HTTP/2

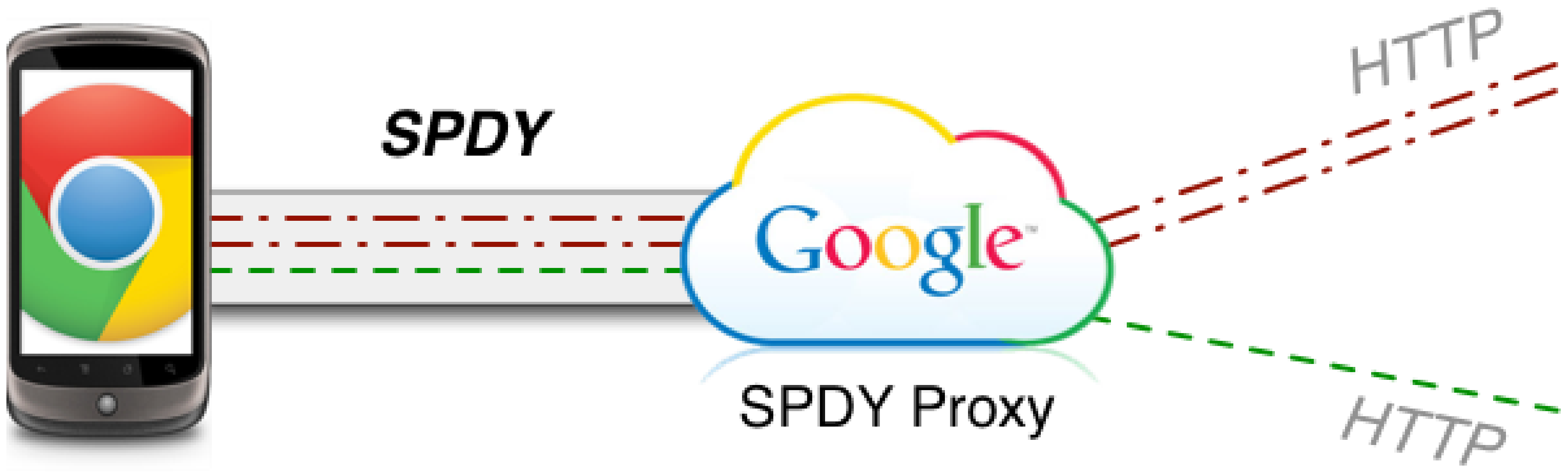
The Evolution Continues

Agenda

- SPDY & HTTP/2
- HTTP 1.X Issues
- Hacks
- HTTP/2

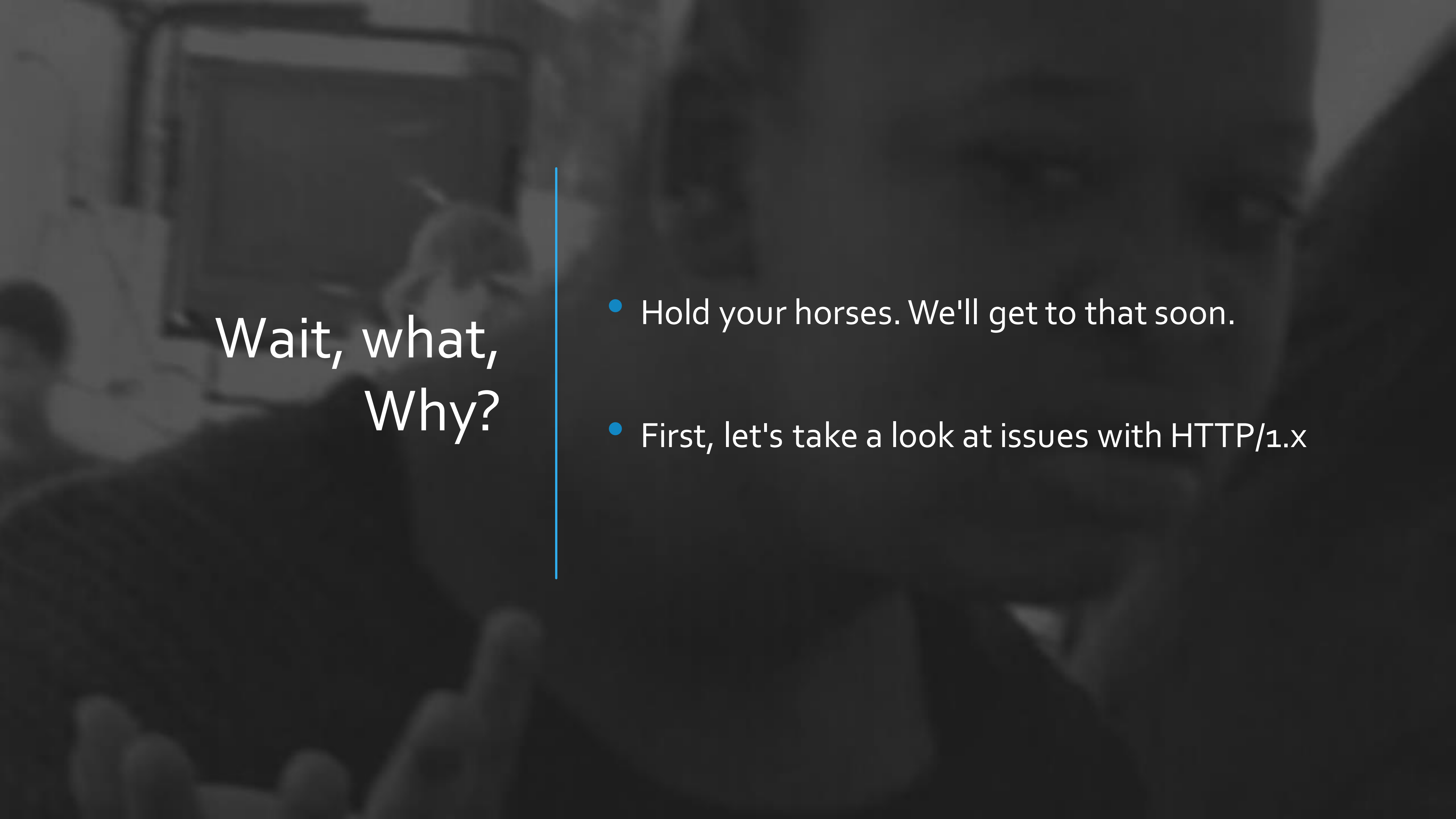
SPDY

A protocol developed by google for manipulating HTTP in a certain way to deliver Web Content.


SPDY
Google
SPDY Proxy
HTTP
HTTP

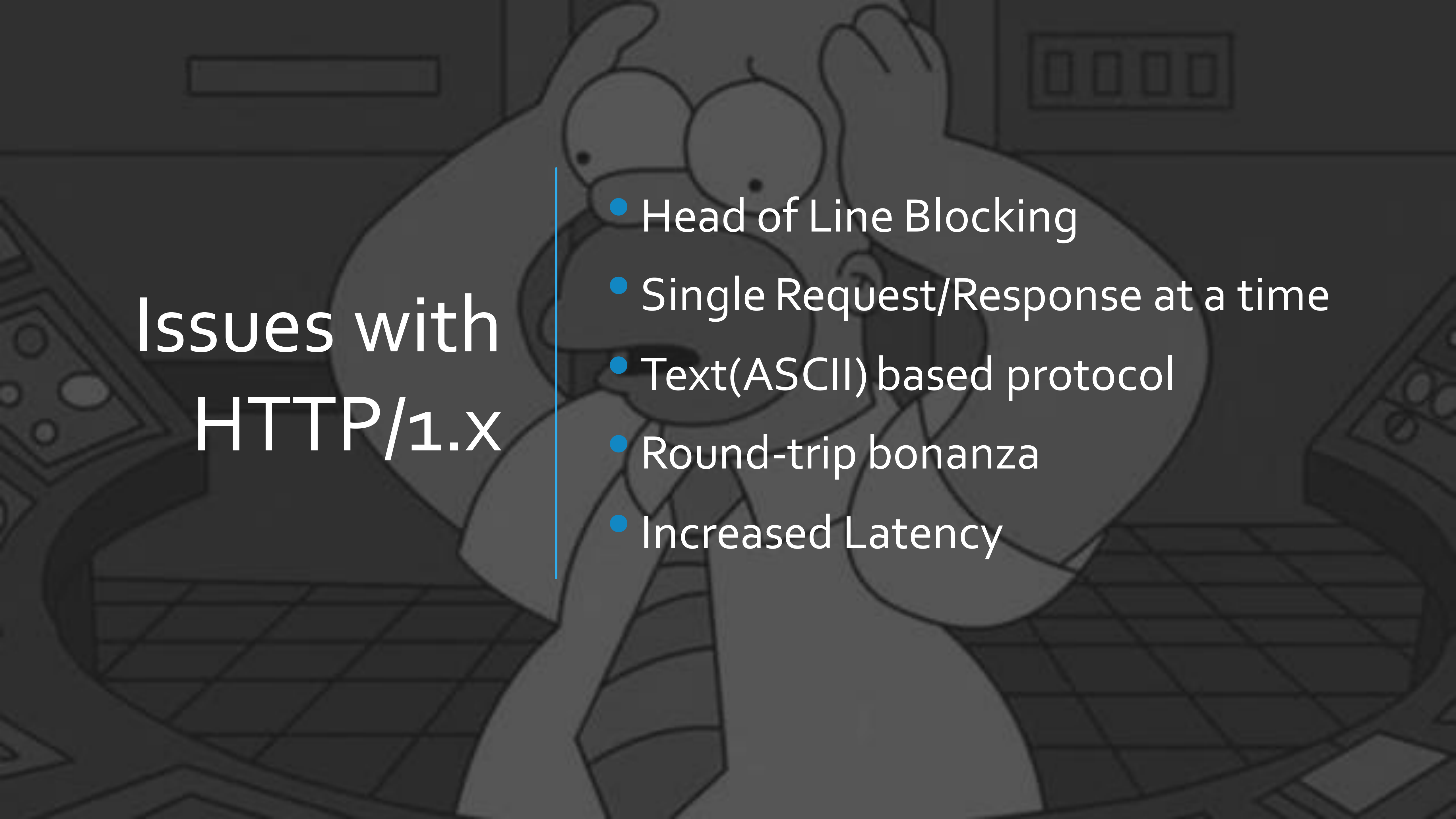
Coming soon ….

