# CS 696 Multi-platform Mobile App Development
Fall Semester, 2020
Doc 17 Props, State, Basic Components
Oct 27, 2020

## Elements

React Element

  Tree of HTML tags and Components

React Native Element

  Tree of Components

Elements

  Describe the interface

## Element

An element has

- Type
- Props (properties)
- Children

To create an element use

```
React.createElement(type, props, children)
```

```
React.createElement(Text, null, "Hello World");
```

```
React.createElement(View, null, React.createElement(Text, null, "Hello World"))
```

This is verbose so there is a short cut

## JSX - JavaScript eXtension

Extension of JavaScript

JSX valid syntax

`<text>Hello World</text>` $\xrightarrow[\text{Babel}]{}$ `React.createElement(Text, null, "Hello World");`

```
<View><Text>Hello World</Text></View>
```

```
React.createElement(View, null,
          React.createElement(Text, null, "Hello World"))
```

## Adding JavaScript

Can embed JavaScript in JSX by putting it in { }

```
<Text>1 + 2 = {1 + 2}</Text>
```

```
React.createElement(Text, null, "1 + 2 = ", 1 + 2)
```

## Props - Properties

```
class Cat extends Component {
  render() {
    return <Text>Hello, I am {this.props.name}!</Text>;
  }
}
```

```
class Cat extends Component {
  constructor(props) {
    super(props);
  }
  render() {
    return <Text>Hello, I am {this.props.name}!</Text>;
  }
}
```

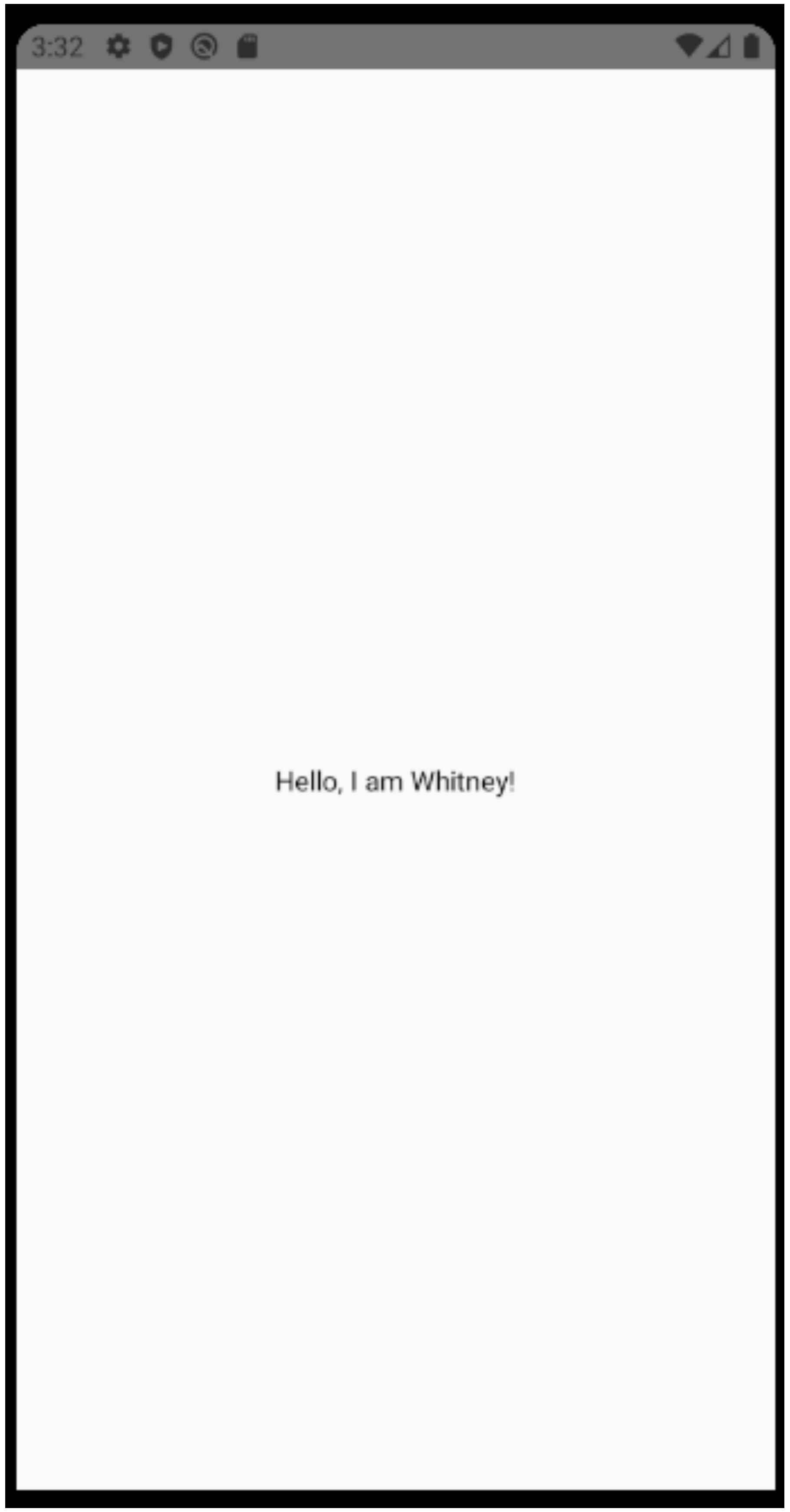
Class Component has two properties

- props
- state

## Setting the prop

```
import React, {Component} from 'react';
import {View, Text} from 'react-native';
const App: () => React$Node = () => {
  return (
    <View
      style={{
        flex: 1,
        justifyContent: 'center',
        alignItems: 'center',
      }}>
      <Cat name="Whitney" />
    </View>
  );
};
```

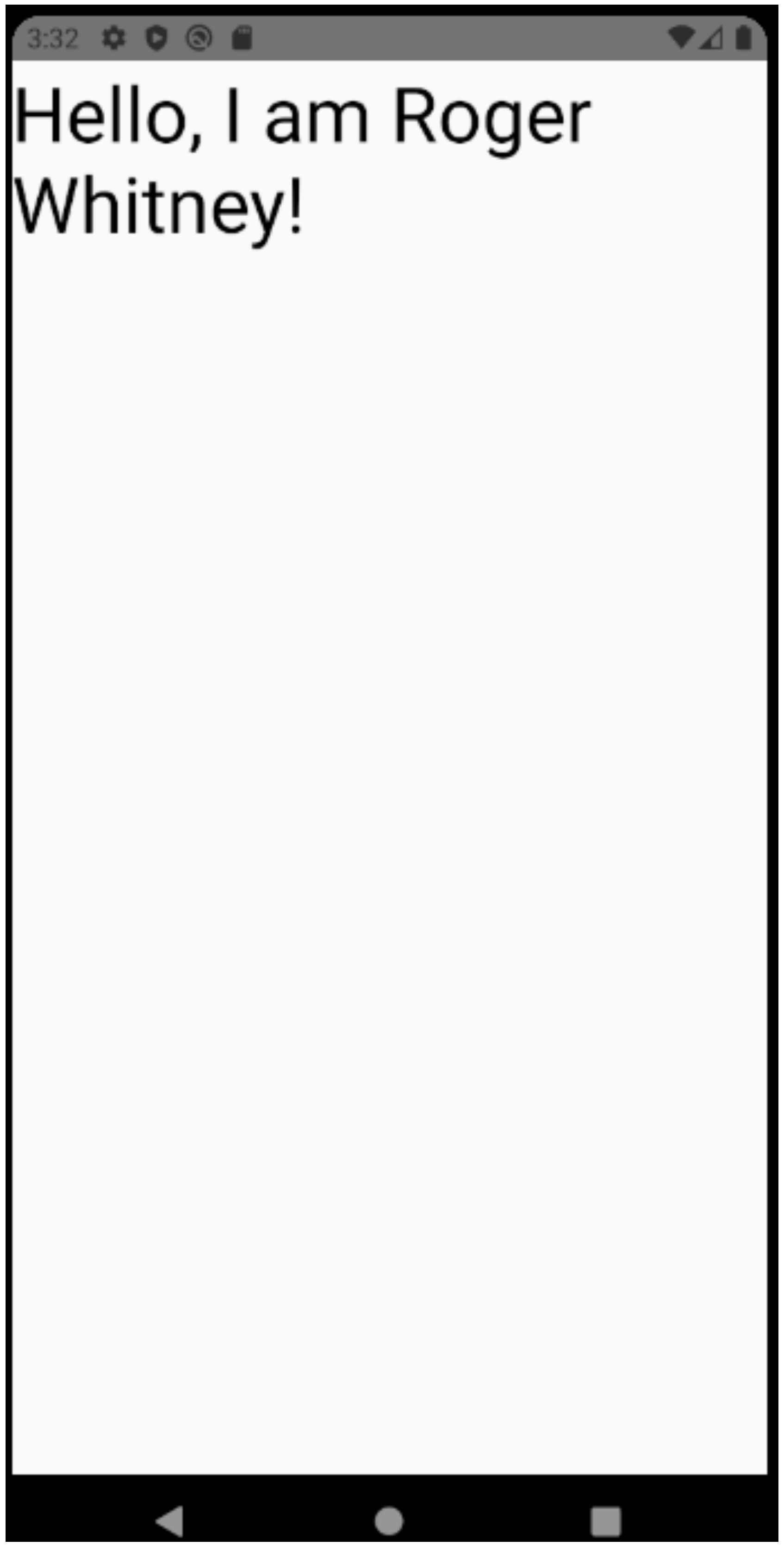
Only Strings can be assigned directly
Other need to be inside { }



Hello, I am Whitney!

