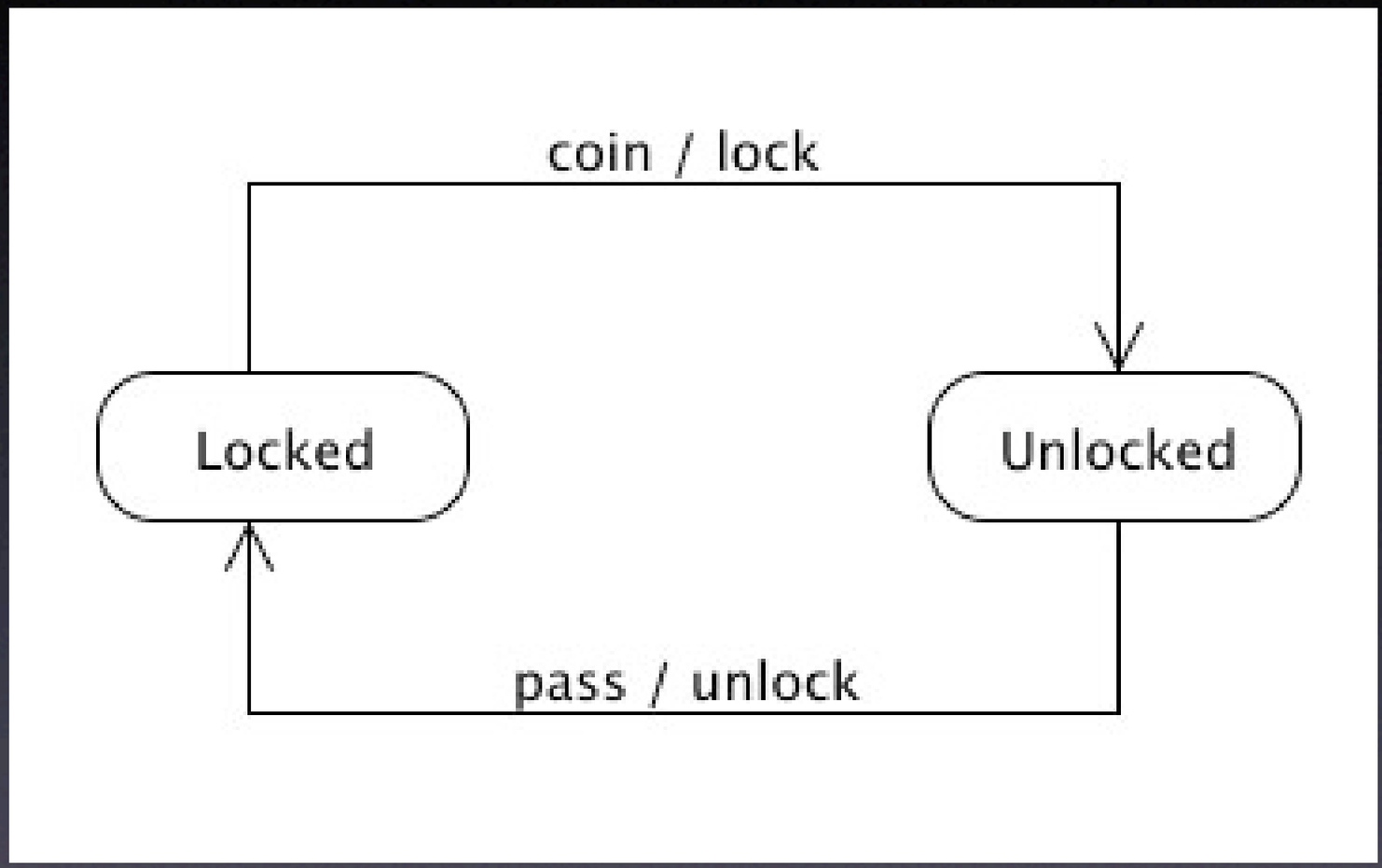
# Finite State Machines and the Statemachine Ruby Gem

Micah Martin
8th Light, Inc.

