State: How to Change Your App

WeStride

Alexa
@alexa99

Alexa was created by Amazon and heles you buy thigs.

Cortana
@cortana32

Cortana was made by Microsoft. Who knows what it does?

Siri
@siri01

Siri was made by Apple and is being phased out

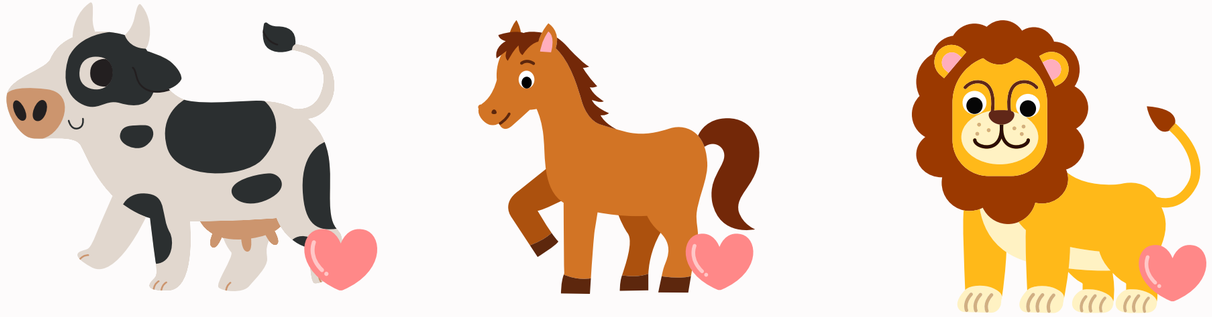
Add Animal

Add Animal

Show a tiny heart next to each animal

Each click should make the heart larger and larger

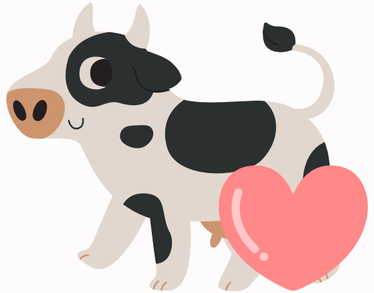
Add Animal



Show a tiny heart next to each animal

Each click should make the heart larger and larger

Add Animal



Show a tiny heart next to each animal

Each click should make the heart larger and larger

Add Animal

Show a tiny heart next to each animal

Each click should make the heart larger and larger

Add Animal

Show a tiny heart next to each animal

Each click should make the heart larger and larger

WeStride

Alexa
@alexa99

Alexa was created by Amazon and heles you buy thigs.

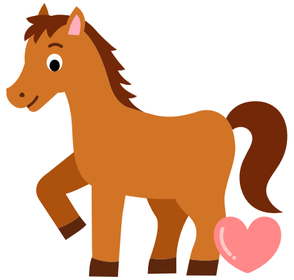
Cortana
@cortana32

Cortana was made by Microsoft. Who knows what it does?

