

Systemutveckling

Ramverk

25 HP

Kursupplägget

| Föreläsning | | | Innehåll (FM) | Övningar (EM) | Inlämning |
|-------------|--------------|---|---|--|--------------------------|
| 1 | mån 25 mars | D | Introduktion Typescript (Basic Types, Type Inference, Variable Declarations, Iterators and Generators) | TypeScripts Hemsida TypeScript in 5 minutes + övning | |
| 2 | ons 27 mars | V | Typescript forts. (Functions, Classes, Interfaces, Modules) | | |
| 3 | fre 29 mars | V | Introduktion React (SPA, Virtuellt DOM, Kap. 1-6) | React's Hemsida Main Concepts inklusive övningar i CodePen Kodövning (RP) | |
| 4 | mån 1 apr. | D | React Playground (1-initial-setup + 2-layout) | | Inlämning 1 ges ut |
| 5 | ons 3 apr. | V | React forts. (Kap. 7-12) | | |
| 6 | fre 5 apr. | V | Create React App TS Intro | Övning "Todo App" | Handledning |
| 7 | tis 9 apr. | D | React Playground (3-navigation-with-state) React Playground (4-navigation-with-routes) | Kodövning (RP) | |
| 8 | ons 10 apr. | V | React Playground (5-code-splitting) React Playground (6-error-boundary) | Kodövning (RP) | Inlämning 1 lämnas in |
| 9 | fre 12 apr. | V | React Playground (7-portals) | Kodövning (RP) | Inlämning 2 ges ut |
| 10 | mån 15 apr. | D | React Playground (8-code-split-app-start) | Kodövning (RP) | |
| 11 | tis 16 apr. | D | React Playground (9-api-lib-axios) | Kodövning (RP) | Handledning |
| 12 | tis 23 apr. | D | React Playground (10-context) | Kodövning (RP) | |
| 13 | tors 25 apr. | V | Tentaplugg | | Inlämning 2 lämnas in |
| 14 | fre 26 apr. | V | TENTAMEN | | |

React

A JavaScript library for building user interfaces

Fast, descriptive, composable and fun!

Kap 7-12

Conditional Rendering

React - Conditional Rendering

To render or not to render, that is the question...

Villkor för att avgöra vad som skall renderas

```
function UserGreeting(props) {  
  return <h1>Welcome back!</h1>;  
}
```

```
function GuestGreeting(props) {  
  return <h1>Please sign up.</h1>;  
}
```



```
function Greeting(props) {  
  const isLoggedIn = props.isLoggedIn;  
  if (isLoggedIn) {  
    return <UserGreeting />;  
  }  
  return <GuestGreeting />;  
}
```

```
ReactDOM.render(  
  // Try changing to isLoggedIn={true}:  
  <Greeting isLoggedIn={false} />,  
  document.getElementById('root')  
);
```

React - Conditional Rendering

Save element to variable

Alla typer av element kan sparas i variabler.

Underlättar conditional rendering.

Gör applikationen mer modulerbar i koden.

```
function LoginButton(props) {  
  return (  
    <button onClick={props.onClick}>  
      Login  
    </button>  
  );  
}  
  
function LogoutButton(props) {  
  return (  
    <button onClick={props.onClick}>  
      Logout  
    </button>  
  );  
}
```

```
render() {  
  const isLoggedIn = this.state.isLoggedIn;  
  let button;  
  
  if (isLoggedIn) {  
    button = <LogoutButton onClick={this.handleLogoutClick} />;  
  } else {  
    button = <LoginButton onClick={this.handleLoginClick} />;  
  }  
  
  return (  
    <div>  
      <Greeting isLoggedIn={isLoggedIn} />  
      {button}  
    </div>  
  );  
}
```

React - Conditional Rendering

Inline if - else

Conditional rendering kan styras genom if-satser på en rad (inline if - else).

Mindre och ”snyggare” kod (upp till utvecklare).

