CS472 WAP
React Communication

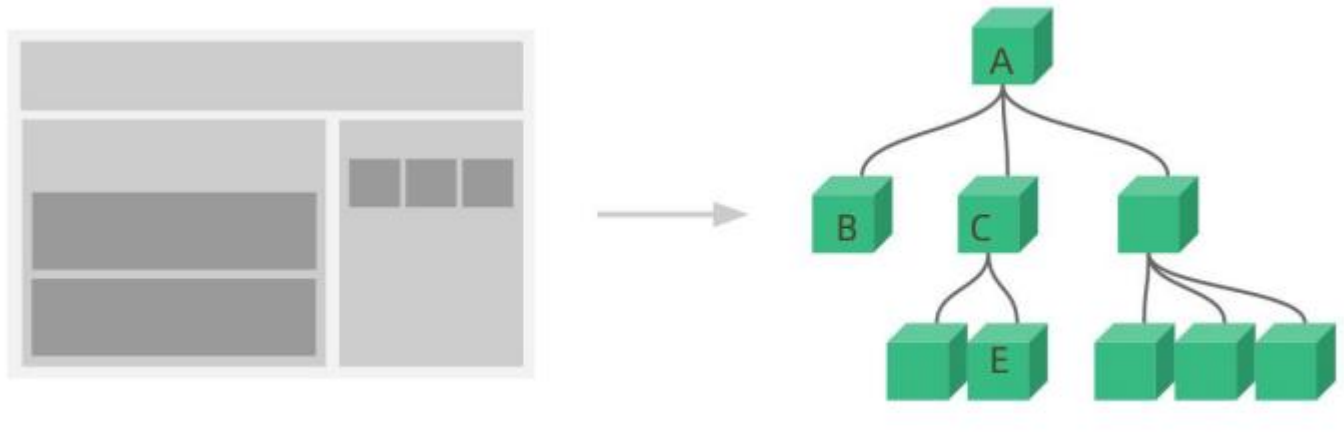
Maharishi International University - Fairfield, Iowa

Communication between Components

- In modern front-end frameworks/libraries, the entire page is divided into multiple smaller components. Component-based architecture makes it easy to develop and maintain an application. During the development, you may come across a situation where you need to share the data with other components.
- React provides several methods for handling the component communication, each with its appropriate use cases.
- Parent / Child communication
  - Parent to child communication
  - Child to parent communication
- Context API
- Centralized state with Redux
- …

Understanding Communication

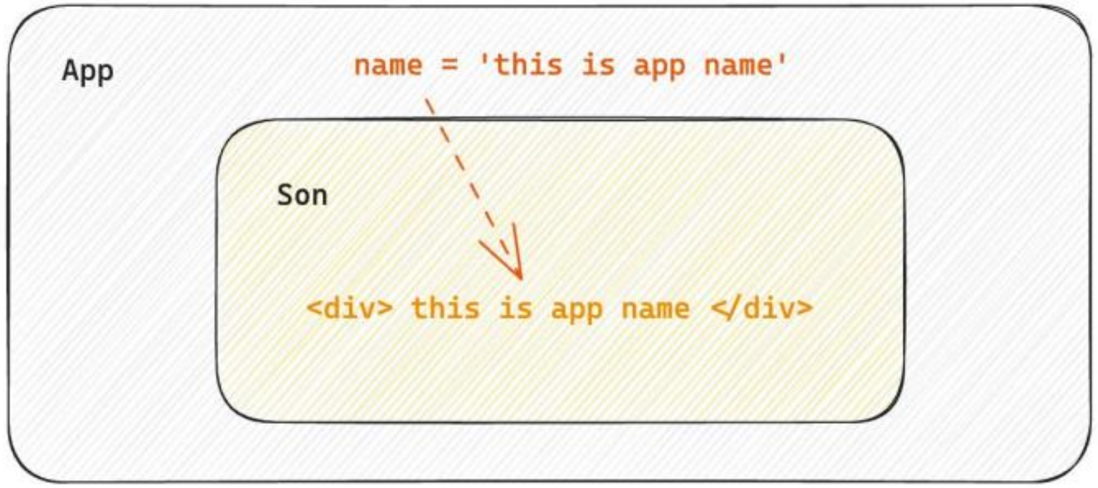
- Component communication refers to the transfer of data between components. Depending on the nesting relationship of the components, there are different communication methods.



