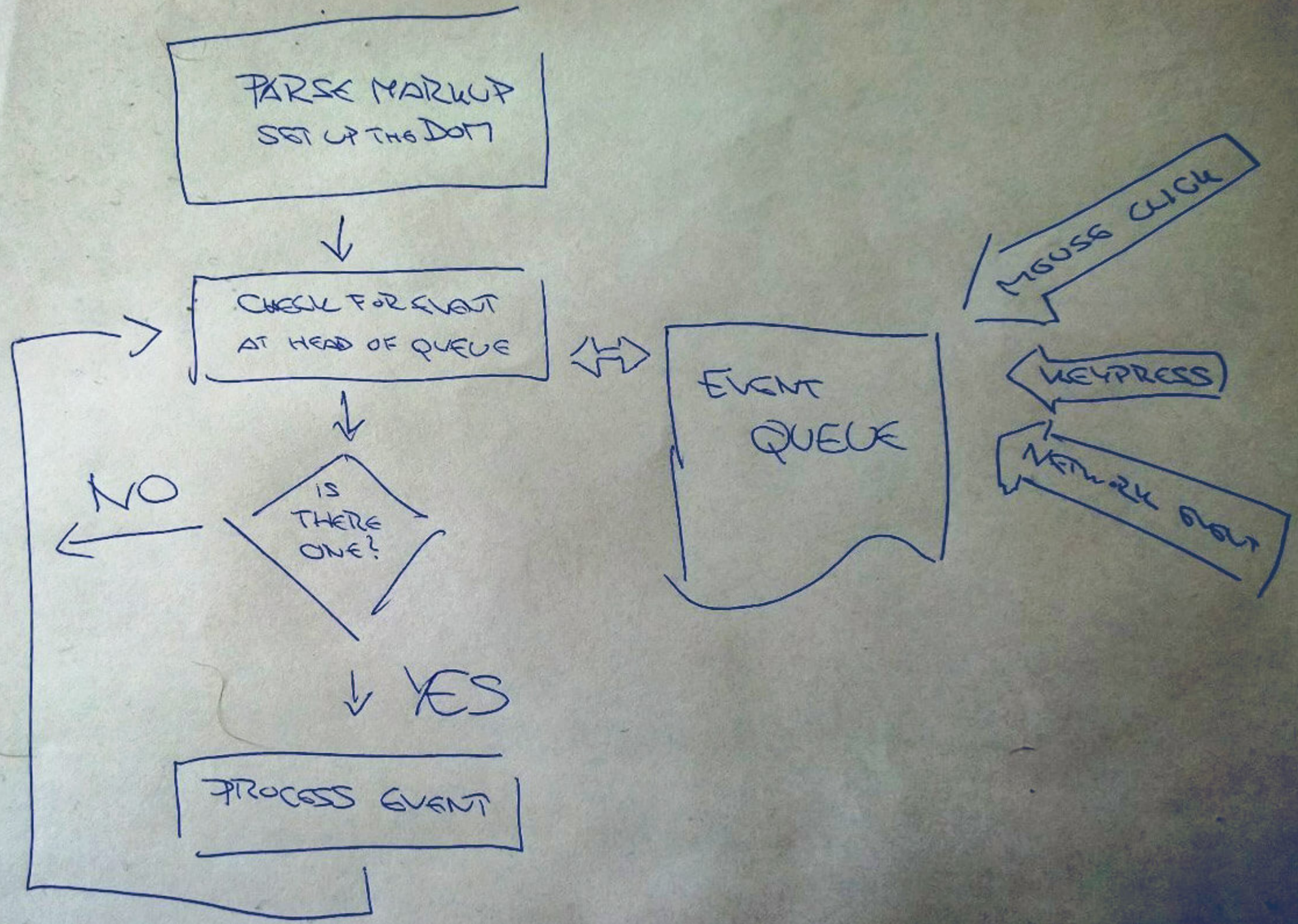

FLOW
PAINT
event
loop
SCRIPT
EVENT


PARSE MARKUP
SET UP THE DOM
CHECK FOR EVENT
AT HEAD OF QUEUE
IS
THERE
ONE?
NO
YES
PROCESS EVENT
EVENT
QUEUE
MOUSE CLICK
KEYPRESS
NETWORK EVENT

