



**drupalcamp**SPAIN  
VALENCIA · 2014

# Frontend Performance: Illusions & browser rendering

Manuel Garcia  
[d.org/user/213194](https://d.org/user/213194)

# The current situation

**80-90%** of the end-user response time is spent on the  
frontend

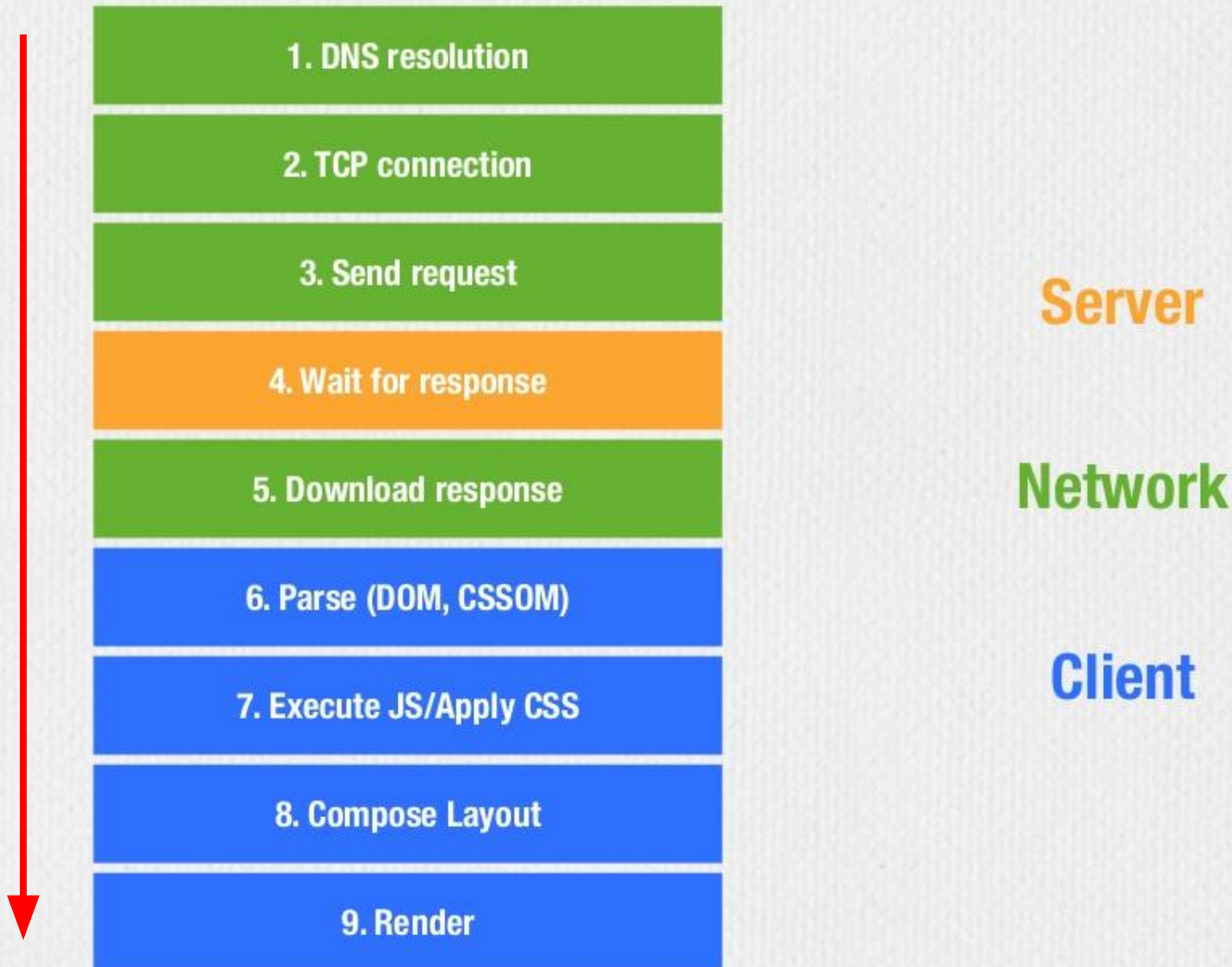
<http://www.stevesouders.com/blog/2012/02/10/the-performance-golden-rule/>

