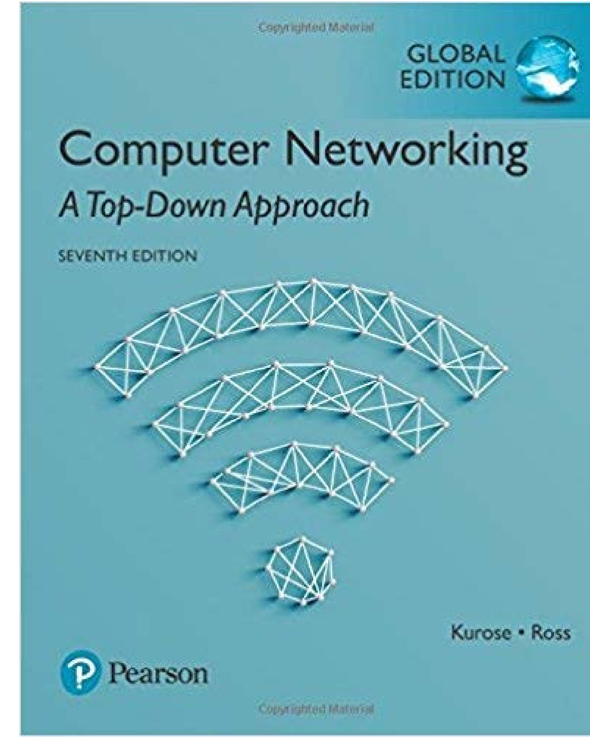
Chapter 2
Application Layer part 1

School of Computing
Gachon Univ.
Joohyung Lee

Most of slides from J.F Kurose and K.W. Ross. And, some slides from Prof. Joon Yoo



Computer Networking: A Top Down Approach

7^{th} edition
Jim Kurose, Keith Ross
Pearson, 2017

가천대학교
Gachon University

Chapter 2: outline

2.1 principles of network applications

2.2 Web and HTTP

2.3 electronic mail

- SMTP, POP3, IMAP

2.4 DNS

2.5 P2P applications

Some network apps

- web
- e-mail
- text messaging
- P2P file sharing
- remote login
- multi-user network games
- streaming stored video (e.g., YouTube)
- voice over IP (e.g., Skype)
- real-time video conferencing
- social networking
- search
- …
- …

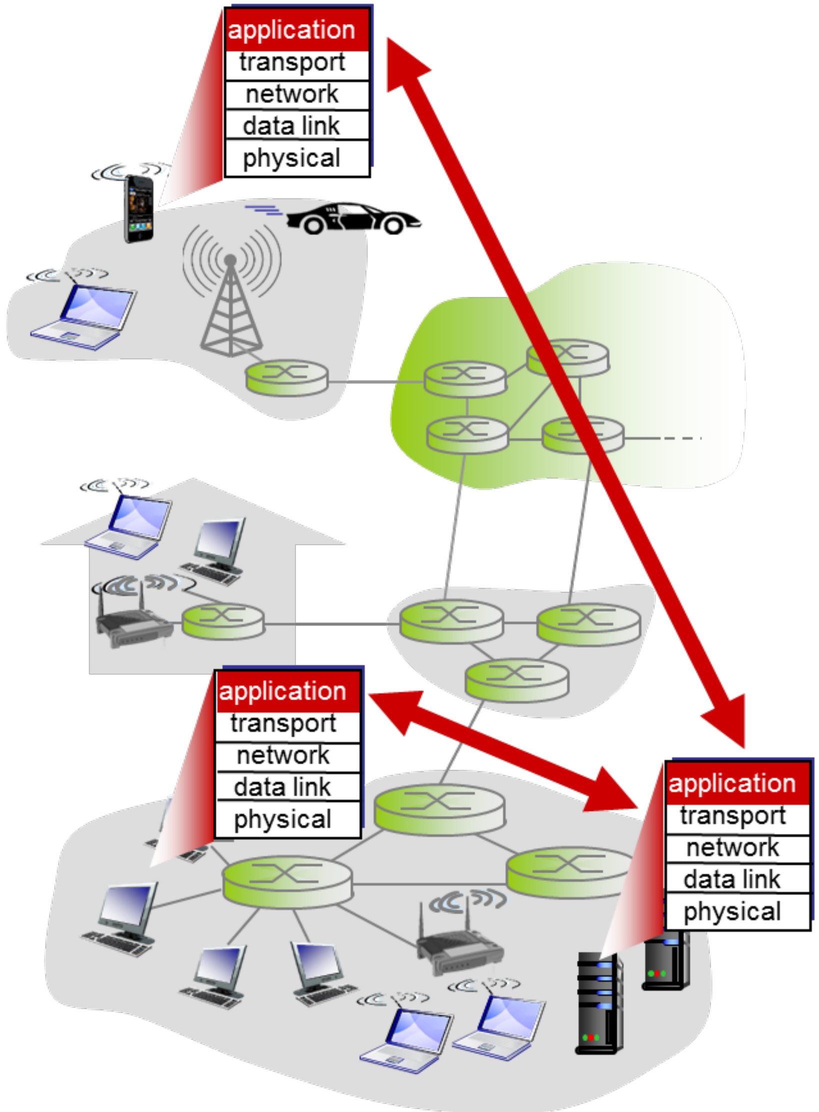
Creating a network app

write programs that:

- run on (different) *end systems*
- communicate over network
  - e.g., web server software communicates with browser software

no need to write software for network-core devices

- network-core devices (e.g., routers) do not run user applications
- 국한시키다
  confining applications to the end systems allows for rapid app development, propagation

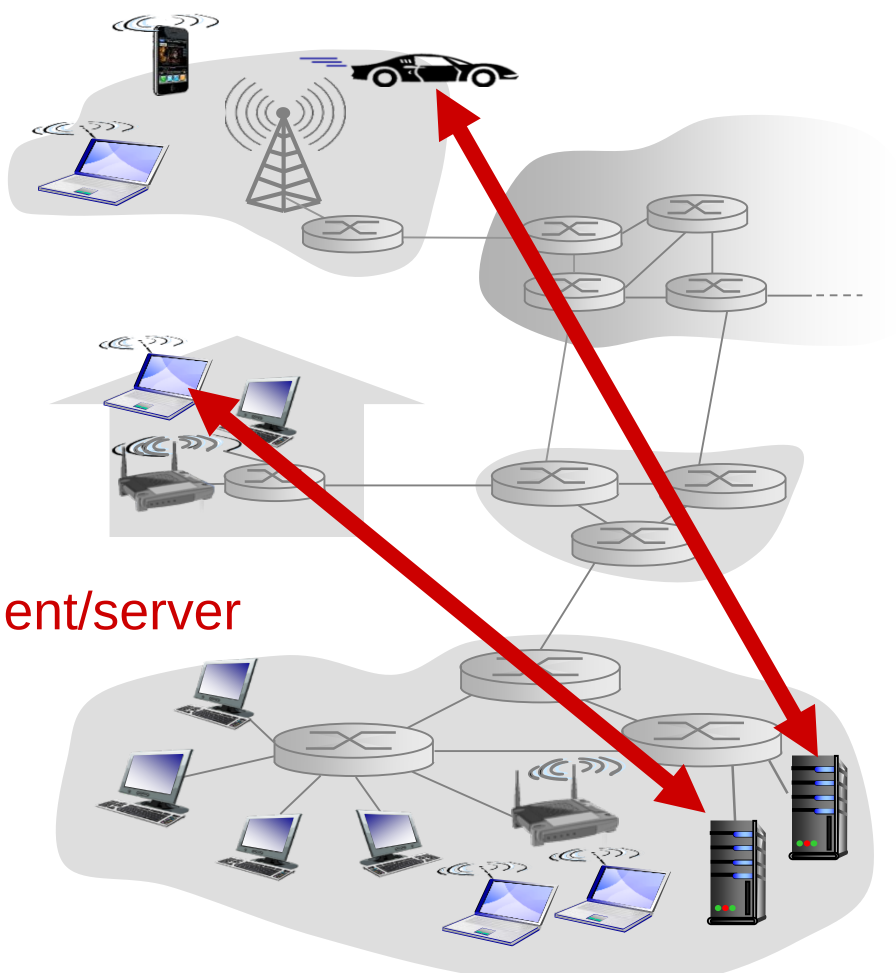


Application architectures

possible structure of applications:

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

Client-server architecture



client/server

server: 

- always-on host, permanent IP address
- data centers for scaling
- e.g., Web servers

clients:

