The DOM API & Events

Anna Azzam | Frontend Developer | anna.a@canva.com

Quick Web Recap

HTML

Markup language creating web documents

CSS

Style sheet language for applying styles to a document

JavaScript

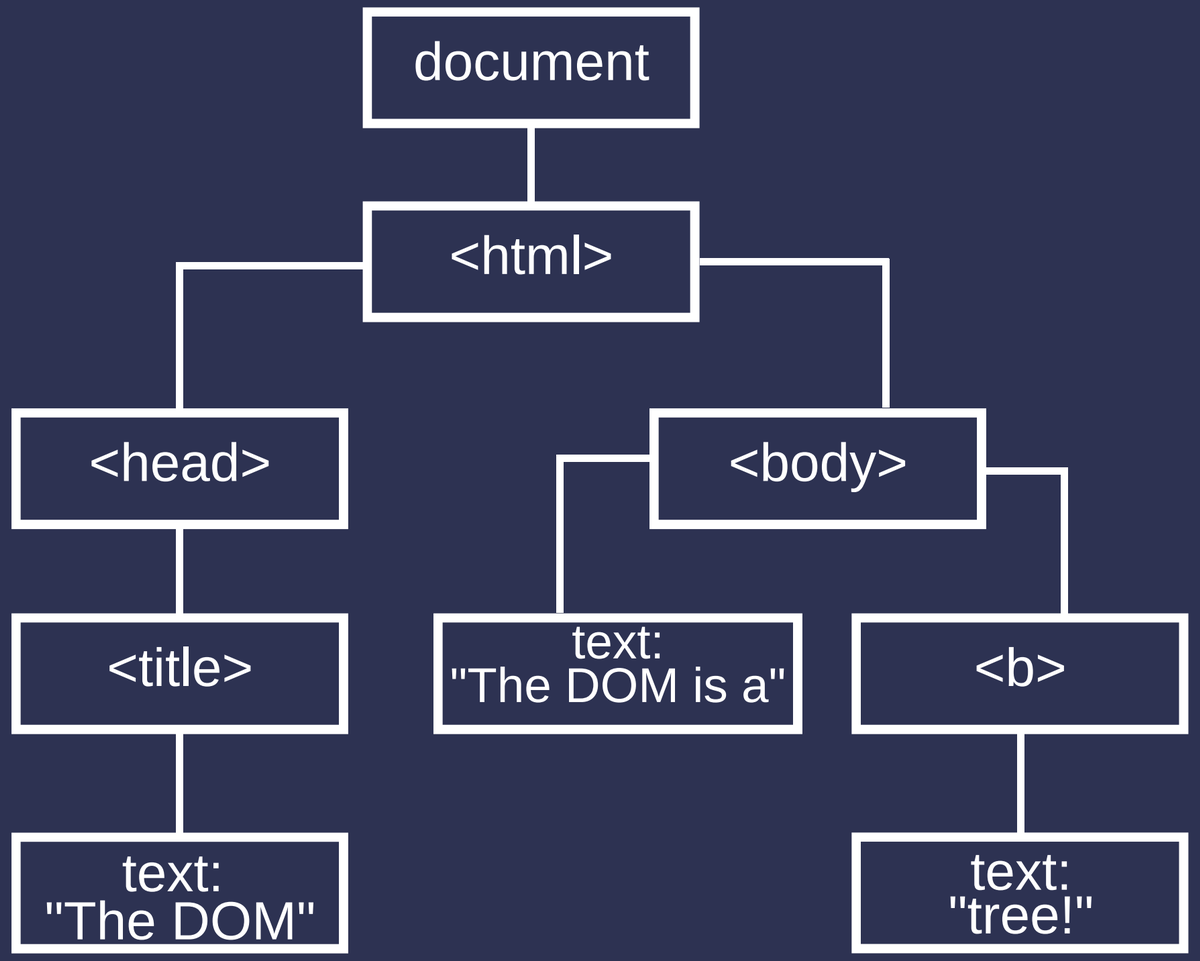
Scripting language to make your web page dynamic

What is the DOM?

