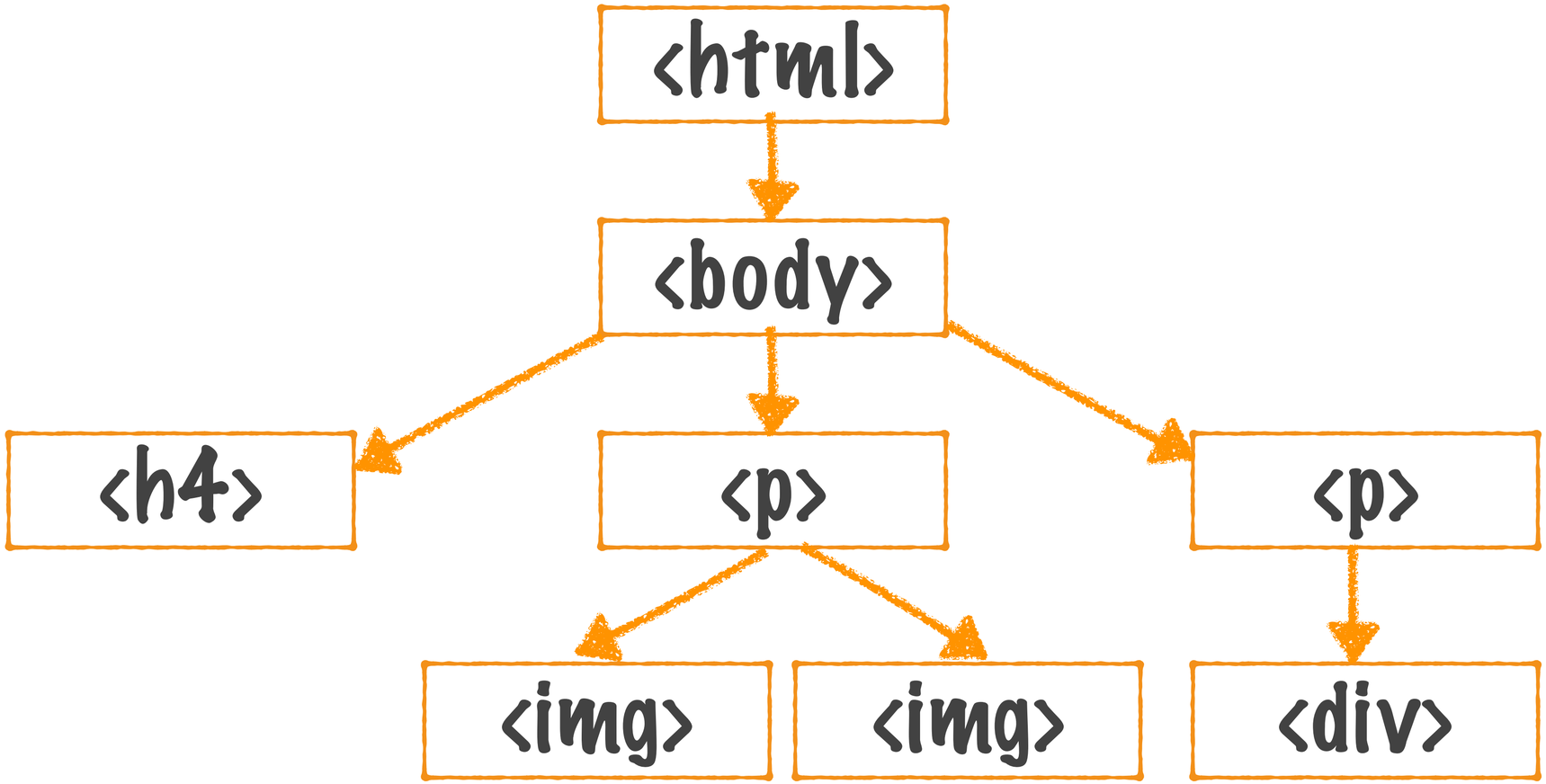
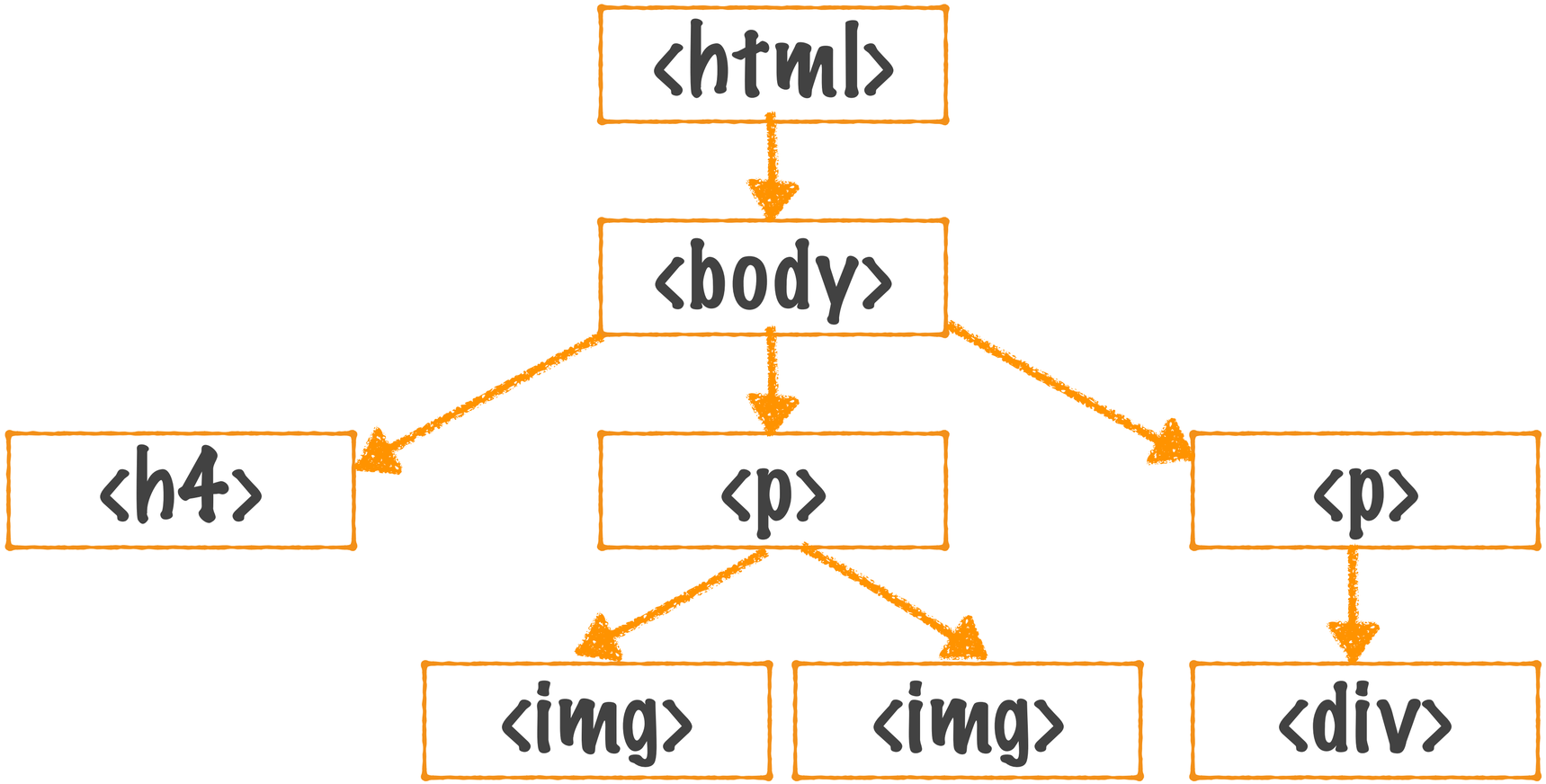
REACT JS

Capture and bubble phase



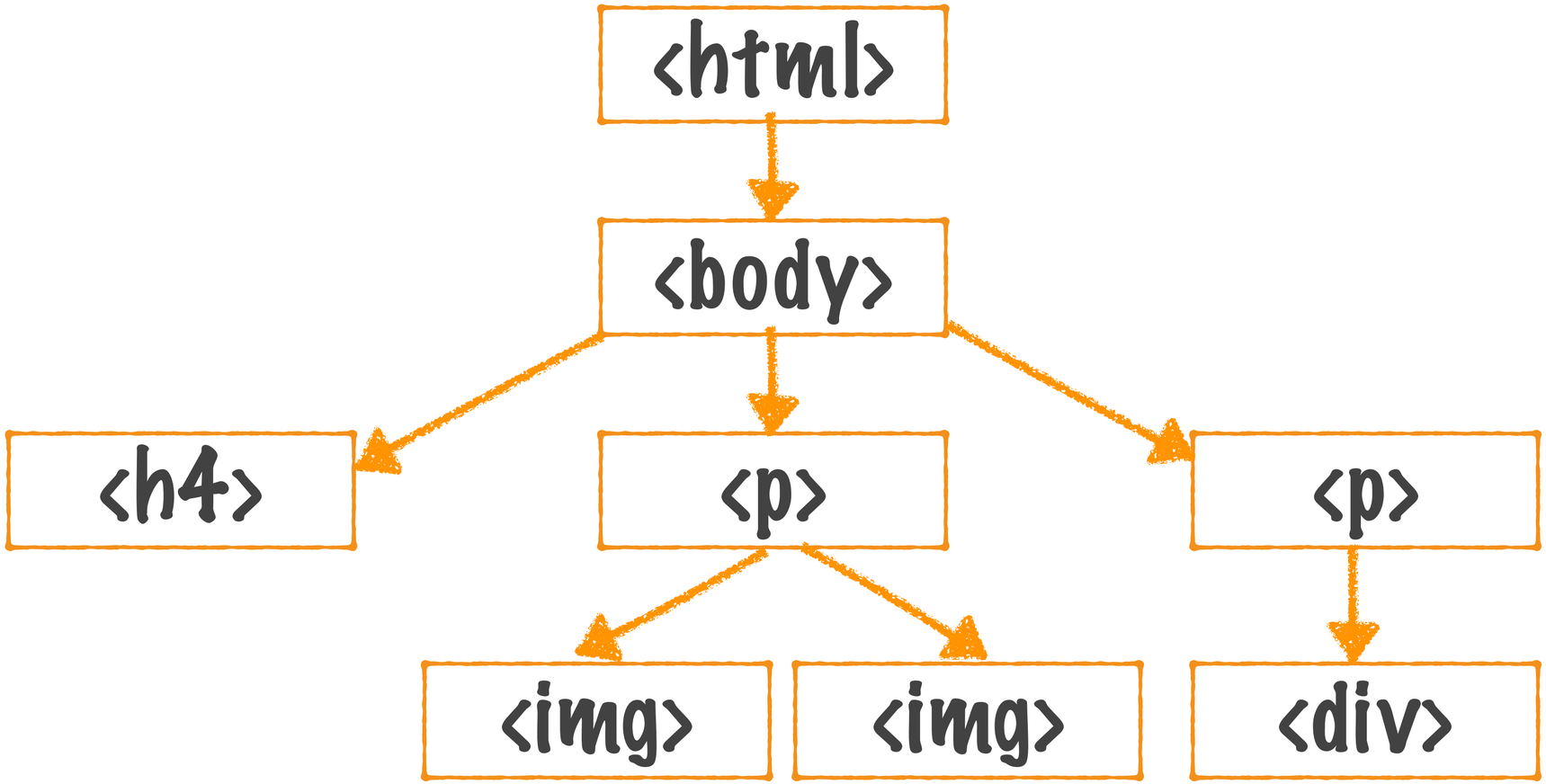
A DOM hierarchy

Capture and bubble phase



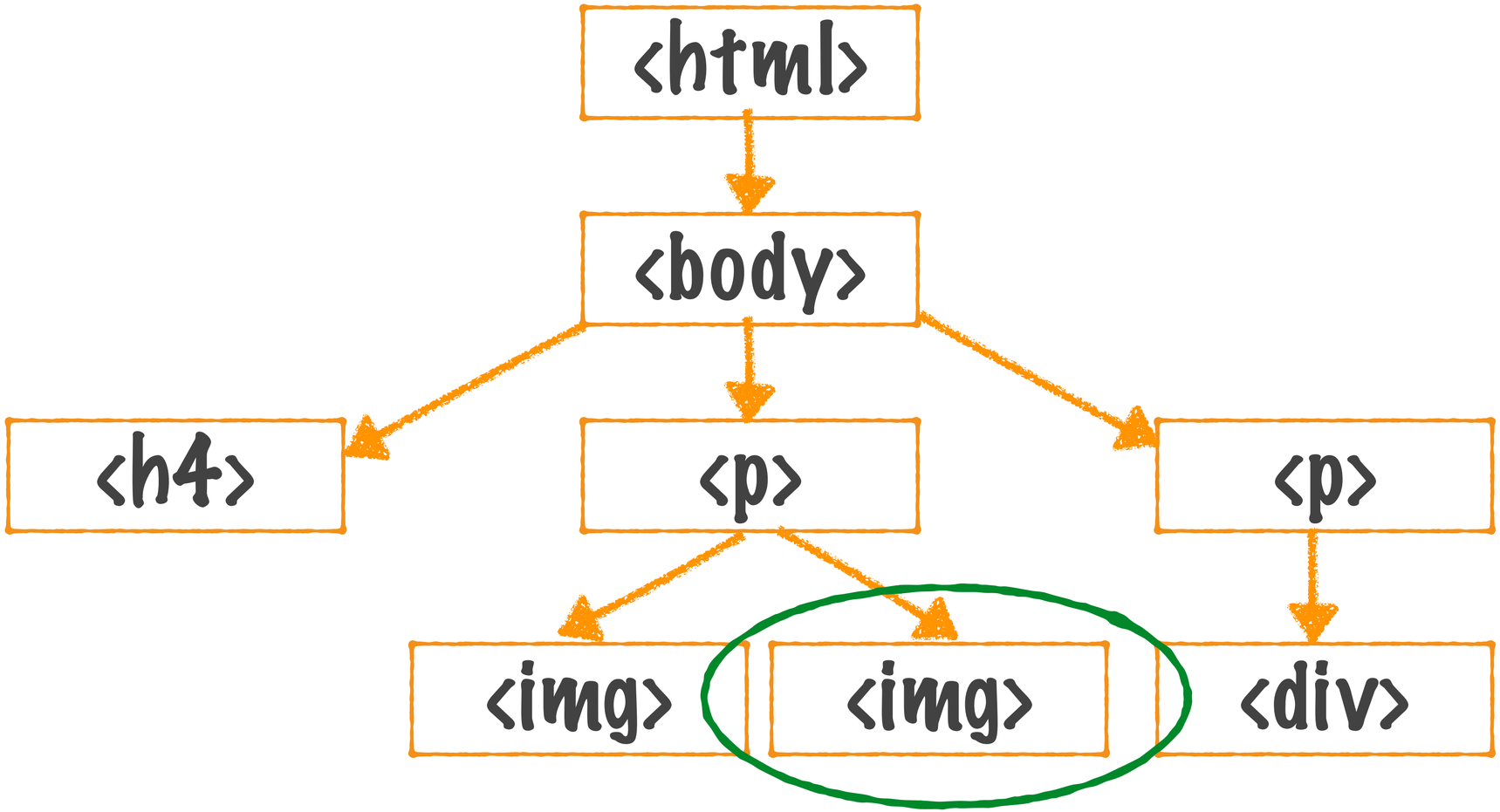
You can hook up event handlers to elements at different levels in the hierarchy

