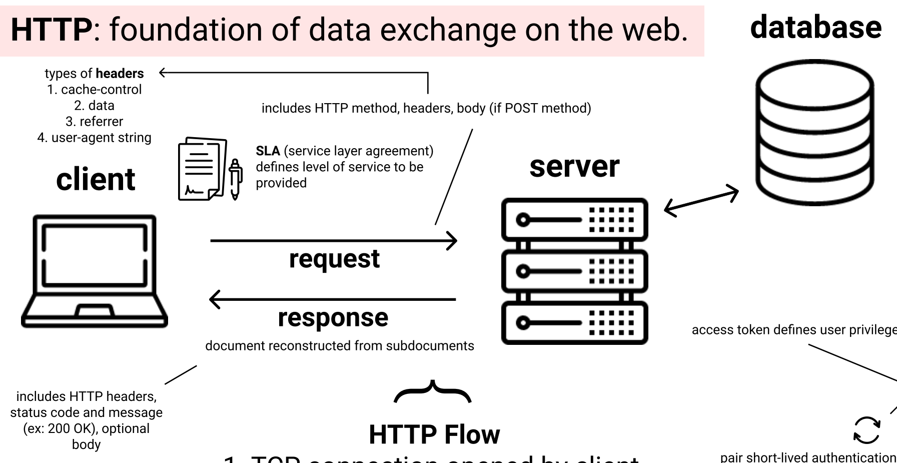
HTTP: foundation of data exchange on the web.



types of **headers**
1. cache-control
2. data
3. referrer
4. user-agent string

includes HTTP method, headers, body (if POST method)

SLA (service layer agreement) defines level of service to be provided

document reconstructed from subdocuments

includes HTTP headers, status code and message (ex: 200 OK), optional body

HTTP Flow
1. TCP connection opened by client.
2. HTTP message sent.
3. Response from server read.
4. Connection either closed or reused.

RESTful Route	**HTTP Method**	**URL**
Index	GET	/obj/
New	GET	/obj/new
Create	POST	/obj/
Show	GET	/obj/:id
Edit	GET	/obj/:id/edit
Update	PUT	/obj/:id
Destroy	DELETE	/obj/:id

GET just reads from resources, should have no side effects.
POST, PUT, and DELETE all affect resource, and may have side effects.

Cookies

could store in MongoDB or Redis

- data stored on user's browser
- keeps HTTP stateless (NOT sessionless), two different pages can share the same context

first-party ex: user authentication

third-party ex: ad tracking

OAuth = token-based authentication framework
- decouples authentication and authorization

access token defines user privileges

pair short-lived authentication tokens with refresh tokens, if too long lasting, will take longer to find out if compromised

Facebook provides authentication token, no password is shared

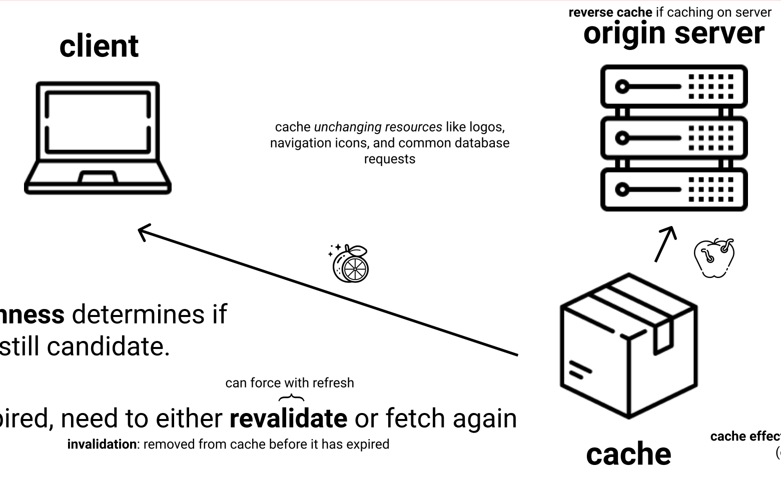
HTTP2 vs. **HTTP1.1**
- HTTP2 encapsulates all messages in binary form, HTTP1.1 uses plain-text
- HTTP2 can send multiple requests over one connection, so can download from server asynchronously