The DOM (Document Object Model) is an interface that allows JavaScript to interact with HTML through the browser.

Tree-like structure of the DOM

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>The DOM</title>
  </head>
  <body>
    The DOM is a <b>tree!</b>
  </body>
</html>
```



DOM Data Types

To understand how the DOM is represented, we will introduce some new data types

Document

The type of the `document` object. Represents the root of the entire DOM.

Element

A node in the DOM tree. Objects of this type implement an interface that allows interacting with the document.

NodeList

An array of elements, like the kind that is returned by the method `getElementsByTagName()`

Understanding DOM Elements

- Element is the base class for all types of objects in the Document
- Different HTML tags/elements correspond to different Element types in JS
- Different Element types include:

`HTMLInputElement`, `HTMLSpanElement`, `HTMLDivElement`, `HTMLScriptElement`, `HTMLHeadingElement`,`HTMLImageElement`...

Understanding DOM Elements

The `Element` interface provides us with many properties and methods, e.g.

`clientHeight, clientWidth, clientLeft, clientTop`
`scrollTo(), setAttribute()`

Specific types of elements may also have different attributes/methods, e.g. for HTMLInputElement

`focus() - focuses the cursor on this input element`
`select() - selects all the text in the input element`

How can we interact with the DOM in JavaScript?

Reading the DOM

```
// Returns an html element with the given id
document.getElementById(id);

// Returns a DOM HTMLCollection of all matches
document.getElementsByTagName(name);
document.getElementsByClassName(classname);

// Returns the first Element that matches the selector
document.querySeletor(query);
```

Reading the DOM example

Writing to the DOM

```
// Create a new div element
let element = document.createElement("div");
// Create a new text node
let textNode = document.createTextNode("Some text");

// Adding and removing elements
element.appendChild(textNode);
element.removeChild(textNode);

// Making changes to attributes
button.setAttribute("disabled", "");
```

Changing the style of an element

An element has a "style" property which corresponds to the "style" attribute of the HTML element. This can be modified in the JavaScript

```
// Changing element.style
element.style.left = "50px"; // Note: don't forget units!

// Adding 5px to the left value
let newLeft = parseInt(element.style.left, 10) + 5 + "px";
element.style.left = newLeft;

element.style.backgroundColor = "red"; // Note: camelCase
```

Getting the style of an element

Note: `element.style.left` will only be present on an element if the `left` property was set in inline styles, or by scripting (not if it was set in CSS)

```
// Getting computed style
let computedStyle = window.getComputedStyle(element, null)
let bgColor = computedStyle.getPropertyValue("background-color")
```

Changing the classes of an element

Another way of modifying the style of the element is to change the classnames which exist on the element. This can be done using the "classList" property.

```
// Changing element.classList
element.classList.add("class");
element.classList.remove("class");
element.classList.toggle("class");
element.classList.contains("class"); // returns true if class exists on element
```

Writing to the DOM example

Scrolling

```
// Get the current scroll position of the page
console.log(window.scrollX);
console.log(window.scrollY);

