The Shift Towards Functional Reactive Programming

WHAT SHOULD MY TEAM DO?

Nick Van Weerdenburg

CEO, Rangle.io

RANGLE.IO

REWRITING THE WEB

Agenda

- What is The Shift
- Why is The Shift Happening?
- A Tour of the Shift in Progress
- Can I Make the Shift?
- Should I Make The Shift?

Classic MVC or FRP?

YES

What is the Shift?

What is FRP?

Not That Much

- Functions
- Streams
- Reactive Patterns of Update
- But...

What is It In Practice?

- A new way of thinking
- A new way of composing applications
- A new set of tools
- A profound sense of discomfort
- A agenda pushed by a self-selected group of enthusiasts
- A whole new set of ways to crash and burn

A Quick Tour

Reactive Extensions for JavaScript (RxJS)

- a reactive streams library for asynchronous data
- based on observables
- “promises vs. observables” debate
- integrating observables into React has been challenging

Why is the Shift Happening?

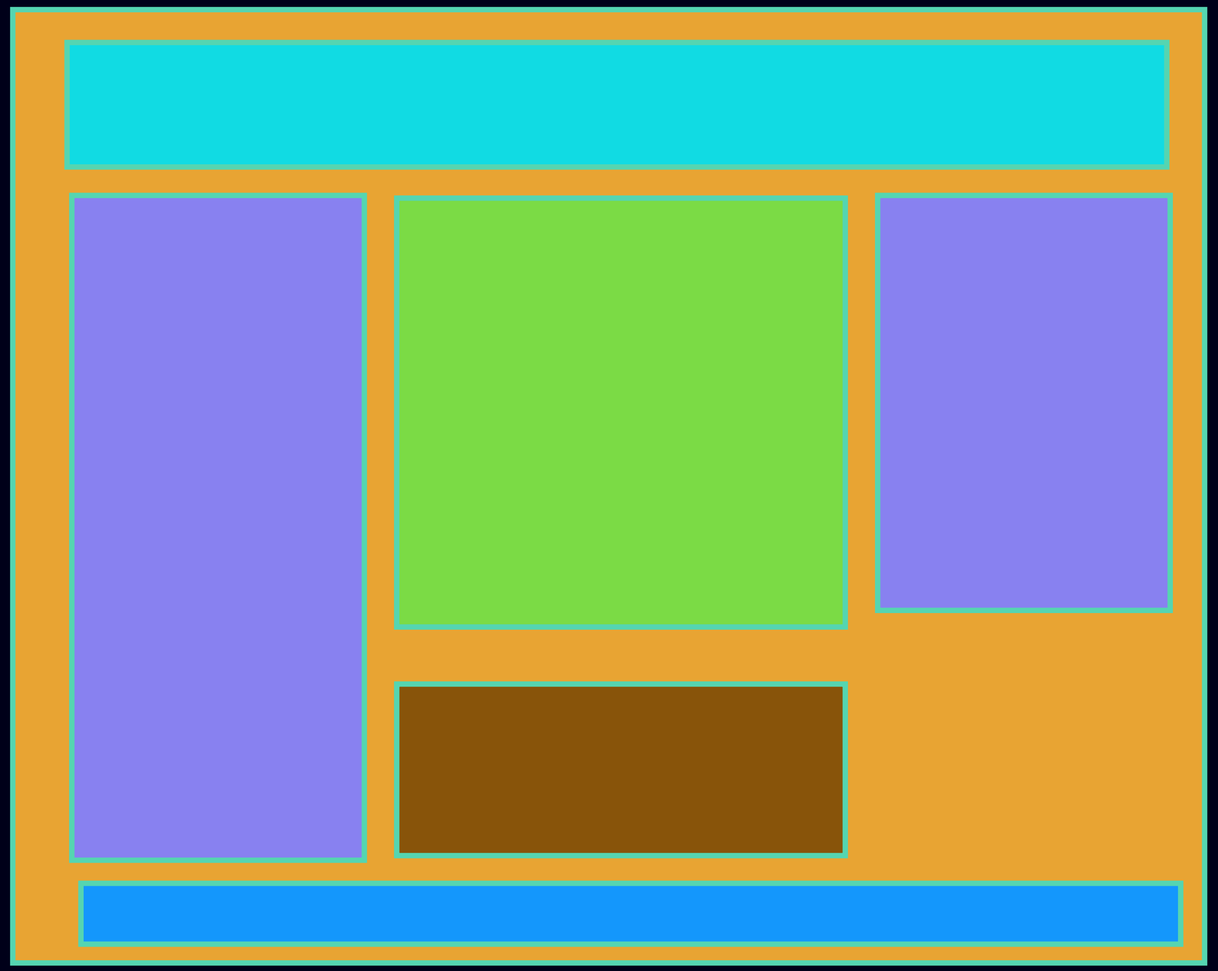
What’s Driving the Shift?

