WEEK 6: WEB APPS

https://github.com/entr451-winter2026/web-apps

AI + CODE

PSA

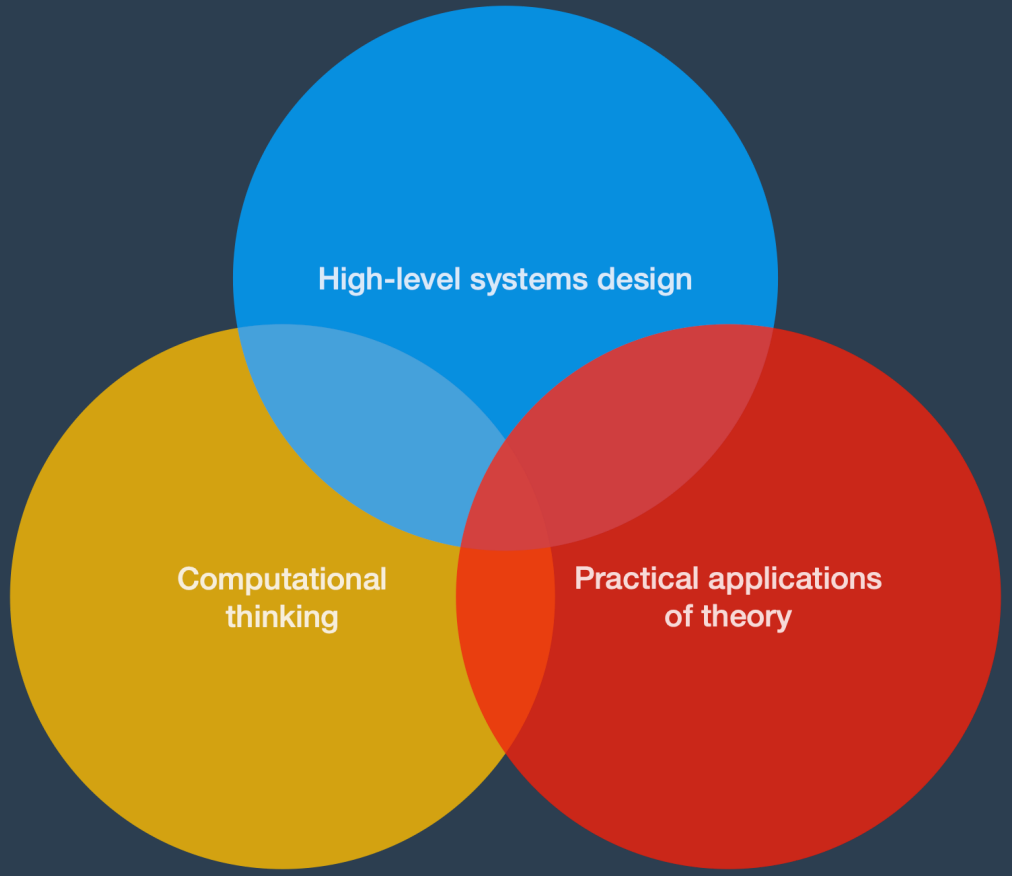
https://1password.com/blog/from-magic-to-malware-how-openclaws-agent-skills-become-an-attack-surface

RETROSPECTIVE

RETROSPECTIVE

0 TO _ ?

RETROSPECTIVE



RETROSPECTIVE

- ✅ Practical applications
  - Basic tools – source code control (Git), Linux command line, code editor
  - Querying and manipulating data (SQL)
  - Ruby
  - Ruby to do the work of SQL
- ✅ Computational thinking
  - Database design/domain modeling
- Systems design
  - Today!

~~30~~ 15 HOURS

SYSTEMS DESIGN

HISTORY LESSON!

- The Internet (at least the consumer Internet as we know it today) started in the 1990s □
- Most people/companies didn't yet have a website
- The ones who did (nerds) – it was just .html files on a hard drive

INFO.CERN.CH IN 1992

World Wide Web

The WorldWideWeb (W3) is a wide-area hypermedia information retrieval initiative aiming to give universal access to a large universe of documents.

