STATE AND PROPS

Like peanut butter and more peanut butter

TRAJECTORY

- **Reusing components with props**
- **Unidirectional data flow via props**
- **Class components vs. functional components**

TWO WAYS TO WRITE A COMPONENT

CLASS

(we won’t cover class components,
but they’re good to be familiar with)

```
class Pizza extends React.Component {
  render () {
    return <div>Pizza Pie!</div>
  }
}
```

FUNCTION

```
const Pizza = () => {
  return <div>Pizza Pie!</div>
}
```

CLASS

```
class Pizza extends React.Component {
  render () {
    return <div>Pizza Pie!</div>
  }
}
```

FUNCTION

```
const Pizza = () => {
  return <div>Pizza Pie!</div>
}
```

Your favorite pizza topping is: Cheese

Cheese

Broccoli

Anchovies

Your favorite pizza topping is: Broccoli

Cheese

Broccoli

Anchovies

```html
<div>
  <h1>Your favorite pizza topping is: ???</h1>
  <ul>
    <li>Cheese</li>
    <li>Broccoli</li>
    <li>Anchovies</li>
  </ul>
</div>
```

```
<ToppingList>
   {/* ingredients go here… */}
</ToppingList>
```

```
<ToppingList>
  <Cheese />
  <Broccoli />
  <Anchovies />
</ToppingList>
```

```
const Cheese = () => {
```

```
  return <li>Anchovies</li>
}
```

```
const Topping = (props) => {
  return <li>{props.type}</li>
}
```

```
<ToppingList>
  <Topping type="cheese" />
  <Topping type="broccoli" />
  <Topping type="anchovies" />
</ToppingList>
```

PROPS

- Conceptually and syntactically very similar to an HTML *attribute*
- All props that are passed into a component become key-value pairs on that component's "props" object

“UNIDIRECTIONAL DATA FLOW”

UNIDIRECTIONAL DATA FLOW

- **We view our UI as a hierarchy of components**
  - Which is intuitive - we already think of HTML this way

