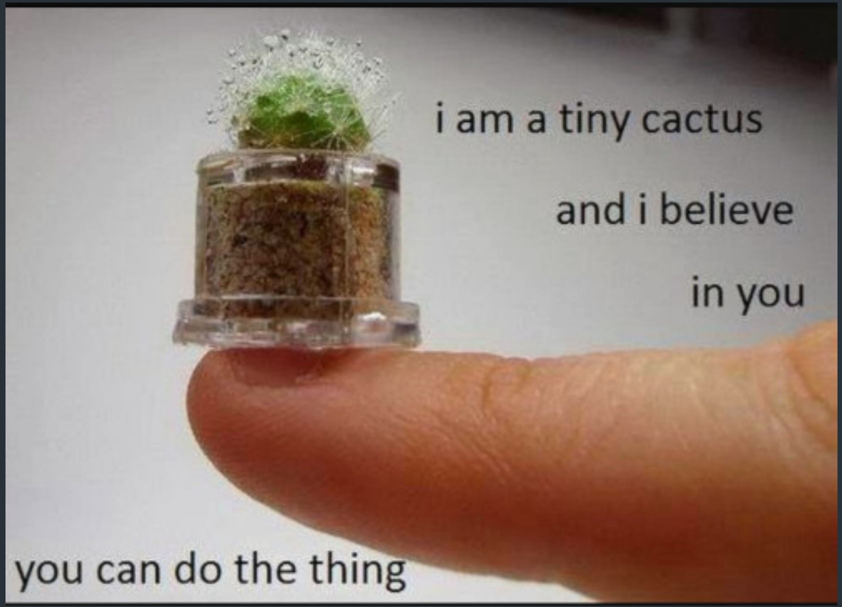
# CSCI-1680
# HTTP II

Nick DeMarinis

Based partly on lecture notes by Rodrigo Fonseca, Scott Shenker and John Jannotti

# Administrivia

- TCP is due next Tuesday

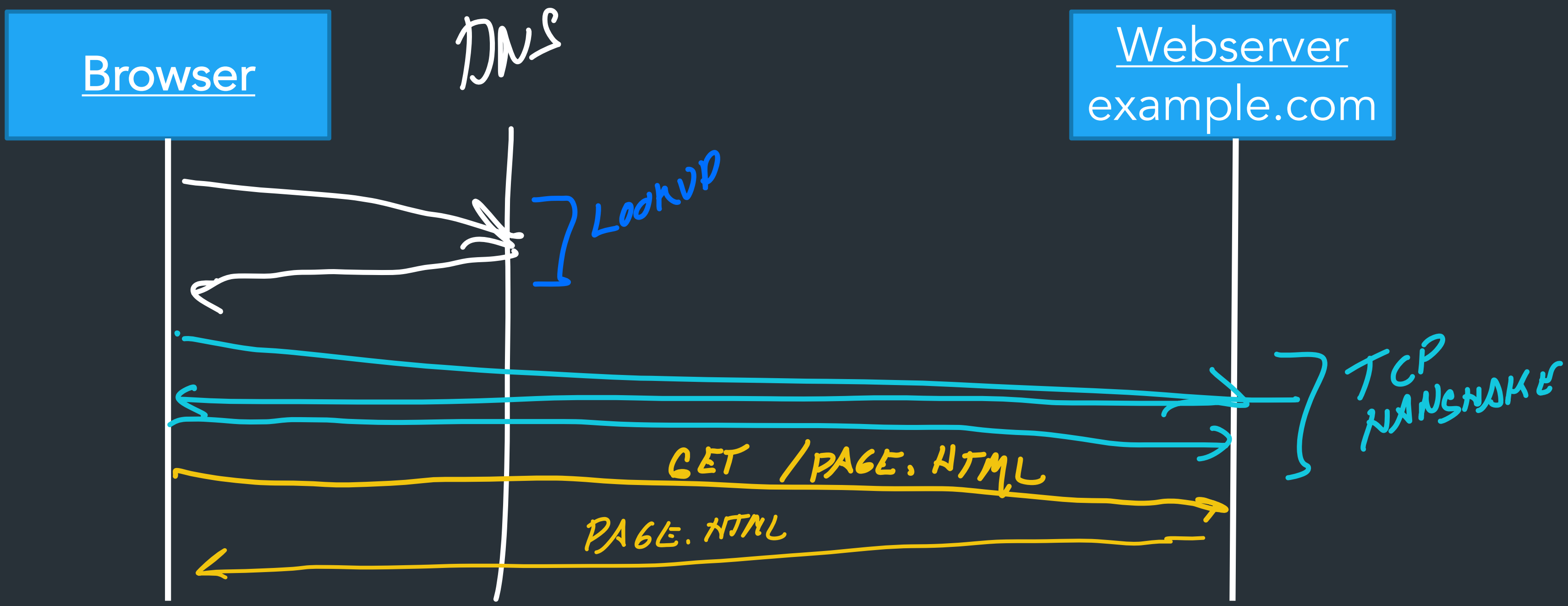



Will announce some final project info, grading feedback soon

# Warmup

Browser wants to fetch: http://example.com/page.html

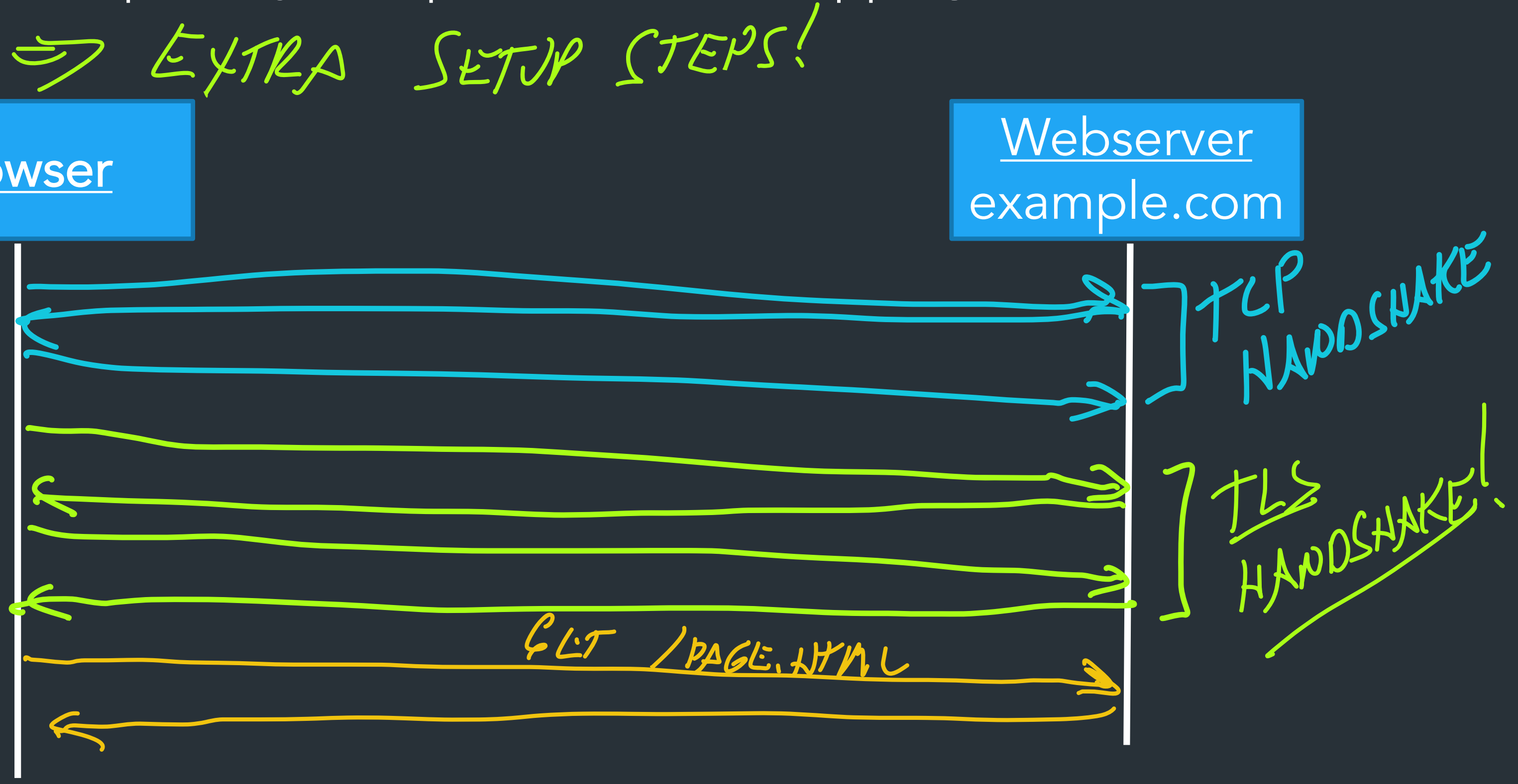
Assuming no caching, what is the minimum number of packets the browser needs to wait for?



