How the browser actually renders a website

Ryan Seddon

Hallo Berlin!

@ryanseddon

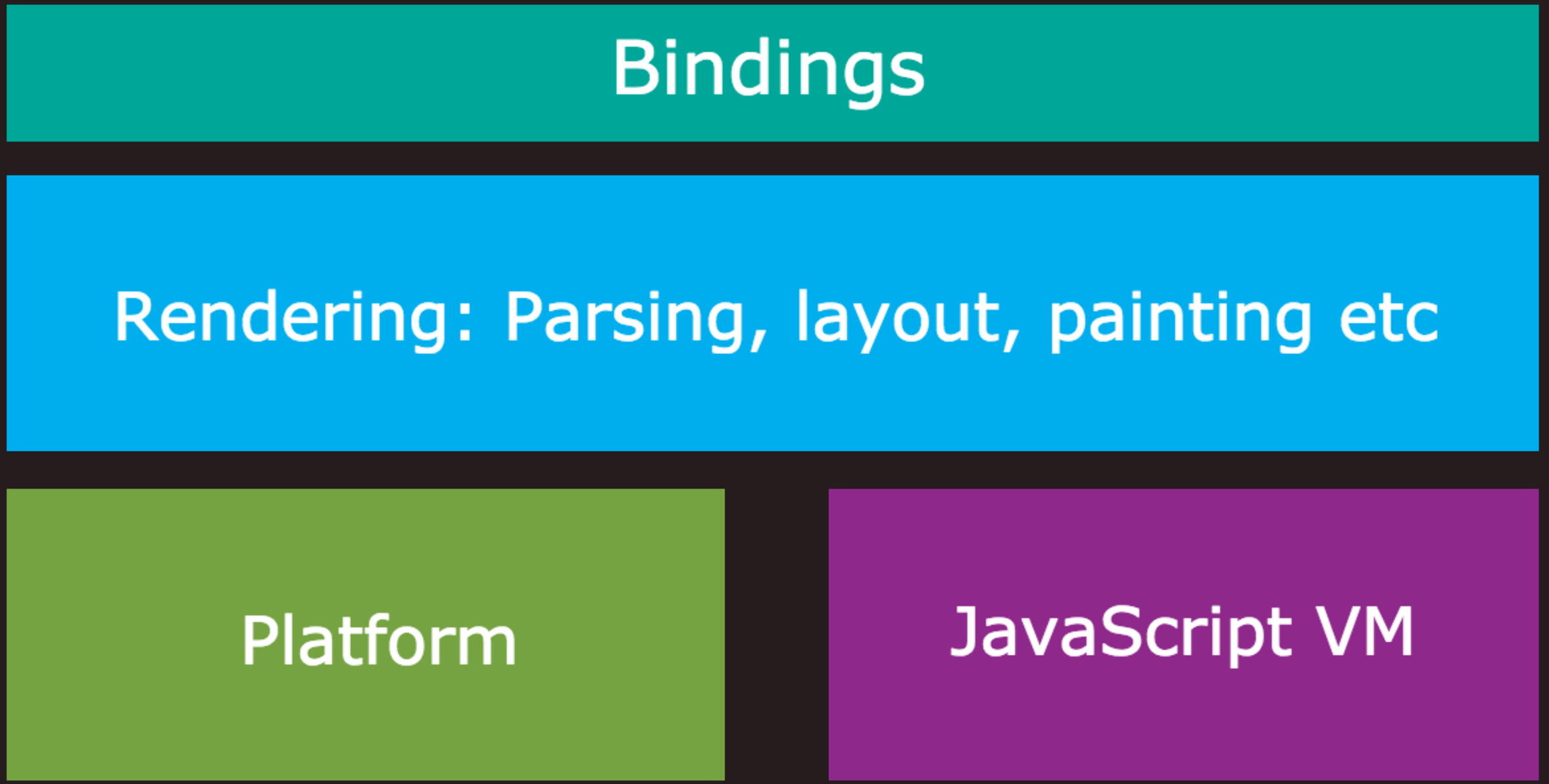
Team Lead @ Zendesk

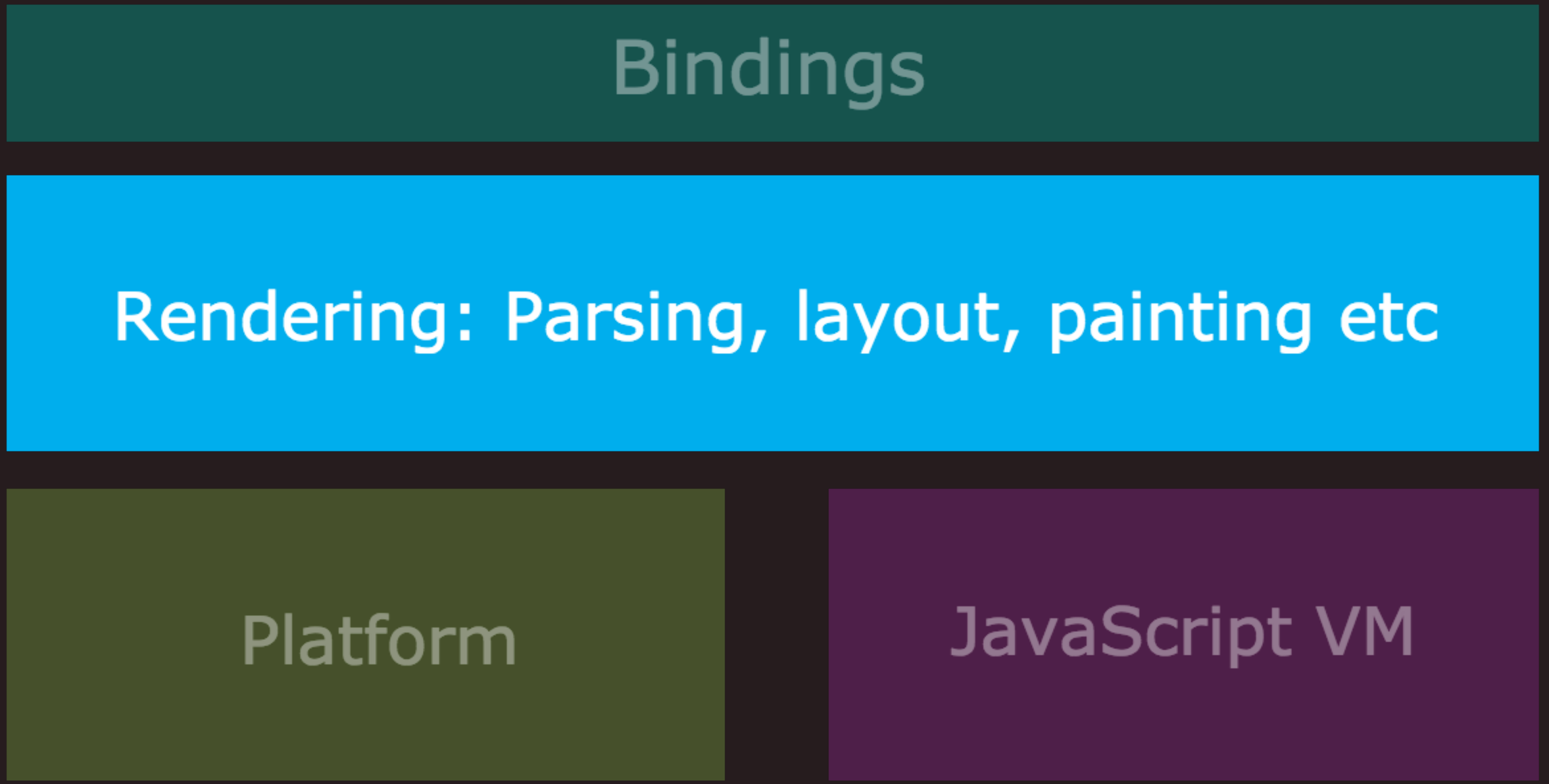