- asynchronous data and promise / callback architectures
- functional programming and immutable.js
- **more complex UX with HTML5 applications**
- **The clean code revolution**
- **The “reasoning about code” test of clean code**
- **The pressures of continuous delivery**
- **Ten years of a “stable” paradigm and true believers in the cost of technical debt**

shifter - UX

CRUD Application?

Or Crud Component



shifter - clean code

shifter - state oppression

Why

- Reason about state
- Maintain a state you can reason about

Global Application State

GAS is Good

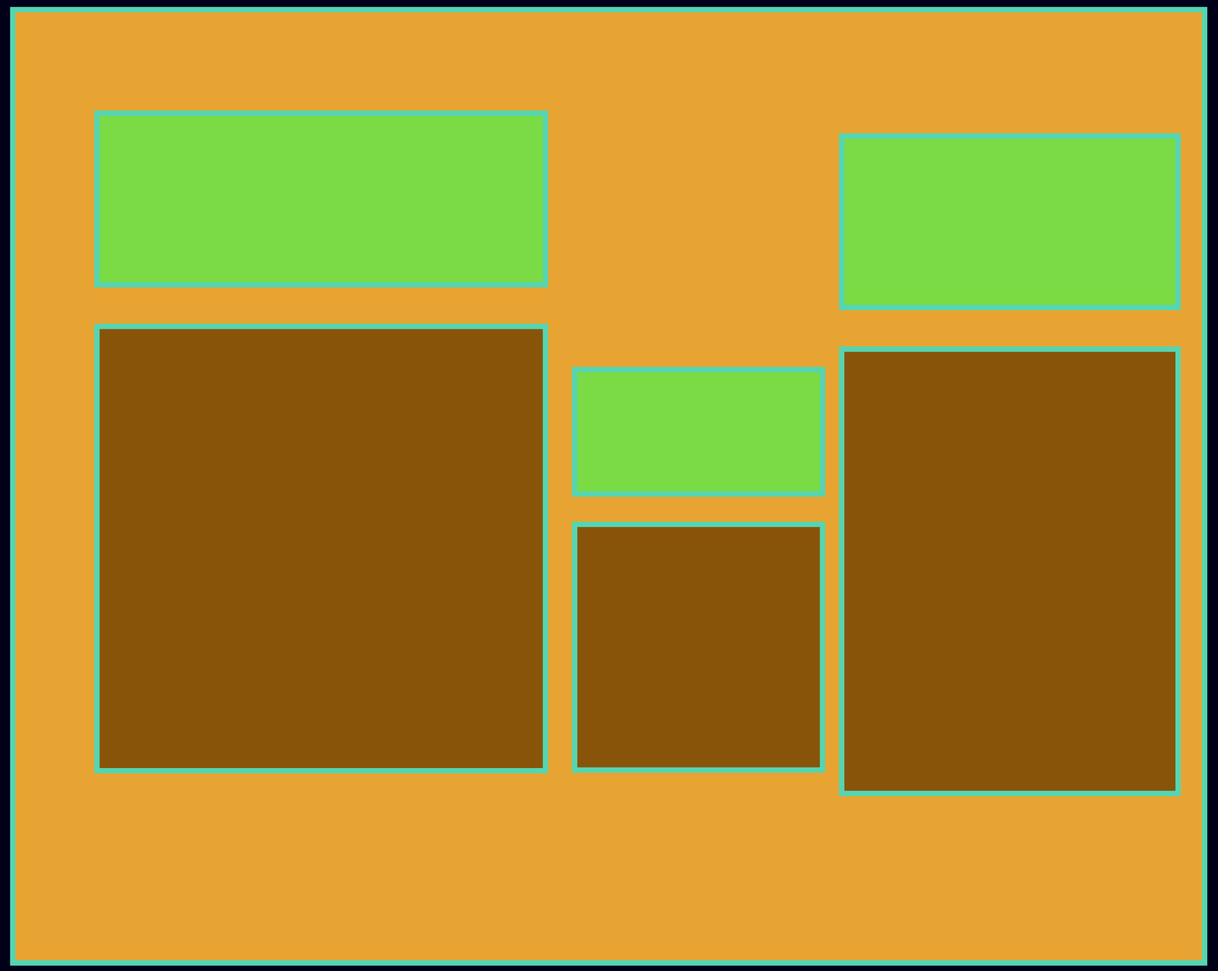
Client Application State

CAS is Good!

