
THE IMMUTABLE FRONTEND IN CLOJURESCRIPT

Logan Linn (@loganlinn)
QCon SF 2014



PRISMATIC

- Personalized, interest-based newsfeeds
- Crawlers, Machine Learning, Clients
- We're very functional
 - 99.9% Clojure backend
 - ClojureScript frontend
- We <3 open-source



Home

Social

Global

Explore

Search

Profile

Activity

Interests

Saved

Feedback

Settings

 **danielszmu**
shared on Twitter

Clojure ✓

Concurrency


verbs, nouns and file watch semantics

Something Same • 2d

I've recently had a fascination with file watchers semantics in clojure libraries. Having trialed bunch of them in the past, I decided that it was time to have a go at one myself and

wanted to share some of my thoughts: Typically file watchers are implemented using either one of two patterns: verb based - (add-watch directory callback options)noun based - (start...

  4    1

 **carolyn**
liked

San Francisco, California ✓

Real Estate

Housing

How Do You Afford S.F.?

Modern Luxury • 3d

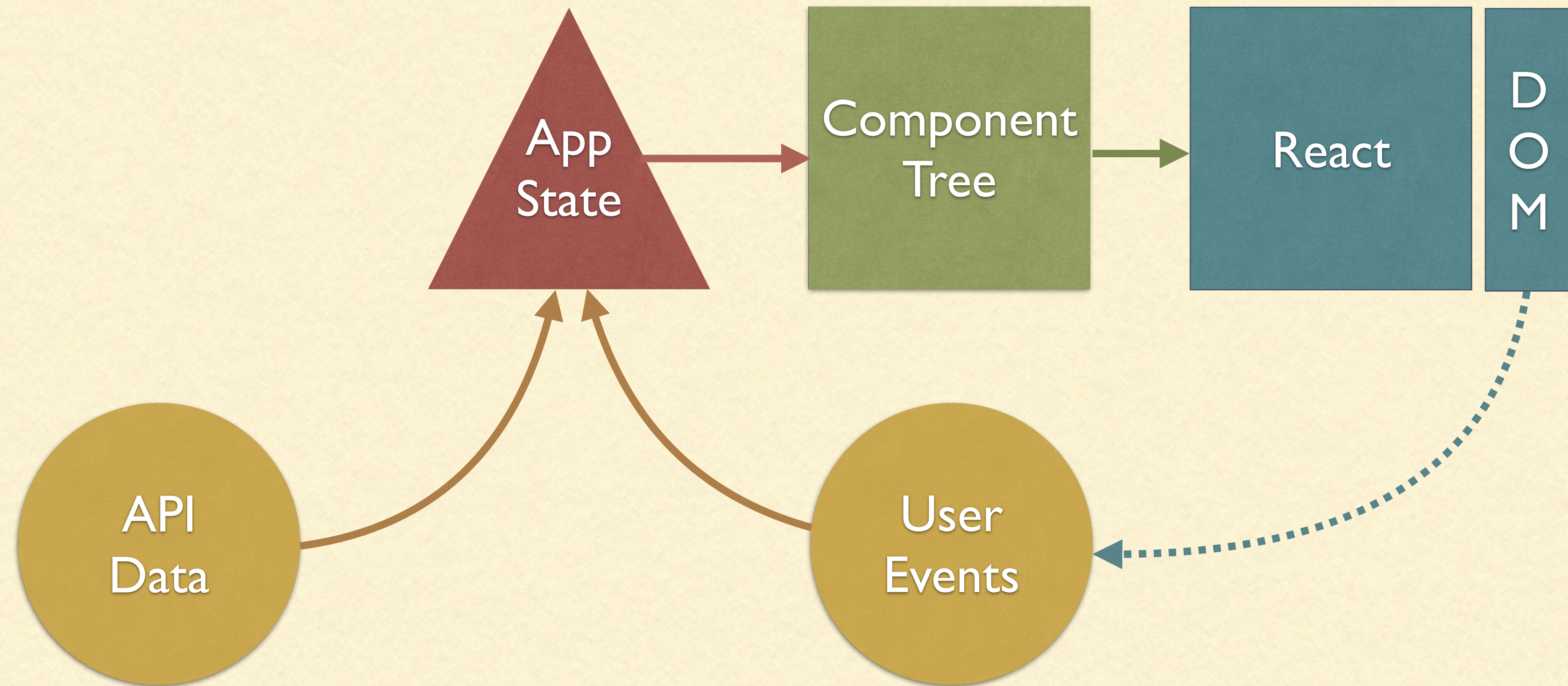
This is part of "Live Large, Spend Less," a comprehensive guide to surviving (and even flourishing) in America's most expensive city. See all of the stories here. Janelle Cruz, 27 Income: \$1,200-\$2,000 a month "I



IMMUTABLE FRONTEND

- ClojureScript gives us immutability and more
- Immutability simplifies data-flow
- React allows us to render predictably with pure functions

IMMUTABLE FRONTEND



OUTLINE

- Challenges of a Building a Frontend
- Immutability
- ClojureScript
- An Immutable Frontend

CHALLENGES OF BUILDING A FRONTEND

- Interactive UIs have a human factor
- Asynchronous programming
- Complexity comes from every angle
 - Software complexity is a compounding debt