HTTP/2



Wait, what, Why?

- Hold your horses. We'll get to that soon.
- First, let's take a look at issues with HTTP/1.x

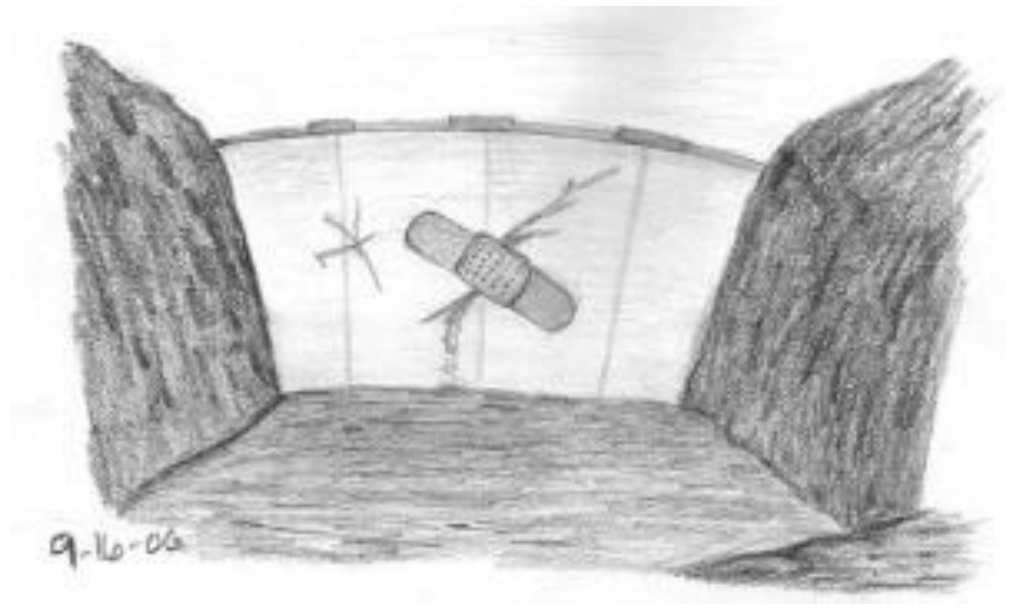


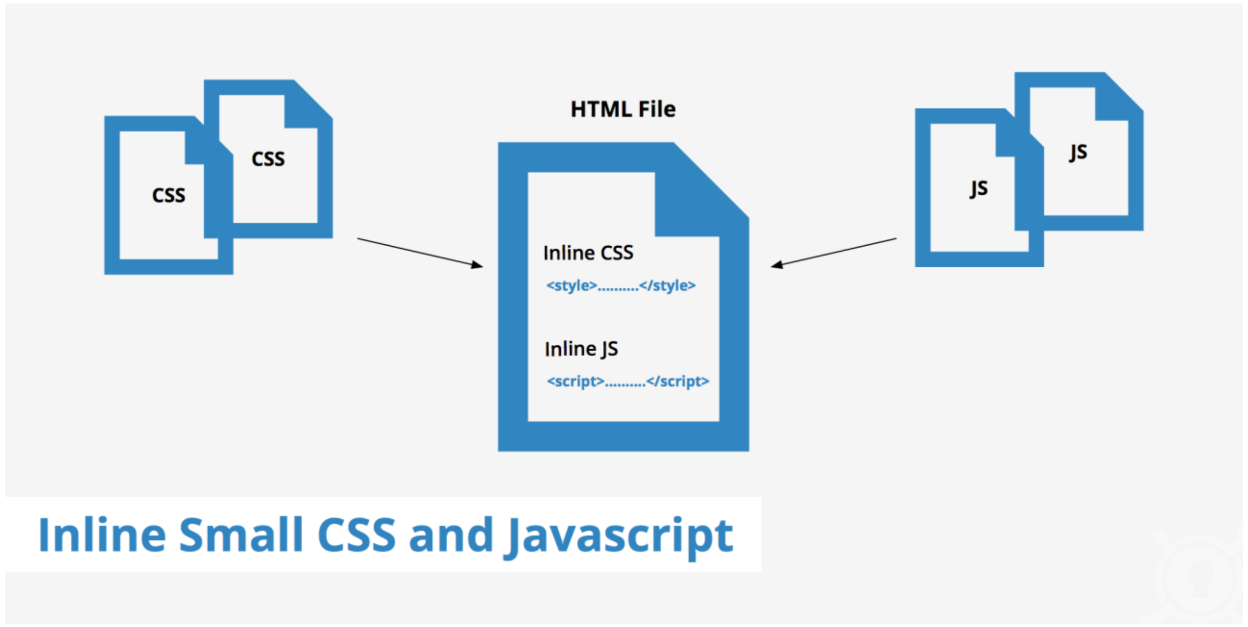
Issues with HTTP/1.x

- Head of Line Blocking
- Single Request/Response at a time
- Text(ASCII) based protocol
- Round-trip bonanza
- Increased Latency

HTTP/1.x Hacks

- Inlining
- Spriting
- Concatination
- Domain Sharding




CSS
CSS
HTML File
Inline CSS
<style>.........</style>
Inline JS
<script>.........</script>
JS
JS
Inline Small CSS and Javascript

Inlining

But ....

