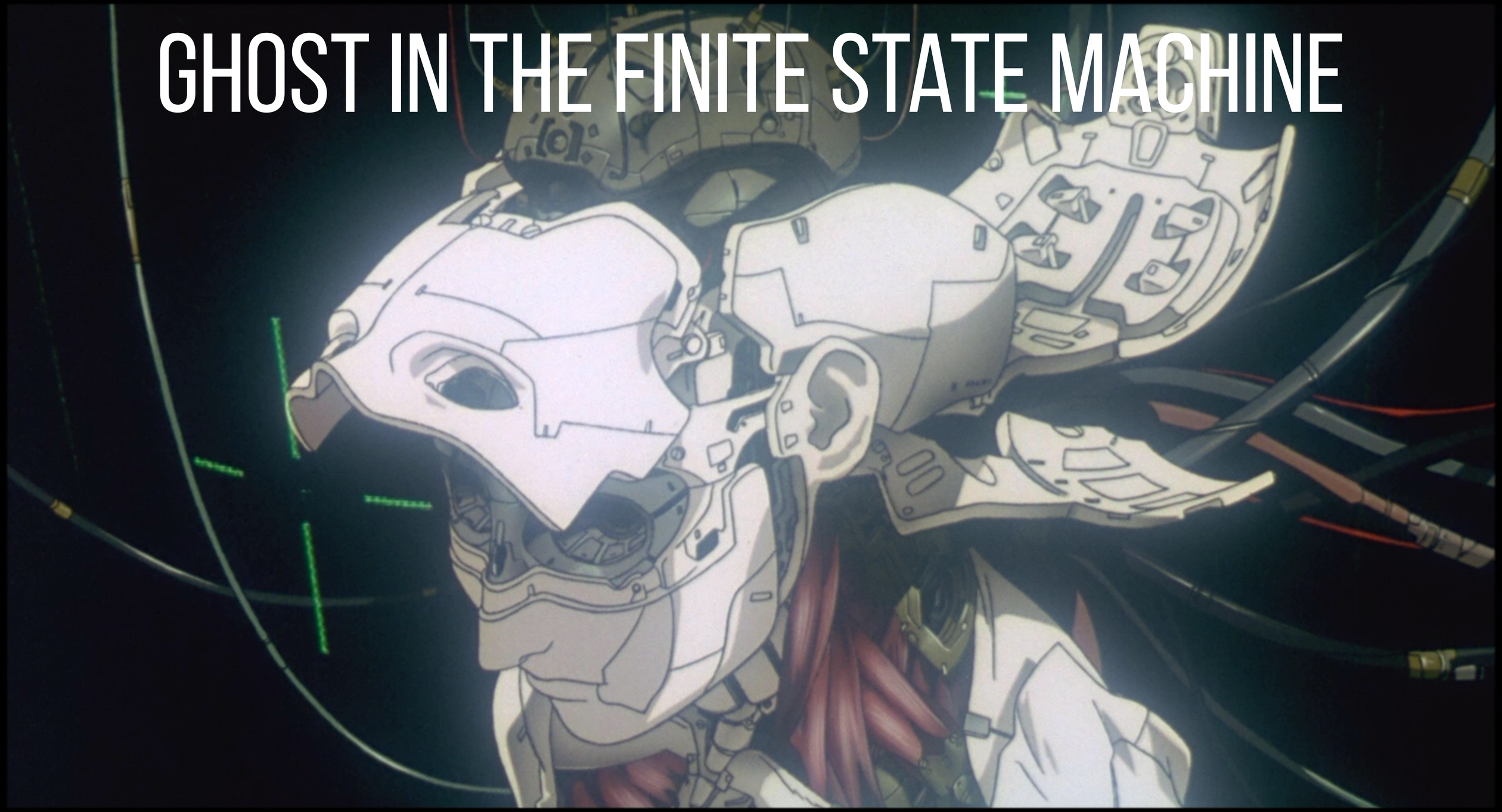
GHOST IN THE FINITE STATE MACHINE




PULL TAB TO OPEN

WHAT HAPPENS AFTER YOU CLICK “BUY NOW”?