- A-B Parent-Child communication
- B-C Sibling communication
- A-E Cross-layer communication

Parent to Child communication

- In React, parent components can communicate to child components using a special property defined by React called Props. Props are read-only and they're passed from parent to child component via HTML attributes.



- Steps:
  - Parent passes data - bind attributes to the child component.
  - Child receives data - the child component receives data through `props` parameter.

Props

- 1. can pass any kind of data
  - number, string, boolean, array, object, function, JSX

```
<Son name={name}
      age={20}
      isTrue={false}
      list={['React', 'Angular']}
      obj={{ name: 'Jack' }}
      cb={() => console.log(123)}
      child={<span>this is a span child</span>}
/>
```

- 2. read only
  - The child component can only read the data in props and cannot directly modify it. The data in the parent component can only be modified by the parent component.

Props Children

- When we nest content within the child component tags, the child component can get the content via the `children` property in props.

```
<Son>
     <span>This is a span in App</span>
</Son>
```

```
function Son(props: React.ReactNode) {
  return <div>this is son, {props.name}, {props.children}</div>
}
```

props & TypeScript

- Use type or interface to add types to component props.

```
type Props = {
  className: string
}

function Button(props: Props){
  const {className} = props;
  return <button className={className}>Click
Me</button>
}
```

- In the above example: The `Button` component can only accept a prop named '`className`', which must be of type `string` and is required.

Props & TypeScript - `children`

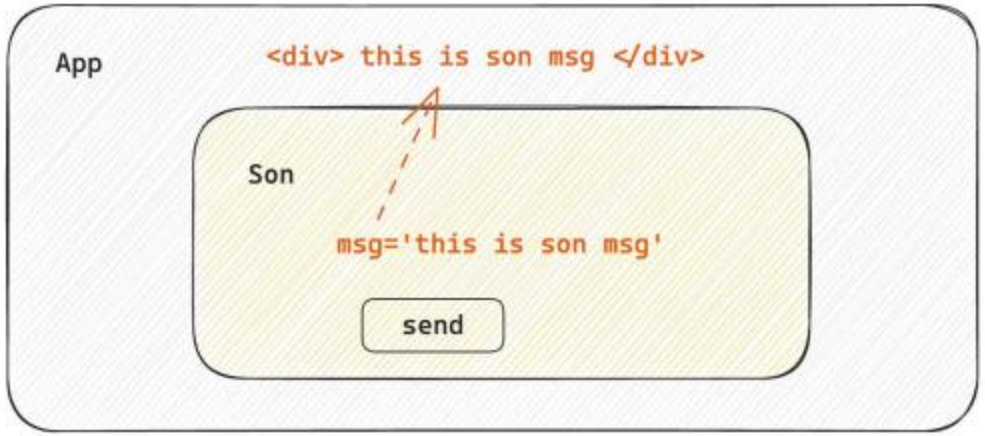
- `children` is a special prop, supporting various types of data, requires internal `ReactNode` type for annotation.
- `children` can be many types, such as： `React.ReactElement,string, number, React.ReactFragment, React.ReactPortal,Boolean, null, undefined`

```
<Son>
     <span>This is a span in App</span>
</Son>
```

```
function Son(props: React.ReactNode) {
  return <div>this is son, {props.name}, {props.children}</div>
}
```

Lifting State up: Child to parent communication

- Data from a child component to parent component can be passed using a callback. We can achieve this by following the below steps.
  - Create a callback method in the parent component and pass it to the child using props.
  - Child component can call this method using "`props.[callbackFunctionName]`" from the child and pass data as an argument.