# The current situation

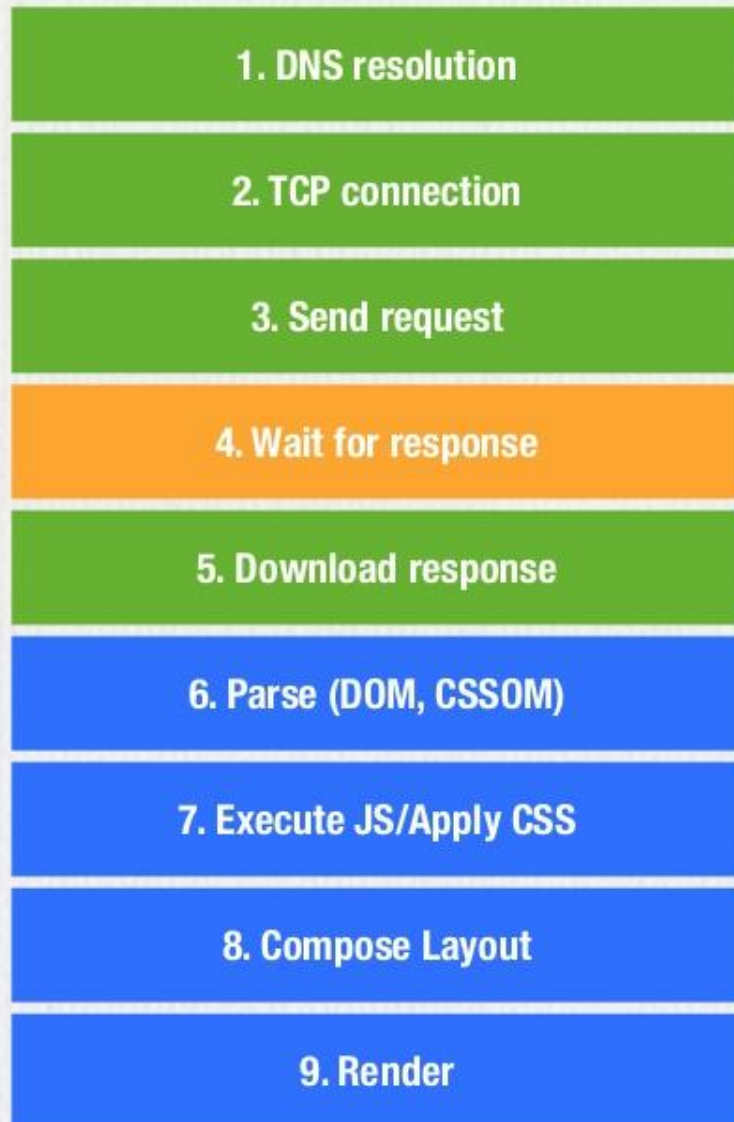
**80-90%** of the end-user response time is spent on the  
frontend

But frontenders are not to blame for ALL of it.

