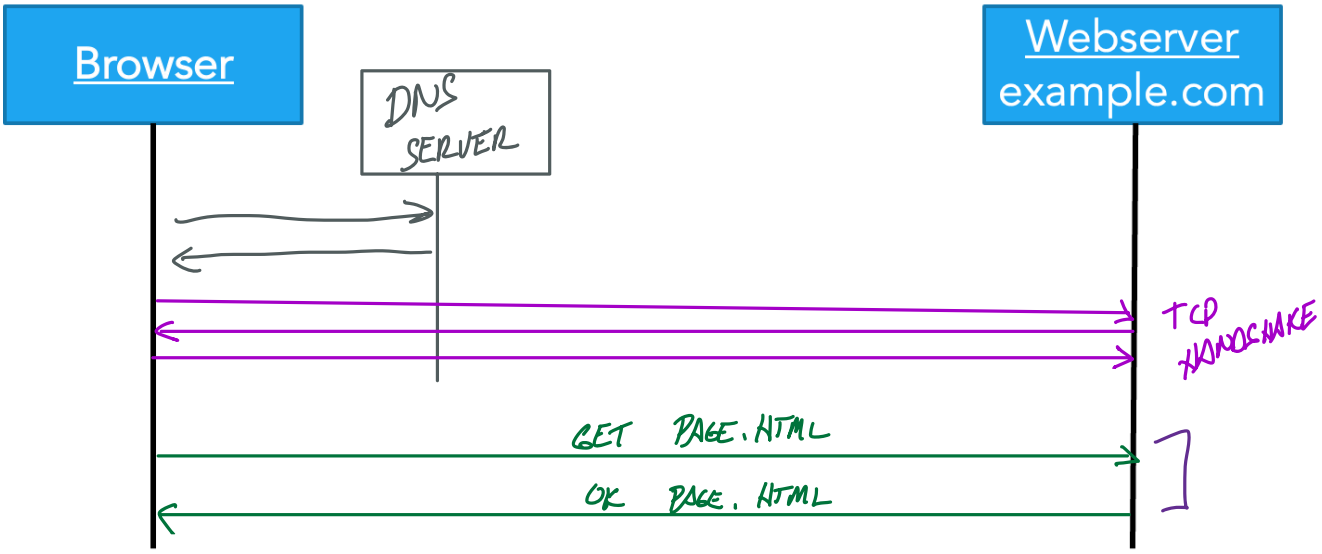
Lecture 20
- Making HTTP fast
- CDNs

Announcements
- TCP milestone II meetings this week
- Gearup III tonight 5pm in CIT 165 (+ recorded)

Lecture 20: HTTP II

Warmup: Suppose browser wants to fetch http://example.com/page.html

Assuming no caching, what is the minimum number of packets the browser needs to wait for?



Aside: HTTP status codes

Anatomy of a request

```
> nc cs.brown.edu 80
GET /courses/index.html HTTP/1.0
Host: cs.brown.edu
Content-Type: text/html
Accept-Language: en


HTTP/1.1 200 OK
Date: Thu, 24 Mar 2011 12:58:46 GMT
Server: Apache/2.2.9 (Debian) mod_ssl/2.2.9 OpenSSL/0.9.8g
Last-Modified: Thu, 24 Mar 2011 12:25:27 GMT
ETag: "840a88b-236c-49f3992853bc0"
Accept-Ranges: bytes
Content-Length: 9068
Vary: Accept-Encoding
Connection: close
Content-Type: text/html


<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
...
```

REQUEST

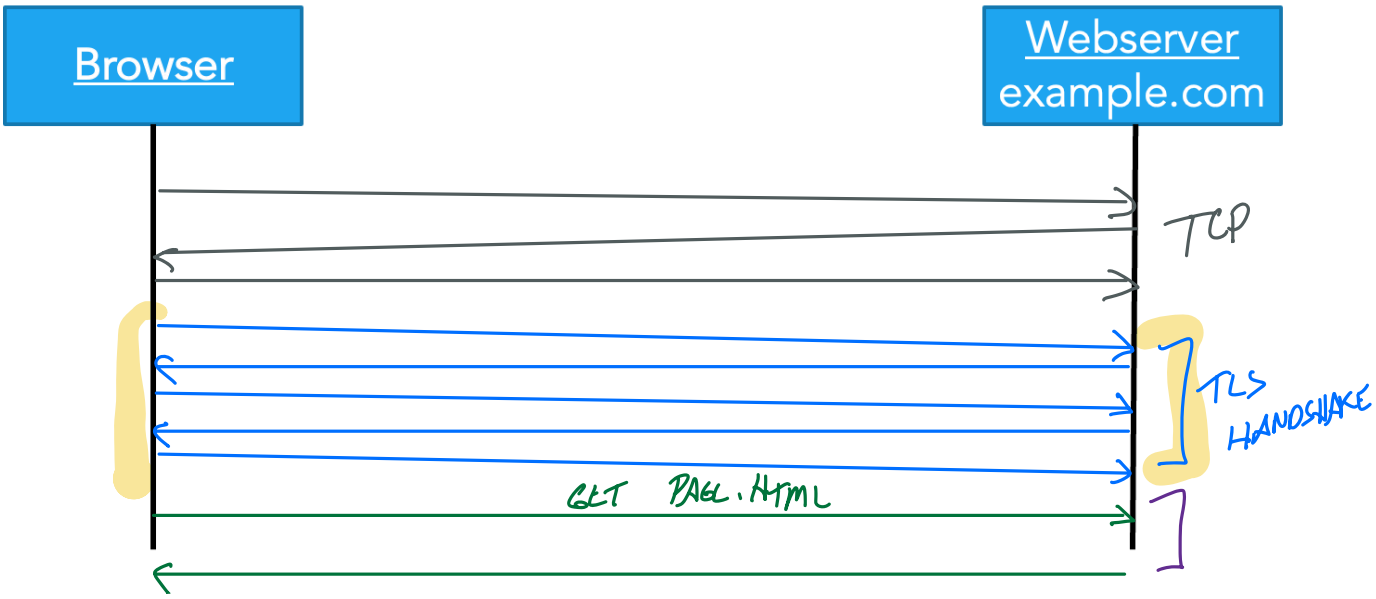
RESPONSE

HTTP Status codes: indicate something about the result

- 1xx: Information — e.g. 100 Continue
- 2xx: Success — e.g. 200 OK
- 3xx: Redirection — e.g. 301 Moved Permanently, 304 Not Modified
- 4xx: Client Error — 403 Forbidden (e.g. permissions error); 404 Not Found
- 5xx: Server Error — 500 Internal Server Error; 503 Service Unavailable

