Internet and Web Architecture

A review

Lecture 2

Srinivas Narayana

http://www.cs.rutgers.edu/~sn624/553-S23

RUTGERS
UNIVERSITY | NEW BRUNSWICK

Software/hardware organization at hosts

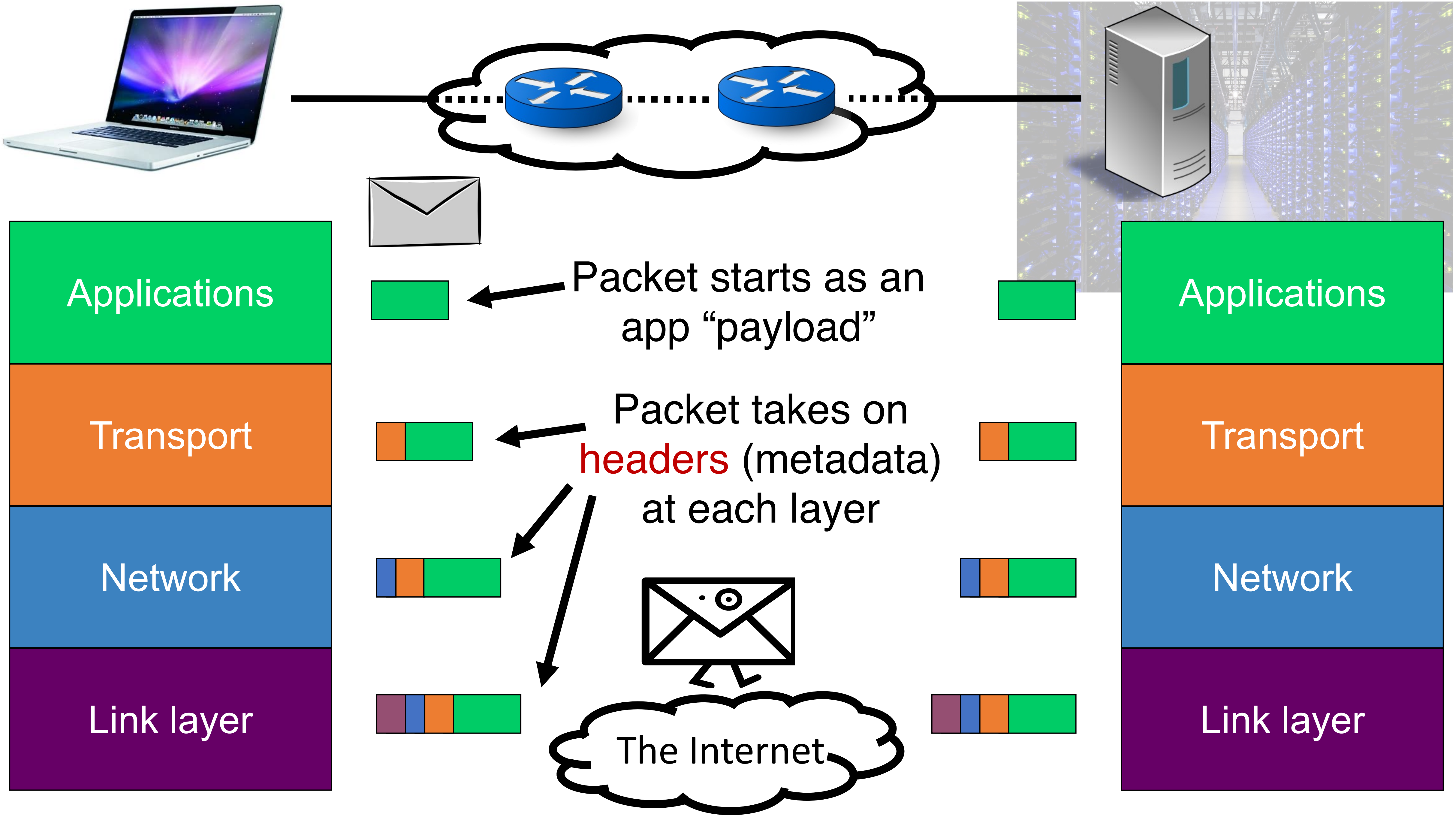
- Application: useful user-level functions
- Transport: provide guarantees to apps
- Network: best-effort global pkt delivery
- Link: best-effort local pkt delivery

Communication functions broken up and “stacked”

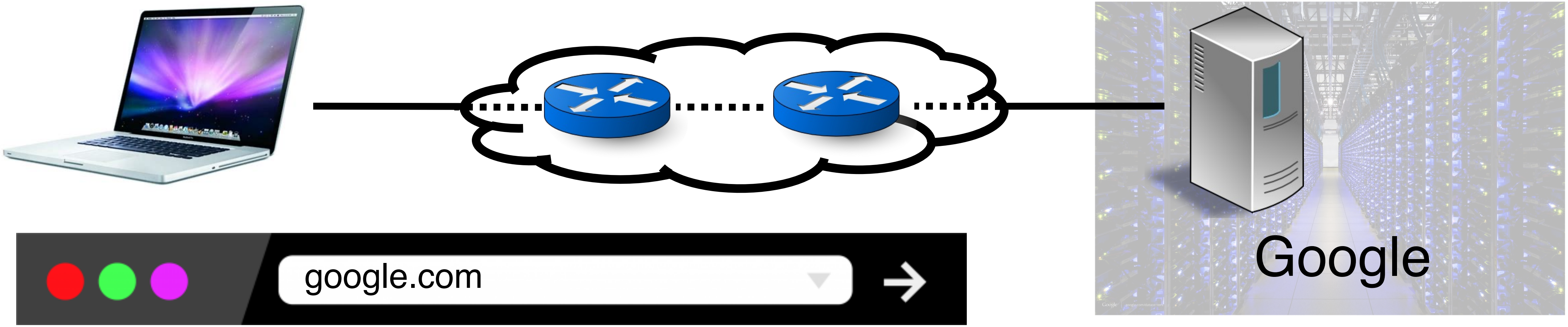
Each layer depends on the one below it.

Each layer supports the one above it.

The interfaces between layers are well-defined and standardized.


Applications
Transport
Network
Link layer
Packet starts as an app "payload"
Packet takes on headers (metadata) at each layer
The Internet
Applications
Transport
Network
Link layer

Name Resolution



Machines communicate using IP addresses and ports
IP addresses: ~12 digits (IPv4) or more
Ports: fixed based on application (e.g., 80: web)

Need a way to turn human-readable addresses into Internet addresses.

Ask someone	Ask everyone	Tell everyone
Directory service	Query broadcast	Information flooding

Asking “someone” could involve asking many machines…

Domain Name Service

DOMAIN NAME	IP ADDRESS
spotify.com	98.138.253.109
cs.rutgers.edu	128.6.4.2
www.google.com	74.125.225.243
www.princeton.edu	128.112.132.86

