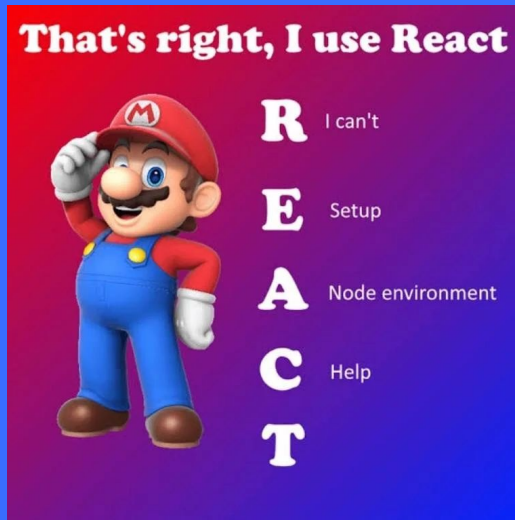
React Continued

Vincent Huang



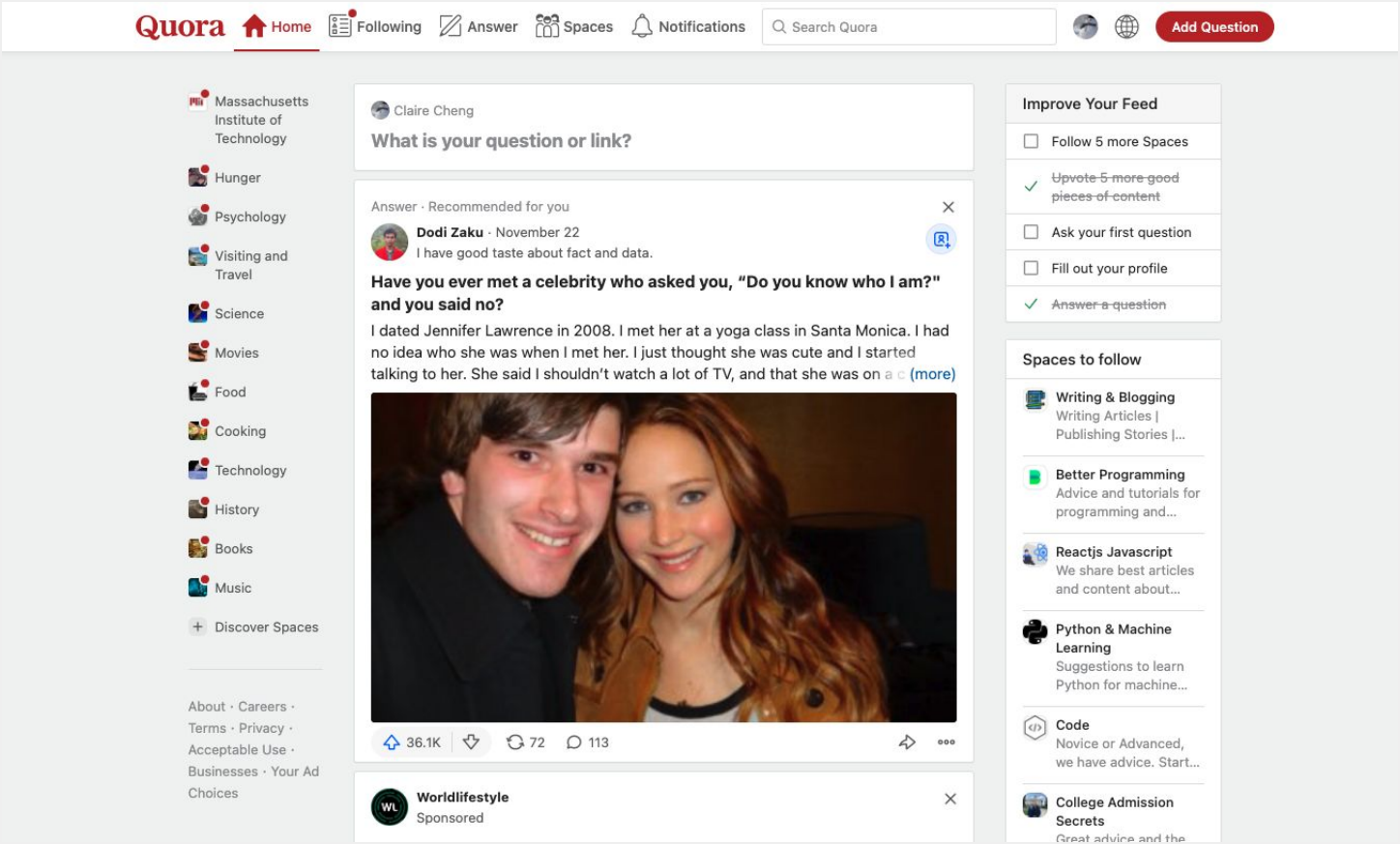
Review: Component Tree

Review: Component Tree

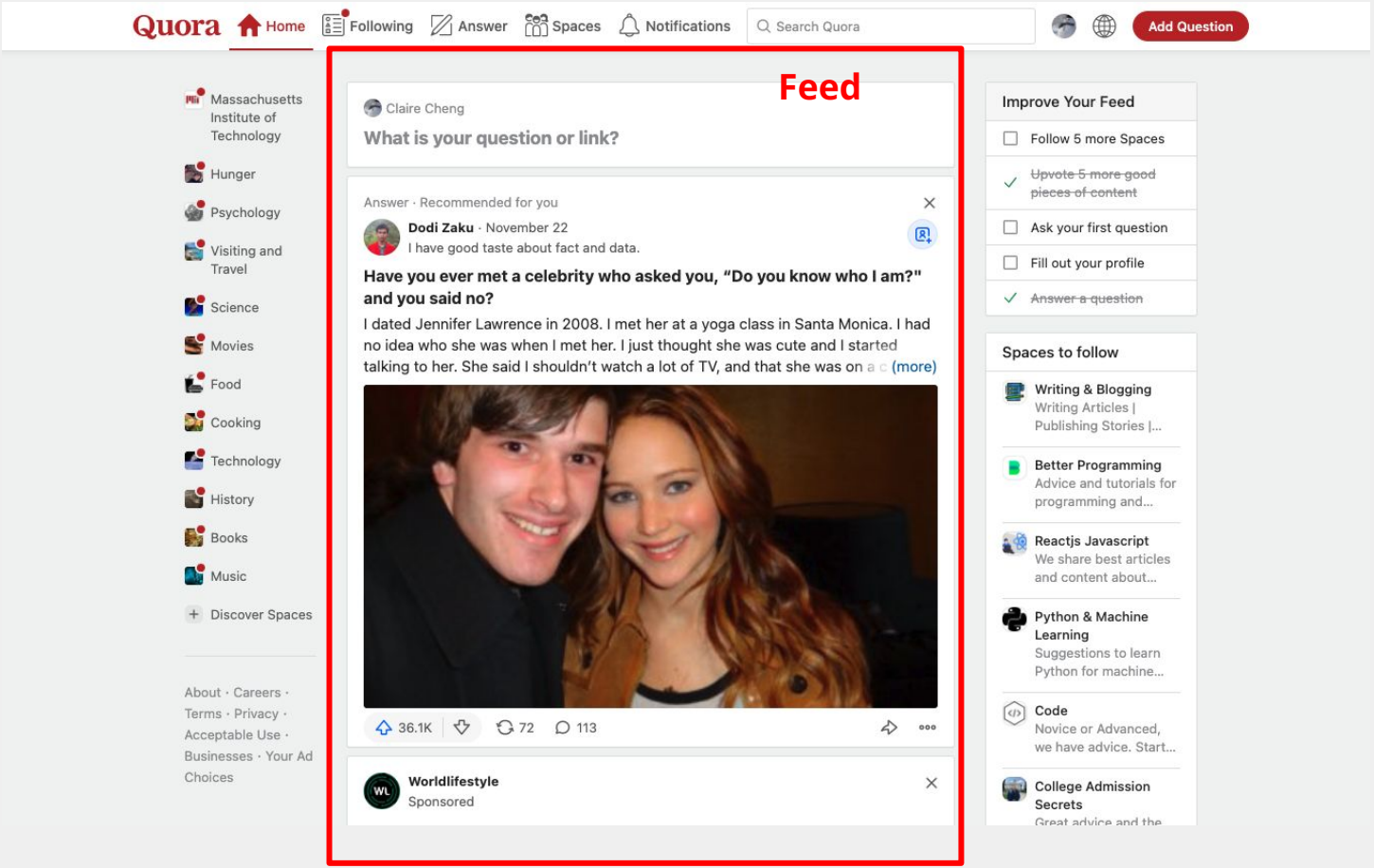
When should you break a component down into smaller parts?

