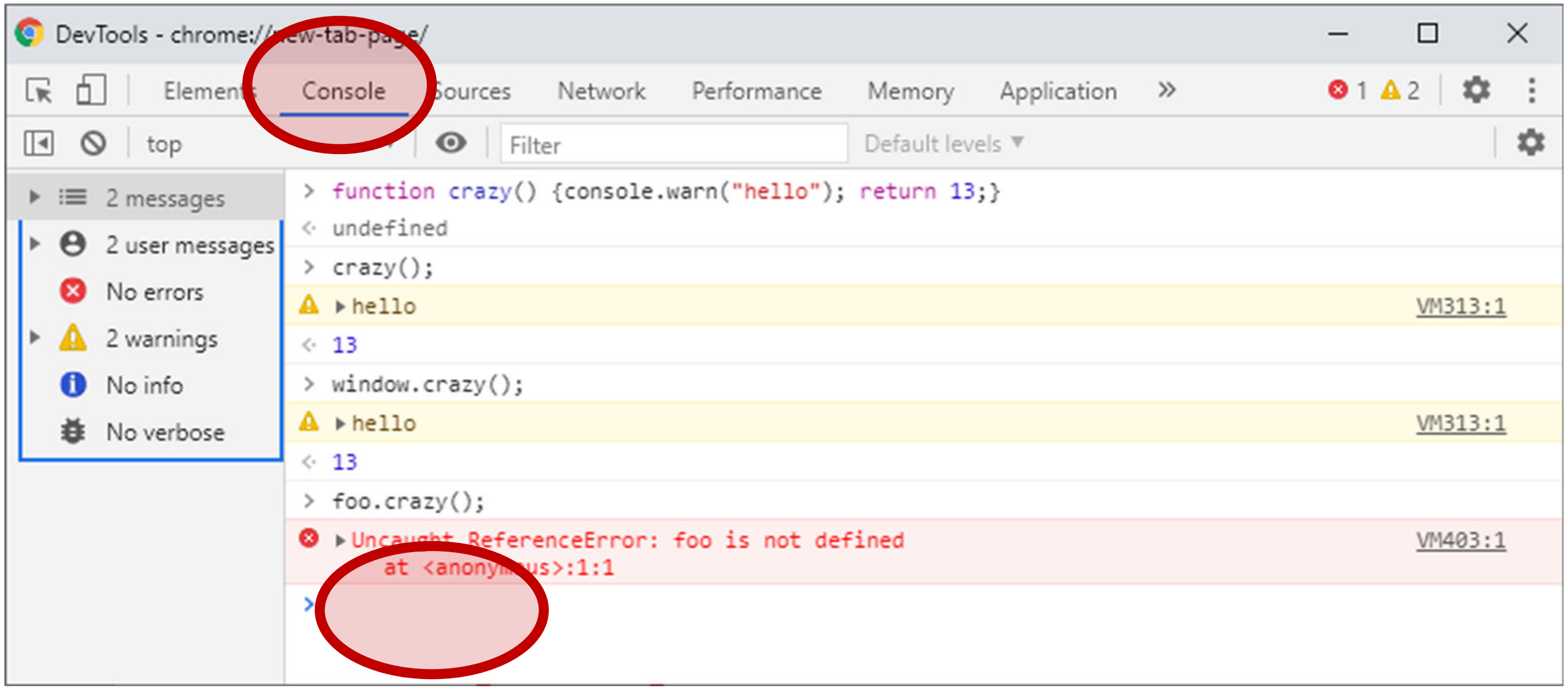
JavaScript: DOM and Events

Lecture 26

Objects are Everywhere

- Global variables in JavaScript are a lie
- Implicitly part of some *global object*, provided by execution environment
  - See Developer Tools: Console



```
> function crazy() {console.warn("hello"); return 13;}
<· undefined
> crazy();
⚠ ▸hello                                                      VM313:1
<· 13
> window.crazy();
⚠ ▸hello                                                      VM313:1
<· 13
> foo.crazy();
⊗ ▸Uncaught ReferenceError: foo is not defined                  VM403:1
      at <anonymous>:1:1
>
```

