (state ("locked")  
    (on (event "kick")  
      ... ;; handle "kick" event  
    )  
    (on (begin)  
      ... ;; do when state entered  
    )  
    (on (update)  
      ... ;; do every frame  
    )  
    (on (end)  
      ... ;; do when state exited  
    )  
  )  
)
```


Event Handler Blocks

- Each state contains zero or more **event handler** blocks

```
(define-state-script ("kickable-gate")  
  (state ("locked")  
    (on (event "kick")  
      ... ;; handle "kick" event  
    )  
    (on (begin)  
      ... ;; do when state entered  
    )  
    (on (update)  
      ... ;; do every frame  
    )  
    (on (end)  
      ... ;; do when state exited  
    )  
  )  
)
```


Event Handler Blocks

- Each state contains zero or more **event handler** blocks

```
(define-state-script ("kickable-gate")
  (state ("locked")
    (on (event "kick")
      ... ;; handle "kick" event
    )
    (on (begin)
      ... ;; do when state entered
    )
    (on (update)
      ... ;; do every frame
    )
    (on (end)
      ... ;; do when state exited
    )
  )
)
```


Runtime Script Code

- Runtime script code inside (on ...) blocks
- Simplified Scheme/Lisp
 - Most “commands” are **native** calls into C++ code
 - **Conditional** expressions
 - Simple looping via **label/goto**

```
(state ("locked")
  (on (begin)
    [print-string "Starting idle!"]
    [animate "self" "locked-idle"]
  )
  (on (event "kicked")
    [when [lock-broken?]
      [print-string "BAM!"]
    ]
    [if [task-complete? "wave2"]
      [print-string "Complete!"]
      [print-string "NOT!!!"]
    ]
  )
)
```


Runtime Script Code

- Runtime script code inside (on ...) blocks
- Simplified Scheme/Lisp
 - Most “commands” are **native** calls into C++ code
 - **Conditional** expressions
 - Simple looping via **label/goto**

```
(state ("locked")  
  (on (begin)  
    [print-string "Starting idle!"]  
    [animate "self" "locked-idle"]  
  )  
  (on (event "kicked")  
    [when [lock-broken?]  
      [print-string "BAM!"]  
    ]  
    [if [task-complete? "wave2"]  
      [print-string "Complete!"]  
      [print-string "NOT!!!"]  
    ]  
  )  
)
```


Runtime Script Code

- Runtime script code inside (on ...) blocks
- Simplified Scheme/Lisp
 - Most “commands” are **native** calls into C++ code
 - **Conditional** expressions
 - Simple looping via **label/goto**

```
(state ("locked")  
  (on (begin)  
    [print-string "Starting idle!"]  
    [animate "self" "locked-idle"]  
  )  
  (on (event "kicked")  
    [when [lock-broken?]  
      [print-string "BAM!"]  
    ]  
    [if [task-complete? "wave2"]  
      [print-string "Complete!"]  
      [print-string "NOT!!!"]  
    ]  
  )  
)
```


Runtime Script Code

- Runtime script code inside (on ...) blocks
- Simplified Scheme/Lisp
 - Most “commands” are **native** calls into C++ code
 - **Conditional** expressions
 - Simple looping via **label/goto**

```
(state ("locked")
  (on (begin)
    [print-string "Starting idle!"]
    [animate "self" "locked-idle"]
  )
  (on (event "kicked")
    [when [lock-broken?]
      [print-string "BAM!"]
    ]
    [if [task-complete? "wave2"]
      [print-string "Complete!"]
      [print-string "NOT!!!"]
    ]
  )
)
```


User-Defined Functions

- Script functions can be defined and called by script programmer

```
(defun verbose-anim ((anim string))  
  [print-string "Starting " anim]  
  [animate "self" anim]  
)  
  
(define-state-script ("kickable-gate")  
  (state ("locked")  
    (on (begin)  
      [verbose-anim "locked-idle"]  
    )  
  )  
  ...  
)
```


User-Defined Functions

- Script functions can be defined and called by script programmer

```
(defun verbose-anim ((anim string))  
  [print-string "Starting " anim]  
  [animate "self" anim]  
)
```

```
(define-state-script ("kickable-gate")  
  (state ("locked")  
    (on (begin)  
      [verbose-anim "locked-idle"]  
    )  
  )  
  ...  
)
```


User-Defined Functions

- Script functions can be defined and called by script programmer

```
(defun verbose-anim ((anim string))  
  [print-string "Starting " anim]  
  [animate "self" anim]  
)  
  
(define-state-script ("kickable-gate")  
  (state ("locked")  
    (on (begin)  
      [verbose-anim "locked-idle"]  
    )  
  )  
  ...  
)
```


User-Defined Functions

- Script functions can be defined and called by script programmer

```
(defun verbose-anim ((anim string))  
  [print-string "Starting " anim]  
  [animate "self" anim]  
)  
  
(define-state-script ("kickable-gate")  
  (state ("locked")  
    (on (begin)  
      [verbose-anim "locked-idle"]  
    )  
  )  
  ...  
)
```

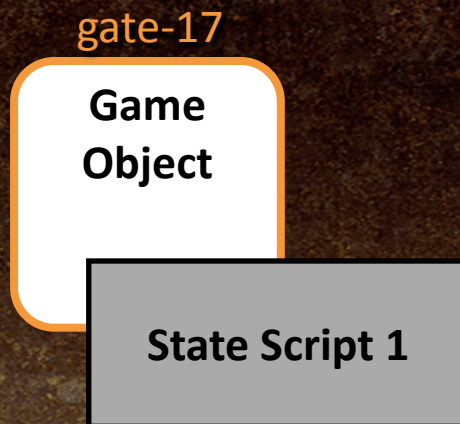

User-Defined Functions

- Script functions can be defined and called by script programmer

```
(defun verbose-anim ((anim string))  
  [print-string "Starting " anim]  
  [animate "self" anim]  
)  
  
(define-state-script ("kickable-gate")  
  (state ("locked")  
    (on (begin)  
      [verbose-anim "locked-idle"]  
    )  
  )  
  ...  
)
```

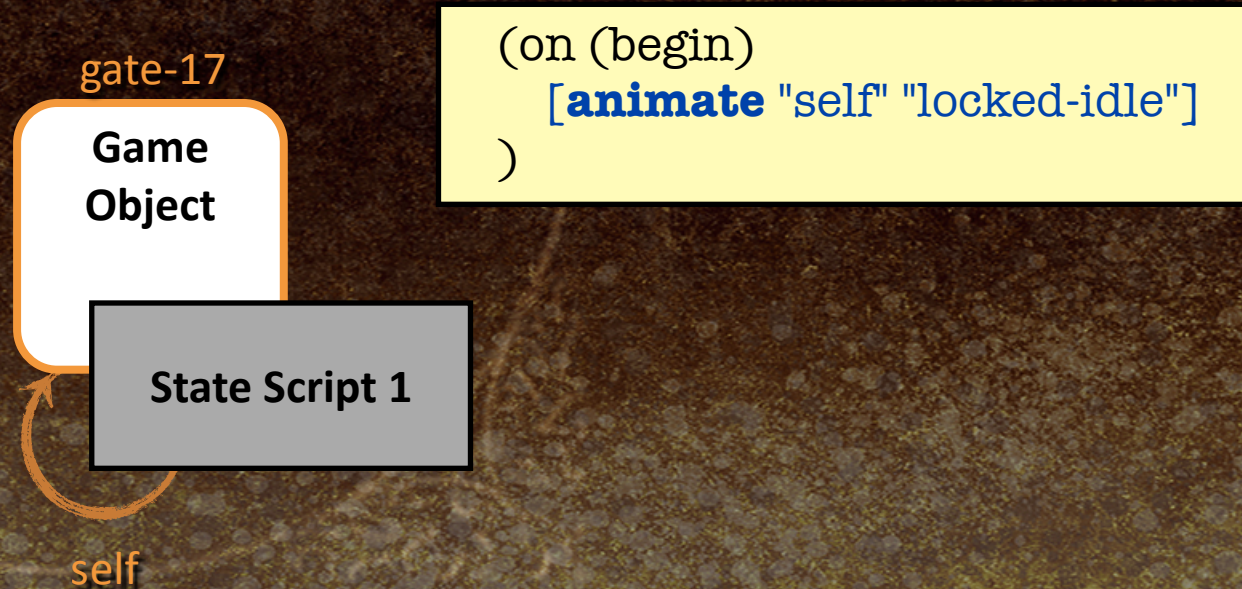

The “Self” Argument

- Any command operating on a game object takes its unique id (UID) as its **first argument**
- Magic UID **"self"** refers to the object to which script is attached



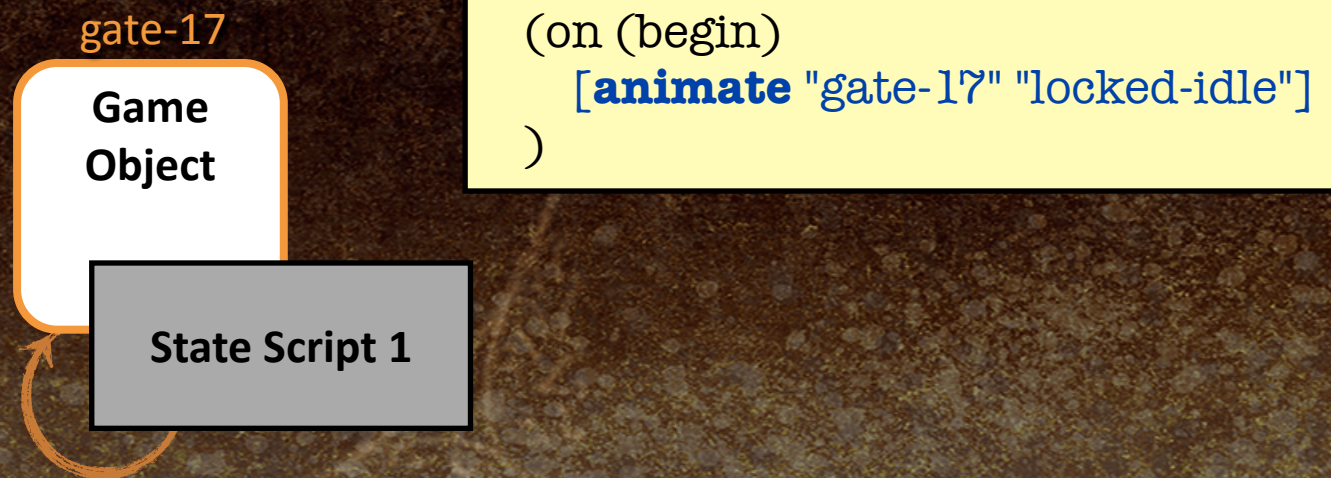
The "Self" Argument

- Any command operating on a game object takes its unique id (UID) as its **first argument**
- Magic UID "**self**" refers to the object to which script is attached



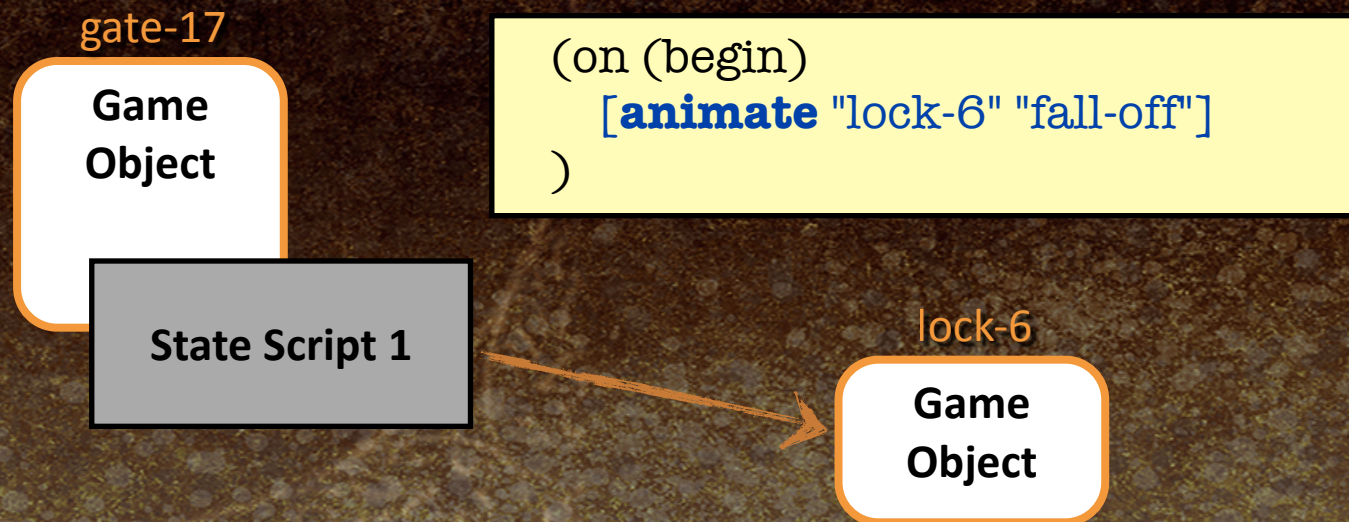
The “Self” Argument

- Any command operating on a game object takes its unique id (UID) as its **first argument**
- Magic UID **"self"** refers to the object to which script is attached



The “Self” Argument

- Any command operating on a game object takes its unique id (UID) as its **first argument**
- Magic UID **"self"** refers to the object to which script is attached



Changing States

- Transitions to other states via (**go** "state-name") command
- State transitions cause current state's (on ...) blocks to be **aborted**
- Use (**on** (**exit**) ...) block for clean-up