What we'll cover

- High level view
- In-depth view
- Performance insights

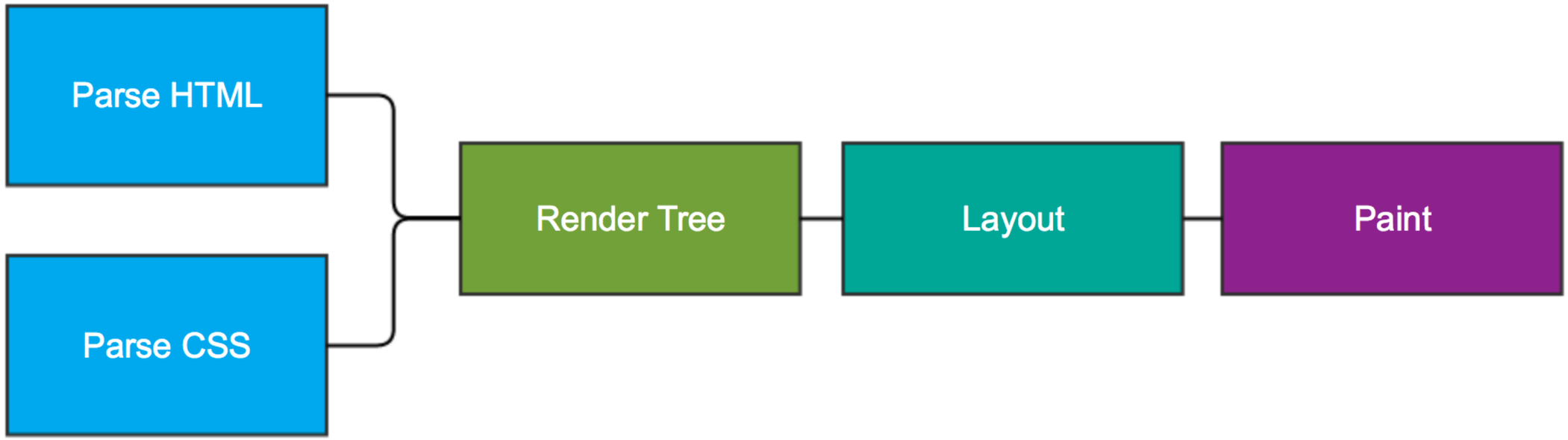
30,000ft view

So the browser...


