CSE127, Computer Security

Catch up Systems Sec II

Intro to Web

UC San Diego

Housekeeping

General course things to know

- PA2 due tonight!
  - Godspeed
- PA3 released at midnight!
- Midterm is **2/12,** during class hours, location TBD
  - Class topics will go through web security (2/5) and include PA3 material
  - One sheet of paper front and back is allowed as a “cheatsheet”
  - No practice midterm, but will put up some sample questions so you get a sense of the style

Meltdown, again

Recall from last time…

- Microarchitecture makes all kinds of optimizations that aren't in the architecture spec
  - Pipelining
    - CPU processes instructions in a **pipeline**; so many instructions can use the CPU resources at the same time
  - Out-of-order execution
    - CPU will opportunistically execute instructions that it thinks it can do because there's no reliance on previous instructions

Out of Order Side Effects

CPU Thought Process

```
if (x < array.length()) {
    value = array[x];
}
```

Out of Order Side Effects

```
if (x < array.length()) {
    value = array[x];
}
```

CPU Thought Process

This conditional is going to take a long time…

Out of Order Side Effects

```
if (x < array.length()) {
    value = array[x];
}
```

CPU Thought Process

This conditional is going to take a long time…

Might as well execute something else; basically, store array[x] in a register

Out of Order Side Effects

```
if (x < array.length()) {
    value = array[x];
}
```

CPU Thought Process

This conditional is going to take a long time…

Might as well execute something else; basically, store array[x] in a register

If x ends up longer than array length, i'll clear the register

Out of Order Side Effects

```
if (x < array.length()) {
    value = array[x];
}
```

CPU Thought Process

This conditional is going to take a long time…

Might as well execute something else; basically, store array[x] in a register

If x ends up longer than array length, i'll clear the register

If x is less than array length, i'm a genius

Out of Order Side Effects

```
if (x < array.length()) {
    value = array[x];
}
```

- Value is still loaded into the cache regardless of if the processor needs to rollback changes.
  - Why?

L1 cache

Out of Order Side Effects

```
if (x < array.length()) {
    value = array[x];
}
```

L1 cache

- Value is still loaded into the cache regardless of if the processor needs to rollback changes.
  - Why?
- Processor can't process another address (it takes too long); all it needs to know is "clear %eax"

Does meltdown work even when I’m doing bad stuff?

- Following code dereferences a null ptr **(bad)** and then assigns something to an array

```
*(char *) 0;

array[0] = 0;
```

Does meltdown work even when I’m doing bad stuff?

- Following code dereferences a null ptr **(bad)** and then assigns something to an array

```
*(char *) 0;

array[0] = 0;
```

- Because of MMU privilege check, the first line will segfault the program, but…

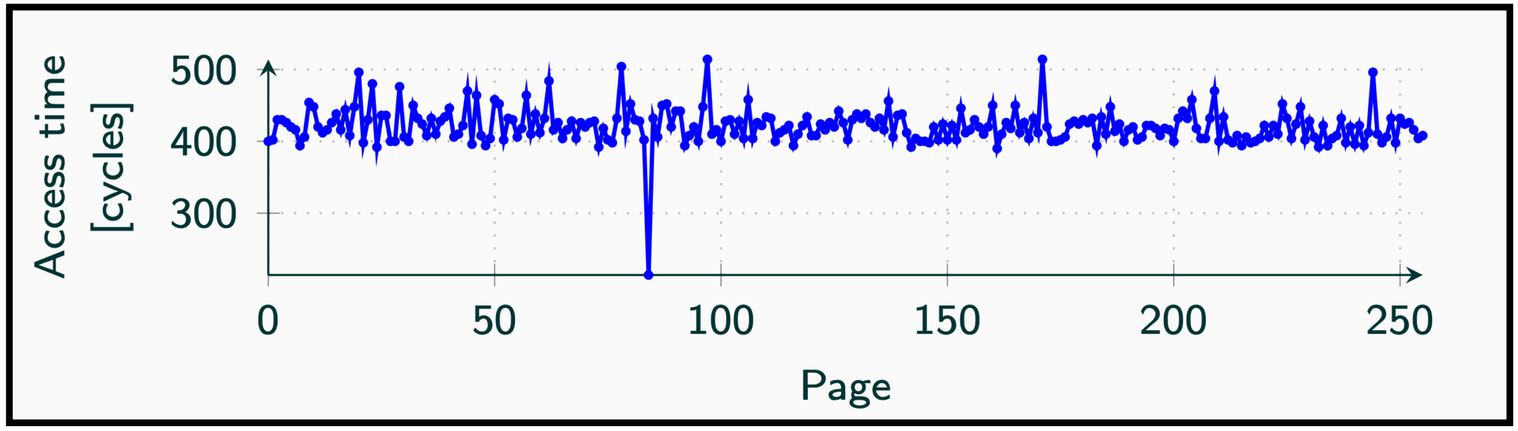
Does meltdown work even when I'm doing bad stuff?

- Following code dereferences a null ptr **(bad)** and then assigns something to an array

```
*(char *) 0;

array[0] = 0;
```

- Because of MMU privilege check, the first line will segfault the program, but…



- A cache timing side channel attack reveals that the array page is still cached!

Out of order execution is the building block of Meltdown

- Out of order instructions that have microarchitectural side effects are called **transient instructions**
  - **These are the building blocks of our attack**
- The previous example shows us that we run out-of-order instructions **even when the previous instructions might crash a program**
  - Key insight of Meltdown: Can we use side channels + this architectural “race condition” to extract protected memory?

Meltdown Playbook

Meltdown Playbook

1. Identify an address that stores some privileged value you care about (e.g., kernel address that stores a buffer for passwords)

Meltdown Playbook

1. Identify an address that stores some privileged value you care about (e.g., kernel address that stores a buffer for passwords)

2. Create a **transient instruction** that can load that privileged value in the cache

Meltdown Playbook

1. Identify an address that stores some privileged value you care about (e.g., kernel address that stores a buffer for passwords)

2. Create a **transient instruction** that can load that privileged value in the cache

3. *Use the privileged value* in some operation (usually in Meltdown, it's to index an array)

Meltdown Playbook

1. Identify an address that stores some privileged value you care about (e.g., kernel address that stores a buffer for passwords)

2. Create a **transient instruction** that can load that privileged value in the cache

3. *Use the privileged value* in some operation (usually in Meltdown, it's to index an array)

4. Check to see if the thing you *used* was cached (e.g., check to see if your array index was cached using a timing side channel attack); if it was, then you *know the value of the privileged information*

Meltdown Playbook

1. Identify an address that stores some privileged value you care about (e.g., kernel address that stores a buffer for passwords)

2. Create a **transient instruction** that can load that privileged value in the cache

3. *Use the privileged value* in some operation (usually in Meltdown, it's to index an array)

4. Check to see if the thing you *used* was cached (e.g., check to see if your array index was cached using a timing side channel attack); if it was, then you *know the value of the privileged information*

5. Repeat

Meltdown Playbook example

1. Goal: Read one byte of kernel memory
   at the location `0xffff00e0`

```
char data = *(char*)0xffff00e0

array[data] = 0;
```


Example adapted from Security & Privacy Academy

Meltdown Playbook example

- Goal: Read one byte of kernel memory at the location `0xffff00e0`
- Transient instruction loads this memory value into the cache
  - Remember, privilege check will be delayed

```
char data = *(char*)0xffff00e0

array[data] = 0;
```


Example adapted from Security & Privacy Academy

Meltdown Playbook example

- Goal: Read one byte of kernel memory at the location `0xffff00e0`
- Transient instruction loads this memory value into the cache
  - Remember, privilege check will be delayed
- Use the data in some operation, in this case, indexing an array.

```
char data = *(char*)0xffff00e0

array[data] = 0;
```


Example adapted from Security & Privacy Academy

Meltdown Playbook example

- Goal: Read one byte of kernel memory at the location `0xffff00e0`
- Transient instruction loads this memory value into the cache
  - Remember, privilege check will be delayed
- Use the data in some operation, in this case, indexing an array
- Run a timing side channel to check if array[data] is in the cache (flush + reload)

```
char data = *(char*)0xffff00e0

array[data] = 0;
```


Example adapted from Security & Privacy Academy

Meltdown Playbook example

- Goal: Read one byte of kernel memory at the location `0xffff00e0`
- Transient instruction loads this memory value into the cache
  - Remember, privilege check will be delayed
- Use the data in some operation, in this case, indexing an array
- Run a timing side channel to check if array[data] is in the cache (flush + reload)

```
char data = *(char*)0xffff00e0

array[data] = 0;
```


Example adapted from Security & Privacy Academy

Meltdown Playbook example

- Goal: Read one byte of kernel memory at the location `0xffff00e0`
- Transient instruction loads this memory value into the cache
  - Remember, privilege check will be delayed
- Use the data in some operation, in this case, indexing an array
- Run a timing side channel to check if array[data] is in the cache (flush + reload)

```
char data = *(char*)0xffff00e0

array[data] = 0;
```

If array[data] is in the cache, then you learn the *value* of data even though you never "read" it directly!

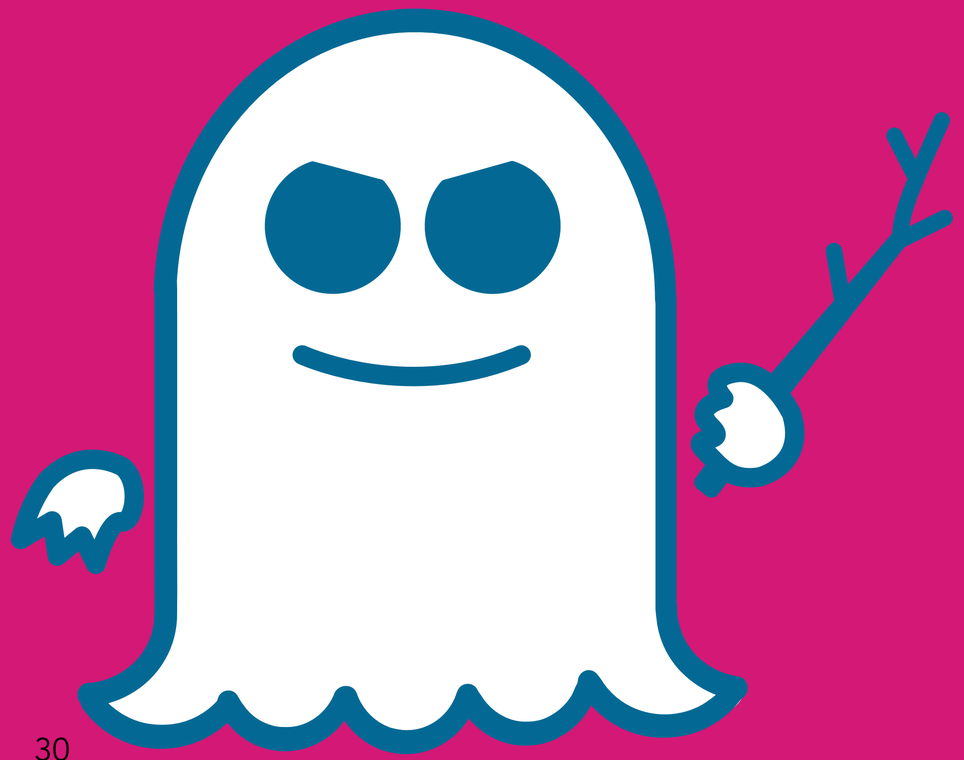
Example adapted from Security & Privacy Academy

Meltdown Bug

- Using out-of-order execution, attacker can read any data at any address
- Privilege checks for the kernel are sometimes too slow to stop this
- Kernel memory is leaked
- Entire physical memory is typically also accessible in kernel space… meaning you can potentially leak other processes private memory as well

What do we do about Meltdown?