När villkor blir för komplexa bör koden brytas ut till flera komponenter.

```
render() {  
  const isLoggedIn = this.state.isLoggedIn;  
  return (  
    <div>  
      {isLoggedIn ? (  
        <LogoutButton onClick={this.handleLogoutClick} />  
      ) : (  
        <LoginButton onClick={this.handleLoginClick} />  
      )}  
    </div>  
  );  
}
```

```
render() {  
  const isLoggedIn = this.state.isLoggedIn;  
  return (  
    <div>  
      The user is <b>{isLoggedIn ? 'currently' : 'not'}</b> logged in.  
    </div>  
  );  
}
```

Lists and Keys

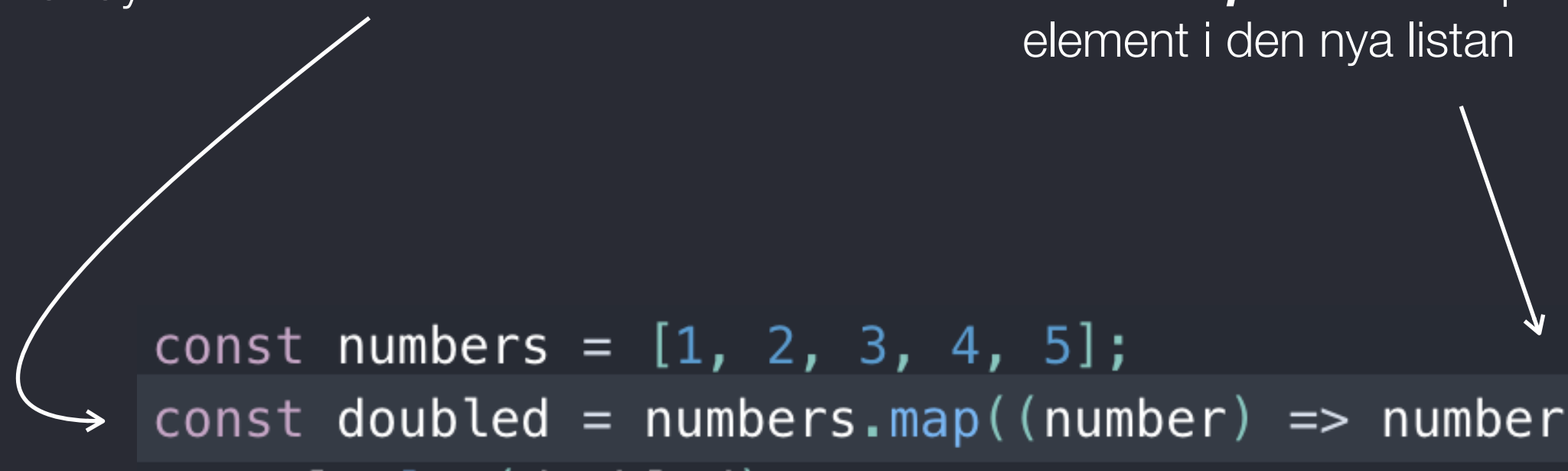
React - List and Keys

What is a map?

Nedanstående map skapar en ny array med dubblerade värden ifrån arrayen "numbers".

En **map** är till för att skapa en ny lista/array

Det som returneras i callbackfunktionen för en **map** kommer representera ett element i den nya listan



```
const numbers = [1, 2, 3, 4, 5];  
const doubled = numbers.map((number) => number * 2);  
console.log(doubled);
```

React - List and Keys

What is a map?

En **map** kan användas för att rendera ut flera element eller komponenter åt gången.

Nedan körs en **map** som kommer generera en lista som innehåller 5 st. `` element.



```
const numbers = [1, 2, 3, 4, 5];  
const listItems = numbers.map((number) =>  
  <li>{number}</li>  
);
```

För att rendera ut en lista med element anges variabeln som listan är sparad i inom `{ }` i `render()`.



```
ReactDOM.render(  
  <ul>{listItems}</ul>,  
  document.getElementById('root')  
);
```

React - List and Keys

What is Keys for?

Keys i en **map** används för att hjälpa **React** förstå att något element ändrats, lagts till eller tagits bort ur listan.

En key anges med prop:en **key** på "root elementet". En **key** skall, hos alla syskonelement, vara ett unikt värde. Unikt ID i elementen som ittereras är lämpligt!



```
const todoItems = todos.map((todo) =>  
  <li key={todo.id}>  
    {todo.text}  
  </li>  
);
```

Elementens index kan användas som **key:s** som en sista utväg om ej ID finns. Detta kan dock trigga oönskade omrenderingar om element byter plats i listan.



```
const todoItems = todos.map((todo, index) =>  
  // Only do this if items have no stable IDs  
  <li key={index}>  
    {todo.text}  
  </li>  
);
```

React - List and Keys

Embedded maps

Maps kan bli inbäddade direkt i JSX-element.