Everything there is online about W3 is linked directly or indirectly to this document, including an executive summary of the project, Mailing lists , Policy , November's W3 news , Frequently Asked Questions .

What's out there?
: Pointers to the world's online information, subjects , W3 servers, etc.

Help
: on the browser you are using

Software Products
: A list of W3 project components and their current state. (e.g. Line Mode ,X11 Viola , NeXTStep , Servers , Tools , Mail robot , Library)

Technical
: Details of protocols, formats, program internals etc

Bibliography
: Paper documentation on W3 and references.

People
: A list of some people involved in the project.

History
: A summary of the history of the project.

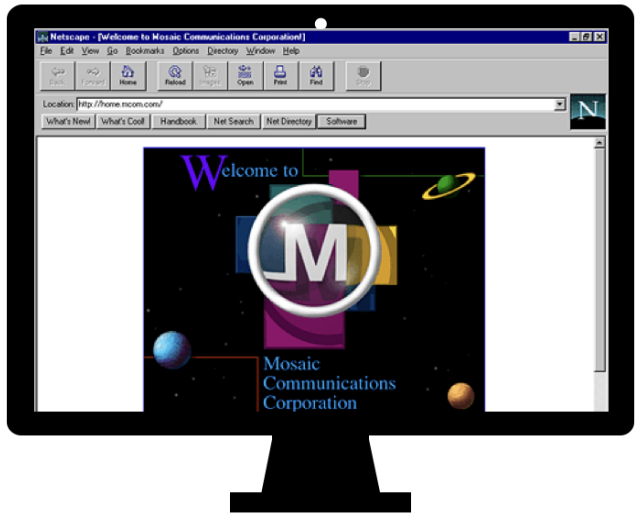
How can I help ?
: If you would like to support the web..

Getting code
: Getting the code by anonymous FTP , etc.

THE LIFECYCLE OF A WEBSITE, CIRCA 1990S

THE LIFECYCLE OF A WEBSITE, CIRCA 1990S

- Dial into the Internet
- Open Netscape Navigator, enter web address – there was no search! 🤯
- Let the Internet do the work of reaching out to the file server where the web address pointed to, get back some HTML, and return it to the browser, which turns it into something I'm able to see



User's Computer

index.html, please?

Internet

here you go!

