

Department of Computer Science
University of California, Los Angeles

Computer Science 144: Web Applications

Spring 2025
Prof. Ryan Rosario

Lecture 1: March 31, 2025

Outline

1 Syllabus / Logistics

2 Course Topics

3 A Bit About Me

4 Lecture 1

1 Syllabus / Logistics

2 Course Topics

3 A Bit About Me

4 Lecture 1

Welcome

Welcome to CS 144!



Syllabus

COMPUTER SCIENCE 144
Web Applications

Spring 2024
Prof. Ryan Rosario

Contact Information

Email:	rrosario@cs.ucla.edu
My Website:	http://ryanrosario.phd
Department Site:	http://web.cs.ucla.edu/~rrosario
Office:	Boelter Hall 3531C for in-person office hours https://ucla.zoom.us/my/rrosario
Office Hours:	Wednesday 3:30-4:30 in person (Zoom on request), Thursday 4-5pm only on Zoom. Day/time/modality is subject to change.

My Background

Ph.D., *Statistics*, UCLA.
M.S., *Computer Science*, UCLA.
B.S. *Mathematics of Computation, Statistics*, UCLA.

I am from industry where I have worked as a Software Engineer, Machine Learning Engineer and Data Scientist. I am originally from Thousand Oaks, CA – born and raised in Southern California and now live just outside Mammoth Lakes, CA.

About CS 144

Developing today's Web applications requires knowledge of a number of diverse topics, including the basic Web architecture, core Web standards (such as HTTP, HTML, CSS), JavaScript, asynchronous and functional programming, internet security, and cluster-based scalable architecture. Traditionally, these topics have been taught in different subdisciplines of computer science, so students had to take a fair number of courses to learn the basic concepts necessary to build effective and safe Web applications. The goal of this class is to teach students the most important concepts and give them first-hand experience with the basic tools for developing Web applications. It is worth noting that developing web applications is becoming more important for students interested in data science, data engineering and data visualization. The topics covered in the class include (but are not limited to):

- Core Web standards, such as HTTP, HTML, CSS, and JSON
- Javascript programming
- Web programming paradigms, including functional programming, asynchronous programming, and MVC
- Web-site architecture
- Web-site scalability
- Web security

Meeting Times

Activity	Day	Time	Location	Staff
LEC 1	Monday/Wednesday	06:00-07:50pm	MS 4000A	Prof. Rosario
LAB 1A	Friday	10:00-11:50am	DODD 78	Christian Aguilar
LAB 1B	Friday	12:00-01:50pm	ROLFE 3134	Mark Theeranantachai
LAB 1C	Friday	12:00-01:50pm	ROYCE 156	Tommy Vo Tran
LAB 1D	Friday	02:00-03:50pm	ROYCE 154	Tommy Vo Tran
LAB 1E	Friday	02:00-03:50pm	DODD 146	Mark Theeranantachai

1

The syllabus is posted on BruinLearn/Canvas. Please read and review. You are responsible for the policies in it.

COMPUTER SCIENCE 144

Web Applications

Spring 2025
Prof. Ryan Rosario

Contact Information

Email: rrosario@cs.ucla.edu
Department Site: <https://web.cs.ucla.edu/~rrosario>
Office: Boelter Hall 3531D for in-person office hours
<https://ucla.zoom.us/my/ryanrosario>
Office Hours: Wednesday 4–5pm in person, Thursday 4–5pm by appointment on Zoom.
Day/time/modality is subject to change.

My Background

Ph.D., *Statistics*, UCLA.

M.S., *Computer Science*, UCLA.

B.S. *Mathematics of Computation, Statistics*, UCLA.

I am from industry where I have worked as a Software Engineer, Machine Learning Engineer and Data Scientist. I am originally from Thousand Oaks, CA — born and raised in Southern California and now live just outside Mammoth Lakes, CA.

About CS 144

Developing today's Web applications requires knowledge of a number of diverse topics, including the basic Web architecture, core Web standards (such as HTTP, HTML, CSS), JavaScript, asynchronous and functional programming, internet security, and cluster-based scalable architecture. Traditionally, these topics have been taught in different subdisciplines of computer science, so students had to take a fair number of courses to learn the basic concepts necessary to build effective and safe Web applications. The goal of this class is to teach students the most important concepts and give them first-hand experience with the basic tools for developing Web applications. It is worth noting that developing web applications is becoming more important for students interested in data science, data engineering and data visualization.

Meeting Times

Activity	Day	Time	Location	Staff
LEC 1	Monday/Wednesday	06:00–07:50pm	MS 4000A	Prof. Rosario
LAB 1A	Friday	10:00–11:50am	BOELTER 5436	Vani Agrawal
LAB 1B	Friday	12:00–01:50pm	DODD 78	Mark Theeranantachai
LAB 1C	Friday	12:00–01:50pm	BOELTER 5436	Vani Agrawal
LAB 1D	Friday	02:00–03:50pm	DODD 78	Mark Theeranantachai

Special Note

This year's offering of CS 144 has been significantly revamped to reduce overlap with other courses and to introduce more systems-based topics.

Prerequisites

CS 143 is one of two options to meet the prerequisites for the class. We will be using at least MongoDB for this class. We may also use other NoSQL systems, and PostgreSQL or MySQL/MariaDB as our RDBMS.

CS 111 is the other option. In the instructor's opinion, completing CS 111 is a testament to understanding the basics of systems and tradeoffs. It also allows tenured students to take the class without having taken CS 143.

PTE Numbers



All PTE requests must be made by Google form: (<https://bit.ly/cs144pte25s>).

No late work will be accepted as a result of waiting for a PTE or Excess Units petition. If you are interested in either, you will need to submit work on time to your TA.

Lecture Activities

Students are responsible, and will be held accountable, for all content, material and announcements made in lecture regardless if they are recorded. This also applies to all Campuswire, BruinLearn, and MyUCLA announcements/emails. Students that choose to disable email notifications must check these resources regularly.

Textbooks

There are no required textbooks for this course. We will provide references to standards and specifications throughout the quarter, as well as other helpful resources.

If you like having some printed reference material, below are some that are recommended by the instructor.

1. [Web Scalability for Startup Engineers](#) by Artur Ejsmont.
2. [Eloquent JavaScript, Third Edition](#) by Marijn Haverbeke.
3. [JavaScript: The Definitive Guide, Seventh Edition](#) by David Flanagan.
4. [CSS in Depth](#) by Keith J. Grant.
5. [Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems](#) by Martin Kleppmann.

You can access some of these books for free as a student via the UCLA Library.

Grading

Mini-Projects	35%	≤ 4 total, individual
Checkpoint Quizzes	30%	Five, online, on Thursdays
Final Project	35%	Individual

Mini Projects

There will be about 4 (\leq) mini-projects throughout the quarter. The purpose of these projects is to give you hands-on experience with the concepts discussed in lecture. Some of these projects may also contain questions for you to answer.

Late Work: To accommodate the emergencies that a student may encounter, each student has a **6-day** grace period for late submission to use throughout the quarter for projects, but no project may be submitted more than 2 days late. For example a student cannot use all 6 days on one assignment. If a project is submitted 1 minute late, this counts as 1 day. *Do not treat the grace period as the deadline. If you have trouble submitting before the grace period ends, we cannot give you additional time.*

Assessment

There are no formal high-stakes exams for this class; however, to make sure that students are keeping up, there will be 5 Canvas or Gradescope quizzes throughout the quarter. Unless otherwise noted, these will open on Thursday and students will have until 11:59pm to take the quiz. You may use class notes and class materials. **More info will be shared in lecture.**

The final exam period is assigned by the Registrar and is **Wednesday, June 11, 6:30-9:30pm**. While there is no final exam, we may use the time for activities related to the final project.

Final Project

You will create a web application (not a web *site*) from scratch using the skills you learned in this class. You may also use other analogous technologies if you wish. The purpose is to show what you learned. We will ask you to write a short proposal that we will review. Then you will provide us with short progress reports. We will give you access to a class Github organization to commit your work. Once you complete the course, you can move the project to your own personal Github account. More information will be provided in a few weeks.

Final grades will be assigned based on how well the class, as a whole, performs compared to my expectations.

Request for Regrades There are typically **no regrades** on any assignment unless we agree that the rubric was not applied correctly. Arguing the size of a point deduction or the purpose of a rubric item is not a valid regrade request. If there are questions about a grade, please ask on Piazza. Reserve Gradescope only for actual regrade requests.

All requests for regrades on an assignment must be communicated to course staff on Gradescope. For work with grades posted up to Friday of Week 9, all regrades must be requested within 7 calendar days (1 week) of the grades being posted. For work with grades posted after Friday of Week 9, this window will be 48 hours. There will be absolutely no regrades permitted after this window for any reason.

Disclaimer

If any changes need to be made to the course or policies due to major events, they will be communicated in writing to all students and course staff. Anything not covered in this syllabus will be decided by the instructor, within the bounds of policy, after verifying with Department or SEAS leadership.

Academic Integrity

The UCLA Department of Computer Science and the instructor both treat academic integrity very seriously. The instructor does not tolerate any form of academic dishonesty. **All submitted work for this class will be rigorously checked with anti-plagiarism software.** Students that are suspected of violating UCLA's academic integrity policies will be referred to the Dean of Students, no exceptions. Any outside source referenced during an assignment should be clearly cited. If a student is ever in the position where they are unsure if they may be about to partake in behavior that could be deemed dishonest, they should ask the instructor.

Using previous solution sets, or publicly posting your own work from this class (including but not limited to Github), is prohibited. In order for any of your work this quarter to be graded, you *must* sign the Academic Integrity Agreement on BruinLearn (listed under Quizzes).

Using ChatGPT or any other LLM technology to complete or assist with *any* part of *any* assignment is prohibited and will be handled according to this policy.

Tentative Schedule

Below is a very tentative schedule of this course. The schedule is **likely to change** based on our pace, or student interest in other topics.

Week	Lecture	Date	Topic
1	1	March 31	Introduction to CS 144, HTTP/DNS
	2	April 2	Evolution of HTML and HTML5 Features
2	3	April 7	CSS and the Cascade
	4	April 9	CSS Layout and Responsive Design
3	5	April 14	JavaScript OOP, HTML DOM, Event Handling
	6	April 16	Asynchronous Programming, Promises
4	7	April 21	TypeScript
	8	April 23	Node.js and the Foundations of Server-Side JavaScript
5	9	April 28	API Design — REST, RPC, GraphQL and Streaming
	10	April 30	Maintaining State
6	11	May 5	WebAssembly and Progressive Web Apps (PWA)
	12	May 7	Frameworks, Part 1
7	13	May 12	Frameworks, Part 2
	14	May 14	Frameworks, Part 3
8	15	May 19	Deploying Web Apps: Docker, Kubernetes, Serverless
	16	May 21	Authentication, Security and Privacy
9	-	May 26	HOLIDAY: Memorial Day
	17	May 28	Info Retrieval, User Modeling, and Monetization
10	18	June 2	Vector Databases, Integrating with AI
	19	June 4	Accessibility or Emerging Trends
Finals	-	June 11	FINAL EXAM PERIOD, 6:30-9:30pm

Logistics: PTE

If you are requesting a Walkup PTE, please fill out the form at
<http://bit.ly/cs144pte25s.png> PTEs will be distributed
Week 2.



Prerequisites

Computer Science 143 and/or Computer Science 111 is an enforced prerequisite for this class. You may be successful without 143. CS 111 was added as an option for tenured CS majors that have not taken 143. It is a way to show some experience with systems —

- ① Work comfortably at the Unix/Linux command line;
- ② Be a “competent” programmer with the ability to learn small parts of other languages when necessary. Aside from what we learn in this class, that would be Python and PHP.
- ③ Write/understand basic SQL and understand how, say, MongoDB is different from relational systems.

We will make passing reference to concepts from CS 111 (Operating Systems), CS 131 (Programming Languages) and CS 118 (Networks). It should not block you.

My Contact Information

My email is `rrosario@cs.ucla.edu` but please use it as a last resort, such as discussing a personal matter related to the course, with the professor only. **This is the only email address that I respond to.**

I will have two office hours per week. These are going to change:

- ① **Wednesday, 4–5pm, Boelter 3531D.**
- ② **Thursday, 4–5pm, Zoom Only.**

I can also be on Zoom
(<https://ucla.zoom.us/my/ryanrosario>) if you send me an email.

Piazza

All students will be added to Piazza shortly.

- ➊ For course questions, clarifications, confusions, etc. please use Piazza.
- ➋ For grading logistics questions specific to you, please post a **private post to Instructors**.
- ➌ Please do not send us course material questions via email. *We use the forum so everyone can benefit.*
- ➍ **Please do not post any of your work on the forum publicly where other students can see it and copy it.**
- ➎ *Students are anonymous to each other, but not to the course staff.*

Google Docs



Each student has free access to Google Drive. I will be posting all course materials on Google Drive, and linking to them from BruinLearn.

This also means that if I make changes to the slides, you will always have the most up to date copy!

That said, **you will need to routinely log into your UCLA Google Apps account to access this material.**

Our TAs

We have **three** TAs this quarter. Get to know them, even the TAs that are not for your section. They will be assisting you with homework etc. and are good people to know.



Vani Agrawal
(1A, 1C)

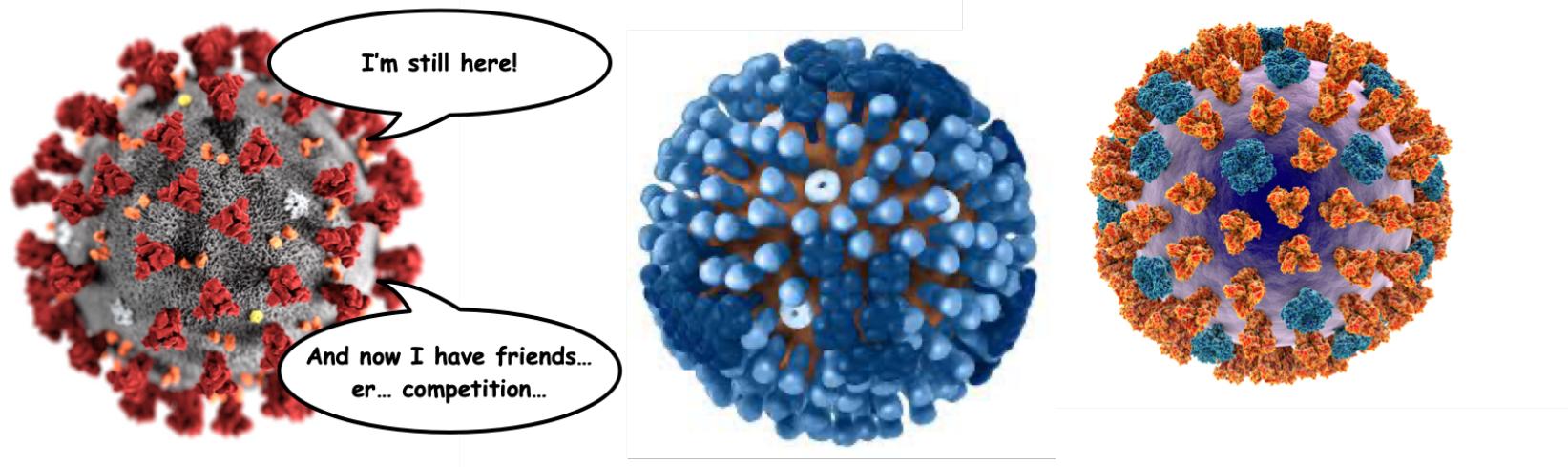


Mark Theeranantachai
(1B, 1D)

Your TAs will share their contact information and office hours in their sections and will be posted on BruinLearn.