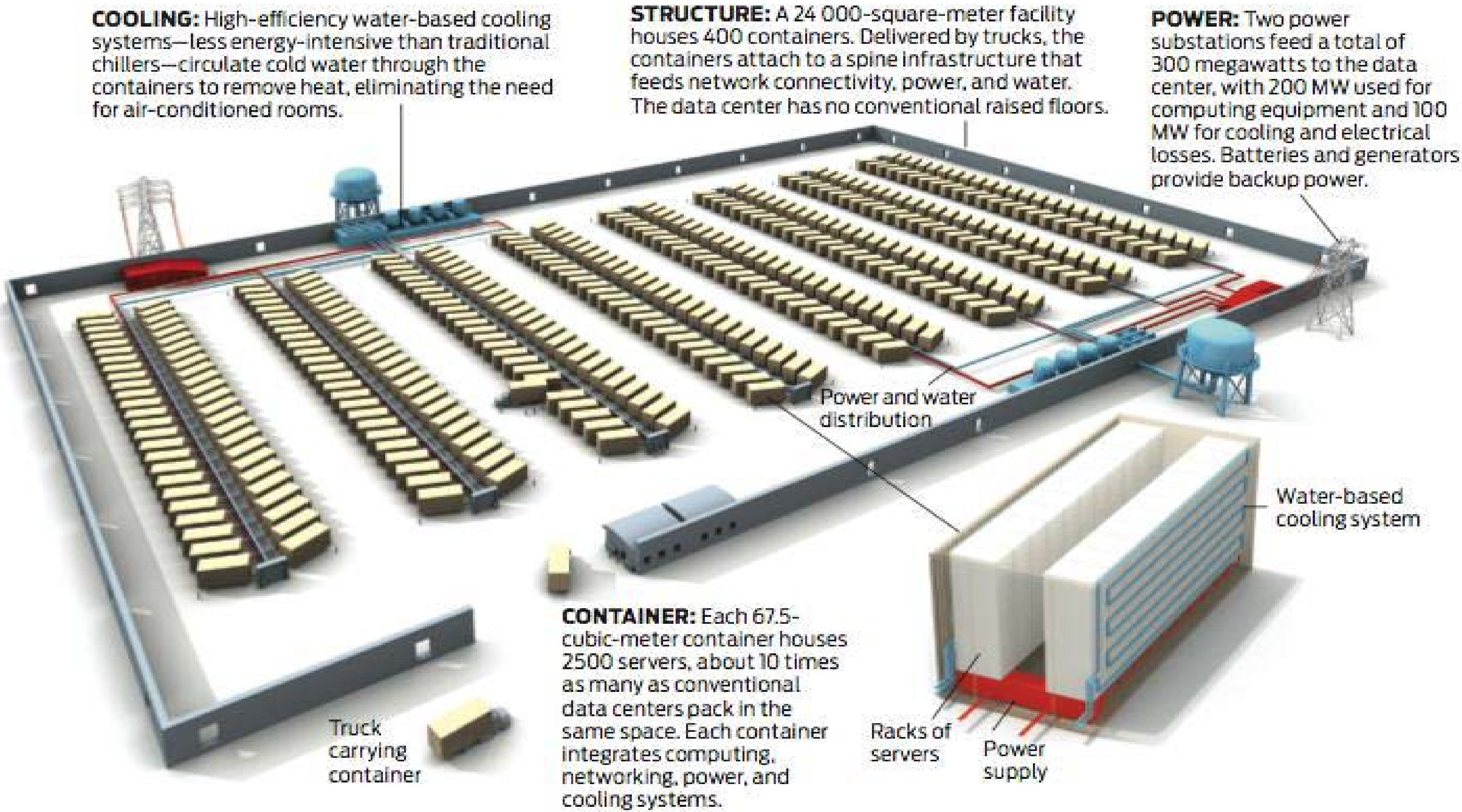
- communicate with server
- 간헐적인
  may be intermittently connected
- may have dynamic IP addresses
- e.g., Web browsers

클라이언트(e.g. 웹서버)는 동적인 ip를 사용. 요청할 때마다 ip 주소를 발급 받아 접속함.

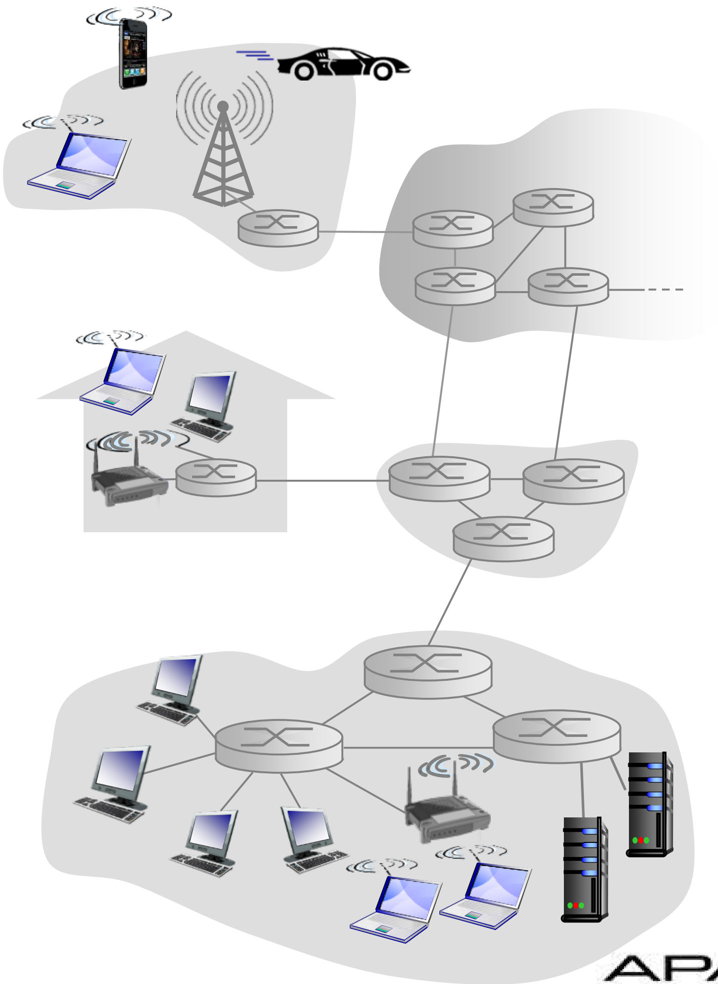
Web Servers?
Data center

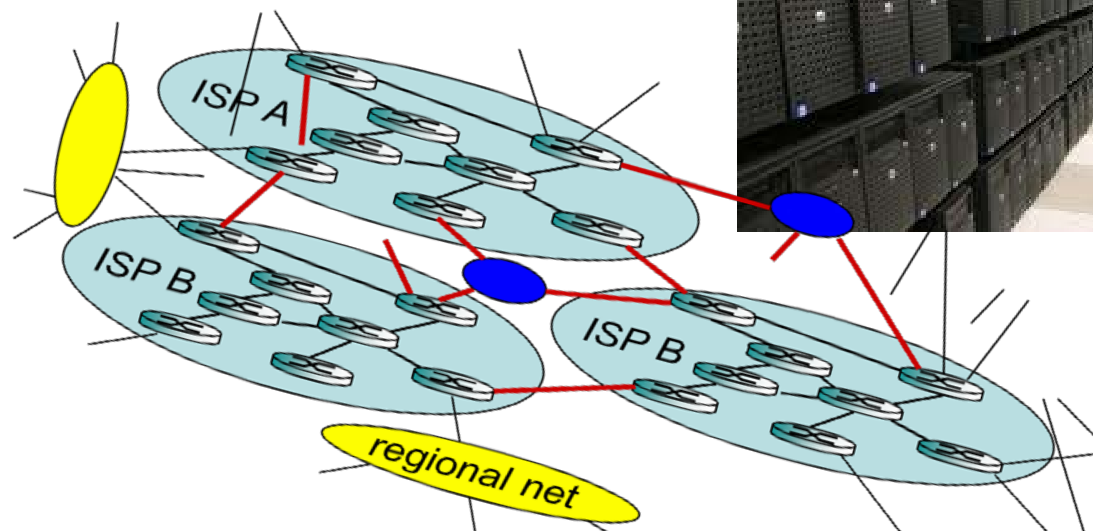
Data center warehouse

가천대학교
Gachon University


COOLING: High-efficiency water-based cooling systems—less energy-intensive than traditional chillers—circulate cold water through the containers to remove heat, eliminating the need for air-conditioned rooms.
STRUCTURE: A 24 000-square-meter facility houses 400 containers. Delivered by trucks, the containers attach to a spine infrastructure that feeds network connectivity, power, and water. The data center has no conventional raised floors.
POWER: Two power substations feed a total of 300 megawatts to the data center, with 200 MW used for computing equipment and 100 MW for cooling and electrical losses. Batteries and generators provide backup power.
Power and water distribution
Water-based cooling system
CONTAINER: Each 67.5-cubic-meter container houses 2500 servers, about 10 times as many as conventional data centers pack in the same space. Each container integrates computing, networking, power, and cooling systems.
Truck carrying container
Racks of servers
Power supply