INCIDENTAL COMPLEXITY

- Managing state

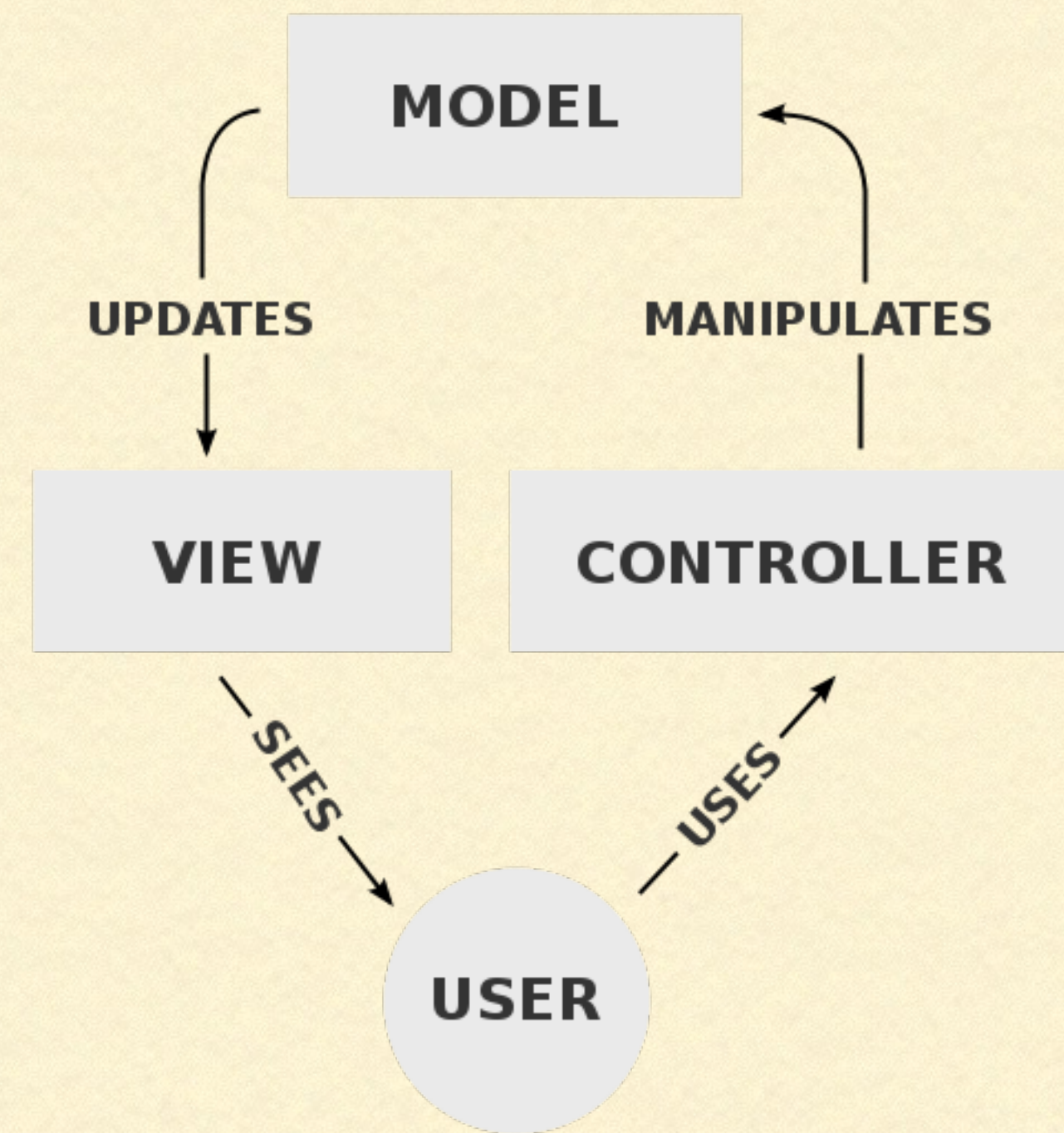
INCIDENTAL COMPLEXITY

- Managing state
- Mutating data

MODEL-VIEW-*

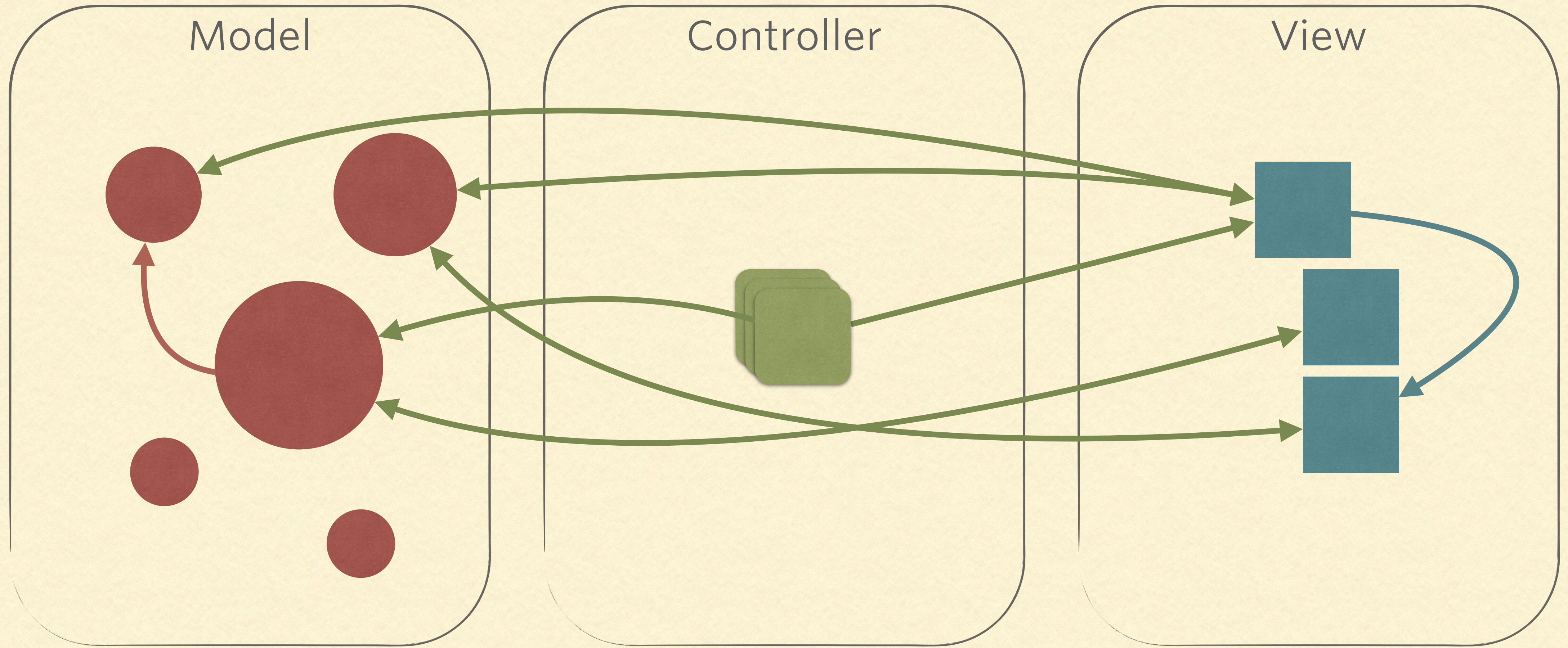
animate

- Domain vs Presentational data
- Keeping data and DOM in sync
- MVC, MVP, MVVM, etc.



EVENTS AND DATA-BINDING

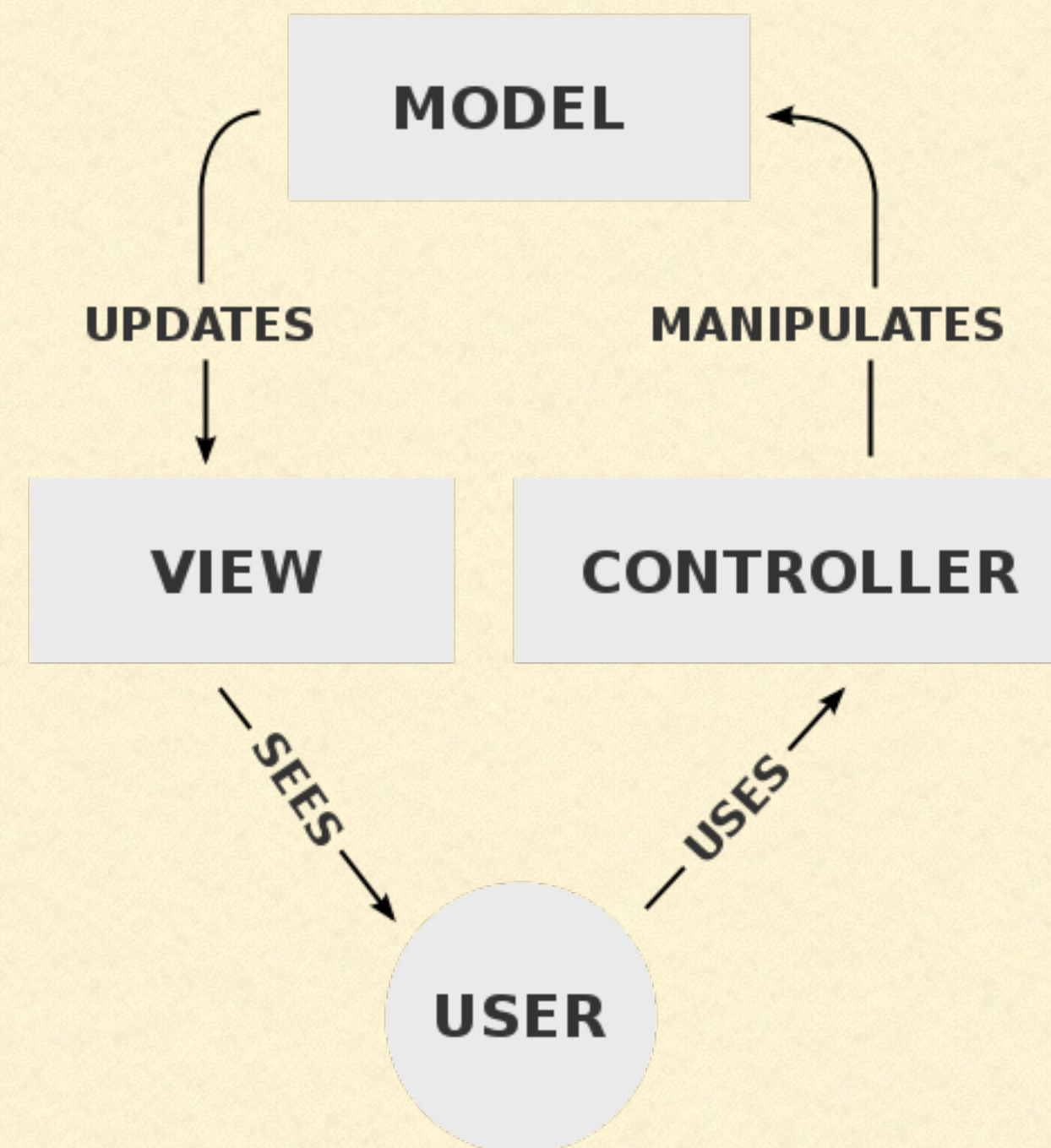
- Most frameworks today structured around Models
 - Models publish changes via global events
 - Views subscribe to changes and update
- Data bindings let you be declarative



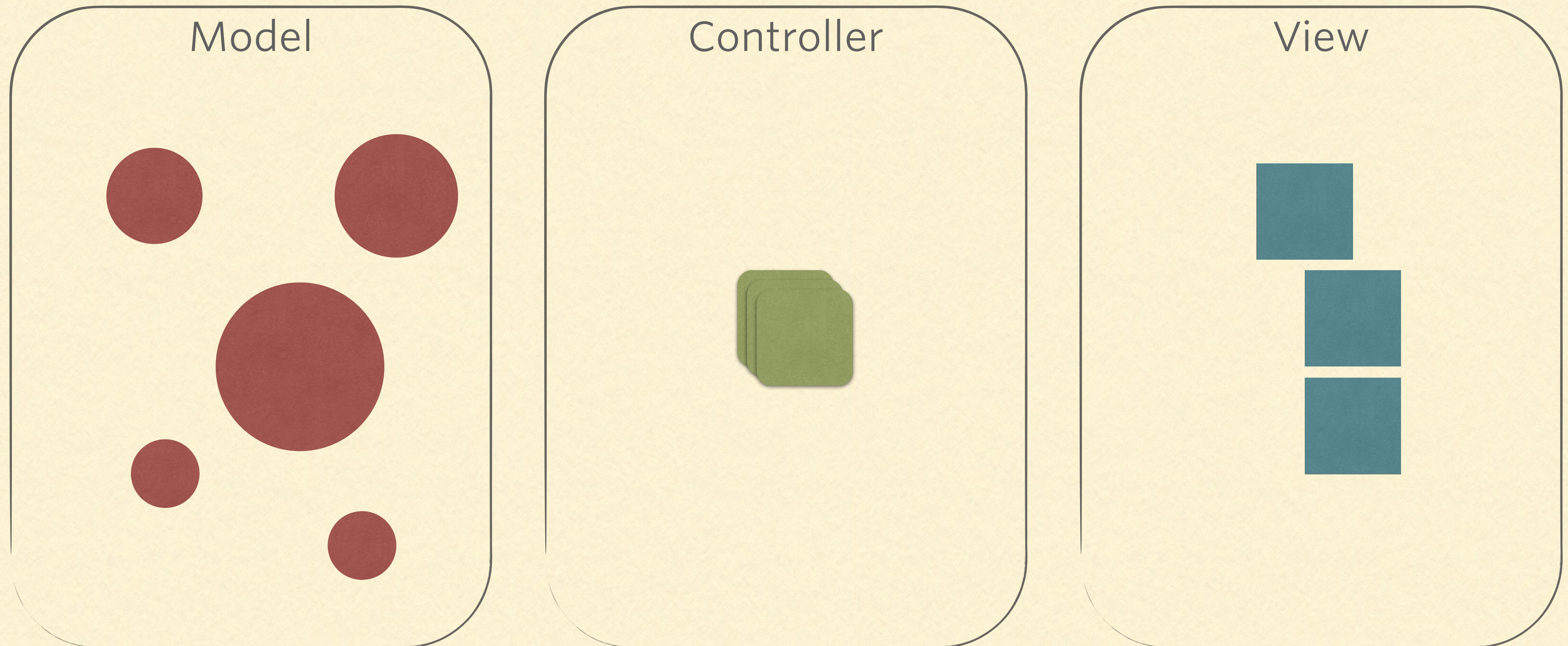
EVENT AND DATA-BINDING

- Encourages mutation
- Data-flow becomes opaque, potentially hazardous
- Makes it easier, but not simpler