Siri
@siri01

Siri was made by Apple and is being phased out

Add Animal



index.js

App.js

shows ‘Add Animal’ button + some number of AnimalShow

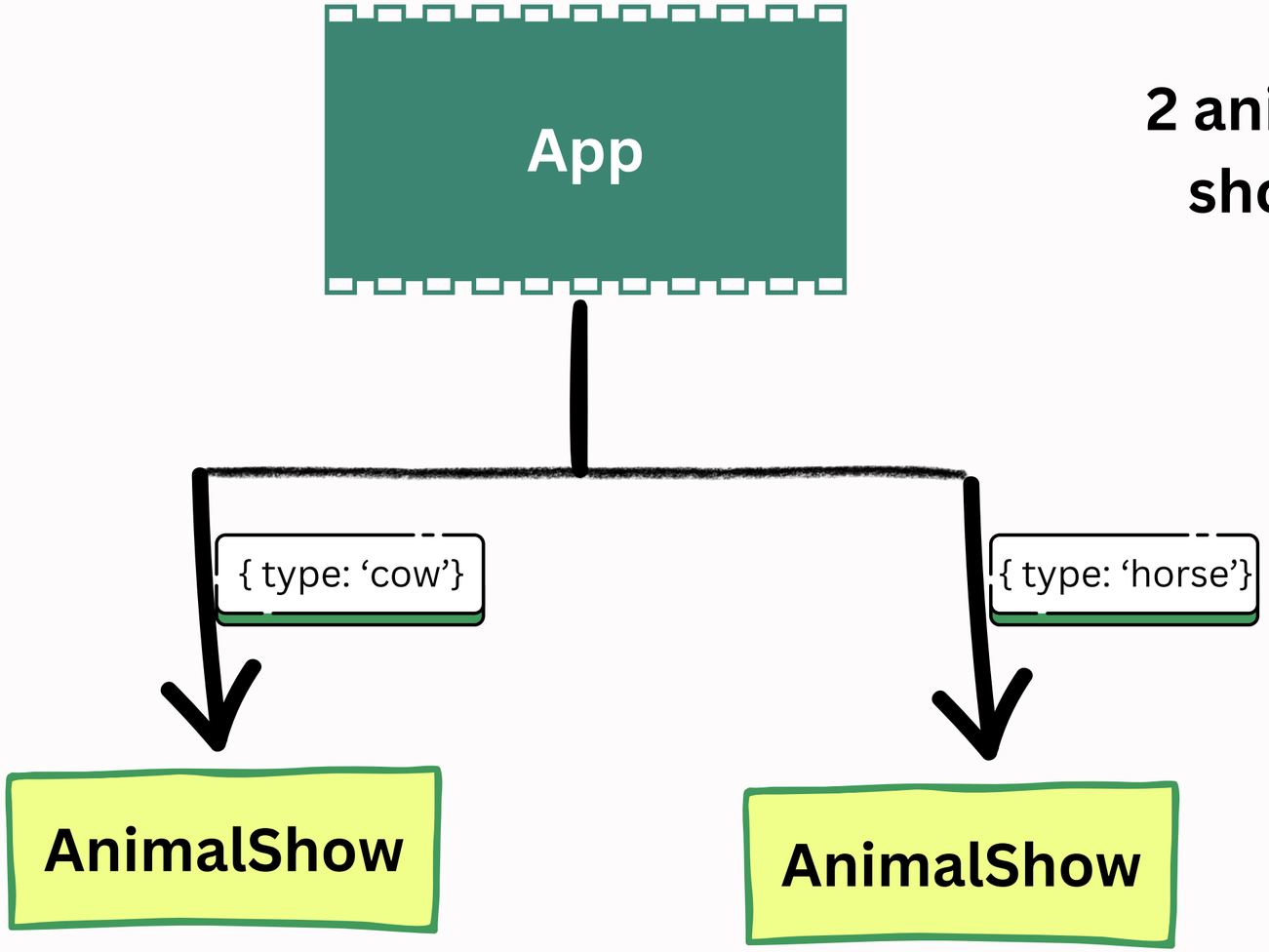
AniamlShow.js

Shows a single animal + heart

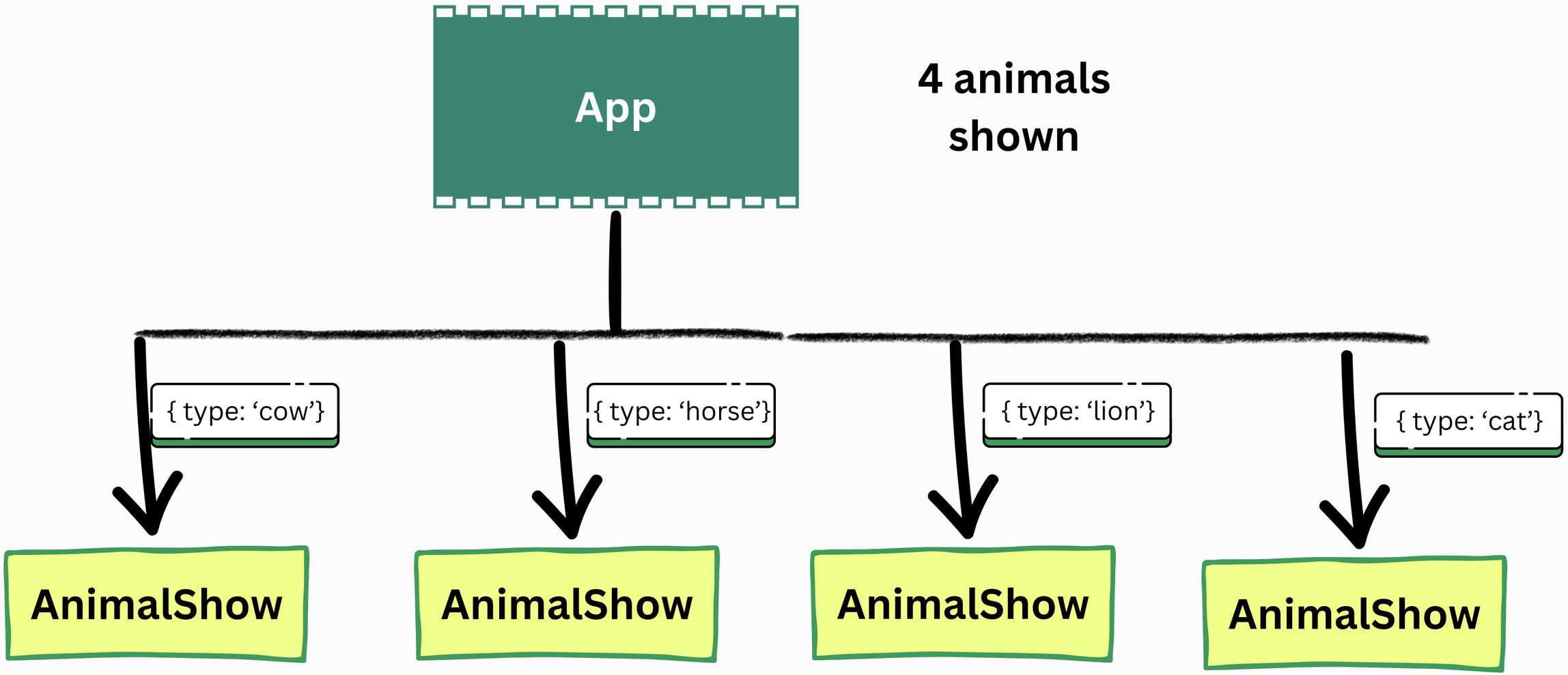
WeStride

App

0 animals shown



2 animals shown


App
4 animals shown
{ type: ‘cow’}
{ type: ‘horse’}