```
(define-state-script ("kickable-gate")  
  (state ("locked")  
    ...  
    (on (event "kicked")  
      [when [lock-broken?]  
        [go "opening"]  
      ]  
    )  
  )  
  (state ("opening")  
    (on (begin)  
      ...  
    )  
    ...  
  )  
)
```


Changing States

- Transitions to other states via (**go** "state-name") command
- State transitions cause current state's (on ...) blocks to be **aborted**
- Use (**on** (**exit**) ...) block for clean-up

```
(define-state-script ("kickable-gate")  
  (state ("locked")  
    ...  
    (on (event "kicked")  
      [when [lock-broken?]  
        [go "opening"]]  
      ]  
    )  
  )  
  (state ("opening")  
    (on (begin)  
      ...  
    )  
    ...  
  )  
)
```


Changing States

- Transitions to other states via (**go** "state-name") command
- State transitions cause current state's (on ...) blocks to be **aborted**
- Use (**on** (**exit**) ...) block for clean-up

```
(define-state-script ("kickable-gate")  
  (state ("locked")  
    ...  
    (on (event "kicked")  
      [when [lock-broken?]  
        [go "opening"]  
      ]  
    )  
  )  
  (state ("opening")  
    (on (begin) ←  
      ...  
    )  
    ...  
  )  
)
```


Tracks

- (on ...) blocks contain one or more **tracks**
- A track is a bit like a **thread** or **fiber**
- Tracks can be put to **sleep**
 - Wait for **duration**
 - e.g., wait 5 seconds,
 - wait until frame 23, ...
 - Wait for an action to be **done**
 - duration-agnostic
- Tracks can be **synchronized** via **signals**

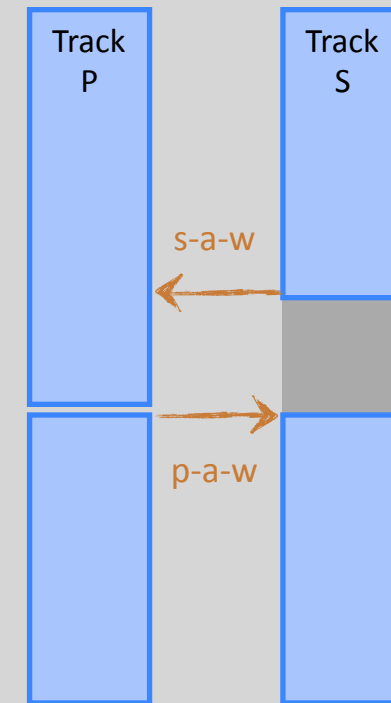


Tracks

```
(state ("shake-hands")  
  (on (begin)  
    (track ("player"))  
    [wait-move-to "player" "waypoint?"]  
    [signal "player-at-waypoint"]  
    [wait-for-signal "sully-at-waypoint"]  
    [wait-animate "player" "shake-sullys-hand"]  
  )  
  (track ("sullivan"))  
  [wait-move-to "sullivan" "waypoint?"]  
  [signal "sully-at-waypoint"]  
  [wait-for-signal "player-at-waypoint"]  
  [wait-animate "sullivan" "shake-drakes-hand"]  
)  
)
```

State "shake-hands"

On Begin

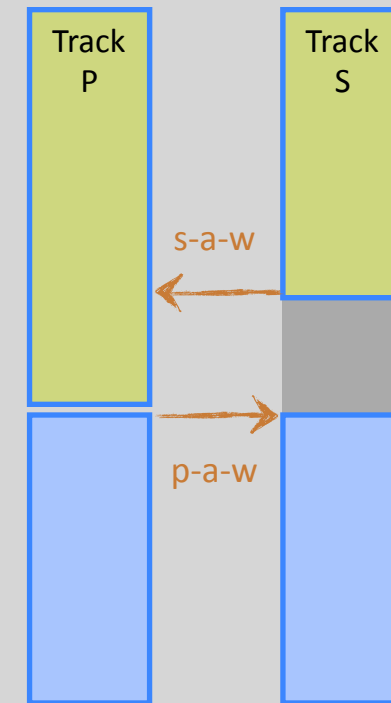


Tracks

```
(state ("shake-hands")  
  (on (begin)  
    (track ("player"))  
    [wait-move-to "player" "waypoint?"]  
    [signal "player-at-waypoint"]  
    [wait-for-signal "sully-at-waypoint"]  
    [wait-animate "player" "shake-sullys-hand"]  
  )  
  (track ("sullivan"))  
  [wait-move-to "sullivan" "waypoint?"]  
  [signal "sully-at-waypoint"]  
  [wait-for-signal "player-at-waypoint"]  
  [wait-animate "sullivan" "shake-drakes-hand"]  
)  
)
```

State "shake-hands"

On Begin

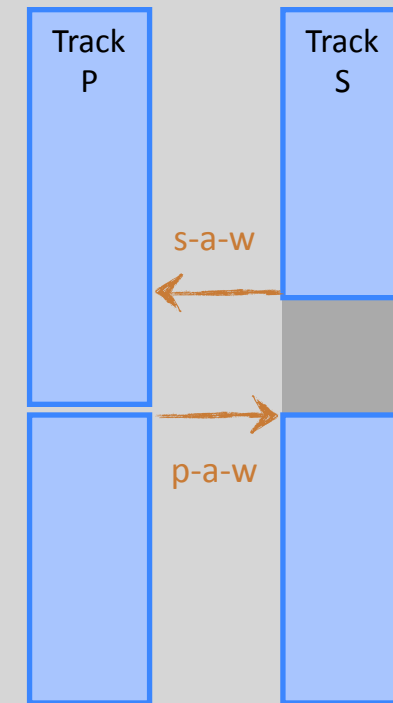


Tracks

```
(state ("shake-hands")  
  (on (begin)  
    (track ("player"))  
    [wait-move-to "player" "waypoint?"]  
    [signal "player-at-waypoint"]  
    [wait-for-signal "sully-at-waypoint"]  
    [wait-animate "player" "shake-sullys-hand"]  
  )  
  (track ("sullivan"))  
  [wait-move-to "sullivan" "waypoint?"]  
  [signal "sully-at-waypoint"]  
  [wait-for-signal "player-at-waypoint"]  
  [wait-animate "sullivan" "shake-drakes-hand"]  
)  
)
```

State "shake-hands"

On Begin

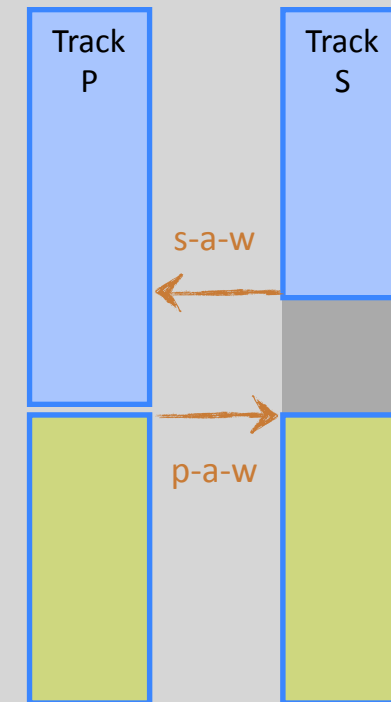


Tracks

```
(state ("shake-hands")  
  (on (begin)  
    (track ("player"))  
    [wait-move-to "player" "waypoint?"]  
    [signal "player-at-waypoint"]  
    [wait-for-signal "sully-at-waypoint"]  
    [wait-animate "player" "shake-sullys-hand"]  
  )  
  (track ("sullivan"))  
  [wait-move-to "sullivan" "waypoint?"]  
  [signal "sully-at-waypoint"]  
  [wait-for-signal "player-at-waypoint"]  
  [wait-animate "sullivan" "shake-drakes-hand"]  
)  
)
```

State "shake-hands"

On Begin

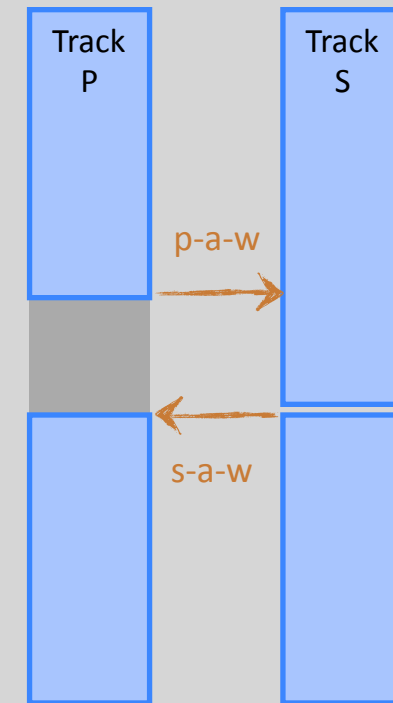


Tracks

```
(state ("shake-hands")  
  (on (begin)  
    (track ("player"))  
    [wait-move-to "player" "waypoint?"]  
    [signal "player-at-waypoint"]  
    [wait-for-signal "sully-at-waypoint"]  
    [wait-animate "player" "shake-sullys-hand"]  
  )  
  (track ("sullivan"))  
  [wait-move-to "sullivan" "waypoint?"]  
  [signal "sully-at-waypoint"]  
  [wait-for-signal "player-at-waypoint"]  
  [wait-animate "sullivan" "shake-drakes-hand"]  
)  
)
```

State "shake-hands"

On Begin



Track Execution Over Time

- Rules for execution of code within a track:
 - Greedily consume instructions sequentially...
 - ... until a `[wait* ...]` command encountered...
 - ... then relinquish control until the action is complete
- NOTE: A `[wait* ...]` command doesn't have to wait; for example:
 - `[wait-seconds 0]`
 - `[wait-npc-move-to "pos-4"]` when she's already there

Options and Variable Declarations

```
(define-state-script ("kickable-gate")
  :initial-state "closed"
  :declarations (decl-list
    (var "num-attempts" :type int32)
    (var "is-locked" :type boolean :default #t)
  )
  (state ("kicked")
    ...
  )
  ...
)
```

- Options declared at top of script:
 - initial state
 - variable declarations
 - debugging options

Options and Variable Declarations

```
(define-state-script ("kickable-gate")  
  :initial-state "closed"  
  :declarations (decl-list  
    (var "num-attempts" :type int32)  
    (var "is-locked" :type boolean :default #t)  
  )  
  (state ("kicked")  
    ...  
  )  
  ...  
)
```

- Options declared at top of script:
 - **initial state**
 - **variable declarations**
 - **debugging options**

Options and Variable Declarations

```
(define-state-script ("kickable-gate")  
  :initial-state "closed"  
  :declarations (decl-list  
    (var "num-attempts" :type int32)  
    (var "is-locked" :type boolean :default #t)  
  )  
  (state ("kicked")  
    ...  
  )  
  ...  
)
```

- Options declared at top of script:
 - **initial state**
 - **variable declarations**
 - **debugging options**

Manipulating Variables

- Simple commands for reading and writing variables
- Lisp/Scheme uses **prefix notation** (a.k.a. Polish notation)
 - `[+ a b]` calculates $(a + b)$
 - `[set "x" [+ 1 [get "x"]]]` increments variable "x"

```
(define-state-script ("kickable-gate")
  :initial-state "closed"
  :declarations (decl-list
    (var "num-attempts" :type int32)
    (var "is-locked" :type boolean :default #t)
  )
  (state ("kicked")
    (on (begin)
      [when [get-boolean "is-locked"]
        [set-int32 "num-attempts"
          [+ 1 [get-int32 "num-attempts"]]]
        ]
      [wait-animate "self" "kick-failure"]
      [go "closed"]
    ]
    ;; else...
    [wait-animate "self" "kick-success"]
    [go "open"]
  )
)
```


Manipulating Variables

- Simple commands for reading and writing variables
- Lisp/Scheme uses **prefix notation** (a.k.a. Polish notation)
 - `[+ a b]` calculates $(a + b)$
 - `[set "x" [+ 1 [get "x"]]]` increments variable "x"