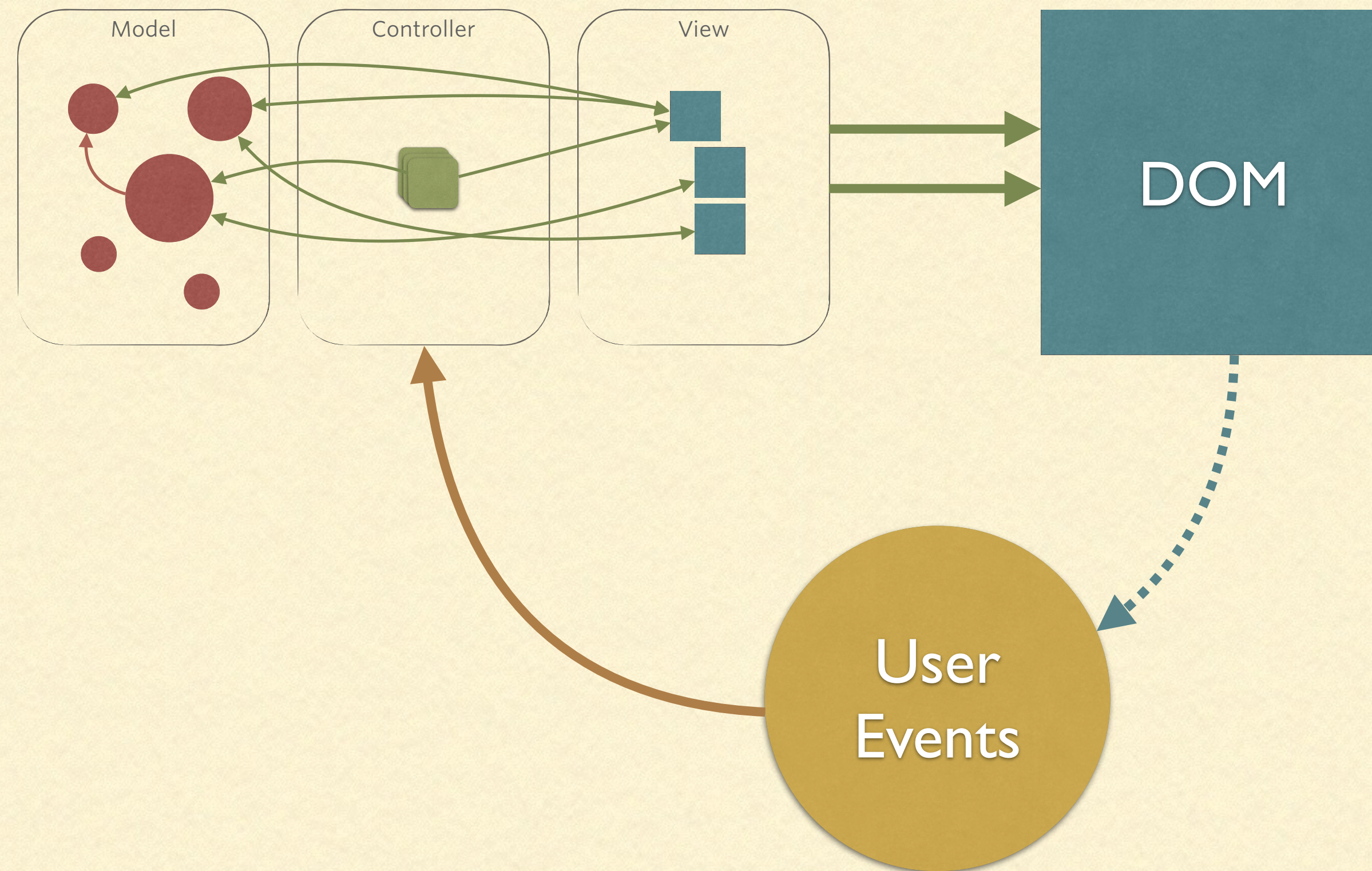
MVC DATA-FLOW



MVC DATA-FLOW

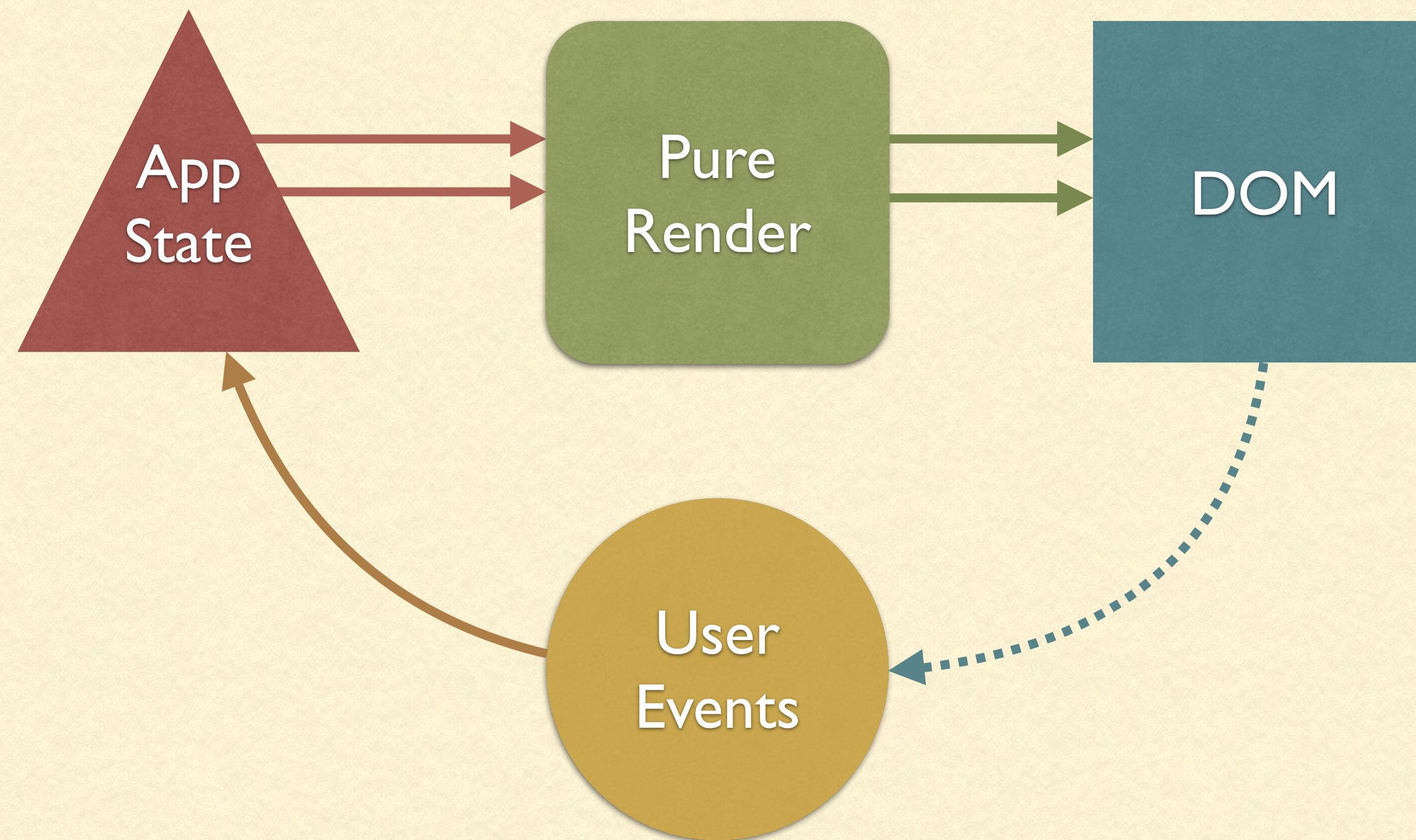


MVC DATA-FLOW



What if we prioritized a *simple* data-flow?

SINGLE-DIRECTION DATA FLOW



IMMUTABILITY

A MUTABLE WORLD

```
x = Domain.List.from(['x'])
```

```
y = x.unshift('y')
```

```
z = x.unshift('z')
```

```
print(z.second()) // 'x' or 'y'?
```

```
print(x) // ['x'] or ['z', 'y', 'x']?
```

AN IMMUTABLE WORLD...

```
x = Domain.List.from(['x'])
```

```
y = x.unshift('y')
```

```
z = x.unshift('z')
```

```
print(z.second()) // 'x', final answer!
```

```
print(x)          // ['x'], fasho!
```

RENDERING WITH PURE FUNCTIONS

$$\begin{array}{l} t \\ \downarrow \\ f(S_1) = D_1 \\ f(S_2) = D_2 \\ f(S_1) = D_1 \end{array}$$

IMMUTABILITY ON THE FRONTEND

- Simplicity & Clarity

IMMUTABILITY ON THE FRONTEND

- Simplicity & Clarity
- Predictability

IMMUTABILITY ON THE FRONTEND

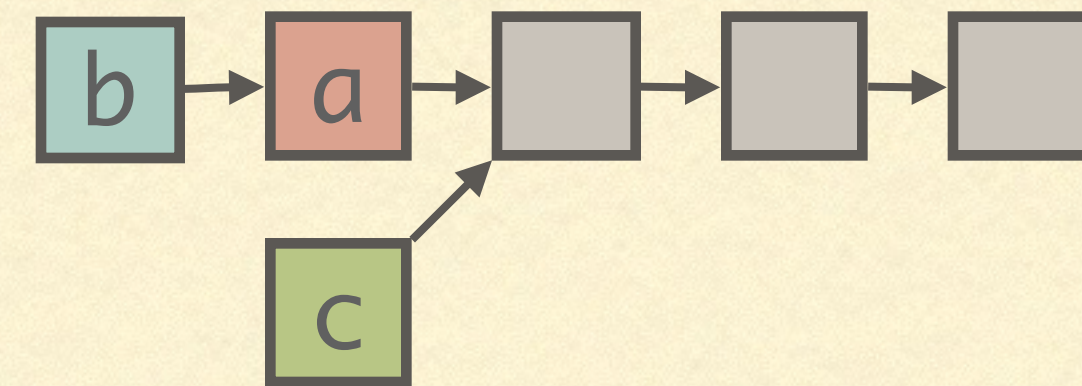
- Simplicity & Clarity
- Predictability
- Less defensive programming, i.e. `_.cloneDeep(obj)`

IMMUTABILITY ON THE FRONTEND

- Simplicity & Clarity
- Predictability
- Less defensive programming, i.e. `_.cloneDeep(obj)`
- Constant time dirty checking

IMMUTABILITY & PERFORMANCE

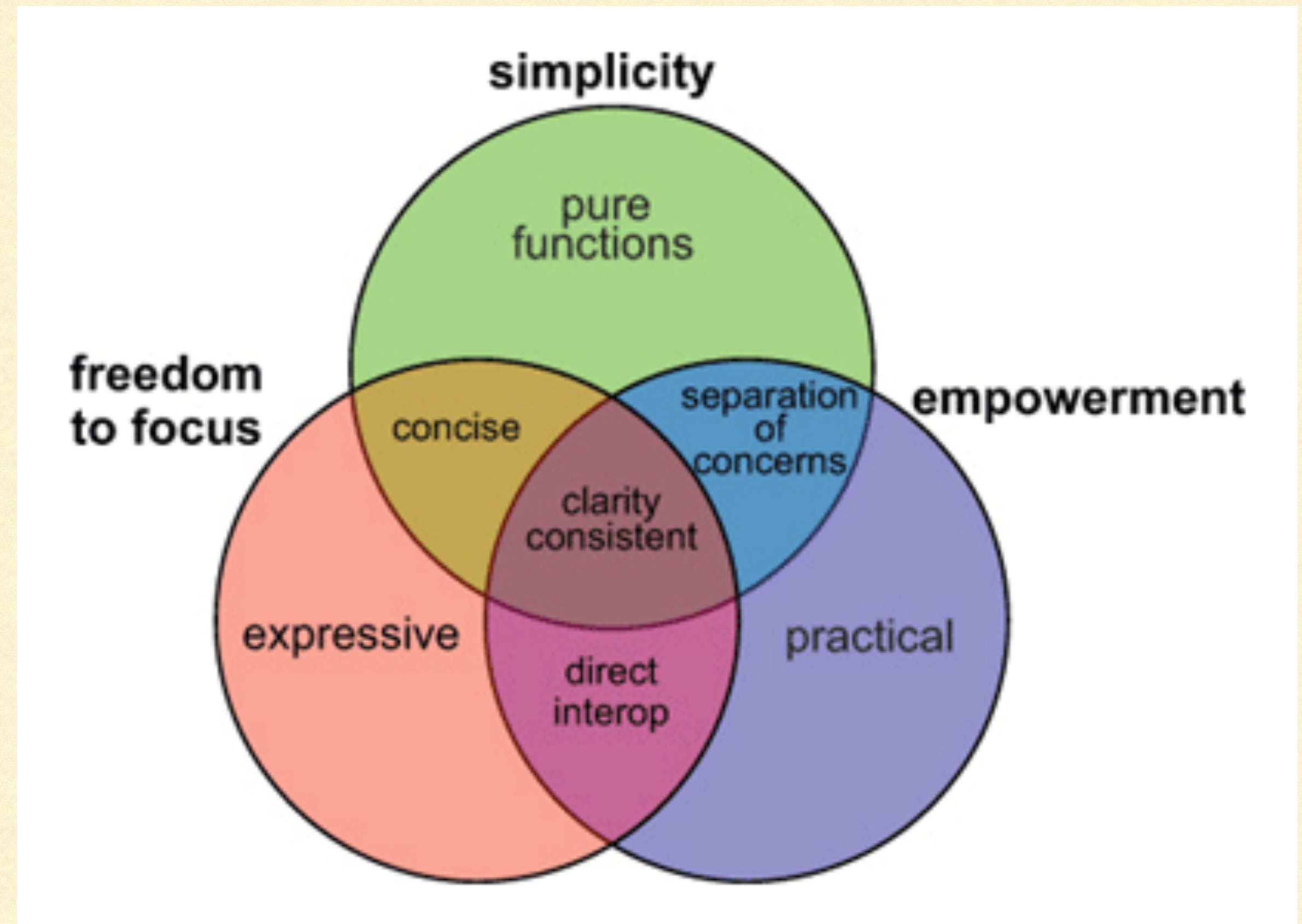
- Persistent data structures
- Structural sharing
 - Memory efficiency
 - Conjoin to collection in $O(1)$
 - Update hash-map in $O(\log_{32} n)$ vs $O(n)$