{ type: ‘lion’}
{ type: ‘cat’}
AnimalShow
AnimalShow
AnimalShow
AnimalShow

Add Animal

Add one picture of
an animal for each click

Add Animal

Show a tiny heart next to each animal

Each click should make the heart larger and larger

WeStride

Short
Term
Goal

Add Animal

Animals to slow: 0

WeStride

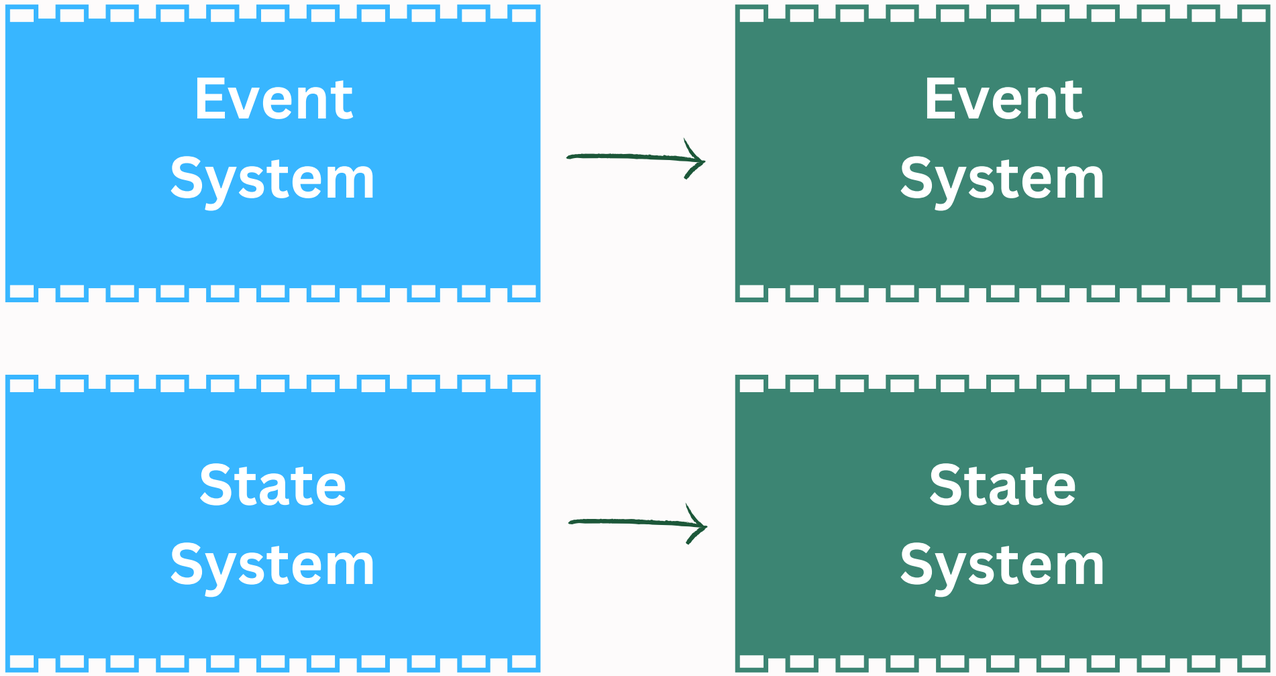
Detect a user clicking the button → Event System