```
(define-state-script ("kickable-gate")
  :initial-state "closed"
  :declarations (decl-list
    (var "num-attempts" :type int32)
    (var "is-locked" :type boolean :default #t)
  )
  (state ("kicked")
    (on (begin)
      [when [get-boolean "is-locked"]
        [set-int32 "num-attempts"
          [+ 1 [get-int32 "num-attempts"]]]
        ]
      [wait-animate "self" "kick-failure"]
      [go "closed"]
    ]
    ;; else...
    [wait-animate "self" "kick-success"]
    [go "open"]
  )
)
```


Manipulating Variables

- Simple commands for reading and writing variables
- Lisp/Scheme uses **prefix notation** (a.k.a. Polish notation)
 - `[+ a b]` calculates $(a + b)$
 - `[set "x" [+ 1 [get "x"]]]` increments variable "x"

```
(define-state-script ("kickable-gate")
  :initial-state "closed"
  :declarations (decl-list
    (var "num-attempts" :type int32)
    (var "is-locked" :type boolean :default #t)
  )
  (state ("kicked")
    (on (begin)
      [when [get-boolean "is-locked"]
        [set-int32 "num-attempts"
          [+ 1 [get-int32 "num-attempts"]]]
        ]
      [wait-animate "self" "kick-failure"]
      [go "closed"]
    ]
    ;; else...
    [wait-animate "self" "kick-success"]
    [go "open"]
  )
)
```


Manipulating Variables

- Simple commands for reading and writing variables
- Lisp/Scheme uses **prefix notation** (a.k.a. Polish notation)
 - `[+ a b]` calculates $(a + b)$
 - `[set "x" [+ 1 [get "x"]]]` increments variable "x"

```
(define-state-script ("kickable-gate")
  :initial-state "closed"
  :declarations (decl-list
    (var "num-attempts" :type int32)
    (var "is-locked" :type boolean :default #t)
  )
  (state ("kicked")
    (on (begin)
      [when [get-boolean "is-locked"]
        [set-int32 "num-attempts"
          [+ 1 [get-int32 "num-attempts"]]]
        ]
      [wait-animate "self" "kick-failure"]
      [go "closed"]
    ]
    ;; else...
    [wait-animate "self" "kick-success"]
    [go "open"]
  )
)
```

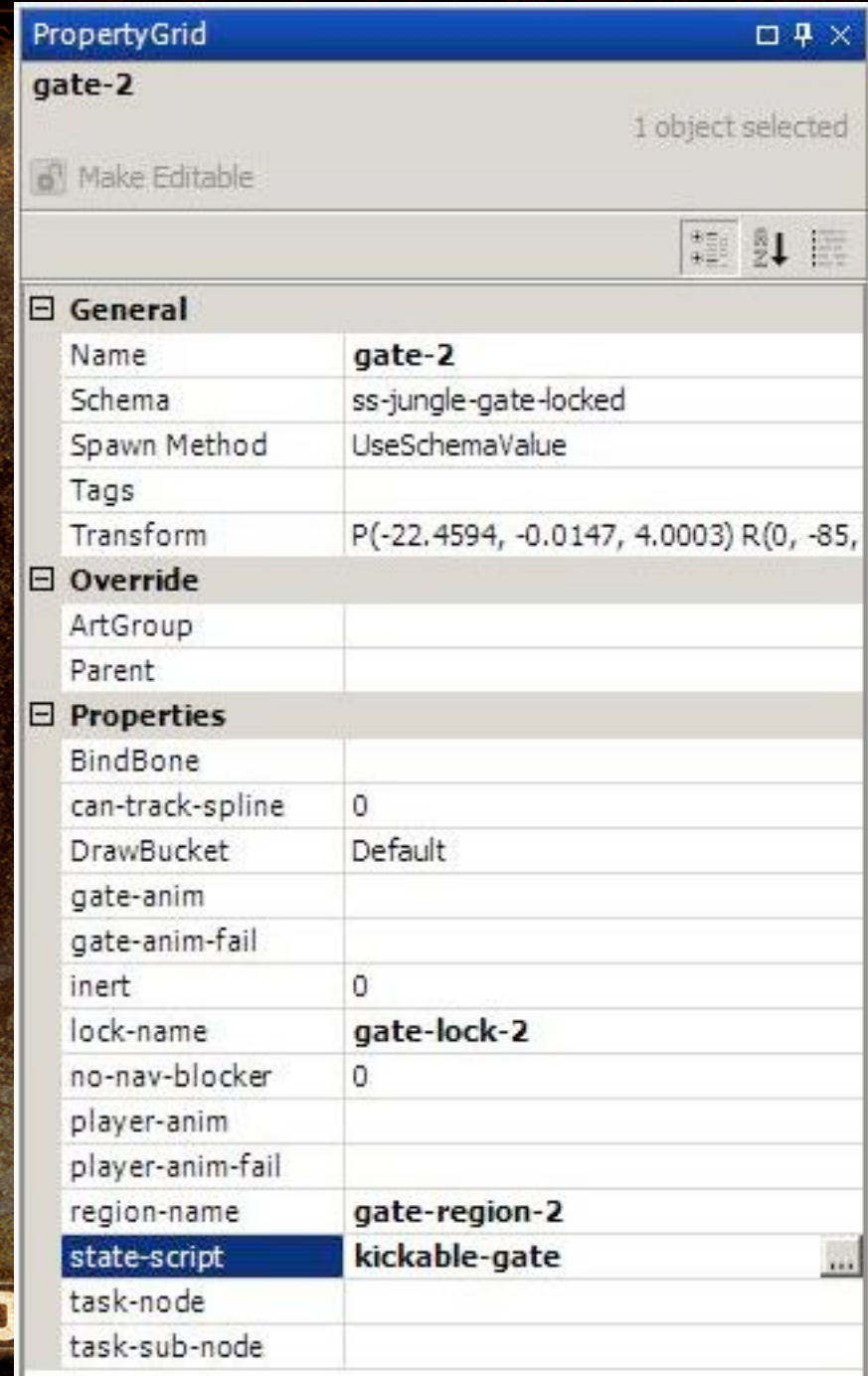

Manipulating Variables

- Simple commands for reading and writing variables
- Lisp/Scheme uses **prefix notation** (a.k.a. Polish notation)
 - `[+ a b]` calculates $(a + b)$
 - `[set "x" [+ 1 [get "x"]]]` increments variable "x"

```
(define-state-script ("kickable-gate")
  :initial-state "closed"
  :declarations (decl-list
    (var "num-attempts" :type int32)
    (var "is-locked" :type boolean :default #t)
  )
  (state ("kicked")
    (on (begin)
      [when [get-boolean "is-locked"]
        [set-int32 "num-attempts"
          [+ 1 [get-int32 "num-attempts"]]]
        ]
      [wait-animate "self" "kick-failure"]
      [go "closed"]
    ]
    ;; else...
    [wait-animate "self" "kick-success"]
    [go "open"]
  )
)
```

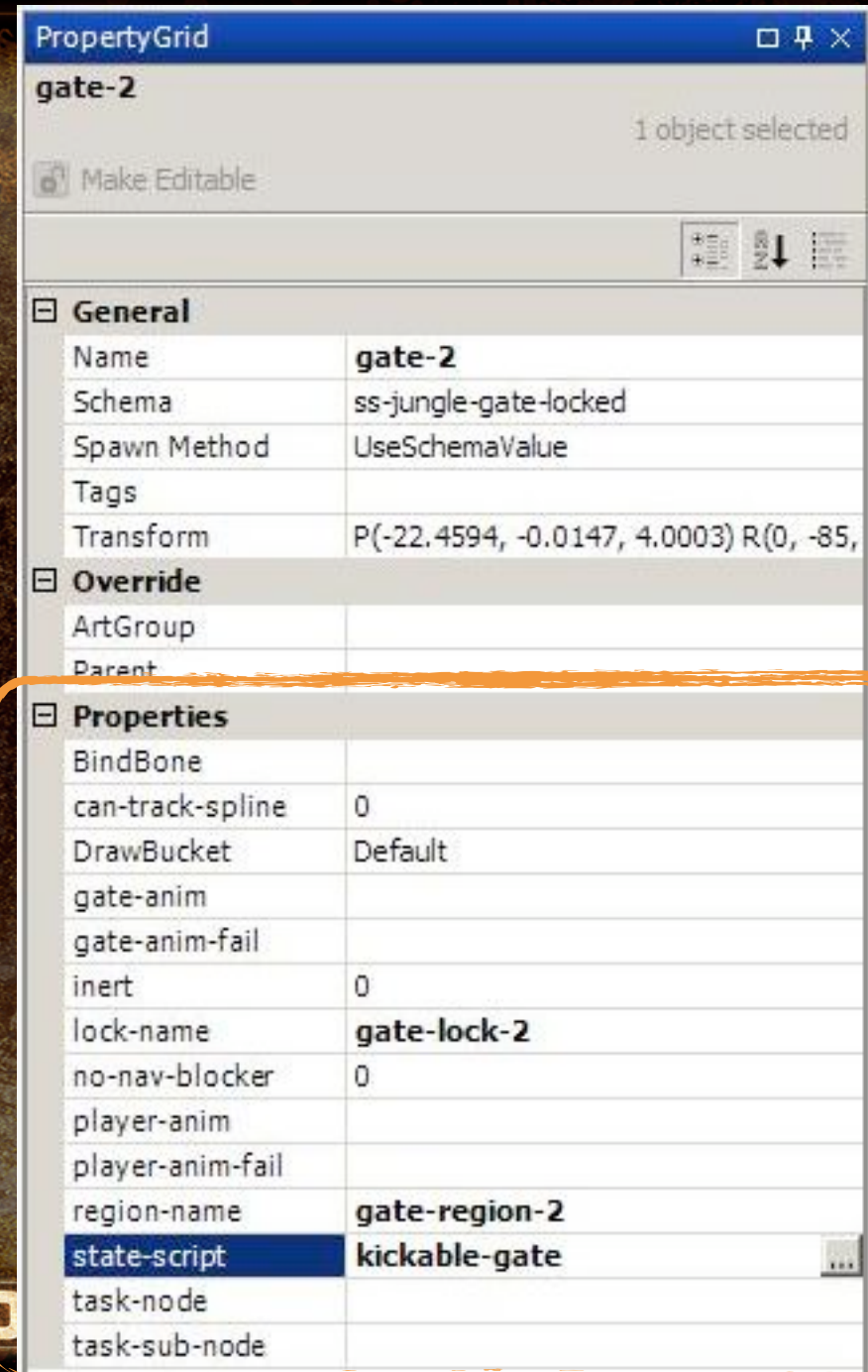

Configuration Parameters

- All game objects have **properties** (key-value pairs)
 - Property values edited in Charter



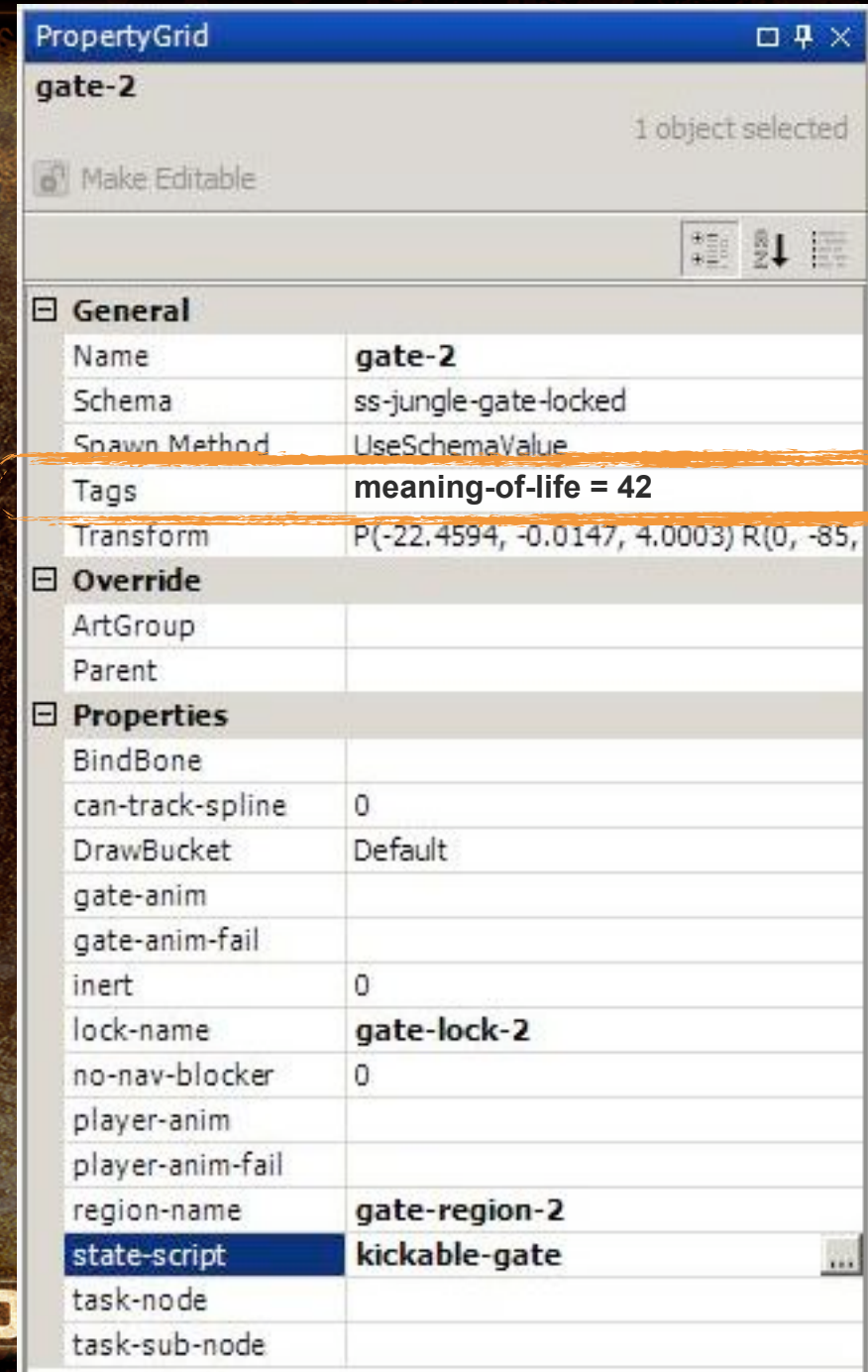
Configuration Parameters

- All game objects have **properties** (key-value pairs)
 - Property values edited in Charter



Configuration Parameters

- Designers can create their own free-form properties, called **tags**
 - Simply type “key = value” in Tags field



Configuration Parameters

- State scripts have read-only access to game object properties...
 - ... and free-form tags

Configuration Parameters

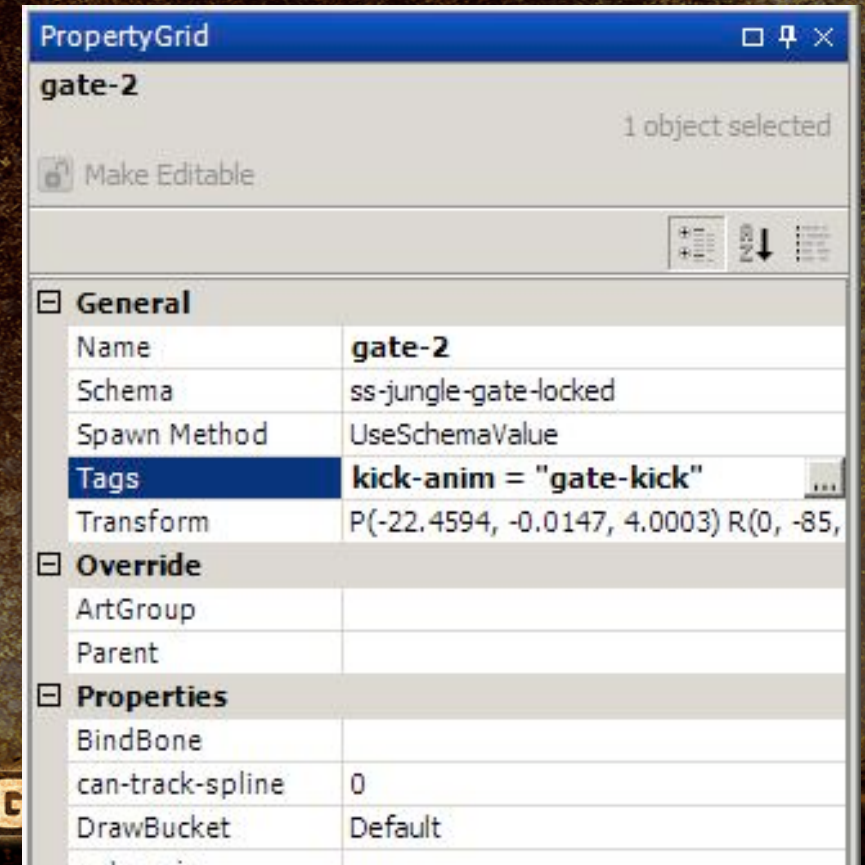
- State scripts have read-only access to game object properties...
 - ... and free-form tags