Not embedded



```
function NumberList(props) {
  const numbers = props.numbers;
  const listItems = numbers.map((number) =>
    <ListItem key={number.toString()}
      value={number} />
  );
  return (
    <ul>
      {listItems}
    </ul>
  );
}
```

Embedded



```
function NumberList(props) {
  const numbers = props.numbers;
  return (
    <ul>
      {numbers.map((number) =>
        <ListItem key={number.toString()}
          value={number} />
      )}
    </ul>
  );
}
```

Forms

React - Forms

Controlled component

En kontrollerad komponent betyder att dess state kontrollerar komponentens innehåll.

Staten uppdateras så fort en ändring görs i inputfälten.

Kontexten för 'this' måste bindas då vi inte använder arrow-functions.

```
class NameForm extends React.Component {
  constructor(props) {
    super(props);
    this.state = {value: ''};

    this.handleChange = this.handleChange.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
  }

  handleChange(event) {
    this.setState({value: event.target.value});
  }

  handleSubmit(event) {
    alert('A name was submitted: ' + this.state.value);
    event.preventDefault();
  }

  render() {
    return (
      <form onSubmit={this.handleSubmit}>
        <label>
          Name:
          <input type="text" value={this.state.value} onChange={this.handleChange} />
        </label>
        <input type="submit" value="Submit" />
      </form>
    );
  }
}
```

React - Forms

Multiple inputs

```
class Reservation extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      isGoing: true,
      numberOfGuests: 2
    };
  }

  this.handleChange = this.handleChange.bind(this);
}

handleChange(event) {
  const target = event.target;
  const value = target.type === 'checkbox' ? target.checked : target.value;
  const name = target.name;

  this.setState({
    [name]: value
  });
}

render() {
  return (
    <form>
      <label>
        Is going:
        <input
          name="isGoing"
          type="checkbox"
          checked={this.state.isGoing}
          onChange={this.handleChange} />
        </label>
      <br />
      <label>
        Number of guests:
        <input
          name="numberOfGuests"
          type="number"
          value={this.state.numberOfGuests}
          onChange={this.handleChange} />
        </label>
      </form>
    );
  }
}
```

Värdet ifrån input-fälten hämtas ut och sparas i komponentens state.

Skall flera input-fält användas kan dessa definieras genom typ eller namn.

Lifting State Up

Lifting State Up

Why?

Ofta är fler komponenter, på ett eller annat sätt, beroende av samma data.

Lyft upp **State** till *"their closest common ancestor"*.

Sträva efter 'Single Source Of Truth'.

Gör koden skalbar!

Lifting State Up

Temperature example

```
function tryConvert(temperature, convert) {  
  const input = parseFloat(temperature);  
  if (Number.isNaN(input)) {  
    return '';  
  }  
  const output = convert(input);  
  const rounded = Math.round(output * 1000) / 1000;  
  return rounded.toString();  
}
```

```
function BoilingVerdict(props) {  
  if (props.celsius >= 100) {  
    return <p>The water would boil.</p>;  
  }  
  return <p>The water would not boil.</p>;  
}
```

```
class TemperatureInput extends React.Component {  
  constructor(props) {  
    super(props);  
    this.handleChange = this.handleChange.bind(this);  
  }  
  
  handleChange(e) {  
    this.props.onTemperatureChange(e.target.value);  
  }  
  
  render() {  
    const temperature = this.props.temperature;  
    const scale = this.props.scale;  
    return (  
      <fieldset>  
        <legend>Enter temperature in {scaleNames[scale]}:</legend>  
        <input value={temperature} />  
        <input type="checkbox" checked={scale === 'Celsius'} /> Celsius  
        <input type="checkbox" checked={scale === 'Fahrenheit'} /> Fahrenheit  
      </fieldset>  
    );  
  }  
}
```

Lifting State Up

Temperature example

```
class Calculator extends React.Component {
  constructor(props) {
    super(props);
    this.handleCelsiusChange = this.handleCelsiusChange.bind(this);
    this.handleFahrenheitChange = this.handleFahrenheitChange.bind(this);
    this.state = {temperature: '', scale: 'c'};
  }

  handleCelsiusChange(temperature) {
    this.setState({scale: 'c', temperature});
  }

  handleFahrenheitChange(temperature) {
    this.setState({scale: 'f', temperature});
  }

  render() {
    const scale = this.state.scale;
    const temperature = this.state.temperature;
    const celsius = scale === 'f' ? tryConvert(temperature, toCelsius) : temperature;
    const fahrenheit = scale === 'c' ? tryConvert(temperature, toFahrenheit) : temperature;

    return (
      <div>
        <TemperatureInput
          scale="c"
          temperature={celsius}
          onTemperatureChange={this.handleCelsiusChange} />

        <TemperatureInput
          scale="f"
          temperature={fahrenheit}
          onTemperatureChange={this.handleFahrenheitChange} />

        <BoilingVerdict
          celsius={parseFloat(celsius)} />

      </div>
    );
  }
}
```

Lifting State Up

Temperature example lessons

Skicka ner funktioner ifrån *"The closest common ancestor"* som props kan ändra dess state.

