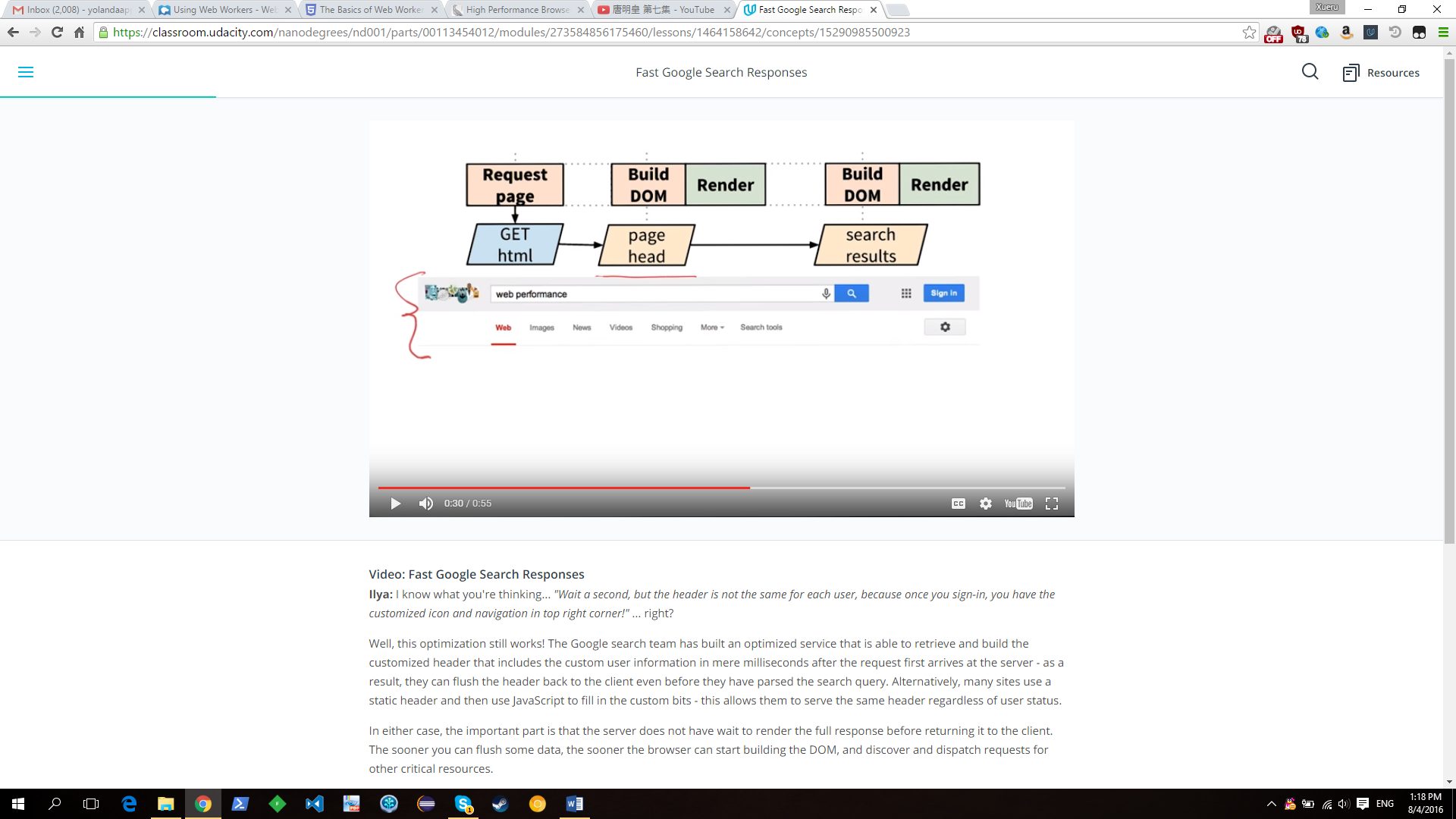
The Critical Rendering Path

# Critical Rendering Path

1. Process HTML markup and build the DOM tree.
2. Process CSS markup and build the CSSOM tree.
3. Combine the DOM and CSSOM into a render tree.
4. Run layout on the render tree to compute geometry of each node.
5. Paint the individual nodes to the screen.

