

HOW I SOLVED CONNECTIVITY HEADACHES WITH MACHINA.JS

JIM COWART / @IFANDELSE

WHO AM I?

- Developer Advocate @ Telerik
- @ifandelse
- OSS Author & Contributor
<http://github.com/ifandelse>

WHY ARE WE HERE?

- Managing online/offline in web & mobile apps can be done quite simply! But how?
- What are Finite State Machines & how do they fit in?
- What FSM behaviors are provided by machina.js?
- How can we use this in the real world?

WHAT'S WRONG WITH THE CURRENT BROWSER APIs?

(BESIDES THE FACT
THAT THEY SUCK)

CONNECTIVITY DETECTION OPTIONS

- jQuery ajaxError event

```
$(document).ajaxError(  
    function(event, req, opt) {  
        //Wow- LOTS of assumptions  
        //being made here...  
        app.setStatus("offline");  
    }  
);
```

*Yes - I know this isn't *browser-level*, hey look...a squirrel!*

Yep - it's useful.

It's also blunt, and based
entirely on 'failures'.

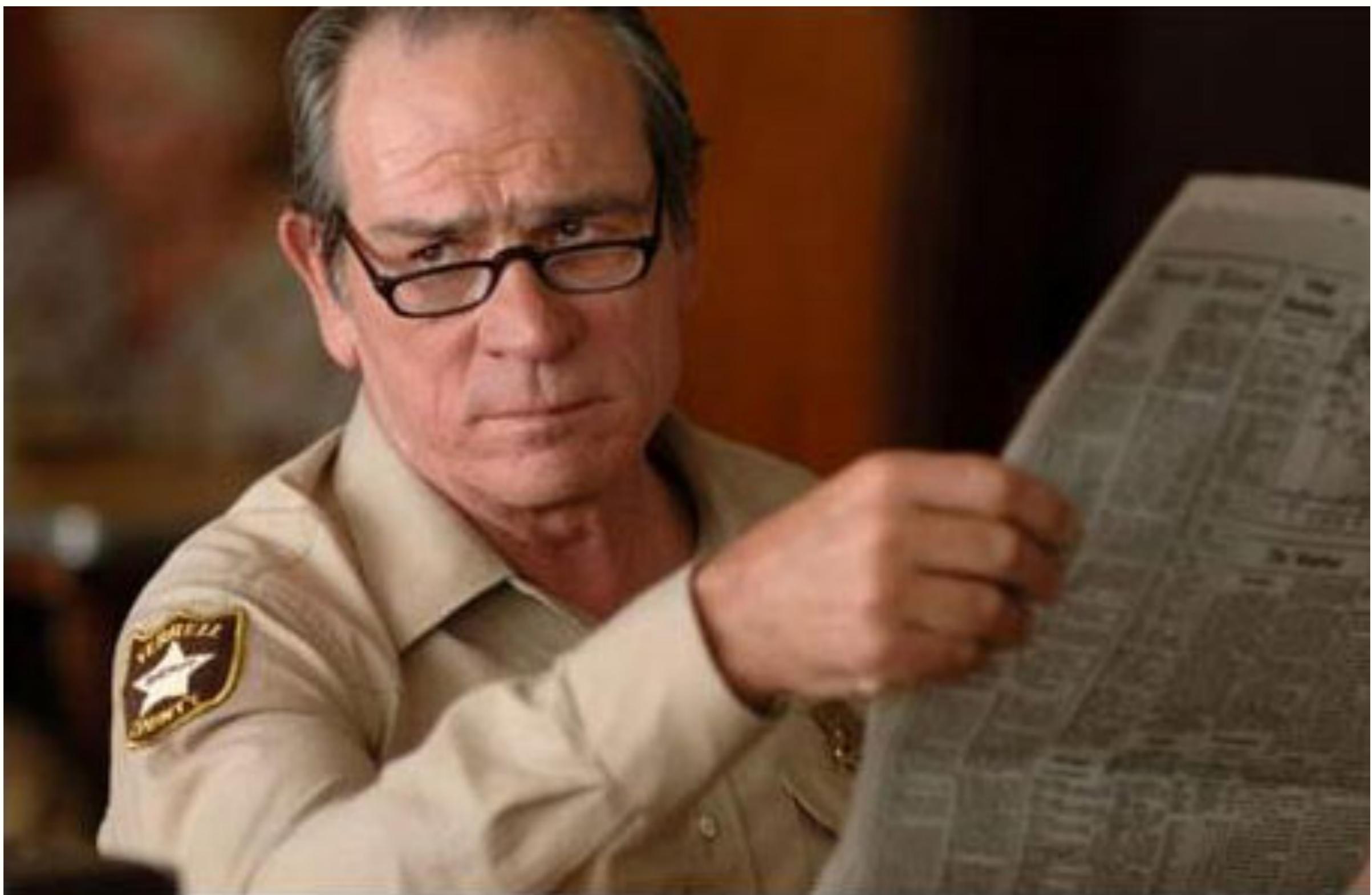
CONNECTIVITY DETECTION OPTIONS

- jQuery ajaxError event
- navigator.onLine

A QUICK ASIDE ABOUT NAVIGATOR.ONLINE

“This attribute is inherently unreliable. A computer can be connected to a network without having Internet access.”

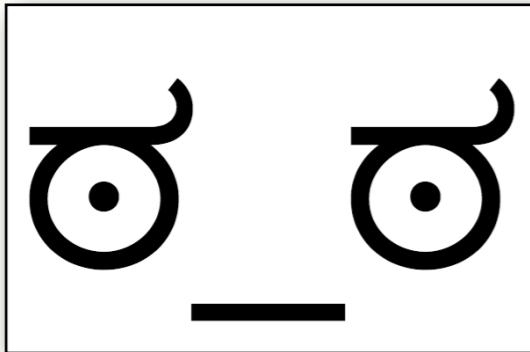
Hugs and Kisses,
- the W3C



CONNECTIVITY DETECTION

```
1 if (navigator.onLine) {  
2     save(customer);  
3 } else {  
4     queueUpForLater(customer);  
5 }
```

CONNECTIVITY DETECTION



Who wants to have this all over the application?

```
1 if (navigator.onLine) {  
2     save(customer);  
3 } else {  
4     queueUpForLater(customer);  
5 }
```

CONNECTIVITY DETECTION OPTIONS

- jQuery ajaxError event
- navigator.onLine
- addEventListener("online", handler);*

CONNECTIVITY DETECTION OPTIONS

- jQuery ajaxError event
- navigator.onLine
- addEventListener("online", handler);*
- window.applicationCache error

```
1 // assuming we have an app object
2 window.addEventListener("offline", function(){
3     app.setStatus("offline");
4 });
5 window.addEventListener("online", function(){
6     app.setStatus("online");
7 });
8 window.applicationCache.addEventListener(
9     "error",
10    function() {
11        app.setStatus("offline");
12    }
13 );
```



Is this better?
Than before, yes. Overall, NO.

```
1 // assuming we have an app object
2 window.addEventListener("offline", function(){
3     app.setStatus("offline");
4 });
5 window.addEventListener("online", function(){
6     app.setStatus("online");
7 });
8 window.applicationCache.addEventListener(
9     "error",
10    function() {
11        app.setStatus("offline");
12    }
13 );
```

CONNECTIVITY DETECTION OPTIONS

- jQuery ajaxError event
- navigator.onLine
- addEventListener("online", handler);*
- window.applicationCache error
- navigator.network.connection.type
(Cordova/PhoneGap)

Sure...

Single Source of Application State

>

Peppered Spaghetti Branching

But What About...

- The Commuter Problem
- False Negatives & Positives
- Deliberate choice to go offline
- Testability

We have lots of
different abstractions
for **similar** input



AN ABSTRACTION THAT...

Reacts differently to
the same input
depending on state

WANTED

MAKE THIS EASY...

