CS2003 W04
Web Computational Model and JS

Özgür Akgün

Slides adapted from Al Dearle and Saleem Bhatti

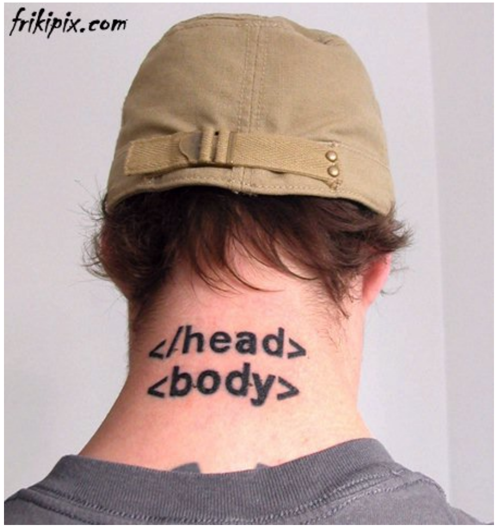
Computation

The Web in 1991

- Interaction:
  - client sends request
  - server returns page
  - web pages stored in files on server
- Information is static
  - same every time it is fetched
  - same for every client
  - once fetched, page stays the same on the client

Modern Requirements

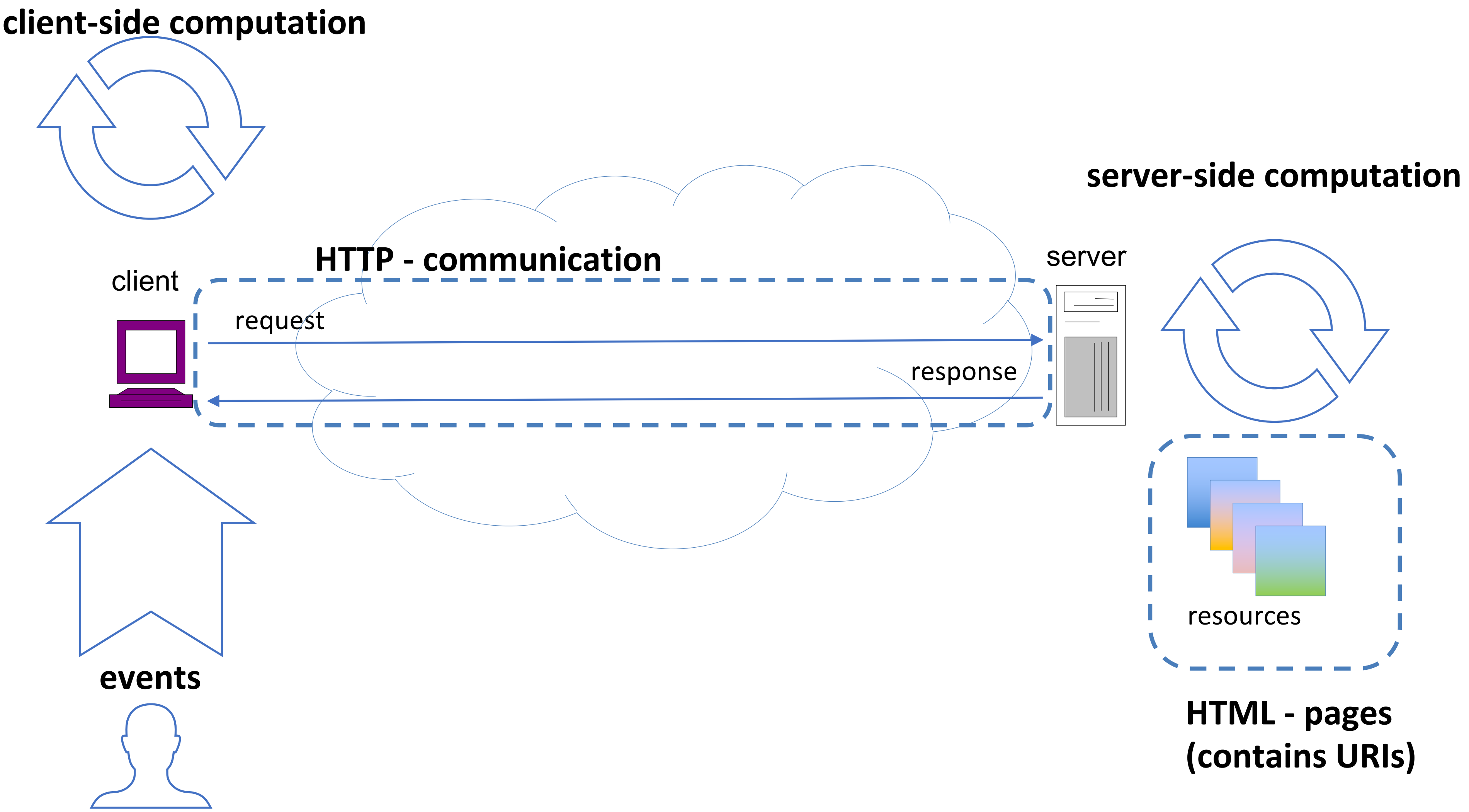
- Upload information from client to server
- Personalised pages
- Forms - update order details…
- Client-side state
  - games etc.
- Validate information being uploaded
- Fetch new information dynamically
- Include dynamic elements in pages
  - e.g. Infinite scroll, Maps, facebook etc.