Capture and bubble phase



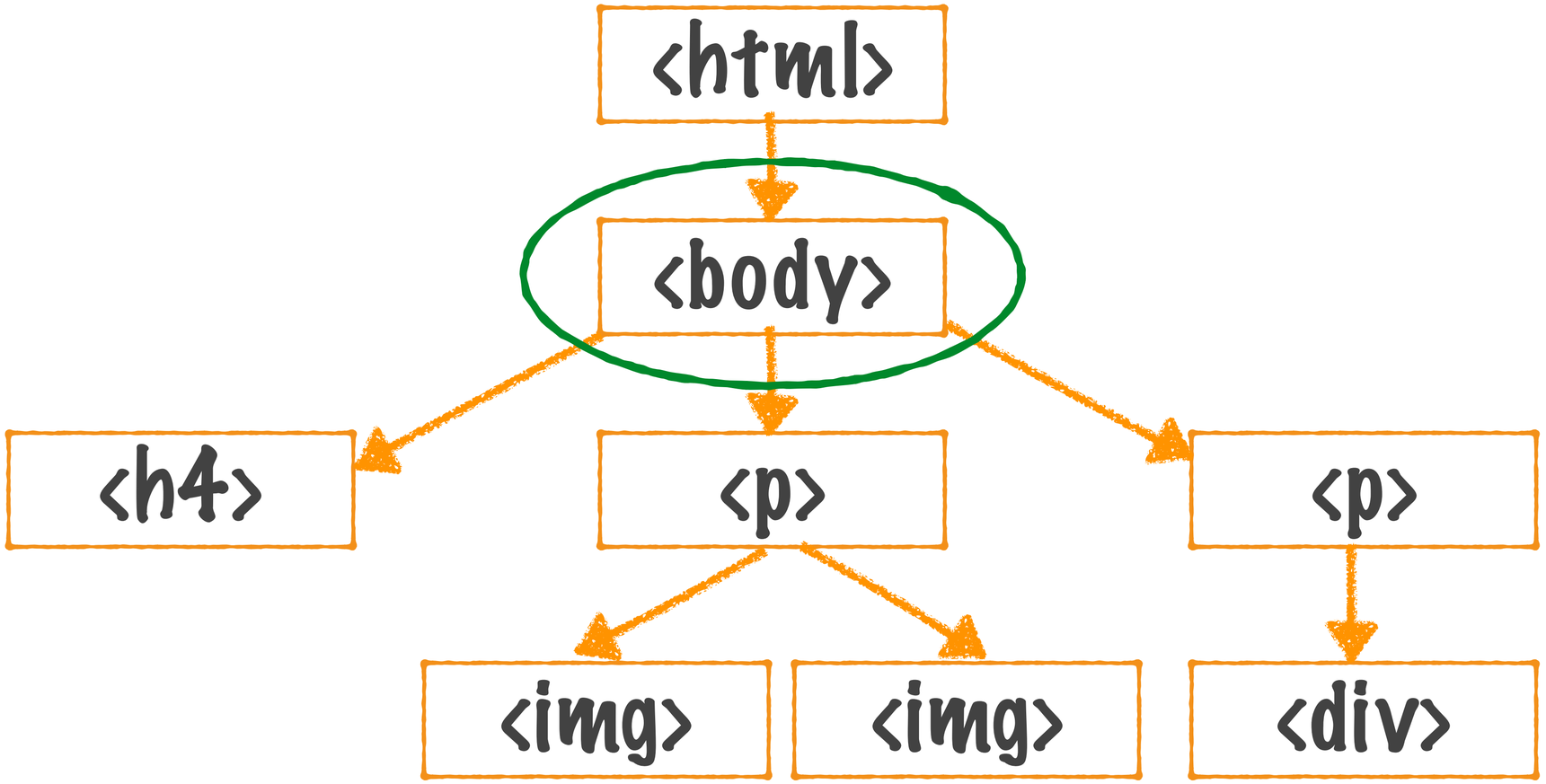
And events can be fired by any element

Capture and bubble phase



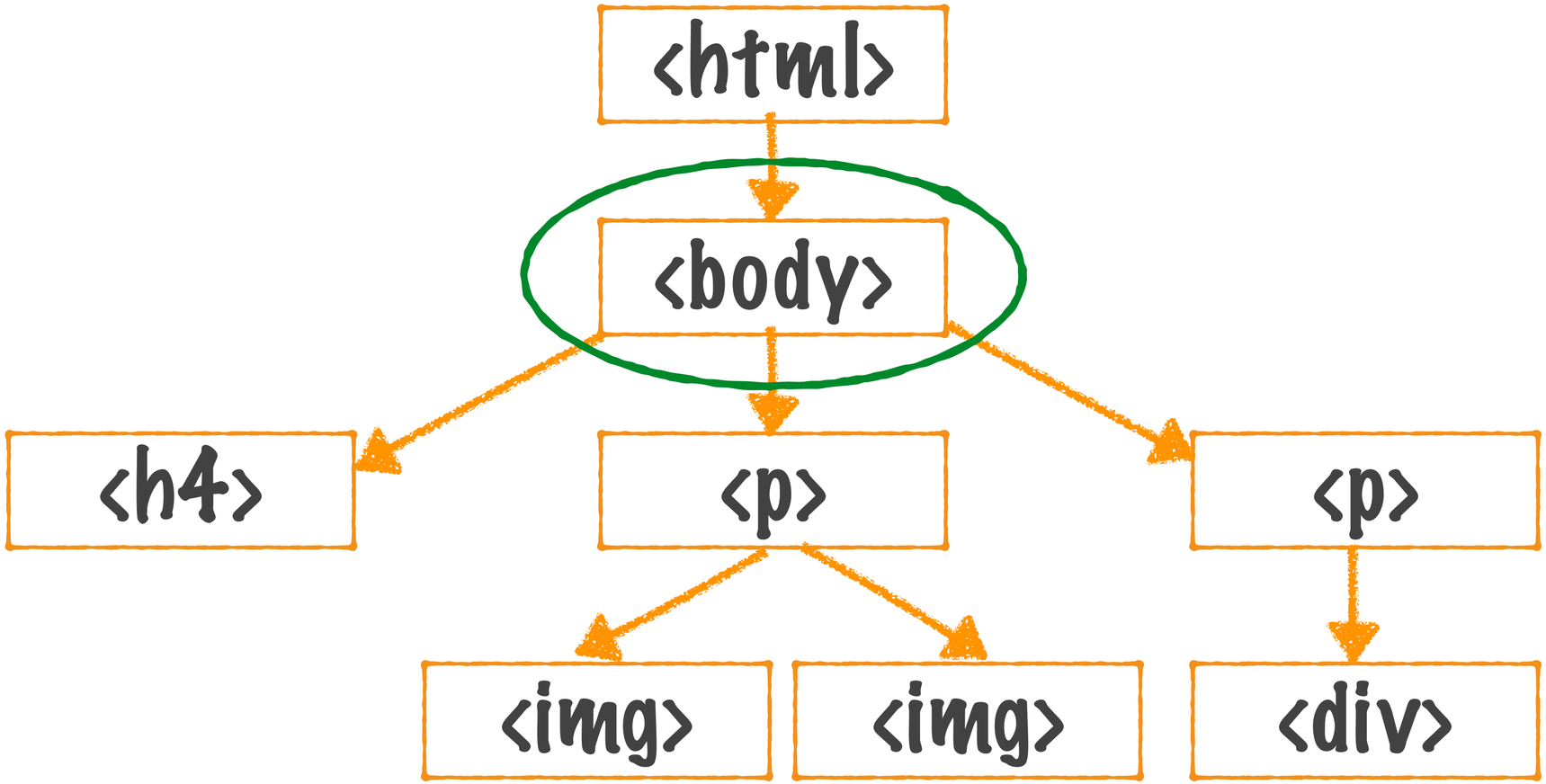
By the leaf elements

Capture and bubble phase



Or an interior element which has children as well as ancestors

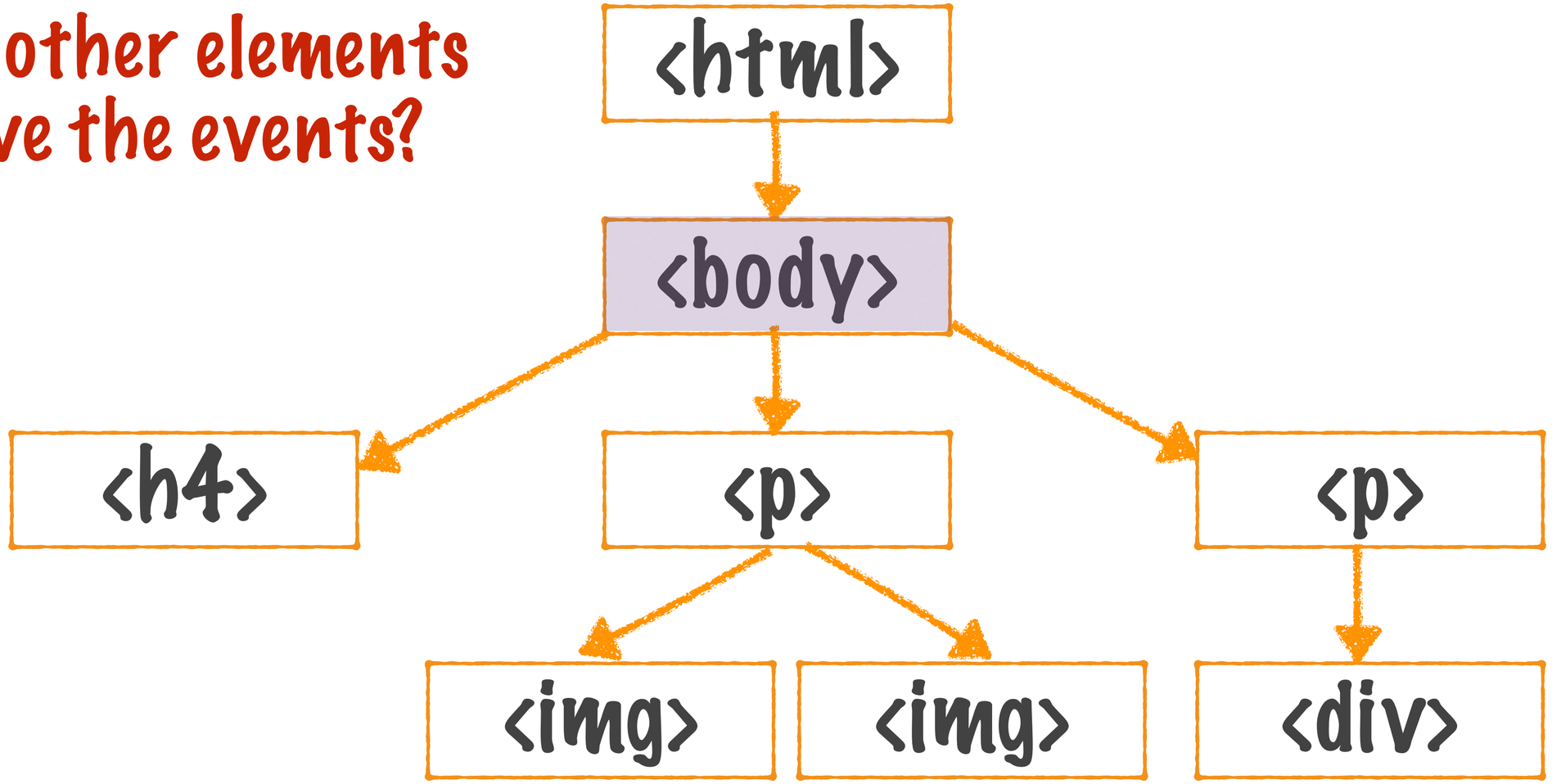
Capture and bubble phase



Which other elements receive the events?

Capture and bubble phase

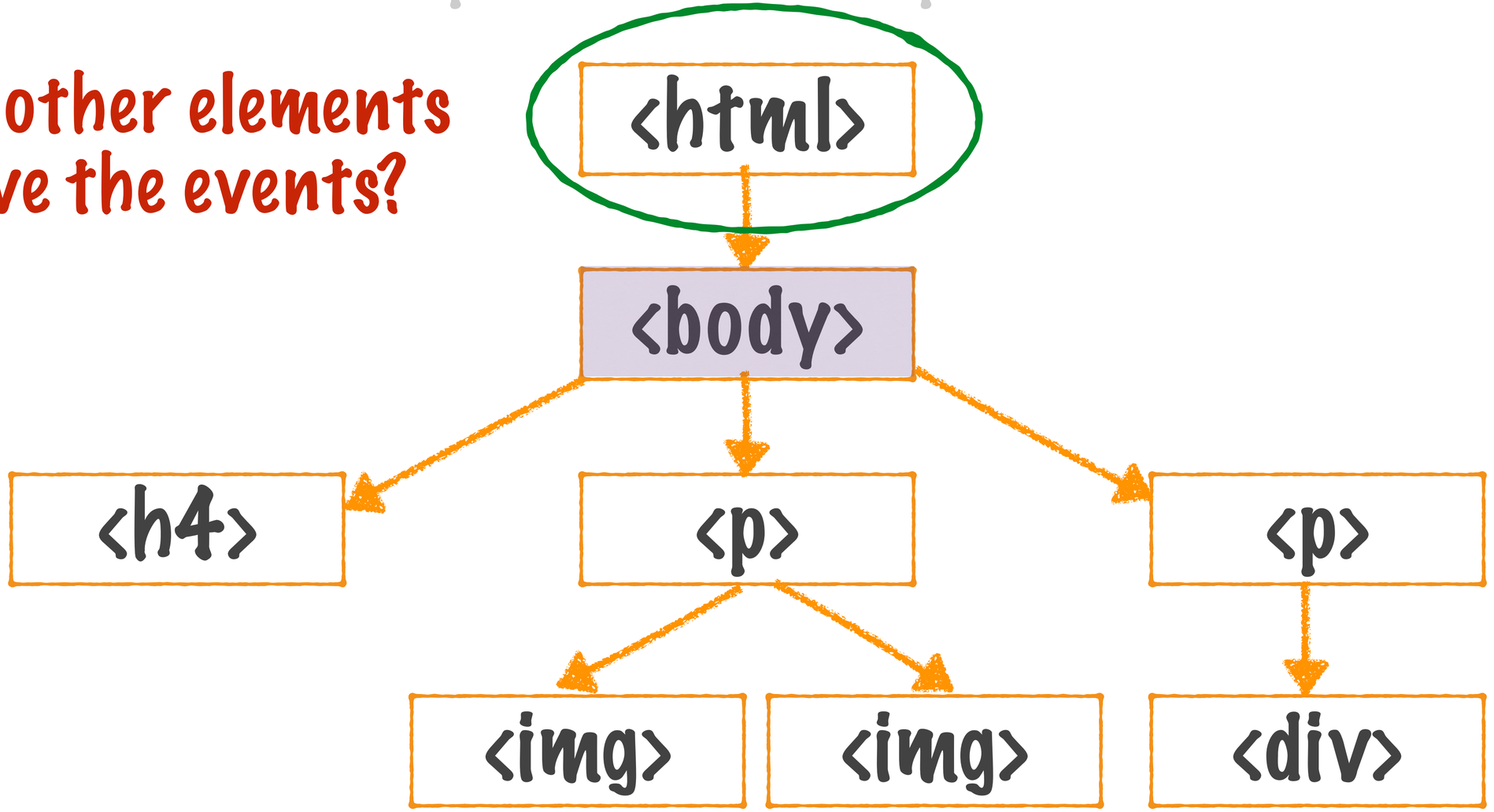
Which other elements receive the events?



The <body> element is where the event is triggered

Capture and bubble phase

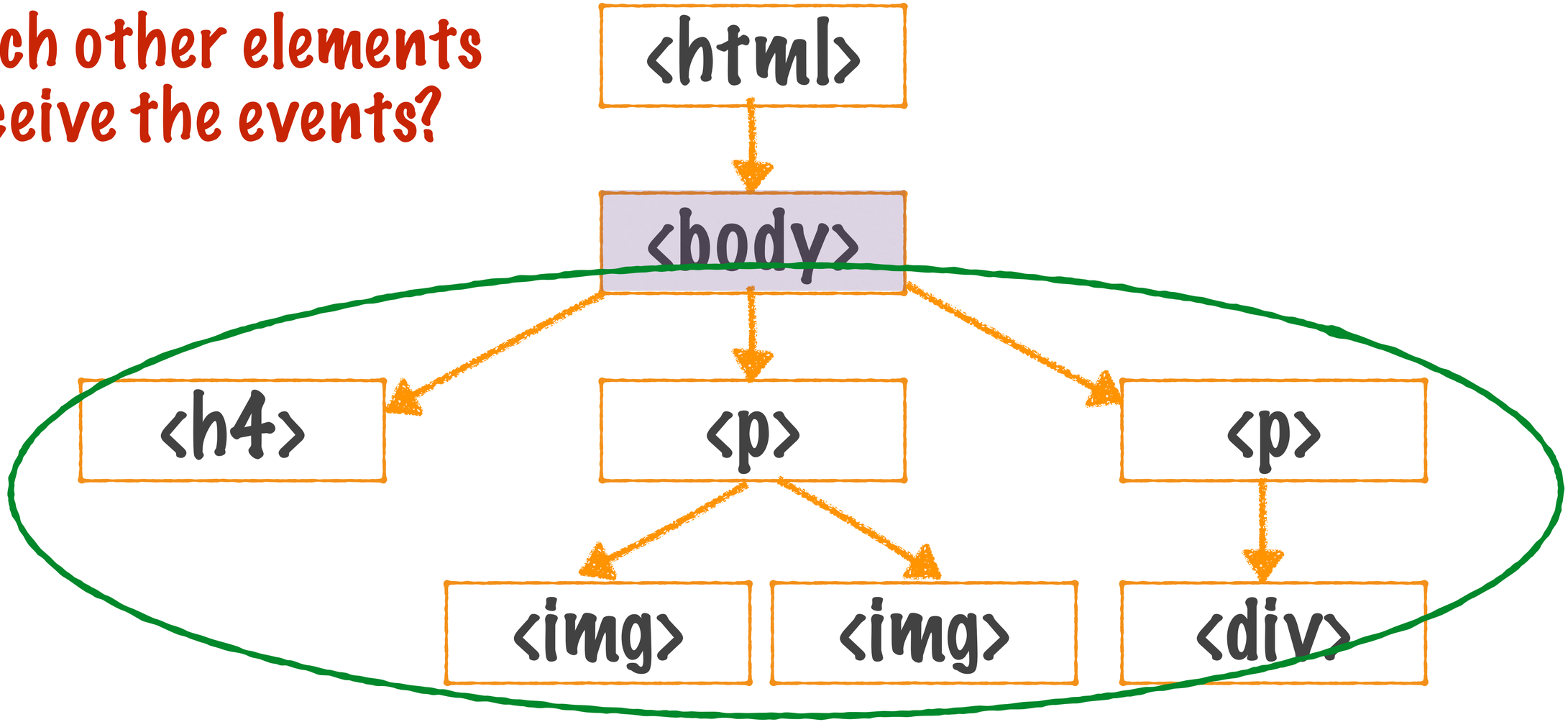
Which other elements receive the events?



You could argue that the <html> encompasses the <body> and so should receive the event as well

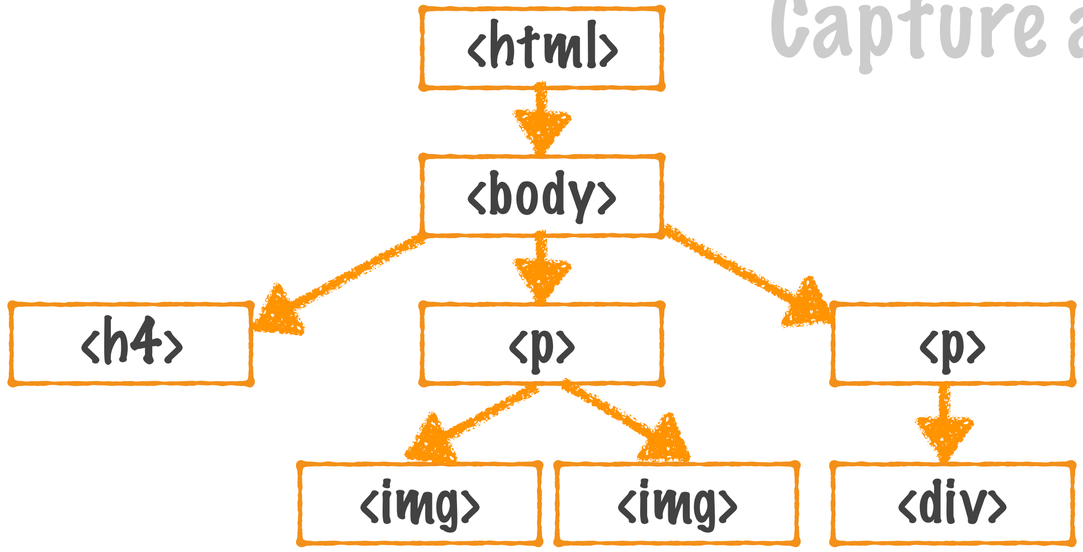
Capture and bubble phase

Which other elements receive the events?



