The University of Melbourne
School of Computing and Information Systems

# COMP30023
## Computer Systems

Semester 1, 2017

# Applications

# Topics to be covered

1. Applications services/protocols
2. DNS
3. Email
4. WWW
    - HTTP
5. Streaming applications (time permitting)
5. VoIP (brief introduction, time permitting)

# Application layer

The Application layer protocols define:

- Types of messages exchanged: e.g., request, response
- Message syntax: what fields in messages and how fields are delineated
- Message semantics: meaning of information in fields
- Rules for when and how processes send and respond to messages

Public-domain protocols (defined in RFCs) allows for interoperability, eg. HTTP, SMTP, BitTorrent.

Proprietary protocols exist. eg: Skype is a well-known example

# What transport service does an app need?

**Data loss**: Some apps (eg, audio) can tolerate some loss, while other apps (eg, file transfer, telnet) require 100% reliable data transfer

**Timing**: Some apps (eg Internet telephony, interactive games) require low delay to be "effective".

**Throughput**: Some apps (eg. multimedia) require a minimum amount of throughput to be "effective", while other apps (elastic apps) make use of whatever throughput they get.

**Security**: issues such as encryption, data integrity must be considered.

# Transport service requirements of common apps

| Application | Data loss | Throughput | Time Sensitive |
|---|---|---|---|
| file transfer | no loss | elastic | no |
| e-mail | no loss | elastic | no |
| Web documents | no loss | elastic | no |
| real-time audio/video | loss-tolerant | audio: 5kbps-1Mbps<br>video:10kbps-5Mbps | yes, 100's msec |
| stored audio/video | loss-tolerant | same as above | yes, few secs |
| interactive games | loss-tolerant | few kbps up | yes, 100's msec |
| instant messaging | no loss | elastic | yes and no |

# 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 (eg Youtube), RTP [RFC 1889] | TCP or UDP |
| Internet telephony | SIP, RTP, proprietary (e.g., Skype) | typically UDP |

# DNS – Domain Name System

DNS is essentially the technology behind mapping `host.domain.com` to an IP address. Four elements comprise the DNS:

**Domain name space**: DNS uses a tree-structured *name space* to identify resources on the Internet.

**DNS database**: Each node/leaf in the name space tree names a set of information that is contained in a resource record (RR). The collection of all RRs is organized into a distributed database.

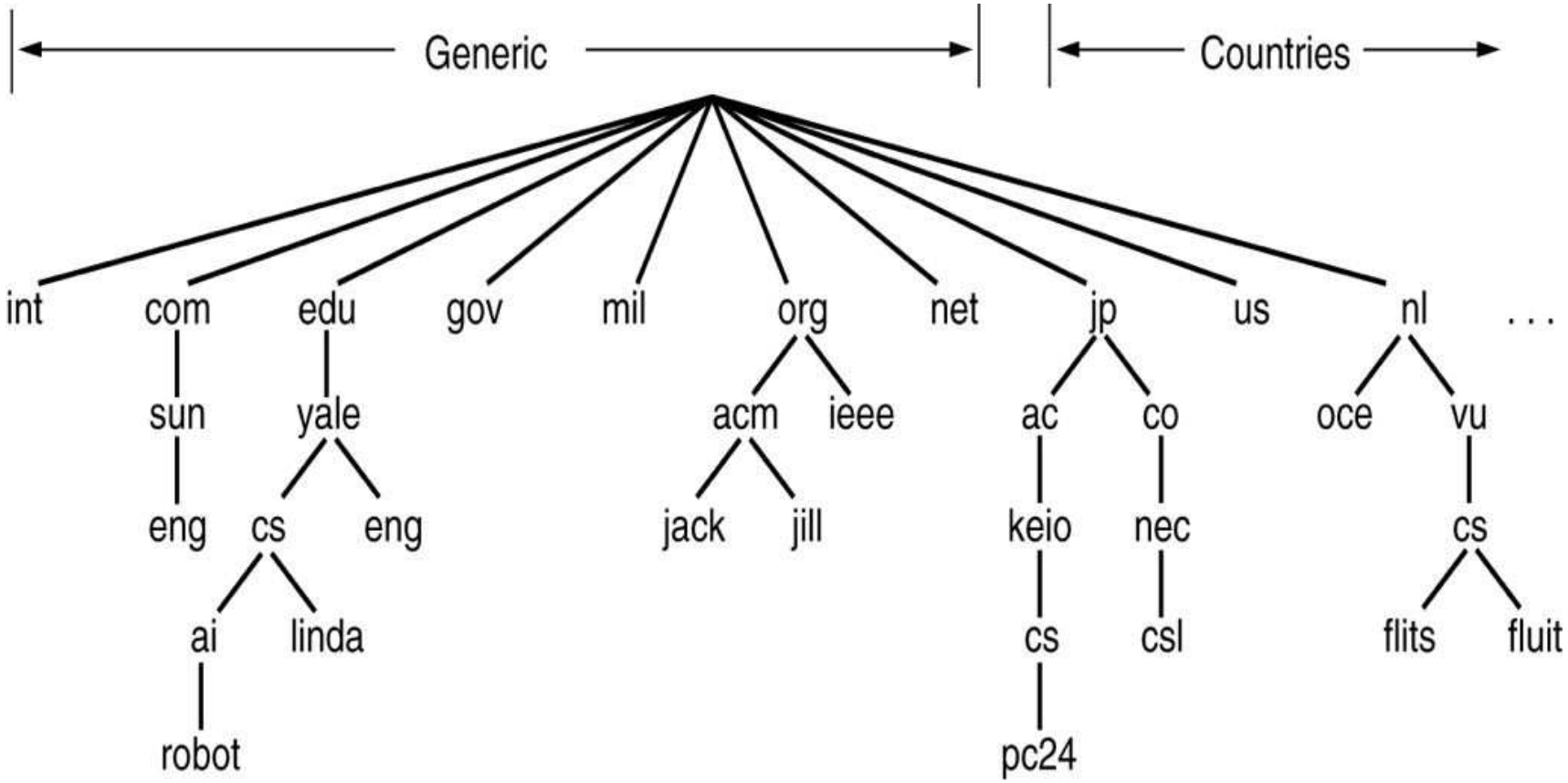
**Name servers**: Server programs that hold information about a portion of the domain name tree structure and the associated RRs.

**Resolvers**: These are programs that extract information from name servers in response to client requests.

# Domain name characteristics

- Domain names:
  - are case insensitive
  - can have up to 63 characters per constituent
  - can have up to 255 chars per path
  - can be internationalised (since 1999)
- Naming conventions usually follow either organisational or physical boundaries eg.
  - `au.ibm.com` / `uk.ibm.com` (for email)
  - `ibm.com.au` / `ibm.co.uk` (for web)
- Absolute domain names ends in a '.'
- Relative domain names end in a constituent eg .com

# Conceptual divisions of DNS namespace




A portion of the Internet domain name space.

# DNS namespace

| Domain | Intended use | Start date | Restricted? |
|---|---|---|---|
| com | Commercial | 1985 | No |
| edu | Educational institutions | 1985 | Yes |
| gov | Government | 1985 | Yes |
| int | International organizations | 1988 | Yes |
| mil | Military | 1985 | Yes |
| net | Network providers | 1985 | No |
| org | Non-profit organizations | 1985 | No |
| aero | Air transport | 2001 | Yes |
| biz | Businesses | 2001 | No |
| coop | Cooperatives | 2001 | Yes |
| info | Informational | 2002 | No |
| museum | Museums | 2002 | Yes |
| name | People | 2002 | No |
| pro | Professionals | 2002 | Yes |
| cat | Catalan | 2005 | Yes |
| jobs | Employment | 2005 | Yes |
| mobi | Mobile devices | 2005 | Yes |
| tel | Contact details | 2005 | Yes |
| travel | Travel industry | 2005 | Yes |
| xxx | Sex industry | 2010 | No |

Generic top-level domains.

# Resource records

The principal DNS resource record types

| Type | Meaning | Value |
|---|---|---|
| SOA | Start of authority | Parameters for this zone |
| A | IPv4 address of a host | 32-Bit integer |
| AAAA | IPv6 address of a host | 128-Bit integer |
| MX | Mail exchange | Priority, domain willing to accept email |
| NS | Name server | Name of a server for this domain |
| CNAME | Canonical name | Domain name |
| PTR | Pointer | Alias for an IP address |
| SPF | Sender policy framework | Text encoding of mail sending policy |
| SRV | Service | Host that provides it |
| TXT | Text | Descriptive ASCII text |

# Resource record example

```
; Authoritative data for cs.vu.nl
cs.vu.nl.       86400   IN  SOA     star boss (9527,7200,7200,241920,86400)
cs.vu.nl.       86400   IN  MX      1 zephyr
cs.vu.nl.       86400   IN  MX      2 top
cs.vu.nl.       86400   IN  NS      star

star            86400   IN  A       130.37.56.205
zephyr          86400   IN  A       130.37.20.10
top             86400   IN  A       130.37.20.11
www             86400   IN  CNAME   star.cs.vu.nl
ftp             86400   IN  CNAME   zephyr.cs.vu.nl

flits           86400   IN  A       130.37.16.112
flits           86400   IN  A       192.31.231.165
flits           86400   IN  MX      1 flits
flits           86400   IN  MX      2 zephyr
flits           86400   IN  MX      3 top

rowboat                 IN  A       130.37.56.201
                        IN  MX      1 rowboat
                        IN  MX      2 zephyr

little-sister           IN  A       130.37.62.23

laserjet                IN  A       192.31.231.216
```

A portion of a possible DNS database for `cs.vu.nl`

# Inserting records into DNS

Example: new startup Network Utopia

Register name `networkuptopia.com` at DNS registrar (e.g., Network Solutions)

- provide names, IP addresses of authoritative name server (primary and secondary)
- registrar inserts two RRs into com TLD server:
  (`networkutopia.com, dns1.networkutopia.com, NS`)
  (`dns1.networkutopia.com, 212.212.212.1, A`)
- create authoritative server:
  Type A record for `www.networkuptopia.com`;
  Type MX record for `networkutopia.com`