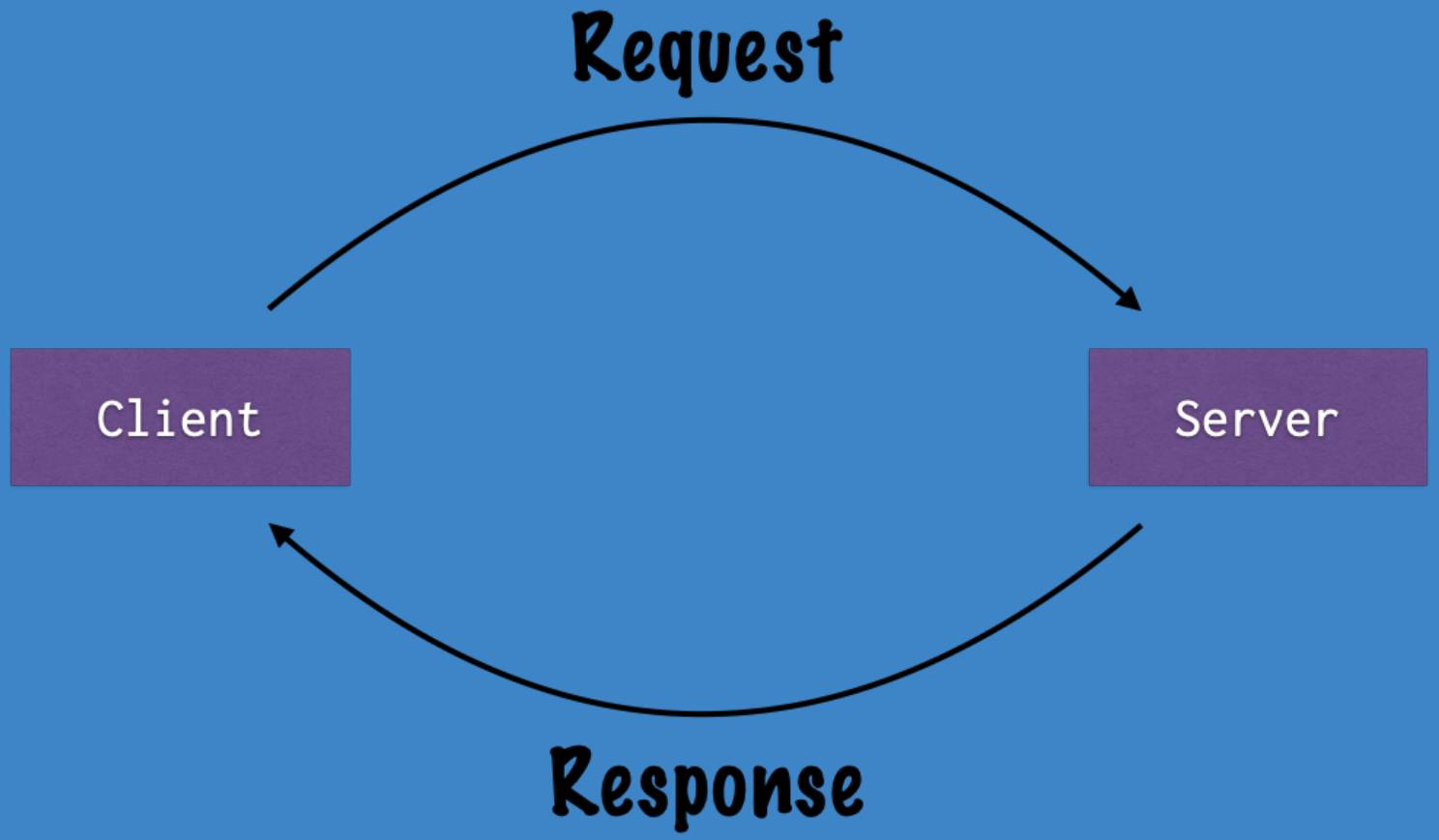
Server aka giant hard drive

ORDERING A MEAL (REQUEST)