- Code getting too long/hard-to-read
  - i.e. Building a house: instead of making the whole house-plan at once, you have specific plans for the bathroom, the master bedroom, the kitchen, etc.
- It conceptually “contains” parts that encapsulate their own functionality
  - i.e. In a shopping mall: each store has its own set of staff, merchandise, etc, so it would make sense to encapsulate a “store” as its own part.
- Controlling information flow between app sections!

Think... refactoring code to make your life easier as the *architect* of a web app.



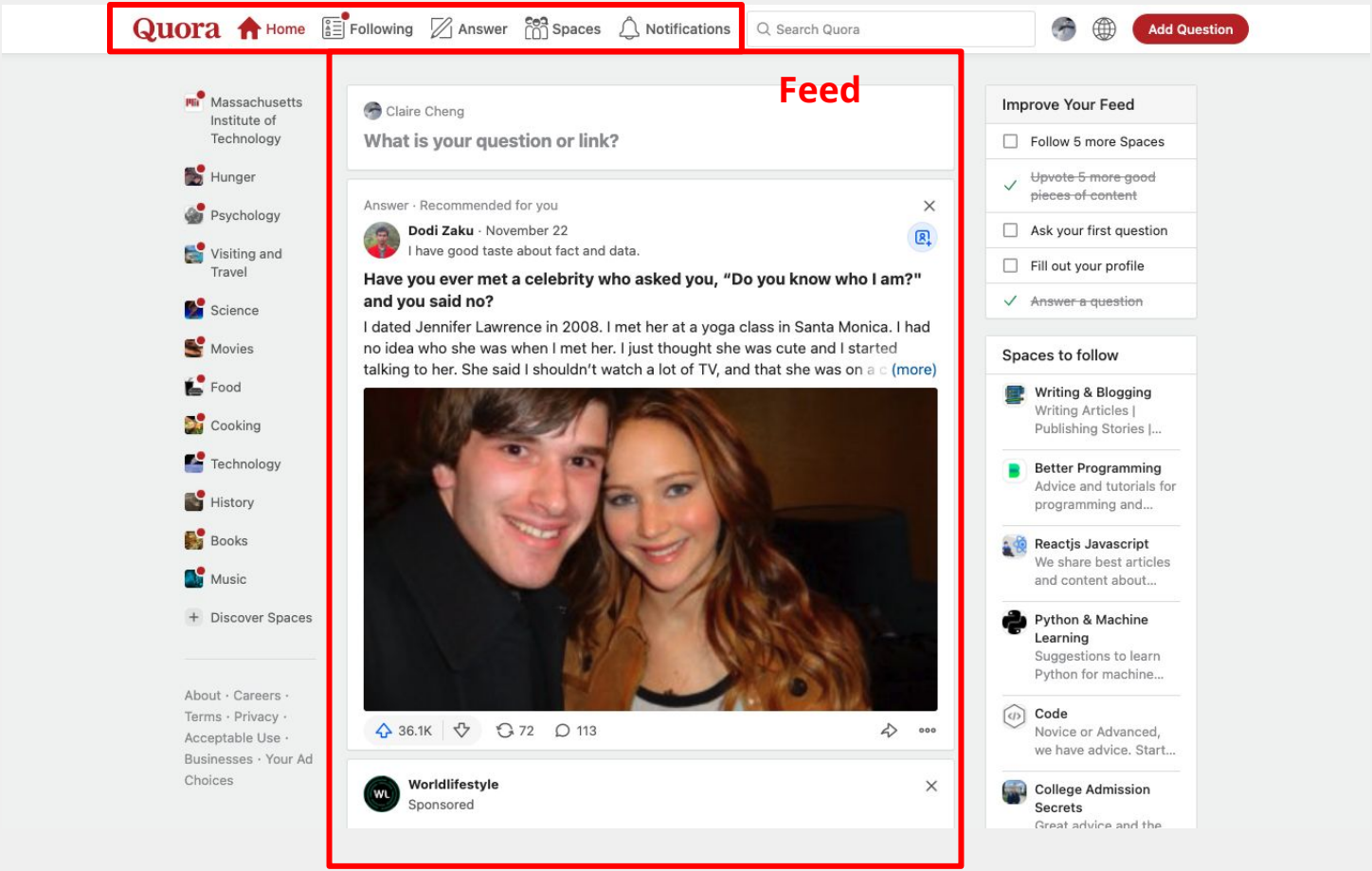
Quora: what are the underlying components?



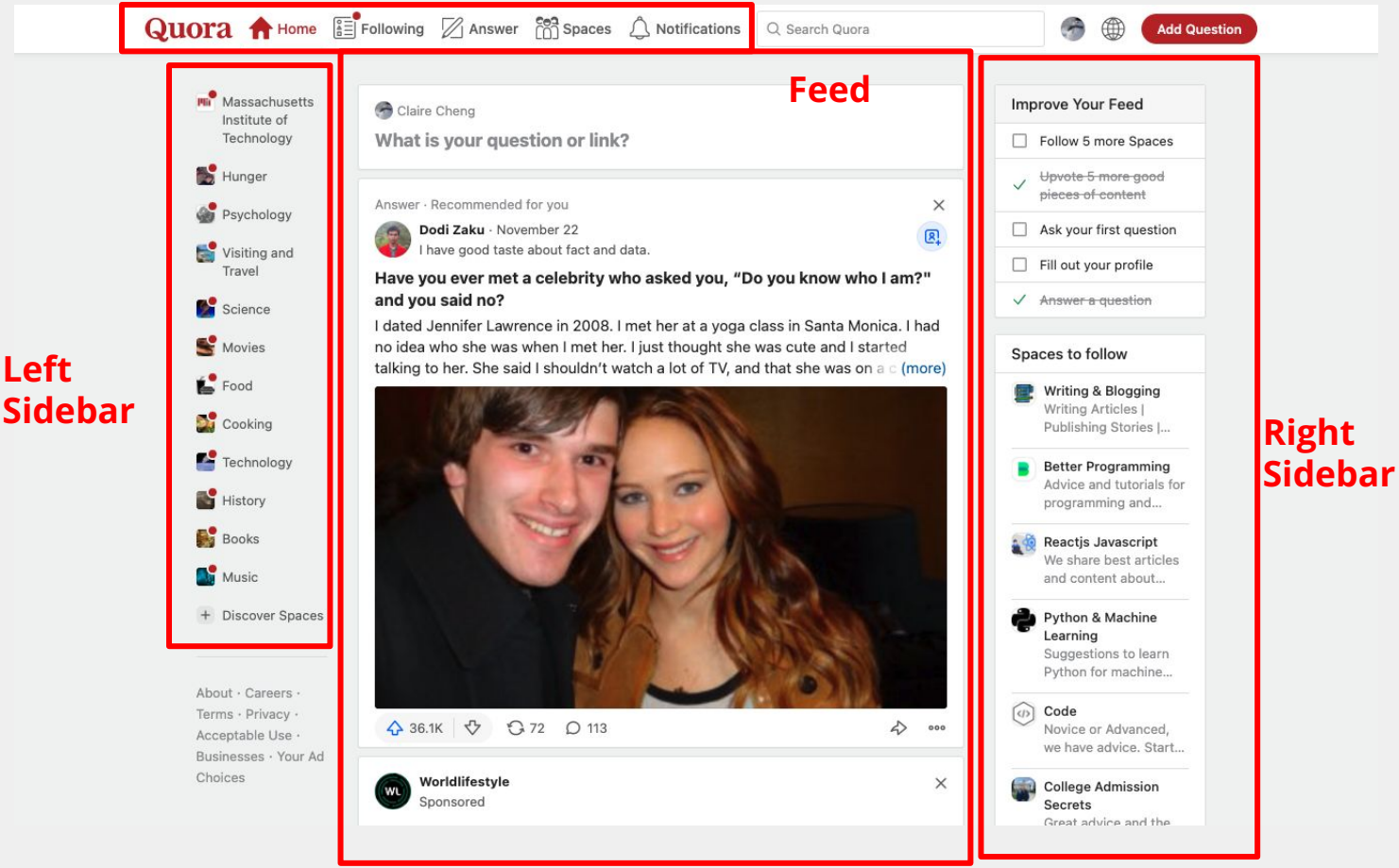
Quora: what are the underlying components?

