Rethinking MVC with React Native & ReactiveCocoa

Sam Ritchie (@FakeSamRitchie)
Ben Teese (@benteese)

Agenda

- What's wrong with MVC?
- React Native
- ReactiveCocoa
- Other things to look at

Mobile Development

Network Connectivity Change

Async Network Callback

Touch Events

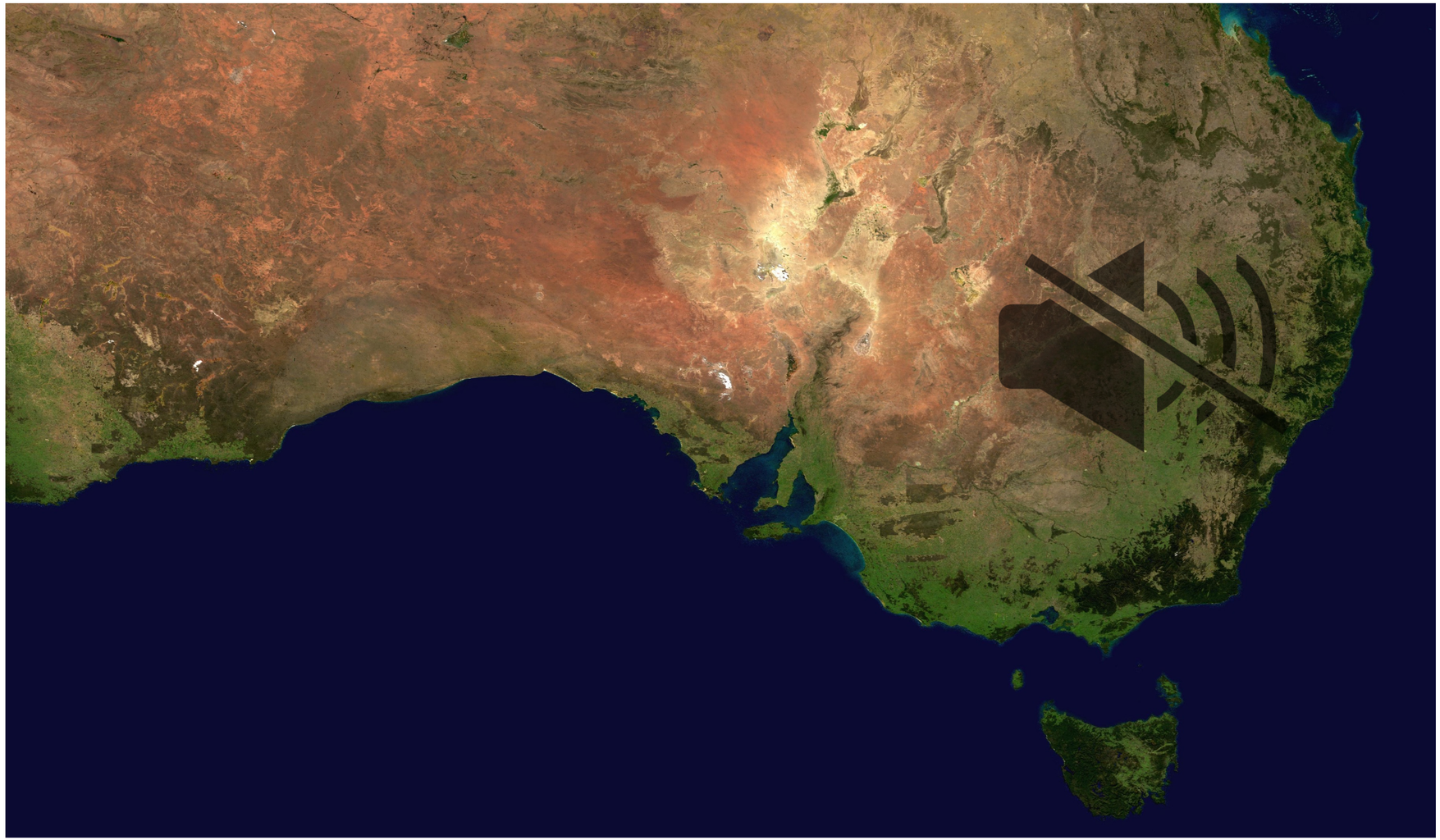
Push Notification

Location Updates

Background Thread

Phone Call

Bluetooth Device/Watch Comms



Mutable State

“UIs are big, messy, mutable, stateful bags of sadness”

- Josh Abernathy

Re-thinking MVC

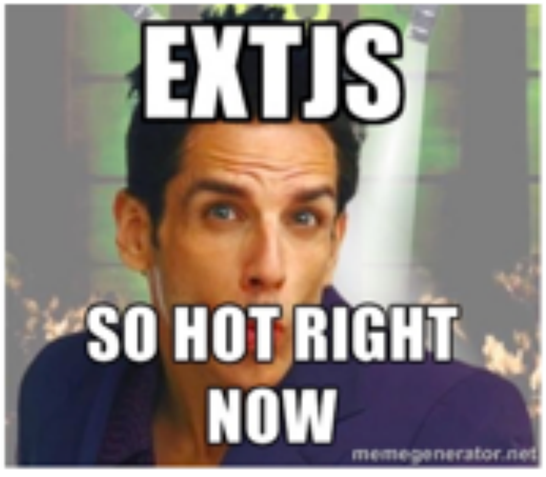
- Minimise/manage state
- Apply functional concepts
- Structured event handling