APACHE
HTTP SERVER





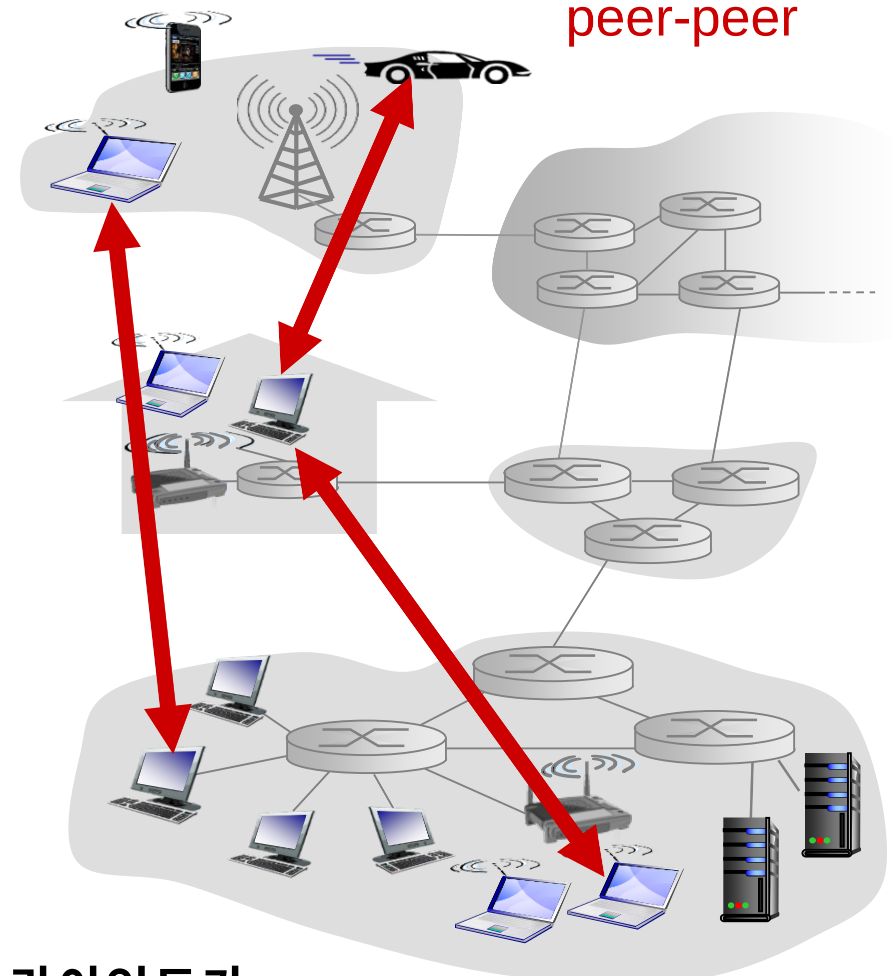
APACHE
HTTP SERVER

P2P architecture

- *no* always-on server
- arbitrary end systems directly communicate

임의적인

- Examples: File sharing (e.g., BitTorrent), Internet Telephony (e.g., Skype)
- peers request service from other peers, provide service in return to other peers
  - *self scalability* – new peers bring new service capacity, as well as new service demands
- peers are intermittently connected and change IP addresses
  - complex management

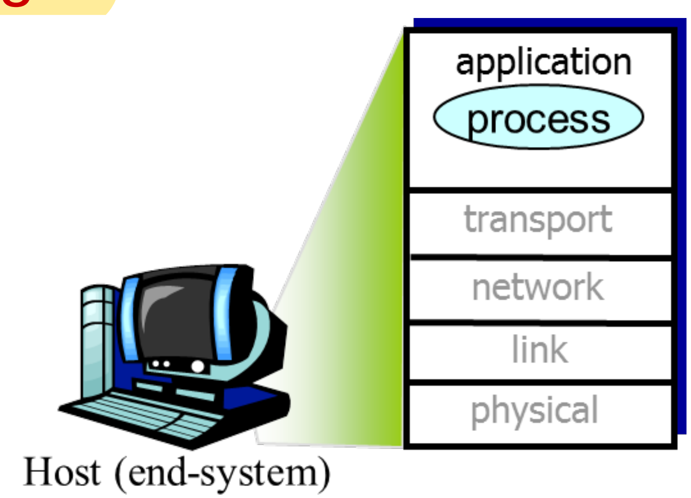


p2p - self scalability : 확장할 필요가 없다. 클라이언트가 늘어남과 동시에 늘어나는 것이니 서버 역할도 한다.

Processes communicating

process: program running within a host

- within same host, two processes communicate using **inter-process communication** (defined by OS)
- processes in different hosts communicate by exchanging **messages**



clients, servers

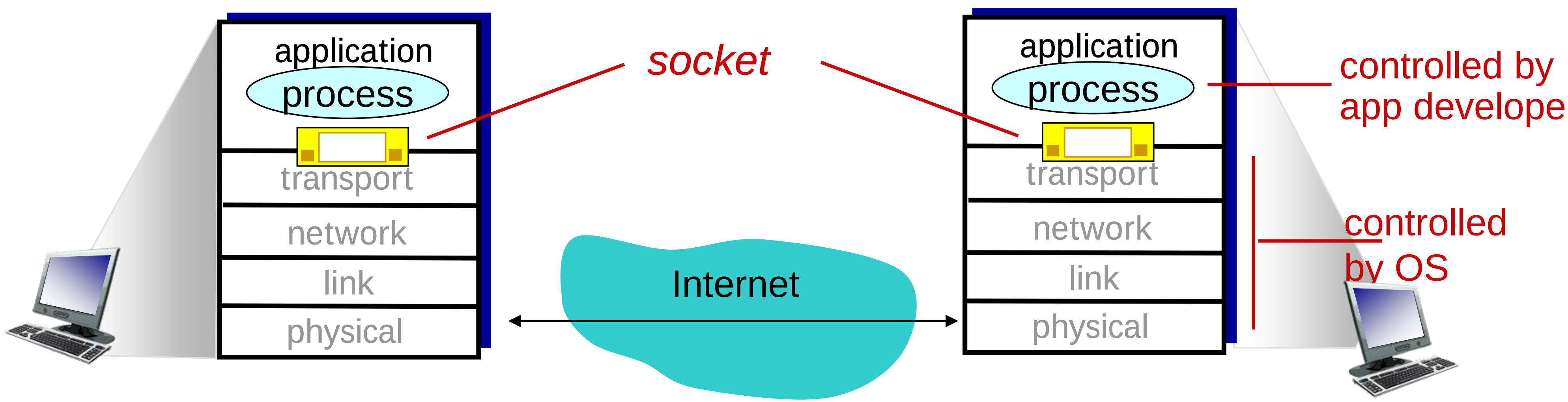
client process: process that initiates communication

server process: process that waits to be contacted

- aside: applications with P2P architectures may have both client processes & server processes

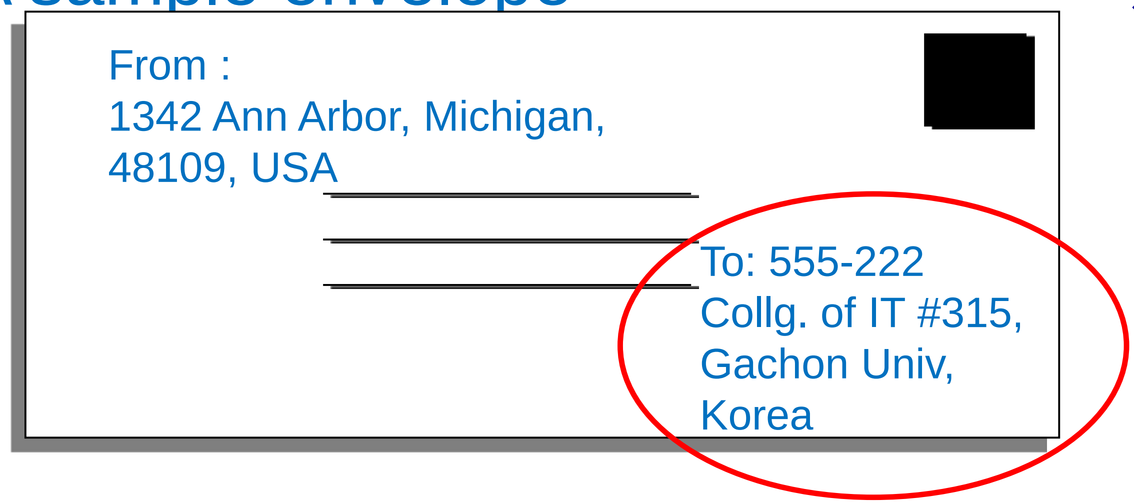
Sockets

- process sends/receives messages to/from its socket
- **Socket** 소켓(api) : application과 transport 사이에 있는 문
  - *Software interface* between the application layer and transport-layer protocol
  - Application Programming Interface (API) between application and network
- Network programming ≈ socket programming

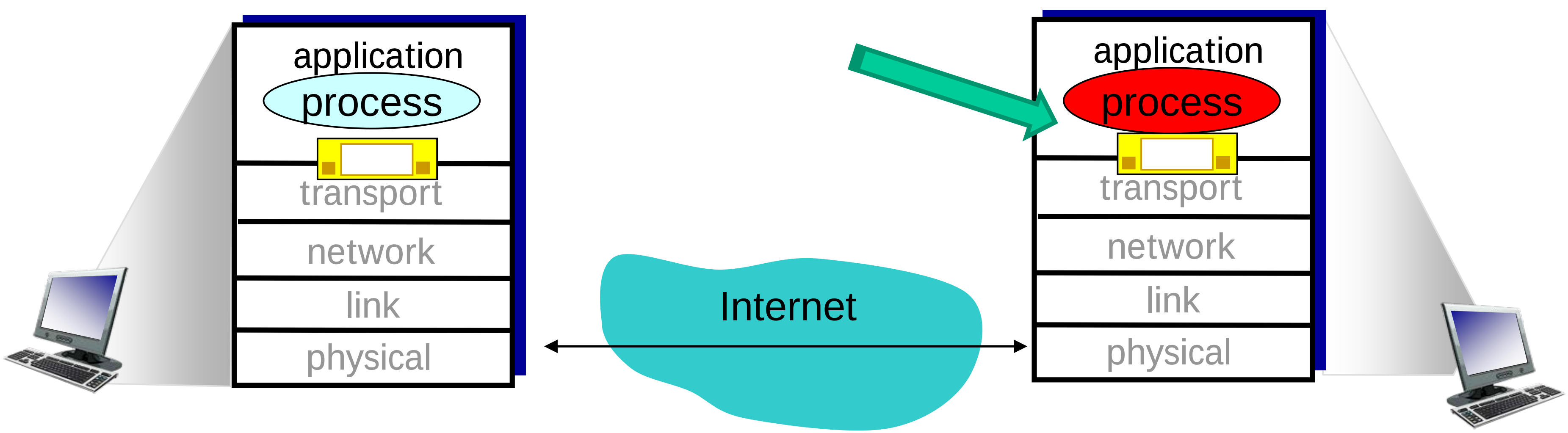


Addressing processes

A sample envelope



- to receive messages, process must have unique *identifier*



Addressing processes (IP address)

- to receive messages, process must have *identifier*
- host device has unique 32-bit **IP address**

identifier : IP 필요
IP만 가지고 다른 host에 있는 것을 알 수 있을까?
상대방의 ip 주소를 알 때 상대방의 host까지 알 수 있음.
process까지 도달해야 되지만 못감
port 번호까지 알아야 process까지 mapping 가능.

