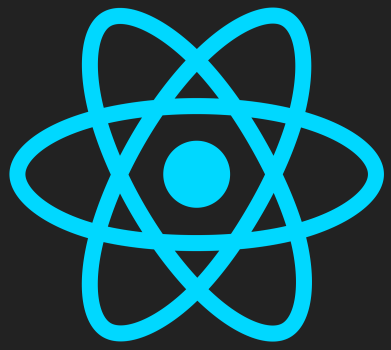


React.js Basics

@RobertWPearce

robertwpearce.com/converge-reactjs
robertwpearce.com/converge-reactjs-demo

whoami

- From Charleston, SC
- Digital Nomad
- Work at 100% remote company, Articulate (articulate.com)
- Full-stack dev
- Worked with React for almost a year

Why You Should Listen

I've made more small mistakes than you.

Talk Notes

Since requiring/exporting modules and dependencies takes up too much space, I've omitted them from the presentation.

However, they naturally are included in the demo code.

Talk Notes

When you see

```
{div, ul, li, span} = React.DOM
```

don't freak out. This is just a CoffeeScript way of assigning each of these to variables which represent, for example

```
React.DOM.div()
React.DOM.ul()
React.DOM.li()
React.DOM.span()
# etc
```

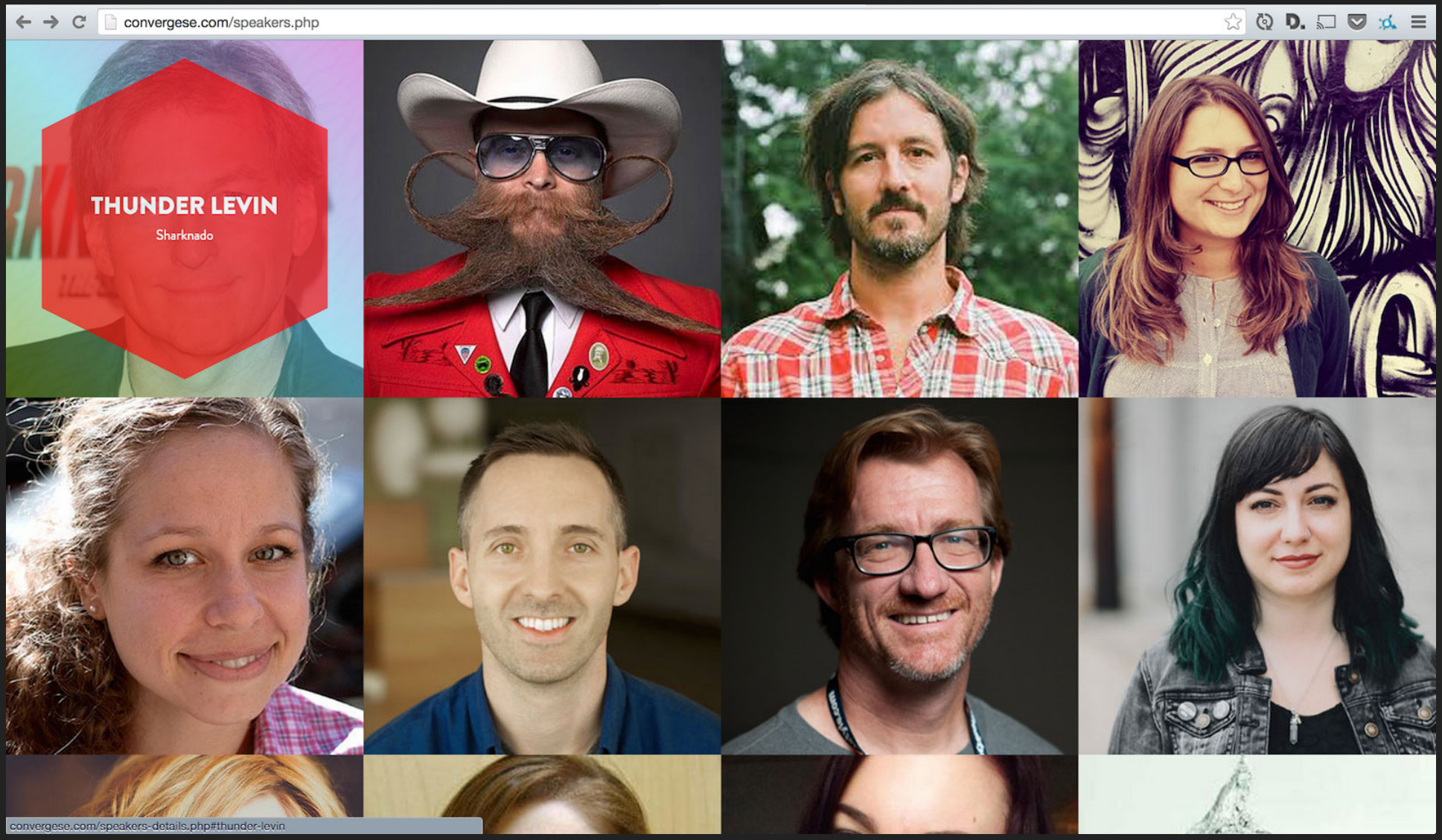
This lets you use div, ul, li and span without prefixing them with React.DOM.___

Talk Notes

There will not be Q&A at the end,
so ask questions as we go.

The Problem

Can't see ConvergeSE's speaker details without navigating to another page


convergese.com/speakers.php
THUNDER LEVIN
Sharknado
convergese.com/speakers-details.php#thunder-levin

convergese.com/speakers-details.php#thunder-levin

THUNDER LEVIN

Sharknado

@ThunderLevin

Thunder Levin is a feature film and television director and writer.

Best known for writing the insanity that is the “Sharknado” franchise for the Syfy Channel, he has also written and directed three films: “Mutant Vampire Zombies From The ‘Hood!”, an independent horror/comedy starring C. Thomas Howell; and more recently “American Warships” a military action film shot in Wilmington, NC, starring Mario Van Peebles & Carl Weathers which premiered on the Syfy Channel in May 2012; and “AE Apocalypse Earth”, a science fiction film starring Adrian Paul & Richard Grieco.

He also wrote the screenplay for the street racing action film “200 MPH” for the

DR. PAUL ROOF

Holy City Beard & Moustache Society

Paul Roof is an Associate Professor of Sociology at Charleston Southern University and specializes in urban studies and popular culture.

In 2007, while teaching at the College of Charleston, Paul founded the Holy City Beard & Moustache Society. This American Beardsman’s recent beard contest victories include 5th Place in the World Freestyle Beard 2011, 1st Place Freestyle Beard in the Miami Beard & Moustache Championships 2011, 3rd Place Full Beard Natural in the North Carolina Beard & Moustache Championships 2012, and 3rd Place Freestyle Beard in the East Coast Beard & Moustache Championships held in Philadelphia 2012, and 2nd Place Freestyle Beard in the National Beard & Moustache Championships held in New Orleans in 2013. His beard has been featured on Buzzfeed, GQ Magazine, UK Telegraph, Reader’s Digest and Ripley’s Believe It or Not!.

