Browser Performance
Week 4 Lecture

Outline

- **Review of HTML, CSS and JavaScript**
  - **Variable Scope**
  - **Passing function parameter**
  - **Style inheritance**
- **Browser Rendering Process**
  - **Critical Rendering Path**
  - **DOM and CSSOM**
  - **Render Path Analysis**

Week 3 solution

```
window.onload = function(){
    var mainForm = document.getElementById("mainForm");

    //all inputs with the class required are looped through
    var requiredInputs = document.querySelectorAll(".required");
    for (var i=0; i < requiredInputs.length; i++){
        requiredInputs[i].onfocus = function(){
            this.style.fontWeight = "bold"
            this.style.backgroundColor = "green";
        }
        requiredInputs[i].onblur = function(){
            this.style.fontWeight = "normal";
            this.style.backgroundColor = "#FFFFFF";
        }
    }
    //on submitting the form, "empty" checks are performed on required inputs.
    mainForm.onsubmit = function(e){
          //var requiredInputs = document.querySelectorAll(".required");
          for (var i=0; i < requiredInputs.length; i++){
                if( isBlank(requiredInputs[i]) ){
                    e.preventDefault();
                    makeRed(requiredInputs[i]);
                }
                else{
                makeClean(requiredInputs[i]);
            }
        }
    }
}
```

Week 3 solution (cont'd)

```
function isBlank(inputField){
    if(inputField.type=="checkbox"){
    if(inputField.checked)
        return false;
    return true;
    }
    if (inputField.value==""){
    return true;
    }
    return false;
}

//function to highlight an error through colour by adding css attributes to the div passed in
function makeRed(inputDiv){
    inputDiv.style.backgroundColor="red"
    inputDiv.parentNode.style.backgroundColor="red";

}
//remove all error styles from the div passed in
function makeClean(inputDiv){
    inputDiv.style.backgroundColor="white"
    inputDiv.parentNode.style.backgroundColor="#FFFFFF";

}
```

JavaScript Review

- Variable Scope
- Two ways to pass arguments to functions (or methods)
  - pass-by-value
  - pass-by-reference
- Pass-by-value
  - a *copy* of the argument's value is made and is passed to the called function
  - In JavaScript, numbers, boolean values and strings are passed to functions by value.
- Pass-by-reference
  - The caller gives the called function direct access to the caller's data and allows it to modify the data if it so chooses
  - Can improve performance because it can eliminate the overhead of copying large amounts of data, but it can weaken security because the called function can access the caller's data
  - All objects are passed to functions by reference

Variables and Closure

```
window.onload = function(){
    var mainForm = document.getElementById("mainForm");

    //all inputs with the class required are looped through
    var requiredInputs = document.querySelectorAll(".required");
    for (var i=0; i < requiredInputs.length; i++){
        requiredInputs[i].onfocus = function(){
            this.style.fontWeight = "bold"
            this.style.backgroundColor = "green";
        }
        requiredInputs[i].onblur = function(){
            this.style.fontWeight = "normal";
            this.style.backgroundColor = "#FFFFFF";
        }
    }
    //on submitting the form, "empty" checks are performed on required inputs.
    mainForm.onsubmit = function(e){
          for (var i=0; i < requiredInputs.length; i++){
             if( isBlank(requiredInputs[i]) ){
                    e.preventDefault();
                    makeRed(requiredInputs[i]);
             }
             else{
                makeClean(requiredInputs[i]);
             }
        }
    }
}
```

A closure is a function having access to the parent scope, even after the parent function has closed.

Variables and Closure

Choose country ▾

Paused in debugger

City

ative Commons Types

Attribution

Noncommercial

No Derivative Works

Share Alike

I accept the software license

Rate this photo:

Date Taken:

dd/mm/yyyy

Color Collection:

Time Taken:

1117px × 2

Elements Console Sources Network Timeline Profiles Application Security Audits

Sources Content scripts

- top
  - file://
    - C:/Users/yzho8449/course/com
      - week3.html
      - week3.js
      - week3.css

week3.js × week3.css

```
36            this.style.fontWeight = "bold"
37            this.style.backgroundColor = "green";
38        }
39        requiredInputs[i].onblur = function(){
40            this.style.fontWeight = "normal";
41            this.style.backgroundColor = "#FFFFFF";
42        }
43    }
44
45    //on submitting the form, "empty" checks are performed on
46    mainForm.onsubmit = function(e){   e = Event {isTrusted: tr
47    //var requiredInputs = document.querySelectorAll(".require
48    for (var i=0; i < requiredInputs.length; i++){
49        if( isBlank(requiredInputs[i]) ){
50            e.preventDefault();
51            makeRed(requiredInputs[i]);
52
53
```

{} Line 48, Column 13

Async

Paused on breakpoint

Watch

Call Stack

mainForm.onsubmit week3.js:48

Scope

Local

e: Event

i: undefined

this: form#mainForm

Closure (window.onload)

requiredInputs: NodeList[2]

Global Wind

Infinity: Infinity

Pass by Reference

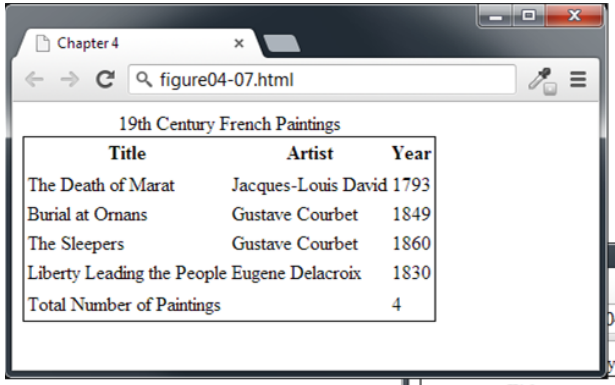
```
mainForm.onsubmit = function(e){
        for (var i=0; i < requiredInputs.length; i++){
            if( isBlank(requiredInputs[i]) ){
                e.preventDefault();
                makeRed(requiredInputs[i]);
            }
            else{
               makeClean(requiredInputs[i]);
            }
        }
    }
```

```
function makeRed(inputDiv){
    inputDiv.style.backgroundColor="red"
    inputDiv.parentNode.style.backgroundColor="red";

}
//remove all error styles from the div passed in
function makeClean(inputDiv){
    inputDiv.style.backgroundColor="white"
    inputDiv.parentNode.style.backgroundColor="#FFFFFF";

}
```