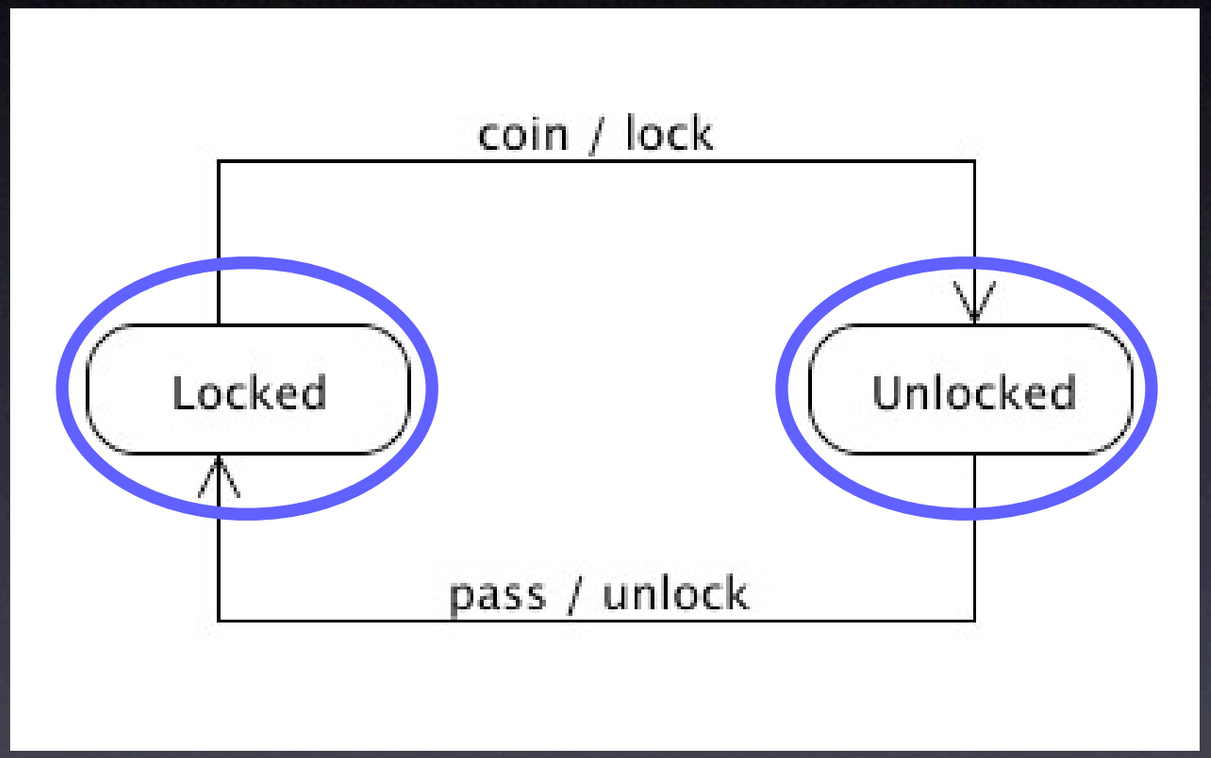
# The Subway Turnstile




4 - fundematal components

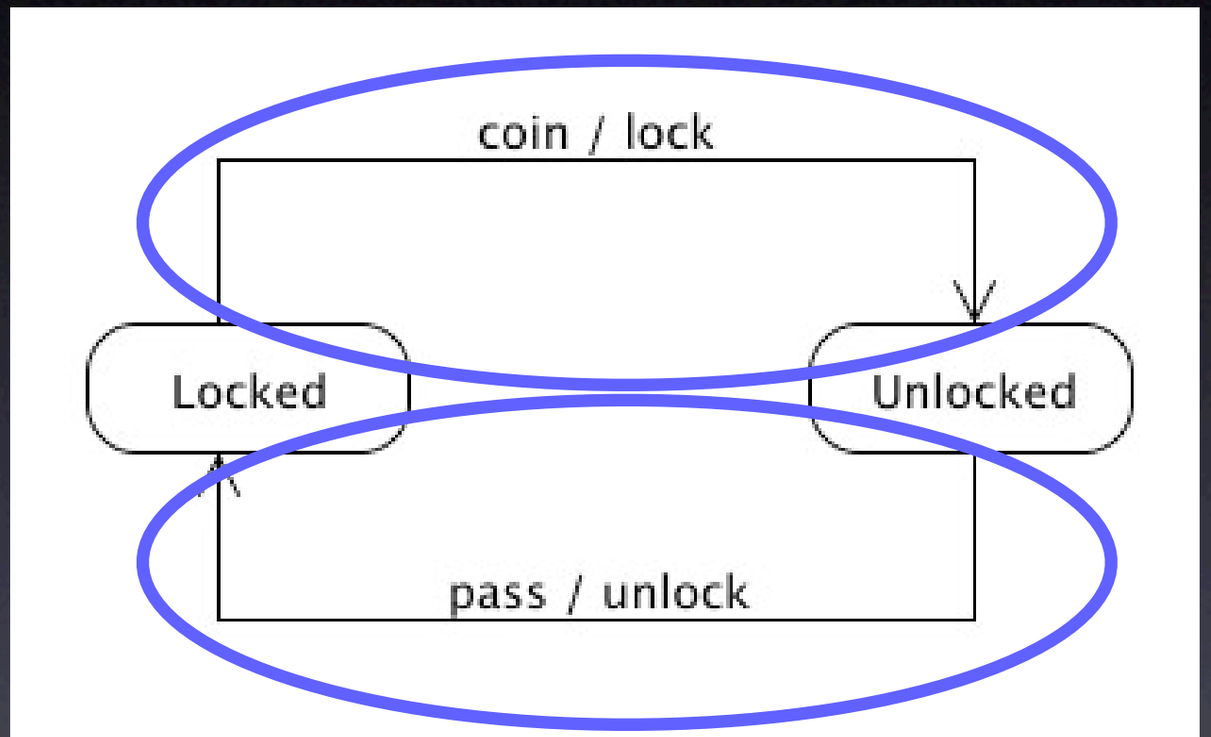
# States

1. States
2. Transitions
3. Events
4. Actions



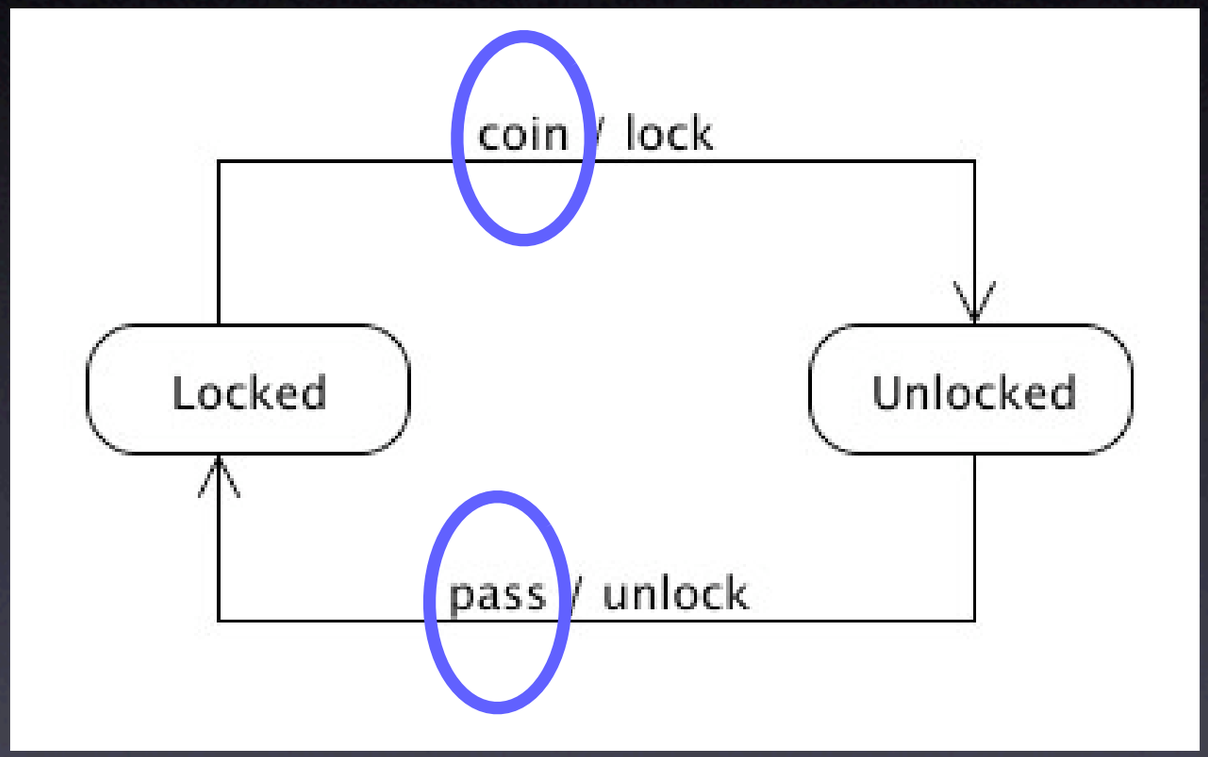