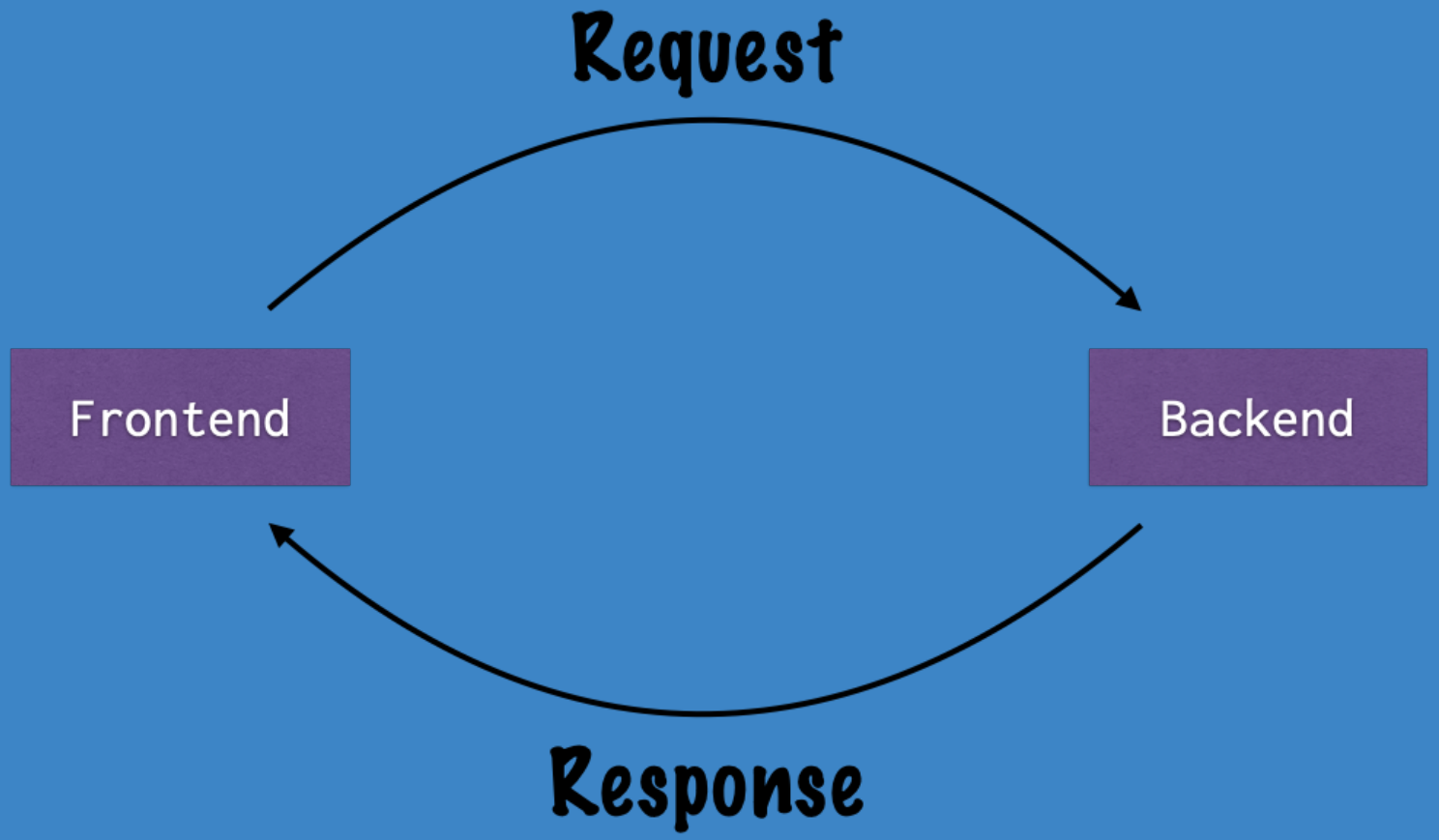
KITCHEN PREP (SERVER LOGIC)

MEAL IS SERVED (RESPONSE)

REQUEST/RESPONSE LIFECYCLE



REQUEST/RESPONSE LIFECYCLE



THE INTERNET...

- Publishing tool
- Content providers maintained a set of files
- Perfect for:
  - Company websites
  - Libraries/research
  - News sources
  - Any publications where the information *hardly ever changed*

AND THIS WAS PERFECT...

- HTML
- Not too difficult to learn
- Had images, formatting
- *Could link to other resources on the Internet*