What We Want

Robert

Search Library

My Music | Playlists | Radio | iTunes Store | Albums

Still the King: Celebrat...
Asleep At the Wheel

Between the Dim & th...
Jump Little Children

Goodnight - EP
Chris Holly

Porcelain Empire
The Winter Sounds

River Songs
The Badlees

Vertigo
Jump Little Children

All Albums

Unknown Album
Allman Brothers Band

The Arcade Fire
Arcade Fire

Funeral
Arcade Fire

Her

Still the King: Celebrat...
Asleep At the Wheel

Audioslave
Audioslave

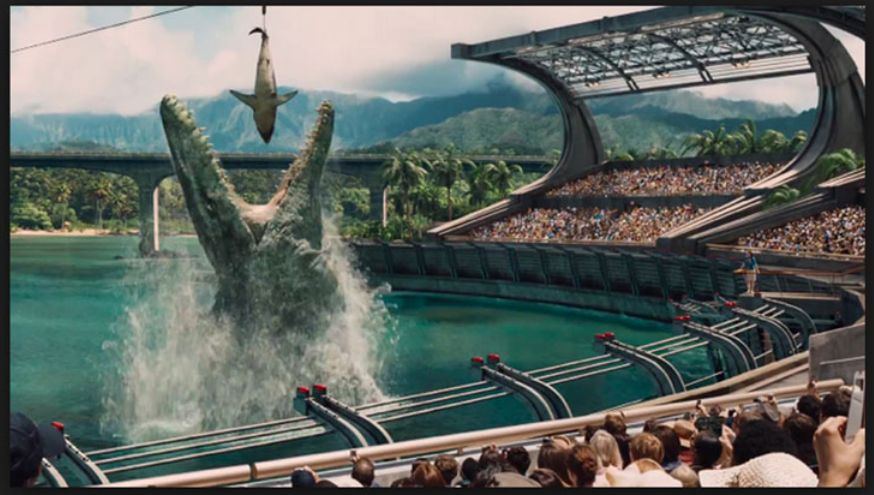
Her

Arcade Fire • 2013

More from the Store

1	Sleepwalker	3:16	8	Loneliness #4 (Other People's Letters)	1:02
2	Milk & Honey	1:29	9	Owl	2:24
3	Loneliness #3 (Night Talking)	3:27	10	Photograph	2:29
4	Divorce Papers	3:17	11	Milk & Honey (Alan Watts & 641)	3:19
5	Morning Talk/Supersymmetry	4:16	12	We're All Leaving	2:32
6	Some Other Place	3:39	13	Dimensions	5:42
7	Song On the Beach	3:33			

What We Want



Jurassic World' Trailer Debuts: Chris Pr…

www.etonline.com - 640 × 360 - Search by image

Control on Isa Nublar has been restored and the resort has been expanded to feature a gyrosphere, a Sea World-inspired water show, and kayaking alongside ...

Visit page | View image

Related images:

Images may be subject to copyright. - Send feedback

vanilla JS or jQuery

We *could* use data selector finagling to hide/show information and divs accordingly, but this can get very ugly and difficult to maintain.

vanilla JS or jQuery

For example, we could

- Save each bit of data in data-* attributes for each item
- onClick -> Calculate the offset top and height for an item
- Give it sufficient padding underneath
- Apply the offset top + height to an absolutely positioned div (100% width) containing the info
- Hope things don't explode
- Discover that our maths suck when there's already an info section on the page
- Start maintaining state somewhere

vanilla JS or jQuery

or, we could

- Determine an item's position on a "row"
- onClick -> Insert a DOM element at the end of said row
- Populate this DOM element with data obtained from data-* attributes (or a lookup in a variable via data-id?)
- Maintain state of what element is currently clicked
- Facepalm when we have to change the # of items per row

"Verba movent, exempla trahunt."

(Words move people, examples compel them.)