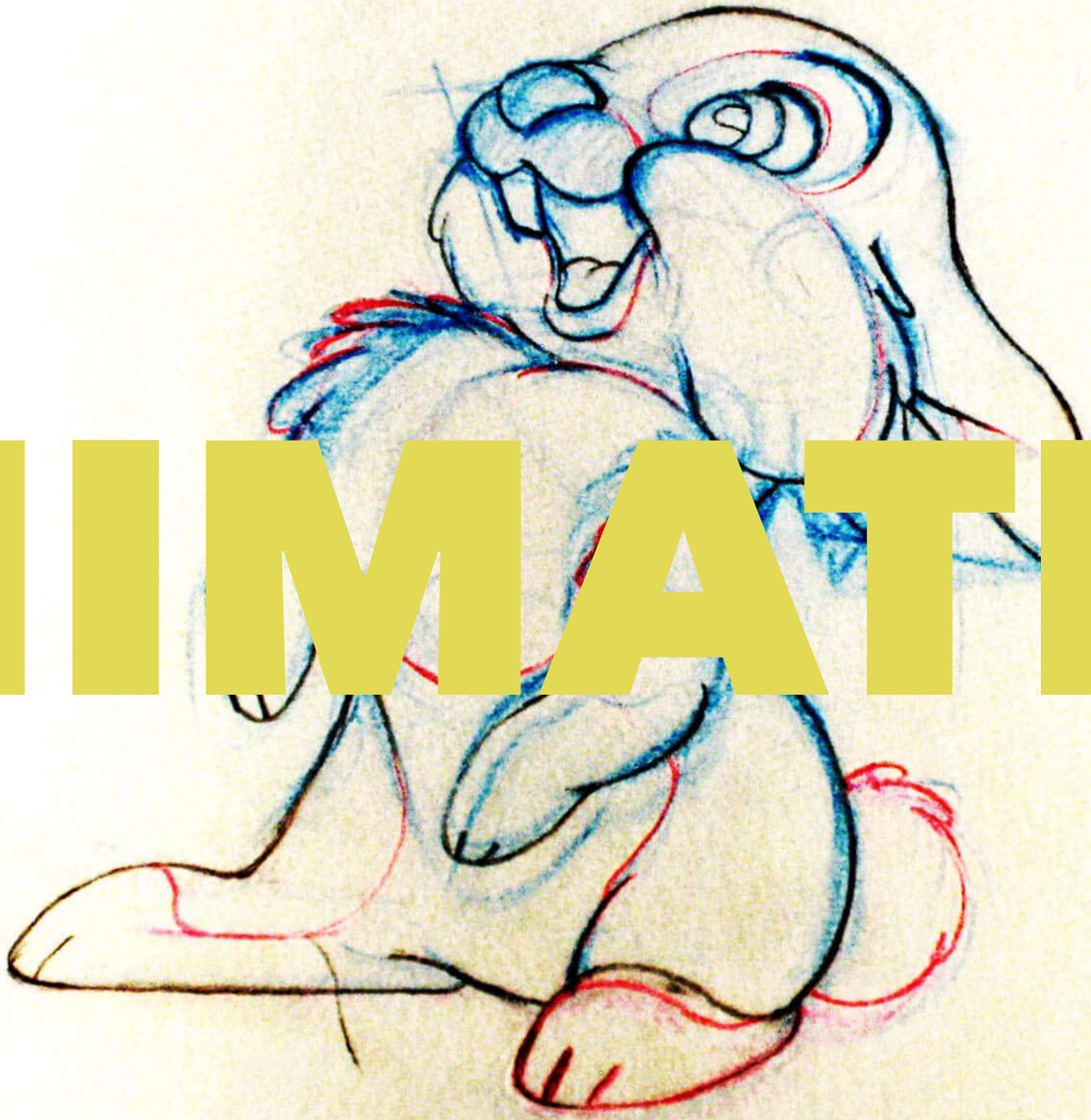
many beautiful types of events

browser events *...page finishes loading*
network events *...response to ajax request*
user events *...mouse clicks, keypresses*
timer events *...timer expires, interval fires*

events are **unpredictable,** so we handle them **asynchronously,** js doens't wait, we're **non-blocking**

