CS3103: Computer Networks Practice

Network Applications & Demo

Principles of Network Applications

HTTP & WWW

E-mail Protocols (SMTP, POP, IMAP)

P2P Applications

Dr. Anand Bhojan
COM3-02-49, School of Computing
banand@comp.nus.edu.sg ph: 651-67351

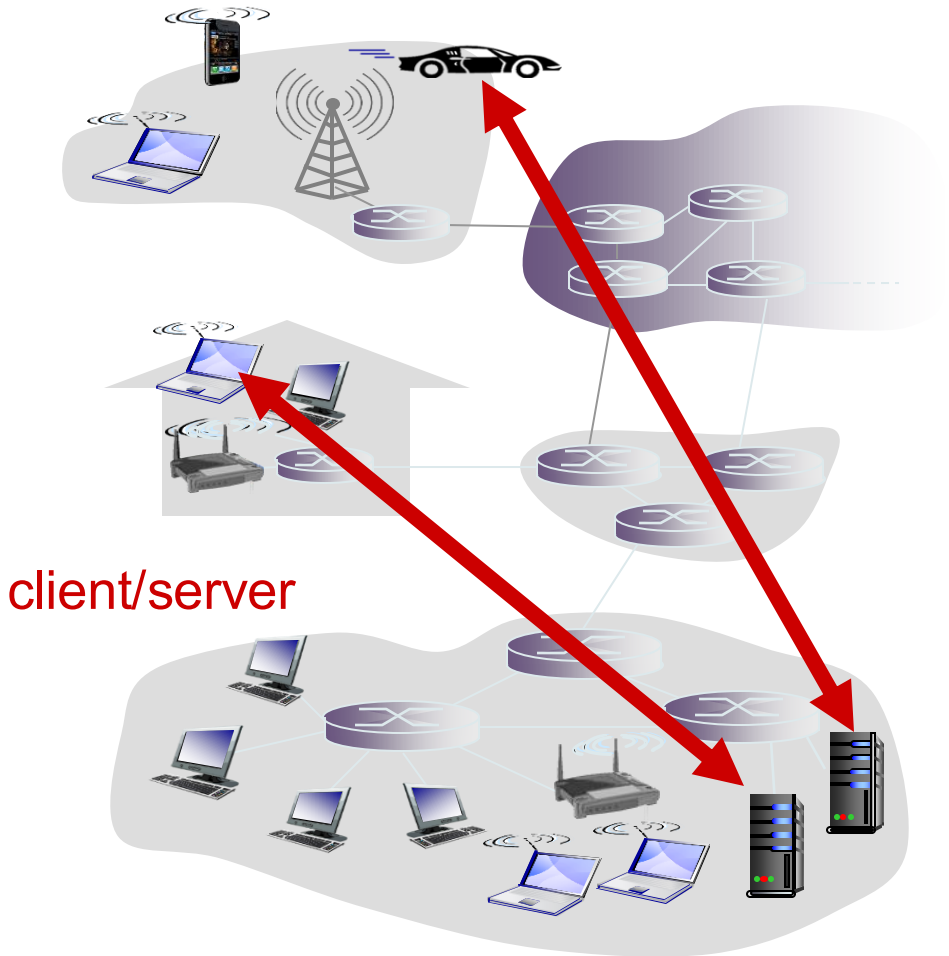
https://pollev.com/banand

Make sure you
LOGIN using
your
NUSNET ID.

possible structure of applications:

- client-server
- peer-to-peer (P2P)

Client-server architecture



Which of the following are common properties of a server?

- always-on host
- uses ephemeral ports client
- uses permanent IP address
- initiates communication client
- uses well-known ports
- waits to be contacted

P2P architecture

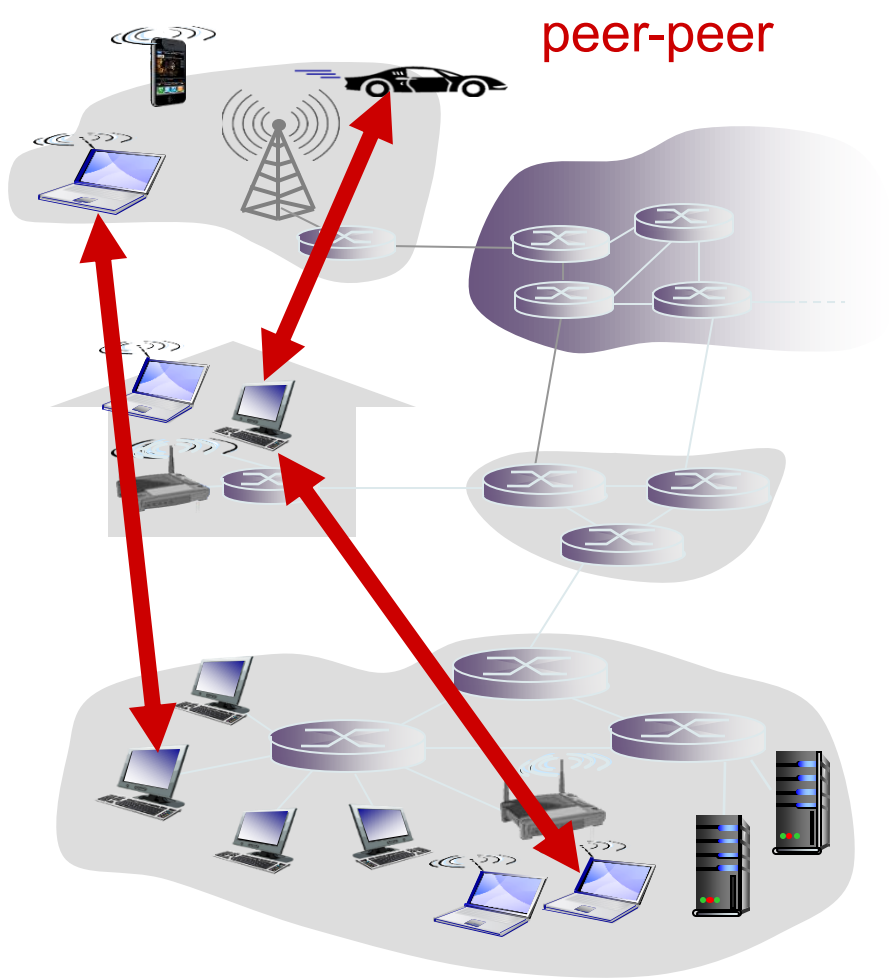
- *no* always-on server
- arbitrary end systems directly communicate and serve each other.
  - Each host runs both server and client processes

Q1: What is the advantage of P2P?

no need to set up

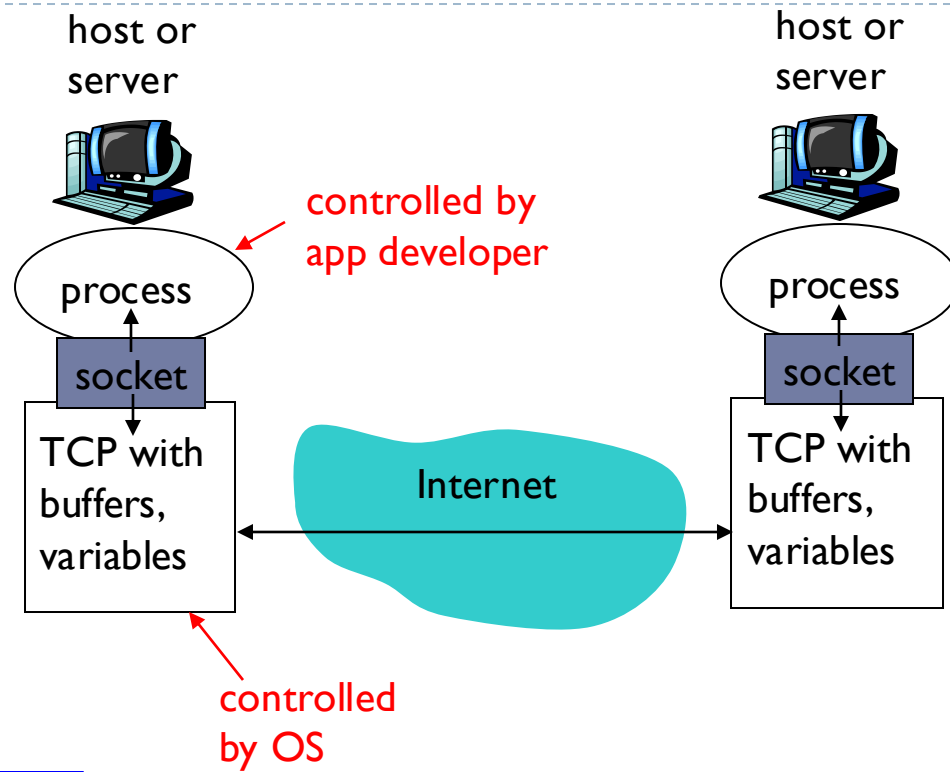
Q2: How the hosts find the IP addresses of each other?

Will discuss in next Section…



Socket

- Socket
  - Interface between the application layer and transport layer
  - OS-controlled interface (a "door") into which application process can both send and receive messages
- Addressing
  - Host address + process identifier
  - Eg. IP address + port number
    - Eg HTTP – port 80, SMTP – port 25, ftp – port 21



Tool:
- Use *ifconfig* (*ipconfig* in windows) to see edit your laptop's IP address.
- Use '*cat /etc/services*' to see all well known port numbers.
- Use '*netstat –a*' to show all active connections.
- Use '*lsof -i –n*' to show the COMMAND, PID (process ID), and USER (user ID) of open Internet sockets.

Socket: endpoint for comm. A combination of host IP address and port number.

Internet transport protocols services

TCP service:

- *reliable transport* between sending and receiving process
- *flow control:* sender won't overwhelm receiver
- *congestion control:* throttle sender when network overloaded
- *does not provide:* timing, minimum throughput guarantee, security
- *connection-oriented:* setup required between client and server processes, maintains *connection state*

UDP service:

- *unreliable data transfer* between sending and receiving process
- *does not provide:* reliability, flow control, congestion control, timing, throughput guarantee, security, or connection setup

udp header to indentify application, have port number

Q: why bother? Why is there a UDP?

Securing TCP

TCP & UDP

- no encryption
- cleartext passwords sent into socket, traverse Internet in cleartext

SSL

- provides encrypted TCP connection
- data integrity
- end-point authentication

SSL is at app layer

- Apps use SSL libraries, which "talk" to TCP

SSL socket API

- cleartext passwords sent into socket, traverse Internet encrypted

Network Applications & Demo

Principles of Network Applications

HTTP & WWW

E-mail Protocols (SMTP, POP, IMAP)

P2P Applications

Dr. Anand Bhojan
COM3-02-49, School of Computing
banand@comp.nus.edu.sg ph: 651-67351

HTTP

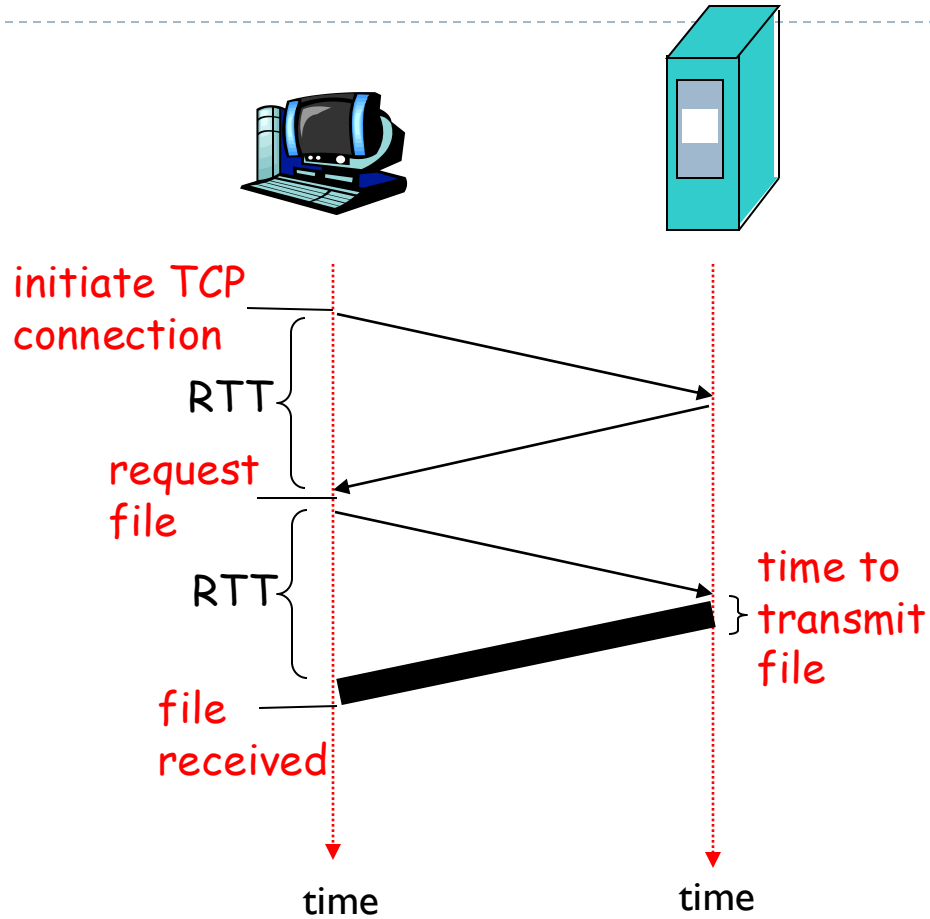
- Uses **TCP Service.**
- HTTP is **stateless** (Note:- TCP maintains connection state, application layer is not maintaining the application/user state)
  - server maintains no information about past client requests
  - if server/client crashes, their views of "state" may be inconsistent, must be reconciled
- Default HTTP Server **port: 80** [HTTPS – port 443]

- HTTP 1.0 (RFC 1945)
- HTTP 1.1 (RFC 2616)
- HTTP 2 (RFC 7540, RFC 9113)
- HTTP 3 (RFC 9114)

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

HTTP

- **Persistence vs non-persistence**
  - **Non-persistence**
    - Server terminates connection after transferring the file/object
    - HTTP 1.0 (RFC 1945)
    - New connection is established each time (new set of TCP variables and buffers)
    - browsers often open parallel TCP connections to fetch referenced objects



✶ **Response time** (Non-persistence)
- one RTT (Round Trip Time) to initiate TCP connection
- one RTT for HTTP request and first few bytes of HTTP response to return
- file transmission time

total = 2RTT+transmit time

RTT- Time taken for a small packet from client to server and then back to the client.

