React

Jan.schulz@devugees.org

Agenda

1. Introduction
2. Set-Up
3. Components
4. JSX
5. Props & States
6. Functional Components
7. Stateful Components Lifecycle
8. AXIOS

1. Introduction

- React
  - Client Side JavaScript framework
  - Created & maintained by Facebook
  - Used to build dynamic user interfaces
  - Renders everything as a component
  - Often to referred as the „V“ in MVC

1. Introduction

- React ...
  - Makes JavaScript easier ... for complex apps
  - Organizes your UI
  - Reusability
  - Scalability & Efficiency
  - Lighter than most frameworks
  - GET A JOB

1. Introduction

- React is used in
  - Facebook
  - Facebook App (React Native)
  - AirBnB
  - Tinder
  - Udemy
  - Instagram
  - Yahoo Mail

1. Introduction

- Requirements:
  - NodeJS
  - Visual Studio Code
    - ES7 React/Redux/GraphQL/React-Native snippets
  - Chrome Extension
    - React Developer Tools
    - Redux Developer Tools

2. Setup

- create-react-app is a predefined Webpack config
  - $ sudo npm install –g create-react-app

2. Setup

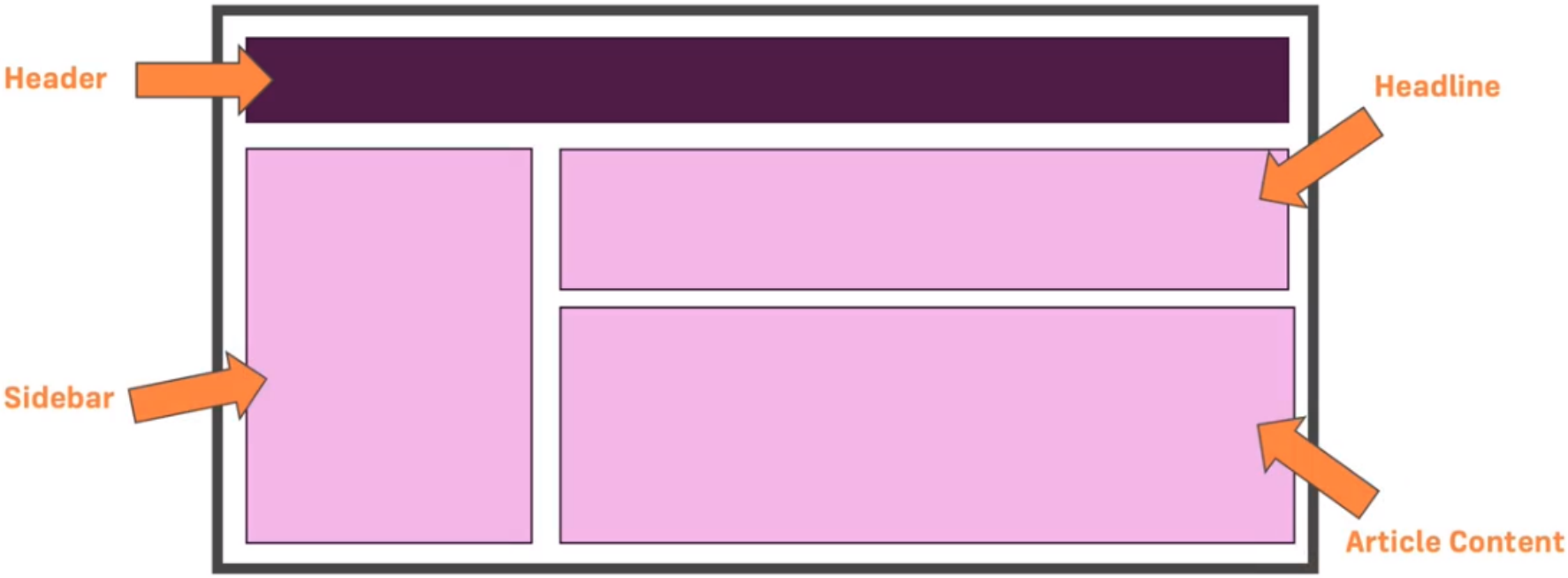
- package.json
  - Manifest that info about the application
- Public folder
  - Entry point to the entire application
  - In index.html, there is **root-div**, where React renders the entire app into

3. Components

- Thinking in React-Way

3. Components

Components:



3. Components

Components:

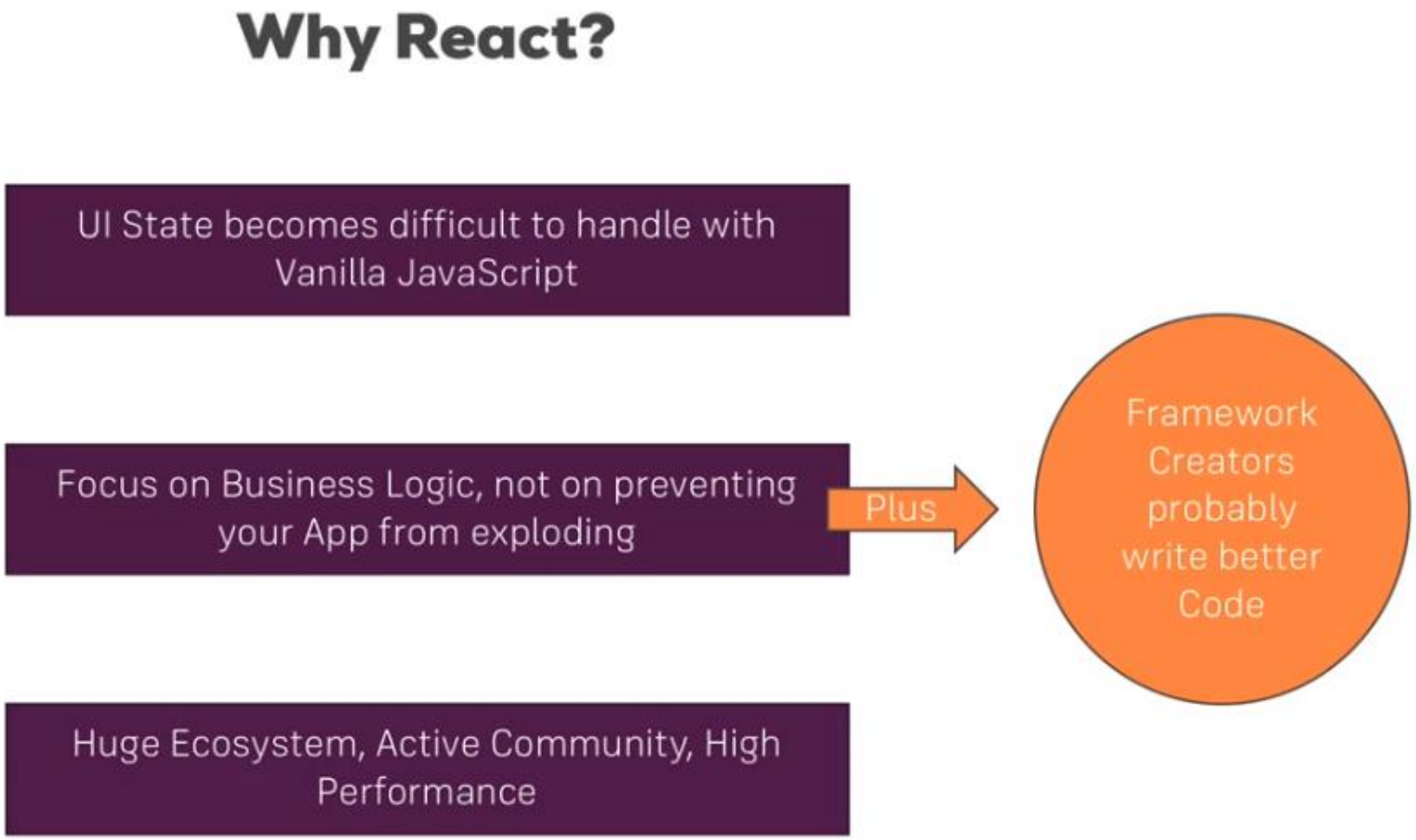
- Each component contains its separate container of code
- We **do not build** our web app as one bigger picture
- Why?