But it gets worse

Modern web traffic almost always uses HTTPS: https://example.com/page.html
=> Creates a secure transport layer to prevent eavesdropping, etc. (more on this later)
=> HTTPS uses the TLS protocol (Transport Layer Security), which we'll discuss soon! The details aren't important now, other than it has its own handshake process that requires 4 packets.



=> Many RTTs just to set up a connection!

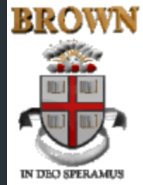
Why is this so important for performance? Consider how a browser loads a web page....

How does a browser load a page?

- Click a link, type in URL => browser fetches main page
- Main page has links to more resources => need to fetch these too!
  - Images, CSS, Javascript, etc.

- New resources might load yet more resources…

Recursive process with many dependencies!



Department of Computer Science

Welcome to the Brown University Computer Science Department Web. Information here is organized into broad categories, which are summarized in the icon bar, above. If you are visiting for th or exploring, the rest of this page offers some details about what you'll find.

If you are visiting us in person, you'll need directions to the CIT building. If not, perhaps you just need our address, phone, fax or other vital statistics.


Calendar of Events
Talks, conferences and soirees both at Brown and elsewhere are described.


Programs of Study
Undergraduate concentration requirements and the masters and phd programs are described, accompanied by the relevant forms, brochures and pointers to related information elsewhere.

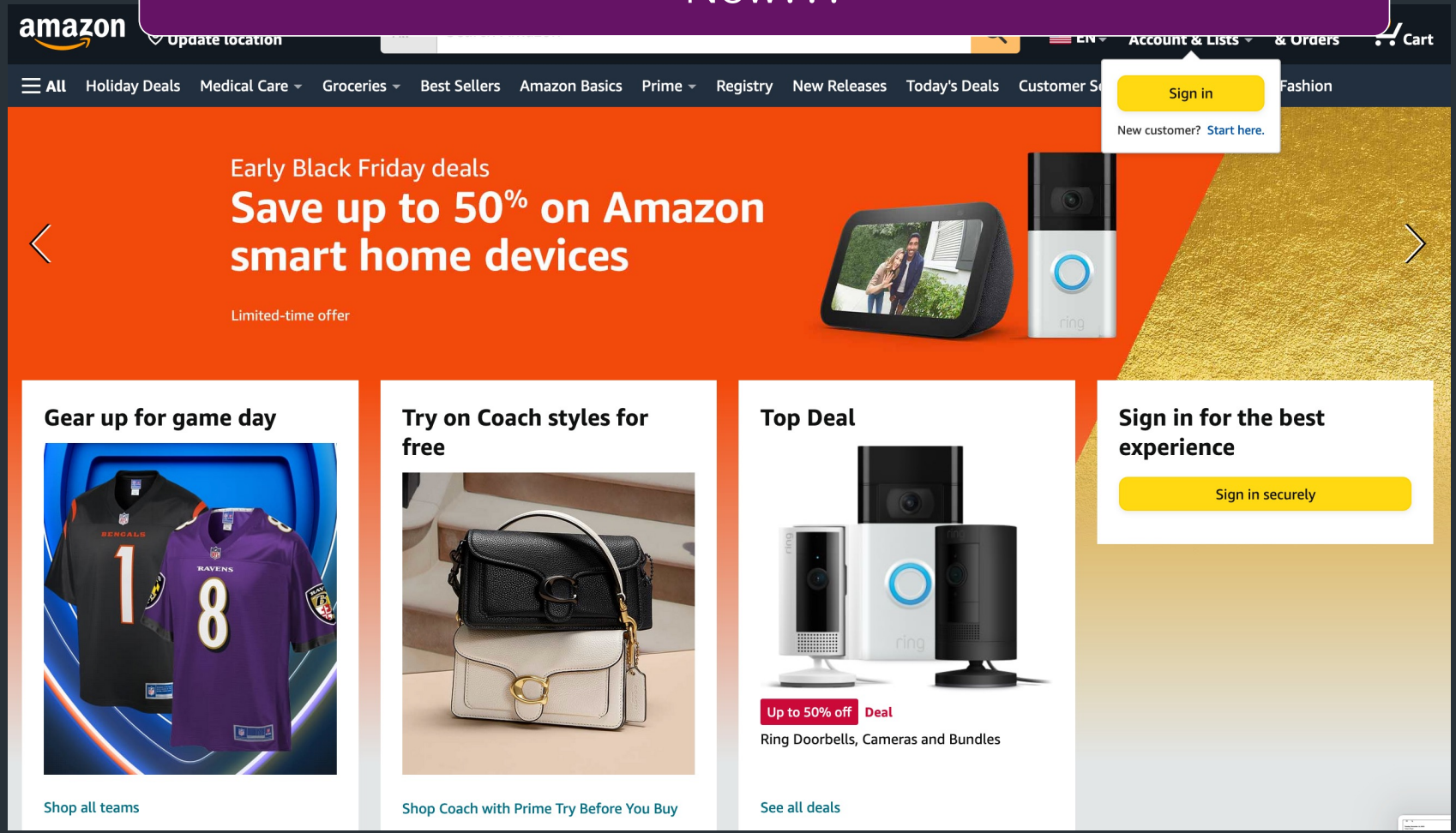

Research Groups
Active research areas in computer science at Brown include graphics, geometric computing, object-oriented databases, artificial intelligence and robotics. Each group maintains a home pag describing their research and activities and links to relevant publications.