COVID-19 / The Three Musketeers

Course staff will abide by the current UCLA requirements etc.



If you are sick, of any kind, please **do not** come to class **or office hours**. There are several ways to catch up, and I have Zoom office hours.

Your Mental Health Also Matters

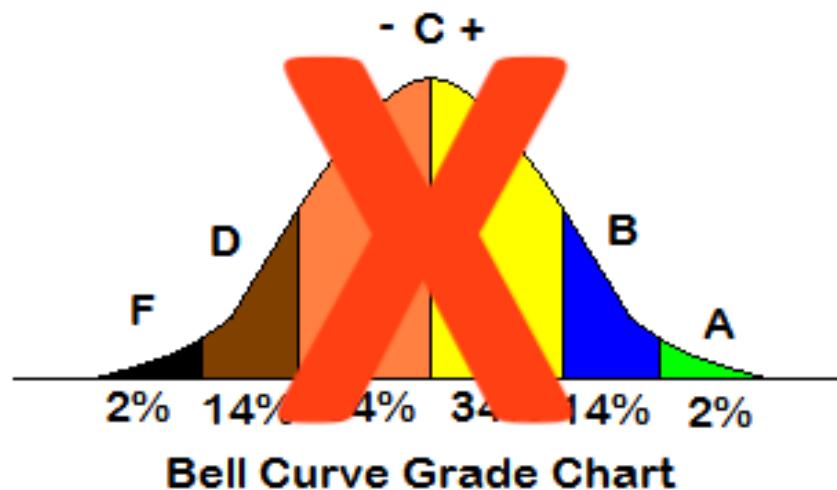
You only get one brain, and one life. University life can be challenging. CAPS is there to help ([click here](#)).



During Weeks 5–10, CAPS offers a satellite counseling service for students in Engineering. I will share the flyer when I receive it. Be good to yourselves!

Grades

Your grade is based on how well you perform compared to my expectations, and the final grade cutoffs for the class will be developed based on these expectations. The cutoffs are then adjusted based on normative performance.



35% Mini-Projects
30% Quizzes
35% Final Project

Mini-Projects

There will be mini-projects throughout the quarter. These are designed to give you hands-on practice with what we have learned.

All assignments must be submitted through Gradescope. No email attachments. Due dates will vary, but will usually be Wednesday, Thursday, Friday.

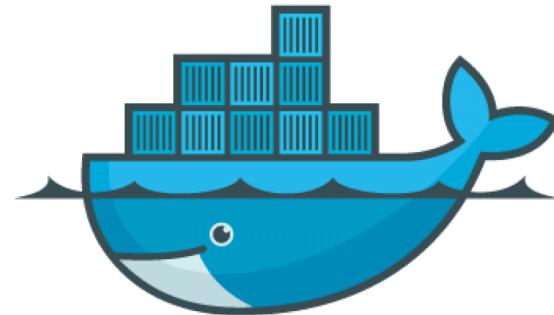
There will be at most four of them.

Mini-Projects (contd.)

We have credits for [Google Cloud](#).



Google Cloud



docker

We have made changes to the process that should be smoother this year. You can also develop on a laptop and launch into Cloud if need be.

Regrade Requests

Typically there are no regrades:

- ① If you have a question about an error, please ask privately on Piazza.
- ② If you believe the rubric was not applied correctly *and we agree or are not sure* make a regrade request on Gradescope and we will review it.
- ③ General arguments about the rubric or the size of deductions will not receive a response.

Late Work

You will have **six** grace period days to use for the mini-projects throughout the quarter.

- ➊ You can use these days for any submission.
- ➋ A maximum of 2 per assignment.
- ➌ 1 minute late = 1 day late.
- ➍ Any additional days late will result in no credit.

Quizzes

There will be no formal exams.

There will be 5 quizzes throughout the quarter on Canvas or Gradescope. **For now, they will be even weeks on Thursdays.** You have the entire day to complete them.

They are open notes and course materials.

5-10 questions, about 20% short answer or short code. The others are objective.

These are meant to be checkpoints to make sure you are following along, not formal high-stakes assessments.

Final Project

June 11, 6:30-9:30pm PDT, In-Person

(There will not be a final exam for this class, but we may still use the time)

Instead, you will work on a final project, of your choosing. You will implement a web app (not a website) from scratch.

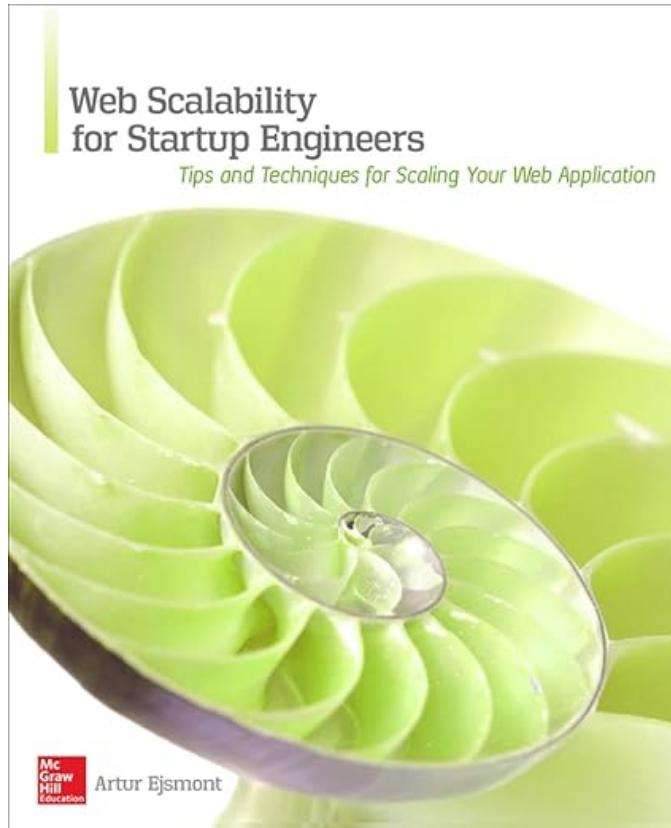
You are expected to use concepts learned in class (though you are not limited to only these) and will need to write a short proposal so we know you are on track.

Textbooks

There are **no** required textbooks for this course!

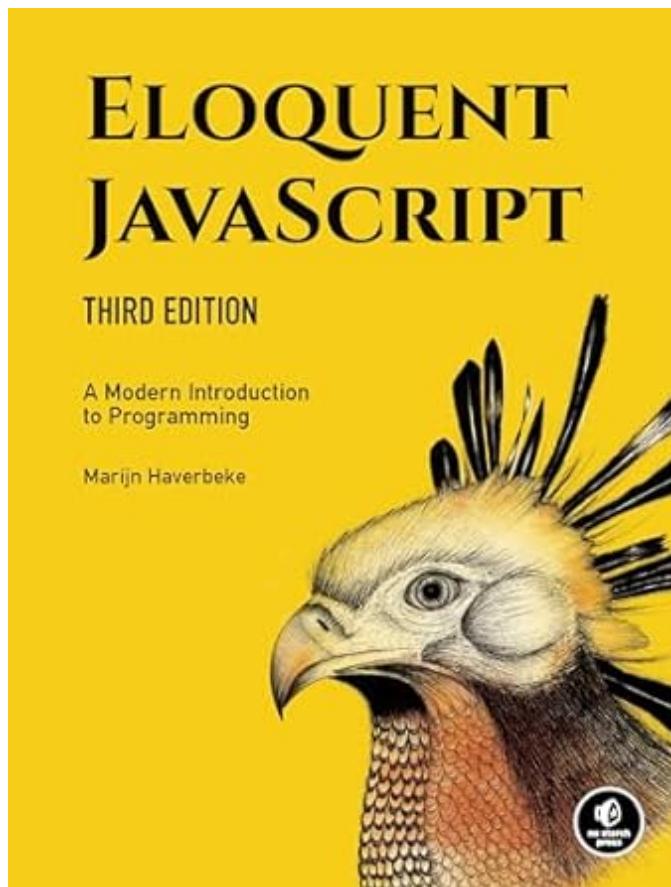
There are some books that were recommended in earlier offerings of CS 144 and by your instructor...

Textbooks (contd.)



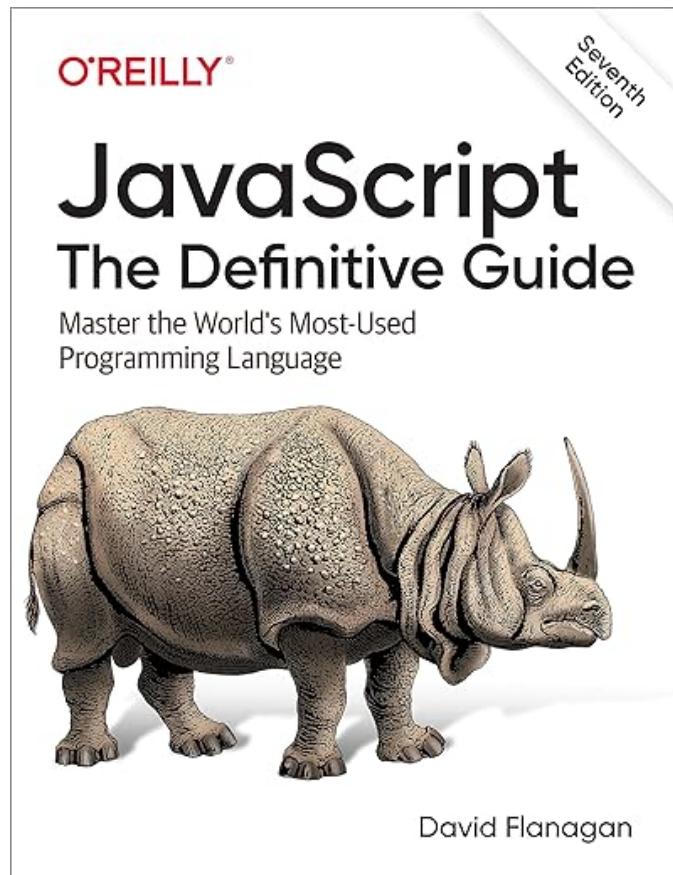
Web Scalability for Startup Engineers
Artur Ejsmont

Textbooks (contd.)



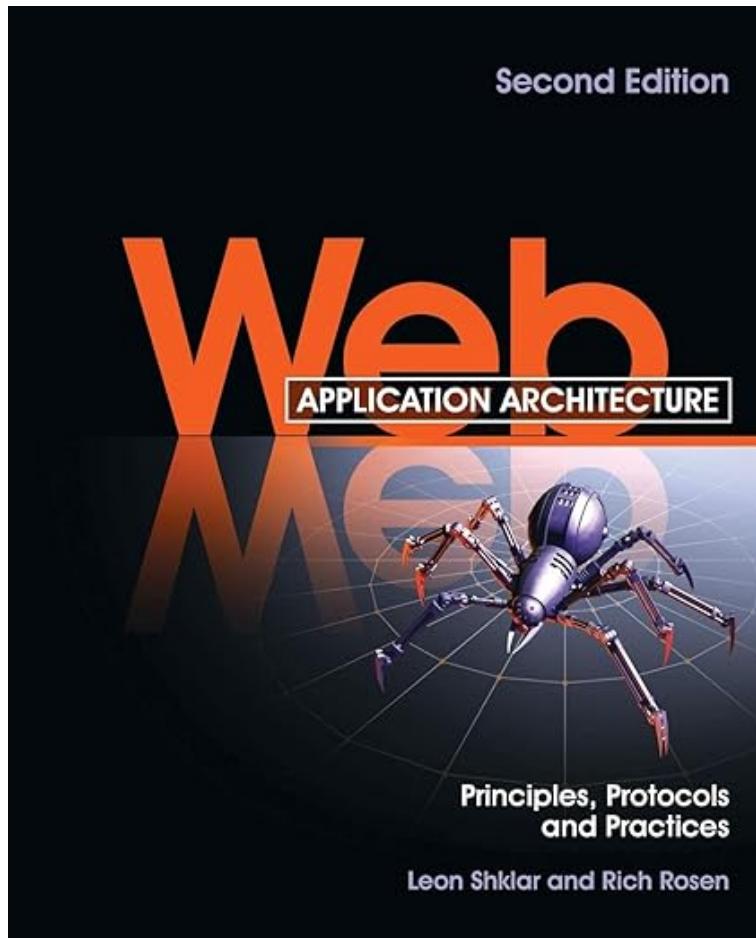
Eloquent Javascript, Third Edition
Marijn Haverbeke

Textbooks (contd.)



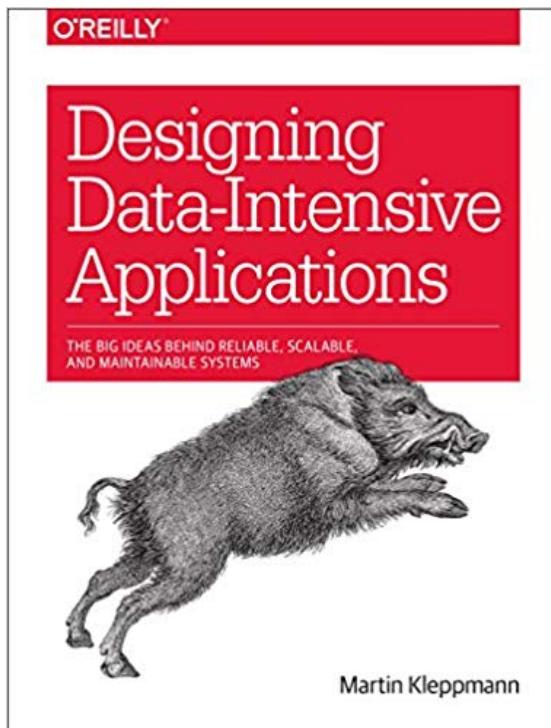
JavaScript: The Definitive Guide, 7th
Edition
David Flanagan

Textbooks (contd.)



