

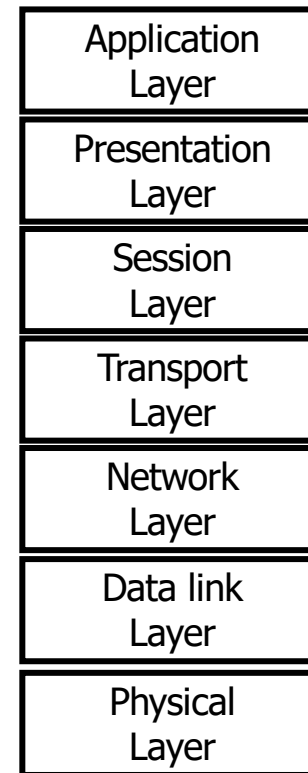
Computer Networks

Application Layer

Chapter

1. Introduction
2. Protocols
3. **Application layer**
 - Client-server-paradigm
 - Domain Name Service
 - Hypertext Transfer Protocol
 - Email
4. Web services
5. Distributed hash tables
6. Time synchronization
7. Transport layer
8. UDP and TCP
9. TCP performance
10. Network layer
11. Internet protocol
12. Data link layer

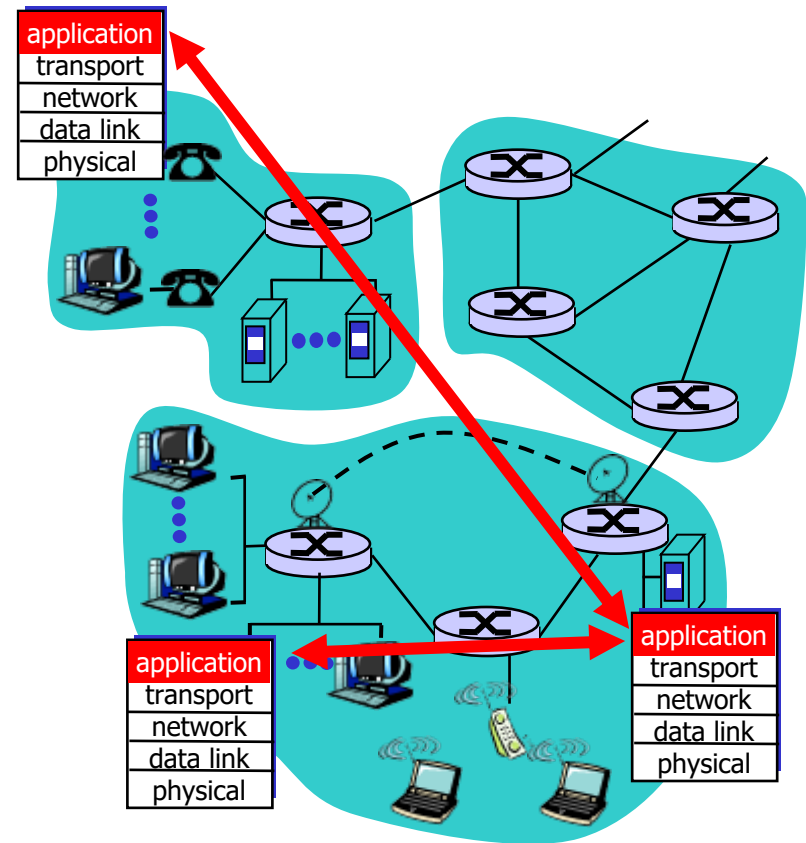
Top-Down-Approach



Application layer

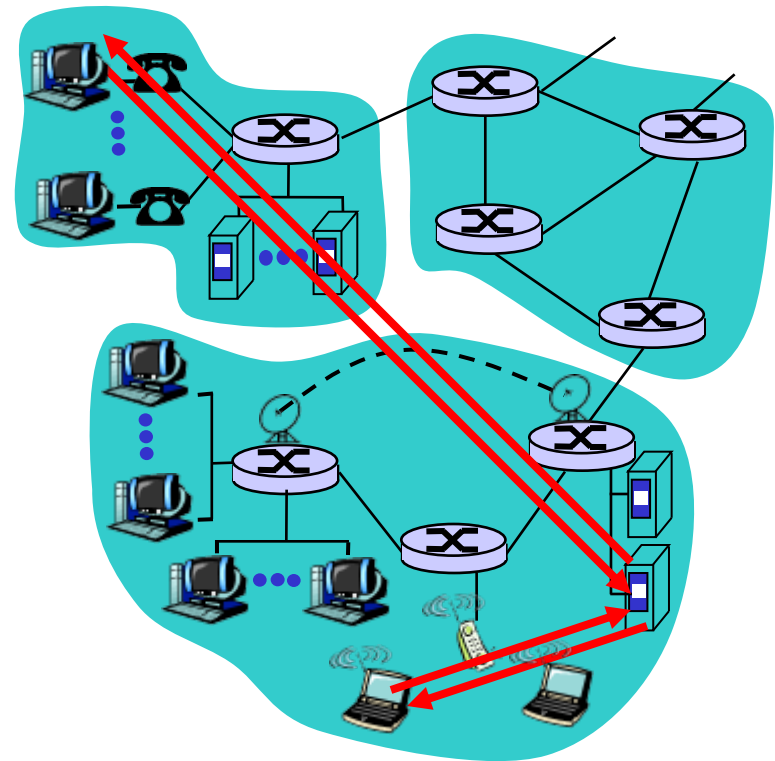
Introduction

- **Application processes** running on end systems (hosts) communicate via message exchange over a network
- Implemented using **services of the transport layer**
- **Application layer protocol** defines messages and procedures
- E.g., web browser and web server
- “lower” layers do not need to understand application logic



Client-server-paradigm

- Server provides services to potential clients
- Most common paradigm for internet applications
- Typical properties of servers
 - Powerful
 - Always-on
- Typical properties of clients
 - Not always connected
 - Communication with server but not (directly) with other clients