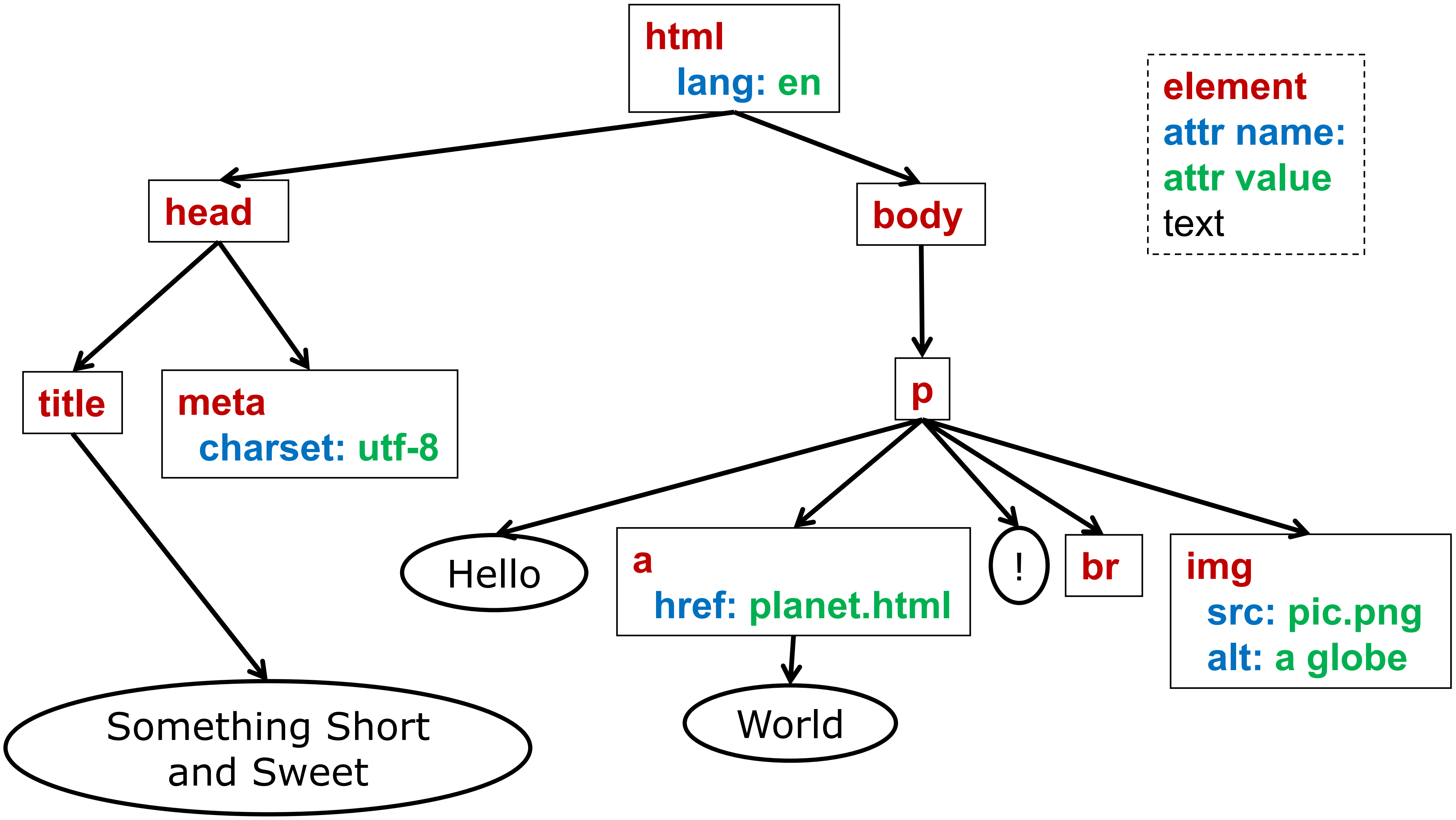
Window Object

- For JavaScript running in a browser, implicit global object is the window

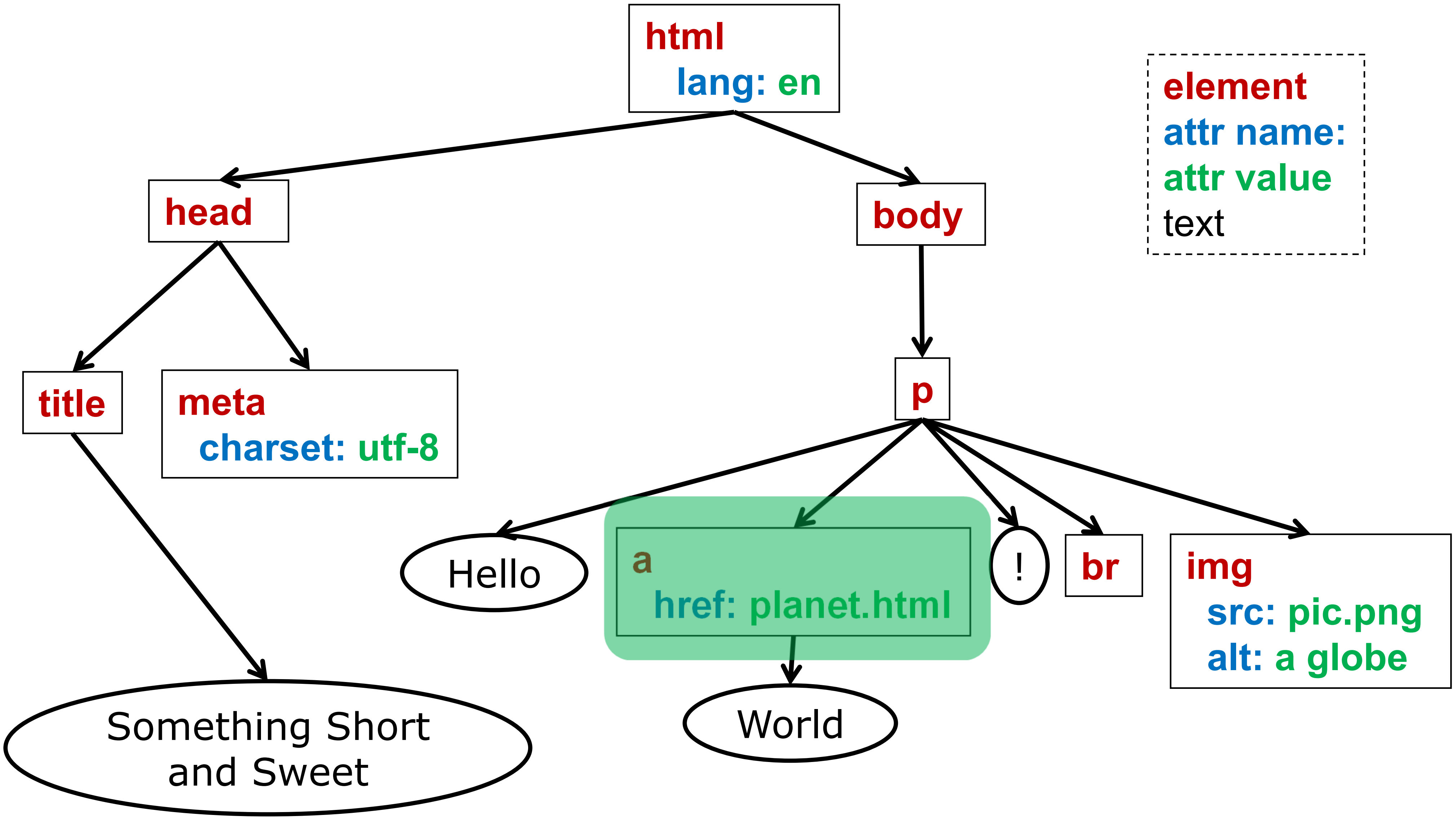
```
>> this
<- Window
```

- The global object has properties, eg
  - `location` (url of displayed document)
  - `history`
  - `innerHeight`, `innerWidth`
  - `sessionStorage`
  - `alert()`, `prompt()`
  - `document` (tree of displayed document)
- For JavaScript in a different environment (eg node.js), the global object is different

Document is a Tree



Document is a Tree



DOM: "Document Object Model"

- DOM is a language-neutral API for working with HTML (and XML) documents
  - Different programming languages have different bindings to this API
  - But all are similar to JavaScript's API
- In JavaScript, tree nodes → objects
  - A tree node (an HTML element, or text node)

    ```
    <input type="text" name="address">
    ```
  - A JavaScript object with many properties

    ```
    { tagName: "INPUT",
      type: "text",
      name: "address", /* lots more… */ }
    ```

