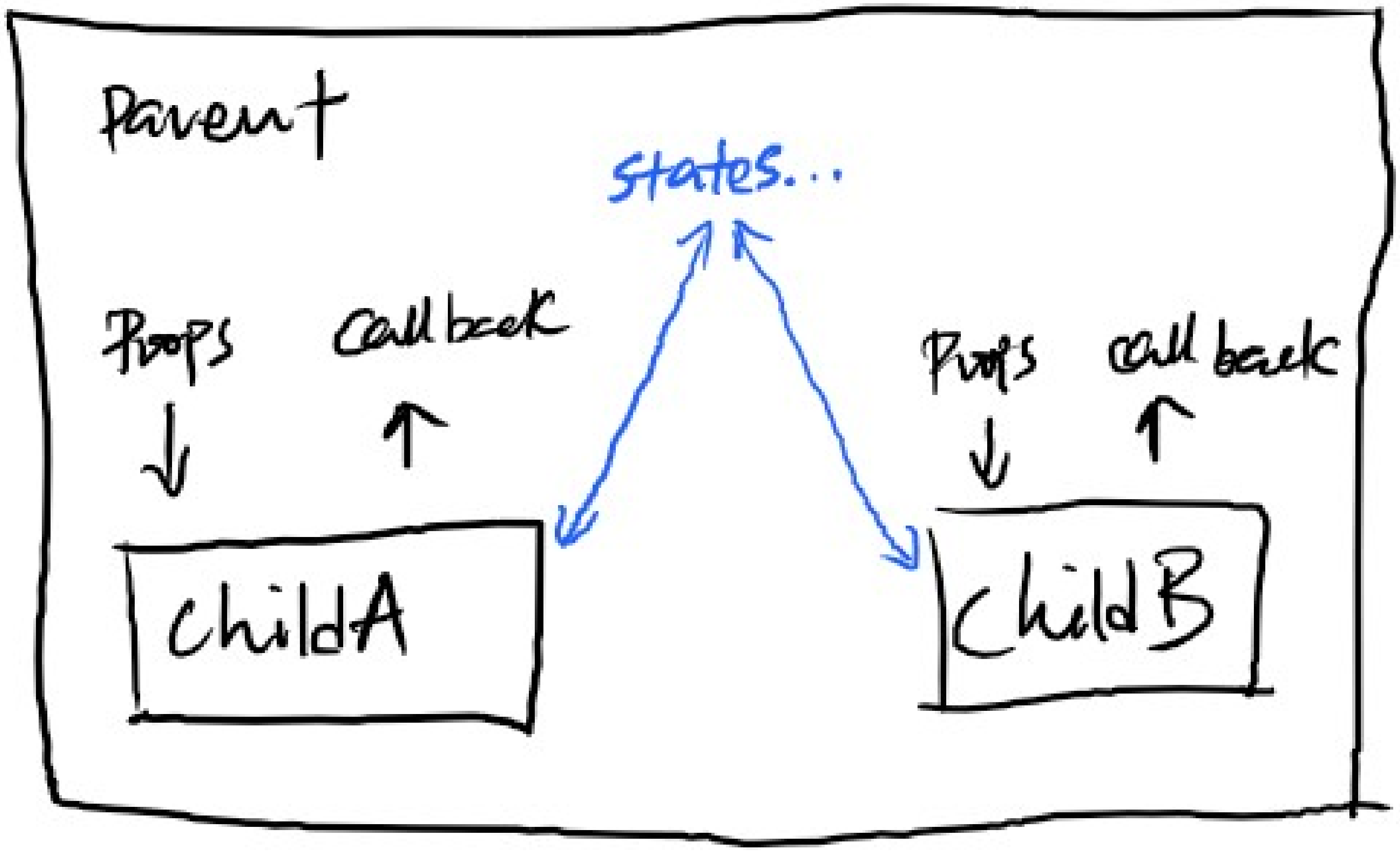
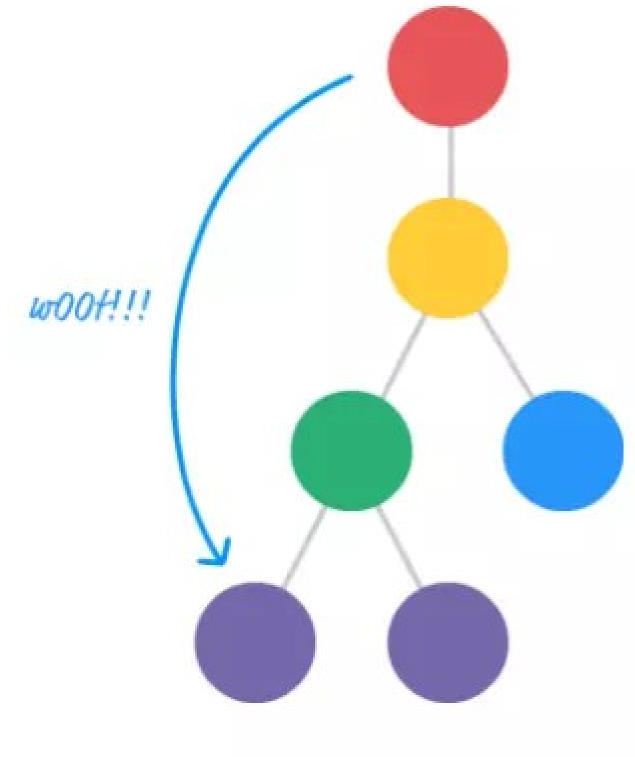
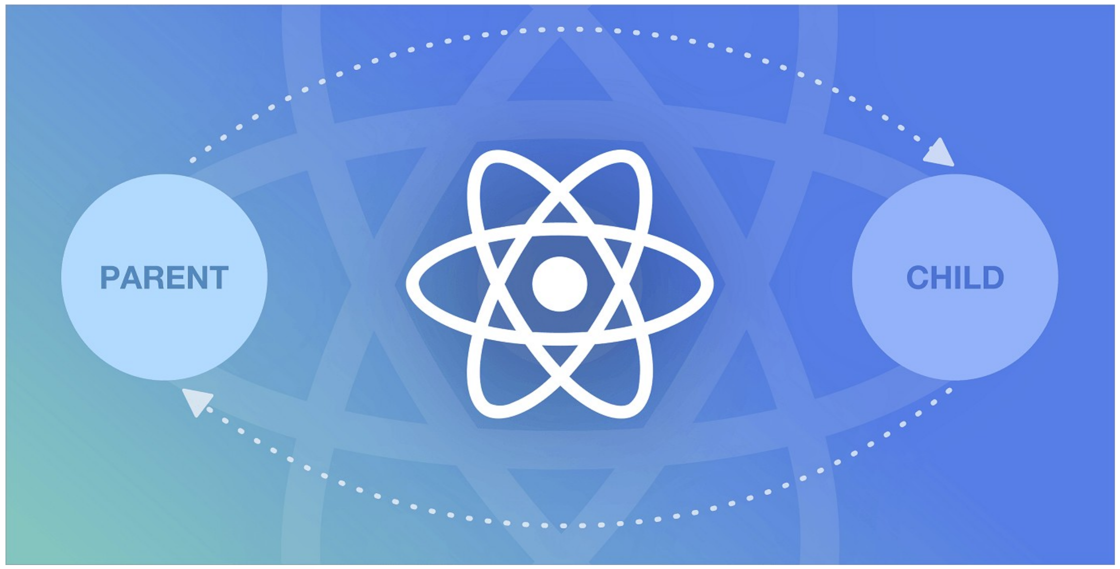
CS385 Mobile Application Development (Lecture 8)