Navbar

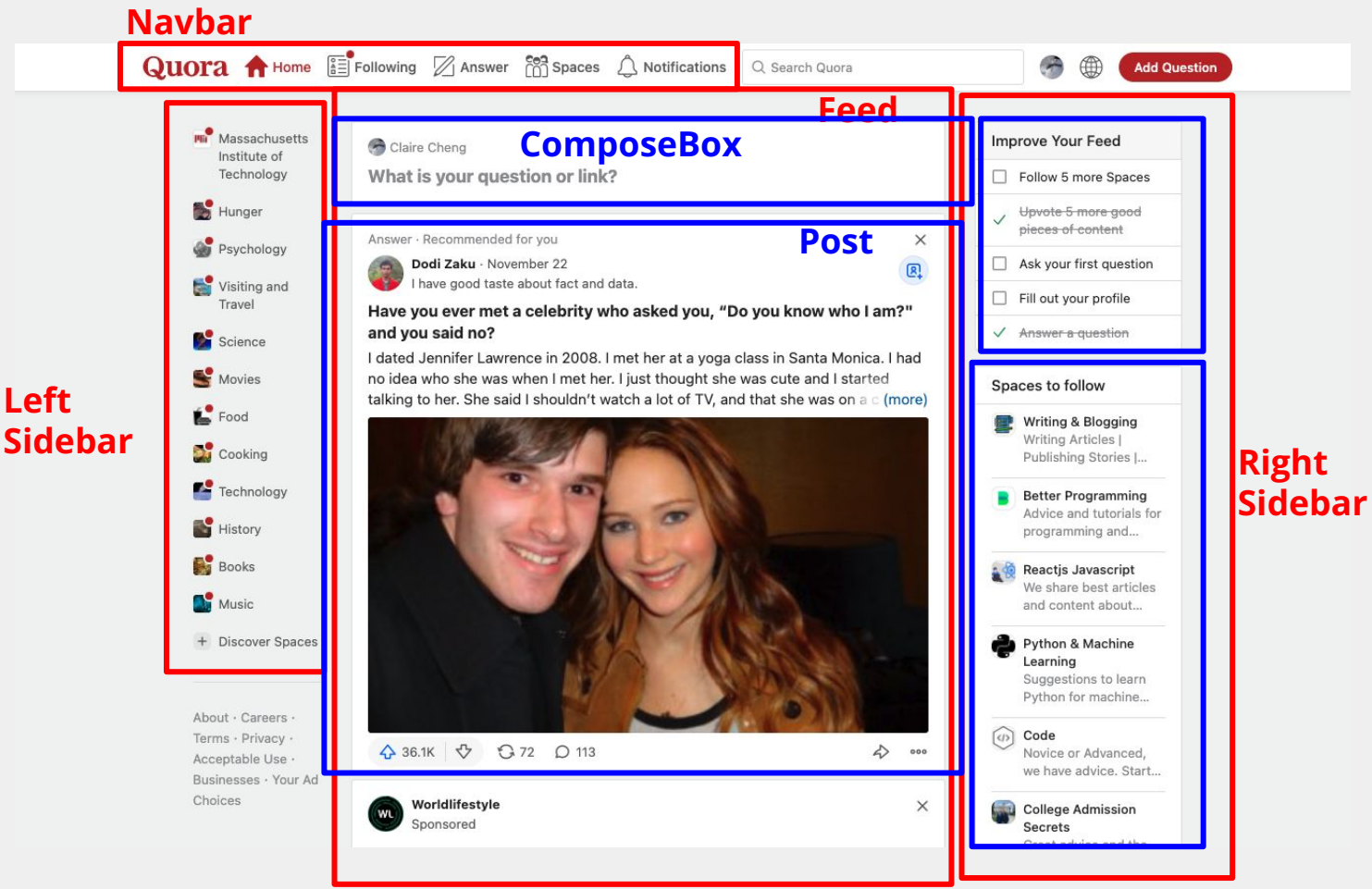
Feed



Quora: what are the underlying components?

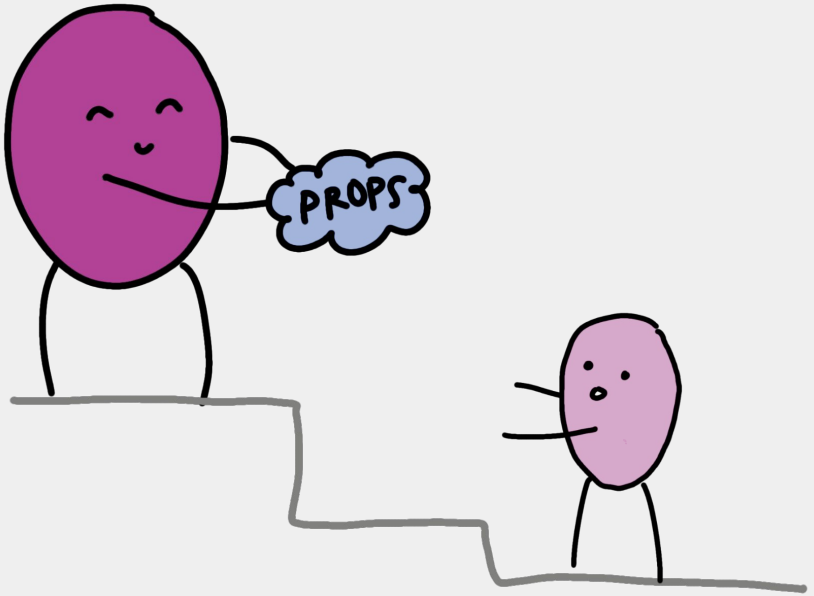


Quora: what are the underlying components?



Quora: what are the underlying components?

Review: States & Props



Important!!!!!!!!!!!!

You can’t directly edit a component’s state from another component!

A Workaround: Callback Functions

passing setState function to child and then calling it from there

```
//ParentComponent
const changeCatHappiness = () => {
     setCatHappiness(catHappiness + 1);
}


return (
     <ChildComponent updateFunction={changeCatHappiness} />
);
```

```
//ChildComponent

props.updateFunction();
```

A workaround...

passing setState function to child and then calling it from there

```
//ParentComponent
const changeCatHappiness = () => {
    setCatHappiness(catHappiness+1);
}


return (
    <ChildComponent updateFunction={changeCatHappiness} />
);
```

keyword:
WORKAROUND

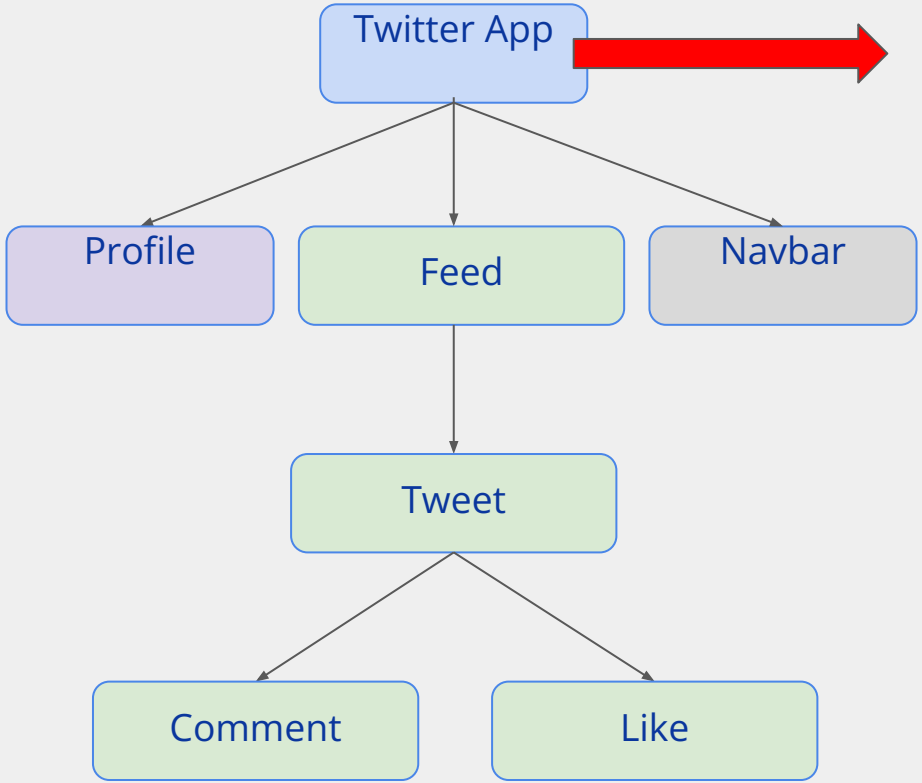
```
//ChildComponent

props.updateFunction();
```