Tractable

React Native

2007

2009

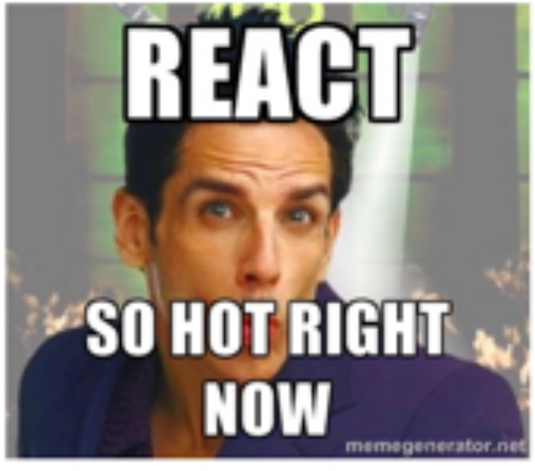


2012

2013

2014

2015

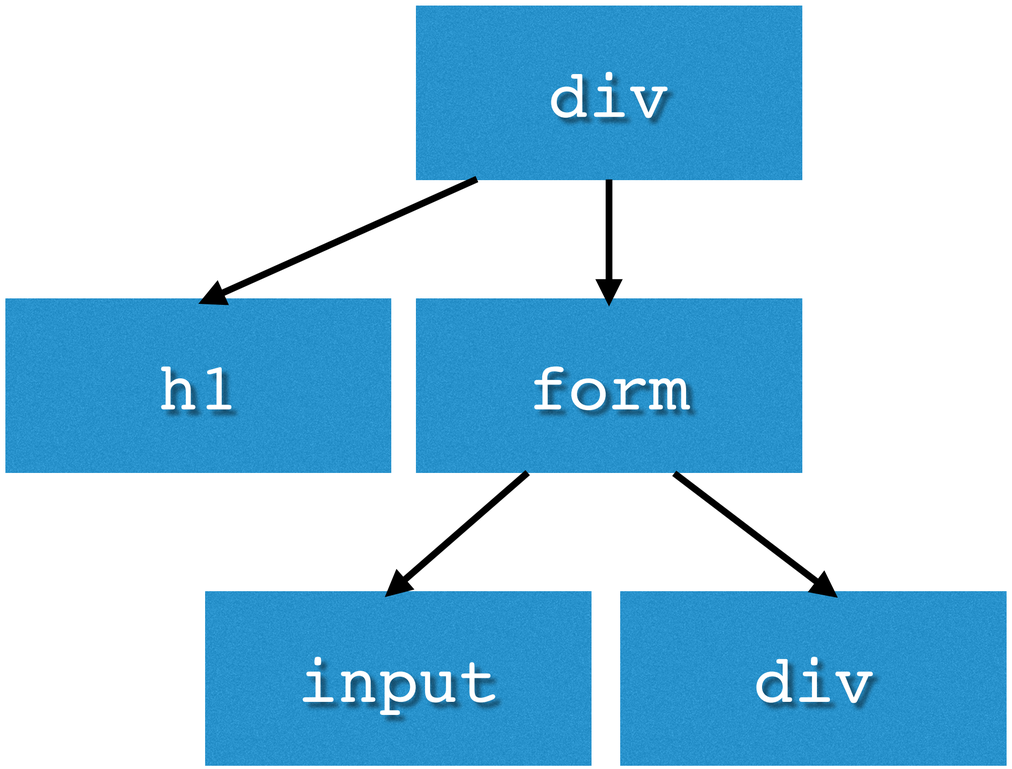


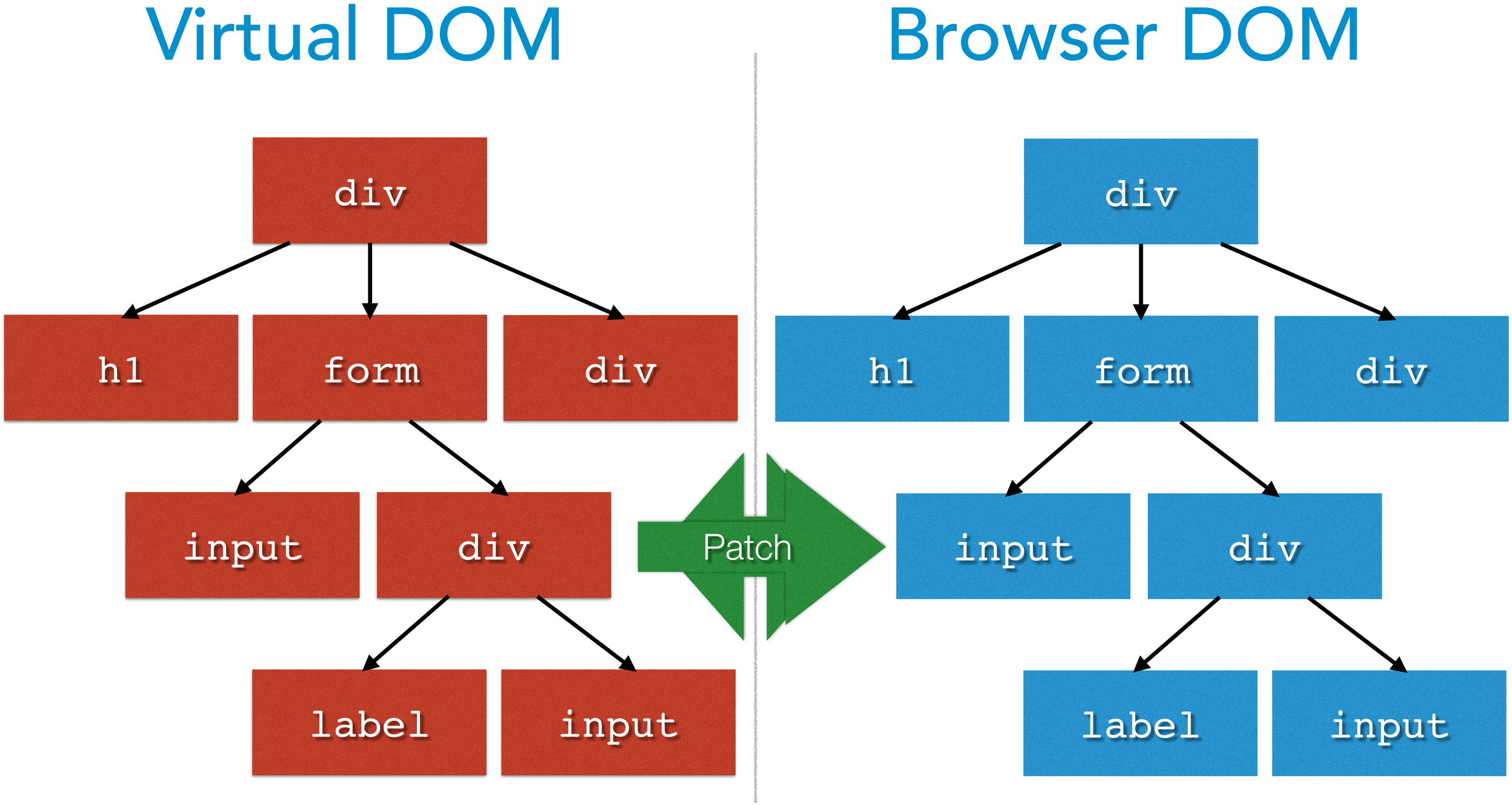
The React Programming Model

- Virtual View Hierarchies
- Declarative UIs
- One-way Data Flow

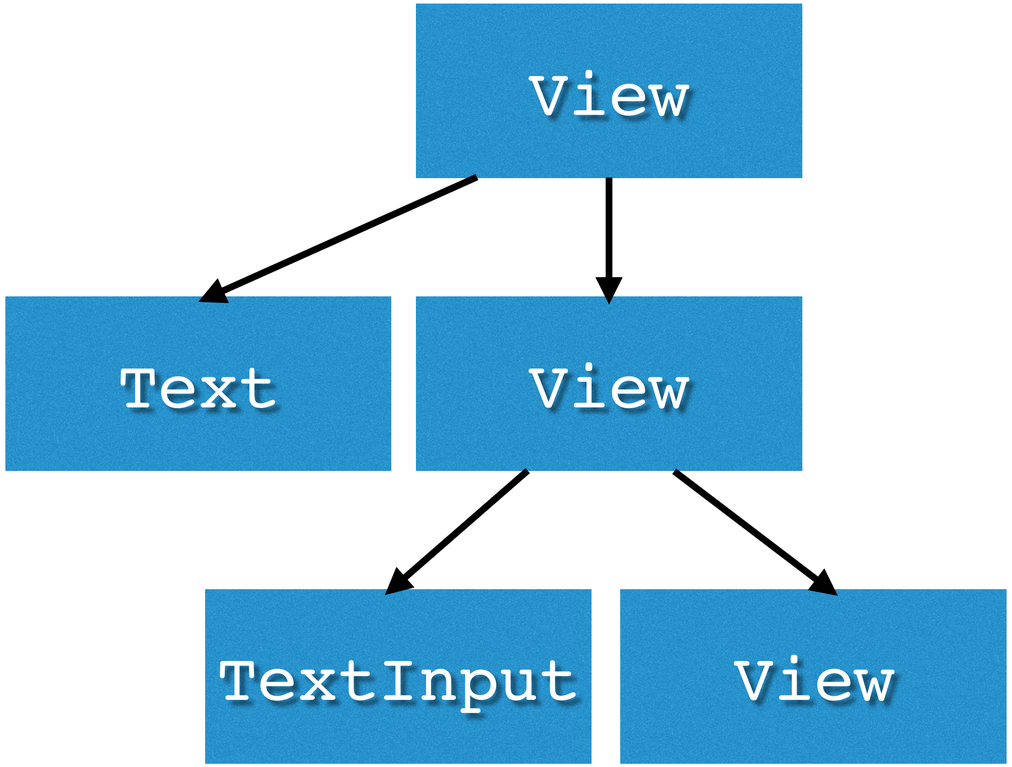
Virtual View Hierarchies

Browser DOM

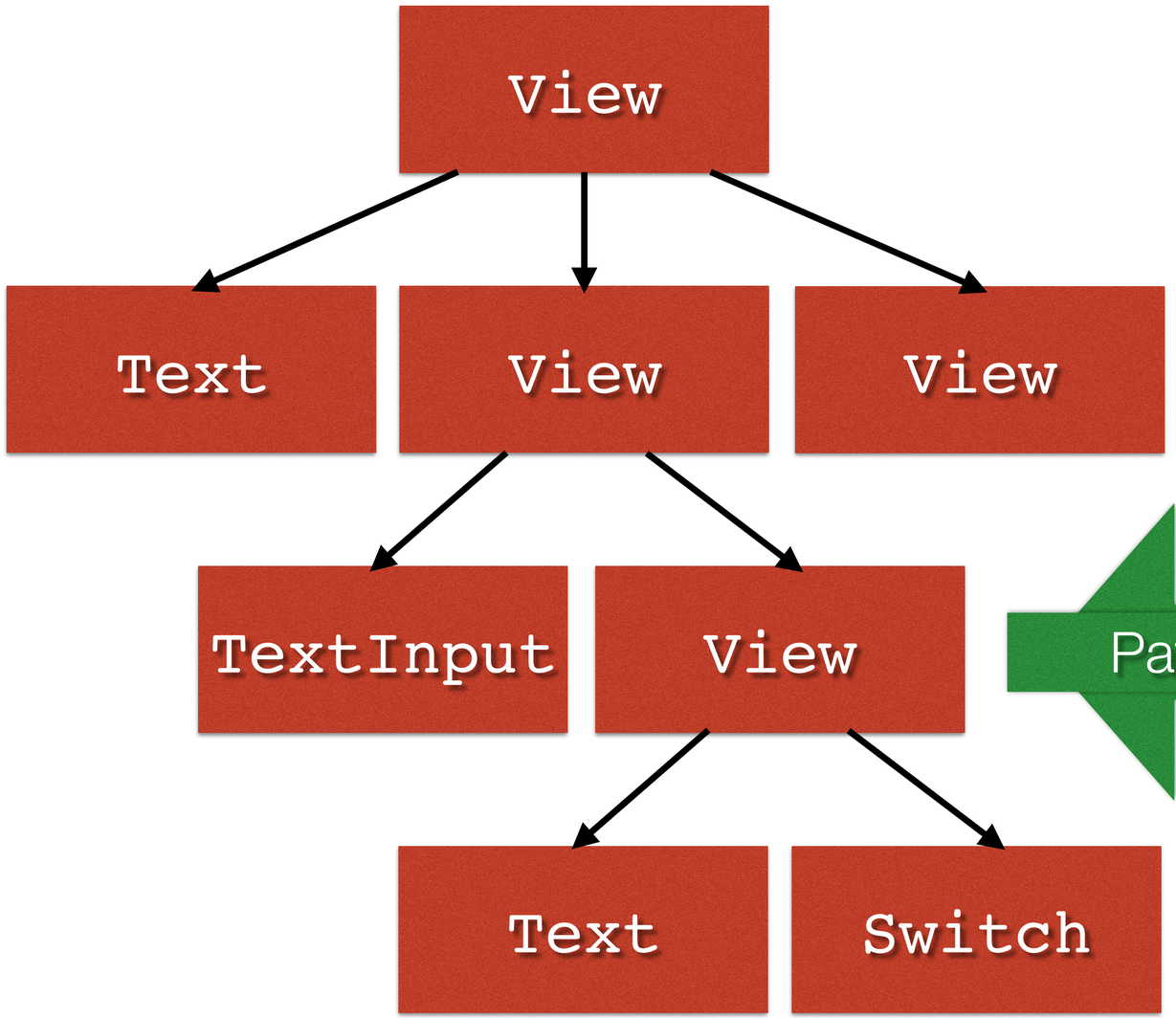



Virtual DOM
Browser DOM
div
h1
form
div
input
div
label
input
Patch
div
h1
form
div
input
div
label
input

