Application Layer
Part 5

Mark Allman
Case / ICSI

EECS 325/425
Fall 2018

"Well, beat the drum and hold the phone,
The sun came out today,
We're born again, there's new grass on the field"

Many of these slides are more-or-less directly from the slide set developed by Jim Kurose and Keith Ross for their book “Computer Networking: A Top Down Approach, 5th edition”.

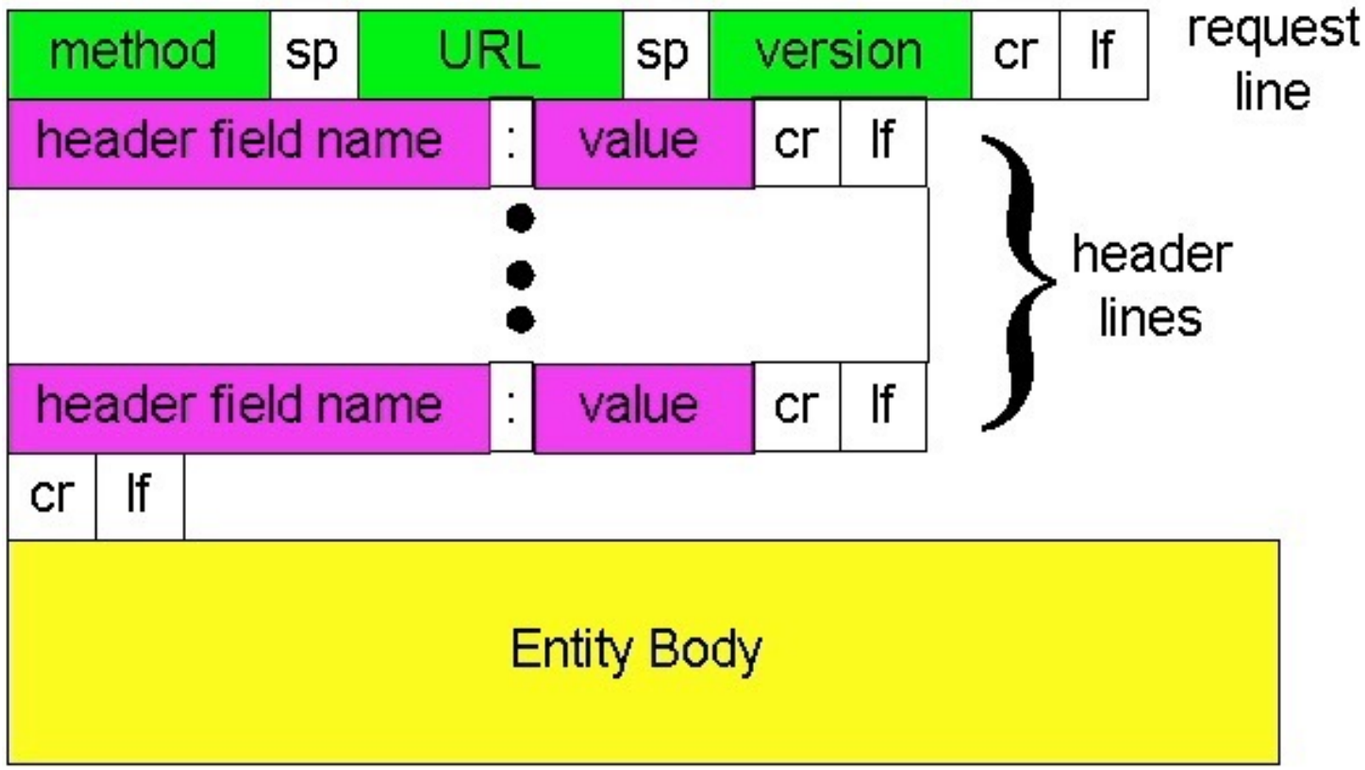
The slides have been lightly adapted for Mark Allman’s EECS 325/425 Computer Networks class at Case Western Reserve University.

All material copyright 1996-2010
J.F Kurose and K.W. Ross, All Rights Reserved

An Aside on Protocols …

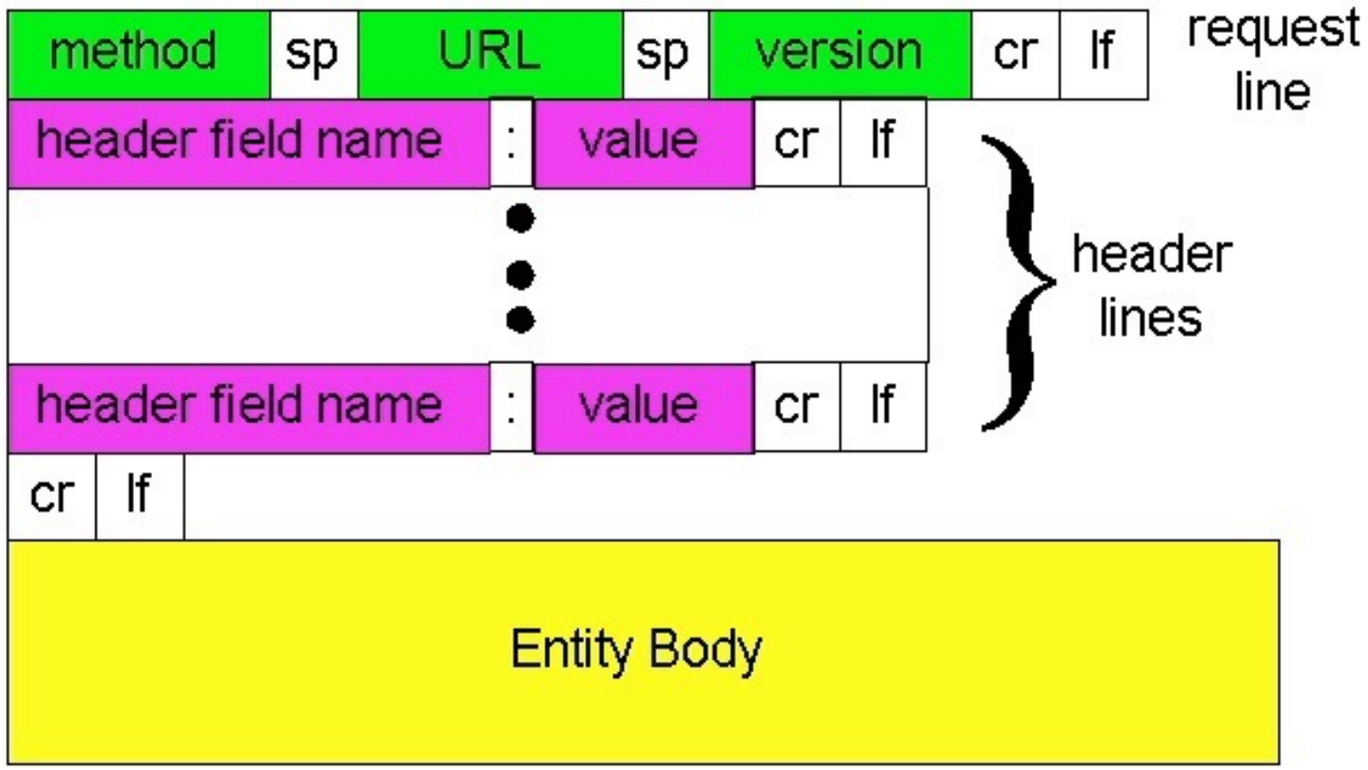
An Aside on Protocols …

- We have been examining HTTP …
- We learned this is the general form of an HTTP request …



An Aside on Protocols …

- We have been examining HTTP …
- We learned this is the general form of an HTTP request …



- Why?!

Original HTTP Specification

Network Working Group
Request for Comments: 1945
Category: Informational

T. Berners-Lee
MIT/LCS
R. Fielding
UC Irvine
H. Frystyk
MIT/LCS

May 1996

Hypertext Transfer Protocol -- HTTP/1.0

Standards

I E T F

Internet Engineering Task Force

IETF

- Develops Internet standards
- Meets three times / year
- Mostly does work on mailing lists
- Open …. anyone can “join”!
  - … as an individual
- Split into working groups based on technology
  - e.g., DNS, TCP, IPv6, …

IETF

- Specifications written in "Request For Comments" (RFCs)
  - name is historic
- Not all RFCs are standards (or specifications)
  - some experimental
  - some informational
  - etc.

RFCs

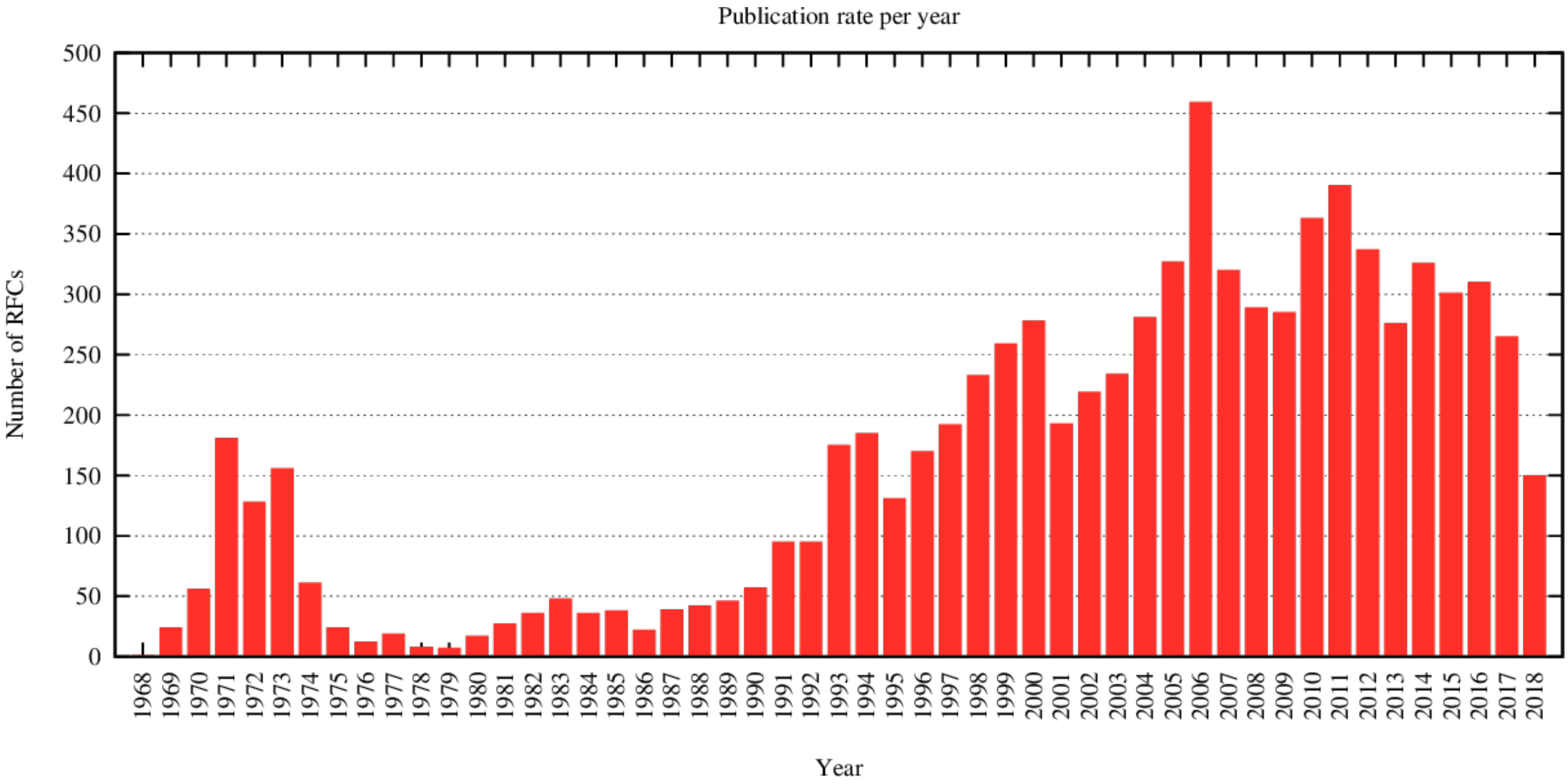
RFCs

- Over 8,400 RFCs published
  - 1st: April 1969
  - Last: September 6, 2018

RFCs

- Over 8,400 RFCs published
  - 1st: April 1969
  - Last: September 6, 2018
- (be wary of RFCs dated April 1!)

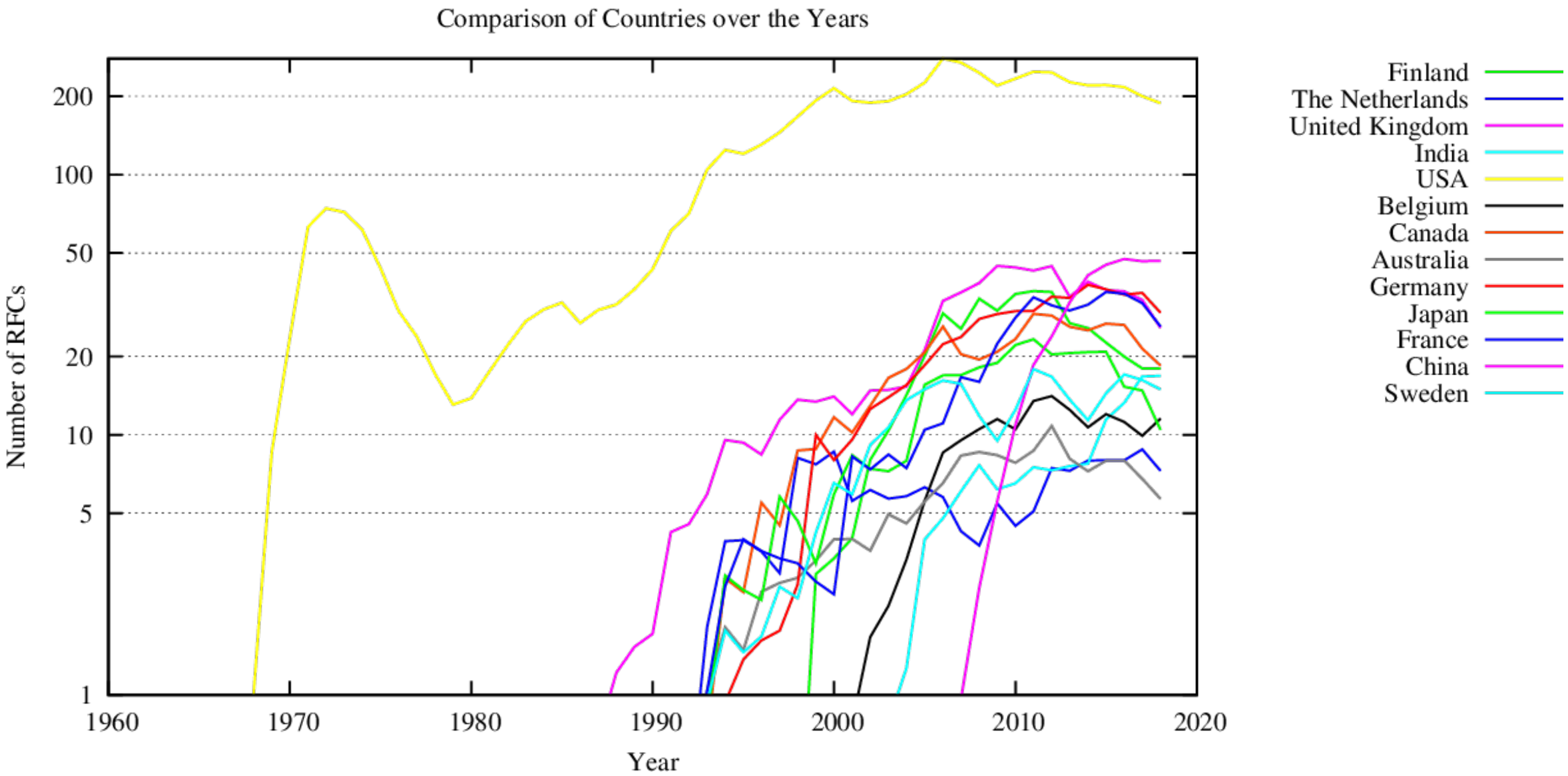
RFCs



Allman

https://www.arkko.com/tools/docstats.html

RFCs



Allman

https://www.arkko.com/tools/docstats.html

IETF

IETF

"We reject kings, presidents and voting. We believe in rough consensus and running code."
—David Clark

Consensus

- WGs and the broader IETF have a variety of ways to assess consensus
- One is to get the "sense of the room" at in-person meetings

Beyond Basic HTTP

- HTTP is a *big* protocol / ecosystem with many extensions and variants
- One more example extension …

HTTP state

HTTP state

HTTP is "stateless"

- server maintains no information about past client requests

HTTP state

HTTP is "stateless"

- server maintains no information about past client requests

protocols that maintain "state" are complex!