Bindings
Rendering: Parsing, layout, painting etc
Platform
JavaScript VM


Bindings
Rendering: Parsing, layout, painting etc
Platform
JavaScript VM

High level flow

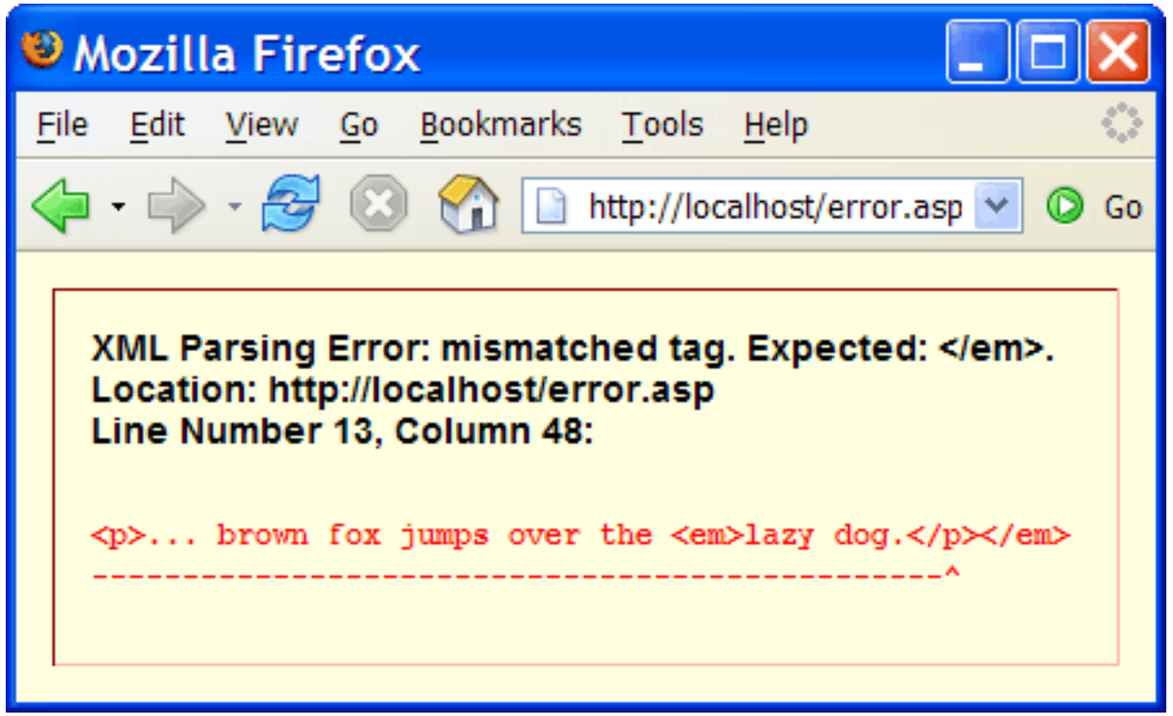


1. Parsing

Parsing HTML

- HTML is forgiving by nature
- Parsing isn't straight forward
- Can be halted
- Will do speculative parsing
- It's reentrant.

