**1** welcome to deep dive

**2** an exploration

**3** of a small subsection of why stuff breaks sometimes – the browser.

**4** (mom) whats the browser

Asked my mom   
(**5**) – whats that pinwheel in the corner? she said Netflix or Google – you get at the rest of the internet through google

Asked my sister   
(**6**), she said, search bar, back button,

(**7**) this is what my sister was talking about, that’s great

(**8**) this is a browser, with UI along the top, and   
(**9**)content, brought to you by rendering engine

(**10**) fun fact, UI is known as chrome.

More stuff – networking layer, JS engine, data persistence layer (cookies)… we’re just going to talk about   
(**11**) rendering engine and   
(**12**) how that relates to the webpage.

KINDS of renderers

**(13-15)** Its good to start off with that there’s a few – see the list

You probably already know this if you’ve assumed if it works for you, it’ll work for everyone – if people are using a browser with a rendering engine incapable of understanding whats there, it won’t treat it right.

**(16)** Caniuse.com

GENERALLY what the renderer does

Lets look at rendering engine. It does two things:

1. **(17)** parse html
2. **(18)** paint it on the screen

Lets move through it by asking what happens when you link.   
**(19)**

1. the new resource is loaded. Browser sends a   
   GET**(20,21),** server returns an index.html**.  
   (23)**
2. HTML is parsed. But this is a SPECIAL PARSER… for background knowledge, parsing is well formulated by now, you can define grammar rules and pass it to a parser-generator and it will be able to parse how you dictate.   
   **(24)** CSV files for example – look for commas, look for strings in quotes, look for new lines. HTML can NOT   
   **(25)** be parsed by a regular parser   
   **(26)**, beause…

**(27)** whats wrong with this?   
**(28)**Right - The syntax is flexible. will still go through. NOT perfect - leaving a stray html tag unclosed might allow the browser to close it in a weird spot, meaning styles leak into other parents.

* 1. Also, **(29)** JS is king  
     **(30)**. You can write and delete and mess with the dom as you see fit. Think of it as adding source code during a compile. Fancy word – re-entrant
  2. **(31)**Tokenizer -> parser back and forth. Tokenizer to words, words parsed to tree.

As it parses, if it hits a script or a stylesheet, it will   
STOP **(32)** parsing DOM and read   
**(33)** stylesheets or whatever instead. The synchronous model of the web. Webkit and Firefox optimize, while executing scripts, another thread will crawl the doc and see if it can start making any network calls. This thread will not modify the dom tree though. You say “this is crappy!” but really this is good. Stylesheets and JS may be interacting, they probably are, and they need the dom in place to give them a battleground.

1. **(34)**Dom tree is made. What happens when this hits the end? It’ll start scripts with “deferred”, this is when the notorious LOAD event is fired…
2. DOM and styles unite to form RENDER TREE. Its not a one-to-one -
   1. Display none wont be there.
   2. <head> wont be there.
   3. Position: absolute and position: relative wont.

We get our BOXES (remember, box model…)

But we don’t know where the boxes go – layout. Position and dimensions. This is where the nature of parent/child and display properties really start to come into play, eg  
**(35)**

Height – parent asks the children, how tall r u guys

* + 1. Width **(36)**– parent decides, and tells childen ok u r this wide

1. Final – PAINT it! Traverses render tree and tells each node to paint yourself.  
   paiting follows rules but of course there are ways to break it   
   **(37)**– z-index lets you control   
   **(38)** the painting elements of nodes. Paint typically has an order…   
   **(39)** if you give an element a background color and a background image, what happens? Background image. Full order:
   1. Bg color, bg image, border, children, outline.
   2. Big ole caveat here is z-index, which does what it wants.

Your SPA is just a massive triggering of new DOM trees into new layouts into new paints. Because of this, you want this process to be as least expensive as possible.

Some changes only involve paint – changing a color.

Some changes involve a layout re-calc. Browser uses   
**(40)** dirty bits system – only changes the node and the nodes affected by it. Efficient when its low in the tree hierarchy

So how do we use this knowledge

**(41)** don’t needlessly torture the browser!

We want to AVOID unnecessary repaints and layout recalcs –   
**(42)**

Requesting a property is expensive (saying document.getElementById(‘butts’) .width()– it has to go find that value by doing a flush of the layout.)

Changing a classname is better than assigning a series of new properties – changing class name is just one repaint as opposed to many.   
**(43)** avoiding the downside of a over-burdened browser engine.

**(44)** THANKS