REACT

Felix Stürmer

May 11, 2015

DEMARCATION

WHAT REACT *IS NOT*

React *is not*...

- an architecture (like MVC)
- a web framework (like Ember.js)
- a functional reactive programming (FRP) library (like RXJS or bacon.js)

React *is*...

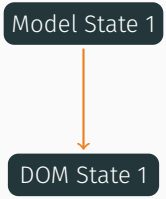
- a **view library**

that implements

- a **virtual DOM**
- a **DOM diffing** and **patching** algorithm

MOTIVATION

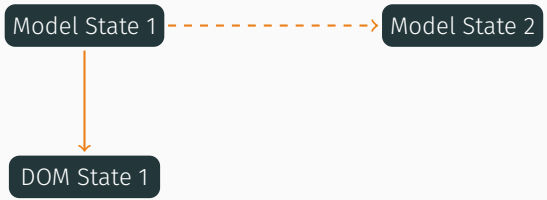
It's full of **state** and **transitions**



state propagation

explicit transition

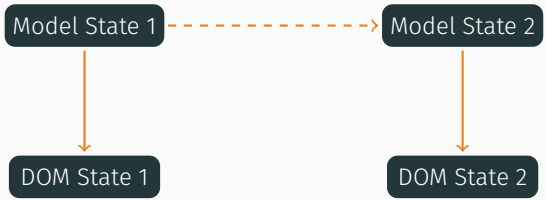
It's full of **state** and **transitions**



state propagation

explicit transition

It's full of **state** and **transitions**



state propagation

explicit transition

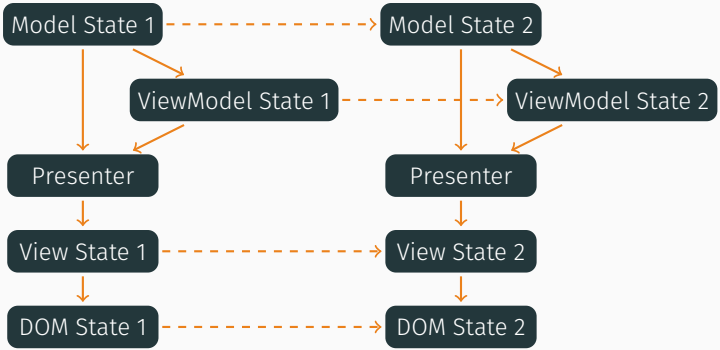
It's full of **state** and **transitions**

state propagation

explicit transition

WHAT MAKES WEB INTERFACES COMPLEX? (2)

Even more so in our case



state propagation

explicit transition

THE PROBLEM

- Complexity of **state** and **transitions**
- Tight **coupling** of
  - ViewModel
  - Presenter
  - View
  - Template
  - DOM
- Impedes
  - Reusability
  - Testability
  - Predictability

> *[...] we should do (as wise programmers aware of our limitations) our utmost to* ***shorten the conceptual gap between the static program and the dynamic process****, to make the correspondence between the program (spread out in text space) and the process (spread out in time) as trivial as possible.*

— E.W. Dijkstra (1968) “A Case against the GO TO Statement”

SIMPLICITY – PREDICTABILITY – ROBUSTNESS

- less context to keep in mind
- fewer mistakes
- fewer bugs
- easier to learn
- easier to adapt

How can React help?

Re-render everything in case of changes

- Minimize redundant and scattered state
- Minimize the number of transitions (managed by the developer)
- Minimize side-effects

State	Internal component state (*which should be minimal*)
Properties	Externally supplied state
The render() function	Turns state and properties into a visual representation

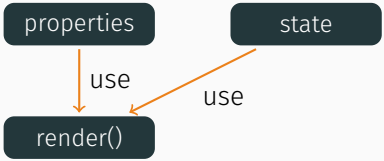
THE RENDER FUNCTION

- stateless
- free of side-effects
- referentially transparent

render()

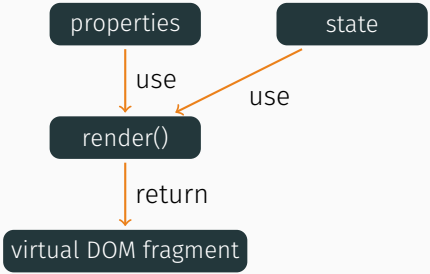
THE RENDER FUNCTION

- stateless
- free of side-effects
- referentially transparent



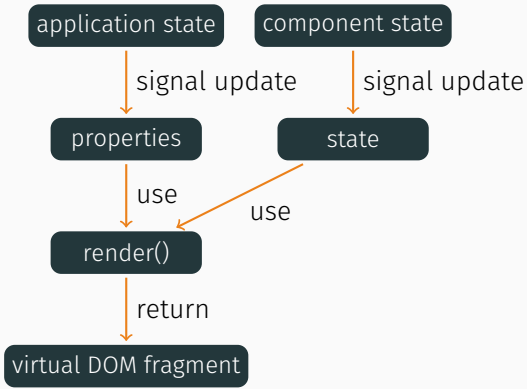
THE RENDER FUNCTION

- stateless
- free of side-effects
- referentially transparent



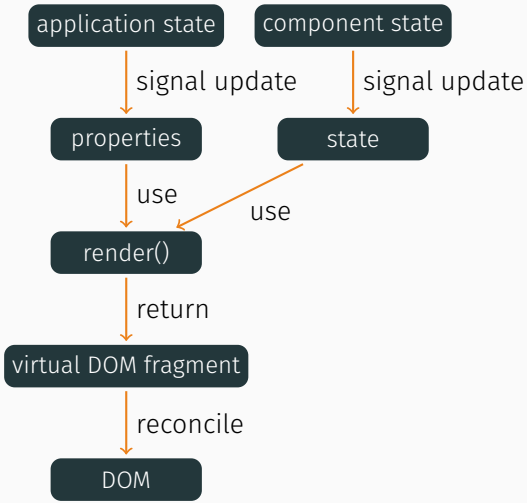
THE RENDER FUNCTION

- stateless
- free of side-effects
- referentially transparent



THE RENDER FUNCTION

- stateless
- free of side-effects
- referentially transparent



DOM diffing and patching rules

- for element lists:
  - pairwise by key if given
  - pairwise by position otherwise
- pairwise comparison
  - different tag type treated as replacements
  - same tag type update attribute by attribute