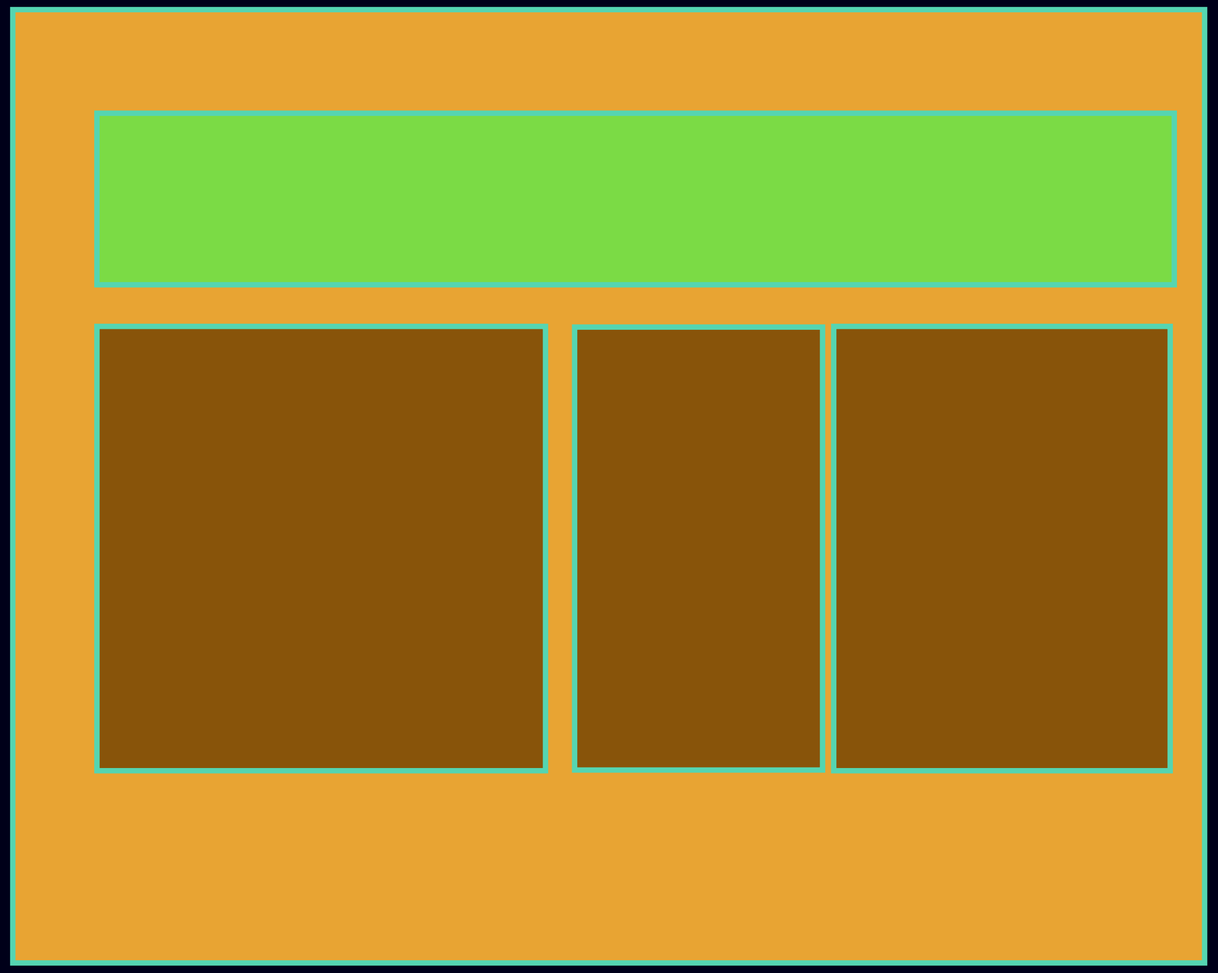
Why Not?

- Designing / Managing a global application state can be complicated
- How does it evolve over time?
- State decay = ease of extension now, but increasing cost per feature in the future

Multiple Application States



Global Appliction States



Why Global State?

- Force updates to state model as application evolves
  - Less technical debt
  - Maintain ability to reason about code
- Functional Programming enables testing and refactoring
- Reduce Your Technical Debt Run Rate

60%

80%

10%

What’s Your Technical Debt Run Rate

- i.e. how much of your budget is going to technical debt?
- what’s the NPV cost of this?
- please, please, please do the math
- and make your decisions based on that

shifter - Functions

Increasing Demands for Functions

- JavaScript
- Immutable.js
- React
- RxJS
- Redux

shifter - Streams

Distributed Reactive

- RxJS
- Bacon
- Falcor
- GraphQL
- Meteor

A Tour of the Shift in Progress

Angular or React?

This is the Wrong Question

REST API or Streams?

Controllers or Components?

Global Application State?

What Are We After

- Taming complexity.

A Typical Complex SPA

- Lots of parts.
- Everything is connected to everything else.
- Changing anything breaks something somewhere.
- This can be achieved with any framework.

Best Solutions Known as of 2015

- Component-based UI.
- A “stateless” approach deeper in the stack.

