

Speak React Native

React Native course by **U+.**

Your homework?

Overview

- Synchronous vs. asynchronous operations
- Component lifecycle
- API

Synchronous vs. asynchronous operations

Sync vs. Async operations

- **Synchronous**

- Subsequent client code does not execute until the operation finishes
- Example: Assign value to variable

- **Asynchronous**

- Client doesn't wait for the end of the previous operation
- Example: setState, HTTP requests

SetState

```
export default class Counter extends React.PureComponent {  
  state = { counter: 0 }  
  addOne = () => {  
    this.setState(prevState => ({ counter: prevState.counter + 1 })), () =>  
  {  
    console.log(this.state.counter) // Good  
  })  
}  
  render() {  
    return <Button onPress={this.addOne}>+1</Button>  
  }  
}
```

Don't use setState synchronously

```
export default class Counter extends React.PureComponent {  
  state = { counter: 0 }  
  addOne = () => {  
    this.setState(prevState => ({ counter: prevState.counter + 1 }))  
    console.log(this.state.counter) // Bad  
  }  
  render() {  
    return <Button onPress={this.addOne}>+1</Button>  
  }  
}
```

Component lifecycle

Component lifecycle

- Lifecycle methods are called automatically in specific cases. E. g.:
 - When component **is loaded** (componentDidMount)
 - When component **props/state is updated** (componentDidUpdate)
 - Right before component **is unmounted** (componentWillUnmount)
 - Etc...