3. Components

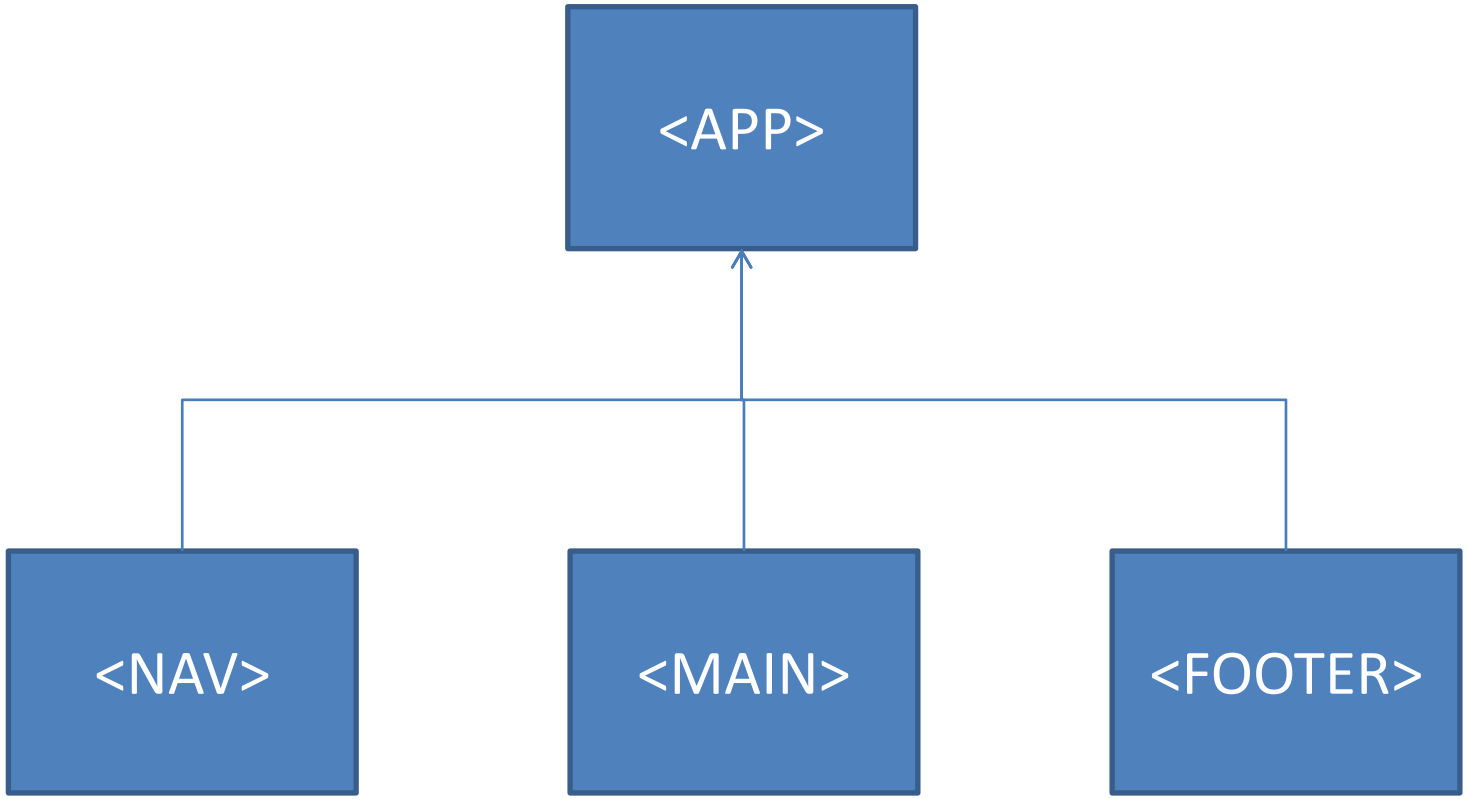
Components:

- Each component contains its separate container of code
- We **do not build** our web app as one bigger picture
- Why? – It makes our code **more manageable**, **more maintanable** and **more reusable**
- A react component is like a custom HTML element

3. Components



3. Components



3. Components

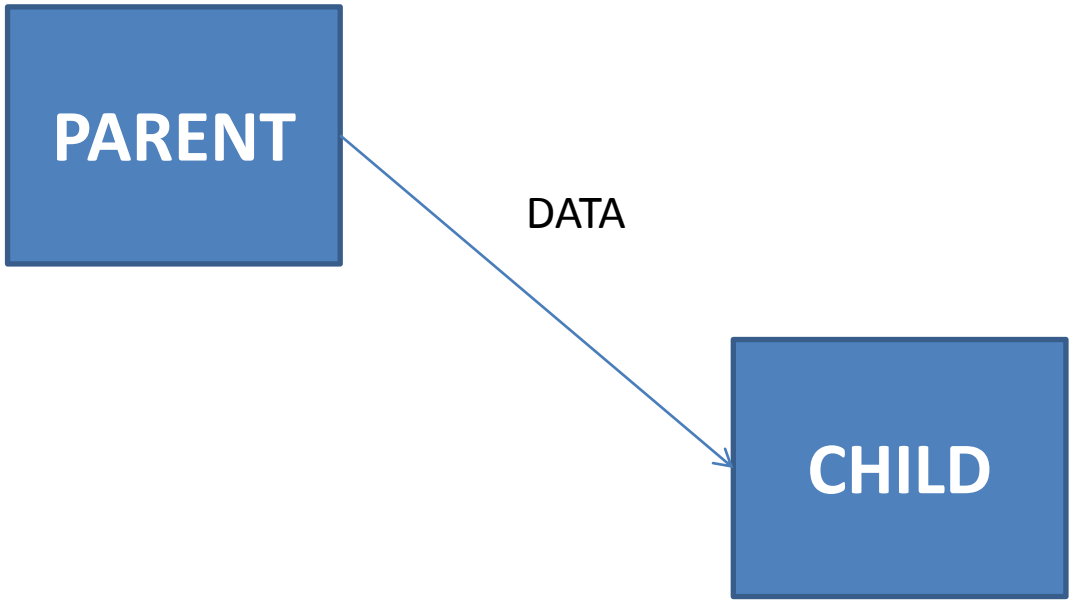
- Stateful Components
  - Have a state object
  - Return JSX
- Functional Components
  - **Do not** have a state object
  - Return JSX

4. JSX

- basically HTML in JavaScript
- Will be transpiled into JavaScript by Babel (as Webpack plugin)

5. Props & States

- Props
  - Data passed down from parent component to child component



5. Props & States

- State
  - Private data of a component

MyComponent

```
state = {
   active: true,
   x: 5
}
```

6. Functional Components

- Functional Components are Components, that do not have a state or lifecycle methods
- Why?