on a single thread

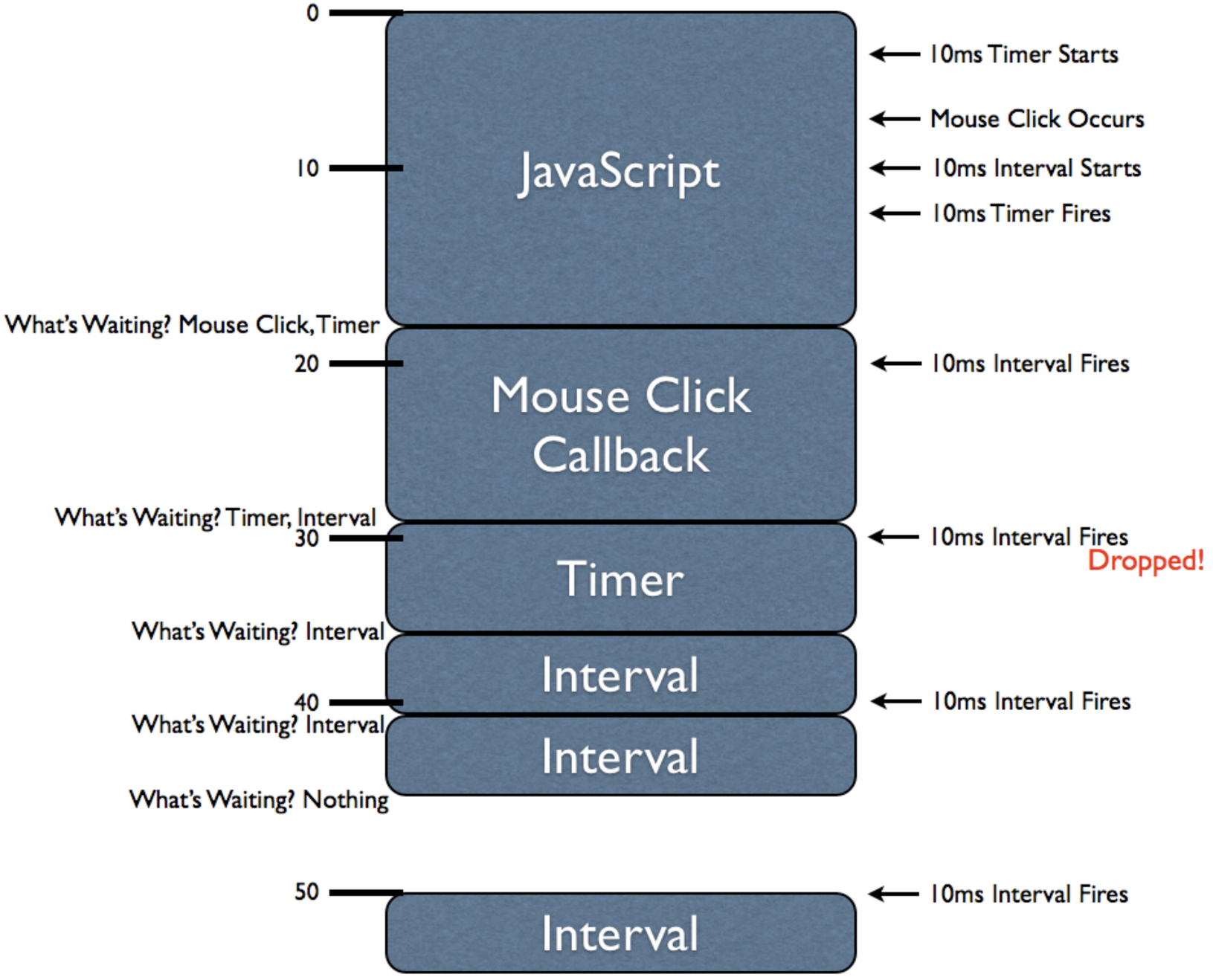
ANIMATION



setTimeout() and **setInterval()** be crazy

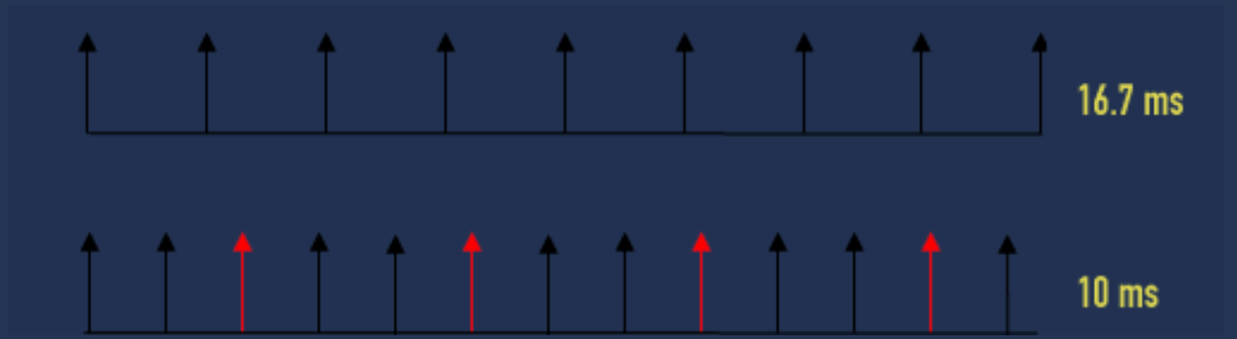
SINGLE-THREADED with events coming in
the delay **WILL NOT BE** exact
setInterval calls **MIGHT STACK**

```
// better than setInterval
function draw(){
    setTimeout(draw, 16);
    // code for this frame
}
draw();
```


0
10
20
30
40
50
JavaScript
Mouse Click Callback
Timer
Interval
Interval
Interval
10ms Timer Starts
Mouse Click Occurs
10ms Interval Starts
10ms Timer Fires
10ms Interval Fires
10ms Interval Fires
Dropped!
10ms Interval Fires
10ms Interval Fires
What's Waiting? Mouse Click, Timer
What's Waiting? Timer, Interval
What's Waiting? Interval
What's Waiting? Interval
What's Waiting? Nothing

timers can

- drop frames
- waste CPU, extra power usage
- animate elements that are not visible
- keep running in background or minimized tabs



you wanna use

requestAnimationFrame

```
function draw(){
    requestAnimationFrame(draw);
    // code for this frame
}
draw();
```

runs code when screen refreshes

so at 60Hz, that's ~16ms


web
workers

CALLBACKS

a function that waits for the right time to strike

like a fucking ninja

```
anime({
  translateY: {
    value: 500,
    duration: 5000
  },
  complete: function() {
    console.log( 'he dead' );
    ball.remove();
  }
});
```


DOM

so what does the browser do, when you open a website?

1/ LOADS THE RESOURCE

dns lookup, tcp handshake, downloads the html

2/ PARSES THE HTML

tag soup into html objects

so what does the browser do, when you open a website?

3/ CREATES DOM TREE

tree of elements based on relationships, API, CCSOM

4/ CREATES RENDER TREE

combines DOM and CSSOM for a rough list of elements to be shown

so what does the browser do, when you open a website?

5/ LAYOUT OF THE RENDER TREE

layout or flow, calculates size & positions of boxes

6/ PAINT

takes all info and turns it into actual pixels

steps **4, 5, 6 get repeated** whenever new info comes in

when image gets loaded
javascript affects css layout
browser window gets resized
user scrolls
...

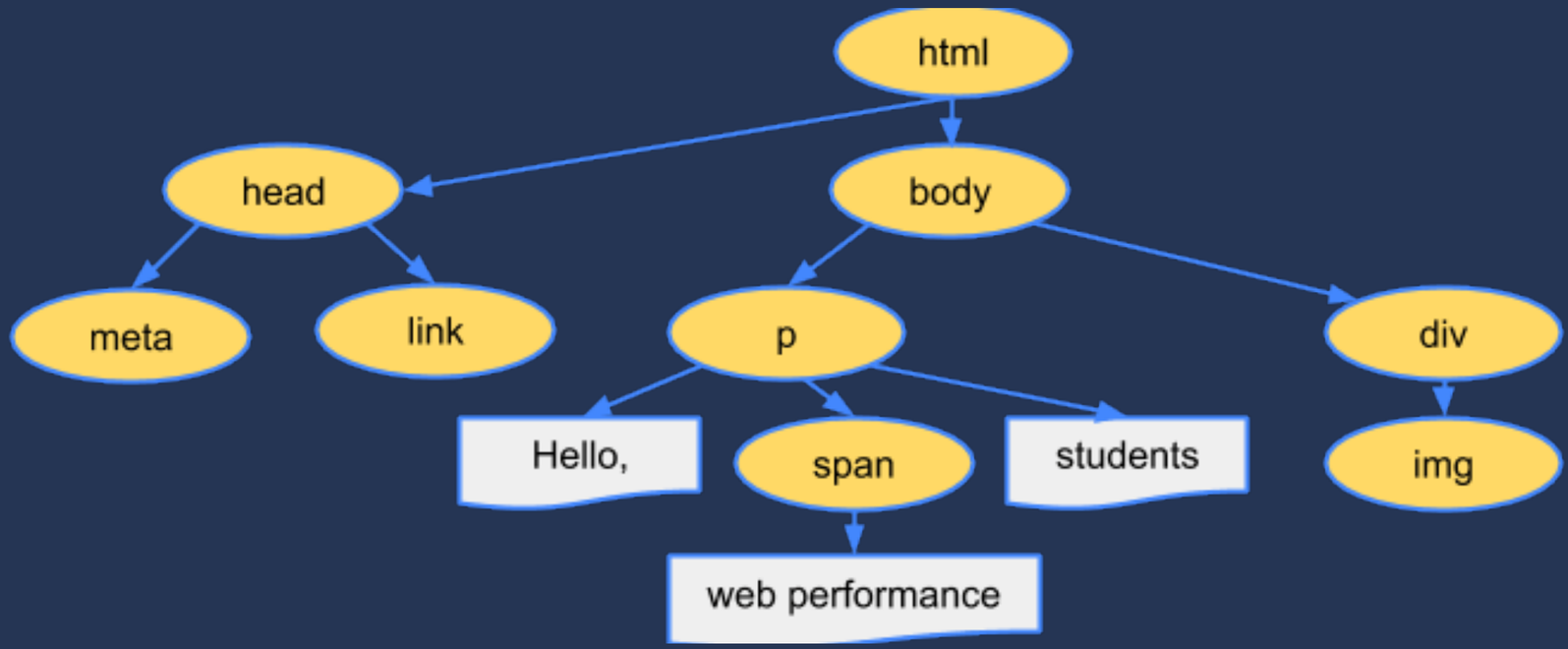
reflow when layout recalculates **repaint** for visual changes