While We Are	And This Happens	Let's Do This
Online	http request	send request to server
	window.offline	set app to offline
Offline	http request	queue request up
	window.online	set app to online

FINITE STATE MACHINE



WHAT IS A FINITE STATE MACHINE?

?

WHAT IS A FINITE STATE MACHINE?

- A computational abstraction that:
 - Has a finite number of states in which it can exist
 - Can only be in one state at any time
 - Accepts input
 - Can produce output determined by state &/ or input
 - Can transition from one state to another*

**THAT WAS A LOT OF
TEXT ON ONE SCREEN**

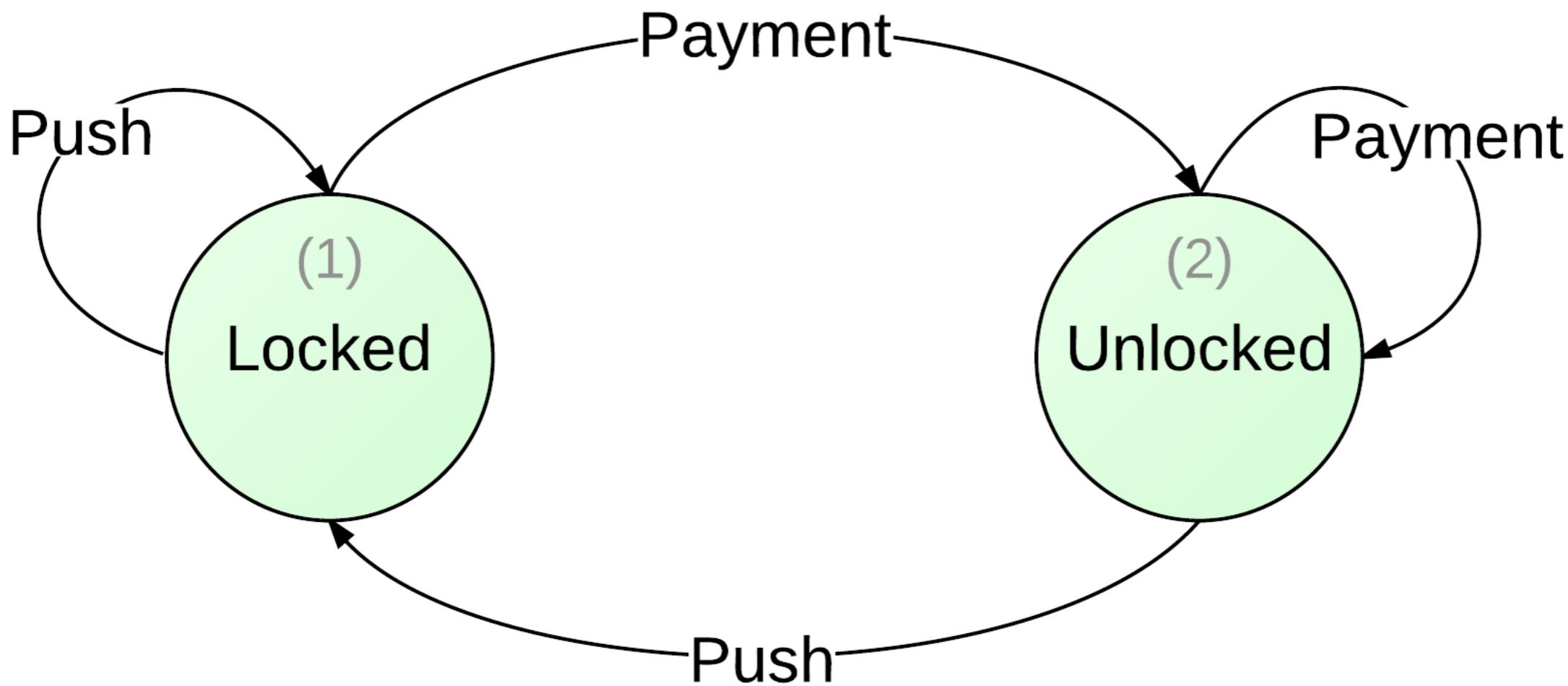


REAL WORLD EXAMPLE*

State	Input	Next State	Output
Locked	payment	Unlocked	turnstile released
	push	Locked	None
Unlocked	payment	Unlocked	None
	push	Locked	locks turnstile



DIAGRAMS FTW?



STILL AWAKE?

GOOD!

**HERE'S SOME FINE
PRINT ON STATE
MACHINES**

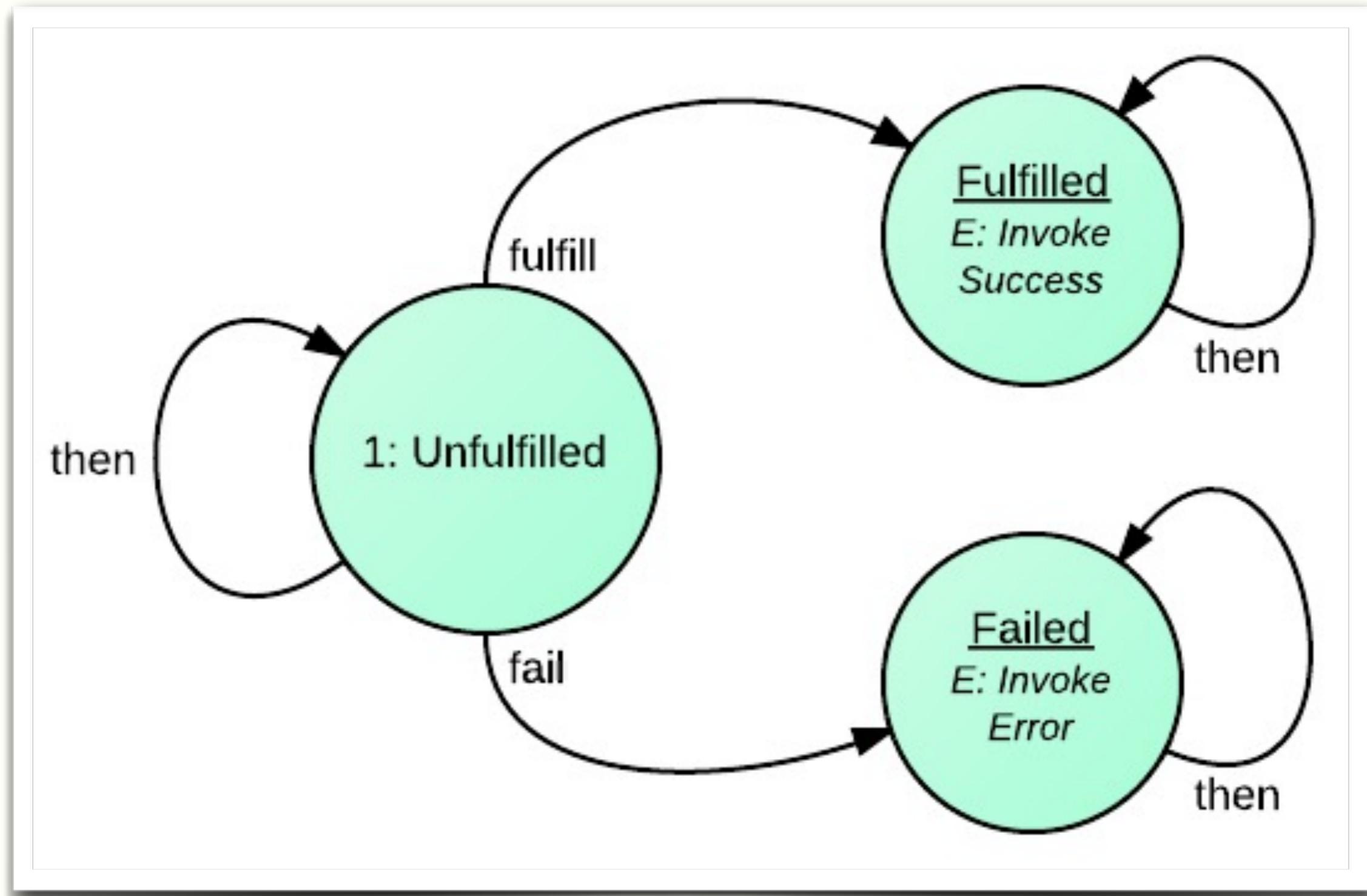
TWO BASIC TYPES

- Acceptor
- Transducer
 - Moore machine - output depends on state (entry actions)
 - Mealy machine - output depends on state and input