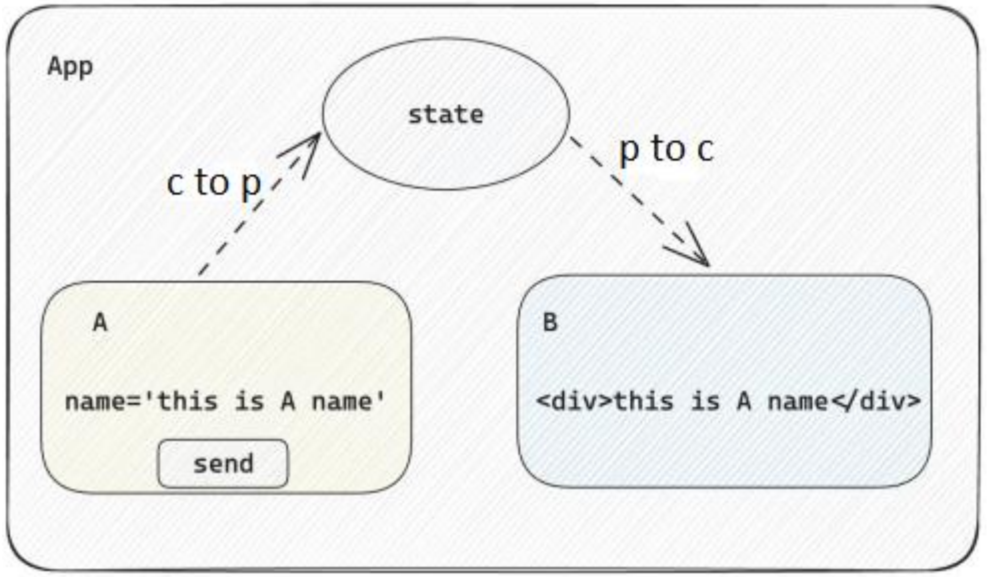
Child to parent communication Demo

```
function App() {
  const [message, setMessage] = useState('');
  const getMsg = (msg:string) => setMessage(msg);
  return (
    <div>
      {message}
      <Son onGetMsg={getMsg}/>
    </div>
  );
}
```

```
function Son({onGetMsg}:{onGetMsg:(msg: string) => void}) {
  const sonMsg = 'This is Son Msg';
  return (
    <div>
      <button onClick={() => onGetMsg(sonMsg)}>Send</button>
    </div>
  );
}
```

Two sibling components passing data to each other

- Sibling communication is merely the combination of prop drilling and lifting state up.



1. The child A that wants to pass data to another sibling, sends it to its parent first using callback mechanism.
2. The parent receives the data from one child and then passes it to the second child B using props of that child.