6. Functional Components

- Functional Components are Components, that do not have a state or lifecycle methods
- Why?
  - **For layout purposes**, components do not need to have an own state
  - **For performance issues**, functional components do not inherit stateful component methods and do not need to be instantiated

Task

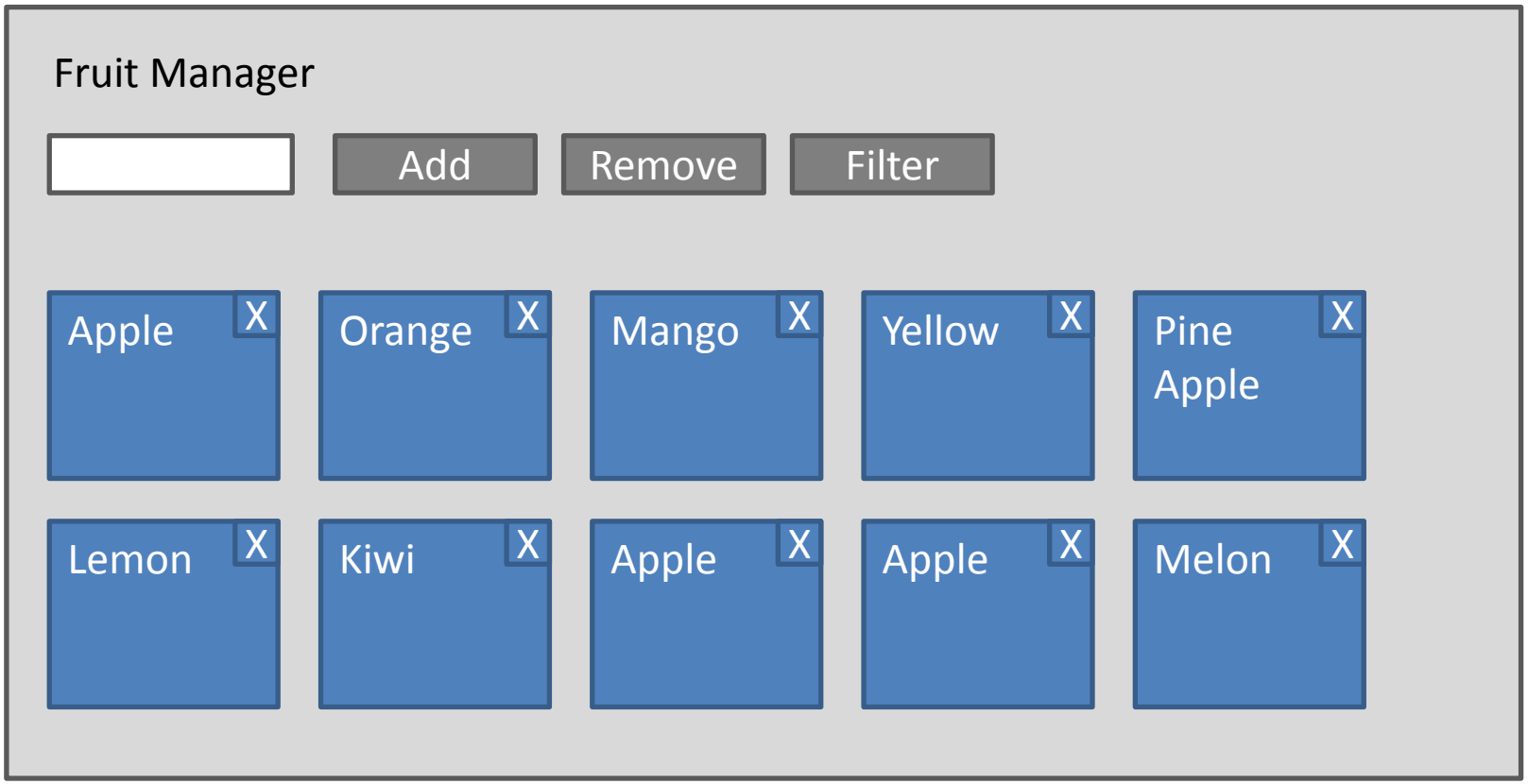
Create a React App „Fruit Manager“ where one can add, remove and filter fruits.

Note:

- Use both stateful and stateless components
- Use flexbox or grid for the arrangement of the items

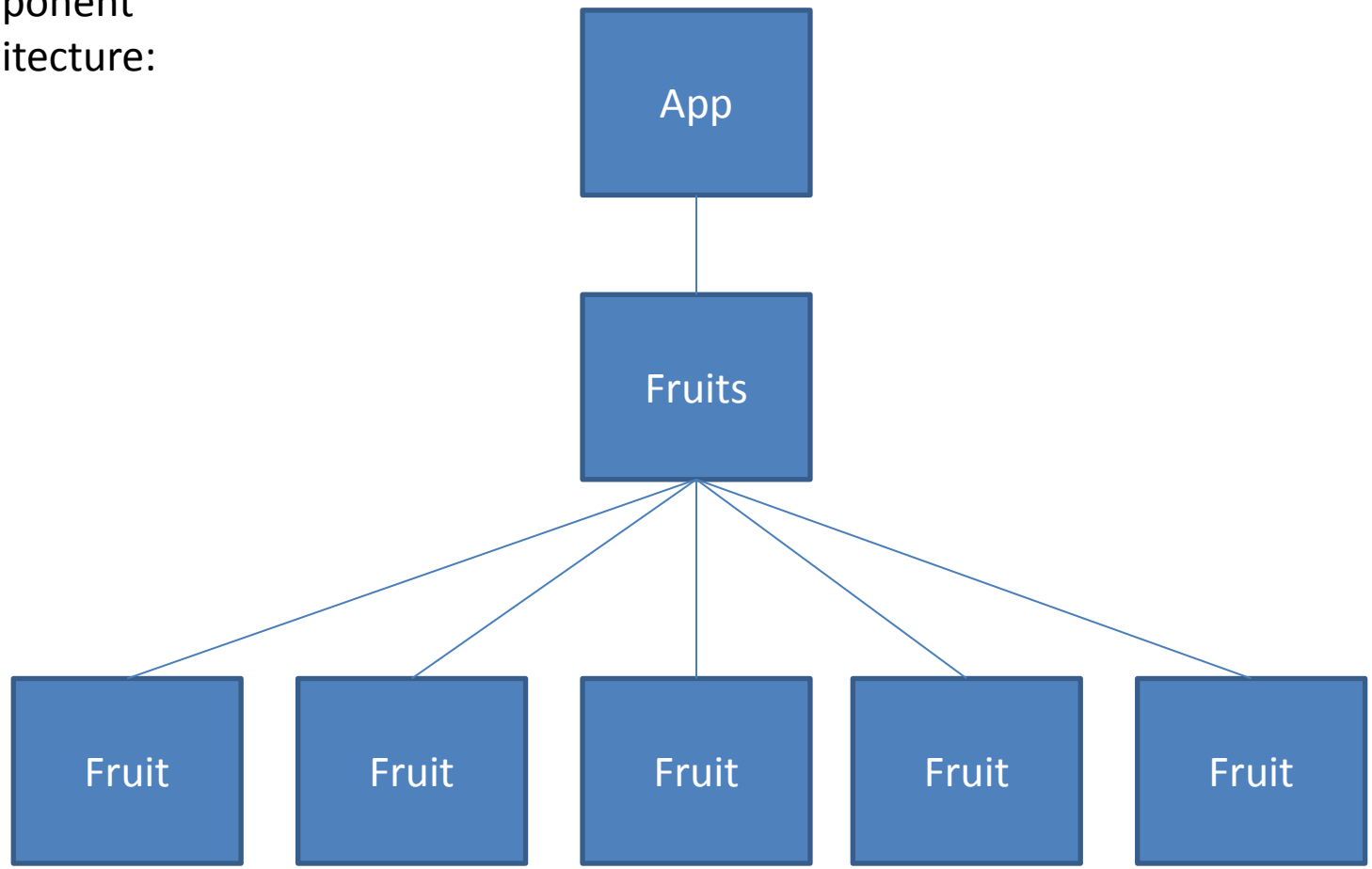
Task

Add: Adds a new fruit with random color
Remove: Removes all fruits that contain the substring of the text input
Filter: Shows only the fruits that contain the substring of the text input
Click on X: Removes the fruit



Task

Suggested Component Architecture:



