DANIEL GRAHAM PHD

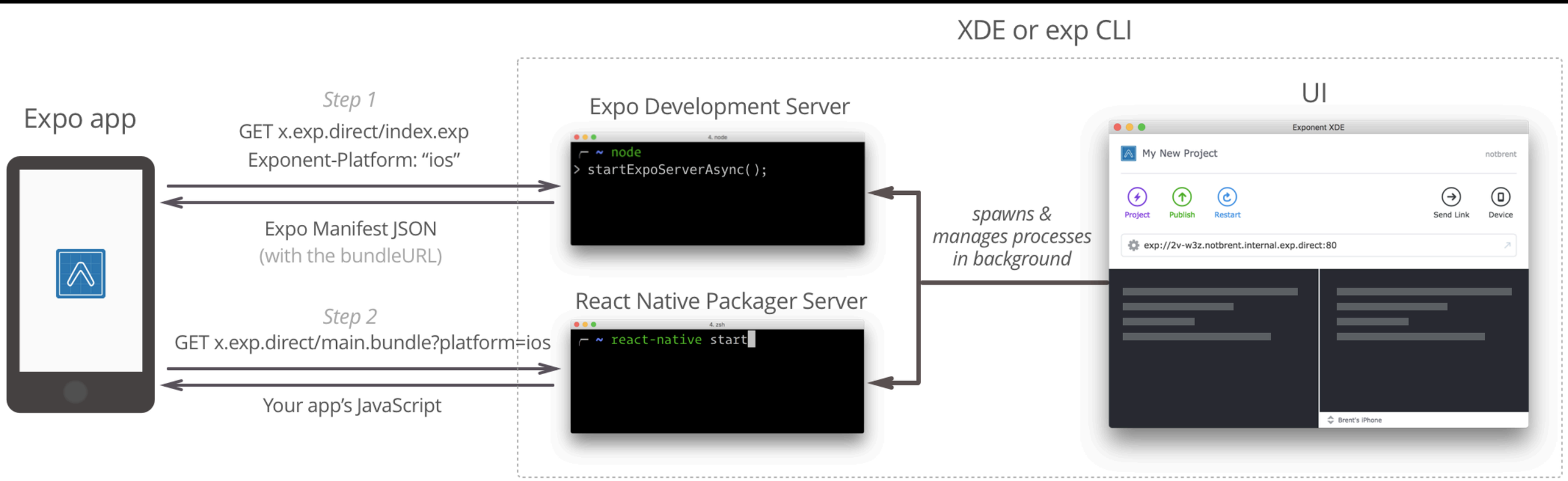
REACT NATIVE DEVELOPMENT

```
  class Clock extends React.Component{
   render(){
        let clockElement = <div>
                          <h1>This is a { this.props.type} Clock</h1>
                          <h1>The Time is Now : {this.props.time}</h1>
                      </div>
        return  clockElement
    }
  }
```

