Networks and Communications

"The Application Layer"

Konstantinos Gkoutzis
Imperial College

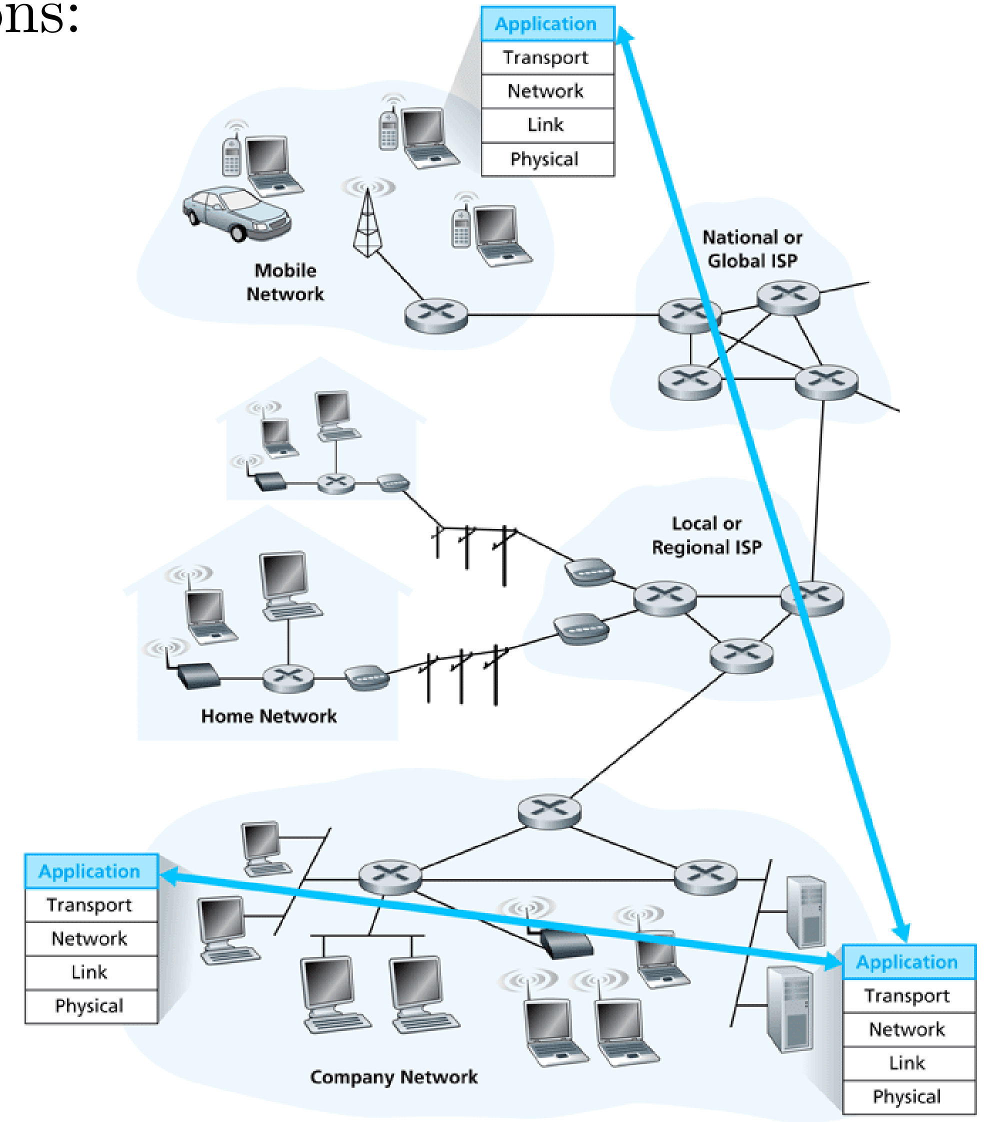
Outline

Imperial College London

- World Wide Web
- Domain Name System
- Content Delivery Networks
- Electronic Mail
- (Other Applications of the Internet)

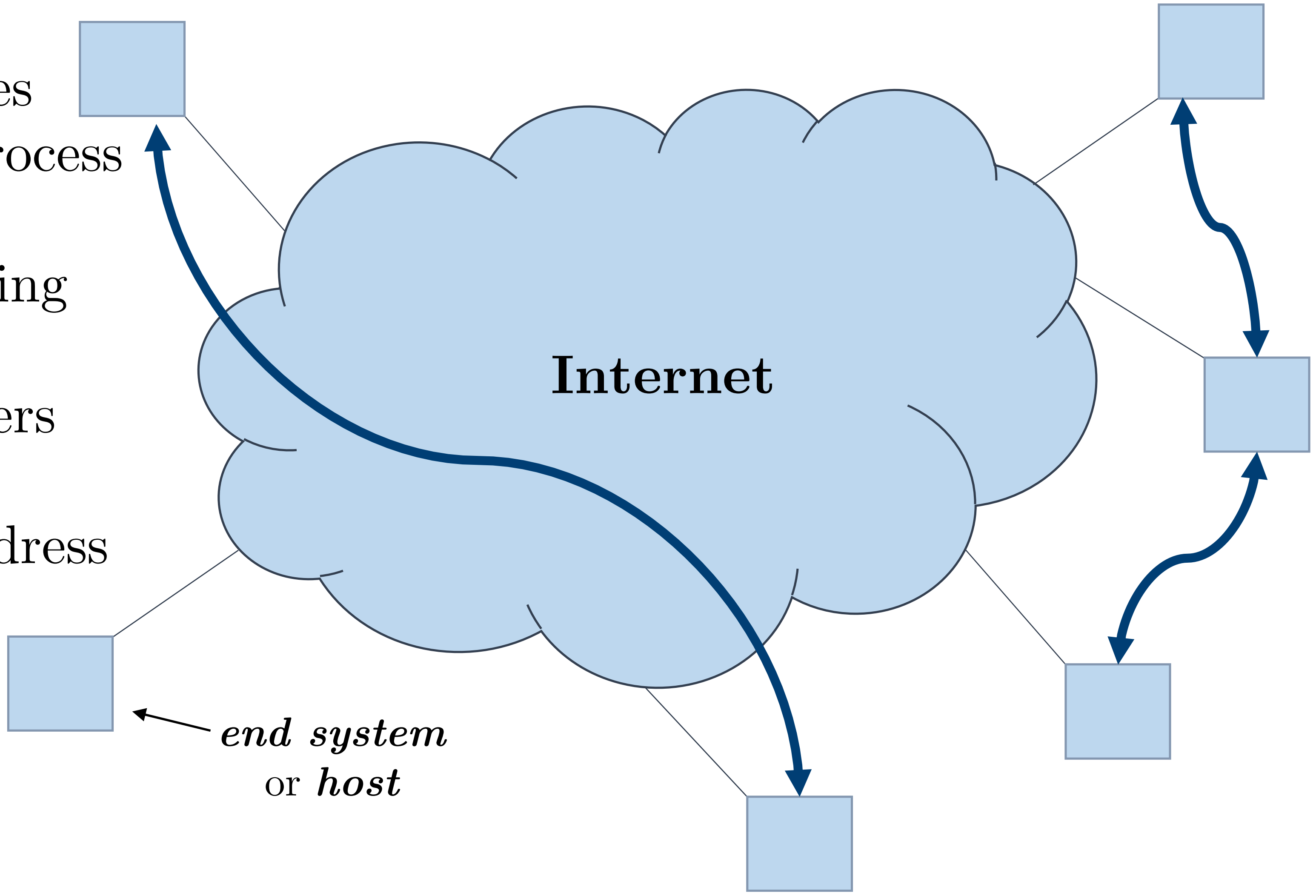
Introduction

- Now, we will look at some real network applications:
- World Wide Web
- Domain Name System
- Content Delivery Networks
- Electronic Mail
- ...and more!



End Systems Applications

- Internet applications are end system applications (*or processes*)
- Processes may exchange messages
  - messages act as input to a process
- Different processes may be running on different end systems
  - possibly on different computers
  - running different OSes
  - a process must be able to address another process



Example #1 (HTTP)

Imperial College London

```
while(browsing) {
    url = read_url(keyboard);
    page = get_web_page(url);
    display_web_page(page);
}
```

```
while(serving_pages) {
page_name = read_web_request(network);
page = read_file(page_name, disk);
write_page(page, network);
}
```

Example #2 (IM)

Imperial College London

```
while(chatting) {
msg = read_message(keyboard);
write_message(msg, network);
msg = read_message(network);
write_message(msg, screen);
}
```

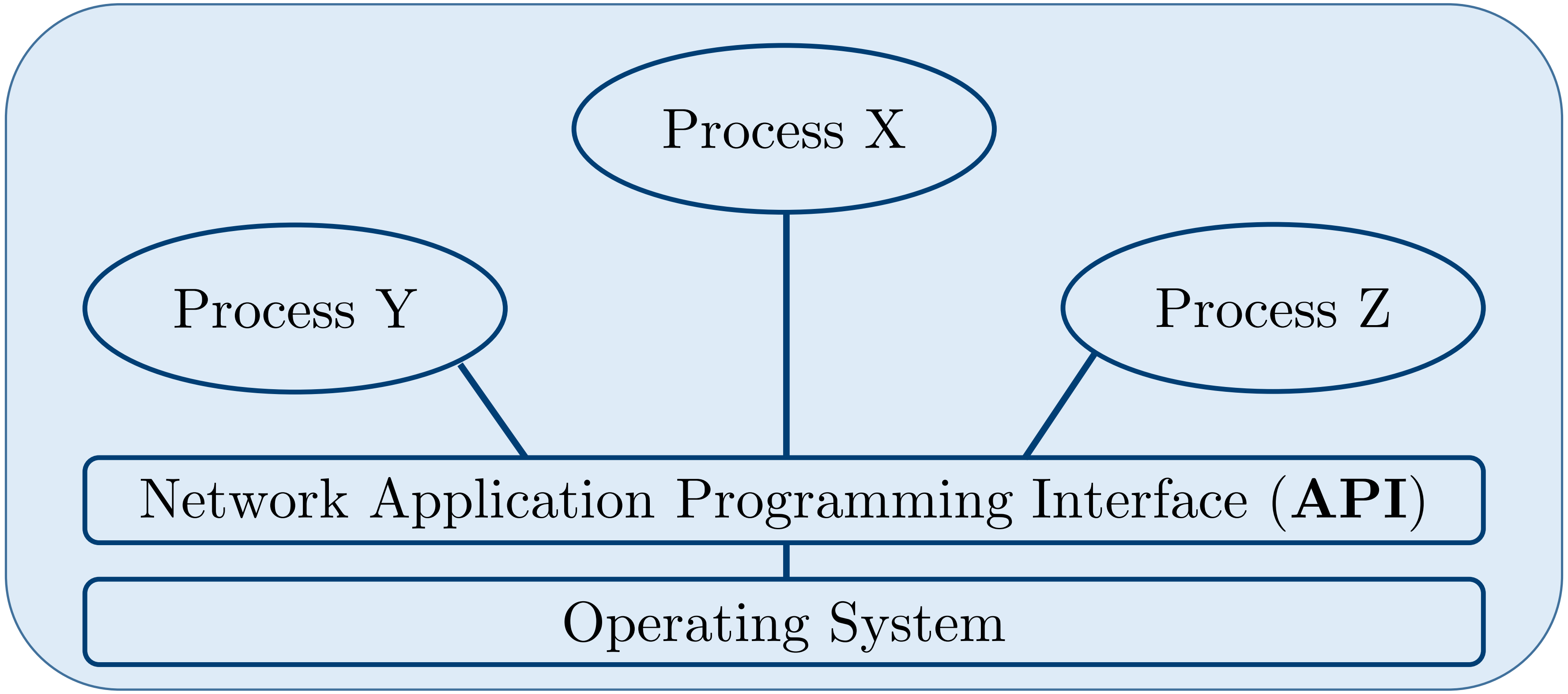
```
while(chatting) {
msg = read_message(network);
write_message(msg, screen);
msg = read_message(keyboard);
write_message(msg, network);
}
```

Clients and Servers

- For each pair of communicating processes, it makes sense to distinguish the two roles
- **Client**: process that *initiates the communication*
  - if the communication is carried over a connection-oriented service, then the client is the process that establishes the connection
- **Server**: process that *waits to be contacted*
  - if the communication is carried over a connection-oriented service, then the server is the process that passively accepts the connection
- Some applications have processes that act both as clients and servers. This is often called *Peer-to-Peer (P2P)* architecture, e.g.
  - BitTorrent (*e.g. μTorrent*)
  - Gnutella (*e.g. gtk-gnutella*)
  - even Skype and Spotify tried it for a while