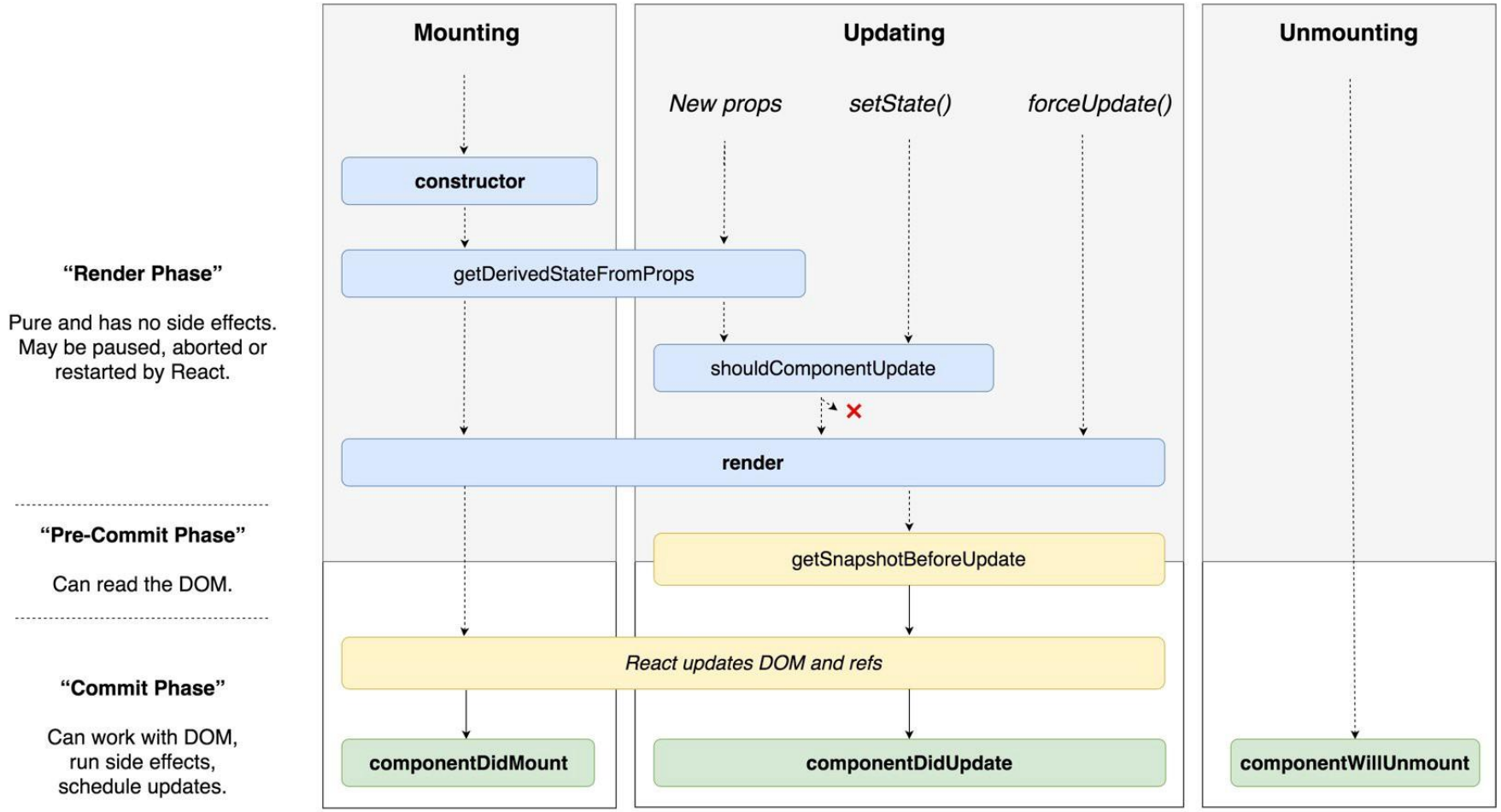
7. Stateful Component Lifecycle

- Each stateful component is
  1. Is born
  2. Lives
  3. Dies

7. Stateful Component Lifecycle

- Each stateful component is
  1. Is born = **mounts**, the parent container includes it in the virtual DOM
  2. Lives = **updates**, new props will be sent to the component or it decides by itself to set a new state
  3. Dies = **unmounts**, the parent container excludes it from the virtual DOM

7. Stateful Component Lifecycle



8. AJAX with AXIOS

- AXIOS is a library that implements the functionality of AJAX using promises
- NodeJS Servers can be integrated into React Apps using a Proxy feature

Task 1

Create a React App UserList that loads all users from

https://jsonplaceholder.typicode.com/users

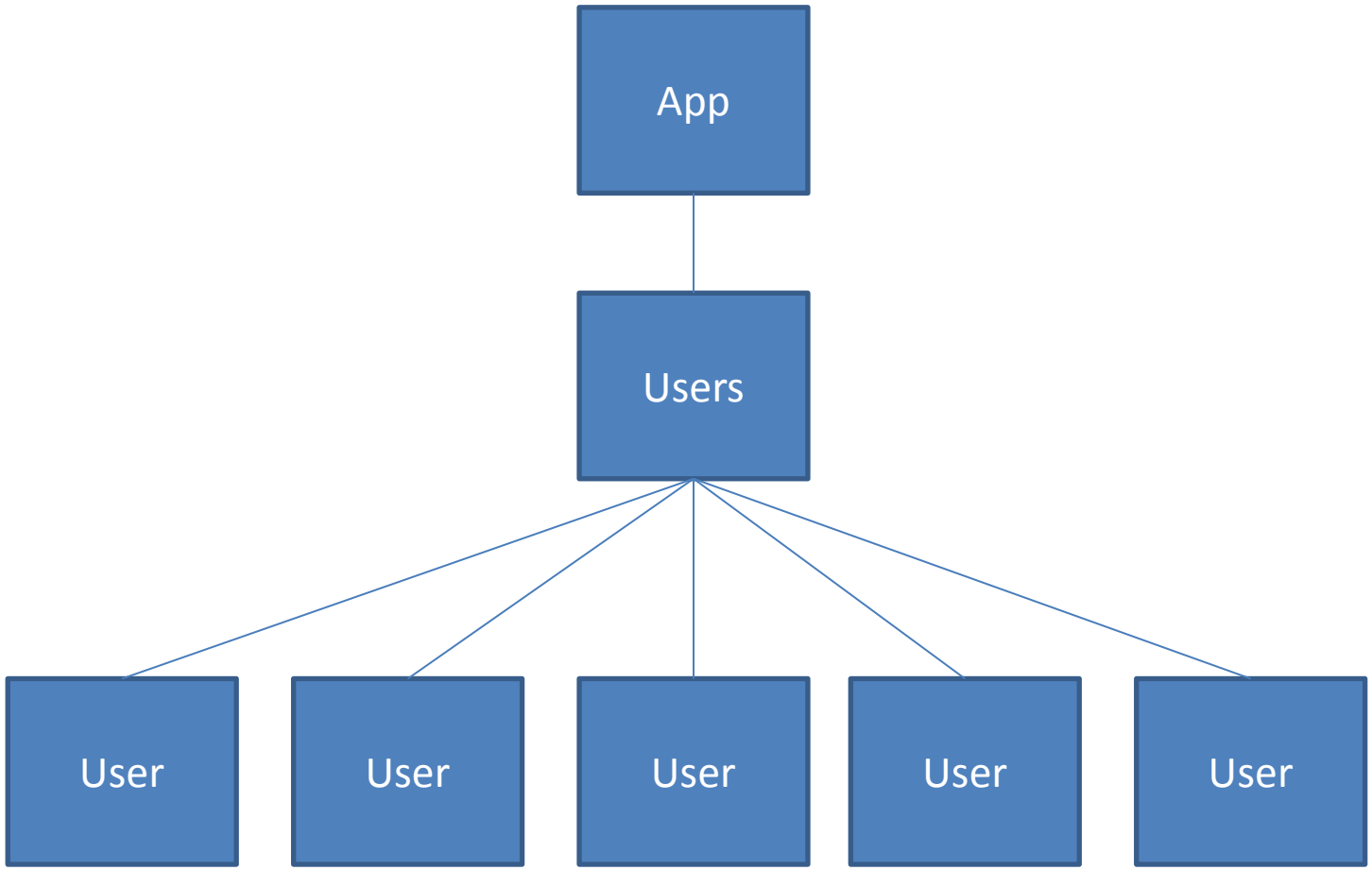
and shows the id, the name and the email in a table. By clicking on the X next to the user, the user will be removed from the internal state.