Style and Inheritance

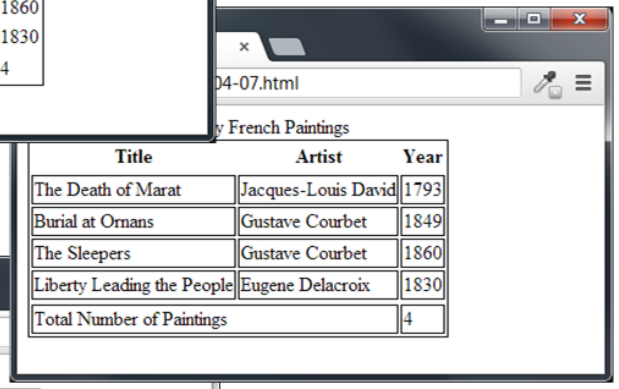
- One of the “Cascading” principle is inheritance
  - Not all properties are inherited by default
  - Default inherited:
    - Text properties such as font, color
  - Default not inherited:
    - Border, sizing, layout, background
- Background-color has a default transparent value
  - Looks like children inherit parent background color sometimes
  - Safer to set both

Other example of style inheritance

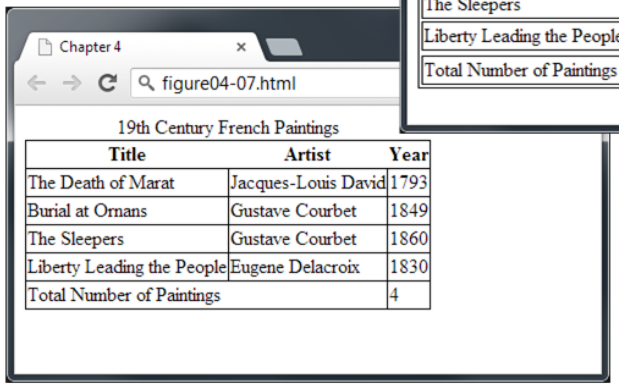



```
table {
   border: solid 1pt black;
}
```

Not inherited




```
table {
   border: solid 1pt black;
}
td {
   border: solid 1pt black;
}
```




```
table {
   border: solid 1pt black;
   border-collapse: collapse;
}
td {
   border: solid 1pt black;
}
```

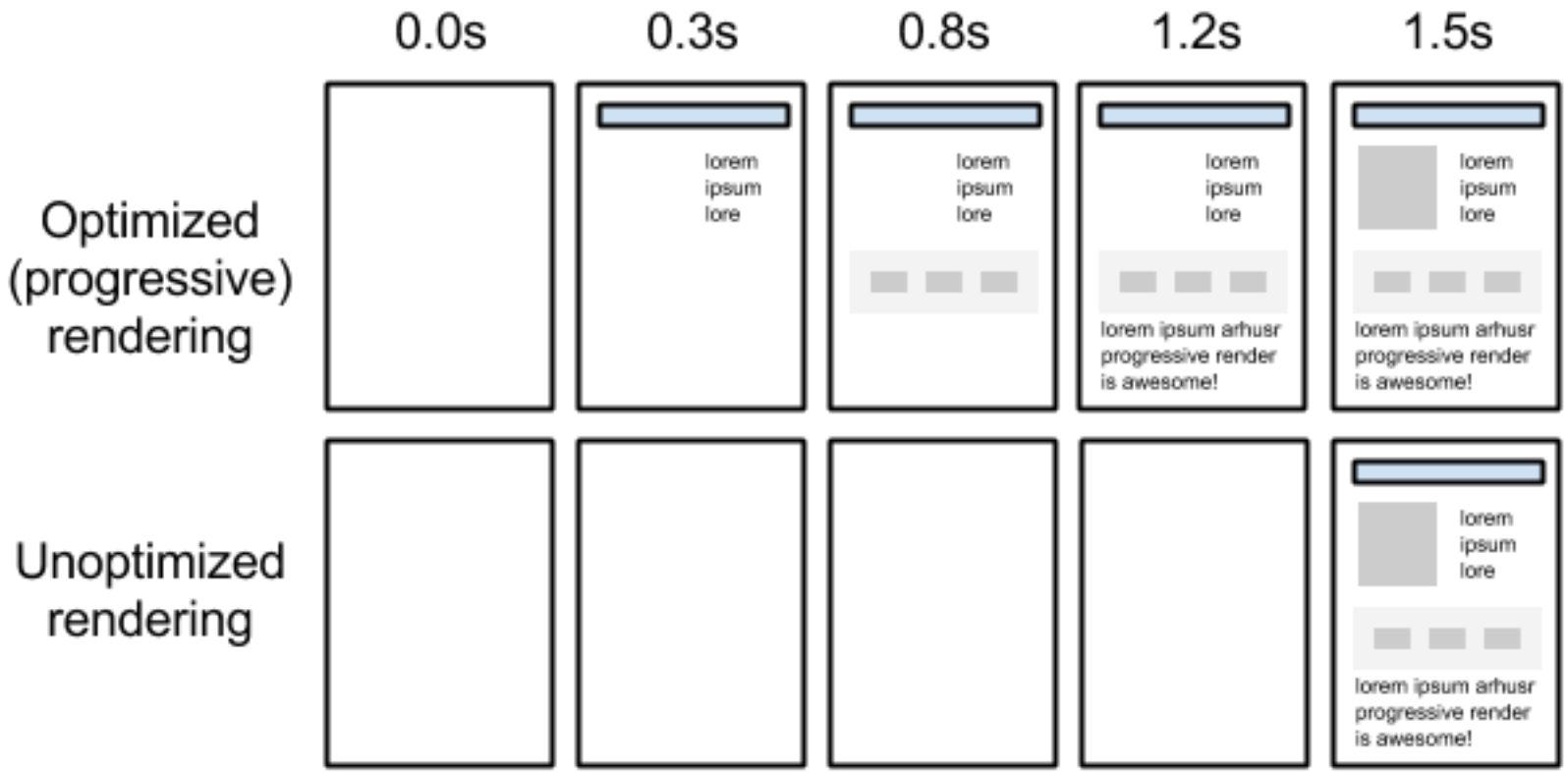
Inherited

Outline

- Review of HTML, CSS and JavaScript
  - Variable Scope
  - Passing function parameter
  - Style inheritance
- Browser Rendering Process
  - Critical Rendering Path
  - DOM and CSSOM
  - Render Path Analysis

Critical Rendering Path

- The actual steps browsers take to receive/parse/display data from web server is called <u>critical rendering path</u>



https://developers.google.com/web/fundamentals/performance/critical-rendering-path/?hl=en