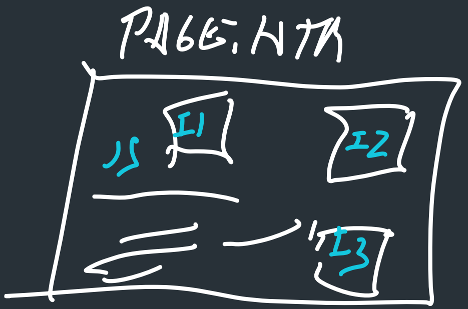
# It gets worse

Modern web traffic almost always uses HTTPS: https://example.com/page.html

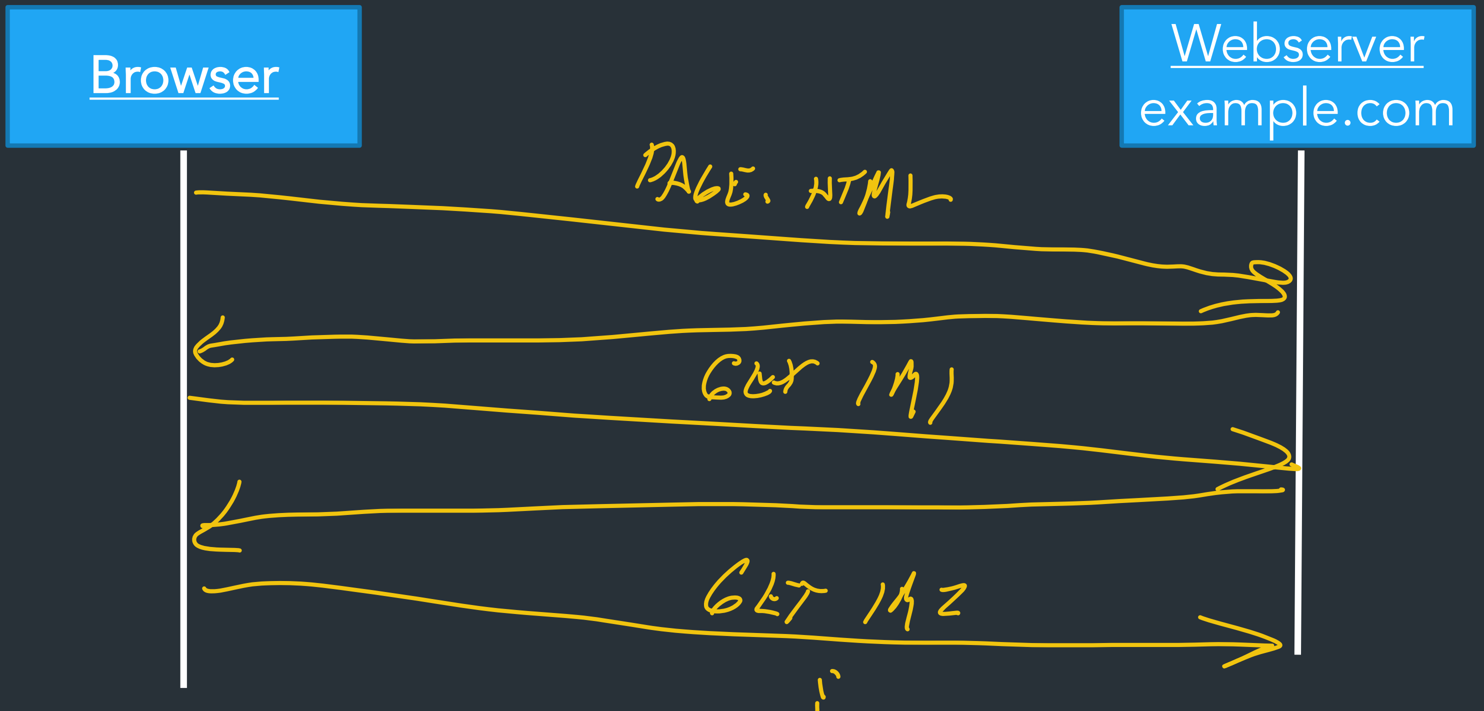
=> Creates a secure transport layer to prevent eavesdropping, etc (more on this later)



# 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.



# 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!

# BROWN Department of Computer Science

IN DEO SPERAMUS

- FEW RESOURCES
- MOST ALL FROM ONE SERVER.

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.