<Client IP, CPort, DNS server IP, 53>

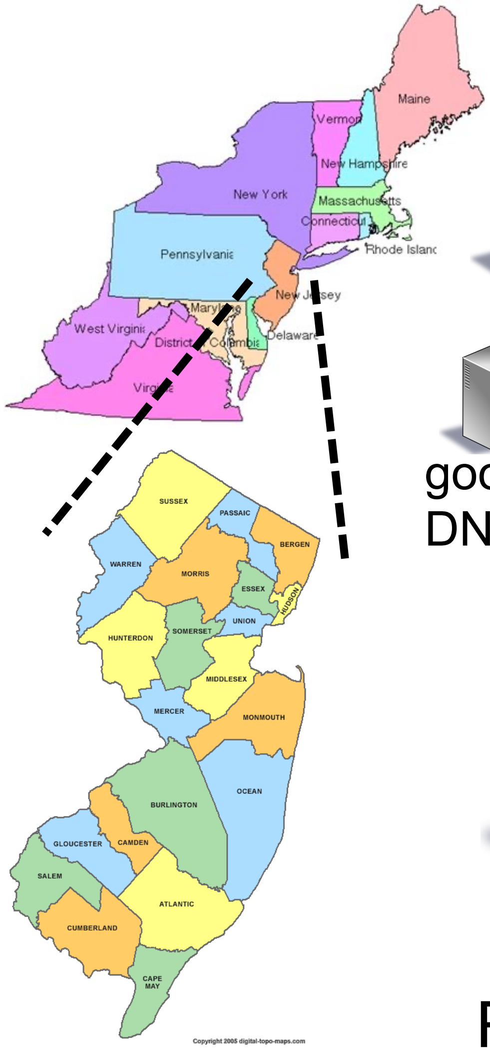
QUERY cs.rutgers.edu

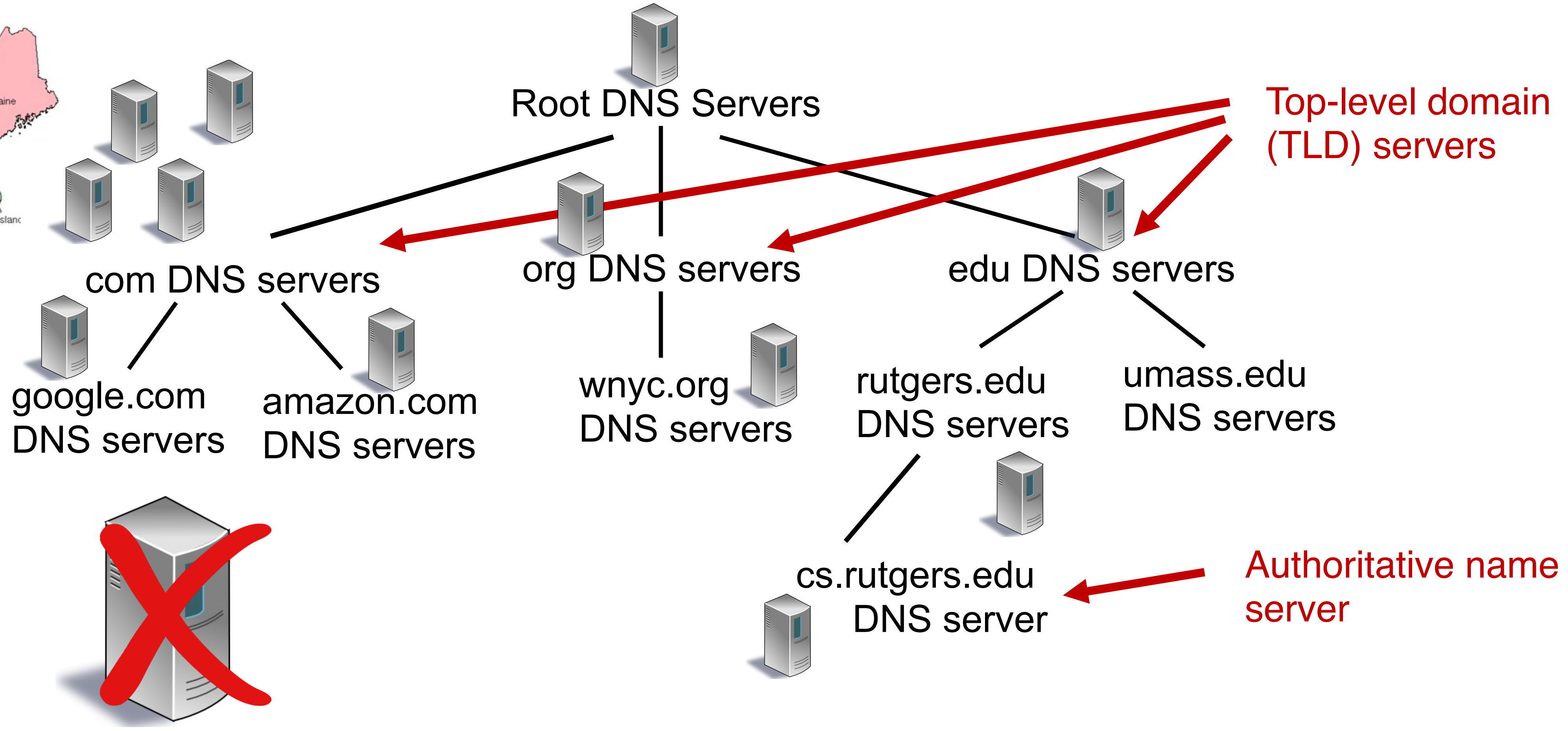
<DNS server, 53, Client IP, Cport>

RESPONSE 128.6.4.2

- Key idea: Implement a server that looks up a table.
- Will this scale?
  - Every new (changed) host needs to be (re)entered in this table
  - Performance: can the server serve billions of Internet users?
  - Failure: what if the server or the database crashes?
  - Security: What if someone “takes over” this server?

Distributed and hierarchical database





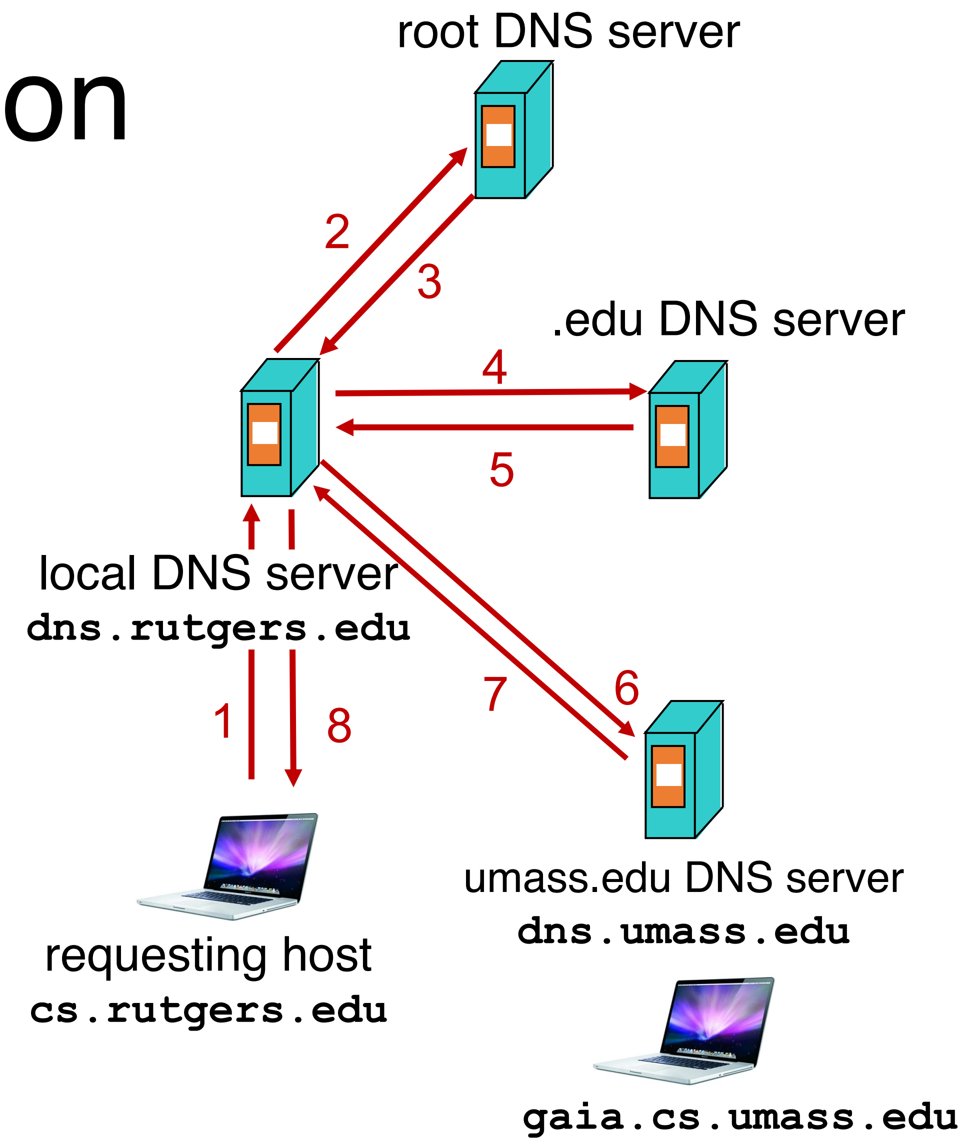
RFC 1034

Hierarchy Replication Indirection

DNS name resolution

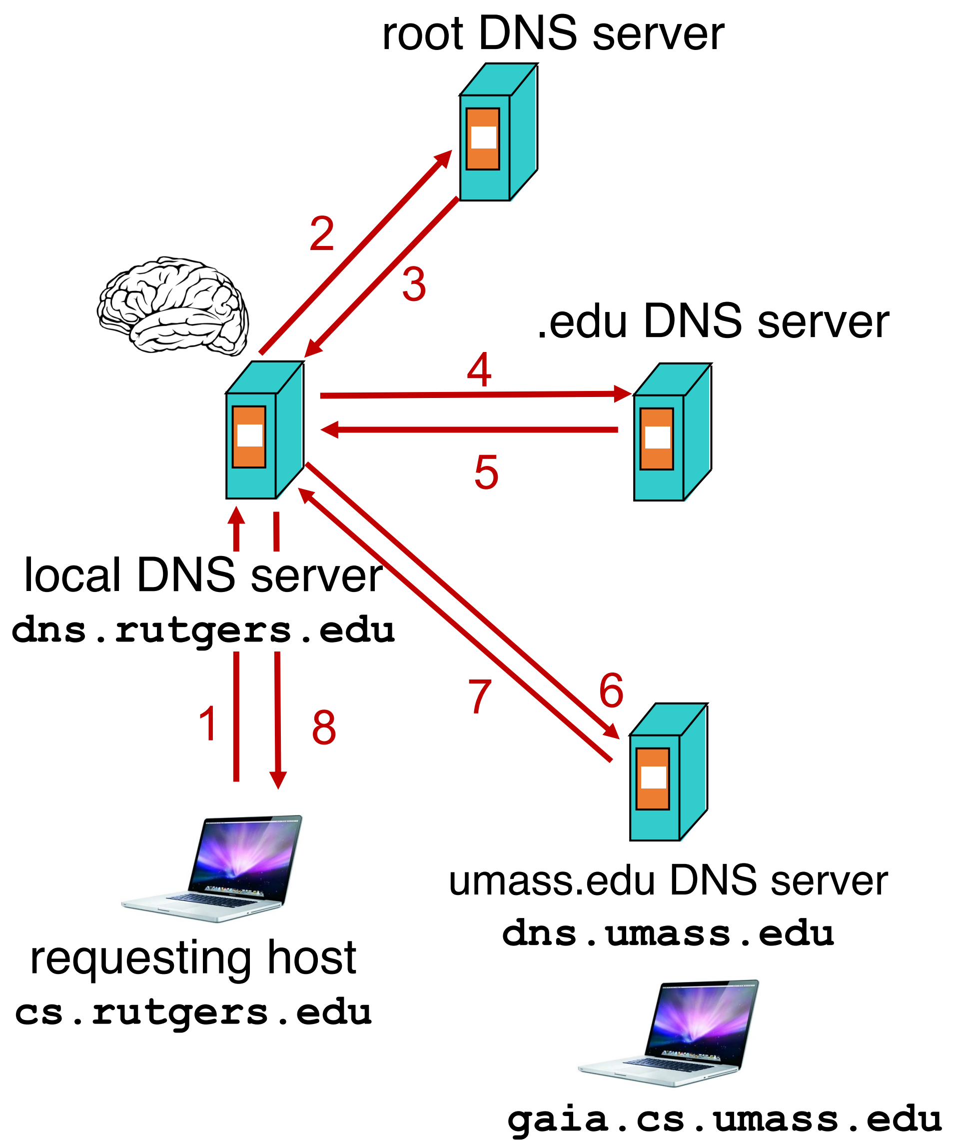
- Host at cs.rutgers.edu wants IP address for gaia.cs.umass.edu