`<Calculator />` är i det här fallet *"The Single Source Of Truth"*.

Applikationen är skalbar och fler `<TemperatureInput />` kan enkelt läggas till.

The screenshot shows a web application with two input fields: "Enter temperature in Celsius:" and "Enter temperature in Fahrenheit:". Below the inputs, a message reads "The water would not boil.".

Below the application, the React DevTools component inspector is open, showing the component tree and the props/state for the selected `<Calculator>` component.

Component Tree:

```
<Calculator>
  <div>
    <TemperatureInput scale="c" temperature="" onTemperatureChan...
    <TemperatureInput scale="f" temperature="" onTemperatureChan...
    <BoilingVerdict celsius=null>...</BoilingVerdict>
  </div>
</Calculator>
```

Props: Empty object

State: scale: "c", temperature: ""

Composition vs Inheritance

Composition vs inheritance

Composition

”React has a powerful composition model, and we recommend using composition instead of inheritance to reuse code between components.”

Någonting som är uppbyggt av flera delar, i fallet med React är dessa delar komponenter.

Ökar återanvändbarheten i koden (komponenter)

Composition is the React way!

Composition vs inheritance

Inheritance

Inheritance (arv) finns för att definiera att någonting är samma sak som något annat.

En subklass är samma sak som dess superklass fast med tillagd data (om detta defineras).

”At Facebook, we use React in thousands of components, and we haven’t found any use cases where we would recommend creating component inheritance hierarchies.”


Composition vs inheritance

Composition Containment

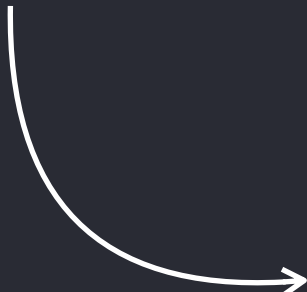
Ibland vet inte en komponent vilka **children** den har, detta kan rundgås genom att använda prop:en **Children**.

Detta betyder att alla children som deklarerats i komponenten **WelcomeDialog()** kommer renderas ut i komponenten **FancyBorder()**

```
function WelcomeDialog() {  
  return (  
    <FancyBorder color="blue">  
      <h1 className="Dialog-title">  
        Welcome  
      </h1>  
      <p className="Dialog-message">  
        Thank you for visiting our spacecraft!  
      </p>  
    </FancyBorder>  
  );  
}
```



```
function FancyBorder(props) {  
  return (  
    <div className={'FancyBorder FancyBorder-' + props.color}>  
      {props.children}  
    </div>  
  );  
}
```



Composition vs inheritance

Composition Specialization

Vill vi ytterligare specificera vad innehållet av komponenten FancyBorder() kan vi använda ytterligare **props** för att definiera detta.



```
function FancyBorder(props) {  
  return (  
    <div className={'FancyBorder FancyBorder-' + props.color}>  
      {props.children}  
    </div>  
  );  
}
```

```
function Dialog(props) {  
  return (  
    <FancyBorder color="blue">  
      <h1 className="Dialog-title">  
        {props.title}  
      </h1>  
      <p className="Dialog-message">  
        {props.message}  
      </p>  
    </FancyBorder>  
  );  
}  
  
function WelcomeDialog() {  
  return (  
    <Dialog  
      title="Welcome"  
      message="Thank you for visiting our spacecraft!" />  
  );  
}
```

Composition vs inheritance

Composition Specialization

Composition med ***Containment*** och
Specialization gäller även då vi
arbetar med ***klasser***



```
function FancyBorder(props) {
  return (
    <div className={'FancyBorder FancyBorder-' + props.color}>
      {props.children}
    </div>
  );
}
```

```
function Dialog(props) {
  return (
    <FancyBorder color="blue">
      <h1 className="Dialog-title">
        {props.title}
      </h1>
      <p className="Dialog-message">
        {props.message}
      </p>
      {props.children}
    </FancyBorder>
  );
}

class SignUpDialog extends React.Component {
  constructor(props) {
    super(props);
    this.handleChange = this.handleChange.bind(this);
    this.handleSignUp = this.handleSignUp.bind(this);
    this.state = {login: ''};
  }

  render() {
    return (
      <Dialog title="Mars Exploration Program"
        message="How should we refer to you?">
        <input value={this.state.login}
          onChange={this.handleChange} />

        <button onClick={this.handleSignUp}>
          Sign Me Up!
        </button>
      </Dialog>
    );
  }

  handleChange(e) {
    this.setState({login: e.target.value});
  }

  handleSignUp() {
    alert(`Welcome aboard, ${this.state.login}!`);
  }
}
```

The REACT way!