Component-Based Architecture

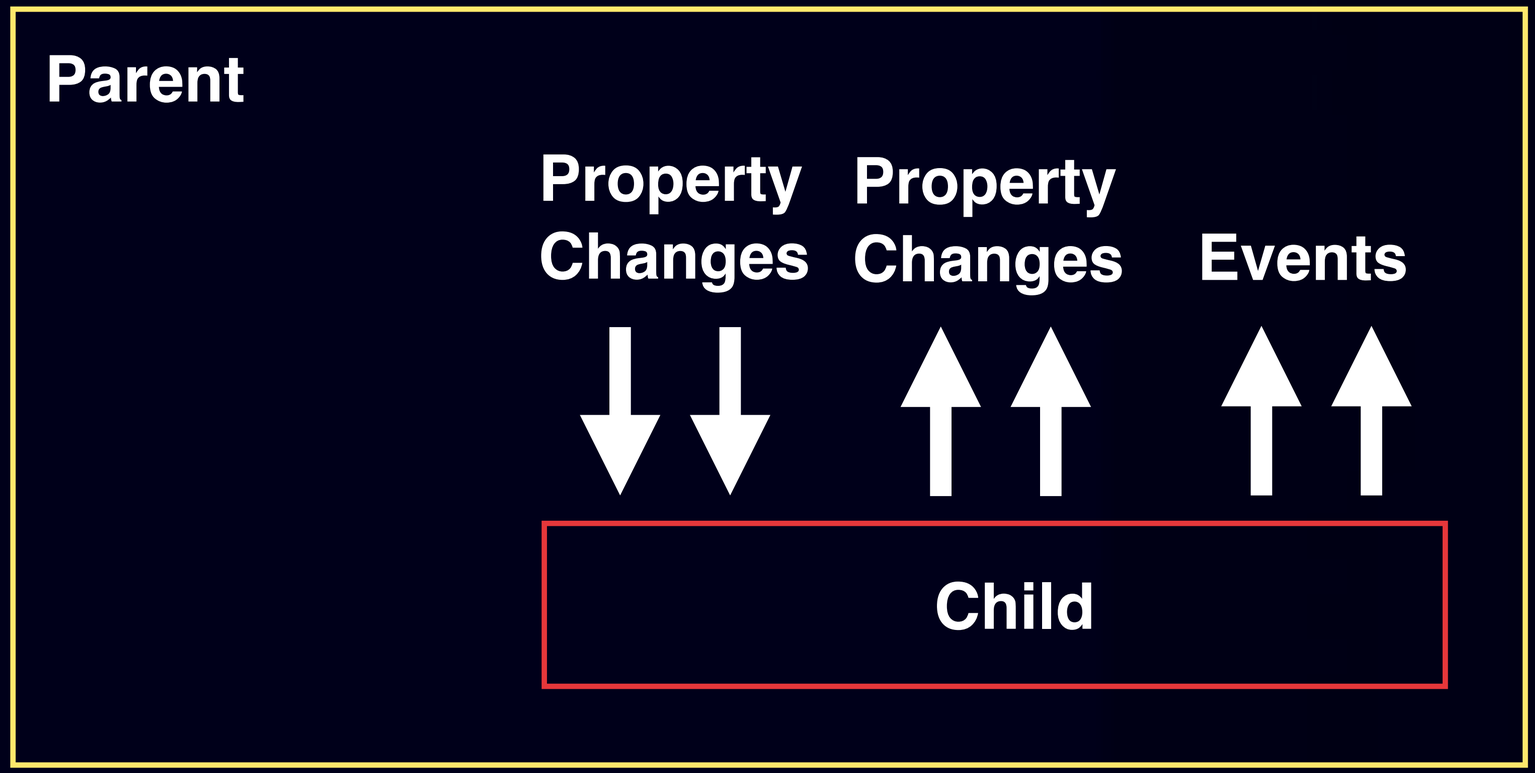
UI Components

- Build the UI layer of your application as a collection of UI components with well-defined interfaces.
- Make most of those components agnostic of the app.
- This helps with code reuse.
- This helps with testing.
- This makes it easy to reason about your code.

