React

A Javascript library for building user interfaces

Just the UI

Just the UI

Renders UI and responds to events
V in MVC
Unopinionated about rest of stack

Just the UI

Renders UI and responds to events

V in MVC

Unopinionated about rest of stack

Just the UI

Renders UI and responds to events

V in MVC

Unopinionated about rest of stack

Just the UI

Renders UI and responds to events

V in MVC

Unopinionated about rest of stack

Components, not templates

Separation of Concerns

Separation of Concerns (MVC)

```
app.controller( 'QuotesController', function( $scope, QuoteApi ) {
  $scope.quotes = QuoteApi.get( ... )
}
```

```
<ul ng-repeat="quote in quotes">
  <li>{{quote.id}} {{quote.name}}</li>
</ul>
```

Separation of Concerns (MVC)

```
<ul ng-repeat="quote in quotes">
  <li>
    <i ng-class="{
      'icon-checked': isQuoteChecked( quote ),
      'icon-unchecked' : !isQuoteChecked( quote )
     }"
     ng-click="toggleCheck( quote )"></i>
    {{quote.id}} {{quote.name}}
  </li>
</ul>
```

Separation of Concerns (MVC)

```
$scope.quotes = QuoteApi.get( ... )

var checked = {};

$scope.isQuoteChecked = function( quote ) {
  return checked[quote.id];
}

$scope.toggleCheck = function( quote ) {
  checked[quote.id] = !checked[quote.id];
}
```

Separation of Concerns

Reduce Coupling

Increase Cohesion

Reduce Coupling

The degree to which each module relies on other modules

Display logic and templates are inevitable tightly coupled - if one changes the other must change

Reduce Coupling

The degree to which each module relies on other modules

Display logic and templates are inevitable tightly coupled - if one changes the other must change

Increase Cohesion

The degree to which elements of a module belong together

Display logic and templates are cohesive - they both represent the UI

Increase Cohesion

The degree to which elements of a module belong together

Display logic and templates are cohesive - they both represent the UI

Separate concerns not technologies

Components

- Encapsulates both "description" and logic
- Are easily composable
- Are just javascript

```
var Avatar = React.createClass( {

   propTypes: {
     name: React.PropTypes.name.isRequired
   },

   // Lifecycle methods
   componentDidMount : function() { },
   componentWillUnmount : function() { },
   ..

   render: function() {
     return React.createElement( ‘div’, { className: ‘’ }, [
              React.createElement( ‘span’, null, ‘Hello ’ + this.props.name )
            ] )
   }
} )

React.render( React.createElement( Avatar, { name: ‘Bob’ } ), document.body );
```

```
class Avatar extends React.Component {

   // Lifecycle methods
   componentDidMount() { }
   componentWillUnmount() { }
   ..

   render() {
     return React.createElement( ‘div’, { className: ‘’ },
              React.createElement( ‘span’, null, ‘Hello ’ + this.props.name )
            )
   }
};

Avatar.propTypes = { name: React.PropTypes.name.isRequired }

React.render( React.createElement( Avatar, { name: ‘Bob’ } ), document.body );
```

JSX

Optional syntax extension to javascript

Provides familiar syntax for defining tree structures with attributes (xml)

```
var elm = <ul>
            <li><i class=”avatar”></i> User</li>
          </ul>;
```

```
var elm = <ul>
            <li><i class=”avatar”></i> User</li>
          </ul>;
```

```
var elm = React.createElement( ‘ul’, null,
            React.createElement( ‘li’, null,
              React.createElement(
               ‘i’, { className: avatar} ), ‘User’
            )
          )
```

```
var user = ..
var elm = <ul>
  <li>
    <i className={user.icon || ‘avatar’}></i> {user.name || ‘User’}
  </li>
</ul>;
```

```
var user = ..
var elm = <ul>
  <li>
    <i className={user.icon || ‘avatar’}></i> {user.name || ‘User’}
  </li>
</ul>;

var elm = React.createElement( ‘ul’, null,
           React.createElement( ‘li’, null,
              React.createElement(
                ‘i’, { className: user.icon || ‘avatar’ }, user.icon || ‘avatar’
              )
            )
          )
```

```
var Avatar = React.createClass( {
   render() {
     var user = this.props.user;
     return <i className={user.icon || ‘avatar’}></i> {user.name || ‘User’}
   }
} )
```

```
var Avatar = React.createClass( {
   render() {
     var user = this.props.user;
     return <i className={user.icon || ‘avatar’}></i> {user.name || ‘User’}
   }
} )

var user = ..
var elm = <ul>
  <li>
    <Avatar user={user}/>
  </li>
</ul>;
```

```
var Avatar = React.createClass( {
   render() {
     var user = this.props.user;
     return <i className={user.icon || ‘avatar’}></i> {user.name || ‘User’}
   }
} )

var user = ..
var elm = React.createElement( ‘ul’, null,
            React.createElement( ‘li’, null,
              React.createElement( Avatar, { user: user } )
            )
          )
```