Task 1

User List

ID	Name:	Email:	
1	Leanne Graham	Sincere@april.biz	X
2	Clementine Bauch	Nathan@yesania.net	X
3	Patricia Lebsack	Julianna.Oconner@kory.org	X

Task 1



Task 2

Create a React App NextUser that consists of two components App and User. By clicking the button „Next User“, the next user will be loaded via AJAX inside the User component. **Warning: Very Difficult**

Task 2

Create a React App UserList that consists of two components App and User. By clicking the button „Next User“, the next user will be loaded via AJAX inside the User component.

1) When the App initializes,
    1) the App component sends the User component the id of the first user (id = 1) via props.
    2) The User component reads the id and loads the name and email via Axios from

https://jsonplaceholder.typicode.com/users/1

    3) The User component shows the name and email of the first user

Task 2

2) When the user clicks on „Next User“
   1) the App component internally updates its id from 1 to 2 (or in general from id to id + 1) and sends it down to the User component via props.
   2) The User component reads the id and loads the name and email via Axios from

https://jsonplaceholder.typicode.com/users/2

   3) The User component shows the name and email of the next user
   4) When the first 10 users are shown, the id shall start with 1 again.

9. Router

- Routing?

9. Router

- Routing?

= **Conditionally Rendering Components based on the URL**

1. / - Start.js
2. /about – About.js
3. /products – Products.js
4. ...

Task

A - Create a React App „UserBrowser“ that has three React routes:

1. / - the start page, only shows „Start Page“
2. /users – A table of users loaded via AXIOS from https://jsonplaceholder.typicode.com/users, here show the name, the email and the phone
3. /user/:id – Shows the name, username, email, phone and website of the user with the param :id https://jsonplaceholder.typicode.com/users/id

Task

B – Create a navigation with the menu points „Start“ and „Users“

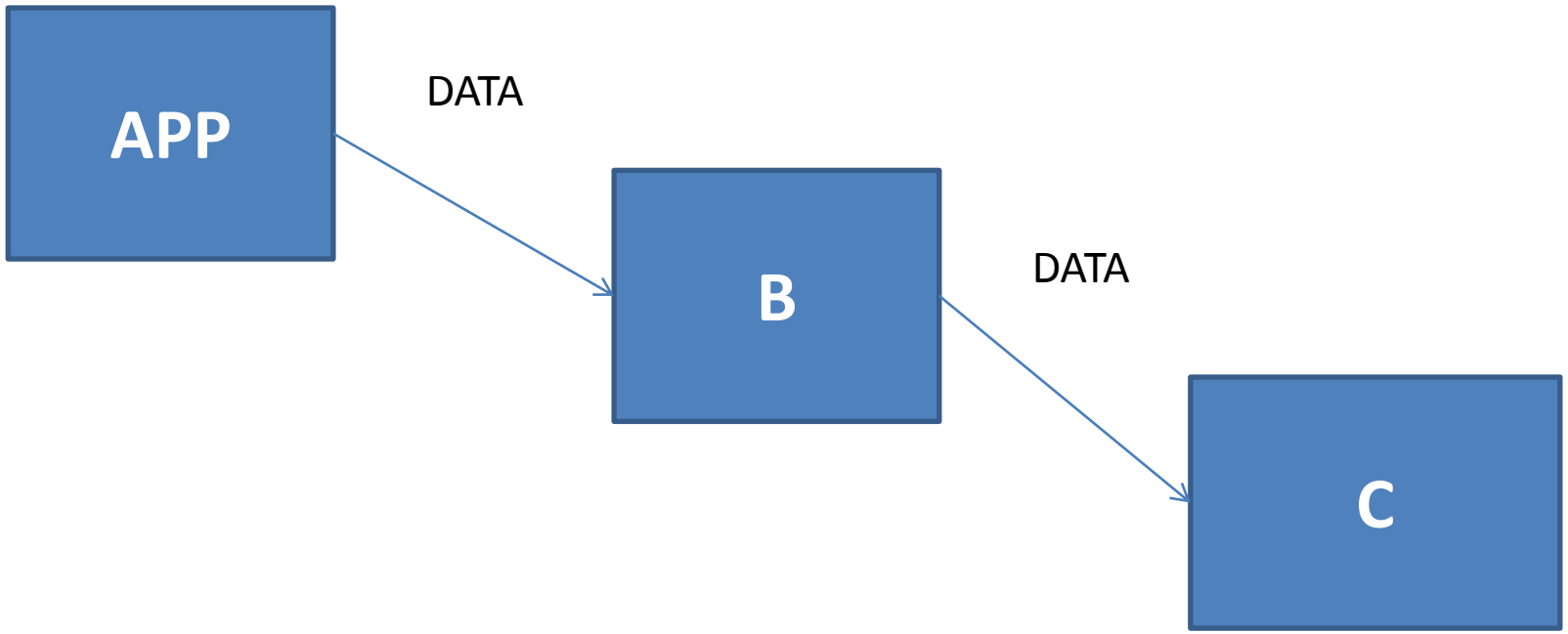
C – For each user in the user table, add a link „Details“ that leads to the user details.

10. Communication between multi-layered components

1. Props Drilling
2. Context API
3. Redux

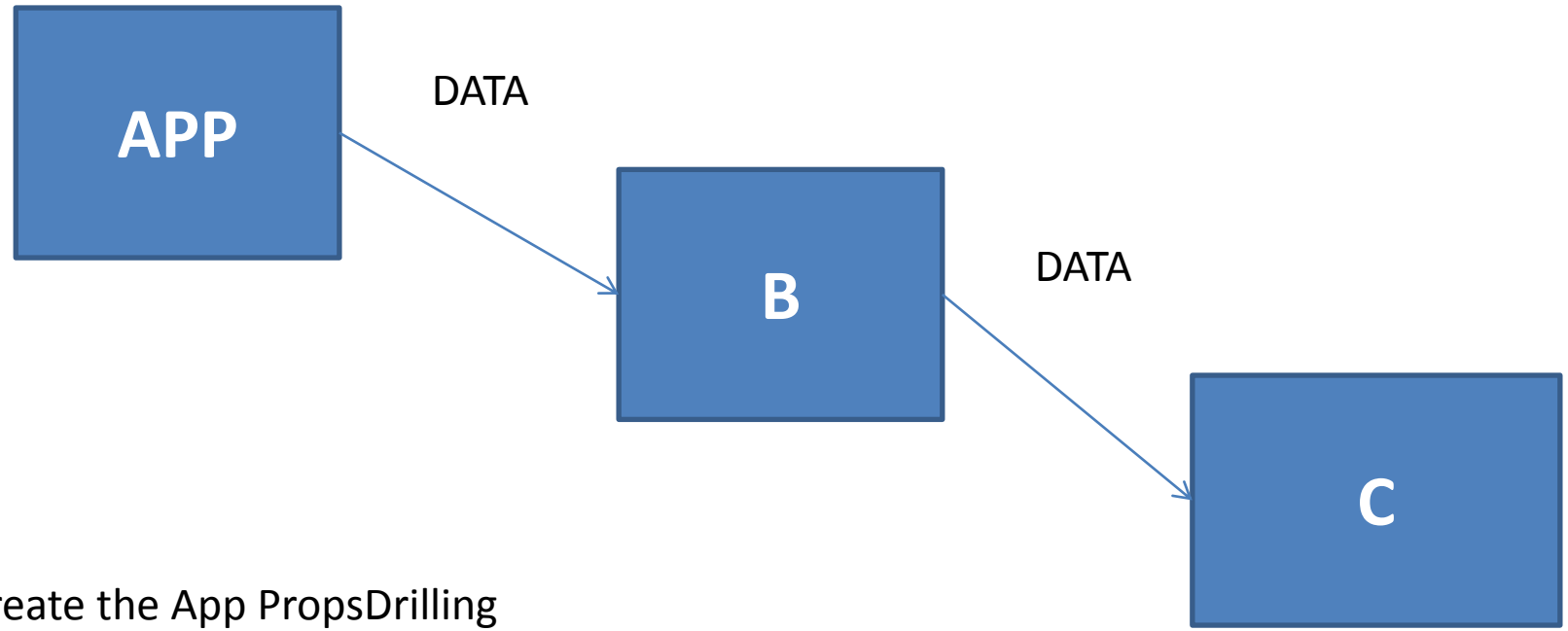
10. Communication between multi-layered components