# Transitions

1. States
2. Transitions
3. Events
4. Actions



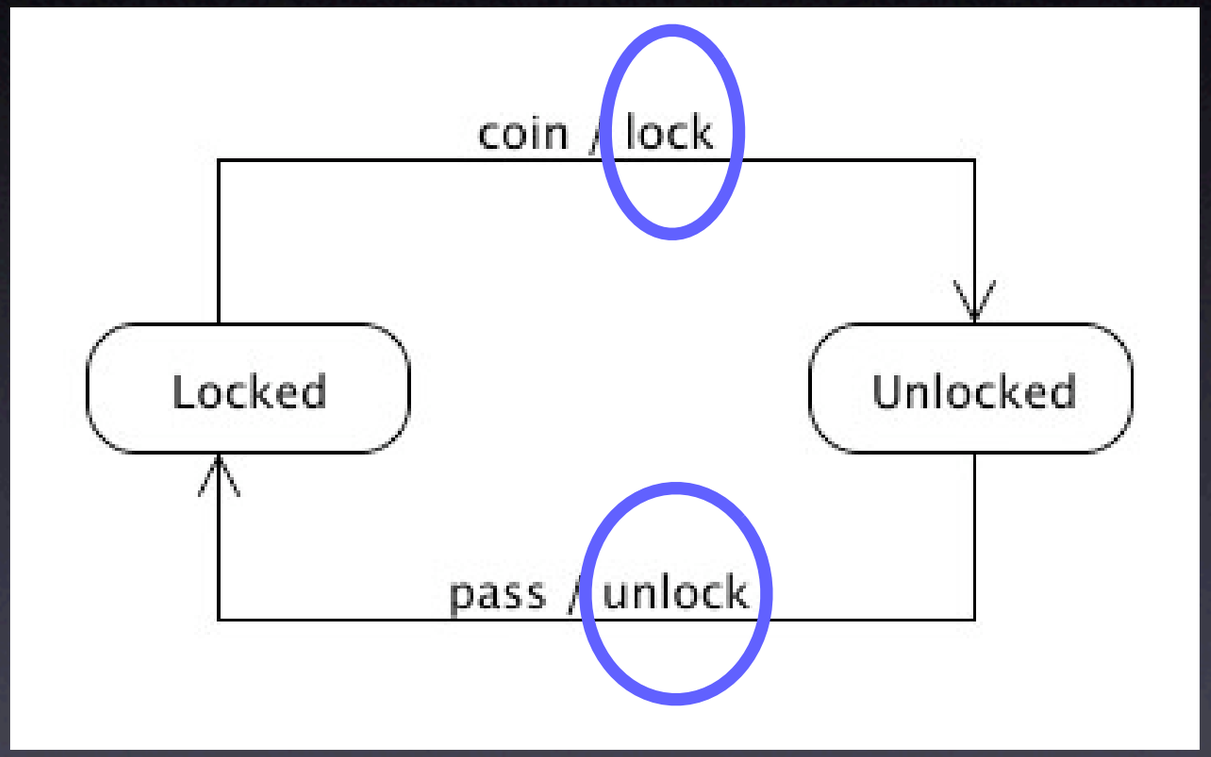
# Events

1. States
2. Transitions
3. Events
4. Actions




# Actions

1. States
2. Transitions
3. Events
4. Actions