Peter Mooney

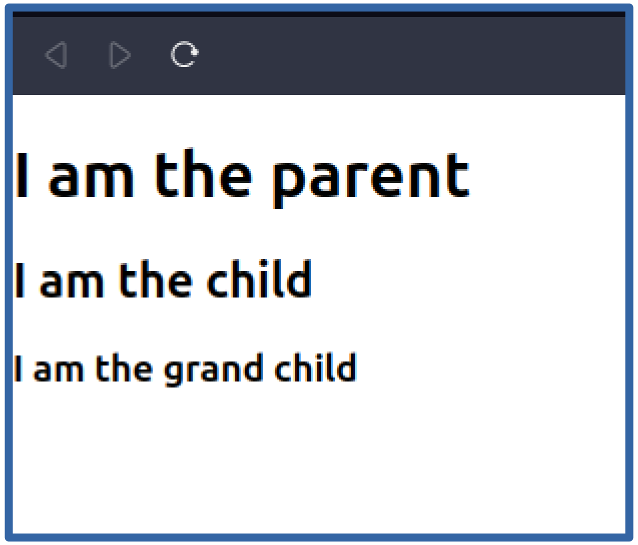
Nested, deep, or multi-level components in React





Introducing multi-level components

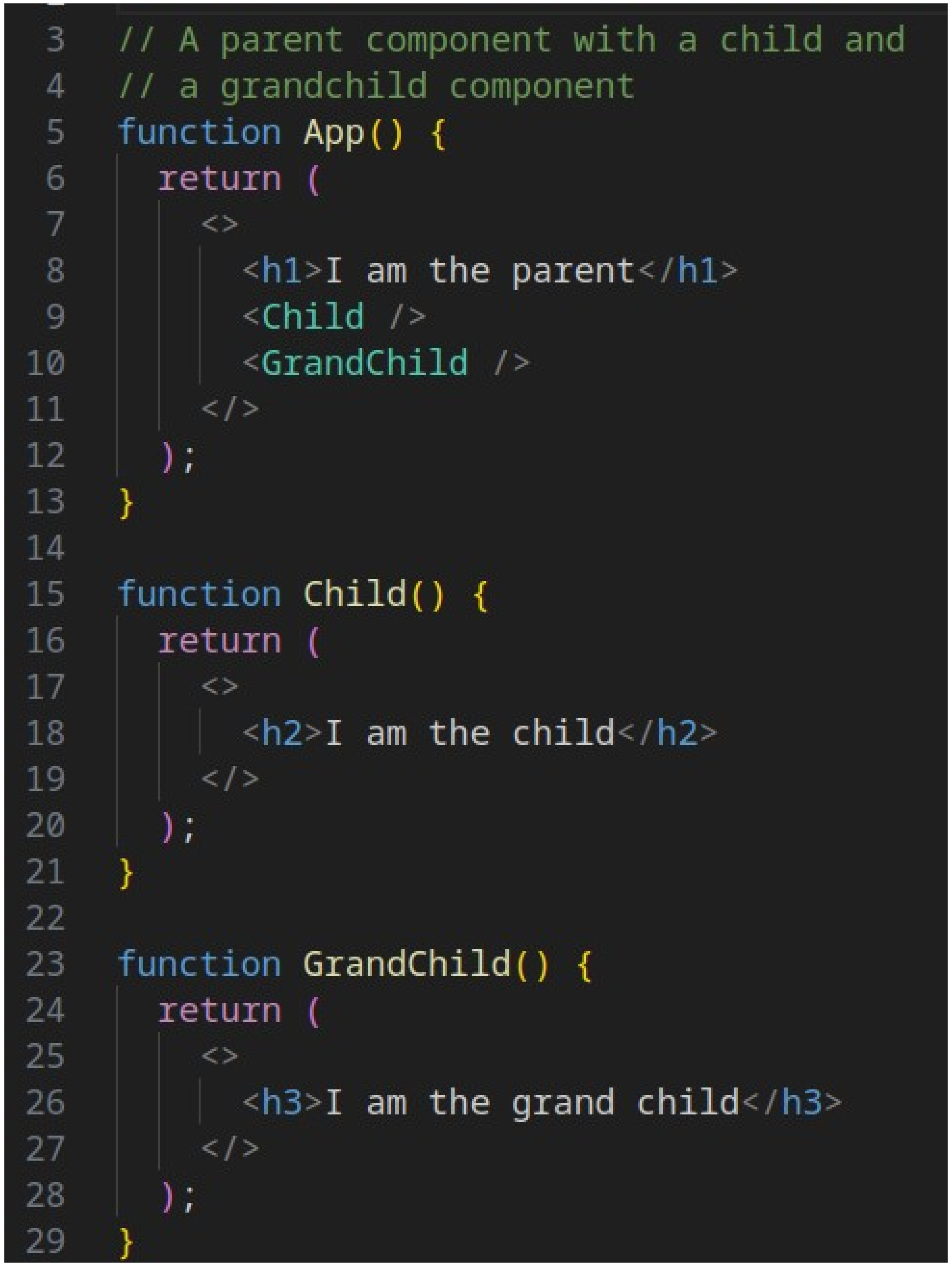
- This looks like **`parent-child-grandparent`** nesting.
- But it isn't! Why not?