Overall Rendering Process

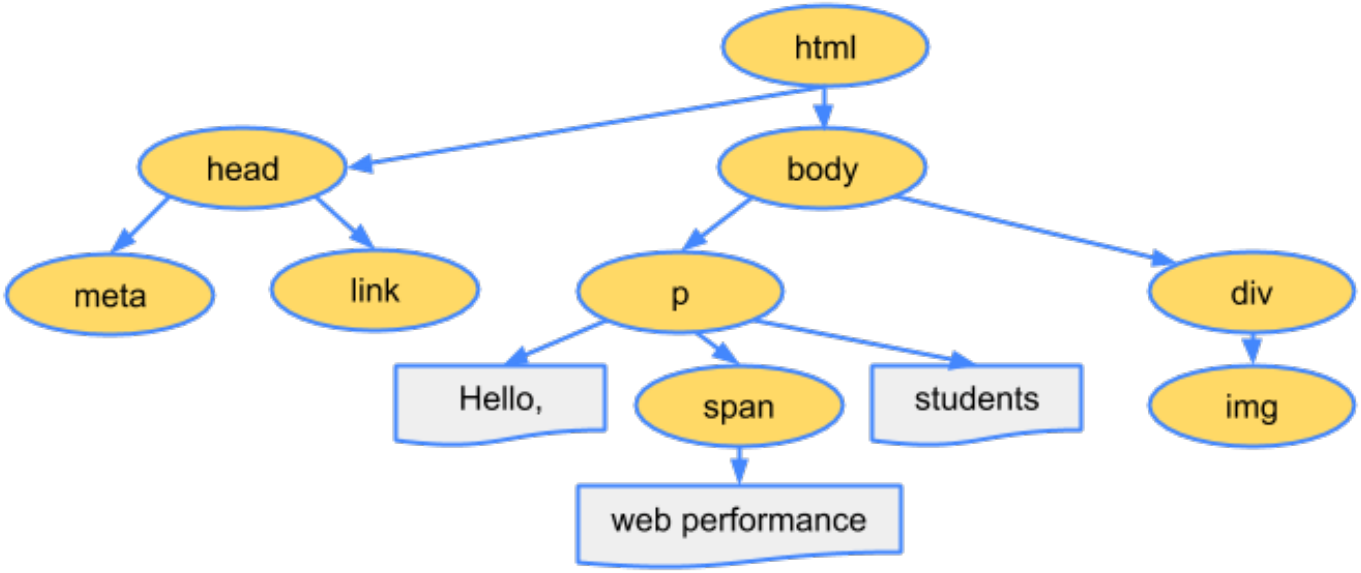
- Process HTML elements and build the *DOM tree*
- Process CSS rules and build the *CSSOM tree*
- Combine the DOM and CSSOM into a *render tree*
- Run layout on the render tree to compute geometry of each node
- Paint them on the screen

Constructing the Object Model

- Document Object Model (DOM) for HTML
  - Each element inside a HTML document is represented as a node
  - Attributes and text between a pair of tags are also nodes
  - Nested element becomes the child node of its parent node
  - The whole HTML document can be represented as a tree called DOM tree
- CSS Object Model (CSSOM) for CSS
  - A tree structure representing CSS rules
  - A working draft of W3C (https://www.w3.org/TR/2011/WD-cssom-20110712/)

Document Object Model

```
<html>
  <head>
    <meta name="viewport" content="width=device-width,initial-scale=1">
    <link href="style.css" rel="stylesheet">
    <title>Critical Path</title>
  </head>
  <body>
    <p>Hello <span>web performance</span> students!</p>
    <div><img src="awesome-photo.jpg"></div>
  </body>
</html>
```



Request Supporting Objects

- Requesting support objects happens at the same time while the DOM is constructed
  - When the browser encounters the `<link>` tag pointing to the style sheet file, it constructs the node, obtain the attribute and send a request to obtain the object specified by the link
  - When the browser encounters the `<img>` tag, it constructs the node, obtain the image's url and sends a request to download the image
- After receiving the style sheet file, the browser starts to parse it and build a CSSOM tree.