# The current situation



# The current situation



**85-90% of performance happens in the browser**

# How it works

they

This is what ~~you~~ control

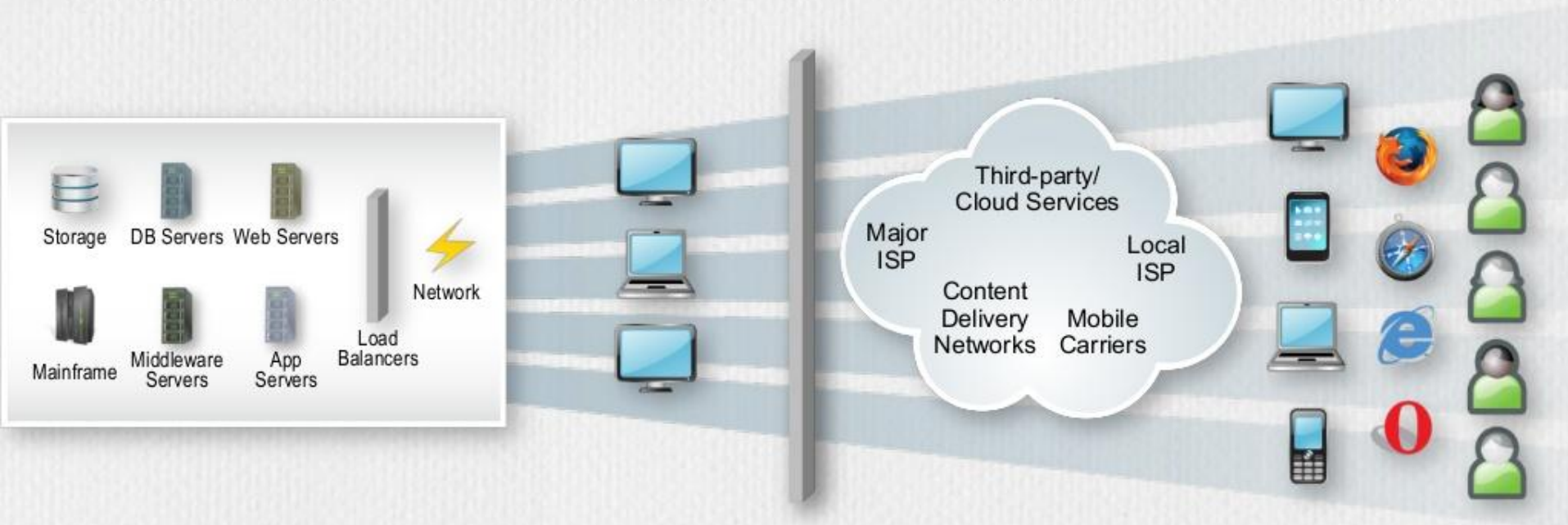
This is what you're blamed for

(CLOUD) DATA CENTER

INTERNAL USERS

INTERNET

CUSTOMERS



## Application Delivery Chain

source: @jerontjepkema

<http://www.slideshare.net/MeasureWorks/measureworks-why-people-hate-to-wait-for-your-website-to-load-and-how-to-fix-that>

# So...

Not *every second* wasted waiting on the browser is the frontenders fault.

**Latency** is a big bottleneck, especially on mobile.

## What I won't be covering:

Anything that happens on the server (gzip, varnish, memcache etc).

Anything that happens from the server to the browser.

## What I will cover:

*Anything* that happens after the browser gets the first packet.



# Outline

1. Brief explanation of how browsers make sense of and render our mess.
2. The path to the first paint - why it is important and how to get there faster.
3. Rendering performance - how not to shoot yourself in the foot.
4. Drupal - the current situation

# The browser

# What the browser does

1. Starts receiving data. First packet is about 14K.
2. Parses the HTML and constructs the DOM.
3. Starts downloading assets (images, css, js) - in the order as they come in the HTML source code.
4. Parses CSS and constructs the CSSOM.
5. Constructs the Render Tree (DOM + CSSOM)
6. Calculates Layout (size & position)
7. Paints & composites the layers.

# What the browser does

HTML - Source optimization

# What the browser does

## HTML - Source optimization

1. Prioritize content delivery - source order. (Serve first what matters to the user)

# What the browser does

## HTML - Source optimization

1. Prioritize content delivery - source order. (Serve first what matters to the user)
2. Limit and minimize assets to download (HTTP requests)

# What the browser does

## HTML - Source optimization

1. Prioritize content delivery - source order. (Serve first what matters to the user)
2. Limit and minimize assets to download (HTTP requests)
3. Keep the number of DOM elements under control. (Divitis)

-

# What the browser does

Optimizing CSSOM construction



# What the browser does

## Optimizing CSSOM construction

- External stylesheets block rendering.

# What the browser does

## Optimizing CSSOM construction

- External stylesheets block rendering.
- Serve CSS as early as possible (in <head>)

# What the browser does

## Optimizing CSSOM construction

- External stylesheets block rendering.
- Serve CSS as early as possible (in <head>)
- Avoid the use of inefficient CSS selectors (body \*, #footer h3)

# What the browser does

## Optimizing CSSOM construction

- External stylesheets block rendering.
- Serve CSS as early as possible (in <head>)
- Avoid the use of inefficient CSS selectors (body \*, #footer h3)
- Remove unused CSS rules. Cleanup!

# What the browser does

## Optimizing CSSOM construction - CSS selectors

*Browser engines evaluate each rule from right to left, starting from the rightmost selector (called the "key") and moving through each selector until it finds a match or discards the rule.*

[https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Writing\\_efficient\\_CSS](https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Writing_efficient_CSS)

# What the browser does

## Optimizing CSSOM construction - CSS selectors

1. Avoid a universal key selector: \* *Allow elements to inherit from ancestors, or use a class to apply a style to multiple elements.*

# What the browser does

## Optimizing CSSOM construction - CSS selectors

1. Avoid a universal key selector: \* *Allow elements to inherit from ancestors, or use a class to apply a style to multiple elements.*
2. Make your rules as specific as possible. *Prefer class and ID selectors over tag selectors.*

# What the browser does

## Optimizing CSSOM construction - CSS selectors

1. Avoid a universal key selector: \* *Allow elements to inherit from ancestors, or use a class to apply a style to multiple elements.*
2. Make your rules as specific as possible. *Prefer class and ID selectors over tag selectors.*
3. Remove redundant qualifiers.
  - ID selectors qualified by class and/or tag selectors
  - Class selectors qualified by tag selectors



# What the browser does

## Optimizing CSSOM construction - CSS selectors

1. Avoid a universal key selector: \* *Allow elements to inherit from ancestors, or use a class to apply a style to multiple elements.*
2. Make your rules as specific as possible. *Prefer class and ID selectors over tag selectors.*
3. Remove redundant qualifiers.
  - ID selectors qualified by class and/or tag selectors
  - Class selectors qualified by tag selectors
4. Use class selectors instead of descendant selectors.