- Lack of Caching
- Poor Accessibility
- Difficult Code Management

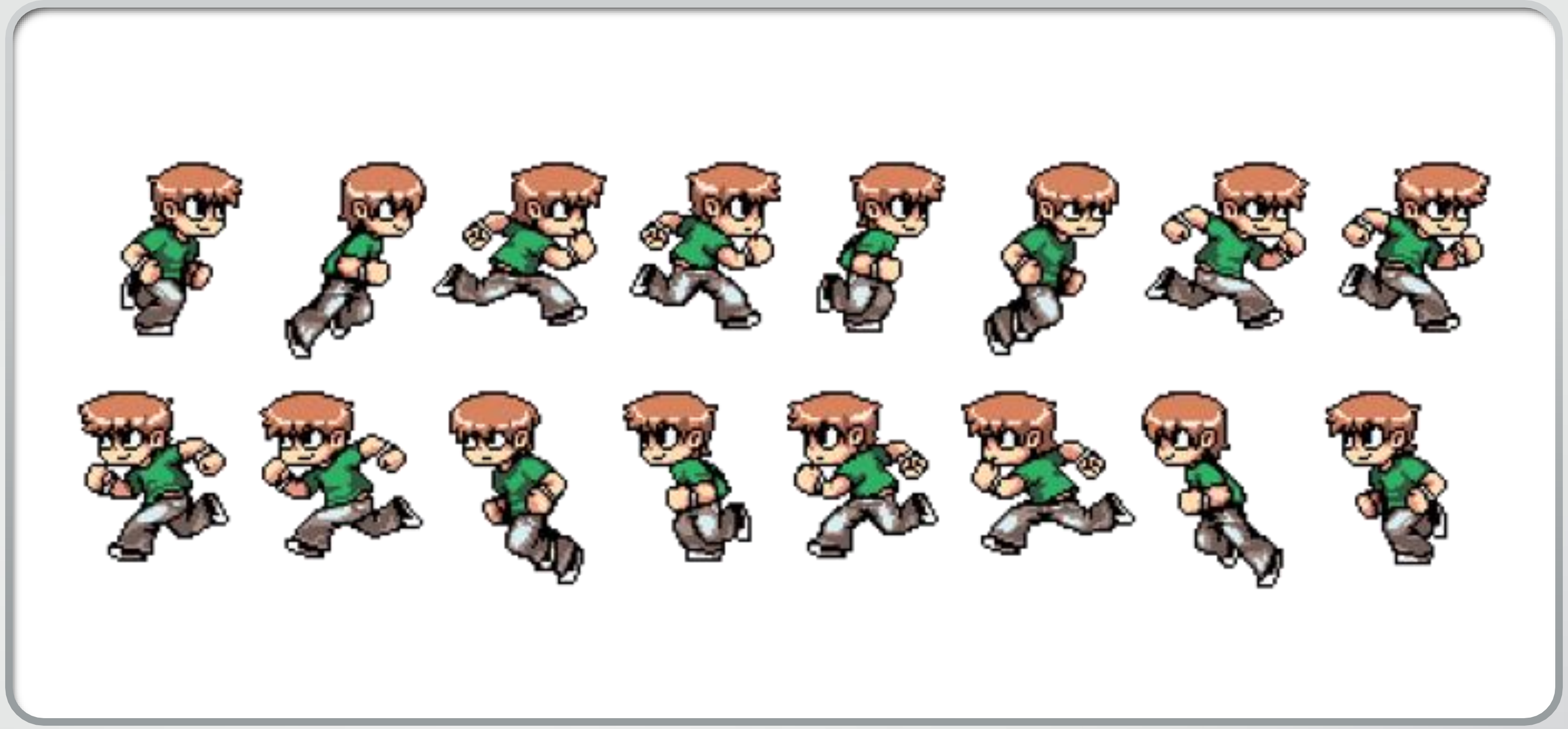
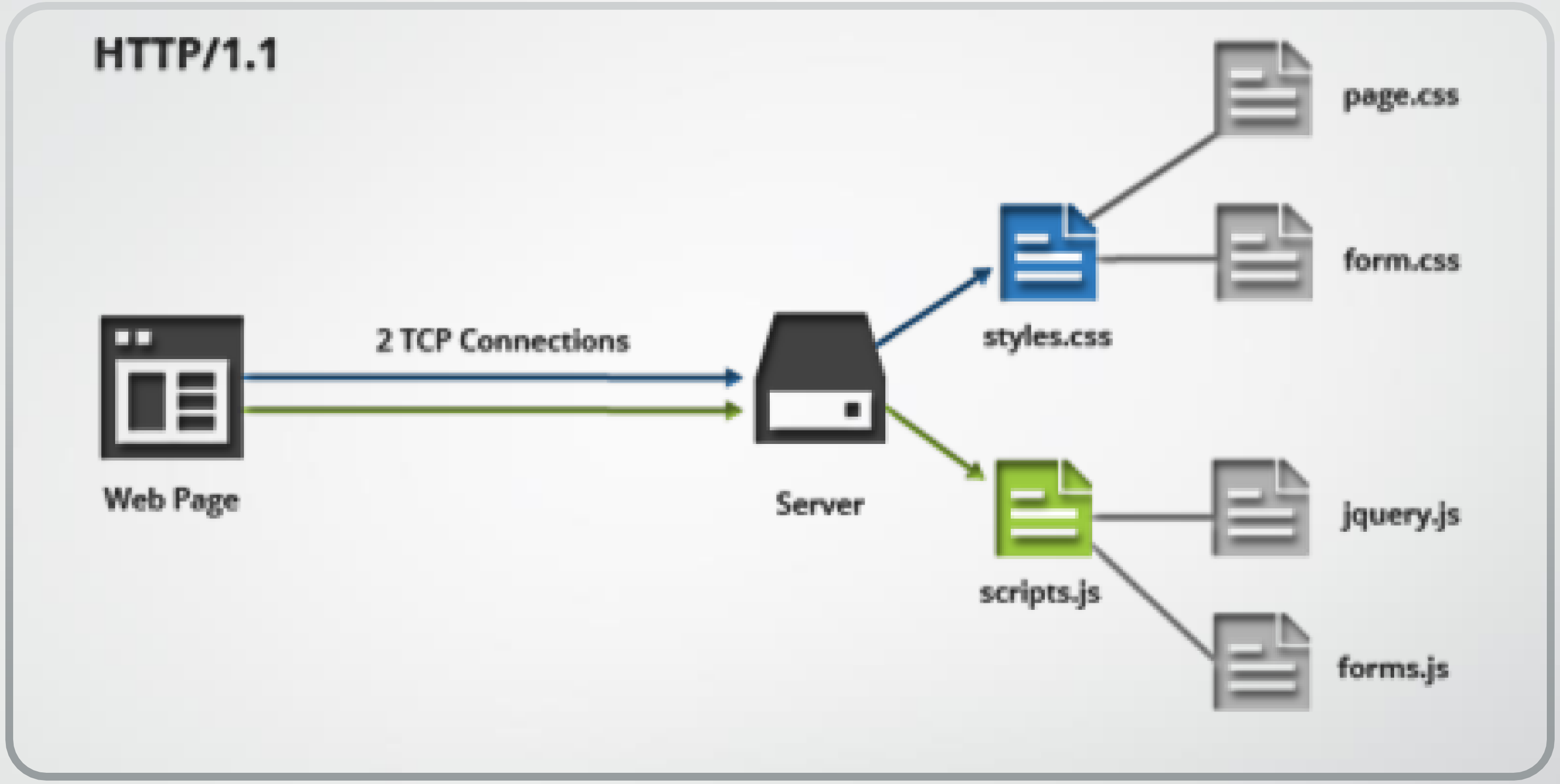


Image Spriting

But there's a catch!

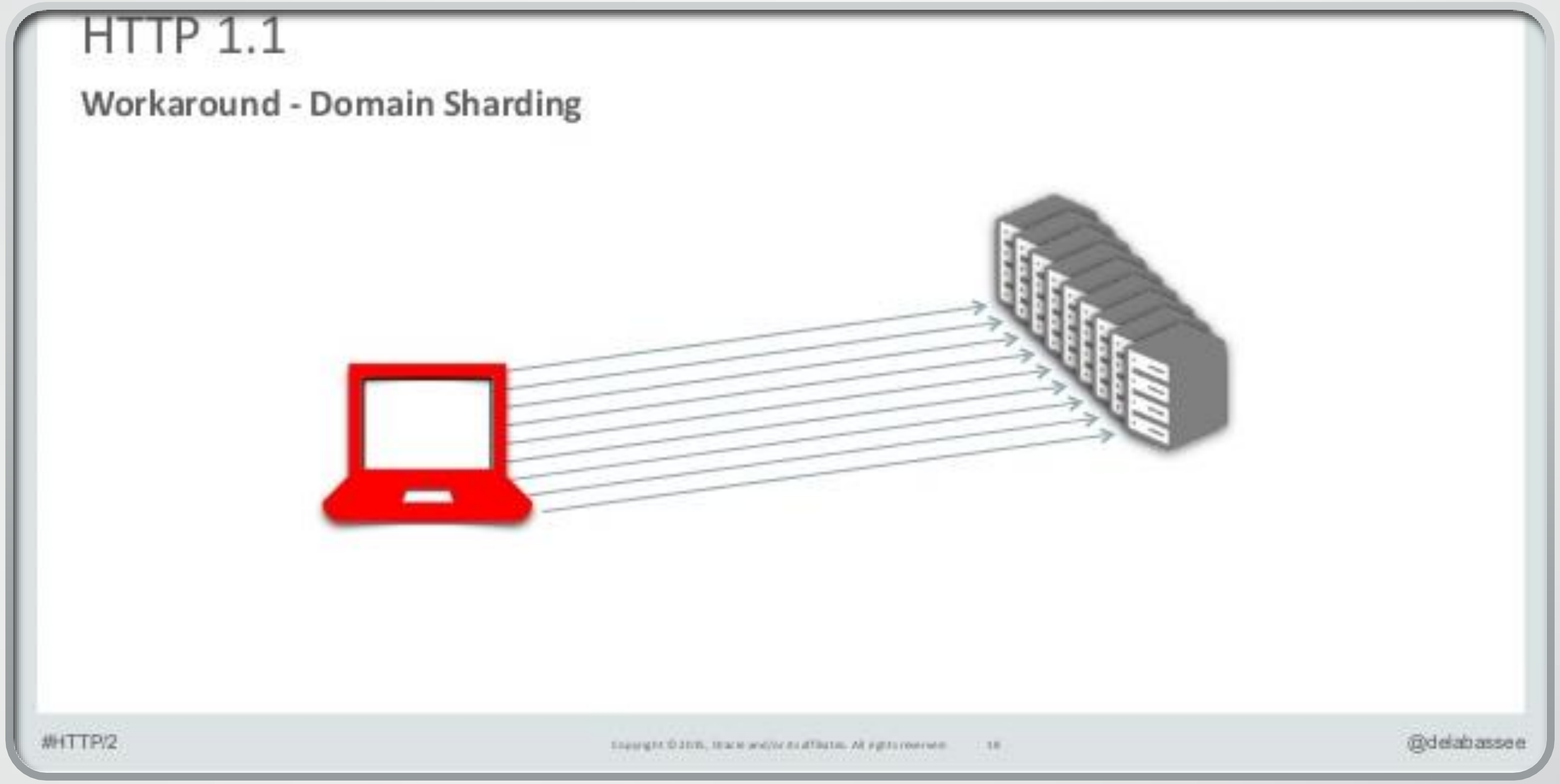
Cache Management .-.



File Concatination

But ....