What’s needed to implement this?

- Client needs to be able to reactively run code
- Need to be able to dynamically update pages - server side or client side or both!
- Server extracts information computes contents of result page rather than reading from static file
- i.e. code is invoked on server to deliver pages
- Often this is interleaved and **asynchronous** (more later)

WWW: updated model



Technologies needed

- Event Handlers - in HTML
- Handlers can call code client side uploaded from server - pretty much exclusively JavaScript these days
- Code can manipulate Web pages using the Document Object Model (**DOM**)
- Server side almost any language possible since server only needs to receive requests and send HTML pages back
- Common technologies - PHP, JavaScript, Ruby, Java, ….
- In this course we will use JavaScript on the client and on the server

Web computation / processing trade-offs

Client-side

- Actions specific to a user.
- Local processing , fast response.
- Takes load off server.
- Reduces network load.
- May incur local storage and processing overhead.

Server-side

- Common content generated on demand.
- Benefit for many users accessing same content.
- Higher overhead at server for each page/document to be dynamically generated.

Javascript

Javascript in three stages

- Introduction to Javascript:
  - basic language features (much detail not covered).
- The Document Object Model (DOM) API:
  - client-side programming with HTML5 and CSS.
  - improving client-side functionality.
- Creating full applications – node.js (node):
  - server-side programming.
  - asynchronous, event-driven API.
  - (later in the course).

Javascript resources

- Official ECMAScript 2015:
  - https://www.ecma-international.org/ecma-262/6.0/
- Mozilla Javascript Developer docs:
  - https://developer.mozilla.org/bm/docs/Web/JavaScript
- Javascript Tutorial (focus on client-side):
  - https://javascript.info
- Node (node.js):
  - https://nodejs.org/en/

Javascript syntax basics

- Case sensitive:
  - A1 and a1 are different.
- Whitespace between tokens are ignored.
- Semi-colons are optional at the end if lines:
  - used to separate statements.

- Comments (C/C++ style):
  - `// at the end of lines`
  - `/* in code */`
- Names (Identifiers):
  - Examples:

    ```
    x
    my_value
    attr_val_1
    _value
    $num
    ```
  - cannot be the same as reserved words

Javascript basics types

- **Dynamic typing**:
  - type is assigned when value is assigned
  - but typing is strong
- Primitive types:
  - numbers
  - strings
  - booleans
- Trivial types:
  - null
  - undefined

- Composite types:
  - – object
  - – Array
- Functions:
  - – more than one way of defining a function.
  - – functions are first-class values (full support).
  - – (as in functional languages)

Numbers

- All are 8-byte floating point representation:
  - no separate integer values
- Literal values as you might expect, e.g.:
  - only decimal: 10, 42
  - `parseInt(s, b)` converts string s in base b to decimal, e.g. `parseInt(“0x2a”,16)` returns decimal 42

- For self-study:
  - the normal mathematical operators
  - Math object
  - `toString()` method

Strings

- Delimited by “ ”
- Can use escapes for special characters, e.g.:
    - \n
    - \t
- Strings are easily used in Javascript:
    - operators

- For self-study:
    - string operators, e.g. “+” for concatenation
    - string comparison
    - string methods (length, substring search/match, case changes, etc.)

