

everything you always
wanted to know about SE*
but were afraid to ask
(*software engineering of web apps)

Daniel Jackson

“Everything
you always
wanted to know
about sex*

* BUT WERE AFRAID TO ASK”



your goals for today's class

understand some basic notions

client/server systems, APIs, authentication

fill in gaps in your understanding of web apps

HTTP protocol, asynchronous JS

hold your own at cocktail parties

be able to talk REST, Jekyll, local first

what's
client-server?

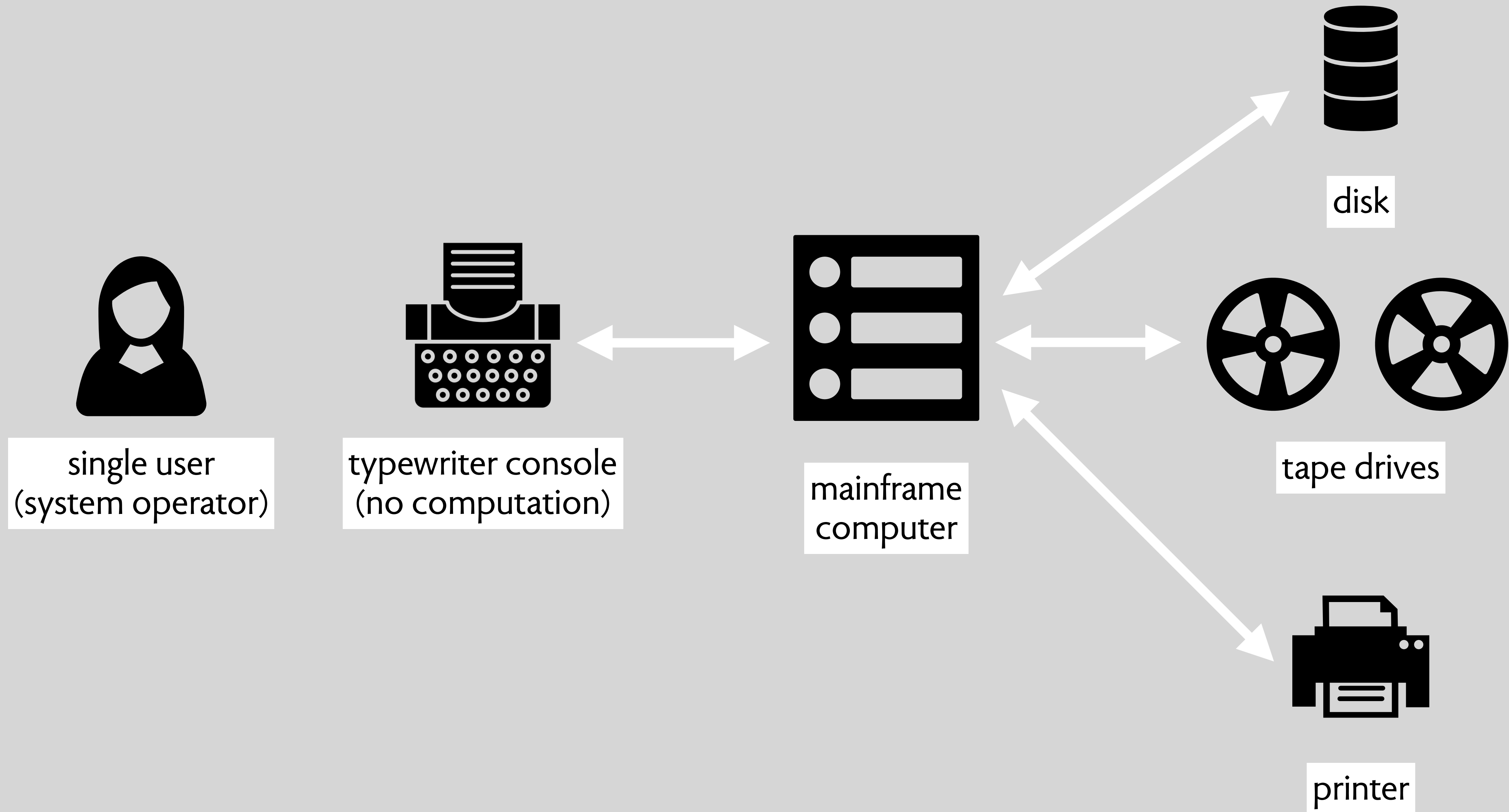
mainframe computer



no real client
console is wired in
part of machine

Univac 1 at Census Bureau: first commercial computer for civilian government agency (1951)

early mainframe architecture (1962)



the invention of time sharing



Christopher Strachey in 1936
patented time sharing (1959)
denotational semantics (1966)



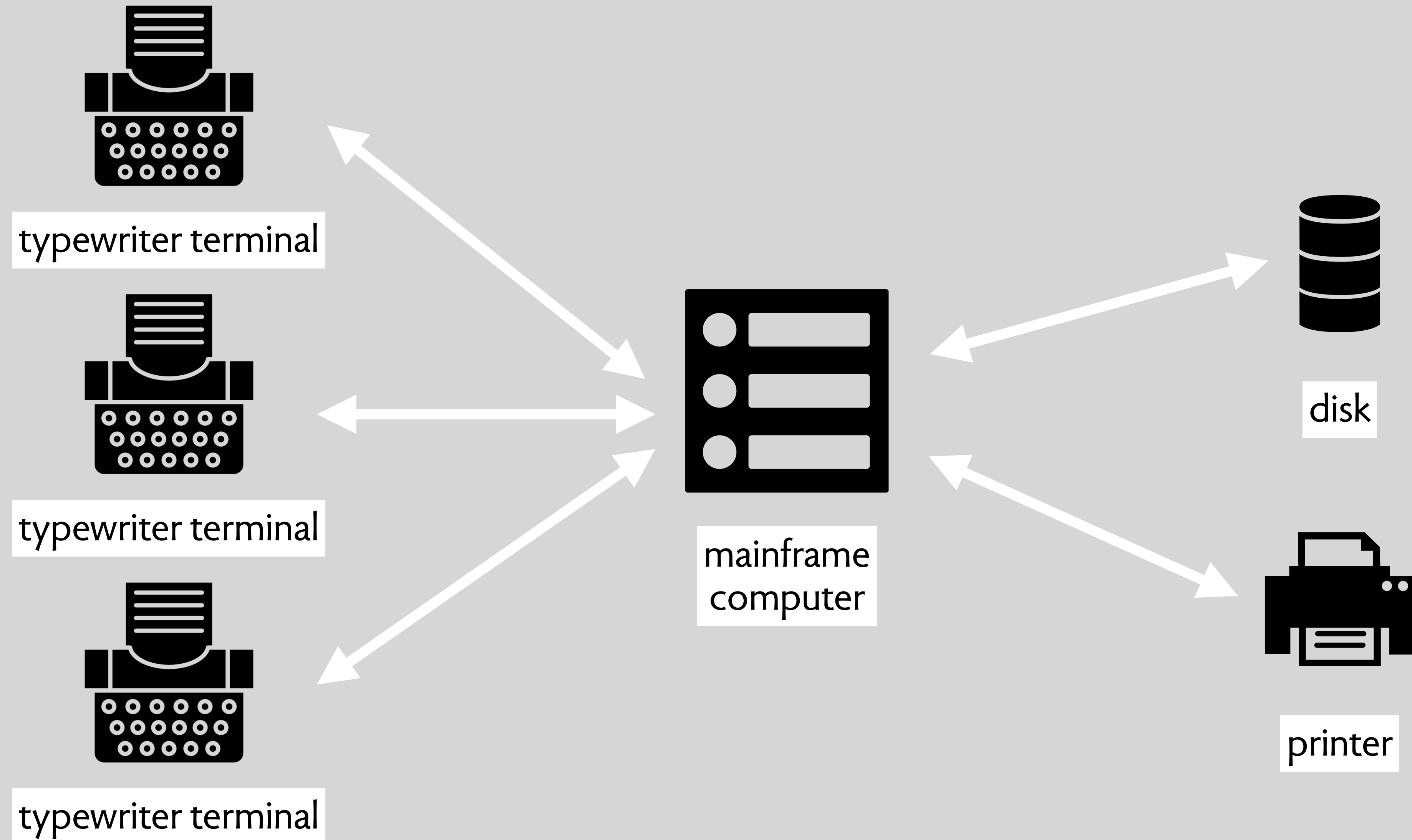
my father's high school teacher
at Harrow School in London
wrote checkers program (1951)
with help from Alan Turing

timesharing at MIT: Project Mac and CTSS



Fernando J. Corbató with IBM 7090 (1962)

time sharing architecture (early 1960s)



first personal computer



Altair 8800 (1974)
hobbyist kit, required assembly



Paul Allen and Bill Gates (1975)
wrote BASIC interpreter for Altair

first mass-market personal computers

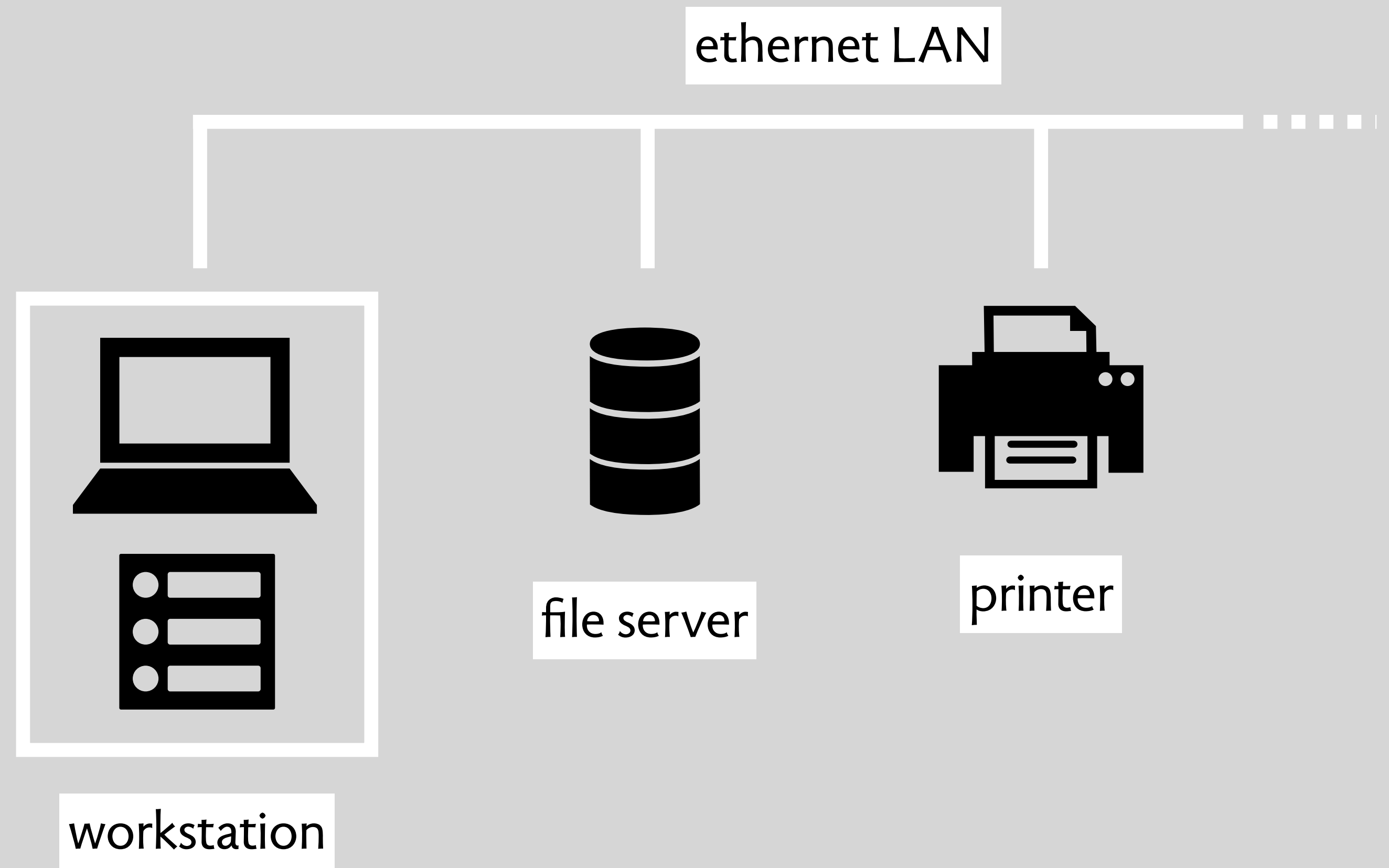


Commodore PET 2001 (1977)
all in one computer
for schools, home users, businesses

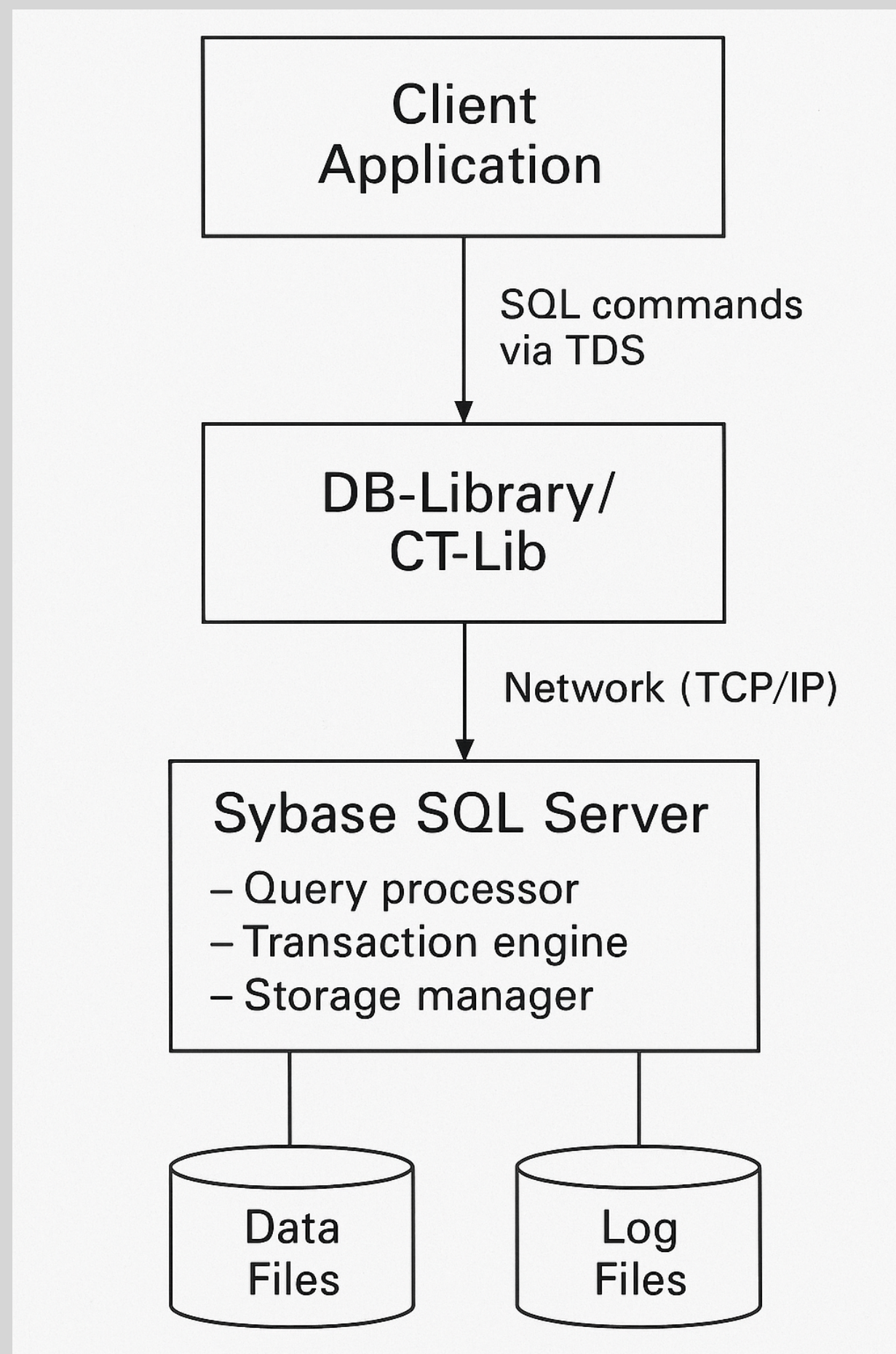
first client-server systems (1970s)