- Local DNS server
- Root DNS server
- TLD DNS server
- Authoritative DNS server



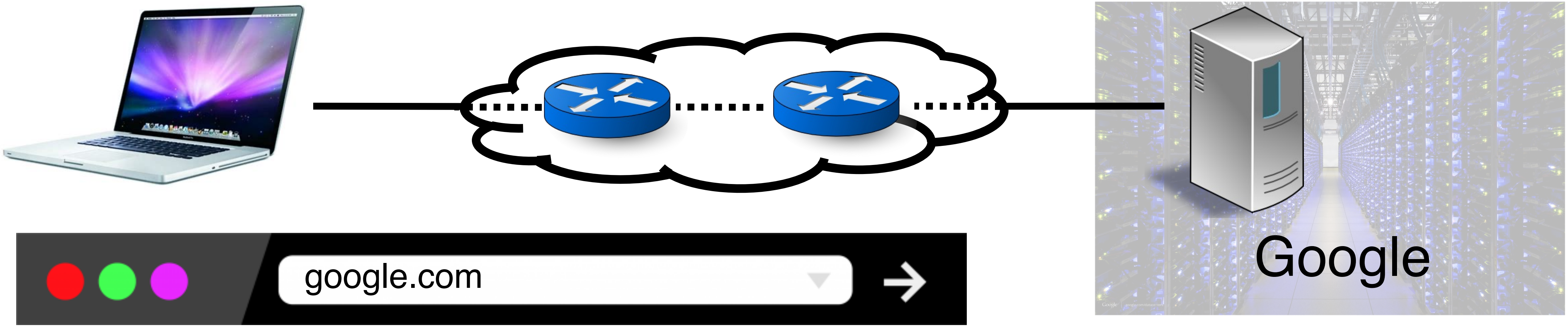
DNS caching

- Once (any) name server learns a name to IP address mapping, it *caches* the mapping
- Cache entries timeout (disappear) after some time
- TLD servers typically cached in local name servers
- In practice, root name servers aren't visited often!
- Caching is pervasive in DNS



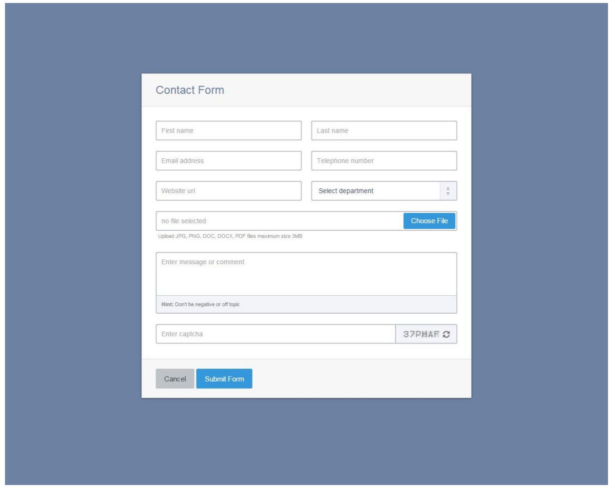
Example DNS interactions

- `dig <domain-name>`
- `dig +trace <domain-name>`
- `dig @<dns-server> <domain-name>`



The web is a *specific* application protocol running over a network: HyperText Transfer Protocol (HTTP)

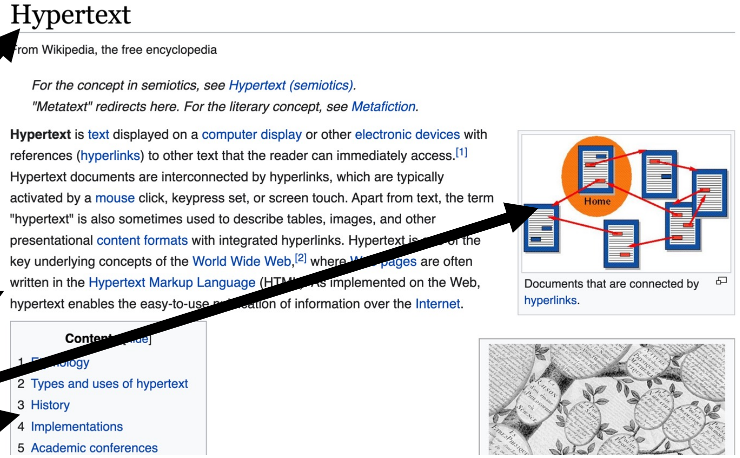
Each object addressable by a name (URL)



Named objects can be static (image, video)

… or the result of a dynamic app process

Objects



Hypertext

From Wikipedia, the free encyclopedia

For the concept in semiotics, see Hypertext (semiotics).
"Metatext" redirects here. For the literary concept, see Metafiction.

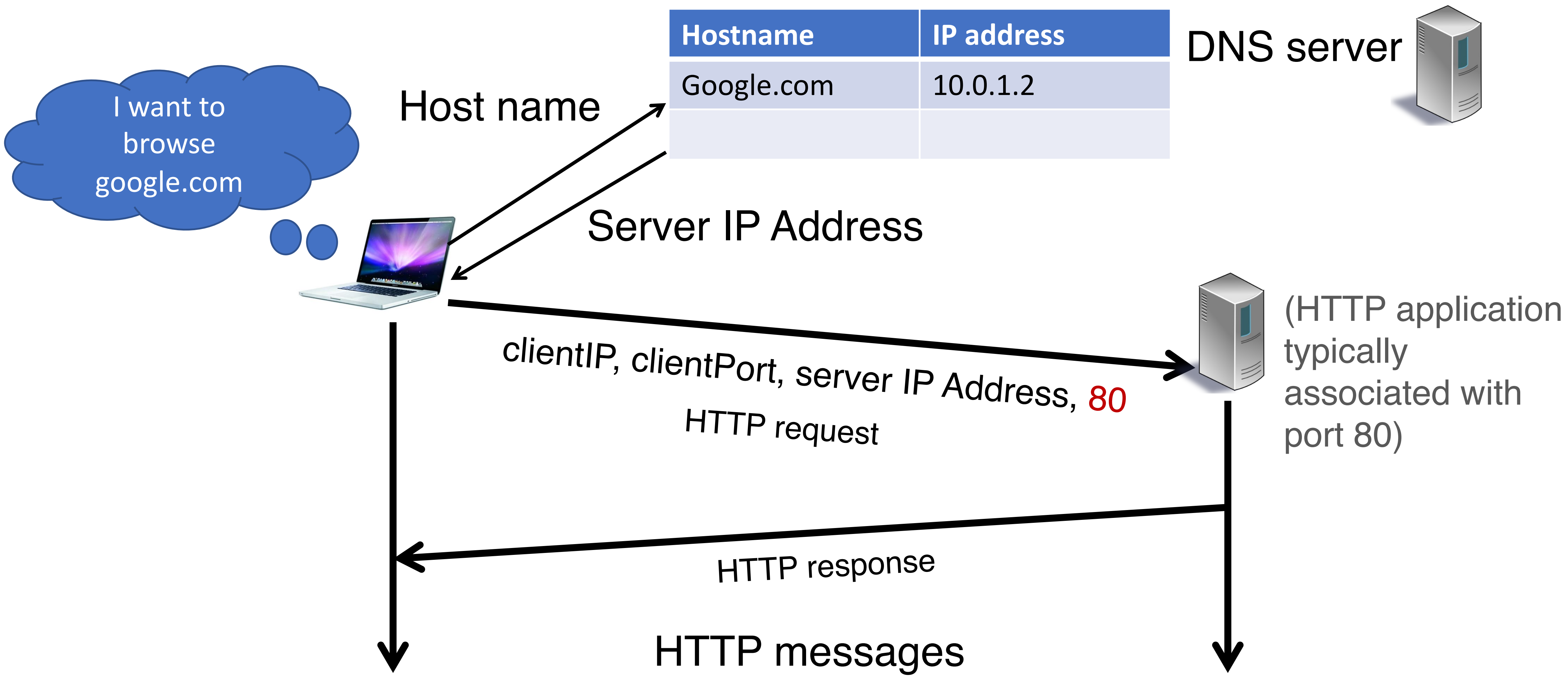
Hypertext is text displayed on a computer display or other electronic devices with references (hyperlinks) to other text that the reader can immediately access.[1] Hypertext documents are interconnected by hyperlinks, which are typically activated by a mouse click, keypress set, or screen touch. Apart from text, the term "hypertext" is also sometimes used to describe tables, images, and other presentational content formats with integrated hyperlinks. Hypertext is [illegible] the key underlying concepts of the World Wide Web,[2] where [illegible] pages are often written in the Hypertext Markup Language (HTML) [illegible] implemented on the Web, hypertext enables the easy-to-use [illegible] of information over the Internet.

Documents that are connected by hyperlinks.