Make states stay, make props pass

Make **states stay**, make **props pass**
(downwards)

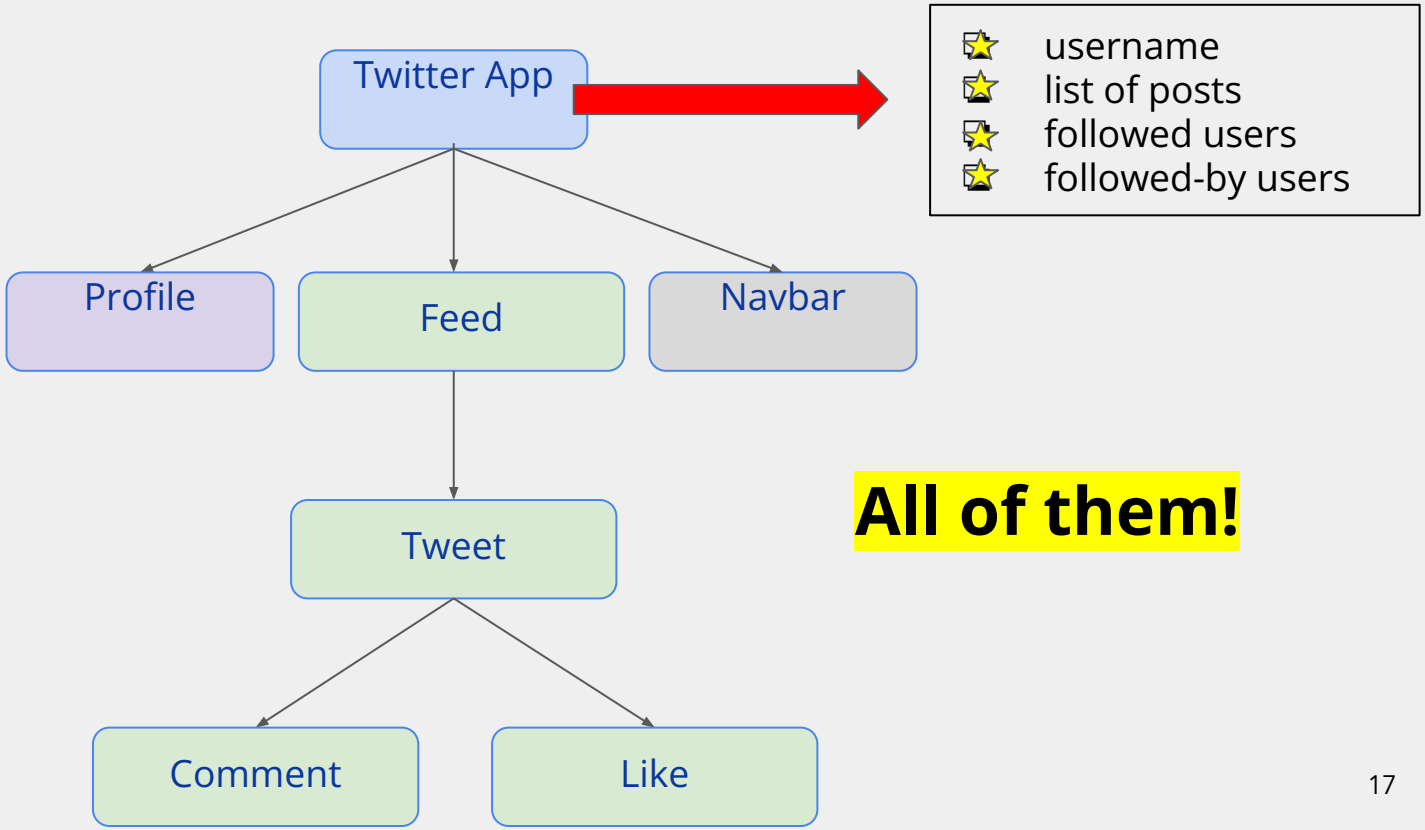
props practice

Which of these should be props?



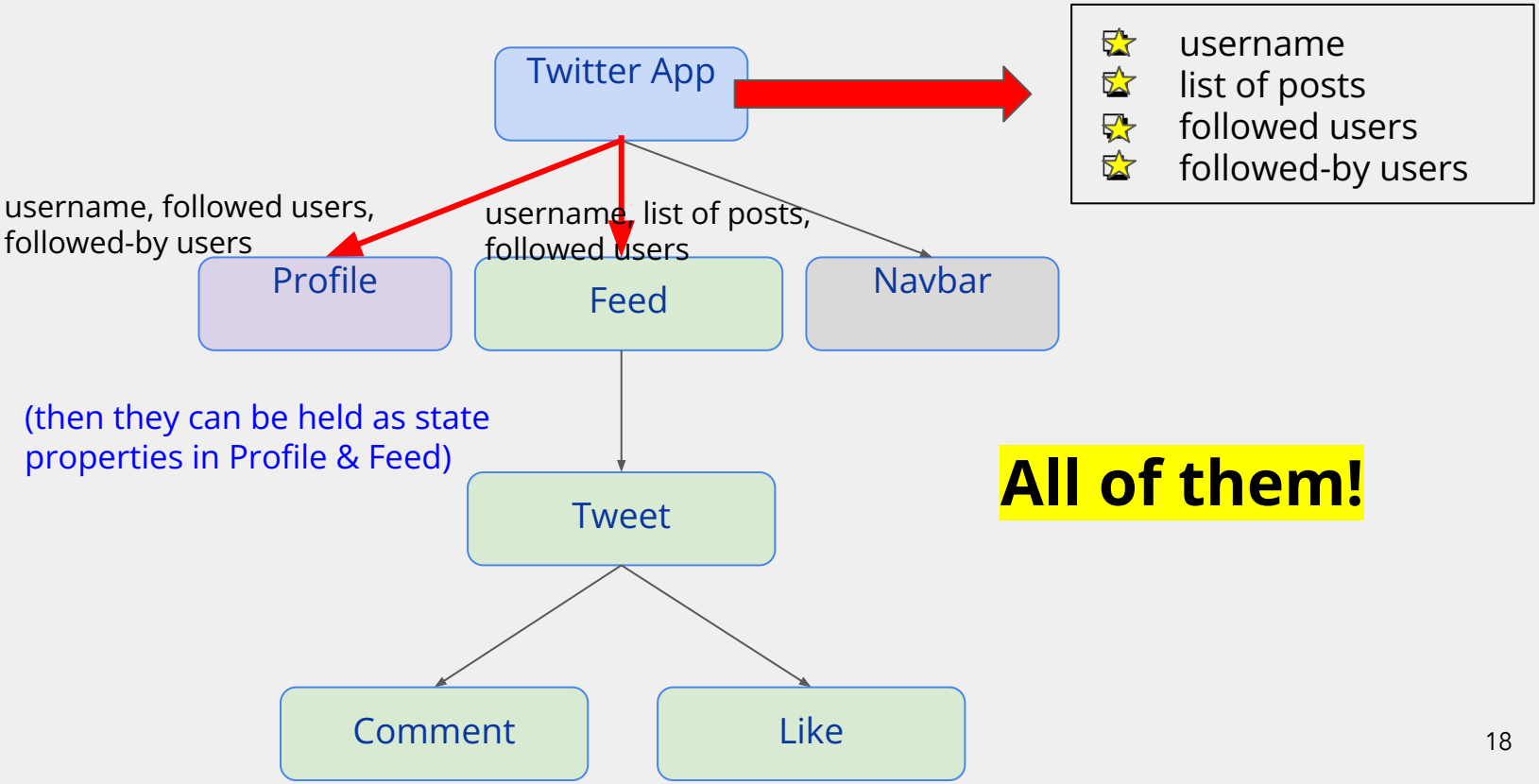
- ❏ username
- ❏ list of posts
- ❏ followed users
- ❏ followed-by users

Which of these should be props?