HTTP

- **Persistence**:
  - Server leaves the TCP connection open after sending the response. Subsequent requests (referenced objects) between the same client and server will use the same connection.
  - **Without pipelining** (one RTT for each referenced object)
  - **With pipelining** (as little as one RTT for all the referenced objects)
  - Default in HTTP 1.1 (RFC 2616): persistence with pipelining

Without Pipelining (one RTT per referenced object)

Client (left, blue) ↔ Server (right, green)

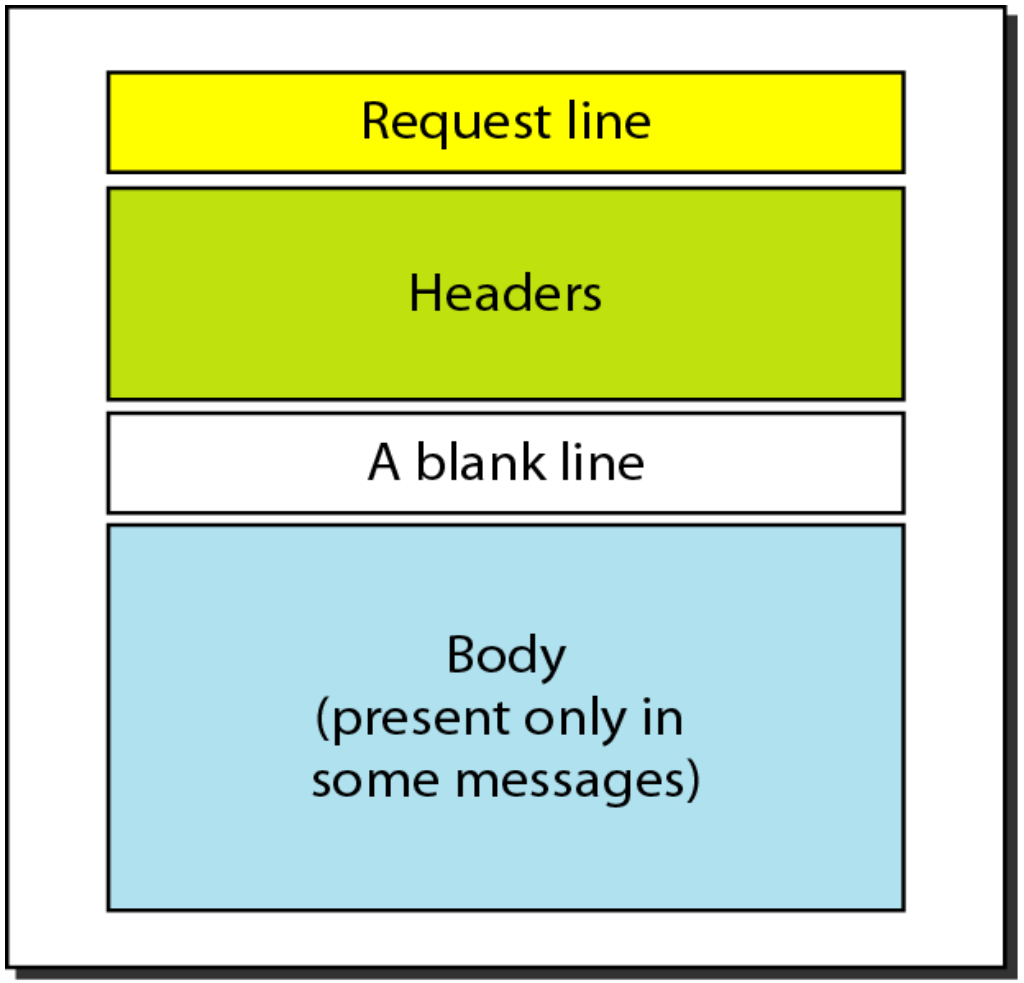
With Pipelining (HTML first, then 4 objects in 1 RTT)

Client (left, blue) ↔ Server (right, green)

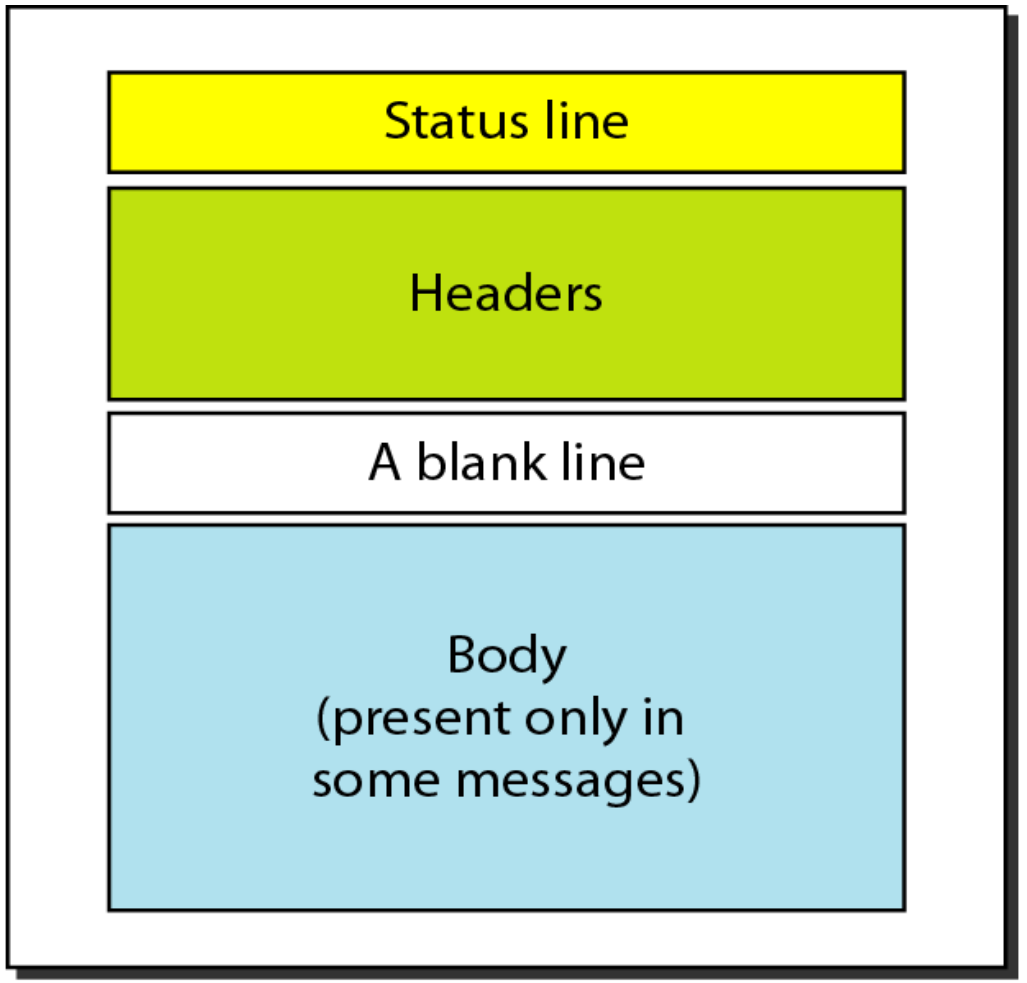
Legend: Request lines are thin; response lines are thicker. Batched response thickness equals the sum of the independent object lines (e.g., 4 objects × 8px each → 32px).
Sides: Client bubbles are blue (left), Server bubbles are green (right). TCP connection is kept open and reused for subsequent messages.
RTT: Highlighted in red inside brackets.

SENDS html file first,
then the objects, if pipelining, sends all at once,
if not then one at at ime

HTTP Message Format

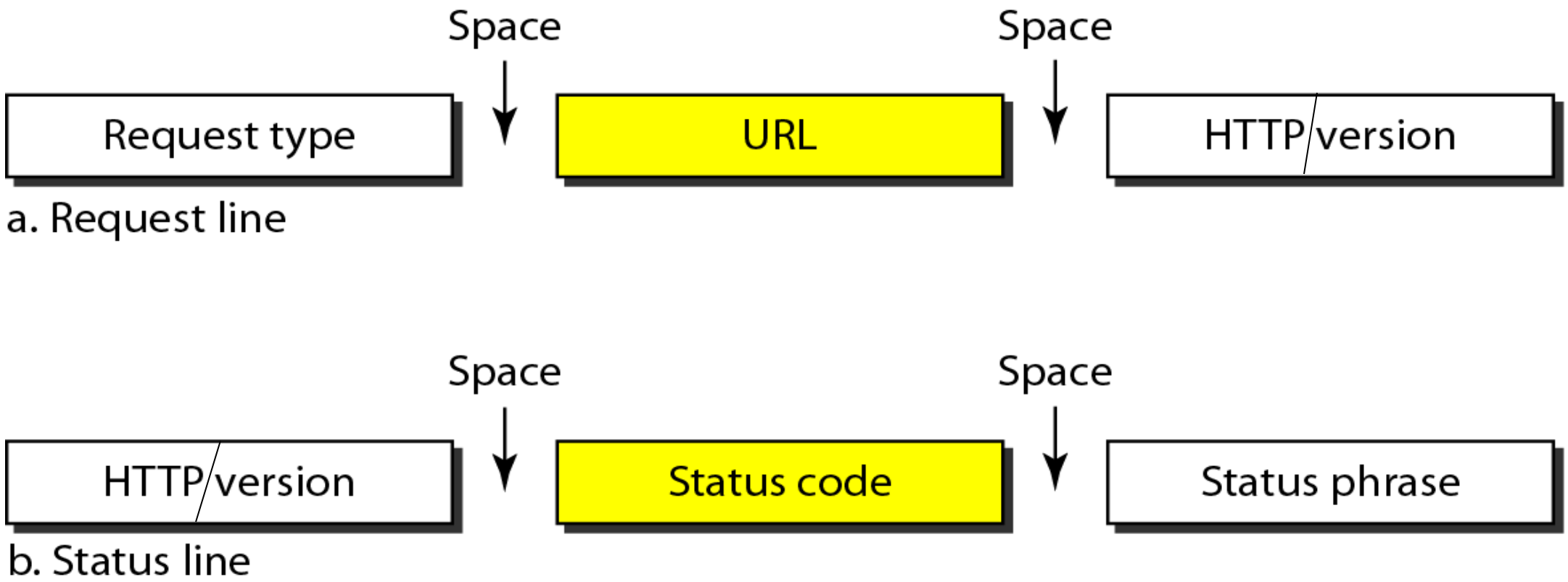


Request message



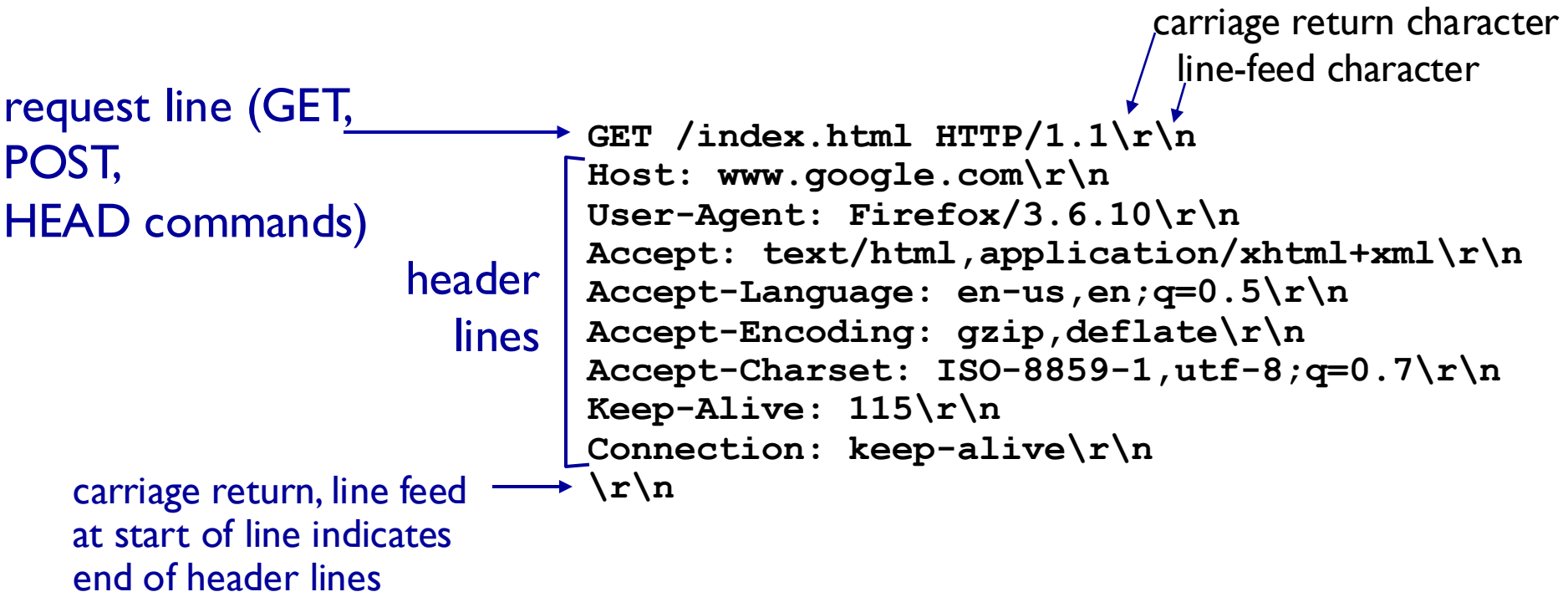
Response message

HTTP - Request line and Response status line



HTTP request message

- HTTP request message:
  - ASCII (human-readable format)



Other HTTP request messages

POST method:

- web page often includes form input
- user input sent from client to server in entity body of HTTP POST request message

GET method (for sending data to server):

- include user data in URL field of HTTP GET request message (following a '?'):

`www.somesite.com/animalsearch?monkeys&banana`

PUT method:

- uploads new file (object) to server
- completely replaces file that exists at specified URL with content in entity body of POST HTTP request message

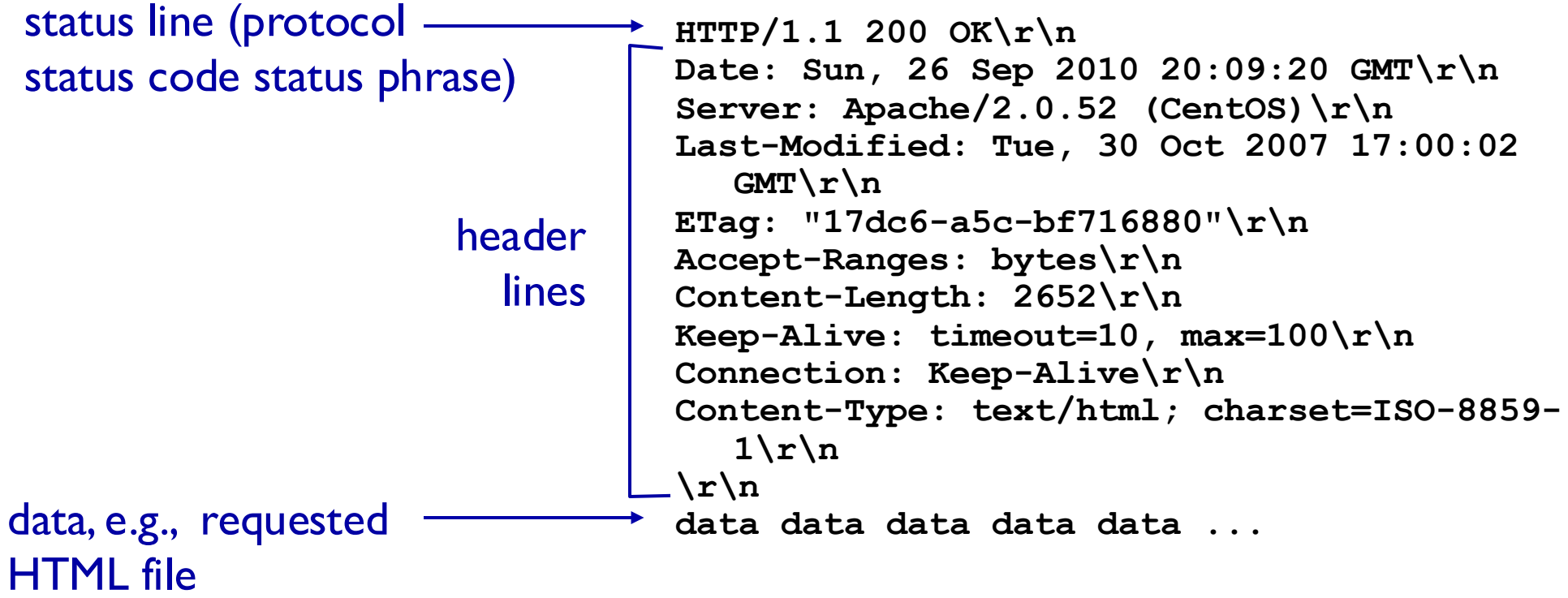
DELETE method:

- Delete specified object (file) in the server

HEAD method:

- requests headers (only) that would be returned *if* specified URL were requested with an HTTP GET method.