DETERMINISM

- Deterministic - only one transition possible for each input
- Non-deterministic - zero or more transitions possible from an input

PROMISES & FSMS



ENTER MACHINA.JS

ENTER MACHINA.JS

- STATES ARE ORGANIZED INTO A ‘STATES’ OBJECT ON THE FSM
- EACH OBJECT PROPERTY IS A STATE
- EACH STATE OBJECT’S MEMBERS ARE FUNCTIONS* THAT RESPOND TO INPUT
- CALLING “transition(stateName)” CHANGES STATE

ENTER MACHINA.JS

- FSM MAPS INPUT TO MATCHING HANDLER NAME:
“handle(inputName, args*)”
- “_onEnter” & “_onExit” & “*”
- “deferUntilTransition([stateName])”
- “deferUntilNextHandler()”
- BUILT-IN EVENT Emitter

Sigh...so
much API info...
what if I need to
blink?



OMG CODE!

Instances and Constructors

```
var fsm = new machina.Fsm({ ... });
```

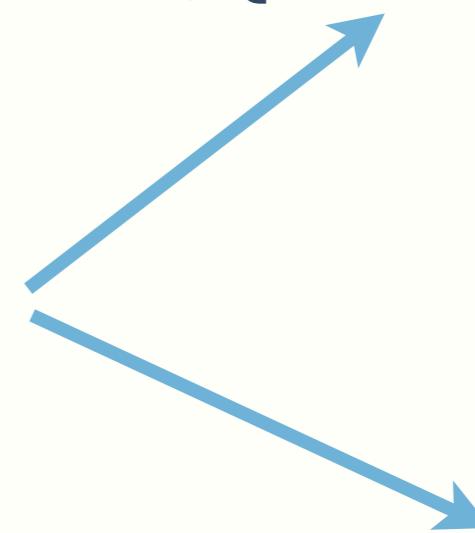
```
var Fsm = machina.Fsm.extend({ ... });
```

OMG CODE!

Instances and Constructors

```
var fsm = new machina.Fsm({ ... });
```

What goes here?



```
var Fsm = machina.Fsm.extend({ ... });
```

```
var turnstile = new machina.Fsm({  
  initialState: "locked",  
  states: {  
    locked: {  
      payment: "unlocked"  
    },  
    unlocked: {  
      push: "locked"  
    }  
  }  
});
```

```
var turnstile = new  
turnstile.  
  locked: {  
    payment: "unlocked"  
  }  
};
```

This is short for this:

```
turnstile.  
  locked: {  
    payment: function() {  
      this.transition("unlocked");  
    }  
  }  
};
```

```
var turnstile = new machina.Fsm({
  initialState: "locked",
  states: {
    locked: {
      payment: "unlocked"
    },
    unlocked: {
      push: "locked"
    }
  }
});
```

OVERSIMPLIFIED USAGE

```
// you could do this  
turnstile.handle("push"); //sorry, not so much  
turnstile.handle("payment"); //trans.-> unlocked  
turnstile.handle("payment"); //oops, wasted $$  
turnstile.handle("push"); //yay, I get through
```

OVERSIMPLIFIED USAGE

```
// but you'll probably prefer to do this
// i.e. - top level methods wrapping
// handle()

turnstile.push(); // sorry, not so much
turnstile.pay(); // transition-> unlocked
turnstile.pay(); // oops, wasted money
turnstile.push(); // yay, I get through
```

**HOW DO WE
APPLY THIS?**

**HOW DOES IT HELP MANAGE
CONNECTIVITY STATE?**

CONNECTIVITY STATES

- Online
- Offline (the user said so!)

CONNECTIVITY STATES

- Online
- Offline (the user said so!)
- Disconnected (Oops, no connection)

CONNECTIVITY STATES

- Online
- Offline (the user said so!)
- Disconnected (Oops, no connection)
- Probing (detecting if we're online)

ONLINE STATE INPUT & TRANSITIONS

State	Input	Next State	Output
Online	window.offline	Probing	Emit Transition Event
	appCache.error	Probing	Emit Transition Event
	request.timeout	Probing	Emit Transition Event
	go.offline	Offline	Emit Transition Event

OFFLINE STATE INPUT & TRANSITIONS

State	Input	Next State	Output
Offline	go.online	Probing	Emit Transition Event

DISCONNECTED STATE INPUT & TRANSITIONS

State	Input	Next State	Output
Disconnected	go.online	Probing	Emit Transition Event
	go.offline	Offline	Emit Transition Event
	window.online	Probing	Emit Transition Event
	appCache. downloading	Probing	Emit Transition Event

PROBING STATE INPUT & TRANSITIONS

State	Input	Next State	Output
Probing	heartbeat	Online	Emit Transition Event
	no-heartbeat	Disconnected	Emit Transition Event
	go.offline	Offline	Emit Transition Event

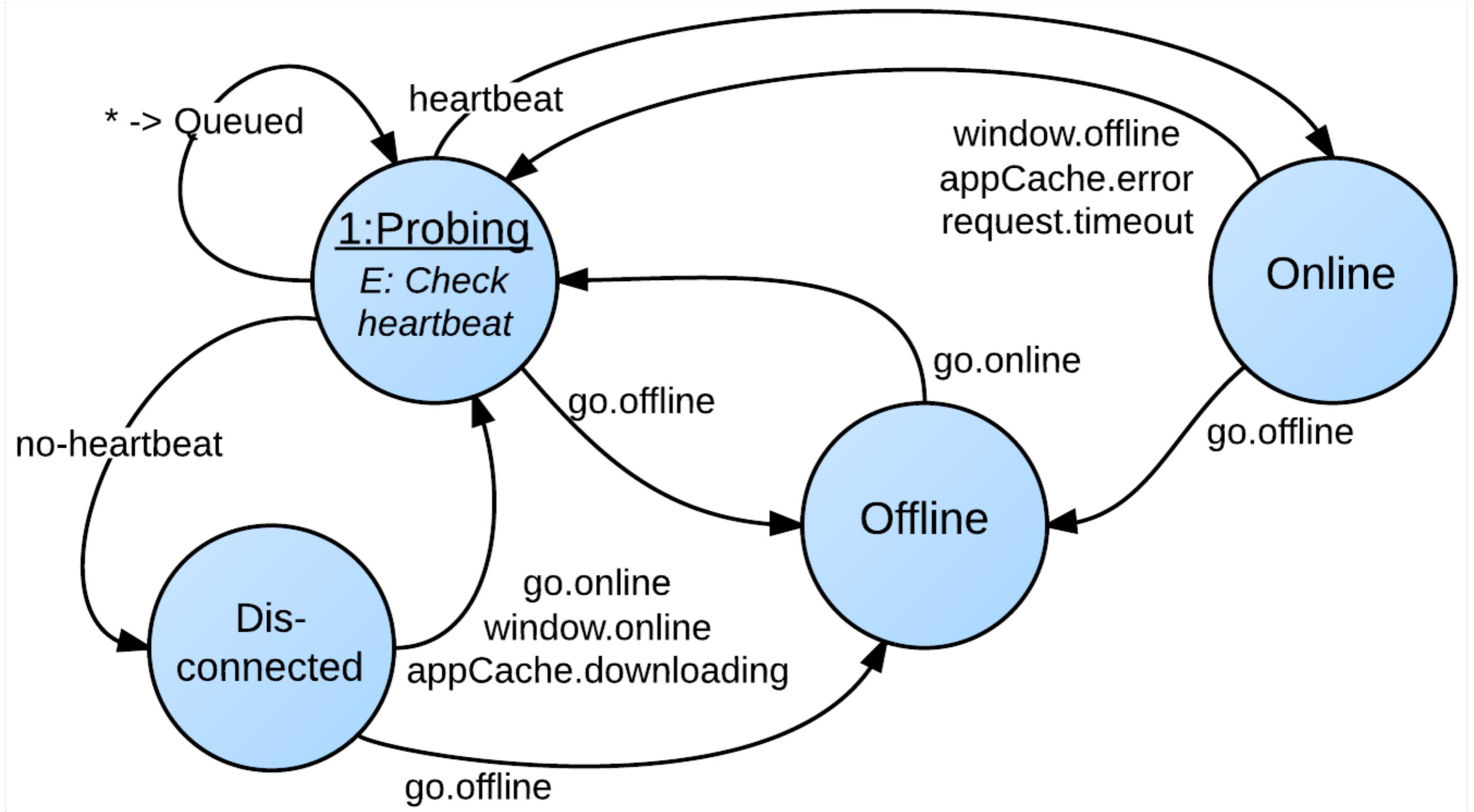
State	Input	Next State	Output
Online	window.offline	Probing	Emit Transition Event
	appCache.error	Probing	Emit Transition Event
	request.timeout	Probing	Emit Transition Event
	go.offline	Offline	Emit Transition Event
Offline	go.online	Probing	Emit Transition Event
Disconnected	go.online	Probing	Emit Transition Event
	go.offline	Offline	Emit Transition Event
	window.online	Probing	Emit Transition Event
	appCache.downloading	Probing	Emit Transition Event
Probing	heartbeat	Online	Emit Transition Event
	no-heartbeat	Disconnected	Emit Transition Event
	go.offline	Offline	Emit Transition Event