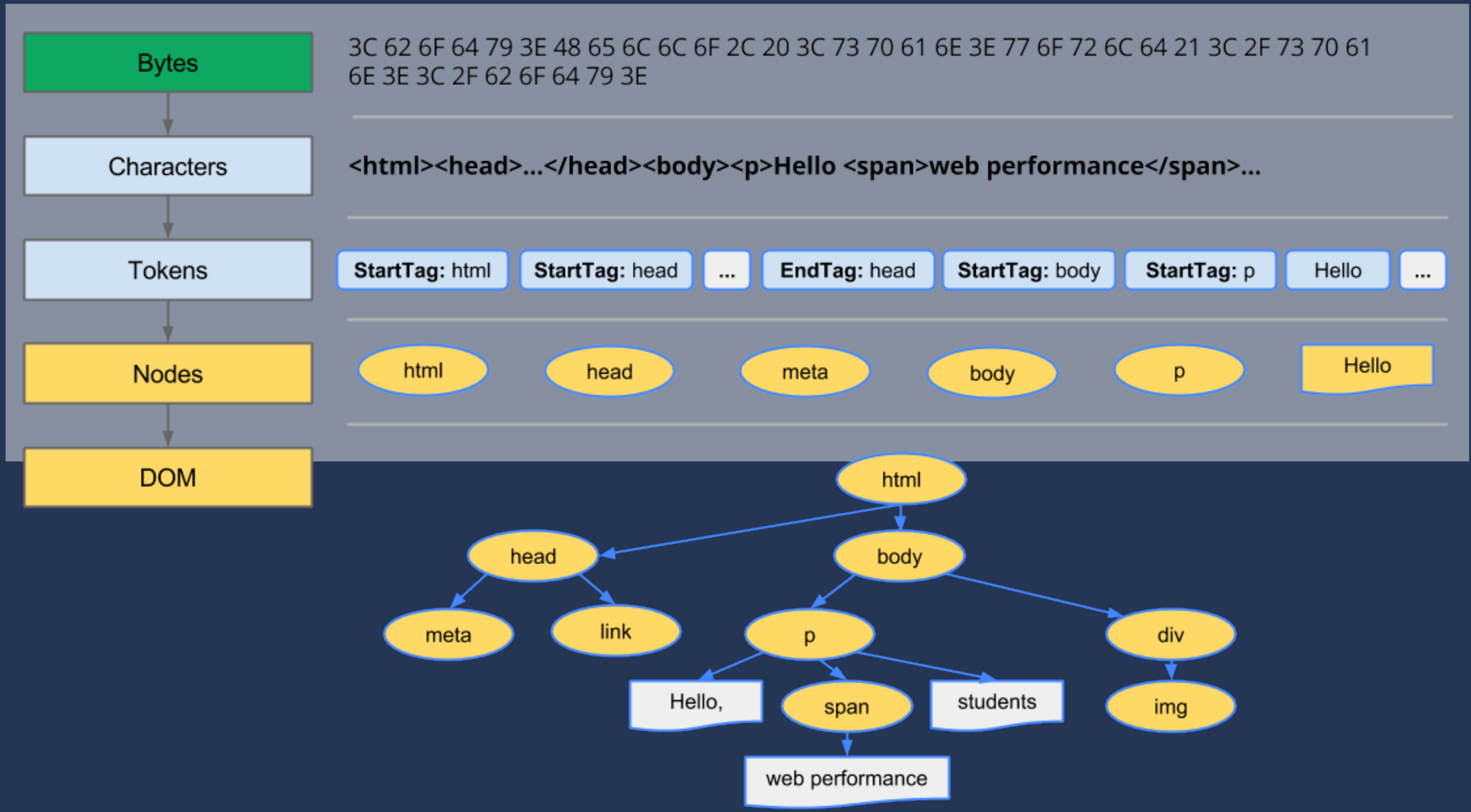
so the browser turns html code like this

```
<html>
 <head>
   <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>
```

into this here **DOM tree**

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




Bytes
3C 62 6F 64 79 3E 48 65 6C 6C 6F 2C 20 3C 73 70 61 6E 3E 77 6F 72 6C 64 21 3C 2F 73 70 61 6E 3E 3C 2F 62 6F 64 79 3E
Characters
<html><head>...</head><body><p>Hello web performance...
Tokens
StartTag: html
StartTag: head
...
EndTag: head
StartTag: body
StartTag: p
Hello
...
Nodes
html
head
meta
body
p
Hello
DOM
html
head
body
meta
link
p
div
Hello,
span
students
img
web performance


FLOW
PAINT
SCRIPT
EVENT
GOES LIKE THIS


DOM PERFORMANCE
gotta limit them reflows and repaints

less

files, DOM elements, styled elements, DOM touching

faster

flexbox, css grid > float

DOM PERFORMANCE

HTML, CSS, JS are **render blocking**

browser has to dl / execute them, before creating render tree

so

put CSS **as close to top**
put JS **as close to bottom**

minimize, compress, cache (module bundlers)

DOM PERFORMANCE

if your .js **doesn't touch the DOM** or CSSOM

```
<script src="script.js" async></script>
```

or you can **defer** till all else is done

```
<script src="script.js" defer></script>
```

```
{
  display: none
}
```

layout AND paint

like, don't force layouts / repaints when not needed

```
{
  visibility: hidden
}
```

just paint

some of what causes reflows **and** repaints

- Adding, removing, updating DOM nodes

- Hiding a DOM node with display: none (reflow and repaint)
  or visibility: hidden (repaint only, because no geometry changes)

- Moving, animating a DOM node on the page

- Adding a stylesheet, tweaking style properties (size, margin, padding, border, font...)

- Retrieving measurements (offsetWidth, offsetHeight, getComputedStyle())

- User action such as resizing the window,
  changing the font size, or even scrolling

...can cause BIG performance hits (so debounce, throttle)

some tips how not to
reflow **or** repaint

1/ ANIMATE POSITION: ABSOLUTE/FIXED

css transform, (not top/left)

2/ CHANGE CLASS, NOT CSS PROPERTIES

```
let toChange = document.getElementById('mainelement');
toChange.style.background = '#333';
toChange.style.color = '#fff';
toChange.style.border = '1px solid #00f';

.highlight { ... }  // <-- do this one
```

3/ CAN'T TOUCH DOM

create 1 huge node, insert in bulk

4/ CONSIDER HIDING ELEMENTS

display: none, then change bunch of values

5/ CACHE CACHE CACHE

dom nodes and such

5/ CACHE CACHE CACHE (DOM nodes)

```
// wrong, fuck you
document.getElementById('test').property1 = 'value1';
document.getElementById('test').property2 = 'value2';
document.getElementById('test').property3 = 'value3';

  vs.

// nice
let dom = document.getElementById('test');
dom.property1 = 'value1';
dom.property2 = 'value2';
dom.property3 = 'value3';
```

6/ READING IS EXPENSIVE (in loops esp.)

