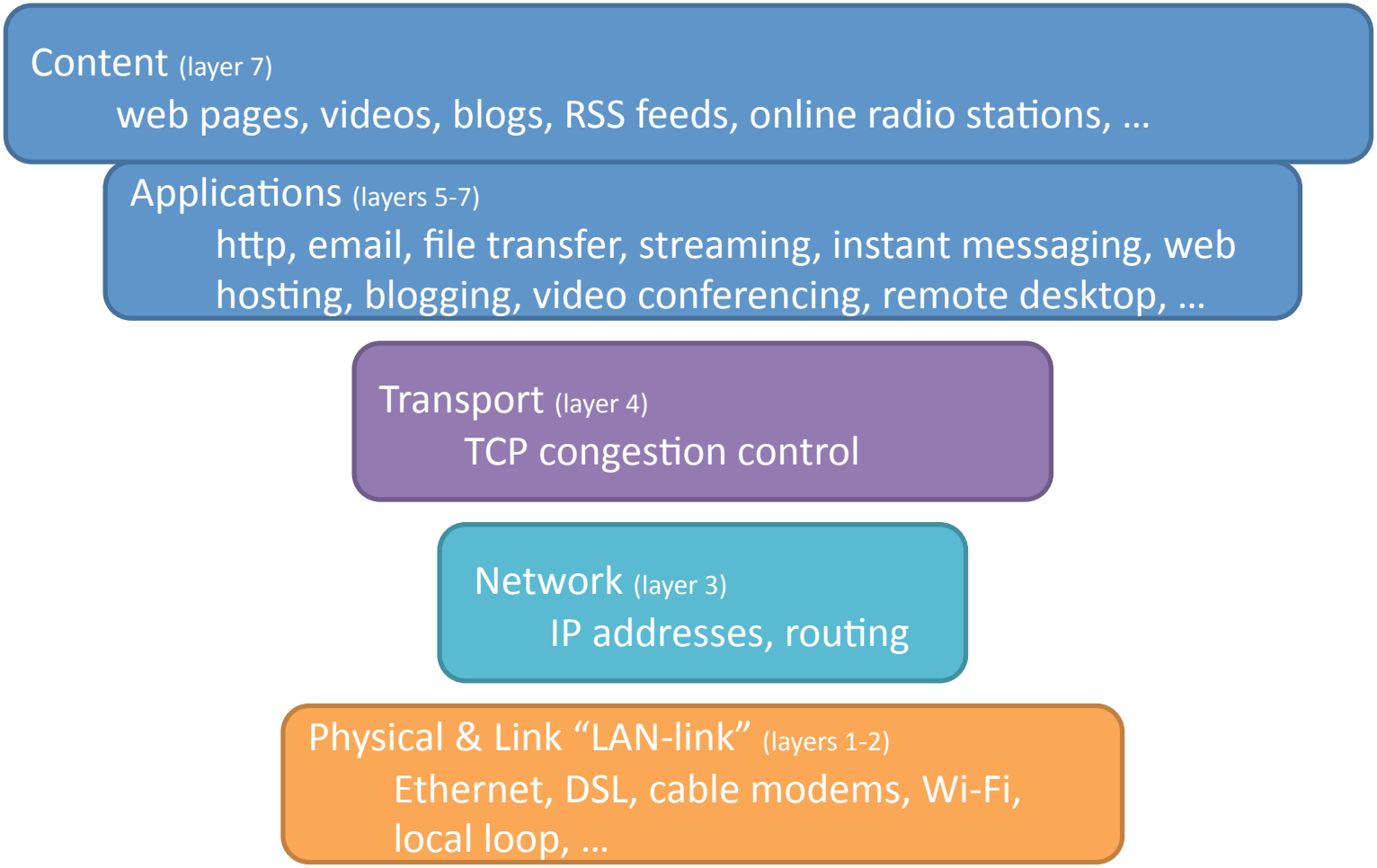
Internet applications

Internet Layering



Application Layer

- Applications
  - Browsers
  - Email programs
  - Media players
  - Etc, etc, etc …
- Protocols that support applications (type of messages, syntax)
  - www: http
  - email: pop, smtp, imap
  - ftp, telnet, …

Internet applications

- Email
- File transfer
- www
- e-commerce
- File sharing
- Streaming
- Gaming
- Voice over IP
- Video over IP
- ……..