- Immediate software only fix: stop mapping kernel memory pages in user processes altogether
  - But this would make programs so slow…
- Instead, we do something called **Kernel Page Table Isolation**
  - Key idea: Map kernel memory in userspace still, but **page everything out** unless you're running in kernel mode
  - 5% — 30% slowdown for most workloads
- Newer processors have a microarchitectural fix where the privilege check happens fast, but many are still vulnerable



Spectre

Processors being Processors: Speculative Execution

- Sometimes control flow depends on output of an earlier instruction
  - E.g., conditional branch, function pointer
- Rather than wait to know for sure which way to go, the processor may ***speculate*** about the direction/target of a branch
  - **Guess** based on the past
  - If guess is correct, performance is improved
  - If guess is wrong, speculated computation is discarded and everythign is re-computed using the correct value

Branch Prediction

- Branch prediction is a feature in CPUs that tries to predict pathways taken to conduct efficient speculative execution

Branch Prediction

- Branch prediction is a feature in CPUs that tries to predict pathways taken to conduct efficient speculative execution

```
for (i = 0; i < 500; i++){
    array[i] = 'h'
}
array2[0] = 0;
```

Branch Prediction

- Branch prediction is a feature in CPUs that tries to predict pathways taken to conduct efficient speculative execution

```
for (i = 0; i < 500; i++){
    array[i] = 'h'
}
array2[0] = 0;
```

- Branch predictor would be pretty damn sure `array[i]` is going to be used; speculatively execute that line instead of the next one

Spectre — it gets worse

- Combines cache side channels + branch prediction for arbitrary memory reading

- Can break the **process invariant —** e.g., malicious processes can trick the CPU into loading *other process memory* into the cache, and leaks that memory voluntarily

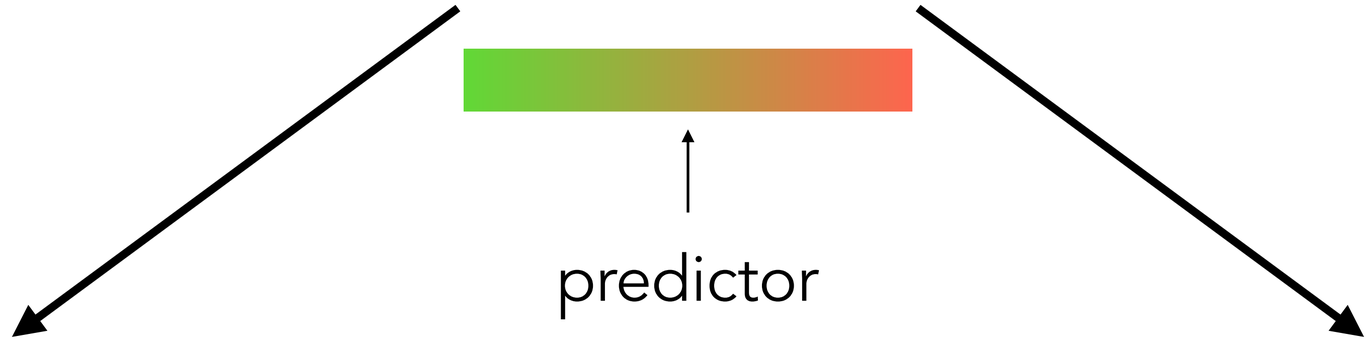
Spectre attacks (simplified)

- Goal: Train branch predictor to leak memory values

```
index = 0;

char* data = "textKEY";

if (index < 4)
```




```
arr[data[index] * 4096]                    0
```

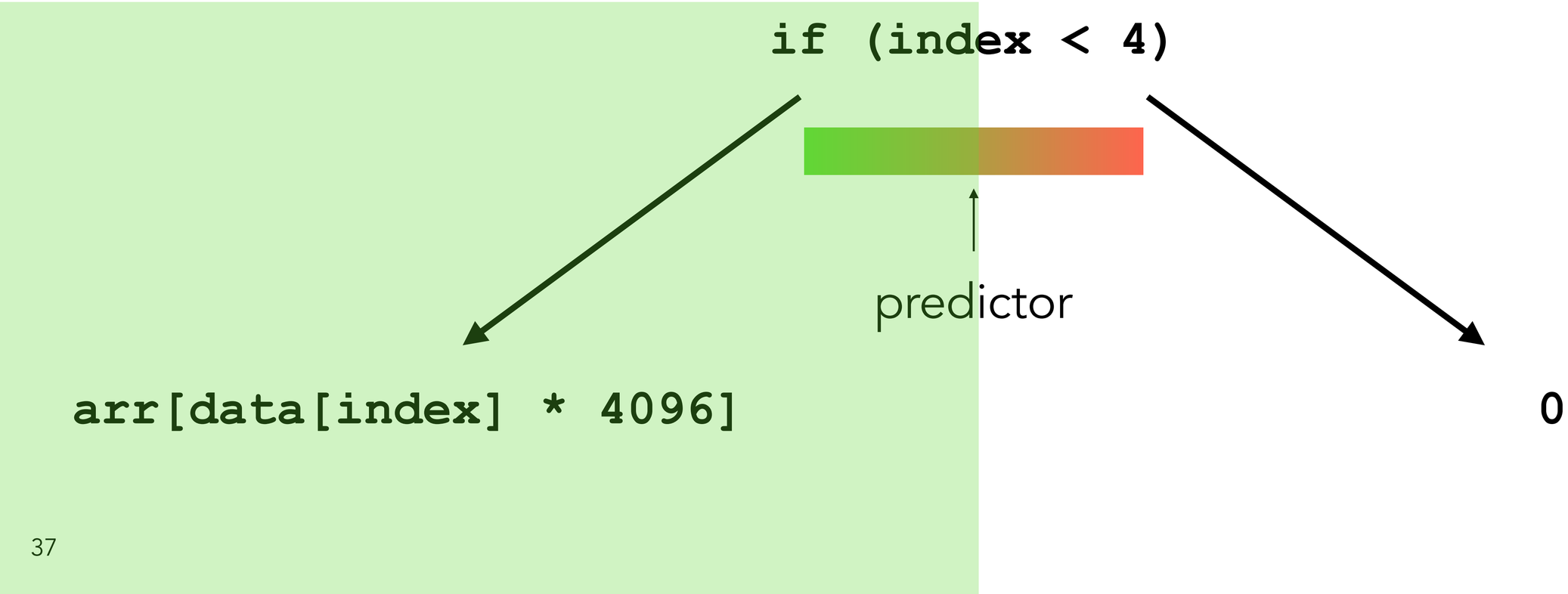
Spectre attacks (simplified)

- Goal: Train branch predictor to leak memory values

```
index = 0;

char* data = "textKEY";

if (index < 4)
```



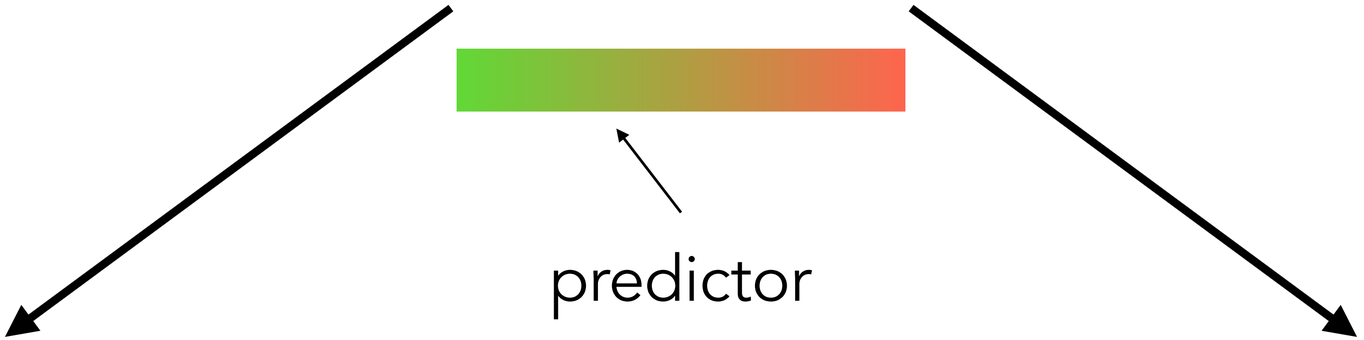
Spectre attacks (simplified)

- Goal: Train branch predictor to leak memory values

```
index = 0;

char* data = "textKEY";

if (index < 4)
```




```
arr[data[index] * 4096]                    0
```

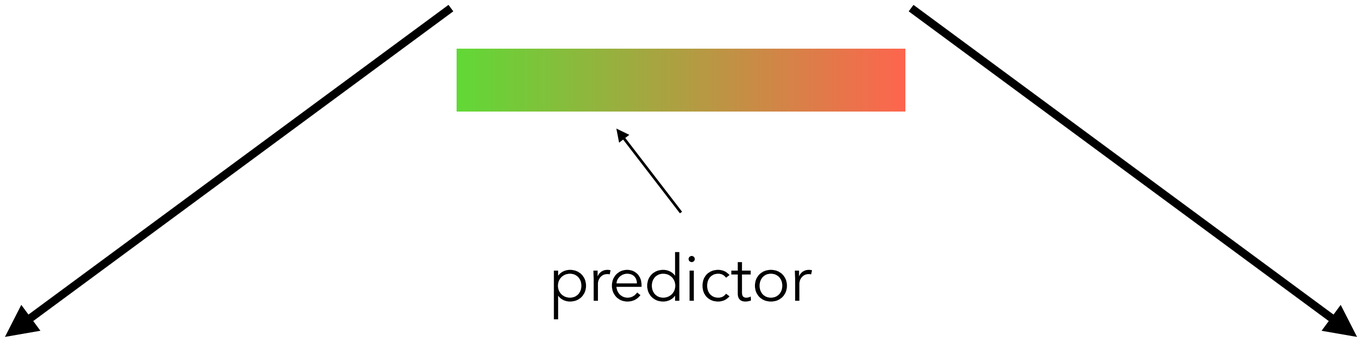
Spectre attacks (simplified)

- Goal: Train branch predictor to leak memory values

```
index = 1;

char* data = "textKEY";

if (index < 4)
```




```
arr[data[index] * 4096]                    0
```

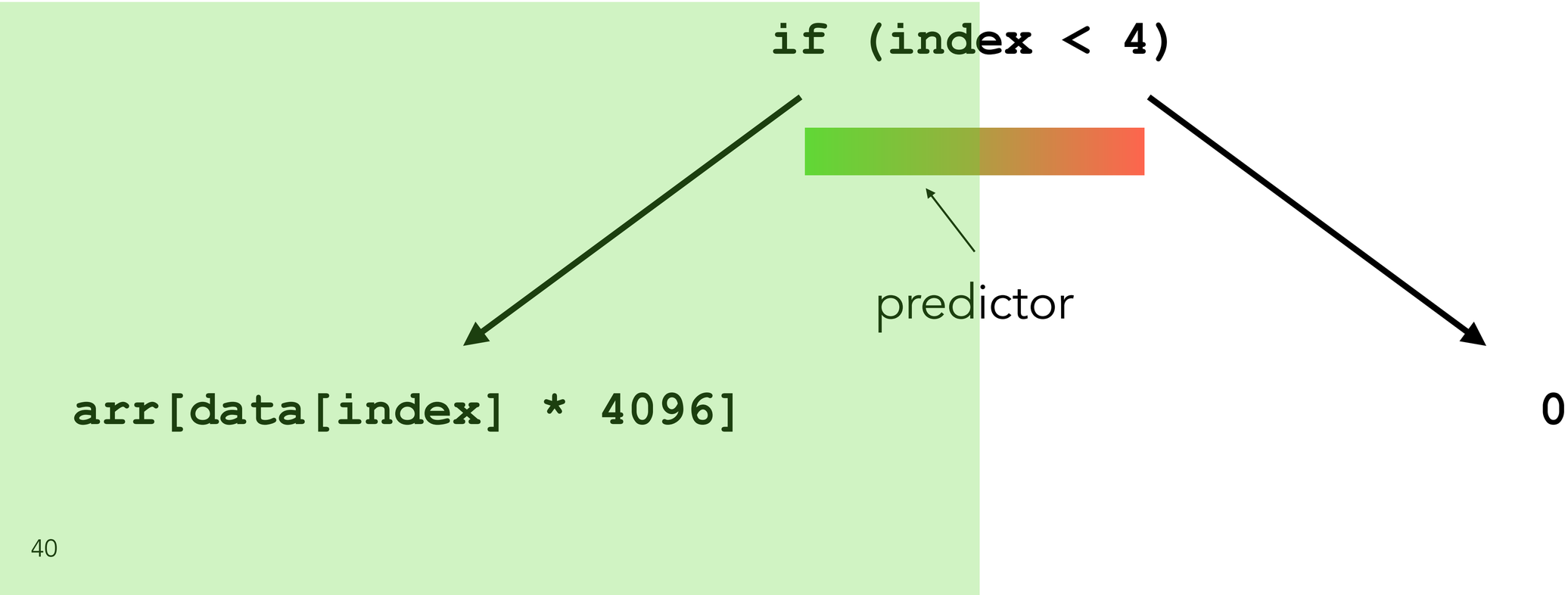
Spectre attacks (simplified)