HTTP state

HTTP is "stateless"

- server maintains no information about past client requests

protocols that maintain "state" are complex!

- past history (state) must be maintained
- if server/client crashes, their views of "state" may be inconsistent, must be reconciled

User-server state: cookies

User-server state: cookies

many Web sites use
cookies

four components:

1) cookie header line of
HTTP response message
2) cookie header line in
HTTP request message
3) cookie file kept on
user's host, managed by
user's browser
4) back-end database at
Web site

User-server state: cookies

many Web sites use cookies

four components:

1) cookie header line of HTTP response message
2) cookie header line in HTTP request message
3) cookie file kept on user's host, managed by user's browser
4) back-end database at Web site

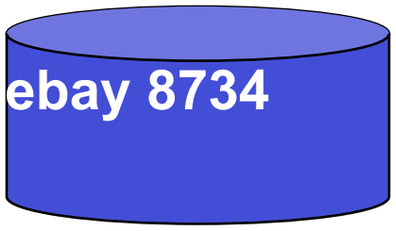
example:

- Susan always access Internet from PC
- visits specific e-commerce site for first time
- when initial HTTP request arrives at site, site creates:
  - unique ID
  - entry in backend database for ID

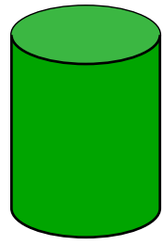
Cookies: keeping "state" (cont.)

client

Amazon server



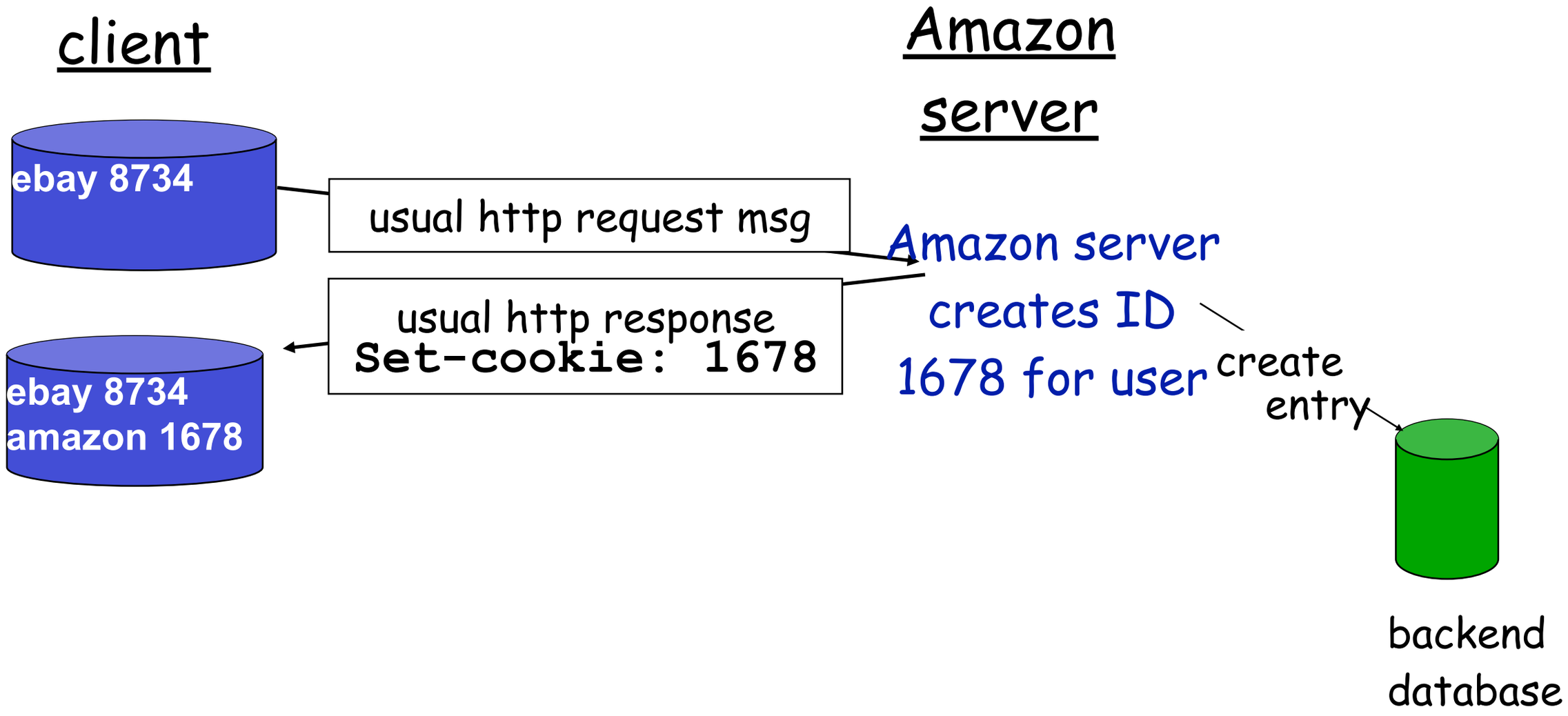
cookie file



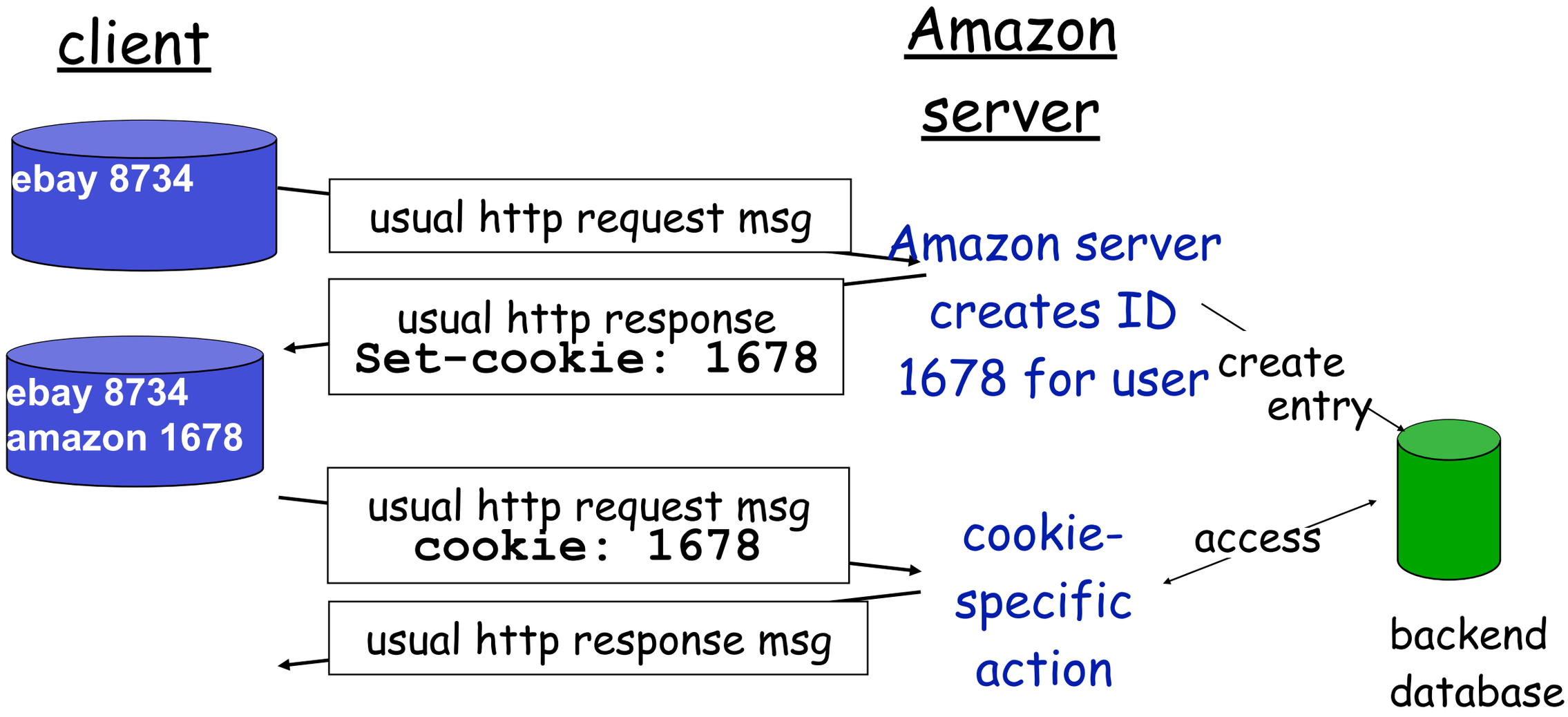
backend database

Cookies: keeping "state" (cont.)

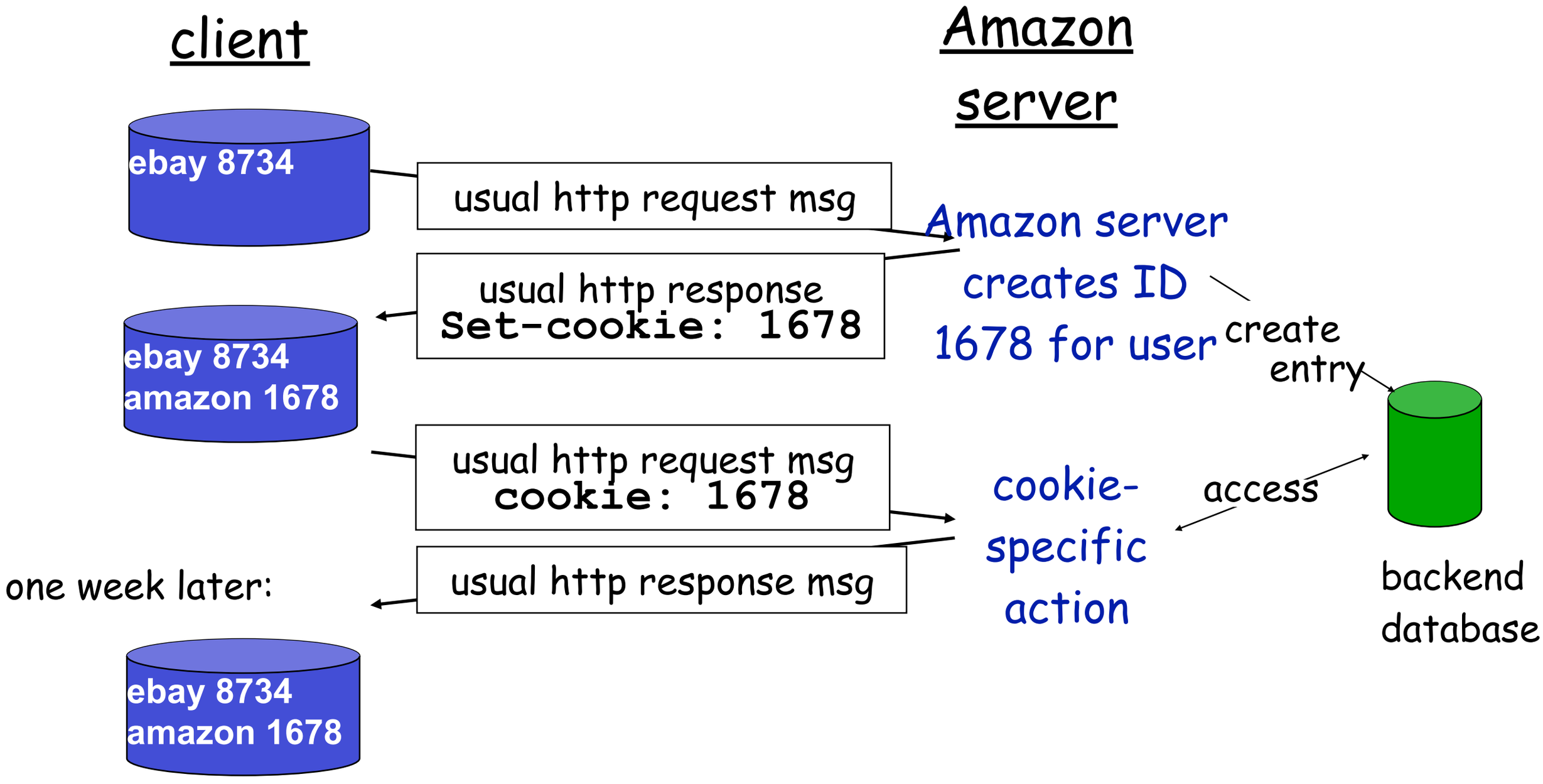
Cookies: keeping "state" (cont.)



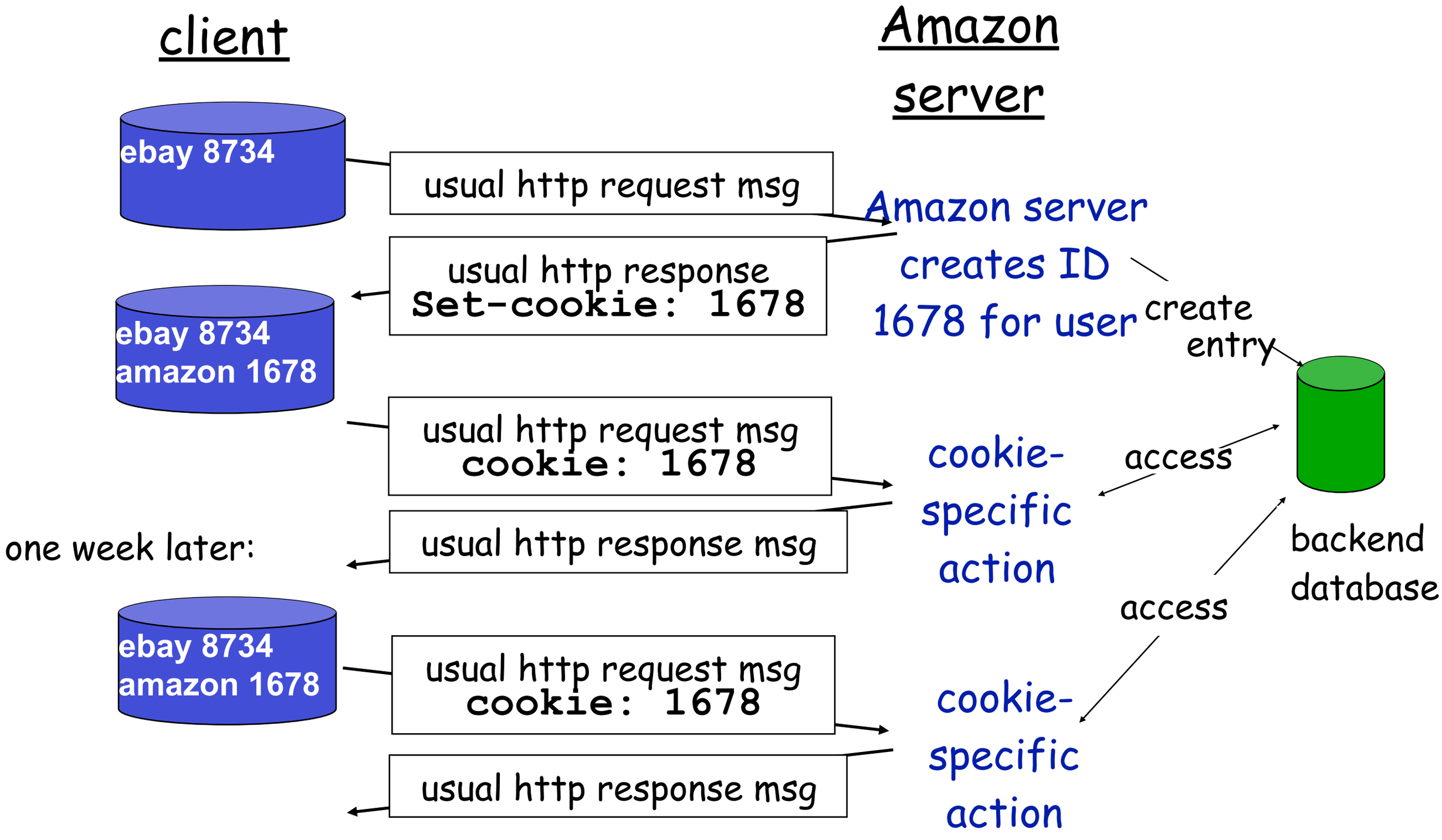
Cookies: keeping "state" (cont.)



