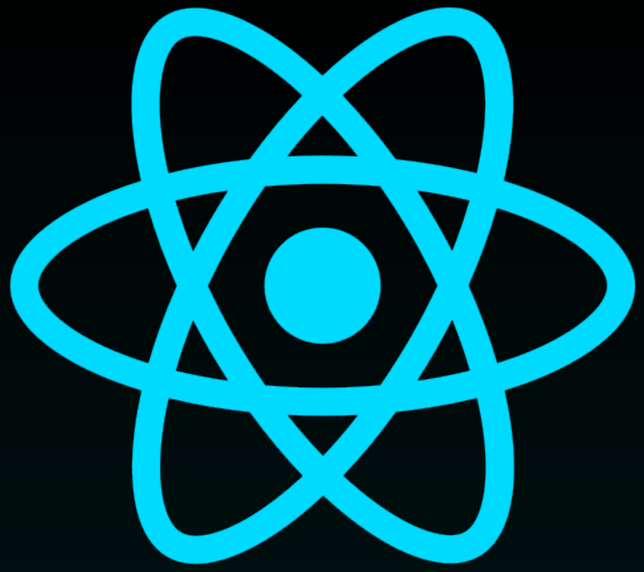


React

Una libreria JavaScript per creare interfacce utente

Febbraio - Marzo 2022

Marco Burrometo

Lezione 1 - 14 Feb 2022

Class component

- Come un componente funzione, con il contenuto renderizzato nella funzione *render* (unico metodo obbligatorio).
- All’interno della classe avremo lo stato che leggeremo con *this.state* e imposteremo con *this.setState(...)*
- L’oggetto *state* è immutabile e in sola lettura, per modificato dovremo sempre usare la funzione *setState*
- In aggiunta ci sono dei metodi detti di Lifecycle

Ciclo di vita di un componente

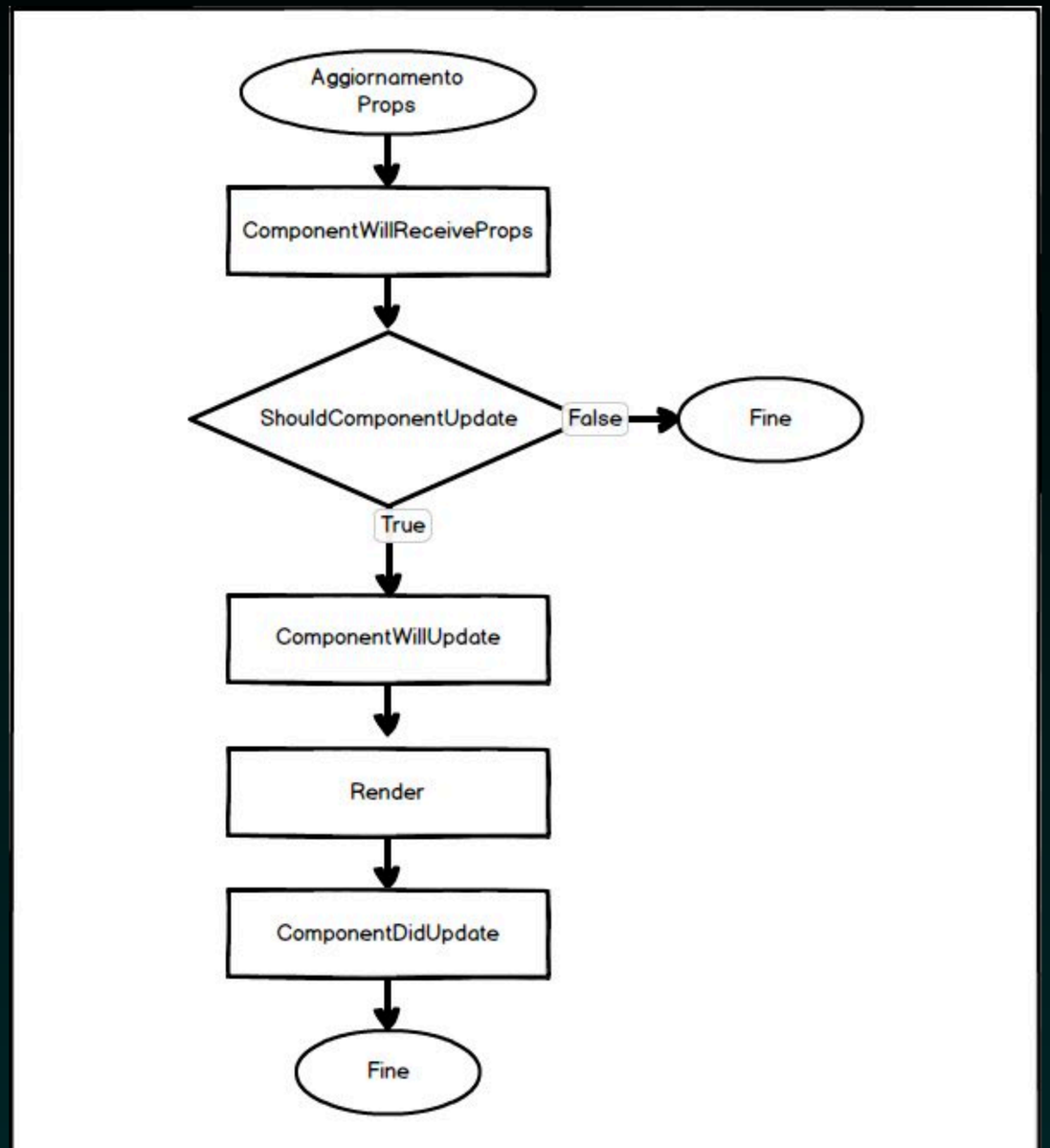
- Un aggiornamento può essere causato da cambiamenti alle props o allo state.
- Ogni componente ha un *ciclo di vita* che ha un inizio (mount), una fine (unmount) e uno stadio intermedio (render)

