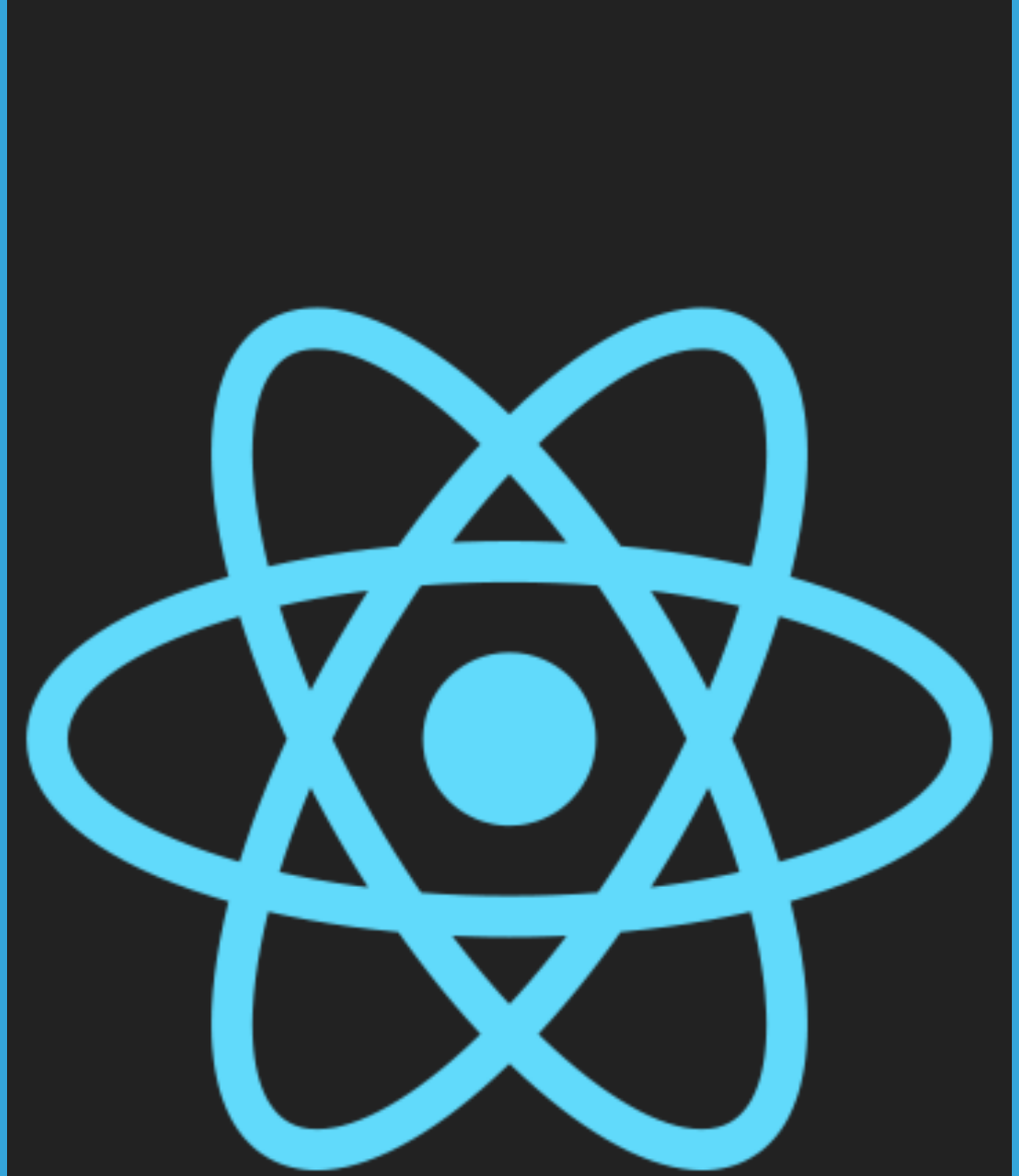
FRONT-END SOFTBINATOR-LABS

CURS 3: REACT

intro 1



CREAREA UNUI PROIECT

create-react-app

Idei principale

- React - A JavaScript library for building user interfaces
- Există mai multe moduri de a începe un proiect React, în funcție de nevoi (SPA, server-rendered, component library)
- create-react-app este un loc comod de unde se poate începe

create-react-app

- Pentru a folosi aceasta comandă nu este nevoie decât de:
  - Node >=14.0.0 *
  - npm >= 5.6 *
- După instalarea acestora, trebuie doar să rulăm
  npx create-react-app my-app
  în terminal