EVOLUTION: DYNAMIC CONTENT

DYNAMIC CONTENT

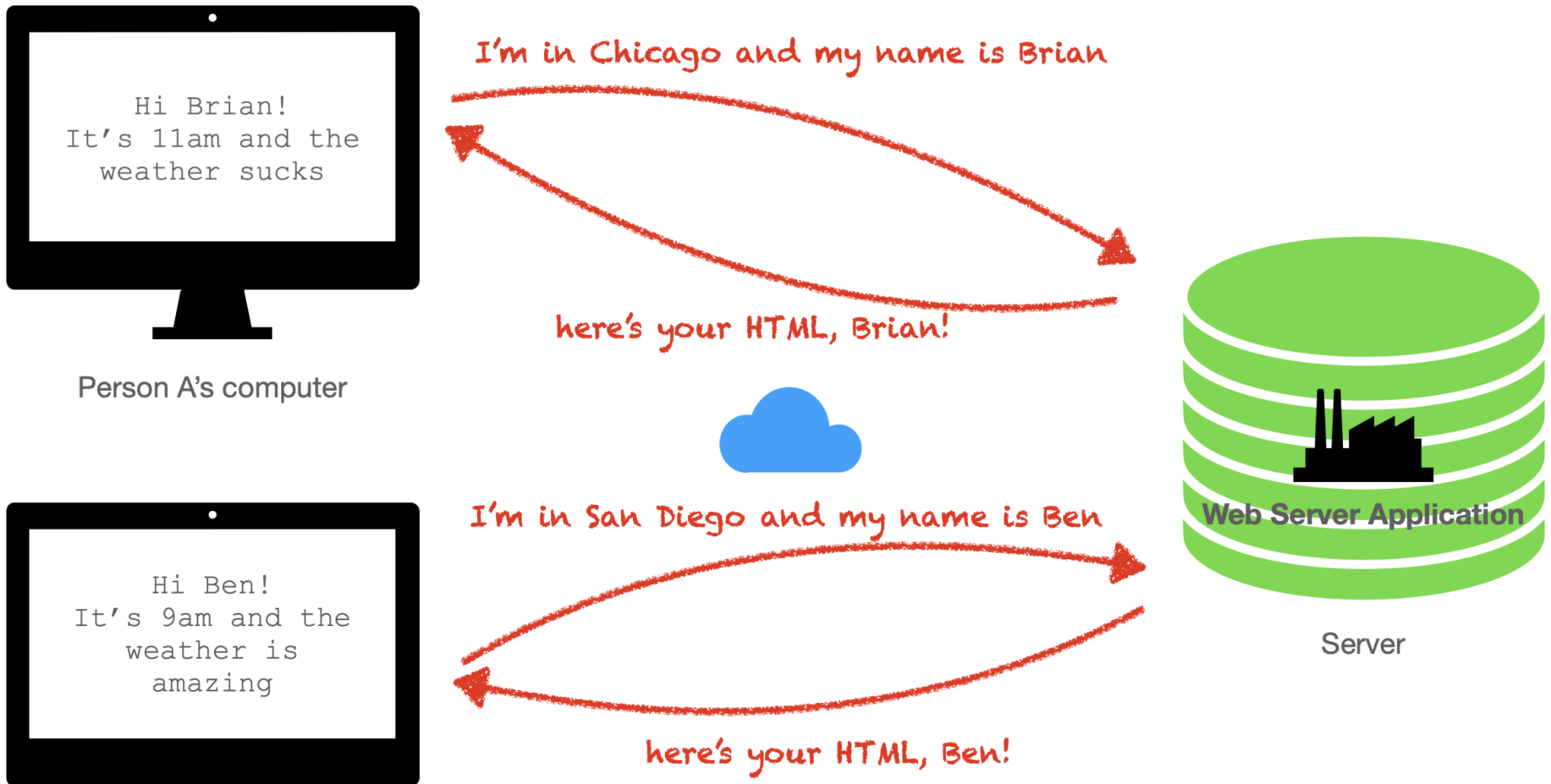
- What if two people viewing a web page could NOT see the same thing? Imagine if...
  - Person A and Person B could be greeted each day by seeing their own respective names on the top of the page?
  - Person A could see a different date and time on the top of the page than Person B, because they live in two different time zones?
  - Person A and Person B could see some different, personalized content, like the current weather where they are?