Web Application Architecture:
Principles, Protocols and
Practices, Second Edition
Leon Shklar and Rich Rosen

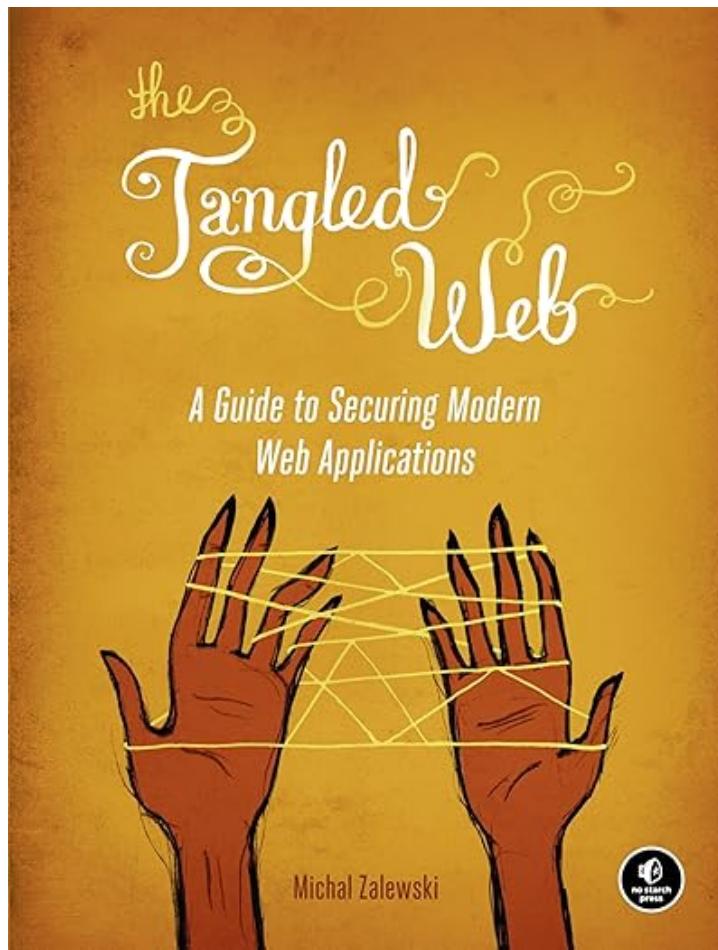
Textbooks (contd.)



Designing Data-Intensive
Applications
Martin Kleppmann

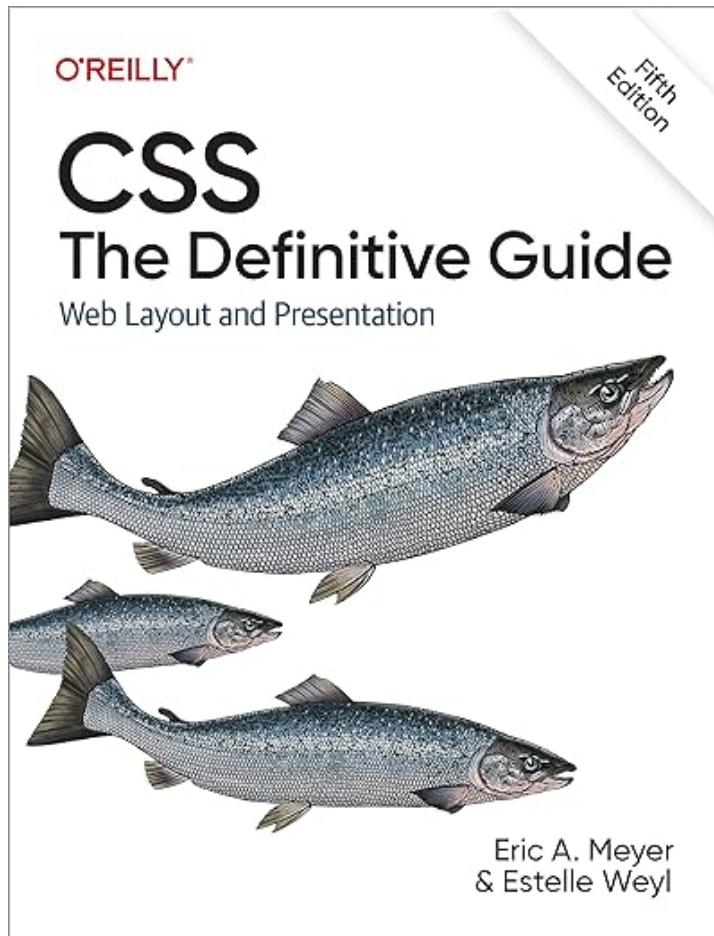
You have access to this as a UCLA
student
<https://bit.ly/cs143-kleppmann>.

Textbooks (contd.)



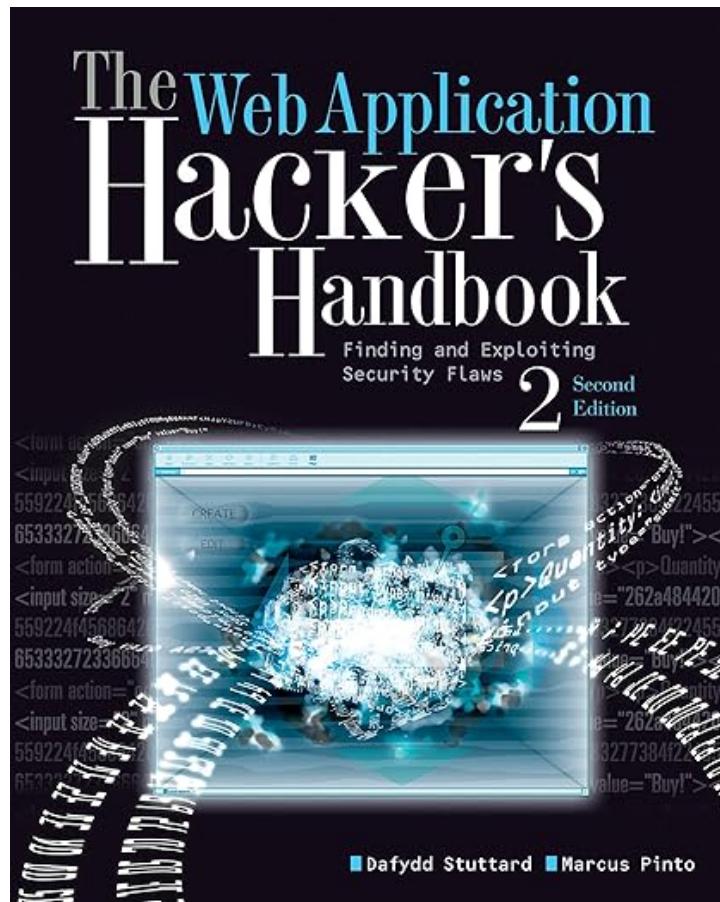
The Tangled Web: A Guide to
Securing Modern Web
Applications
Michał Zalewski

Textbooks (contd.)



CSS: The Definitive Guide
Eric Meyer and Estelle Weyl

Textbooks (contd.)



The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws
Stuttard and Pinto

Academic Integrity

The *UCLA Department of Computer Science* and the instructor both treat academic integrity very seriously. The instructor absolutely does not and will not tolerate any form of academic dishonesty.

Students that are suspected of violating UCLA's academic integrity policies will be referred to the *Office of the Dean of Students*.

If a student is ever in the position where they are unsure if they may be about to partake in behavior that could be deemed as violating this policy, they should ask the instructor.

Academic Integrity: Particularly for CS

There are additional things to be aware of specifically for CS classes.

- ① Clarifying with other students the requirements of the assignment **is OK**.
- ② Discussing algorithms for solving a problem, *without code*, **is OK**.
- ③ Helping someone find a minor bug in their code **is OK**.
- ④ Using code provided in the required textbook, from the instructor, from the TAs, or from reference materials provided in the class **is OK**.

Academic Integrity: Particularly for CS

There are additional things to be aware of specifically for CS classes.

- ① Submitting any portion of someone's work, if not from the resources on the previous slide **is NOT OK**.
- ② Using solutions from previous offerings of this or any other class **is NOT OK**.
- ③ *Soliciting* help from online sources, including (but not limited to) AI agents like ChatGPT **is NOT OK**.
- ④ Writing for, or with, another student a code fragment that solves any portion of a programming assignment **is NOT OK** unless otherwise indicated by the instructor.

Academic Integrity: Particularly for CS

There are a few other policies to bring to your attention:

- ① Posting your solutions and answers publicly from this class such that others from this offering or any future offering of the course may violate Academic Integrity **is NOT OK**.
- ② Impersonating another individual or falsifying your identity on any of the systems used for this course **is NOT OK**.
- ③ Behavior that disrupts the instructional process (e.g. Zoombombing) **is NOT OK**.

Academic Integrity: ChatGPT and LLMs



ChatGPT and other interfaces to LLMs have revolutionized how we get information and perform tasks.

These technologies must **never** be used to complete or assist **any** part of **any** assignment in this class. If we find that students have matching weird answers, the situation will be treated as a standard academic integrity case.

Academic Integrity: ChatGPT and LLMs (contd.)

These technologies **can** be used for studying and creating practice problems, but not for completing work. Here are some examples:

I don't understand asynchronous programming at all. Given the following code my professor gave me in lecture, convert it to be asynchronous. What is the difference and what is the purpose?

The code provided by the professor is synchronous and will block, or wait, for a response from the web server before continuing. This is not ideal. Asynchronous programs do not block and instead register a callback that will execute when the proper data is ready...

But it is quite often **wrong!** (just not in this prompt)

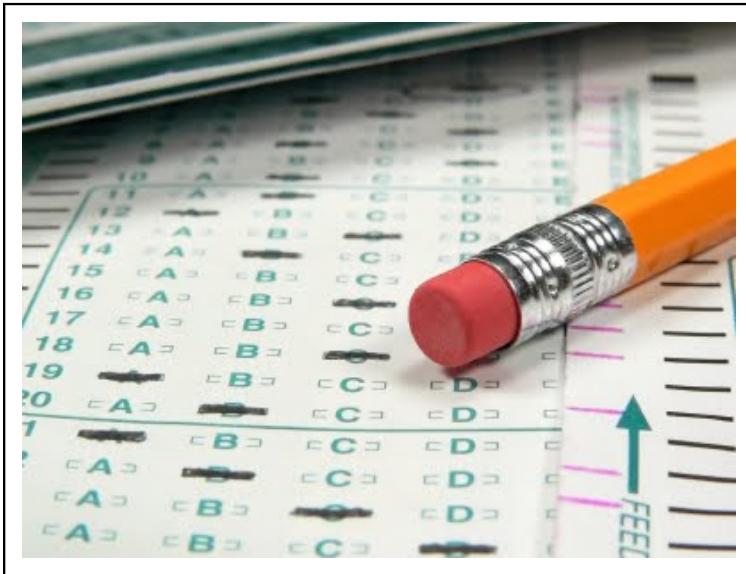
Academic Integrity: ChatGPT and LLMs (contd.)

Never use it to help with **any** course work:

I am having trouble creating a puffy blue button. When I click on it, it should look like it's being pressed in, but it's not working. My code is below. How can I fix it?

Sure! Here's the code...

Academic Integrity Agreement



[Click Here](#)

To receive credit for your work, you will need to digitally sign the Academic Integrity Agreement. It is on BruinLearn/Canvas under Week 1.

1 Syllabus / Logistics

2 Course Topics

3 A Bit About Me

4 Lecture 1

Purpose of this Course

The purpose of CS 144 is to teach students about web technologies and architecture both at a conceptual and practical level.

As this course is offered in the 14x series, the data systems and concepts series, it is worth noting that web applications are becoming more important in data science, data engineering and data visualization.

Revamped

I have revamped this course considerably since last year. We even changed the course description.

- ➊ Less focus on the basics. Given the prerequisites, you can learn the basics quickly if need be.
- ➋ More focus on more advanced topics and **system elements**.
- ➌ More focus on web data and less on presentation – the **full stack** of web development.

Still, there will be more work to do. **I will post a survey/form asking you about topics you are interested in that are not in the syllabus.**

Revamped (contd.)

Still...

- ➊ The schedule of topics **will** change and we may not make it through everything. That is OK.
- ➋ Our pace will vary.
- ➌ Our lab sessions may cover additional material needed for projects (or you can learn it on your own).
- ➍ Certain concepts may be covered a first time, and then revisited later on.

This course fills a gap. You can use it to pivot to be a **full stack** software engineer.

Some Technologies We Will Work With



So, Why Offer CS 144?

Most CS courses at UCLA (and most research universities):

- ➊ Focus on a specific CS *research area*.
- ➋ Are more theoretical than practical.
- ➌ Focus on backend development, not frontend.