* versiunile cerute la momentul actual

create-react-app

```
(base) ictatuta@MacBook-Pro-8 softbinator % npx create-react-app test-app
Need to install the following packages:
  create-react-app@5.0.1
Ok to proceed? (y) y

Success! Created test-app at /Users/ictatuta/Documents/Projects/softbinator/test-app
Inside that directory, you can run several commands:

  npm start
    Starts the development server.

  npm run build
    Bundles the app into static files for production.

  npm test
    Starts the test runner.

  npm run eject
    Removes this tool and copies build dependencies, configuration files
    and scripts into the app directory. If you do this, you can't go back!

We suggest that you begin by typing:

  cd test-app
  npm start

Happy hacking!
```

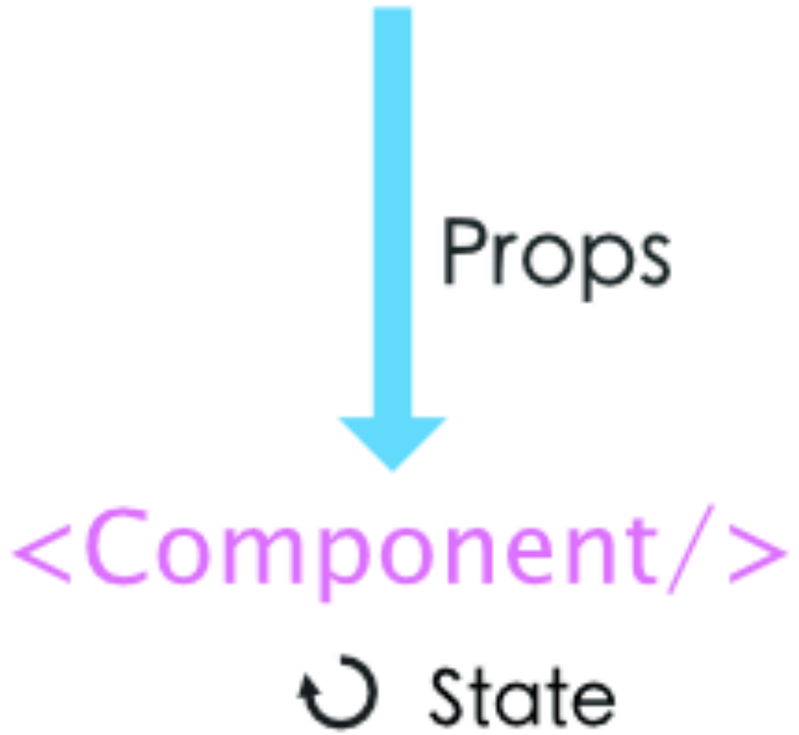
create-react-app

- La prima rulare se va cere instalarea modulului create-react-app ce va fi folosit de npx

```
(base) ictatuta@MacBook-Pro-8 softbinator % npx create-react-app test-app
Need to install the following packages:
  create-react-app@5.0.1
Ok to proceed? (y) y
```

```
Success! Created test-app at /Users/ictatuta/Documents/Projects/softbinator/test-app
Inside that directory, you can run several commands:

  npm start
    Starts the development server.

  npm run build
    Bundles the app into static files for production.

  npm test
    Starts the test runner.

  npm run eject
    Removes this tool and copies build dependencies, configuration files
    and scripts into the app directory. If you do this, you can't go back!

We suggest that you begin by typing:

  cd test-app
  npm start

Happy hacking!
```

- După creare, putem naviga in folderul cu numele proiectului



STATE AND PROPS

what are they

Idei principale

- Într-o componentă React de tip clasă avem:
  - state: Un obiect JavaScript ce conține toate datele *interne* necesare componentei
  - props: Un obiect JavaScript ce conține toate datele *primite* de componentă
- Valoarea lor se poate citi din this.state respectiv this.props
- state poate fi modificat intern folosind this.setState() și poate ție valori din formulare, rezultate ale unor operații, etc.
- props nu se pot modifica
  este o colecție de valori transmise “copiilor” ce pot fi folosite intern pentru diferite funcționalități