— Latin Proverb

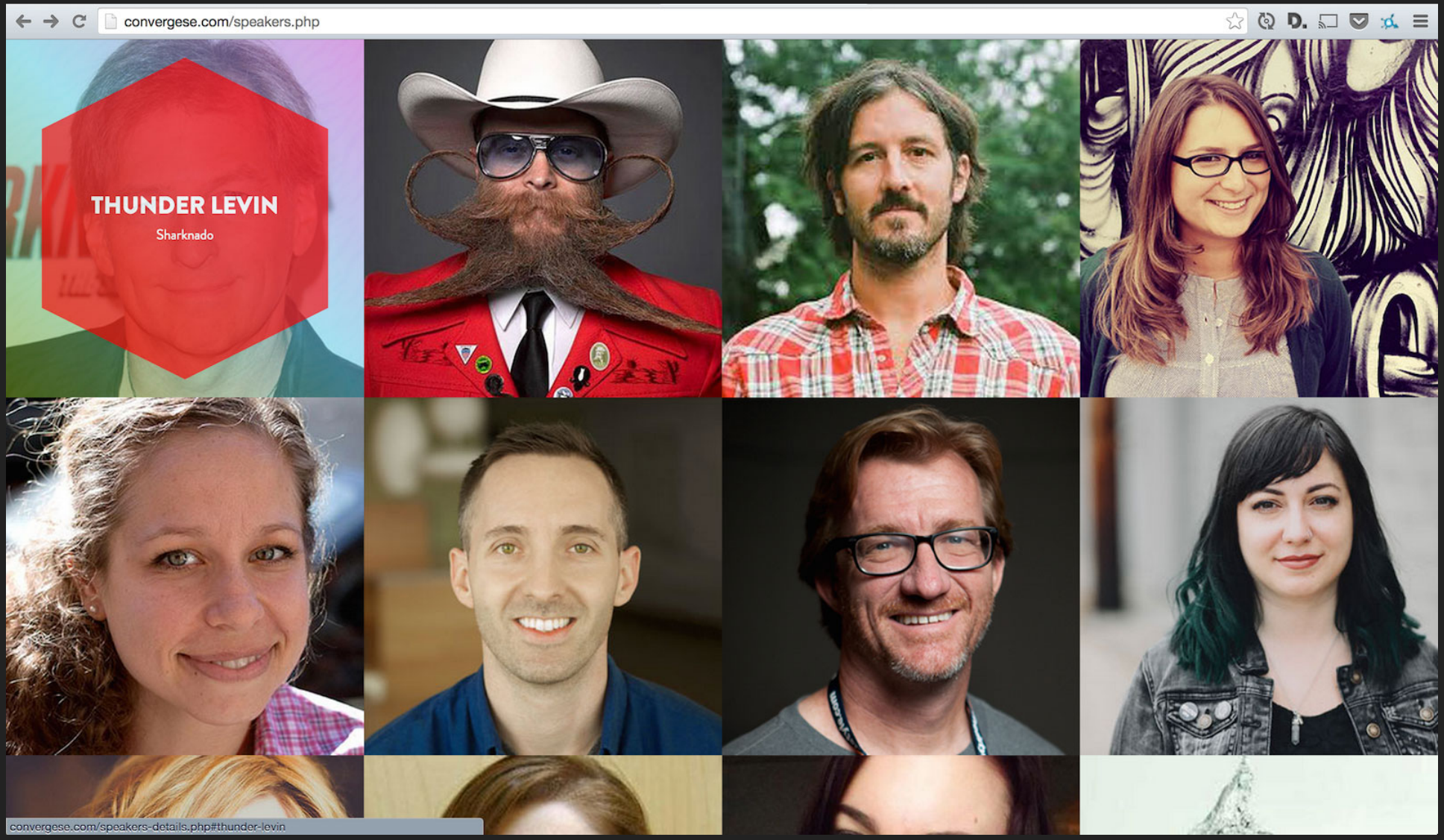
First the How

Then the Why

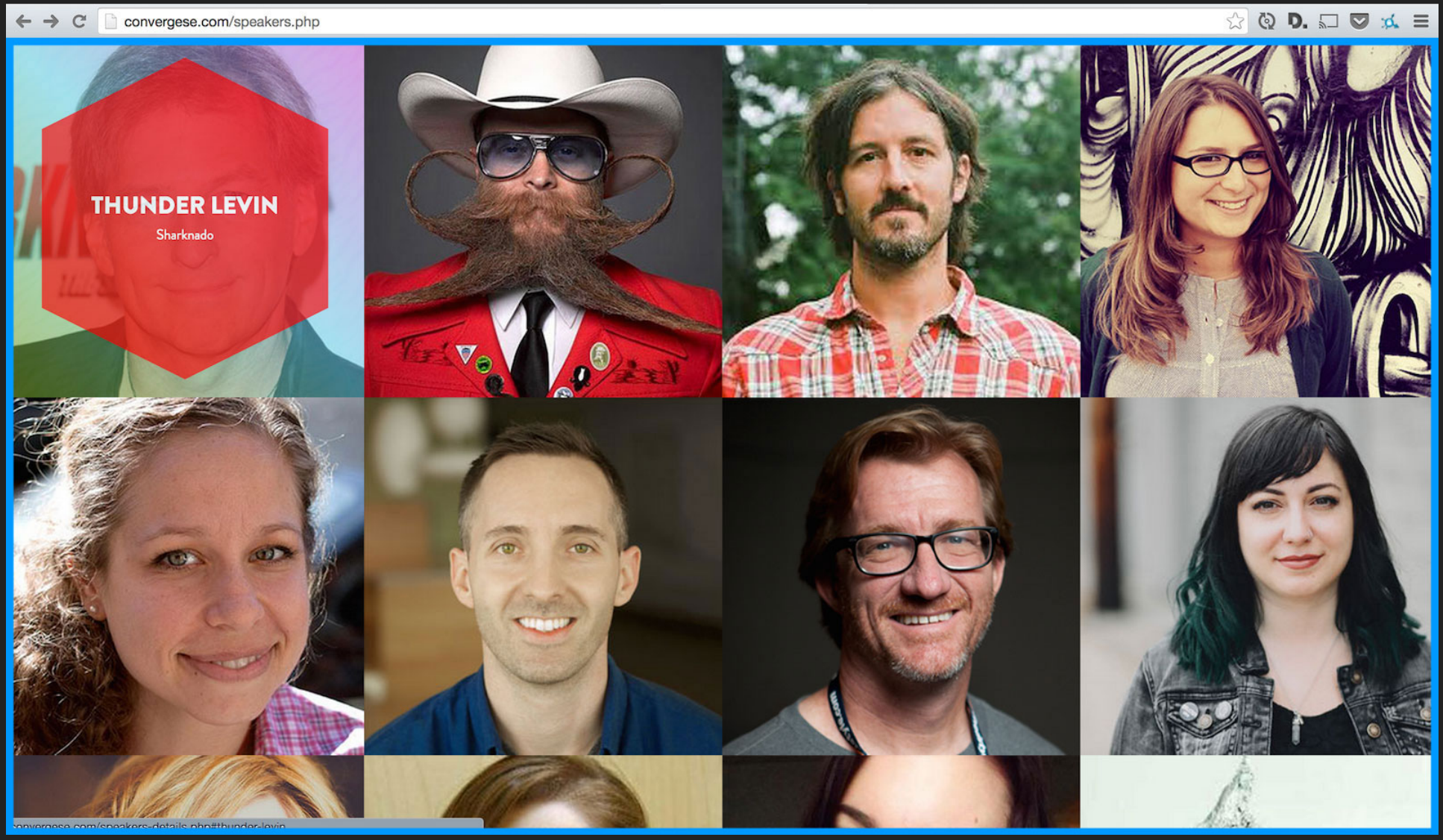
Then the Future

How

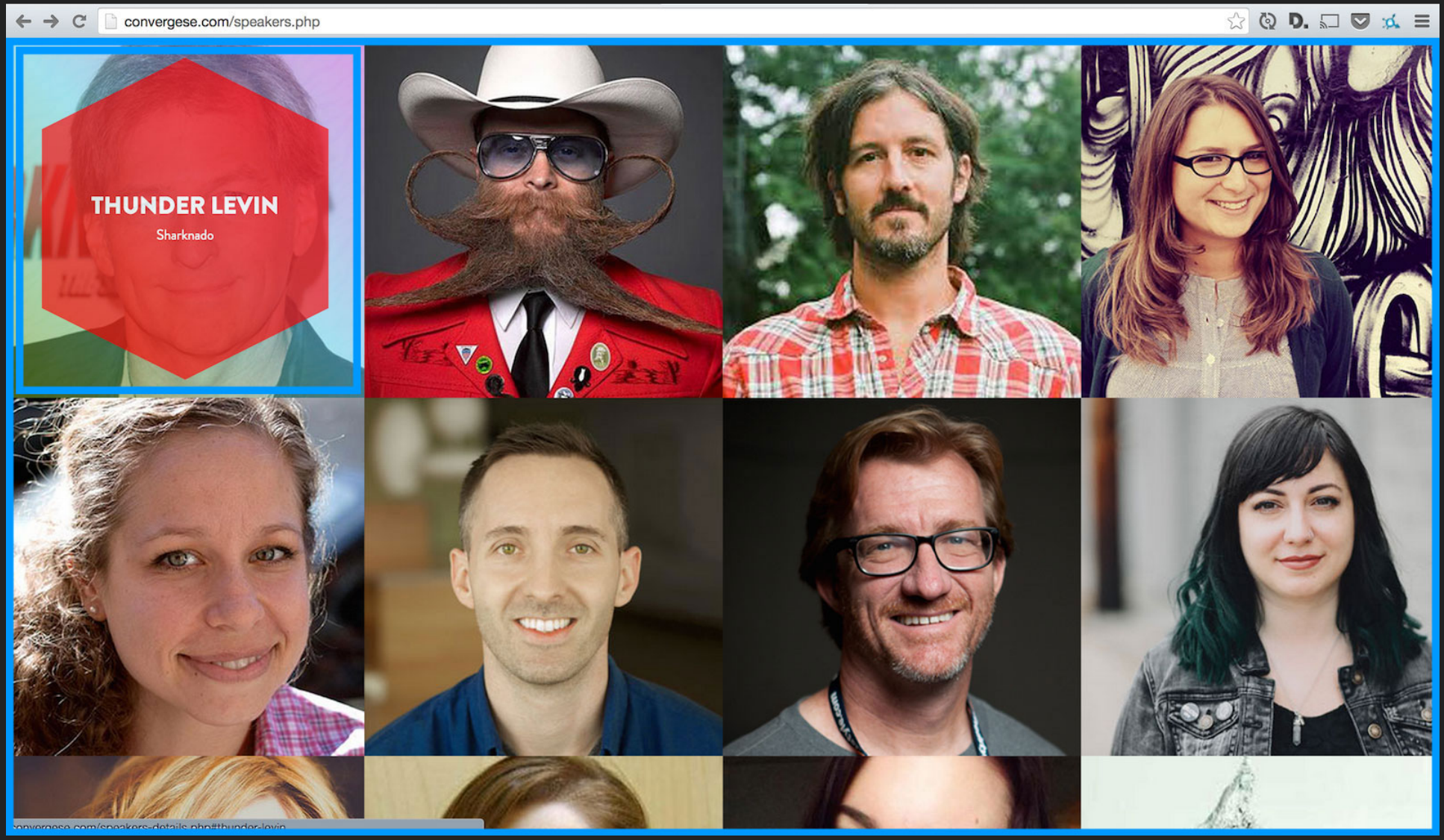
Thinking in Components



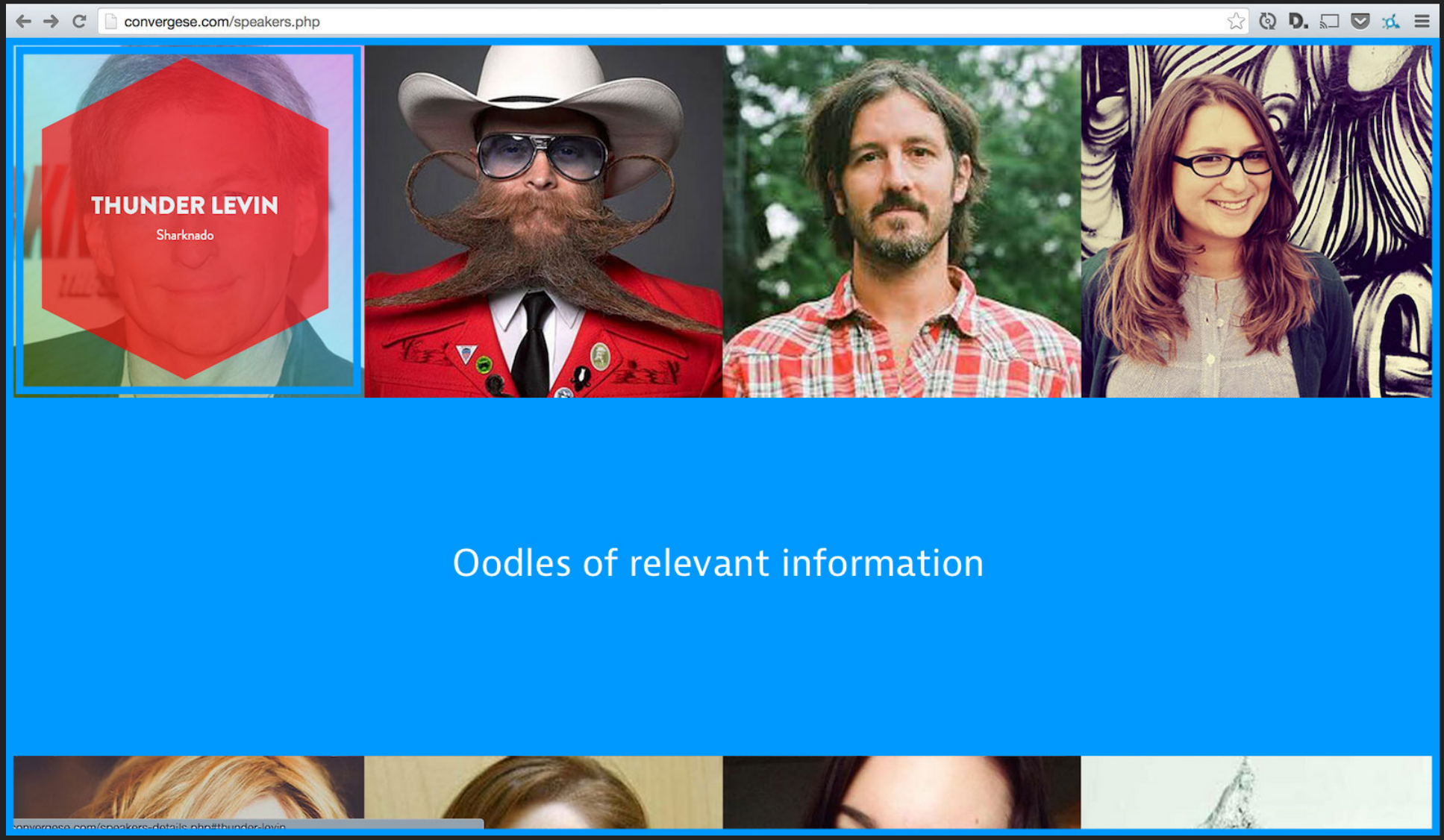
Thinking in Components



Thinking in Components



Thinking in Components



Speakers

- Rows of Speakers (grouped every 4)
- Info (conditionally added to a given Row)

Component Structure

```
Speakers
  Row
    Speaker
    Speaker
    Speaker
    Speaker
  Row
    Speaker
    Speaker
    Speaker
    Speaker
```

Component Structure (selected)

```
Speakers
  Row
    Speaker
    Speaker
    Speaker
    Speaker
  Row
    Speaker
    Speaker (selected)
    Speaker
    Speaker
    Info
```

What do we do first?

- Group speakers data in to rows of 4
- Map over each row item
- Map over each speaker within a row

Proof of Concept (try #1)

```
# CoffeeScript
speakersData = [{ id: 1, name: 'Emily' }, { id: 2, name: 'Lucy' }]

Speakers = React.createClass
```

Proof of Concept (try #1)

```
speakersData = [{ id: 1, name: 'Emily' }, { id: 2, name: 'Lucy' }]

Speakers = React.createClass
  render: ->
```

Proof of Concept (try #1)

```
speakersData = [{ id: 1, name: 'Emily' }, { id: 2, name: 'Lucy' }]

Speakers = React.createClass
  render: ->
    rows = _.chunk(speakersData, 4)
```

Proof of Concept (try #1)

```
