React Training

Understand 100%

by

Many Small Examples

&

Many Small Tasks

jan.schulz@cileria.com

About this course

- Not just a video tutorial - <u>you need to solve many tasks</u>
- Therefore, it is rather a **training program**
- It requires you have knowledge in
  - <u>Basic JavaScript</u>: Variables, Functions, Loops, Arrays, Objects, Conditional Statements
  - <u>Basic HTML/CSS</u>: Selectors, Div, Block VS Inline Elements, Box-Model (margin, padding, border), Flexbox, Tables
- I promise you: **Right after you finished this very intensive training program, you will be ready to be productive in React.**

Agenda – Part 1
Advanced JavaScript

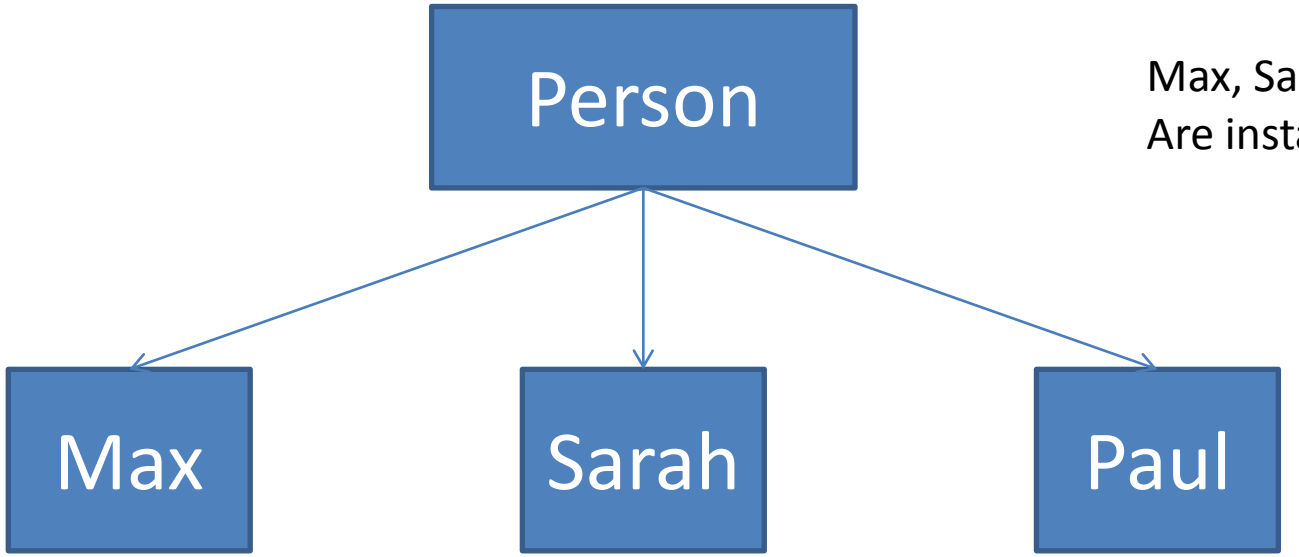
1. Const, Let and Var
2. Function Constructors & Classes
3. Arrow Functions
4. Function References
5. Promises / Async / Await
6. Destructuring
7. Spread Operator
8. Key Interpolation

1.1. const, let and var

- Var
  - Defines a variable that can be re-assigned and which is visible outside of an anonymous block
- Let
  - Defines a variable that can be re-assigned and which is not visible outside of an anonymous block
- Const
  - Defines a variable that cannot be re-assigned and which is not visible outside of an anonymous block

1.2. Function Constructors and Classes

- Function Constructors create objects
  - Like a blueprint for objects



Max, Sarah and Paul
Are instances of Person

1.2. Function Constructors and Classes

- Function constructors are **blueprints for objects**

```
function Person(name, age, job) {
    this.name = name;
    this.age = age;
    this.job = job;
}

Let max = new Person(‘Max‘, 35, ‘coder‘);
Let sarah = new Person(‘Sarah‘, 30, ‘designer‘);
Let paul = new Person(‘Max‘, 45, ‘taxi-driver‘);
```

3. Constructors and Instances

```
var john = {
        Name: ‘John‘,
        yearOfBirth: 1990,
        isMarried: false
}
```

```
var jane = {
        Name: ‘Jane‘,
        yearOfBirth: 1991,
        isMarried: true
}
```

```
var mark = {
        Name: ‘Mark‘,
        yearOfBirth: 1948,
        isMarried: true
}
```

3. Constructors and Instances

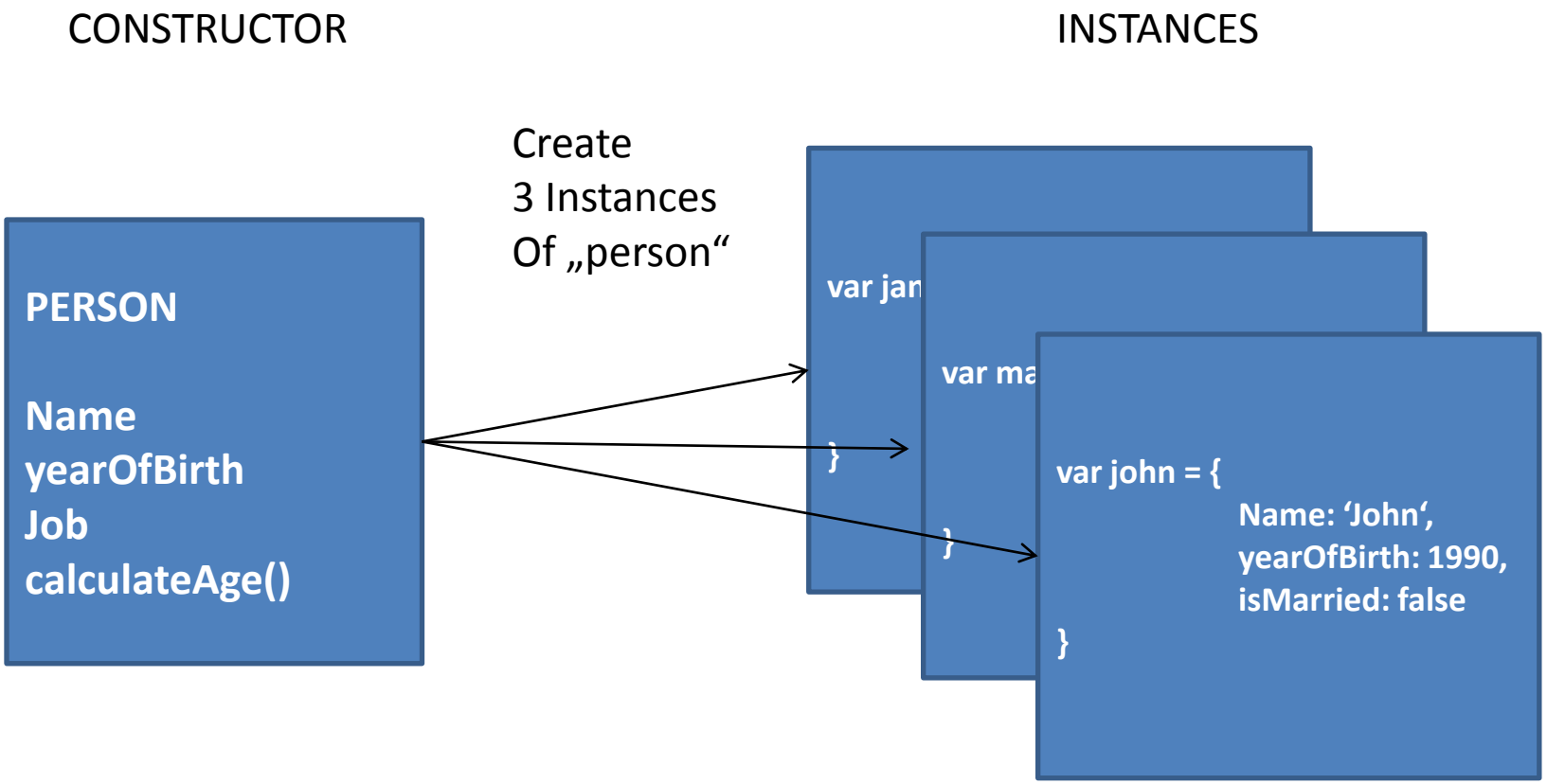
```
var john = {
        Name: 'John',
        yearOfBirth: 1990,
        isMarried: false
}
```

```
var jane = {
        Name: 'Jane',
        yearOfBirth: 1991,
        isMarried: true
}
```

```
var mark = {
        Name: 'Mark',
        yearOfBirth: 1948,
        isMarried: true
}
```

3 Objects = A lot of typing

3. Constructors and Instances



4. Inheritance

PERSON

Name
yearOfBirth
Job
calculateAge()

4. Inheritance

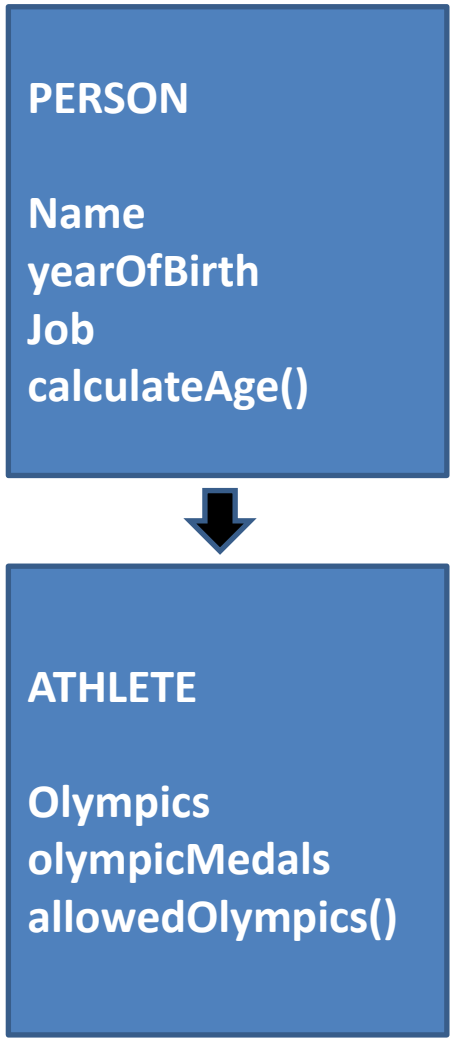
PERSON

Name
yearOfBirth
Job
calculateAge()

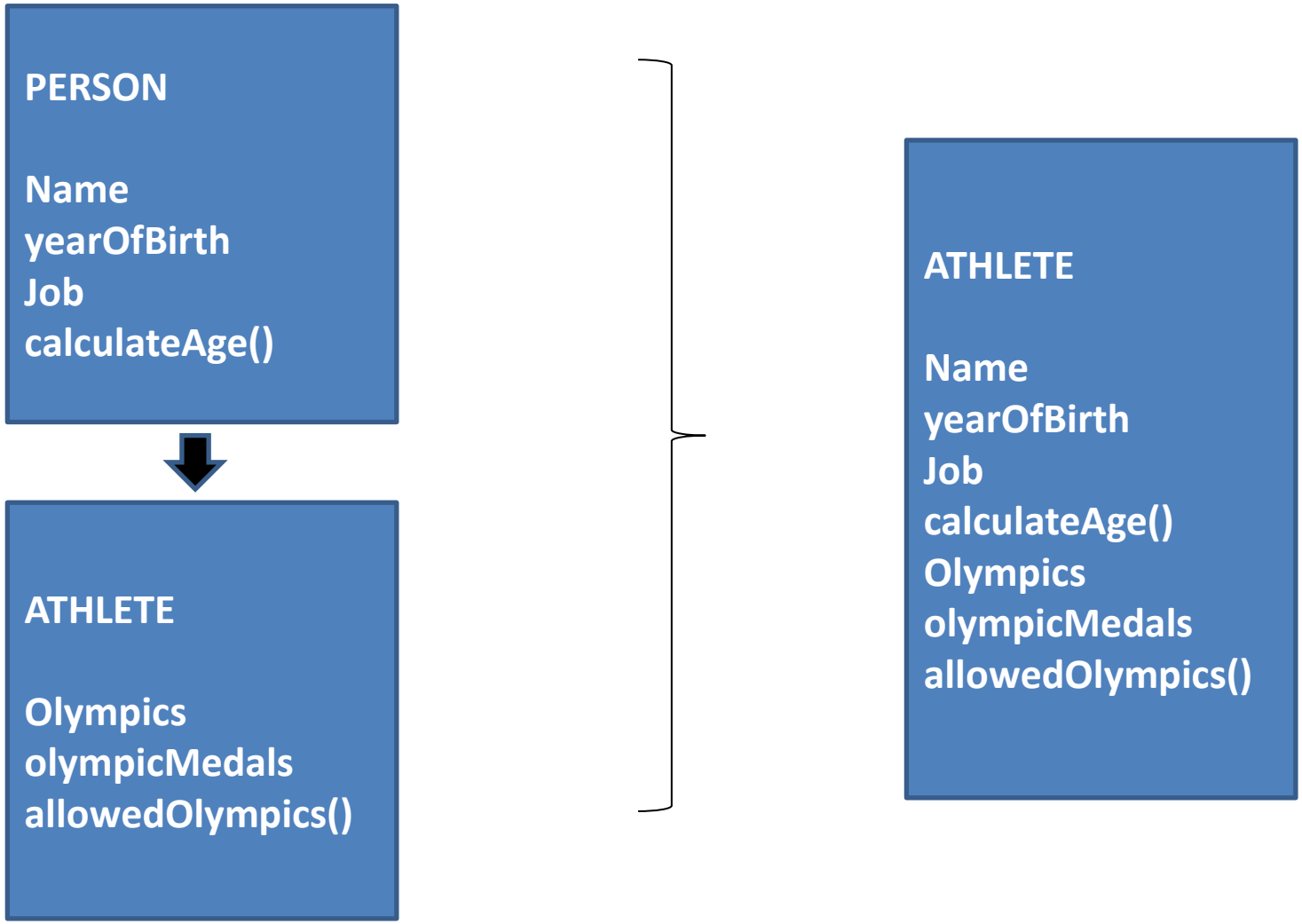
ATHLETE

Olympics
olympicMedals
allowedOlympics()

4. Inheritance



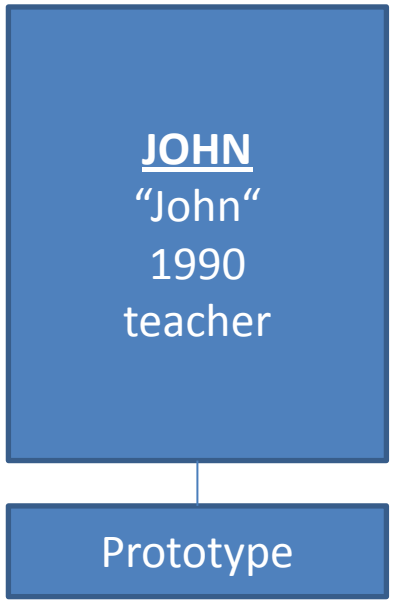
4. Inheritance



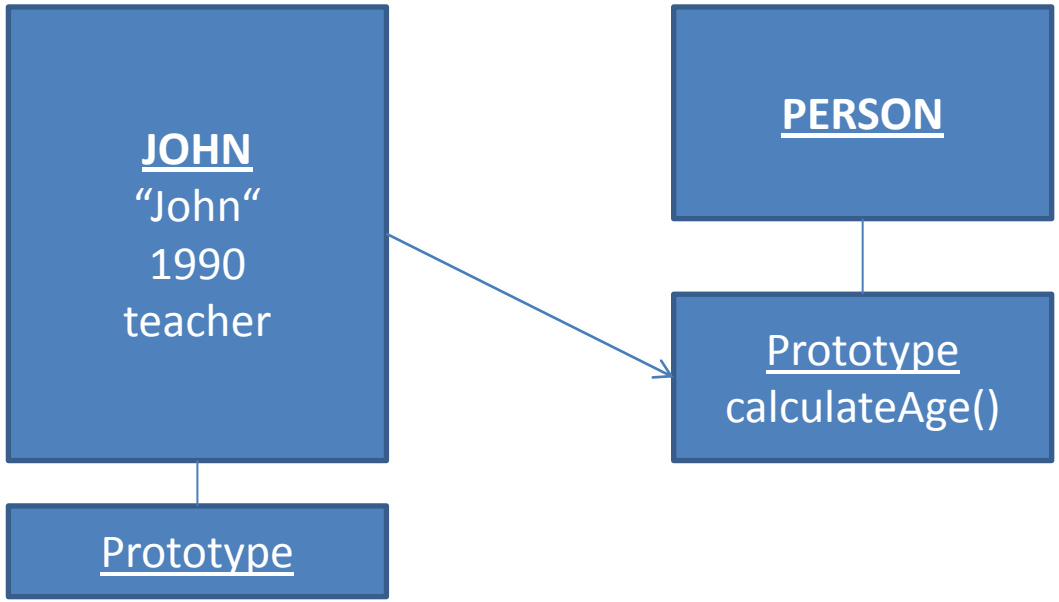
5. Prototype

- Every object in JavaScript has an attribute called **prototype**
- Each prototype has an attribute, which itself a prototype
- This goes on, **until prototype is null**

