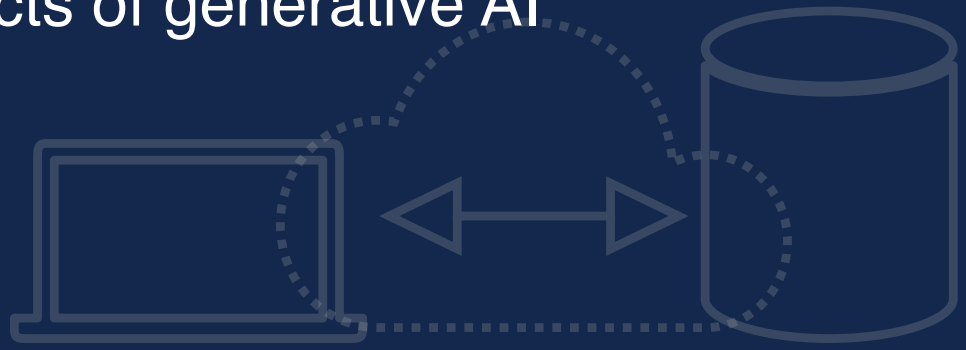
CSE 135
Online Analytics

The Class Formerly Called Web Server Technologies

Thomas A. Powell
tpowell2@ucsd.edu

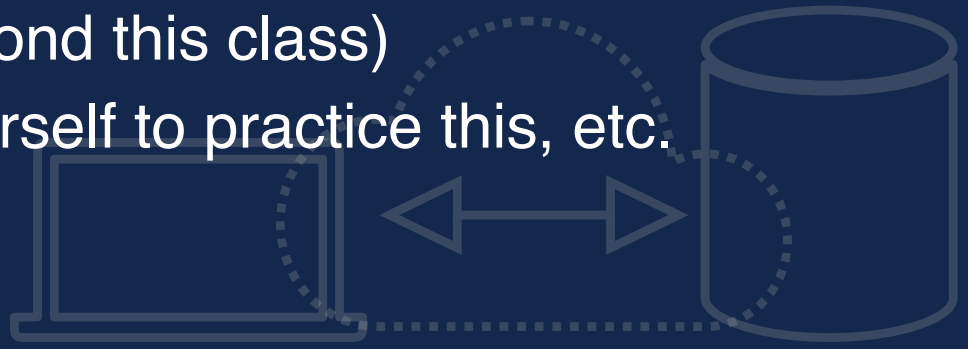
Agenda

- Brief Intros
- Class Details
  - History, Intention, etc.
  - Canvas
  - Slack Overview
- Academic issues, engagement, and in-person learning
- Embracing and Studying the effects of generative AI
- Introduction Lecture



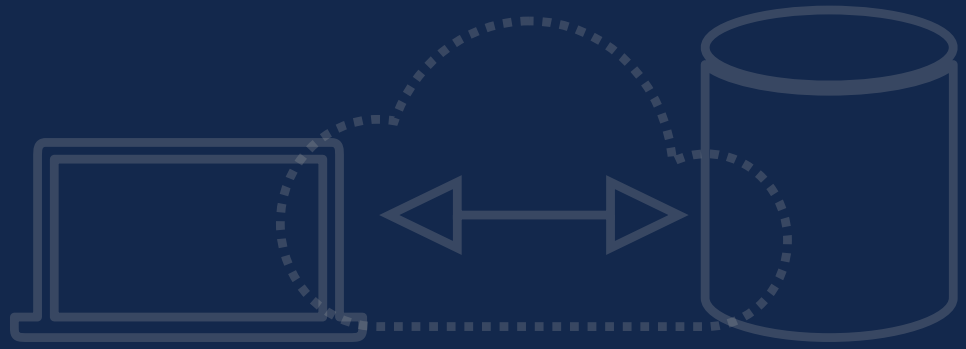
Interacting with the Teaching Staff Tips

- Don’t be too shy, we can’t help you if you don’t interact with us
- We will give you many ways
  - Prof will hang out after class every day, DMs will be open, office hours will be provided, chat in class, etc. etc.
- Use Slack well
  - Threads, Reminders, FAQ posts
- **Don’t be transactional** please!
  - Humans are on both ends of the effort here.
  - Win with respect! (Tip well beyond this class)
    - Timing, optics, allowing yourself to practice this, etc.



Course Goals

- This course aims to present both a technical discussion of web based data collection and use as well as help student's gain an awareness of possible good and bad outcomes of this activity.



Goals After Taking the Course

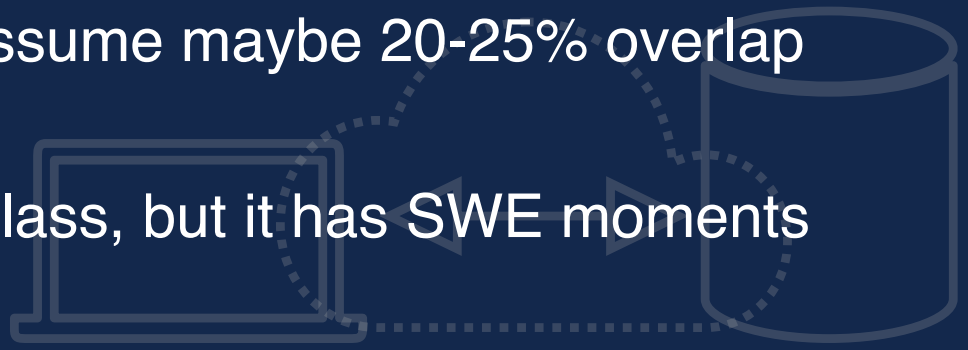
- Understand the basic architecture of web sites and apps
- Gain familiarity with core web tech including HTML, CSS, JavaScript, HTTP, Web Browsers, Web Servers, server-side programming with NodeJS or PHP, SQL or NoSQL databases and related patterns*
- Understand web data collection including access logs, sniffing, collecting scripts, bots and various identification ideas
- Understand the lifecycle of data driven decisions online
- Learn the basics of data presentation with charts
- Gain an awareness of data collection concerns and practices

Realistically this is not a deep dive on any web tech, but we emphasize server web tech in 135. A deep dive of client web tech is found in 134.

Course Non-Goals

- The course cannot do everything, and given the assumptions students make and the poor course description, we need to be clear about what won't be covered.

- The course will NOT
  - Make you a full stack dev (**but hopefully full stack aware!**)
  - Present an in-depth discussion of database tech
  - Address large-scale web data handling
  - Serve as a substitute for 134, assume maybe 20-25% overlap with far less depth
  - Be like 110 - this isn't an SWE class, but it has SWE moments

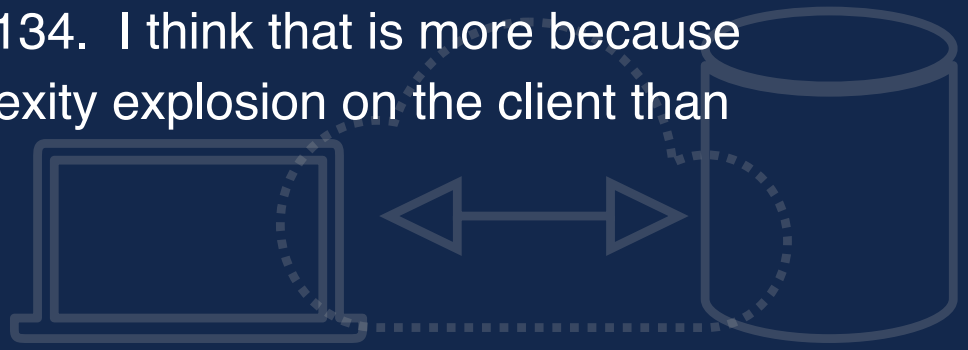


FAQs

- Q: Can I take this course without 134?
- A: Yes, but you get more out of it with 134, and it will be much harder

- Q: I want to do backend. Why do we need HTML, etc.?
- A: Unless you are just doing endpoint stuff, you can't be so separated from the client. For analytics, it provides context as well!

- Q: Is this class challenging?
- A: It can be, but it certainly isn't thought of as hard by many who gave the class its time and respect. In fact, the feedback I have received suggests students consider this course easier than 134. I think that is more because of the nature of server-side and the complexity explosion on the client than some sort of course design issue.



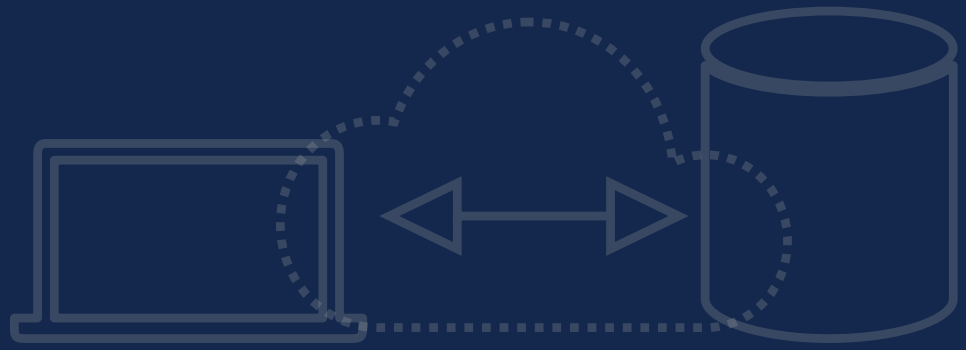
FAQs

- Q: I know some of this already, I want more! Can I have more?
- A: I'll present the core content, if you want more I probably have it come see me or I can certainly point you to it.

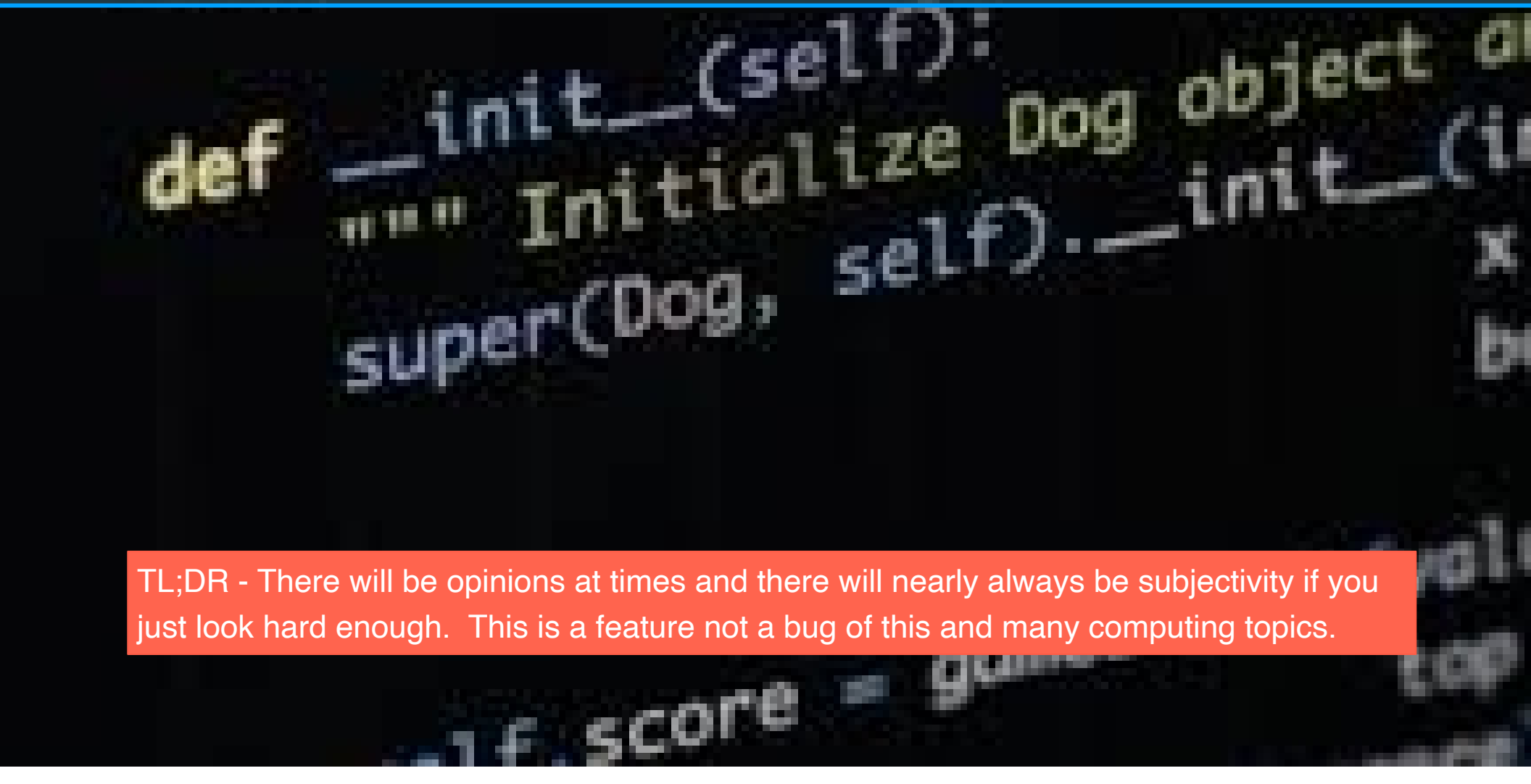
- Q: Are you gonna get all <insert some feeling word> about the (big)web tech, data collection, privacy, security, data use, viz, … ?
- A: Yes. If I didn't, you shouldn't take this class. Just read a Wikipedia entry , let ChatGPT spit some code, or whatever; it likely would be more direct.

🤔 *Reliance on data without contextual awareness and careful thinking in my view is more dangerous than it has ever been in my lifetime. Yes this is my opinion obviously, but I am not seeing much to shake the opinion quite the opposite actually.*

Foundations



A purely technical introduction of any topic can disastrous if it trains us inadvertently to think everything is solved. The certainty and safety implied stifles discovery and growth.



TL;DR - There will be opinions at times and there will nearly always be subjectivity if you just look hard enough. This is a feature not a bug of this and many computing topics.

Online Data Collection

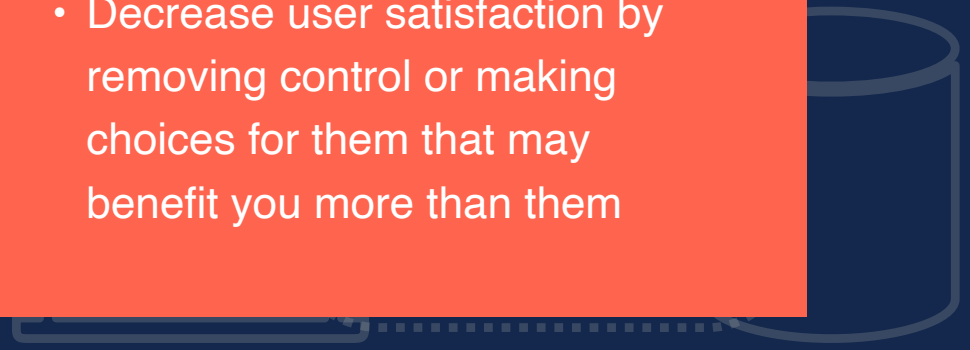
- The Internet allows us to collect an enormous volume of information about users.

Pros

- Improve user satisfaction by monitoring for errors and problems
- Provide customization and personalization to meet individual dates and preferences
- Guide development practices to better suit users and conditions

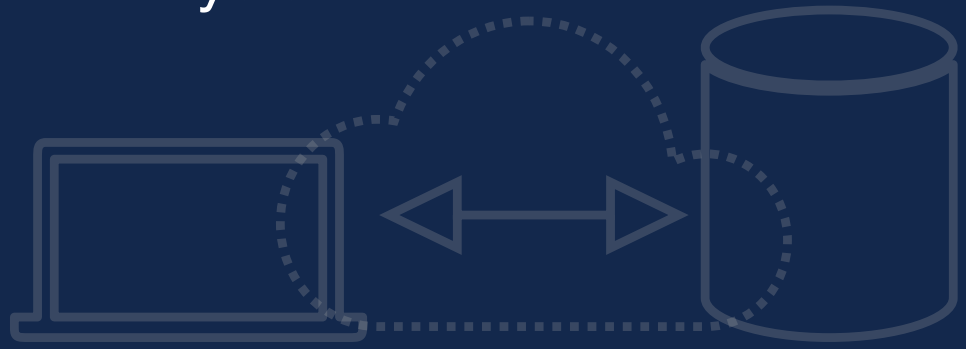
Cons

- Gather data without user consent or awareness
- Abuse collected data
- Decrease user satisfaction by removing control or making choices for them that may benefit you more than them



A Rough General Analytics Flow

- Form questions
- Collect Data
  - Sanitize and Refine
- Analyze Data
  - Raw
  - Visualized
- React
  - Ask more questions or act upon analysis



Yet some think differently

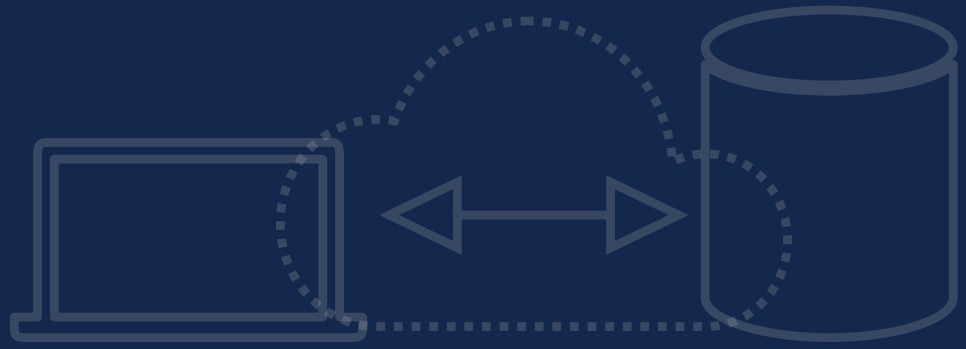
- They believe you collect data and figure out questions later
- They even believe that insights arise from the data
- I really find that belief an except not the rule
- I also find that too much data leads to over confidence, obfuscation and even depending on how messy it is deeply deeply wrong answers
- Example
  - Page view metrics and acceptability story

Morale Spoiler: Be very careful, ask questions, think, and be skeptical like a (Computer) **Scientist** - Big Data != Big Wisdom in fact it often leads to Big Idiocy

Online Data Collect Approaches

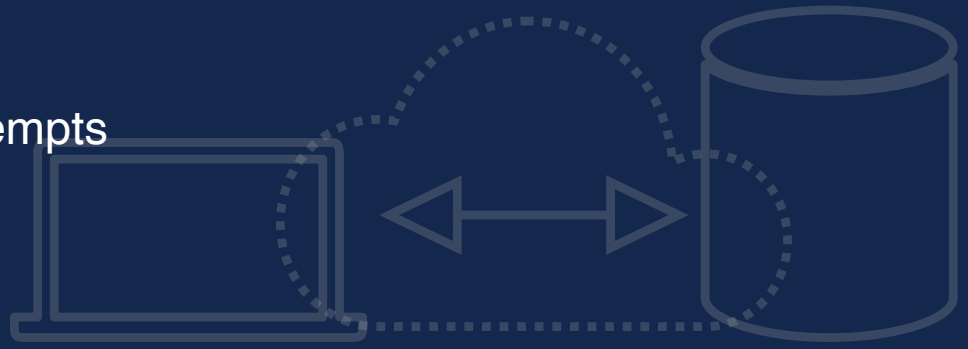
- Data is collected primarily in two ways:
    1. Actively - you are directly asking for it
    2. Passively - it is being provided whether you asked for it or not

- We might also express data collection as either using some agent (often just a script) or more agent-less and built-in to the protocol or delivery environment
    - Rough trade-off here is you tend to find more information collected via agent based solutions than agent-less, but agent-less require little to no setup nor configuration.



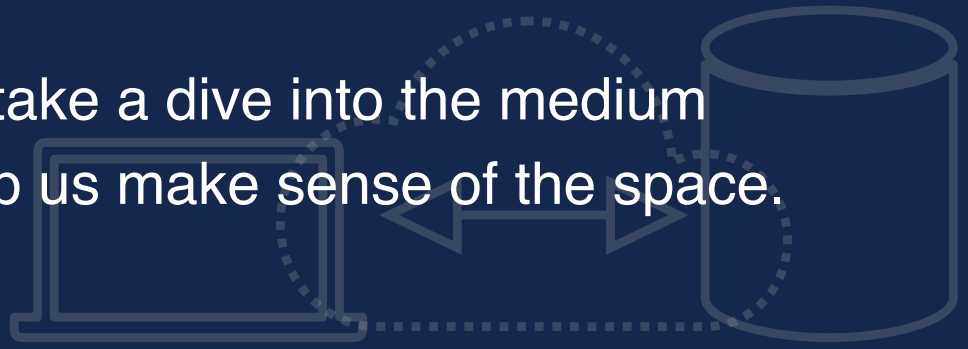
Internet Data Collection Uses

- Usage ranges from innocuous to questionable
  - Example: Tracking error rates and correcting problems as they emerge or collecting techno graphics to drive requirements planning
  - Example: Measuring advertising performance & adjusting to improve click rates
  - Example: Building a social profile of user based upon various signals to feed them content for more engagement
  - Example: Collecting data to sell to others
  - Example: Collecting data for intrusion attempts

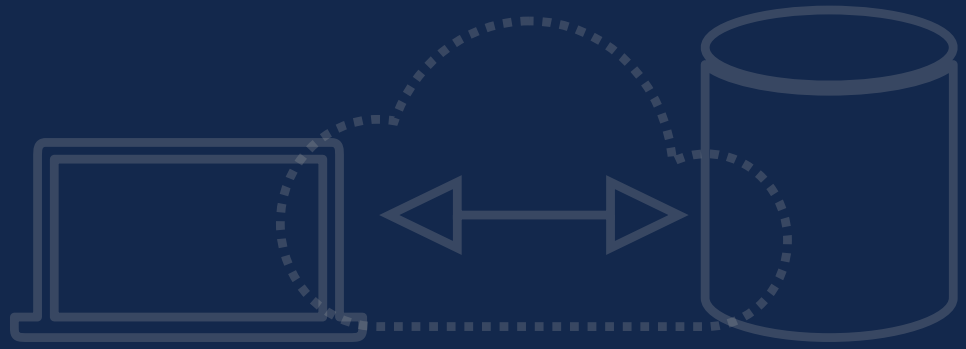


Where and How

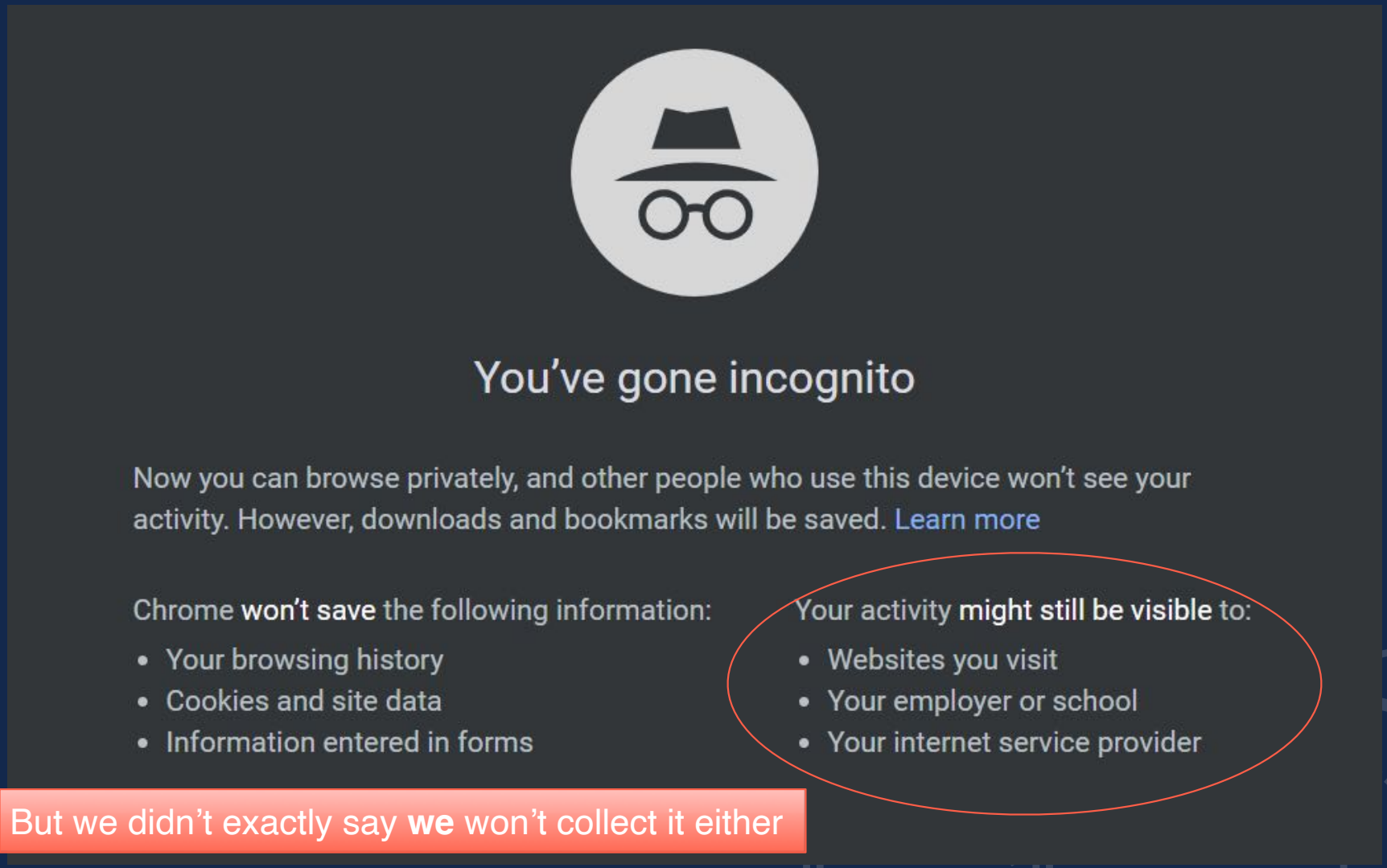
- Data can be collected in three major zones
  - Client Side (your device, browser, etc.)
  - Network (intercepting and service level)
  - Server Side (logged by a server you access)
- The client-side has the richest data, the network usually has the broadest reach collection wise, and the server-side has the most inaccessible from a hiding point of view unless you don't use the server
- To understand the how we need to take a dive into the medium and explore mental models that help us make sense of the space.



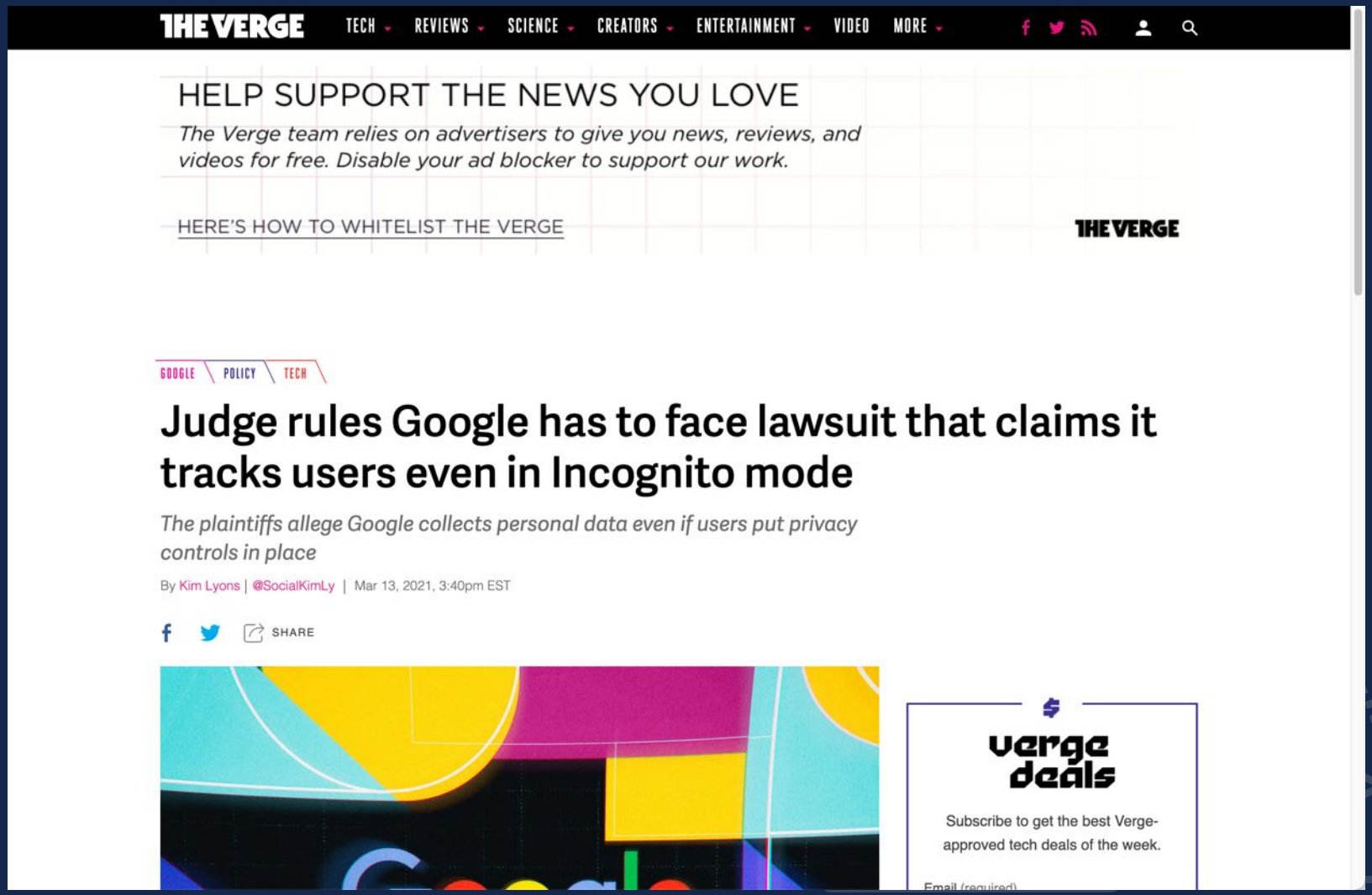
Let's Get "Real"!



Alert: I'm About to Do Something Interesting!