# Demonstration

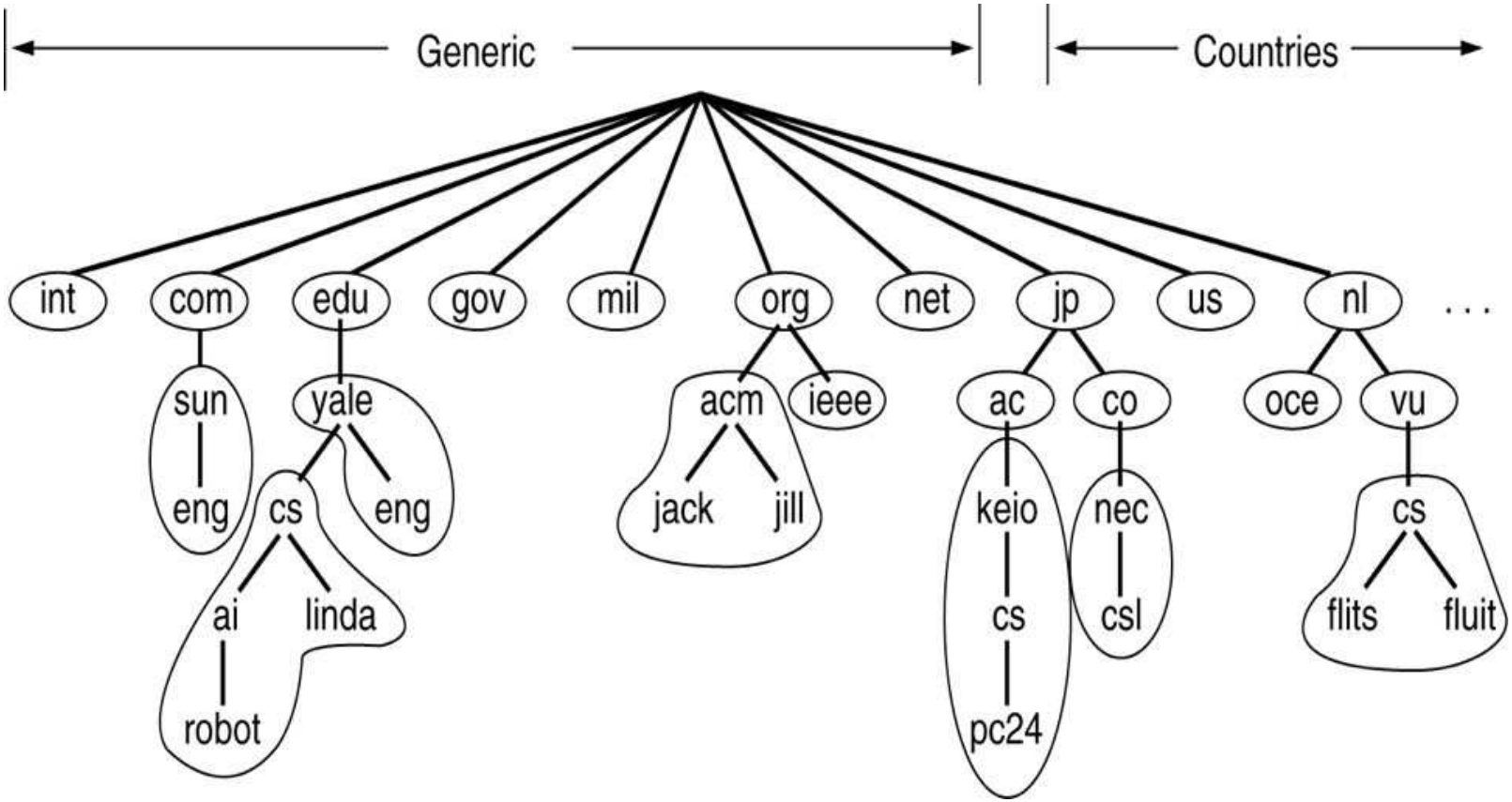
Using DNS query tools:

`nslookup`

`dig`

`host`

# Name servers




Part of the DNS name space divided into zones

# Name servers

Zones:

- DNS namespace is divided into overlapping zones. The name servers are authoritative for that zone.
  - usually two nameservers for a zone
- Name servers are arranged in a hierarchical manner extending from a set of root servers

Root name servers:

- The root servers form the authoritative cluster for enquiries. The root servers are contacted by a local name server that can not resolve name.
- There are 13 root names servers globally (a root server may be a cluster of servers in IP space)

F-ROOT 13 servers; J-ROOT 8 servers

# Name servers

**Top-level domain DNS servers**: responsible for com, org, net, edu, etc, and all top-level country domains uk, fr, au, jp. Example include: *Network Solutions* maintains servers for com; and *Educause* for edu

**Authoritative DNS servers**: organizations DNS servers, providing authoritative hostname to IP mappings for organizations servers (e.g., Web, mail). Can be maintained by organization or service provider.

**Local DNS server**: does not strictly belong to hierarchy. Typically, each ISP (residential ISP, company, university) has a "default name server" which handles DNS queries – a query is sent to its local DNS server acts as proxy, forwards query into hierarchy

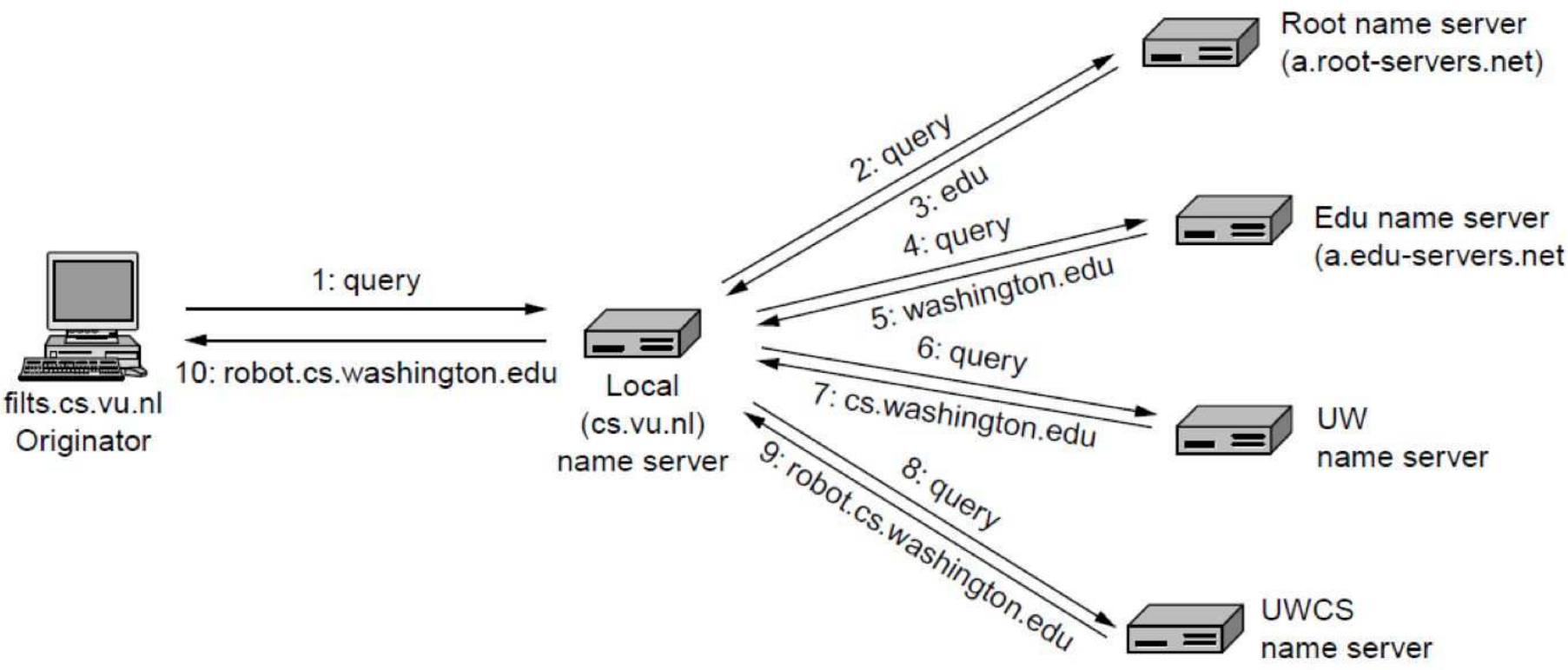
# Resolving a query

A **resolver client** asks the **local DNS** for the domain to IP mapping:

- if answer is known by the local DNS, then it sends the answer.
- if answer is not known, then the **local DNS** queries up the hierarchy to the **top level (root) DNS** for the domain and then relays the answer to the resolver client.

Essentially, this is a recursive query mode. Queries are subject to timers to avoid longer than necessary response times.

# Resolving a query




Example of a resolver looking up a remote name in 10 steps

# Email services and architecture

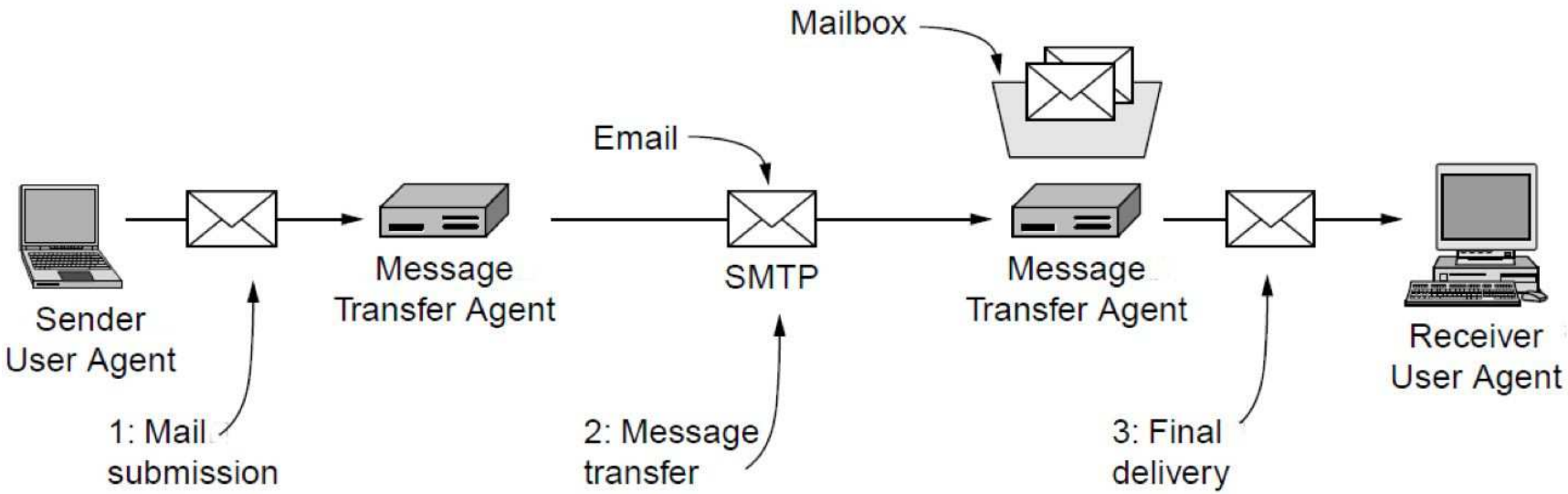
Email has a long heritage (since 1960's), in this time evolutionary steps in infrastructure and standards have been taken.

- Standards for Internet-enabled email are based on 2 RFC's
  - RFC 821 (transmission)
  - RFC 822 (message format)
  - RFC 2821 and RFC 2822 (revised versions of earlier RFCs)
- Architecture and Services
  - **User agents** (UA's/ MUA's)

    allow user to read and send email
  - **Message transfer agents** (MTA's)

    transport messages from source - destination

# Email services and architecture



# User agent (or mail program)

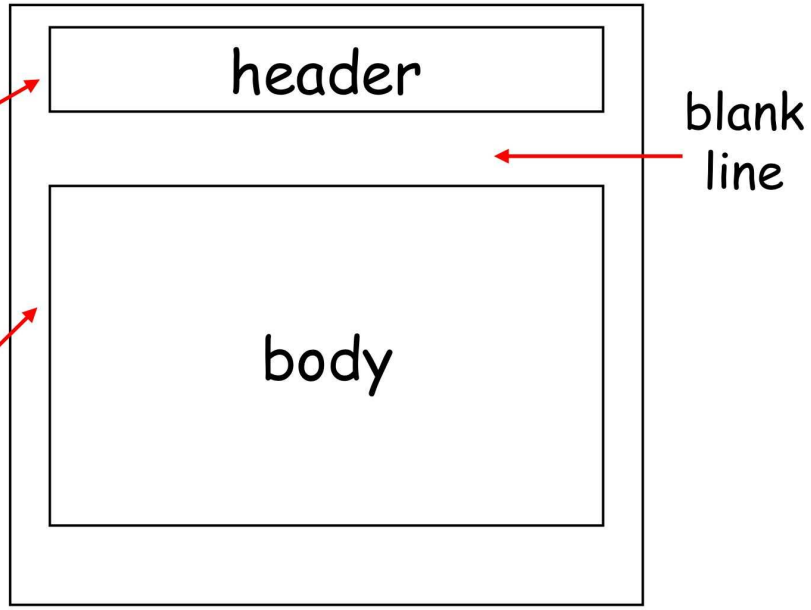
- Basic functions: compose, report, display, dispose
- **Envelope** and **contents**: encapsulation of transport related information
- **Header** and **body**: header - user agent control info; body for human recipient
- User must provide message, destination, optional other parameters
- Addressing scheme `user@dns-address`