Publications
The Department publishes brochures, technical reports, a newsletter, *conduit!*, and, for locals, house rules.


Courses
Many cou

Early websites: not many dependencies, usually served by one server

Now???


amazon
Update location
Account & Lists
& Orders
Cart
All
Holiday Deals
Medical Care
Groceries
Best Sellers
Amazon Basics
Prime
Registry
New Releases
Today's Deals
Fashion
Sign in
New customer? Start here.
Early Black Friday deals
Save up to 50% on Amazon smart home devices
Limited-time offer
Gear up for game day
Shop all teams
Try on Coach styles for free
Shop Coach with Prime Try Before You Buy
Top Deal
Up to 50% off
Deal
Ring Doorbells, Cameras and Bundles
See all deals
Sign in for the best experience
Sign in securely

On a modern webpage…

- Huge number of dependencies, external resources
  - … from many different locations, not just one server!

- Lots of asynchronous operations => loading new resources as you are using the page

- Lots of dynamic content => generated by the server specifically for you (your feed, ad data, …)

How to make this fast?

What does this mean for performance?

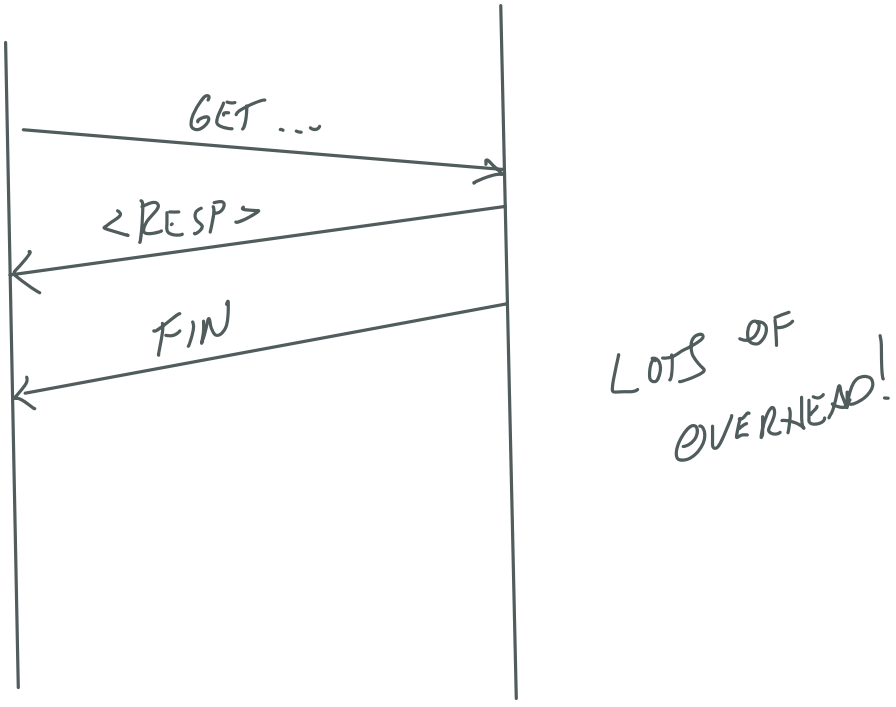
Consider: a modern webpage

Fetching a modern webpage involves a lot of requests
=> Lots of dependencies and external resources, often from many different servers
=> Lots of async operations as page loads, dynamic content for you (eg. your feed, ad data, ...)

How to make this fast? What matters most for performance?

HTTP/1.0: the problem

Many RTTs just to fetch one resource!
=> In original version, server closes connection as soon as it's done sending content

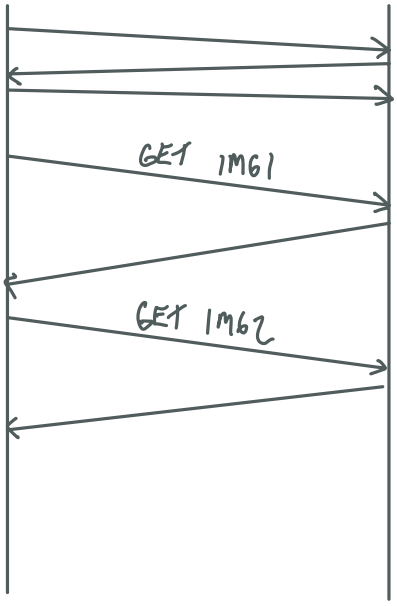


Can we do better?

HTTP/1.1 (1996): Persistent connections
=> Reuse TCP connection for multiple requests

```
GET / HTTP/1.1
Host: localhost:8000
User-Agent: Mozilla/5.0 . . .
[ . . . ]
Keep-Alive: 300
Connection: keep-alive
```

=> Can make more requests without repeating handshake



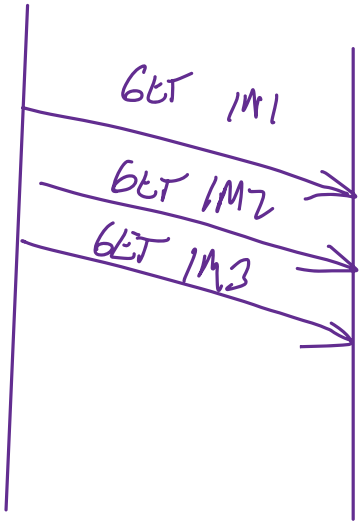
Problems??
=> Keeping some state on the server
=> How long?
=> Resource constraints on server
=> Lots of sequential requests
- Not great responsiveness
- Still one sequential stream (more on this later)

So what can we do about it?