- Goal: Train branch predictor to leak memory values

```
index = 1;

char* data = "textKEY";

if (index < 4)
```



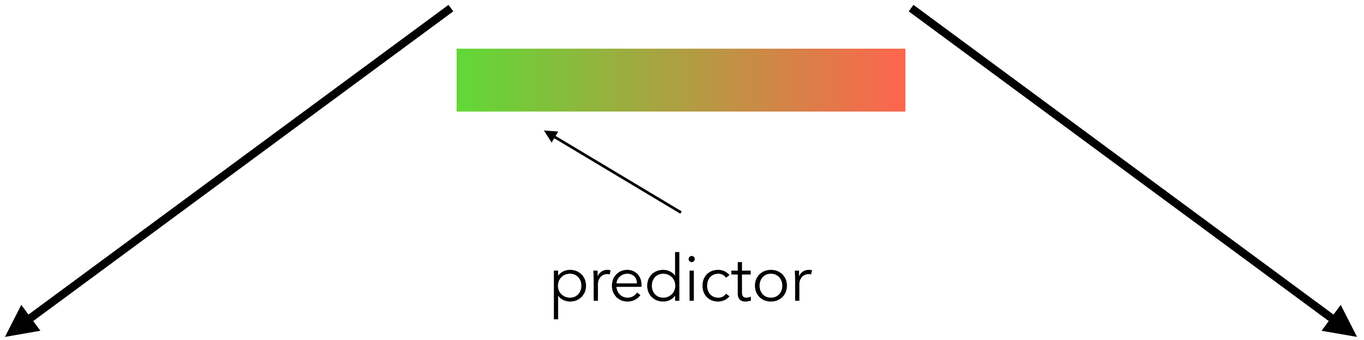
Spectre attacks (simplified)

- Goal: Train branch predictor to leak memory values

```
index = 2;

char* data = "textKEY";

if (index < 4)
```




```
arr[data[index] * 4096]                    0
```

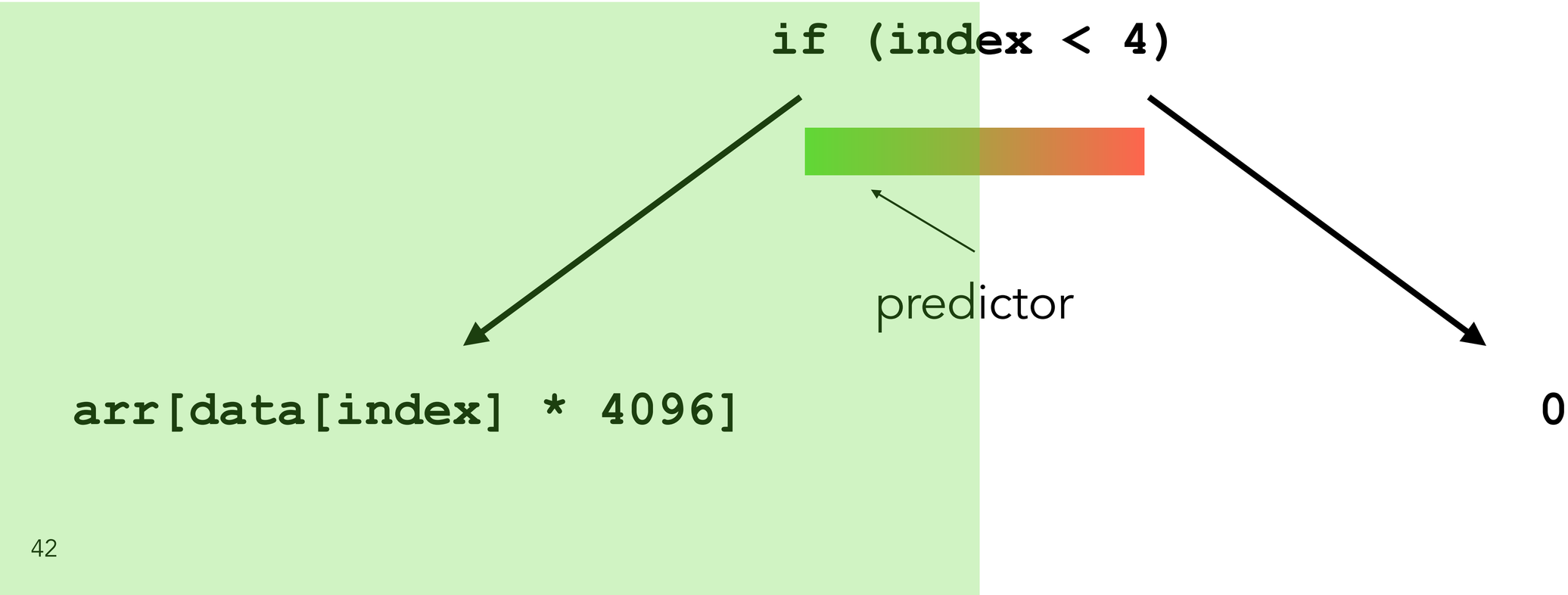
Spectre attacks (simplified)

- Goal: Train branch predictor to leak memory values

```
index = 2;

char* data = "textKEY";

if (index < 4)
```



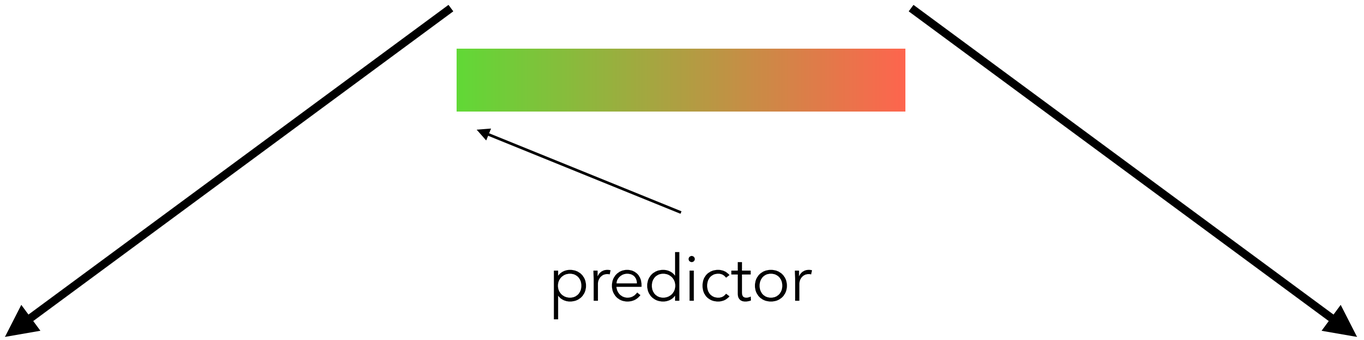
Spectre attacks (simplified)

- Goal: Train branch predictor to leak memory values

```
index = 3;

char* data = "textKEY";

if (index < 4)
```




```
arr[data[index] * 4096]                    0
```

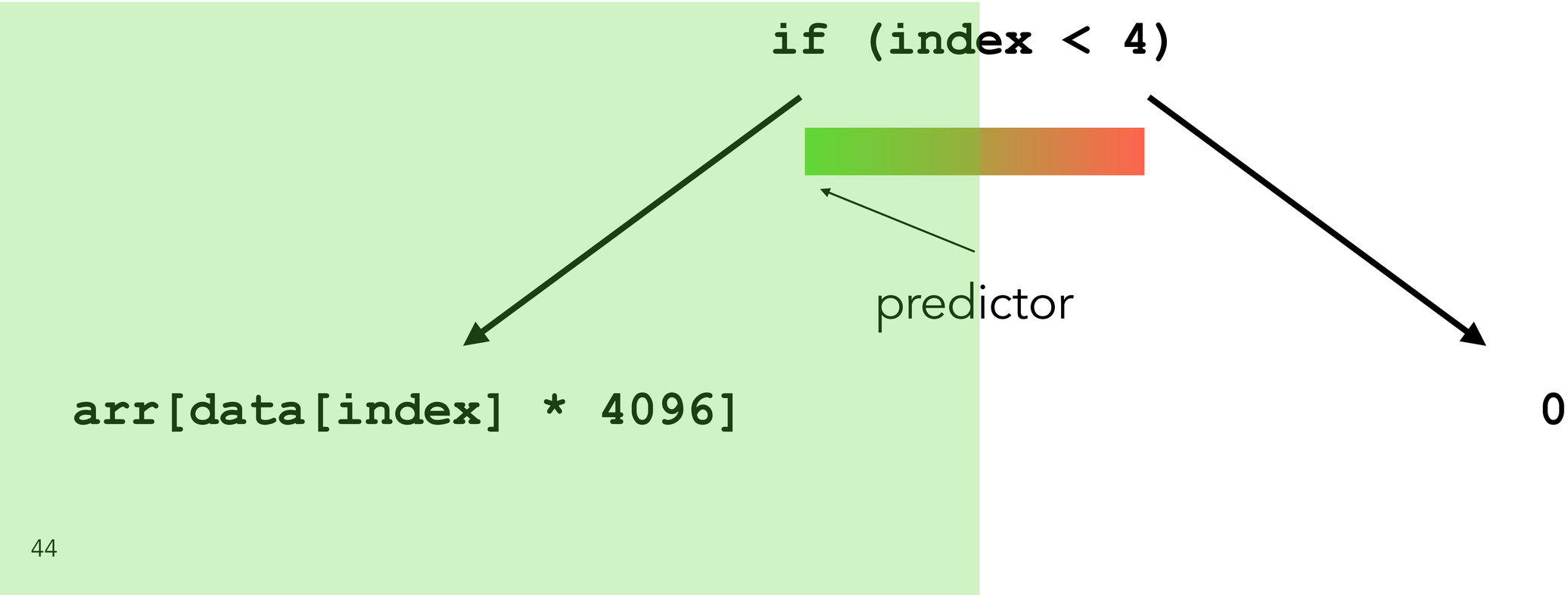
Spectre attacks (simplified)

- Goal: Train branch predictor to leak memory values

```
index = 3;

char* data = "textKEY";

if (index < 4)
```