HTTP response message



HTTP response status codes

- status code appears in 1st line in server-to-client response message.
- some sample codes:

200 OK
- request succeeded, requested object later in this message

301 Moved Permanently
- requested object moved, new location specified later in this message (in Location: field)

400 Bad Request
- request msg not understood by server

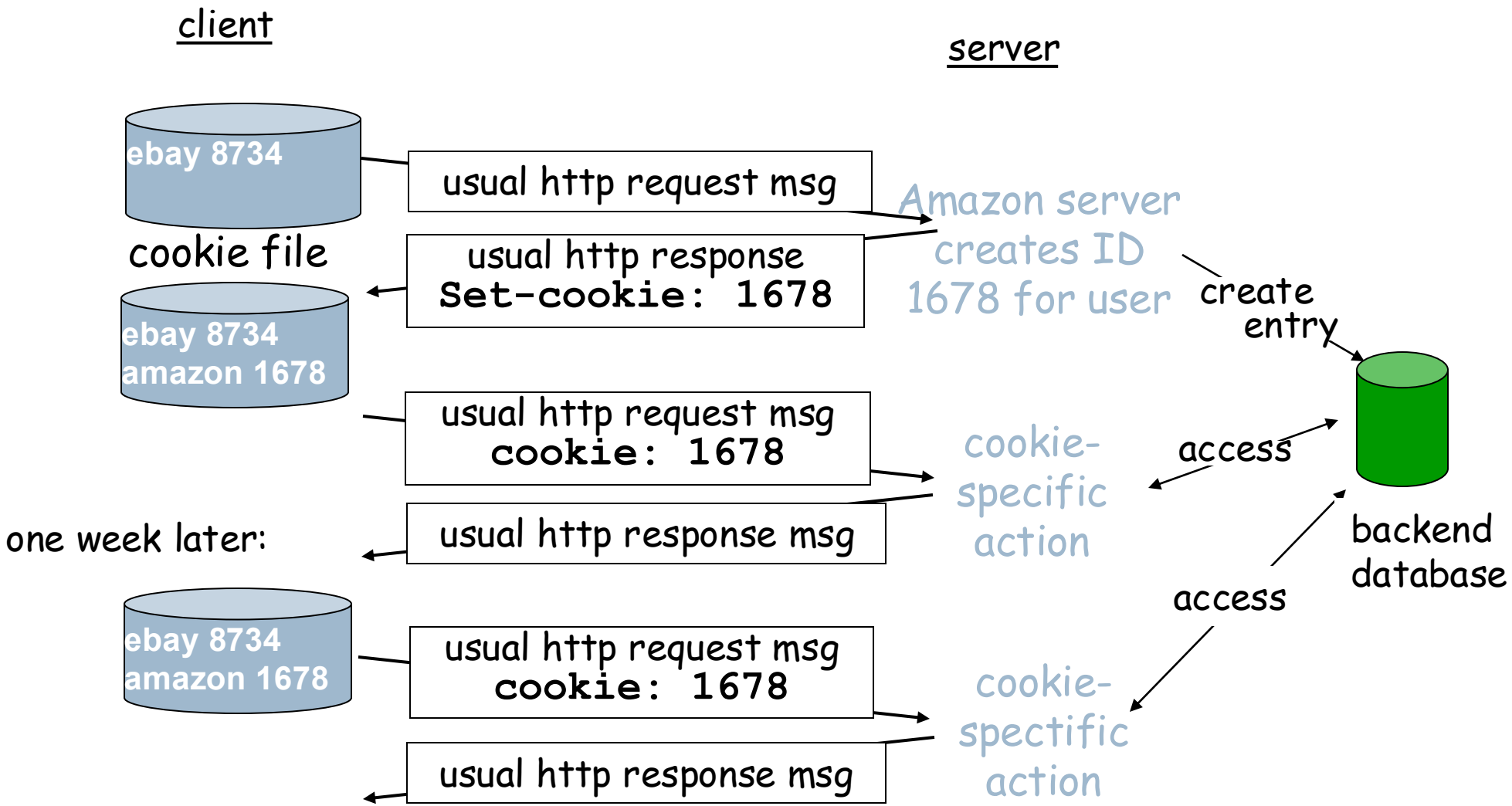
404 Not Found
- requested document not found on this server

505 HTTP Version Not Supported

HTTP - User State Management : Cookies

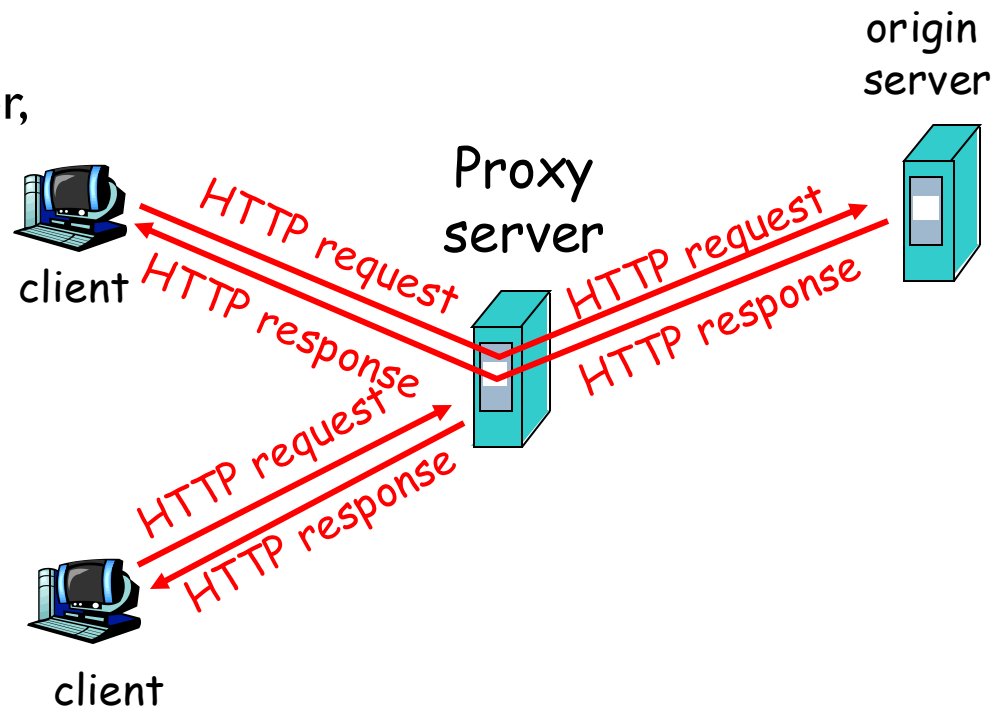
- HTTP server is stateless.
- HTTP uses cookies to manage user identity and state
- Cookies have four components.
  - HTTP response header. (by web server upon initial request from client)
    - Set-cookie: 1232341
  - HTTP request header. (every subsequent request by client)
    - Cookie: 1232341
  - Cookie file is maintained by the browser (client)
  - Back-end database at the web site (web server)

HTTP - User State Management : Cookies



HTTP - Web caches (proxy server)

- **Goal:** satisfy client request without involving origin server. Improve response time; reduce traffic.
  - user sets browser: Web accesses via cache
  - browser sends all HTTP requests to cache
    - object in cache: cache returns object
    - else cache requests object from origin server, then returns object to client



Conditional GET

Goal: don't send object if cache has up-to-date cached version

- no object transmission delay
- lower link utilization

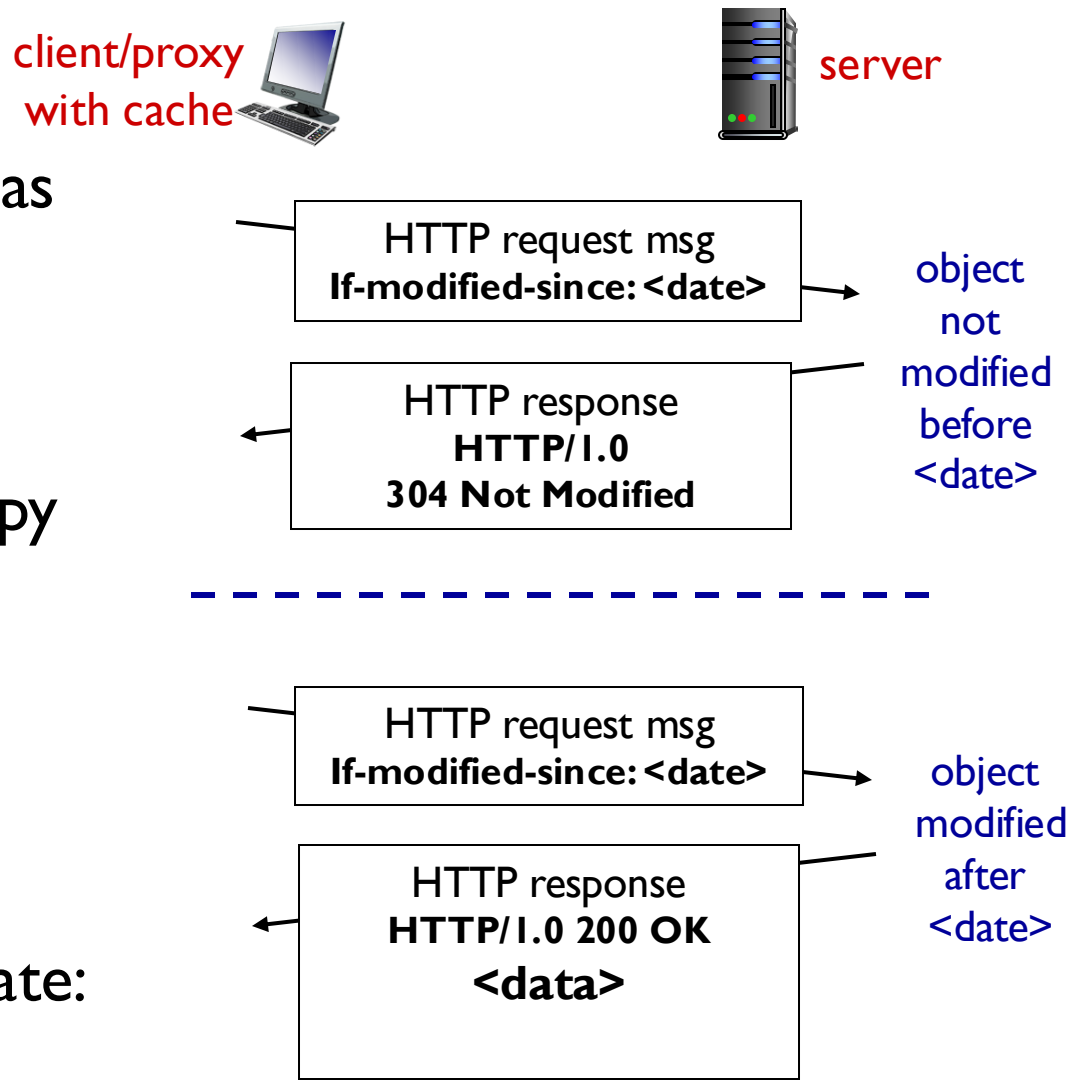
- *cache:* specify date of cached copy in HTTP request

  If-modified-since: <date>

  <date> - taken from **Last-modified:**

- *server:* response contains no object if cached copy is up-to-date:

  HTTP/1.0 304 Not Modified



HTTP demo

bhojan — nc -v example.org 80 — 104×55

```
(base) bhojan@adminsocs-MacBook-Pro-9 ~ % nc -v example.org  80
Connection to example.org port 80 [tcp/http] succeeded!
GET / HTTP/1.1
Host: example.org

HTTP/1.1 200 OK
Accept-Ranges: bytes
Age: 218964
Cache-Control: max-age=604800
Content-Type: text/html; charset=UTF-8
Date: Mon, 04 Sep 2023 17:46:05 GMT
Etag: "3147526947"
Expires: Mon, 11 Sep 2023 17:46:05 GMT
Last-Modified: Thu, 17 Oct 2019 07:18:26 GMT
Server: ECS (oxr/830D)
Vary: Accept-Encoding
X-Cache: HIT
Content-Length: 1256

<!doctype html>
<html>
<head>
    <title>Example Domain</title>

    <meta charset="utf-8" />
    <meta http-equiv="Content-type" content="text/html; charset=utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <style type="text/css">
    body {
        background-color: #f0f0f2;
        margin: 0;
```

-v : verbose

bhojan — -bash — 68×27

```
BadduBlueMountain:~ bhojan$ telnet arivu.comp.nus.edu.sg 80
Trying 137.132.83.224...
Connected to arivu.d2.comp.nus.edu.sg.
Escape character is '^]'.
GET /index.html HTTP/1.1
Host: arivu.comp.nus.edu.sg

HTTP/1.1 200 OK
Date: Wed, 21 Jun 2017 07:03:14 GMT
Server: Apache/2.4.7 (Ubuntu)
Last-Modified: Thu, 22 Mar 2012 12:25:22 GMT
ETag: "b8-4bbd40082d39a"
Accept-Ranges: bytes
Content-Length: 184
Vary: Accept-Encoding
Content-Type: text/html

<html>
<head>
<title>ARIVU</title>
</head>
<body><h1>Hi, You are in ARIVU Server @ SoC</h1>
<p>It runs multiple services</p>
<p>Pls point to the appropriate service</p>
</body></html>
Connection closed by foreign host.
BadduBlueMountain:~ bhojan$
```

HTTP demo

curl -v -l https://varlabs.comp.nus.edu.sg/tools/index.html

bhojan — -zsh — 104×55

```
(base) bhojan@adminsocs-MacBook-Pro-9 ~ % curl -v -L  https://varlabs.comp.nus.edu.sg/tools/index.html
*   Trying 137.132.84.180:443...
* Connected to varlabs.comp.nus.edu.sg (137.132.84.180) port 443 (#0)
* ALPN, offering http/1.1
* successfully set certificate verify locations:
*   CAfile: /opt/anaconda3/ssl/cacert.pem
  CApath: none
* TLSv1.3 (OUT), TLS handshake, Client hello (1):
* TLSv1.3 (IN), TLS handshake, Server hello (2):
* TLSv1.3 (OUT), TLS change cipher, Change cipher spec (1):
* TLSv1.3 (OUT), TLS handshake, Client hello (1):
* TLSv1.3 (IN), TLS handshake, Server hello (2):
* TLSv1.3 (IN), TLS handshake, Encrypted Extensions (8):
* TLSv1.3 (IN), TLS handshake, Certificate (11):
* TLSv1.3 (IN), TLS handshake, CERT verify (15):
* TLSv1.3 (IN), TLS handshake, Finished (20):
* TLSv1.3 (OUT), TLS handshake, Finished (20):
* SSL connection using TLSv1.3 / TLS_AES_256_GCM_SHA384
* ALPN, server accepted to use http/1.1
* Server certificate:
*  subject: CN=*.comp.nus.edu.sg
*  start date: Nov 21 00:00:00 2022 GMT
*  expire date: Dec 22 23:59:59 2023 GMT
*  subjectAltName: host "varlabs.comp.nus.edu.sg" matched cert's "*.comp.nus.edu.sg"
*  issuer: C=GB; ST=Greater Manchester; L=Salford; O=Sectigo Limited; CN=Sectigo RSA Domain Validation S
ecure Server CA
*  SSL certificate verify ok.
> GET /tools/index.html HTTP/1.1
> Host: varlabs.comp.nus.edu.sg
> User-Agent: curl/7.71.1
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Server: nginx/1.10.2
< Date: Mon, 04 Sep 2023 17:32:17 GMT
< Content-Type: text/html
< Content-Length: 185
```

Alternatively, you can use 'wget'

-v : verbose
-l : follow redirects

HTTP/2

Key goal: decreased delay in multi-object HTTP requests

HTTP1.1: introduced multiple, pipelined GETs over single TCP connection

- server responds *in-order* (FCFS: first-come-first-served scheduling) to GET requests
- with FCFS, small object may have to wait for transmission (head-of-line (HOL) blocking) behind large object(s)
- loss recovery (retransmitting lost TCP segments) stalls object transmission

HTTP/2

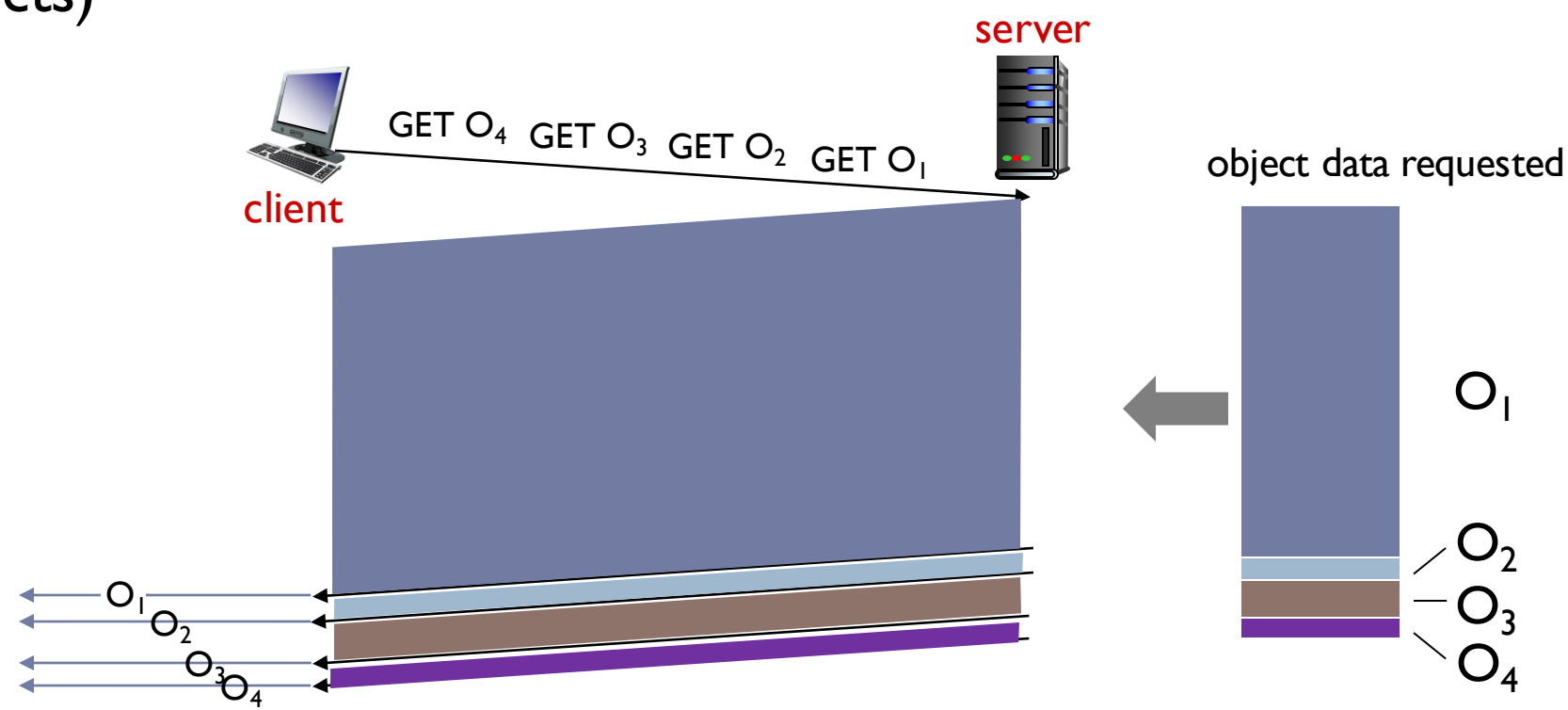
Key goal: decreased delay in multi-object HTTP requests