- **The big difference: our state is also communicated via that hierarchy**

- **Means of communication: passing down props to components**

Your favorite pizza topping is: Cheese

Cheese

Broccoli

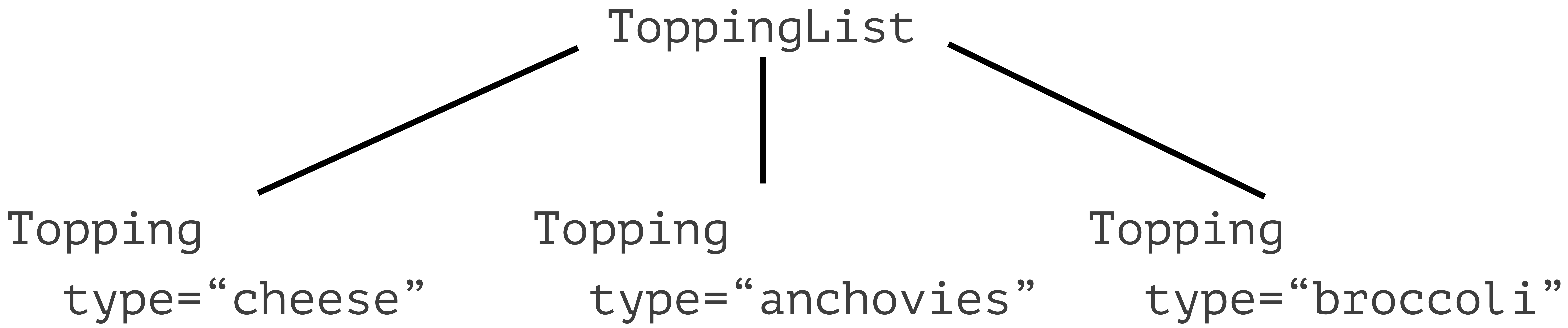
Anchovies

Your favorite pizza topping is: Broccoli

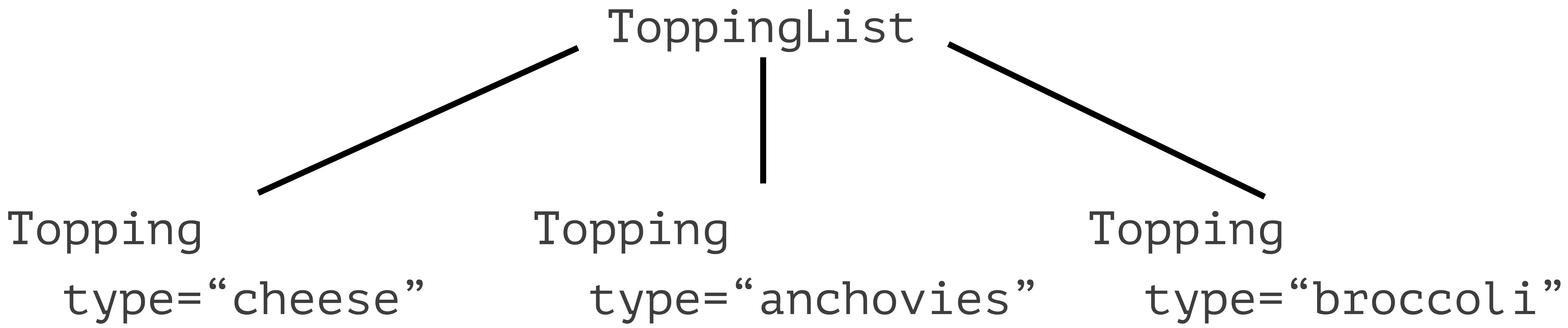
Cheese

Broccoli

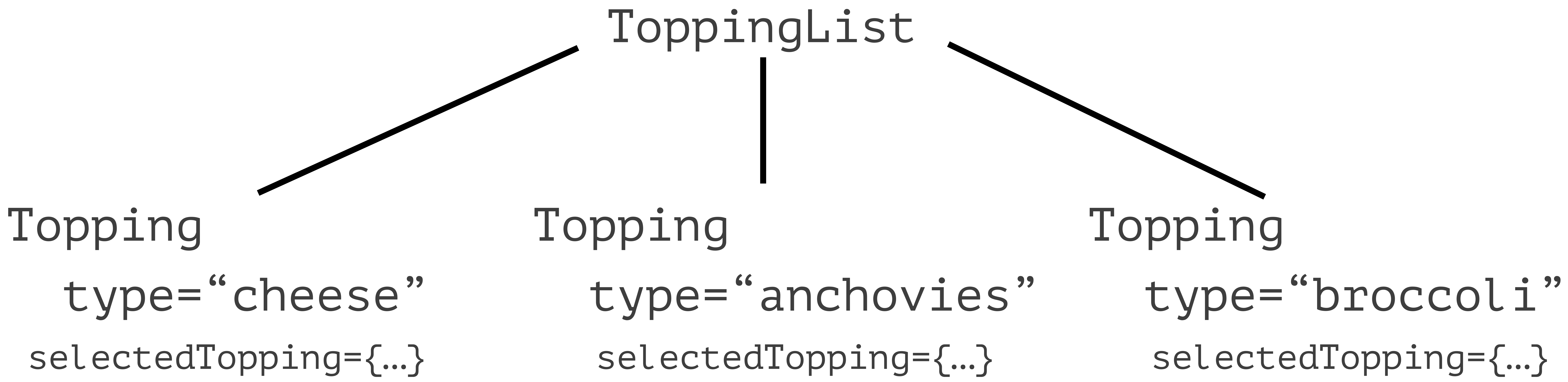
Anchovies


ToppingList
Topping
type="cheese"
Topping
type="anchovies"
Topping
type="broccoli"

```
const [selectedTopping, setSelectedTopping] = useState('cheese');
```



```
const [selectedTopping, setSelectedTopping] = useState(‘cheese’);
```