Contents [hide]

Web interactions

Hostname	IP address
Google.com	10.0.1.2



Example HTTP interactions

- `wget google.com (or) curl google.com`

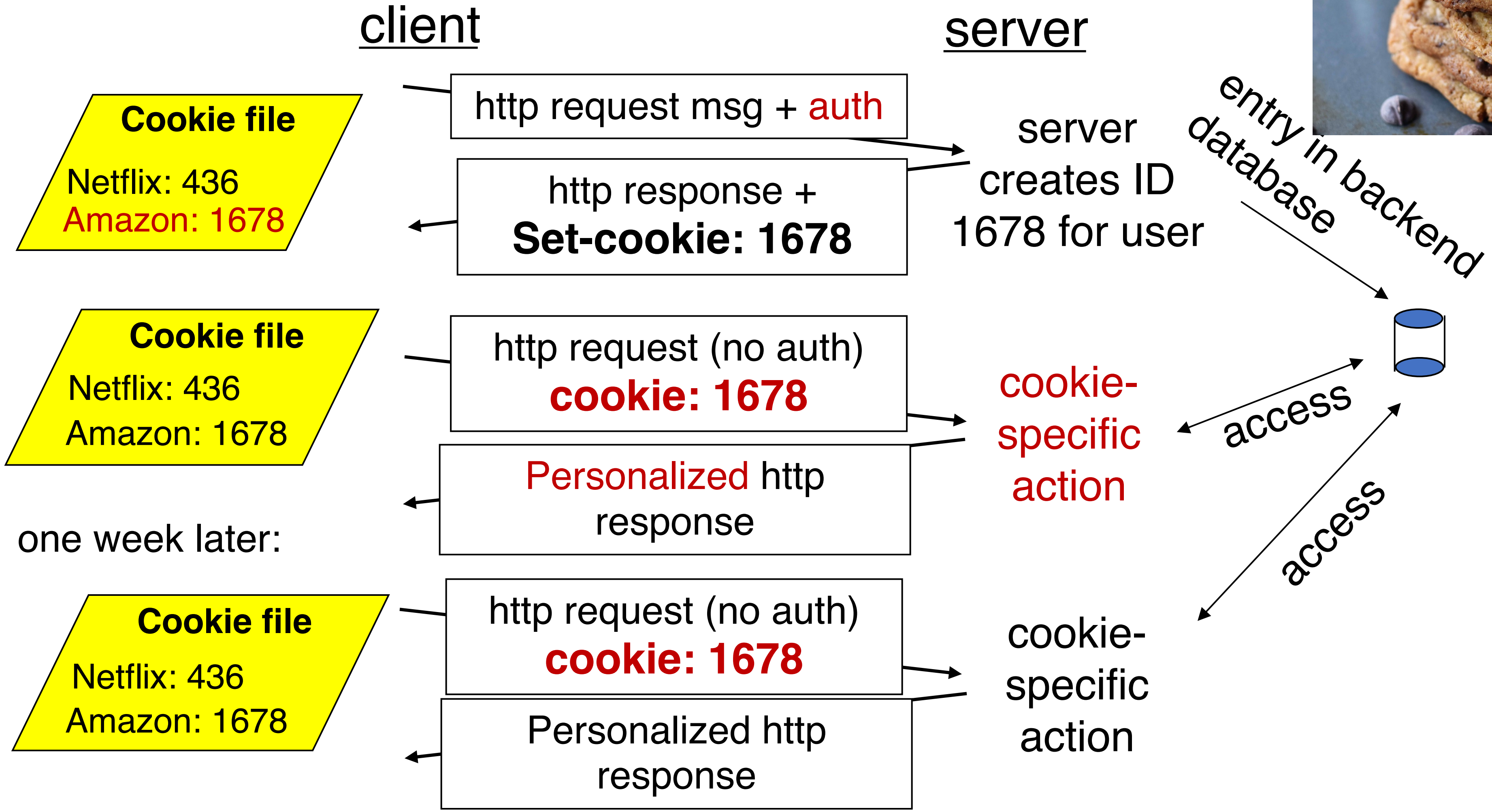
- `telnet example.com 80`
  - `GET / HTTP/1.1`
  - `Host: example.com`

(followed by two enter’s)

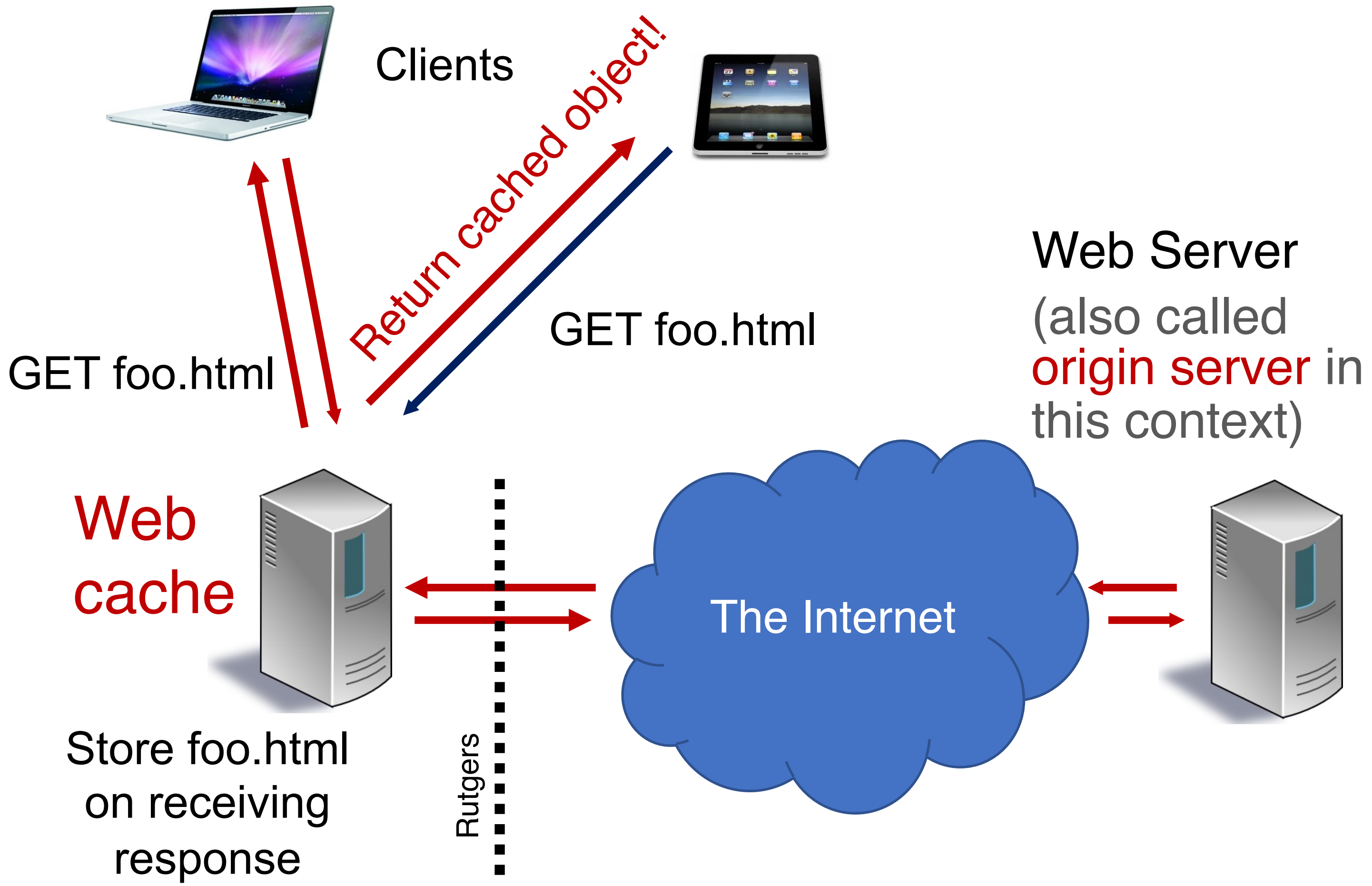
- Exercise: try
  - `telnet google.com 80`
  - `telnet web.mit.edu 80`

Remembering users: cookies

Cookie is typically opaque to client.



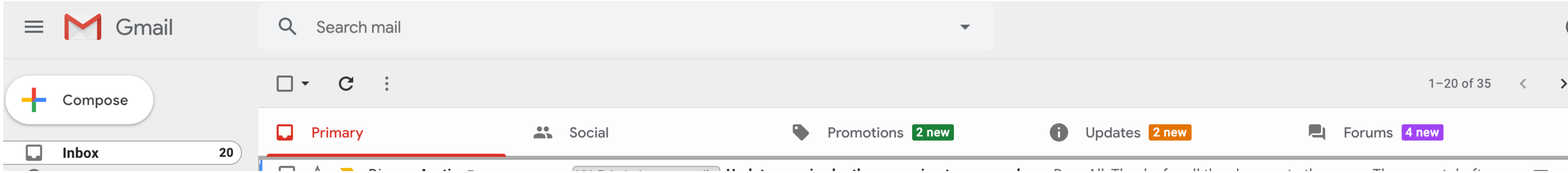
Improving performance: Web caching



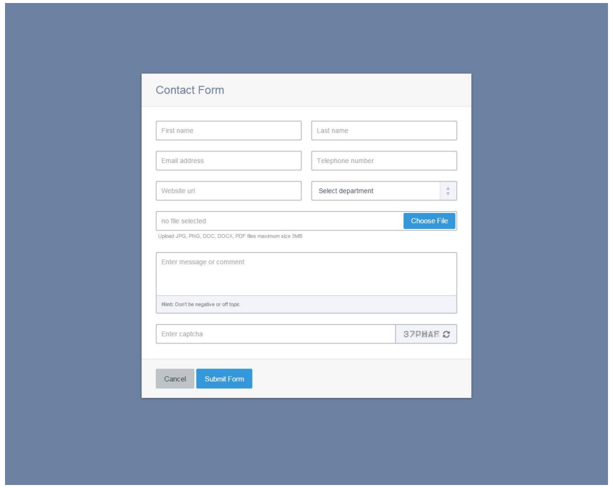
- Network administrators (e.g., Rutgers) may run web caches to remember popular web objects
- Hit: cache returns object
- Miss: obtain object from originating web server (origin server) and return to client
  - Also cache the object locally
- Reduce response time
- Reduce traffic requirements (and $$) on an organization's network connections