State

```
1   import React from 'react';
2
3   class ClassComponent extends React.Component {
4       constructor (props) {
5           super(props);
6           this.state = {
7               aNumber: 10,
8               aString: 'ten',
9               anArray: [8,9,10],
10              anObject: {
11                  surprise: 'hello'
12              }
13          }
14      }
15  }
16
17  export default ClassComponent;
```

State

- Valorile inițiale sunt definite în constructor
- Acestea pot fi modificate în metodele lifecycle, sau cu alte funcții handler

```
1   import React from 'react';
2
3   class ClassComponent extends React.Component {
4       constructor (props) {
5           super(props);
6           this.state = {
7               aNumber: 10,
8               aString: 'ten',
9               anArray: [8,9,10],
10              anObject: {
11                  surprise: 'hello'
12              }
13          }
14      }
15  }
16
17  export default ClassComponent;
```

State

- Aceste valori sunt folosite în metoda render ce definește structura interfeței

```
16      render() {
17          const { aNumber, aString, anArray, anObject } = this.state;
18
19          return (
20              <div>
21                  <div>
22                      Here's the number: {aNumber}
23                  </div>
24
25                  <div>
26                      Here's the string: {aString}
27                  </div>
28
29                  <div>
30                      Here's the string elements, mapped to individual rows:
31                      {
32                          anArray.map((element) => <div key={element} >{element}</div>)
33                      }
34                  </div>
35
36                  <div>
37                      You can't really see Objects in the UI, so this is what's inside: {anObject.surprise}
38                  </div>
39              </div>
40          )
41      }
```

*va fi explicat în cursul următor

Props

```
return (
  <div className="App">
    <header className="App-header">
      <img src={logo} className="App-logo" alt="logo" />
      <p>
        Edit <code>src/App.js</code> and save to reload.
      </p>
      <a
        className="App-link"
        href="https://reactjs.org"
        target="_blank"
        rel="noopener noreferrer"
      >
        Learn React
      </a>
    </header>
    <ClassComponent aPassedValue={10} anotherPassedValue />
    <ClassComponent>
      <div>this is the 'children' prop</div>
    </ClassComponent>
  </div>
);
```

Props

- Valorile props se trimit prin structura jsx a unei componente
- Se pot trimite prin sintaxa de tip atribut=valoare, sau sub forma elementelor children

```
return (
  <div className="App">
    <header className="App-header">
      <img src={logo} className="App-logo" alt="logo" />
      <p>
        Edit <code>src/App.js</code> and save to reload.
      </p>
      <a
        className="App-link"
        href="https://reactjs.org"
        target="_blank"
        rel="noopener noreferrer"
      >
        Learn React
      </a>
    </header>
    <ClassComponent aPassedValue={10} anotherPassedValue />
    <ClassComponent>
      <div>this is the 'children' prop</div>
    </ClassComponent>
  </div>
);
```