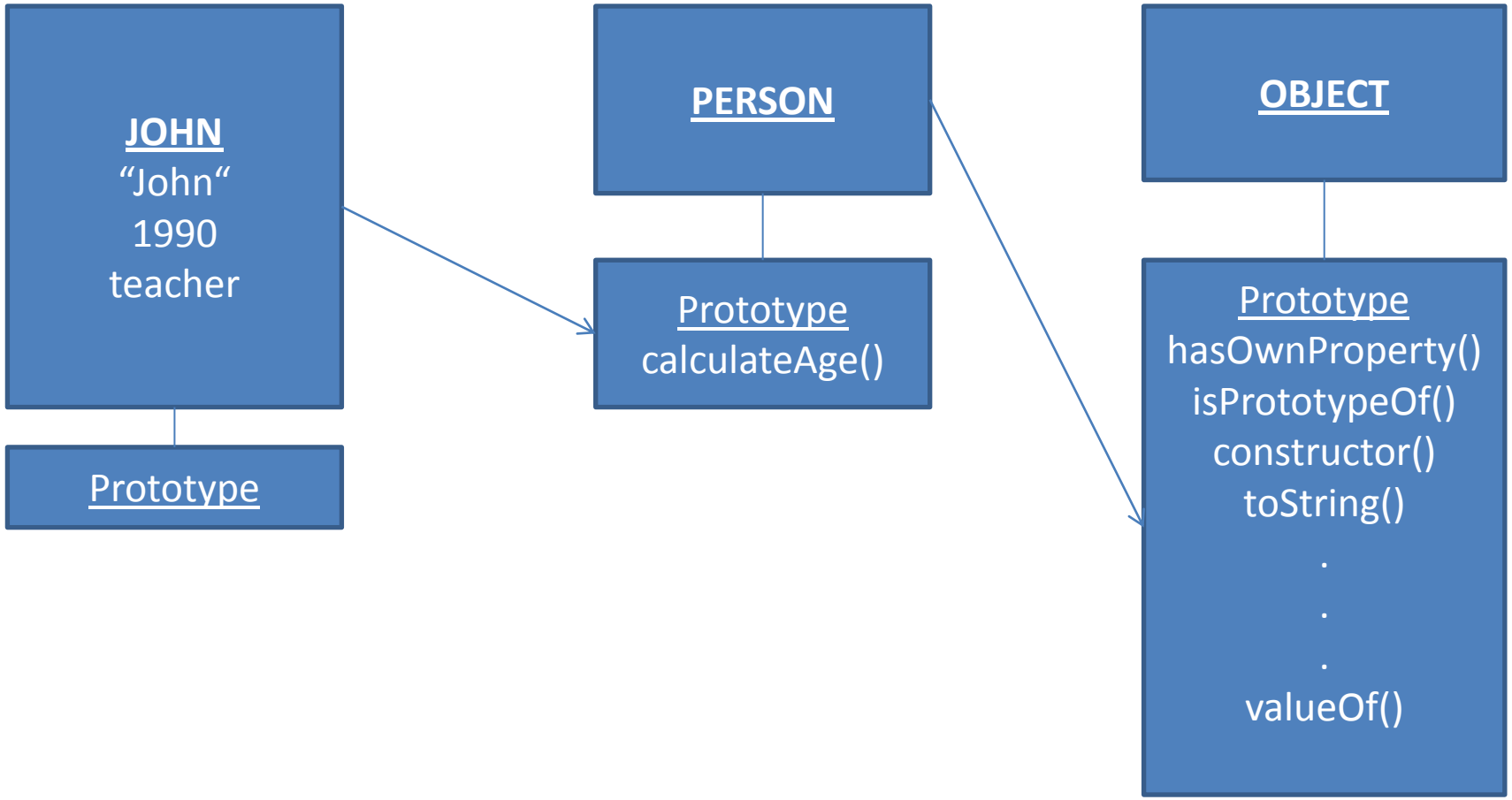
6. Prototype-Chain



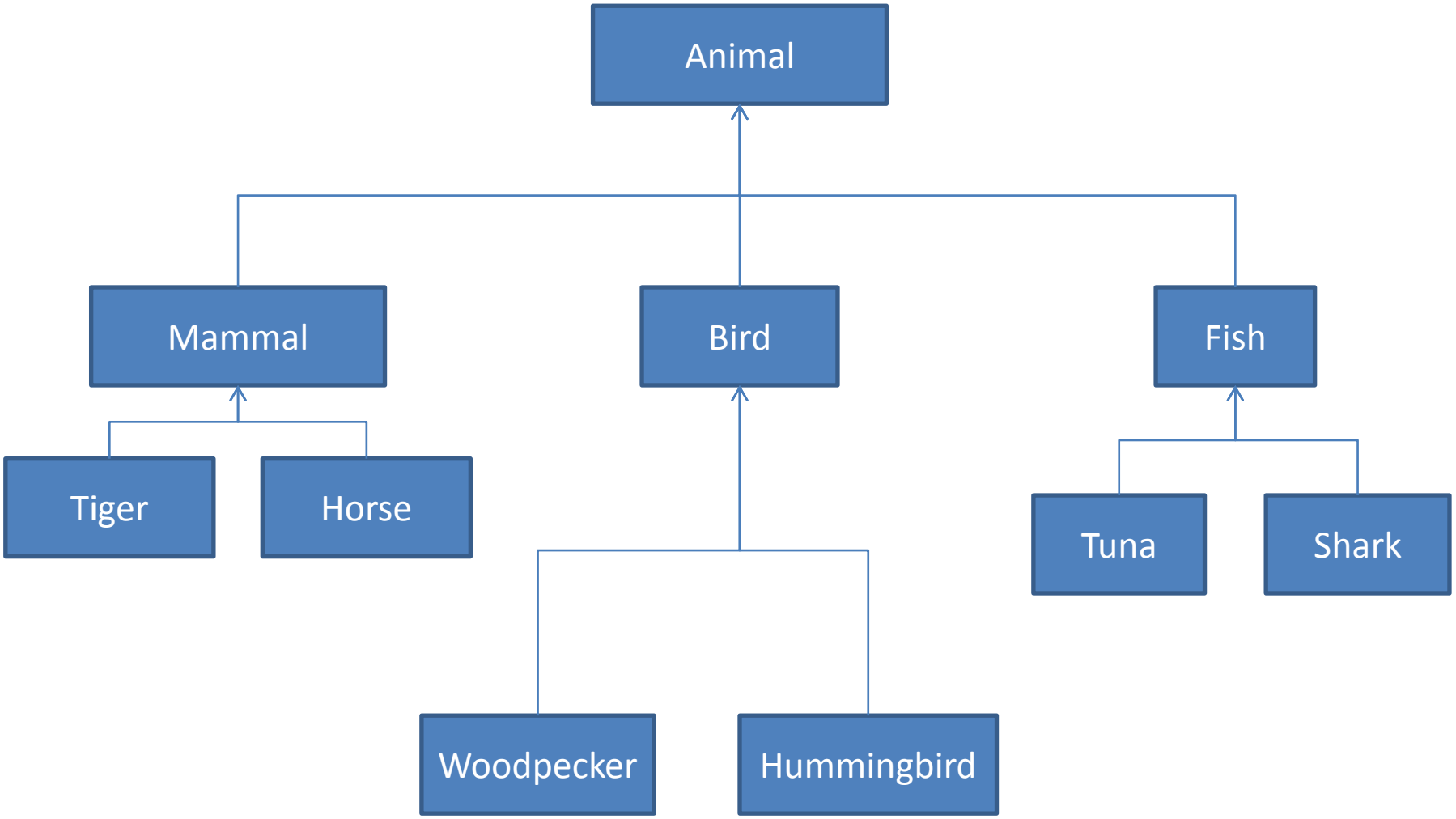
6. Prototype-Chain



6. Prototype-Chain



1.2. Task: Multi-Level Inheritance



12. Multi-Level Inheritance: Task

1. With your knowledge about Inheritance, please create the Function constructors according to the Animal diagram and consider the following rules:
   1. Each animal has a **name** that is set when it is constructed.
   2. All animals can **sleep**, **eat** and **die** (use functions for this, e.g. **sleep()**)
   3. Mammals and birds can **breathe**.
   4. Fishes can **swim**.
   5. Birds can **fly**.
   6. Tigers and Sharks can kill, whereas **kill()** expects one parameter **otherAnimal.** Kill() calls the die() function o**f otherAnimal**.
2. Create one tiger with name „Vitaly“, one Shark with name „Nemo“, one horse with name „Fury“.
3. Nemo is hungry and kills Fury and Vitaly. Then Nemo eats.
4. Nemo dies.

13. Class Keyword

- The class keyword is **syntactic sugar** for defining prototypes

```
class Person {
    constructor(name, age, job) {
        this.name = name;
        this.age = age;
        this.job = job;
    }

    calculateAge() {
        return 2018 – this.age;
    }
}
```

Agenda – Part 2
Introduction to React

1. HalloWorld
2. JSX
3. CSS
4. Sub-Components

Task 2.3.

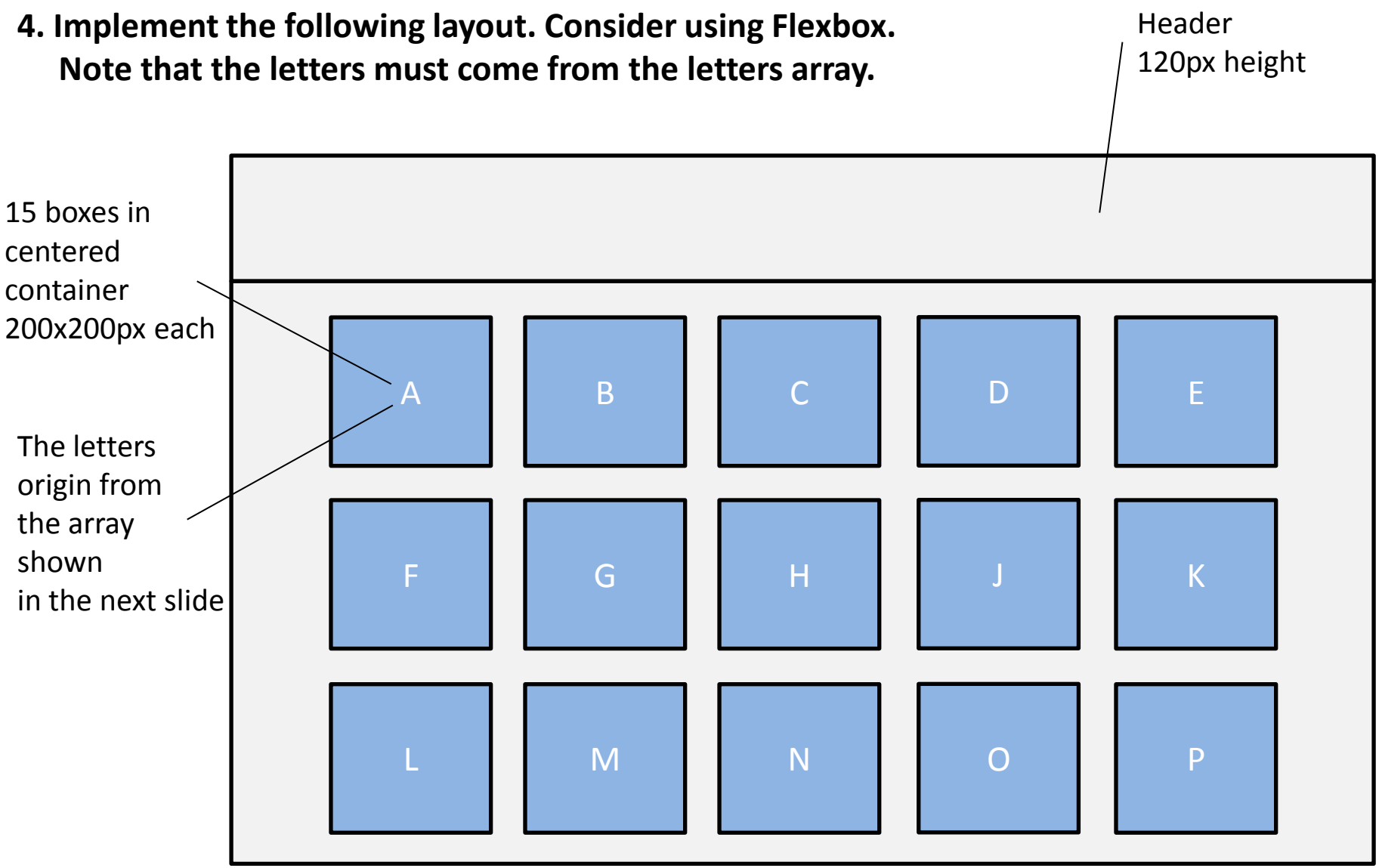
1. Create a new react web app „task-2-3“
2. First, create the following constant string inside the render() – method.

   const letters = ‘abcdefghijklmnop‘;

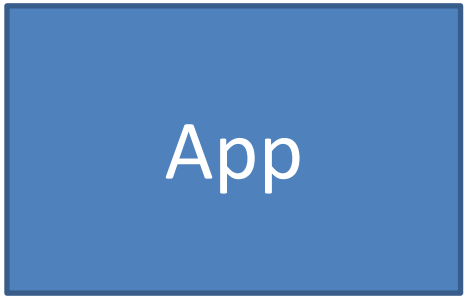
3. Convert the string into an array of characters.

Task 2.3.

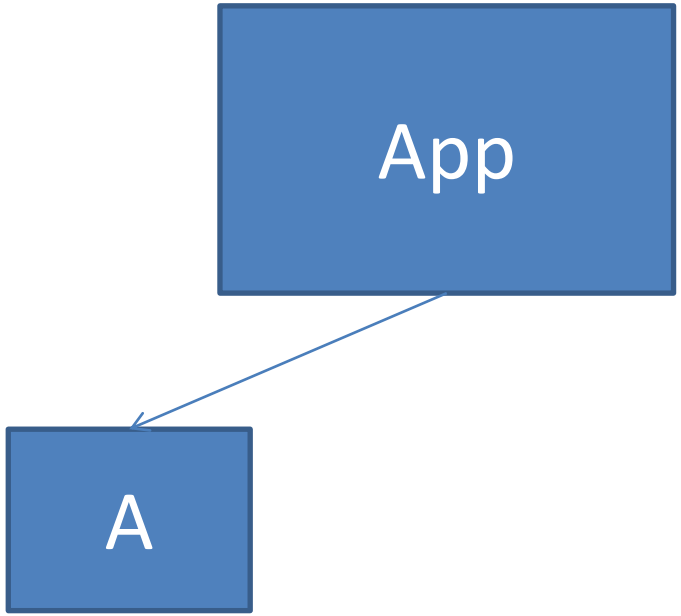
4. Implement the following layout. Consider using Flexbox.
Note that the letters must come from the letters array.



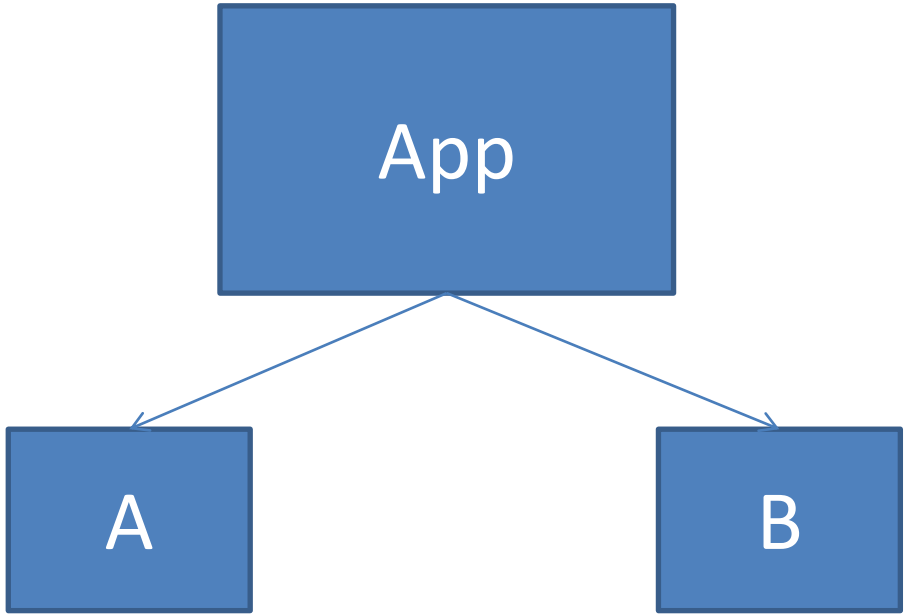
4. Sub-Components



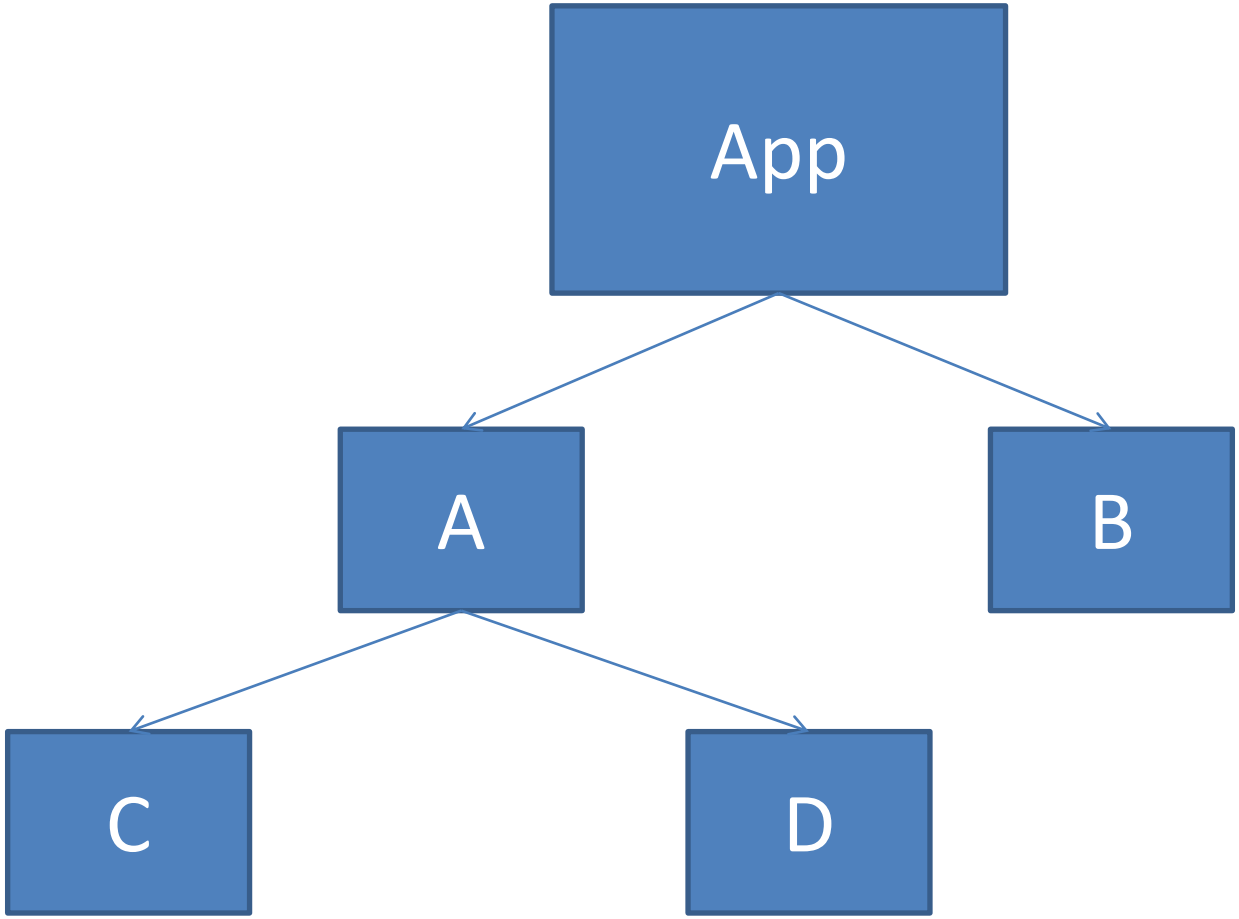
4. Sub-Components



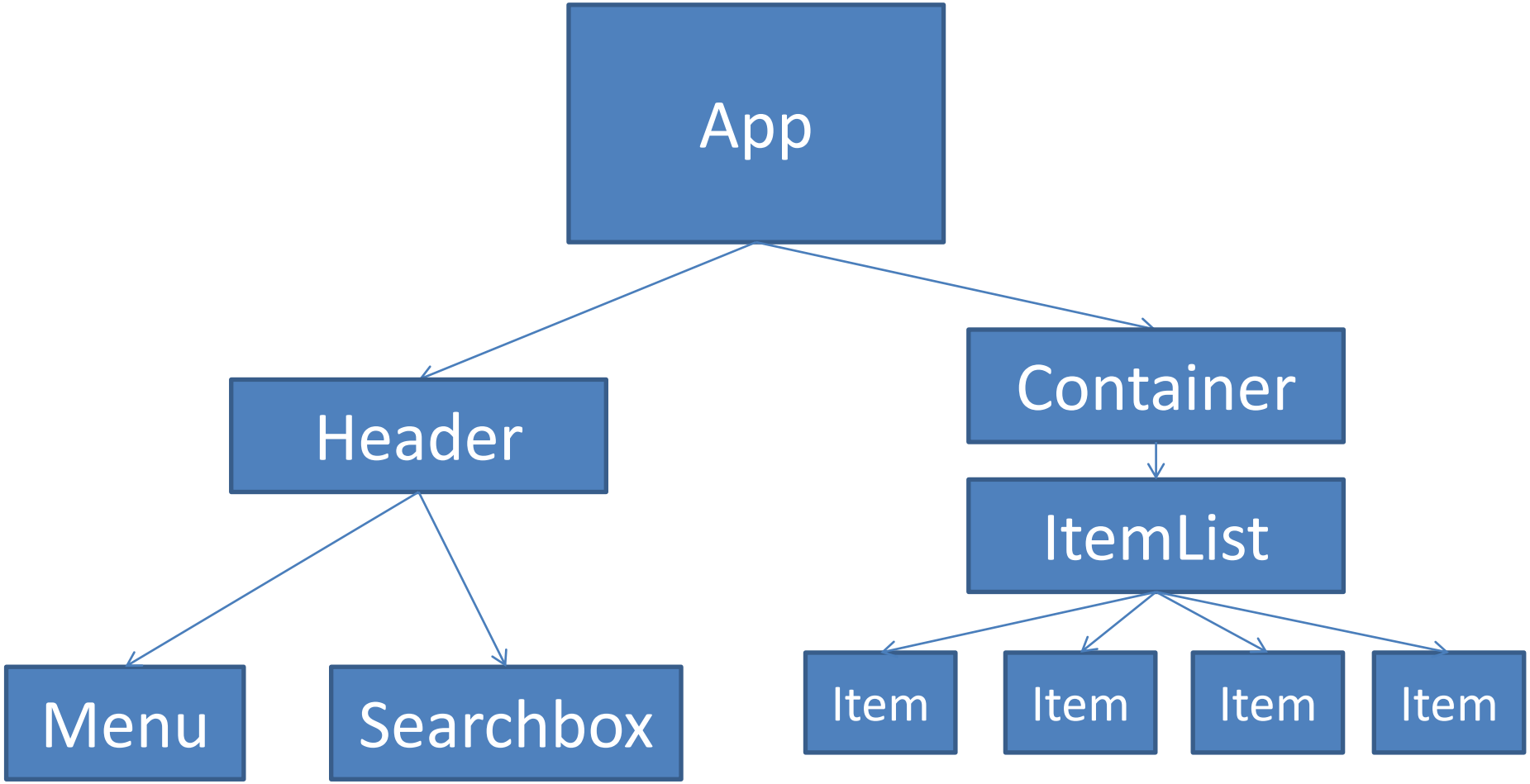
4. Sub-Components



4. Sub-Components



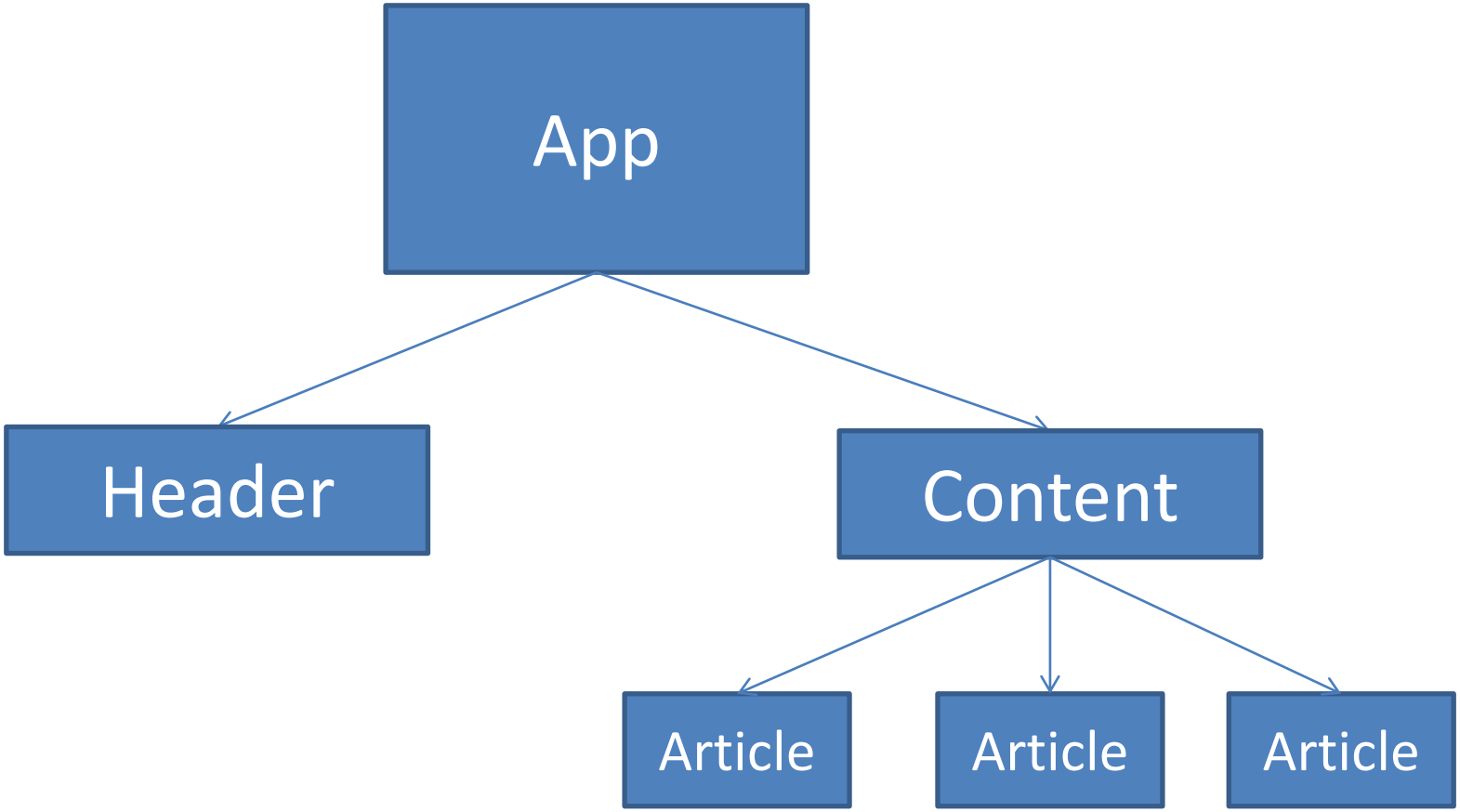
4. Sub-Components



Task 3.4. – Sub-Components

1. Create a new React web app „task-3-4“
2. Create two components „Header“ and „Content“. Put them in the App component.
3. In the Header component, create an h1-Element containing the words „Hallo World“. Give Header the background-color „cornflowerblue“ and a height of 80px.
4. Create the component „Article“. It contains a div with background-color a bit brighter than cornflowerblue. Inside the div, write a random text of 25 characters.
5. Put three instances of Article inside the „Content“-component.