cache-control HTTP header

max-age: how long fetched responses valid for (**s-maxage** for shared caches)
public: can be cached by any cache
private: resource is user-specific (only cache on client)
no-cache: must revalidate each time
no-store: cannot store in any form — i.e. banking info
must-revalidate: if older than max-age, must revalidate
proxy-validate: if serving from shared cache, must revalidate

proxy = intermediate server, can cache, filter (parental controls), load-balance, authenticate, log

Caching: Storing responses to make subsequent requests faster



cache *unchanging resources* like logos, navigation icons, and common database requests

Freshness determines if still candidate.

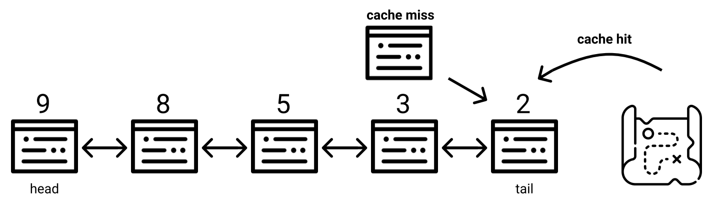
can force with refresh

if expired, need to either **revalidate** or fetch again

invalidation: removed from cache before it has expired

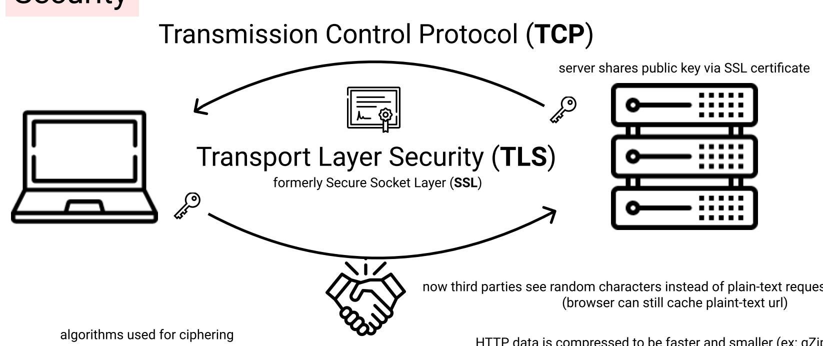
cache effectiveness measured with cache hit ratio (cache hits : requests made)

PROS:
Minimizers network traffic, improves perceived responsiveness, content available during network interruptions.



caches are often implemented using an LRU cache, with a doubly linked list and a hash map (**memcache**)

Security


Transmission Control Protocol (**TCP**)
Transport Layer Security (**TLS**)
formerly Secure Socket Layer (**SSL**)

server shares public key via SSL certificate

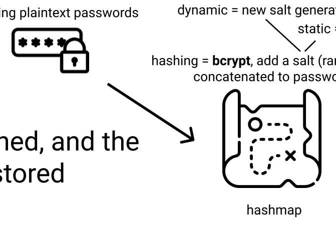
now third parties see random characters instead of plain-text requests and responses (browser can still cache plaint-text url)

algorithms used for ciphering

HTTP data is compressed to be faster and smaller (ex: gZip)

SSL Handshake:
1. Client sends SSL, ciphersuites, and compression versions.
2. Server sends over SSL certificate.
3. Client and server agree on session key.
4. Server verifies **MAC** and that data is being correctly decrypted.

message authentication code



no storing plaintext passwords

dynamic = new salt generated with each password
static = same for everyone

hashing = **bcrypt**, add a salt (random data concatenated to password)

passwords are hashed, and the hashmap is stored

HTTPS PROS:
Increased trust from users, increased integrity of data, better SEO
HTTPS CONS:
SSL costs money, if not done properly, may give scary warning message

cross-origin resource sharing
CORS: can access external API, override same-origin policy

XSS/malicious payload: client tries to inject malicious script; should sanitize input
cross-site scripting