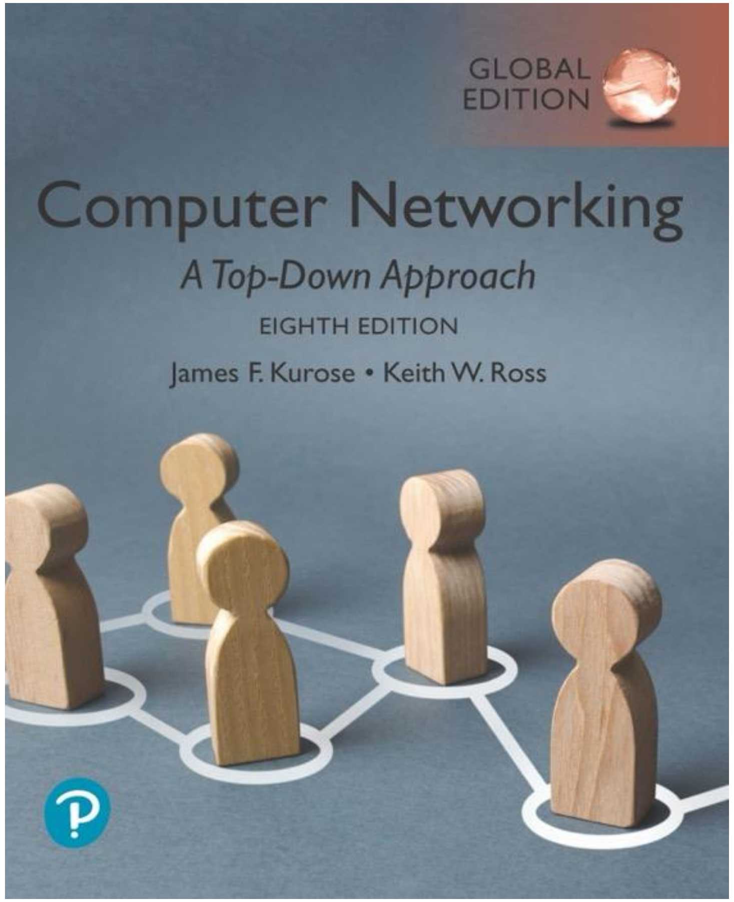
Chapter 2
Application Layer

Ren Ping Liu
renping.liu@uts.edu.au
adapted from textbook slides by JFK/KWR

11 March 2024



Computer Networking: A Top-Down Approach
8th Edition, Global Edition
Jim Kurose, Keith Ross

Application layer: overview

2.1 Principles of network applications

2.2 Web and HTTP

2.3 E-mail, SMTP, IMAP

2.4 The Domain Name System DNS

~~2.5 P2P applications~~

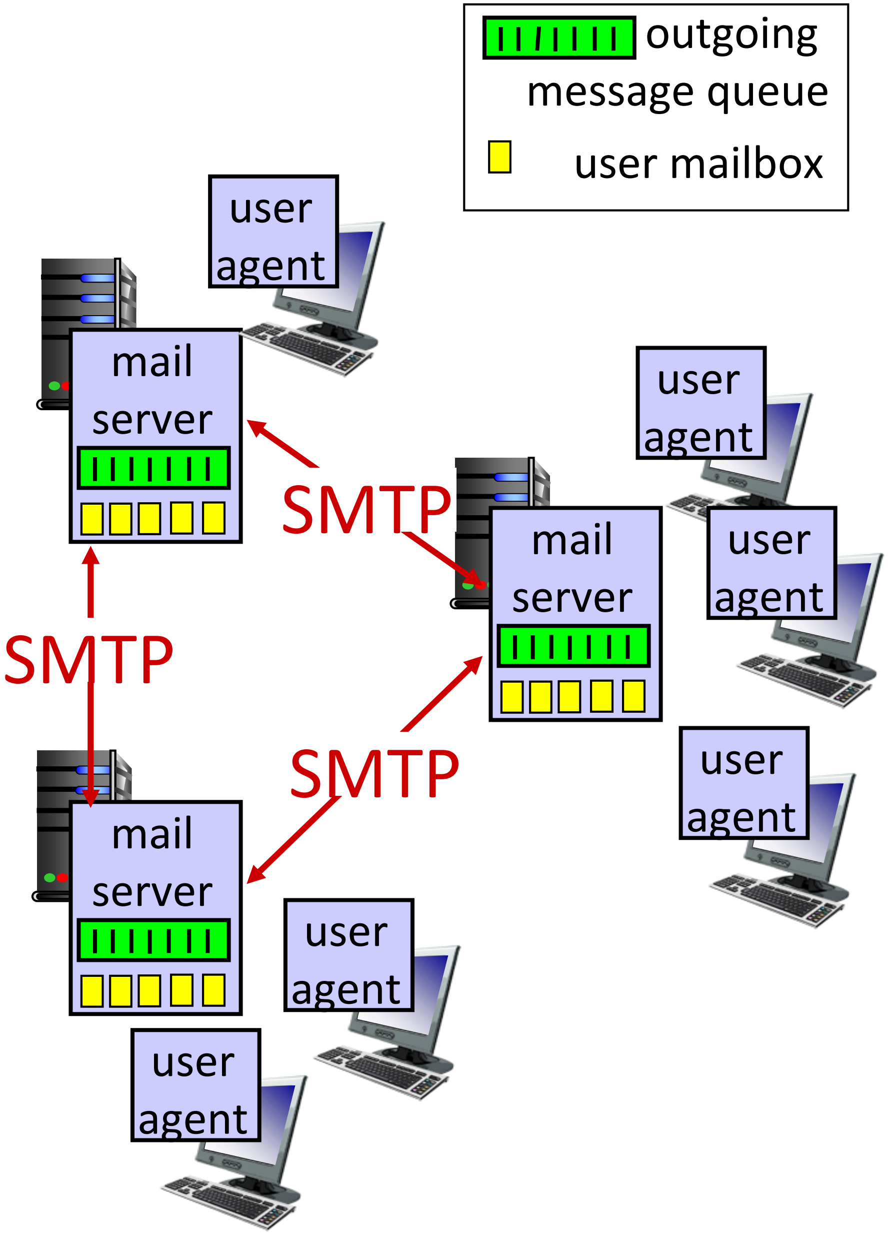
~~2.6 video streaming and content distribution networks~~