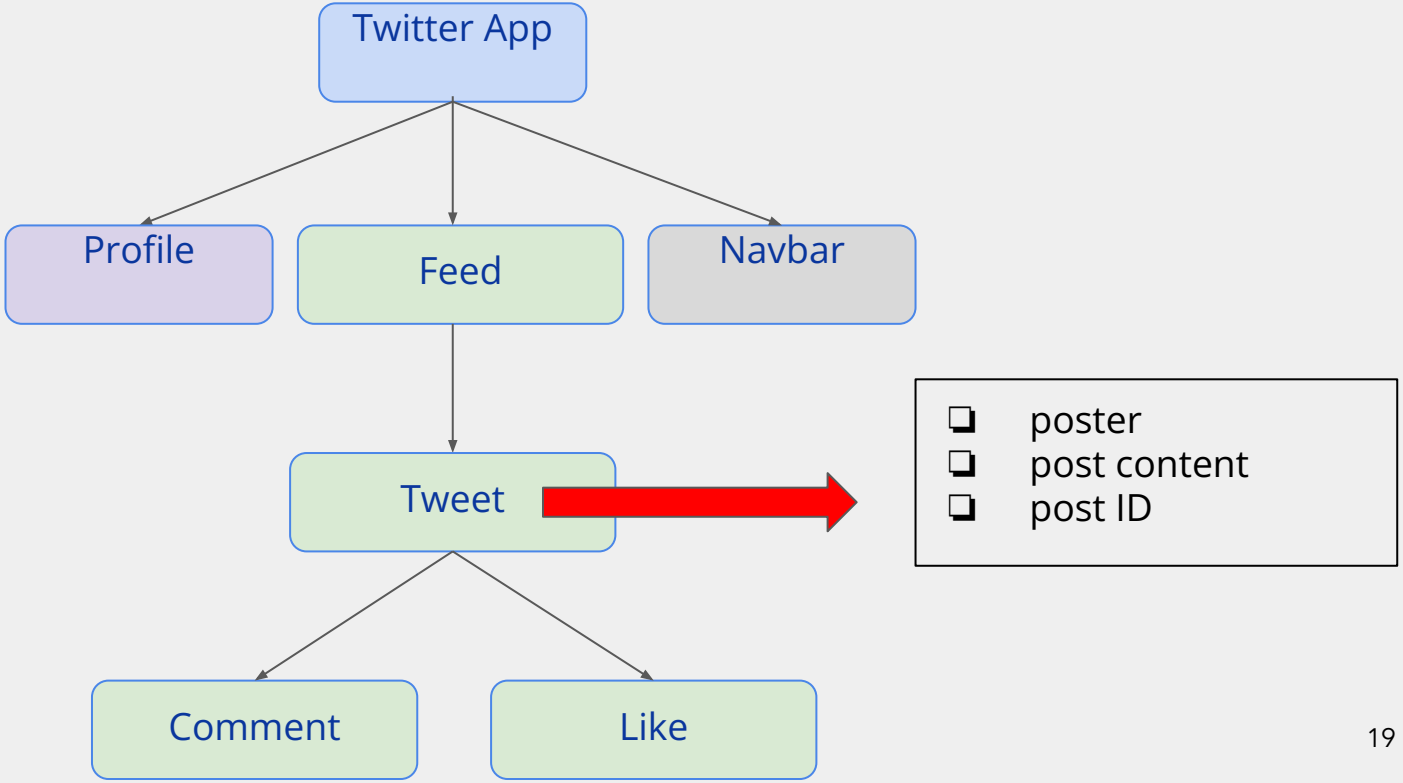
All of them!

Which of these should be props?

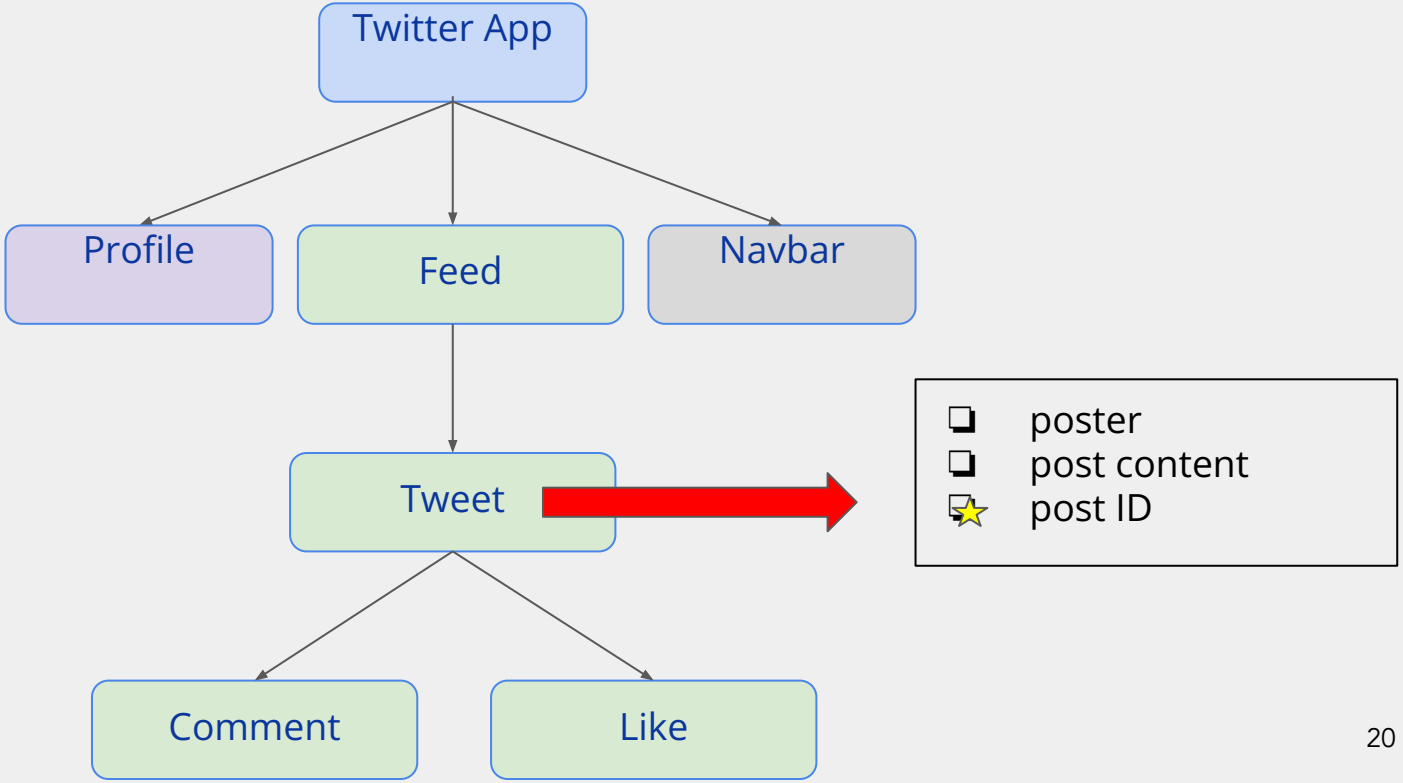


All of them!