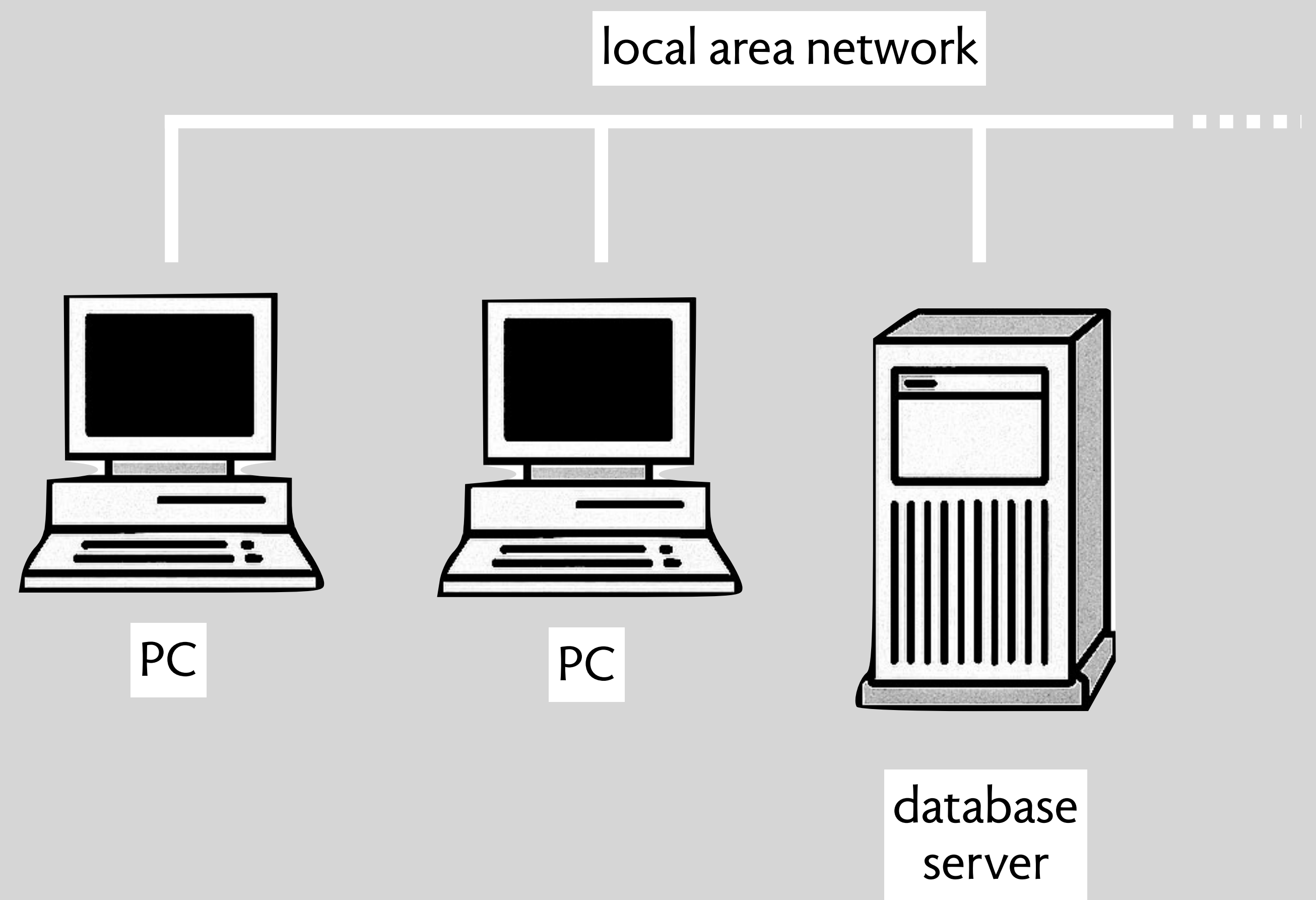
Xerox Alto (1973)
first "client-server" system
client is not just a terminal



first client-server database systems (1980s)



Sybase SQL Server (1986)
apps running on PCs
SQL queries to database server
no internet yet



what's HTTP?

TBL's proposal (1989)

CERN DD/OC

Information Management: A Proposal

Tim Berners-Lee, CERN/DD

March 1989

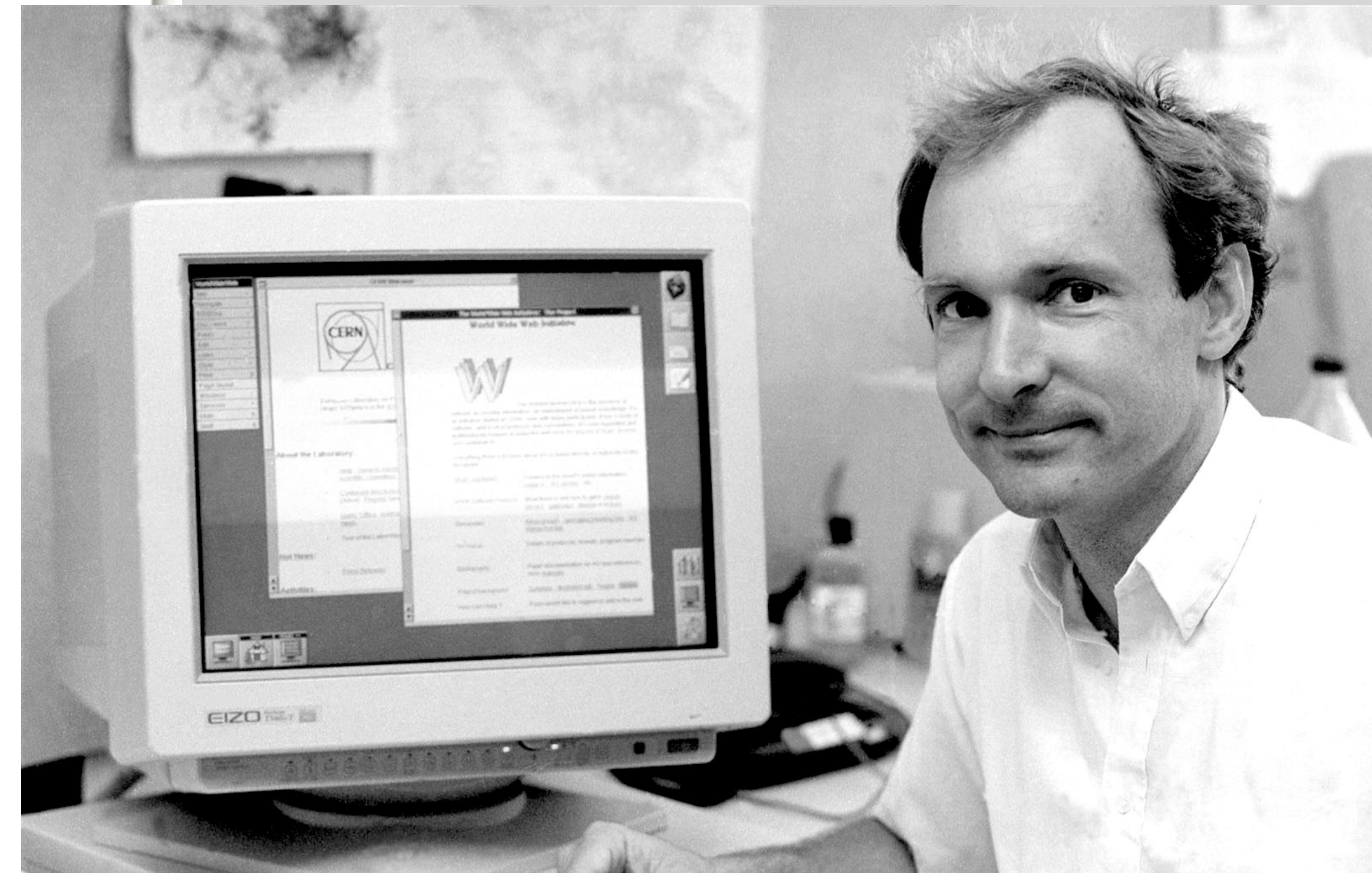
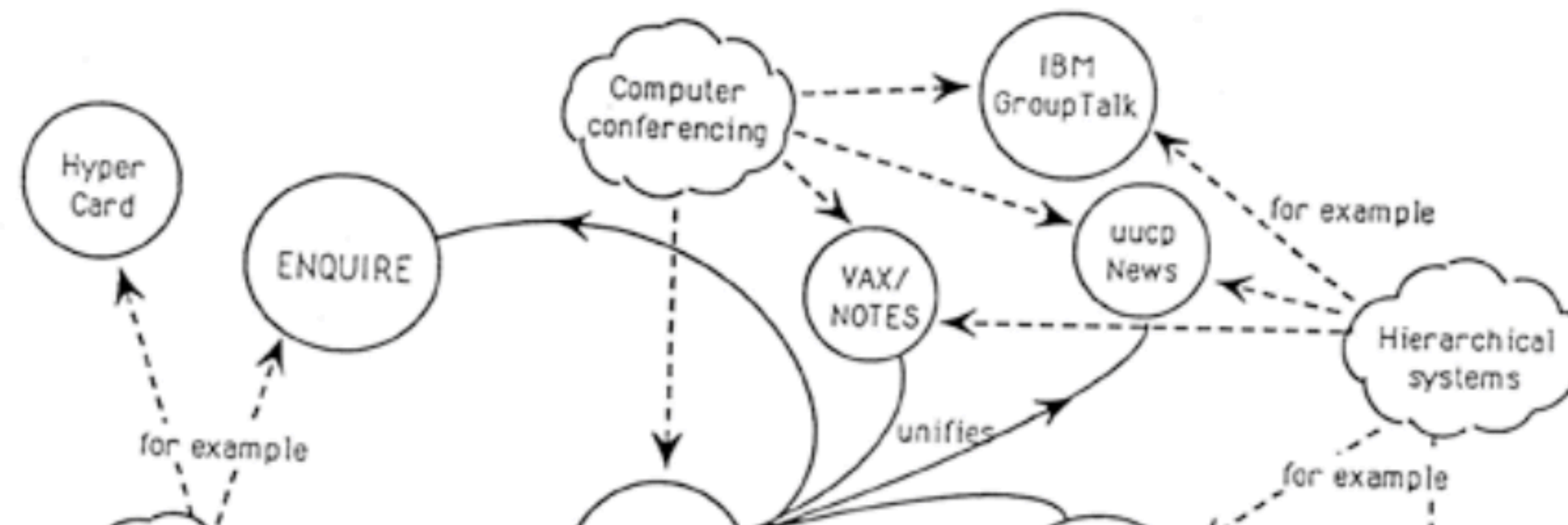
Vague but exciting ...

Information Management: A Proposal

Abstract

This proposal concerns the management of general information about accelerators and experiments at CERN. It discusses the problems of loss of information about complex evolving systems and derives a solution based on a distributed hypertext system.

Keywords: Hypertext, Computer conferencing, Document retrieval, Information management, Project control



the first web page (1991)

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..

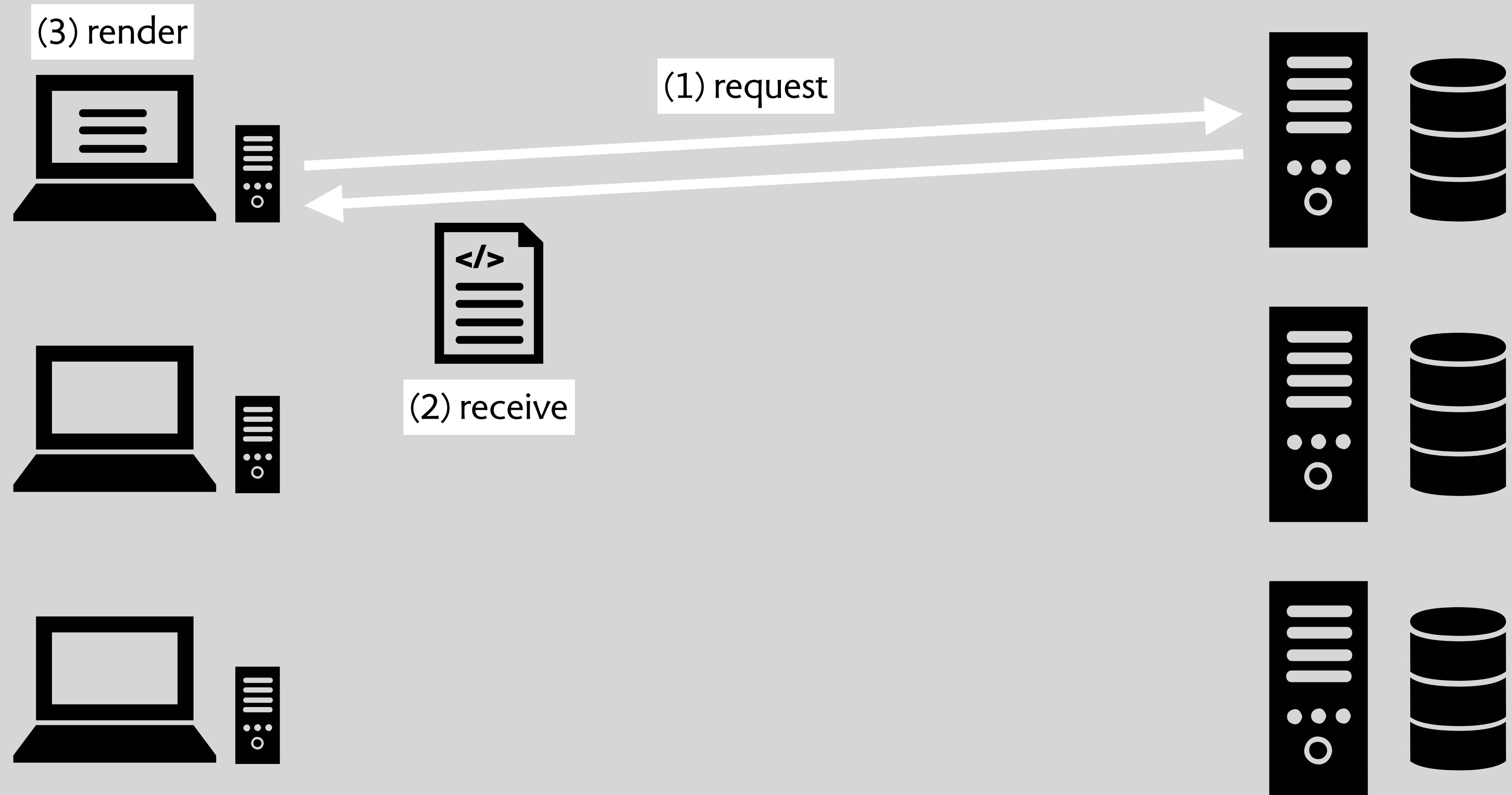
[Getting code](#)

Getting the code by [anonymous FTP](#) , etc.

essential ingredients

URLs + HTML + HTTP

TBL's web (1991)



the web as a document editing system: HTTP verbs

Create	→	POST
Read	→	GET
Update	→	PUT
Delete	→	DELETE

HTTP verbs

many originally proposed (1992)
eventually settles on these

distributed document editing

TBL's original concept of the web
never happens in this form

structure of a URL

`https://api.example.com/users/123/profile?view=full&include=posts&lang=en`



Component

Example

Description

Scheme / Protocol

`https://`

Specifies secure HTTP (HTTPS)

Domain name

`api.example.com`

The server's hostname

Path

`/users/123/profile`

The route to a specific resource

Query parameters

`?view=full&include=posts&lang=en`

Key-value pairs providing extra info for the request

Full meaning

"Request the full profile of user 123 from the API, including their posts, in English."

elements of an HTTP request

Element	Example	Description
Method / Verb	GET , POST , PUT , PATCH , DELETE	Specifies the action the client wants the server to perform on the resource.
Path	/api/users/42? verbose=true	The resource path and optional query parameters — relative to the host. (In most requests, this replaces the full URL.)
HTTP Version	HTTP/1.1	Indicates the HTTP protocol version being used.
Headers	Content-Type: application/json Authorization: Bearer abc123	Key–value pairs carrying metadata about the request: content type, authentication, caching, cookies, etc.
Host	Host: api.example.com	Specifies the target host (required in HTTP/1.1 and later).
Body / Payload	{ "email": "new@example.com" }	Optional data sent with the request — used mainly in POST , PUT , and PATCH methods.