MY

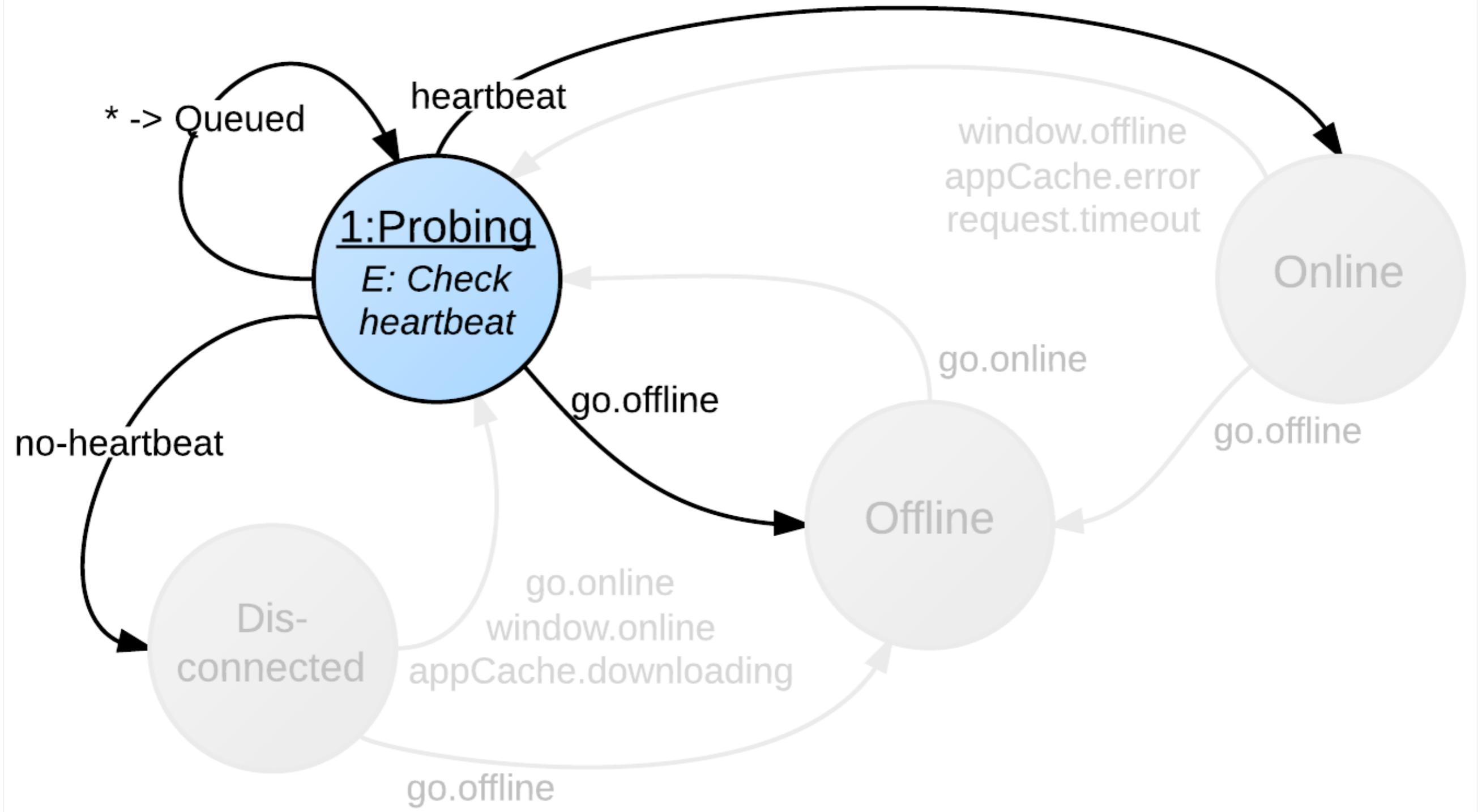
EYES

DIYLOL.COM

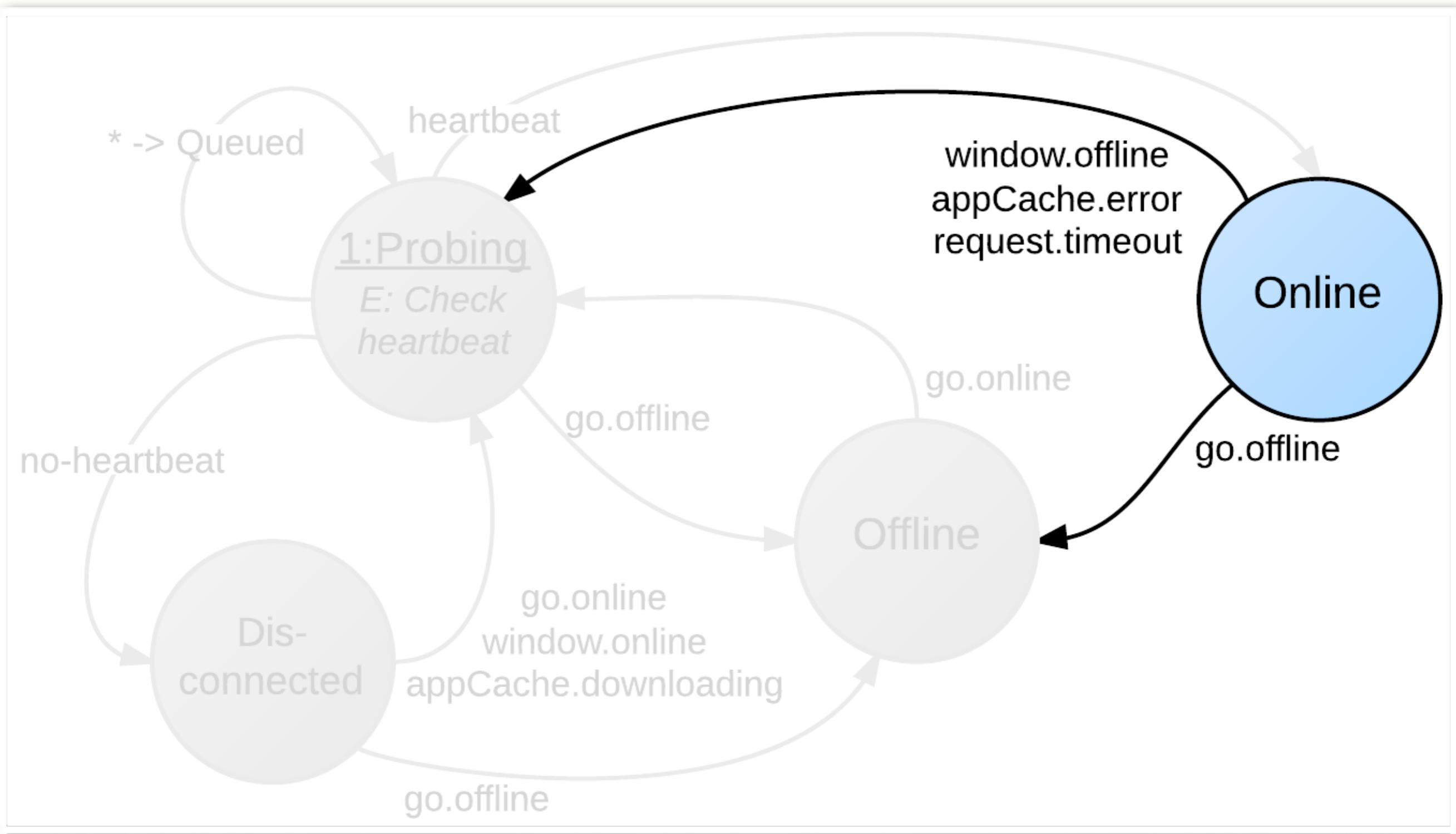
CONNECTIVITY STATE MACHINE



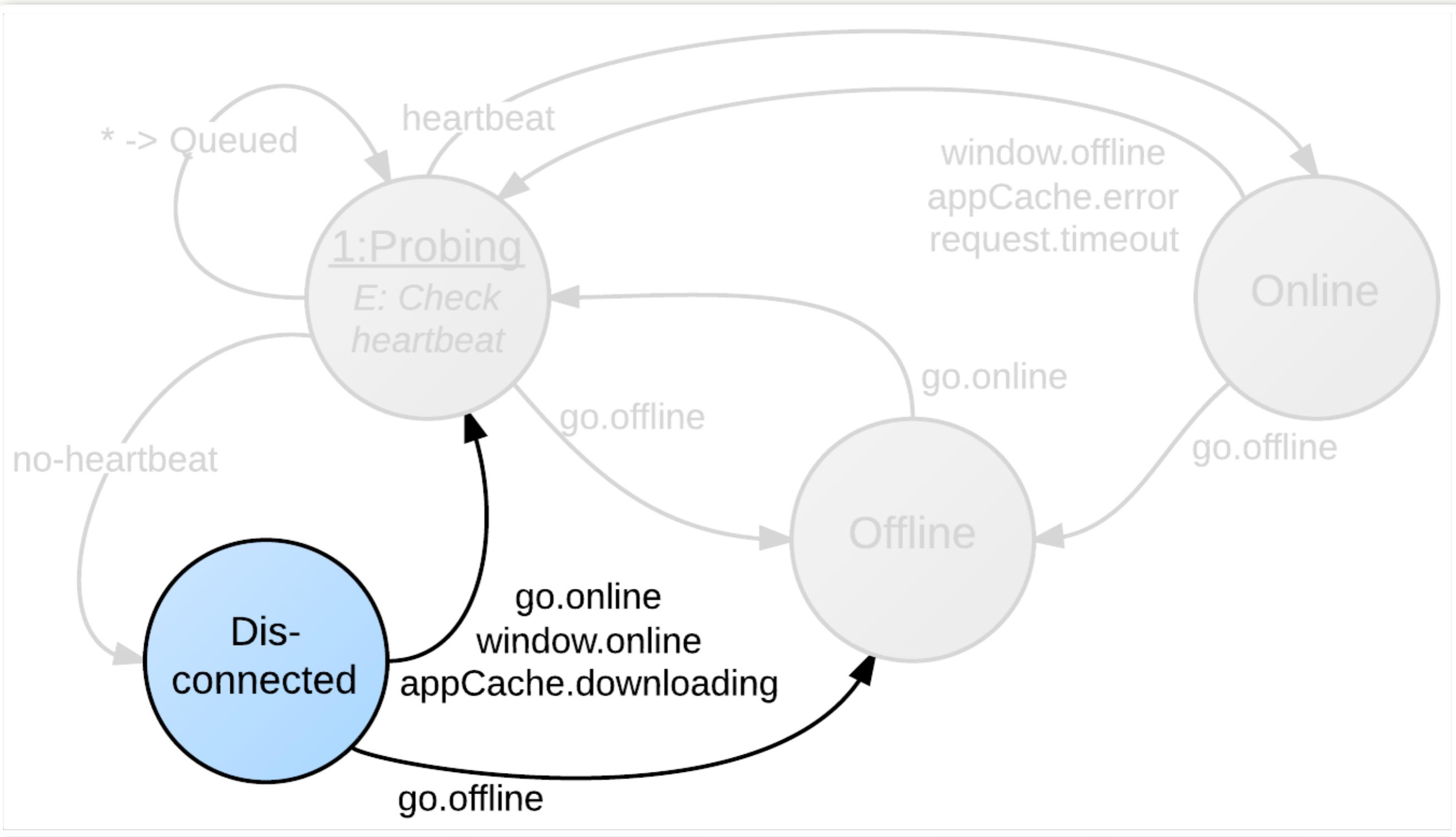
CONNECTIVITY STATE MACHINE



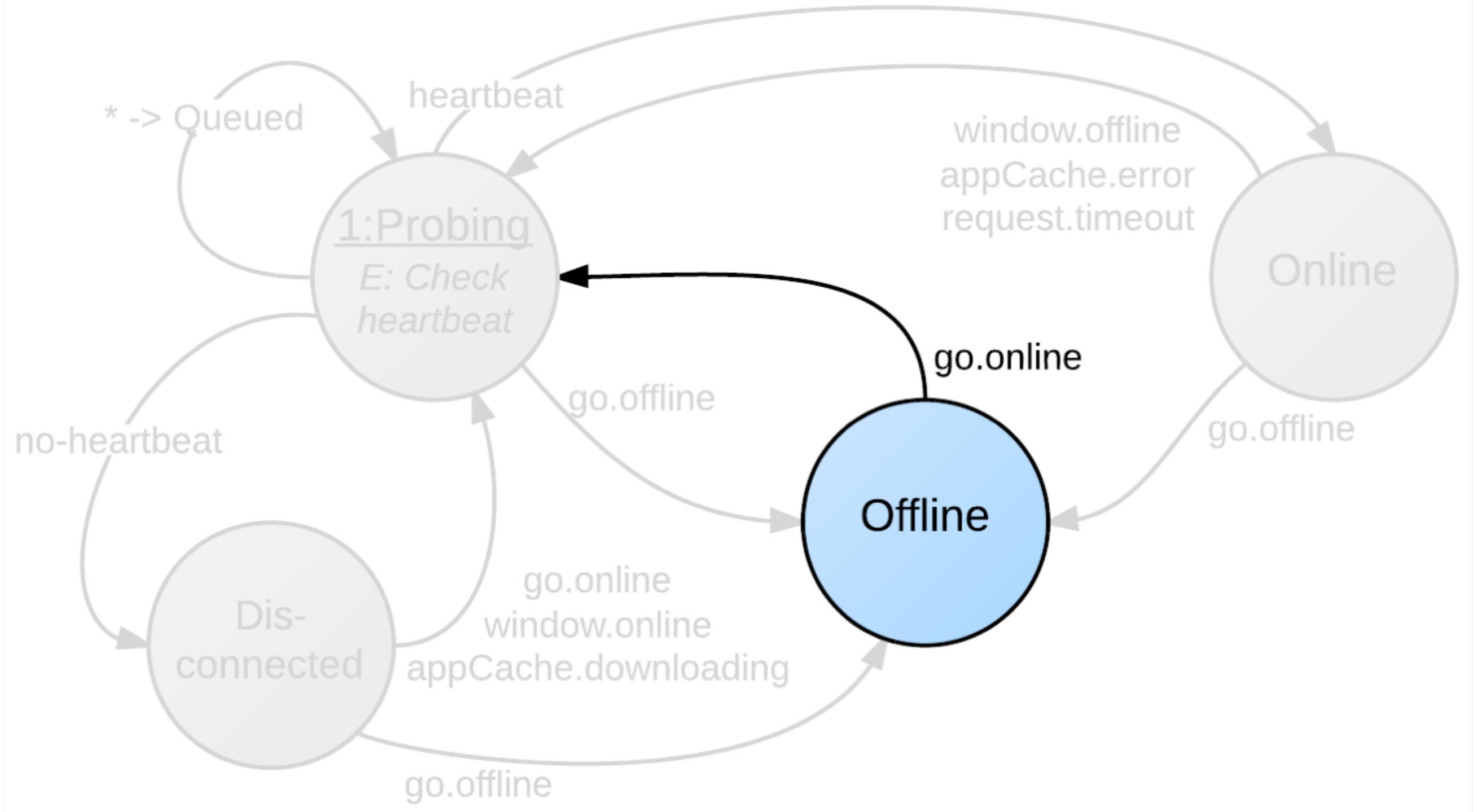