Update content on screen → State System

Add Animal

Animals to slow: 0

WeStride



Add Animal

Animals to slow: 0

Using Events

01 Decide what kind of event you want to watch

02 Create a function Usually called an event handler or callback function

03 Name the function using pattern of handle + Event Name
(Not a requirement! Community convention)

04 Pass the function as a prop to a plain element

05 Make sure you pass the function using a valid event name (‘onClick’, ‘onMouseOver’, etc)

06 Make sure you pass a reference to the function (don’t call it!!)

```
function App() {
    const handleClick = () => {
        console.log('Button was Clicked!!!')
    };

    return (
        <div>
            <button onClick={handleClick}>
                Add Animal
            </button>
        </div>
    )
}
```

reactjs.org/docs/events.html

Listing of event names

This link might break soon (docs being updated)

Event Name	Called When...
onClick	A user clicks on some thing
onChange	A user types in a text input

Using Events

01 Decide what kind of event you want to watch

02 Create a function Usually called an event handler or callback function

03 Name the function using pattern of handle + Event Name
(Not a requirement! Community convention)

04 Pass the function as a prop to a plain element

05 Make sure you pass the function using a valid event name (‘onClick’, ‘onMouseOver’, etc)

06 Make sure you pass a reference to the function (don’t call it!!)

```
function App() {
    const handleClick = () => {
        console.log('Button was Clicked!!!')
    };

    return (
        <div>
            <button onClick={handleClick}>
                Add Animal
            </button>
        </div>
    )
}
```

WeStride

```
const doGreeting = () => {
    return 'hi there'
};

console.log( doGreeting )

console.log( doGreeting() )
```

What’s the difference between these two lines?

WeStride

```
const doGreeting = () => {
    return 'hi there'
};

console.log( doGreeting )

console.log( doGreeting() )
```

Logs the ‘doGreeting’ function.

Dose not call it.

Immediately calls

‘doGreeting’ function and logs the return value

WeStride

```
function App() {
    const handleClick = () => {
        console.log('Button was Clicked!!!')
    };

    return (
        <div>
            <button onClick={handleClick}>
                Add Animal
            </button>
        </div>
    )
}
```

Good.

Gives the button the whole function so the button can call the function in the future when the button is clicked

WeStride

```
function App() {
    const handleClick = () => {
        console.log('Button was Clicked!!!')
    };

    return (
        <div>
            <button onClick={handleClick}>
                Add Animal
            </button>
        </div>
    )
}
```

Probably bad.

Immediately calls ‘onButtonClick’ and gives the button the return value

EXERCISE WITH EVENTS

Goal: Add a click event handler to the button element.

Make sure that whennever the button is clicked, you console.log a message of ‘hello there!’ or someting similar.

WeStride

Detect a user clicking the button → Event System

Update content on screen → State System

Add Animal

Animals to slow: 0

count = 0
user clicks

count = 1
user clicks

count = 2
user clicks

count = 3
user clicks

Add Animal

Animals to slow: 0

count = 0
user clicks

count = 1
user clicks

count = 2
user clicks

count = 3
user clicks

Add Animal

Animals to slow: 0

- Need some kind of ‘count’ variable
- Should start with a value of 0
- Add 1 whenever the use clicks
- Whenever the value is changed, update content on screen (rerender)

First time component is rendered

```
function App() {

    const [count, setCount] = useState(0)

    const handleClick = () => {
        setCount(count + 1);
    }


    return (
        <div>
            <button onClick={handleClick}>Add Animal</button>
            <div>Number of animals to show: {count}</div>
        </div>
    )
}
```

count

default

First time component is rendered

```
function App() {

    const [count, setCount] = useState(0)

    const handleClick = () => {
        setCount(count + 1);
    }


    return (
        <div>
            <button onClick={handleClick}>Add Animal</button>
            <div>Number of animals to show: {count}</div>
        </div>
    )
}
```

count

default

First time component is rendered

```
function App() {

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

    const handleClick = () => {
        setCount( ------- count + 1);
    }


    return (
        <div>
            <button onClick={handleClick}>Add Animal</button>
            <div>Number of animals to show: {------- count}</div>
        </div>
    )
}
```