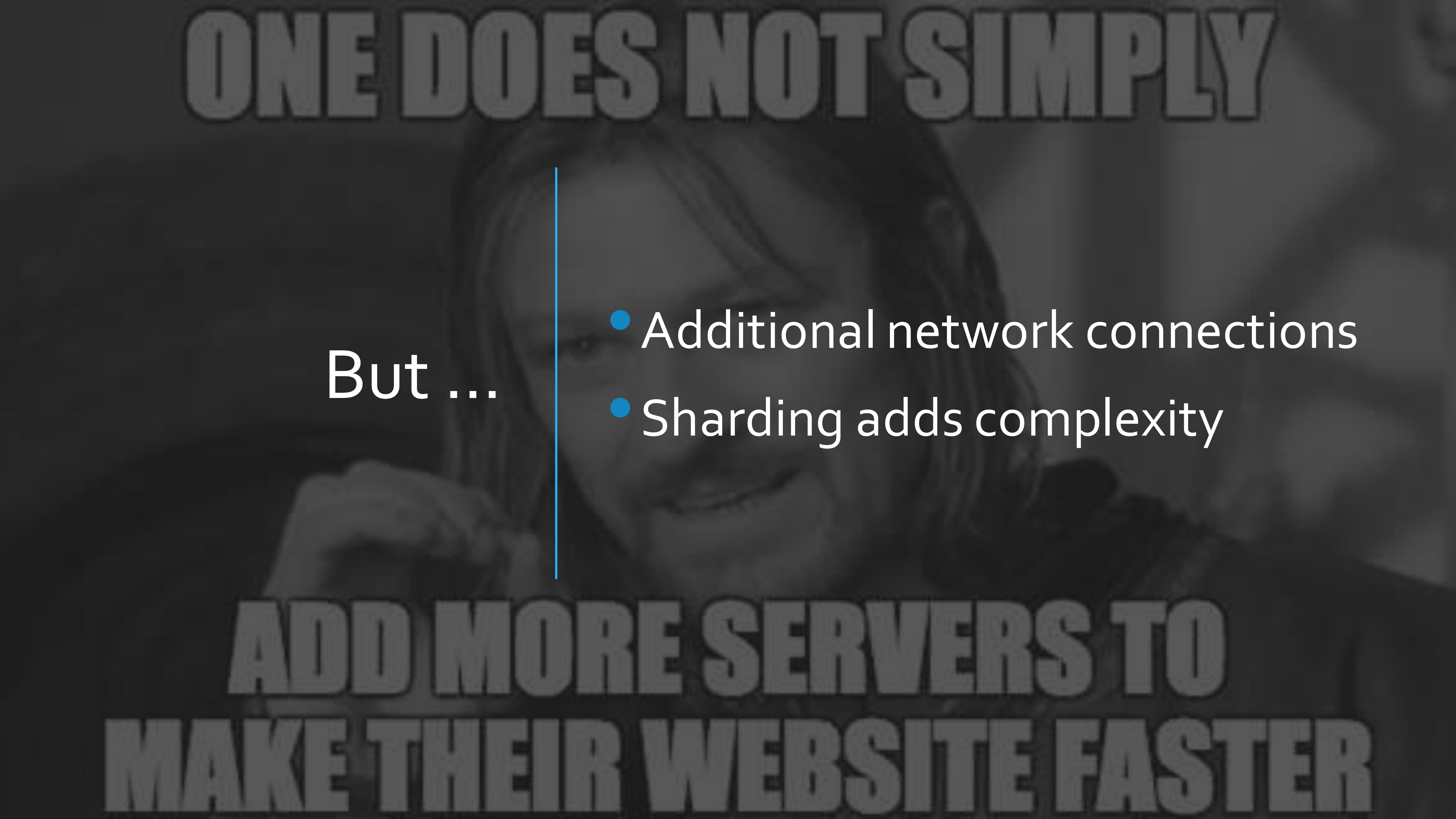
- Load time increases
- Bundled file management



HTTP 1.1

Workaround - Domain Sharding

Domain Sharding



But …

- Additional network connections
- Sharding adds complexity

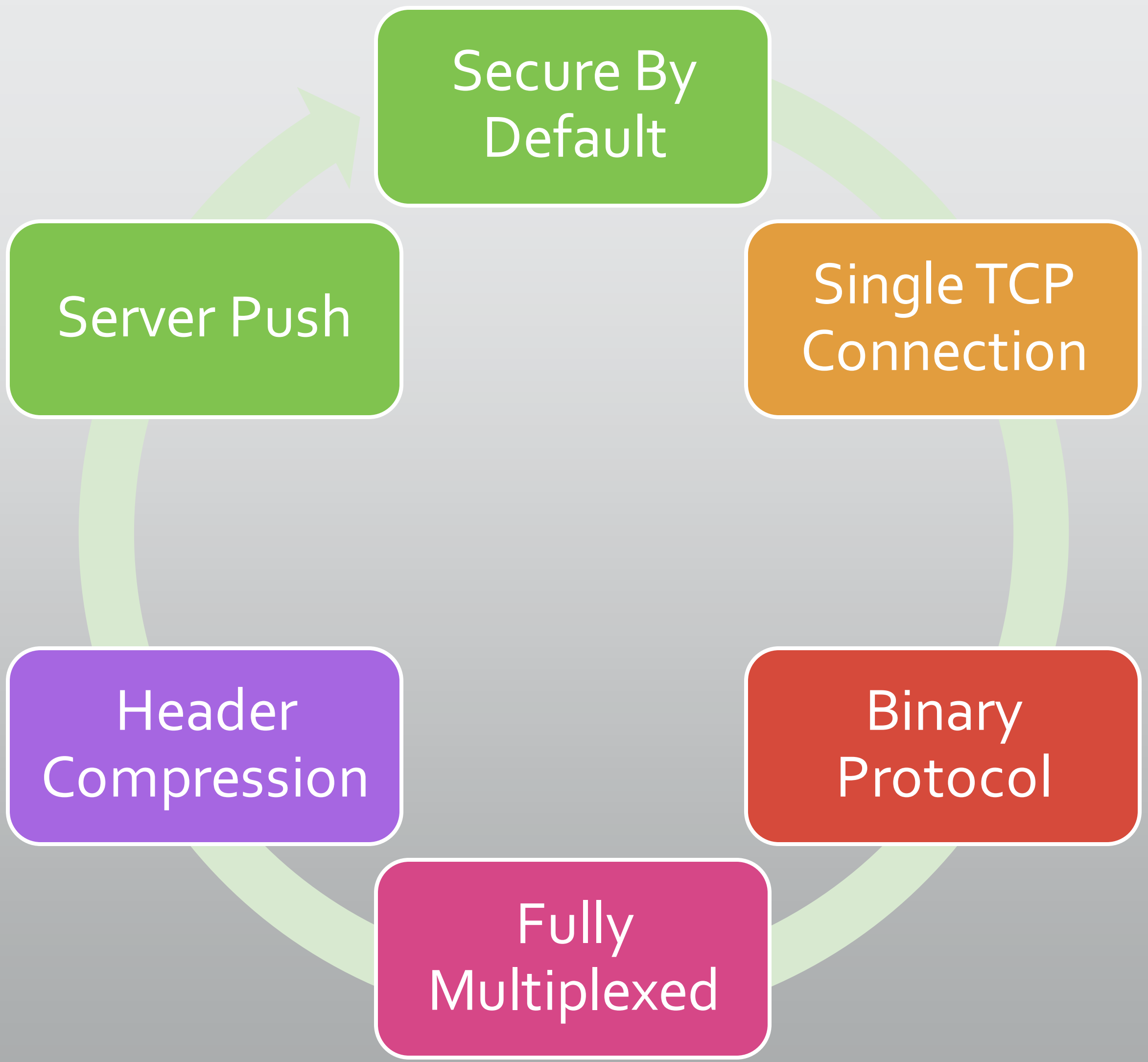
Don’t you worry, kitty cat! We've got you covered ^.^

drum roll please...

HTTP/2

For Faster and Safer Internet

Why HTTP/2 ?


Secure By Default
Single TCP Connection
Binary Protocol
Fully Multiplexed
Header Compression
Server Push

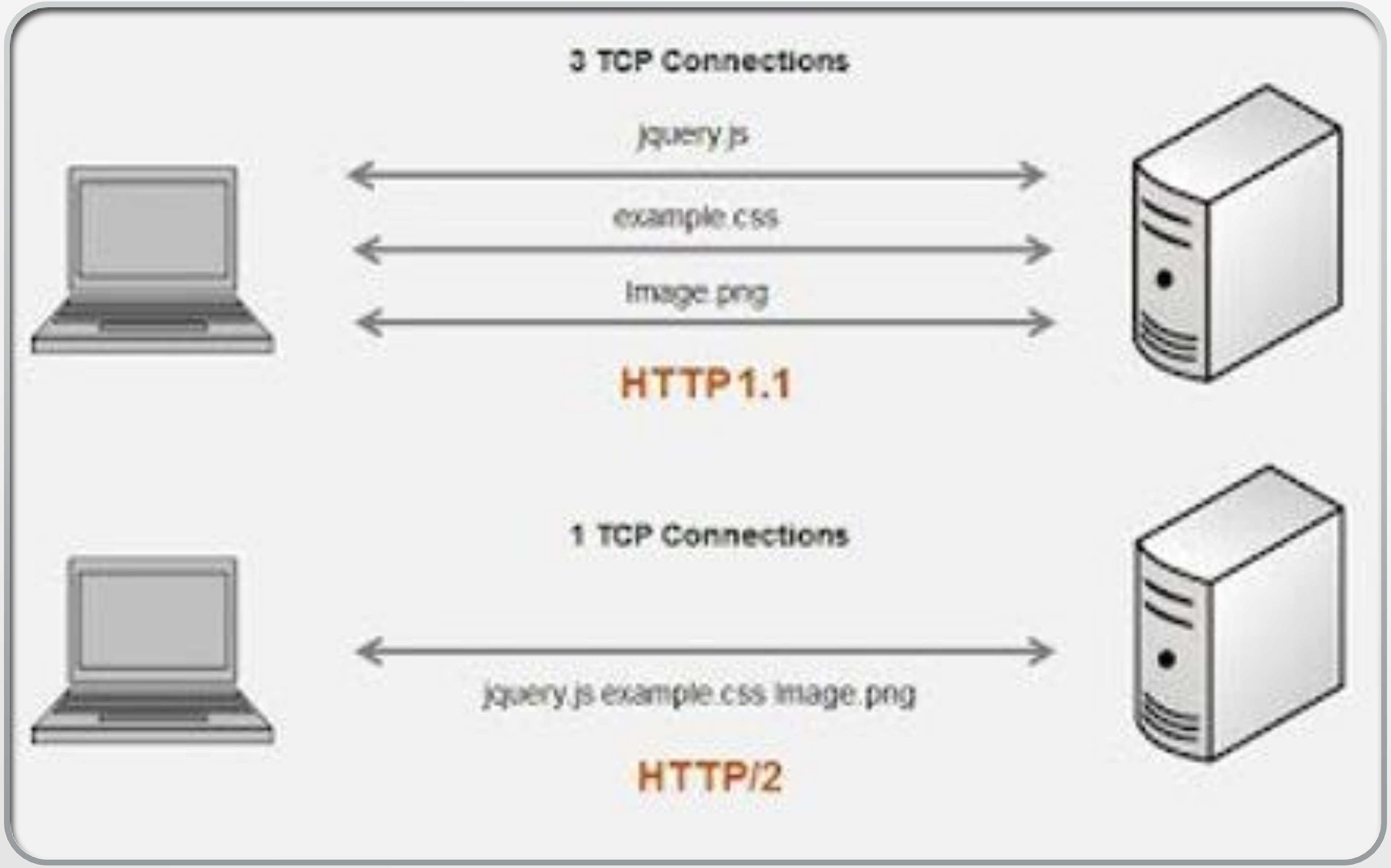
Secure By Default

- Almost all implementations of HTTP/2 require TLS.
- All browsers that currently support HTTP/2 require TLS connection.