REST: a standard pattern for routes

HTTP Request Type	URL Path	Description
GET	'/articles'	display a list of all articles
GET	'/articles/new'	return an HTML form for creating a new article
POST	'/articles'	create a new article
GET	'/articles/:id'	display a specific article
GET	'/articles/:id/edit'	return an HTML form for editing a article
PUT	'/articles/:id'	update a specific article
DELETE	'/articles/:id'	delete a specific article

Representational State Transfer (Roy Fielding, 2000)
example here for Ruby on Rails style

essence of the idea
nouns (URL, tree)
verbs (HTTP method)

our simple scheme

verb is
always **POST**

path is
concept/action

args passed as
raw JSON in body

good for this class
in industrial setting
use GETs for reads
(caching, reloads)

The screenshot shows a REST client interface with the following details:

- URL:** `http://localhost:8000/api/UserAuth/register`
- Method:** `POST`
- Body (raw JSON):** `{"username": "Mitchell", "password": "foobar"}`
- Response:** `200 OK` (55 ms, 178 B)
- Response Body (JSON):** `{ "user": "0199d994-f003-73c8-82e4-c50beed621cb" }`

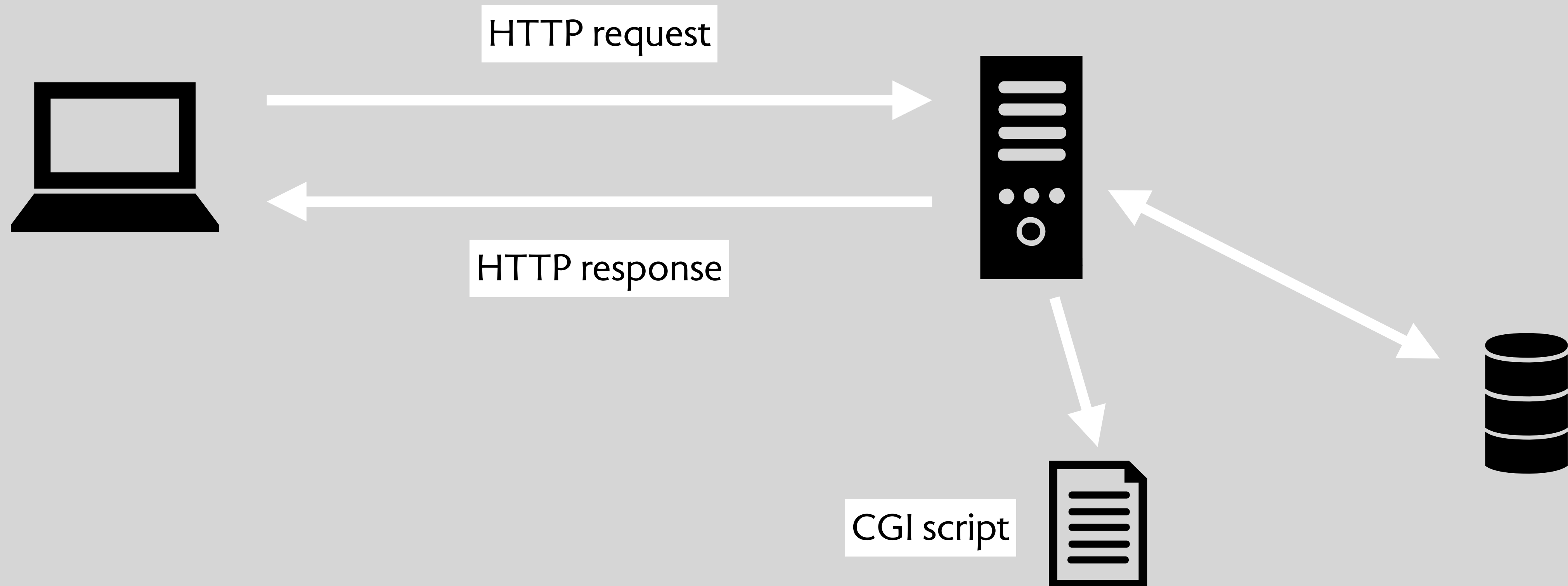
comedic
interlude

Mitchell and Webb: downloading the internet



how did web sites
become web apps?

common gateway interface (1993)



hello world CGI script

```
#!/usr/bin/perl  
print "Content-type: text/html\n\n";  
print "Hello, World.";
```

a famous CGI script

```
#!/usr/bin/perl

# Read form input
read(STDIN, $buffer, $ENV{'CONTENT_LENGTH'});
@pairs = split(/&/, $buffer);
foreach $pair (@pairs) {
    ($name, $value) = split(/=/, $pair);
    $value =~ tr/+/ /;
    $value =~ s/%(..)/pack("C", hex($1))/eg;
    $FORM{$name} = $value;
}

# Get recipient from the form field
$recipient = $FORM{'recipient'};

# Send mail
open(MAIL, "|/usr/lib/sendmail -t");
print MAIL "To: $recipient\n";
print MAIL "From: $FORM{'email'}\n";
print MAIL "Subject: Form submission\n\n";
print MAIL "Here are the results:\n";
foreach $key (keys %FORM) {
    print MAIL "$key = $FORM{$key}\n";
}
close(MAIL);
```

FormMail (1995)
contact form email
written by Matt Wright
a high school student



recipient=admin@example.com%0ABcc:spamlist@example.org

script didn't sanitize input
client can add extra fields
so use as spam relay
by early 2000s, blacklisted

a CGI-powered web app (1995)



Welcome to Amazon.com Books!

*One million titles,
consistently low prices.*

(If you explore just one thing, make it our [personal notification service](#). We think it's very cool! Also, check out [what our customers are saying about us](#).)

SPOTLIGHT! -- JANUARY 16TH

These are books we love, offered at Amazon.com low prices. The spotlight moves **EVERY** day so please come often.



ONE MILLION TITLES

Search Amazon.com's [million title catalog](#) by author, subject, title, keyword, and more... Or take a look at the [books we recommend](#) in over 20 categories... Check out our [customer reviews](#) and the [award winners](#) from the Hugo and Nebula to the Pulitzer and Nobel... and [bestsellers](#) are 30% off the publisher's list...



EYES & EDITORS, A PERSONAL NOTIFICATION SERVICE

Like to know when that book you want comes out in paperback or when your favorite author releases a new title? Eyes, our tireless, automated search agent, will send you mail. Meanwhile, our human editors are busy previewing galleys and reading advance reviews. They can let you know when especially wonderful works are published in particular genres or subject areas. Come in, [meet Eyes](#), and have it all explained.

YOUR ACCOUNT

Check the status of your orders or change the email address and password you have on file with us. Please note that you

running code in the browser

```
<div id="box" style="position:absolute; left:0px; top:100px;">🌟</div>
<script>
  let x = 0;
  function move() {
    x += 5;
    document.getElementById("box").style.left = x + "px";
    if (x < 300) setTimeout(move, 30);
  }
  move();
</script>
```

JavaScript (1995) and standard DOM (1998)



AJAX: asynchronous JavaScript and XML


```
var xhr = new XMLHttpRequest();
xhr.open("GET", "example.txt", true);
xhr.onreadystatechange = function() {
    if (xhr.readyState === 4 && xhr.status === 200) {
        console.log(xhr.responseText);
    }
};
xhr.send();
```

```
var xhr = new XMLHttpRequest();
xhr.open("POST", "/submit", true);
xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
xhr.send("name=Daniel&message=Hello");
```

XMLHttpRequest (1998)

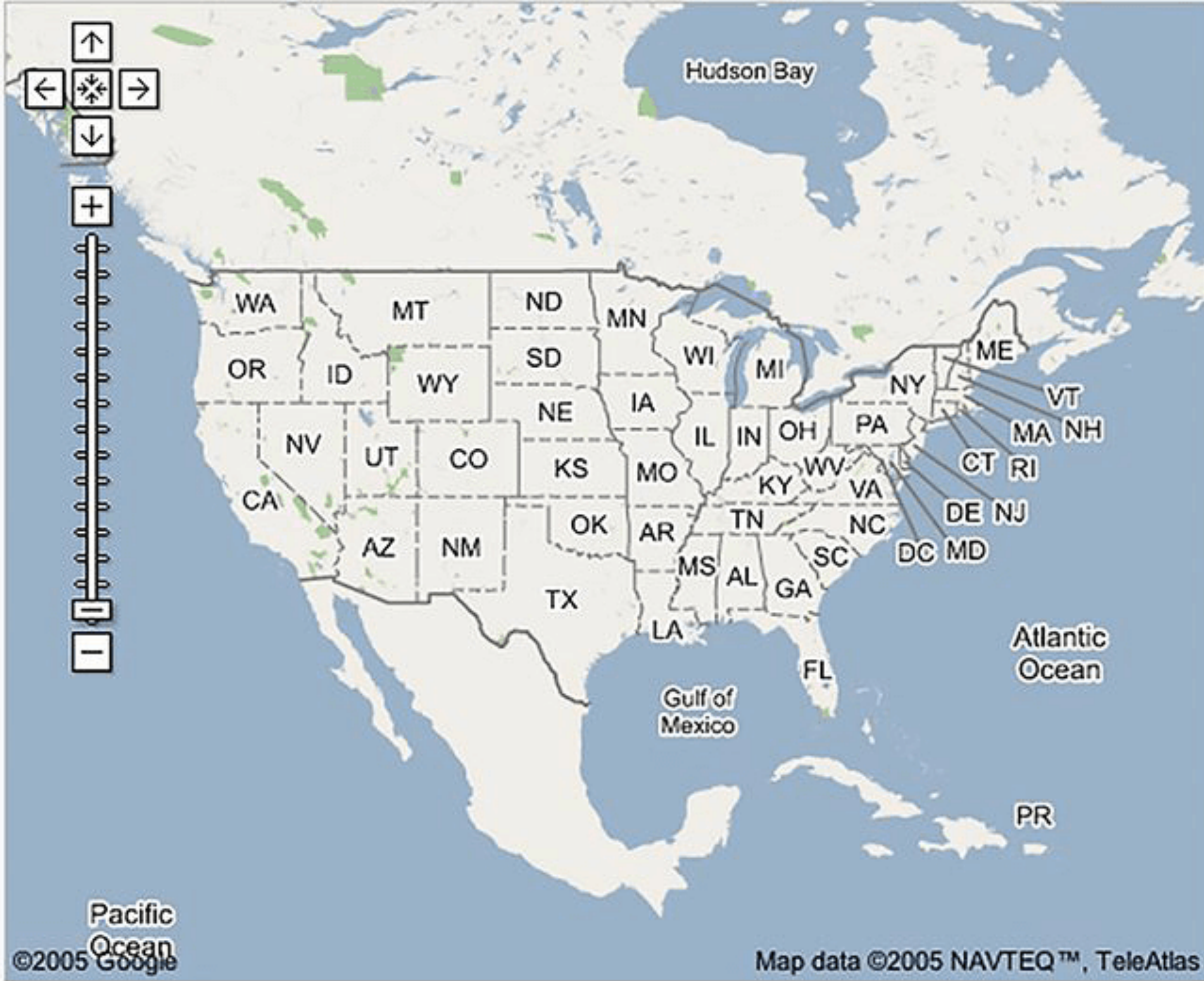
calling server inside a script in the browser
introduced by Microsoft
later standardized for all browsers

an AJAX-powered web app (2005)

**Maps** [Local Search](#) [Directions](#)

[Help](#) [Send Feedback](#)

Maps



©2005 Google

Map data ©2005 NAVTEQ™, TeleAtlas

[Print](#) [Email](#) [Link to this page](#)

Example searches

Go to a location:

Find a business:

Get directions:

Drag the map with your mouse, or double-click to center. [Take a tour »](#)

what server calls look like now

```
const api = axios.create({
  baseURL: '/api',
  headers: {
    'Content-Type': 'application/json',
    'Accept': 'application/json'
  },
  timeout: 10000
})
```

```
async register(username, password) {
  const response = await api.post('/UserAuth/register', {
    username,
    password
  })
  return response.data
},
```