CONNECTIVITY STATE MACHINE



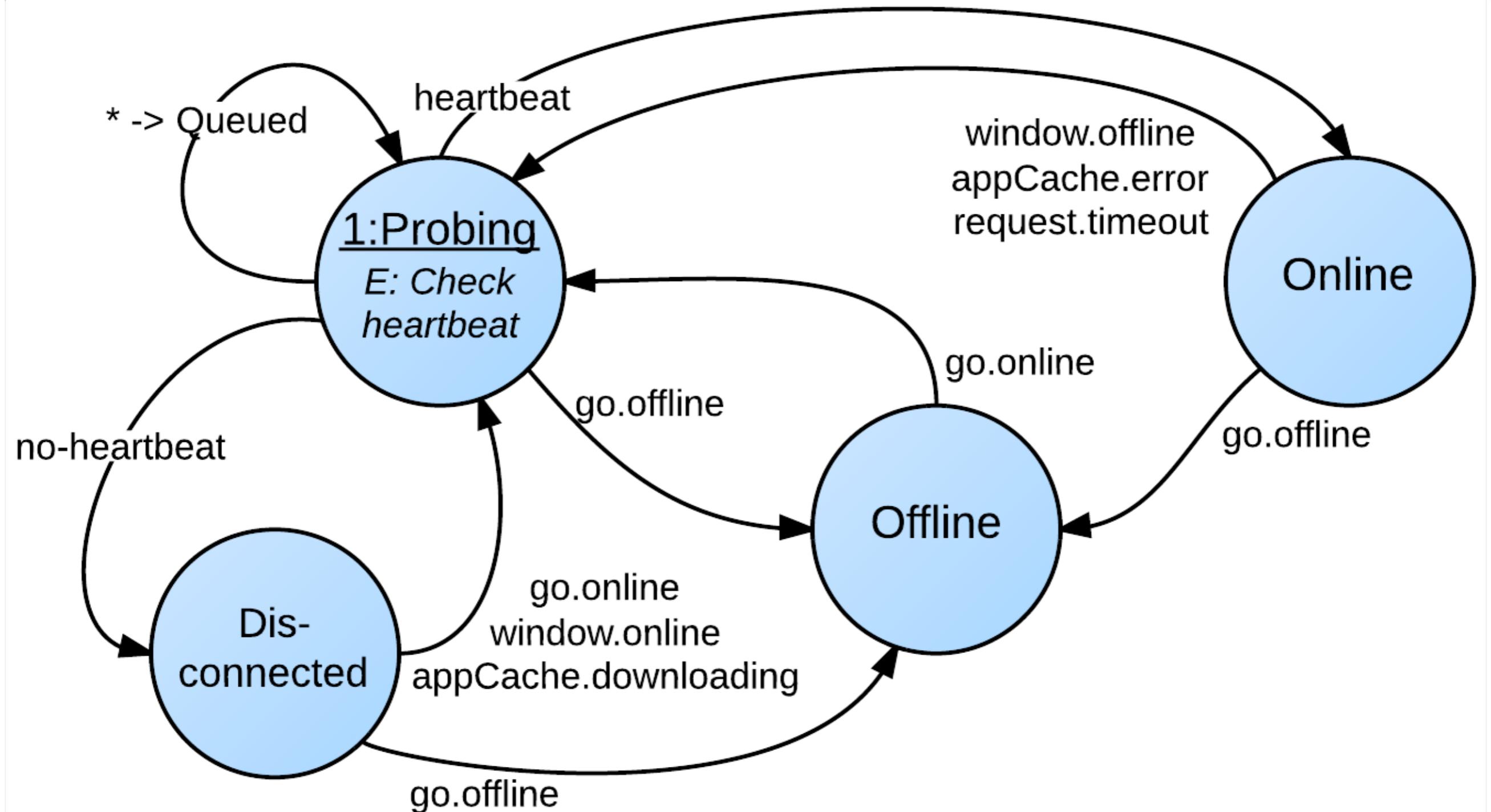
CONNECTIVITY STATE MACHINE



CONNECTIVITY STATE MACHINE



CONNECTIVITY STATE MACHINE



CODE



Child's Toy Simplicity

**WAIT...WHAT
HAPPENED TO THE
HTTP BEHAVIOR?**