Single TCP Connection

One TCP connection per server

Avoids network congestion.


3 TCP Connections
jquery.js
example.css
Image.png
HTTP1.1
1 TCP Connections
jquery.js example.css Image.png
HTTP/2

Binary Protocol

Binary protocols are more efficient to parse.

More compact "over the wire".

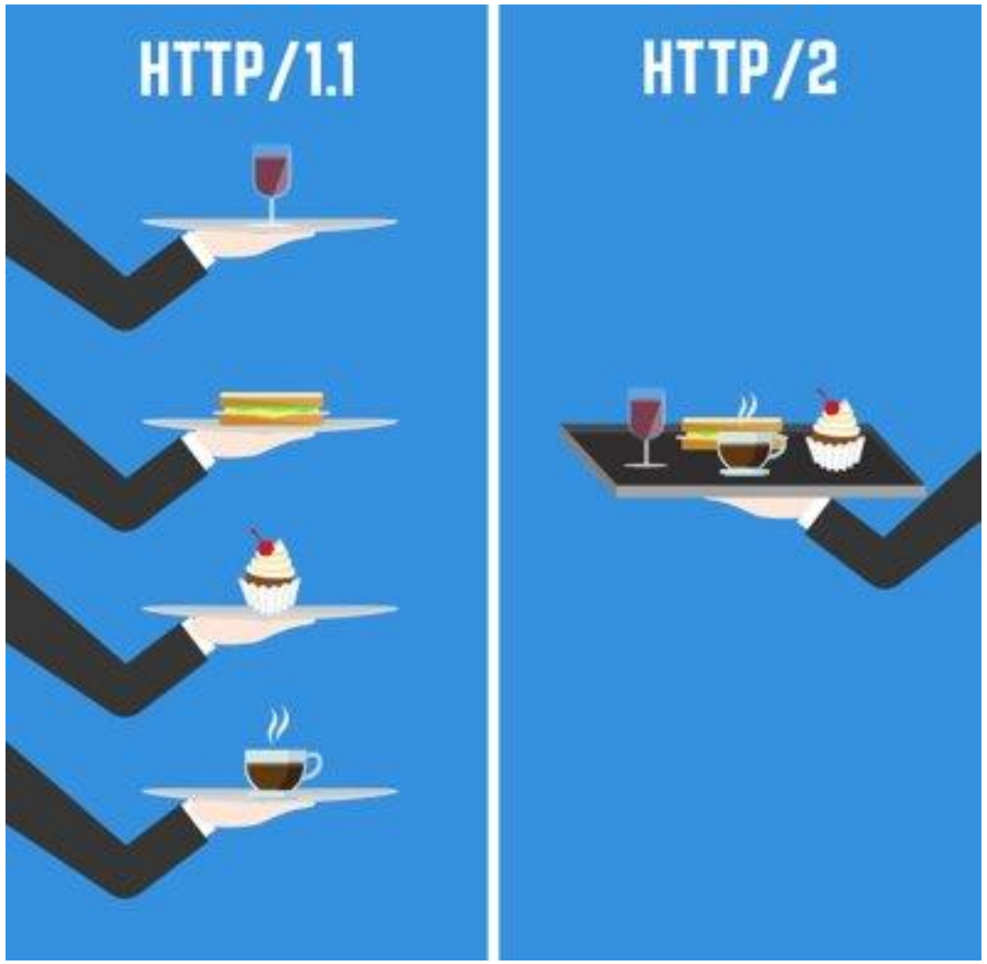
Less prone to errors with whitespaces, text cases etc.

Binary Protocol

A request/response in HTTP1.1 is a single enclosed unit, in HTTP/2 messages are split up in *frames*

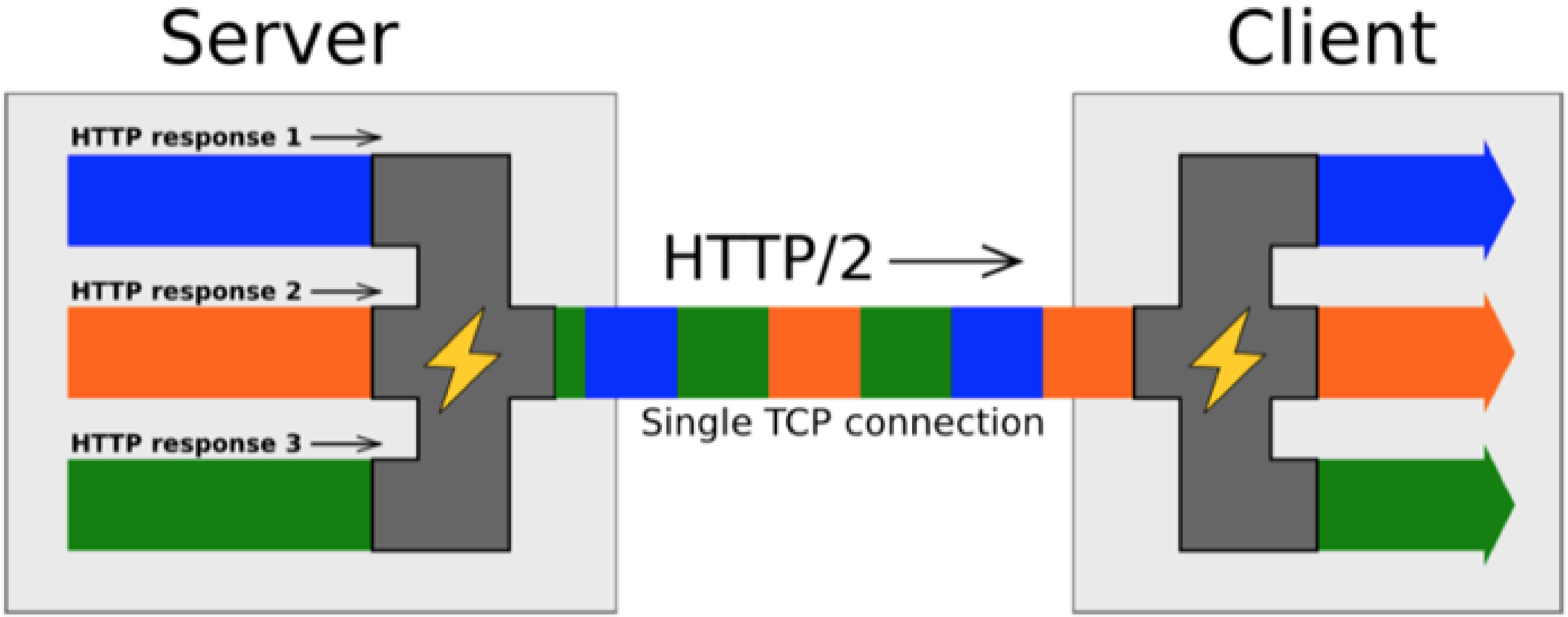
Every frame can be assigned to a *stream* by its *stream-id*

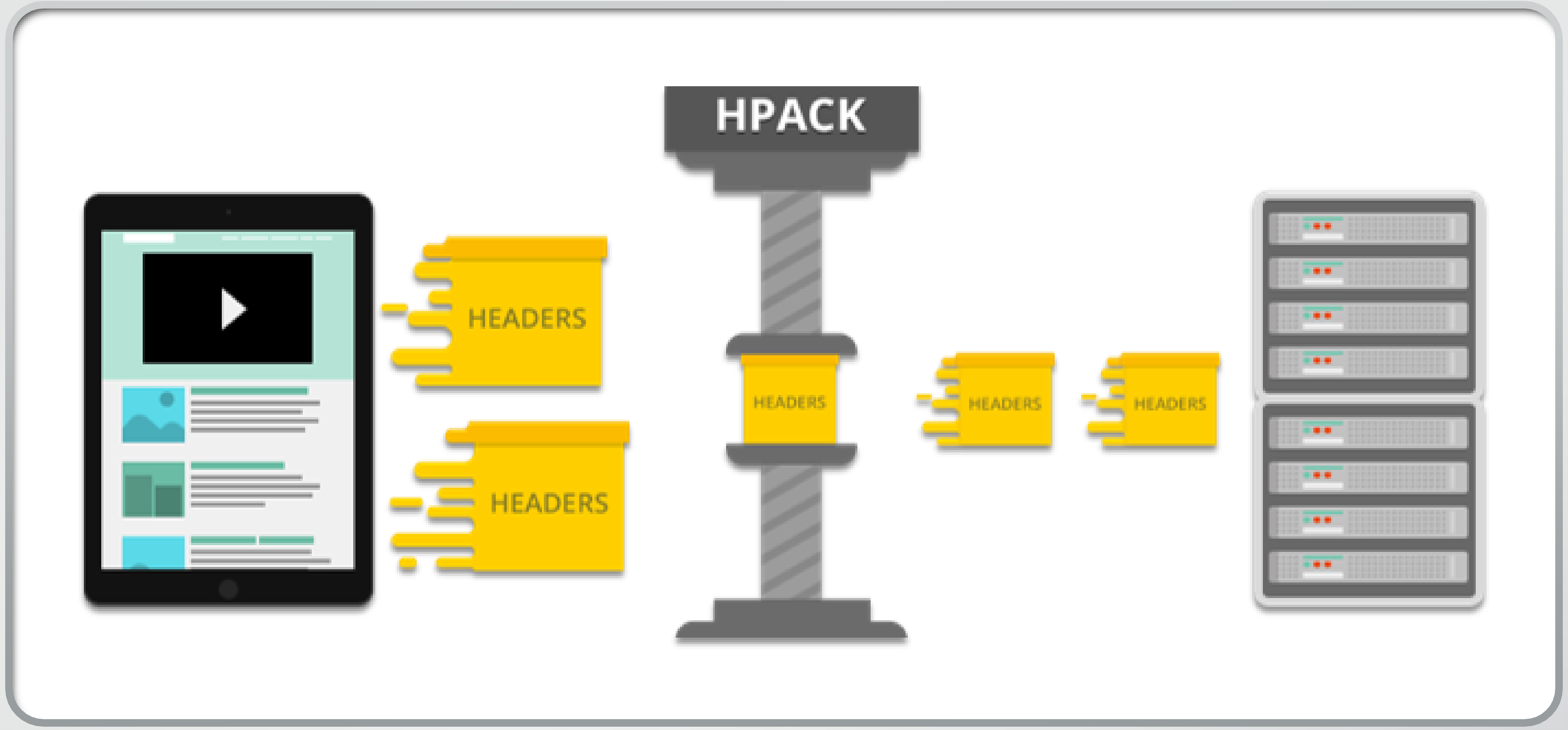
These frames can be sent/received asynchronously