item = el->FirstChildElement
ElementDesc elDesc;
sp_name = item->Attribute
spritename = item->Attribute
x = boost::lexical_cast<float>
y = boost::lexical_cast<float>
offset = boost::lexical_cast<float>
layer = 50; // default
item->Attribute("layer")
layer = boost::lexical_cast
elDesc.name_ = sp_name;
elDesc.spriteName_ = spritename;
elDesc.x_ = x;
DEMO

Building UI is Hard

Building UIs is hard

Because of **state** management

State of UI elements, State of user input, state of domain model.

Building UIs is hard

Because of **state** management

State of UI elements, state of user input, state of domain model.

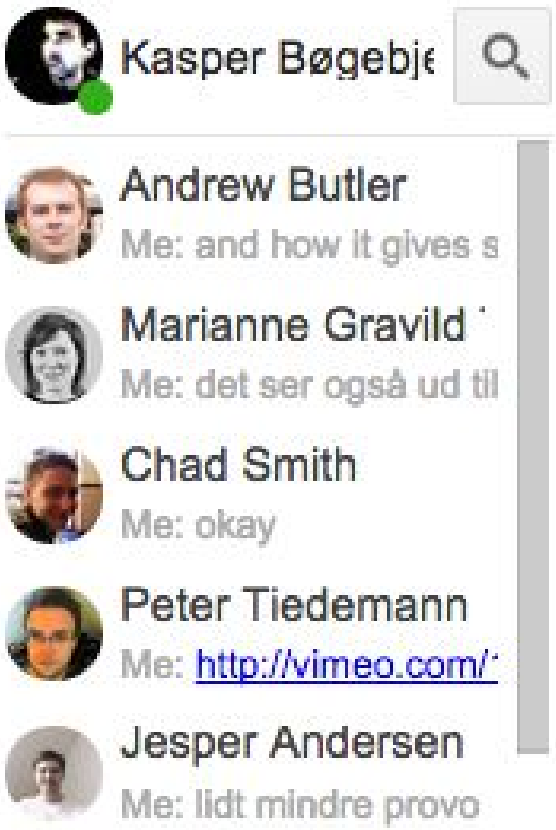
Building UIs is hard

Because of **state** management
State of UI elements, **state** of user input, state of domain model.

