React Native

Shan-Hung Wu & DataLab
CS, NTHU

Outline

- Hello React Native
  - How it works?
  - Components, props, and states
  - Styling
  - Event handling
  - Images and icons
  - Data access
- WeatherMoodMobile
  - NativeBase
  - ScrollView and ListView
  - Navigation
- Animations

Prerequisite:

HTML5

CSS3

ES6

React JS

Redux

Outline

- Hello React Native
  - How it works?
  - Components, props, and states
  - Styling
  - Event handling
  - Images and icons
  - Data access
- WeatherMoodMobile
  - NativeBase
  - ScrollView and ListView
  - Navigation
- Animations

React Native

- A framework that let you write apps in React JS way

Installation Guide

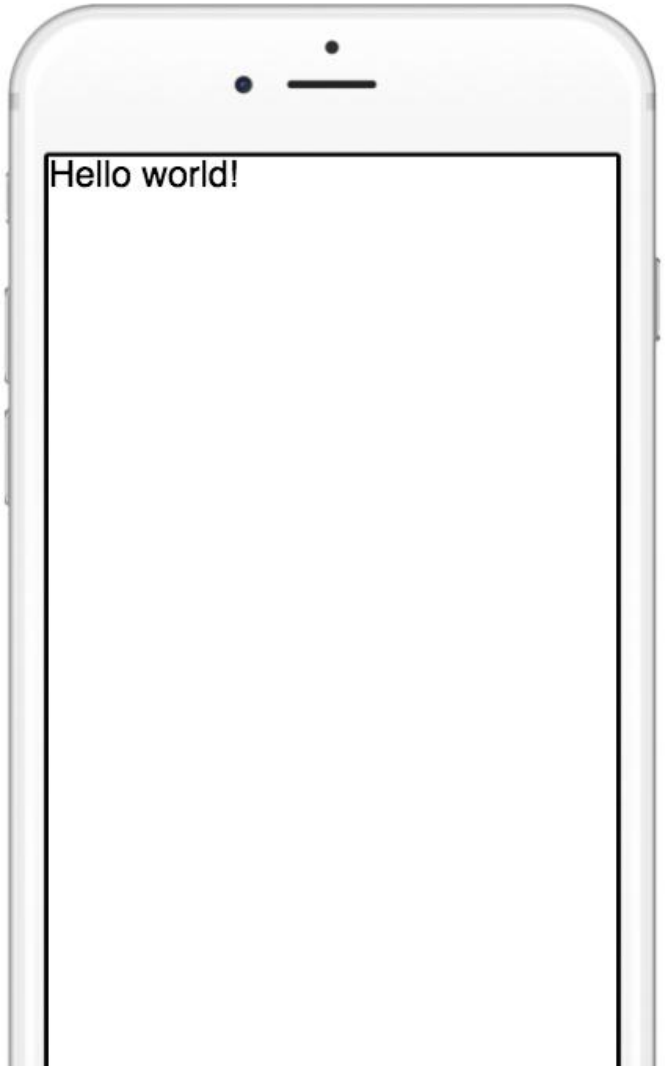
```
> react-native init HelloReactNative

// iOS
> react-native run-ios

// on Android, start AVD first
> react-native run-android
```

HelloReactNative

```
> react-native init HelloReactNative

// in HelloReactNative/index.[ios|android].js
import React from 'react';
import {AppRegistry, Text} from 'react-native';

class MyApp extends React.Component {
  render() {
    return (
      <Text>Hello world!</Text>
    );
  }
}

AppRegistry.registerComponent(
  'HelloReactNative',
  () => MyApp
);
```

- Camel-case convention
- ES6 features
- JSX with RN components
  - *.js files
- AppRegistry instead of ReactDOM

Running and Dynamic Reloading



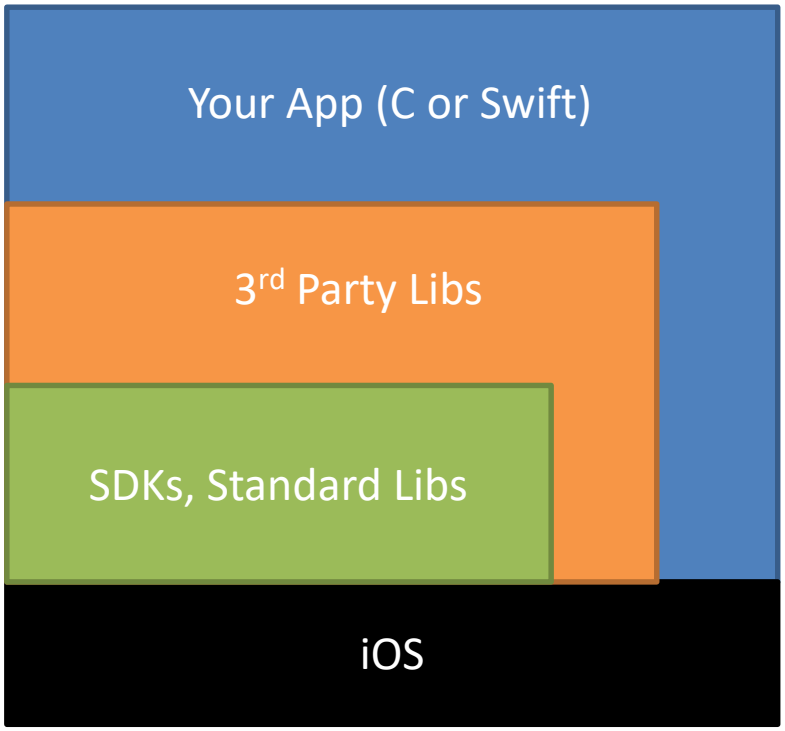
- Packager is like Webpack
- Reload:
  - Cmd + R (iOS)
  - R + R (Android)
- Dev menu:
  - Cmd + D (iOS)
  - Cmd + M (Android)
- Debugging:
  - `console.log()`
  - `debugger`

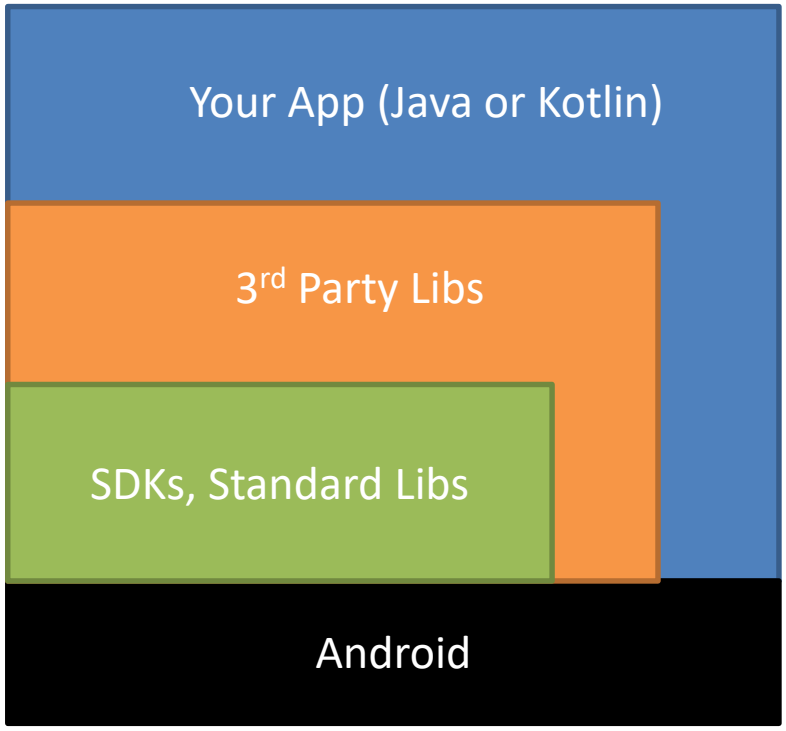
Why app written by JS is ***native***?

Outline

- Hello React Native
  - How it works?
  - Components, props, and states
  - Styling
  - Event handling
  - Images and icons
  - Data access
- WeatherMoodMobile
  - NativeBase
  - ScrollView and ListView
  - Navigation
- Animations