WE NOW HAVE A NEW ORDER

WAITING FOR PAYMENT

WHAT IF PAYMENT DOESN’T ARRIVE?

PAYMENT HAS ARRIVED

Ship It!

“HEY, YOUR PACKAGE HAS SHIPPED”

CANCEL YOUR ORDER?

“WHAT WE'VE GOT HERE IS FAILURE TO ~~COMMUNICATE~~ IMPLEMENT A FSM”

SIGNS YOU HAVE A STATE MACHINE, BUT MAYBE DON’T KNOW IT YET

1) LOTS OF BOOLEANS

(TIMESTAMPS COUNT TOO)

2) YOU HAVE A FIELD CALLED `state` OR `status`

HOW TO IMPLEMENT A STATE MACHINE?

WE NEED STATES

WE NEED TRANSITIONS BETWEEN STATES

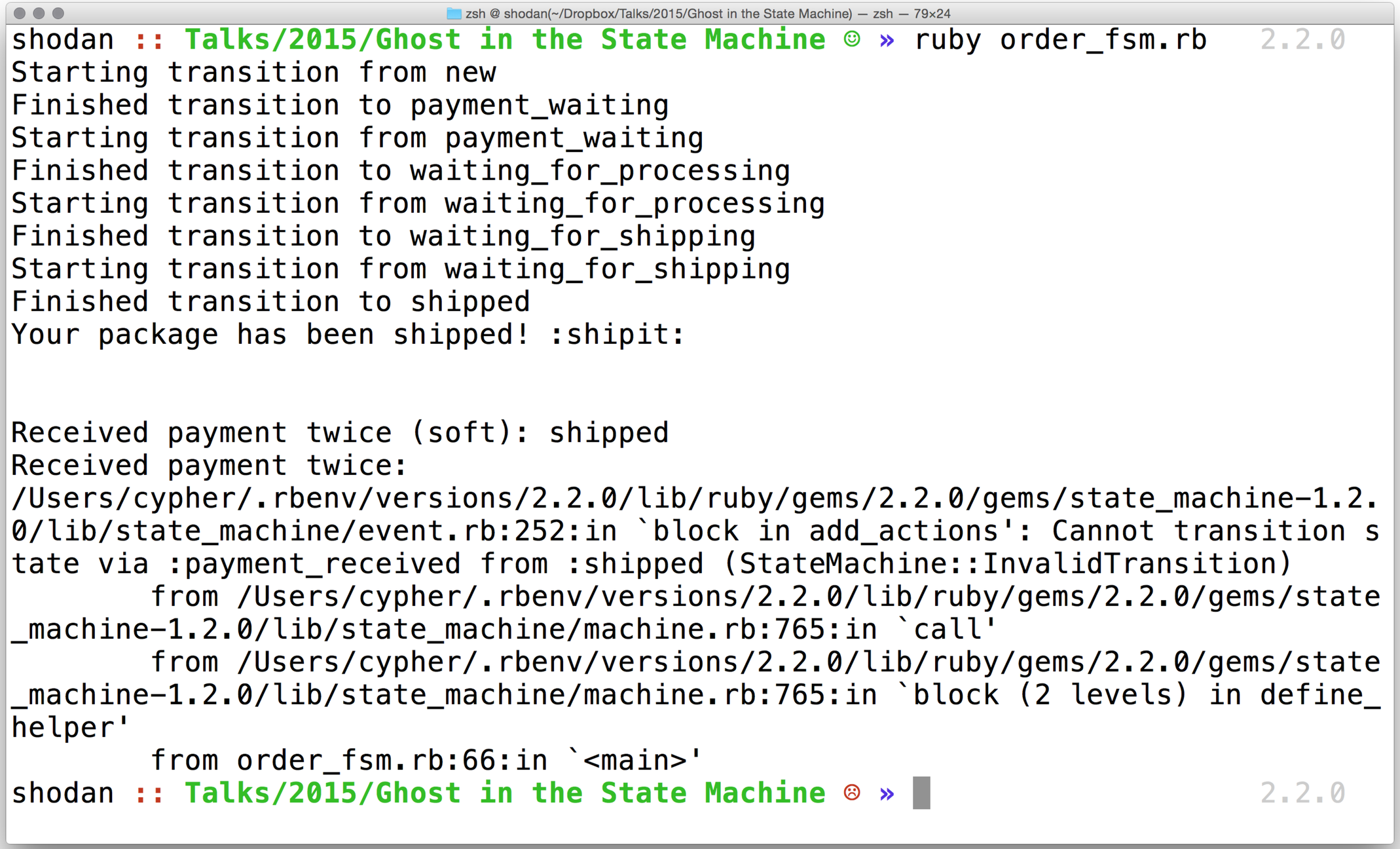
WE CAN IMPLEMENT IT OURSELVES

WE HAVE THE TECHNOLOGY

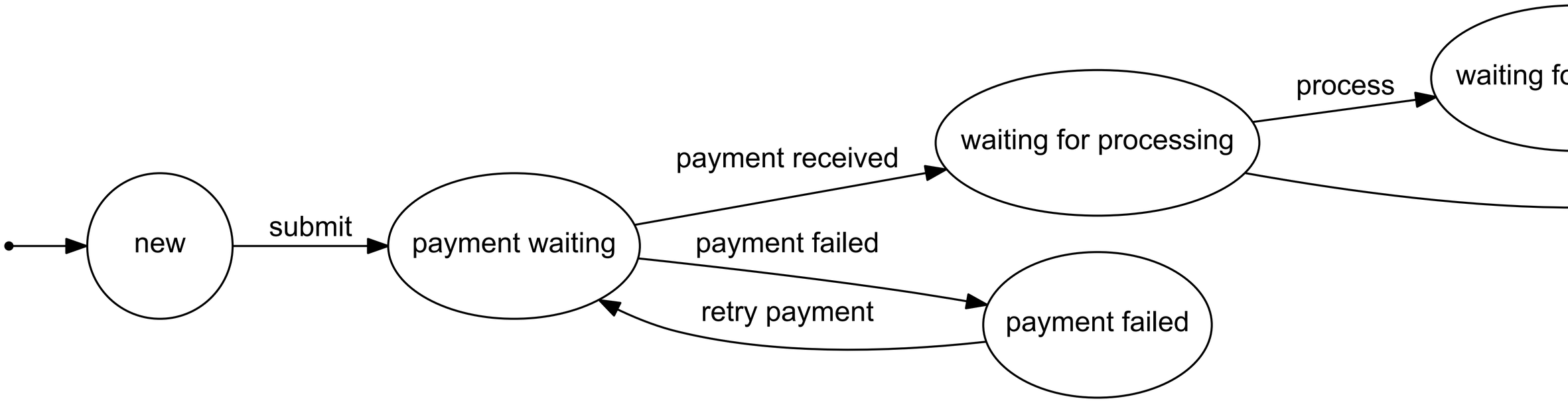
THE 6_000_000 STATES MACHINE MAN

state_machine

```ruby
require 'state_machine'

class Order
  state_machine :state, :initial => :new do
    around_transition do |order, transition, block|
      puts "Starting transition from #{transition.from}"
      block.call
      puts "Finished transition to #{transition.to}"
    end

    event :submit do
      transition :new => :payment_waiting
    end

    event :payment_received do
      transition :payment_waiting => :waiting_for_processing
    end

    event :payment_failed do
      transition :payment_waiting => :payment_failed
```