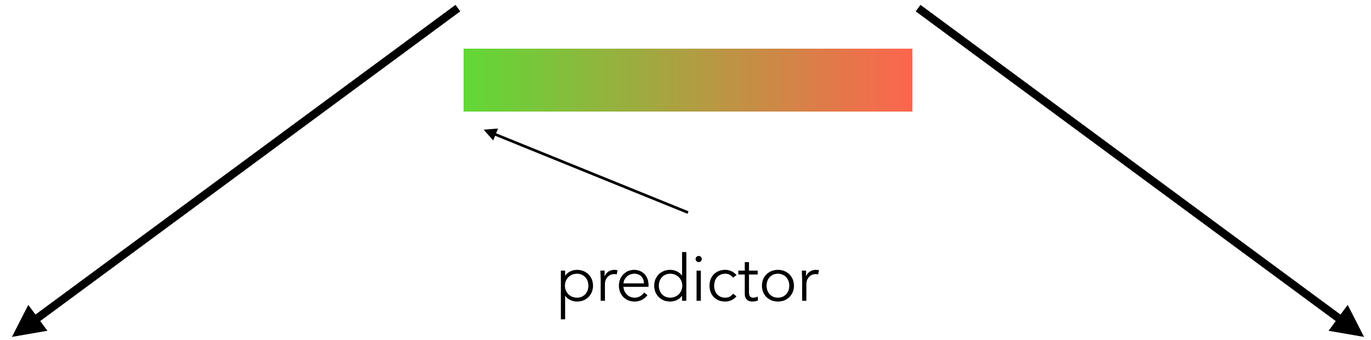
Spectre attacks (simplified)

- Goal: Train branch predictor to leak memory values

```
index = 4;

char* data = "textKEY";

if (index < 4)
```




```
arr[data[index] * 4096]                    0
```

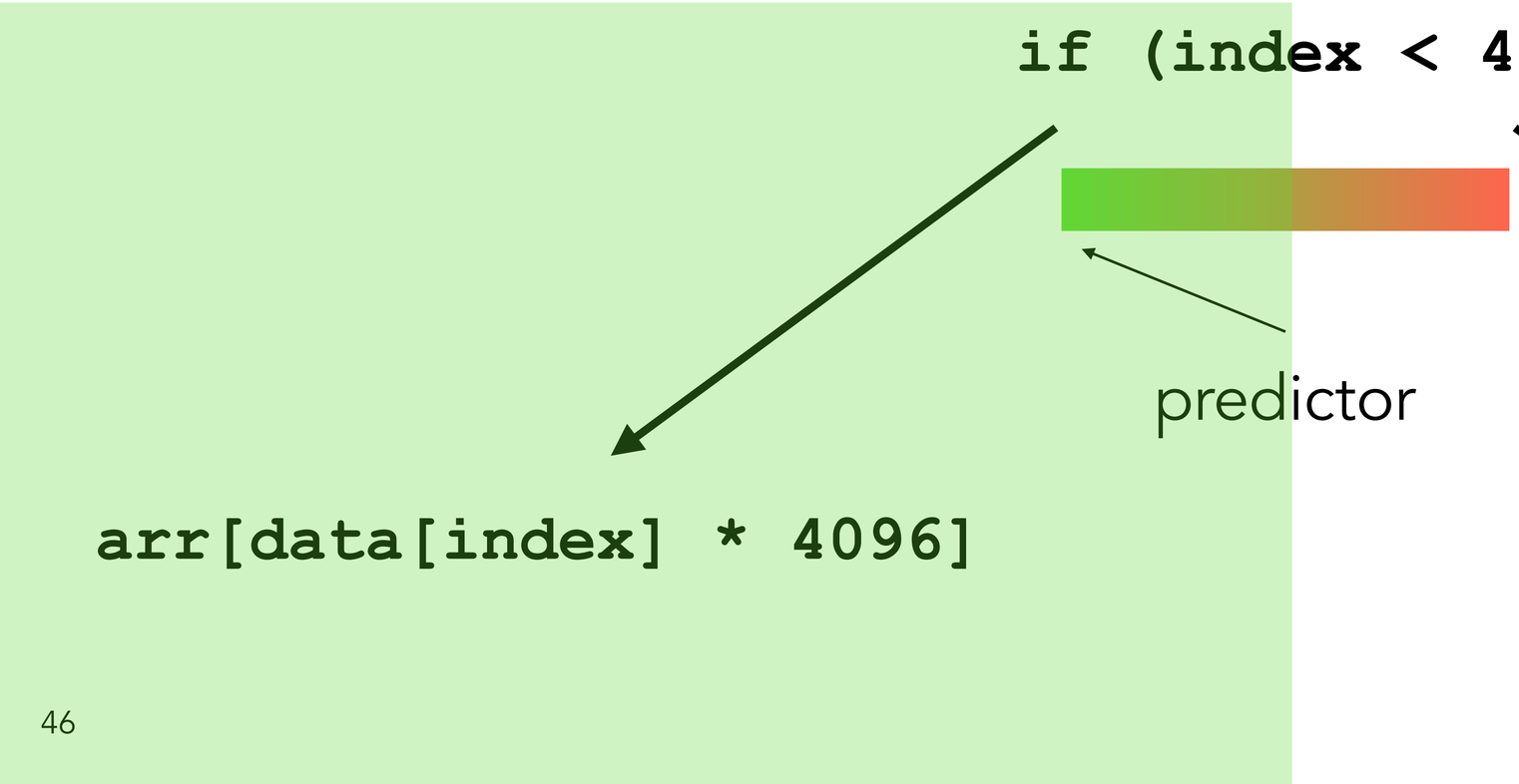
Spectre attacks (simplified)

- Goal: Train branch predictor to leak memory values

```
index = 4;

char* data = "textKEY";

if (index < 4)
```



Branch prediction will erroneously place "K" in the cache…

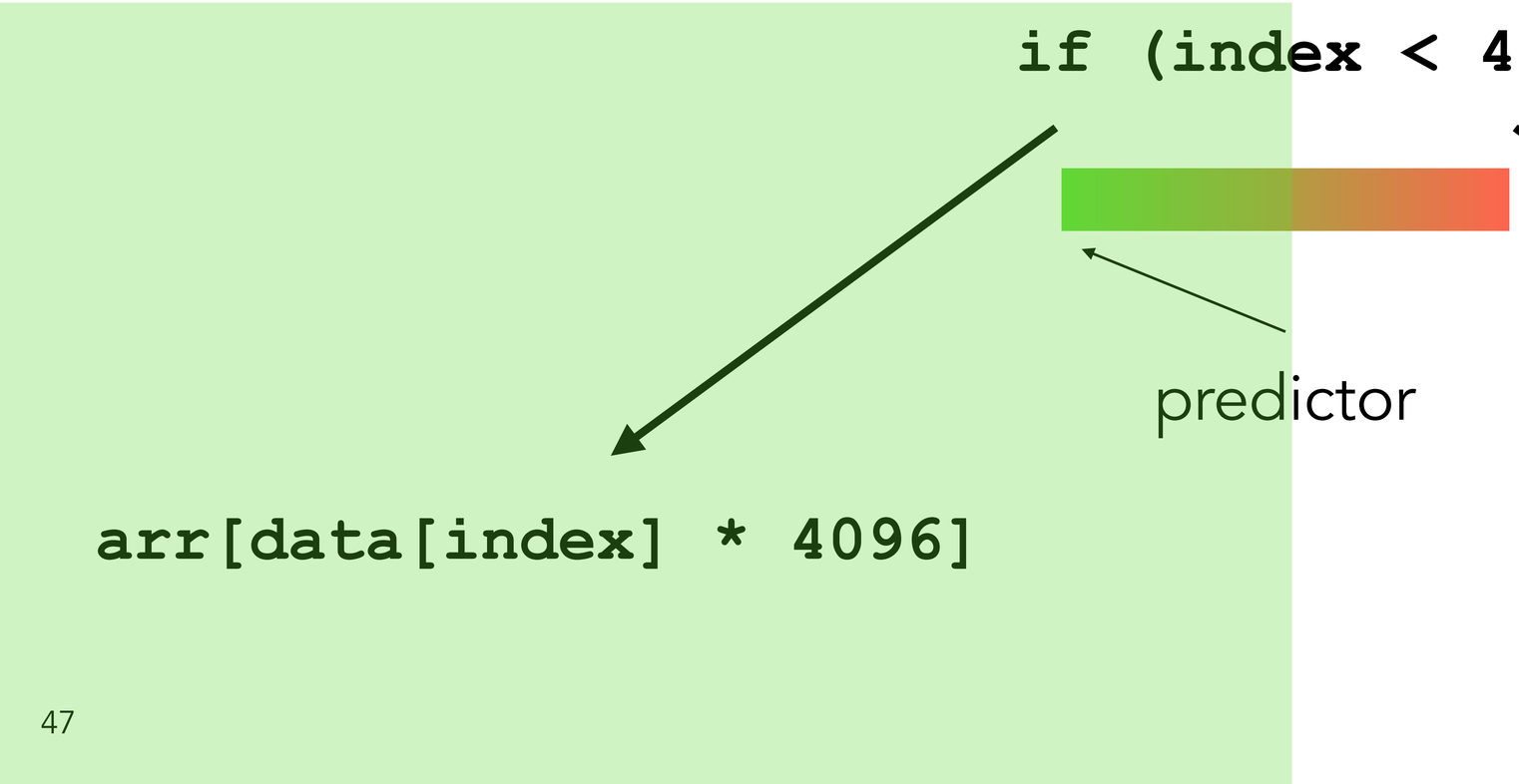
Spectre attacks (simplified)

- Goal: Train branch predictor to leak memory values

```
index = 4;

char* data = "textKEY";

if (index < 4)
```



Branch prediction will erroneously place "K" in the cache…

You can extract just like you would in Meltdown!

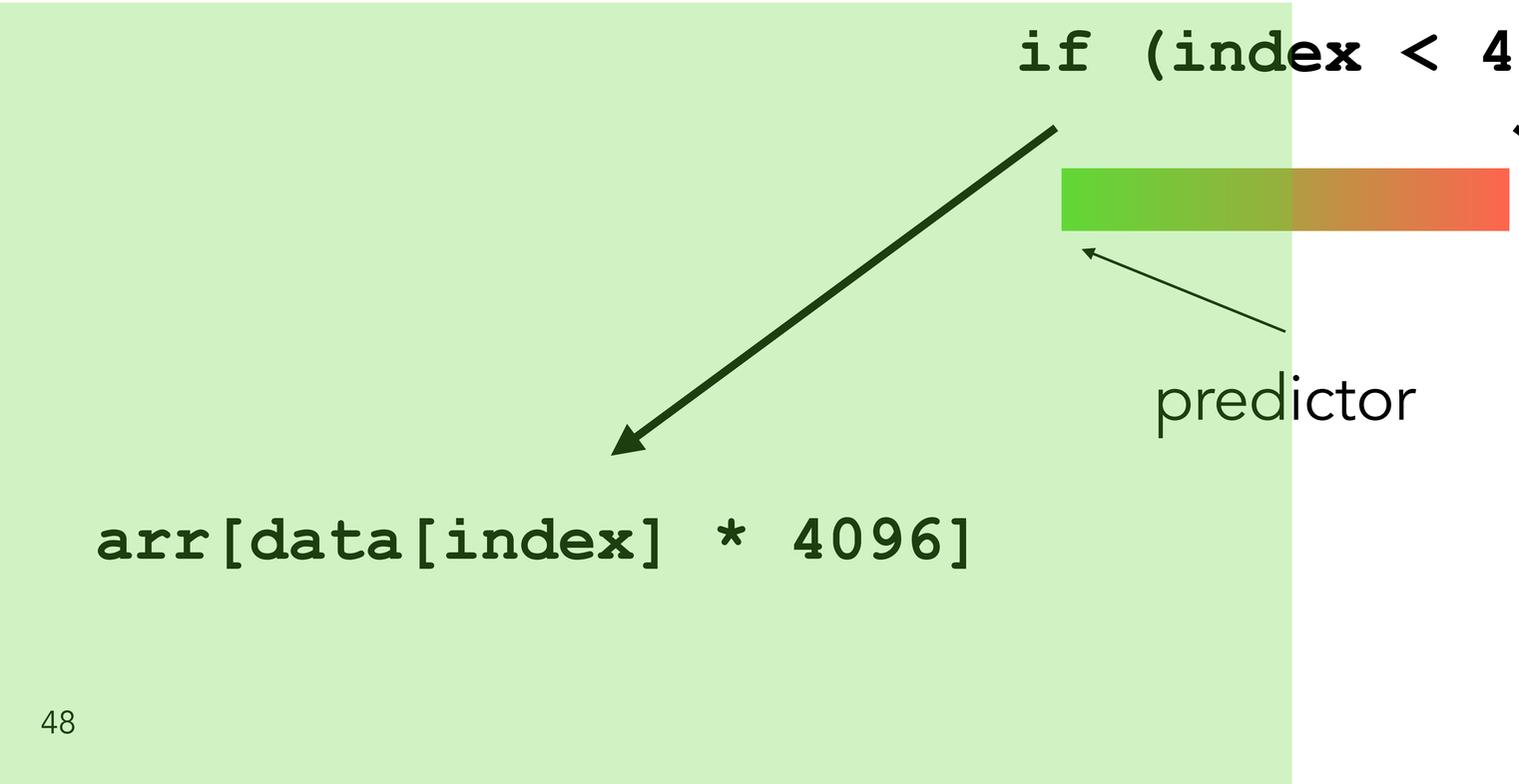
Spectre attacks (simplified)

- Goal: Train branch predictor to leak memory values

```
index = 4;

char* data = "textKEY";

if (index < 4)
```



Branch prediction will erroneously place "K" in the cache…

You can extract just like you would in Meltdown!

game over gg

What do we do about Spectre?

