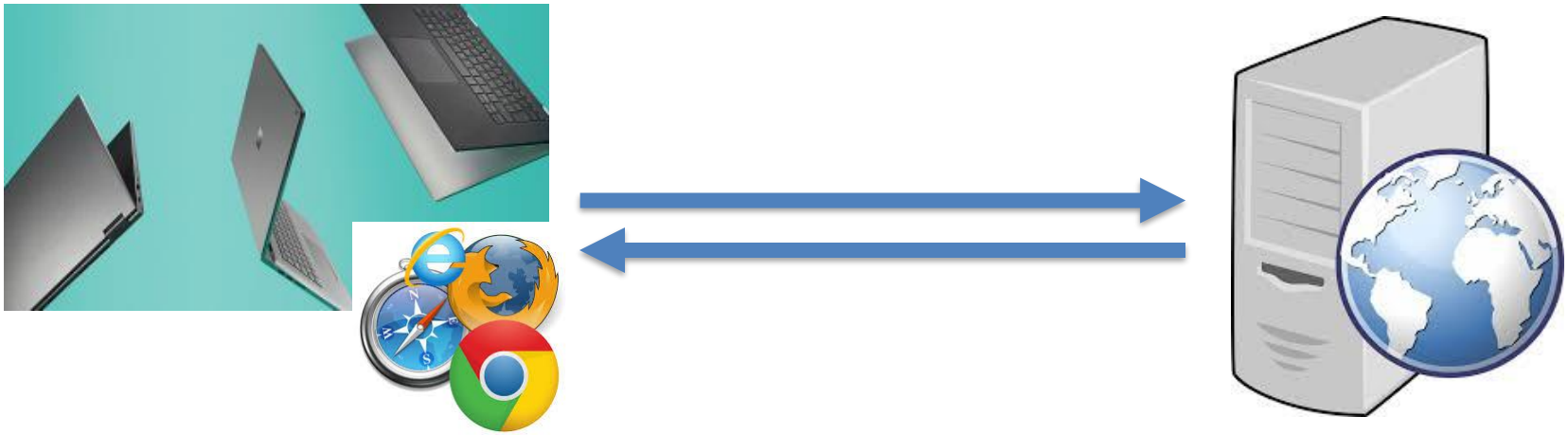
08. How the Web Works

Blase Ur and David Cash
January 24th, 2020
CMSC 23200 / 33250

THE UNIVERSITY OF CHICAGO

Your interface to the web

- Your web browser contacts a web server



A 10,000 Foot View of Technologies

- Where things run:

HTTP

HTML / CSS

JavaScript
(Angular/React)

Browser Extensions

CGI / PHP / Django
/ Node.js / Ruby on Rails

Databases (MySQL)