Multiplexing

HTTP/2 Inside: multiplexing





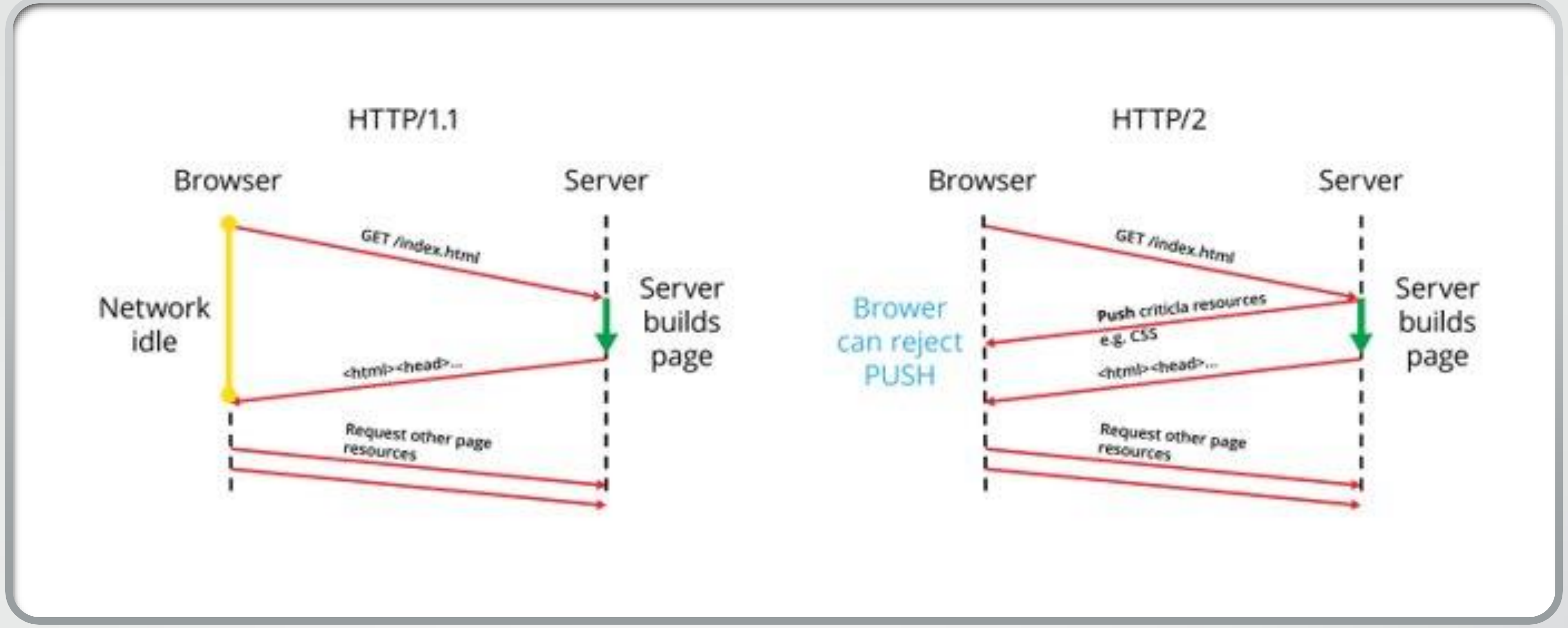
Header Compression

HPACK

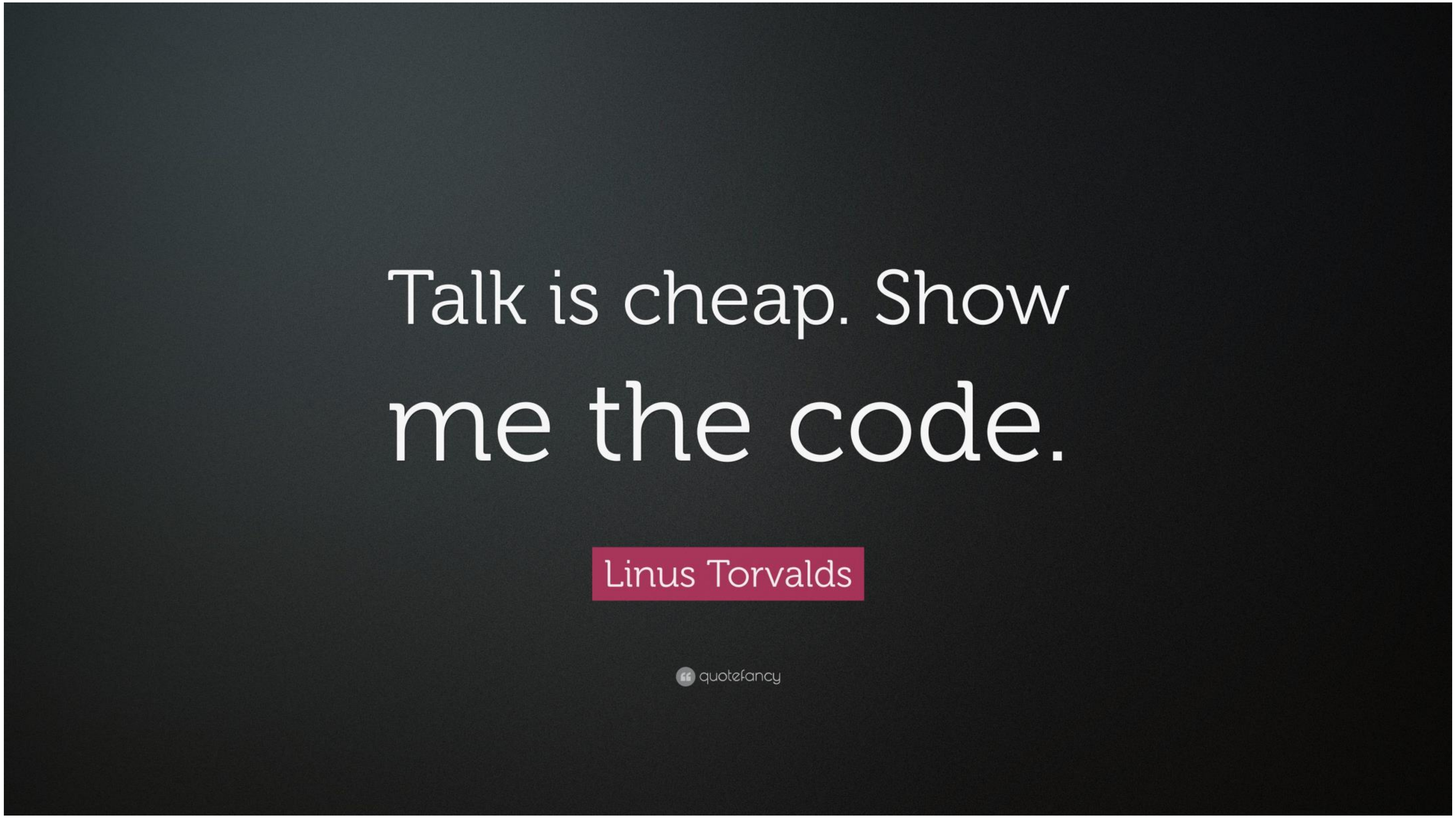
Specialized Algorithm for compressing Headers