Props

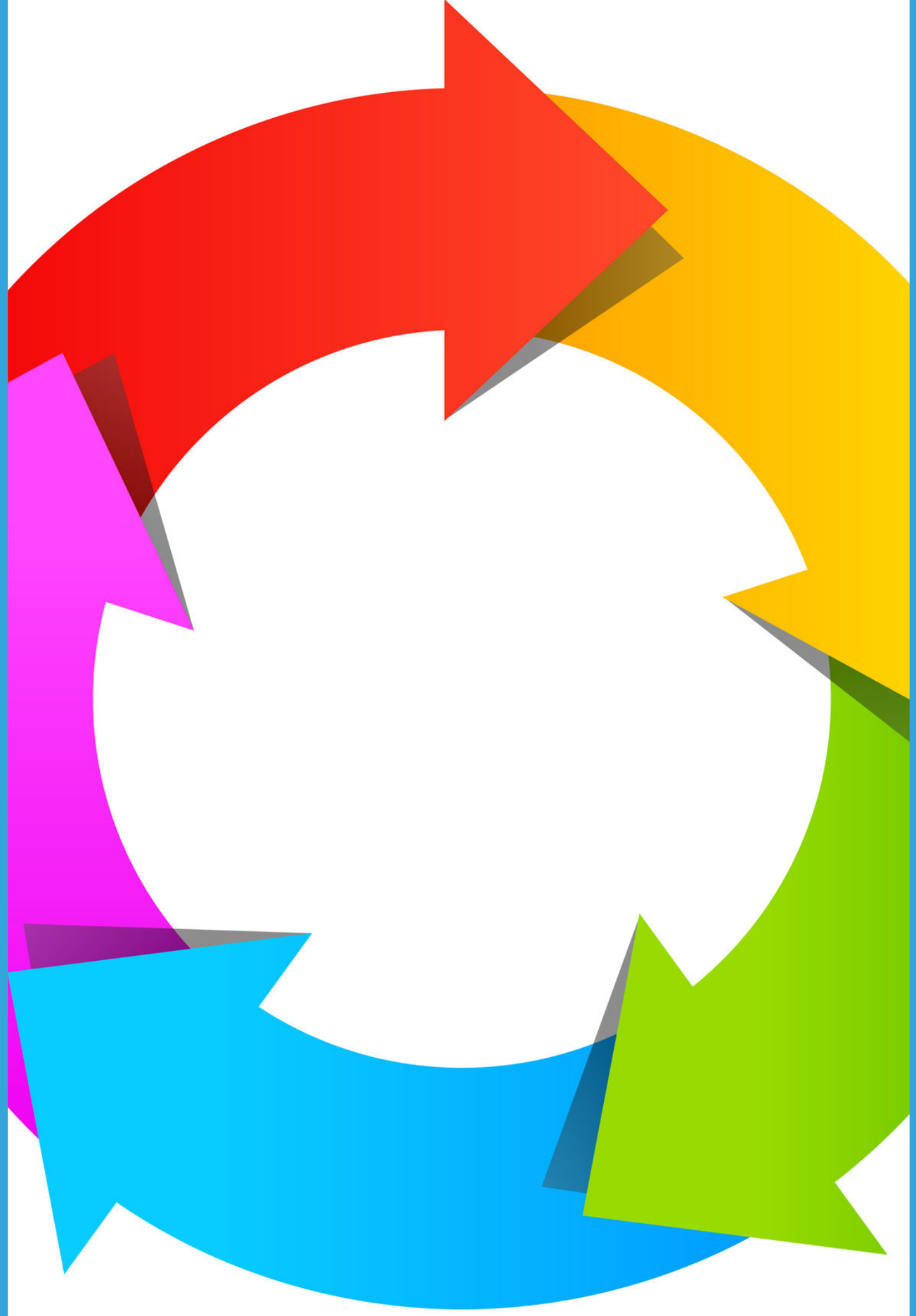
- Pot fi folosite în metoda render

```
render() {
    const { aNumber, aString, anArray, anObject } = this.state;

    const { aPassedValue, anotherPassedValue, children } = this.props;

    return (
        <div>
            <p>Props:</p>
            <div>
                Here's a passed value: {aPassedValue}
            </div>

            <div>
                Here's another passed value (this is a boolean): {anotherPassedValue.toString()}
            </div>

            <div>
                Here are the children: {children}
            </div>
```

Props

- Sau pentru valoarea default a state-ului

```
constructor (props) {
    super(props);
    this.state = {
        aNumber: 10,
        aString: 'ten',
        anArray: [8,9,10],
        anObject: {
            surprise: 'hello'
        },
        someDefaultedValue: props.aPassedValue
    }
}
```