Not all content is effectively cacheable

- Personalized content

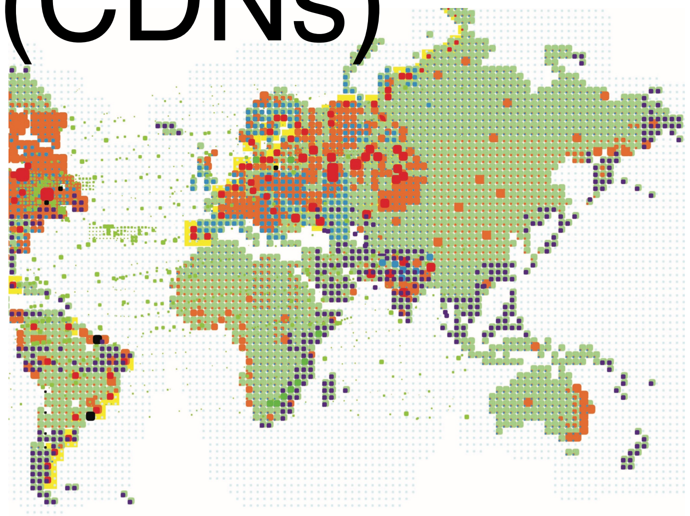


- Interactive processing
  - e.g., forms, shopping carts, ajax, etc.



- Long tail of (obscure) content

Content Distribution Networks (CDNs)



A global network of web caches

- Provisioned by ISPs and network operators
- Or content providers, like Netflix, Google, etc.

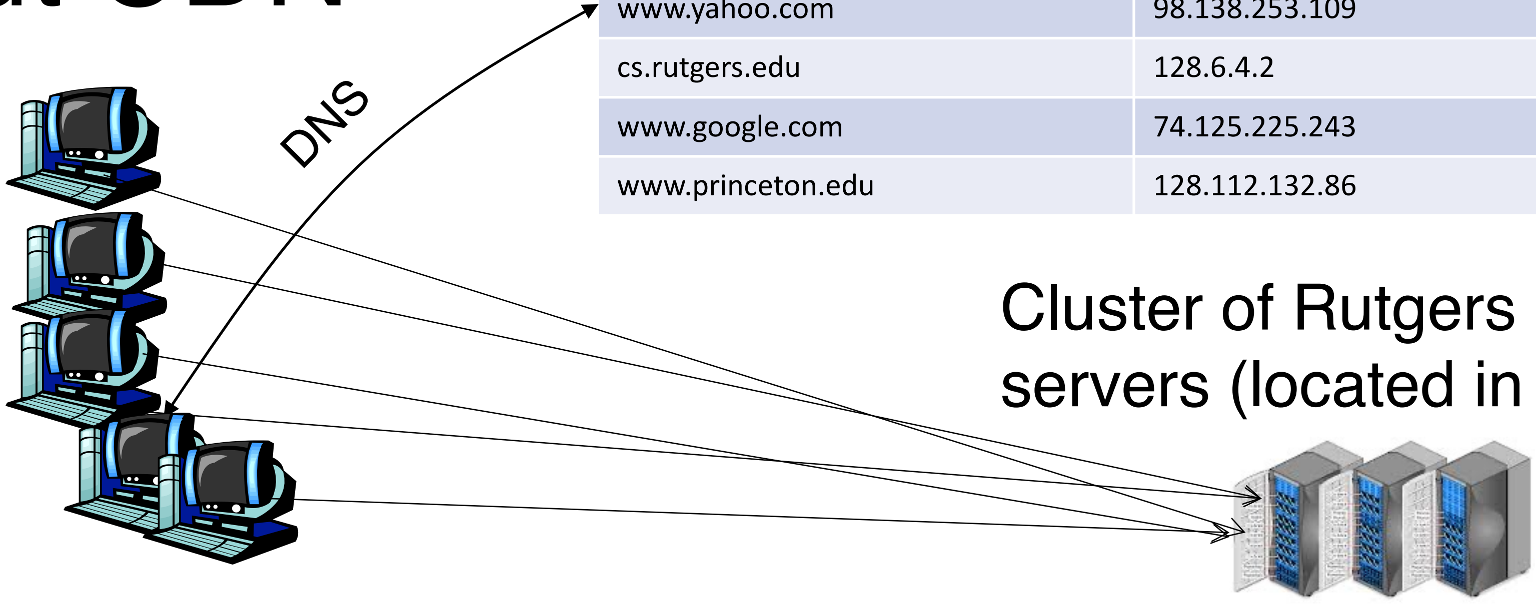
Uses

- Reduce traffic on a network’s Internet connection, e.g., Rutgers
- Improve response time for users: CDN nodes are closer to users than origin servers (servers holding original content)
- Reduce bandwidth requirements on content provider
- Reduce $$ to maintain origin servers

Without CDN

DOMAIN NAME	IP ADDRESS
www.yahoo.com	98.138.253.109
cs.rutgers.edu	128.6.4.2
www.google.com	74.125.225.243
www.princeton.edu	128.112.132.86

Clients distributed all over the world



Cluster of Rutgers CS origin servers (located in NJ, USA)

128.6.4.2

- Problems:
- Huge bandwidth requirements for Rutgers
- Large propagation delays to reach users

Where the CDN comes in

- Distribute content of the origin server over geographically distributed CDN servers

- But how will users get to these CDN servers?

- Use DNS!
  - DNS provides an additional layer of indirection
  - Instead of returning IP address, return another DNS server (NS record)
  - The second DNS server (run by the CDN) returns IP address to client

- The CDN runs its own DNS servers (CDN name servers)
  - Custom logic to send users to the "closest" CDN web server

With CDN

DOMAIN NAME	IP ADDRESS
www.yahoo.com	98.138.253.109
cs.rutgers.edu	124.8.9.8 (NS record pointing to CDN name server)
www.google.com	74.125.225.243

DNS reply

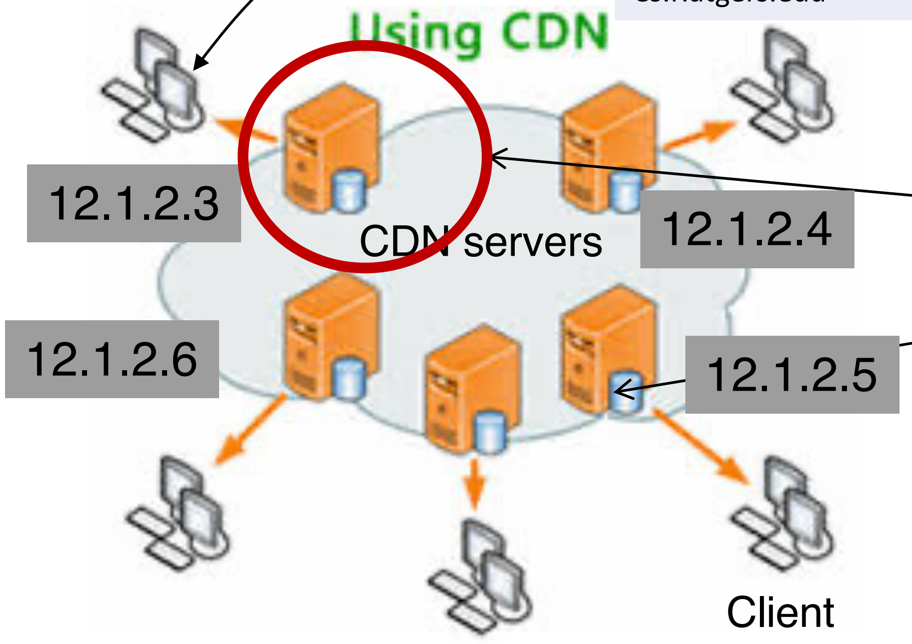
NS record delegates the choice of IP address to the CDN name server.

CDN Name Server (124.8.9.8)

DOMAIN NAME	IP ADDRESS
Cs.Rutgers.edu	12.1.2.3
Cs.Rutgers.edu	12.1.2.4
Cs.Rutgers.edu	12.1.2.5
Cs.Rutgers.edu	12.1.2.6

Custom logic to map ONE domain name to one of many IP addresses!