```
(define-state-script ("kickable-gate")  
  (state ("kicked")  
    (on (begin)  
      [wait-animate "self"  
        [tag-string "kick-anim"]  
      ]  
    )  
    ...  
  )
```


Configuration Parameters

- State scripts have read-only access to game object properties...
 - ... and free-form tags

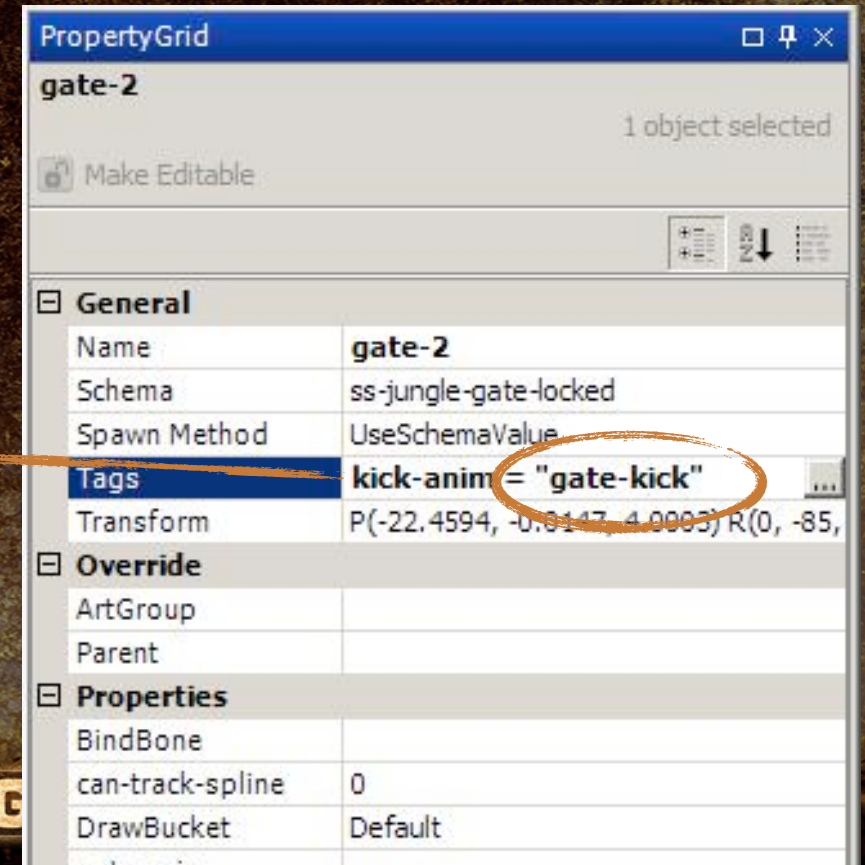
```
(define-state-script ("kickable-gate")  
  (state ("kicked")  
    (on (begin)  
      [wait-animate "self"  
        [tag-string "kick-anim"]  
      ]  
    )  
    ...  
  )  
)
```



Configuration Parameters

- State scripts have read-only access to game object properties...
 - ... and free-form tags

```
(define-state-script ("kickable-gate")  
  (state ("kicked")  
    (on (begin)  
      [wait-animate "self"  
        [tag-string "kick-anim"]  
      ]  
    )  
    ...  
  )  
)
```



Data Definition and Runtime

- Offline **data-definition** language (Scheme) is intermingled with **runtime** code

```
(define-state-script ("kickable-gate")
  (state ("locked")
    (on (begin)
      [print-string "Starting idle!"]
      [animate "self" "locked-idle"]
    )
    (on (event "kicked")
      [when [lock-broken?]
        [print-string "BAM!"]
      ]
      [if [task-complete? "wave2"]
        [print-string "Complete!"]
        [print-string "NOT!!!"]
      ]
    )
  )
  ...
)
```


Data Definition and Runtime

- Offline **data-definition** language (Scheme) is intermingled with **runtime** code

```
(define-state-script ("kickable-gate")
  (state ("locked")
    (on (begin)
      [print-string "Starting idle!"]
      [animate "self" "locked-idle"]
    )
    (on (event "kicked")
      [when [lock-broken?]
        [print-string "BAM!"]
      ]
      [if [task-complete? "wave2"]
        [print-string "Complete!"]
        [print-string "NOT!!!"]
      ]
    )
  )
  )
  ...
)
```


Data Definition and Runtime

- Really no different than the distinction between **declarations** and **definitions** in C++

```
class Vector3
{
private:
    float x, y, z;
public:
    float Dot(const Vector3& b)
    {
        return (x * b.x
                + y * b.y
                + z * b.z);
    }
    ...
};
```


Data Definition and Runtime

- Really no different than the distinction between **declarations** and **definitions** in C++

```
class Vector3
{
private:
    float x, y, z;
public:
    float Dot(const Vector3& b)
    {
        return (x * b.x
                + y * b.y
                + z * b.z);
    }
    ...
};
```


Case Studies



Custom Object Type: Breakable Sign

Custom Object Type: Breakable Sign

Custom Object Type: Breakable Sign

```
(define-state-script ("falling-sign")  
  (state ("untouched")  
    (on (update)  
      [when [task-complete? "wz-post-combat"]  
        [go "fallen"]  
      ]  
    )  
    (on (event "hanging-from")  
      [go "breaking"]  
    )  
  )  
  ...
```



Custom Object Type: Breakable Sign

```
(define-state-script ("falling-sign")  
  (state ("untouched")  
    (on (update)  
      [when [task-complete? "wz-post-combat"]  
        [go "fallen"]  
      ]  
    )  
    (on (event "hanging-from")  
      [go "breaking"]  
    )  
  )  
  ...
```



Custom Object Type: Breakable Sign

```
(define-state-script ("falling-sign")  
  (state ("untouched")  
    (on (update)  
      [when [task-complete? "wz-post-combat"]  
        [go "fallen"]  
      ]  
    )  
    (on (event "hanging-from")  
      [go "breaking"]  
    )  
  )  
  ...
```



Custom Object Type: Breakable Sign



```
(state ("breaking")
  (on (begin)
    [spawn-particles-at-joint "self"
      "hinge"
      "sign-break-dust"]
    [wait-animate "self" "sign-break"]
    [go "fallen"]
  )
)

(state ("fallen")
  (on (begin)
    [animate "self" "sign-broken"] ;; looping
  )
)
)
```


Custom Object Type: Breakable Sign



```
(state ("breaking")
  (on (begin)
    [spawn-particles-at-joint "self"
      "hinge"
      "sign-break-dust"]
    [wait-animate "self" "sign-break"]
    [go "fallen"]
  )
)
(state ("fallen")
  (on (begin)
    [animate "self" "sign-broken"] ;; looping
  )
)
)
```


Custom Object Type: Breakable Sign



```
(state ("breaking")
  (on (begin)
    [spawn-particles-at-joint "self"
      "hinge"
      "sign-break-dust"]
    [wait-animate "self" "sign-break"]
    [go "fallen"]
  )
)
(state ("fallen")
  (on (begin)
    [animate "self" "sign-broken"] ;; looping
  )
)
)
```


Custom Object Type: Breakable Sign



```
(state ("breaking")
  (on (begin)
    [spawn-particles-at-joint "self"
      "hinge"
      "sign-break-dust"]
    [wait-animate "self" "sign-break"]
    [go "fallen"]
  )
)
(state ("fallen")
  (on (begin)
    [animate "self" "sign-broken"] ;; looping
  )
)
)
```


Custom Object Type: Breakable Sign



```
(state ("breaking")
  (on (begin)
    [spawn-particles-at-joint "self"
      "hinge"
      "sign-break-dust"]
    [wait-animate "self" "sign-break"]
    [go "fallen"]
  )
)
(state ("fallen")
  (on (begin)
    [animate "self" "sign-broken"] ;; looping
  )
)
)
```


Custom Object Type: Breakable Sign



```
(state ("breaking")
  (on (begin)
    [spawn-particles-at-joint "self"
      "hinge"
      "sign-break-dust"]
    [wait-animate "self" "sign-break"]
    [go "fallen"]
  )
)
(state ("fallen")
  (on (begin)
    [animate "self" "sign-broken"] ;; looping
  )
)
)
```


Custom Object Type: Breakable Sign



```
(state ("breaking")
  (on (begin)
    [spawn-particles-at-joint "self"
      "hinge"
      "sign-break-dust"]
    [wait-animate "self" "sign-break"]
    [go "fallen"]
  )
)
(state ("fallen")
  (on (begin)
    [animate "self" "sign-broken"] ;; looping
  )
)
)
```


Custom Object Type: Breakable Sign



```
(state ("breaking")
  (on (begin)
    [spawn-particles-at-joint "self"
      "hinge"
      "sign-break-dust"]
    [wait-animate "self" "sign-break"]
    [go "fallen"]
  )
)
(state ("fallen")
  (on (begin)
    [animate "self" "sign-broken"] ;; looping
  )
)
)
```


Custom Object Type: Breakable Sign