SIBLING STATE MACHINES

Connectivity
FSM

Communications
FSM
State: Queuing

SIBLING STATE MACHINES

Connectivity
FSM

I emit
events

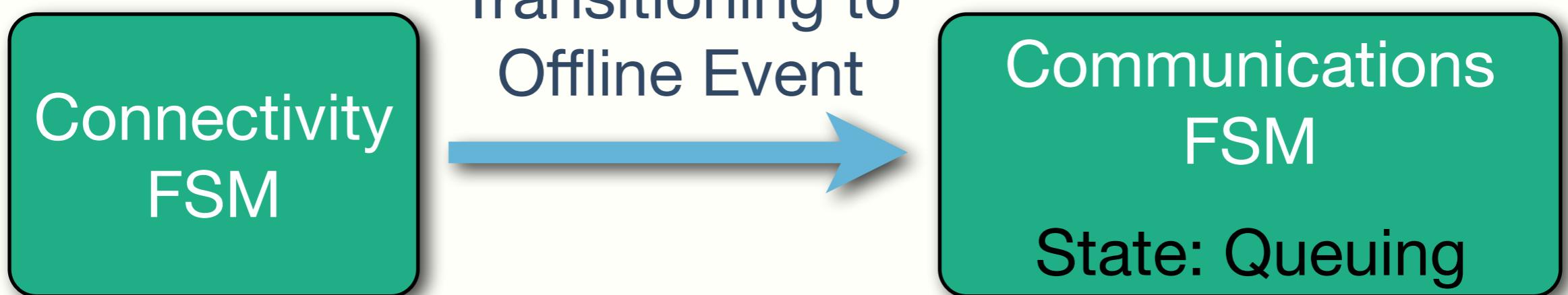
Communications
FSM
State: Queuing

Sweet. I'll
listen...