// Scroll to a position on the page:
window.scrollTo({
  top: 100,
  left: 0,
  behavior: "smooth",
});
```

Scrolling to an element example

What is an Event?

An *event* is a signal that a "thing" has happened to a DOM element

This "thing" can be the element *loading*, a *click*, or a *key press*

We can use events to run JavaScript code in response to user actions

What are some examples of events?

Mouse events

- click
- dblclick
- mouseup
- mouseenter
- mousedown
- mouseleave
- mouseover

Keyboard events

- keydown
- keypress
- keyup

and more!

- error
- load
- fullscreenchange
- submit
- canplay
- animationstart
- focus

Cross Platform Compatibility

- Event handlers need to be touch-only-device friendly
- Responsive design
- Performance matters on low-end devices!

Cross Platform Compatibility

Q: What kinds of events should we be weary of for cross-platform compatability?

Cross Platform Compatibility

Q: What kinds of events should we be weary of for cross-platform compatability?

A:

1) Touch-only devices cannot hover!

Avoid using hover events to reveal core functionality

2) Fullscreen API doesn't work on iOS/Safari, check before using onfullscreenchange

Cross Platform Compatibility

Q: How would you check if a device is a touch-only device?

Cross Platform Compatibility

Q: How would you check if a device is a touch-only device?

A:

- Screen size is a simple way to check
- Add an event listener for `touchstart`
- `window.matchMedia('(hover: none)').matches`

Cross Platform Compatibility

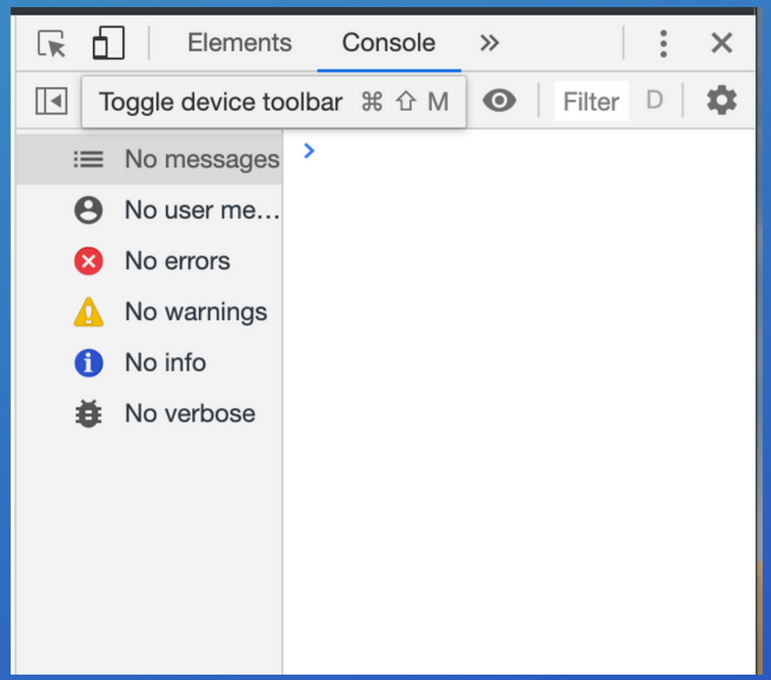
Q: How would you check if a device is a touch-only device?