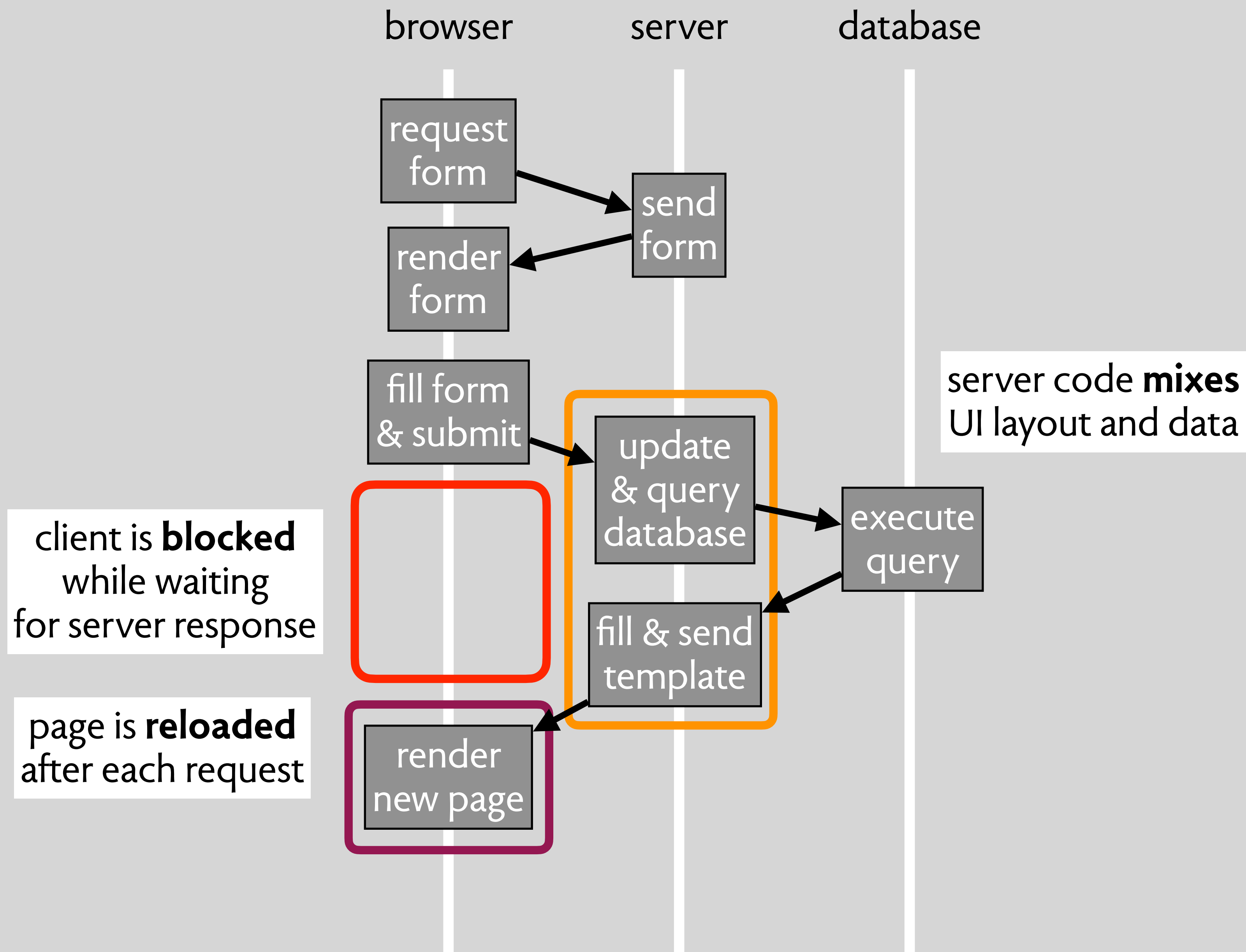
```
const result = await register(form.value.username, form.value.password)

if (result.success) {
  // Redirect to home page after successful registration
  router.push('/')
}
```

now like a local call
but asynchronous

what's an SPA?

the flow for a “multi-page” app



using a template to render HTML in the server

```
@app.route("/user/<name>")
def user_profile(name):
    # Data fetched from a database, for example
    user_info = {
        "name": name,
        "age": 32,
        "hobbies": ["music", "sailing", "AI"]
    }
    return render_template(
        "profile.html",
        user=user_info
    )
```

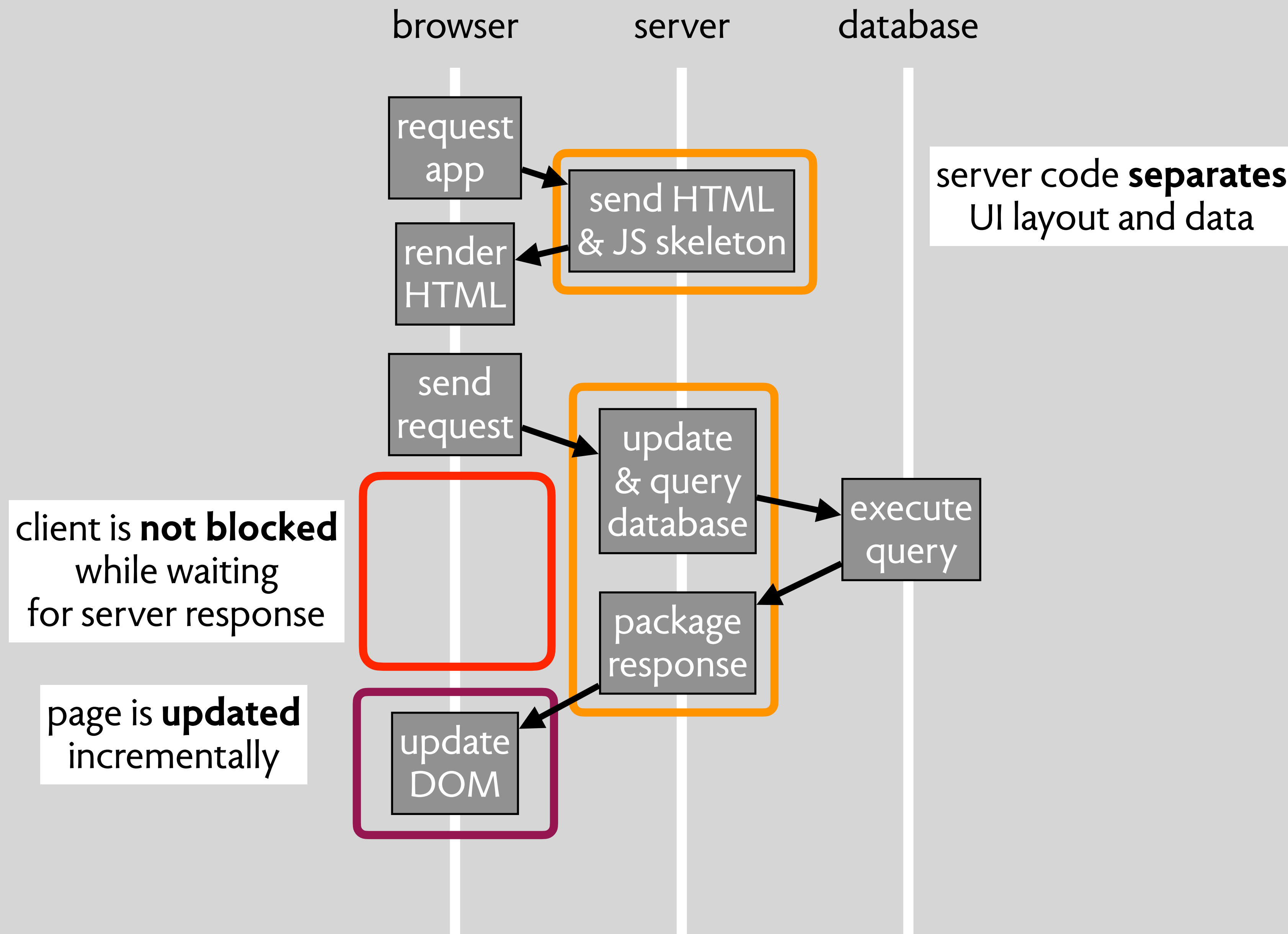
app.py

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8" />
    <title>{{ user.name }}'s Profile</title>
</head>
<body>
    <h1>Welcome, {{ user.name }}!</h1>
    <p>Age: {{ user.age }}</p>

