Computer Networks
@CS.NCTU

Lecture 2: Applications

Instructor: Kate Ching-Ju Lin (林靖茹)

Slides modified from
"Computer Networking: A Top-Down Approach" 7th Edition

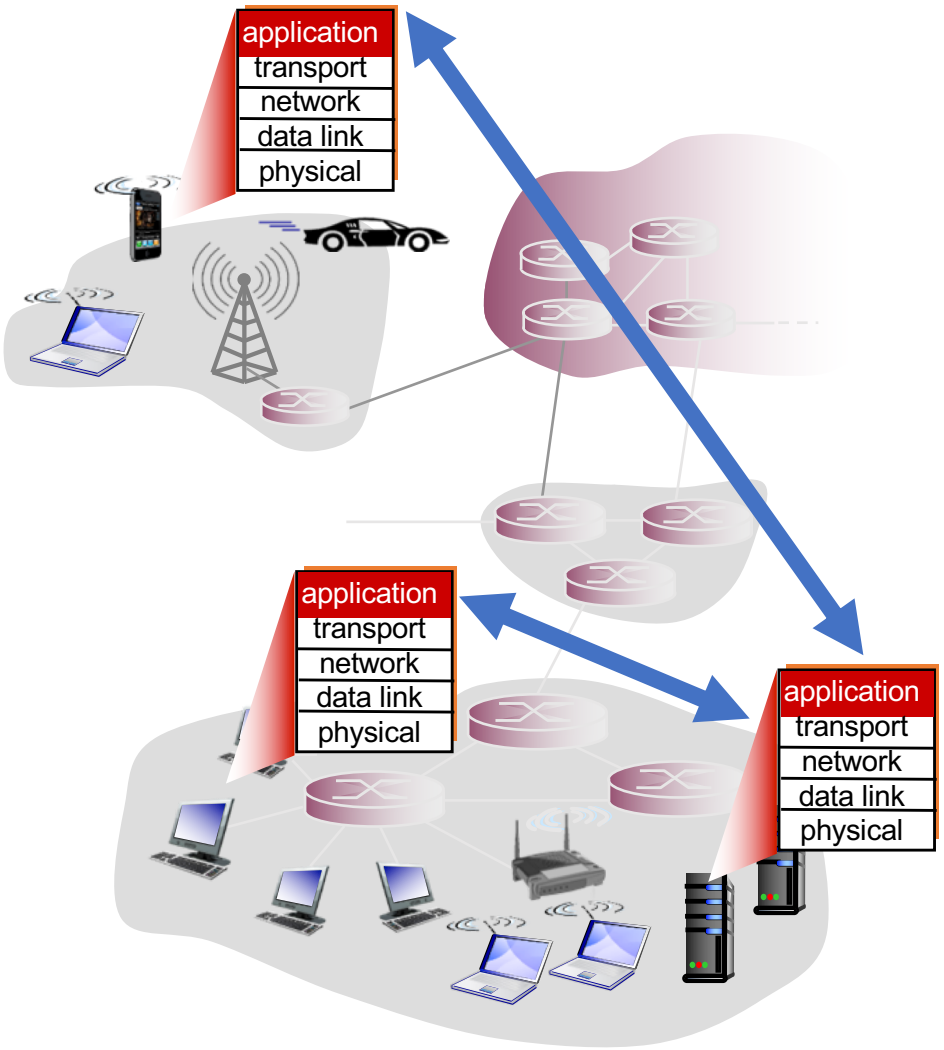

http://www
YouTube
NETFLIX
LINE
Spotify
POKÉMON
GO
WAR OF
WORDCRAFT

Outline

- Principles of network applications
- Web and HTTP
- E-Mail and SMTP
- DNS
- Peer-to-peer applications
- Video streaming

Outline

- **Principles of network applications**
- Web and HTTP
- E-Mail and SMTP
- DNS
- Peer-to-peer applications
- Video streaming



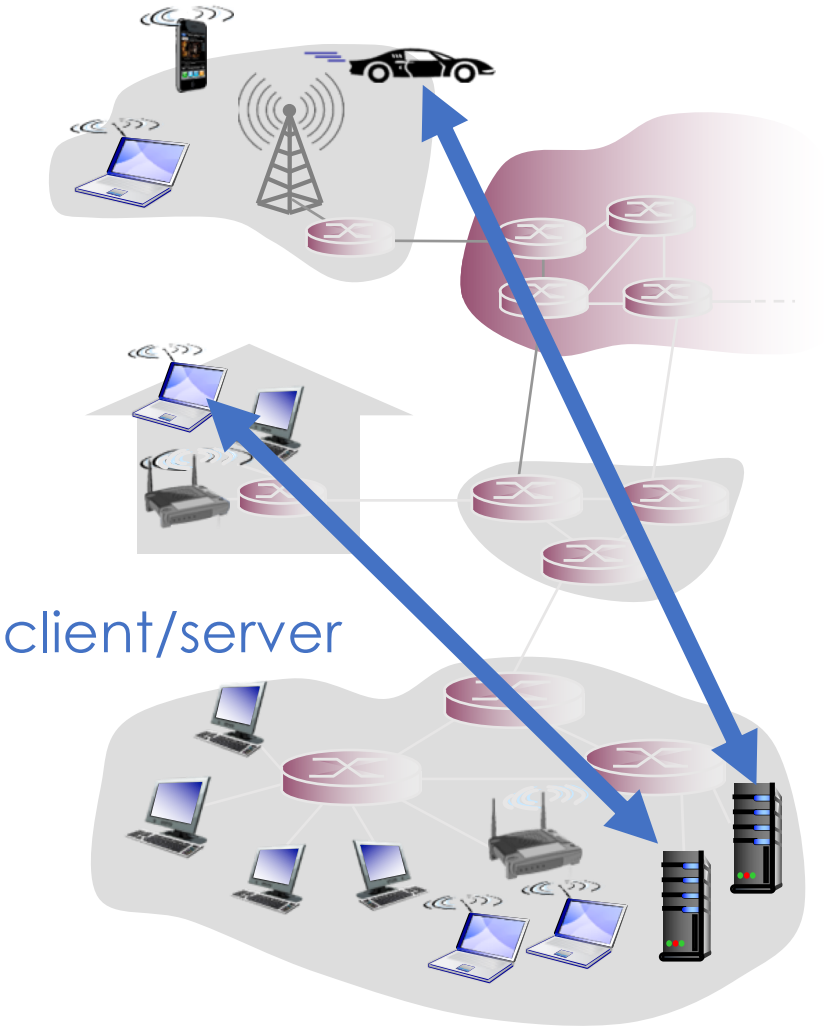
- Enable two end-systems to communicate with each other
- No need to worry about what's happening in core networks

Application Architecture

- **Client-server architecture**
  - All clients send request to the server
  - The server is always on and as a well-known IP address
  - (optional) Load can be share among multiple distributed servers