Application requirements

- Metrics
  - Packet loss
  - Delay
  - Throughput
- Bounds
  - Max/Min
  - Variations
  - Firm/flexible

Application requirement examples

- File sharing:
  - Loss – not ok
  - Delay – very flexible, often hours
  - Throughput – higher is better, but flexible
- Email:
  - Loss – not ok
  - Delay – few minutes ok, but flexible
  - Throughput – higher is better, but flexible
- http:
  - Loss – not ok
  - Delay – few seconds ok, but flexible
  - Throughput – higher is better, but flexible

Application requirement examples

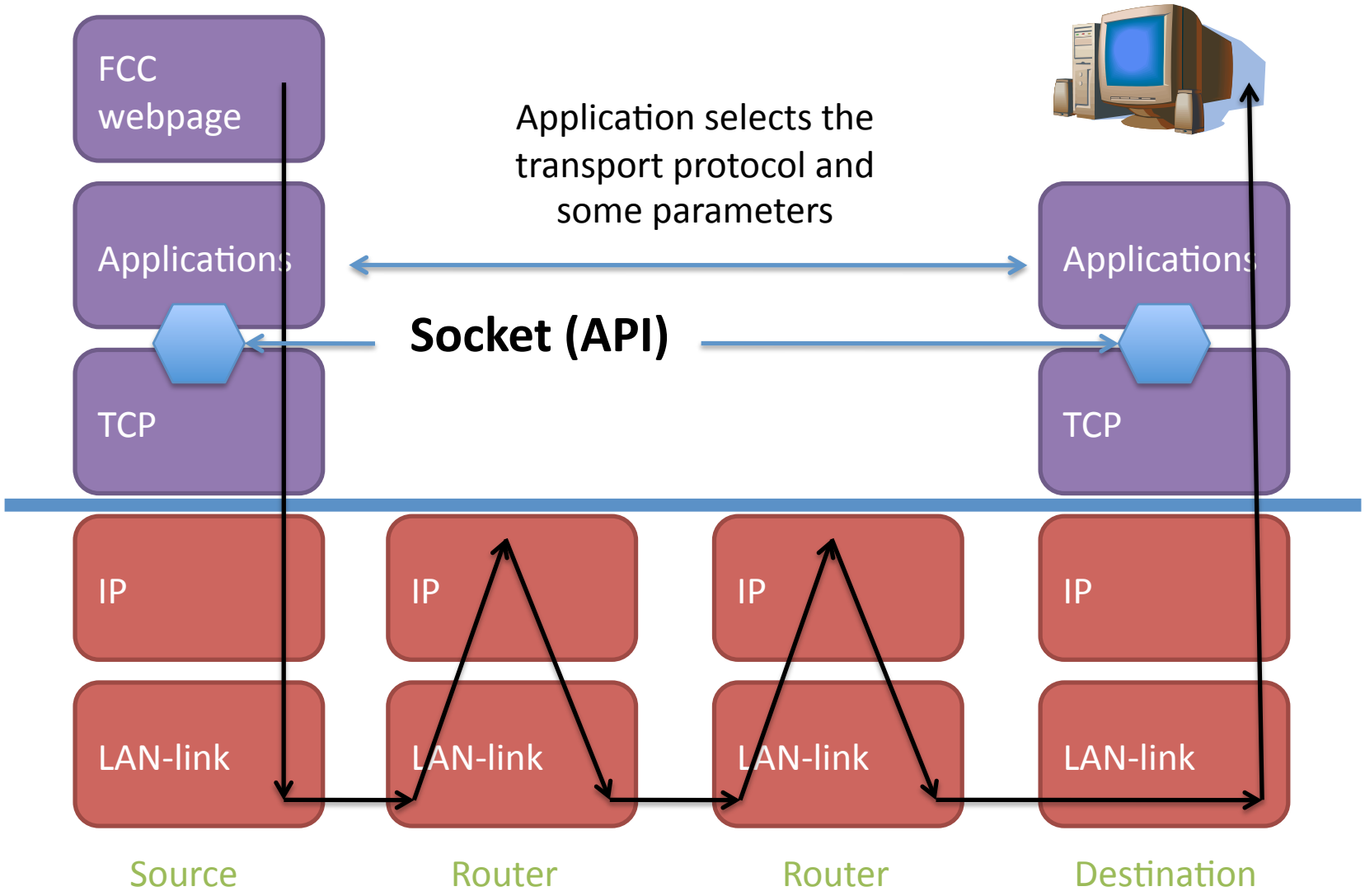
- Streaming:
  - Loss – small amount ok
  - Delay – seconds, firm once stream started
  - Throughput – fixed
- VoIP:
  - Loss – small amount ok
  - Delay – a few tenths of a second
  - Throughput – fixed