# Mail message format

SMTP: protocol for exchanging email messages

RFC 822: standard for text message format:

- header lines, e.g.,
  - To:
  - From:
  - Subject:
- body
  - the "message", ASCII characters only



# Message header fields

- To:
- Cc:
- Bcc:
- From:
- Sender:
- Received:
- Return-Path:
- Date:
- Reply-To:
- Message-Id:
- In-Reply-To:
- References
- Keywords:
- Subject:

# SMTP – Simple Message Transfer Protocol

SMTP

- uses TCP to reliably transfer email message from client to server, port 25
- direct transfer: sending server to receiving server
- three phases of transfer (1) handshaking (greeting), (2) transfer of messages, and (3) closure
- command/response interaction: commands in ASCII text and response consists of status code and phrase
- messages must be in 7-bit ASCII

# SMTP interaction example

```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250  Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?
C: How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```

# MIME – Multipurpose Internet Mail Extentsions

In the early days of email, messages were in English and used only ASCII - RFC 822 reflects these simple constraints. In time, the inadequacy of RFC822 became apparent: eg., other language requirements and alternative message content type (audio/images)

MIME has 5 additional message headers:

- MIME-Version: identifies the MIME version
- Content-Description: human readable describing contents
- Content-Id: unique identifier
- Content-Transfer-Encoding: how body is wrapped for transmission
- Content-Type: type and format of content

# MIME

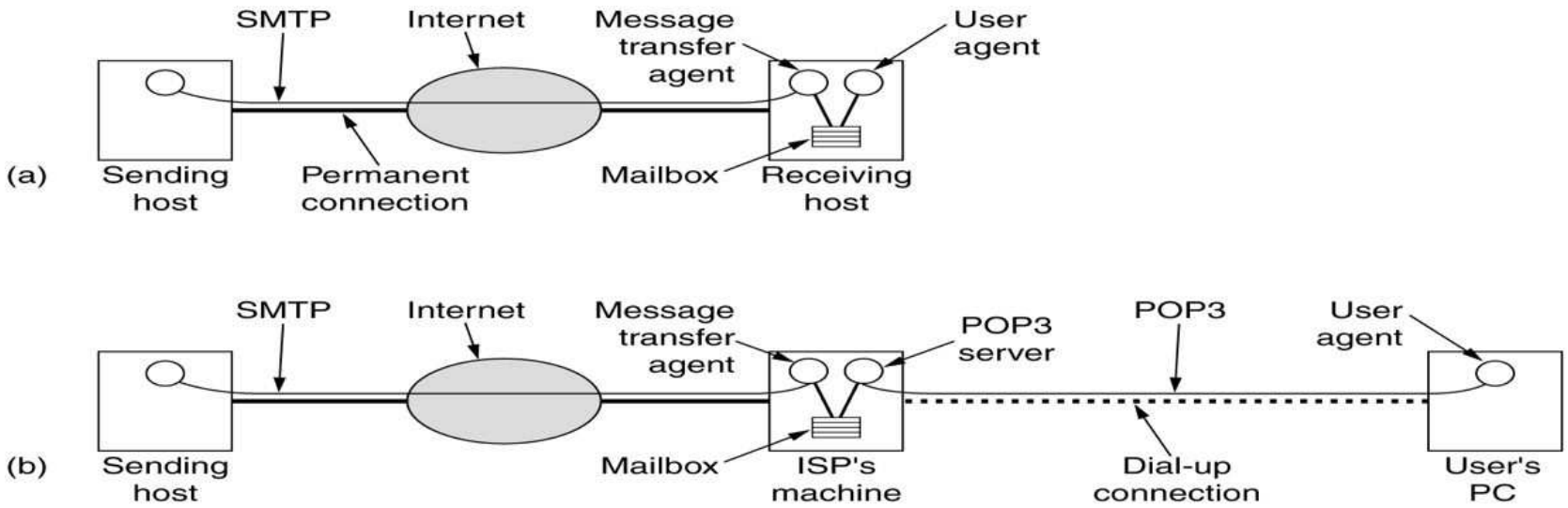
| Type | Example subtypes | Description |
|---|---|---|
| text | plain, html, xml, css | Text in various formats |
| image | gif, jpeg, tiff | Pictures |
| audio | basic, mpeg, mp4 | Sounds |
| video | mpeg, mp4, quicktime | Movies |
| model | vrml | 3D model |
| application | octet-stream, pdf, javascript, zip | Data produced by applications |
| message | http, rfc822 | Encapsulated message |
| multipart | mixed, alternative, parallel, digest | Combination of multiple types |

MIME content types and example subtypes.

# Message transfer and access protocol

- Transfer
  - **SMTP**: delivery/storage to receiver's server
- Delivery
  - **Local**
  - **POP3** : Post Office Protocol; authorization (agent – server) and download
  - **IMAP**: Internet Mail Access Protocol; more features (more complex); provides for the manipulation of stored messages on server
  - **HTTP**: gmail, Hotmail, Yahoo! Mail, etc.

# Receiving mail: local vs remote




(a) Sending and reading mail when the receiver has a permanent Internet connection and the user agent runs on the same machine as the message transfer agent.

(b) Reading e-mail when the receiver has a dial-up connection to an ISP.

# POP3 – Post Office Protocol

- Three states of a POP3 transaction
  - Authorisation
  - Transactions
  - Update
- Syntax
  - USER / PASS
  - LIST
  - RETR / DELE
  - QUIT (update)
- Issue: "*download and delete*" mode does not allow messages to be re-read.

# POP3

## authorization phase

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

## transaction phase, client:

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

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

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

# IMAP – Internet Message Access Protocol

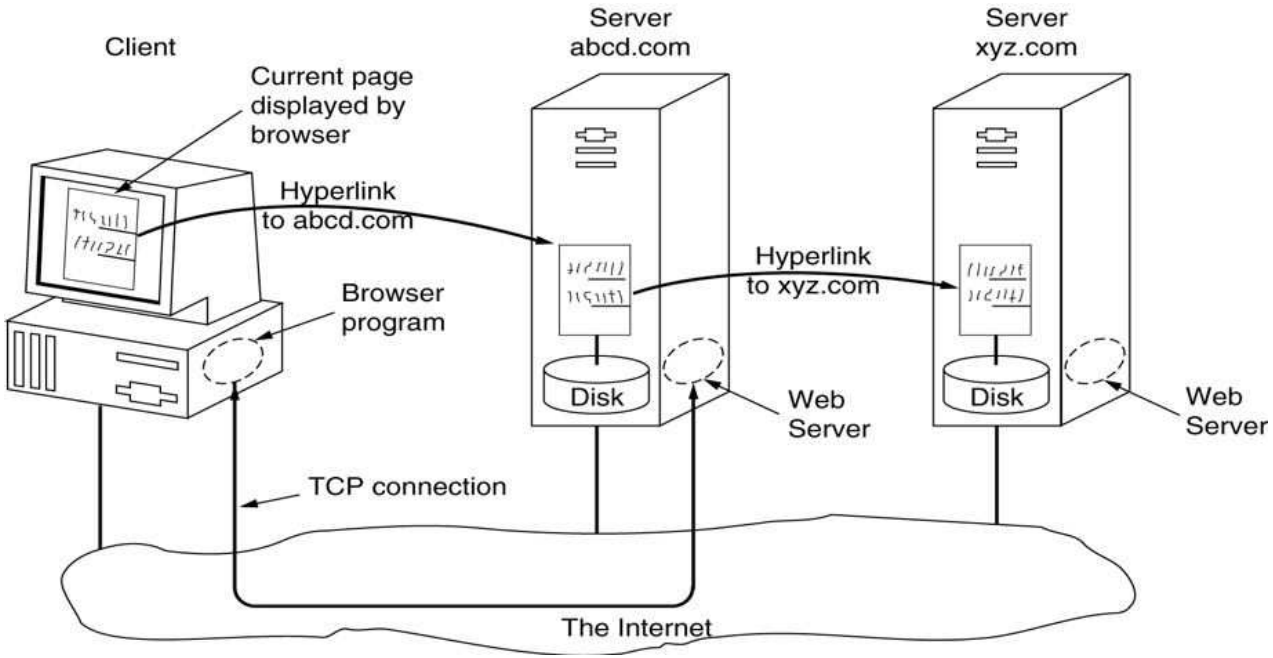
IMAP keeps user state across sessions.

- Retain mailbox contents online (server) and allow manipulation of online and offline messages and mailbox folders