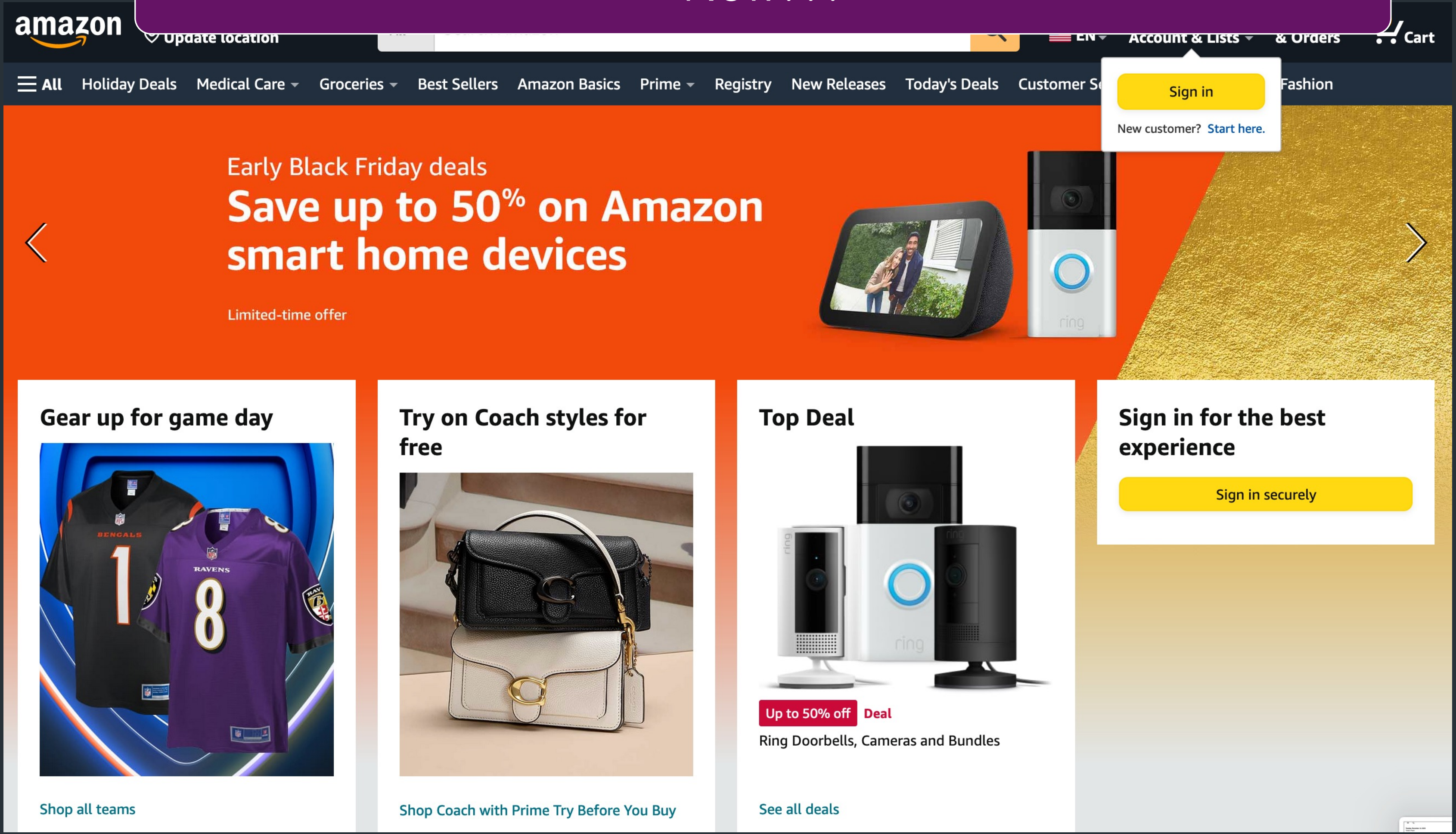
**Research Groups**
Active research areas in computer science at Brown include graphics, geometric computing, object-oriented databases, artificial intelligence and robotics. Each group maintains a home pag
describing their research and activities and links to relevant publications.

**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
EN
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 COMPUTATION

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

How to make this fast?

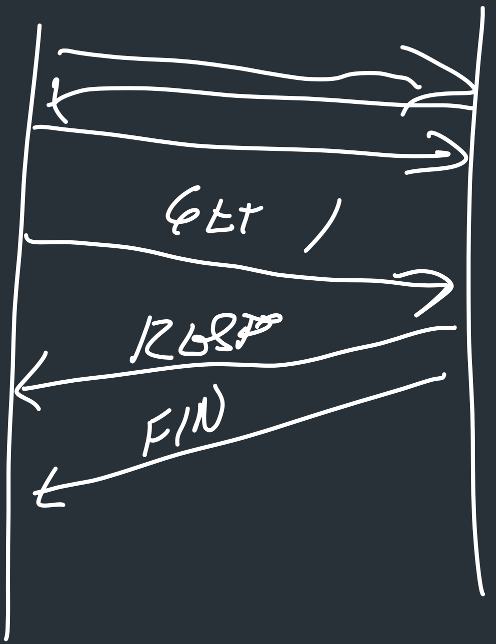
*How to make this fast?*

*What's important for performance?*

# Observation: lots of small requests

Latency is a problem! Need many RTTs just to fetch one resource!

HTTP/1.0: One TCP connection per request!




- LOTS OF OVERHEAD
- LOTS OF LATENCY

# Can we do better?

(1997)

HTTP/1.1: Persistent connections

=> Reuse TCP connection to for multiple requests

PROBLEMS
- MULTIPLE SERVERS.
- CACHING,
- REQUESTS IN ORDER


- NOT TRULY STATELESS ANYMORE.
- BUT NO HANDSHAKE EVERY TIME.

PROBLEMS?

# Can we do better?

HTTP/1.1: Persistent connections

=> Reuse TCP connection to for multiple requests

Problems?

# Can we do better?

HTTP/1.1 (1996): Persistent connections

=> Reuse TCP connection to for multiple requests

Problems?

⇒ One big request blocks others => head of line blocking

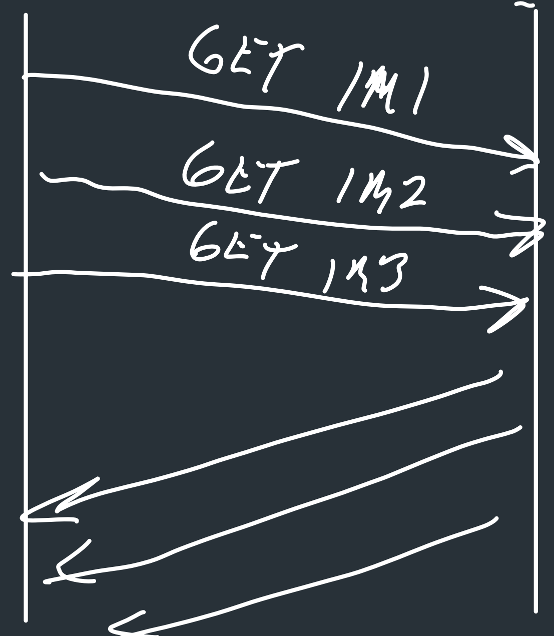
=> Same if connection has packet loss

=> Doesn't help when fetching from multiple locations

# What can be done?

GOAL: PIPELINING

— MULTIPLE CONNECTIONS — BROWSERS HAVE A POOL OF REQUEST THREADS (MULTIPLE SERVERS)




— WOULD LIKE TO HAVE MULTIPLE STREAMS ON SAME TCP CONNECTION.

# HTTP/1.1 Request

```
GET / HTTP/1.1
Host: localhost:8000
User-Agent: Mozilla/5.0 (Macinto ...
Accept: text/xml,application/xm ...
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
```

# What can be done?

Pipelining: have multiple “in-flight” requests at once

Two methods

- Multiple TCP connections in parallel

- Change the HTTP protocol: multiple requests per connection

# What can be done?

Pipelining: have multiple “in-flight” requests at once

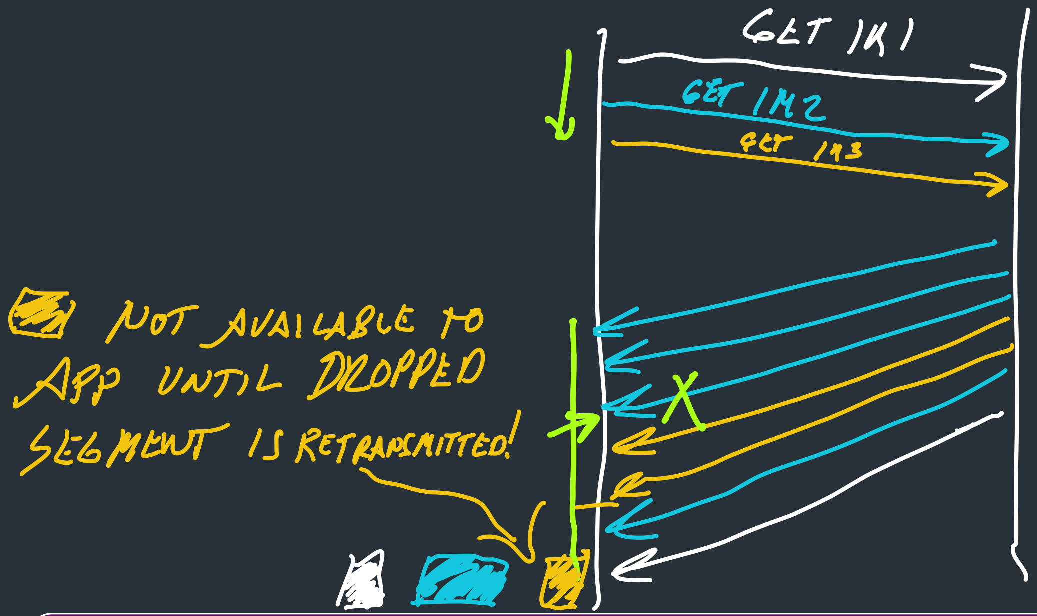
Two methods

- Multiple TCP connections in parallel
  => Browsers often do this (up to a limit)