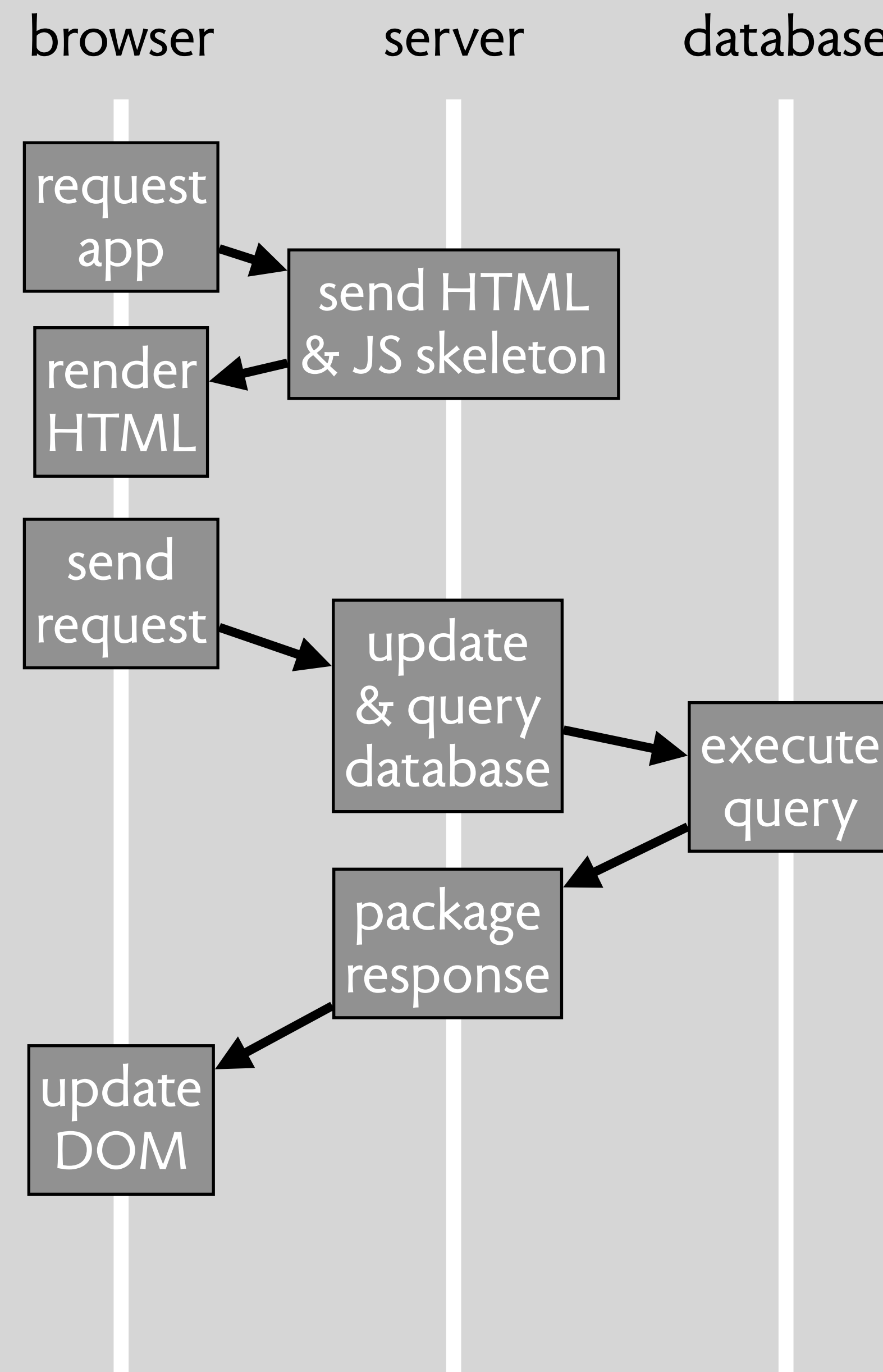
    <h2>Hobbies</h2>
    <ul>
        {% for hobby in user.hobbies %}
        <li>{{ hobby }}</li>
        {% endfor %}
    </ul>
</body>
</html>
```

profile.html

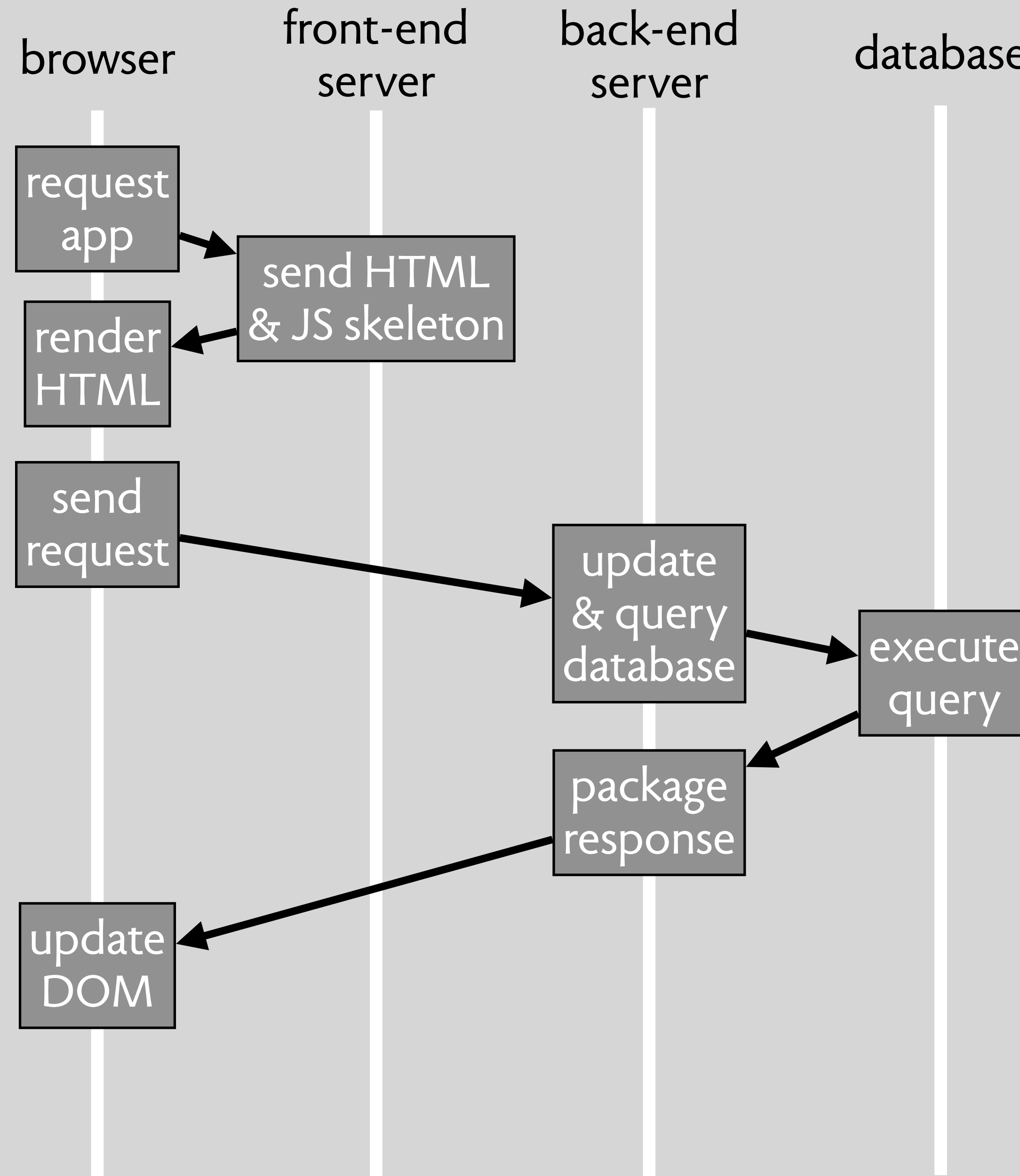
the flow for a "single page" app



the flow for a "single page" app



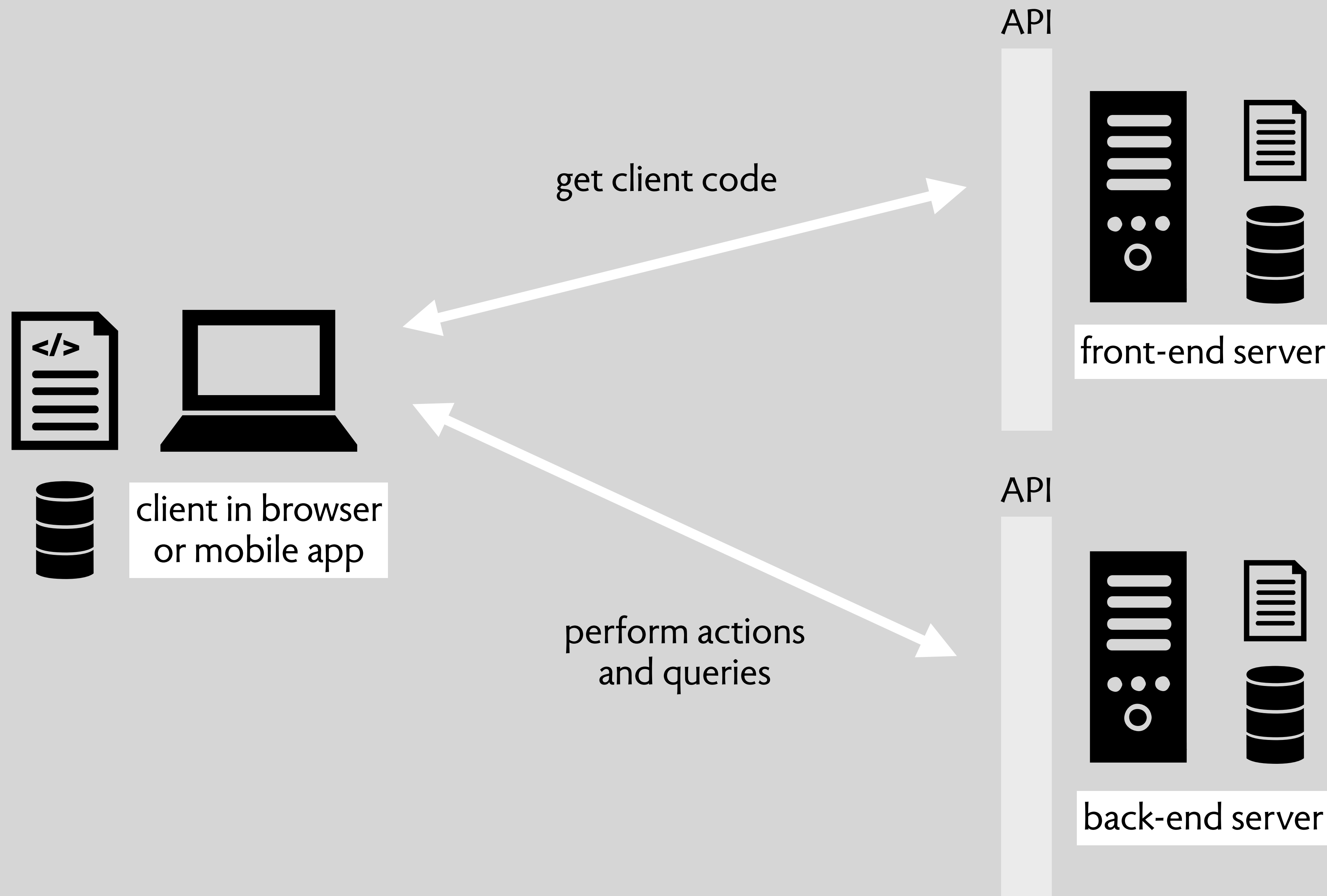
separating the roles of an SPA into two servers



a client-server
web app!

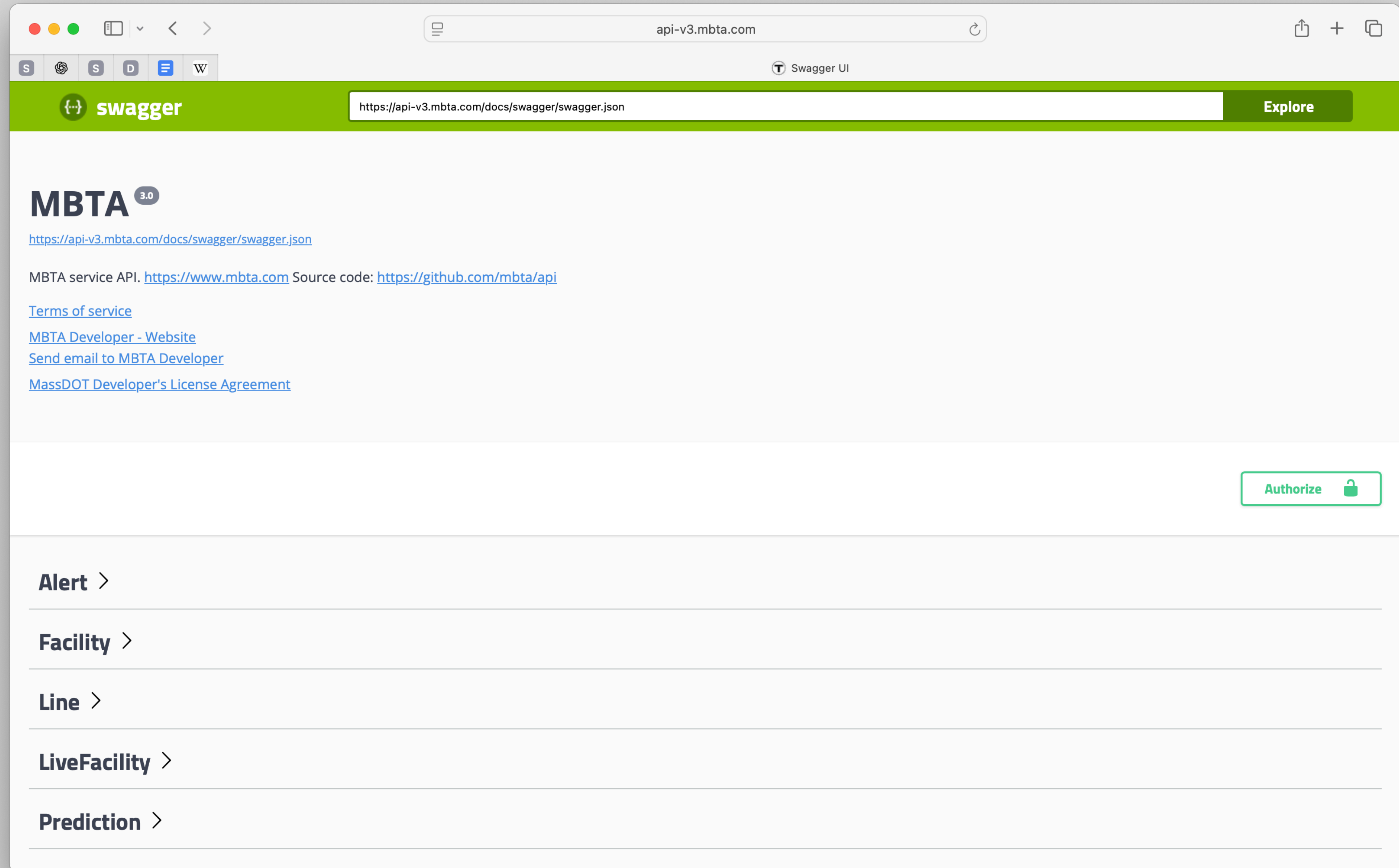
what's an API?

a typical setup

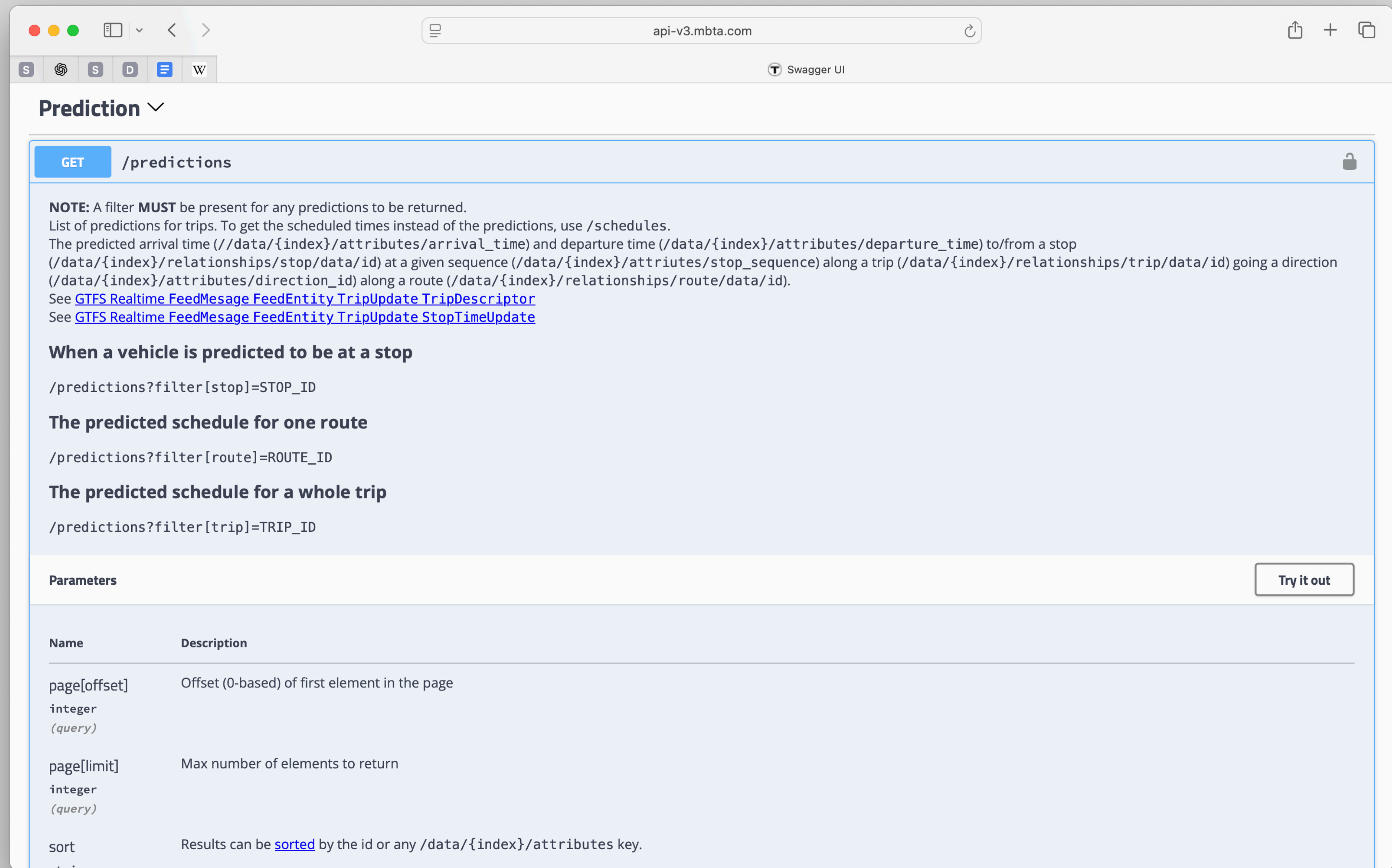


APIs give decoupling
minimize knowledge
hide changes to server
like all specs...

MBTA's API for live transport information



structure of a request



sample request & response

Example

https://api-v3.mbtta.com/predictions?filter%5Bstop%5D=place-sstat&filter%5Bdirection_id%5D=0&include=stop
returns predictions from South Station with direction_id=0, below is a truncated response with only relevant fields displayed:

```
{
  "data": [
    {
      "id": "prediction-CR-Weekday-Fall-18-743-South Station-02-1",
      "relationships": {
        "stop": {
          "data": {
            "id": "South Station-02",
            "type": "stop"
          }
        },
      },
      "type": "prediction"
    }
  ],
  "included": [
    {
      "attributes": {
        "platform_code": "2",
      },
      "id": "South Station-02",
      "type": "stop"
    }
  ],
}
```

Note the stop relationship; use it to cross-reference stop-id with the included stops to retrieve the platform_code for the given prediction.

JSON pointers

sort

Results can be [sorted](#) by the id or any /data/{index}/attributes key.

string (query)	JSON pointer	Direction	sort
	/data/{index}/attributes/arrival_time	ascending	arrival_time
	/data/{index}/attributes/arrival_time	descending	-arrival_time
	/data/{index}/attributes/arrival_uncertainty	ascending	arrival_uncertainty
	/data/{index}/attributes/arrival_uncertainty	descending	-arrival_uncertainty
	/data/{index}/attributes/departure_time	ascending	departure_time
	/data/{index}/attributes/departure_time	descending	-departure_time
	/data/{index}/attributes/departure_uncertainty	ascending	departure_uncertainty
	/data/{index}/attributes/departure_uncertainty	descending	-departure_uncertainty
	/data/{index}/attributes/direction_id	ascending	direction_id
	/data/{index}/attributes/direction_id	descending	-direction_id

Twilio: a service for sending text messages

Twilio Docs

General Usage

> Get started

> Twilio APIs

Overview

API requests

API responses

API mutation and conflict resolution

API best practices

REST API requests in PowerShell

> Accounts

Addresses

Applications

> BulkExport API

Key

Usage records

Usage triggers

> Monitor REST API

Browse all of Twilio's APIs

> Webhooks

Messaging

Voice

Video

Serverless

Flex

Studio

All docs...

SDKs

Help Center

Search

Log in

Sign up

Twilio API requests

Learn how to authenticate your requests, what content type to use for API requests, and how the Twilio APIs handle webhooks. You'll also see examples of how to make requests to the Twilio APIs.

Ways to make requests to the Twilio APIs

There are several ways you can make an HTTP request to the Twilio API.

- Make a raw HTTP request either in your code (for example, by using a module like [got in NodeJS](#)) or with a tool like [Postman](#).
- Use a [Twilio SDK](#) for your preferred programming language.
- Use the [Twilio CLI](#) if you prefer working in the terminal.

Authentication

Environment variables

Always store your credentials in environment variables before sharing any code or deploying to production. Learn more about [setting environment variables](#).

Page tools

View as Markdown

Copy as Markdown

On this page

Ways to make requests to the Twilio APIs

Authentication

- Using API keys (recommended)
- Using your Account SID and Auth Token
- Twilio SDKs

HTTP Methods

Content type

Global Edge Locations & How Requests Enter

Looking for more inspiration?

Visit the [Developer Hub](#)

COOKIE PREFERENCES

creating a message

Twilio Docs

Programmable Messaging

> Getting Started

> API reference

API overview

> Messages resource

Messages

Feedback

Media

> Services resource

Deactivations resource

ShortCodes resource

Verifications resource

> A2P 10DLC

> Pricing API

> Preventing Fraud

> Tutorials

> Messaging Services

> Messaging Features

> Usage Guides

Messaging

Voice

Video

Serverless

Flex

Studio

All docs...

SDKs

Help Center

Search

Log in

Sign up

Create a Message resource

POST https://api.twilio.com/2010-04-01/Accounts/{AccountSid}/Messages.json

To send a new outgoing message, send an HTTP POST request to your Account's Messages list resource URI.

Every request to create a new Message resource requires a recipient, a sender, and content.

A recipient is specified via the To parameter.

The sender is specified via one of the following parameters:

From

MessagingServiceSid

The message content is specified via one of the following parameters:

MediaUrl

Body

ContentSid

The table below describes these parameters in more detail.

Path parameters

accountSid SID<AC> required

The SID of the Account creating the Message resource.

Page tools

View as Markdown

Copy as Markdown

On this page

Message Properties

Message Status values

NumSegments property

Create a Message resource

Path parameters

Request body parameters

Twilio's request to the StatusCallback URL

SMS/MMS


WhatsApp and other messaging channels

Looking for more inspiration?

Visit the Developer Hub

COOKIE PREFERENCES

request body parameters


Request body parameters 

Encoding type: application/x-www-form-urlencoded

Schema


Example

to string<phone-number> **required**



The recipient's phone number in [E.164](#) format (for SMS/MMS) or [channel address](#), e.g.
`whatsapp: +15552229999`.

statusCallback string<uri>



The URL of the endpoint to which Twilio sends [Message status callback requests](#). URL must contain a valid hostname and underscores are not allowed. If you include this parameter with the `messaging_service_sid`, Twilio uses this URL instead of the Status Callback URL of the [Messaging Service](#).