- Implications of server infrastructure to support high volume of IMAP users. This implies storage projections by the provider, and hence limitations.

# World Wide Web - a potted history

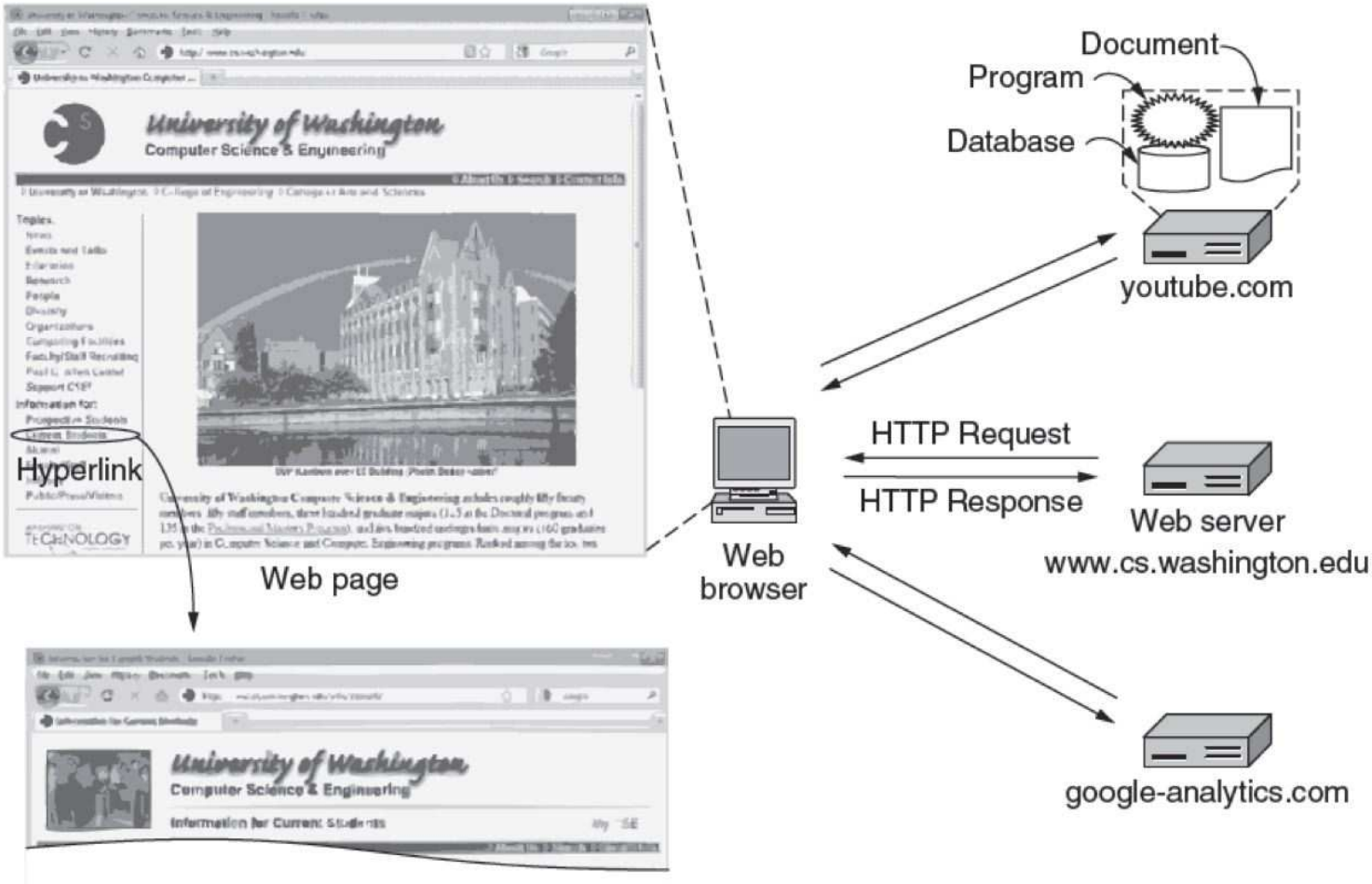
- History: CERN 1989, Hypertext 1991 Conf - Tim Berners-Lee
- Marc Andreessen @ UIUC Mosaic 1993, then Netscape
- Browser wars MS and Netscape 1994-1998
- W3C 1994 (CERN, MIT) internet standards body
- 1999-2001 “dot com” era
- 2002+ the ubiquitous nature of the Web
- Web 2.0 towards a semantic web; multimedia apps; “social networking”

# WWW component technologies




- Client - typically a browser based access to pages
- Server - daemon based content delivery of pages
- URL – **Protocol** + **DNS Name** + **file name**

# WWW architectural overview



# WWW architectural overview

Some common URL schemes:

| Name | Used for | Example |
|---|---|---|
| http | Hypertext (HTML) | http://www.ee.uwa.edu/~rob/ |
| https | Hypertext with security | https://www.bank.com/accounts/ |
| ftp | FTP | ftp://ftp.cs.vu.nl/pub/minix/README |
| file | Local file | file:///usr/suzanne/prog.c |
| mailto | Sending email | mailto:JohnUser@acm.org |
| rtsp | Streaming media | rtsp://youtube.com/montypython.mpg |
| sip | Multimedia calls | sip:eve@adversary.com |
| about | Browser information | about:plugins |

# HTTP – Hypertext Transfer Protocol

Overview:

- 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

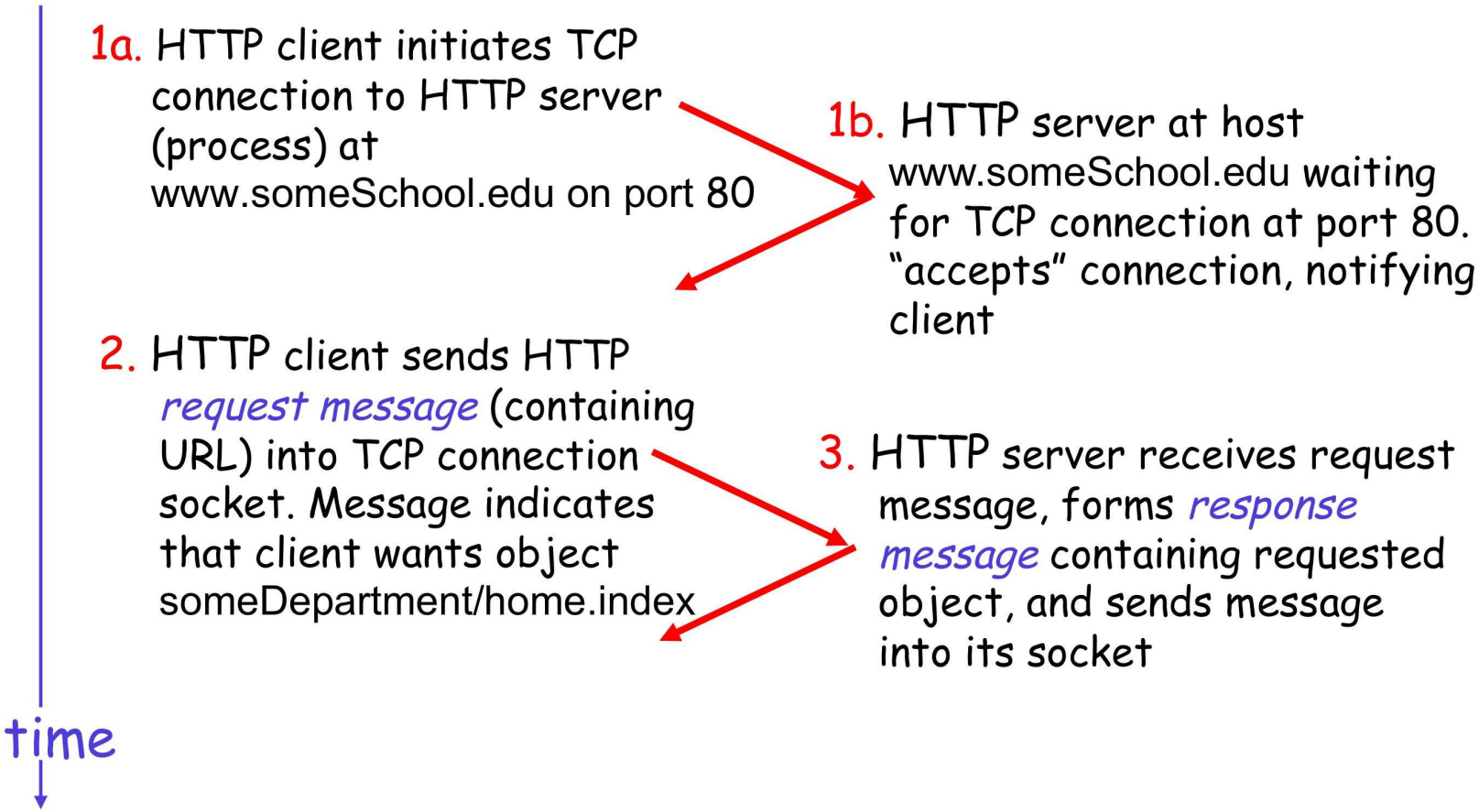
Connections:

- HTTP 1.0 – single connect for each transaction for each client-server pair,
- HTTP 1.1 – persistent connections per server-client pair

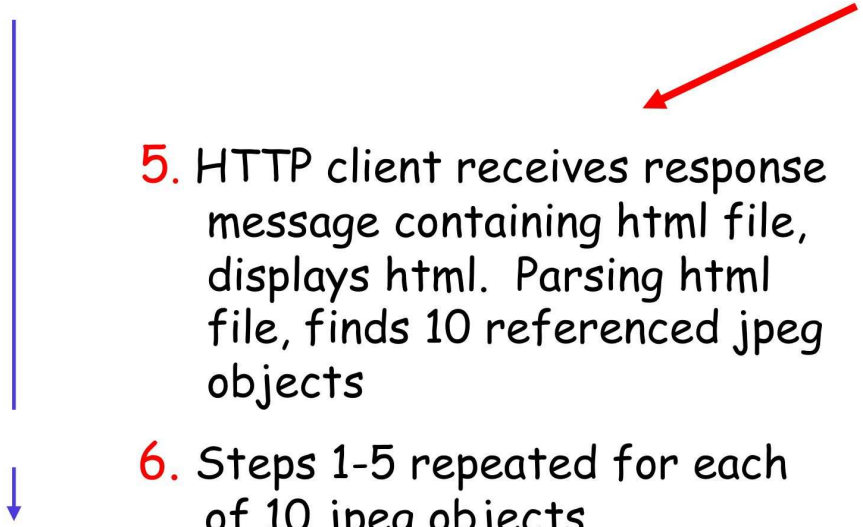
# Non-peristent HTTP example

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

(contains text, references to 10 jpeg images)



# Non-peristent HTTP example



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

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

# Non-persistent vs persistent

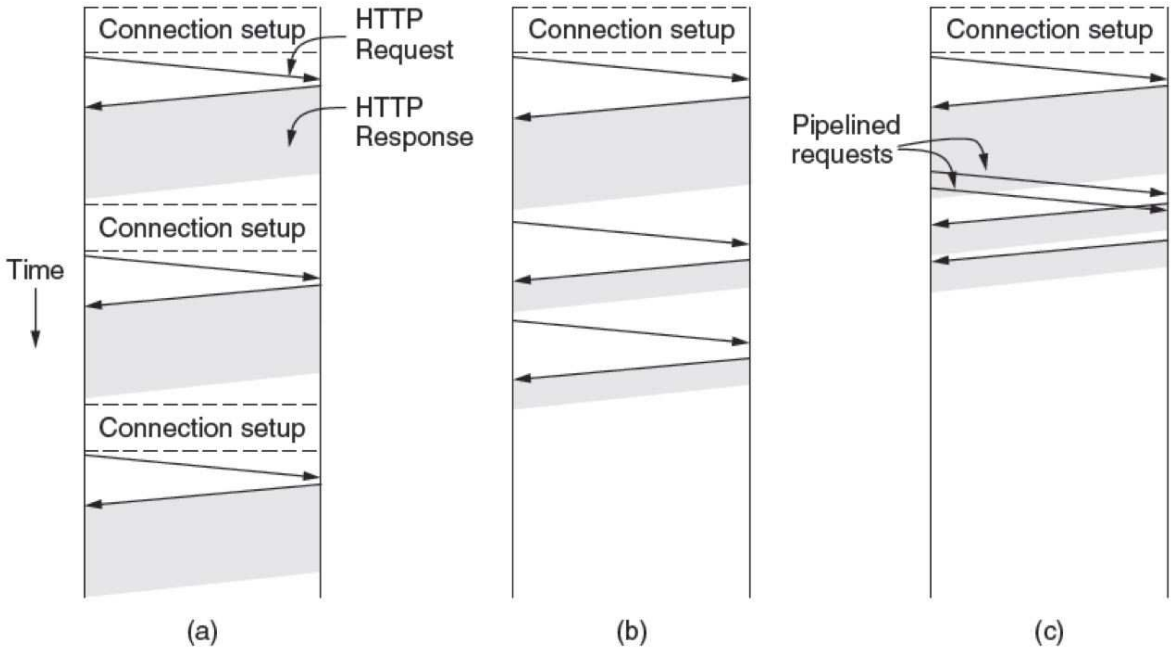
Non-persistent:

- requires 2 "response times" (one to initiate TCP connection and one for initial HTTP request) per object + file transmission time
- OS overhead for each TCP connection
- browsers often open parallel TCP connections to fetch referenced objects

Persistent:

- server leaves connection open after sending response
- subsequent HTTP messages between same client/server sent over open connection
- client sends requests as soon as it encounters a referenced object, reducing overall response time

# HTTP request connection




HTTP with (a) multiple connections and sequential requests. (b) A persistent connection and sequential requests. (c) A persistent connection and pipelined requests.

# HTTP – summary of key steps

Steps that occur when a link is selected:

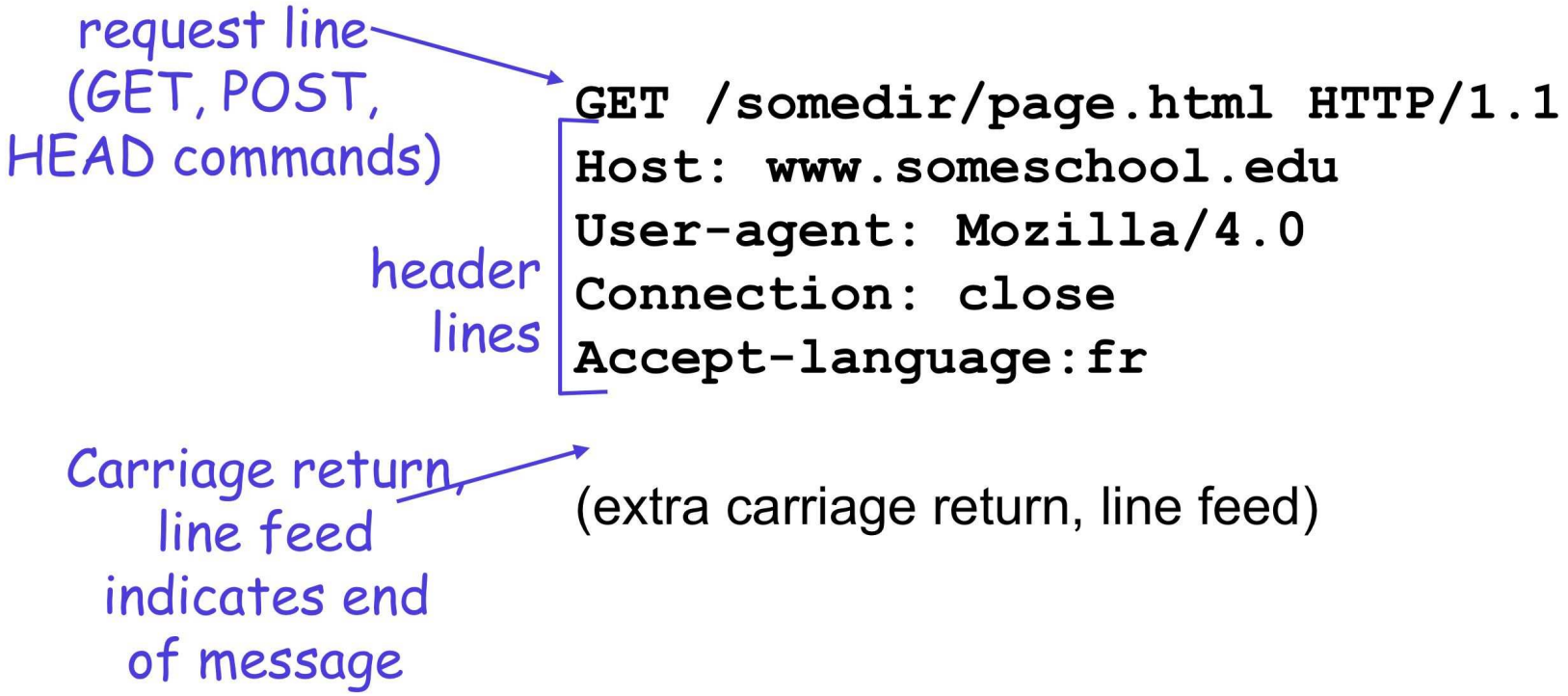
- Browser determines the URL
- Browser asks DNS for the IP address of the server
- DNS replies
- The browser makes a TCP connection
- Sends HTTP request for the page
- Server sends the page as HTTP response
- Browser fetches other URLs as needed
- The browser displays the page
- The TCP connections are released

# HTTP request methods

The built-in HTTP request methods.

| Method | Description |
|---|---|
| GET | Read a Web page |
| HEAD | Read a Web page's header |
| POST | Append to a Web page |
| PUT | Store a Web page |
| DELETE | Remove the Web page |
| TRACE | Echo the incoming request |
| CONNECT | Connect through a proxy |
| OPTIONS | Query options for a page |

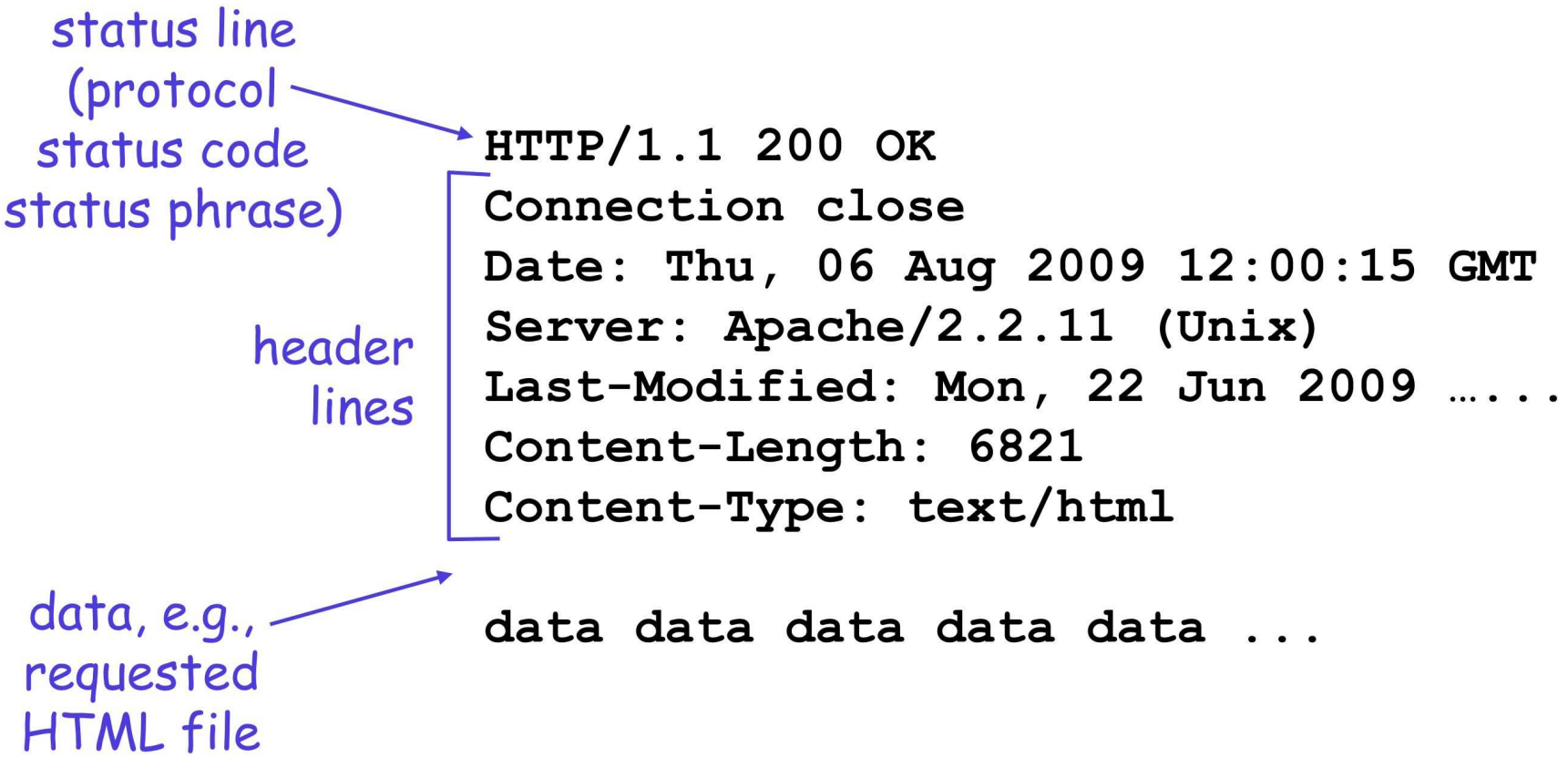
# HTTP request example



# HTTP status code response groups

| Code | Meaning | Examples |
|---|---|---|