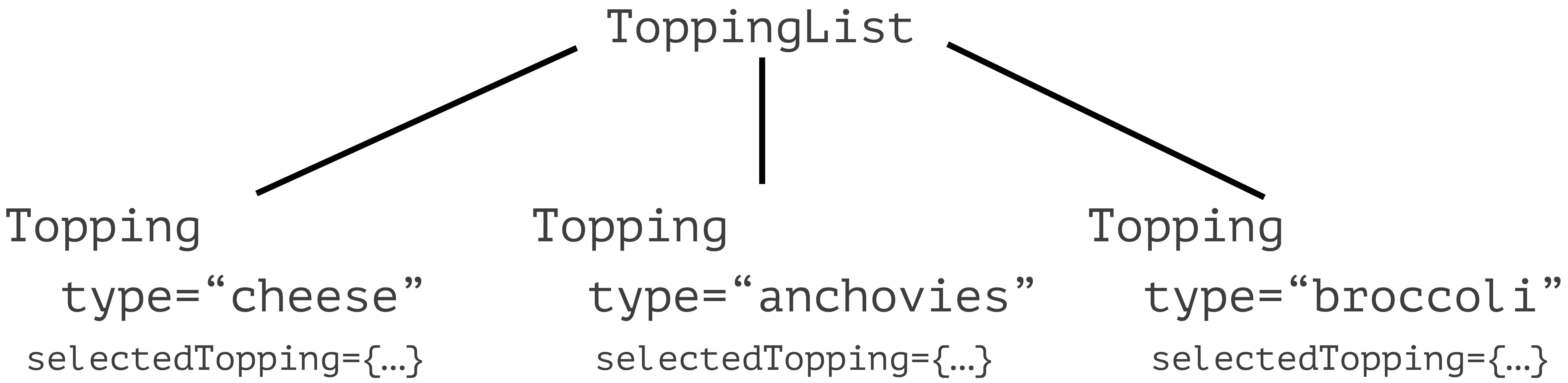


```
const ToppingList = () => {
  const [selectedTopping, setSelectedTopping] = useState(‘cheese’);

  return (
    <div>
      <h1>Your Favorite Topping is: {selectedTopping}</h1>
      <ul>
        <Topping selectedTopping={selectedTopping} type=“cheese” />
        <Topping selectedTopping={selectedTopping} type=“broccoli” />
        <Topping selectedTopping={selectedTopping} type=“anchovies” />
      </ul>
    </div>
  )
}
```

```
class ToppingList extends React.Component {
  constructor () {
    super()
    this.state = {
      selectedTopping: 'cheese'
    }
  }

  render () {
    return (
      <div>
        <h1>Your favorite topping is: {this.state.selectedTopping}</h1>
        <ul>
          <Topping selectedTopping={this.state.selectedTopping} type='cheese' />
          <Topping selectedTopping={this.state.selectedTopping} type='broccoli' />
          <Topping selectedTopping={this.state.selectedTopping} type='anchovies' />
        </ul>
      </div>
    )
  }
}
```

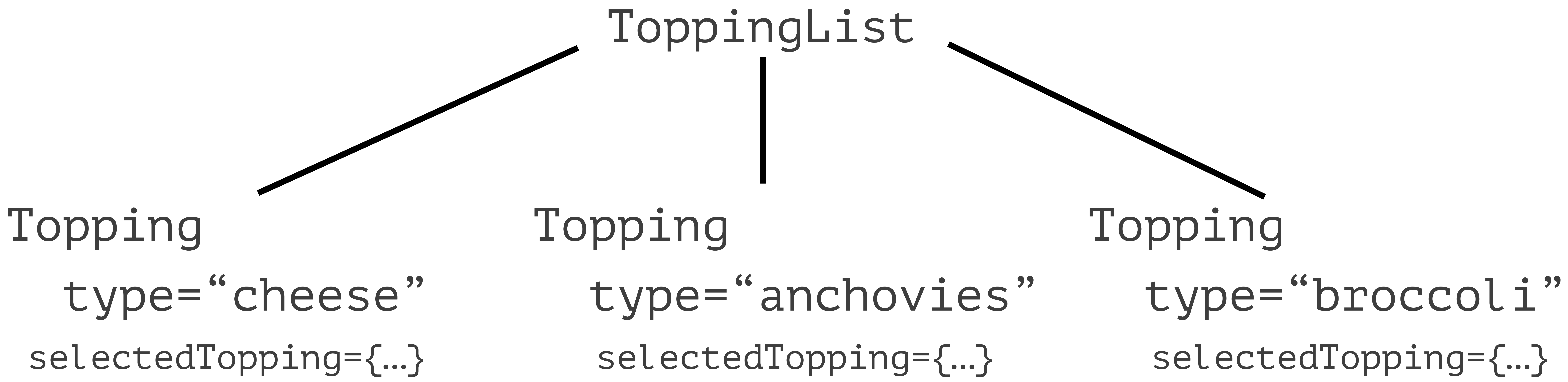
```jsx
const Topping = (props) => {
  const isSelected = props.selectedTopping === props.type
  return (
    <div className={isSelected && 'selected'}>{props.type}</div>
  )
}
```

```
const [selectedTopping, setSelectedTopping] = useState('cheese');
```