# What the browser does

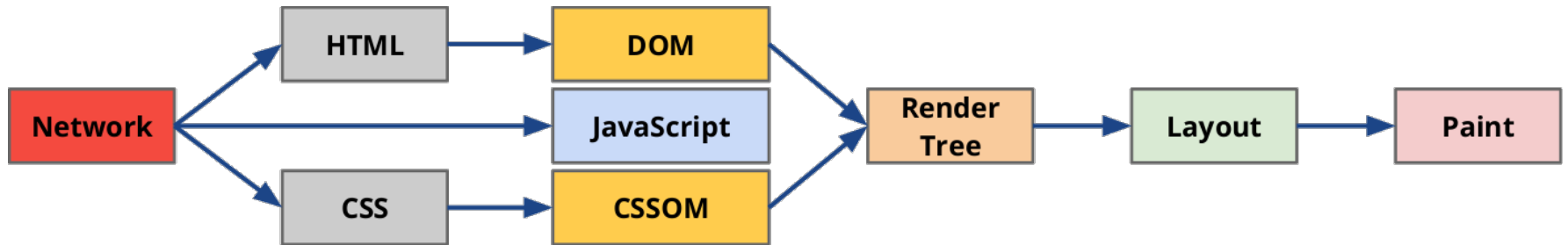
And what about Javascript?

# What the browser does

And what about Javascript?

Because it can change the DOM and the CSSDOM, when the browser sees a `<script>` tag it will **block** downloading of other assets until the js file has been downloaded and executed.

# What the browser does



source: ([Building Faster Websites: Crash Course on Web Performance](#), Fluent 2013)

# What the browser does

## Javascript:

- Avoid it if not necessary.

# What the browser does

## Javascript:

- Avoid it if not necessary.
- Inline it in `<head>` *if necessary* for above the fold, if it is small, and you are NOT changing the DOM or CSSOM.

# What the browser does

## Javascript:

- Avoid it if not necessary.
- Inline it in `<head>` *if necessary* for above the fold, if it is small, and you are NOT changing the DOM or CSSOM.
- Normally just place it at the bottom of the page.

# What the browser does

## Javascript:

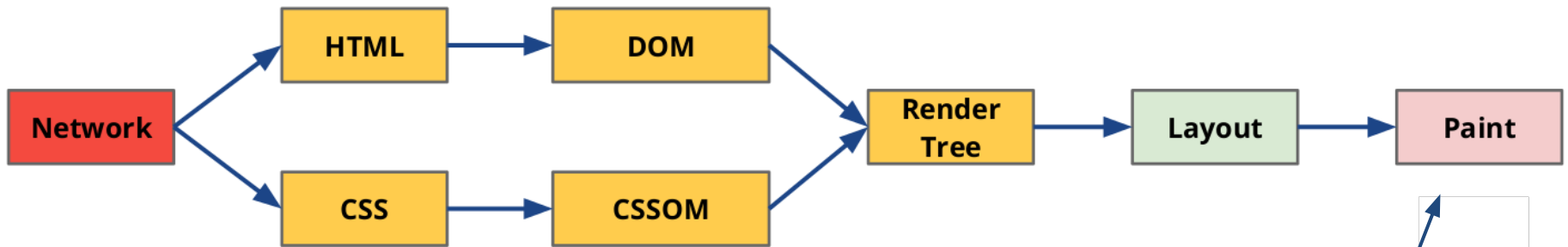
- Avoid it if not necessary.
- Inline it in `<head>` *if necessary* for above the fold, if it is small, and you are NOT changing the DOM or CSSOM.
- Normally just place it at the bottom of the page.
- And/or defer it: `<script async src="progressively-enhancing.js">`

-



**The path to first paint**

# The path to first paint



Make it here fast  
Make it count!

# The path to first paint

It's **what your users first see.**

It's what the user is stuck with on mobile while waiting to load your  $10^6$  assets.