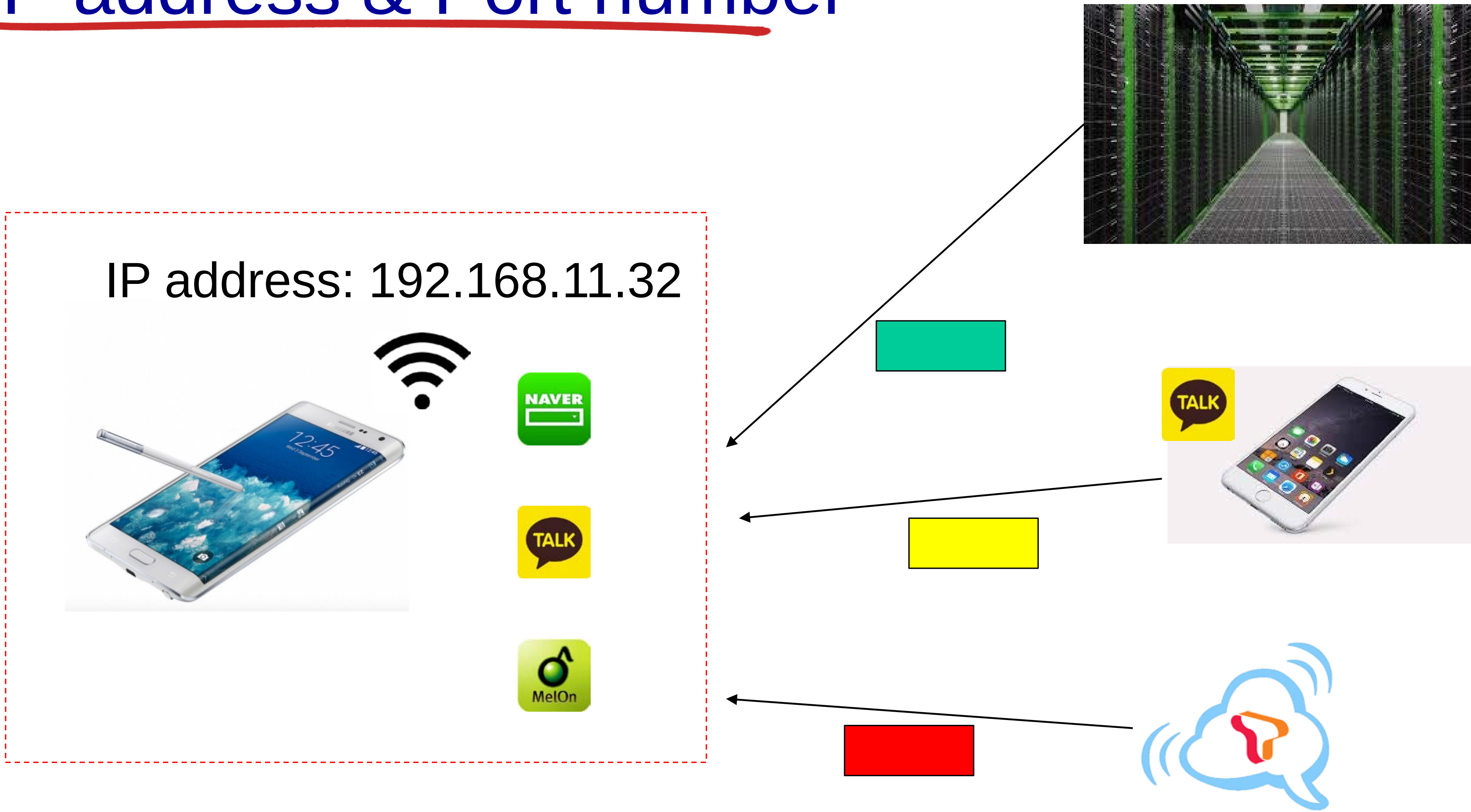
```
C:\Users\jjyoo>ipconfig

Windows IP 구성


이더넷 어댑터 로컬 영역 연결:

   연결별 DNS 접미사. . . . :
   링크-로컬 IPv6 주소 . . . . : fe80::51f3:ffd1:6e46:8d13%14
   IPv4 주소 . . . . . . . . . : 192.168.0.11
   서브넷 마스크 . . . . . . . : 255.255.255.0
   기본 게이트웨이 . . . . . . : 192.168.0.1
```

- *Q:* does IP address of host on which process runs suffice for identifying the process?

충분하다

  - *A:* no, *many* processes can be running on same host

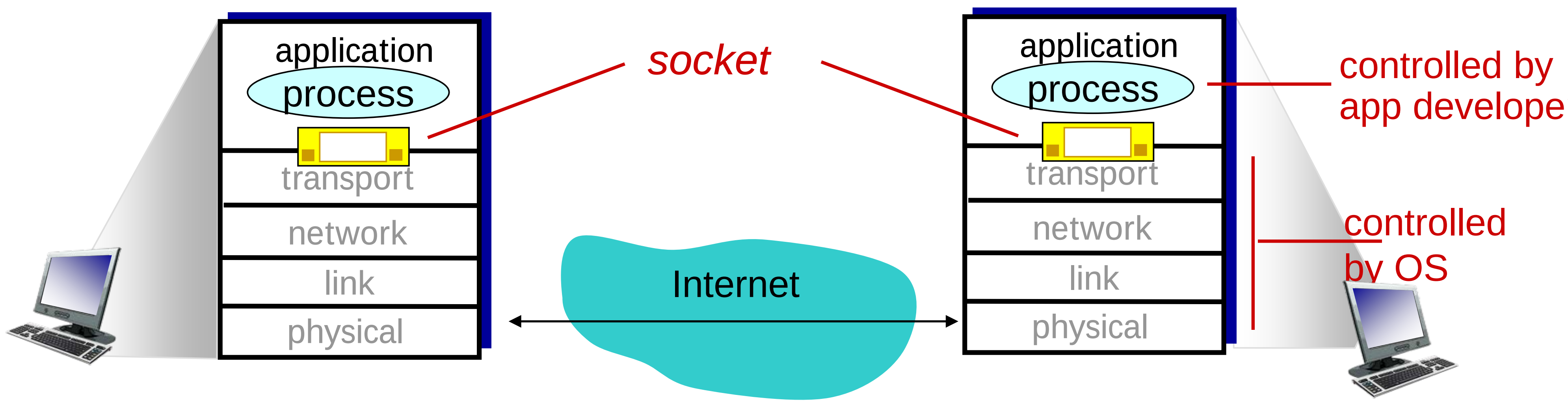
IP address & Port number



Addressing processes

- *identifier* includes both IP address and port numbers associated with process on host.
- example port numbers:
  - HTTP server: 80
  - mail server: 25
- to send HTTP message to sw.gachon.ac.kr web server:
  - IP address: 192.9.89.249
  - port number: 80

Processes communicating (Again)



Two processes can communicate :

- processes in different hosts
  - communicate by exchanging messages

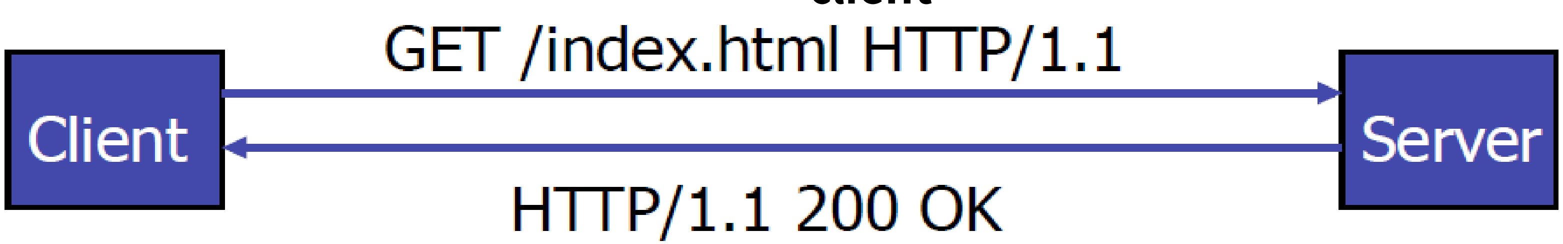
- Message ?? → App-layer protocol

Application-layer Protocols

- Messages exchanged between "applications"
  - Syntax and semantics of the messages between applications (end hosts)　　　　의미
    - Syntax : fields in the message and how the fields are delineated (기술되다)
    - Semantics : the meaning of the information in the field
  - Custom tailored to the specific application (e.g., Web, e-mail)

- Popular application-layer protocols
  - Telnet, FTP, SMTP, NNTP, HTTP, … method / 무엇을 / protocol version

client ========================> server



App-layer protocol defines

- types of messages:
  - e.g., request, response
- message syntax:
  - what fields in messages & how fields are delineated (기술되다)
- message semantics:
  - meaning of information in fields
- rules for when and how processes send & respond to messages

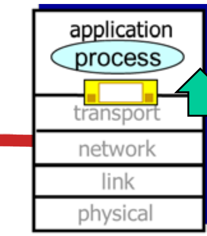
open protocols:
- defined in RFCs
- allows for interoperability
- e.g., HTTP, SMTP

proprietary protocols:
- e.g., Skype

- Application vs. Application Layer protocol
  - Read Chapter 2.1.5 (page 125) in textbook!

proprietary : 어떤 protocol을 쓰는지 open 되어있지 않아 자회사만 있음(자체 프로토콜)