Remember xhtml strict



JavaScript and HTML: Forgiveness by Default

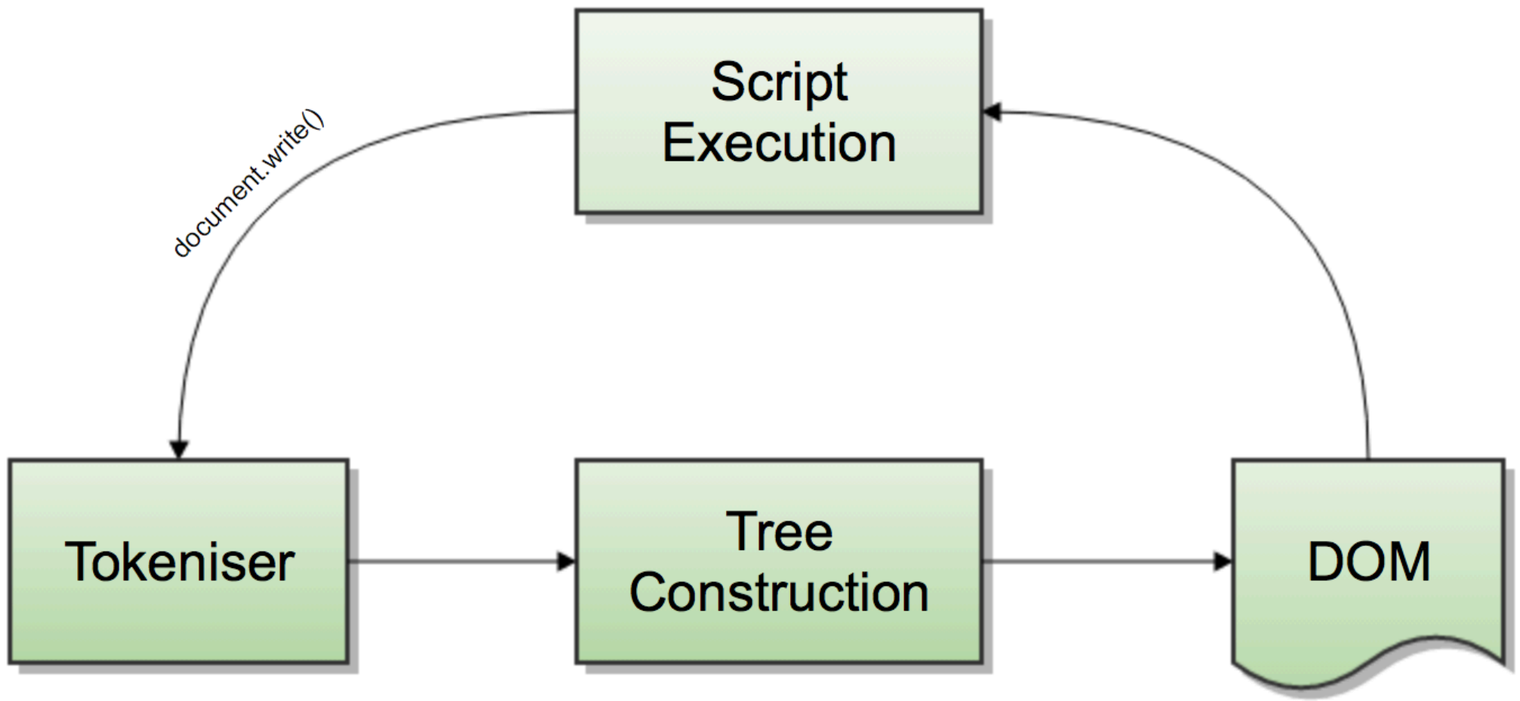
Valid HTML5

```
<body>
<p class=wat>My first website
<div><span>Visitor count: 0
```

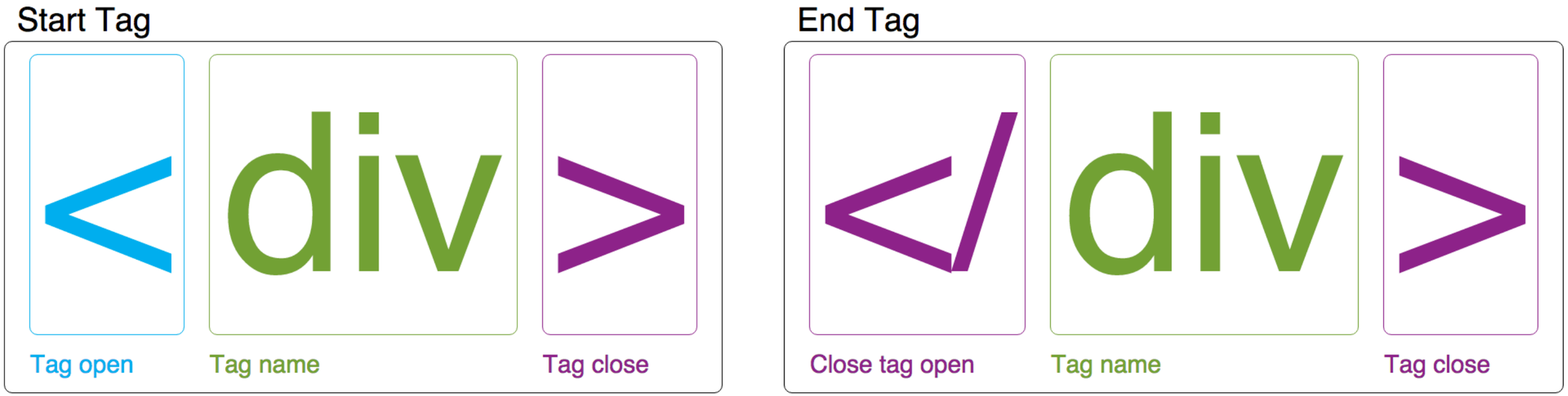
Would output

```
<html>
  <head></head>
  <body>
    <p class="wat">
      My first website
    </p>
    <div>
      <span>
        Visitor count: 0
      </span>
    </div>
  </body>
</html>
```

Parsing flow



Tokenizer


Start Tag
<
div
>
Tag open
Tag name
Tag close
End Tag
</
div
>
Close tag open
Tag name
Tag close

Parse Tree

```
html
|-- head
`-- body
    |-- p.wat
    |   `-- #text
    `-- div
        `-- span
            `-- #text
```

DOM Tree

```
HTMLHtmlElement
|-- HTMLHeadElement
`-- HTMLBodyElement
    |-- HTMLParagraphElement
    |   `-- Text
    `-- HTMLDivElement
        `-- HTMLSpanElement
            `-- Text
```

<script>, <link> & <style>

Will halt the parser as a script can alter the document.