Client-server-paradigm

■ Client-server is a centralized architecture

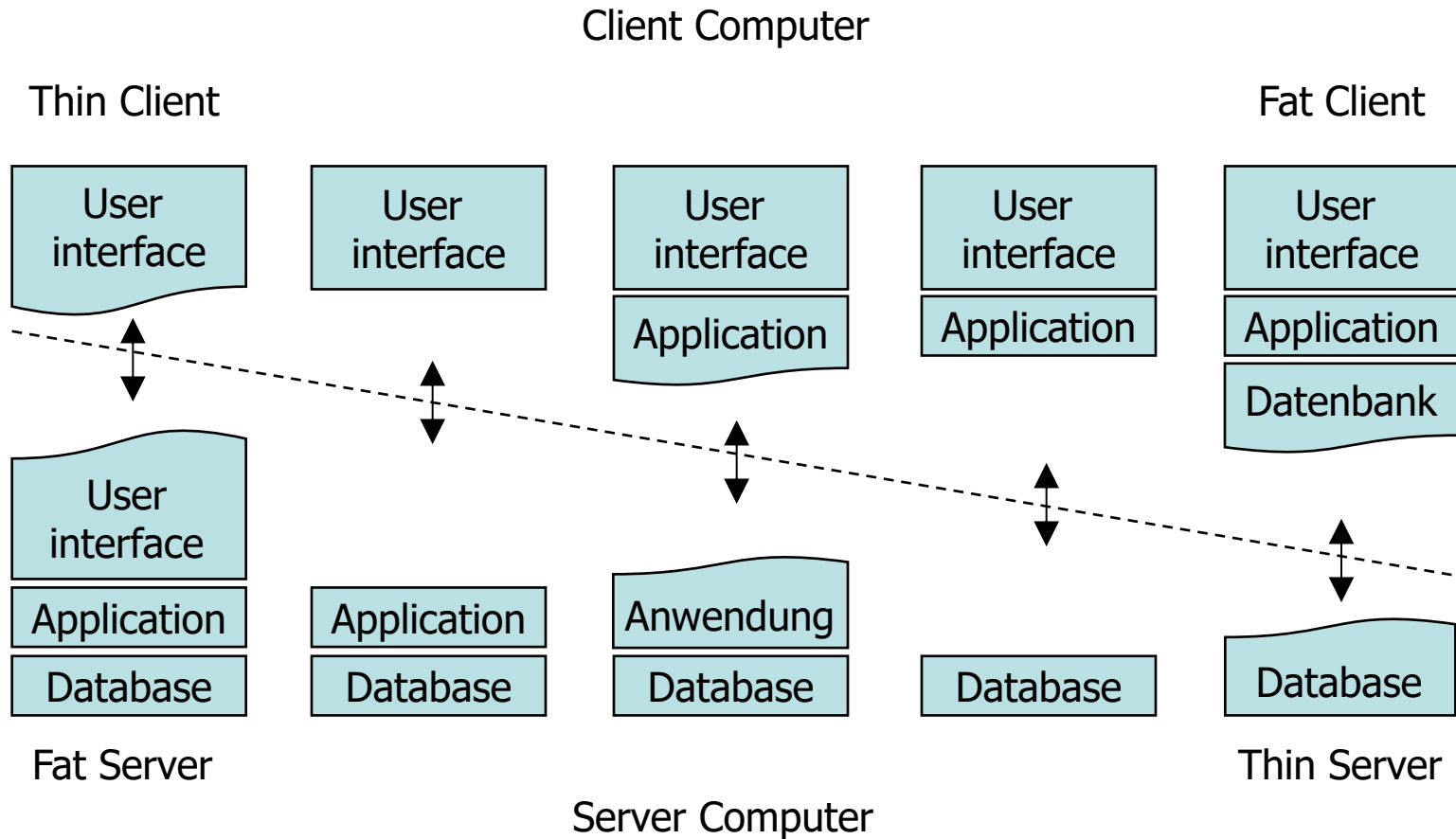
- **Role of client and server may change**; hosts may take one of the roles depending on state (e.g., web caching or email via SMTP)

■ Alternatives

- **Distributed architecture** multiple independent applications can be combined to a single solution (from user perspective it appears as a single monolithic application); coordination is distributed but single point of access (e.g., web shop with application server and database)
- Peer-to-peer systems represent a **fully distributed architecture**; autonomous self-organizing systems without centralized control
- **Hybrid architectures**: initialization using a centralized client-server system and then peer-to-peer-based data exchange (e.g., internet telephony)

Client-server-paradigm

■ Variants



Applications use transport layer

- Transport layer services
 - Dominating transport layer protocols in the internet
 - **TCP**: connection-oriented (realized a byte stream between end systems), reliable
 - **UDP**: connectionless (transport of independent datagrams), unreliable
 - Typically implemented in operating system
 - **Socket interface**

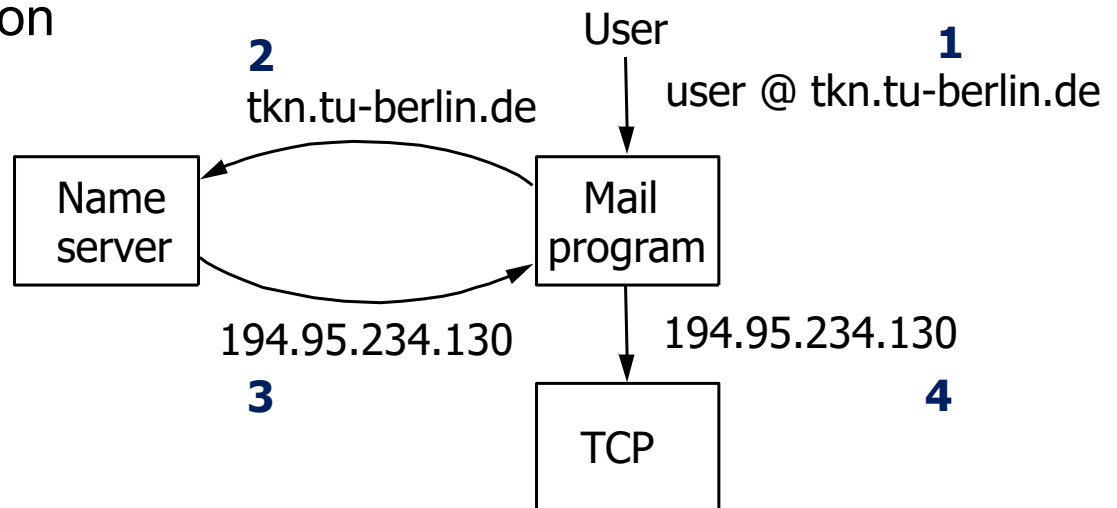
Applications and used transport protocols

Application	Application layer protocol	Transport layer protocol
Email	SMTP [RFC 2821]	TCP
Remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
File transfer	FTP [RFC 959]	TCP
Remote file server	NFS [McKusik 1996]	UDP or TCP
Multimedia streaming	RTP [RFC 1889, 3550]	UDP or TCP
Internet telephony	RTP [RFC 1889, 3550]	usually UDP

Domain Name Service (DNS)