What transport service does an app need?



data integrity (데이터가 파괴 변경되지 않은 상태)

- some apps (e.g., file transfer, web transactions) require 100% reliable data transfer
- other apps (e.g., audio) can tolerate some loss

timing

- some apps (e.g., Internet telephony, interactive games) require low delay to be effective”

Transport service requirements: common apps

application	data loss	time sensitive
file transfer	**no loss**	no
e-mail	**no loss**	no
Web documents	**no loss**	no
audio/video	loss-tolerant	**yes, 100's msec**
interactive games	loss-tolerant	**yes, 100's msec**

Internet transport protocols services

TCP service:

- *reliable transport* between sending and receiving process
- *connection-oriented:* **setup** required between client and server processes

tcp : 연결을 맺으니 연결을 맺는 delay가 필요.
잘못 받았으면 다시 보내줌
udp service : 주소를 아니까 send하고 끝.
손실이 나도 상관 없으니 서로 상태를 관리할 필요x.
더 빠르게 보낼 수 있음

UDP service:

- *unreliable data transfer* between sending and receiving process
- *does not provide:* reliability, or connection setup,

Q: why bother? Why is there a UDP? (Discussed further in Ch. 3)

Internet apps: application, transport protocols

application	application layer protocol	underlying transport protocol
e-mail	SMTP [RFC 2821]	TCP
remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
file transfer	FTP [RFC 959]	TCP
streaming multimedia	HTTP (e.g., YouTube), RTP [RFC 1889]	TCP or UDP
Internet telephony	SIP, RTP, proprietary (e.g., Skype)	TCP or UDP

사람들이 multi device기 때문에
IoT를 위해서 하는건 개발중이지만 웹 프로토콜을 사용하는중

Chapter 2: outline

2.1 principles of network applications
- app architectures
- app requirements

2.2 Web and HTTP

2.3 electronic mail
- SMTP, POP3, IMAP

2.4 DNS

2.5 P2P applications

Web and HTTP

First, a review…

- *web page* consists of *objects*
  - Web objects: HTML file, JPEG image, Java applet, video/audio file,…
- *HTML-file* includes several referenced *objects*
  - each object is addressable by a *URL,* e.g.,

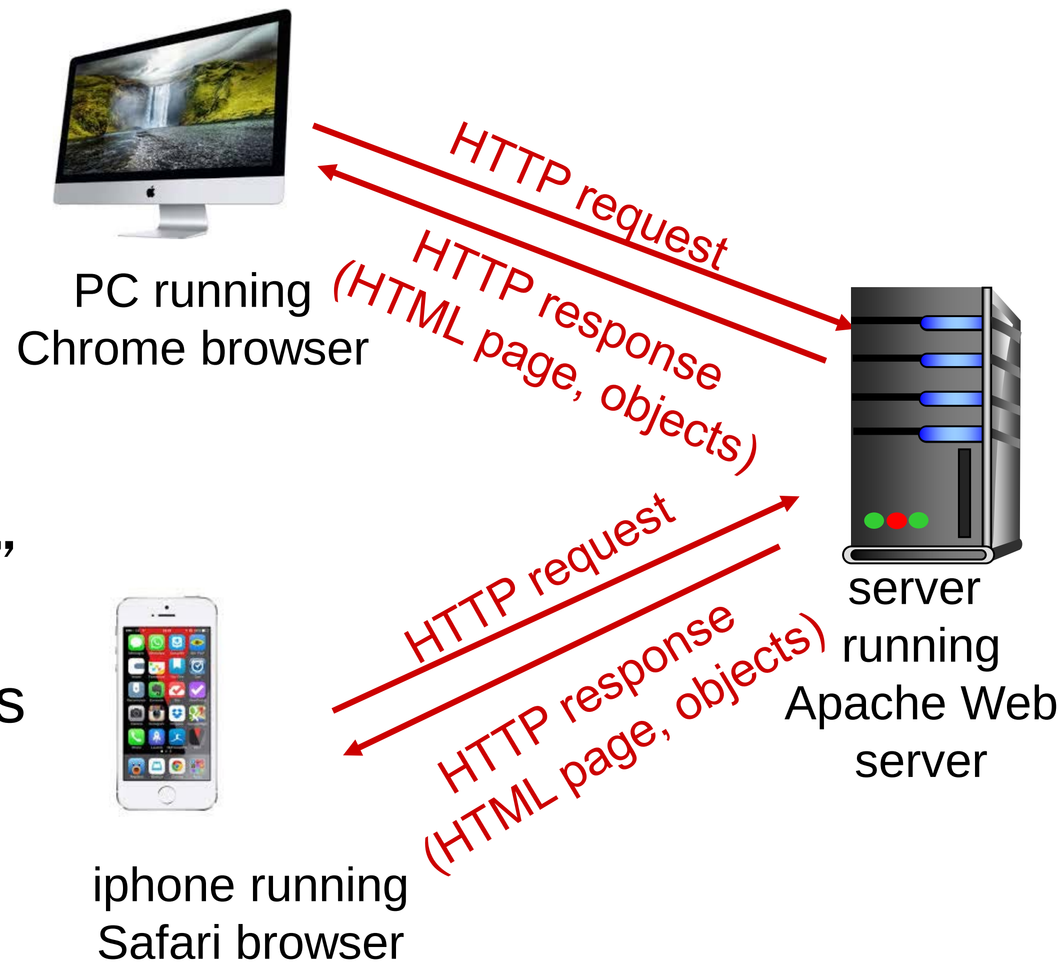
$$\underbrace{\texttt{www.someschool.edu}}_{\text{host name}}\underbrace{\texttt{/someDept/pic.gif}}_{\text{path name}}$$

<img src="URL">…

HTTP overview

HTTP: hypertext transfer protocol

- Web's application layer protocol
- client/server model
  - *Web client:* browser that requests, receives, (using HTTP protocol) and "displays" Web objects
  - *Web server:* Web server sends (using HTTP protocol) objects in response to requests



http는 request를 받으면 그에 대응되는 response를 보냄

HTTP overview (continued)

uses TCP:

- client initiates TCP connection (creates socket) to server, port 80
- server accepts TCP connection from client
- HTTP messages (application-layer protocol messages) exchanged between browser (HTTP client) and Web server (HTTP server)
- TCP connection closed

*HTTP is **stateless***

- server maintains no information about past client requests

tcp를 관리(state를 기억)
계속 기억하면 좋겠지만 그때마다 state를 생성한다면
connection을 받으면 없어짐 connection을 받고 닫아야한다.
나중에 또 request가 올 수 있는데 그 다음에 request가 와도
누구의 request인지 state를 기억하지 않는다.
서버를 최대한 가볍게 해주자. 그냥 response만 해주고 역할 끝

HTTP connection

suppose user enters URL:
`www.someSchool.edu/someDepartment/home.index`

(contains text, references to 10 jpeg images)

1a. HTTP client initiates **TCP connection** to HTTP server (process) at www.someSchool.edu on port 80

1b. HTTP server at host www.someSchool.edu waiting for **TCP connection** at port 80. "**accepts**" connection, notifying client

2. HTTP client sends ***HTTP request message*** (containing URL) into TCP connection socket. Message indicates that client wants object someDepartment/home.index

3. HTTP server receives request message, forms ***response message*** containing requested object, and sends message into its socket

time

HTTP connection(cont.)

4. HTTP server **closes TCP connection**.

5. HTTP client **receives response message** containing html file, displays html. Parsing html file, finds 10 referenced jpeg objects

time

6. Steps 1-5 repeated for each of 10 jpeg objects

HTTP connection: response time

Tx. delay?

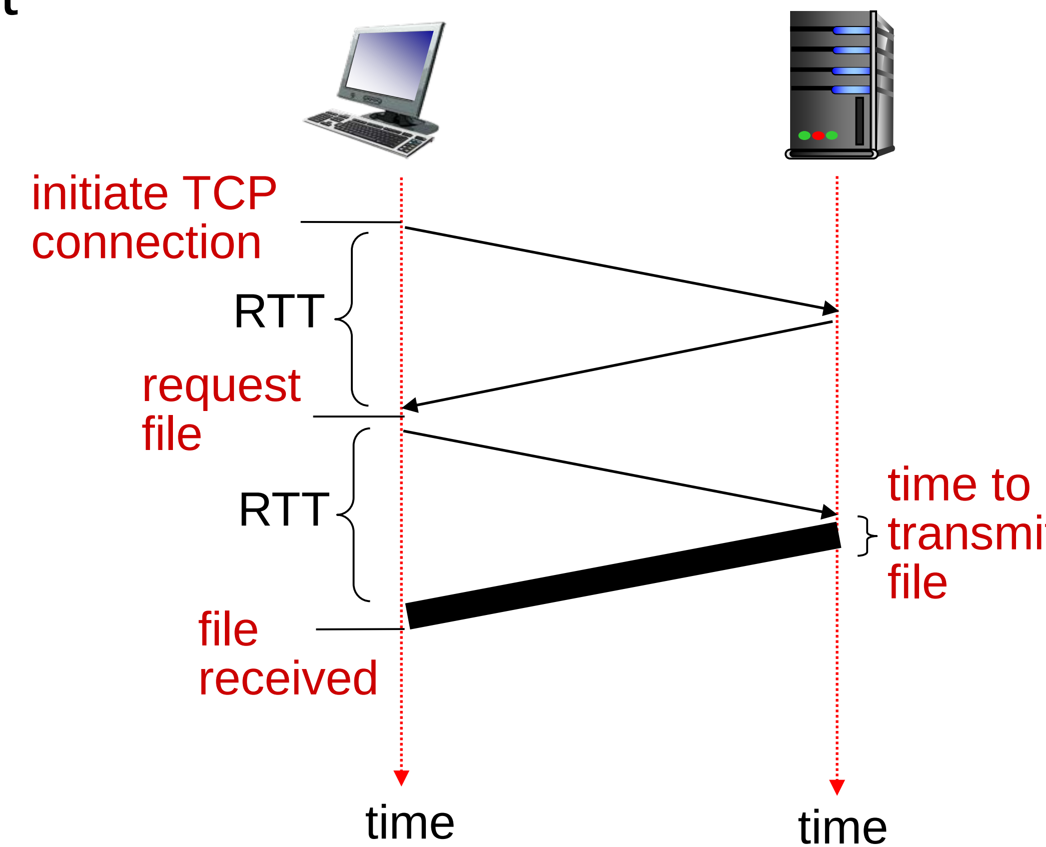
Round-trip time (RTT): time for a small packet to travel from client to server and back