speakersData = [{ id: 1, name: 'Emily' }, { id: 2, name: 'Lucy' }]

Speakers = React.createClass
  render: ->
    rows = _.chunk(speakersData, 4)
    div className: 'speakers',
```

Proof of Concept (try #1)

```
speakersData = [{ id: 1, name: 'Emily' }, { id: 2, name: 'Lucy' }]

Speakers = React.createClass
  render: ->
    rows = _.chunk(speakersData, 4)
    div className: 'speakers',
      rows.map (row) ->
```

Proof of Concept (try #1)

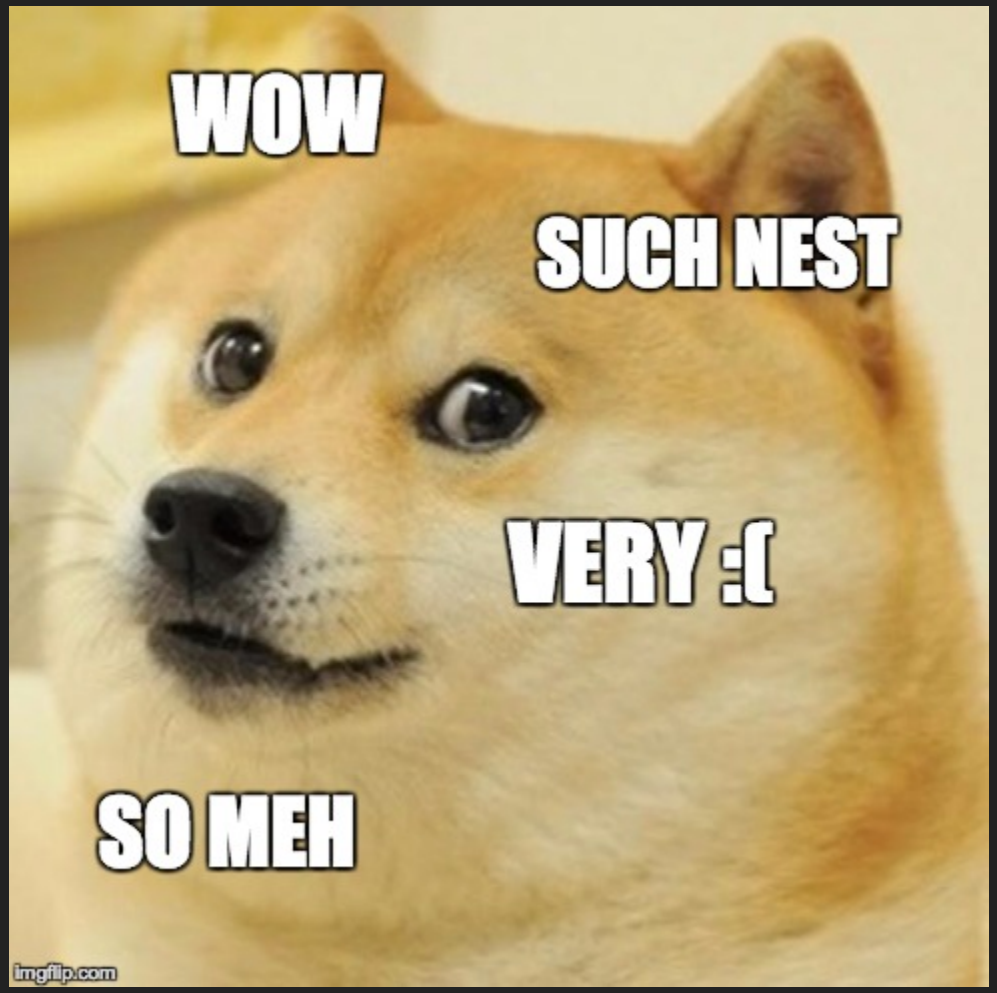
```
speakersData = [{ id: 1, name: 'Emily' }, { id: 2, name: 'Lucy' }]

Speakers = React.createClass
  render: ->
    rows = _.chunk(speakersData, 4)
    div className: 'speakers',
      rows.map (row) ->
        row.map (speaker) ->
```

Proof of Concept (try #1)

```
speakersData = [{ id: 1, name: 'Emily' }, { id: 2, name: 'Lucy' }]

Speakers = React.createClass
  render: ->
    rows = _.chunk(speakersData, 4)
    div className: 'speakers',
      rows.map (row) ->
        row.map (speaker) ->
          div className: 'speaker', speaker.name
```


WOW
SUCH NEST
VERY :(
SO MEH
imgflip.com

Proof of Concept (try #2)

```
speakersData = [{ id: 1, name: 'Emily' }, { id: 2, name: 'Lucy' }]

Speakers = React.createClass
  render: ->
    rows = _.chunk(speakersData, 4)
    div className: 'speakers',
      rows.map (row) ->
        row.map (speaker) ->
          div className: 'speaker', speaker.name
```