```
(state ("breaking")
  (on (begin)
    [spawn-particles-at-joint "self"
      "hinge"
      "sign-break-dust"]
    [wait-animate "self" "sign-break"]
    [go "fallen"])
  )
)
(state ("fallen")
  (on (begin)
    [animate "self" "sign-broken"] ;; looping
  )
)
)
```


Custom Object Type: Breakable Sign

```
(define-state-script ("falling-sign")
  (state ("untouched")
    (on (update)
      [when [task-complete? "wz-post-combat"]
        [go "fallen"]
      ]
    )
    (on (event "hanging-from")
      [go "breaking"]
    )
  )
  ...)
```

```
(state ("breaking")
  (on (begin)
    [spawn-particles-at-joint "self"
      "hinge"
      "sign-break-dust"]
    [wait-animate "self" "sign-break"]
    [go "fallen"]
  )
)
(state ("fallen")
  (on (begin)
    [animate "self" "sign-broken"] ;; looping
  )
)
)
```


Custom Object Type: Breakable Sign

```
(define-state-script ("falling-sign")  
  (state ("untouched")  
    (on (update)  
      [when [task-complete? "wz-post-combat"]  
        [go "fallen"]  
      ]  
    )  
    (on (event "hanging-from")  
      [go "breaking"]  
    )  
  )  
  ...  
)
```

```
(state ("breaking")  
  (on (begin)  
    [spawn-particles-at-joint "self"  
      "hinge"  
      "sign-break-dust"]  
    [wait-animate "self" "sign-break"]  
    [go "fallen"]  
  )  
)  
(state ("fallen")  
  (on (begin)  
    [animate "self" "sign-broken"] ;; looping  
  )  
)  
)
```


Custom Object Type: Breakable Sign

```
(define-state-script ("falling-sign")  
  (state ("untouched")  
    (on (update)  
      [when [task-complete? "wz-post-combat"]  
        [go "fallen"]  
      ]  
    )  
    (on (event "hanging-from")  
      [go "breaking"]  
    )  
  )  
  ...  
)
```

```
(state ("breaking")  
  (on (begin)  
    [spawn-particles-at-joint "self"  
      "hinge"  
      "sign-break-dust"]  
    [wait-animate "self" "sign-break"]  
    [go "fallen"]  
  )  
)  
(state ("fallen")  
  (on (begin)  
    [animate "self" "sign-broken"] ;; looping  
  )  
)  
)
```


Making the Sign Script Generic

```
(define-state-script ("simple-animating-obj")
  (state ("initial")
    (on (update)
      [when [task-complete?
              [tag-string "done-task"]]
        [go "done"]
      ]
    )
    (on (event "hanging-from")
      [go "animating"]
    )
  )
  ...)
```

```
(state ("animating")
  (on (begin)
    [spawn-particles-at-joint "self"
      [tag-string "particle-joint"]
      [tag-string "particle-name"]]
    [wait-animate "self"
      [tag-string "anim-name"]]
    [go "done"]
  )
)
(state ("done")
  (on (begin)
    [animate "self"
      [tag-string "done-anim-name"]]
  )
)
)
```


Making the Sign Script Generic

```
(define-state-script ("simple-animating-obj")
  (state ("initial")
    (on (update)
      [when [task-complete?
              [tag-string "done-task"]]
        [go "done"]
      ]
    )
    (on (event "hanging-from")
      [go "animating"]
    )
  )
  ...)
```

```
(state ("animating")
  (on (begin)
    [spawn-particles-at-joint "self"
      [tag-string "particle-joint"]
      [tag-string "particle-name"]]
    [wait-animate "self"
      [tag-string "anim-name"]]
    [go "done"]
  )
)
(state ("done")
  (on (begin)
    [animate "self"
      [tag-string "done-anim-name"]]
  )
)
)
```


Making the Sign Script Generic

```
(define-state-script ("simple-animating-obj")
  (state ("initial")
    (on (update)
      [when [task-complete?
              [tag-string "done-task"]]
        [go "done"]
      ]
    )
    (on (event "hanging-from")
      [go "animating"]
    )
  )
  ...)
```

```
(state ("animating")
  (on (begin)
    [spawn-particles-at-joint "self"
      [tag-string "particle-joint"]
      [tag-string "particle-name"]]
    [wait-animate "self"
      [tag-string "anim-name"]]
    [go "done"]
  )
)
(state ("done")
  (on (begin)
    [animate "self"
      [tag-string "done-anim-name"]]
  )
)
)
```


Making the Sign Script Generic

```
(define-state-script ("simple-animating-obj")
  (state ("initial")
    (on (update)
      [when [task-complete?
              [tag-string "done-task"]]
        [go "done"]
      ]
    )
    (on (event "hanging-from")
      [go "animating"]
    )
  )
  ...)
```

```
(state ("animating")
  (on (begin)
    [spawn-particles-at-joint "self"
      [tag-string "particle-joint"]
      [tag-string "particle-name"]]
    [wait-animate "self"
      [tag-string "anim-name"]]
    [go "done"]
  )
)
(state ("done")
  (on (begin)
    [animate "self"
      [tag-string "done-anim-name"]]
  )
)
)
```


Making the Sign Script Generic

```
(define-state-script ("simple-animating-obj")
  (state ("initial")
    (on (update)
      [when [task-complete?
              [tag-string "done-task"]]
        [go "done"]
      ]
    )
    (on (event "hanging-from")
      [go "animating"]
    )
  )
  ...)
```

```
(state ("animating")
  (on (begin)
    [spawn-particles-at-joint "self"
      [tag-string "particle-joint"]
      [tag-string "particle-name"]]
    [wait-animate "self"
      [tag-string "anim-name"]]
    [go "done"]
  )
)
(state ("done")
  (on (begin)
    [animate "self"
      [tag-string "done-anim-name"]]
  )
)
)
```


Making the Sign Script Generic

```
(define-state-script ("simple-animating-obj")
  (state ("initial")
    (on (update)
      [when [task-complete?
              [tag-string "done-task"]]
        [go "done"]
      ]
    )
    (on (event "hanging-from")
      [go "animating"]
    )
  )
  ...)
```

```
(state ("animating")
  (on (begin)
    [spawn-particles-at-joint "self"
      [tag-string "particle-joint"]
      [tag-string "particle-name"]]
    [wait-animate "self"
      [tag-string "anim-name"]]
    [go "done"]
  )
)
(state ("done")
  (on (begin)
    [animate "self"
      [tag-string "done-anim-name"]]
  )
)
)
```


Making the Sign Script Generic

```
(define-state-script ("simple-animating-obj")
  (state ("initial")
    (on (update)
      [when [task-complete?
              [tag-string "done-task"]]
        [go "done"]
      ]
    )
    (on (event "hanging-from")
      [go "animating"]
    )
  )
  ...)
```

```
(state ("animating")
  (on (begin)
    [spawn-particles-at-joint "self"
      [tag-string "particle-joint"]
      [tag-string "particle-name"]]
    [wait-animate "self"
      [tag-string "anim-name"]]
    [go "done"]
  )
)
(state ("done")
  (on (begin)
    [animate "self"
      [tag-string "done-anim-name"]]
  )
)
)
```


Making the Sign Script Generic

```
(define-state-script ("simple-animating-obj")
  (state ("initial")
    (on (update)
      [when [task-complete?
              [tag-string "done-task"]]
        [go "done"]
      ]
    )
    (on (event "hanging-from")
      [go "animating"]
    )
  )
  ...)
```

```
(state ("animating")
  (on (begin)
    [spawn-particles-at-joint "self"
      [tag-string "particle-joint"]
      [tag-string "particle-name"]]
    [wait-animate "self"
      [tag-string "anim-name"]]
    [go "done"]
  )
)
(state ("done")
  (on (begin)
    [animate "self"
      [tag-string "done-anim-name"]]
  )
)
)
```


Making the Sign Script Generic

```
(define-state-script ("simple-animating-obj")
  (state ("initial")
    (on (update)
      [when [task-complete?
              [tag-string "done-task"]]
        [go "done"]
      ]
    )
    (on (event "hanging-from")
      [go "animating"]
    )
  )
  ...)
```

```
(state ("animating")
  (on (begin)
    [spawn-particles-at-joint "self"
      [tag-string "particle-joint"]
      [tag-string "particle-name"]]
    [wait-animate "self"
      [tag-string "anim-name"]]
    [go "done"]
  )
)
(state ("done")
  (on (begin)
    [animate "self"
      [tag-string "done-anim-name"]]
  )
)
)
```


Making the Sign Script Generic

```
(define-state-script ("simple-animating-obj")
  (state ("initial")
    (on (update)
      [when [task-complete?
              [tag-string "done-task"]]
        [go "done"]
      ]
    )
    (on (event "hanging-from")
      [go "animating"]
    )
  )
  ...)
```

```
(state ("animating")
  (on (begin)
    [spawn-particles-at-joint "self"
      [tag-string "particle-joint"]
      [tag-string "particle-name"]]
    [wait-animate "self"
      [tag-string "anim-name"]]
    [go "done"]
  )
)
(state ("done")
  (on (begin)
    [animate "self"
      [tag-string "done-anim-name"]]
  )
)
)
```


In-Game Debugging



In-Game Debugging



In-Game Debugging



In-Game Debugging



In-Game Debugging



In-Game Debugging



In-Game Cinematic: Bus Crash



In-Game Cinematic: Bus Crash

```
(define-state-script ("wz-bus-crash")
  (state ("spawn-soldiers")
    (on (begin)
      [player-disable-controls
       (controls all-but-right-stick)]
      [spawn-npc-in-combat "npc-wz-52"]
      [spawn-npc-in-combat "npc-wz-53"]
      ...
      [go "crash"]
    )
  )
  ...)
```

```
(state ("crash")
  (on (begin)
    (track ("bus")
      [wait-animate "bus-1" "bus-crash"
       [get-locator "ref-bus-crash-1"]]
      [signal "bus-done"]
    )
    (track ("player")
      [animate "player" "player-watch-crash"
       [get-locator "ref-bus-crash-1"]]
      [wait-until-frame 250]
      [say "player" "vox-wz-drk-01-what-the"]
      [signal "drake-done"]
    )
  )
  ...)
```


In-Game Cinematic: Bus Crash

```
(define-state-script ("wz-bus-crash")
  (state ("spawn-soldiers")
    (on (begin)
      [player-disable-controls
       (controls all-but-right-stick)]
      [spawn-npc-in-combat "npc-wz-52"]
      [spawn-npc-in-combat "npc-wz-53"]
      ...
      [go "crash"]
    )
  )
  ...)
```

```
(state ("crash")
  (on (begin)
    (track ("bus")
      [wait-animate "bus-1" "bus-crash"
       [get-locator "ref-bus-crash-1"]]
      [signal "bus-done"]
    )
    (track ("player")
      [animate "player" "player-watch-crash"
       [get-locator "ref-bus-crash-1"]]
      [wait-until-frame 250]
      [say "player" "vox-wz-drk-01-what-the"]
      [signal "drake-done"]
    )
  )
  ...)
```


In-Game Cinematic: Bus Crash

```
(define-state-script ("wz-bus-crash")  
  (state ("spawn-soldiers")  
    (on (begin)  
      [player-disable-controls  
        (controls all-but-right-stick)]  
      [spawn-npc-in-combat "npc-wz-52"]  
      [spawn-npc-in-combat "npc-wz-53"]  
      ...  
      [go "crash"]  
    )  
  )  
  ...
```

```
(state ("crash")  
  (on (begin)  
    (track ("bus")  
      [wait-animate "bus-1" "bus-crash"  
        [get-locator "ref-bus-crash-1"]]  
      [signal "bus-done"]  
    )  
    (track ("player")  
      [animate "player" "player-watch-crash"  
        [get-locator "ref-bus-crash-1"]]  
      [wait-until-frame 250]  
      [say "player" "vox-wz-drk-01-what-the"]  
      [signal "drake-done"]  
    )  
  )  
  ...
```


In-Game Cinematic: Bus Crash

```
(define-state-script ("wz-bus-crash")  
  (state ("spawn-soldiers")  
    (on (begin)  
      [player-disable-controls  
        (controls all-but-right-stick)]  
      [spawn-npc-in-combat "npc-wz-52"]  
      [spawn-npc-in-combat "npc-wz-53"]  
      ...  
      [go "crash"]  
    )  
  )  
  ...
```

```
(state ("crash")  
  (on (begin)  
    (track ("bus")  
      [wait-animate "bus-1" "bus-crash"  
        [get-locator "ref-bus-crash-1"]]  
      [signal "bus-done"]  
    )  
    (track ("player")  
      [animate "player" "player-watch-crash"  
        [get-locator "ref-bus-crash-1"]]  
      [wait-until-frame 250]  
      [say "player" "vox-wz-drk-01-what-the"]  
      [signal "drake-done"]  
    )  
  )  
  ...
```


In-Game Cinematic: Bus Crash