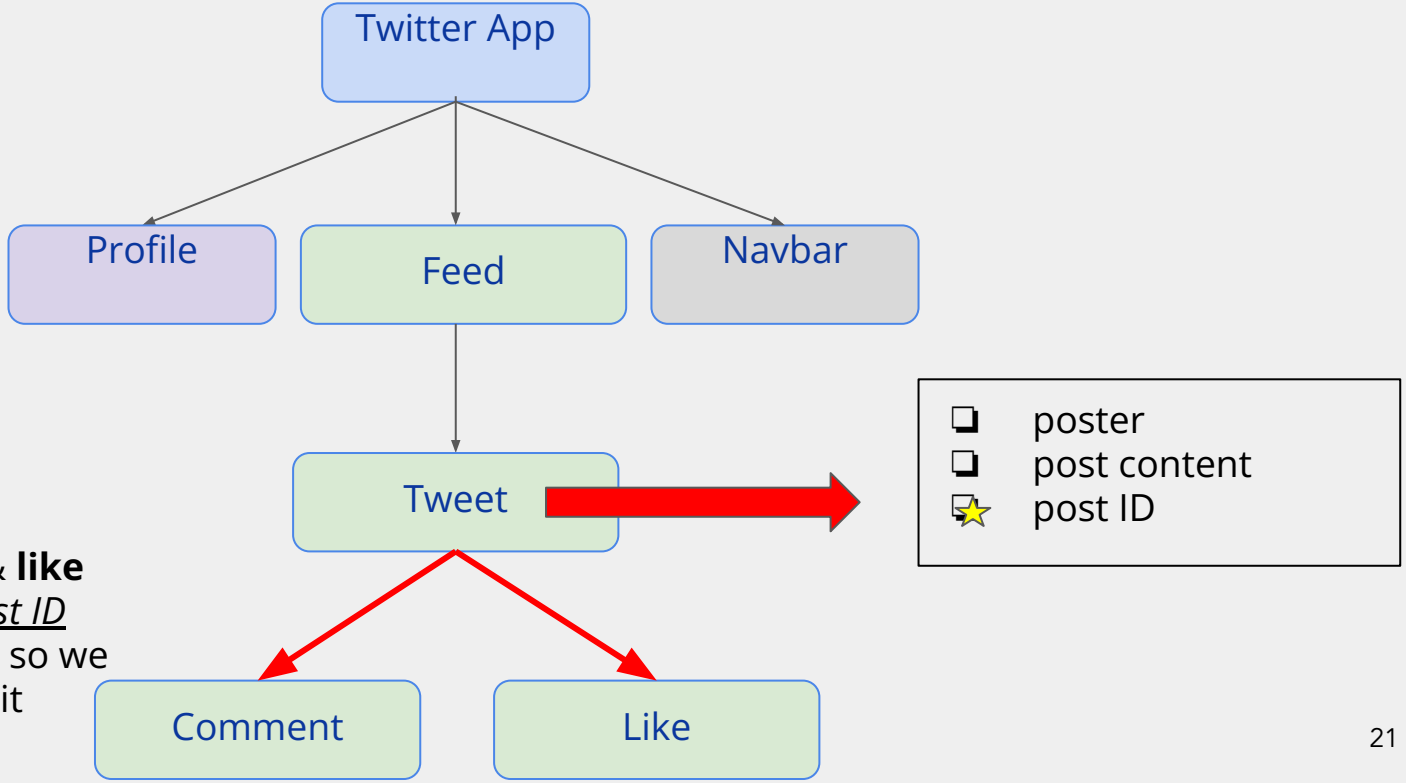
Which of these should be props?



Which of these should be props?



Which of these should be props?



each **comment** & **like** should have a *post ID* associated with it so we know which post it belongs to!

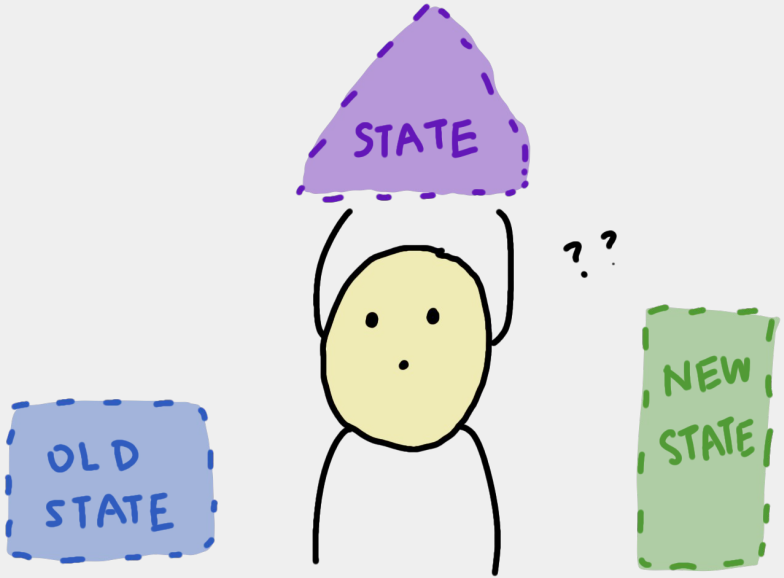
now… more on state

In-depth: Utilizing state

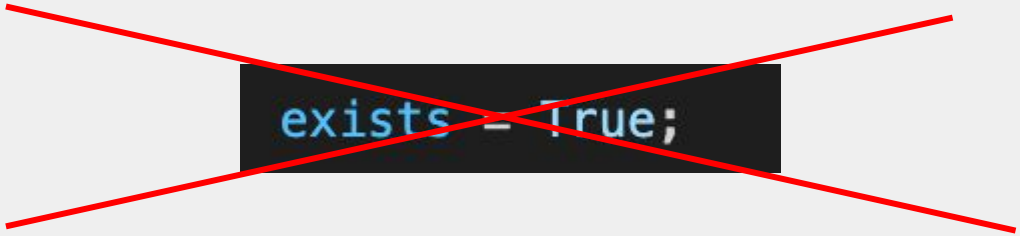
Scenario: Initializing state

```
const MyHouse = (props) => {

    const [persons, setPersons] = useState([]);
    const [pets, setPets] = useState([]);
    const [exists, setExists] = useState(false);
```

In-depth: how to modify state



Modifying state → setting state



In-depth: Utilizing state

Scenario: Setting new values

```
setExists(true);
```

In-depth: Utilizing state

Scenario: Modifying original state

```
buyDog = () => {
    setPets(pets.concat["dog"]);
}
```

In-depth: Utilizing state

Scenario: Alternative way to update state with object destructuring (generally recommended)

```
buyDog = () => {
    setPets([...pets, "dog"]);
}
```

What is this function going to print?

```
testingStuff = () => {
    /* assume persons starts as an empty array [] */
    setPersons([...persons, "me"]);
    console.log(persons);
}
```

A. []

B. [“me”]

What is this function going to print?

```
testingStuff = () => {
    /* assume persons starts as an empty array [] */
    setPersons([...persons, "me"]);
    console.log(persons);
}
```

NO GUARANTEE!

A. []

B. [“me”]

Setting State is *ASYNC*.

What if we want to do something immediately after state is changed?

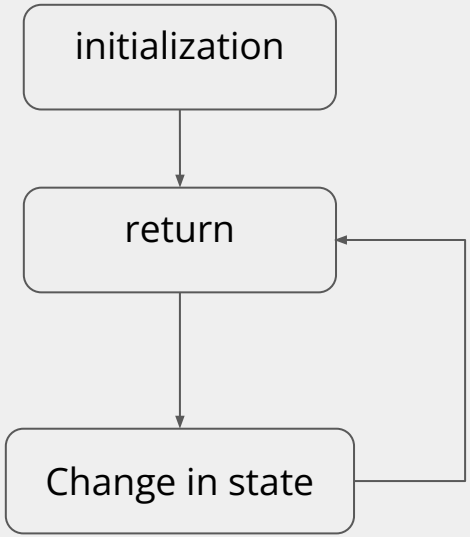
eg. add someone to persons and then print the persons array

We need a new React feature for this:
the useEffect hook!

But first...