HTTP/2: [RFC 7540, 2015] increased flexibility at *server* in sending objects to client:

- methods, status codes, most header fields unchanged from HTTP 1.1
- transmission order of requested objects based on client-specified object priority (not necessarily FCFS)
- *push* unrequested objects to client
- divide objects into frames, schedule frames to mitigate HOL blocking

HTTP/2: mitigating HOL blocking

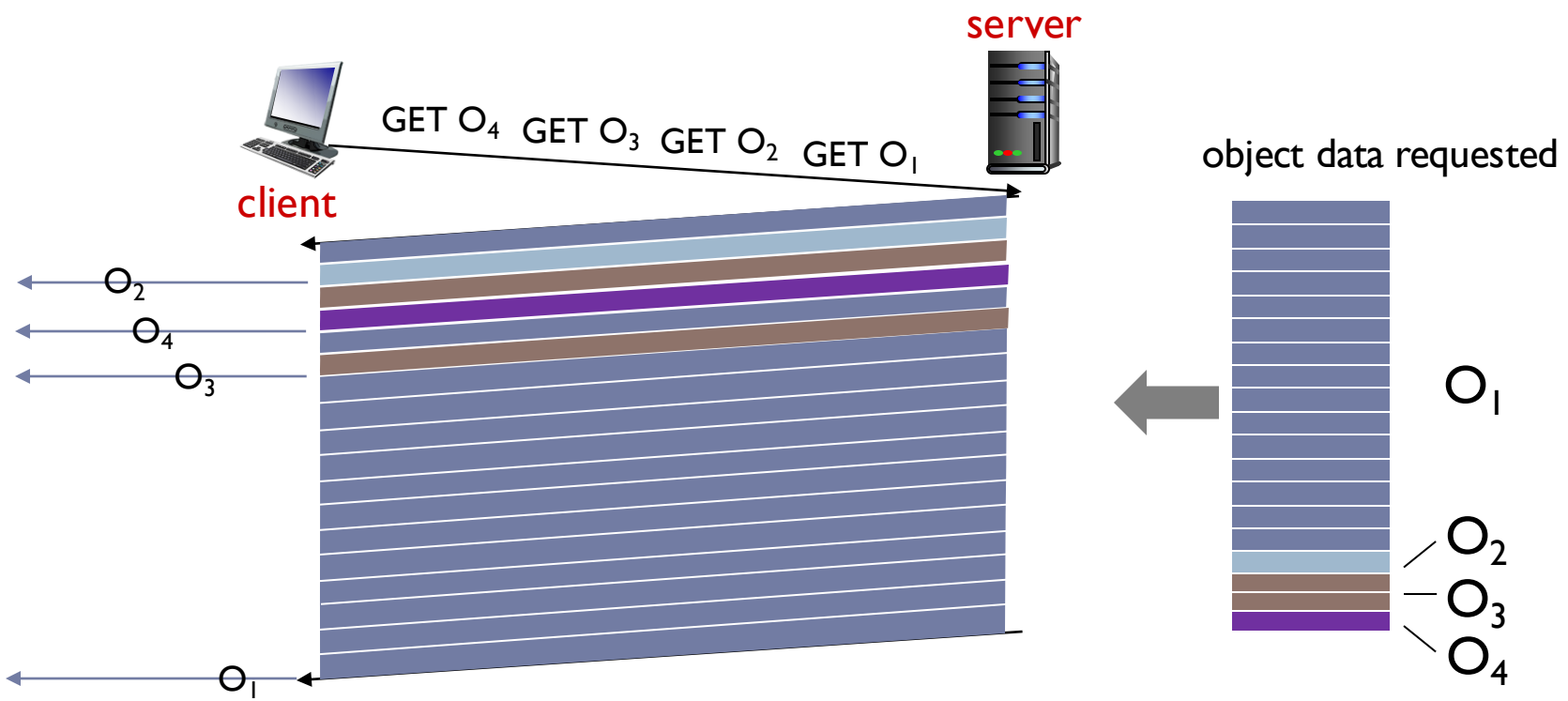
HTTP 1.1: client requests 1 large object (e.g., video file, and 3 smaller objects)



objects delivered in order requested: O_2, O_3, O_4 wait behind O_1

HTTP/2: mitigating HOL blocking

HTTP/2: objects divided into frames, frame transmission interleaved



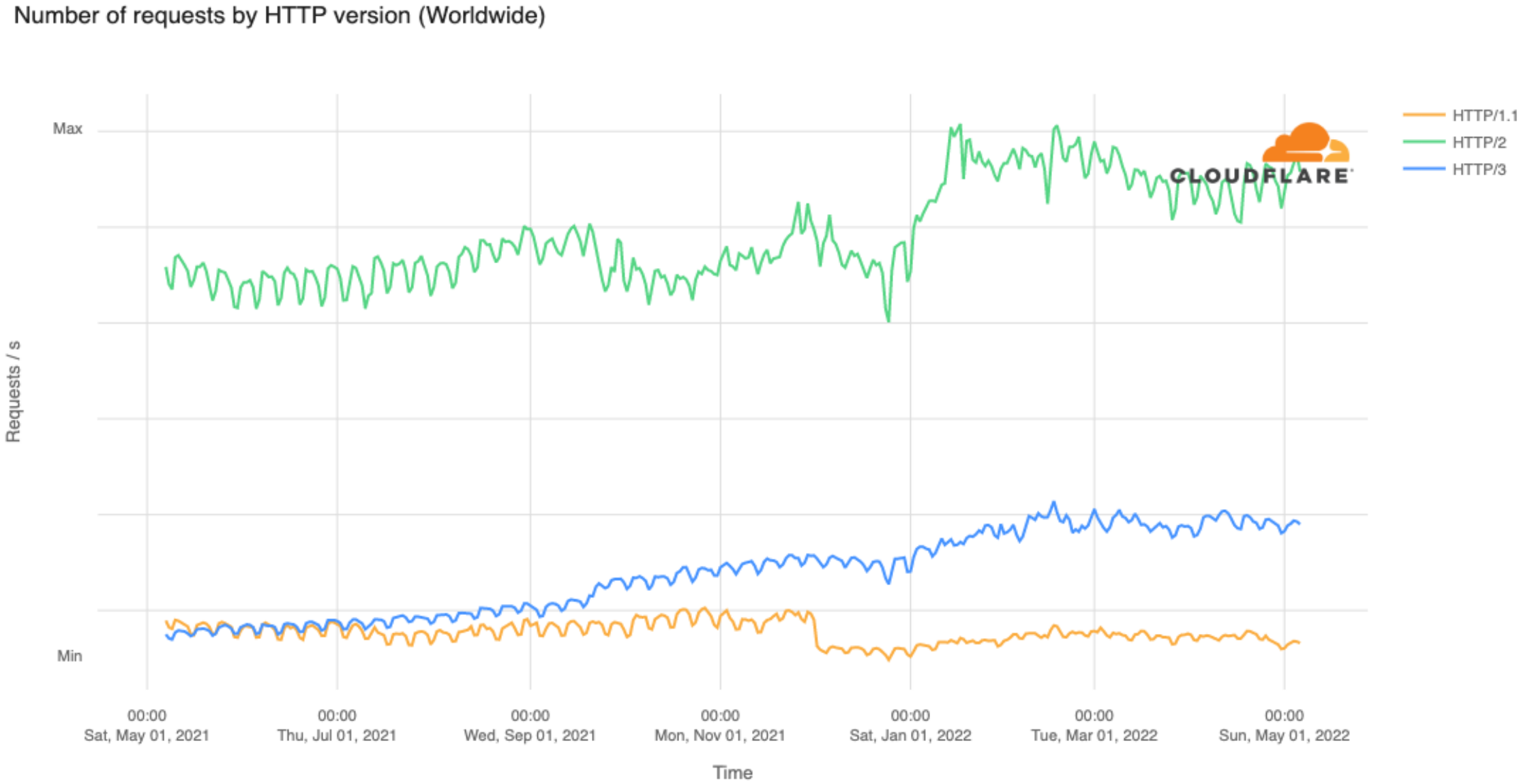
O_2, O_3, O_4 delivered quickly, O_1 slightly delayed

HTTP/2 to HTTP/3

Key goal: decreased delay in multi-object HTTP requests

HTTP/2 over single TCP connection means:

- recovery from packet loss still stalls all object transmissions
  - as in HTTP 1.1, browsers have incentive to open multiple parallel TCP connections to reduce stalling, increase overall throughput
- no security over vanilla TCP connection
- HTTP/3: adds security , per object error- and congestion-control (more pipelining) over UDP. [uses, QUIC: Quick UDP Internet Connections, A UDP-Based Multiplexed and Secure Transport]



Source: https://blog.cloudflare.com/cloudflare-view-http3-usage/

Network Applications & Demo

Principles of Network Applications

HTTP & WWW

E-mail Protocols (SMTP, POP, IMAP)

P2P Applications

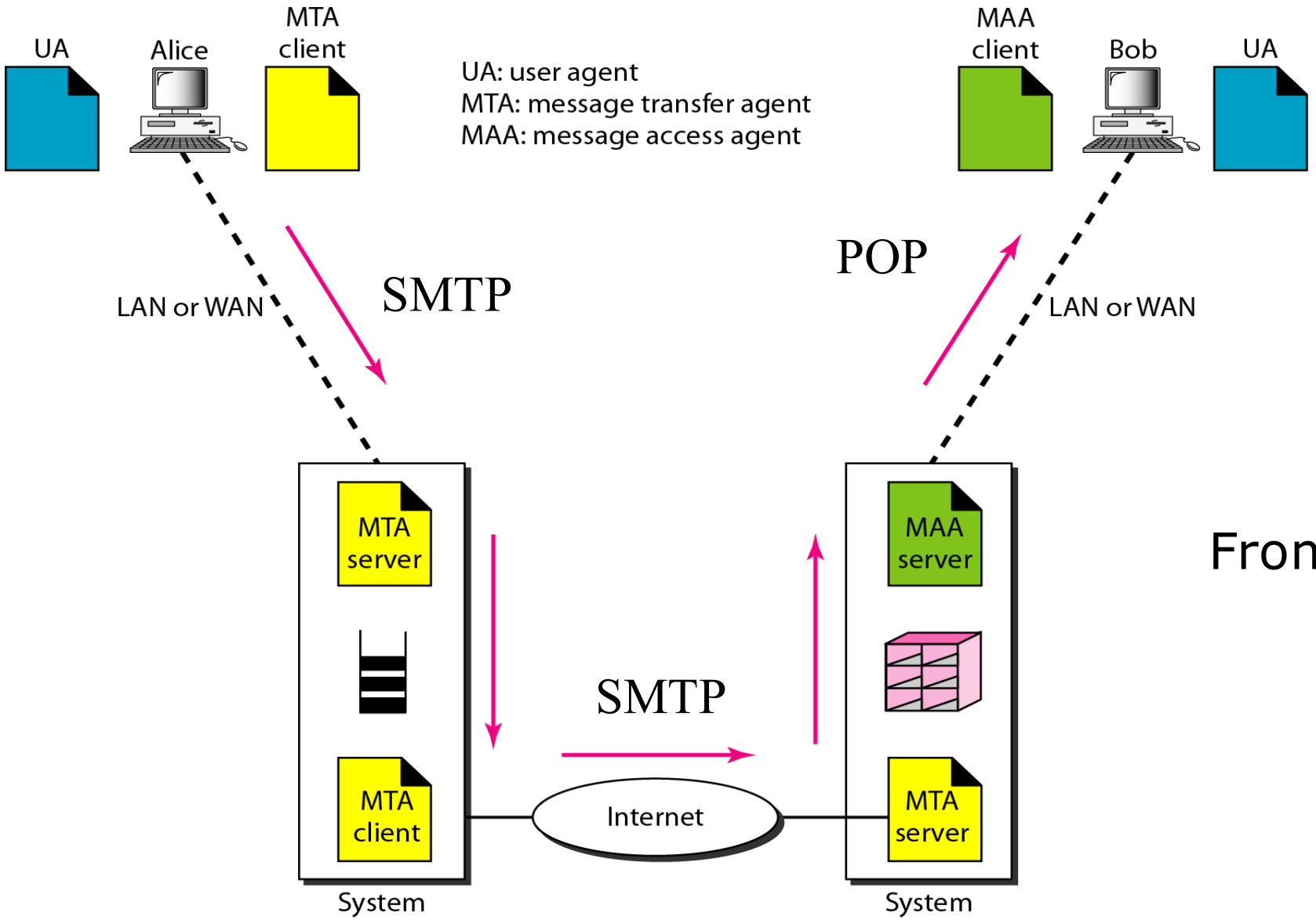
Dr. Anand Bhojan
COM3-02-49, School of Computing
banand@comp.nus.edu.sg ph: 651-67351

E-Mail : Three major components

- User Agents/Mail Readers (eg. Eudora, Outlook, elm, Mozilla Thunderbird, pine)
- Message Transfer Agents (SMTP - port 25)
  - SMTP : Simple Mail Transfer Protocol
- Message Access Agents (POP3 – port 110 and IMAP4 - port 143)
  - POP3 – Post Office Protocol v3
  - IMAP4 – Internet Mail Access Protocol v4