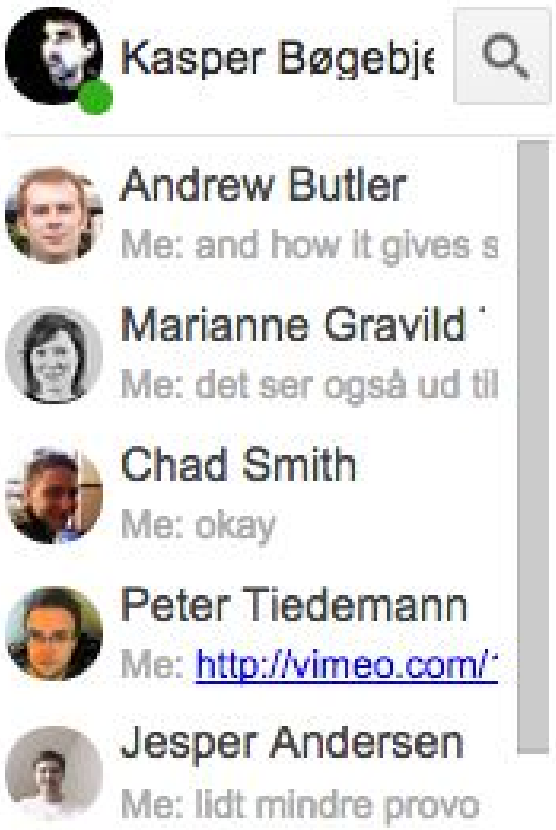
Building UIs is hard

Because of **state** management
State of UI elements, state of user input,
state of domain model.

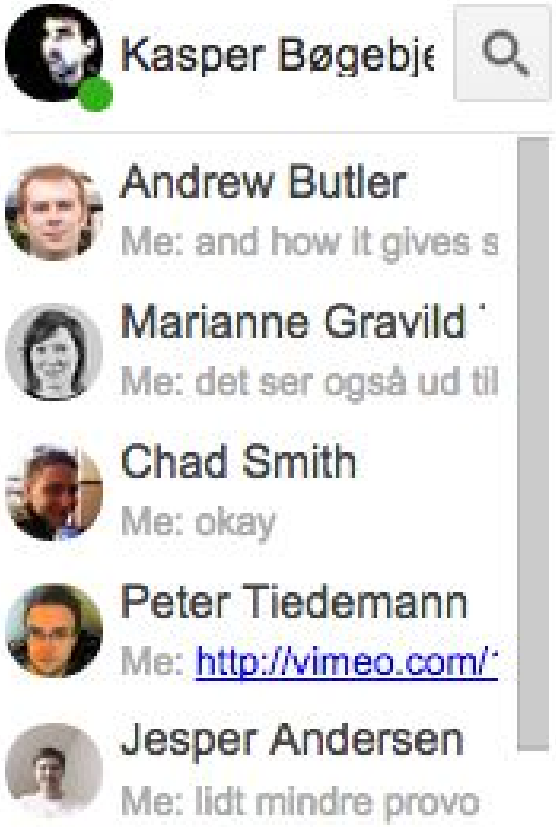
Challenge


Kasper Bøgebje
Andrew Butler
Me: and how it gives s
Marianne Gravild
Me: det ser også ud til
Chad Smith
Me: okay
Peter Tiedemann
Me: http://vimeo.com/
Jesper Andersen
Me: lidt mindre provo

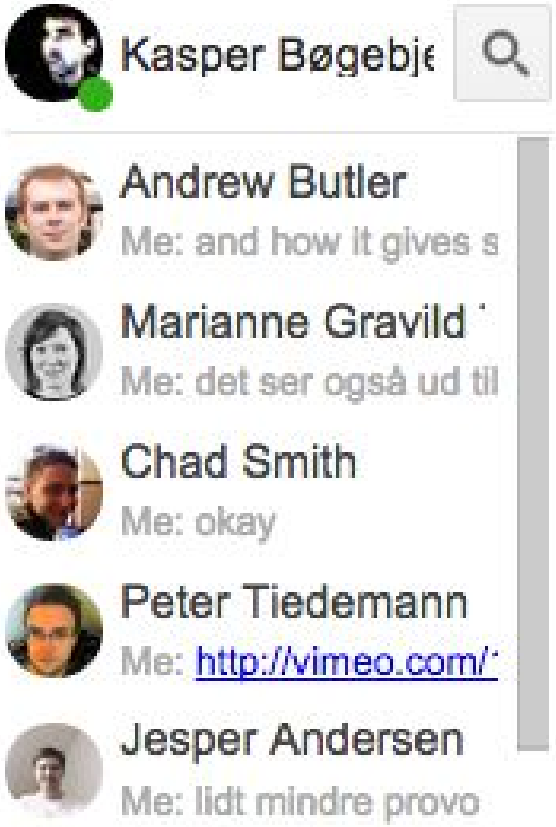
```
[{
  "name" : "Kasper",
  "status" : "online",
  "client"  : "web",
  ..
},
{
  "name" : "Andrew",
  "status" : "offline",
  "client"  : "mobile",
  ..
},
{
  "name" : "Marianne",
  "status" : "idle",
  "client"  : "web",
  ..
}]
```