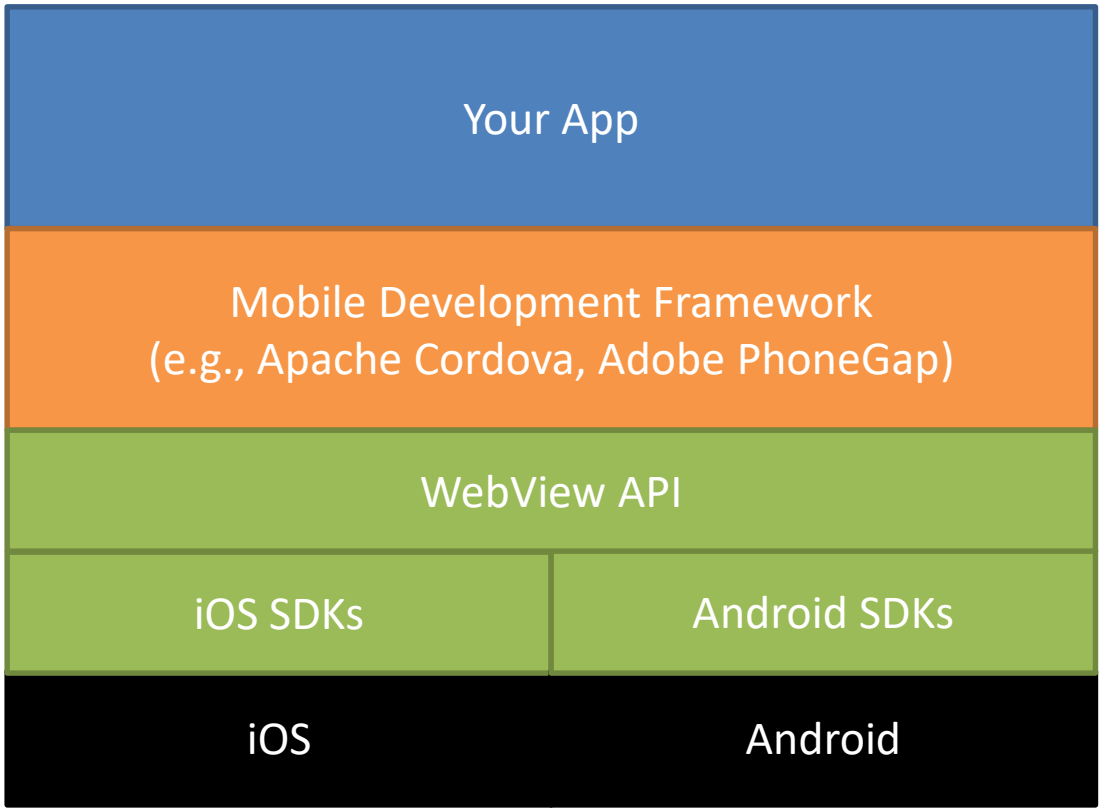
Native Apps





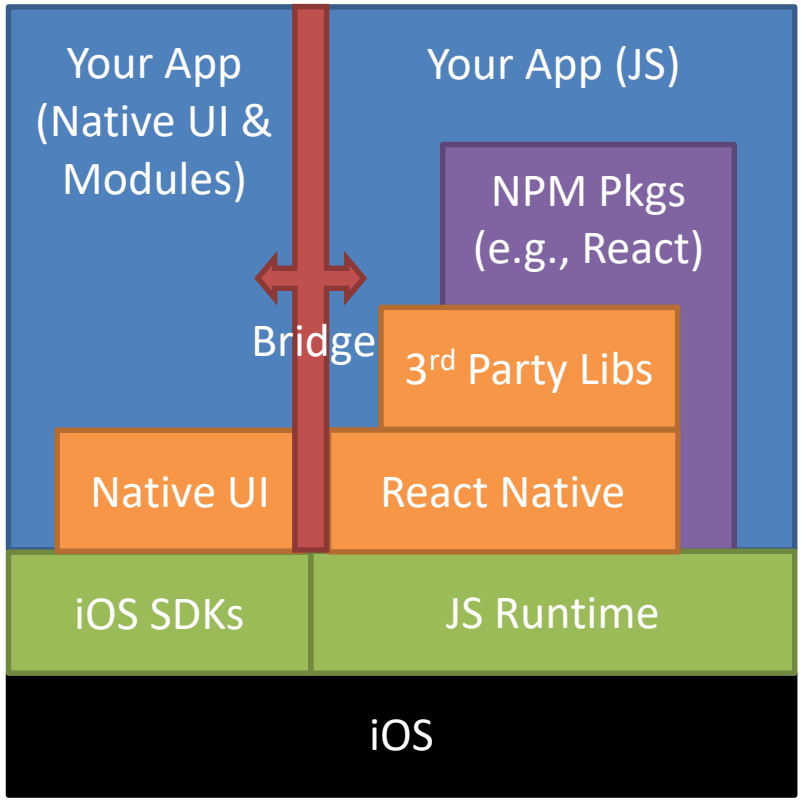
- Different code and ***language*** for different OS's

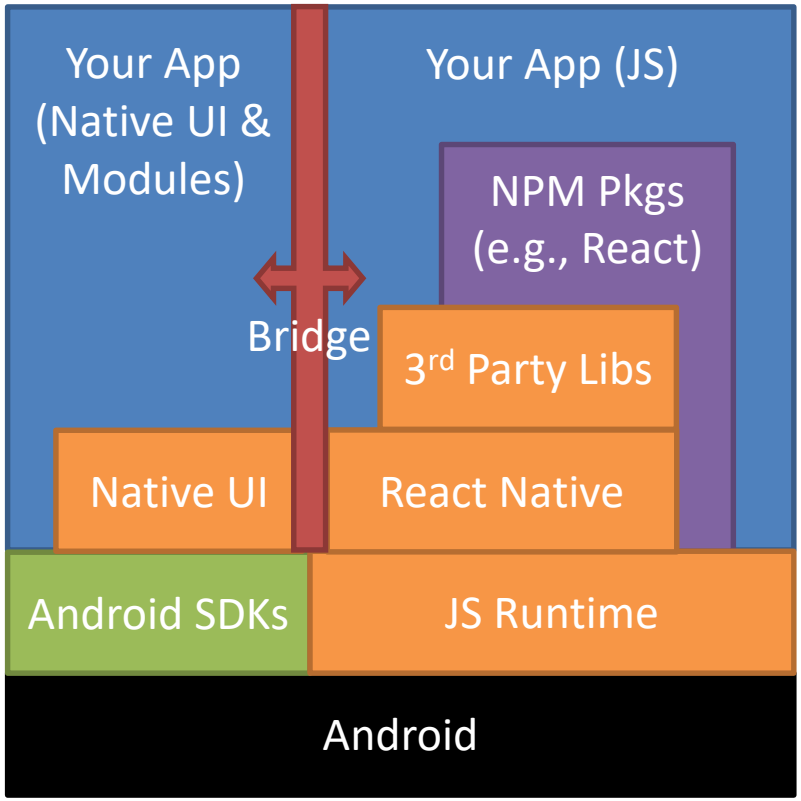
WebView Apps



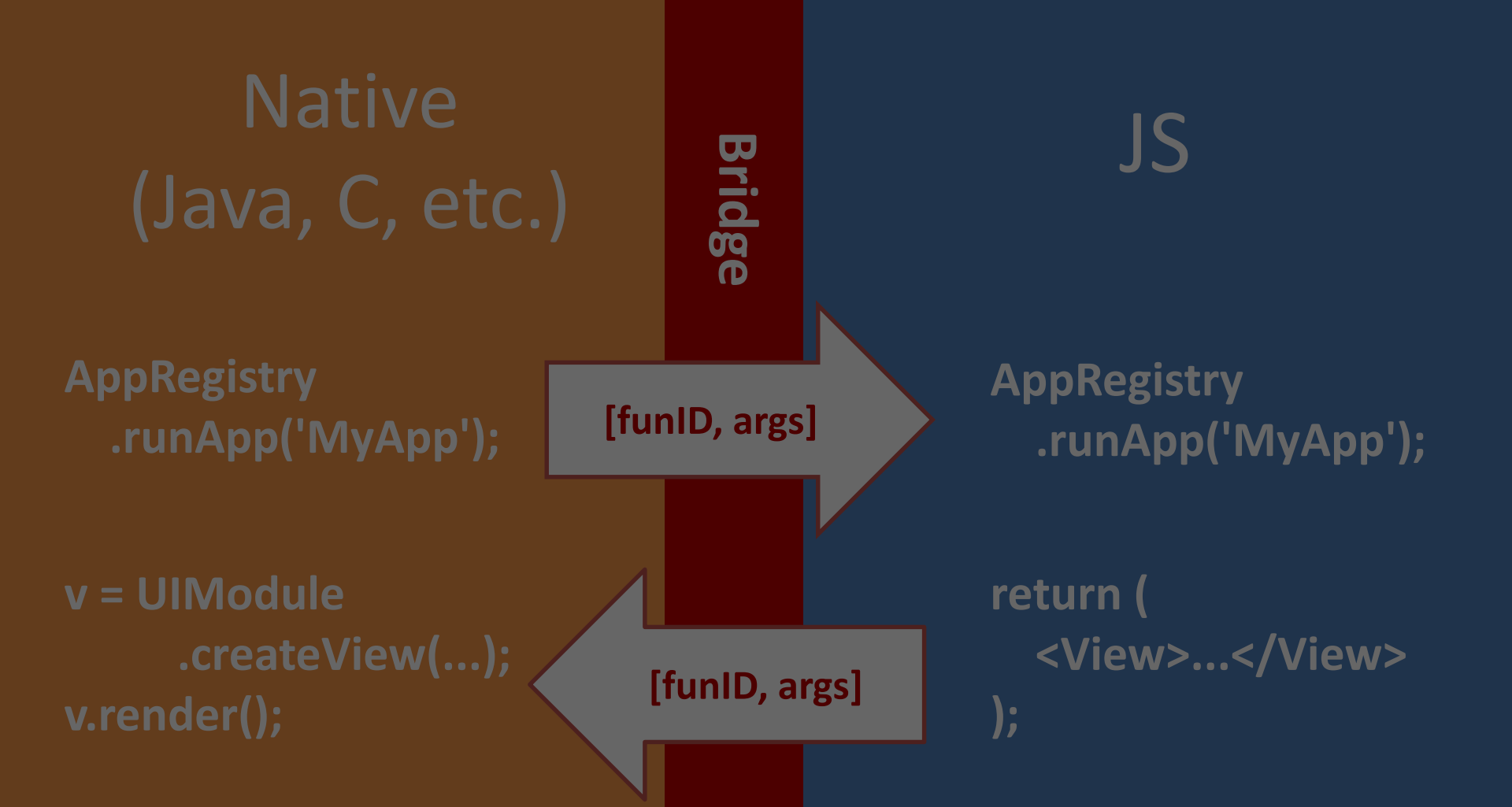
- Write once, run everywhere
- Slow and not feeling native

React-Native Apps





- JS components render as native ones
- Learn once, write everywhere



- Calls through bridge are
  - ***Asynchronous*** (event loops are separated)
  - ***Batched*** (to save overhead)

Outline

- Hello React Native
  - How it works?
  - Components, props, and states
  - Styling
  - Event handling
  - Images and icons
  - Data access
- WeatherMoodMobile
  - NativeBase
  - ScrollView and ListView
  - Navigation
- Animations

RN Components (see Doc)

- `<View>` is like `<div>` in HTML
- `<Text>` is like `<span>`
  - Text must be wrapped in `<Text>...</Text>`
- Custom components:

```
// in MyComponent.js
export defaut class MyComponent extends React.Component {
  render() {
    ...
  }
}
// in App.js
import MyComponent from './MyComponent';
// use <MyComponent /> in render()
```