```
(define-state-script ("wz-bus-crash")  
  (state ("spawn-soldiers")  
    (on (begin)  
      [player-disable-controls  
        (controls all-but-right-stick)]  
      [spawn-npc-in-combat "npc-wz-52"]  
      [spawn-npc-in-combat "npc-wz-53"]  
      ...  
      [go "crash"]  
    )  
  )  
  ...
```

```
(state ("crash")  
  (on (begin)  
    (track ("bus")  
      [wait-animate "bus-1" "bus-crash"  
        [get-locator "ref-bus-crash-1"]]  
      [signal "bus-done"]  
    )  
    (track ("player")  
      [animate "player" "player-watch-crash"  
        [get-locator "ref-bus-crash-1"]]  
      [wait-until-frame 250]  
      [say "player" "vox-wz-drk-01-what-the"]  
      [signal "drake-done"]  
    )  
  )  
  ...
```


In-Game Cinematic: Bus Crash

```
(define-state-script ("wz-bus-crash")  
  (state ("spawn-soldiers")  
    (on (begin)  
      [player-disable-controls  
        (controls all-but-right-stick)]  
      [spawn-npc-in-combat "npc-wz-52"]  
      [spawn-npc-in-combat "npc-wz-53"]  
      ...  
      [go "crash"]  
    )  
  )  
  ...
```

```
(state ("crash")  
  (on (begin)  
    (track ("bus")  
      [wait-animate "bus-1" "bus-crash"  
        [get-locator "ref-bus-crash-1"]]  
      [signal "bus-done"]  
    )  
    (track ("player")  
      [animate "player" "player-watch-crash"  
        [get-locator "ref-bus-crash-1"]]  
      [wait-until-frame 250]  
      [say "player" "vox-wz-drk-01-what-the"]  
      [signal "drake-done"]  
    )  
  )  
  ...
```


In-Game Cinematic: Bus Crash

```
(define-state-script ("wz-bus-crash")  
  (state ("spawn-soldiers")  
    (on (begin)  
      [player-disable-controls  
        (controls all-but-right-stick)]  
      [spawn-npc-in-combat "npc-wz-52"]  
      [spawn-npc-in-combat "npc-wz-53"]  
      ...  
      [go "crash"]  
    )  
  )  
  ...
```

```
(state ("crash")  
  (on (begin)  
    (track ("bus")  
      [wait-animate "bus-1" "bus-crash"  
        [get-locator "ref-bus-crash-1"]]  
      [signal "bus-done"]  
    )  
    (track ("player")  
      [animate "player" "player-watch-crash"  
        [get-locator "ref-bus-crash-1"]]  
      [wait-until-frame 250]  
      [say "player" "vox-wz-drk-01-what-the"]  
      [signal "drake-done"]  
    )  
  )  
  ...
```


In-Game Cinematic: Bus Crash

```
(define-state-script ("wz-bus-crash")
  (state ("spawn-soldiers")
    (on (begin)
      [player-disable-controls
       (controls all-but-right-stick)]
      [spawn-npc-in-combat "npc-wz-52"]
      [spawn-npc-in-combat "npc-wz-53"]
      ...
      [go "crash"]
    )
  )
  ...)
```

```
(state ("crash")
  ...
  (track ("guy-hit-by-bus")
    [wait-animate "npc-wz-52" "npc-hit-by-bus"
     [get-locator "ref-bus-crash-1"]]
    [npc-die "npc-wz-52"]
    [signal "npc-dead"]
  )
  (track ("wait-for-all-done")
    [wait-for-signal "bus-done"]
    [wait-for-signal "drake-done"]
    [wait-for-signal "npc-dead"]
    [go "done"]
  )
)
)
)
(state ("done")
  ...)
```


In-Game Cinematic: Bus Crash

```
(define-state-script ("wz-bus-crash")
  (state ("spawn-soldiers")
    (on (begin)
      [player-disable-controls
       (controls all-but-right-stick)]
      [spawn-npc-in-combat "npc-wz-52"]
      [spawn-npc-in-combat "npc-wz-53"]
      ...
      [go "crash"]
    )
  )
  ...)
```

```
(state ("crash")
  ...
  (track ("guy-hit-by-bus")
    [wait-animate "npc-wz-52" "npc-hit-by-bus"
     [get-locator "ref-bus-crash-1"]]
    [npc-die "npc-wz-52"]
    [signal "npc-dead"]
  )
  (track ("wait-for-all-done")
    [wait-for-signal "bus-done"]
    [wait-for-signal "drake-done"]
    [wait-for-signal "npc-dead"]
    [go "done"]
  )
  ...
)
(state ("done")
  ...)
```


In-Game Cinematic: Bus Crash

```
(define-state-script ("wz-bus-crash")
  (state ("spawn-soldiers")
    (on (begin)
      [player-disable-controls
       (controls all-but-right-stick)]
      [spawn-npc-in-combat "npc-wz-52"]
      [spawn-npc-in-combat "npc-wz-53"]
      ...
      [go "crash"]
    )
  )
  ...)
```

```
(state ("crash")
  ...
  (track ("guy-hit-by-bus")
    [wait-animate "npc-wz-52" "npc-hit-by-bus"
     [get-locator "ref-bus-crash-1"]]
    [npc-die "npc-wz-52"]
    [signal "npc-dead"]
  )
  (track ("wait-for-all-done")
    [wait-for-signal "bus-done"]
    [wait-for-signal "drake-done"]
    [wait-for-signal "npc-dead"]
    [go "done"]
  )
  ...
)
(state ("done")
  ...)
```


In-Game Cinematic: Bus Crash


```
(define-state-script ("wz-bus-crash")
  (state ("spawn-soldiers")
    (on (begin)
      [player-disable-controls
       (controls all-but-right-stick)]
      [spawn-npc-in-combat "npc-wz-52"]
      [spawn-npc-in-combat "npc-wz-53"]
      ...
      [go "crash"]
    )
  )
  ...)
```

```
(state ("crash")
  ...
  (track ("guy-hit-by-bus")
    [wait-animate "npc-wz-52" "npc-hit-by-bus"
     [get-locator "ref-bus-crash-1"]]
    [npc-die "npc-wz-52"]
    [signal "npc-dead"]
  )
  (track ("wait-for-all-done")
    [wait-for-signal "bus-done"]
    [wait-for-signal "drake-done"]
    [wait-for-signal "npc-dead"]
    [go "done"]
  )
  ...
)
(state ("done")
  ...)
```


In-Game Cinematic: Bus Crash

```
(define-state-script ("wz-bus-crash")
  (state ("spawn-soldiers")
    (on (begin)
      [player-disable-controls
       (controls all-but-right-stick)]
      [spawn-npc-in-combat "npc-wz-52"]
      [spawn-npc-in-combat "npc-wz-53"]
      ...
      [go "crash"]
    )
  )
  ...)
```

```
(state ("crash")
  ...
  (track ("guy-hit-by-bus")
    [wait-animate "npc-wz-52" "npc-hit-by-bus"
     [get-locator "ref-bus-crash-1"]]
    [npc-die "npc-wz-52"]
    [signal "npc-dead"]
  )
  (track ("wait-for-all-done")
    [wait-for-signal "bus-done"]
    [wait-for-signal "drake-done"]
    [wait-for-signal "npc-dead"]
    [go "done"]
  )
  ...
)
)
)
(state ("done")
  ...)
```



Implementation



Virtual Machine


- Scheme-like runtime language implemented by a simple VM
 - Each track compiled into block of **byte code** called a **lambda**

```
(define-state-script ("kickable-gate")  
  (state ("locked")  
    (on (begin)  
      (track ("gate")  
        ...  
      )  
      (track ("lock")  
        ...  
      )  
    )  
  )  
)
```


Virtual Machine

- Scheme-like runtime language implemented by a simple VM
 - Each track compiled into block of **byte code** called a **lambda**