Task 3.4. – Sub-Components



Agenda – Part 4
States & Events

1. Changing State
2. Task
3. Events
4. Task
5. Events that change the state
6. Task

4.1. What is a state?

- **A component‘s state is an object named „state“ inside the component**
  1. It is accessible via this.state
  2. If the state changes, the render() method will be called again
  3. The state can only be changed using this.setState() – it cannot be changed directly

Task 4.1.

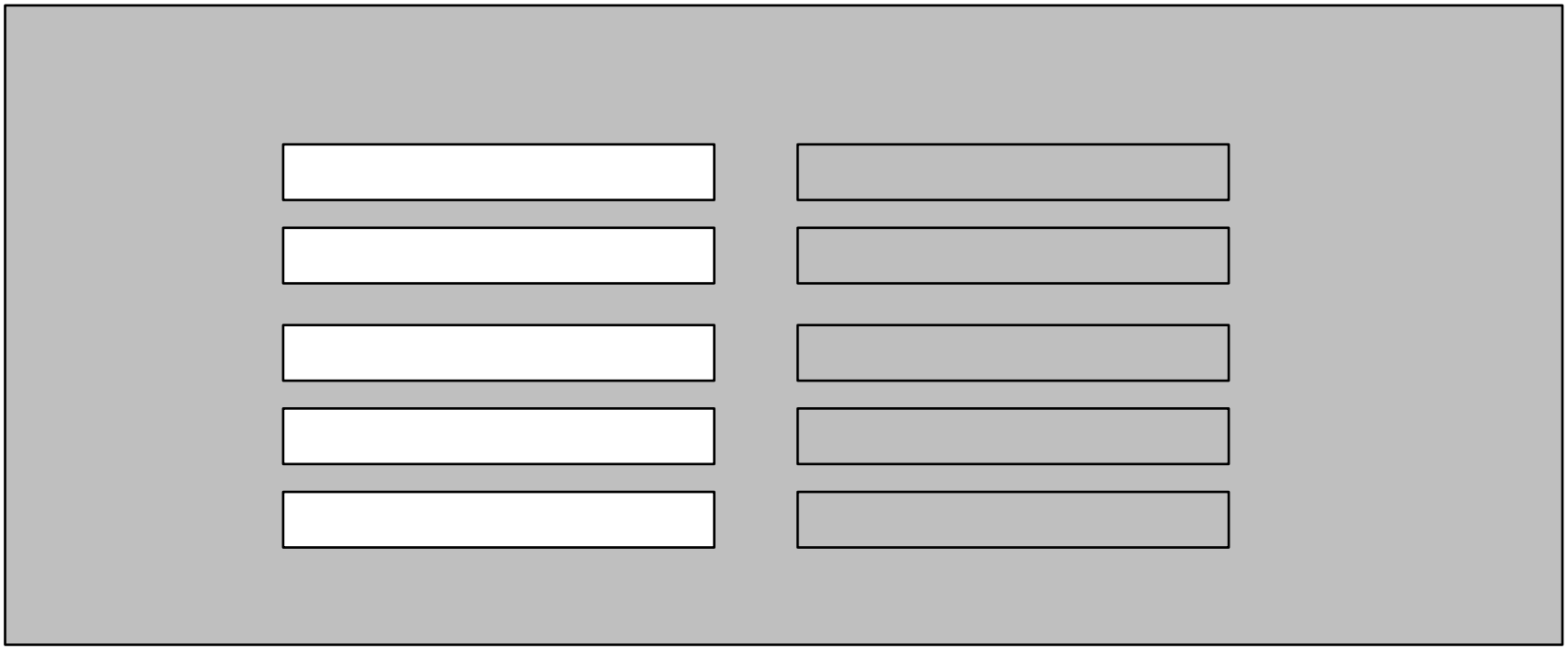
1. Create a new web app „task-4-2“
2. Create a new state in the App component and add a new empty array „randomstrings“ to it.
3. „randomstring“ is an array of strings. Create a function randomstring(n) inside the App component that returns a random string of length n. I.e.
   1. randomstring(3) could return ‘ghj‘.
   2. randomstring(7) could return ‘fkdllbx‘
4. Add an interval that adds a new randomstring to randomstrings array every second.
5. Each time a new randomstring is added, append a new div-Element which contains the new randomstring to the App component.

4.2. - Events

- **What is an event?**
  - Something that happens.
  - The user clicks the mouse, types text, presses a button on the keyboard, the window is being resized, ... etc.
- In React, we attach events to elements
- Overview of events:
  - https://reactjs.org/docs/events.html

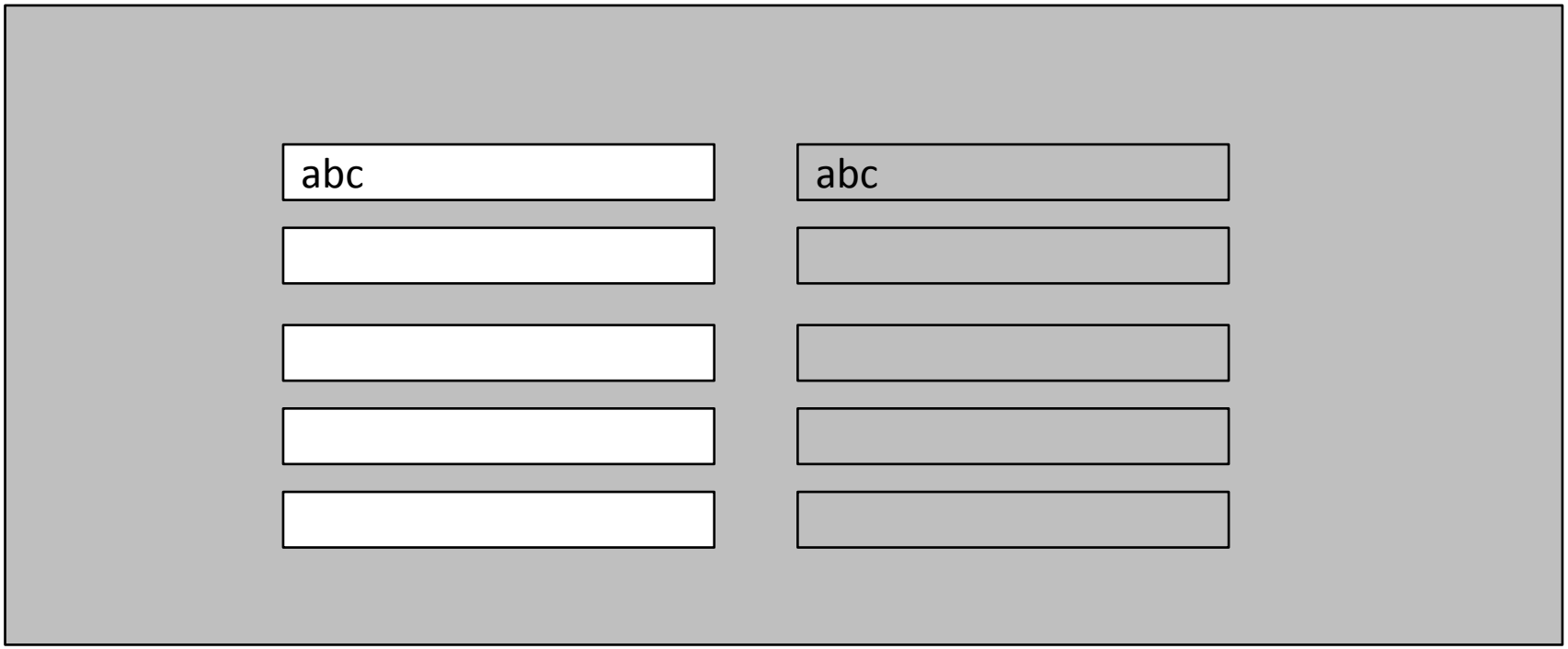
Task 4.2.

1. Create a new web app „task-4-2“
2. Inside the app component, create 5 text boxes and next to them a span each. Give each textbox an unique name attribute, e.g. <input type=“text“ name=“txtBox1“ />



Task 4.2.

3. When in text box 1 the user enters text, the text should appear in the span next to it, like this:



Task 4.2.

4. Do the same for the textboxes 2 to 5 by creating 4 more handler functions.
5. Now you have 5 more or less identlical handler functions. Go into one of these handler functions (does not matter which one) and take a closer look at event.target.name - What do you notice?
6. Try to replace the 5 handler functions with one single handler function. Consider using Key Interpolation. (also covered in Chapter 1.8. of this course). In case you forgot about Key Interpolation, it is accessing an object‘s key by a string, e.g.:

   const key = ‘foo‘;
   const obj = {x: 1, foo: ‘hallo‘}
   obj[key] = ‘hi‘; // changes foo to ‘hi‘

Task 4.4.

1. Make a copy of the 03_state example and save it as 04_task
2. Change the text of the button „show index“ to „remove“.
3. Implement the functionality of the remove – button. Try to use the index to access the array in the state for the removal of the element, consider using splice() – or any other method that would do the same job.
4. Add another button for each fruit with the the text „new color“.
5. Implement the functionality of the „new color“ – button.

Agenda – Part 5
Communication between two components

1. Stateful VS Stateless Components
2. Task
3. Downward Communication via Passing Props
4. Task
5. Upward Communcation via Passing Function References
6. Excourse: Radio Buttons
7. Task
8. Subcomponents with an Ending-Tag

5.1. Stateful VS Stateless Components

- A stateful component is a component
  - That has the state-Object inside
- A stateless component is a component
  - That **DOES NOT** have the state-Object inside

- Why?

5.1. Stateful VS Stateless Components

- A stateful component is a component
  - That has the state-Object inside
- A stateless component is a component
  - That **DOES NOT** have the state-Object inside

- Why?
  - Stateless Components (also known as **Functional Components**) are supposed to have only little logic inside and their purpose is to represent layout parts of the app

5.1. Stateful VS Stateless Components

- A stateful component is a component
  - That has the state-Object inside
- A stateless component is a component
  - That **DOES NOT** have the state-Object inside

- What does this imply?

5.1. Stateful VS Stateless Components

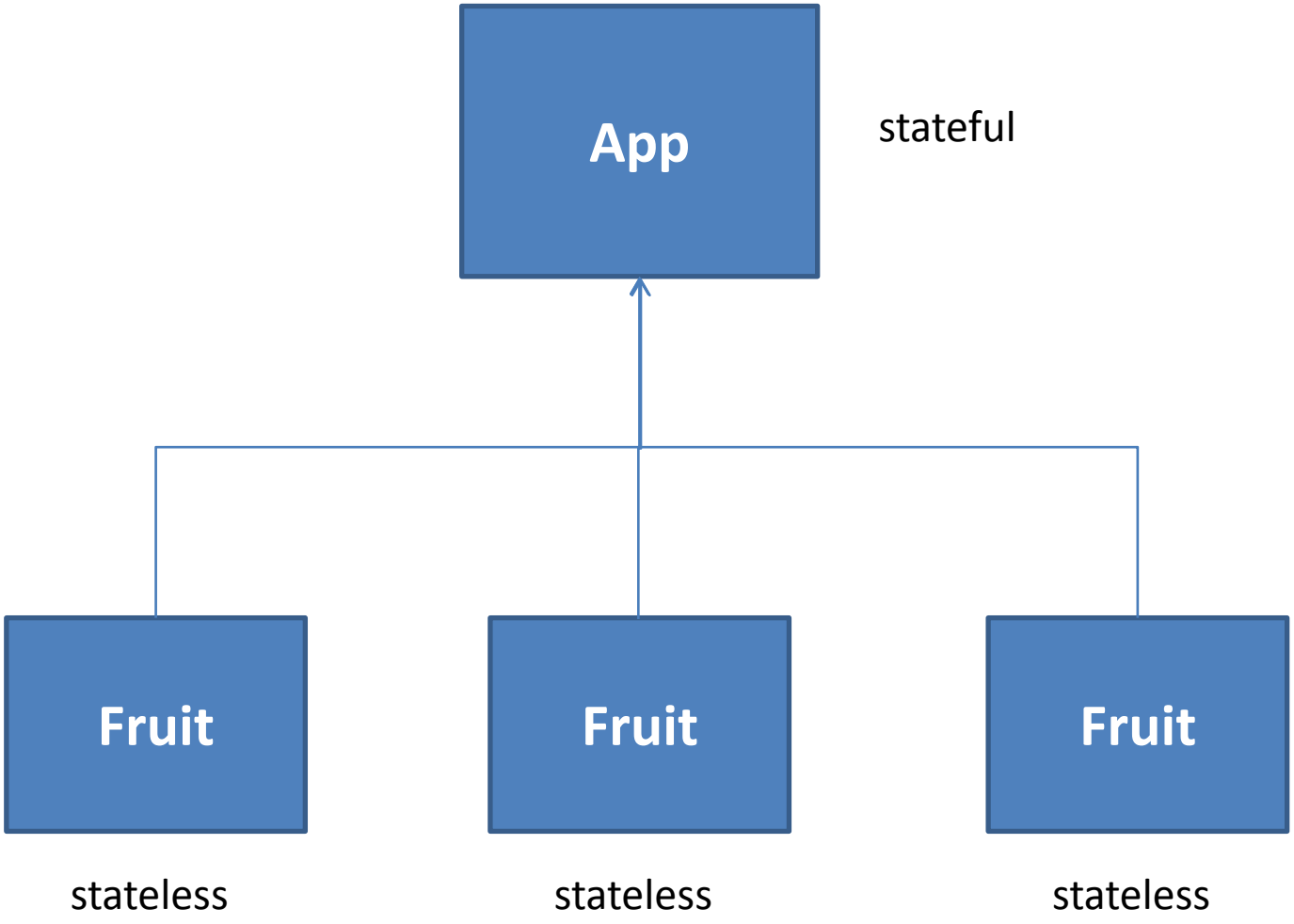
- A stateful component is a component
  - That has the state-Object inside
- A stateless component is a component
  - That **DOES NOT** have the state-Object inside

- What does this imply?
  - Stateful components have the setState() method to re-render themselves
  - Stateless components **DO NOT** have the setState() method to re-render themselves

5.1. Stateful VS Stateless Components

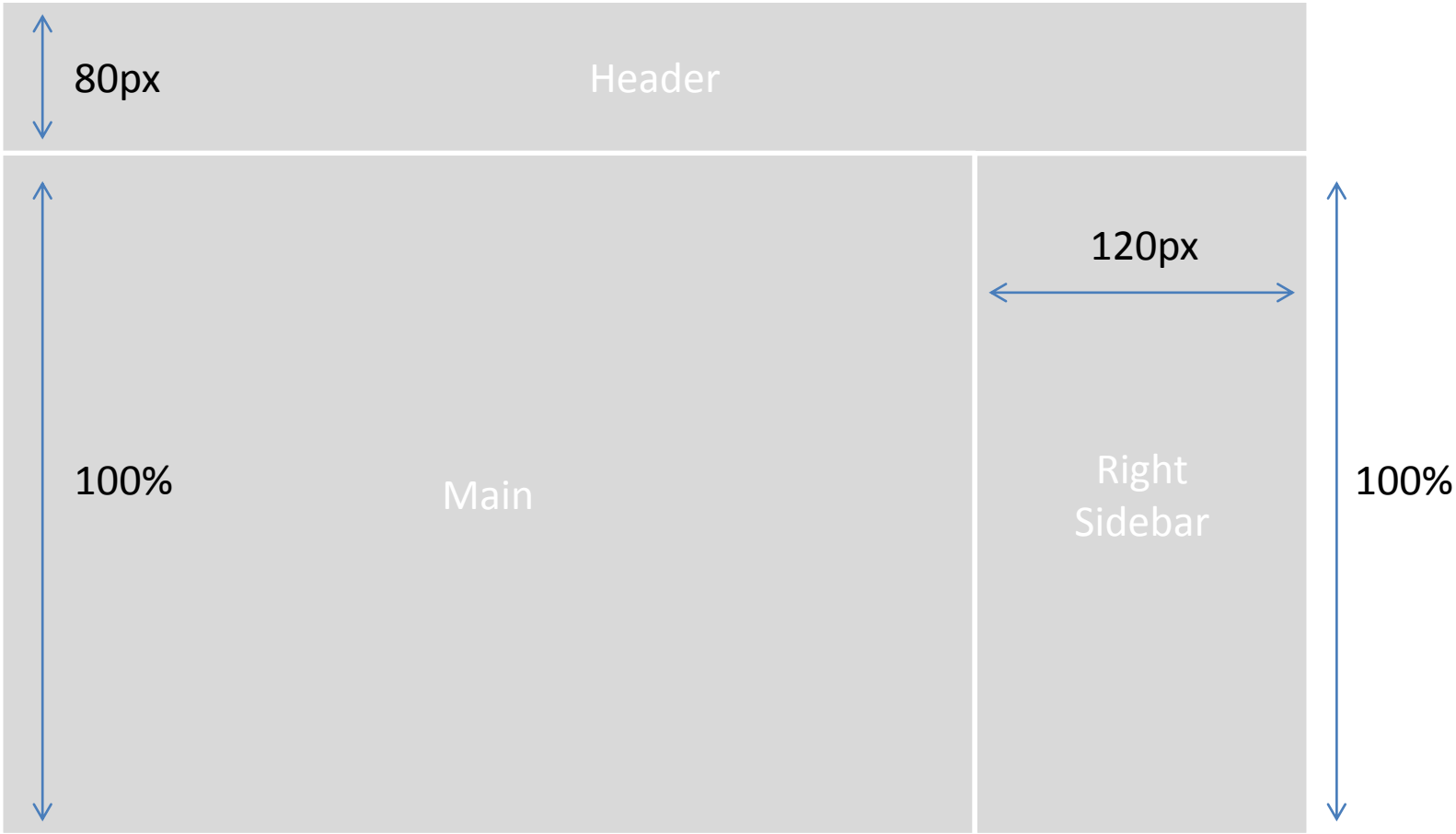
- How do Stateless Components re-render themselves?
  - **They do not** – The rendering is done when the first Stateful Parent Component re-renders, meaning it calls setState()
  - The App-Component can only be a Stateful Component – so when the App-Component calls setState(), all immediate Stateless Components will be re-rendered

5.1. Stateful VS Stateless Components



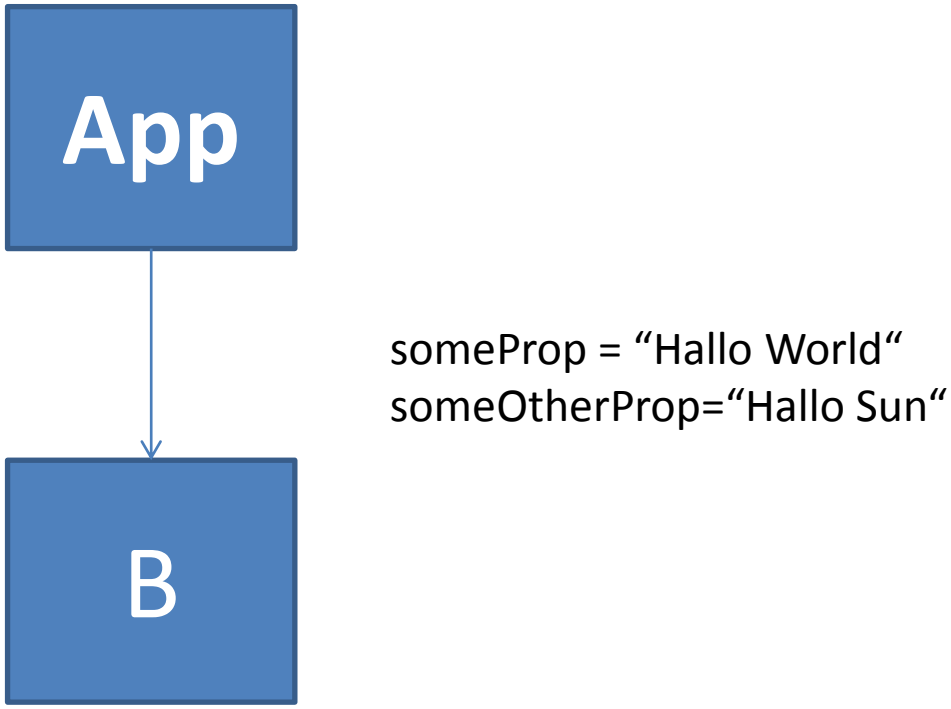
5.2. Task

1. Create a new react app task-5-2
2. Implement the following layout and create three components Header, Main and RightSidebar. Make Header and RightSidebar a stateless component and Main stateful.
3. The background-color of the Main component should change every 2 seconds to a random color.