The Anatomy of a Webpage

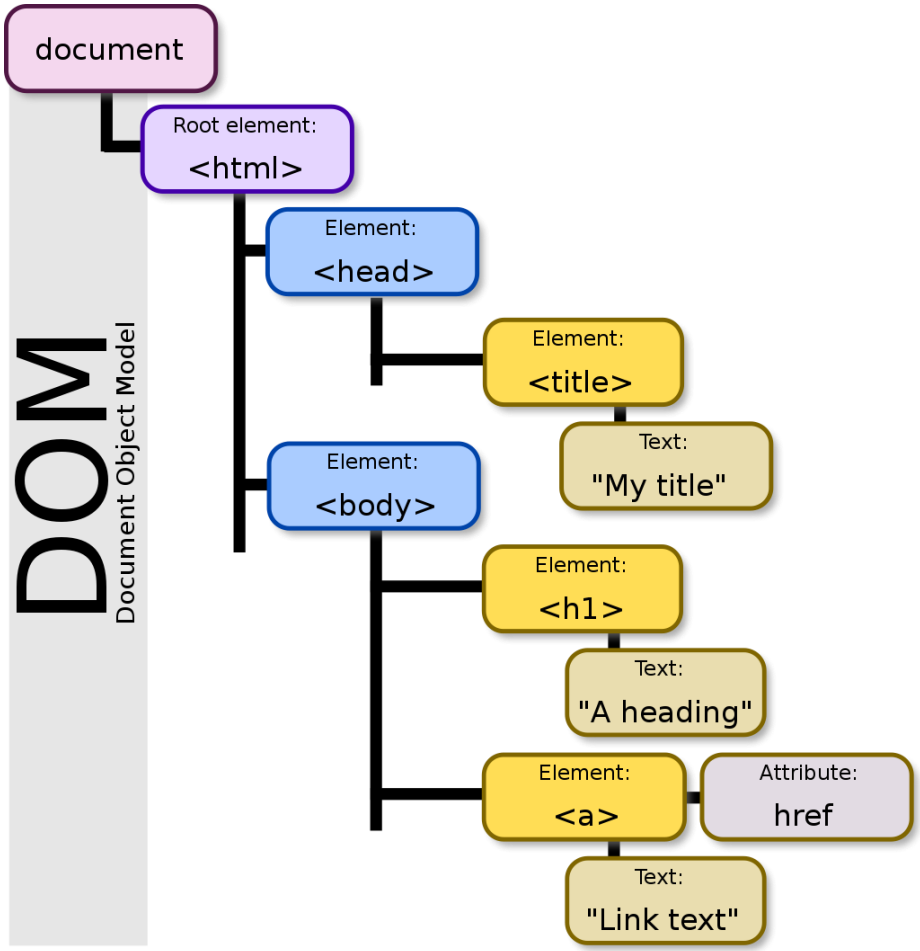
- view-source:https://www.cs.uchicago.edu/
- HTML (hypertext markup language)
  - Formatting of a page
  - All sorts of formatting:
  - Links: Click here
  - Pictures:
  - Forms
- HTML 5 introduced many media elements

The Anatomy of a Webpage

- CSS (cascading style sheets)
- <link href="/css/main.css?updated=20181020002547" rel="stylesheet" media="all">
- view-source:https://www.cs.uchicago.edu/css/main.css?updated=20181020002547

The Anatomy of a Webpage

- DOM (document object model)