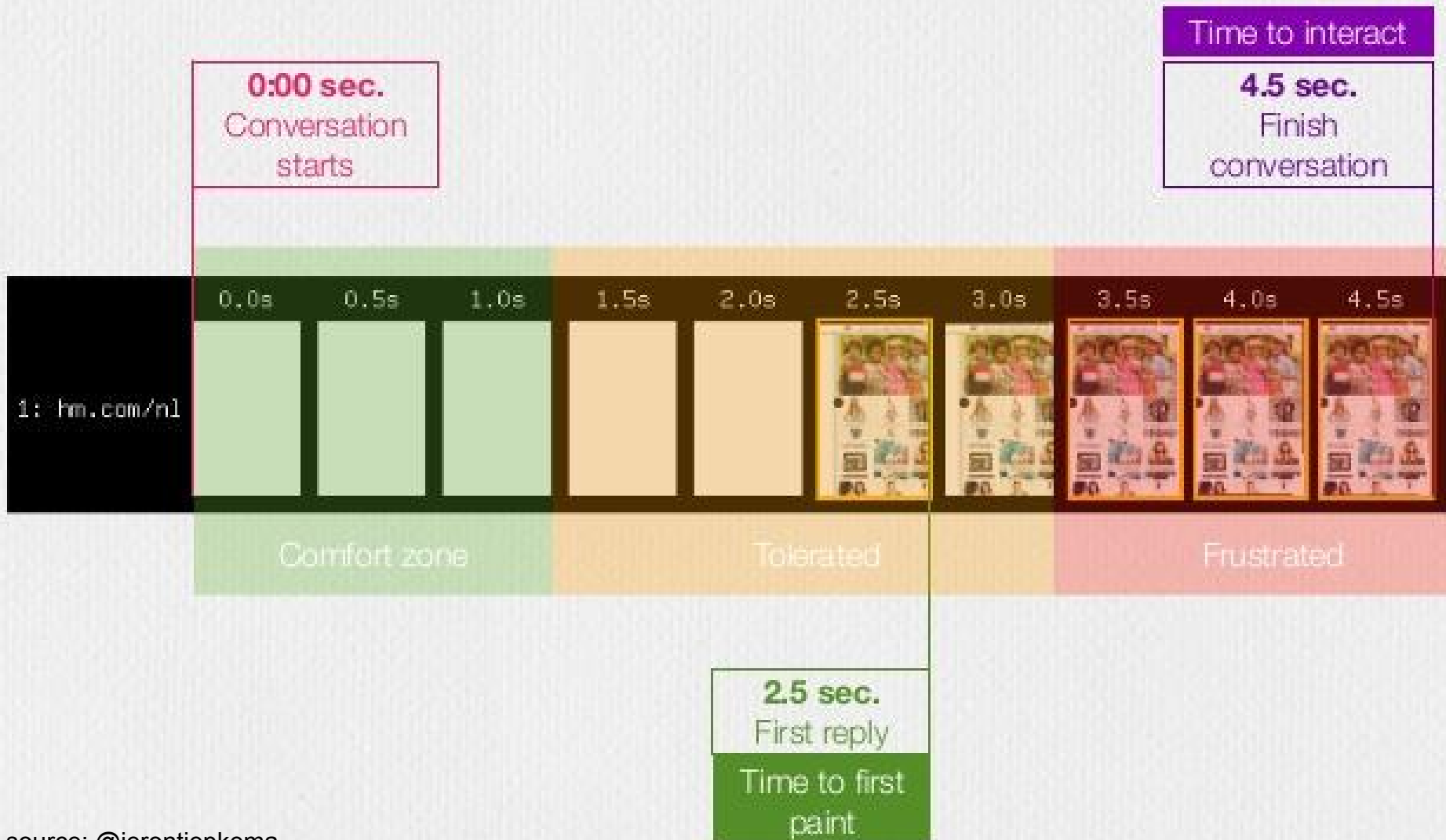
First paint should be a **styled version without JS** of your website.

It should be **functional**, which is especially important on slow/unstable connections and old devices. **Impacts UX!**

The fastest first paint would be a flash of unstyled content, if CSS is placed at the end of the page.

If the browser has the page title, and shows white screen for seconds long, you have work to do.

# The path to first paint



source: @jerontjepkema

<http://www.slideshare.net/MeasureWorks/measureworks-why-people-hate-to-wait-for-...>

# The path to first paint

How to avoid delaying the first paint

1. Do **NOT** put external JS in the header.

# The path to first paint

## How to avoid delaying the first paint

1. Do **NOT** put external JS in the header.
2. Prioritize delivery of critical content & assets.

# The path to first paint

## How to avoid delaying the first paint

1. Do **NOT** put external JS in the header.
2. Prioritize delivery of critical content & assets.
3. Minimize the *number* of assets to download (reduces latency impact).