7/ CLUMP READS/WRITES TOGETHER

```
// sucks, causes layout twice
let newWidth = aDiv.offsetWidth + 10; // Read
aDiv.style.width = newWidth + 'px'; // Write
let newHeight = aDiv.offsetHeight + 10; // Read
aDiv.style.height = newHeight + 'px'; // Write

// better, only one layout
let newWidth = aDiv.offsetWidth + 10; // Read
let newHeight = aDiv.offsetHeight + 10; // Read
aDiv.style.width = newWidth + 'px'; // Write
aDiv.style.height = newHeight + 'px'; // Write
```

8/ CACHE CACHE CACHE (values)

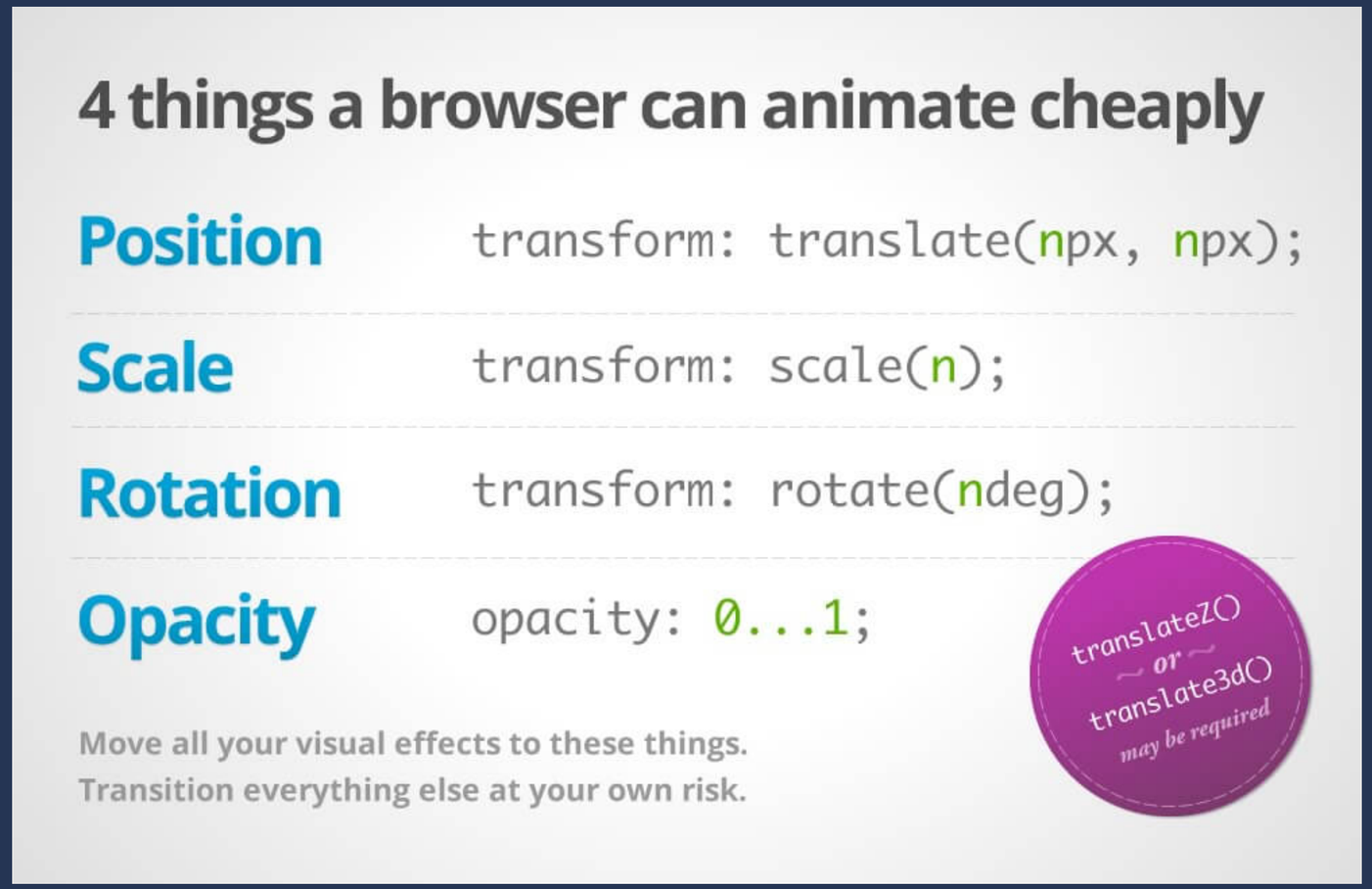
```
// bad, sucks
for (let i = 0; i < paragraphs.length; i++) {
  paragraphs[i].style.width = box.offsetWidth + 'px';
}

// nice, is good
let width = box.offsetWidth;
for (let i = 0; i < paragraphs.length; i++) {
  paragraphs[i].style.width = width + 'px';
}
```

CSS TRANSFORM

not javascript animation

animate CSS instead of JS position, scale, rotation, opacity



http://www.html5rocks.com/en/tutorials/speed/high-performance-animations/

layers and hardware acceleration

```
/* the OLD way */
.myAnimation {
    animation: someAnimation 1s;
    transform: translate3d(0, 0, 0); /* force hardware acceleration */
}

/* the NEW way */
.element {
    transition: transform 1s ease-out;
}
.element:hover {
    will-change: transform; /* the magic happens here */
}
.element:active {
    transform: rotateY(180deg);
}

/* use SPARINGLY */
```


and
DON'T TOUCH
THE DOM


debounce

limit how often some events *(mousemove scroll)* fire

```
let timeout; // DEBOUNCE
input.addEventListener('keyup', () => { // only fire ajax after user stopped typing
    clearTimeout(timeout);
    timeout = setTimeout(() => {
        fauxAjax();
    }, 500);
});

let scheduled = false; // THROTTLE
addEventListener('scroll', () => { // don't draw on every scroll, but only every 70ms
  if ( !scheduled ) {
      scheduled = true;
      setTimeout(() => {
          scheduled = false;
        // draw...
    }, 70);
  }
});
```