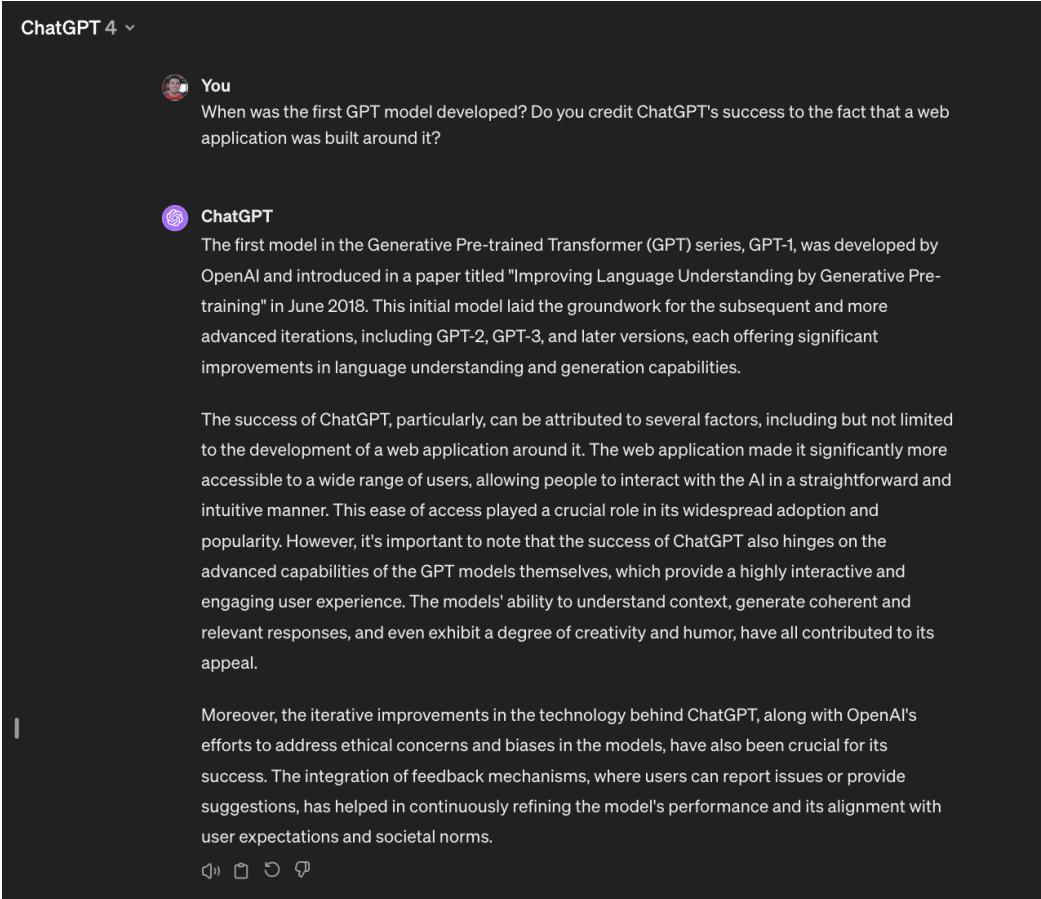
So, Why Offer CS 144? (contd.)

The thing is, your application or system is not useful to users if they... **cannot use it.**

For more users, that means providing a user interface they can use to interact with it.

So, Why Offer CS 144? (contd.)

Example: ChatGPT.



The screenshot shows a dark-themed interface of the ChatGPT 4 web application. At the top, it says "ChatGPT 4". A user message from "You" asks, "When was the first GPT model developed? Do you credit ChatGPT's success to the fact that a web application was built around it?". Below it, ChatGPT responds: "The first model in the Generative Pre-trained Transformer (GPT) series, GPT-1, was developed by OpenAI and introduced in a paper titled "Improving Language Understanding by Generative Pre-training" in June 2018. This initial model laid the groundwork for the subsequent and more advanced iterations, including GPT-2, GPT-3, and later versions, each offering significant improvements in language understanding and generation capabilities. The success of ChatGPT, particularly, can be attributed to several factors, including but not limited to the development of a web application around it. The web application made it significantly more accessible to a wide range of users, allowing people to interact with the AI in a straightforward and intuitive manner. This ease of access played a crucial role in its widespread adoption and popularity. However, it's important to note that the success of ChatGPT also hinges on the advanced capabilities of the GPT models themselves, which provide a highly interactive and engaging user experience. The models' ability to understand context, generate coherent and relevant responses, and even exhibit a degree of creativity and humor, have all contributed to its appeal. Moreover, the iterative improvements in the technology behind ChatGPT, along with OpenAI's efforts to address ethical concerns and biases in the models, have also been crucial for its success. The integration of feedback mechanisms, where users can report issues or provide suggestions, has helped in continuously refining the model's performance and its alignment with user expectations and societal norms." At the bottom of the message area, there are small icons for reply, delete, and other actions.

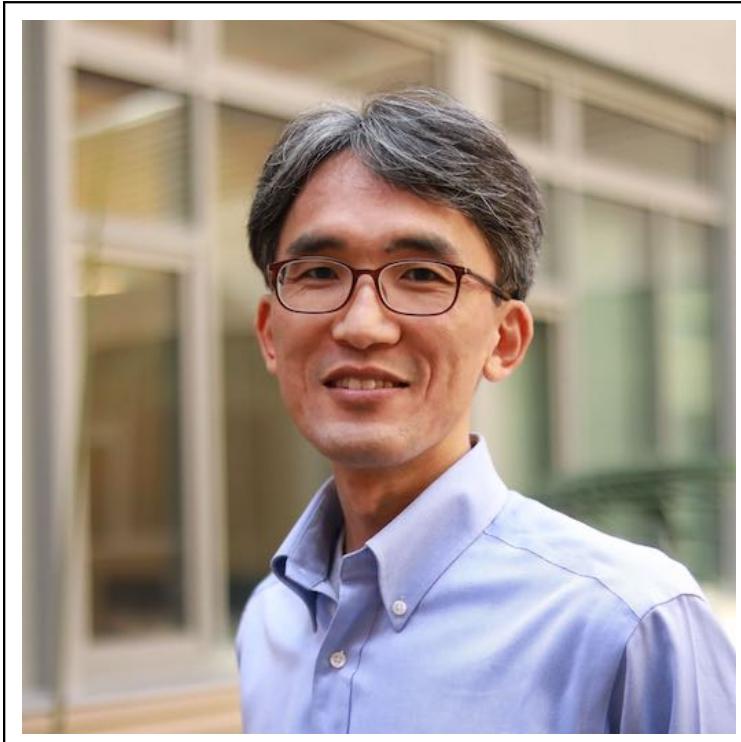
So, Why Offer CS 144? (contd.)

GPT was first released in *2018* but was mostly used by researchers and engineers, not end users.

That did not come until *2022* when OpenAI released a chatbot based on GPT – a web application.

Users could now *use* the GPT LLM, and the rest is history.

So, Why Offer CS 144? (contd.)



Prof. Junghoo Cho, creator of CS 144

Oh, and web development **is** part of a research area that focuses on web information management.

Systems like search engines and recommender systems require good web development hygiene.

1 Syllabus / Logistics

2 Course Topics

3 A Bit About Me

4 Lecture 1

whoami



UCLA

Ph.D., *Statistics*

M.S., *Computer Science*

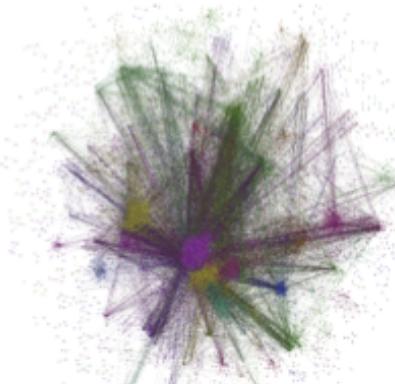
B.S., *Mathematics of Computation*

I've been at UCLA forever...

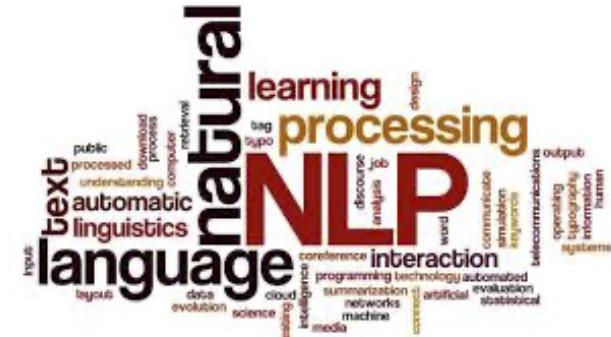
Research Interests



**Machine
Learning**



**Web Mining/
Web Research**



**Natural Language
Processing**

Research Interests



I am From Industry

Some places I've worked:



Google: Software Engineer (Current!)



Google Cloud



kubernetes

My team's goal is supporting AI/ML workloads on Google Kubernetes Engine, particularly for **inferencing**.

Google: Data Scientist (Past!)



- Develop metrics, experiments and testing methodology to make sure that new search ranking algorithms or features positive affect user experience
- Support engineering launches by checking important user satisfaction metrics.
- Research users' experience with controversial content
- Research issues involving quality of the search human rater program (psychometrics).

Google: Data Scientist (Past!) (contd.)

It looks like there aren't many great matches for your search

Tip: Try using words that might appear on the page you're looking for. For example, "cake recipes" instead of "how to make a cake."

Need help? Check out other tips for searching on Google.

It looks like these results are changing quickly

If this topic is new, it can sometimes take time for results to be added by reliable sources

What I Did at Facebook: Machine Learning Engineer



- Developed the company-wide sentiment analysis system.
- Developed a system for topic identification and topic summarization.
- Worked on automated tagging of IT tickets.
- Worked with HR's People Analytics team using classical statistics to analyze inequities in performance review text and leveling by demographics.

1 Syllabus / Logistics

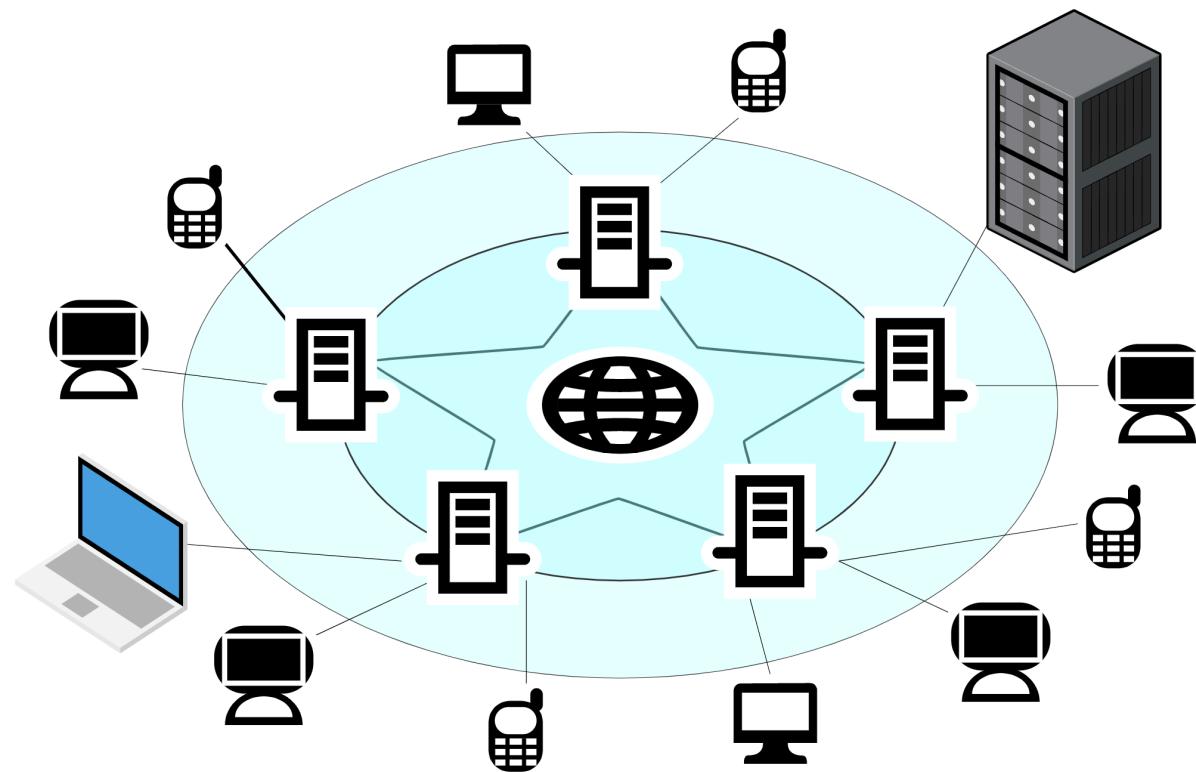
2 Course Topics

3 A Bit About Me

4 Lecture 1

- Elementary Concepts
- History of HTTP
- Brief Discussion of DNS

To understand our web technologies, we need to know a bit about how computer networks, particularly the Internet works.



Web apps can be exposed on the Internet or only on a LAN or WAN.

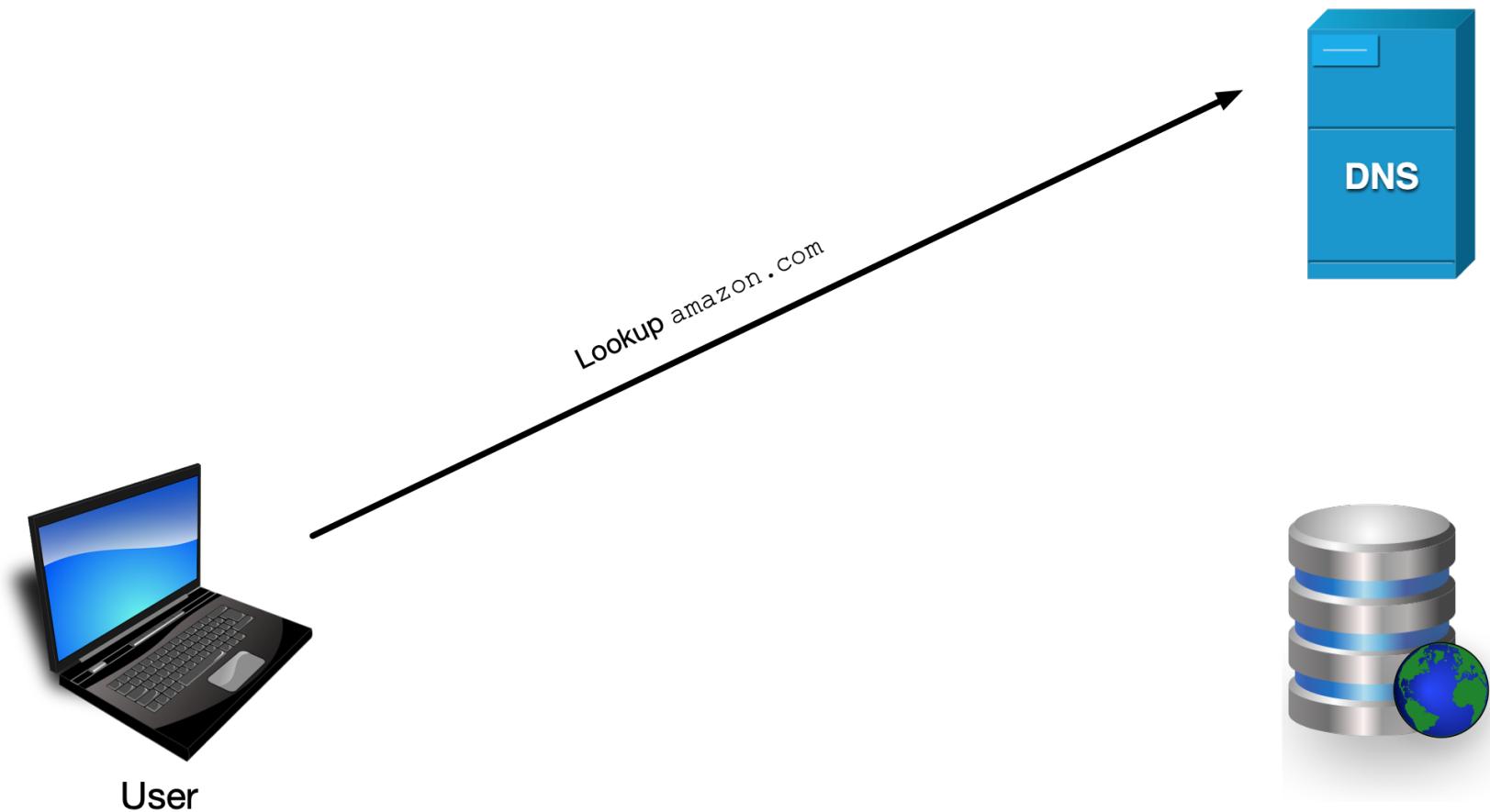
What Happens When We Visit amazon.com?



