UNIVERSITAS MIKROSKIL | Fakultas INFORMATIKA

TI2104 – Pengembangan Aplikasi Web

Lifecycle dan Hooks

- Pengenalan lifecycle dalam komponen react js
- Pengenalan hooks dalam react js
- Pengunaan useEffect dan useState

Strata - 1

Teknologi Informasi

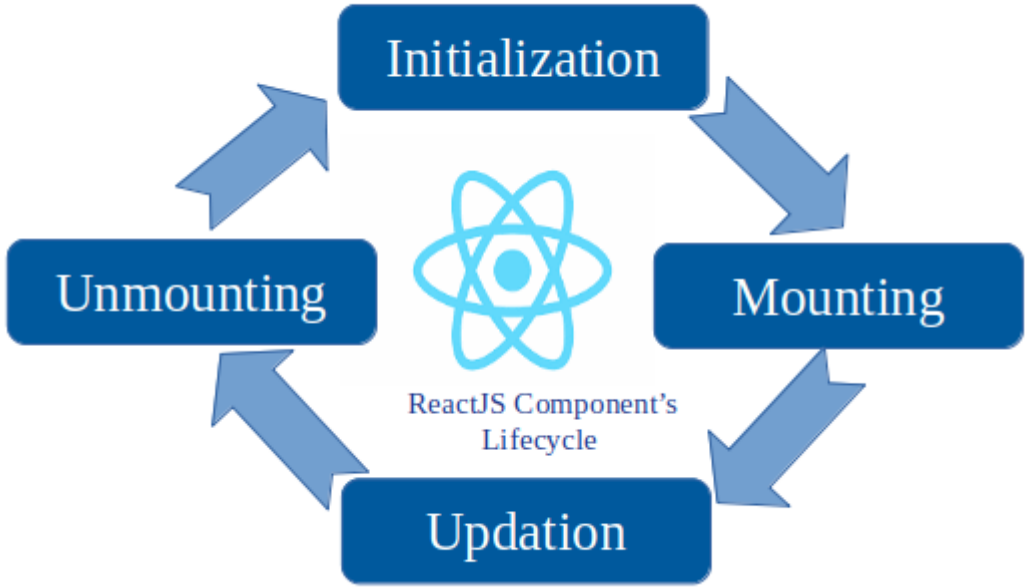
React Lifecycle

Chapter 1

UNIVERSITAS MIKROSKIL

Lifecycle dalam ReactJs

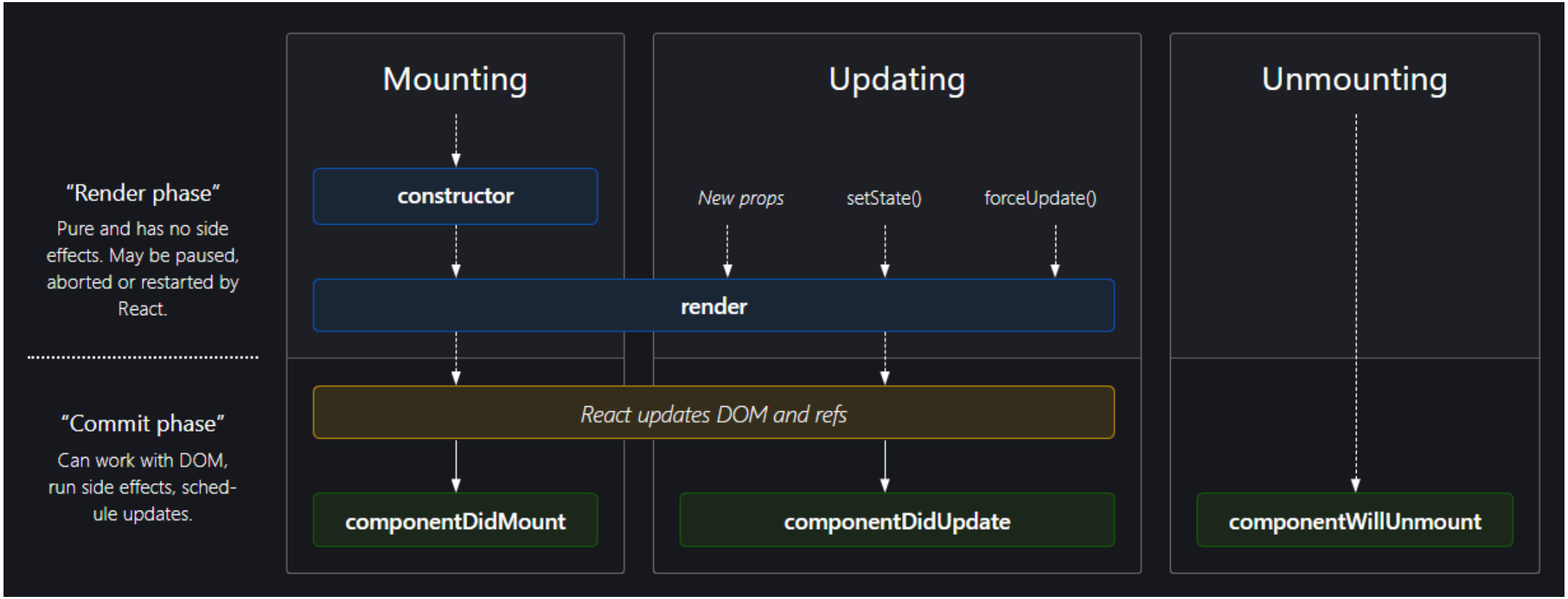
- Lifecycle komponen dalam React merujuk pada **serangkaian peristiwa** atau **tahapan** yang dialami oleh sebuah komponen di dalam aplikasi React. Seperti : ketika sebuah komponen **dibuat**, **digunakan, diperbarui, atau dihapus dari DOM**.
- Hal jadi Ini memungkinkan pengembang untuk **mengontrol perilaku komponen pada titik-titik tertentu** dalam siklus aplikasi.



