CS 330: Network Applications & Protocols

Application Layer: P2P, Sockets

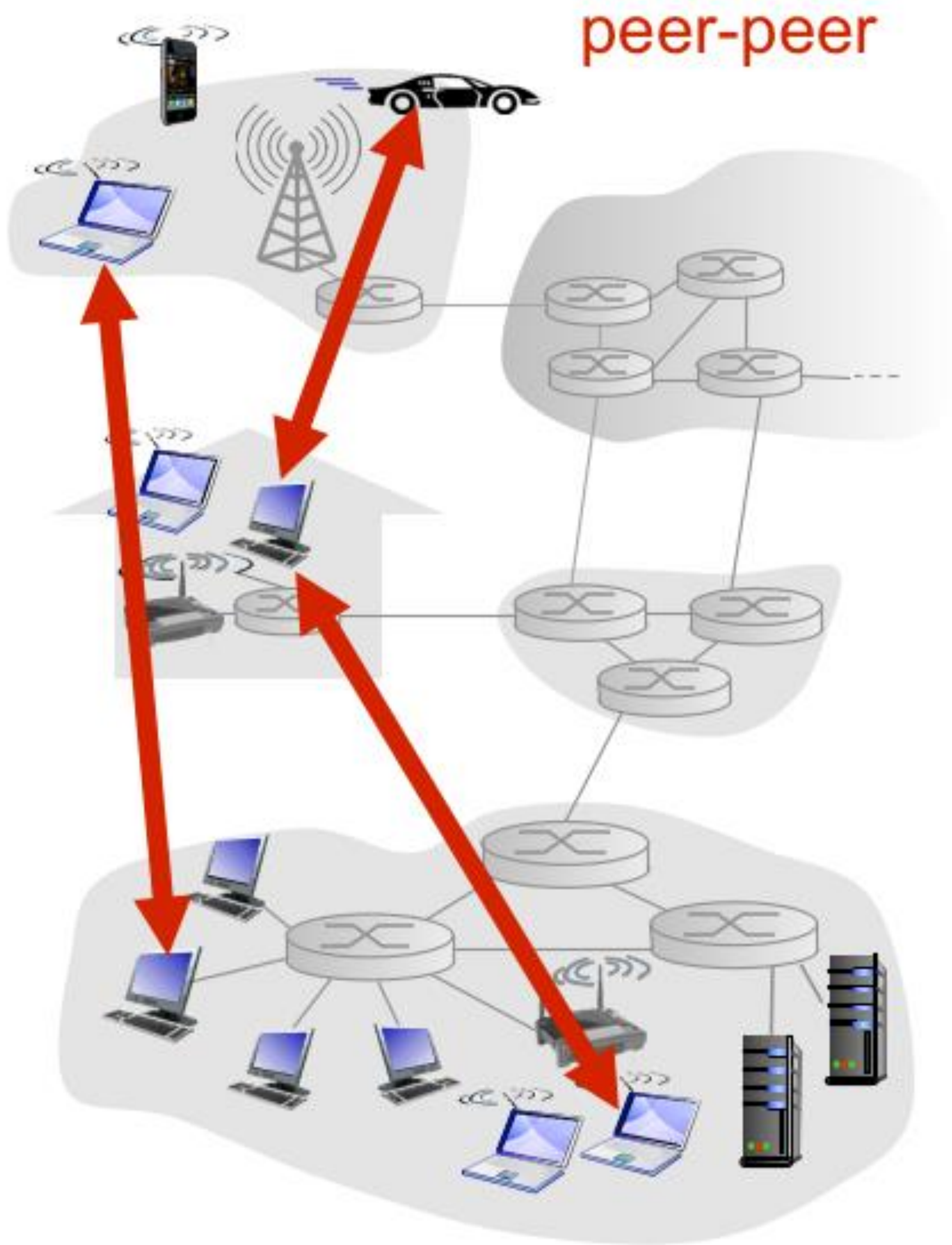
Galin Zhelezov
Department of Physical Sciences
York College of Pennsylvania

Overview of Application Layer

- **Network Application Architectures**
- **HyperText Transfer Protocol (HTTP)**
- **File Transfer and Email protocols (FTP, SMTP)**
- **Domain Name System (DNS)**
- **Peer-to-Peer Applications (P2P)**
  - File Distribution
- **Video Streaming and Content Distribution Networks**
- **Socket Programming**

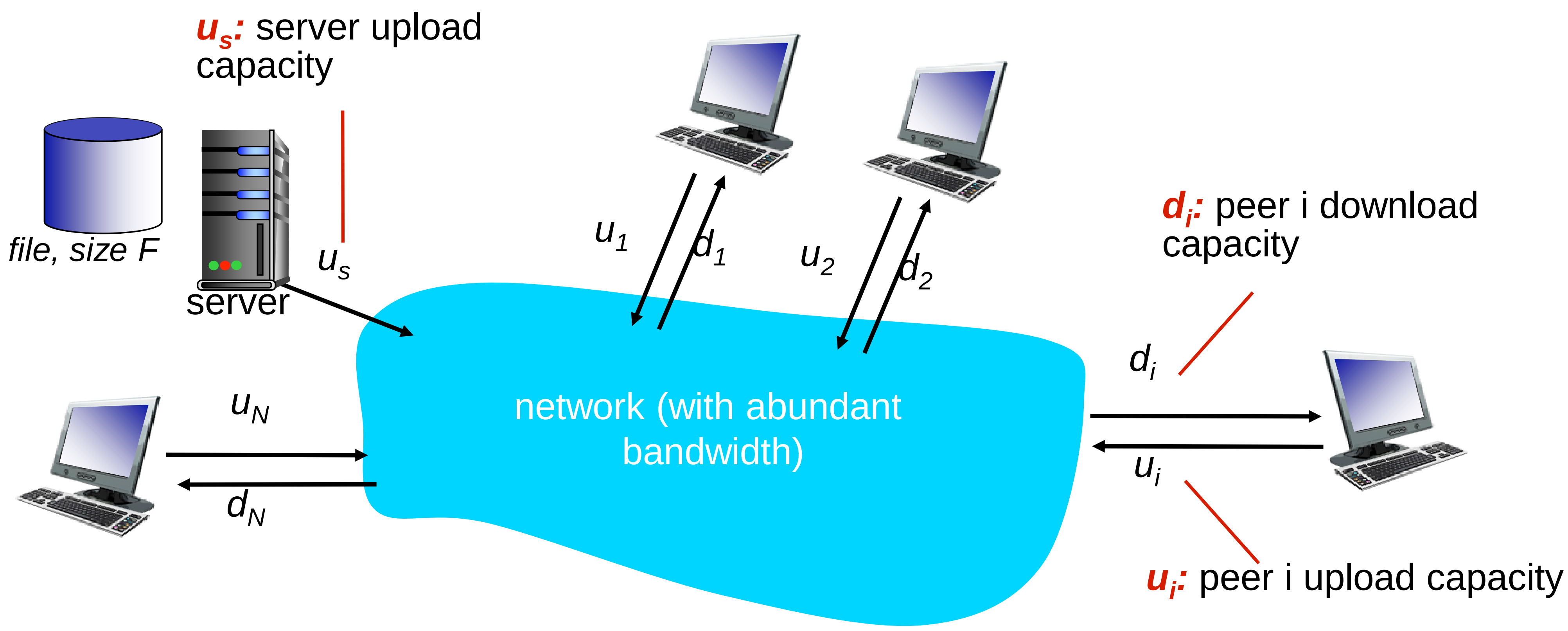
Peer-to-Peer Architecture

- Does not require always-on servers
- Hosts communicate directly with each other
- Peers request service from other peers, and provide service in return to other peers
- Highly scalable
- Self scalability – new peers bring new service capacity, as well as new service demands
- Hosts are intermittently connected and may change IP addresses
- Difficult to authenticate - possibly insecure
- Hosts need incentive to share data