Component lifecycle



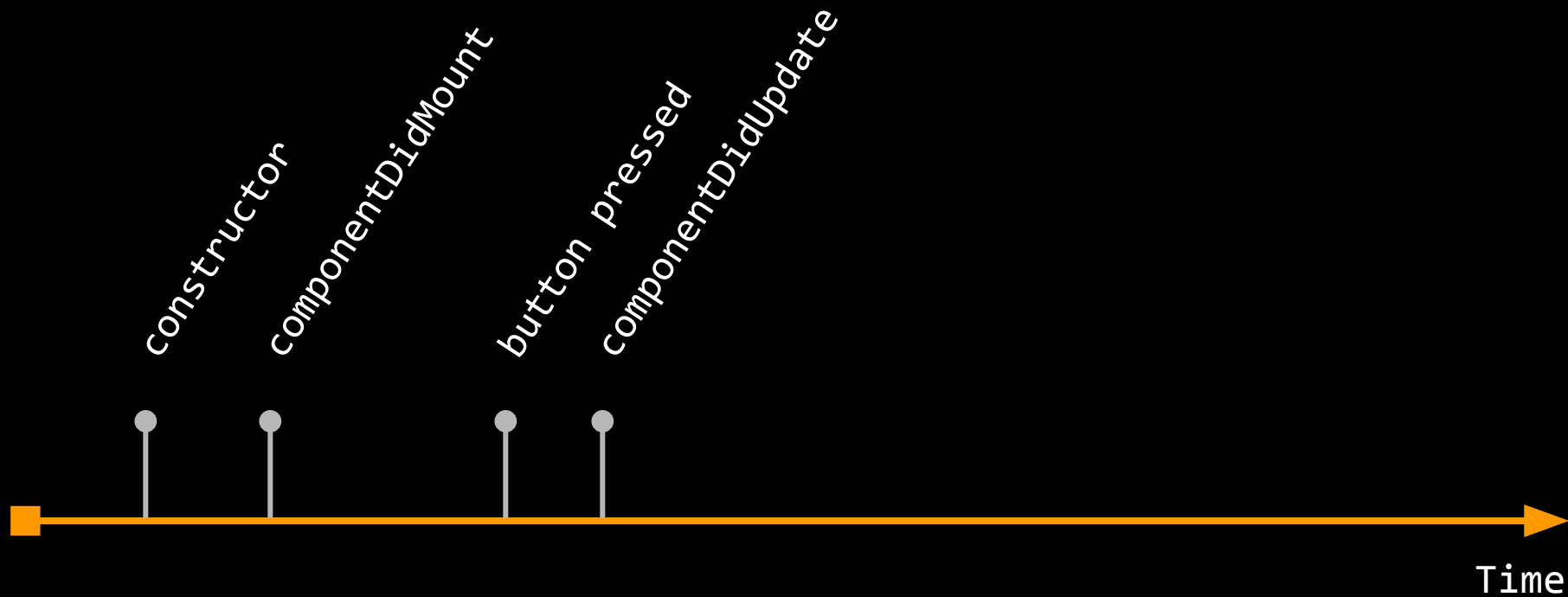
Component lifecycle



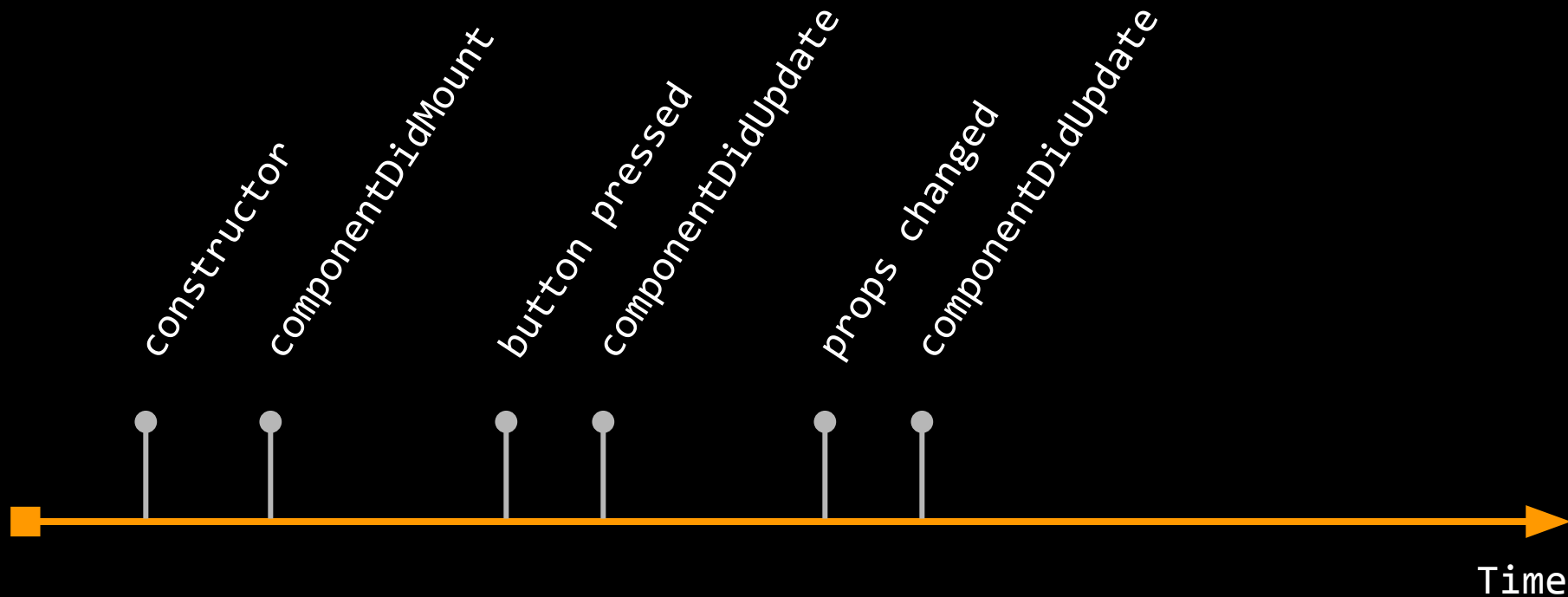
Component lifecycle



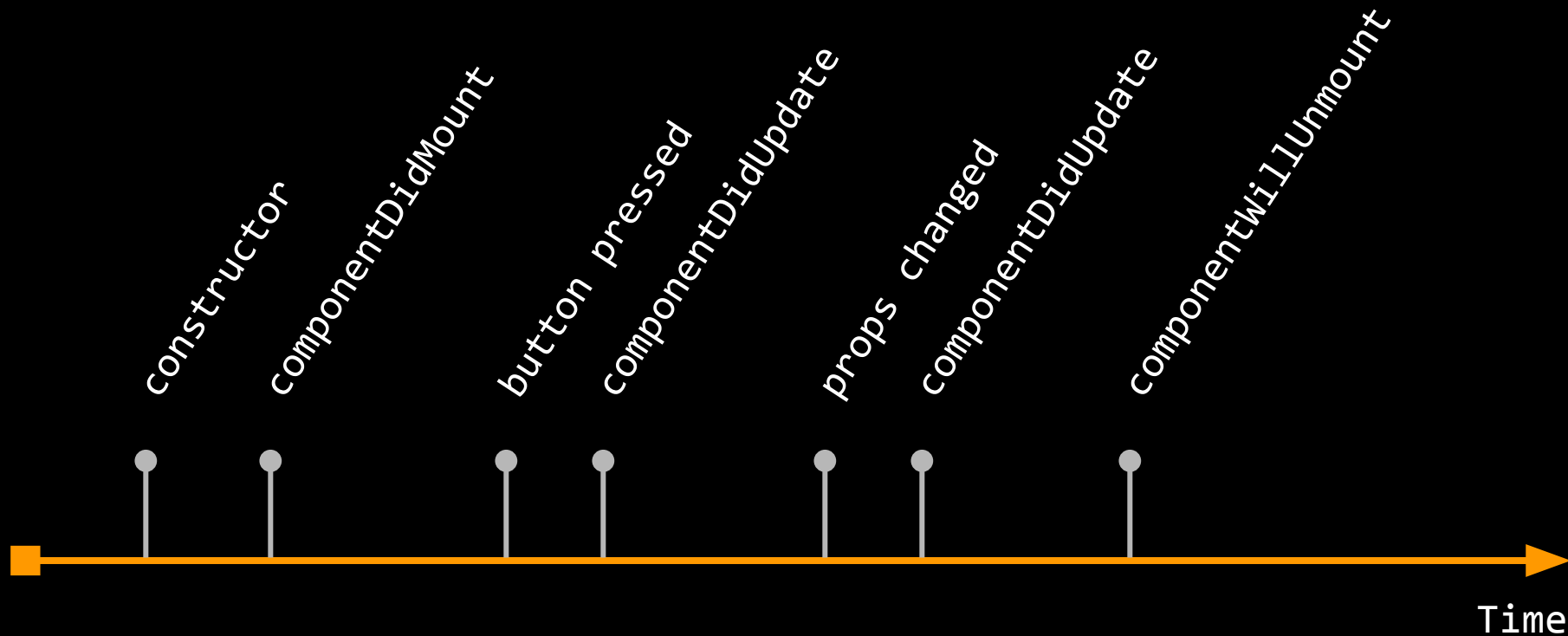
Component lifecycle



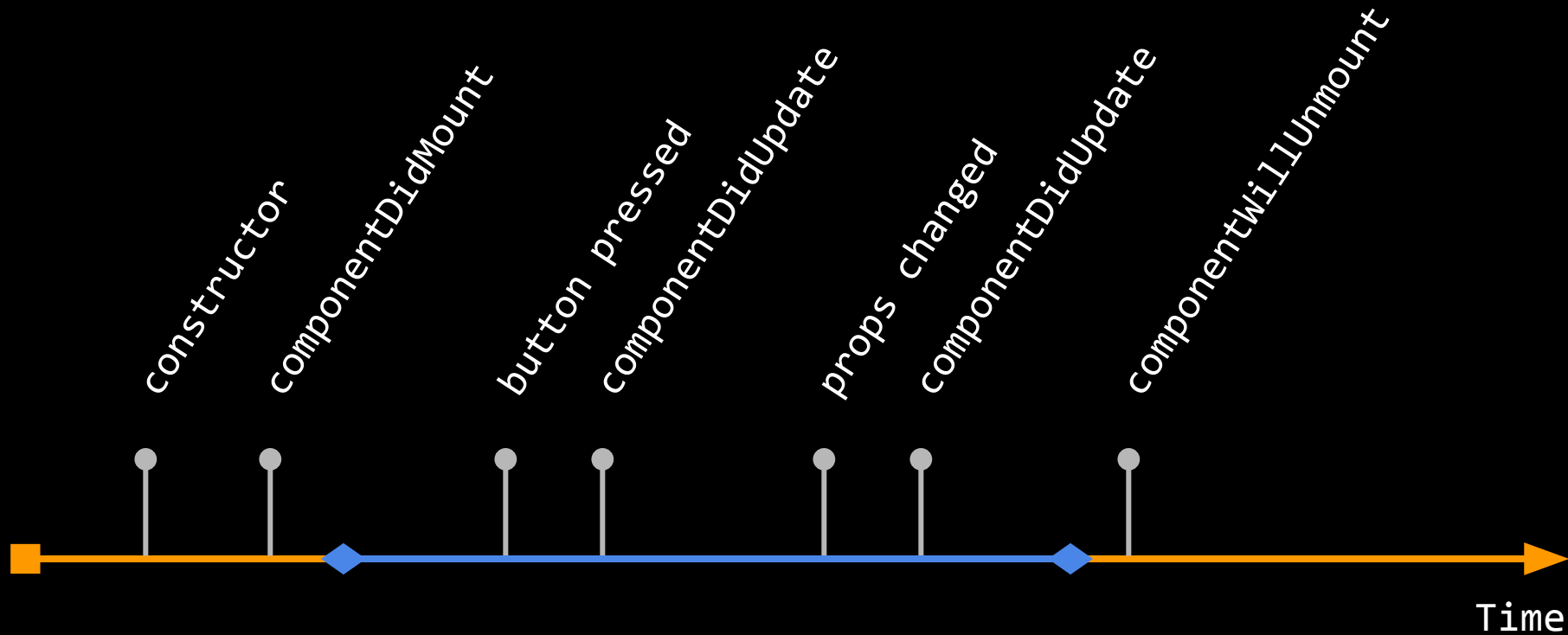
Component lifecycle



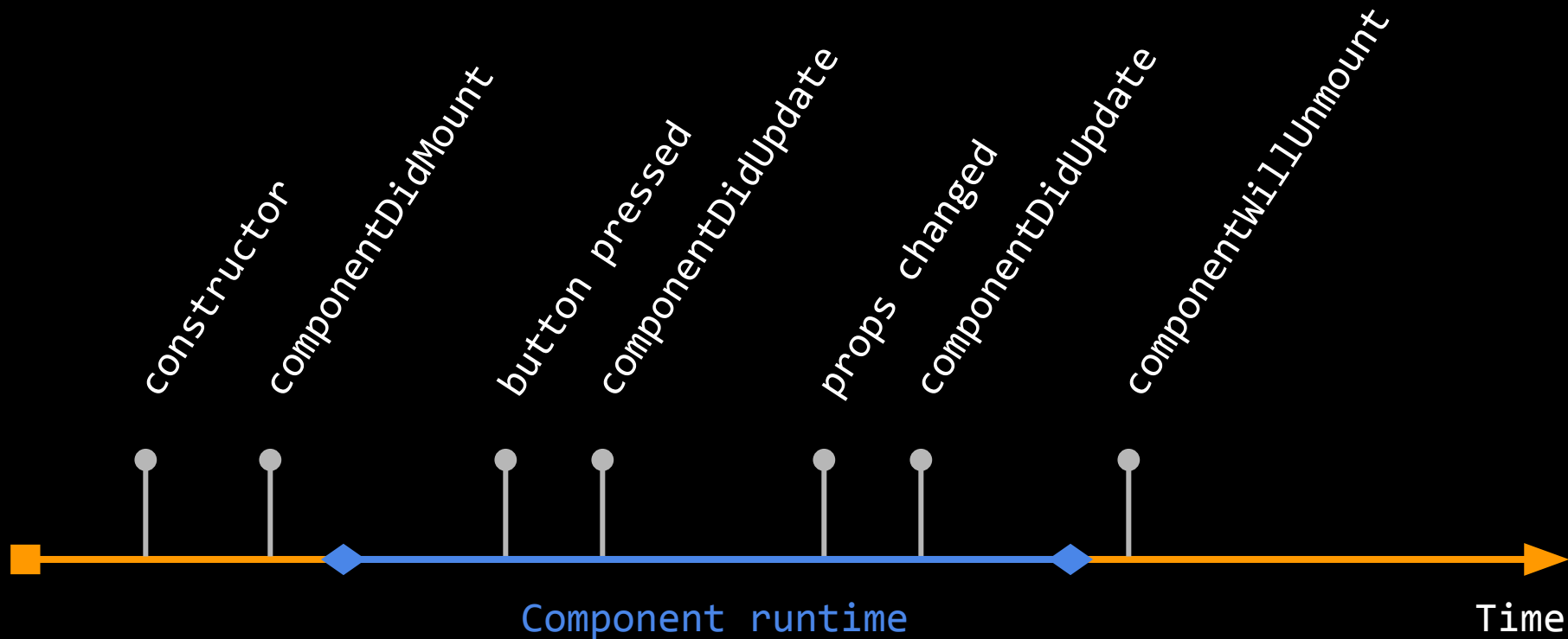
Component lifecycle



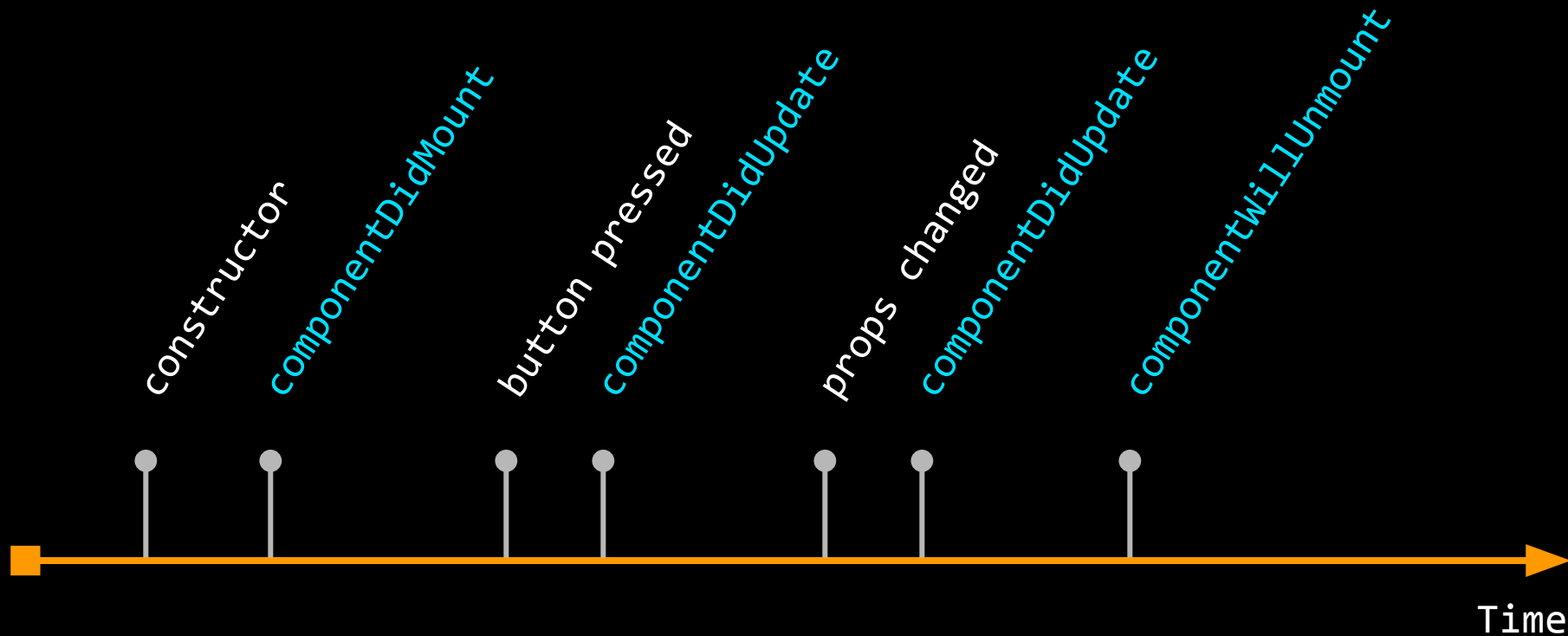
Component lifecycle



Component lifecycle



Component lifecycle methods



Why we need lifecycle methods

- For example we want get data from server when screen is loaded

Why we need lifecycle methods

```
export default class App extends React.PureComponent {  
  componentDidMount() {  
    this.getDataFromApi()  
  }  
  getDataFromApi = () => {  
    // @TODO  
  }  
  render() {  
    return <Navigator />  
  }  
}
```

API

Why we use API



The diagram illustrates the role of an API in connecting different applications to a database. On the left, three red hearts represent various applications: 'Your mobile app' (top), 'Some web app' (middle), and an ellipsis '...' (bottom). On the right, an orange cylinder represents 'Some DB'. The applications are positioned to the left of the database, suggesting they interact with it through an API.

Your
mobile
app

Some
web app

...

Some DB

Why we use API



The diagram illustrates the role of an API in connecting various applications to a central database. On the left, three red hearts represent different types of applications: 'Your mobile app' (top), 'Some web app' (middle), and an ellipsis '...' (bottom). On the right, an orange cylinder represents 'Some DB'. The text 'Stores all data (including non-public)' is positioned below the database cylinder. The entire diagram is set against a black background.