Processes and Hosts

- An end system (=host) may run multiple programs which run multiple processes

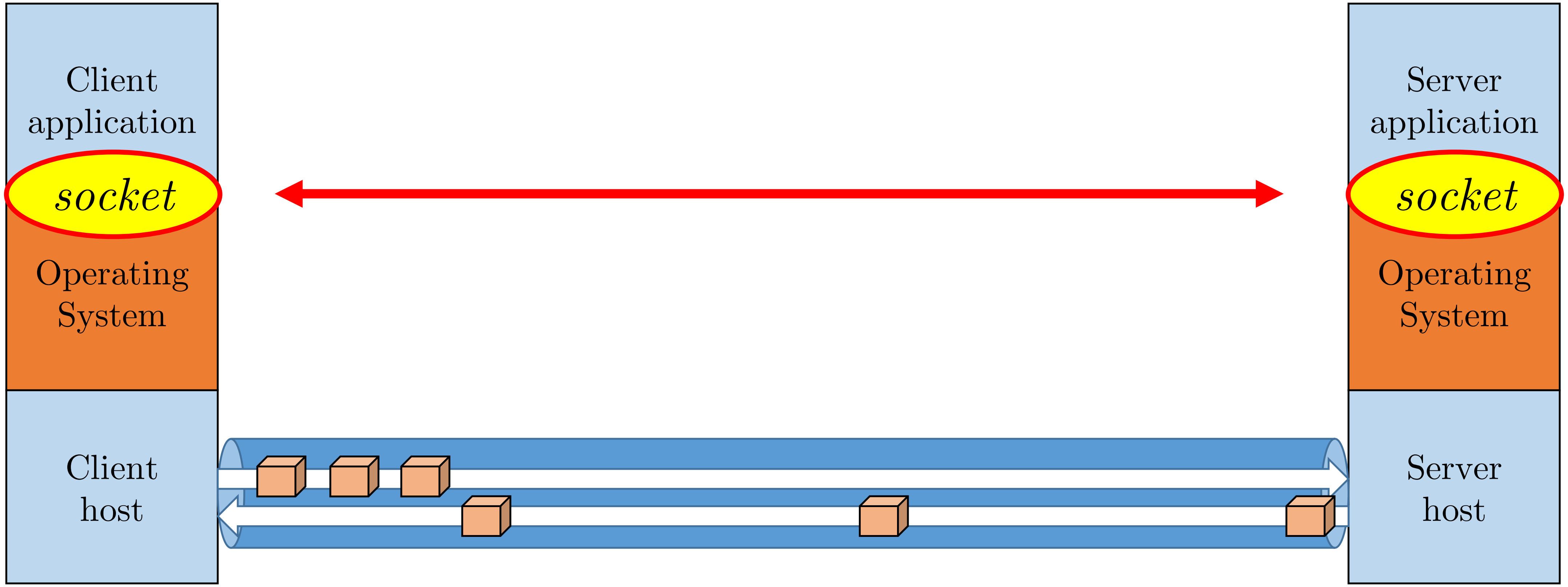


- A process is addressed within its host by a **port number** (*Transport Layer*)

Sockets

Imperial College London

- The operating system manages the network interfaces
- Applications use the network through sockets

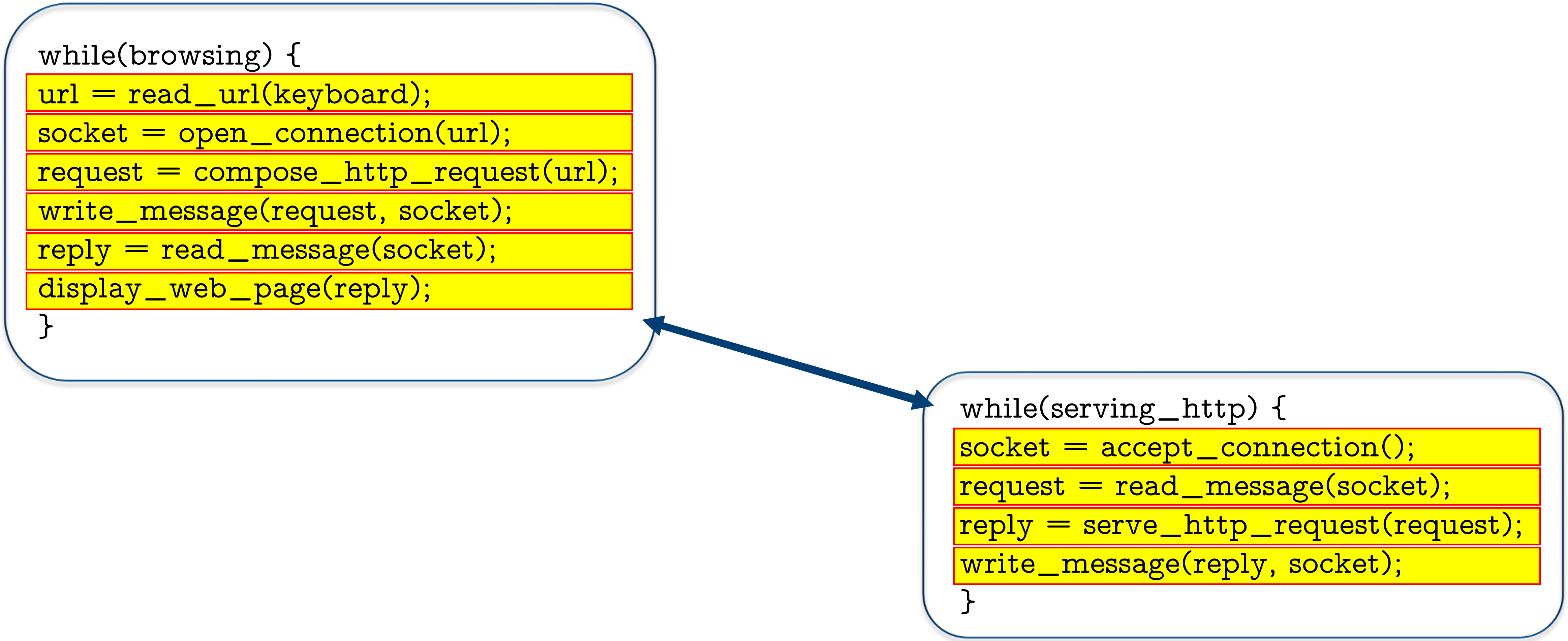


Application Programs

- Client application
  1. create a socket C by "connecting" to the server application
     - i.e. connect to host H on port P
  2. use socket C by reading and writing data into it
     - this is the body of the client application protocol
  3. disconnect and destroy C

- Server application (running on host H)
  1) create a socket S by "accepting" a connection on port P
     - a port is often called a "server socket"
  2) use socket S by reading and writing data into it
     - this is the body of the server application protocol
  3) disconnect and destroy S

Example #3 (HTTP)



```
while(browsing) {
url = read_url(keyboard);
socket = open_connection(url);
request = compose_http_request(url);
write_message(request, socket);
reply = read_message(socket);
display_web_page(reply);
}
```

```
while(serving_http) {
socket = accept_connection();
request = read_message(socket);
reply = serve_http_request(request);
write_message(reply, socket);
}
```

The World Wide Web

- Invented in 1989 by Sir Tim Berners-Lee; formally defined in 1991
- Based on the idea of **hypertext** and **(hyper)links**
- Extremely successful, even though
  - the HyperText Transfer Protocol (HTTP) is just a “glorified” file transfer protocol
  - the idea of hypertext and links was already quite old at the time HTTP was developed (*based on a proposal by William Tunnicliffe in the late 1960s*)
- Success factors
  - simplicity (*openness*) of the HTML language and
  - simplicity of HTTP (*a stateless protocol*)
  - low entry barrier for “publishers”
  - GUI browsers (Mosaic, Netscape => Firefox, Chrome, etc.), search engines, etc.

Web Terminology

Imperial College London

- **document** – a webpage is also called a *document*
  - *one website has one or more webpages*
- **objects** – a *document* may contain several *objects* (*images, videos, stylesheets, etc.*)
  - in other words, an *object* is simply a file
- **URL** – or Uniform Resource Locator specifies the address of an *object*
- **browser** – the program that users run to get and display *documents* (*also called user agent*)
- **Web server** – an application that houses *documents* and *objects*, and makes them available through the HTTP protocol

Overview HTTP

- The main purpose of HTTP is to provide access to Web objects
- Uses a connection-oriented transport mechanism (*i.e. TCP*)
  - *although it could also work over UDP*
- Consists of a sequence of requests issued by the client, and responses issued by the server, each one in response to a single request
- HTTP is stateless
  - the behaviour/meaning (*semantics*) of an HTTP request does not depend on any previous request
- HTTP/1.0 RFC https://tools.ietf.org/html/rfc1945
- HTTP/1.1 RFC https://tools.ietf.org/html/rfc2068
- HTTP/2~~.0~~ RFC https://tools.ietf.org/html/rfc7540 (*based on SPDY*)

Example: HTTP Request

Imperial College London

- Client Request
  - `telnet www.doc.ic.ac.uk 80`

```
GET /~kgk/212/index.html HTTP/1.1
Host: www.doc.ic.ac.uk
User-agent: Mozilla/5.0
Accept-Language: en-GB
```

Example: HTTP Response

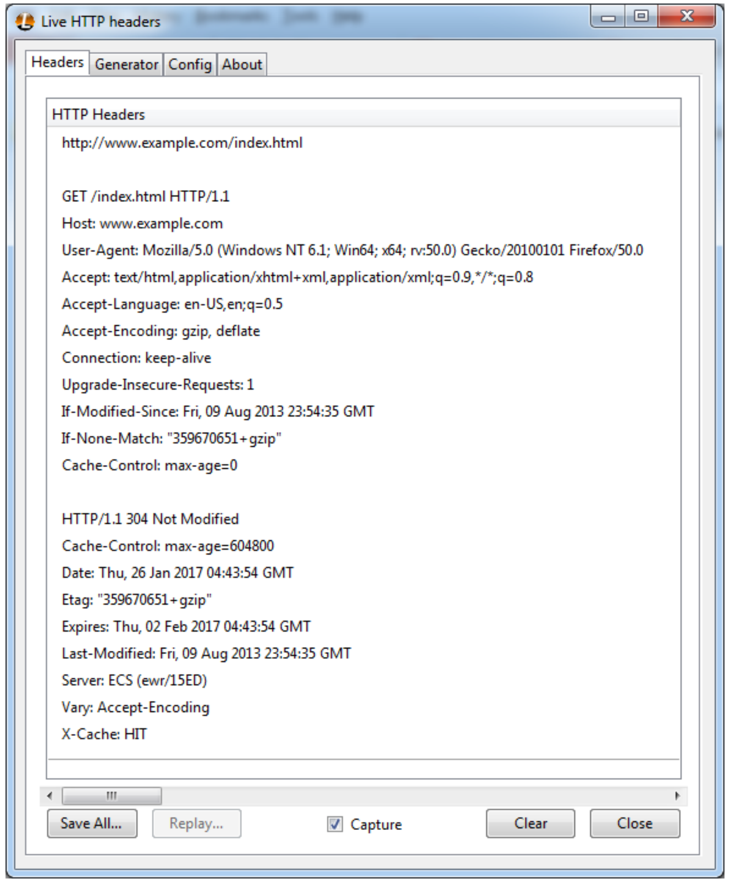
Imperial College London

- Server Response
  - telnet www.doc.ic.ac.uk 80

```
HTTP/1.1 200 OK
Date: Thu, 26 Jan 2017 16:00:00 GMT
Server: Apache
[...]
Last-Modified: Sat, 21 Jan 2017 20:05:55 GMT
[...]
Accept-Ranges: bytes
Content-Length: 5470
[...]
Content-Type: text/html
[...]
<!DOCTYPE html>
[...]
```

How do I see this?

- You can see these HTTP headers even without using telnet
- Just install the Live HTTP Headers add-on for Firefox
  - Open it up and visit a website to see HTTP in action!
  - A similar add-on is Firebug
  - You can also play with TamperData and ModifyHeaders
- Alternatively, you can use a .NET based tool called Fiddler
  - (*it can run on Linux with some effort*)
- Another way is using a Packet Analyser/Sniffer
  - (*to be discussed later on*)
- More advanced tools: WebScarab & mitmproxy



```
http://www.example.com/index.html

GET /index.html HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:50.0) Gecko/20100101 Firefox/50.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
If-Modified-Since: Fri, 09 Aug 2013 23:54:35 GMT
If-None-Match: "359670651+gzip"
Cache-Control: max-age=0

HTTP/1.1 304 Not Modified
Cache-Control: max-age=604800
Date: Thu, 26 Jan 2017 04:43:54 GMT
Etag: "359670651+gzip"
Expires: Thu, 02 Feb 2017 04:43:54 GMT
Last-Modified: Fri, 09 Aug 2013 23:54:35 GMT
Server: ECS (ewr/15ED)
Vary: Accept-Encoding
X-Cache: HIT
```

Protocol Features

- Request
  - protocol version
  - URL specification
  - connection attributes
  - content/feature negotiation

- Reply
  - protocol version
  - reply status/value
  - connection attributes
  - object attributes
  - content specification (type, length)
  - content

Protocol Version

Imperial College London

```
GET /~kgk/212/index.html HTTP/1.1
Host: www.doc.ic.ac.uk
User-agent: Mozilla/5.0
Accept-Language: en-GB
```

```
HTTP/1.1 200 OK
Date: Thu, 26 Jan 2017 16:00:00 GMT
Server: Apache
Last-Modified: Sat, 21 Jan 2017 20:05:55 GMT
Accept-Ranges: bytes
Content-Length: 5470
Content-Type: text/html
<!DOCTYPE html>
```

Protocol Version (cont'd)

- **General principle**: a protocol should always include a version number
  - usually in the very first bits of the protocol (*negotiation messages*)
- A mechanism to negotiate the protocol version allows the protocol design to change
  - i.e. designed for change
- HTTP/2 will be replacing HTTP/1.x within the next few years
  - Why? See: https://http2.github.io/faq/
- In HTTP/2:
  - content is binary (*i.e. not plain text any more*)
  - connection is fully multiplexed (*not ordered/blocking*)
  - can thus use a single (*TCP*) connection for parallelism
  - ...and more!