Mengapa Lifecycle Penting dalam React?

- **Kontrol Perilaku**: Dengan lifecycle, pengembang dapat mengontrol perilaku komponen saat dibuat, diperbarui, atau dihapus, sehingga memungkinkan penggunaan sumber daya yang efisien.
- **Manajemen Sumber Daya Eksternal**: lifecycle memungkinkan komponen untuk berinteraksi dengan sumber daya eksternal seperti permintaan jaringan, aliran data, atau pembersihan sumber daya ketika komponen tidak lagi diperlukan.
- **Optimisasi Kinerja**: Dengan memanfaatkan method lifecycle, pengembang dapat melakukan optimisasi kinerja, seperti membatasi pembaruan yang tidak perlu dengan menggunakan shouldComponentUpdate.
- **Integrasi Ekosistem React**: lifecycle memungkinkan integrasi yang lebih baik dengan komponen dan pustaka lain dalam ekosistem React, seperti React Router atau Redux.

Tahapan React Lifecycle



https://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/

React Lifecycle (Mounting)

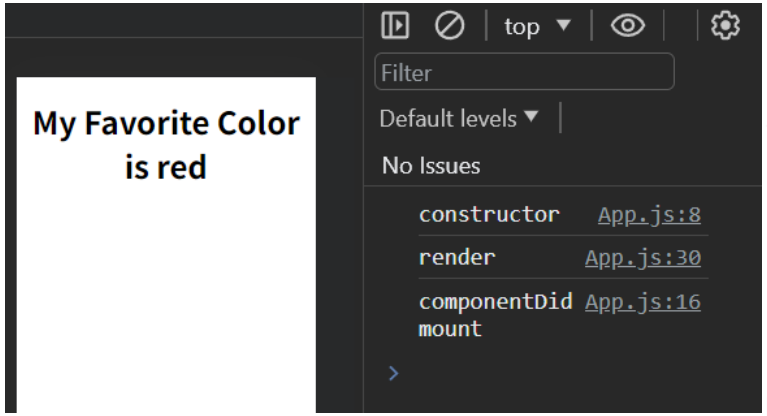
Mounting (Pemasangan) : Tahap ini **terjadi ketika komponen pertama kali dibuat dan ditambahkan ke DOM**. Beberapa metode siklus hidup yang terkait dengan tahap ini antara lain:

- **constructor**: Metode ini digunakan untuk **inisialisasi awal dan mengatur state**.
- **render**: Metode ini menghasilkan **tampilan komponen**.
- **componentDidMount**: Metode ini **dipanggil setelah komponen berhasil dipasang pada DOM**, biasanya digunakan untuk memulai permintaan data atau tugas asinkron lainnya.

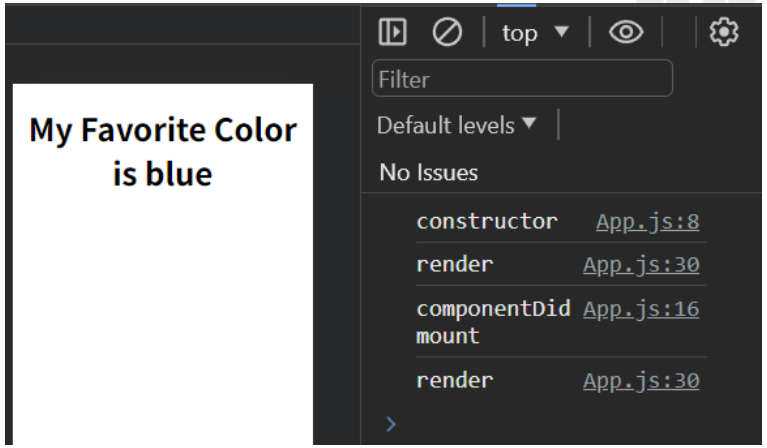
UNIVERSITAS MIKROSKIL

React Lifecycle (Mounting)