CSSOM

body { **font-size:** 16px }

p { **font-weight: bold** }

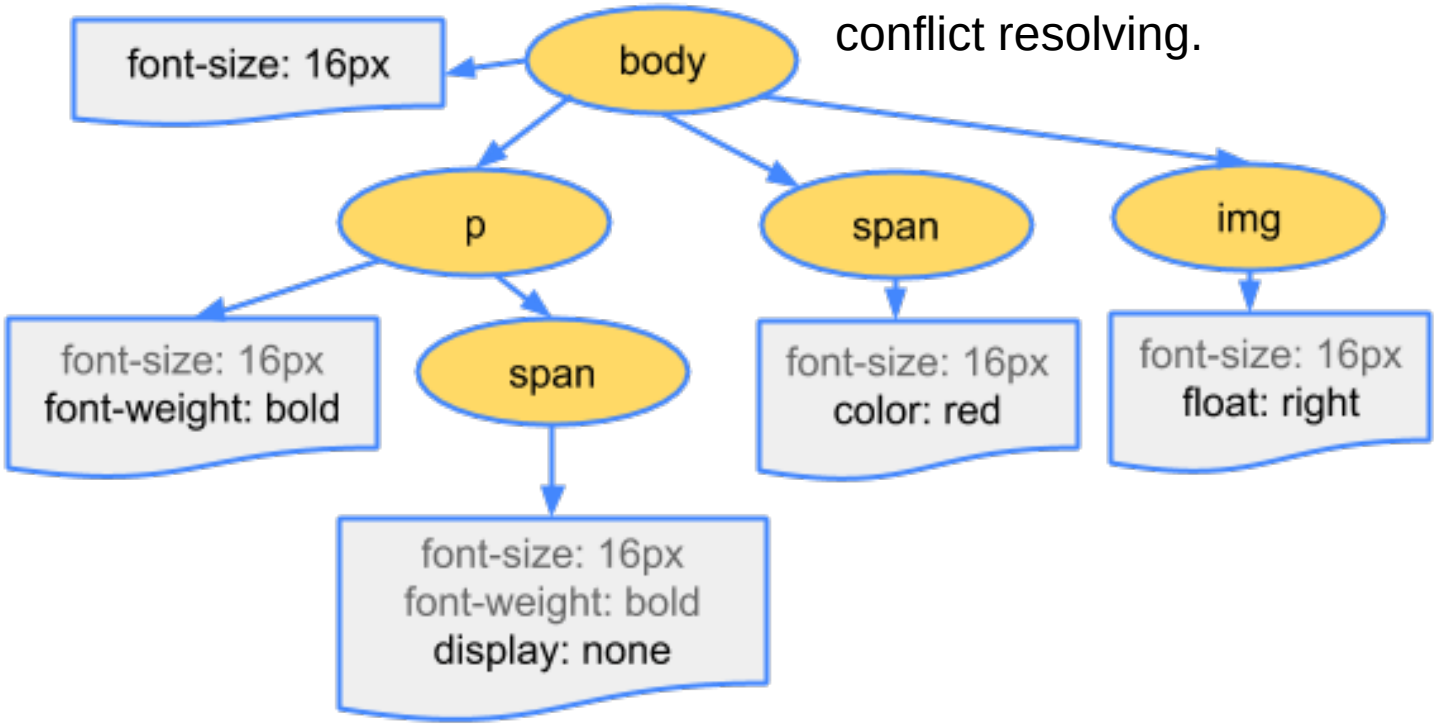
span { **color:** red }

p span { **display: none** }

img { **float: right** }

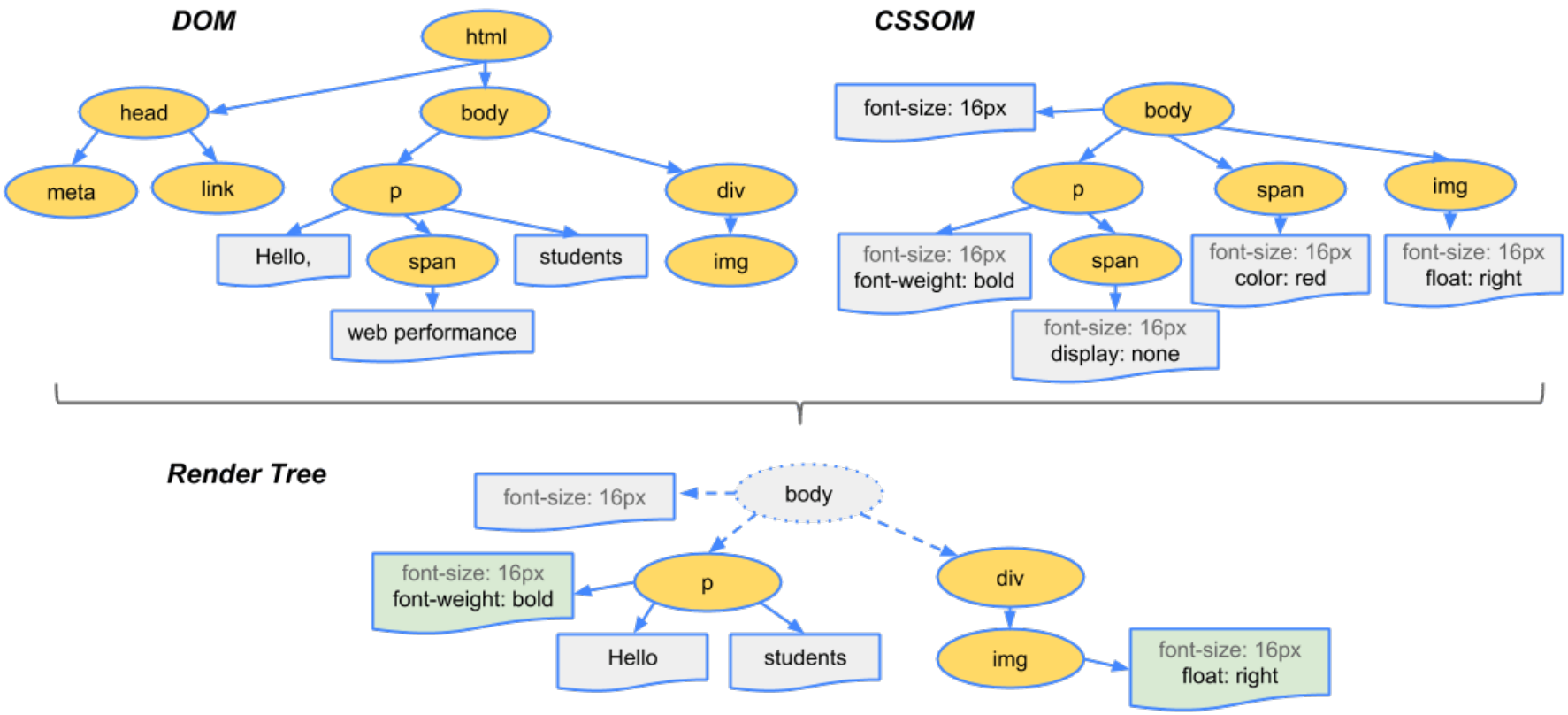
The tree structure is organized following the "cascading" principle

The final set of rules an element has is the result of inheritance and conflict resolving.



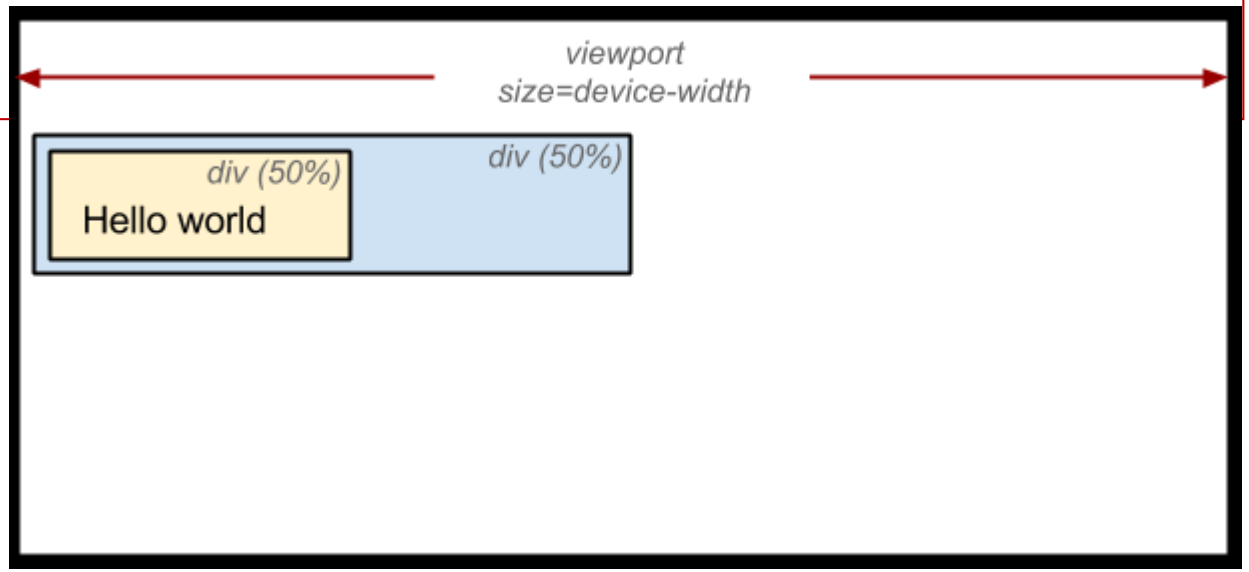
Render Tree Construction

- Render tree is constructed by merging DOM and CSSOM
- Only elements that will be displayed appear in the render tree