I am the parent

I am the child

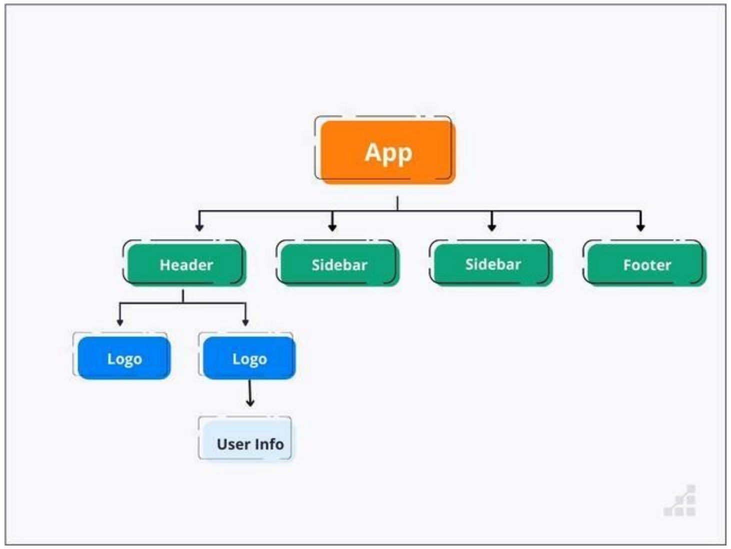
I am the grand child

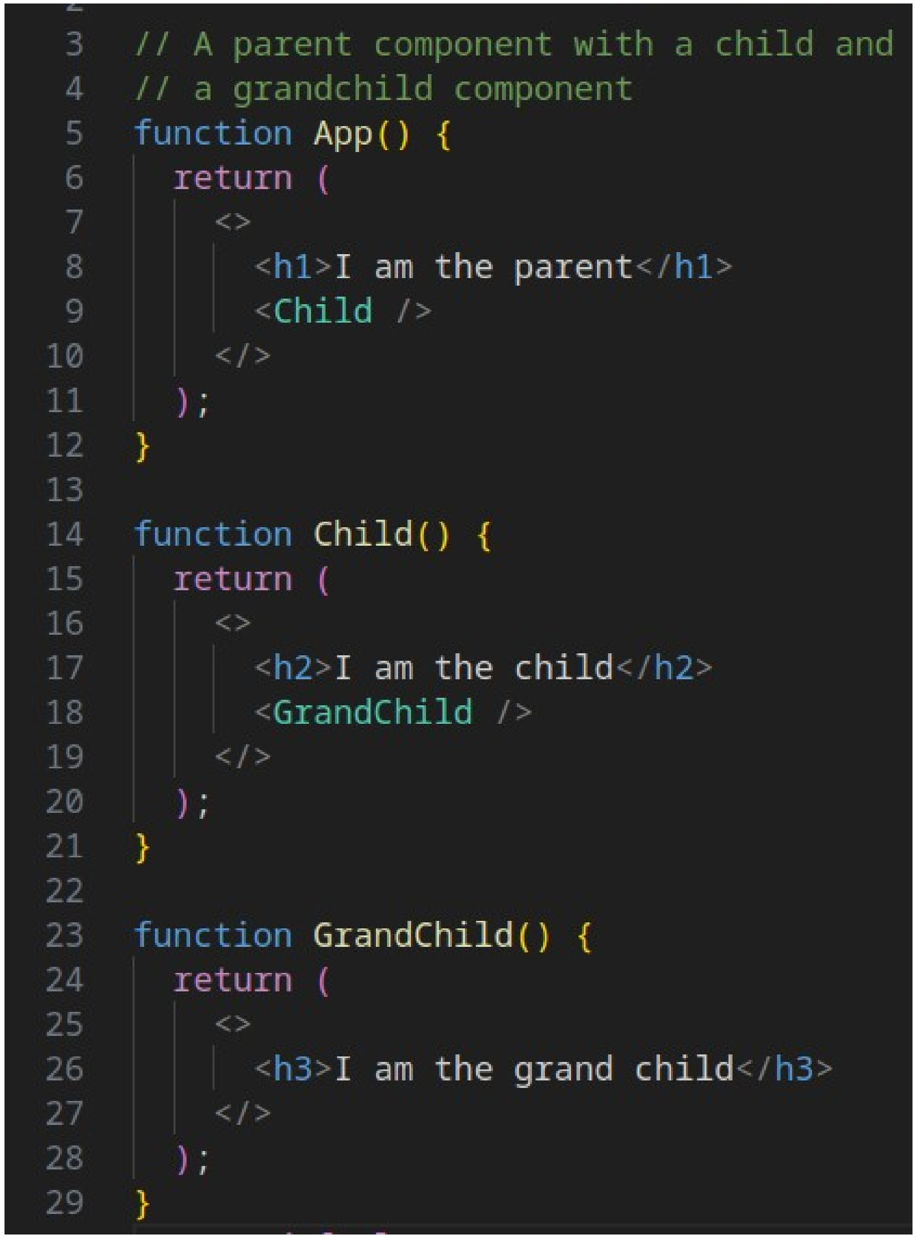


```
// A parent component with a child and
// a grandchild component
function App() {
  return (
    <>
      <h1>I am the parent</h1>
      <Child />
      <GrandChild />
    </>
  );
}

function Child() {
  return (
    <>
      <h2>I am the child</h2>
    </>
  );
}

function GrandChild() {
  return (
    <>
      <h3>I am the grand child</h3>
    </>
  );
}
```

Correct component nesting

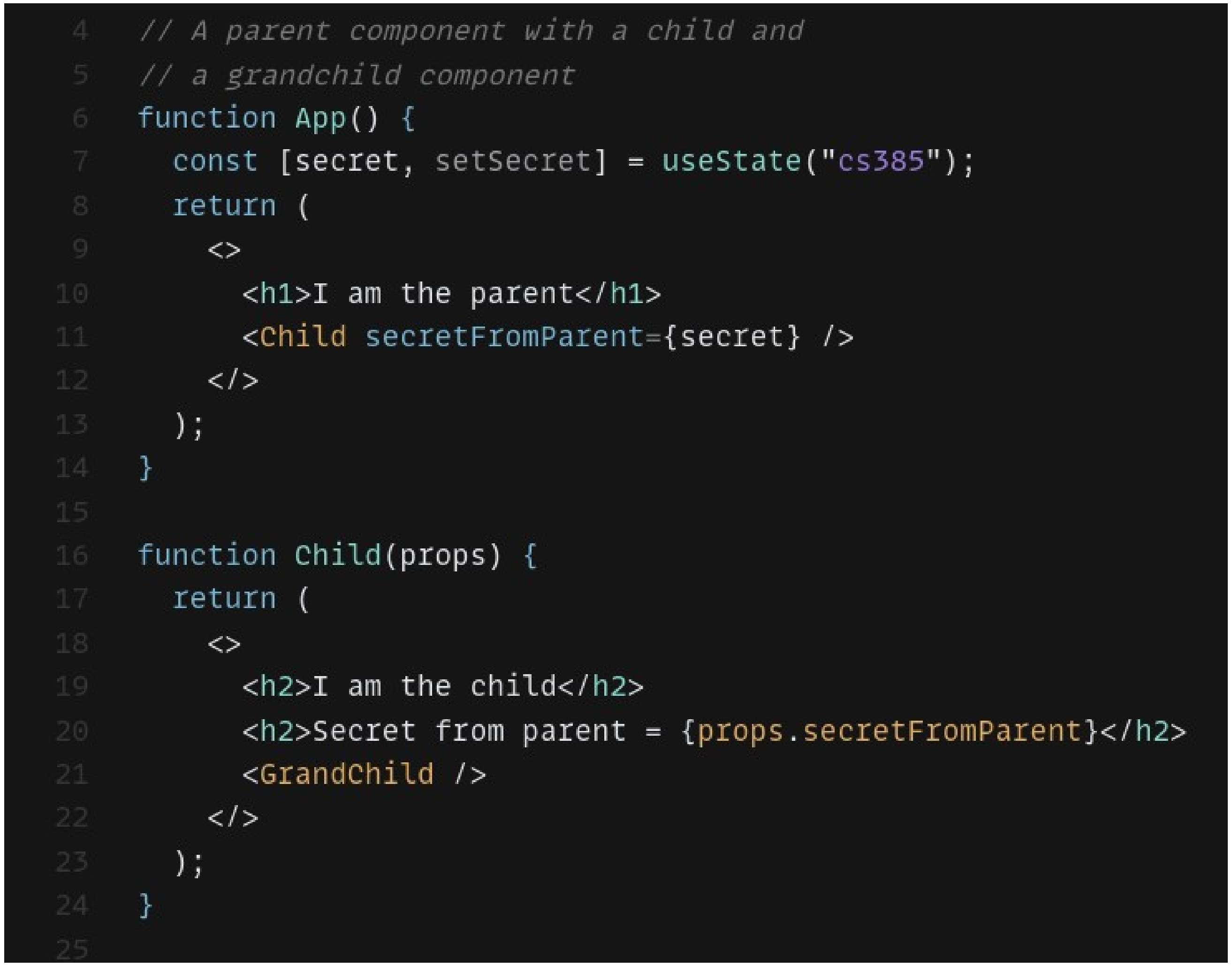
- This is the proper hierarchy
- The **<Child/>** component is invoked in the Parent (**<App/>**
- The **<GrandChild/>** is invoked within the **<Child/>**





