

# **Systemutveckling**

# **Ramverk**

**25 HP**

# Kursupplägget

Föreläsning			Innehåll (FM)	Övningar (EM)	Inlämning
1	mån 25 mars	D	Introduktion Typescript (Basic Types, Type Inferrence, 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, Virtuell DOM, Kap. 1-6)	Reacts Hemsida Main Concepts inklusive övningar i CodePen Kodövning (RP)	Inlämning 1 ges ut
4	mån 1 apr.	D	React Playground (1-initial-setup + 2-layout)		
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 1-6**

# React - Know How

React filer använder filendelsen **.jsx**

För att en webbläsare skall kunna tolka koden måste den kompileras till JavaScript först.

React har ingen egen inbyggd kompilator - istället använder man sig av tex Webpack eller Babel för att göra om en **.jsx** fil till en **.js** fil.

Vi kommer att använda oss av Webpack - hur detta går till kommer vi att gå igenom på nästa lektion.

# React - Know How

React är ett JavaScript bibliotek, framtaget av facebook.

Grundtanken är att vi som utvecklare skapar isolerade komponenter som vi sedan kan återanvända runt om i vår applikation.

En komponent innehåller alltså strukturen (html), stilens (css) och funktionaliteten (JavaScript) - för att tex beskriva hur en knapp ser ut och fungerar.

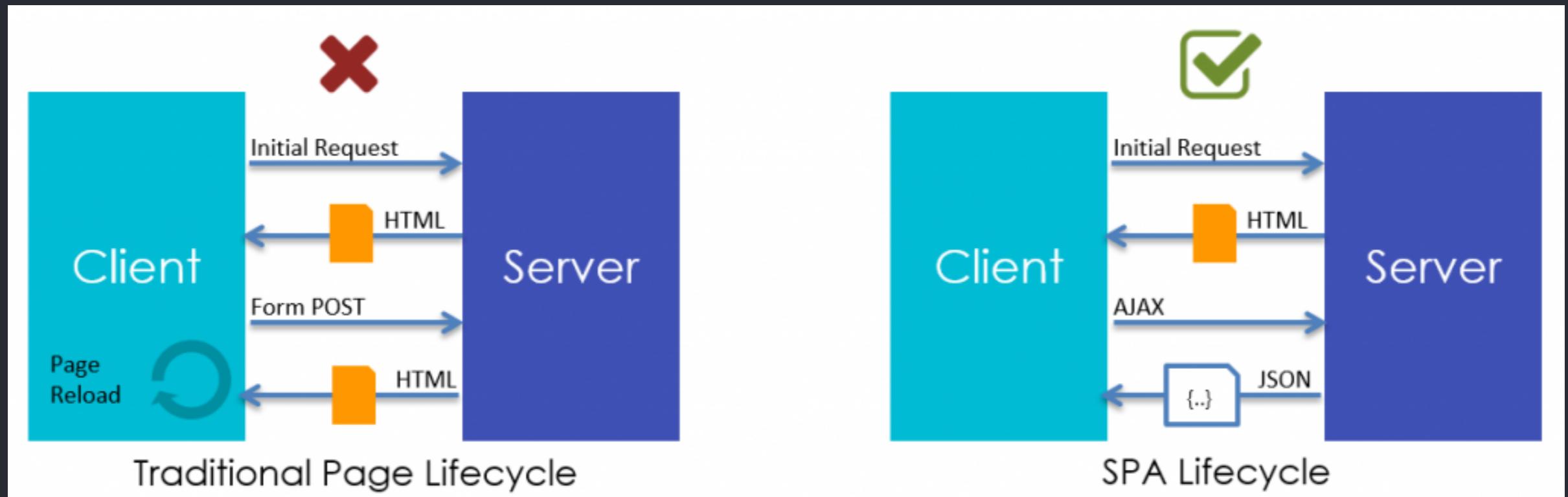
Om det sker ändringar i en komponent när applikationen körs, behöver vi inte tänka på hur DOM'en skall uppdateras utan det sköter React åt oss med hjälp av något kallas för en Virtuell DOM.