Popular CDNs:
CloudFlare
Akamai
Level3
…



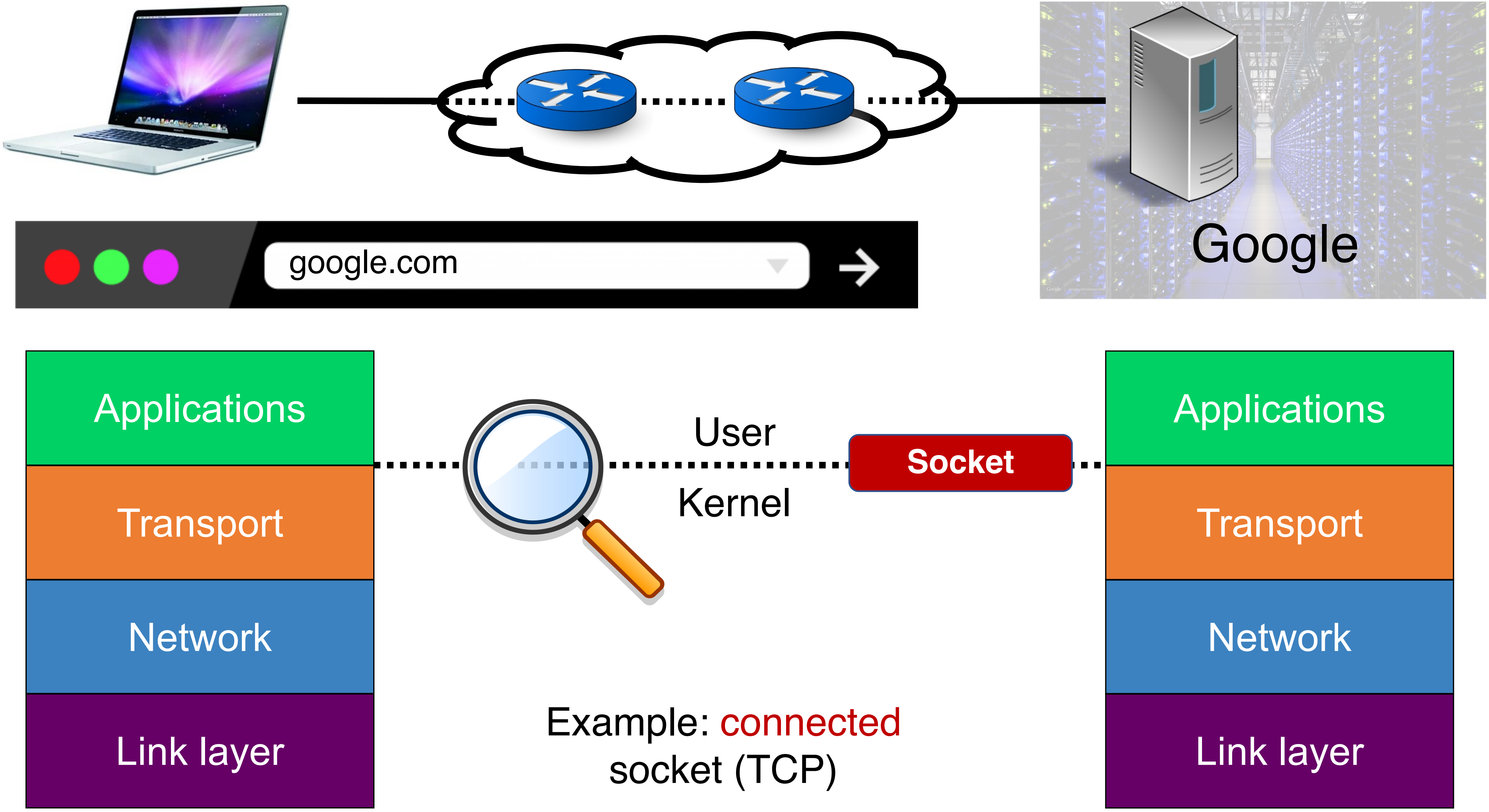
128.6.4.2
Origin server

Most requests go to CDN servers (caches).
CDN servers may request object from origin
Few client requests go directly to origin server

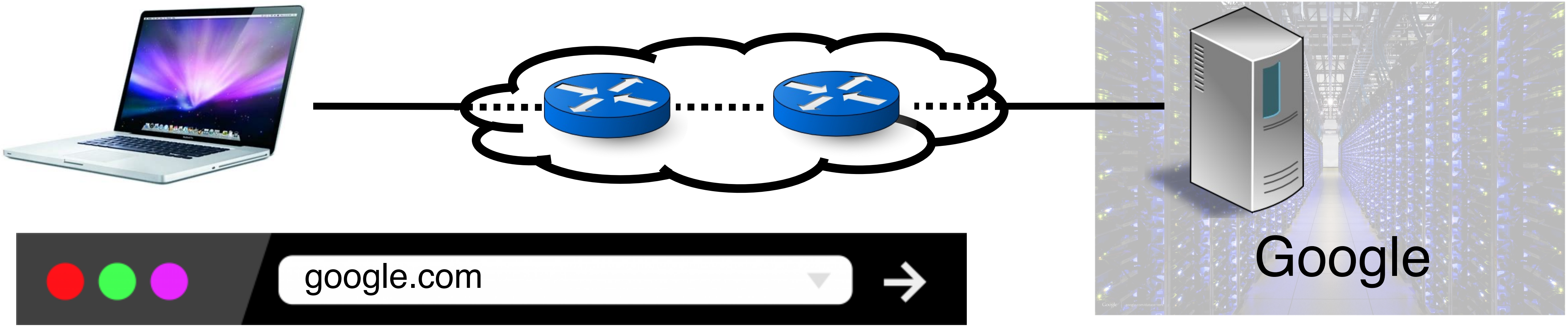
Seeing a CDN in action

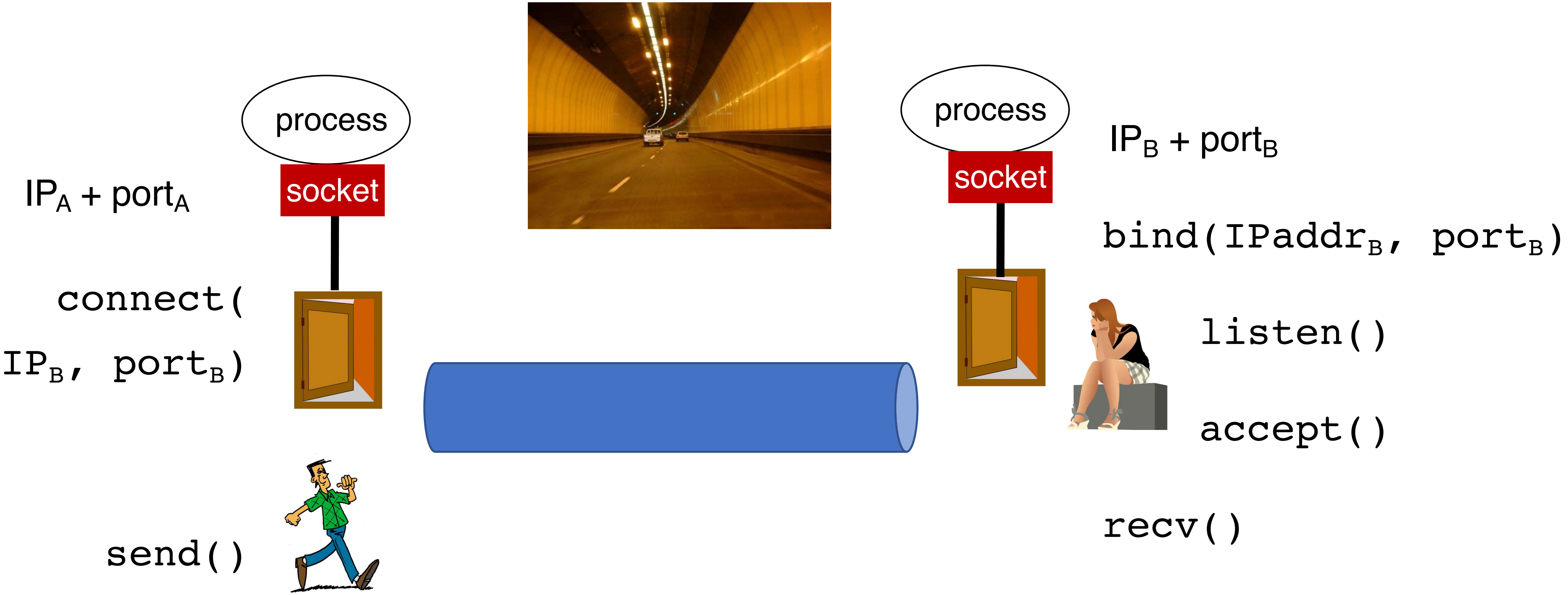
- `dig web.mit.edu` (or) `dig +trace web.mit.edu`
- `telnet web.mit.edu 80`