- Ideas?

What do we do about Spectre?

- Ideas?
- Admittedly very rough; but good news is it's hard to exploit
  - Broad issue about speculation and branch prediction
- Could disable speculation on branches… but htere's a huge performance impact as a result
  - Also can only be done by chip manufacturer
- Selectively insert instructions to stop speculation at sensitive branches
  - LFENCE
- All of these are bandaids, not solutions

Sea-change shift in last 8 years

- Computer architects spent the last 20 years optimizing for common use case
  - Assumption was that if optimization doesn't change output then we're all good
- Sadly, every optimization is now under intense, immense scrutiny
  - New microarchitectual side-channel papers are published almost every conference…
- Hardware vendors and computer architects are retooling to figure out how to still offer optimizations without huge security hit
  - ¯_(ツ)_/¯ it's an ongoing field

My 2c on side channel attacks

- Side channels are epicly cool and fun
  - They entertain the part of your brain that likes puzzles
- Are they the most important harm / security threat facing people today?
  - No. They’re niche, hard to execute, and often probabilistic in practice
- But they’re worth studying… even if to teach us all the ways in which our assumptions might be flawed :)

The Web

Where we've been and where we're going

- So far, we've been talking **low level,** asking questions about C programs, buffers, operating systems, and processors…
  - A lot of cybersecurity action still remains here, the most traditional "exploits" of computer systems live here
- Now, we're shifting our focus to higher level systems and those security challenges
  - The web
  - Networks
  - Cryptography (and TLS which protects us all)
  - People

Learning objectives

- Understand (basically) how Web browsing works and the fundamental architecture that underpins the Web
- Discuss cookies, tracking, and how **state** is applied on the web
- Discuss JavaScript and the mechanisms that allow websites to operate as programs

Learning objectives

- Understand (basically) how Web browsing works and the fundamental architecture that underpins the Web
- Understand the basic Web security model (same origin policy)
- Learn what a cookie is and the ways in which cookies can get attacked

Polling the room

- How many people have built a website before?
- How many people have built a web app before?
- How many people have *deployed* a web app before?
  - Where?

What is the web?

What is the web?

Information system that runs on the Internet that allows *documents* to be connected to other *documents*, increasingly enabled through *scripting* and *server-side logic*

Web Fundamentals

- Take three minutes to answer these questions with folks around you.
  - What is a web server? Examples?
  - What is a web client? Examples?
  - What is HTTP?

Web Fundamentals

- Take three minutes to answer these questions with folks around you.
  - **What is a web server? Examples?**
  - **What is a web client? Examples?**
  - What is HTTP?

Interfacing with the Web

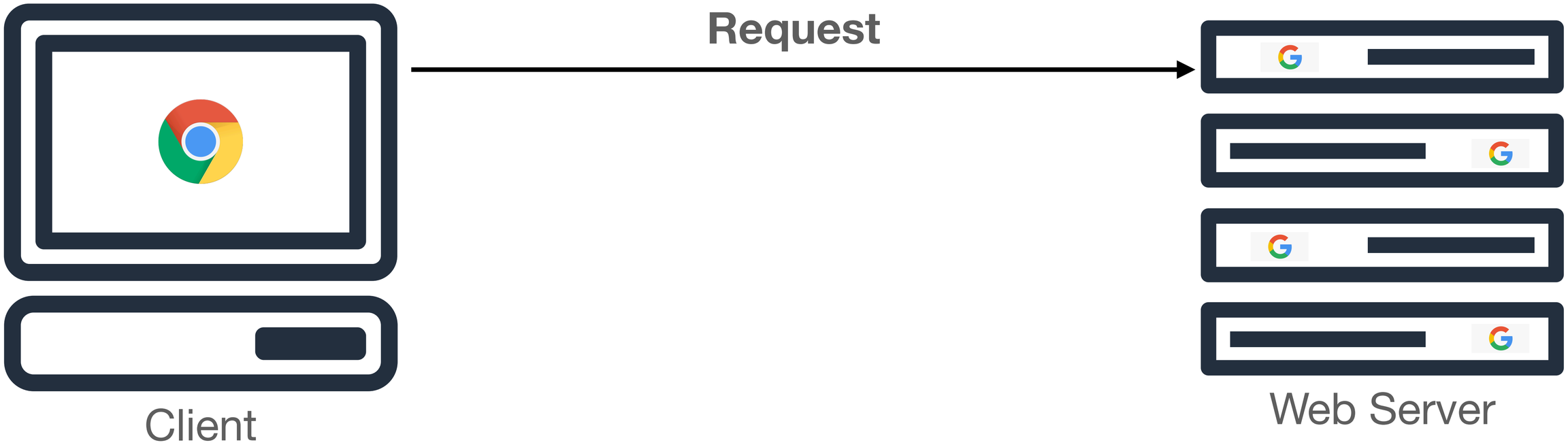
Client / Server Model

Client

Web Server

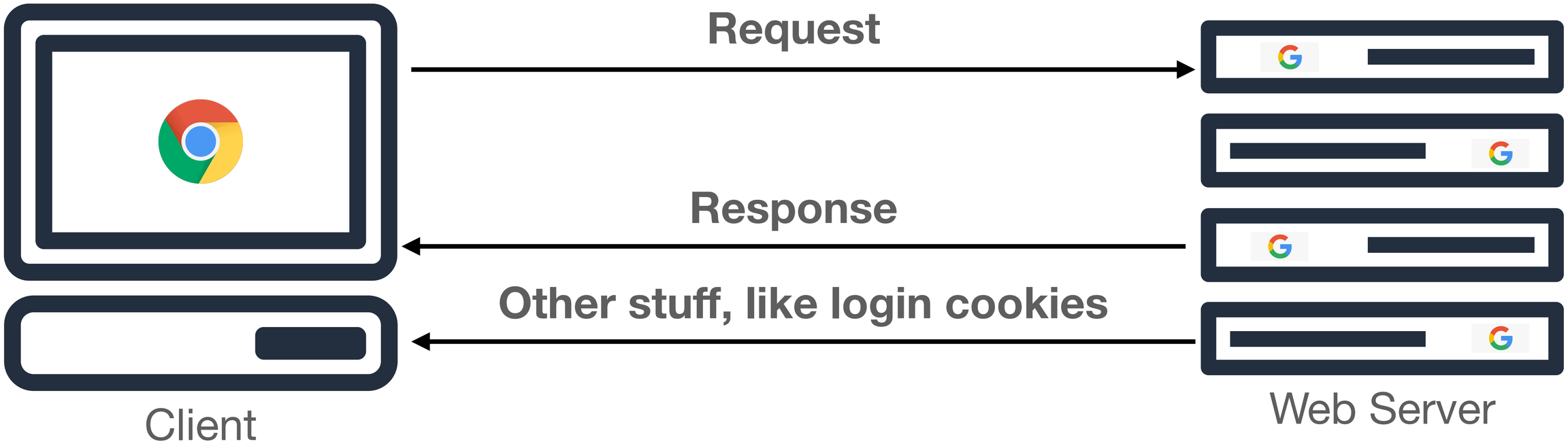
Interfacing with the Web

Client / Server Model



Interfacing with the Web

Client / Server Model



Interfacing with the Web

Client / Server Model

Client

Reques

Response

Other stuff, like login cookies

Web Server

Clients can be all kinds of things!

– Web browsers
– Phone Apps
– Desktop Apps

Modern web architecture

- Web client (typically a browser) issues requests
- Web server responds
- Web client will handle the response for the user

Web clients

- How do web browsers make requests to web servers?

Web clients

- How do web browsers make requests to web servers?
  - User types in a URL into the browser bar
  - User re-loads a page
  - User clicks on a link
  - Web server responds with a redirect (aka, browser should request a new page)
  - Web page could embed another page (independent of user interaction)
  - Script within a webpage can issue a request

Web servers

- What are the ways in which web servers can respond to client requests?

Web servers

- What are the ways in which web servers can respond to client requests?
  - Return a document (HTML) for the page
  - Return any other resource (e.g., images, scripts, CSS, etc.)
  - Run some custom logic or code and return outputs (e.g., APIs)
  - Talk to some external system and return output (e.g., database)
  - Do nothing and waste your time

Web client response handling

- What are some ways that web clients handle server responses?

Web client response handling

- What are some ways that web clients handle server responses?
  - Render HTML + CSS
  - Execute embedded JavaScript
  - Invoke some external plugin (e.g., PDF reader)

Web Fundamentals

- Take three minutes to answer these questions with folks around you.
  - What is a web server? Examples?
  - What is a web client? Examples?
  - **What is HTTP?**

Hypertext Transfer Protocol (HTTP)

- HTTP — a protocol invented in 1989 that still power the web today
  - Basic concept is to allow fetching of resources (e.g., HTML documents, nowadays, basically anything)
- How do we identify where a resource is on the web?

Hypertext Transfer Protocol (HTTP)

- HTTP — a protocol invented in 1989 that still power the web today
  - Basic concept is to allow fetching of resources (e.g., HTML documents, nowadays, basically anything)
- How do we identify where a resource is on the web?
  - We use what's called a Uniform Resource Locator (URL)

URLs demystified

- All resources on the web have a URL, which is broken into several parts:

`https://www.kumarde.com:443/classes/wi26/cse127/lectures/web_lec.pdf?lang=en`

URLs demystified

- All resources on the web have a URL, which is broken into several parts:
  - Scheme: Tells you what protocol you're using to communicate. On web, typically HTTP or HTTPS.

`https://www.kumarde.com:443/classes/wi26/cse127/lectures/web_lec.pdf?lang=en`

URLs demystified

- All resources on the web have a URL, which is broken into several parts:
  - Domain: The human-readable name that describes the server you're trying to talk to. kumarde.com and www.kumarde.com are *different domains.*

`https://www.kumarde.com:443/classes/wi26/cse127/lectures/web_lec.pdf?lang=en`

URLs demystified

- All resources on the web have a URL, which is broken into several parts:
  - Port: The network port to communicate with the server. HTTP default runs on 80, HTTPS on 443. Lots of default ports (22 for SSH, 23 for Telnet, etc.)

`https://www.kumarde.com:443/classes/wi26/cse127/lectures/web_lec.pdf?lang=en`

URLs demystified

- All resources on the web have a URL, which is broken into several parts:
  - Path: The full “file path” to the resource you’re looking for. Mirrors file path structure on most servers. APIs leverage paths to designate resources.

`https://www.kumarde.com:443/classes/wi26/cse127/lectures/web_lec.pdf?lang=en`

URLs demystified

- All resources on the web have a URL, which is broken into several parts:
  - Query string: Typically includes parameters delimited by “&” that contain useful information, e.g., language, etc.