# Single Page Application

(SPA)

# React - SPA

Endast en html fil skickas från servern. Därefter skickas data(json) mellan klient och server och sidan uppdateras av klienten(js) baserat på datan.



# React - SPA

## Fördelar

Snabbare, inga reloads av sidan

Mindre utvecklingstid

Enklare att underhålla

Sidan beter sig mer som en applikation  
(animationer)

# React - SPA

## Nackdelar

Långsam initial laddning av sidan

Sämre SEO

Ingen webbläsarhistorik

Säkerhet (mer kod på klienten)

## Lösningar

Code Splitting

Server Side Rendering

React Router

Lägg inte känslig data i klientfilerna

# React - SPA

## Mer läsning

<https://rubygarage.org/blog/single-page-app-vs-multi-page-app>

<https://security.stackexchange.com/questions/41239/do-spa-applications-have-different-security-considerations-than-html5-sites>

<https://vuejsdevelopers.com/2018/04/09/single-page-app-seo/>

<https://resources.whitesourcesoftware.com/blog-whitesource/5-practical-tips-to-help-you-secure-your-single-page-application>

# Main Concepts (1-6)

We recommend that you don't skip topics because they build on each other.

**Vi kommer gå igenom utvalda delar av Main Concepts på föreläsningarna men det är sedan upp till er att själva läsa igenom allt på egen hand. Både under och utanför föreläsningstiden.**

# Tutorial (hemuppgift)

## Tip

This tutorial is designed for people who prefer to **learn by doing**. If you prefer learning concepts from the ground up, check out our [step-by-step guide](#). You might find this tutorial and the guide complementary to each other.

<https://reactjs.org/tutorial/tutorial.html>

# Hello World

# React - Hello World

**Det minsta React exemplet ser ut såhär**

```
ReactDOM.render(  
  <h1>Hello, world!</h1>,  
  document.getElementById('root')  
);
```

**En heading med texten "Hello world!" skrivas ut på sidan i elementet med Id't root.**

# Introducing JSX

# React - Introducing JSX

Ta en till på den är variabeldeklarationen

```
const element = <h1>Hello, world!</h1>;
```

Den är syntaxen är vaken en sträng eller html. Det vi tittar på är JSX som är en syntax utbyggnad av JavaScript.

# React - Introducing JSX

## Varför JSX?

Hela poängen med React är att bygga isolerade **Komponenter** som ansvarar för sin egna struktur (html), stil (css) och funktion (js). Därför finns JSX för att göra det möjligt att skriva html & css direkt i JavaScript.

React kräver inte att JSX används men det har sina fördelar och många utvecklare har hoppas på fåget!

# React - Introducing JSX

## Baka in JS uttryck i JSX

```
const name = 'Josh Perez';
const element = <h1>Hello, {name}</h1>;
```

```
ReactDOM.render(
  element,
  document.getElementById('root')
);
```

Om vi i JSX wrappar något  
i {} kan vi skriva JS  
uttryck som tex en  
variabel eller till och med  
"one-line if-satser"

{ someCondition ? variable1 : variable2 }

# React - Introducing JSX

## JSX är även det ett Uttryck

Det innebär att vi kan skriva JSX kod i if-satser och for-loopar.

```
function getGreeting(user) {  
  if (user) {  
    return <h1>Hello, {formatName(user)}!</h1>;  
  }  
  return <h1>Hello, Stranger.</h1>;  
}
```

# React - Introducing JSX

## Attribut i JSX

Skrivs som på samma sätt som i HTML

```
const element = <div tabIndex="0"></div>;
```

Attribut är alltid camelCase.

OBS: HTML attributet **class** heter **className** i JSX.

Kan även skrivas med "måsvingar" för att kunna skriva JS uttryck

```
const element = <img src={user.avatarUrl}></img>;
```

# React - Introducing JSX

## Barnelement med JSX

Skrivs som på samma sätt som i HTML

```
const element = (  
  <div>  
    <h1>Hello!</h1>  
    <h2>Good to see you here.</h2>  
  </div>  
) ;
```

Till skillnad mot HTML kan alla element vara självstängande

```
const element = <img src={user.avatarUrl} />;
```

# Rendering Elements

# React - Rendering Elements

## Den minsta byggstenen

```
const element = <h1>Hello, world</h1>;
```

**Element** är de minsta byggstenarna i React och är vad en **Komponent** består utav. (Det är HTML-ish koden)

**Note:**

One might confuse elements with a more widely known concept of “components”. We will introduce components in the [next section](#). Elements are what components are “made of”, and we encourage you to read this section before jumping ahead.