- What type of delays does RTT include? (Recall Ch. 1.4)

HTTP response time:

- one RTT to initiate TCP connection
- one RTT for HTTP request and first few bytes of HTTP response to return
- file transmission time (why include this?)
- non-persistent HTTP response time =

  2RTT+ file transmission time



RTT : transmission delay, processiong delay은 적음. - propagation delay, queueing delay가 dominant하게 작용함.
굵은 선 : transmission delay (object가 실려 오기 때문에) 저만큼 transmission time 필요

HTTP

HTTP issues:

- requires 2 RTTs per object
- browsers often open parallel TCP connections to fetch referenced objects
  - Typically 5~10 TCP connections

http issues : 너무 느리다.
- 극복 방법 : tcp(http???) connection 을 parallel하게(동시 다발적). (이 방법만으론 충분치 않음)
- +default tcp(non-persistent http) 말고 persistent http를 제안하게 됨. 연결 끊지 않고 재활용

HTTP connections

non-persistent HTTP

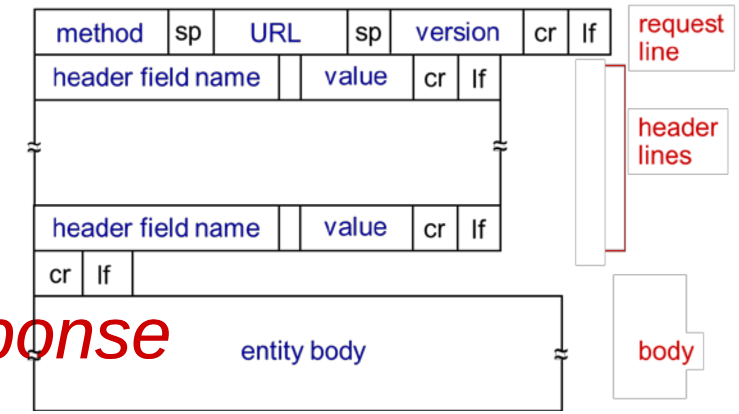
- at most one object sent over TCP connection
  - connection then closed
  - requires 2 RTTs per object
- downloading multiple objects required multiple connections

persistent http : 많이 사용하면 부하 걸림
request가 올지 안 올지 모름. 연속적일지도 모름.
persistent를 쓰면 일단 connection을 날리지 않고
일정 시간(time out value)동안

persistent HTTP

- server leaves connection open after sending response
  - **연속적인** subsequent HTTP messages between same client/server sent over open connection
  - client sends requests as soon as it encounters a referenced object
- multiple objects can be sent over single TCP connection between client, server
- HTTP server closes connection after certain time (timeout)

HTTP request message



- two types of HTTP messages: *request, response*
- HTTP request message:
  - ASCII (human-readable format)
    - Read Chapter 2.2.3 (page 131-136) in textbook!

carriage return character

line-feed character

request line (GET, POST, HEAD commands)

http 1.1 버전을 이용하여 index.html 오브젝트를 가지고 싶다

```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\r\n
User-Agent: Firefox/3.6.10\r\n
Accept: text/html,application/xhtml+xml\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7\r\n
Keep-Alive: 115\r\n
Connection: keep-alive\r\n
\r\n
```

호스트의 ip 주소

브라우저 타입

header lines

호스트들의 부가적인 정보들이 header line에 포함.

carriage return, line feed at start of line indicates end of header lines

₩rr

Method types

HTTP/1.0:

객체의 갱신

- GET
- POST
  - web page often includes form input
  - input is uploaded to server in entity body

회원가입할 때 생년월일

- HEAD
  - asks server to leave requested object out of response

object가 없는 response를 요청할때
서버 상태를 확인 or 디버깅 등

Response의 body를 제외한 header 만 요구

HTTP/1.1:

객체의 추가나 업로드

- GET, POST, HEAD
- PUT
  - uploads file in entity body to path specified in URL field

클라이언트가 서버에
file을 제공함 (보통과는 달리)

- DELETE
  - deletes file specified in the URL field

서버에 있는 file을 클라이언트가
삭제할 수 있는 method

가천대학교 Gachon University

HTTP response message

200 OK : 너가 request한 것을 잘 해결했다는 메세지

status line
(protocol
status code
status phrase)

header
lines

Persistent or non-persistent?

keep-alive : persistent

data, e.g.,
requested
HTML file

```
HTTP/1.1 200 OK\r\n
Date: Sun, 26 Sep 2010 20:09:20 GMT\r\n
Server: Apache/2.0.52 (CentOS)\r\n
Last-Modified: Tue, 30 Oct 2007 17:00:02
   GMT\r\n
ETag: "17dc6-a5c-bf716880"\r\n
Accept-Ranges: bytes\r\n
Content-Length: 2652\r\n
Keep-Alive: timeout=10, max=100\r\n
Connection: Keep-Alive\r\n
Content-Type: text/html; charset=ISO-8859-1\r\n
\r\n
(data data data data data ... )
```

가천대학교
Gachon University

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 msg

`301 Moved Permanently`

- requested object moved, new location specified later in this msg (Location:)

`400 Bad Request`

- request msg not understood by server

`404 Not Found`

- requested document not found on this server

`505 HTTP Version Not Supported`

Cookies: keeping state at server

But sometimes server wishes to…
- ✓ restrict user access
- ✓ serve content as a function of user identity
- ☞ **Cookies** allow sites to keep track of users

many Web sites use cookies

four components:

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

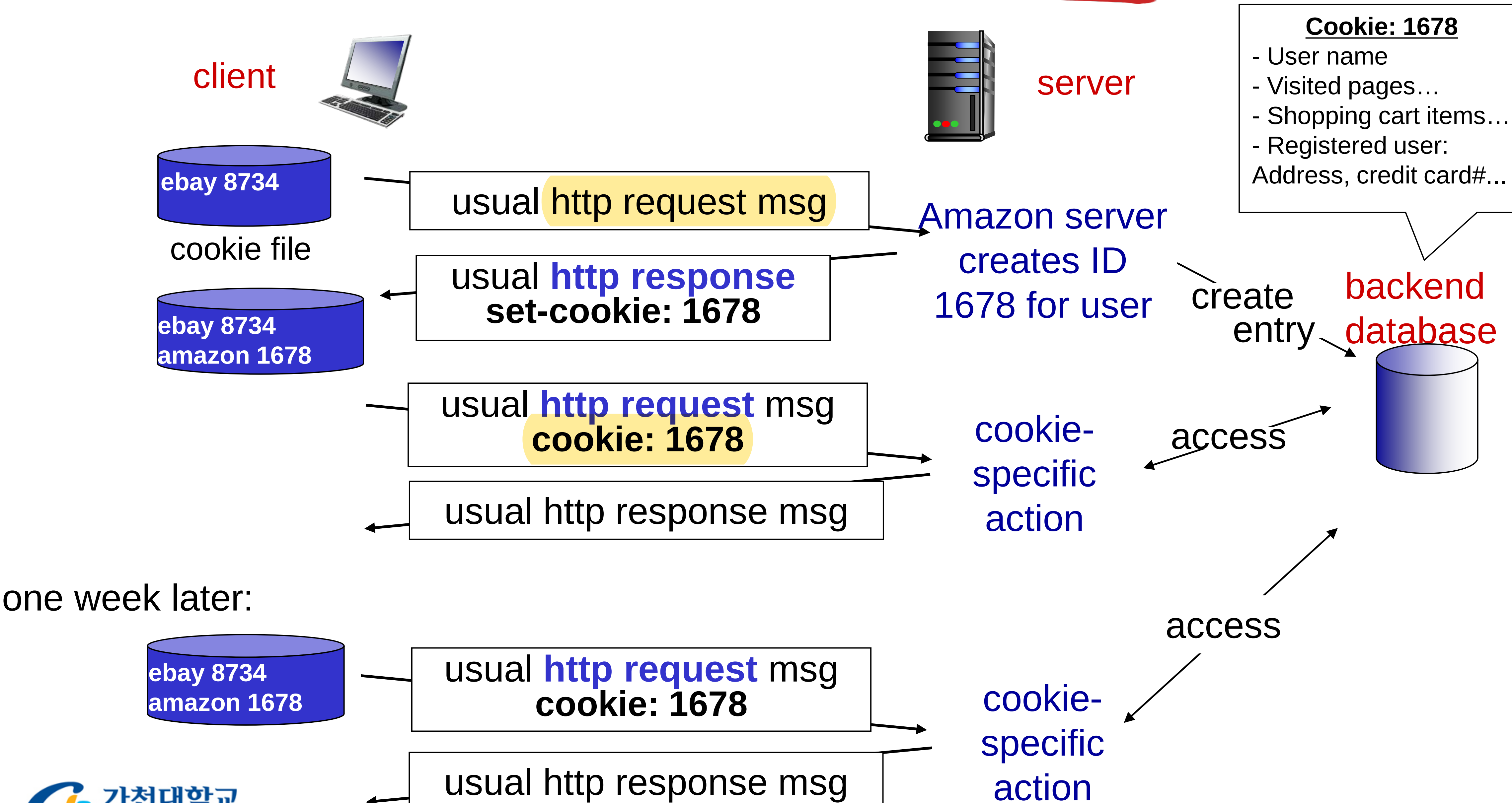
쿠크 헤더에 이 사람의 쿠키 번호를
부여하여 유저 상태를 계속 추적

example:

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

자신이 request를 보낼때마다 자신의
cookie id를 보내 서버에게 알려준다
서버가 cookie id를 어떤 host인지 mapping 해둠
-> 유저에 맞는 맘춤 서비스

Cookies: keeping "state" (cont.)