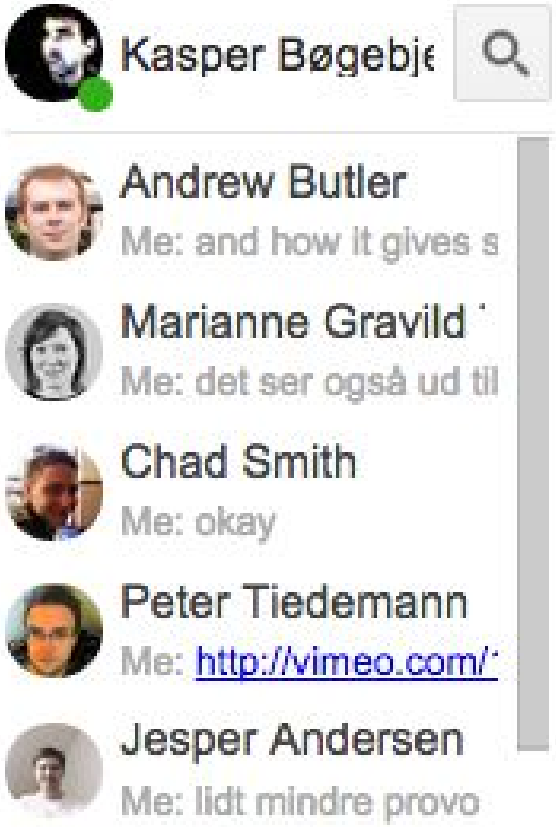
Andrew went online



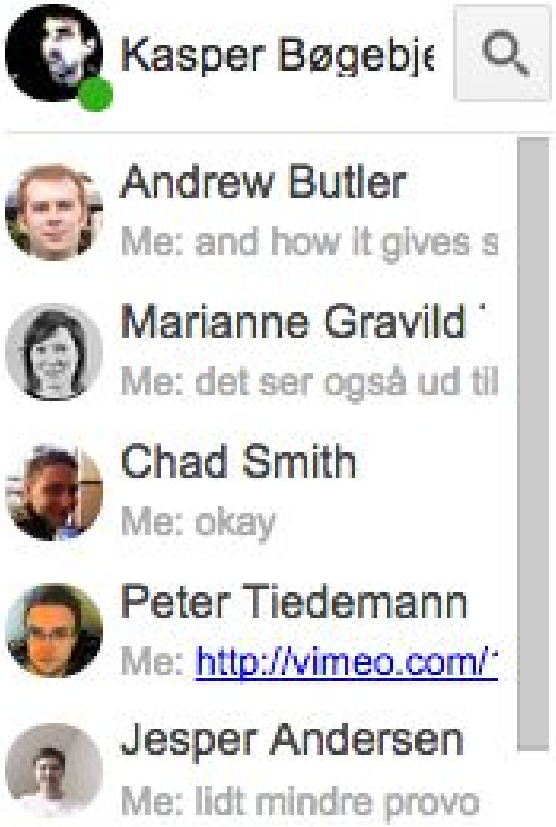
Chad went online



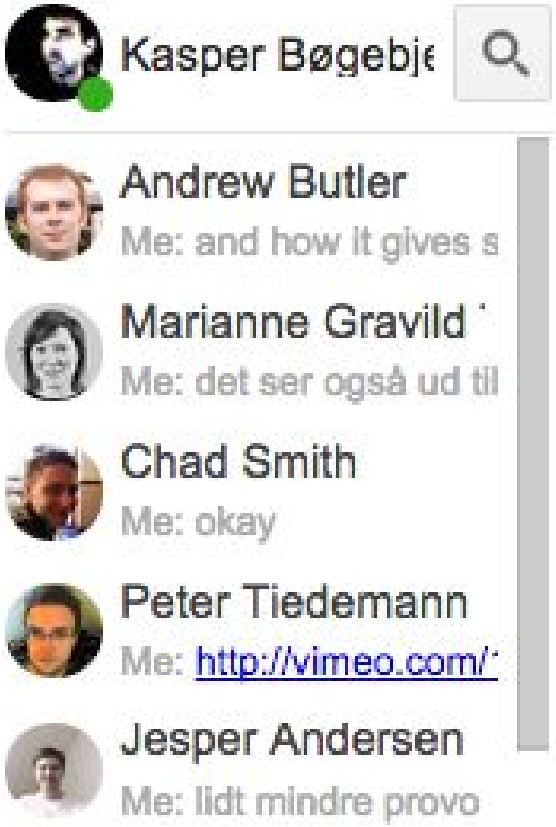