# HTML 🡪 DOM

Characters -> Tokens -> Nodes -> DOM



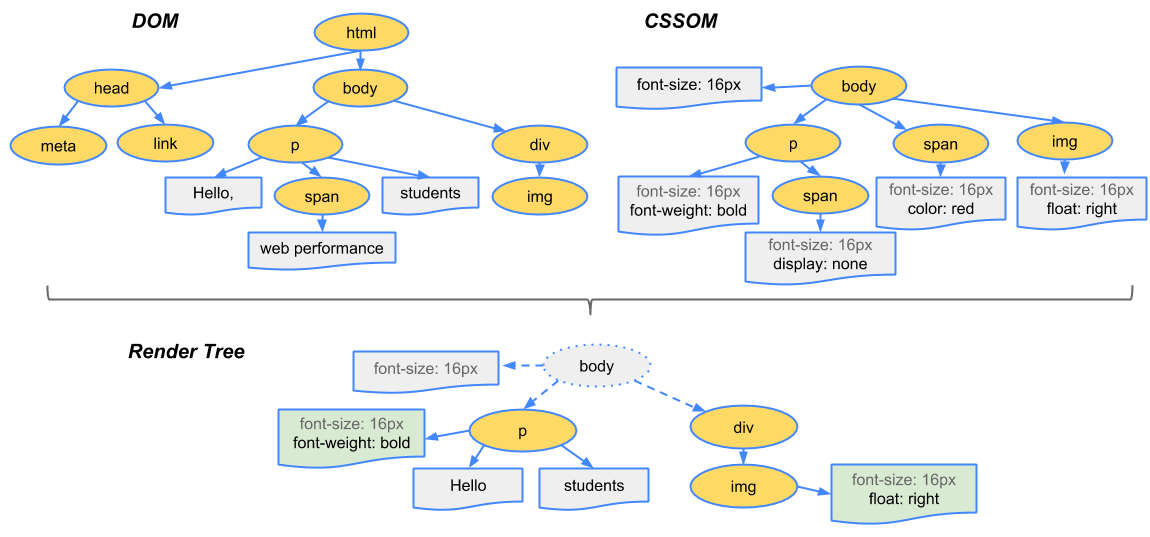
# CSS 🡪 CSSOM

Characters -> Tokens -> Nodes -> CSSOM

\*CSS cascade down!

CSS is a render blocking resource, get it down to the client as soon and as quickly as possible to optimize the time to first render!

# Render Tree



To construct the render tree, the browser roughly does the following:

1. Starting at the root of the DOM tree, traverse each visible node.
   * Some nodes are not visible at all (e.g. script tags, meta tags, and so on), and are omitted since they are not reflected in the rendered output.
   * Some nodes are hidden via CSS and are also omitted from the render tree - e.g. the span node in example above is missing from the render tree because we have an explicit rule that sets “display: none” property on it.
2. For each visible node find the appropriate matching CSSOM rules and apply them.
3. Emit visible nodes with content and their computed styles.

# Optimization

**1 Minify the HTML, CSS and JavaScript files, get rid of comments and white space**

**2 Compress file (HTML, CSS and JavaScript)**

**3 Cache (HTML, CSS and JavaScript)**

**4 Inline CSS, JS**

**5 Add media to CSS, Async JS**

**6 Build CSSOM ASAP and put JS at the end of body (defer execution)**

## Patterns

**1 Minimize Bytes**

**2 Reduce critical resources**

**3 Shorten CRP length**

## CSS Optimization

<link href="style.css" rel="stylesheet"> (blocking)