Layout and Paint

```
<html>
        <head>
                <meta name="viewport" content="width=device-width,initial-scale=1">
                <title>Critial Path: Hello world!</title>
        </head>
        <body>
                <div style="width: 50%">
                        <div style="width: 50%">Hello world!</div>
                </div>
        </body>
</html>
```



https://developers.google.com/web/fundamentals/performance/critical-rendering-path/render-tree-construction?hl=en

Render Blocking CSS

- Both HTML and CSS are render blocking resources
  - Browser needs to have all of them before it can start to display something on its window
    - CSS links always appear near the top of HTML page so that the browser can send request to obtain them as early as possible
  - Some pages have multiple css to be used under different conditions
    - When you need to print an article/email, all side bars should not appear
    - When the screen size is too small, less important content can be hidden
  - CSS not intended for the current condition will not block the rendering process

```
<link href="style.css" rel="stylesheet">
<link href="style.css" rel="stylesheet" media="all">
<link href="portrait.css" rel="stylesheet" media="orientation:portrait">
<link href="print.css" rel="stylesheet" media="print">
```

JavaScript

- JavaScript is able to modify about every aspect of a page: content, styling usually by responding to user input
- JavaScript may block DOM construction and delay when the page is rendered.
  - Depends on location of JavaScript code
  - Embedded or inline script may block DOM construction, which also delays the initial render
  - When the browser encounters a script tag, DOM construction pauses until the script finishes executing.
  - If JavaScript is stored in an external file, the browser will stop constructing the DOM tree and wait for the file to be downloaded and executed then continue with the rest of the DOM tree construction.

Embedded JavaScript Example

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Week 2</title>
</head>
<body>

<h3>Welcome to <span>HTML5</span>!</h3>
<img src="http://www.w3.org/html/logo/downloads/HTML5_Logo_256.png" alt="HTML5">
<script>
    var span = document.getElementsByTagName('span')[0];
    span.textContent = 'the world of HTML5'; // change DOM text content
    // create a new element, style it, and append it to the DOM
    var loadTime = document.createElement('div');
    loadTime.textContent = 'You loaded this page on: ' + new Date();
    loadTime.style.color = 'blue';
    document.body.appendChild(loadTime);
</script>
<p> Hi, I am after the script </p>
</body>

</html>
```

`document` object represents the current page, it is the starting point to access all other HTML elements

`Date` is a build-in object to work with dates
`new Date()` returns the current date and time

individual element's style is accessed using this syntax:
`element`*`.style.`*`property`

`Document.body` is a convenient way of returning the body element

What the page looks like in browser

sydney.edu.au/engineering/it/~comp5347/2016/week2-js.html

Welcome to the world of HTML5!

You loaded this page on: Tue Mar 08 2016 1

Hi, I am after the script

Elements Console Sources Network Timeline Profiles Resources Security Audits

```
<!DOCTYPE html>
<html>
▶ <head>…</head>
▼ <body>
  ▼ <h3>
      "Welcome to "
      <span>the world of HTML5</span>
      "!"
    </h3>
    <img src="http://www.w3.org/html/logo/downloads/HTML5_Logo_256.png" alt="HTML5">
  ▶ <script>…</script>
  ▼ <div style="color: blue;">
      "You loaded this page on: Tue Mar 08 2016 10:35:44 GMT+1100 (AUS Eastern Daylight
      Time)"
    </div>
    <p> Hi, I am after the script</p>
  </body>
</html>
```

Global Variable Example

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Week 2</title>
</head>
<body>
<h3>Welcome to <span>HTML5</span>!</h3>
<img src="http://www.w3.org/html/logo/downloads/HTML5_Logo_256.png" alt="HTML5">
<script>
      var span = document.getElementsByTagName('span')[0];
      span.textContent = 'the world of HTML5'; // change DOM text content
      // create a new element, style it, and append it to the DOM
      var loadTime = document.createElement('div');
      loadTime.textContent = 'You loaded this page on: ' + new Date();
      loadTime.style.color = 'blue';
      document.body.appendChild(loadTime);
</script>
<p> Hi, I am after the script </p>
<script>
      var anotherLoadTime = document.createElement('div');
      anotherLoadTime.innerHTML = loadTime.innerHTML
      document.body.appendChild(anotherLoadTime);
</script>
</body>
</html>
```

Global Variables Example (cont'd)

Welcome to the world of HTML5!

You loaded this page on: Mon Mar 27 2017 16:38:18 GMT+1100 (AUS Eastern Daylight Time)