CLOSURE(SCRIPT)

CLOJURE & CLOJURESCRIPT

- Dynamic, Functional, Lisp
- Clojure
 - Compiles to JVM bytecode
 - 7 years old
- ClojureScript
 - Compiles to JavaScript
 - 3 years old



WHY WE LIKE CLOJURESCRIPT

- Clarity & Consistency
- Strong core library over clean abstractions
- Macros
- Share code with rest of code-base

**"It is better to have 100
functions operate on one
data structure than 10
functions on 10 data
structures."**

—Alan Perlis, 1982

MUTATION REQUIRES OPT-IN

- Immutable data by default
- State modeled with reference to immutable value
- Special functions to mutate reference & dereference value
- Easy to identify side-effects

```
(def state-ref (atom {}))
```

```
(deref state-ref)      ;; => {}
```

```
(reset! state-ref {:a 1})
```

```
@state-ref             ;; => {:a 1}
```

```
(defn increment-a [state]
  (update-in state [:a] inc))
```

```
(increment-a @state-ref) ;; => {:a 2}
```

```
@state-ref             ;; => {:a 1}
```

```
(swap! state-ref increment-a)
```

```
@state-ref             ;; => {:a 2}
```

SEPARATION OF CONCERNS

- **Time**

- Relative moments when events occur

- **State**

- A value at a point in time

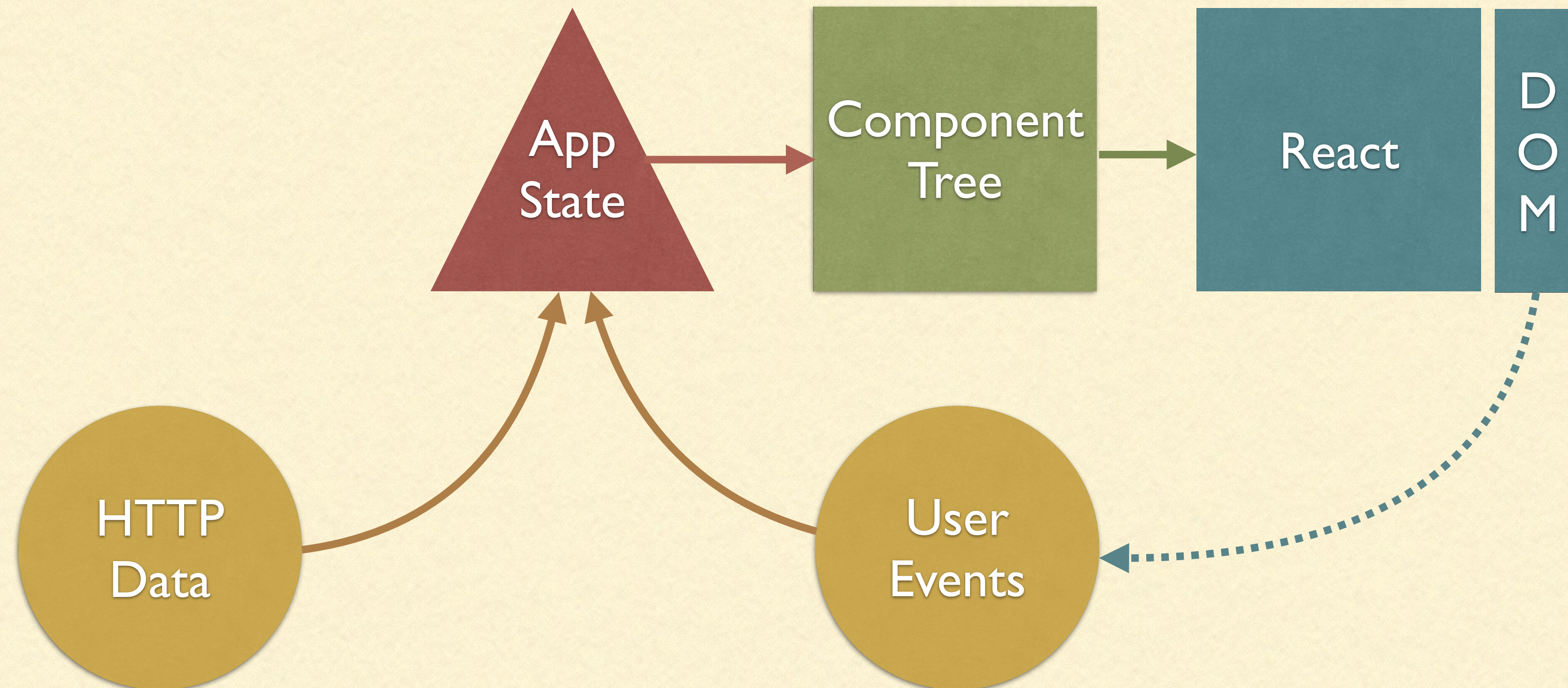
- **Identity**

- Entity associated with state over time

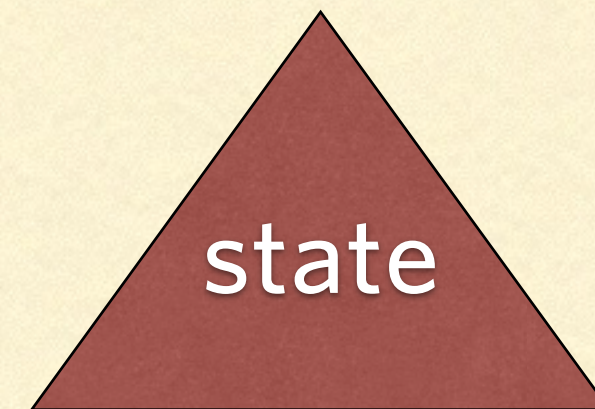


AN IMMUTABLE ARCHITECTURE

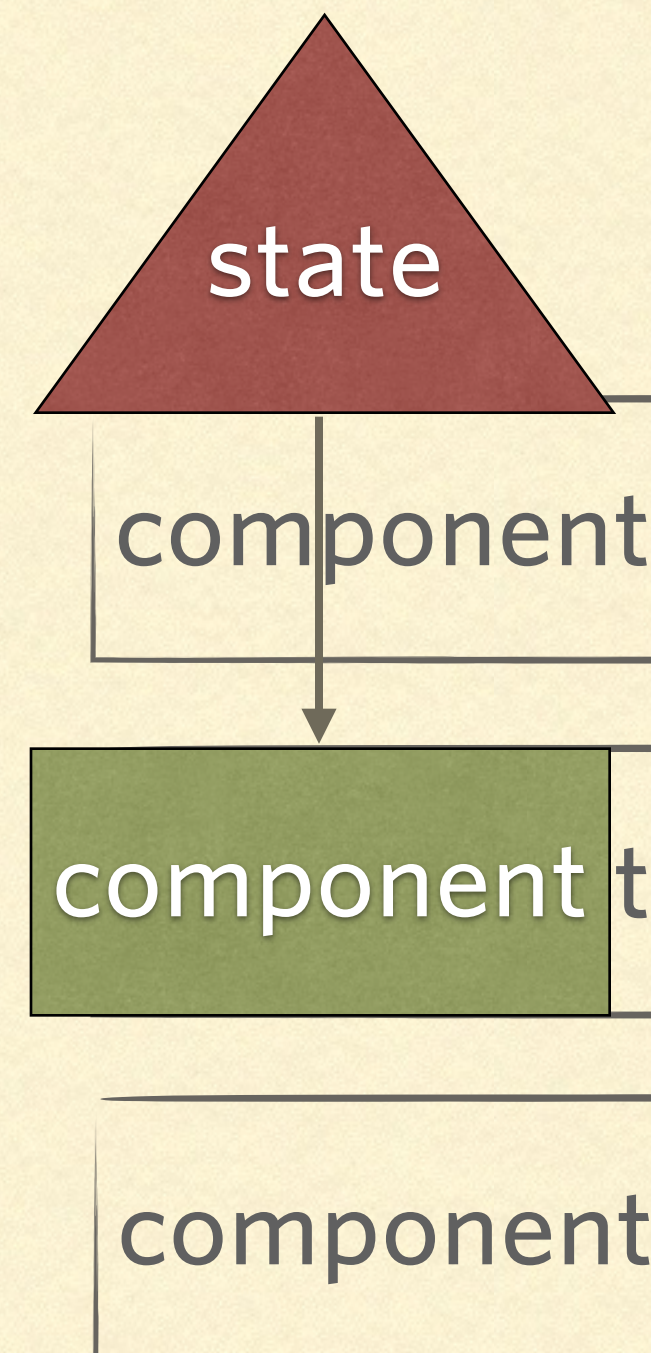
AN IMMUTABLE ARCHITECTURE



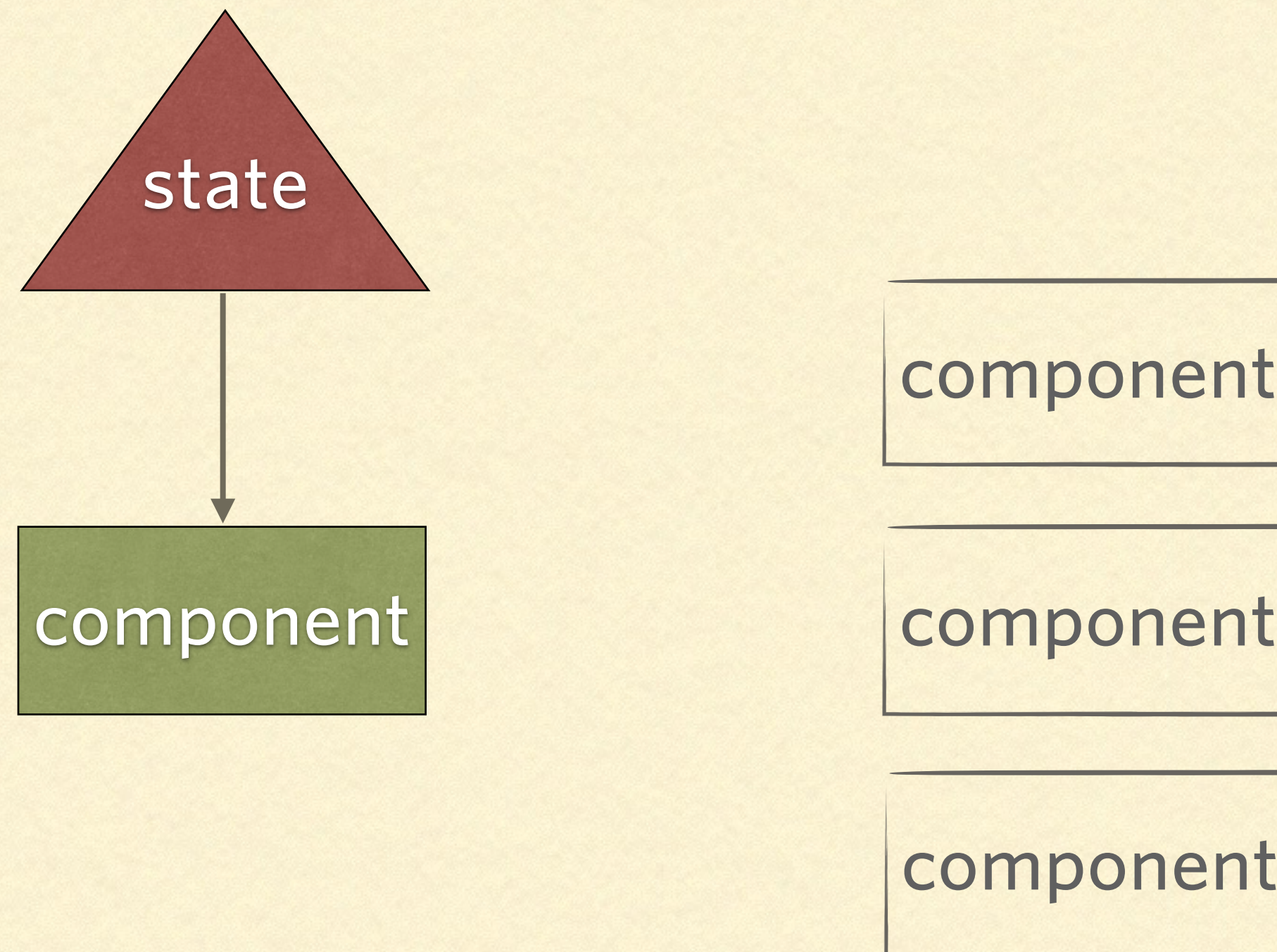
AN IMMUTABLE ARCHITECTURE



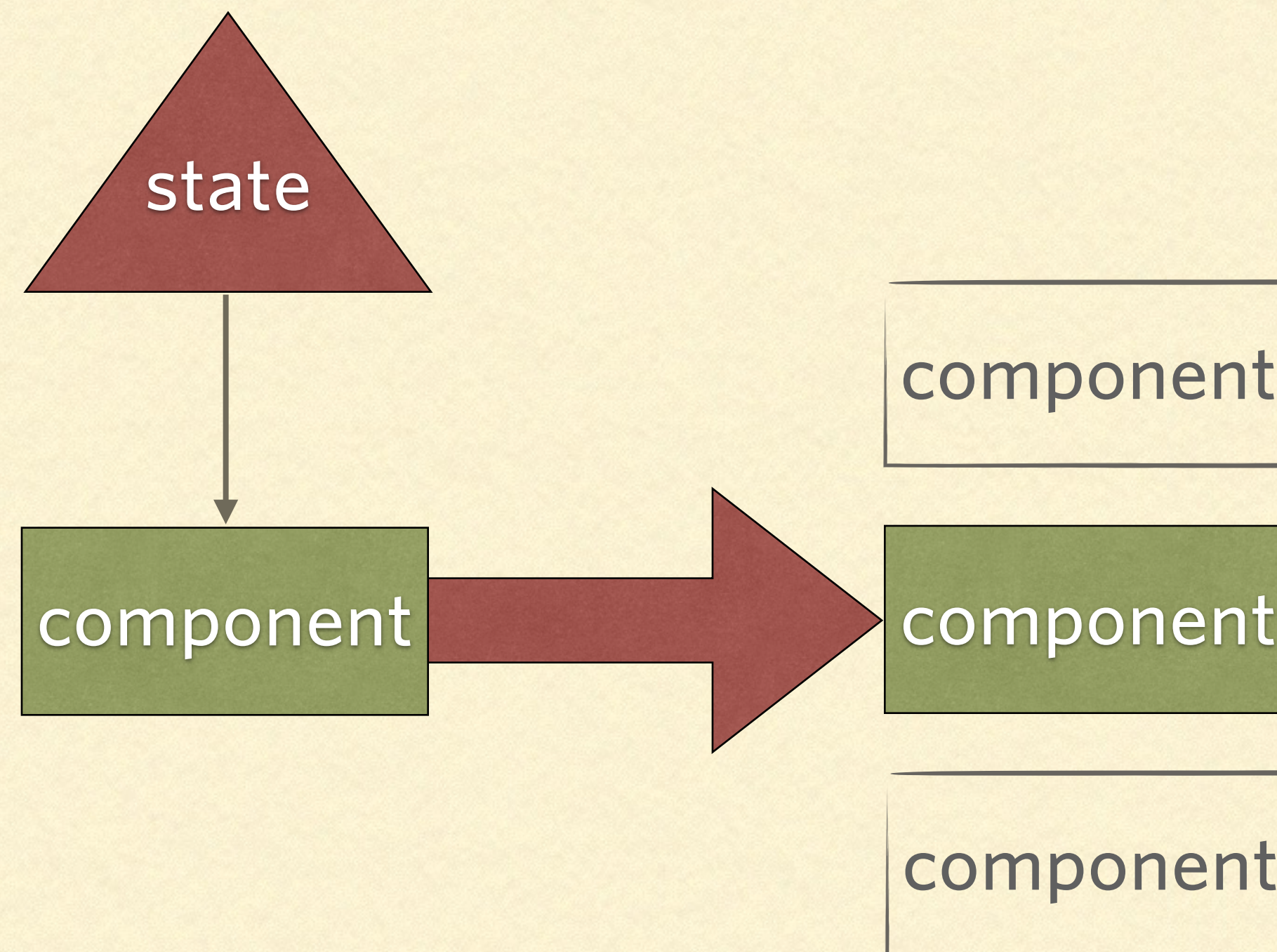