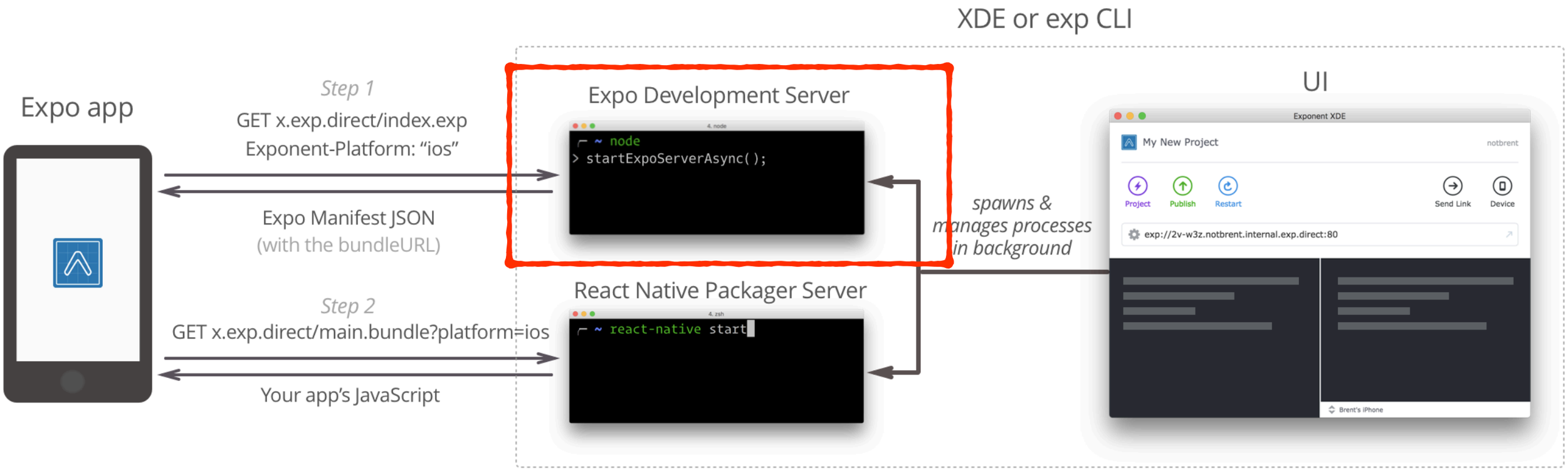
Watch The Video Below

https://www.youtube.com/watch?v=IQI9aUlouMI

HOW EXPO WORKS

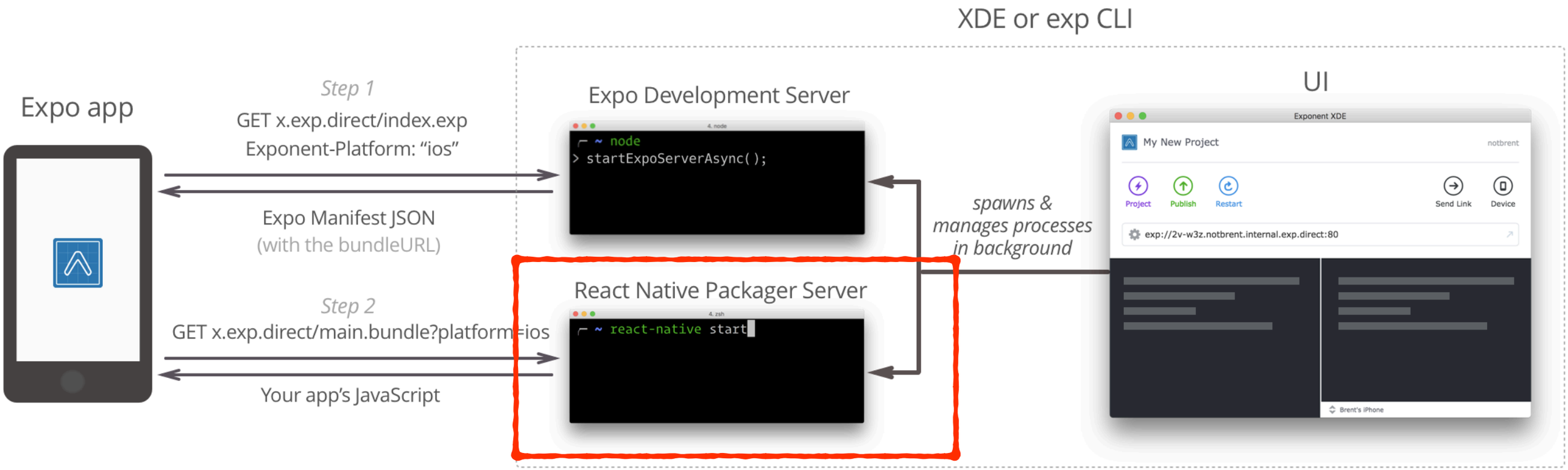


HTTPS://GITHUB.COM/EXPO/EXPO



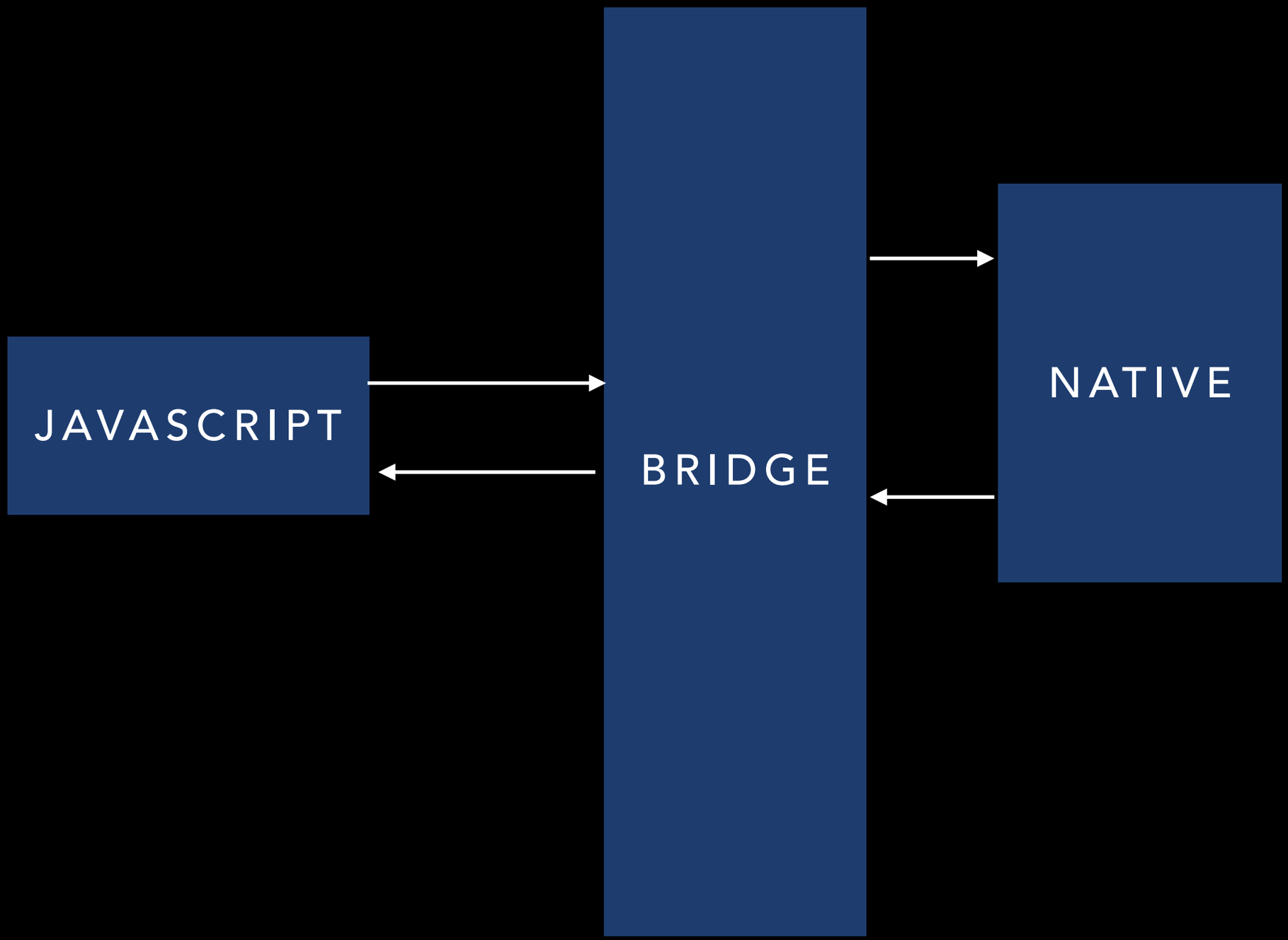
Serves The Expo manifest file

Tells Expo The configuration that App Needs
Also includes a link to JS bundle to download



Supply the Javascript

But How is that Javascript run?



HTTPS://WWW.YOUTUBE.COM/WATCH?V=RRECZR6DMEM

LET'S SWITCH OVER TO OUR NEW ENVIRONMENT

HTTPS://SNACK.EXPO.IO/

I will post snacks on course website

https://expo.io/learn

Setting up local dev Environment

Navigate to a new directory

```
npm install expo-cli --global

expo init my-new-project
cd my-new-project
expo start
```

CHOOSE BLANK AND MANAGED

App.js — InClassCode

EXPLORER

OPEN EDITO... 1 UNSAVED

- App.js inc... 6, M
- Untitled-1
- index.js incl... M
- App.js ~/Doc... U

INCL...