zsh @ shodan(~/Dropbox/Talks/2015/Ghost in the State Machine) — zsh — 79×24
shodan :: Talks/2015/Ghost in the State Machine ☺ » ruby order_fsm.rb 2.2.0
Starting transition from new
Finished transition to payment_waiting
Starting transition from payment_waiting
Finished transition to waiting_for_processing
Starting transition from waiting_for_processing
Finished transition to waiting_for_shipping
Starting transition from waiting_for_shipping
Finished transition to shipped
Your package has been shipped! :shipit:
Received payment twice (soft): shipped
Received payment twice:
/Users/cypher/.rbenv/versions/2.2.0/lib/ruby/gems/2.2.0/gems/state_machine-1.2.0/lib/state_machine/event.rb:252:in `block in add_actions': Cannot transition state via :payment_received from :shipped (StateMachine::InvalidTransition)
from /Users/cypher/.rbenv/versions/2.2.0/lib/ruby/gems/2.2.0/gems/state_machine-1.2.0/lib/state_machine/machine.rb:765:in `call'
from /Users/cypher/.rbenv/versions/2.2.0/lib/ruby/gems/2.2.0/gems/state_machine-1.2.0/lib/state_machine/machine.rb:765:in `block (2 levels) in define_helper'
from order_fsm.rb:66:in `<main>'
shodan :: Talks/2015/Ghost in the State Machine ☹ » 2.2.0

```
rake state_machine:draw \
    CLASS=Order \
    FILE=./order_fsm.rb \
    FORMAT=pdf \
    ORIENTATION=landscape \
    HUMAN_NAMES=true
```


new
submit
payment waiting
payment received
payment failed
retry payment
waiting for processing
payment failed
process
waiting fo

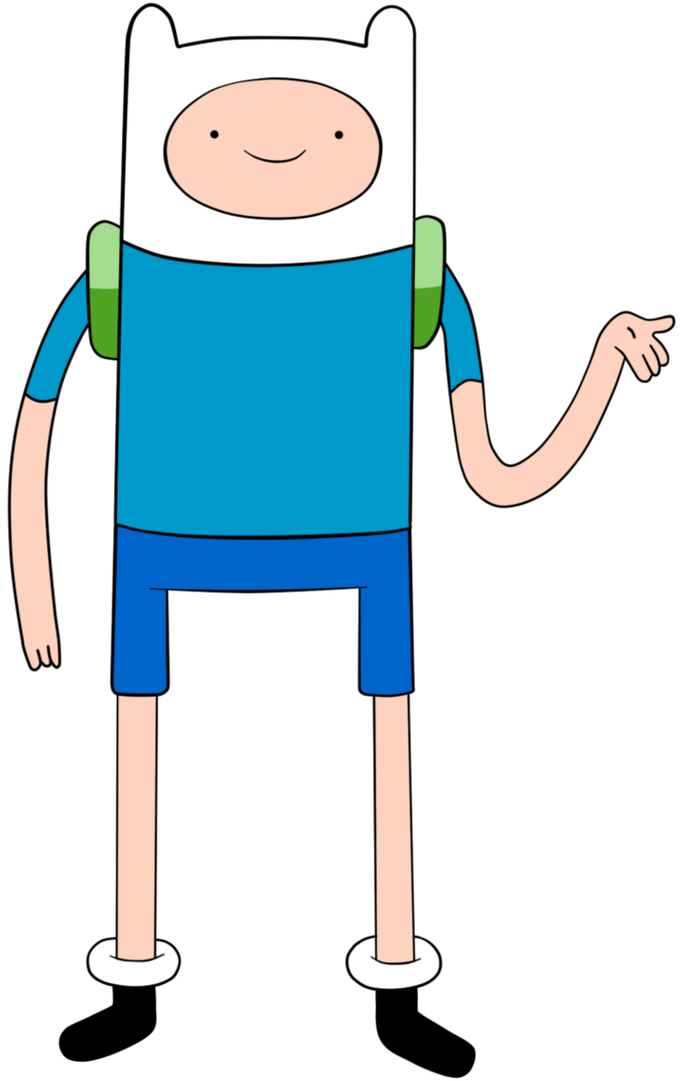
ACTS AS STATE MACHINE

(aasm)