# The path to first paint

## How to avoid delaying the first paint

1. Do **NOT** put external JS in the header.
2. Prioritize delivery of critical content & assets.
3. Minimize the *number* of assets to download (reduces latency impact).
4. Minimize the *size* of assets to download.



# The path to first paint

## How to avoid delaying the first paint

1. Do **NOT** put external JS in the header.
2. Prioritize delivery of critical content & assets.
3. Minimize the *number* of assets to download (reduces latency impact).
4. Minimize the *size* of assets to download.
5. Optimize DOM generation.

# The path to first paint

## How to avoid delaying the first paint

1. Do **NOT** put external JS in the header
2. Prioritize delivery of critical content & assets
3. Minimize the number of assets to download (reduces latency impact).
4. Minimize the size of assets to download.
5. Optimize DOM generation.
6. Optimize CSSOM generation.

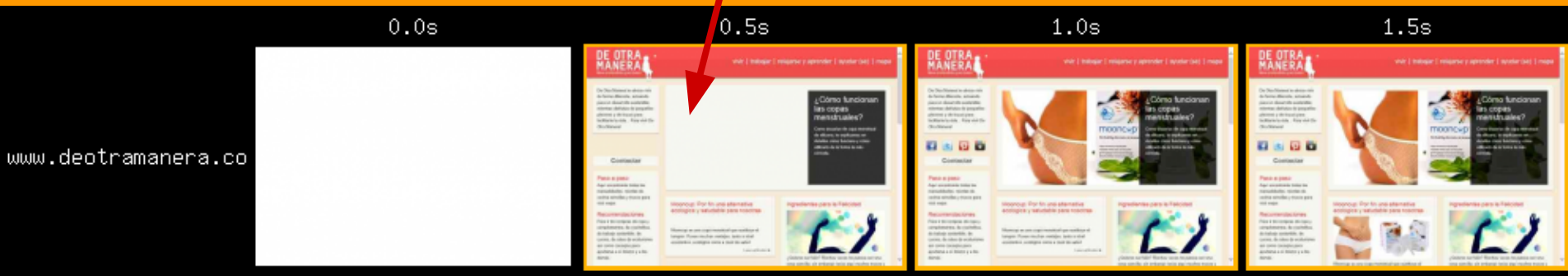
# The path to first paint

## How to avoid delaying the first paint

1. Do **NOT** put external JS in the header
2. Prioritize delivery of critical content & assets
3. Minimize the number of assets to download (reduces latency impact).
4. Minimize the size of assets to download.
5. Optimize DOM generation.
6. Optimize CSSOM generation.
7. Put Ads and other 3rd party nastiness as low in the source code as possible.

# The path to first paint

Already useful / useable



View full test: [http://www.webpagetest.org/video/compare.php?tests=140515\\_0K\\_NNE-r%3A1-c%3A0&thumbSize=150&ival=100&end=full](http://www.webpagetest.org/video/compare.php?tests=140515_0K_NNE-r%3A1-c%3A0&thumbSize=150&ival=100&end=full)

Website tested: [www.deotramanera.co](http://www.deotramanera.co) (Drupal 7)

# The path to first paint

**BONUS:** rendering performance++

www.deotramanera.co



View full test: [http://www.webpagetest.org/video/compare.php?tests=140515\\_0K\\_NNE-r%3A1-c%3A0&thumbSize=150&ival=100&end=full](http://www.webpagetest.org/video/compare.php?tests=140515_0K_NNE-r%3A1-c%3A0&thumbSize=150&ival=100&end=full)

Website tested: [www.deotramanera.co](http://www.deotramanera.co) (Drupal 7)

# The path to first paint

## Some general recommendations

- First rendered frame should contain element positioning and dimensions.
- Style with disabled JS, as if it was ready to be used (Bonus: Your site will not crumble when your JS breaks.) Use [NoScript](#) or similar.
- The browser uses your CSS, not your SASS file. Try not to nest too much.

# The path to first paint

## Some general recommendations

- First rendered frame should contain element positioning and dimensions.
- Style with disabled JS, as if it was ready to be used (Bonus: Your site will not crumble when your JS breaks.) Use [NoScript](#) or similar.
- The browser uses your CSS, not your SASS file. Try not to nest too much.

## Dangers to our render times outside our control:

- Ads - Get them out of the first paint path. They should not hurt UX.
- Bad content writing habits (too many iframes, embedded crap) - educate your content creators and/or remove the tags through input filters.