Or you could argue that the <body> encompasses its child elements and they should receive the event

Capture and bubble phase



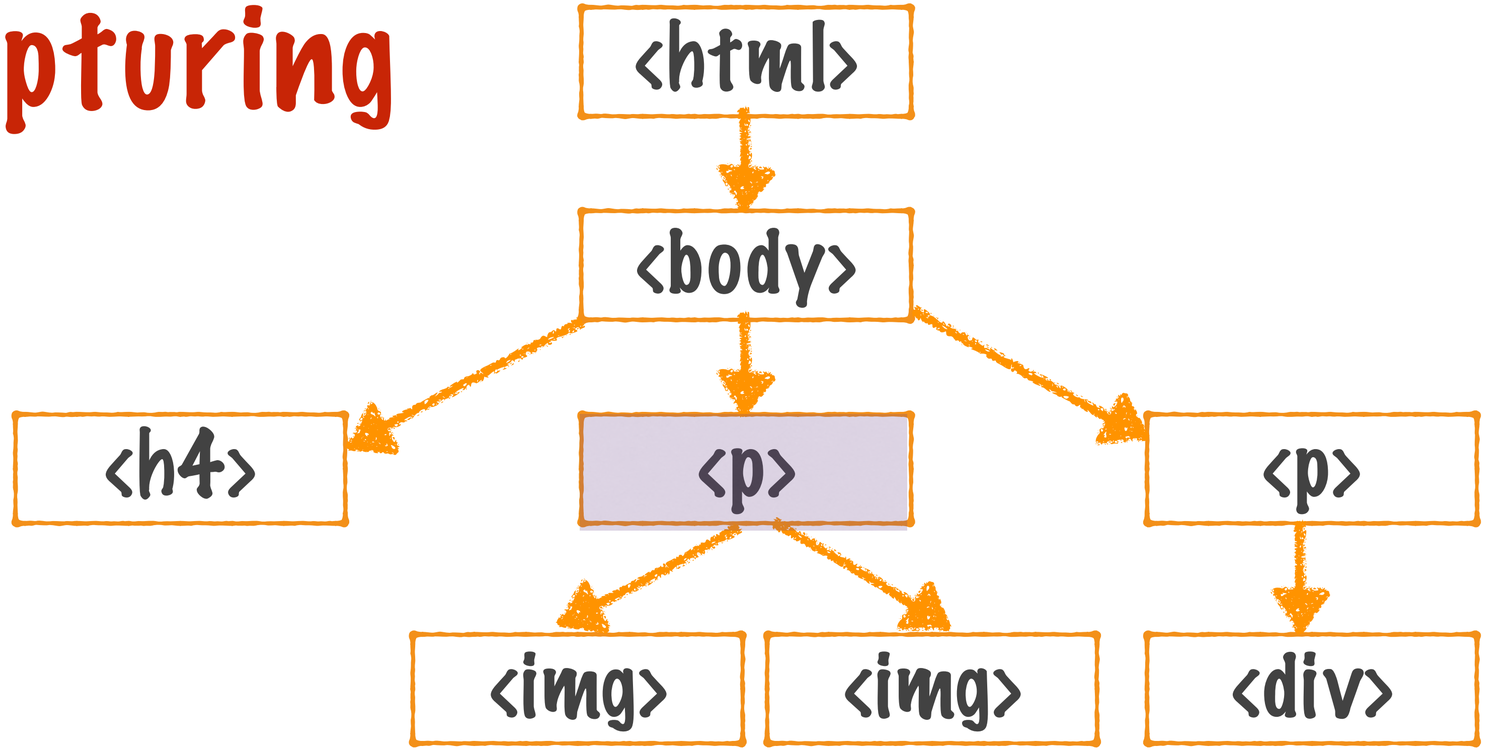
Which other elements receive the events?

These 2 ways in which events can travel are called:

Event Capturing

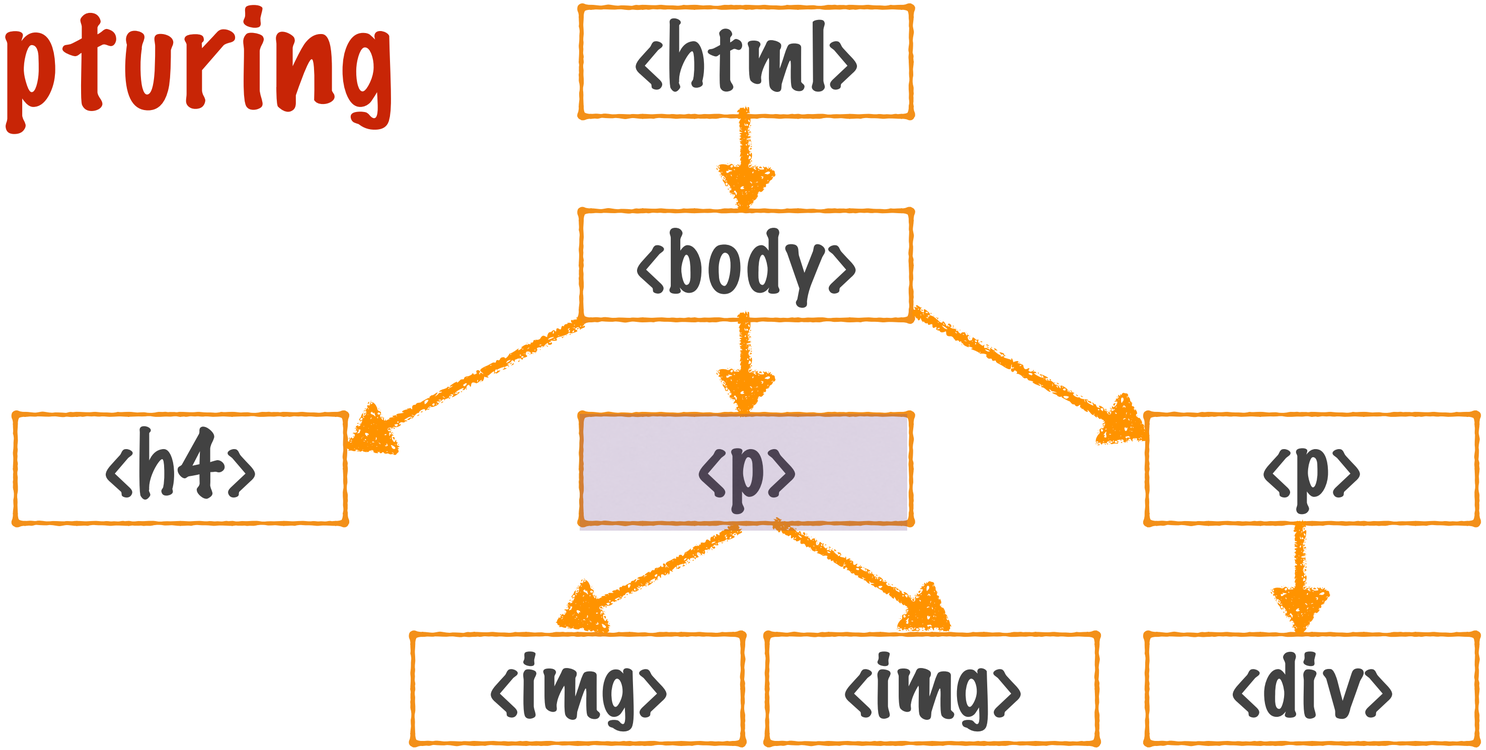
Event Bubbling

Event Capturing



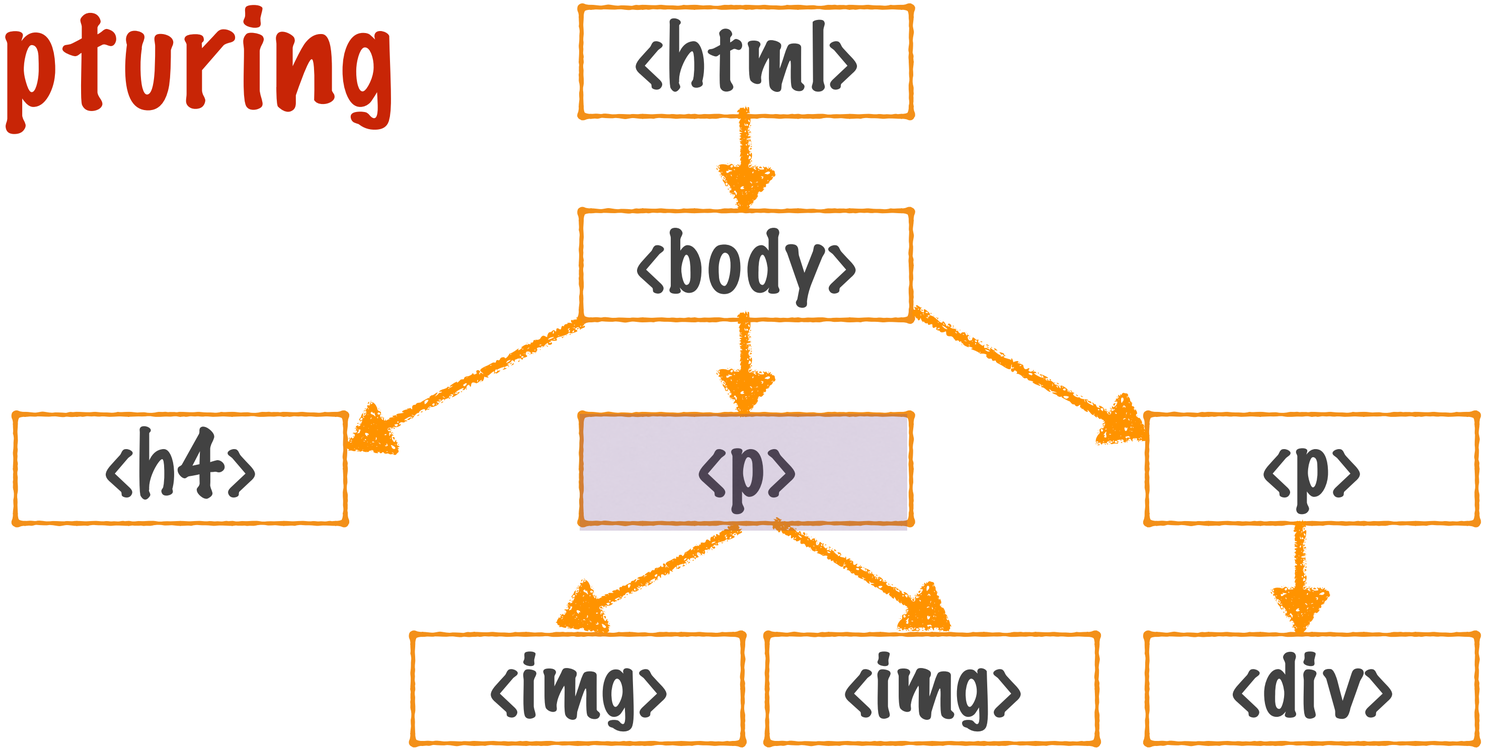
Element <p> is clicked on

Event Capturing



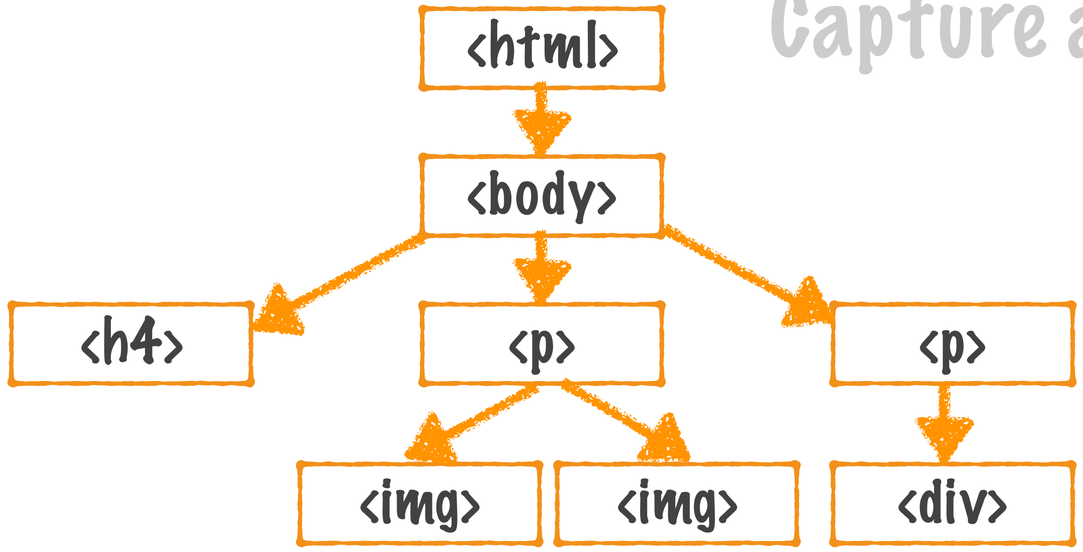
It gets a chance to react to the event

Event Capturing



The click is then propagated downwards to child elements

Capture and bubble phase



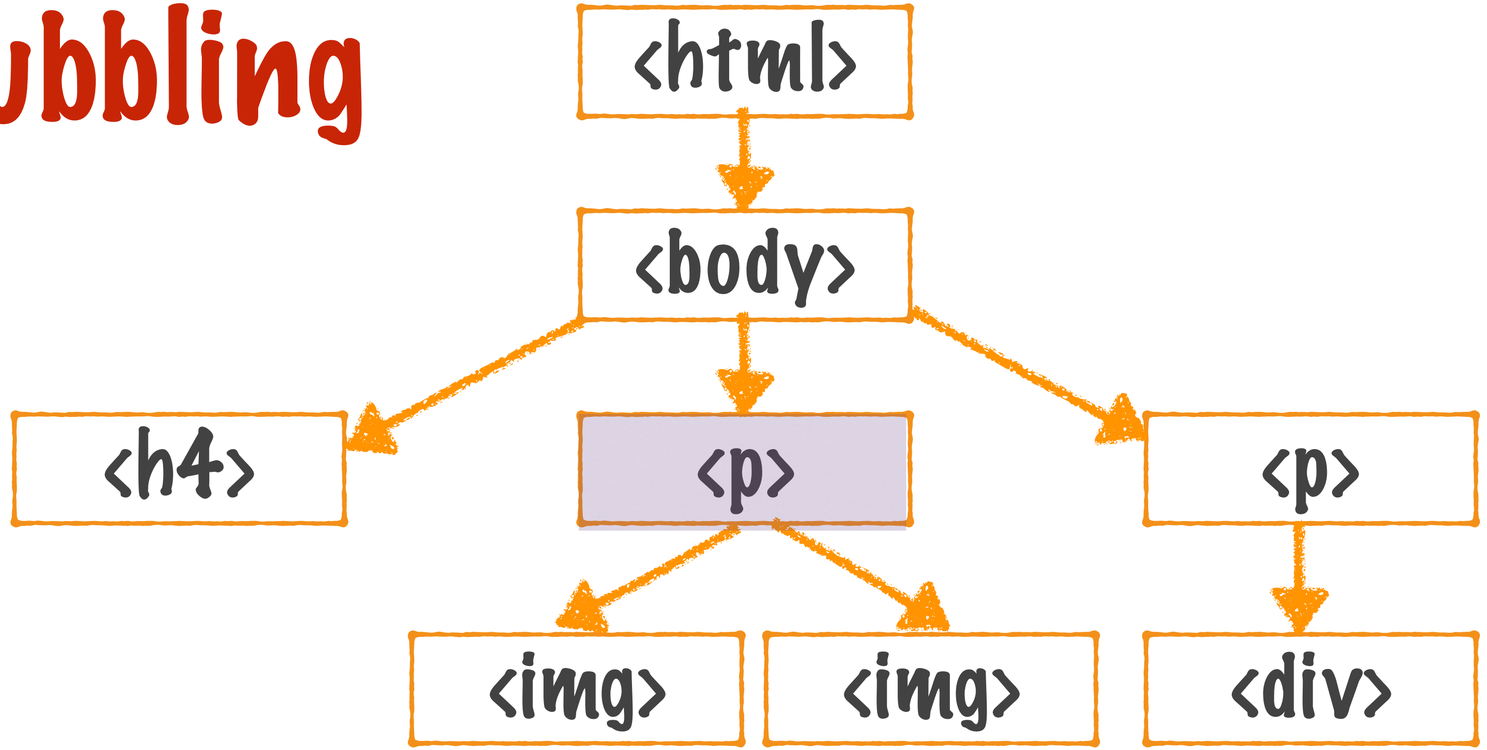
Which other elements receive the events?