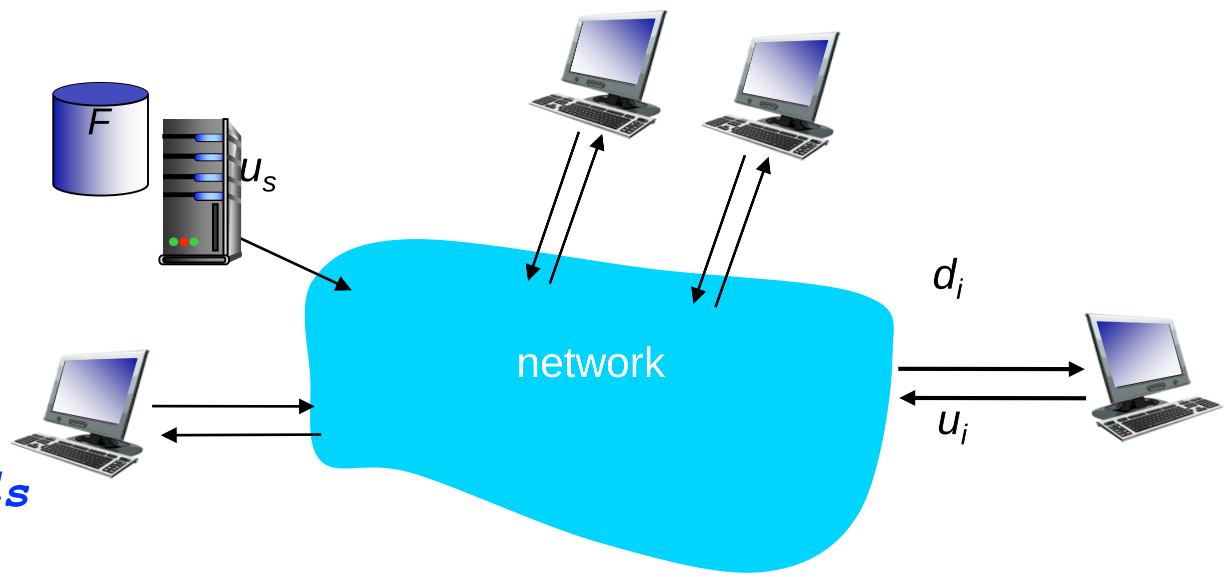
File Distribution: Client-Server vs P2P

- **Question: How much time does it take to distribute a file of size *F* from one server to *N* peers?**
  - Assume the core network has abundant bandwidth
  - Assume clients are not using bandwidth for other tasks



File Distribution: Client-Server

- **Server transmission: must sequentially upload *N* copies of the file:**
  - Time to send one copy: F/u_s
  - Time to send N copies: $(N * F)/u_s$



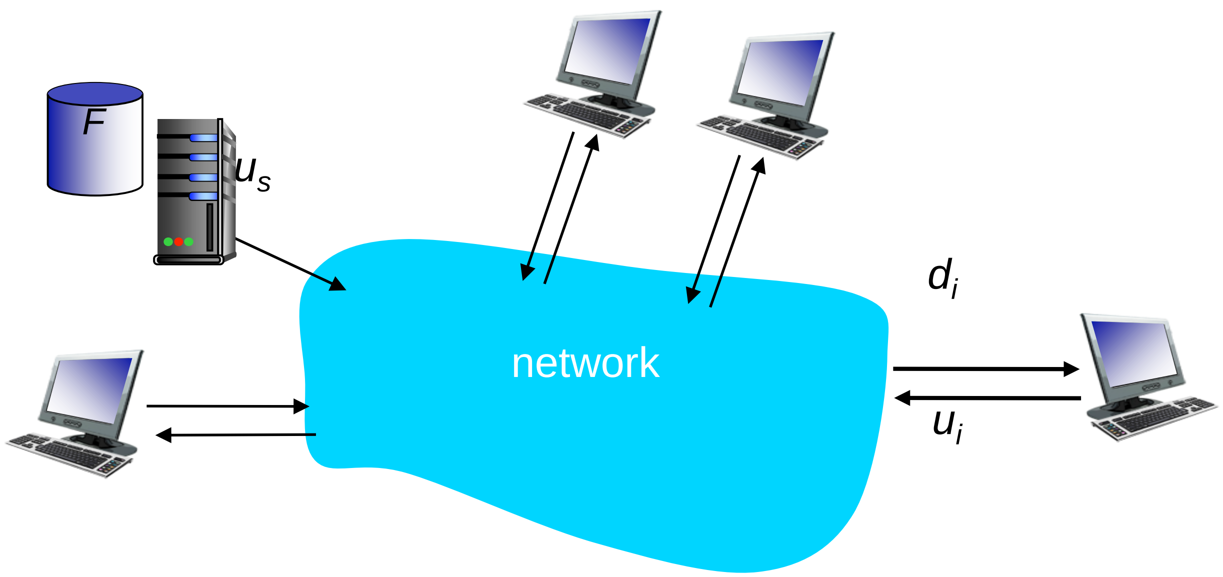
- **Client: each client must download copy of the file**
  - d_{min} = download rate of slowest client
  - Minimum download time for clients: F/d_{min}

Time to distribute file of size F to N clients using client-server approach $\Rightarrow$ $D_{cs} \geq \max\left\{\frac{NF}{u_s}, \frac{F}{d_{min}}\right\}$

**Distribution time increases linearly with the number of clients!

File Distribution: P2P

- **Server transmission: must upload at least one copy of the file**
  - Time to send one copy: F/u_s