Works like gzip

Has a look-up table of ~62 entries from most popular websites



HTTP/2 Server Push

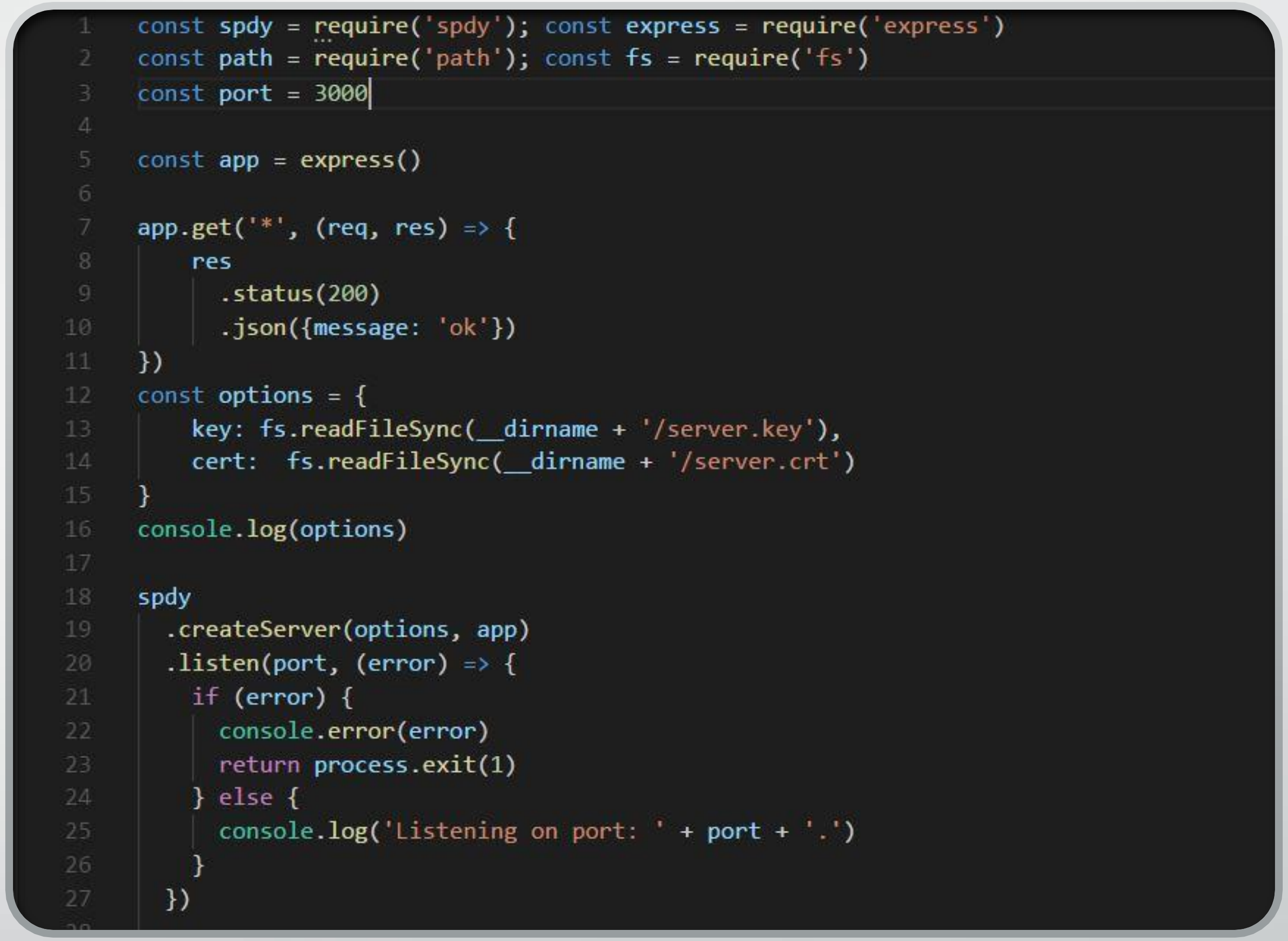


Talk is cheap. Show me the code.

Linus Torvalds

NodeJS Example

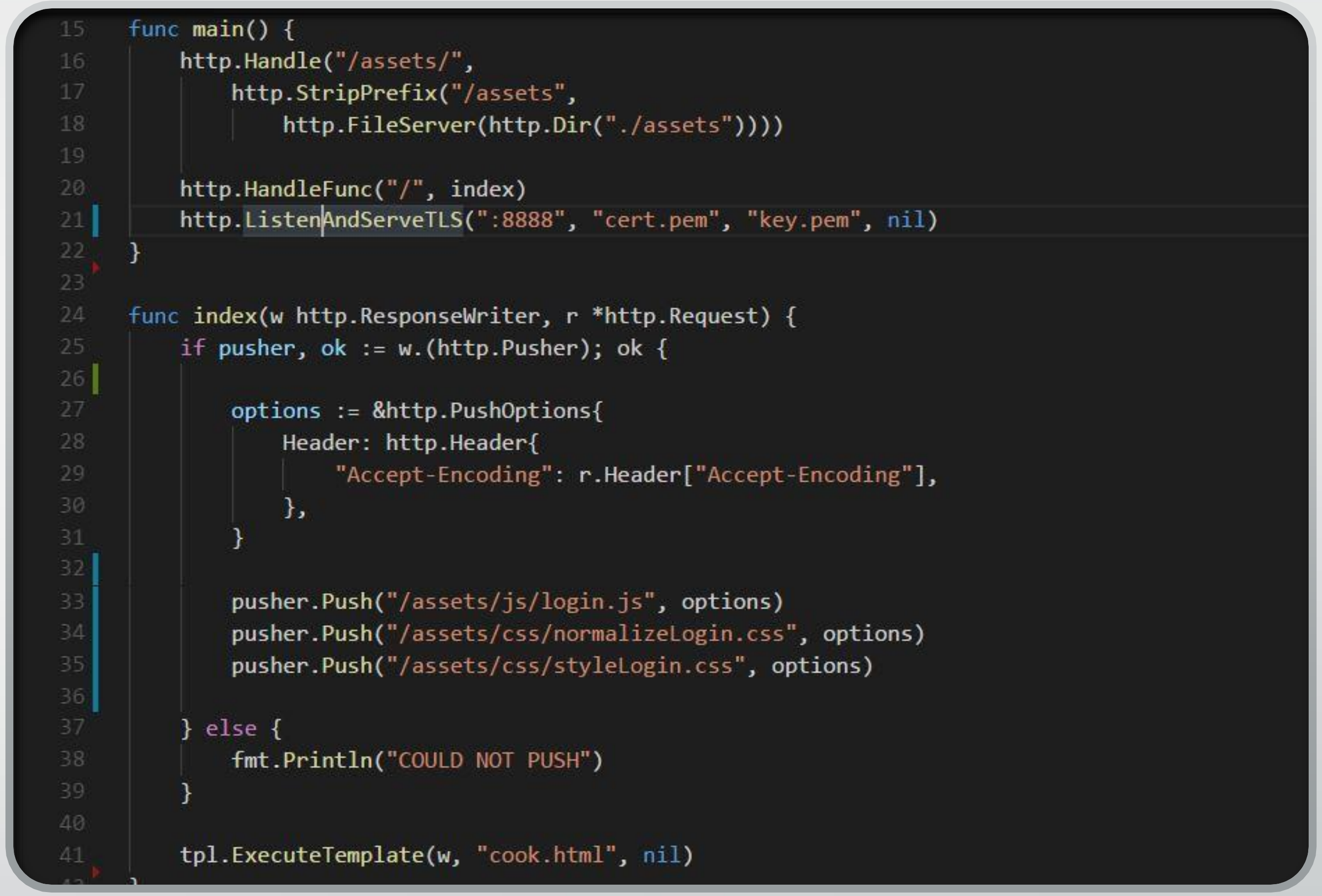
Source: https://bit.ly/2HzYMHj



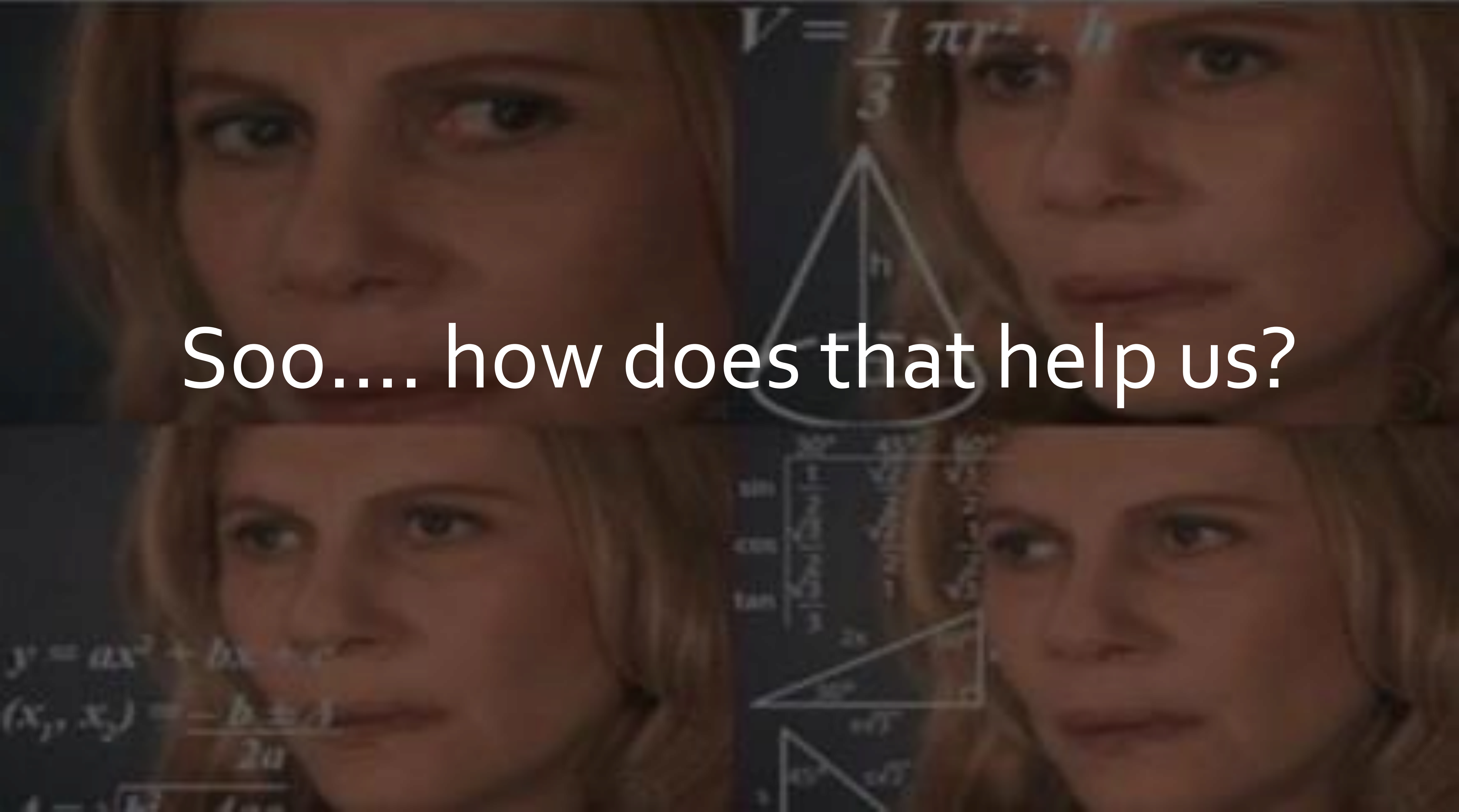
```
const spdy = require('spdy'); const express = require('express')
const path = require('path'); const fs = require('fs')
const port = 3000

const app = express()

app.get('*', (req, res) => {
    res
      .status(200)
      .json({message: 'ok'})
})
const options = {
    key: fs.readFileSync(__dirname + '/server.key'),
    cert:  fs.readFileSync(__dirname + '/server.crt')
}
console.log(options)

spdy
  .createServer(options, app)
  .listen(port, (error) => {
    if (error) {
      console.error(error)
      return process.exit(1)
    } else {
      console.log('Listening on port: ' + port + '.')
    }
  })
```