```
(define-state-script ("kickable-gate")  
  (state ("locked")  
    (on (begin)  
      (track ("gate")  
        ...  
      )  
      (track ("lock")  
        ...  
      )  
    )  
  )  
)
```



opcode1 operand1 operand2
opcode2
opcode3 operand3

Virtual Machine

- Scheme-like runtime language implemented by a simple VM
 - Each track compiled into block of **byte code** called a **lambda**

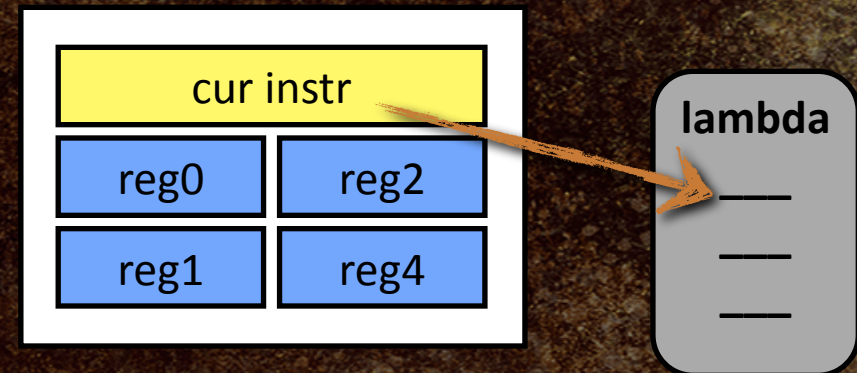
```
(define-state-script ("kickable-gate")  
  (state ("locked")  
    (on (begin)  
      (track ("gate")  
        ...  
      )  
      (track ("lock")  
        ...  
      )  
    )  
  )  
)
```

opcode1 operand1 operand2
opcode2
opcode3 operand3

opcode4 operand4
opcode5 operand5
opcode6

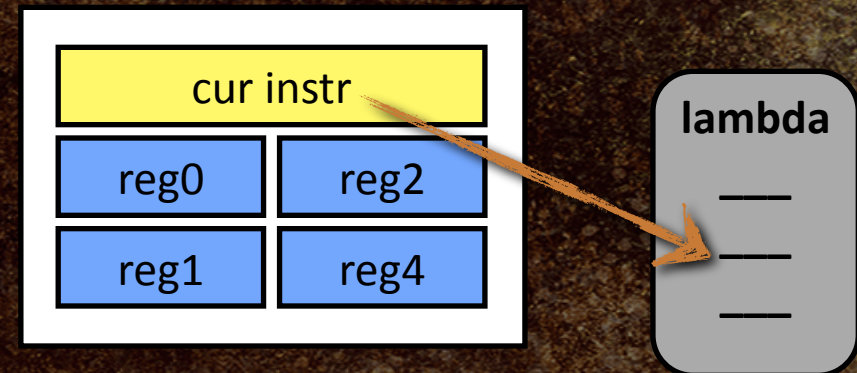
Virtual Machine

- Internal state of VM consists of:
 - Pointer to current **lambda** (byte code program)
 - Index to **current instruction**
 - Bank of **registers** for temporary and immediate data
- Registers are of type **variant**



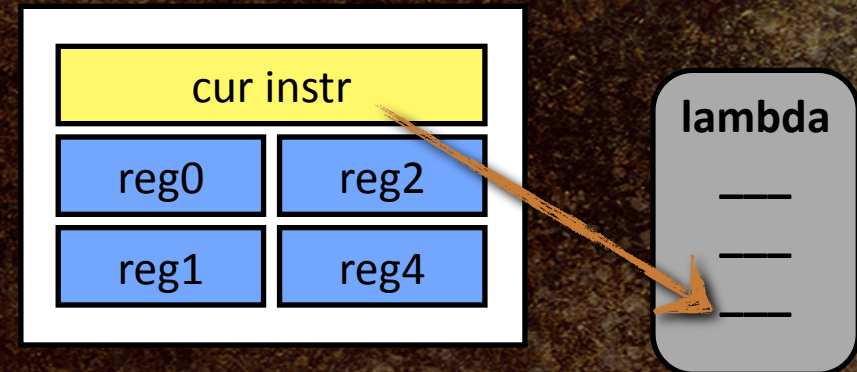
Virtual Machine

- Internal state of VM consists of:
 - Pointer to current **lambda** (byte code program)
 - Index to **current instruction**
 - Bank of **registers** for temporary and immediate data
- Registers are of type **variant**



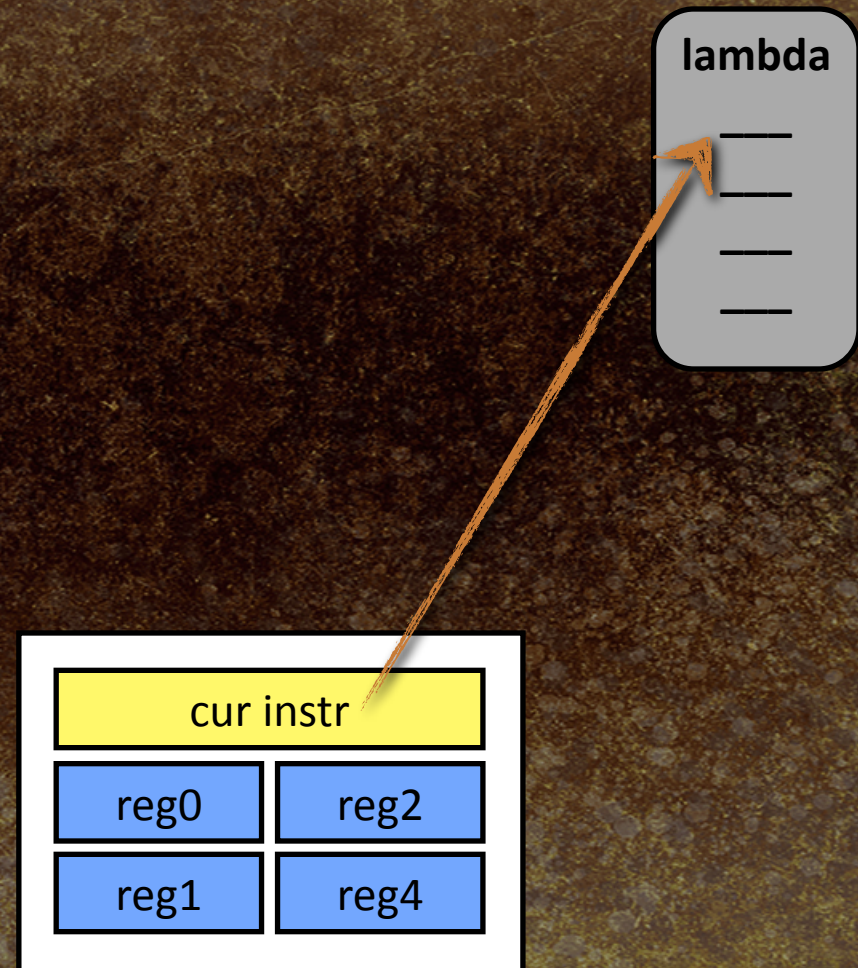
Virtual Machine

- Internal state of VM consists of:
 - Pointer to current **lambda** (byte code program)
 - Index to **current instruction**
 - Bank of **registers** for temporary and immediate data
- Registers are of type **variant**



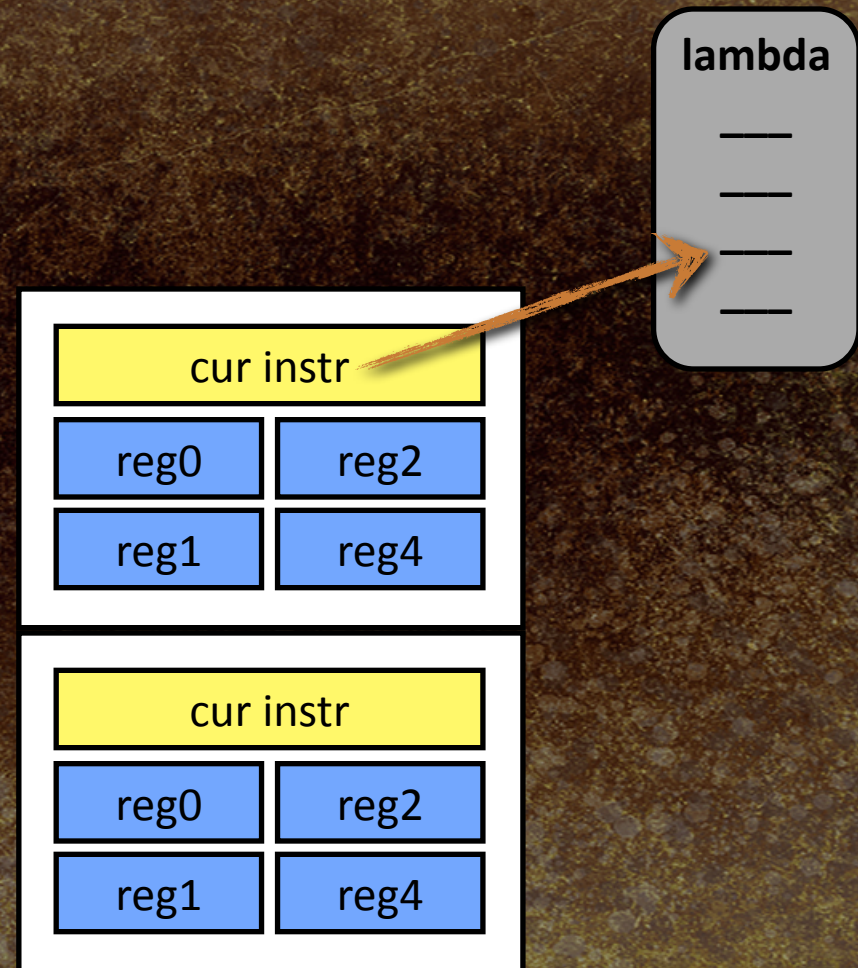
Virtual Machine

- Language supports nested function calls
- Hence requires **call stack**
 - Stack frame = bank of registers + program counter



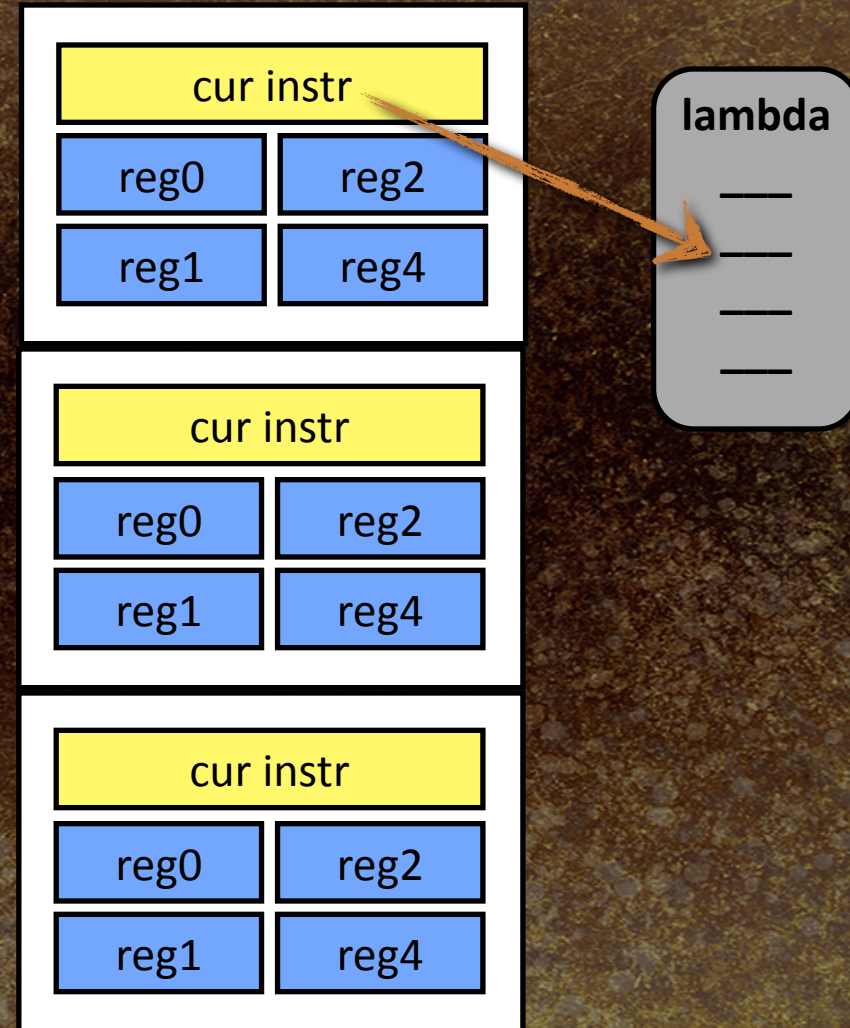
Virtual Machine

- Language supports nested function calls
- Hence requires **call stack**
 - Stack frame = bank of registers + program counter



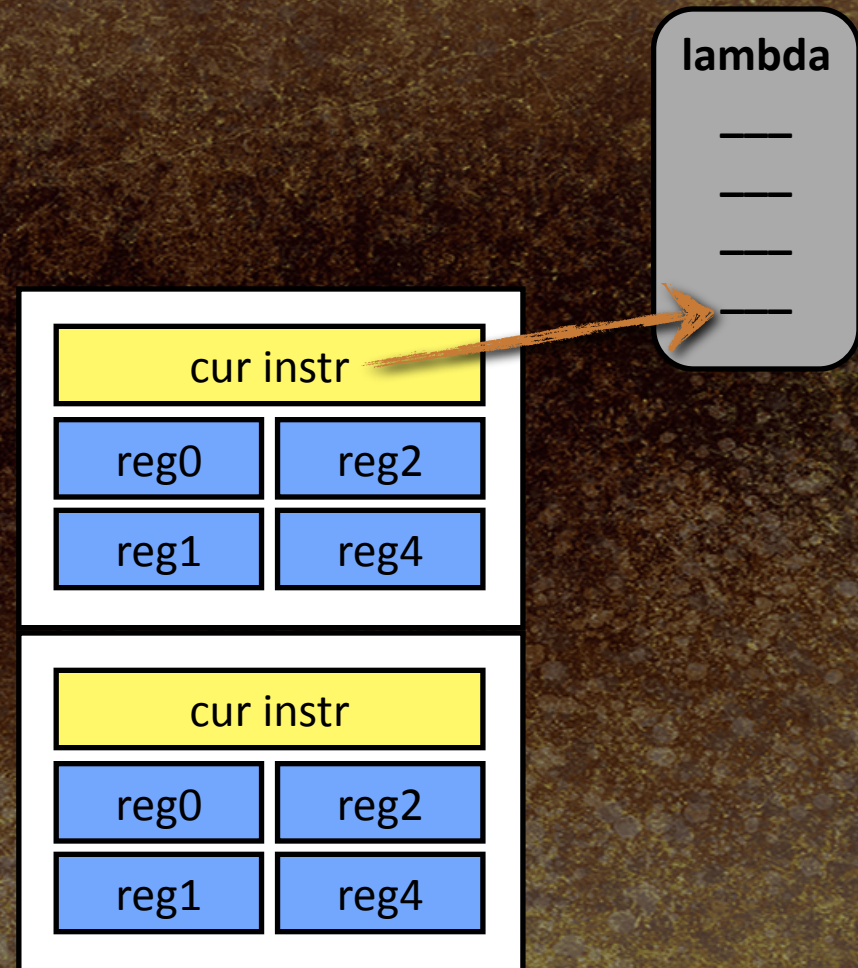
Virtual Machine

- Language supports nested function calls
- Hence requires **call stack**
 - Stack frame = bank of registers + program counter



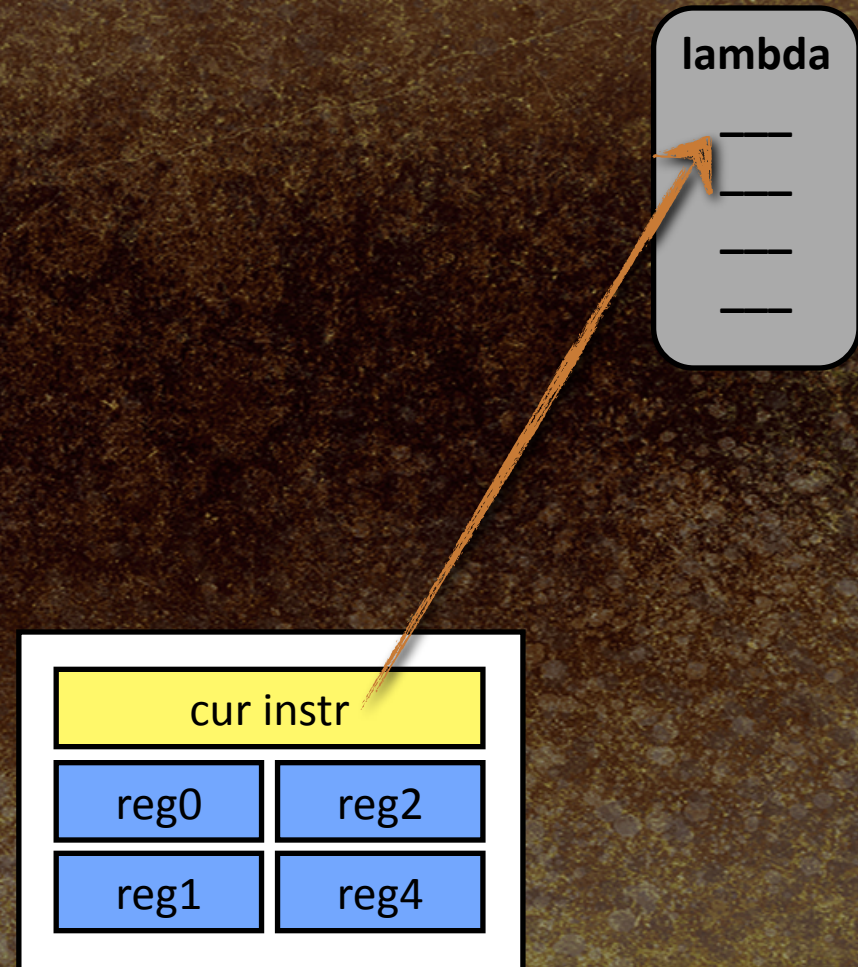
Virtual Machine

- Language supports nested function calls
- Hence requires **call stack**
 - Stack frame = bank of registers + program counter



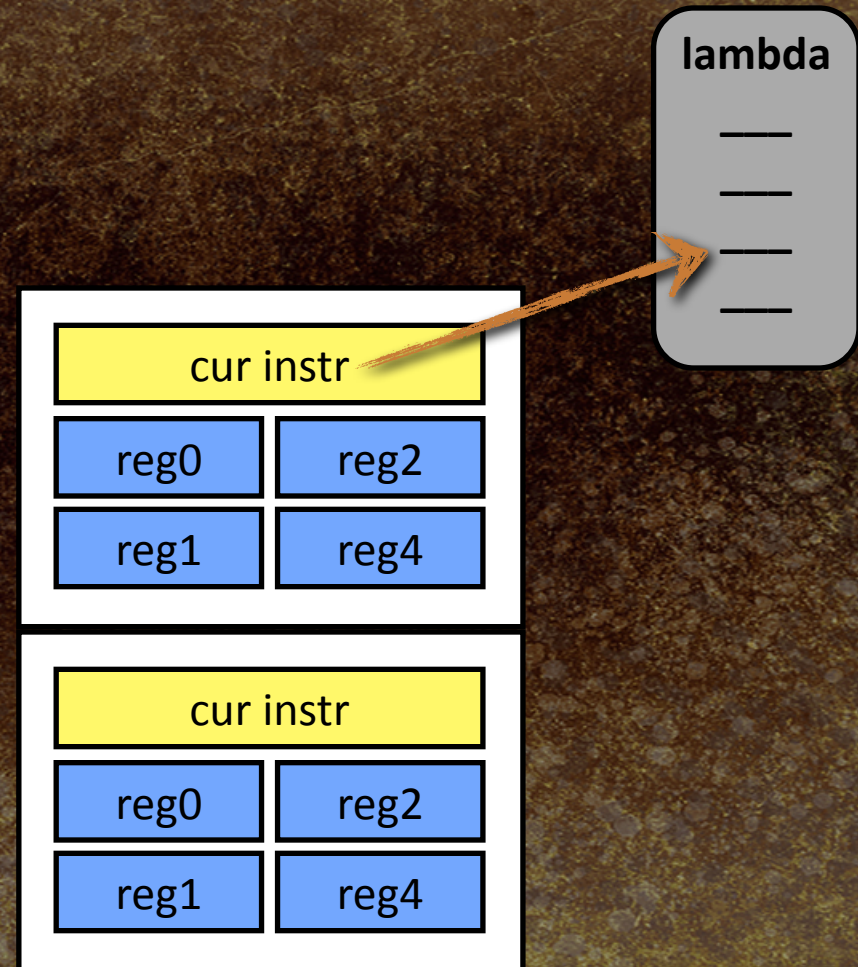
Continuations

- State script code can **wait** (sleep)
- Implemented via something known as a **continuation**



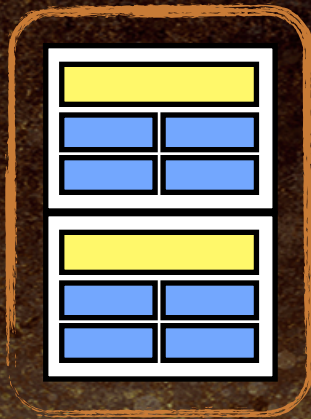
Continuations

- State script code can **wait** (sleep)
- Implemented via something known as a **continuation**

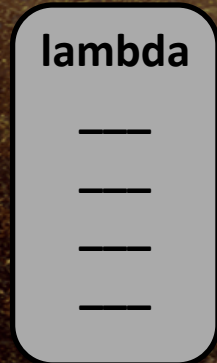


Continuations

- State script code can **wait** (sleep)
- Implemented via something known as a **continuation**

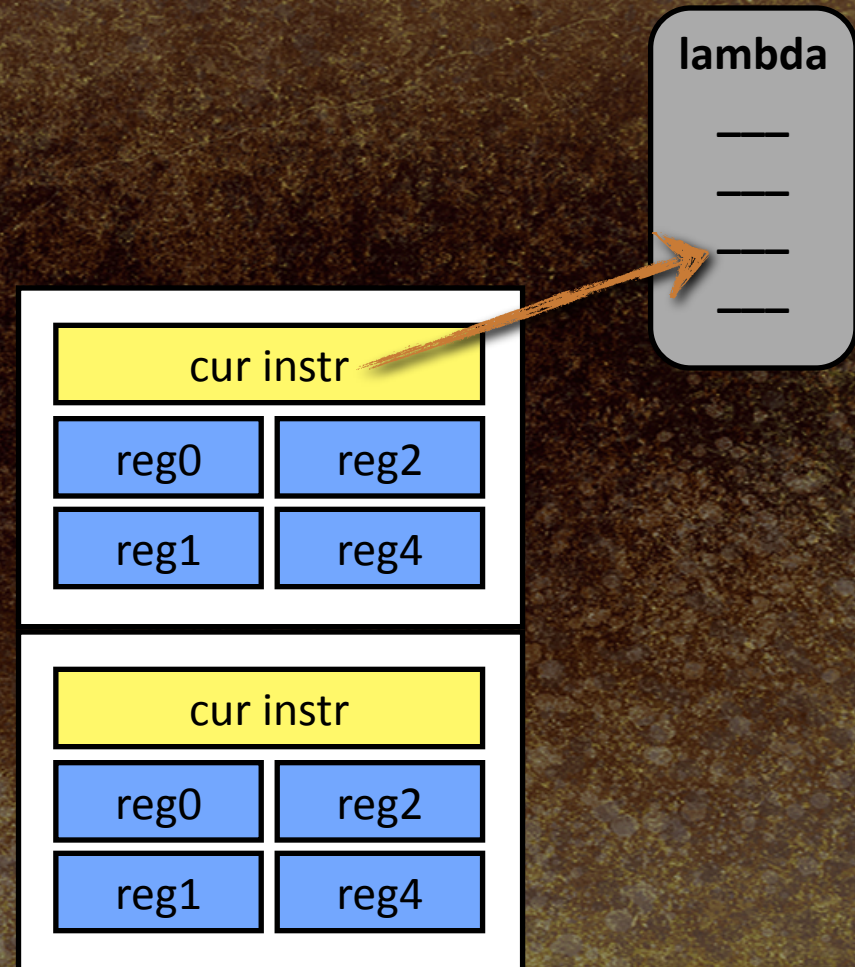


continuation



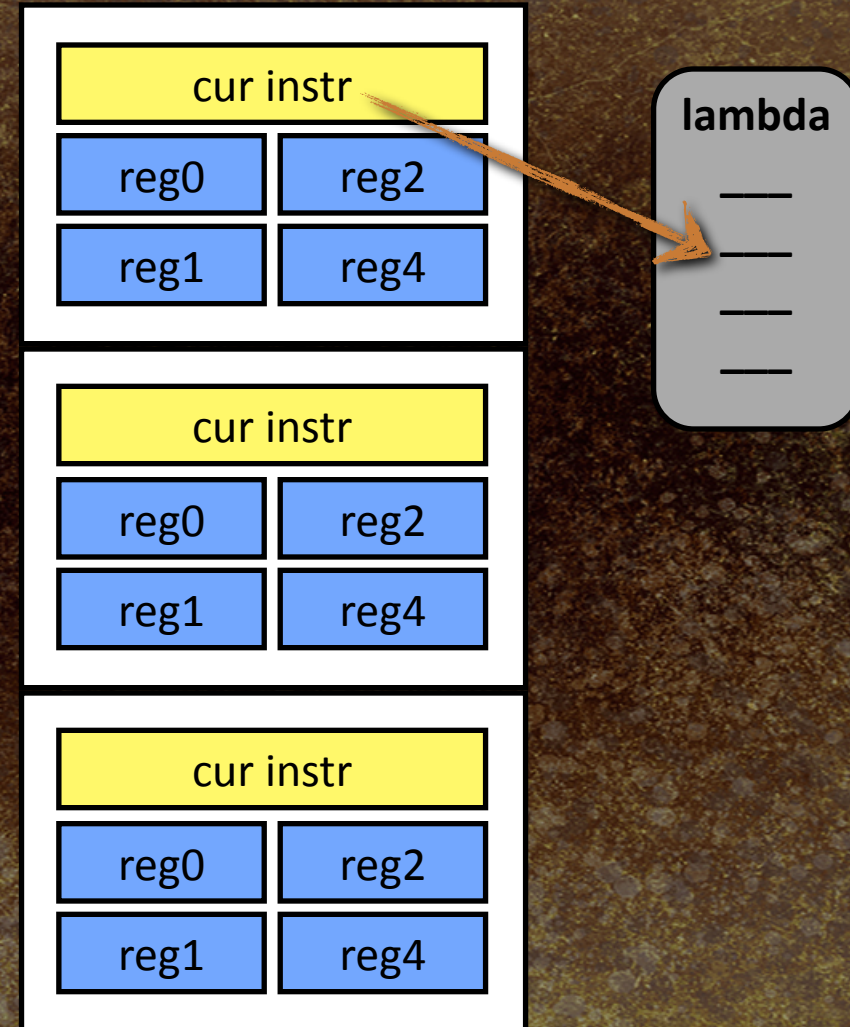
Continuations

- State script code can **wait** (sleep)
- Implemented via something known as a **continuation**



Continuations


- State script code can **wait** (sleep)
- Implemented via something known as a **continuation**



Native C++ Functions

- The “meat” of an (on ...) block is its **command sequence**
 - Some commands are user-defined: (**defun** func1 (...))
 - Most commands are C++ hooks into the engine
- Simple **hash table** maps function names to C++ code