Native View Hierarchy

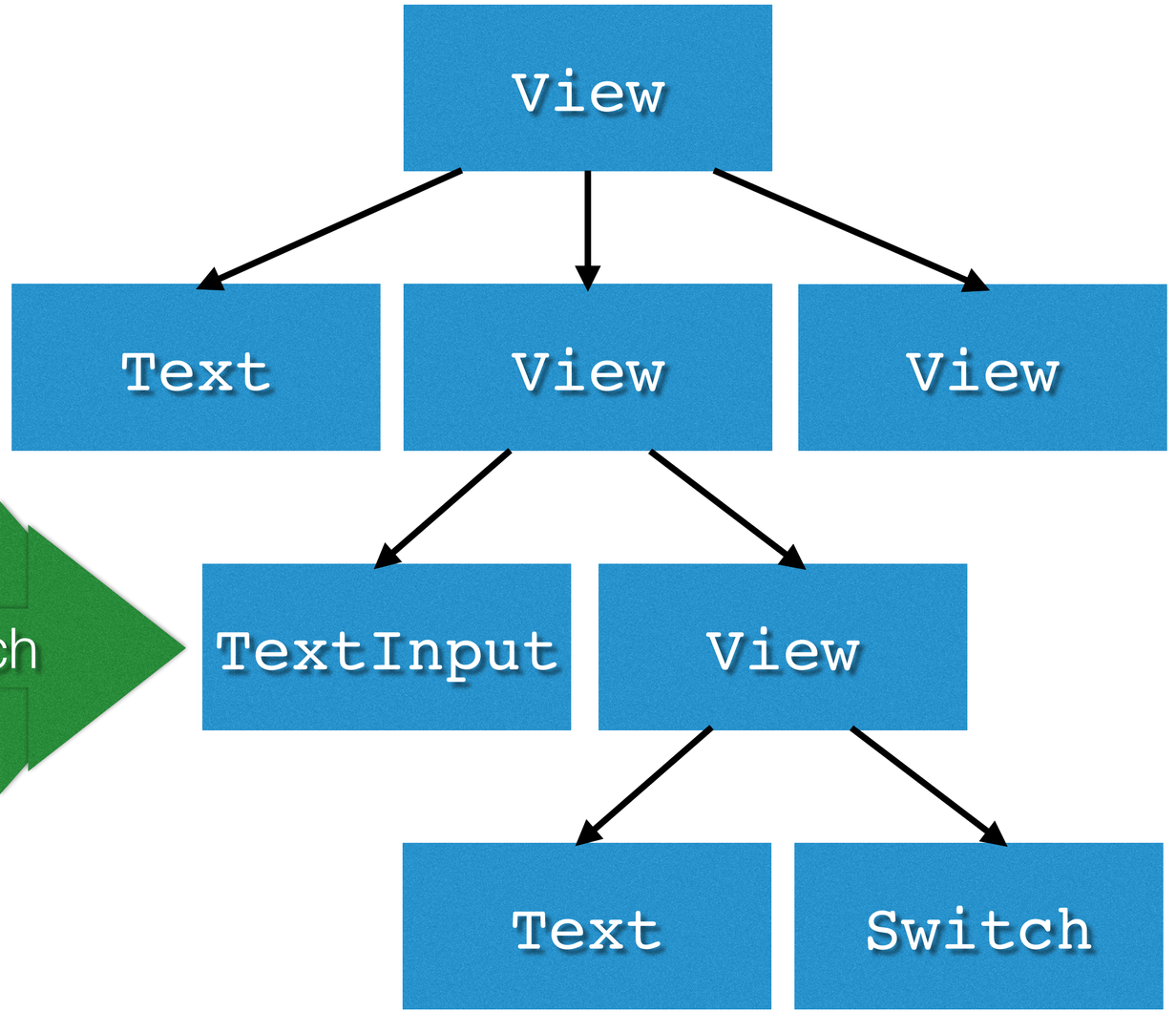


Virtual View Hierarchy

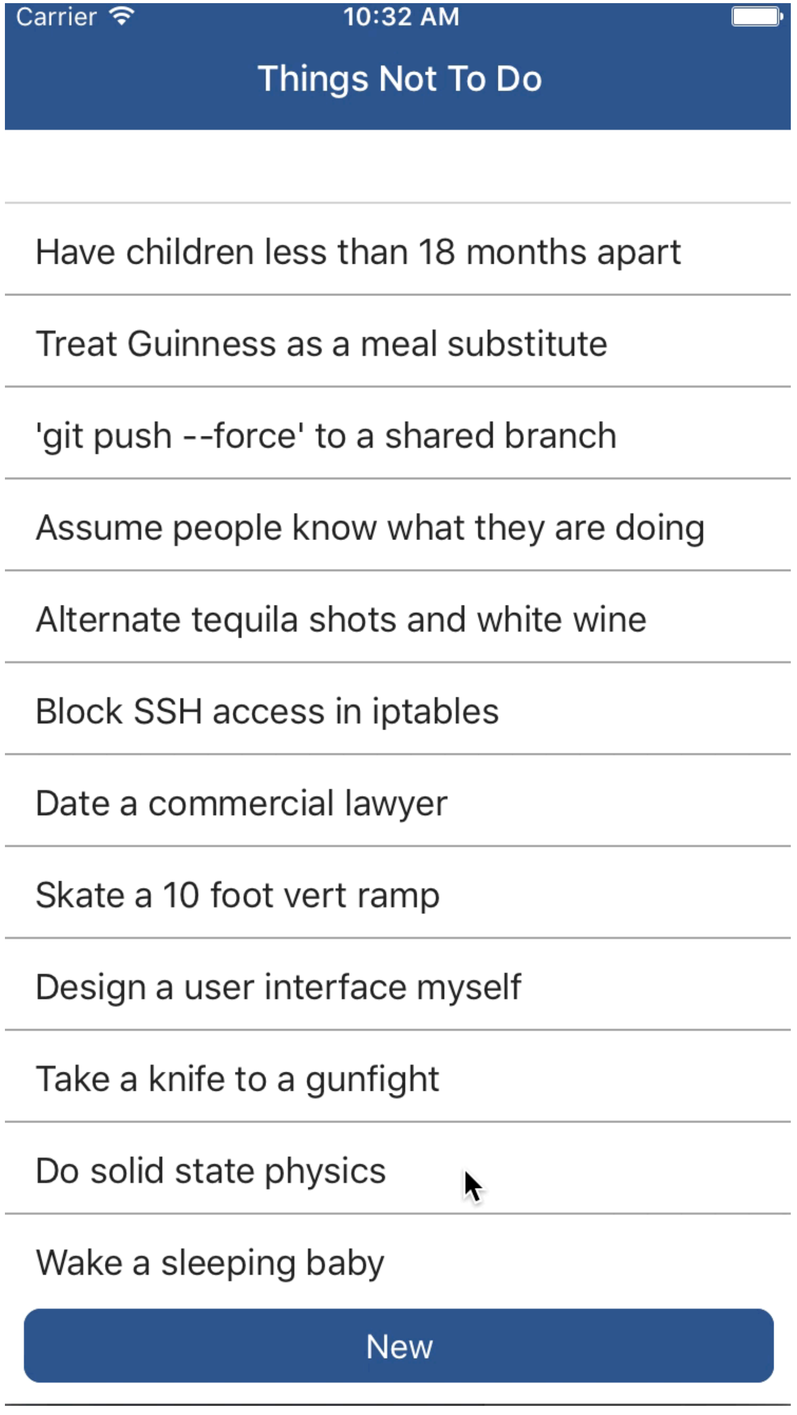


Patch

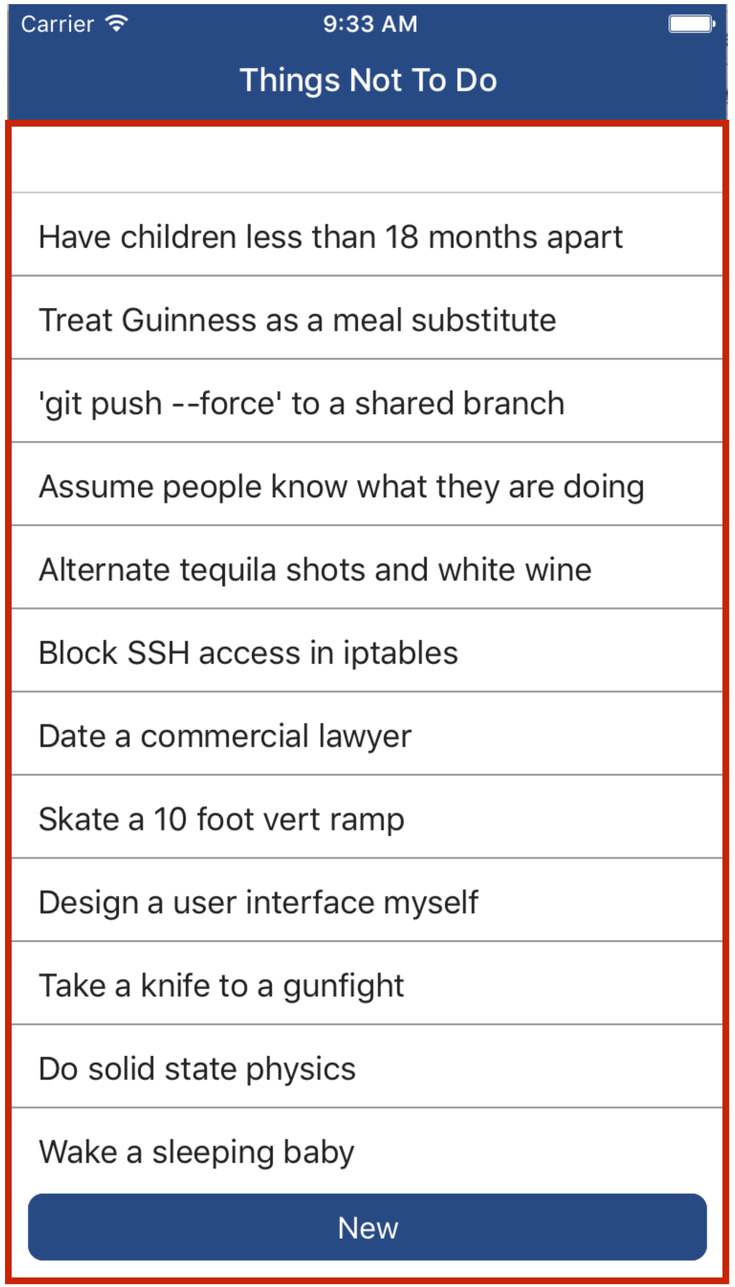
Native View Hierarchy



Declarative UIs


Carrier
10:32 AM
Things Not To Do
Have children less than 18 months apart
Treat Guinness as a meal substitute
'git push --force' to a shared branch
Assume people know what they are doing
Alternate tequila shots and white wine
Block SSH access in iptables
Date a commercial lawyer
Skate a 10 foot vert ramp
Design a user interface myself
Take a knife to a gunfight
Do solid state physics
Wake a sleeping baby
New

TDMain


Carrier
9:33 AM
Things Not To Do
Have children less than 18 months apart
Treat Guinness as a meal substitute
'git push --force' to a shared branch
Assume people know what they are doing
Alternate tequila shots and white wine
Block SSH access in iptables
Date a commercial lawyer
Skate a 10 foot vert ramp
Design a user interface myself
Take a knife to a gunfight
Do solid state physics
Wake a sleeping baby
New

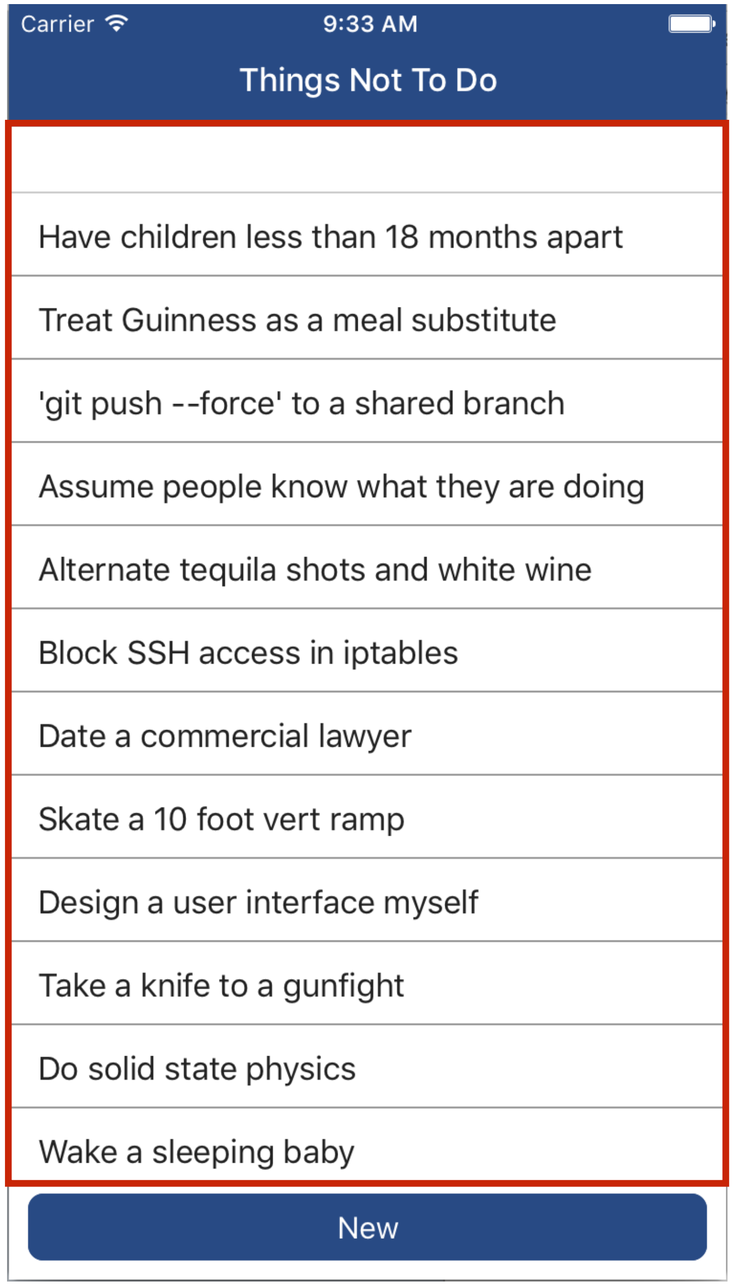
TDMain

```
var React = require('react-native');
var TDFilterableList = require('./TDFilterableList');
var TDButton = require('./TDButton');

var { Component, View } = React;

var items = [
  {description: 'Have children less than 18 months apart'},
  {description: 'Treat Guinness as a meal substitute'},
  …
];

class TDMain extends Component {
  render() {
    return (
      <View>
        <TDFilterableList items={items}/>
        <TDButton text="New"/>
      </View>
    );
  }
}
```

```
return (
  <View>
    <TDFilterableList items={items}/>
    <TDButton>New</TDButton>
  </View>
);
```