- inclass-app
  - node_modules
  - public
    - favicon.ico
    - index.html
    - manifest.json
  - src
    - App.css
    - App.js 6, M
    - App.js.zip U
    - App.test.js
    - index.css
    - index.js M
    - logo.svg

OUTLINE

App.js ~/.../ackeePod

```
1  import React from 'react';
2  import { StyleSheet, Text, View }
3
4  export default class App extends
5    render() {
6      return (
7        <View style={styles.containe
8          <Text>Open up App.js to st
9        </View>
10     );
11   }
12 }
13
14 const styles = StyleSheet.create(
15   container: {
16     flex: 1,
17     backgroundColor: '#fff',
18     alignItems: 'center',
19     justifyContent: 'center',
20   },
21 });
22
```

master* Ln 4, Col 1 (207 selected) Spaces: 2 UTF-8 LF JavaScript

localhost

Metro Bundler

PROCESS (3) - 3:35:34 PM

iPhone

DEVICE (1) - 3:35:35 PM

Run on Android device/emulator

Run on iOS simulator

Send link with email/SMS...

Publish or republish project...

PRODUCTION MODE

CONNECTION Tunnel LAN Local

exp://172.25.143.77:19000

LOGGED IN AS

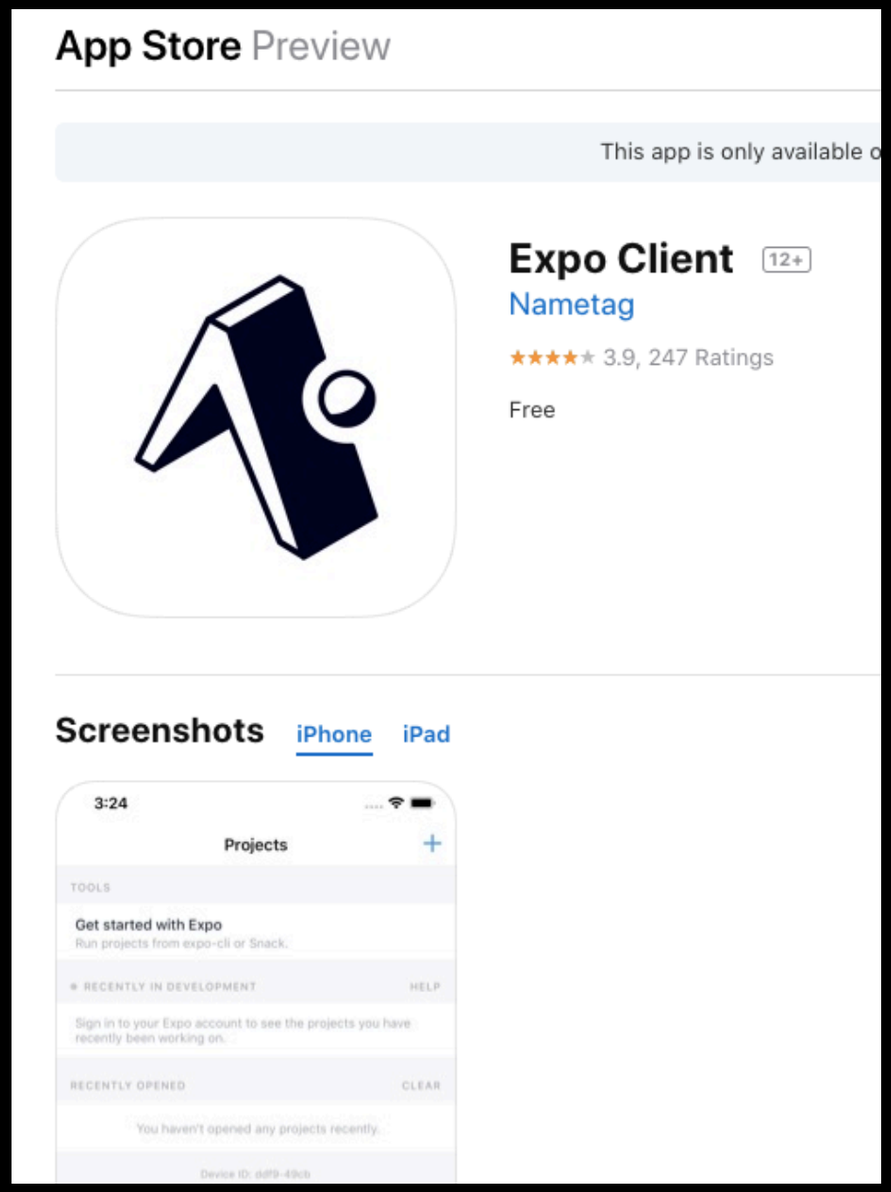
METRO BUNDLER

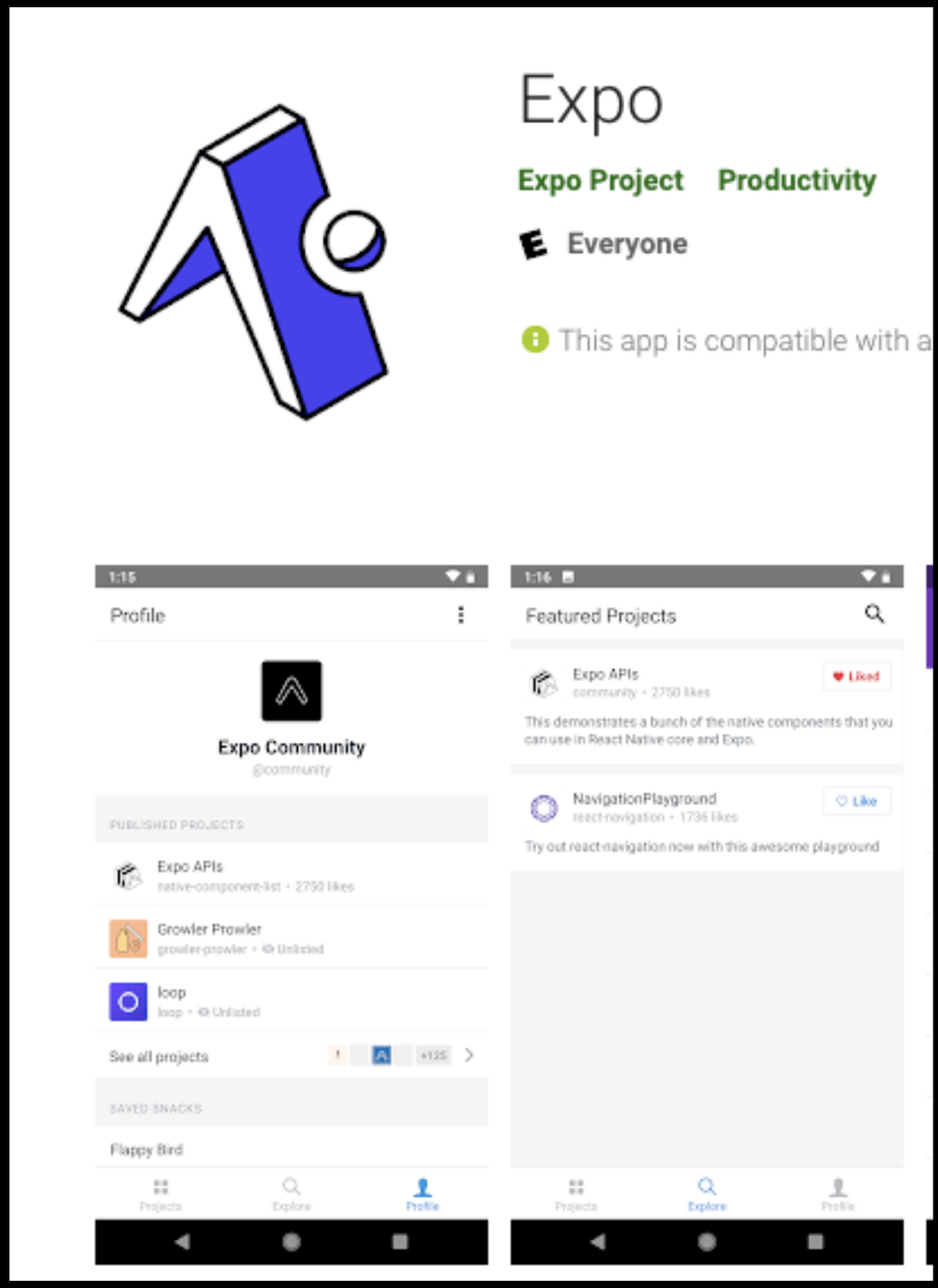
INFO 15:22 Starting Metro Bundler on port 19001.