`https://www.kumarde.com:443/classes/wi26/cse127/lectures/web_lec.pdf?lang=en`

HTTP is request-response

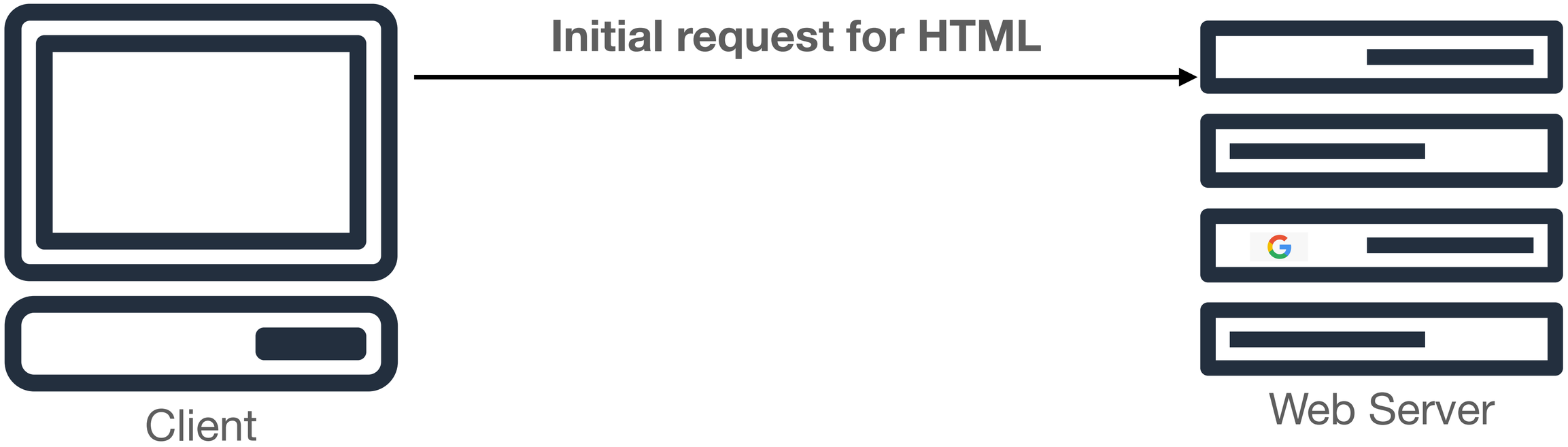
- Unlike other network protocols, HTTP is strictly **request-response**; meaning clients and servers exchange individual messages (as opposed to a stream of data)

Client

Web Server

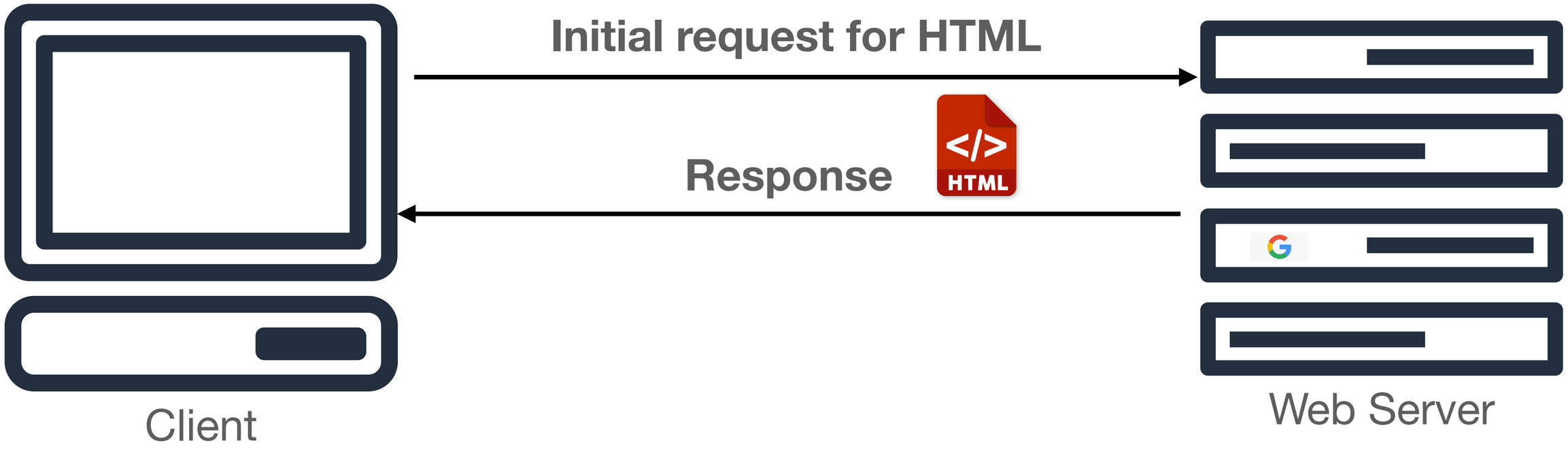
HTTP is request-response

- Unlike other network protocols, HTTP is strictly **request-response**; meaning clients and servers exchange individual messages (as opposed to a stream of data)



HTTP is request-response

- Unlike other network protocols, HTTP is strictly **request-response**; meaning clients and servers exchange individual messages (as opposed to a stream of data)



HTTP is request-response

- Unlike other network protocols, HTTP is strictly **request-response**; meaning clients and servers exchange individual messages (as opposed to a stream of data)

HTTP is request-response

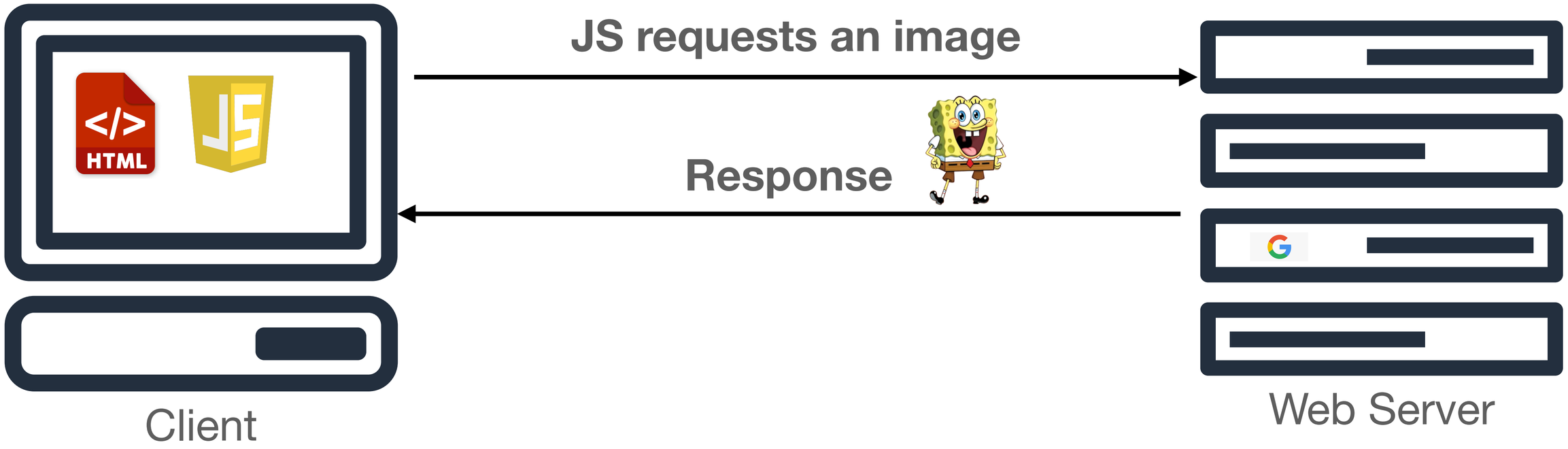
- Unlike other network protocols, HTTP is strictly **request-response**; meaning clients and servers exchange individual messages (as opposed to a stream of data)

HTTP is request-response

- Unlike other network protocols, HTTP is strictly **request-response**; meaning clients and servers exchange individual messages (as opposed to a stream of data)

HTTP is request-response

- Unlike other network protocols, HTTP is strictly **request-response**; meaning clients and servers exchange individual messages (as opposed to a stream of data)



HTTP is request-response

- Unlike other network protocols, HTTP is strictly **request-response**; meaning clients and servers exchange individual messages (as opposed to a stream of data)

HTTP Request Anatomy

- What does an HTTP Request look like?

```
GET /index.html HTTP/1.1

Accept: image/gif, image/x-bitmap, image/jpeg, */*
Accept-Language: en
Connection: Keep-Alive
User-Agent: Mozilla/1.22 (compatible; MSIE 2.0; Windows 95)
Host: www.example.com
Referer: http://www.google.com?q=dingbats
```

HTTP Request Anatomy

- What does an HTTP Request look like?

Method Path Protocol Version

```
GET /index.html HTTP/1.1

Accept: image/gif, image/x-bitmap, image/jpeg, */*
Accept-Language: en
Connection: Keep-Alive
User-Agent: Mozilla/1.22 (compatible; MSIE 2.0; Windows 95)
Host: www.example.com
Referer: http://www.google.com?q=dingbats
```

HTTP Request Anatomy

- What does an HTTP Request look like?

```
GET /index.html HTTP/1.1

Accept: image/gif, image/x-bitmap, image/jpeg, */*
Accept-Language: en
Connection: Keep-Alive
User-Agent: Mozilla/1.22 (compatible; MSIE 2.0; Windows 95)
Host: www.example.com
Referer: http://www.google.com?q=dingbats
```

HTTP Headers

HTTP Request Anatomy

- What does an HTTP Request look like?

```
GET /index.html HTTP/1.1

Accept: image/gif, image/x-bitmap, image/jpeg, */*
Accept-Language: en
Connection: Keep-Alive
User-Agent: Mozilla/1.22 (compatible; MSIE 2.0; Windows 95)
Host: www.example.com
Referer: http://www.google.com?q=dingbats
```

Body (usually empty for requests)

Aside: HTTP Versions

- Today's servers generally support a mix of HTTP 1.1, 2, and 3
  - Differences are *mainly* about efficiency; solving head-of-line blocking problems, relying on different transport protocols, etc.
  - Modern browsers can speak everything
- HTTP/2 used by 34.1% of all websites (per w3techs)… actually **declining over time**
  - Pipelining requests for multiple objects, header compression, server push
- HTTP/3 used by 37% of all websites (per w3techs)
  - Uses QUIC instead of TCP

HTTP Response Anatomy

- What does an HTTP Response look like?

```
HTTP/1.0 200 OK
Date: Sun, 21 Apr 1996 02:20:42 GMT
Server: Microsoft-Internet-Information-Server/5.0
Connection: keep-alive
Content-Type: text/html
Last-Modified: Thu, 18 Apr 1996 17:39:05 GMT
Set-Cookie: ...
Content-Length: 2543
```