2.7 socket programming with UDP and TCP

E-mail

Three major components:

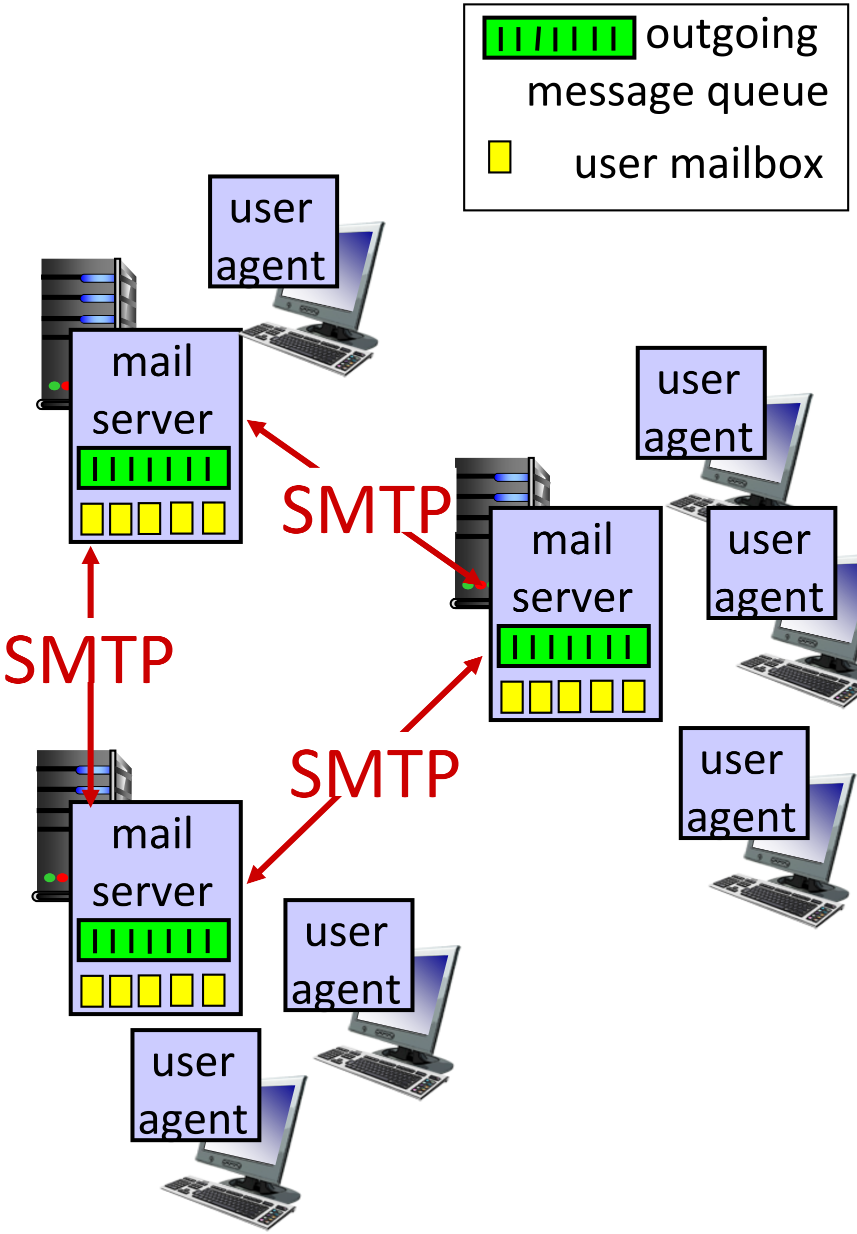
- user agents
- mail servers
- email protocols:
  - SMTP (simple mail transfer protocol)
  - Mail access protocols: POP3, IMAP