- Network latency
- link & style could halt JS execution

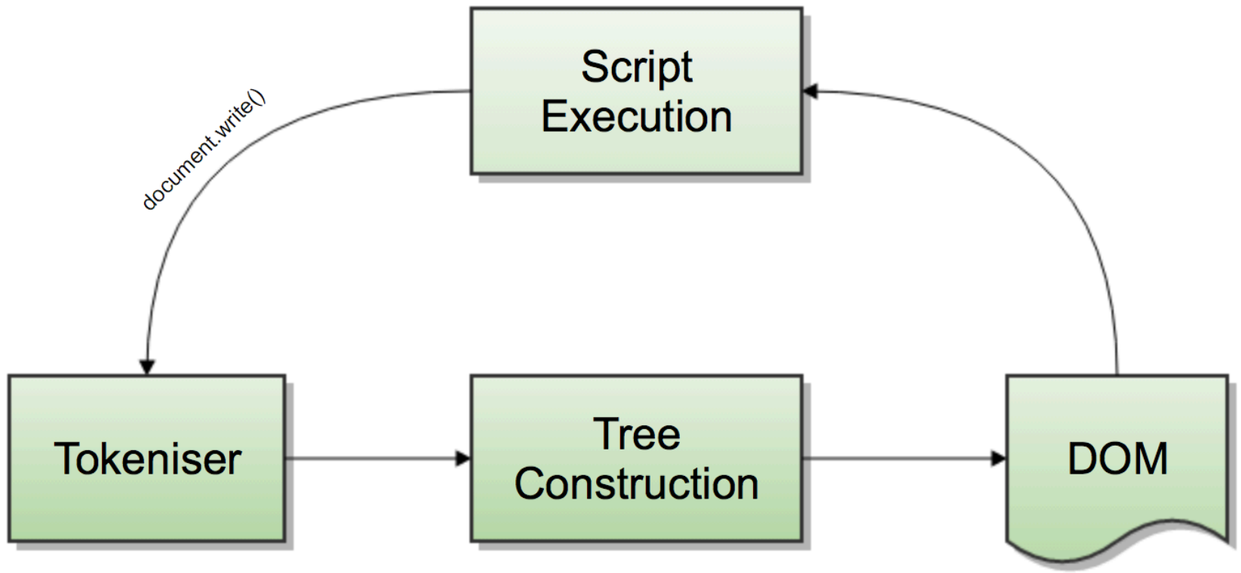
Speculative parsing

- Will look ahead
- External images, scripts, css

```
<script src='script.js'>
//....
<img src='cat.gif' />
<link href='styles.css' />
```

Reentrant

Means the parsing process can be interrupted



Performance insight

1

<script /> at the bottom

- Parse uninterrupted
- Faster to render
- defer and async attributes
- Trade off