First time component is rendered

```
function App() {

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

    const handleClick = () => {
        setCount(count + 1);
    }


    return (
        <div>
            <button onClick={handleClick}>Add Animal</button>
            <div>Number of animals to show: {count}</div>
        </div>
    )
}
```

count

1

count

1

count

1

First time component is rendered

```
function App() {

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

    const handleClick = () => {
        setCount(count + 1);
    }


    return (
        <div>
            <button onClick={handleClick}>Add Animal</button>
            <div>Number of animals to show: {count}</div>
        </div>
    )
}
```

First time component is rendered

```
function App() {

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

    const handleClick = () => {
        setCount(count + 1);
    }


    return (
        <div>
            <button onClick={handleClick}>Add Animal</button>
            <div>Number of animals to show: {count}</div>
        </div>
    )
}
```

count

2

count

2

count

2

Only Things to Understand Right row

Use the state system when you want to update content on the screen

Calliing ‘useState’ deines a new piece of state

The first argument to ‘useState’ is used as the initial value

Updating state is done only using the setter function

Calling the setter function causes React to rerender the component

Don’t Worray About These Yet

What’s the weird syntax on the ‘useState’ line?

How many piece of state should i make ?

What should i call my state variable?

What type of data should my state be?

Does state work with Redux?

Does state work with Redux?

WeStride

Array destrucuring. Fancy way to get access to the piece of state + setter

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

The piece of state! Starts as 0, changes over time

Setter function. Used to update our piece of state

Default value for this piece of state

Add Animal

Add one picture of an animal for each click

Animal displayed should be **pick at random**

WeStride

App Component

‘animals’ state
[ ]

0 animals shown

WeStride

App Component

‘animals’ state
[‘cow’, ‘bird’]

2 animals shown

{ type: ‘cow’ }

{ type: ‘bird’ }

AnimalShow

AnimalShow

Only Things to Understand Right row

Use the state system when you want to update content on the screen

Calliing ‘useState’ deines a new piece of state

The first argument to ‘useState’ is used as the initial value

Updating state is done only using the setter function

Calling the setter function causes React to rerender the component

Don’t Worray About These Yet

What’s the weird syntax on the ‘useState’ line?

How many piece of state should i make ?

What should i call my state variable?

What type of data should my state be?

Does state work with Redux?

Does state work with Redux?

Add Animal

Show a tiny heart next to each animal

Each click should make the heart larger and larger

WeStride

App Component

‘count’ state
[‘cow’, ‘bird’]