5.3. Downward Communication With Props

- Communication from a parent component A down to its child component B can be done using **<u>props</u>**

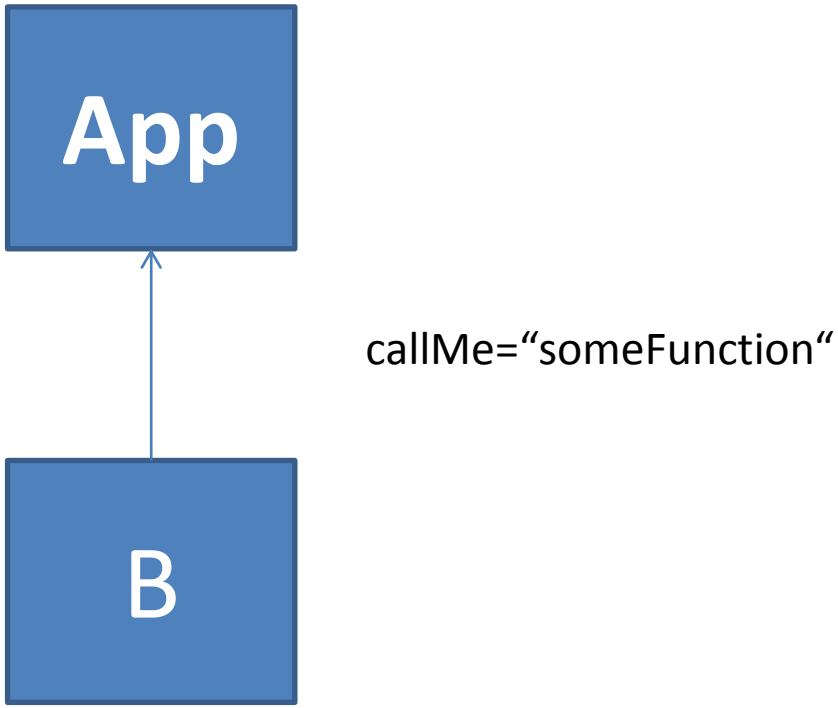


Task 5.4

1. Create a new react app task-5-4
2. Create a new stateless component “Fruit“.
3. Inside fruit, there is only one div with the background color red and text inside „Apple“.
4. Create 3 instances of Fruit in your App component.
5. Add one prop the Fruit-component: color
6. Now, give each Fruit component it‘s color via the color prop.
7. Add a button to the App-Component: „Randomize Apples“.
8. Add the following functionality: If the user clicks the „Randomize Apples“ button, each of Fruit components gets a new background color. Possible colors are red, blue, green, black, yellow, pink, fuchsia and grey.

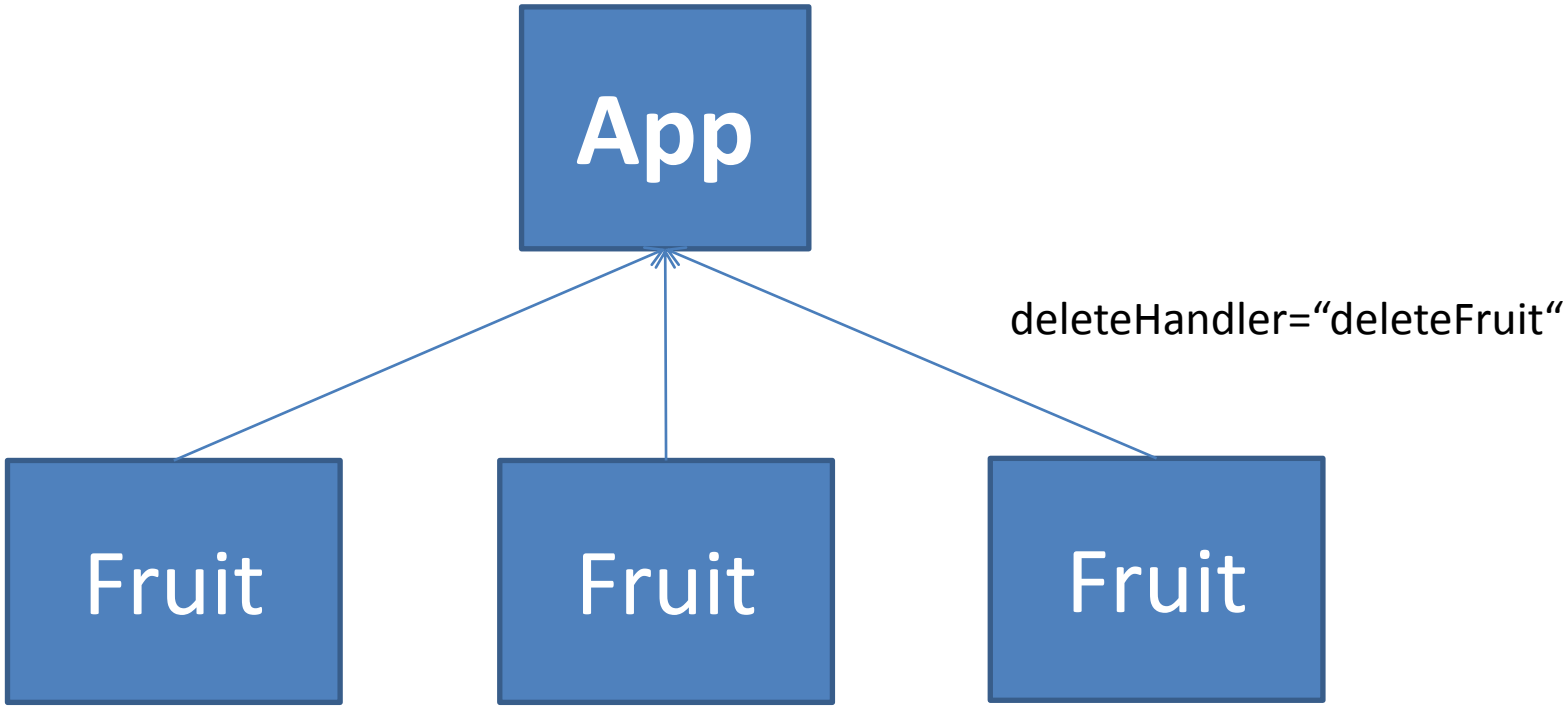
5.5. Upward Communication With Function References

- Communication from child component B upward to parent component A can be done using **<u>function references</u>**



5.5. Upward Communication With Function References

- A function being called in the parent component can change its state



Task 5.7

1. Create a new react app task-5-7.
2. In the state object, create an array “users“ which consists of 3 example user objects whereas each has a name and a distinct id. I.e. take the users Peter with id = 1, Sandra with id = 2 and Steven with id = 3
3. Create a textbox with the name “username“.
4. Create a component User.
5. Inside User, create a radio button and next to it, an empty span element.
6. Give the User component two props “username“ and “id“. The value of username is supposed to be shown inside the span element. The value of id is supposed to be the value of radio buttons.
7. Based on the users array in the App component‘s state, create instances of the User object.

Your layout now may look like this ... (next slide)

Task 5.7

- Peter
- Sandra
- Steven

5.8. Subcomponents with an Ending Tag

- Until now, all of our components are self-closing components:

  <A />

  <Fruit />

  <User />

- Self-Closing components cannot contain children, there we need Subcomponents with an Ending-Tag, i.e.

  <B></B>

Task 5.9.

1. Create a new react app task-5-9.
2. Create a new stateless component “MyButton“.
3. Inside MyButton, there is a div and inside that div, there is a button. All children of MyButton should be inside the button-Element.
4. Each MyButton has the background color of „cornflowerblue“, the font size of 20 pixels and a padding of 4 pixels.
5. When the button inside of button is clicked, the function reference onClick of MyButton will be called.
6. Create two MyButtons, one that contains the text “Hallo World“ and another one that contains a nice picture of a beautiful beach.
    1. When the first MyButton is clicked, an alert box shall appear saying “HalloWorld“
    2. When the second MyButton is clicked, an alert box shall appear saying “Beach Life! Me gusta!“

6. Component Lifecycle

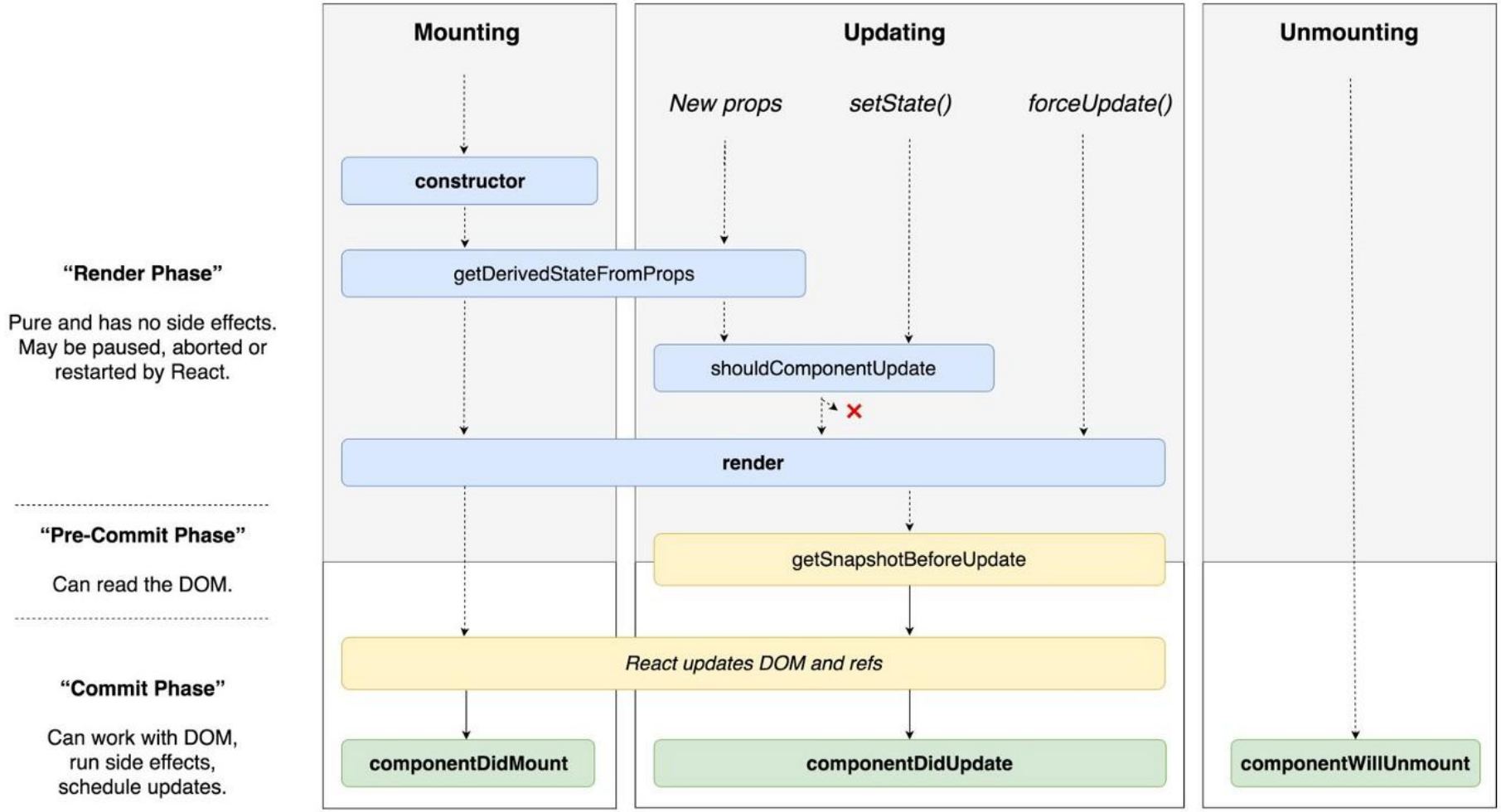
1. Overview
2. Mounting Phase
3. Updating Phase
4. Unmounting Phase
5. Task 1
6. Task 2 – Difficult

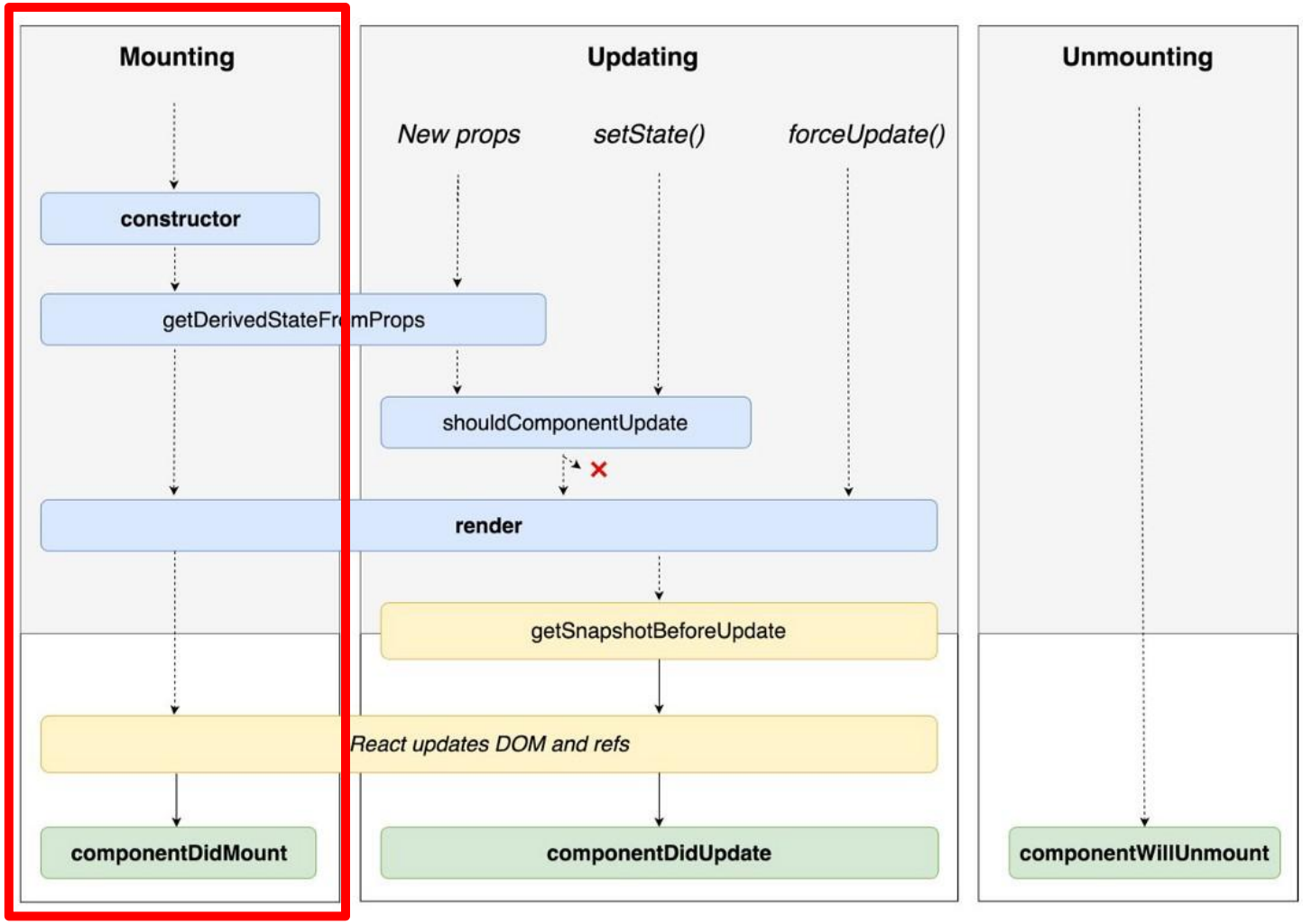
6.1. Overview

- Each Stateful Component
  - Is born = The component **mounts**
  - Lives = The component **updates**
  - Eventually Dies = The component **unmounts**
- In each of these lifecycle phases, React calls certain methods of the component
- Each method is initially empty!
- the developer writes code do define the component‘s behaviour when a lifecycle phase occurs

6.1. Overview

- Three lifecycle phases
  - Mounting Phase
    - React creates an instance of a component and inserts it into the DOM
  - Updating Phase
    - When props or state of a component are changed
  - Unmounting Phase
    - When the component is removed from the DOM


Mounting
Updating
Unmounting
New props
setState()
forceUpdate()
constructor
getDerivedStateFromProps
shouldComponentUpdate
render
getSnapshotBeforeUpdate
React updates DOM and refs
componentDidMount
componentDidUpdate
componentWillUnmount
"Render Phase"
Pure and has no side effects. May be paused, aborted or restarted by React.
"Pre-Commit Phase"
Can read the DOM.
"Commit Phase"
Can work with DOM, run side effects, schedule updates.



"Render Phase"

Pure and has no side effects. May be paused, aborted or restarted by React.

"Pre-Commit Phase"

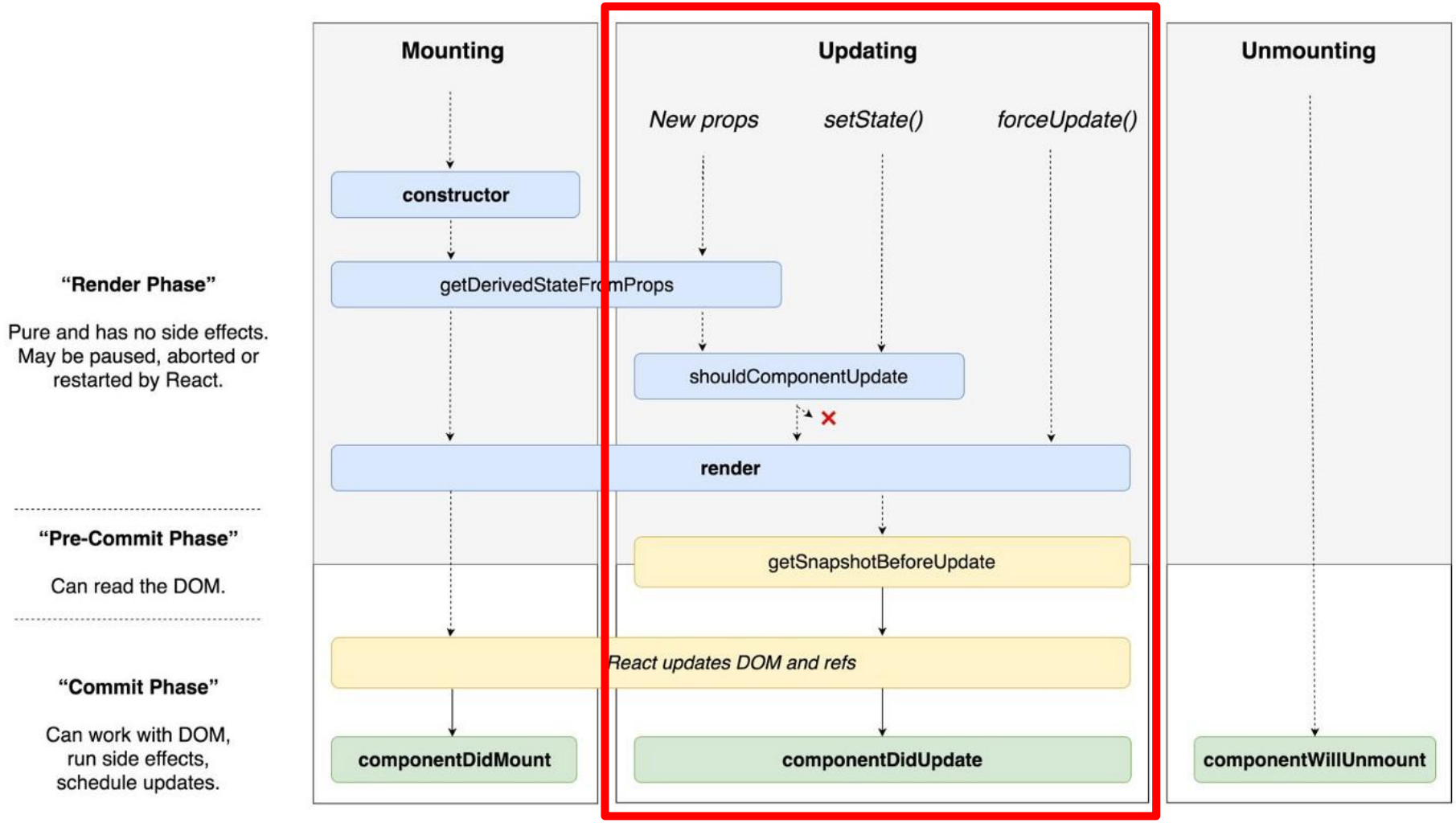
Can read the DOM.