Would like to have "pipelining": multiple requests "in-flight" at once

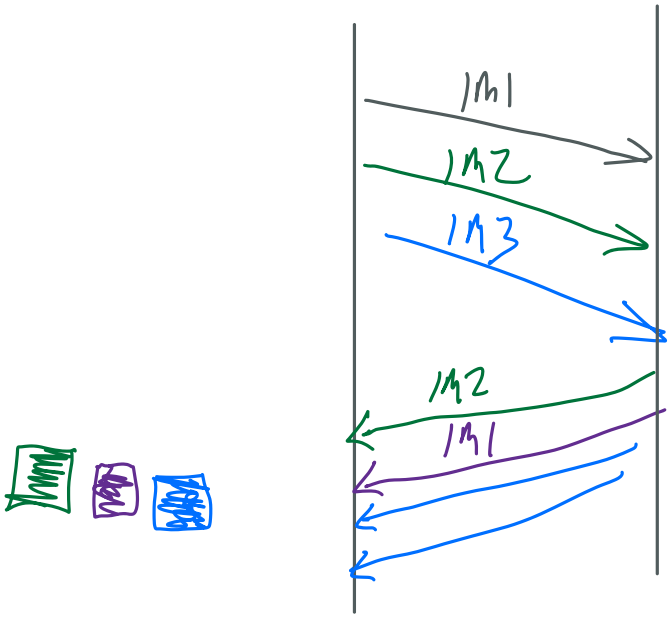
Two ways to do this
- Use multiple TCP connections in parallel
=> Browsers already do this
(e.g., pool of threads making requests)

- Change the HTTP protocol: create a way to make multiple requests in one connection
=> HTTP/2 HTTP/3



Enter: HTTP/2 (2015) (SPDY)

New variant, built on TCP
Adds support for multiplexed streams on one connection



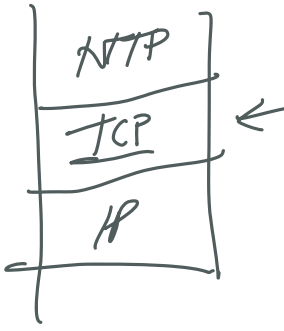
HTTP/2 lets us have multiple pending streams
=> All of this is part of one TCP connection

send(img2)
send(img1)
send(img3)

But what happens if you lose a packet??
TCP is designed to provide a single, ordered stream of bytes--it doesn't know about the multiple streams inside HTTP/2 (which are just sent via a normal TCP socket!
=> Therefore, if a packet is lost, other streams are on hold until packet arrives for the preceeding one
=> This concept is called head-of-line blocking: one request at the start of the queue holds up other requests that may already be done

What does this mean? HTTP/2 is encumbered by TCP's semantics
=> If a packet is lost, all streams suffer

Would like to decouple HTTP from the semantics of TCP
=> This is what HTTP/3 does



HTTP/3 (2022): QUIC

Internet Engineering Task Force (IETF) J. Iyengar, Ed.
Request for Comments: 9000 Fastly
Category: Standards Track M. Thomson, Ed.
Published: May 2021 Mozilla
ISSN: 2070-1721

QUIC: A UDP-Based Multiplexed and Secure Transport

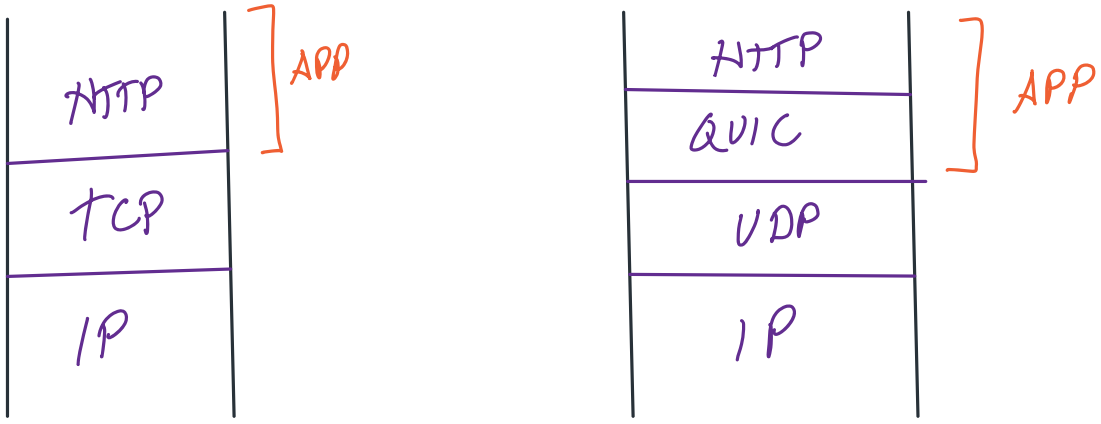
Abstract

This document defines the core of the QUIC transport protocol. QUIC provides applications with flow-controlled streams for structured communication, low-latency connection establishment, and network path migration. QUIC includes security measures that ensure confidentiality, integrity, and availability in a range of deployment circumstances. Accompanying documents describe the integration of TLS for key negotiation, loss detection, and an exemplary congestion control algorithm.