AN IMMUTABLE ARCHITECTURE



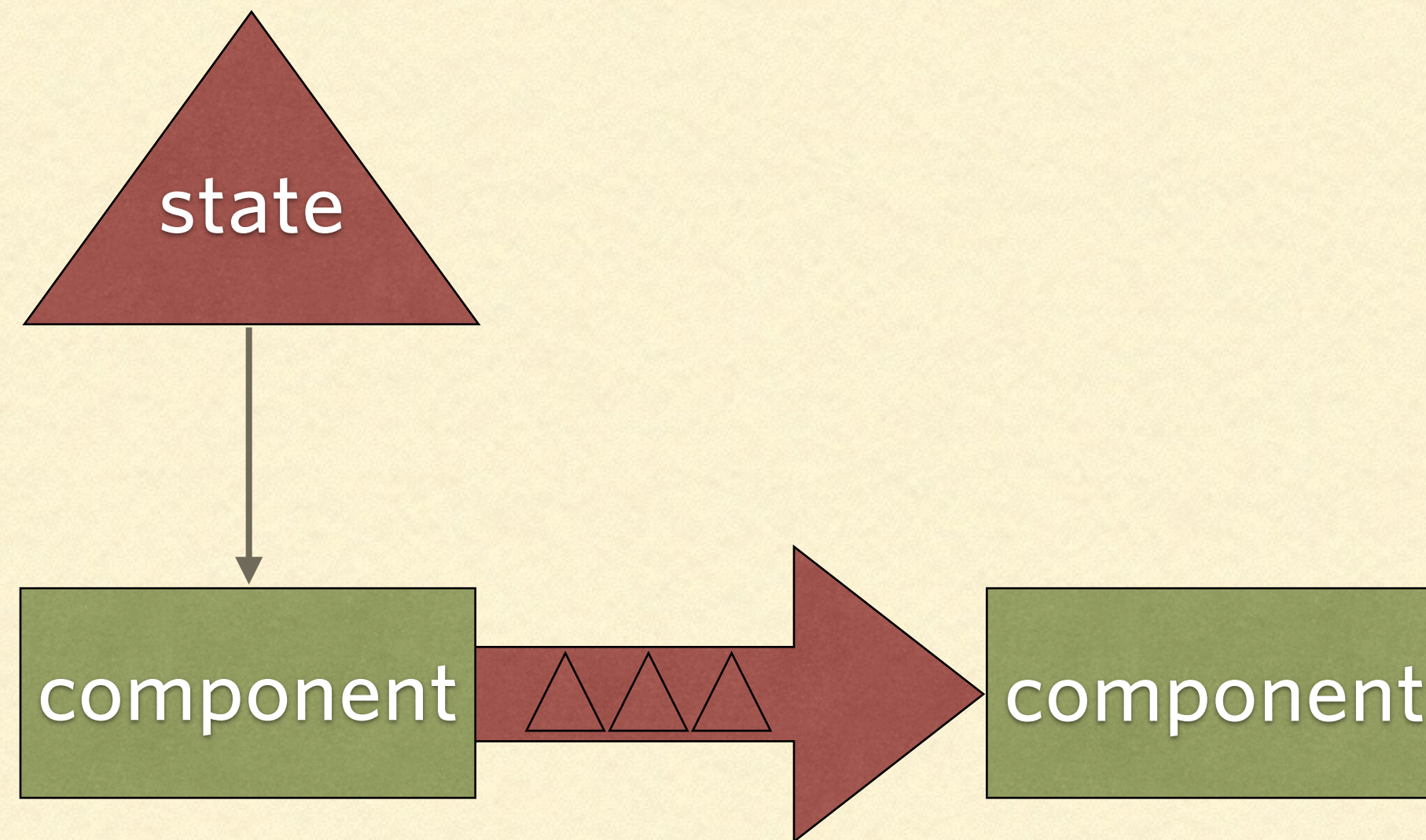
AN IMMUTABLE ARCHITECTURE



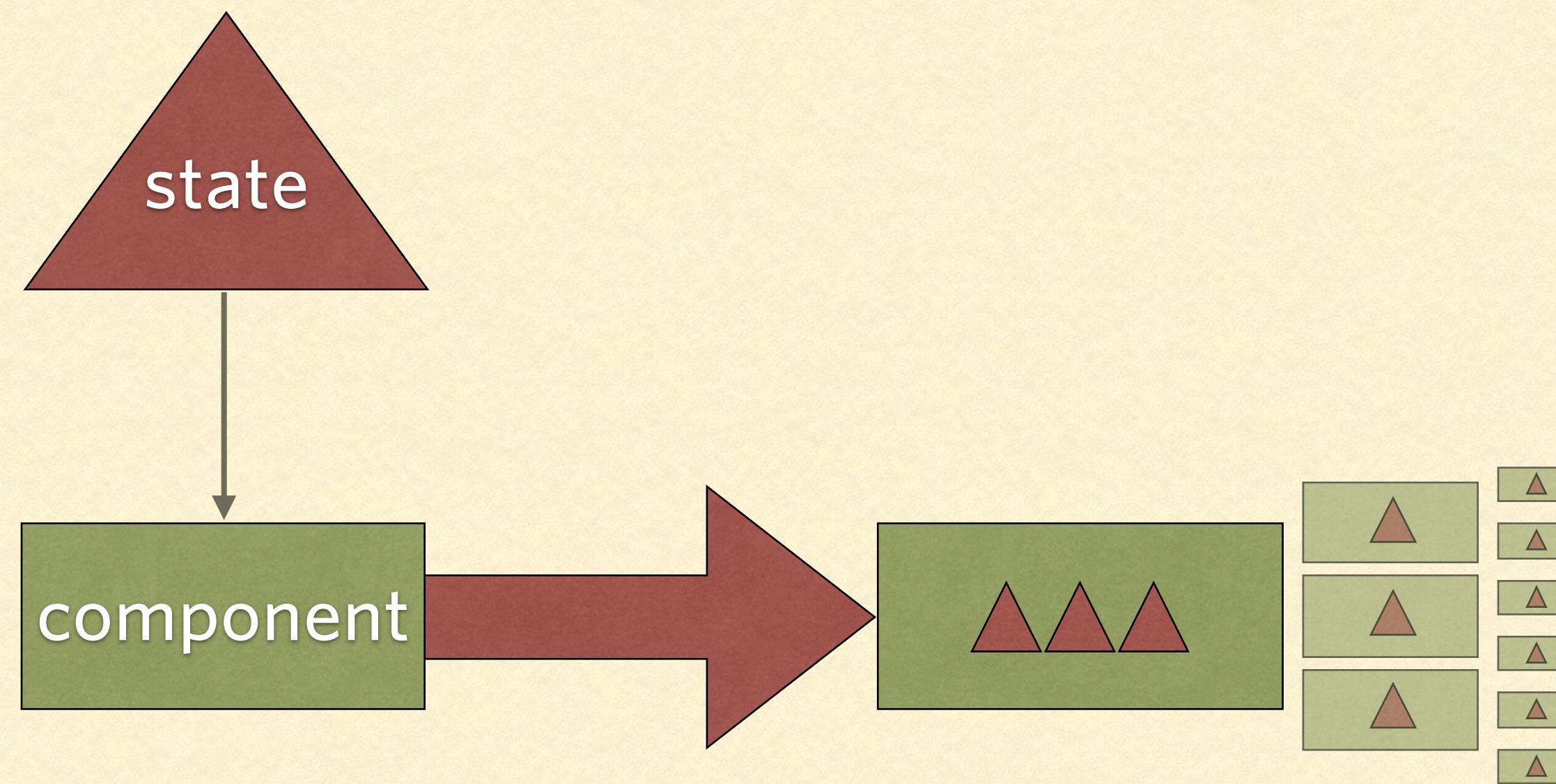
AN IMMUTABLE ARCHITECTURE



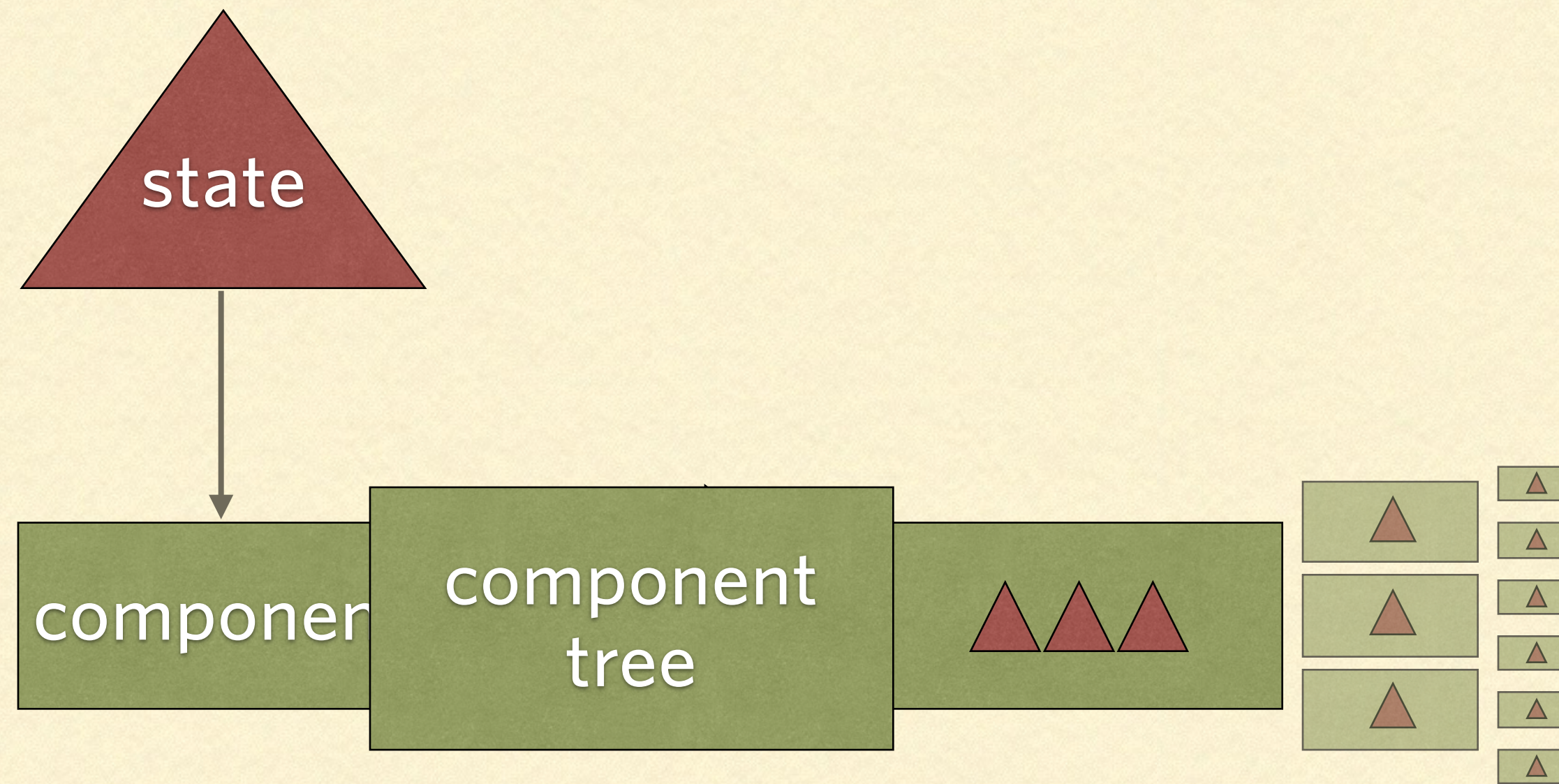
AN IMMUTABLE ARCHITECTURE



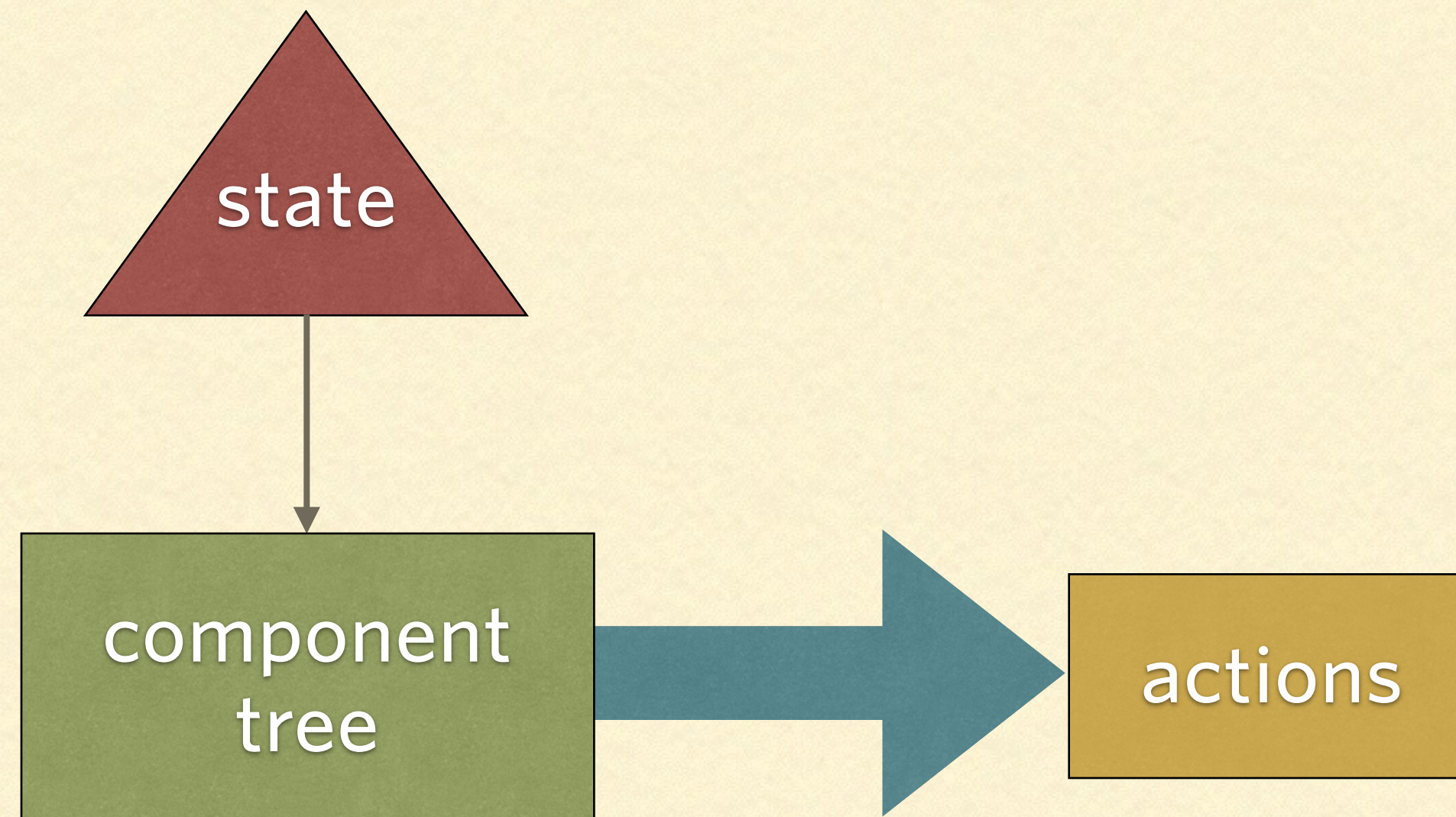
AN IMMUTABLE ARCHITECTURE



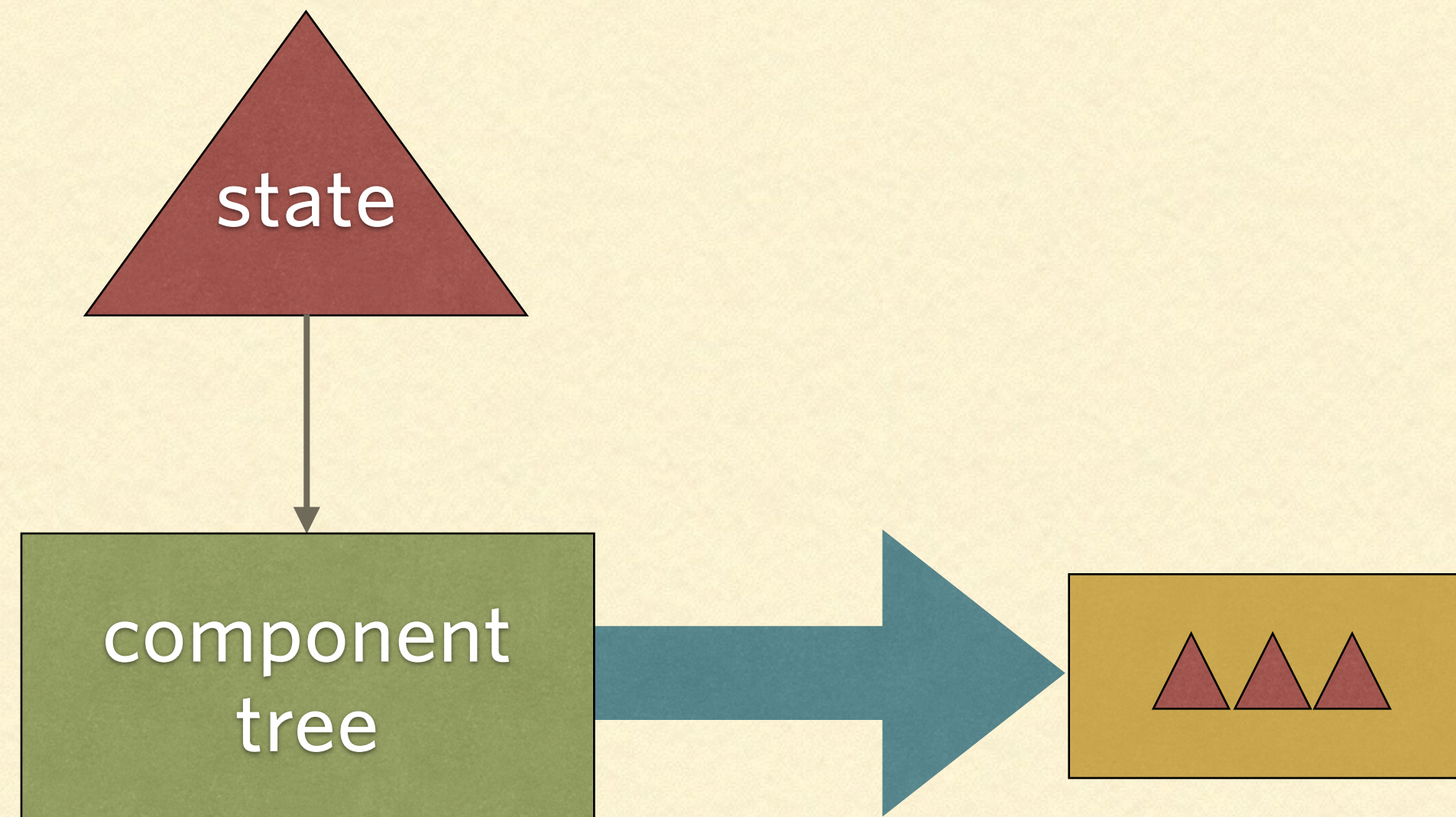
AN IMMUTABLE ARCHITECTURE



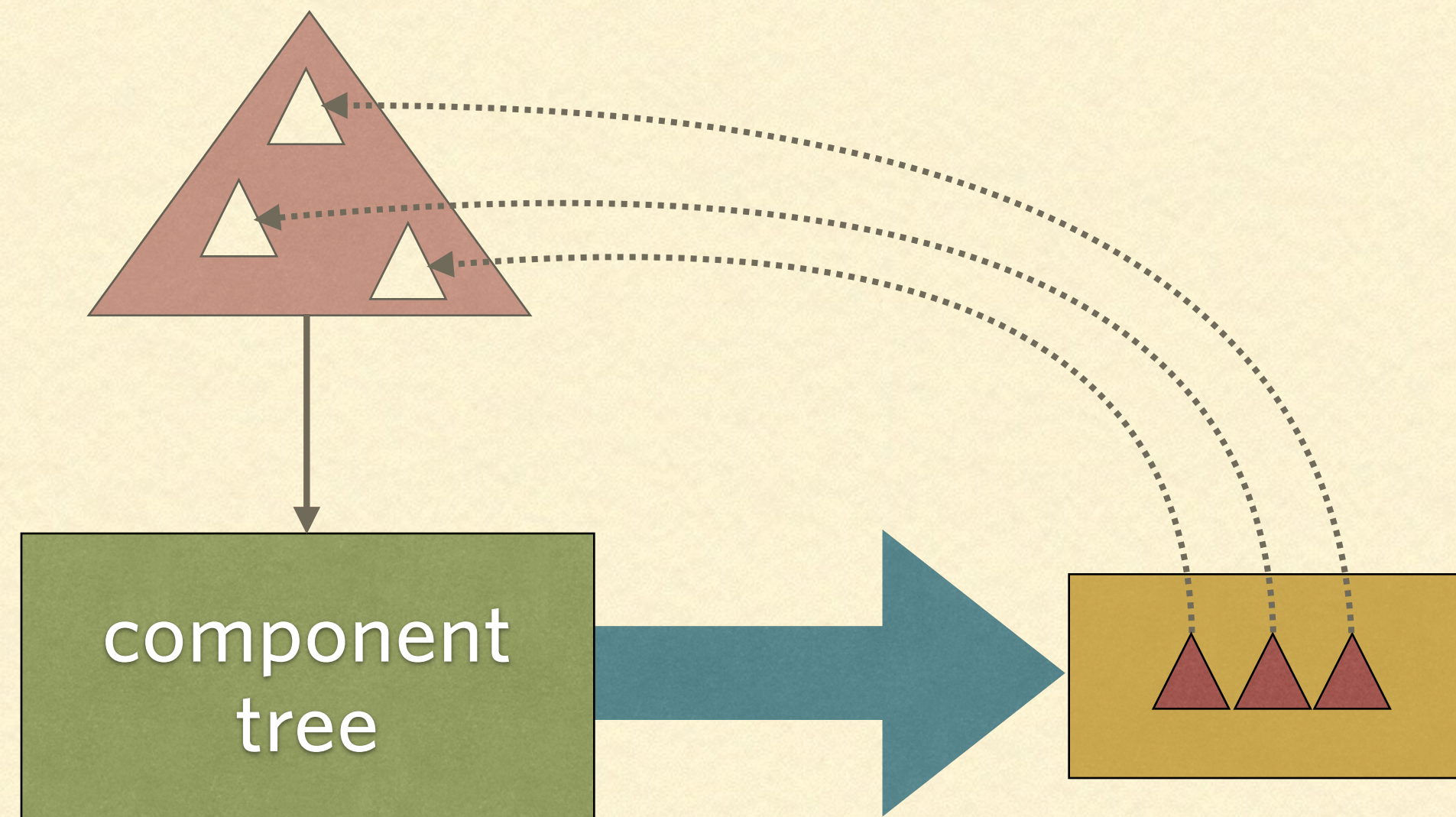
AN IMMUTABLE ARCHITECTURE



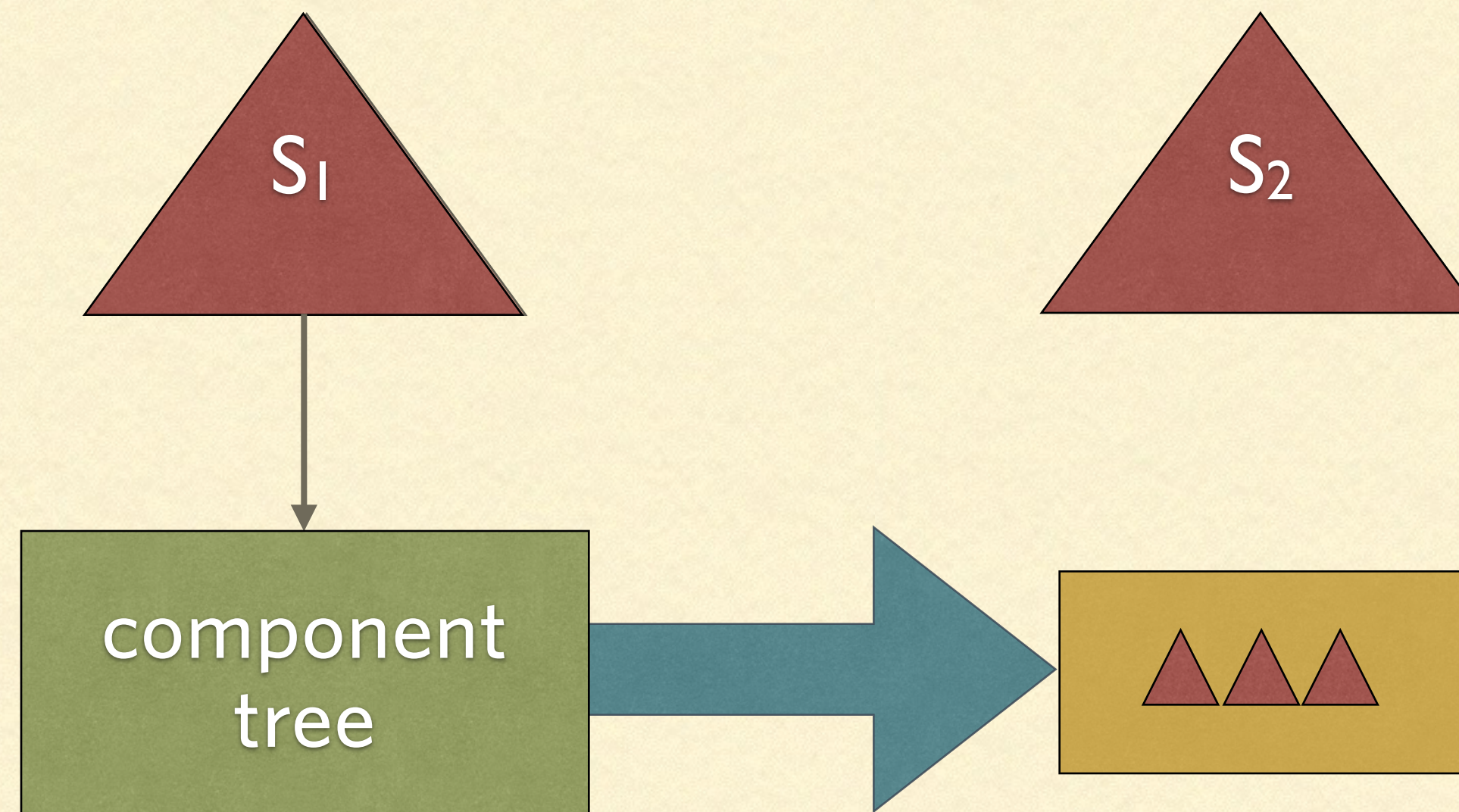
AN IMMUTABLE ARCHITECTURE



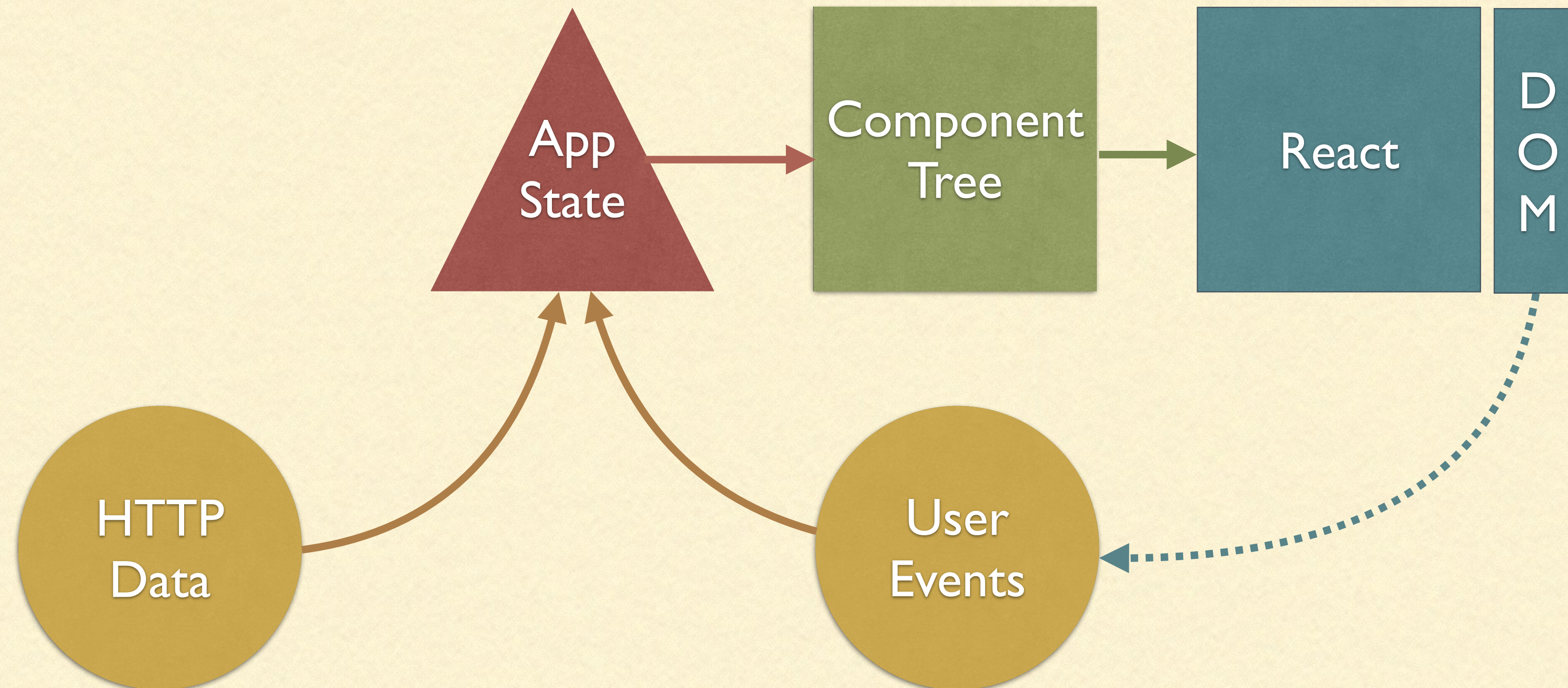
AN IMMUTABLE ARCHITECTURE



AN IMMUTABLE ARCHITECTURE



AN IMMUTABLE ARCHITECTURE



AN IMMUTABLE ARCHITECTURE

- Immutable application state
- Business logic written as pure functions