Cookies: keeping "state" (cont.)



Cookies: keeping "state" (cont.)



Cookies (continued)

Cookies (continued)

what cookies can bring:

- authorization
- shopping carts
- recommendations
- user session state (Web e-mail)

Cookies (continued)

what cookies can bring:

- authorization
- shopping carts
- recommendations
- user session state (Web e-mail)

how to keep "state":

- protocol endpoints: maintain state at sender/receiver over multiple transactions
- cookies: http messages carry state

Cookies (continued)

what cookies can bring:

- authorization
- shopping carts
- recommendations
- user session state (Web e-mail)

aside

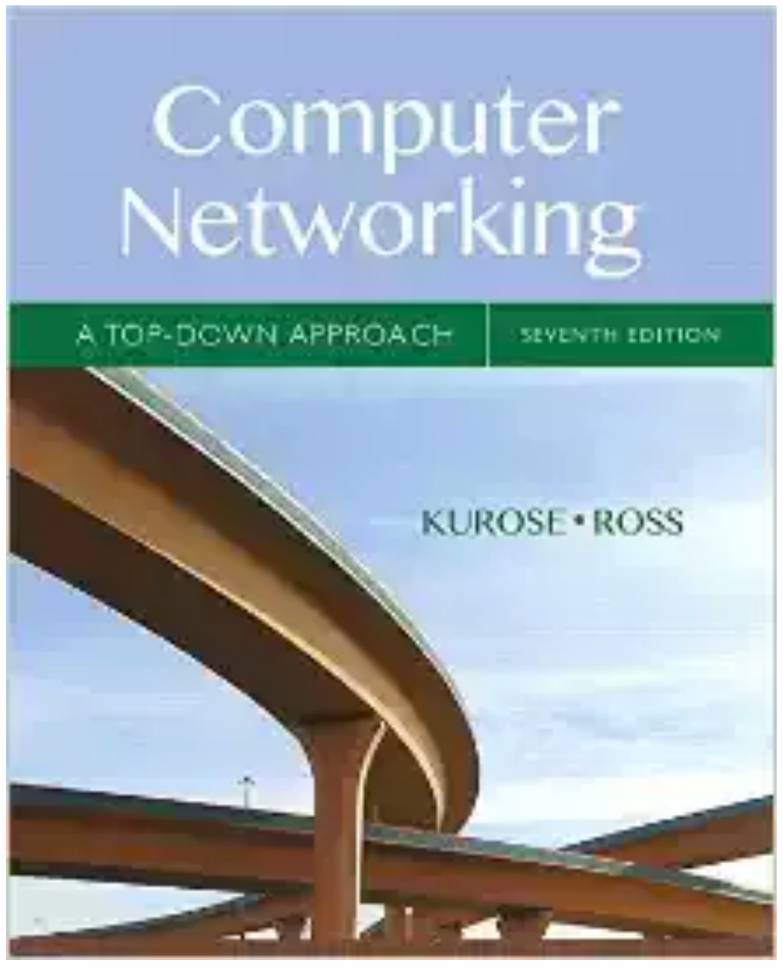
cookies and privacy:

- cookies permit sites to learn a lot about you
- e..g., you may supply name and e-mail to sites

how to keep "state":

- protocol endpoints: maintain state at sender/receiver over multiple transactions
- cookies: http messages carry state

Reading Along ...



- 2.4: Email

Electronic Mail

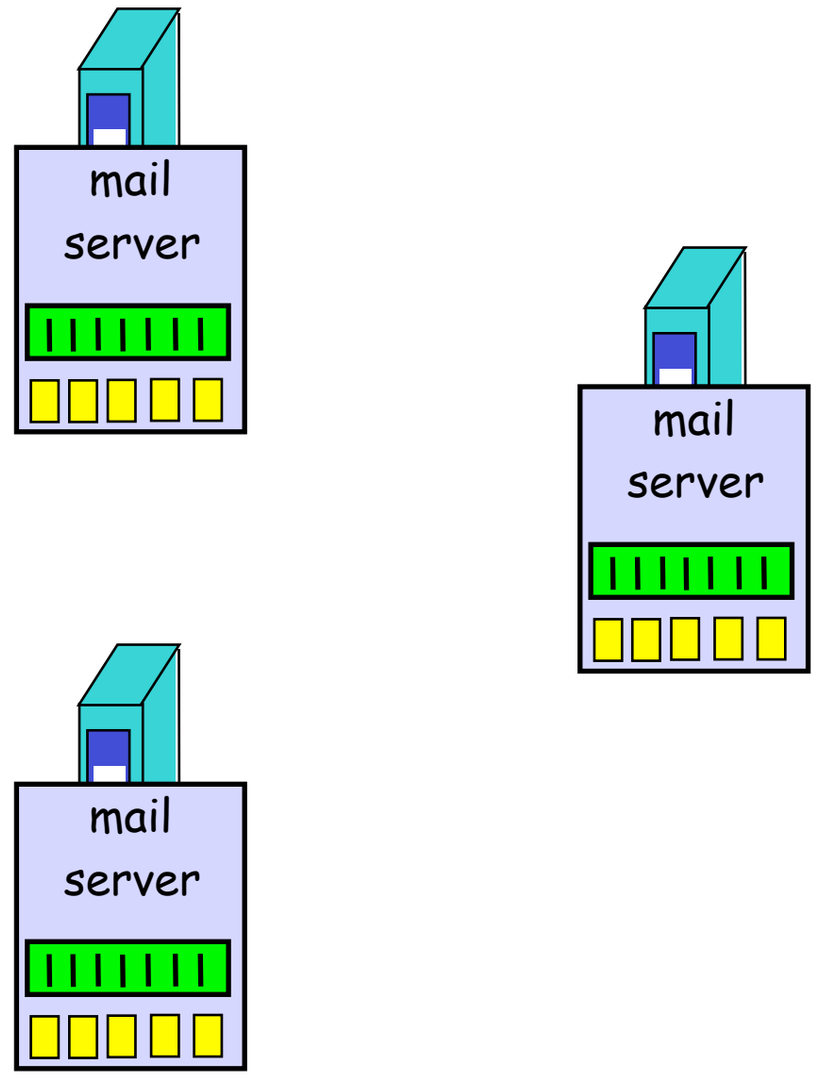
Electronic Mail

Three major components:

Electronic Mail

Three major components:

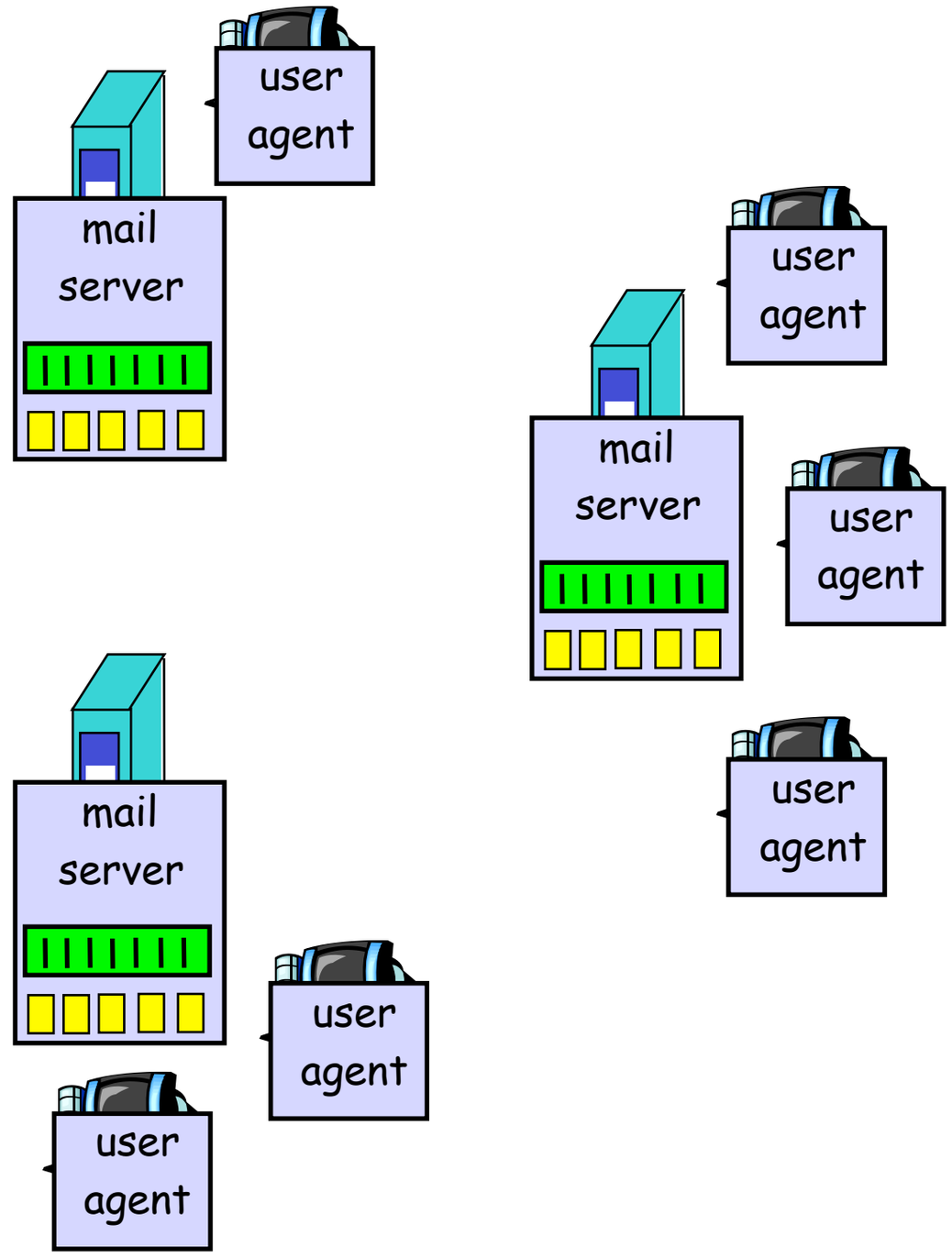
- mail servers (MTA)



Electronic Mail

Three major components:

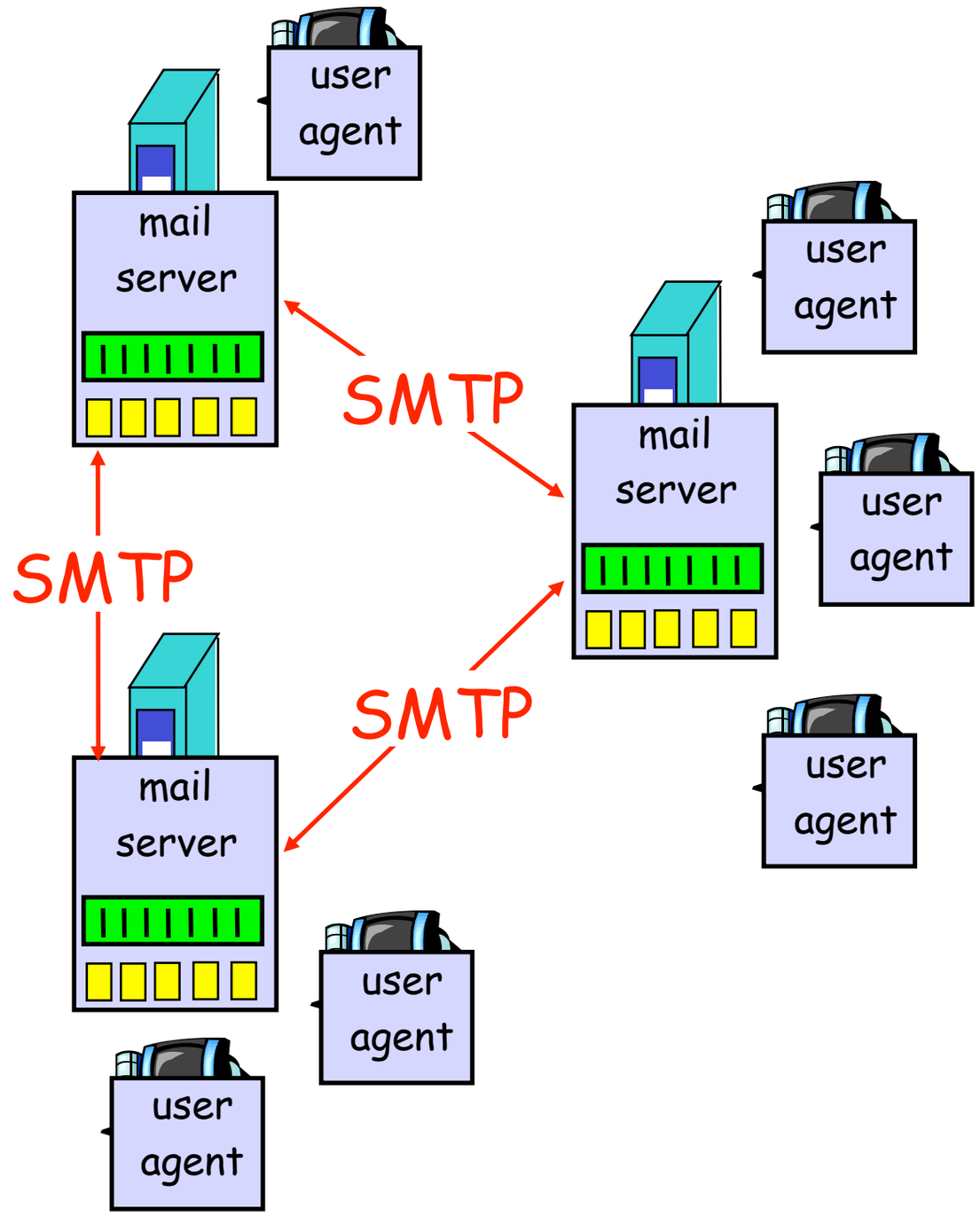
- mail servers (MTA)
- user agents (MUA)



Electronic Mail

Three major components:

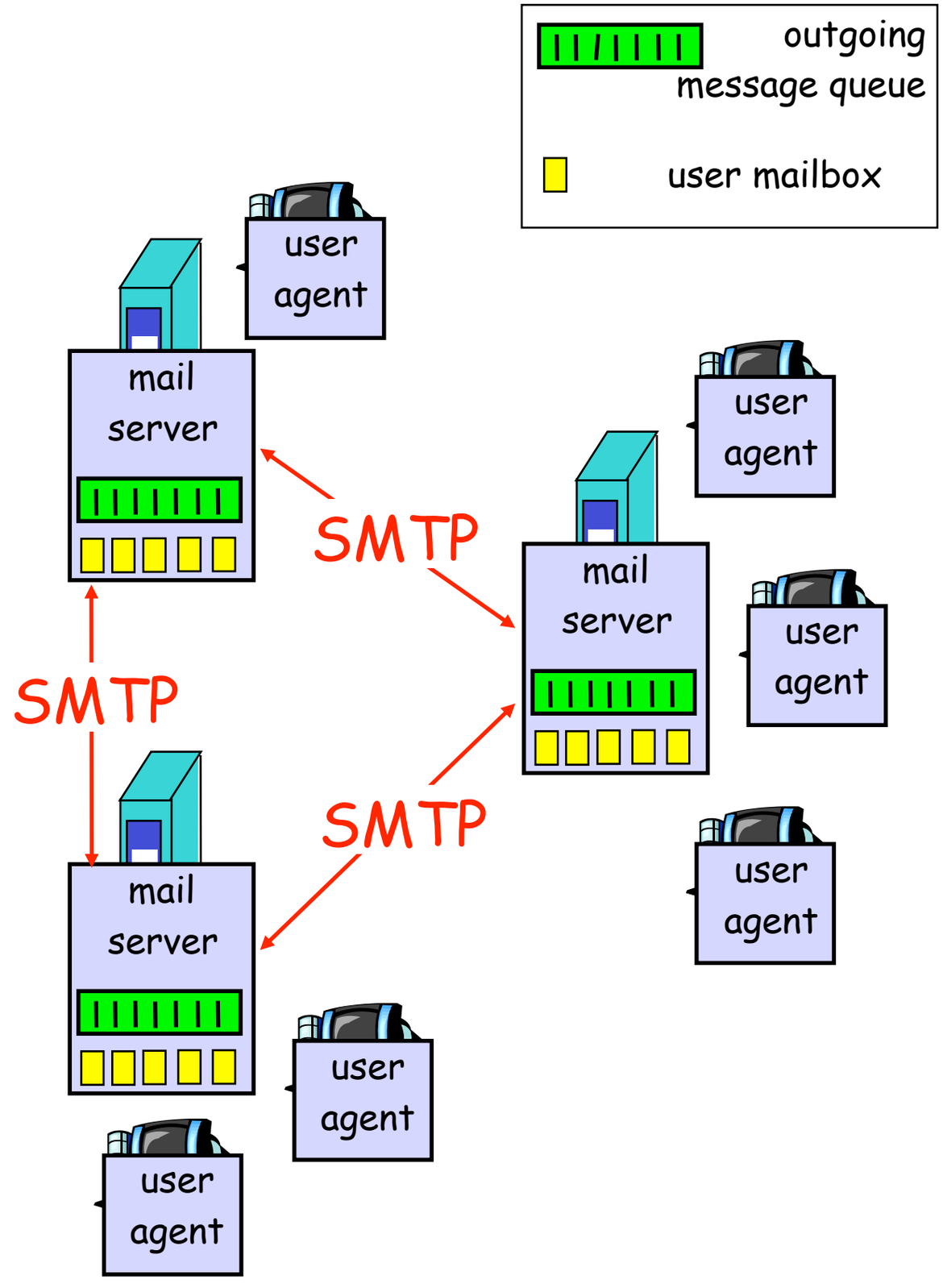
- mail servers (MTA)
- user agents (MUA)
- simple mail transfer protocol: SMTP