E-mail: User Agent

User Agent

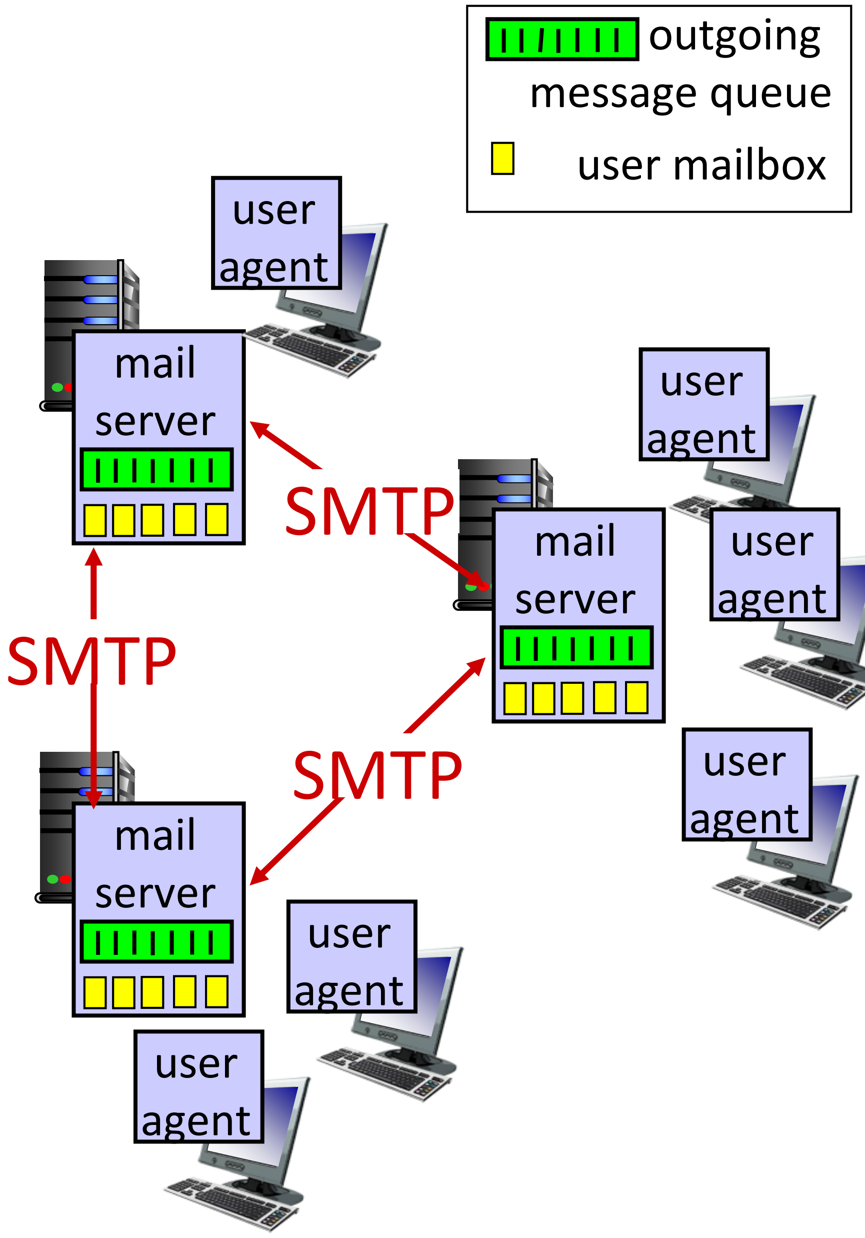
- a.k.a. "mail reader"
- composing, editing, reading mail messages
- e.g., Outlook, iPhone mail client
- outgoing, incoming messages stored on server



E-mail: mail servers

mail servers:

- *mailbox* contains incoming messages for user
- *message queue* of outgoing (to be sent) mail messages
- *SMTP protocol* between mail servers to send email messages
  - client: sending mail server
  - “server”: receiving mail server



E-mail protocol: SMTP (RFC5321)

- uses TCP to reliably transfer email message from client (mail server initiating connection) to server, port 25
- direct transfer: sending server (acting like client) to receiving server
- three phases of transfer
  - handshaking (greeting)
  - transfer of messages
  - closure
- command/response interaction (like HTTP)
  - commands: ASCII text
  - response: status code and phrase
- messages must be in 7-bit ASCII

Scenario: Alice sends e-mail to Bob

1) Alice uses UA to compose e-mail message “to” bob@someschool.edu

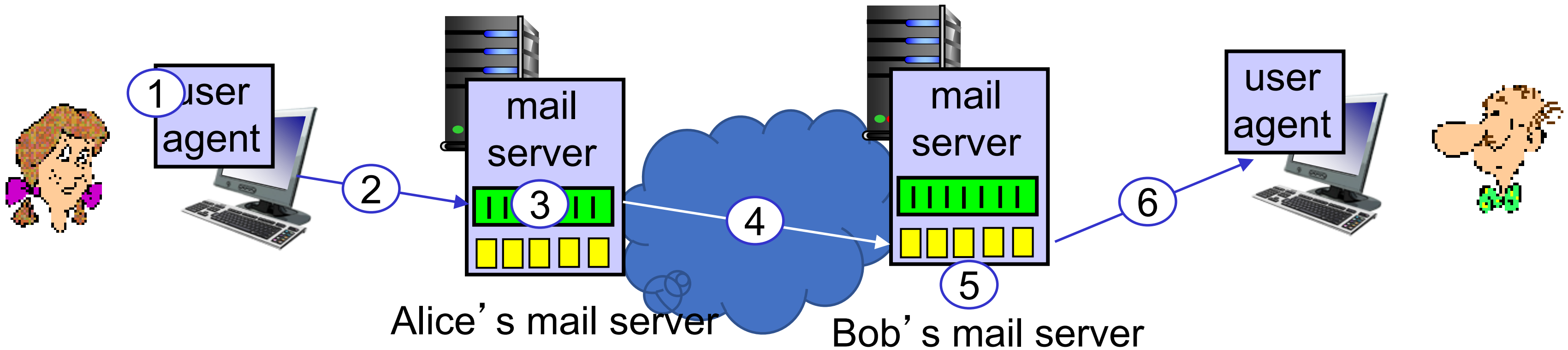
2) Alice’s UA sends message to her mail server; message placed in message queue

3) client side of SMTP opens TCP connection with Bob’s mail server

4) SMTP client sends Alice’s message over the TCP connection

5) Bob’s mail server places the message in Bob’s mailbox

6) Bob invokes his user agent to read message



Sample SMTP interaction

telnet mail.uts.edu.au 25

```
S: 220 mail.uts.edu.au
C: HELO uts.edu.au
S: 250  Hello uts.edu.au, pleased to meet you
C: MAIL FROM: <alice@uts.edu.au>
S: 250 alice@uts.edu.au... 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: From: alice@uts.edu.au
C: To: bob@hamburger.edu
C: Subject: Searching for the meaning of life.
C: Do you like ketchup?
C: How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 mail.uts.edu.au closing connection
```

contact server (HELO)