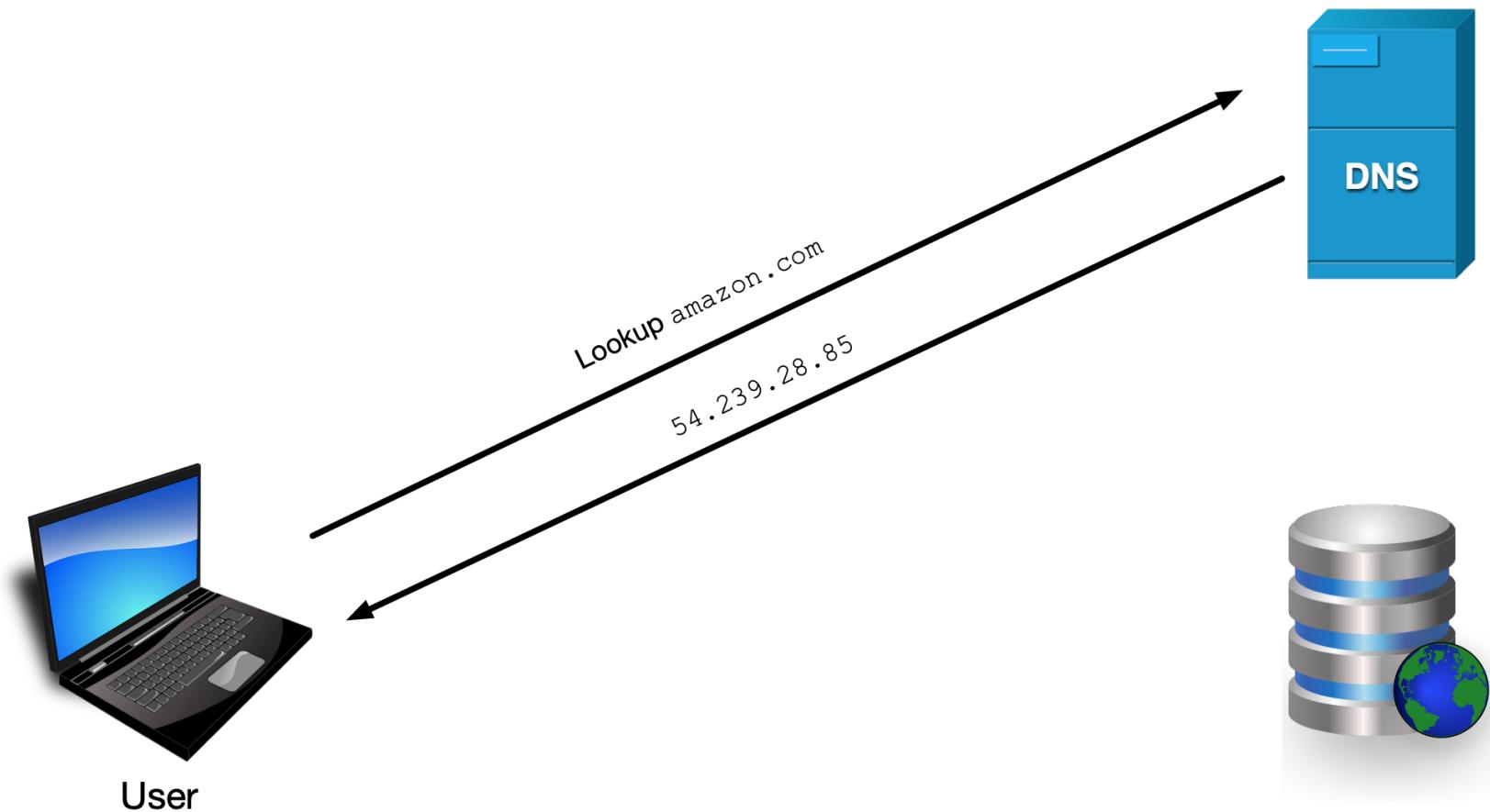
User



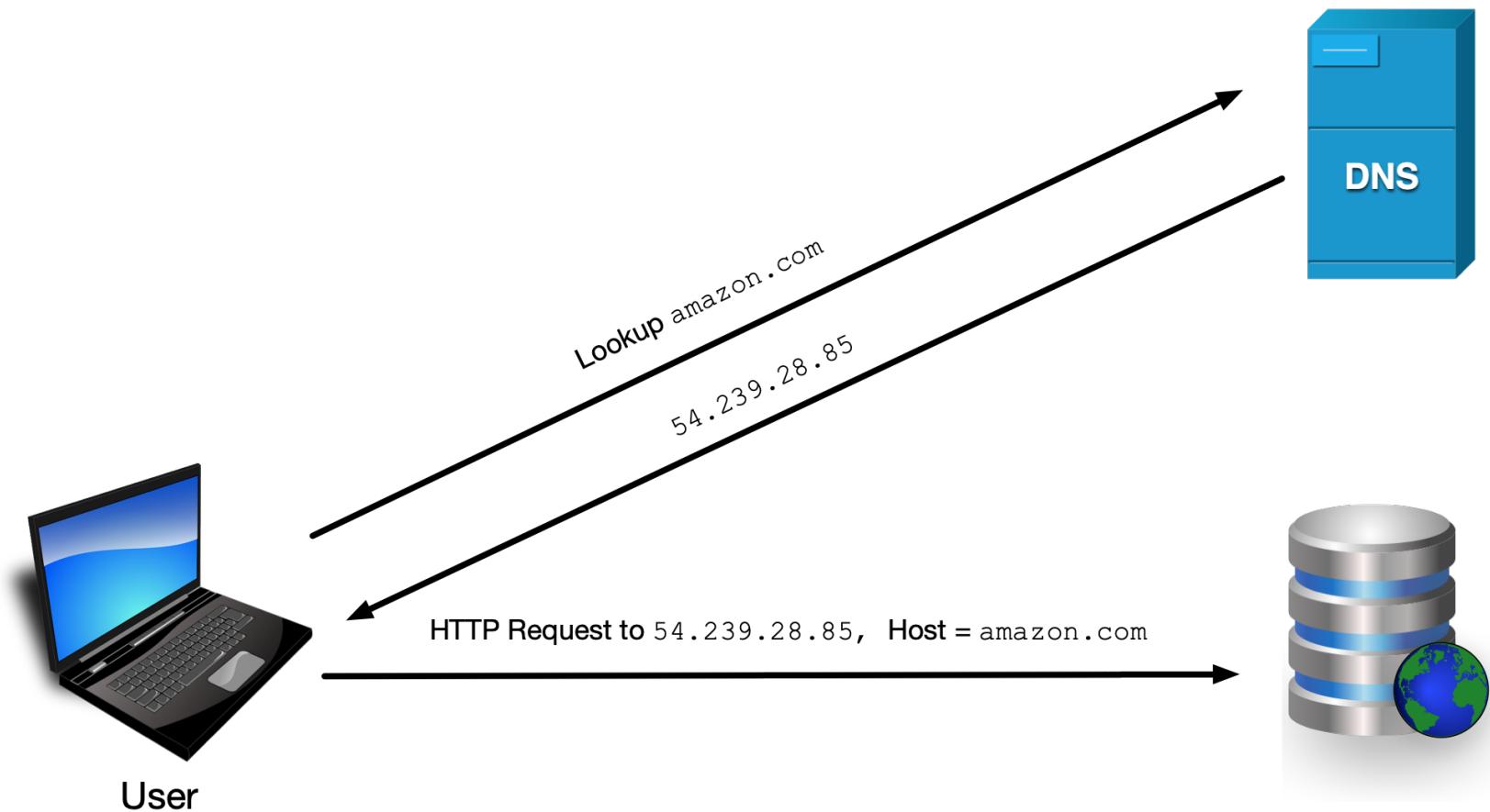
What Happens When We Visit amazon.com?



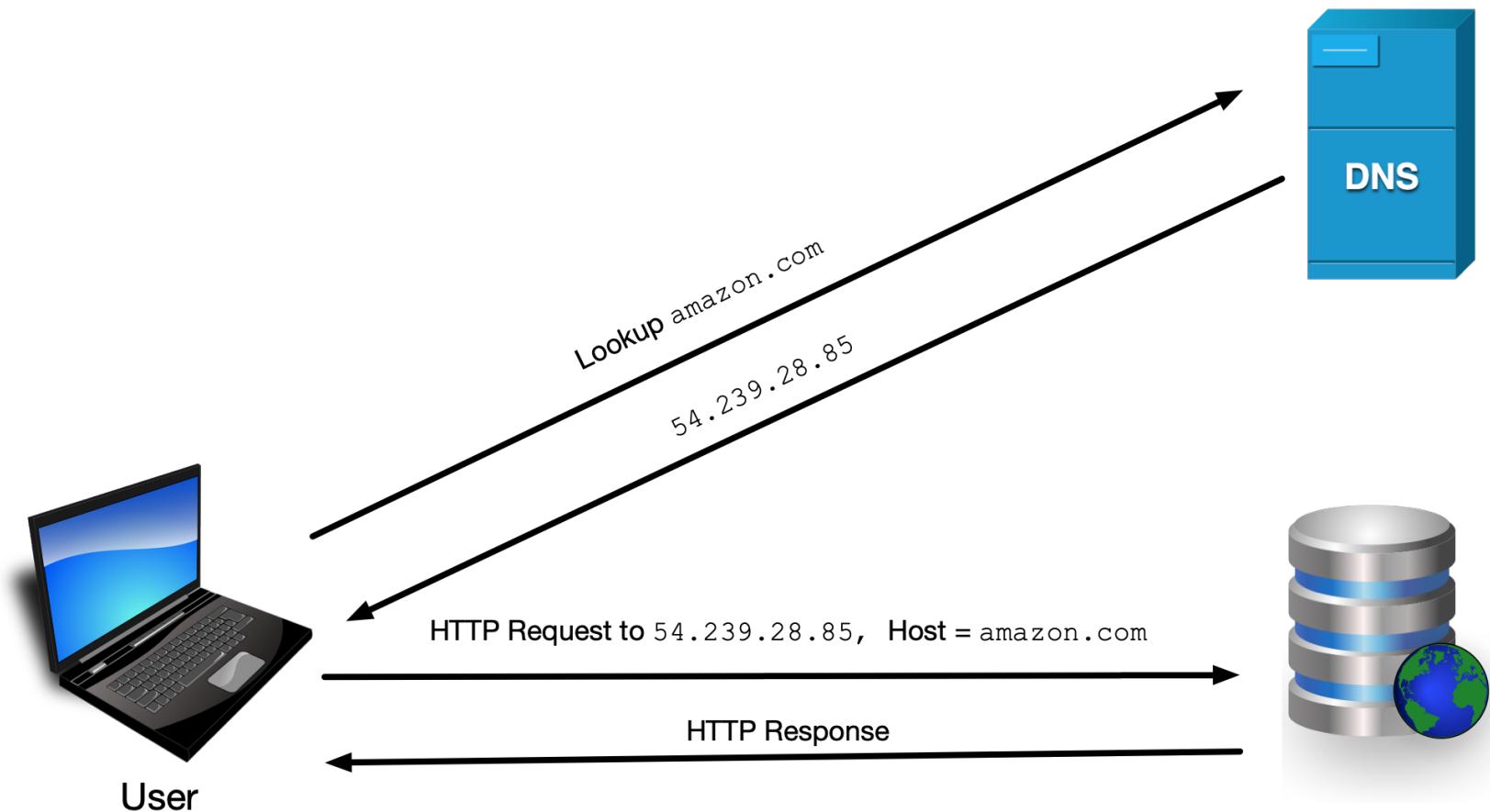
What Happens When We Visit amazon.com?



What Happens When We Visit amazon.com?



What Happens When We Visit amazon.com?



What Happens When We Visit amazon.com? (contd.)

The user enters `amazon.com` into the search bar of their browser.
Then:

- ➊ a request is made to the *Domain Name Service (DNS)* to get the IP address for `amazon.com`.
- ➋ DNS returns the IP address, maybe `54.239.28.85`.
- ➌ browser issues an HTTP request to `54.239.28.85`, host `amazon.com` and requests the home page `/`.
- ➍ the web server returns an HTTP response containing the home page (HTML) in the body.

But We Are Not Done...

The `amazon.com` home page (HTML) has lots of stuff on it: a video, an ad, styling (CSS) etc. We must fetch it.



Note that DNS queries must be made for each server, but we ignore them here.

But We Are Not Done...

The `amazon.com` home page (HTML) has lots of stuff on it: a video, an ad, styling (CSS) etc. We must fetch it.



Note that DNS queries must be made for each server, but we ignore them here.

But We Are Not Done...