1. Props Drilling = Passing data down child by child



10. Communication between multi-layered components

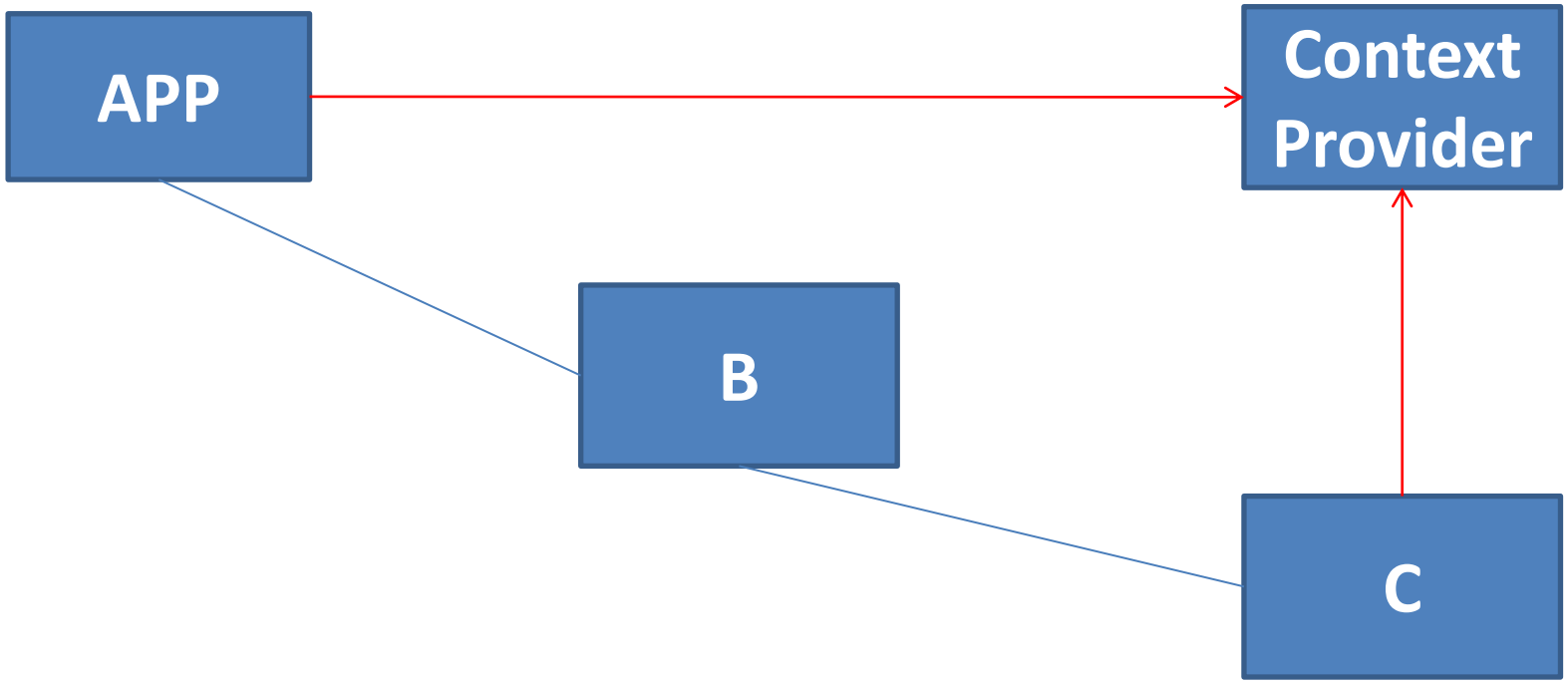
1. Props Drilling = Passing data down child by child



Task: Create the App PropsDrilling
that sends „Hallo World“ from App
down to component C

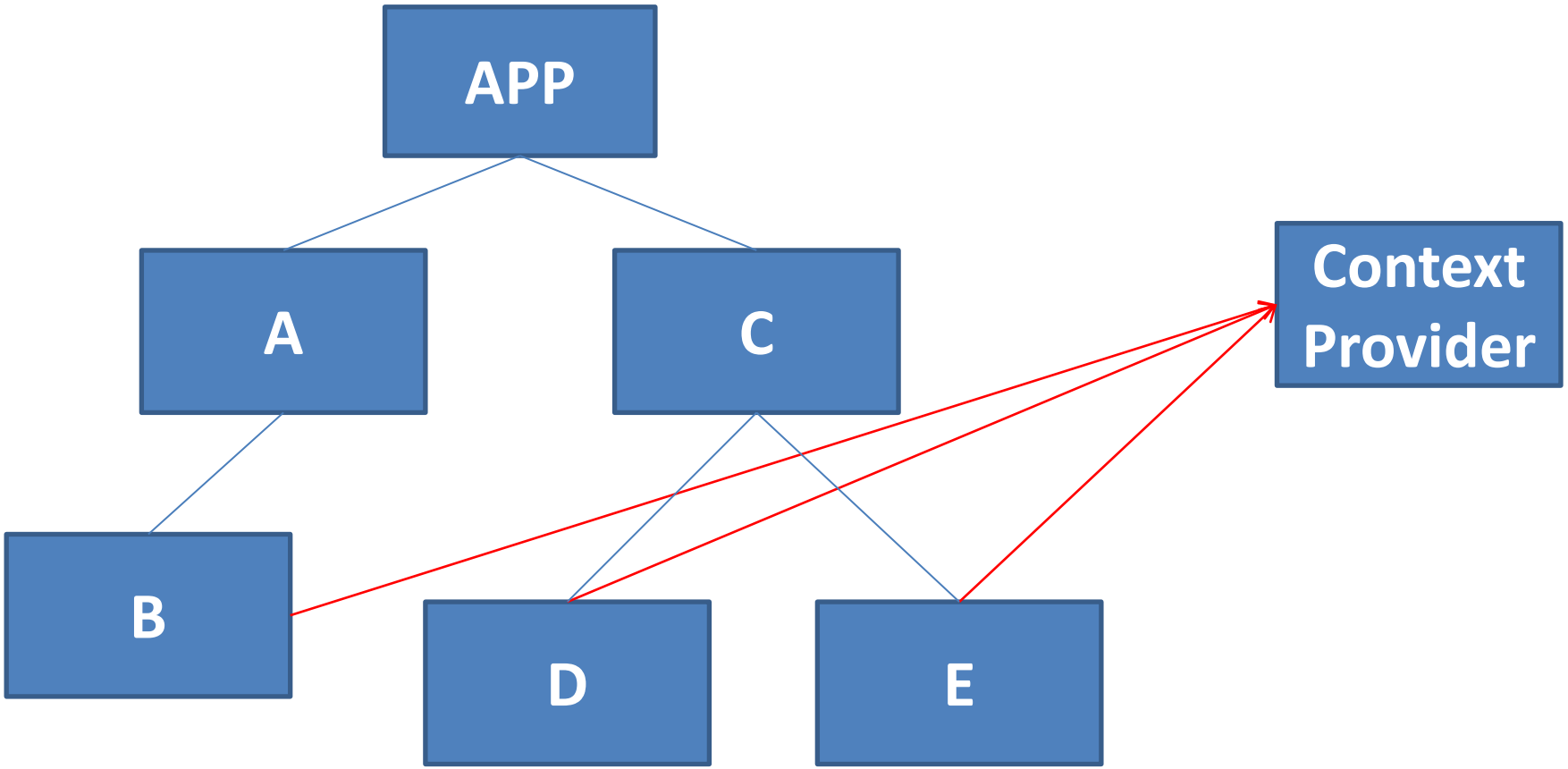
10. Communication between multi-layered components