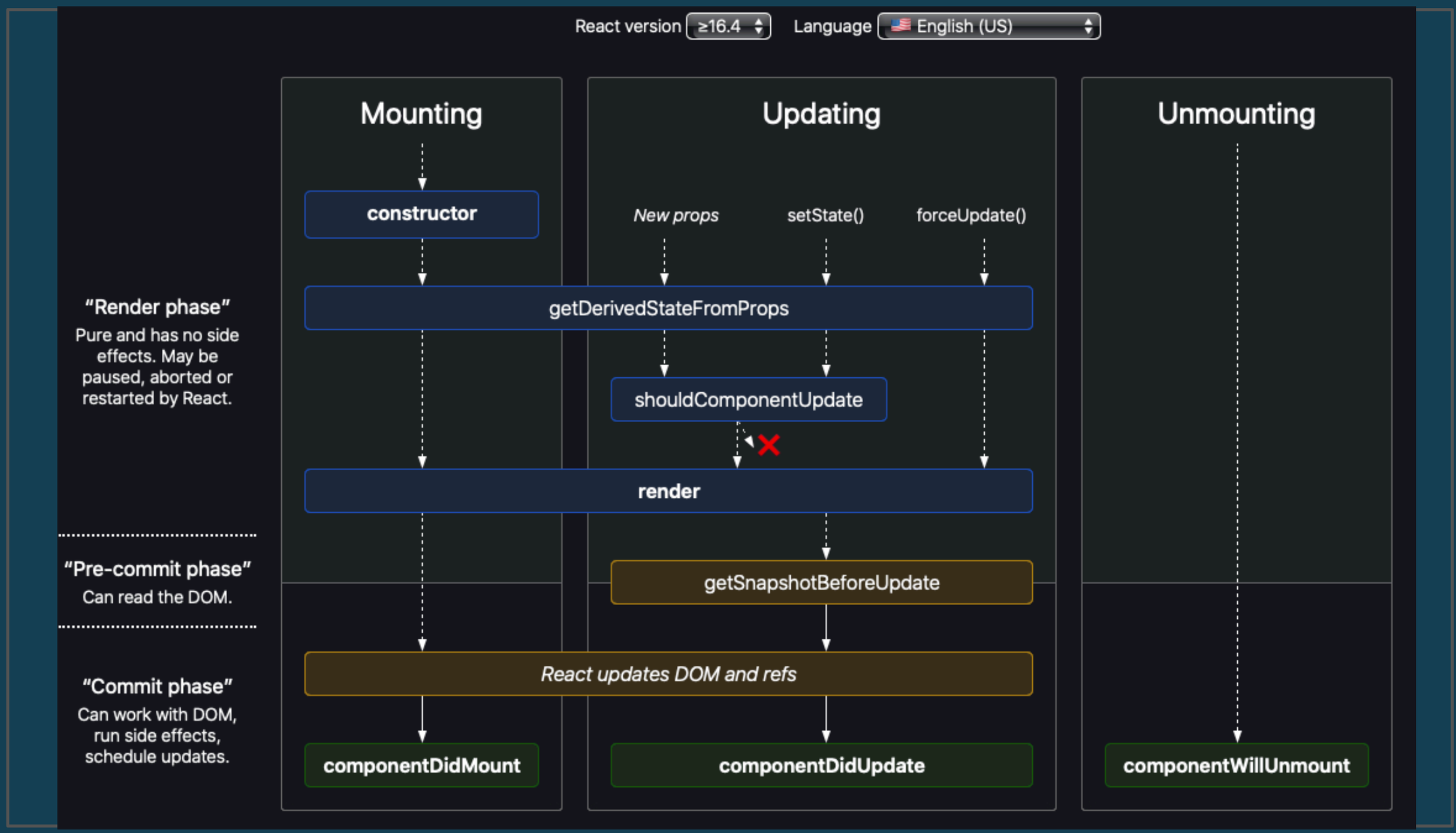
COMPONENT LIFECYCLE

component Did What ??

Idei principale

- Metodele de lifecycle sunt disponibile componentelor de tip clasa (nu și cele de tip funcție)
- Sunt 3 categorii principale: Mounting, Updating si Unmounting
- Majoritatea sunt similare cu useEffect (React hook - va fi prezentat într-un curs viitor)

Overview


React version ≥16.4
Language English (US)
Mounting
Updating
Unmounting
constructor
New props
setState()
forceUpdate()
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()

- Este cea mai folosita metoda lifecycle
- Trebuie sa fie functie pura, fara side-effects (nu avem voie setState)

- Putem returna null daca nu vrem sa afisam nimic

```
class Hello extends Component{
    render () {
        return <div>Hello {this.props.name}</div>
    }
}
```

componentDidMount()

```
class Hello extends Component{
  constructor(props) {
    super(props);
    this.state = {
      age: 0,
    }
  }

  componentDidMount() {
    this.props.fetchServerAge(this.props.userName).then(resp => {
      this.setState({age: resp});
    })
  }

  render () {
    return (
      <div>
        Hello {this.props.name}
        and I'm {this.state.age} years old
      </div>
    )
  }
}
```

componentDidMount()

- Dupa cum sugereaza numele, se apeleaza imediat dupa mount
- Putem folosi setState() in el insa trebuie sa avem grija
- Recomandarea este sa dam valori initiale state-ului in constructor

```
class Hello extends Component{
  constructor(props) {
    super(props);
    this.state = {
      age: 0,
    }
  }

  componentDidMount() {
    this.props.fetchServerAge(this.props.userName).then(resp => {
      this.setState({age: resp});
    })
  }

  render () {
    return (
      <div>
        Hello {this.props.name}
        and I'm {this.state.age} years old
      </div>
    )
  }
}
```

componentDidUpdate(prevProps, prevState)

```
class Hello extends Component{
  componentDidUpdate(prevProps, prevState) {
    //Typical usage, don't forget to compare the props
    if (this.props.userName !== prevProps.userName) {
      this.props.fetchServerAge(this.props.userName).then(resp => {
        this.setState({age: resp});
      });
    }
  }

  render () {
    return <div>Hello {this.props.name}</div>
  }
}
```

componentDidUpdate(prevProps, prevState)