# React - Rendering Elements

## Virtual DOM

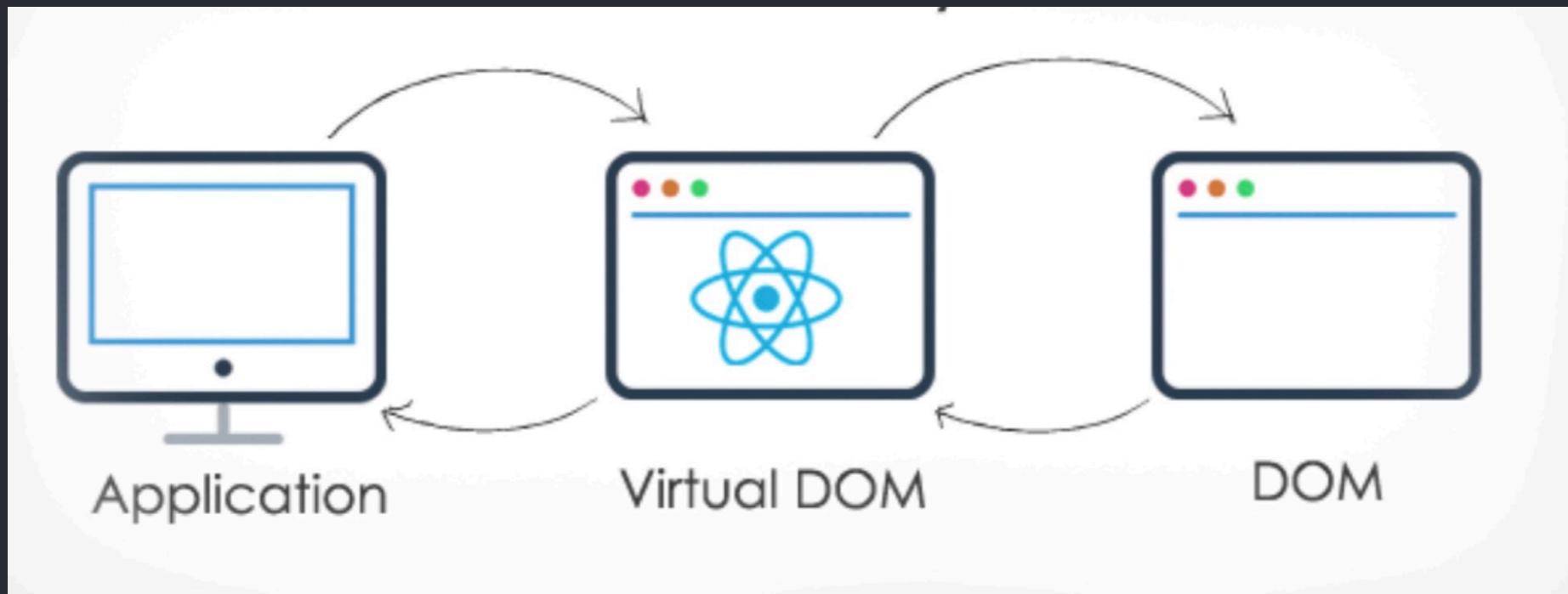
Inbyggt i react finns något som kallas för en virtuell DOM.

När en ändring görs i React-koden sköter React uppdateringen av den virtuella DOM'en och uppdaterar sedan även den riktiga DOM'en.

Detta gör att omrendering av sidan blir snabb och precis.