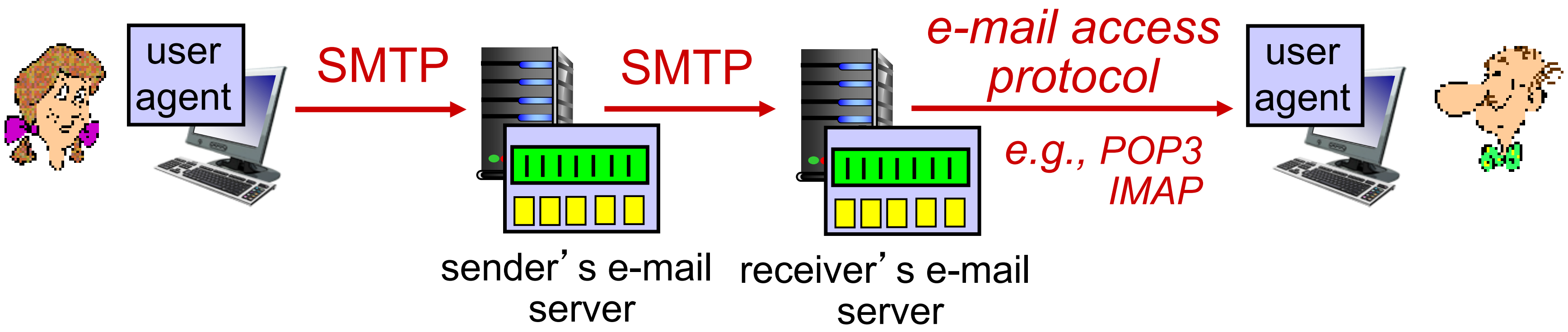
beginning of mail message (DATA)

end of mail message (.)

Could be: <president@uts.edu.au>

Can be different from header <president@whitehouse.gov>

Mail access protocols



- SMTP: delivery/storage of e-mail messages to receiver’s server
- mail access protocol: retrieval from server
  - POP3 or IMAP: provides retrieval, deletion of stored messages on server
- HTTP: gmail, Hotmail, Yahoo!Mail, etc. provides web-based interface on top of SMTP (to send), IMAP (or POP3) to retrieve e-mail messages

SMTP: closing observations

comparison with HTTP:

- HTTP: pull
- SMTP: push
- POP3 and IMAP: pull or push?
- both have ASCII command/response interaction, status codes

- SMTP uses persistent connections
- SMTP requires message (header & body) to be in 7-bit ASCII
- SMTP server uses `.` in one line by itself to signal the end of message

Application layer: overview

2.1 Principles of network applications

2.2 Web and HTTP

2.3 E-mail, SMTP, IMAP

2.4 The Domain Name System DNS

~~2.5 P2P applications~~

~~2.6 video streaming and content distribution networks~~

2.7 socket programming with UDP and TCP

DNS: Domain Name System

people: many identifiers:

- SSN, name, passport #

Internet hosts, routers:

- IP address (32 bit) - used for addressing datagrams
- "name", e.g., cs.umass.edu - used by humans

Q: how to map between IP address and name, and vice versa ?

Domain Name System:

- *distributed database* implemented in hierarchy of many *name servers*
- *application-layer protocol:* hosts, name servers communicate to *resolve* names (address/name translation)
  - note: core Internet function, *implemented as application-layer protocol*
  - complexity at network's "edge"

DNS: services, structure

DNS services

- hostname to IP address translation
- host aliasing
  - canonical, alias names: one IP address corresponds to multiple host names
- mail server aliasing
- load distribution
  - replicated Web servers: many IP addresses correspond to one name

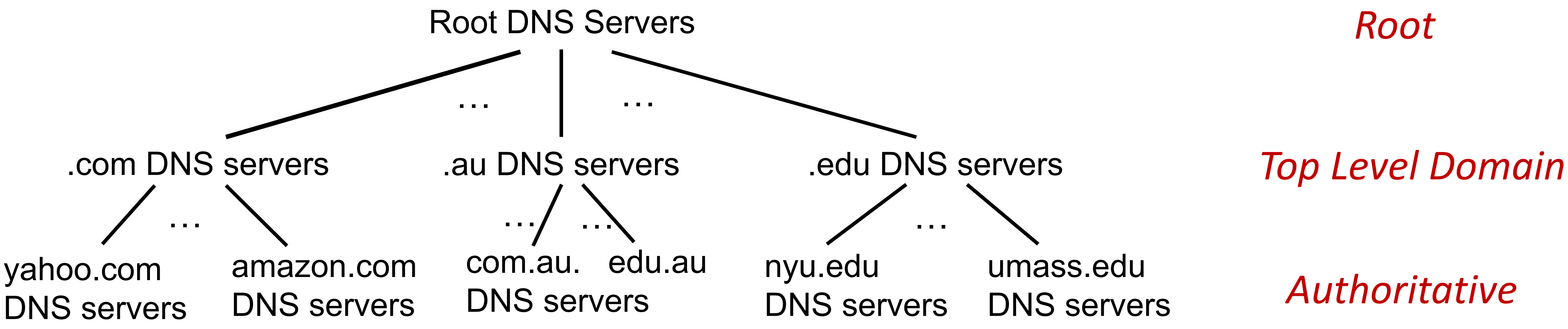
Q: Why not centralize DNS?

- single point of failure
- traffic volume
- distant centralized database
- maintenance

A: doesn't scale!