Your
mobile
app

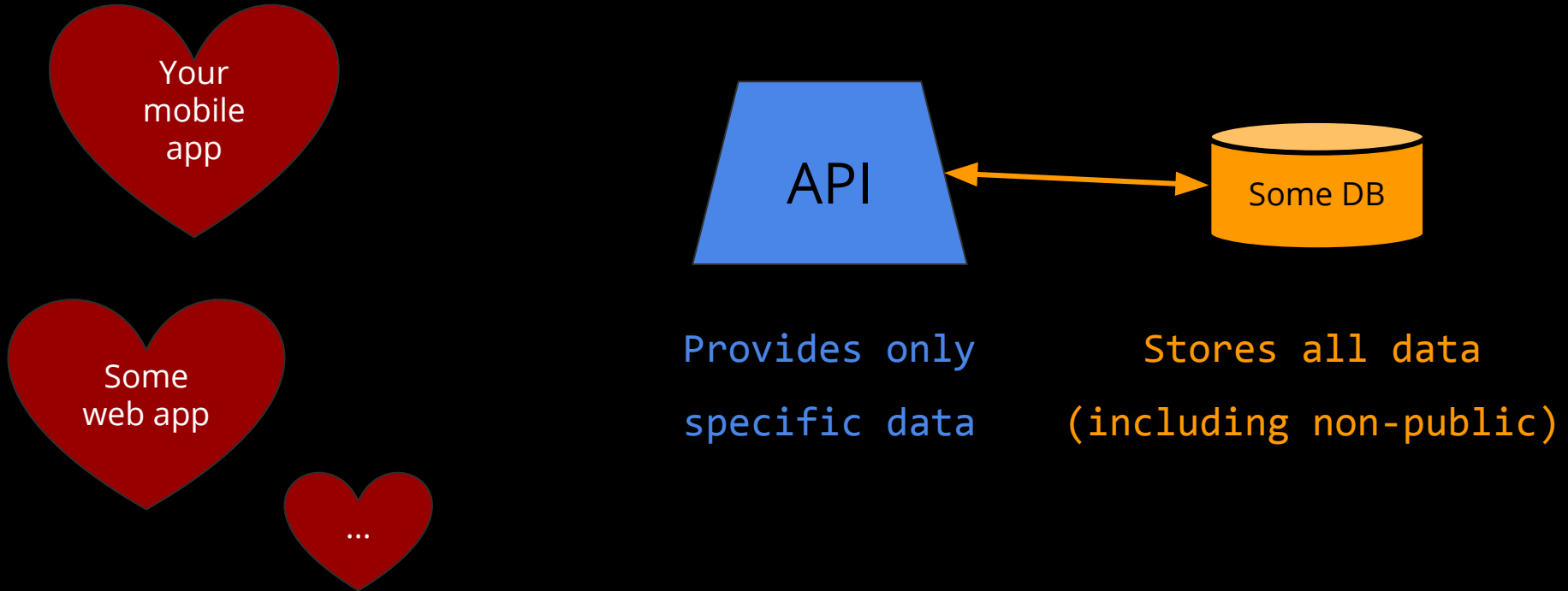
Some
web app

...

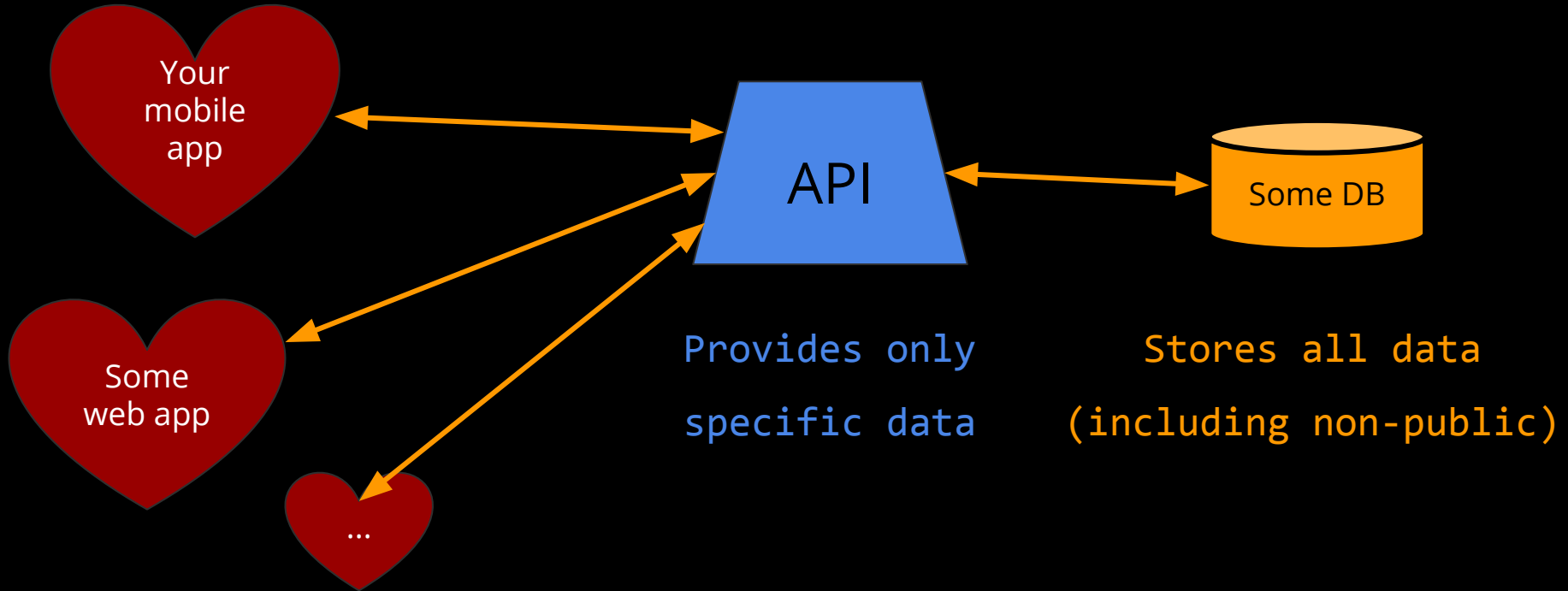
Some DB

Stores all data
(including non-public)

Why we use API



Why we use API



API

- HTTP Request & response
- Postman
- REST, GraphQL
- Axios

HTTP

- **H**yper**T**ext **T**ransfer **P**rotocol
- For requesting content from server and sending data