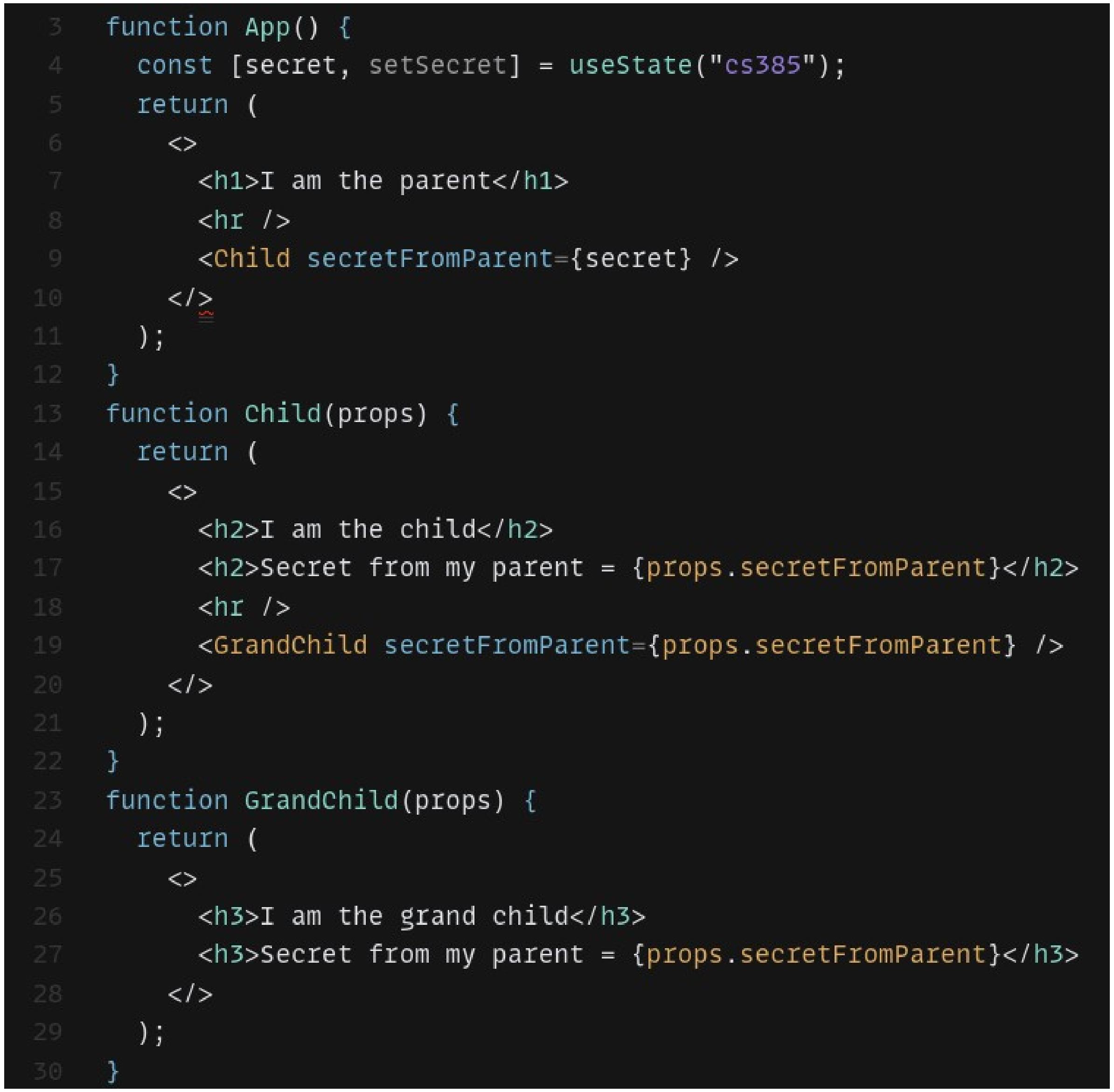
```
// A parent component with a child and
// a grandchild component
function App() {
  return (
    <>
      <h1>I am the parent</h1>
      <Child />
    </>
  );
}

function Child() {
  return (
    <>
      <h2>I am the child</h2>
      <GrandChild />
    </>
  );
}

function GrandChild() {
  return (
    <>
      <h3>I am the grand child</h3>
    </>
  );
}
```

Passing data from parent to child (revision from Lecture 5)



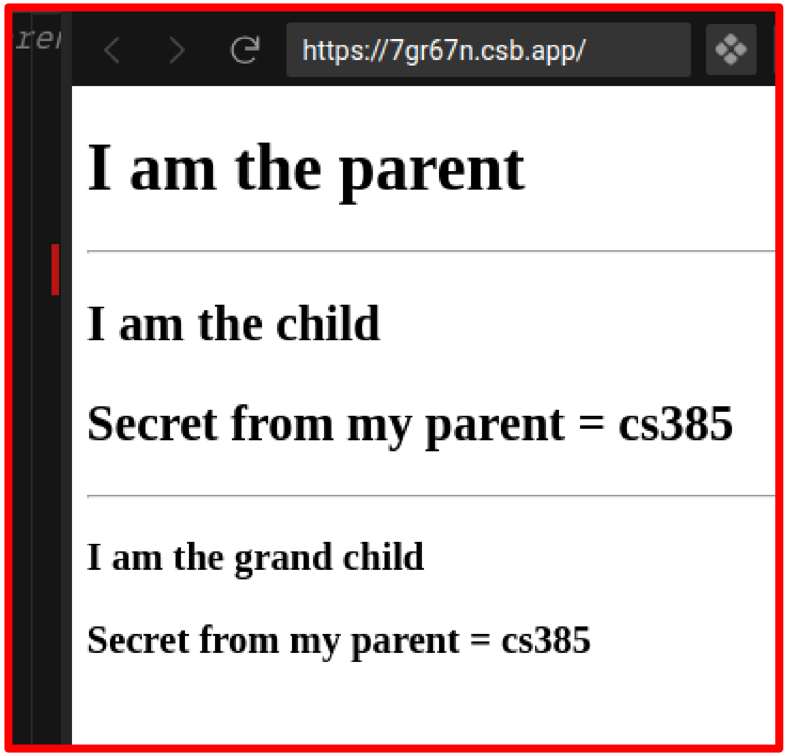
```
// A parent component with a child and
// a grandchild component
function App() {
  const [secret, setSecret] = useState("cs385");
  return (
    <>
      <h1>I am the parent</h1>
      <Child secretFromParent={secret} />
    </>
  );
}

function Child(props) {
  return (
    <>
      <h2>I am the child</h2>
      <h2>Secret from parent = {props.secretFromParent}</h2>
      <GrandChild />
    </>
  );
}

```

IMPORTANT – moving a data value down multiple components



```
function App() {
  const [secret, setSecret] = useState("cs385");
  return (
    <>
      <h1>I am the parent</h1>
      <hr />
      <Child secretFromParent={secret} />
    </>
  );
}
function Child(props) {
  return (
    <>
      <h2>I am the child</h2>
      <h2>Secret from my parent = {props.secretFromParent}</h2>
      <hr />
      <GrandChild secretFromParent={props.secretFromParent} />
    </>
  );
}
function GrandChild(props) {
  return (
    <>
      <h3>I am the grand child</h3>
      <h3>Secret from my parent = {props.secretFromParent}</h3>
    </>
  );
}
```

- It is very important to understand how the secret variable from the **App** component is communicated down to the grandchild via props.
- The grandchild can only know the variable secret came from its own parent (not the grand parent)