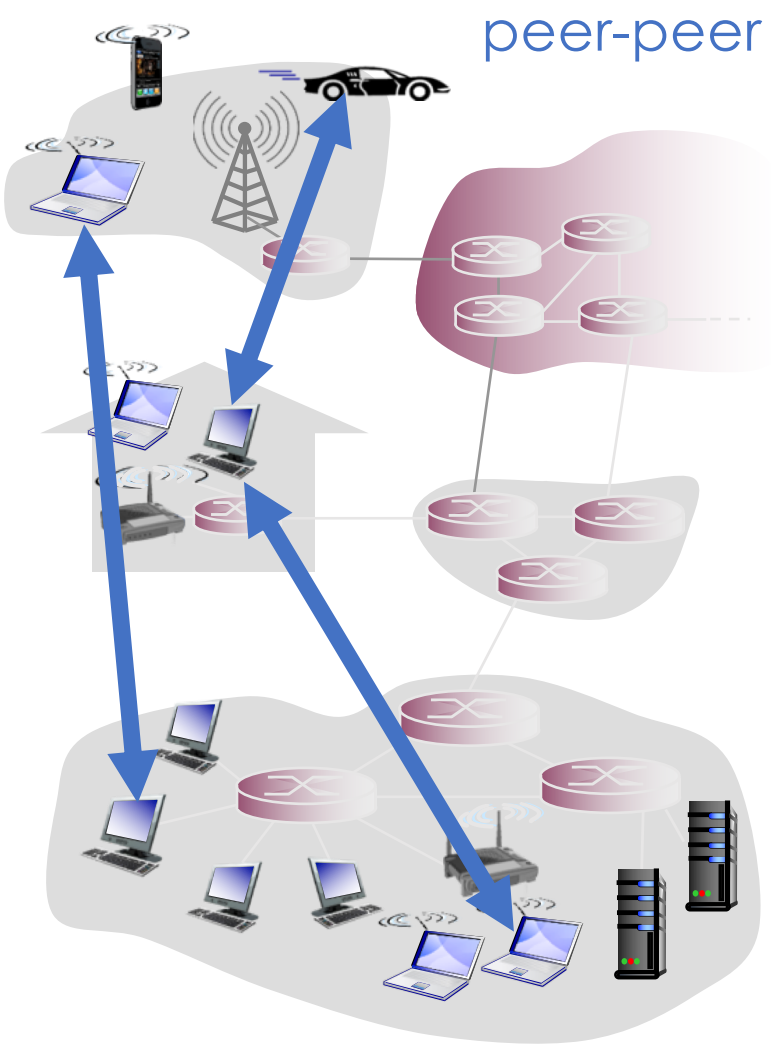
- **Peer-to-peer (P2P) architecture**
  - Any host (usually called peer) can be either a client or a server, or both
  - Well-known applications: skype, BitTorrent

Client-Server Architecture



- **Pros**
  - Easy to manage
  - Easy to locate the content
- **Cons**
  - Scalability
  - Single point failure
  - Might incur longer latency
  - No direct communication among clients

Peer-to-Peer Architecture



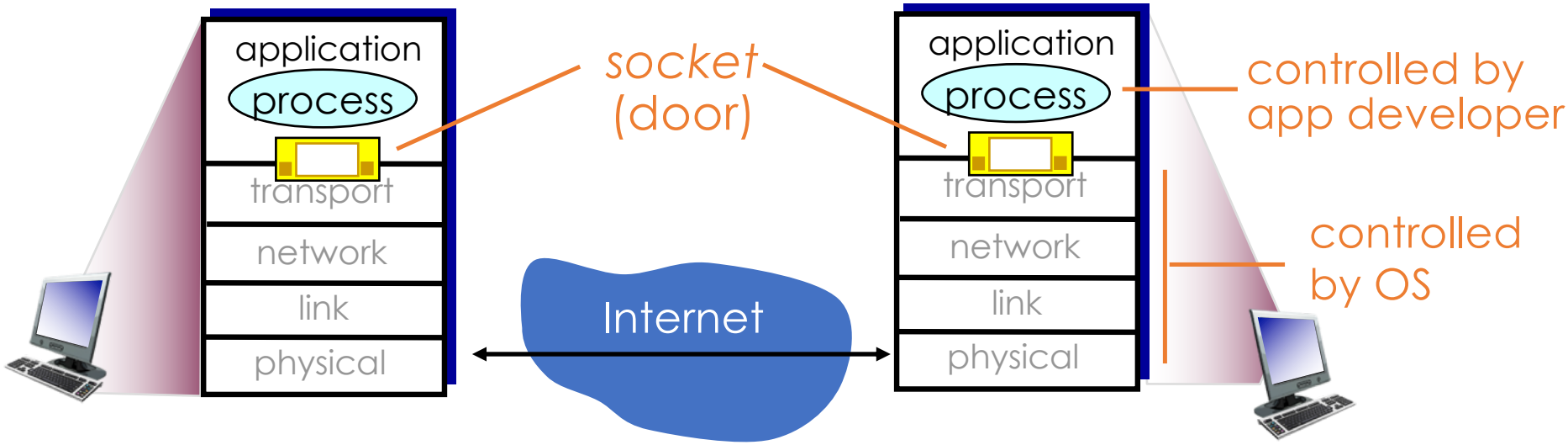
- **Pros**
  - Direct communication among hosts
  - Load sharing
  - **Self-scalability**
- **Cons**
  - Complex management
  - Serving capability
  - Dynamic IP and NAT
  - Security
  - Incentive and copyright

Process Communicating

- **Process**: a programing running within a host
  - Exchange message with another (either in a different host or in the same host)
- Typically, a network application needs a pair of processes
  - Client process: initiate communication
  - Server process: wait for being connected
- Addressing process
  - Each host can have multiple process
  - Unique identifier: **IP address + port number**
  - Well known ports: FTP(21), SSH(22), Telnet(23), SMTP(25), DNS(53), HTTP(80), etc. (check more in Wikipedia)

Socket

- An interface used to connect to another process
- Little control on lower layers
  - Choice of the transport protocol (TCP or UDP)
  - Configure the maximum buffer and segment size



App-Layer Protocols

- Types of messages exchanged
  - Request or response
- Message syntax
  - What fields in messages
  - Field format
- Message semantics
  - Meaning of information in fields
- Rules
  - *When* and *How* processes send and respond to messages

Two types:

1. open protocols:
   - defined in **RFC**s
   - allow for interoperability
   - e.g., HTTP, SMTP
2. proprietary protocols:
   - Define by the developers
   - e.g., Skype

App vs. App-Layer Protocols

- A network application consists of
  - A server (web/mail server)
  - Clients (browsers or mailbox software)
  - A standard defining the structure (HTML or mail message)
  - Application-layer protocols (HTTP or SMTP)

Outline

- Principles of network applications
- **Web and HTTP**
- E-Mail and SMTP
- DNS
- Peer-to-peer applications
- Video streaming

Web and HTTP

- **World Wide Web (WWW)**
  - Providers make the content public, accessible on demand
  - A web page consists of objects
    - Base HTML file and several objects, e.g., a page with 5 images includes 6 objects
    - Each object has a unique URL

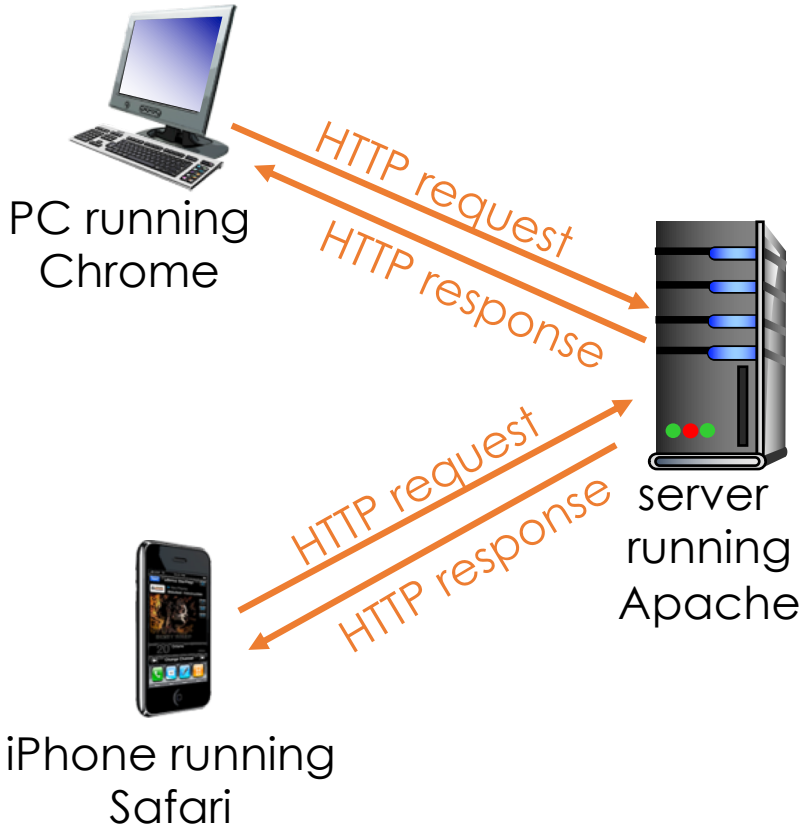
`www.someschool.edu`/`someDept/pic.gif`

host name | object path

- **HyperText Transfer Protocol (HTTP)**
  - RFC 1945, RFC 2616
  - A client program (browser) and a server program (web server) talk to each other