```
(define-c-function wait-animate  
  (object-name string)  
  (anim-name string)  
)
```



```
Variant ScriptWaitAnimate(int argc, Variant* argv)  
{  
  ...  
}
```



```

Variant ScriptWaitAnimate(int argc, Variant* argv)
{
    StringId objName    = SC_ARG(0, StringId, NULL);
    StringId animName = SC_ARG(1, StringId, NULL);
    if (!objName)
        return ScriptError("wait-animate: expected object name (arg1)\n");
    if (!animName)
        return ScriptError("wait-animate: expected animation name (arg2)\n");
    // find the object
    ProcessGameObject* pObj = g_processMgr.Lookup(objName);
    if (!pObj)
        return ScriptError("wait-animate: could not find %s\n",
                               StringIdToString(objName));
    // instruct object to play animation, and wake up this script when done
    pObj->WaitAnimate(animName, g_scriptContext);
    g_scriptContext.Suspend(); // go to sleep until anim is done
    return Variant(true);
}

```




Conclusion



Key Features

- Key features of a successful scripting system:
 - **Virtual machine** integrated into game engine
 - Ability to run code every frame (**update**)
 - Ability to respond to **events**, and **send** events
 - Ability to **reference game objects** (by handle, unique id, etc.)
 - Ability to **manipulate** game objects
 - Designers can define **new object types** in script

Architectural Styles

- Many different engine-script architectures:
 - **script drives engine** (engine just a library called by script)
 - **engine drives script...**
 - simple scripted **event handlers**
 - scripted **properties** or **components**
 - scripted **game object classes**

Some Advice...

- Good debugging tools are crucial
 - Understandable error messages from compiler
 - In-game display of running scripts, or some kind of debugger
 - Simple logging to TTY/console (e.g. `[print-string "message"]`)
- Watch for **race conditions**
 - Event arrives too early or too late—missed
 - Object you're trying to control hasn't spawned yet, or has died
 - Tracks not synchronized properly

With Great Power...

- ... comes great **responsibility**
- Your designers can now:
 - **break the build**
 - introduce progression stoppers
 - **crash the game**
- Take the leap of faith, but...
 - **PLAN FOR IT!**

With Great Power...

- ... comes great **responsibility**
- Your designers can now:
 - **break the build**
 - introduce progression stoppers
 - **crash the game**
- Take the leap of faith, but...
 - **PLAN FOR IT!**



Thanks For Listening!

- Free free to send questions to me at:
jason_gregory@naughtydog.com



- Naughty Dog is hiring! Send resumes to Candace Walker at:
candace_walker@naughtydog.com