Hi, I am after the script

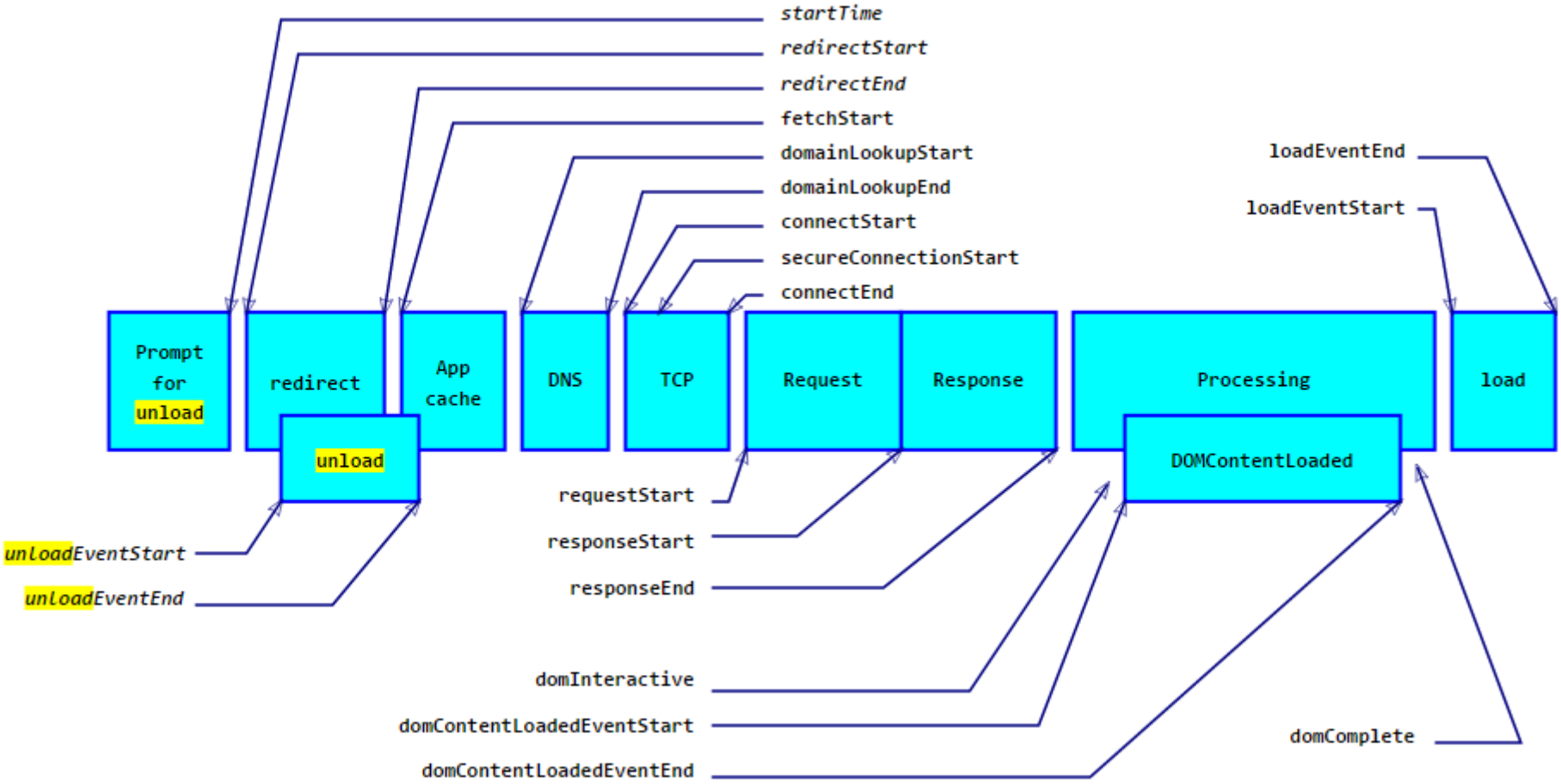
You loaded this page on: Mon Mar 27 2017 16:38:18 GMT+1100 (AUS Eastern Daylight Time)

Navigation Time API

- An interface for web application to access the complete timing information for navigation of a document
- This is yet another W3C working draft
  - Browsers are expected to implement to capture the time of various stage and also to fire relevant events
  - JavaScript codes are able to access the timing information and to listen to the event
- "Navigation started by clicking on a link, or entering the URL in the user agent's address bar, or form submission, or initializing through a script operation other than the ones used by reload and back_forward".

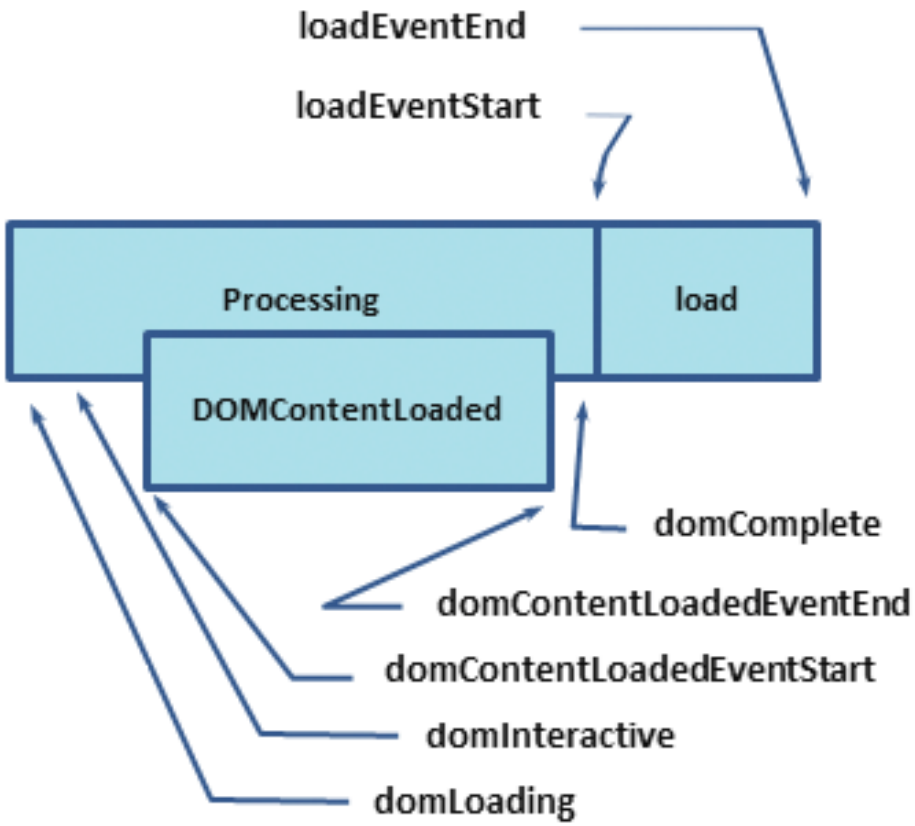
https://www.w3.org/TR/navigation-timing-2/

Overall Navigation Process



https://www.w3.org/TR/navigation-timing-2/

Process related with Rendering



domLoading: this is the starting timestamp of the entire process, the browser is about to start parsing the first received bytes of the HTML document.

domContentLoaded: marks the point when browser has finished parsing the HTML document the DOM is constructed.

domComplete: as the name implies, all of the processing is complete and all of the resources on the page (images, etc.) have finished downloading, **onLoad** event will file

Analyzing critical rendering path performance

- Simple page with only HTML and an image
- A more complex page with HTML, an image and external CSS and JS file
- An example with HTML, an image and embedded CSS and JS file

https://developers.google.com/web/fundamentals/performance/critical-rendering-path/analyzing-crp?hl=en

Page with only HTML and image

```
<html>
 <head>
<meta name="viewport" content="width=device-width,initial-scale=1">
 <title>Critical Path: No Style</title>
</head>
<body>
<p>Hello <span>web performance</span> students!</p>
<div><img src="awesome-photo.jpg"></div>
</body>
</html>
```