Component pertama kali dijalankan



Setelah 3 detik dijalankan




```
class MyComponent extends React.Component {
  /*
  method constructor akan dijalankan ketika
  component class di instance*/
  constructor(props) {
    console.log("constructor");
    super(props);
    this.state = { favoritecolor: "red" };
  }
  /*
  - componentDidmount akan dijalankan ketika component
    selesai digambarkan.*/
  componentDidMount() {
    console.log("componentDidmount");
    /*set timeout digunakan untuk meberikan jeda
    saat program hendak dijalankan*/
    setTimeout(() => {
      /*setState akan dijalankan setelah 3000ms (3s),
      setelah 3s, favorite color akan diubah jadi blue
      dan component akan di render ulang*/
      this.setState({ favoritecolor: "blue" });
    }, 3000);
  }

  /*
  - method render akan dijalankan ketika component
    akan di gambarkan.
  - dan akan di panggil jika ada perubahan state*/
  render() {
    console.log("render");
    return <h1>My Favorite Color is {this.state.favoritecolor}</h1>;
  }
}
```

React Lifecycle (Updating)

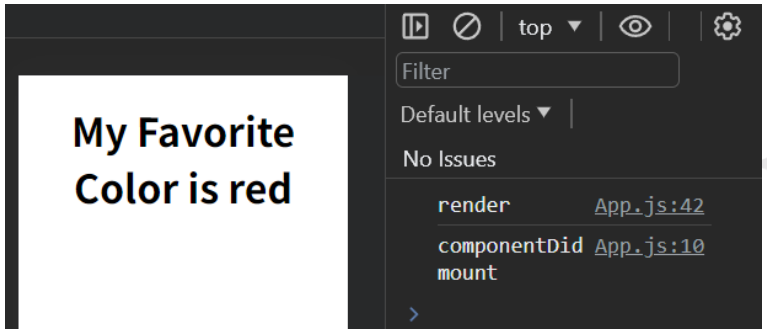
Updating (Pembaruan) : Tahap ini **terjadi ketika komponen menerima pembaruan pada props atau state-nya**. Beberapa metode siklus hidup yang terkait dengan tahap ini antara lain:

- **shouldComponentUpdate** : Metode ini memungkinkan pengembang untuk **memutuskan apakah komponen harus diperbarui atau tidak** berdasarkan perubahan props atau state.
- **render**: Seperti pada tahap mounting, metode render digunakan untuk menghasilkan tampilan yang diperbarui.
- **componentDidUpdate**: Metode ini **dipanggil setelah pembaruan komponen selesai**, biasanya digunakan untuk menangani efek samping atau tindakan pasca-pembaruan.

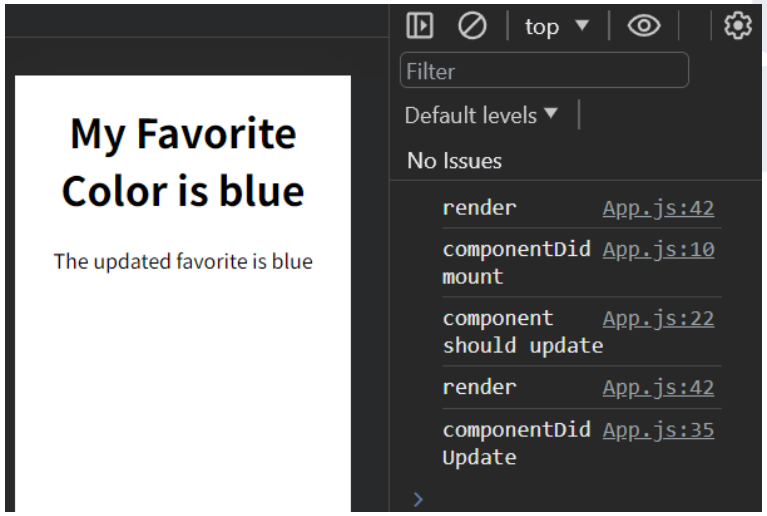
UNIVERSITAS MIKROSKIL

React Lifecycle (Updating)

Component pertama kali dijalankan



Setelah 3 detik componentDidMount dijalankan




```
class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.state = { favoritecolor: "red" };
  }

  componentDidMount() {
    console.log("componentDidmount");
    setTimeout(() => {
      this.setState({ favoritecolor: "blue" });
    }, 3000);
  }

  /*method ini mengembalikan nilai boolean
  yang menentukan apakah react melanjutkan
  rendering atau tidak, nilai defaultnya adalah
  true
  */
  shouldComponentUpdate() {
    console.log("component should update");
    return true;
  }

  /*
  - dipanggil setelah component di perbarui pada DOM.
  - pada code ini state awal kita buat red, setelah di
    render 3 detik state akan diubah menjadi blue.
  - karena state berubah otomatis render jalan
  - setelah render jalan dan terdapat perubahan pada component
    makan componentDidmount di panggil
  */
  componentDidUpdate() {
    console.log("componentDidUpdate");
    /*setelah state di update*/
    document.getElementById("mydiv").innerHTML =
      "The updated favorite is " + this.state.favoritecolor;
  }

  render() {
    console.log("render");
    return (
      <div>
        <h1>My Favorite Color is {this.state.favoritecolor}</h1>
        <div id="mydiv"></div>
      </div>
    );
  }
}
```