Proof of Concept (try #2)

```
speakersData = [{ id: 1, name: 'Emily' }, { id: 2, name: 'Lucy' }]

Speakers = React.createClass
  render: ->
    rows = _.chunk(speakersData, 4)
    div className: 'speakers',
      @_buildRows()

  _buildRows: ->
    rows.map (row) ->
      row.map (speaker) ->
        div className: 'speaker', speaker.name
```

Proof of Concept (try #2)

```
speakersData = [{ id: 1, name: 'Emily' }, { id: 2, name: 'Lucy' }]

Speakers = React.createClass
  render: ->
    rows = _.chunk(speakersData, 4)
    div className: 'speakers',
      @_buildRows()

  _buildRows: ->
    rows.map(@_buildSpeakers)

  _buildSpeakers: (row) ->
    row.map (speaker) ->
      div className: 'speaker', speaker.name
```


imgflip.com

Proof of Concept (try #3)

```
speakersData = [{ id: 1, name: 'Emily' }, { id: 2, name: 'Lucy' }]

Speakers = React.createClass
  render: ->
    rows = _.chunk(speakersData, 4)
    div className: 'speakers',
      rows.map (row) -> Row(row: row)
```

```
Row = React.createClass
  render: ->
    div null, # render must return a React.DOM element or null
      @props.row.map (speaker) -> Speaker(speaker: speaker)
```

```
Speaker = React.createClass
  render: ->
    div className: 'speaker', @props.speaker.name
```

Nice work!

Take a breath.

Asynchronous Data Fetching

```
Speakers = React.createClass



  render: ->
    # ...
```

```
Speakers = React.createClass
  componentWillMount: ->
    @_fetchSpeakers()

  render: ->
    # ...
```

```coffee
Speakers = React.createClass
  componentWillMount: ->
    @_fetchSpeakers()

  render: ->
    # ...

  _fetchSpeakers: ->
    SomeAjaxHelper
      .get('speakers.json')
      .then(@_onSuccess)

  _onSuccess: (data) ->
    # triggers a re-render
    @setState(speakers: data)
```

```coffee
Speakers = React.createClass
  componentWillMount: ->
    @_fetchSpeakers()

  render: ->
    # what do you think happens here?
    rows = _.chunk(@state.speakers, 4)
    div className: 'speakers',
      rows.map (row) -> Row(row: row)

  _fetchSpeakers: ->
    # ...

  _onSuccess: (data) ->
    @setState(speakers: data)
```

React waits for no code.

Speed is the name of the game.

React calls the render method unless specifically told not to.

When there is no initial state nor inital speakers property, @state (and thus, speakers) is undefined.

We could do...

```
Speakers = React.createClass
  componentWillMount: ->
    @_fetchSpeakers()

  render: ->
    if @state and @state.speakers and @state.speakers.length > 0
      rows = _.chunk(@state.speakers, 4)
      div className: 'speakers',
        rows.map (row) -> Row(row: row)
    else
      null

  # ...
```

or we could do...

```
Speakers = React.createClass
  componentWillMount: ->
    @_fetchSpeakers()

  render: ->
    if @state?.speakers?.length > 0
      rows = _.chunk(@state.speakers, 4)
      div className: 'speakers',
        rows.map (row) -> Row(row: row)
    else
      null

  # ...
```

The Right Way™

```
Speakers = React.createClass
  # With this we now have an initial bit of
  # data for our speakers state.
  getInitialState: ->
    speakers: []

  componentWillMount: ->
    @_fetchSpeakers()

  render: ->
    if @state.speakers.length > 0
      rows = _.chunk(@state.speakers, 4)
      div className: 'speakers',
        rows.map (row) -> Row(row: row)
    else
      null
```

Building the Child Components

Row Component

```
Row = React.createClass
  render: ->
    div null,
      @props.row.map (item) =>
        Speaker(speaker: item)
```

Speaker Component

```
Speaker = React.createClass
  render: ->
    div className: 'speaker',
      img className: 'speaker__image', src: @props.speaker.image
      div className: 'speaker__infoBox',
        div className: 'speaker__info',
          h3 className: 'speaker__name', @props.speaker.name
          span null, @props.speaker.work
```

Adding info toggling functionality

What do we need?

- Keep track of currently selected item
- Ability to insert the info at the end of a group
- Click event that toggles the info for an item

```
Speakers = React.createClass
  getInitialState: ->
    speakers: []
    selectedId: null

  render: ->
    # ...
    rows.map (row) ->
      Row(
        row: row
        selectedId: @state.selectedId
        updateSelectedId: @_updateSelectedId
      )

  _updateSelectedId: (selectedId) ->
    @setState(selectedId: selectedId)
```

```
Row = React.createClass
  render: ->
    div null, @_buildRow()

  _buildRow: ->
    selectedItems = @_filterSelected()
    rendered = @props.row.map (item) =>
      Speaker(
        isSelected: item.id is @props.selectedId
        speaker: item
        updateSelectedId: @props.updateSelectedId
      )
    if selectedItems.length > 0
      rendered.push(Info(speaker: selectedItems[0]))
    rendered

  _filterSelected: ->
    @props.row.filter (item) =>
      item.id is @props.selectedId
```

```
Speaker = React.createClass
  render: ->
    div className: 'speaker', onClick: @_handleToggleClick,
      img className: 'speaker__image', src: @props.speaker.image
      div className: 'speaker__infoBox',
        div className: 'speaker__info',
          h3 className: 'speaker__name', @props.speaker.name
          span null, @props.speaker.work

  # Determines selectedId value and
  # triggers the callback function
  # passed down from Speakers
  _handleToggleClick: ->
    # b/c i don't have room for a proper if/else
    selectedId = @props.speaker.id
    selectedId = null if @props.isSelected # reset!
    @props.updateSelectedId(selectedId)
```

```
Info = React.createClass
  render: ->
    div className: 'speakerInfo',
      h3 null, @props.speaker.name
      p null,
        div null, @props.speaker.work
        div null, @props.speaker.twitter
      p null, @props.speaker.bio
```

Rendering to the DOM

```
// index.js

var React = require('react');
var Speakers = require('./Speakers.react');

var div = document.createElement('div');
document.body.insertBefore(div, document.body.firstChild);

React.render(Speakers(), div);
           //    |        |
           // component  container
```

Ship It and Pray...

Why

Truth

Maintaining state in web applications. How...?

In web applications, what has been the source of data truth since day one?

The server.


Hello, IT, have you tried
turning it off and on again?

Mutating the DOM never made sense to me.

The solution?


RE-RENDER
ALL THE THINGS
memegenerator.net

Isn't that expensive on browsers?

Nope.