Electronic Mail

Three major components:

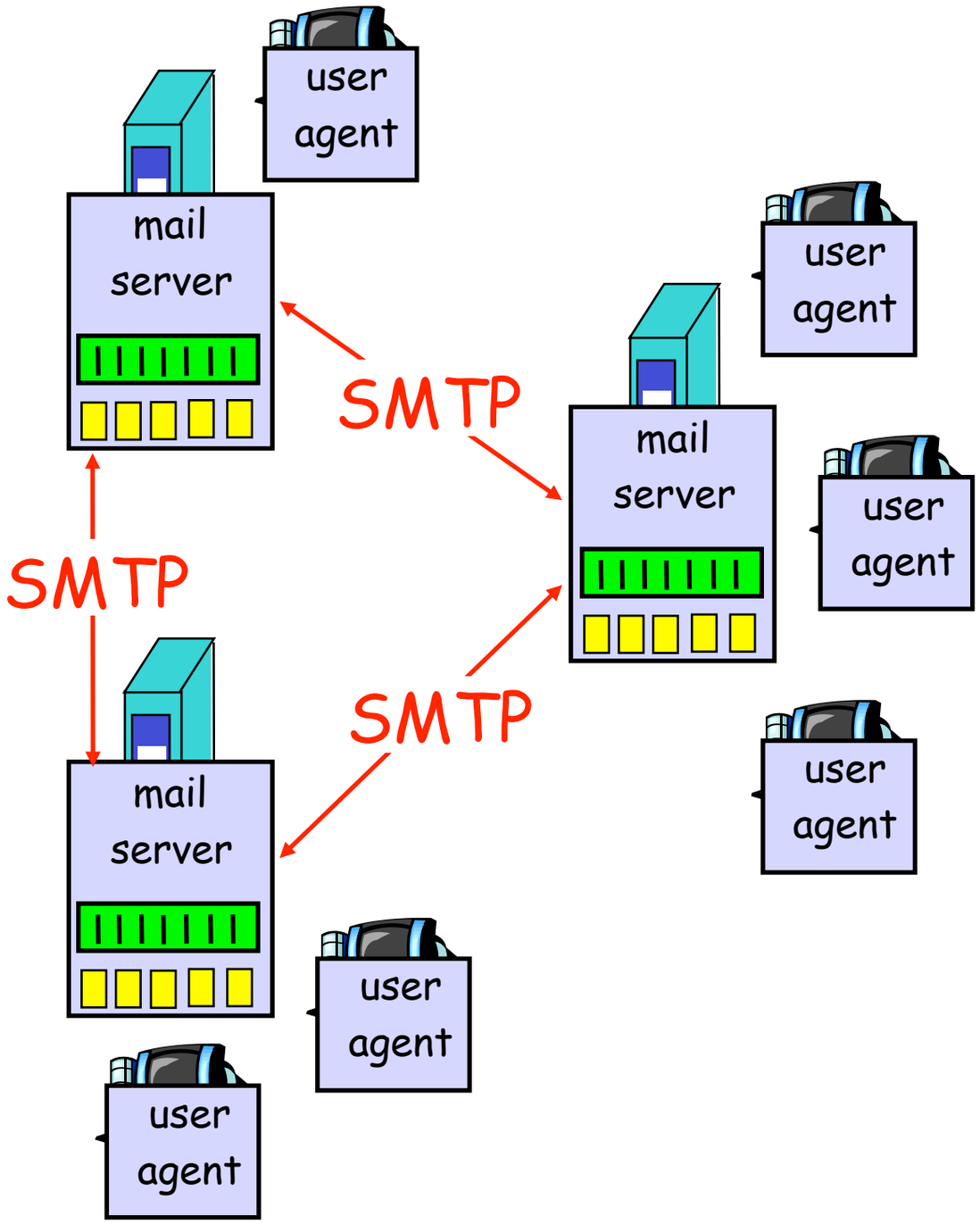
- mail servers (MTA)
- user agents (MUA)
- simple mail transfer protocol: SMTP



User Agent

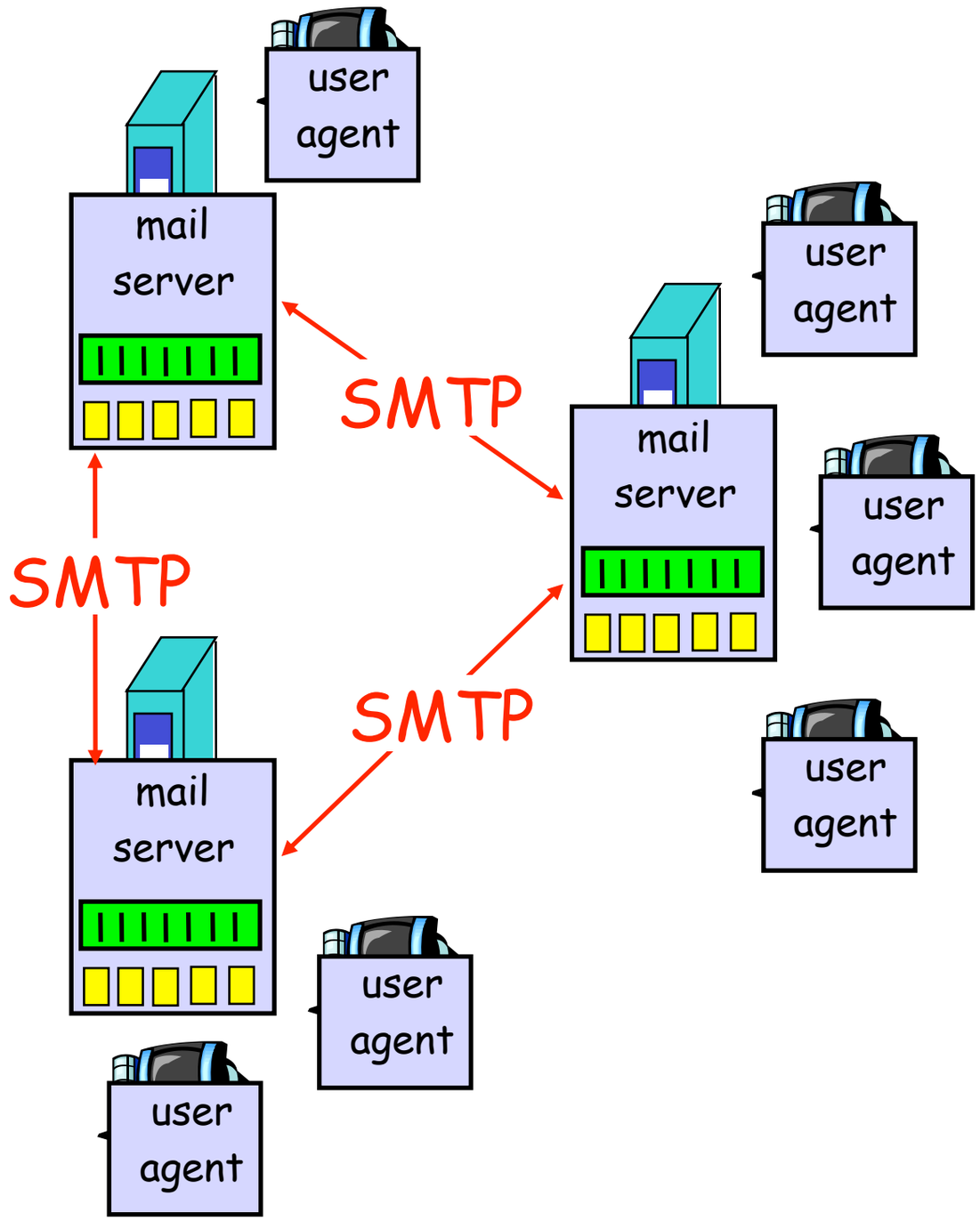
- a.k.a. "mail reader"
- composing, editing, reading mail messages
- e.g., Outlook, elm, Mozilla Thunderbird, iPhone mail client, web mail
- outgoing, incoming messages stored on server

Electronic Mail: mail servers



Electronic Mail: mail servers

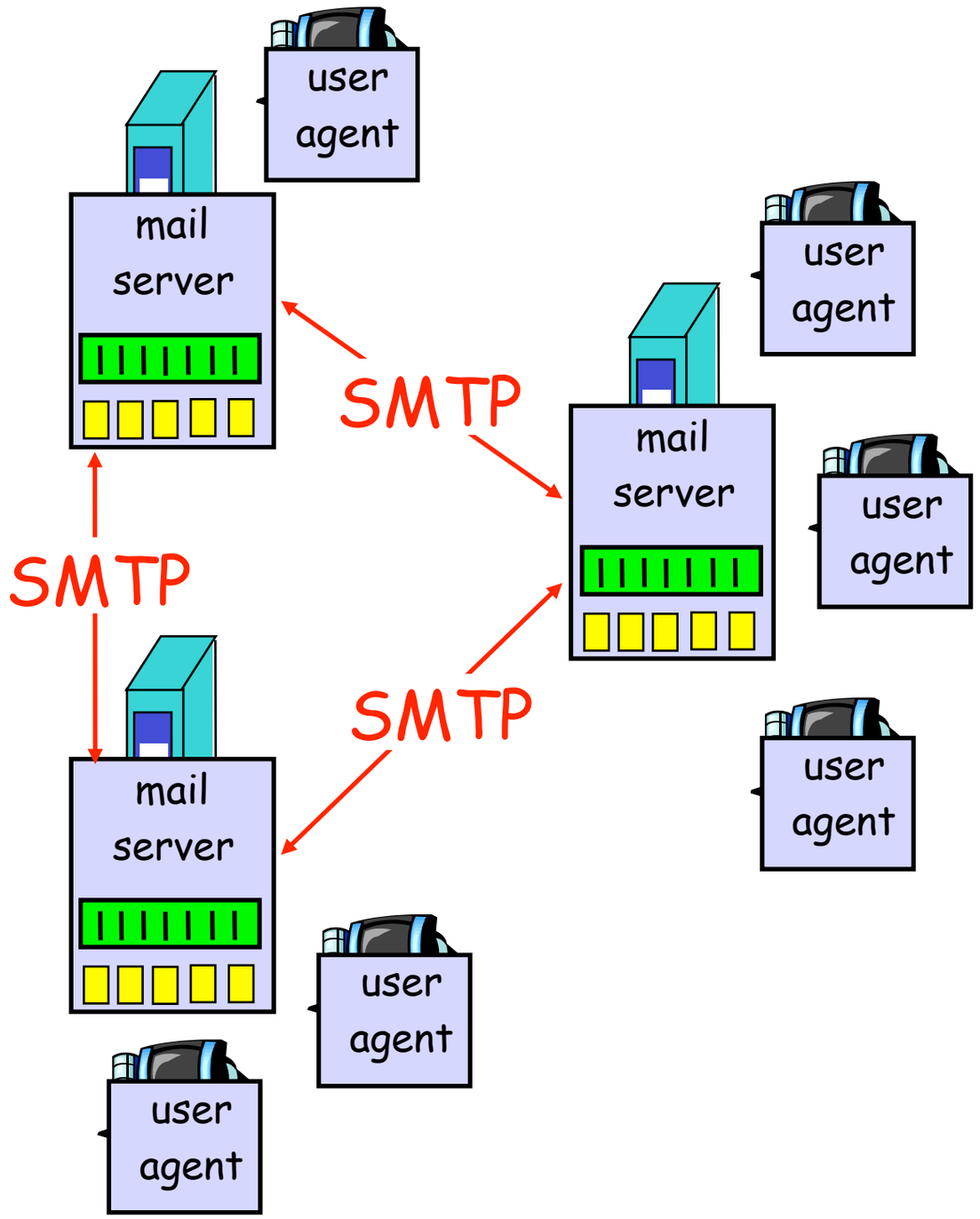
Mail Servers



Electronic Mail: mail servers

Mail Servers

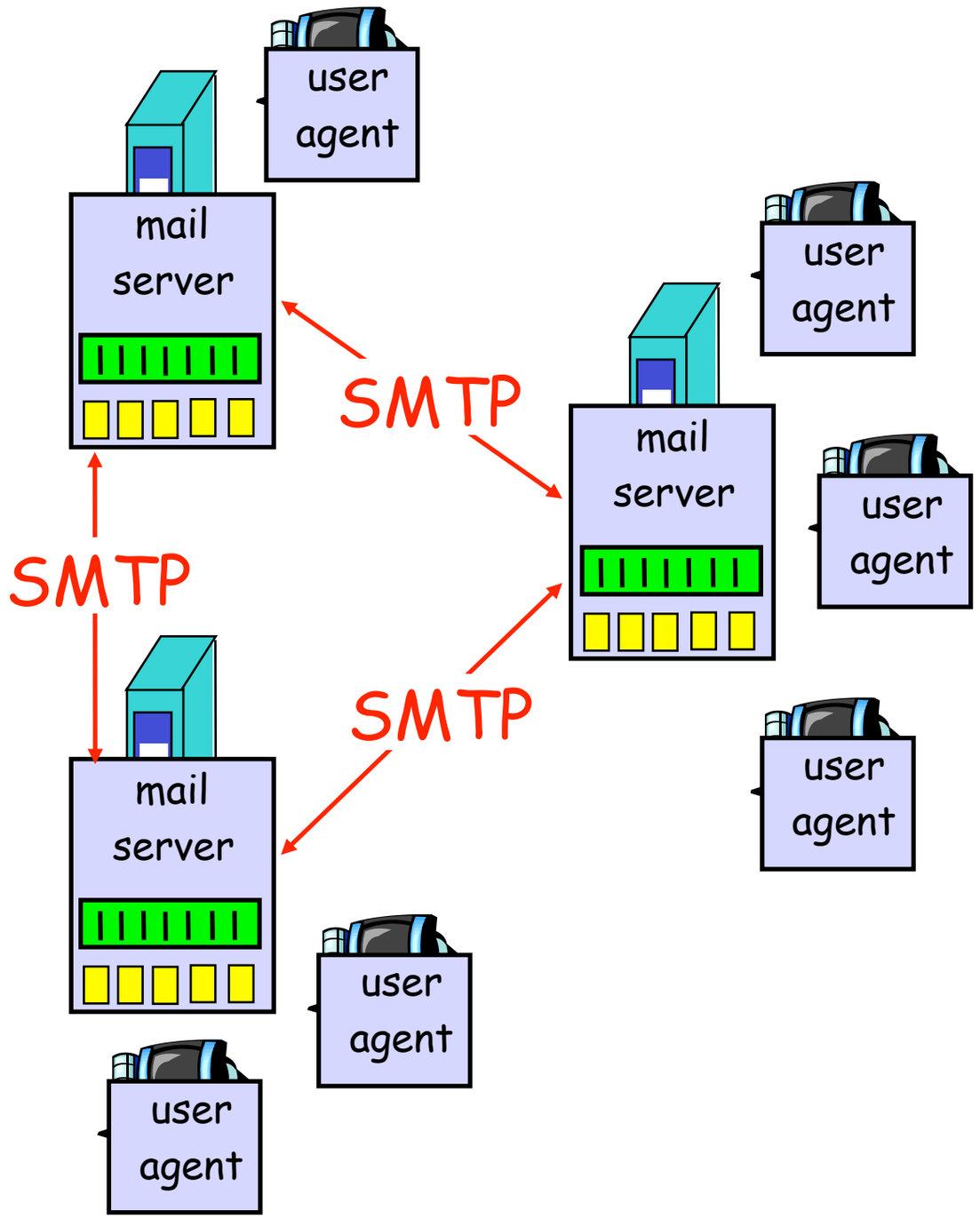
- mailbox contains incoming messages for user