## JS Code in Props

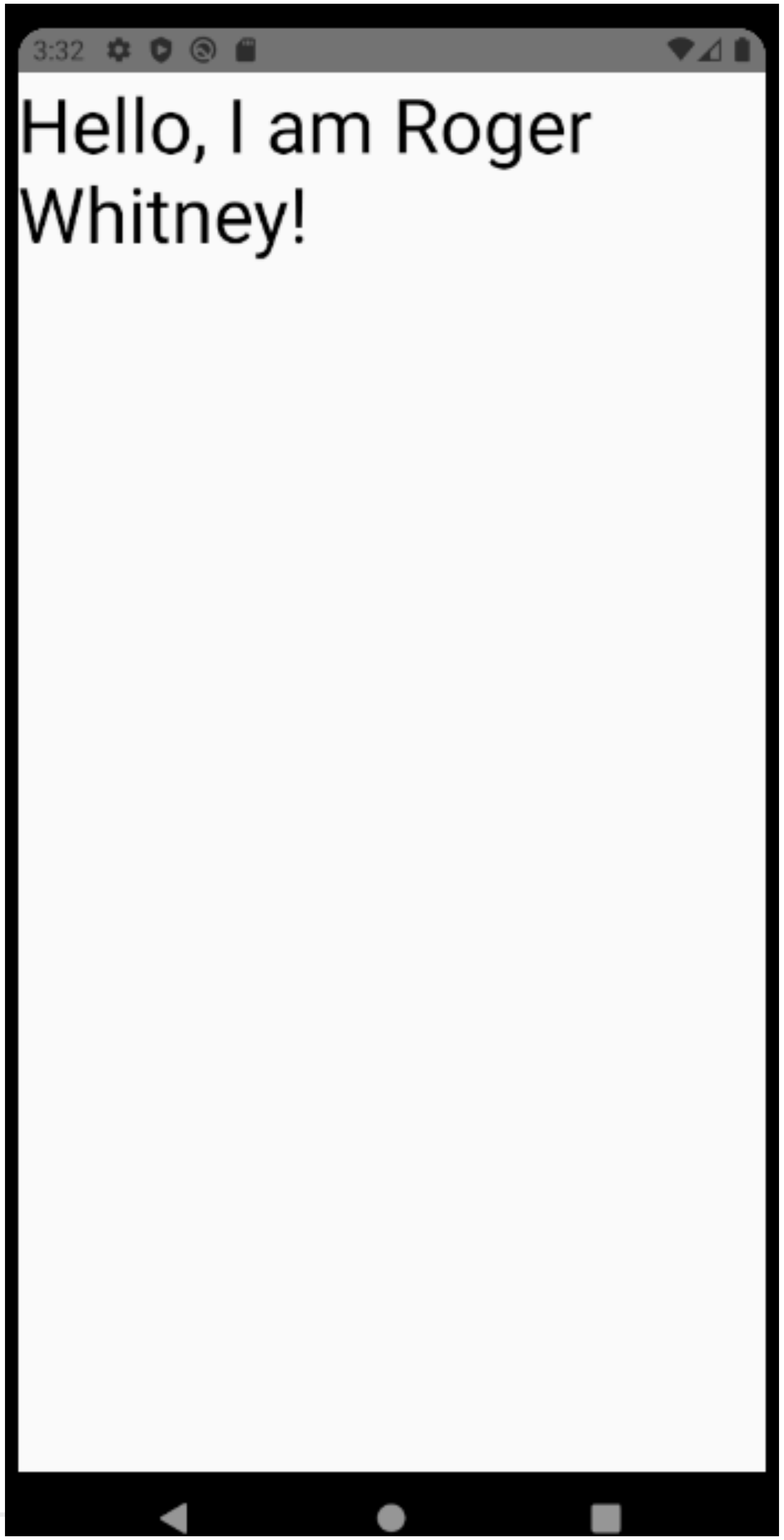
```
import React, {Component} from 'react';
import {View, Text} from 'react-native';
const App: () => React$Node = () => {
  return (
    <View>
      <Cat name={'Roger ' + 'Whitney'} />
    </View>
  );
};

class Cat extends Component {
  constructor(props) {
    super(props);
  }
  render() {
    return <Text style={{fontSize: 40}}>
      Hello, I am {this.props.name}!
    </Text>;
  }
}
```



Hello, I am Roger Whitney!

```
import React, {Component} from 'react';
import {View, Text} from 'react-native';
const App: () => React$Node = () => {
  return (
      <Cat firstName={'Roger'} lastName={'Whitney'} />
  );
};

class Cat extends Component {
  fullName() {
    return this.props.firstName + ' ' + this.props.lastName;
  }

  render() {
    return <Text style={{fontSize: 40}}>
      Hello, I am {this.fullName()}!
    </Text>;
  }
}
```

Calling JS Code



Hello, I am Roger Whitney!

## Using Functions

```
import React, {Component} from 'react';
import {Text} from 'react-native';
const App: () => React$Node = () => {
  return (
      <Cat firstName={'Roger'} lastName={'Whitney'} />
  );
};

const Cat = (props) => {
  return <Text style={{fontSize: 40}}>Hello, I am {fullName(props)}!</Text>;
};

function fullName(props) {
  return props.firstName + ' ' + props.lastName;
}
```

## State

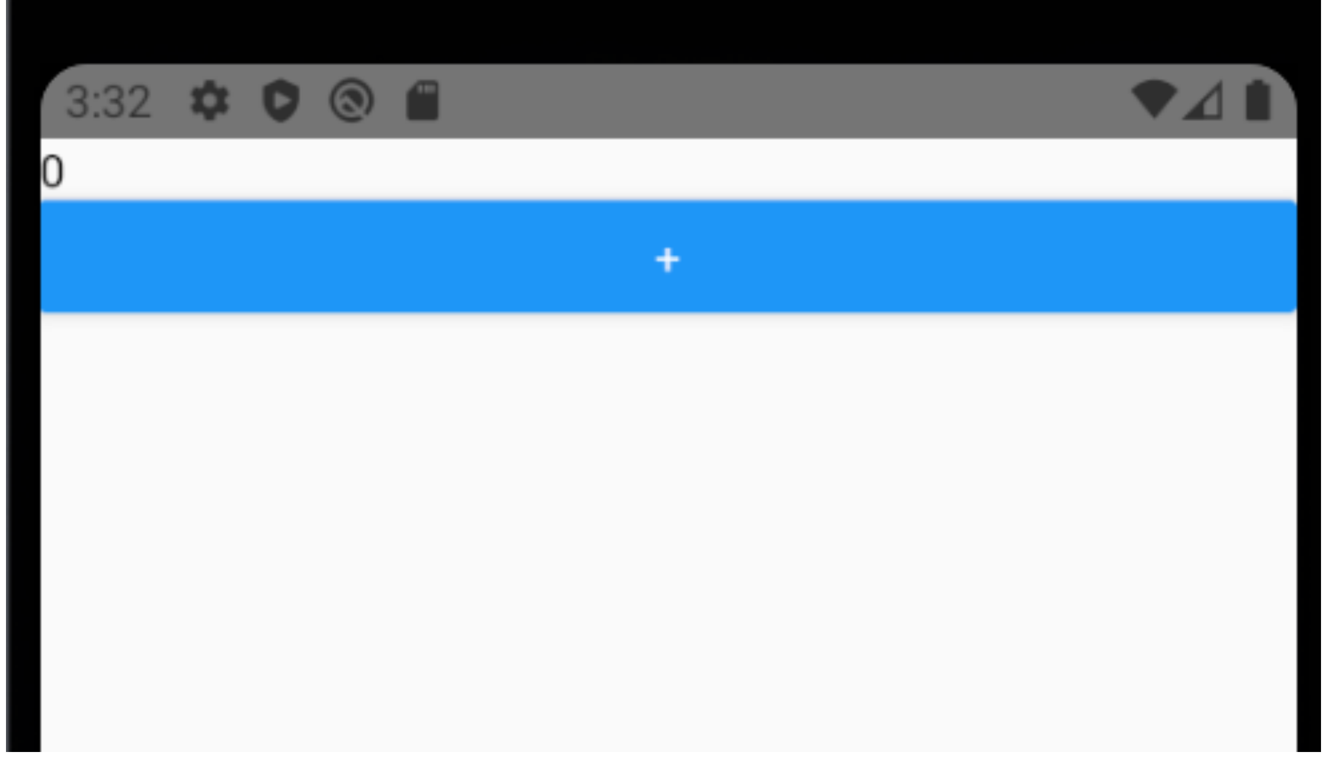
Data to be displayed

In components state is stored as object

    {firstName: 'Roger' lastName: 'Whitney'}

setState
    Used to update state

## State



```
const App: () => React$Node = () => {
  return <Counter />;
};

class Counter extends Component {
  state = {count: 0};
  render() {
    return (
      <View>
        <Text>{this.state.count}</Text>
        <Button onPress={() =>
                this.setState({count: this.state.count + 1})} title={'+'} />
      </View>
    );
  }
}
```