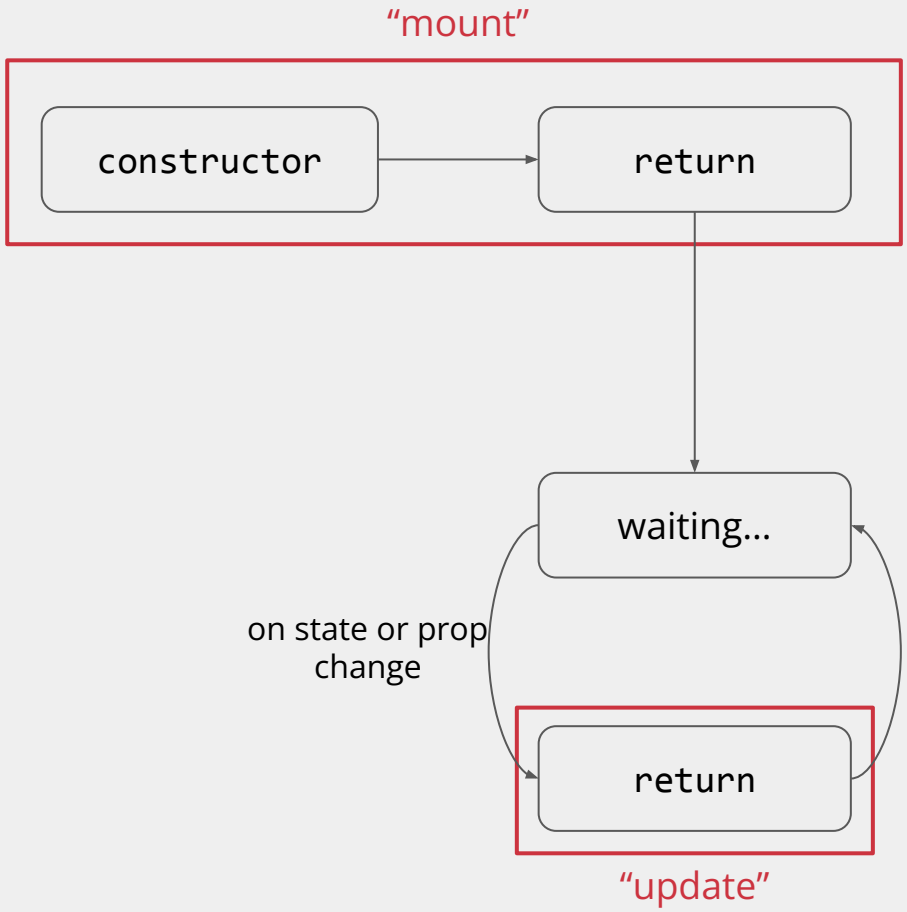
Review: Lifecycle Timeline

Lifecycle Timeline

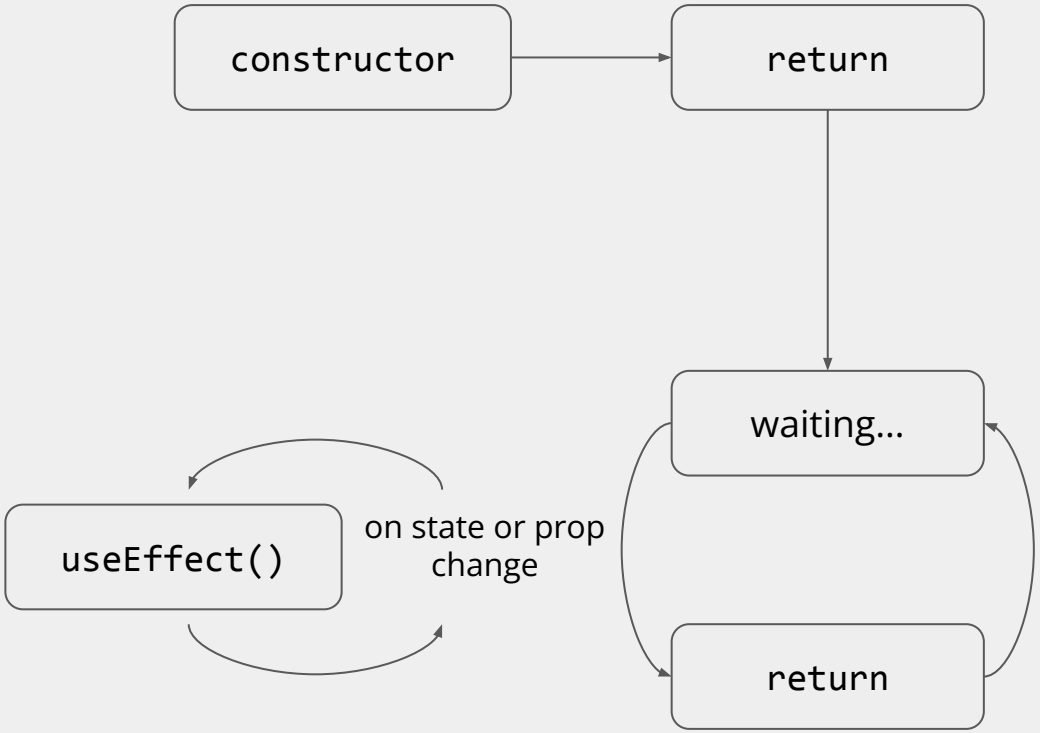


(continue to render after every state variable change)

Lifecycle Timeline



Lifecycle Timeline



`useEffect()` hook

- Runs after specific variables change
  - Can think of as response to state change
  - Commonly used to load data into state
  - Call an API/perform some computation/etc. at specific times

- syntax: `useEffect(function,` *optional* `dependency array)`
  - runs `function` every time a variable in `dependency array` changes

useEffect() example

```
testingStuff = () => {
    /* assume persons starts as an empty array [] */
    setPersons([...persons, "me"]);
    console.log(persons);
}
```

```
testingStuff = () => {
    setPersons([...persons, "me"]);
}

useEffect(() => {
    console.log(persons);
}, [persons])
```

`useEffect()` hook - dependency array examples

- `useEffect(function, [var1, var2])` runs `function` every time `var1` or `var2` changes
- `useEffect(function, [])` runs `function` once, immediately after initialization
- `useEffect(function)` runs every time *any* state variable changes

`get()` and `post()`

Nitty gritty details: utilities.js

```
get("/api/foo", { bar: 0 }).then((res) => console.log(res));
```

Fetching data

```
useEffect(() => {
    get("/api/packages").then((packageList) => {
        setPackages(packageList);
    })
}, [])
```

Sending data

```
<Button onClick={handleToggleAdmin} />
```

```
const handleToggleAdmin = () => {
    post("/api/user/admin", {admin: !admin}).then(() => {
        setAdmin(!admin);
    })
}
```

Review: JSX

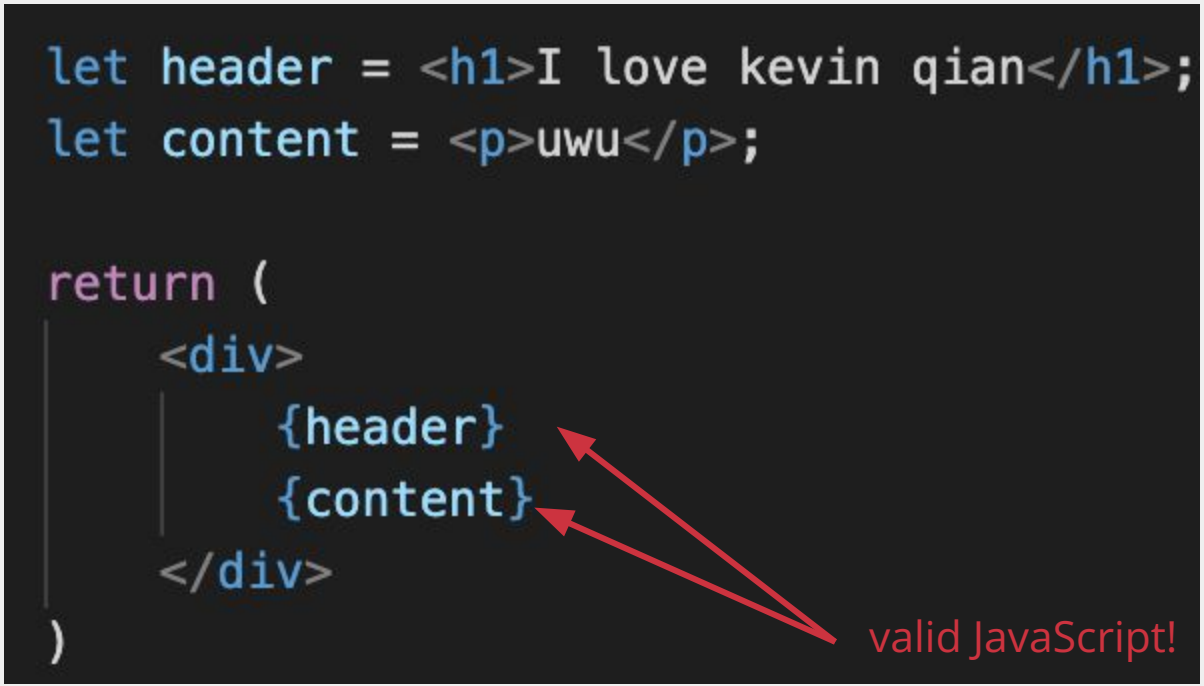
Just a way to write HTML onto the page, but from JavaScript!

```
const element = <h1> Follow me at http://twitter.com/vvhuang_ </h1>;
```