Virtual DOM

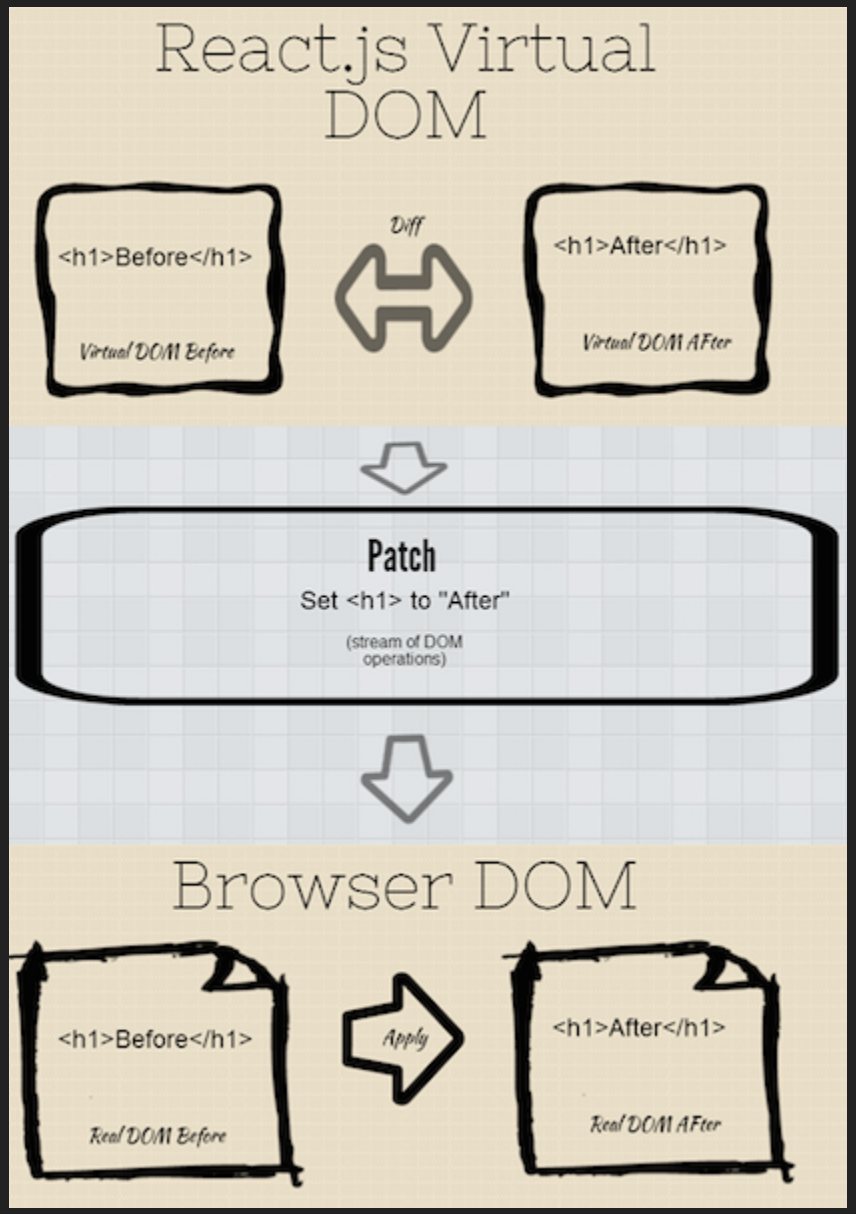


Image credit: Steven Hollidge

Re-rendering subset (good)

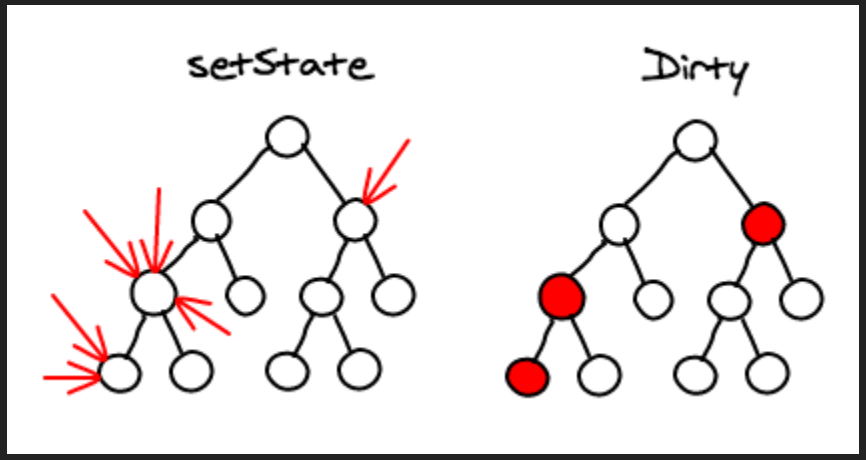


Image credit: Christopher Chedeau

Re-rendering subset (good)

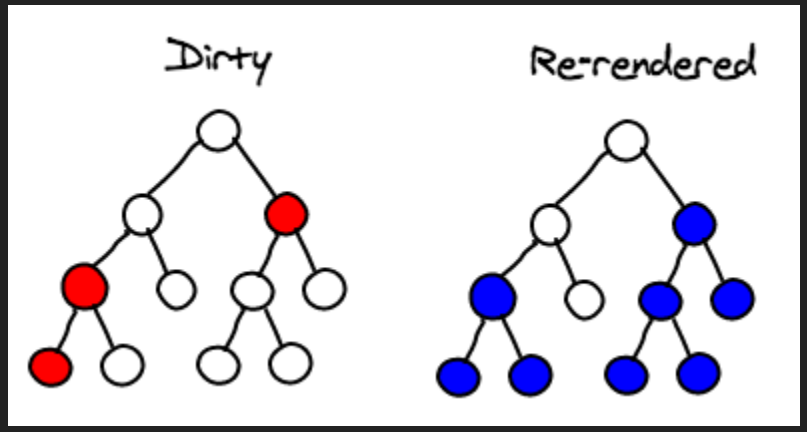


Image credit: Christopher Chedeau

but...

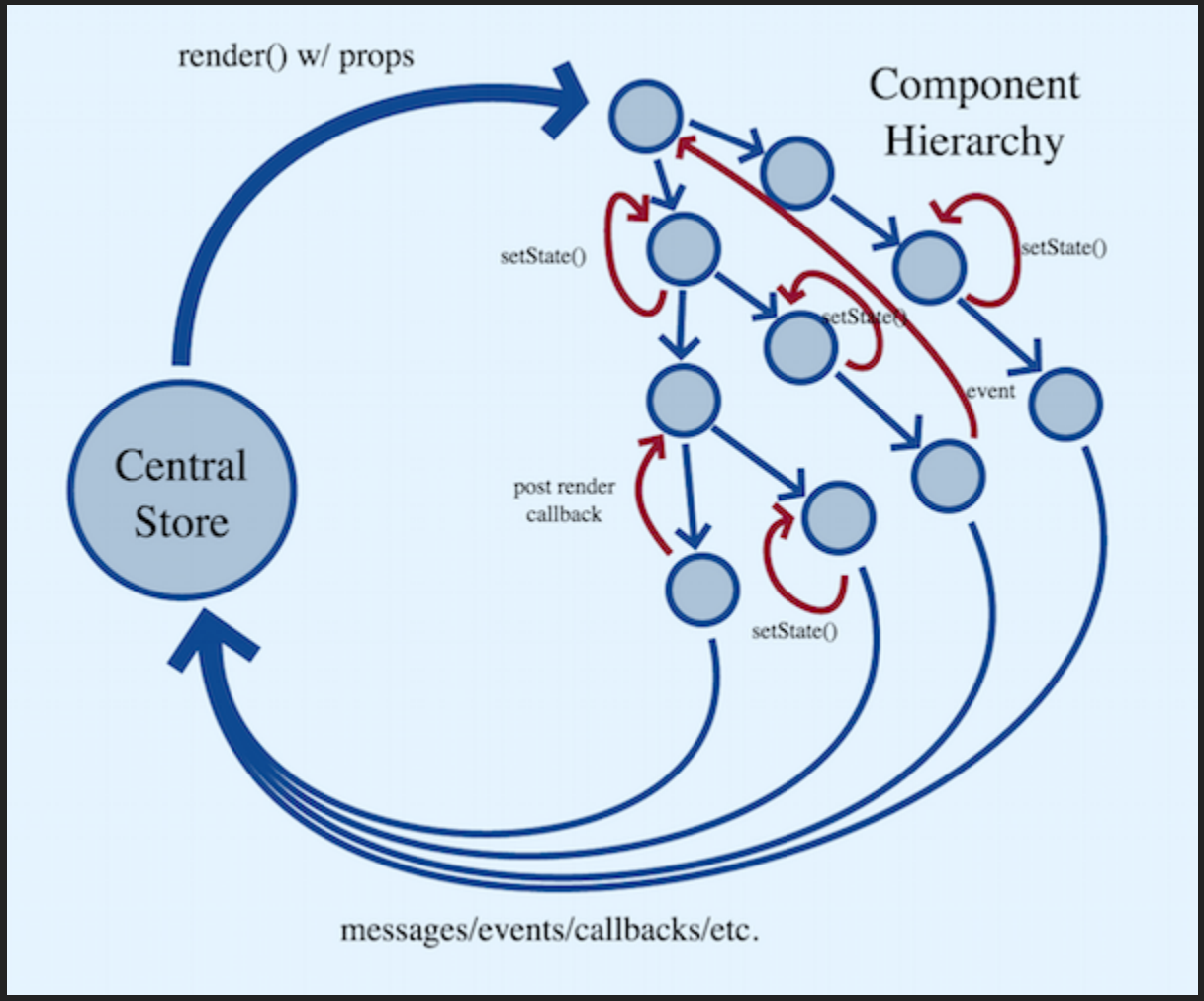
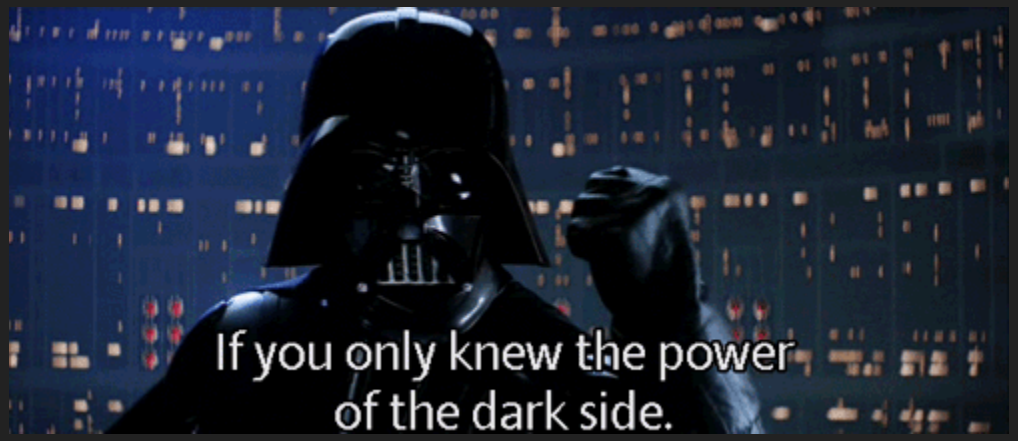