Derp! Things aren't as private as maybe you'd hoped



Fingerprinting Preview

IEEE.ORG IEEE XPLORE DIGITAL LIBRARY IEEE STANDARDS MORE SITES

IEEE Spectrum

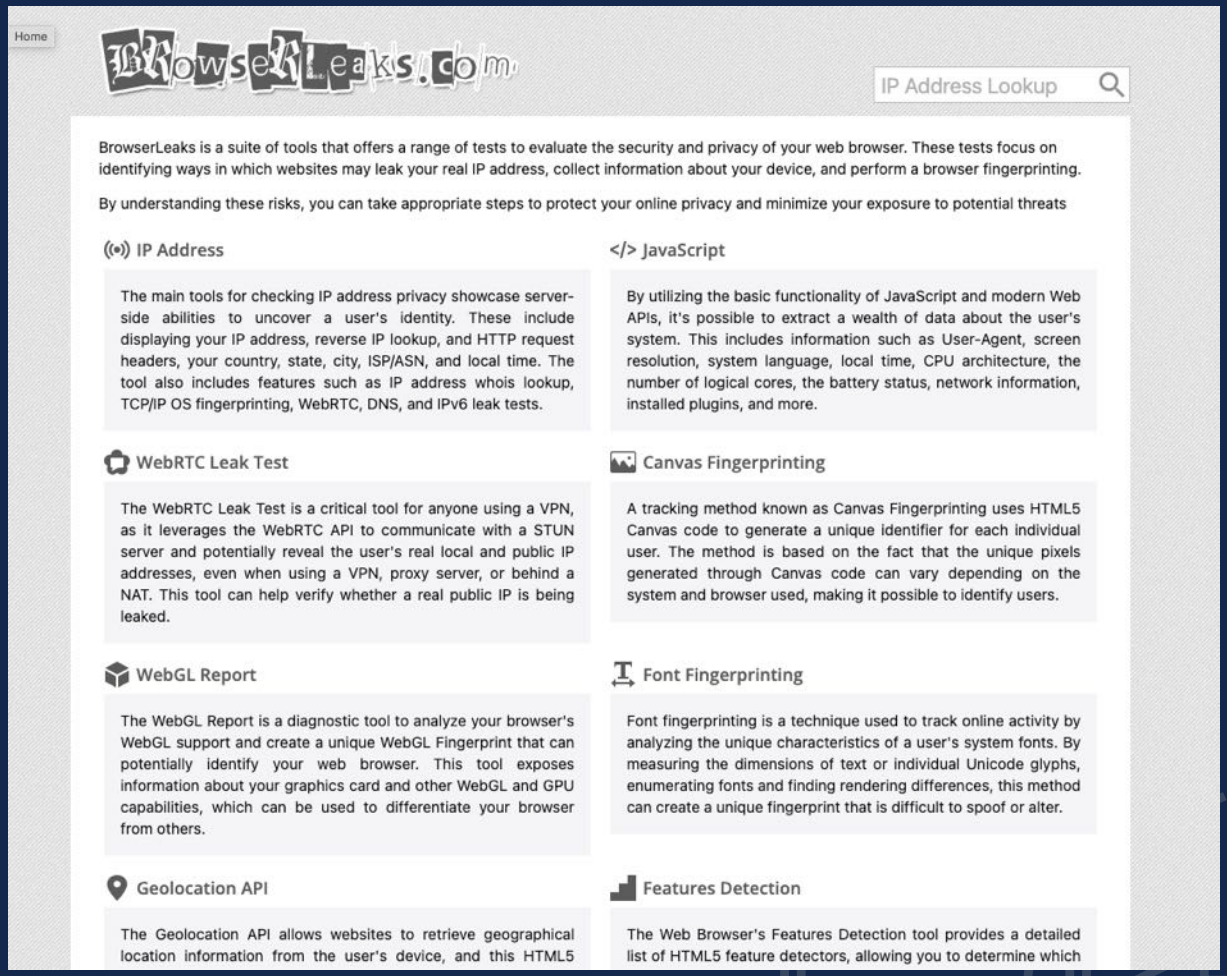
Type to search

NEWS | TELECOMMUNICATIONS

Browser Fingerprinting Tech Works Across Different Browsers for the First Time › Web browsing just got a little less anonymous: New browser fingerprinting software correctly identifies 99% of online users even if they switch browsers

BY AMY NORDRUM | 24 FEB 2017 | 3 MIN READ |

Lots of Data to Fingerprint With…



https://browserleaks.com/

Fingerprint Demo

Demo at https://fingerprint.com/

Some reference material -

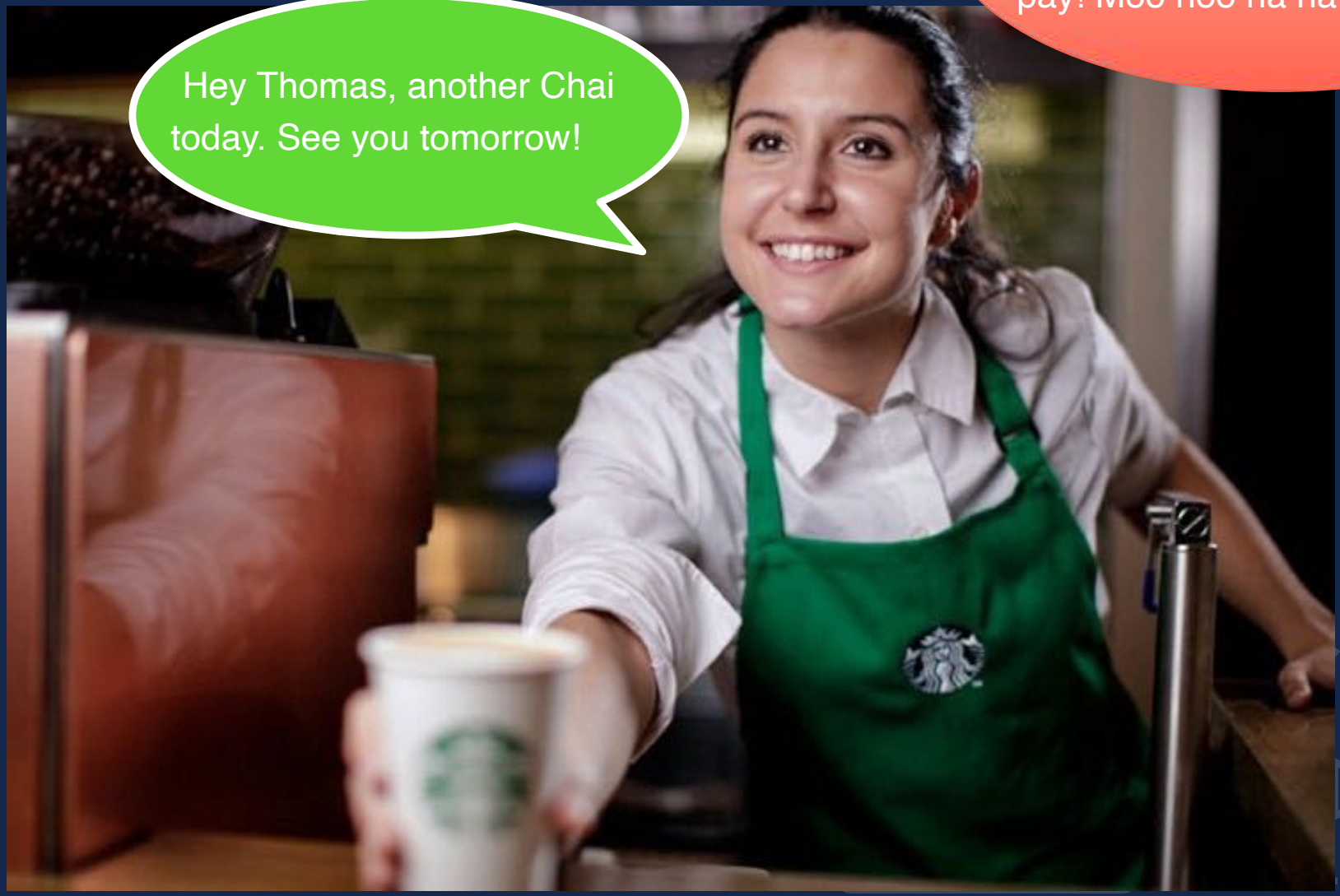
https://coveryourtracks.eff.org/

https://www.bitestring.com/posts/2023-03-19-web-fingerprinting-is-worse-than-I-thought.html

https://css-tricks.com/css-based-fingerprinting/

Sanity Check: Starbucks

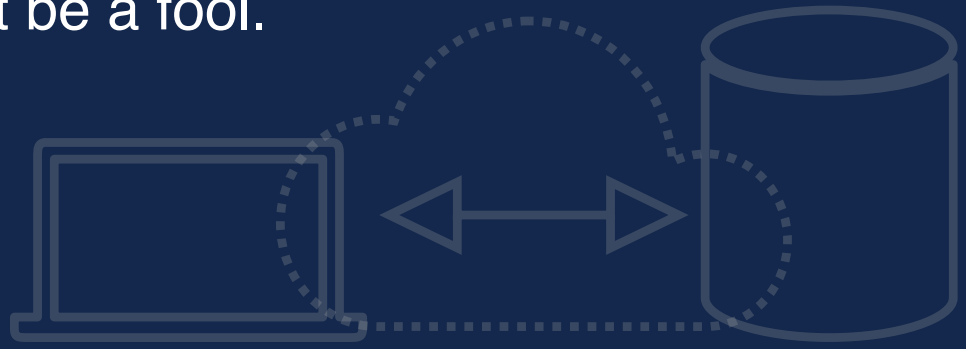
Sanity Check: Evilbucks



Morale

- Data collection isn't evil per se
- Fingerprinting isn't either
- The problem comes from the relationship expectations, lack of disclosure, and yes far too often actual abuse

- To be fair, online much of the problem comes with our lack of admitting that things that are free really aren't free! What did you expect if they give you something they gotta get paid, quite often by your privacy and attention! Don't be a fool.



Online Free is Rarely Free

- Part of the challenge with the surveillance economy and privacy issue is they are realistic and maybe appropriate reactions to the “free” expectations

- If you get something for “free”, it probably isn’t free. Things cost and unless there are other non-economical driven motives people are likely needing to get paid somehow typically via data collection and selling, advertising, or both.

Pro Tip - Always Check Your Thinking

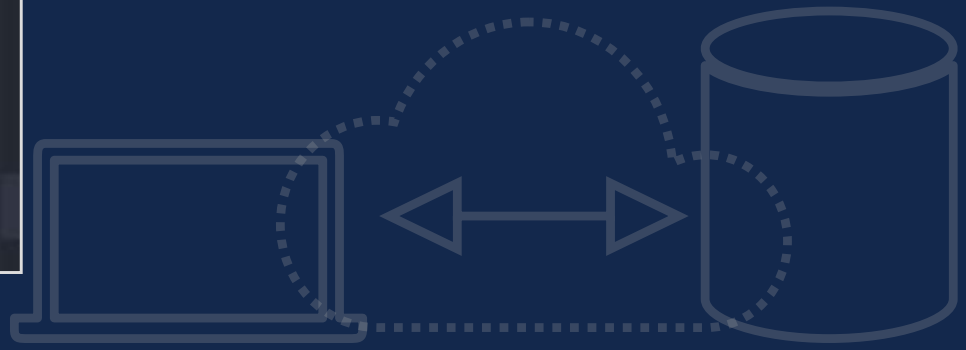
ThomasAPowell boosted

During one of my lectures my professor told us that "intelligent people are not less likely to have illogical beliefs, they're just better at rationalising their illogical beliefs" and I think about that a lot

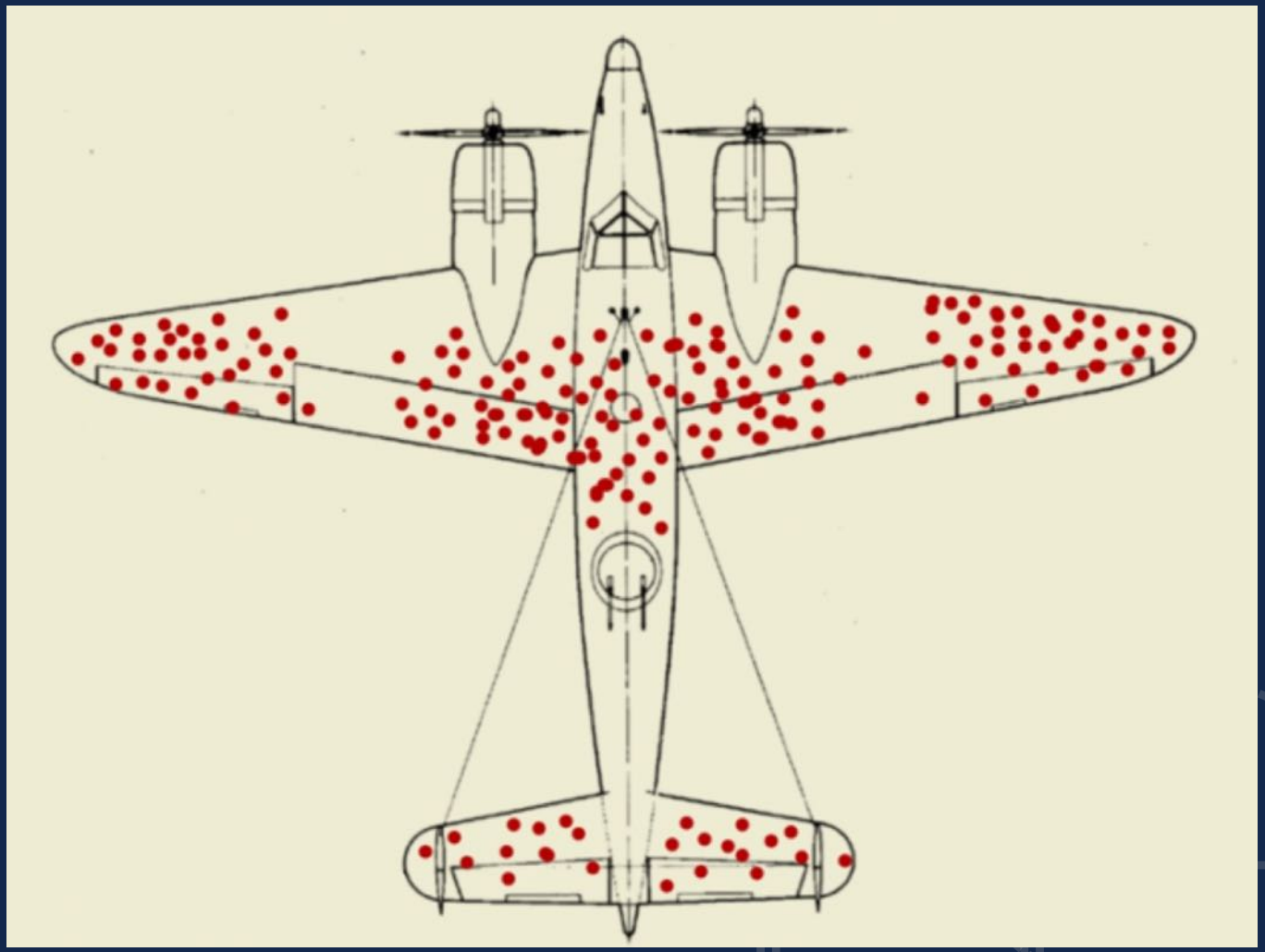
Edit: Just checked the exact quote. More specifically he said that "Intelligent people are good at rationalising their beliefs, including those at which they arrived irrationally"
Wanted to make sure that I didn't get it wrong ^^'

Related Inconvenient Follow-on

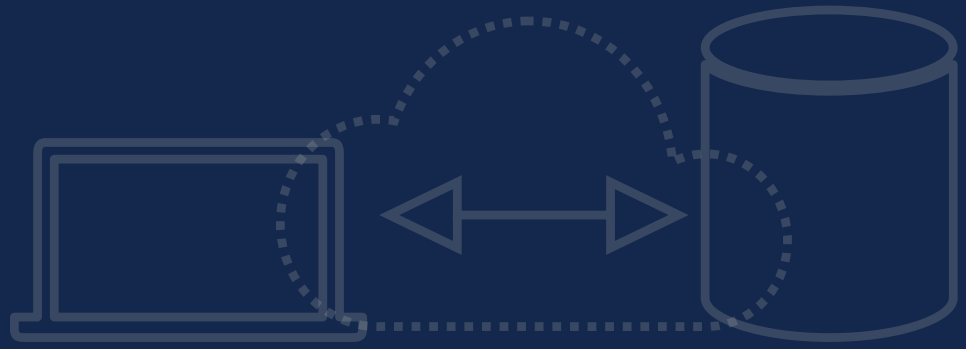
Add in this issue and “big data” can often lead to big face palm!



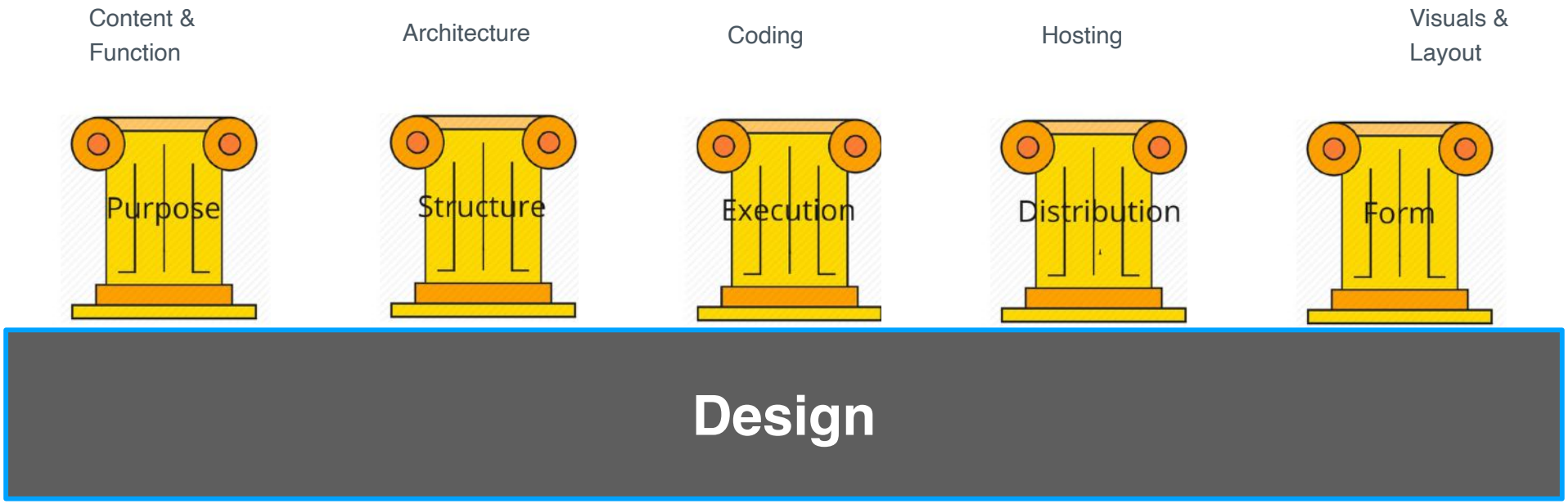
Constant Danger - Data isn't truth, it's just data!



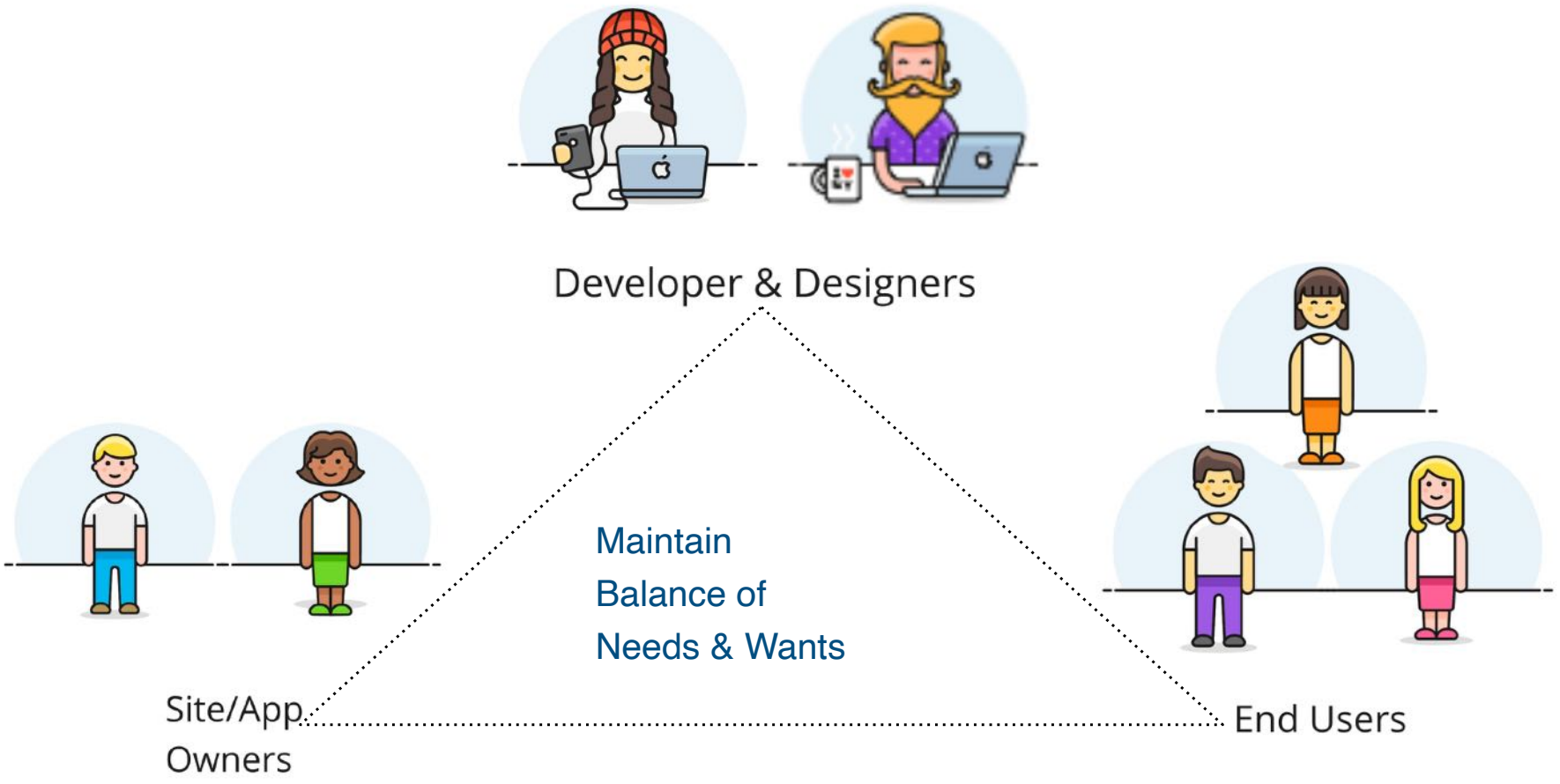
Web Tech Mental Models



Prof's 5 Pillars Model



Three Participant Groups

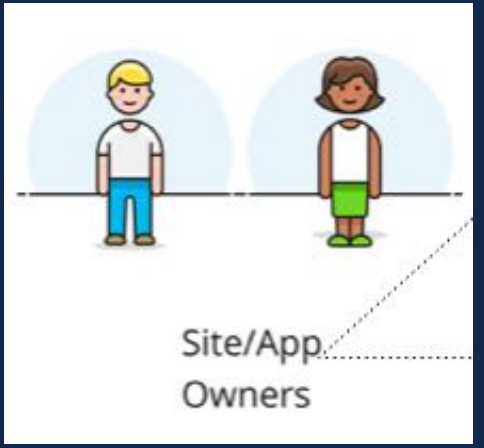


Participant Groups - What do we want to know?

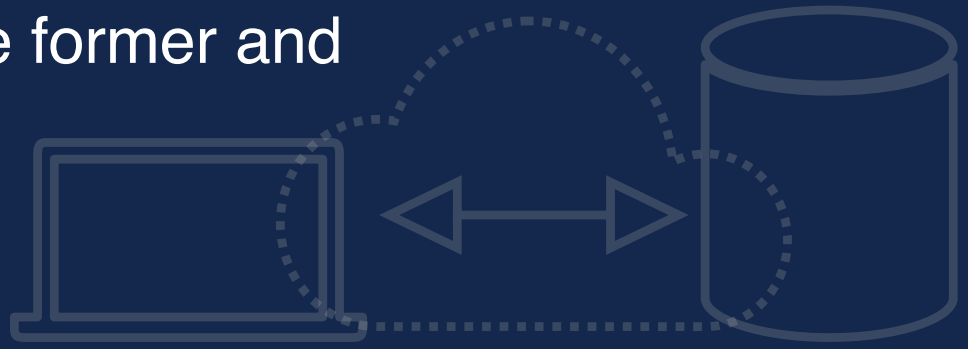
- Is my app working right now?
- Are errors being thrown?
- Is performance within tolerance?
- What type of technical characteristics does my ACTUAL audience exhibit versus what I am supporting?
  - Specifics
    - Browsers and versions, tech availability, CPU power, screen and display characteristics, etc.
- What is the range of broad personal characteristics of my ACTUAL audience? (This addresses a11y and i18n)
- What can I see about a user's understanding and enjoyment of my software or site?

Participant Groups - What do we want to know?

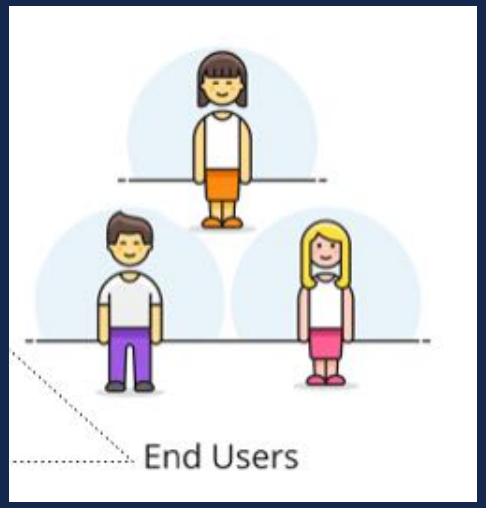
- What types of things lead to “conversions”?
- What types of things don’t lead to “conversions”?
- What paid efforts perform and result in more income than cost?
- What is my revenue or conversion rate?



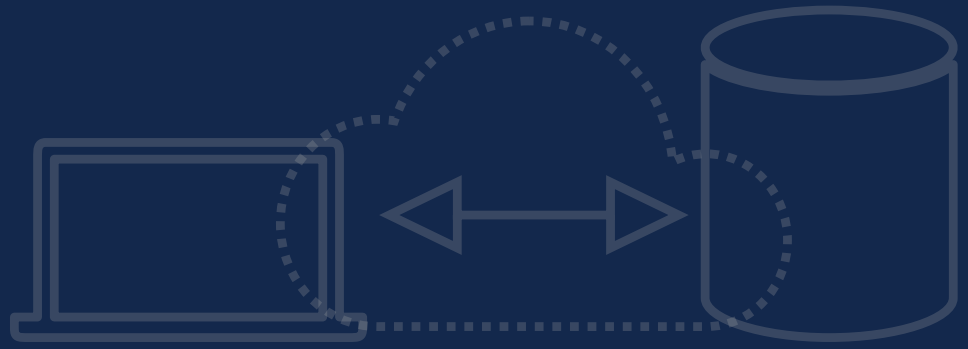
- TL;DR - what are my costs and what is my revenue and how can I lower the former and grow the later?



Participant Groups - What do we want to know?



- What data are they collecting about me?
- Are they doing this with/without my consent?
  - Can I opt out of the data collection?
- Can I remove past data (right to be forgotten)?
- Do they keep my data safe if I decide to trust them with it?
- Do they sell my data?
- <insert whatever privacy or data abuse worry you have>



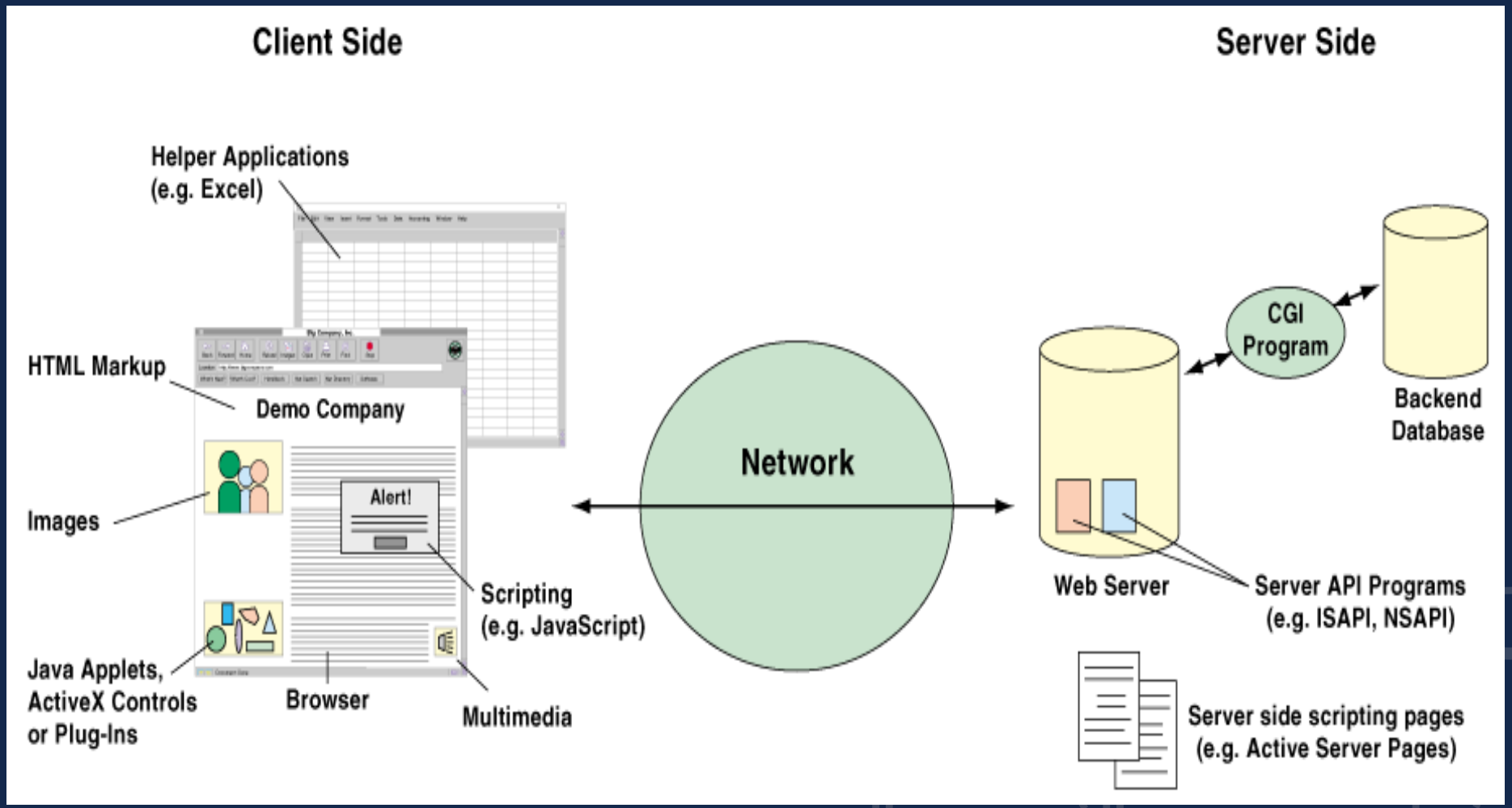
A Fourth Group Enters the Chat… Government

- Government: Are the parties collecting data like this doing so in a responsible and regulation / law-abiding way?

- Examples: GDPR (Europe)
  - https://en.wikipedia.org/wiki/General_Data_Protection_Regulation

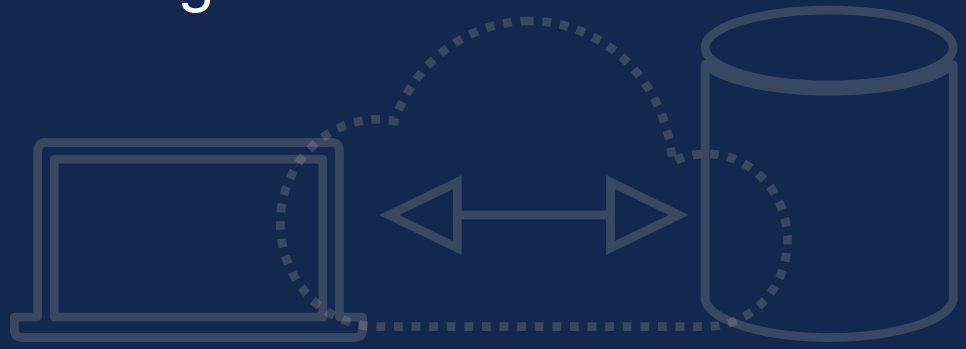
- CCPA (California)
  - Consumer rights to know what personal information is collect and right to request deletion and opt out of sale, must have a privacy policy in the open. (https://en.wikipedia.org/wiki/California_Consumer_Privacy_Act)

The Vast Web Medium

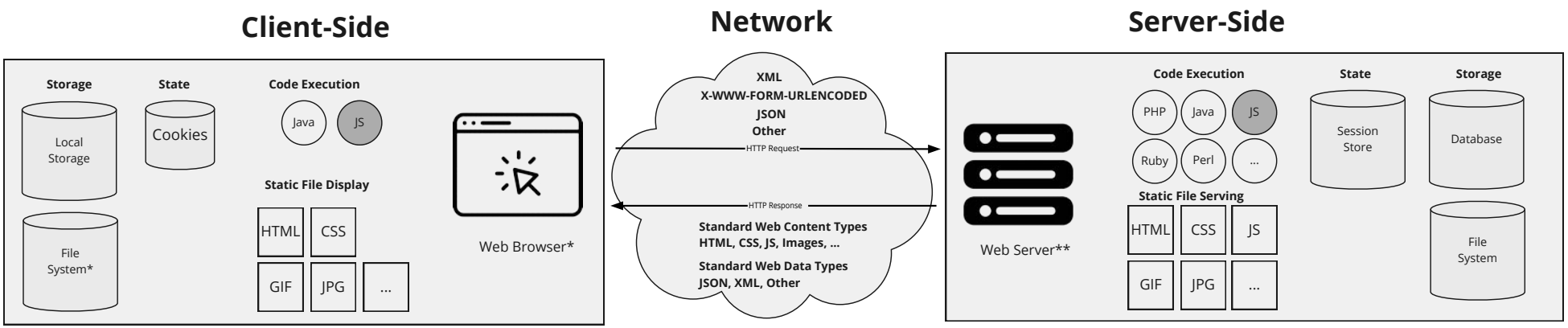


Initial Observations

- There seem to be many moving parts on both sides of the model (client and server)
- Generally, many moving parts = likely system fragility so we need to watch ourselves
- Interstingly we will see some symmetry to the model in terms of what runs where and what it's goals are



Medium In Context



We notice that some things only make sense one place or the other, but somethings are bit more fluid and might work on both sides…but one thing is very clear there is A LOT!