2. Context API = A component, that shares its one state with other components



Task

1 - Implement the following component architecture using the Context API.



Task

2 – The context has two variables, x and y. Both are initially set to 0. Furthermore, the context has two functions incrementX and decrementY. incrementX sets x to x + 1 and decrementY sets y to y – 1. Implement the context!

3 – component B has a button that calls incrementX

4 – component D shows the current value of x and y.

5 – component E has a button that calls decrementY

10. Communication between multi-layered components

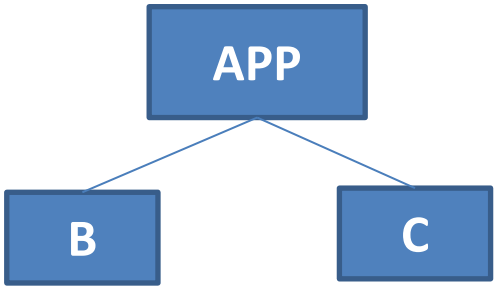
Context API functions can be indirectly called via a **Reducer** which maps messages to function calls.

„INCREMENT_X“ -> incrementX()

„DECREMENT_Y“ -> decremenY()

10. Communication between multi-layered components

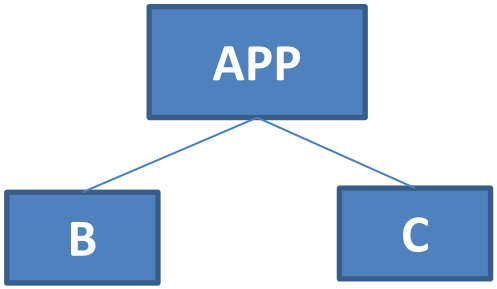
3. Redux = One or multiple shared states managed by a store

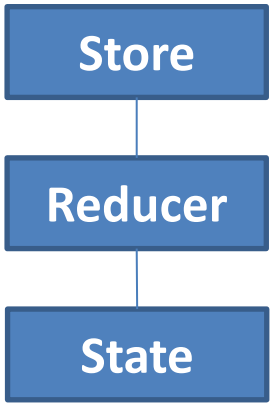


1. Create your App and it‘s child components

10. Communication between multi-layered components

3. Redux = One or multiple shared states managed by a store



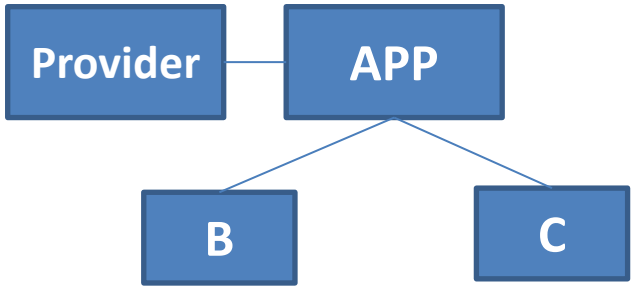


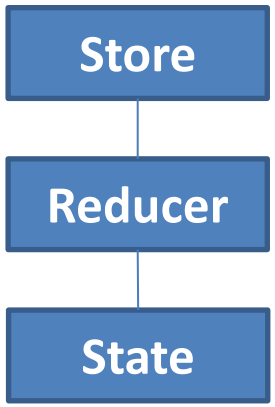
x=0, y=0

1. Create your App and it‘s child components
2. Create your Store, a Reducer and an initial State

10. Communication between multi-layered components

3. Redux = One or multiple shared states managed by a store



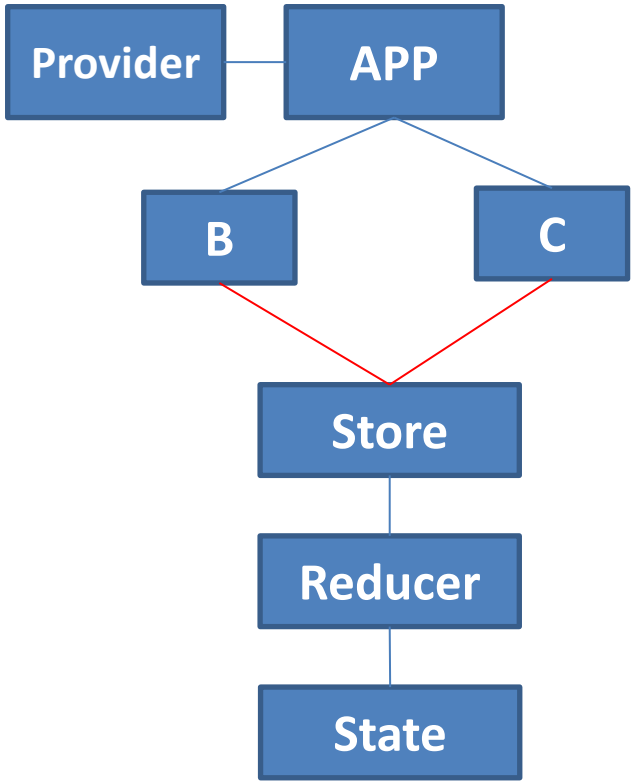


x=0, y=0

1. Create your App and it‘s child components
2. Create your Store, a Reducer and an initial State
3. Wrap your Provider around the App component

10. Communication between multi-layered components

3. Redux = One or multiple shared states managed by a store

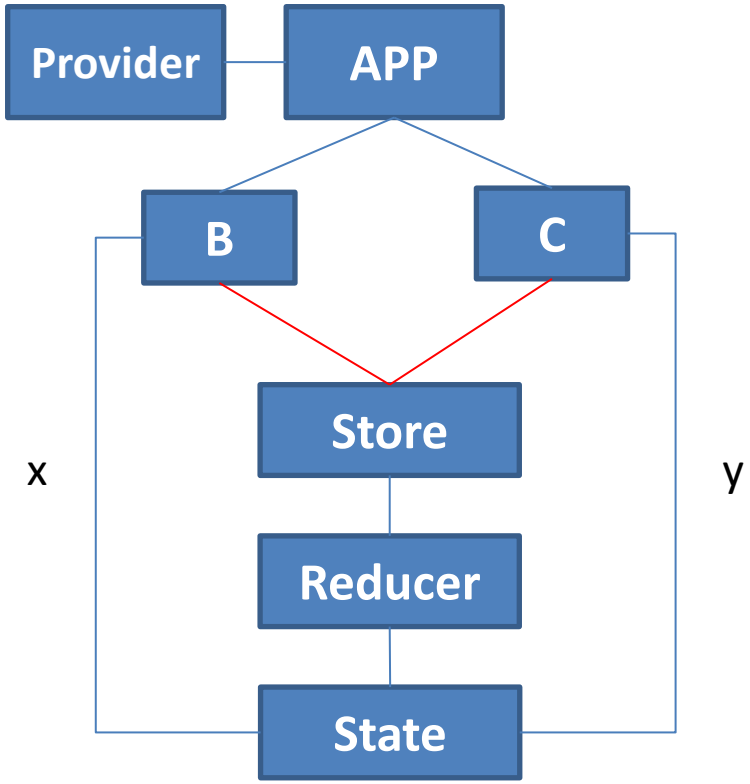


x=0, y=0

1. Create your App and it‘s child components
2. Create your Store, a Reducer and an initial State
3. Wrap your Provider around the App component
4. Connect B and C the store