Component Typology

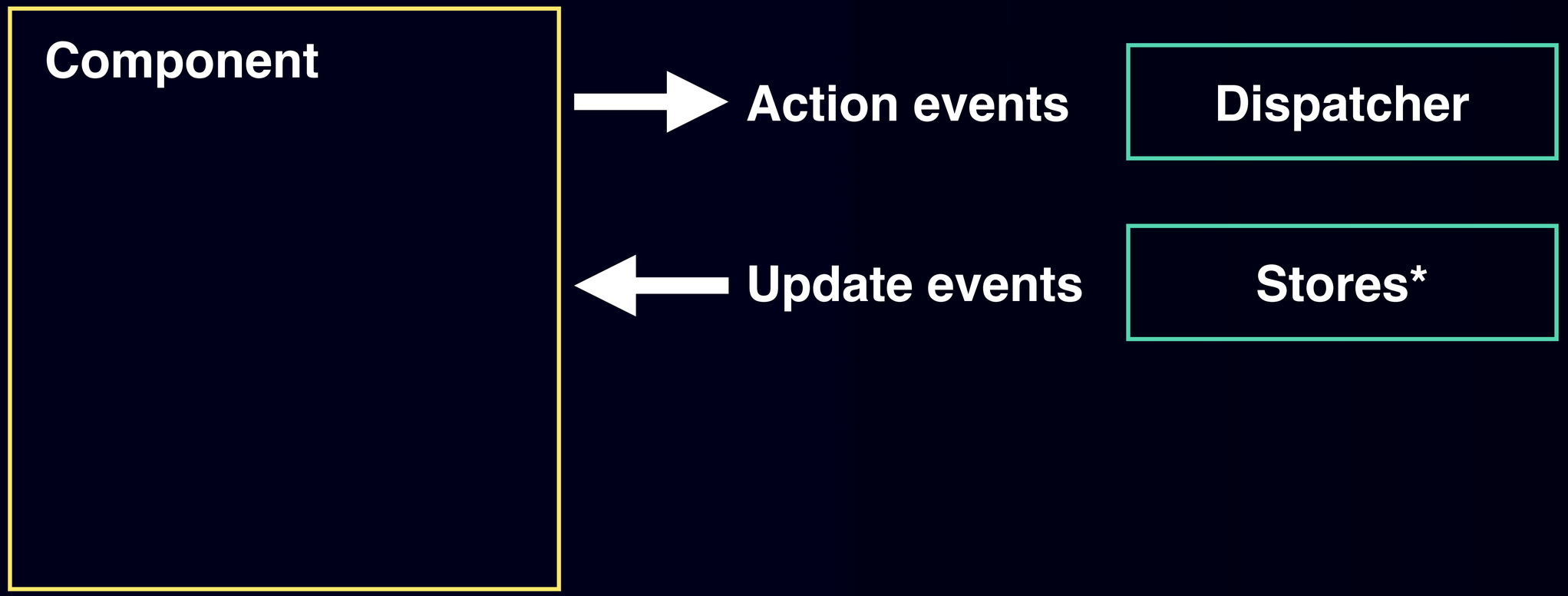
- “Dumb” components: not aware of your app.
- “Smart” components: hook into the rest of your app.
- “Smartish” components: in between.

Dumb Components



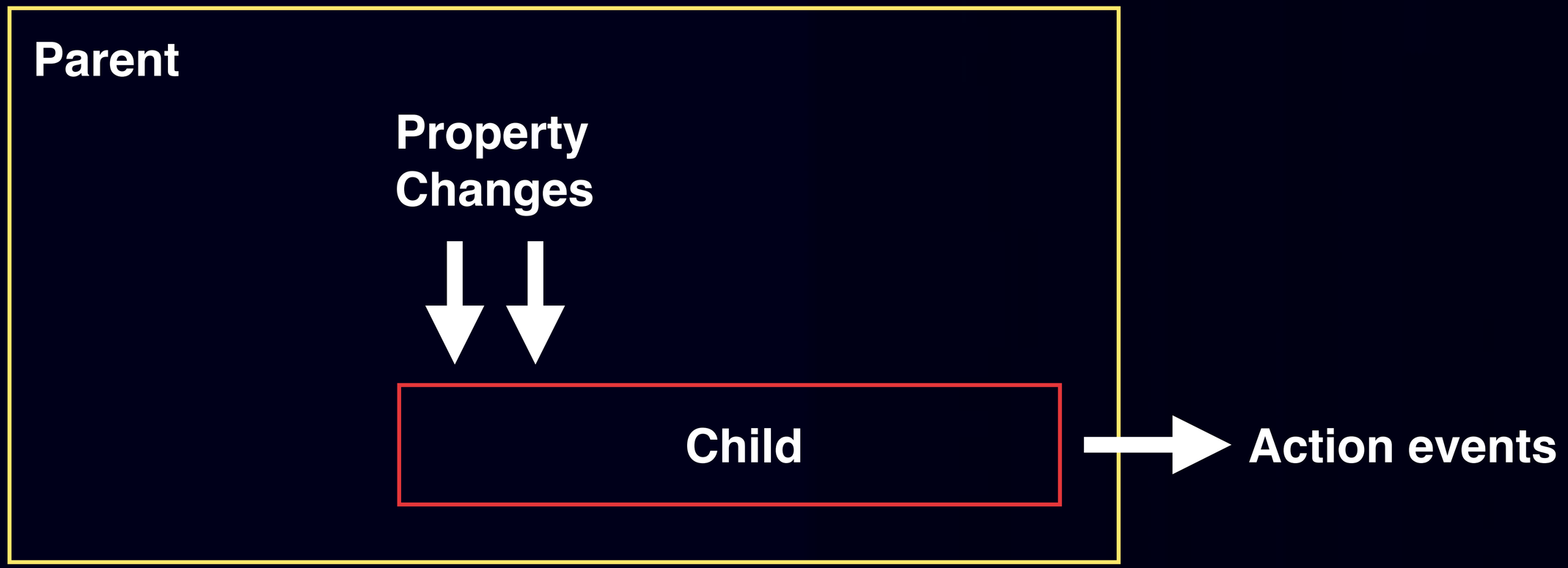
☛ Maximize your use of “dumb” components.