I am the parent

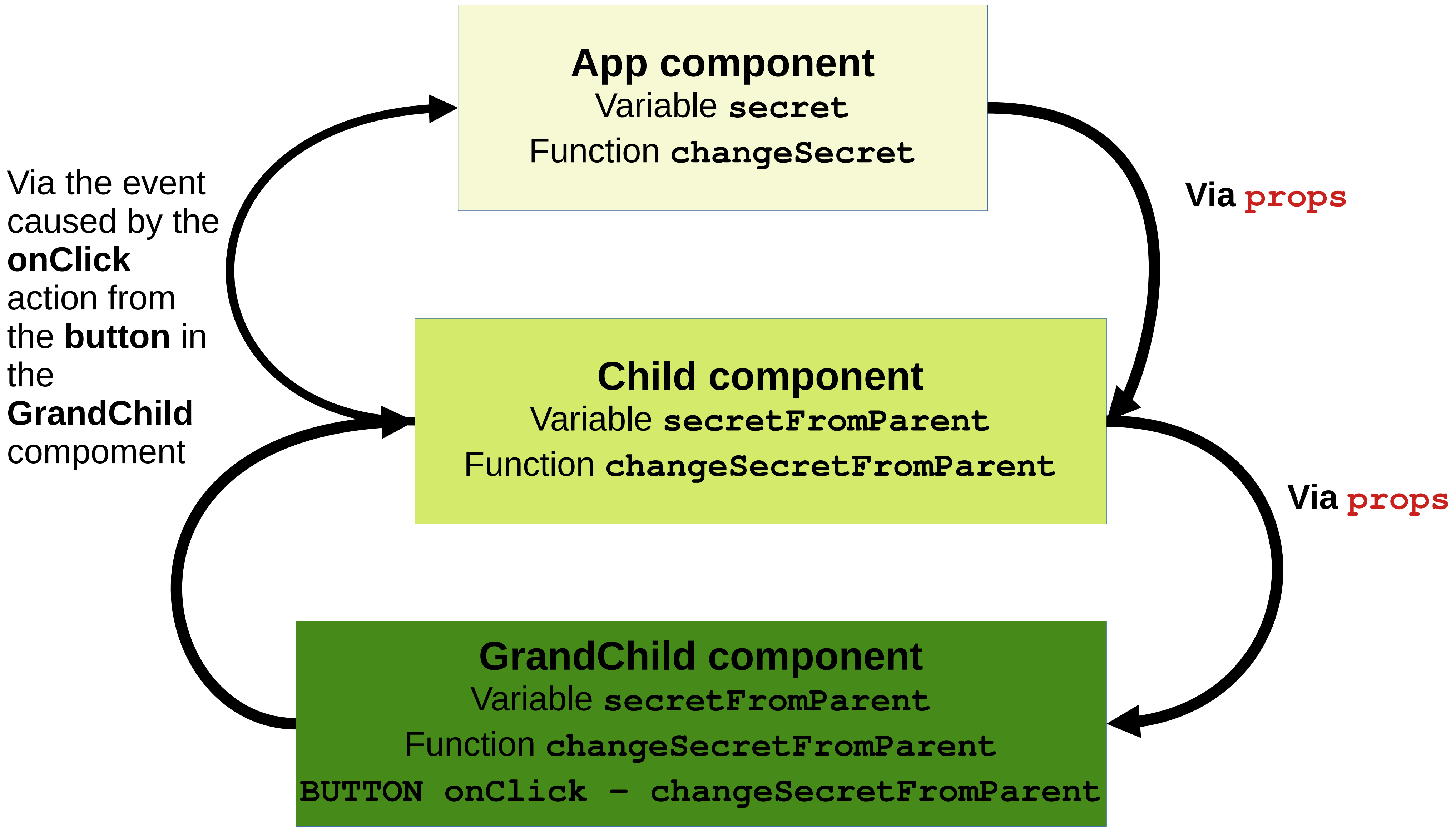
I am the child

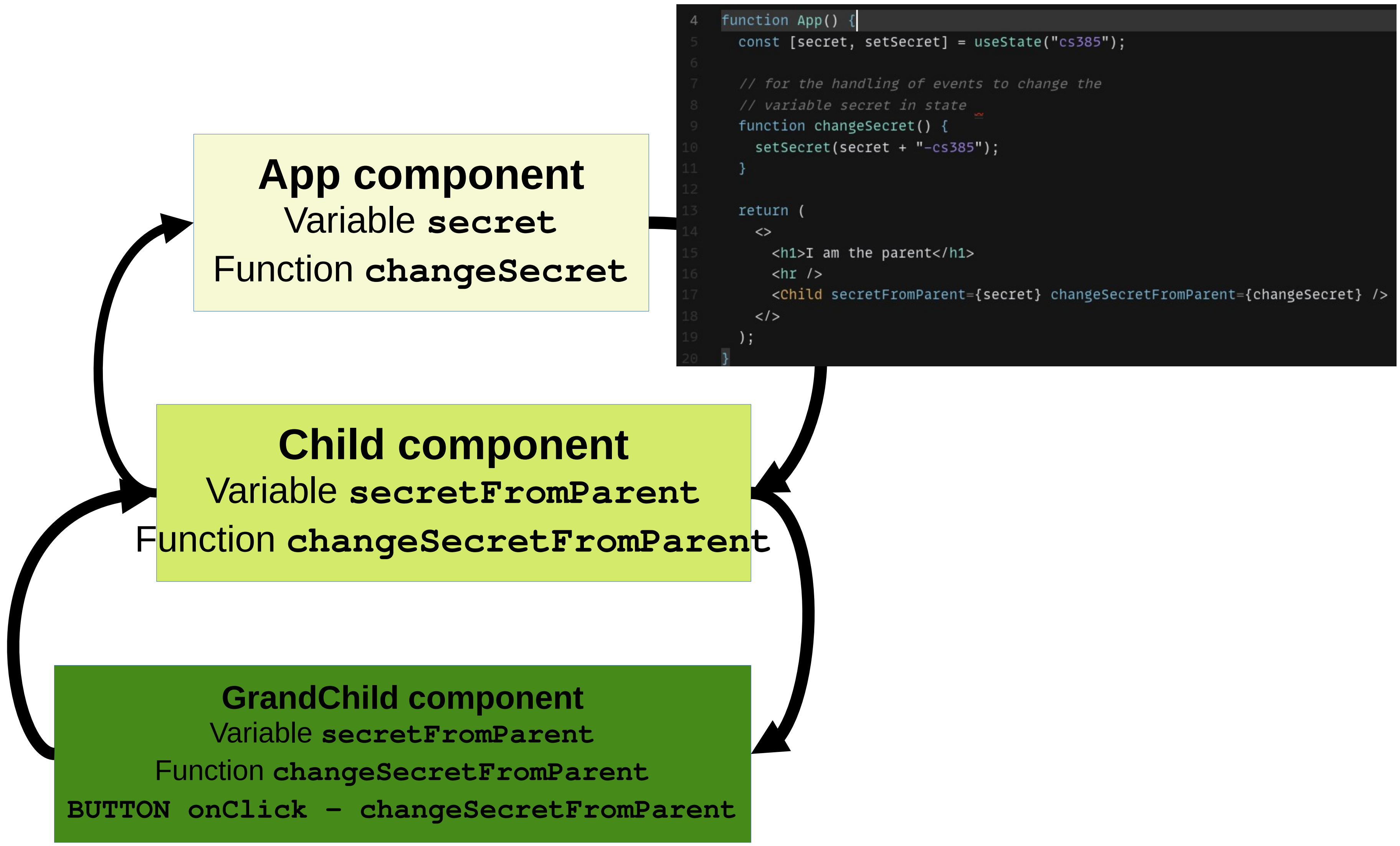
Secret from my parent = cs385

I am the grand child

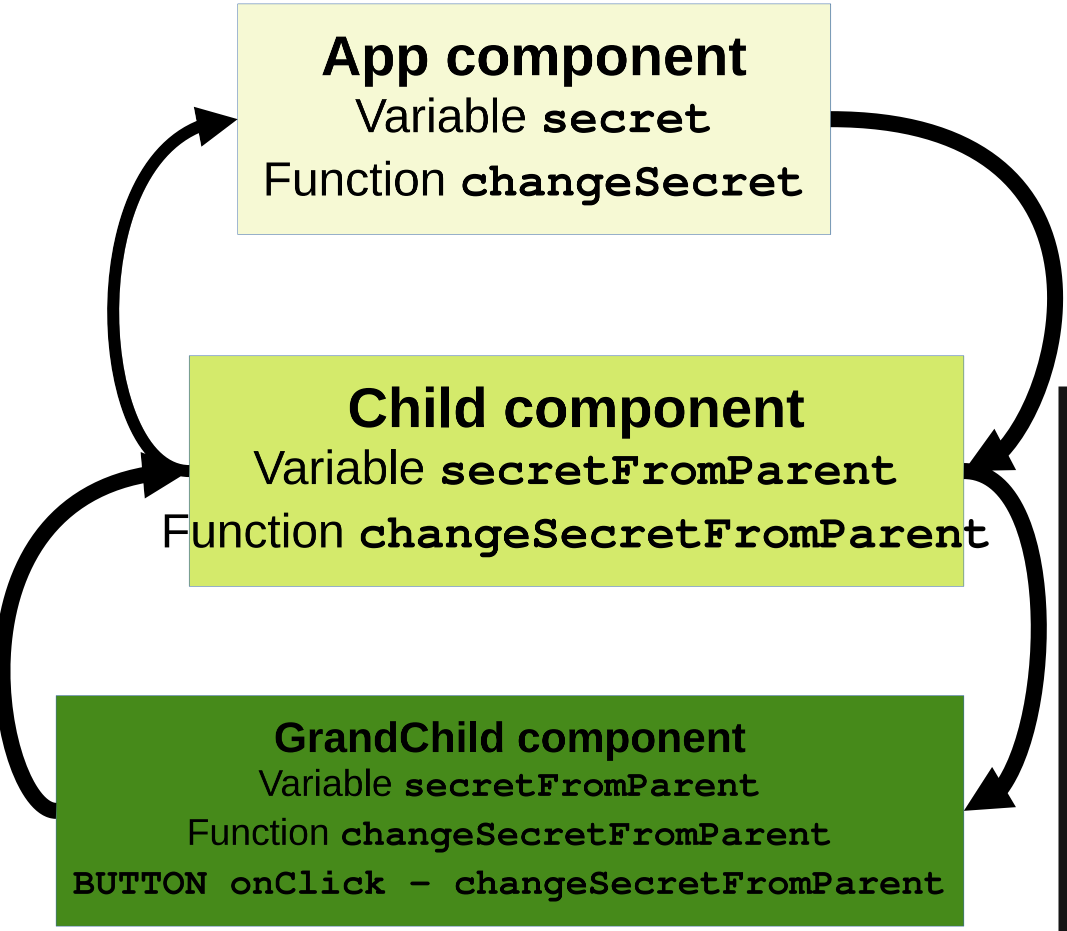
Secret from my parent = cs385

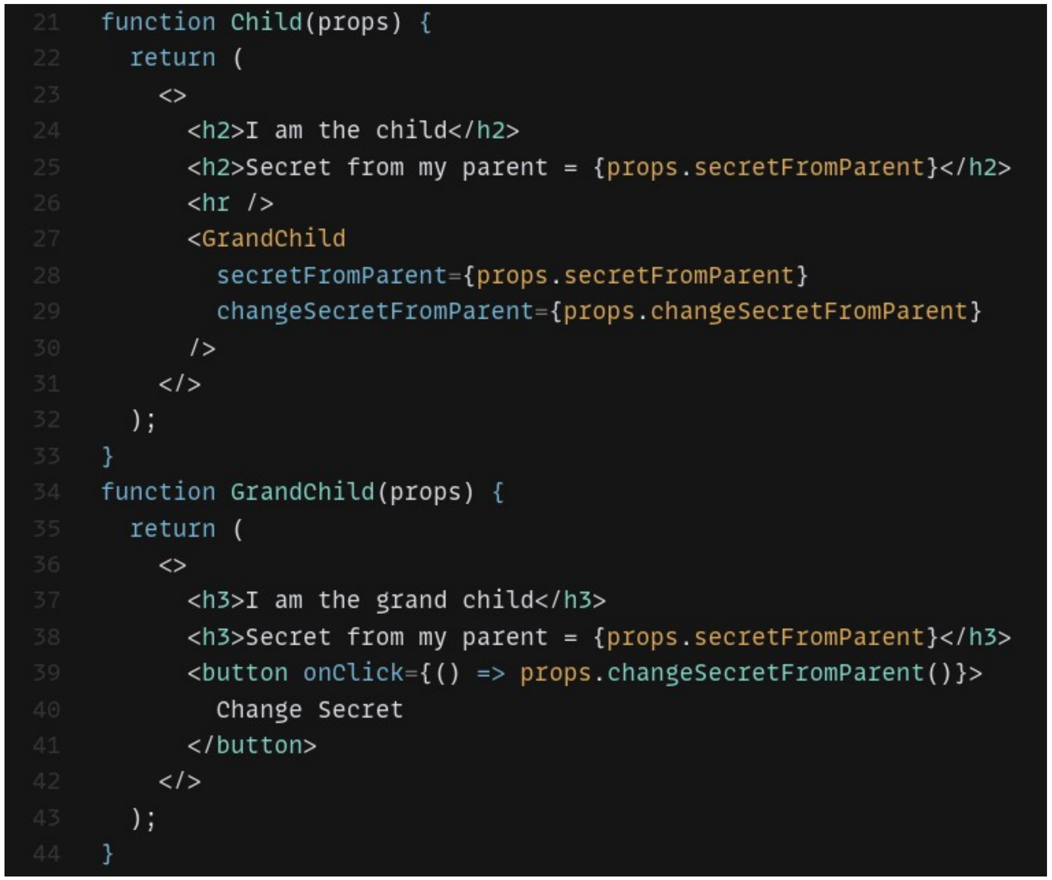
Multiple components – communication


App component
Variable **secret**
Function **changeSecret**
Via **props**
Child component
Variable **secretFromParent**
Function **changeSecretFromParent**
Via **props**
GrandChild component