Parsing a HTML document

Visualised

CSS parsing

CSSOM

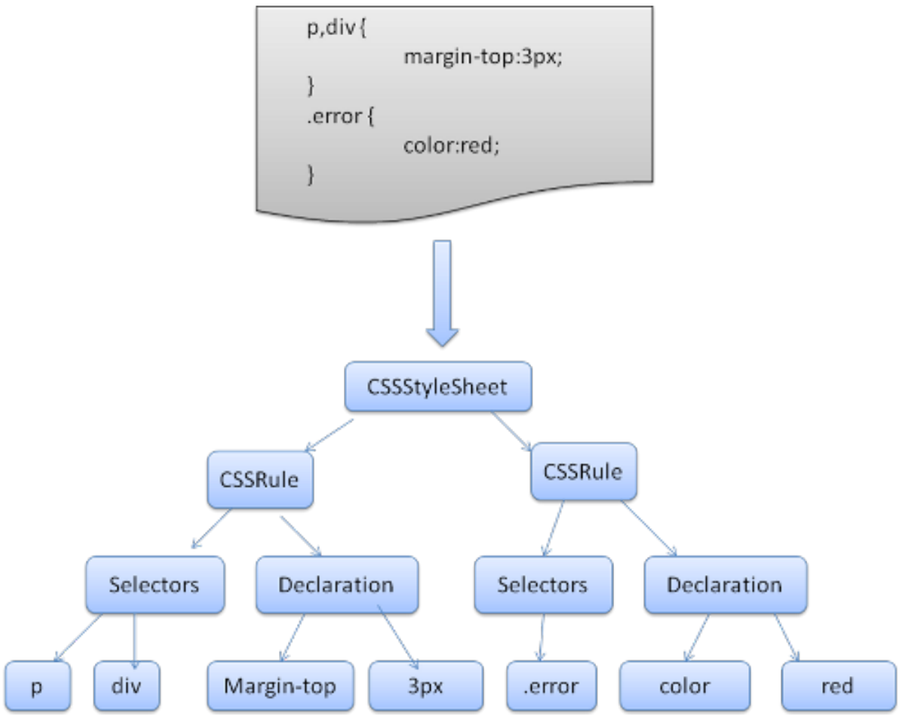
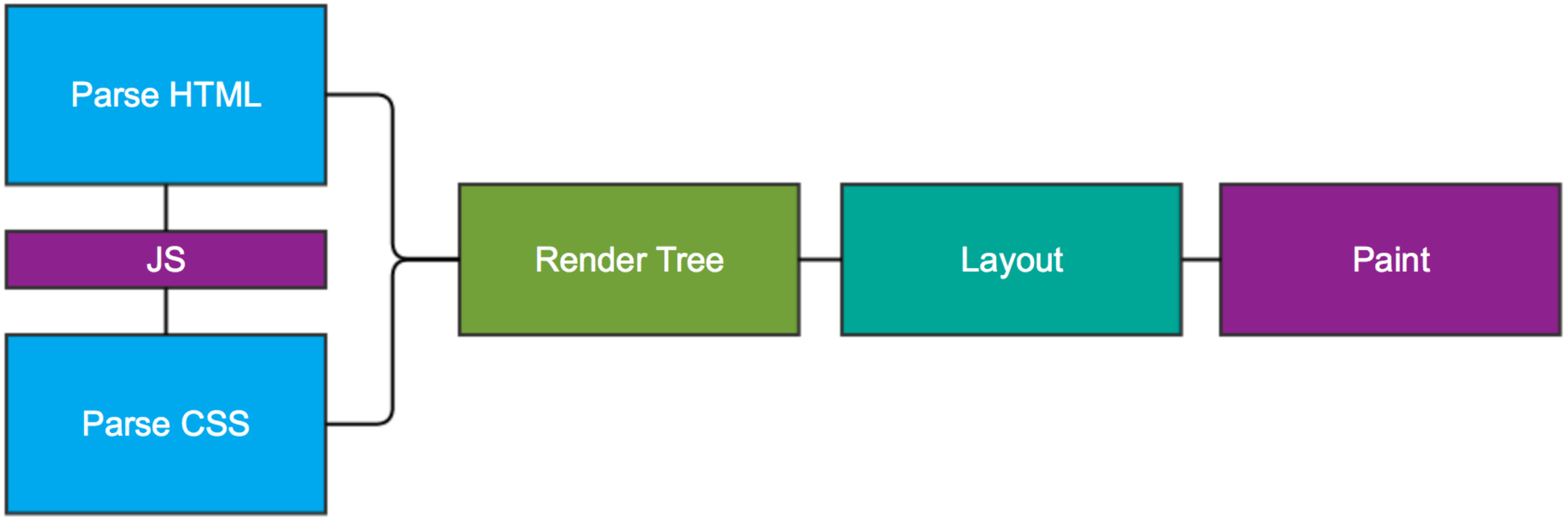


image source

2. Render/Frame tree

DOM + CSSOM

- Combines the two object models, style resolution
- This is the actual representation of what will show on screen
- Not a 1-to-1 mapping of your HTML


Parse HTML
JS
Parse CSS
Render Tree
Layout
Paint

Multiple trees

- RenderObjects
- RenderStyles
- RenderLayers
- Line boxes

Not in the render tree

- Non-visual elements head, script, title etc
- Nodes hidden via display: none;