Smart Components



☛ Use “smart” components sparingly.

Smartish Components



Components in Angular and React

- They've been in Angular all along: isolated directives.
- They were never really loved.
- React brought components into the limelight.
- The React community has really helped clarify our thinking about components.
- But again, you can have components in Angular 1.x.
- And Angular 2 is really putting them at the center.

Stateless Architecture

Why Stateless is Good

- A lot of us were raised on OOP. Objects are stateful.
- The effect of calling a method depends on the arguments and the object's state.
- Very painful to test.
- Very hard to understand.
- Aims to mimic the real world.
- But why replicate the unnecessary complexity?

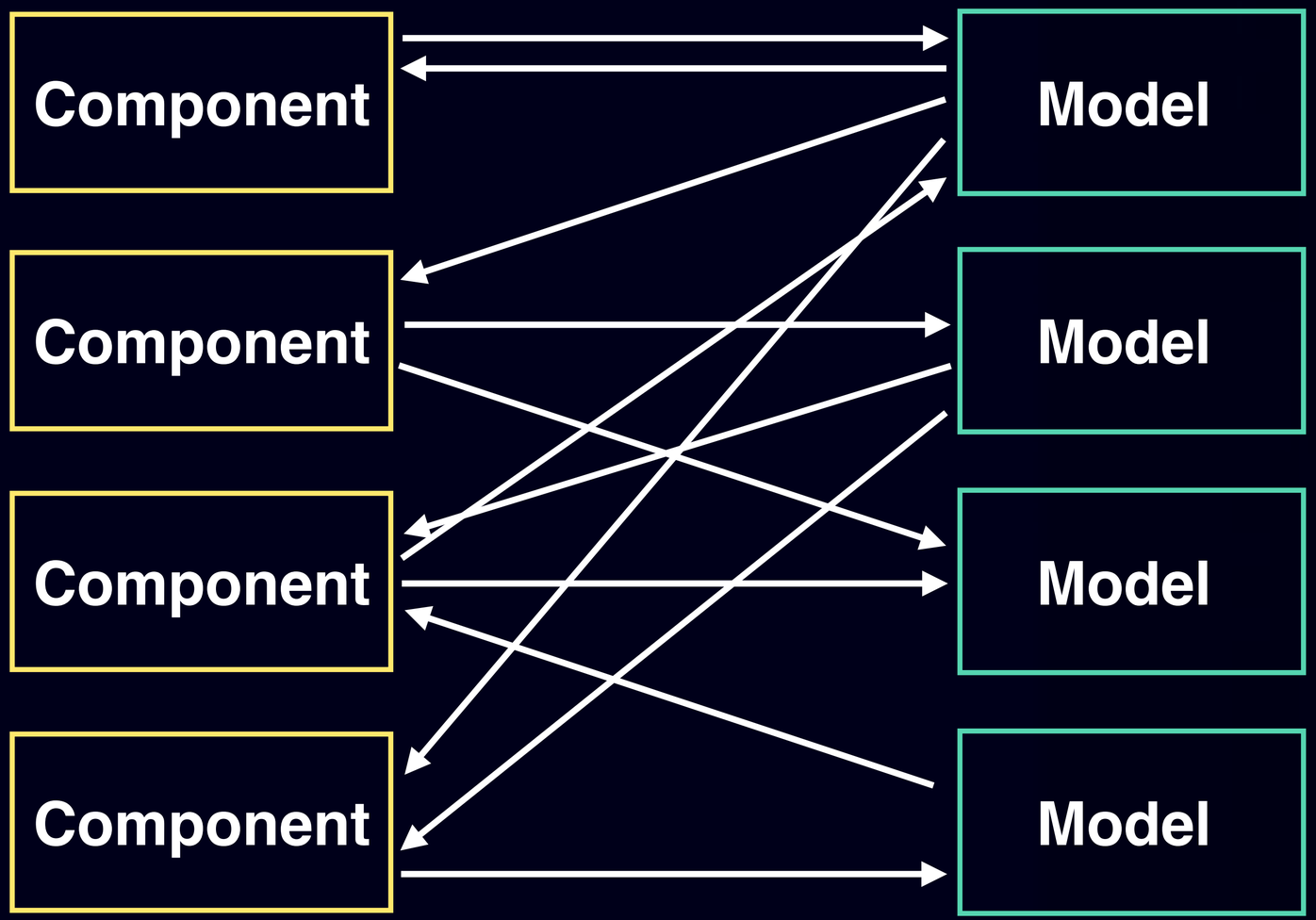
Alternative: Pure Functions

- The output of a pure function depends only on inputs.
- Pure functions have no side-effects.
- Pure functions have no state.
- Much easier to understand.
- Much easier to test.
- Very easy to build through composition.