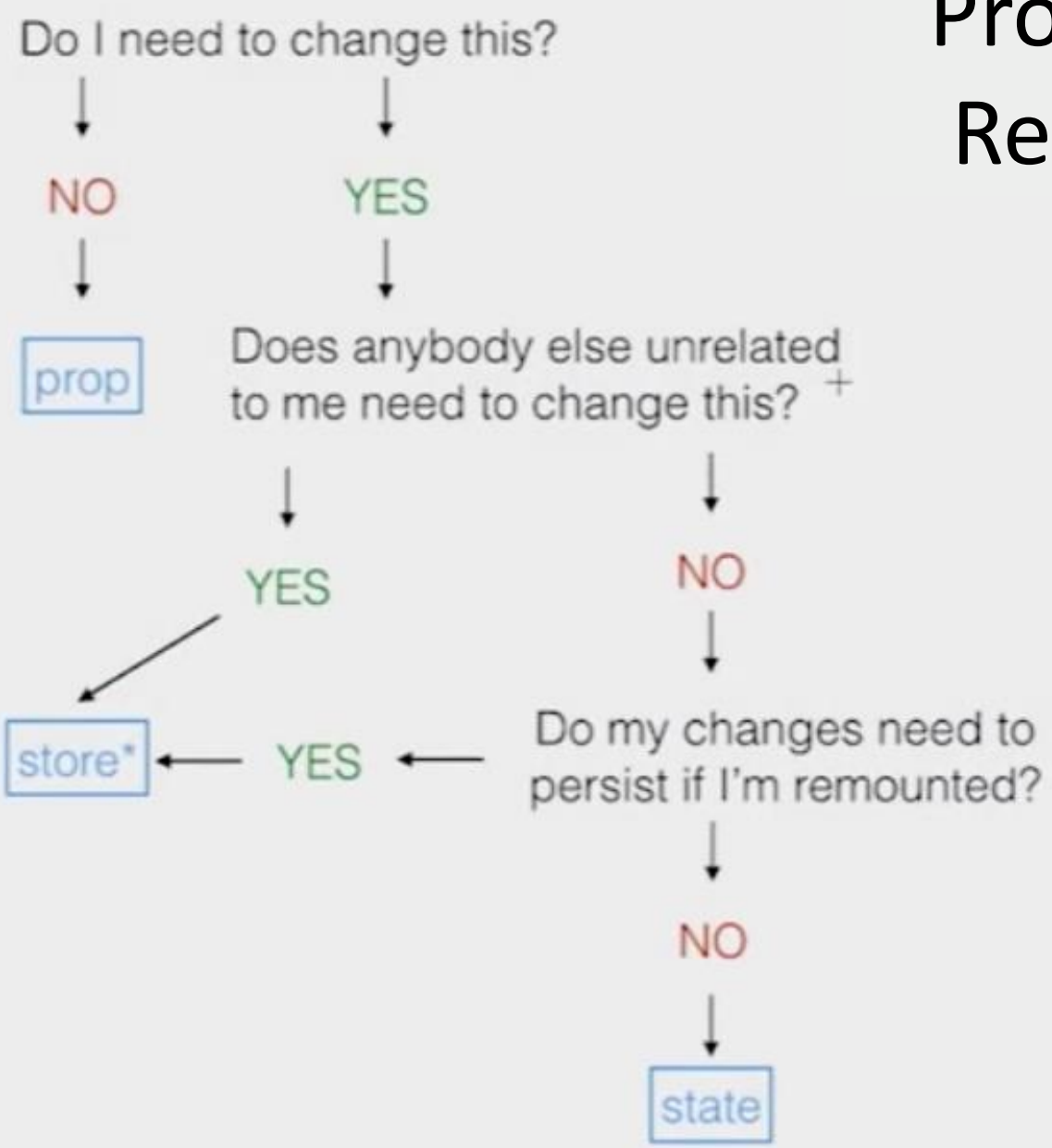
Props and States, as Usual

```
// in App.js
<MyComponent name={'Bob'} />

// in MyComponent.js
class MyComponent extends React.Component {
  constructor(props) {
    ...
    this.state = {
      isNew: true
    }
  }
  render() {
    const {name} = this.props;
    return (
      <Text>Hello {name}, {
        this.state.isNew ? 'welcome' : 'welcome back'
      }</Text>
    );
  }
}
```

Redux, as Usual

```
import {connect} from 'react-redux';

class MyComponent extends React.Component {
  render() {
    const {name, isNew} = this.props;
    return (
      <Text>Hello {name}, {
        isNew ? 'welcome' : 'welcome back'
      }</Text>

    );
  }
}

export default connect(state => ({
  isNew: state.user.isNew  // 'user' reducer
}))(MyComponent);
```

Prop, State, or Redux Store?



Outline

- Hello React Native
  - How it works?
  - Components, props, and states
  - Styling
  - Event handling
  - Images and icons
  - Data access
- WeatherMoodMobile
  - NativeBase
  - ScrollView and ListView
  - Navigation
- Animations

Styling in RN

- No CSS
- Instead, assign `style` prop to components

```
render() {
  return (
    <View>
      <Text style={{color: 'blue'}}>...</Text>
      <Text style={styles.red}>...</Text>
      // cascade
      <Text style={[styles.red, styles.title]}>...</Text>
    </View>
  );
}
const styles = {
  red: {color: 'red'},
  title: {fontSize: 24}
};
```

- List of supported styles
- Values have no unit

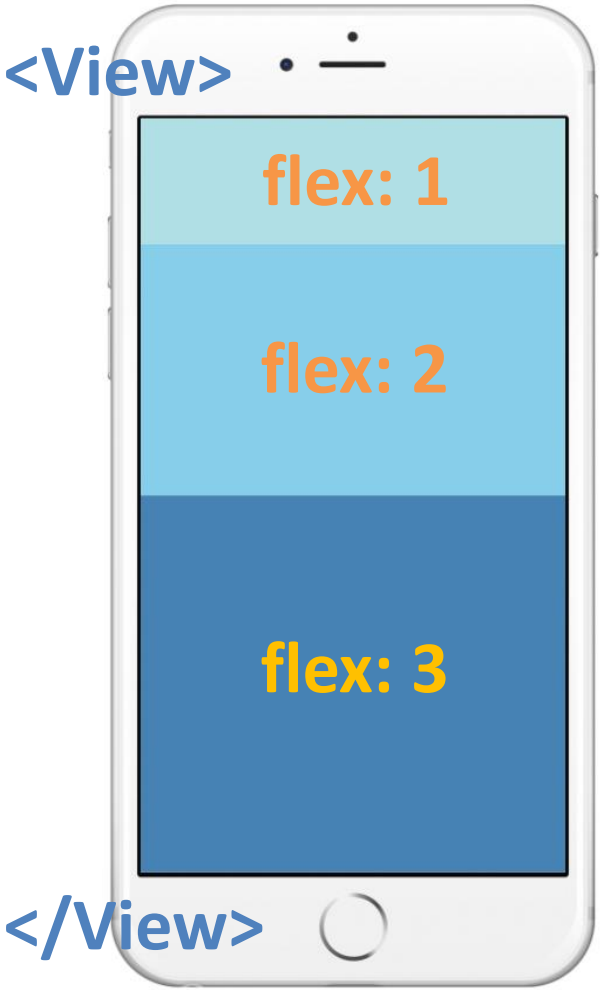
StyleSheet

```
import {StyleSheet} from
  'react-native';

render() {
  return (
    <View>
      <View style={styles.listItem}>...</View>
      <View style={styles.listItem}>...</View>
      ...
      <View style={styles.listItem}>...</View>
    </View>
  );
}
const styles = StyleSheet.create({
  listItem: {...}
});
```

- Allows multiple native components to refer to same style object (by ID)
  - Useful for, e.g., list items

Sizing and Layout

- Every “container” component (e.g., View) is a flexbox
  - flexDirection: ‘column’ by default
  - justifyContent: ‘flex-start’
  - alignItems: ‘stretch’
- Contained component:
  - alignSelf
  - width/height: number
  - ***flex***: number
- Use inspector at runtime



Outline

- Hello React Native
  - How it works?
  - Components, props, and states
  - Styling
  - Event handling
  - Images and icons
  - Data access
- WeatherMoodMobile
  - NativeBase
  - ScrollView and ListView
  - Navigation
- Animations

Event Handling

```
render() {
  return (
    <TouchableHighlight
      onPress={this.handlePress}
      onLongPress={() => alert('Yo')}>
      <View>
        <Text>Press me!</Text>
      </View>
    </TouchableHighlight>
  );
}
```

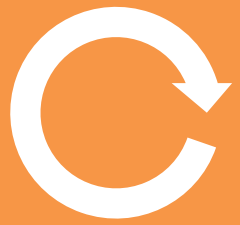
- TouchableHighlight
- TouchableOpacity
- TouachableNativeFeedback (Android only)

Controlled Components

```
render() {
  return (
    <TextInput
      placeHolder='Type here'

      value={this.state.text} // controlled component
      onChangeText={text => this.setState({text})}

      ref={el => this.inputEl}
      onEndEditing={() => {
        ...
        this.inputEl.clear();
      }}
    />
  );
}
```

How are native events handled in JS?