HTTP Overview

- Define how Web clients request Web pages and retrieve objects from Web servers
- Client-server architecture
- Client:
  - browser that sends requests and displays the content
  - e.g., Chrome, Firefox, Safari, IE
- Server:
  - store content and push them to clients
  - e.g., Apache, Microsoft IIS



HTTP Overview

- Use **TCP** as the underlying transport protocol
- Default port: **80**

- Clients
  - Initiate TCP connections
  - **Stateless**: by default, server does not keep the information about clients
- Server
  - Always on, with a fixed IP or domain name (hostname)
  - Accept requests from different clients
  - Number of served clients may be limited

HTTP Connections

- **Non-persistent HTTP**
  - Send each object over a distinct TCP connection
    - k objects need k connections
  - Can be sent
    - back-to-back
    - periodically
    - intermittently
- **Persistent HTTP**
  - Default mode
  - Send multiple objects over a single TCP connection

How Non-Persistent HTTP Works?

suppose user requests a Web page with 10 JPEG images:

`www.school.edu/department/home.index`

1. client initiates TCP connection to server at `www.school.edu` on port 80

2. server at `www.school.edu` waiting for TCP connection at port 80 and "accepts" connection

3. client sends HTTP *request message* into TCP connection indicating the object path `department/home.index`

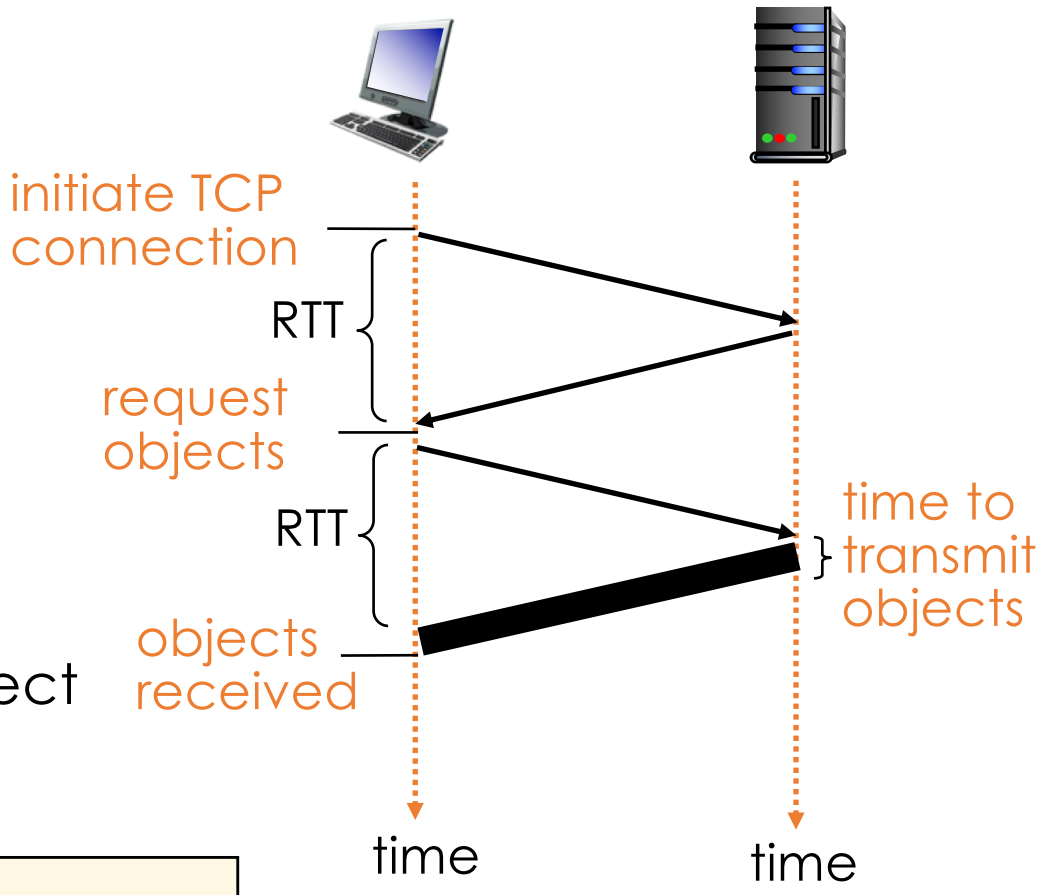
4. server sends *response message* containing requested object and close connection

5. client gets html file, displays html and finds the paths of 10 objects

repeat steps 1-4 for every object!

Non-Persistent HTTP: Response Time

- Assume a browser can create only one connection at a time
- **RTT (round trip time)**
  - Time to traverse from client to server and back
- **Latency**
  - One RTT to establish TCP connection
  - One RTT to request and retrieve the first bit of an object
  - Object transmission time



(2RTT+file transmission time) x N objects

Q: What if a browser supports *k* parallel connection?

Why Persistent HTTP

- **Issues of non-persistent HTTP**
  - Each TCP connection needs a buffer
    → burden on server
    → few clients served as #connections are limited
  - Longer latency
    → each object suffers from 2RTT latency

- **Persistent HTTP**
  - One RTT for all objects
  - Lower load
  - Requests can be sent back-to-back, without waiting for response
  - Server closes the connection after timeout

HTTP Message Format

- Two types of HTTP messages: request, response
- HTTP request message:
  - ASCII (human-readable format)
  - See response message format in the book

Some Reference

- RFC1945 Hypertext Transfer Protocol -- HTTP/1.0
- RFC2068 Hypertext Transfer Protocol -- HTTP/1.1 (obsoleted by RFC2616)
- RFC2616 Hypertext Transfer Protocol -- HTTP/1.1
- HTTP Specifications and Drafts page at the W3C
- RFC2660 The Secure HyperText Transfer Protocol (S-HTTP)

- Tutorial given by Wireshark
  - https://wiki.wireshark.org/Hyper_Text_Transfer_Protocol
- Capture HTTP packets using Wireshark
  - https://www.youtube.com/watch?v=tLNCG-Jrq90

HTTP Method Types

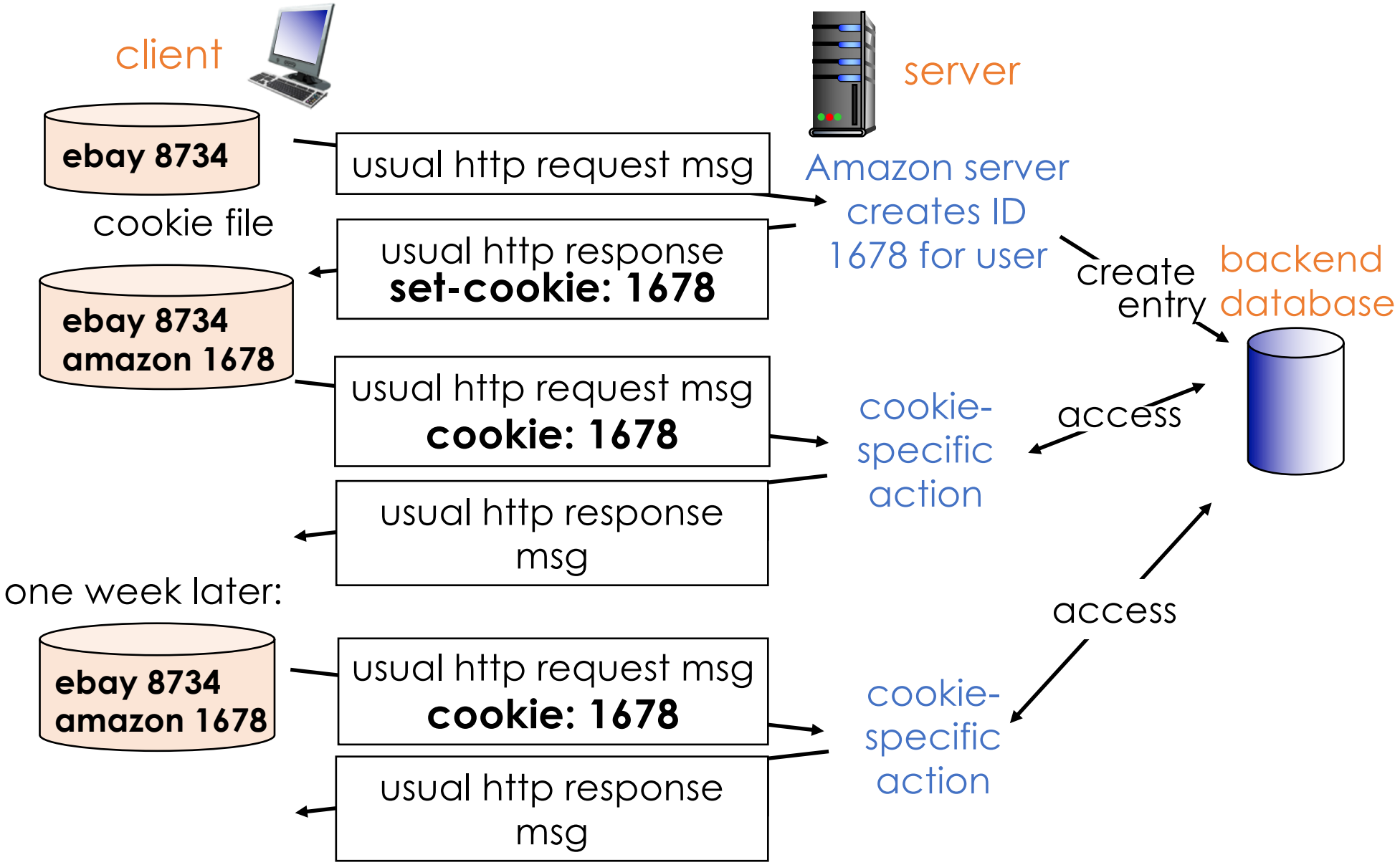
- **GET**
  - Request objects from server
- **POST**
  - User enters forms or gives input
  - Alternative: user GET but "embed" input in the URL