These 2 ways in which events can travel are called:

Event Capturing

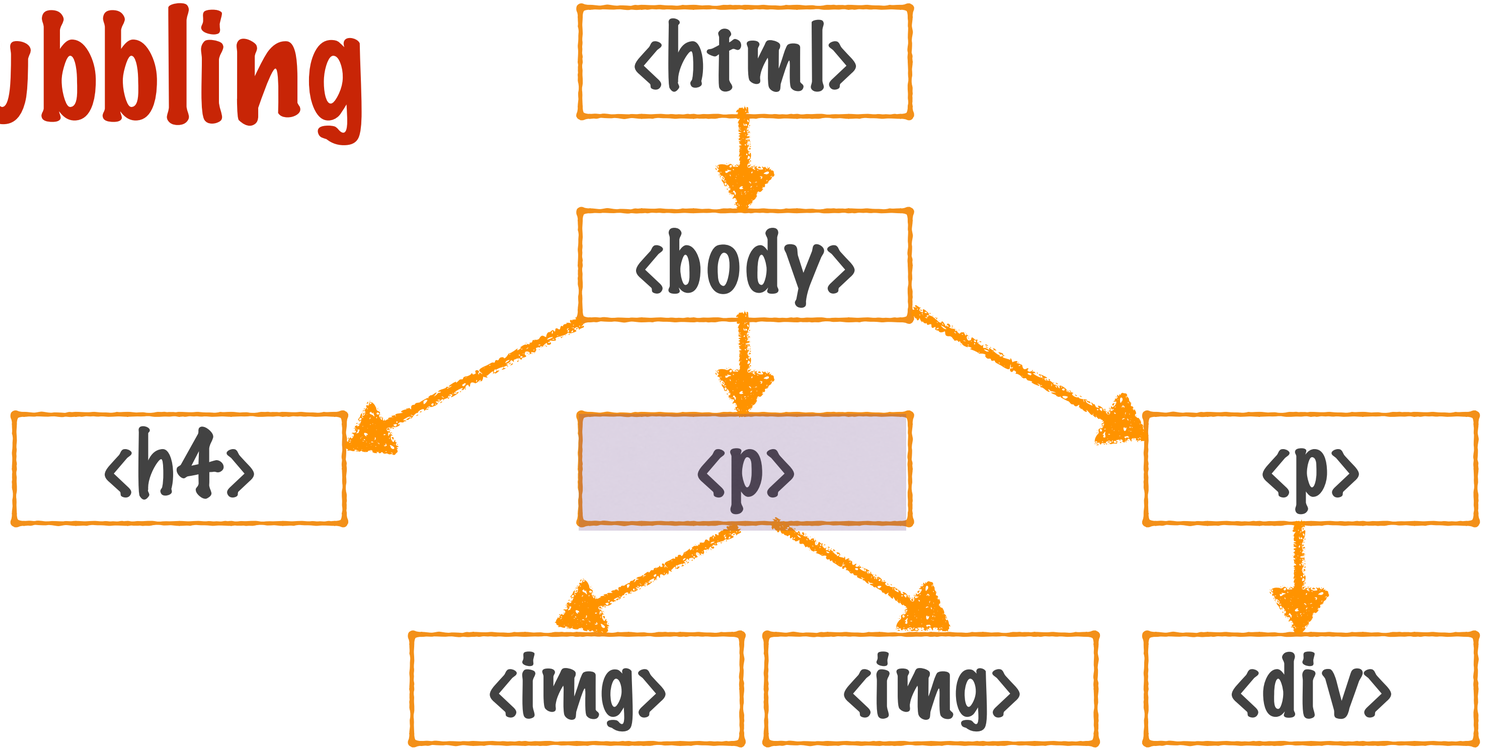
Event Bubbling

Event Bubbling



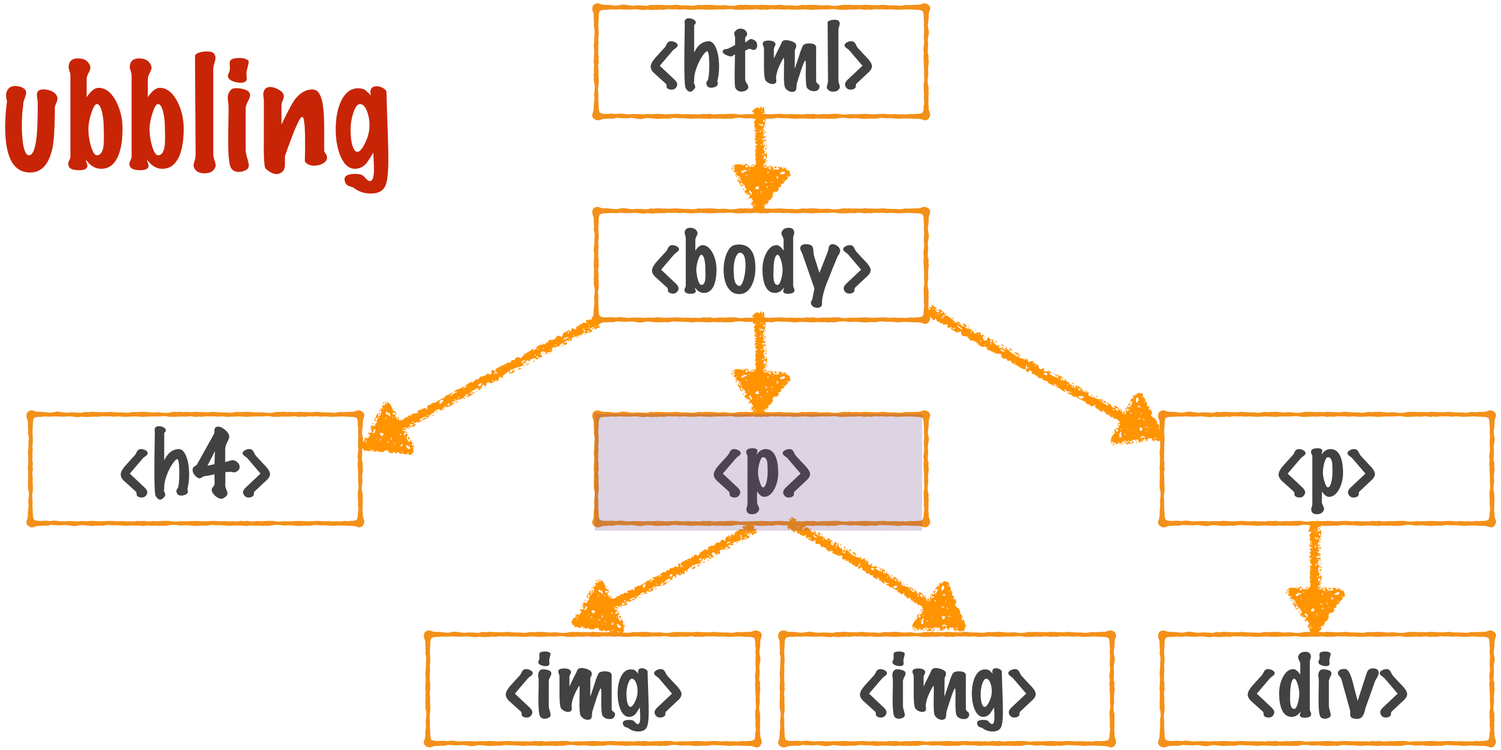
Element <p> is clicked on

Event Bubbling



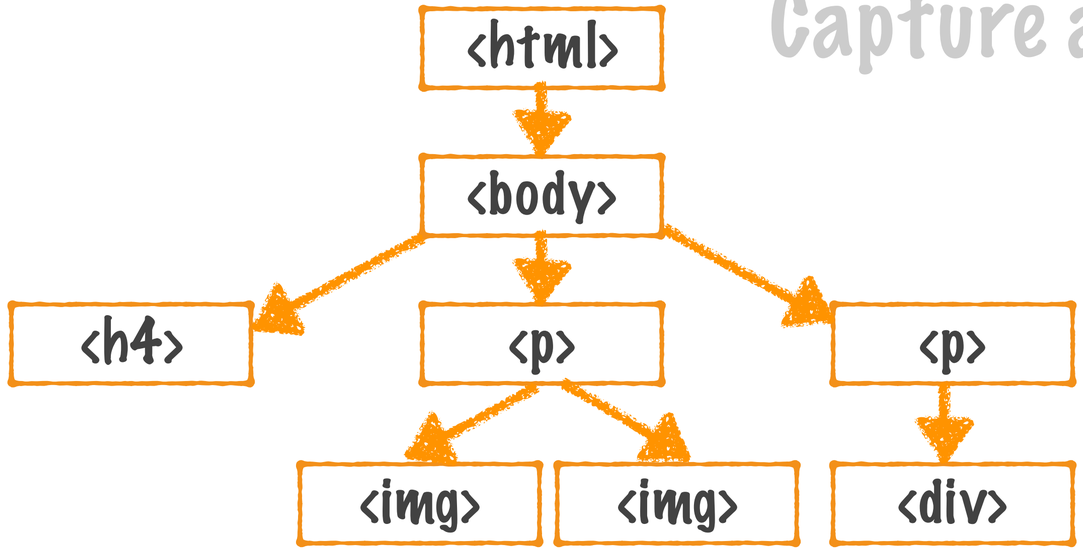
It gets a chance to react to the event

Event Bubbling



This click is them propagated upwards to parent elements

Capture and bubble phase



Which other elements receive the events?

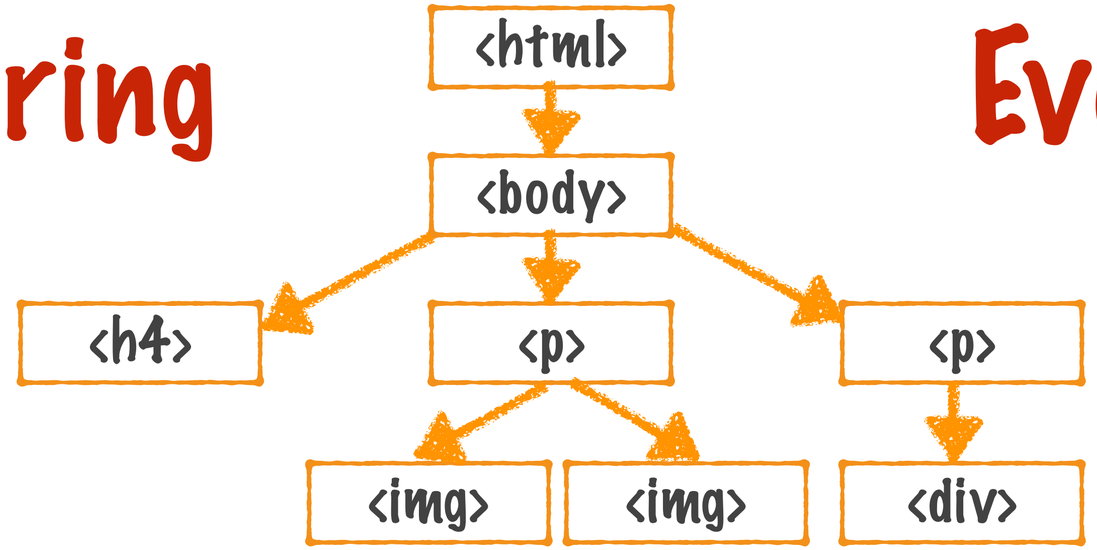
These 2 ways in which events can travel are called:

Event Capturing

Event Bubbling

Capture and bubble phase

Event Capturing



Event Bubbling

So which of these methods do browsers use?

Both!

Capture and bubble phase

So which of these methods do browsers use?

Event Capturing

Event Bubbling

Different browsers originally decided on different models for event propagation

Capture and bubble phase

So which of these methods do browsers use?

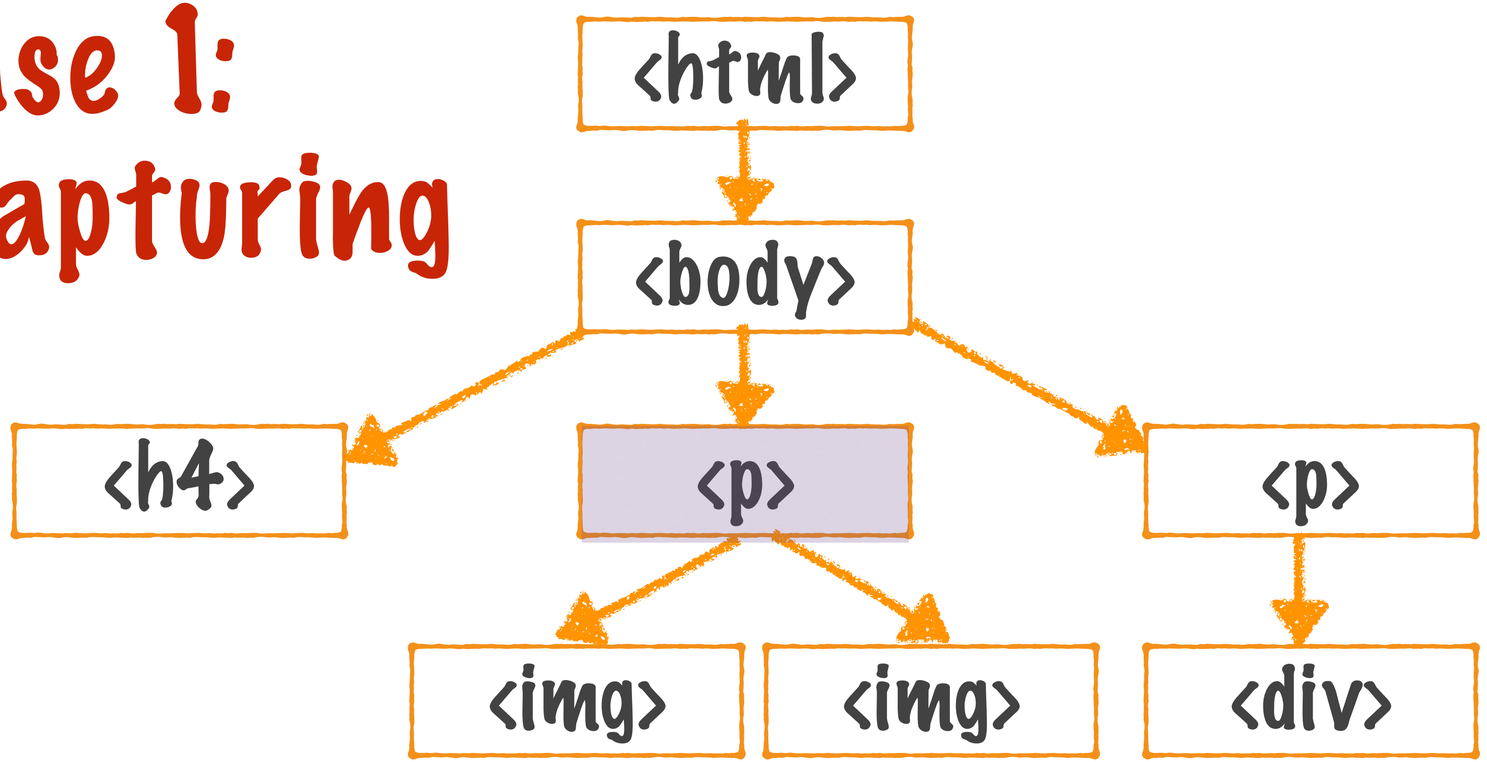
Event Capturing

Event Bubbling

The DOM standard that was ultimately adopted was that both should be supported

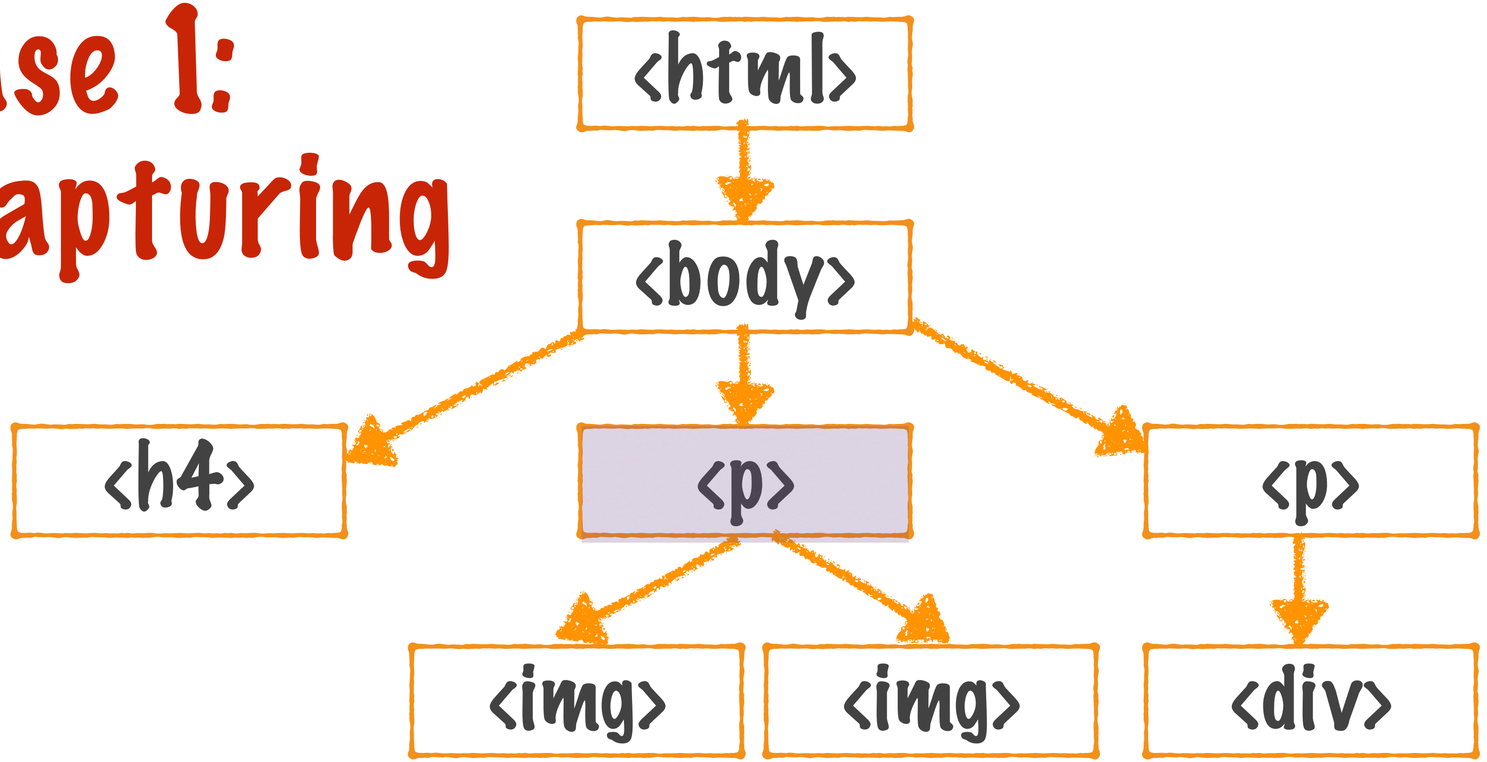
How does that work?