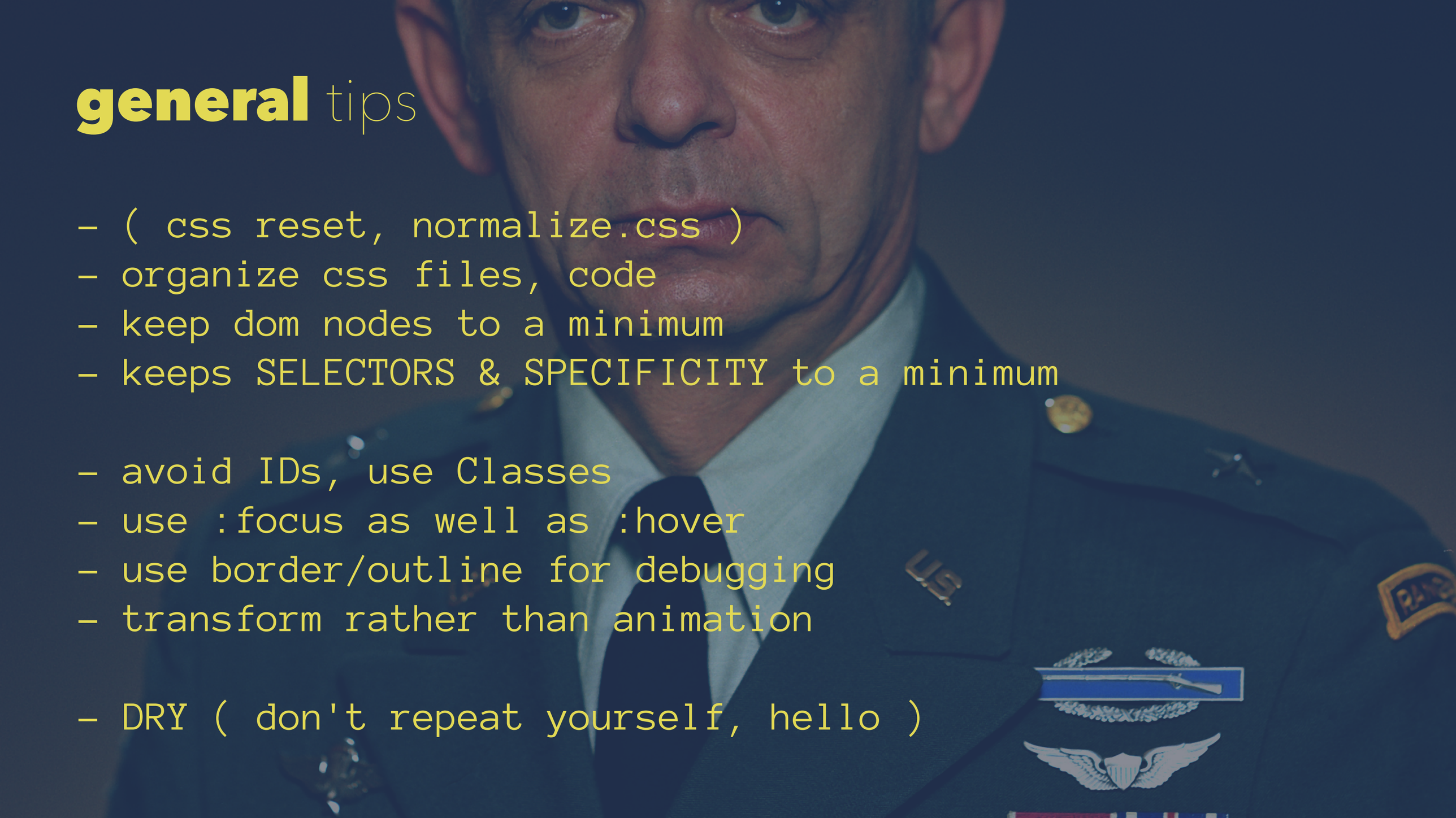
CSS BEST PRACTICES

general tips

- (css reset, normalize.css)
- organize css files, code
- keep dom nodes to a minimum
- keeps SELECTORS & SPECIFICITY to a minimum

- avoid IDs, use Classes
- use :focus as well as :hover
- use border/outline for debugging
- transform rather than animation

- DRY (don't repeat yourself, hello)



```
/* general */
html { box-sizing: border-box; }
*, *:before, *:after { box-sizing: inherit; }

/* clearfix */
.group:before,
.group:after { content: " "; display: table; }
.group:after { clear: both; }

/* responsive images */
img {
    width: 100%;
    max-width: 100%;
    vertical-align: middle;
}

/* <meta name="viewport" content="width=device-width, initial-scale=1"> */
```

```css
/* DRY */
.btn {
    font-size: 1.0625em;
    font-family: 'Montserrat', sans-serif;
    display: inline-block;
    padding: 1.25em 1.5em; /* 20/16 */ /* 24/16 */
    border-radius: 0.375em; /* 6/16 */
    background-color: rgba(84, 87, 102, 1);
}

.btn-red {
    background-color: rgba(253, 104, 91, 1);
}

.btn-red:hover,
.btn-red:focus {
    background-color: rgba(253, 104, 91, 0.9);
}
```

```
/* a background that covers the whole screen */
body {
    background: url('../img/city.jpg') center center no-repeat;
    background-attachment: fixed;
    background-size: cover;
}

/* for a small screen, load smaller image */
@media ( max-width: 600px ) {
    body {
        background-image: url('../img/city600.jpg');
    }
}

/* styling inputs */
input:not([type="checkbox"]):not([type="radio"]):not([type="file"]) {
```


optimize for
CH-CH-CH CHANGES

Andy Hume: CSS for Grownups

```
/* instead of */
.post h1 {
    font-size: 1.2em;
    text-trasform: uppcercase;
}

/* do */
.post .post-title {
    font-size: 1.2em;
    text-trasform: uppcercase;
}

/* or */
.post-h {  }
```

headings are pretty important

```
/* doing this */
h1 { font-size: 3em; }
h2 { font-size: 2.3em; }
h3 { font-size: 2.1em; }
h4 { font-size: 1.8em; }
h5 { font-size: 1.3em; }

/* will force you */
h3 { font-size: 2.1em; }
.sidebar h3 { font-size: 1.8em; } /* size of h4 */
```

headings

```
/* so instead you might do classes */
.h1 { font-size: 3em; }
.h2 { font-size: 2.3em; }
.h3 { font-size: 2.1em; }
.h4 { font-size: 1.8em; }
.h5 { font-size: 1.3em; }

/* or even better */
.h-headline    { font-size: 3em; }
.h-subheadline { font-size: 2.3em; }
.h-byline      { font-size: 2.1em; }
.h-related     { font-size: 1.8em; }
.h-promo       { font-size: 1.3em; }
```

say you have **products**

```
/* instead of */
ul.product-list li {

}

/* do just */
.product-item {

}

/* you might even */
.product-item { }
.product-item-h { }
```