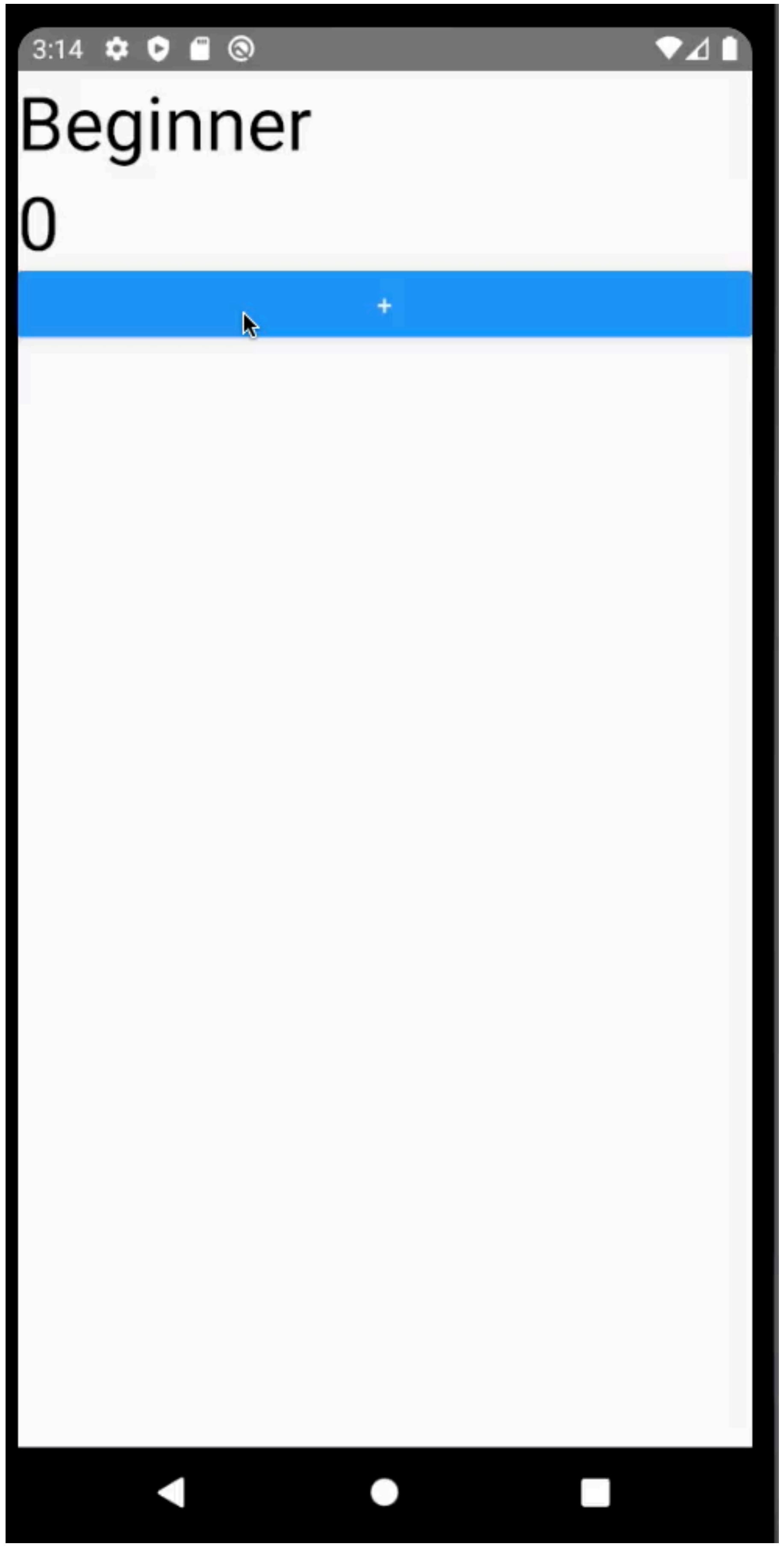
## State in Function

```
import React, {Component, useState} from 'react';
import {Text, Button, View} from 'react-native';

const App: () => React$Node = () => {
  return <CounterFunction />;
};
const CounterFunction = (props) => {
  const [count, setCount] = useState(0);

  return (
    <View>
      <Text>{count}</Text>
      <Button
        onPress={() => setCount(count + 1)}
        title={'+'}
      />
    </View>
  );
};
```

count
    The state
setCount
    function to change state
    Generated for you

## Multiple States

```
const CounterFunction = (props) => {
  const [count, setCount] = useState(0);
  const [level, setLevel] = useState('Beginner');

  function updateCount() {
    setCount(count + 1);
    if (count > 3) {
      setLevel('Advanced');
    }
  }
  return (
    <View>
      <Text style={{fontSize: 40}}>{level}</Text>
      <Text style={{fontSize: 40}}>{count}</Text>
      <Button onPress={updateCount} title="+" />
    </View>
  );
};
```


Beginner
0

## Button Props

onPress

title

- String

accessibilityLabel

color

disabled

nextFocusDown (Android, TV)

nextFocusForward (Android, TV)

nextFocusLeft (Android, TV)

nextFocusRight (Android, TV)

nextFocusUp (Android, TV)

testID

- Used to locate this view in end-to-end tests

## How to Make Button Text Big?

In Flutter everything is a widget

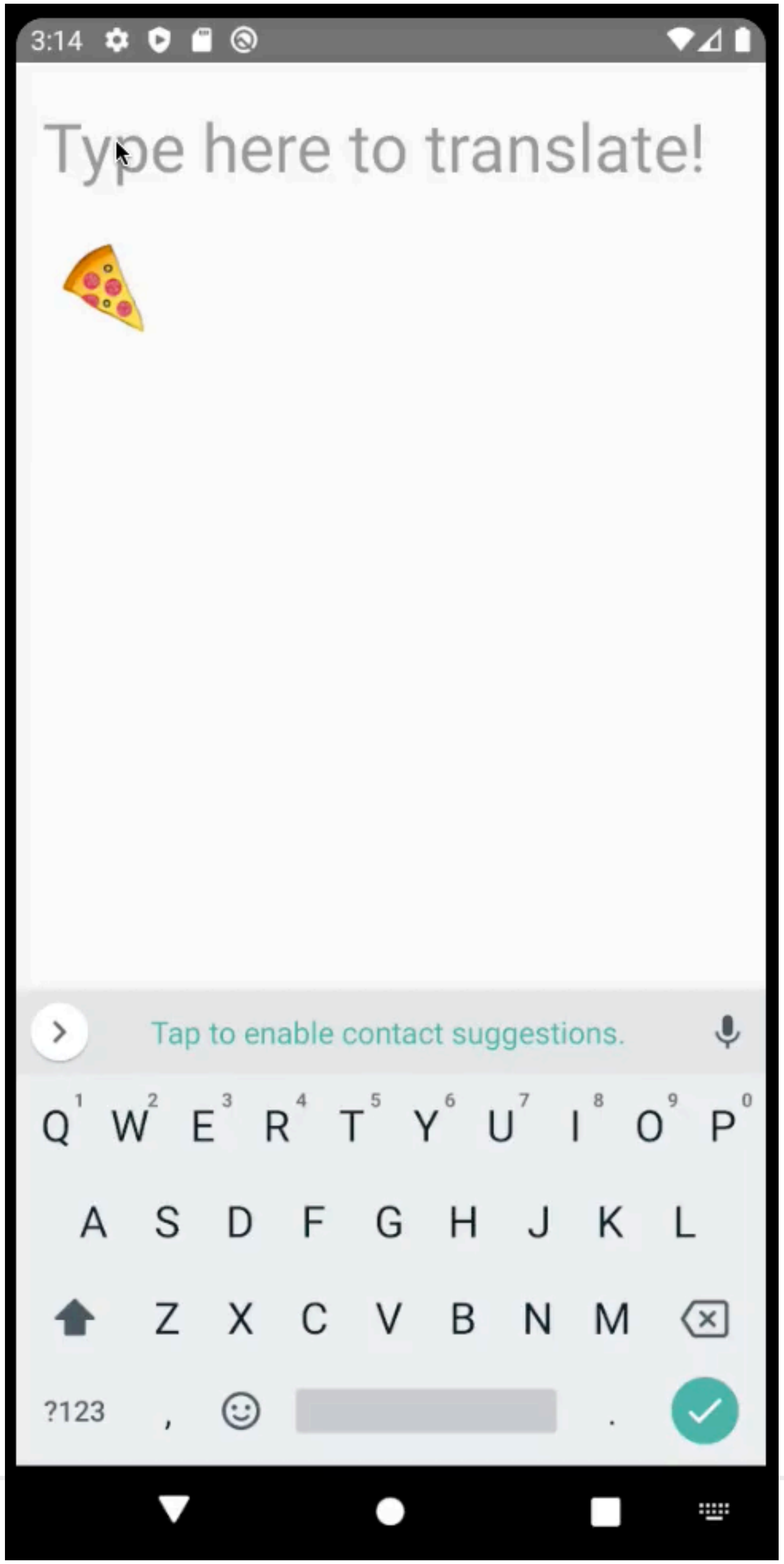
```
FlatButton(
        child: Text('Flat', style: TextStyle(fontSize: 30)),
        onPressed: null,)
```

React Native button title is a string

```
<Button onPress={updateCount} title="-" />
```

## TextInput

```
const PizzaTranslator = () => {
 const [text, setText] = useState('');
 return (
  <View style={{padding: 10}}>
   <TextInput
    style={{height: 40}}
    placeholder="Type here to translate!"
    onChangeText={(text) => setText(text)}
    defaultValue={text}
   />
   <Text style={{padding: 10, fontSize: 42}}>
    {text
     .split(' ')
     .map((word) => word + '🍕')
     .join(' ')}
   </Text>
  </View>
 );
};
```



## TextInput Props

Large number

A lot depend on platform

| Android only | iOS Only |
|---|---|
| autoCompleteType | clearButtonMode |
| disableFullscreenUI | clearTextOnFocus |
| importantForAutofill | dataDetectorTypes |
| inlineImageLeft | enablesReturnKeyAutomatically |
| inlineImagePadding | inputAccessoryViewID |
| numberOfLines | keyboardAppearance |
| textBreakStrategy | textContentType |
| underlineColorAndroid | |