Kasper is idle



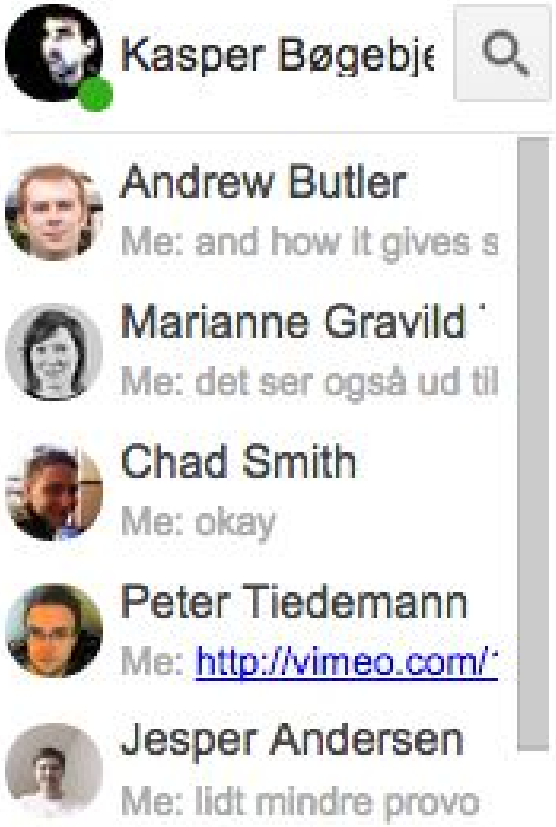
Andrew went offline



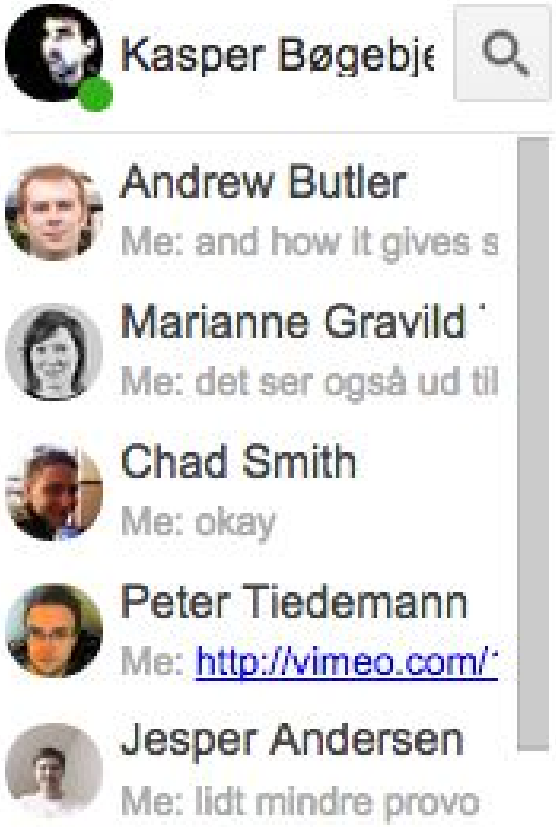
Jesper went online