```
{
  "ApplicationSid": "APaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa",
  "Body": "Hello! 👍",
  "From": "+14155552345",
  "MediaUrl": [
    "https://c1.staticflickr.com/3/2899/14341091933_1e92e62d12_b.jpg"
  ],
  "PersistentAction": [
    "mailto:test@example.com"
  ],
  "StatusCallback": "https://example.com",
  "To": "+14155552345",
  "Tags": "{ \"campaign_name\": \"Spring Sale 2022\", \"message_type\": \"cart_abandoned\" }"
}
```

using the Twilio client-side library

Send an SMS message 

Node.js 



```
1 // Download the helper library from https://www.twilio.com/docs/node/install
2 const twilio = require("twilio"); // Or, for ESM: import twilio from "twilio";
3
4 // Find your Account SID and Auth Token at twilio.com/console
5 // and set the environment variables. See http://twil.io/secure
6 const accountSid = process.env.TWILIO_ACCOUNT_SID;
7 const authToken = process.env.TWILIO_AUTH_TOKEN;
8 const client = twilio(accountSid, authToken);
9
10 async function createMessage() {
11   const message = await client.messages.create({
12     body: "Hi there",
13     from: "+15557122661",
14     to: "+15558675310",
15   });
16
17   console.log(message.body);
18 }
19
20 createMessage();
```