URL

```
GET /~kgk/212/index.html HTTP/1.1
Host: www.doc.ic.ac.uk
User-agent: Mozilla/5.0
Accept-Language: en-GB
```

```
HTTP/1.1 200 OK
Date: Thu, 26 Jan 2017 16:00:00 GMT
Server: Apache
Last-Modified: Sat, 21 Jan 2017 20:05:55 GMT
Accept-Ranges: bytes
Content-Length: 5470
Content-Type: text/html
<!DOCTYPE html>
```

The Host Header

Imperial College London

```
GET /~kgk/212/index.html HTTP/1.1
Host: www.doc.ic.ac.uk
User-agent: Mozilla/5.0
Accept-Language: en-GB
```

- http://**www.doc.ic.ac.uk**/~kgk/212/index.html
- The host name in the URL determines where the request goes
  - host name maps to a network address
- The host name is also passed as a parameter within the request, so that the server knows the full URL
  - this is to allow a single server to serve multiple "virtual" sites (*e.g. www.google.com and www.l.google.com*)

Anatomy of a Request

Imperial College London

- Request line
  - GET /~kgk/212/index.html HTTP/1.1
- Header lines (*zero or more*)
  - Host: www.doc.ic.ac.uk
  - User-agent: Mozilla/5.0
  - Accept-Language: en-GB
  - + more
- Empty line
- Object body
  - (*could be empty*)

Anatomy of a Request (cont'd)

- Request line
  - Method: GET
  - Space
  - URL: /~kgk/212/index.html
  - Space
  - Protocol/Version: HTTP/1.1
- Header lines (*zero or more*)
  - Header name: Host:
  - Space
  - Header value: www.doc.ic.ac.uk
  - etc.
- Line terminator: CRLF (*"carriage return" and "line feed"*)
  - 2 Bytes: numeric values 13 and 10

Methods

- **GET** retrieve the object identified by the URL
- **POST** allows the submission of data to the server
  - e.g. a mail message in a webmail system, a form in an e-commerce site, etc.
  - the given URL is the object that handles the posting
- **OPTIONS** requests the available communication options for the given object
- **HEAD** like *GET*, but without the body
  - useful for testing the validity of links
- **PUT** requests that the enclosed object be stored under the given URL
- **DELETE** deletes the given object

Anatomy of a Response

Imperial College London

- Status line
  - HTTP/1.1 405 Method Not Allowed
- Header lines (*zero or more*)
  - Date: Thu, 26 Jan 2017 16:00:00 GMT
  - Server: Apache
  - Allow: GET,HEAD,POST,OPTIONS
  - Content-Length: 312
  - Content-Type: text/html;
- Empty line
- Object body
  - (*possibly empty*)
  - `<html><head><title>405 Method Not Allowed</title></head><body>`

Anatomy of a Response (cont'd)

- Status line
  - Protocol/Version: HTTP/1.1
  - Status code: 405
  - Brief description: Method Not Allowed
- The status code is a 3-digit value (*e.g. 200 or 404*)
- The rest has exactly the same structure as a request

Status Codes

- **1xx**: Informational (*see Section 10.1 of RFC 2616*)
- **2xx**: Successful operation (*see Section 10.2 of RFC 2616*)
- **3xx**: Redirection; e.g., indicates that the object has moved, either temporarily or permanently
- **4xx**: Client error; e.g., malformed request (400), unauthorized (401), object not found (404), method not allowed (405)
- **5xx**: Server error; e.g., internal server error (500), service overloaded (503)

Headers

- Object characterisation
  - e.g. Content-Type, Content-Length, Content-Encoding
- Content negotiation
  - e.g. Accept-Charset, Accept-Encoding
- Object properties useful for cache management
  - e.g. Expires, Last-Modified, ETag
- Explicit cache control
  - e.g. Cache-Control
- Method-specific responses
  - e.g. Allow as a response to OPTIONS
- Authorization/identification
  - e.g. Authorization

You Try It – Telnet

Imperial College London

- telnet allows us to send plain-text commands directly to a server listening on a specific port (*e.g. 80 for HTTP*)

- During the break, go to a lab machine and use **telnet** to retrieve the main Imperial College – Department of Computing website

- `telnet www.imperial.ac.uk 80`

- `GET /computing/ HTTP/1.1`
- `Host: www.imperial.ac.uk`

- What was the response?
  - Can you see which Web server we are using?

How HTTP Uses (TCP) Connections

Imperial College London

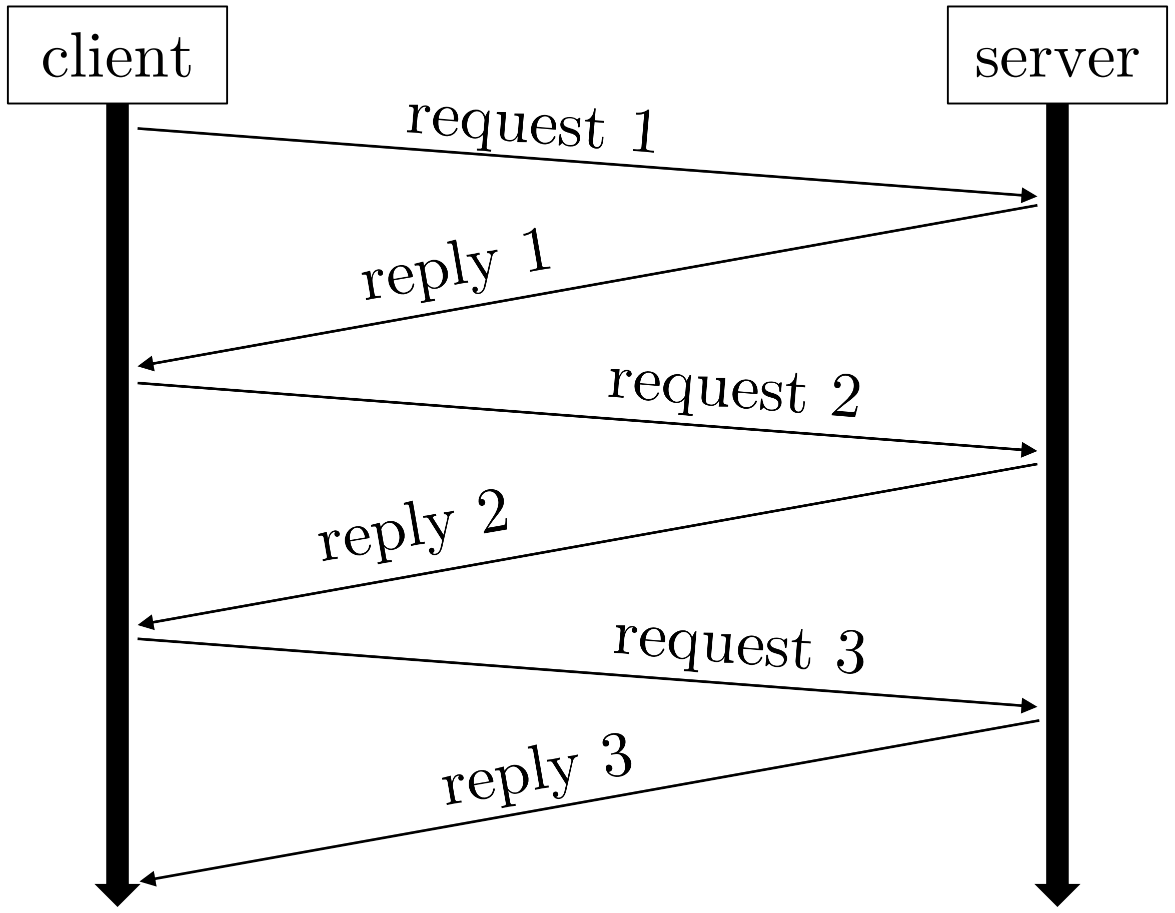
- The first version of HTTP used one (*TCP*) connection **per object**
  - inefficient use of the network
  - inefficient use of the operating system

- HTTP/1.1 introduces persistent connections
  - the same (*TCP*) connection can be used by the client to issue **multiple** requests, and by the server to return multiple replies, and possibly multiple objects
  - the default behavior is to use ***persistent*** connections
  - "`Connection: close`" in the request and response indicates the intention, of the client and server, respectively, to not use a persistent connection

How HTTP Uses Persistent Connections

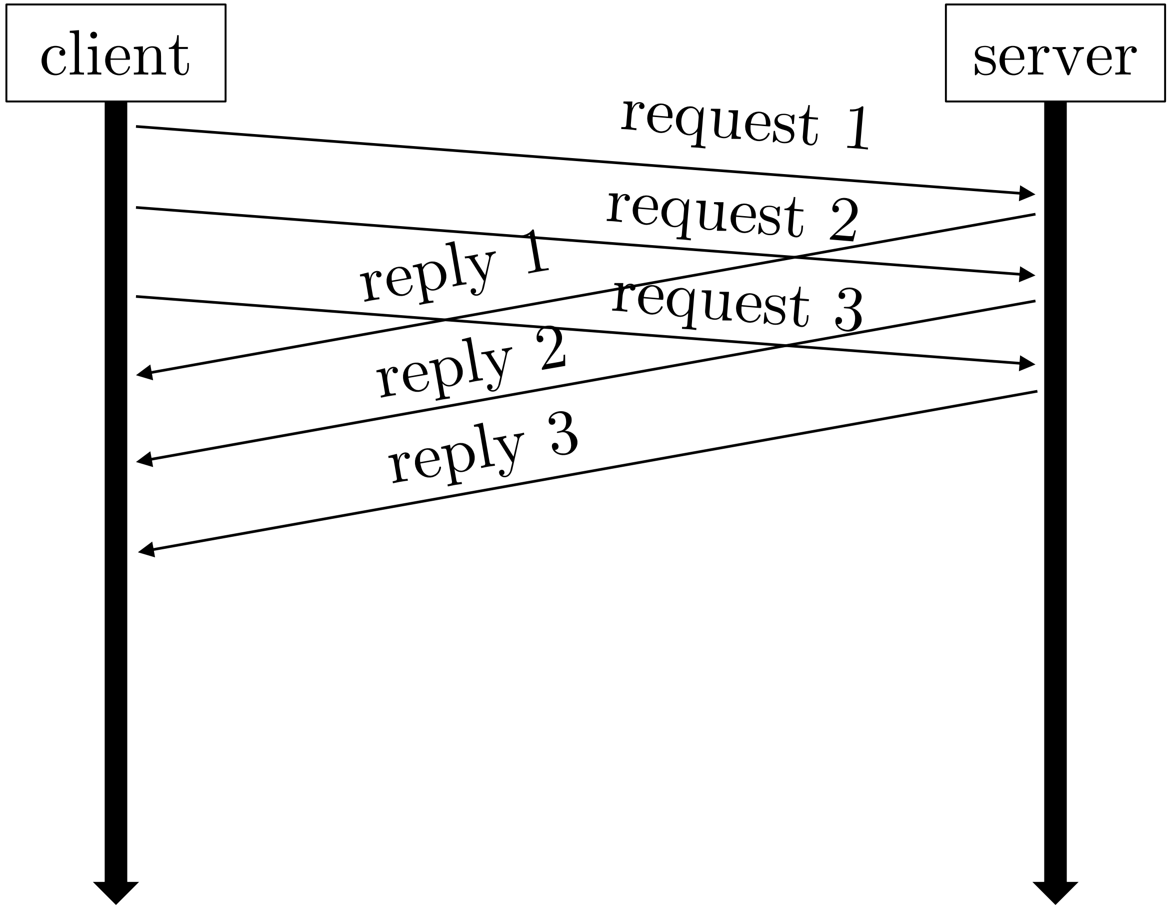
Imperial College London

- A persistent connection can be used to request and transfer more than one objects (*connection open/close is not shown*)