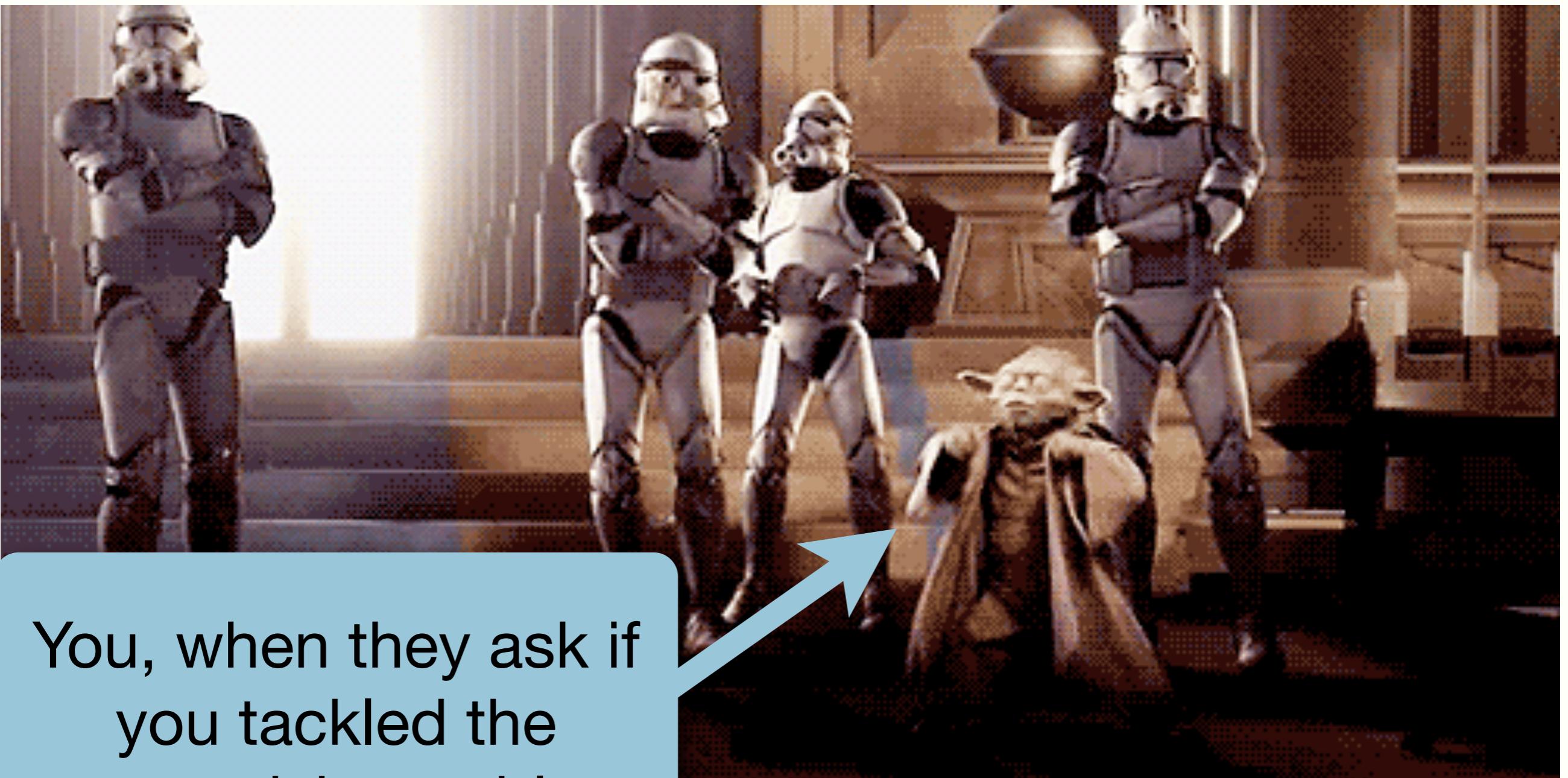
SIBLING STATE MACHINES



SIBLING STATE MACHINES



**FSMS WORKING TOGETHER:
POWERFUL WAY TO MANAGE
& ISOLATE COMPLEXITY**



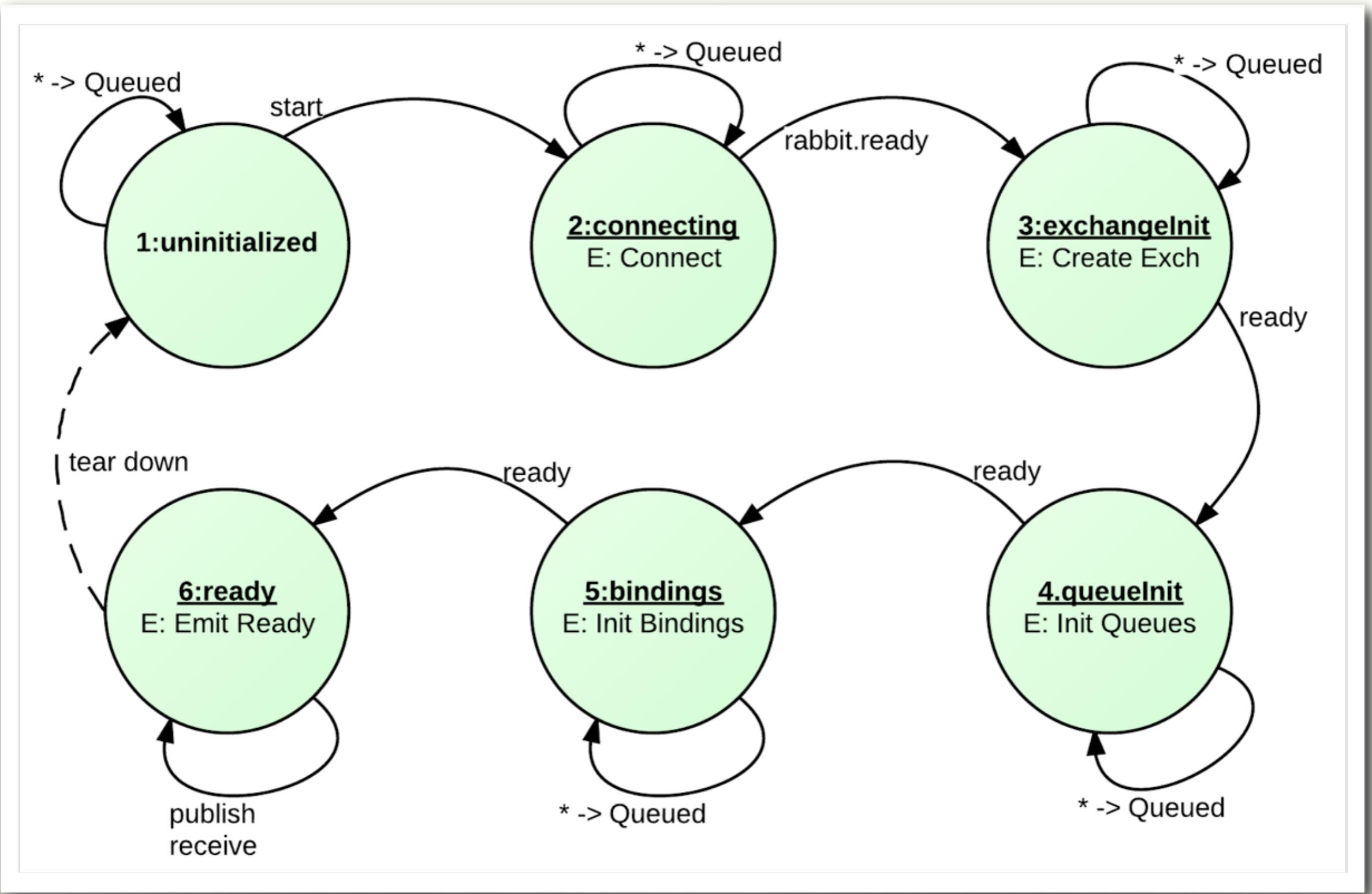
You, when they ask if
you tackled the
connectivity problem
in an extensible way...