Native
UI (Main) Thread

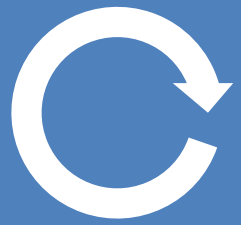


Event Queue

Native
Modules Thread

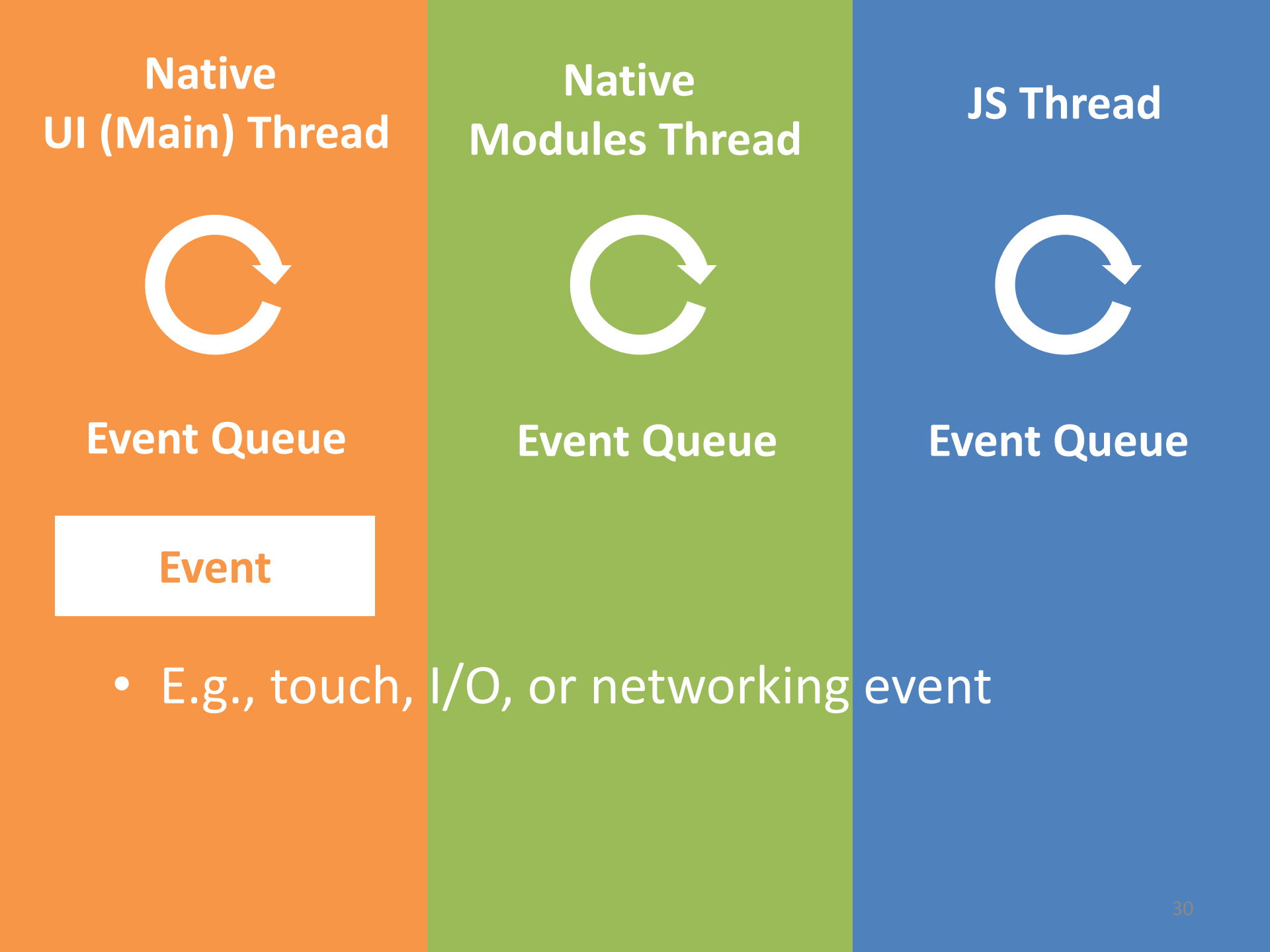
Event Queue

JS Thread

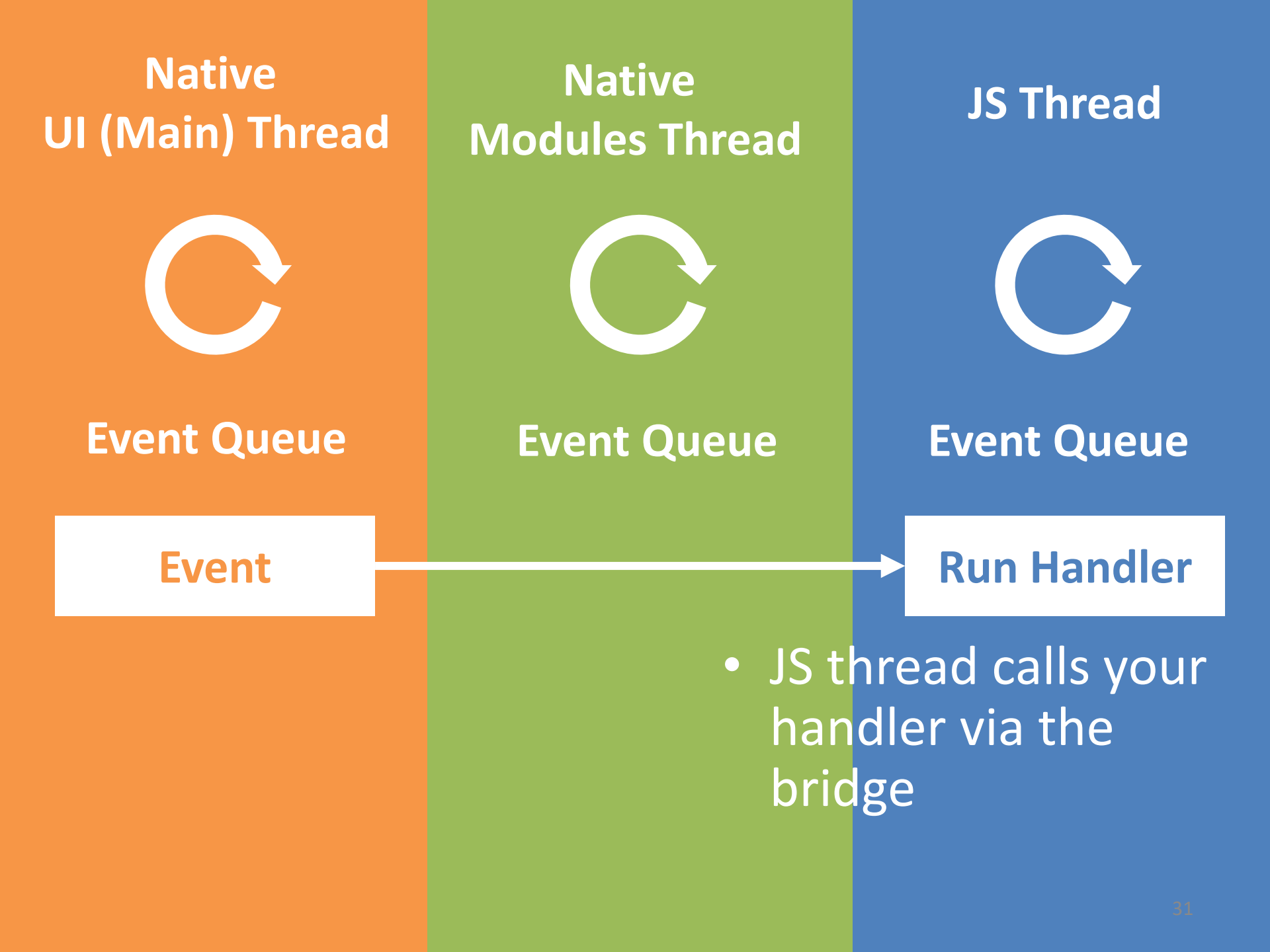


Event Queue

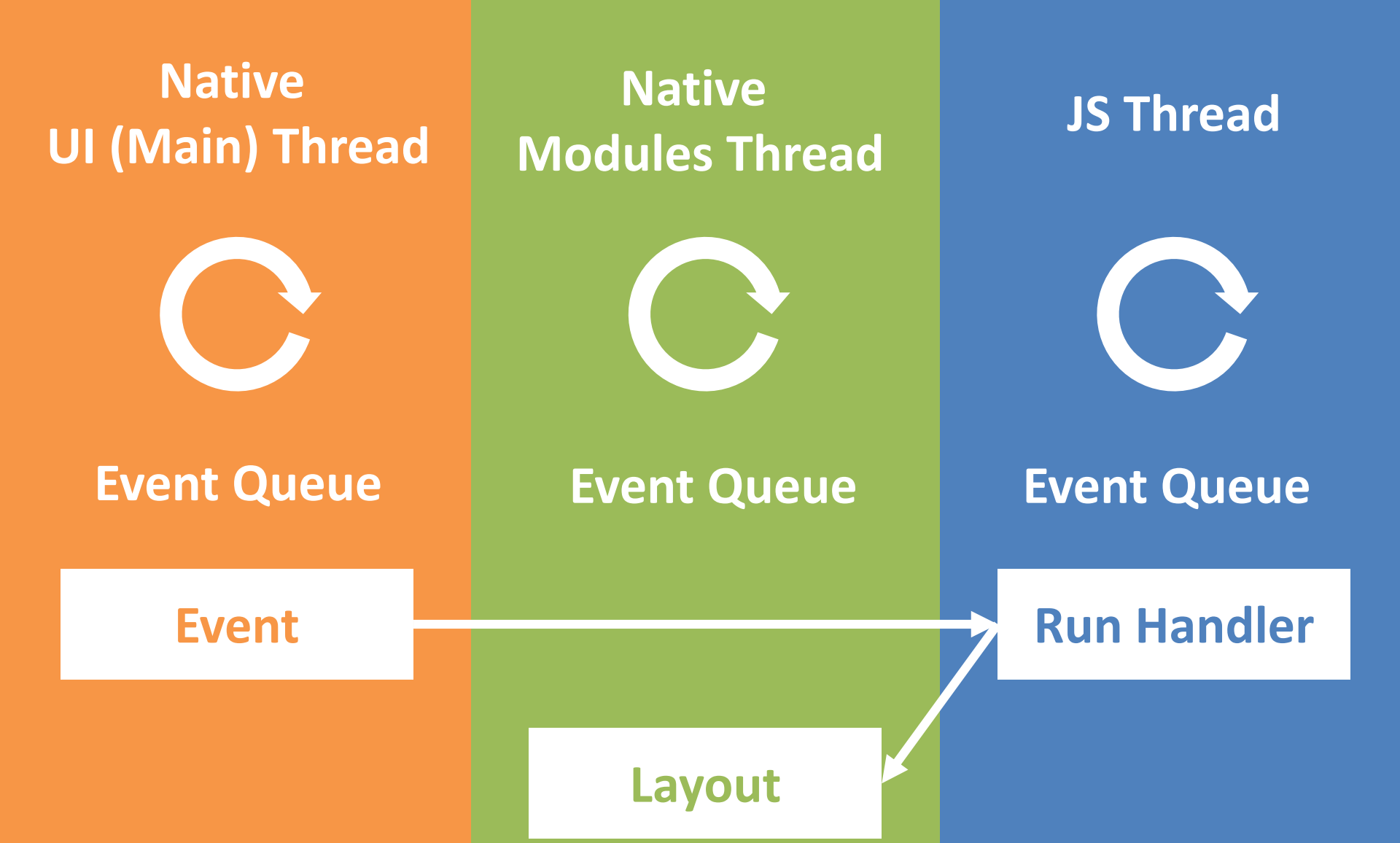
Threads and Queues



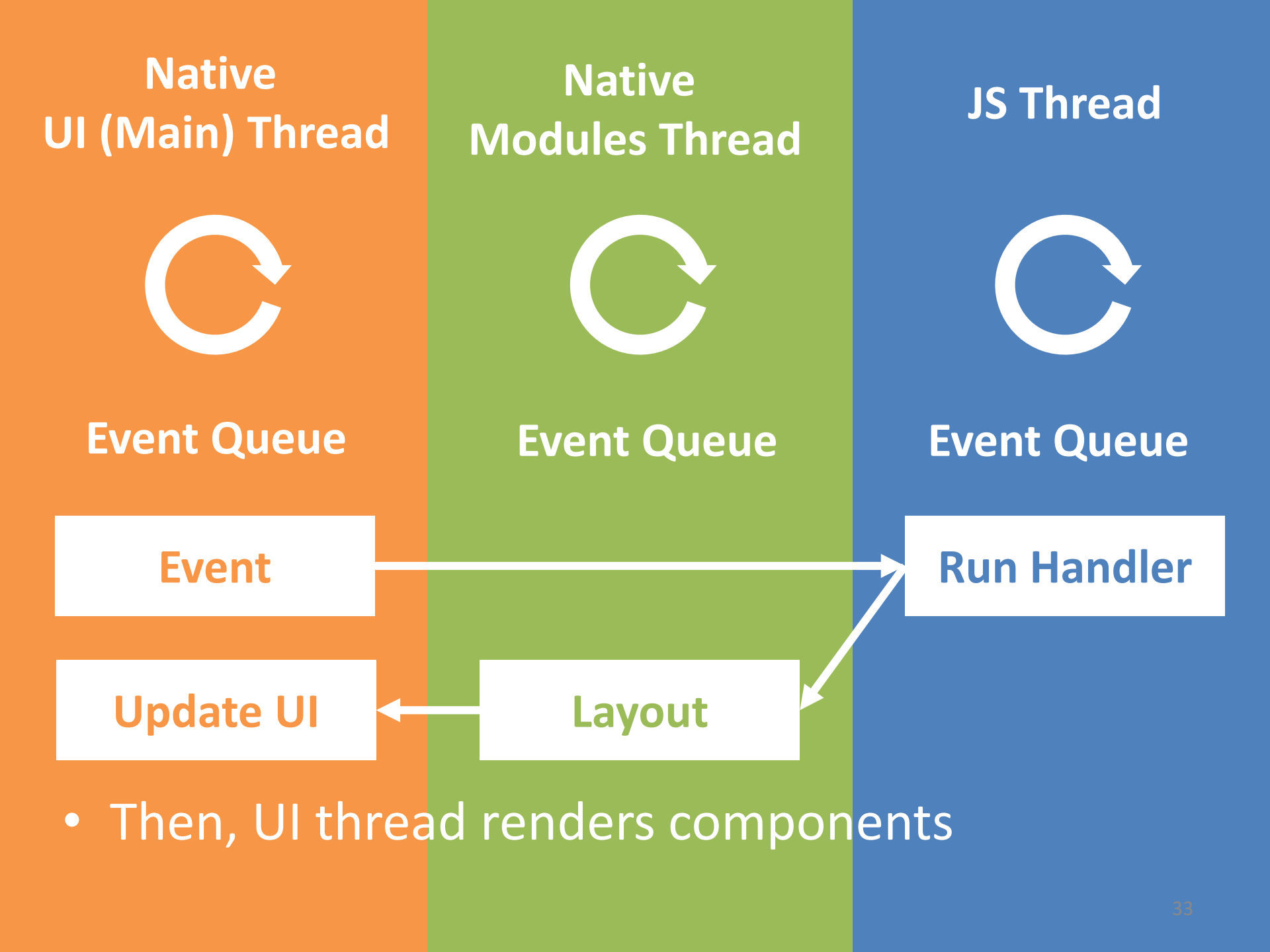
- E.g., touch, I/O, or networking event



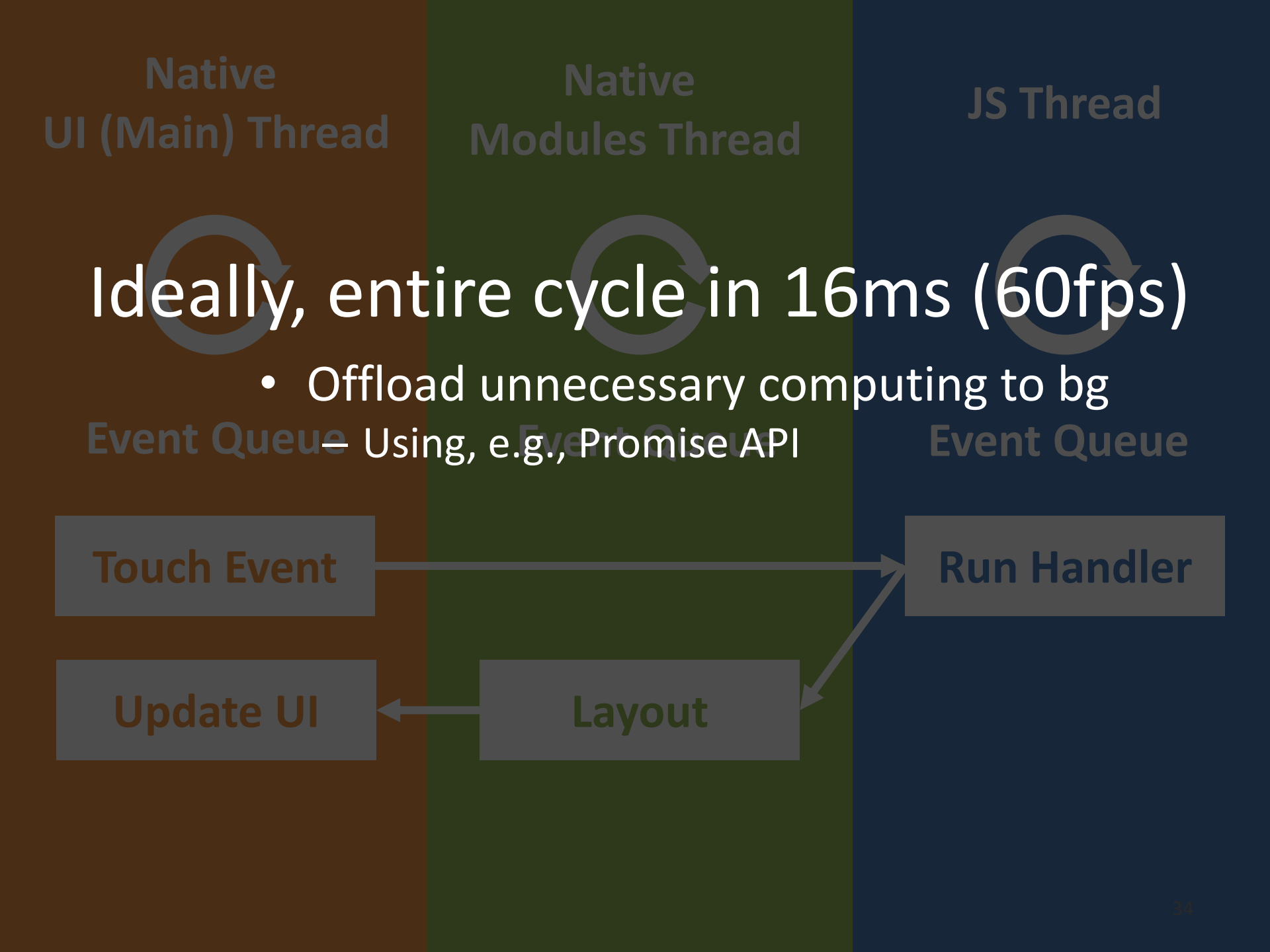
- JS thread calls your handler via the bridge



- If UI changed in JS, module thread performs layout first (e.g., measuring size of text)



- Then, UI thread renders components



Ideally, entire cycle in 16ms (60fps)

- Offload unnecessary computing to bg
  - Using, e.g., Promise API

Outline

- Hello React Native
  - How it works?
  - Components, props, and states
  - Styling
  - Event handling
  - Images and icons
  - Data access
- WeatherMoodMobile
  - NativeBase
  - ScrollView and ListView
  - Navigation
- Animations

Images

```
// JSX
<Image source={require('dir/image.png')} style={{...}} />

// in dir
image@2x.png // iPhone 7
image@3x.png // iPhone 7 Plus or Nexus 5
```

- RN handles ***off-thread*** decoding for you
- Size inferred from source file by default
  - To scale image dynamically (with `flex`), set `width` and `height` to `undefined`
- Background image?

```
<Image source={...} resizeMode='cover' style={{...}}>
  <View>...</View>
</Image>
```

Network Images

```
<Image
  source={{
    uri: 'https://.../image.png',
    cache: 'reload' // or 'force-cache' or 'only-if-cached'
  }}
  style={{width: 200, height: 200}}
  onLoad={...}
/>
```

- RN handles caching for you
- But you need to specify size manually
- It's a good practice to display a static placeholder before loaded

```
// in JSX
{this.state.isLoaded ?
  <Image source={{uri: ...}}
    onLoad={() => this.setState({isLoaded: true})} /> :
  <Image source={require('dir/placeholder.png')}>}
```

Font Icons

```
> npm install --save react-native-vector-icons
> react-native link

// in JS
import Icon from 'react-native-vector-icons/FontAwesome';