```
1 {
2   "account_sid": "ACXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
3   "api_version": "2010-04-01",
4   "body": "Hi there",
5   "date_created": "Thu, 24 Aug 2023 05:01:45 +0000",
6   "date_sent": "Thu, 24 Aug 2023 05:01:45 +0000",
7   "date_updated": "Thu, 24 Aug 2023 05:01:45 +0000",
8   "direction": "outbound-api",
9   "error_code": null,
10  "error_message": null,
11  "from": "+15557122661",
12  "num_media": "0",
13  "num_segments": "1",
14  "price": null,
15  "price_unit": null,
16  "messaging_service_sid": "MGaaaaaaaaaaaaaaaaaaaaaaaaaaaa",
17  "sid": "SMaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa",
18  "status": "queued",
19  "subresource_uris": {
20    "media": "/2010-04-01/Accounts/ACaaaaaaaaaaaaaaaaaaaaaaaaaaaa/Messages/SMaaaaaaaaaaaaaaaaaaaaaaaaaaaa",
21  },
22  "to": "+15558675310",
23  "uri": "/2010-04-01/Accounts/ACaaaaaaaaaaaaaaaaaaaaaaaaaaaa/Messages/SMaaaaaaaaaaaaaaaaaaaaaaaaaaaa",
24 }
```

what's a static site?

static web sites

what the user sees

resource returned depends on HTTP request
and not on previous HTTP requests

what the developer sees

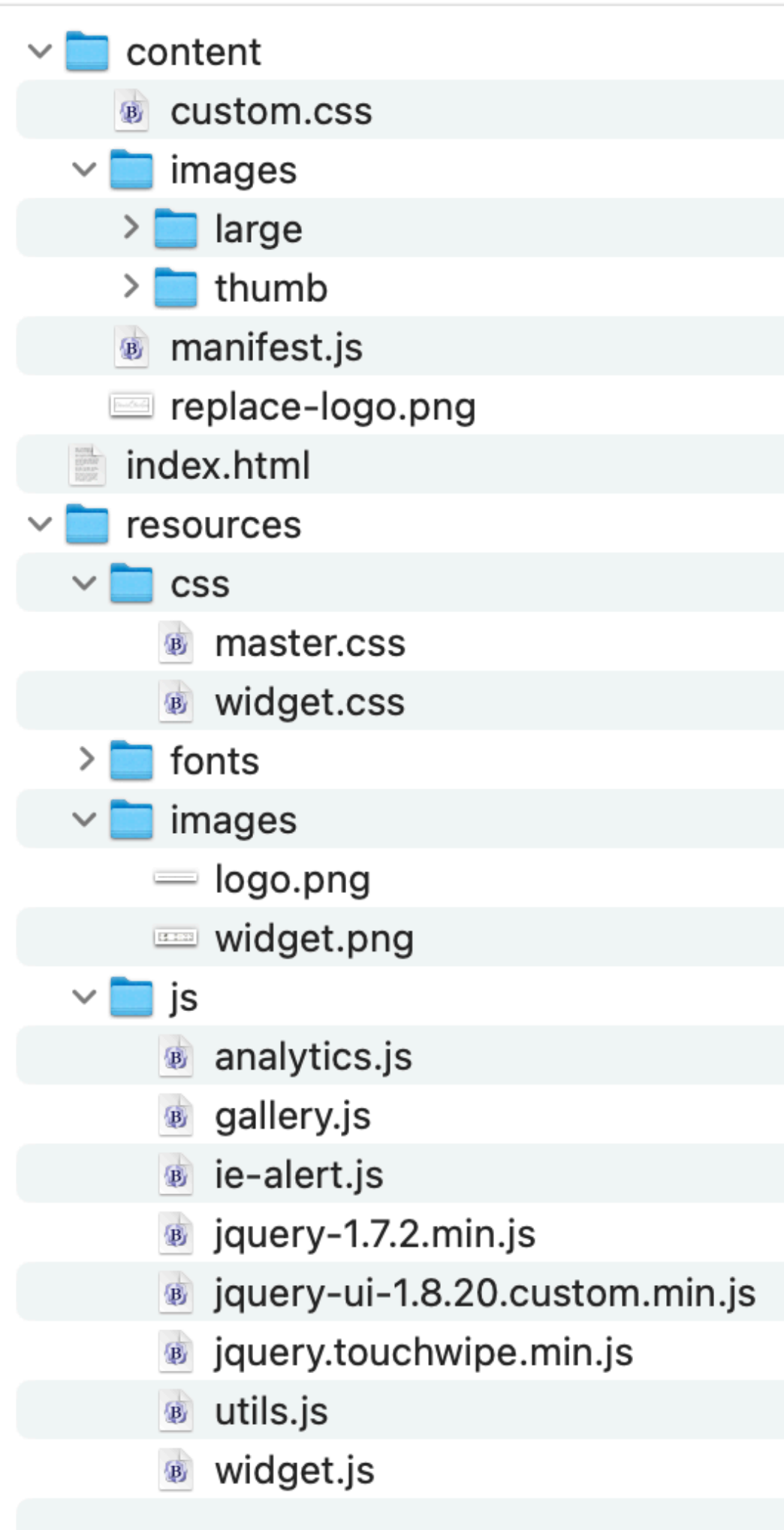
site is just a collection of files, not programs or databases
served directly by a web server
can include JavaScript that runs in the browser

how to create a static site

set up a server or find a hosting service (eg, GitHub Pages)
edit files locally and push to server (eg, with Git or Unison)
or, use a static site builder or CMS

front-end server
is a static website!

example: my photo web pages



files that produce this page

static website builders

6.1040 Fall 2025

Schedule

Assignments

Preps

Recitation/Office Hours

Resources

Instructors

Class Guide

FAQ

Schedule

L: Lecture / R: Recitation; links to slides are provided once available

Date	Topic	Assignment Due (11:59PM)	Prep Due (10am)	
Wed Sep 3	L: Starting to Design			
Thu Sep 4	R: GitHub, readmes, markdown		Prep 1: Markdown	
Sun Sep 7		Assignment 1: Problem Framing		
Mon Sep 8	L: Conceptual Design			
Wed Sep 10	L: Designing Behavior			
Thu Sep 11	R: States and Actions			

what they offer

editing in markdown, not HTML
consistent styles, packaged in themes
index pages (eg, of blog posts by date)
easy management of nav bars
live view while editing locally

how they work

you edit in markdown
run build command
push to repo

popular examples

Jekyll (Ruby)

Hugo (Go)

disadvantages

sometimes obscure, can be limiting
themes aren't truly interchangeable

61040-fa25.github.io /

1-schedule.md

in

main

Cancel changes

Commit changes...

Edit

Preview

Spaces

2

Soft wrap

1

2

title: Schedule

3

layout: page

4

permalink: /schedule

5

6

7

L: Lecture / R: Recitation; links to slides are provided once available

8

9

| Date | Topic | Assignment Due

10

| (11:59PM) | |

11

| Wed Sep 3 | [[L: Starting to Design](#)](assets/lecture-notes/problem-framing.pdf) |

12

| Thu Sep 4 | [[R: GitHub, readmes, markdown](#)](assets/recitation_notes/61040-Rec1-Slides.pdf) |

13

| Sun Sep 7 | [[Prep 1: Markdown](#)](preps/prep-1.md) | [[Assignment 1:](#)

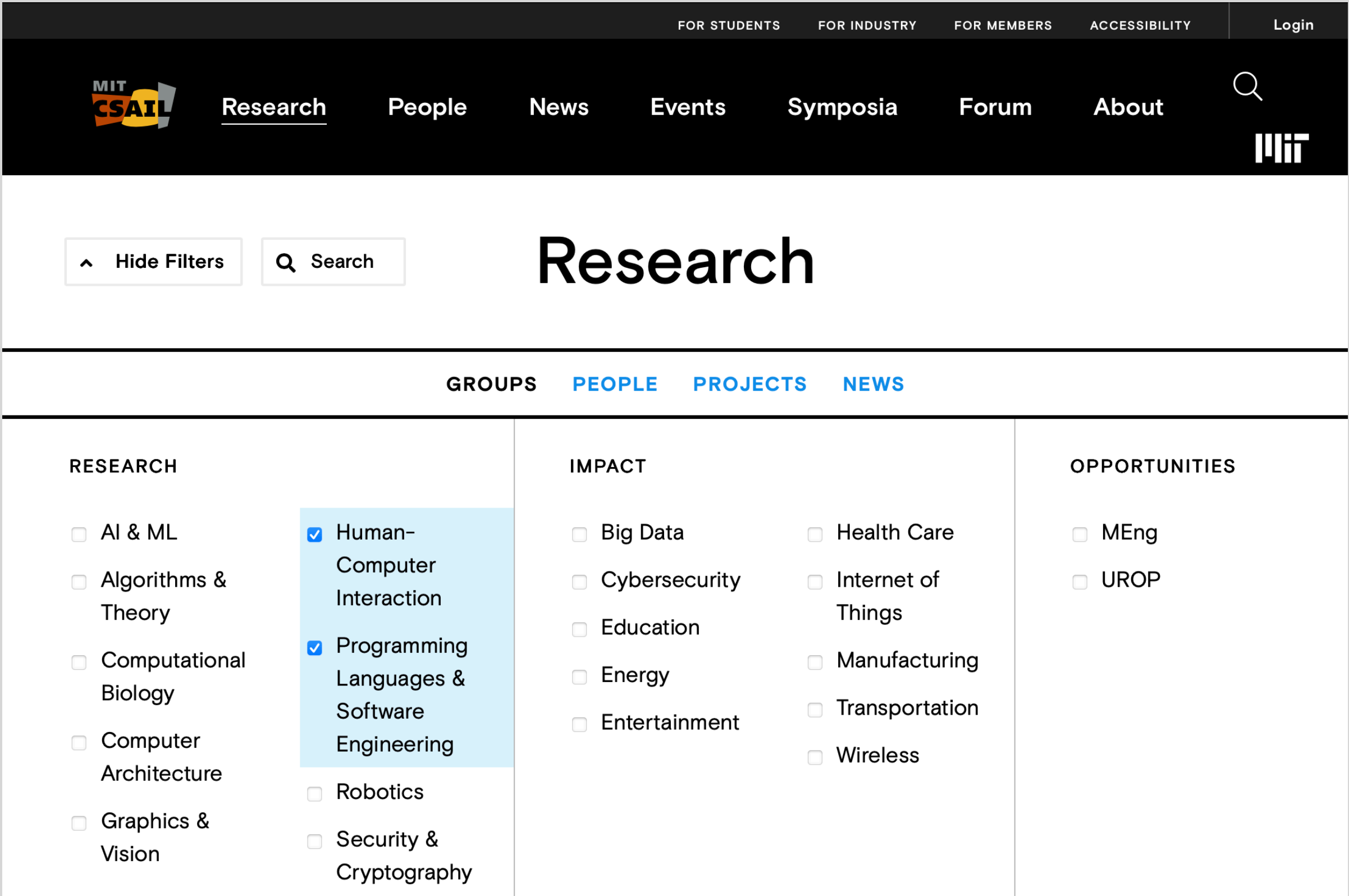
content management systems

what they are

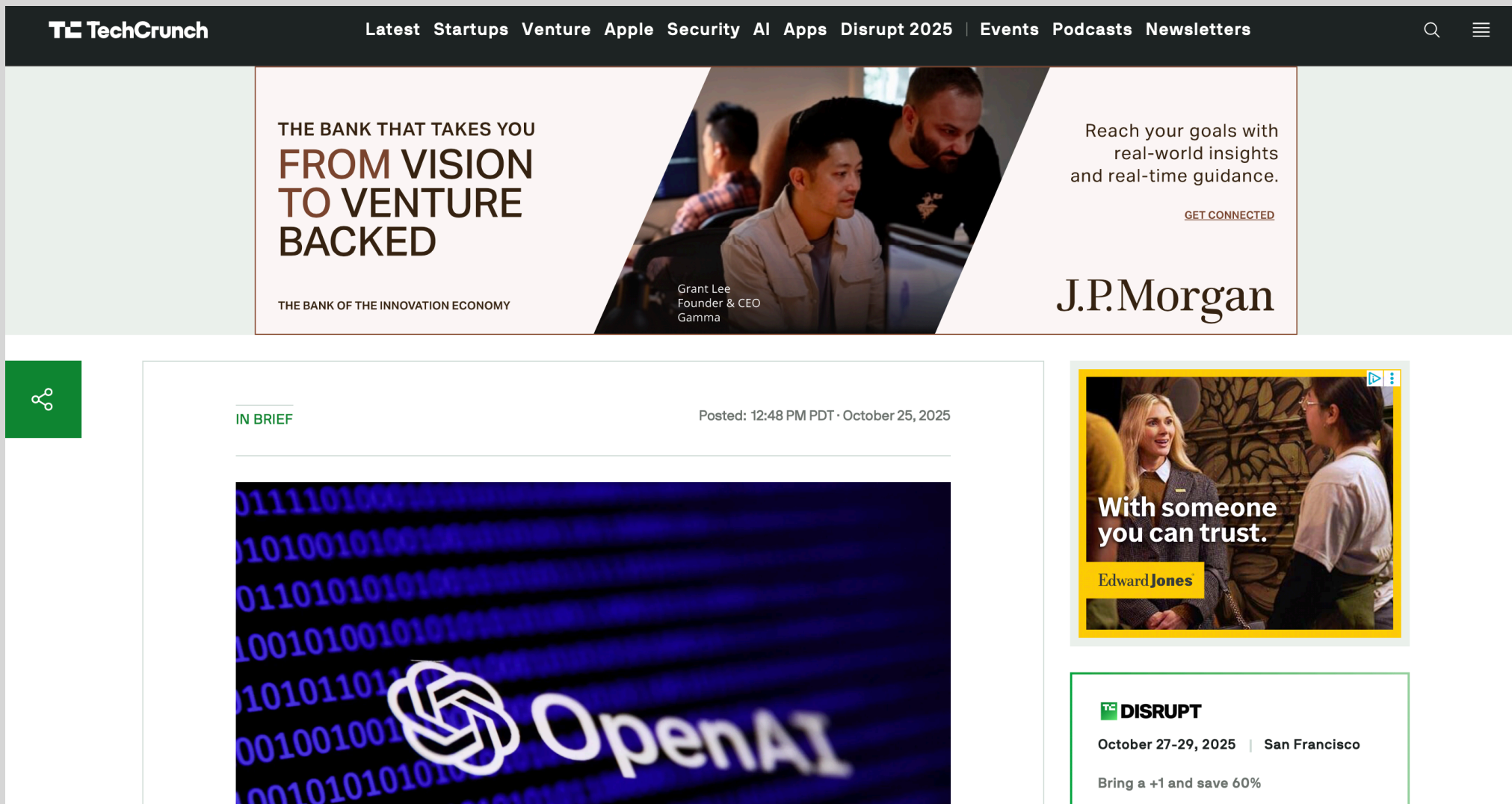