Phase 1: Event Capturing



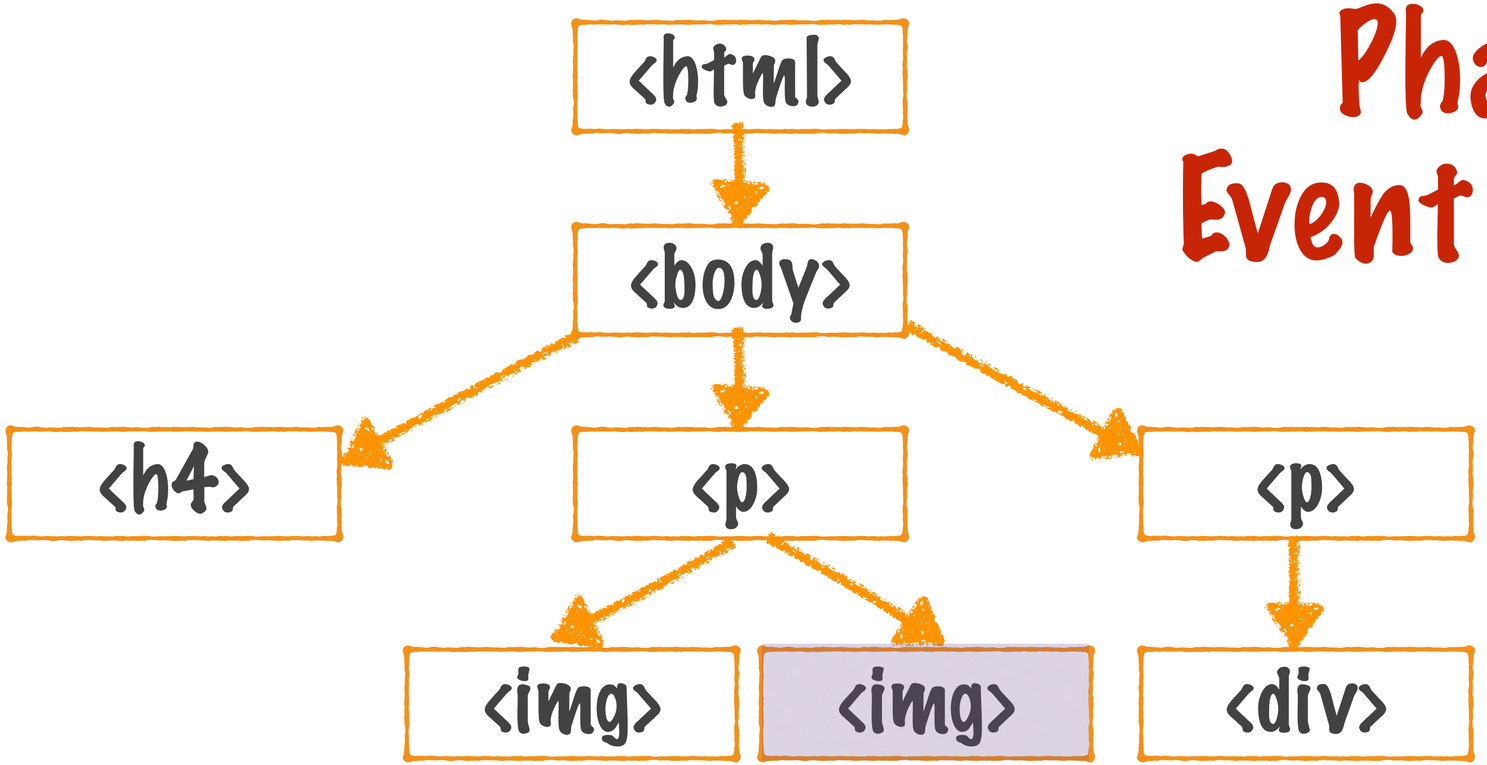
Element <p> is clicked on

Phase 1: Event Capturing



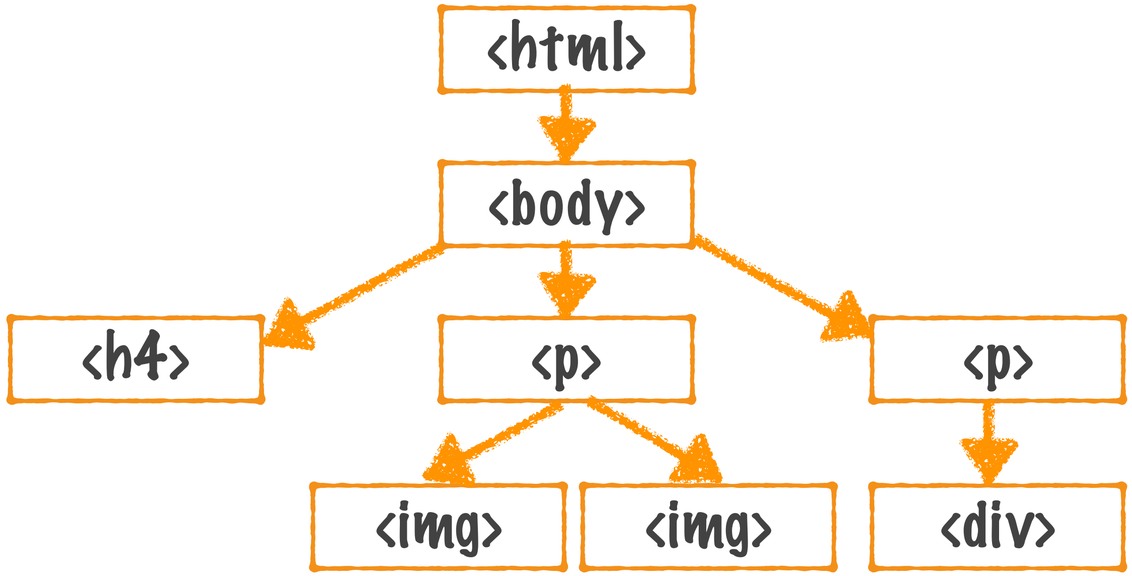
It is propagated downwards to child elements

Phase 2: Event Bubbling



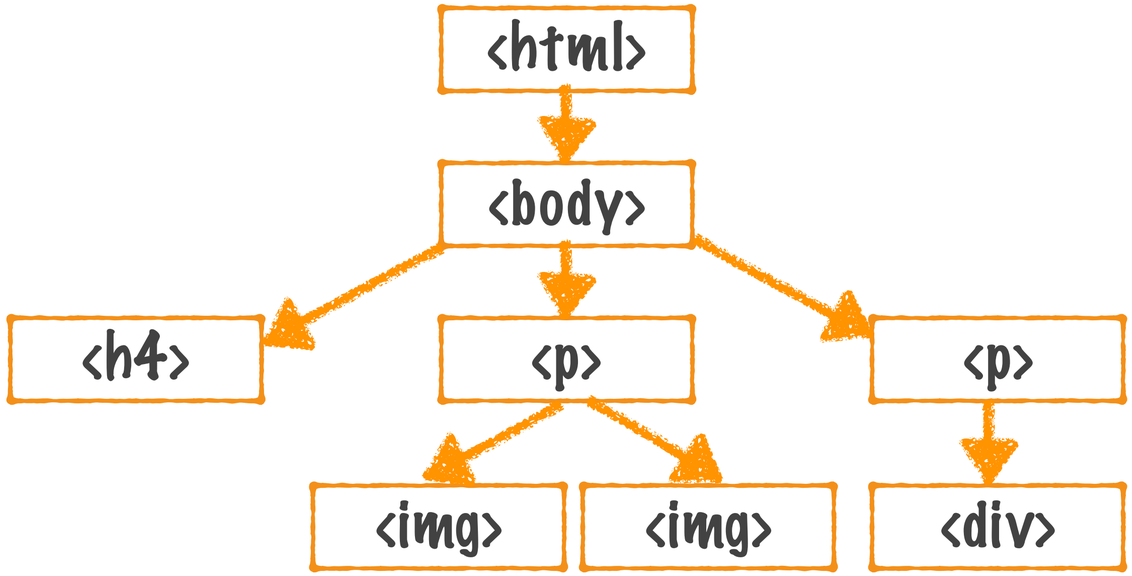
And then propagated upwards to parent elements

Capture and bubble phase



This can cause some strange behavior while handling events

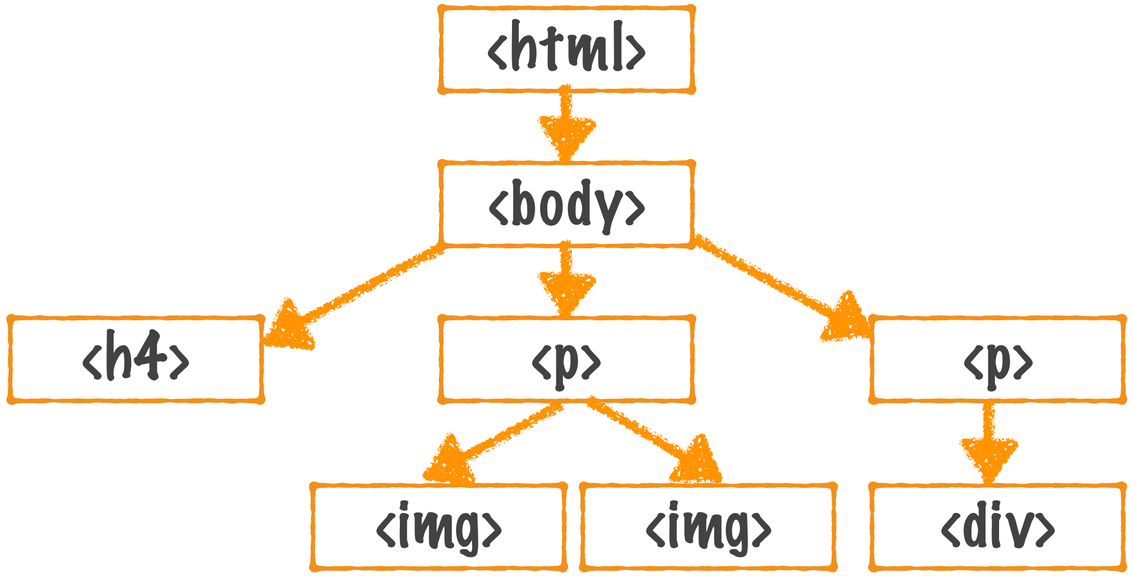
Capture and bubble phase



The <body> element will receive events from its children as well as its ancestors

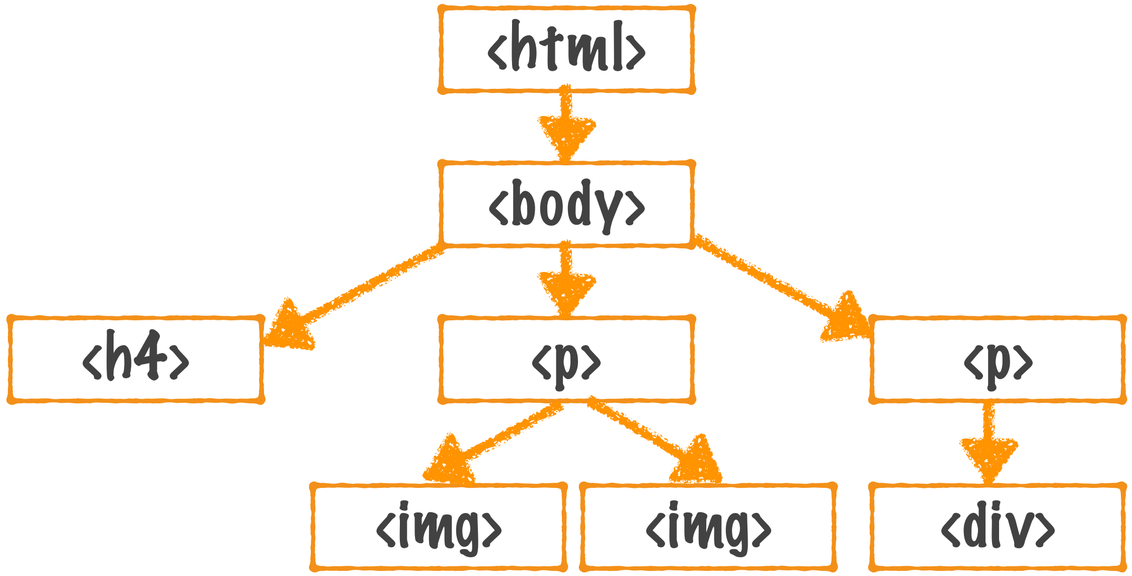
When should the event be handled?