Booleans

- Reserved words:
  - true
  - false
- Comparisons in Javascript result in boolean values:
  - `a == b`
  - `a === b`
  - the second one of these checks the type also: **strict** equality.

- Ternary operator uses boolean conditional:

```
a == 1.0 ? b = 4 : b = 2
```

  - if a == 1.0 then b = 4
  - else b = 2

Trivial types

- null
  - No value, reserved word.
  - can be assigned to a variable of any type.
- undefined
  - usually arises from an error.
  - e.g. a declared variable with no value assigned.
  - e.g. an unknown name (could be due to a typo)

Declarations and assignments

- keyword: **`let`**
  to introduce a declaration
- Keyword: **`var`**
  to introduce a declaration
- Keyword: **`const`**
  to introduce a declaration
- Examples:

```
let x
const myString = “str”
var myVal_1 = 42;
watch_out = 7
```

- let/var new 2015
- Let - like Java decl (block scope):
- var - has function or global scope
- If you don’t use let/var/const a name is **globally scoped**!!!
- also Implicit globals

Declarations

```
function foo() {
    var variable1, variable2;

    variable1 = 5;
    varaible2 = 6;
    return variable1 + variable2;
}
```

Declarations

- Use **const** if you can
- Otherwise use **let** if you can
- Otherwise use **var** if you can
- **Try not to <u>not</u> use any of above! i.e. implicit decls**

BE VERY CAREFUL

- The purpose of `"use strict"` is to indicate that the code should be executed in "strict mode"
- With strict mode, you can not, for example, use undeclared variables

Dynamic typing

- Types are implied by assignment of a value
- Types can be changed by assigning different values
- Examples:

```
let a_string = “str”;
let a_num = 42;
a_string = 42;
a_num = “str”;
```

- Comparison operators:

  == converts type

  === **strict**: compares type

- Examples:

```
let a = “7”;
let b = 7;
a == b; // true
a === b; // false
```

Iteration

- For self-study
- for:
  ```
  for(let x = 0; x < x_max; ++x) { /* statements */ }
  ```
- for … in:
  ```
  let o = {a:1, b:2, c:3};
  for(let e in o) { /* statements */ }
  ```
- while:
  ```
  while (x != true) { /* statements */ }
  ```
- do … while:
  ```
  do { /* statements */ } while (x == true);
  ```

Functions

- function

```
function myFunction(param1, param2) {
  /* statements */
  return rValue;
}
```

- DOM functions can be assigned to event handlers that relate to the web page, e.g. onclick:

```
function myButtonFunction() { /* ... */ }
button.onclick = myButtonFunction;
```

Server side JS

NODE.JS

- Node.js is an **asynchronous event driven** JavaScript runtime
- Node is designed to build **scalable** network applications.
- Upon each connection the **callback** is fired, but if there is no work to be done, Node will sleep.
- Node presents an event loop as a runtime construct instead of as a library.

Running node on School clients

```
$ node <filename>
```

Example Server

```
const http = require('http');

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((request, response) => {
  response.statusCode = 200;
  response.setHeader('Content-Type', 'text/plain');
  response.end(‘Hello World\n');
});

server.listen(port, hostname, () => {
  console.log(`Server running at
http://${hostname}:${port}/`);
});
```

https://nodejs.org/en/about/

Examples: web/node/web2/serverside.js

Node.js - server

```
function generateBody() {
        var strVar="";
        strVar += "<body>";
        strVar += "<h1>Header<\/h1>";
        strVar += "<\/body>";
        return strVar;
}

function generateHTML() {
        var strVar="";
        strVar += "<!DOCTYPE html>";
        strVar += "<html>";
        strVar += generateBody();
        strVar += "<\/html>";
        return strVar;
}
```

```
function handleGet( req, res ) {
        console.log('get: ' + req.url);
        res.writeHead( 200,
{'ContentType': ‘text/html'} );
        res.write( generateHTML() );
        res.end();
}

const server = http.createServer(
   function (req, res) {
        if(req.method == "GET"){
           handleGet( req, res );
        } else if(req.method == 'POST'){
           handlePost( req, res );
        }
   }
);
server.listen(10111, '127.0.0.1');
```

Simple Javascript examples

- In web/node/web2/node_01:

  hello_world.js

  hello_world.html*

  hello_world_html.js

  variables.js

  output.js

  loops.js

  conditional.js

  functions.js

  server_info.js

  web_serve.js

These can all be run on the command line, e.g.:

```
    $ node script_name.js
```