static to users, dynamic to admins
updates via HTTP requests (ie, edit site in browser)

examples

WordPress: popular CMS for small sites
Wix, Google Sites: combined site builder and CMS
SquareSpace: site builder/CMS for small businesses
Drupal: CMS for complex sites, often built by pros



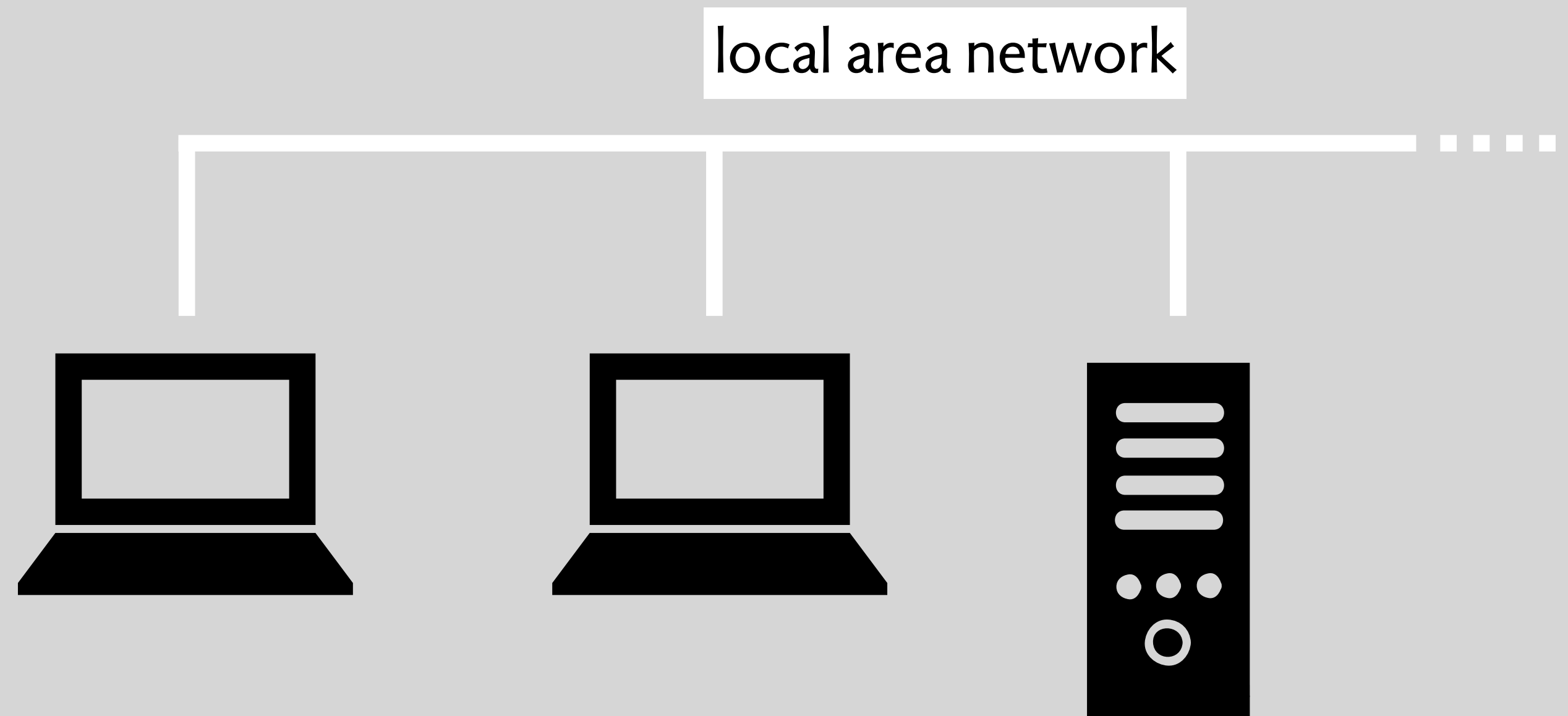
csail.mit.edu (Drupal)



techcrunch.mit.edu (WordPress)

how are clients
authenticated?

how to authenticate clients?

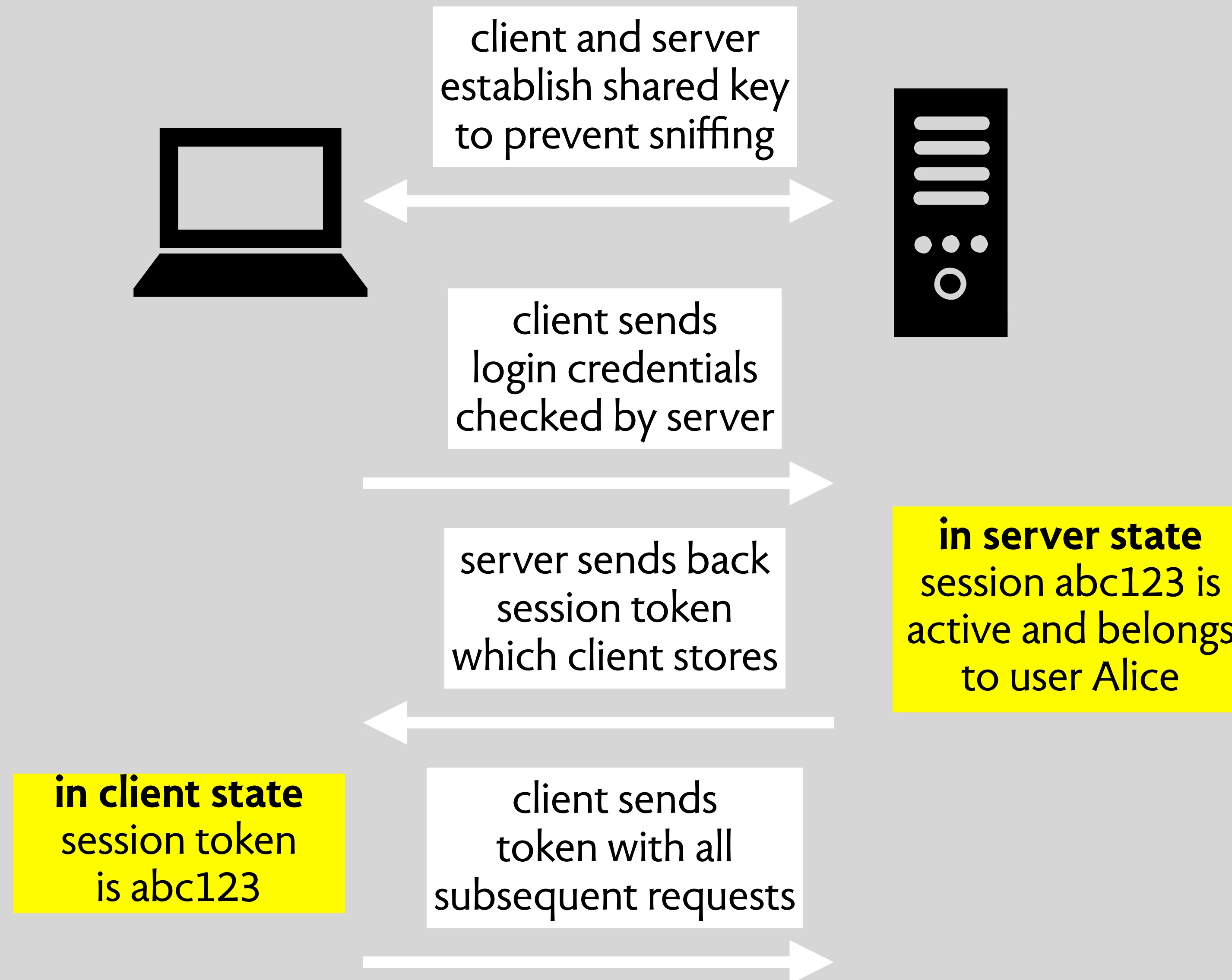


“trust by wire” in early LANs
if you could access the physical network
you were considered authenticated

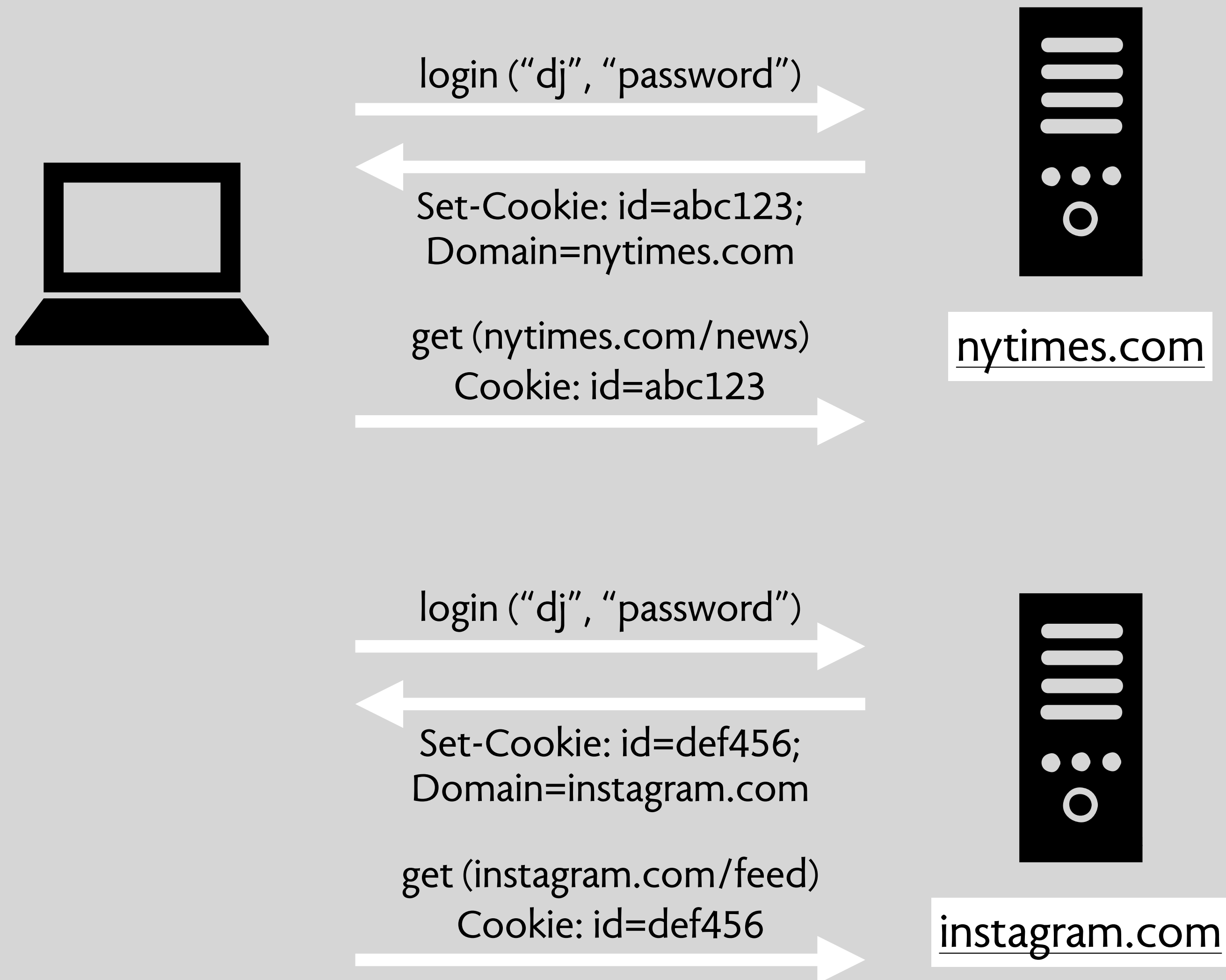
password auth in early LANs
server opens connection, user logs in & access files
passwords sent in clear, easy to hijack connection too

dial-up modems were a bit safer
client gets next available modem line
modem connection was a private circuit

a better scheme: session tokens



cookies: a way to store session tokens

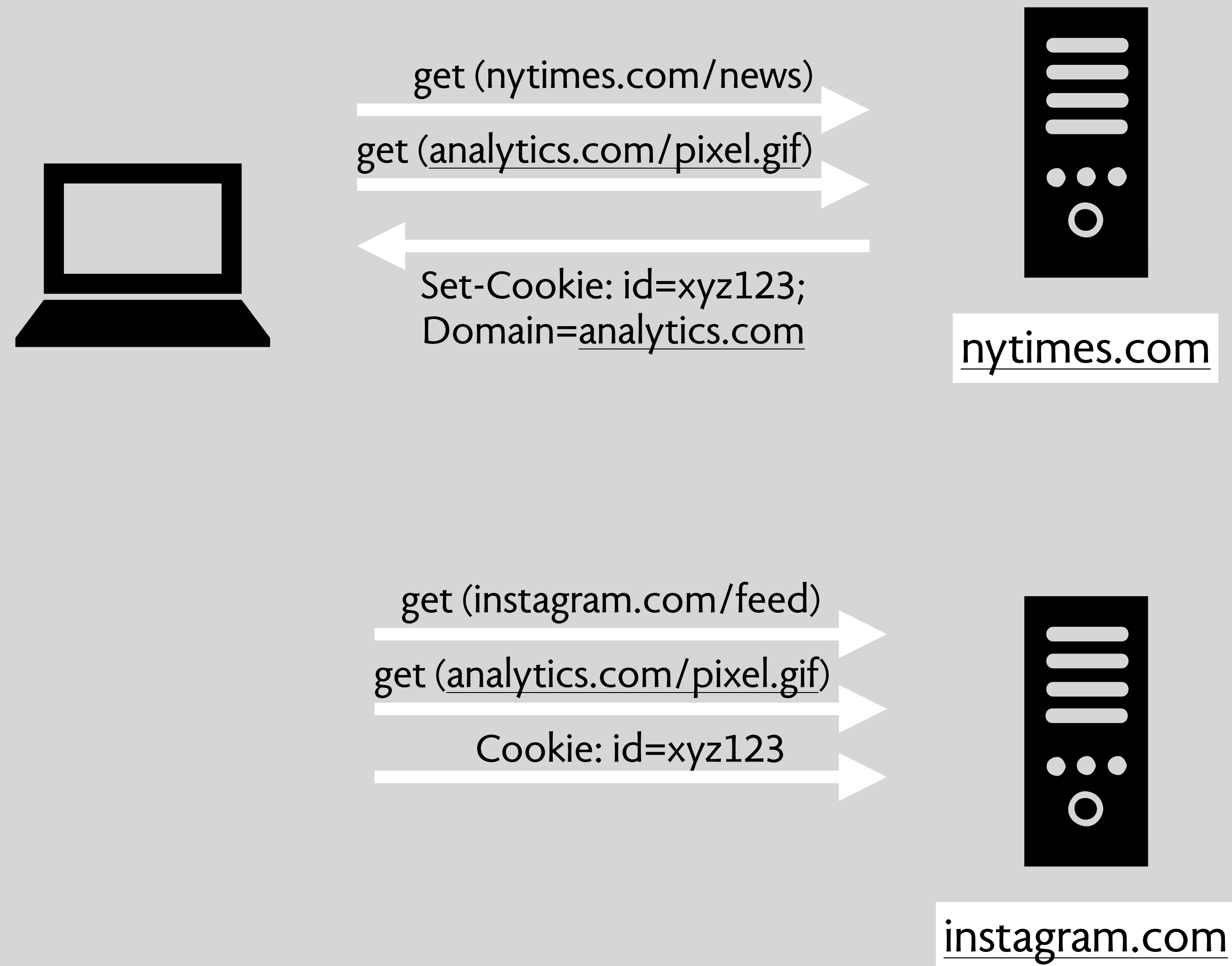


other uses of cookies
shopping carts eg

nice for programmer
cookies sent by default

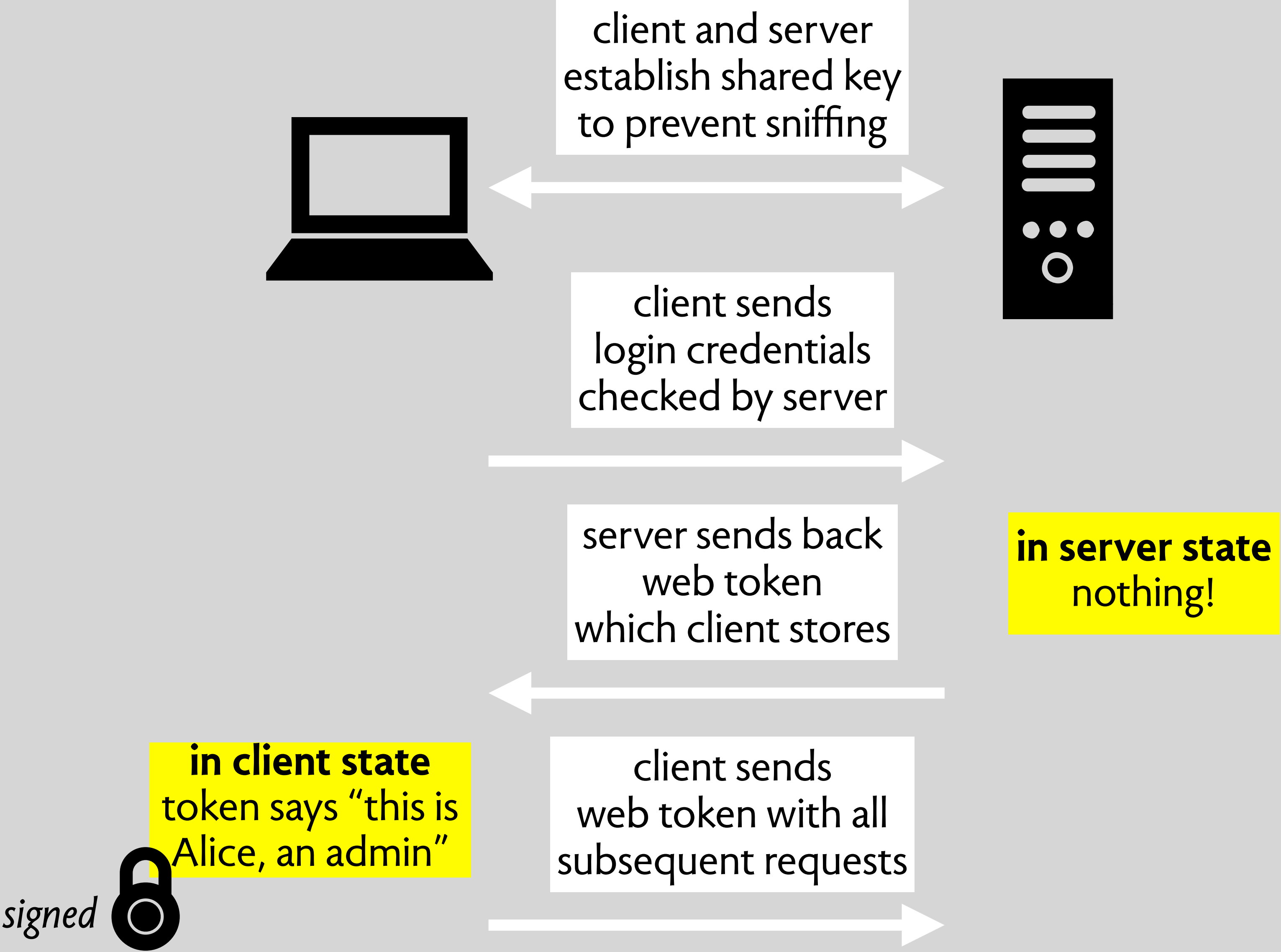
security model
domains stay separate

third-party cookies



cross site tracking
GDPR requires consent

JSON web tokens (JWTs)



refreshing web tokens

a problem: long or short?

with conventional sessions, server can always close a session
but with JWTs, can access until token expires
insecure if token is long lived, but inconvenient if short lived!

solution: two kinds of token

long-lived refresh tokens: rarely used, safe in browser (eg 14 days)

short-lived access tokens: used for all accesses (eg 10 mins)

when access token expires, client uses refresh token to get a new one

when refresh token expires, have to log in again

what goes on
the client or server?

security considerations

code and data in the browser

are visible to and modifiable by the user
with developer tools

user can issue any HTTP requests

by modifying JS in the browser document
by commands in the browser JS console
by using curl or Postman



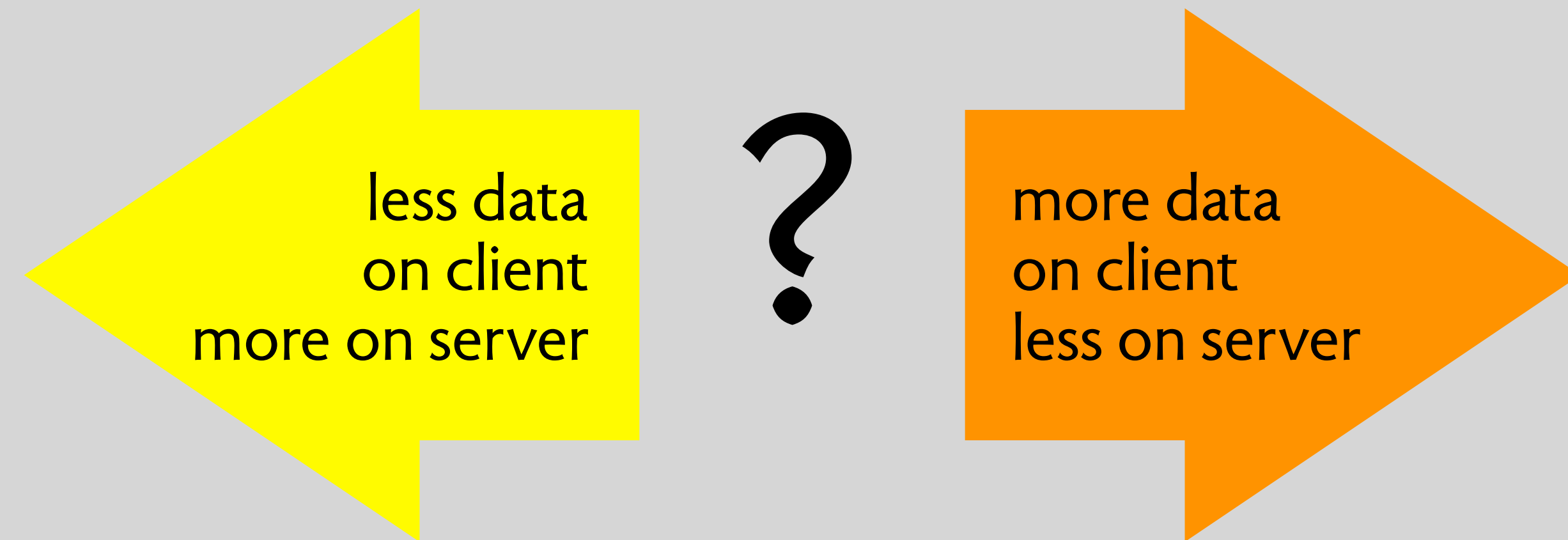
so which of these are good strategies?

to prevent access to another user's data
ensure client code passes username with request

to prevent access to sensitive pages
navigate first through login page

to prevent access to another user's data
use counter for session ids and store id in cookie

to prevent access to another user's data
generate random session id and store in cookie



performance considerations



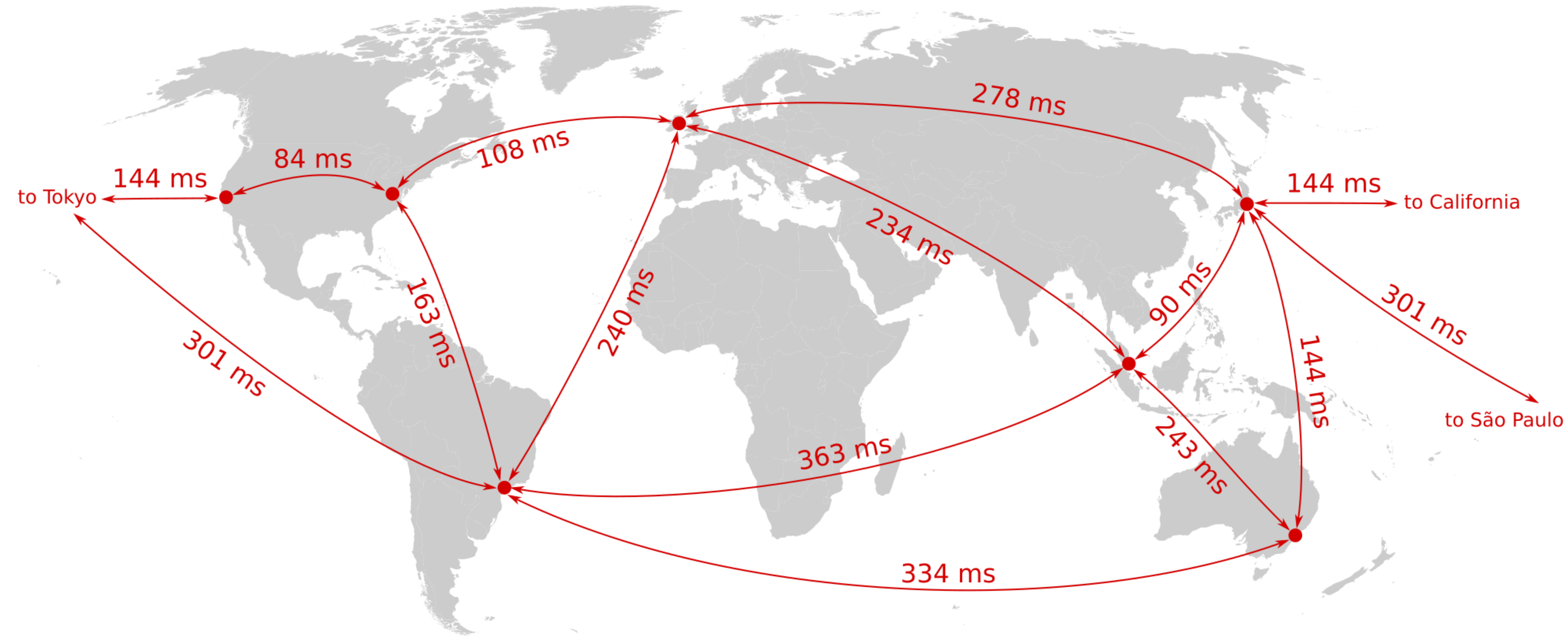
speed of queries
ability to work offline
scaling to more clients

local storage usage
initial startup time
risk of privacy violation
observability by devs

no data
on the
client

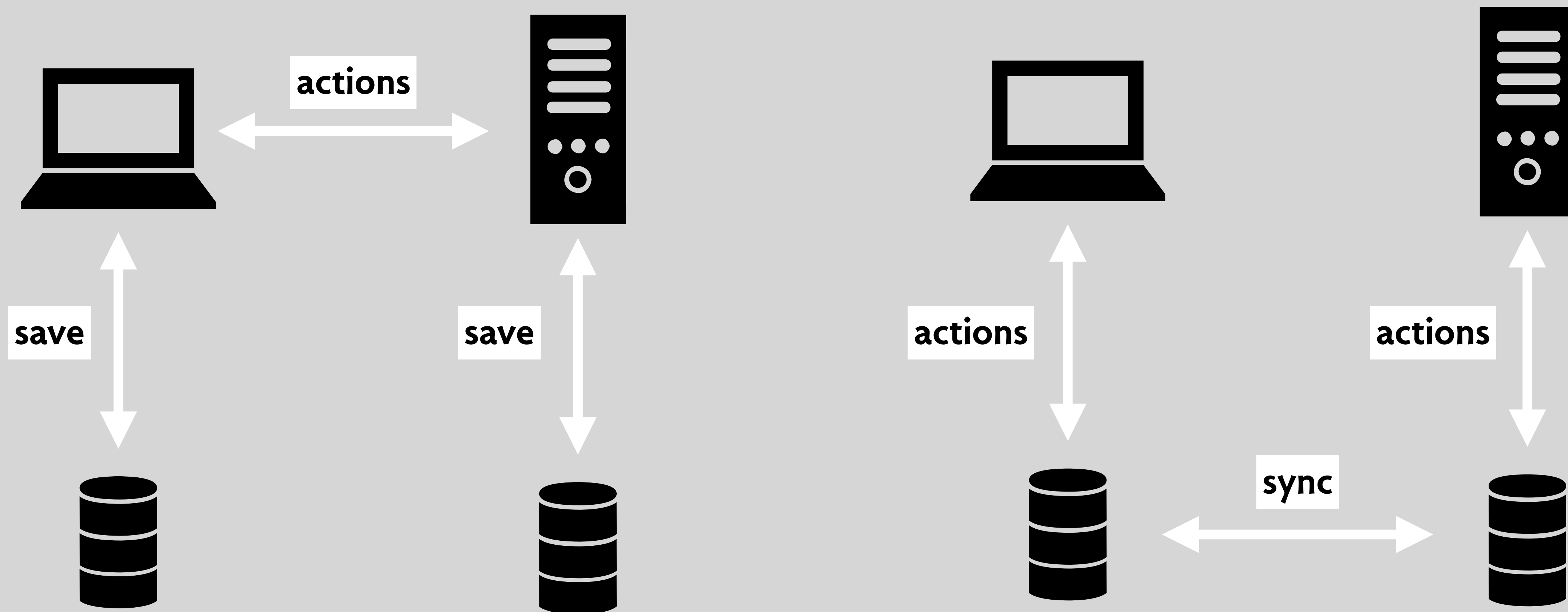
all data
on the
client

online apps are slow!



server-to-server round trip times between AWS data centers (Ink & Switch)

local first: a proposal for a new kind of app



takeaways

key ideas from this lecture

client server architecture

web apps just one example

APIs

specs, our old friend

tricky challenges

many inherent to all apps

web-specific notions

HTTP protocol, DOM

web challenges

how good a sandbox?