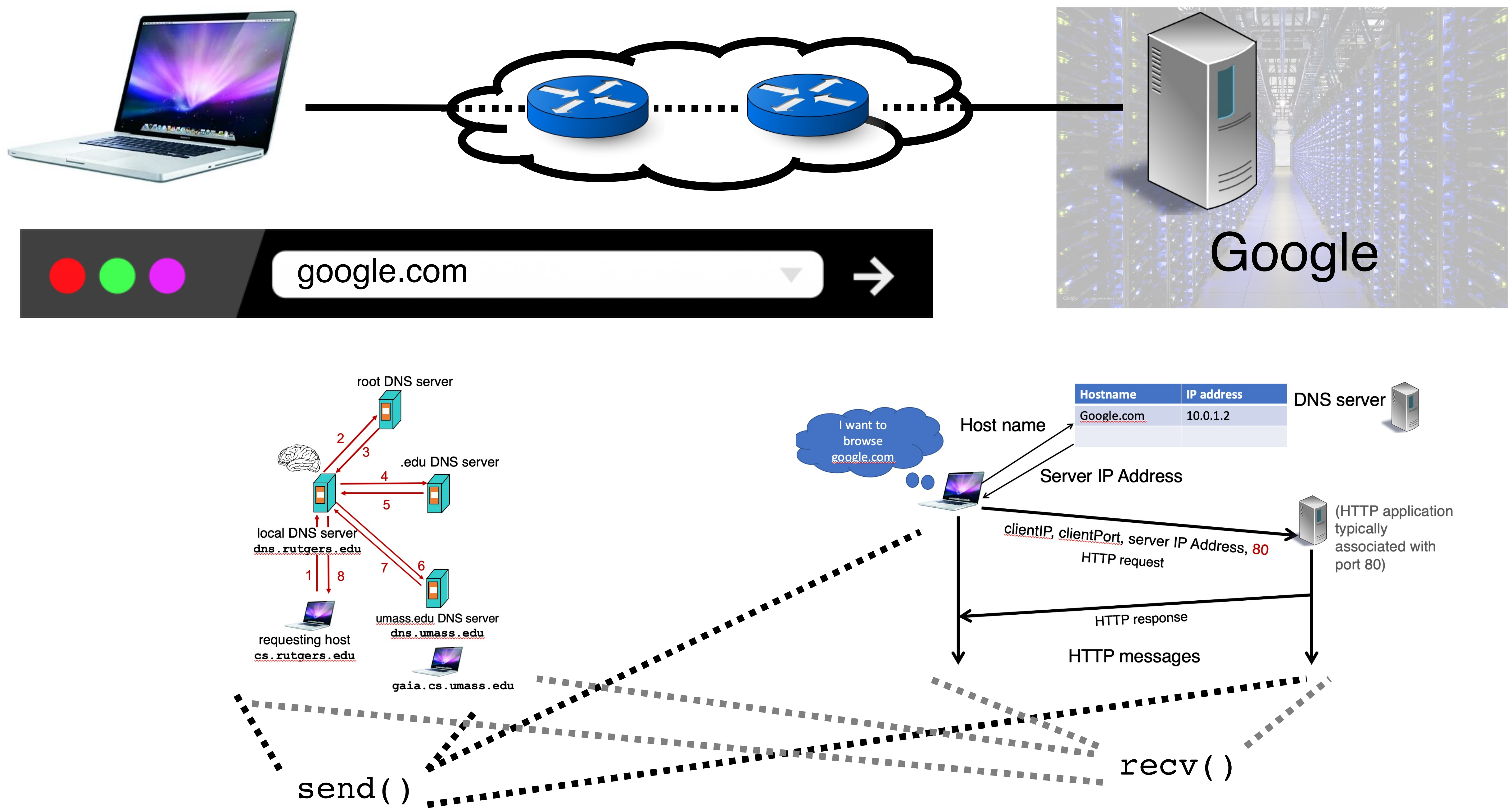
Application-OS interface



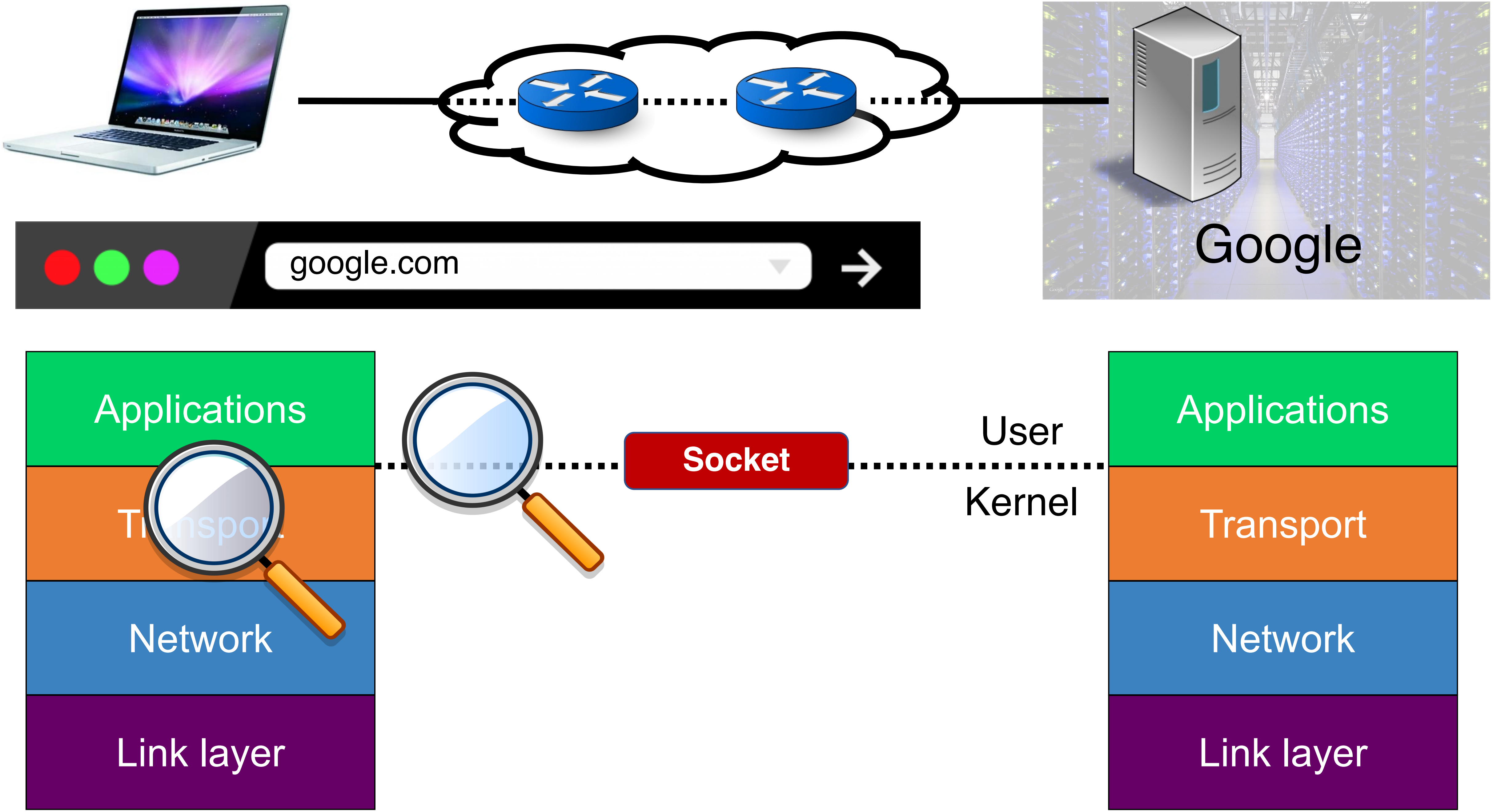
Example: connected socket (TCP)


google.com
Google


process
socket
IP_A + $port_A$
connect(
IP_B, $port_B$)
send()
process
socket
IP_B + $port_B$
bind($IPaddr_B$, $port_B$)
listen()
accept()
recv()

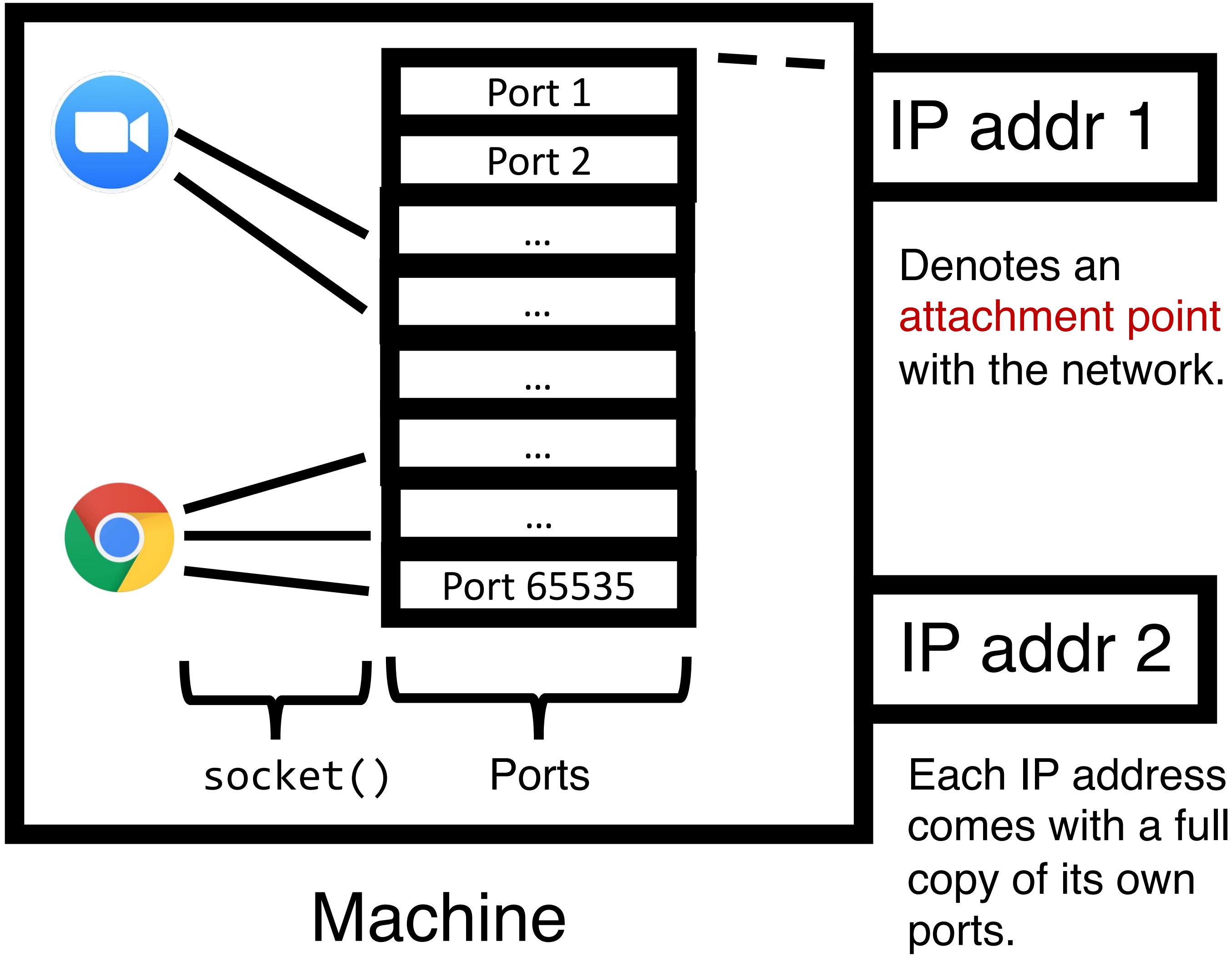


google.com
Google
root DNS server
.edu DNS server
local DNS server
dns.rutgers.edu
umass.edu DNS server
dns.umass.edu
requesting host
cs.rutgers.edu
gaia.cs.umass.edu
1
2
3
4
5
6
7
8
I want to browse google.com
Host name
Server IP Address
Hostname
IP address
Google.com
10.0.1.2
DNS server
clientIP, clientPort, server IP Address, 80
HTTP request
(HTTP application typically associated with port 80)
HTTP response
HTTP messages
send()
recv()


google.com
Google
Applications
Transport
Network
Link layer
Socket
User
Kernel
Applications
Transport
Network
Link layer