INFO 15:22 Tunnel ready.

INFO 15:35 Building JavaScript bundle: finished in 28864ms.

INSTALL EXPO ON YOUR DEVICE





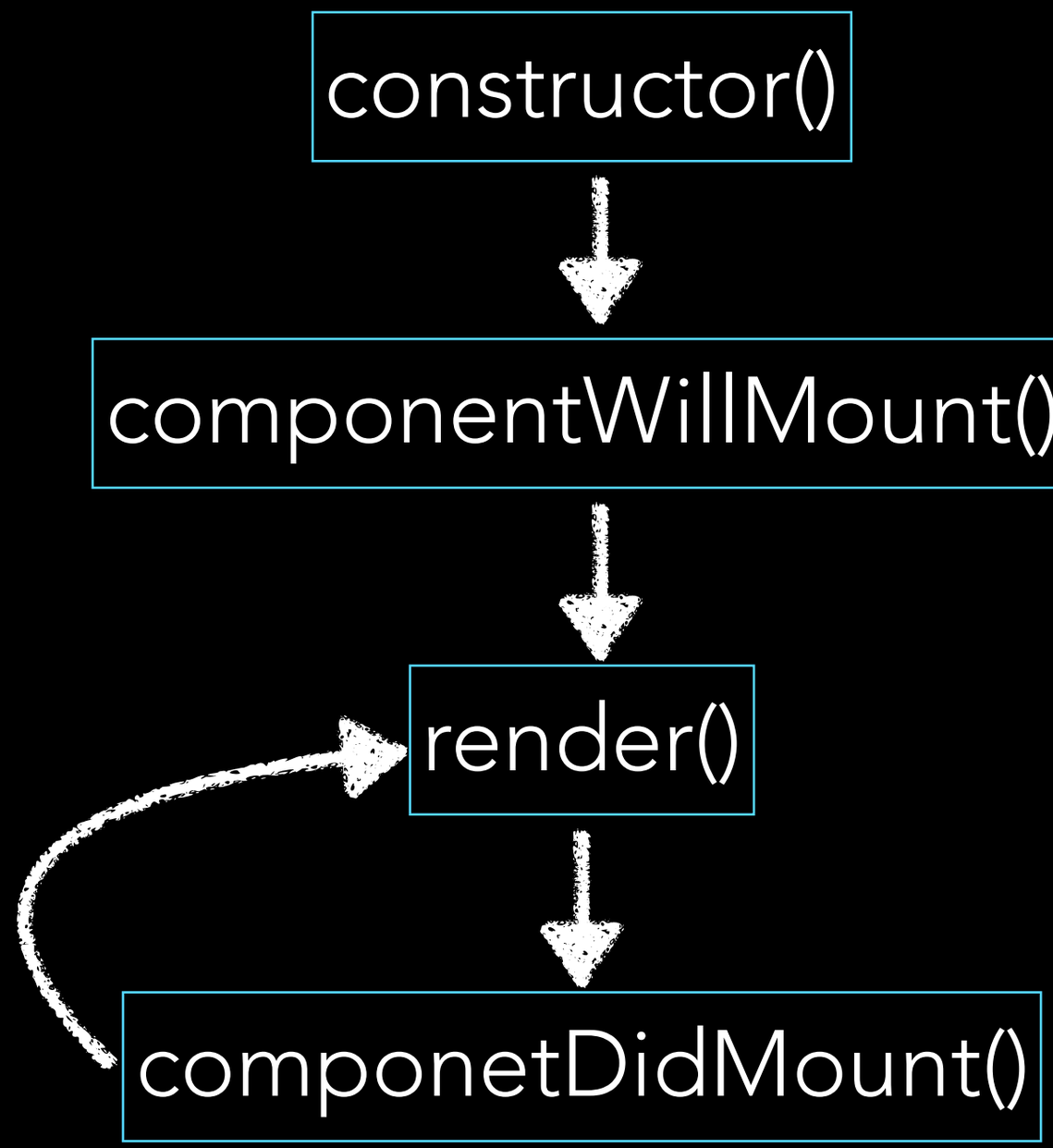
OPEN OUR FIRST SNACK

HTTPS://SNACK.EXPO.IO/@PROFESSORXII/BROKEN-CLOCK

NOTICE NO INDEX.HTML

- Notice there are no index.html
- Reactive native and Expo environment will simply render the time string.
- Well how do we get the time to update.
    - This were architecture ideals of life cycles come in