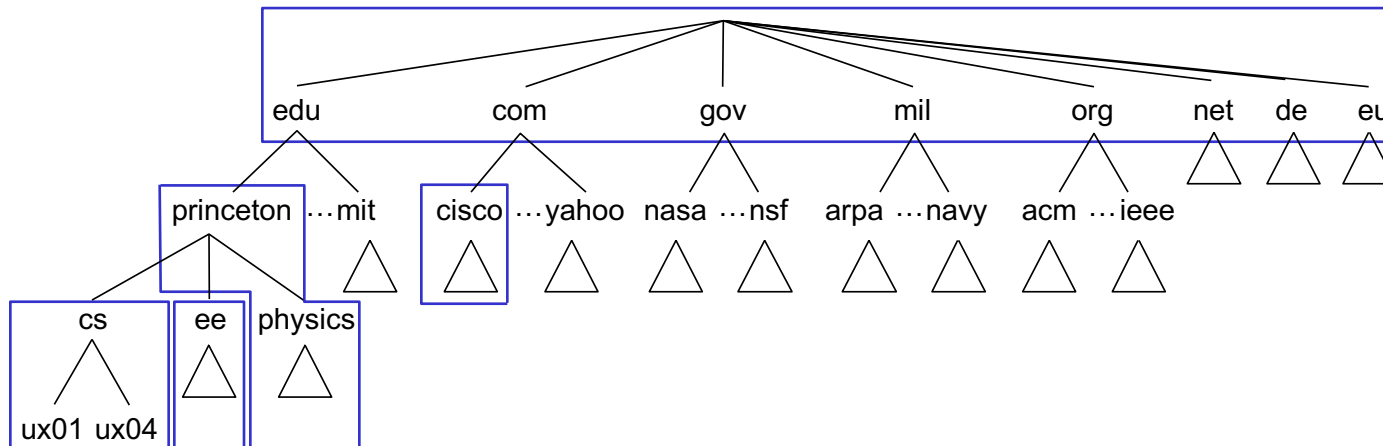
Domain Name Service (DNS)

- Addresses on application layer should be human readable
- On lower layers, IP addresses matter
- DNS translates both host and domain names to IP addresses (and vice versa)
- DNS is realized as a distributed database (many DNS servers)
- Example: address resolution for sending an email



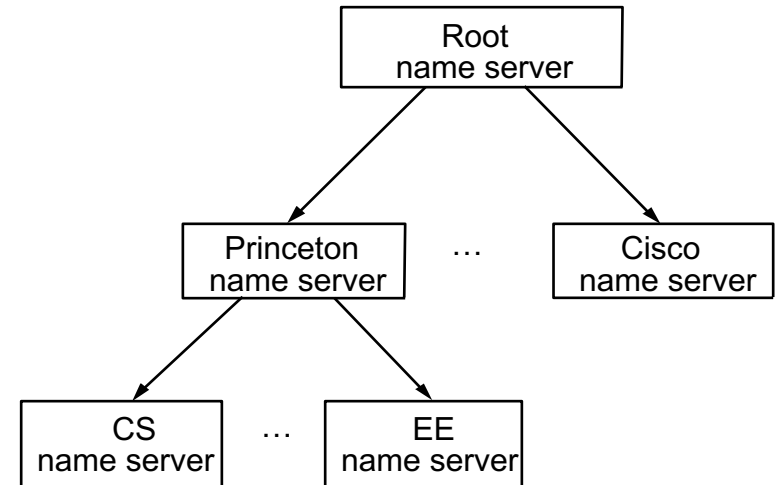
■ Domain structure

- DNS implements hierarchical name space for internet objects
- Resulting names are read left to right but processed right to left
- Every **zone** is managed by a name server (and some backup servers)
- Combination of zones implements hierarchy



■ Name server hierarchy

- **Root name server**
a few
- **Top level domain server**
für com, org, net, edu, uk, de, eu, ...
- ...
- **authoritative name server**
lowest level, used by single organization



Resource Records

■ Resource Records

- Record as stored by name server (domain name, value, type, TTL)
- TTL: Time to live
- Type = A
 - Value = IP address
 - Ex: (www2.tkn.tu-berlin.de, 130.149.110.75, A, TTL)
- Type = NS
 - Value = domain name of the host providing a name service
 - Ex: (tkn.tu-berlin.de, ns.tu-berlin.de, NS, TTL)
- Type = CNAME (canonical name)
 - Value = canonical name of a host, alias name
 - Ex: (cic.cs.princeton.edu, cicada.cs.princeton.edu, CNAME, TTL)
- Type = MX (mail exchange)
 - Value = domain name of the email server
 - Ex: (tkn.tu-berlin.de, b1861.mx.srv.dfn.de, MX, TTL)

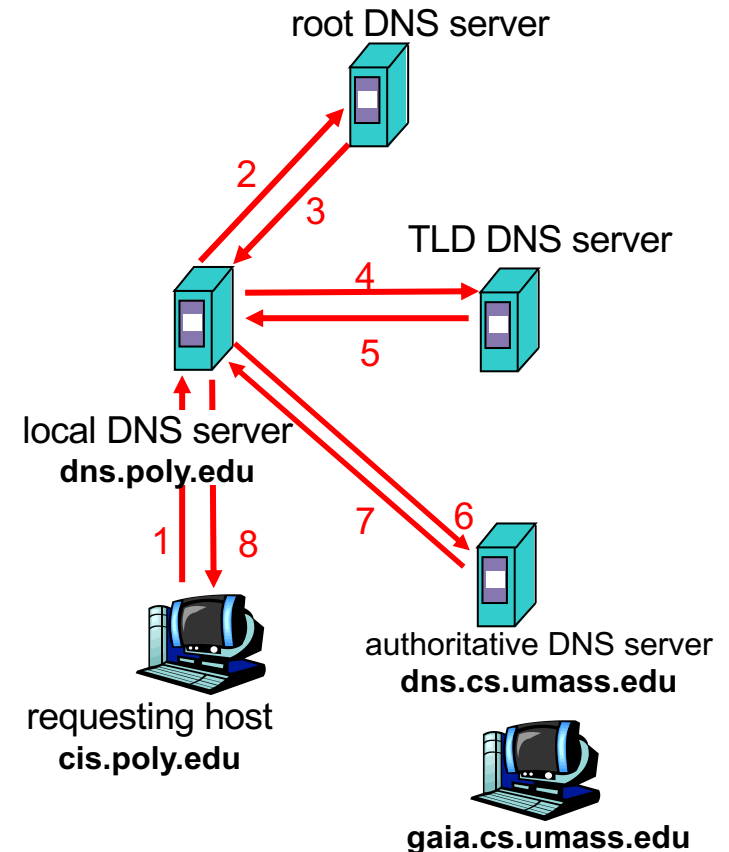
Autoritative/Local Name Server

- When a host makes a DNS Query, query is sent to its local DNS server.
- If this server is not able to resolve, query starts walking up the hierarchy, until somebody can point out the Authoritative server for the queried domain...
- Afterwards the query „walks down“ until the proper sub-domain is found
- Query “walks” its way up and down the hierarchy
 - Iterated query – I don’t know, but here’s who to ask next
 - Recursive query – I don’t know right now, but I’ll get back to you

DNS Name Resolution

- Example: host at *cis.poly.edu* wants IP address for *gaia.cs.umass.edu*
- **Iterative query:**
 - Contacted server replies with name of server to contact
 - “I don’t know this name, but ask this server”