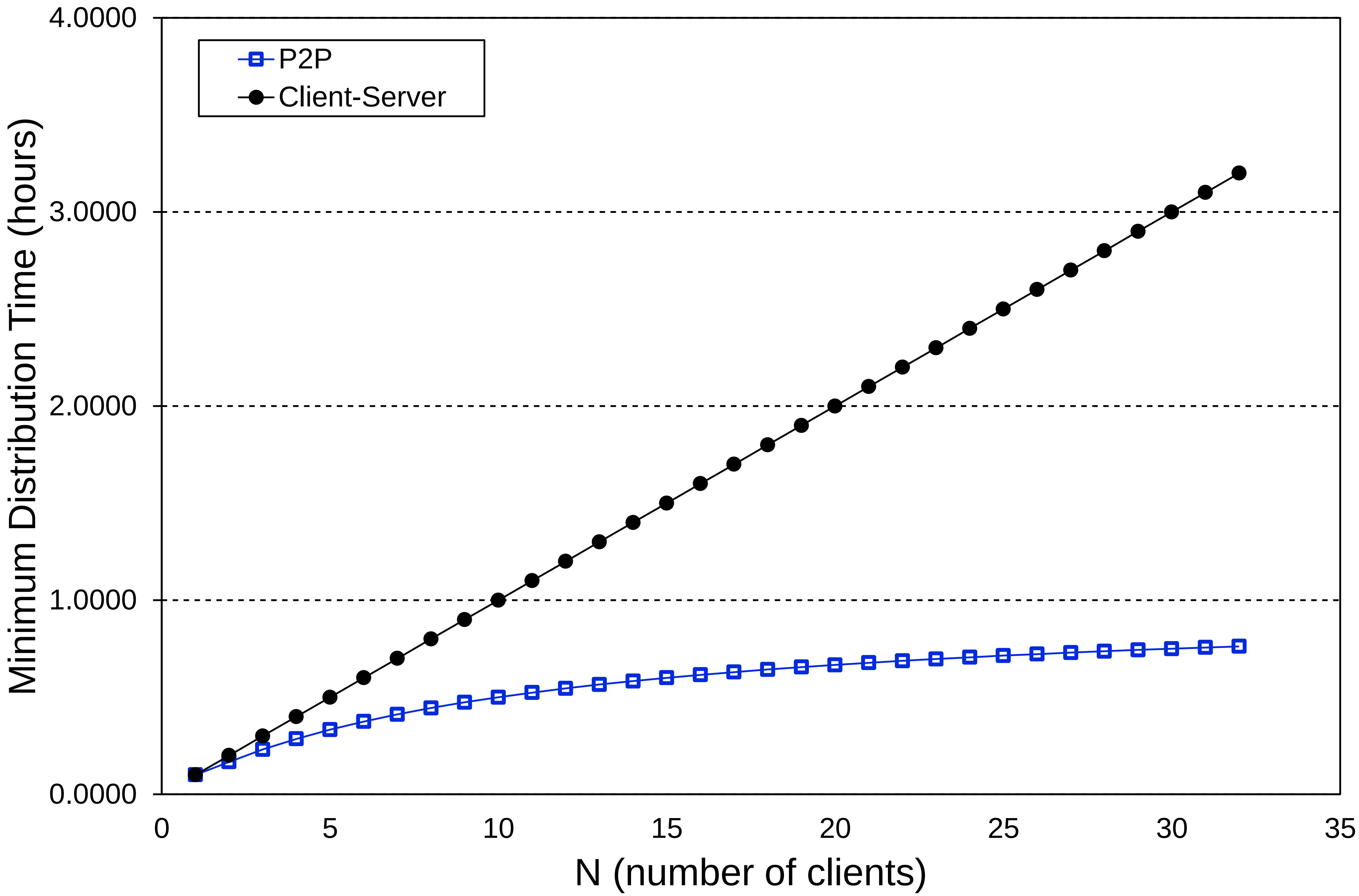
- **Client: each client must download a copy of the file**
  - Minimum download time for clients: F/d_{min}
- **Clients: as an aggregate must download *NF* bits**
  - Total upload capacity is upload rate of server plus upload rate of each of the individual peers: $u_s + \Sigma u_i$

Time to distribute file of size F to N clients using P2P approach $\Rightarrow$ $$D_{P2P} \geq \max\left\{ \frac{F}{u_s}, \frac{F}{d_{min}}, \frac{NF}{u_s + \sum_{i=1}^{N} u_i} \right\}$$