POSITIVE EFFECTS

Composable components can be nested

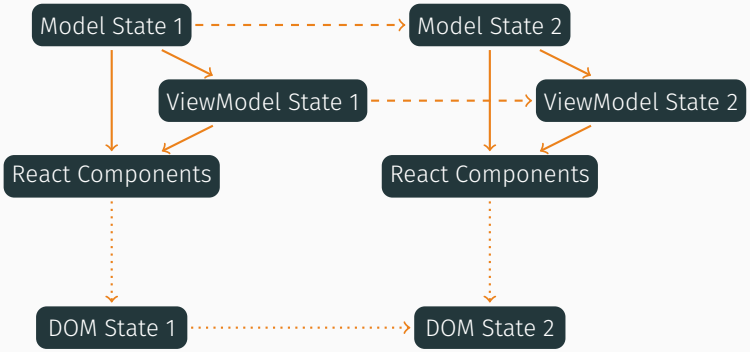
Cachable return values can be cached safely

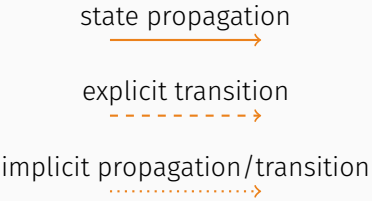
Predictable no hidden state

Flexible virtual DOM can be rendered to DOM, string, native widgets, etc...

POSITIVE EFFECTS (2)

- Simpler data flow
- Just update component properties on model change





CODE EXAMPLES

A SIMPLE COMPONENT

```
1  React = require 'react'
2  {div} = React.DOM
3
4  class LineNumberColumn extends React.Component
5      @propTypes:
6          value: React.PropTypes.number.isRequired
7
8      render: ->
9          div className: 'lineNumber', @props.value
10
11 module.exports = {
12     LineNumberColumn
13 }
```

RENDERING COMPONENTS

```
1  React = require 'react'
2  {LineNumberColumn} = require './columns'
3
4  targetElement = document.querySelector '#demoTarget'
5
6  viewModel.on 'change', ->
7      virtualElement = React.createElement(LineNumberColumn, value: viewModel.lineNumber)
8      React.render virtualElement, targetElement
```

COMPOSING COMPONENTS

```coffee
{DOM: {div}} = React = require 'react'

{LineNumberColumn, ExpandColumn, NameColumn} = require './columns'

class TreeRowContent extends React.Component
    @propTypes:
        planningObject: React.PropTypes.object.isRequired

    render: ->
        {planningObject} = @props
        div className: 'content',
            div {},
                React.createElement LineNumberColumn, value: planningObject.lineNumber
                div className: 'rowContent',
                    React.createElement ExpandColumn {}
                    React.createElement NameColumn value: planningObject.name
                    # ...
```

REACTING TO USER ACTIONS

```coffee
React = require 'react'
{button} = React.DOM

class CounterButton extends React.Component
    constructor: (props) ->
        @state =
            counter: 0

    increment: =>
        @setState
            counter: @state.counter + 1

    render: ->
        button onClick: @increment, "Counter: #{@state.counter}"
```

```
{DOM: {li, ul}} = React = require 'react'
classNames = require 'classnames'

class TreeRow extends React.Component
    @propTypes:
        planningObject: React.PropTypes.object.isRequired
        level: React.PropTypes.number
    @defaultProps:
        level: 0
    render: ->
        {planningObject, level} = @props
        classes = classNames "level-#{level}",
            hasChildren: planningObject.children.length > 0

        li className: classes,
            React.createElement TreeRowContent, planningObject: planningObject
            ul className: 'children', for child in planningObject.children
                React.createElement TreeRow, planningObject: child, level: @level + 1
```

PERFORMANCE

Performance is not the main focus, but a bonus

- The virtual DOM is fast
- The real (slow) DOM is only modified selectively
- Plain JavaScript automatically benefits from future runtime optimization
- Can be rendered server-side to reduce initial load time

`shouldComponentUpdate()`

- Called when state or properties might have changed
- Component decides whether `render()` will be called by comparing old and new state
- The default implementation always returns `true`
- If it returns `false`, the previous virtual DOM is reused

```
1  class LineNumberColumn extends React.Component
2      shouldComponentUpdate: (newProps, newState) ->
3          newProps.value isnt @props.value
4
5      render: ->
6          div className: 'lineNumber', @props.value
```

Immutable State

- Pass state and properties as **immutable data structures**, e.g. Facebook's Immutable.js
- $O(1)$ equality checks of arbitrarily complex objects
- Easier reasoning about state changes

```
1  class TreeRowContent extends React.Component
2      shouldComponentUpdate: (newProps, newState) ->
3          not Immutable.is newProps.planningObject, @props.planningObject
4
5      render: ->
6          # ...
```

QUESTIONS, PLEASE – THANK YOU