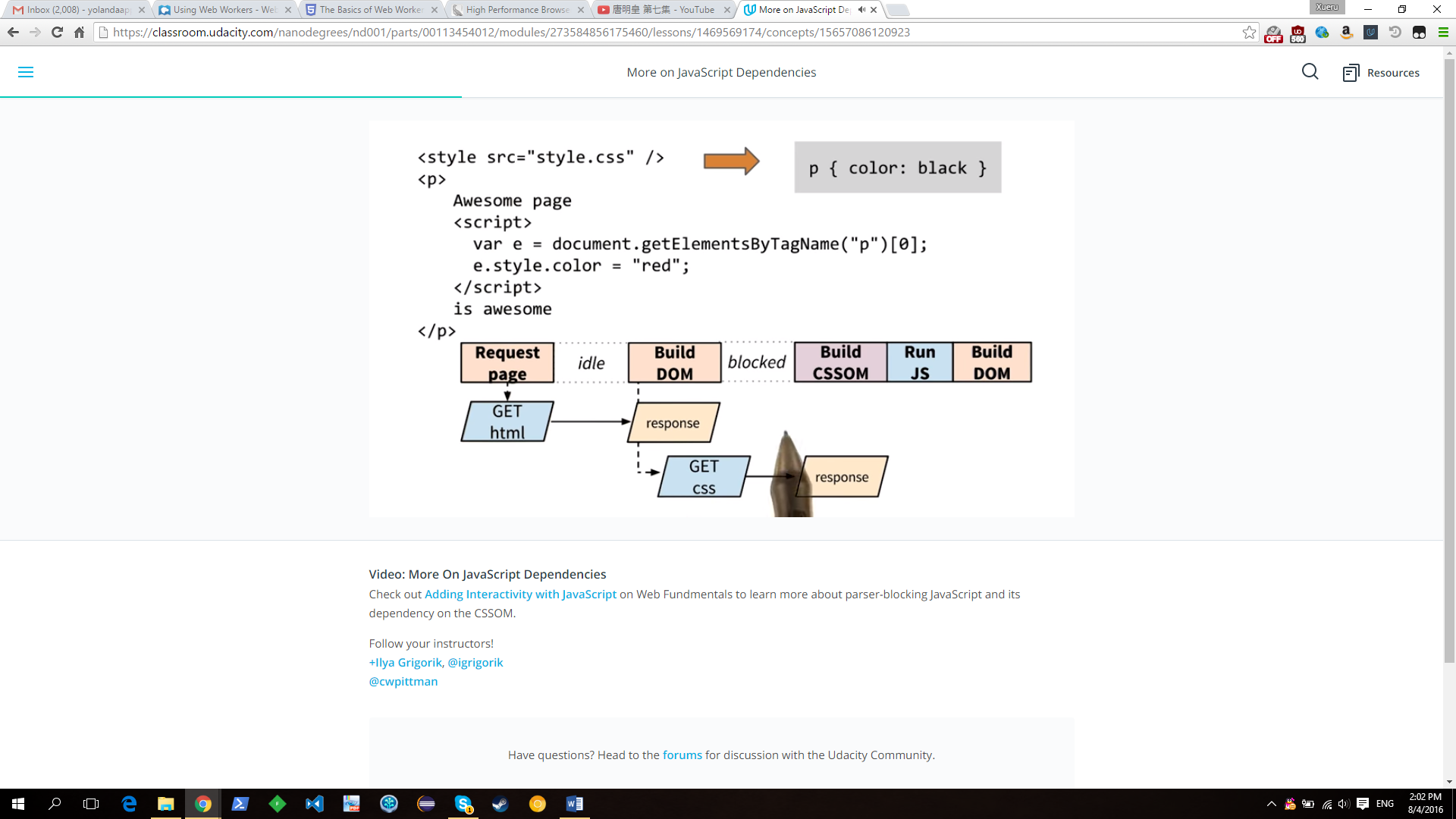
<link href="style.css" rel="stylesheet" media="all"> (same as above, they are equal)

<link href="portrait.css" rel="stylesheet" media="orientation:portrait"> (not blocking)