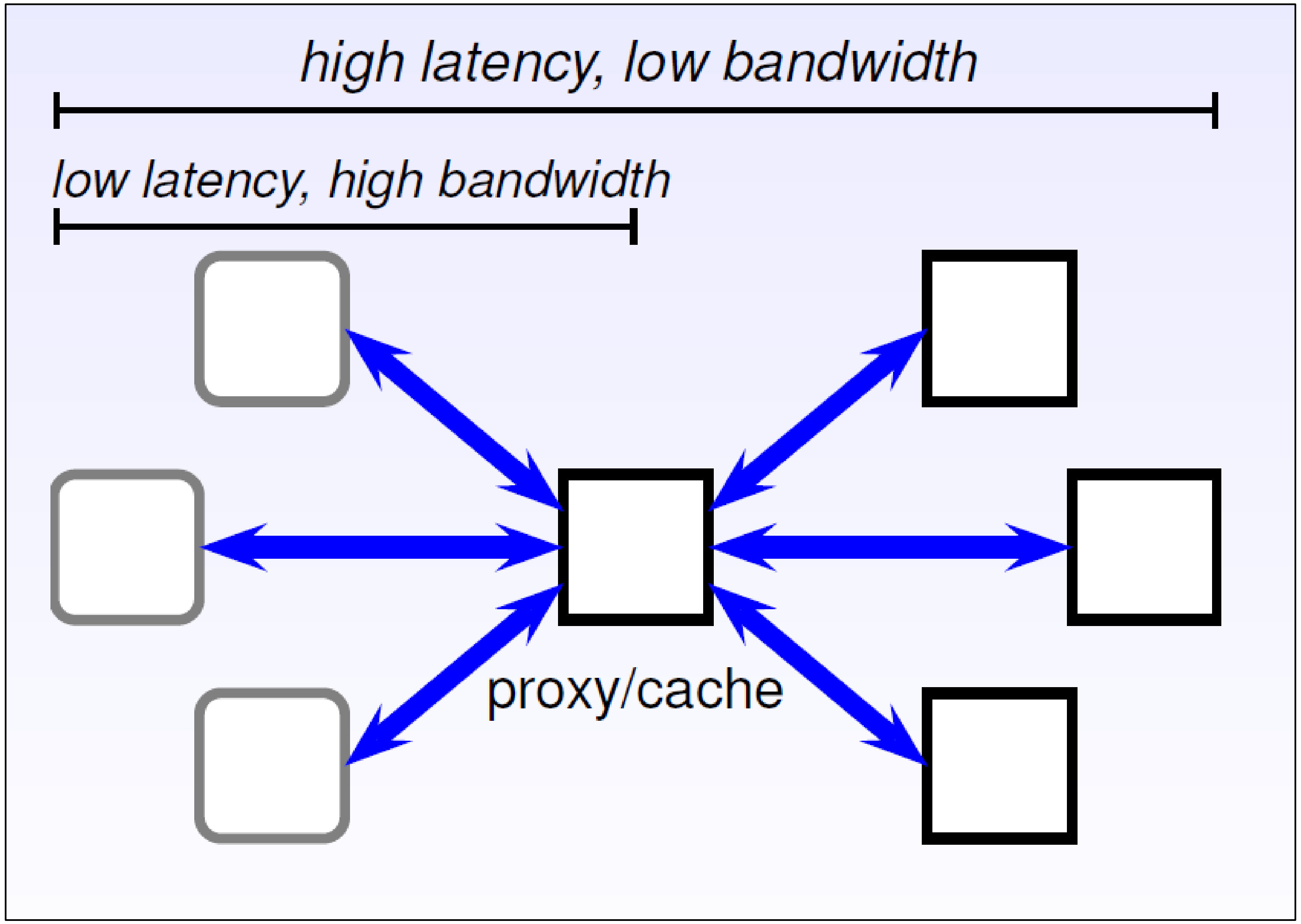
Persistent Connections With Pipelining

- A more efficient use of a connection is by ***pipelining*** requests

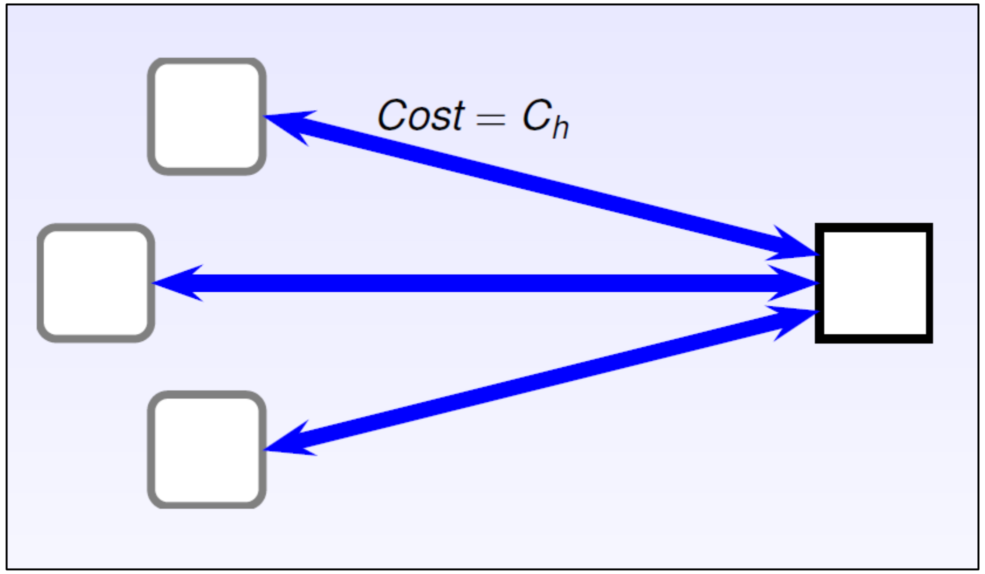


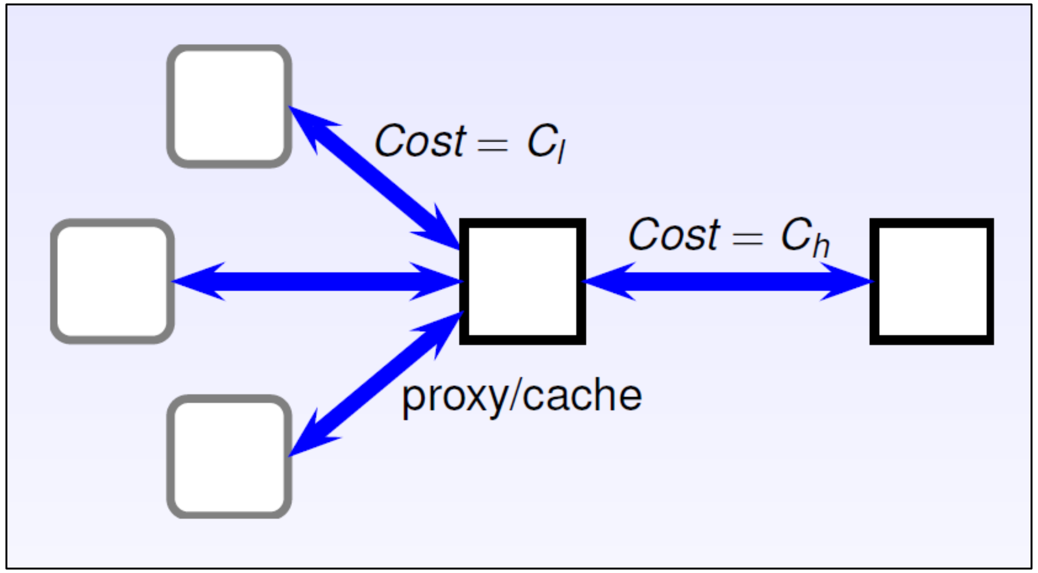
Web Caching

- Same idea as caching in a memory hierarchy



Example: Web Caching





- Without proxy/cache: total cost $= 3C_h$
- With proxy/cache: total cost $= C_h + 3C_l$
  - and now, it only costs C_l for cached

Example: Web Caching (cont'd)

Imperial College London

- A client request goes to a proxy (*cache*) server
- The proxy may
  1. forward the request to the origin server, thereby acting as a client
  2. get the response from the origin server
  3. possibly store (*cache*) the object for some time
  4. forward the response back to the client
- Or, the proxy may
  1. respond immediately to the client, possibly using a cached object

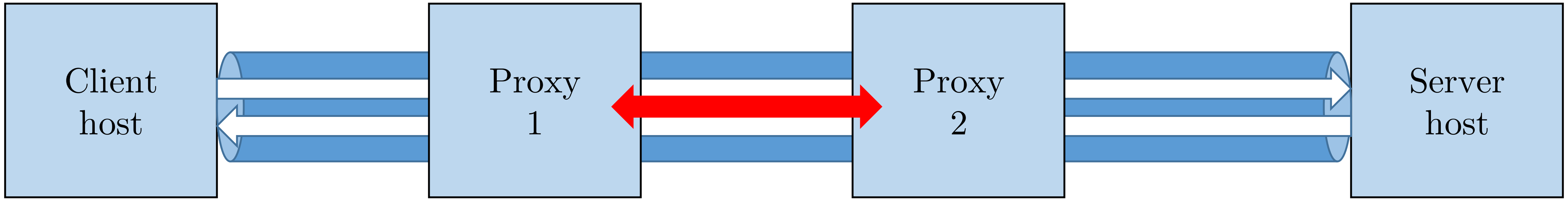
Example: Web Caching (cont'd)

Imperial College London

- Benefits of the proxy/cache architecture
  - performance
    - reduced latency
    - reduced network traffic
  - security
    - privacy; the server sees **the proxy** as a client
    - protection from intrusions, combined with a firewall
- Problems
  - latency (*just like any other caching system*)
  - complexity
  - data “freshness”

HTTP and Caching

- The proxy/cache architecture is central to several features of HTTP – in fact, it *affects* its overall design
- HTTP is defined as a request/response protocol, where requests and responses are explicitly passed through *a request chain*



- HTTP defines:
  - how protocol versions are handled on the request chain
  - how each method must be handled with regard to the request chain
    - e.g. responses to OPTIONS requests are not cacheable; only cacheable if indicated by "Cache-Control" or "Expires" header field
  - specific authentication mechanisms for proxies
  - a lot of headers to control caching along the request chain

HTTP and Caching (cont'd)

- Cached pages may become stale
- A `HEAD` request could be used to see if an object has been updated, in which case the cache can be invalidated
  - but how does a proxy decide that it is okay to respond to a client with a cached object?
- Servers specify explicit expiration times using either the Expires header, or the `max-age` directive of the Cache-Control header
- A client or proxy can use a conditional `GET` by including an “`If-Modified-Since`” header

Cache-Control in Requests

```
GET /~kgk/212/index.html HTTP/1.1
Host: www.doc.ic.ac.uk
Cache-Control: no-cache
```

- "Please, do not use cached objects!"
  - proxies must go to the origin server

```
GET /~kgk/212/index.html HTTP/1.1
Host: www.doc.ic.ac.uk
Cache-Control: max-age=20
```

- "Please, give me a cached object only if it is less than 20 seconds old"

Cache-Control in Replies

```
HTTP/1.1 200 Ok
Cache-Control: no-cache
...
```

- "Please, do not cache this object !"

```
HTTP/1.1 200 Ok
Cache-Control: maxage=100; must-revalidate
...
```

- "You may use this object up to 100 seconds from now. After that, you must revalidate the object."
  - without the must-revalidate directive, a client may use a stale object

Sessions in HTTP

- HTTP is a **stateless** protocol
  - *so how do you implement, for example, a "shopping cart"?*

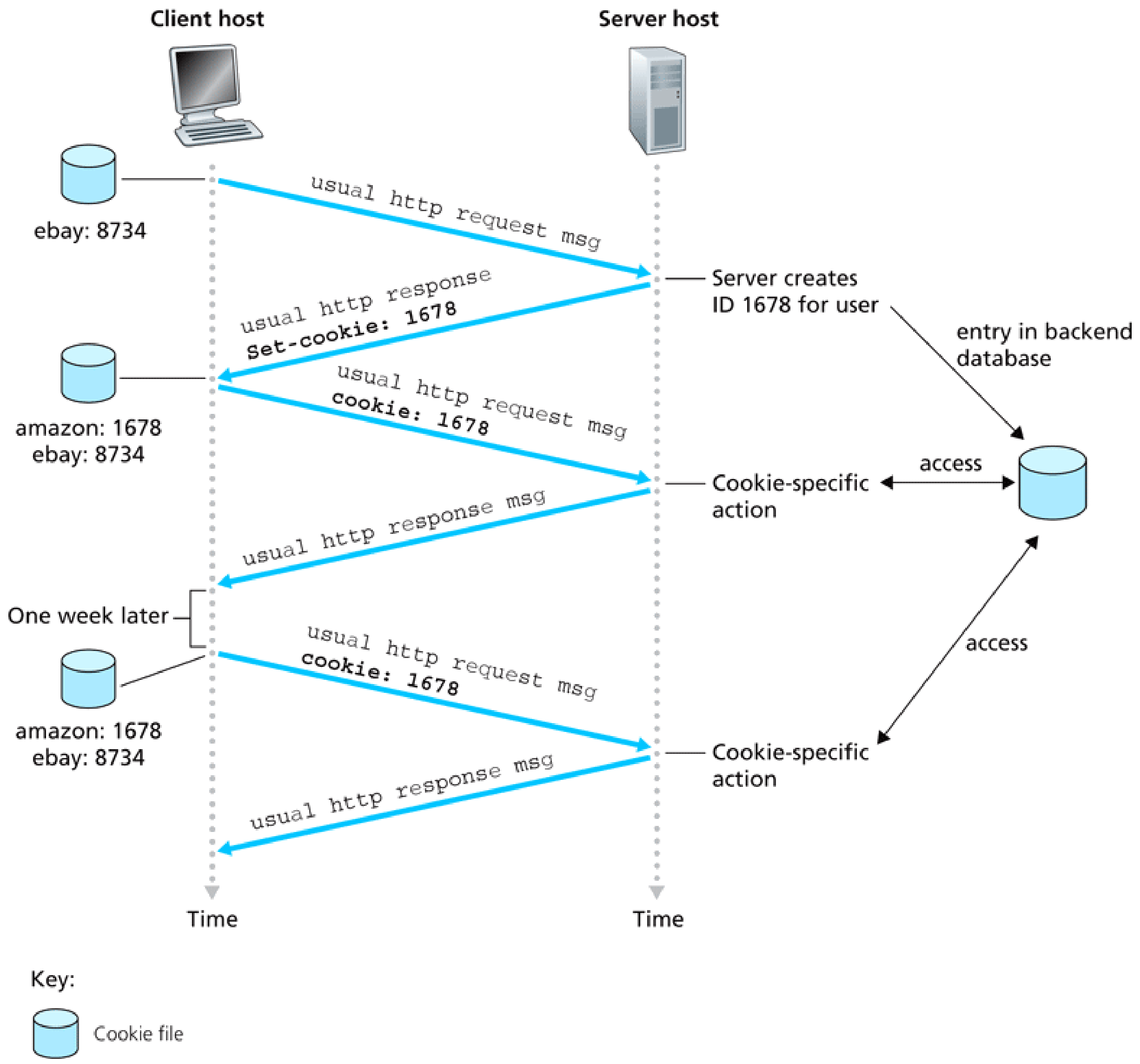
- HTTP provides the means for higher-level applications to maintain **stateful** sessions (*see RFC 2109*)