    `www.somesite.com/animalsearch?monkeys&banana`
- **PUT**
  - Upload objects to server
- **HEAD**
  - For debugging
- **DELETE**
  - Clients delete objects on a server

How to Interact with Clients?

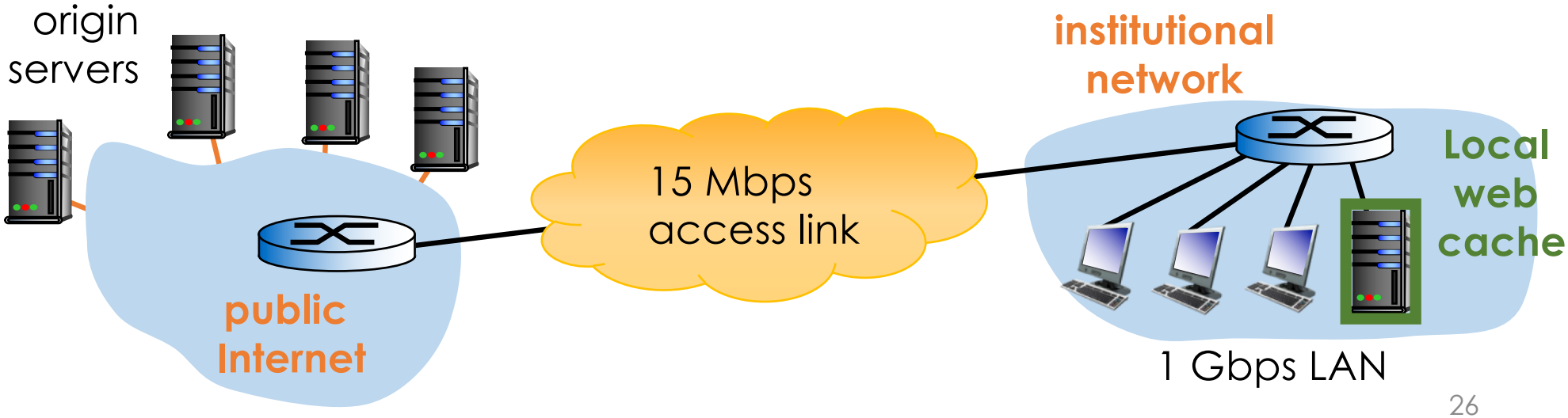
- If HTTP is stateless, how to provide **"personal"** service?
  - Why don't you need to re-login to gmail server every time?
  - Why PCHome can keep your personal information?
- How to **"identify user"**? **Cookies** [RFC 6265]
- Four components of cookie
  1. Cookie header line in HTTP response
  2. Cookie header line in HTTP request
  3. Cookie file kept in the end system and managed by the browser
  4. Back-end database at the server

How Cookies Work?



Web Cache (Proxy)

- Why do we need **cache**?
  - Server accepts burst requests → Share loading
  - Traffic through the core network experiences long latency and bottleneck → Reduce latency/traffic
  - Underutilize LAN capacity → Respond faster
- How cache works?
  - Store web objects in local machines (cheap!)

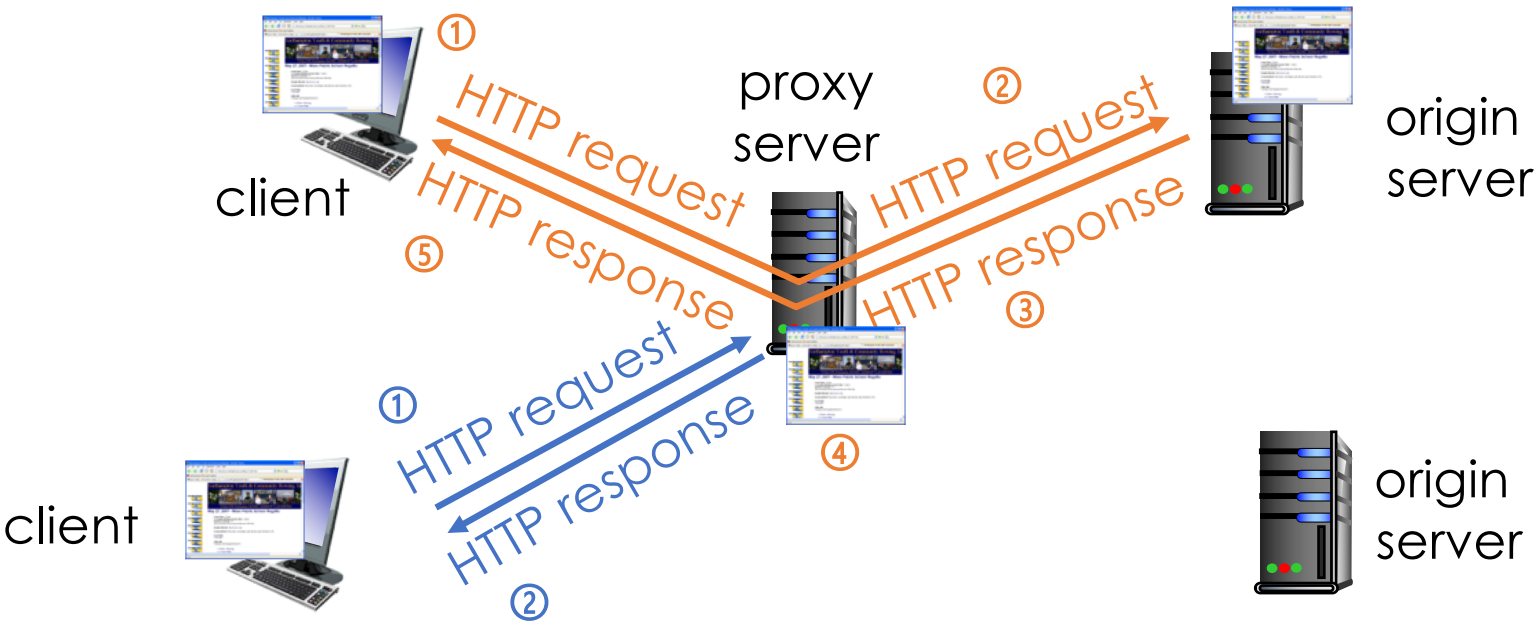


Web Cache (Proxy)

- Typically installed by an ISP, configured by users (in the browser)
- Why less used now? **Contend Delivery Network (CDN)**