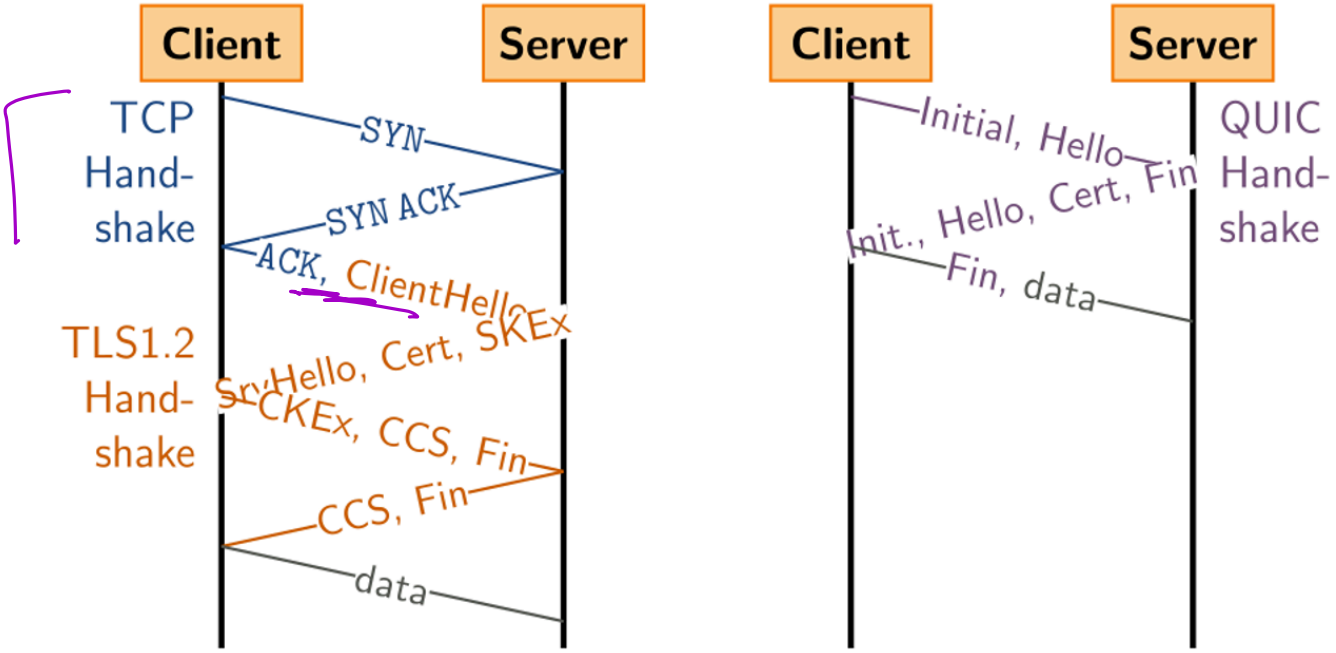
QUIC (RFC 9000): Newer transport layer protocol, same goals as TCP

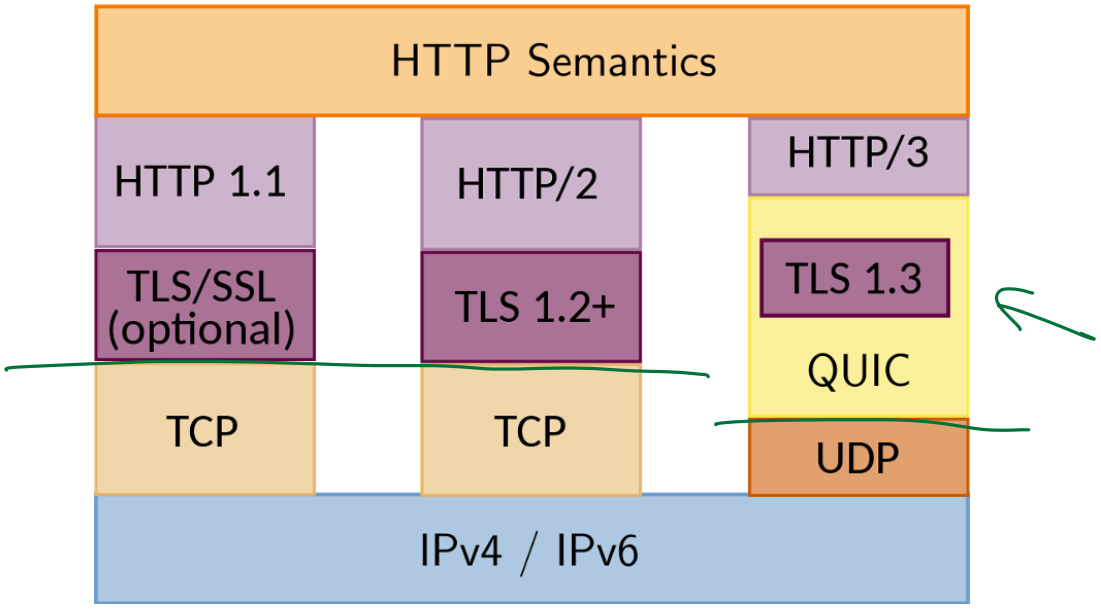
- Supports multiple streams at once
- Lots of tricks to reduce message size => Handshake is really short
  => Includes security by default

- Implemented via UDP, streams handled by application, not OS



=> Multiplexing in the transport layer => can be implemented in a way that benefits from HTTP => No head-of-line blocking!!
=> QUIC implemented as a library that is part of the application, rather than part of the kernel (as is typical for TCP)
=> allows application to extend/innovate/update faster





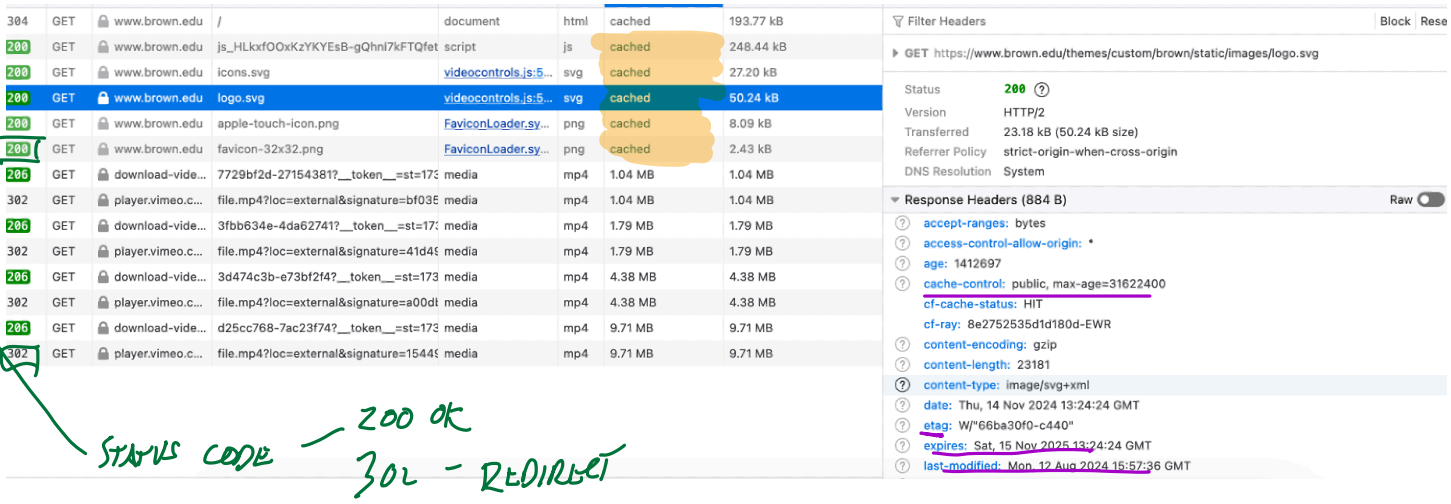
http://httpwg.org/specs/rfc7540.html

What else can we do to improve performance?