| 1xx | Information | 100 = server agrees to handle client's request |
| 2xx | Success | 200 = request succeeded; 204 = no content present |
| 3xx | Redirection | 301 = page moved; 304 = cached page still valid |
| 4xx | Client error | 403 = forbidden page; 404 = page not found |
| 5xx | Server error | 500 = internal server error; 503 = try again later |

# HTTP response example



# HTTP message header

| Header | Type | Contents |
|---|---|---|
| User-Agent | Request | Information about the browser and its platform |
| Accept | Request | The type of pages the client can handle |
| Accept-Charset | Request | The character sets that are acceptable to the client |
| Accept-Encoding | Request | The page encodings the client can handle |
| Accept-Language | Request | The natural languages the client can handle |
| If-Modified-Since | Request | Time and date to check freshness |
| If-None-Match | Request | Previously sent tags to check freshness |
| Host | Request | The server's DNS name |
| Authorization | Request | A list of the client's credentials |
| Referer | Request | The previous URL from which the request came |
| Cookie | Request | Previously set cookie sent back to the server |
| Set-Cookie | Response | Cookie for the client to store |
| Server | Response | Information about the server |

# HTTP message header

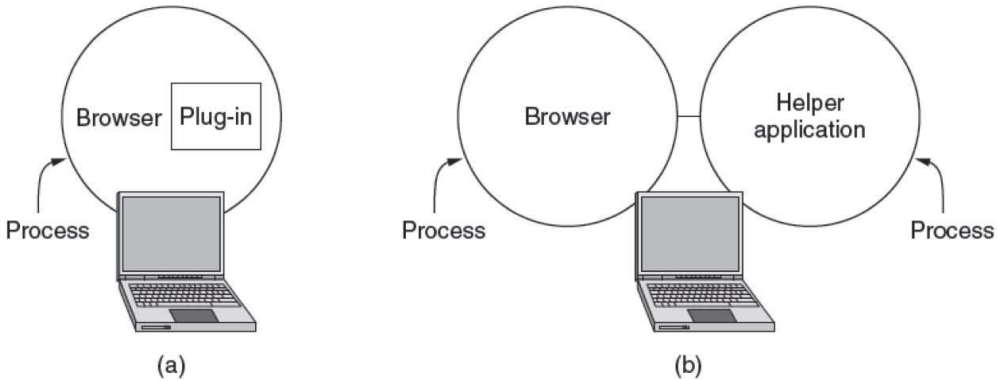
| | | |
|---|---|---|
| Content-Encoding | Response | How the content is encoded (e.g., *gzip*) |
| Content-Language | Response | The natural language used in the page |
| Content-Length | Response | The page's length in bytes |
| Content-Type | Response | The page's MIME type |
| Content-Range | Response | Identifies a portion of the page's content |
| Last-Modified | Response | Time and date the page was last changed |
| Expires | Response | Time and date when the page stops being valid |
| Location | Response | Tells the client where to send its request |
| Accept-Ranges | Response | Indicates the server will accept byte range requests |
| Date | Both | Date and time the message was sent |
| Range | Both | Identifies a portion of a page |
| Cache-Control | Both | Directives for how to treat caches |
| ETag | Both | Tag for the contents of the page |
| Upgrade | Both | The protocol the sender wants to switch to |

# Client side content – plugins and helpers

**Plugins** - integrated software module which executes inside the browser, direct access to online context

**Helper** - separate program which can be instantiated by the browser, but can only access local cache of file content

- application/pdf
- application/msword

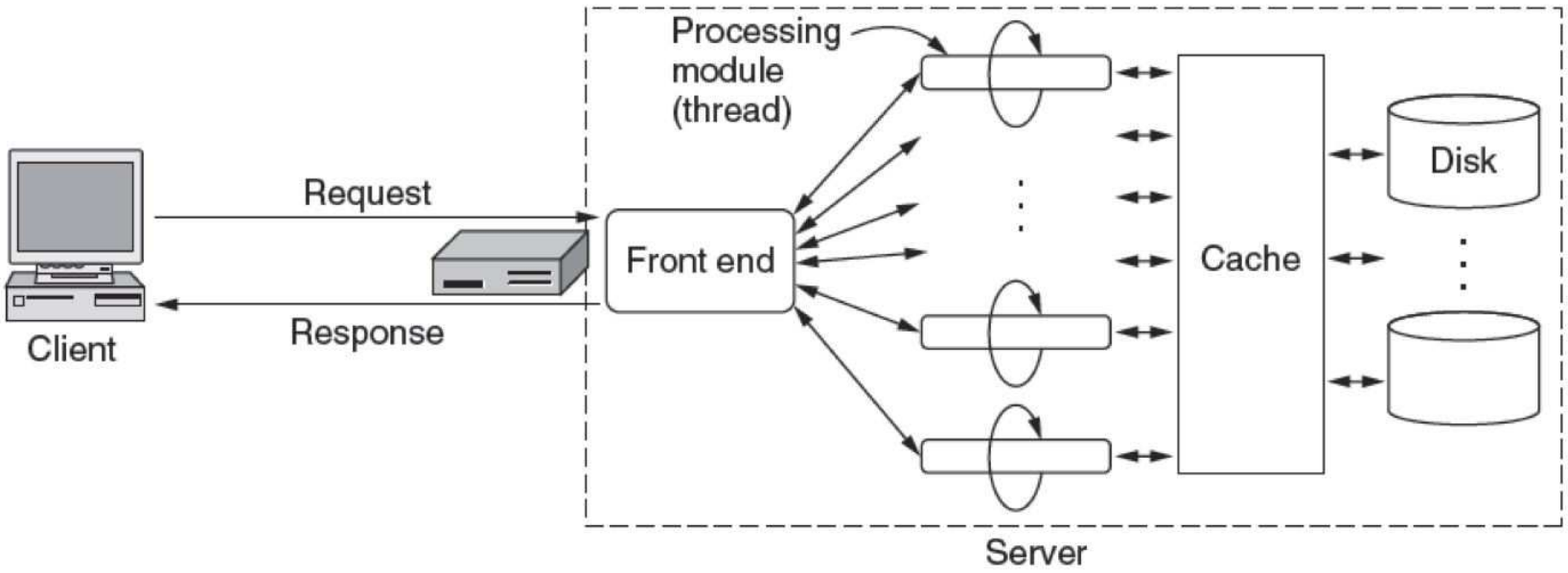


# Server side processing – a summary

5 step process:

- Accept TCP Connection from client (browser)
- Identify the file requested
- Get the specified file from the local disk storage
- Send the file to the client
- Release the TCP connection

# Multithreaded Web server




A multithreaded Web server with a front end and processing modules.

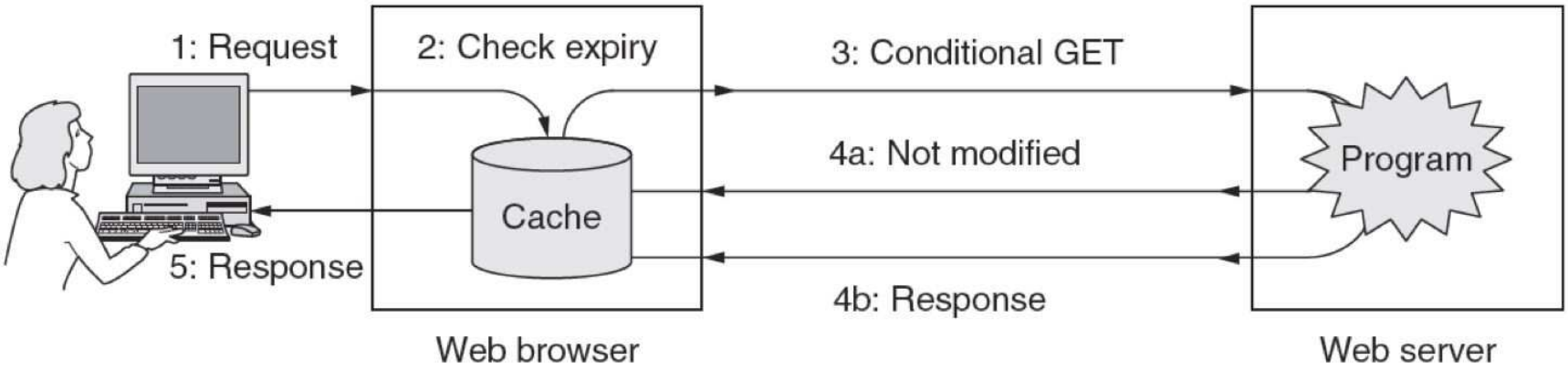
# Multithreaded Web server

A processing module performs a series of steps:

- Resolve name of Web page requested.
- Perform access control on the Web page.
- Check the cache.
- Fetch requested page from disk or run program
- Determine the rest of the response
- Return the response to the client.
- Make an entry in the server log.

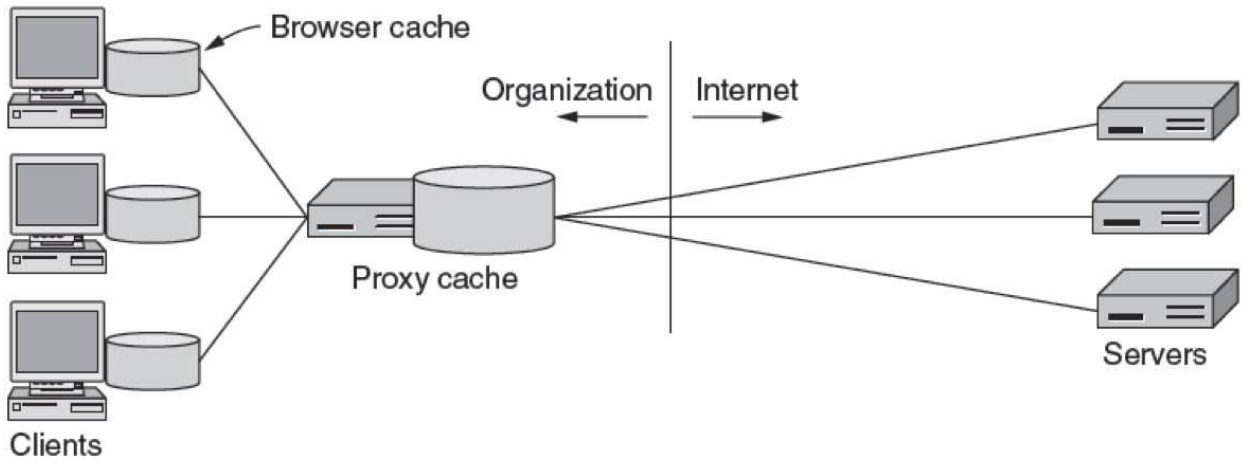
# Web cache

**Goal:** satisfy client request without involving origin server – reduce response time.



# Web proxies

A proxy cache between Web browsers and Web servers.