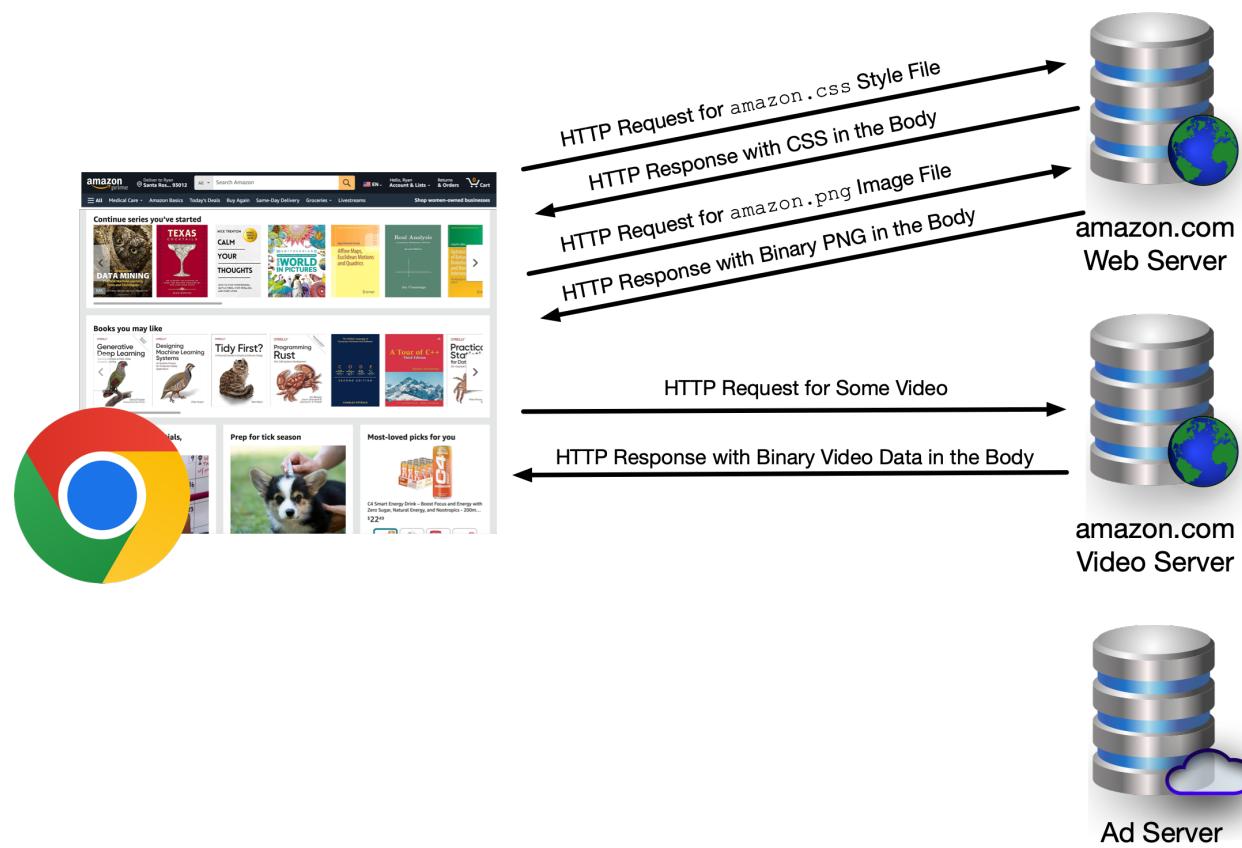
The `amazon.com` home page (HTML) has lots of stuff on it: a video, an ad, styling (CSS) etc. We must fetch it.



Note that DNS queries must be made for each server, but we ignore them here.

But We Are Not Done...

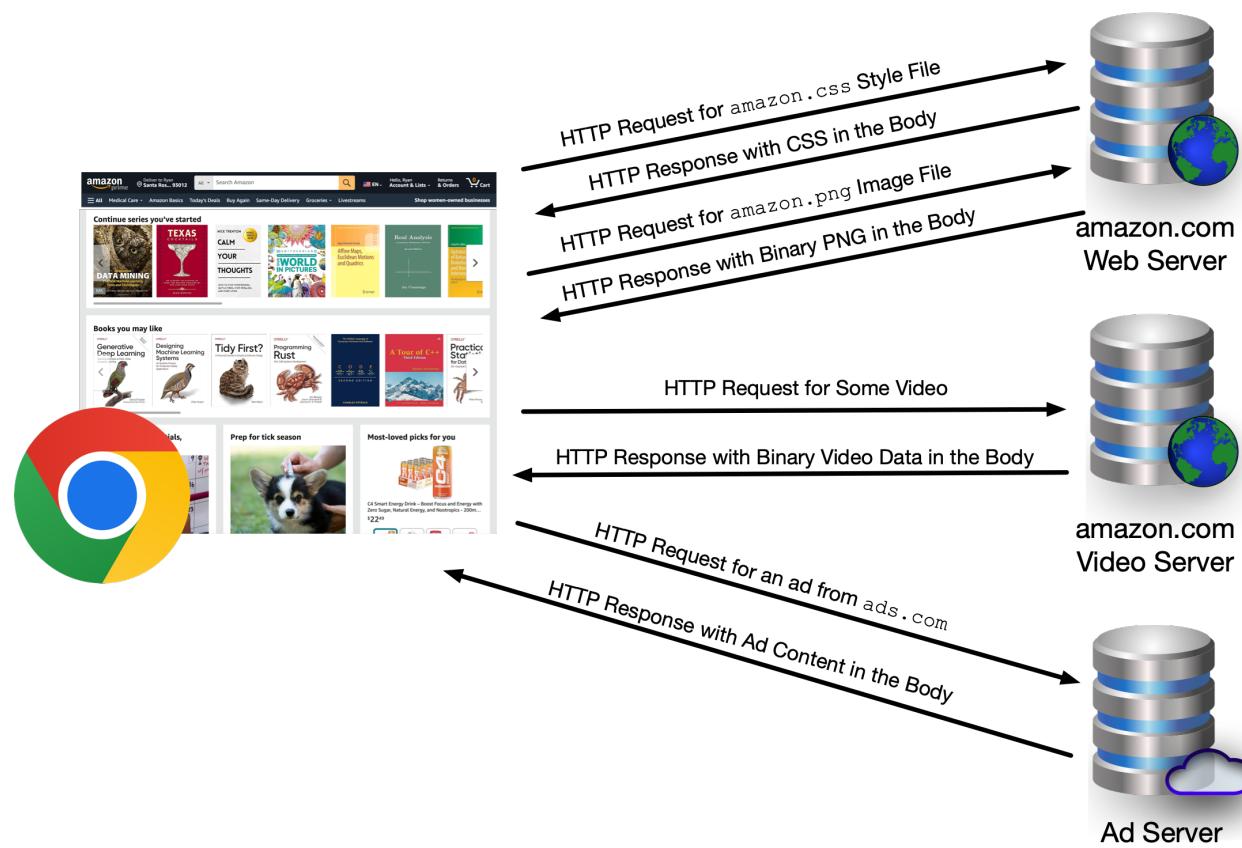
The `amazon.com` home page (HTML) has lots of stuff on it: a video, an ad, styling (CSS) etc. We must fetch it.



Note that DNS queries must be made for each server, but we ignore them here.

But We Are Not Done...

The `amazon.com` home page (HTML) has lots of stuff on it: a video, an ad, styling (CSS) etc. We must fetch it.



Note that DNS queries must be made for each server, but we ignore them here.

There's a Lot Going On...

- **DNS** looks up the IP address for various `amazon.com` subdomains.
 - ICANN manages top level domains.
- **HTTP** is the protocol over which web servers and web clients communicate.
- **Encoding Standards**
 - Text: ASCII, Unicode (UTF-8)
 - Multimedia: JPEG, PNG, H.264, H.265
- **HTML**: Hypertext Markup Language
 - How we represent the structure of a web page
- **CSS**: Cascading Style Sheets
 - How we format, style and present the content
- **Javascript**: De facto programming language
 - Server-side there are many including PHP, Ruby (Rails), PHP, Python (Flask etc.) and Java (Tomcat).

There's a Lot Going On... (contd.)

Each one of these has a special **MIME** file type name (Multipurpose Internet Mail Extensions).



Some MIME specifiers:

- `image/png`, `image/jpeg`, `image/gif`
- `multipart/form-data`
- `text/plain`, `text/css`, `text/html`

The Beginning

Tim Berners-Lee conceptualized what we know as the World Wide Web (WWW) while he was working at CERN (1989). It was originally called *The Mesh*. It had 4 components:



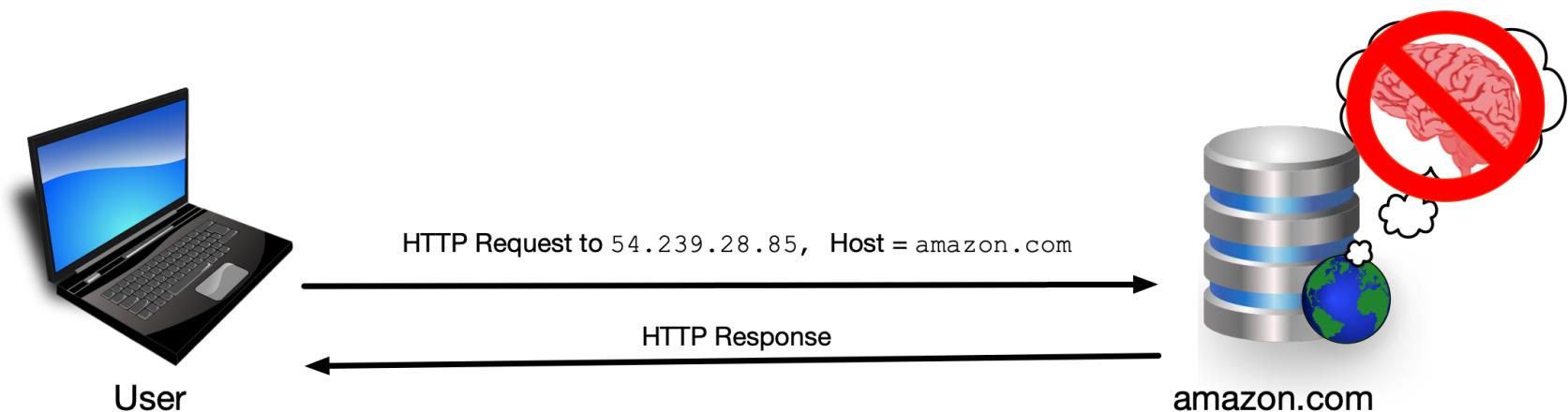
- ① a text format to represent hypertext documents (HTML)
- ② a simple protocol to exchange these messages/documents (HTTP)
- ③ a client to display and edit hypertext documents, now called a browser (it was called WorldWideWeb).
- ④ a server to host and serve hypertext documents (httpd).

The World Wide Web proposal was accepted in 1992

HTTP

HTTP stands for HyperText Transfer Protocol and is the main protocol for the WWW. It was originally created in 1990.

It was designed to be **stateless**. Once the web server sends a response to the client, it retains no memory of the exchange.



HTTP (contd.)

A client, or **user agent** can be a browser, or some other application. The HTTP standard defines it as one of the following:

- browser
- spiders and crawlers (web-based robots)
- command-line tools (e.g. wget, curl)
- custom applications
- mobile apps

The machine that serves as the server and the client can vary depending on the session.

HTTP/0.9 (1990)

An HTTP request was very simple

```
GET /mypage.html
```

The response was equally simple

```
<html>
<head></head>
<body>
```

The response was a raw HTML page with no other metadata

```
</body>
</html>
```

HTTP/1.0 (1996)

Version 1.0 was a major step forward but still not quite to where we are today:

- ➊ Added the concept of HTTP headers – metadata about each request and response.
- ➋ Added error status codes in responses (e.g. 404 Not Found)
- ➌ HTTP can now serve files other than HTML, including images, via the Content-Type header field.

HTTP/1.1 (1997)

Several improvements were made:

- HTTP 1.0 assumed a 1:1 correspondence between IP addresses and web servers. 1.1 added a Host field to allow virtual hosting.
- Added Keep-Alive support so clients could reuse an existing connection for multiple request/response pairs. In 1.0, the connection was closed after the response was sent.
- Cache controls were added. The server can tell the client how long they can keep copies of the web page.
- Content negotiation for language, encoding and type was introduced.

A Flurry of Activity

HTTP/1.1 was the prominent standard for a long time. Other technologies were developed in the 1990s-2010.

- 1994: Netscape developed secure implementations of HTTP with SSL 2.0/3.0. Eventually standardized as TLS.
- 2000: REST was developed to create web services for retrieving or modifying raw data.
- 2005: Server-sent events (occasional push) and WebSockets
- 2009: The Do Not Track DNT header field was added

HTTP/2 (2015)

Originally an experiment by Google called SPDY. Parts evolved into HTTP/2.

Purpose: Many objects needed to display a page, and mobile clients are high-latency/limited-bandwidth.

- Entirely binary (Saves space)
- Permits parallel unrelated requests over the same session, not just multiple requests
- Prioritization of certain streams. Requests of the same type are called “stream.”
- Headers are compressed. They are often repetitive (HPACK for security)
- Server can push metadata to a client (server push) that it thinks client will need.

Read more at <https://daniel.haxx.se/http2/>.

HTTP/3 (2022)

Designed to be faster than HTTP/2 and can be up to 4x as fast.

Relies on QUIC, which is a protocol on top of UDP, instead of TCP \implies quicker.

Supported by 97% of tracked web browser installation, 29% of the top 10M websites. Enabled in all major browsers except Safari.

It solves a niche problem that occurs in HTTP-over-TCP (v1.1 and v2).

HTTP/3 (2022) (contd.)

When an HTTP session is opened, we can pass multiple streams of requests and responses, including those are not related to each other but involve the same server.

When a packet is lost or reordered, regardless of which stream it belonged to, the entire connection will stall.

TCP cannot distinguish packets among the various streams – it is transport layer and HTTP is application level.

QUIC sits between application and transport layer and its multiplexing ability can localize transmissions issues to the affected stream.

HTTP Messages: Request and Response

An HTTP message is either a request or a response and have similar anatomy:

- ① Request (request) or status line (response)
- ② Header
- ③ (empty line)
- ④ content

HTTP Request: The Request Line

The request line has the format:

```
GET /apps/echo.py HTTP/1.1
```

Method Path Version

HTTP Request Methods

The first part of the request line is the method. There are several:

- GET: Retrieves data. Read only. No side effect on server
- POST: Post data to the specified URL. May have a side effect on server
- HEAD: Similar to GET, but only retrieve the header.
- PUT: “place” the data at the URL
 - Used for uploading or updating files at a URL
 - Modifying resources
 - Used frequently with REST APIs
- DELETE: delete the data at the URL
 - Typically used for REST APIs.

HTTP Request Methods (contd.)