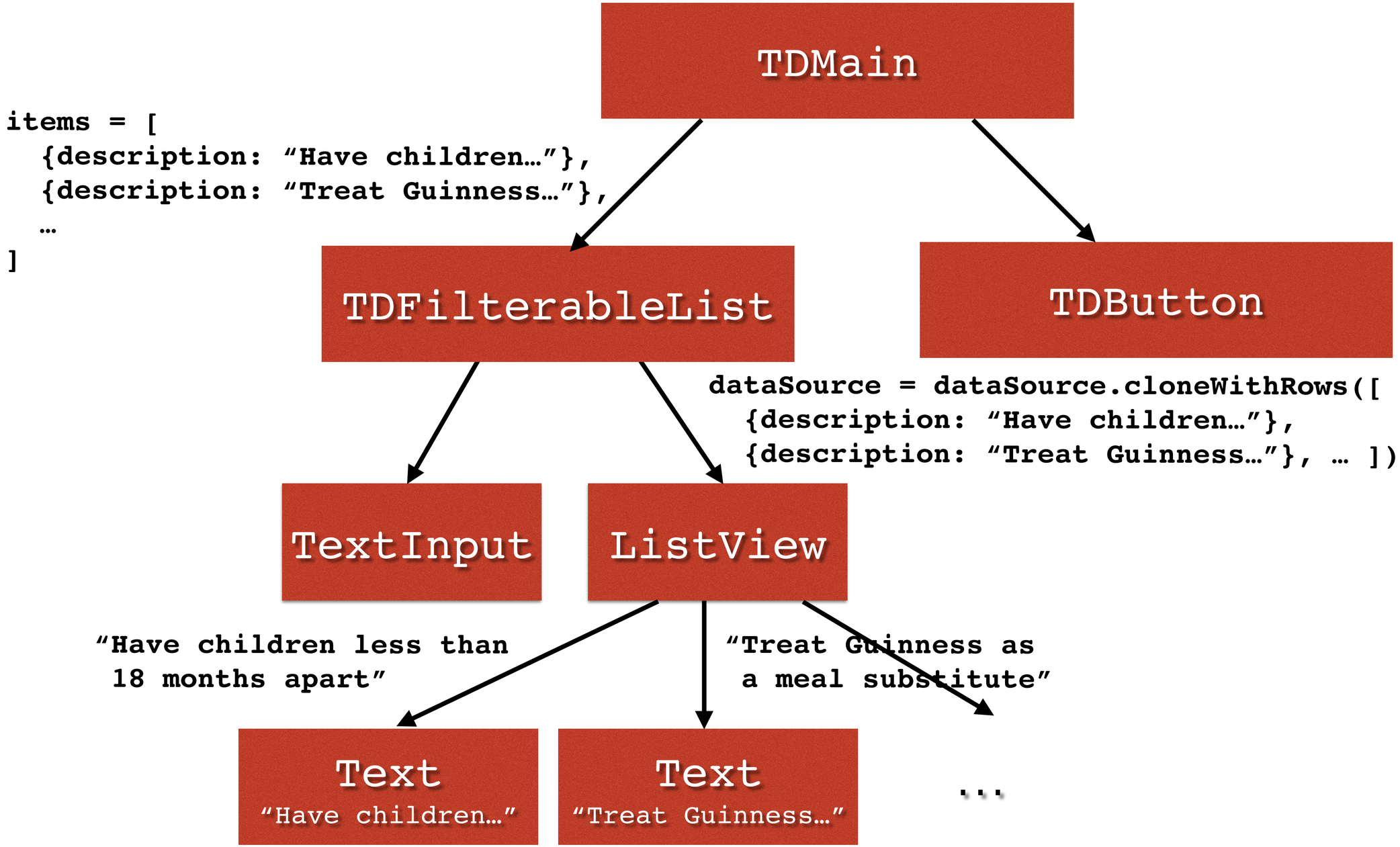
```
return (
  React.createElement(View, null,
    React.createElement(TDFilterableList, {items: items}),
    React.createElement(TDButton, null, “New")
  )
);
```

TDFilterableList

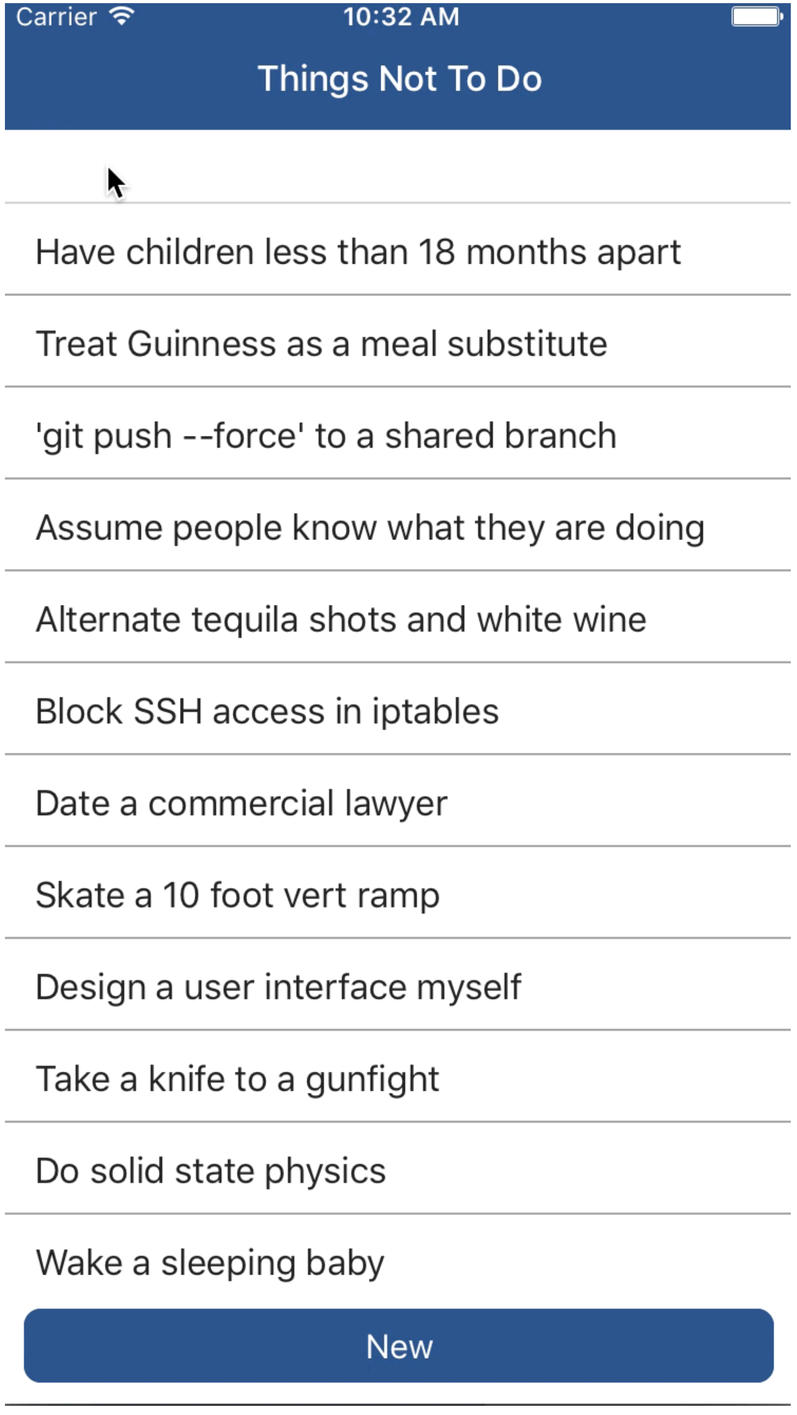


TDFilterableList

```
class TDFilterableList extends React.Component {
  render() {
    return (
      <View>
        <TextInput/>

        <ListView
          dataSource={dataSource.cloneWithRows(this.props.items)}
          renderRow={rowData => <Text>{rowData.description}</Text>}
        />
      </View>
    );
  }
}
```