# React - Rendering Elements

## Virtual DOM



Om den riktiga DOM'en manipuleras (med tex JQuery) kan DOM'en och den Virtuella DOM'en hamna ur synk och skapa konstiga buggar. Försök därför att aldrig manipulera DOM'en manuellt i ett React projekt. **Do it the React way!**

# Components and Props

# React - Components & Props

Funktionskomponenter

Klasskomponenter

# React - Components & Props

## Funktionskomponenter

```
function Welcome(props) {  
  return <h1>Hello, {props.name}</h1>;  
}
```

Stor första bokstav

Returnerar ett **element**

# React - Components & Props

## Klasskomponenter

```
class Welcome extends React.Component {  
  render() { ←  
    return <h1>Hello, {this.props.name}</h1>;  
  }  
}
```

Ärver från `React.Component`

Implementation av  
livscykelmetod `render`

Returnerar ett **element**

# React - Components & Props

## Rendera en komponent & Props

```
function Welcome(props) {  
  return <h1>Hello, {props.name}</h1>;  
}
```

```
const element = <Welcome name="Sara" />;
```

Props är ett objekt som innehåller alla attribut som har skickats in till komponenten.

# React - Components & Props

## Ett komplett exempel

```
function Welcome(props) {  
  return <h1>Hello, {props.name}</h1>;  
}
```

```
const element = <Welcome name="Sara" />;  
ReactDOM.render(  
  element,  
  document.getElementById('root'))  
;
```

Anropet till **ReactDOM.render** görs  
bara en gång och "kopplar"  
react applikation till DOM'en.

# State and Lifecycle

# React - State and Lifecycle

## State

Alla klasskomponenter har ett internt state. När statet sätts triggas en omrendering av komponenten och vi kan på så sätt uppdatera vad användaren ser.

En komponents states uppdateras med metoden **setState**.

Låt oss kolla på en enkel komponent som uppdaterar sitt state varje sekund för att visa hur mycket klockan är.

Men först tar vi och går egenom livscykel metoderna eftersom dom ofta hänger ihop!

# React - State and Lifecycle

## Lifecycles

(Constructor)

Render

**ComponentDidMount**

**ComponentDidUpdate**

**ComponentWillUnmount**

Det här är de mest grundläggande livscykelmетодer och är vad vi kommer fokusera på under den här kursen.

**Render** anropas alltid före **ComponentDidMount & ComponentDidUpdate**

(Livscykelmetoderna kan bara användas i klasskomponenter)

# React - State and Lifecycle

Lifecycles 

(Constructor)

Render

ComponentDidMount

ComponentDidUpdate

ComponentWillUnmount

Deprikerade 

ComponentWillMount

ComponentWillReceiveProps

ComponentWillUpdate

Ersättare 

GetDerivedStateFromProps (static)