Variable **secretFromParent**
Function **changeSecretFromParent**
BUTTON onClick – changeSecretFromParent
Via the event caused by the **onClick** action from the **button** in the **GrandChild** compoment

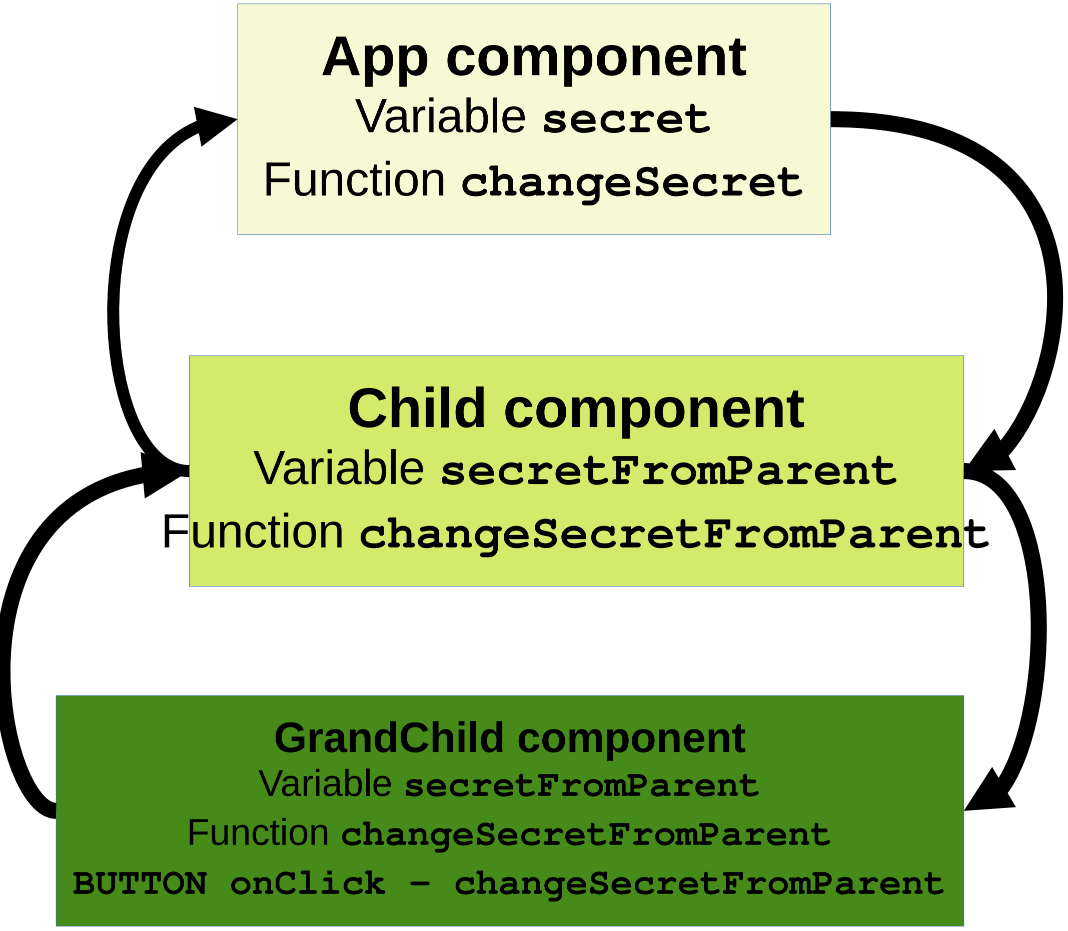

App component
Variable secret
Function changeSecret
Child component
Variable secretFromParent
Function changeSecretFromParent
GrandChild component
Variable secretFromParent
Function changeSecretFromParent
BUTTON onClick - changeSecretFromParent