**As *N* increases, so does service capacity

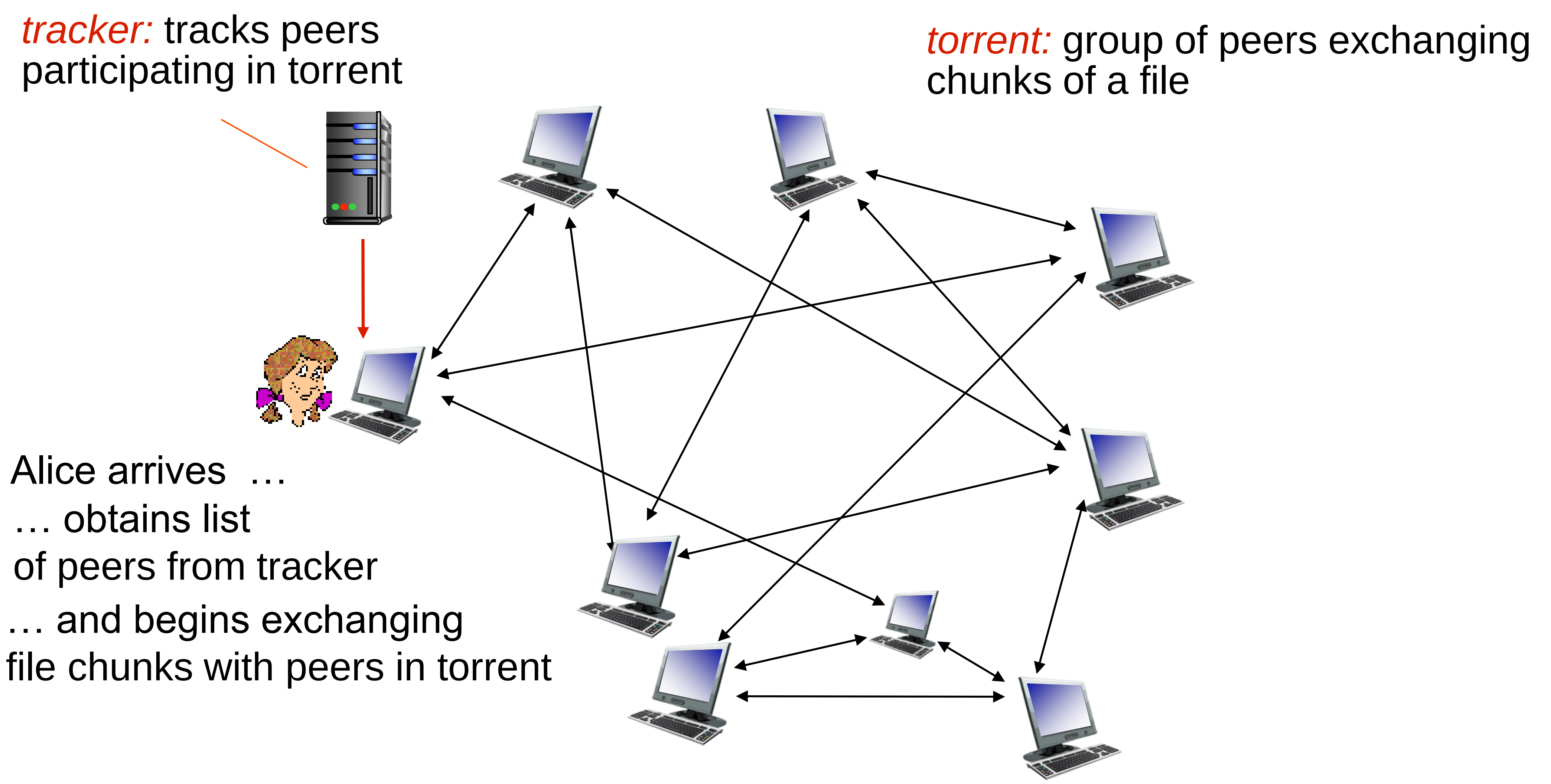
Client-Server vs. P2P

Client upload rate = u, F/u = 1 hour, u_s = 10u, $d_{min} \geq u_s$



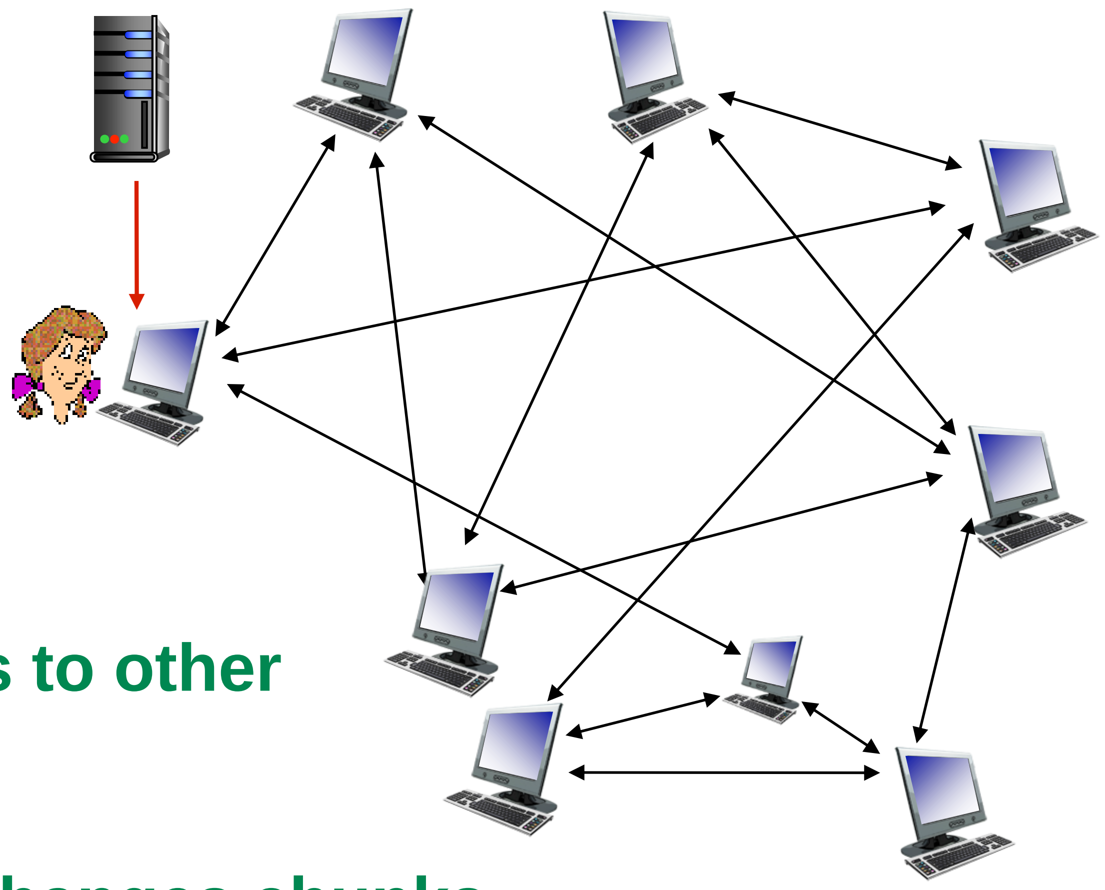
P2P File Distribution: BitTorrent

- **File divided into 256 KByte chunks**
- **Peers in torrent send/receive file chunks**



P2P File Distribution: BitTorrent

- **Peer joining torrent:**
  - Has no chunks, but will accumulate them over time from other peers
  - Registers with tracker to get list of peers, connects to subset of peers ("neighbors")



- **While downloading, peer uploads chunks to other peers**
- **Peer may change peers with whom it exchanges chunks**
- **Churn: peers may come and go**
- **Once peer has entire file, it may (selfishly) leave or (altruistically) remain in torrent**

BitTorrent: Requesting/Sending File Chunks

- **Requesting chunks:**
  - At any given time, different peers have different subsets of file chunks
  - Periodically, Alice asks each peer for a list of chunks that they have
  - Alice requests missing chunks from peers, rarest first

- **Sending chunks:**

Sending chunks: tit-for-tat

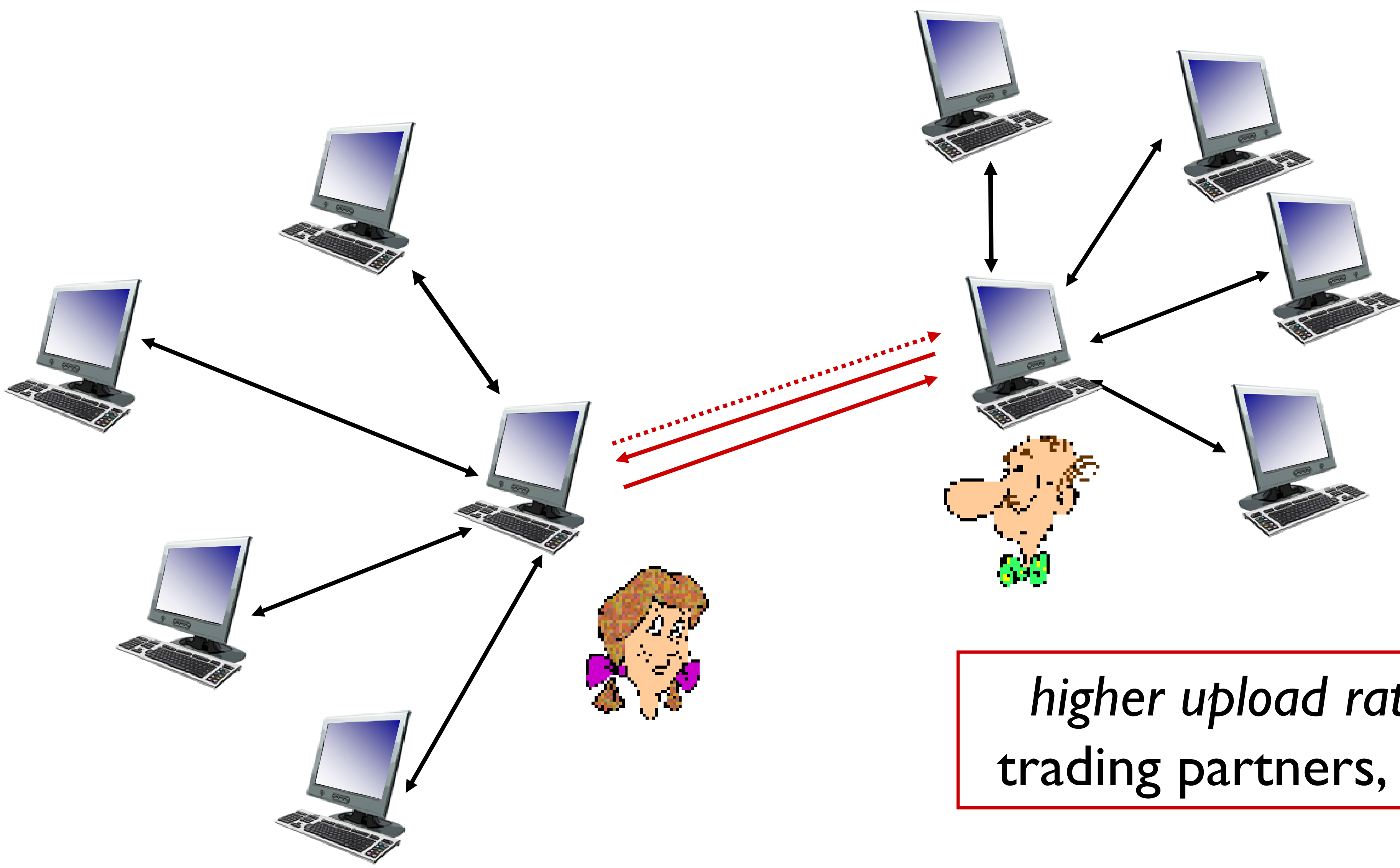
- Alice sends chunks to the four peers currently sending her chunks at highest rate
  - Other peers are choked by Alice (do not receive chunks from her)
  - Re-evaluate top 4 every 10 secs
- Every 30 secs, randomly select another peer, start sending chunks
  - Optimistically unchoke this peer
  - Newly chosen peer may join top four