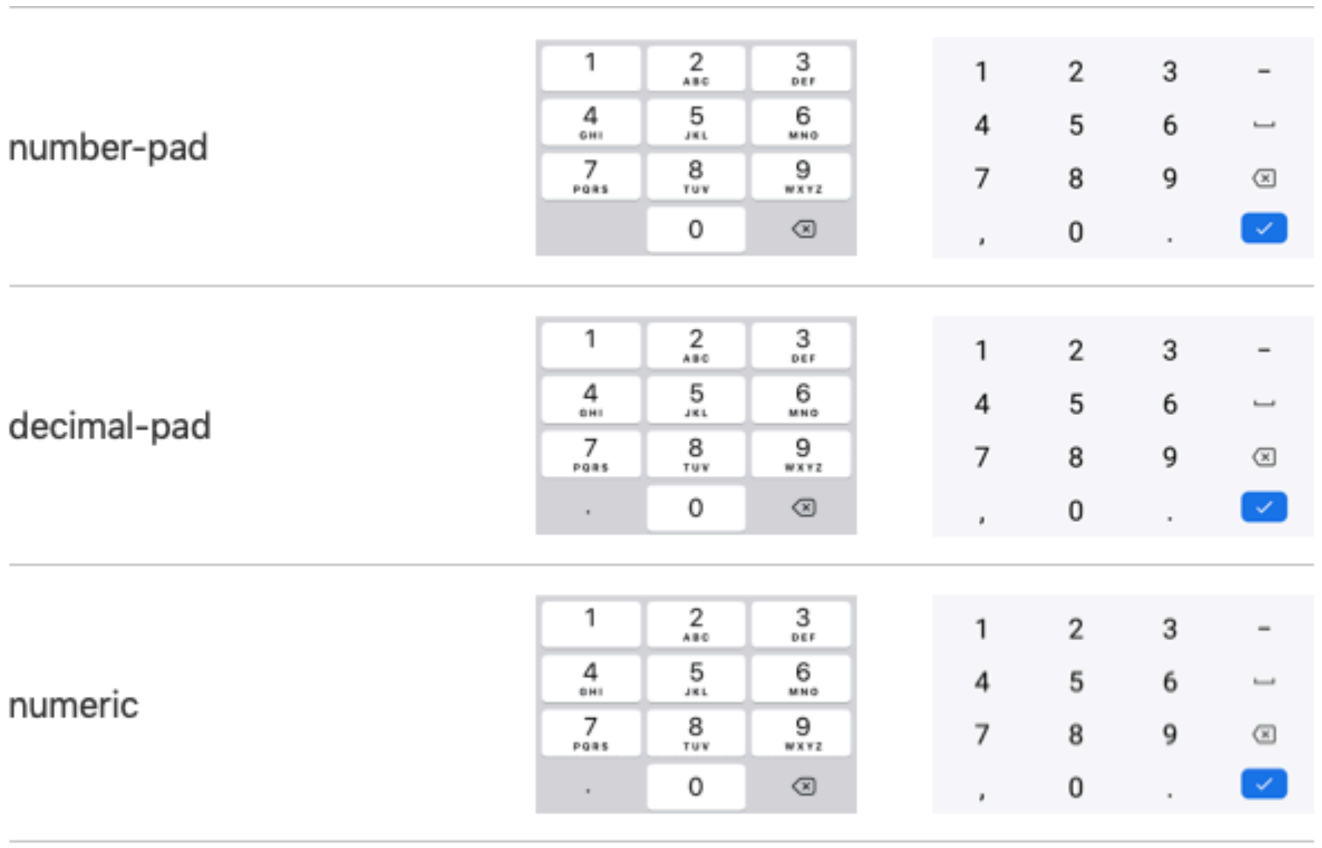
autoCapitalize

    enum('none', 'sentences', 'words', 'characters')

## TextInput Props - keyboardType

| All | iOS | Android |
|---|---|---|
| default | ascii-capable | visible-password |
| number-pad | numbers-and-punctuation | |
| decimal-pad | url | |
| numeric | name-phone-pad | |
| email-address | twitter | |
| phone-pad | web-search | |

| Flutter |
|---|
| text |
| multiline |
| number |
| phone |
| datetime |
| emailAddress |
| url |
| visiblePassword |
| name |
| address |



https://lefkowitz.me/visual-guide-to-react-native-textinput-keyboardtype-options/

## ScrollView

```
const logo = {
  uri: 'https://reactnative.dev/img/tiny_logo.png',
  width: 64,
  height: 64
};

export default App = () => (
  <ScrollView>
    <Text style={{ fontSize: 96 }}>Scroll me plz</Text>
    <Image source={logo} />
    <Image source={logo} />
    <Image source={logo} />
    <Image source={logo} />
    <Image source={logo} />
    <Text style={{ fontSize: 96 }}>If you like</Text>
    <Image source={logo} />
    <Image source={logo} />
    <Image source={logo} />
    <Image source={logo} />
    <Image source={logo} />
    <Text style={{ fontSize: 96 }}>Scrolling down</Text>
```

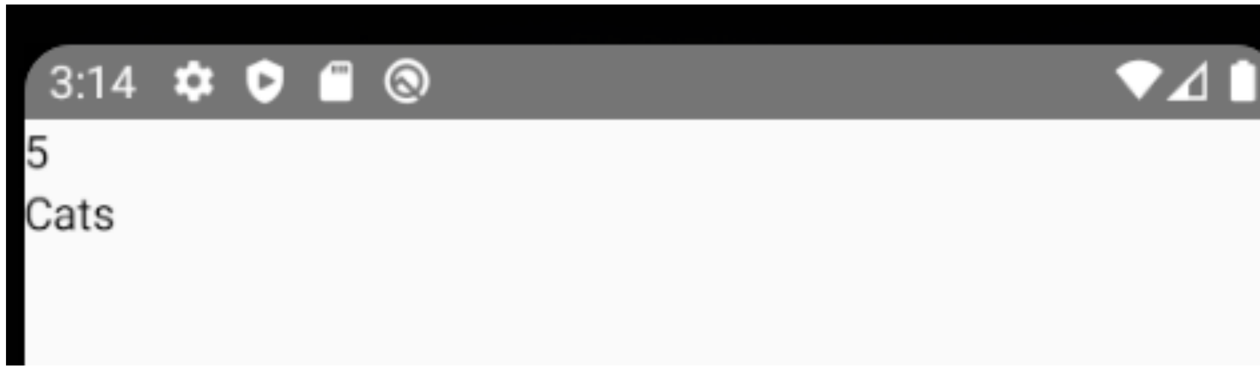
## render method

- Needs to examine this.props and this.state
- Pure
  - No side effects

- Can return
  - React elements.
    - Typically created via JSX
  - Arrays and fragments.
    - Return multiple elements from render.
  - Portals
    - Render children into a different DOM subtree
  - ~~String and numbers~~
  - Booleans or null.
    - Render nothing

## Examples

```
class App extends Component {
  render() {
    return [<Text>5</Text>,<Text>Cats</Text>];
  }
}
```


3:14

5
Cats

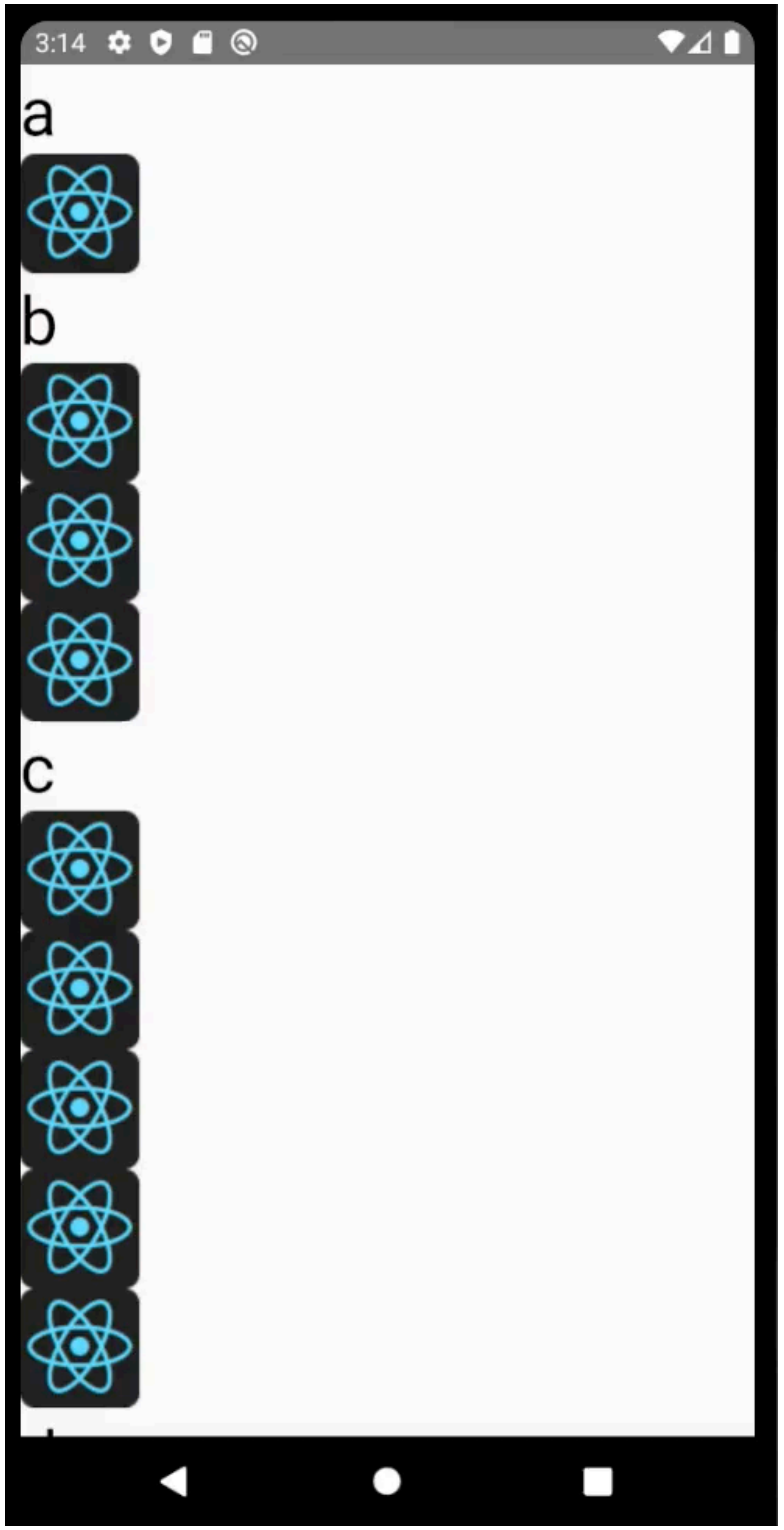
```
class App extends Component {
  render() {
    return (
      <React.Fragment>
        <Text>5</Text>
        <Text>Cats</Text>
      </React.Fragment>
    );
  }
}
```

## ScrollView without Repeats

```
import {Text, Image, ScrollView} from 'react-native';

const App: () => React$Node = () => {
  return <ScrollExample />;
};
const logo = {
  uri: 'https://reactnative.dev/img/tiny_logo.png',
  width: 64,
  height: 64,
};

function textImages(text, count) {
  let group = [];
  group.push(<Text key={text} style={{fontSize: 36}}>{text}</Text>);
  for (let k = 0; k < count; k++) {
    group.push(<Image key={text + k} source={logo} />);
  }
  return group;
}
```

## ScrollView without Repeats

```
const ScrollExample = () => {
  let textGroups = [];
  let count = 1;
  for (let message of ['a', 'b', 'c', 'd', 'e']) {
    textGroups.push(textImages(message, count++));
  }
  return (
    <ScrollView>
        {textGroups}
    </ScrollView>
  );
};

export default App;
```