=> Caching!

Caching in the browser

What parts of this can be cached?
=> Can easily cache static content that doesn't change
=> Dynamic content can't be cached as easily



How to know what to cache?

- Headers returned with response
    - if caching possible, how long to cache
- Also possible to do "conditional requests"
    - Leverage "if-modified-since header"

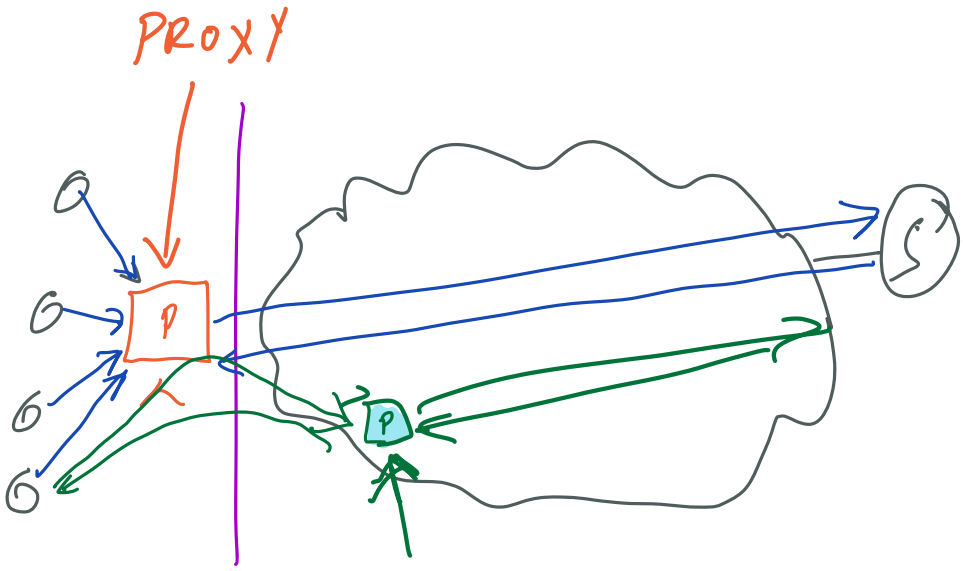
    => Server can either respond with data (if cache miss) or "304 not modified" to indicate data is unchanged

    => Still requires a request, but saves bandwidth

Caching in the network

=> Most classic form of Internet caching

Classic way: proxy cache

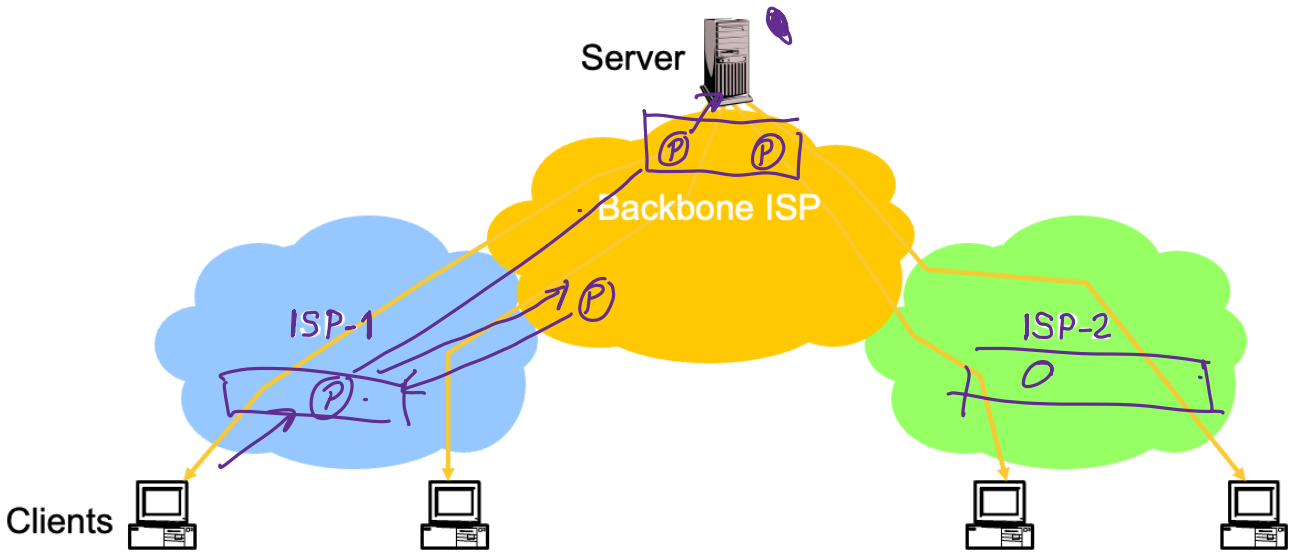


Proxy lives close to clients (locality)
Clients are configured to make request to proxy
Proxy does request on user's behalf (and acts as cache)

Proxies are also a way to do filtering, content blocking
=> Common in businesses or schools with security policies to block certain websites

Also a way to evade content blocking
=> Proxy servers that live on the public Internet, and make requests for yo
=> Example: connecting to clubpenguin.com in library at school

Caching *throughout* the network?