- Este invocata imediat dupa un update
- De cele mai multe ori e vorba de o schimbare din props sau state
- Putem folosi setState() insa exista riscul de loop infinit, astfel se recomanda utilizarea unei conditii

```
class Hello extends Component{
  componentDidUpdate(prevProps, prevState) {
    //Typical usage, don't forget to compare the props
    if (this.props.userName !== prevProps.userName) {
      this.props.fetchServerAge(this.props.userName).then(resp => {
        this.setState({age: resp});
      });
    }
  }

  render () {
    return <div>Hello {this.props.name}</div>
  }
}
```

shouldComponentUpdate(nextProps, nextState)

```
class Hello extends Component{
  shouldComponentUpdate(nextProps, nextState) {
    return this.props.title !== nextProps.title
      || this.state.input !== nextState.input
    }

  render () {
    return (
      <div>
        The title:
        {this.props.title}
      </div>
    )
  }
}
```

shouldComponentUpdate(nextProps, nextState)

- Este o metoda mai putin uzuala ce poate preveni render la anumite schimbari de props/state
- Valoarea returnata (true/false) este folosita pentru a decide daca e nevoie de re-render
- Nu permite apelarea lui setState
- In mod normal componenta face render la fiecare apel de setState, insa prin aceasta metoda putem preveni asta
- Mentiune: trebuie folosita cat de rar posibil si exista doar pentru optimizari de performanta

```
class Hello extends Component{
  shouldComponentUpdate(nextProps, nextState) {
    return this.props.title !== nextProps.title
      || this.state.input !== nextState.input
    }

  render () {
    return (
      <div>
        The title:
        {this.props.title}
      </div>
    )
  }
}
```

componentWillUnmount()