‘Toolbox’ Over Flows

Technologies & Patterns

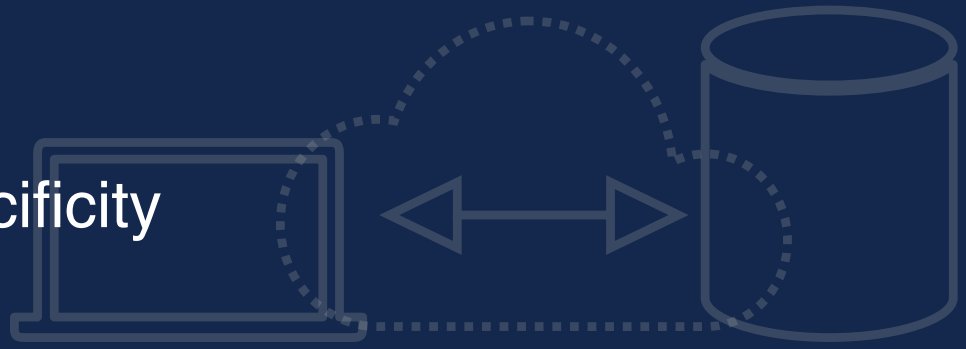
“Cloud”
SaaS, PaaS, IaaS
CRUD
MVC
Ajax
MVVM
SPA
REST
PWA
GraphQL
Traditional Server
Side Apps
NodeJS
Apache
“Serverless”
SSR
Cloud Functions
SQL
NoSQL
Apache
“BigData”
Microservices

HTML
CSS
Images
Fonts
Multimedia
(Video/Audio)
Binaries
(Plugins, NaCl)

JavaScript
ES5
ES6
TypeScript
Libraries &
jQuery
WebViews
Frameworks
VueJS
React
Electron

Trade-off Dimensions

- Where the code runs – client vs server
- What kind of network – public vs private
  - Code Complexity/Effort (simple/easy <-> complex/hard)
- Performance (Lower <-> Higher)
  - Often relates to effort / complexity
- Platform Specificity (Specific <-> General)
  - Often discussed as portability
- Coupling (Tight <-> Loose)
  - Often related to platform specificity



Web Programming Toolbox 2.0

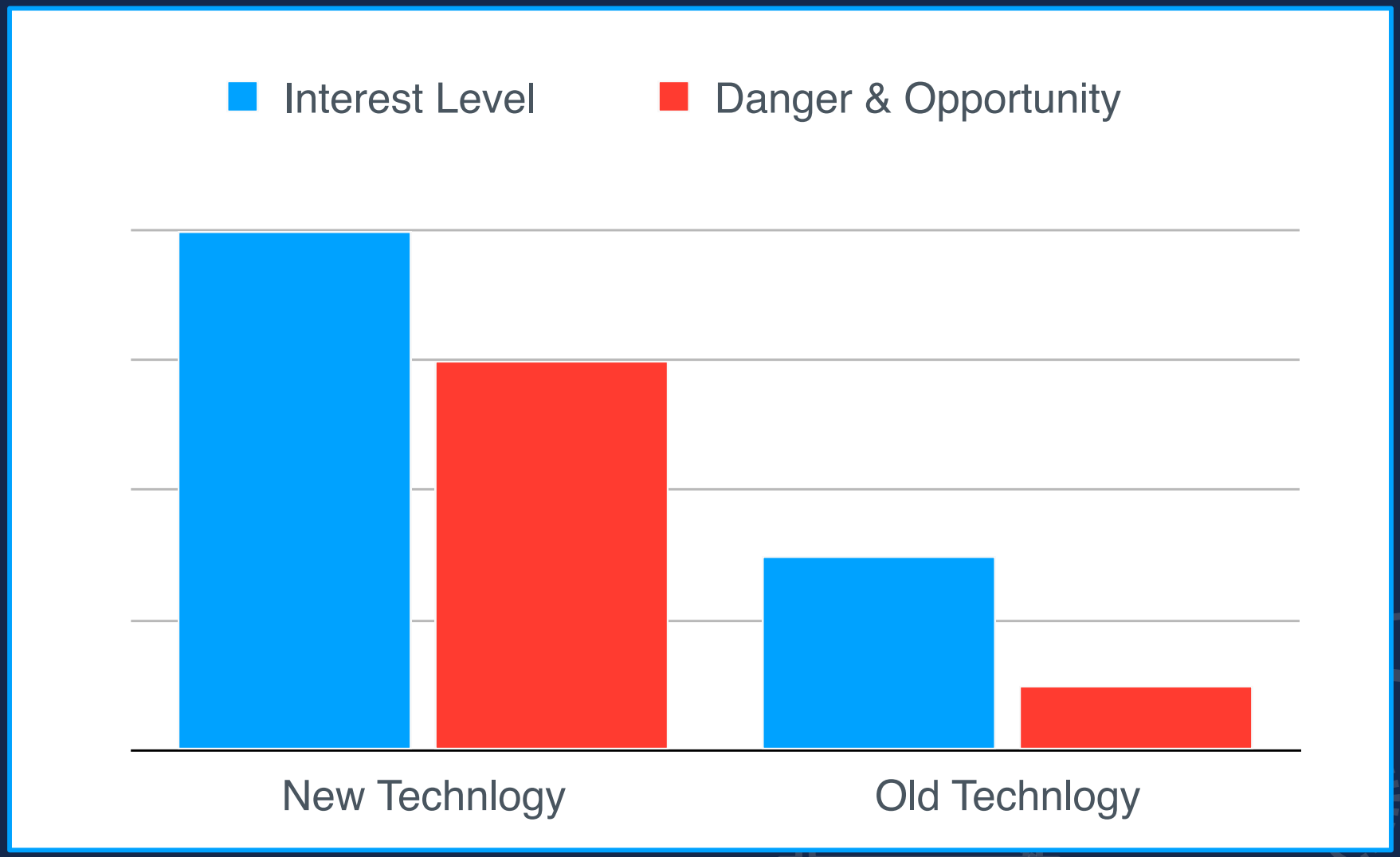
Client Side	Server Side	Characteristics
Browser Extensions - Netscape plugins - ActiveX Controls - NaCl - Chrome Extensions and Similar	Server API Programs - Apache Modules - ISAPI Filters and Extensions - Ngnix Dynamic Modules	Proprietary Coupled Complex Fastest
Java Applets	Java Servlets	Portable Semi-Coupled Complex Fast(ish)
Client Side Scripting 1st Gen -JavaScript ~~-VBScript~~ 2nd Gen ~~-ASM~~ - WebAssembly aka WASM	Server Side Scripting 1st Gen - Classic Active Server Pages (ASP) - ColdFusion - PHP - Ruby 2nd Gen -JSP - ASP.NET	Portable Semi-Coupled Less Complex Slower
Helper Applications - Scripted - Compiled	CGI Programs - Scripted - Compiled	Portable / Not Not Coupled Complex / Simple Slow

PSA About Our Proper and Too Often Abused Desire for Solution Simplicity

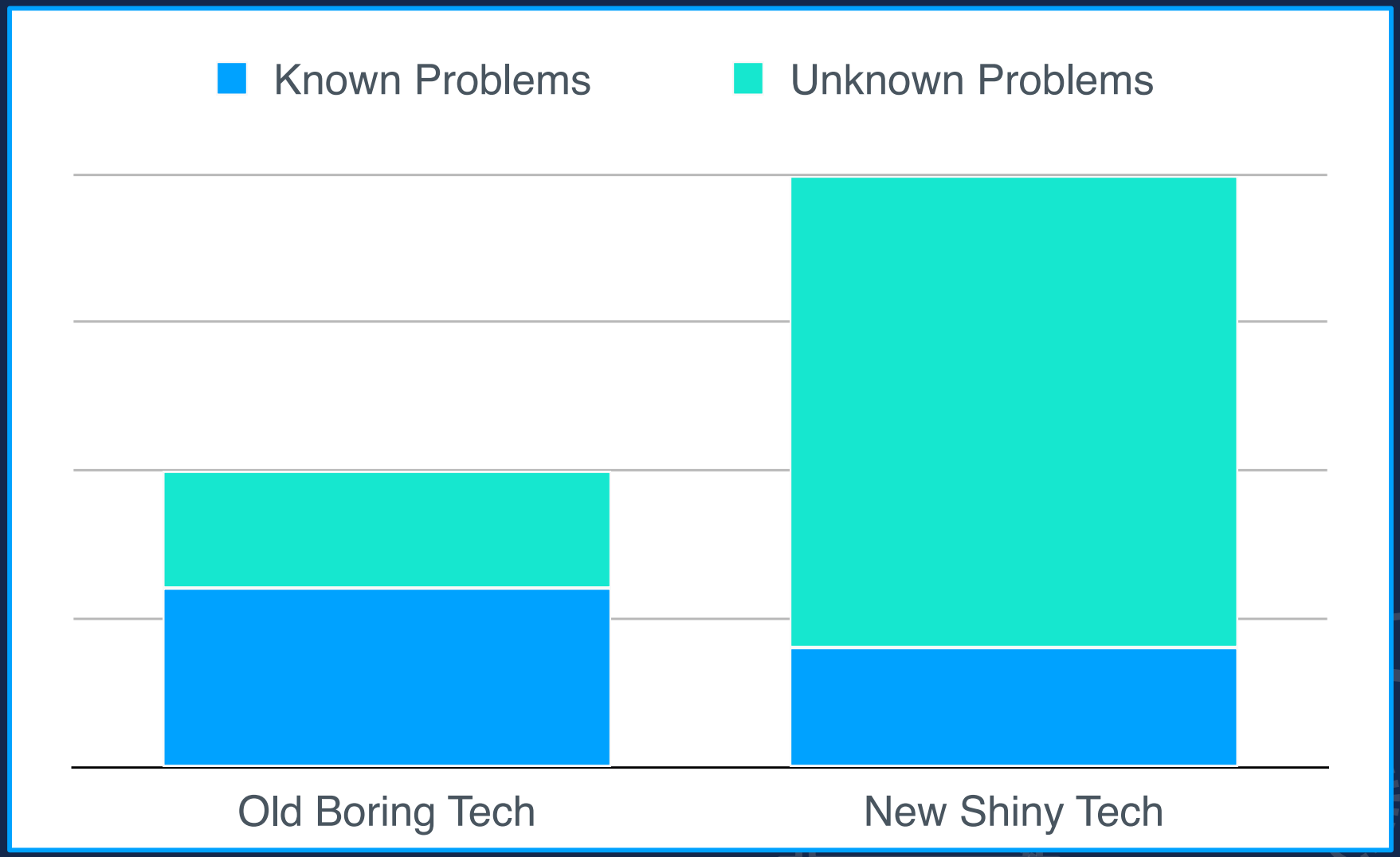
No golden hammers!

Beware the claims of a tool or tech being the best, things are rarely that direct. Trade-offs abound!

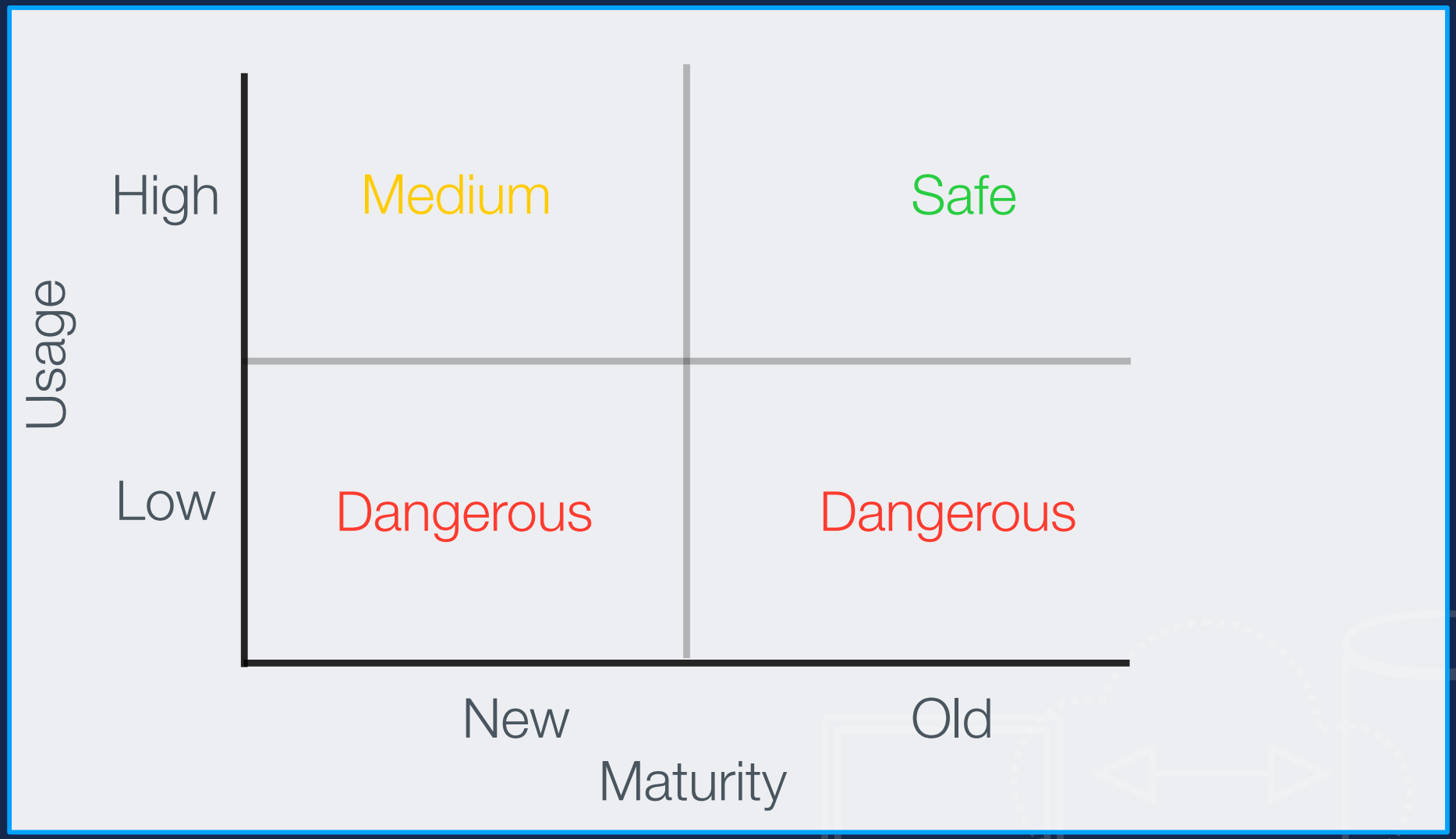
And…our own curiosity



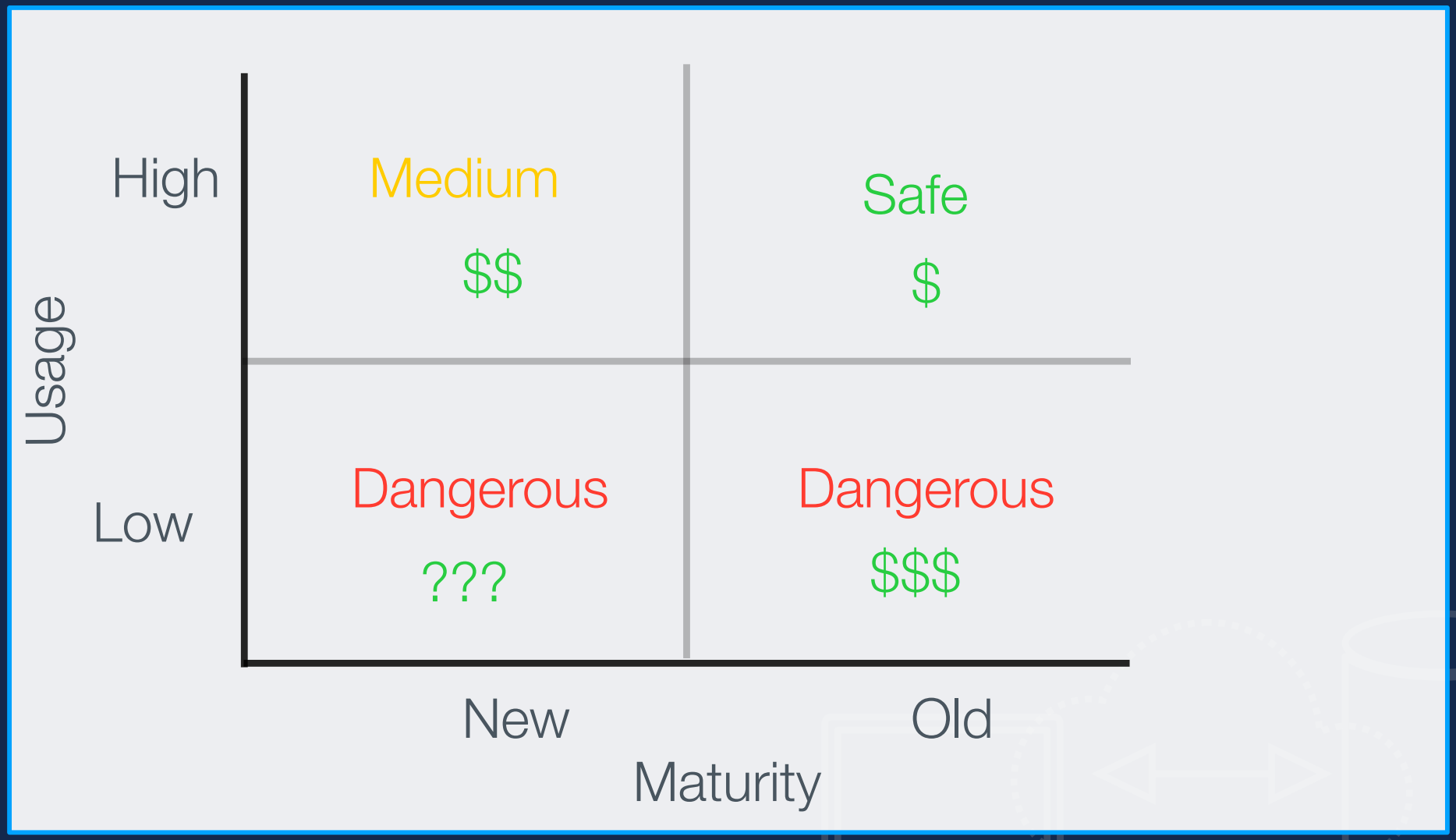
Balanced against actual risk



Maturity Safety Matrix



Yet…


High
Usage
Low
Medium
$$
Safe
$
Dangerous
???
Dangerous
$$$
New
Old
Maturity

Something rarely or ever said by users

“I can’t tell you how glad I am you built the application with <insert cool thing>”

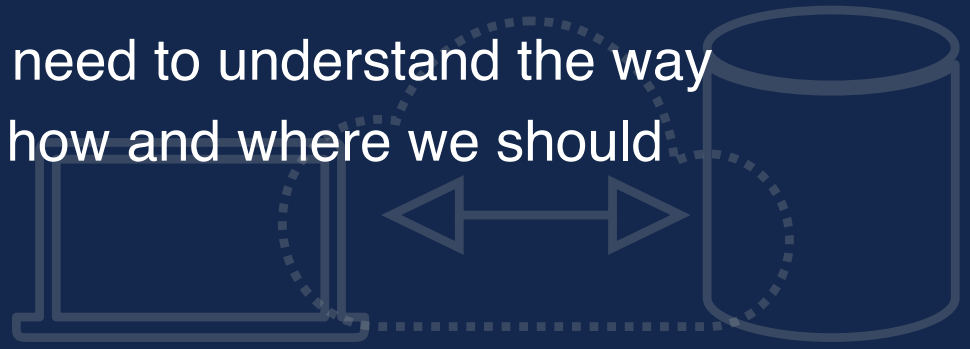

Avoid Scape Goating!
Our project failed because of _____*
* Blame tech, process, least liked person or whatever else can help us avoid accountability

The whole point of this…

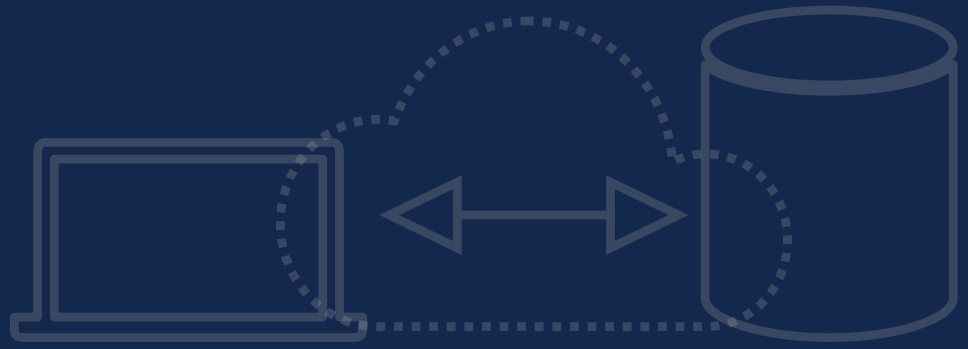
- To try to move ourselves a sense of forming our opinions and making decisions based as much as we can on objective data and a careful weighing of trade-offs.

- I sometimes like to say to move us away from guessing to testing and evaluating.
  - The challenge here is that actually doing things so deliberately may take time and may even invalidate our priors!

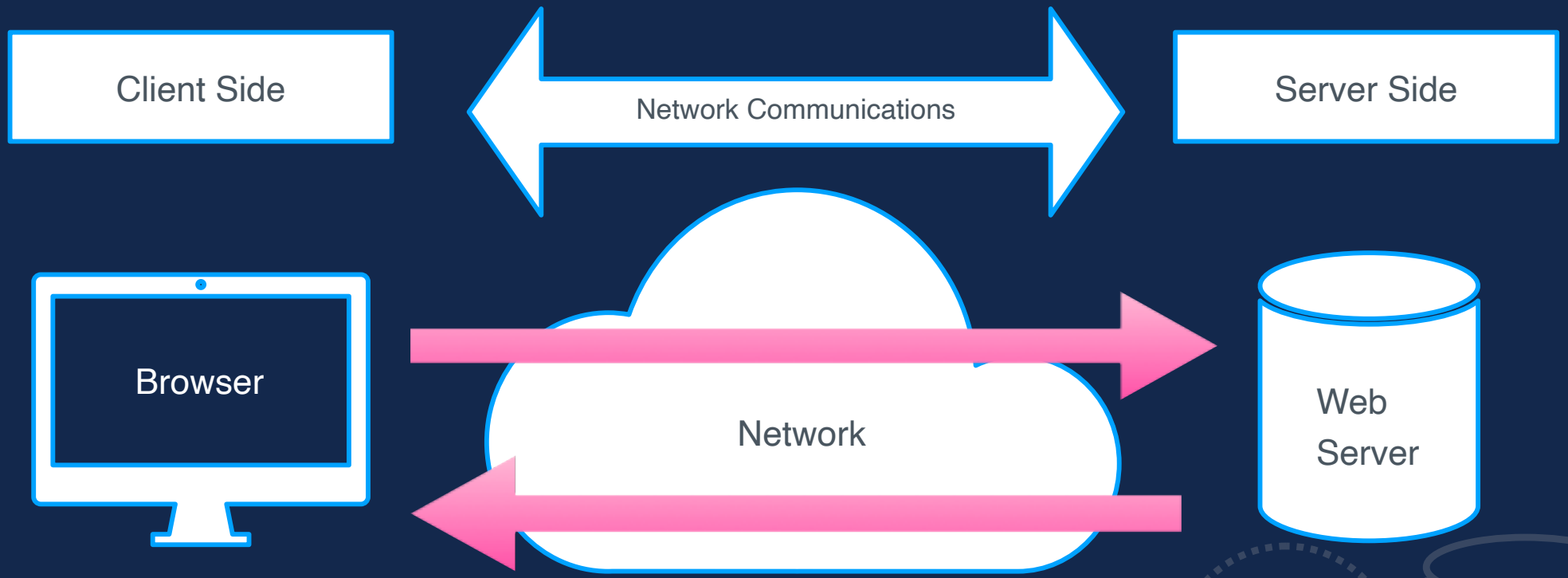
- If we are going to do this properly, we need to understand the way the web works so we can understand how and where we should collect data.



Web Request Overview



Super Simple View

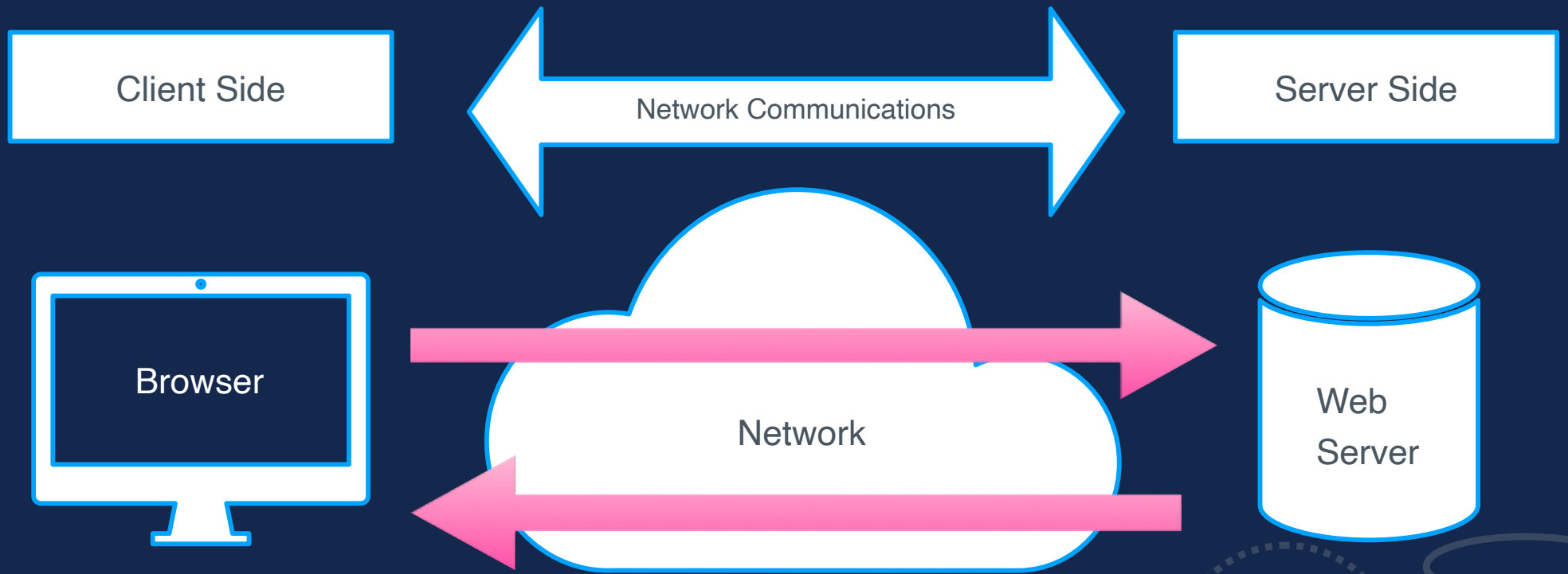


Render and
Interactions here

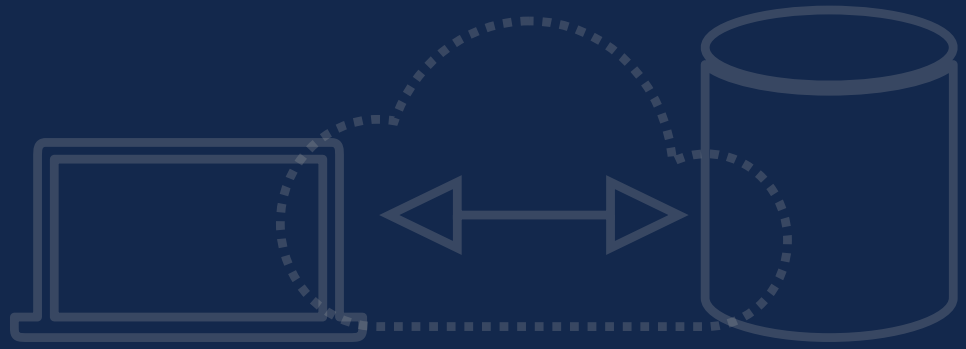
Communicate over this

Storage and
Computations here

Super Simple View - Data Collection



At 10k Meters



A Simple Fetch

- Task: Visit www.example.com
- Step 1: Type in URL or click URL from some link or search query
- 🚨 Stop! 🚨 Do we really know URLs that well?
  - Case sensitive - Y or N?
  - Max length or none?
  - Do we need www's or not?
    - http://www.example.com or http://example.com ?
  - What about trailing slashes are those needed?
    - http://www.example.com/ or just http://www.example.com ?
  - What's with the port numbers
    - http://localhost:8080, http://localhost:3000, etc.

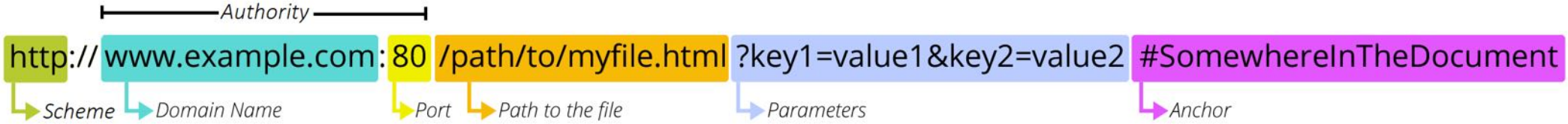
🔎 If something on your client was tracking you could you see all this? Does your browser do that? Could extensions do that?

URL Parts Visually

Not case sensitive

Might be case sensitive due to origin server OS

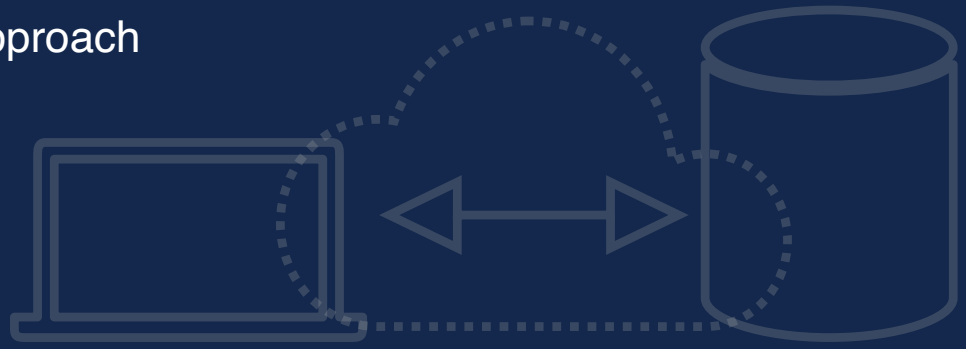
Could be case sensitive depending on tech and rendering mode



Domain will not be but authority such as userid / password could be

tpowell@example.com

Might be case sensitive due to language or code approach

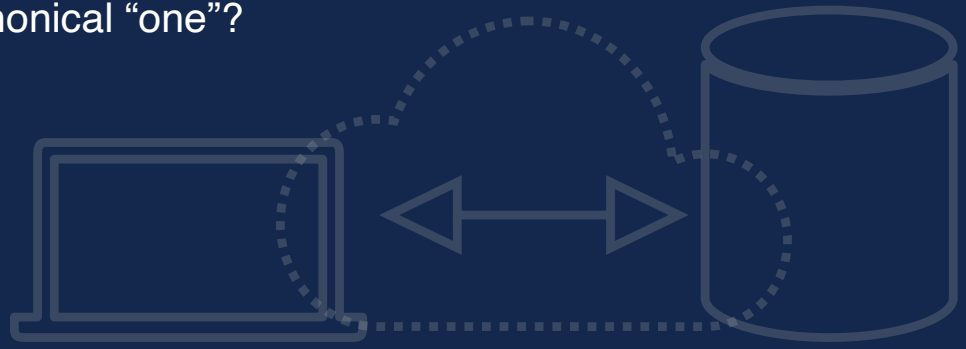


URL Aside

- URL (Uniform Resource Locator)
  https://url.spec.whatwg.org/
  https://datatracker.ietf.org/doc/html/rfc3986

  - Also for history, thinking broadly and for disambiguation
    - URI (Uniform Resource Identifier) -> Stand-in for URL now?
    - URN (Uniform Resource Name) -> Someday I hope!
    - ~~URC (Uniform Resource Characteristics)~~ -> Various <meta> data approaches

- Think abstractly about a “space” of information say “Book Space”
  - What would an instance of a book be?
    - Where found? Multiple copies? Is there a canonical “one”?
  - Is there a unique identifier?
  - What are characteristics of a “book”?