Two sibling communication Demo

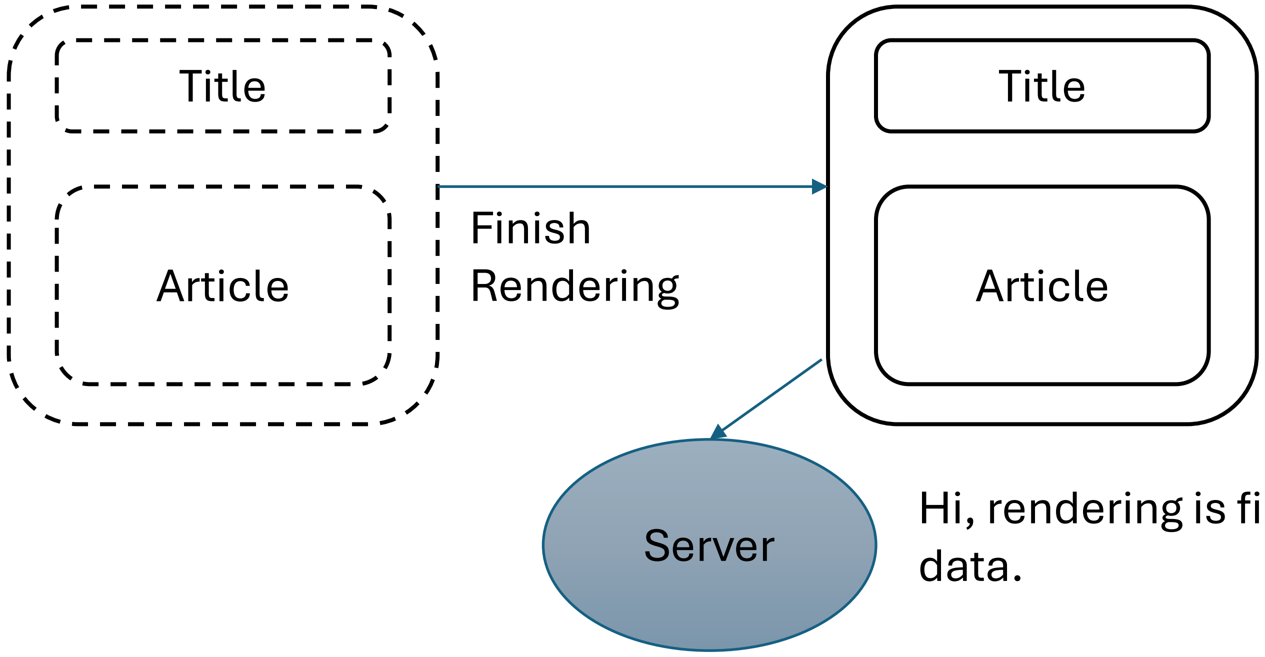
```
function A({onGetMsg}: {onGetMsg: (msg: string) => void}){
   const name = 'this is A name';
   return <button onClick={() => onGetMsg(name)}></button>
}
```

```
function App() {
  const [message, setMessage] = useState('');
  const getMsg = (msg:string) => setMessage(msg);
  return (
    <div>
      <A onGetMsg={getMsg}/>
      <B message={message}/>
    </div>
  );
}
```

```
function B({message}: {message: string}){
  return <div>{message}</div>
}
```

useEffect Hook

- The useEffect is a React Hook function used to perform operations (side effects) in React components that are not triggered by events but by the rendering itself, such as sending AJAX requests, modifying the DOM, and so on.



In the diagram, there are no user events occurring. After the component is rendered, it needs to fetch data from the server. The entire process belongs to "operations triggered solely by rendering”.

Hi, rendering is finished, I need some data.

Problem

useEffect basic usage

- Requirement: fetch data from server and display on the page once the component is rendered.

```
useEffect(() => { //Runs only on the first render }, []);
```

- Parameter 1 is a function, which can be called the effect function. Inside this function, you can place the operations to be executed.
- Parameter 2 is an array (optional), where you place dependencies. Different dependencies will affect the execution of the function passed as the first parameter. When it's an empty array, the effect function will only be executed once after the component is rendered.

Demo

```
const URL = 'https://tinaxing2012.github.io/jsons/data.json';

function App() {

  const [list, setList] = useState([]);

  useEffect(() => {
    async function getList(){
      const res = await fetch(URL);
      const data = await res.json();
      setList(data.data.channels);
    }
    getList();
  })
  return (
    <div>
      this is app
      <ul>
        {list.map((item: {id: number, name:string}) => <li key={item.id}>{item.name}</li>)}
      </ul>
    </div>
  );
}
```

useEffect dependencies

- The execution of the useEffect effect function varies depending on the dependencies passed in. Different execution behaviors will occur based on the provided dependencies.