## Use Array to Collect Components

```
function textImages(text, count) {
  let group = [];
  group.push(<Text key={text} style={{fontSize: 36}}>{text}</Text>);
  for (let k = 0; k < count; k++) {
    group.push(<Image key={text + k} source={logo} />);
  }
  return group;
}
```

## Keys

React considered that this was a list
Complained about lacking keys

```
function textImages(text, count) {
  let group = [];
  group.push(<Text key={text} style={{fontSize: 36}}>{text}</Text>);
  for (let k = 0; k < count; k++) {
    group.push(<Image key={text + k} source={logo} />);
  }
  return group;
}
```

## Keys

Like Flutter used to identify which items need to be rebuilt

Without keys adding to the beginning of the list will cause all items to be rerendered

```
<ul>
  <li>Duke</li>
  <li>Villanova</li>
</ul>
```

```
<ul>
  <li>Connecticut</li>
  <li>Duke</li>
  <li>Villanova</li>
</ul>
```

With key when adding current element are not rerendered

```
<ul>
  <li key="2015">Duke</li>
  <li key="2016">Villanova</li>
</ul>
```

```
<ul>
  <li key="2014">Connecticut</li>
  <li key="2015">Duke</li>
  <li key="2016">Villanova</li>
</ul>
```

## Keys

Keys used within arrays should be unique among their siblings

In the current implementation, you can express the fact that a subtree has been moved amongst its siblings, but you cannot tell that it has moved somewhere else. The algorithm will rerender that full subtree.

## FlatList

Fully cross-platform.
Optional horizontal mode.
Configurable viewability callbacks.
Header support.
Footer support.
Separator support.
Pull to Refresh.
Scroll loading.
ScrollToIndex support.
Multiple column support.

Only displays items on screen

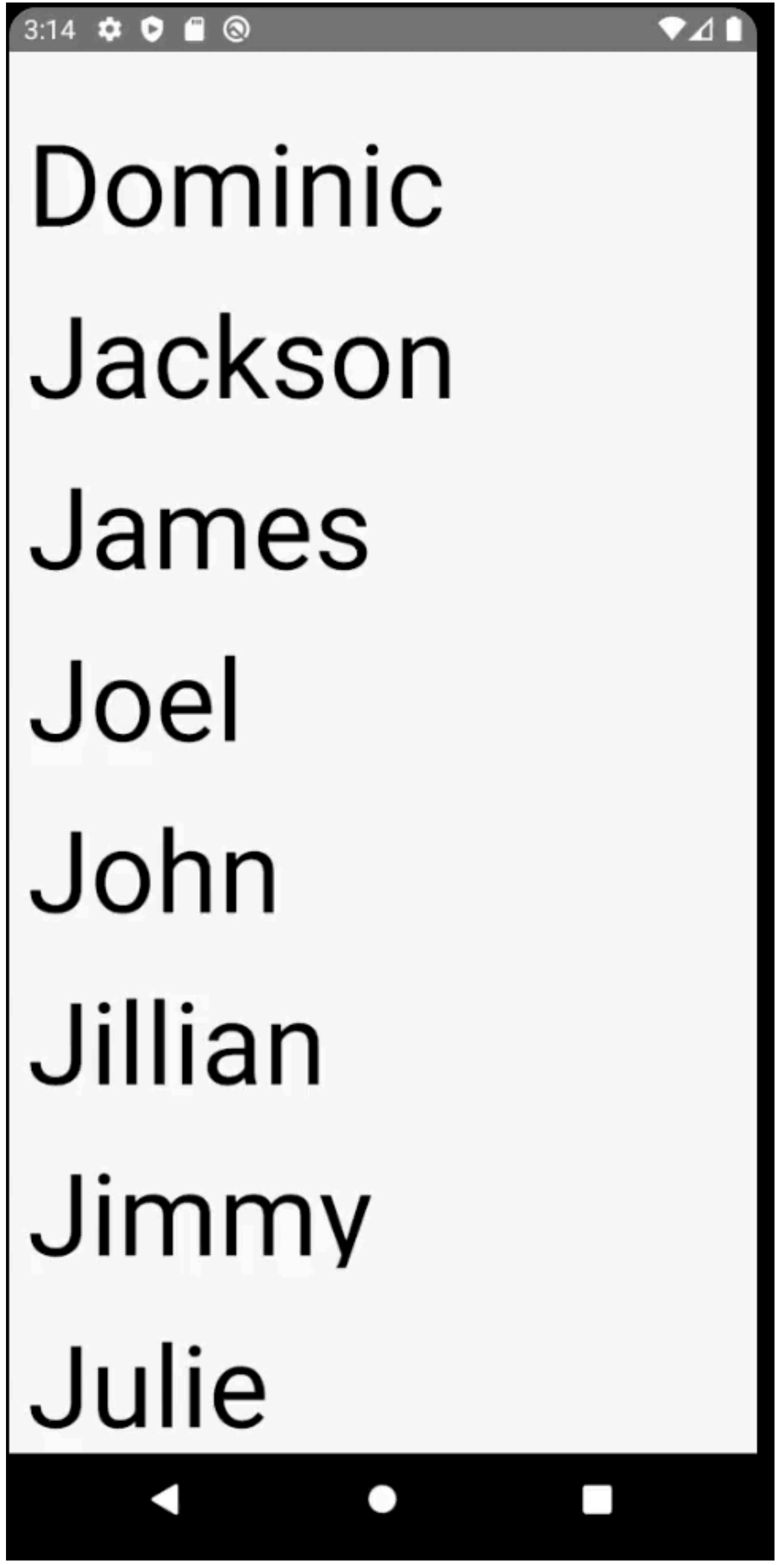
Needs
- data
- renderItem

## FlatList Example 1

```
import {Text, View, FlatList, StyleSheet} from 'react-native';

const App: () => React$Node = () => {
  return <FlatListBasics />;
};

const styles = StyleSheet.create({
  container: {
    flex: 1,
    paddingTop: 22,
  },
  item: {
    padding: 10,
    fontSize: 60,
    height: 90,
  },
});
```

## FlatList Example 1

```
const FlatListBasics = () => {
  return (
    <View style={styles.container}>
      <FlatList
        data={[
          {key: 'Devin'},
          {key: 'Dan'},
          {key: 'Dominic'},
          {key: 'Jackson'},
          {key: 'James'},
          {key: 'Joel'},
          {key: 'John'},
          {key: 'Jillian'},
          {key: 'Jimmy'},
          {key: 'Julie'},
        ]}
        renderItem={({item}) => <Text style={styles.item}>{item.key}</Text>}
      />
    </View>
  );
};
```



3:14

Dominic
Jackson
James
Joel
John
Jillian
Jimmy
Julie

## FlatList Example 1

```
const FlatListBasics = () => {
 return (
   <View style={styles.container}>
     <FlatList
       data={[
         {key: 'Devin'},
         {key: 'Dan'},
         {key: 'Dominic'},
         {key: 'Jackson'},
         {key: 'James'},
         {key: 'Joel'},
         {key: 'John'},
         {key: 'Jillian'},
         {key: 'Jimmy'},
         {key: 'Julie'},
       ]}
       renderItem={({item}) => <Text style={styles.item}>{item.key}</Text>}
     />
   </View>
 );
};
```

If you don't provide a key
React looks for a key prop on each item

Or add a keyExtractor prop

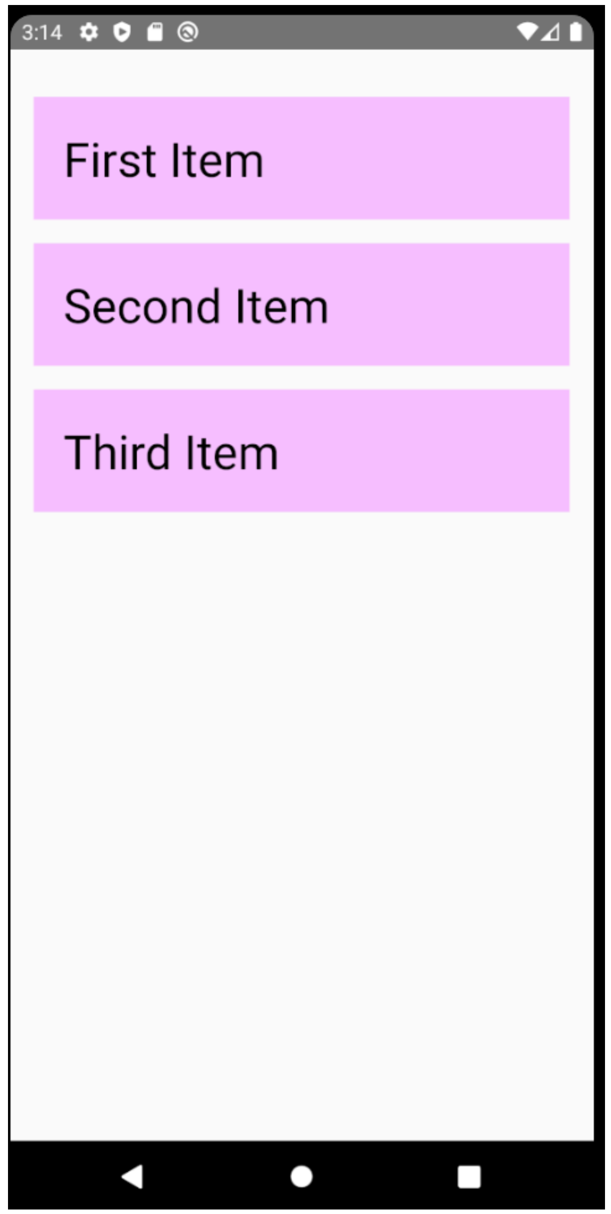
data, renderItem
    Required

## FlatList props

**renderItem**
**data**
**ItemSeparatorComponent**
ListEmptyComponent
ListFooterComponent
ListFooterComponentStyle
ListHeaderComponent
ListHeaderComponentStyle
columnWrapperStyle
**extraData**
getItemLayout
**horizontal**
initialNumToRender
initialScrollIndex
inverted
**keyExtractor**
**numColumns**
onEndReached
onEndReachedThreshold
onRefresh
onViewableItemsChanged
progressViewOffset
legacyImplementation
refreshing
removeClippedSubviews
viewabilityConfig
viewabilityConfigCallbackPairs