Electronic Mail: mail servers

Mail Servers

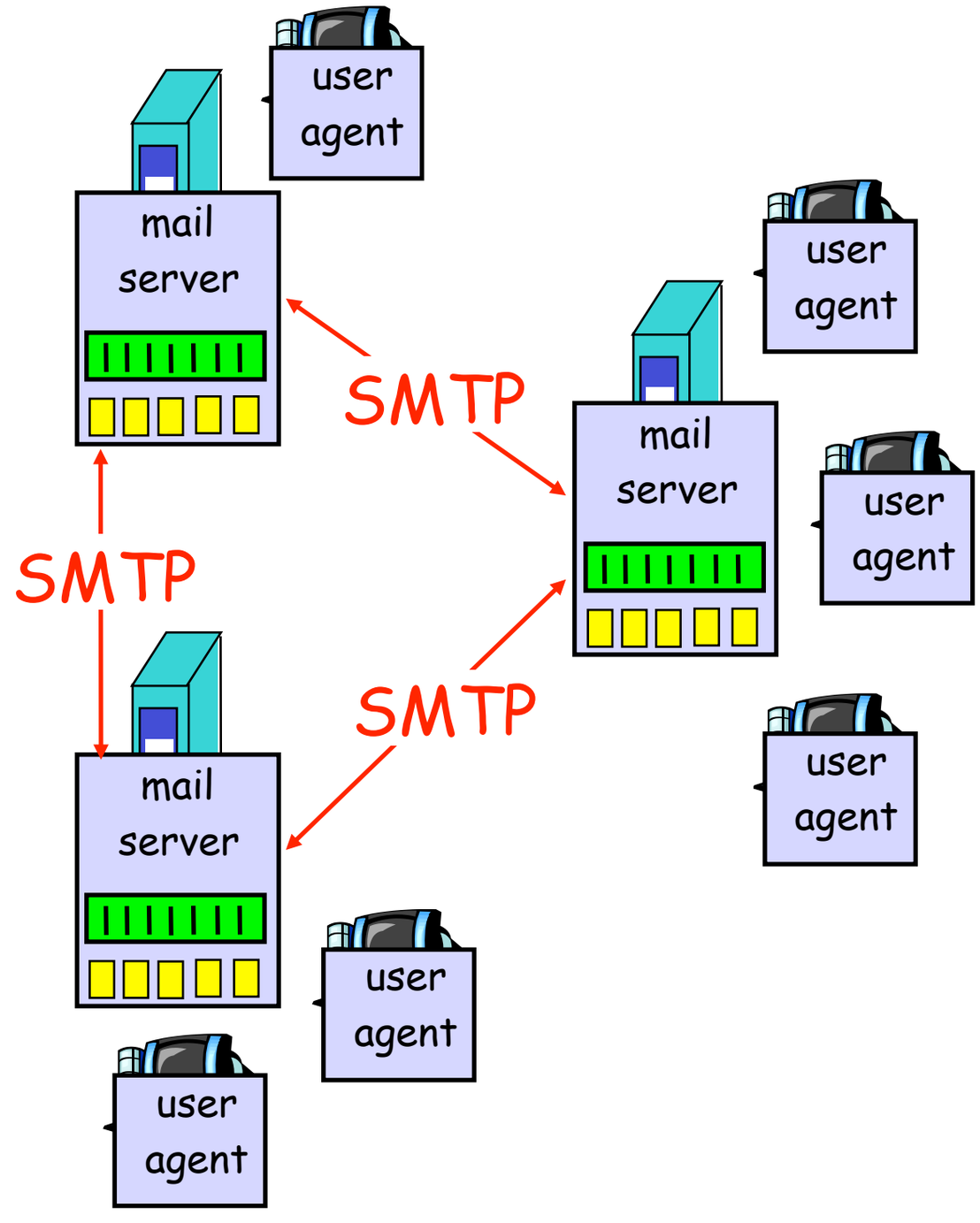
- mailbox contains incoming messages for user
- message queue of outgoing (to be sent) mail messages



Electronic 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



Electronic Mail: SMTP [RFC 2821]

Electronic Mail: SMTP [RFC 2821]

- uses TCP to transfer email message from client to server, port 25

Electronic Mail: SMTP [RFC 2821]

- uses TCP to transfer email message from client to server, port 25
- direct transfer: sending server to receiving server

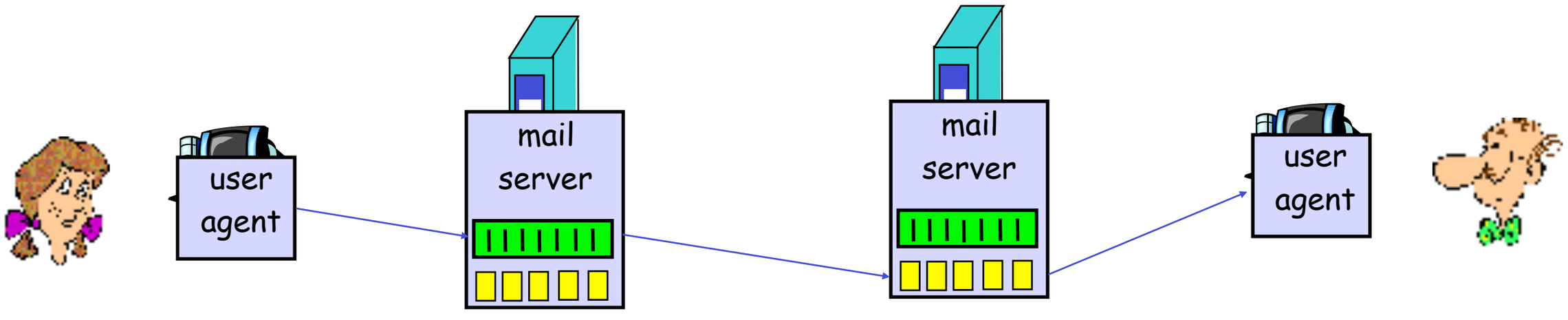
Electronic Mail: SMTP [RFC 2821]

- uses TCP to transfer email message from client to server, port 25
- direct transfer: sending server to receiving server
- three phases of transfer
  - handshaking (greeting)
  - transfer of messages
  - closure

Electronic Mail: SMTP [RFC 2821]

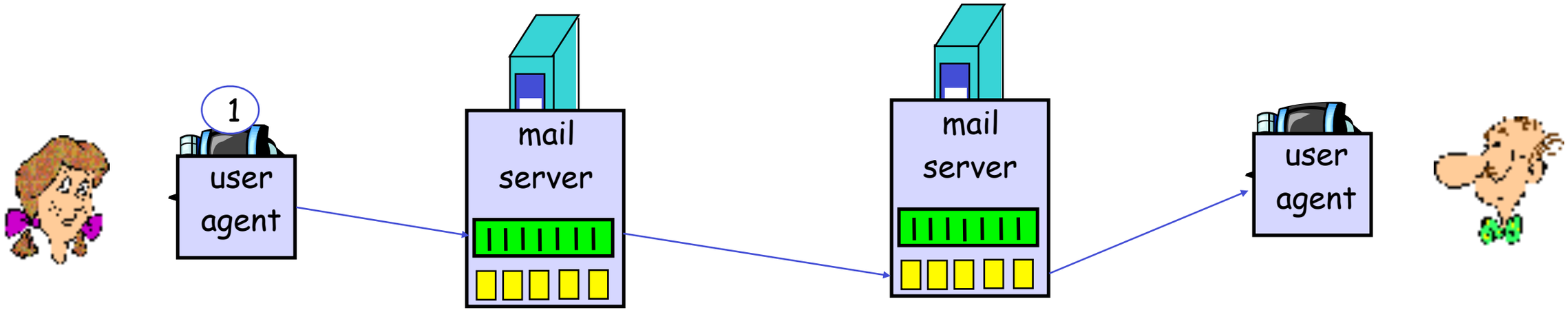
- uses TCP to transfer email message from client to server, port 25
- direct transfer: sending server to receiving server
- three phases of transfer
  - handshaking (greeting)
  - transfer of messages
  - closure
- command/response interaction
  - commands: ASCII text
  - response: status code and phrase

Scenario: Alice sends message to Bob



Scenario: Alice sends message to Bob

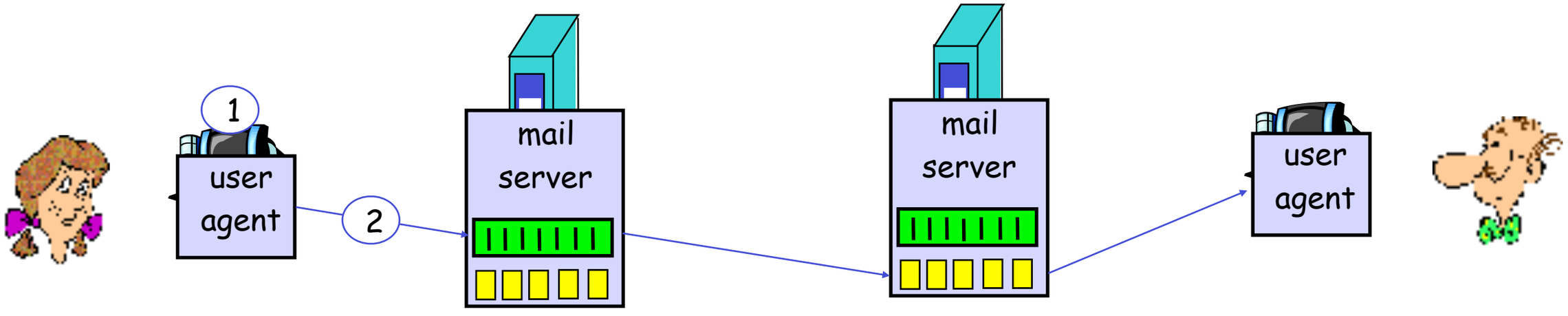
1) Alice uses UA to compose message "to" `bob@someschool.edu`



Scenario: Alice sends message to Bob

1) Alice uses UA to compose message "to" `bob@someschool.edu`

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

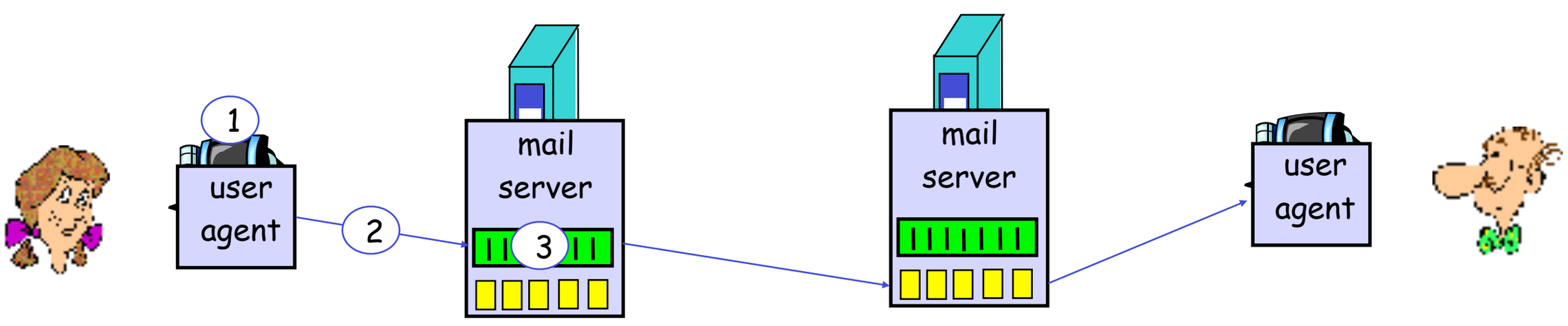


Scenario: Alice sends message to Bob

1) Alice uses UA to compose 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



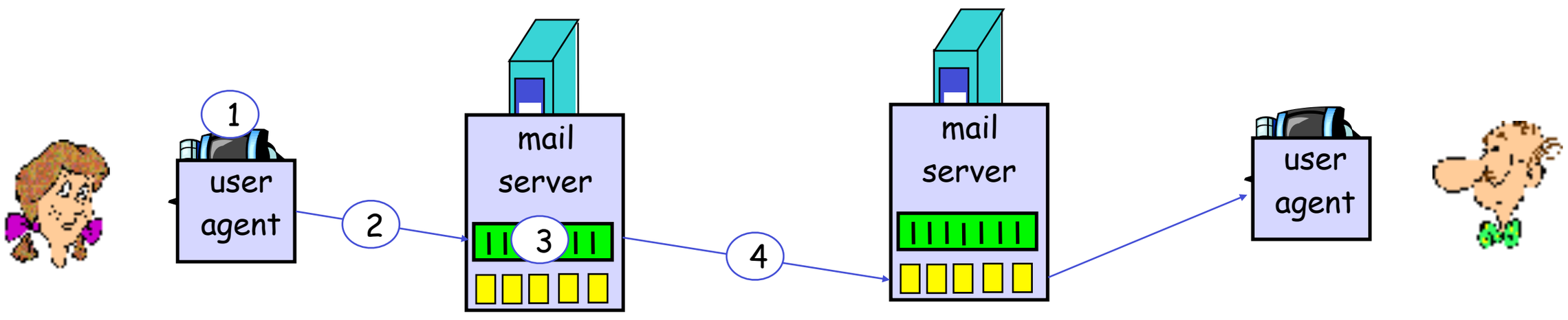
Scenario: Alice sends message to Bob

1) Alice uses UA to compose 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



Scenario: Alice sends message to Bob

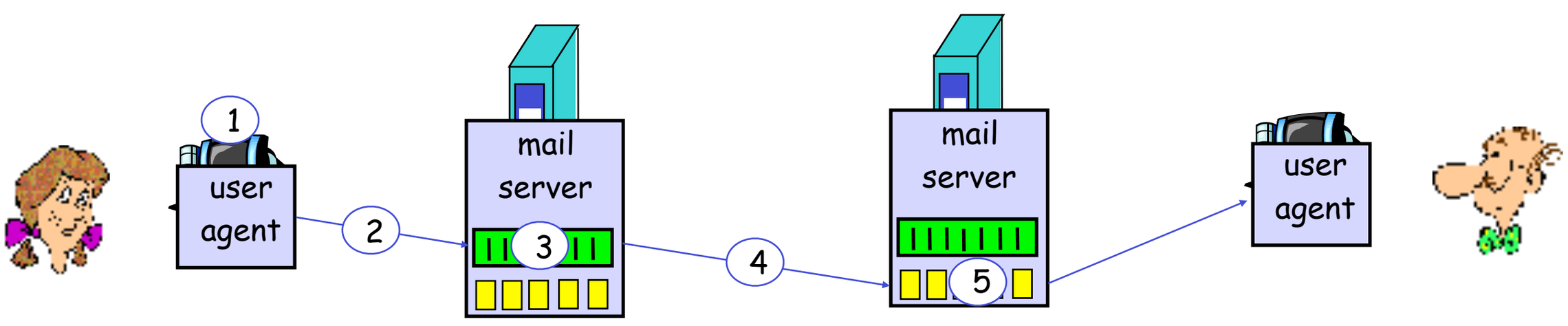
1) Alice uses UA to compose message "to" `bob@someschool.edu`