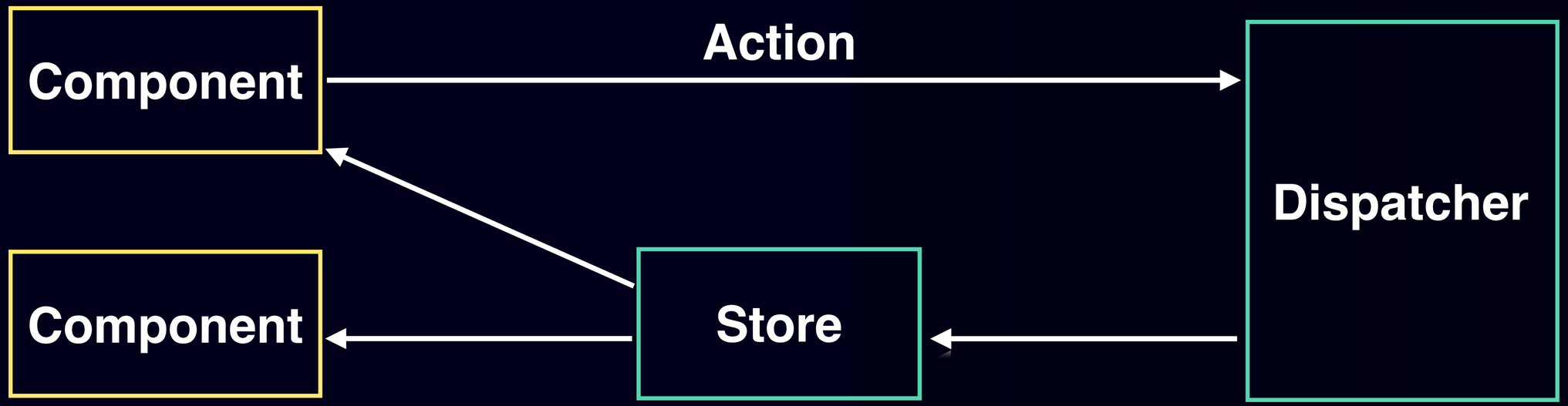
React and Functional DOM

- **React's render() method can be almost like a pure function.**
- **I.e., DOM is not "manipulated", but rather "calculated" from props.** (Component's have "state", but its use must be limited and you can't modify it directly.)
- **This is attractive, though requires some doublethink.**
- **In contrast, Angular's components are "stateful" in a normal OOP style by default. (But you can emulate React's style.)**
- **But what about your app's *data*?**