DOM History

- Ad hoc DOM existed from the beginning of JavaScript
  - Core purpose of client-side execution: Enable user interaction with the document
  - Need a connection between programming language (JavaScript) and the document
- DOM 1 specification (W3C) in '98
  - Standardized mapping tree→objects and functions for *modifying* the tree
- DOM 2 ('00): added styles and event handling
- DOM 3 ('04): fancier tree traversal & indexing schemes
- DOM "4" ('15…):
  - Actually just a "living document"
  - Some non-backwards-compatible changes

How to Find a Node in Tree

1. Hard coding with “flat” techniques
   - Array of children

     `document.forms[0].elements[0]`
   - Downside: too brittle
   - If the document structure changes a little, everything breaks
2. Using an element's *name attribute*
   - In HTML:

     ```
     <form name="address"> …
     <input name="zip"... />  </form>
     ```
   - In JavaScript:

     `document.address.zip`
   - Downside: direct path still hard coded

How to Find a Node in Tree

3. Using an element's *id attribute*
   - In HTML

     ```
     <span id="shipping">...</span>
     ```
   - In JavaScript

     ```
     document.getElementById("shipping")
     ```
   - Downside: element must have (unique) ID
4. Using a *CSS selector*
   - Find one match or all matches

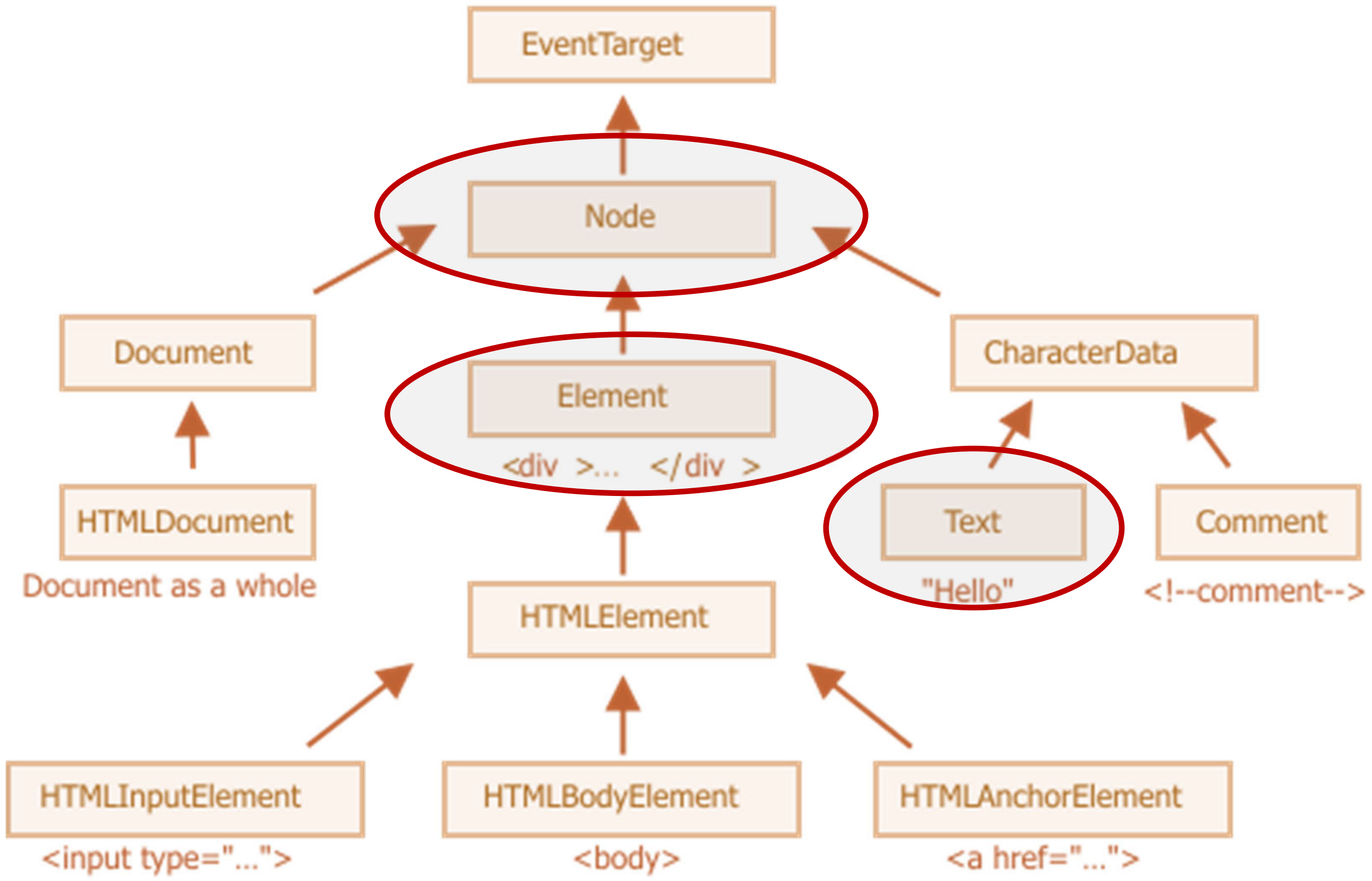
     ```
     document.querySelector("#shipping");
     document.querySelectorAll(".nav li");
     ```
   - Search a subtree