Some lesser used ones:

- OPTIONS: Request the capabilities of the web server
- TRACE: The server echos back the received request. Used to trace what changes intermediaries made to the request.
- CONNECT: Sets up a tunnel through proxies for HTTPS. Also used for WebSockets
- PATCH: Makes a partial update to a resource rather than replacing (PUT) it.

Example HTTP Request

```
GET /index.html HTTP/1.1
Host: ryanrosario.phd
Connection: keep-alive
Cache-Control: max-age=0
Dnt: 1
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/13.1.3 Safari/605.1.15
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: iwe_term_student=24S;
```

HTTP Response

An HTTP response has the following anatomy:

- Status line
- Header
- (empty line)
- Body

HTTP Response: Status Line

The status line looks as follows:

HTTP/1.1 200 OK

Of the form:

Version StatusCode Description

HTTP Response: Status Codes

Each response contains a status code:

- 1xx: Informational (rarely used)
- 2xx: Success (200 OK is the most common and 204 No Content)
- 3xx: Redirection (target URL is returned in response)
- 4xx: Client Error:
 - 404 — Not Found
 - 400 — Bad Request (malformed request)
 - 401 — Unauthorized (need to be authenticated)
 - 403 — Forbidden (permission denied)
 - 418 — I'm a Teapot (an April Fool's Joke in 1998/2014)
 - 429 — Too Many Requests (rate-limited)
- 5xx: Server Error
 - 500 - Internal Server Error
 - 502 - Bad Gateway

Example HTTP Response

```
HTTP/1.1 200 OK
Server: nginx
Date: Thu, 28 Mar 2024 06:45:30 GMT
Content-Type: text/html
Content-Length: 6961
Last-Modified: Sat, 30 Sep 2023 08:00:08 GMT
Connection: keep-alive
ETag: "6517d588-1b31"
Accept-Ranges: bytes

<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="utf-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>Ryan R. Rosario, Ph.D.</title>
  ...
</html>
```

Demo: HTTP/0.9

telnet ryanrosario.phd 80

At the prompt enter:

GET /

What do you notice? Anything weird?

Demo: HTTP/1.1

```
telnet ryanrosario.phd 80
```

At the prompt enter:

```
GET / HTTP/1.1
```

```
Host: ryanrosario.phd
```

Hit enter twice. What do you notice?

Recursive DNS

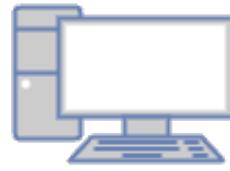
- **HTTP/0.9.** No headers, simple request, only HTML response.
- **HTTP/1.0.** Headers, error codes, more than HTML.
- **HTTP/1.1.** Virtual hosting, keep-alive, cache control, content negotiation.
 - Democratized access to web hosting, particularly *cheaper* web hosting. This was huge.
- **HTTP/2.** Performance improvements via compression and binary representation of messages
- **HTTP/3,** Additional performance improvements in HTTP session by replacing TCP with QUIC on top of UDP.

HTTP and Intermediary Servers

To start off, we assumed that a user agent communicates directly with a web (origin) server. This is not always how it works. There can be **intermediary** servers between the user agent and the web server hosting the desired content.

An intermediary can be an HTTP proxy server (e.g. caching) or gateway (e.g. load balancing, authentication/access control, DDOS protection/security) or tunnel.

HTTP and Intermediary Servers (contd.)



User Agent
Browser, Bot/Spider, App, API

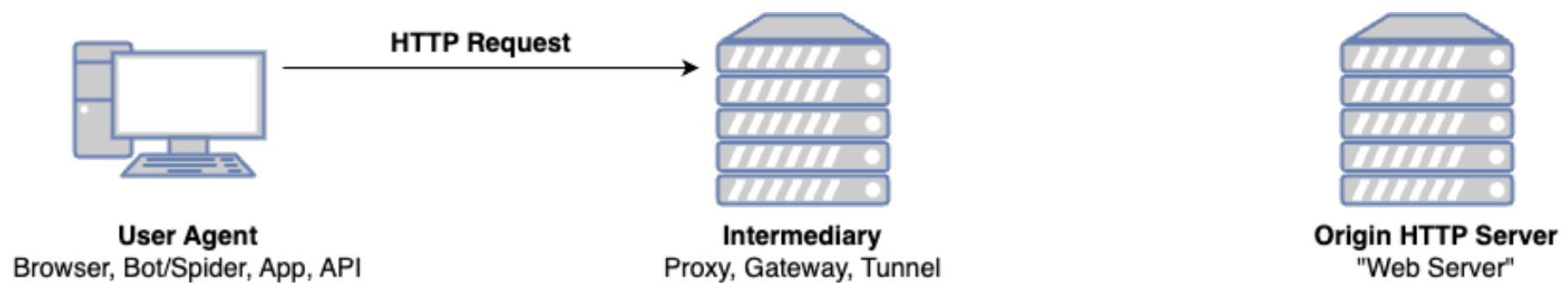


Intermediary
Proxy, Gateway, Tunnel

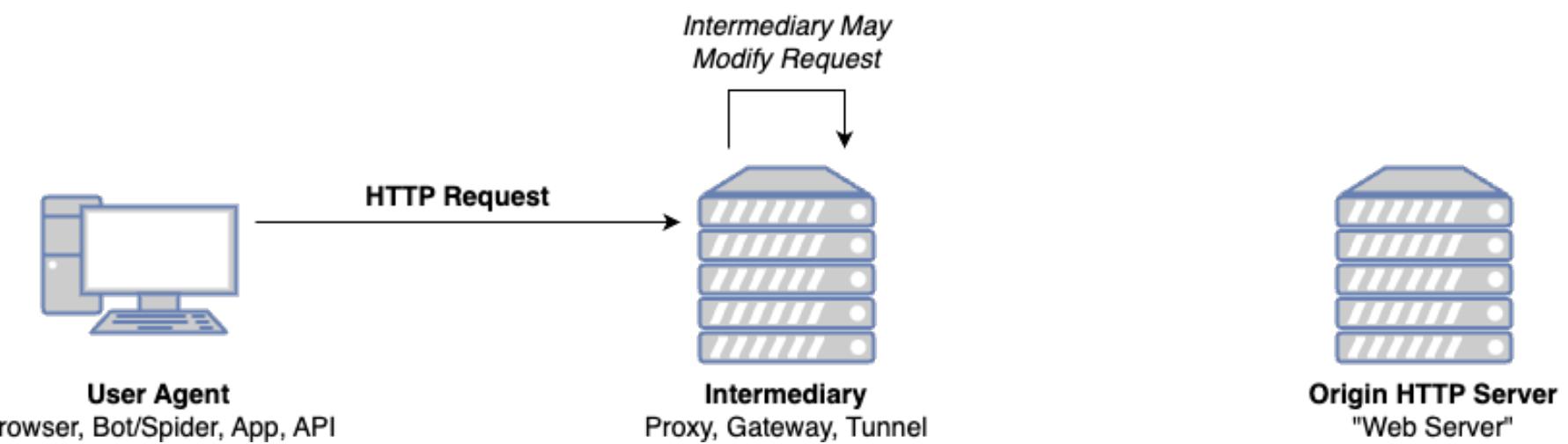


Origin HTTP Server
"Web Server"

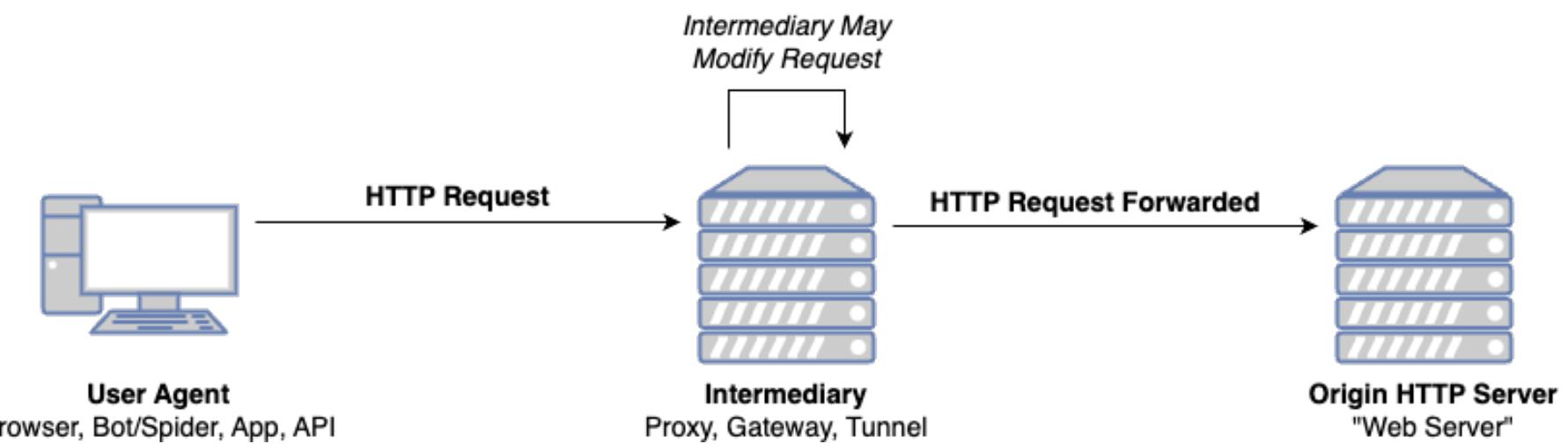
HTTP and Intermediary Servers (contd.)



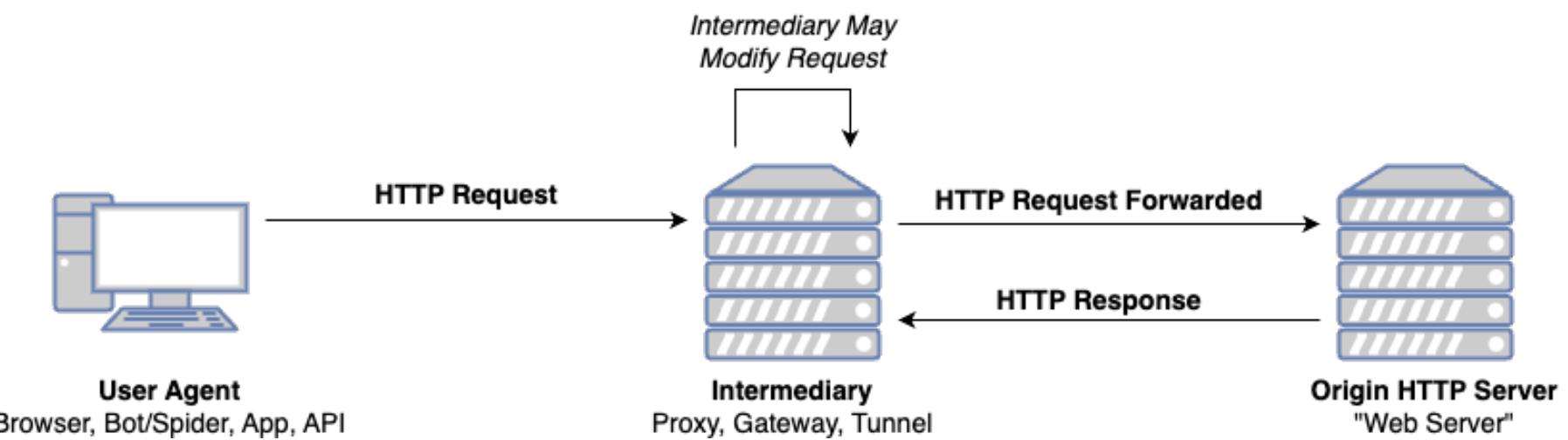
HTTP and Intermediary Servers (contd.)



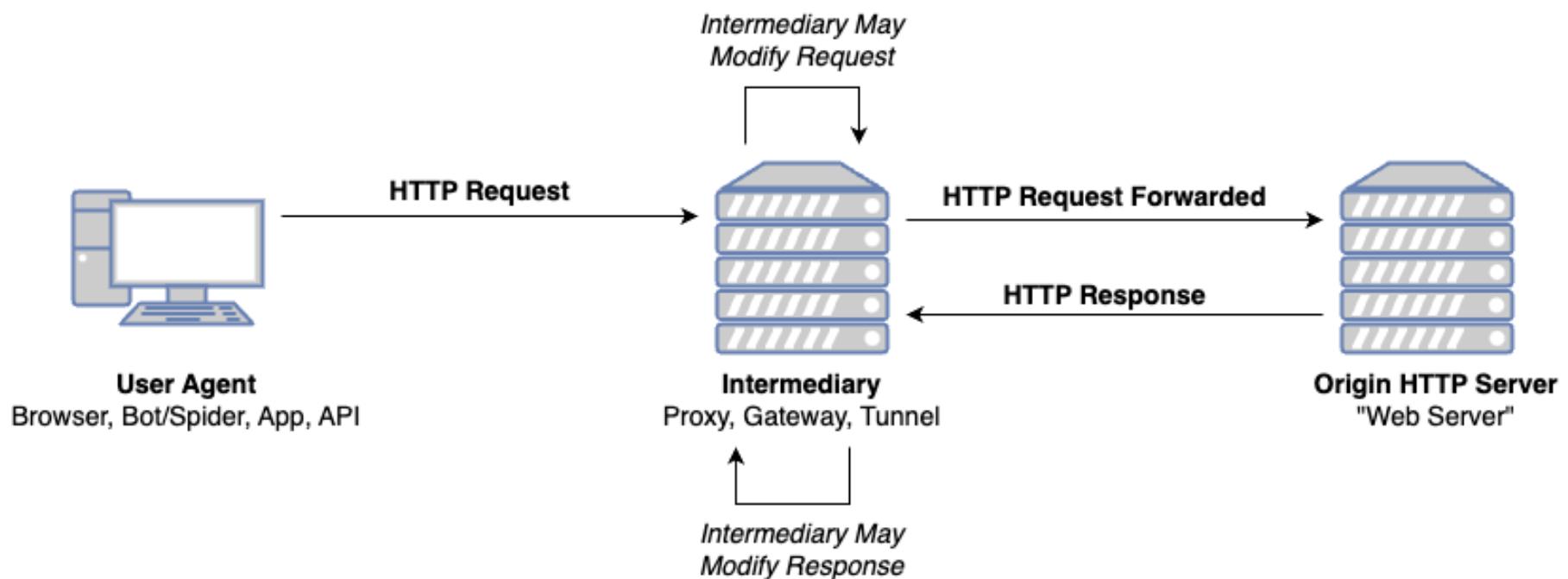
HTTP and Intermediary Servers (contd.)