EVERYTHING HAS STATE

USE AN FSM BEFORE IT IS TOO LATE

“WHY DEVELOPERS SHOULD BE FORCE-FED STATE MACHINES”



fin

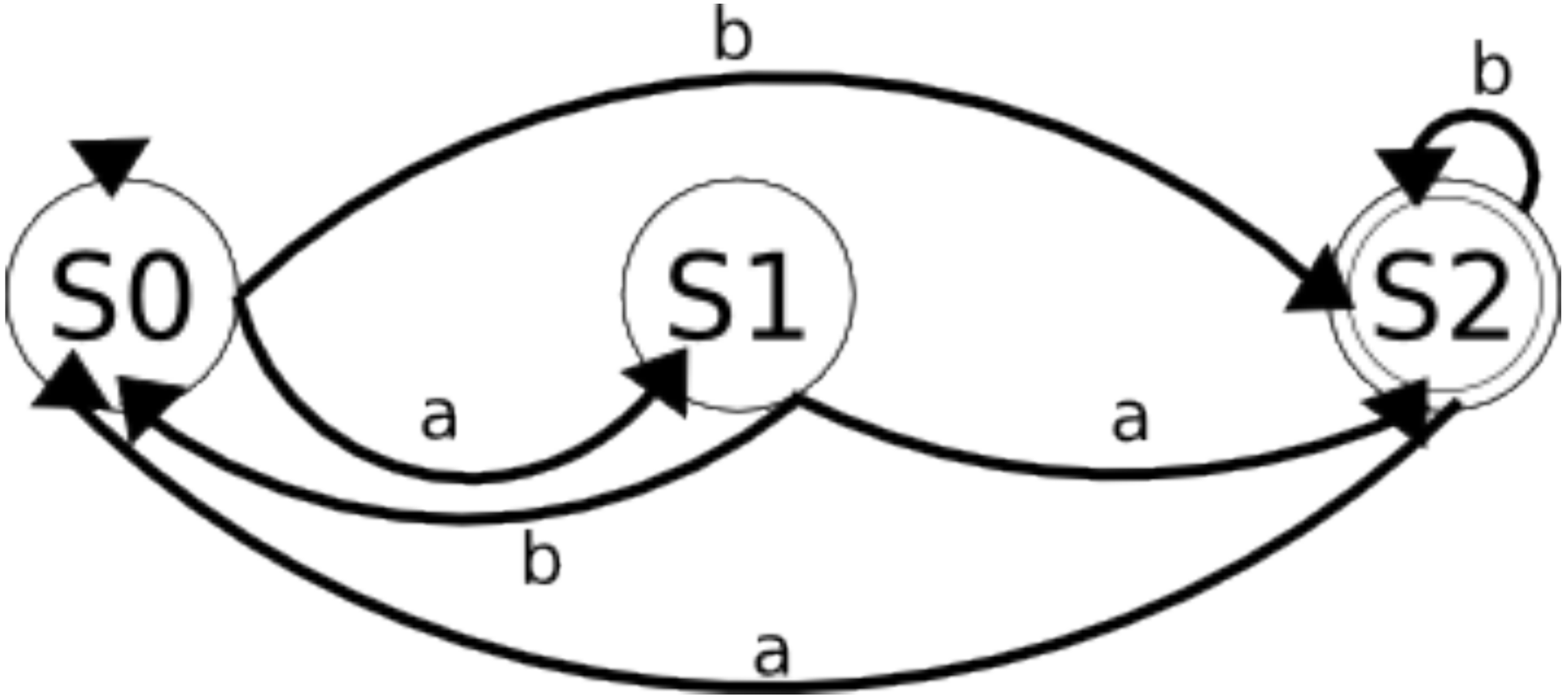
BONUS!