# Rendering performance



# Rendering performance

*The ability for the browser to render composited images fast enough so that it can actually give you at least 25 fps while using the site.*

You put effort into first paint, do not throw it all away after the fact.

Keep your site snappy to use!

# Rendering performance

*The ability for the browser to render composited images fast enough so that it can actually give you at least 25 fps while using the site.*

You put effort into first paint, do not throw it all away after the fact.

Keep your site snappy to use!

## **What hurts the render pipeline?**

- Things that invalidate the DOM

# Rendering performance

*The ability for the browser to render composited images fast enough so that it can actually give you at least 25 fps while using the site.*

You put effort into first paint, do not throw it all away after the fact.

Keep your site snappy to use!

## **What hurts the render pipeline?**

- Things that invalidate the DOM
- Things that invalidate the CSSOM

# Rendering performance

*The ability for the browser to render composited images fast enough so that it can actually give you at least 25 fps while using the site.*

You put effort into first paint, do not throw it all away after the fact.

Keep your site snappy to use!

## **What hurts the render pipeline?**

- Things that invalidate the DOM
- Things that invalidate the CSSOM
- JS animations (use `requestAnimationFrame`, not `jQuery.animate`)
- Flash
- Ads
- ...

# Rendering performance

## Things that hurt the render pipeline (in-app):

- Adding/removing/changing HTML elements.
- Adding/removing CSS classes, adding inline styles.
- Showing / hiding elements.

These cause the browser to **invalidate the render tree / layout**. It means doing a bunch of expensive things, and if the recalculation gets big and/or gets very frequent, you could lose frames, which results in shuttering etc.

If possible, provide the markup for your JS goodies in the original source code.

**If you get the first paint right,  
you have most of the job done,  
don't mess things up!**

**Drupal**

# Drupal

## Getting Drupal (7) to render fast

### Minifying and aggregation of CSS and JS:

- [Advanced CSS/JS Aggregation](#) (more features)
- [Speedy](#) (minified versions of core JavaScript)
- [Simple aggregation](#) (reduces the number of aggregated files)


# Drupal

## Getting Drupal (7) to render fast

### Minifying and aggregation of CSS and JS:

- [Advanced CSS/JS Aggregation](#) (more features)
- [Speedy](#) (minified versions of core JavaScript)
- [Simple aggregation](#) (reduces the number of aggregated files)

### Reduce the number of DOM elements:

- [Fences](#) (leaner markup for fields)
  - [Entity view modes](#)
  - [Display Suite](#)
  - Optimize your page.tpl, panels tpls, etc
- Use the minimum number of elements necessary
- 



# Drupal

## Getting Drupal (7) to render fast

### Reduce the amount of CSS

```
/**
 * Implement hook_css_alter().
 */
function MYTHEME_css_alter(&$css) {
  // Remove a single css file.
  unset($css[drupal_get_path('module', 'system') . '/defaults.css']);
}
```

# Drupal

## Getting Drupal (7) to render fast

### Reduce the amount of CSS

```
/**
 * Implement hook_css_alter().
 */
function MYTHEME_css_alter(&$css) {
  // remove all core css files
  foreach ($css as $key => $file) {
    if (preg_match('/^modules/', $key)) {
      unset($css[$key]);
    }
  }
}
```

# Drupal

## Getting Drupal (7) to render fast

### Reduce the amount of CSS

```
/**
 * Implement hook_css_alter().
 */
function MYTHEME_css_alter(&$css) {
  // Remove all but my theme's css files
  $theme_path = drupal_get_path('theme', 'MYTHEME');
  $string_match = '/^'. str_replace('/', '\\/', $theme_path) .'/';

  foreach ($css as $key => $file) {
    if (!preg_match($string_match, $key)) {
      unset($css[$key]);
    }
  }
}
```

# Drupal

Getting Drupal (7) to render fast

**async** Javascript (D7, done in D8!):

**Needs backport: [#1140356](#) OR [#1664602](#)**

# Drupal

Getting Drupal (7) to render fast

**async** Javascript (D7, done in D8!):

**Needs backport: [#1140356](#) OR [#1664602](#)**

- [advagg](#) 7.x-2.6 does it without need to patch core
- [aloha](#) does it in template.php (a bit nasty)

# Drupal

## Getting Drupal (7) to render fast

**async** Javascript (D7, done in D8!):

aloha does it in `hook_process_html()`:

```
function aloha_process_html(&$variables) {
  if (strpos($variables['scripts'], '/lib/aloha.js') !== FALSE) {
    $variables['scripts'] = preg_replace('/(\/lib\/aloha\.js[^\"]*["])
/', '$1 data-aloha-defer-init="true"', $variables['scripts'], 1);
  }
}
```