Cookies (continued)

what cookies can be used for:

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

aside

cookies and privacy:

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

how to keep "state":

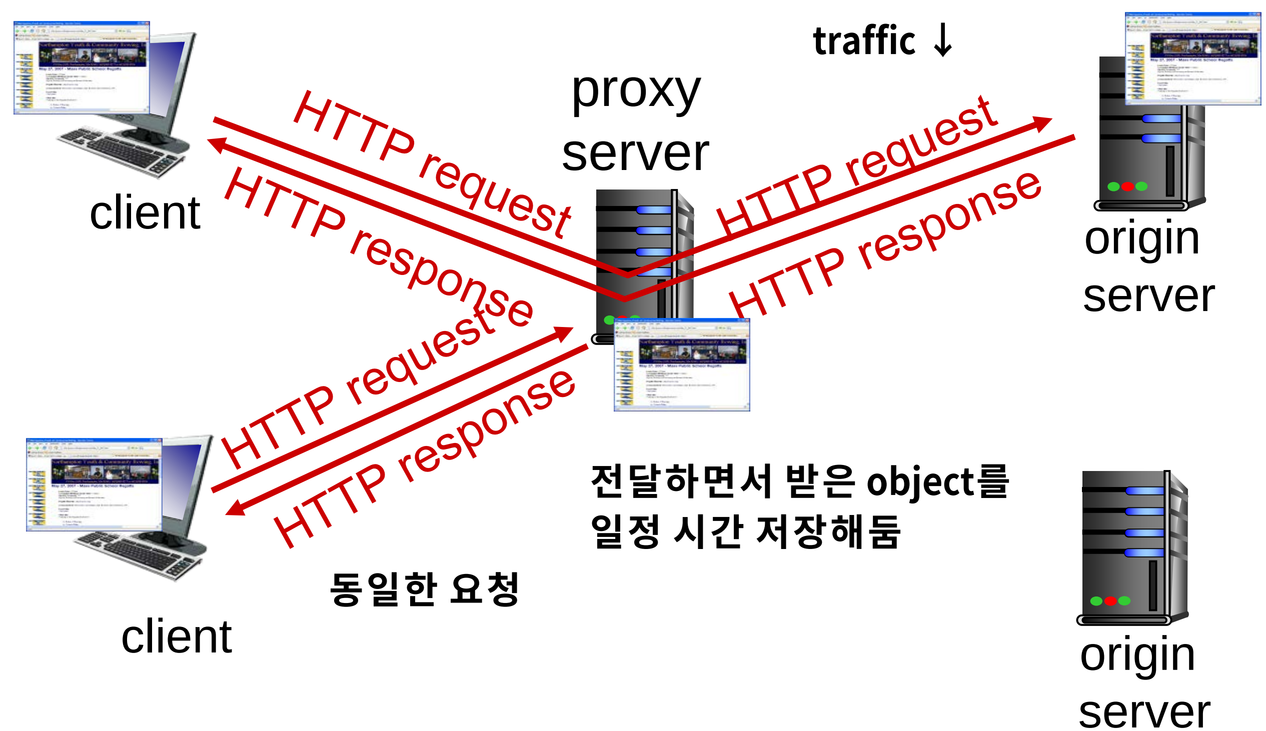
- protocol endpoints: maintain state at sender/receiver over multiple transactions
- cookies: http messages carry state

Web caches (proxy server)

빨리 웹사이트에 접속하기 위해

goal: satisfy client request without involving origin server

- 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



사람들이 자주 사용하는 것을 저장
-> web caches

More about Web caching

- cache acts as both client and server
  - server for original requesting client 그 object를 가지고 있을 때
  - client to origin server 가지고 있지 않을 때
- typically cache is installed by ISP (university, company, residential ISP)

why Web caching? 상대적으로 가까이에 있는 web cache에게 받는게 반응속도가 빠름

- reduce **response time** for client request
- reduce **traffic** on an institution's access link
- Internet dense with caches: enables "poor" content providers to effectively deliver content (so too does P2P file sharing)

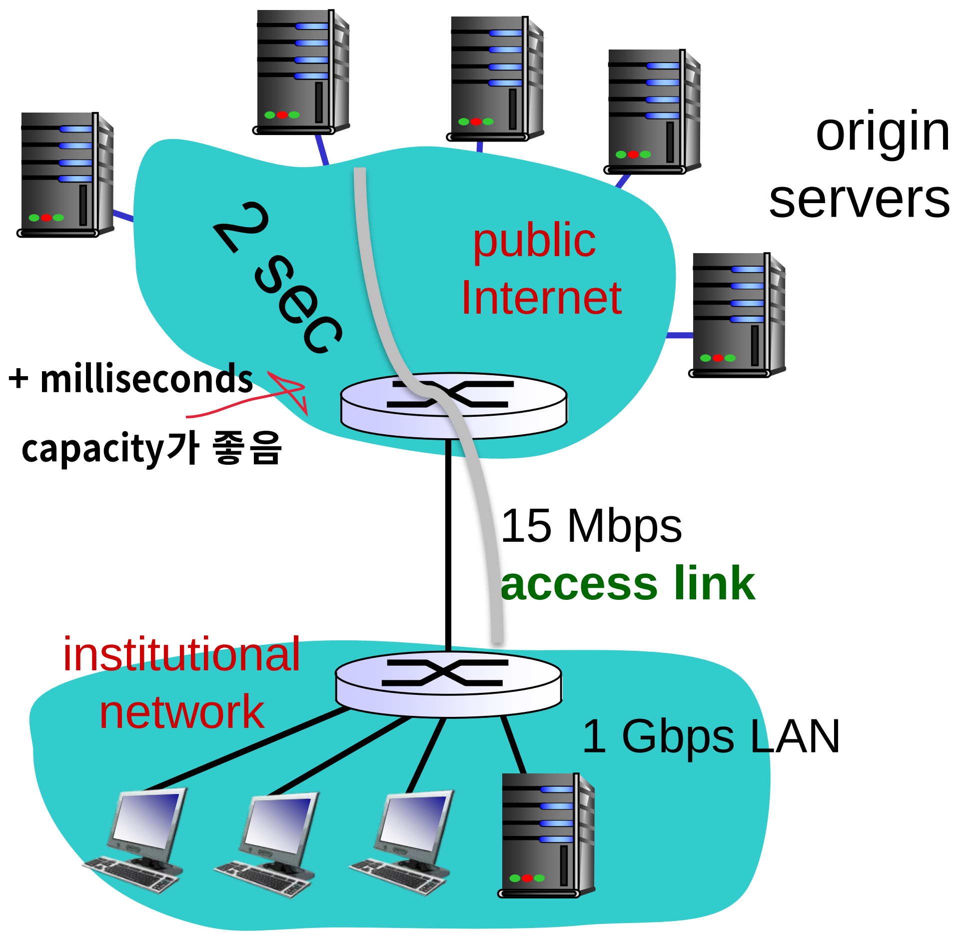
Caching example:

Assumptions:

- avg object size: 1-Mbits
- avg request rate from browsers to origin servers:15/sec 1M x 15/s
- avg data rate to browsers: 15 Mbps
  - RTT from institutional router to any origin server: 2 sec
- **access link** rate: 15 Mbps

Consequences:

- LAN utilization**: 1.5%**
- access link utilization >= 99%
- Total delay = Internet delay + access delay + LAN delay
  = 2 sec + minutes + milliseconds = **minutes**



Possible solution: fatter access link

assumptions:

- avg object size: 1-Mbits
- avg request rate from browsers to origin servers:15/sec
- avg data rate to browsers: 15 Mbps
  - RTT from institutional router to any origin server: 2 sec
- access link rate: 15 Mbps