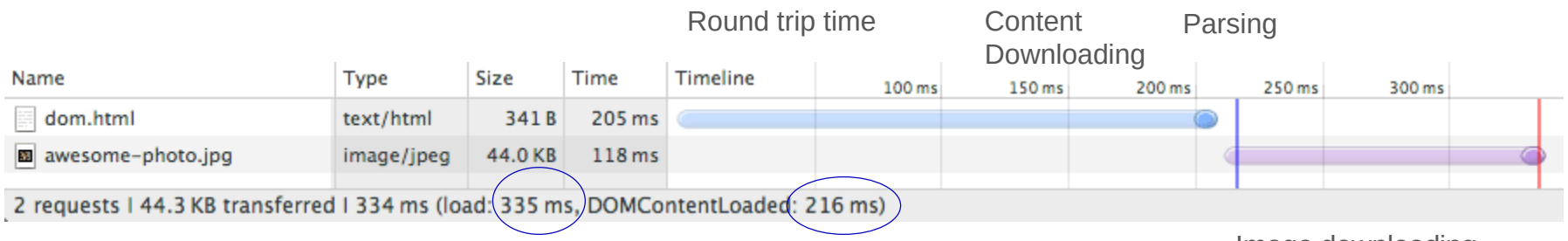
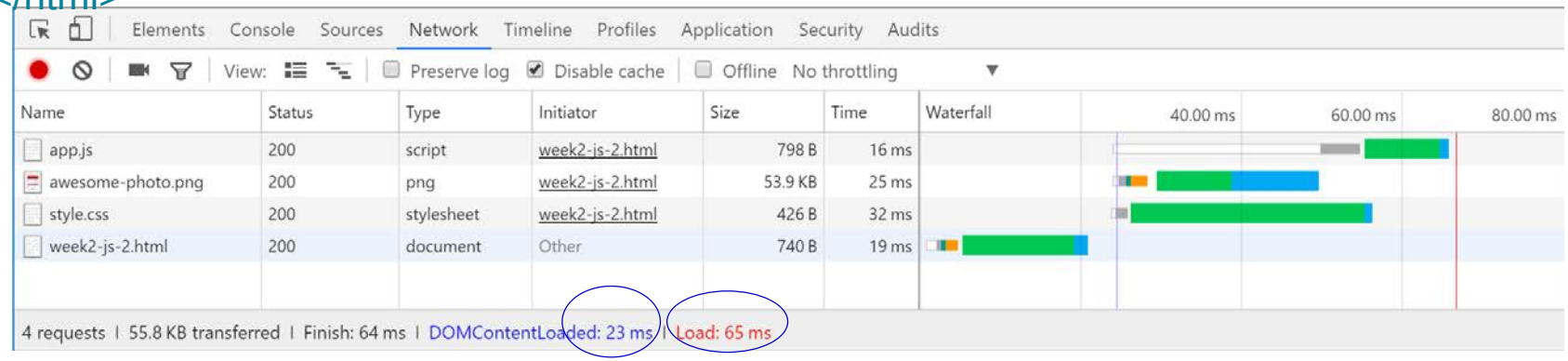


Image downloading starts during page parsing

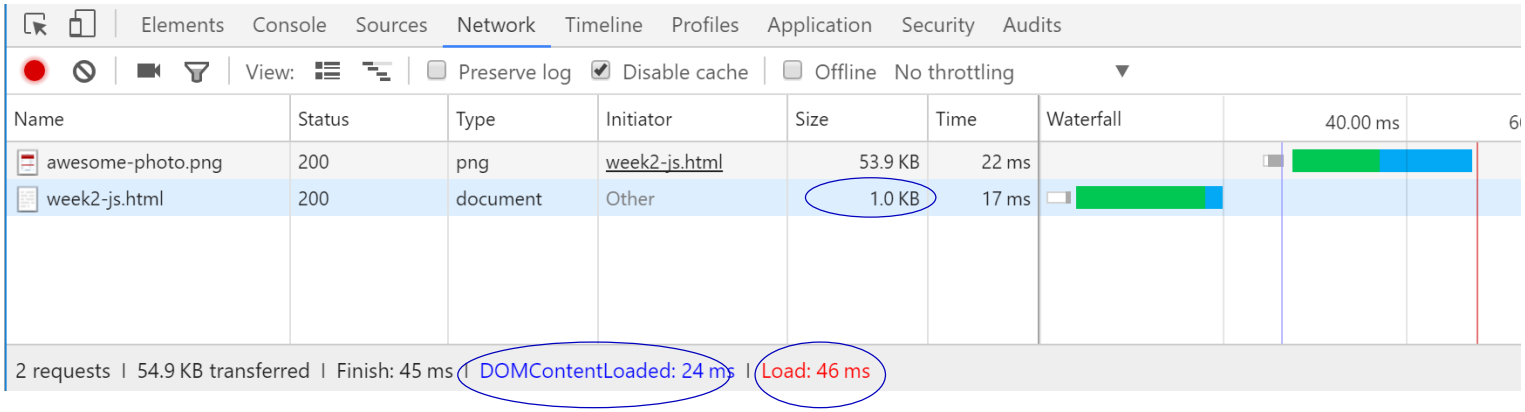
Page with external CSS and JS file

```
<html>
<head>
<title>Critical Path: Measure Script</title>
 <meta name="viewport" content="width=device-width,initial-scale=1">
<link href="style.css" rel="stylesheet">
</head>
<body onload="measureCRP()">
<p>Hello <span>web performance</span> students!</p>
 <div><img src="awesome-photo.jpg"></div>
<script src="timing.js"></script>
 </body>
</html>
```



Page with embedded CSS and JS

```
<html>
 <head>
<title>Critical Path: Measure Inlined</title>
<meta name="viewport" content="width=device-width,initial-scale=1">
 <style>
      p { font-weight: bold }
  </style>
</head>
 <body>
    <p>Hello <span>web performance</span> students!</p> <div>
    <img src="awesome-photo.jpg"></div>
    <script>
        var span = document.getElementsByTagName('span')[0];
      …
    </script>
</body>
</html>
```



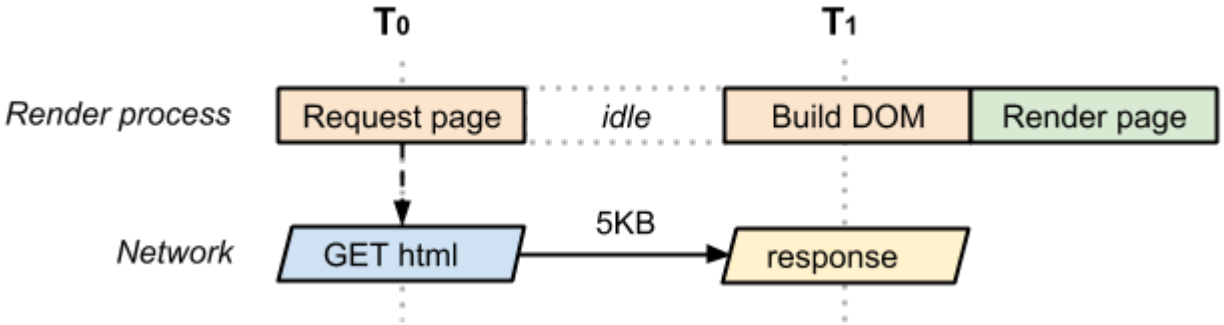
Performance Patterns

- **Critical Resource**: resource that needs to be downloaded before rendering the page
- **Critical Path Length**: number of round trips to fetch all critical resources; ignore the initial tcp connection set up time
- **Critical Bytes**: total amount of bytes required get before rendering the page