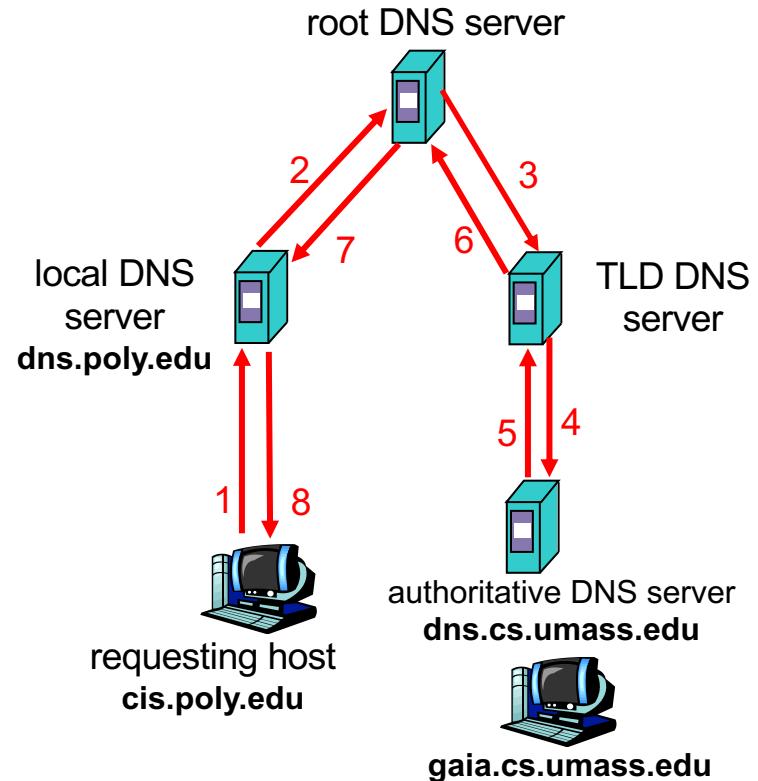
TLD = Top-Level-Domain



Alternatively: Recursive Query Resolution

■ Recursive query:

- Puts burden of name resolution on contacted name server
- Heavy load?



DNS Caching

- Performing all these queries takes time
 - And all this before actual communication takes place
 - E.g., 1-second **latency** before starting Web download
- **Caching** can greatly reduce overhead
 - The top-level servers very rarely change
 - Popular sites (e.g., www.cnn.com) visited often
 - Local DNS server often has the information cached
- How DNS caching works
 - DNS servers cache responses to queries
 - Responses include a “**time to live**” (TTL) field
 - Server deletes cached entry after TTL expires

DNS and Virtual IP Addresses

- DNS records don't have to store the real IP address of host
- Multiple names can map onto the same address
 - E.g.: `www.berkeley.edu` and `arachne.berkeley.edu` map to the same machine (i.e., the same IP address)
 - E.g.: All hosts in the `acme.com` may have the same IP address – in reality a gatekeeper
 - The Gatekeeper decides whether to “admit” a transport level connection to the host `x.acme.com` (firewall functionality)
 - The Gatekeeper decides to forward the connection to one of several identical servers (load balancer functionality)

Hypertext Transfer Protocol (HTTP)

World Wide Web (WWW)

■ WWW

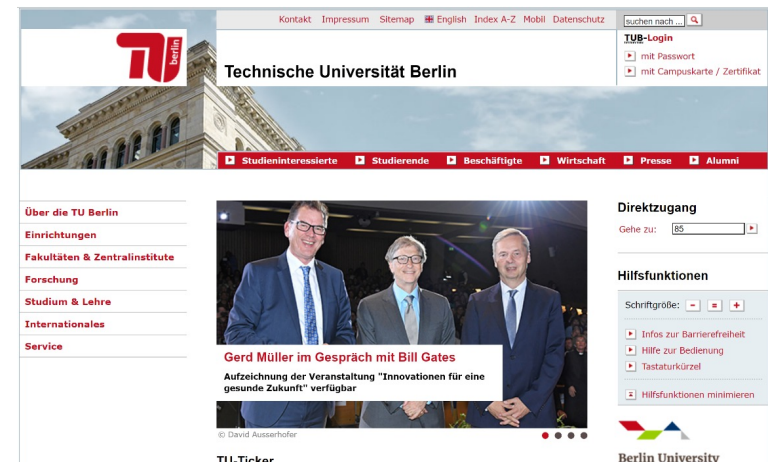
- We use it,
- Do we understand it?

- The notion of a hyperlink is in fact very old
- The technical implementation: information system instead of books- makes the difference!

WELCOME TO THE UNIVERSITY OF EAST PODUNK'S WWW HOME PAGE

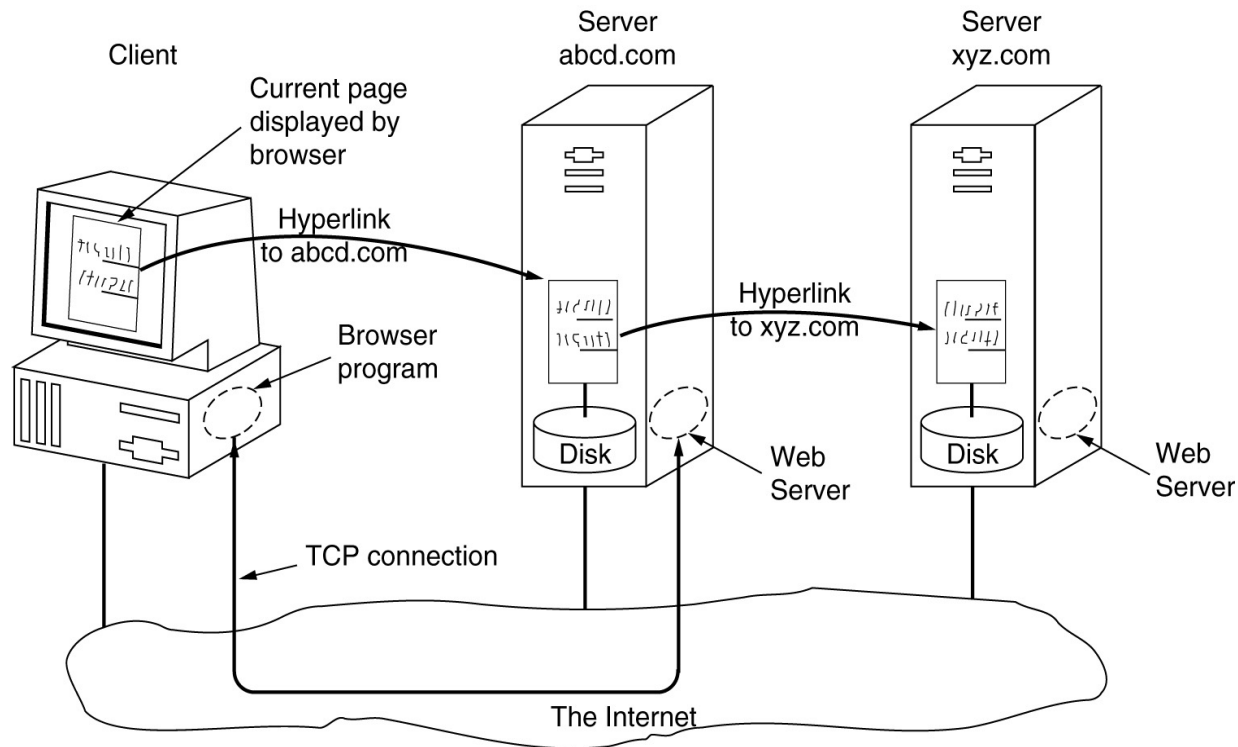
- Campus Information
 - [Admissions information](#)
 - [Campus map](#)
 - [Directions to campus](#)
 - [The UEP student body](#)
- Academic Departments
 - [Department of Animal Psychology](#)
 - [Department of Alternative Studies](#)
 - [Department of Microbiotic Cooking](#)
 - [Department of Nontraditional Studies](#)
 - [Department of Traditional Studies](#)