Thinking in React

Thinking in React

5 steps

”React is, in our opinion, the premier way to build big, fast Web apps with JavaScript. It has scaled very well for us at Facebook and Instagram.”

Arbeta med React i 5 steg



Thinking in React

Step 1

Bryt ner UI:t till en komponent hierarki.

| Name | Price |
|----------------|----------|
| Sporting Goods | |
| Football | \$49.99 |
| Baseball | \$9.99 |
| Basketball | \$29.99 |
| Electronics | |
| iPod Touch | \$99.99 |
| iPhone 5 | \$399.99 |
| Nexus 7 | \$199.99 |

1. **FilterableProductTable (orange)**: contains the entirety of the example
2. **SearchBar (blue)**: receives all *user input*
3. **ProductTable (green)**: displays and filters the *data collection* based on *user input*
4. **ProductCategoryRow (turquoise)**: displays a heading for each *category*
5. **ProductRow (red)**: displays a row for each *product*

Thinking in React

Step 2

Bygg dina komponenter utan interaktivitet (dumma komponenter) utifrån komponent hierarkin.



Thinking in React

Step 3

Finn minimalt antal state som behövs för att applikationen skall fungera.

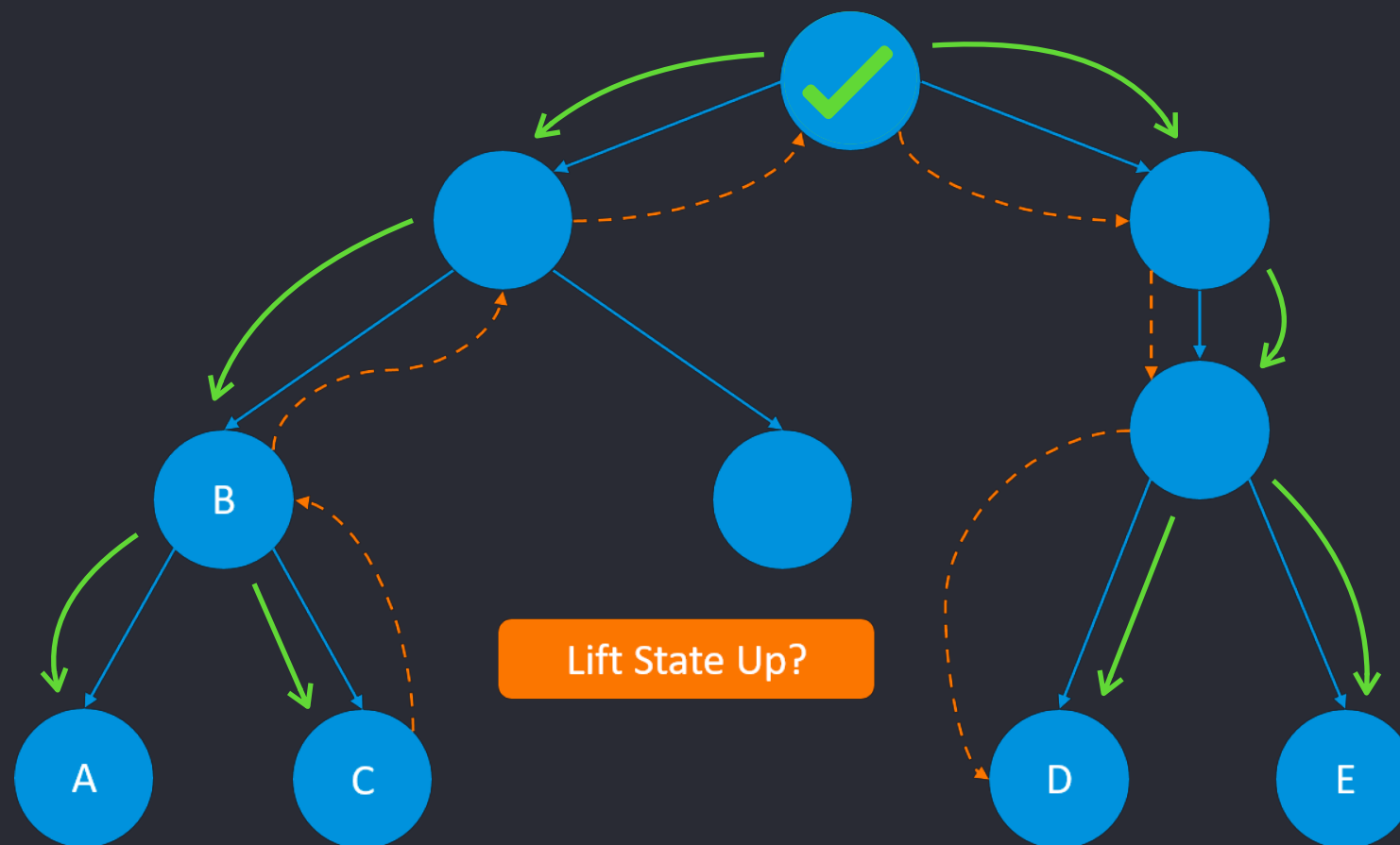


1. Is it passed in from a parent via props? If so, it probably isn't state.