*never make **assumptions** about HTML*

be prepared to **reuse modules**

```
/* instead of */
.box {
    background: black;
}

/* focusing on the placement */
.sidebar .box {
    background: white;
}

/* focus on what changes */
.box-light {
    background: white;
}
```

same thing goes for **sub-pages**

```
/* this */
.product-page .box {

}

/* leads to this */
.product-page .box,
.about-page .box,
.contact-page .box,
.home-page. box {

}
```

just create one class, ***independent*** *of HTML*


CLOSURES
a function has access to all variables of the function it's contained within
it sees the variables of its parent
even after
THE PARENT DIEEEEEEEEES

```
function yellowFade( element ) {
  var level = 1, // step fn has access to this, even after i'm long dead
    fps = 20;

  function step() {
    setTimeout(function() {
      var h = level.toString(16);
      element.style.backgroundColor = '#FFFF' + h + h;

      if ( level < 15 ) {
        level += 1;
        requestAnimationFrame(step);
      }
    }, 1000 / fps);
  }

  requestAnimationFrame(step);
}
```

CLASSICAL *VS.* PROTOTYPAL INHERITANCE

objects have a secret...

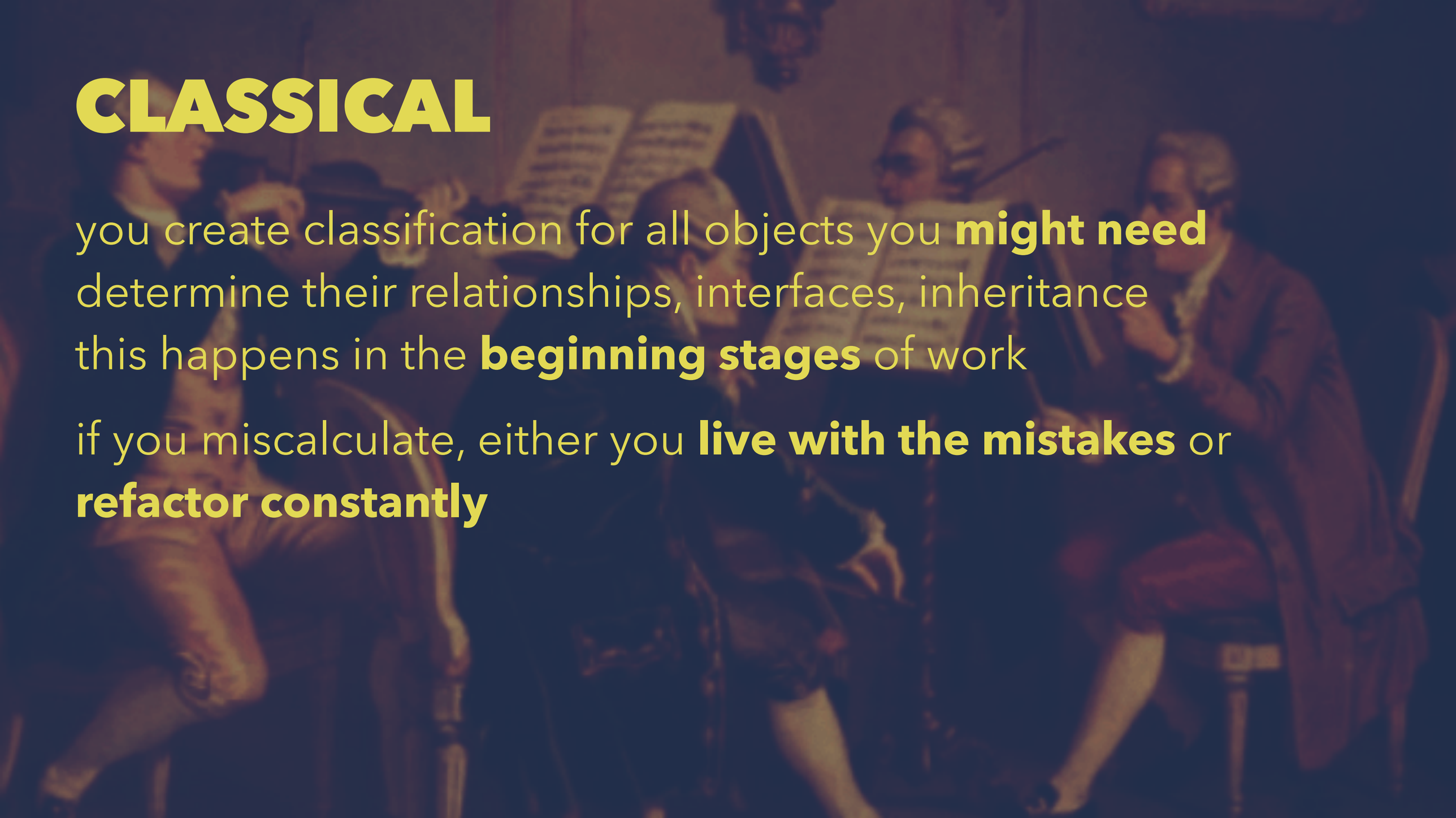
...link to another object

sometimes called ***__proto__***

```
var butt = {
  "for": "pooping",
  size: 2
}

console.dir( butt );

false.toString(); // 'false'
[1, 2, 3].toString(); // '1,2,3'
```



CLASSICAL

you create classification for all objects you **might need**
determine their relationships, interfaces, inheritance
this happens in the **beginning stages** of work

if you miscalculate, either you **live with the mistakes** or **refactor constantly**