- Declarative rendering

APP STATE



App
State

- Reference to an immutable value
 - Updating state changes reference
- Business logic as pure functions
 - Compose multiple operations & apply at once

REACT

React

D
O
M

- UI rendering library from Facebook
- “Not templating. Not MVC. It’s like a *declarative* jQuery” -Pete Hunt
- Manipulates DOM & handles events (thats all)
- Trending, and rightfully so!

REACT

```
var HelloMessage = React.createClass({
  render: function() {
    return DOM.div({}, "Hello " + this.props.name);
  }
});

React.render(HelloMessage({name: "QCon"}), mountNode);
React.render(HelloMessage({name: "QCon SF"}), mountNode);
```


REACT

React

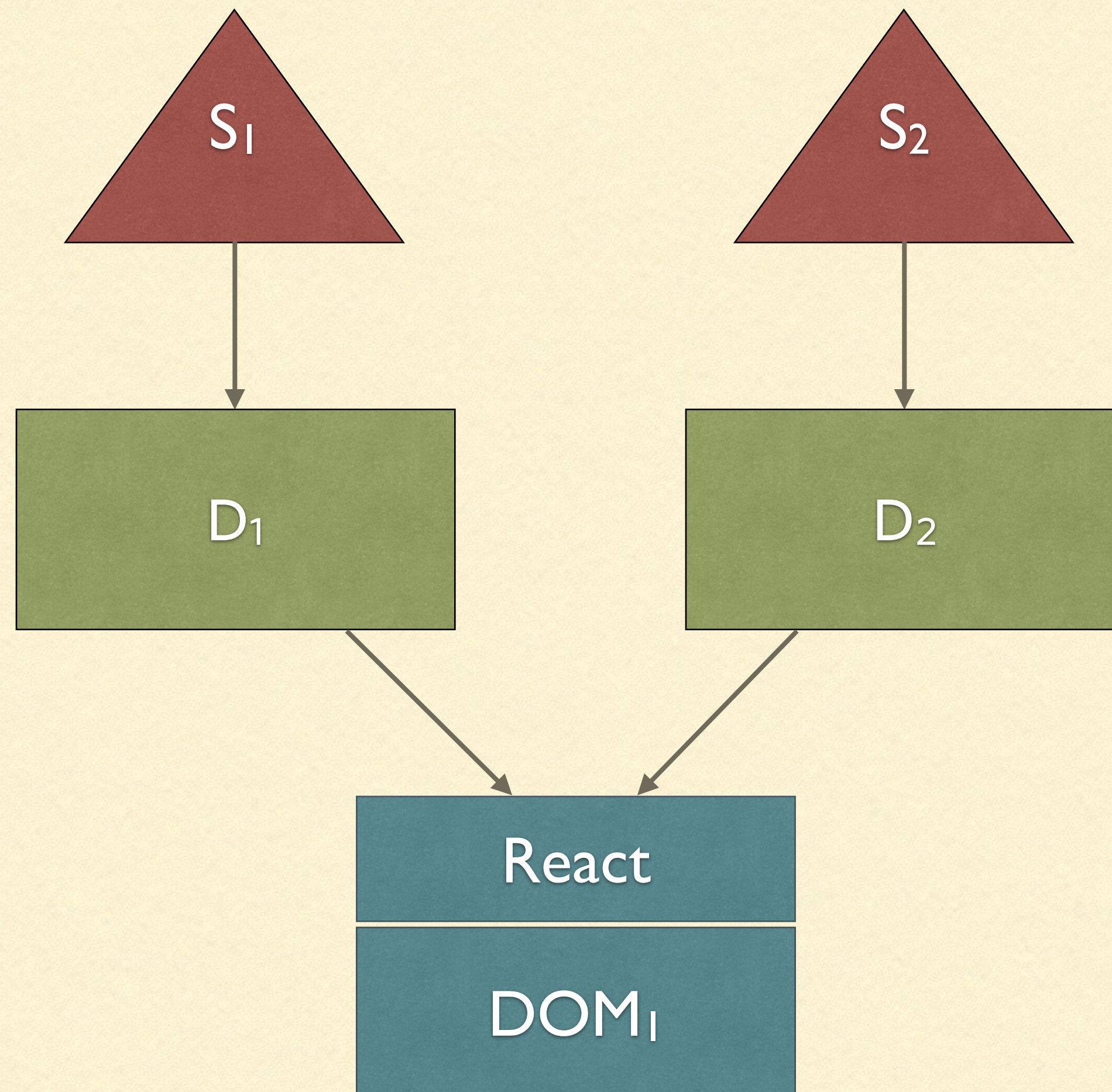
D
O
M

- Basic building block is a component with `render()` method
- Data in, “virtual DOM” out
- When data changes, `render()` is re-run
- Performs diff between vDOM and actual DOM
- Pushes out minimal set of changes to keep in sync

REACT & CLOJURESCRIPT

- Shared design principles
 - pure and composable functions
 - simplicity through predictability
- Actually useful API
 - Easy to compose from ClojureScript
 - Libraries like Om, Reagent, Quiescent

REACT & CLOJURESCRIPT



SIMPLE OM EXAMPLE

WRAP-UP

- Immutability & referential transparency have many benefits
 - Testing & Reasoning
 - Application architecture
- Invest in languages & tools that prioritize simplicity
 - Clojure & ClojureScript are great!
 - React is great!

THANKS!

@loganlinn
@prismatic