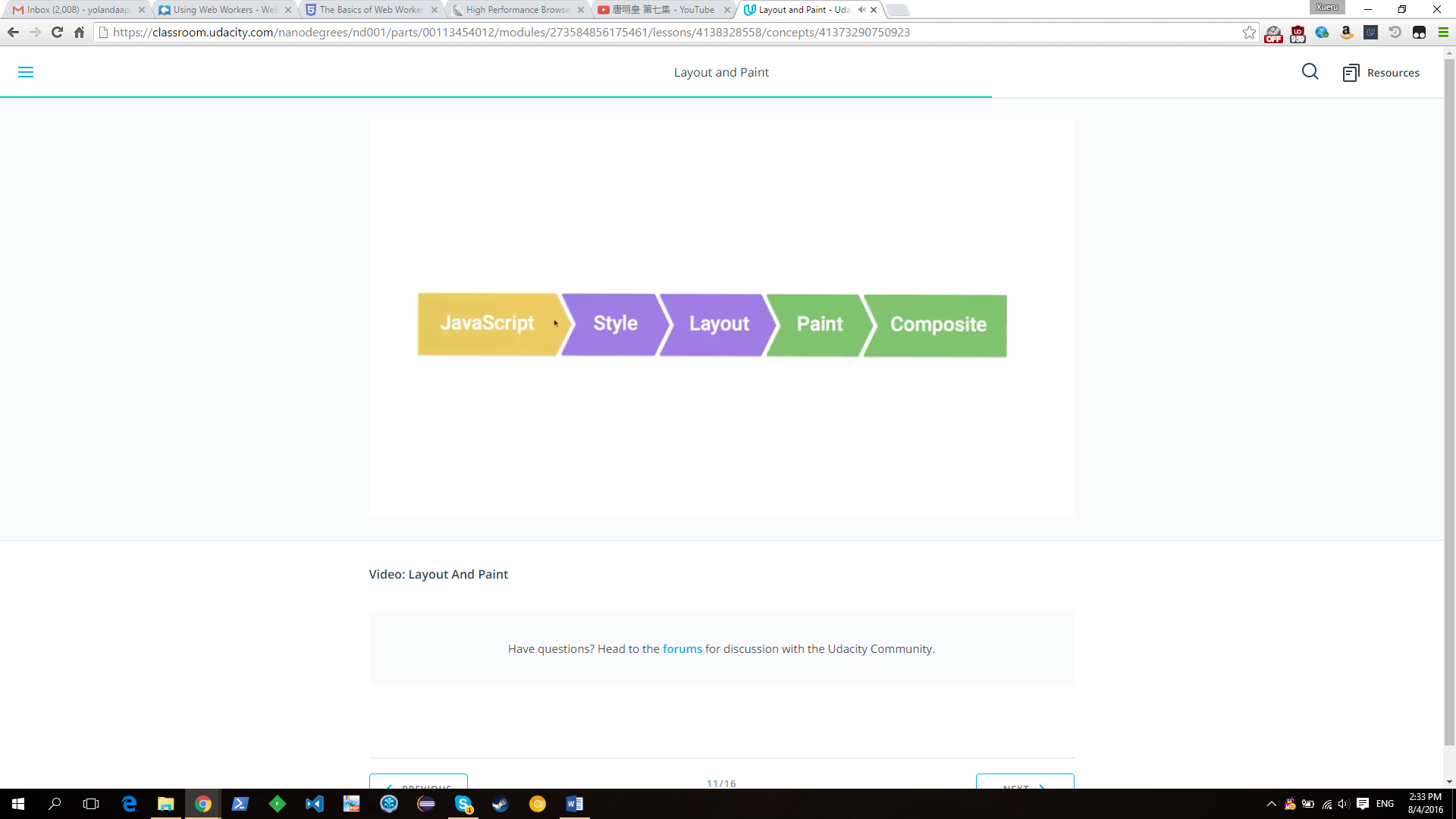
<link href="print.css" rel="stylesheet" media="print"> (not blocking)