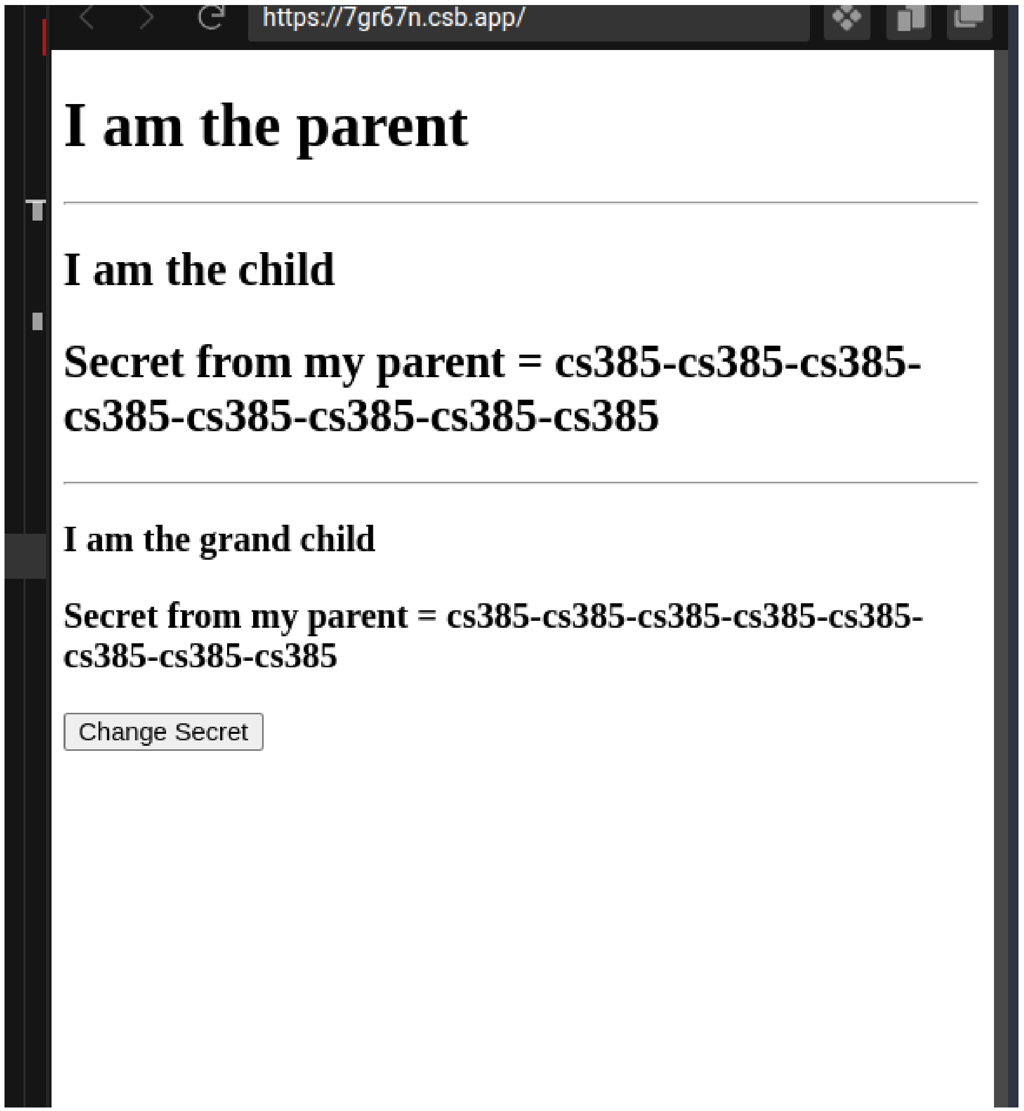
```
function App() {
  const [secret, setSecret] = useState("cs385");

  // for the handling of events to change the
  // variable secret in state
  function changeSecret() {
    setSecret(secret + "-cs385");
  }

  return (
    <>
      <h1>I am the parent</h1>
      <hr />
      <Child secretFromParent={secret} changeSecretFromParent={changeSecret} />
    </>
  );
}
```


App component
Variable secret
Function changeSecret
Child component
Variable secretFromParent
Function changeSecretFromParent
GrandChild component
Variable secretFromParent
Function changeSecretFromParent
BUTTON onClick - changeSecretFromParent



```
function Child(props) {
  return (
    <>
      <h2>I am the child</h2>
      <h2>Secret from my parent = {props.secretFromParent}</h2>
      <hr />
      <GrandChild
        secretFromParent={props.secretFromParent}
        changeSecretFromParent={props.changeSecretFromParent}
      />
    </>
  );
}
function GrandChild(props) {
  return (
    <>
      <h3>I am the grand child</h3>
      <h3>Secret from my parent = {props.secretFromParent}</h3>
      <button onClick={() => props.changeSecretFromParent()}>
        Change Secret
      </button>
    </>
  );
}
```


App component
Variable secret
Function changeSecret
Child component
Variable secretFromParent
Function changeSecretFromParent
GrandChild component
Variable secretFromParent
Function changeSecretFromParent
BUTTON onClick - changeSecretFromParent



I am the parent

I am the child

Secret from my parent = cs385-cs385-cs385-cs385-cs385-cs385-cs385-cs385

I am the grand child

Secret from my parent = cs385-cs385-cs385-cs385-cs385-cs385-cs385-cs385

Advantages of multi-level communication with components

- **It is relatively easily faciliated with props and good event management.**
- There is **theoretically no limit** to the number of child-parent combinations you can have.
- **Each component can be given different tasks to perform without having a negative effect on the flow of information** (child parent commnunication)

Disadvantages of multi-level communication with components

- Can **led to overly complicated and complex code that is difficult to maintain and debug**
- **Is your software design effective and efficient?** Why do you need such deep nesting of components?
- **Which components REALLY need to be able to effect change on their parents' or grandparents' state variables?**

Multi-level components – best practice approaches

- **There are other ways to achieve this using React** – such as using the ContextAPI from React
- **Always check your design before implementation** – this is the key to ensuring if you really need to pass data or functions down to multiple levels.
- Context API lets the parent component make some information available to any component in the tree below it—no matter how deep—without passing it explicitly through **props**.

Question **1**

Not yet answered

Marked out of 1.00

Flag question

Edit question

In the fully working React application below there is one parent component and two child components (as shown). By ignoring any considerations around formatting, how many times is the word 'Barcelona' rendered to the screen when this application runs?