Webmaster@eastpodunk.edu



The screenshot shows the official website of the Technische Universität Berlin. At the top, there is a navigation bar with links for Kontakt, Impressum, Sitemap, English, Index A-Z, Mobil, and Datenschutz. A search bar is also present. Below the navigation bar is a large banner image of the university's main building. To the right of the banner is a login section for TUB-Login with options for password and campus card. Below the banner is a horizontal menu with links for Studieninteressierte, Studierende, Beschäftigte, Wirtschaft, Presse, and Alumni. On the left side, there is a vertical menu with links for Über die TU Berlin, Einrichtungen, Fakultäten & Zentralinstitute, Forschung, Studium & Lehre, Internationales, and Service. The main content area features a news item titled 'Gerd Müller im Gespräch mit Bill Gates' with a photograph of the two men. To the right of the news item is a 'Direktzugang' section with a search bar and a 'Hilfsfunktionen' section with links for accessibility and keyboard shortcuts. The footer includes the TU-Ticker and the Berlin University logo.

Architectural Overview

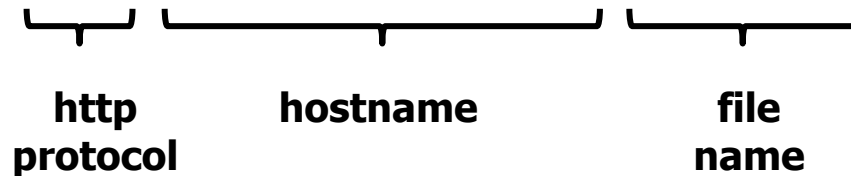


[Tanenbaum, op. cit.]

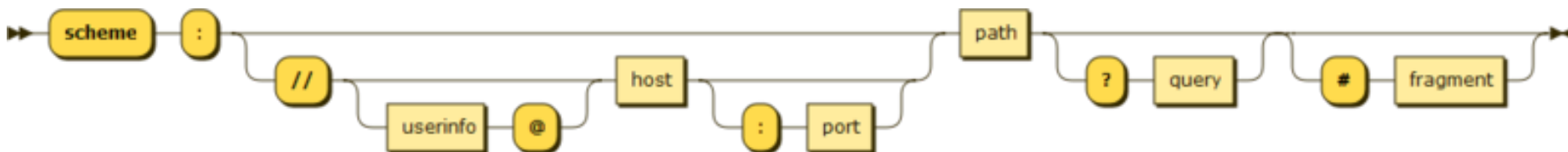
- A distributed database of URLs (Uniform Resource Locators)
- Core components:
 - Servers which store “web pages” and execute remote commands
 - Browsers retrieve and display “pages” of content linked by hypertext
 - Each link is a URL
 - Can build arbitrarily complex applications, all of which share a uniform client application!
- Need a language to define the objects and the layout
 - HTML, XML
- Need a protocol to transfer information between clients and servers
 - **HTTP**

Uniform Resource Locator (URL)

- Defines (roughly) the access method and the path
- protocol://host-name:port/directory-path/resource
- E.g.: http://www.example.com/index.html



- More precise: URL is a specific type of Uniform Resource Identifier (URI):



HTML – HyperText Markup Language

- (a) The HTML code for a sample Web page.
- (b) The formatted (rendered) page.

```
<html>
<head><title> AMALGAMATED WIDGET, INC. </title> </head>
<body> <h1> Welcome to AWI's Home Page</h1>
 <br>
We are so happy that you have chosen to visit <b> Amalgamated Widget's </b>
home page. We hope <i> you </i> will find all the information you need here.
<p>Below we have links to information about our many fine products.
You can order electronically (by WWW), by telephone, or by fax. </p>
<hr>
<h2> Product information </h2>
<ul>
  <li> <a href="http://widget.com/products/big"> Big widgets</a>
  <li> <a href="http://widget.com/products/little"> Little widgets </a>
</ul>
<h2> Telephone numbers</h2>
<ul>
  <li> By telephone: 1-800-WIDGETS
  <li> By fax: 1-415-765-4321
</ul>
</body>
</html>
```

(a)

Welcome to AWI's Home Page



We are so happy that you have chosen to visit **Amalgamated Widget's** home page. We hope you will find all the information you need here.

Below we have links to information about our many fine products. You can order electronically (by WWW), by telephone, or by FAX.

Product Information

- [Big widgets](#)
- [Little widgets](#)

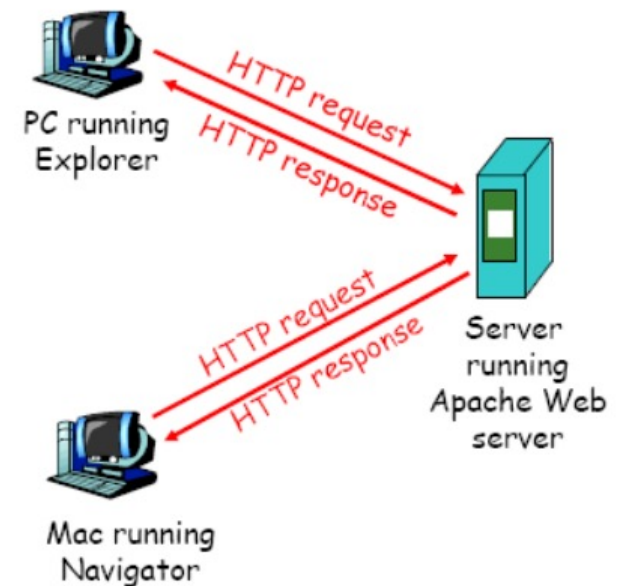
Telephone numbers

- 1-800-WIDGETS
- 1-415-765-4321

(b)