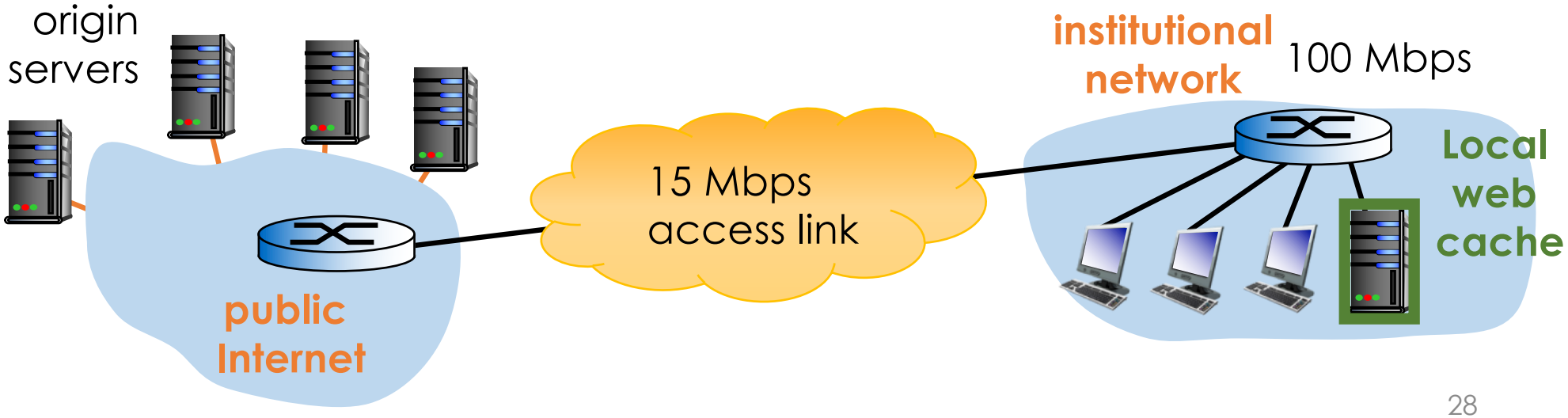
Case 1: cache miss
Request (2), store (4) and forward (5)!

Case 2: cache hit
Forward directly



Cache: How Latency Reduced?

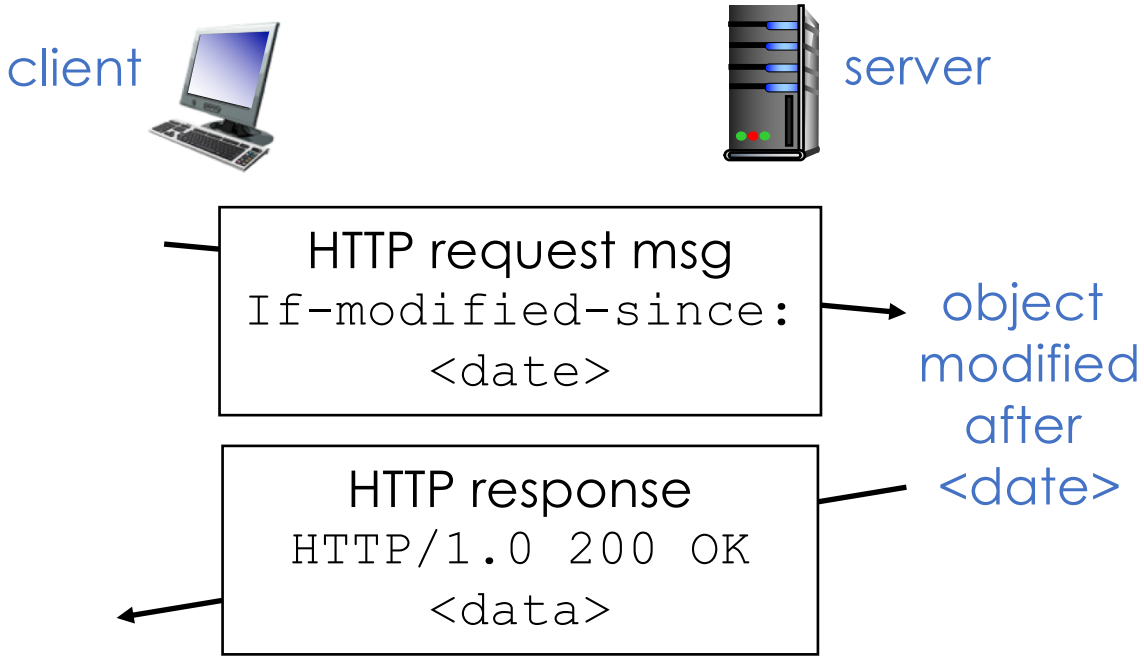
- Recall that "latency increases rapidly with traffic intensity (TI)"
- No cache: 15 requests/sec for an 1 Mb object
  - TI on access: (15 req/sec)*(1Mb/req)/(15Mbps) = 1
    → queuing delay grows infinitely
- With cache: assume 40% requests served locally
  - TI on LAN: (15*0.4 req/sec)*(1Mb/req)/(100Mbps) = 0.06
  - TI on access: (15*0.6 req/sec)*(1Mb/req)/(15Mbps) = 0.6



Issues of Cache

- How to ensure consistency between caches and servers?
  - Conditional GET: keep cached data up-to-date
  - Only fetch the modified files

- Cache:
  - specify date of cached copy
- Server:
  - respond "304 not modified"
  - or respond "200 OK" + modified files



Try It Yourself

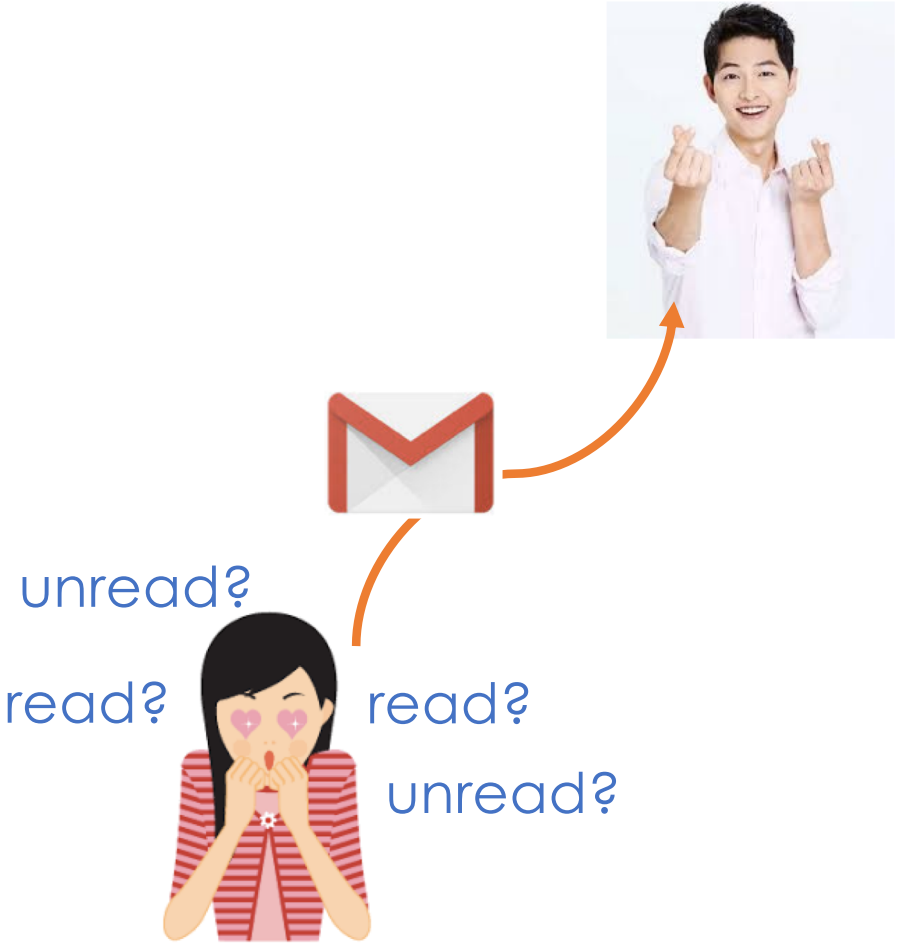
- Build a web server
  - Linux + Apache + PHP + MySQL + Authentication/SSL
- Download web objects via commend lines
  - wget URL
- Trace HTTP traffic using WireShark