DYNAMIC CONTENT

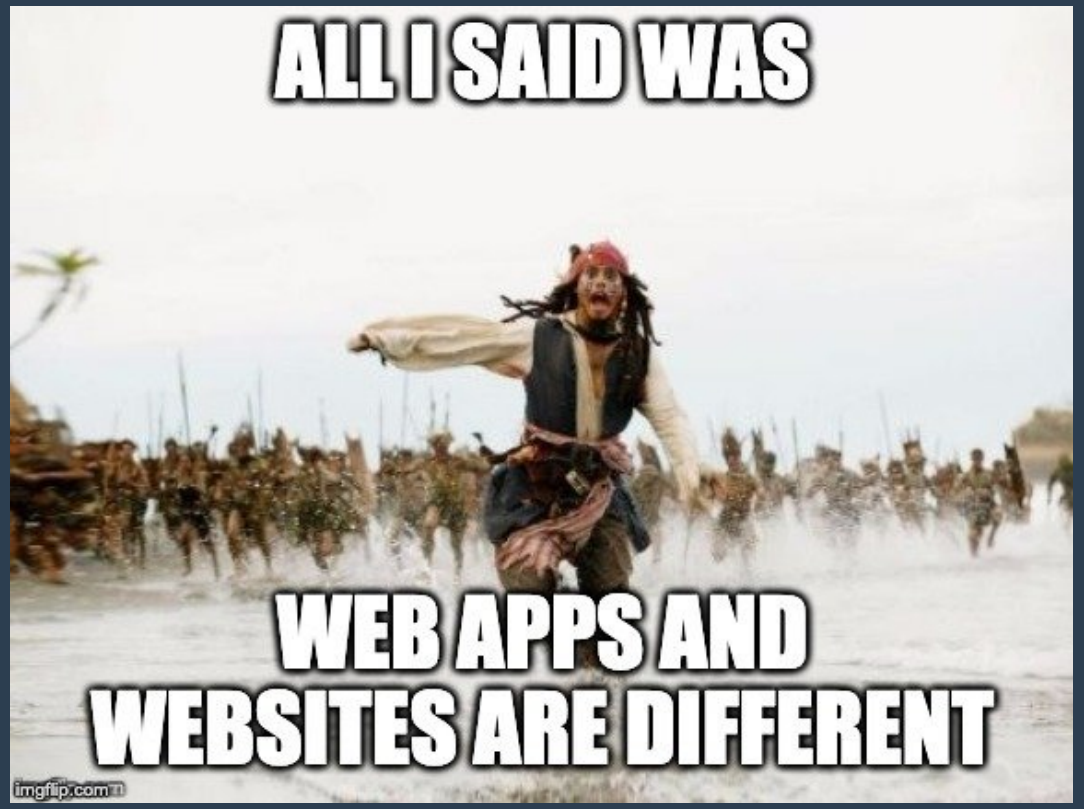
- Can't be done with HTML – HTML is static
- We'd need a way to give each person *different HTML*
- The answer was to have the web server do just that – generate different HTML for every person viewing the web page
- Server applications
  - CGI
  - Perl
  - PHP

DYNAMIC CONTENT

- These applications interpreted the user's **request**
- And generated a dynamic HTML **response**