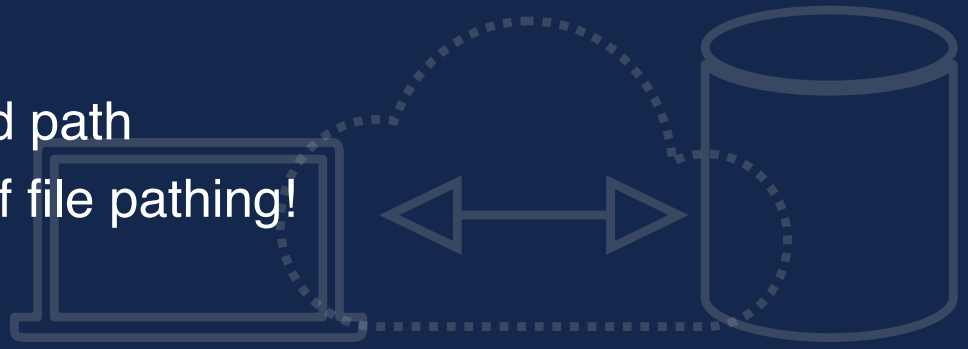
URL Aside - Some Examples

```
http://www.ietf.org/rfc/rfc2396.txt

https://example.com/

https://store.example.com/products?id=5&sale=true

http://localhost:8080/products/5/sale/true

mailto:JaneUser@example.com

wss://socketserver.example.com:443/pusher/

file:///sam/My%20Documents/README.md

tel:+1-858-555-1212

sms:1-408-555-1212

web+myapp:doSomething
```

You could always have made your own - see Things on OSX for example, but soon it will be quite easy - https://web.dev/url-protocol-handler/

Absolute and Relative URLs

- On the web we type in full URLs though the browser may “help” us adding on https:// or with server redirecting to the /
- In development we need to understand more and write full URLs (absolute URLs) an know relative ones properly
- A relative URL assumes certain things from the context of the current absolute URL
  - //example.com/view2 - same protocol
  - /view3/ - same protocol and server
  - page2.html - same protocol, server and path
  - ../../../homepage.html - enjoy the use of file pathing!



“Dirty URLs”, Flat URLs and Semantic URLs

- As a user do you prefer a URL like http://example.com/press/about-urls or http://example.com/press.php?id=5&sessionid=234cascb
- Clearly the former is more semantic and does not expose executional details (more abstract)
- Good URLs are short, understandable, permanent, and avoid revealing implementation details or other “wiring” if possible (aka Dirty) as opposed to clean and simple .
  - Such URLs are flatter http://example.com/about-us vs http://example.com/about/general/about-us typically

Data URLs

data: [*media MIME type*] [; *encoding*], < *data* >

A Simple Fetch

- Step 2: Domain Name Service Resolution
  - Local host file reference
  - DNS Cache
  - Local Name Server Resolution
    - Cache Resolution
  - Root Domain Resolution
  - Top Level Domain Resolution (.com)
  - Domain Server Resolution (example[.com])
  - Local Machine Resolution (www.[example.com])
  - Returning IP address to you for use

🔎 DNS queries might be something somebody could collect for interesting value!

🤔 User: Encrypted DNS at a 3rd party - just shifting the worry though!

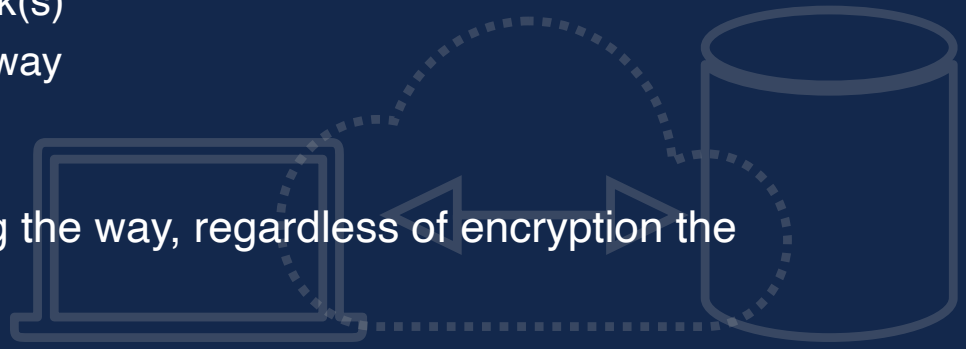
A Simple Fetch Contd.

- Step 3: Make a TCP/IP connection to the destination IP
  - Route request to address
  - Establish communication channel

- Step 4: Invoke an HTTP request to send to server

```
GET / HTTP/1.1
Host: www.example.com
User-Agent: Chrome
Other-Header: Here
/n/n
```

- Step 5: Transmission of HTTP request across network(s)
  - May be intercepted by various proxies along the way



🔎 If not encrypted there might be monitoring along the way, regardless of encryption the end server will know a request was made

🤔 User: Better use an SSL VPN or favor https sites/apps

A Simple Fetch Contd.

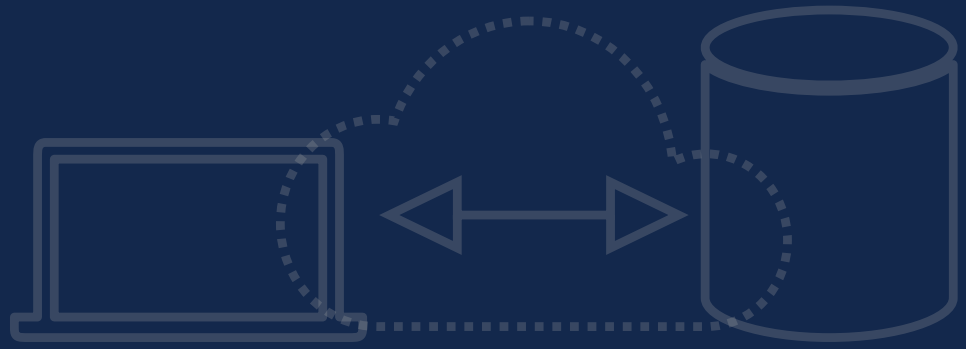
- Step 6: Server receives HTTP request
- Step 7: Attempt to service the request
  - [Authorize]
  - [Authenticate]
  - Access File or Run Program
- Step 8: Return Result or Error

🔎 Logging!

🤔 User: How to avoid being logged?

```
HTTP/1.1 200 OK        [ Alternatively error code like 404, 500, etc ]
Date: Jan 6, 2020
Server: Apache 2.x
Content-Length: 34589
Content-Type: text/html
/n/n
<!doctype html>
<html>
…snip…
```

- Step 9: Response sent back via network

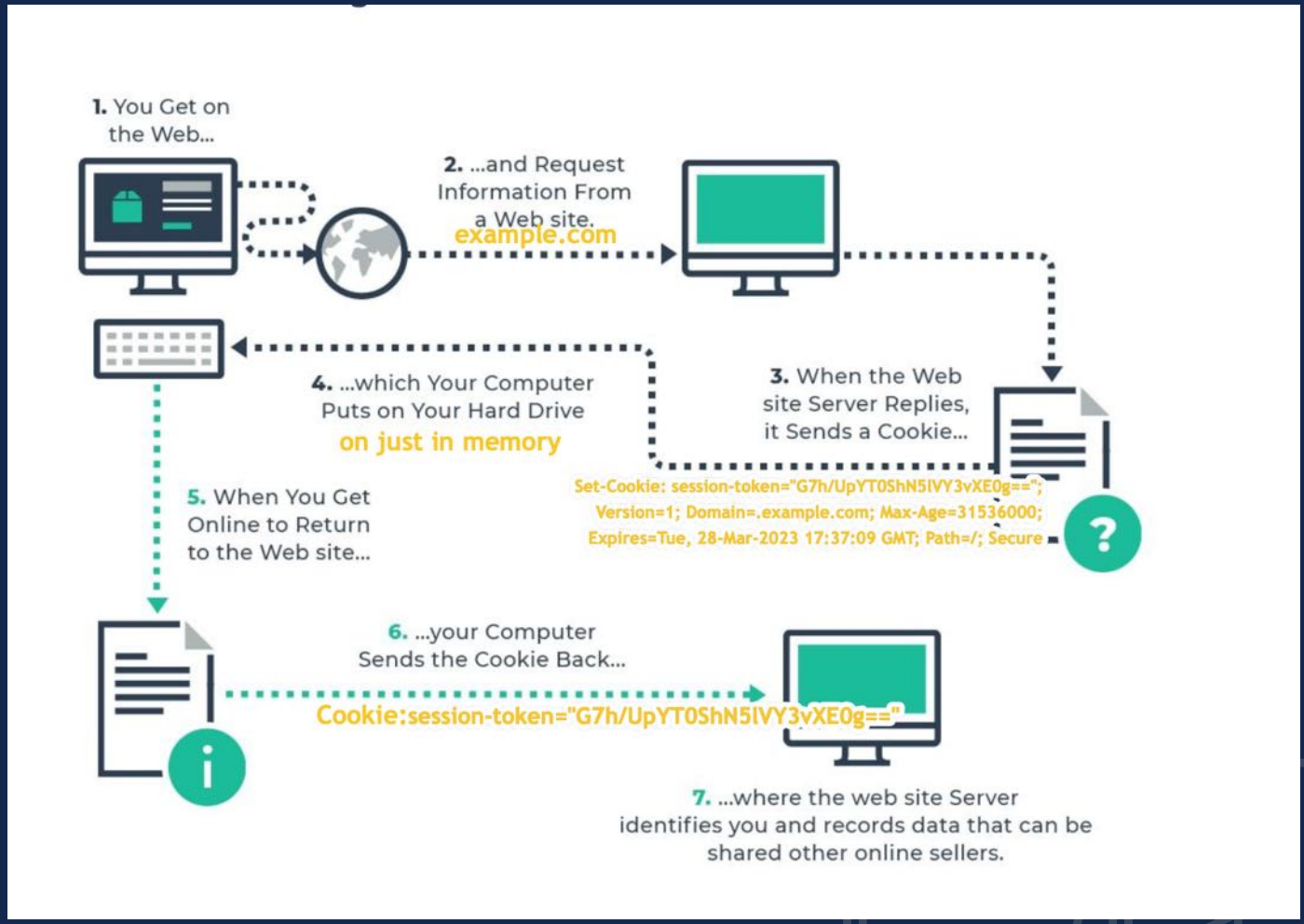


A Simple Fetch Contd.

- Step 10: Browser receives response
- Step 11: Decode MIME type (Content-Type header) and determine action
  - In this case HTML so pass data stream to HTML parsing engine
- Step 12: Parse HTML and form a Document Object Model Tree
  - HTML tree formation may need to address malformed markup and make corrections
  - Items encountered during the parse may cause the process to repeat
  - The HTML document is often the root object of a request, but HTML or not we should distinguish between content which can trigger more requests and that which is terminal (no further requests)

🔎 Cookies might be inbound and outbound on the header exchange here

Cookie Preview



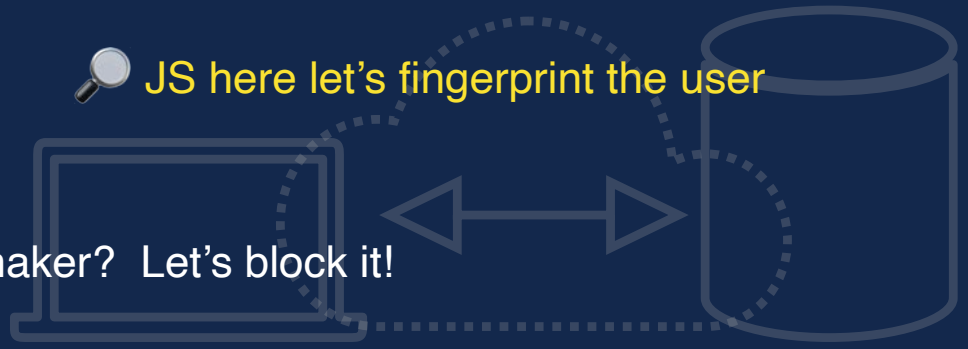
It's all done with HTTP headers!

A Simple Fetch Contd.

- Step 13: Parsed HTML tags may encounter other objects to load
  - CSS - repeat process, parse CSS and apply styles to DOM tree, CSS may contain more things to fetch like fonts or images
  - Images - repeat process and embed images in foreground or background
  - JavaScript - repeat process and run JavaScript code which may
    - Modify the DOM tree
    - Make others network requests

🔎 JS here let's collect some data!!!!

🔎 JS here let's fingerprint the user



🤔 User: So JS is the troublemaker? Let's block it!

A Simple Fetch Contd.

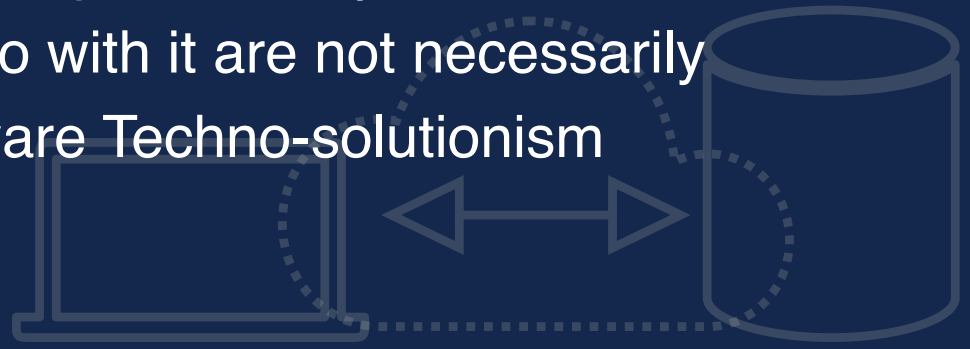
- Other subsequent requests may not necessarily be sent to example.com but could be sent to other 3rd party domains like other.com which would cause DNS lookups and such.
- Remember most sites don't just call one thing, they might call many things and all those things can see a piece (if not much more) of what is going on
  - Possibility - data sharing
  - Possibility - data leakage
- Observation: free things tend to get value from data collection so act appropriately

🔎 3rd Party Requests log part of this - everything from analytics to ads to login systems!

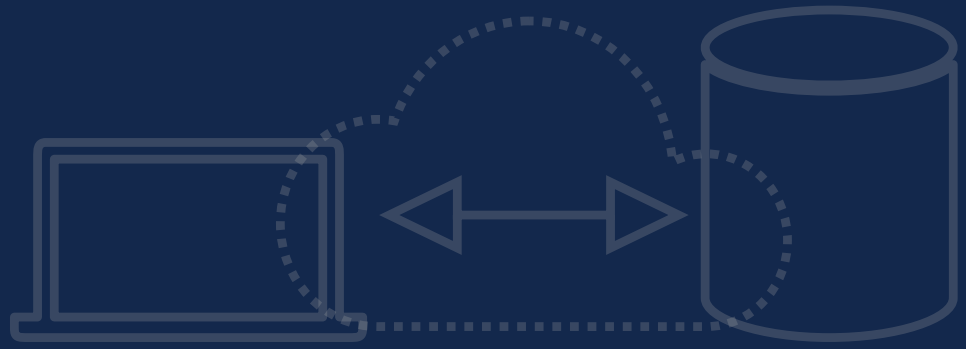
🤔 User: 3rd Party Abuse? Let's block it even at IP level!

A Simple Fetch Wrap-Up

- There are many individual pieces that make up the request - response cycle that powers web sites and apps
- It is important to understand this part of the web because this where trouble is found and power is to be gained
- Data collection can happen all along this path, but try to understand the ability to collect and what people do with it are not necessarily solely technical topics. Warning: Beware Techno-solutionism



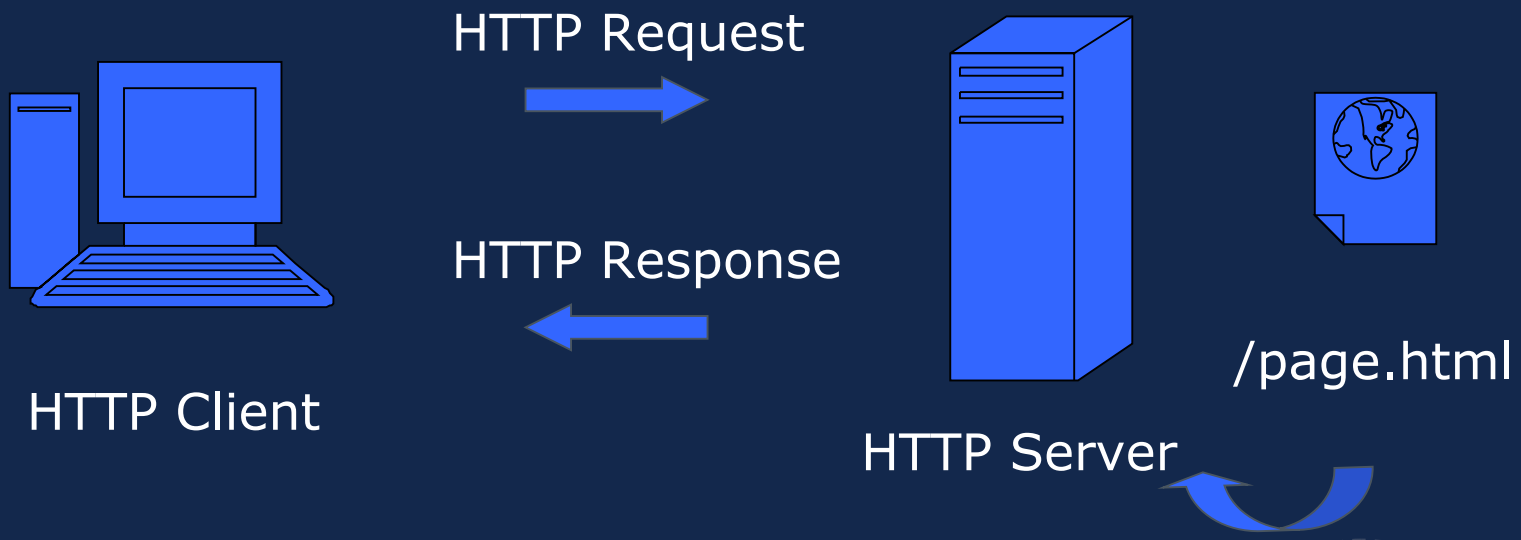
At 1k Meters



Request Response

Asks for resource by its URL:

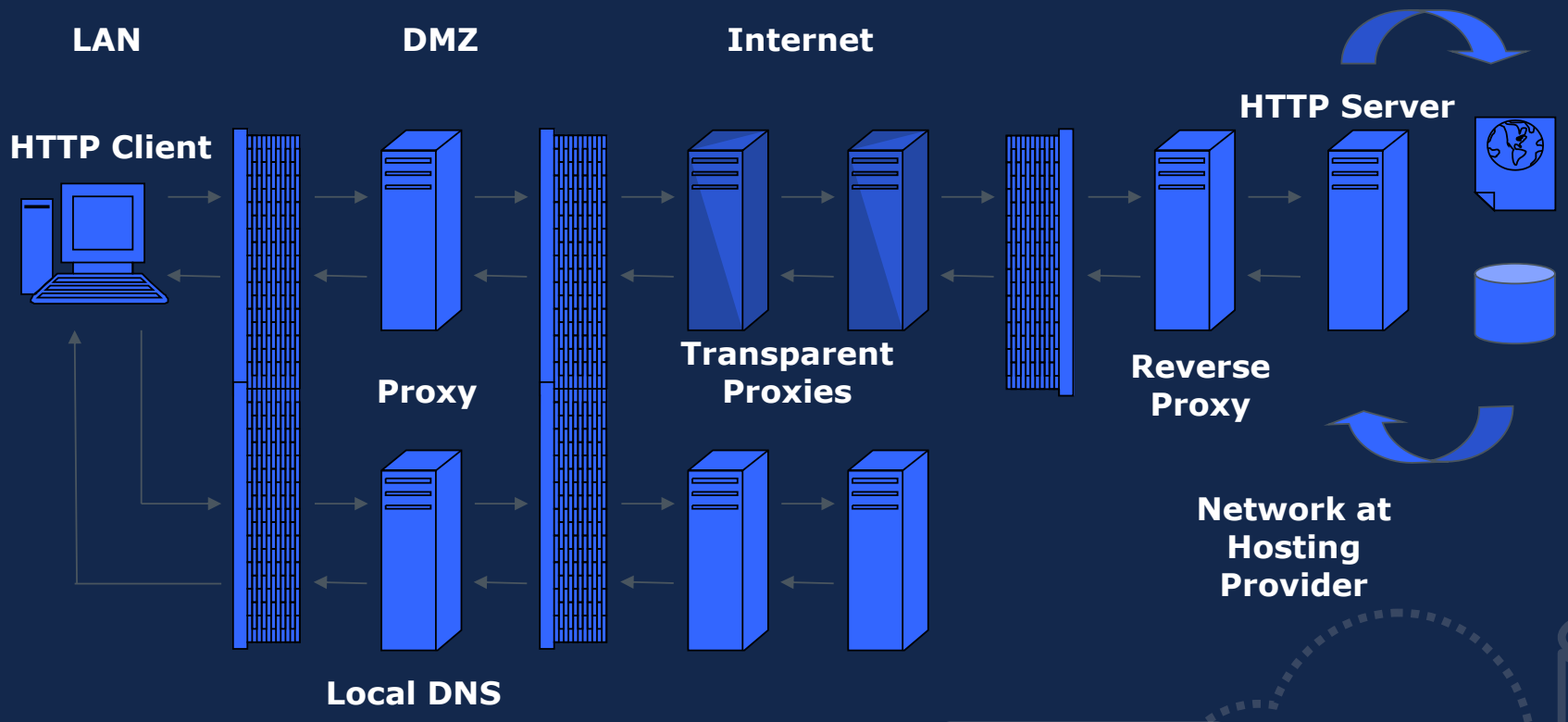
http://www.foo.com/page.html



Browser decodes MIME type and determines action

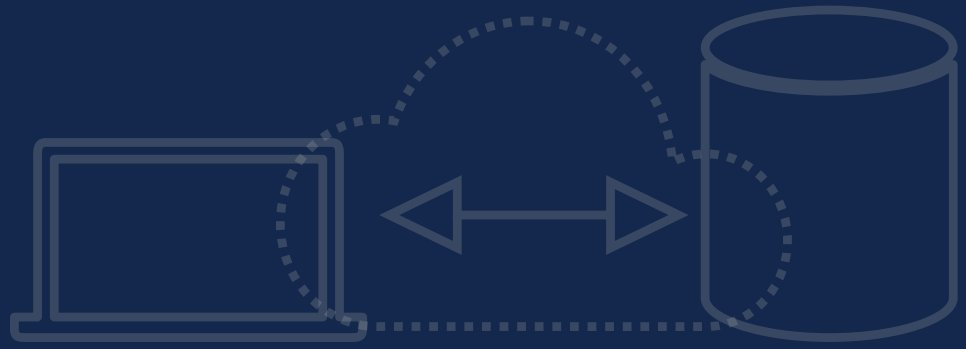
maps file extension .html to appropriate MIME type: text/html

Gross Oversimplification



And Browser Actions!

At 10 Meters



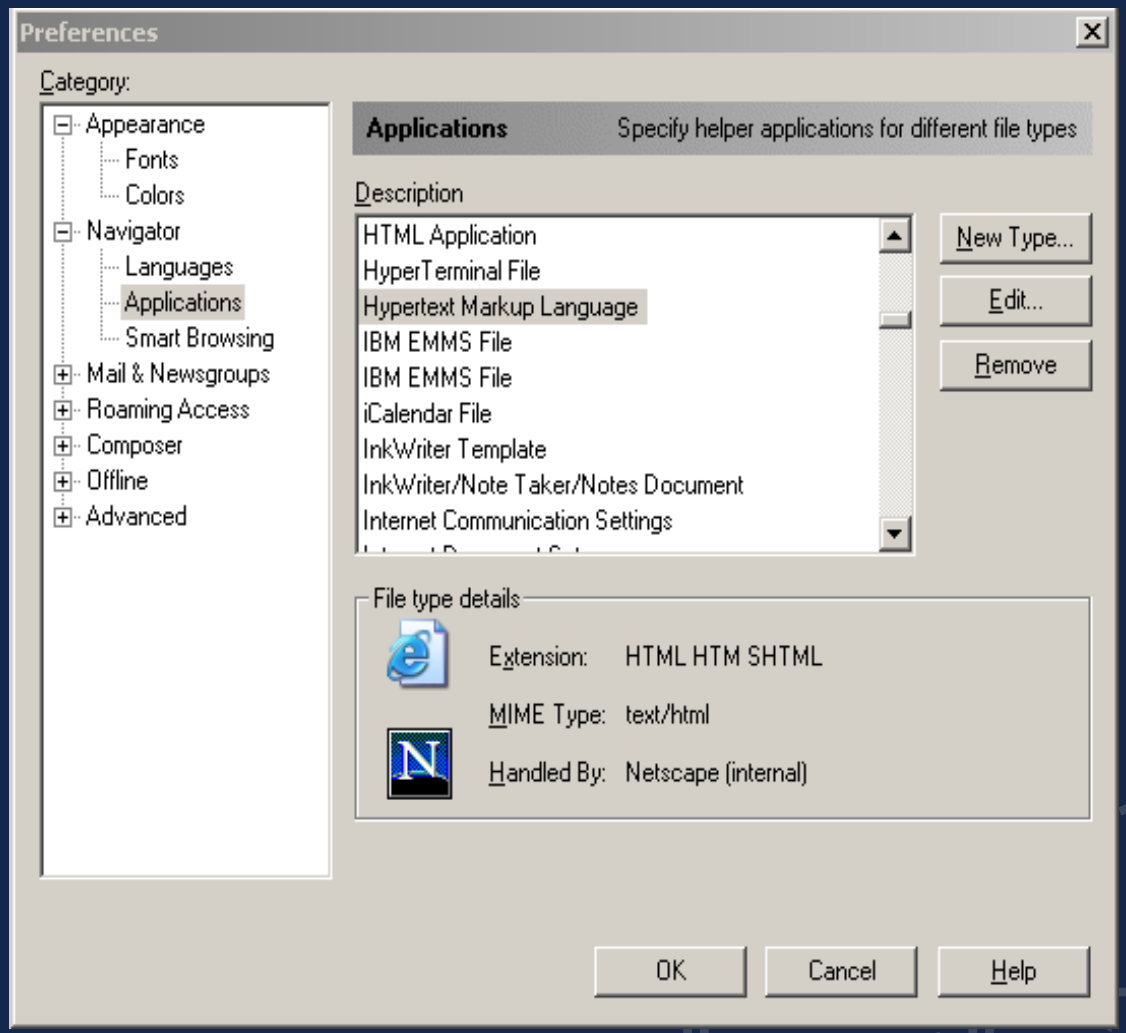
Simple HTTP Request

```
GET /index.html HTTP/1.1
Host: example.com
User-Agent: Simple Demo Browser
/n/n
```

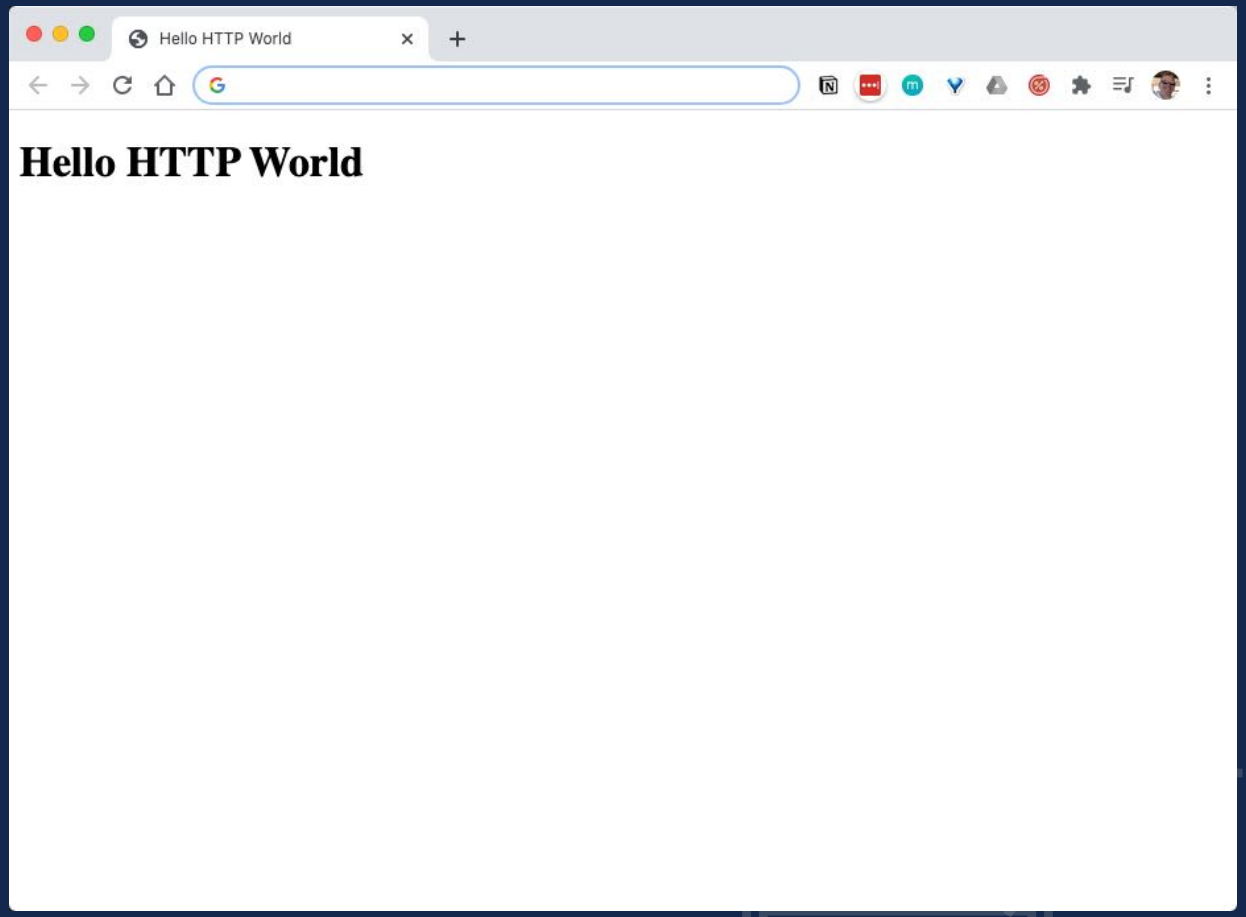
Simple HTTP Response

```
HTTP/1.1 200 OK
Date: Sun, 02 Feb 2020 22:33:22 GMT
Content-Type: text/html
/n/n

<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8" />
<title>Hello HTTP World</title>
</head>
<body>
<h1>Hello HTTP Example</h1>
</body>
</html>
```

Browser Action



Web Page Rendered



Profit!