React Lifecycle (Unmounting)

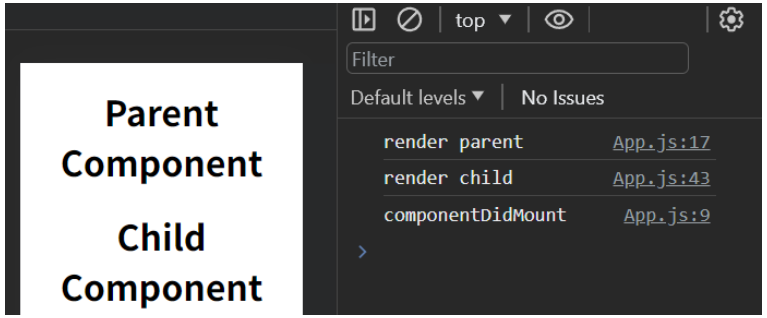
Unmounting (Pelepasan) : Tahap ini **terjadi ketika sebuah komponen akan dihapus dari DOM**.

- Metode lifecycle yang terkait dengan tahap ini adalah **componentWillUnmount**.

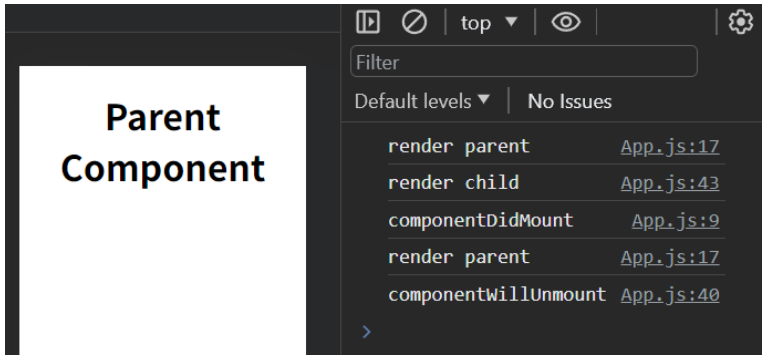
UNIVERSITAS MIKROSKIL

React Lifecycle (Unmounting)

Component pertama kali dijalankan



Setelah 3 detik componentDidMount berjalan



Child component

```
class ChildComponent extends React.Component {
  /*componentWillUnmount akan terpanggil bila
  componentnya terhapus dari DOM*/
  componentWillUnmount() {
    console.log("componentWillUnmount");
  }
  render() {
    console.log("render child");
    return <h1>Child Component</h1>;
  }
}
```

Parent Component

```
class ParentComponent extends React.Component {
  constructor(props) {
    super(props);
    this.state = { show: true };
  }
  componentDidMount() {
    console.log("componentDidMount");
    /*setelah 3 detik code ini akan mengubah
    state show menjadi true*/
    setTimeout(() => {
      this.setState({ show: false });
    }, 3000);
  }
  render() {
    console.log("render parent");
    let MyComponent;
    /*cek apakah show true,
    - jika true ComponentChild di render
    - jika false ComponentChild di remove dari DOM
      hal ini akan mentrigger method componentWillUnmpunt
      dijalankan*/
    if (this.state.show) {
      MyComponent = <ChildComponent />;
    }
    return (
      <div>
        <h1>Parent Component</h1>
        {MyComponent}
      </div>
    );
  }
}
```

(?) Cari Tahu

- Mengapa pemahaman lifecycle komponen React penting dalam pengembangan aplikasi React?
- Apakah ada cara lain untuk menerapkan lifecycle pada React ?