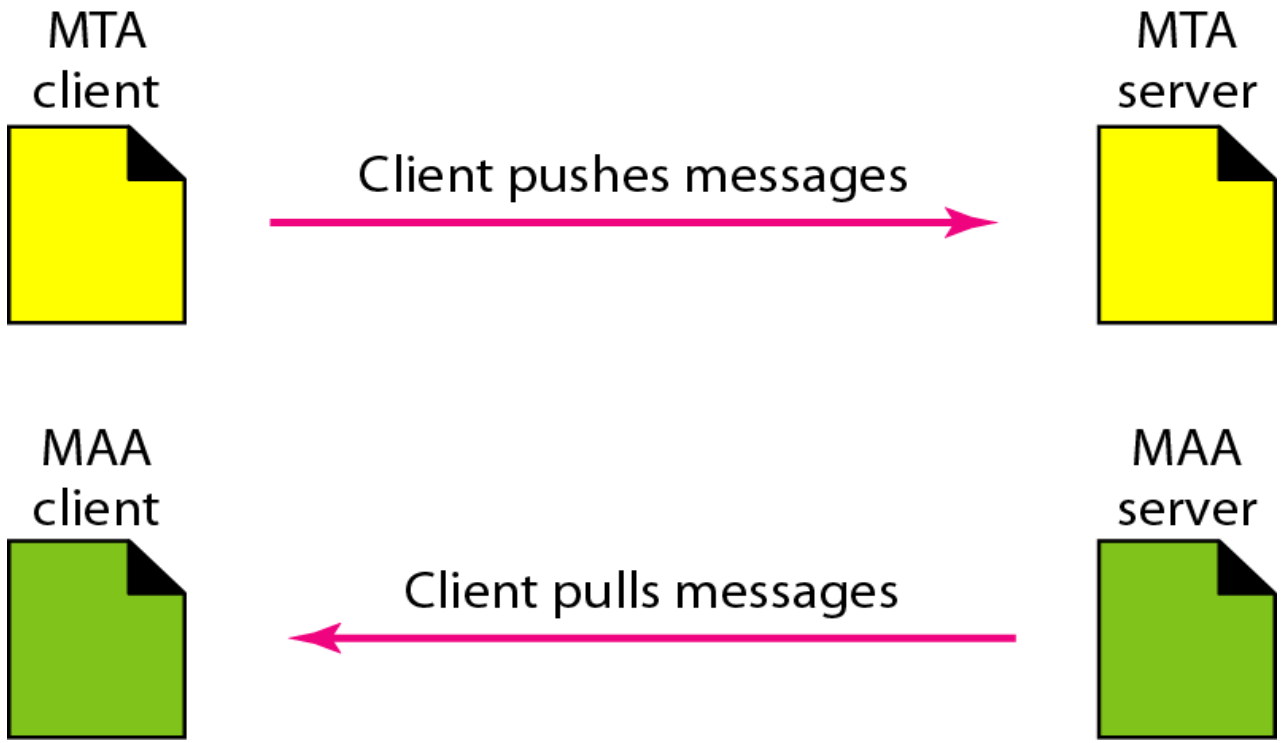
E-mail

wats gg on here



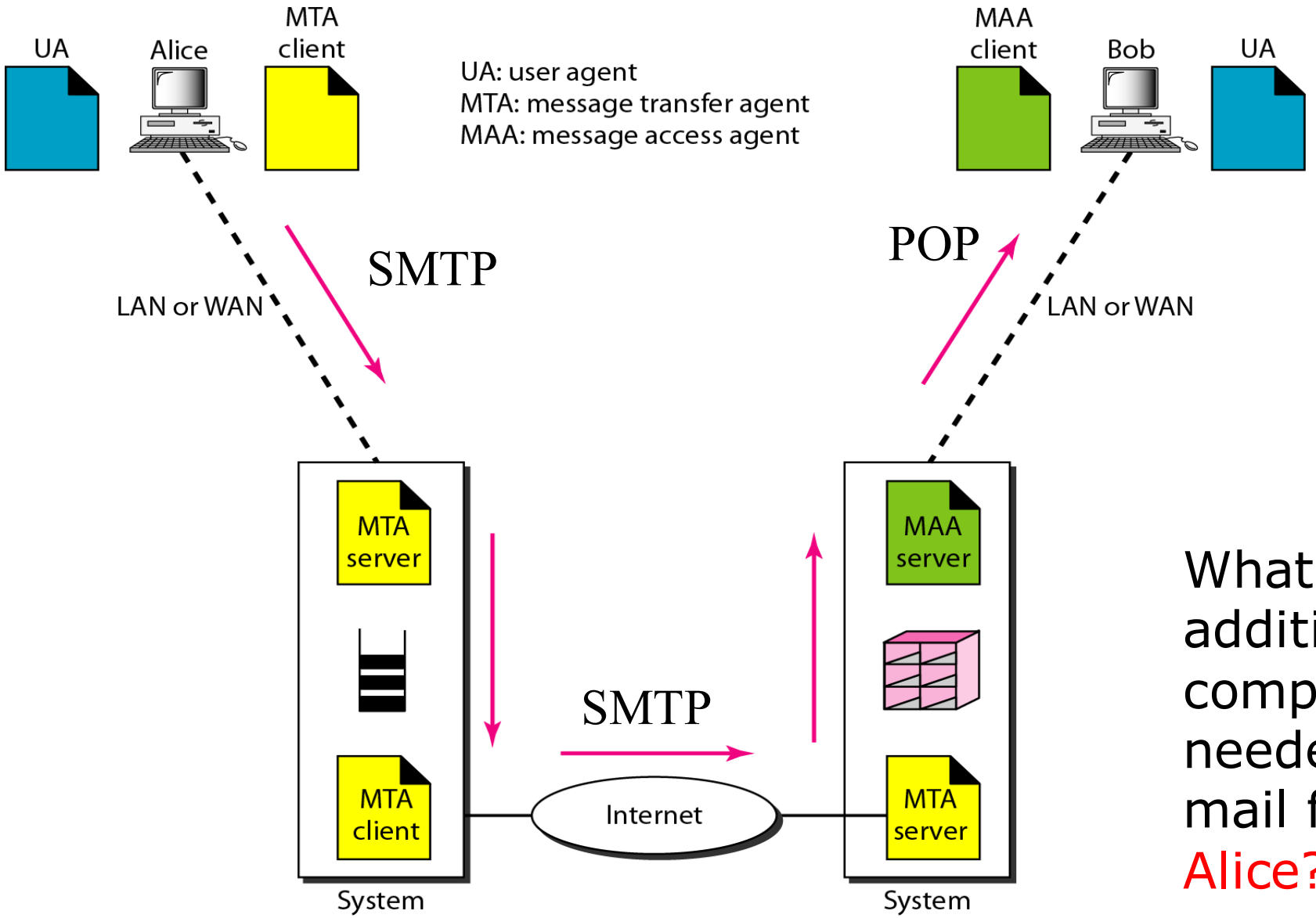
From Alice to Bob

E-mail : PUSH vs PULL



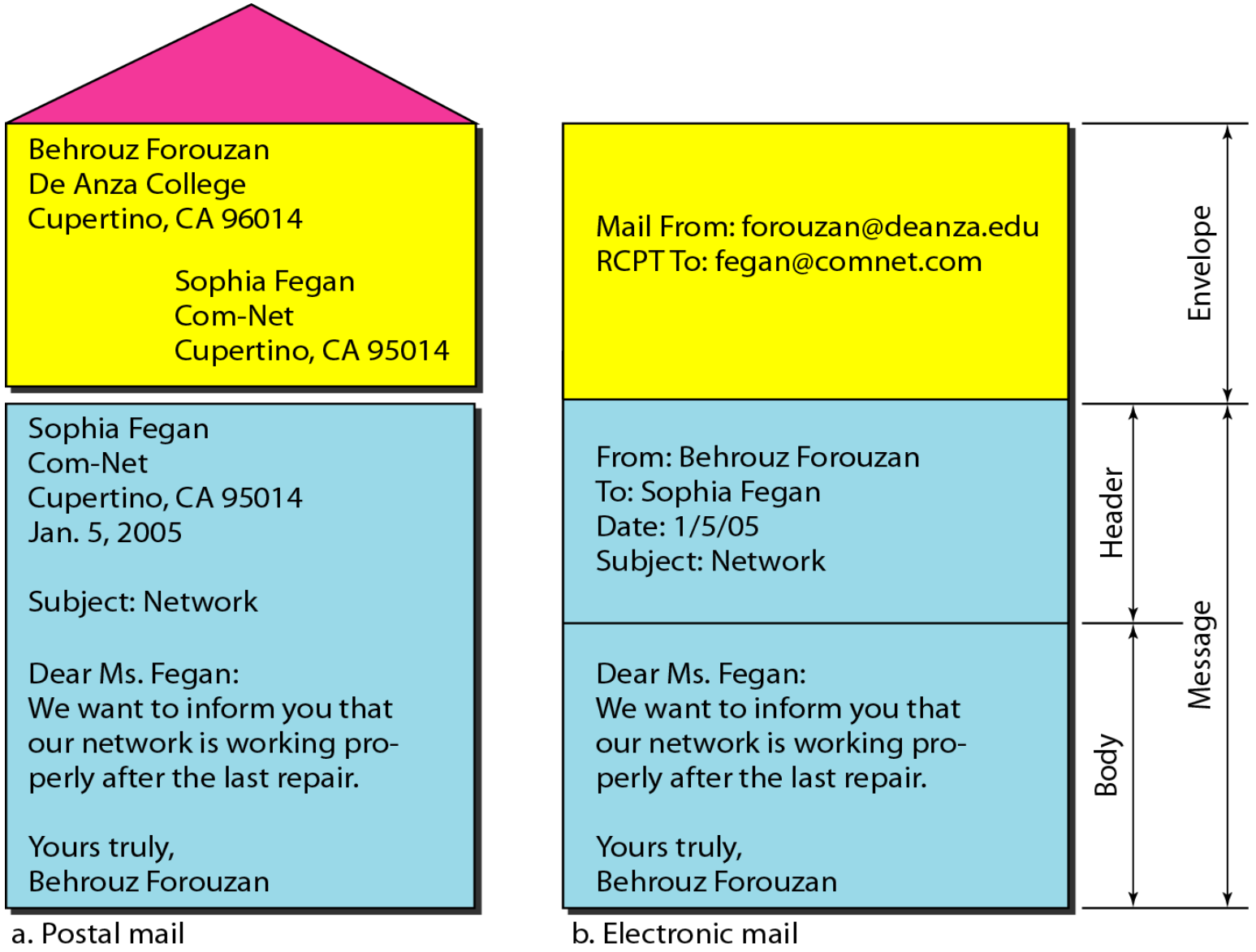
- POP- PULL protocol
- SMTP – PUSH protocol
- Then, what is HTTP/1.1? (PUSH or PULL)

E-mail

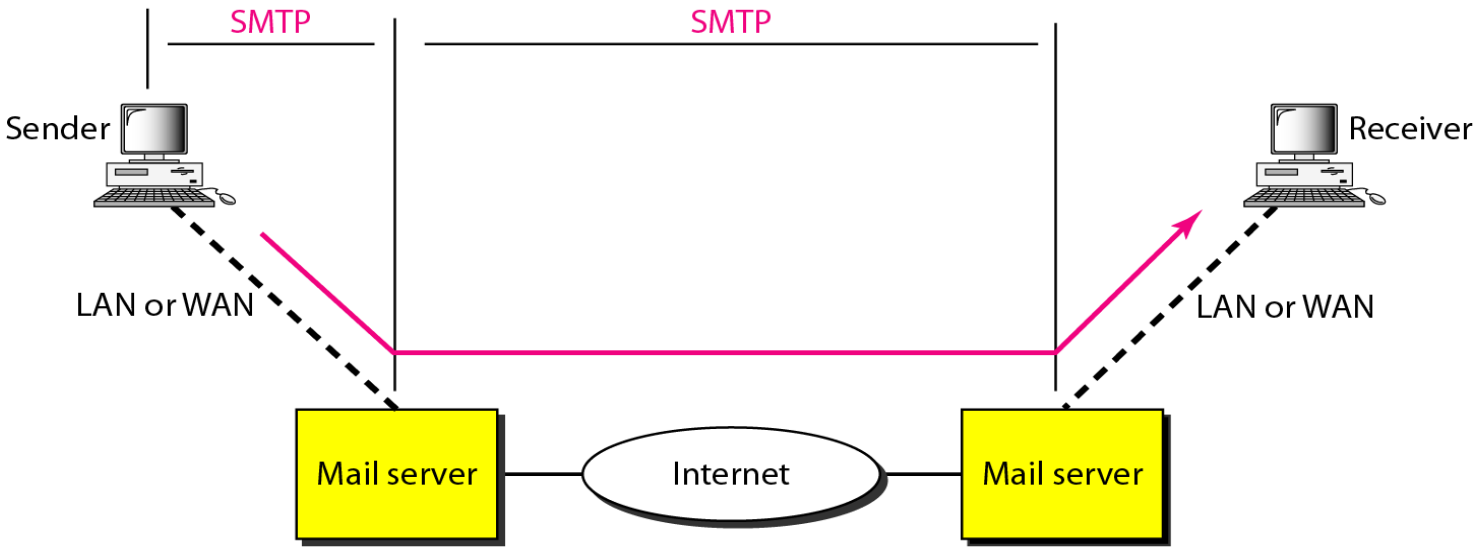


What are the additional components needed to send mail from Bob to Alice?

Email - Format



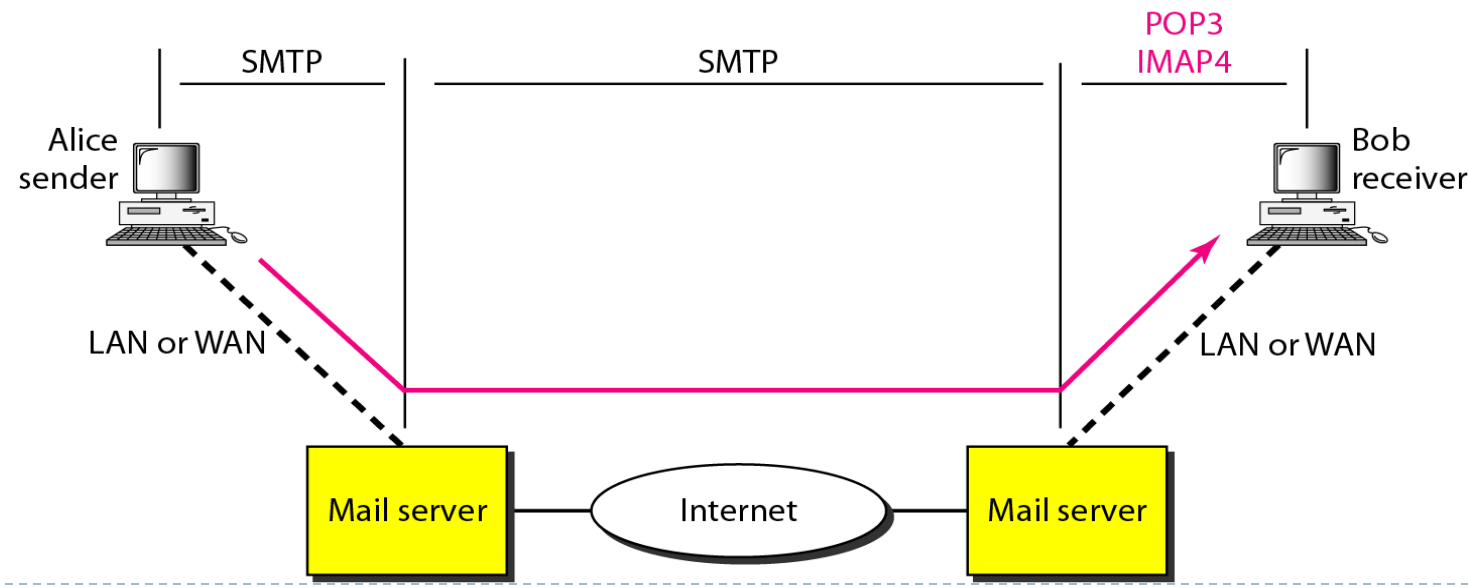
E-Mail:- MTA (eg. SMTP), MAA (eg. POP/IMAP)



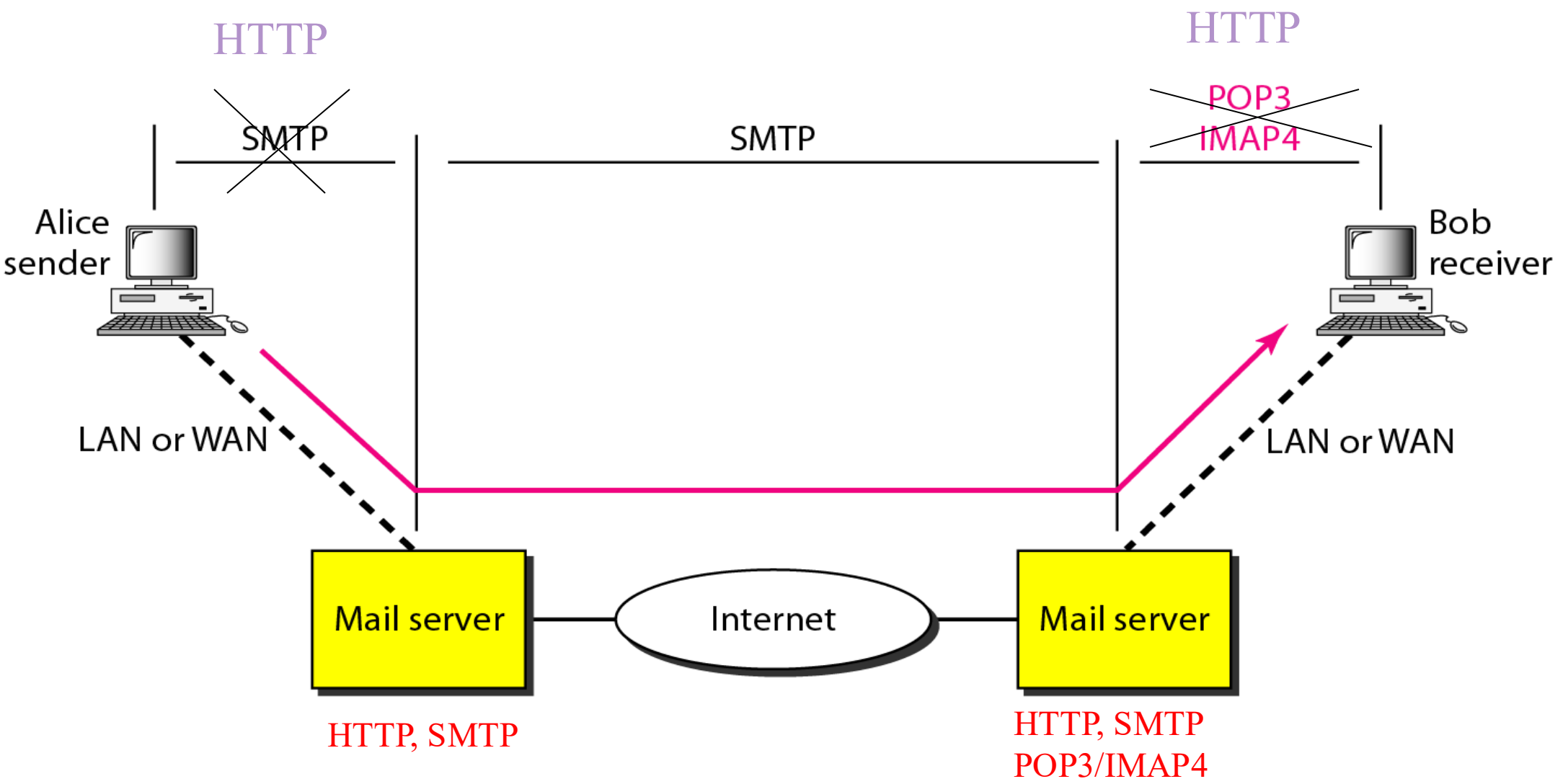
SMTP - Simple Mail Transfer Protocol

POP3-Post Office Protocol (v3) [RFC 1939]

IMAP4-Internet Mail Access Protocol (v4) [RFC 1730]