Image credit: Alexander Early

The previous image occurs when one component does not control all of the data for an application, or at least for a set of subcomponents.

It's very tempting to want to manage state inside the component that the state affects.



Remember this?

```
Speaker = React.createClass
  render: ->
    # ...
    Row(updateSelectedId: @_updateSelectedId)

  _updateSelectedId: (selectedId) ->
    # ...
```

```
Row = React.createClass
  render: ->
    # ...
    Speaker(updateSelectedId: @_updateSelectedId)
```

```
Speaker = React.createClass
  render: ->
    # ...
    div className: 'speaker', onClick: @_handleClick
      # ...

  _handleClick: ->
    # ...
    @props.updateSelectedId(selectedId)
```

What that can soon look like

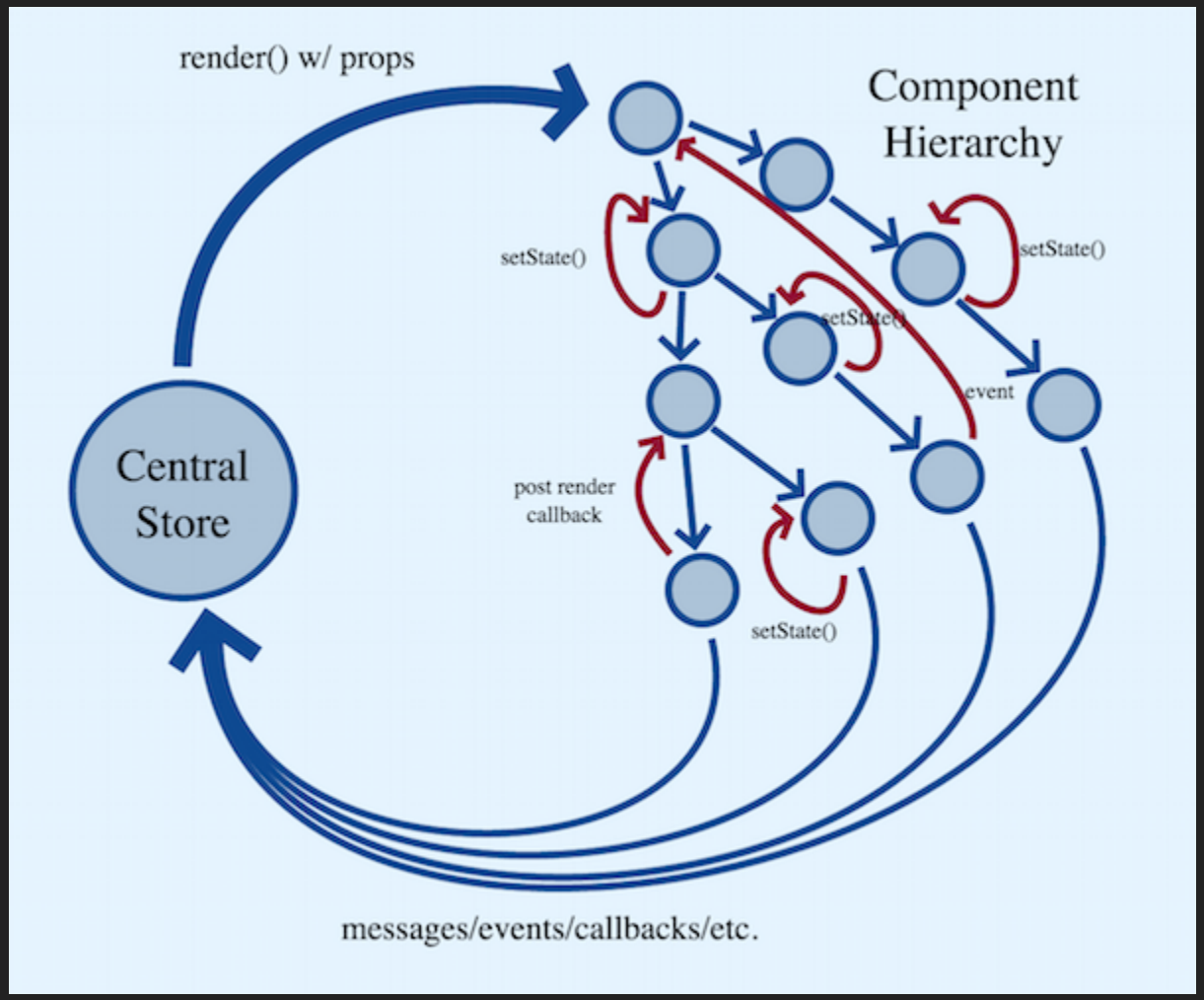


Image credit: Alexander Early

Utilizing Events

```
Speaker = React.createClass
  componentDidMount: ->
    SpeakerEmitter.addSelectionChangeListener(@_updateSelectedId)

  componentWillUnmount: ->
    SpeakerEmitter.removeSelectionChangeListener(@_updateSelectedId)

  render: ->
    Row()

  _updateSelectedId: (selectedId) ->
    # ...
```

```
Row = React.createClass
  render: ->
    Speaker()
```

```
Speaker = React.createClass
  render: ->
    div className: 'speaker', onClick: @_handleClick

  _handleClick: ->
    SpeakerEmitter.emitSelectionChange(selectedId)
```