## JS Optimization

* **Blocking**: <script src="anExteralScript.js"></script>
* **Inline**: <script>document.write("this is an inline script")</script>
* **Async**: <script async src="anExternalScript.js"></script>



Browser Rendering Optimization

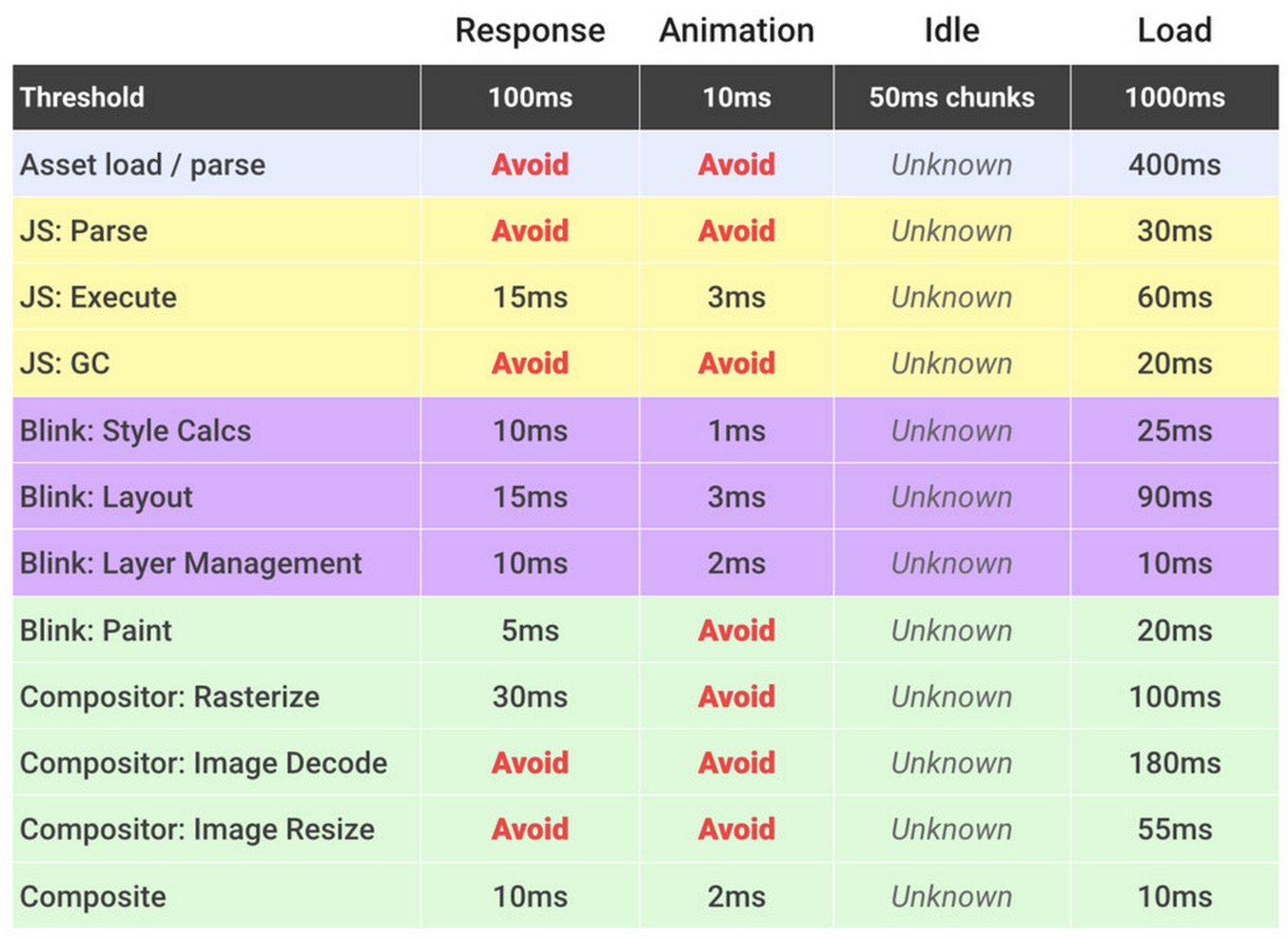


1 Use JS or CSS to change width, height etc. of the element. It will trigger layout change, painting and composite

2 Change the background etc. of an element. It will only trigger paint and composite

[CSS trigger.com](https://csstriggers.com/)