     ```
     elt.querySelector("tr"); // below elt
     ```

Node is a JavaScript Object

- Properties
  - **`parentNode`**, **`childNodes`**
  - **`firstChild`**, **`lastChild`**, **`nextSibling`**, **`previousSibling`**
  - **`textContent`**
    - Concatenation of text descendants (leaves)
    - Read/write
  - **`nodeType`**
    - Tree nodes include elements, text, comments…
  - **`nodeName`**
    - "IMG", "TABLE", "FOOTER"… , or "#text"
- Methods
  - **`appendChild(node)`**, **`removeChild(node)`**
  - **`replaceChild(new, old)`**

Inheritance: Node/Element/Text



Element (and HTMLElement)

- Properties
  - **`tagName`**
    - HTML upper case ("A"), XML lower case ("a")
  - **`id`**, **`className`**
  - **`attributes`**
  - **`style`**
    - Hyphenated property in CSS (“font-size”) becomes camelCase in JavaScript (“fontSize”)
  - **`innerHTML`**
- Methods
  - **`hasAttribute(attr)`**, **`removeAttribute(attr)`**, **`getAttribute(attr)`**, **`setAttribute(attr)`**
  - **`insertAdjacentHTML(position, html)`**

Demo: Web Console (Reading)

```
> let b = document.body;
> b.tagName;     // 'BODY'
> b.childNodes; // a NodeList
> for (let n of b.childNodes) {
    console.info(n.nodeName)
  }
> b.style; // inspect css properties


> let x = document.querySelector("footer");
> x.innerHTML;
> x.childNodes;
```

Demo: Web Console (Writing)

```
> let b = document.body;
> b.style.backgroundColor; //=> ""
> b.style.backgroundColor = "green";

> let x = document.querySelector("footer");

// bad
> x.innerHTML;
> x.innerHTML = "<h2>Hello</h2>";

//good
> let h = document.createElement("h2");
> h.className = "demo";
> h.textContent = "World";
> x.appendChild(h);
```

Interactive Documents

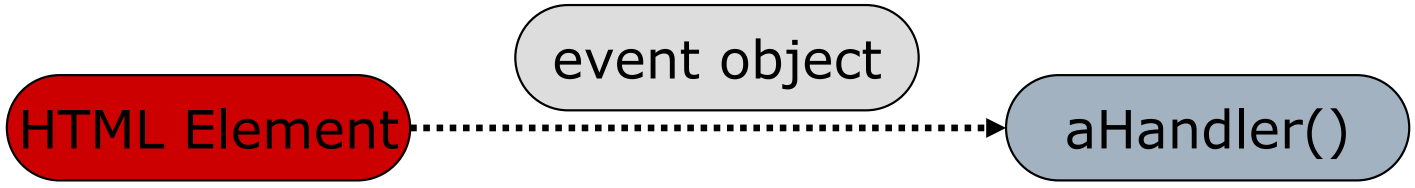
- To make a document interactive, you need:
  - Widgets (ie HTML elements)
    - Buttons, windows, menus, etc.
  - Events
    - Mouse clicked, window closed, button clicked, etc.
  - Event listeners
    - Listen (ie wait) for events to be triggered, and then perform actions to handle them