Capture and bubble phase



React by default handles events in the bubble phase

Capture and bubble phase



If you want events to be handled in the capture phase use the "Capture" suffix

onClick -> onClick**Capture**
onMouseOver -> onMouseOver**Capture**

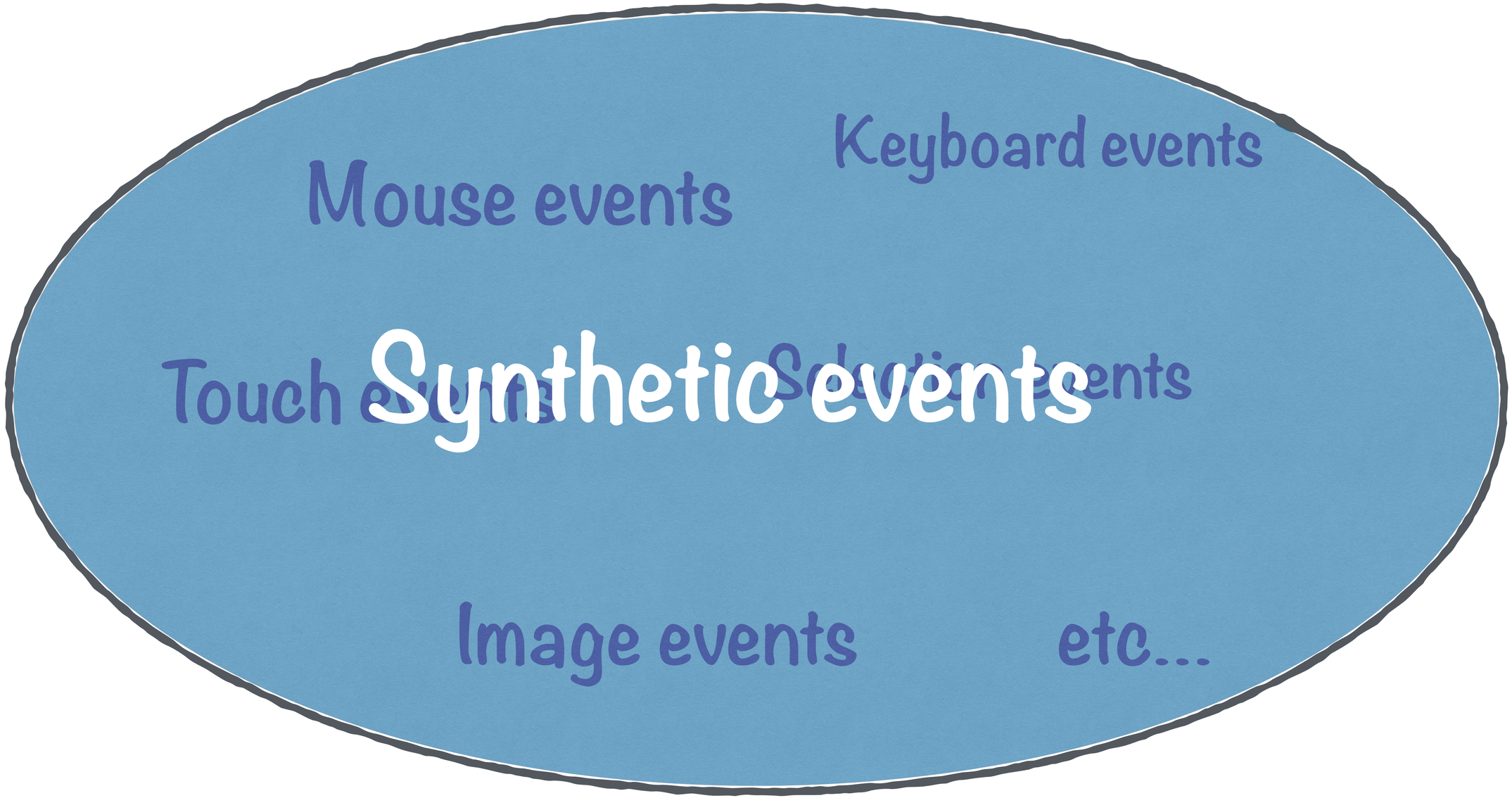
EXAMPLE 19

Events.html

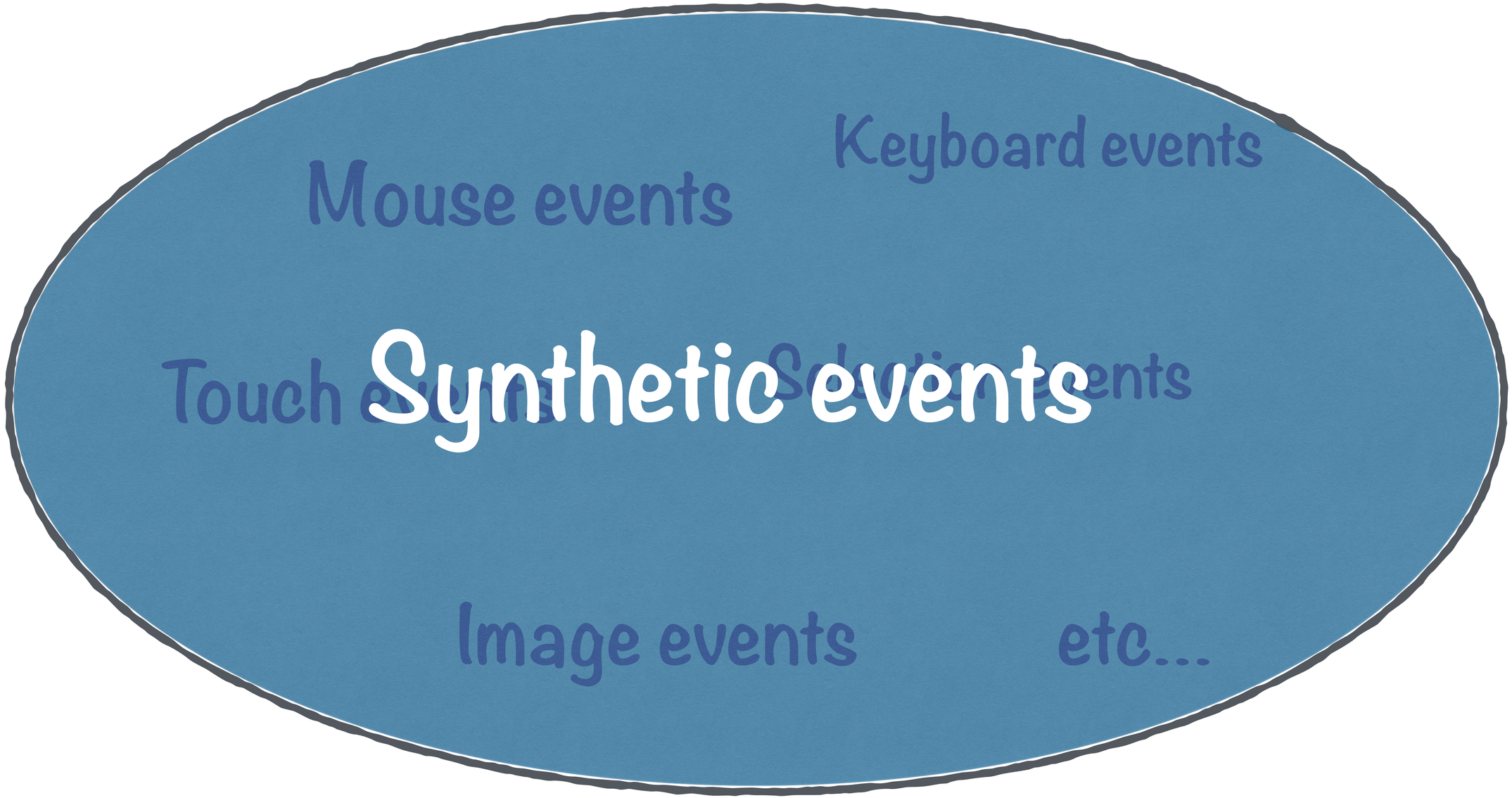
REACT JS

Synthetic events

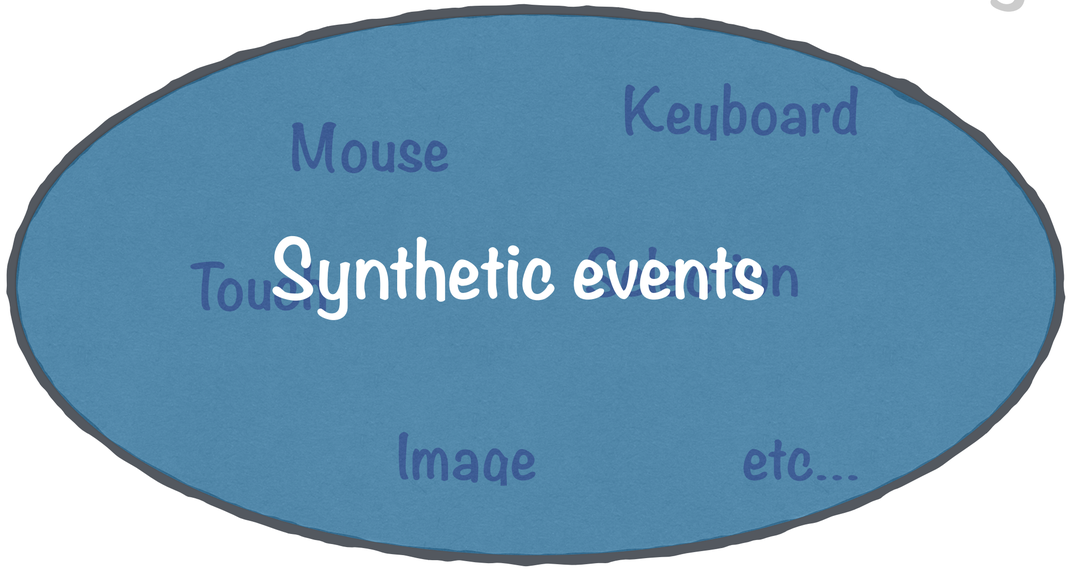
Synthetic events



Synthetic events

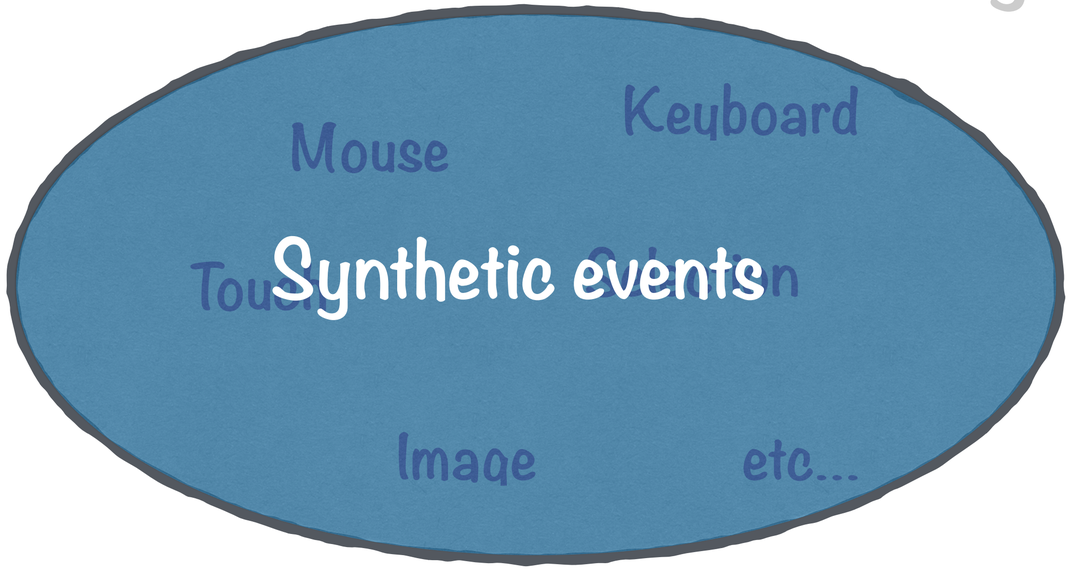


Synthetic events



React wraps the different event types in the native browser into a Synthetic event

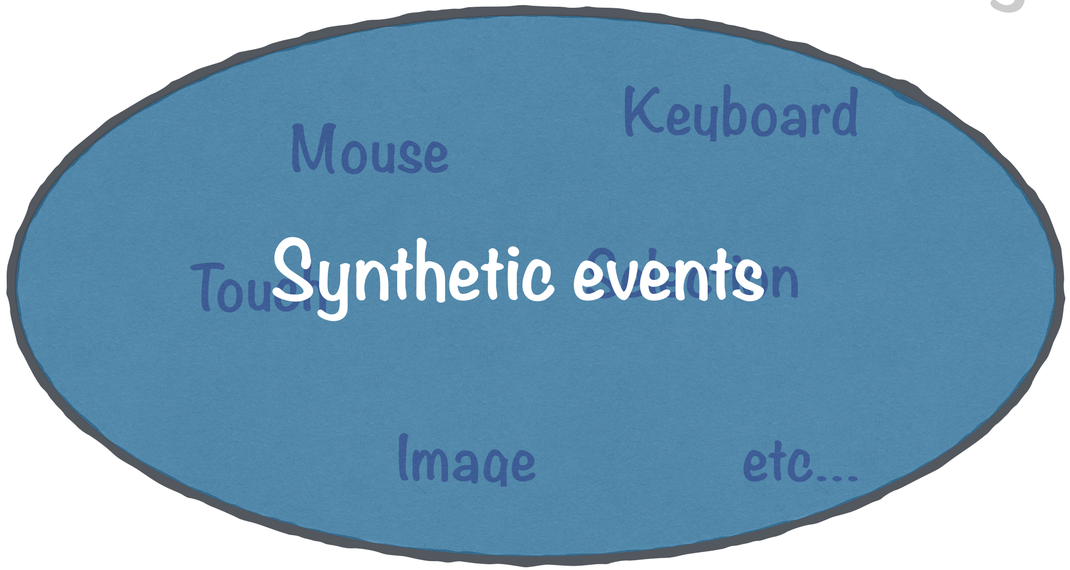
Synthetic events



eliminates cross browser differences

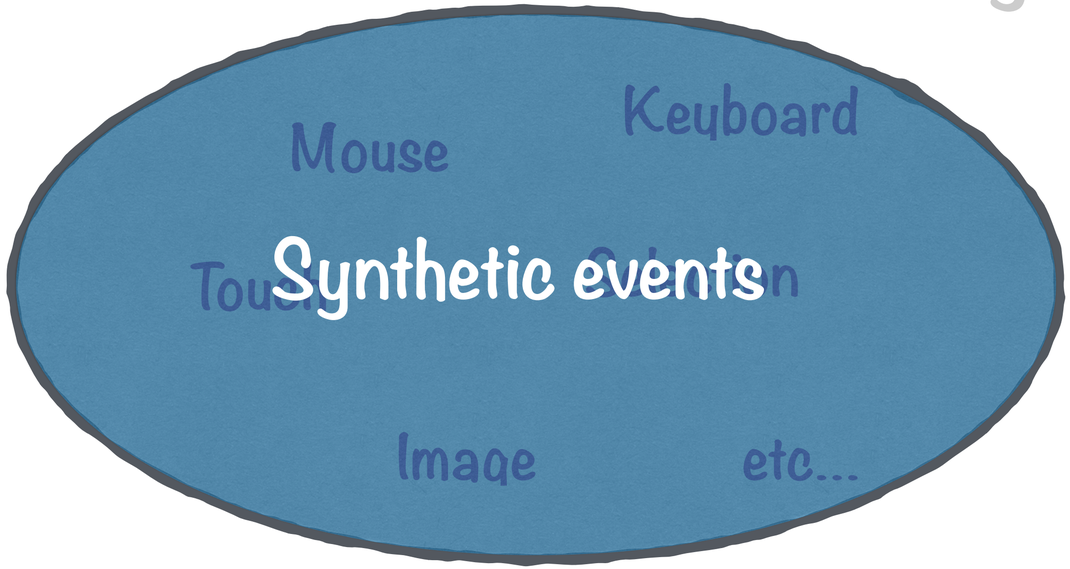
normalizes how different types of events are handled

Synthetic events



boolean bubbles
boolean cancelable
DOMEventTarget currentTarget
boolean defaultPrevented
number eventPhase
boolean isTrusted
DOMEvent nativeEvent
void preventDefault()
boolean isDefaultPrevented()
void stopPropagation()
boolean isPropagationStopped()
DOMEventTarget target
number timeStamp
string type

Synthetic events

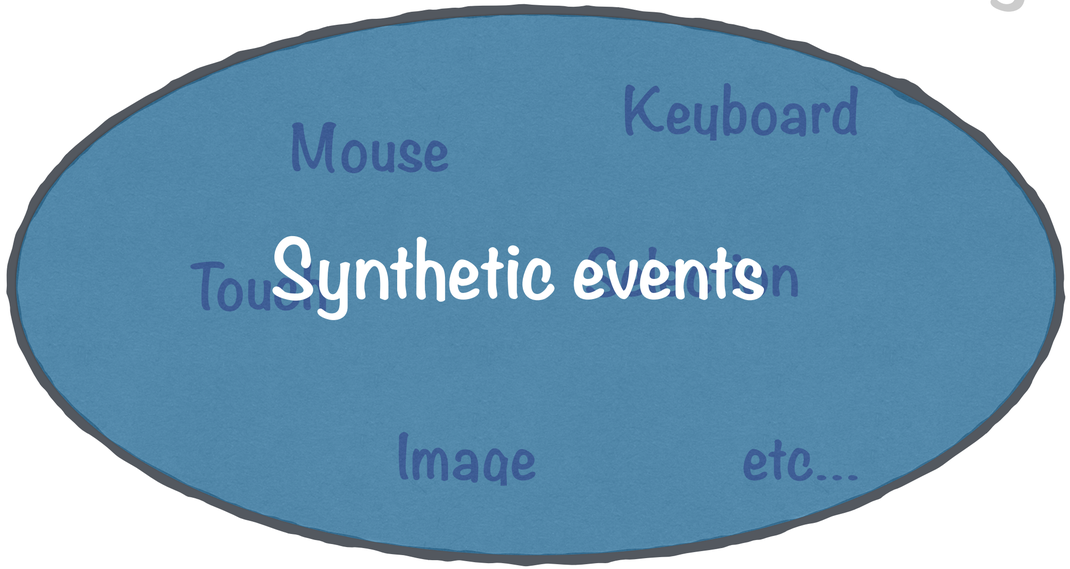



```
void preventDefault()
void stopPropagation()
```

The interface is very similar to the browser's native event

And behaves the same way across browsers!

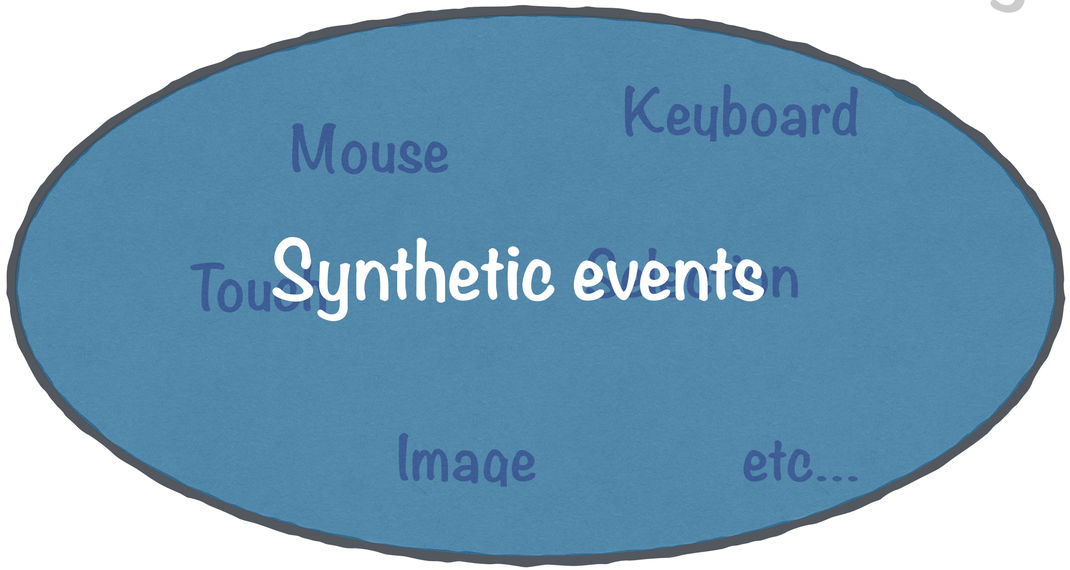
Synthetic events




```
DOMEvent nativeEvent
```

This property allows you to access the browser's native event

Synthetic events



boolean bubbles
boolean cancelable
DOMEventTarget currentTarget
boolean defaultPrevented
number eventPhase
boolean isTrusted
DOMEvent nativeEvent
void preventDefault()
boolean isDefaultPrevented()
void stopPropagation()
boolean isPropagationStopped()
DOMEventTarget target
number timeStamp
string type

```
boolean bubbles
boolean cancelable
DOMEventTarget currentTarget
boolean defaultPrevented
number eventPhase
boolean isTrusted
DOMEvent nativeEvent
void preventDefault()
boolean isDefaultPrevented()
void stopPropagation()
boolean isPropagationStopped()
DOMEventTarget target
number timeStamp
string type
```

All synthetic events have these properties

Synthetic events

```
boolean bubbles
boolean cancelable
DOMEventTarget currentTarget
boolean defaultPrevented
number eventPhase
boolean isTrusted
DOMEvent nativeEvent
void preventDefault()
boolean isDefaultPrevented()
void stopPropagation()
boolean isPropagationStopped()
DOMEventTarget target
number timeStamp
string type
```

In addition they have properties of the specific kind of event

Keyboard events

boolean bubbles
boolean cancelable
DOMEventTarget currentTarget
boolean defaultPrevented
number eventPhase
boolean isTrusted
DOMEvent nativeEvent
void preventDefault()
boolean isDefaultPrevented()
void stopPropagation()
boolean isPropagationStopped()
DOMEventTarget target
number timeStamp
string type

boolean altKey
number charCode
boolean ctrlKey
boolean
getModifierState(key)
string key
number keyCode
string locale
number location
boolean metaKey
boolean repeat
boolean shiftKey
number which

Mouse events

```
boolean bubbles
boolean cancelable
DOMEventTarget currentTarget
boolean defaultPrevented
number eventPhase
boolean isTrusted
DOMEvent nativeEvent
void preventDefault()
boolean isDefaultPrevented()
void stopPropagation()
boolean isPropagationStopped()
DOMEventTarget target
number timeStamp
string type
```

```
boolean altKey
number button
number buttons
number clientX
number clientY
boolean ctrlKey
boolean getModifierState(key)
boolean metaKey
number pageX
number pageY
DOMEventTarget relatedTarget
number screenX
number screenY
boolean shiftKey
```