Page with only HTML and image

```
<html>
<head>
<title>Critical Path: Measure Script</title>
 <meta name="viewport" content="width=device-width,initial-scale=1">
</head>
<body>
<p>Hello <span>web performance</span> students!</p>
 <div><img src="awesome-photo.jpg"></div>
</body>
</html>
```

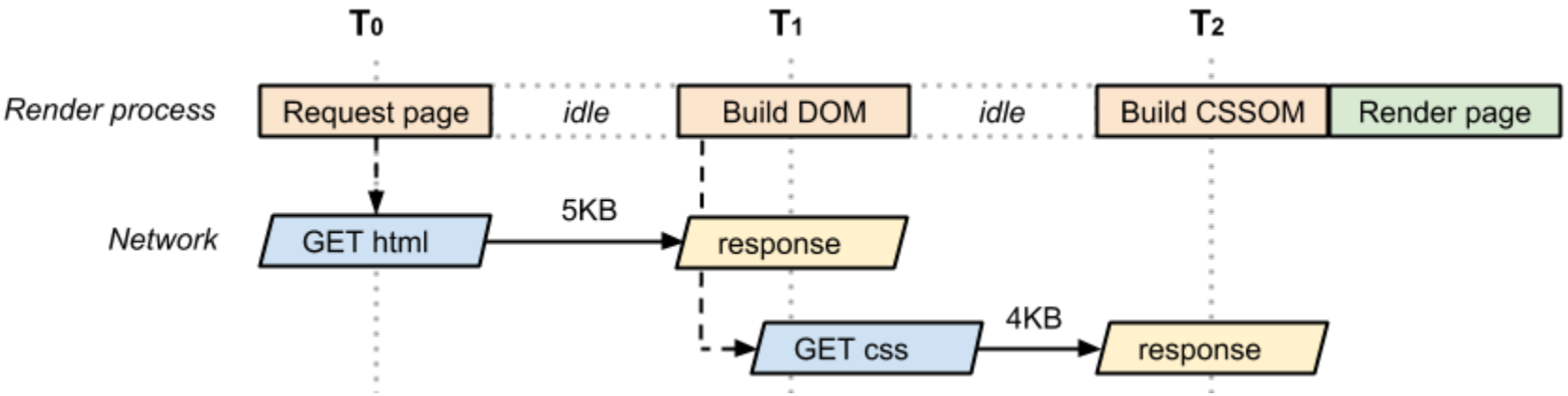
One critical resource
One round trip
5KB critical bytes



Page with external CSS

```
<html>
<head>
<title>Critical Path: Measure Script</title>
 <meta name="viewport" content="width=device-width,initial-scale=1">
<link href="style.css" rel="stylesheet">
</head>
<body onload="measureCRP()">
<p>Hello <span>web performance</span> students!</p>
 <div><img src="awesome-photo.jpg"></div>
</body>
</html>
```

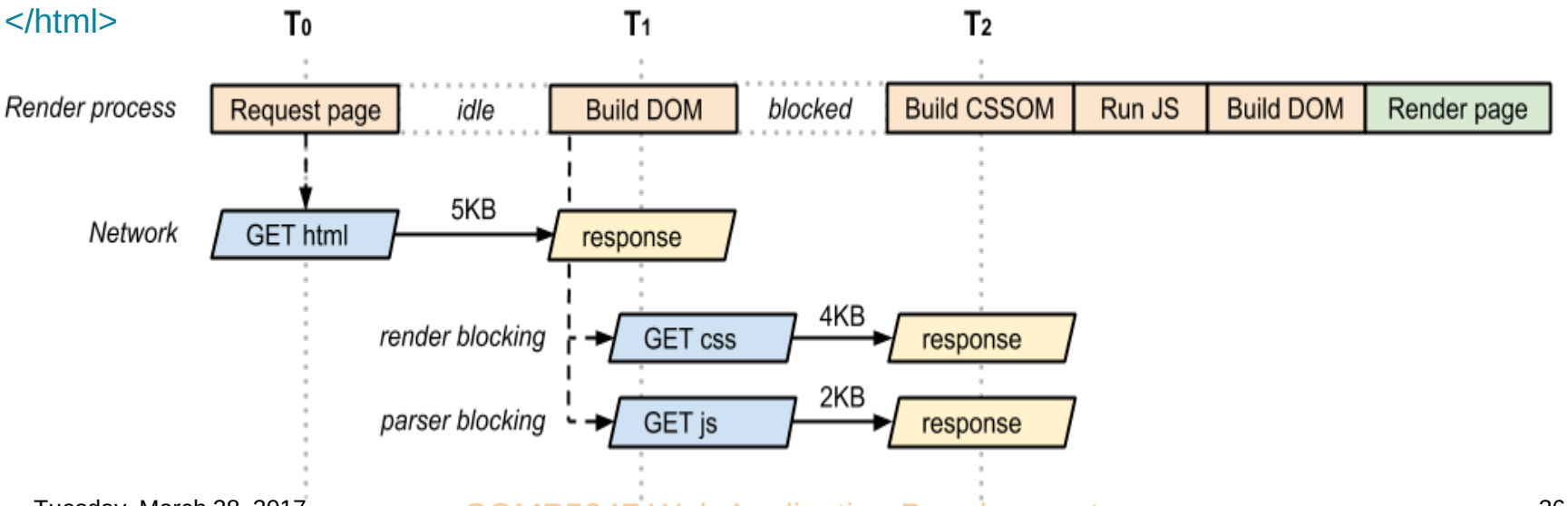
Two critical resources
Two round trips
9KB critical bytes



Page with external CSS and JS

```
<html>
<head>
<title>Critical Path: Measure Script</title>
 <meta name="viewport" content="width=device-width,initial-scale=1">
<link href="style.css" rel="stylesheet">
</head>
<body>
<p>Hello <span>web performance</span> students!</p>
 <div><img src="awesome-photo.jpg"></div>
<script src="app.js"></script>
 </body>
</html>
```

Three critical resources
Two round trips
11KB critical bytes



References

- Google Developers Web Fundamentals[https://developers.google.com/web/fundamentals/?hl=en]
  - Critical Rendering path

  [https://developers.google.com/web/fundamentals/performance/critical-rendering-path/?hl=en]