UNIVERSITAS MIKROSKIL

React Hooks

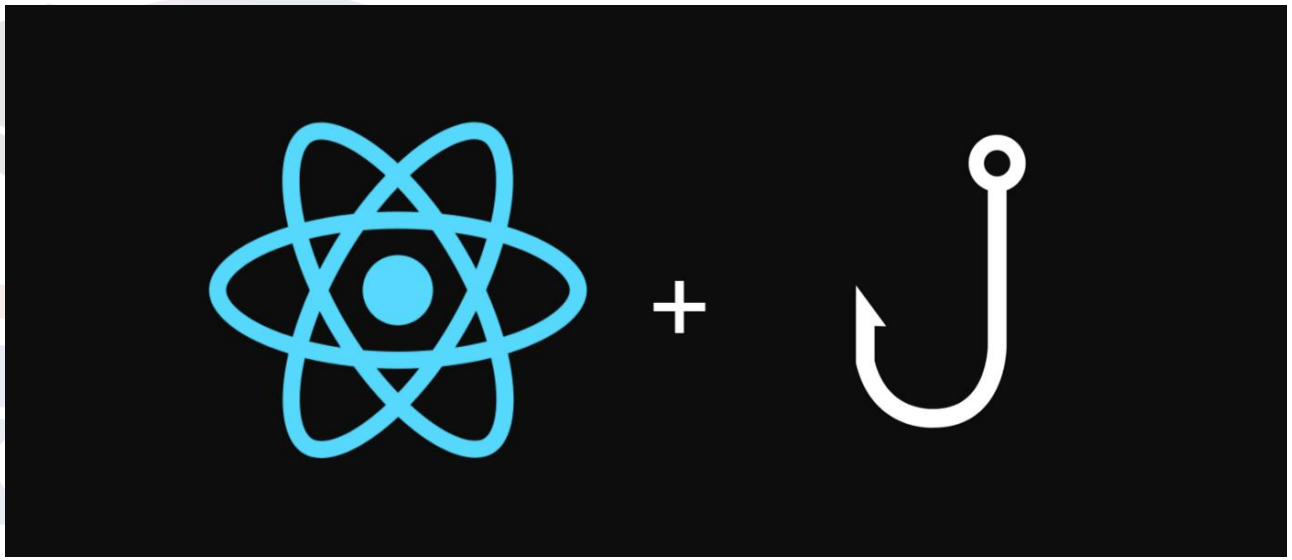
Chapter 2

UNIVERSITAS MIKROSKIL

Pengenalan React Hooks

React Hooks adalah **fitur baru** dalam React yang diperkenalkan dalam versi **React 16.8**. Mereka adalah **fungsi khusus** yang memungkinkan pengembang untuk **"mengaitkan" perilaku React state dan fitur lainnya ke dalam komponen berbasis fungsi**, yang sebelumnya hanya tersedia dalam komponen berbasis kelas.

Dengan React Hooks, kita dapat menggunakan state, efek samping, dan konteks dalam komponen berbasis fungsi **tanpa perlu mengubah komponen menjadi kelas**.



Keunggulan Penggunaan React Hooks

- **Sederhana dan Mudah Dimengerti**: React Hooks membuat kode lebih sederhana dan lebih mudah dimengerti daripada komponen berbasis kelas. Kita tidak perlu lagi memahami konsep this atau metode lifecycle.
- **Reuseable Logic**: Kita dapat mengisolasi logika yang dapat digunakan kembali dalam bentuk hook Custom. Ini memungkinkan Anda untuk menggabungkan dan menggunakan logika tersebut di berbagai komponen.
- **Peningkatan Kinerja**: React Hooks dapat membantu kita mengoptimalkan kinerja komponen kita dengan menghindari pembaruan yang tidak perlu.
- **Menggantikan Lifecycle Methods**: kita dapat menggantikan lifecycle method yang rumit dalam komponen berbasis kelas dengan penggunaan hook seperti useEffect.
- **Kode yang Lebih Bersih**: Hooks membantu dalam menghasilkan kode yang lebih bersih dan lebih terorganisir dengan mengelola state dan efek samping dalam satu tempat.

UNIVERSITAS MIKROSKIL

Perbedaan React Hooks dan Class Components

React Hooks	Classes
Used in functional components in React	Used in class based components in React
Does not require the declaration of any kind of constructor	Declaration of constructor has to be made inside of the class component.
There is no need of using *this* keyword in state declaration or modification	*this* keyword is used in state declaration i.e. this.state and in modification - this.setState()
Easier to use because of the useState() functionality	No specify function that helps us access the state and its corresponding setState variable.
React Hooks can help in the implementation of Redux and context API	Due to the long setup of state declarations, class states generally not preferred.

Hooks yang umum digunakan

- **useState**: Digunakan untuk mengelola state dalam komponen berbasis fungsi. Ini memungkinkan Anda untuk menyimpan dan memperbarui data state dalam komponen.
- **useEffect**: Digunakan untuk menangani efek samping dalam komponen berbasis fungsi. Ini memungkinkan Anda untuk melakukan tindakan setelah rendering komponen atau ketika komponen diubah.
- **useContext**: Digunakan untuk mengakses konteks dalam komponen berbasis fungsi. Konteks adalah cara untuk berbagi data global antara komponen dalam pohon komponen.
- **useRef**: Digunakan untuk mengelola referensi ke elemen DOM atau nilai lain dalam komponen. Ini sering digunakan untuk mengakses elemen DOM secara langsung.
- **useReducer**: Mirip dengan useState, digunakan untuk mengelola state dalam komponen berbasis fungsi. Namun, ini lebih cocok untuk mengelola state yang kompleks yang melibatkan perubahan yang lebih kompleks.
- **useCallback**: Digunakan untuk menghindari pembuatan ulang fungsi dalam komponen yang mengakses prop. Ini dapat meningkatkan kinerja komponen.
- **useMemo**: Digunakan untuk menghindari perhitungan yang mahal secara berulang dalam komponen. Ini memungkinkan Anda untuk menghitung nilai hanya saat prop atau state yang relevan berubah.