You type uchicago.edu into Firefox

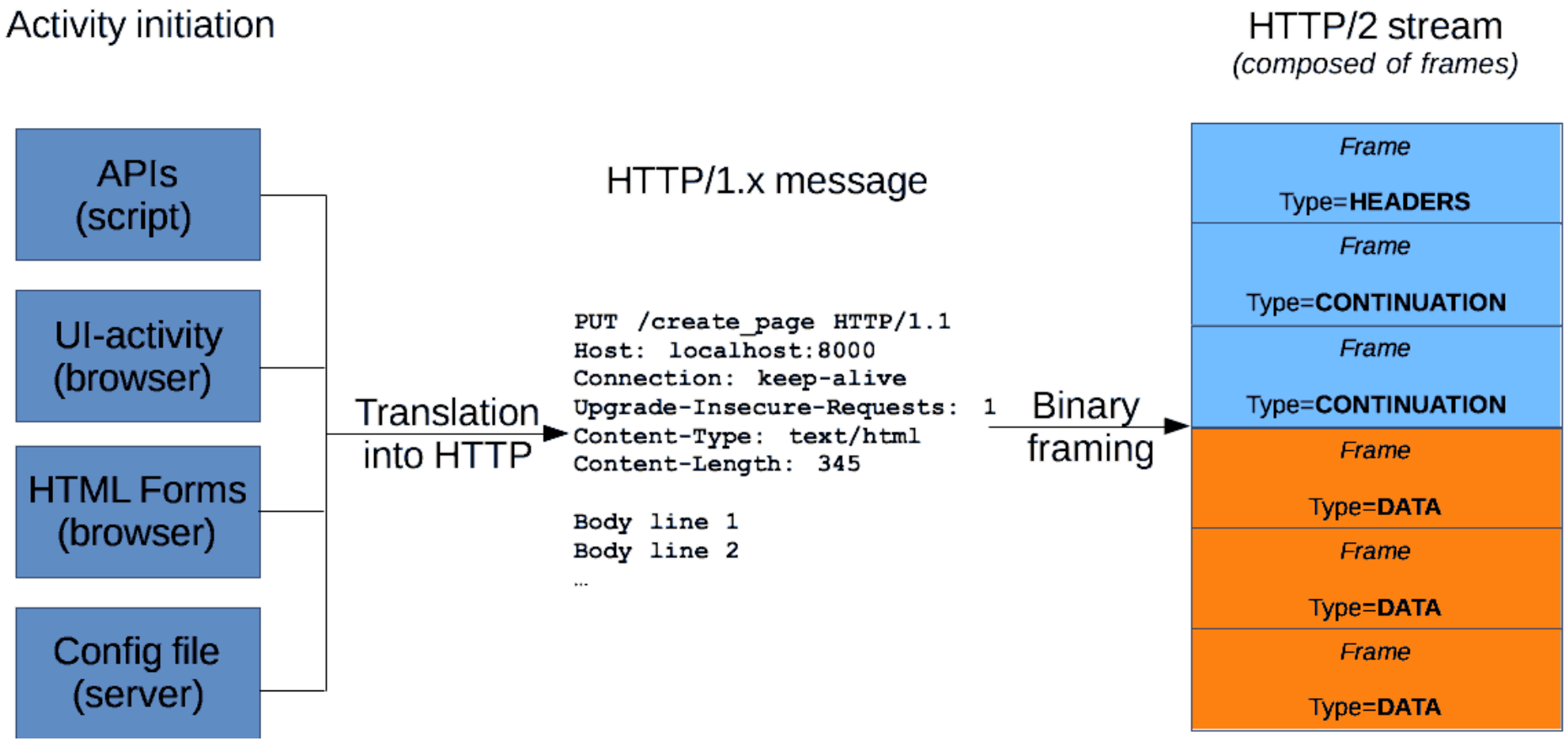
- DNS (domain name service)
  - Resolves to IP address 128.135.164.125
- URL (uniform resource locator)
- https://www.cs.uchicago.edu
  - Protocol: https
  - Hostname: www.cs.uchicago.edu
  - Filename: index.html or similar (implicit)

HTTP Request

- HTTP = Hypertext Transfer Protocol
- Start line: method, target, protocol version
  - GET /index.html HTTP/1.1
  - Method: GET, PUT, POST, HEAD, OPTIONS
- HTTP Headers
  - Host, User-agent, Referer, many others
  - https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers
- Body (not needed for GET, etc.)
- In Firefox: F12, “Network” to see HTTP requests