# Why use Statemachines?

- Finite State Automata (Statemachine) > Finite Automata
- Constructs of Computational Theory
- Can solve any solvable problem.

# Why use Statemachines?

- Some algorithms are easily modeled as Statemachines
  - GUIs
  - Drawing
  - Parsing
  - Reactionary Systems

# Implementing Statemachines

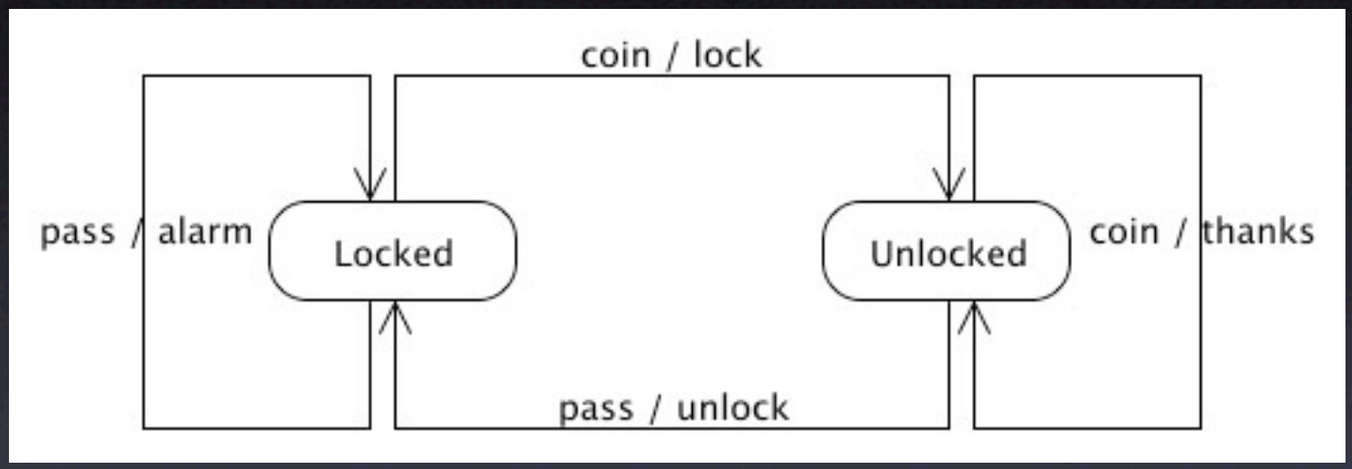
- Switching
- Transition mapping
- State Patterns

# SMC

- State Machine Compiler
- Written by Robert C Martin
- Generated code using the State Pattern
- C++, Java, C#