Event pooling

The synthetic event object wrapper is re-used to improve performance

Event pooling

After the event handler callbacks have been called the event properties will b nullified

Event pooling

```
function onClick(event) {
    console.log(event);
    console.log(event.type);
    const eventType = event.type;

    setTimeout(function() {
        console.log(event.type);
        console.log(eventType);
    }, 0);

    this.setState({clickEvent: event});

    this.setState({eventType: event.type});
}
```

Accessible within the event handler

Event pooling

```
function onClick(event) {
    console.log(event);
    console.log(event.type);
    const eventType = event.type;

    setTimeout(function() {
        console.log(event.type);
        console.log(eventType);
    }, 0);

    this.setState({clickEvent: event});

    this.setState({eventType: event.type});
}
```

Store the event type in another variable

Event pooling

```
function onClick(event) {
    console.log(event);
    console.log(event.type);
    const eventType = event.type;

    setTimeout(function() {
        console.log(event.type);
        console.log(eventType);
    }, 0);

    this.setState({clickEvent: event});

    this.setState({eventType: event.type});
}
```

This executes just after the handler

Event pooling

```
function onClick(event) {
    console.log(event);
    console.log(event.type);
    const eventType = event.type;

    setTimeout(function() {
        console.log(event.type);
        console.log(eventType);
    }, 0);

    this.setState({clickEvent: event});

    this.setState({eventType: event.type});
}
```

The event object will be null

Event pooling

```
function onClick(event) {
    console.log(event);
    console.log(event.type);
    const eventType = event.type;

    setTimeout(function() {
        console.log(event.type);
        console.log(eventType);
    }, 0);

    this.setState({clickEvent: event});

    this.setState({eventType: event.type});
}
```

The local variable eventType still has its value

Event pooling

```
function onClick(event) {
    console.log(event);
    console.log(event.type);
    const eventType = event.type;

    setTimeout(function() {
        console.log(event.type);
        console.log(eventType);
    }, 0);

    this.setState({clickEvent: event});

    this.setState({eventType: event.type});
}
```

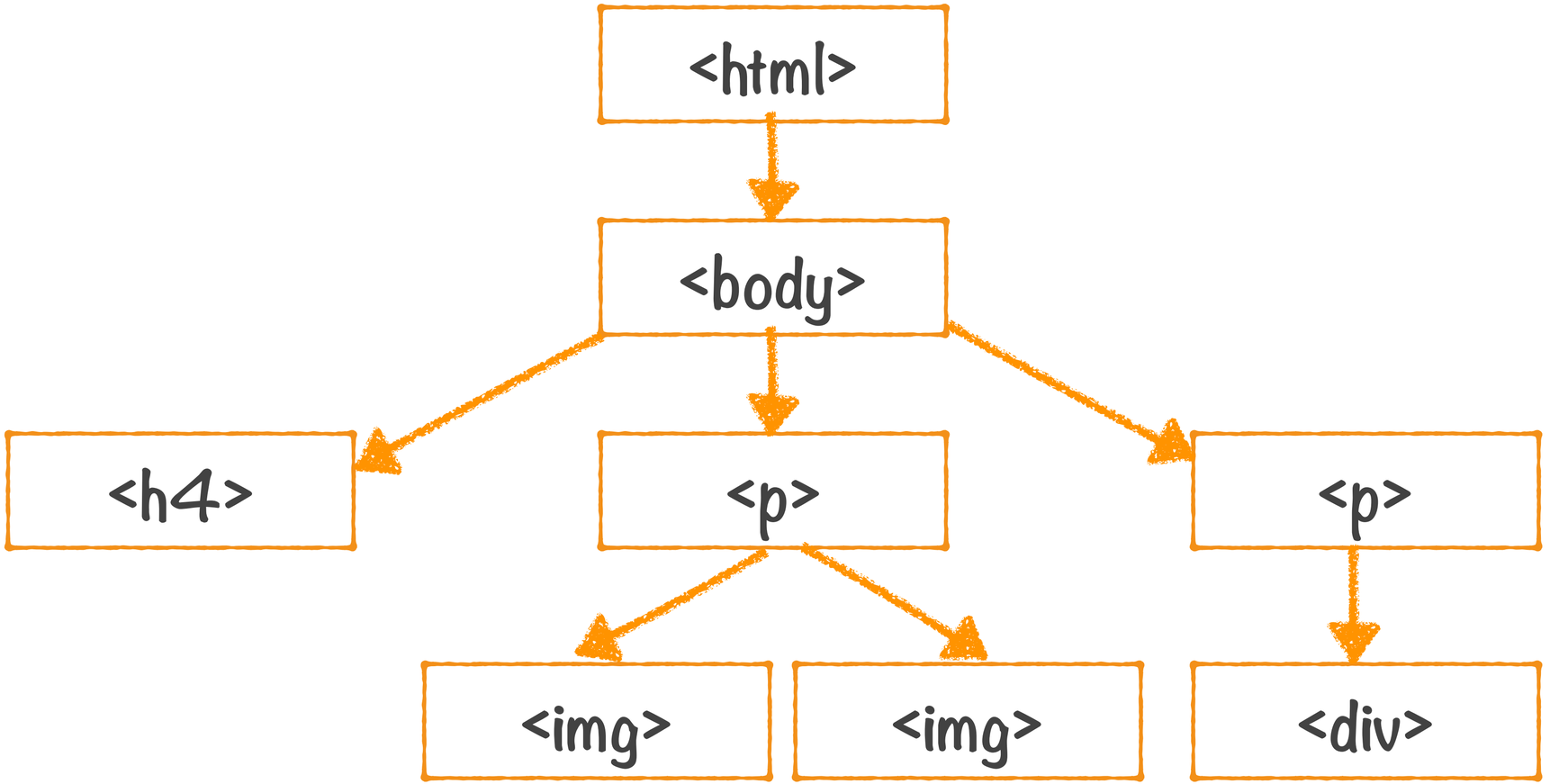
All event properties will be nullified

Event pooling

```
function onClick(event) {
    console.log(event);
    console.log(event.type);
    const eventType = event.type;

    setTimeout(function() {
        console.log(event.type);
        console.log(eventType);
    }, 0);

    this.setState({clickEvent: event});

    this.setState({eventType: event.type});
}
```

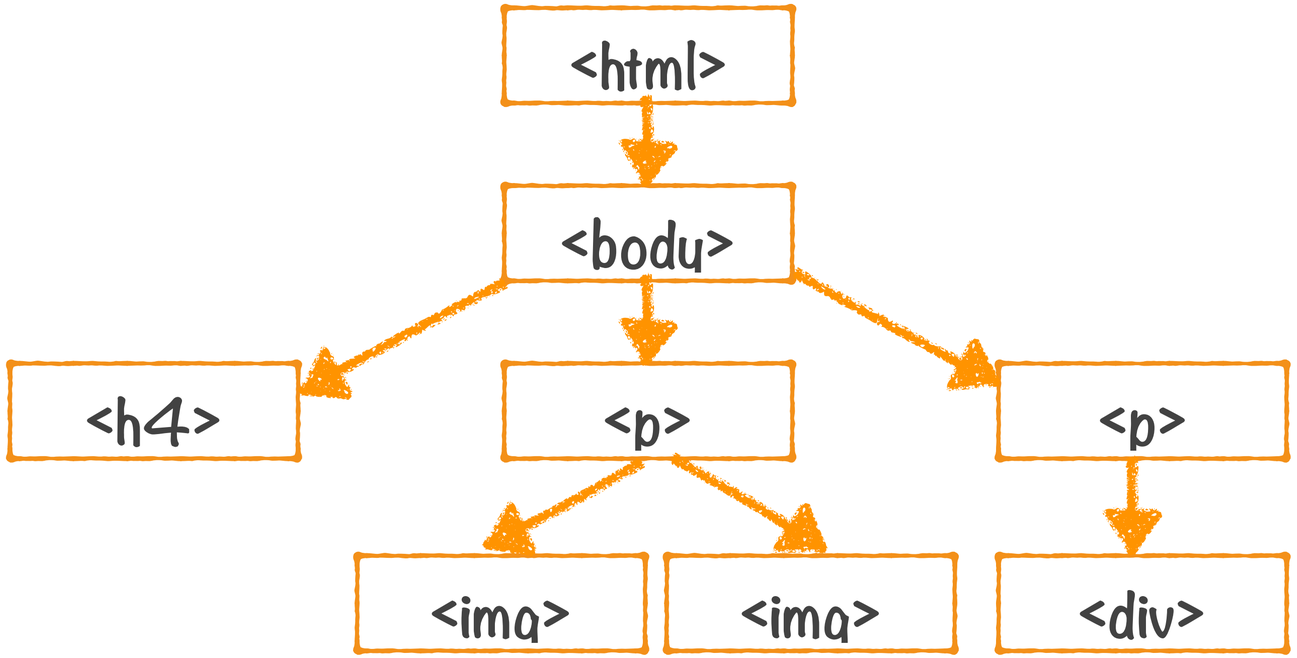
The eventType is copied over so the value remains

event delegation



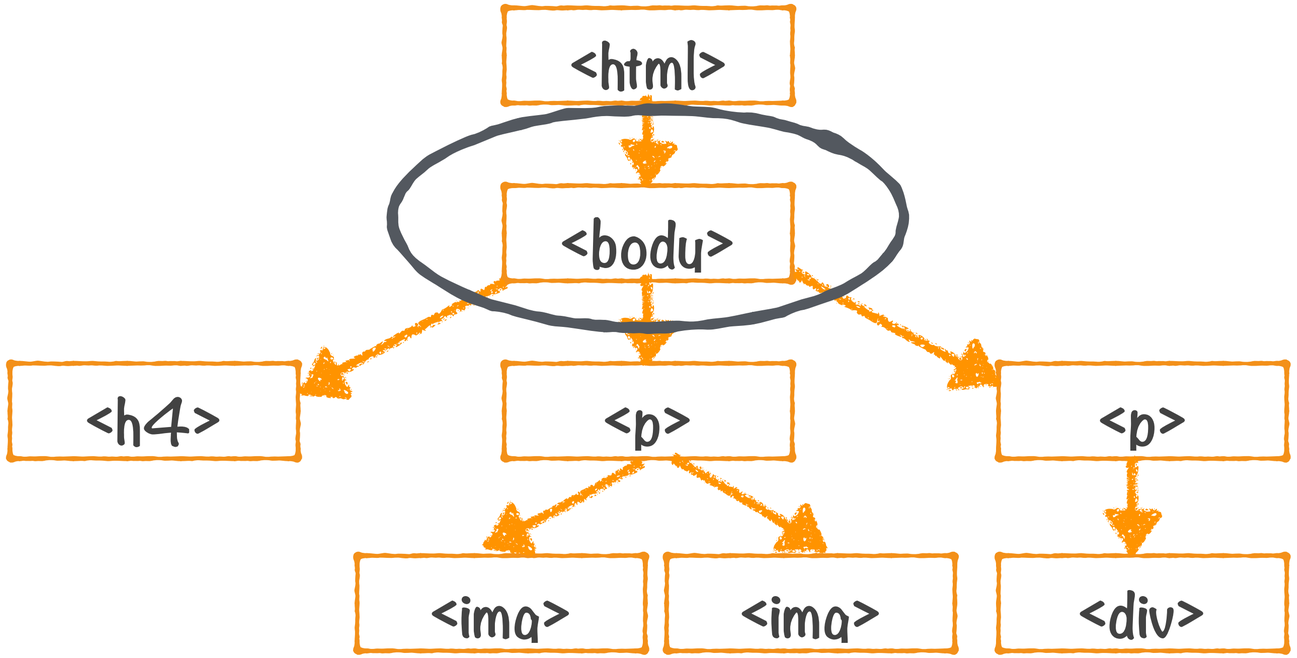
event delegation

For performance reasons React does not associate event listeners with individual elements