BitTorrent: tit-for-tat

(1) Alice "optimistically unchokes" Bob

(2) Alice becomes one of Bob's top-four providers; Bob reciprocates

(3) Bob becomes one of Alice's top-four providers



higher upload rate: find better trading partners, get file faster !

Overview of Application Layer

- **Network Application Architectures**
- **HyperText Transfer Protocol (HTTP)**
- **File Transfer and Email protocols (FTP, SMTP)**
- **Domain Name System (DNS)**
- **Peer-to-Peer Applications (P2P)**
- **Video Streaming and Content Distribution Networks**
- **Socket Programming**

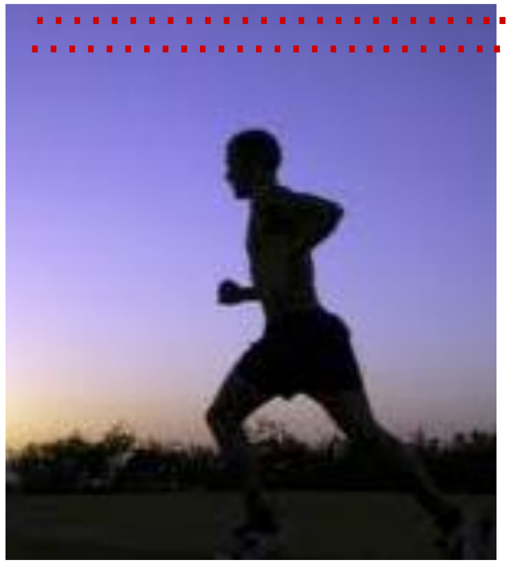
Video Streaming and CDNs: context

- video traffic: major consumer of Internet bandwidth
  - Netflix, YouTube: 37%, 16% of downstream residential ISP traffic
  - ~1B YouTube users, ~75M Netflix users
- challenge: scale - how to reach ~1B users?
  - single mega-video server won't work (why?)
- challenge: heterogeneity
  - different users have different capabilities (e.g., wired versus mobile; bandwidth rich versus bandwidth poor)
- *solution:* distributed, application-level infrastructure

Multimedia: video

- **video: sequence of images displayed at constant rate**
  - e.g., 24 images/sec
- **digital image: array of pixels**
  - each pixel represented by bits
- **coding: use redundancy *within* and *between* images to decrease # bits used to encode image**
  - spatial (within image)
  - temporal (from one image to next)

spatial coding example: instead of sending *N* values of same color (all purple), send only two values: color value (*purple*) *and number of repeated values* (N)



frame *i*

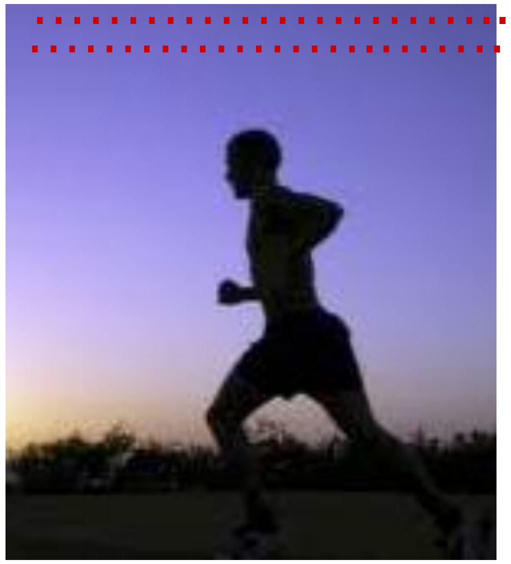
temporal coding example: instead of sending complete frame at i+1, send only differences from frame i

frame *i+1*

Multimedia: video

- CBR: (constant bit rate): video encoding rate fixed
- VBR: (variable bit rate): video encoding rate changes as amount of spatial, temporal coding changes
- examples:
  - MPEG 1 (CD-ROM) 1.5 Mbps
  - MPEG2 (DVD) 3-6 Mbps
  - MPEG4 (often used in Internet, < 1 Mbps)

spatial coding example: instead of sending *N* values of same color (all purple), send only two values: color value (*purple*) *and number of repeated values* (N)



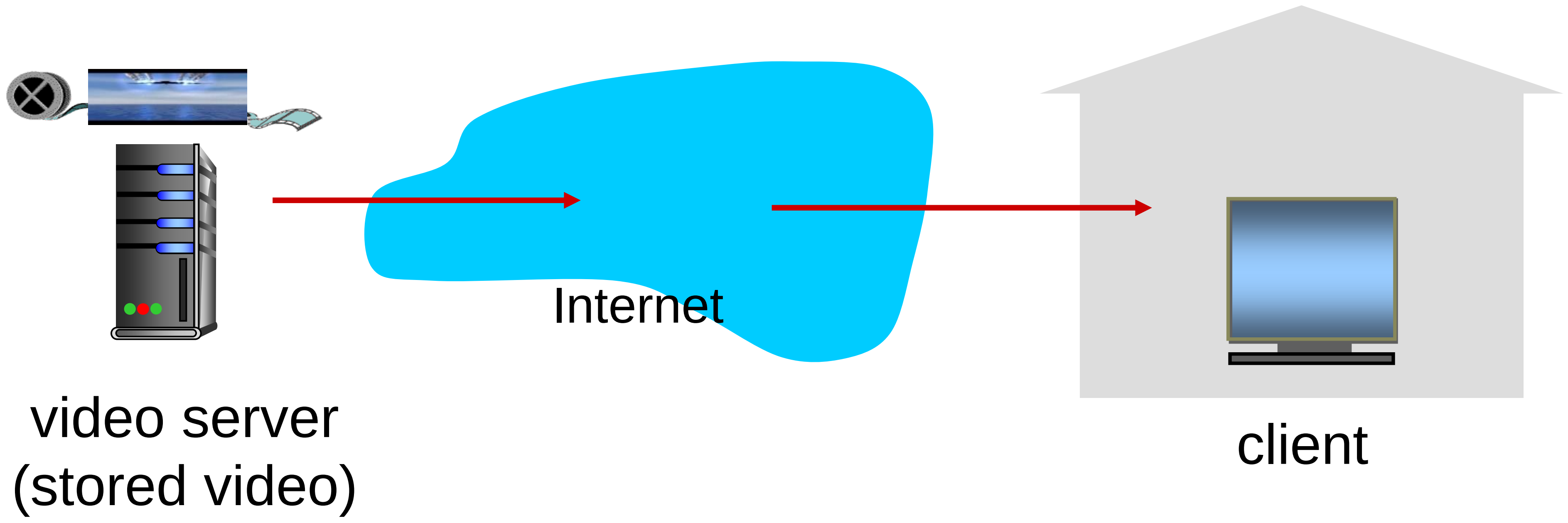
frame *i*

temporal coding example: instead of sending complete frame at i+1, send only differences from frame i

frame *i+1*

Streaming stored video:

simple scenario:



Streaming multimedia: DASH

- ***DASH: Dynamic, Adaptive Streaming over HTTP***
- ***server:***
  - divides video file into multiple chunks
  - each chunk stored, encoded at different rates
  - *manifest file:* provides URLs for different chunks
- ***client:***
  - periodically measures server-to-client bandwidth
  - consulting manifest, requests one chunk at a time
    - chooses maximum coding rate sustainable given current bandwidth
    - can choose different coding rates at different points in time (depending on available bandwidth at time)

Streaming multimedia: DASH

- ***DASH: D**ynamic, **A**daptive **S**treaming over **H**TTP*
- ***"intelligence"* at client: client determines**
  - *when* to request chunk (so that buffer starvation, or overflow does not occur)
  - *what encoding rate* to request (higher quality when more bandwidth available)
  - *where* to request chunk (can request from URL server that is "close" to client or has high available bandwidth)

Content distribution networks

- ***challenge:*** **how to stream content (selected from millions of videos) to hundreds of thousands of *simultaneous* users?**
- ***option 1:*** **single, large "mega-server"**
  - single point of failure
  - point of network congestion
  - long path to distant clients
  - multiple copies of video sent over outgoing link

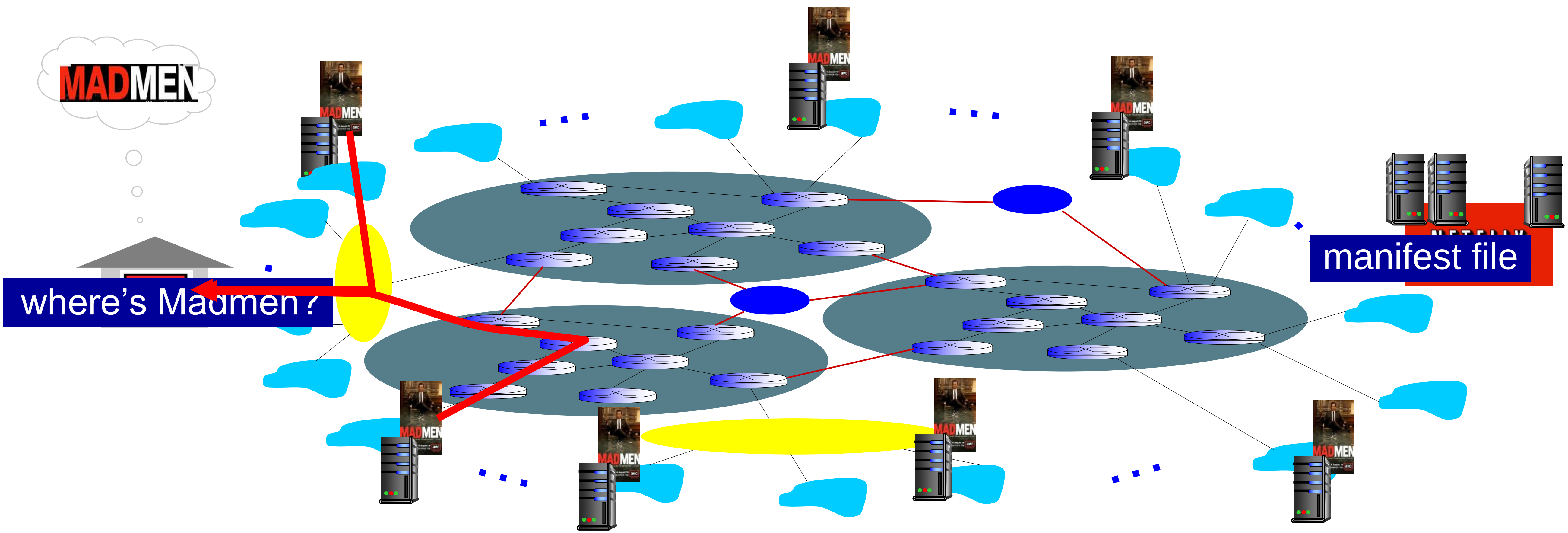
….quite simply: this solution *doesn't scale*

Content distribution networks

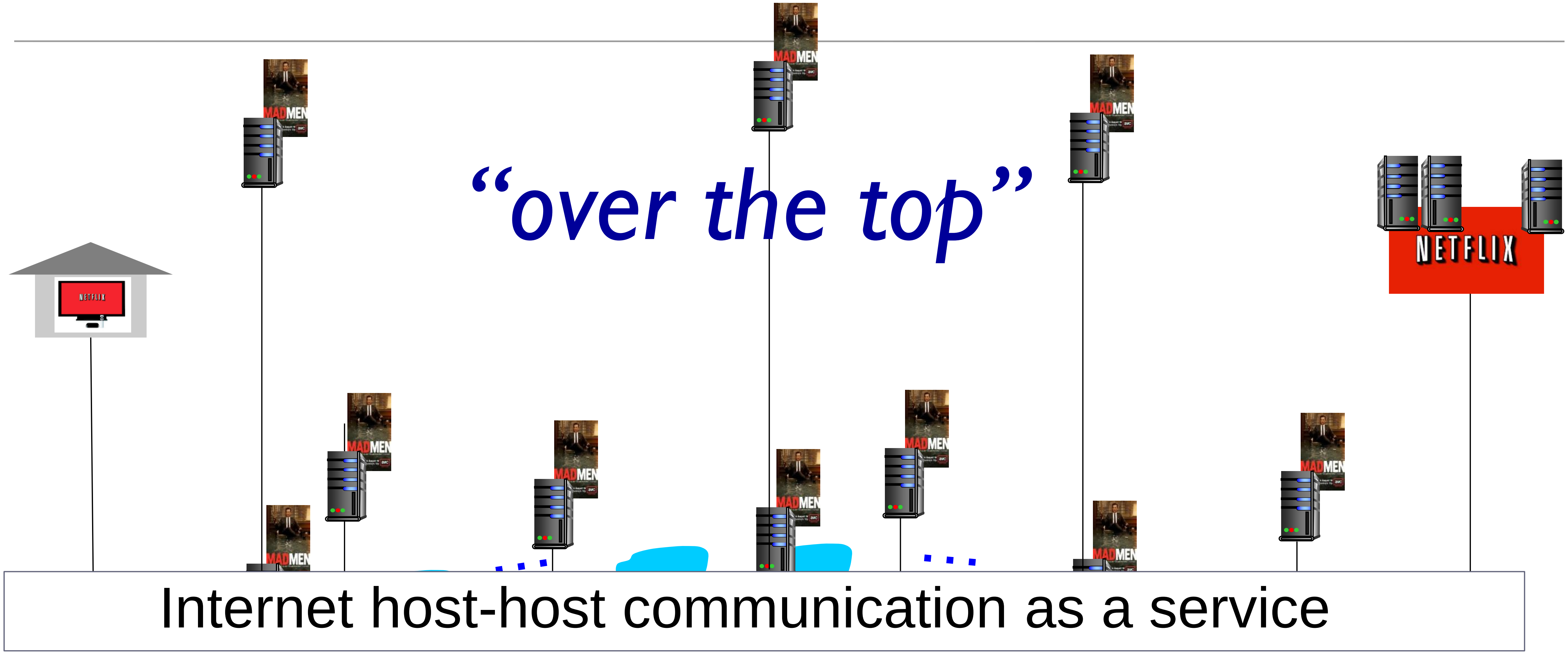
- ***challenge:*** **how to stream content (selected from millions of videos) to hundreds of thousands of simultaneous users?**
- ***option 2:*** **store/serve multiple copies of videos at multiple geographically distributed sites *(CDN)***
  - *enter deep:* push CDN servers deep into many access networks
    - close to users
    - used by Akamai, 1700 locations
  - *bring home:* smaller number (10's) of larger clusters in POPs near (but not within) access networks
    - used by Limelight