- Set-Cookie header
  - sent within an HTTP response, from the server to the client
  - tells the client to store the given "cookie" as a session identifier for that site

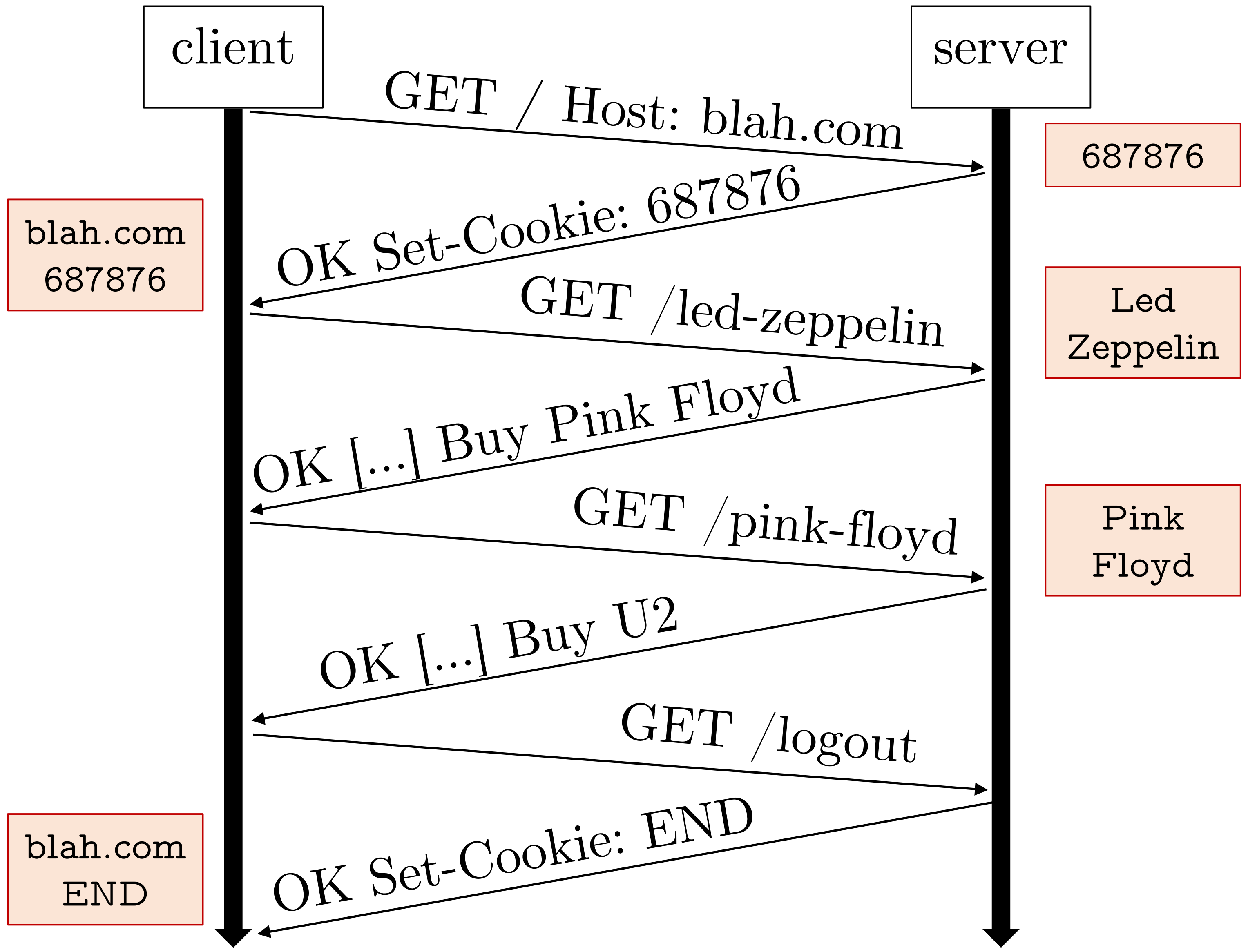
- Cookie header
  - sent within an HTTP request, from the client to the server
  - tells the server that the request belongs to the given session

Imperial College London

Example: Cookies



Example

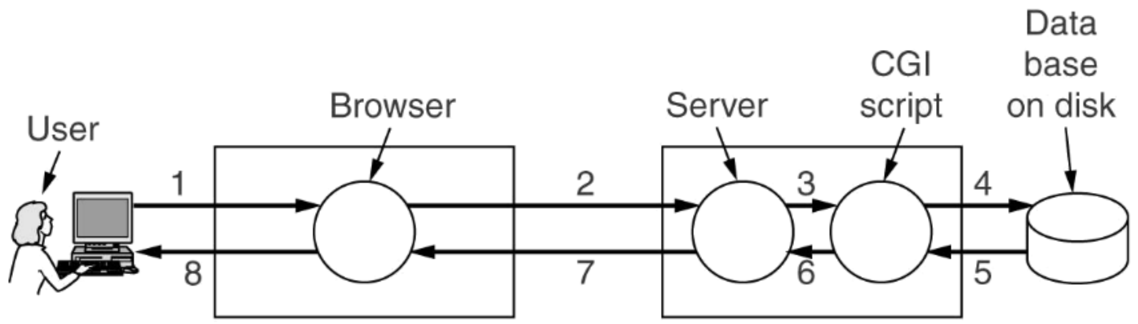

client
server
GET / Host: blah.com
687876
OK Set-Cookie: 687876
blah.com
687876
GET /led-zeppelin
Led
Zeppelin
OK [...] Buy Pink Floyd
GET /pink-floyd
Pink
Floyd
OK [...] Buy U2
GET /logout
OK Set-Cookie: END
blah.com
END

Cookies and User Privacy

- A “session” identifies the actions of a user
- Websites may use cookies to compile and collect user profiles
  - ...and of course they do exactly that!
- In our example, we can infer that user #687876:
  - likes rock-blues music from the sixties and seventies
  - stays in the UK
    - *how do we know this?*
    - *is it definitely true?*
  - [...]
- If user #687876 buys something on-line with a credit card, then he or she could also be identified

Dynamic Web Pages

- Instead of storing and returning statically defined webpages, servers often generate pages on-the-fly



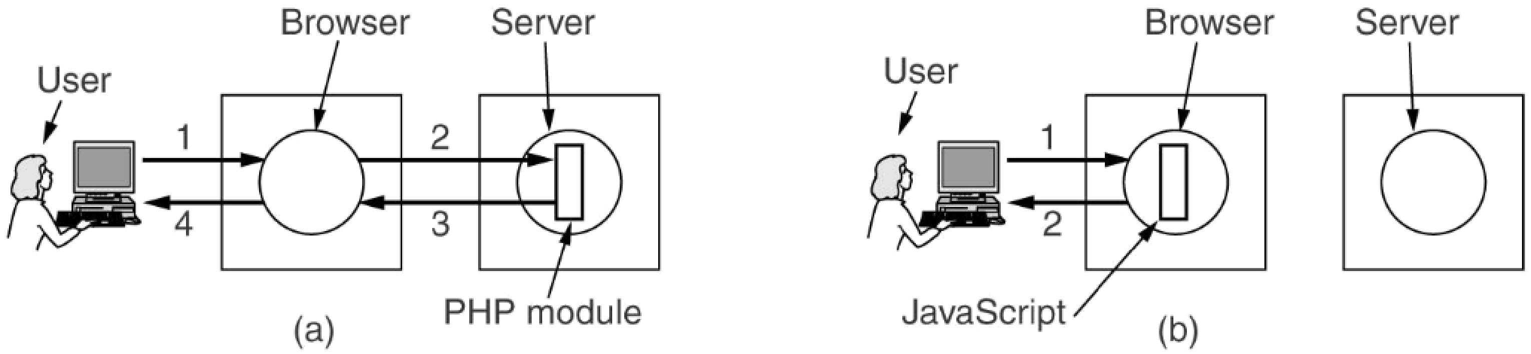
1. User fills in form
2. Form sent back
3. Handed to CGI
4. CGI queries DB
5. Record found
6. CGI builds page
7. Page returned
8. Page displayed

- **Common Gateway Interface**: CGI essentially allows you to identify a program and its parameters in a URL
  - The server will start a process to execute that program, which, in turn, will return its results (*if any*) as a regular webpage

- **Servlets** are the Java-based solution
  - the webserver contains an instance of the Java virtual machine
  - they can maintain state (CGI is *stateless*)

Scripting and Web Pages

- Alternative approach: Let Web pages incorporate interpretable code; when a page is being processed, the embedded script is simply executed
- Distinguish server-side **(a)** and client-side solutions **(b)**:



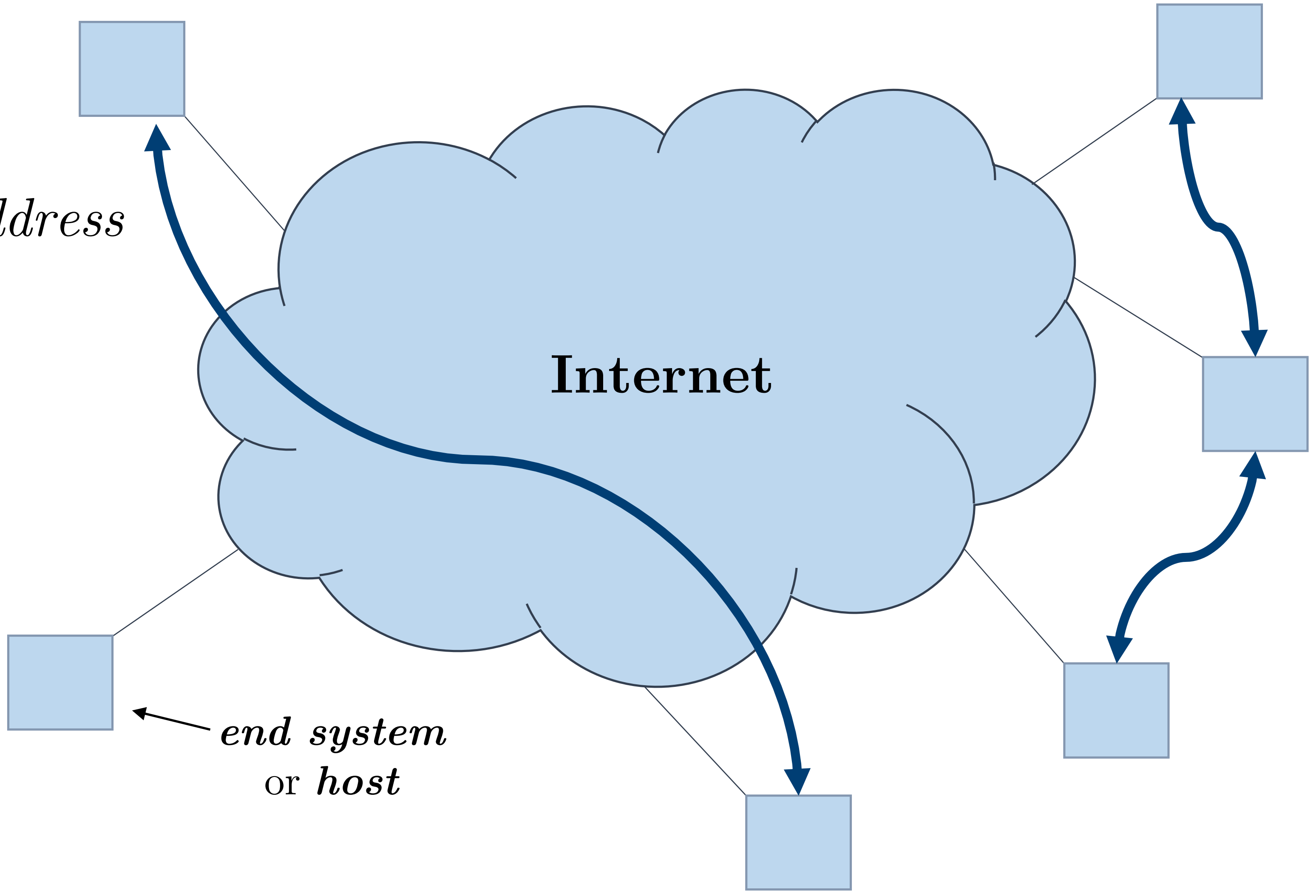
  - Example#1: **Server-side**: **PHP**: **H**ypertext **P**reprocessor (PHP)
  - Example#2: **Client-side**: **J**ava**S**cript (JS)

```
<html>
<head><title>PHP Test</title></head>
<body><? print("Hello World!"); ?></body>
</html>
```

```
<html><body>
<script language="javascript">
document.write("Hello World!");
</script>
</body></html>
```

End Systems

- Internet applications involve end-system communication
- *But how does one end system address another end system?*