In the render tree, not in the DOM

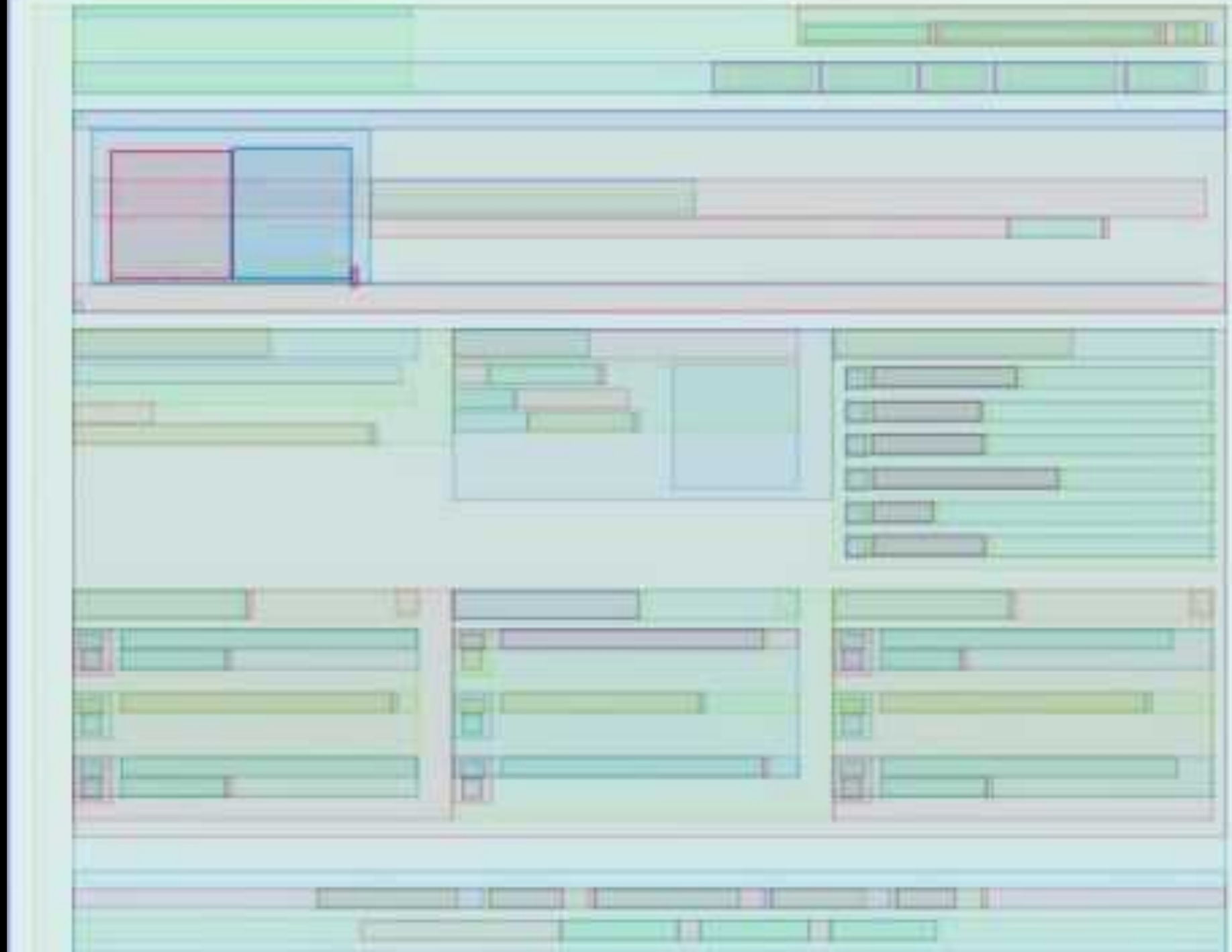
```
<p>The quick brown <strong>fox</strong>
<em>jumps<em> over the lazy dog</p>
```

```
RenderBlock(p)
|-- RenderText('The quick brown')
|-- RenderInline(strong)
|   `-- RenderText('fox')
|-- RenderInline(em)
|   `-- RenderText('jumps')
`-- RenderText('over the lazy dog')
```

```
RenderBlock(p)
|-- RootLineBox (line 1)
|   |-- InlineBox (text)
|   `-- InlineBox (strong)
|       `-- InlineBox (text)
`-- RootLineBox (line 2)
    |-- InlineBox (em)
    |   `--InlineBox (text)
    `-- InlineBox (text)
```

DOM Node to RenderObject

- Visual output
- Geometric info
- Can layout and paint
- Holds style & computed metrics



Calculating visual properties

- Combines all styles
- defaults, external, style elements & inline
- Complexity around matching rules for each element
- Style computation

3. Layout

Recursive process

- Traverse render tree
- Nodes position and size
- Layout its children

Will batch layouts

- Incremental layouts
- The browser will intelligently batch changes
- Render tree items will flag themselves as dirty
- The batch will traverse the tree and find all dirty trees
- Asynchronous

Immediate layout

- Doing a font-size change will relayout the entire document
- Same with browser resize
- Accessing certain properties via JavaScript e.g. node.offsetHeight

Performance insight

2

Take note from the browser and batch

- Act like the browser and batch your DOM changes
- Do all your reads in one pass
- Followed by writes

Bad

Here we read then write, read then write.

```
var divHeight = div.clientWidth / 1.7;
div.style.height = divHeight + 'px';

var div2Height = div2.clientWidth / 1.7;
div2.style.height = div2Height + 'px';
```

Good

```
var divHeight = div.clientWidth / 1.7;
var div2Height = div2.clientWidth / 1.7;

div.style.height = divHeight + 'px';
div2.style.height = div2Height + 'px';
```

Real world

- FastDom, Preventing layout thrashing
- Most modern JS frameworks do this internally

4. Paint

Paint setup

- Will take the layed out render trees
- Creates layers
- Incremental process
- Builds up over 12 phases

RenderLayers

- Creates layers from RenderObjects
- Position nodes, transparency, overflow, canvas, video etc
- Many-to-1 relationship a RenderLayer could contain multiple RenderObjects