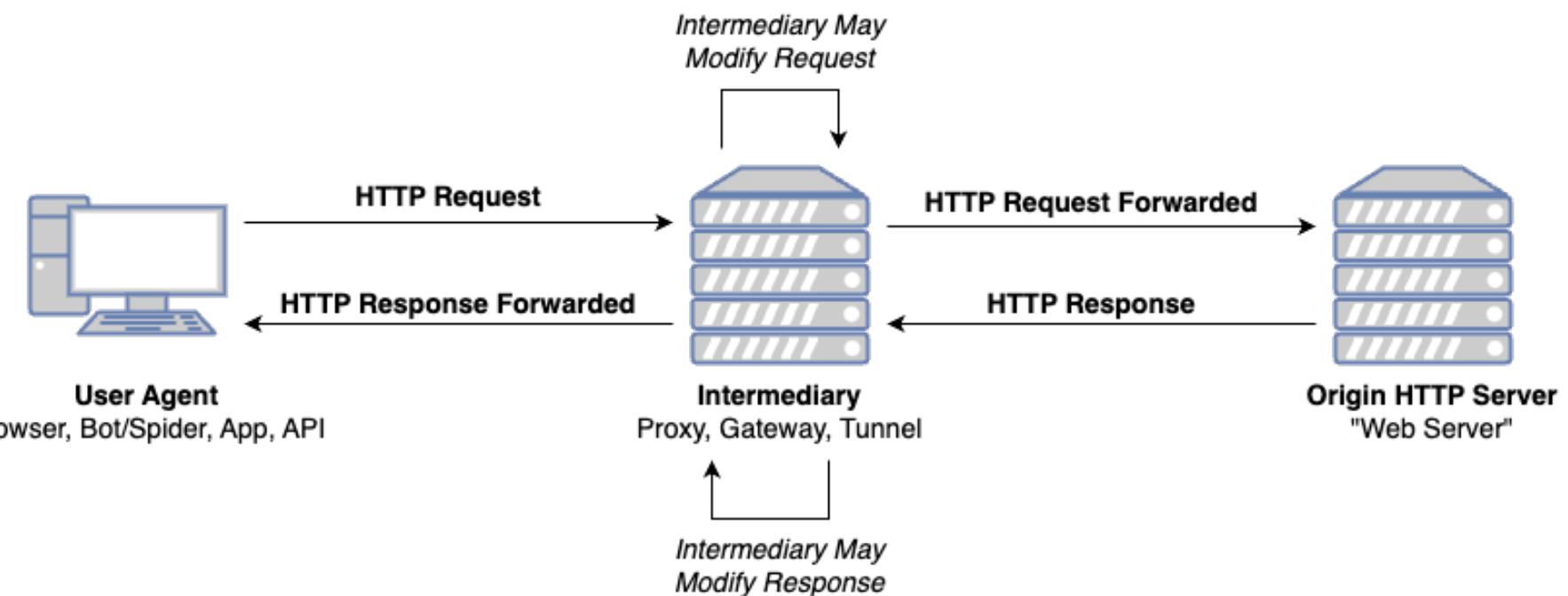
HTTP and Intermediary Servers (contd.)



HTTP and Intermediary Servers (contd.)



HTTP and Intermediary Servers (contd.)



HTTP and Intermediary Servers (contd.)

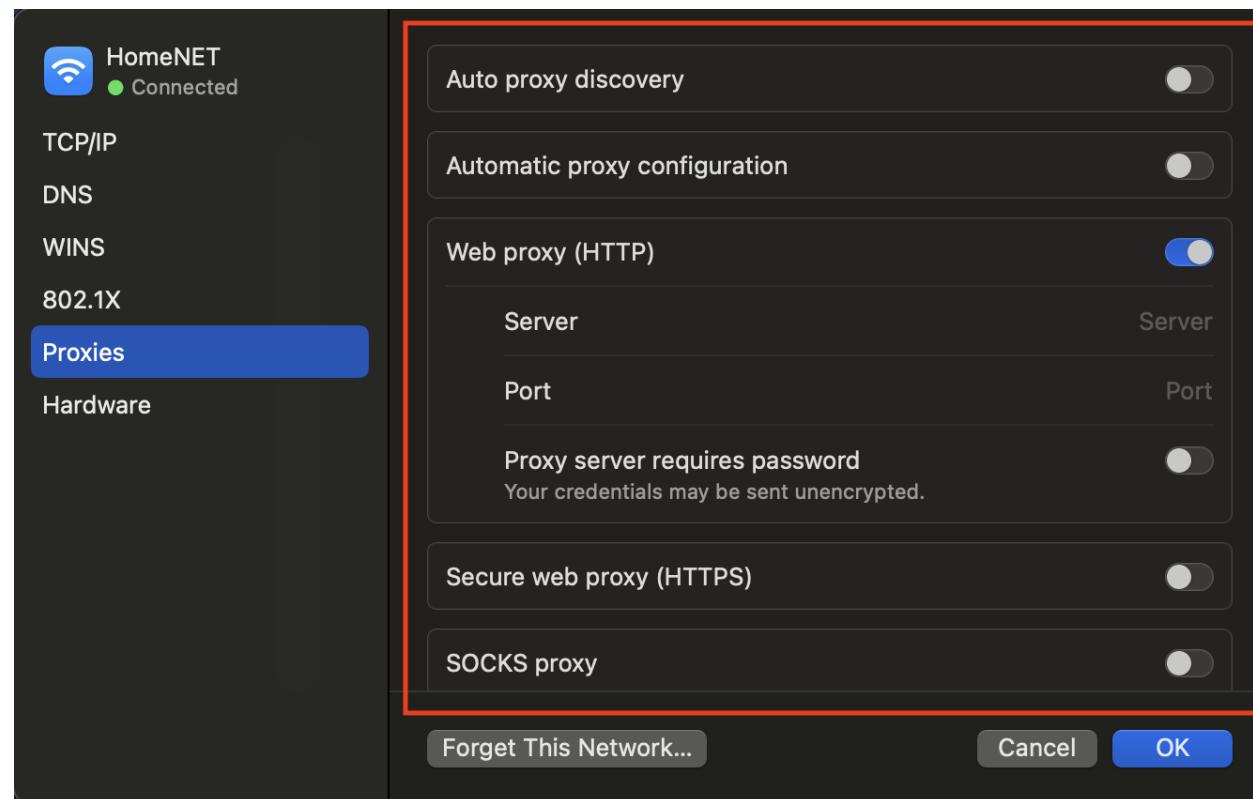
So, in the previous example:

- The user agent sends the HTTP request to the intermediary.
- The intermediary processes the HTTP request and may modify it if necessary.
- The intermediary sends the modified HTTP request to the web server (or to another intermediary if applicable) on the UA's behalf.
- The web server processes the modified HTTP request, and sends an HTTP response to the intermediary.
- The intermediary processes the HTTP response and may modify it if necessary.
- The intermediary delivers the possibly modified HTTP response to the user agent.

This is called a **chained request/response**.

HTTP and Intermediary Servers (contd.)

An HTTP proxy server is a message forwarding agent that is chosen by the client.



HTTP and Intermediary Servers (contd.)

With a **transparent proxy** (sometimes called a *captive portal*), the client does not choose the proxy. It simply intercepts traffic and forwards messages.

This is really nothing more complicated than a “man-in-the-middle” attack, and can have security ramifications.



Cloudflare is an example of a transparent proxy — one that has good intentions.

References

- [RFC1945 \(HTTP/1.0\)](#)
- [RFC7230-7237 \(HTTP/1.1\)](#)
- [RFC7540 \(HTTP/2\)](#)
- [RFC9114 \(HTTP/3\)](#)

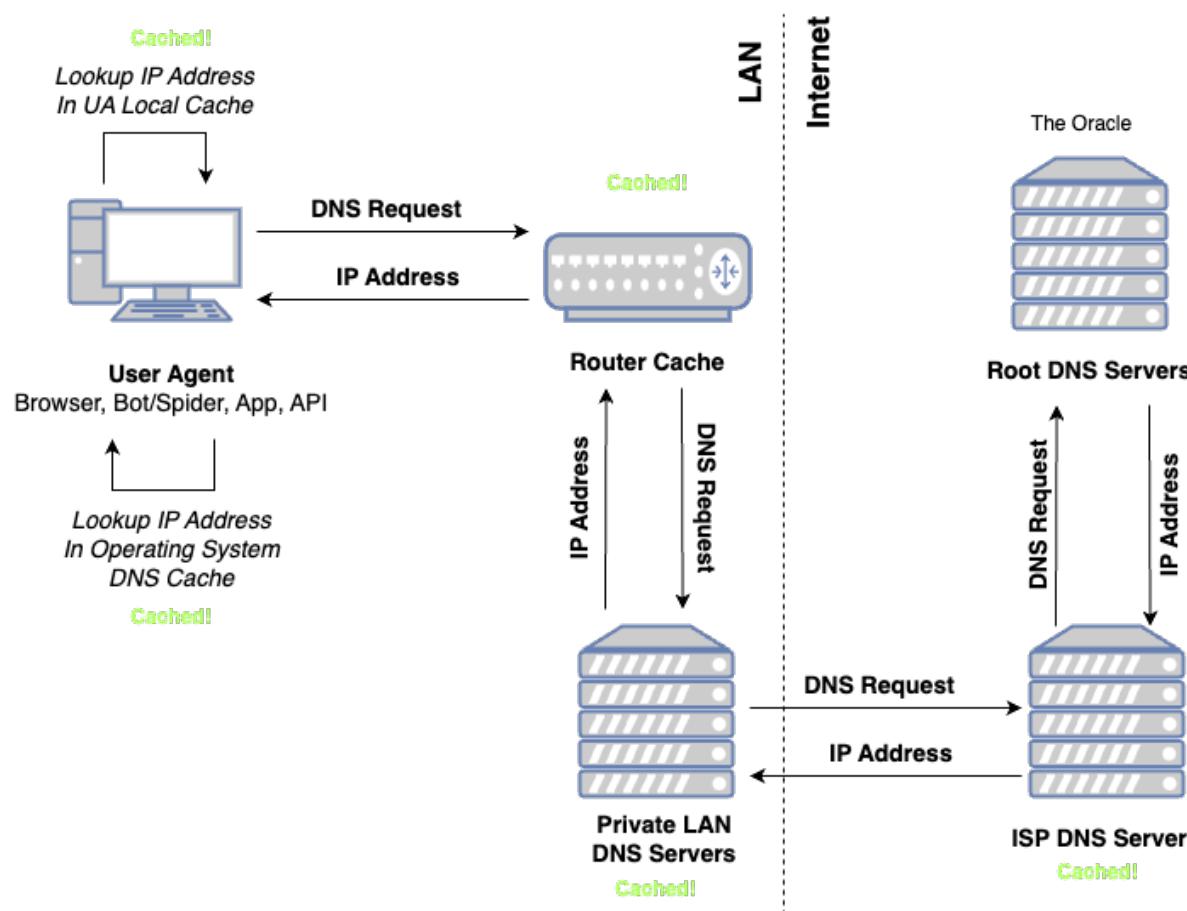
DNS

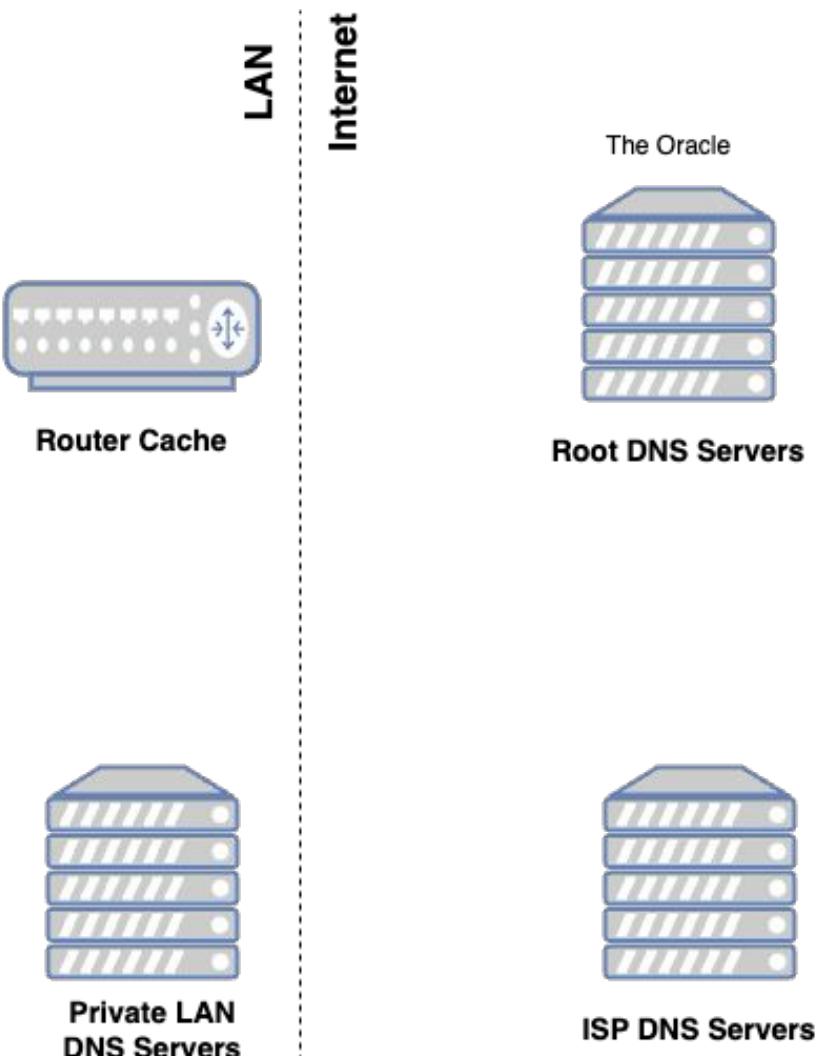
DNS lookups can be *recursive* or *iterative* and quite brilliant in its implementation.

It uses **caching** heavily.

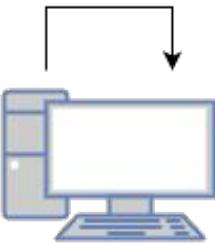
Brief Discussion of DNS

DNS (contd.)





*Lookup IP Address
In UA Local Cache*



User Agent
Browser, Bot/Spider, App, API



LAN
Internet

The Oracle



Root DNS Servers



**Private LAN
DNS Servers**



ISP DNS Servers