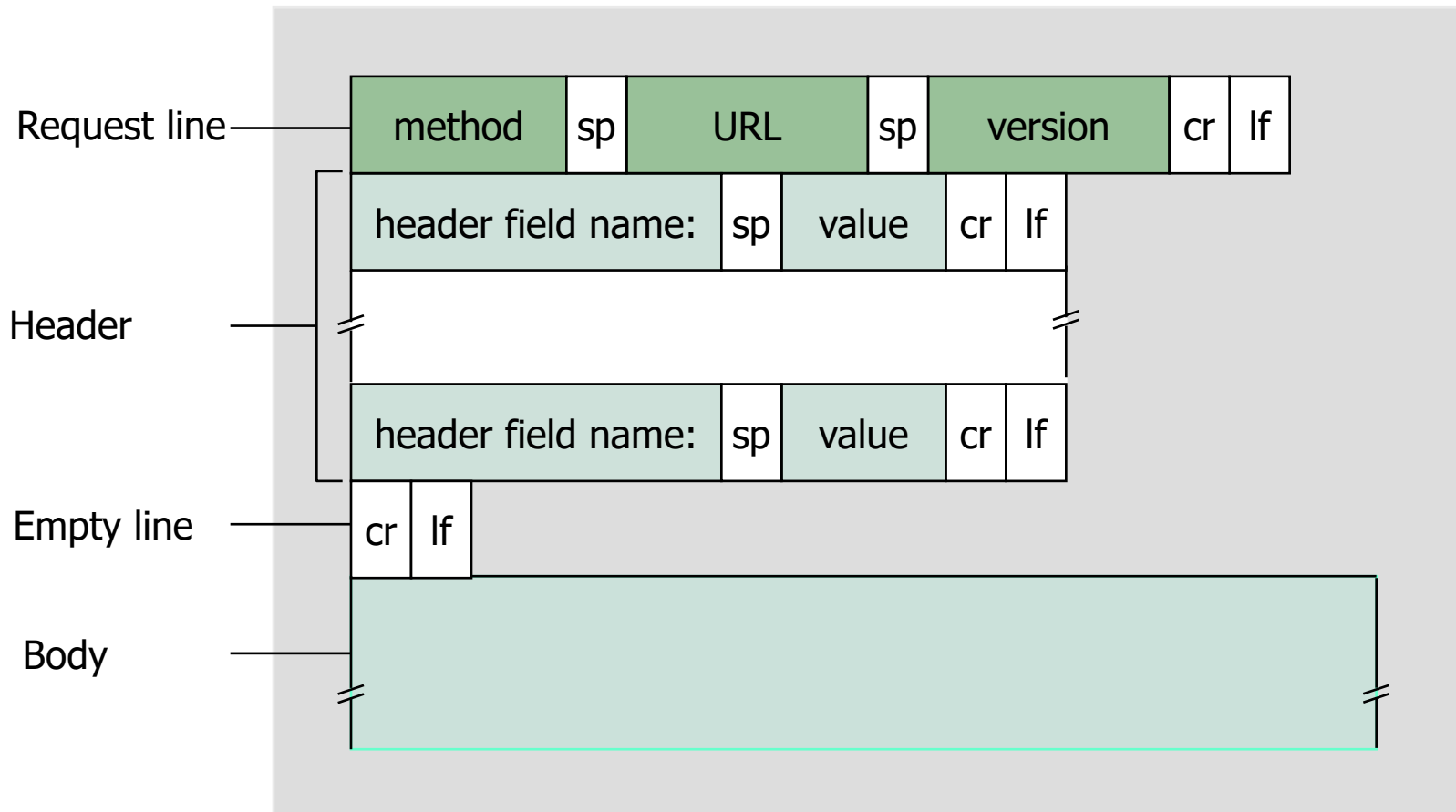
Hypertext Transfer Protocol (HTTP)

- Web's application layer protocol
- Client-server model
 - Client: browser that requests, receives, “displays” Web objects
 - Server: Web server sends objects in response to requests
- HTTP standards:
 - 1.0: RFC 1945 (1996)
 - 1.1: RFC 2068 (1999)
 - 2.0: RFC 7540 (2015)



HTTP Request

■ Format



HTTP Request

■ Methods

- **GET**: request of object, requires method, URL, version
- **HEAD**: request of meta information of an object
- **POST**: push of an object to server
- Put, Delete, Trace, Options

■ Header

- Type-value-pairs, types: Host, User-agent, ...

■ Body

- Empty for GET, object data for POST

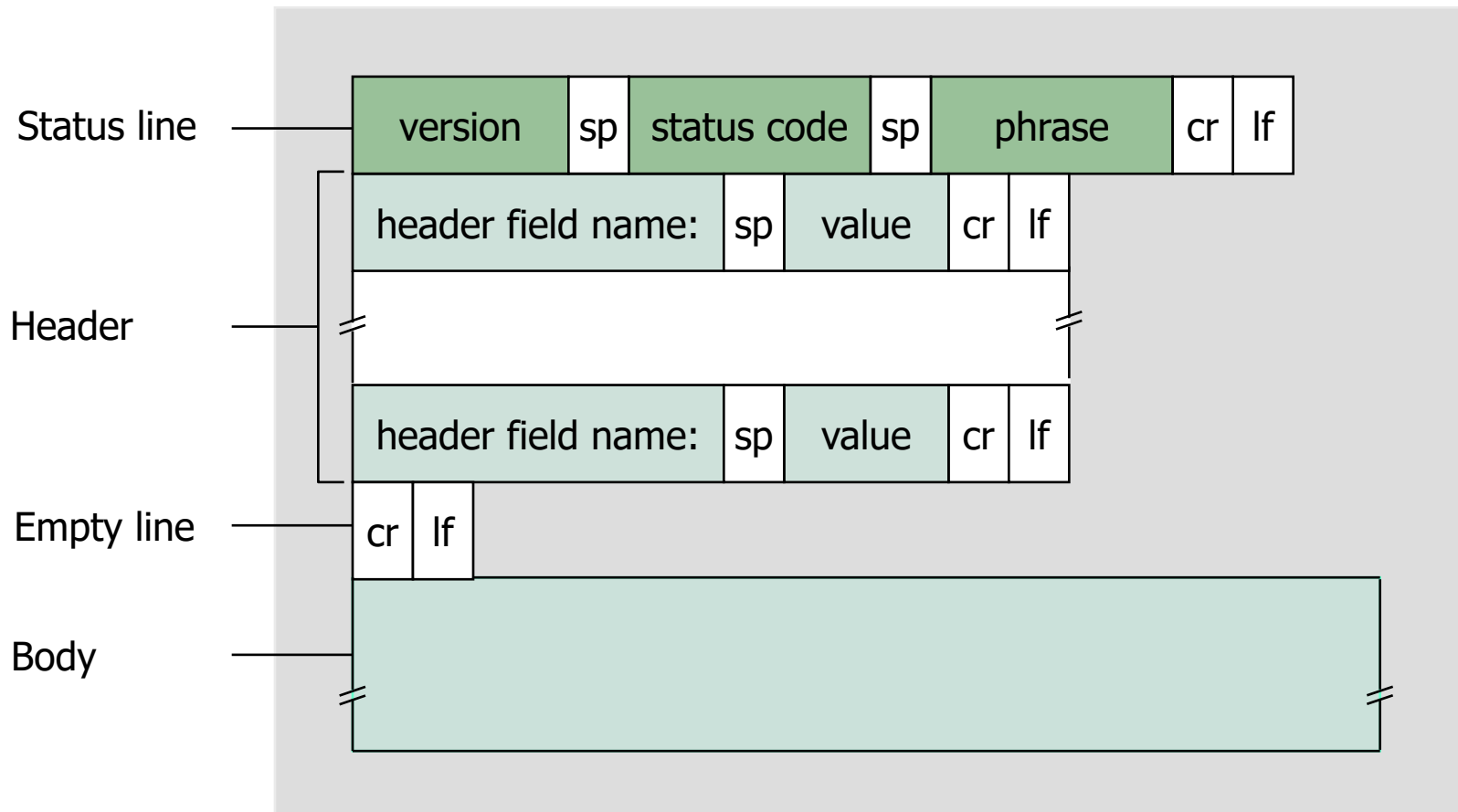
■ Example:

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language: fr
```

(extra carriage return, line feed)

HTTP Response

■ Format



HTTP Response

■ Examples of status codes

- 200 OK
- 301 Moved Permanently (redirection to new location of object)
- 400 Bad Request (request not understood)
- 404 Not Found (object not found)
- 505 HTTP Version Not Supported

■ Example:

```
HTTP/1.1 200 OK
Connection: close
Date: Thu, 06 Aug 1998 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 ...
Content-Length: 6821
Content-Type: text/html

data data data data data ...
```

How does it work – Example

- Calling: <http://www.mylife.org/mypictures.htm>
- After finding out the IP address of the host - DNS
 1. http client initiates a TCP connection on port 80
 2. Client sends the get request via socket established in 1
 3. Server sends the html file, which is encapsulated in its response
 4. http server tells TCP to terminate connection
 5. http client receives the file and the browser parses it ... contains ten jpeg images
 6. Client repeats steps 1-4 for all images

[Kurose & Ross, op. cit.]

Persistency of Connection Usage

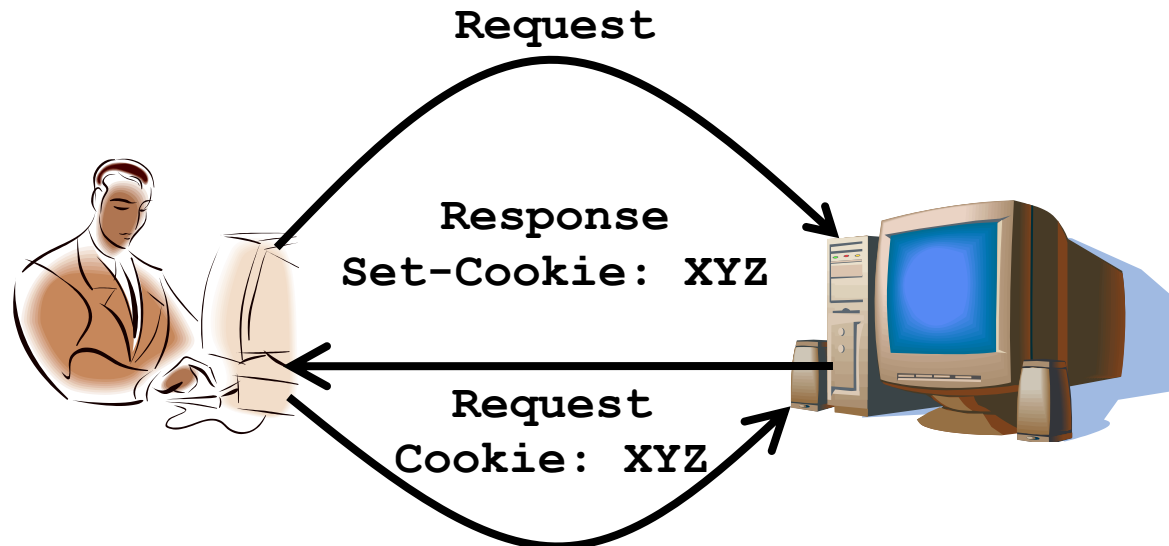
- A web page typically contains many objects
 - E.g., images
 - Each object must be requested with a separate HTTP GET method
 - Non-Persistent Connection:
 - Different TCP connections for each object requested
 - HTTP 1.0
 - Persistent Connection:
 - Reuse the same TCP connection for each object requested
 - HTTP 1.1
 - Pipelining:
 - Support for multiple consecutive requests “back-to-back”
 - HTTP 2.0

HTTP is Stateless

- **Stateless** protocol (e.g., GET, PUT, DELETE)
 - Each request-response exchange treated independently
 - Servers not required to retain state
- Still, requests are **not idempotent**
 - Consider PUT then DELETE then GET vs. PUT then GET then DELETE
- This is **good** as it improves scalability on the server-side