QoS characterizations

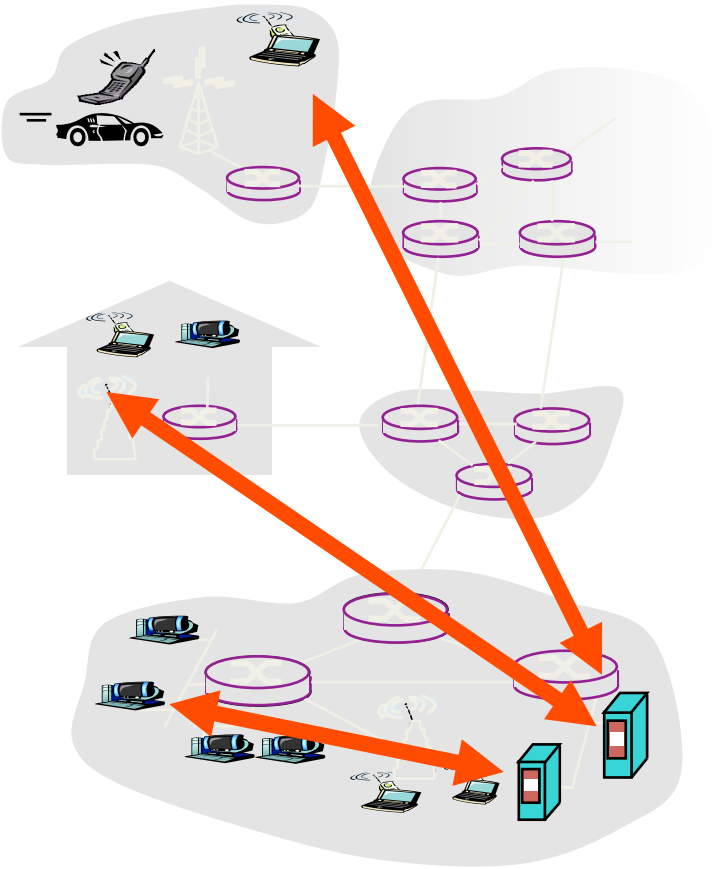
Application	Bandwidth	Delay	Jitter	Loss
Email	Low	Low	Low	Medium
File sharing	High	Low	Low	Medium
Web access	Medium	Medium	Low	Medium
Remote login	Low	Medium	Medium	Medium
Audio on demand	Low	Low	High	Low
Video on demand	High	Low	High	Low
Telephony	Low	High	High	Low
Videoconferencing	High	High	High	Low