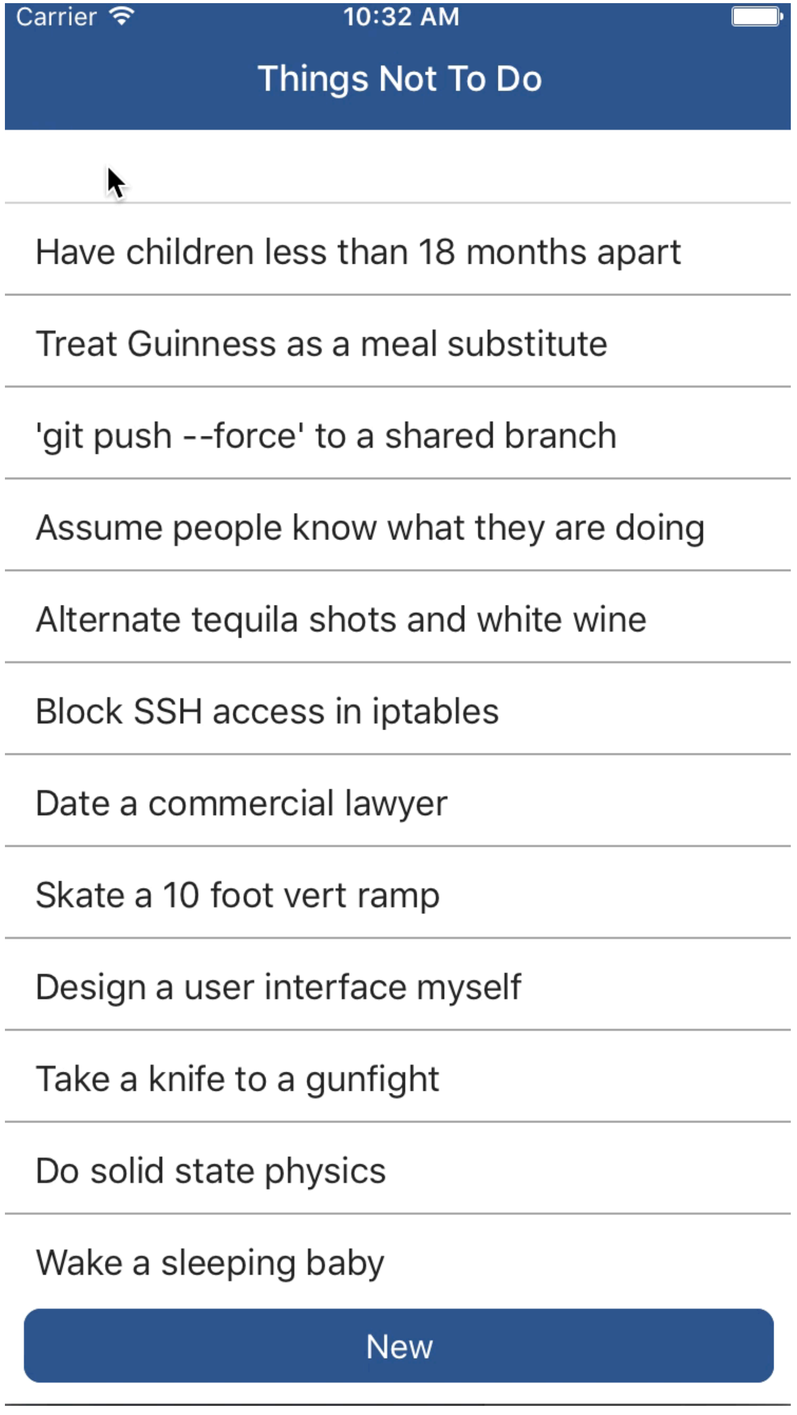

TDMain
items = [
{description: “Have children…”},
{description: “Treat Guinness…”},
…
]
TDFilterableList
TDButton
dataSource = dataSource.cloneWithRows([
{description: “Have children…”},
{description: “Treat Guinness…”}, …])
TextInput
ListView
“Have children less than
18 months apart”
“Treat Guinness as
a meal substitute”
Text
“Have children…”
Text
“Treat Guinness…”
...

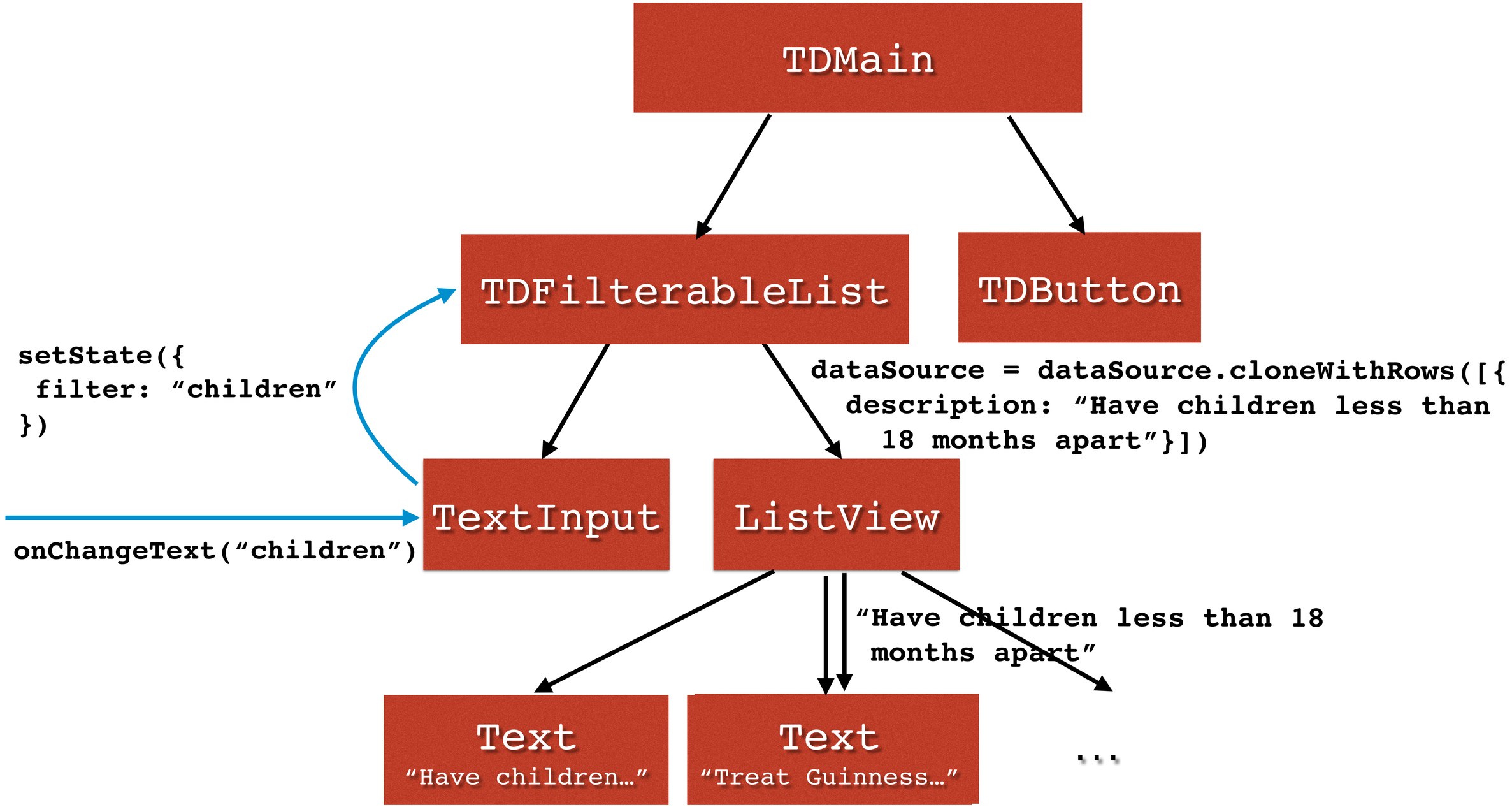
One-Way Data Flow


Carrier
10:32 AM
Things Not To Do
Have children less than 18 months apart
Treat Guinness as a meal substitute
'git push --force' to a shared branch
Assume people know what they are doing
Alternate tequila shots and white wine
Block SSH access in iptables
Date a commercial lawyer
Skate a 10 foot vert ramp
Design a user interface myself
Take a knife to a gunfight
Do solid state physics
Wake a sleeping baby
New

```
class TDFilterableList extends React.Component {
  constructor() {
    super();
    this.state = {filter: ''};
  }

  render() {
   return (
      <View>
        <TextInput
          onChangeText={text => this.setState({filter: text})}/>

        <ListView … />
      </View>
    );
  }
}
```

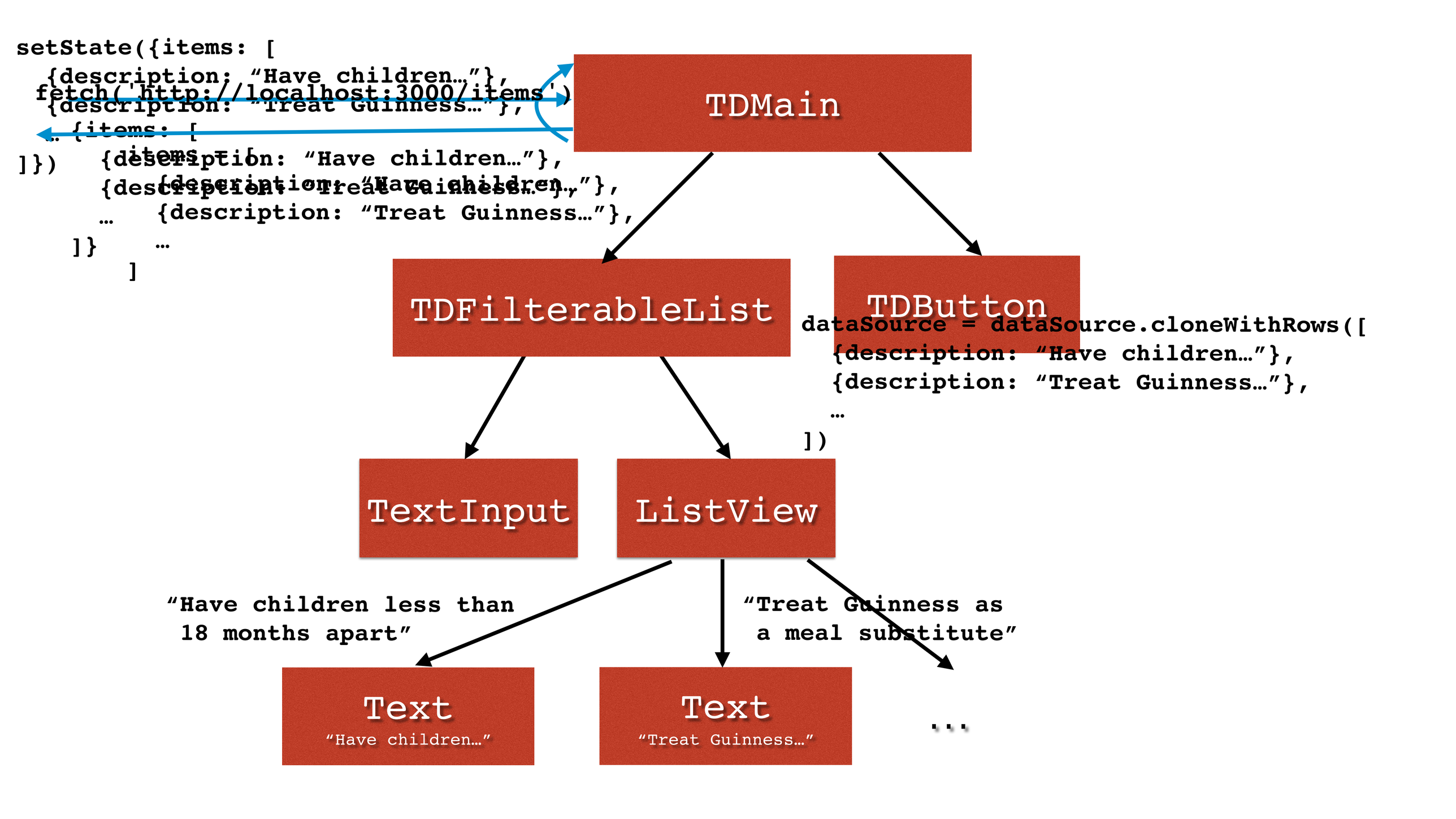
```
render() {
  var filteredItems = this.props.items.filter(item => {
    return item.description.includes(this.state.filter);
  });

  return (
    <View>
      <TextInput
        onChangeText={text => this.setState({filter: text})}/>
      <ListView
        dataSource={dataSource.cloneWithRows(filteredItems)}
        renderRow={rowData => <Text>{rowData.description}<Text/>}
      />
    </View>
  );
}
```


Carrier
10:32 AM
Things Not To Do
Have children less than 18 months apart
Treat Guinness as a meal substitute
'git push --force' to a shared branch
Assume people know what they are doing
Alternate tequila shots and white wine
Block SSH access in iptables
Date a commercial lawyer
Skate a 10 foot vert ramp
Design a user interface myself
Take a knife to a gunfight
Do solid state physics
Wake a sleeping baby
New


TDMain
TDFilterableList
TDButton
setState({
filter: “children”
})
dataSource = dataSource.cloneWithRows([{
description: “Have children less than
18 months apart”}])
TextInput
ListView
onChangeText(“children”)
“Have children less than 18
months apart”
Text
“Have children…”
Text
“Treat Guinness…”
...