"Commit Phase"

Can work with DOM, run side effects, schedule updates.

6.2. Methods of Mounting Phase

1. constructor()
   - An instance of the component class is created
2. static getDerivedStateFromProps()
   - Called on every render() – the first render happens in the mounting phase
3. render()
   - Converts VDOM into DOM
4. componentDidMount()
   - After the render method is called

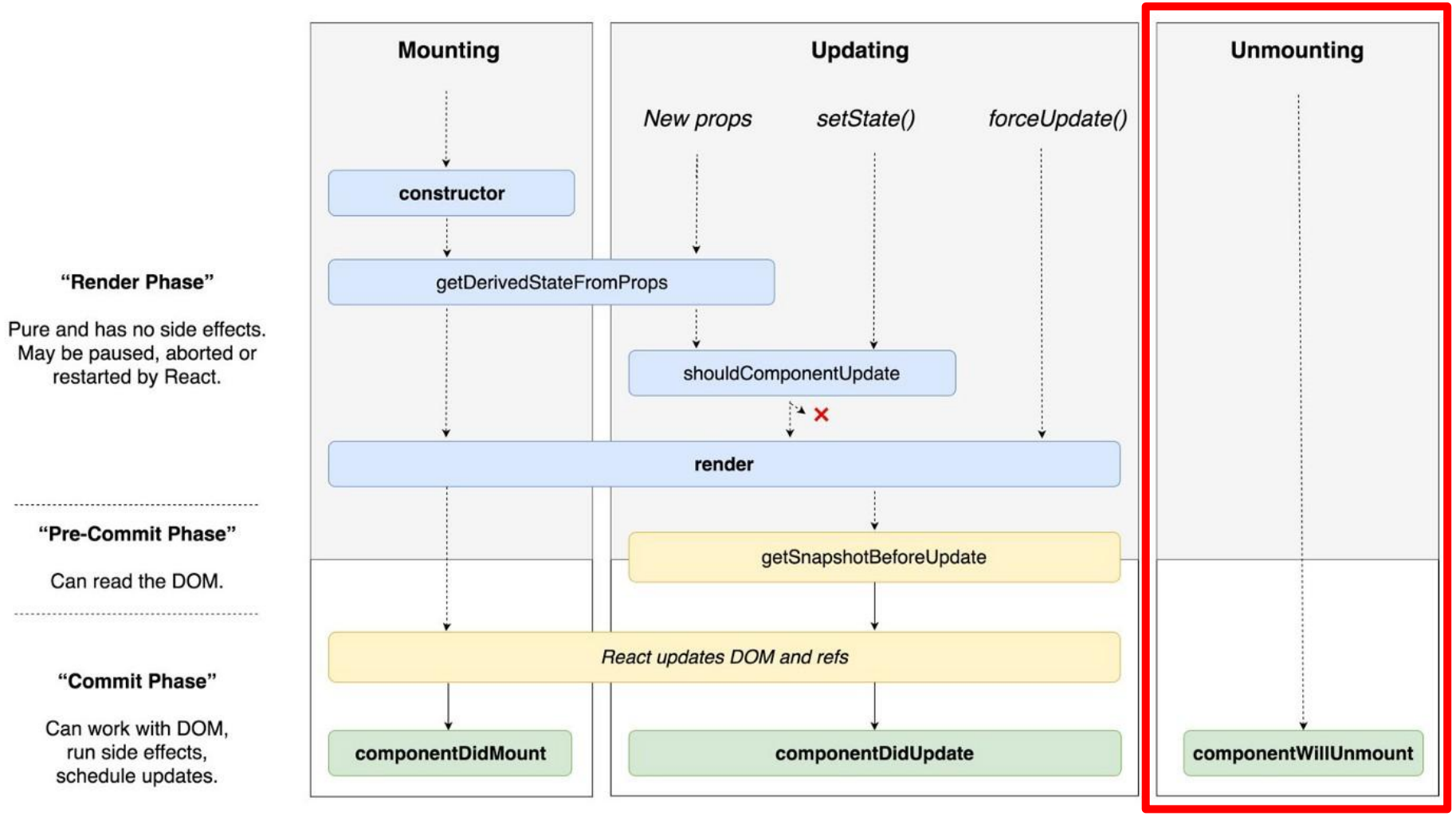

Mounting
Updating
Unmounting
New props
setState()
forceUpdate()
constructor
getDerivedStateFromProps
shouldComponentUpdate
render
getSnapshotBeforeUpdate
React updates DOM and refs
componentDidMount
componentDidUpdate
componentWillUnmount
"Render Phase"
Pure and has no side effects. May be paused, aborted or restarted by React.
"Pre-Commit Phase"
Can read the DOM.
"Commit Phase"
Can work with DOM, run side effects, schedule updates.

6.3. Methods of Updating Phase

1. static getDerivedStateFromProps()
2. shouldComponentUpdate()
3. getSnapshotBeforeUpdate()
4. render()
5. componentDidUpdate()

6.4. Methods of Unmounting Phase

1. componentWillUnmount()


Mounting
Updating
Unmounting
New props
setState()
forceUpdate()
constructor
getDerivedStateFromProps
shouldComponentUpdate
render
getSnapshotBeforeUpdate
React updates DOM and refs
componentDidMount
componentDidUpdate
componentWillUnmount
"Render Phase"
Pure and has no side effects. May be paused, aborted or restarted by React.
"Pre-Commit Phase"
Can read the DOM.
"Commit Phase"
Can work with DOM, run side effects, schedule updates.

6.5. Task

1. Create a new react-app task-6-5
2. In the App component‘s state, create an array randomStringLengths consisting of three random numbers. Each number can either be 4, 5 or 6.
3. Add a button labeled as „Generate New Random String Lengths“
4. When the button of 3) is clicked, new values for randomStringLengths will be generated.
5. Create a new stateful subcomponent „RandomStringGenerator“ that only has one div.
6. The state of RandomStringGenerator solely consists of one variable randomString.
7. Inside RandomStringGenerator‘s div, the current value of this.state.randomString must be shown.
8. In RandomStringGenerator, create the method generateRandomString(n) that returns a random string of length n.
9. Add a new prop “stringLength“ for the RandomStringGenerator component.
10. Create three instances of RandomStringGenerator inside the App component, based on the randomStringLengths array. For each instance, pass down the number as „stringLength“-prop.
11. Implement the following behaviour with your knowledge of component lifecycles:

    a) Whenever RandomStringGenerator receives a new value for stringLength, it generates a new value of this.state.randomString.

    b) Whenever it receives the same value again, RandomStringGenerator does not render.

Hint: Using the Chrome-Debugger might be helpful.

Agenda – Part 7
Routing

1. Definition of a Route
2. Task

7.1. Definition of a Route

- **<u>A route is an address of a resource</u>**
- Resources are exposed by webservers to the Internet
  - https://www.google.com -> Resource / of google.com
  - https://www.linkbox.io/jan -> Resource /jan of Linkbox.io
  - https://www.linkbox.io/jan/music -> Resource /jan/music of linkbox.io
- **<u>In React, a resource is the address of a component</u>**
  - /home -> Renders component Home in App
  - /home -> Renders can also render component Imprint in App
  - / -> Renders App component

7.2. Task

1. Create a new react app task-7-2
2. Create a subcomponent Start that solely contains one div which says „This is the Startpage“.
3. Create another component Users.
4. Open the link: http://jsonplaceholder.typicode.com/users and copy everything into the clipboard. Afterwards, paste it into your Users and assign it as new constant. I.e.

   const users = [{ “id“: 1, “name“: “Leanne Graham“, ...]
5. Create another component UserDetail.
6. Do exactly the same as in 4) for the UserDetail Component.
7. In the App Component, create a Router and a navigation that links the following Routing to the respective Components

   / -> Start
   /users -> Users
   /userdetail/:id -> UserDetail

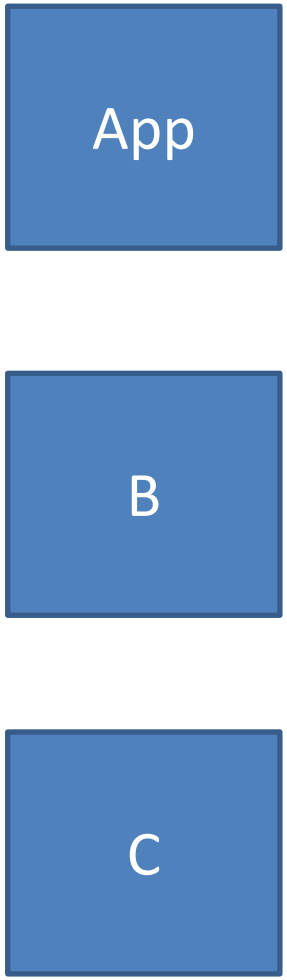
   Please note, that the last route expects an parameter.
8. In the Users component, show the name, the email and the phone of each user in a tabe.
9. In the table of 8), add another column with a Link “Details“ that links the UserDetail component with the id of the respective user.
10. In the component UserDetail, show only one div that consists of the name of the user with id passed as parameter.

Agenda – Part 8
Communication between three or more layers of Components

1. Props Drilling
2. Task
3. Context API
4. Task
5. Redux
6. Task

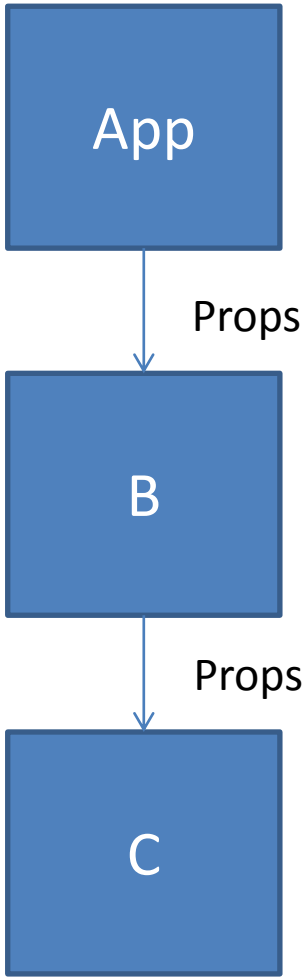
8.1. Props Drilling

<u>Props Drilling</u> =



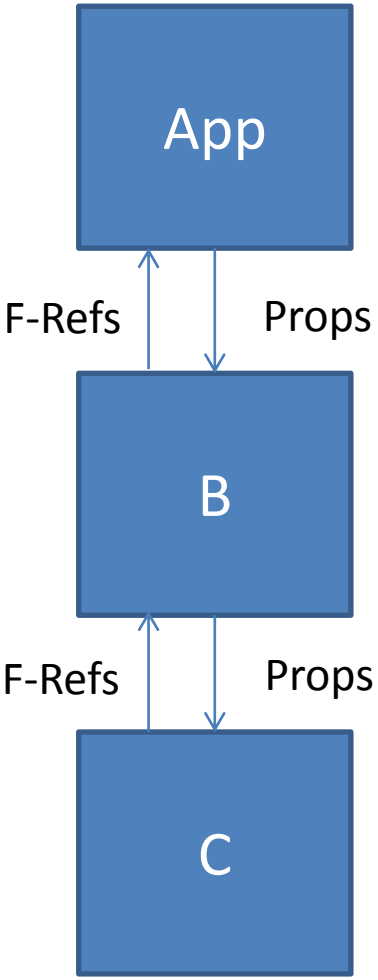
8.1. Props Drilling

<u>Props Drilling</u> = Passing props down child by child



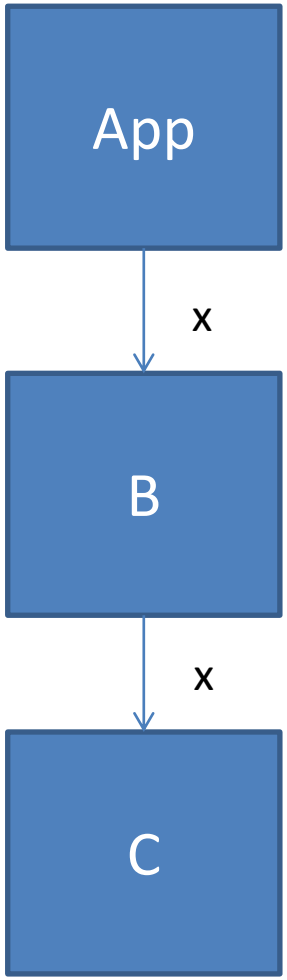
8.1. Props Drilling

<u>Props Drilling</u> = Passing props down child by child
Passing function references up parent by parent



8.1. Props Drilling

<u>Props Drilling</u> = Passing props down child by child
Passing function references up parent by parent



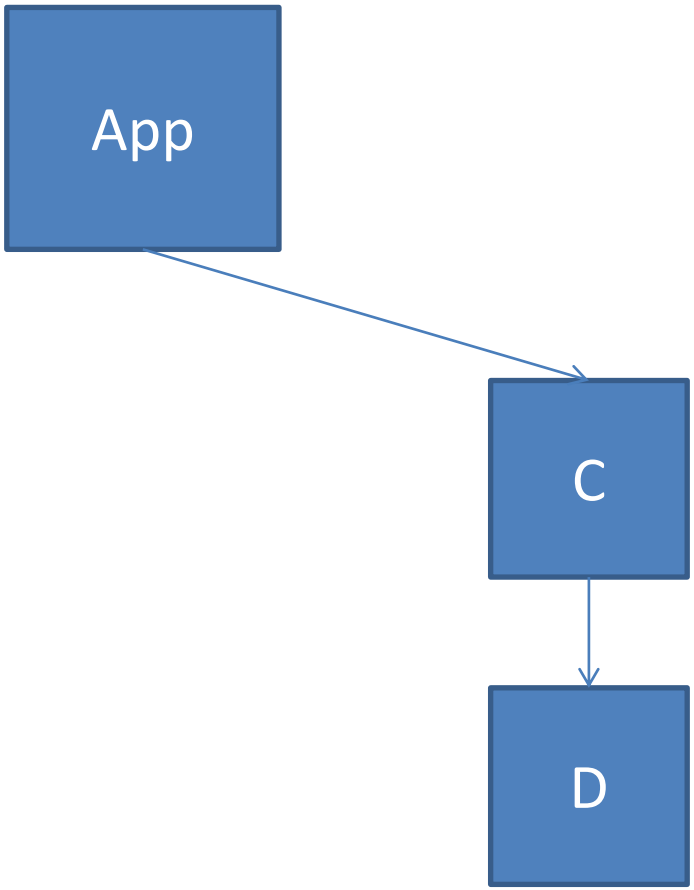
8.2. Task

1. Create a new react-app task-8-2
2. Inside the App component, create a header that says “App Component“ and underneath a button with the value „Generate Random Number“.
3. In App‘s state, create a new variable “isGreaterThan100“ which is false by default and new method “greaterThan100“ with one parameter isIt. greaterThan100
4. When the button of 2) is clicked, a new random number between 0 and 9 must be generated and saved as randomNumber inside App‘s state. Show the current value of randomNumber underneath the button of 2).
5. Create a stateful component C and instantiate it in the App component with one prop “randomNumber“, which should be equal to randomNumber. C has a header that only says “Component C“.
6. When C receives a new prop randomNumber from App, C generates its own random number “randomNumberOfC“ which is between 0 and 9. Afterwards, randomNumberOfC shall be multiplied with the received prop randomNumber. In C, the product shall be saved as “product“.
7. Create a stateful component D and instantiate it in component C with one prop “randomNumber“. Pass C‘s “product“ down to D as “randomNumber“.
8. When D receives a new prop randomNumber from C, D generates its own random number “randomNumberOfD“ which is between 0 and 9. Afterwards, randomNumberOfD shall be multiplied with the received prop randomNumber. In D, the product shall be saved as “product“.
9. Now, if the D‘s product is greater than 100, D should call the App‘s method greaterThan100(true). Find a way to implement that.
10. When App

8.2. Task

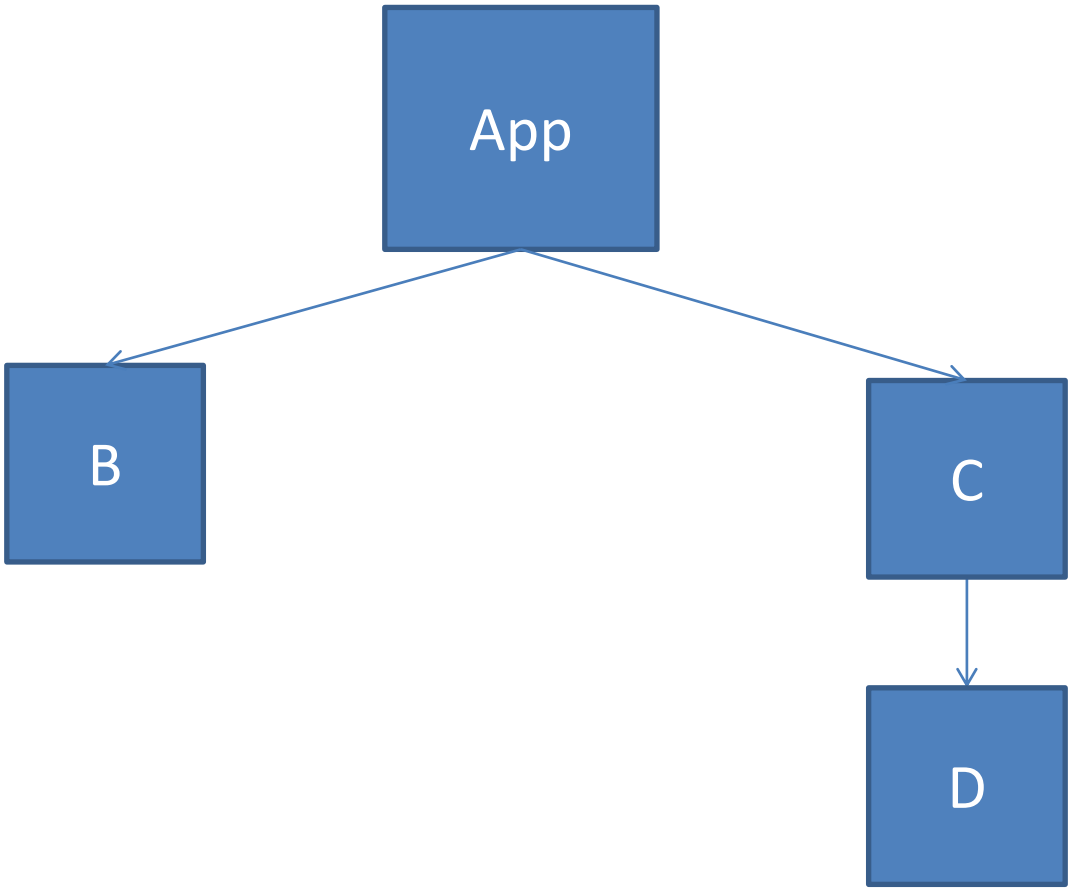
1. Create a new react-app task-8-2
2. Inside the App component, create a header that says “App Component“ and underneath a button with the value „Generate Random Number“.
3. When the button of 2) is clicked, a new random number between 0 and 9 shall be generated and saved in App‘s state as randomNumberOfApp.

8.2. Task



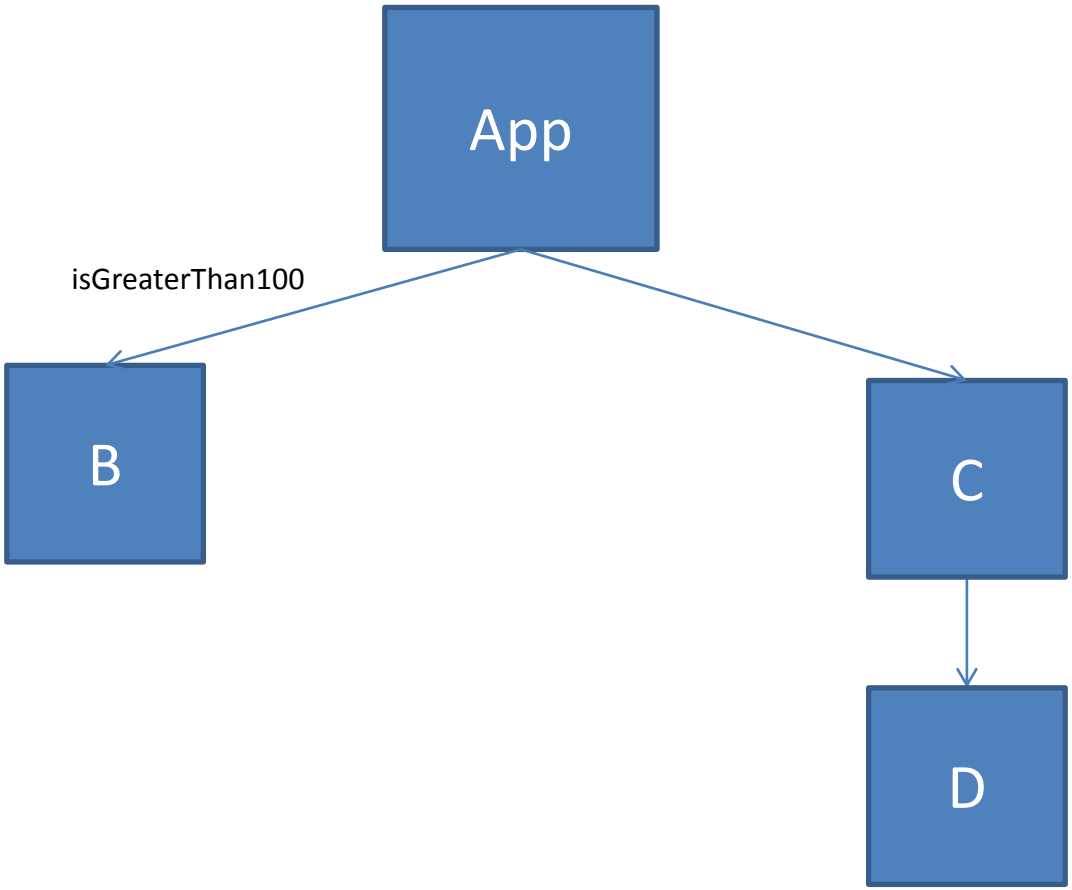
4. Create two stateful Components C and D.
5. Instantiate C inside of App
6. Instantiate D inside of C.