# SMC

## Descriptive Syntax

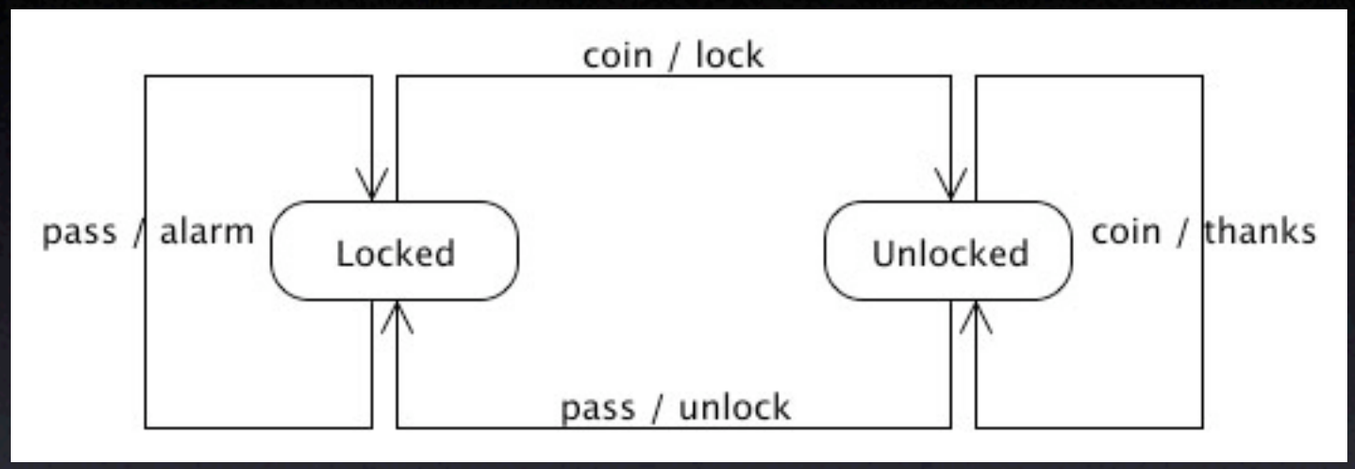



```
locked   coin   unlocked lock
locked   pass   locked   alarm
unlocked pass   locked   unlock
unlocked coin   unlocked thanks
```

# Statemachine Gem

- Maintains descriptive language
- Doesn't generate code
- The definition executes

# Statemachine Gem




```
sm = Statemachine.build do
  trans :locked,     :coin, :unlocked, :unlock
  trans :locked,     :pass, :locked,   :alarm
  trans :unlocked,   :pass, :locked,   :lock
  trans :unlocked,   :coin, :unlocked, :thanks
end
```

# Building Statemachines

```
sm = Statemachine.build do
  ...
end
```

- trans - declares states(start, end) and transition
- event - decalers transition and end state
- state - declares a state
- superstate - declares a superstate
- startstate - specifies start state
- context - specify the context
- on_entry, on_entry_of, on_exit, on_exit_of, default

# Using Statemachines

- One method call per event
- sm.process_event(event, *args)
- or go through method_missing

```
sm.coin
sm.pass
sm.pass
sm.coin
sm.coin
```

# Context

- Where all the actions takes place
- Provides separation of responsibility
- Allows for reusability


```
sm = Statemachine.build do
   ...
   context some_context_object
end
```

# Actions

- :symbol - a method on the context object
- block - called with the binding of the context object
- string - eval'ed with the binding of the context object

*Everything executes in the context object!!!*

# Action Parameters

- All actions can take parameters
- parameters are used if the action takes parameters

```
sm.coin(:dime)
sm.pass
```

```
class TurnstileContext
  def lock
    #lock
  end

  def unlock(coin)
    # use coin, unlock
  end
end
```

# Statemachines in Rails

- Are useful in AJAX rich screens
- Statemachine plugin : use at your own risk
- ruby script/plugin install svn://rubyforge.org/var/svn/statemachine/tags/0_2_2!/rails_plugin/vendor/plugins/statemachine
- Demo

# Exercise

- Ruby comment remover
- Download source code from:
  http://blog.8thlight.com/files/statemachine_exercise.zip
- When in trouble:
  `sm.tracer = $stdout`