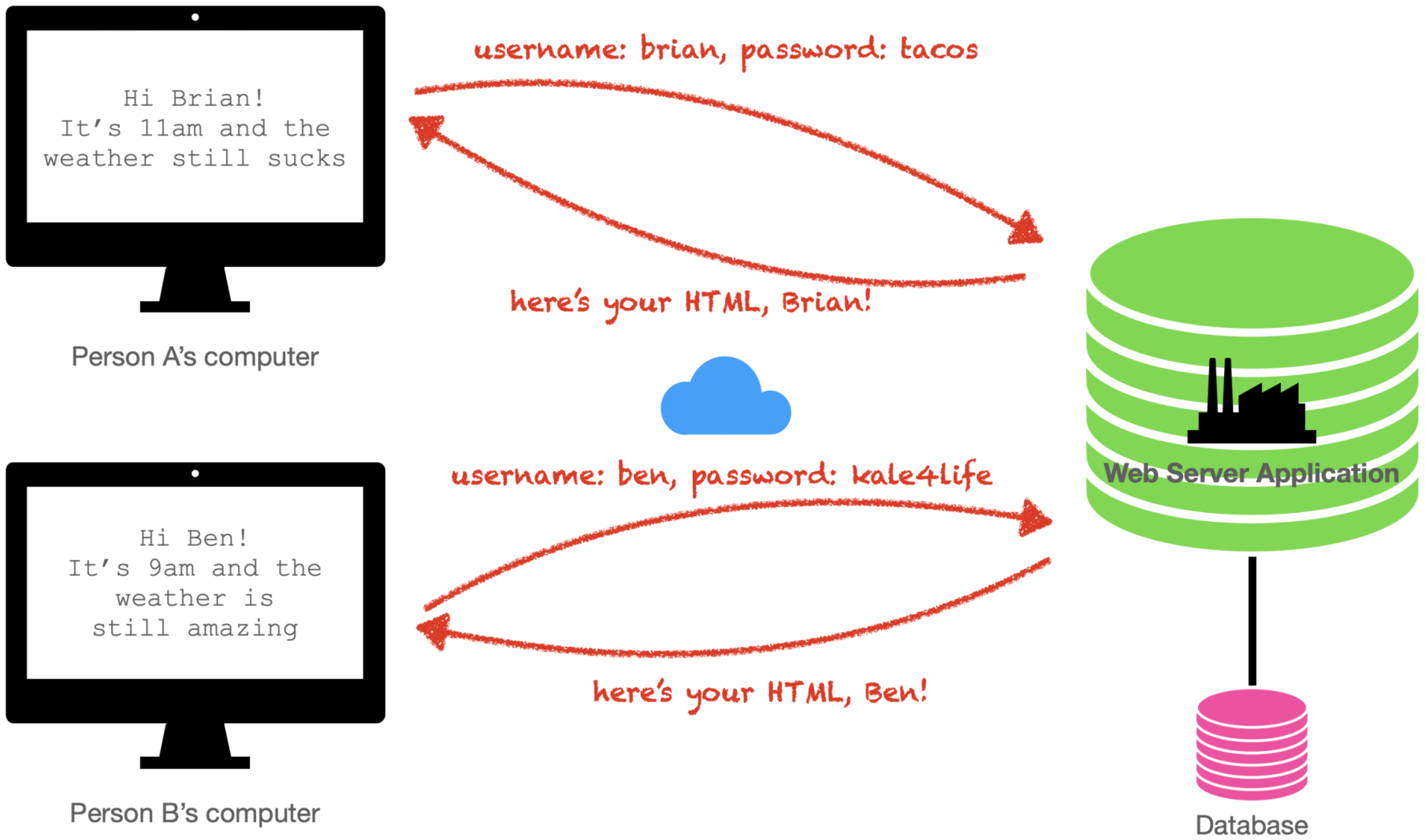

Hi Brian!
It's 11am and the weather sucks
I'm in Chicago and my name is Brian
here's your HTML, Brian!
Person A's computer
Hi Ben!
It's 9am and the weather is amazing
I'm in San Diego and my name is Ben
here's your HTML, Ben!
Person B's computer
Web Server Application
Server

EVOLUTION: DATA-DRIVEN DYNAMIC WEB APPLICATIONS



DATA-DRIVEN DYNAMIC WEB APPLICATIONS

- Add a database... next level!
- Instead of the user sending information, we already have it
- All we need is a username/password


Hi Brian!
It's 11am and the
weather still sucks
Person A's computer
username: brian, password: tacos
here's your HTML, Brian!
Hi Ben!
It's 9am and the
weather is
still amazing
Person B's computer
username: ben, password: kale4life
here's your HTML, Ben!
Web Server Application
Database