Transport

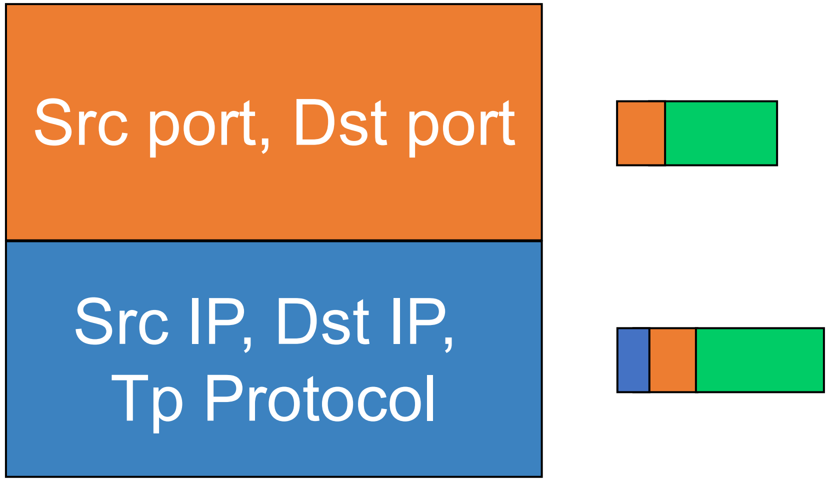
(1) (De)multiplexing



Denotes an attachment point with the network.

Each IP address comes with a full copy of its own ports.

Connection lookup: The operating system does a lookup using these data to determine the right socket and app.



UDP or TCP listening:
(dst IP, dst port, TCP)

TCP established:
(dst IP, dst port, src IP, src port, TCP)

TCP sockets of different types

Listening (bound but unconnected)

```
# On server side
ls = socket(AF_INET, SOCK_STREAM)
ls.bind(serv_ip, serv_port)
ls.listen() # no accept() yet
```

(dst IP, dst port)

➔

Socket `(ss)`

Enables new connections to be demultiplexed correctly

Connected (Established)

```
# On server side
cs, addr = ls.accept()

# On client side
connect(serv_ip, serv_port)
```

`accept()` creates a new socket with the 4-tuple (established) mapping

(src IP, dst IP, src port, dst port)

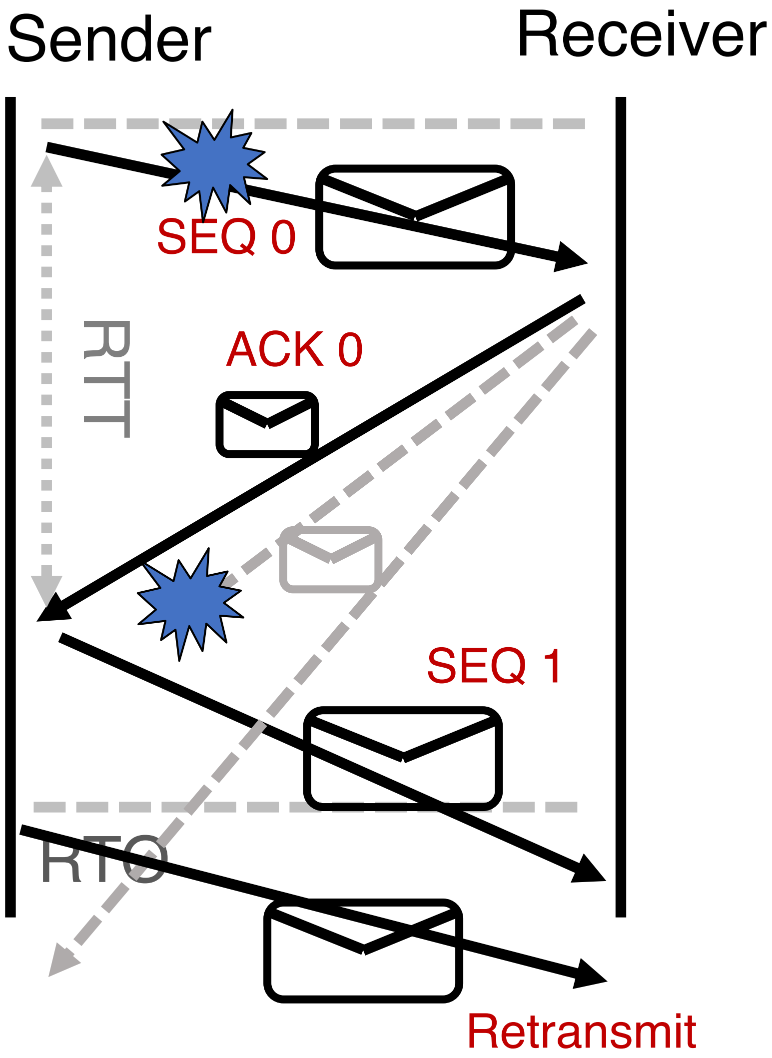
➔

Socket `(cs NOT ls)`

Enables established connections to be demultiplexed correctly

(2) Reliability: Stop and Wait. 3 Ideas

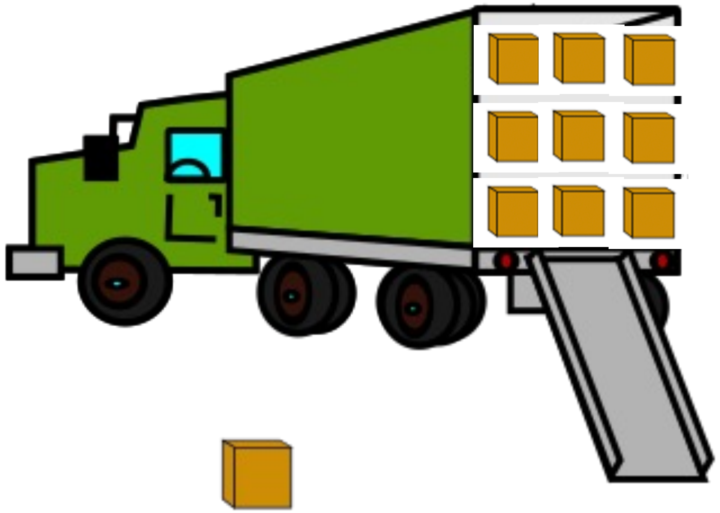
- ACKs: Sender sends a single packet, then waits for an ACK to know the packet was successfully received. Then the sender transmits the next packet.
- RTO: If ACK is not received until a timeout, sender retransmits the packet
- Seq: Disambiguate duplicate vs. fresh packets using sequence numbers that change on "adjacent" packets



Sending one packet per RTT makes the data transfer rate limited by the time between the endpoints, rather than the bandwidth.

Ensure you got the (one) box safely; make N trips

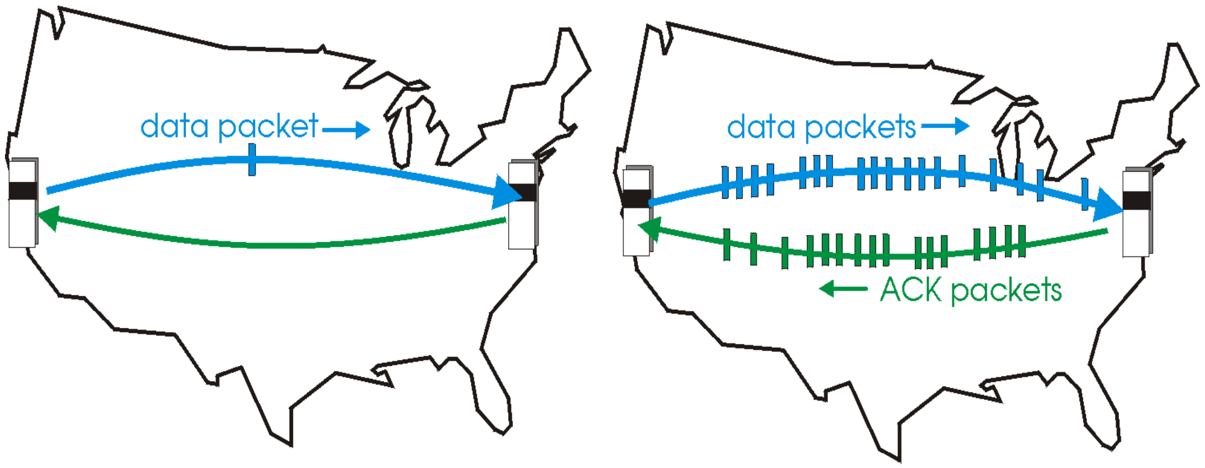
Ensure you get N boxes safely; make just 1 trip!



Keep many packets in flight

Pipelined reliability

- Data in flight: data that has been sent, but sender hasn't yet received ACKs from the receiver
  - Note: can refer to packets in flight or bytes in flight
- New packets sent at the same time as older ones still in flight
- New packets sent at the same time as ACKs are returning
- More data moving in same time!
- Improves throughput
  - Rate of data transfer



(a) a stop-and-wait protocol in operation (b) a pipelined protocol in operation