Dependencies	Execution of Effect function
No dependency passed	Runs on every render
An empty array	Runs only on the first render
Props or state values	Runs on first render + And any time any dependency value changes

useEffect Demo 1 – doesn’t depend on anything

```
function App() {

  const [count, setCount] = useState(0);
  useEffect(() => {
    console.log('Effect function is called');
  });

  return (
    <div>
      this is app
      <button onClick={() => setCount(count+1)}>+1</button>
    </div>
  );

}
```

useEffect Demo – depend on empty array []

```
function App() {

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

  useEffect(() => {
    console.log('Effect function is called');
  }, []);

  return (
    <div>
      this is app
      <button onClick={() => setCount(count+1)}>+1</button>
    </div>
  );

}
```

useEffect Demo 3 - depend on a variable

```
function App() {

  const [count, setCount] = useState(0);
  const [msg, setMsg] = useState('');

  useEffect(() => {
    console.log('Effect function is called');
  }, [count]);

  return (
    <div>
      this is app
      <button onClick={() => setCount(count+1)}>+1</button>
      <button onClick={() => setMsg(msg +'hi')}>Message</button>
    </div>
  );
}
```

Effect cleanup

- Some effects require cleanup to reduce memory leaks.
- Timeouts, subscriptions, event listeners, and other effects that are no longer needed should be disposed.
- We do this by including a return function at the end of the `useEffect` Hook.
- Note: The most common timing for executing effect cleanup is component unmounting.

```
useEffect(() => {
    return () => {
    }
  }, []);
```

Cleanup Demo

```
function Son(){
  useEffect(() => {
    const timerId = setInterval(() => {console.log('interval...')}, 1000);
    return () => {
      clearInterval(timerId);
    }
  }, []);
  return <div>This is Son!</div>
}

function App() {

  const [show, setShow] = useState(true);

  return (
    <div>
      this is app
      <button onClick={() => setShow(!show)}>Toggle</button>
      {show && <Son/>}
    </div>
  );

}
```

CSS

React

CSS

- Two ways to add CSS in React:
  - Inline Style

```
<button style={{color: 'red'}}>Change Name</button>
```

  - Use className

```
import './App.css';

<p className='foo'>use className</p>
```

```
.foo {
  color: pink
}
```

Modularize CSS in React

- To support CSS Modules alongside regular stylesheets using the [name].module.css file naming convention.

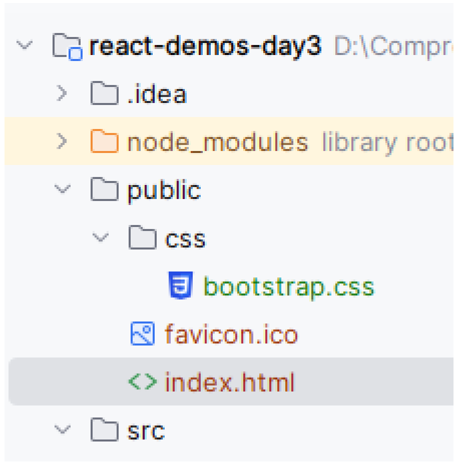
```
import hello from './Hello.module.css';
export default function Hello(){
    return (
        <div>
            <h1 className={hello.title}>Hello Component!!</h1>
        </div>
    )
}
```

- Alternatives:
  - Use Less, Sass
  - styled-components

Bootstrap

1. Download bootstrap.css
2. Place in `public` -> `css` folder
3. Link in `index.html`

```
<!DOCTYPE html>
<html lang="en">
 <head>
  <meta
   name="description"
   content="Web site created using create-react-app"
  />
  <link href="css/bootstrap.css" type="text/css" rel="stylesheet" />
  <title>React App</title>
 </head>
 <body>
  <noscript>You need to enable JavaScript to run this app.</noscript>
  <div id="root"></div>
 </body>
</html>
```



```
react-demos-day3 D:\Compr
  .idea
  node_modules library root
  public
    css
      bootstrap.css
    favicon.ico
    index.html
  src
```