Example: JSX for multiple pieces of content

```
let header = <h1>I love kevin qian</h1>;
let content = <p>uwu</p>;

return (
    <div>
        {header}
        {content}
    </div>
)
```

Example: JSX for multiple pieces of content



Example: display a loading indicator

Some state indicating if the page is still loading data or not

Show “Loading…” while page is loading data, otherwise the content

Conditional rendering

```
return loading ? (
  <div>
    <h1>Title</h1>
    <p>Loading... Buy bitcoin while waiting for the page to load</p>
  </div>
) : (
  <div>
    <h1>Title</h1>
    <p>Actual page content</p>
  </div>
);
```

```
condition ? resultIfTrue : resultIfFalse
```

More modular conditional rendering

```
let content = loading ? (
  <p>Loading... Buy bitcoin while waiting for the page to load</p>
) : (
  <p>Actual page content</p>
);

return (
  <div>
    <h1>Title</h1>
    {content}
  </div>
);
```

Concise conditional rendering

```
return (
  <div>
    <h1>Title</h1>
    <p>
      {loading
        ? "Loading... Buy bitcoin while waiting for the page to load"
        : "Actual page content"}
    </p>
  </div>
);
```

Loop rendering

- Common pattern: render an array of data in React
- Need a simple way to *map* data to HTML

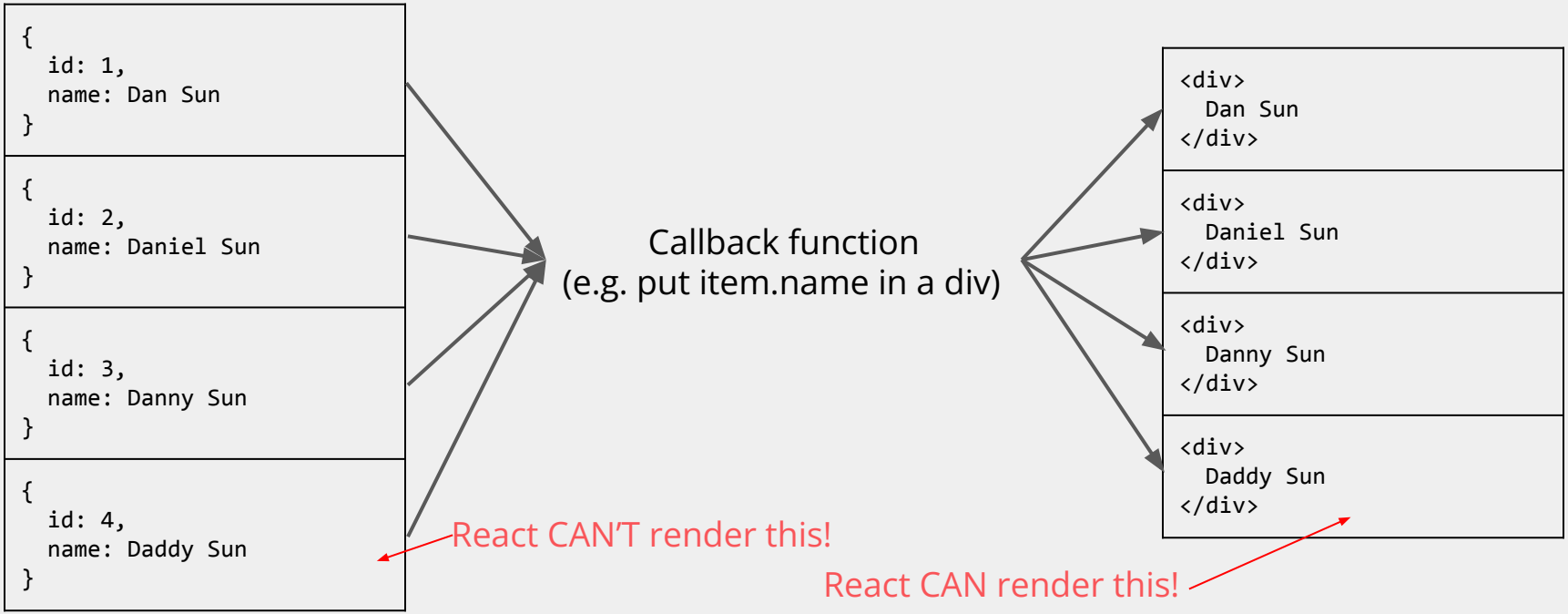
Review: `map()`

Creates a new array by applying the callback function to every element of the starting array.

```
let myArray = [1, 2, 3, 4, 5];
let modifiedArray = myArray.map(x => x * 3);
// modifiedArray === [3, 6, 9, 12, 15]
```

```
const celsiusToFahrenheit = tempC => {
    const tempF = tempC * 1.8 + 32;
    return tempF;
}

let celsius = [-40, -20, 0, 20, 40];
let fahrenheit = celsius.map(celsiusToFahrenheit);
// fahrenheit === [-40, -4, 32, 68, 104]
```

Using `map()` in React



Example: HTML

```
let data = [
  { id: 0, text: "My favorite food is doritos" },
  { id: 1, text: "I like nachos because they look like doritos" },
];
return data.map((item) => {
  return <div key={item.id}>{item.text}</div>;
});
//<>
//<div key={0}>My Favorite food is doritos</div>
//<div key={1}>I like nachos because they look like doritos</div>
//</>
```

Example: Components

```
let data = [
  { id: 0, text: "My favorite food is doritos" },
  { id: 1, text: "I like nachos because they look like doritos" },
];
return data.map((item) => {
  return <ItemComponent key={item.id}>{item.text}</ItemComponent>;
});
//<>
//<ItemComponent key={0}>My Favorite food is doritos</ItemComponent>
//<ItemComponent key={1}>I like nachos because they look like doritos</ItemComponent>
//</>
```

Example: Components (more concise)

Don't need to `return` if you skip the curly braces — otherwise you do!

```
let data = [
  { id: 0, text: "My favorite food is doritos" },
  { id: 1, text: "I like nachos because they look like doritos" },
];
return data.map((item) =>
  <ItemComponent key={item.id}>{item.text}</ItemComponent>
);
//<>
//<ItemComponent key={0}>My Favorite food is doritos</ItemComponent>
//<ItemComponent key={1}>I like nachos because they look like doritos</ItemComponent>
//</>
```

Questions?

Oh wait we still have extra time…
Let's talk more about JS and React!

(but feel free to zone out if you need a break)

Math

As with many languages, JS has a Math object with helpful methods:

- `Math.abs()`
- `Math.min(), Math.max()`
- `Math.PI, Math.sin()`
- etc.

Dates

JS has built-in Date objects that will save your life when making certain kinds of apps (eg. anything with a calendar):

```
let bday = new Date(1912, 10, 30) // Nov 30
let nextDay = new Date(bday);
let nextDay.setDate(nextDay.getDate() + 1); // nextDay is Dec 1
```

Sets

- Arrays and objects should be enough for storing most of your site data
- Sometimes you'll also want to use sets, which only store distinct values.

```
let s = new Set()
let s.add(1)
let s.add(2)
let s.add(1) // s = {1, 2}
```

JSON

Convert objects into strings and vice versa!

```
let x = {1: 2, foo: {uwu: 'owo'}}
let y = JSON.stringify(x) // y is '{"1":2,"foo":{"uwu":"owo"}}'
let z = JSON.parse(y) // retrieves x
```

- Useful if you need to store info as a string (eg. in a database) and then retrieve it later
- Our utilities.js get/post request code does this

Other React info if you're bored

Hooks reference: https://reactjs.org/docs/hooks-reference.html

- React provides other hooks besides `useState` and `useEffect` to support features like global context and memoization. (We won't cover these because they're used much less often.)

OK now we’re actually done