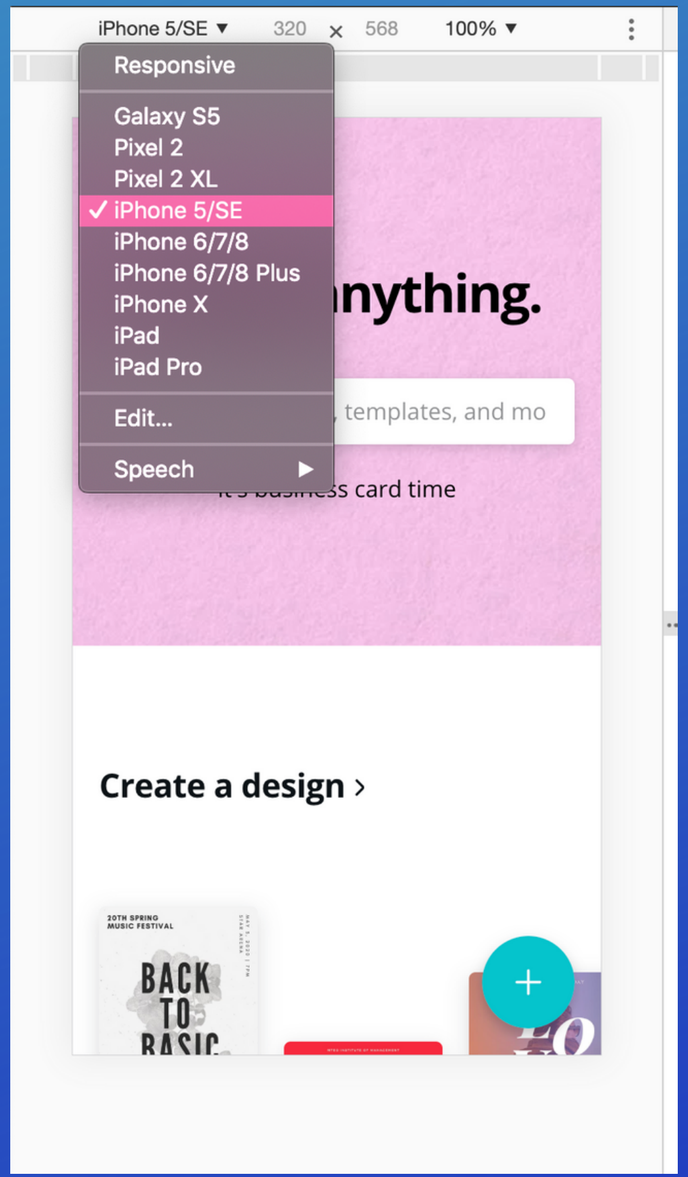
A:

- Screen size is a simple way to check
- Add an event listener for `touchstart`
- `window.matchMedia('(hover: none)').matches`

Mobile Dev Tools

Simuate testing on a mobile device:





Remote Debugging on Devices:
https://developers.google.com/web/tools/chrome-devtools/remote-debugging/

Performance

Q: What are some ways that you can make your webapp more performant?

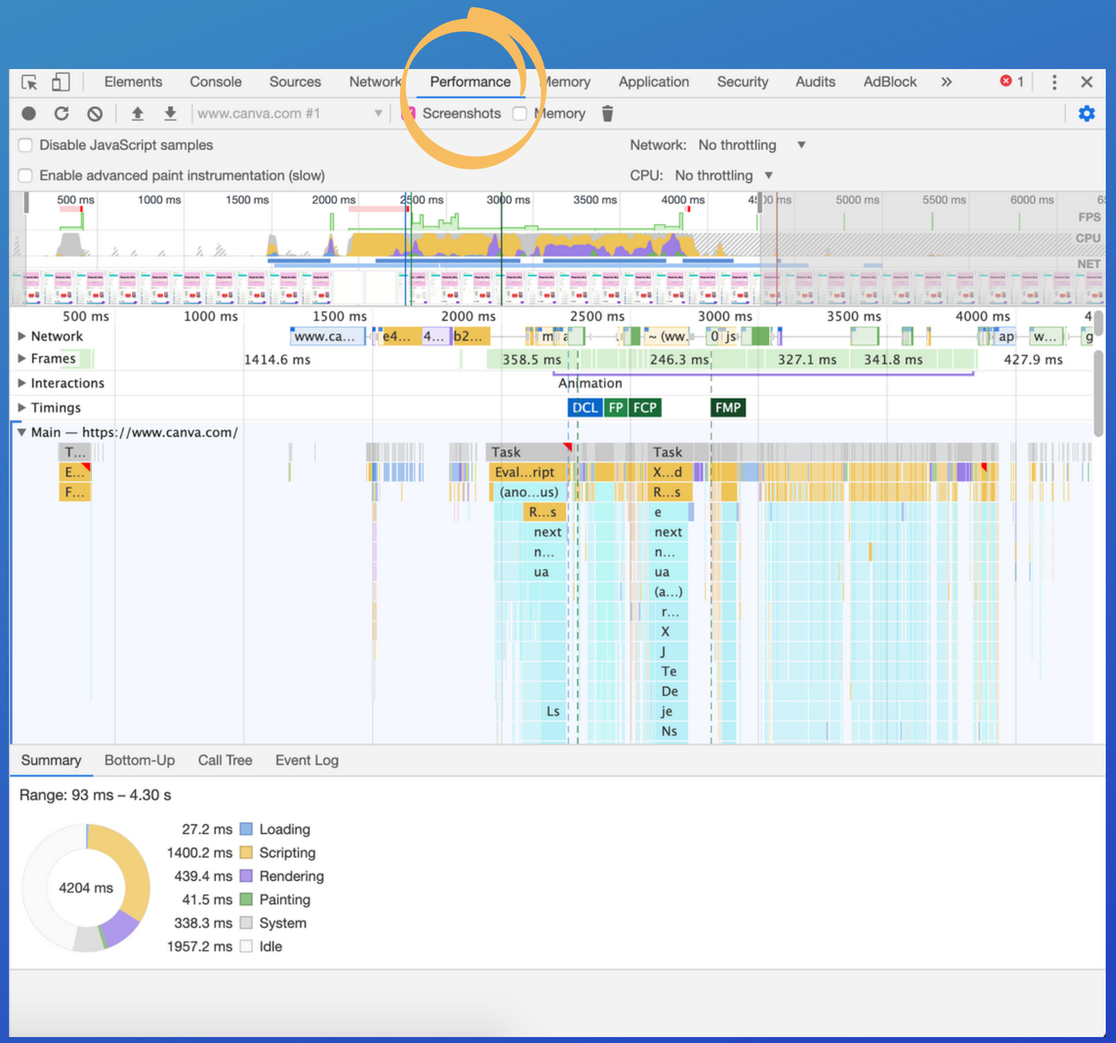
Performance

Q: What are some ways that you can make your webapp more performant?

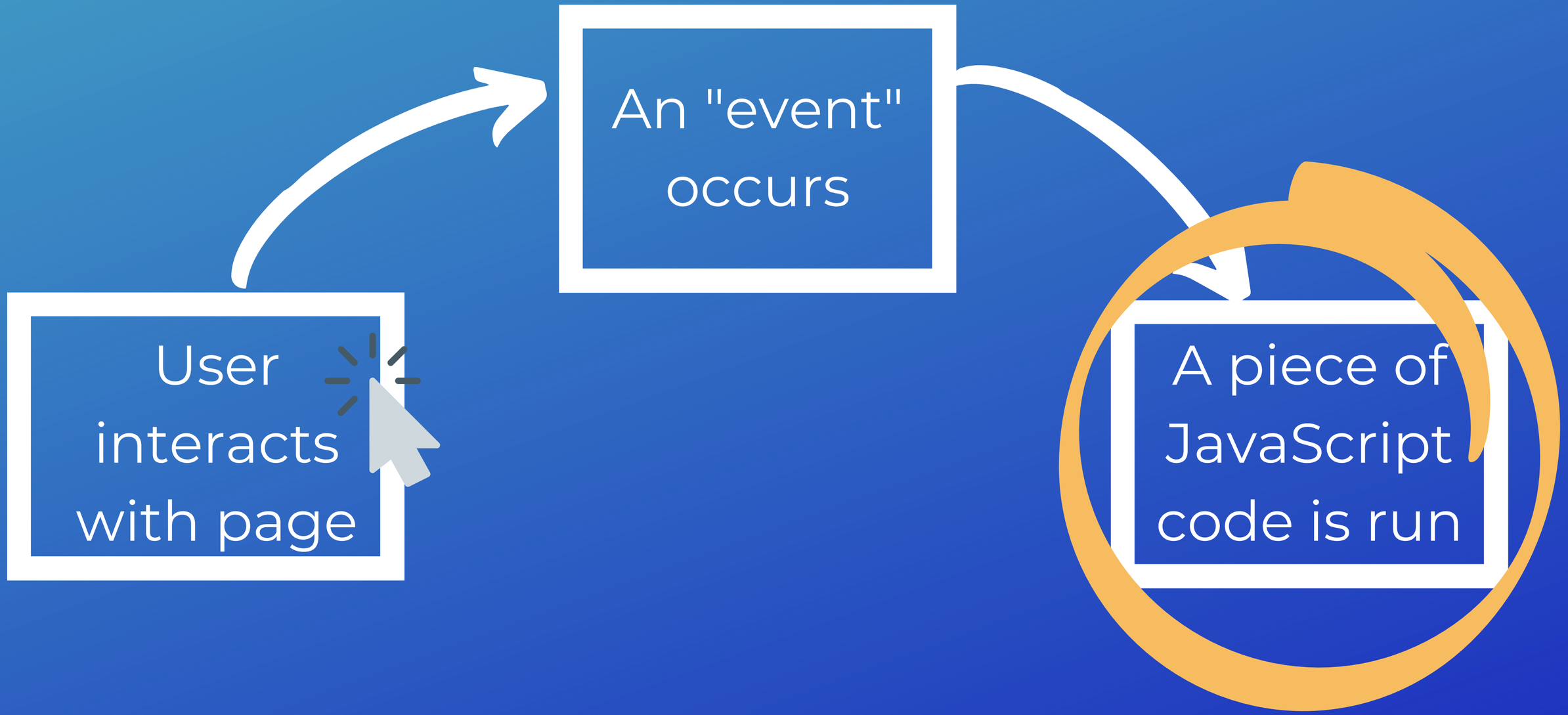
A:

- Minify your assets
- Avoid unperformant CSS
- Choose appropriate image resolutions
- Avoid unnecessary packages

Performance Dev Tools



Event Handlers



Adding Event Handlers

1) In HTML

2) DOM Property

3) addEventListener