Tanenbaum fig. 5-27

Layering + Peering

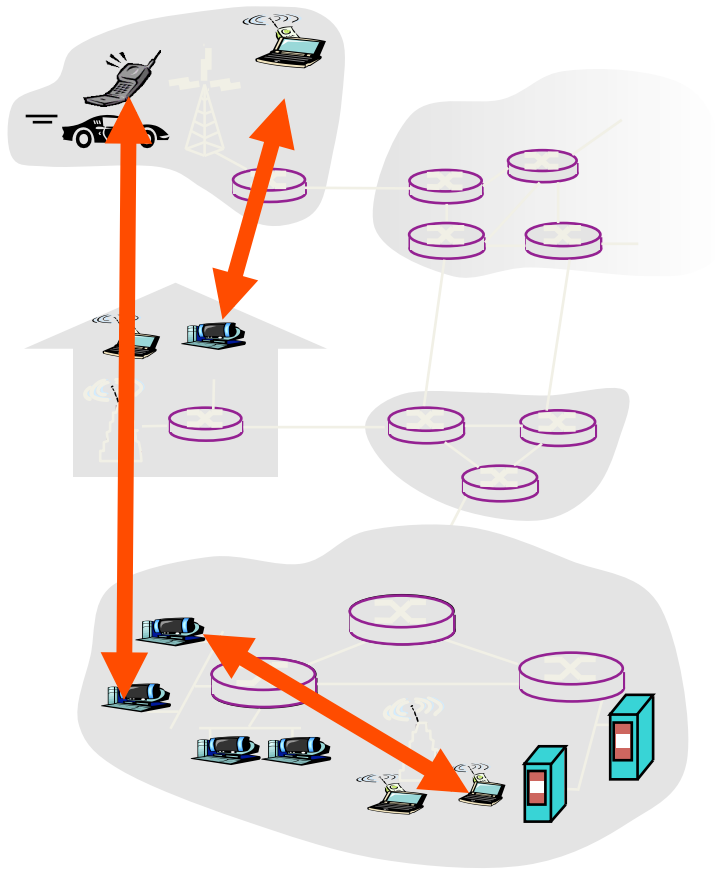


Application Architecture: client-server



- client requests content
  - client must know server name or address
- server replies with content
  - server always on to listen for requests
- client doesn't have to listen for requests

Application Architecture: peer-to-peer



- each machine acts as both a client and a server
- Same task
- as a client:
  - must have some way to find peer acting as a server that is on and has desired content
- as a server:
  - must have some way of advertising availability and content
- Scalable
- Hybrid architectures

HTTP

HyperText Transfer Protocol (HTTP)

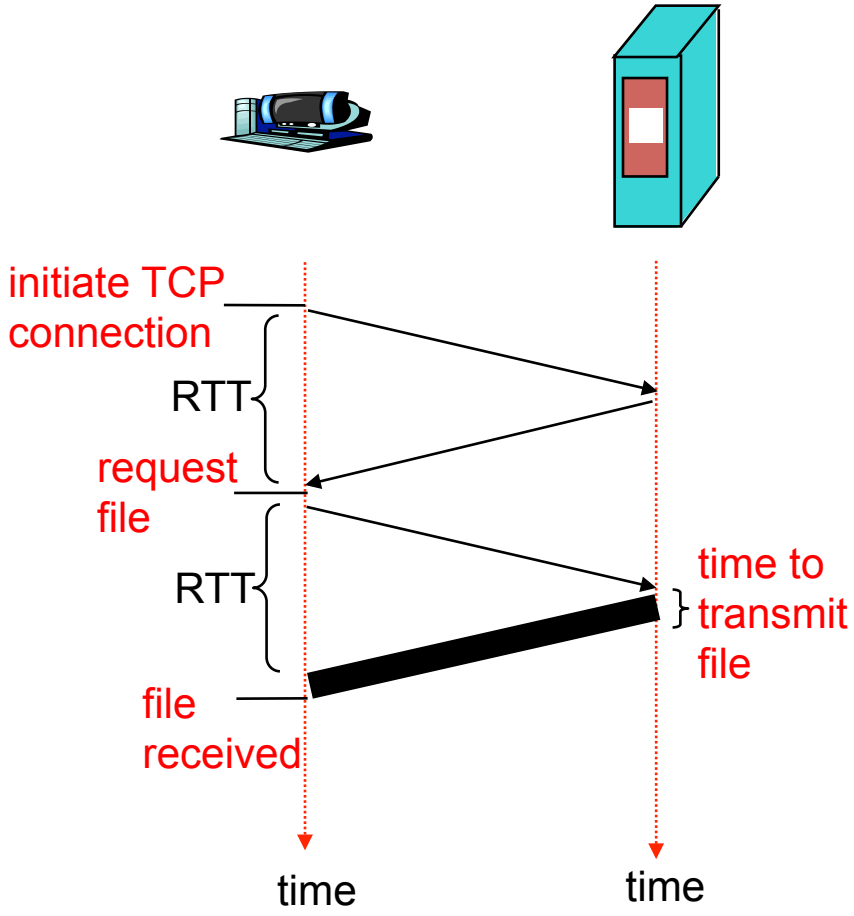
- Purpose: request and transfer a webpage from a server to a user (client-server)
- Service characteristics:
  - Pages requested and sent at random times
  - Connected only for duration of download
- Performance:
  - Loss – not ok
  - Delay – few seconds ok, but flexible
  - Throughput – higher is better, but flexible

http: connection management

- Connection Initiation
  - Well-known ports (e.g. 80)
  - Client (browser) – Server (webserver)
  - Connection access control (limited resources, TCP)
    - Server may block a connection
  - Server balancing
    - Server may redirect a connection

TCP connection

- RTT= round trip time (transmission, propagation, and queuing times for all links in between user and server)
- Number of packets depends on file size
- Timing of packets depends on layers below