## FlatList Example 2

```
import React from 'react';
import {SafeAreaView, View, FlatList, StyleSheet, Text, StatusBar, } from 'react-native';

const styles = StyleSheet.create({
  container: {
    flex: 1,
    marginTop: StatusBar.currentHeight || 0,
  },
  item: {
    backgroundColor: '#f9c2ff',
    padding: 20,
    marginVertical: 8,
    marginHorizontal: 16,
  },
  title: {
    fontSize: 32,
  },
});
```

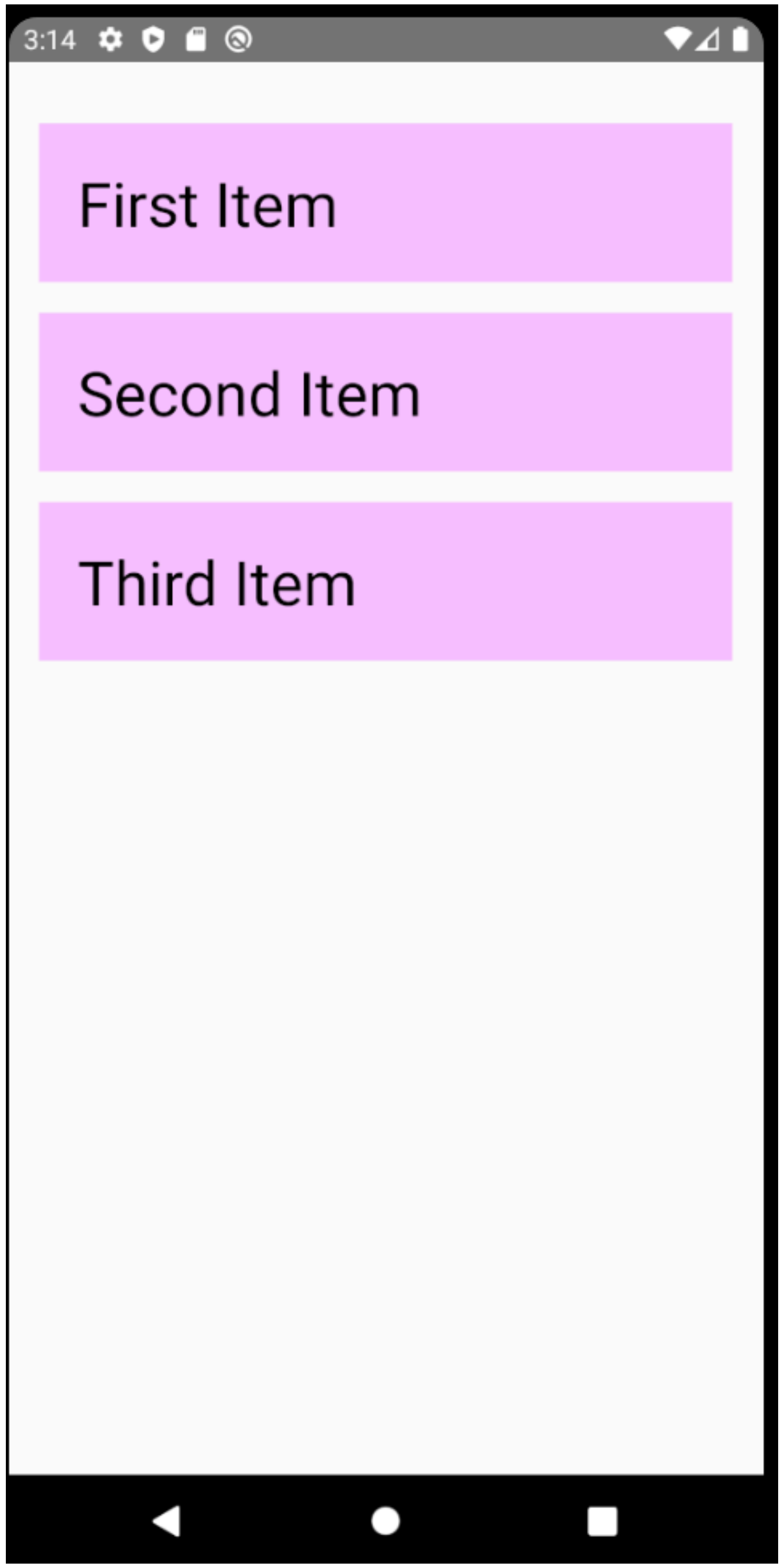


## FlatList Example 2

```
const DATA = [
  {
    id: 'bd7acbea-c1b1-46c2-aed5-3ad53abb28ba',
    title: 'First Item',
  },
  {
    id: '3ac68afc-c605-48d3-a4f8-fbd91aa97f63',
    title: 'Second Item',
  },
  {
    id: '58694a0f-3da1-471f-bd96-145571e29d72',
    title: 'Third Item',
  },
];
```

## keyExtractor

```
const Item = ({title}) => (
  <View style={styles.item}>
    <Text style={styles.title}>{title}</Text>
  </View>
);

const App = () => {
  const renderItem = ({item}) => <Item title={item.title} />;

  return (
    <SafeAreaView style={styles.container}>
      <FlatList
        data={DATA}
        renderItem={renderItem}
        keyExtractor={(item) => item.id}
      />
    </SafeAreaView>
  );
};
```


3:14

First Item

Second Item

Third Item

## Component Lifecycle Methods

Mounting
- constructor()
- static getDerivedStateFromProps()
- render()
- componentDidMount()

Updating
- static getDerivedStateFromProps()
- shouldComponentUpdate()
- render()
- getSnapshotBeforeUpdate()
- componentDidUpdate()

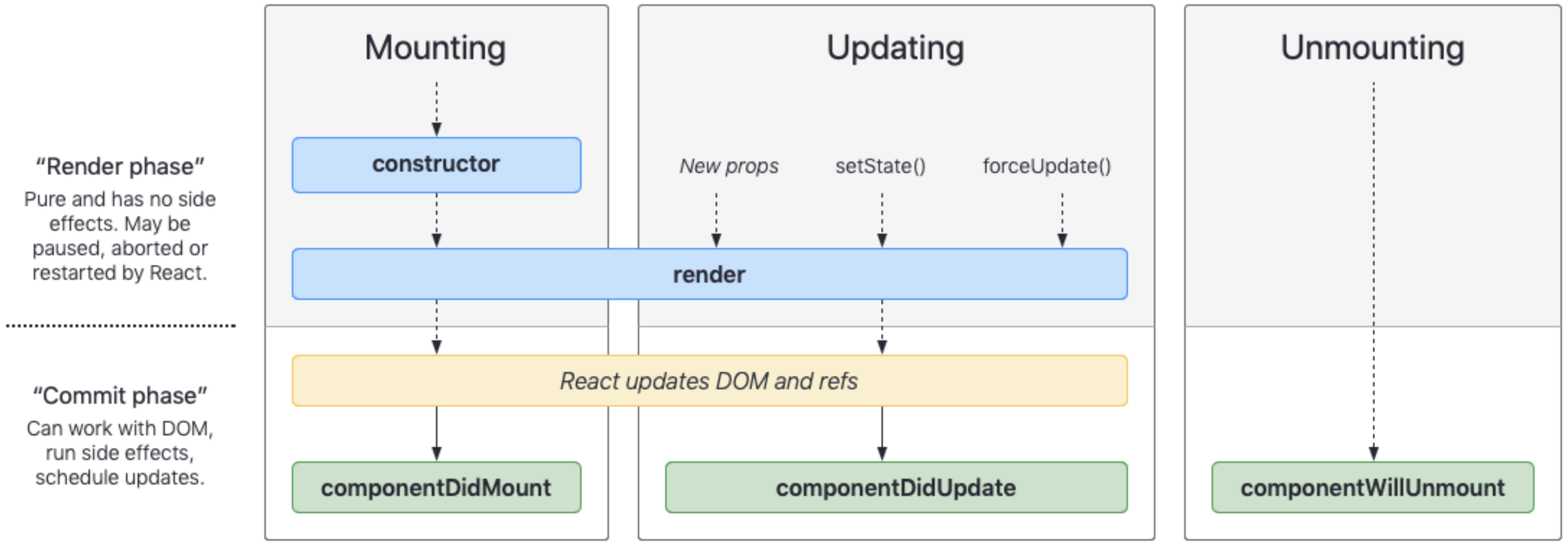
Unmounting
- componentWillUnmount()

Error Handling
- static getDerivedStateFromError()
- componentDidCatch()