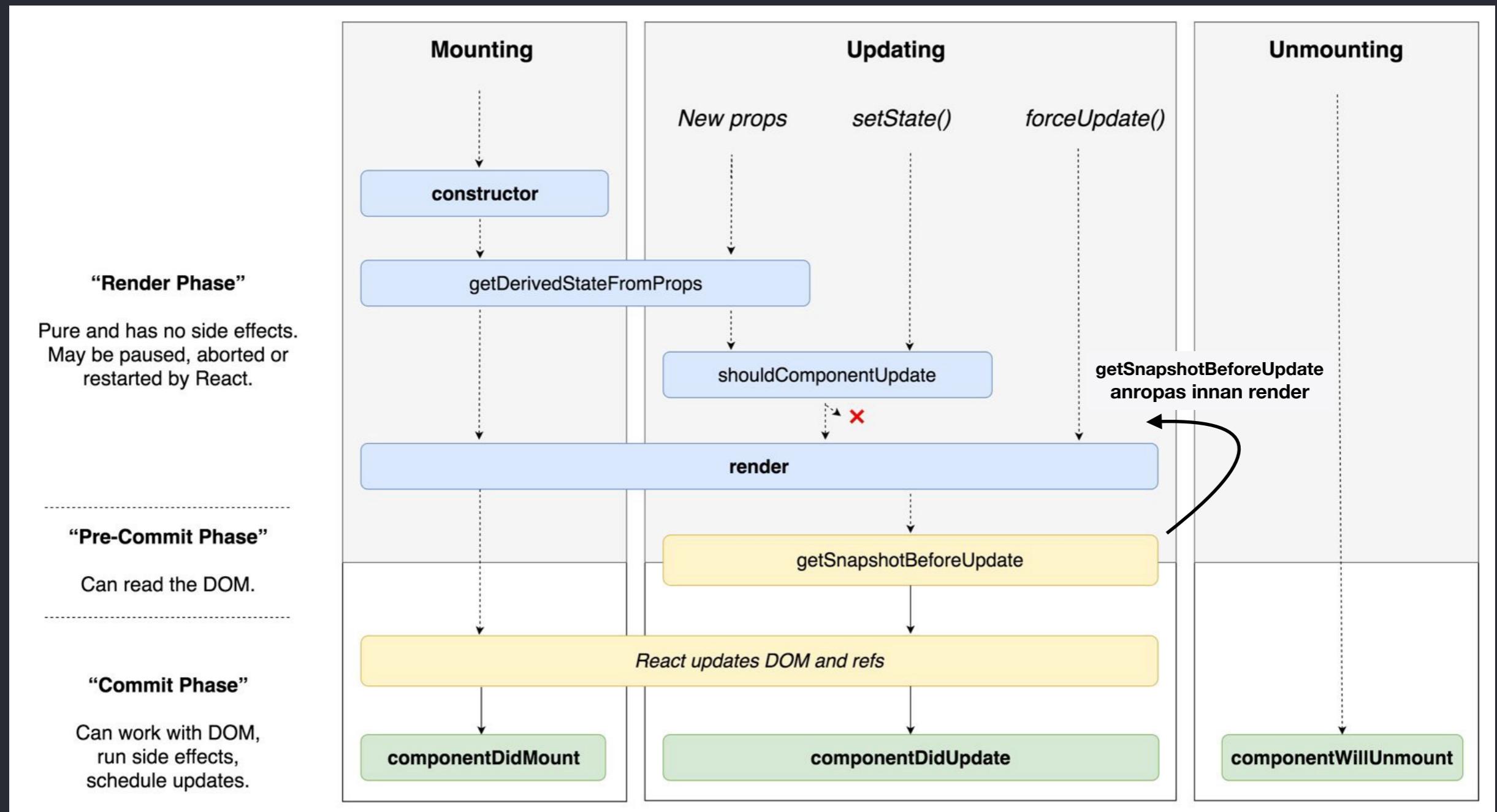
GetSnapshotBeforeUpdate (static)