HTTP Request

- GET /index.html HTTP/1.1

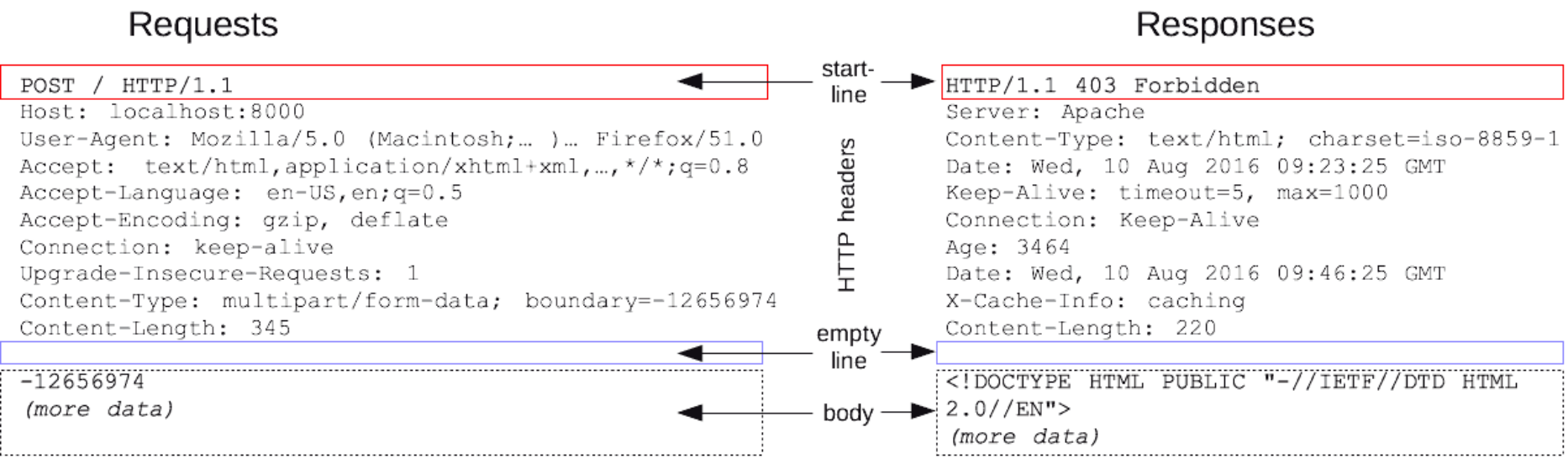


From https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages

HTTP Response

- Status
  - 200 (OK)
  - 404 (not found)
  - 302 (redirect)
- HTTP Headers
- Body

HTTP



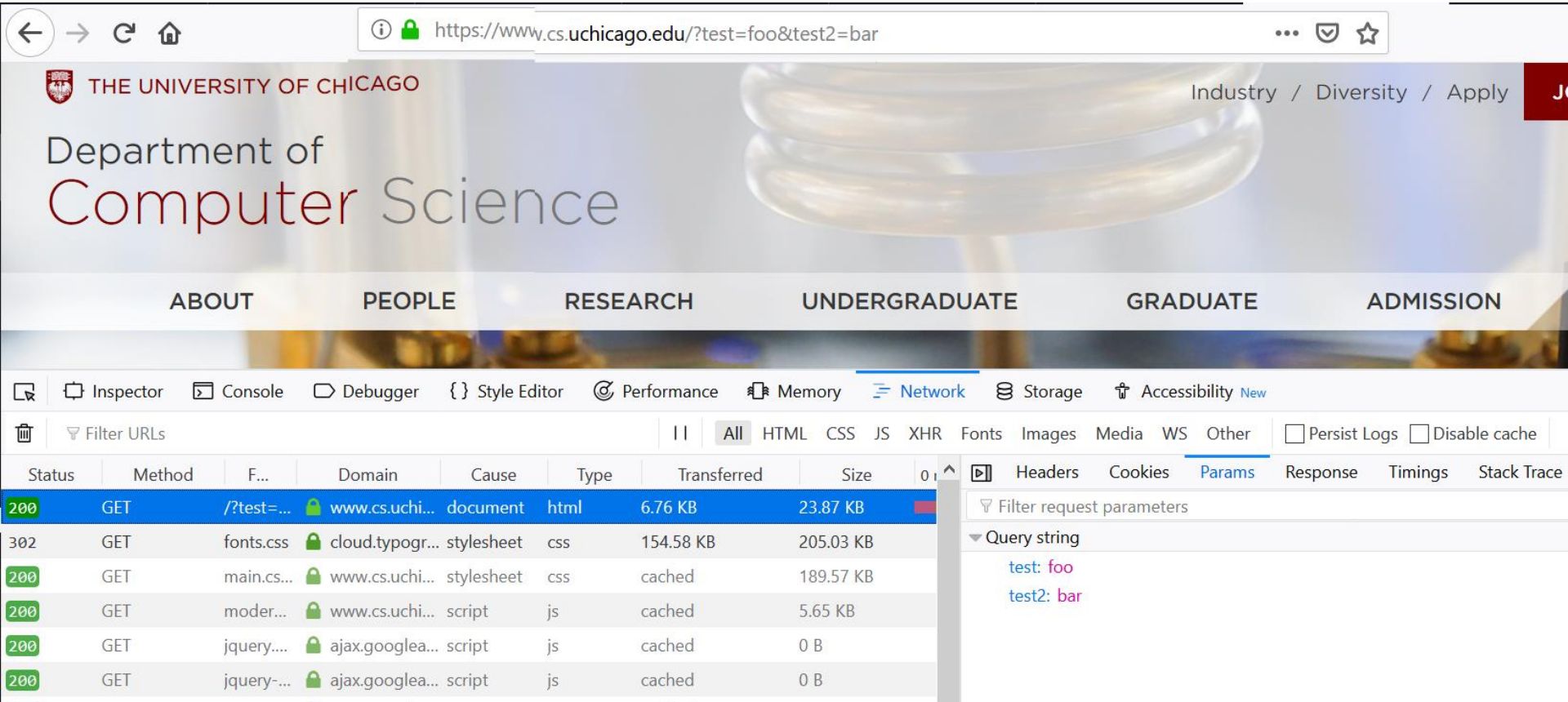
From https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages

Sending Data to a Server

- GET request
  - Data at end of URL (following “?”)
- POST request
  - Typically used with forms
  - Data *not* in URL, but rather (in slightly encoded form) in the HTTP request body
- PUT request
  - Store an entity at a location

URL Parameters / Query String

- End of URL
  - https://www.cs.uchicago.edu/?test=foo&test2=bar



Keeping State Using Cookies

- Cookies enable persistent state
- Set-Cookie HTTP header
- Cookie HTTP header
  - *Cookie: name=value; name2=value2; name3=value3*
- Cookies, once stored locally, are automatically sent with all requests your browser makes
- Session cookies vs. persistent cookies

Other Ways to Keep State

- Local storage
- Flash cookies
- (Many more)

HTTPS

- An extension of HTTP over TLS (i.e., the request/response itself is encrypted)
- Which CAs (certificate authorities) does your browser trust?
  - Firefox: Options → Privacy & Security → (all the way at the bottom) View Certificates
- How do you know if a cert is still valid
  - CRLs (certificate revocation lists)
  - OCSP (online certificate status protocol)

So… Interactive Pages?

- Javascript!
  - The core idea: Let's run (somewhat) arbitrary code on the **client's** computer
- Math, variables, control structures
- Imperative, object-oriented, or functional
- Modify the DOM
- Request data (e.g., through AJAX)
- Can be multi-threaded (web workers)

Common Javascript Libraries

- JQuery (easier access to DOM)
  - $(".test").hide() hides all elements with class="test"
- JQueryUI
- Bootstrap
- Angular / React
- Google Analytics *(sigh)*

What If You Make Poor Life Decisions?

Processing Data on the Server

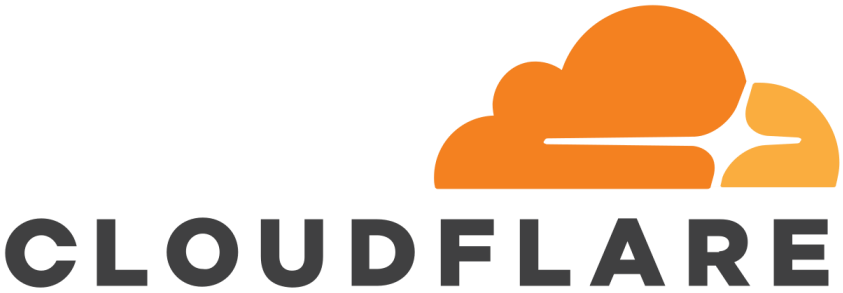
- Javascript is client-side
- Server-side you find Perl (CGI), PHP, Python (Django)
- Process data on the server
- What happens if this code crashes?