Kurose fig. 2.7

http: connection management

- Connection Termination
  - Non-persistent:
    - Each file requires a separate connection, e.g.
      - first request main page, close connection
      - if main page has images:
        - » for each image, open connection, request image, close connection.
    - Workaround: open parallel connections, one for each image (but, server may have limited resources in terms of # of tcp connections).
  - Persistent:
    - Close connection only after client is done with server (or timeout)
    - Serial objects transfer

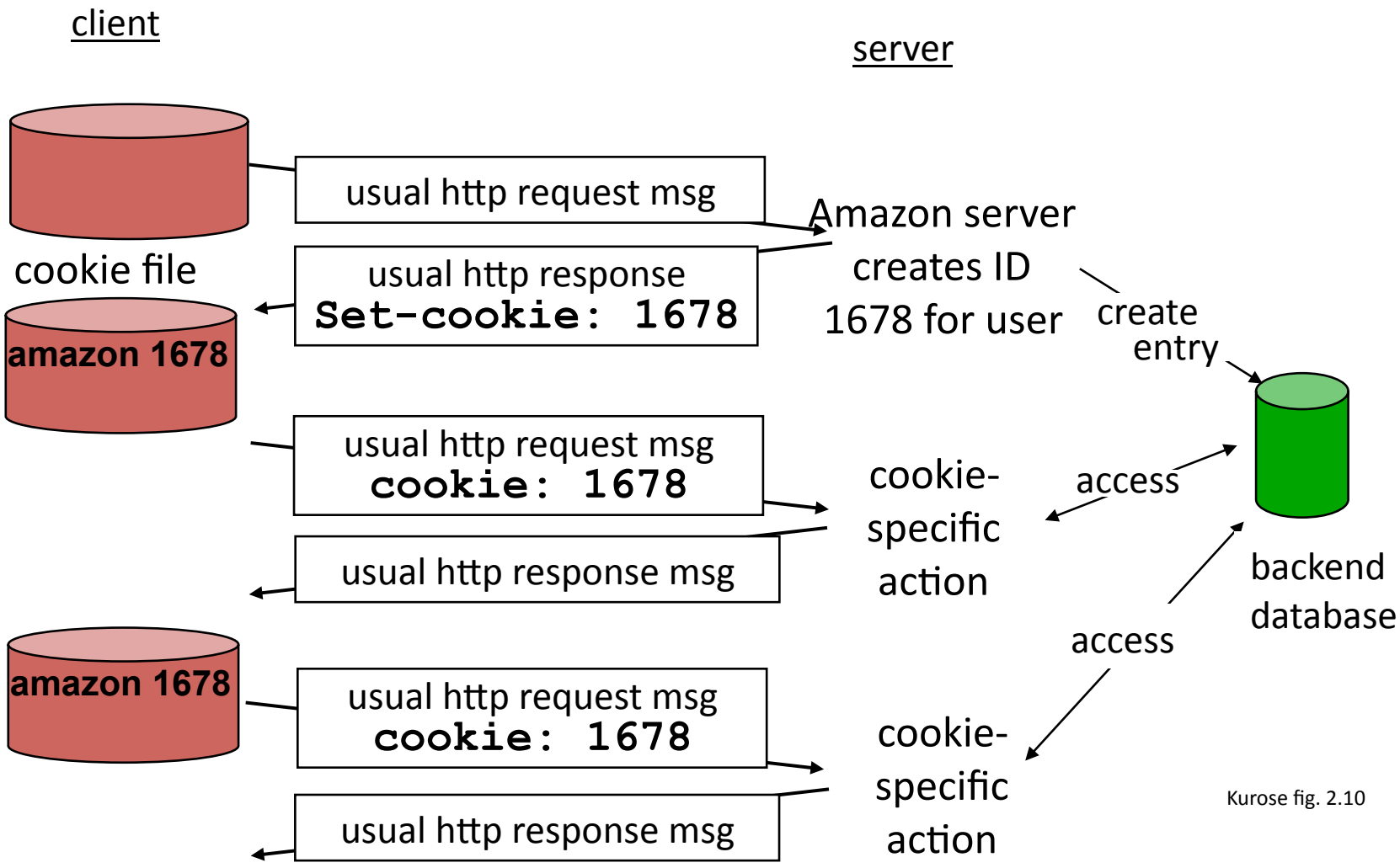
E-commerce over http

- Purpose: purchase something on the www
- Service characteristics:
  - login using id and password
  - select items
  - checkout

E-commerce requires “state”