Painting

- Produces a bitmap from each layer
- Bitmap is uploaded to the GPU as a texture
- Composites the textures into a final image to render to the screen

Performance insight

3

inline critical CSS

- The most important bits of your site/app
- Speeds up first paint times
- External js and css can block
- Delta last bitmap

July 22nd, 2014

UITableView in JavaScript, list view with re-usable cells using flexbox

If you're familiar with iOS development you will know that a UITableView is very efficient when displaying a list of data. A simplification of what it does is display enough cells to fill the viewport plus a few more either side. As you scroll it re-uses cells that are now out of the viewport so a list with thousands of items will only ever use a fixed amount of cells. Highly recommend reading The fine art of UITableViews. Now this has certainly been done before in JavaScript, the best known project being infinity.js, but my approach takes an interesting turn, I avoid heavy DOM operations by using flexbox.

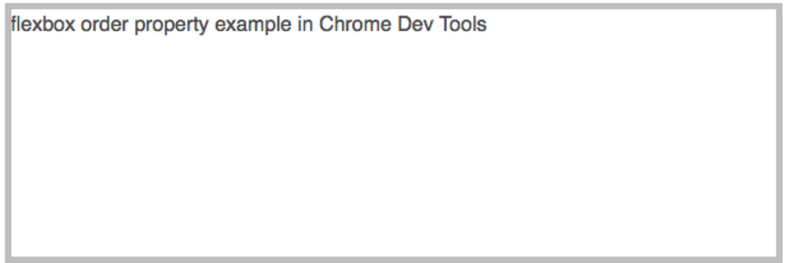
How does flexbox help?

If you're familiar with flexbox you may have come across the `order` property which allows you to reorder flex items in a flexbox container and this is the magic that allows us to create reusable list items without having to actually rip the element out of the DOM and then re-inject it in the right place as a user scrolls.

Check out the project on github and have a look at the live demo

How does it work

The whole technique boils down to the flexbox `order` property which allows reordering of a flex item within a flexbox container, essentially I can say to the browser repaint the DOM node here instead of where it is in the actual DOM structure.



THE CSS NINJA

July 22nd, 2014

UITableView in JavaScript, list view with re-usable cells using flexbox

If you're familiar with iOS development you will know that a UITableView is very efficient when displaying a list of data. A simplification of what it does is display enough cells to fill the viewport plus a few more either side. As you scroll it re-uses cells that are now out of the viewport so a list with thousands of items will only ever use a fixed amount of cells. Highly recommend reading The fine art of UITableViews. Now this has certainly been done before in JavaScript, the best known project being infinity.js, but my approach takes an interesting turn, I avoid heavy DOM operations by using flexbox.

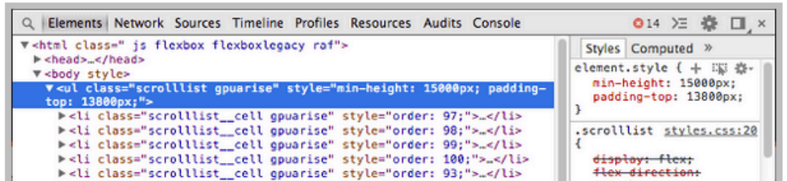
How does flexbox help?

If you're familiar with flexbox you may have come across the `order` property which allows you to reorder flex items in a flexbox container and this is the magic that allows us to create reusable list items without having to actually rip the element out of the DOM and then re-inject it in the right place as a user scrolls.

Check out the project on github and have a look at the live demo

How does it work

The whole technique boils down to the flexbox `order` property which allows reordering of a flex item within a flexbox container, essentially I can say to the browser repaint the DOM node here instead of where it is in the actual DOM structure.



Popular Posts

- How to create offline webapps on the iPhone
- Custom radio and checkbox inputs using CSS
- Pure CSS collapsible tree menu
- Create the accordion effect using CSS3
- Accessing the GPS in iPhone Safari
- Futurebox, lightbox without the javascript and target pseudo-class

Categories

- angularjs
- css
- hacking
- html
- html5
- javascript
- mobile
- rant
- xhtml

All of these steps

Can apply after page load

Recap

- Parsing --> DOM Tree
- DOM Tree --> Render Tree
- Is actually 4 trees
- Layout computes where a Node will be on the screen
- Painting computes bitmaps and composites to screen

The browser is complicated

- http://www.webkit.org/projects/layout/index.html
- http://limpet.net/mbrubeck/2014/08/08/toy-layout-engine-1.html
- http://www.html5rocks.com/en/tutorials/internals/howbrowserswork/
- https://www.youtube.com/watch?gl=US&v=RVnARGhhs9w
- https://www.chromium.org/developers/design-documents/gpu-accelerated-compositing-in-chrome
- https://www.youtube.com/watch?v=LpkIdYdo62o

Go hug a browser engineer

They have a hard job

Thanks