E-mail- Web based



SMTP Example

- **$openssl s_client -connect smtp.gmail.com:465 -crlf -ign_eof**
- CONNECTED(00000005)
- **EHLO gmail.com**
- 250-mx.google.com at your service, [137.132.84.20]….
- **AUTH LOGIN yourBASE64encodedUserID**
- 334 UAPLGGFdskzcmQ6
- **yourBase64encodedPassword**
- 235 2.7.0 Accepted
- **MAIL FROM: <yourmailID@gmail.com>**
- 250 2.1.0 OK ys4sm2252854lpbb.9 - gsmtp
- **RCPT TO: <yourMailID@nus.edu.sg>**
- 250 2.1.5 OK ys4sm2252854lpbb.9 - gsmtp
- **DATA**
- 354 Go ahead ys4sm2252854lpbb.9 - gsmtp
- **FROM:me@gmail.com**
- **TO:you@nus.edu.sg**
- **Subject: It works**
- *The last line should have a*
- ***DOT.***
- **My first email with openssl!**
- •
- 250 2.0.0 OK 1378814309 ys4sm2252854lpbb.9 - gsmtp
- **QUIT**
- 221 2.0.0 closing connection ys4sm2252854lpbb.9 – gsmtp
- **$**

- SMTP uses persistent connections
- SMTP requires message (header & body) to be in 7-bit ASCII
- SMTP: multiple objects sent in multipart msg

POP Example

- **telnet pop.nus.edu.sg 110**
- *+OK The Microsoft Exchange POP3 service is ready.*
- **USER myUsername**
- *+OK*
- **PASS myPassword**
- *+OK User successfully logged on.*
- **LIST**
- *1 1051*
- *2 908*
- *3 10884*
- .
- **RETR 1**
  - *+OK*
  - *From: "somesender" <somesender@nus.edu.sg>*
  - *Content-Class: urn:kvsplc-com:cc:vault:shortcut*
  - *Date: Fri, 17 Sep 2010 17:21:03 +0800*
  - *Subject:Title of the mail*
  - *...*
- **QUIT**
  - *+OK Microsoft Exchange Server 2007 POP3 server signing off.*

Use OpenSSL for Secured POP access:

$openssl s_client -connect pop.nus.edu.sg:995 -crlf -ign_eof

<or>

$openssl s_client -crlf -connect pop.nus.edu.sg:110 -starttls pop3

POP3 (more) and IMAP

more about POP3

- previous example uses POP3 "download and delete" mode
  - Bob cannot re-read e-mail if he changes client
- POP3 "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

P2P Applications

Principles of Network Applications

HTTP & WWW

E-mail Protocols (SMTP, POP, IMAP)

P2P Applications

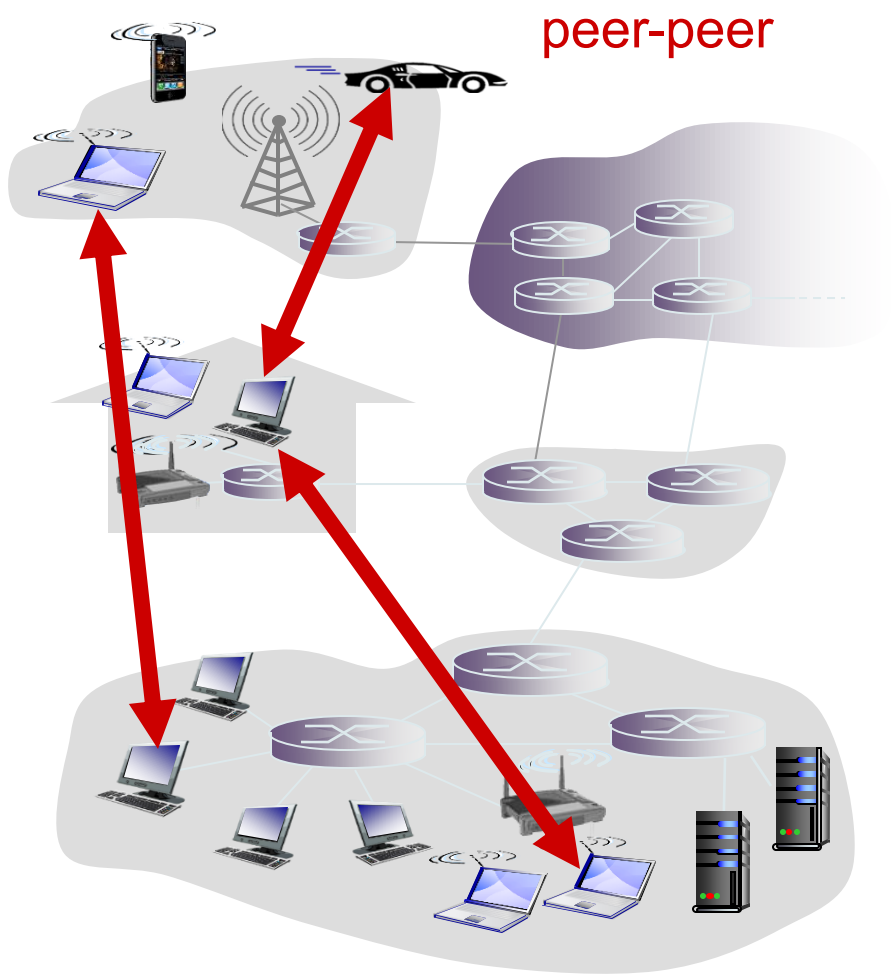
Dr. Anand Bhojan
COM3-02-49, School of Computing
banand@comp.nus.edu.sg ph: 651-67351

P2P architecture

- *no* always-on server
- arbitrary end systems directly communicate and serve each other.
  - Each host runs both server and client processes

examples:

- file distribution (BitTorrent)
- Streaming (KanKan)
- VoIP (Skype)
- Multiplayer Games
- Blockchain (bitcoin)



Q1: What is the advantage of P2P?

- Scalability (explained next)

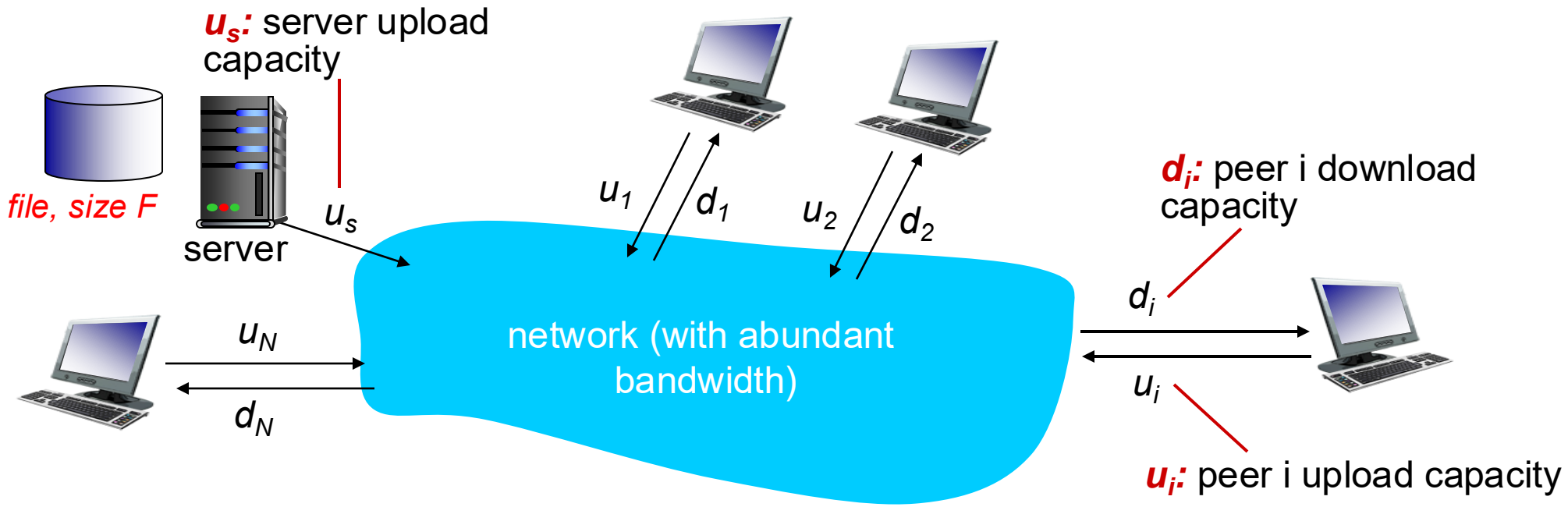
Q2: How the hosts find the IP addresses of each other?

An interesting article to read: https://www.cse.wustl.edu/~jain/cse570-21/ftp/p2p/index.html

File distribution: client-server vs P2P

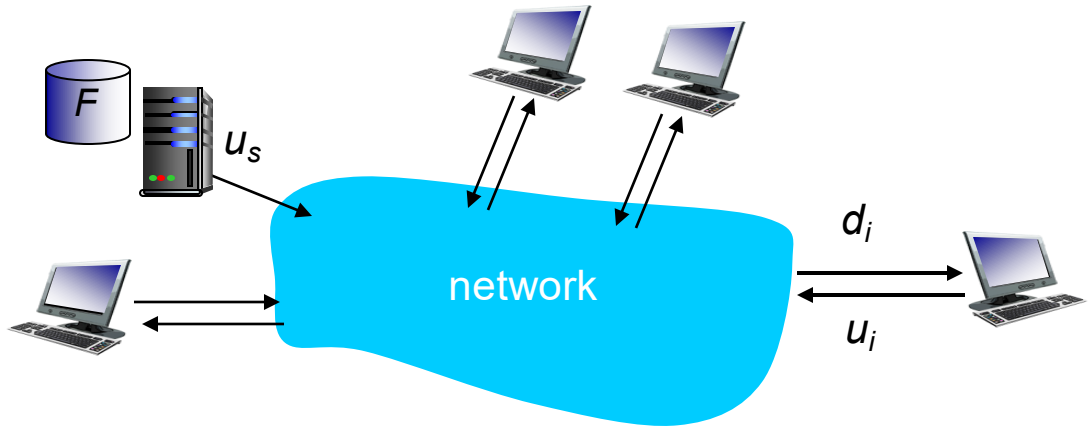
Question: how much time to distribute file (size *F*) from one server to *N* peers?

- peer upload/download capacity is limited resource



File distribution time: client-server

- *server transmission:* must sequentially send (upload) *N* file copies:
  - time to send one copy: F/u_s
  - time to send N copies: NF/u_s



- *client:* each client must download file copy
  - d_{min} = min client download rate
  - min client download time: F/d_{min}

time to distribute F to N clients using client-server approach

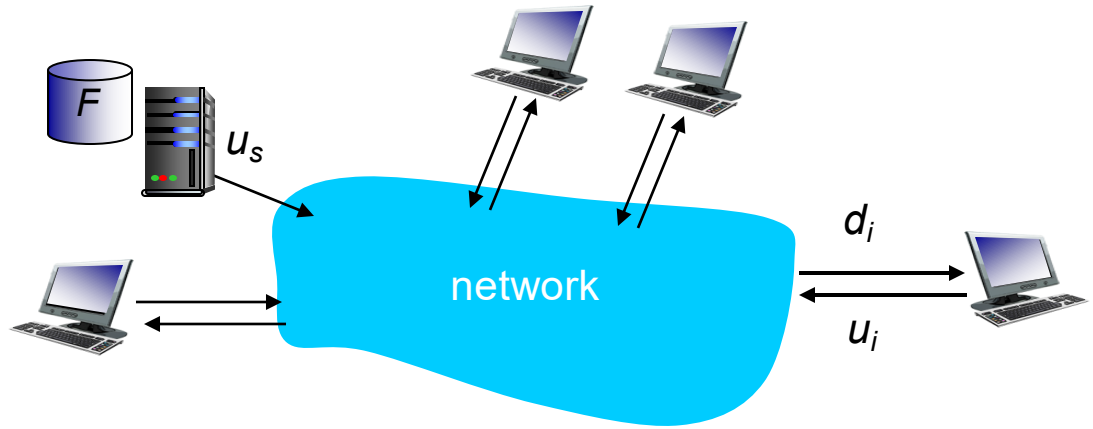
$$D_{c\text{-}s} \geq max\{NF/u_s, F/d_{min}\}$$

increases linearly in N

File distribution time: P2P

Question: how much time to distribute file (size *F*) from one server to *N* *peers*?

- *server transmission:* must upload at least one copy
  - time to send one copy: F/u_s



- *client:* each client must download file copy
  - min client download time: F/d_{min}
- *clients:* as aggregate must download *NF* bits
  - max upload rate (limting max download rate) is $u_s + \Sigma u_i$

time to distribute F to N clients using P2P approach

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

increases linearly in *N* …

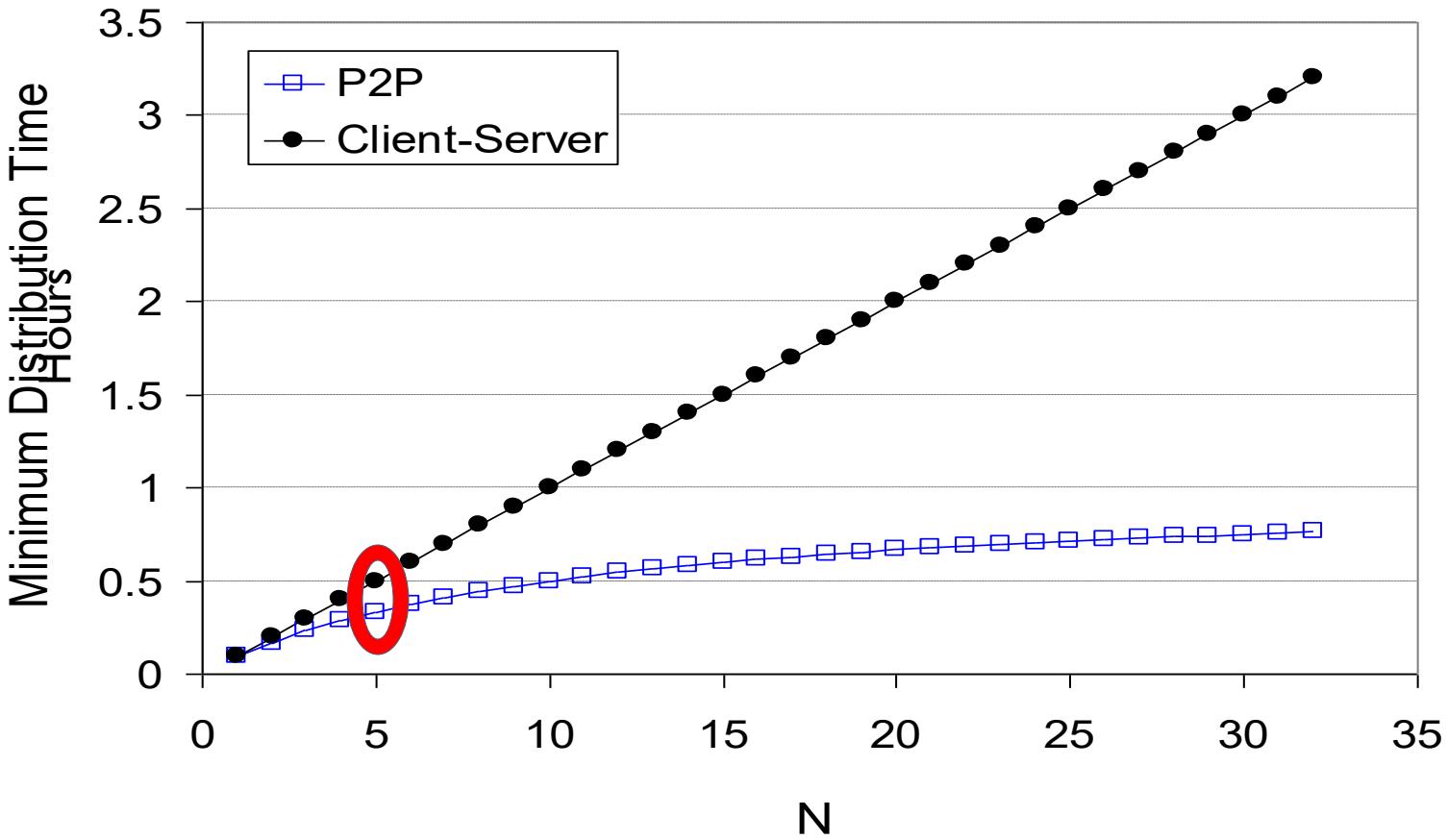
… but so does this, as each peer brings service capacity

Client-server vs. P2P: example (Just based only on upload rate)

- Client upload rate = *u*
- Server upload rate u_s = *10u*

To keep is simple, Assume client download rate $d_{min} \geq u_s$
[So, the bottleneck is upload rate in C-S mode!]