// JSX
<Icon name="rocket" size={30} color="#900" />
```

- See more supported fonts and features

Outline

- Hello React Native
  - How it works?
  - Components, props, and states
  - Styling
  - Event handling
  - Images and icons
  - Data access
- WeatherMoodMobile
  - NativeBase
  - ScrollView and ListView
  - Navigation
- Animations

Networking

```
// GET
fetch('https://...')
  .then((res) => {
    if (res.status !== 200) throw new Error('...');
    return res.json();
  })
  .then(data => ...)
  .catch(err => ...)

// POST
fetch('https://...', {
  method: 'POST',
  headers: {
    Accept: 'application/json',
    'Content-Type': 'application/json',
    ...
  }
  body: JSON.stringify(...)
})
```

- Fetch API
- Plaintext HTTP requests will be blocked on iOS
  - Apps on Apple's App Store shall use HTTPS

App Transport Security (ATS) Exception

```
// in [PROJ_ROOT]/ios/[PROJ_NAME]/Info.plist
<key>NSAppTransportSecurity</key>
<dict>
  <key>NSExceptionDomains</key>
  <dict>
    ...
    <key>yourdomain.com</key>
    <dict>
      <!--Include to allow subdomains-->
      <key>NSIncludesSubdomains</key>
      <true/>
      <!--Include to allow HTTP requests-->
      <key>NSTemporaryExceptionAllowsInsecureHTTPLoads</key>
      <true/>
    </dict>
  </dict>
</dict>
```

- Re-run `react-native run-ios`

Persistent Storage

- In mobile landscape, Internet may ***not*** always be available
- It's a good practice to allow offline data access
- AsyncStorage (global to app):

```
// API similar to HTML5 LocalStorage
AsyncStorage.setItem(key, value); // strings
AsyncStorage.mergeItem(key, delta);
AsyncStorage.getItem(key).then(value => ...);
AsyncStorage.multiGet(keys).then(values => ...);
```

Persisting Redux States

```
// save when any state changes
store.subscribe(() => {
  AsyncStorage.setItem('states',
    JSON.stringify(store.getState()));
});

// load
AsyncStorage.getItem('states').then(value => {
  store = createStore(
    combineReducers(...),
    JSON.parse(value),
    compose(...)
  );
});
```