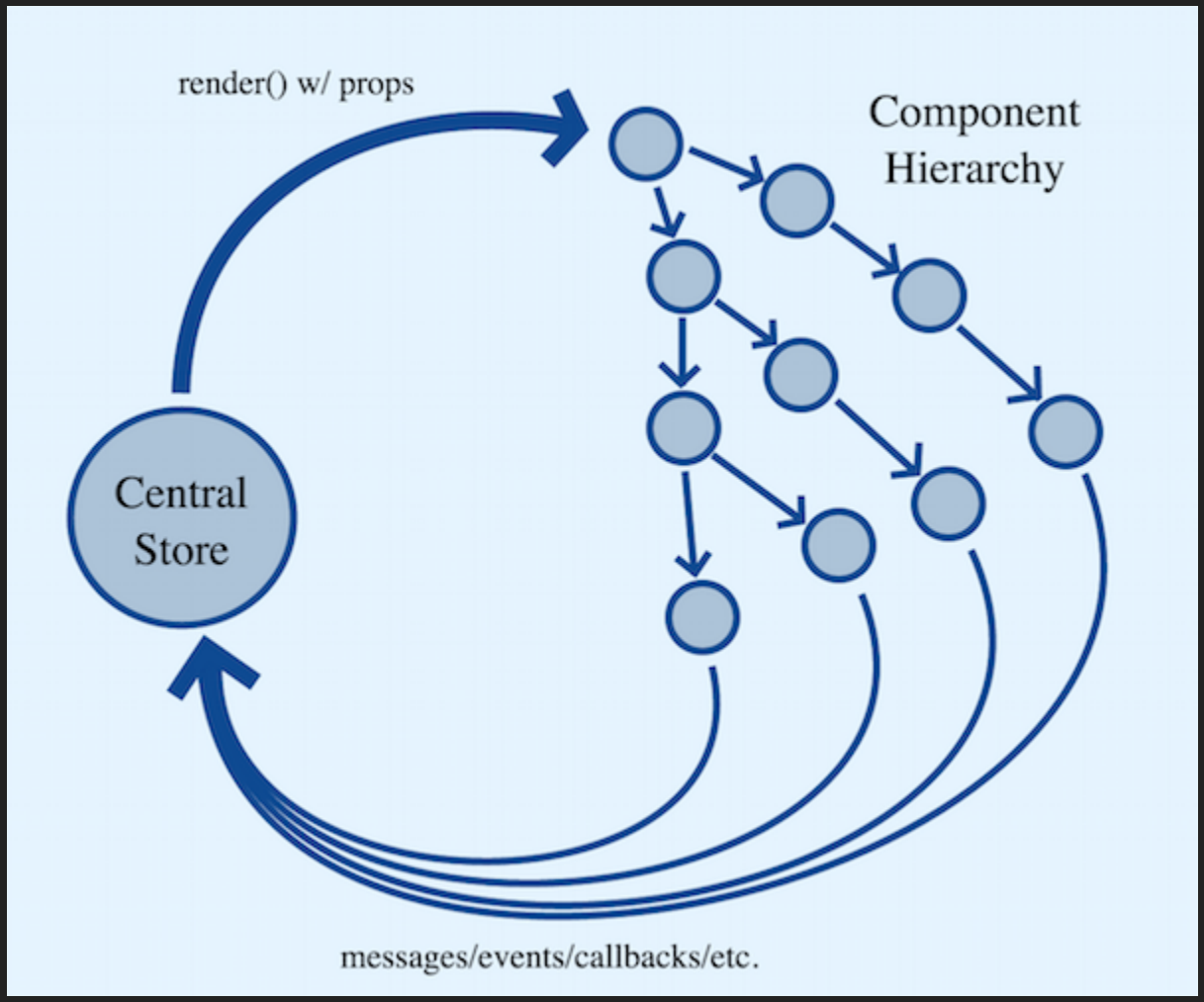
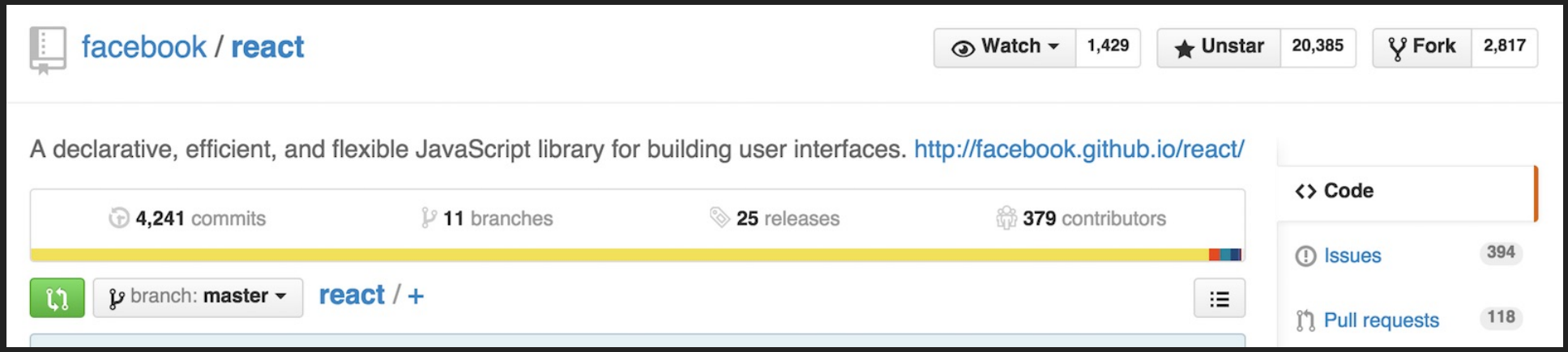


Image credit: Alexander Early

The Future

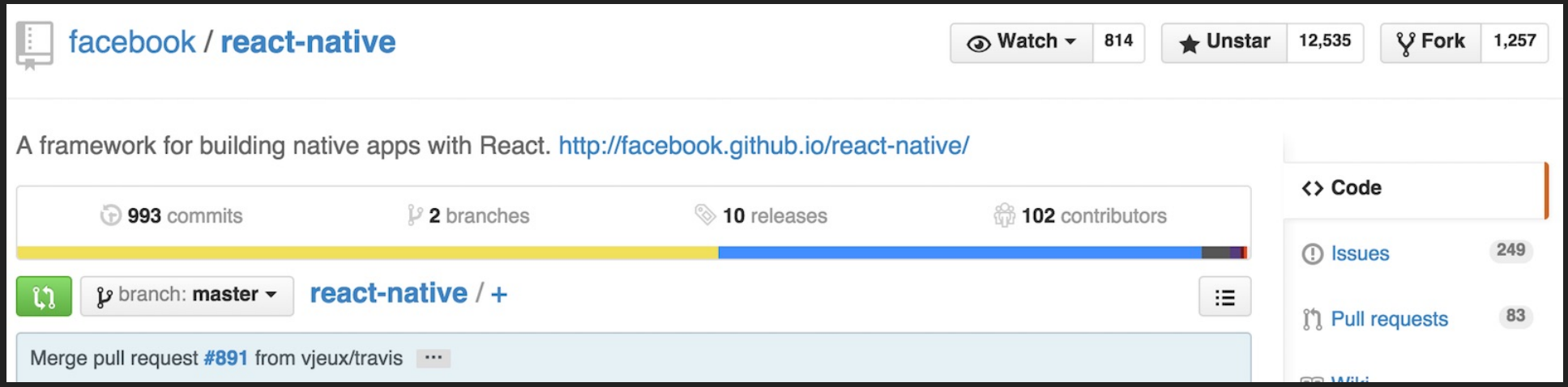
It's quite popular



Client Side & Server-side

You can compile React on the server and use the same templates for server-side rendering as you do for client-side.

It's gone native



They're Open Source Committed

Updating Our Open Source Patent Grant

James Pearce

At Facebook, we strive to open source innovative, world-class software. These are the same technologies that we ourselves use in production, and we strive to ensure that developers feel comfortable about their quality. But we also know that there are other things people look at when choosing to use or build on open source software projects, such as their licenses and terms of use.

We default to using the BSD license when we open source projects. While this is simple and permissive, we also know that intellectual property is a concern to many developers. So over a year ago, we started offering an additional grant which provided rights to any Facebook patents relevant to each given project.

This grant was designed to ensure that developers can use our projects with confidence. But we've also continued to think about how to make its scope clearer, and its intent less ambiguous. Any inadvertent doubts it introduces detract from what we want our program to be about: world-class communities and world-class software.

So today we're pushing a new version of the grant. Its main significant improvement is that it is clearer with regard to what we mean by 'affiliates' and 'assertions' and the circumstances under which the grant terminates, all of which had caused confusion. You can see the full text of the new grant on our **osquery project**, for example.

As of today, this version will be used on all new BSD-licensed Facebook open source projects. What's more, we are currently updating all the projects that used the previous version, and they can also be used under these revised conditions as soon as the diffs land.

We are confident that these changes will make everyone - individuals and companies alike - feel more comfortable about using our projects and taking part in their open communities. And now... back to **shipping software**!

TL;DR

If your current JS framework/lib/stack makes you feel like

give React a shot and see if if makes you feel like