2. Does it remain unchanged over time? If so, it probably isn't state.
3. Can you compute it based on any other state or props in your component? If so, it isn't state.

Thinking in React

Step 4

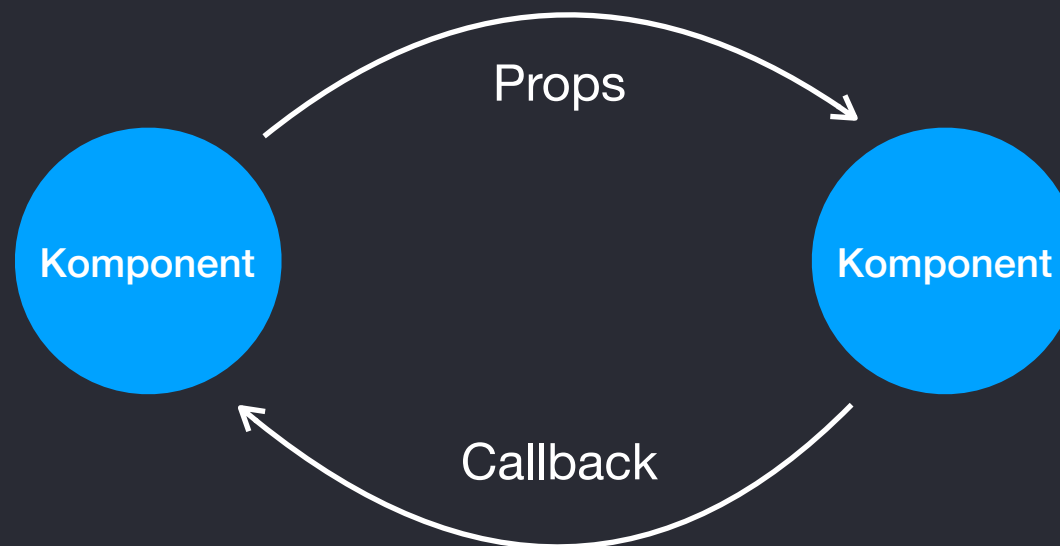
Identifiera i vilken komponent respektive state skall finnas Tänk på "Lifting State Up".
Implementera samtliga **State** och **Props** med flöde ner i applikationen.



Thinking in React

Step 5

Lätt till dataflöde tillbaka upp till "ovanstående" (Parent-) komponenter.
Callbacks används för detta.



Läsanvisningar

React Main Concepts

<https://reactjs.org/docs/hello-world.html>

Kapitel.

- 7. Conditional Rendering
- 8. Lists and Keys
- 9. Forms
- 10. Lifting State up
- 11. Composition vs Inheritance
- 12. Thinking in React

Nästa lektion

Crete React App

Utgå ifrån ett redan konfigurerat projekt för att snabbt komma igång

Resten av dagen

React Main Concepts

<https://reactjs.org/docs/hello-world.html>

Kapitel. 6-12

Läs noga och försök förstå så mycket som möjligt. Öppna Code Pen länkarna - ändra koden och se vad som händer.

Är du redan klar? Lek runt och testa mer i Code Pen, alternativt jobba vidare i React Playgorund.

Tack