- Comcast DNS servers alone: 600B DNS queries per day

DNS: a distributed, hierarchical database



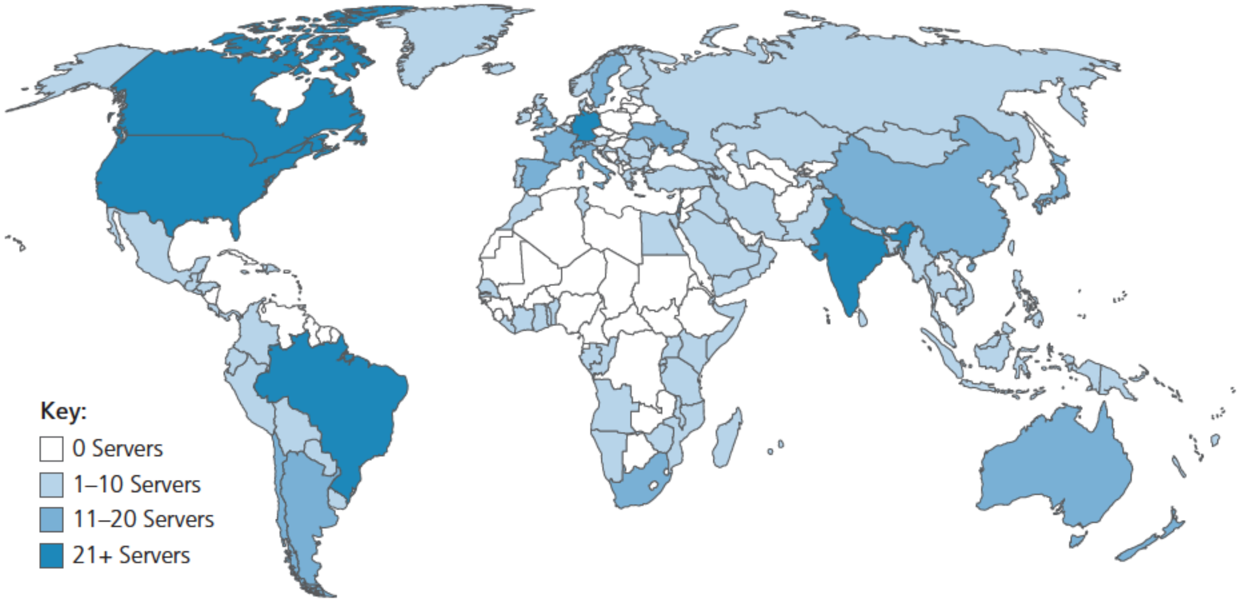
Client wants IP address for www.amazon.com; 1st approximation:

- client queries root server to find .com DNS server
- client queries .com DNS server to get amazon.com DNS server
- client queries amazon.com DNS server to get IP address for www.amazon.com

DNS: root name servers

- official, contact-of-last-resort by name servers that can not resolve name
- *incredibly important* Internet function
  - Internet couldn't function without it!
  - DNSSEC – provides security (authentication and message integrity)
- ICANN (Internet Corporation for Assigned Names and Numbers) manages root DNS domain

13 logical root name "servers" worldwide each "server" replicated many times (~200 servers in US)



TLD: authoritative servers

Top-Level Domain (TLD) servers:

- responsible for .com, .org, .net, .edu, .aero, .jobs, .museums, and all top-level country domains, e.g.: .au, .cn, .uk, .fr, .ca, .jp
- Network Solutions: authoritative registry for .com, .net TLD
- Educause: .edu TLD

Authoritative DNS servers:

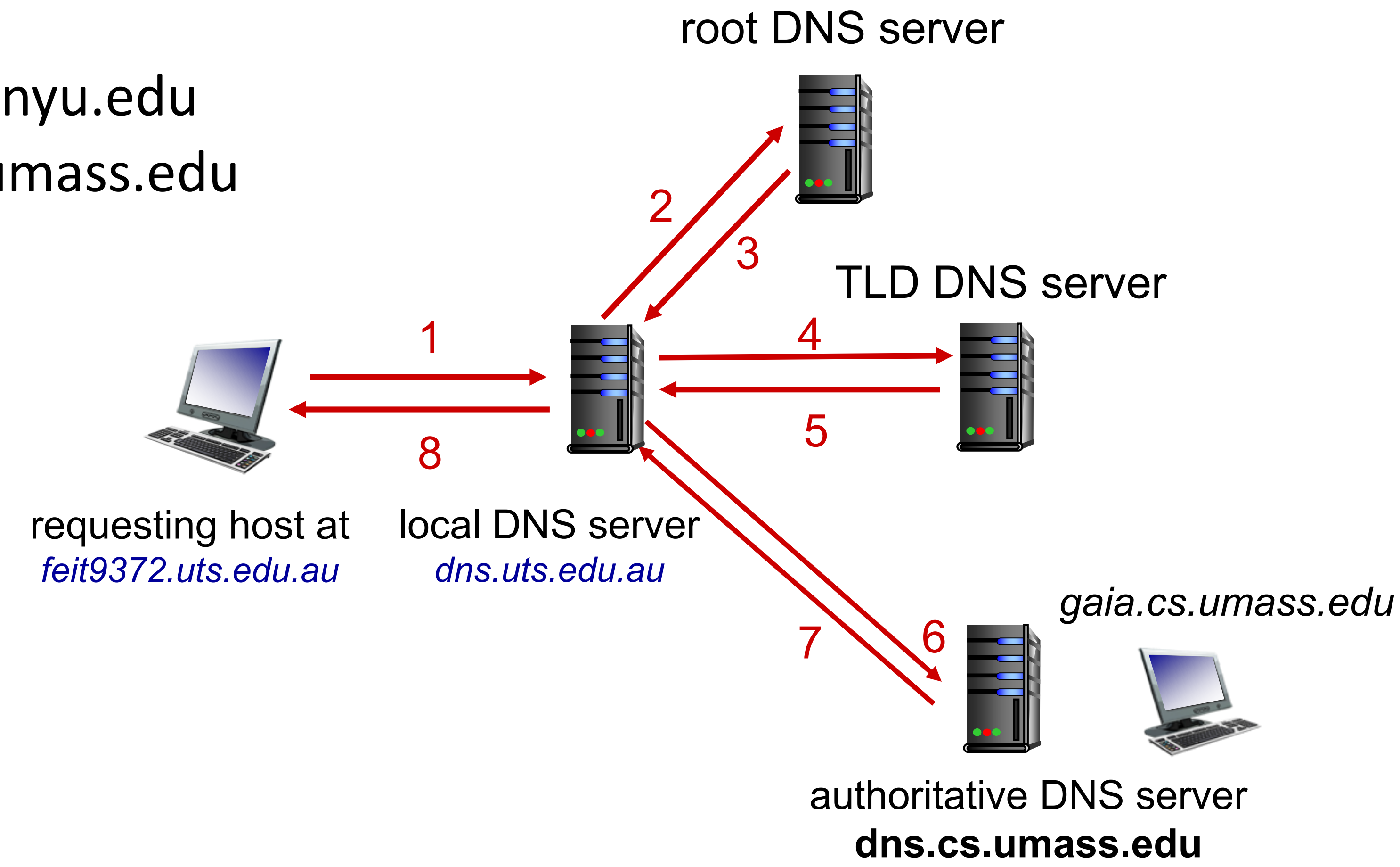
- organization's own DNS server(s), providing authoritative hostname to IP mappings for organization's named hosts
- can be maintained by organization or service provider:
  - amazon.com, google.com, ...