# Time amount for different tasks



# Response Animate Idle Load

Response: 100 ms

Animate: 16 ms (usually 10 ms due to browser overhead)

Idle: 50 ms

Load: 1 second

# First Last Invert Play

Example: <https://github.com/udacity/devsummit/blob/master/src/static/scripts/components/card.js>

(Strategy: Move expensive things up front)

# Timeline Testing Strategy

1 Quit other apps

2 go incognito (avoid extensions)

3 focus on the cause of bottlenecks, not symptoms

4 measure first, then optimize

5 hard reload (Ctrl + F5)

Practice of finding Jank:

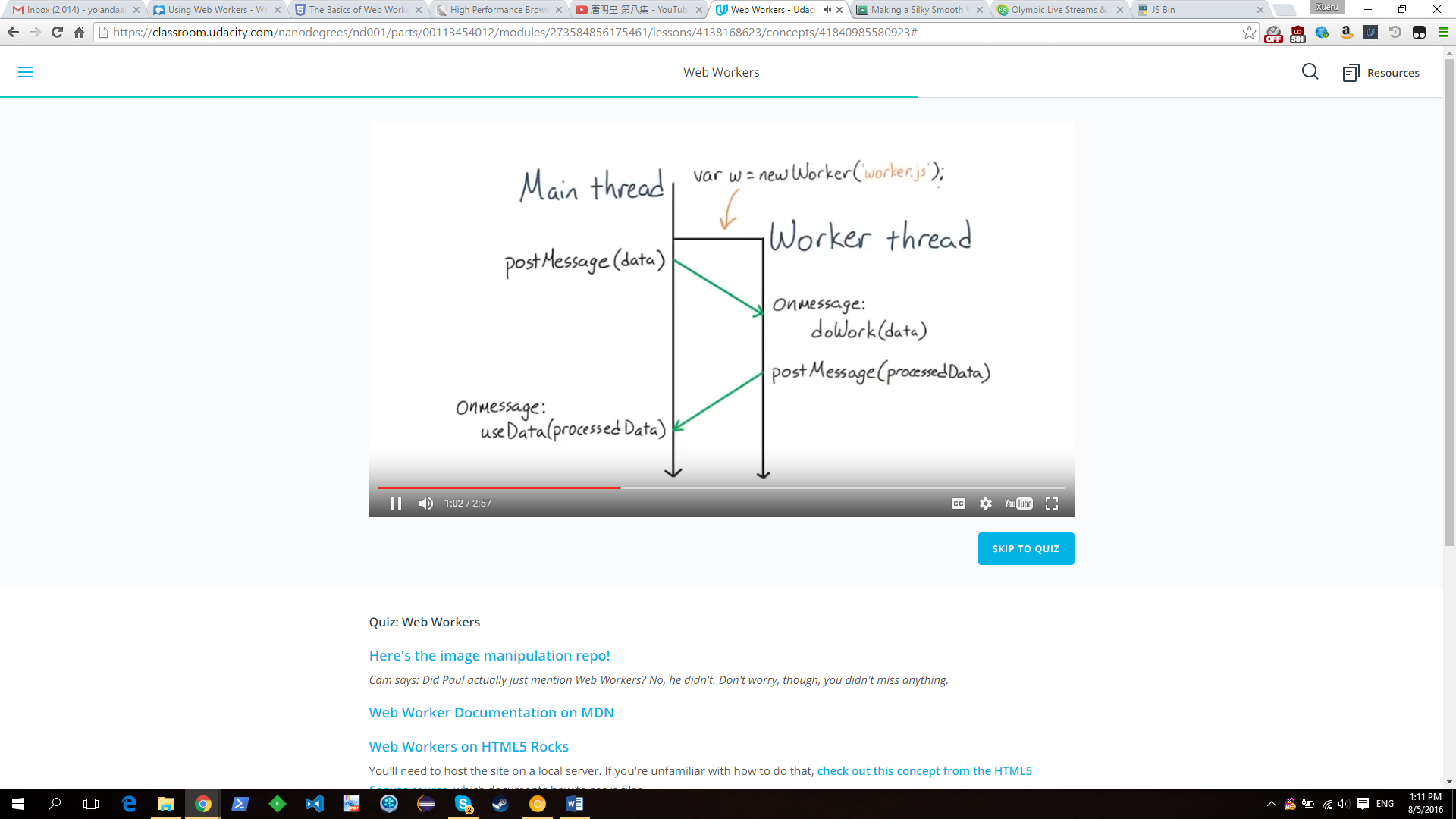
<http://output.jsbin.com/nanana/2/quiet#>