- Change the HTTP protocol: multiple requests per connection
  => Newer HTTP versions: HTTP/2, HTTP/3

# HTTP/2 (2015)

Adds support for multiplexed streams on one connection




TCP + MULTIPLEX

TCP doesn’t know about multiple streams
=> If packet loss on one stream, others are blocked until packet comes in
=> Head of line blocking

What happens if a packet gets dropped?

https://www.twilio.com/blog/2017/10/http2-issues.html

# HTTP/2 (2015)

Adds support for multiplexed streams on one connection

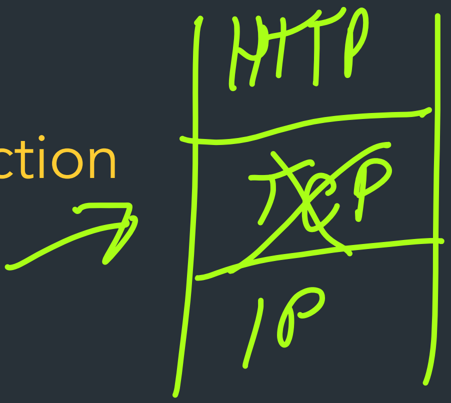
TCP provides a  single, ordered byte stream
=> doesn’t know about multiple connections!

Encumbered by TCP’s semantics:
If a packet is lost, all streams suffer! 😭 😭 😭

https://www.twilio.com/blog/2017/10/http2-issues.html

# HTTP/2 (2015)

Adds support for multiplexed streams on one connection




TCP provides a single, ordered byte stream

=> doesn't know about multiple connections!

=> WOULD LIKE TO DECOUPLE HTTP FROM TCP SEMANTICS

Encumbered by TCP's semantics:
If a packet is lost, all streams suffer! 😭 😭 😭

=> Head of line blocking

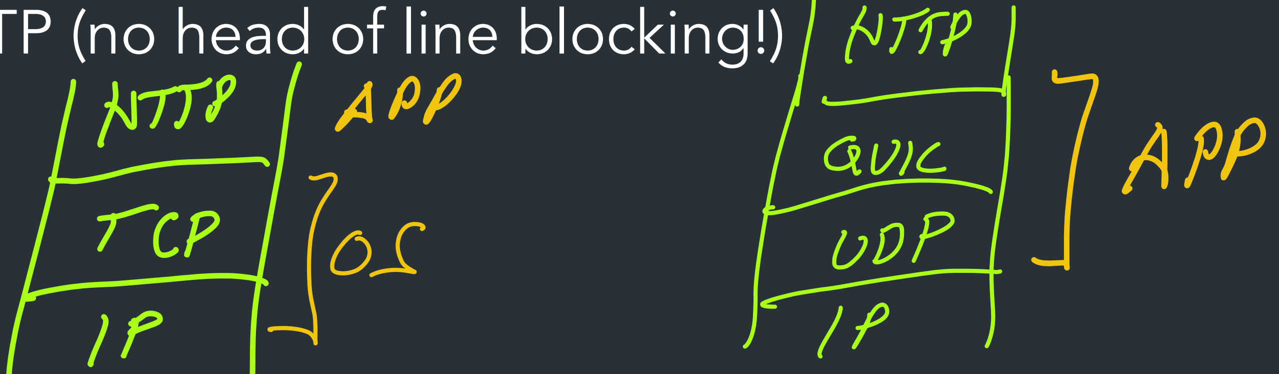
https://www.twilio.com/blog/2017/10/http2-issues.html

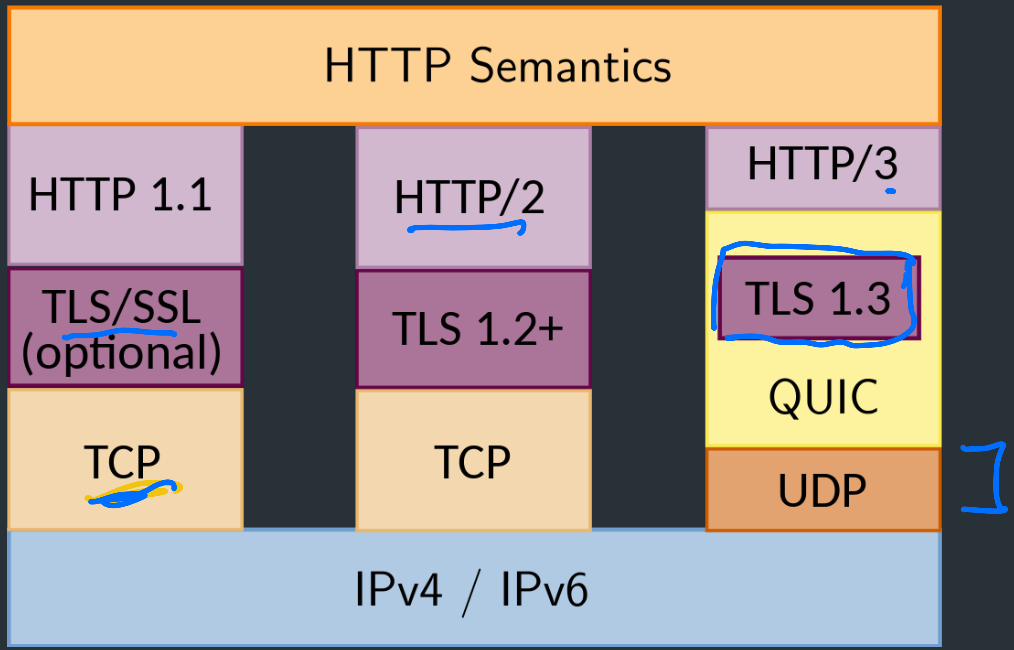
# HTTP/3 (2022): HTTP + QUIC

QUIC (RFC9000): Newer transport-layer protocol, same goals as TCP

- Supports multiple streams at once
- Various tricks to reduce message size and latency → SHORTER HANDSHAKE
- Integrates security by default (TLS)