- “state” = information from previous communication
  - e.g. who you are, your basket, checkout status
- Problem: http doesn’t keep track of state
- solution: client & server application (above http) must jointly keep track of state
  - client: cookie
  - server: database

E-commerce over http



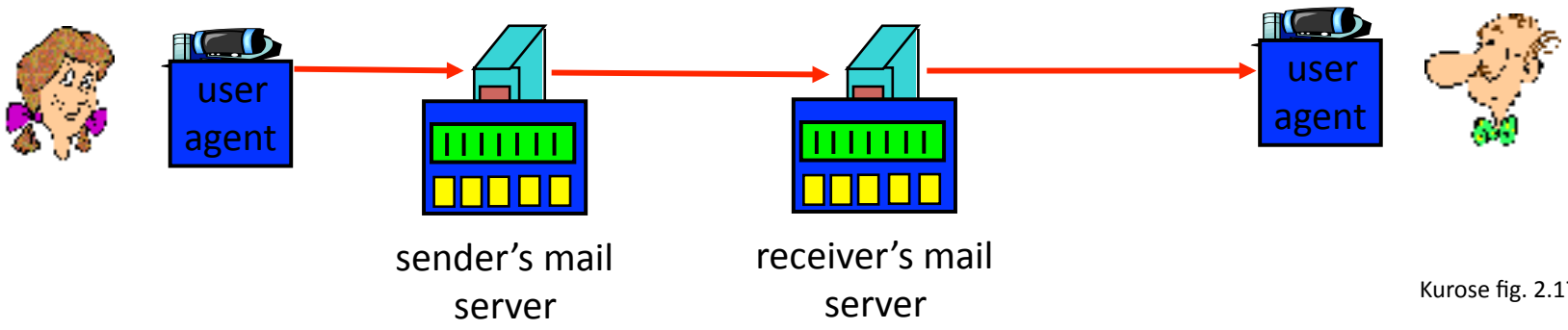
Kurose fig. 2.10

Email

email

- Purpose: transfer a message from one user to another user
- Service characteristics:
  - Messages sent at random times
  - Connected only for duration of upload or download
- Performance:
  - Loss – not ok
  - Delay – few minutes ok, but flexible
  - Throughput – higher is better, but flexible

email: method & connection management



Kurose fig. 2.17

User agent: (client)

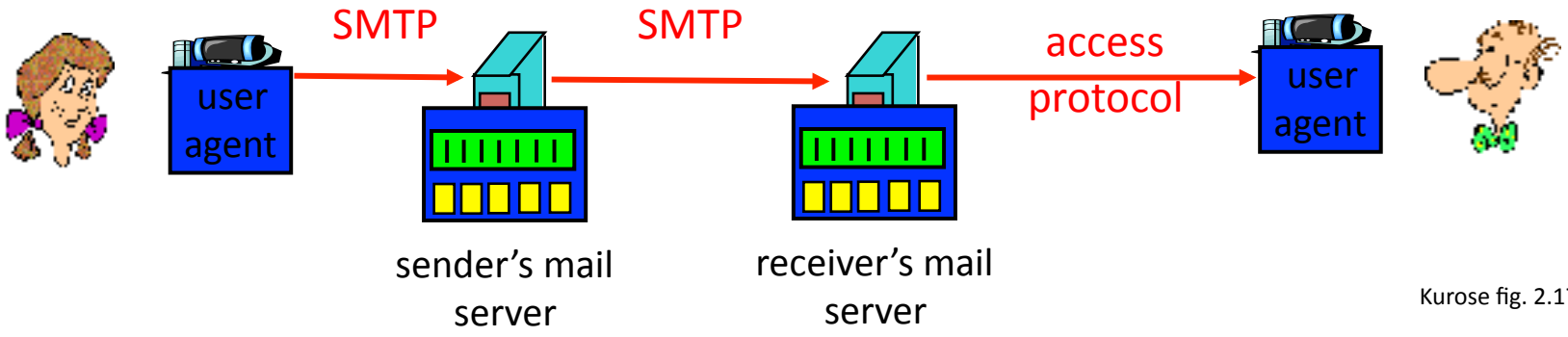
- dedicated program, e.g. outlook, elm, iPhone mail, …
- or webmail, e.g. uci webmail, gmail, …

Mail server: (server)

- server that talks to a dedicated program, e.g. smtp.uci.edu, smtp.cox.net, …
- or server that runs webmail

All user agents and mail servers are at application layer!