Time left 1:59:39

```
// App is the parent component
function App() {
  return (
    <>
      <Barcelona /> <b>Barcelona</b>
      <Madrid /> <Madrid />
    </>
  );
}

function Barcelona() {
  // Barcelona is a child component
  return (
    <>
      <p>Barcelona</p>
    </>
  );
}

function Madrid() {
  // Madrid is a child component
  return (
    <>
      <p>Madrid</p>
    </>
  );
}
```

- ○ a. 1
- ○ b. 3
- ○ c. 2 times
- ○ d. Barcelona is not printed or rendered to the screen
- ○ e. 0

Question **2**

Not yet answered

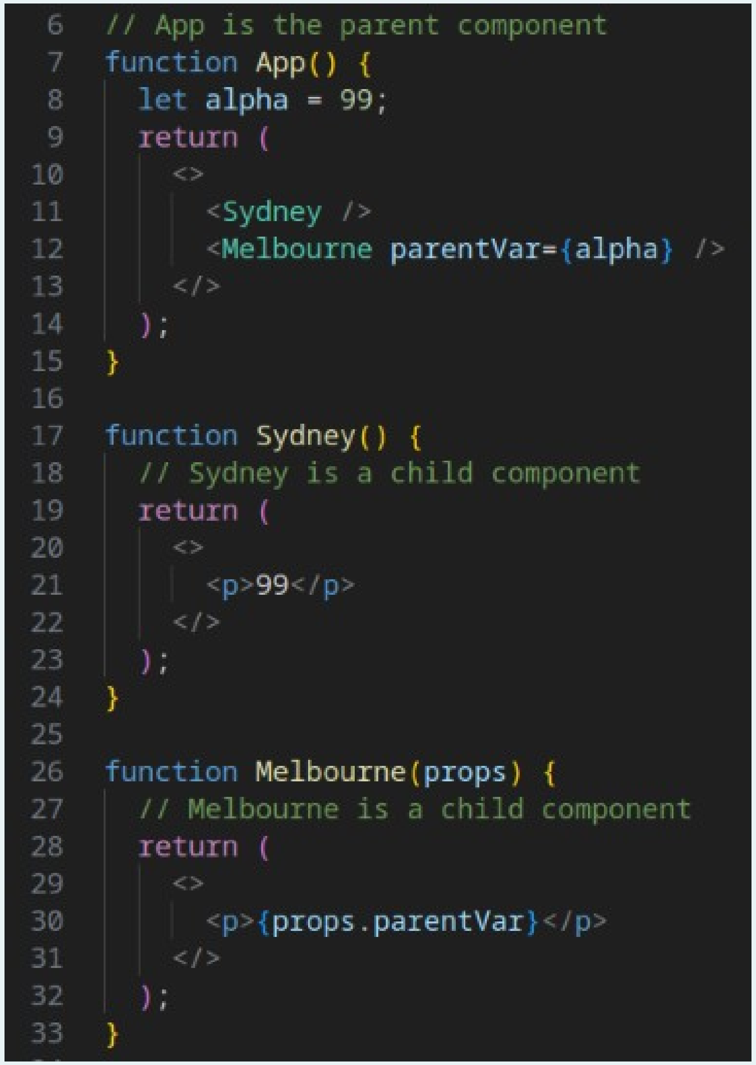
Marked out of 1.00

Flag question

Edit question

Given a fully working React application below featuring Parent-Child communication answer the following question. You can ignore formatting in the rendered output. How many times is the number 99 rendered to the screen when this application executes successfully?

Time left 1:54:16



```
// App is the parent component
function App() {
  let alpha = 99;
  return (
    <>
      <Sydney />
      <Melbourne parentVar={alpha} />
    </>
  );
}

function Sydney() {
  // Sydney is a child component
  return (
    <>
      <p>99</p>
    </>
  );
}

function Melbourne(props) {
  // Melbourne is a child component
  return (
    <>
      <p>{props.parentVar}</p>
    </>
  );
}
```

- a. 3
- b. 2
- c. 1
- d. 99 is never printed or rendered
- e. 0

CS385 Project Update

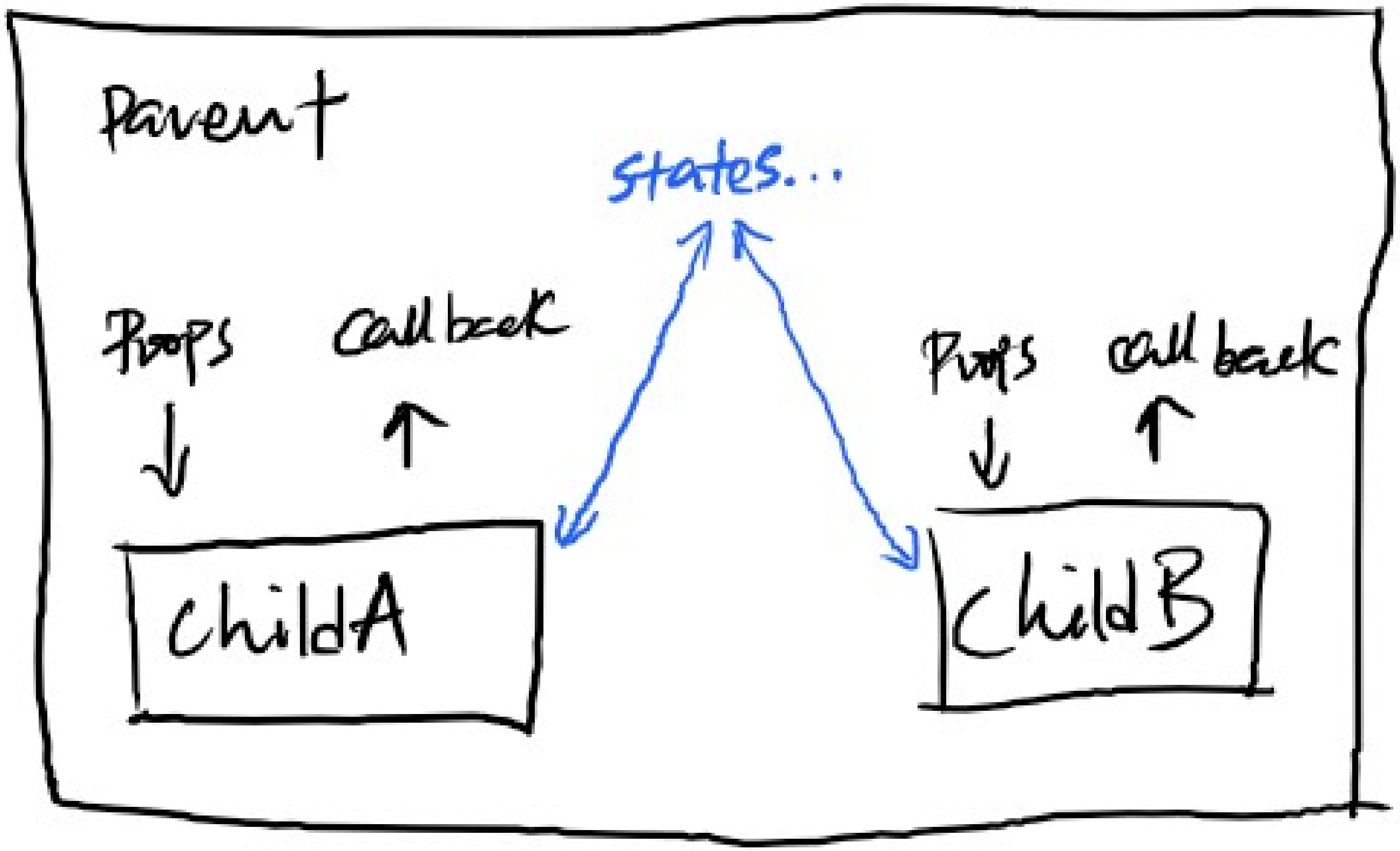
- **Are you working as a group (2, 3, or 4) or as an individual?**
- **YOU need to inform me [via email only]** of your project status before lectures on Tuesday 7th November 2023 (after mid-term break)
- One person per group can email.
- It is YOUR RESPONSIBILITY to inform me.

SCAN ME

Lab 3 – Friday 20th October 2023

- **Will strongly connect with the content from Lecture 7** (please refer to the slides)
- **Solutions to Lab 2 are now available** (after 4pm on lecture day)

CS385 Mobile Application Development (Lecture 8)



Peter Mooney