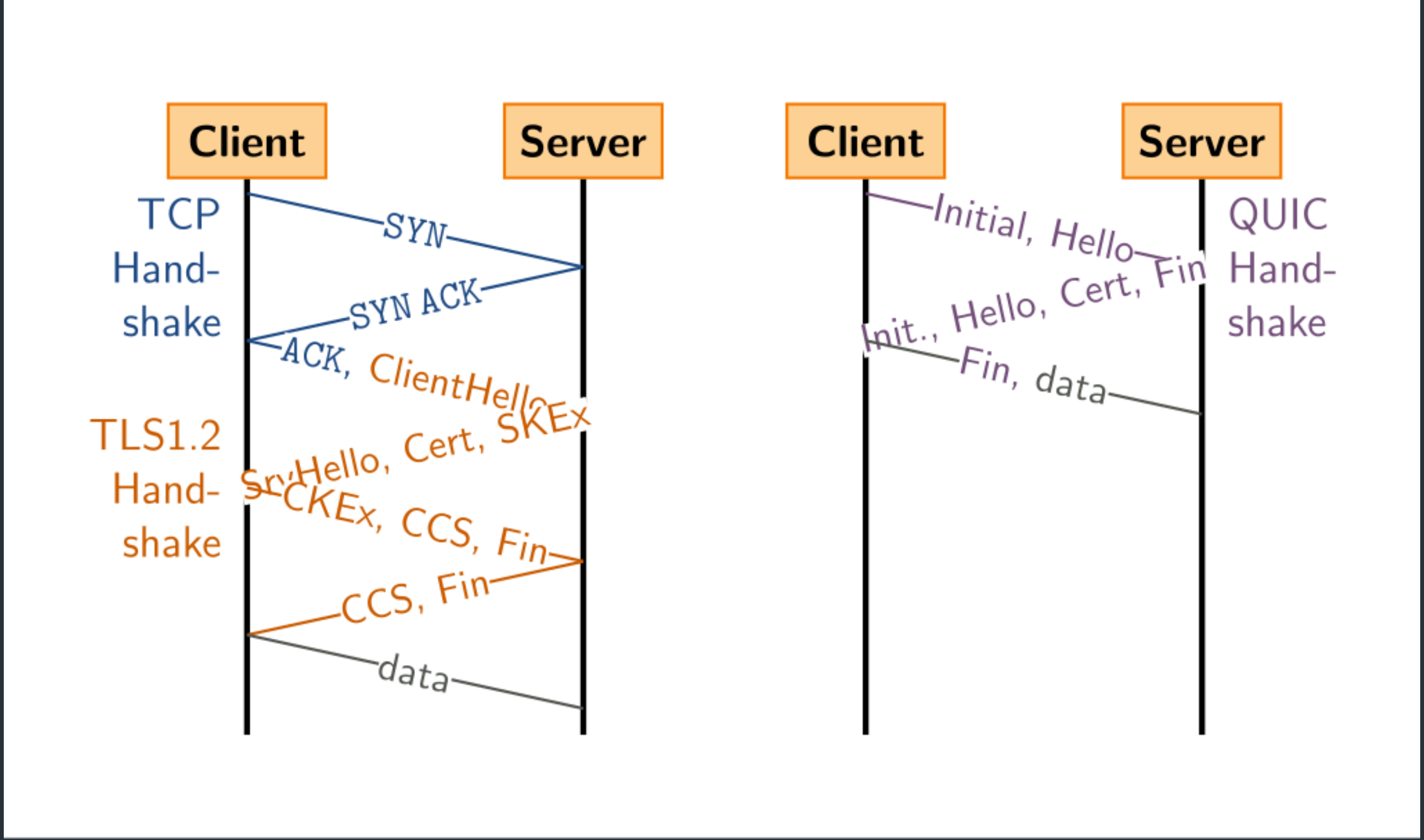
- By moving multiplexing into the transport layer, can do so in a way that benefits HTTP (no head of line blocking!)






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

# Comparison: QUIC's handshake

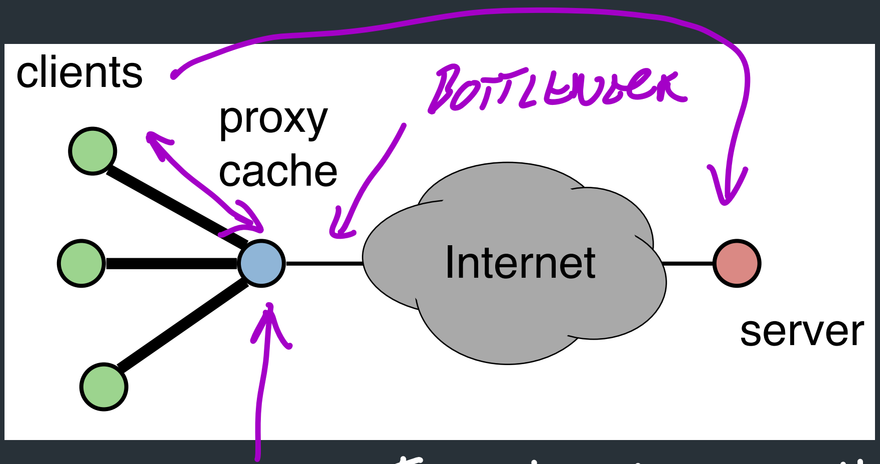


# What else can we do?

# Caching

Place caches throughout network

- Use locality: closer to clients => lower latency
- Improve throughput by avoiding bottleneck links