COMPONENT LIFE CYCLE

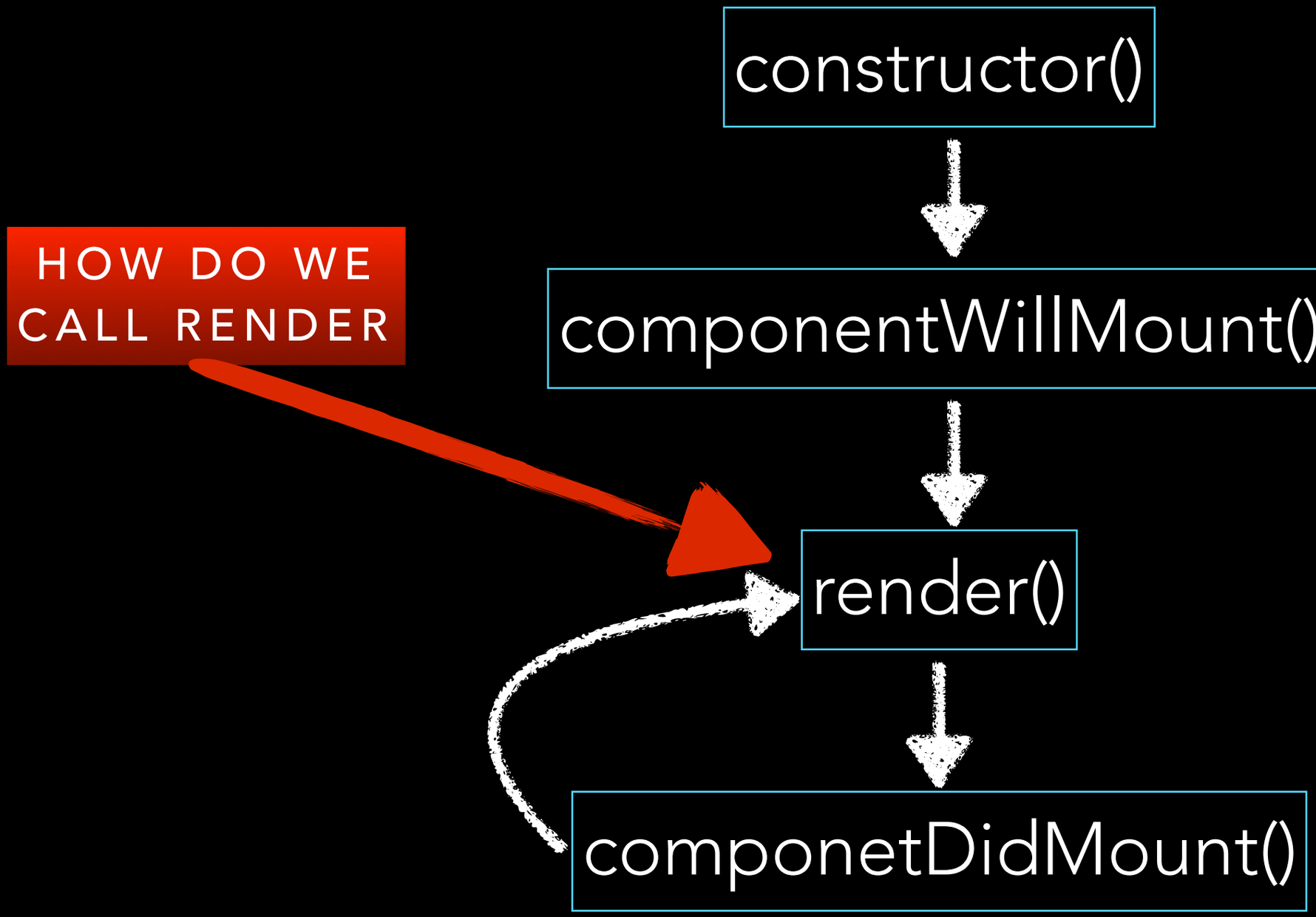


VIRTUAL DOM LIFE CYCLE METHODS

- componentWillUnmount is called immediately before the component is tore down or ‘unmounted'.