10. Communication between multi-layered components

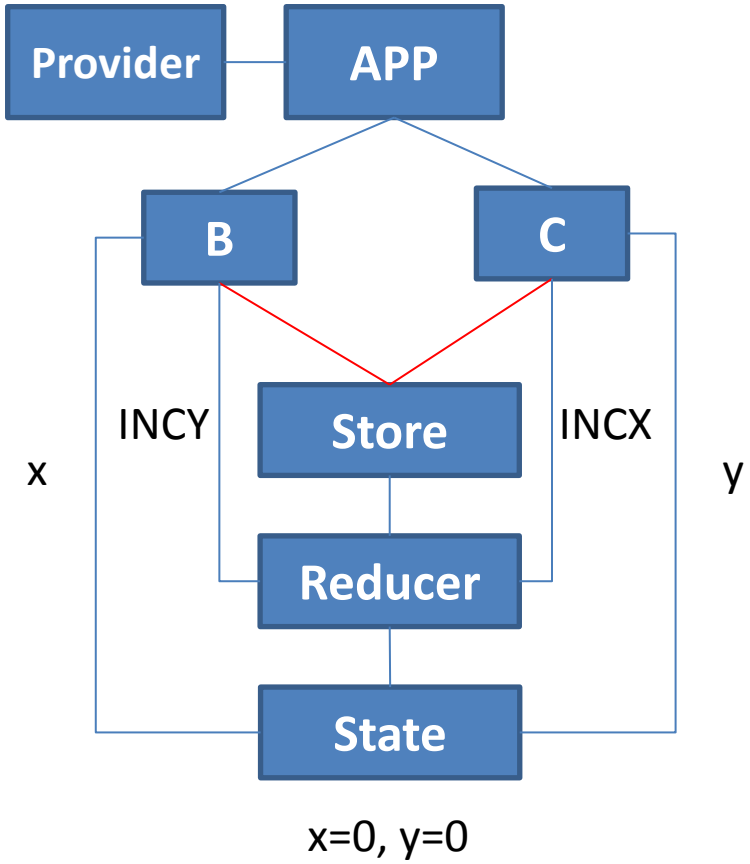
3. Redux = One or multiple shared states managed by a store



1. Create your App and it‘s child components
2. Create your Store, a Reducer and an initial State
3. Wrap your Provider around the App component
4. Connect B and C the store
5. Map the Reducer‘s state variables to the props B and C

10. Communication between multi-layered components

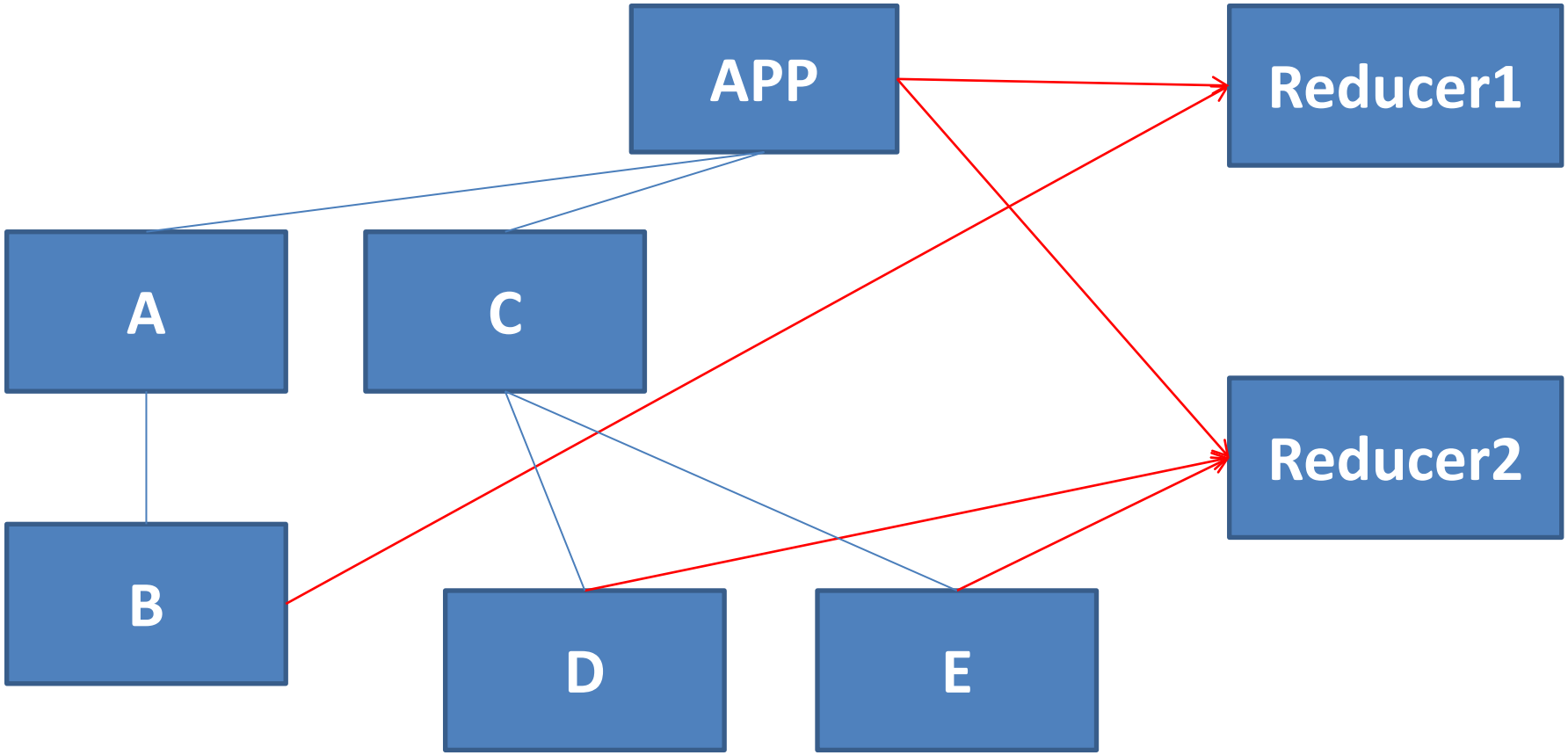
3. Redux = One or multiple shared states managed by a store



1. Create your App and it‘s child components
2. Create your Store, a Reducer and an initial State
3. Wrap your Provider around the App component
4. Connect B and C the store
5. Map the Reducer‘s state variables to the props B and C
6. Map the Reducert to props of B and C

Task

1 - Implement the following component architecture using Redux.



Task

2 – Reducer1 has one variable a. Reducer2 has two variables b = 0 and c = 1.

3 – component B has a button that generates a new randomstring with length 10 and saves it as Reducer1‘s a.

4 – component D has a button that sets Reducer2‘s b to b + 2.

5 – component E has a button that sets Reducer2‘s c to c + 2.

6 – The App component shows all variables a, b and c.

7 – The App component decides, that whenever b > 10 or c > 11, b will be reset to 0 and c to 1.