HTTP Example #2

Request

```
GET /image/jpeg HTTP/1.1
Host: httpbin.org
User-Agent: Chrome/79.0.3945.130
```

Response

```
HTTP/1.1 200 OK
Date: Sun, 02 Feb 2020 22:33:22 GMT
Content-Type: image/jpeg
Content-Length: 35588

ÿØÿà JFIF  ÿþ5Edited by Paul Sherman for WPClipart,
 Public DomainÿÛC

ÿÛC
ÿÄ  ÿÄ
ÿÄM  !  1#A"$Q  34a 2Dq BCST ¡ %cdr±áð5RbsÁ
Ñ&Ut  çñÿÄ  
```

HTTP Example #3

Request

```
GET /image/jpeg HTTP/1.1
Host: httpbin.org
User-Agent: Chrome/79.0.3945.130
```

Response

```
HTTP/1.1 404 NOT FOUND
Date: Mon, 03 Feb 2020 02:15:46 GMT
Content-Type: text/html
Content-Length: 144
Connection: close

<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8" />
<title>Page Not Found</title>
</head>
<body>
<h1>Resource Not Found</h1>
</body>
</html>
```

HTTP Request Complexity Example

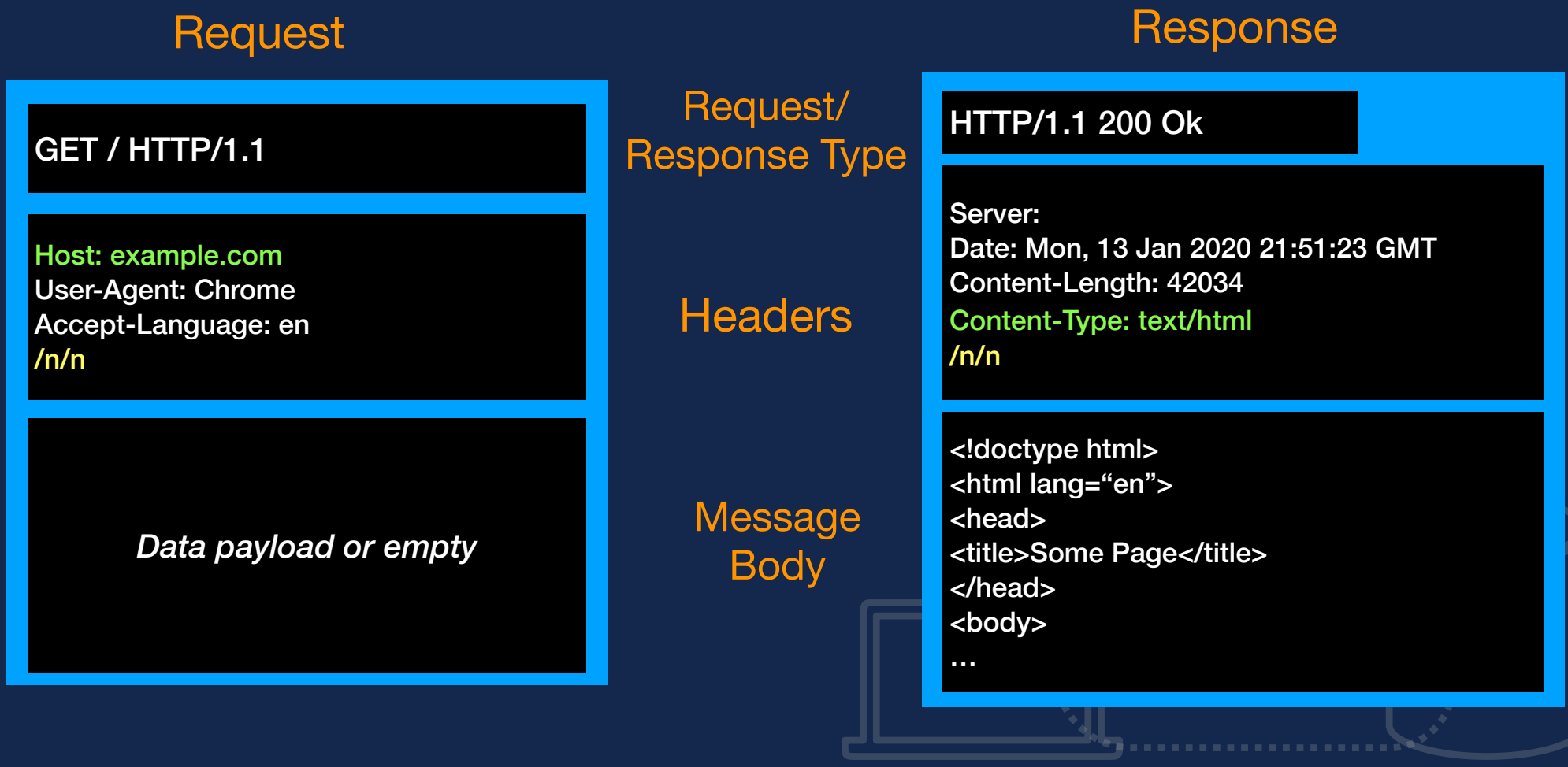
```
GET /_resources/img/tab_fun.jpg HTTP/1.1
Accept-Language: en-US, en; q=0.9
Cache-Control: no-cache
Accept-Encoding: gzip, deflate, br
Accept: text/html, application/xhtml+xml,application/xml;q=0.9, image/webp, image/apng, *.*, q=0.8
Cookie: SessionID=23423452523452352525
DNT: 1
Pragma: no-cache
X-Requested-With: Some Coding Environment
X-Powered-By: thomas
Host: ucsd.edu
Connection: close
User-Agent: Paw/3.1.10 (Macintosh; OS X/10.15.5) GCDHTTPRequest
```