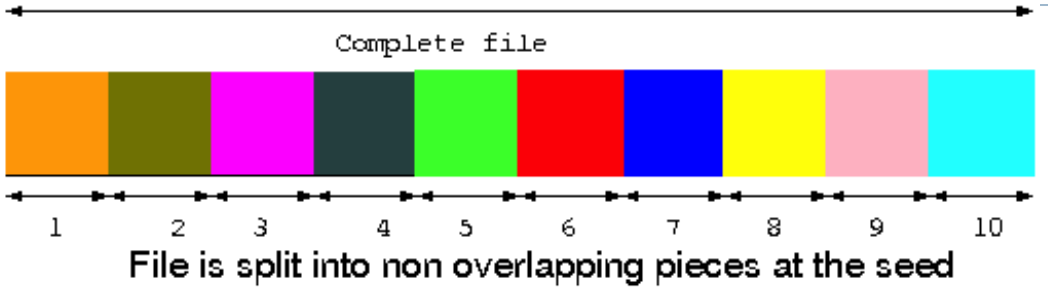
- File size F = 60 GBits
- Assume, client upload time *F/u* = 1 hour
  - => Server upload time F/10u = 60/10 = 6 mins
  - So, to transmit (upload) to 5 clients in seq, it takes 5 x 6 mins



- In p2p transmission, it takes N*60/(10u+5u)
- Which is, 5 * 60/15u = 5 * 4 mins

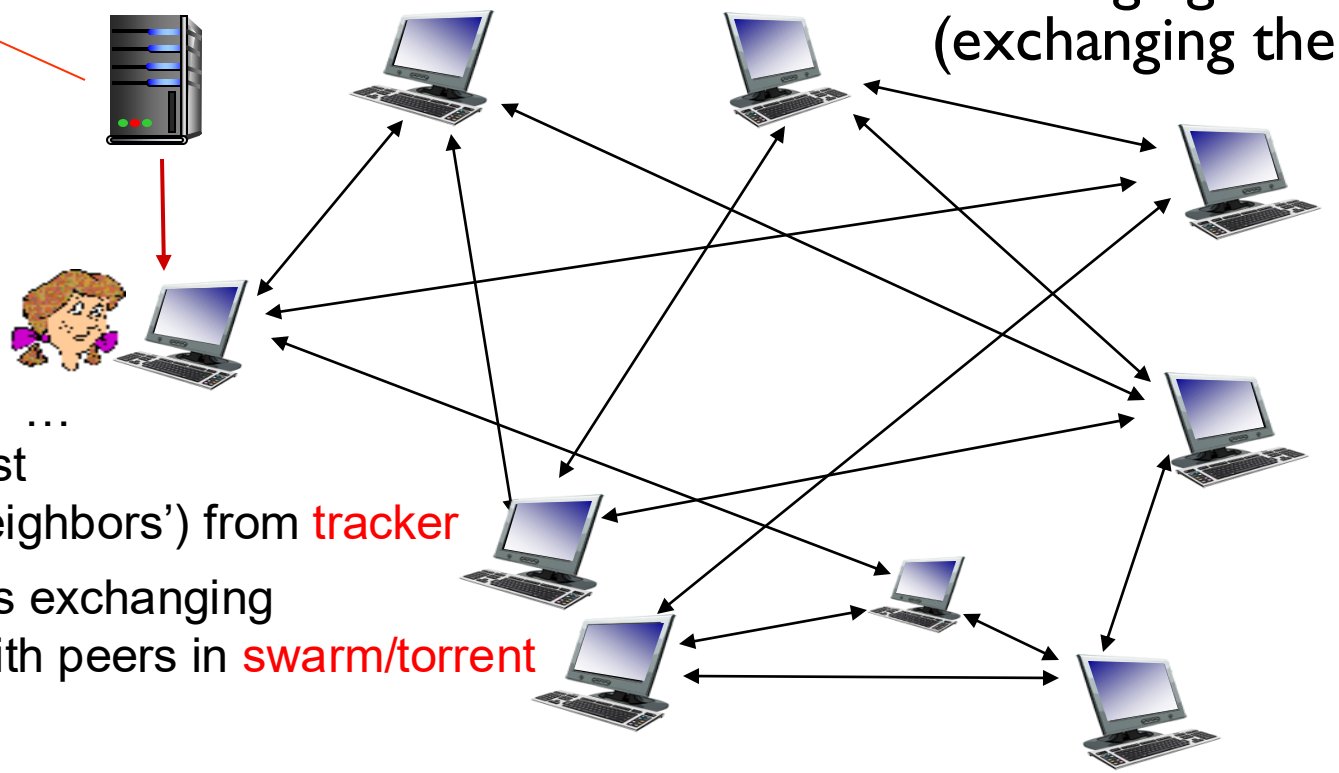
P2P file distribution: BitTorrent (BT)

- file divided into 256Kb chunks
- peers in torrent (aka, swarm) send/receive file chunks



tracker: tracks peers participating in a torrent

Torrent/swarm: group of peers exchanging chunks of a file. (exchanging the same file)



Alice arrives …
… obtains list
of peers ('neighbors') from tracker

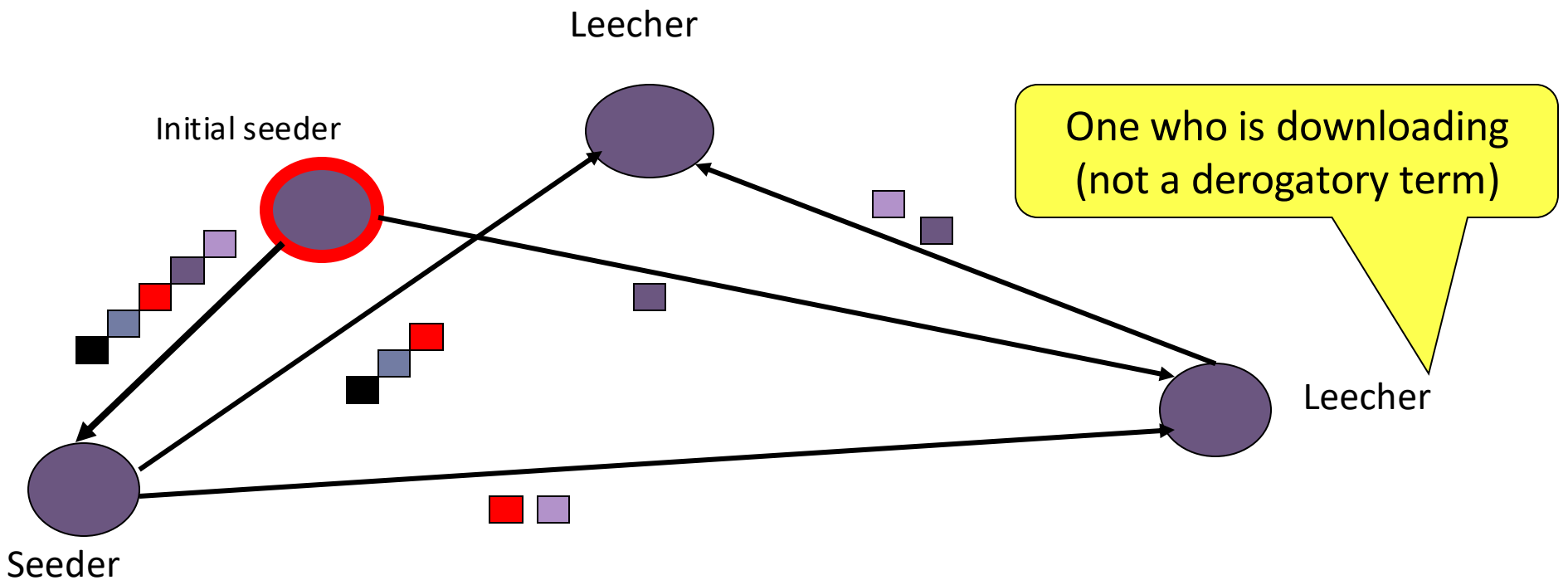
… and begins exchanging
file chunks with peers in swarm/torrent

churn: peers may come and go.
once peer has entire file, it may (selfishly) leave or (altruistically) remain in the torrent as seeder

BitTorrent Lingo

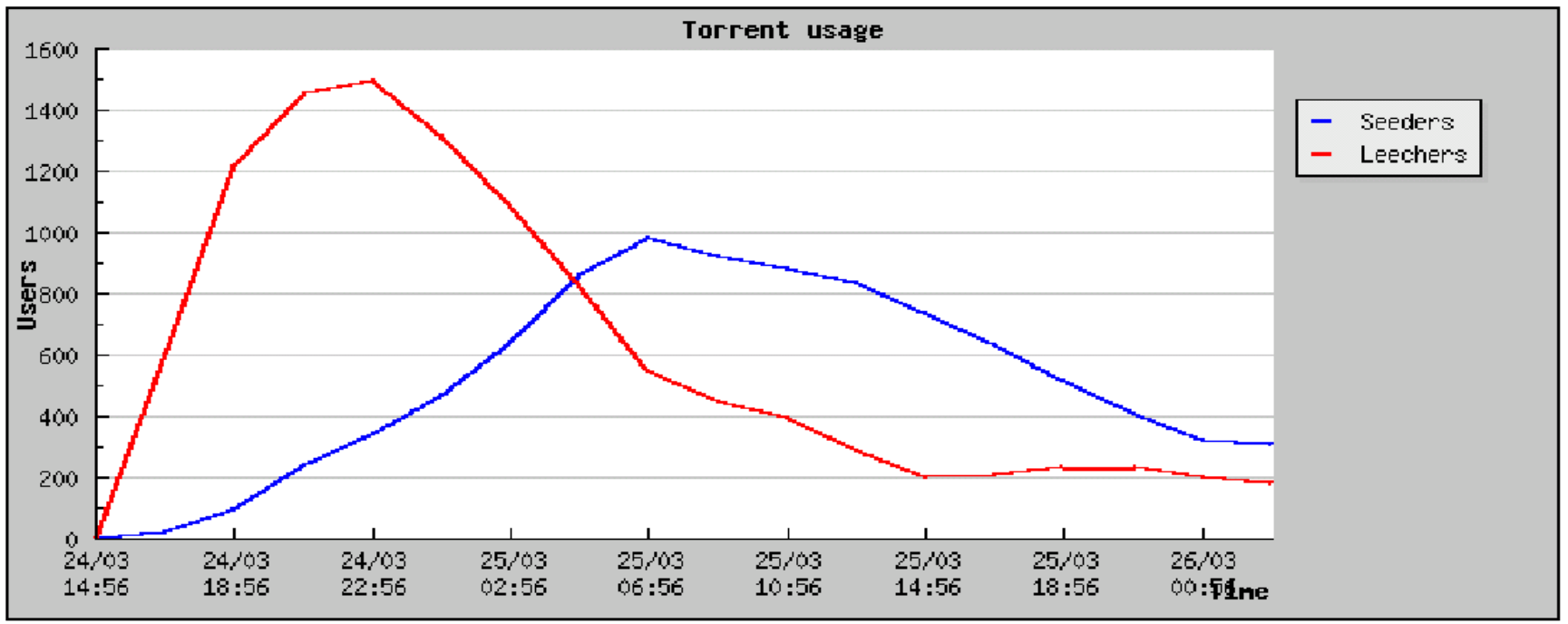
Seeder = a peer that provides the complete file.

Initial seeder = a peer that provides the initial copy.



Basic Idea

- As a leecher downloads pieces of the file, replicas of the pieces are created. ***More downloads mean more replicas available.***
- As soon as a leecher has a complete piece, it can potentially share it with other downloaders. Eventually each leecher becomes a seeder by obtaining all the pieces, and assembles the file. Verifies the checksum.

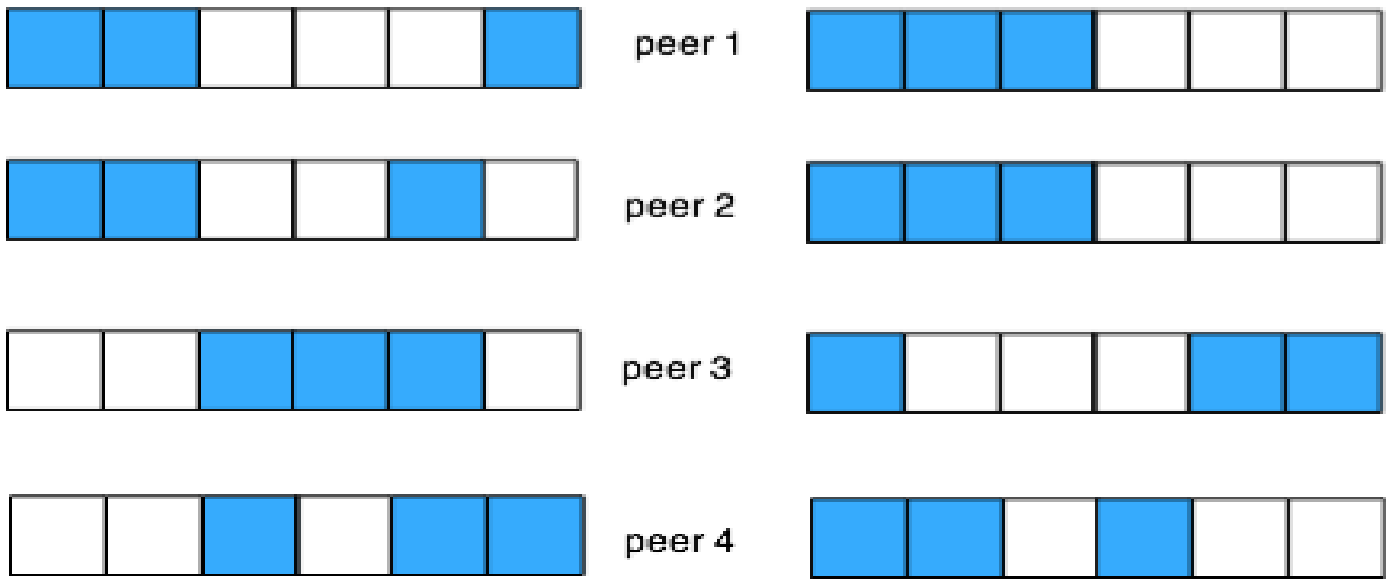


BitTorrent: requesting, sending file chunks

requesting chunks:

- at any given time, different peers have different subsets of file chunks
- periodically, Alice asks each peer for list of chunks that they have
- Alice requests missing chunks from peers.
- **Requires - Piece Selection** Policies:
  - The order in which pieces are selected by different peers is critical for good performance
  - If an inefficient policy is used, then peers may end up in a situation where each has all identical set of easily available pieces, and none of the missing ones.
  - If the original seed is prematurely taken down, then the file cannot be completely downloaded! What are "good policies?"

Piece Selection



Small overlap is good

Large overlap is bad
-- wastes bandwidth

Piece selection

- **Strict Priority**
  - Not common
- **Random First Piece**
  - Special case, at the beginning
- **Rarest First**
  - General rule
- **Endgame Mode**
  - Special case, at the end

Random First Piece

- Initially, a peer has nothing to trade
- Important to get a complete piece ASAP
- Select a random piece of the file and download it

Rarest Piece First

- Determine the pieces that are most rare among your peers, and download those first.
- Logically, the most commonly available pieces are left till the end to download.

Endgame Mode

- Near the end, missing pieces are requested from every peer containing them.
- This ensures that a download is not prevented from completion due to a single peer with a slow transfer rate.
- Some bandwidth is wasted, but in practice, this is not too much.

BT: internal mechanism

How to discourage free riding?

- Built-in **incentive** mechanism (where all the magic happens):
  - Choking Algorithm
  - Optimistic Unchoking

Choking

- **Choking** is a *temporary refusal* to upload to free riders. It is one of BT's most powerful idea to deal with **free riders (those who only download but never upload)**.
- ***Tit-for-tat strategy*** is based on game-theoretic concepts.