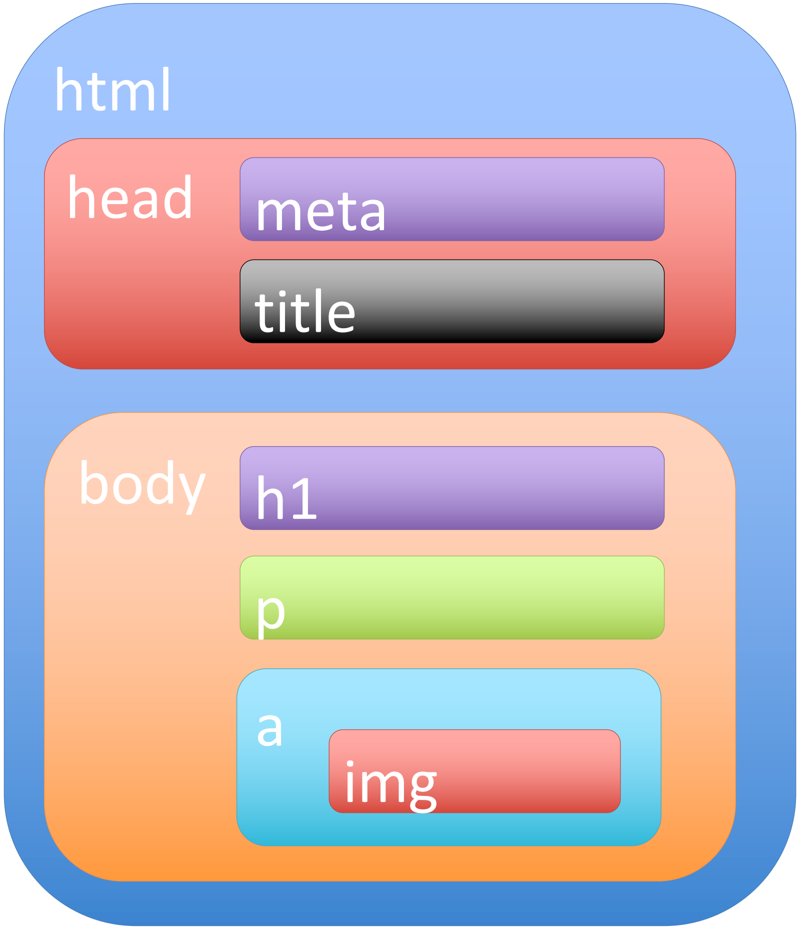
The HTML file, marked with '*' will not work with node, but will work remotely from a server with a web client, e.g. browser or curl.