  Possiamo eseguire determinate azioni quando un componente viene creato, aggiornato o distrutto o, addirittura, decidere se un componente deve essere modificato in seguito alla variazione di almeno una delle proprietà contenute nel suo oggetto State o alla ricezione di nuove Props

Ciclo di vita di un componente

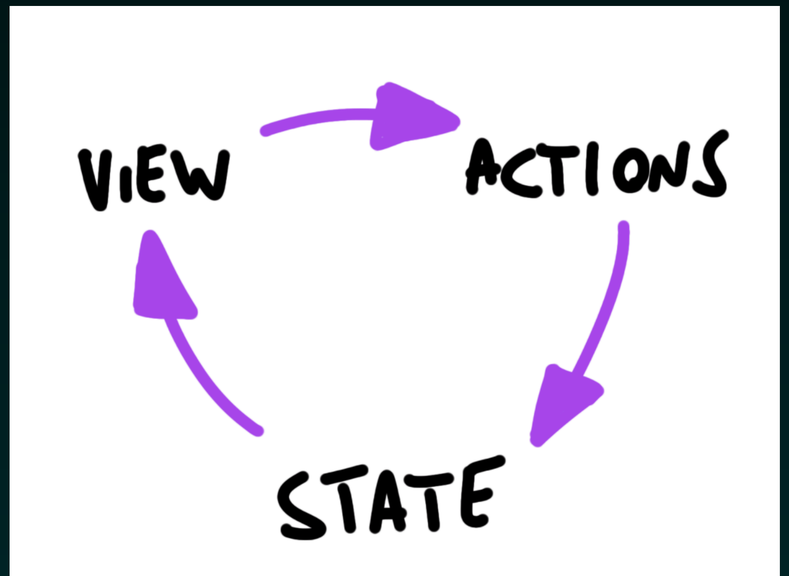
- *componentDidMount():* eseguito dopo che l'output del componente è stato renderizzato nel DOM per la prima volta. Invocato immediatamente dopo il primo *render*
- *shouldComponentUpdate():* Eseguito prima di ogni render, quando o lo stato o le props cambiano.
  riceve come argomenti gli oggetti *nextProps* e *nextState* i quali rappresentano il prossimo valore dell'oggetto *Props* e dell'oggetto *State*.
  Restituisce un booleano. Se questo metodo ritorna false i metodi successivi non vengono chiamati (utile per ottimizzazione)
- *componentDidUpdate():* invocato subito dopo il render
- *componentWillUnmount():* Invocato alla distruzione di un componente Quando, ad esempio a una determinata condizione il componente viene rimosso