Storing Data on the Server

- Run a database on the server
- MySQL, SQLite, MongoDB, Redis, etc.
- You probably don’t want to allow access from anything other than *localhost*
- You definitely don’t want human-memorable passwords for these

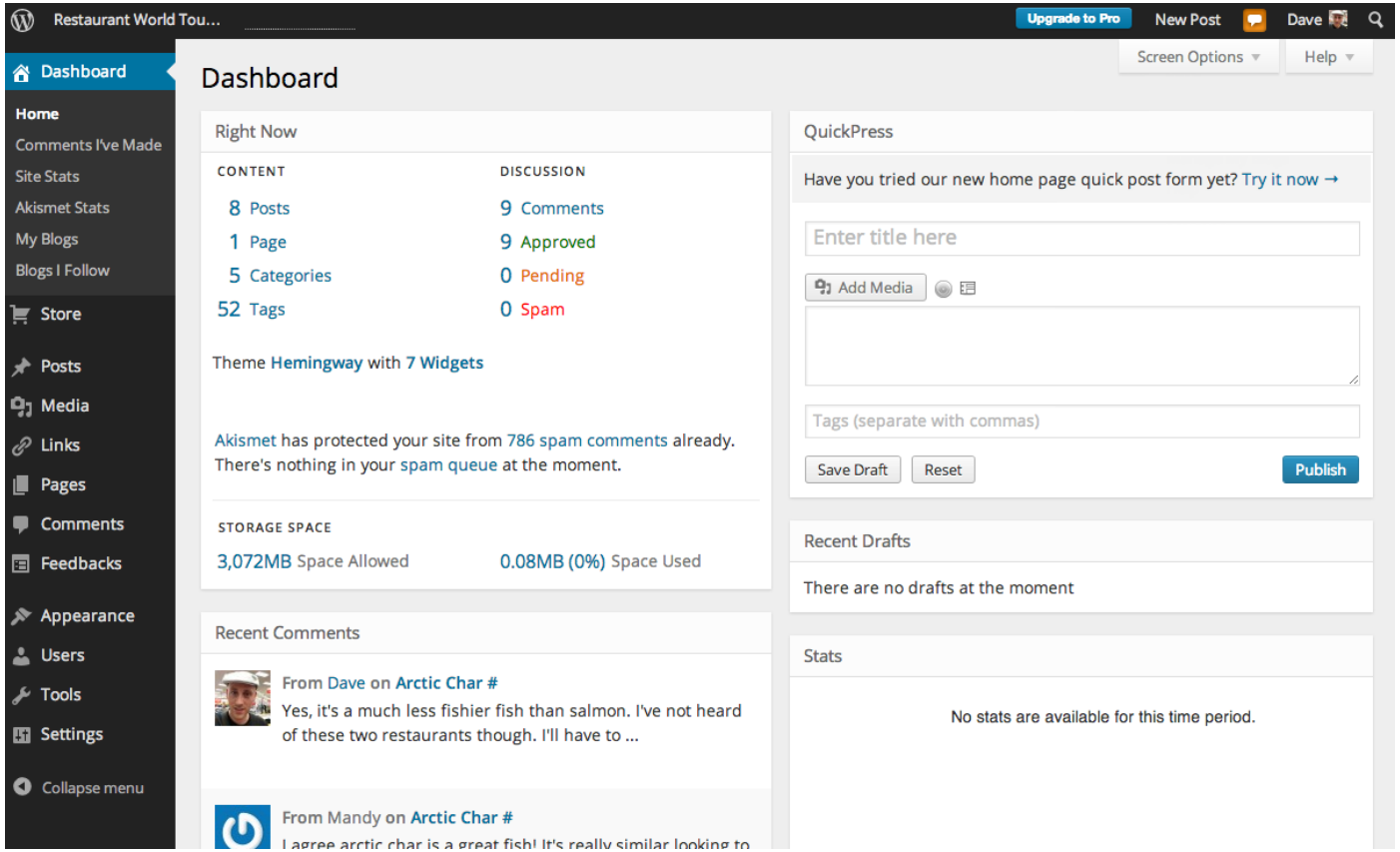
What If You Get Lots of Traffic?