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

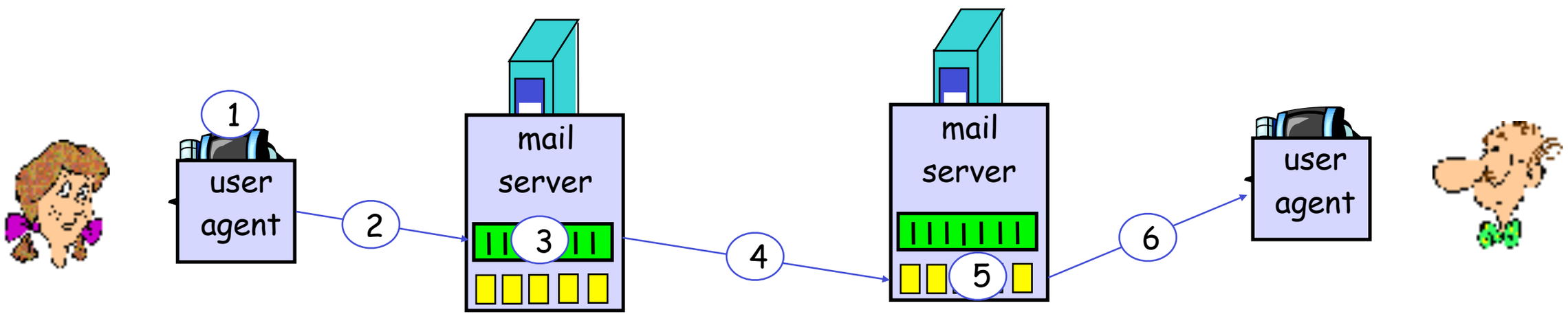
4) SMTP client sends Alice's message over the TCP connection

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



Scenario: Alice sends message to Bob

1) Alice uses UA to compose 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

Sample SMTP interaction

```
S: 220 hamburger.edu
```

Sample SMTP interaction

```
S: 220 hamburger.edu
C: HELO crepes.fr
```

Sample SMTP interaction

```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250  Hello crepes.fr, pleased to meet you
```

Sample SMTP interaction

```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250  Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
```

Sample SMTP interaction

```
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
```

Sample SMTP interaction

```
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>
```

Sample SMTP interaction

```
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
```

Sample SMTP interaction

```
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
```

Sample SMTP interaction

```
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
```

Sample SMTP interaction

```
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: .
```

Sample SMTP interaction

```
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
```

Sample SMTP interaction

```
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
```

Sample SMTP interaction

```
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
```

Sample SMTP interaction

```
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
```

Sample SMTP interaction

```
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
```

Try SMTP interaction for yourself:

- **`telnet servername 25`**
- see 220 reply from server
- enter HELO, MAIL FROM, RCPT TO, DATA, QUIT commands

above lets you send email without using email client (reader)

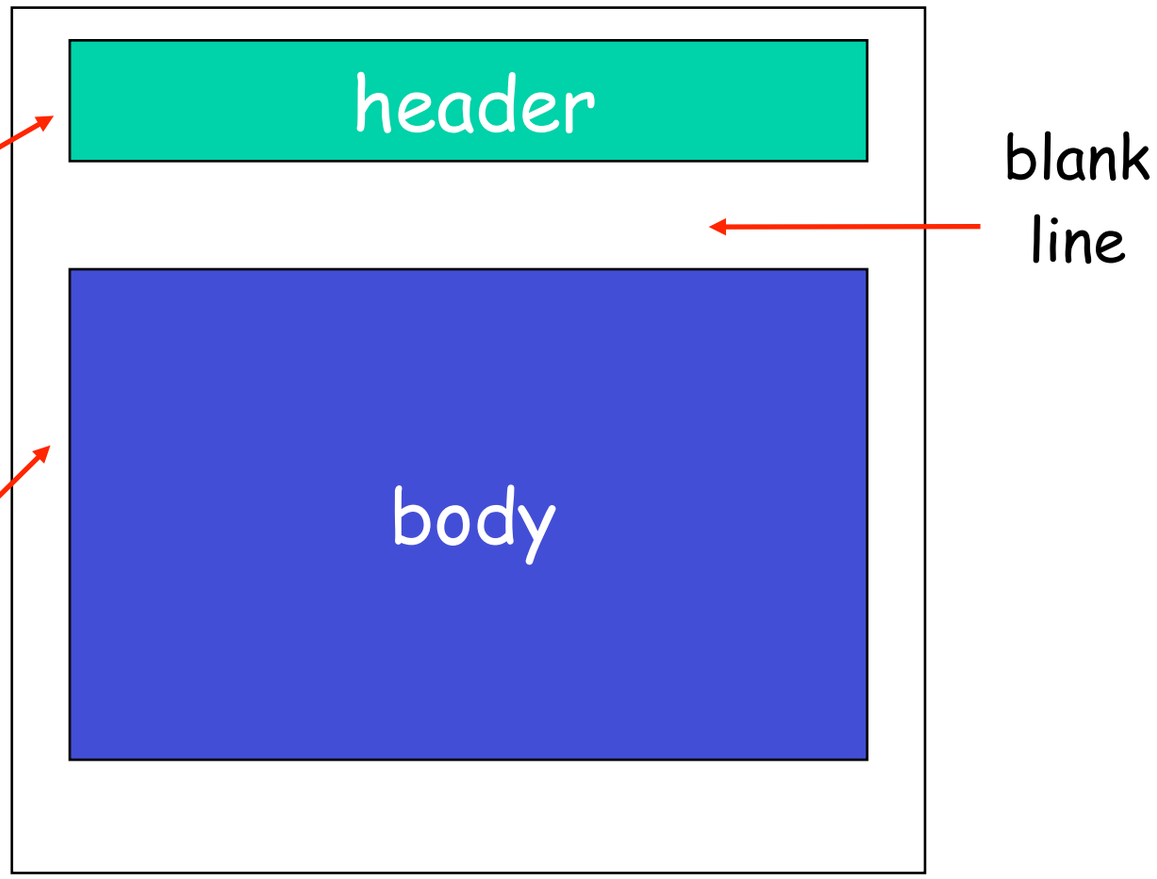
Mail message format

SMTP: protocol for exchanging email msgs

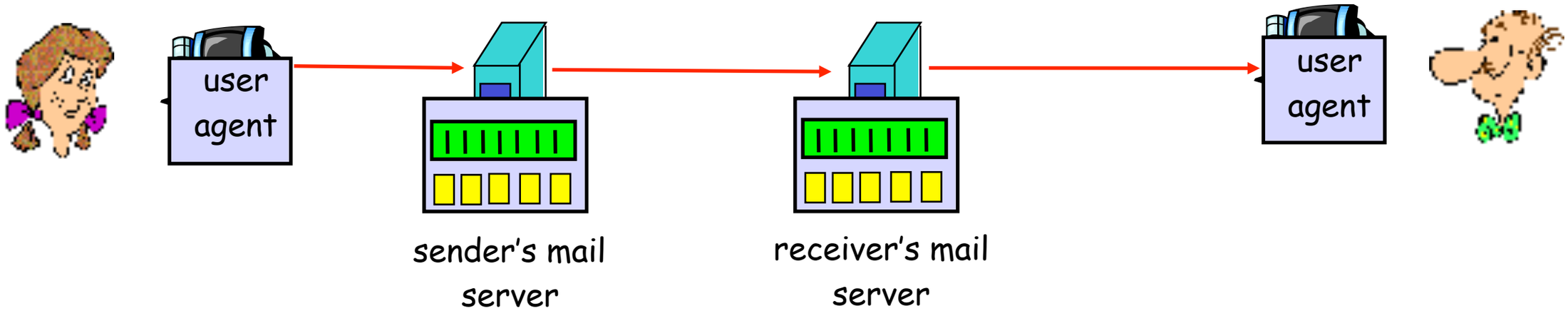
RFC 822: standard for text message format:

- header lines, e.g.,
  - To:
  - From:
  - Subject:

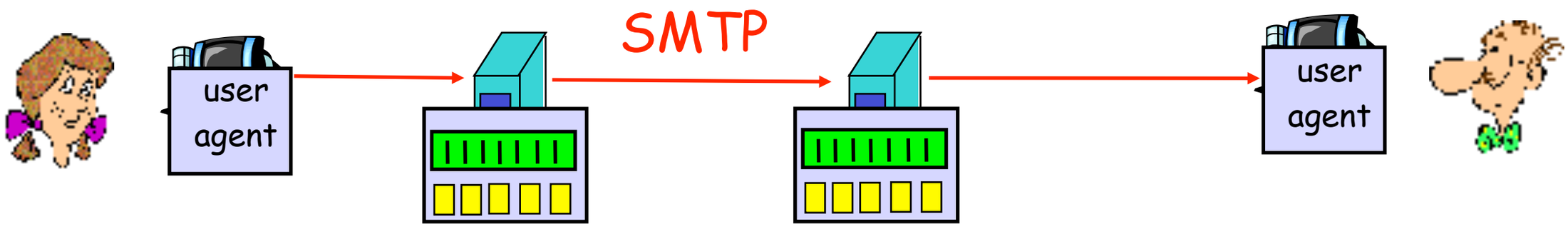
  different from SMTP commands!
- body
  - the "message"



Refining Mail Delivery

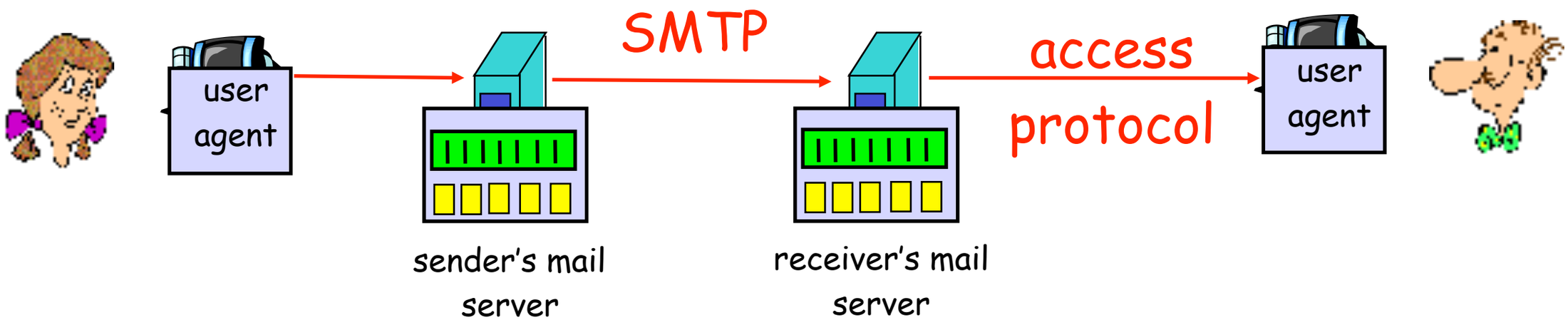


Refining Mail Delivery



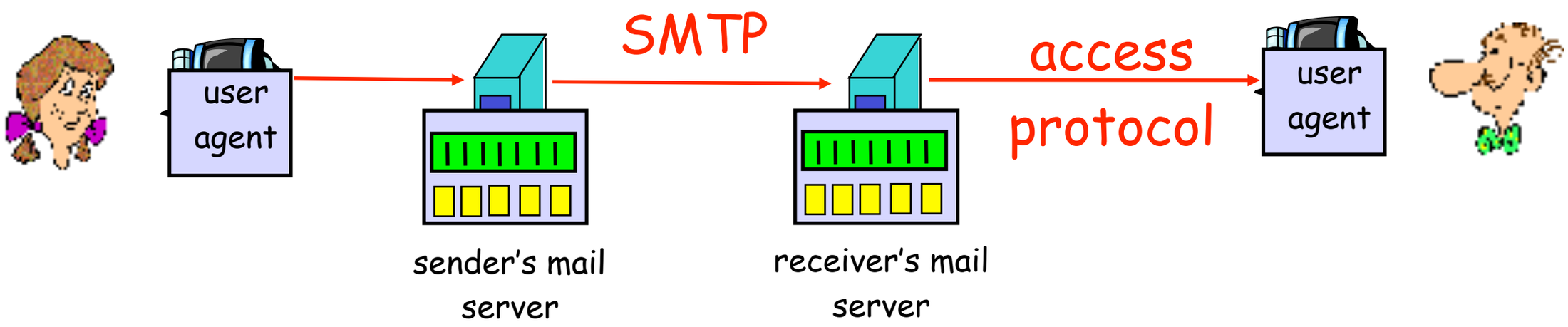
- SMTP: delivery/storage to receiver's server

Refining Mail Delivery



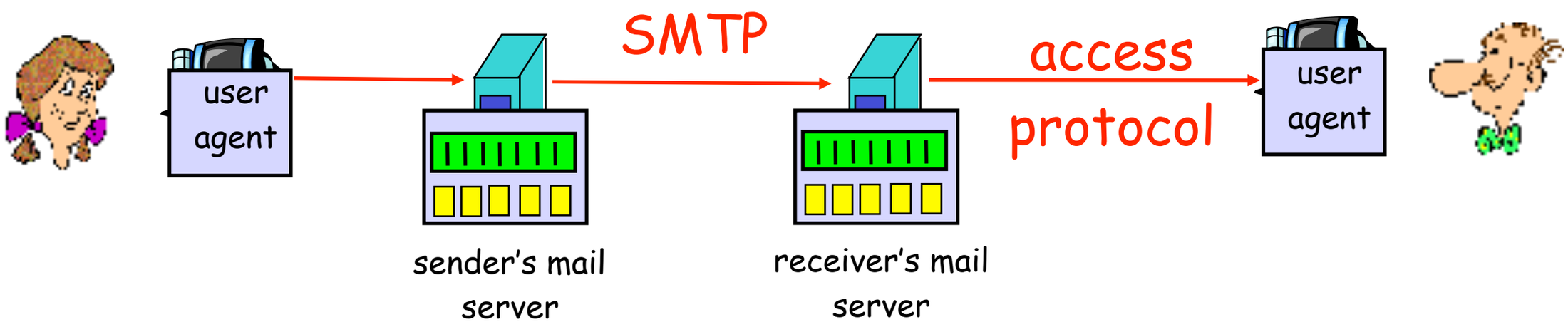
- SMTP: delivery/storage to receiver's server
- mail access protocol: retrieval from server

Refining Mail Delivery



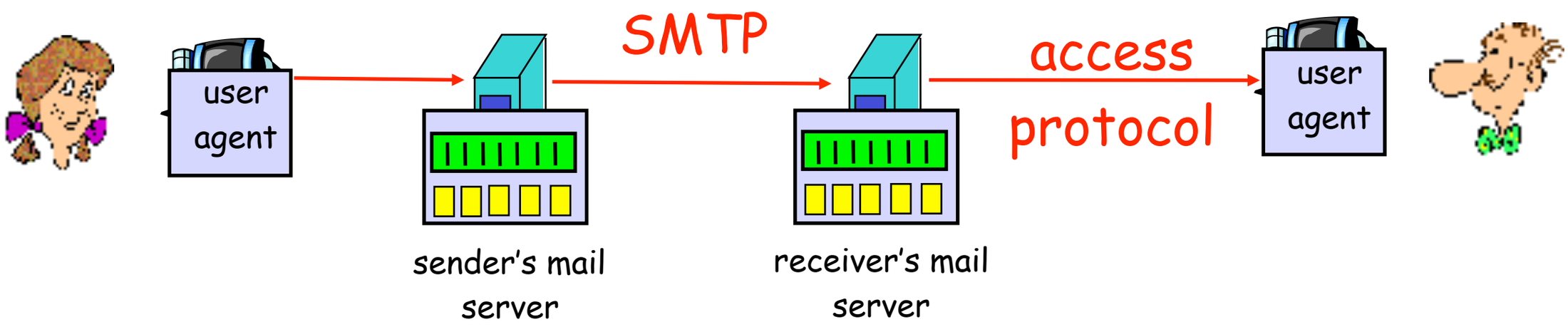
- SMTP: delivery/storage to receiver's server
- mail access protocol: retrieval from server
  - POP: Post Office Protocol [RFC 1939]
    - authorization (agent <-->server) and download

Refining Mail Delivery



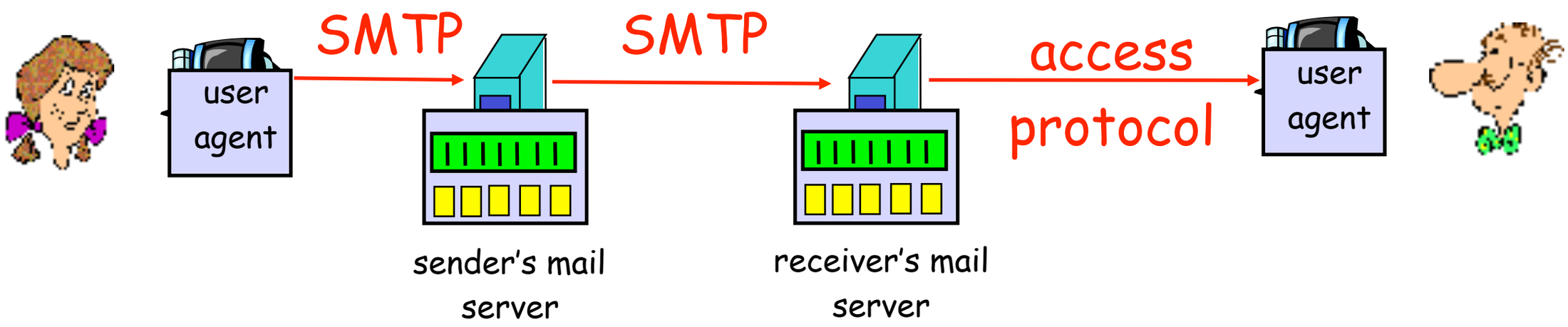
- SMTP: delivery/storage to receiver's server
- mail access protocol: retrieval from server
  - POP: Post Office Protocol [RFC 1939]
    - authorization (agent <-->server) and download
  - IMAP: Internet Mail Access Protocol [RFC 1730]
    - more features (more complex)
    - manipulation of stored msgs on server

Refining Mail Delivery



- SMTP: delivery/storage to receiver's server
- mail access protocol: retrieval from server
  - POP: Post Office Protocol [RFC 1939]
    - authorization (agent <-->server) and download
  - IMAP: Internet Mail Access Protocol [RFC 1730]
    - more features (more complex)
    - manipulation of stored msgs on server
  - HTTP: gmail, Hotmail, Yahoo! Mail, etc.