Läs mer här

<https://reactjs.org/blog/2018/03/27/update-on-async-rendering.html>

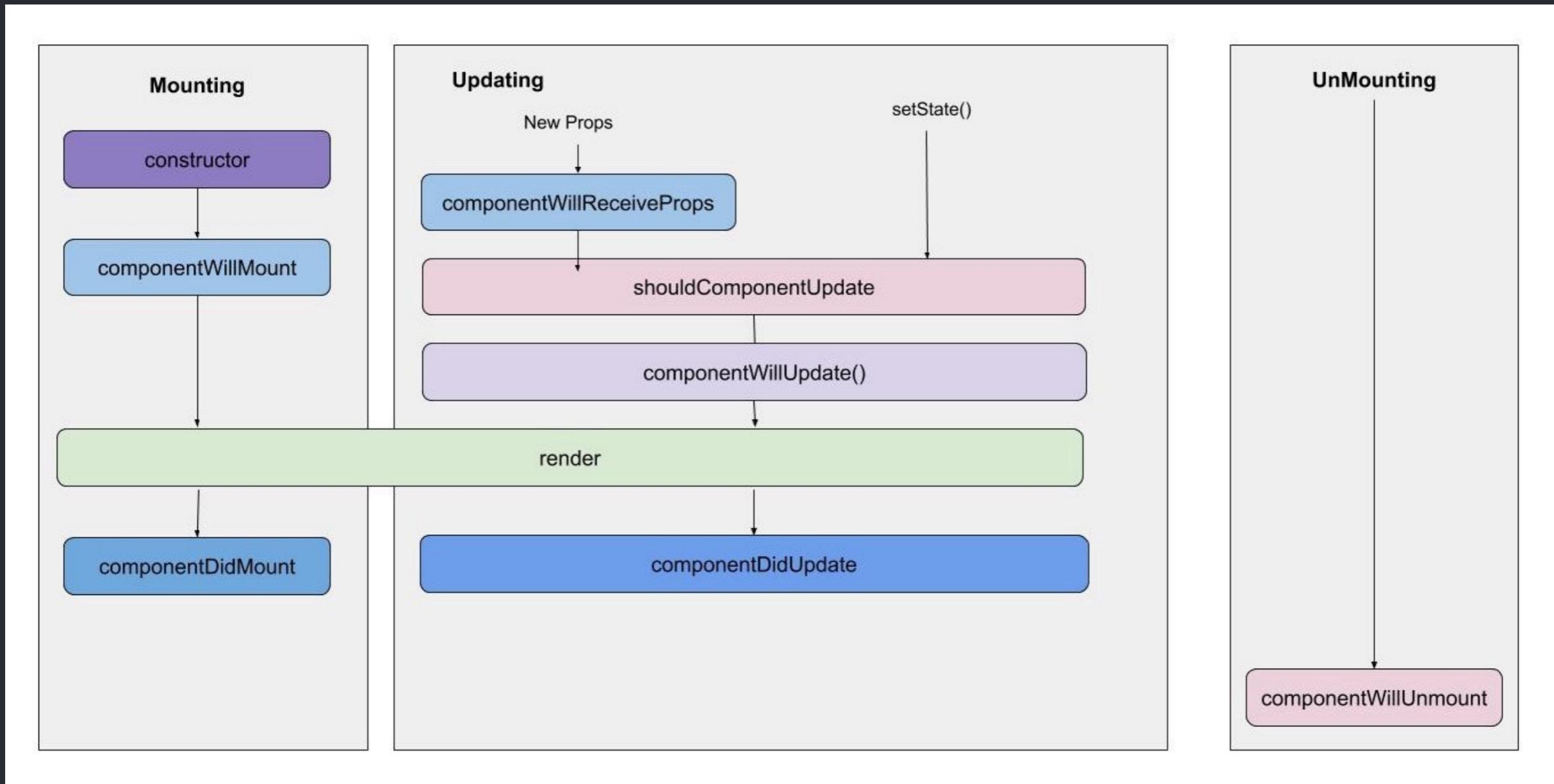
# React - State and Lifecycle

## Lifecycle (post 16.3)



# React - State and Lifecycle

## Lifecycle (pre 16.3)



# React - State and Lifecycle

## Clock component

Okej, låt oss nu ta en titt på hur vi kan bygga en komponent som visa hur mycket klockan är.

# React - State and Lifecycle

## Clock component - read from state

```
class Clock extends React.Component {  
  render() {  
    return (  
      <div>  
        <h1>Hello, world!</h1>  
        <h2>It is {this.state.date.toLocaleTimeString()}.</h2>  
      </div>  
    );  
  }  
}
```

Hämta tiden från state



# React - State and Lifecycle

## Clock component - init state

```
class Clock extends React.Component {  
  constructor(props) {  
    super(props); ← Notera super(props)  
    this.state = {date: new Date()};  
  }  
  
  render() {  
    return (  
      <div>  
        <h1>Hello, world!</h1>  
        <h2>It is {this.state.date.toLocaleTimeString()}.</h2>  
      </div>  
    );  
  }  
}
```

Initiera statet när komponenten skapas

# React - State and Lifecycle

## Clock component - update state

```
class Clock extends React.Component {  
  constructor(props) {  
    super(props);  
    this.state = {date: new Date()};  
  }  
  
  componentDidMount() {  
  }  
  
  componentWillUnmount() {  
  }  
  
  render() {  
    return (  
      <div>  
        <h1>Hello, world!</h1>  
        <h2>It is {this.state.date.toLocaleTimeString()}.</h2>  
      </div>  
    );  
  }  
}
```

Sedan lägger vi till två livscykelmetoder för att skapa och ta bort ett interval med 1 sekunds interval som uppdaterar tiden.