IP Addresses

- An end system is **identified** and **addressed** by its **IP address**
  - 32 bits (4 bytes) in IPv4
    - e.g. 146.169.13.11
  - 128 bits (16 bytes) in IPv6
    - e.g. fe80::211:43ff:fecd:30f5/64
- **Advantages**
  - computers (*e.g. routers*) are good at processing bits
    - especially in small packs of a size that is a power of two
- **Disadvantages**
  - not practical for use by people
    - i.e. not mnemonic
  - e.g. "look it up on 216.58.212.99"

Host Names

Imperial College London

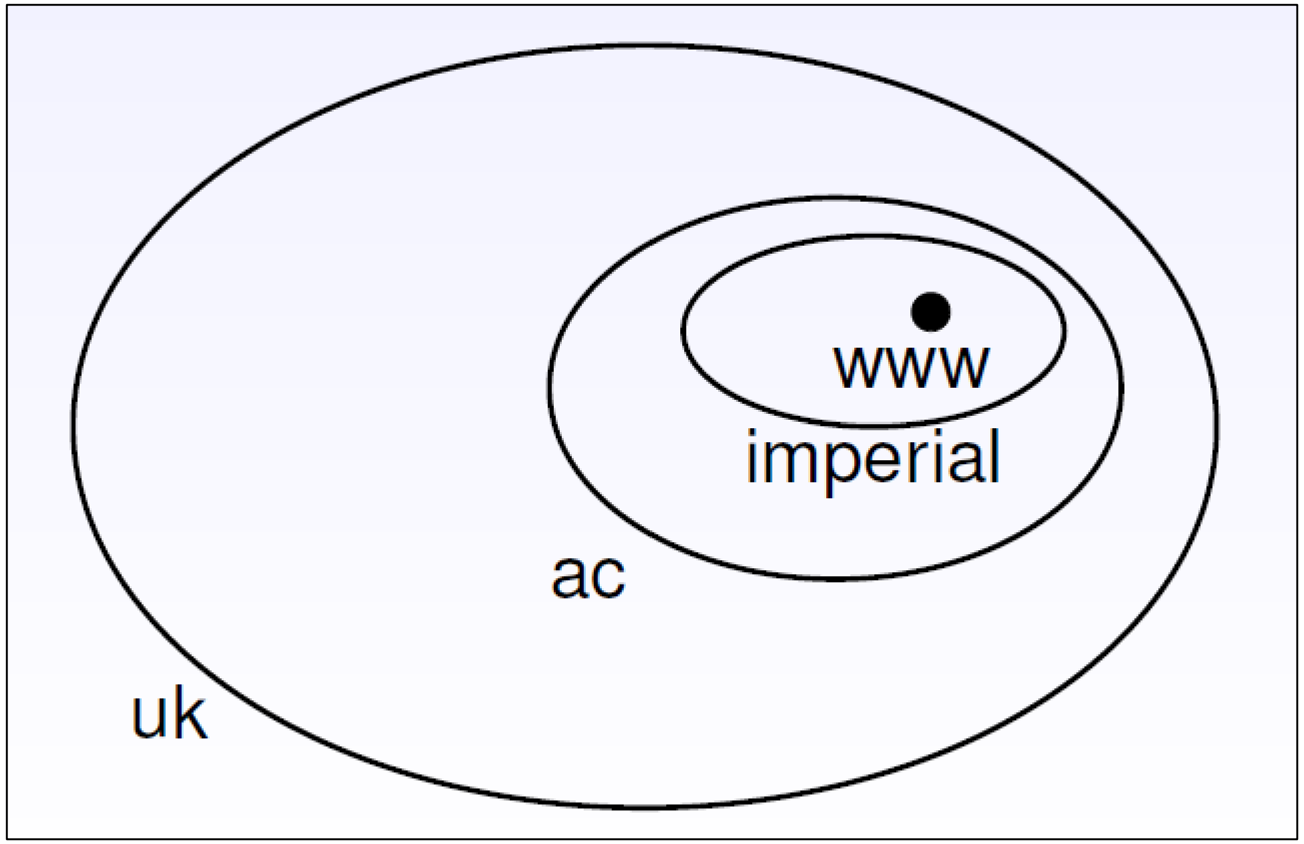
- Goal: help the human users of the Internet
  - human-readable, mnemonic addresses, aliases
- Solution (*up to 1983*): all mapping was contained in the file ”`hosts`”
  - (*not that many hosts back then*)
- Solution (*from 1983 onwards*): Domain Name System (DNS)
  - Distributed lookup facility
- Primary function of the Domain Name System:
  - map a name to an IP Address
  - e.g. www.doc.ic.ac.uk => 146.169.13.11

Host Names (cont'd)

- Example:

www.imperial.ac.uk

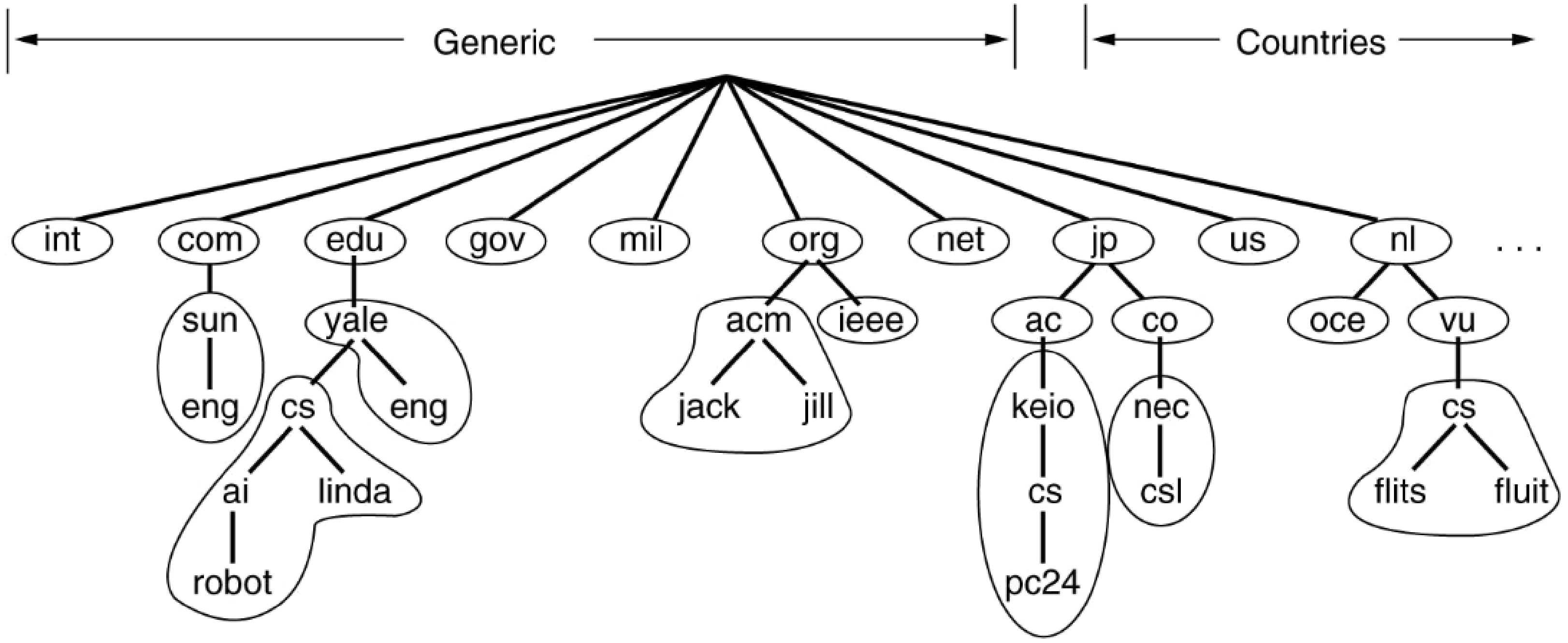
- Hierarchical name space
- Start from the "**T**op-**L**evel **D**omain"
  - which is *at the end* of the URL
- ...and move backwards/inwards
  - Rm228.
  - DepartmentOfComputing.
  - HuxleyBuilding.
  - ImperialCollegeLondon.
  - 180Queen'sGate.
  - SW72AZ.
  - London.
  - UK



Architecture of DNS

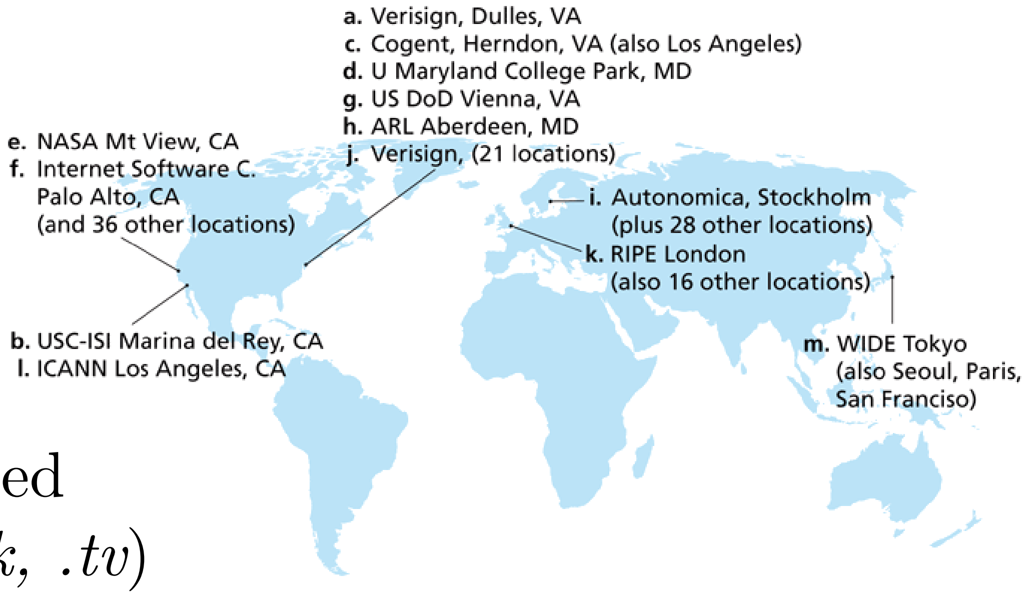
Imperial College London

- The name space was divided into a collection of non-overlapping zones
- Each one is handled/served by one or more **name servers**



DNS Architecture

- **Root servers**: 13 "root" DNS servers know where the top-level servers are (*labelled A through M*)
  - operated by 12 independent organisations
  - see http://root-servers.org/


a. Verisign, Dulles, VA
c. Cogent, Herndon, VA (also Los Angeles)
d. U Maryland College Park, MD
g. US DoD Vienna, VA
h. ARL Aberdeen, MD
j. Verisign, (21 locations)
e. NASA Mt View, CA
f. Internet Software C. Palo Alto, CA (and 36 other locations)
i. Autonomica, Stockholm (plus 28 other locations)
k. RIPE London (also 16 other locations)
b. USC-ISI Marina del Rey, CA
l. ICANN Los Angeles, CA
m. WIDE Tokyo (also Seoul, Paris, San Franciso)

- **Top-level domain servers**: each one is associated with a top-level domain (*e.g. .com, .edu, .org, .uk, .tv*)

- **Authoritative servers**: for each domain, there is an authoritative DNS server that holds the map of public hosts within that domain

- Most "root" servers, as well as servers at lower levels, are themselves implemented by a distributed set of machines

Observations on DNS

Imperial College London

- A lot of messages just to figure out where to connect to
  - DNS can indeed be a major bottleneck for some applications (*typically, the Web*)
  - it is also, to a large extent, a critical point of failure
- It is a perfect demonstration of the "end-to-end principle"
  - it implements a (*crucial*) network functionality at the end-system level
- How can one improve the performance and reliability of DNS?

DNS Caching

- Caching is clearly important, as it can dramatically:
  - improve the performance of DNS
  - reduce the load on the DNS infrastructure

- How does caching work in DNS?
  - Same as always:
    - a DNS server may cache a reply (*i.e. the mapping*) for a name $\boldsymbol{n}$
    - if the server receives a subsequent request for $\boldsymbol{n}$, it may respond directly with the cached address, even if the server is not the *authoritative* server for that domain

- Useful?
  - Definitely.
    - Dangerous?
      - Absolutely!

DNS Features

Imperial College London

- DNS is essentially a “directory service” database
- The database contains Resource Records (RRs)

Name	Value	Type	TTL
www.imperial.ac.uk	146.179.40.24	A	...
shell1.doc.ic.ac.uk	146.169.2.83	A	...
...	...	...	...

- *Name* and *value* have the intuitive meaning
- What about *type*?

DNS Query Types

Imperial College London

- "**A**" maps host_name => address
  - *name* is a host name, and *value* is its (IP) *address*

Name	Value	Type	TTL
www.imperial.ac.uk	146.179.40.24	A	...
...	...	...	...

- "**NS**" is a query for a name server
  - *name* is a domain name, and *value* is the *authoritative name server* for that domain

Name	Value	Type	TTL
imperial.ac.uk	ns0.ic.ac.uk	NS	...
...	...	...	...

- "**CNAME**" is a query for a *canonical name*
  - Canonical name is the "primary" name of a host; a host may have one or more mnemonic *aliases*

Name	Value	Type	TTL
www**3**.imperial.ac.uk	www.imperial.ac.uk	CNAME	...
...	...	...	...