React useState Hook (Initialize state)

- **Inisialisasi useState**
- Kita menginisialisasi state dengan memanggil useState di komponen fungsi.
- useState menerima initial value dan mengembalikan dua nilai:
- Index-0 penampung nilai state.
- Index-1 fungsi untuk memperbarui nilai state.

```
1  import React, { useState } from "react";
2
3  const MyComponent = () => {
4    const [color, setColor] = useState("");
5  };
6
```

Perhatikan bahwa kita melakukan destructuring nilai yang dikembalikan dari useState.

- Nilai pertama color adalah state saat ini.
- Nilai kedua, setColor, adalah fungsi yang digunakan untuk memperbarui nilai state.

Nama tersebut adalah variable, jadi sebenarnya kita bebas mementukan nama apapun

Terkahir kita mengatur state awal color dalah string kosong : useState(“”)

React useState Hooks (read and update state)

Untuk memperbarui nilai state, kita menggunan state update function, yang sudah di defenisikan di awal.

Kita tidak boleh memperbarui status secara langsung. Contoh: color = "merah" tidak diperbolehkan.

Sebelum tekan update

My favorite color is red!

update

Setelah tekan update

My favorite color is blue!

update

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

const MyComponent = () => {
  /*inisialisasi awal state, dimana color sebagai
    penampung data, dan setColor untuk merubah nilai
    nilai color pada awal di set 'red'*/
  const [color, setColor] = useState("red");

  /*funsgi yang akan mengubah nilai state
   yang akan di jalankan ketika button update
   di click */
  const updateButtonHandler = () => {
    setColor("blue");
  };

  return (
    <>
      <h1>My favorite color is {color}!</h1>
      <button type="button" onClick={updateButtonHandler}>
        update
      </button>
    </>
  );
};
```

React UseEffect Hook

UseEffect Hook memungkinkan kita menangani efek samping pada komponen berbasis fungsi.

Beberapa contoh efek samping adalah: **mengambil data, memperbarui DOM secara langsung, dan pengatur waktu**.

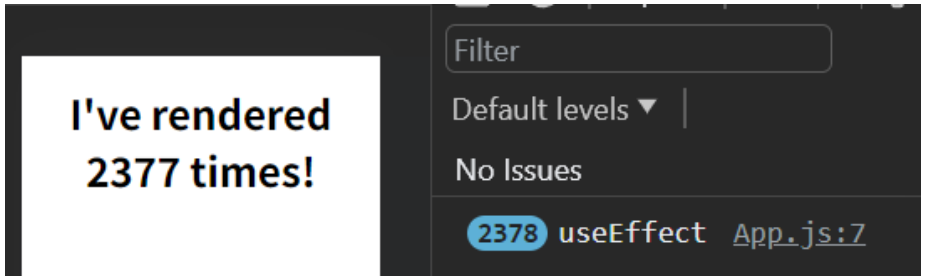
useEffect menerima **dua argumen**.

Argumen pertama adalah fungsi atau callback, Argumen kedua adalah dependency (optional).

```
useEffect(<function>, <dependency>)
```

useEffect pada Program ini akan dijalankan terus menerus

```jsx
import React, { useState, useEffect } from "react";

const MyComponent = () => {
  const [count, setCount] = useState(0);

  useEffect(() => {
    console.log("useEffect");
    setTimeout(() => {
      setCount((count) => count + 1);
    }, 1000);
  });