HTTP Response Complexity Example

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 27703
Cache-Control: max-age=3600
Content-Encoding: gzip
Content-Type: text/html; charset=utf-8
Etag: de687d4b95ee9ed2c1e15dee0ef5e2de8df5f7eac8b60c54c4b15e6b288d54b5
Last-Modified: Sat, 01 Feb 2020 01:10:13 GMT
Strict-Transport-Security: max-age=31556926
Accept-Ranges: bytes
Date: Sun, 02 Feb 2020 22:18:55 GMT
X-Served-By: cache-bur17562-BUR
X-Cache: MISS
X-Cache-Hits: 0
X-Timer: S1580681935.453703,VS0,VE166
Vary: x-fh-requested-host, accept-encoding
```

The HTTP Law of Three

HTTP packets have three pieces and all that matters are those three pieces

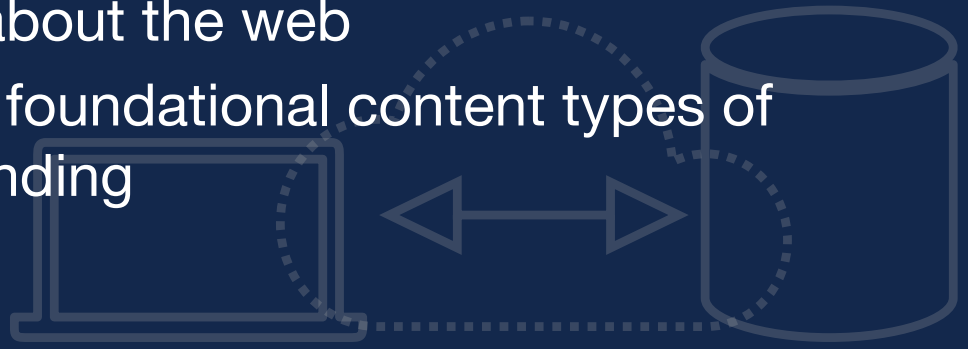


Take-aways for Now

1. HTTP is simple and stateless
   - Text protocol, request/response

2. Yet it is complicated in execution
   - 10s or even 100+ requests
   - Lots of details - headers, methods, networking, etc.
   - Interactions and relationship between data types

3. From all of it rises everything else about the web
   - Understanding of HTTP and the foundational content types of the web leads to deep understanding



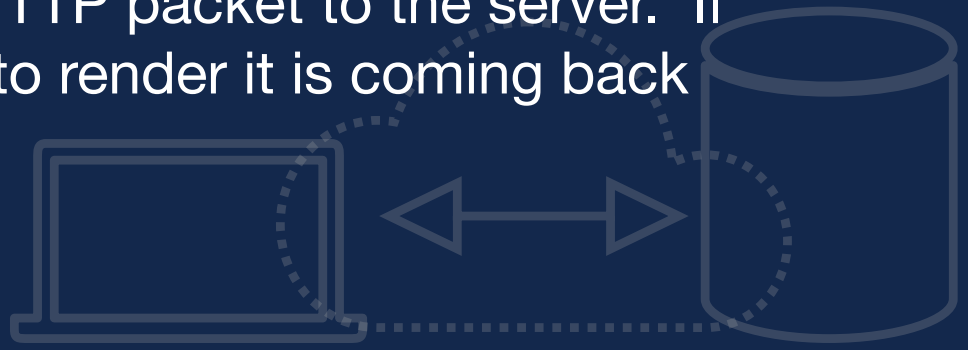
More Take-aways

1. All that comes in on the Web is 3 things

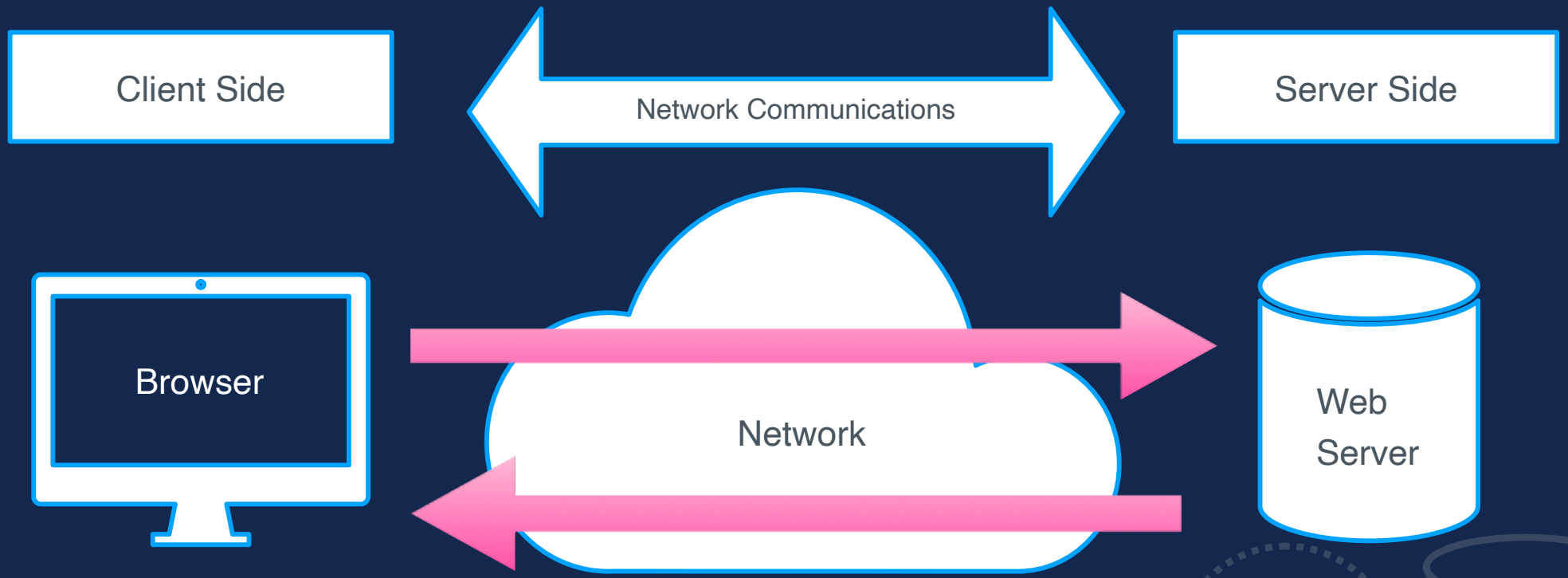
   "3 things in, 3 things out"

2. Master what is coming in and out as that is all there really is to know.

3. Practically Rephrased: We transmit data with this so if we collect a form worth of data it is sent in an HTTP packet to the server. If we get a response with some JSON to render it is coming back this way too!



Client Server Dimensions Visualized



Goals of code?
How much code?
How code made?

Nature of network
Reliability Performance
Security

Goals of code?
How much code?
How code made?

The Client Side

- Usually a browser, but could also be…
  - An app (just using HTTP + data)
  - A browser in an app (with full HTML, CSS, …) aka “web view”
- Client side is not under developer control
- Client side may be hostile
- Client side may be limited in ability
- Primary duty is presentation and user interaction and data collection
  - Speed and look and feel focus!
  - Faster than server-side from a response point of view because no network requirements!

The Server Side

- The server side, sometimes dubbed the cloud, representing computing power remote from the end user
- Server side is under developer control
  - Choices and scale ability more under our control
- Security can be addressed
- Server side is constrained by the network from the user's perspective
- Server side focuses on storage and computation
  - Emphasis on security and scale - it's where the data is! There might be some worry if we should transmit data or keep it where it is ?

The Client Server Pendulum

- Server Focused aka “Thin Client”
- Client Focused aka “Fat Client”

- Server focused ~ “centralized”
- Client focused ~ “decentralized”

- Server focused ~ “backend”
- Client focused ~ “frontend”

Trade-offs and pendulum metaphor

Client & Server-Side Applied

Server-Side Pros

Pro: Works well with big compute and large data sets, securable

Con: Interactivity challenged

Client-Side Pros

Pro: Rich and interactive

Con: Data set sizes and compute may be constrained, not easily securable

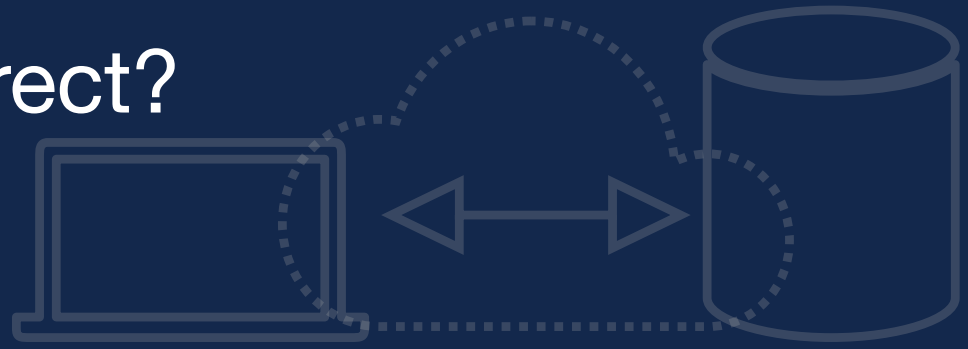
*Obviously a hybrid seems most appropriate depending on need of of application

Common Assumptions

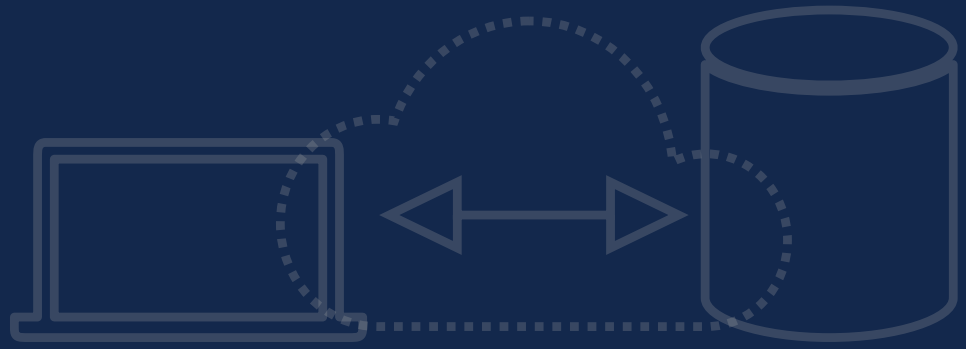
Given the discussion so far, in practice we tend to see the following:

- Server-side solutions for older things
- Client-side solutions for newer things

Is this correct?



Applying Models to Analytics



Analytics 10K

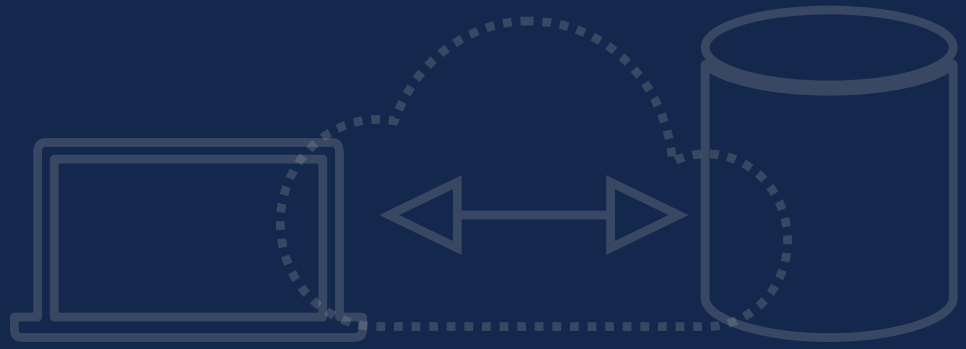
Where we can collect analytical information include:

1. Server Side
    - Logfiles
    - Pro: Can’t be opt-ed out, private on your system
    - Con: Limited in what they can record, hard to work wtih
2. Client Side
    - Google Analytics
    - Pro: Lots of tracking possibility (rich data), Easy to use
    - Con: Can be blocked, some data quality problems, abuse
3. Network
    - Network Capture
    - Pro: Can’t be opted out of, capture much in one place
    - Con: Like log files limited in what it can see, likely $$$

Analytics Intro Points

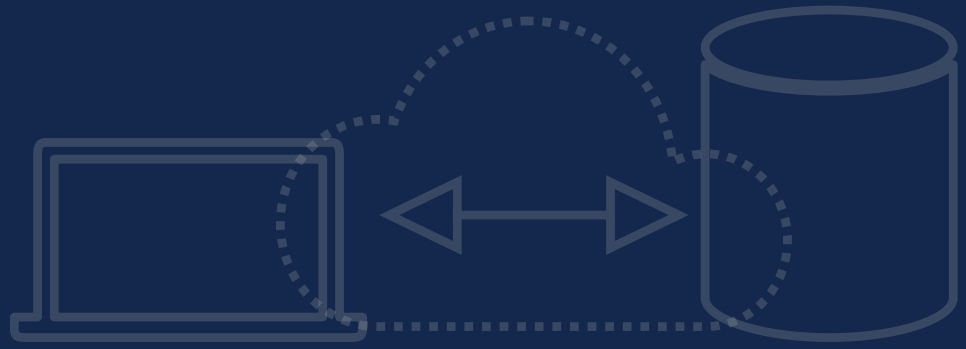
- You can collect an immense amount of data once you understand the medium
- What you do with the data isn't inherently bad, but it is pretty easy to get yourself to a "bad" place if you aren't thoughtful
  - Not everyone will understand what you are doing, they might not consent if they understood
  - Just because there isn't a law that keeps you from doing something doesn't mean you should just do
- Even if you have lots of data it doesn't mean it will provide a full picture of things
  - Be careful of the fallacy of "TIA" (Total Information Awareness)

What We Might Measure



What to measure

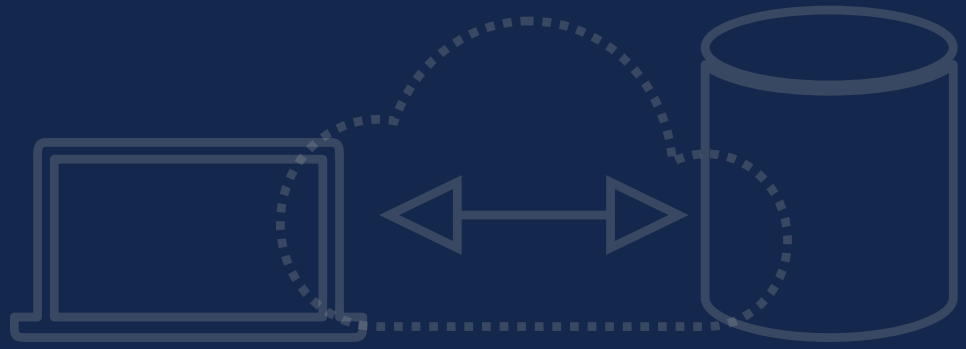
- Obviously there are a lot of things we COULD measure
- In this course we are going to focus on
  - Technographics - device types, screen dimensions, color, network connectivity, etc.
  - Performance - page load time and interaction times
  - Error rates - both code and user errors
  - User activities - clicking, scrolling, etc,
  - Basic User Identification



Why we measure these things

- Technographics - Stop guessing about what “everyone” has or what their experience is and actually build our requirements to that
- Errors - fix code errors and redesign approach to reduce user errors
- Activities - see if the app/site encourages desired user behavior
- Identification - so we can group actions and correlate

- **Performance** - because other than availability and errors it correlates massively to success



Measurement Motivation: RAIL




Response | **Animation** | **Idle** | **Load**

https://developers.google.com/web/fundamentals/performance/rail

RAIL TL;DR

- Response to actions/input has to be <100ms

  User tap, click, etc. to reaction has to be <100ms

- Animation 60fps, each frame ~10ms, otherwise.. JANK!

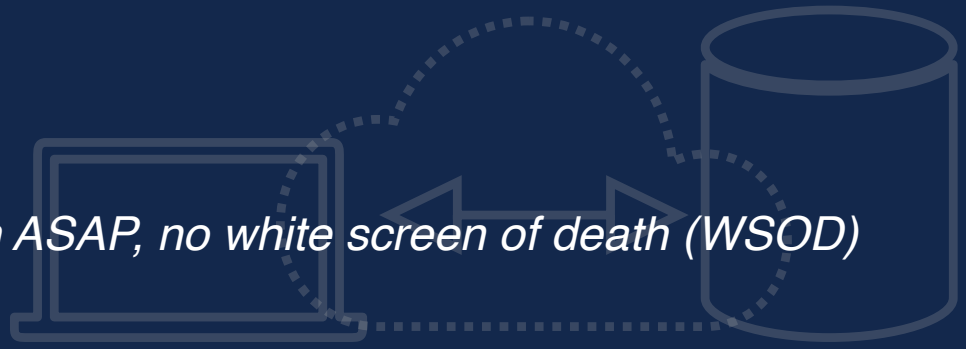
  Actually 1000/60 - 16.66ms per frame but shoot for ~10ms otherwise the render effort may not complete in time.

- Idle work loads 0-50ms to avoid JANK!

  Given you share the UI thread with your app, you can't do something longer than 50ms without setting yourself up for JANK later.

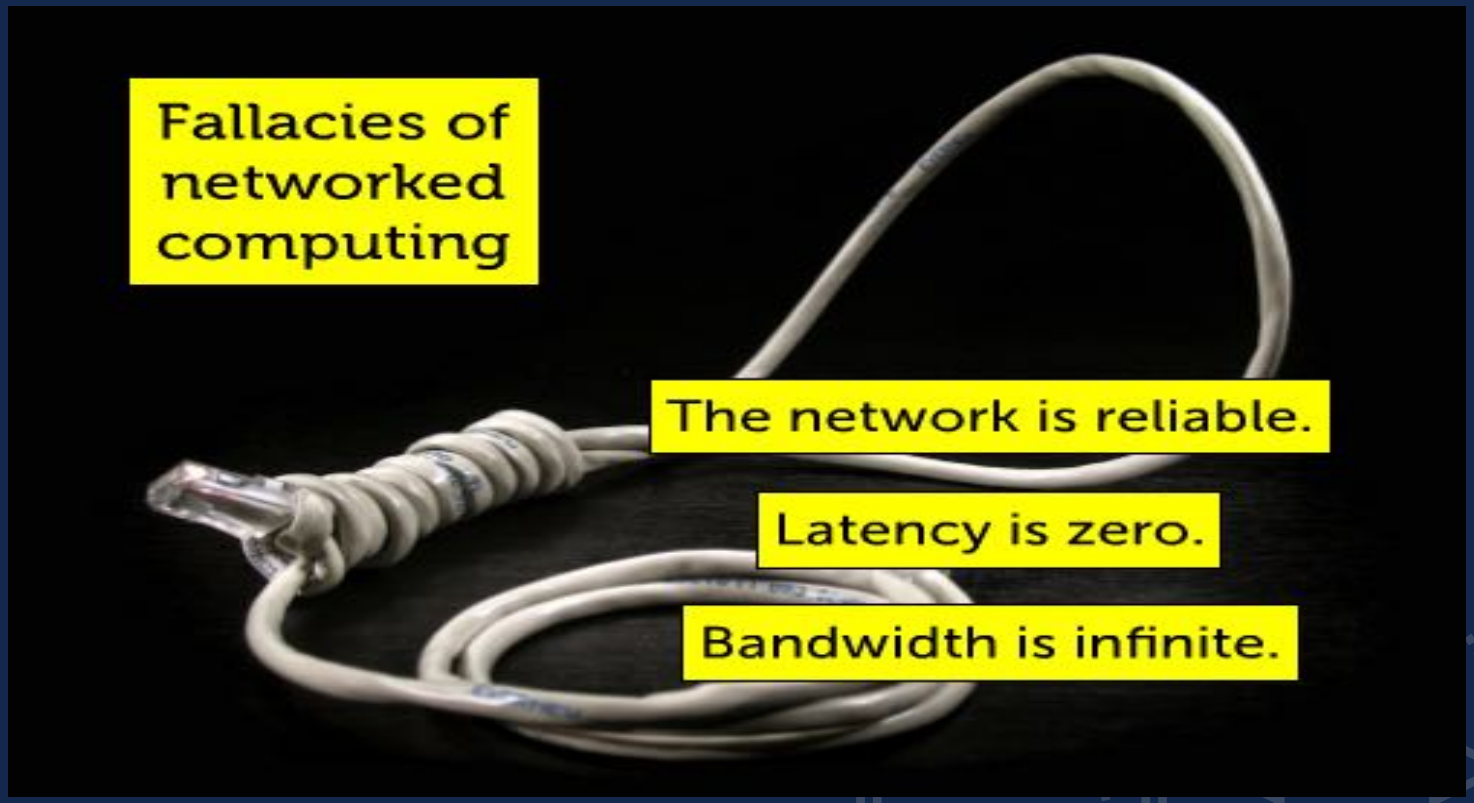
- Load < 1000ms

  Get items (especially above fold) on screen ASAP, no white screen of death (WSOD)



Bad News!

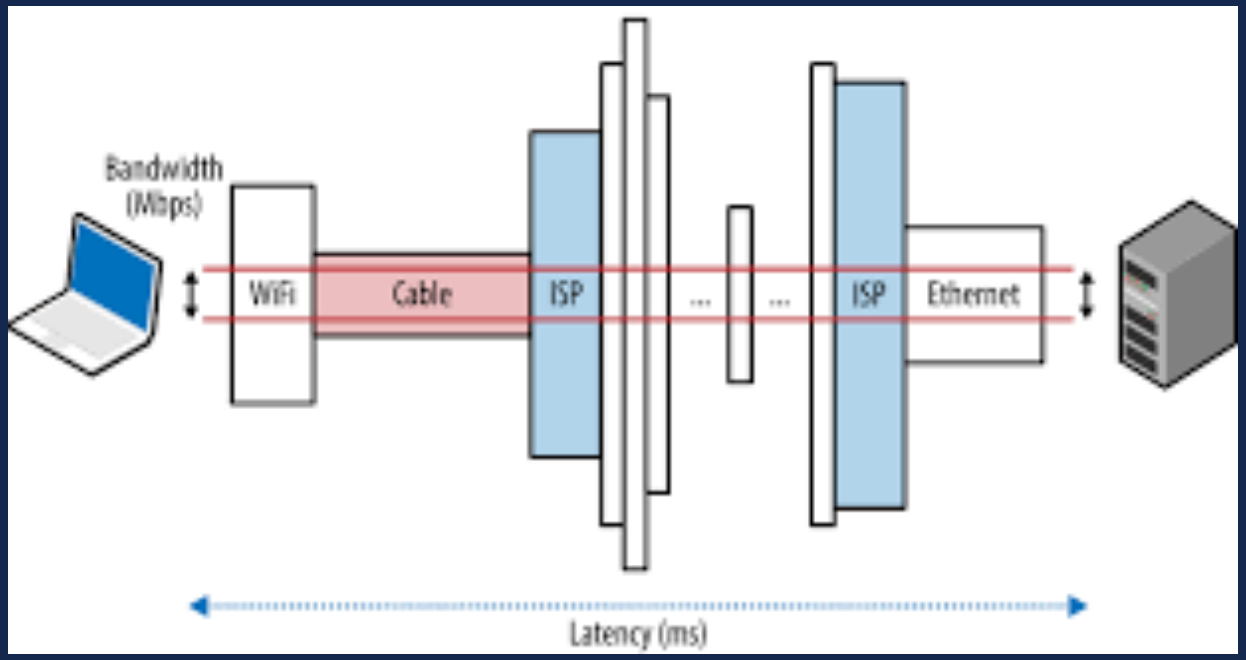
We do our work over a network and it is TROUBLE!



Don’t Worry xG is coming soon!

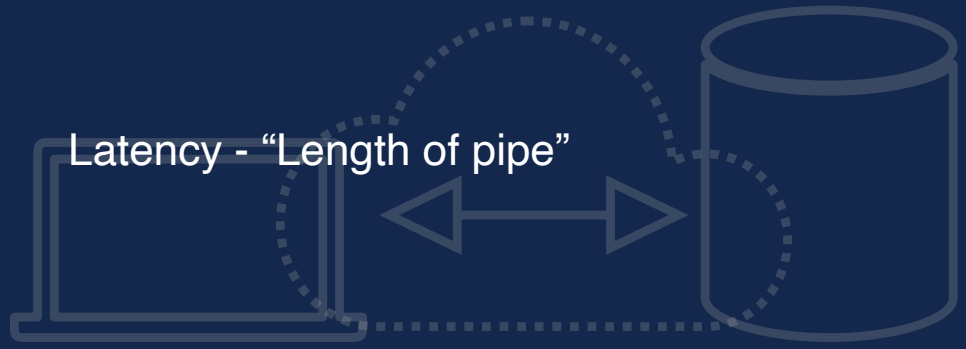
I often call this the bandwidth fallacy

Bandwidth & Latency

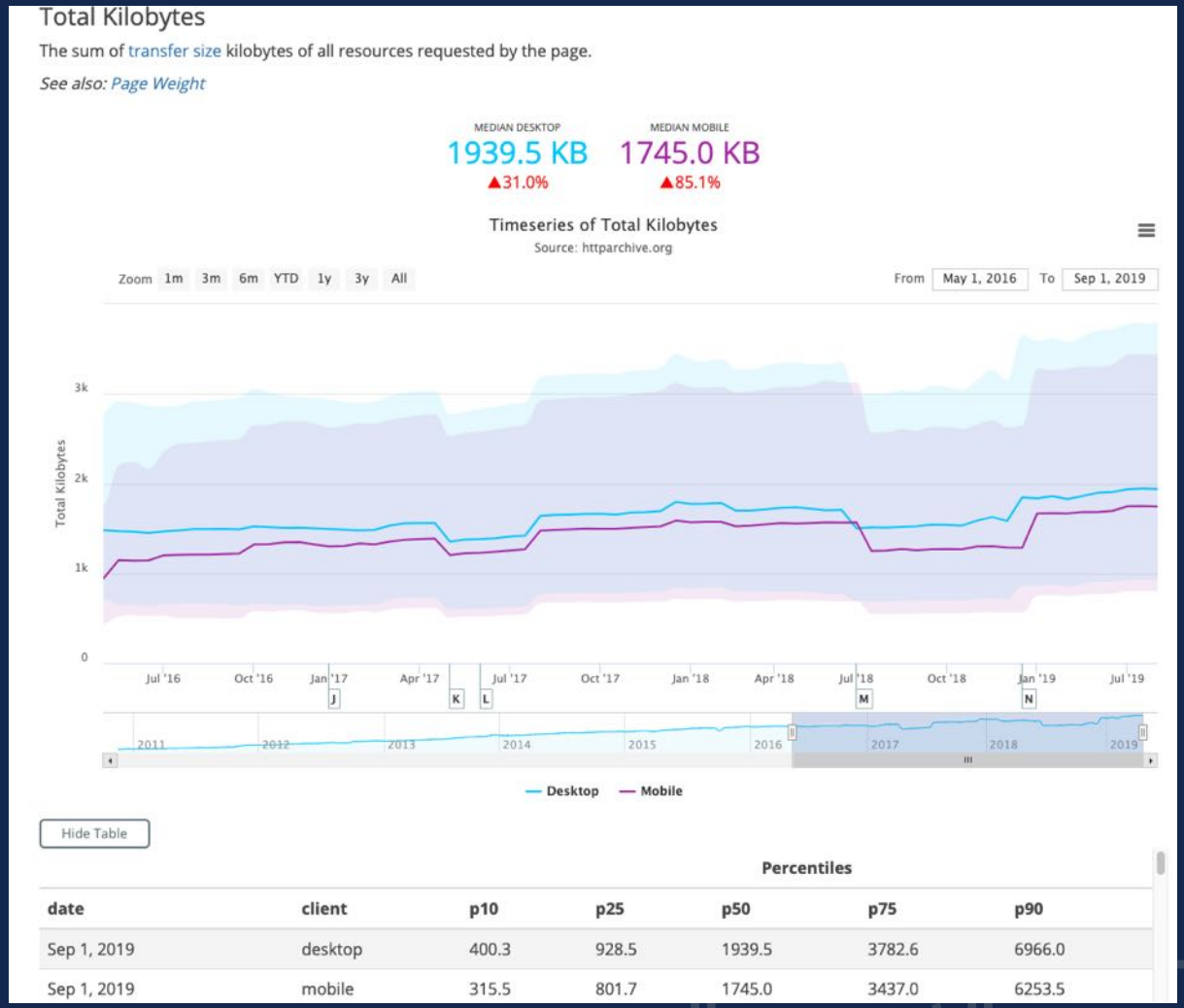


Bandwidth - “Pipe size”

Latency - “Length of pipe”



Up and to the right = BAD!?



Percentiles

date	client	p10	p25	p50	p75	p90
Sep 1, 2019	desktop	400.3	928.5	1939.5	3782.6	6966.0
Sep 1, 2019	mobile	315.5	801.7	1745.0	3437.0	6253.5

https://httparchive.org/reports/state-of-the-web#bytesTotal

Reality Check

- 1s response time is going to be very tough over the public Internet

  Can you figure out how many byte you get? How much JavaScript and images can you send then?

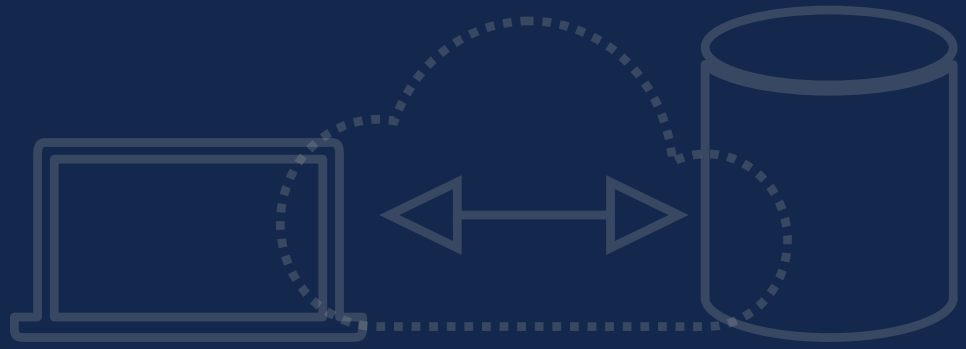
- 100ms is equally tough on low CPU powered devices

  Stop with the everyone has or will get iPhone 13s and Galaxy s22s!