- You can persist partial states

Outline

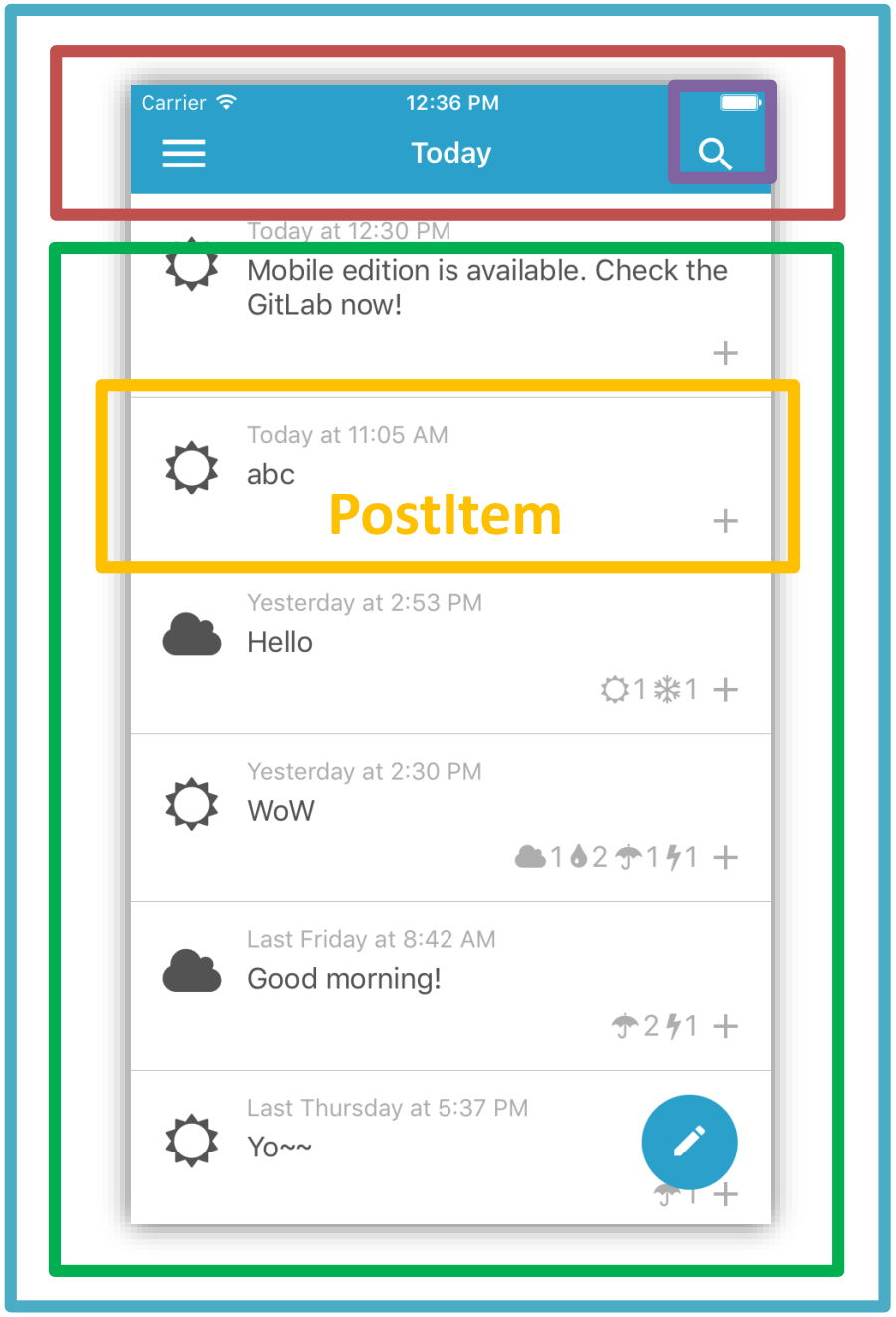
- Hello React Native
  - How it works?
  - Components, props, and states
  - Styling
  - Event handling
  - Images and icons
  - Data access
- WeatherMoodMobile
  - NativeBase
  - ScrollView and ListView
  - Navigation
- Animations

Clone WeatherMoodMobile

- Checkout the `redux-post` branch

```
> npm install --save \
  native-base color \
  react-native-infinite-scroll-view \
  react-navigation
> react-native link
```

- NativeBase and Color for UI
- RN Infinite Scroll View
- React Navigation for client-side routing



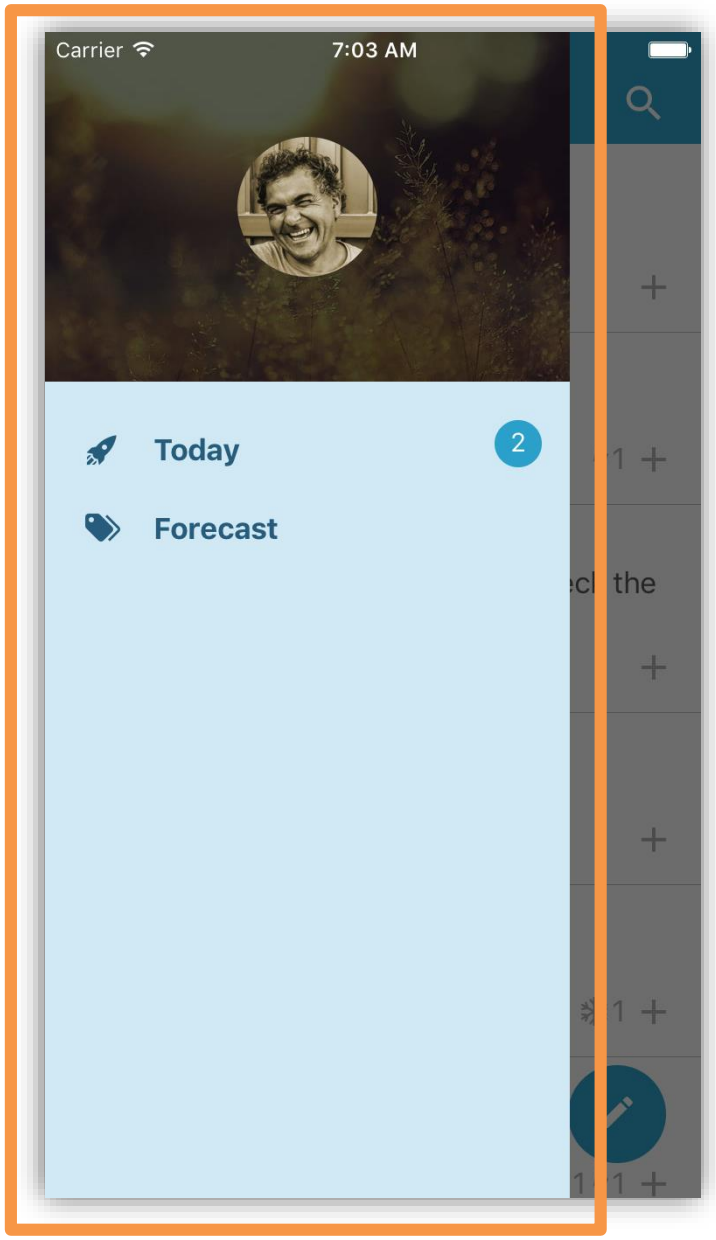
NavigationContainer

SearchButtonWithModal

Components

PostList

TodayScreen



Components

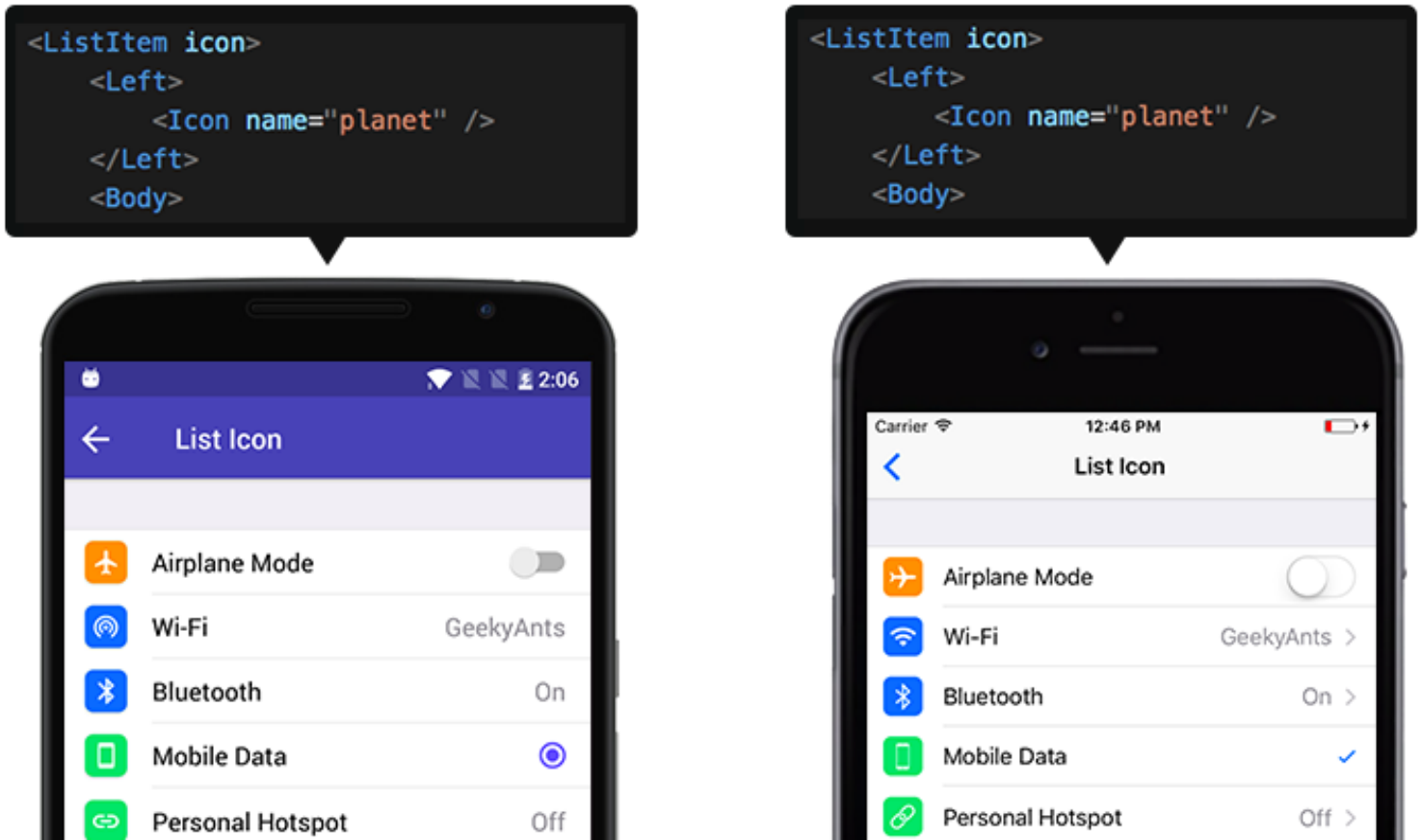
DrawerSideBar

Outline

- Hello React Native
  - How it works?
  - Components, props, and states
  - Styling
  - Event handling
  - Images and icons
  - Data access
- WeatherMoodMobile
  - NativeBase
  - ScrollView and ListView
  - Navigation
- Animations