Local DNS name servers

- does not strictly belong to hierarchy
- each ISP (residential ISP, company, university) has one
  - also called "default name server"
- when host makes DNS query, query is sent to its local DNS server
  - has local cache of recent name-to-address translation pairs (but may be out of date!)
  - acts as proxy, forwards query into hierarchy

DNS name resolution: iterated query

Example: host at engineering.nyu.edu wants IP address for gaia.cs.umass.edu



Iterated query:

- contacted server replies with name of server to contact
- "I don't know this name, but ask this server"

Mid-break

- Q & A

Application layer: overview

2.1 Principles of network applications

2.2 Web and HTTP

2.3 E-mail, SMTP, IMAP

2.4 The Domain Name System DNS

~~2.5 P2P applications~~

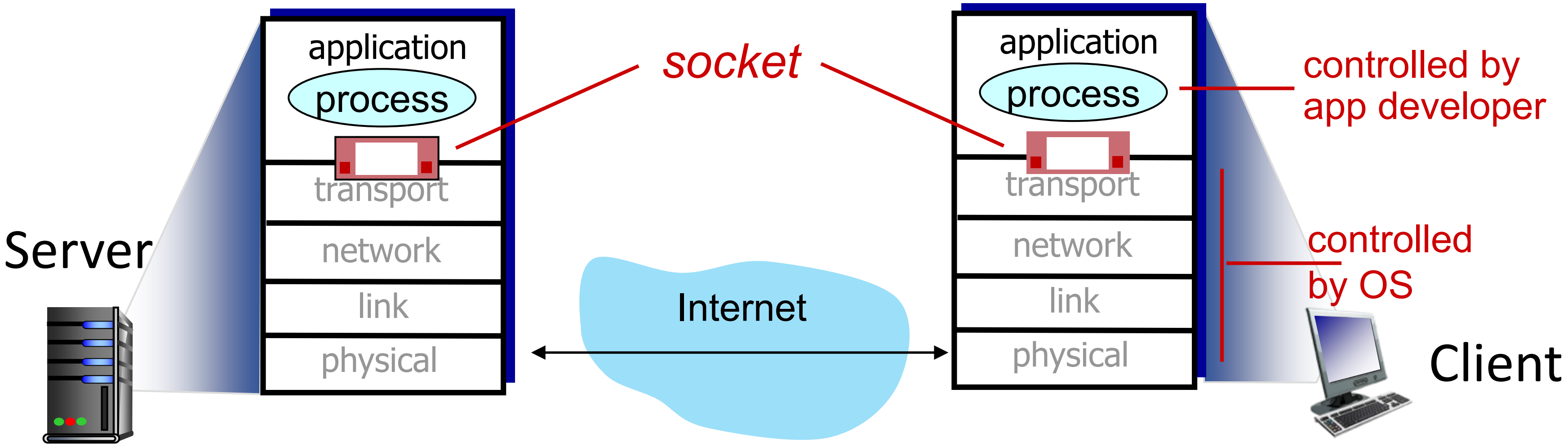
~~2.6 video streaming and content distribution networks~~

2.7 socket programming with UDP and TCP

Socket programming

goal: learn how to build client/server applications that communicate using sockets

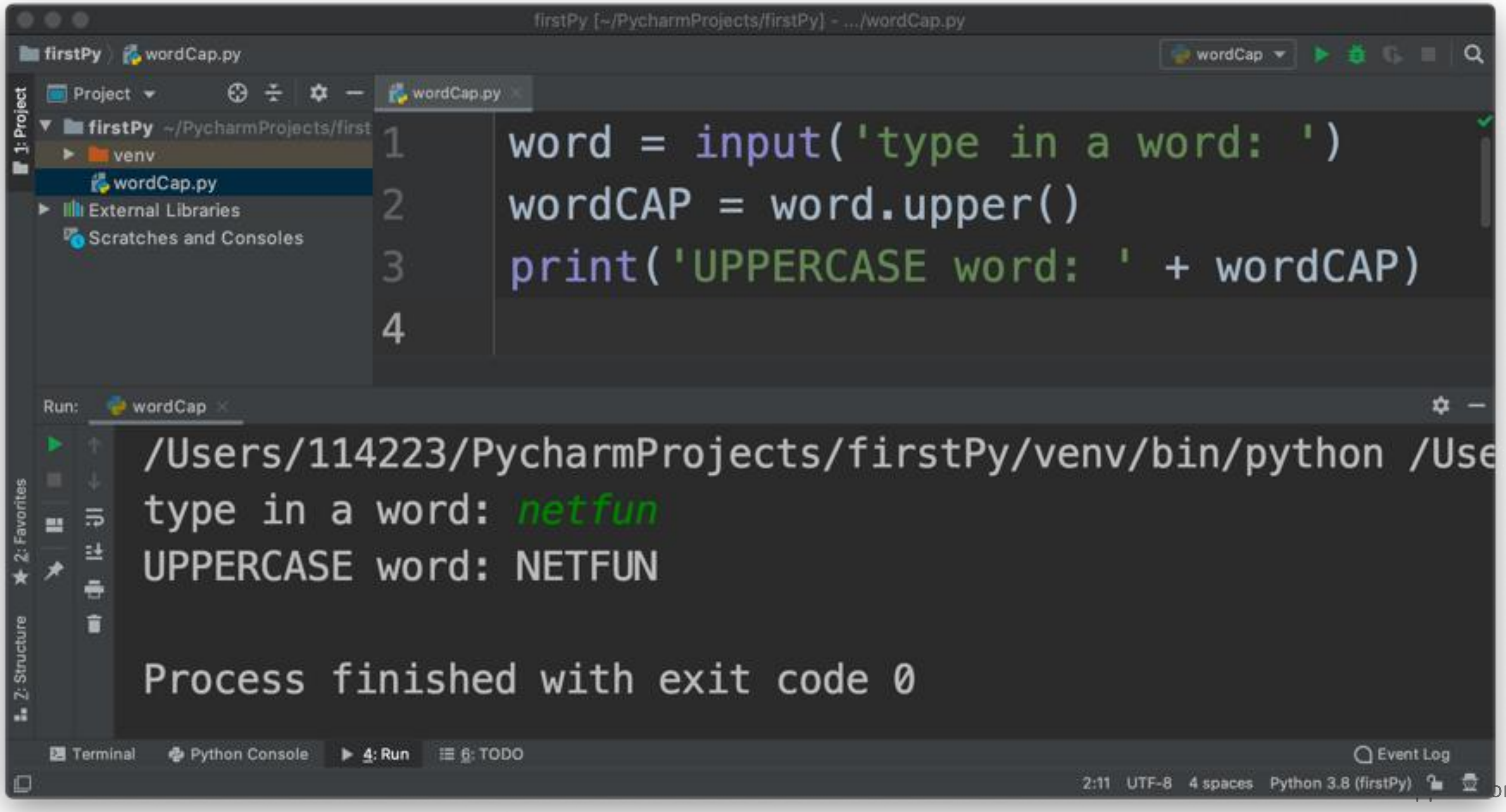
socket: door between application process and end-end-transport protocol