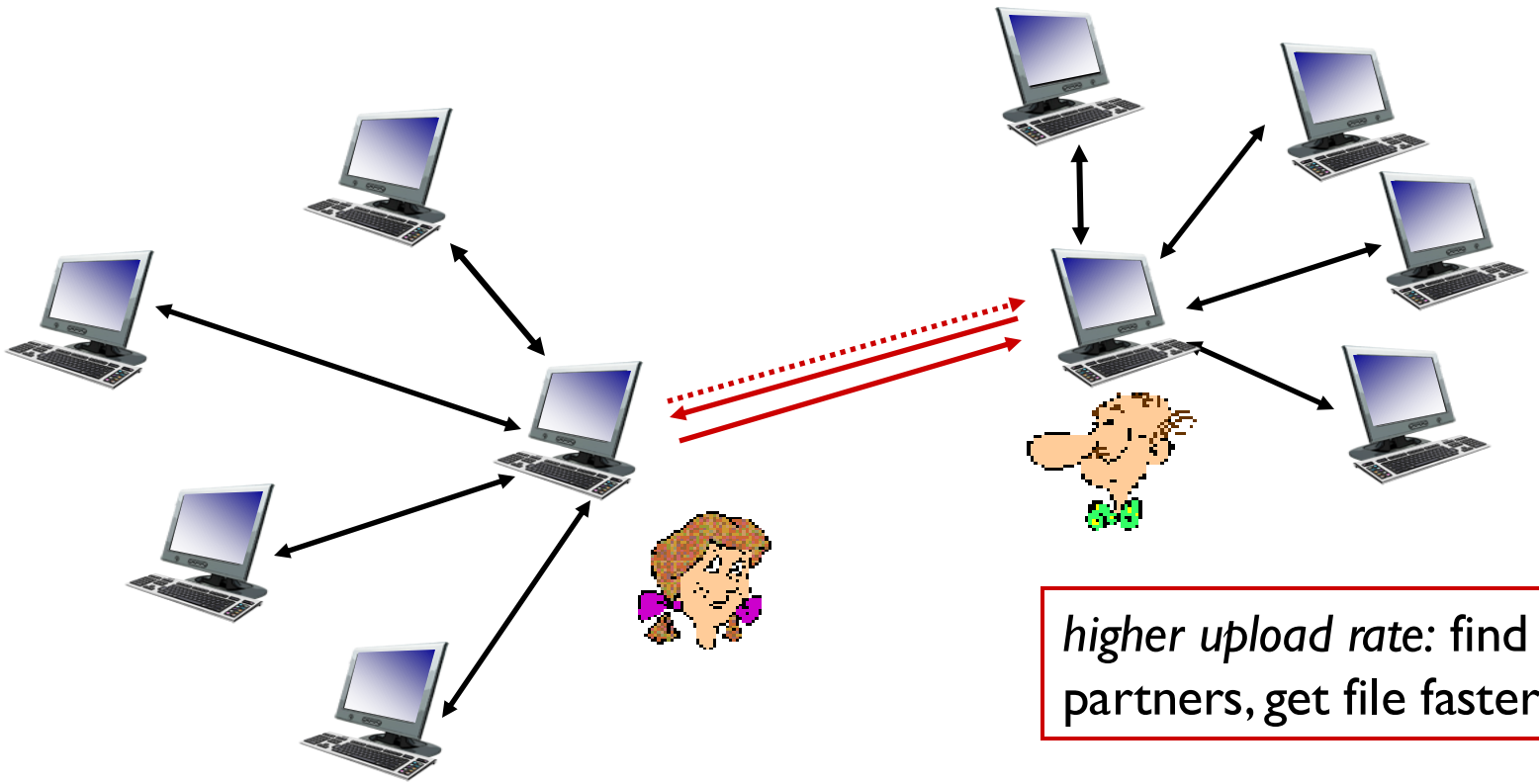
BitTorrent: requesting, sending file chunks

sending chunks: tit-for-tat

- Alice sends chunks to 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 every10 secs
- every 30 secs: randomly select another peer, starts sending chunks
  - "optimistically unchoke" this peer
  - newly chosen peer may join top 4 [See next slide ….]

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 !

Upload-Only mode

- Once download is complete, a peer can only upload. The question is, which nodes to upload to?
- Policy: Upload to those with the best upload rate. This ensures that pieces get replicated faster, and new seeders are created fast

What is Trackerless torrents

- BitTorrent also supports "trackerless" torrents, featuring a DHT implementation that allows the client to download torrents that have been created without using a BitTorrent tracker.

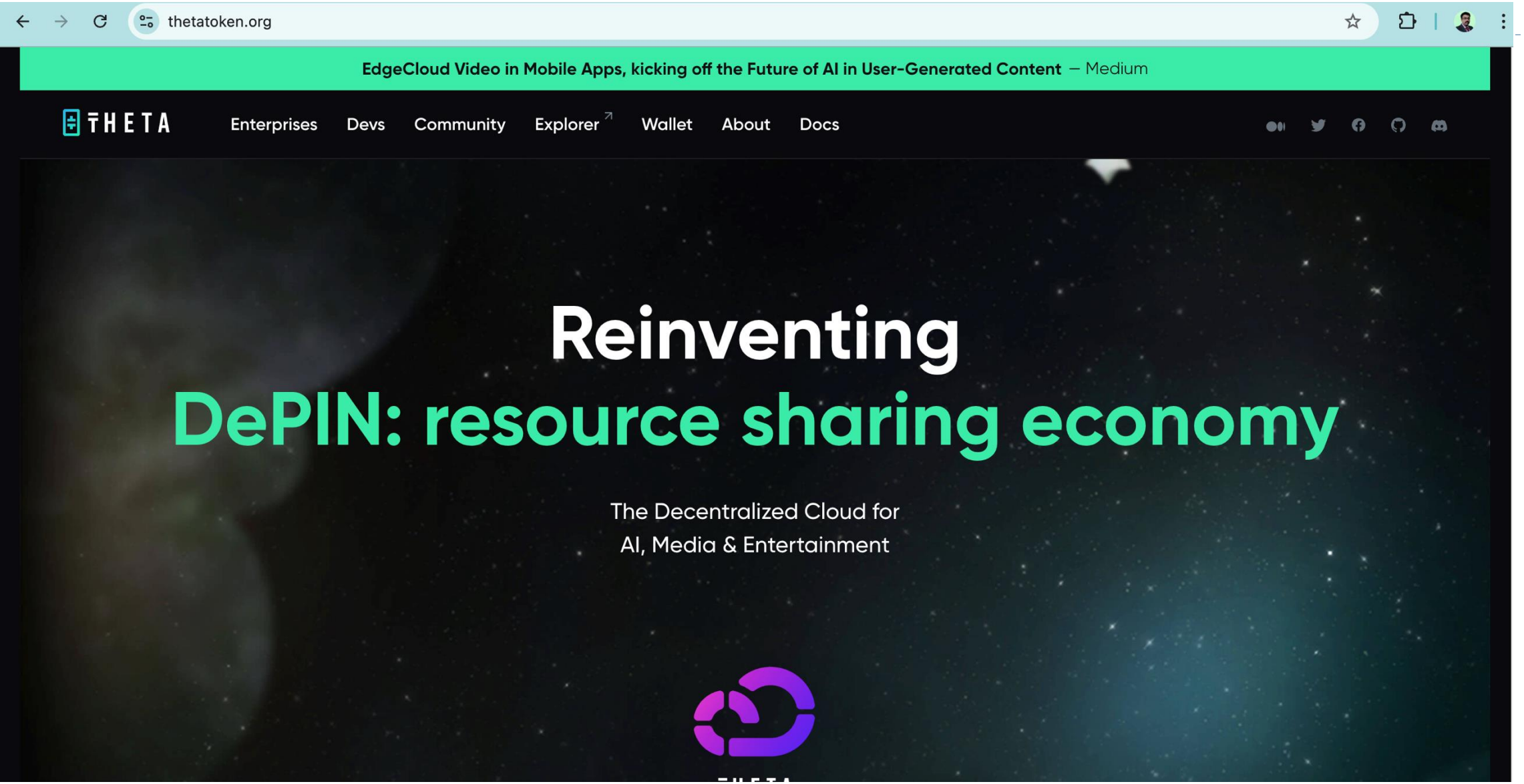
- Question to ponder: How and where to maintain the peer database, when there is no centralised *tracker ?*

Questions about BT to discuss…

- What is the effect of bandwidth constraints?
- Is the Rarest First policy really necessary?
- Must nodes perform seeding after downloading is complete?
- How serious is the Last Piece Problem?
- Does the incentive mechanism affect the performance much?
- What if the peers are behind NAT (using private IP)?

Some Decentralised Systems/Apps (p2p)

For your reading pleasure (FRYOP)…


thetatoken.org
EdgeCloud Video in Mobile Apps, kicking off the Future of AI in User-Generated Content – Medium
THETA
Enterprises
Devs
Community
Explorer
Wallet
About
Docs
Reinventing
DePIN: resource sharing economy
The Decentralized Cloud for
AI, Media & Entertainment

Beaker is a discontinued[4] free and open-source web browser[5] developed by Blue Link Labs.[6][7] Beaker Browser peer-to-peer technology allows users to self-publish websites and web apps[8] directly from the browser, without the need to set up and administrate a separate web server or host their content on a third-party server. All files and websites are transferred using Dat, a hypermedia peer-to-peer protocol, which allows files to be shared and hosted by several users.[9] The browser also supports the HTTP protocol to connect to traditional servers.[5]

Beaker

Screenshot of the Beaker browser on Fedora 31

Developer(s)	Blue Link Labs.[1]
Initial release	1 August 2017; 7 years ago
Stable release(s)	
1.1.0 (December 8, 2020; 3 years ago[2]) [+]	

brave.com

Browser Brave Search Why Brave Search API Private advertising Get Brave Search The Web

The browser that puts you first

Block ads. Save data. And get way faster webpages. Just by switching your browser.

Get Brave

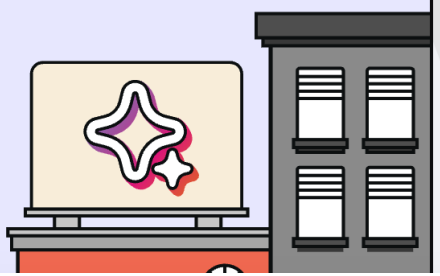
Private search

Search the web privately...

Better results, less SEO spam, and zero profiling.

Built-in AI assistant

Get answers, generate content, & more. Right in the browser.



Powerful VPN

Protect every app, on your entire device.

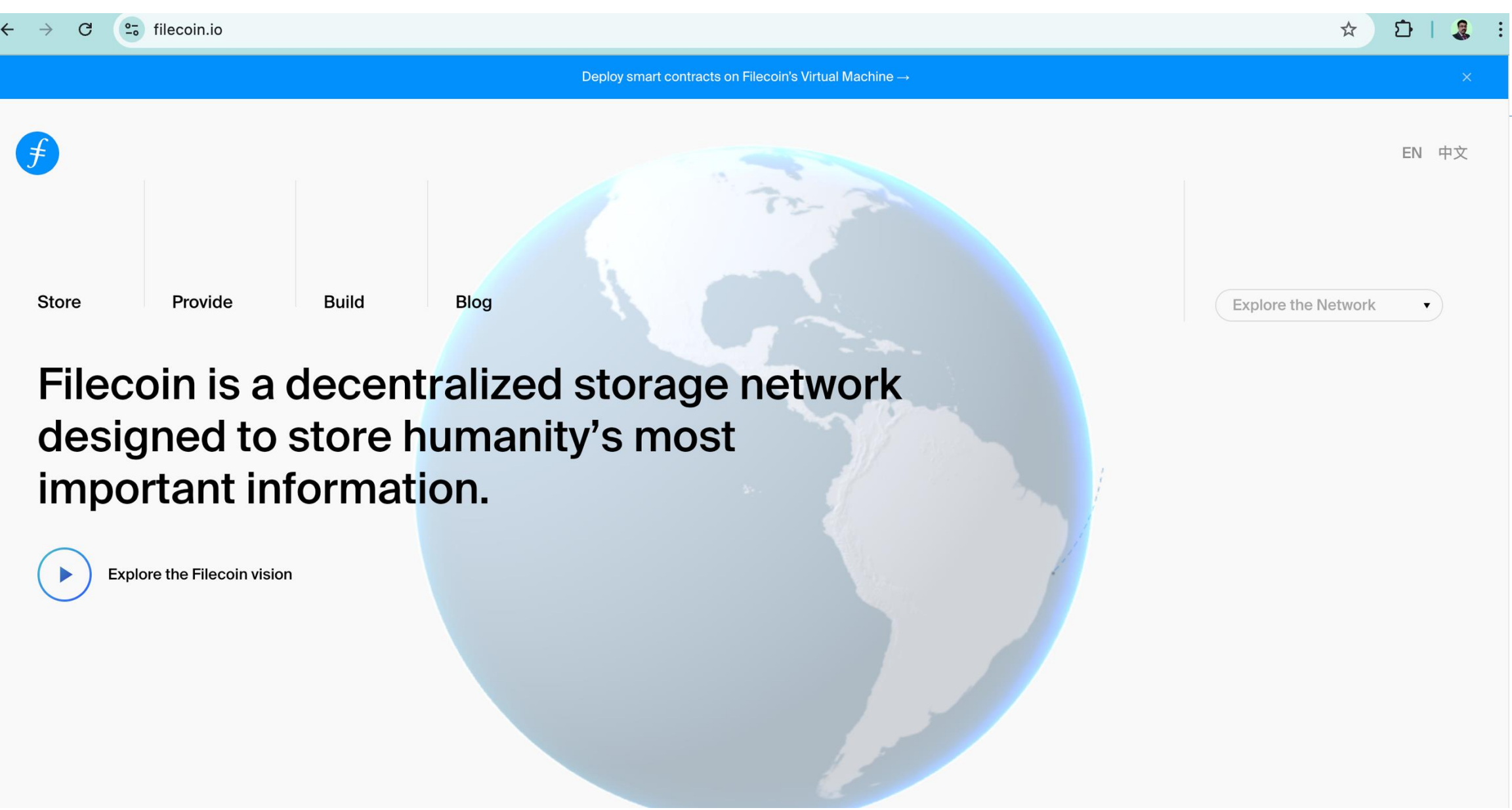
Signal

Get Signal Help Blog Developers Careers Donate English

Speak Freely

Say "hello" to a different messaging experience. An unexpected focus on privacy, combined with all of the features you expect.

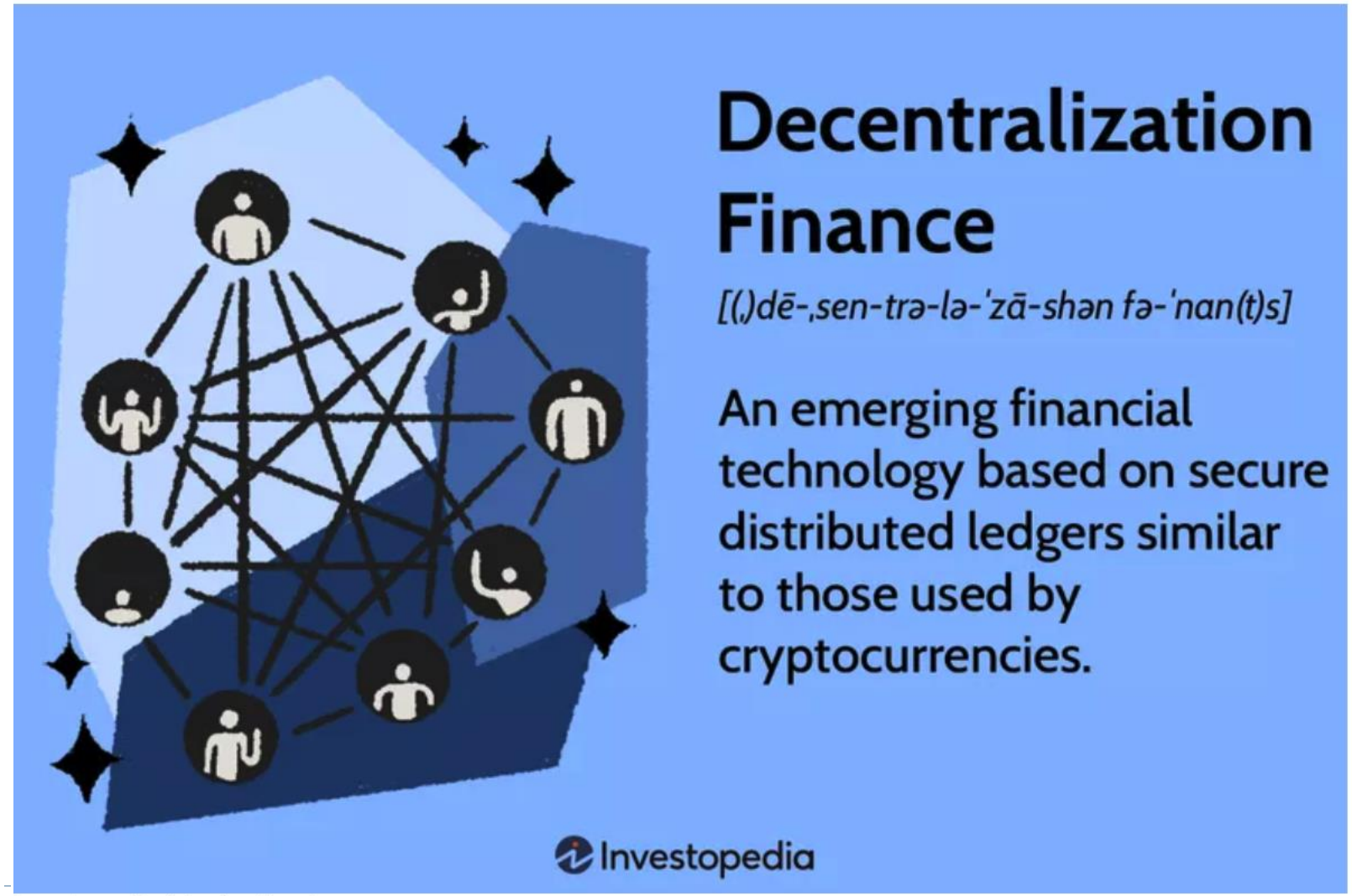
Get Signal



- Uses IPFS IPFS (InterPlanetary File System) Protocol

DeFi

- P2P transactions for lending, borrowing, and trading assets without intermediaries.



Investopedia / Joules Garcia

THE END

- Principles of Network Applications
- HTTP & WWW
- E-mail Protocols (SMTP, POP, IMAP)
- P2P Applications

Activities:

LAB2 goes virtual with mini-net. It is take-home Lab.
You will have post lab Quiz

Assignment 2 (NW tools) will be released tomorrow.

Q: How bitcoin's p2p network works?
Q: What are the problems with current c/s social apps? Can that be solved using p2p?

Attendance

https://inetapps.nus.edu.sg/ctr/