Ciclo di vita di un componente



Unidirectional data flow

- Il flusso di dati segue un’unica direzione, il che significa che i dati hanno un *solo modo* per essere trasferiti ad altre parti dell’applicazione
- I componenti figlio non sono in grado di aggiornare i dati provenienti dal componente padre (lo stato è solo di un componente)
- Lo stato è sempre posseduto da uno specifico componente, e tutti i dati o la UI derivati da quello stato possono influenzare solamente i componenti “più in basso” nell’albero.



Stile

- Importo file css da file javascript

```
import "./App.css";
```

- Utilizzo un CSS preprocessor (es. SASS/SCSS) installando un package

```
npm install sass
```

JSX - Gestione liste

- Array.map

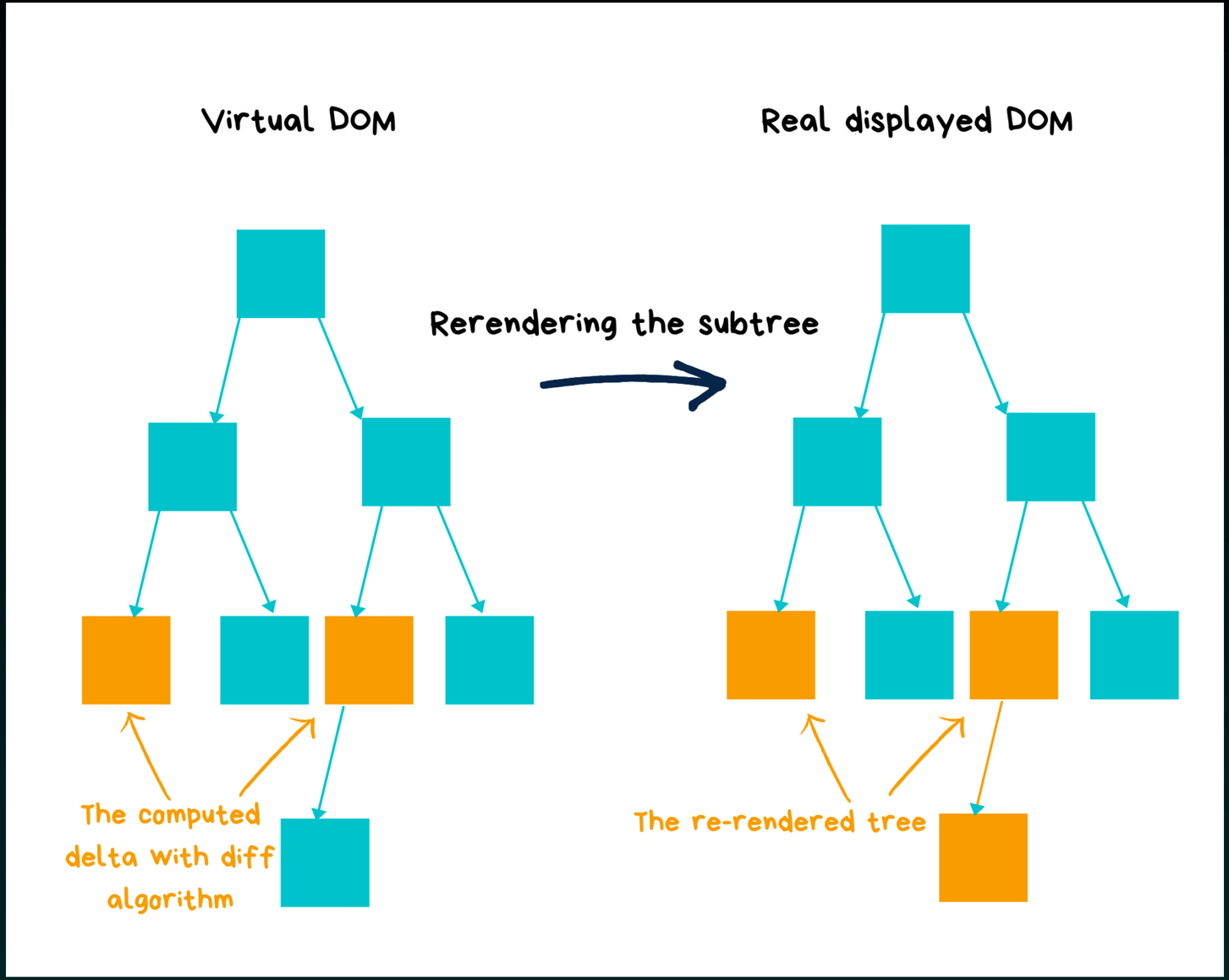
```
const numeri = [1, 2, 3, 4, 5];
const lista = numeri.map((numero) =>
  <li key={numero.toString()}>
    {numero}
  </li>
);
```