HTTP Response Anatomy

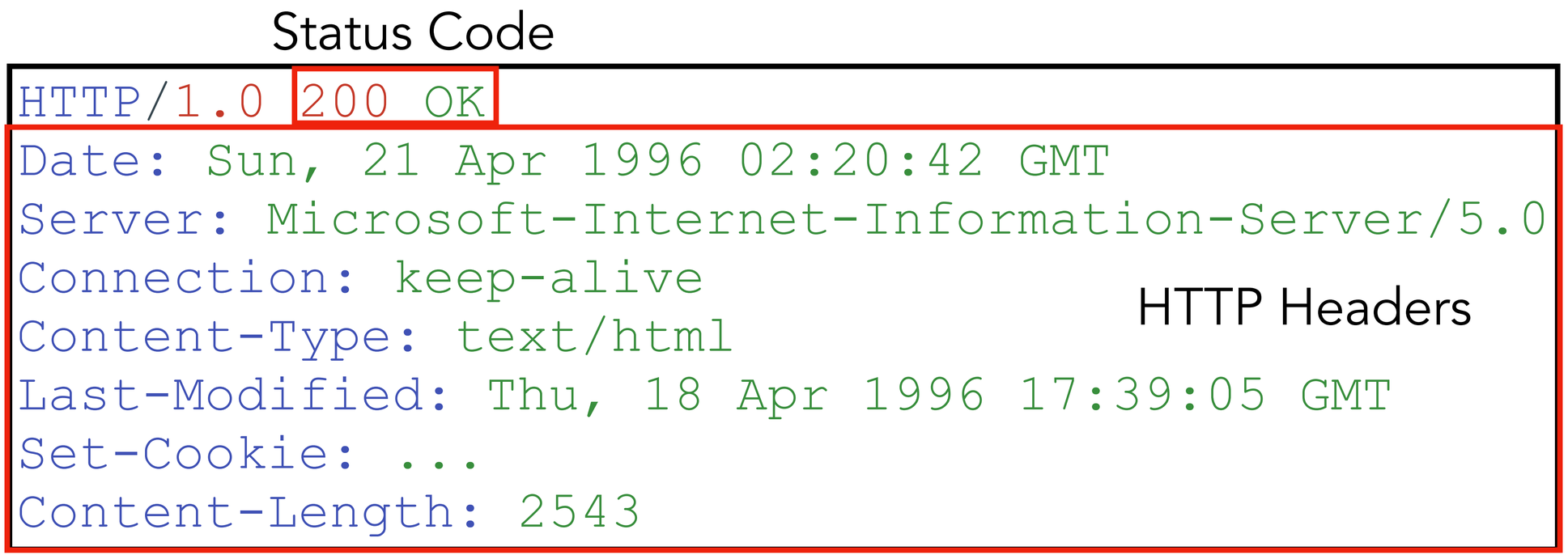
- What does an HTTP Response look like?

Status Code

```
HTTP/1.0 200 OK
Date: Sun, 21 Apr 1996 02:20:42 GMT
Server: Microsoft-Internet-Information-Server/5.0
Connection: keep-alive
Content-Type: text/html
Last-Modified: Thu, 18 Apr 1996 17:39:05 GMT
Set-Cookie: ...
Content-Length: 2543
```

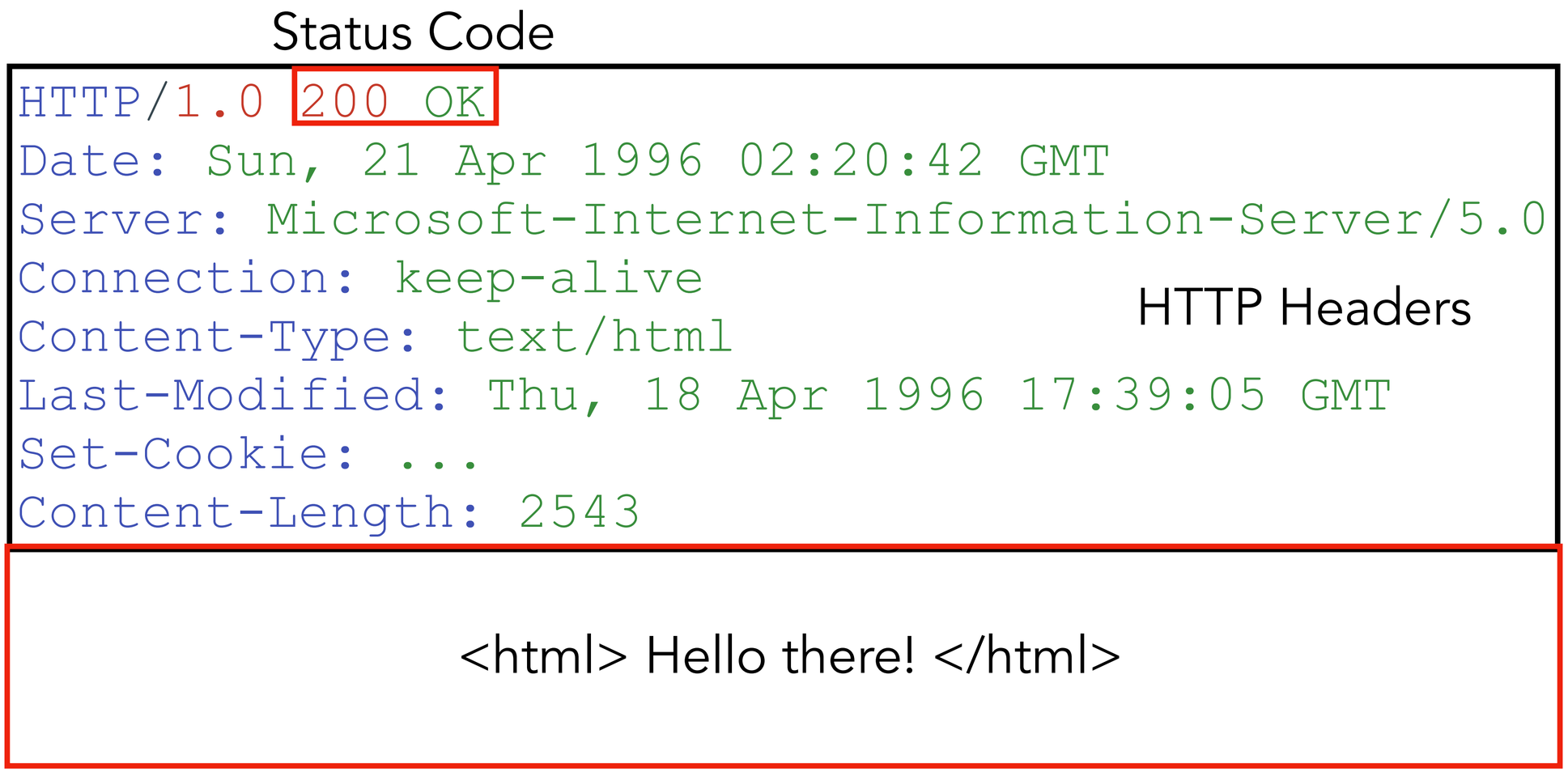
HTTP Response Anatomy

- What does an HTTP Response look like?



HTTP Response Anatomy

- What does an HTTP Response look like?

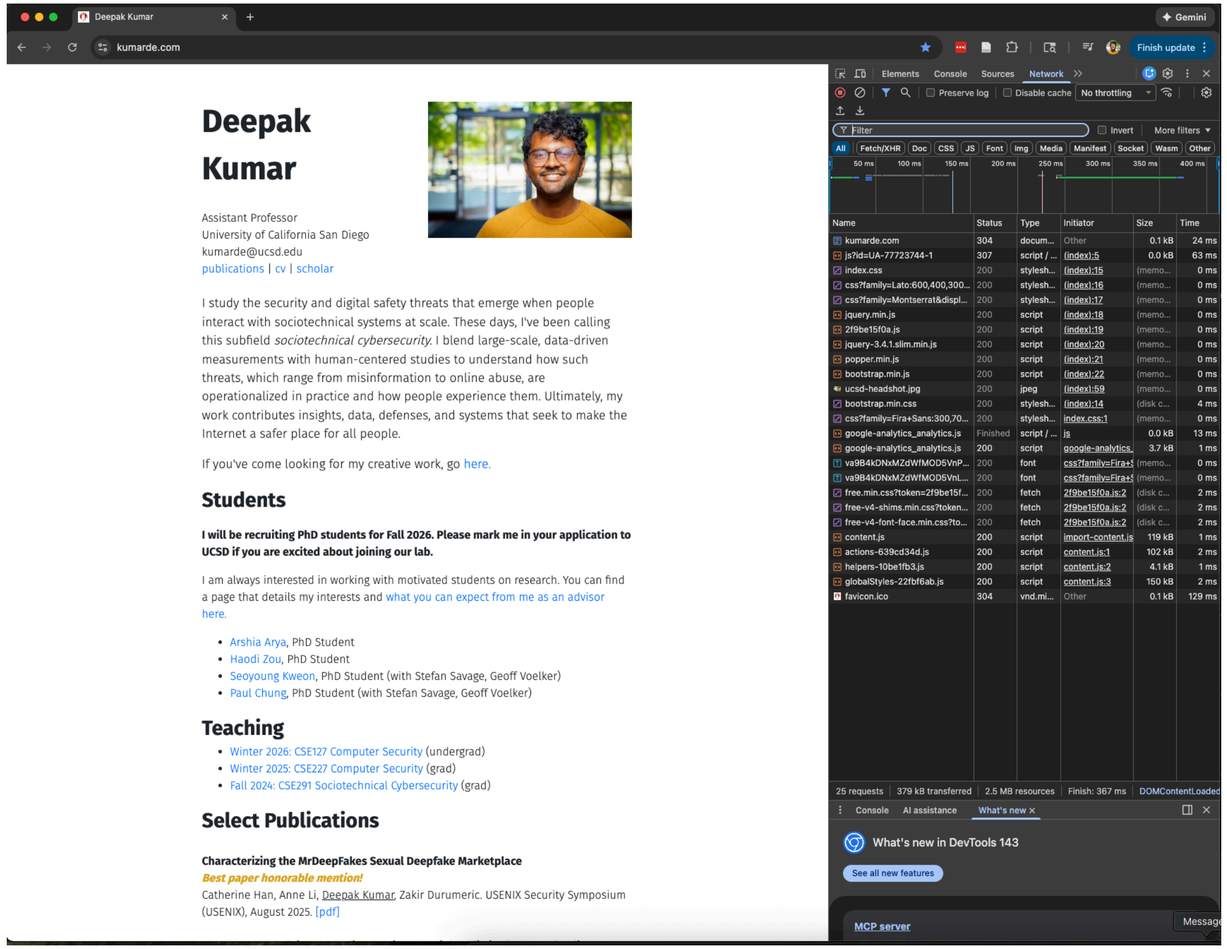


HTTP Basics

- Client sends `HTTP` requests, each come with a **method**
  - `GET`: retrieve a resource
  - `POST`: update a resource (submit a form, publish a post, etc.)
  - Anyone have a guess as to what `HEAD` does?
  - There are a few others (`PUT`, `PATCH`, `DELETE`); older browsers don't use them
- Servers send `HTTP` responses, each come with a **status code**
  - `200: OK`
  - `302: Found` (redirect)
  - `404: Not found`

HTTP Basics

- Websites have lots of resources: you should “Inspect Element” in your browser to see them



Web sessions

- HTTP is a **stateless protocol.** What does this mean?
  - Requests are processed independently; no notion of a *session* by default
- But…. most web applications are session-based, so how?

Web sessions

- HTTP is a **stateless protocol.** What does this mean?
  - Requests are processed independently; no notion of a *session* by default
- But…. most web applications are session-based, so how?
- Cookies! 

Web sessions

- HTTP is a **stateless protocol.** What does this mean?
  - Requests are processed independently; no notion of a *session* by default
- But…. most web applications are session-based, so how?
- Cookies! 
- Cookies are used for lots of things, good and bad, including
  - Sessions (e.g., login, shopping carts)
  - Personalization (e.g., user preferences, themes, etc.)
  - Tracking (e.g., online advertising)

How do web cookies work?

- The web server sets cookies in a response to the web browser (using an HTTP header)
  - `Set-Cookie: <cookie-name>=<cookie-value>;`
  - Also defines “properties” for each cookie
    - e.g., when they expire, what domains they’re for, what protocols to use with, etc.
- Browser automatically attaches cookies to **every subsequent request to that web server**
  - Unless you disable them, but… you’ll be annoyed

Web Cookie Typology

- Session cookies
  - Expiration property is not set
  - Exist *only* during current browser session
  - Deleted when your browser is shut down
  - Examples?
- Persistent cookies
  - Saved until server-defined expiration time
  - Examples?