Outline

- Principles of network applications
- Web and HTTP
- **E-Mail and SMTP**
- DNS
- Peer-to-peer applications
- Video streaming

Electronic Mail (E-Mail)

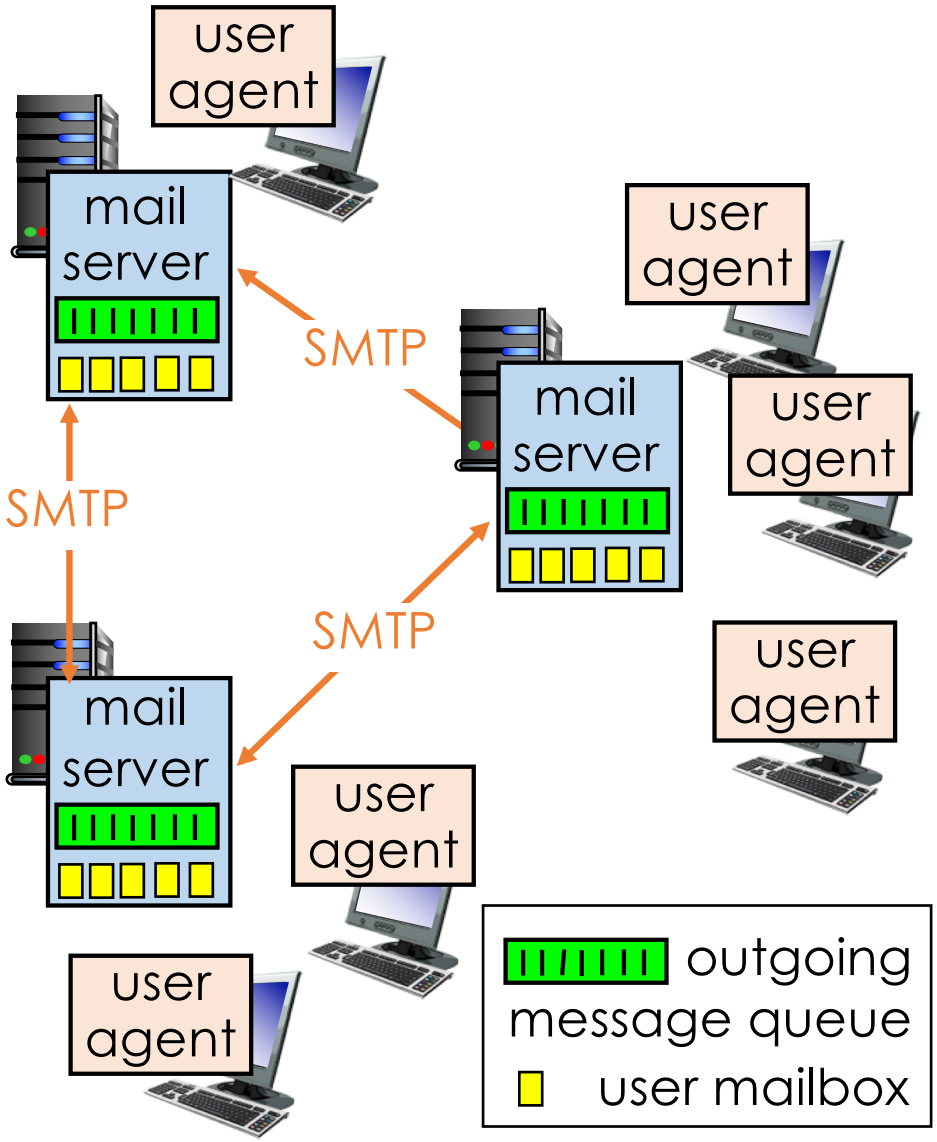
- **Asynchronous communication**



- Synchronous communication

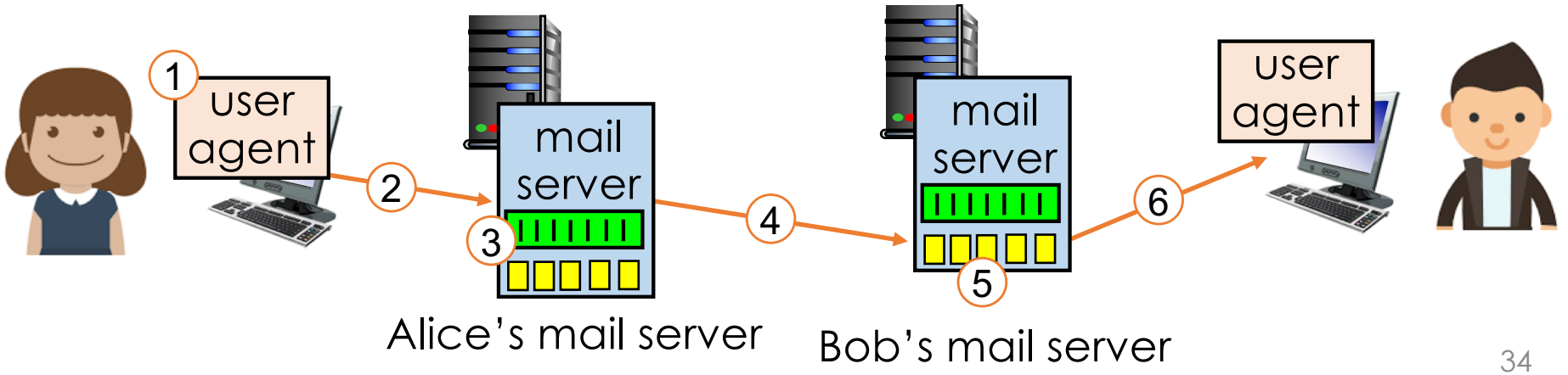
Electronic Mail (E-Mail)

- Client-server architecture
- Three components
  - **User agents**: Outlook (Windows), Mail (Apple), Mutt (Linux), Pine (Linux), Browser
  - **Mail Servers:** manages **mailbox** of users
  - **SMTP: Simple Mail Transfer Protocol**



Mail Servers

- A mailbox contains message for a particular user
- Sending mailbox
  - Receiving mails from clients
  - In charge of forwarding mails to receiving mailbox (retry every few minutes if receiving server is off)
- Receiving mailbox
  - Cache and forward mails to the agent of receiver
- **SMTP**: protocol among mail servers



SMTP Interaction

See here for ssl access
$: openssl s_client -starttls smtp -connect smtp.gmail.com:587