8.2. Task



6. Create a stateless component B and instantiate it in the App component. In B, create a header saying “Component B“.

8.2. Task



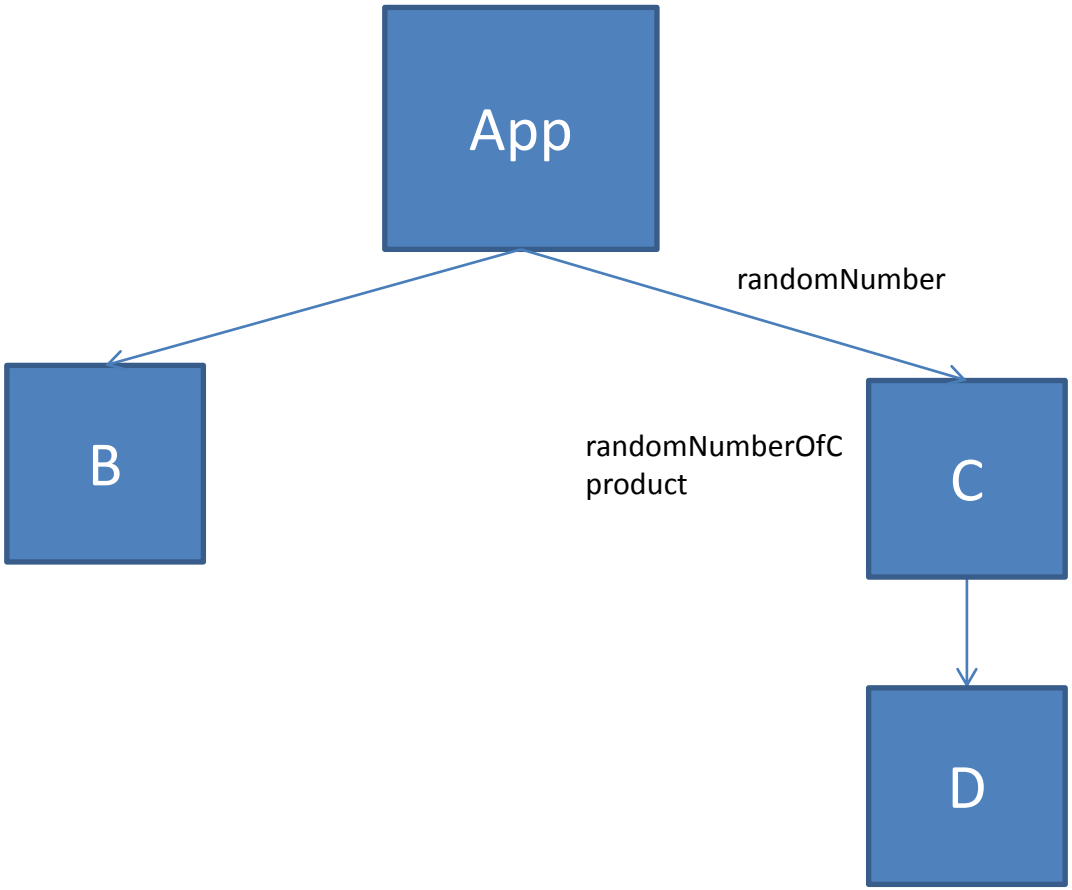
7. Inside App, create the method “greaterThan100“ with one boolean parameter isIt.

If isIt is true, B (<u>not App</u>) shall show the text *“The product of the three random numbers is greater than 100“*

If isIt is false, B shall show the text *“The product of the three random numbers is less or equal than 100.“*

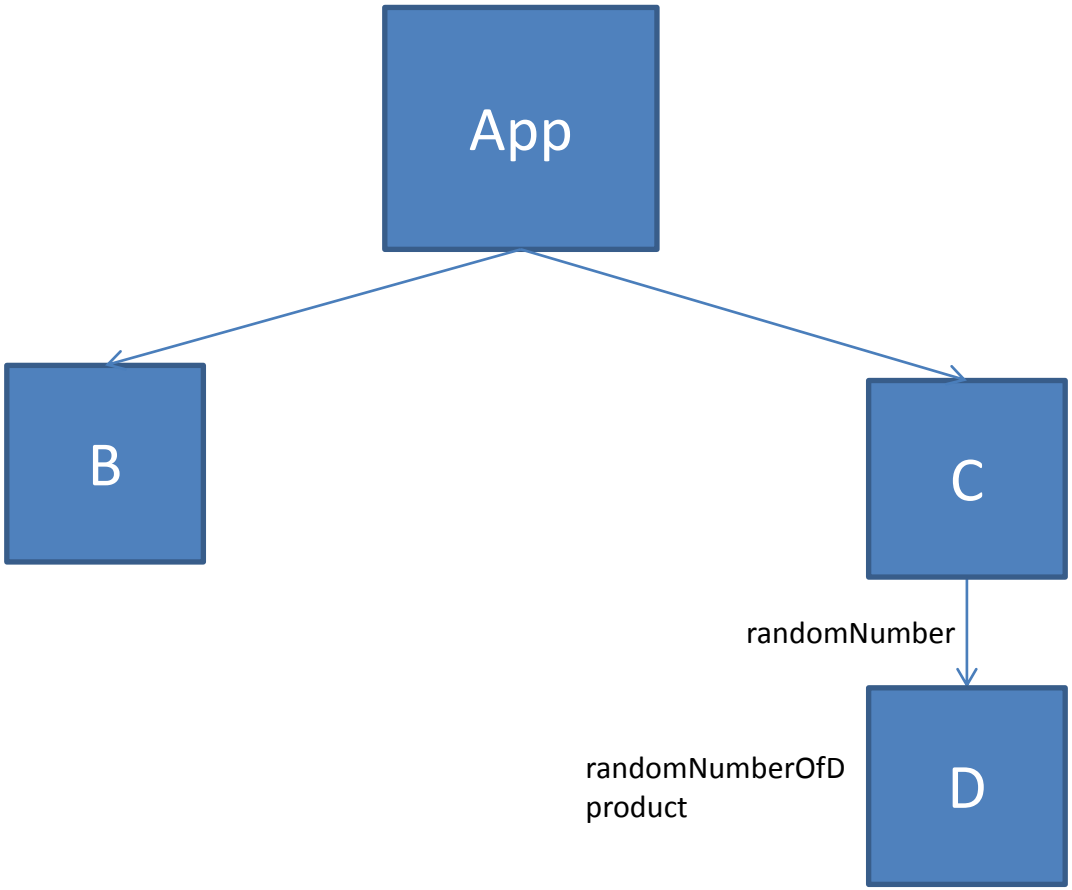
-> use a prop “isGreaterThan100“ for this

8.2. Task



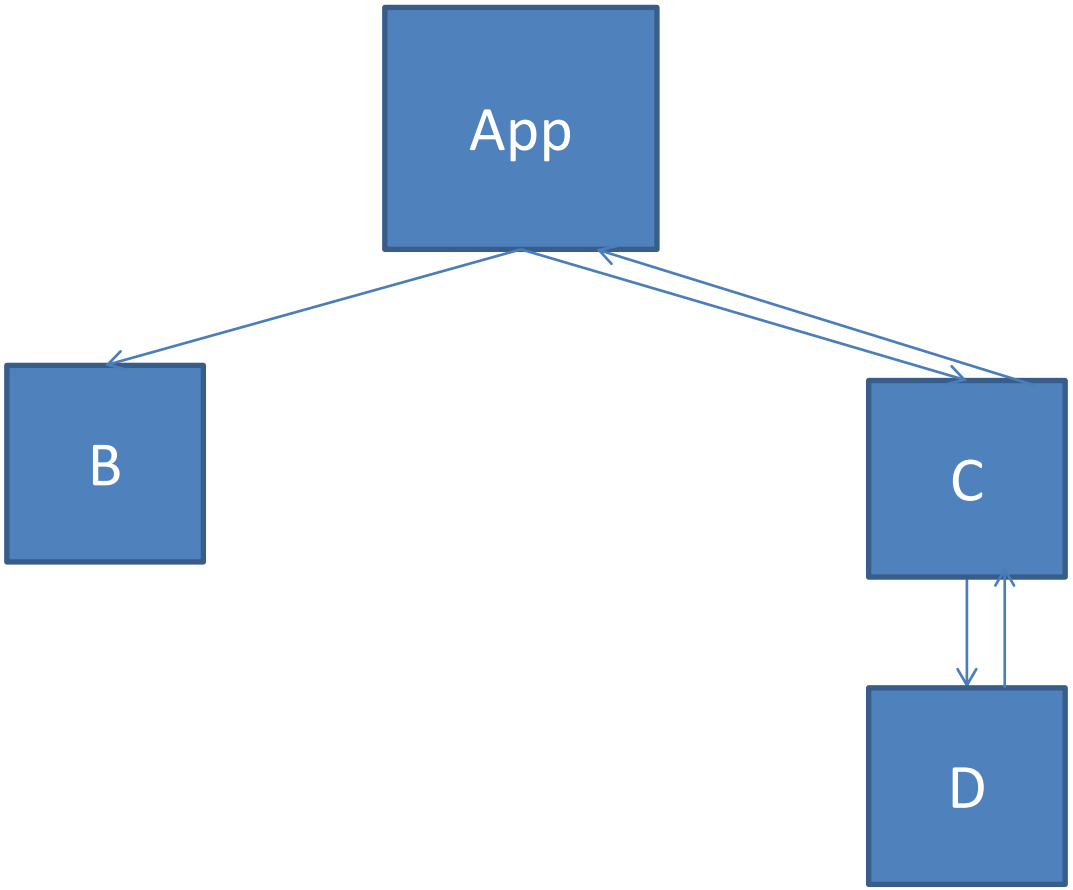
8. Pass App‘s randomNumberOfApp down to C as prop “randomNumber“
9. After C has received App‘s randomNumber, C generates an Internal random number between 0 and 9 and saves it as randomNumberOfC. Then, C calculcates the product of randomNumberOfC and the randomNumber received by App and saves that internally as “product“.

8.2. Task



10. Pass C's randomNumberOfC down to D as prop "randomNumber"
11. After D has received C's randomNumber, D generates an Internal random number between 0 and 9 and saves it as randomNumberOfD. Then, D calculcates the product of randomNumberOfD and the randomNumber received by C and saves that internally as "product".

8.2. Task

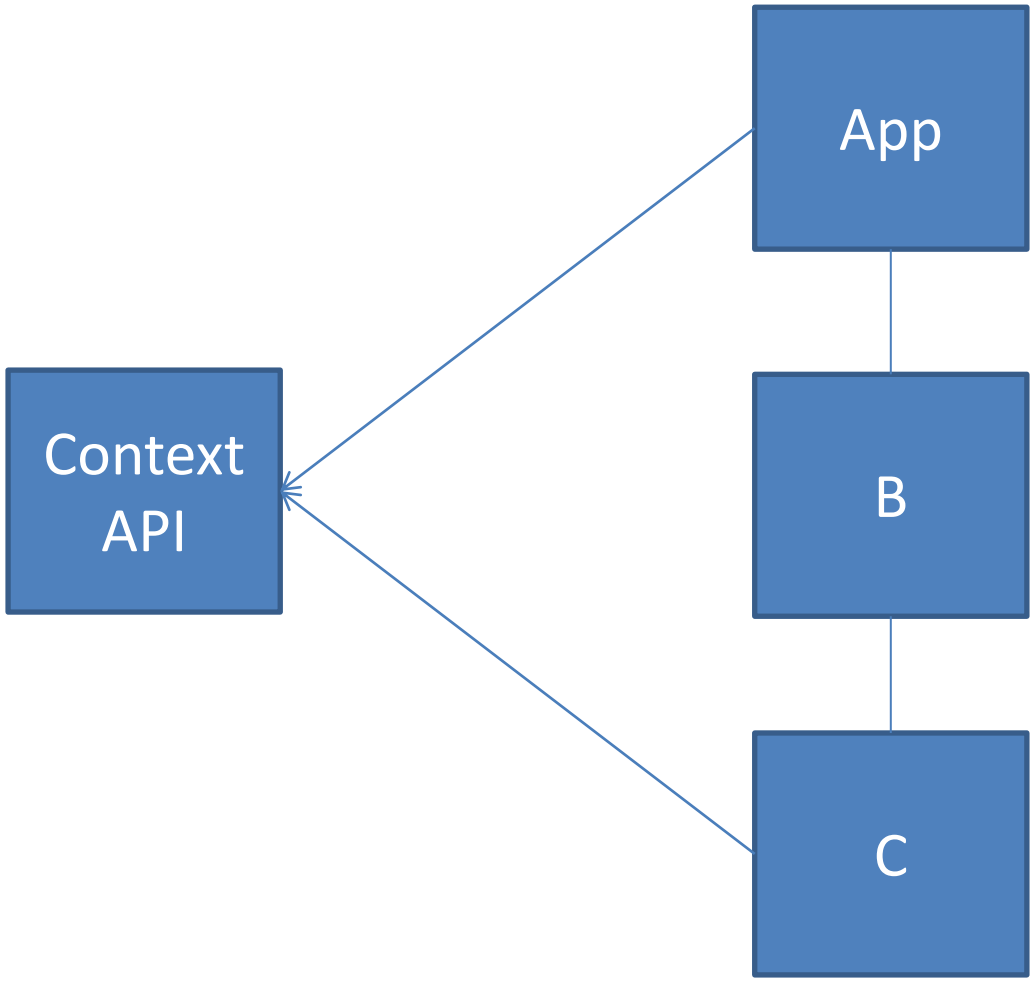


12. If D's product is greater than 100, D calls App's method greaterThan100(true). If D's product is smaller or equal 100, D calls greaterThan100(false). Therefore, implement an upward communication from D to App.

13. Make sure that the render methods of C and D are not called too many times by React.

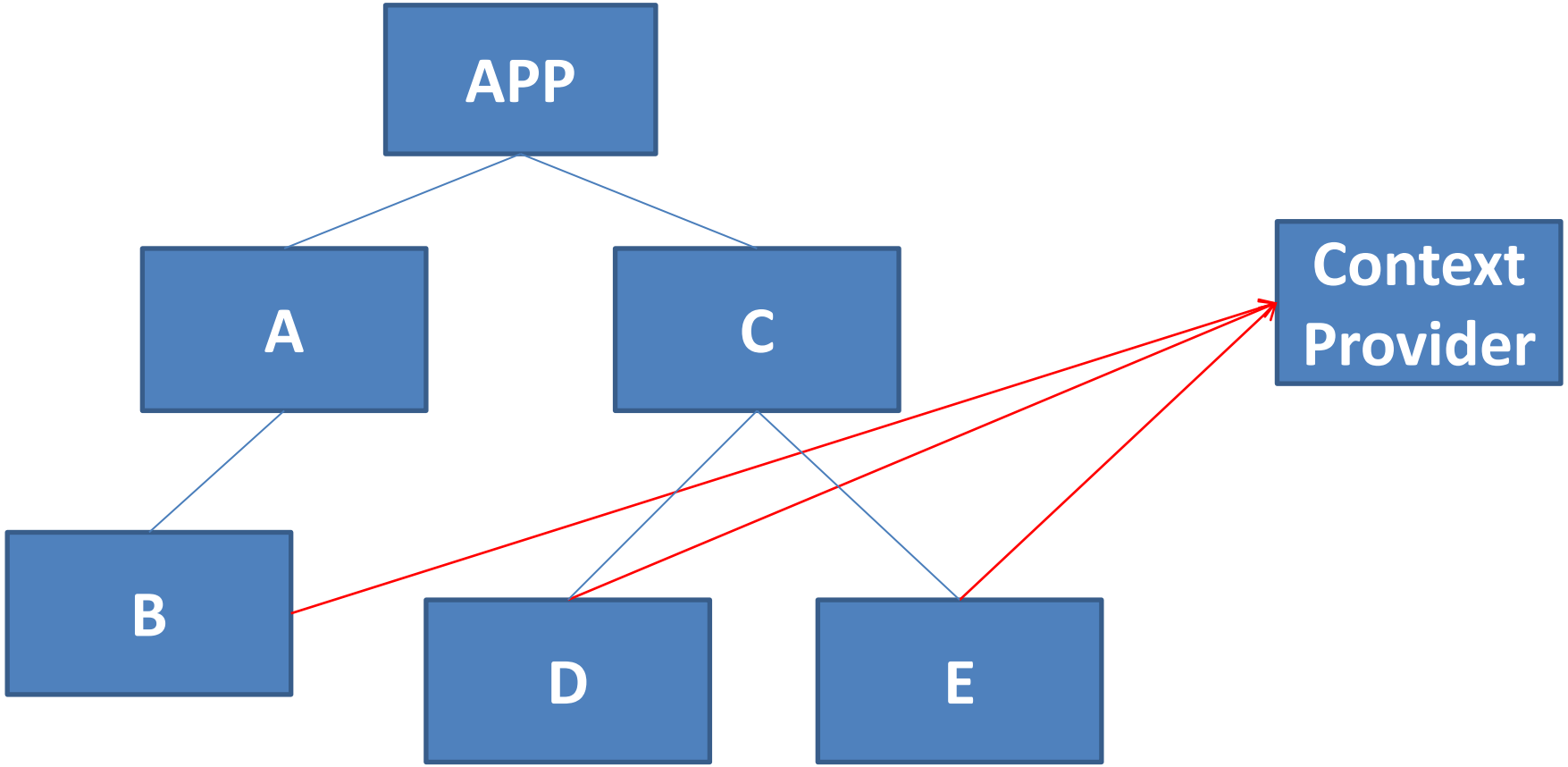
8.3. Context API

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



8.4. Task

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



8.4. 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

8.5. Reducer

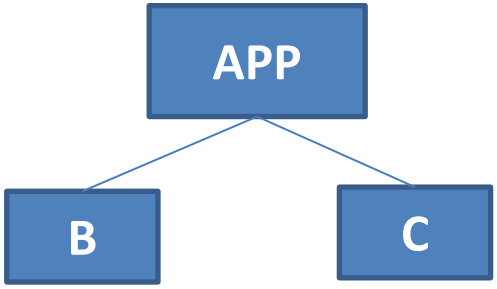
Context API functions can be indirectly called via a **Reducer** which maps messages to function calls.