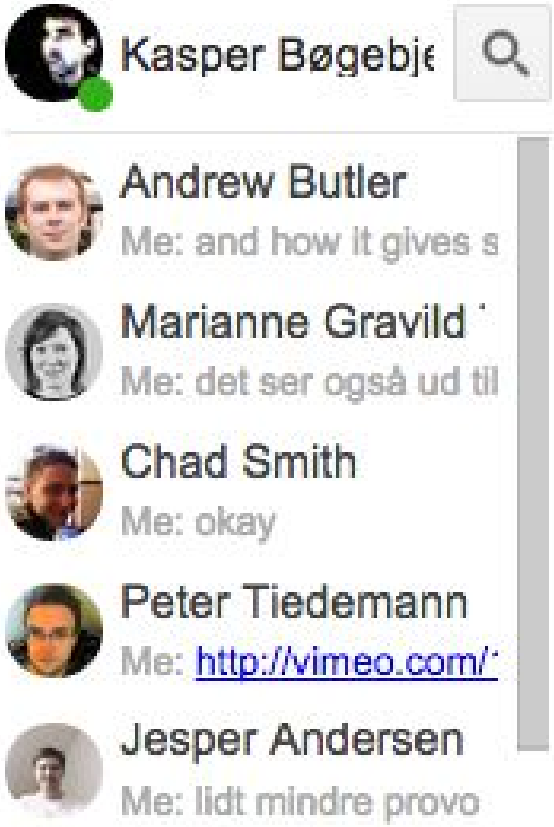
Kasper is on mobile



Peter is busy

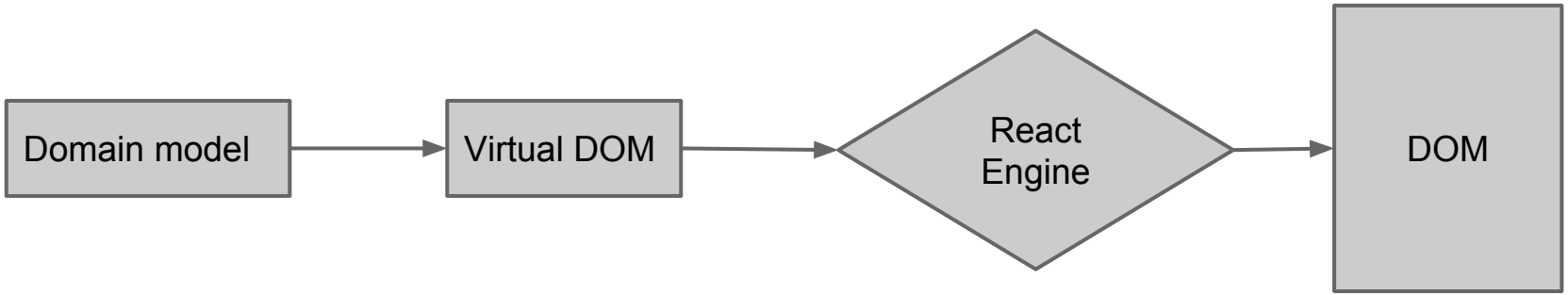


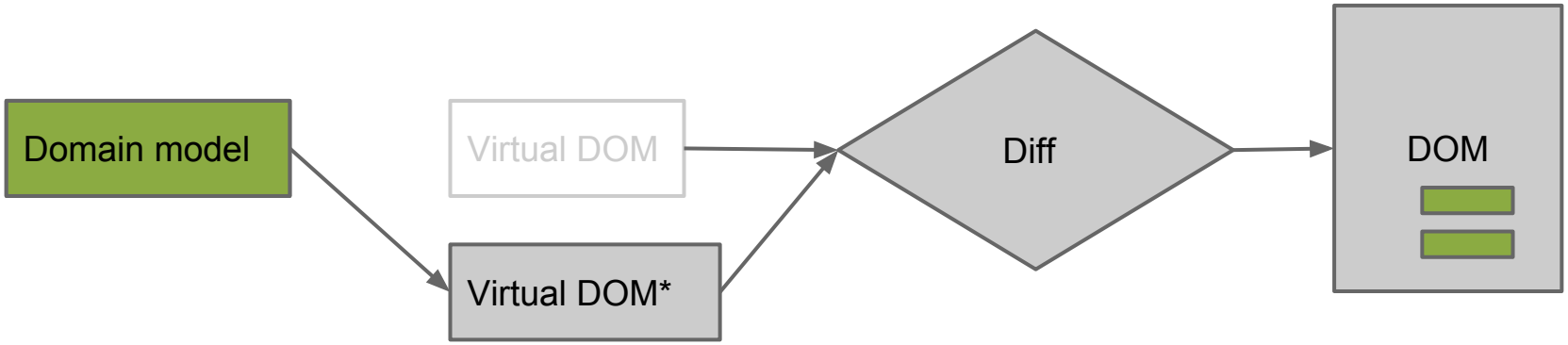
?