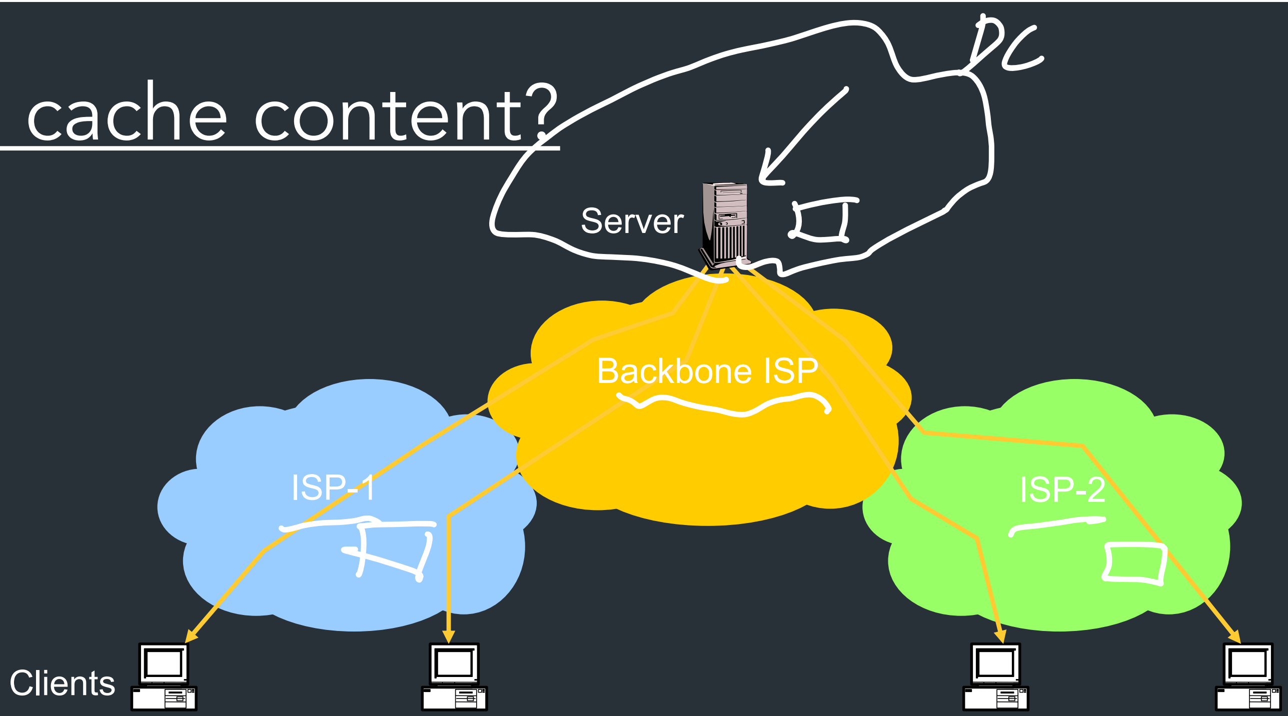

— FETCH AS MUCH AS POSSIBLE FROM CACHES

# How to Control Caching?

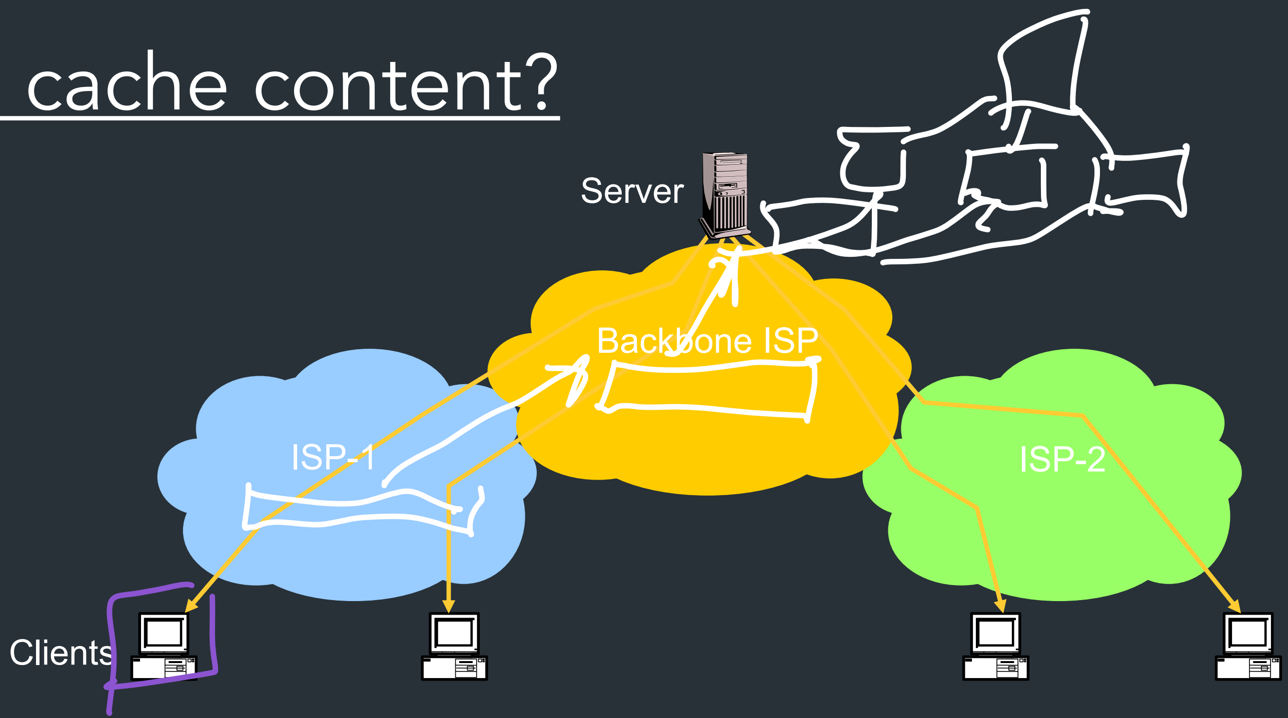
- Server sets options
  - **Expires** header
  - No-Cache header
- Client can do a conditional request:
  - Header option: if-modified-since
  - Server can reply with 304 NOT MODIFIED

USED BY
- BROWSER ITSELF
- IN THE NETWORK

# Where to cache content?



# Where to cache content?




- Client (browser): avoid extra network transfers ← (EG, MULTIPLE PAGES ON SAME SITE)
- Server: reduce load on the server
- Service Provider: reduce external traffic

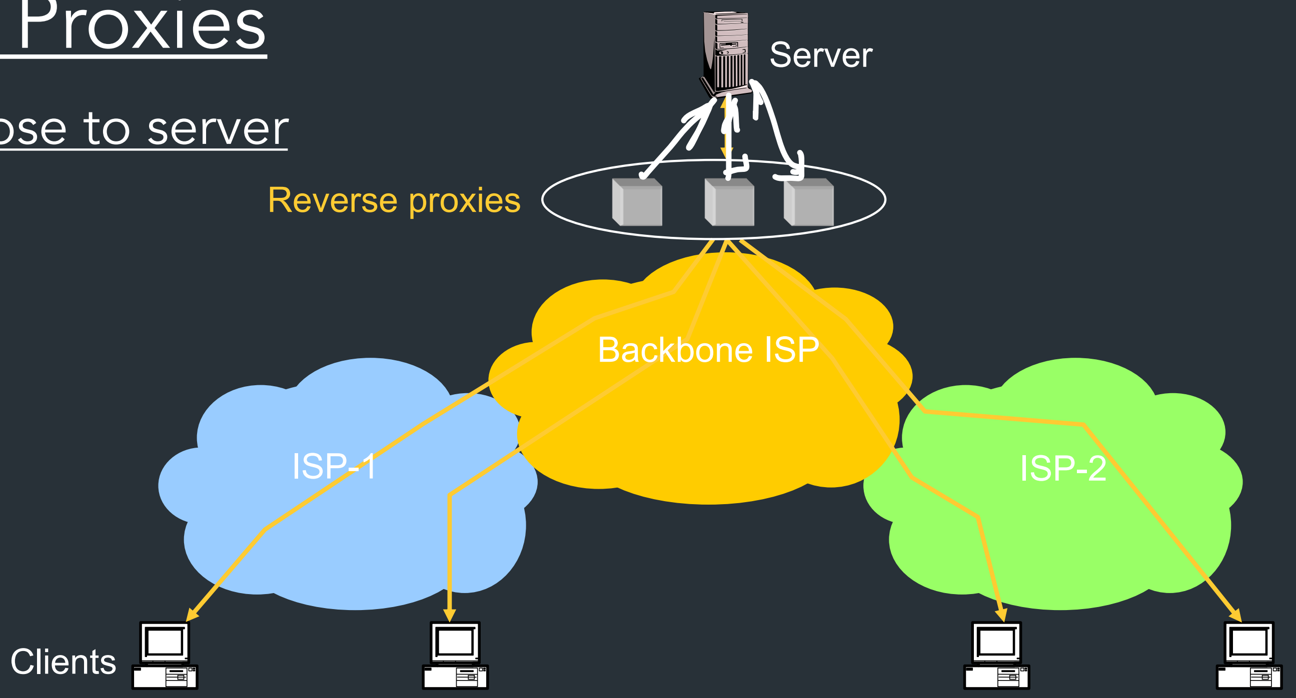
# How well does caching work?