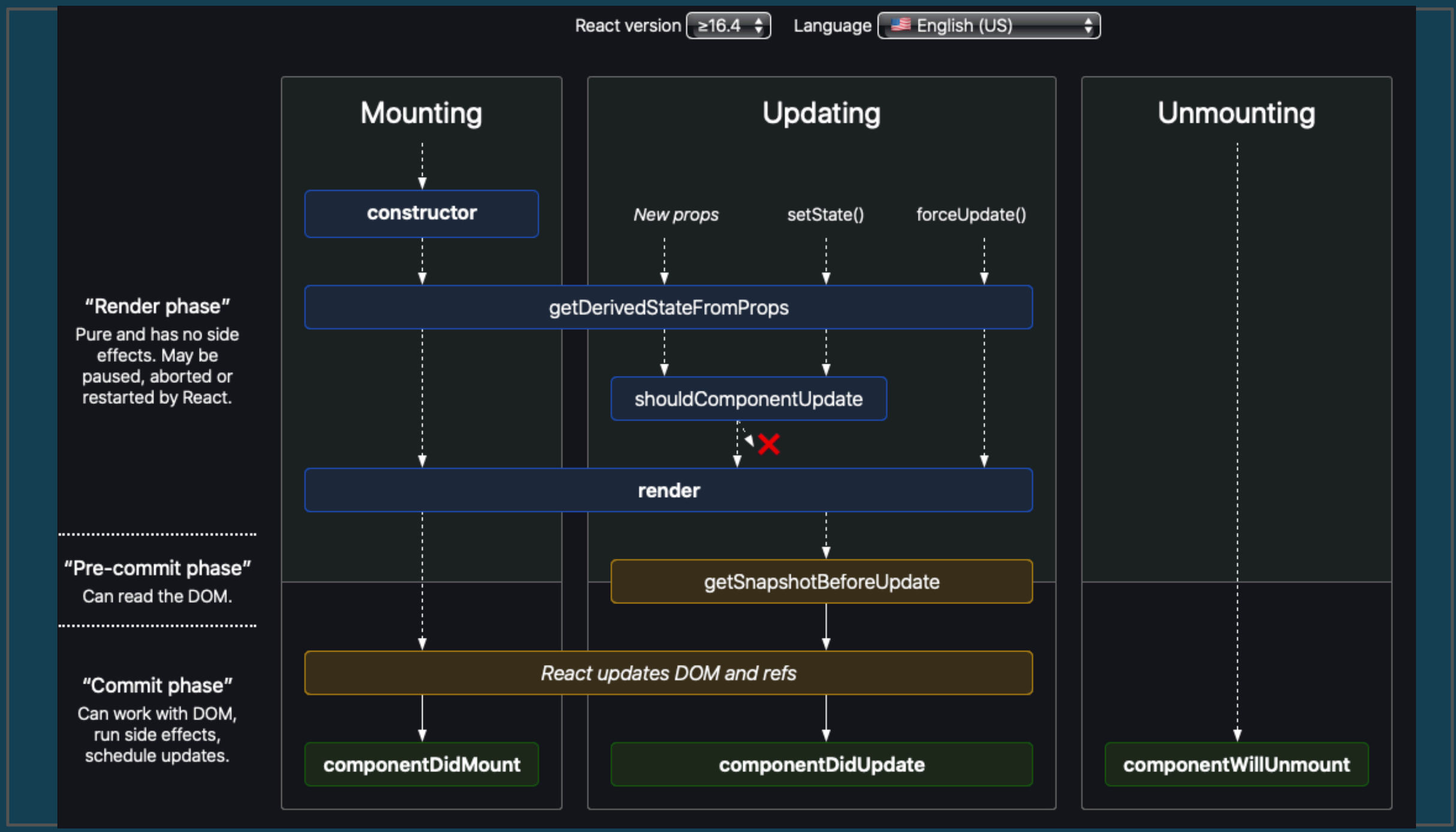
```
class Hello extends Component{
  doSomething = (e) => {
    console.log('window resized', e)
  }

  componentDidMount() {
    window.addEventListener('resize', this.resizeListener)
  }

  componentWillUnmount() {
    window.removeEventListener('resize', this.resizeListener)
  }

  render () {
    return (
      <div>
        Try resizing
      </div>
    )
  }
}
```

componentWillUnmount()

- Metoda aceasta este apelata fix inainte de unmount si distrugerea componentei.
- Este folosita pentru cleanup (stergere de event listeners, oprirea unui fetch, clearTimeout/ clearInterval, etc.)

```
class Hello extends Component{
  doSomething = (e) => {
    console.log('window resized', e)
  }

  componentDidMount() {
    window.addEventListener('resize', this.resizeListener)
  }

  componentWillUnmount() {
    window.removeEventListener('resize', this.resizeListener)
  }

  render () {
    return (
      <div>
        Try resizing
      </div>
    )
  }
}
```

Recap


React version ≥16.4
Language English (US)
Mounting
Updating
Unmounting
constructor
New props
setState()
forceUpdate()
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.

HOOKS

(only useState for now)

Idei principale

- Permit utilizarea unui state in componente funcționale (useState)
- Sunt modulare => permit refolosirea logicii
- Codul este usor de inteles chiar si in componente complexe

useState

```jsx
import React, { useState } from 'react';

function Example() {
  // Declare a new state variable, which we'll call "count"
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

useState

- Primeste un singur parametru: valoarea initiala
- Poate fi folosit de mai multe ori in aceeasi componenta

```jsx
import React, { useState } from 'react';

function Example() {
  // Declare a new state variable, which we'll call "count"
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

Final Thoughts

- Deși în forme diferite, componentele React sunt bazate pe state, props, și metode ce “reacționează” la schimbările acestora
- Componentele funcționale cu Hooks sunt mai compacte și sunt de preferat
- Metodele de lifecycle din componentele clasa sunt inca utile; desi sunt mai vechi sunt inca folosite in multe proiecte

Link-uri utile

- Creating a React App
  https://reactjs.org/docs/create-a-new-react-app.html
- State
  https://reactjs.org/docs/faq-state.html
- Props
  https://reactjs.org/docs/components-and-props.html
- Component Lifecycle:
  https://programmingwithmosh.com/javascript/react-lifecycle-methods/
  https://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/
  https://reactjs.org/docs/state-and-lifecycle.html
- Props and useState:
  https://reactjs.org/docs/hooks-overview.html
  https://reactjs.org/docs/hooks-state.html