Events Drive the Flow of Control

- This style is *event driven* programming
- Event handling occurs as a loop:
  - Program is idle
  - User performs an action
    - Eg moves the mouse, clicks a button, types in a text box, selects an item from menu, …
  - This action generates an event (object)
  - That event is sent to the program, which responds
    - Code executes, could update document
  - Program returns to being idle

Handling Events Mechanism

- Three parts of the event-handling mechanism
  - *Event source*: the widget with which the user interacts
  - *Event object*: encapsulated information about the occurred event
  - *Event listener*: a function that is called when an event occurs, and responds to the event



Simple Example: Color Swaps

```
<p>This page illustrates changing colors</p>
<form>
  <p>
    <label> background:
      <input type="text" name="back" size="10"
        onchange="foo('bg', this.value)" />
    </label> <br />
    <label> foreground:
      <input type="text" name="fore" size="10"
        onchange="foo('fg', this.value)" />
    </label>
  </p>
</form>
```

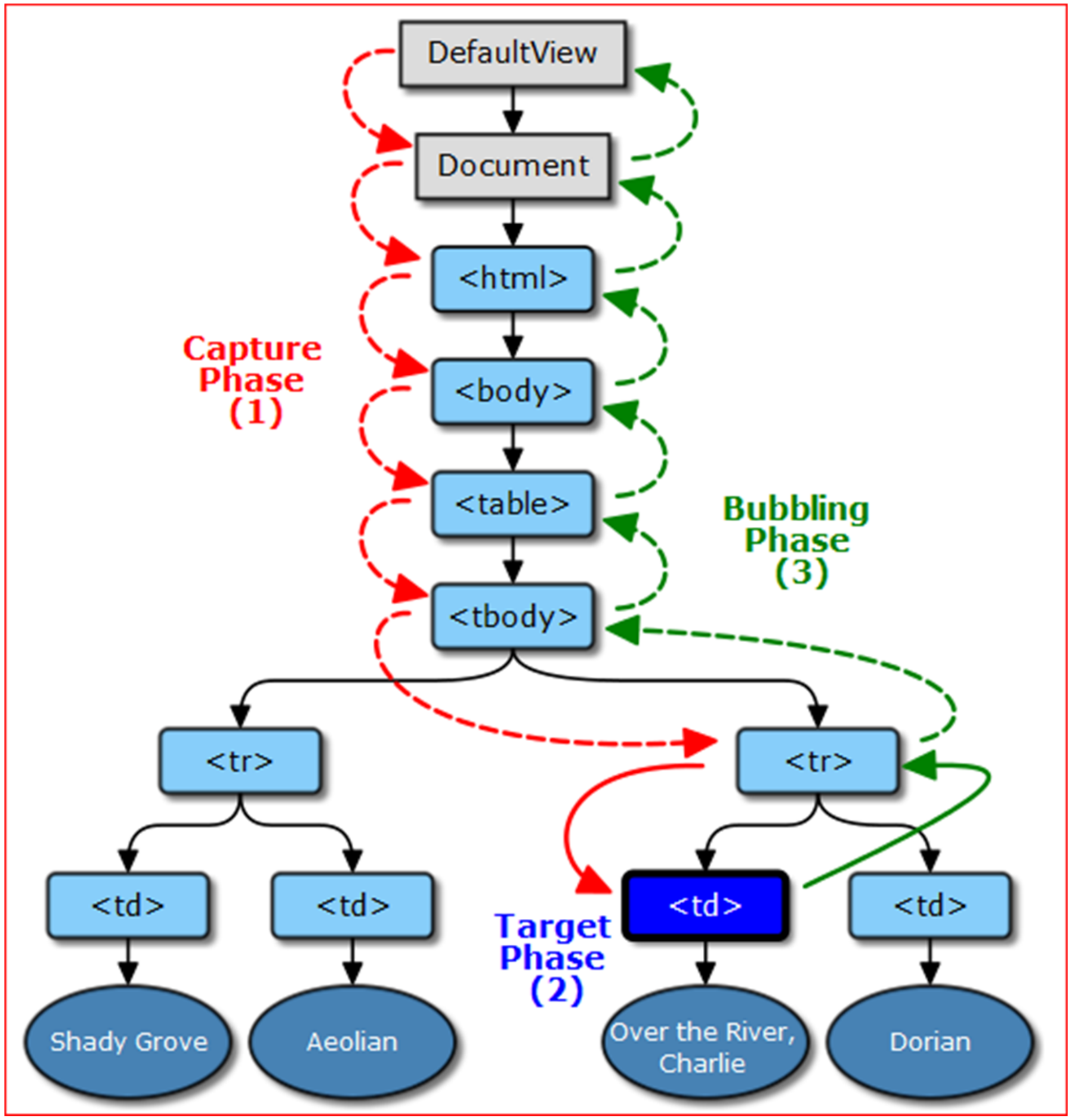
Color Swaps (JavaScript)

```
function foo(place, color) {
  if (place === "bg")
    document.body.style.backgroundColor =
        color;
  else
    document.body.style.color = color;
}
```