<http://output.jsbin.com/woyoce/1/quiet>

<http://output.jsbin.com/feloni/3/quiet>

# Web Worker

Best used for image processing.



<http://www.html5rocks.com/en/tutorials/workers/basics/>

<https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API/Using_web_workers>

Practice:

<https://github.com/udacity/web-workers-demo>

# JS Memory Management

* A better alternative to manual de-referencing is to use variables with an appropriate scope. I.e. instead of a global variable that’s nulled out, just use a function-local variable that goes out of scope when it’s no longer needed. This means cleaner code with less to worry about.
* Ensure that you’re unbinding event listeners where they are no longer required, especially when the DOM objects they’re bound to are about to be removed
* If you’re using a data cache locally, make sure to clean that cache or use an aging mechanism to avoid large chunks of data being stored that you’re unlikely to reuse

<https://www.smashingmagazine.com/2012/11/writing-fast-memory-efficient-javascript/>

<http://buildnewgames.com/garbage-collector-friendly-code/>

Practice: (web worker, requestAnimationFrame)

<https://github.com/udacity/qrcode>

# Block Element Modifier (Reduce CSS Selector)

Use single class to modify elements.

Example of CSS selectors: <http://output.jsbin.com/gozula/1/quiet>

Slow: div.box:not(:empty):last-of-type .title

Fast: .box:nth-last-child(-n+1) .title

Fastest: .box—last > .title-container > .title

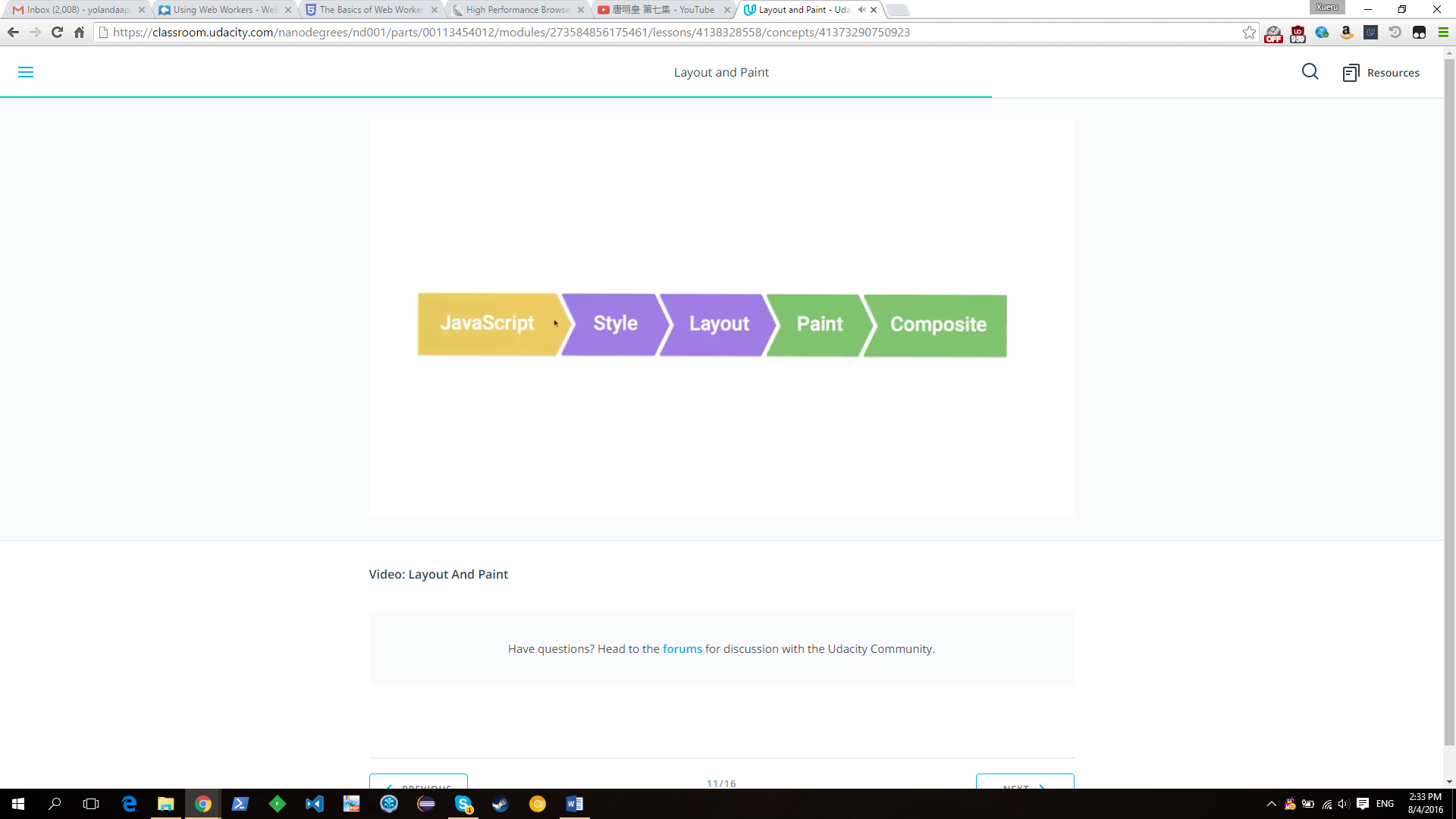