- render is the most important lifecycle method and the only required one in any component. It is usually called every time the component's state is updated, reflecting changes in the user interface.

COMPONENT LIFE CYCLE



Broken Clock 2

https://snack.expo.io/@professorxii/broken-clock-i

VIRTUAL DOM LIFE CYCLE METHODS

HOOKS INTO A COMPONENTS LIFE CYCLE

- shouldComponentUpdate allows the developer to prevent unnecessary re-rendering of a component by returning false if a render is not required.

- componentDidMount is called once the component has 'mounted' (the component has been created in the user interface, often by associating it with a DOM node). This is commonly used to trigger data loading from a remote source via an API.

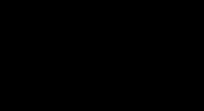
THERE ARE TWO WAYS TO GET DATA TO A COMPONENT

- Props:
  - Can’t be modified with render the component.
    - Because of the pure function restriction
- State
  - Used for data is going to change
  - Usually initialize in the constructor
  - When state on component change a rerender is triggered

Broken Clock 3

HTTPS://SNACK.EXPO.IO/@PROFESSORXII/BROKEN-CLOCK-II

```
class App extends React.Component {
    constructor(props) {
      super(props)
      this.state = {date: new Date()}
    }
     componentDidMount() {
      this.timerID = setInterval(
        () => this.tick(),
        1000
      )
    }
    componentWillUnmount() {
      clearInterval(this.timerID);
    }
    tick() {
      this.setState({
        date: new Date()
      })
    }
    render() {
      return (
        <View style={styles.container}>
          <Text>It is {this.state.date.toLocaleTimeString()}.</Text>
        </View>
      )
    }
  }
```



ASYNCHRONOUSLY CALLS RENDER

LET MAKE CLOCK COMPONET

CLOCK COMPONENT

HTTPS://SNACK.EXPO.IO/@PROFESSORXII/BROKEN-CLOCK-IV

LET'S BUILD A STOCK REPORTING REACT APP

https://iextrading.com/developer/docs/#price

https://api.iextrading.com/1.0/stock/aapl/price

JUST SOME NOTES ON STATE

Since react components are treated as pure Functions. If want update property dynamically Use instance variables or special variable call state.

FETCH

IS A PROMISE

RETURNS A PROMISE MAKER

Promise maker
text()
json()

CALL THE TEXT METHOD
RETURN A PROMISE

Promise
result

RESULT OF PROMISE
IS THE VALUE ON WEBPAGE

LET'S MODIFY THE TICK

```
tick() {
    this.setState({
       date: new Date()
     });
   }
```

LET'S DO THE FETCH HERE

FETCH IS A PROMISE

```
tick() {
     let fetchPromise  = fetch("https://api.iextrading.com/
1.0/stock/aapl/price")
     fetchPromise.then((resultPromise)=>{
       return resultPromise.text()
     }).then( (result)=>{
         this.setState({
           price: result
         })
       }
     )
   }
```

PROMISE CHAINING

USING THE AWAIT APPROACH

```
    async tick() {
      let response = await fetch("https://api.iextrading.com/1.0/
stock/aapl/price")
      let responseText = await response.text();
          this.setState({
            price: responseText
          })
        }
```

MORE DETAILS ON

STATE STATE

```
// Wrong
this.setState({
  counter: this.state.counter + this.props.increment,
});
```

```
// Correct
this.setState((state, props) => ({
  counter: state.counter + props.increment
}));
```

```
  constructor(props) {
    super(props);
    this.state = {
      posts: [],
      comments: []
    }
  }

  componentDidMount() {
    fetchPosts().then(response => {
      this.setState({
        posts: response.posts
      });
    });

    fetchComments().then(response => {
      this.setState({
        comments: response.comments
      });
    });
  }
```

OPEN SNACK FROM COURSE SITE

https://snack.expo.io/@professorxii/dmlld3

https://snack.expo.io/@professorxii/viewsdemo