NativeBase

- Same component, different (native) looks



Theme Customization

```
> node node_modules/native-base/ejectTheme.js
> vim native-base-theme/variables/platform.js

// in app.js
import {StyleProvider} from 'native-base';
import getTheme from '../native-base-theme/components';
import platform from
  '../native-base-theme/variables/platform';

class MyApp extends React.Component {
  render() {
    return (
      <StyleProvider style={getTheme(platform)}>
        <View>...</View>
      </StyleProvider>
    );
  }
}
```

- Read more about customization

Platform-Specific Code

- Platform-specific files:

```
index.ios.js
index.android.js

images/banner@2x.ios.jpg
images/banner@2x.android.jpg
```

- Or use Platform:

```
import {Platform} from 'react-native';

// in JS
const styles = StyleSheet.create({
  toolbar: {
    height: (Platform.OS === 'ios') ? 64 : 56
  }
});
```

Flat Style Objects

```
import {Button} from
  'native-base';

class MyComponent extends React.Component {
  render() {
    return (
      <Button style={styles.btn}>  // error
        <Text>...</Text>
      </Button>
    );
  }
}
const styles = StyleSheet.create({
  btn: {...}
});
```

- NB components create StyleSheets automatically
  - Use plain objects, or
  - `Stylesheet.flatten(styles.btn)`

Outline

- Hello React Native
  - How it works?
  - Components, props, and states
  - Styling
  - Event handling
  - Images and icons
  - Data access
- WeatherMoodMobile
  - NativeBase
  - ScrollView and ListView
  - Navigation
- Animations

ScrollView

```
<ScrollView
  horizontal={true}
  onScroll={e => {
    const y = e.nativeEvent.contentOffset.y;
    ...
  }}
  style={{flex: 1}} // or set height directly
>
  <View>...</View>
  <Image>...</Image>
  <Text>...</Text>
  ...
</ScrollView>
```

- Elements can be heterogeneous
- Horizontal or vertical scroll
- Unbounded child height → must have bounded height

ListView

```
// in a component
constructor(props) {
  ...
  const ds = new ListView.DataSource({
    rowHasChanged: (r1, r2) => r1.id !== r2.id
  });
  this.state = {
    dataSource: ds.cloneWithRows([{/* r1 */}, ...])
  }
}
```

- Optimized for large #items:
  - Lazy and rate-limited row rendering
  - Only re-renders changed rows

```
render() {
  return (
    ...
    <ListView
      ... // props of ScrollView
      dataSource={this.state.dataSource}
      renderRow={r => <Text>r.text</Text>}
    />
  );
}
```

Refreshing & Scrolling

```
import RefreshControl from 'react-native';
import InfiniteScrollView from
  'react-native-infinite-scroll-view';


// in JSX
<ListView
  dataSource={...}
  renderRow={...}
  refreshControl={
    <RefreshControl refreshing={this.state.refreshing}
      onRefresh={() => ... /* list posts */} />
  }
  rederScrollComponent={
    props => <InfiniteScrollView {...props} />
  }
  distanceToLoadMore={300}
  canLoadMore={this.state.hasMoreRows}
  onLoadMoreAsync={() => ... /* list more posts */}
/>
```

Outline

- Hello React Native
  - How it works?
  - Components, props, and states
  - Styling
  - Event handling
  - Images and icons
  - Data access
- WeatherMoodMobile
  - NativeBase
  - ScrollView and ListView
  - Navigation
- Animations

Navigation

```
// in app.js
import {StackNavigator} from
  'react-navigation';
const App = StackNavigator({
  Home: {screen: HomeScreen},
  Contact: {screen: ContactScreen}
});
```

```
class HomeScreen extends React.Component {
  render() {
    const {navigate} = this.props.navigation;
    return (
      <Button onPress={() => navigate('Contact')}>...</Button>
    );
  }
}

class ContactScreen extends React.Component {
  render() {
    const {goBack} = this.props.navigation;
    return (
      <Button onPress={() => goBack()}>...</Button>
    );
  }
}
```

- Supports Redux integration

Outline

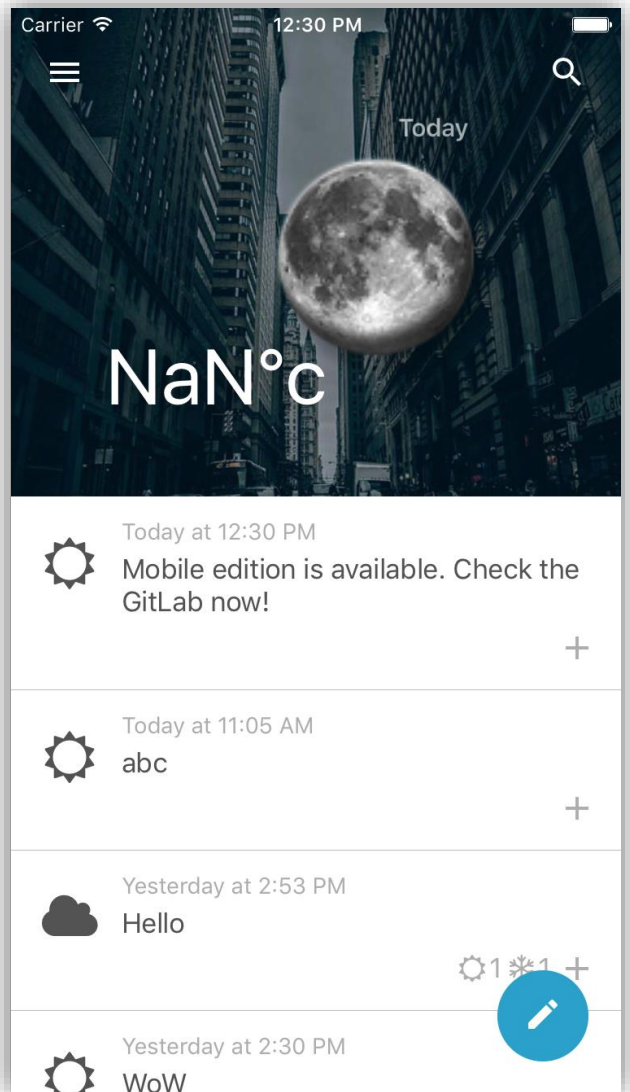
- Hello React Native
  - How it works?
  - Components, props, and states
  - Styling
  - Event handling
  - Images and icons
  - Data access
- WeatherMoodMobile
  - NativeBase
  - ScrollView and ListView
  - Navigation
- Animations

People expect great UX from apps...

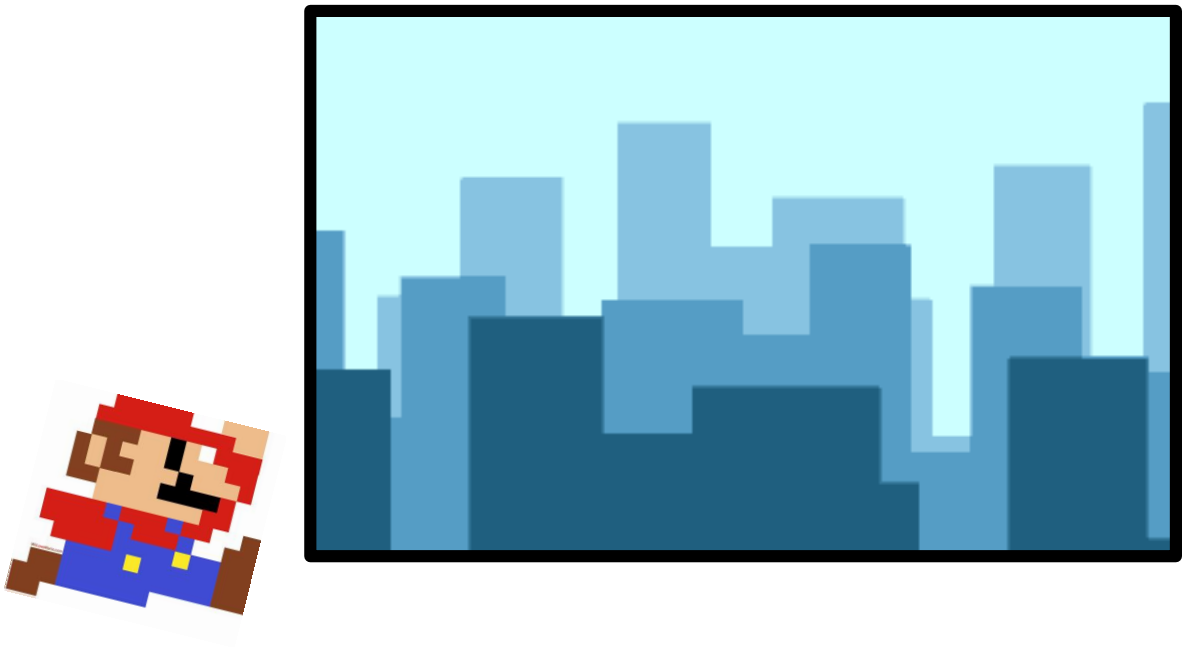
So, animation is a “must”

WeatherMoodMobile

- Checkout the `parallax-header` branch



What does “parallax” mean?