telnet serverName 25

```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250  Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?
C: How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```

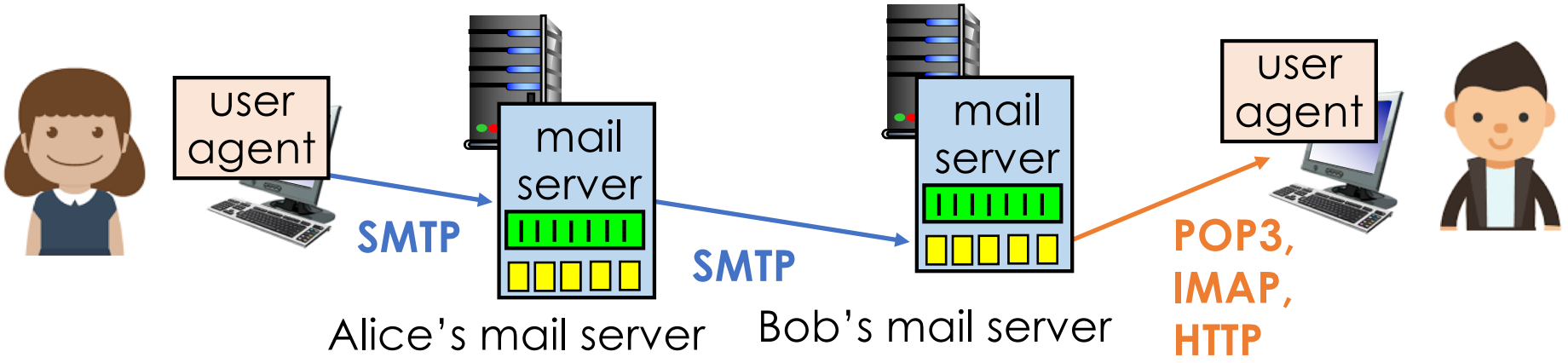
SMTP

- Persistent connection
  - Send messages on a single TCP connection
- Messages in 7-bit ASCII
  - All files (including executable files or binary images) should be encoded into plain text format
- **Push protocol**
  - unlike HTTP, which is a pull protocol

Mail Access Protocols

- How to read messages from the mail server?
  - User agent may not always be on
  - A user may have different devices, all attempting to access the same messages
- Note that "SMTP is used between two mail servers, not between a mail server and a user agent"
- **Mail Access protocol**: retrieve mails from mail server to user agent
  - **POP**: (port 110) Post Office Protocol [RFC 1939]: authorization, download
  - **IMAP**: (port 143) Internet Mail Access Protocol [RFC 1730]: more features, e.g., creating folders
  - **HTTP**: gmail, Hotmail, Yahoo! Mail, etc.

Mail Access Protocols



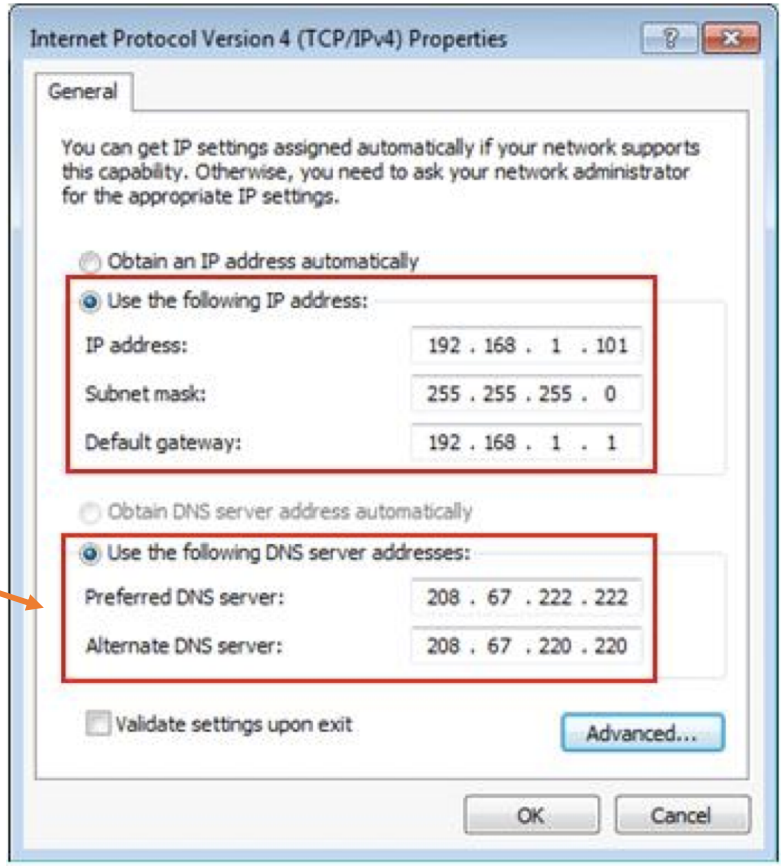