Reverse proxy: proxy server that lives inside the network, transparent to the client

=> Can provide caching => take advantage of locality
=> Can provide load balancing => distribute requests across regions
=> Transformations on data: transcoding, TLS, caching

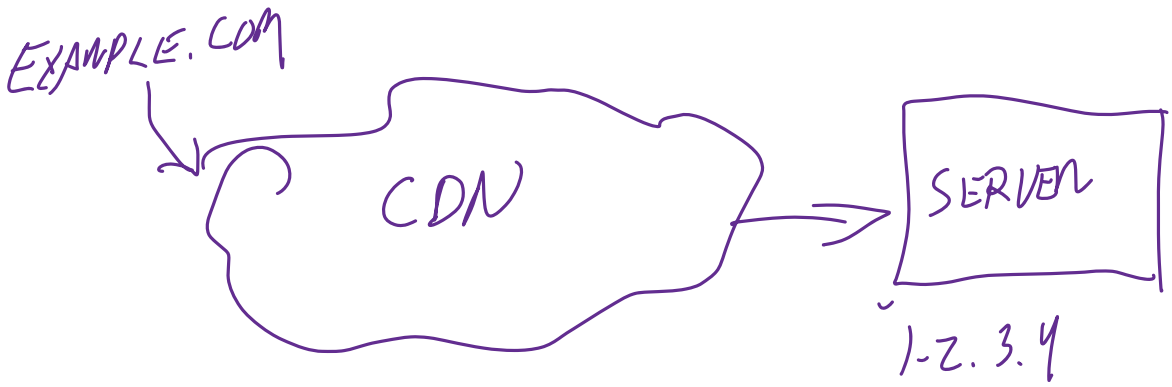
Some types also called accelerators

Can place caches at different locations in the network
- At the server / edge of server's network: reduce load on the server
- In the ISP's network: reduce traffic leaving the ISP
    => Beneficial for both the content provider and the ISP!

Content Distribution Networks (CDNs)

Companies that specialize in providing caching services (among other things)
=> Akamai, Cloudflare, ...

=> Have caches at many points across the network
=> Customer push data into CDN's caches
=> CDN redirects clients to their caches
=> CDN does lots of tricks to send users to "best" (usually closest) cache



Also useful for security => avoiding denial of service (DDoS) attacks where an attacker sends lots of request attempting to overwhelm your server
=> CDN networks are widely distributed => can absorb load of an attack
=> CDN has dedicated monitoring and tools to deal with these kind of events
=> Organizations outsource this important job to the CDN

Example: Brown's main website (www.brown.edu) uses Cloudflare

```
% dig www.brown.edu @10.1.1.10

; <<>> DiG 9.10.6 <<>> www.brown.edu @10.1.1.10
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 1051
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 3, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1220
;; QUESTION SECTION:
;www.brown.edu.            IN    A

;; ANSWER SECTION:
www.brown.edu.        3600    IN    CNAME    www.brown.edu.cdn.cloudflare.net.
www.brown.edu.cdn.cloudflare.net. 156 IN A    104.18.3.173
www.brown.edu.cdn.cloudflare.net. 156 IN A    104.18.2.173

;; Query time: 254 msec
;; SERVER: 10.1.1.10#53(10.1.1.10)
;; WHEN: Thu Apr 09 08:44:39 EDT 2026
;; MSG SIZE  rcvd: 120
```

Example

From Brown

```
dig www.bestbuy.com
;; ANSWER SECTION:
www.bestbuy.com.  3600  IN    CNAME www.bestbuy.com.edgesuite.net.
www.bestbuy.com.edgesuite.net. 21600 IN  CNAME a1105.b.akamai.net.
a1105.b.akamai.net.     20    IN    A     198.7.236.235
a1105.b.akamai.net.     20    IN    A     198.7.236.240
```

- Ping time: 2.53ms

From Berkeley, CA

```
a1105.b.akamai.net.     20    IN    A     198.189.255.200
a1105.b.akamai.net.     20    IN    A     198.189.255.207
```

- Ping time: 3.20ms

```
dig www.bestbuy.com
;; QUESTION SECTION:
;www.bestbuy.com. IN A

;; ANSWER SECTION:
www.bestbuy.com. 2530 IN CNAME www.bestbuy.com.edgekey.net.
www.bestbuy.com.edgekey.net. 85 IN CNAME e1382.x.akamaiedge.net.
e1382.x.akamaiedge.net. 16 IN A 104.88.86.223

;; Query time: 6 msec
;; SERVER: 192.168.1.1#53(192.168.1.1)
;; WHEN: Thu Nov 16 09:43:11 2017
;; MSG SIZE  rcvd: 123

traceroute to 104.88.86.223 (104.88.86.223), 64 hops max, 52 byte packets
 1  router (192.168.1.1)  2.461 ms  1.647 ms  1.178 ms
 2  138.16.160.253 (138.16.160.253)  1.854 ms  1.509 ms  1.462 ms
 3  10.1.18.5 (10.1.18.5)  1.886 ms  1.705 ms  1.707 ms
 4  10.1.80.5 (10.1.80.5)  4.276 ms  6.444 ms  2.307 ms
 5  lsb-inet-r-230.net.brown.edu (128.148.230.6)  1.804 ms  1.870 ms  1.727 ms
 6  131.109.200.1 (131.109.200.1)  2.841 ms  2.587 ms  2.530 ms
 7  host-198-7-224-105.oshean.org (198.7.224.105)  4.421 ms  4.523 ms  4.496 ms
 8  5-1-4.bear1.boston1.level3.net (4.53.54.21)  4.099 ms  3.974 ms  4.290 ms
 9  * ae-4.r00.bstnma07.us.bb.gin.ntt.net (129.250.66.93)  4.689 ms  4.109 ms
10  ae-6.r24.nycmny01.us.bb.gin.ntt.net (129.250.4.114)  8.863 ms  10.205 ms  10.477 ms
11  ae-1.r08.nycmny01.us.bb.gin.ntt.net (129.250.5.62)  9.298 ms
    ae-1.r07.nycmny01.us.bb.gin.ntt.net (129.250.3.181)  10.008 ms  8.677 ms
12  ae-0.a00.nycmny01.us.bb.gin.ntt.net (129.250.3.94)  8.543 ms  7.935 ms
    ae-1.a00.nycmny01.us.bb.gin.ntt.net (129.250.6.55)  9.836 ms
13  a104-88-86-223.deploy.static.akamaitechnologies.com (104.88.86.223)  9.470 ms  8.483
ms  8.738 ms
```