Le chiavi aiutano React a identificare quali elementi sono stati aggiornati, aggiunti o rimossi. Le chiavi dovrebbero essere fornite agli elementi all’interno dell’array per dare agli elementi un’identità stabile, quindi dovranno essere univoche

Virtual DOM

è un concetto di programmazione in cui una rappresentazione ideale, o "virtuale", di un'interfaccia utente viene conservata in memoria e sincronizzata con il DOM "reale" da una libreria come React.
Questo processo si chiama *riconciliazione*.

Virtual DOM


Virtual DOM
Real displayed DOM
Rerendering the subtree
The computed delta with diff algorithm
The re-rendered tree

Hook

https://it.reactjs.org/docs/hooks-intro.html

- Gli Hooks sono funzioni che ti permettono di “ancorarti” all’interno delle funzioni di React state e lifecycle da componenti funzione.
- Puoi richiamare gli Hooks solo al livello più alto. Non richiamare gli Hooks all’interno di cicli, condizioni o funzioni nidificate.
- Puoi richiamare gli Hooks solo da componenti funzione di React. Non richiamare gli Hooks da normali funzioni JavaScript.

useState

- Restituisce una coppia: il valore dello stato corrente (sola lettura) ed una funzione che ci permette di aggiornarlo.
- Accetta un unico parametro: Lo stato iniziale.

```
function Esempio() {
  // Dichiara una nuova variabile di stato, che chiameremo "contatore"
  const [contatore, setContatore] = useState(0);

  return (
    <div>
      <p>Hai cliccato {contatore} volte</p>
      <button onClick={() => setContatore(contatore + 1)}>
        Cliccami
      </button>
    </div>
  );
}
```

useEffect

- Usato per gestire “Effetti collaterali” in un componente React.
- Possiamo quindi decidere di svolgere determinate azioni quando desideriamo

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

function Esempio() {
  const [contatore, setContatore] = useState(0);

  // Simile a componentDidMount e componentDidUpdate:
  useEffect(() => {
    // Aggiorna il titolo del documento usando le API del browser
    document.title = `Hai cliccato ${contatore} volte`;
  });

  return (
    <div>
      <p>Hai cliccato {contatore} volte</p>
      <button onClick={() => setContatore(contatore + 1)}>
        Cliccami
      </button>
    </div>
  );
}
```

Aggiorna il titolo *dopo* ogni render

useEffect

- *useEffect* accetta anche un parametro, che è un array di dipendenze.
- L'*effetto collaterale* verrà scatenato solo quando almeno una di queste dipendenze cambierà. Possiamo passare un array vuoto per chiamare questa funzione solo una volta all'inizio

```
useEffect(() => {
  document.title = `You clicked ${count} times`;
}, [count]); // Only re-run the effect if count changes
```

Think in React 😎

https://it.reactjs.org/docs/thinking-in-react.html