„INCREMENT_X“ -> incrementX()

„DECREMENT_Y“ -> decremenY()

8.6. Redux

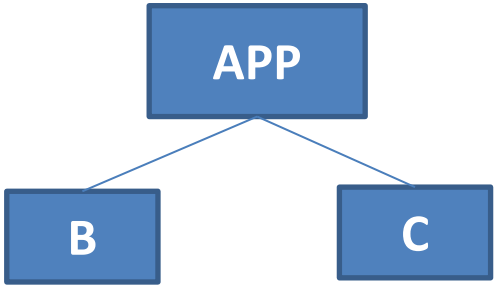
3. Redux = One or multiple shared states managed by a store

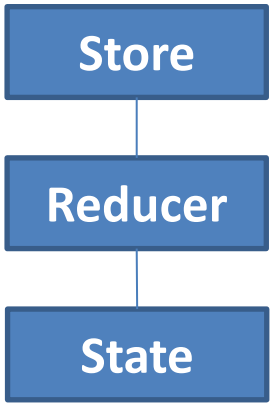


1. Create your App and it‘s child components

10. Communication between multi-layered components

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



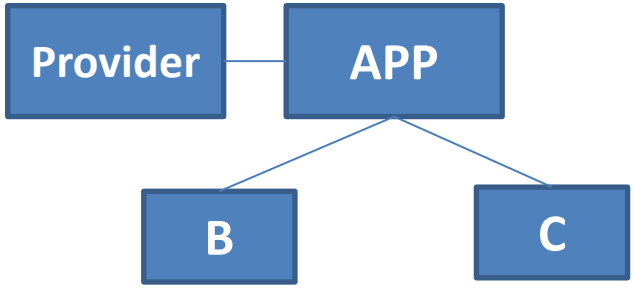


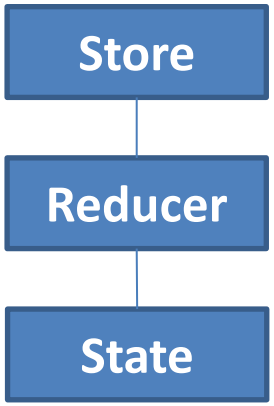
x=0, y=0

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



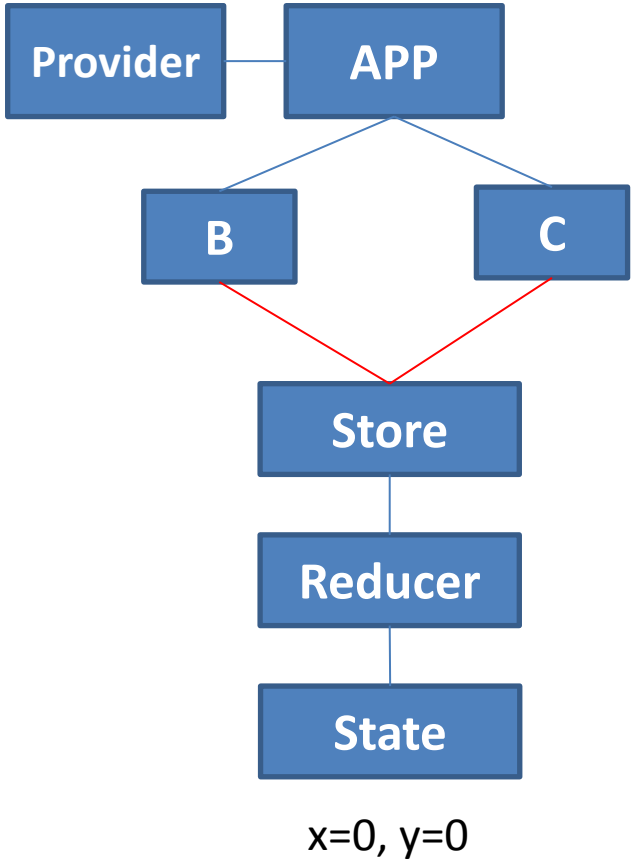


x=0, y=0

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

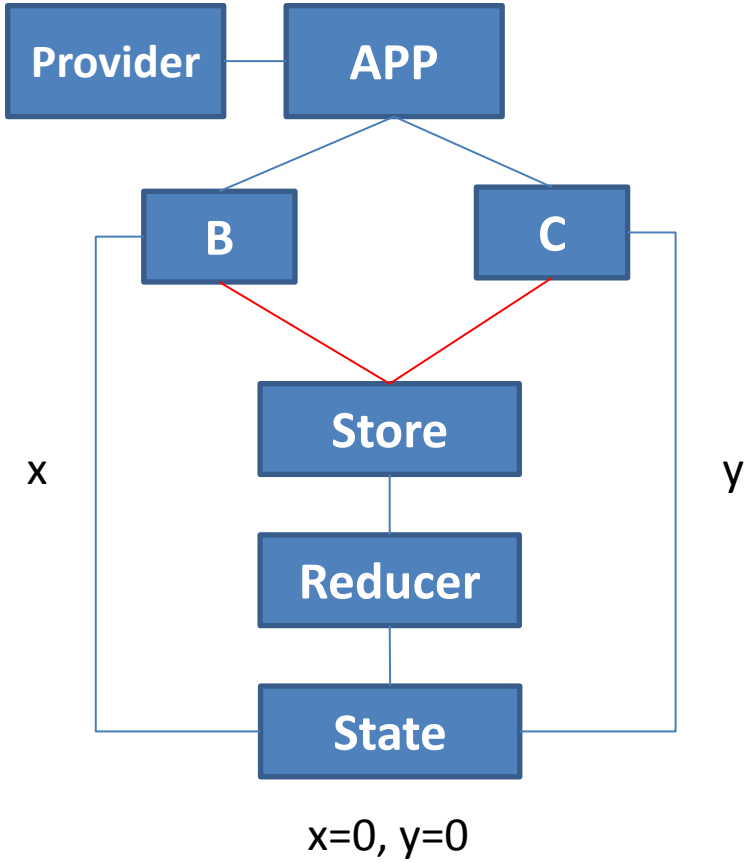
3. Redux = One or multiple shared states managed by a store



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

8.6. Redux

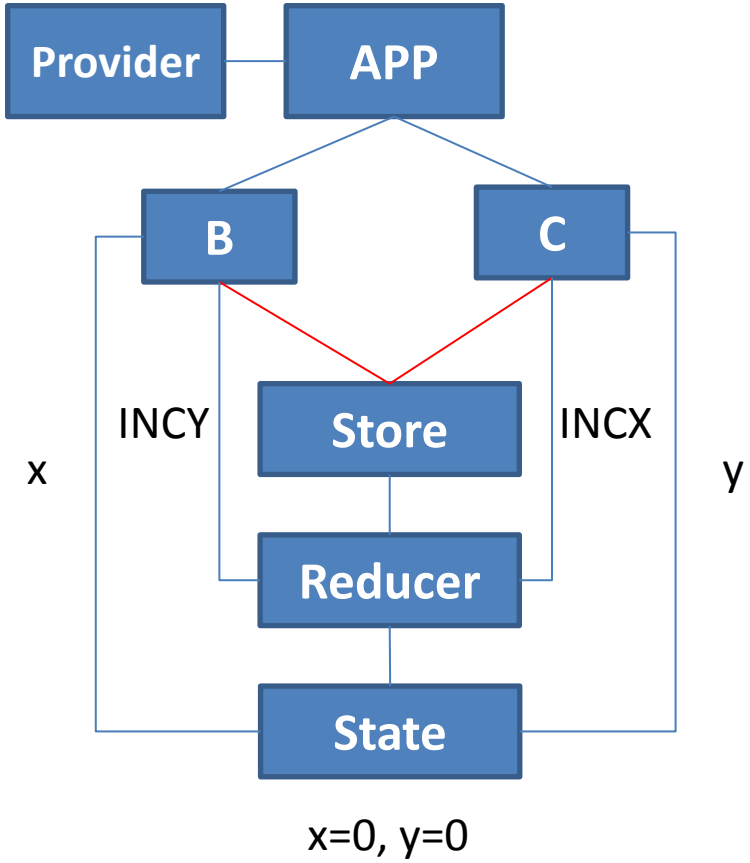
3. Redux = One or multiple shared states managed by a store



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

8.6. Redux

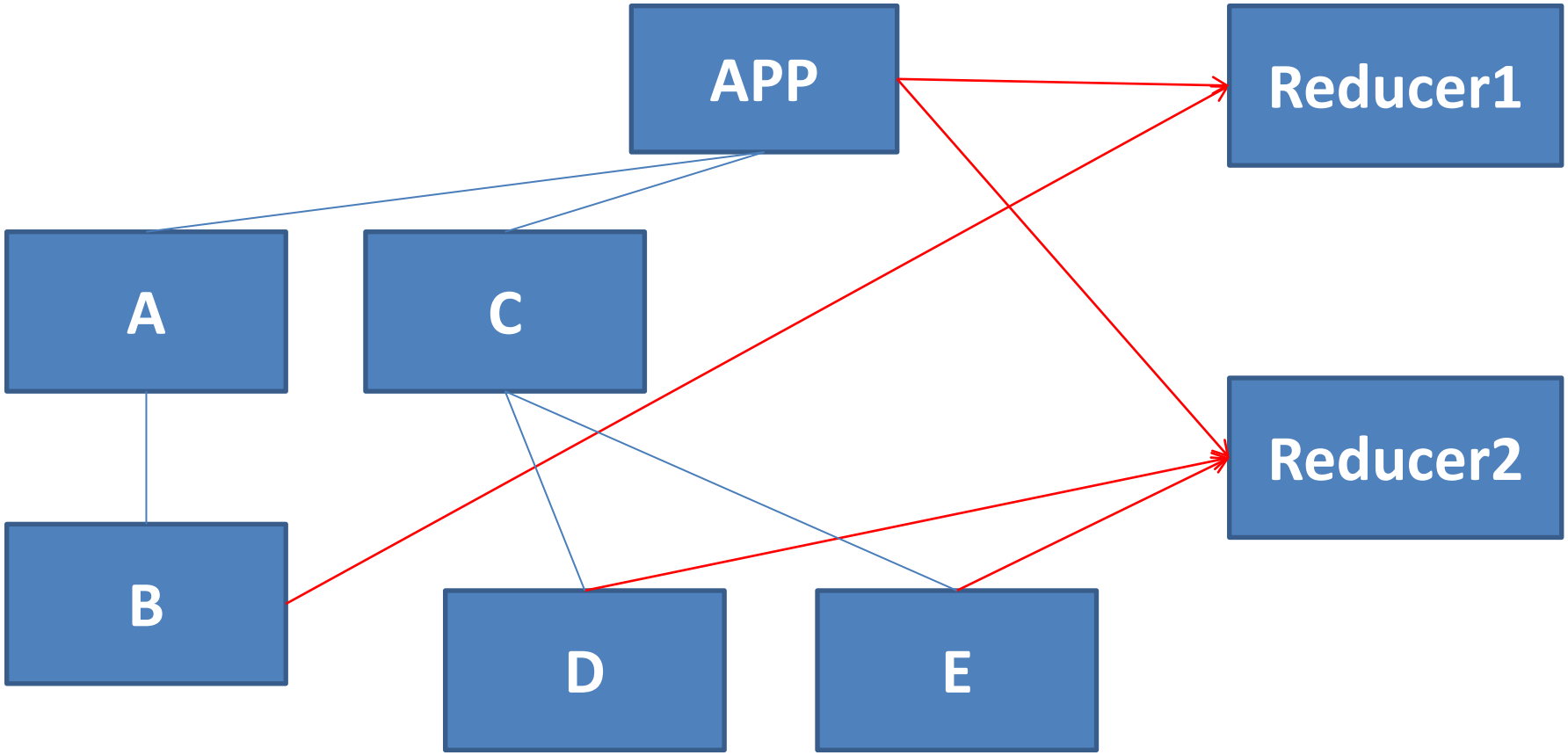
3. Redux = One or multiple shared states managed by a store



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 Reducer to props of B and C

8.7. Task

1 - Implement the following component architecture using Redux.



8.7.

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.

10. Communication between multi-layered components

- Redux Thunk
  - The dispatchers waits for a AXIOS call to finish, then dispatches the messages to the reducer

Agenda – Part 9
AJAX

1. AXIOS
2. Task
3. Task (Difficult)
4. Redux Thunk
5. Localhost as Backend

9.1. 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

9.2. Task

1. Create a new React App task-9-2
2. This app shall load all user data from the following URL:

https://jsonplaceholder.typicode.com/users

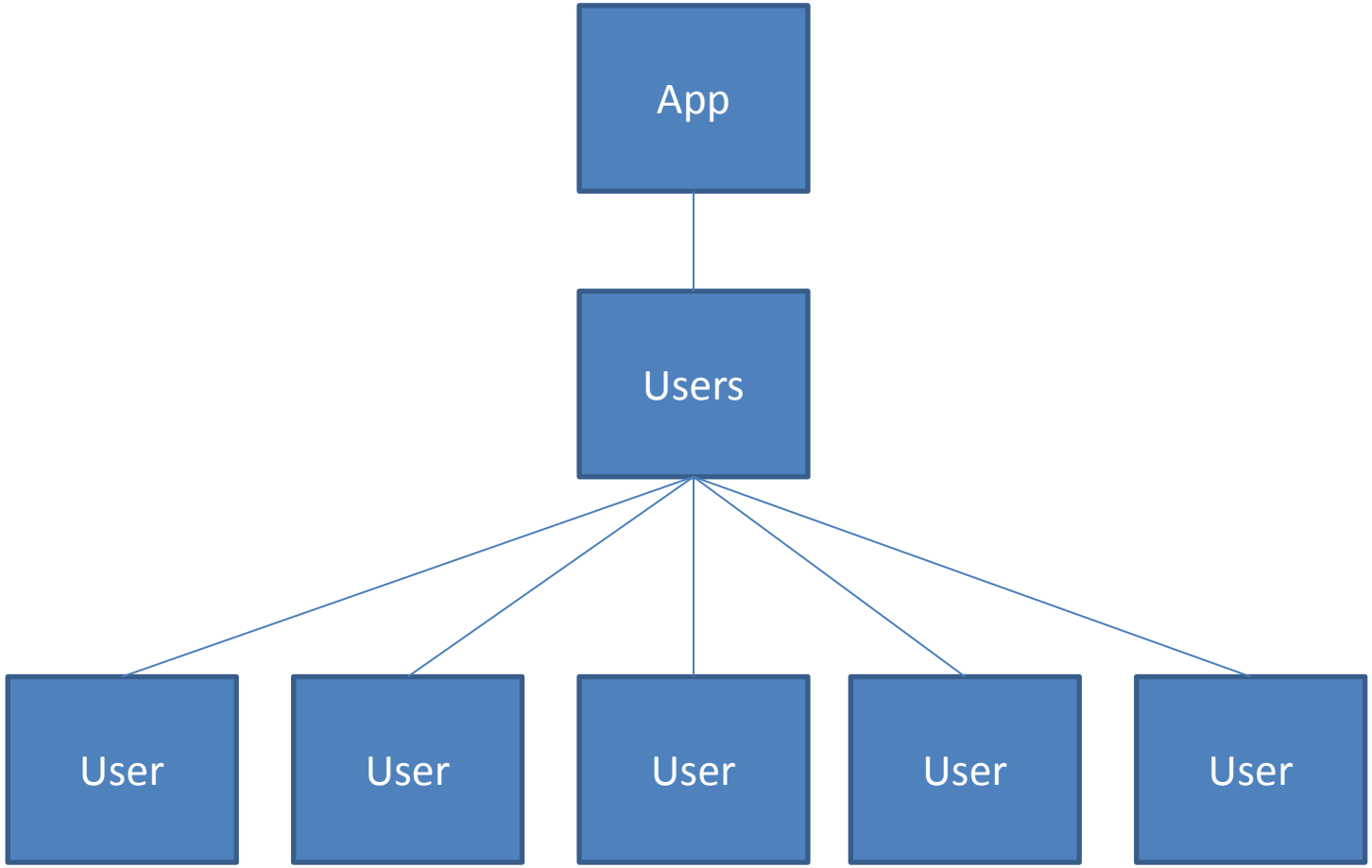
The app should shows the id, the name and the email in a table. By clicking on the X next to the user, the user will be removed from the internal state.

9.2. Task

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

9.2. Task



9.3. Task (Difficult)

1. Create a React App task-9-3
2. Inside the App component, create a button labeled “Next User“.
3. Create another component User and instantiate it in the App component underneath the “Next User“ button.

9.3. Task (Difficult)

4. Implement the following behaviour:

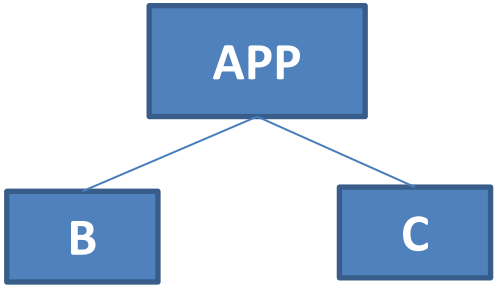
- When the App component initializes
  1) the App component sends the User component the id of the first user (id = 1) via props.
  2) The User component reads the id and loads the name and email via Axios from https://jsonplaceholder.typicode.com/users/1
  3) The User component shows the name and email of the first user (id=1).

9.3. Task (Difficult)

5. Furthermore, implement the following behaviour:
   1) When the App‘s “Next User“ button is clicked, 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/{id}
   3) The User component shows the name and email of the next user
   4) Afther the 10th user, the id shall start with 1 again.

9.4. Redux-Thunk

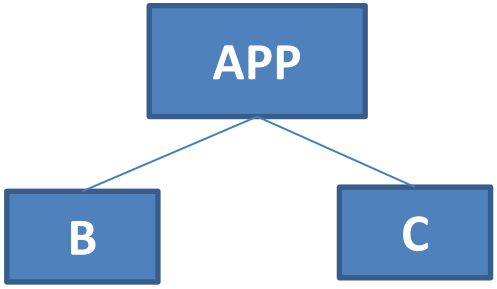
3. Redux = One or multiple shared states managed by a store

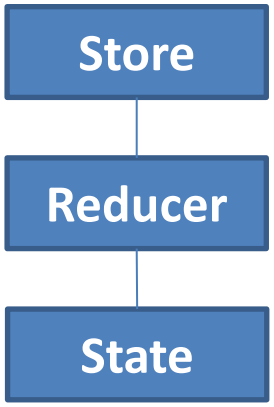


1. Create your App and it‘s child components

9.4. Redux-Thunk

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



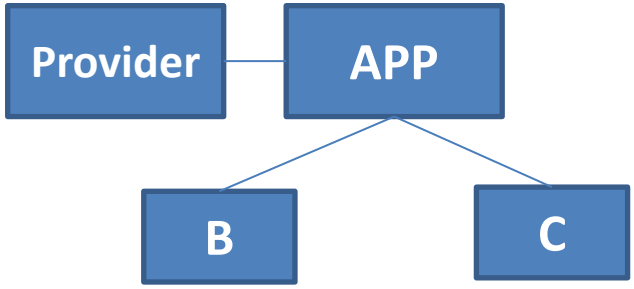


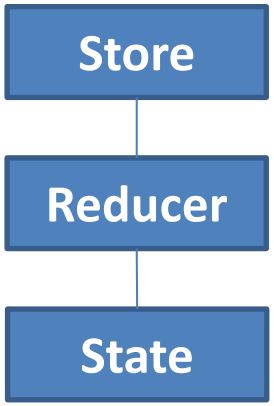
users=null

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

9.4. Redux-Thunk

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



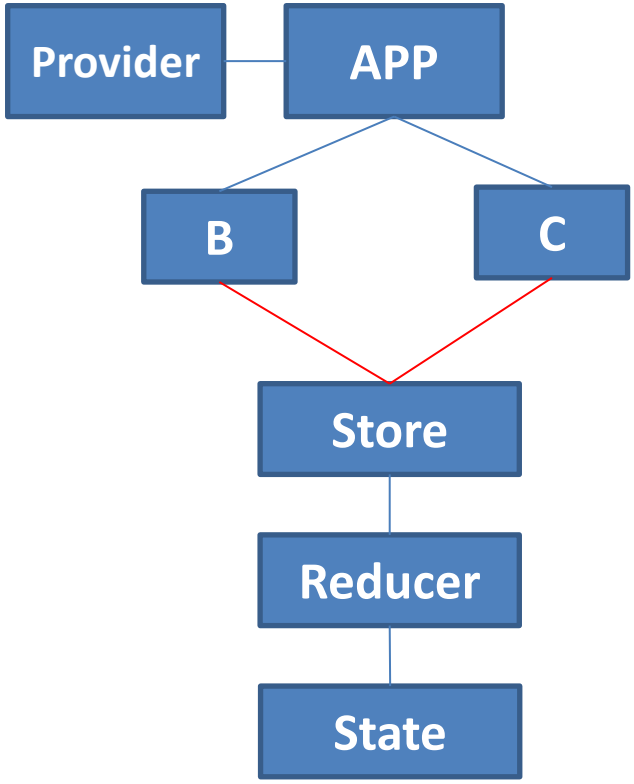


users=null

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

9.4. Redux-Thunk

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

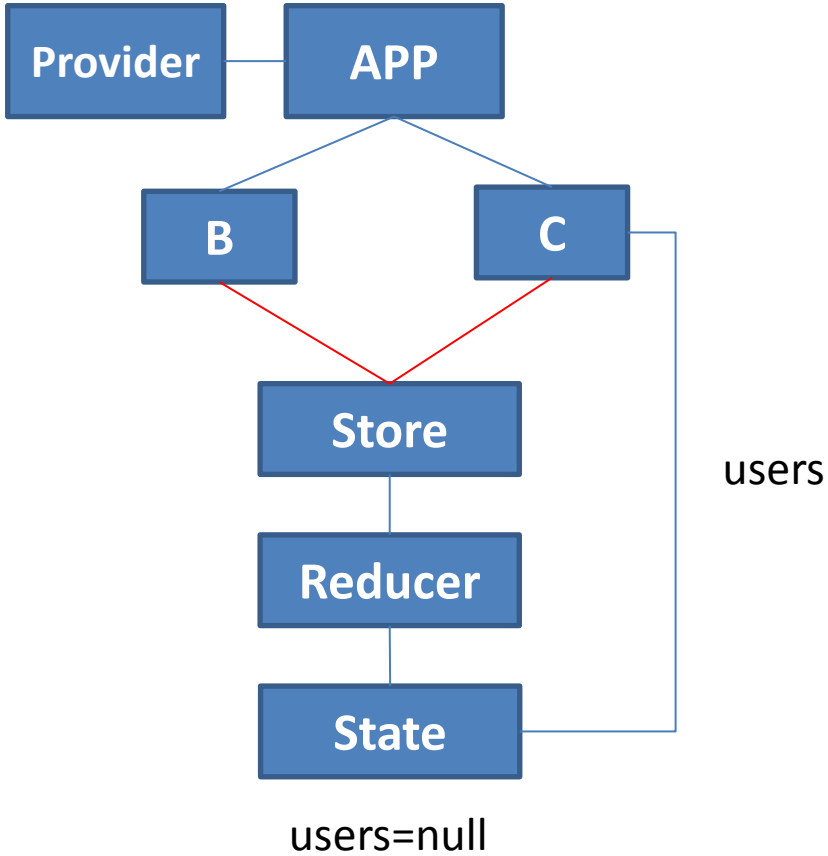


users=null

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

9.4. Redux-Thunk

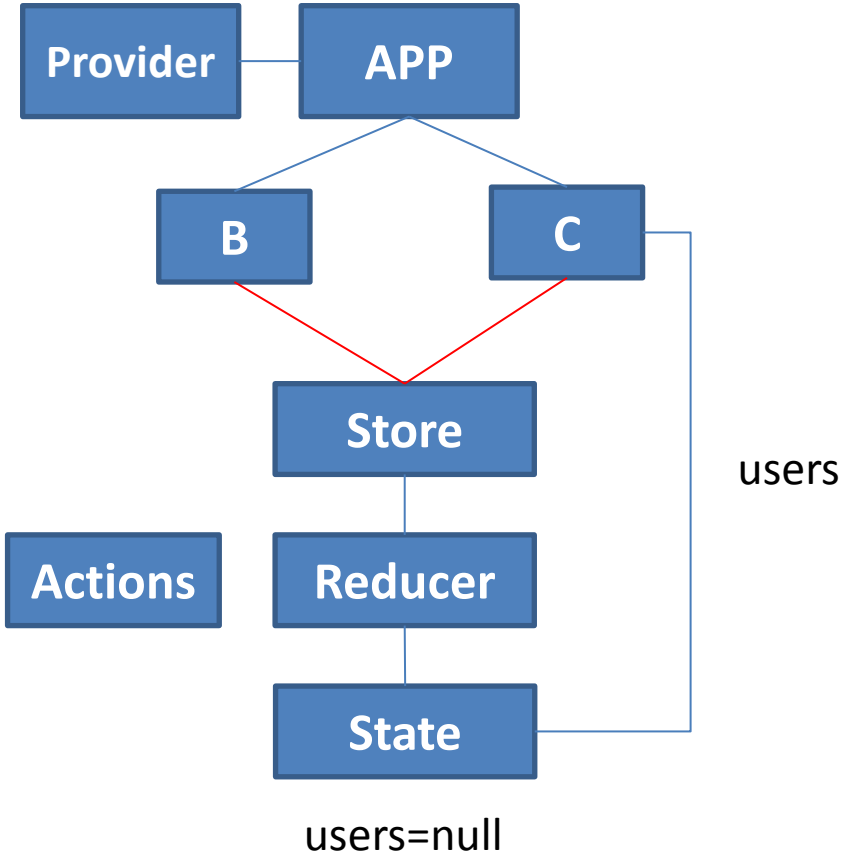
3. Redux = One or multiple shared states managed by a store