Content Distribution Networks (CDNs)

- CDN: stores copies of content at CDN nodes
  - e.g. Netflix stores copies of MadMen
- **subscriber requests content from CDN**
  - directed to nearby copy, retrieves content
  - may choose different copy if network path congested



Content Distribution Networks (CDNs)



OTT challenges: **coping with a congested Internet**

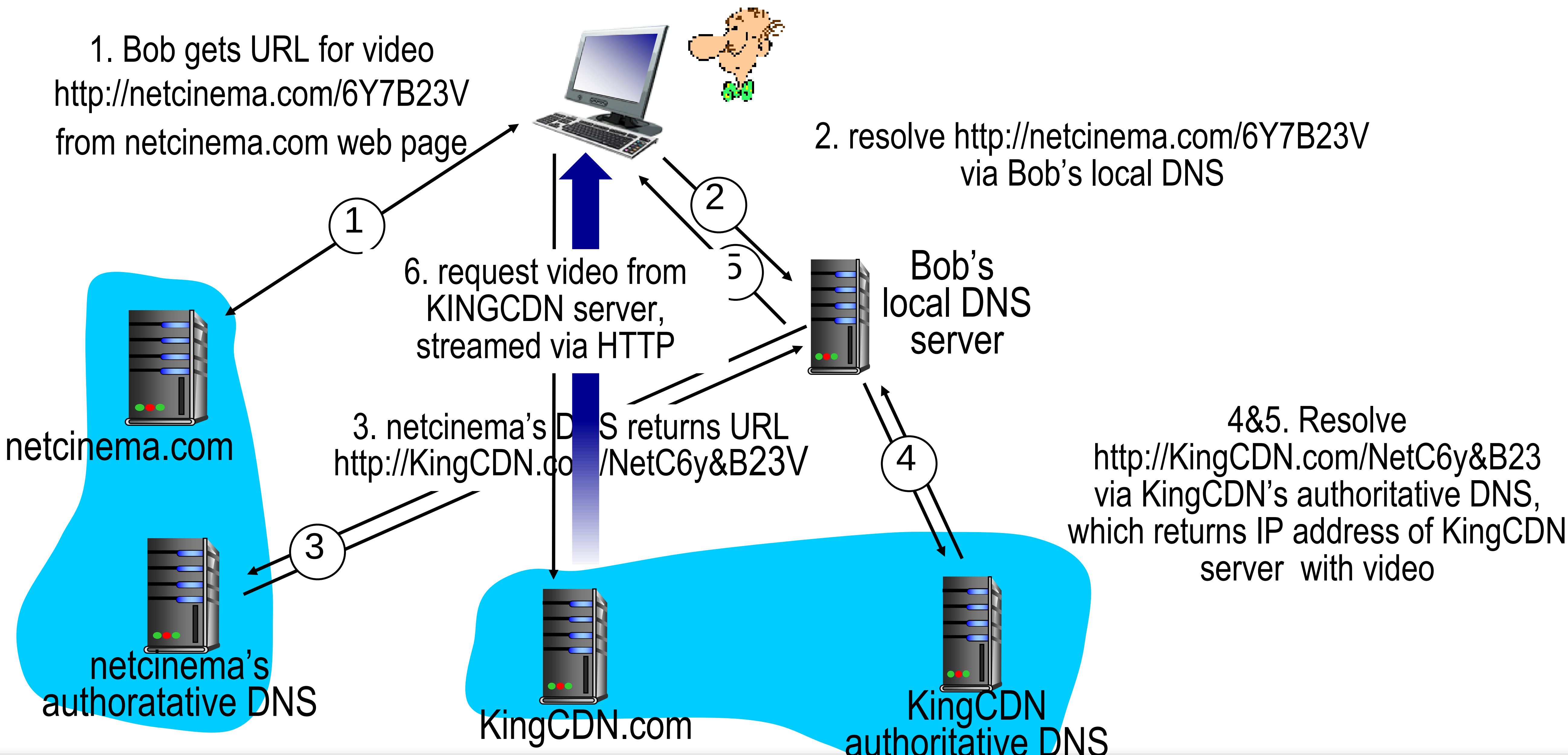
- from which CDN node to retrieve content?
- viewer behavior in presence of congestion?
- what content to place in which CDN node?