- CDNs (content delivery networks)



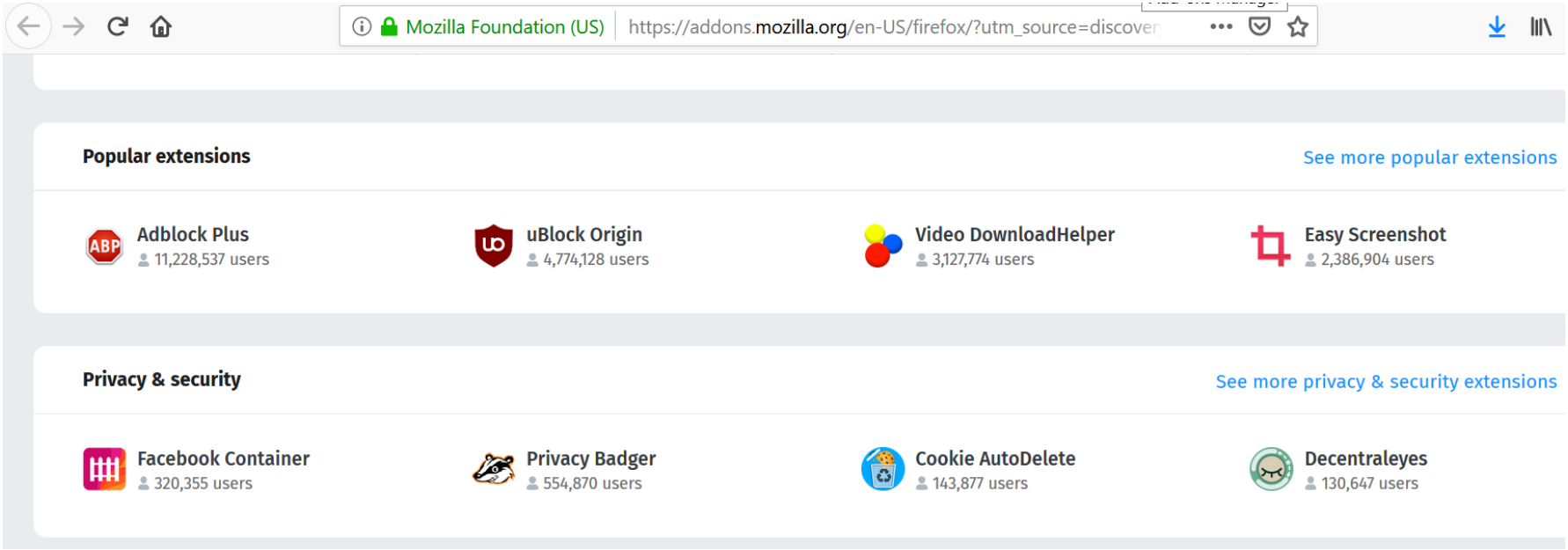
What If You Don’t Want To Code?

- CMS (content management system)
  - WordPress (PHP + MySQL), Drupal



Browser Extensions

- Can access most of what the browser can
- Requires permissions system
- Malicious extensions!



Same-Origin Policy

- Prevent malicious DOM access
- Origin = URI scheme, host name, port
- Only if origin that loaded script matches can a script access the DOM
  - Not where the script ultimately comes from, but what origin ***loads*** the script
- Frames / iframes impact origin
- CORS (Cross-Origin Resource Sharing)