100 Mbps

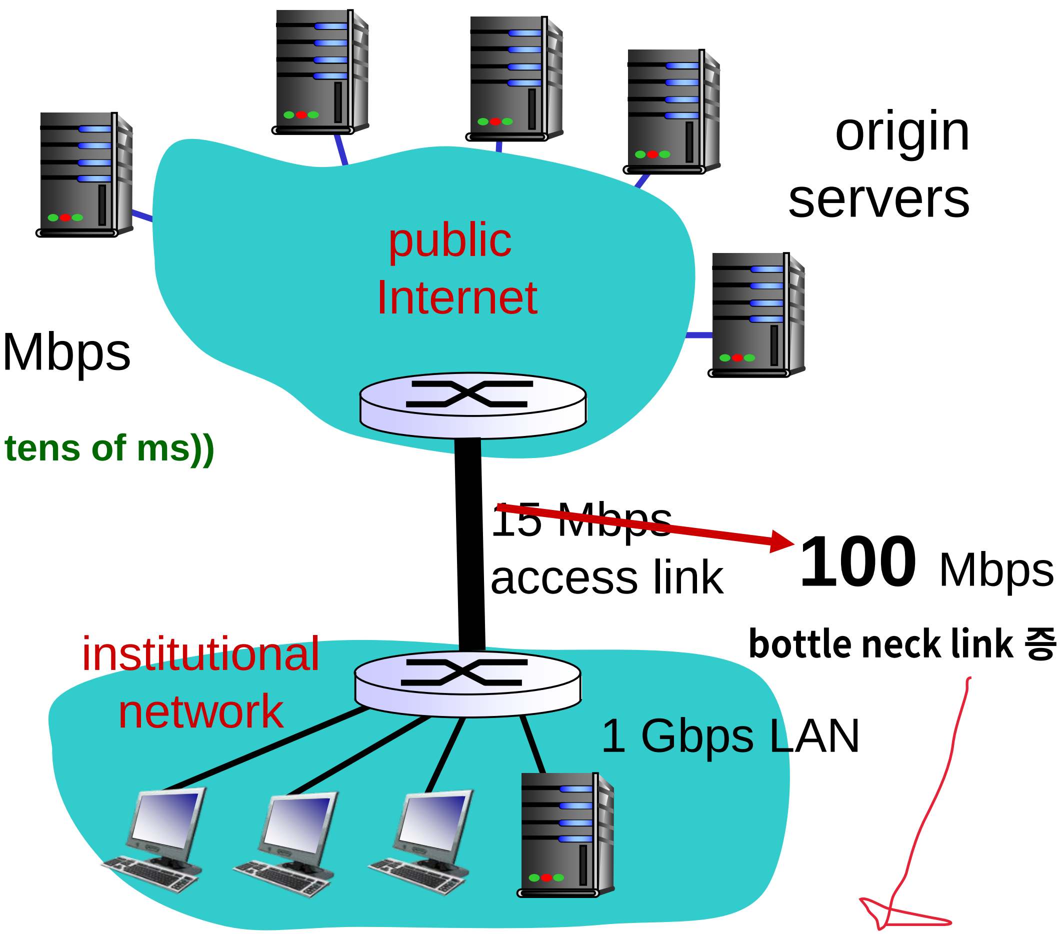
(= **bottle neck**)

15% **(small delay (e.g., tens of ms))**

consequences:

- LAN utilization: 1.5%
- access link utilization = 99%
- total delay = Internet delay + access delay + LAN delay
  = 2 sec + minutes + msecs = **~3 secs**

msecs



bottle neck link 증설

비용이 큼

Cost: increased access link speed (not cheap!)

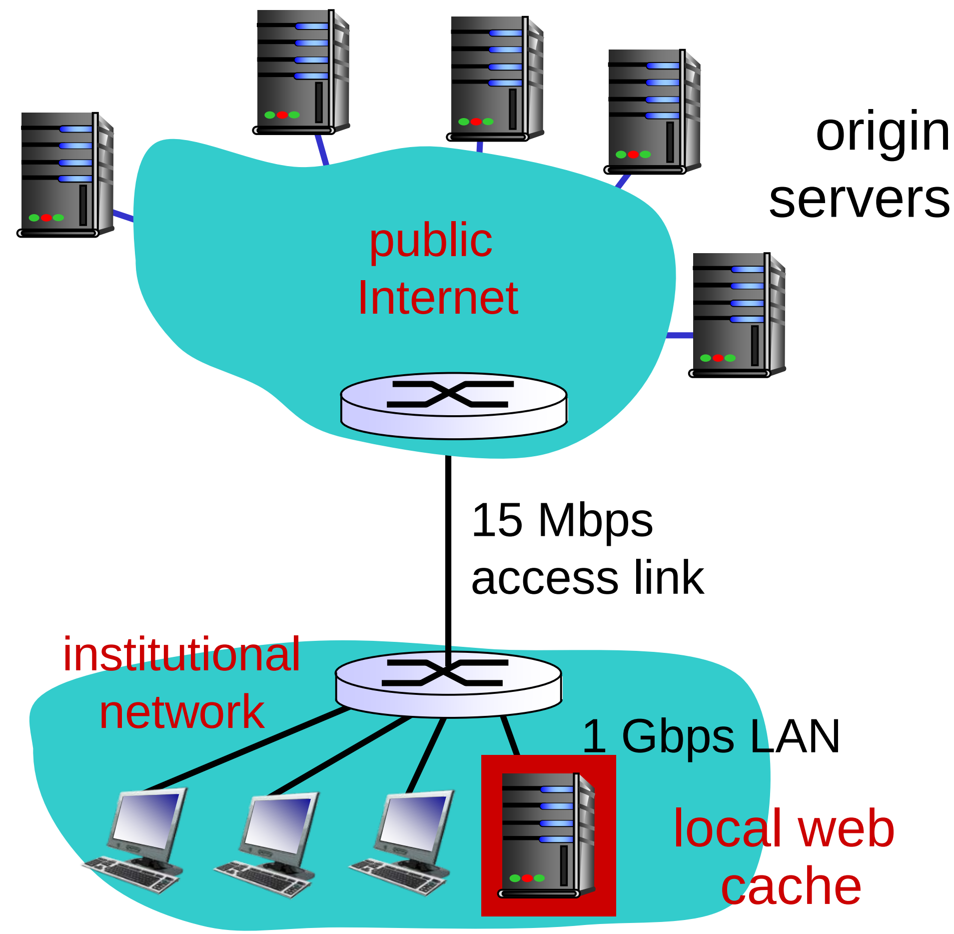
Caching example: install local Web cache

assumptions:

- avg object size: 1-Mbits
- avg request rate from browsers to origin servers:15/sec
- avg data rate to browsers: 15 Mbps
  - RTT from institutional router to any origin server: 2 sec
- access link rate: 15 Mbps

consequences:

- LAN utilization: 1.5%
- access link utilization =
- total delay = Internet delay + access delay + LAN delay =

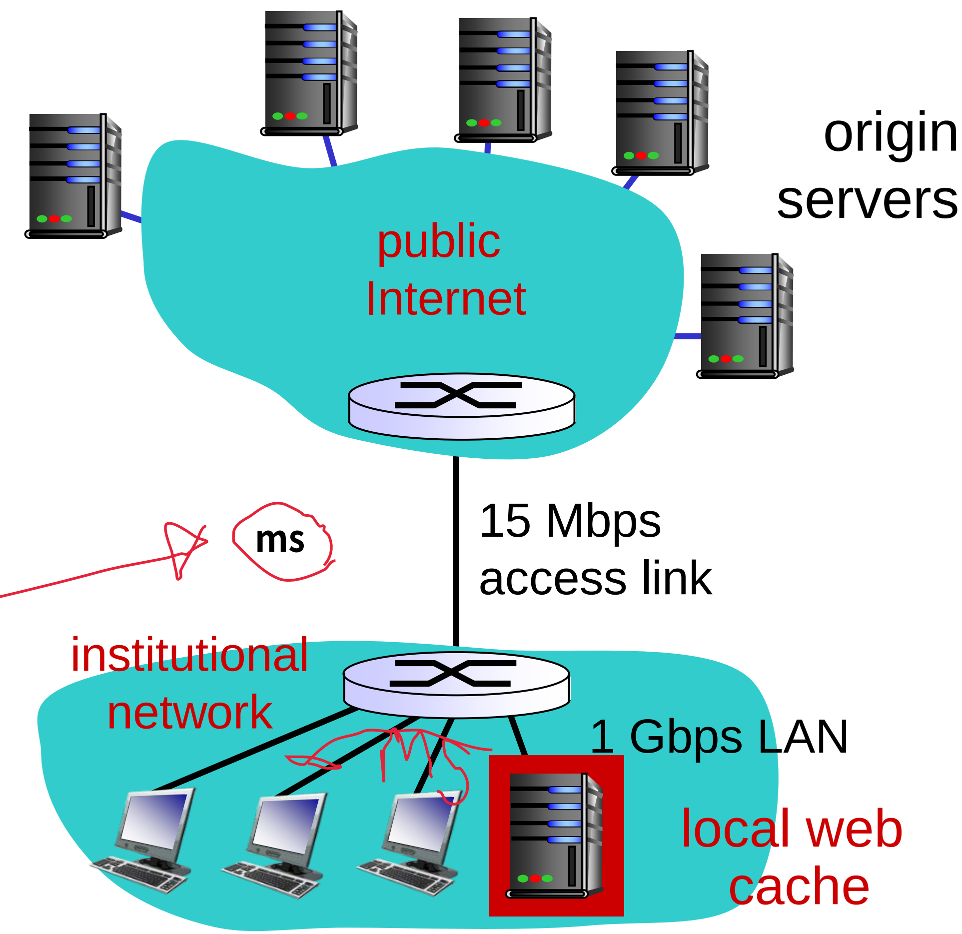


Cost: web cache (cheap!)

Caching example: install local cache

Calculating access link utilization, delay with cache:

- suppose cache hit rate is **0.4 (0.2~0.7 in practice)**
  - 40% requests will be satisfied almost immediately !!
  - 60% requests satisfied by origin server
- access link utilization:
  - 60% of requests use access link
- data rate to browsers over access link = 0.6*15 Mbps = 9 Mbps
  - utilization = 9/15 = **0.6** (from 0.99)
- total delay **2초 + ms**
  - = 0.6 * (delay from origin servers) +0.4 * (delay when satisfied at cache)
  - = 0.6 (2.01) + 0.4 (0.01) = **~ 1.2 secs**
  - less than with 100 Mbps link (~3 secs) and cheaper too!



- reduce **response time** for client request
- reduce **traffic** on an institution's access link

cache hit / miss

link capacity를 올리는 것보다 싸고 성능이 좋다

가천대학교 Gachon University