Refining Mail Delivery



- SMTP: delivery/storage to receiver's server
- mail access protocol: retrieval from server
  - POP: Post Office Protocol [RFC 1939]
    - authorization (agent <-->server) and download
  - IMAP: Internet Mail Access Protocol [RFC 1730]
    - more features (more complex)
    - manipulation of stored msgs on server
  - HTTP: gmail, Hotmail, Yahoo! Mail, etc.

POP3 protocol

POP3 protocol

authorization phase

- client commands:
  - **`user:`** declare username
  - **`pass:`** password
- server responses
  - **`+OK`**
  - **`-ERR`**

POP3 protocol

authorization phase

- client commands:
  - **`user:`** declare username
  - **`pass:`** password
- server responses
  - **`+OK`**
  - **`-ERR`**

```
S: +OK POP3 server ready
C: user bob
S: +OK
C: pass hungry
S: +OK user successfully logged on
```

POP3 protocol

authorization phase

- client commands:
  - **`user:`** declare username
  - **`pass:`** password
- server responses
  - **`+OK`**
  - **`-ERR`**

```
S: +OK POP3 server ready
C: user bob
S: +OK
C: pass hungry
S: +OK user successfully logged on
```

transaction phase, client:

- **`list:`** list message numbers
- **`retr:`** retrieve message by number
- **`dele:`** delete
- **`quit`**

POP3 protocol

authorization phase

- client commands:
  - **`user`**: declare username
  - **`pass`**: password
- server responses
  - **`+OK`**
  - **`-ERR`**

transaction phase, client:

- **`list`**: list message numbers
- **`retr`**: retrieve message by number
- **`dele`**: delete
- **`quit`**

```
S: +OK POP3 server ready
C: user bob
S: +OK
C: pass hungry
S: +OK user successfully logged on
```

```
C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: <message 1 contents>
S: .
C: dele 1
C: retr 2
S: <message 1 contents>
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```

POP3 (more) and IMAP

more about POP3

- previous example uses "download and delete" mode.
- Bob cannot re-read e-mail if he changes client
- "download-and-keep": copies of messages on different clients
- POP3 is stateless across sessions

POP3 (more) and IMAP

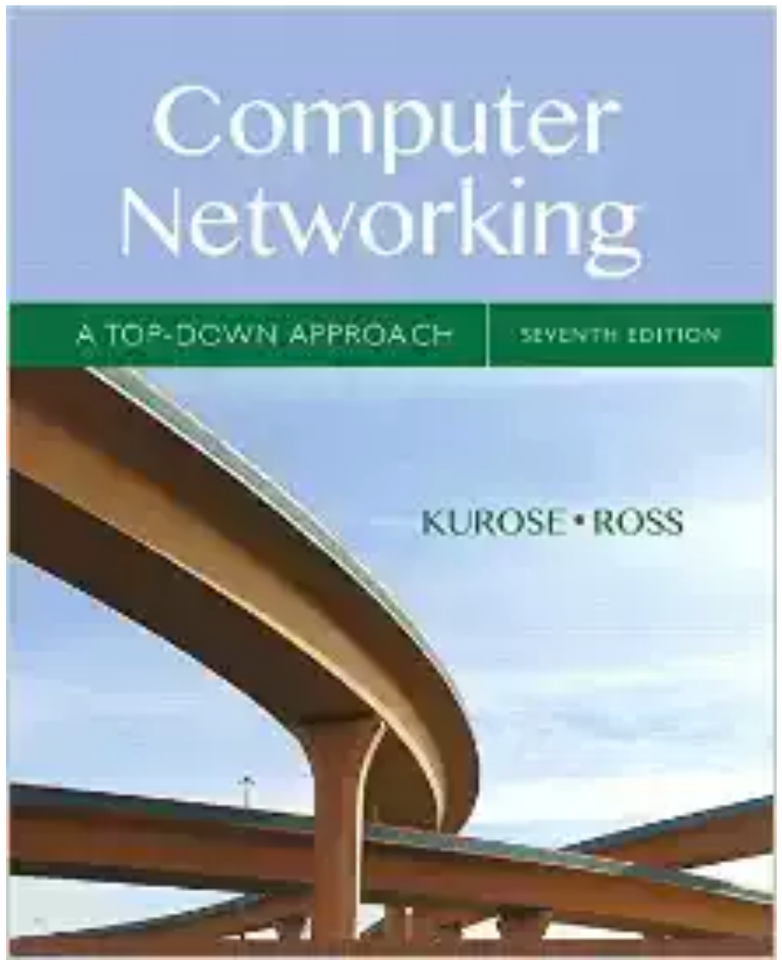
more about POP3

- previous example uses "download and delete" mode.
- Bob cannot re-read e-mail if he changes client
- "download-and-keep": copies of messages on different clients
- POP3 is stateless across sessions

IMAP

- keeps all messages in one place: at server
- allows user to organize messages in folders
- keeps user state across sessions:
  - names of folders and mappings between message IDs and folder name
- more complex!

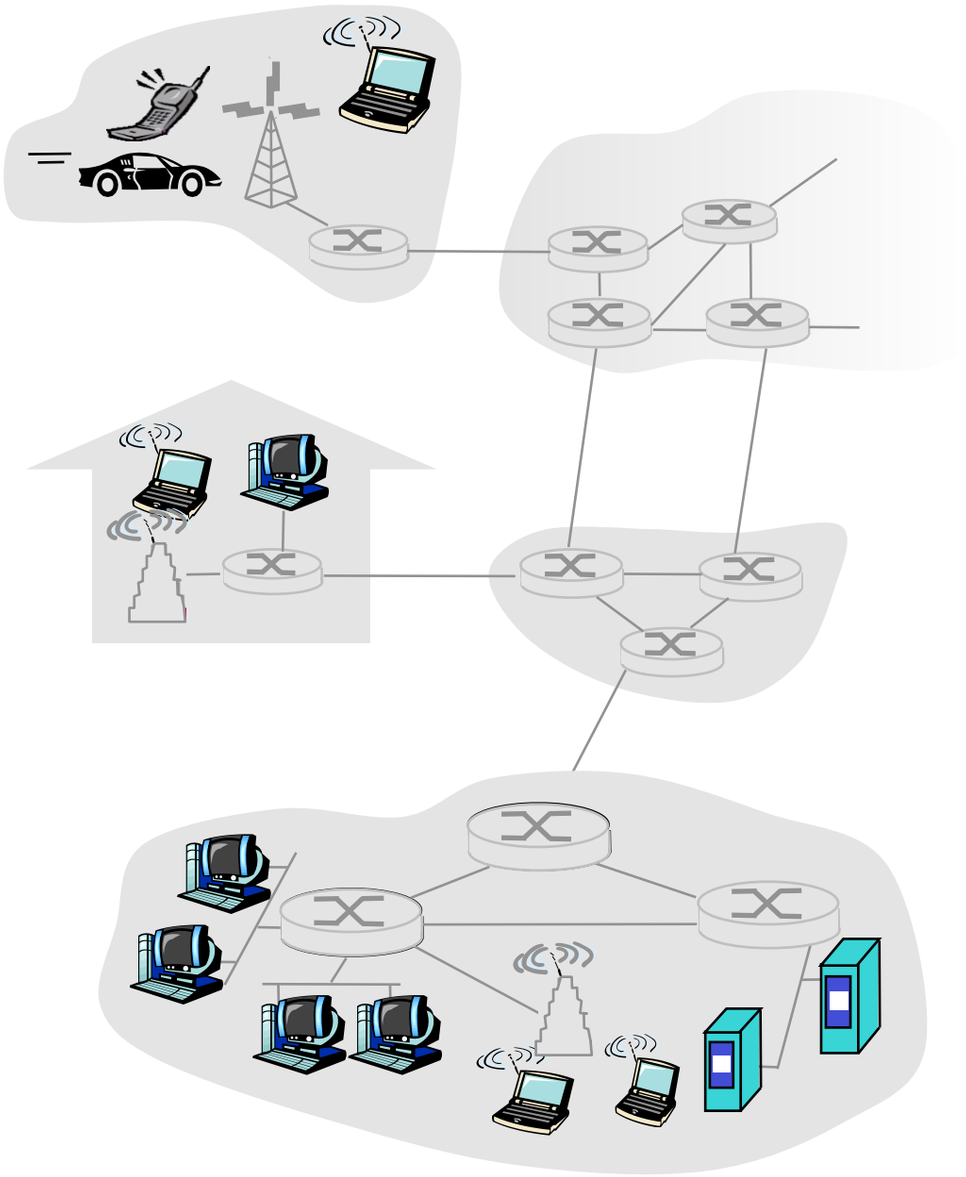
Reading Along ...



- 2.5 P2P Applications

Pure P2P architecture

- no always-on server
- arbitrary end systems directly communicate
- peers are intermittently connected and change IP addresses

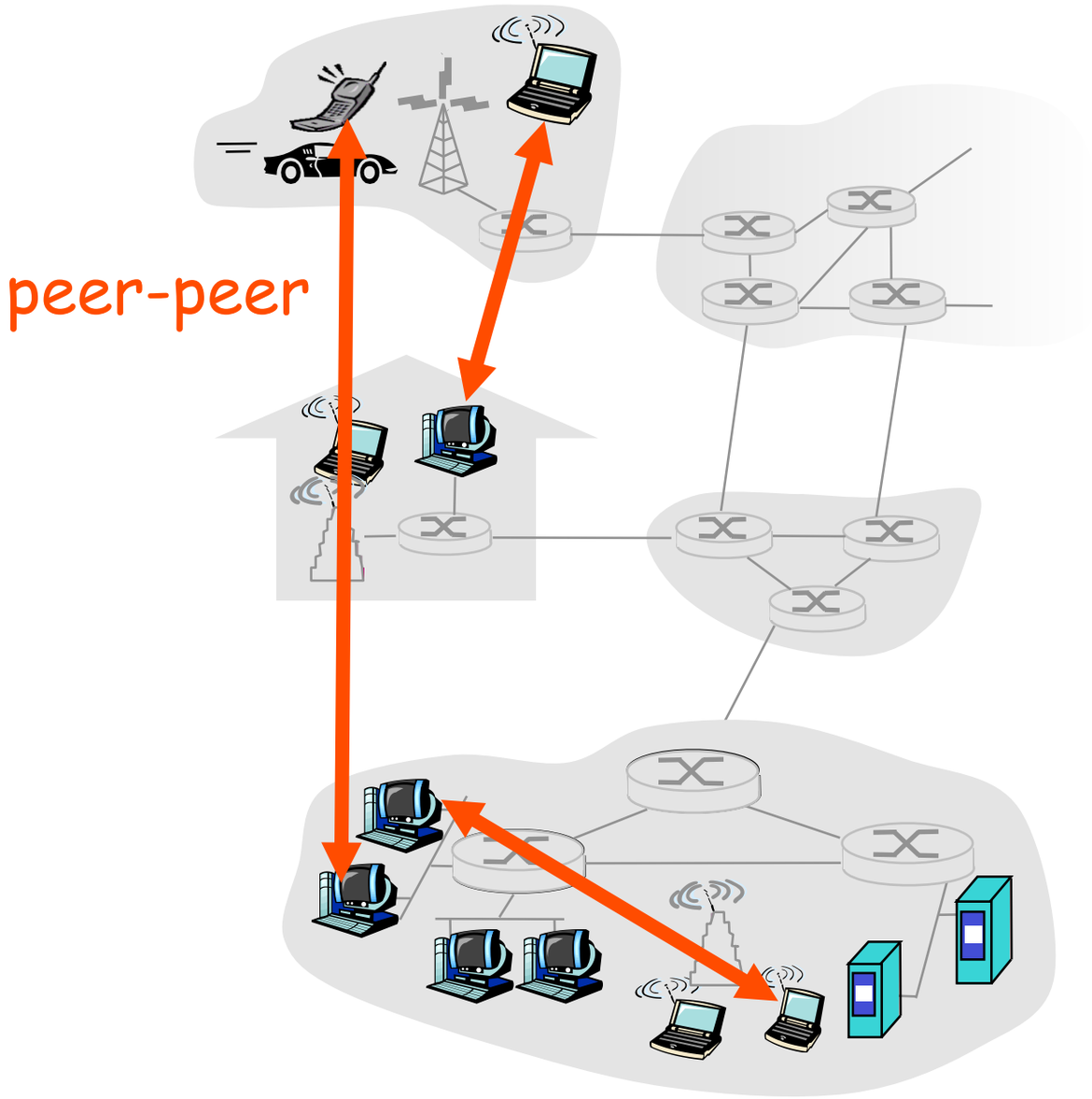


Three topics:

- file distribution
- searching for information
- case Study: Skype

Pure P2P architecture

- no always-on server
- arbitrary end systems directly communicate
- peers are intermittently connected and change IP addresses

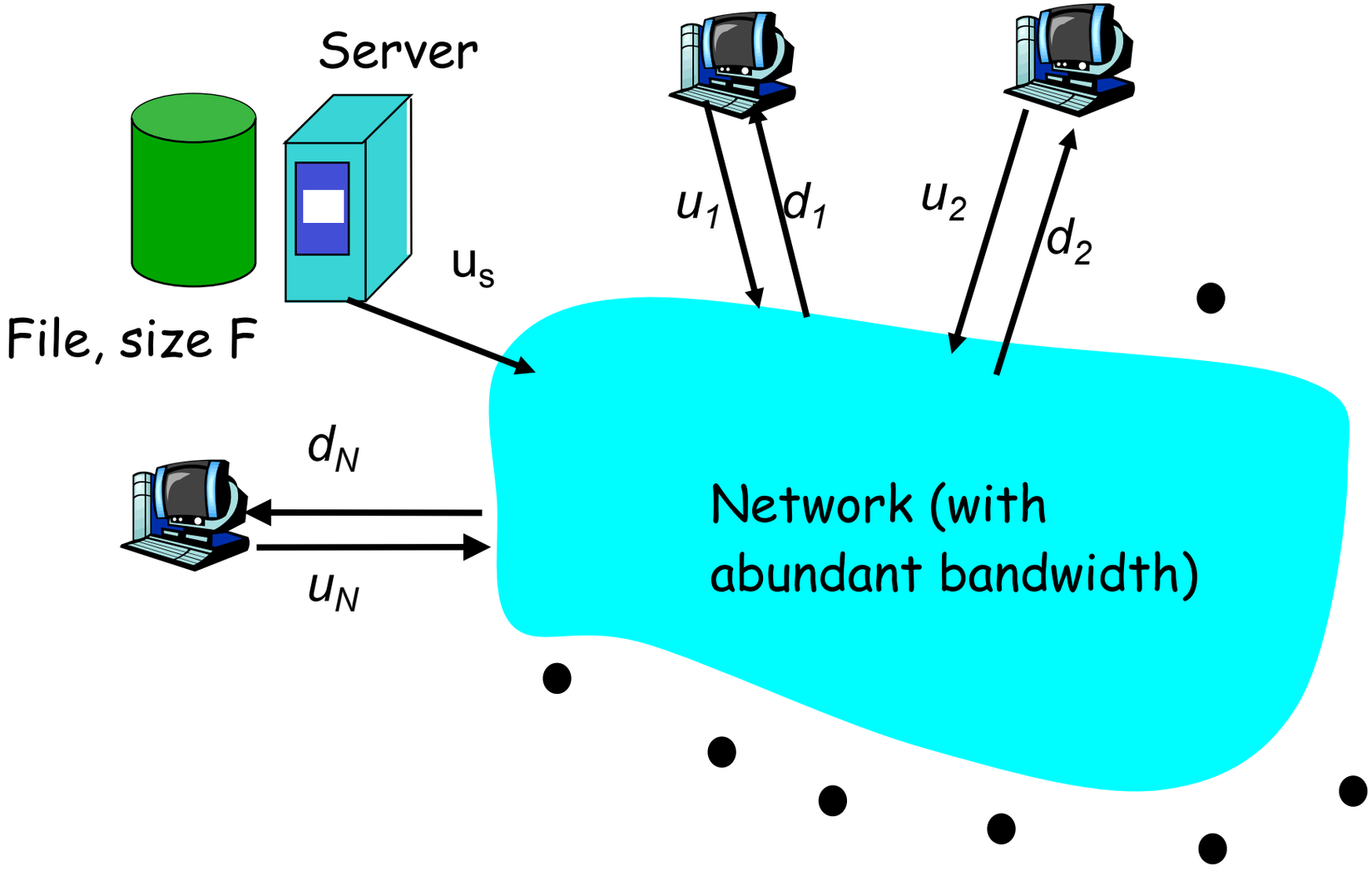


Three topics:

- file distribution
- searching for information
- case Study: Skype

File Distribution: Server-Client vs P2P

Question : How much time to distribute file of size F from one server to N peers?



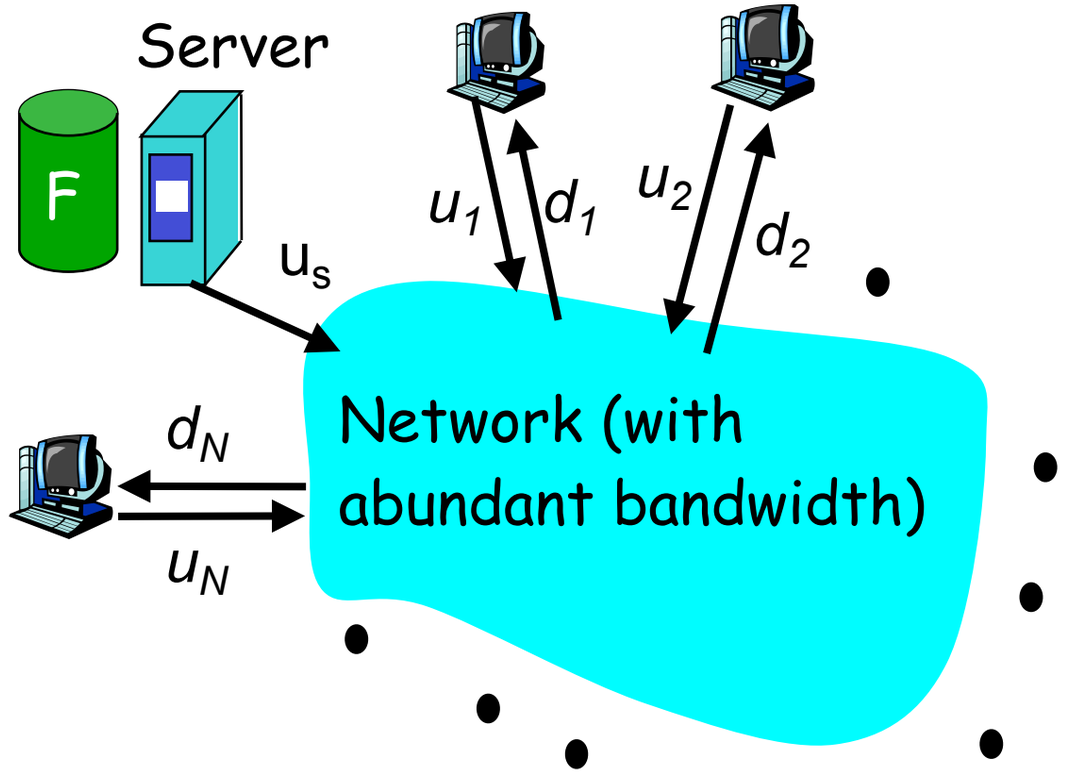
u_s: server upload bandwidth

u_i: peer i upload bandwidth

d_i: peer i download bandwidth

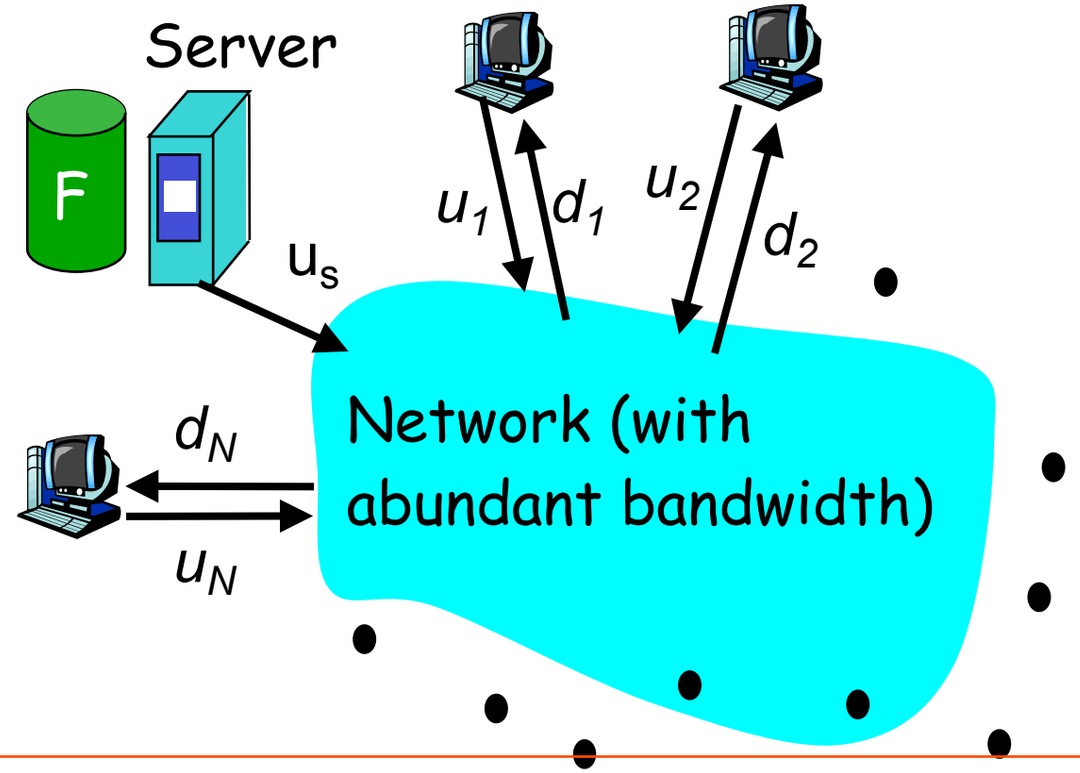
File distribution time: server-client

- server sequentially sends N copies:
  - NF/u_s time
- client i takes F/d_i time to download



File distribution time: server-client

- server sequentially sends N copies:
  - NF/u_s time
- client i takes F/d_i time to download

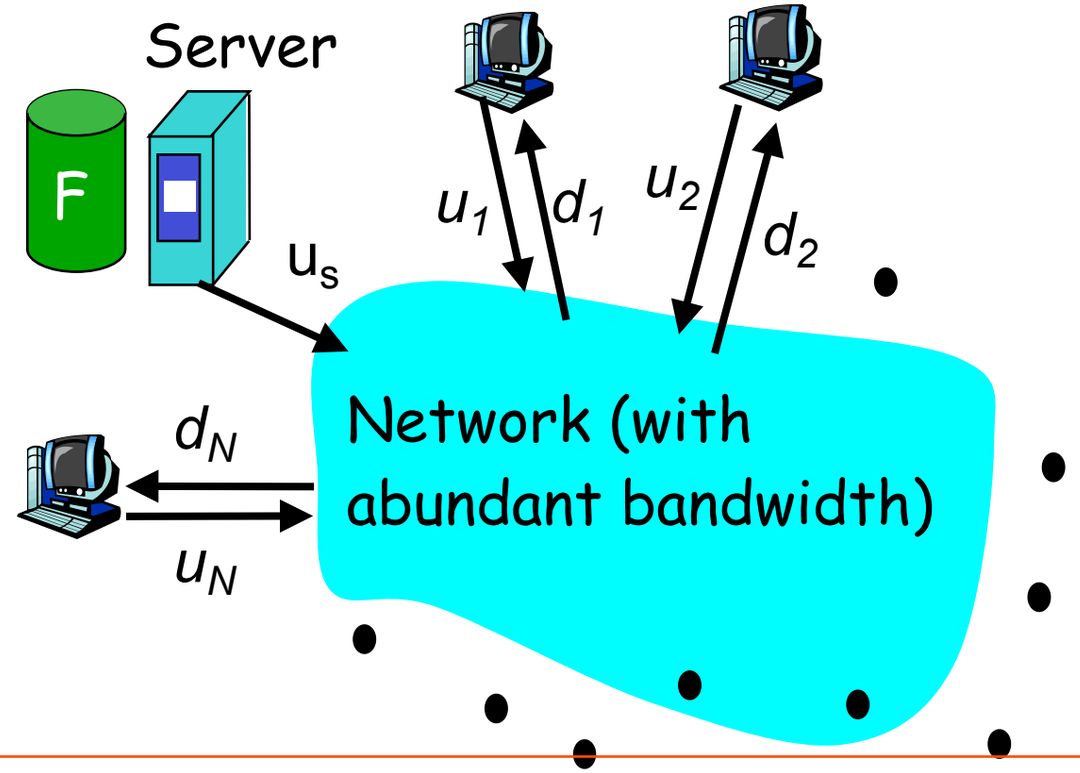


Time to distribute F to N clients using client/server approach

$= d_{cs} = \max \{ NF/u_s, F/\min(d_i) \}$

File distribution time: server-client

- server sequentially sends N copies:
  - NF/u_s time
- client i takes F/d_i time to download



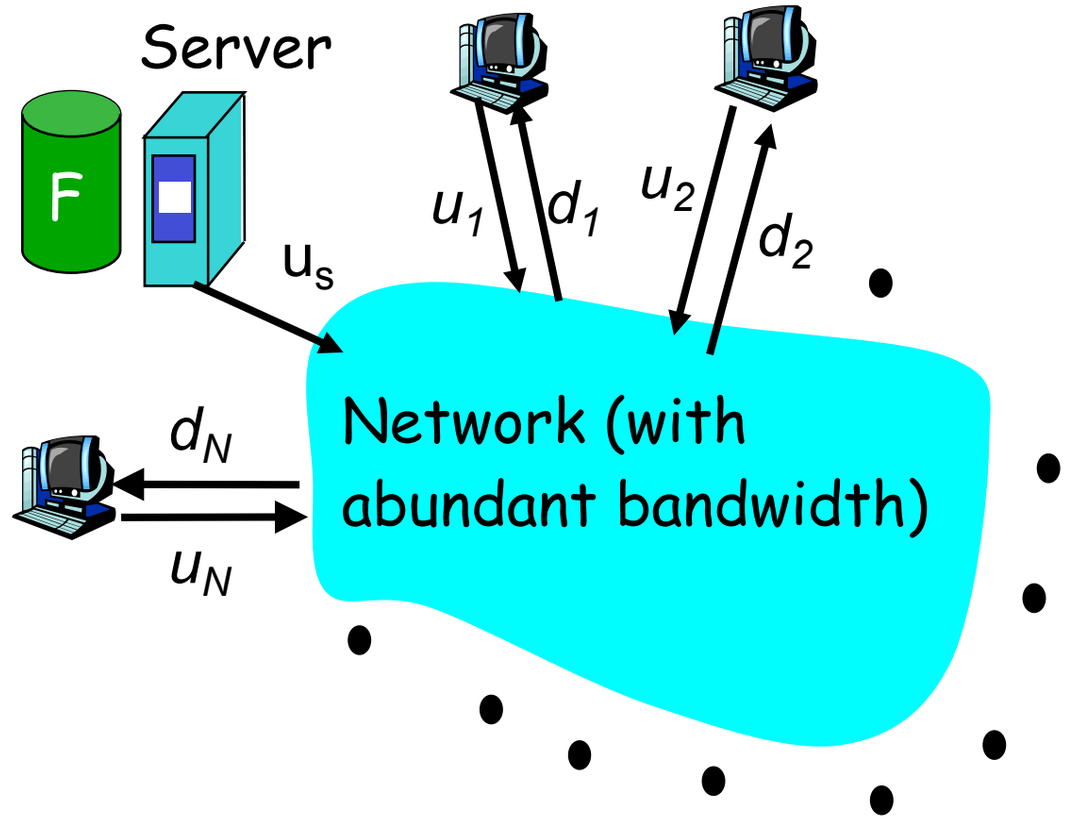
Time to distribute F to N clients using client/server approach

$= d_{cs} = \max \{ NF/u_s, F/\min(d_i) \}$

increases linearly in N (for large N)

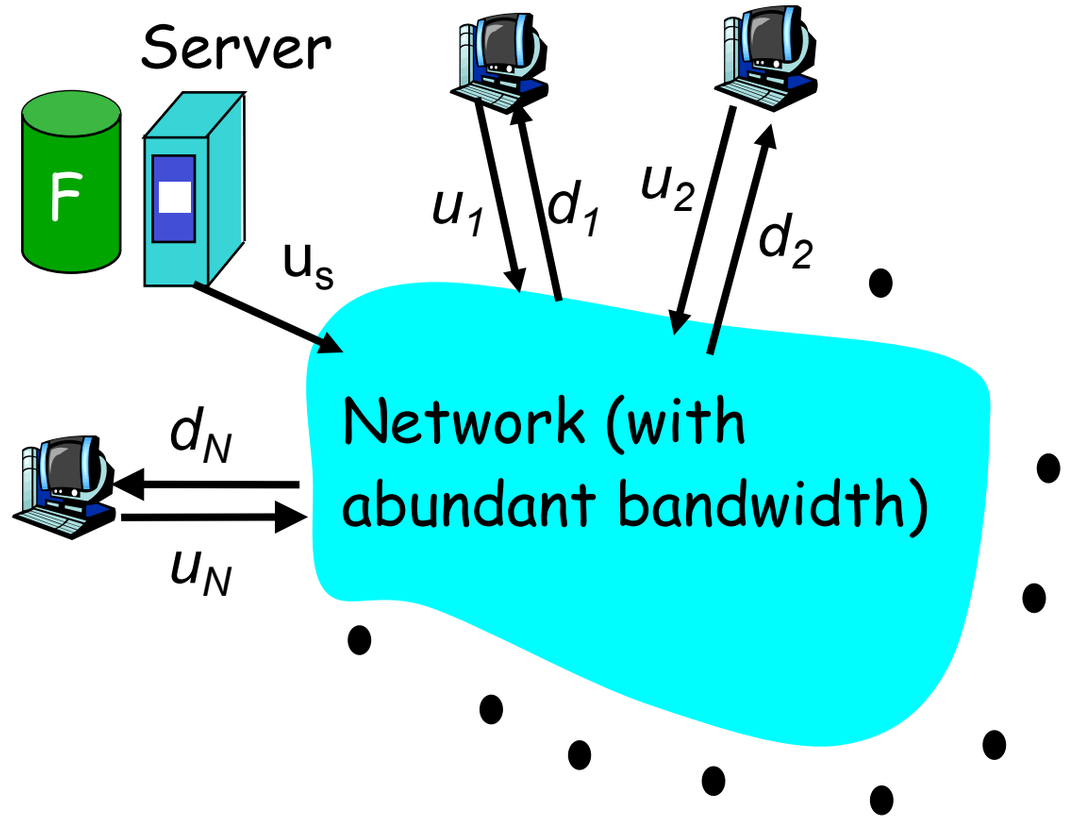
File distribution time: P2P

- server must send one copy: F/u_s time
- client i takes F/d_i time to download
- NF bits must be downloaded (aggregate)
  - fastest possible upload rate: $u_s + \Sigma u_i$



File distribution time: P2P

- server must send one copy: F/u_s time
- client i takes F/d_i time to download
- NF bits must be downloaded (aggregate)
  - fastest possible upload rate: $u_s + \Sigma u_i$



$$d_{P2P} = \max \{ F/u_s, F/\min(d_i) , NF/(u_s + \Sigma u_i) \}$$

Server-client vs. P2P: example

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