```
$ telnet mailServer 110
$ openssl s_client -starttls pop3
    -connect pop3.cs.nctu.edu.tw:110
S: +OK NCTU CS Dovecot ready.
C: user katelin
S: +OK
C: pass ********
S: +OK Logged in.
C: list
C: retr 299
C: dele 299
```

Outline

- Principles of network applications
- Web and HTTP
- E-Mail and SMTP
- **DNS**
- Peer-to-peer applications
- Video streaming

Domain Name System (DNS)

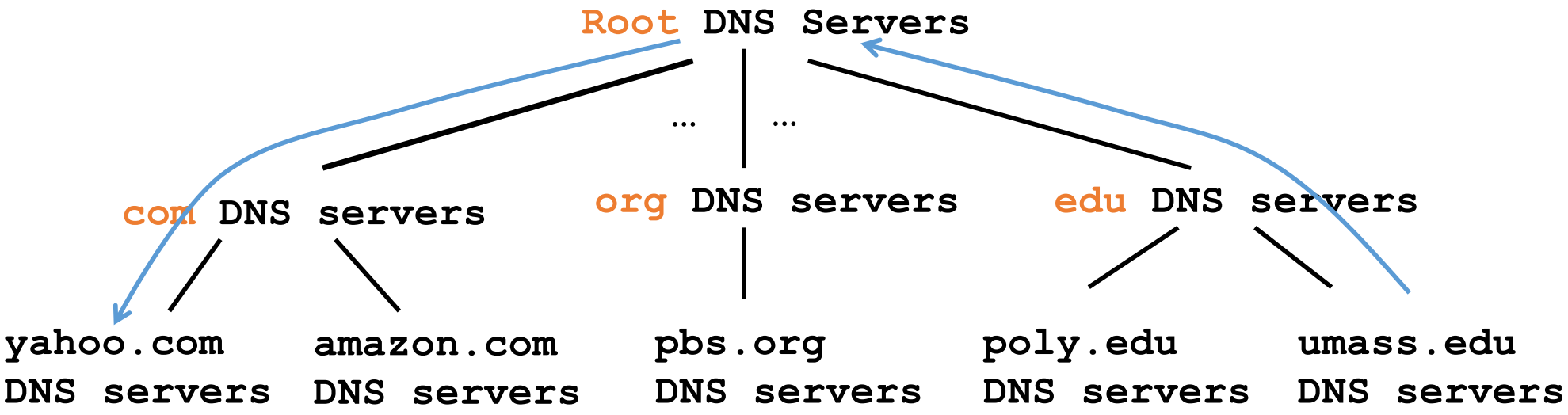
- Translator: hostname ⇔ IP address
  - Hostname: mnemonic name (e.g., www.cs.nctu.edu.tw)
  - IP address: unique ID (e.g., 140.113.235.47)
- NCTU DNS:
  - 140.113.1.1
  - 140.113.6.2
  - 140.113.250.135
- Run on **UDP**, port **53**



Why Need a Protocol?

- How to **"quickly"** find the entry in a huge dictionary?
  - **Distributed + Hierarchy**

Distributed, Hierarchical Database



- No worry about single point failure
- Load sharing
- Smaller storage for each server
- Shorter latency
- Maintenance

Different Types of Server

- **Top-level Domain (TLD) severs**
  - responsible for com, org, net, edu, aero, jobs, museums, and all top-level country domains, e.g., uk, fr, ca, jp
  - *Network Solutions* maintains *.com* TLD; *Educause* for *.edu* TLD
- **Authoritative** DNS servers
  - organization's DNS servers (e.g., NCTU, CS)
  - authorize hostname to IP mappings
  - TLD may only know intermediate server (e.g., dns.cs.nctu.edu.tw), which forwards query to authoritative server
- **Local DNS Server**
  - does not strictly belong to the hierarchy
  - each ISP (residential, company, university) has one
  - also called default name server

Two Query Methods

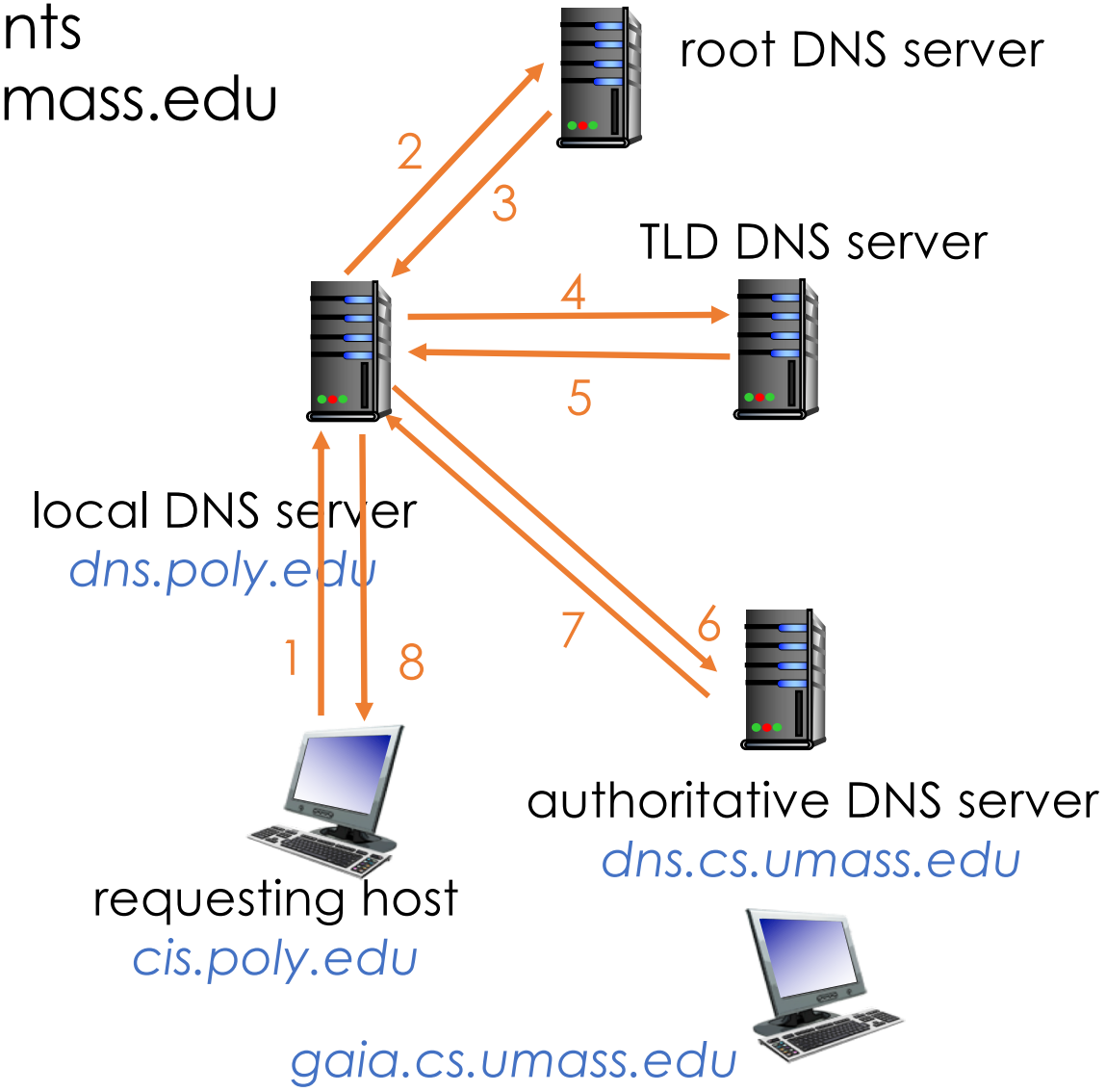
- **Iterative query**
  - Just help find the hostname of the next DNS server
  - No forwarding
- **Recursive query**
  - Not only find the next DNS server
  - But also forward the request

"Iterative" Query

host at cis.poly.edu wants
IP address for gaia.cs.umass.edu

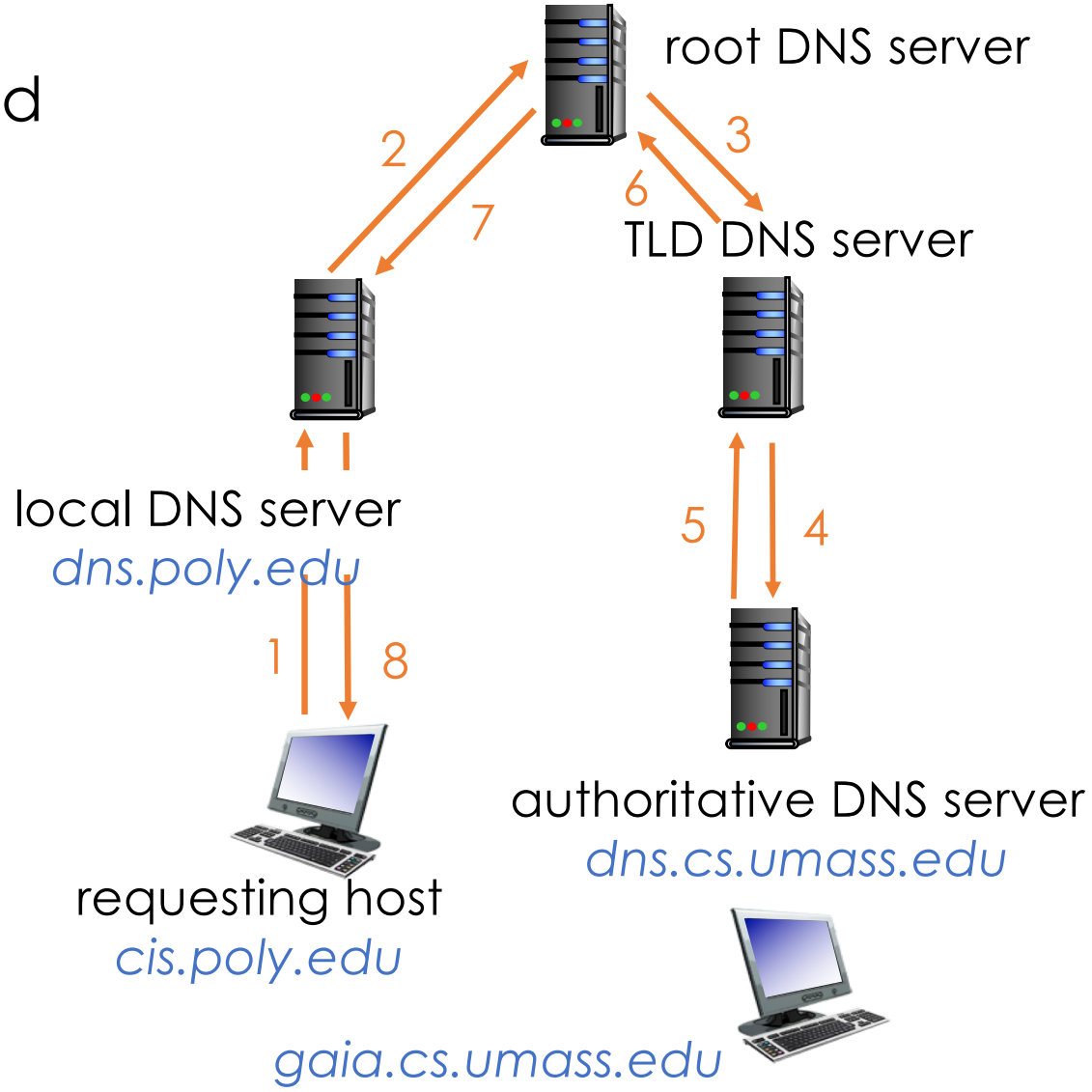
1. contacted server replies with name of server to contact
   → "I don't know this name, but ask this server"
2. no forwarding

Steps 1 and 8 are recursive
Steps 2-7 are iterative



"Recursive" Query

- Puts burden of name resolution on contacted name server
- Heavy load at upper levels of hierarchy?



DNS Caching

- Once a server gets a mapping, it can cache it
  - Need a timeout (TTL) since IP might be dynamic
  - TLD typically cached in local name servers
    → root server rarely accessed
- Update/notify mechanisms proposed IETF standard
  - RFC 2136

DNS Records

distributed database storing **resource records (RR)**

RR format: `(name, value, type, ttl)`

- **type=A**
  - **`name`**: hostname
  - **`value`**: IP addr
- **type=NS**
  - **`name`**: domain (e.g., nctu.edu.tw)
  - **`value`**: hostname of authoritative name server
- **Type=CNAME**
  - **`name`**: alias name for some canonical (real) name
  - **`value`**: canonical name
- **type=MX**
  - **`value`**: name of mailserver

Insert Records into DNS

- Example: new startup *"Network Utopia"*
- Register name `networkuptopia.com` at DNS **registrar** (e.g., Network Solutions)
  - provide names, IP addr of authoritative name server
  - registrar inserts two RRs into .com TLD server:

```
(networkutopia.com, dns1.networkutopia.com, NS)
(dns1.networkutopia.com, 212.212.212.1, A)
```

- **Authoritative DNS**: create authoritative server type A record for `www.networkuptopia.com`; type MX record for `networkutopia.com`