# Drupal

## Getting Drupal (7) to render fast

### Moving all JS to the footer:

```
/**
 * Implements hook_js_alter().
 */
function MYTHEME_js_alter(&&javascript) {
  // Move all JS to the footer.
  foreach ($javascript as $name => $script) {
    $javascript[$name]['scope'] = 'footer';
  }
  // Forces Modernizr to header if the module is enabled.
  if (module_exists('modernizer')) {
    $javascript[modernizer_get_path()]['scope'] = 'header';
  }
}
```

# Drupal

Getting Drupal (7) to render fast

**Getting rid of ALL Javascript:**

```
/**
 * Implements hook_js_alter().
 */
function MYTHEME_js_alter(&$javascript) {
  // Remove all JS
  $javascript = array(); // 0_o
}
```



# Drupal

And what about Drupal 8? (pull from 16-mar-2014)

**Out of the box, front page:**

# Drupal

And what about Drupal 8? (pull from 16-mar-2014)

**Out of the box, front page:**

- CSS & JS aggregation on by default - WIN!

# Drupal

And what about Drupal 8? (pull from 16-mar-2014)

**Out of the box, front page:**

- CSS & JS aggregation on by default - WIN!
- 4 CSS files for anonymous, 7 for admin. CSS in <head>.

# Drupal

And what about Drupal 8? (pull from 16-mar-2014)

**Out of the box, front page:**

- CSS & JS aggregation on by default - WIN!
- 4 CSS files for anonymous, 7 for admin. CSS in <head>.
- JS is in <head> & not async by default :(

# Drupal

And what about Drupal 8? (pull from 16-mar-2014)

**Out of the box, front page:**

- CSS & JS aggregation on by default - WIN!
- 4 CSS files for anonymous, 7 for admin. CSS in <head>.
- JS is in <head> & not async by default :(
- First paint at ~400ms for admin, ~200ms for anonymous.

# Drupal

And what about Drupal 8? (pull from 16-mar-2014)

**Out of the box, front page:**

- CSS & JS aggregation on by default - WIN!
- 4 CSS files for anonymous, 7 for admin. CSS in <head>.
- JS is in <head> & not async by default :(
- First paint at ~400ms for admin, ~200ms for anonymous.
  - With JS at the bottom, first paint as admin goes down to ~200ms!

# Drupal

And what about Drupal 8? (pull from 16-mar-2014)

## Out of the box, front page:

- CSS & JS aggregation on by default - WIN!
- 4 CSS files for anonymous, 7 for admin. CSS in <head>.
- JS is in <head> & not async by default :(
- First paint at ~400ms for admin, ~200ms for anonymous.
  - With JS at the bottom, first paint as admin goes down to ~200ms!
- In-app paint time is 2-5ms for anonymous, 4-6ms for admin - pretty good, but its an empty page ;)

# Drupal

And what about **Drupal 8**?

It is a **lot** better than 7.x

We only provide the JS *needed* for each page - WIN!

Work's still being done - tag: [frontend performance](#)

## Get involved:

- [Meta] selectors clean-up [#1574470](#)
- jQuery and Drupal JavaScript libraries and settings are output even when no JS is added to the page [#1279226](#)
- [META] Improving CSS and JS preprocessing [#1490312](#)



# Tools & Resources

## Tools

- Webpagetest: <http://www.webpagetest.org/>
- Firefox & Chrome dev tools
- Google PageSpeed insights: <http://developers.google.com/speed/pagespeed/insights/>

## Further reading, sources and resources

- Performance profiling with the Timeline: <https://developer.chrome.com/devtools/docs/timeline>
- <https://developers.google.com/speed/>
- <http://www.stevesouders.com/blog/>
- <https://www.igvita.com/>
- On Layout & Web Performance <http://www.kellegous.com/j/2013/01/26/layout-performance/>
- Writing efficient CSS [https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Writing\\_efficient\\_CSS](https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Writing_efficient_CSS)
- Best Practices for Speeding Up Your Web Site <https://developer.yahoo.com/performance/rules.html>
- Profiling CSS for fun and profit <http://perfectionkills.com/profiling-css-for-fun-and-profit-optimization-notes/>



**drupalcamp**SPAIN  
**VALENCIA · 2014**

**Thanks!**

Manuel Garcia  
[d.org/user/213194](https://d.org/user/213194)