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 of C

9.4. Redux-Thunk

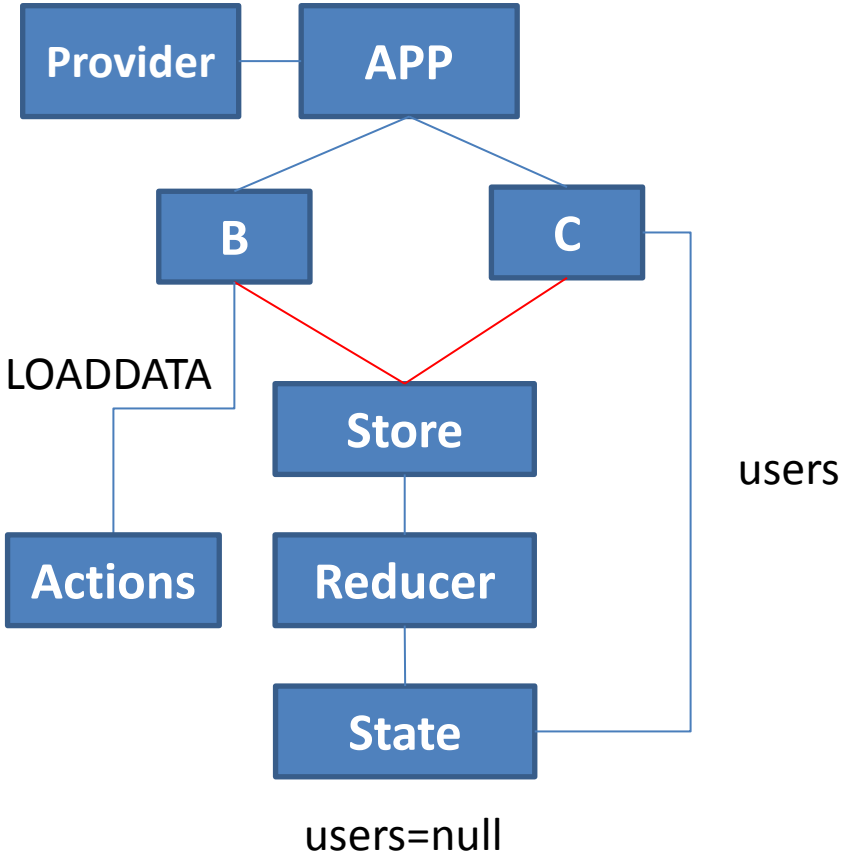
3. Redux = One or multiple shared states managed by a store



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 of C
6. Create an asynchronous function that calls dispatch after it is done.

9.4. Redux-Thunk

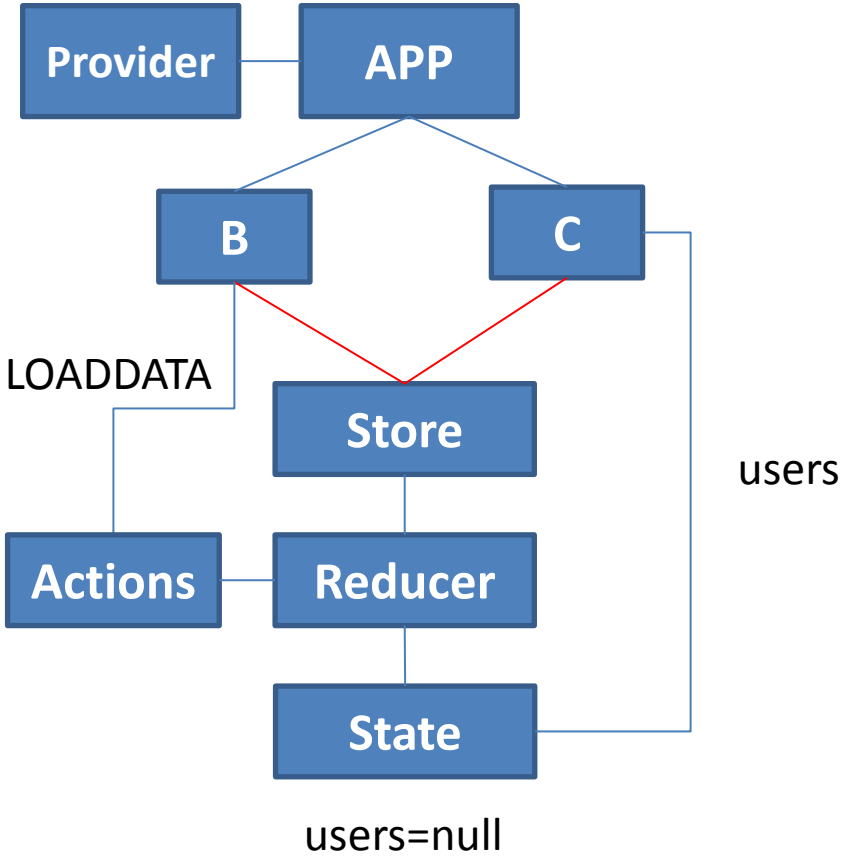
3. Redux = One or multiple shared states managed by a store



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 of C
6. Create an asynchronous function that calls dispatch after it is done.
7. Import the 6.) into B

9.4. Redux-Thunk

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 of C
6. Create an asynchronous function that calls dispatch after it is done.
7. Import the 6.) into B
8. Connect B and the asynchronous functions with the Reducer

9.5. Localhost as Backend

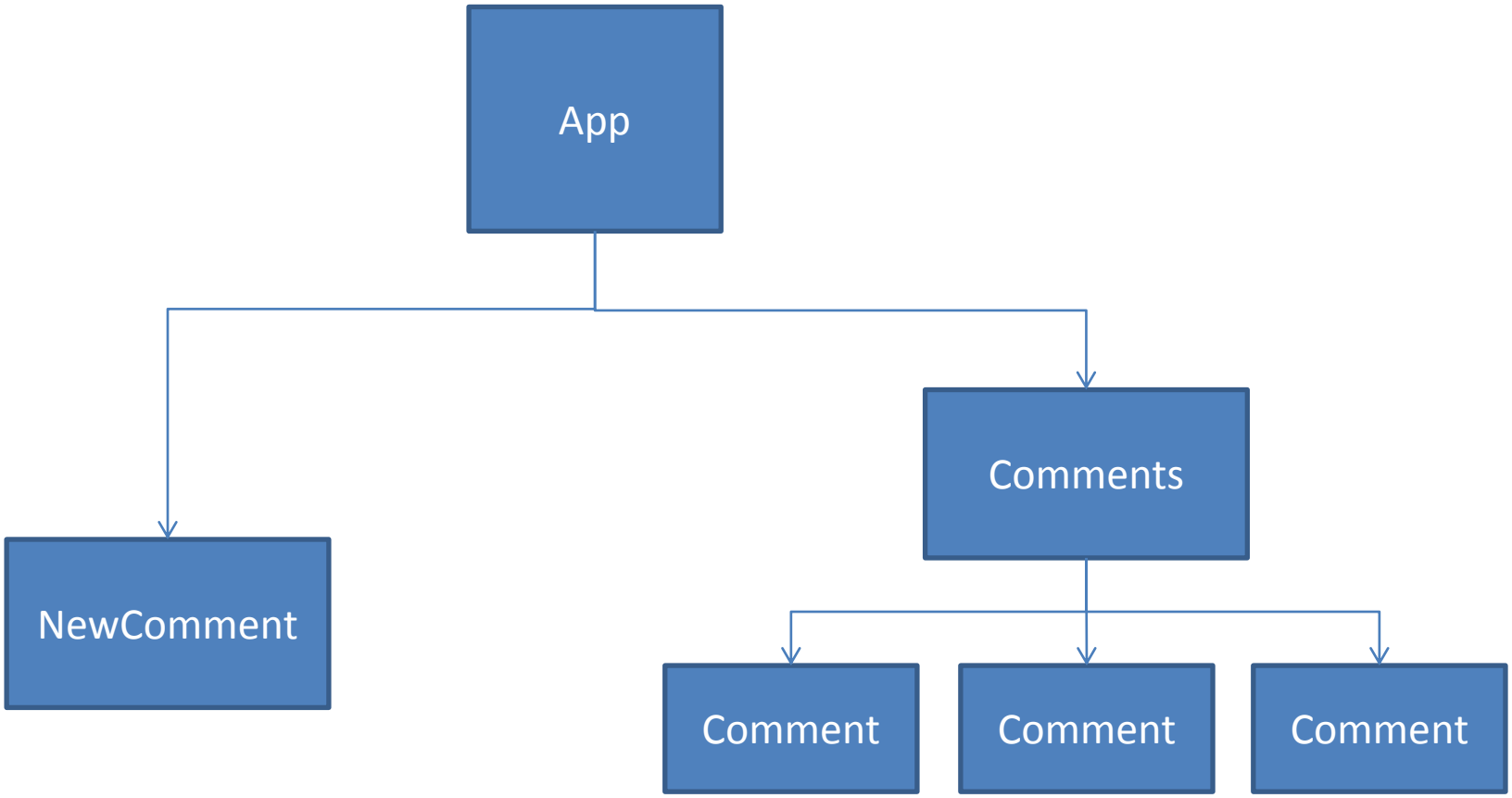
- “npm start“ starts a webserver that exposes the „/public“ – folder to HTTP clients
  - Chrome
  - Firefox
- The routes of that particular webserver are limited to
  - / - root
  - routes defined by our React Router
- The routes can be joined with the routes of a localhost backend server coded in
  - NodeJS
  - Java
  - PHP
  - ...
- Why?
  - Login
  - Signup
  - Protected Routes
  - ...

9.6. Task

1. Create a new react app task-9-6.
2. Create the two stateful components “NewComment“ and “Comments“ and instantiate them in the App component.
3. The NewComment component must contain of two input fields for name and text and a button labeled as „Create comment“.
4. Create a stateless component “Comment“. Each comment should be represented by its own Comment-component. Therefore, from the Comments-component, pass the props “name“ and “text“ down to each Comment-component. Create 3 instances of the Comment-component with the names “John“, “Bob“, “Mary“ and three texts “Hi whatsup“, “How are you?“ and “Good weather today!“
   - The Comment component must consist of three Divs for the name and the text. Inside the third Div, add a button with an “X“ inside.

9.6. Task

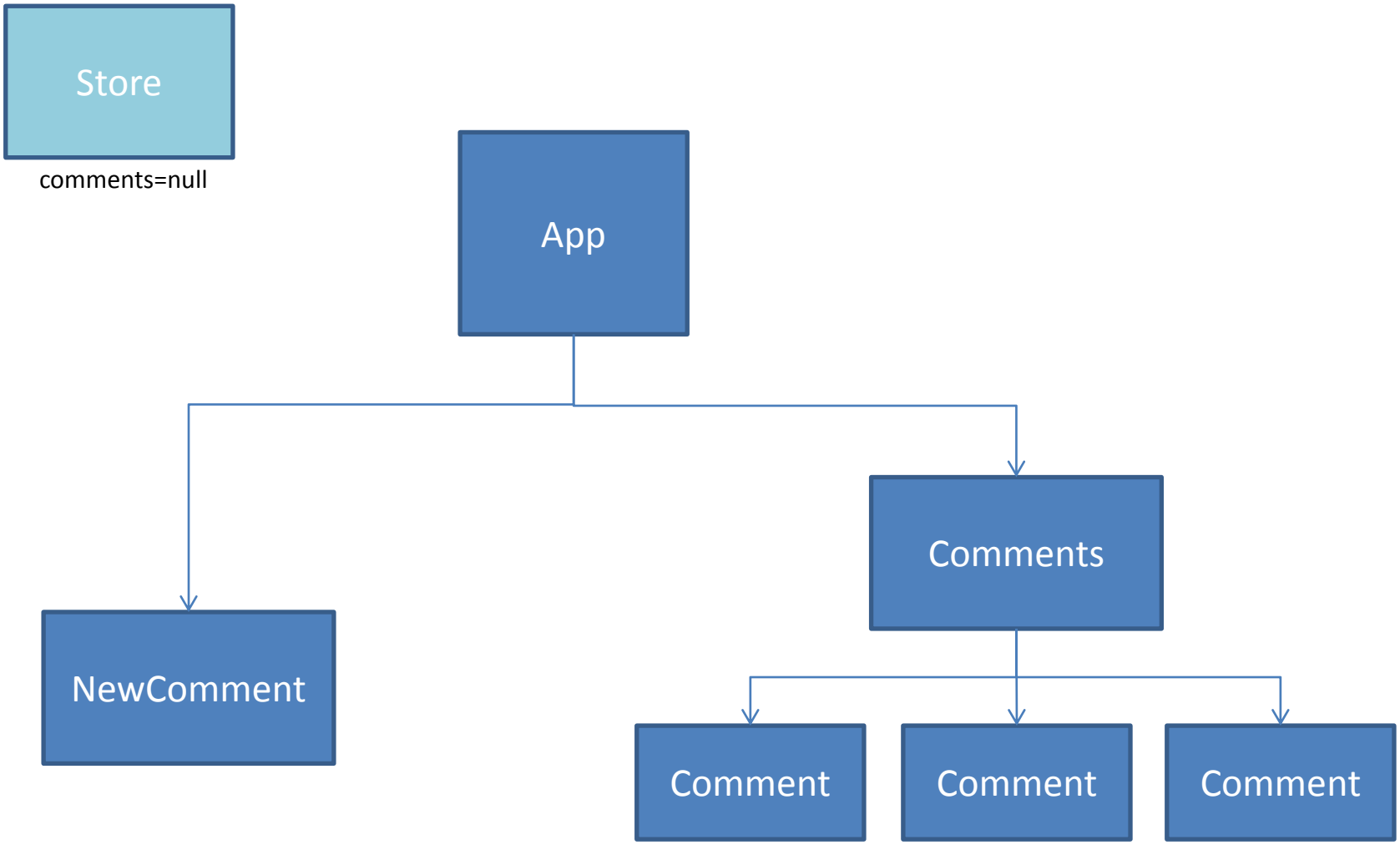
Your component structure should like this:



9.6. Task

5. Create a Redux store with one reducer and one state. The state contains of one variable “comments“ which is initially set to null.
6. Integrate the Redux-Thunk middleware. Do not define any actions yet. Just make Thunk work that the App can restart without any errors.

9.6. Task


Store
comments=null
App
NewComment
Comments
Comment
Comment
Comment

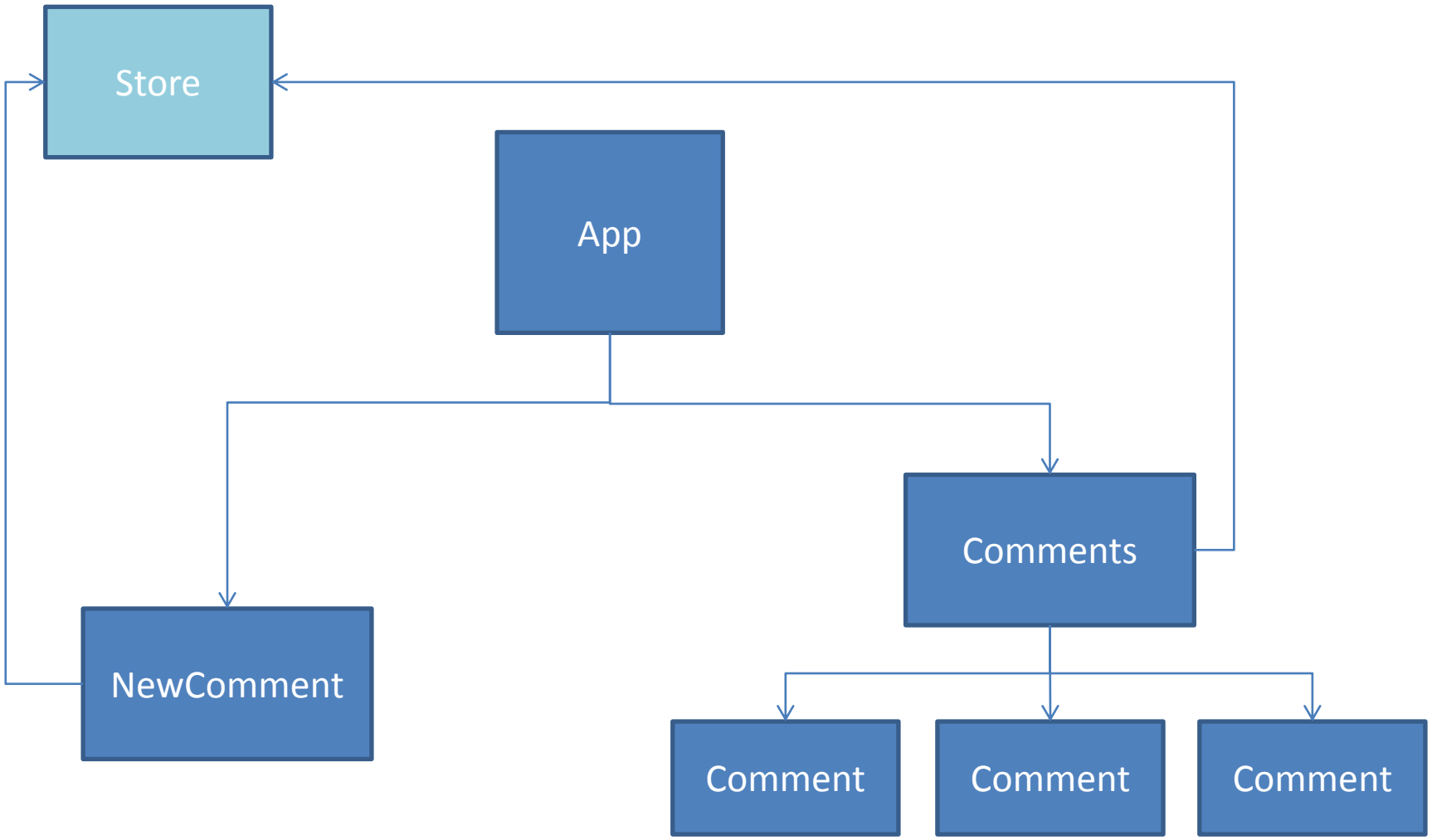
9.6. Task

7. For Thunk, create three asynchronous actions getComments, postComment and deleteComment.
   - getComments: sends an Axios GET request to http://localhost:3001/comments and updates the state's comments variable with the response
   - postComments: expect two parameters name and text and sends Axios POST request to http://localhost:3001/comments
     - The POST-body must consist of the name and the text, i.e. { name: "Paul", comment: "Great weather!" }
     - The response will be the sent POST-body plus an additional id generated by the server. This id must also be saved for each comment saved in the state's comments variable
   - deleteComment: sends an Axios DELETE request to http://localhost/comments/:id whereas :id is the id of the respective comment. If the comment was successfully deleted, the response will be { errorId: 0 }

9.6. Task

8. Connect the NewComment component‘s dispatcher with the Redux store. Make sure you have all of the three actions getComments, postComment and deleteComment available as action.
9. Connect the Comments component‘s props with the Redux store.

9.6. Task



9.6. Task

10. Implement the following behaviour:
    - When the Comments component mounts, all comments will be loaded via getComments
    - When the “Create comment“ button is clicked, a new comment based on the two input boxes will created
    - When the “X“-button is clicked, the comment will be removed.

Agenda – Part 9

1. React Hooks
2. Task
3. GraphQL
4. Task

9.1. React Hooks

- Until now, components could only be stateless or stateful.
- Stateful components: classes
  - Or in other ES5-words: function prototypes
  - They have a state
- Stateless components: functions
  - They do not have a state