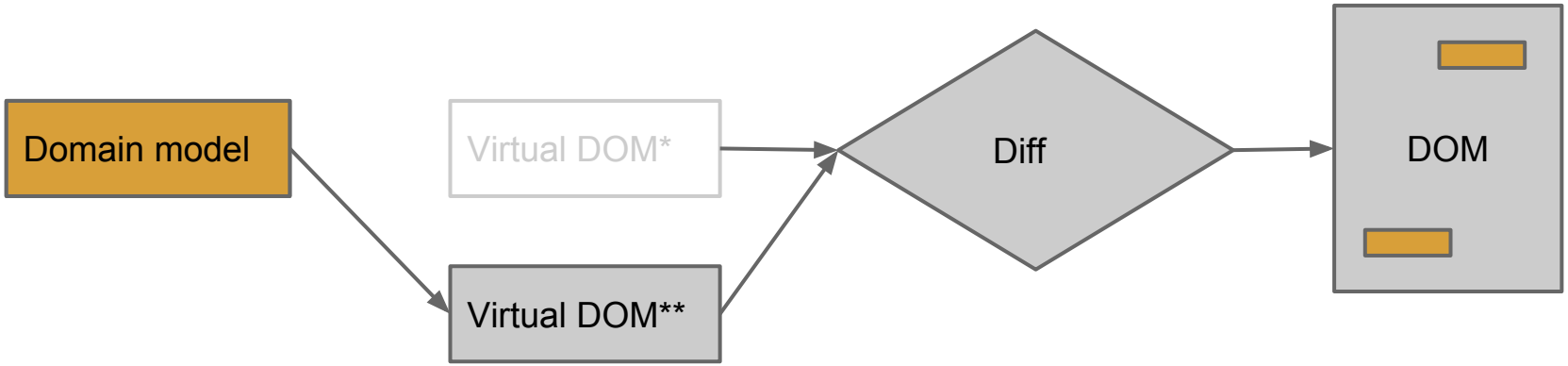

Kasper Bøgebje
Andrew Butler
Me: and how it gives s
Marianne Gravild
Me: det ser også ud til
Chad Smith
Me: okay
Peter Tiedemann
Me: http://vimeo.com/
Jesper Andersen
Me: lidt mindre provo

Let’s do it 90s style

When data changes refresh the page




Domain model
Virtual DOM
Virtual DOM*
Diff
DOM



Components are functions

- Given a set of inputs (props), return a virtual DOM
- Components (functions) can have local variables (state)
- Can be composed of other components


item = el->FirstChildElement
ElementDesc elDesc;
sp_name = item->Attribute
spritename = item->Attribute
x = boost::lexical_cast<float>
y = boost::lexical_cast<float>
offset = boost::lexical_cast<float>
layer = 50; // default
item->Attribute("layer")
layer = boost::lexical_cast
elDesc.name_ = sp_name;
elDesc.spriteName_ = spritename;
elDesc.x_ = x;
DEMO

Resources

- Demos https://github.com/configit/lunch.js/tree/master/01
- Rethinking best practices https://www.youtube.com/watch?v=x7cQ3mrcKaY
- React homepage http://facebook.github.io/react/
- React Training https://github.com/rpflorence/react-training
- Diff algorithm http://facebook.github.io/react/docs/reconciliation.html

Flux - Application Architecture