# React - State and Lifecycle

## Clock component - update state

```
componentDidMount() {  
  this.timerID = setInterval(  
    () => this.tick(),  
    1000  
  );  
}  
}  
  
componentWillUnmount() {  
  clearInterval(this.timerID);  
}
```

I componentDidMount skapar vi en timer med **setInterval**, där metoden tick anropas varje sekund.

```
tick() {  
  this.setState({  
    date: new Date()  
  });  
}
```

I componentWillUnmount tar vi bort timern eftersom klockan inte ska visas längre.

```
tick() {  
  this.setState({  
    date: new Date()  
  });  
}
```

I tick anropar vi react-funktionen **setState( )** för att spara en ny tid i statet som i sin tur trigger en omrendering av komponenten.

(setState tar alltid in ett objekt)

```
class Clock extends React.Component {
  constructor(props) {
    super(props);
    this.state = {date: new Date()};
  }
  componentDidMount() {
    this.timerID = setInterval(
      () => this.tick(),
      1000
    );
  }
  componentWillUnmount() {
    clearInterval(this.timerID);
  }
  tick() {
    this.setState({
      date: new Date()
    });
  }
  render() {
    return (
      <div>
        <h1>Hello, world!</h1>
        <h2>It is {this.state.date.toLocaleTimeString()}.</h2>
      </div>
    );
  }
}
```

# React - State and Lifecycle

## Clock component - full example

Notera lambda funktionen här  
för att this ska få rätt kontext



# React - State and Lifecycle

## Clock component

Vi måste även rendera komponenten på vår sida med ReactDOM.render för att kunna se den live-uppdaterade klockan!

```
ReactDOM.render(  
  <Clock />,  
  document.getElementById('root')  
)
```

# Handling Events

# React - Handling Events

Hantering av event's i React-element är väldigt likt hur events hanteras av DOM-element.

HTML

```
<button onclick="activateLasers()">  
  Activate Lasers  
</button>
```

React

```
<button onClick={activateLasers}>  
  Activate Lasers  
</button>
```

javascript uttryck

camelCase



# React - Handling Events

## Toggle Component - full example

```
class Toggle extends React.Component {  
  constructor(props) {  
    super(props);  
    this.state = {isToggleOn: true};  
  
    // This binding is necessary to make `this` work in the callback  
    this.handleClick = this.handleClick.bind(this);  
  }  
  
  handleClick() {  
    this.setState(state => ({  
      isToggleOn: !state.isToggleOn  
    }));  
  }  
  
  render() {  
    return (  
      <button onClick={this.handleClick}>  
        {this.state.isToggleOn ? 'ON' : 'OFF'}  
      </button>  
    );  
  }  
}
```

Om en event-funktion inte  
är en implementerad som  
en lamda funktion måste  
man binda **this**.

# React - Handling Events

## Skicka argument till eventhanteraren

```
<button onClick={(e) => this.deleteRow(id, e)}>Delete Row</button>
<button onClick={this.deleteRow.bind(this, id)}>Delete Row</button>
```

Här får man dock vara lite försiktig så att man inte skapar några prestandaproblem eftersom det kan orsaka onödiga omrenderingar.

# React - Handling Events

## Lamda funktion

```
class LoggingButton extends React.Component {  
  // This syntax ensures `this` is bound within handleClick.  
  // Warning: this is *experimental* syntax.  
  handleClick = () => {  
    console.log('this is:', this);  
  }  
  
  render() {  
    return (  
      <button onClick={this.handleClick}>  
        Click me  
      </button>  
    );  
  }  
}
```

# Läsanvisningar

## React Main Concepts

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

### Kapitel.

1. Hello World
2. Introducing JSX
3. Rendering Elements
4. Components and Props
5. State and Lifecycle
6. Handling Events

# Nästa lektion

## React Playground

Hur man konfigurerar upp ett projekt på egen hand

# Resten av dagen

## React Main Concepts

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

## Kapitel. 1-6 + Tutorial

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

Är du redan klar? Gör Tic Tac Toe spelet med timetravel som du hittar på Reacts tutorial sida!

# Tack