more .. in chapter 7

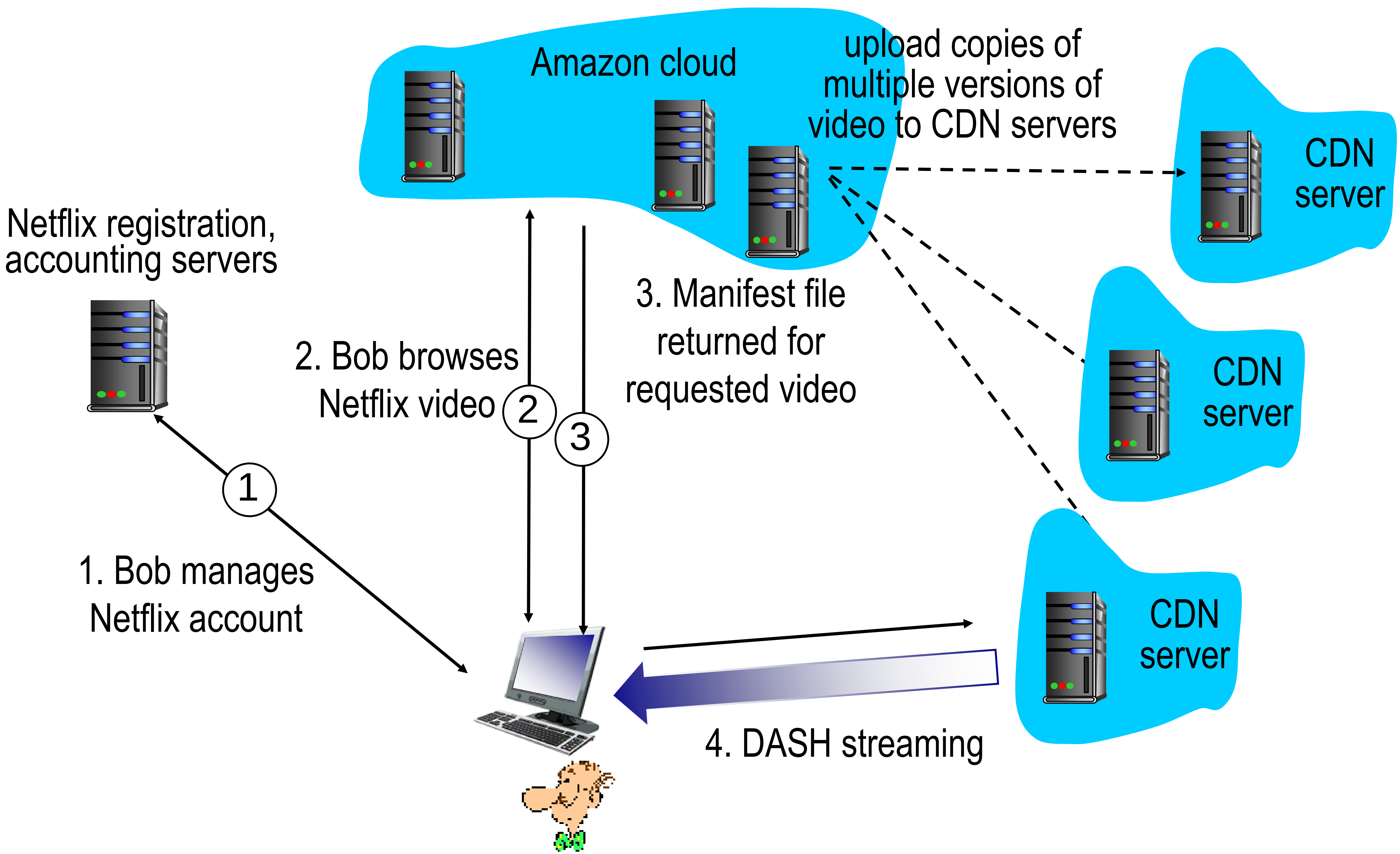
CDN content access: a closer look

Bob (client) requests video http://netcinema.com/6Y7B23V

- video stored in CDN at http://KingCDN.com/NetC6y&B23V



Case study: Netflix

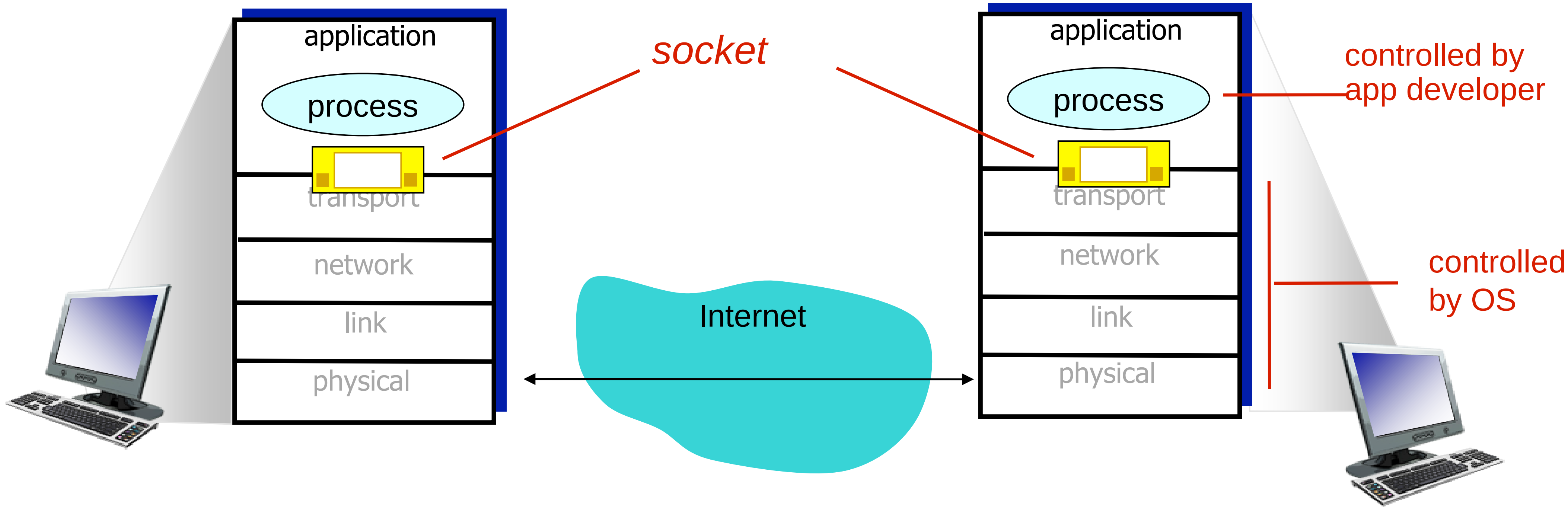


Overview of Application Layer

- **Network Application Architectures**
- **HyperText Transfer Protocol (HTTP)**
- **File Transfer and Email protocols (FTP, SMTP)**
- **Domain Name System (DNS)**
- **Peer-to-Peer Applications (P2P)**
- **Video Streaming and Content Distribution Networks**
- **Socket Programming**

Socket Programming

- **Goal: learn how to build client/server applications that communicate using sockets**
- **Socket: door between application process and end-end-transport protocol**



Socket Programming

- **Two socket types for two transport services:**
  - **UDP**: unreliable datagram
  - **TCP**: reliable, byte stream-oriented
- **Application Example:**

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

  (2) The server receives the data and converts characters to uppercase.

  (3) The server sends the modified data to the client.

  (4) The client receives the modified data and displays the line on its screen.

Socket Programming with UDP

- **UDP: no "connection" between client & server**
  - No handshaking before sending data
  - Sender explicitly attaches IP destination address and port number to each packet
  - Receiver extracts sender IP address and port number from received packet
- **Transmitted data may be lost or received out-of-order**
- **Application viewpoint:**
  - UDP provides unreliable transfer of datagrams between client and server

Socket Programming with TCP

- **Client must contact server**
  - Server process must first be running
  - Server must have created socket that welcomes client's connection
- **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
- **Application viewpoint:**
  - TCP provides reliable, in-order byte-stream transfer (a "pipe") between client and server

Chapter 2: summary

most importantly: learned about protocols!

- **typical request/reply message exchange:**
  - client requests info or service
  - server responds with data, status code
- **message formats:**
  - *headers*: fields giving info about data
  - *data:* info(payload) being communicated

important themes:

- control vs. messages
  - in-band, out-of-band
- centralized vs. decentralized
- stateless vs. stateful
- reliable vs. unreliable message transfer
- "complexity at network edge"