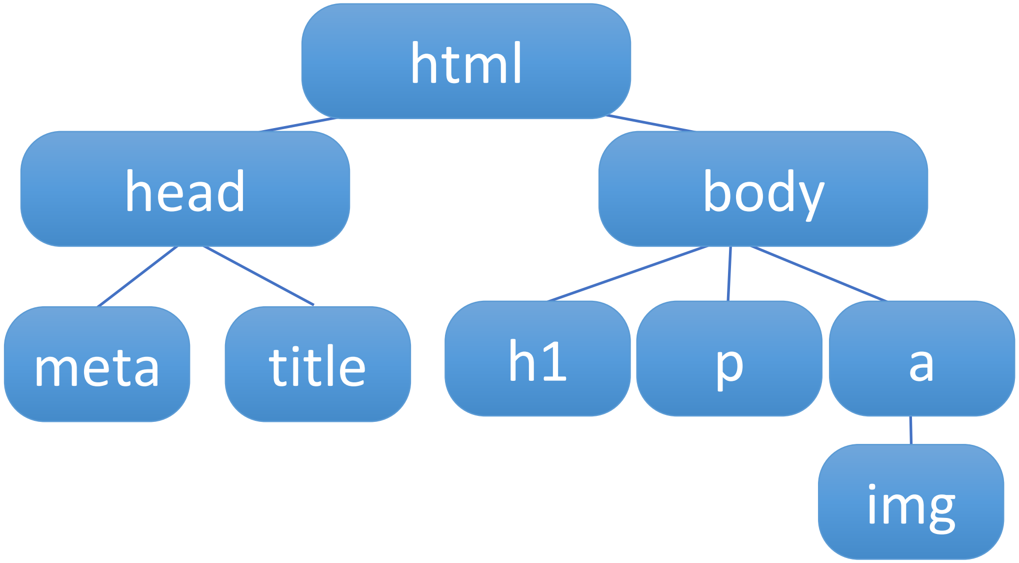
The DOM

Web page structure

```
<!DOCTYPE html>
  <html>

  <head>
    <meta author=“The Author” />
    <title>Sample Page</title>
  </head>

  <body>

    <h1>Sample Page</h1>

    <p>A very small example.</p>

    <a href=https://www.cs.st-andrews.ac.uk/><img src="photo.jpg"></a>

  </body>

</html>
```

Web page – structure





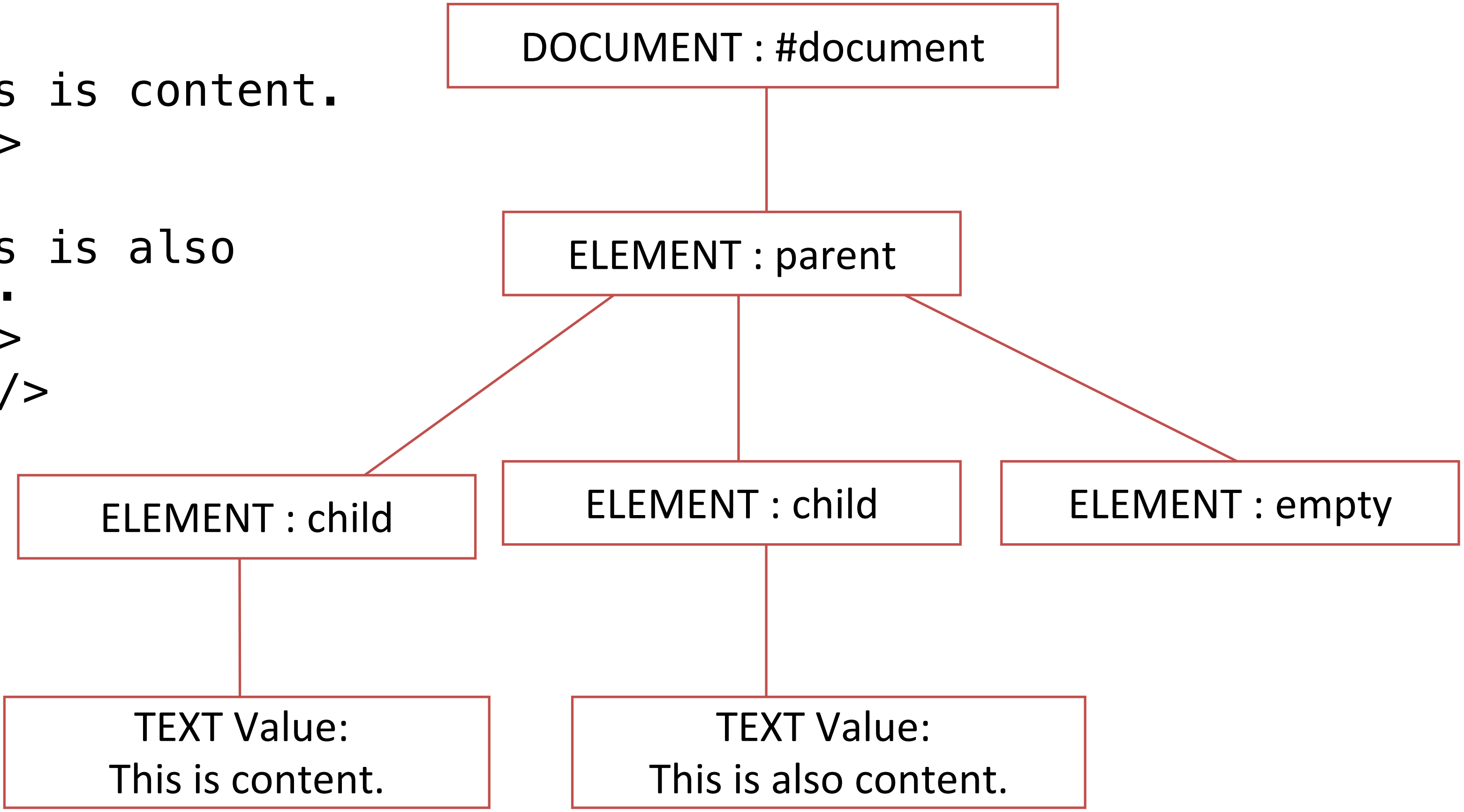
Document Object Model (DOM)