Create and Run your first Python!

In PyCharm: + Create New Project: firstPy

- Create new file: wordCap.py

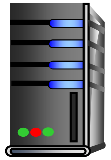


Socket programming

Two socket types for two transport services:

- *UDP:* unreliable datagram
- *TCP:* reliable, byte stream-o

This week's Lab!



Server

Application Example:

hello ←

Client

1. client reads a line of characters (data) from its keyboard and sends data to server

hello

HELLO ←

2. server receives and converts characters to uppercase

HELLO

→

3. server sends modified data to client
4. client receives modified data and displays line on its screen

HELLO →

Socket programming with UDP

UDP: no "connection" between client & server

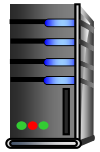
- no handshaking before sending data
- sender explicitly attaches IP destination address and port # to each packet
- receiver extracts sender IP address and port# from received packet

UDP: transmitted data may be lost or received out-of-order

Application viewpoint:

- UDP provides *unreliable* transfer of groups of bytes ("datagrams") between client and server

Client/server socket interaction: UDP



server (running on serverIP)

client

server:

create socket, port= x:
serverSocket =
socket(AF_INET,SOCK_DGRAM)

read datagram from
serverSocket

write reply to
serverSocket
specifying
client address,
port number

client:

create socket:
clientSocket =
socket(AF_INET,SOCK_DGRAM)

Create datagram with server IP and
port=x; send datagram via
clientSocket

read datagram from
clientSocket

close
clientSocket

Example app: UDP server

Python UDPServer

include Python's socket library

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(("", serverPort))
print("The server is ready to receive")
while True:
    message, clientAddress = serverSocket.recvfrom(2048)
    modifiedMessage = message.decode().upper()
    serverSocket.sendto(modifiedMessage.encode(), clientAddress)
```

UDP

create UDP socket

bind socket to local port number 12000

loop forever

Read from UDP socket into message, getting client's address (client IP and port)

send upper case string back to this client

Example app: UDP client

Python UDPClient

The destination

create UDP socket for client

get user keyboard input

Attach server name, port to message; send into socket

read reply characters from socket into string

print out received string and close socket