TDMain

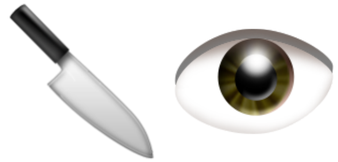
```
class TDMain extends React.Component {
  constructor() {
    super();
    this.state = {items: []};
  }

 render() {
    return (
      <View>
        <TDFilterableList items={this.state.items}/>
        <TDButton text="New"/>
      </View>
    );
  }

  componentDidMount() {
    fetch('http://localhost:3000/items').
      then(response => response.json()).
      then(json => {
        this.setState({items: JSON.parse(json).items})
      });
  }
}
```


setState({items: [
{description: “Have children…”},
{description: “Treat Guinness…”},
…
]})
fetch('http://localhost:3000/items')
{items: [
{description: “Have children…”},
{description: “Treat Guinness…”},
…
]}
TDMain
TDFilterableList
TDButton
dataSource = dataSource.cloneWithRows([
{description: “Have children…”},
{description: “Treat Guinness…”},
…
])
TextInput
ListView
“Have children less than 18 months apart”
“Treat Guinness as a meal substitute”
Text
“Have children…”
Text
“Treat Guinness…”
...

The React Programming Model

- Virtual View Hierarchies
- Declarative UIs
- One-way Data Flow

…but what if you don’t like JavaScript?



Doing it React-style in Swift

- ReactiveCocoa == awesome
- But how is Reactive UI formed?
- Incorporating React techniques

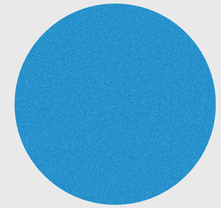
Time to Dive Deeper!

YOU WOULDN'T WRITE YOUR OWN SORT FUNCTION

ReactiveCocoa

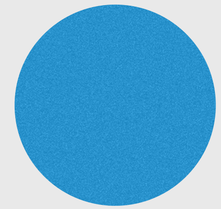
```
let signal: Signal<Marble, NoError>
```

signal



```
signal.filter { m in m.colour == .Purple }
```

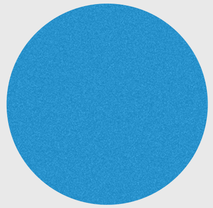
signal



output

```
signal.delay(1.5.seconds, onScheduler: mainQueueScheduler)
```

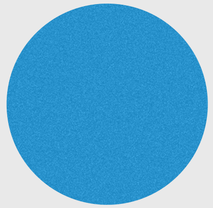
signal



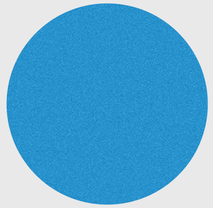
output

```
signal.skipRepeats { a, b in a.colour == b.colour }
```

signal

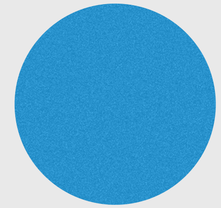


output

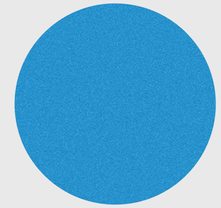


```
signal.throttle(2.seconds, onScheduler: mainQueueScheduler)
```

signal

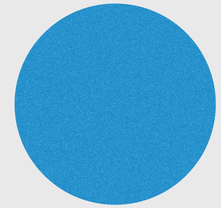


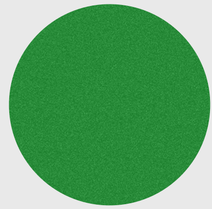
output



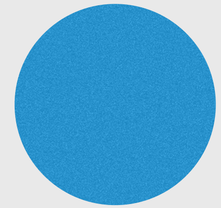
```
signals.flatten(.Merge)
```

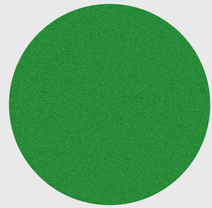
signals





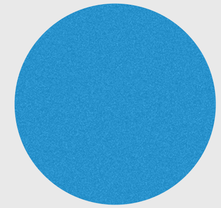
output



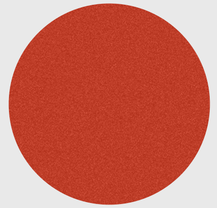


```
signal1.sampleOn(signal2)
```

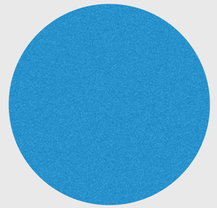
signal1



signal2

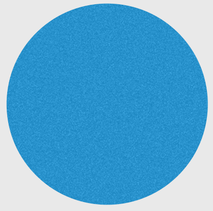


output

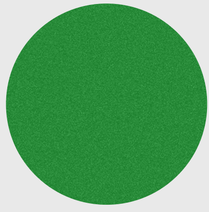


```
signal1.combineLatestWith(signal2)
```

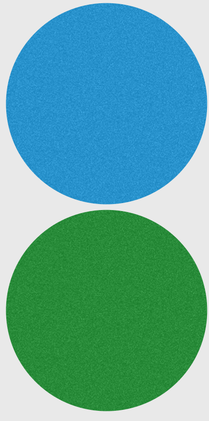
signal1

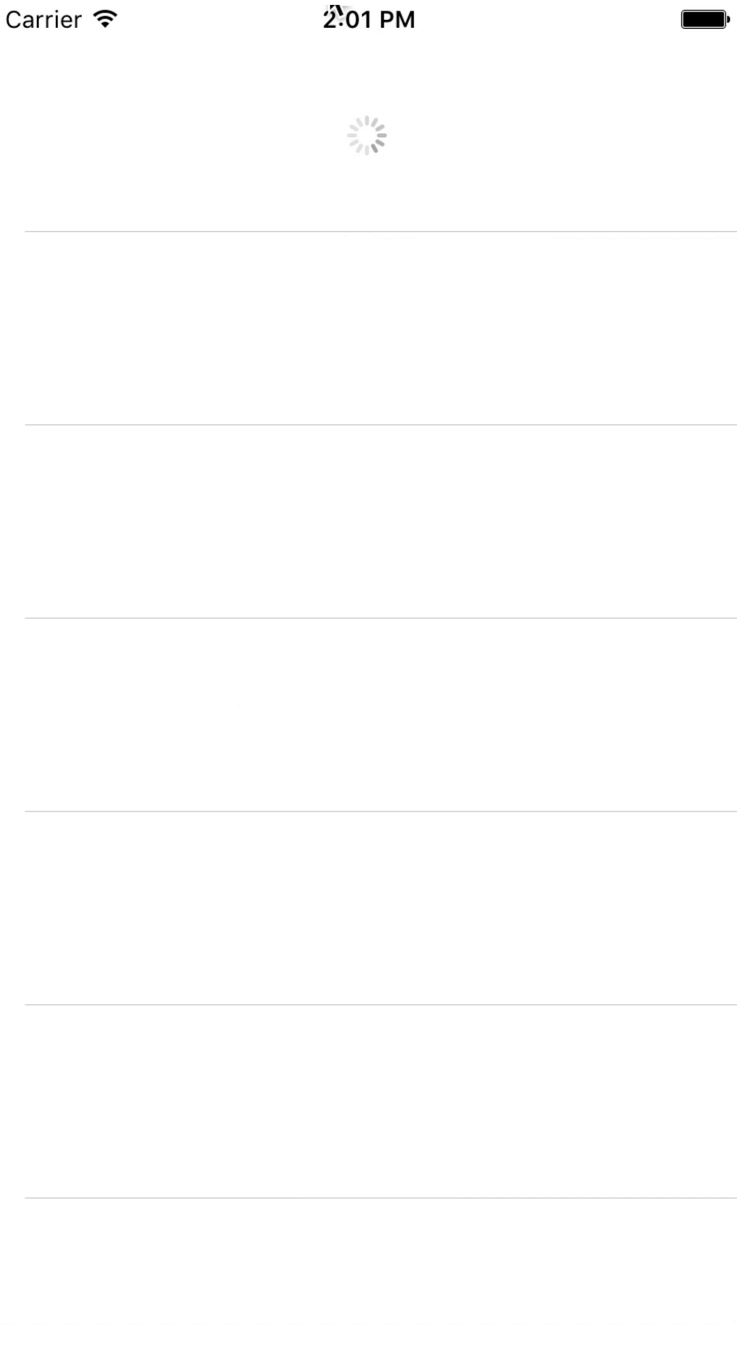


signal2



output





Comic Book Storefront

1. Download top 20 from API
2. Download thumbnails from API
3. Get prices from App Store
4. GOTO 1.

```
class ViewController: UITableViewController {

    var comics = [ComicBook]()
    var prices = [SKProduct]()
    var images = [Int: UIImage]()
```

```
override func viewDidLoad() {
    super.viewDidLoad()

    loadComics()
}

func loadComics(from: Int = 1) {
    StoreAPI.sharedAPI.getComics(from: from) { c in
        self.comics.appendContentsOf(c)
        self.tableView.reloadData()
        self.loadPrices(c.map { $0.productIdentifier })
    }
}

func loadPrices(ids: [String]) {
    StoreKit.sharedStoreKit.getProducts(ids) { p in
        self.prices.appendContentsOf(p.products)
        self.tableView.reloadData()
    }
}
```