```
<input
    value="Click me"
    onclick="alert('Clicked!')"
    type="button"
>
```

Adding Event Handlers

1) In HTML

2) DOM Property

3) addEventListener

```
let element = document.getElementById('btn');

element.onclick = () => {
  alert('Button was clicked!');
};
```

Quiz

Q: What's wrong with this code?

```
function doSomething() {
  alert('hello');
}

element.onclick = doSomething();
```

Quiz

```
function doSomething() {
  alert('hello');
}

element.onclick = doSomethirg();
```

A: by adding parentheses, we are executing doSomething rather than assigning a function to onclick

Adding Event Handlers

1) In HTML

2) DOM Property

3) addEventListener

```
// Definition:
target.addEventListener(
  type, // e.g. 'click', 'mousemove'
  listener, // the callback
  [options]
);

// Example:
let element = document.getElementById('btn');
let handler = () => {
  alert('button was clicked');
})

element.addEventListener('click', handler);
element.removeEventListener('click', handler);
```

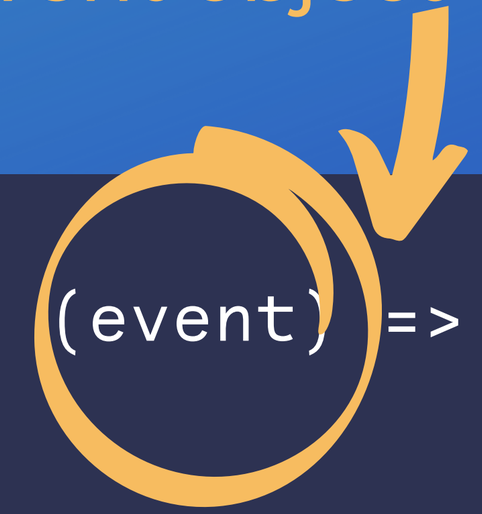
"this" binding

- `this` is all about *where* a function is invoked
- When a function is in a object, `this` refers to the object itself
- When a event is triggered, `this` is bound to the element on which the listener is attached

"this" binding example

Event Interface

The parameter to an event handler is an event object



```
document.addEventListener('mousemove', (event) => {
  console.log(event.clientX);
  console.log(event.clientY);
});
```

The event object represents the event that has taken place, and has properties describing details of the event

Event Interface Properties

Some of the properties on the event interface include:

```
event.currentTarget // current element handler is running on
event.timeStamp // time the event was created (in ms)
event.type // name of the event, e.g. 'click'
```

Different types of events have specific properties:

```
event.clientX // A MouseEvent has the X and Y coordinate
event.key // A KeyboardEvent has the keycode of the key that
was pressed
```

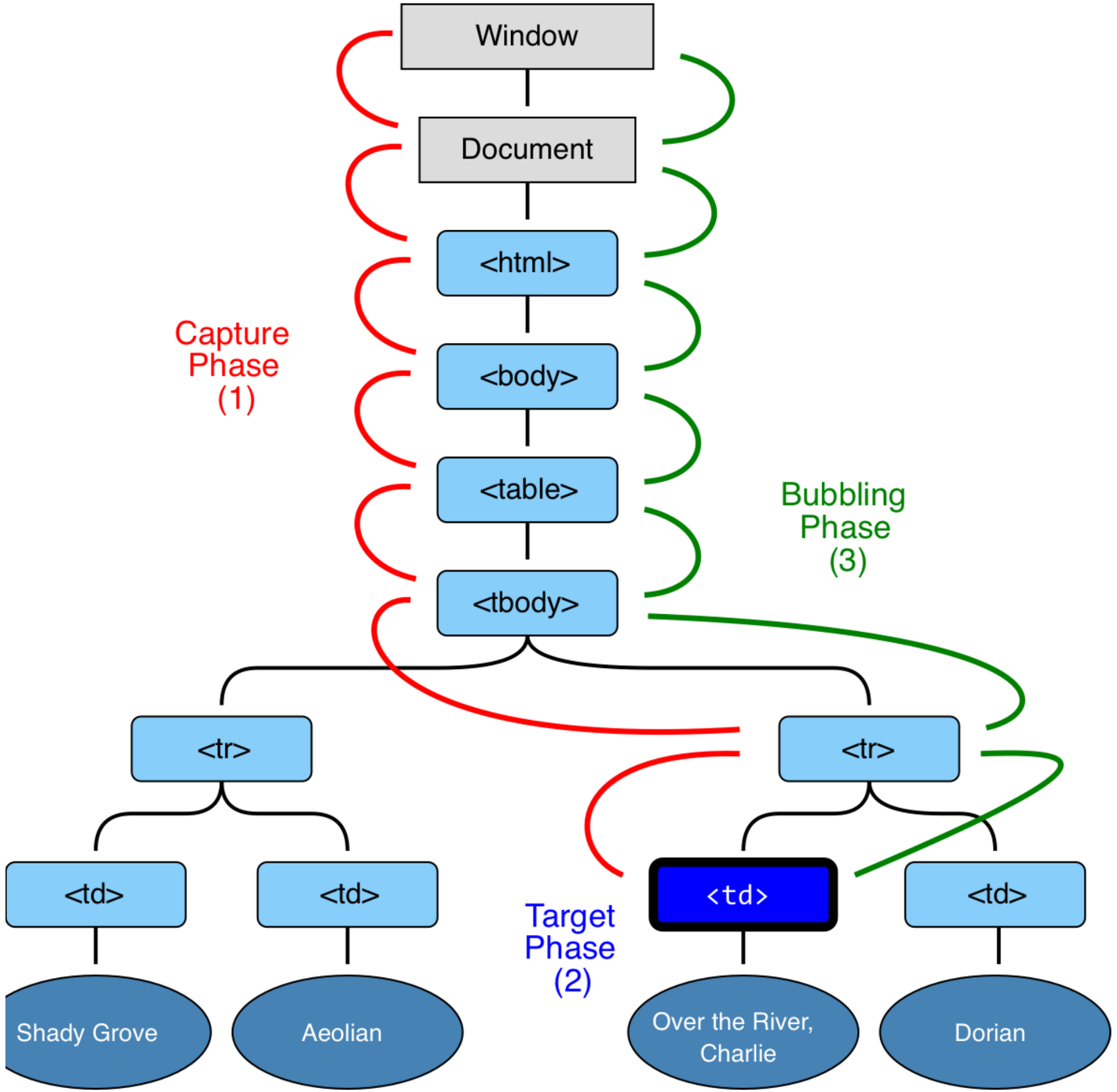
Keyboard Events Example

The Event Loop

- The event loop is a single-threaded loop which runs in the browser, and manages all events.
- When an event is triggered, the event is added to the queue.
- JavaScript uses a run-to-completion model, meaning it will not handle a new event until the current event has completed.

Event Capturing and Bubbling

Image Source: https://javascript.info/bubbling-and-capturing



Event Capturing and Bubbling Example

Prevent Default

Some types of DOM elements have default behaviour, e.g.

1. Clicking an input checkbox toggles the checkbox
2. Images have a default drag and drop behaviour to allow you to drag them into another location
3. Key presses into a text input field has the default behaviour of entering that text into the input field

To stop the default behaviour of an event, use:

```
event.preventDefault()
```

Prevent Default Example

Eyedropper Example

Canva

Want To Join Australia's Best Place to Work?

Check **canva.com/careers** for more details.

anna.a@canva.com