0 animals shown

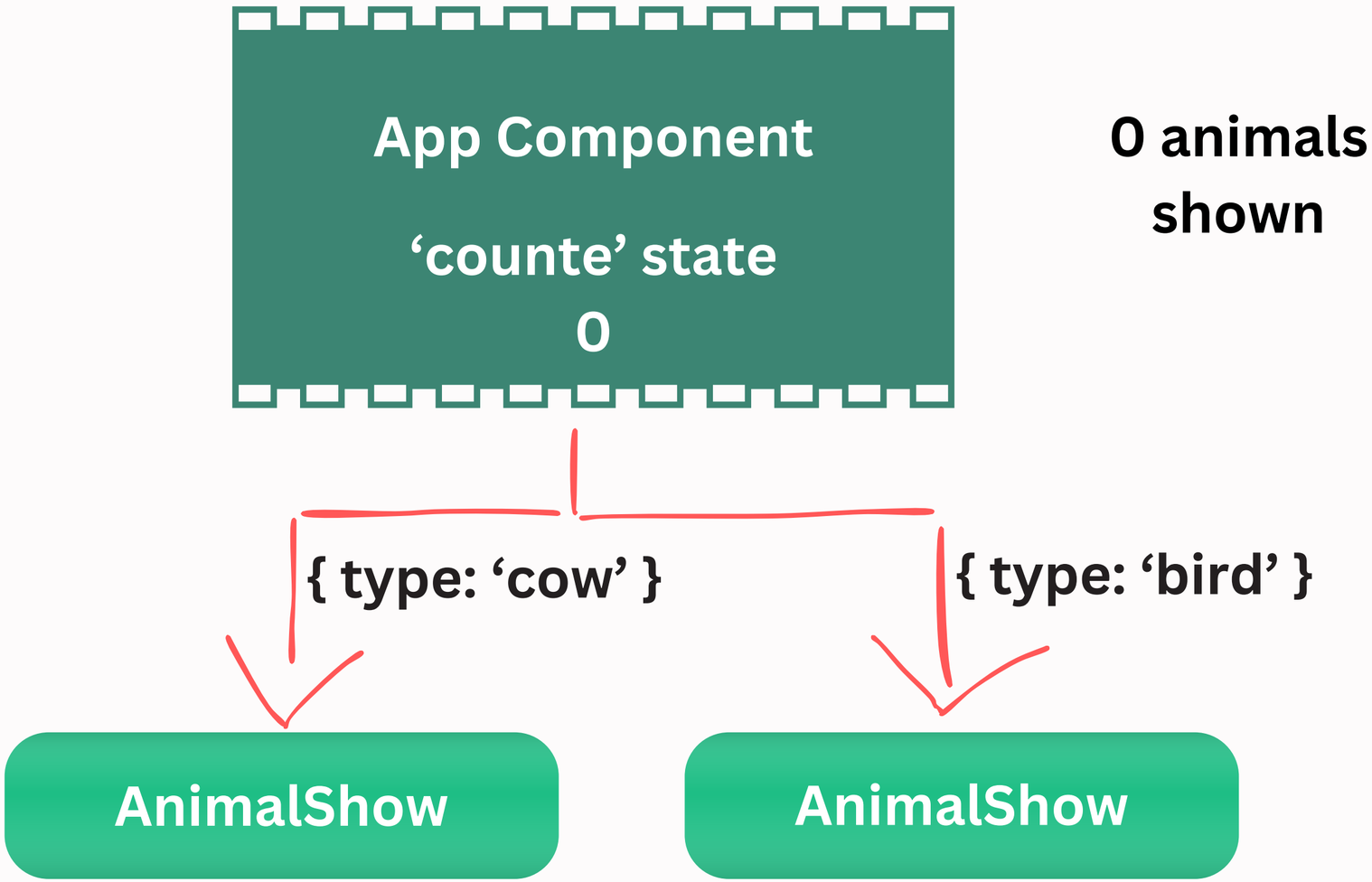
{ type: ‘cow’ }

{ type: ‘bird’ }

AnimalShow

AnimalShow

WeStride


App Component
‘counte’ state
0
0 animals shown
{ type: ‘cow’ }
{ type: ‘bird’ }
AnimalShow
AnimalShow

WeStride

App Component

‘animals’ state
[ ]

0 animals shown

WeStride

App Component

‘count’ state
[‘cow’, ‘bird’]

0 animals shown

{ type: ‘cow’ }

{ type: ‘bird’ }

AnimalShow

AnimalShow

WeStride

App Component

‘animals’ state
[‘cow’, ‘bird’]

0 animals shown

{ type: ‘cow’ }

{ type: ‘bird’ }

AnimalShow

AnimalShow

WeStride

App Component

‘animals’ state
[ ]

0 animals shown

{ type: ‘cow’ }

{ type: ‘bird’ }

AnimalShow

AnimalShow

WeStride

App Component

‘animals’ state
[‘cow’, ‘bird’]