Document Object Model (DOM)

- Documents have the logical structure of a tree.
- Nodes of the tree represent different types of content in a document:
  - as objects with identity, attributes and methods.
- As an object model, the DOM identifies:
  - the interfaces and objects used to represent and manipulate a document.
  - the semantics of these interfaces and objects.
  - the relationships among these interfaces and objects.

DOM tree

```
<parent>
  <child>
      This is content.
  </child>
  <child>
      This is also 
  content.
  </child>
  <empty />
</parent>
```



DOCUMENT : #document

ELEMENT : parent

ELEMENT : child

ELEMENT : child

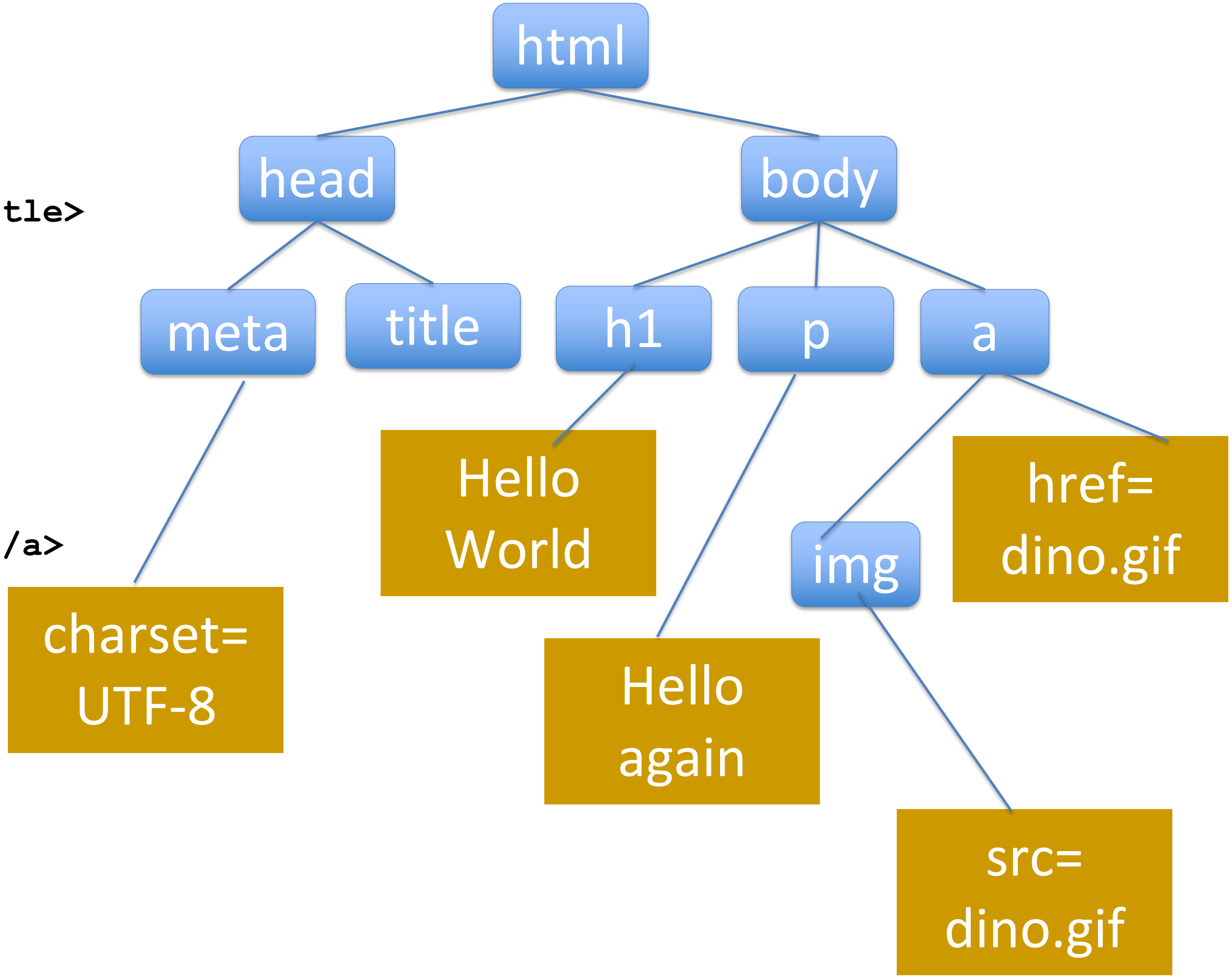
ELEMENT : empty

TEXT Value:
This is content.