The browser sends all HTTP requests to cache. The cache returns objects or else the cache requests object from *origin server*, then returns object to client. Note: the cache ( proxy server) acts as both client and server.

# Cookies

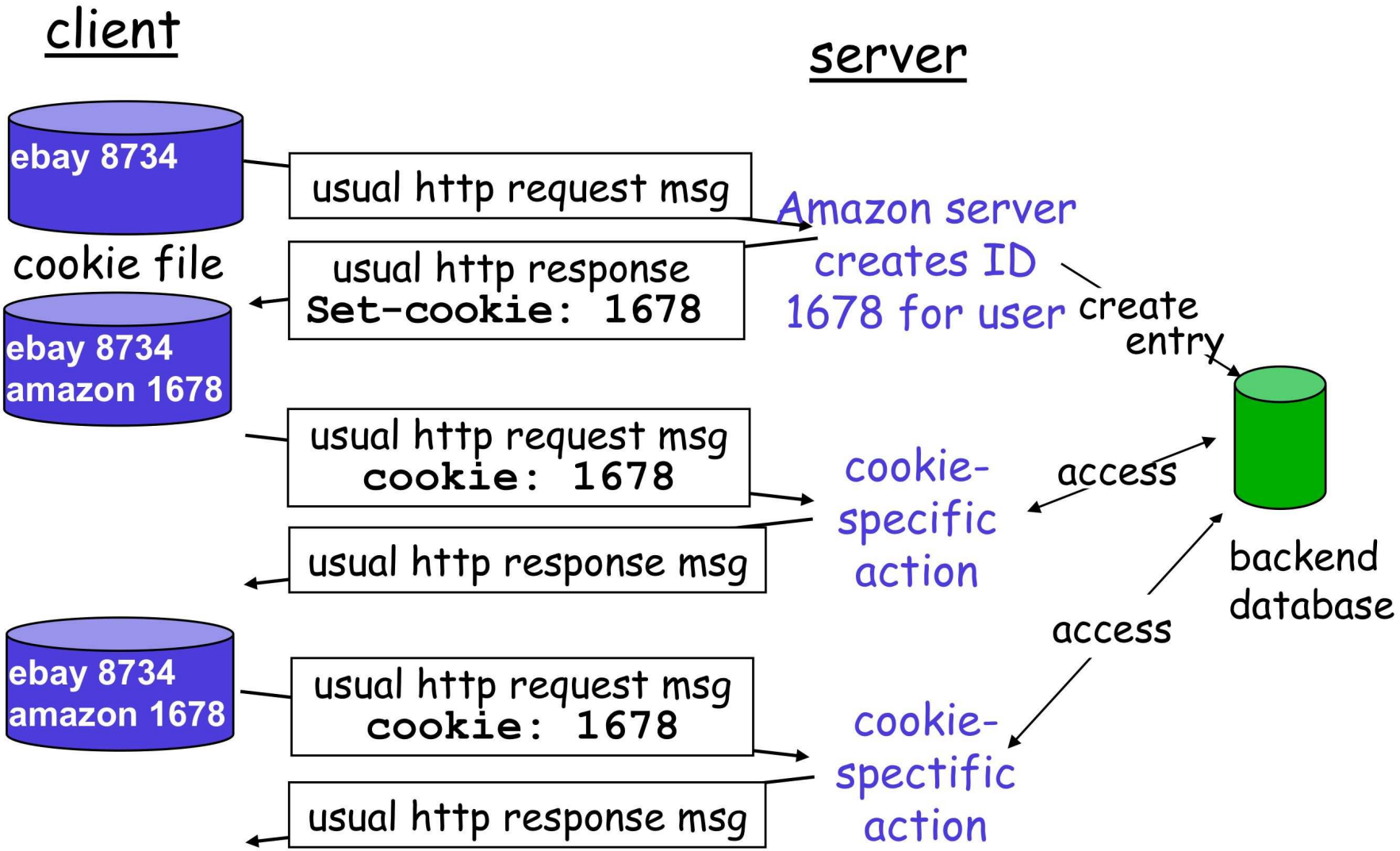
- The Web is basically stateless
- Cookies can place small amount (<4Kb) of information on the users computer and re-use deterministically (RFC 2109)
- Cookies have 5 fields: domain, path, content, expiry, security
- How to keep state – maintain state at sender/receiver over multiple transactions; http messages carry "state"
- Questionable mechanism for tracking users (invisibly perhaps) and learning about user behaviour eg competitor snooping, undesirable content etc

# Cookies

Example cookies:

| Domain | Path | Content | Expires | Secure |
|---|---|---|---|---|
| toms-casino.com | / | CustomerID=297793521 | 15-10-10 17:00 | Yes |
| jills-store.com | / | Cart=1-00501;1-07031;2-13721 | 11-1-11 14:22 | No |
| aportal.com | / | Prefs=Stk:CSCO+ORCL;Spt:Jets | 31-12-20 23:59 | No |
| sneaky.com | / | UserID=4627239101 | 31-12-19 23:59 | No |

# Cookies


client
server
ebay 8734
usual http request msg
Amazon server creates ID 1678 for user
cookie file
usual http response
Set-cookie: 1678
create entry
ebay 8734
amazon 1678
usual http request msg
cookie: 1678
cookie-specific action
access
usual http response msg
backend database
access
ebay 8734
amazon 1678
usual http request msg
cookie: 1678
cookie-spectific action
usual http response msg

# Static web documents

- HTML - Hypertext Markup Language
  - a simple language designed to encode both content and presentational information
  - Plain text encoding, with browser based rendering
  - Restricted to ISO-8859 Latin-1 character set (internationalisation not introduced until XHTML with UTF encodings)
- Web Page Components
  - Structural divisions:
    - Head <head> ... </head>
    - Body <body> ... </body>
  - Syntactically Restricted Tag Sets
  - Attributes & Values <img src="pic1.gif" alt="pic1.gif" >

# Static web documents

| Item | HTML 1.0 | HTML 2.0 | HTML 3.0 | HTML 4.0 | HTML 5.0 |
|---|---|---|---|---|---|
| Hyperlinks | x | x | x | x | x |
| Images | x | x | x | x | x |
| Lists | x | x | x | x | x |
| Active maps & images | | x | x | x | x |
| Forms | | x | x | x | x |
| Equations | | | x | x | x |
| Toolbars | | | x | x | x |
| Tables | | | x | x | x |
| Accessibility features | | | | x | x |
| Object embedding | | | | x | x |
| Style sheets | | | | x | x |
| Scripting | | | | x | x |
| Video and audio | | | | | x |
| Inline vector graphics | | | | | x |
| XML representation | | | | | x |
| Background threads | | | | | x |
| Browser storage | | | | | x |
| Drawing canvas | | | | | x |

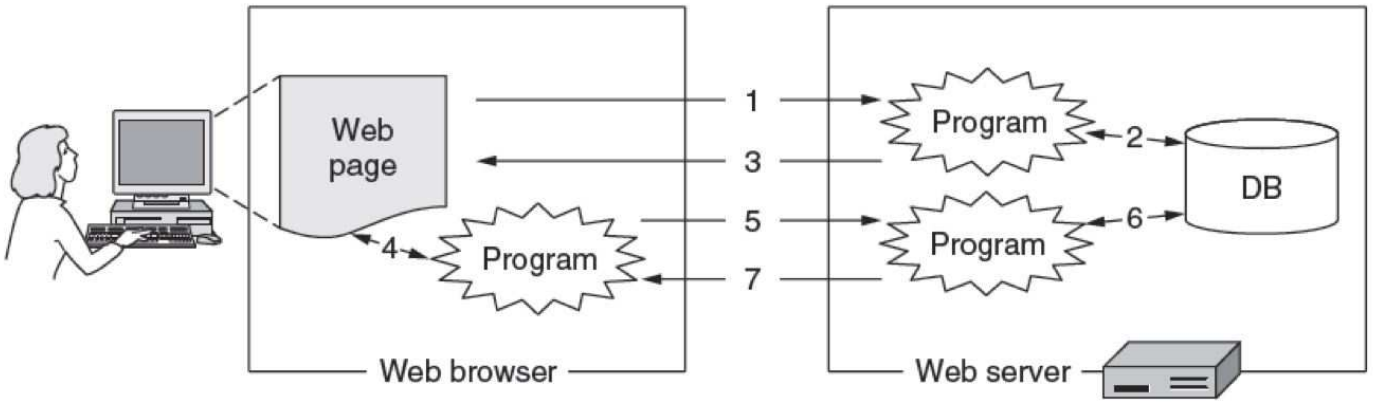
A comparison of HTML version functionality.

# Beyond HTML ...

- XML (Extensible Markup Language) & XSL (Extensible Stylesheet Language)
  - Primary feature of these technologies is the separation of content and presentational markup
  - Stringent validation requirements
- XHTML -
  - Essentially an expression of HTML 4.0 as valid XML
  - Major differences to HTML 4.0 are the requirements for conformance, case folding, well-formedness, attribute specification, nesting and embedding, and inclusion of a document type identifier

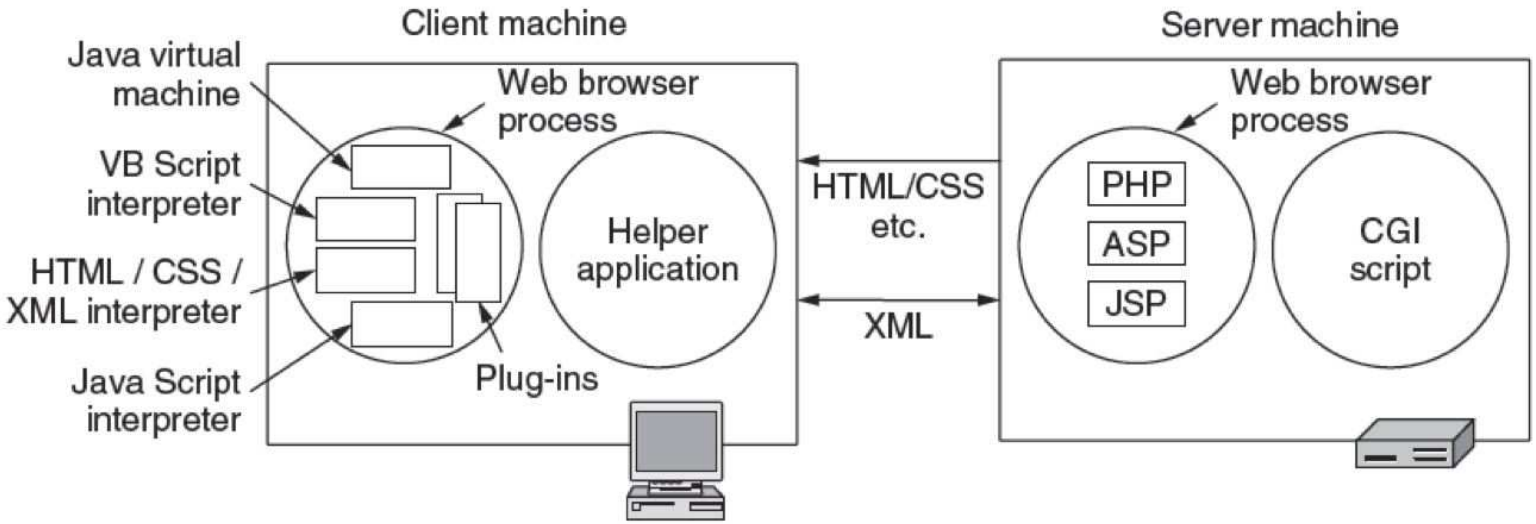
# Dynamic web documents

The need for dynamism in the web document environment has been motivated both by interactivity (user) requirements as well as functionality and scalability (infrastructural) requirements.



# Creating dynamic content

Various technologies used to generate dynamic pages:



# Dynamic web documents