- Unless we change laws of physics or replicate everywhere network delivery will continue to remain tough…

And busy humans don't have time for our FAIL

Back to Models



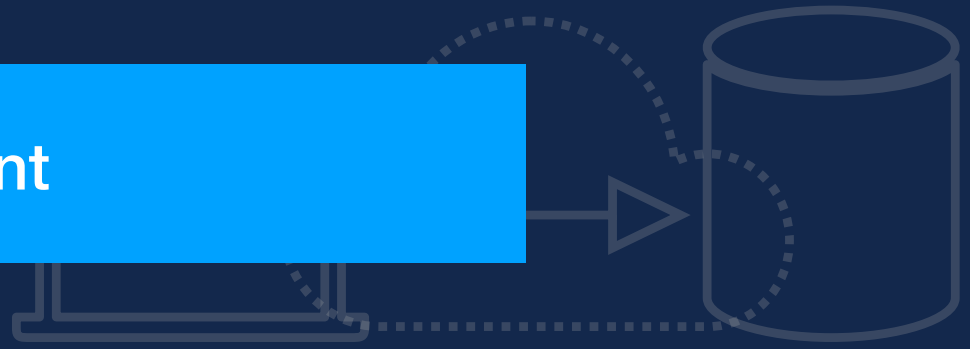
Abstract Stack

Interactivity

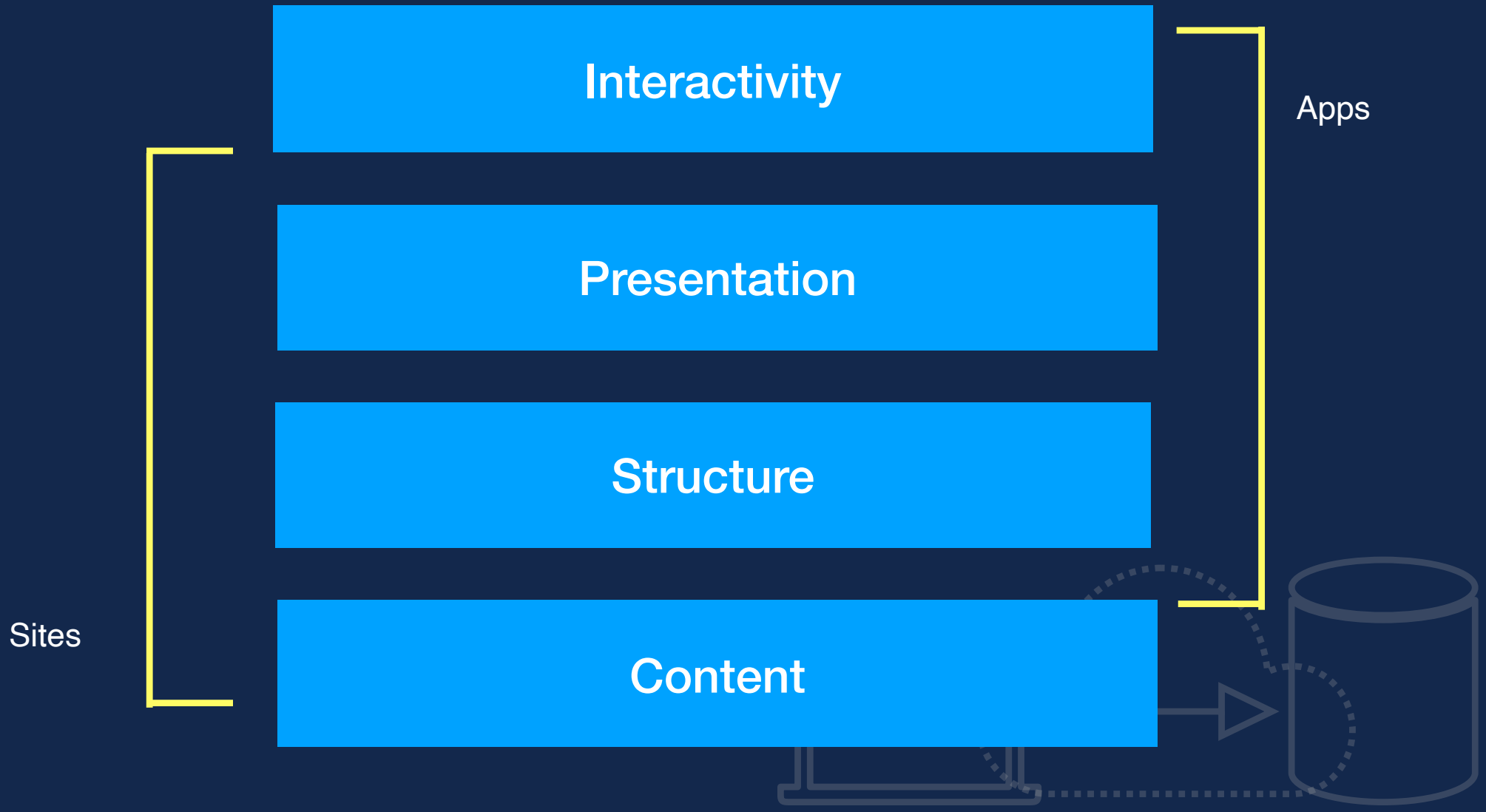
Presentation

Structure

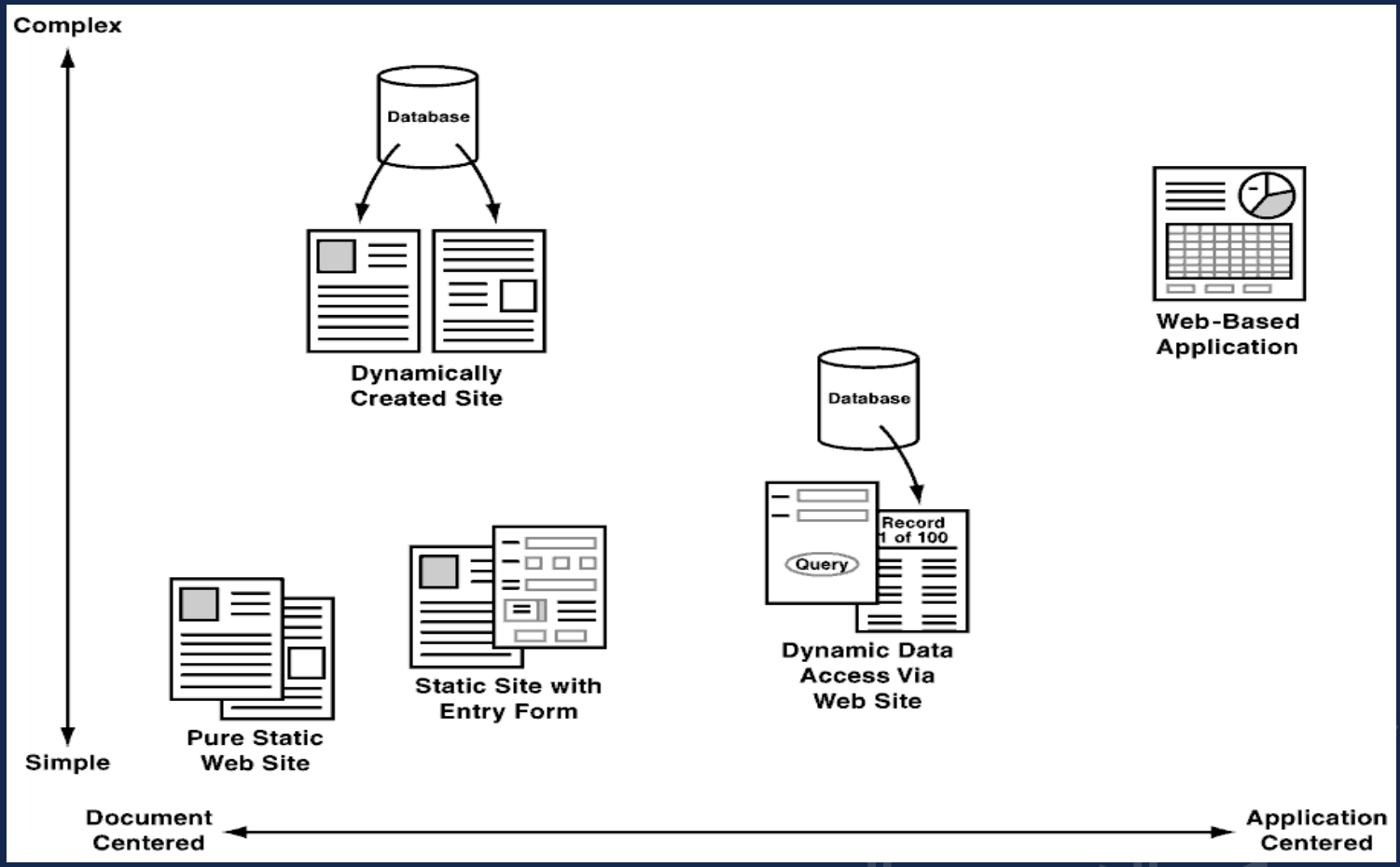
Content



Sites and Apps



Sites and Apps



Discussion: Is there are trend in this continue and if so is it right?

Sites and Apps

- If something is mostly non-interactive content, it's probably a site

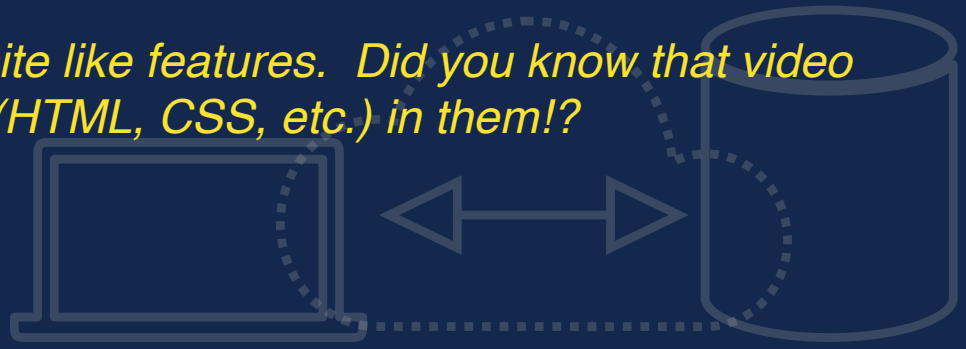
  Don't use programming to manage it unless at scale, you are likely overcomplicating it and may introduce adverse side-effects (e.g. Performance, SEO, a11y, …)

- If something is task or interactive based it's likely an app

  Bending traditional web tech for apps can be problematic, though it is tried and true so it may be more expedient than something newer

- The degree of site-ness and app-ness is not absolute

  Apps can be mostly app based but have site like features. Did you know that video games often have web tech (HTML, CSS, etc.) in them!?

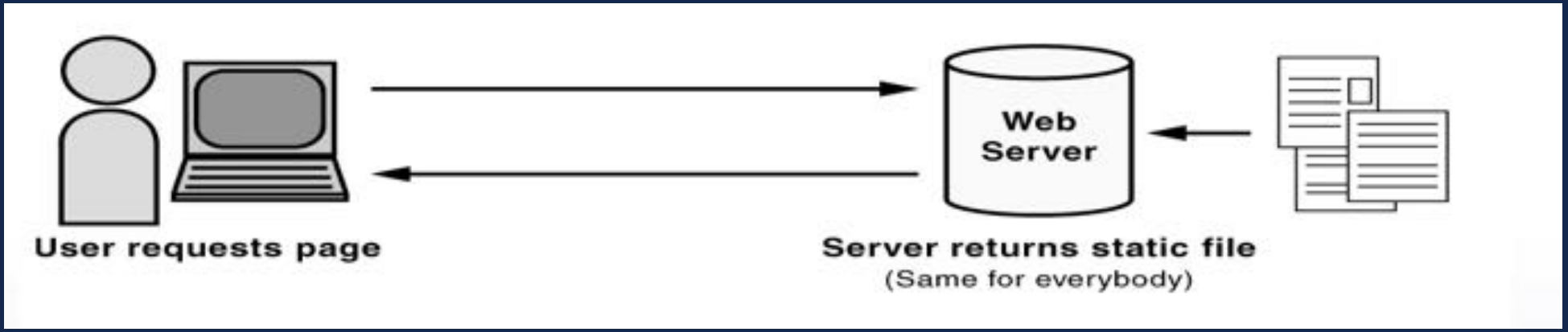


Web Site Types and Characteristics

Type of network used has some significant effects

	Intranets	Extranets	Public
Info about Users	High	Medium	Low
Capacity Planning	Possible	Usually possible	Difficult to impossible
Bandwidth	High	Varies	Varies greatly
Ability to set technology	Yes	Sometimes	Rarely

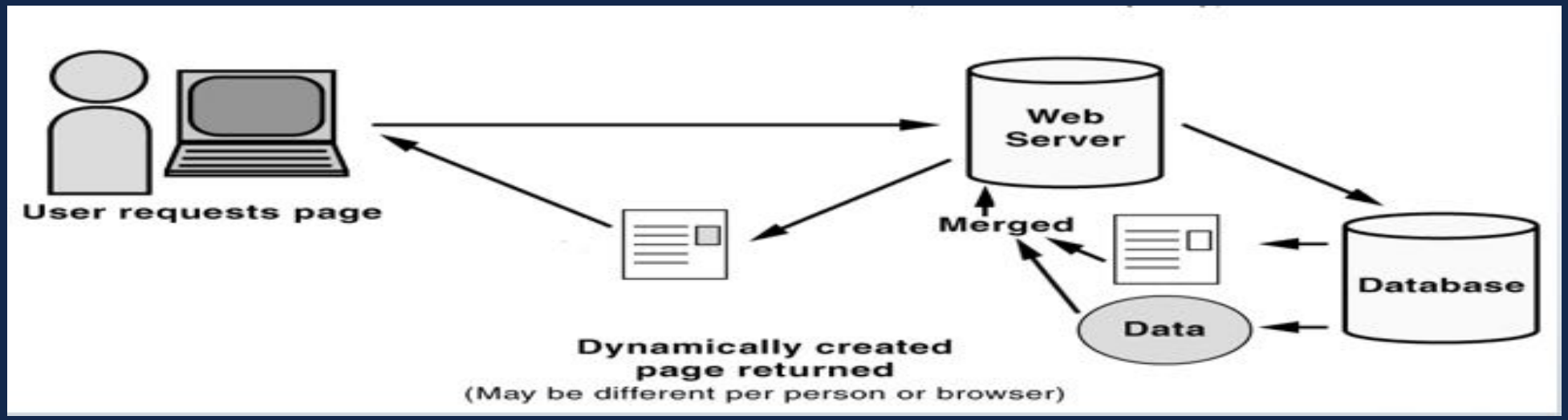
Simple Site Idea



Web server as "file server"

Use HTTP ask for a document by URL such as the home page and it is found and returned if exists and allowed

Dynamic Site



Web server as “application server”

Use HTTP ask for a application by URL and it composes a result usually by binding data into templates. The pattern used is usually an MVC style [Model View Controller]

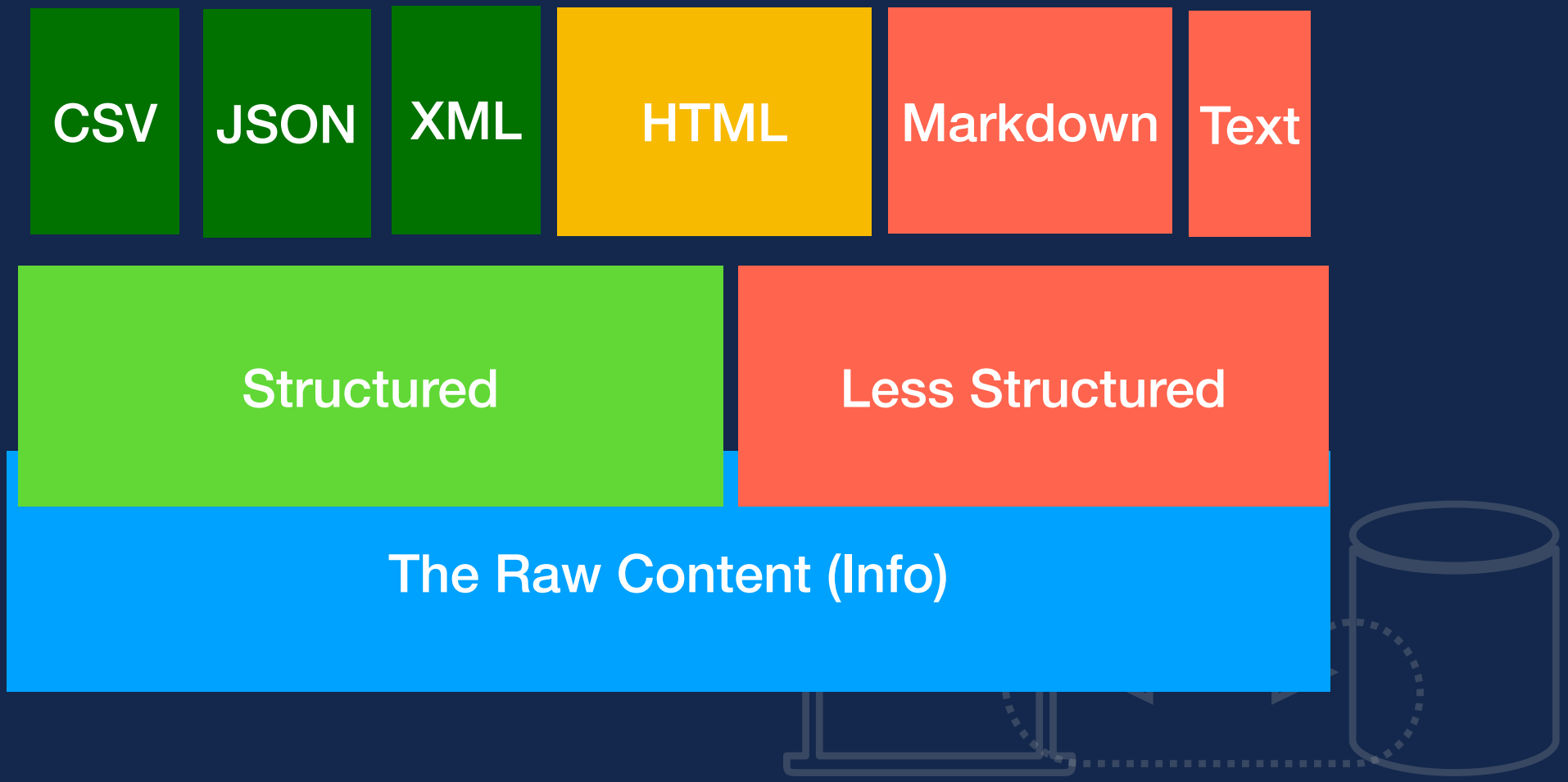
Dynamic Site Details

- Built on the fly for users which allows for
  - Dynamic content
  - Personalized content
  - These types of sites appeal to us in the “programmed” sense, but are they always the right idea?

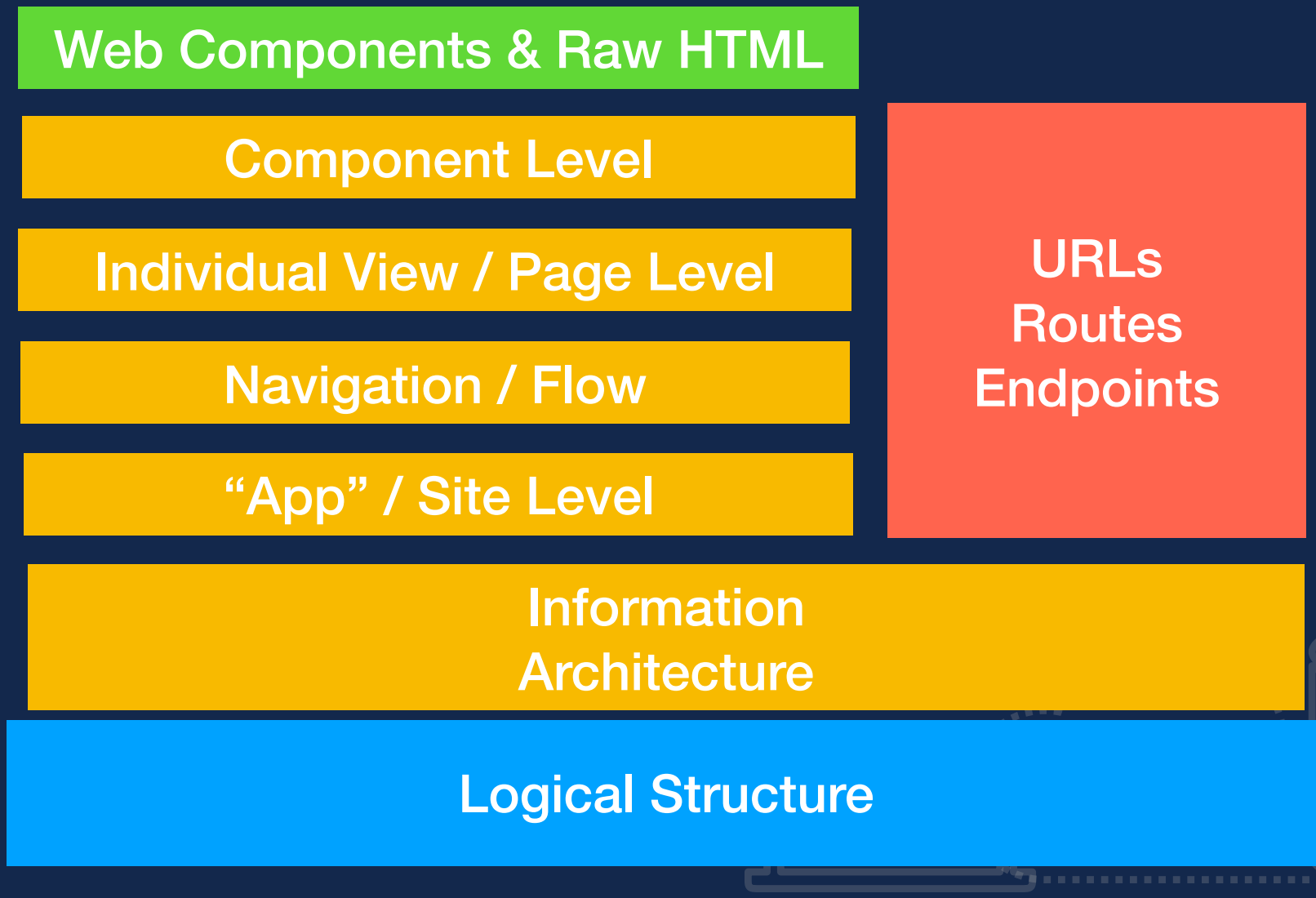
- “Static” Dynamic Site – a database driven site that is static for its visitors
  - Needless complexity
  - Poor Performance
    - Solution: Caching, “Publishing”

- Site approaches need to fit their purpose! Finally seeing this acknowledged
  - JAM stack (https://jamstack.org/)
  - SSG - Static Site Generation (ex. Https://www.11ty.dev)
  - SSR - Server Side Rendering

Content Drill Down



Structure Drill Down



Structure Drill Down

Virtual Components

Ad hoc Components

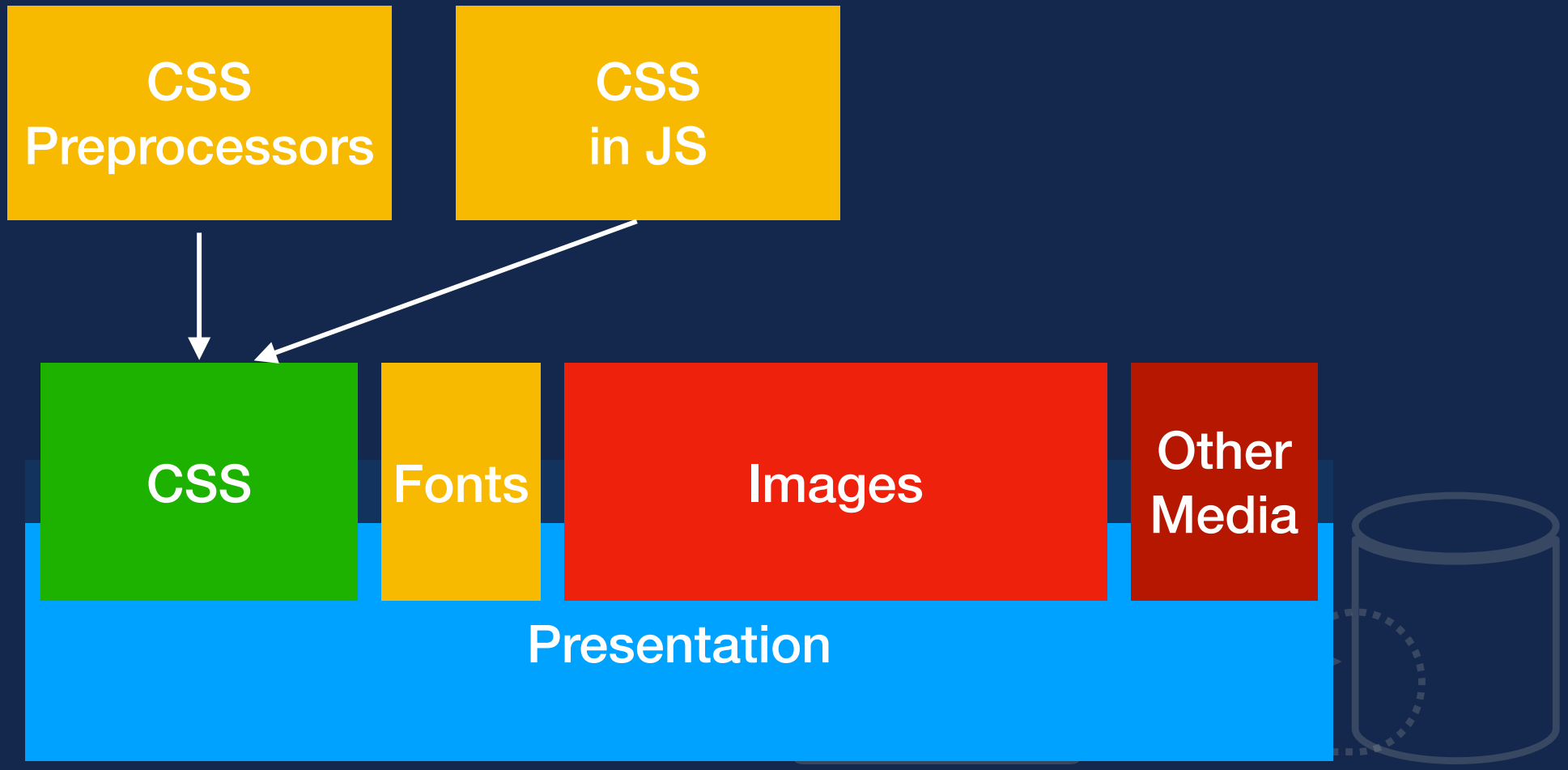
Web Components

Standard HTML Element

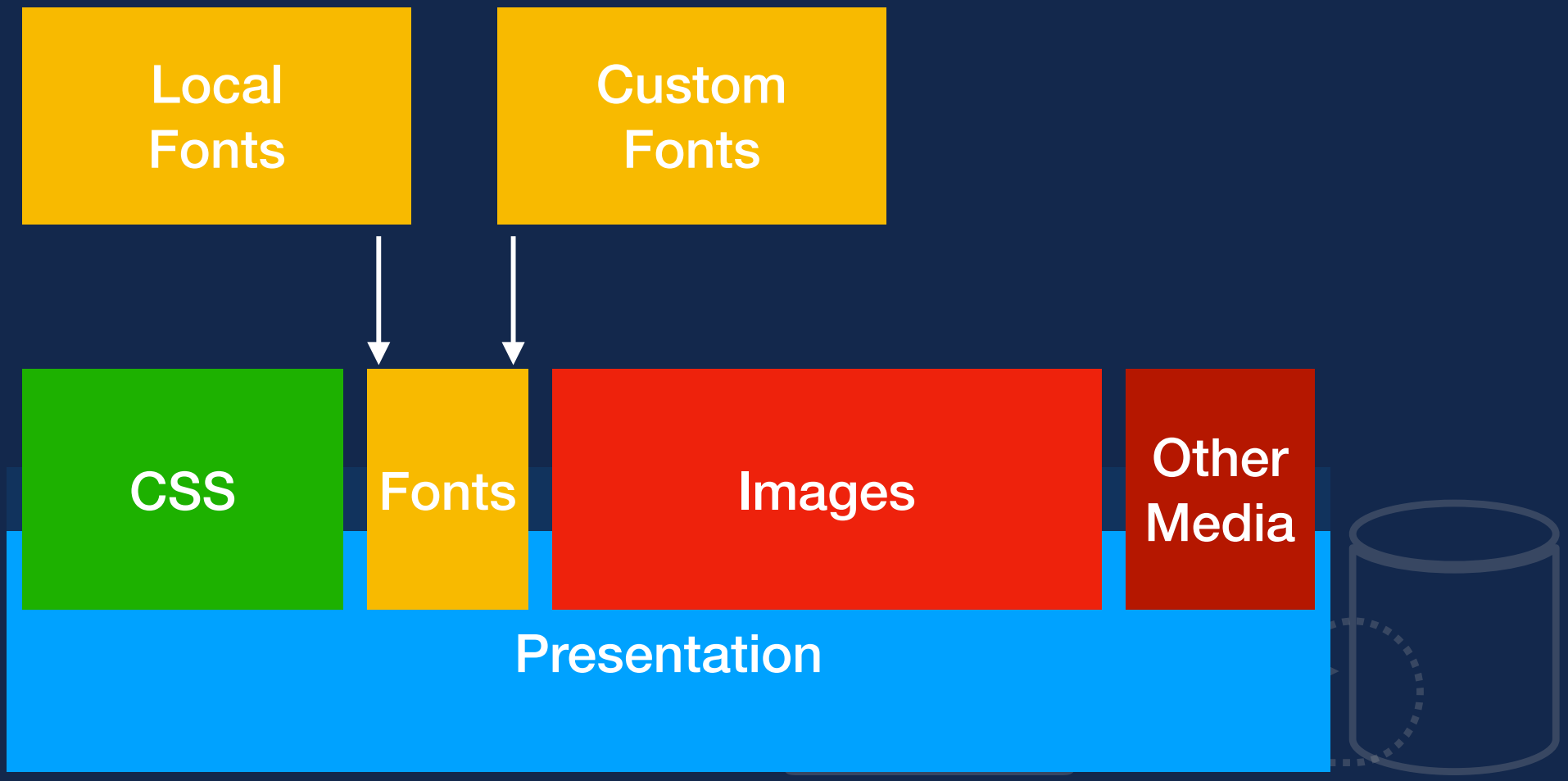
Static Custom Elements

Component Level

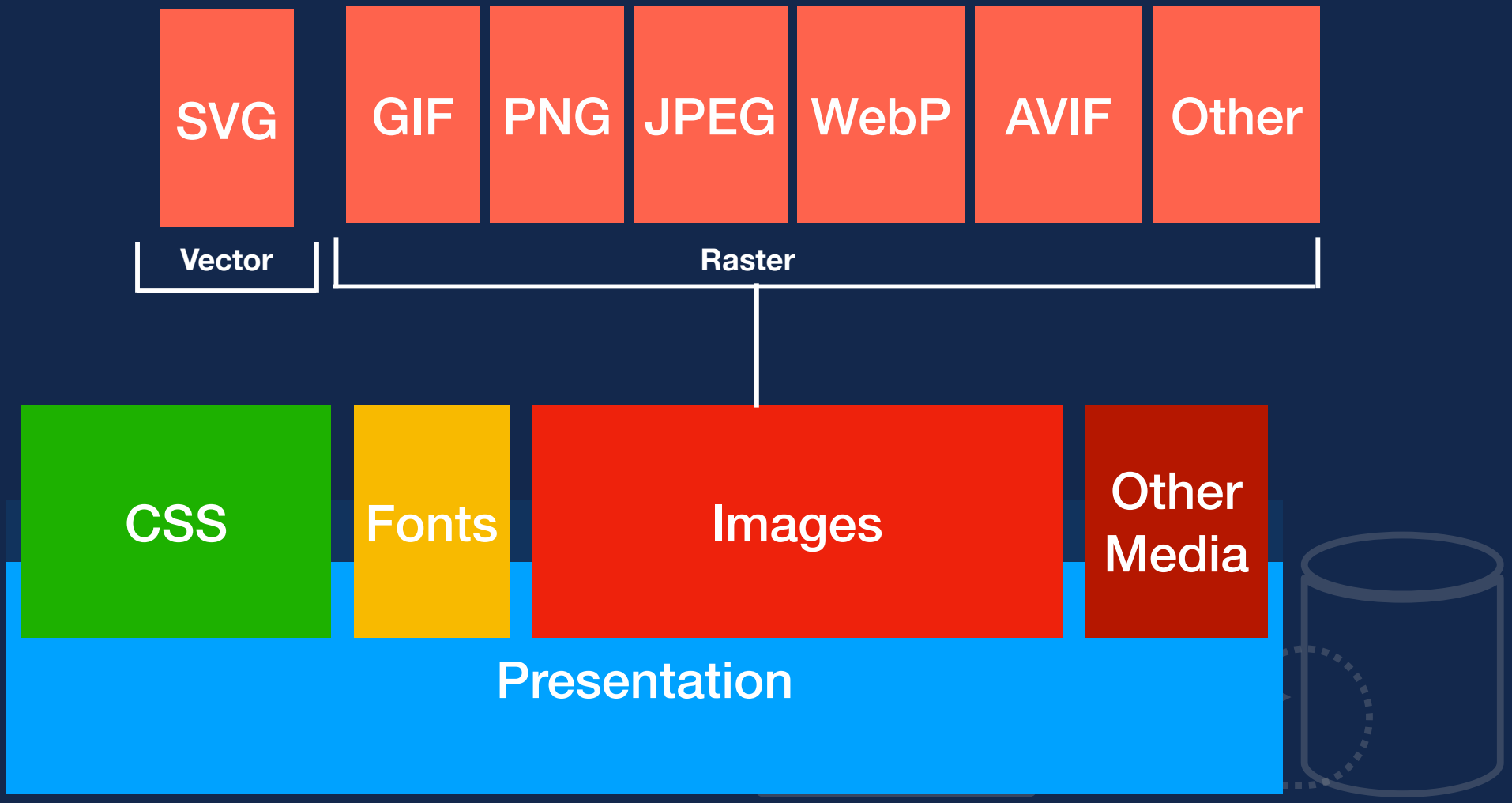
Presentation Drill Down



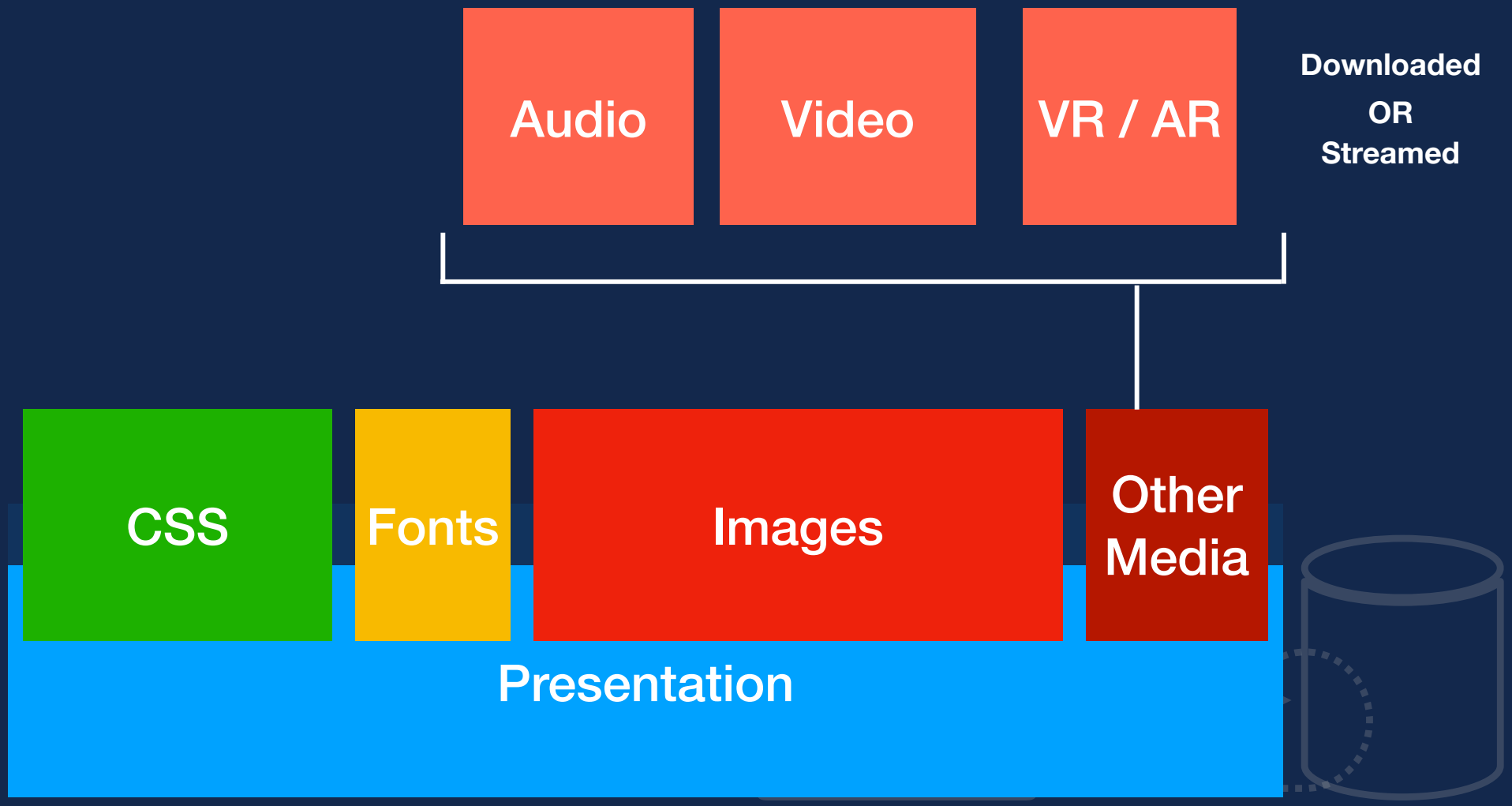
Presentation Drill Down



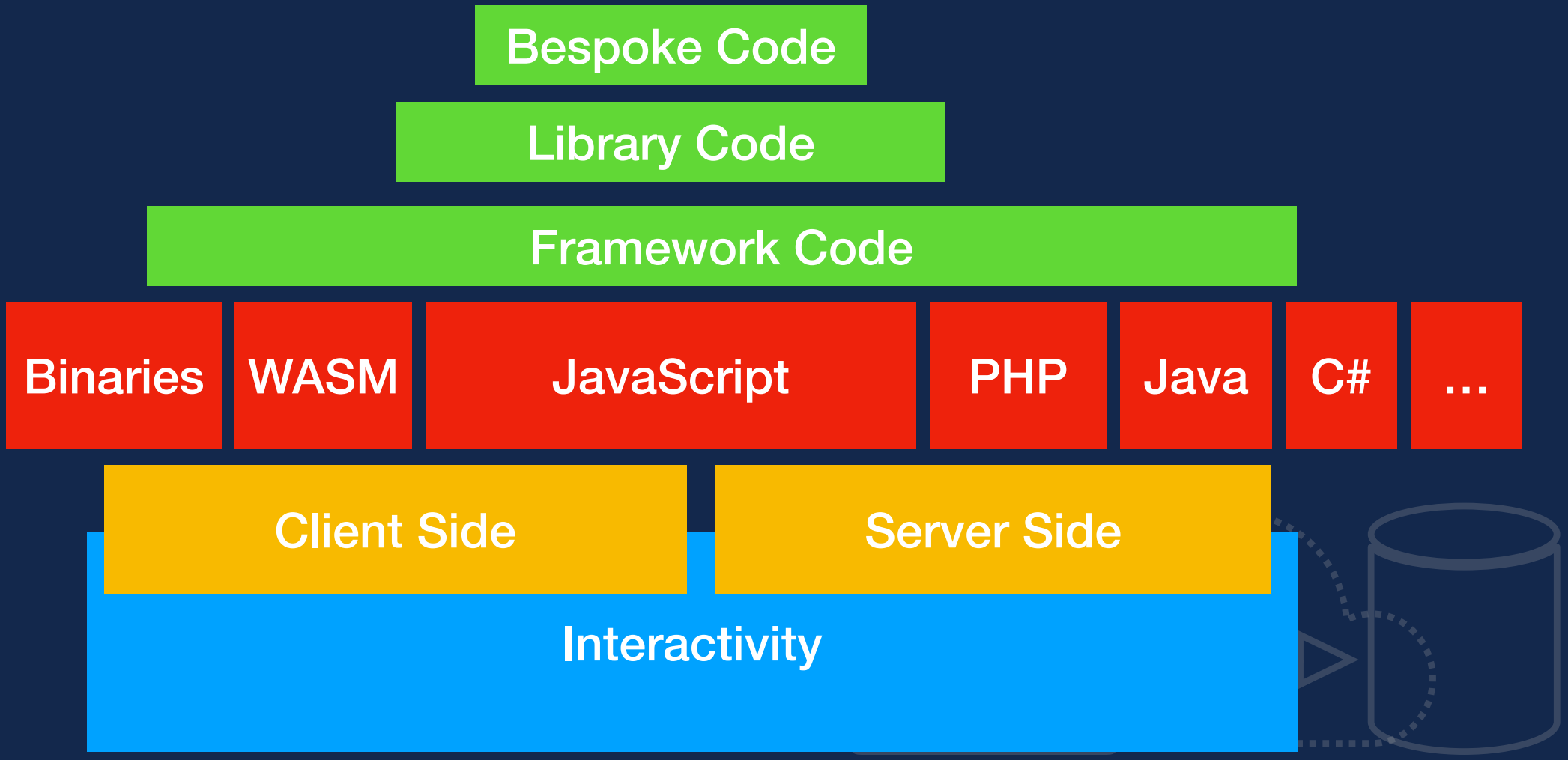
Presentation Drill Down



Presentation Drill Down



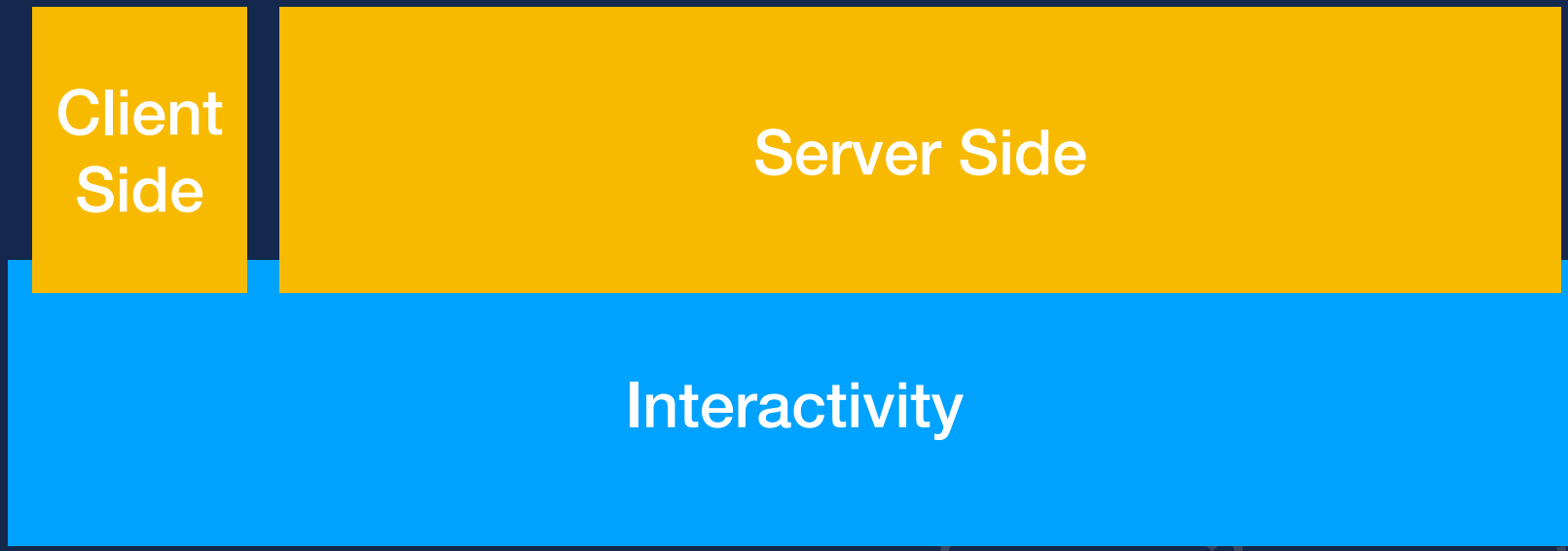
Interactivity Drill Down



Server Centric

Mostly Presentation
Limited Client Side Interaction

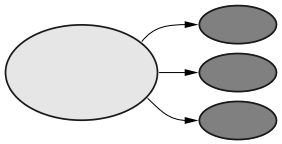
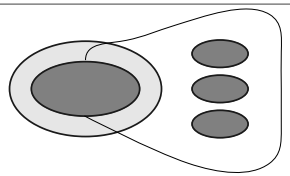
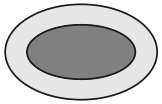
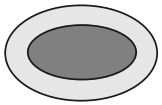
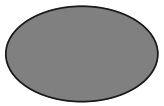
Authentication, Storage, Business Logic,
Page/View Rendering, etc.



Thin Client

Server-Side Models

Visual Models

Model Name	Visual Model	Examples	Portability	Performance	Difficulty
Fork/Exec		CGI, PHP*, Python (WSGI)*, Ruby (Rake)*	Variable* (Low-High)	Variable* (Low-Medium)	Variable *^ (Medium-High)
Server-Side Scripting		PHP, ColdFusion, Classic ASP, JSP*, Python*, Ruby*	High	Medium	Low
Embedded Container		Java Servlets	Medium	Variable^ (Medium-High)	Variable^ (Medium-High)
Embedded Native		Apache Modules ISAPI	Low	High	High
Code as Server		Node, Python, Perl,...	High	Variable^ (Medium-High)	Variable^ (Medium-High)

*Depends on language
^Depends on architecture

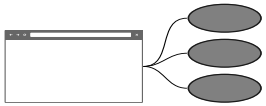
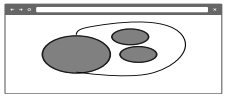
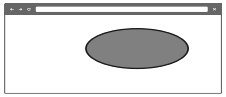
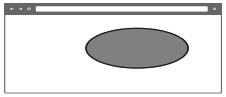
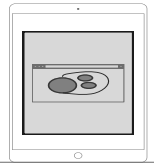
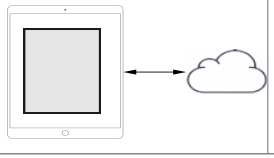
Client Centric

Significant Business Logic,
Page/View Rendering, etc.

Mostly Authentication & Storage
Possibly Compute

Fat Client

Client Side Models

	Model Name	Visual Model	Example	Portability	Performance	Difficulty
Browser Based	Helpers		External Launched Apps for MIME streams (ex: Winzip, Acrobat, etc.)	Variable* (Low-High)	Variable* (Low-High)	Variable *^ (Medium-High)
	Client Side Scripting		JavaScript ~~VBScript~~ WASM	High	Medium-High	Low - Medium
	Applet		Java Applet	Medium	Variable^ (Medium-High)	Variable^ (Medium-High)
	Plug-in		Active X Control (Netscape) Plug-in NaCL & PNaCl	Low	High	High
Native App Based	Native Code Wrapping Web View		Electron Apps	High	Variable^ (Medium-High)	Variable^ (Medium-High)
	Native Code Calling Web		Swift, Java, ObjectiveC using HTTP and some web tech	Low	High	High

*Depends on language
^Depends on architecture

Web app vs Native app

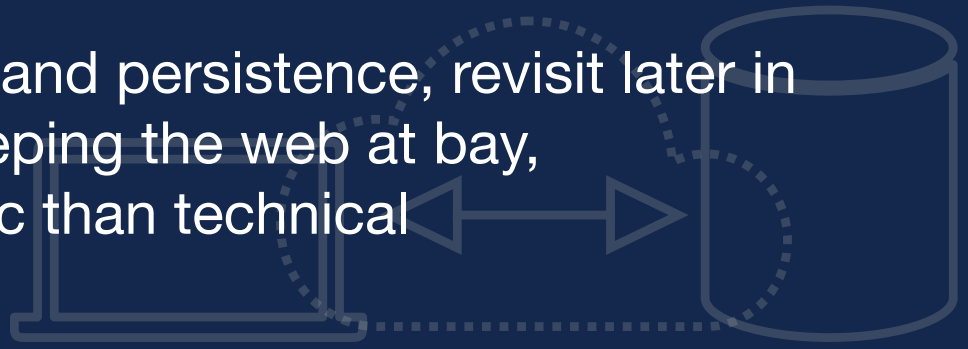
Native apps are better than Web apps?

- Language/Tech (Swift/ObjC, Java vs Web Tech), installed, direct OS vs in browser
  - Native app can be written using web tech even!
  - Higher speed of native app? Better check into that.
  - OS access - better check into that too.

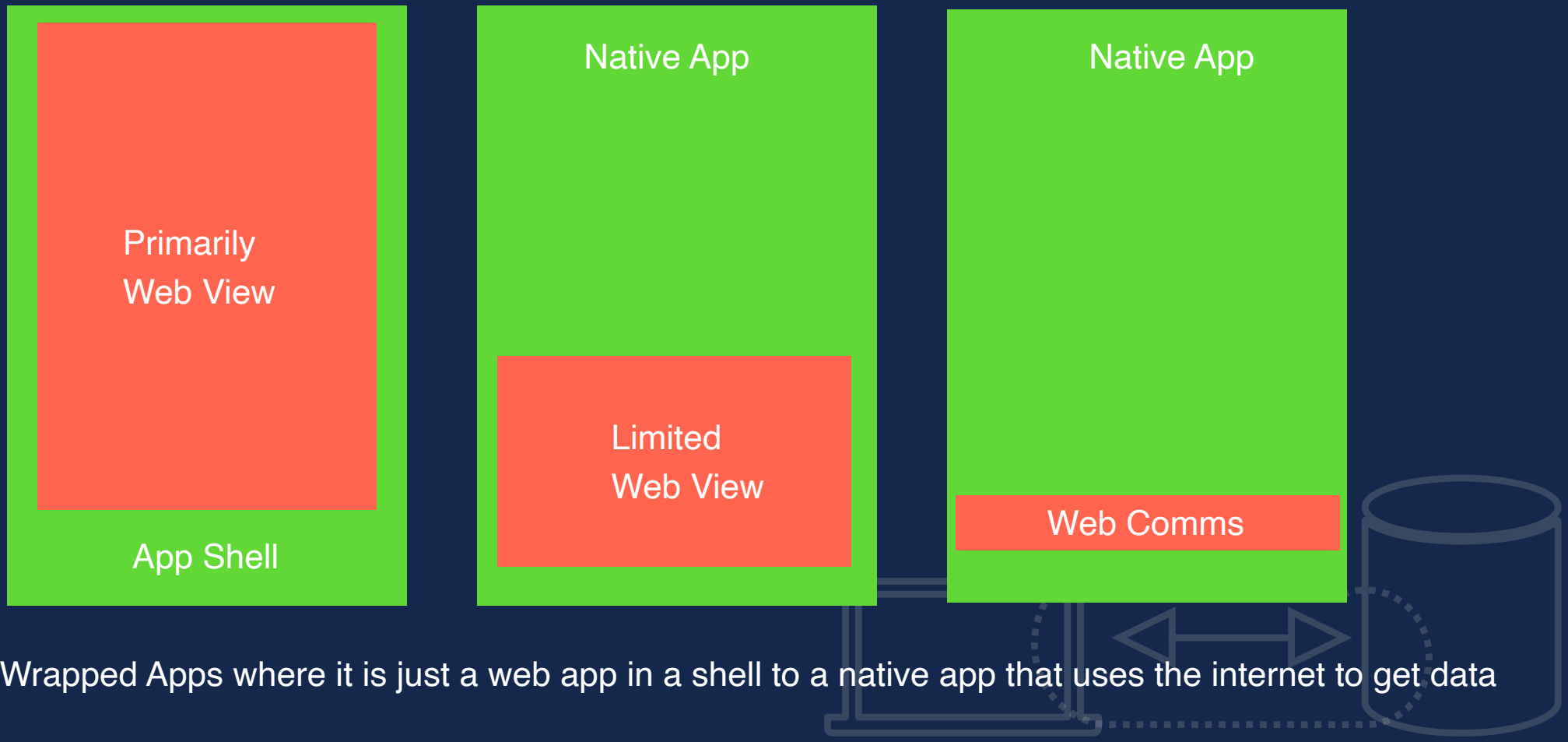
But native app may use network or browser

- App uses HTTP to send and receive XML, JSON, etc.
- Native app wraps a “web view”, basically an embedded browser - app shell pattern

Real issue has to do with distribution and persistence, revisit later in the course. Careful Apple is busy keeping the web at bay, distribution effects are more economic than technical



WebViews, App Shells and Just Networked Powered Apps is a Big Range



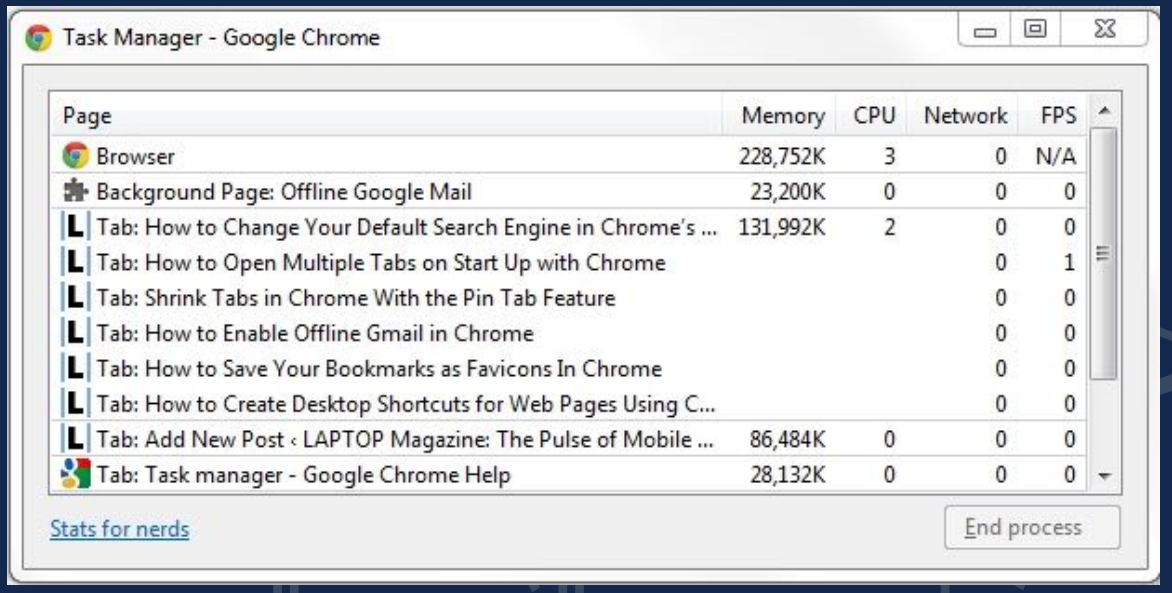
Wrapped Apps where it is just a web app in a shell to a native app that uses the internet to get data

Browser as Client

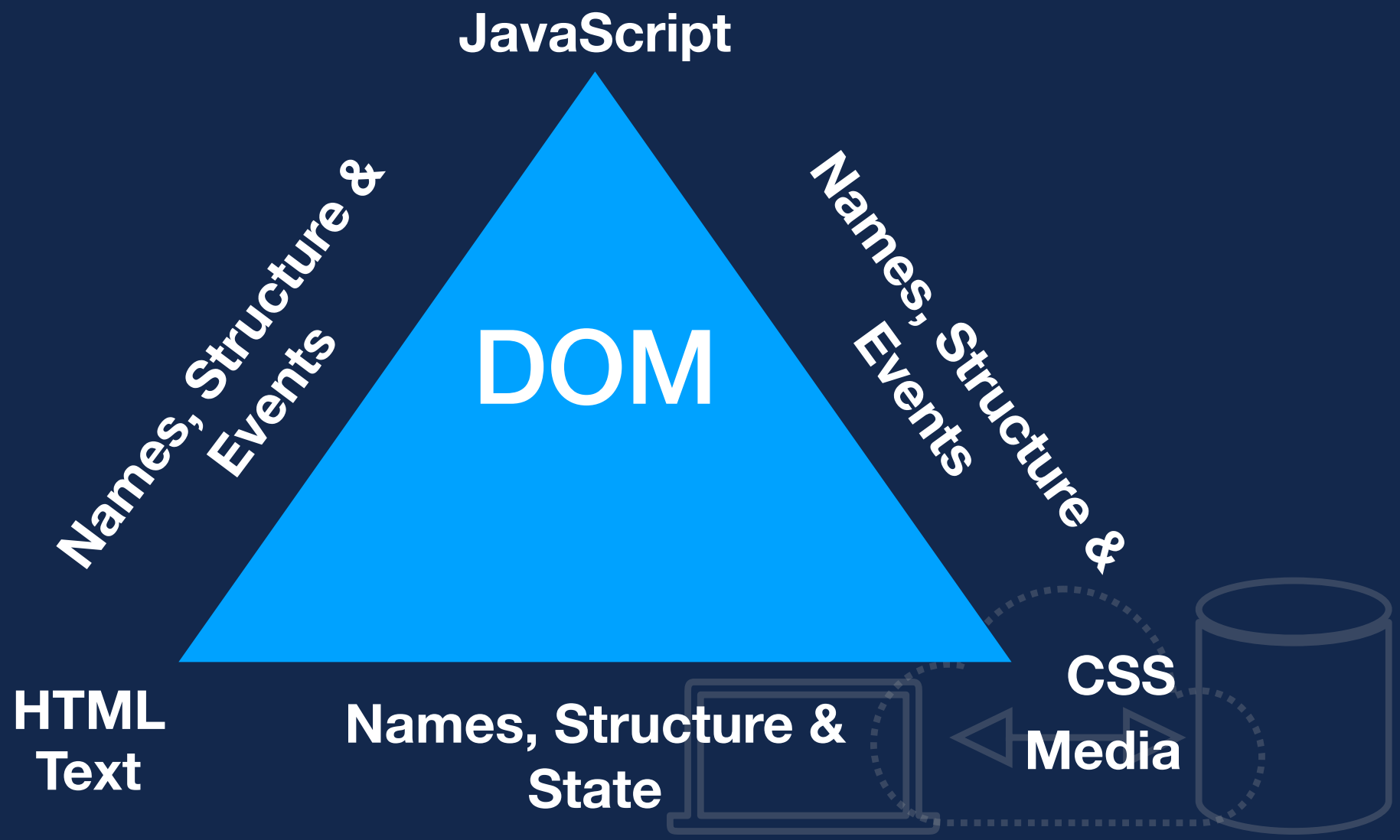
- If the browser is the client, the app runs in the context of the Web browser (Chrome, IE, etc.) and is sandboxed in a sense by it.
- In this pattern, the browser is the OS for the web app

🤔 Chrome OS I get it now

- Aside: Browser Stats and browser engines

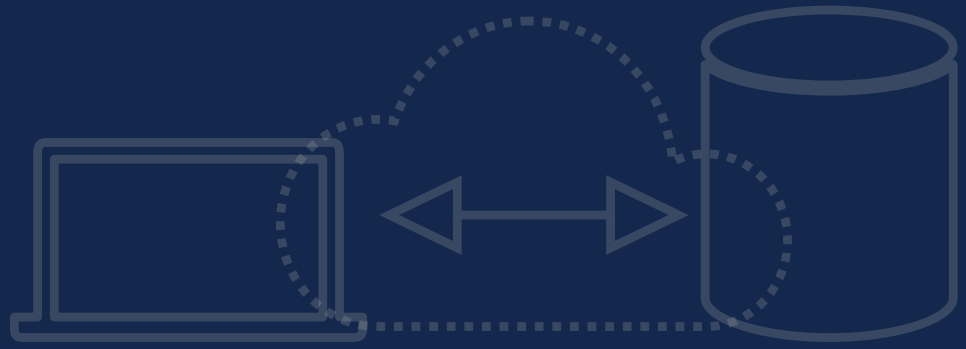


Client Model Alt Form

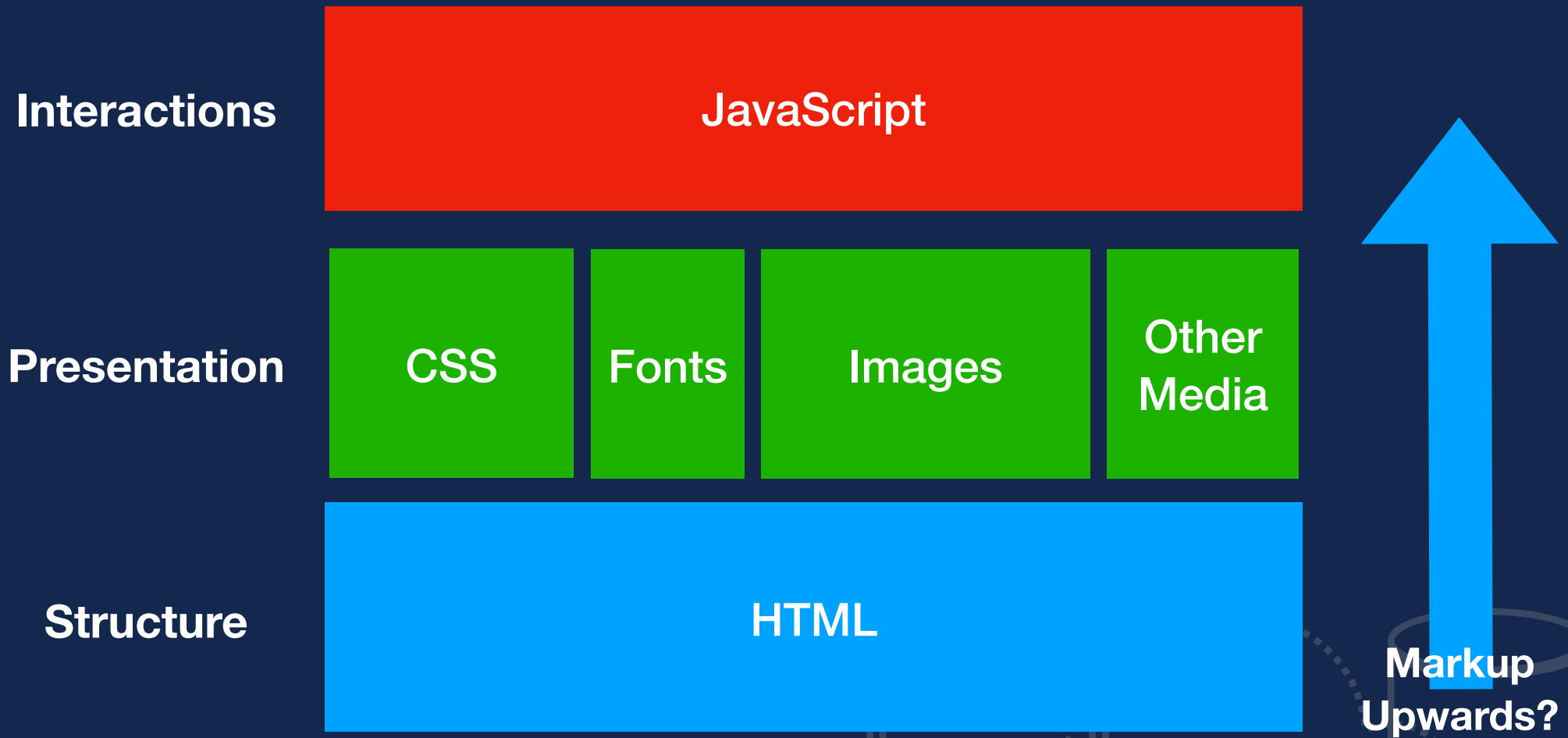


How to Apply This

Looking for the “Best” Answer Part 1 of Infinity



Progressive Enhancement Model



Progressive Enhancement has us add technology one layer at a time

Good for sites as it degrades without script and lookup

Graceful Degradation Model

We start with requirements and degrade with grace letting people know about issues. Careful consideration of a11y, SEO, resilience, etc. must be performed.

<u>Good for apps</u> where there is a minimal requirement to use

Approach Tension

We just saw two competing philosophies with adoption driven both by actual need (site vs app) as well as developer comfort (markup first vs code first)

Yet code, markup and style is in every execution

You can hide, but you really can't separate

Sure you can loosely couple, but a full decouple is pretty much impossible

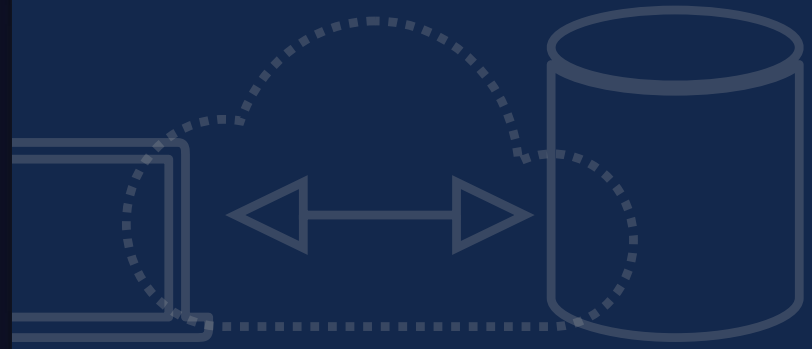
Answer the Right Way Riddle Part 1

Is your markup in my code?