Practice of reducing affected elements/ reduce selector complexity:

<https://dl.dropboxusercontent.com/u/2272348/codez/udacity/box-recalc-style-slow.html>

<https://dl.dropboxusercontent.com/u/2272348/codez/udacity/box-recalc-style-slow.html?dl=1>

# Forecd Synchronous Layout (Forced reflow)

Calculate layout in Javascript that changes style will cause moving layout infront of style.



Example: <http://udacity.github.io/60fps/lesson5/stopFSL/index.html>

To avoid it, read layout properties then batch style changes.

for (var p=0; p<paragraphs.length; p++){

var blockWidth = greenBlock.offsetWidth; (Forecd Synchronous Layout)

paragraphs[p].style.width = blockWidth + 'px';

}

===================>

var blockWidth = greenBlock.offsetWidth;

for (var p=0; p<paragraphs.length; p++){

paragraphs[p].style.width = blockWidth + 'px';

}

Practice: <https://github.com/udacity/pizza-perf>

# Managing Layers

Layers are usually managed by the browser automatically.

If you hit a paint issue, you might want to promote an element to its own layer.

This could avoid painting on the fly.

1 Analysis timeline and paint profile

2 Fix: tell browser which element is going to be changing

.circle{

will-change: transform;

}

.circle {

Transform: translate(0); //For older browsers, will surely generate a new layer

Will-change: transform; //allow browser to decide if it should generate a new layer

}

Practice:

<http://udacity.github.io/60fps/lesson6/willChange/index.html>

<http://www.html5rocks.com/static/demos/parallax/demo-1a/demo.html>