DNS query using local resolver in Providence

Enters CDN network in NYC

```
dig www.bestbuy.com @109.69.8.51

e1382.x.akamaiedge.net. 12 IN A 23.60.221.144


traceroute to 23.60.221.144 (23.60.221.144), 64 hops max, 52 byte packets
 1  router (192.168.1.1)  44.072 ms  1.572 ms  1.154 ms
 2  138.16.160.253 (138.16.160.253)  2.460 ms  1.736 ms  2.722 ms
 3  10.1.18.5 (10.1.18.5)  1.841 ms  1.649 ms  3.348 ms
 4  10.1.80.5 (10.1.80.5)  2.304 ms  15.208 ms  2.895 ms
 5  lsb-inet-r-230.net.brown.edu (128.148.230.6)  1.784 ms  4.744 ms  1.566 ms
 6  131.109.200.1 (131.109.200.1)  3.581 ms  5.866 ms  3.238 ms
 7  host-198-7-224-105.oshean.org (198.7.224.105)  4.288 ms  6.218 ms  8.332 ms
 8  5-1-4.bear1.boston1.level3.net (4.53.54.21)  4.209 ms  6.103 ms  5.031 ms
 9  ae-4.r00.bstnma07.us.bb.gin.ntt.net (129.250.66.93)  3.982 ms  5.824 ms  4.514 ms
10  ae-6.r24.nycmny01.us.bb.gin.ntt.net (129.250.4.114)  9.735 ms  12.442 ms  8.689 ms
11  ae-9.r24.londen12.uk.bb.gin.ntt.net (129.250.2.19)  81.098 ms  81.343 ms  81.120 ms
12  ae-6.r01.mdrdsp03.es.bb.gin.ntt.net (129.250.4.138)  102.009 ms  110.595 ms  103.010
ms
13  81.19.109.166 (81.19.109.166)  99.426 ms  93.236 ms  101.168 ms
14  a23-60-221-144.deploy.static.akamaitechnologies.com (23.60.221.144)  94.884 ms  92.779
ms  93.281 ms
```

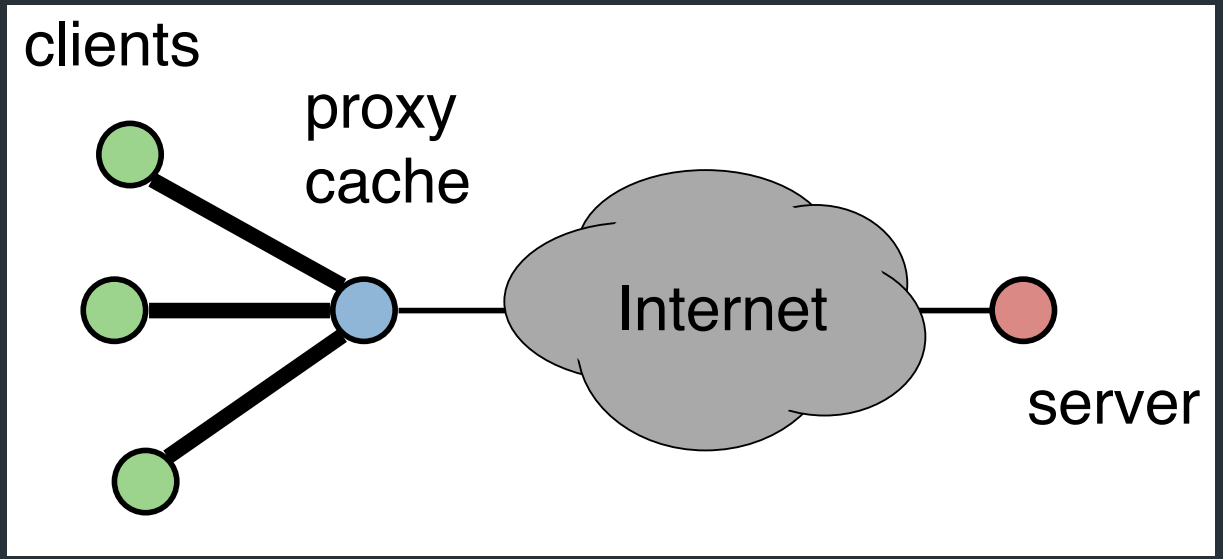
DNS query using reso in the UK

Traffic enters CDN in London

Can also see the difference with traceroute!

=> Extra written notes on CDNs (in case these are clearer)

Classic way: proxy cache



⇒ Client first sends traffic to proxy server, which forwards to Internet
⇒ Proxy acts as cache

Implications
- Cache data close to clients (locality)
- Can also use to enforce security policies, or circumvent them (eg. open proxies)

Content Distribution Networks (CDNs)

Companies that specialize in providing caching services (among other things)

=> Akamai, Cloudflare, …

- Provides caching throughout network
- Can also do some processing

Content Distribution Networks (CDNs)

Companies that specialize in providing caching services (among other things)

=> Akamai, Cloudflare, …

- Provide both caching throughout network
  - Pull: result from client requests
  - Push: expectation of high access rates to some objects
- Can also do some processing
  - Deploy code to handle some dynamic requests
  - Can do other things, such as transcoding

How Akamai works

Akamai has cache servers deployed close to clients

- Co-located with many ISPs

- Challenge: make same domain name resolve to a proxy close to the client

- Lots of DNS tricks. BestBuy is a customer
  - Delegate name resolution to Akamai (via a CNAME)

Other CDNs

- Akamai, Limelight, Cloudflare
- Amazon, Facebook, Google, Microsoft
- Netflix
- Where to place content?
- Which content to place? Pre-fetch or cache?