```
let school = 'UCSD';
let major = 'CSE';
let str = `<p>Hello <strong>${school}</strong> <em>${major}</em> students!</p>`;
```

Is your code in my markup?

```
<script>
  var major = 'CSE'; var school = 'UCSD';
</script>
<p>Hello
<strong>
 <script>document.write(school);</script>
</strong>
<em>
 <script>document.write(major);</script>
</em>
 students</p>
```



The Answer

It Depends™

The appropriate answer may reveal itself by

The volume of each

The type of the problem

The practitioners involved

The practitioners in may influence the answer greatly

The Answer Contd.

Non-appropriate answers tend to

Focus mostly on dev interests
Overly celebrate automation
Overly celebrate purity
Assume “one true way”

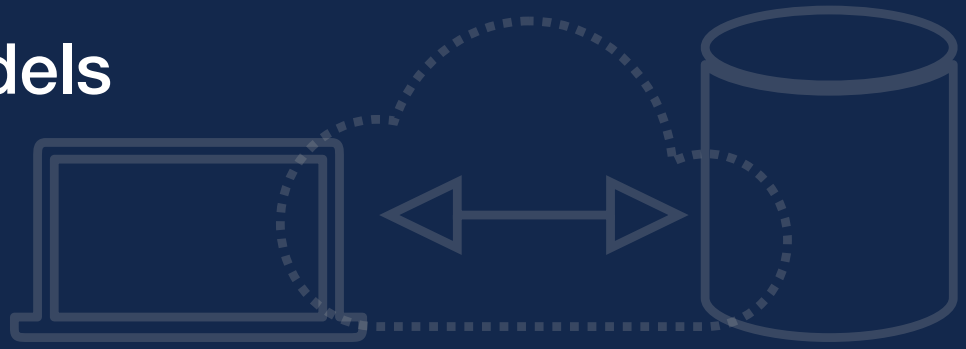
And more often than not solve problems we don’t have yet*

The Answer Contd.

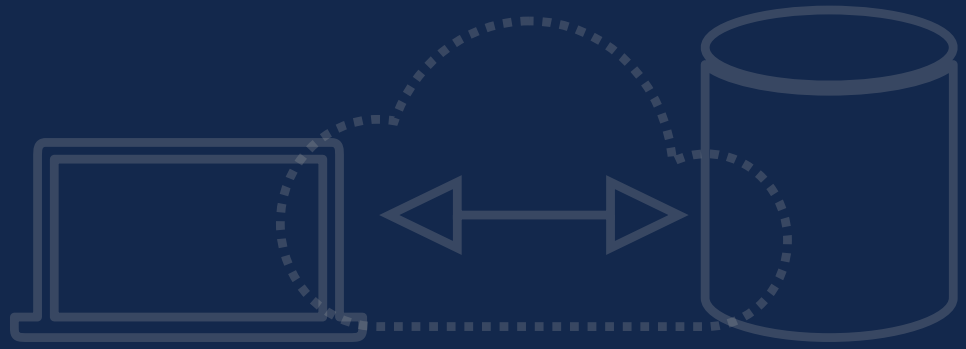
All the models presented so far have been quite focused on the big three (HTML, CSS and JS) in a browser

Other browser models exist with their own tensions and concerns

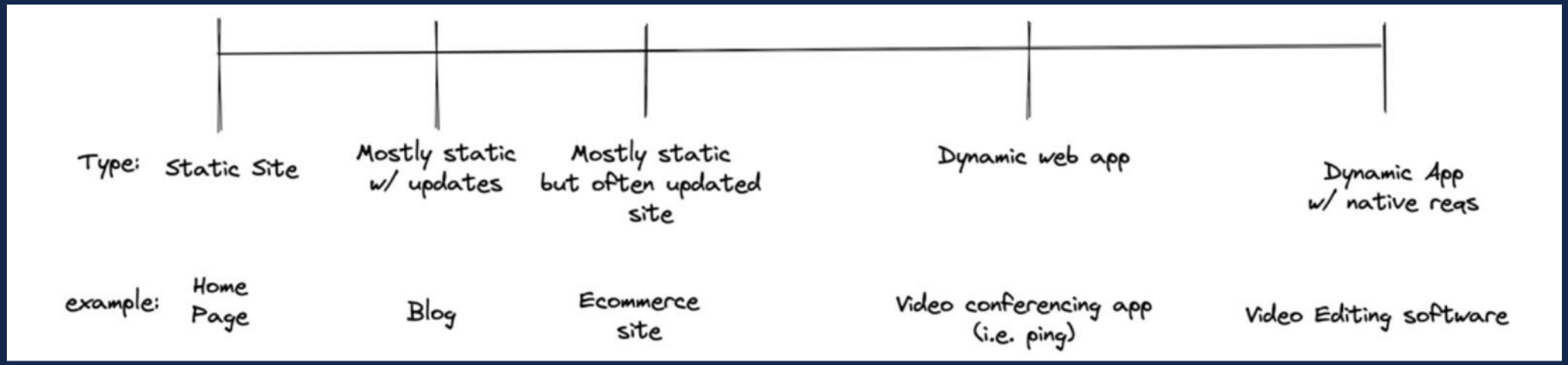
Also there are non-browser models

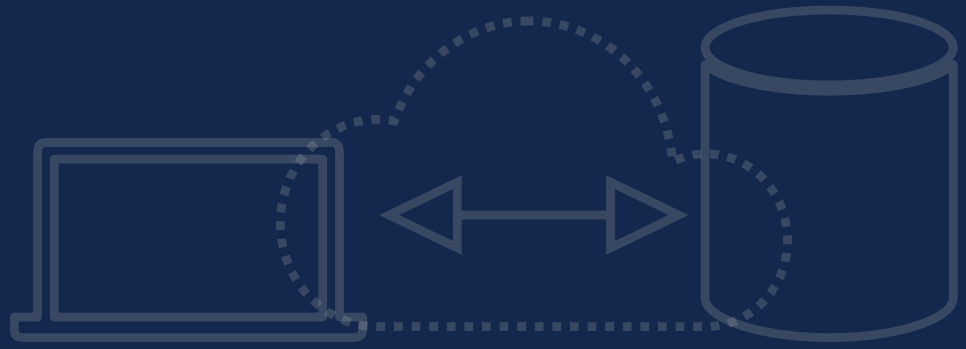


Take 2 - What Pattern Should We Choose?



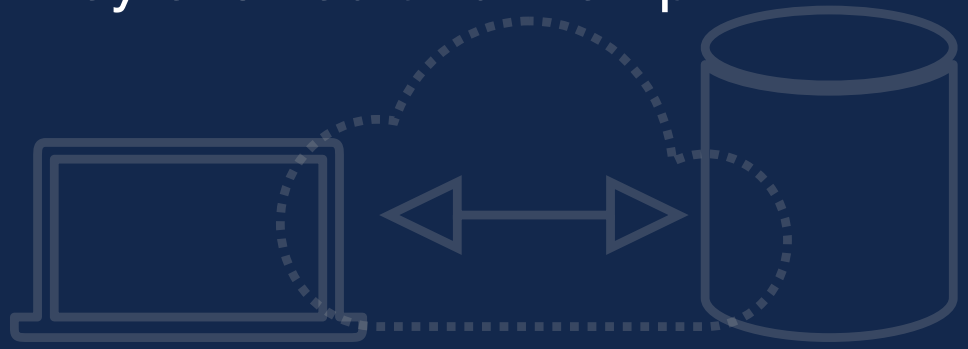
Is the Form Always Clear?





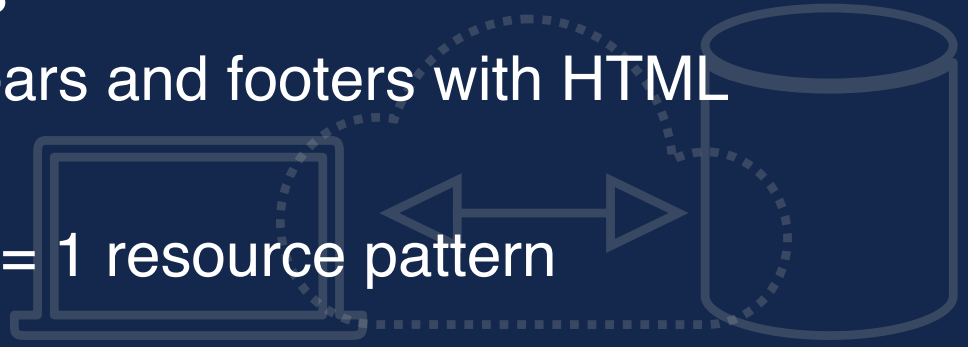
Web Patterns

- The way we have approached web solutions has changed over the years
- Commonly there is an assumption that an emerging or popular way is the right way, but we should be cautious to such “consumer” style engineering thought
- Let’s survey the patterns and how they evolved and then pick



Gen 1 - Server Side Focused

- Thin Client Pattern
  - Pros: Simple (Rule of Web Below) Secure-able, Controlled
  - Cons: Slow due to network round-trips = bad interactivity
- Rule of Web - 1 page = 1 URL Pattern
  - Pro: Predictable, Back Button, “Easy” to code/reason
  - Con: Full page refresh, problematic usability
- Partial refresh solution: Frames
  - Pin static portions like nav bars and footers with HTML frame
  - Big problem: Breaks 1 URL = 1 resource pattern



Gen 1.5 - Server Side Enhanced Client Side

- Round-Trips are a problem and interactivity is problematic so fix it with some optional JavaScript
  - Introduction of *progressive enhancement* practice
    - Standard functionality for those without tech, enhanced situation for those with support for more modern functions

- Common examples
  - Form validation – check local for UI, server for security
  - Client Side interactions – tooltips, rollovers, etc.

Generation 2 – Client Side Take 1 – “Ajax”

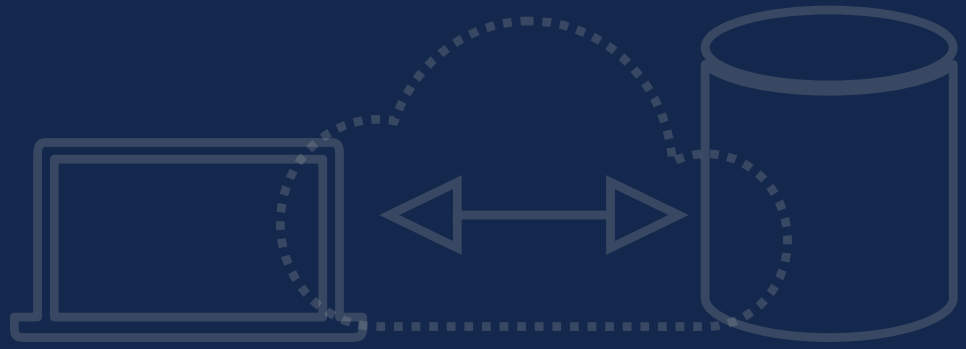
- Using JavaScript to update pages in place with behind the scenes communication using the `XMLHttpRequest` object
  - Pro
    - Significant speed improvements possible
    - Richer interfaces
  - Cons
    - Complexity
    - Fragility and Network Concerns
    - Break 1 URL = 1 Resource pattern
      - Fix using “shebang” style #!state eventually use state API

Generation 3 – Client Side Take 2: Native Apps

- Two dominant operating systems with 3 languages generally used
  - iOS – Objective C, Swift
  - Android – Java
- Can* be fast native execution
- App Store Distribution Patterns
  - Serious Pros and Serious Cons
- Reactions
  - Web Views, Connectivity Requirements, ”Small Fixes” constant updates

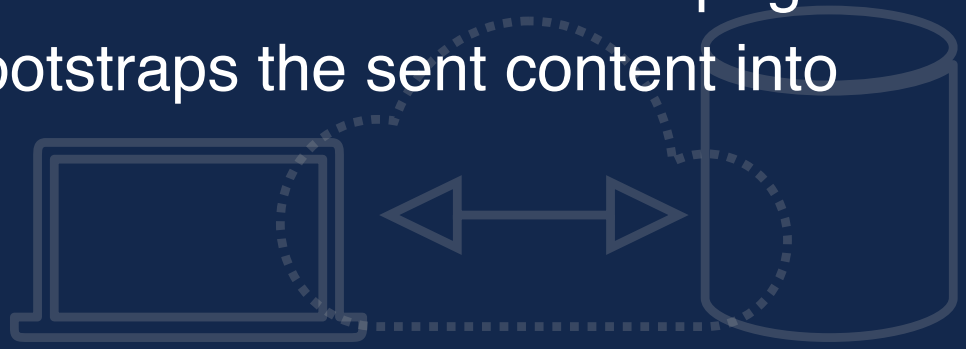
Generation 4 - PWAs

- Generation 3 assumptions
  - JavaScript can’t be fast enough
  - JavaScript/Web APIs lack some power
- Those assumptions are actually false today
- Progressive Web Apps (PWAs) is a design pattern often dubbed offline first that focuses on speed of native with the ease of distribution from the web



Next Generation? SSR + Hydration

- Given client-heavy interactions often come with a non RAIL friendly download or “install” penalty we are driven to adopt an SSR + Hydrate pattern
- SSR = Server Side Render
  - Take a dynamic page and “static”-ize it to send a snapshot or a partial scaffold quickly
- Hydration
  - After initial load ”energize” / “animate” / “enhance” the page by applying JavaScript that bootstraps the sent content into a truly interactive application

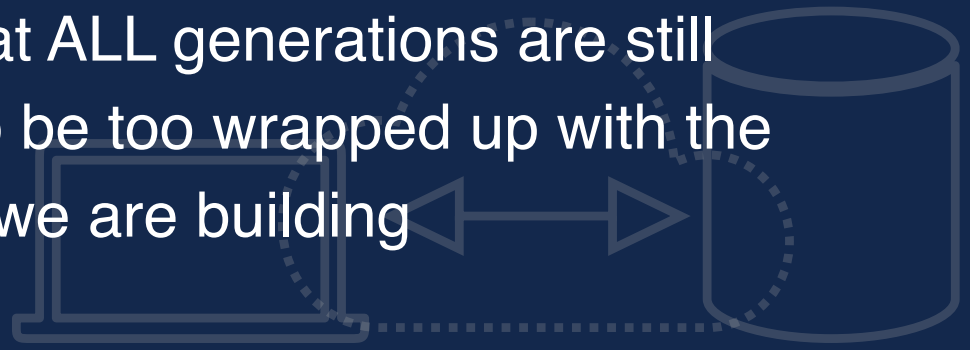


So…then SSR?

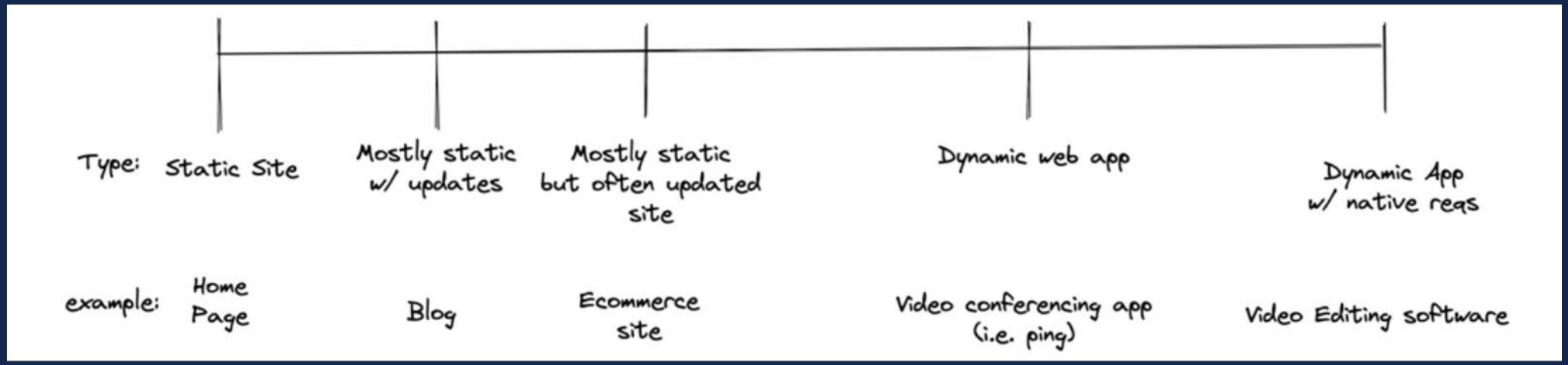
- Remember the point here isn't picking something new it is picking something appropriate

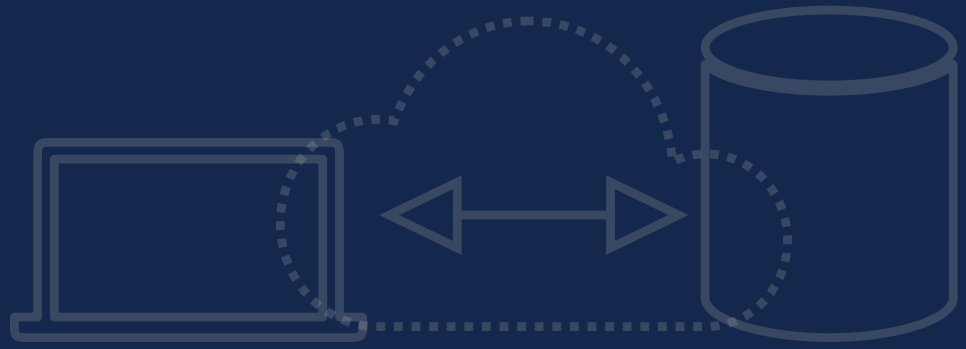
- First is to think of the form - site or app
- Second is to think of the degree of dynamism (change)
- Third is to think of the context of the site / app
- Finally think of the constraints of the situation (time, $, etc.)

- What we see if we do the work that ALL generations are still legit and to believe differently is to be too wrapped up with the form and not the function of what we are building

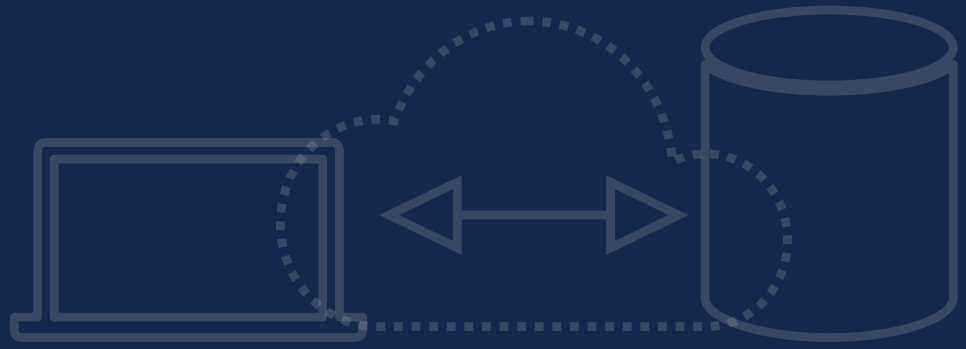


Nothing changed with the app, so why is the solution now SPA or SSR or … for all?





Solutions Can Reveal Themselves with Thought



Form Follows Function

- Traditionally the phrase “The Form Follows the Function” is used to imply that what something is and how it should be built is driven directly by the purpose of the thing
- If we follow this along properly we can see that in many cases the type of site or app should follow from what we are doing and who we do it for
- Let’s illustrate this with an example of considering using web technology to make a data visualization

Form Function Example: Viz Tech Chooser

1) Is the info we want to visualize … ?

Conceptual Measurable Mixed

2) Is the purpose … ?

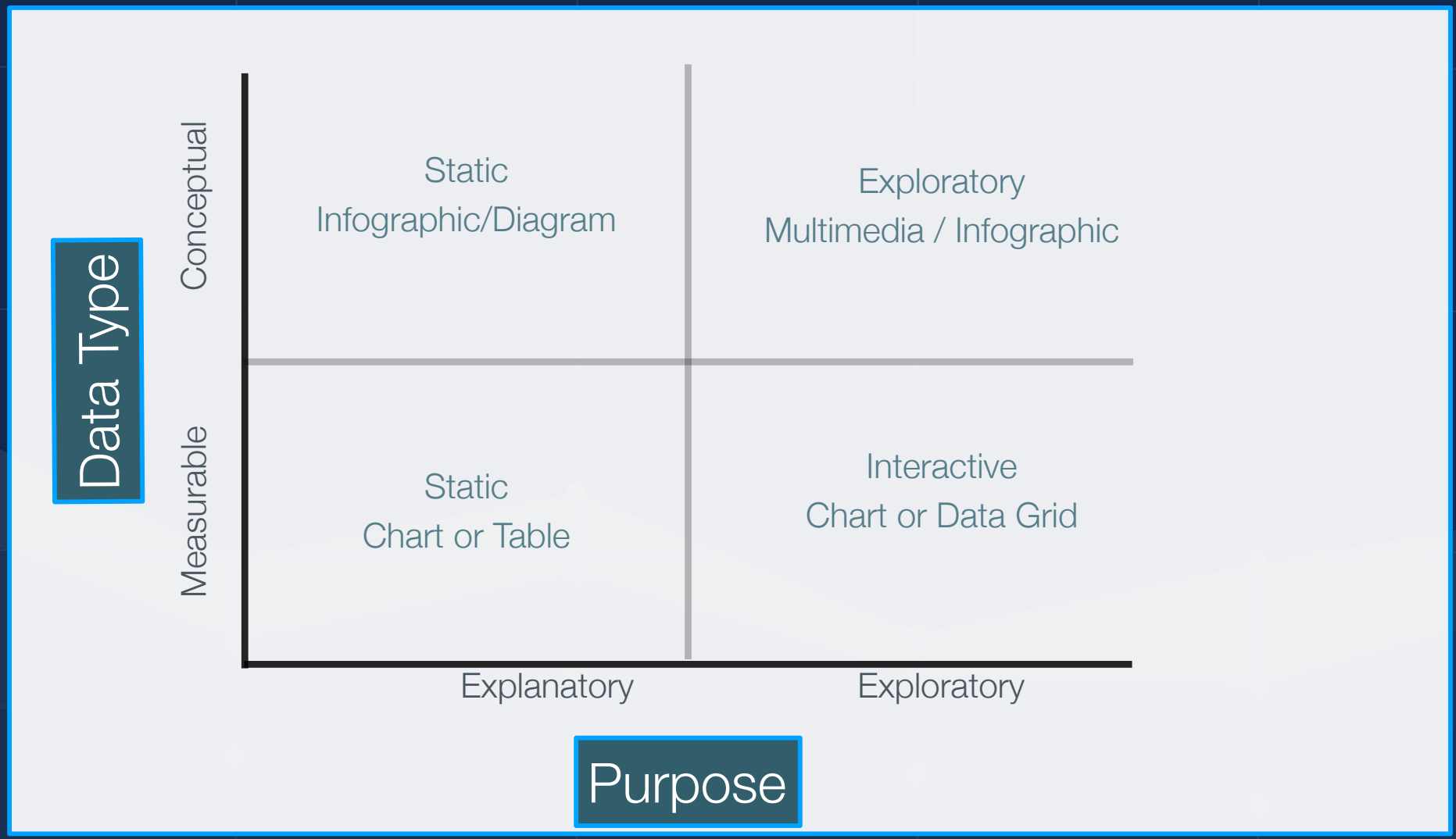
Explanatory

to state or declare a particular finding ?

Exploratory

for the viewer to explore and determine a finding ?

Form Function: Choosing Presentation



Form Function: Interactivity Continuum

Static Image | **Server-Side Rendering** | **Client-Side Rendering** | **Client-Side Rendering or Client Side App**

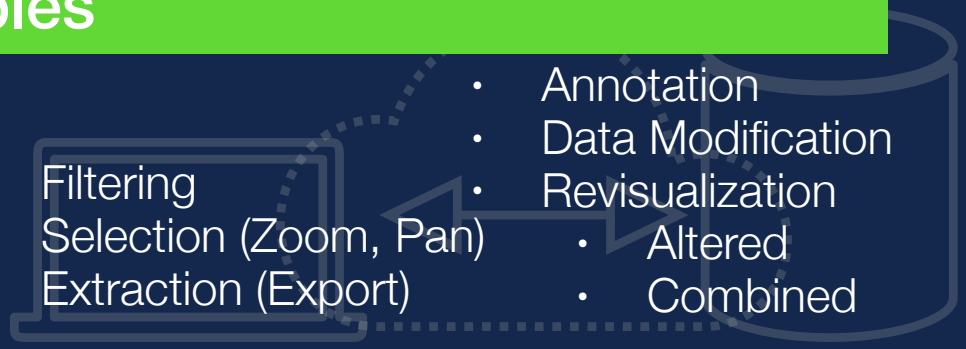
Appropriate Execution

Less ← **Passive Inspection** — **Active Inspection** — **Moderate Manipulation** — **Heavy Manipulation** → **More**

Degree of interaction

Examples

- Animation and Guided Presentation
- Tooltips
- Interactive Guides
- Legend Toggles
- Filtering
- Selection (Zoom, Pan)
- Extraction (Export)
- Annotation
- Data Modification
- Revisualization
  - Altered
  - Combined



Form Function: Technical Selection Continuum

Less

More

Interaction

Server-Side

Client-Side

Unfortunately that conflicts with…

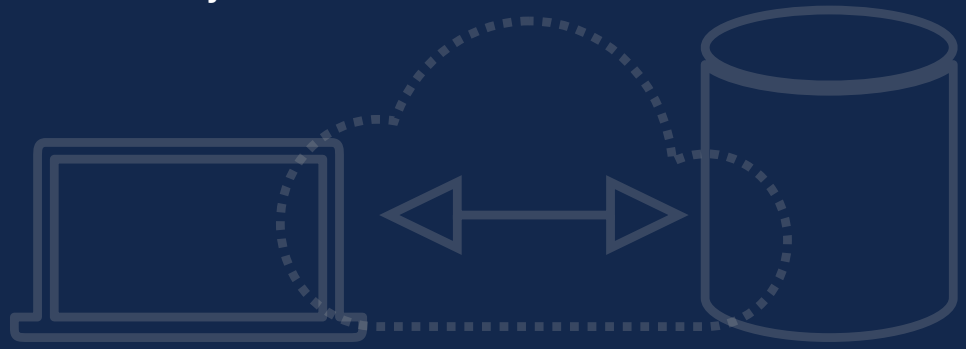
Data

Client-Side

Server-Side

Setting Expectations Redux

And how data might bring some sanity…

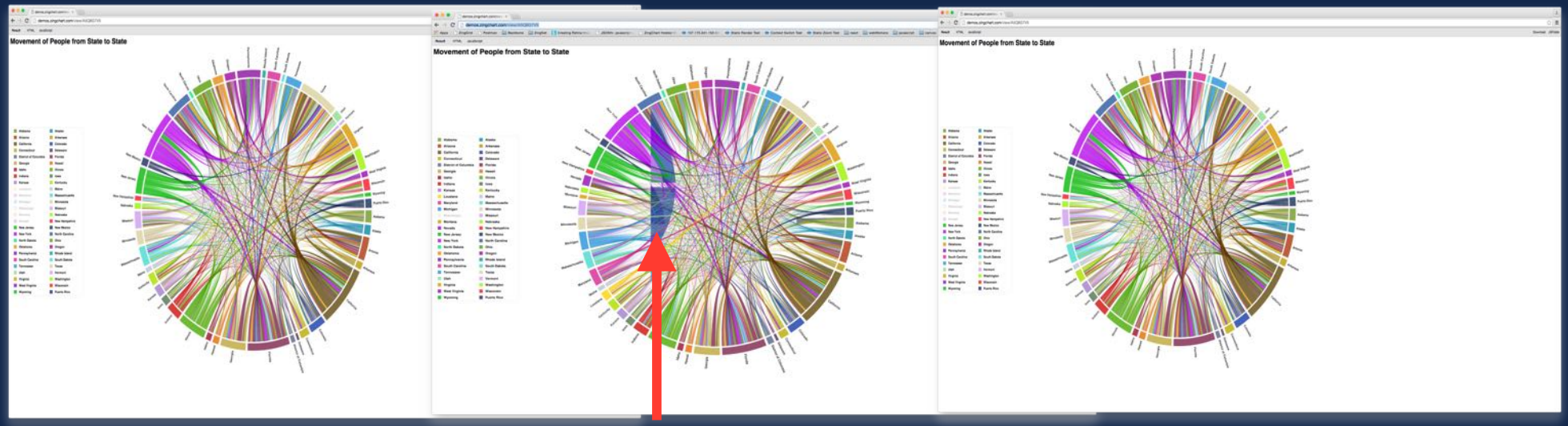


Reasonable Expectations Redux

Belief: Web Dev is Hard!
Possibility: Verify with data and measurement?

Example: Browser Stability

Browser Version X | Browser Version X+1 | Browser Version X+2

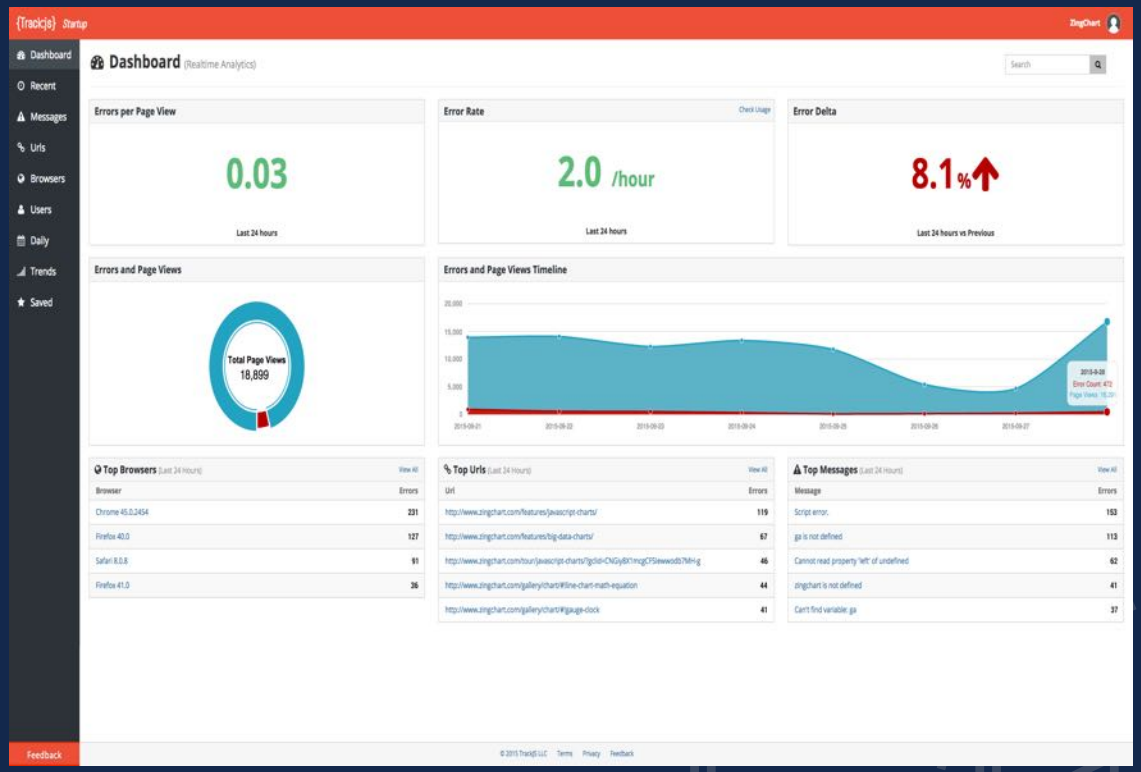


Correct Render | Visual Distortion | Correct Render

Code Constant, Browser Version Only Variable

Example: Errors Happen

Your machine and browser isn't the world, errors happen!



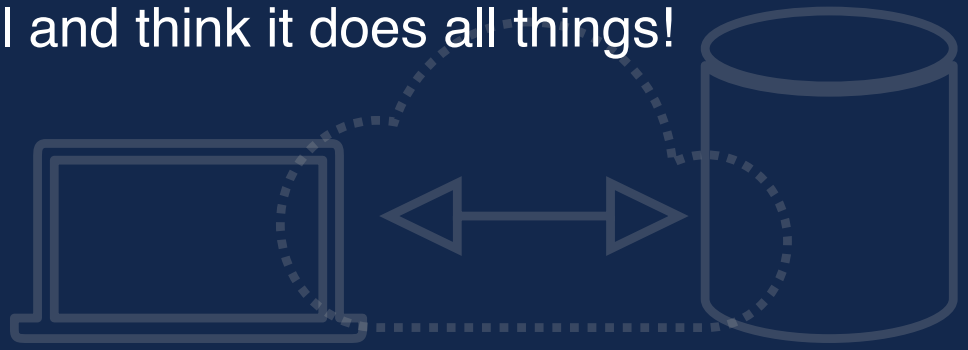
Monitor client errors and see reality or wait until users report them?

The Importance of the Last Slide

- So full circle now. This last slides shows we can see that we can capture information about how “good” or “bad” the technical execution is when it hits “REAL” users.
  - Localhost Effect callback!

- Premise: We can know all many things, but what we want to know and what questions that answers it the main thing.

- Follow-On: We should design the analysis and approach to fit this last premise, not choose something general and think it does all things!



Summary

- Review the web medium and the primary mental models to understand the context of online analytics
- Respect the core aspects of the client/server medium to avoid huge mistakes
- Drill into the layers of the tech to see the complexity and volume required for mastery
- Avoid looking for perfect solutions to your building and collection problems as they generally don't exist at least not in a permanent sense