```
<html>
<body>
<form action="action.php" method="post">
<p> Please enter your name: <input type="text" name="name"> </p>
<p> Please enter your age: <input type="text" name="age"> </p>
<input type="submit">
</form>
</body>
</html>
```

(a)

```
<html>
<body>
<h1> Reply: </h1>
Hello <?php echo $name; ?>.
Prediction: next year you will be <?php echo $age + 1; ?>
</body>
</html>
```

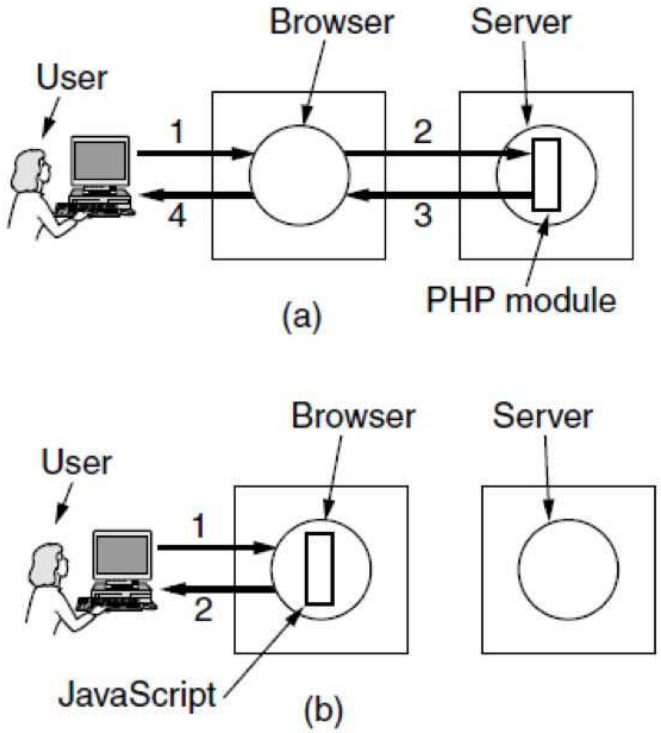
(b)