HTTP request

- Client requests some data from a server
- Headers
 - Content-type (application/json)
 - Method: GET, POST, PUT, DELETE,...
 - Authorization
- Body
 - Data

HTTP response

- Server sends back data in the requested format
- Headers
 - Meta information
 - E. g.: Content-type, status,...
- Body
 - Data which we requested

HTTP communication example

HTTP communication

Client



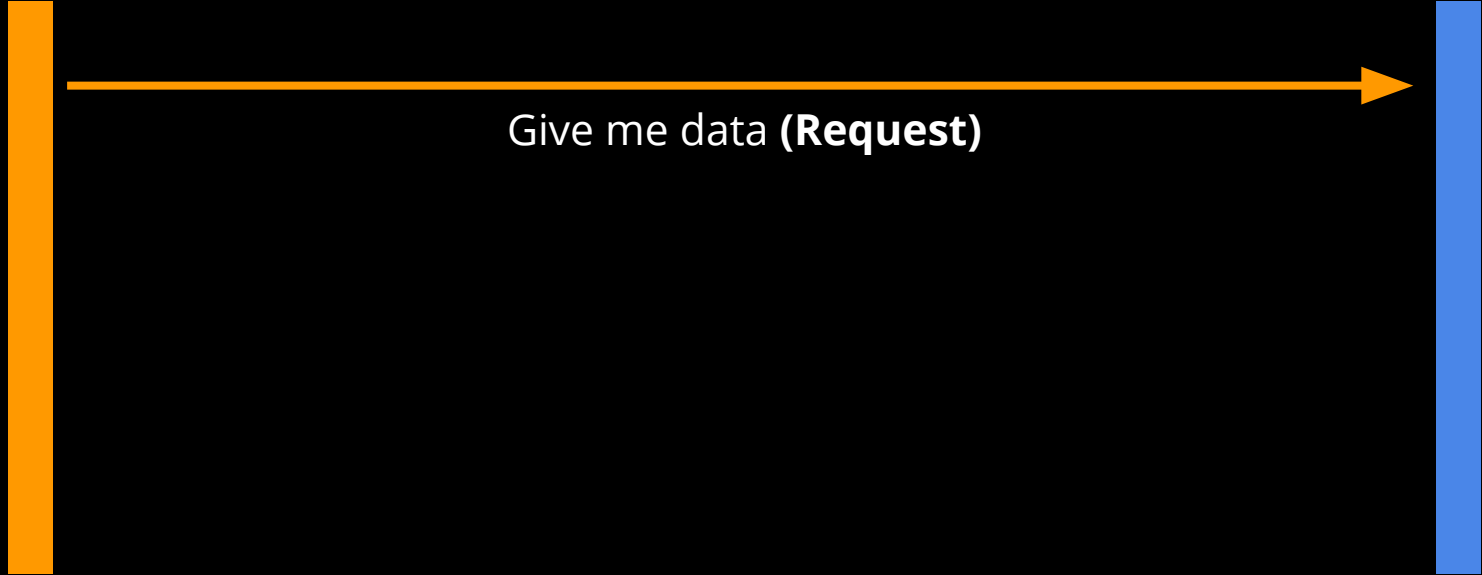
Server



HTTP communication

Client

Server



HTTP communication

Client

Server



HTTP communication example

Client

Server



GET <http://api.themoviedb.org/3/movie/122>

HTTP communication example

Client

Server


```
GET http://api.themoviedb.org/3/movie/122
```

```
{
  "title": "The Lord of the Rings 3",
  "release_date": "2003-12-01"
  , ...
}
```

HTTP communication example


Client

Server



The diagram shows a Client on the left and a Server on the right, each represented by a vertical bar. An orange arrow points from the Client to the Server, representing the outgoing request.

```
GET http://api.themoviedb.org/3/movie/122  
[Method]
```




The diagram shows a Server on the right and a Client on the left, each represented by a vertical bar. A blue arrow points from the Server to the Client, representing the outgoing response.

```
{  
  "title": "The Lord of the Rings 3",  
  "release_date": "2003-12-01"  
  , ...  
}
```

HTTP communication example


Client

Server



GET `http://api.themoviedb.org/3/movie/122`
[API URL]

The diagram shows a thick orange vertical bar on the left representing the Client and a thick blue vertical bar on the right representing the Server. A horizontal orange arrow points from the Client to the Server, indicating the direction of the request.




{
 "title": "The Lord of the Rings 3",
 "release_date": "2003-12-01"
 , ...
}

The diagram shows a horizontal blue arrow pointing from the Server back to the Client, indicating the direction of the response.