email: method & connection management

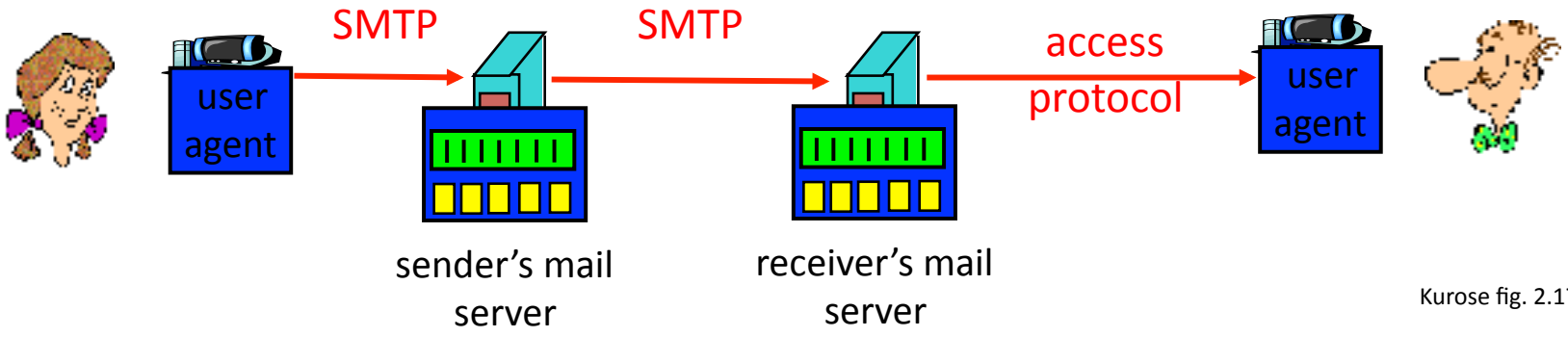


Kurose fig. 2.17

Upload protocol:

- SMTP (Simple Mail Transfer Protocol)
  - open TCP connection
  - upload messages
  - close TCP connection

email: method & connection management

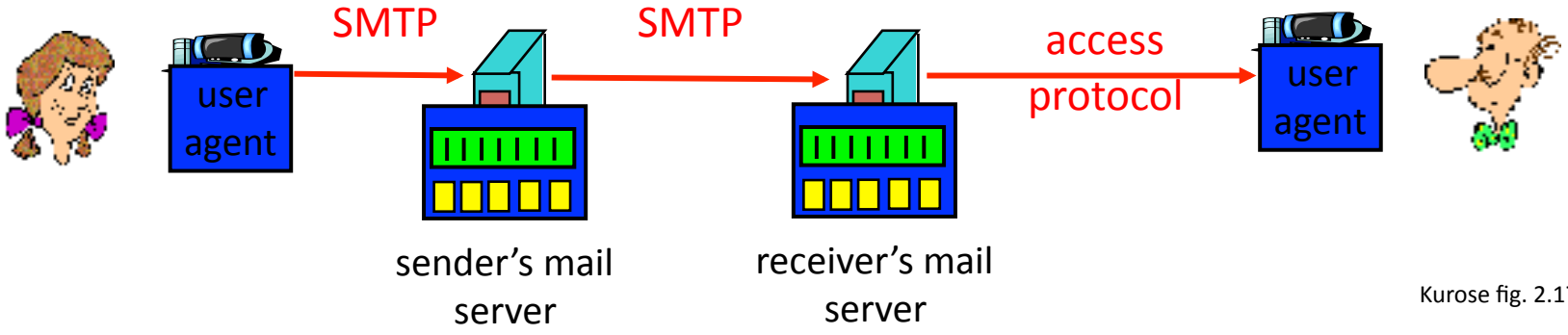


Kurose fig. 2.17

Download protocols:

- POP = Post Office Protocol
  - messages downloaded from mailbox on server to mailbox on non webmail client
- IMAP = Internet Mail Access Protocol
  - messages kept in (multiple) server mailboxes
  - or moved to mailboxes on non webmail clients

email: method & connection management



Kurose fig. 2.17

- Well-known ports (e.g. 25, 110, 143)
- SMTP, POP, IMAP standards plus email format standard mean Alice and Bob don't need to use same email program

email: location

- Client-server or peer-to-peer?
- Two functions:
  - Upload/download
    - Client-server
  - Mail transfer
    - Peer-to-peer (really server-to-server)