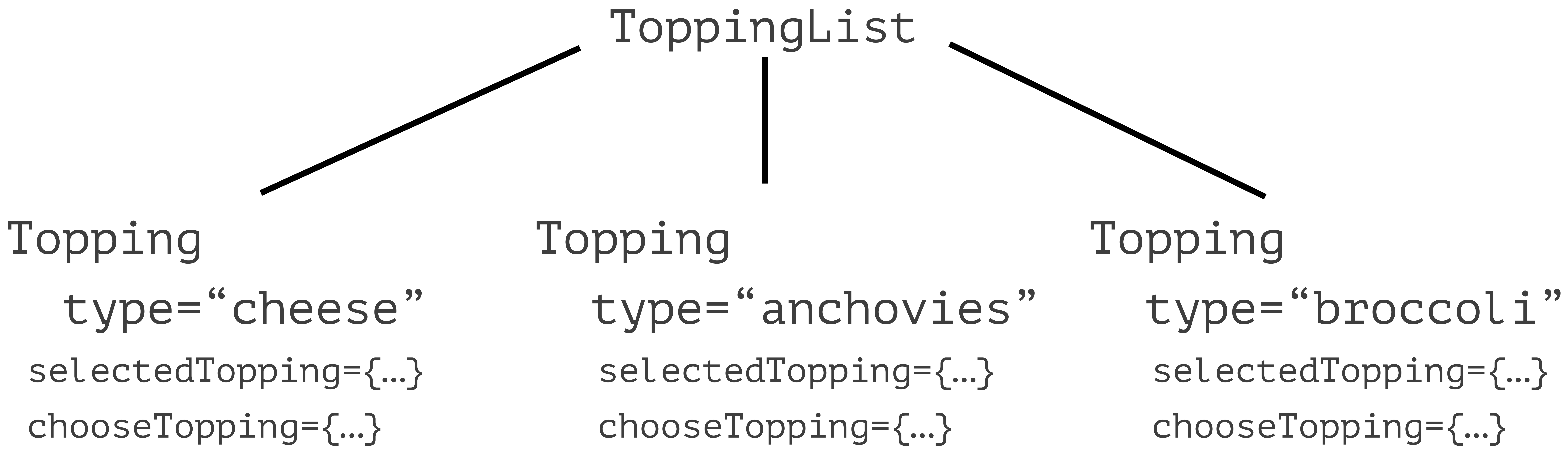
```
const [selectedTopping, setSelectedTopping] = useState('cheese');
```