```
from socket import *
serverName = '192.168.0.20'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_DGRAM)
message = input('Input lowercase sentence:')
clientSocket.sendto(message.encode(), (serverName, serverPort))
modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
print(modifiedMessage.decode())
clientSocket.close()
```

Socket programming with TCP

Client must contact server

- server process must first be running
- server must have created socket (door) that welcomes client's contact

Client contacts server by:

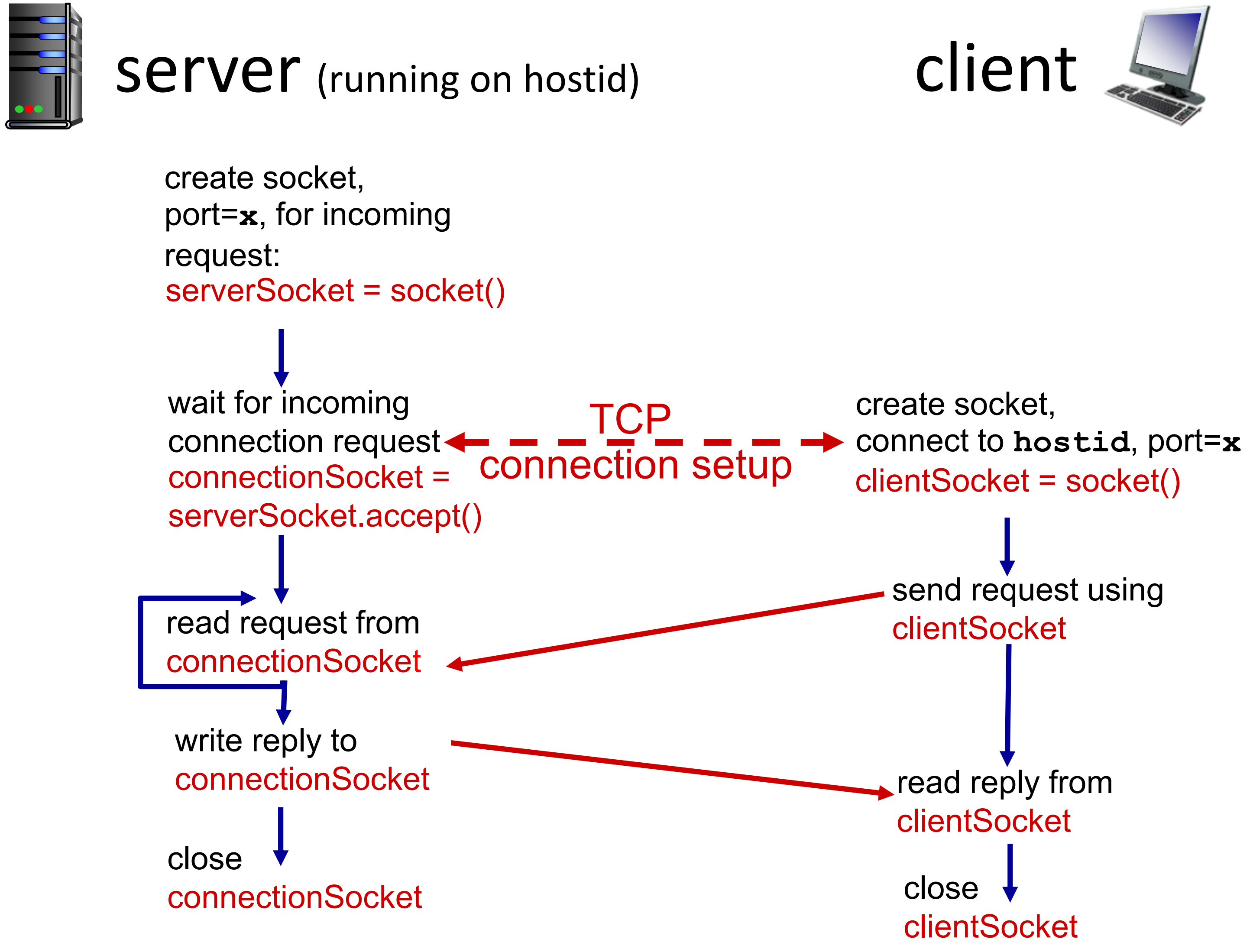
- Creating TCP socket, specifying IP address, port number of server process
- *when client creates socket:* client TCP establishes connection to server TCP

- when contacted by client, *server TCP creates new socket* for server process to communicate with that particular client
  - allows server to talk with multiple clients
  - source port numbers used to distinguish clients (more in Chap 3)

Application viewpoint

TCP provides reliable, in-order byte-stream transfer ("pipe") between client and server

Client/server socket interaction: TCP



Example app: TCP server

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind(("", serverPort))
serverSocket.listen(1)
print("The server is ready to receive")
while True:
    connectionSocket, addr = serverSocket.accept()
    sentence = connectionSocket.recv(1024).decode()
    capitalizedSentence = sentence.upper()
    connectionSocket.send(capitalizedSentence.encode())
    connectionSocket.close()
```

create TCP welcoming socket

server begins listening for incoming TCP requests

loop forever

server waits on accept() for incoming requests, new socket created on return

read bytes from new socket

Send back message (no need for client address – why?)

close connection to this client (but *not* serverSocket)

Example app: TCP client

create TCP server socket

connect to remote server at port 12000

No need to attach server name, port

Wait to receive from server

print out received string and close connection

```
from socket import *
serverName = '192.168.0.20'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName, serverPort))
sentence = input('Input a lower case sentence : ')
clientSocket.send(sentence.encode())
modifiedSentence = clientSocket.recv(1024)
print('From Server : ' + modifiedSentence.decode())
clientSocket.close()
```

Chapter 2: Summary

our study of network application layer is now complete!

- application architectures
  - client-server
  - P2P
- application service requirements:
  - reliability, bandwidth, delay
- Internet transport service model
  - connection-oriented, reliable: TCP
  - unreliable, datagrams: UDP
- specific protocols:
  - HTTP
  - SMTP, IMAP
  - DNS
  - ~~P2P: BitTorrent~~
- ~~video streaming, CDNs~~
- socket programming: TCP, UDP sockets

Week4 Lab – Socket Programming

- Preparation – complete these before entering the lab!
  - Install Python and PyCharm CE in your laptop
  - Test run TCP/UDP Client/Server programs in your laptop
    - Python codes available in week4 lab sheet, in Appendix
    - Follow setup and running procedures described in lab sheet.
- Lab Tasks
  - Complete lab tasks in your lab with your group
  - client and server communicating via socket programming
- You may make a start with Project1 if you finish lab early.

Quiz - 20%

- **Time**: 12:00-23:59, Friday 15th March. **Duration**: 90 minutes
  - must submit within the quiz window (before 23:59)
  - If your Internet is down during quiz, switch to mobile phone hotspot to continue
- **Format**: online open book.
  - **Access:** “Quiz” in Assignment.
  - Answer questions one at a time, cannot go back to previous questions.
  - Please answer questions in specified format - **strictly!**
- **Preparation**:
  - **Scope:** week 1-4 lecture notes, textbook, review questions, tutorial problems, labs.
  - “Quiz_Practice” in Week4 module, Canvas “Discussions” board
  - Consultation @NetFun MS Teams, 2:00pm Thursday 14th March.

Student Feedback Survey (SFS)

- **Your voice on Teaching and Learning:**
  - lectures, learning materials: helpful?
  - assignment, tutorial / lab / UPass: adding value?
- **Please Complete the survey**
  - at **www.sfs.uts.edu.au**
  - if you are happy: what’s going well?
  - any issues: what can be improved?

- **Your feedback appreciated!**

Lecture done

- Q & A