THE STORY

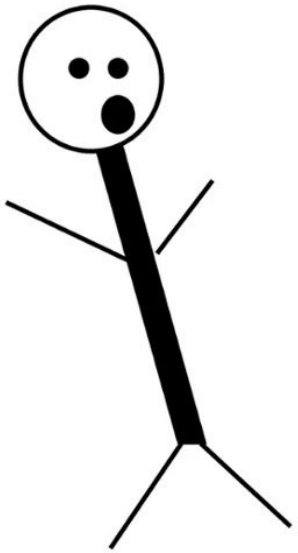
- An end-user visits your application
  - i.e. they go to *https://YourDomainName.com/things*
- That **request** is received by your web application, and it figures out what the appropriate response is (i.e. what HTML to send)
- The **response** is sent back over the wire and is received by the web browser, which then shows it to the end-user

URL
(UNIFORM RESOURCE LOCATOR)

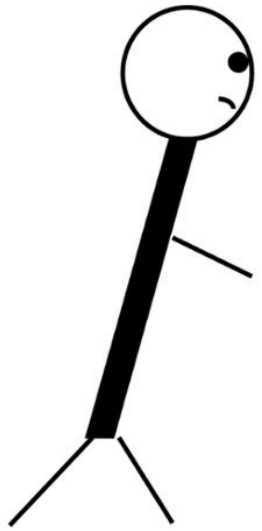
```
https://YourDomainName.com/books
https://YourDomainName.com/shoes
```

- Locator – provides a location for a unique result
- Resource – books is the resource (e.g. a page about books and another page about shoes)
- Uniform – they all look the same:
  - The protocol – https or http
  - :// ("colon-slash-slash")
  - Host name – *YourDomainName.com*
  - Path to resource – */books* or */shoes*

PSA



Backslash



Forward Slash

JUMP IN

https://github.com/entr451-winter2026/web-apps

LAB - DICE

- Rebuild the *dice* app in the browser
  - Create a resource (in *config/routes.rb*)
  - Create a controller

```
rails generate controller _____
```

  - Create an "index" view (in *views/dice*)
  - In the view:
    - "Roll" 2 dice (i.e. random number between 1-6)

```
rand(1..6)
```

    - Display the dice values and total
      - Challenge: use images

```
<img src="/images/dice/1.svg">
```

    - Add link to roll again

entr451.com/programming-fundamentals-lab-1-solution

LAB - CARDS

- Similar to dice, rebuild the *cards* app from Week 4/Lab 5:
  - https://entr451.com/programming-fundamentals-lab-5-solution/
- Deal a 5-card hand of poker in the browser
- Images are located in */public/images/cards*

```
ranks = [2, 3, 4, 5, 6, 7, 8, 9, 10, "jack", "queen", "king", "ace"]
suits = ["clubs", "diamonds", "hearts", "spades"]
```

FAST-FORWARD 30 YEARS

- Technical stacks have evolved...
  - But the goal: still the same – show each user their own HTML
- We're still using HTML!
- And some other stuff:
  - CSS – Cascading Style Sheets
  - JavaScript – a programming language – the only language that is natively understood by major browsers and is capable of manipulating HTML
- We still have server applications

TECHNICAL STACK

DATABASE
(sql)

BACK-END
(server)

FRONT-END
(html, css, js)

THE CURRENT STATE OF WEB DEVELOPMENT

- LOTS of choices
- Very fragmented
- But can be distilled into:
  - Server-side rendered (SSR) applications
  - Single page applications (SPA)

LET'S MODERNIZE

DATA IN VIEWS

BASIC TOOLS

+ SQL

+ RUBY

+ RUBY ORM (ACTIVERECORD)

+ WEB

= "YOUR OWN APPS" 🎉

LAB - CONTACTS

- Repeat companies for contacts
  - Create a resource
  - Create a controller
  - Create an "index" view
  - In the view:
    - Loop through all contacts
    - Display:
      - Contact's full name
      - Name of the contact's company

PRACTICE

- Posted in Canvas

NEXT WEEK

- "C" in CRUD (creating data)
- Design patterns (REST)