React Native

Navigation

Lifecycle

Navigation Lifecycle

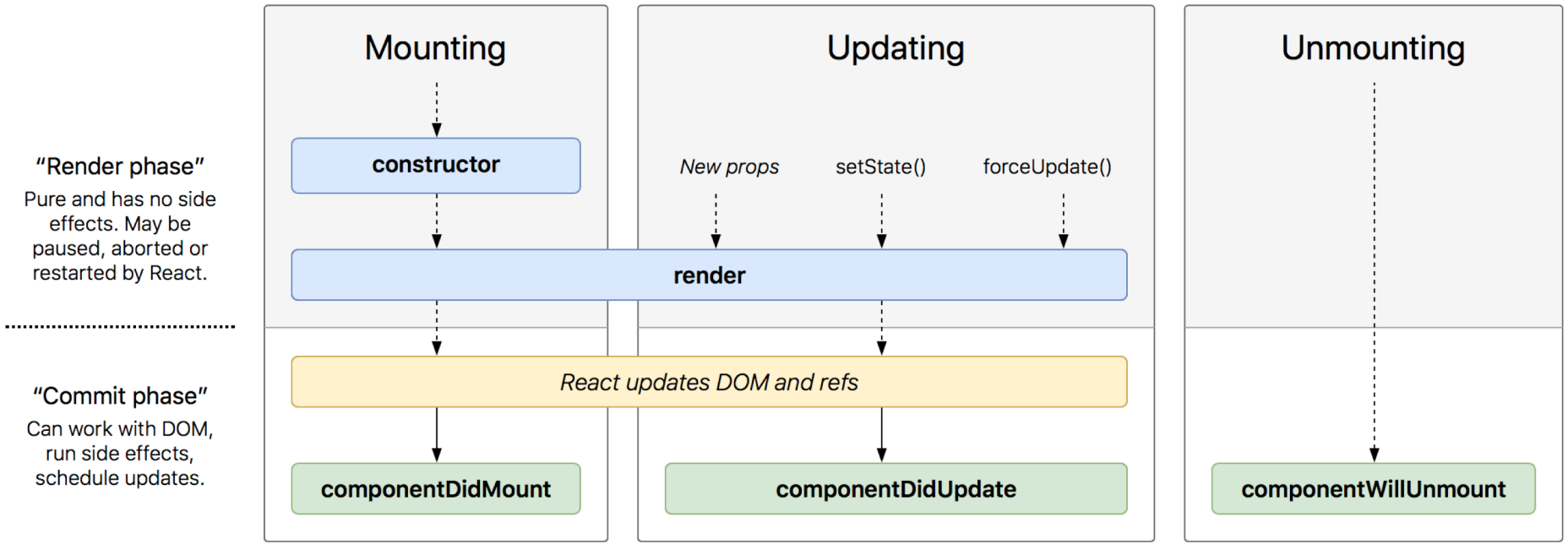
- The navigation library follows the normal React Native lifecycle.
- It also provides it’s own lifecycle functions.

Navigation Lifecycle

- What happens with Home when we navigate away from it, or when we come back to it?
- How does a route find out that a user is leaving it or coming back to it?
- Two important React lifecycle calls are used:
  - **`componentDidMount`**
  - **`componentWillUnmount`**
- These will become important when we work with a repository like Redux

Component Lifecycle (normal)

- Each component has several “lifecycle methods” that you can override to run code at particular times in the process.



See project on GitHub

Mounting components

- Consider a stack navigator with screens A and B.
- After navigating to A, A's **`componentDidMount`** is called.
- When pushing B, B's **`componentDidMount`** is also called,
- but A remains mounted on the stack and its **`componentWillUnmount`** is therefore not called.
- When going back from B to A, **`componentWillUnmount`** of B is called since B is popped off the stack
- but **`componentDidMount`** of A is not called because A remained mounted the whole time.

React Navigation

- React Navigation emits events to screen components that subscribe to them.
- There are four different events that you can subscribe to:
  - willFocus - the screen will focus
  - didFocus - the screen focused (if there was a transition, the transition completed)
  - willBlur - the screen will be unfocused
  - didBlur - the screen unfocused (if there was a transition, the transition completed)
- Event listeners are added by navigation.addListener(…).
- API reference: https://reactnavigation.org/docs/en/navigation-prop.html#addlistener-subscribe-to-updates-to-navigation-lifecycle

addListener

```
const didBlurSubscription = this.props.navigation.addListener(
  'didBlur',
  payload => {
    console.debug('didBlur', payload);
  }
);

// Remove the listener when you are done
didBlurSubscription.remove();
```

The payload argument is configured as:

```
{
  action: { type: 'Navigation/COMPLETE_TRANSITION', key: 'StackRouterRoot' },
  context: 'id-1518521010538-2:Navigation/COMPLETE_TRANSITION_Root',
  lastState: undefined,
  state: undefined,
  type: 'didBlur',
};
```

Example

```
 constructor(props){
      super(props);
      const willFocusSubscription = this.props.navigation.addListener(
           'willFocus',
           this._updateState
      );
 }

 _updateState = payload => {
      Alert.alert('received willFocus!');
}
```

isFocused

- Returns **`true`** if the screen is focused and **`false`** otherwise.

```
let isFocused = this.props.navigation.isFocused();
```

- You probably want to use **`withNavigationFocus`** instead of using this directly, it will pass in an **`isFocusedboolean`** a prop to your component.

state - The screen's current state/route

A screen has access to its route via **`this.props.navigation.state`**. It will return an object with the following:

```
{
  // the name of the route config in the router
  routeName: 'profile',
  //a unique identifier used to sort routes
  key: 'main0',
  //an optional object of string options for this screen
  params: { hello: 'world' }
}
```

- This is most commonly used to access the params for the screen, passed in through navigate or **`setParams`**.

```
class ProfileScreen extends React.Component {
    render() {
        return <Text>Name: {this.props.navigation.state.params.name}</Text>;
    }
}
```