HTTP communication example

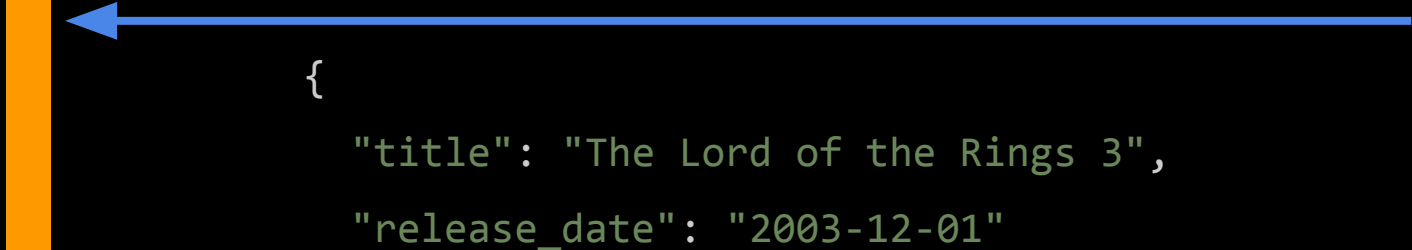
Client

Server



GET `http://api.themoviedb.org/3/movie/122`
[endpoint]

The diagram shows a horizontal orange arrow pointing from the Client to the Server. The text 'GET http://api.themoviedb.org/3/movie/122' is written above the arrow, and '[endpoint]' is written below it.



```
{  
  "title": "The Lord of the Rings 3",  
  "release_date": "2003-12-01"  
  , ...  
}
```

The diagram shows a horizontal blue arrow pointing from the Server to the Client. The JSON response is written below the arrow.

Request methods (REST)

- **GET** – Get data from server
- **POST** – Create data on server
- **PUT** – Update data on server
- **DELETE** – Delete data on server

Status codes + examples

- 1.. – **Informational**
- 2.. – **Success** (200 – OK)
- 3.. – **Redirect** (301 – Moved permanently)
- 4.. – **Client error** (404 – Not found)
- 5.. – **Server error** (500 – Internal server error)

REST

- **R**epresentational **S**tate **T**ransfer
- “Standard” set of operations of a web service
- Basically a set of abstract recommendations on how an API should work

GraphQL

- Different approach to REST, but the same purpose
- Removes the need of different endpoints which results in less requests to the server
- More about it later

Postman

- An application for exploring or testing APIs
- Used for displaying data structure without writing code
- [Download page](#)
- Try The Movie DB API
 - <https://developers.themoviedb.org/3/>
 - Api key: 4aa883f95999ec813b8bfaf319f3972b

API calls

Axios

- HTTP client library for handling requests
- Not necessary for making requests, but makes it much easier
- Has many alternatives

```
$ yarn add axios
```

Axios initialization

```
// api.js
import axiosLib from "axios"

export const axios = axiosLib.create({
  baseURL: "http://api.themoviedb.org/3/",
  timeout: 10000,
  headers: {
    "Content-Type": "application/json",
    Accept: "application/json",
  },
})
```

Making requests

```
// api.js
```

```
// define endpoint
```

```
export const loadMovies = () => {  
  return axios.get(  
    "/movie/top_rated?api_key=4aa883f95999ec813b8bfaf319f3972b"  
  )  
}
```

Making requests

```
// your screen
```

```
import { loadMovies } from "../api"
```

```
// ...
```

```
componentDidMount() {  
  loadMovies().then(response => {  
    this.setState({  
      movies: response.data.results  
    })  
  })  
}
```


Authorization

// not ideal when we have more endpoints

```
axios.get(  
  "/movie/latest?api_key=4aa883f95999ec813b8bfaf319f3972b"  
)
```

Authorization

// not ideal when we have more endpoints

```
axios.get(  
  "/movie/latest?api_key=4aa883f95999ec813b8bfaf319f3972b"  
)
```

Authorization

```
// better
```

```
const apiKey = "4aa883f95999ec813b8bfaf319f3972b"
```

```
axios.interceptors.request.use(request => {  
  return {  
    ...request,  
    url: `${request.url}?api_key=${apiKey}`,  
  }  
})
```

```
axios.get(  
  "/movie/latest"  
)
```

Homework

Homework

- Create app with the list of top rated movies
 - <https://developers.themoviedb.org/3/movies/get-top-rated-movies>
- After click on movie navigate user to movie detail
 - <https://developers.themoviedb.org/3/movies/get-movie-details>
 - Don't forget to pass movieID to MovieDetail screen
- Create new request for Movie detail
- For advanced developers:
 - Create infinite scroll for movie list

Questions?



Sources

- <https://medium.com/@baphemot/understanding-react-react-16-3-component-life-cycle-23129bc7a705>
- <https://reactjs.org/docs/react-component.html>