```
<html>
<body>
<h1> Reply: </h1>
Hello Barbara.
Prediction: next year you will be 33
</body>
</html>
```

(c)

(a) A Web page containing a form. (b) A PHP script for handling the output of the form. (c) Output from the PHP script when the inputs are "Barbara" and "32", respectively.

# Dynamic web documents




(a) Server-side scripting with PHP.
(b) Client-side scripting withJavaScript.

# Dynamic processing

Technologies for producing interactive web applications include:

- JavaScript
- Java Applets - compiled Java code (platform independent)
- ActiveX - compiled code for Windows
- AJAX
  - HTML and CSS: present information as pages.
  - DOM: change parts of pages while they are viewed.
  - XML: let programs exchange data with the server.
  - An asynchronous way to send and retrieve XML data.
  - JavaScript as a language to bind all this together

# VoIP technologies

Within the VoIP domain, there are 3 distinct models for service provision:

- infrastructural - PSTN/PABX integration
- virtual - media gateways, virtual directories
- value-added - voice mail

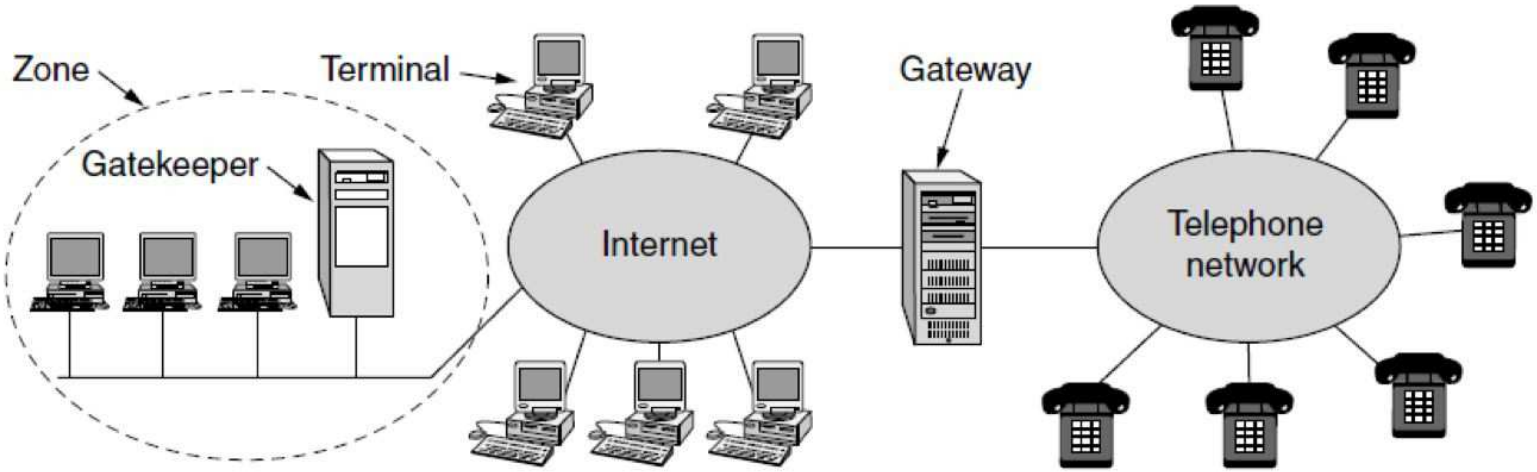
Alternative VoIP technologies:

- H.323
- SIP

# H.323

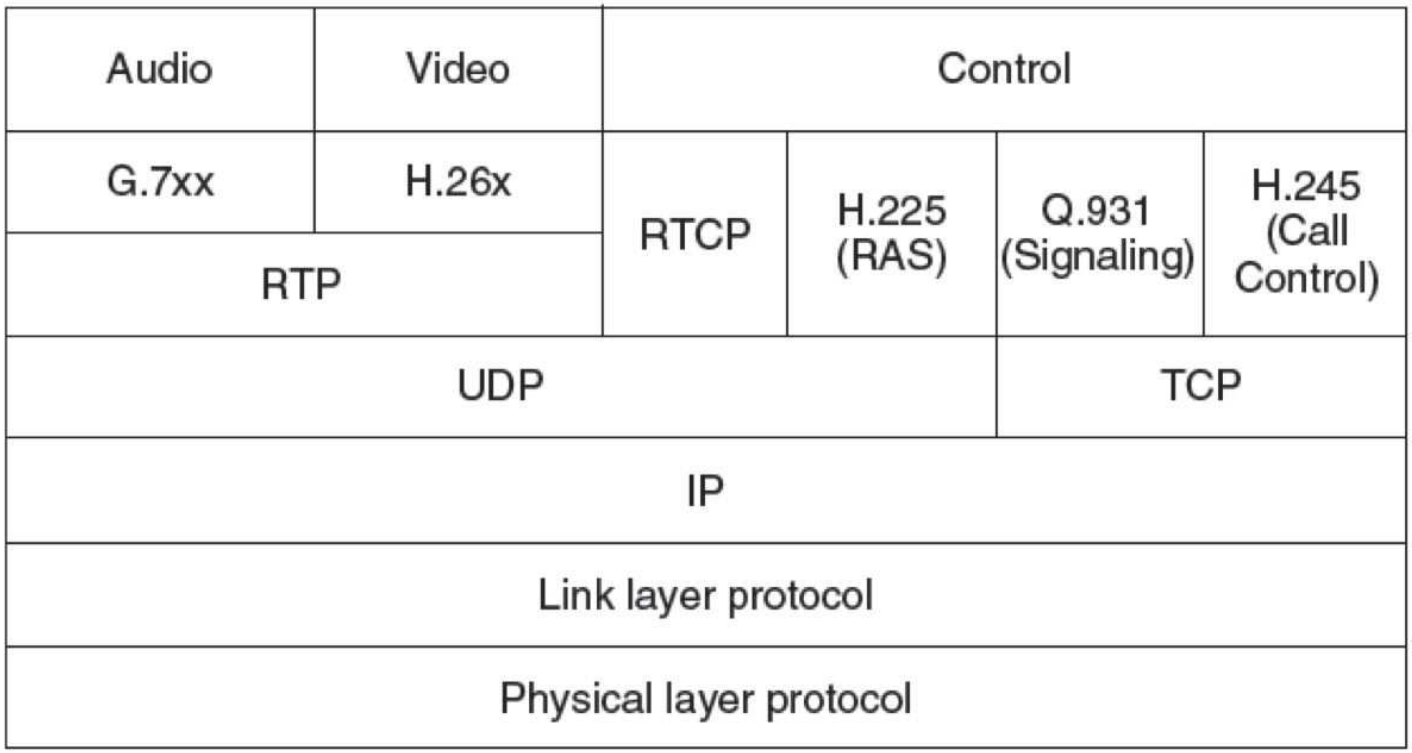
- H.323 is an international standard that specifies how multimedia traffic is carried over packet networks
- H.323 integrates existing standards to provide a layered protocol stack
- H.323 was originally developed to enable multimedia applications to run over *unreliable data networks* - includes support for audio (including VoIP), video and data sharing services within a single protocol stack

# H.323 architectural view

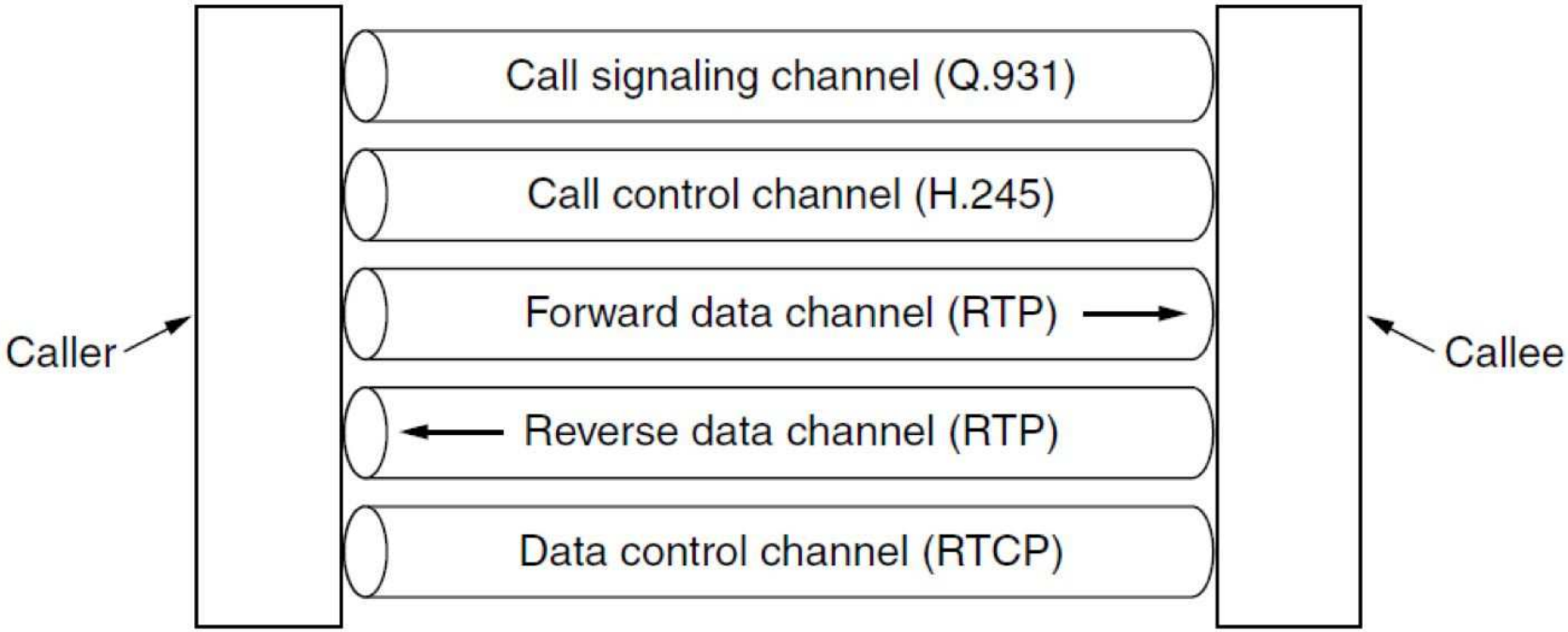



The H.323 architectural model for Internet telephony.

# H.323 protocol stack



# Logic channels in H.323 VoIP




Logical channels between the caller and callee during a call.

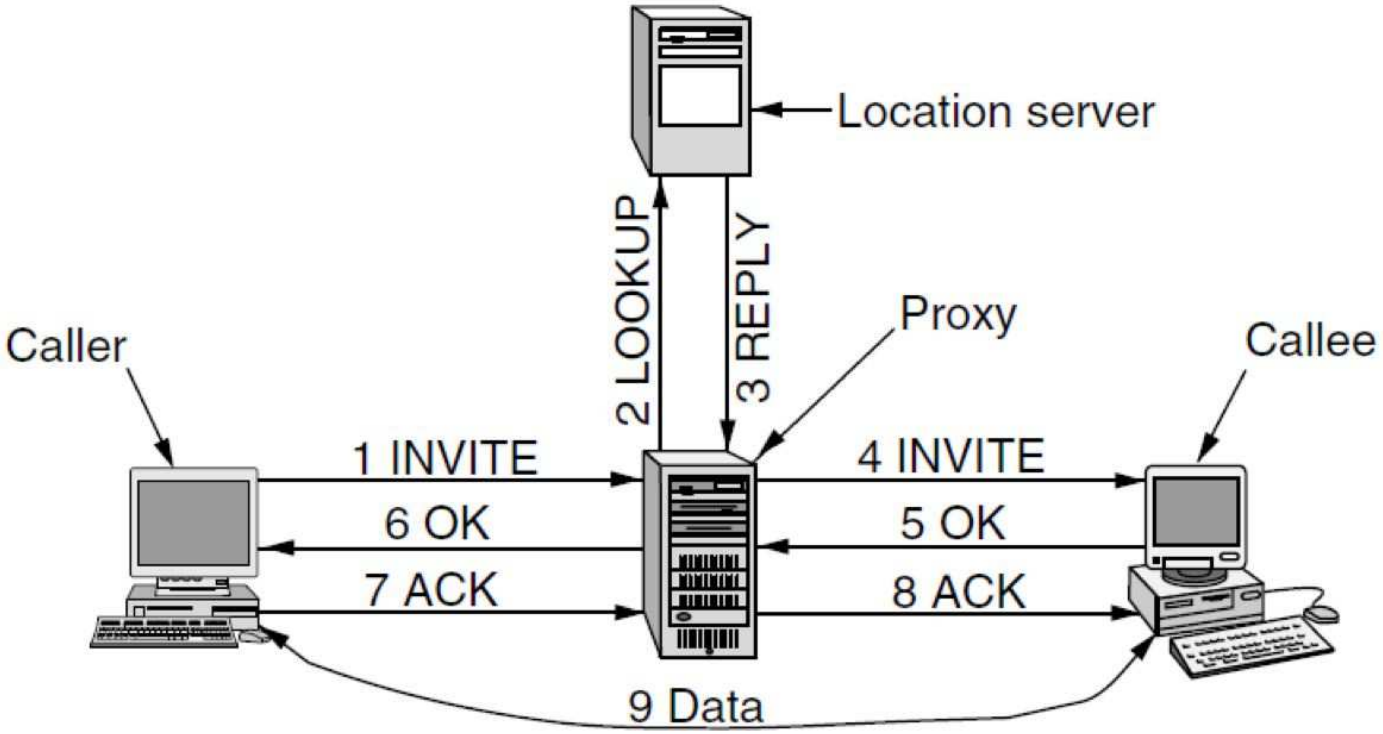
# SIP – Session Initiation Protocol

Modelled on HTTP, simple ASCII based protocol, method + parameters, similar headers to MIME

- SIP Architecture
  - a single network module rather than a complete protocol suite
  - SIP only handles the setup, management and termination of sessions - requires other protocols such as RTP/RTCP for data transport. SIP is an application layer protocol and can run over UDP or TCP
- SIP Functionality
  - two party, multiparty and multicast
  - callee location, callee capabilities, call setup, call termination
- SIP Addressing: addressing based on a URL type schema
  - sip:mkirley@estragon.cs.mu.oz.au
  - SIP URLs can contain IPv4 or IPv6 addresses or actual telephone numbers

# SIP

| Method | Description |
|---|---|
| INVITE | Request initiation of a session |
| ACK | Confirm that a session has been initiated |
| BYE | Request termination of a session |
| OPTIONS | Query a host about its capabilities |
| CANCEL | Cancel a pending request |
| REGISTER | Inform a redirection server about the user's current location |




Use of a proxy server and redirection with SIP.