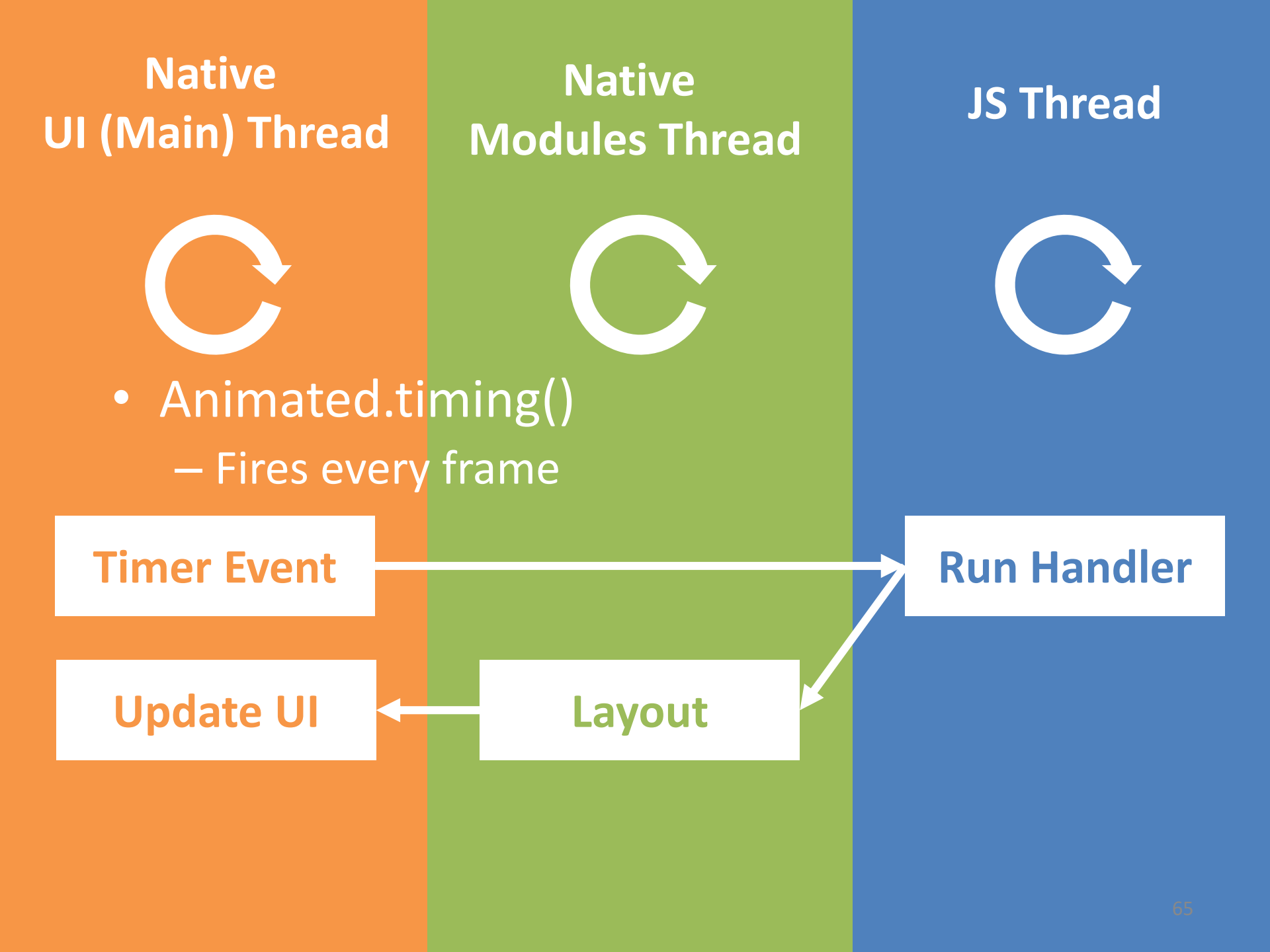
```
import {Animated, Easing} from 'react-native';

class FadeInComponent extends React.Component {
  constructor(props) {
    ...
    this.opacityAnim = new Animated.value(0);
  }
  componentDidMount() {
    Animated.timing(this.opacityAnim, {
      toValue: 1,
      easing: Easing.back // or bounce, etc.
      duration: 1000 // in ms,
      useNativeDriver: true
    }).start();
  }
  render() {
    return (
      <Animated.View style={{opacity: this.opacityAnim}}>
        ...
      </Animated.View>
    );
  }
}
```

Animations

- Or, try canned animations

```
class FadeInComponent extends React.Component {
  constructor(props) {
    ...
    this.state = {
      opacity: 0
    };
  }
  componentDidMount() {
    this.fadeInId = setTimeout(() => {
      if (this.state.opacity < 1.0)
        this.setState({opacity: this.state.opacity + 0.0167});
      else clearTimeout(this.fadeInId);
    }, 16); // 60 fps
  }
  render() {
    return (
      <View style={{opacity: this.state.opacity}}>
        ...
      </>
    );
  }
}
```

Why not use state?

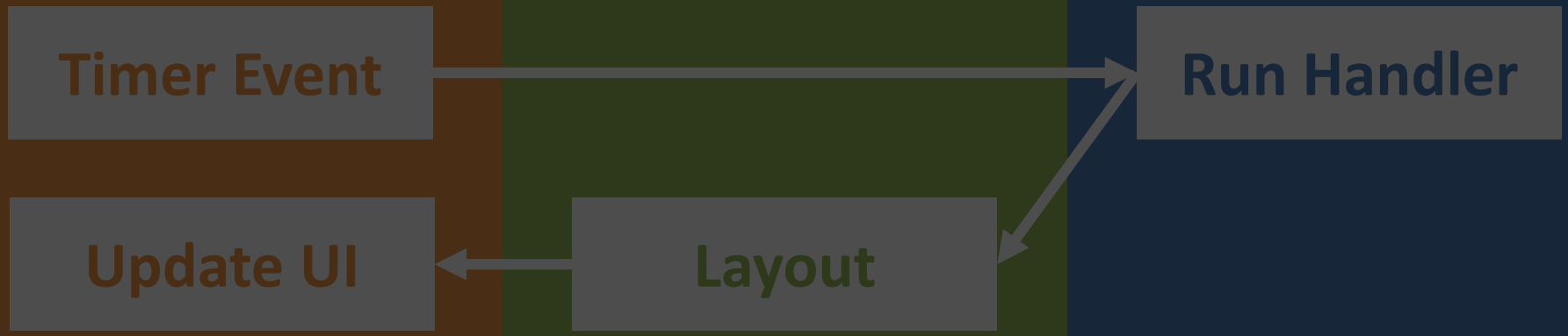

Native
UI (Main) Thread
Native
Modules Thread
JS Thread
• Animated.timing()
– Fires every frame
Timer Event
Run Handler
Update UI
Layout

Native
UI (Main) Thread

Native
Modules Thread

JS Thread

- Transition of animated value is declarative
  - Known before start()
- Optimization:
  - High priority threads
  - Native driver allowing native-only animation



Interpolation of Animated Values

```
componentDidMount() {
  Animated.timing(this.opacityAnim, {
    toValue: 1,
    ...
  }).start();
}
render() {
  return (
    <Animated.View style={{
      opacity: this.opacityAnim, // fade in
      transform: [{
        translateY: this.opacityAnim.interpolate({ // slide in
          inputRange: [0, 1],  // or [0, 0.5, 1],
          outputRange: [150, 0],  // or [150, 50, 0]
          extrapolate: 'clamp' // or 'extend'
        })
      }]
    }}>
      ...
    </Animated.View>
  );
}
```

- Also supports multiple segments

Tracking Gestures

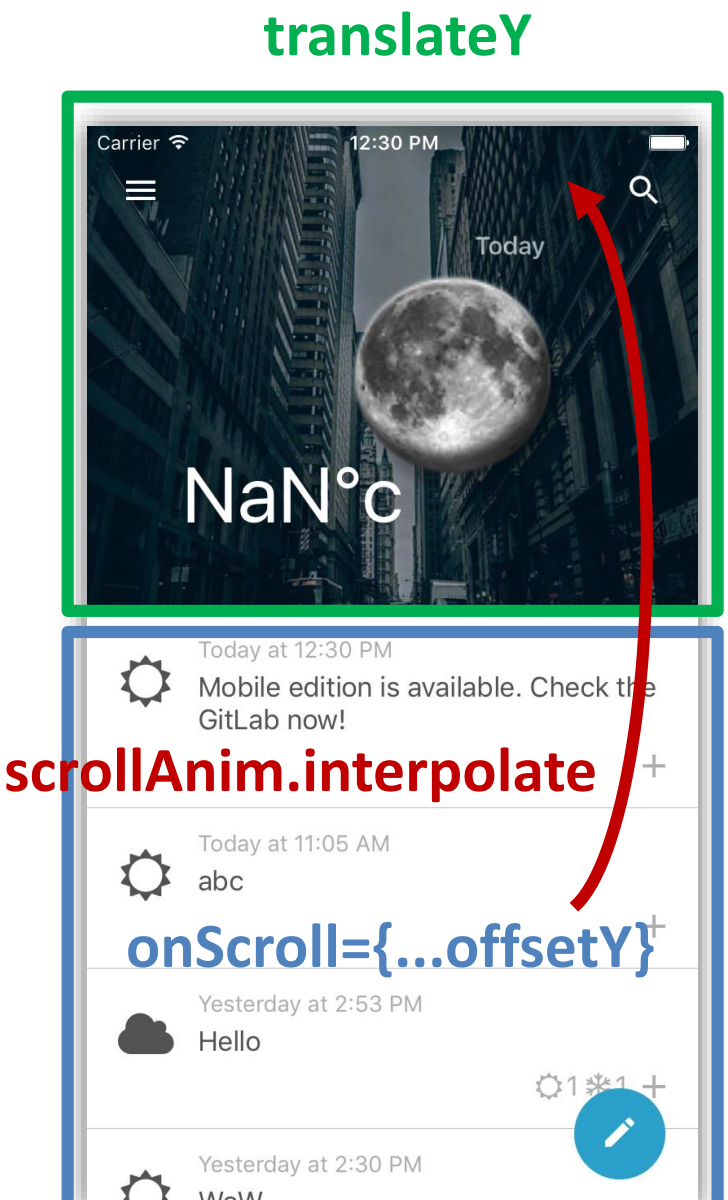
```
// in constructor
this.scrollAnim = new Animated.Value(0);

// in JSX
<ListView
  ...
  onScroll={e => {
    const y = e.nativeEvent.contentOffset.y;
    this.scrollAnim.setValue(y);
  }}
/>
```

```
    Animated.event(
      [{nativeEvent:{contentOffset: {y: this.scrollAnim}}}],
      {useNativeDriver: true}
    )
```

```
<Animated.View
  style={{
    opacity: this.scrollAnim.interpolate(
      inputRange: [0, 200],
      outputRange: [1, 0],
      extrapolate: 'clamp'
    )
  }}
>...</Animated.View>
```

- Declarative transition of animated value?

Parallax Header



- Pitfall: the scroll view itself is translating
- Fluctuate content offset (y)
  - y depends not only on gesture
- Solution: average multiple y's within a small window

Readings

- Official Guides
- RN internals
  - Android
  - iOS
- Awesome React Native

Publishing

- Apple's App Store:
  - Checklist
- Google Play Store:
  - Sign your APK first
  - Checklist

Assignment

- Complete the "weather part" in `Today.js`
- Design and implement `Forecast.js`
  - Show forecast in the next few days
  - TODO list
- Put settings to the `Settings` screen
  - E.g., "location" for weather queries
- Once submission per group
- ***Bonus (up to 50%): creative animations***