Server sets cookies in headers

```
HTTP/1.0 200 OK
Date: Sun, 21 Apr 1996 02:20:42 GMT
Server: Microsoft-Internet-Information-Server/5.0
Connection: keep-alive
Content-Type: text/html
Last-Modified: Thu, 18 Apr 1996 17:39:05 GMT
Set-Cookie: trackingID=253407892435087
Content-Length: 2543
```

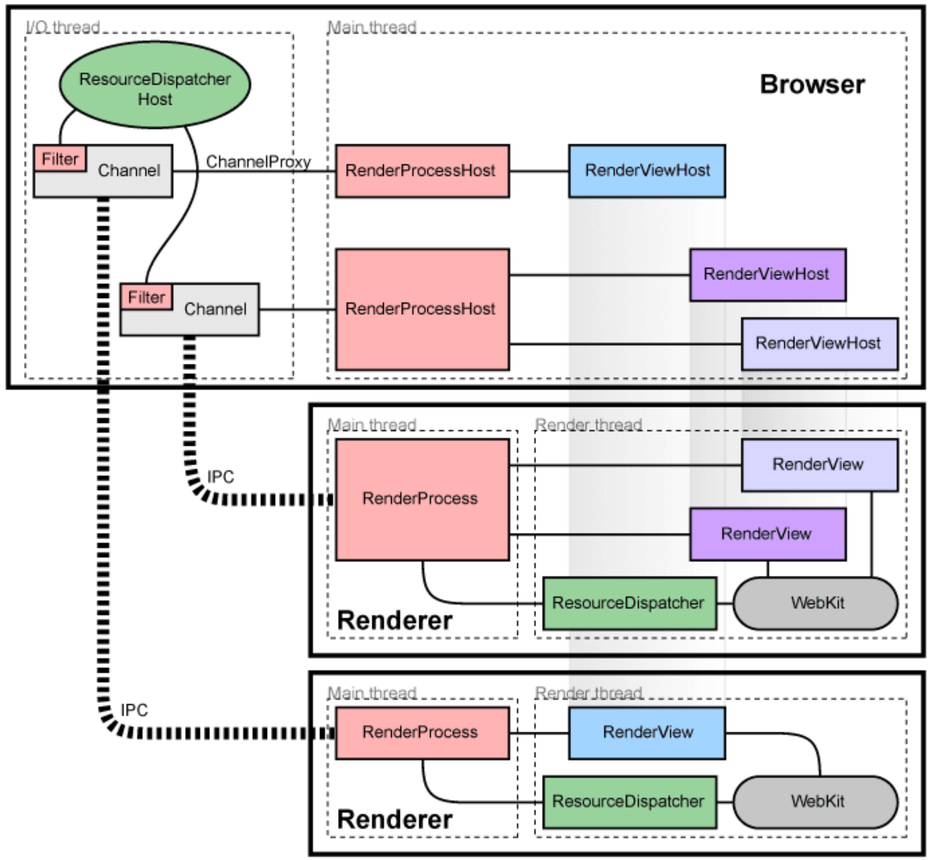
Client sends cookies every request

```
GET /index.html HTTP/1.1

Accept: image/gif, image/x-bitmap, image/jpeg, */*
Accept-Language: en
Connection: Keep-Alive
User-Agent: Mozilla/1.22 (compatible; MSIE 2.0; Windows 95)
Host: www.example.com
Cookie: trackingID=253407892435087
Referer: http://www.google.com?q=dingbats
```

Browser Execution Model

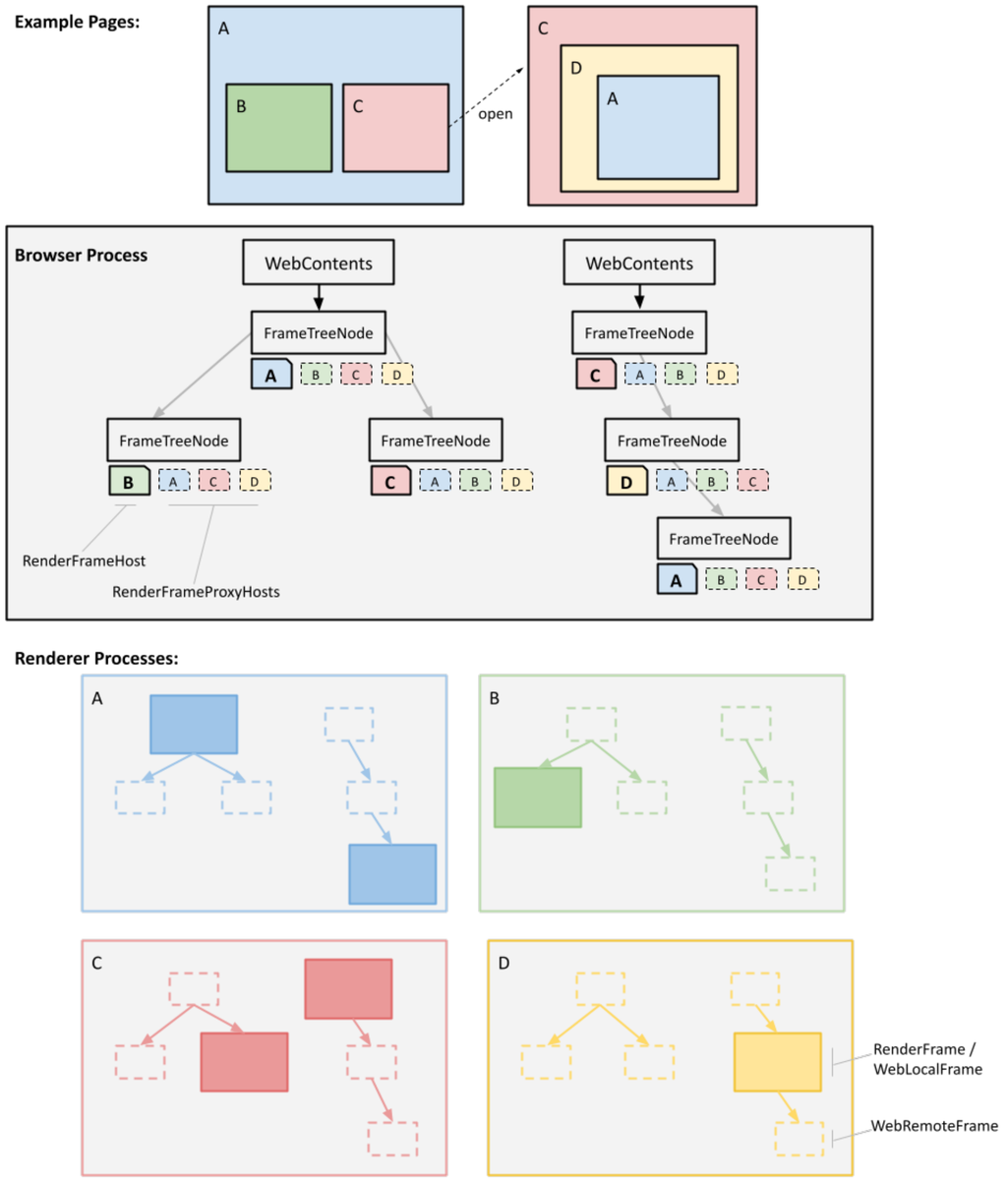
- Each browser window / tab…
  - Loads and renders content
  - Parses HTML and runs JavaScript
  - Fetches subresources (e.g., images, CSS, JavaScript)
  - Responds to events like onClick, onMouseover, onLoad, onTimeout



https://www.chromium.org/developers/design-documents/multi-process-architecture/

Nested Execution Model

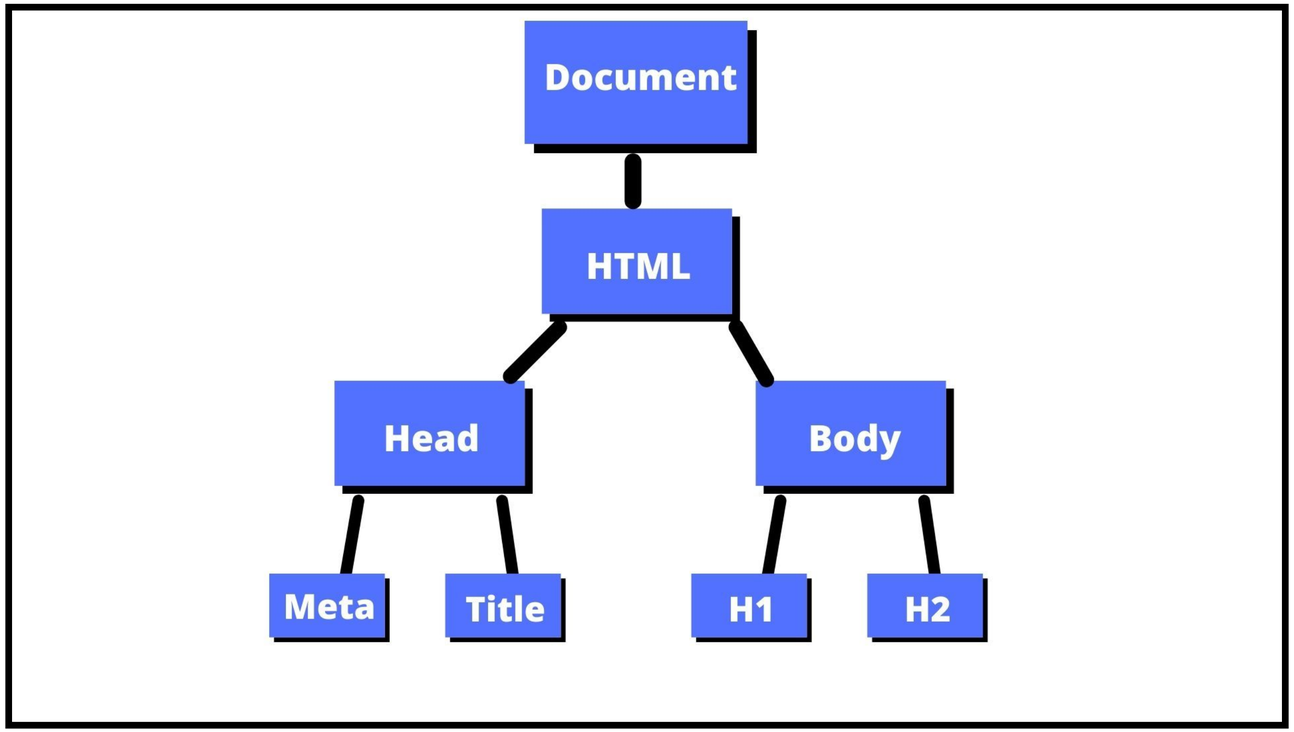
- Websites may contain frames from other sources
  - Frame: rigid, visible division
  - iFrame: floating, inline frame
- Why use frames?
  - Delegate screen area to content from another source (e.g., ads)
  - Browser provides **isolation** based on frame (remember site isolation?); each frame gets its own rendering process



https://www.chromium.org/developers/design-documents/oop-iframes/

What is the layout of webpages?

- Webpages follow a Document Object Model (DOM) — this is the structure of the page itself (encoded via HTML, sometimes XML)
  - The page itself is a *tree* of *nodes* designated by *tags* (e.g.,)



Can I modify the layout on the fly?

Can I modify the layout on the fly?

- Yes!
- The DOM can be manipulated via **code** or **scripts** included on the page
  - AKA: everything you see can be rendered dynamically
- Even the *browser* can be manipulated via code
  - Code can change your window, move you back and forward through browsing history, read cookies… *anything*

In that sense, websites are just programs

- Partially executed on the client side
  - HTML rendering, JavaScript, extensions, etc.
- Partially executed on the server side
  - Python, CGI, PHP, ASP, server-side JavaScript (ew), etc.
- And programs, as we know, can have lots of probelms

Next time…

- We get into the web security model
  - Same-Origin Policy
- We get into web attacks (SQL injection, XSS, CSRF)!
  - You'll implement a bunch of these in PA3