DNS Query Types (cont'd)

Imperial College London

- "MX" is a query for the **mail exchange** server for a given domain
  - *name* is a host or domain name, and *value* is the name of the mail server that handles (*incoming*) mail for that host or domain

Name	Value	Type	TTL
imperial.ac.uk	mx1.cc.ic.ac.uk	MX	…
…	…	…	…

- There are several other types

DNS Protocol

Imperial College London

- DNS is a *connectionless* protocol
- Runs on top of *UDP* (port 53)
- DNS has *query* and *reply* messages
  - since DNS is connectionless, queries and replies are linked by an identifier
- Both queries and replies have the same format
  - a DNS message can carry queries and answers
- **Question**:
  - Why use UDP (*best effort*) instead of TCP (*reliable*) delivery?
- **Answer**:
  - Since it always only involves **two** network packets (*request and response*), setting up and tearing down a TCP connection every time would be wasteful

Round robin DNS

Imperial College London

- Round robin DNS is a technique for balancing the load of geographically-distributed Web servers
- Round robin works by responding to DNS requests *not with a single IP address*, but a *list of IP addresses*
  - the *order* in which IP addresses are returned is the key
  - the IP address at the *top* of the list is returned *a set number of times* before it is moved to the bottom, thus promoting the second IP address to the top of the list
- For example, executing `host outlook.com` returns
  - outlook.com has address 40.97.164.146
  - outlook.com has address 40.97.170.154
  - outlook.com has address 40.97.153.146
  - outlook.com has address 40.97.113.210
  - outlook.com has address 40.97.160.2
  - outlook.com has address 40.97.166.138
  - outlook.com has address 40.97.113.34
  - outlook.com has address 40.97.155.26
  - outlook.com has address 40.96.32.50
  - outlook.com has address 40.97.156.114
- Each DNS reply has an associated TTL
  - if *load-balancing* is used, TTL must be small (*e.g. 18 seconds*)

You Try It: nslookup

Imperial College London

- The **nslookup** command can be used to find various details relating to DNS
- For example, by typing `nslookup www.imperial.ac.uk` locally, we get

Locally:

```
Server: 127.0.1.1
Address: 127.0.1.1#53
www.imperial.ac.uk      canonical name = wrp.cc.gslb.ic.ac.uk.
Name:   wrp.cc.gslb.ic.ac.uk
Address: 146.179.40.24
```

Remotely:

```
Server:  MyRouter
Address:  192.168.1.1
Non-authoritative answer:
Name:    wrp.cc.gslb.ic.ac.uk
Addresses:  2001:630:12:600:1:2:0:10b
            146.179.40.24
Aliases:  www.imperial.ac.uk
```

- The first line specifies the address of the DNS server used
- **Non-authoritative** means that the reply has been extracted from a previous *cache*
  - `nslookup -type=NS imperial.ac.uk` tells us the address of the authoritative DNS server
  - `nslookup www.imperial.ac.uk ns0.ic.ac.uk` asks the server ns0 (*the authoritative DNS*) for a reply

You Try It: dig

Imperial College London

- Domain Information Groper (**dig**) is a network tool that queries DNS name servers
  - e.g. **dig www.imperial.ac.uk**
- dig also enables querying for other DNS records
  - e.g. for the mail server, type:
    - **dig MX imperial.ac.uk**
- similar information can also be obtained by running the **host** tool with the -v option

```
; <<>> DiG 9.10.3-P4-Ubuntu <<>> www.imperial.ac.uk
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 11799
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 2, ADDITIONAL: 3

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.imperial.ac.uk.            IN      A

;; ANSWER SECTION:
www.imperial.ac.uk.     300     IN      CNAME   wrp.cc.gslb.ic.ac.uk.
wrp.cc.gslb.ic.ac.uk.   16      IN      A       146.179.40.24

;; AUTHORITY SECTION:
gslb.ic.ac.uk.          86400   IN      NS      gslb2.net.ic.ac.uk.
gslb.ic.ac.uk.          86400   IN      NS      gslb1.net.ic.ac.uk.

;; ADDITIONAL SECTION:
gslb1.net.ic.ac.uk.     900     IN      A       192.195.116.3
gslb2.net.ic.ac.uk.     900     IN      A       192.195.117.3

;; Query time: 1 msec
;; SERVER: 127.0.1.1#53(127.0.1.1)
;; WHEN: Thu Jan 26 18:00:00 GMT 2017
;; MSG SIZE  rcvd: 168
```

Content Distribution Networks (CDNs)

- Challenge
  - How to stream content (*selected from millions of videos*) to hundreds of thousands of *simultaneous* users?

- Option #1
  - single, large, "mega-server"
    - single point of failure
    - point of network congestion
    - long path to distant clients
    - multiple copies of video sent over outgoing link

- This solution does not scale..!

Content Distribution Networks (CDNs)

- Challenge
  - How to stream content (*selected from millions of videos*) to hundreds of thousands of *simultaneous* users?

- Option #2
  - Store/serve multiple copies of videos at multiple geographically distributed sites (CDN)
    - **enter deep**: push CDN servers deep into many access networks
      - close to users
      - used by Akamai (*216,000+ servers, 120+ countries, 1,500+ networks*)
    - **bring home**: smaller number (10s) of larger clusters in PoPs (Points of Presence) near (*but not within*) access networks
      - used by Limelight (*80+ PoPs*)

CDN: “simple” content access scenario

Imperial College London

- Bob (=*client*) requests a video `http://netcinema.com/6Y7B23V`
- The video is stored on a CDN at `http://KingCDN.com/224408`

1. Bob clicks on the URL for video `http://netcinema.com/6Y7B23V` from the Google search results page
2. `netcinema.com` is resolved via Bob’s local DNS server
3. Bob connects to netcinema but the page returns that the video is at `http://KingCDN.com/224408`
4. Bob’s system asks for the IP of `KingCDN.com`, but its authoritative DNS server is stored instead
5. Bob’s DNS server asks the KingCDN DNS server for the IP Bob requested, and then returns it
6. Bob’s system requests the video from the KingCDN server, streamed via HTTP

CDN cluster selection strategy

Imperial College London

- An essential issue:
  - How does a CDN DNS select a “good” CDN node to stream to the client?
    a) pick a CDN node geographically closest to client
    b) pick a CDN node with shortest delay to client
    (*CDN nodes periodically ping access ISPs, reporting results to CDN DNS*)

- How does the CDN know the IP address of the client?
  - It doesn’t.
  - It only knows the address of the local DNS. This may or may not be fully accurate.

- Google Public DNS
  - Google offers a free public DNS service (*8.8.8.8 and 8.8.4.4*)
    - This allows it to track the client’s actual IP address and perform better redirection

CDN cluster selection strategy

- Alternative:
    - let the client decide
        - give the client a list of several CDN servers
            - client pings servers and picks "best"
                - usually based on **RTT** (Round-Trip Time)
                - (*i.e. the time it takes for a message to reach its destination* + *the time we waited for the server's acknowledgement of receipt*)
    - Netflix approach

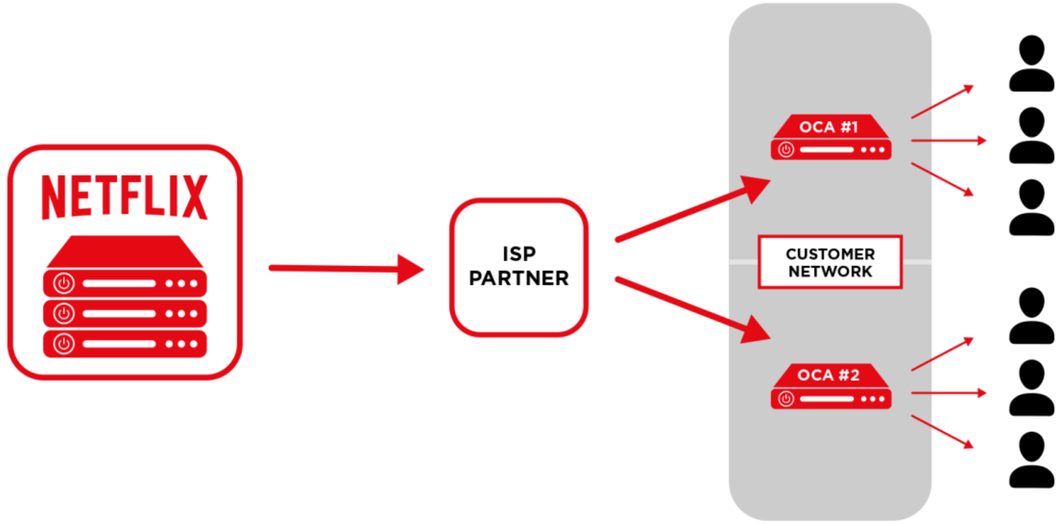
Case study: Netflix

- ~37% downstream US traffic in 2015-2016
- Used to own very little infrastructure; used 3rd party services
  - own registration and payment servers
  - Amazon (*i.e. 3rd party*) Web Services (AWS):
    - Netflix uploads studio master to Amazon cloud
    - Creates multiple versions of movie (*different encodings*) in the cloud
    - Uploads versions from cloud to CDNs
    - Cloud hosts Netflix webpages for user browsing
- However, in 2012 it started moving to its own "Open Connect" peering CDN platform
  - "*to reduce Netflix's reliance (and associated costs) on commercial CDNs*"
    - their website is still on AWS

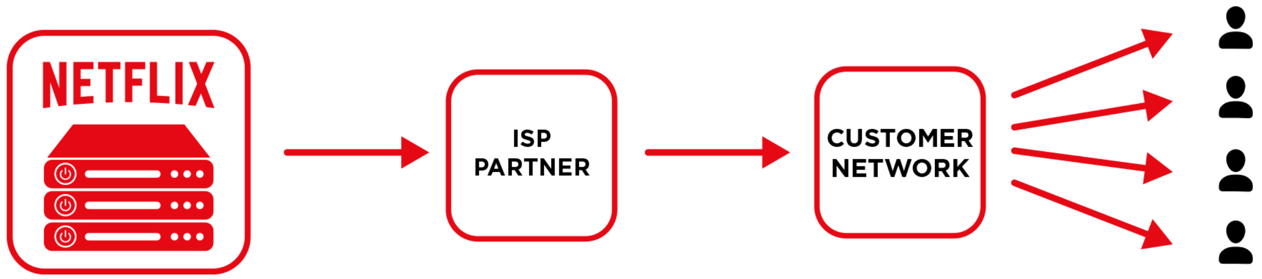
Case study: Netflix (cont'd)

Imperial College London

- The Netflix "Open Connect" peering network uses both CDN approaches:
  - enter-deep equivalent



  - bring-home equivalent



A Postal Service for the Internet

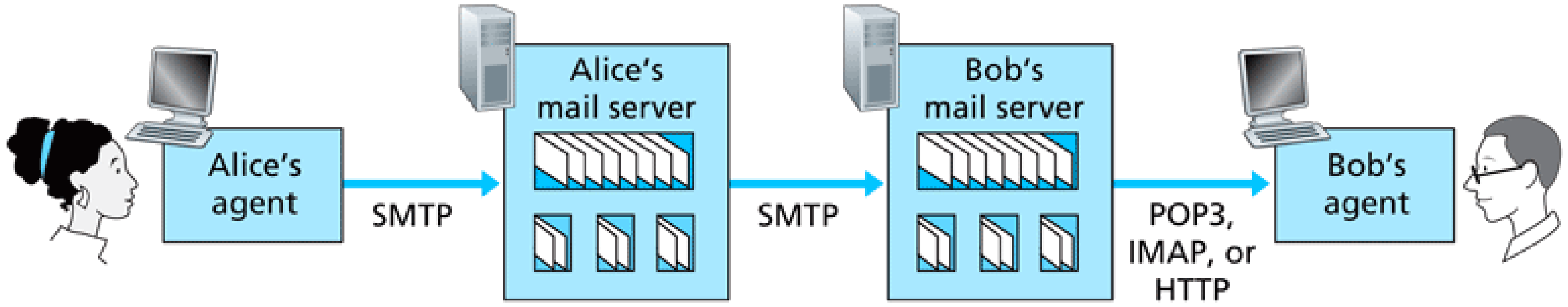
From:
Bob Smith
180 Queen's Gate
London, SW7 2AZ, UK
To:
Alice Doe
56 Wood Ln
London, W12 7SB, UK

Electronic Mail

Imperial College London

- **Features and Goals**
  - Asynchronous communication
    - Alice sends a message when it is convenient to her
    - Bob reads Alice's message whenever he has time to do that
  - One-to-many communication
    - Alice can send a message to Bob and Charlie
    - A mailing list sends messages to several receivers
  - Multimedia content
    - Images and all sorts of attachments as well as normal text
- **Limitations**
  - No authentication
    - messages can be modified
    - messages can be forged
  - No confidentiality
    - the message can be read by others
  - Little or no delivery guarantees
    - messages can be accidentally lost or intentionally blocked
    - no reliable acknowledgement system

Architecture



- User agent
  - allows a user to read, compose, reply to, send, and forward messages, as well as to save, classify, sort, search, [...]
- Mail servers
  - accept messages for remote delivery
    - store messages in a local persistent queue
    - deliver messages to a remote (*destination*) server using the *transport protocol*
  - accept messages for local delivery
    - save messages in some local *persistent* mailbox
  - allow user agents to access local mailboxes
    - user agents can retrieve and/or delete messages
    - this is done through an access protocol