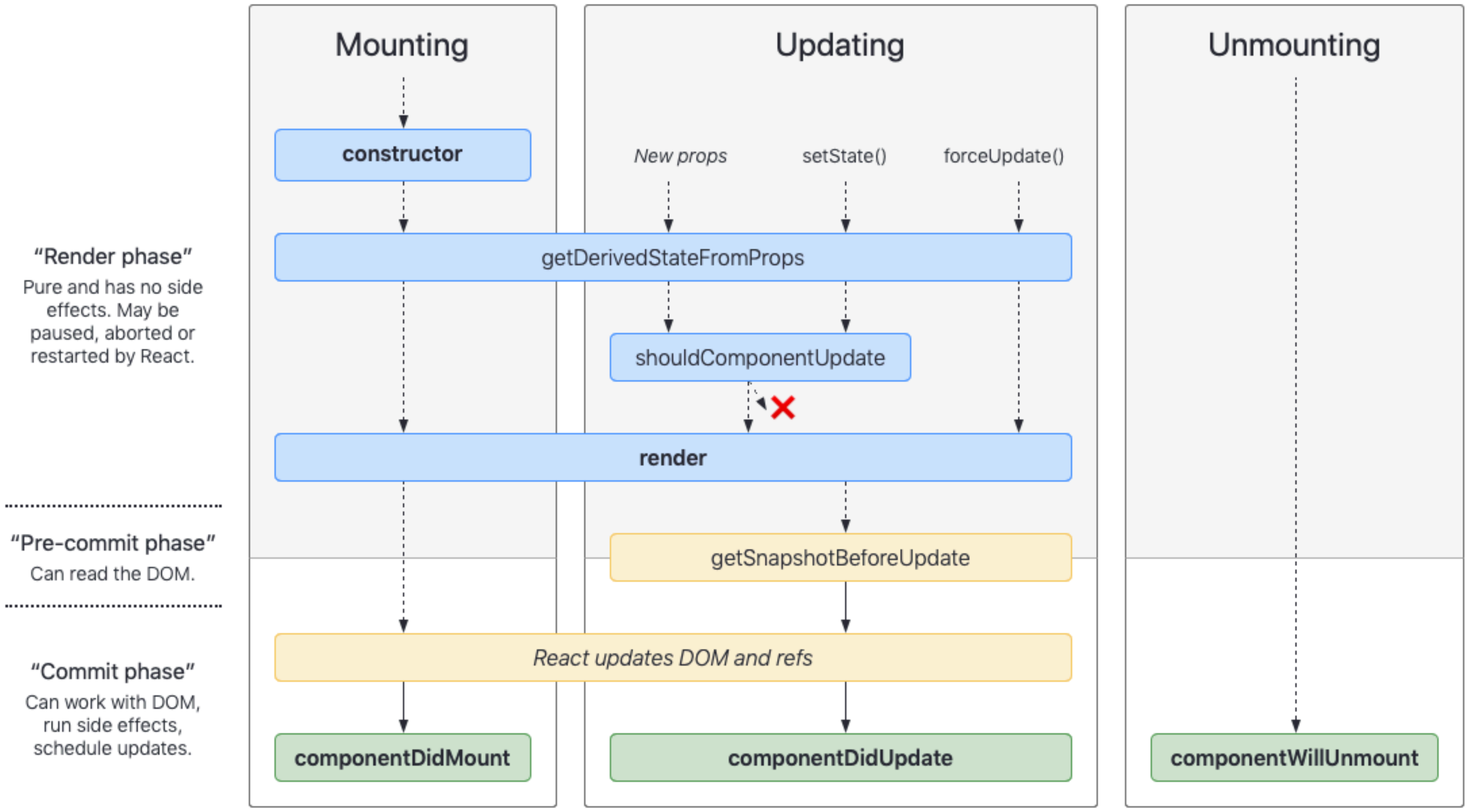
Other APIs
- setState()
- forceUpdate()

Hooks
- Lifecycle for functions as components

## Main Lifecycle Methods



# All Lifecycle Methods



## shouldComponentUpdate()

Called on component to determine if it needs to be rendered

If returns false

- not rendered

## Hooks - Lifecycle Methods for functions

Basic Hooks
- useState
- useEffect
- useContext

Additional Hooks
- useReducer
- useCallback
- useMemo
- useRef
- useImperativeHandle
- useLayoutEffect
- useDebugValue

## Types of Components

React.Component

Does not implement shouldComponentUpdate()

React.PureComponent

Does implement shouldComponentUpdate()

Shallowly compares objects

```
{'instructor': {'name': 'Whitney',
                'location': {'building':'GMCS', 'room':516}}
                            ↓
{'instructor': {'name': 'Whitney',
                'location': {'building':'GMCS', 'room':316}}
```

Skips prop updates for the whole component subtree

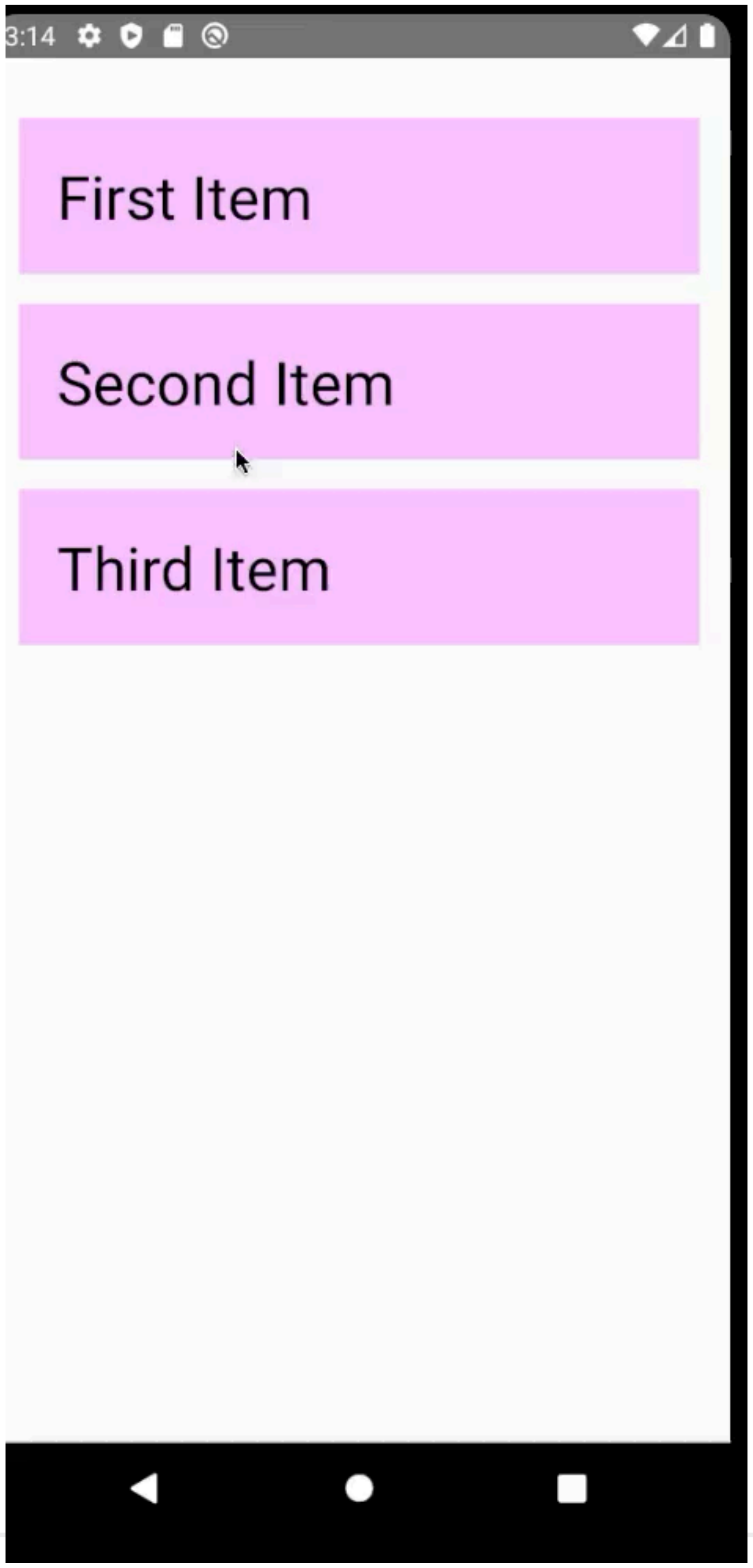
## FlatList & State

FlatList is a PureComponent so only shallow compare on props for re-rendering

Content is rendered asynchronously
- So can scroll faster than React can render

If depends on state
- Use extraData -
    - lists the data that list depends on
    - When data changes list is re-rendered

## FlatList & State Example


3:14

First Item

Second Item

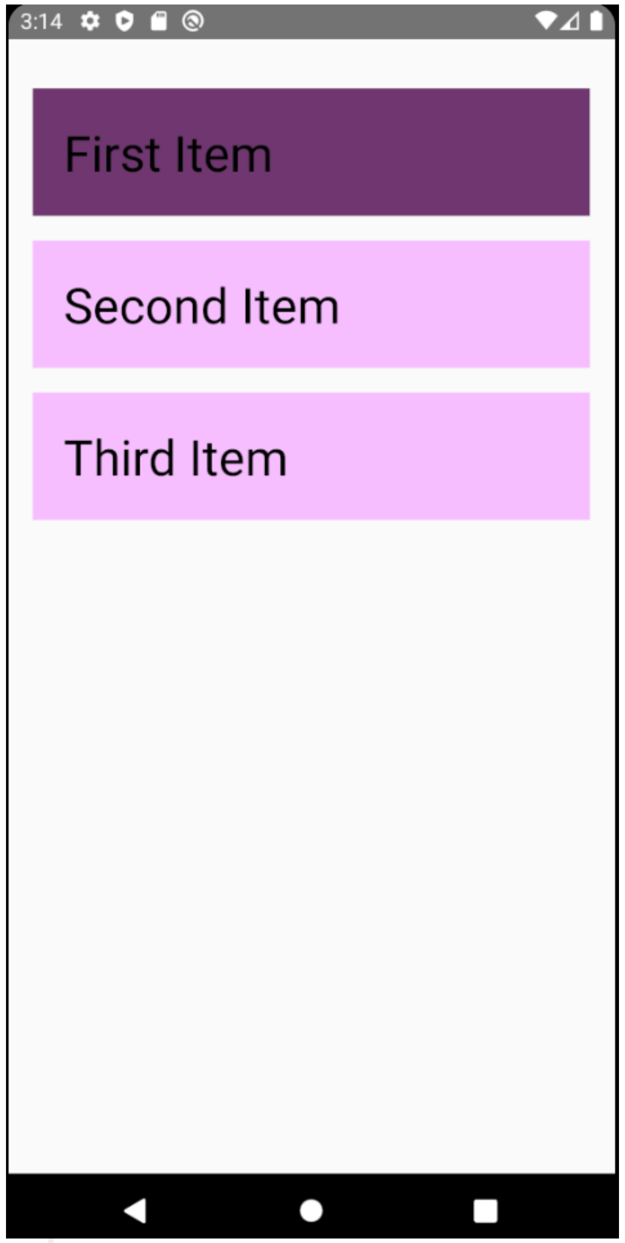
Third Item

## FlatList & State Example

```
import React, {useState} from 'react';
import {FlatList, SafeAreaView, StatusBar, StyleSheet, Text, TouchableOpacity,
} from 'react-native';

const DATA = [
  {
    id: 'bd7acbea-c1b1-46c2-aed5-3ad53abb28ba',
    title: 'First Item',
  },
  {
    id: '3ac68afc-c605-48d3-a4f8-fbd91aa97f63',
    title: 'Second Item',
  },
  {
    id: '58694a0f-3da1-471f-bd96-145571e29d72',
    title: 'Third Item',
  },
];
```