SO IT TURNS OUT THAT...

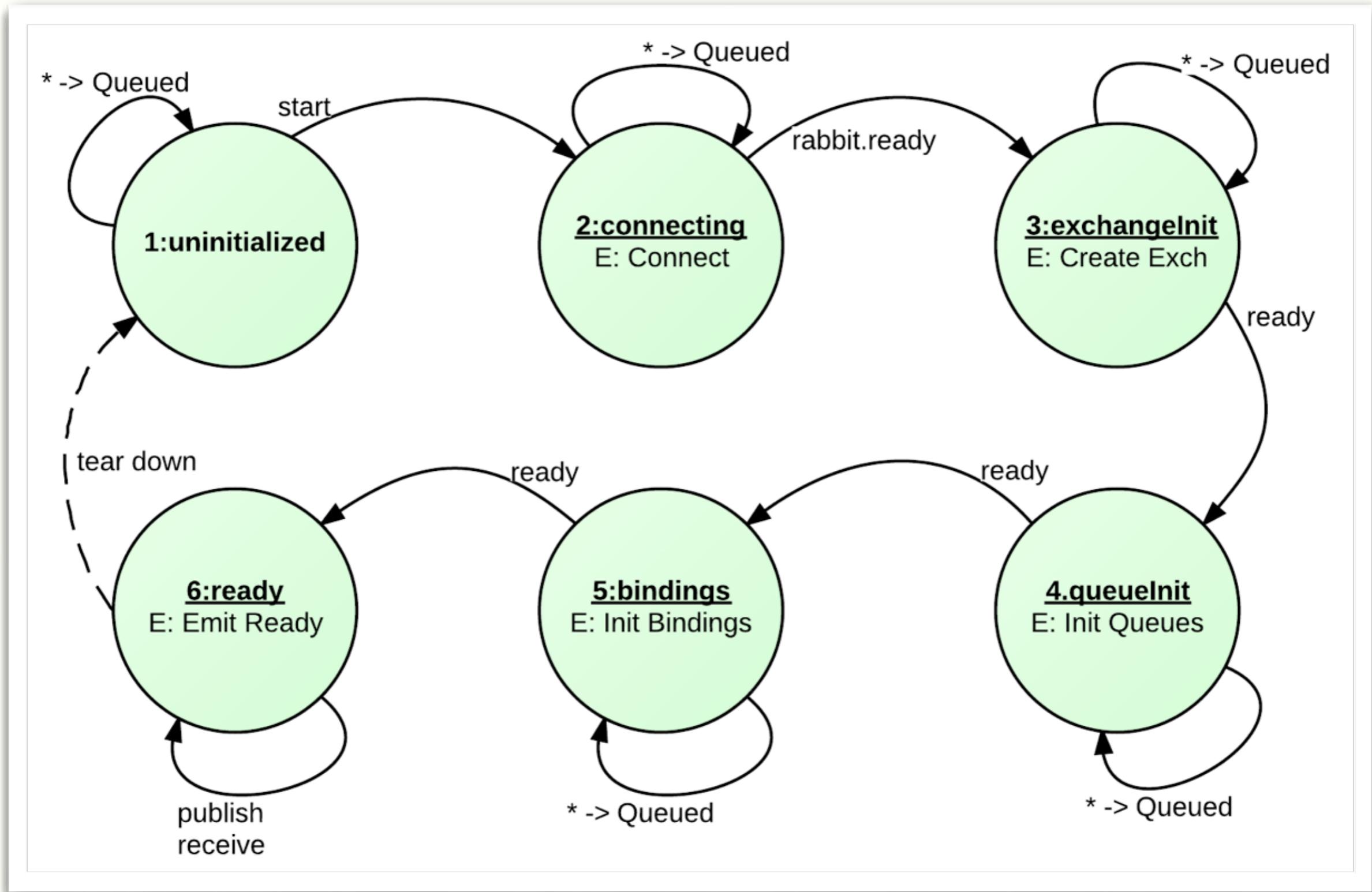
- This pattern can be useful for other things:
 - Handling complex UI workflow
 - Initialization

UI WORKFLOW

CODE



INITIALIZATION STATE MACHINE



CODE

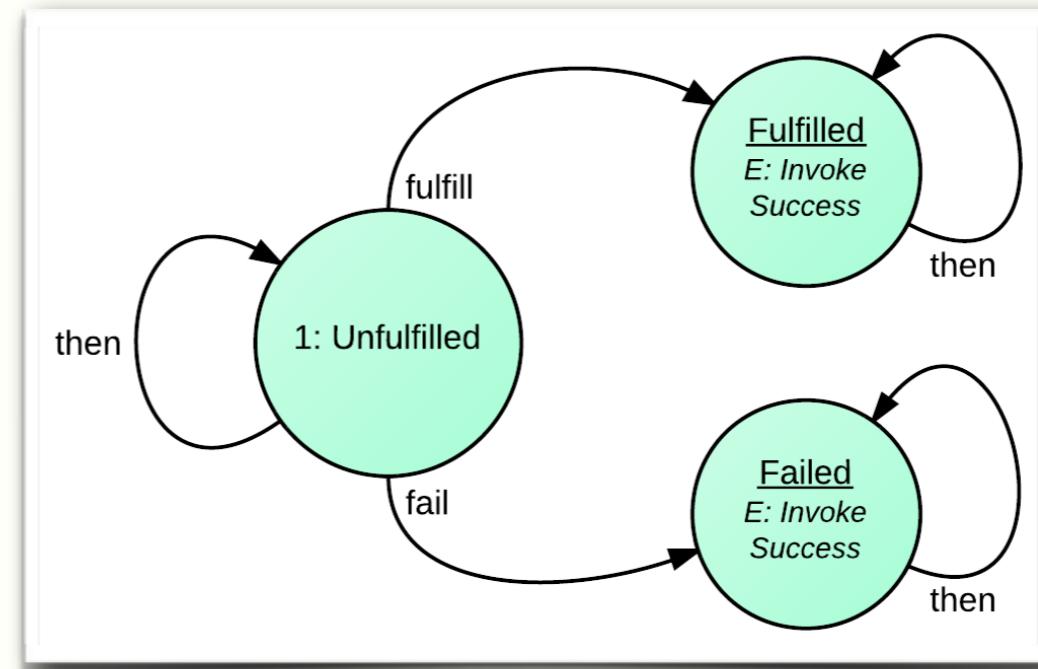
See: <https://github.com/a2labs/amqp-bootstrapper>

PROMISES & FSMS

CODE

(FEEL FREE TO CHECK THIS OUT ON YOUR OWN)

[HTTPS://GITHUB.COM/A2LABS/MACHINA.PROMISE](https://github.com/a2labs/machina.promise)



PROS & CONS

- PROS
- EXTREMELY VERSATILE
- LENDS WELL TO GOOD SEPARATION OF CONCERNS
- GREAT FOR LONG-RUNNING ASYNC WORKFLOWS
- EXPRESSIVE

PROS & CONS

- CONS
- MODELING COMPLEX/HIERARCHICAL FSMS IS “HARD”
- LESS FAMILIAR PATTERN (FOR MANY)

FURTHER RESOURCES

- [Finite State Machine - Wikipedia](#)
- [Taking Control With machina \(Doug Neiner\)](#)
- [Learn You Some Erlang - Finite State Machines](#)
- [machina.js on FreshBrewedCode](#)
- [machina.js on github](#)
- “[state](#)” - cool FSM project by Nick Fargo
- [Harvey Mudd CS paper on FSMs](#)

THESE SLIDES

[https://github.com/ifandelse/
DenverJS-December-2013](https://github.com/ifandelse/DenverJS-December-2013)

Q & A

PRESENTATION OVER



QUESTION ALL THE THINGS!