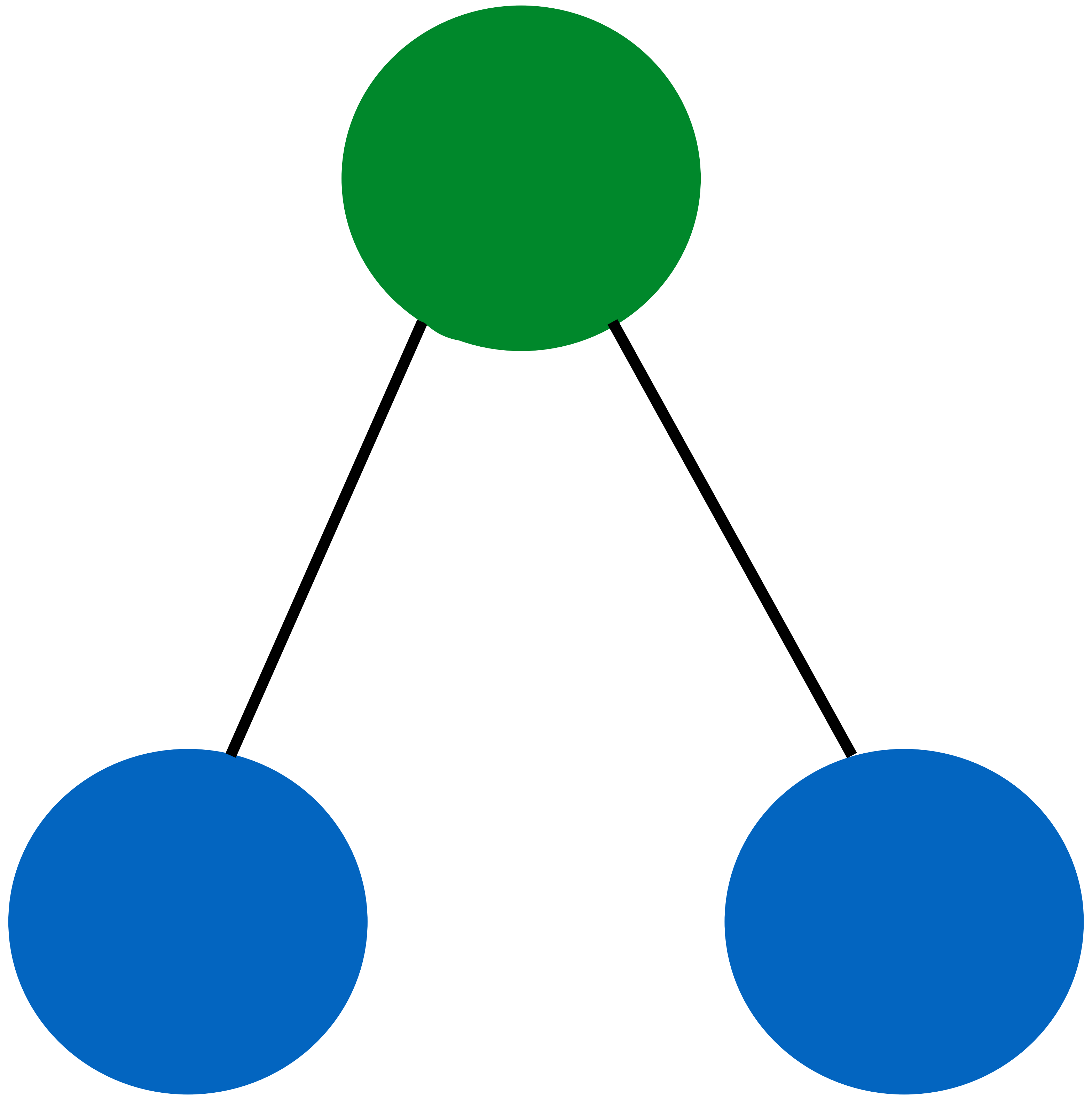
```
chooseTopping (topping) {
  setSelectedTopping(topping)
}
```



```
const [selectedTopping, setSelectedTopping] = useState('cheese');
```

```
chooseTopping (topping) {
  setSelectedTopping(topping)
}
```





```
setSelectedTopping(‘pepper’)
```

```
chooseTopping () {}
```

```
props.selectedTopping: ‘pepper’
props.chooseTopping: fn
```

CLASS COMPONENTS VS FUNCTIONAL COMPONENTS

CLASSES

(we won't cover class components,
but they're good to be familiar with)

- **Defined using the `class` keyword**
- **Must have a constructor with `this.state`**
- **Must have a render method**
- **May have additional methods**
- **Accesses props passed to it via `this` context (i.e. `this.props`)**

FUNCTIONS

- **Just a function**
- **Also has state, can update view based on state change**
- **The function's return value *is* the "render"**
- **Accesses props passed to it via the first argument to that function (i.e. `const Topping = (props) => {…}`)**

WHICH WOULD YOU PREFER?

FUNCTIONS!

- Functional components are simple. Classes can get complex.
- Functional components are easy to re-use and easy to test
- *“Simplicity is a prerequisite for reliability”*
- Rule of thumb: write lots of functional components, and not as many classes