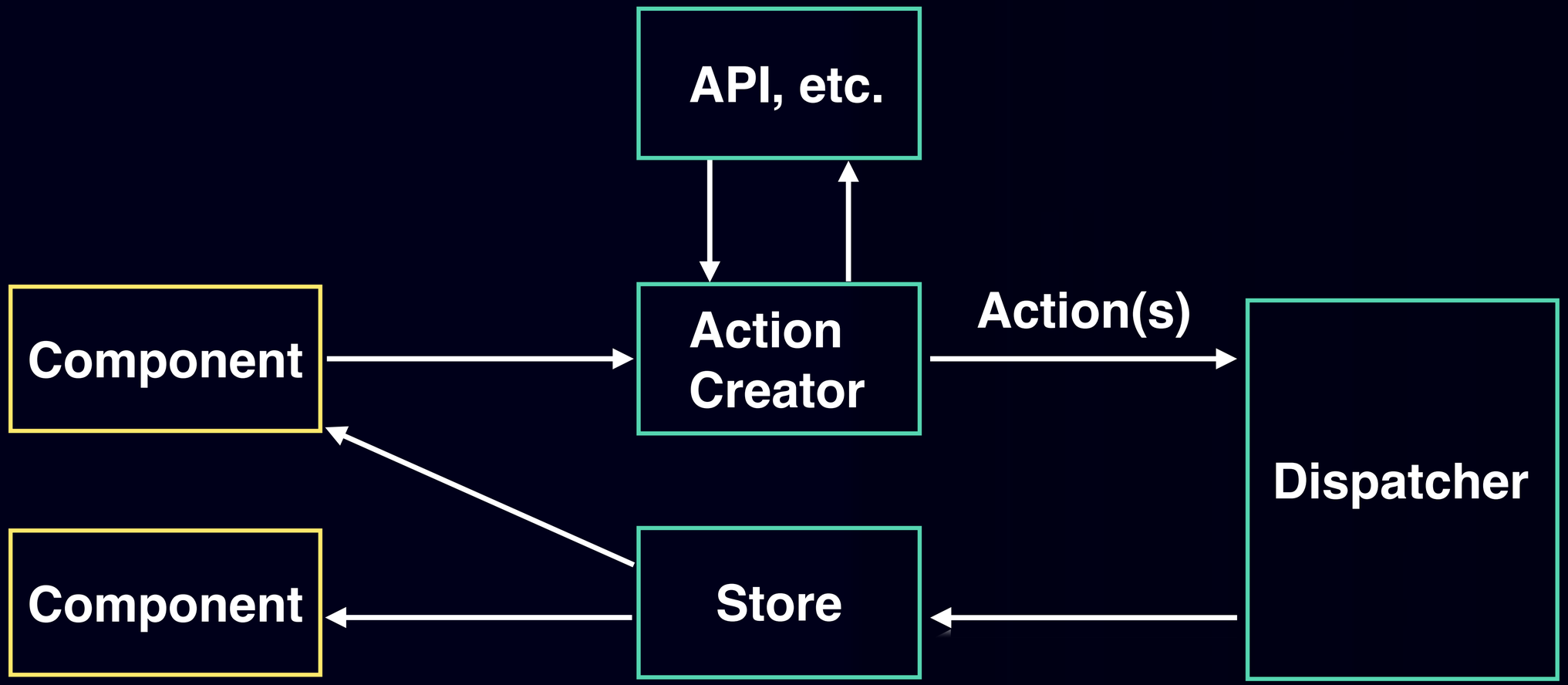
How State Really Becomes Painful



Unidirectional Data Flow with Flux



Unidirectional Data Flow with Flux



Data as Streams

- Create a stream.
- Push with: `myStream.onNext(data);`
- Share the stream.
- Others subscribe with a handler:
  `someStream.subscribe(handlerFunction);`
- Your components and stores are still stateful, but inputs are better controlled.
- Extra benefit: stream operations.

Misbehaving Subscribers

- Every subscriber gets a reference to the emitted object.
- What if one of the receivers decides to modify it?

Immutables

- Immutable data structures store data without introducing “state”.
- Object.freeze() - shallow.
- “Seamless Immutable”: frozen all the way down.
- But what happens when you want to modify your data?

Derived Immutables

- **You can’t change immutable objects. But you need to.**
- **So, you derive new ones.** I.e., make a new immutable that is different in a specified way, without altering the original.
- **“Immutable.JS” is the library to use.**

```
var newData = data.setIn(
  ['foo', 'bar', 'baz'],
  42
);
```

- **This is a key building block for stateless architecture.**

Can We Get More Stateless?

- Redux: https://github.com/rackt/redux
- Key concept: reducing actions.

Action Reducers

- **Basic** `reduce()`:

```
[1,2,3,4,5].reduce(
  (value, state) => state + value,
  0
);
```

- **In Redux your app's state is a reduction over the history of actions.**

An Example

```
export const notes = createReducer({
  [UPDATE_NOTE_CONTENT]: (state, action) => state.setIn(
    ['byId', action.payload.noteId, 'html'],
    action.payload.newContent
  ),
  [SOME_OTHER_ACTION]: (state, action) => {
    // do something else or just
    return state;
  }),
});
```

Advantages

- Each reducer is a pure function.
- Reducers are independent of each other.
- History can be rolled back and replayed.
- Reducers can stay synchronous.

Handling Asyncronous Actions

- Call an action creator to initiate a request.
- Action creator emits an action to inform everyone that a request was made.
- Action creator emits an action to inform everyone that a request was completed. (Or when it fails.)
- Push details of making a request into a module.

An Example, Part 1

```
function makeFetchNotesAction(status, data) {
  return {
    type: FETCH_NOTES,
    payload: {
      status: status,
      data: data
    }
  };
}
```

An Example, Part 1

```
export function fetchNotes() {
  return dispatch => {
    dispatch(makeFetchNotesAction('request'));
    return fetchData()
      .then(json => dispatch(makeFetchNotesAction('success', json)))
      .catch(error => dispatch(makeFetchNotesAction('error', error)));
  };
}
```

Can I Make The Shift?

Waves of Disruption

- Structured Programming
- Waterfall Development
- Agile
- Object Oriented Programming
- Failed Incursions of Functional Programming from academia
- Ruby on Rails
- Angular 1.x
- React

Key Aspects of the Modern Stack

- Components
- Predictable State Containers
- A Move to Global Application State
- Reactive, Functional with Global Application State
- Row, Row, Row Your Boat, Gently Down the…

Should I Make The Shift

Key Influences on Application Architecture

	State	Components	Asynchronous	Reason
Simple Forms	low	medium	low	low
Complex Forms	medium	medium	medium	low
Dashboards	medium	high	medium	medium
Graphical Interactions	high	high	medium	high
Reporting	low	medium	low	medium
Size	medium	high	medium	high

MVC or FRP?

Recommendations

- Start with functions, not streams
- Embrace global application state
- Redux, Not Flux; Components, Not Specific Frameworks
- Get Your Reference architecture
  - Fred Brooks- Prepare to Throw One Away
  - Ask me about Java, Rails, Angular 1.x Horror Stories and Scars. Get help, get your reference architecture.

Unaswered Questions

- Learning to Build with FRP
- FRP Architecture Decay- with scale
- FRP Architecture Decay- with new teams
- React + Observables - How to Do It? (don’t care)

THANK YOU!

Nick Van Weerdenburg
CEO, Rangle.io

@n1cholasv

rangle

RANGLE.IO

REWRITING THE WEB