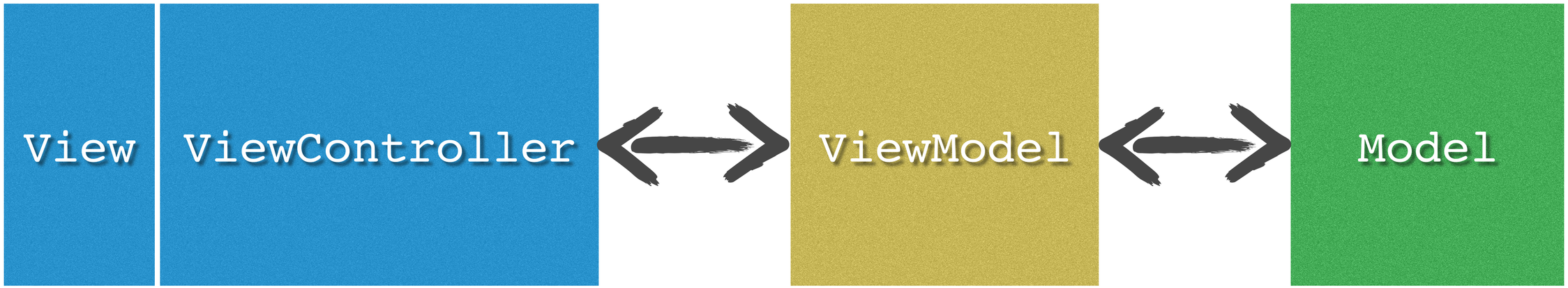
```
    override func tableView(tableView: UITableView,
cellForRowAtIndexPath indexPath: NSIndexPath) ->
UITableViewCell {
        let cell =
tableView.dequeueReusableCellWithIdentifier("StoreItemCell",
forIndexPath: indexPath) as! StoreItemCell

        let comic = comics[indexPath.row]
        cell.numberLabel.text = (indexPath.row +
1).description
        cell.titleLabel.text = comic.name
        cell.publisherLabel.text = comic.publisher

        if prices.count > indexPath.row {
            cell.downloadButton.hidden = false
            cell.downloadButton.state = .Buy
            cell.downloadButton.buyTitle = "$\
(prices[indexPath.row].price.doubleValue)"
```

```swift
class ViewController: UITableViewController, SKProductsRequestDelegate {

    var comics = [ComicBook]()
    var prices = [SKProduct]()
    var images = [Int: UIImage]()

    override func viewDidLoad() {
        super.viewDidLoad()

        loadComics()
    }

    func loadComics(from: Int = 1) {
        StoreAPI.sharedAPI.getComics(from: from) { c in
            self.comics.appendContentsOf(c)
            self.tableView.reloadData()
            self.loadPrices(c.map { $0.productIdentifier })
        }
    }

    func loadThumbnail(indexPath: NSIndexPath, url: NSURL) {
        StoreAPI.sharedAPI.downloadCover(url) { i in
            self.images[indexPath.row] = i
            self.tableView.reloadRowsAtIndexPaths([indexPath], withRowAnimation: .None)
        }
    }

    func loadPrices(productIdentifiers: [String]) {
        let req = SKProductsRequest(productIdentifiers: Set(productIdentifiers))
        req.delegate = self
        req.start()
    }

    func productsRequest(request: SKProductsRequest, didReceiveResponse response: SKProductsResponse) {
        self.prices.appendContentsOf(response.products)
        self.tableView.reloadData()
    }

    // MARK: UITableViewDataSource

    override func numberOfSectionsInTableView(tableView: UITableView) -> Int {
        return 1
    }

    override func tableView(tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        return comics.count + 1
    }

    override func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell {
        let cell = tableView.dequeueReusableCellWithIdentifier("StoreItemCell", forIndexPath: indexPath) as! StoreItemCell

        let comic = comics[indexPath.row]
        cell.numberLabel.text = (indexPath.row + 1).description
        cell.titleLabel.text = comic.name
        cell.publisherLabel.text = comic.publisher

        if prices.count > indexPath.row {
            cell.downloadButton.hidden = false
            cell.downloadButton.state = .Buy
            cell.downloadButton.buyTitle = "$\(prices[indexPath.row].price.doubleValue)"
        } else {
            cell.downloadButton.hidden = true
        }

        if let image = images[indexPath.row] {
            cell.coverImageView.image = image
        } else {
            cell.coverImageView.image = UIImage(named: "placeholder")
            loadThumbnail(indexPath, url: comic.coverURL)
        }

        return cell
    }

    override func tableView(tableView: UITableView, willDisplayCell cell: UITableViewCell, forRowAtIndexPath indexPath: NSIndexPath) {
        if indexPath.row > 0 && indexPath.row == comics.count {
            loadComics(comics.count + 1)
        }
    }
}
```

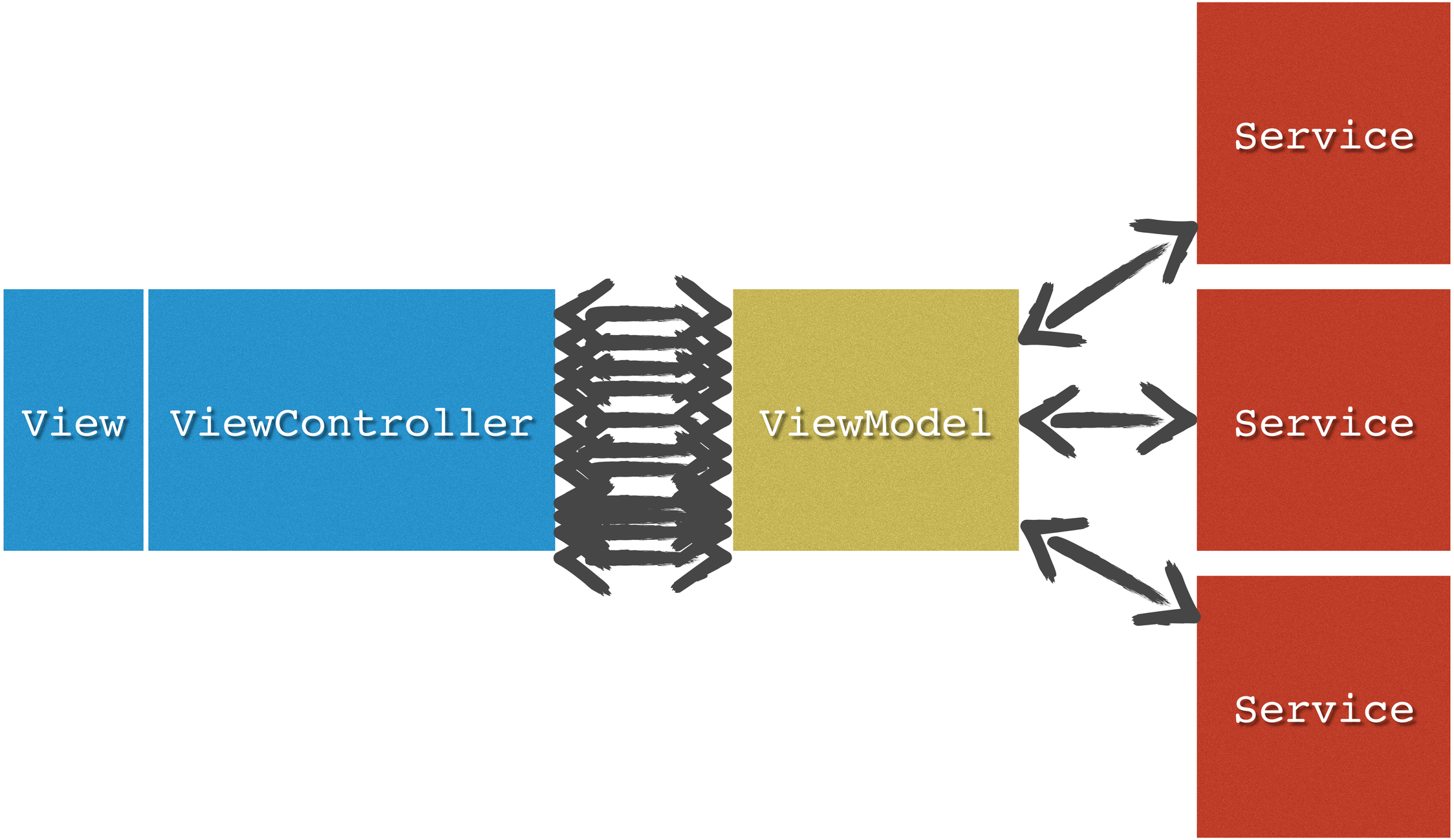
```
    func loadComics(from: Int = 1) {
        StoreAPI.sharedAPI.getComics(from: from) { c in
            self.comics.appendContentsOf(c)
            self.tableView.reloadData()
            self.loadPrices(c.map { $0.productIdentifier })
        }
    }

    func loadPrices(ids: [String]) {
        StoreKit.sharedStoreKit.getProducts(ids) { p in
            self.prices.appendContentsOf(p.products)
            self.tableView.reloadData()
        }
    }

    func loadThumbnail(indexPath: NSIndexPath, url: NSURL) {
        StoreAPI.sharedAPI.downloadCover(url) { i in
            self.images[indexPath.row] = i
            self.tableView.reloadRowsAtIndexPaths([indexPath],
withRowAnimation: .None)
        }
    }
```