A STATE MACHINE IN SWIFT

via http://www.figure.ink/blog/2015/2/9/swift-state-machines-part-4-redirect

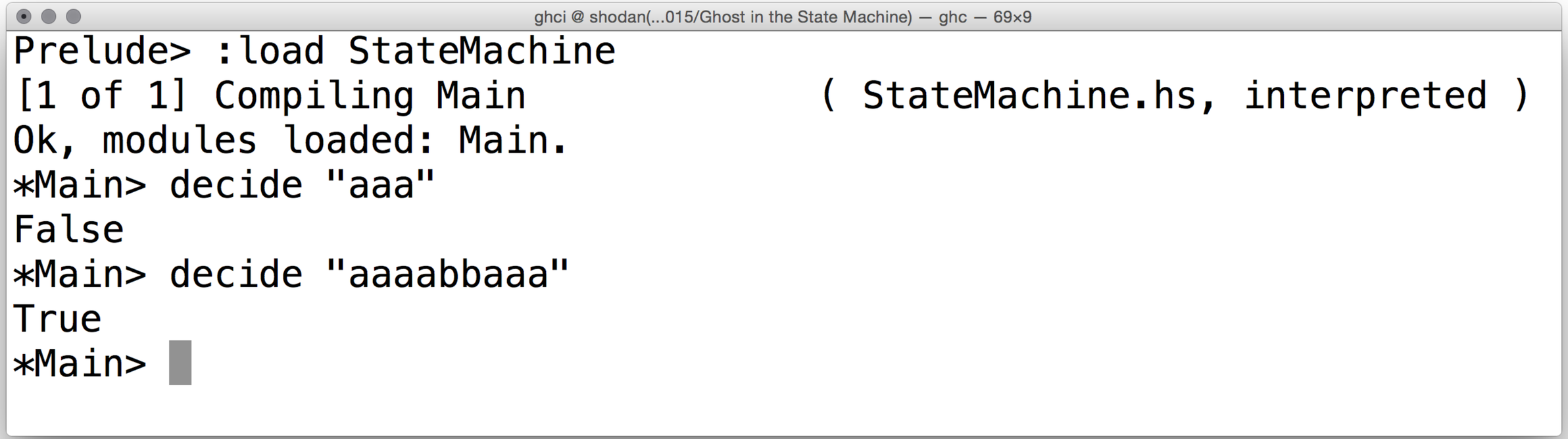
```
import Foundation

class StateMachine<P:StateMachineDelegateProtocol> {
    private unowned let delegate:P
    private let validTransitions: [P.StateType: [P.StateType]]

    private var _state:P.StateType{
        didSet {
            delegate.didTransitionFrom(oldValue, to:_state)
        }
    }

    var state:P.StateType {
        get{
            return _state
        }
        set{ // Can't be an observer because we need the option to
CONDITIONALLY set state
            attemptTransitionTo(newValue)
        }
```

```
import UIKit

class Example : UIView {

    private var machine:StateMachine<Example>!

    enum TrafficLight : Int {
        case Stop, Go, Caution
    }


    required init(coder aDecoder: NSCoder) {
        super.init(coder: aDecoder)

        let tx = [
            TrafficLight.Stop: [TrafficLight.Go],
            TrafficLight.Caution: [TrafficLight.Stop],
            TrafficLight.Go: [TrafficLight.Caution]
        ]
```



via https://daniel-levin.github.io/2015/01/19/primitive-state-machine-in-haskell.html

```haskell
data State = S0 | S1 | S2

accepts :: State -> String -> Bool
accepts S0 ('a':xs) = accepts S1 xs
accepts S0 ('b':xs) = accepts S2 xs
accepts S1 ('a':xs) = accepts S2 xs
accepts S1 ('b':xs) = accepts S0 xs
accepts S2 ('a':xs) = accepts S0 xs
accepts S2 ('b':xs) = accepts S2 xs
accepts S2 _        = True
accepts _ _         = False


decide :: String -> Bool
decide = accepts S0
```


ghci @ shodan(...015/Ghost in the State Machine) — ghc — 69×9
Prelude> :load StateMachine
[1 of 1] Compiling Main (StateMachine.hs, interpreted)
Ok, modules loaded: Main.
*Main> decide "aaa"
False
*Main> decide "aaaabbaaa"
True
*Main>