  return <h1>I've rendered {count} times!</h1>;
};
```



React UseEffect Hook

Pada contoh sebelumnya useEffect akan di jalankan terus menerus.

Untuk menangani hal tersebut kita perlu menambahkan argument kedua yaitu dependency.

Pemberian dependency akan membuat perilaku useEffect seperti keterangan pada gambar di samping

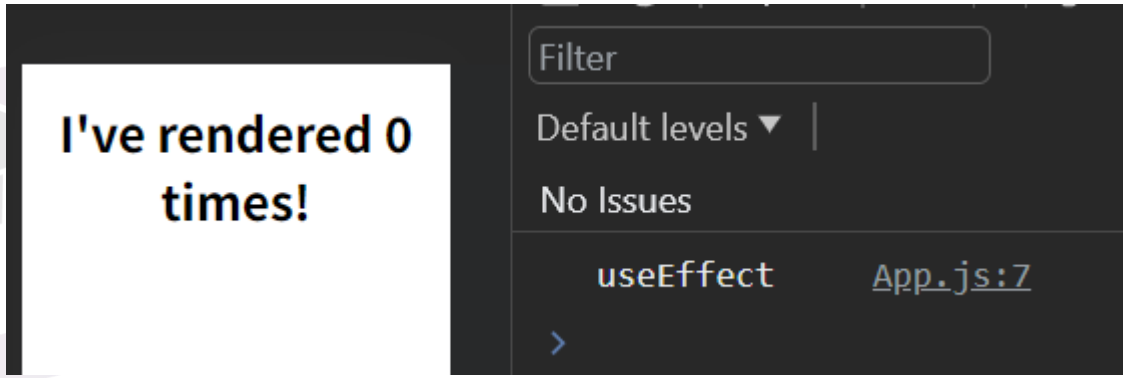
```
1  //1. Tanpa dependency
2  useEffect(()=>{
3    //Dijalankan setiap kali render
4  });
5
6  //2. dependency array kosong
7  useEffect(()=>{
8    //Dijalankan sekali saat pertama kali di render
9  },[]);
10
11 //3. dependency berupa array berisi state atau props
12 useEffect(()=>{
13   //Dijalankan sekali saat pertama kali di render
14   /*Dan akan dijalankan kembali saat nilai dari
15     berubah*/
16 }, [props, state]);
```

useEffect dengan dependency (array kosong)

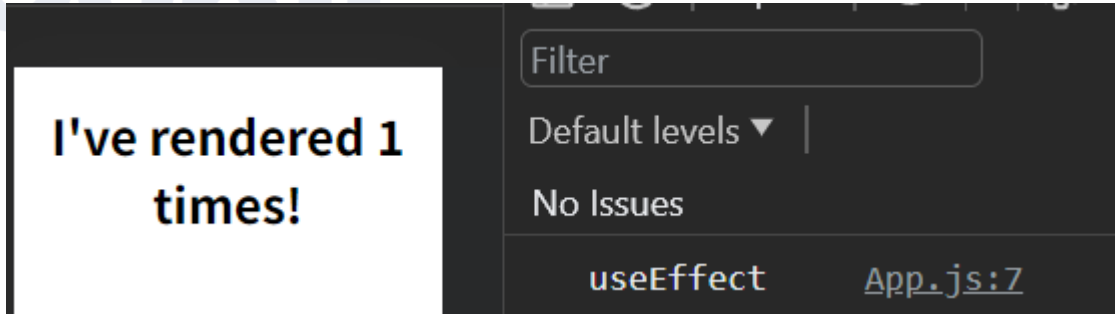
Hal ini membuat useEffect sama seperti method componentDidMount()

```jsx
const MyComponent = () => {
  const [count, setCount] = useState(0);

  useEffect(() => {
    console.log("useEffect");
    setTimeout(() => {
      setCount((count) => count + 1);
    }, 3000);
  }, []);

  return <h1>I've rendered {count} times!</h1>;
};
```

Awal di jalankan, useEffect di panggil sekali



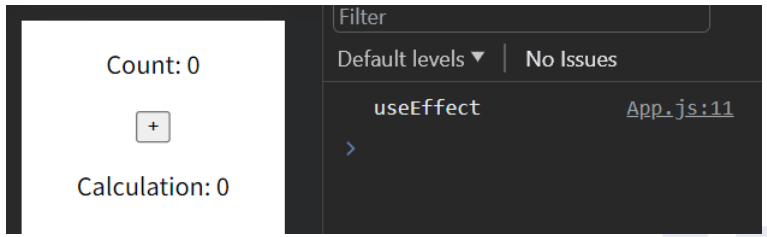
Setelah 3 detik useEffect di panggil, nilai state di ubah



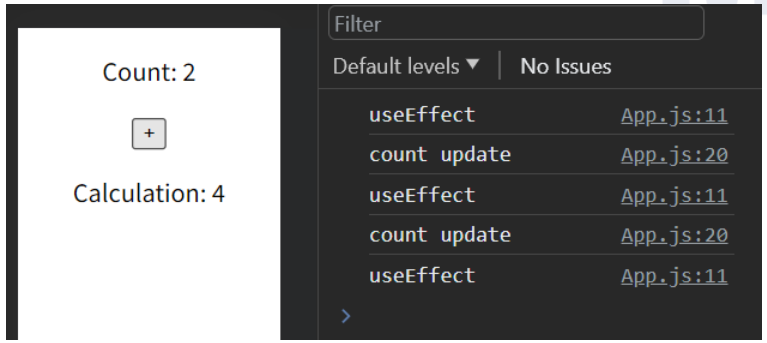
useEffect dengan dependency (array prop dan state)

Berikut adalah contoh useEffect Hook yang bergantung pada variabel. Jika variabel count diperbarui, useEffect akan berjalan Kembali.

Pertama kali component dijalankan



Setelah 2 kali tekan tombol +




```jsx
const MyComponent = () => {
  /**inisialisasi state awal */
  const [count, setCount] = useState(0);
  const [calculation, setCalculation] = useState(0);