PROTOTYPAL

you make an object, if you like it, you make another one
you **create** and **customize**

javascript is **class free**
you use functions to create objects

object inherit from each other

it's possible to augment the built in types


inheritance

PSEUDO-CLASSICAL *inheritance*

```
var Loader = function(selector) { // create constructor
    this.elem = $(selector)[0]; // set up properties
}

Loader.prototype.start = function() { // add inheritable methods
    this.elem.style.opacity = 1;
};

var FunkyLoader = function(selector) {
    Loader.call(this, selector); // we call the parent's constructor to set things up
    // add new properties
}

FunkyLoader.prototype = Object.create(Loader.prototype); // inherit from Loaders constructor
FunkyLoader.prototype.constructor = FunkyLoader; // the previous call "dirtied up" our new constructor

var loader = new Loader('.spinner');
var funky = new FunkyLoader('.secondary');
```

PARASITIC *inheritance*

```
function Loader(selector) {
    var elem = $(selector)[0];
    return {
        start: function() { elem.style.opacity = 1  },
        stop:  function() { elem.style.opacity = 0 }
    }
}

function FunkyLoader(selector) {
    var that = Loader(selector); // this is the IMPORTANT bit, inherit
    that.bounce = function() { ... } // modify
    return that;
}

var loader = Loader('.spinner');
var funky  = FunkyLoader('.secondary');
```

BEHAVIOR *DELEGATION*

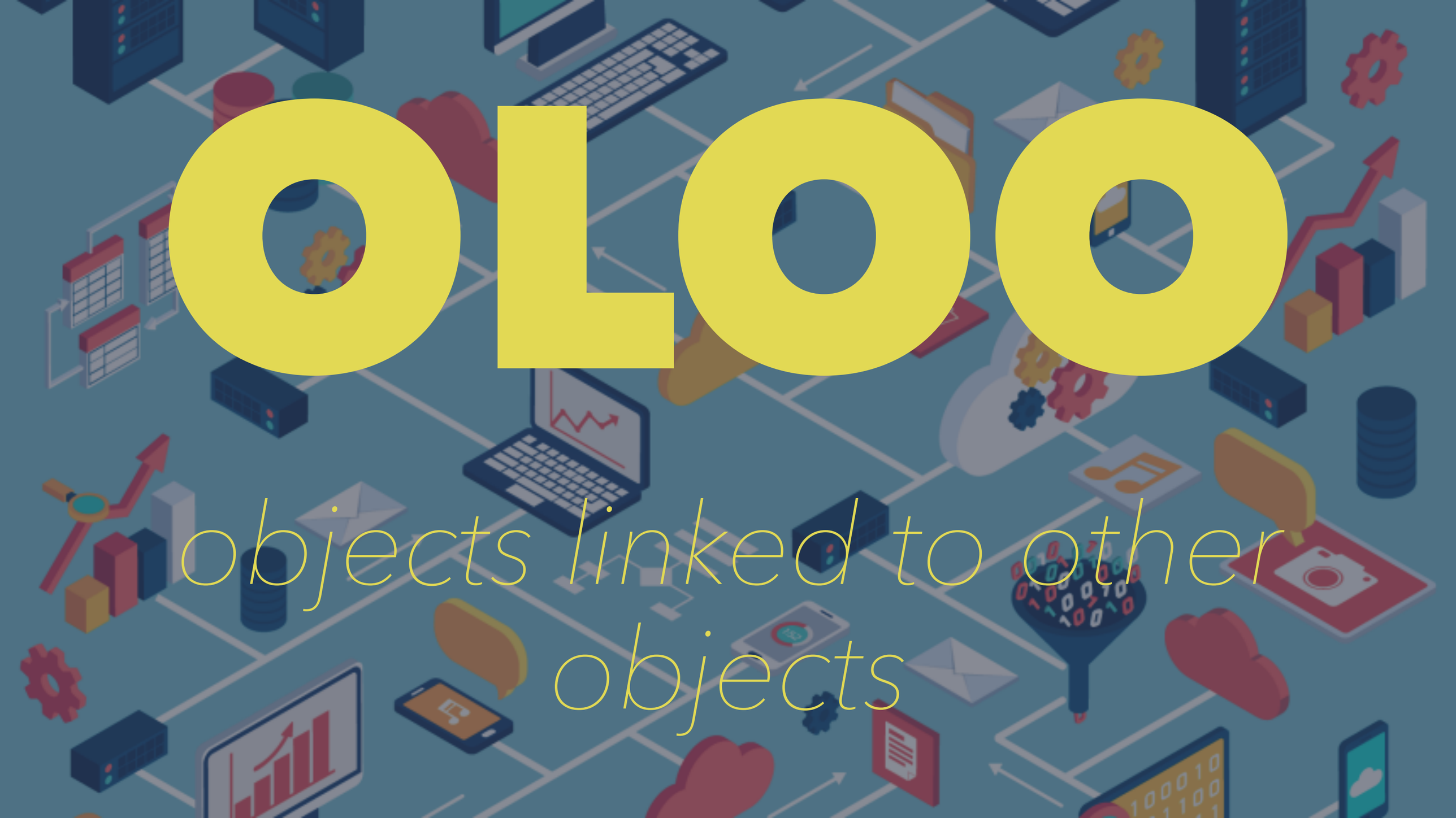
```
// we simply create a lean, mean object
var Loader = {
    init:  function(selector) { this.elem = $(selector)[0] },
    start: function() { this.elem.style.opacity = 1 },
    stop:  function() { this.elem.style.opacity = 0 }
};

// create a new object with parents proto and props
var FunkyLoader = Object.create( Loader );

// we simple change what we dont like, or add
FunkyLoader.init = function(selector) {
    this.elem = $(selector)[0];
    this.elem.classList.add('funky');
}

var loader = Object.create(Loader);
var funky  = Object.create(FunkyLoader);
```


those were just 3 patterns
js is super versatile


OLOO
objects linked to other objects

OLOO

```
var Mom = {
    init: function(who)  { this.me = who },
    identify: function() { return `I am ${this.me}` }
};

var Son = Object.create( Mom );

Son.speak = function() {
    console.log( `Hello, ${this.identify()}.` );
};

var b1 = Object.create( Son );
b1.init( "b1" );
var b2 = Object.create( Son );
b2.init( "b2" );

b1.speak(); b2.speak();
```


now u
pro@js

@yablko
yablko.sk