Event Propagation

- Elements are nested in tree
- When an event occurs, which element's handler(s) is(are) notified?
- First, *propagation path* is calculated: from root to smallest element
- Then event dispatch occurs in 3 phases
  1. Capture (going *down* the path)
  2. Target (smallest element)
  3. Bubble (going *up* the path, reverse of 1)

http://www.w3.org/TR/DOM-Level-3-Events/



Bubbling Up

- Handling is usually done in phase 2 and 3
- Example: mouse click on hyperlink
  - Handler for **`<a>`** element displays a pop-up ("Are you sure you want to leave?")
  - Once that is dismissed, event flows up to enclosing **`<p>`** element, then **`<div>`** then… *etc.* until it arrives at root element of DOM
  - This root element (*i.e.* **`window`**) has a handler that loads the new document in the current window

Programmer Tasks

- Define a handler
  - Easy, any function will do
- Register handler
  - Link (HTML) tree element with (JavaScript) function(s)
- Invoke the handler when event occurs
  - Ha! Not our job
- Get information about triggering event
  - Handler is invoked with a parameter: an event object

Registering an Event Handler

- Three techniques, ordered from:
  - Oldest (most brittle, simplest) to
  - Newest (most general)

1. Inline (link in HTML itself)

```
<a href="page.html" onclick="foo()">…
```

2. Direct property (link in JavaScript)

```
let e = … // find source element in tree
e.onclick = foo;
```

3. Chained (link in JavaScript)

```
let e = … // find source element in tree
e.addEventListener("click", foo, false);
```

Example

```
let divs =
      document.querySelectorAll("div");
for (let d of divs) {
  d.onmouseover = function() {
    this.style.backgroundColor = "red"
  }
  d.onmouseout = function() {
    this.style.backgroundColor = "blue"
  } // *this* will be the element (div)
    // that listener is registered with
}
```

Handler Registration in DOM

- Each element has a *collection* of handlers
- Add/remove handler to this collection

```
let e = … // find source element in tree
e.addEventListener("click", foo);
```

- First parameter: event name
  - Note: no "on" in event names, just "**`click`**"
- Second parameter: handler function
  - This function takes an argument: event
- Third parameter: handling phase
  - Default is **`false`** (target or bubbling phase)
  - For capture phase (unusual) use **`true`**

Example

```
let divs =
      document.querySelectorAll("div");
for (let d of divs) {
  d.addEventListener ("click",
    function(event) {
      this.act = this.act || false;
      this.act = !this.act;
      this.style.backgroundColor =
        (this.act ? "red" : "gray");
    });
}
```

Pitfall: Wrong this with =>

```
let divs =
      document.querySelectorAll("div");
for (let d of divs) {
  d.addEventListener ("click",
    (event) => { // wrong this
      this.act = this.act || false;
      this.act = !this.act;
      this.style.backgroundColor =
        (this.act ? "red" : "gray");
    });
}
```

Better: Use event Argument

```
let divs =
      document.querySelectorAll("div");
for (let d of divs) {
  d.addEventListener ("click",
    (event) => { // use param, not this
      let t = event.currentTarget;
      t.act = t.act || false;
      t.act = !t.act;
      t.style.backgroundColor =
        (t.act ? "red" : "gray");
    });
}
```

Summary

- DOM: Document Object Model
  - Programmatic way to use document tree
  - Get, create, delete, and modify nodes
- Event-driven programming
  - Source: element in HTML (a node in DOM)
  - Handler: JavaScript function
  - Registration: in-line, direct, chained
  - Event is available to handler for inspection