- Very well, up to a point
  - Large overlap in requested objects
  - Objects with one access place upper bound on hit ratio
  - Dynamic objects not cacheable*
- Example: Wikipedia
  - About 400 servers, 100 are HTTP Caches (Squid)
  - 85% Hit ratio for text, 98% for media

* But can cache portions and run special code on edges to reconstruct

# Reverse Proxies

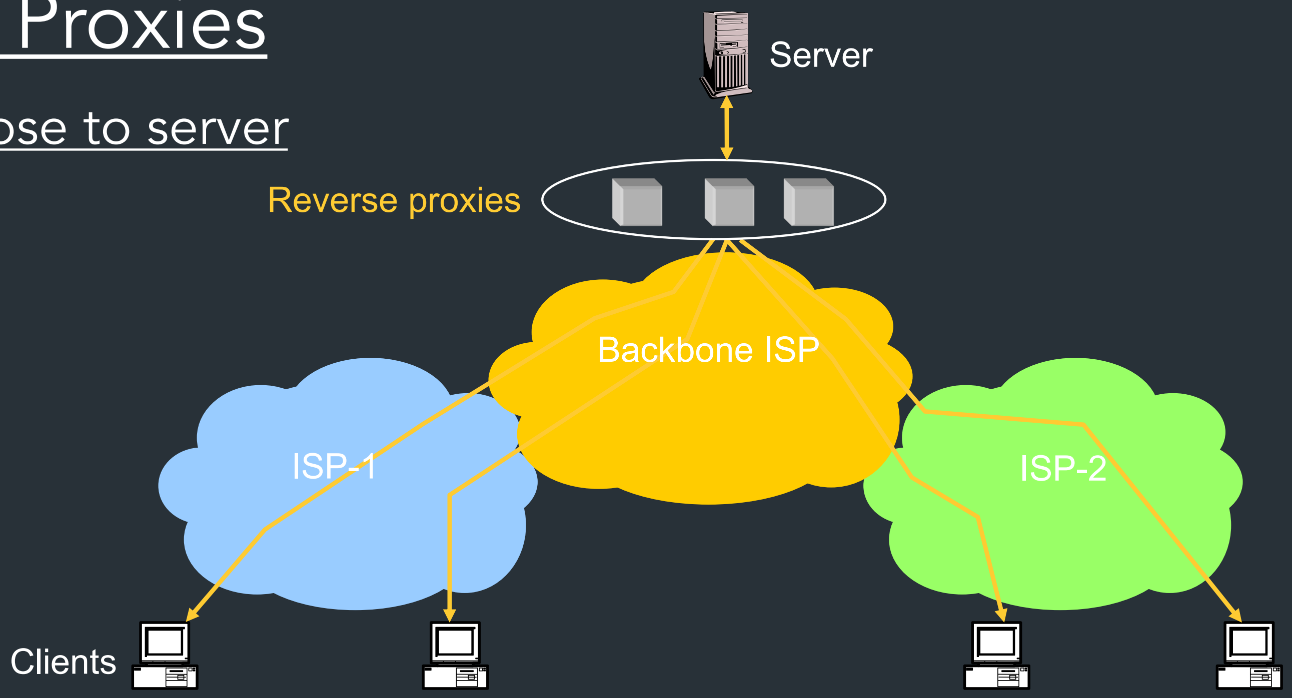
=> Cache close to server




- WITHIN SERVER'S NETWORK CLOSE TO IT
- DISTRIBUTE LOAD, CACHE COMMON RESOURCES,
  => ACCELERATOR.