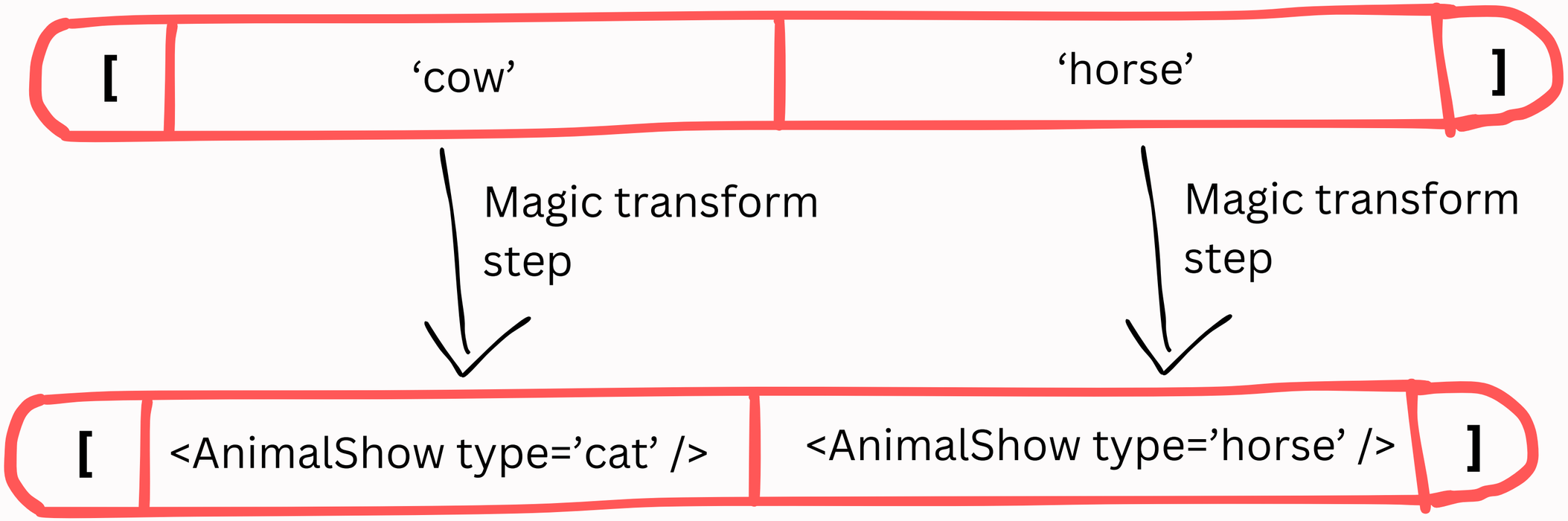
2 animals shown

{ type: ‘cow’ }

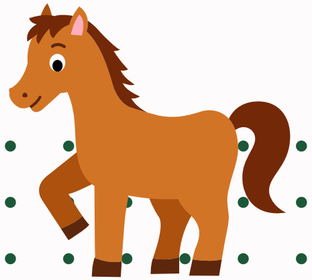
{ type: ‘bird’ }

AnimalShow

AnimalShow



We want to transform each element in the array into acomponent

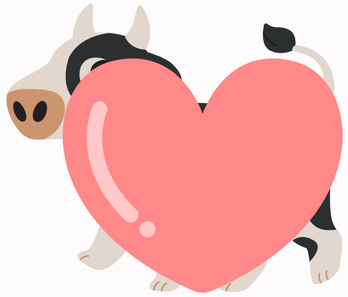


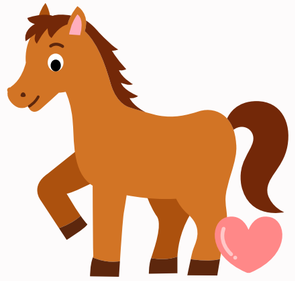
[‘cow’ | ‘horse’]

```
(anaimal, i) => {
    return <AnimalShow type={animal} key={i} />
};
```

```
(anaimal, i) => {
    return <AnimalShow type={animal} key={i} />
};
```

[<AnimalShow type=’cat’ /> | <AnimalShow type=’horse’ />]





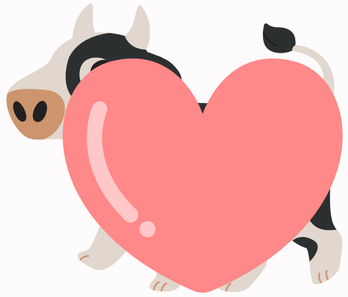
[| | ‘horse’ |]

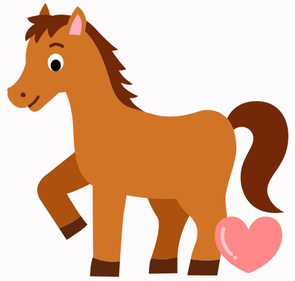
‘cow’

```
(anaimal, i) => {
    return <AnimalShow type={animal} key={i} />
};
```

```
(anaimal, i) => {
    return <AnimalShow type={animal} key={i} />
};
```

[| <AnimalShow type=’cat’ /> | <AnimalShow type=’horse’ /> |]





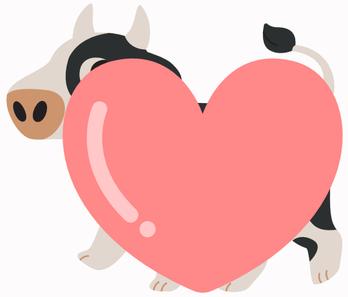
[‘cow’]

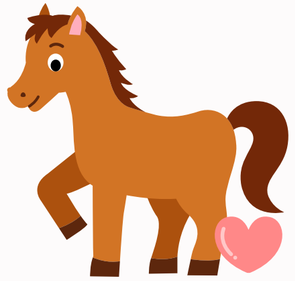
‘horse’

```
(anaimal, i) => {
    return <AnimalShow type={animal} key={i} />
};
```

```
(anaimal, i) => {
    return <AnimalShow type={animal} key={i} />
};
```

[<AnimalShow type=’cat’ /> <AnimalShow type=’horse’ />]





WeStride

App Component

‘animals’ state
[‘cow’, ‘bird’]

0 animals shown

{ type: ‘cow’ }

{ type: ‘bird’ }

AnimalShow

AnimalShow

Images for the app

Attached to this lecture is a file called svg.zip Make sure you download this file. We will use it in the next lecture.