  /*use Effect akan di panggil sekali
  karena punya dependecy array*/
  useEffect(() => {
    console.log("useEffect");
    setCalculation(() => count * 2);
  }, [count]);
  /*namun karena array tidak kosong, maka
    ketika count berubah useEffect akan
    dijalankan kembali*/

  //fungsi untuk  mengubah nilai count
  const increament = () => {
    console.log("count update");
    setCount((prevCount) => prevCount + 1);
  };

  return (
    <>
      <p>Count: {count}</p>
      <button onClick={increament}>+</button>
      <p>Calculation: {calculation}</p>
    </>
  );
};
```

useEffect dengan Cleanup

Beberapa useEffect **memerlukan pembersihan untuk mengurangi kebocoran memori.**

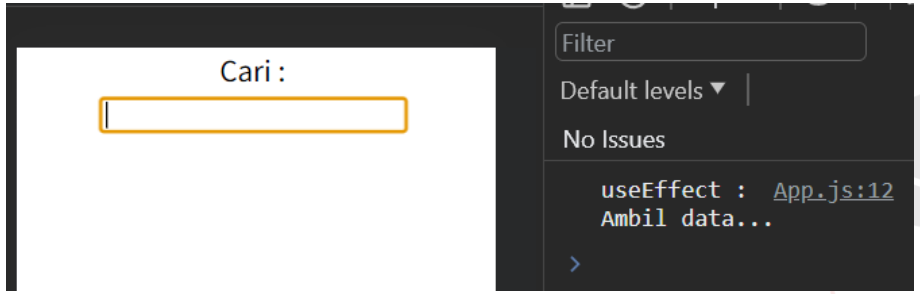
Timeouts, subscriptions, event listender, dan efek lainnya yang tidak lagi diperlukan harus dibuang. Kita bisa melakukan ini dengan menyertakan fungsi return di akhir useEffect Hook.

Penerapan useEffect dengan cleanup **membantu mencegah memory leaks dan masalah lain yang terkait dengan efek samping yang tidak dihentikan secara tepat waktu**. Hal ini juga merupakan **praktik baik dalam pengembangan React** yang memastikan aplikasi Anda berjalan dengan baik dalam berbagai skenario.

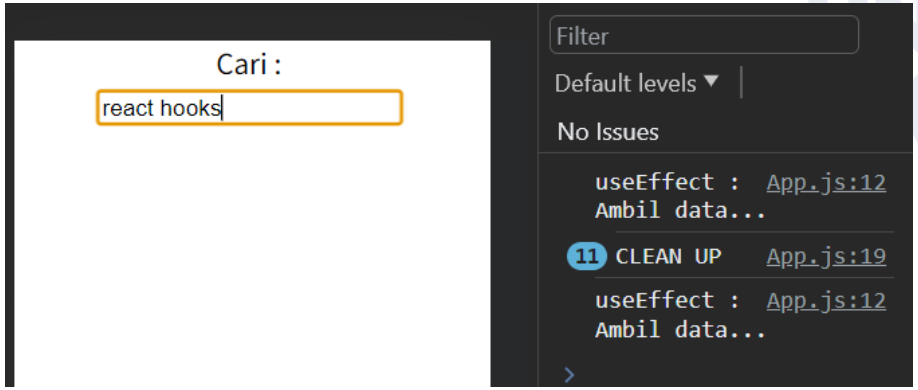
```
useEffect(() => {
  // Efek samping di sini
  // ...
  // Cleanup (bersihkan efek samping di sini)
  return () => {
    // Membersihkan efek samping
    // ...
  };
}, [dependencies]);
```

useEffect dengan Cleanup

Pertama kali component dijalankan



Setelah user selesai mengetik




```jsx
const MyComponent = () => {
  const [keyword, setKeyword] = useState("");
  const [data, setData] = useState([]);

  useEffect(() => {
    //untuk mengambil hasil pencarian
    //kita perlu menunggu user berhenti mengetik
    //selama 1 detik
    const identifier = setTimeout(() => {
      console.log("useEffect : Ambil data...");
      setData([]);
    }, 1000);

    //jika jeda ketikan kurang dari 1 detik
    //maka proses pengambilan data di skip dan clean up akan langsung dijalankan.
    return () => {
      console.log("CLEAN UP");
      clearTimeout(identifier);
    };
  }, [keyword]);

  const searchInputHandler = (e) => {
    setKeyword(e.target.value);
  };

  return (
    <div>
      <label htmlFor="pencarian">Cari : </label>
      <br />
      <input id="pencarian" onChange={searchInputHandler} />
    </div>
  );
};
```

(?) Cari Tahu

- Apa perbedaan antara lifecycle komponen berbasis kelas dengan komponen berbasis fungsi menggunakan Hooks?
- Kapan sebaiknya menggunakan komponen berbasis kelas daripada komponen berbasis fungsi dengan Hooks?

UNIVERSITAS MIKROSKIL

Diskusi

UNIVERSITAS
MIKROSKIL