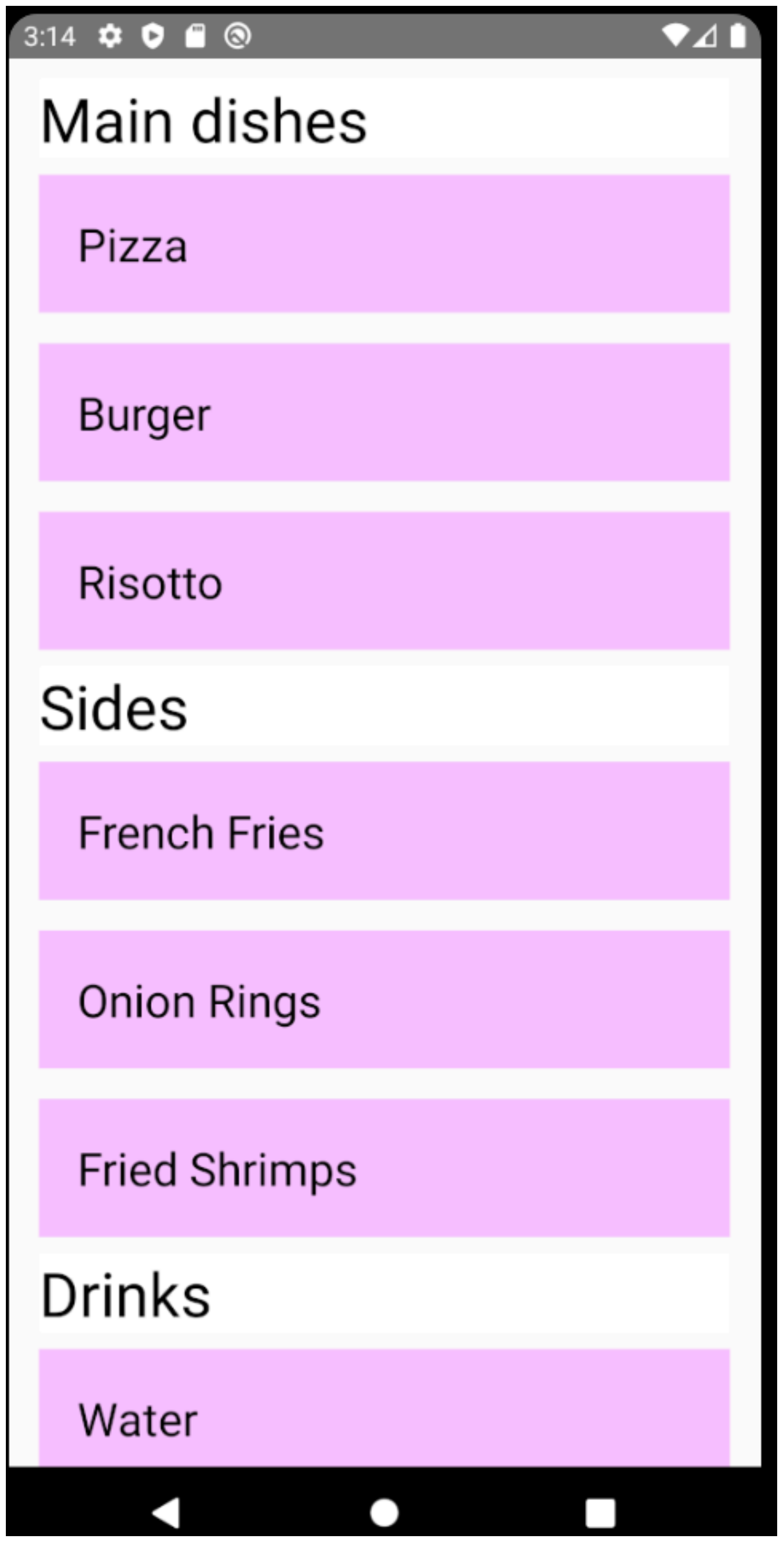
```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    marginTop: StatusBar.currentHeight || 0,
  },
  item: {
    padding: 20,
    marginVertical: 8,
    marginHorizontal: 16,
  },
  title: {
    fontSize: 32,
  },
});
```

```
const App = () => {
  const [selectedId, setSelectedId] = useState(null);

  const renderItem = ({item}) => {
    const backgroundColor = item.id === selectedId ? '#6e3b6e' : '#f9c2ff';

    return (
      <Item
        item={item}
        onPress={() => setSelectedId(item.id)}
        style={{backgroundColor}}
      />
    );
  };

  return (
    <SafeAreaView style={styles.container}>
      <FlatList
        data={DATA}
        renderItem={renderItem}
        keyExtractor={(item) => item.id}
        extraData={selectedId}
      />
    </SafeAreaView>
  );
};
```



## SectionList

```
const DATA = [
  {
    title: 'Main dishes',
    data: ['Pizza', 'Burger', 'Risotto'],
  },
  {
    title: 'Sides',
    data: ['French Fries', 'Onion Rings', 'Fried Shrimps'],
  },
  {
    title: 'Drinks',
    data: ['Water', 'Coke', 'Beer'],
  },
  {
    title: 'Desserts',
    data: ['Cheese Cake', 'Ice Cream'],
  },
];
```


3:14

Main dishes

Pizza

Burger

Risotto

Sides

French Fries

Onion Rings

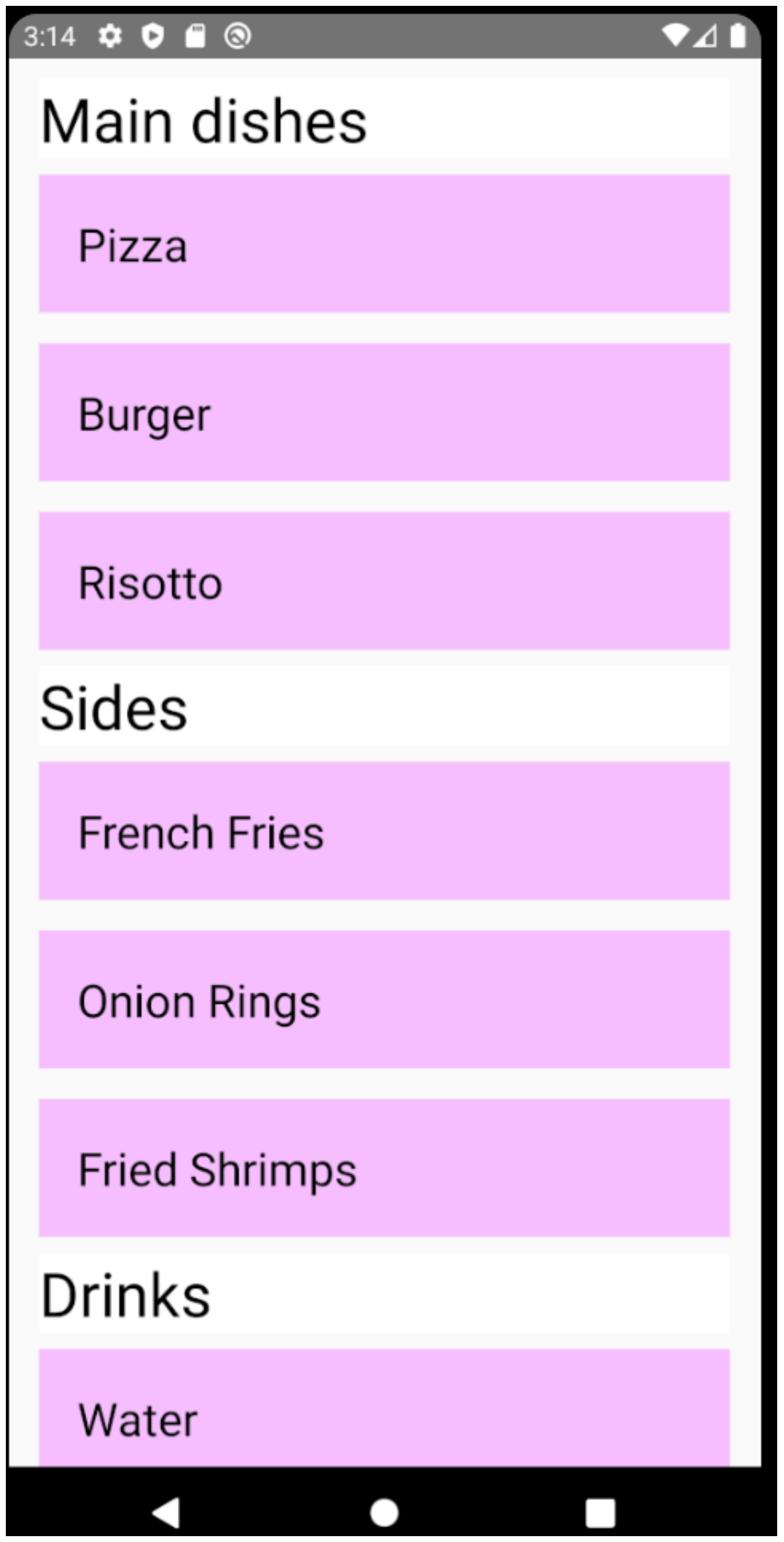
Fried Shrimps

Drinks

Water

## SectionList

```
const Item = ({title}) => (
  <View style={styles.item}>
    <Text style={styles.title}>{title}</Text>
  </View>
);

const App = () => (
  <SafeAreaView style={styles.container}>
    <SectionList
      sections={DATA}
      keyExtractor={(item, index) => item + index}
      renderItem={({item}) => <Item title={item} />}
      renderSectionHeader={({section: {title}}) => (
        <Text style={styles.header}>{title}</Text>
      )}
    />
  </SafeAreaView>
);
```


3:14
Main dishes
Pizza
Burger
Risotto
Sides
French Fries
Onion Rings
Fried Shrimps
Drinks
Water