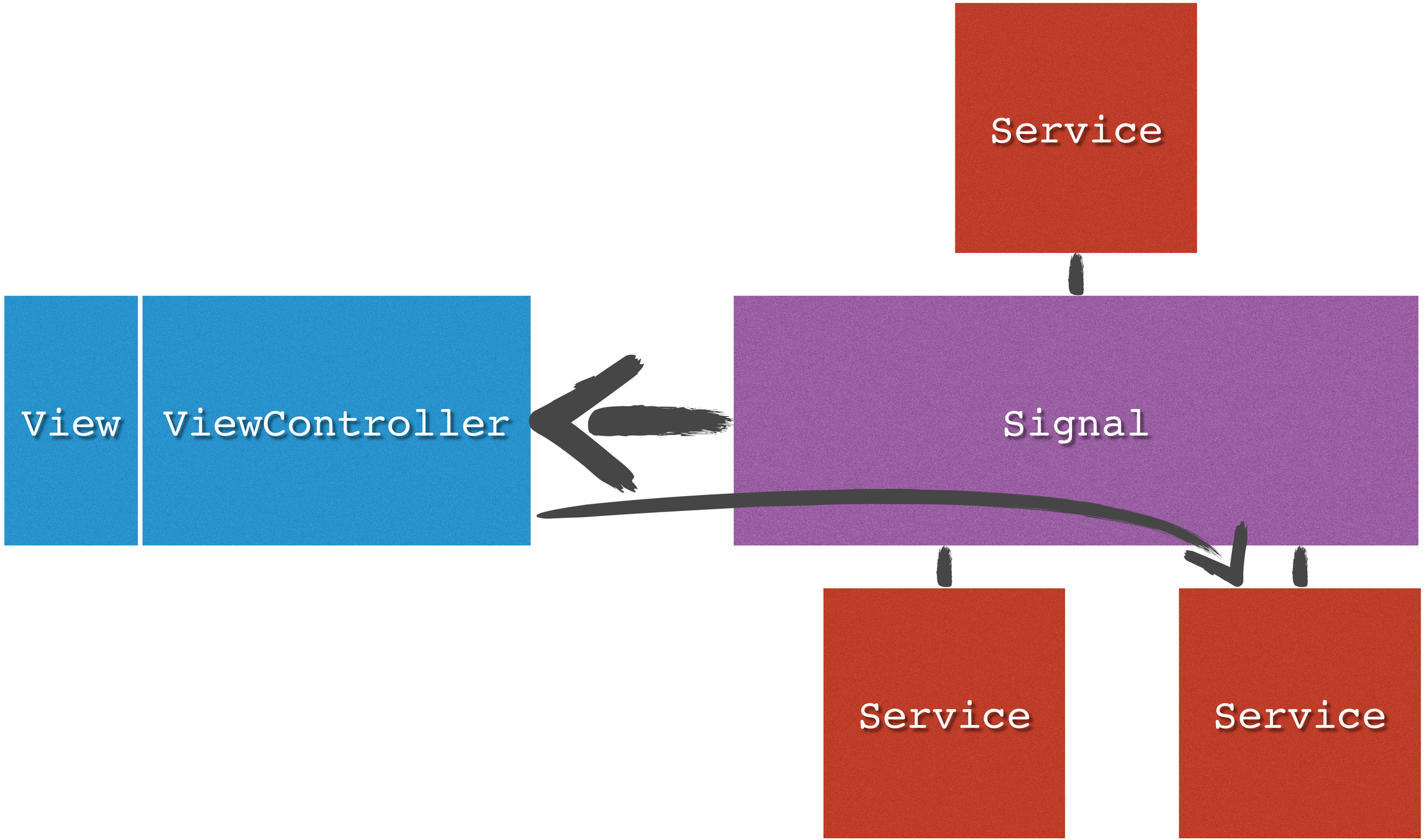
```
    func loadComics(from: Int = 1) {
        StoreAPI.sharedAPI.getComics(from: from).startWithNext { c in
            self.comics.appendContentsOf(c)
            self.tableView.reloadData()
            self.loadPrices(c.map { $0.productIdentifier })
        }
    }

    func loadPrices(ids: [String]) {
        StoreKit.sharedStoreKit.getProducts(ids).startWithNext { p in
            self.prices.appendContentsOf(p.products)
            self.tableView.reloadData()
        }
    }

    func loadThumbnail(indexPath: NSIndexPath, url: NSURL) {
        StoreAPI.sharedAPI.downloadCover(url).startWithNext { i in
            self.images[indexPath.row] = i
            self.tableView.reloadRowsAtIndexPaths([indexPath],
withRowAnimation: .None)
        }
    }
```

View Model


View
ViewController
ViewModel
Model

```
class ComicBookViewModel {
    let number = MutableProperty<Int>(0)
    let name = MutableProperty<String>("")
    let publisher = MutableProperty<String>("")
    let coverImage = MutableProperty<UIImage>(UIImage(named:
"placeholder")!)
    let price = MutableProperty<Double>(0)
    let downloadState =
MutableProperty<PKDownloadButtonState>(.Buy)
}
```


View
ViewController
ViewModel
Service
Service
Service


Service
View
ViewController
Signal
Service
Service

The Uber-Signal

[item, item, item]

flatMap()

[$2.99, $3.99, $0.99]

flatMap()

scan()

[item: ]

[item:  , item: ]

[item:  , item:  , item: ]

combineLatest()

([item, item, item], [], [:])

([item, item, item], [], [item: ])

([item, item, item], [$2.99, $3.99, $0.99], [item: ])

([item, item, item], [$2.99, $3.99, $0.99], [item:  , item: ])

([item, item, item], [$2.99, $3.99, $0.99], [item:  , item:  , item: ])

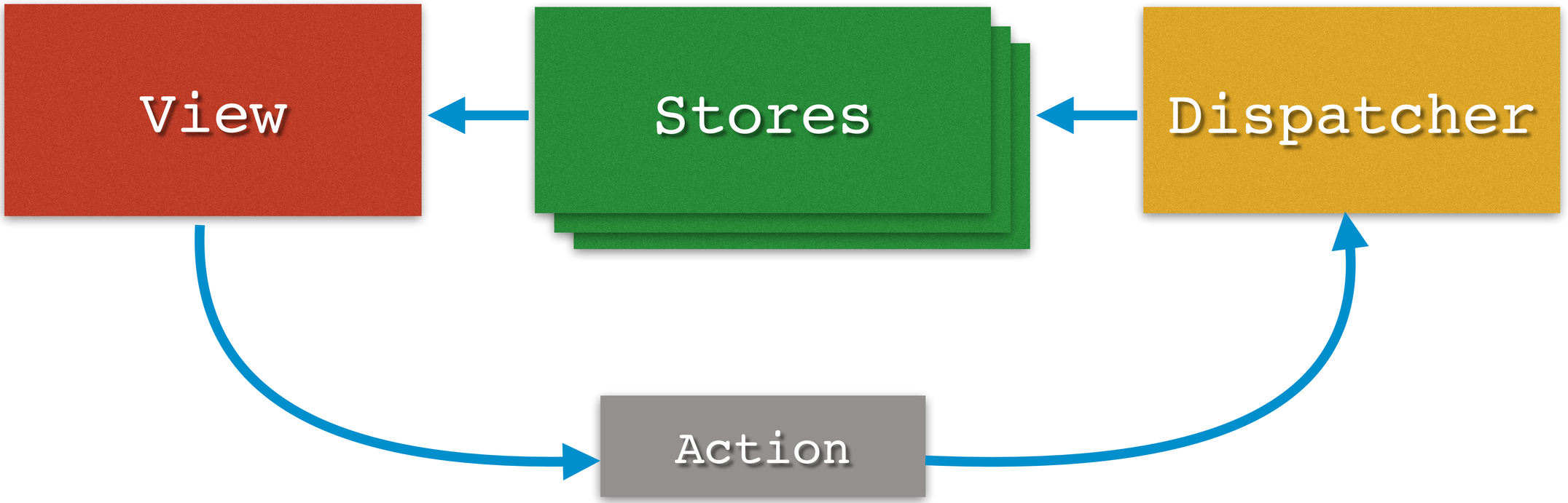
map()

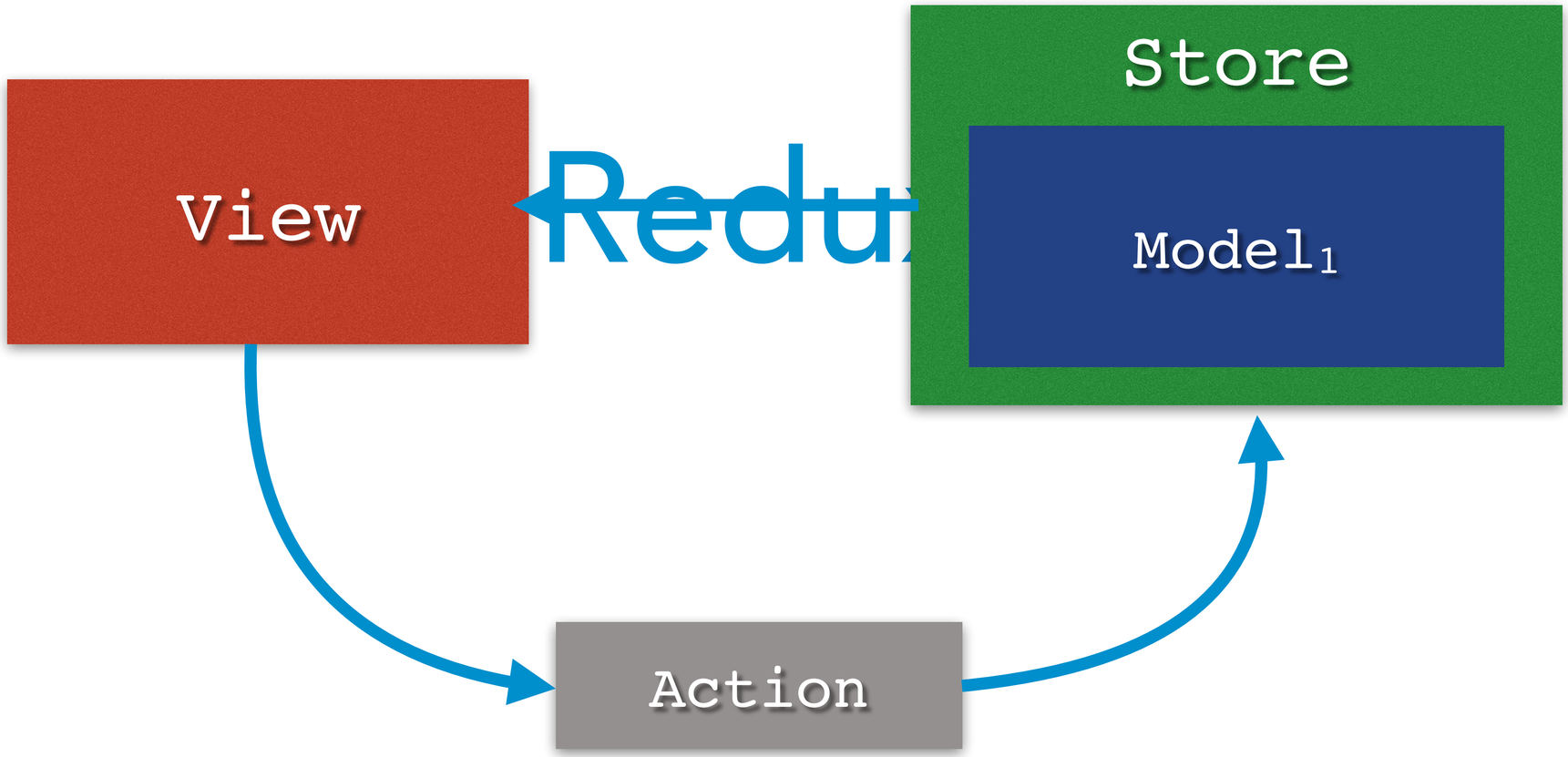
```
[(item, nil,  ), (item, nil,  ), (item, nil,  )]

[(item, nil,  ), (item, nil,  ), (item, nil,  )]

[(item, $2.99,  ), (item, $3.99,  ), (item, $0.99,  )]

[(item, $2.99,  ), (item, $3.99,  ), (item, $0.99,  )]

[(item, $2.99,  ), (item, $3.99,  ), (item, $0.99,  )]
```

Demonstration

Other things to look at

```
func renderApp(component: Component<Int>, count: Int) -> Element {
    return View()
        .justification(.Center)
        .childAlignment(.Center)
        .direction(.Column)
        .children([
            Label("You've clicked \(count) times!"),
            Button(title: "Click me!", action: {
                component.updateState { $0 + 1 }
            })
                .margin(Edges(uniform: 10))
                .width(100),
            ])
}
```


View
Stores
Dispatcher
Action


Store
View
Redux
Model1
Action

Elm

Take-home Messages

- Alternatives to traditional MVC
- Embrace functional techniques
- Get started today

Thanks

@BenTeese

@FakeSamRitchie