TEXT Value:
This is also content.

DOM– tree / hierarchy of (elements) tags

```
<!DOCTYPE html>
<html>

<head>
<meta charset="UTF-8">
<title>Hello World!</title>
</head>

<body>

<h1>Hello World!</h1>
<p>Hello again</p>
<a href=“dino.gif”>
<img src=“dino.gif” /></a>

</body>

</html>
```



Javascript and DOM

- **`document`**
  - **the HTML currently displayed**
- `window`
  - the OS window hosting the page
- `navigator`
  - the browser application in use
- `screen`
  - the physical monitor in use

- These are all objects accessible via the **Javascript DOM API.**
- The most commonly used object is:
  - `document`
- The other environment information could be used for adjusting presentation or layout.
- Often just use:

  document.getElementById()

  to find objects

JS in the browser & DOM

Event-based API

- In the DOM, executions of Javascript code are bound to **events** that occur, e.g.:
  - a page starts loading.
  - a user clicks on a link.
  - a user hovers their mouse over some content.
  - a form is submitted.
- Mozilla Developer – Introduction to events:

  https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/Events

DOM JavaScript example - DOM1.html

```
<!DOCTYPE html>
<html>
<head>
  <title>DOM Example</title>
</head>
<body>
  <h2>Finding HTML Elements Using document.title</h2>
  <p id="demo"></p>
<script>
document.getElementById("demo").innerHTML =
    "The title of this document is: " + document.title;
</script>
</body>
</html>
```

In web/nginx/web2/DOM1.html

DOM JavaScript example - DOM2.html

```
<!DOCTYPE html>
<html>
<body>
<p>Enter names in the fields, then click "Submit" to submit the form:</p>
<form id="my_form" action="/action_page.php">
  First name: <input type="text" name="fname"><br>
  Last name: <input type="text" name="lname"><br><br>
  <input type="button" onclick="myFunction()" value="Submit">
</form>
<script>
function myFunction() {
  document.getElementById("my_form").submit();
}
</script>
</body>
</html>
```

In web/nginx/web2/DOM2.html

DOM JavaScript example - DOM3.html

```
<!DOCTYPE html>
<html>
<body>
<div id="text1" style="display: block">
  Here is some text that is visible
</div>
<div id="text2" style="display: none">
  Here is some text that was hidden!
</div>
<input type="button" value="Click me" onclick="showText()">
<script>
    function showText() {
     let text1 = document.getElementById("text1");
     let text2 = document.getElementById("text2")
     text1.style.display = "none";
     text2.style.display = "block";
  }
</script>
</body>
</html>
```

In web/nginx/web2/DOM3.html

Using Javascript with HTML

- Best to include Javascript at the bottom of the page:
  - just before </body>
  - allows page to load and as much of the DOM tree to be created.
- Care must be taken where the JavaScript code is placed
  - Loading via a slow link.
  - Javascript code can be cached at the client-side.
- Javascript is downloaded to the client:
  - it is visible to the user!
- Some users may disable Javascript (security).

When are scripts executed?

- Scripts are executed in order of appearance during the browsers HTML parsing process
- If a script is in a <HEAD> … </HEAD> part of a Web page, none of the <BODY> … </BODY> will have been defined
- Consequently none of the JavaScript objects that represent the body will not have been created
- You must think about pages being loaded via a slow link
- One way to ensure this is to define all JavaScript elements in the <HEAD> section since this is always completely processed before the <BODY>

Useful documentation pages

- Main page for Web API (API is huge): https://developer.mozilla.org/en-US/docs/Web/API
- Document: https://developer.mozilla.org/en-US/docs/Web/API/Document
- Node: https://developer.mozilla.org/en-US/docs/Web/API/Node
- JS Tute:

https://www.w3schools.com/js/