Email: Message Transfer

Imperial College London

- The message transfer agent extracts the destination host from the message, and queries DNS to obtain the destination address
  - **Remember**: DNS also keeps track of mailers in MX records

```
nslookup -type=MX imperial.ac.uk
Server: 127.0.1.1
Address: 127.0.1.1#53
imperial.ac.uk  mail exchanger = 10 mx3.cc.ic.ac.uk.
[...]
```

- **Example**: mail for `email@ic.ac.uk` is sent to the message transfer agent on `mx3.cc.ic.ac.uk`
  - of course, mx3 is then looked up as well to retrieve its IP address
- The message transfer agent extracts the username and makes an attempt to deposit the incoming message into the user's mailbox. The user may then be notified
- **Note**: We are assuming that the user's mailbox is accessible for the agent
  - protocols exist that allow a user to remotely access their mailbox

Simple Mail Transfer Protocol (SMTP)

- **SMTP**: Simple Mail Transfer Protocol
  - connection-oriented protocol
    - TCP

- Really simple (*a reason of its success*):
  1. set up TCP/IP connection between client and server
  2. client requests server to accept its messages
  3. server responds, so that client can send

- It is an *old* protocol, compared to HTTP
  - the first RFCs date back to the early 80s
  - it has some archaic characteristics:
    - lines should be 1000 characters or less
    - it is restricted to 7-bit characters (i.e. US ASCII only)
      - but we worked around this

SMTP – Example

Imperial College London

```
HELO gmail.com
MAIL FROM: imperialcollege@gmail.com
RCPT TO: email@ic.ac.uk
DATA
From: Imperial College <imperialcollege@gmail.com>
To: Test User <email@ic.ac.uk>
Subject: This is a test

Dear Test,
Neque porro quisquam est, qui dolorem ipsum, quia dolor sit, amet, consectetur, adipisci velit, sed quia non numquam eius modi tempora incidunt, ut labore et dolore magnam aliquam quaerat voluptatem.
Thanks,
-IC

.
QUIT
```

SMTP – Message Format

Imperial College London

```
HELO gmail.com
MAIL FROM: imperialcollege@gmail.com
RCPT TO: email@ic.ac.uk
DATA
From: Imperial College <imperialcollege@gmail.com>
To: Test User <email@ic.ac.uk>
Subject: This is a test

Dear Test,
Neque porro quisquam est, qui dolorem ipsum, quia dolor sit,
quia non numquam eius modi tempora incidunt, ut labore
voluptatem.
Thanks,
-IC

.
QUIT
```

$\}$ headers

$\}$ empty line

$\}$ content

$\}$ dot to end email

$\}$ QUIT to exit

SMTP – Message Format (cont'd)

Imperial College London

- Basic idea: Don't describe anything concerning the content of a message; only specify the header
  - **To**: e-mail address(es) main destination
  - **Cc**: e-mail address(es) to send copies
  - **Bcc**: e-mail address(es) to send blind copies
  - **From**: name of sender(s)
  - **Sender**: e-mail address sender
  - **Received**: line added by each intermediate transfer agent
  - **Return-path**: return address
  - **Date**: the date and time the message was sent
  - **Subject**: short summary of the message
  - **Reply-To**: E-mail address to which replies should be sent

- The *From:* field is often the same as *Sender:*, so the latter can be left out

Received: Headers

- SMTP is almost completely oblivious to the content of a message.
  - One exception is the *Received:* header

- Every receiving SMTP server must add a Received: header
  - MIME-Version: 1.0 Received: from mx3.cc.ic.ac.uk
  - (2001:630:12:601::154) by icexch-h2.ic.ac.uk (2001:630:12:610::112)
  - with Microsoft SMTP Server id 14.3.174.1; Fri, 27 Jan 2017 11:30:00
  - +0000 Received: from ictmm6.cc.ic.ac.uk ([155.198.5.59]) by
  - mx3.cc.ic.ac.uk with esmtp (Exim 4.80.1) (envelope-from
  - <imperialcollege@gmail.com>) id 1W6j8R-0005K> for email@ic.ac.uk;
  - Fri, 27 Jan 2017 11:30:00 +0000 Received: from mx2.cc.ic.ac.uk (Not
  - Verified[155.198.5.152]) by ictmm6.cc.ic.ac.uk with MailMarshal
  - (v6,8,4, 9558) id <B52e28d6f0002>; Fri, 27 Jan 2017 11:30:00 +0000
  - Received: from mail-qc0-x234.google.com ([2607:f8b0:400d:c01::234]) by […]

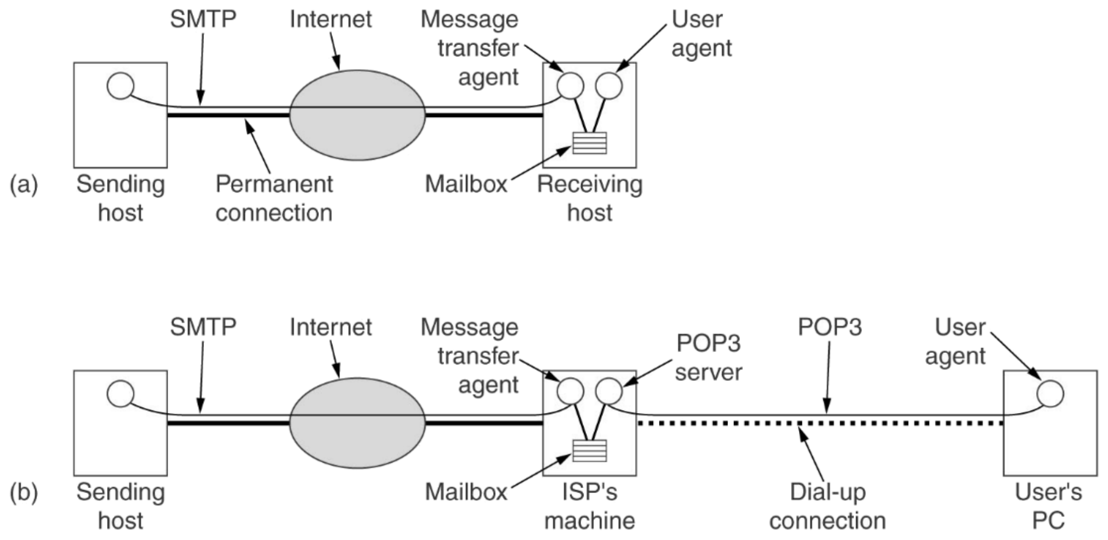
Limitations of the Message Format

Imperial College London

- The standard message format has some serious limitations
  - 7-bit (text) content
  - only text
  - essentially good exclusively for the English language
- Any content that does not fit the 7-bit (ASCII) character set must be *encoded*
- The *Multipurpose Internet Mail Extensions* (MIME) specification (*RFC 2045 and RFC 2046*) defines useful extensions
- Valid types include:
  - `text/plain` — this is a normal ASCII message
  - `text/html` — this is an HTML-formatted message
  - `image/jpeg` — this message contains (only) an image file
  - `multipart/mixed` — this message consists of multiple parts

POP3

- **Problem**: a user's mailbox may be stored on a different machine than the user agent. We need remote access to incoming (*and also outgoing*) messages
- **Solution**: use the **Post Office Protocol** (POP3):



- The POP3 protocol is defined in RFC 1939

POP3 – Example

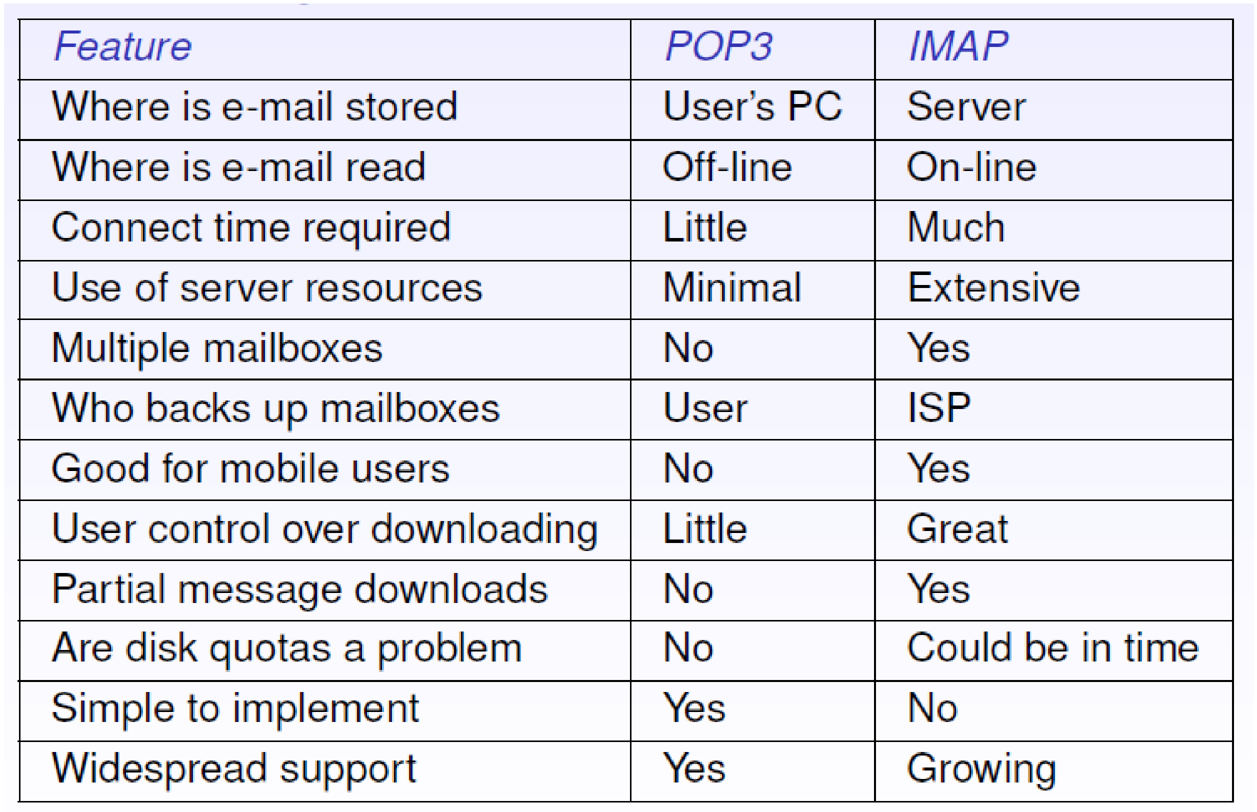
Imperial College London

- You can also try to connect to you ISP's POP3 server
- `telnet pop3.example.com 110`
- +OK POP3 server ready
- USER kgk
- +OK
- PASS longcatislong
- +OK login successful
- LIST
- 1 2505
- .
- RETR 1
- *(pops/sends message #1)*
- DELE 1
- QUIT
- +OK POP3 server disconnecting

IMAP

- POP3 (*implicitly*) assumes that retrieved mail is deleted at the server
  - not very useful for people wanting to access mail from different computers

- The Internet Message Access Protocol solves this problem.



Feature	*POP3*	*IMAP*
Where is e-mail stored	User's PC	Server
Where is e-mail read	Off-line	On-line
Connect time required	Little	Much
Use of server resources	Minimal	Extensive
Multiple mailboxes	No	Yes
Who backs up mailboxes	User	ISP
Good for mobile users	No	Yes
User control over downloading	Little	Great
Partial message downloads	No	Yes
Are disk quotas a problem	No	Could be in time
Simple to implement	Yes	No
Widespread support	Yes	Growing

The "dark" side of the Internet

- Dark Web, as in *hidden*
  - *but sometimes also potentially dangerous*

- Tor
- ZeroNet
- i2p
- Freenet
- IPFS

- Also:
  - VPN
  - ssh (tunnel)
    - not telnet

- (*Not to be confused with the Deep Web*)

Fun times with Networking

- Use the command **tcpdump** to analyse packets that are being transmitted over the network. A very nice tutorial is available at: http://www.thegeekstuff.com/2010/08/tcpdump-command-examples/
- To view and analyse packets in a user-friendly manner, you can install and use **Wireshark** (http://www.wireshark.org), a free open-source Packet Analyser (*a.k.a. sniffer*).
  Caution: *You should not run this tool on the Imperial College network.*
- You have seen that SMTP email is sent as plain-text and can potentially be read by other parties (*such as your email provider*). GnuPG is the free implementation of the **OpenPGP** standard and allows you to encrypt and sign your data and communication. Use it to send and receive encrypted email. You can send me one as well if you want (*my public key is available on my IC website*).
- On Linux, what information do you get from the file **/etc/resolv.conf**? Is it the same on different machines in different environments? If not, why?
- Use a *command line* tool to find out the IP address of your machine. How different is the information provided on different OSs?
- On Linux, what information do you get from the file **/etc/services**?

Q&A

Imperial College London

- You will find the second **exercise worksheet** on the course website(s) today
  - *Solutions will be uploaded on the DoC website at the end of next week*
- **Suggested reading**: Tanenbaum#7,8; Kurose#2; (*Peterson#9,8; Stallings#8*)
- Please provide *anonymous* feedback on www.menti.com using the code **49 80 49**
  - *always active throughout the term*
- You can also provide *eponymous* feedback or ask questions via email (*username: **kgk***)
- Thank you for your attention
- **Movie of the week**: Pirates of Silicon Valley
- **Next time**: To Connect, Or Not To Connect? (*The Transport Layer*)