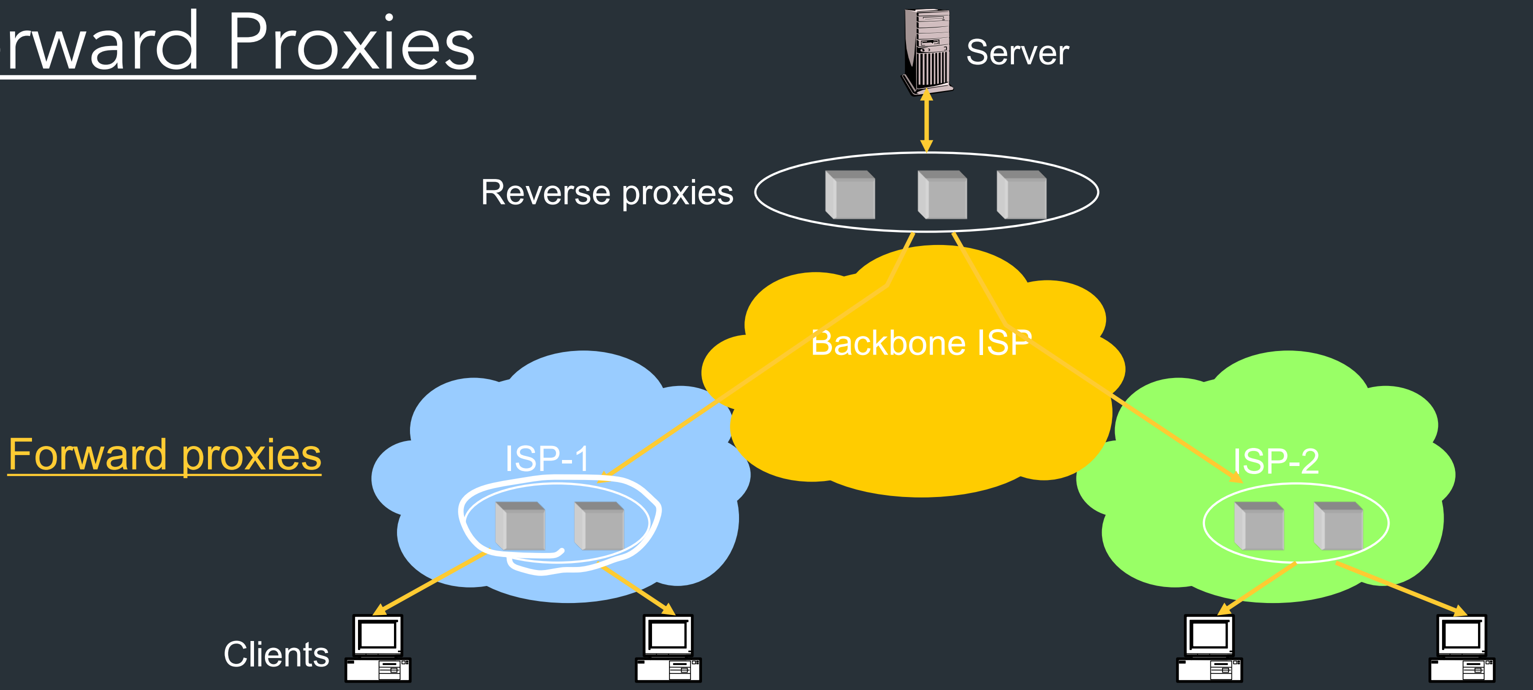
# Reverse Proxies

=> Cache close to server



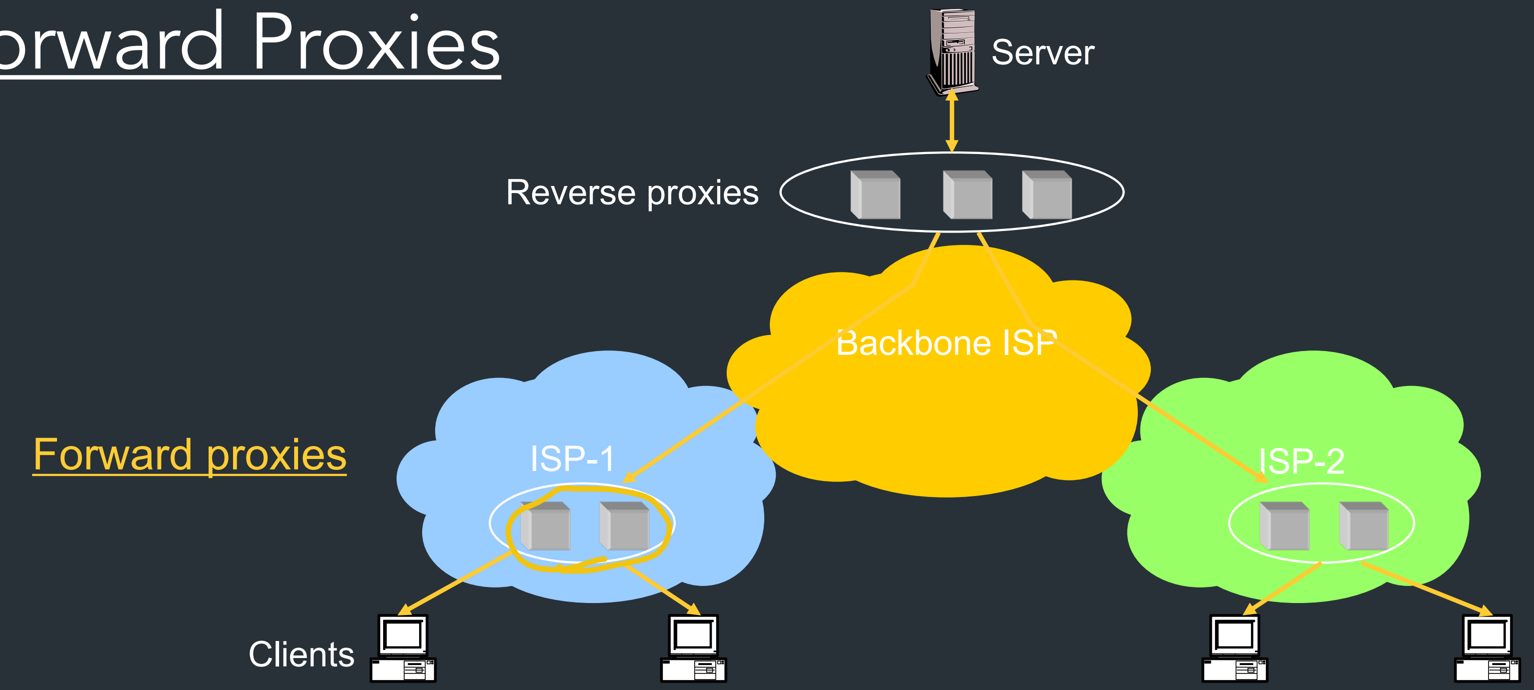

- Also called Accelerators
- Can distribute load within datacenter

# Forward Proxies



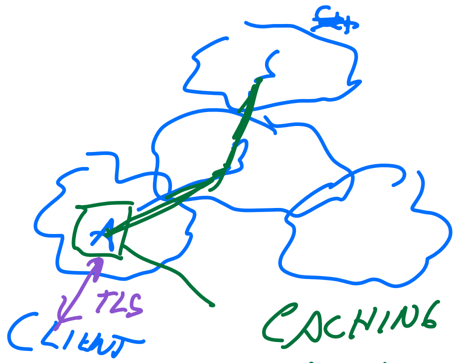

- REDUCE TRAFFIC
- WORKS BEST FOR STATIC CONTENT

# Forward Proxies




Typically done by ISPs or Enterprises

- Reduce network traffic and decrease latency
- May be transparent or configured




## Q: CACHING + HTTPS?

CACHING SERVER NEEDS TO KNOW THE DATA YOU WANT.

⇒ CAUSES PROBLEMS FOR HTTPS, WHICH ENCRYPTS TRAFFIC BETWEEN ENDPOINTS!

⇒ SOLUTION: HTTPS CONNECTION "ENDPOINT" IS USUALLY THE CACHING SERVER ITSELF — NEEDS TO USE OTHER MEANS TO DO SECURE CONNECTION TO BACKEND SERVER

# Content Distribution Networks (CDNs)

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

=> Akamai, Cloudflare, …

=> HAVE CACHES AT MANY POINTS ACROSS NETWORK

- CUSTOMERS PUSH DATA INTO CDN'S CACHES
- CDN REDIRECTS CLIENTS TO THEIR CACHES

# Content Distribution Networks (CDNs)

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

=> Akamai, Cloudflare, …

- Provide both forward and reverse caching

- Can also do some processing

  — TLS/CRYPTO

  — VIDEO TRANSCODING

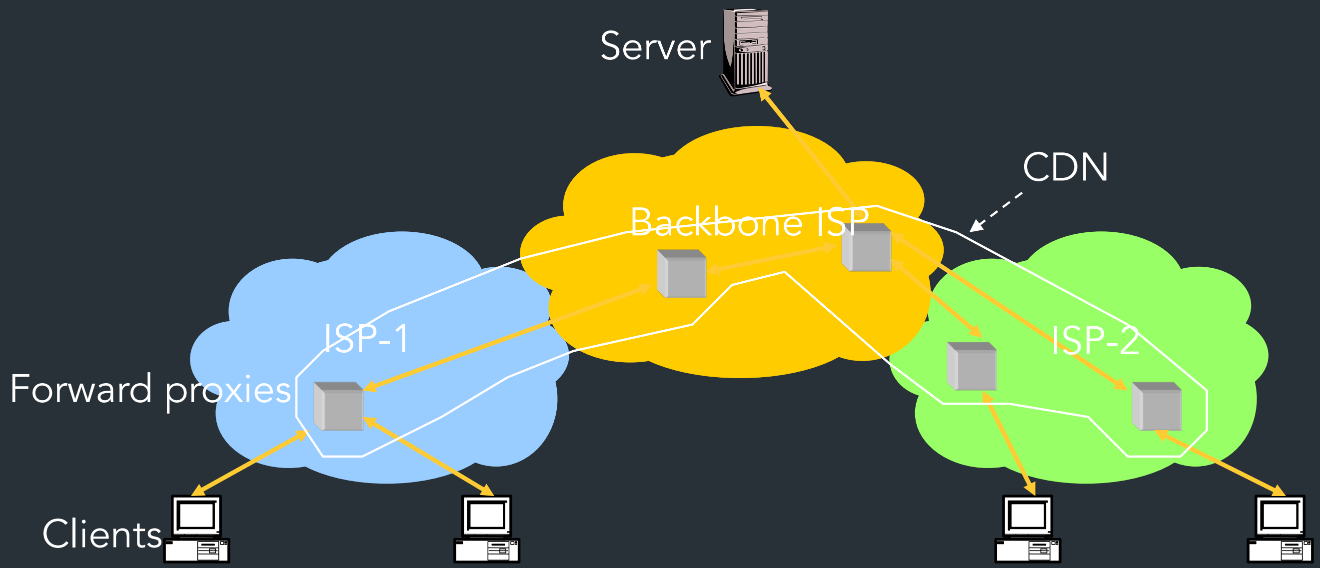
# Content Distribution Networks (CDNs)

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

=> Akamai, Cloudflare, …

- Provide both forward and reverse caching
  - 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

# An Example CDN



How a CDN works
- Eg. Best Buy sets up CDN services with CDN like Akamai
- DNS for www.bestbuy.com controlled by Akamai

When client C resolves bestbuy.com, CDN tries to find best possible cache within CDN for client
=> DNS response points to “best” server within Akamai

How you select the “best” server”
Example:
- Leverage location info for client (GeoIP, AS, …)
- Might look up IP, do active measurements like ping/traceroute, etc.

=> DNS resolver, other caching elements are more intelligent than standard DNS server, etc.

# 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?

# DNS Resolution

```
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
;; AUTHORITY SECTION:
b.akamai.net.             1101     IN       NS       n1b.akamai.net.
b.akamai.net.             1101     IN       NS       n0b.akamai.net.
;; ADDITIONAL SECTION:
n0b.akamai.net.           1267     IN       A        24.143.194.45
n1b.akamai.net.           2196     IN       A        198.7.236.236
```

- n1b.akamai.net finds an edge server close to the client's local resolver
  - Uses knowledge of network: BGP feeds, traceroutes. *Their secret sauce…*

# Example

<u>From Brown</u>

```
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

<u>From Berkeley, CA</u>

```
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
```

```
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  92.77
ms  93.281 ms
```