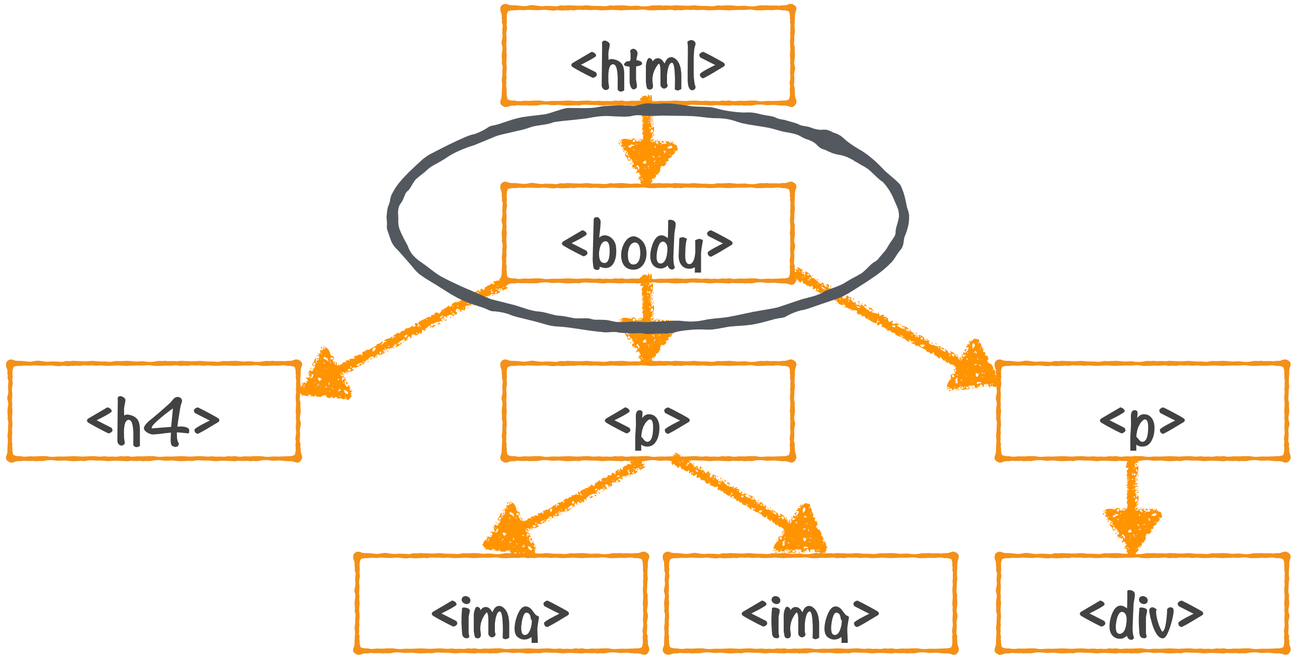
event delegation

One listener is assigned to the document.body at the top of the hierarchy



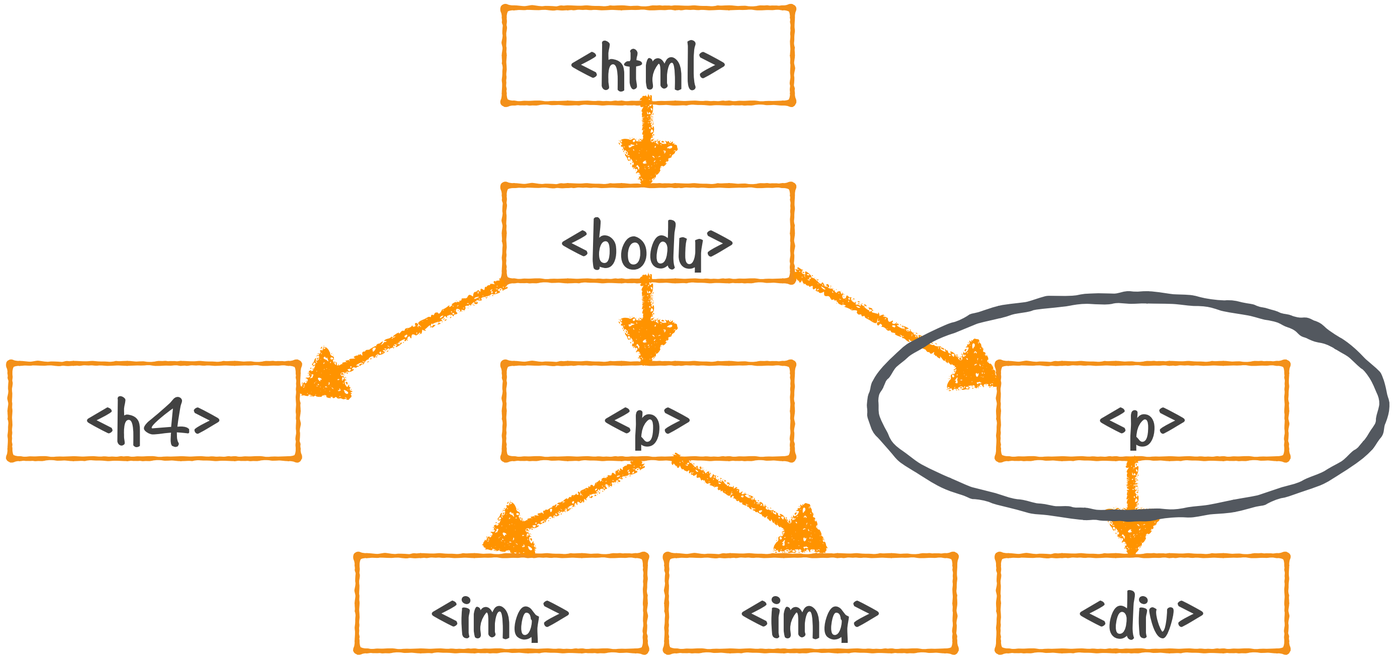
event delegation

That top-level handler is responsible for **all** elements



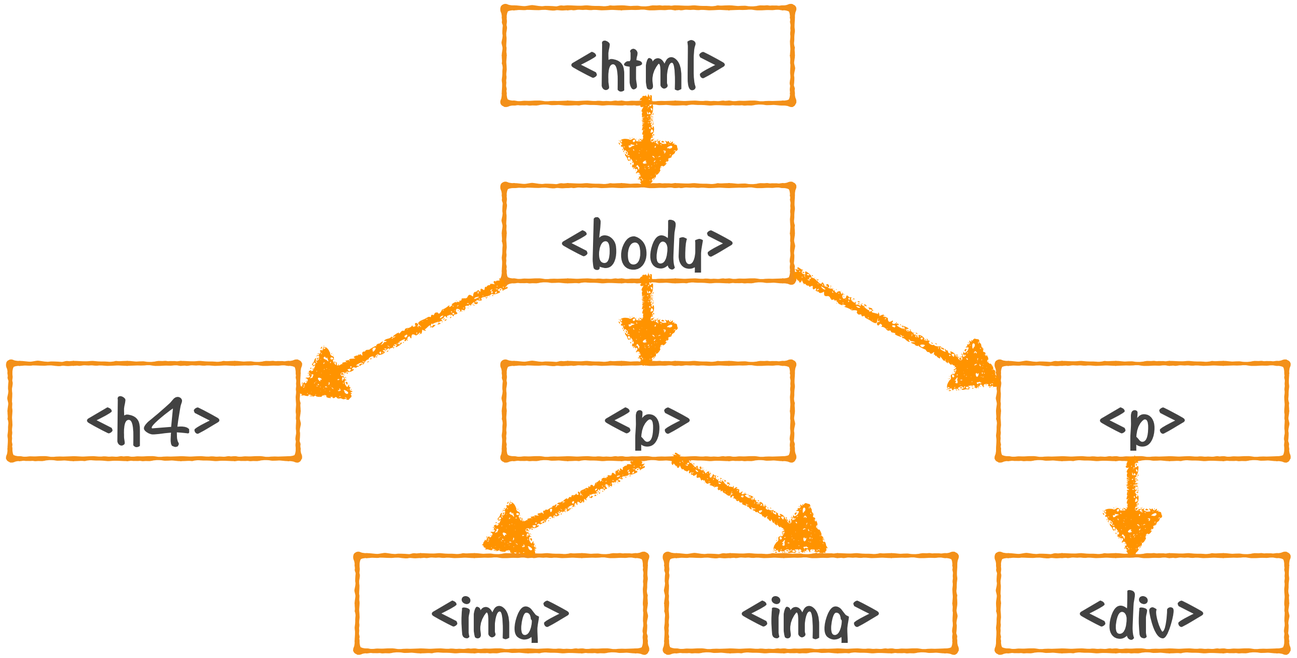
event delegation

It calls the appropriate event handler behind that scenes



event delegation

The event handling is super efficient and optimized



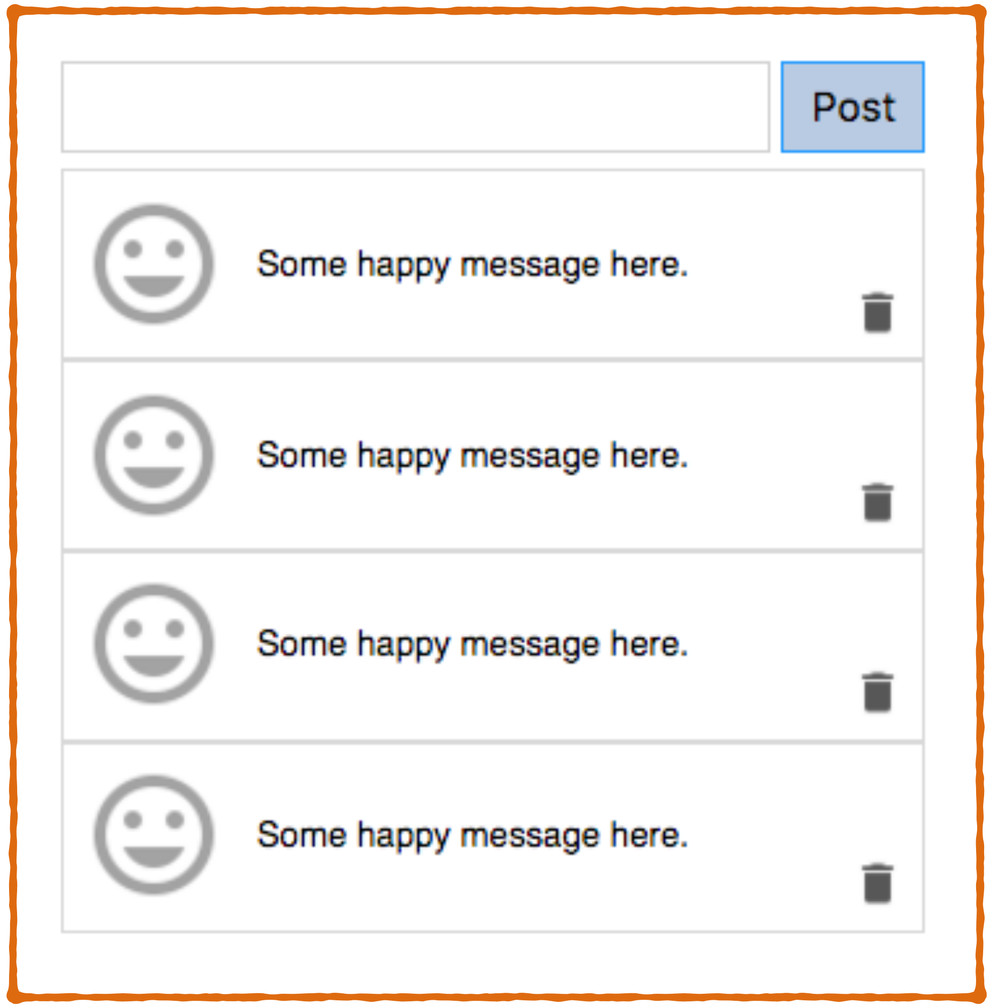
EXAMPLE 20

SyntheticEvents.html

REACT JS

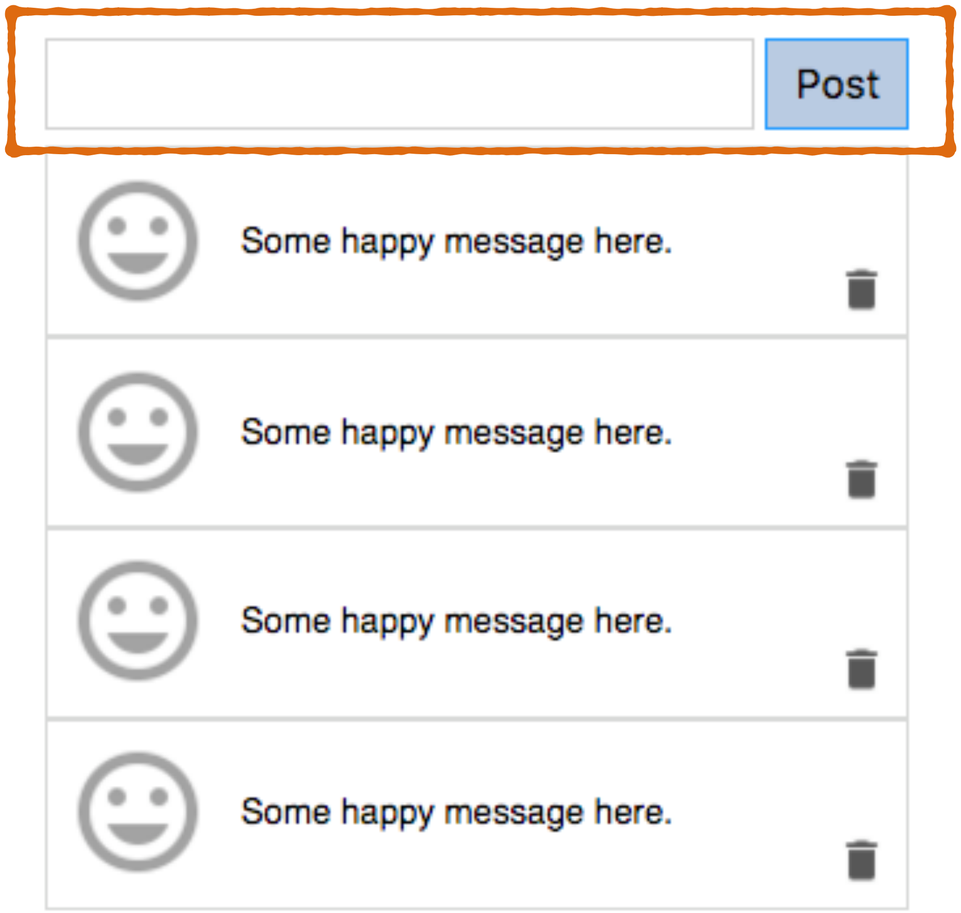
Component composition

Component composition



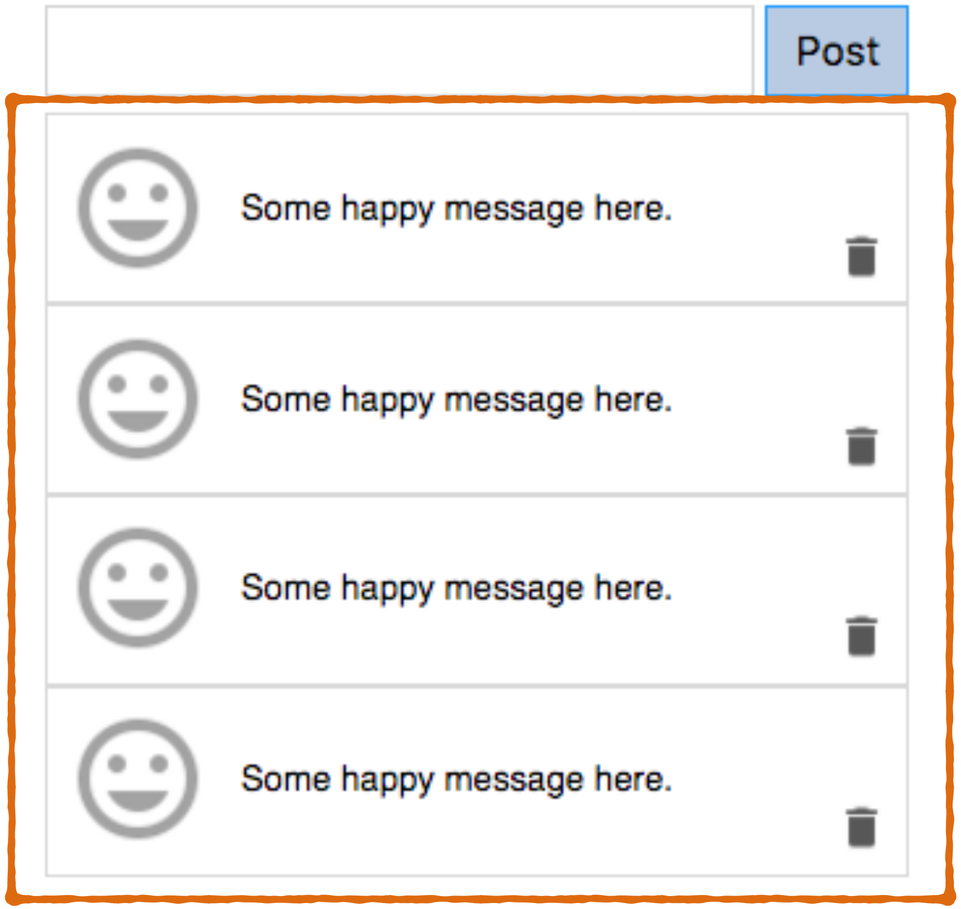
CommentApp

Component composition



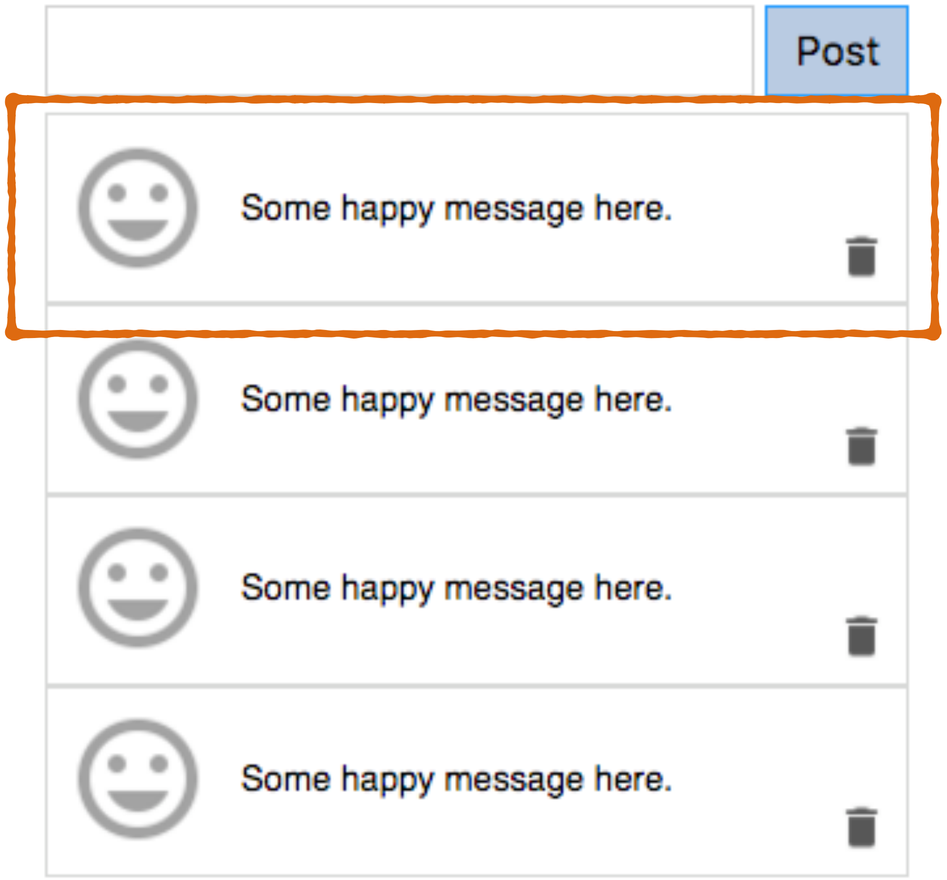
CommentBox

Component composition



CommentList

Component composition



Comment

EXAMPLE 21

CommentApp.html

EXAMPLE 22

CommentAppWithProps.html

EXAMPLE 23

CommentAppChildComponentsChangeState.html

REACT JS

Component lifecycle

React components have a lifecycle

inserted into the DOM — mounting

rendered to the DOM — updating

removed from the DOM — unmounting

React components have a lifecycle

mounting

updating

unmounting

Component lifecycle

React components have a lifecycle

mounting

updating

unmounting

There are methods in each of these phases that we can hook into and customize

Component lifecycle

mounting

```
getInitialState()

componentWillMount()

render()

componentDidMount()
```

Component lifecycle

mounting

`getInitialState()`

Called before a component is inserted to set up the initial state for the component

Component lifecycle

mounting

`componentWillMount()`

Invoked immediately before the component is inserted into the DOM

Component lifecycle

mounting

`componentWillMount()`

Calling setState() here will not trigger a re-rendering, render() will be called just once

Component lifecycle

mounting

`render()`

The component draws itself onto the screen - do not modify state in this function!

Component lifecycle

mounting

`componentDidMount()`

Invoked immediately after the component is inserted into the DOM

Component lifecycle

mounting

`componentDidMount()`

DOM manipulations and interactions can be performed in this method

EXAMPLE 24

LifecycleMountPhase.html

React components have a lifecycle

mounting

updating

unmounting

unmounting

`componentWillUnmount()`

Called immediately before a component is removed from the DOM

unmounting

`componentWillUnmount()`

Component clean up method to remove event handlers or anything new DOM elements that you've set up

EXAMPLE 25

LifecycleUnmountPhase.html

React components have a lifecycle

mounting

updating

unmounting

updating

Here the component is already inserted into the DOM and is updated in response to state changes

updating

React has a bunch of hooks into state mutations and deciding whether the actual update should occur

Component lifecycle

updating

```
componentWillReceiveProps()

shouldComponentUpdate()

componentWillUpdate()

render()

componentDidUpdate()
```

Component lifecycle

updating

`componentWillReceiveProps(nextProps)`

Invoked each time a component receives new properties from the parent

updating

`componentWillReceiveProps(nextProps)`

Used to compare the old and new values of props and determine whether an update is to be made

EXAMPLE 26

LifecycleUpdateComponentWillReceiveProps.html

Component lifecycle

updating

```
componentWillReceiveProps()

shouldComponentUpdate()

componentWillUpdate()

render()

componentDidUpdate()
```

Component lifecycle

updating

```
shouldComponentUpdate(nextProps, nextState)
```

Allows us to decide whether the next state should trigger a re-rendering or not

Component lifecycle

updating

```
shouldComponentUpdate(nextProps, nextState)
```

Do not call setState() in this method - the next state has already been determined

Component lifecycle

updating

```
shouldComponentUpdate(nextProps, nextState)
```

true

```
componentWillUpdate()
render()
componentDidUpdate()
```

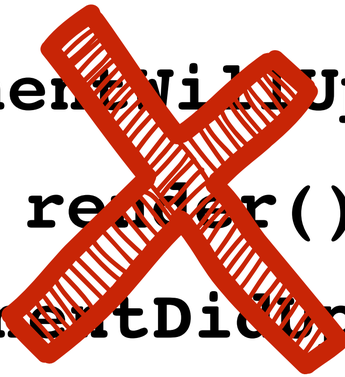
Will be called

Component lifecycle

updating

```
shouldComponentUpdate(nextProps, nextState)
```

false



Will not be called

EXAMPLE 27

LifecycleUpdateShouldComponentUpdate.html

Component lifecycle

updating

```
componentWillUpdate(nextProps, nextState)
```

Invoked immediately before

React updates the DOM

Component lifecycle

updating

```
componentDidUpdate(prevProps, prevState)
```

Invoked immediately after

React updates the DOM

EXAMPLE 28

LifecycleUpdatePhase.html