Go by Example

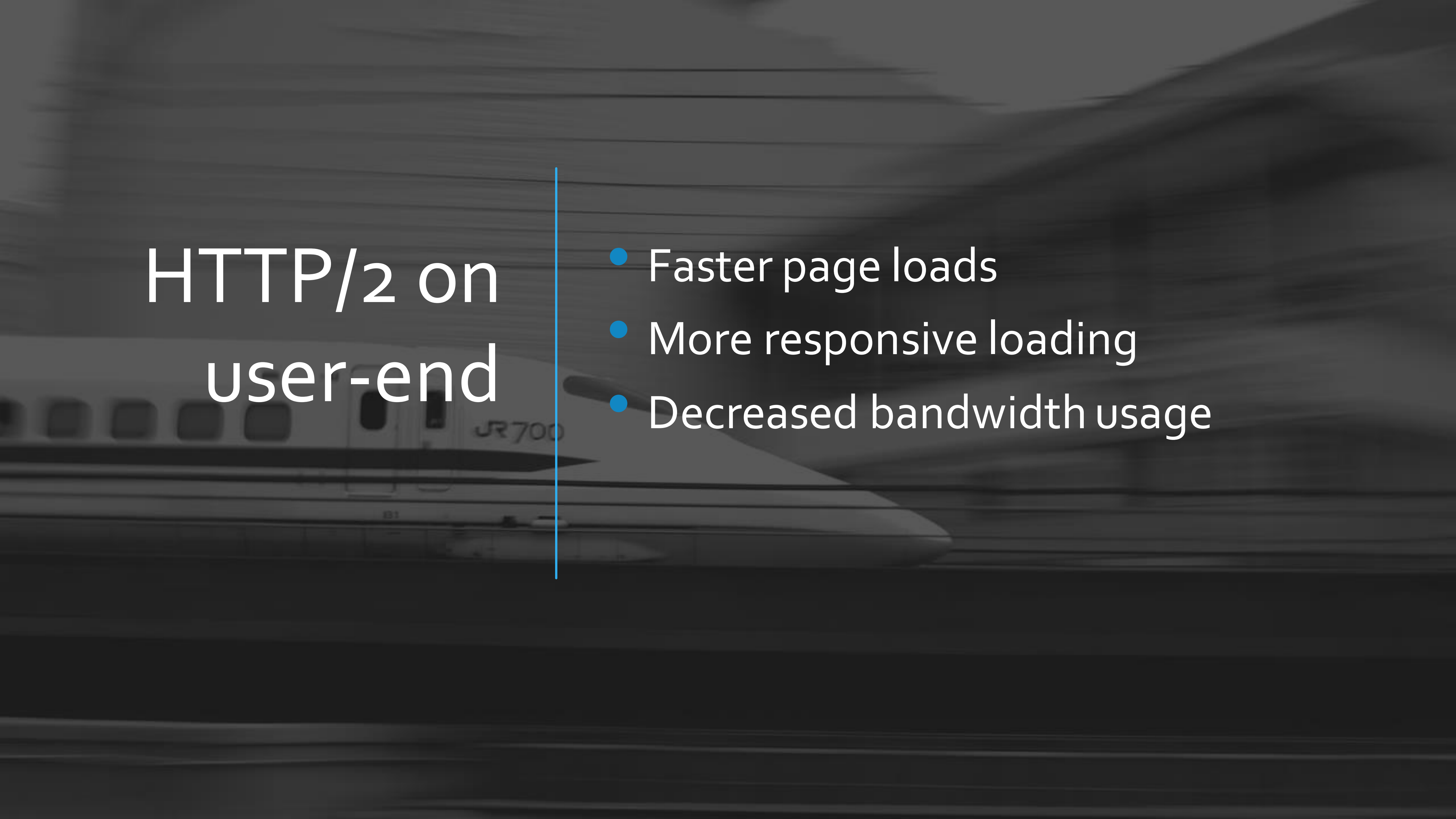
Source: https://bit.ly/2GZyZXQ



```go
func main() {
    http.Handle("/assets/",
        http.StripPrefix("/assets",
            http.FileServer(http.Dir("./assets"))))

    http.HandleFunc("/", index)
    http.ListenAndServeTLS(":8888", "cert.pem", "key.pem", nil)
}

func index(w http.ResponseWriter, r *http.Request) {
    if pusher, ok := w.(http.Pusher); ok {

        options := &http.PushOptions{
            Header: http.Header{
                "Accept-Encoding": r.Header["Accept-Encoding"],
            },
        }

        pusher.Push("/assets/js/login.js", options)
        pusher.Push("/assets/css/normalizeLogin.css", options)
        pusher.Push("/assets/css/styleLogin.css", options)

    } else {
        fmt.Println("COULD NOT PUSH")
    }

    tpl.ExecuteTemplate(w, "cook.html", nil)
```


$V = \frac{1}{3}\pi r^2 \cdot h$
h
sin
cos
tan
$y = ax^2 + bx + c$
$(x_1, x_2) = \frac{-b \pm \Delta}{2a}$

Soo.... how does that help us?



HTTP/2 on user-end

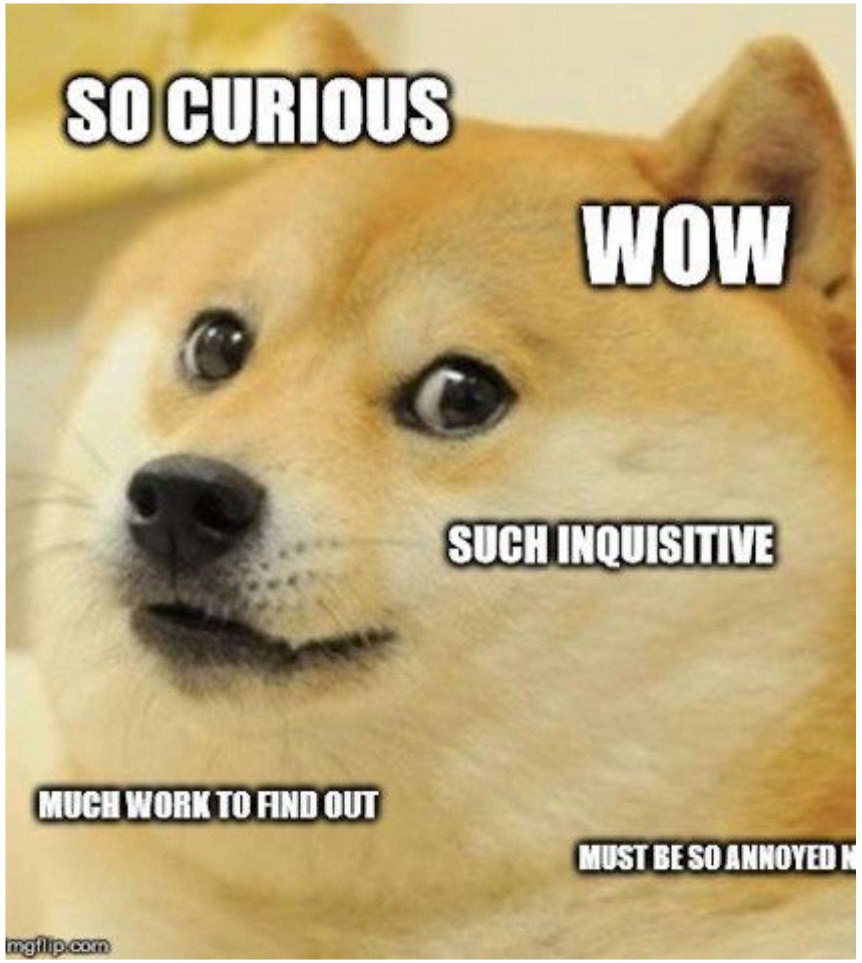
- Faster page loads
- More responsive loading
- Decreased bandwidth usage

HTTP/2 on developer's end

- No need for HTTP/1.X *"hacks".*
- Decreases CPU & Bandwidth usage on server end.
- Decreases overall server cost.

Curious about HTTP/3?

IT MIGHT JUST HAPPEN SOONER THAN IT TOOK US TO MOVE FROM HTTP/1.1 TO HTTP/2



Thank You

“All things being equal, the simplest solution tends to be the best one.” — William of Ockham