Resources for this lecture

svg.zip

Add Animal

Show a tiny heart next to each animal

Each click should make the heart larger and larger

Add Animal

Add one picture of an animal for each click

Animal displayed should be **pick at random**

Add Animal

Show a tiny heart next to each animal

Each click should make the heart larger and larger

Project

App.js

↓

App.css

Contains CSS to style the App component

AniamlShow.js

↓

AniamlShow.css

Contains CSS to style AnimalShow component

Using Events

01 Decide what kind of event you want to watch

02 Create a function Usually called an event handler or callback function

03 Name the function using pattern of handle + Event Name
(Not a requirement! Community convention)

04 Pass the function as a prop to a plain element

05 Make sure you pass the function using a valid event name (‘onClick’, ‘onMouseOver’, etc)

06 Make sure you pass a reference to the function (don’t call it!!)

```
function App() {
    const handleClick = () => {
        console.log('Button was Clicked!!!')
    };

    return (
        <div>
            <button onClick={handleClick}>
                Add Animal
            </button>
        </div>
    )
}
```

WeStride

```
function App() {
    const handleClick = () => {
        console.log('Button was Clicked!!!')
    };

    return (
        <div>
            <button onClick={handleClick}>
                Add Animal
            </button>
        </div>
    )
}
```

Good.

Gives the button the whole function so the button can call the function in the future when the button is clicked

WeStride

01

```
function App() {
    return (
        <div>
            <button onClick={() => console.log('Button clicked!')}>
                Add Animal
            </button>
        </div>
    )
}
```

Inline Handler

Arrow function used if we only need to do one thing when the event is triggered

Using Events

01 Decide what kind of event you want to watch

02 Create a function Usually called an event handler or callback function

03 Name the function using pattern of handle + Event Name
(Not a requirement! Community convention)

04 Pass the function as a prop to a plain element

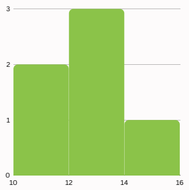
05 Make sure you pass the function using a valid event name (‘onClick’, ‘onMouseOver’, etc)

06 Make sure you pass a reference to the function (don’t call it!!)

```
function App() {
    const handleClick = () => {
        console.log('Button was Clicked!!!')
    };

    return (
        <div>
            <button onClick={handleClick}>
                Add Animal
            </button>
        </div>
    )
}
```

STATE



Data

Data that changes as the use interacts with our app

Changes

When this data changes, React will update content on the screen automatically

One-and-only

This is the one-and-only way we can change what content React shows.

Even other libs that update content use the state system behind the scenes

First time component is rendered

```
function App() {

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

    const handleClick = () => {
        setCount(count + 1);
    }


    return (
        <div>
            <button onClick={handleClick}>Add Animal</button>
            <div>Number of animals to show: {count}</div>
        </div>
    )
}
```

0
count

0
count

0
count

First time component is rendered

```
function App() {

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

    const handleClick = () => {
        setCount(count + 1);
    }


    return (
        <div>
            <button onClick={handleClick}>Add Animal</button>
            <div>Number of animals to show: {count}</div>
        </div>
    )
}
```

count

1

count

1

count

1

First time component is rendered

```
function App() {

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

    const handleClick = () => {
        setCount(count + 1);
    }


    return (
        <div>
            <button onClick={handleClick}>Add Animal</button>
            <div>Number of animals to show: {count}</div>
        </div>
    )
}
```

count

2

count

2

count

2

Only Things to Understand Right row

Use the state system when you want to update content on the screen

Calliing ‘useState’ deines a new piece of state

The first argument to ‘useState’ is used as the initial value

Updating state is done only using the setter function

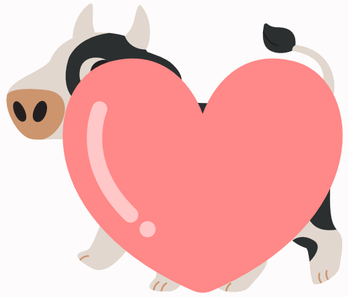
Calling the setter function causes React to rerender the component

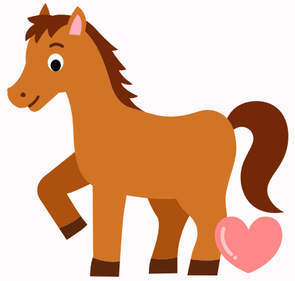
[‘cow’ | ‘horse’]

```
(anaimal, i) => {
    return <AnimalShow type={animal} key={i} />
};
```

```
(anaimal, i) => {
    return <AnimalShow type={animal} key={i} />
};
```

[<AnimalShow type=’cat’ /> | <AnimalShow type=’horse’ />]





This app was truly challenging

Lost? Confused?
That’s OK.