 - Don't have to retain info across requests
 - Can handle higher rate of requests
 - Order of requests doesn't matter
- This is also **bad** as some applications need persistent state
 - Need to uniquely identify user or store temporary info
 - e.g., shopping cart, user preferences/profiles, usage tracking, ...

State in a Stateless Protocol: Cookies

- Client-side state maintenance
 - Client stores small (?) state on behalf of server
 - Client sends state in future requests to the server
- Can provide authentication

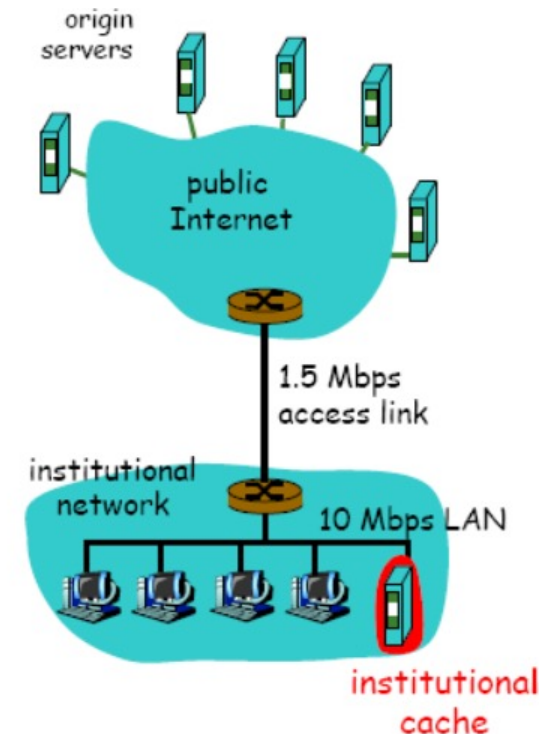


Performance and Reliability

- Problem: You are a web content provider
 - How do you handle millions of web clients?
 - How do you ensure that all clients experience good performance?
 - How do you maintain availability in the presence of server and network failures?
- Solutions:
 - Add more servers at different locations → If you are CNN this might work!
 - Caching
 - Content Distribution Networks (replication)

Caching

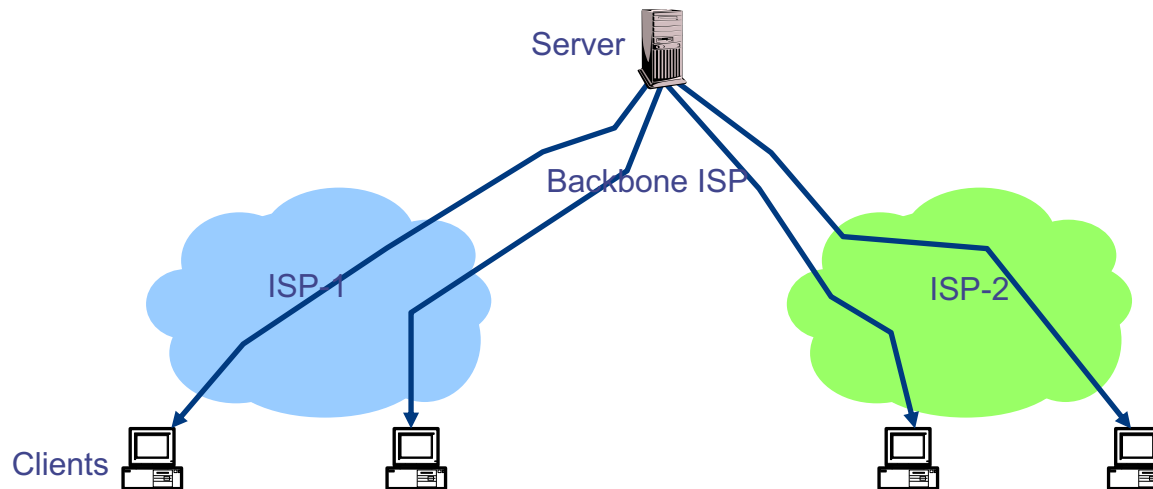
- Store frequently referenced objects closer to the clients
 - Saves time: no need to go all the way to the server (access could look “instantaneously”)
 - Saves access bandwidth
 - Saves web server resources
- Limitations?
 - Frequently changing objects
 - Hit counts
 - Privacy
- Placement? Possibly everywhere:
 - At the servers, at your network, at the client



[Kurose & Ross, op. cit.]

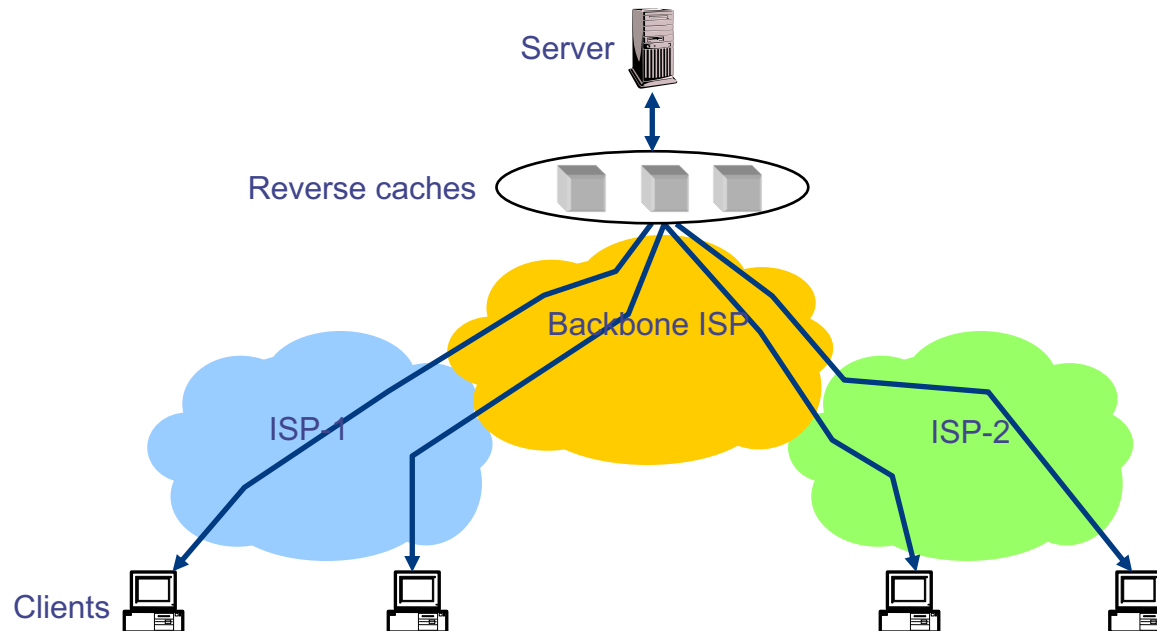
Baseline

- Many clients transfer same information
 - Generate unnecessary server and network load
 - Clients experience unnecessary latency



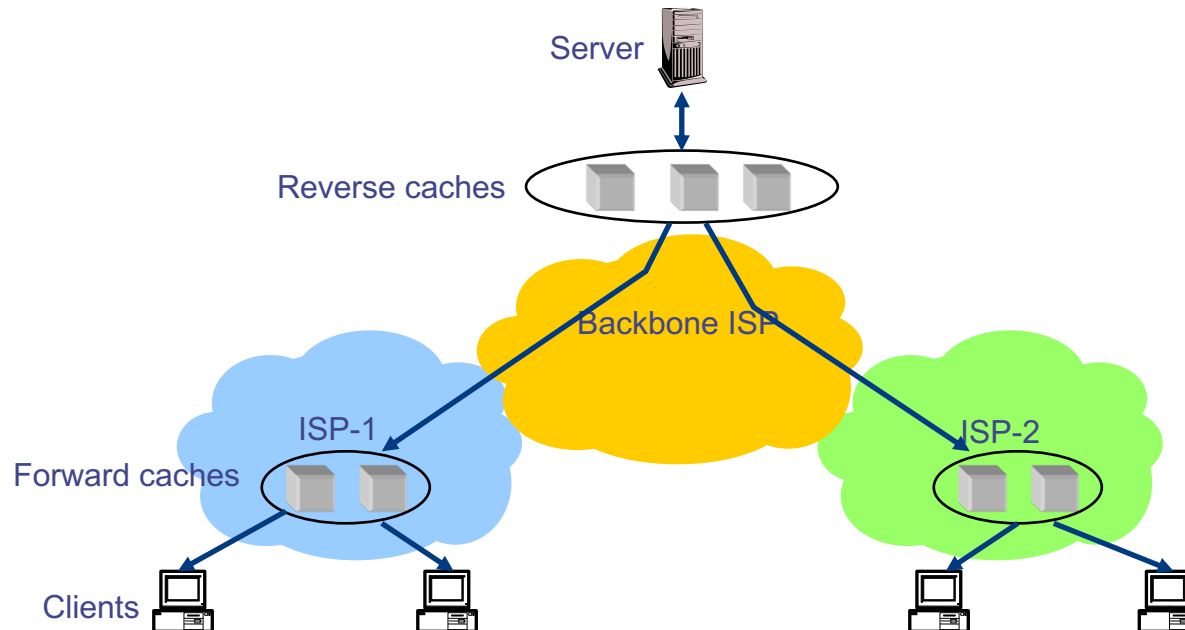
Reverse Caches

- Cache documents close to server → decrease server load
- Typically done by content providers



Forward Proxies

- Cache documents close to clients → reduce network traffic and decrease latency
- Typically done by ISPs or corporate LANs

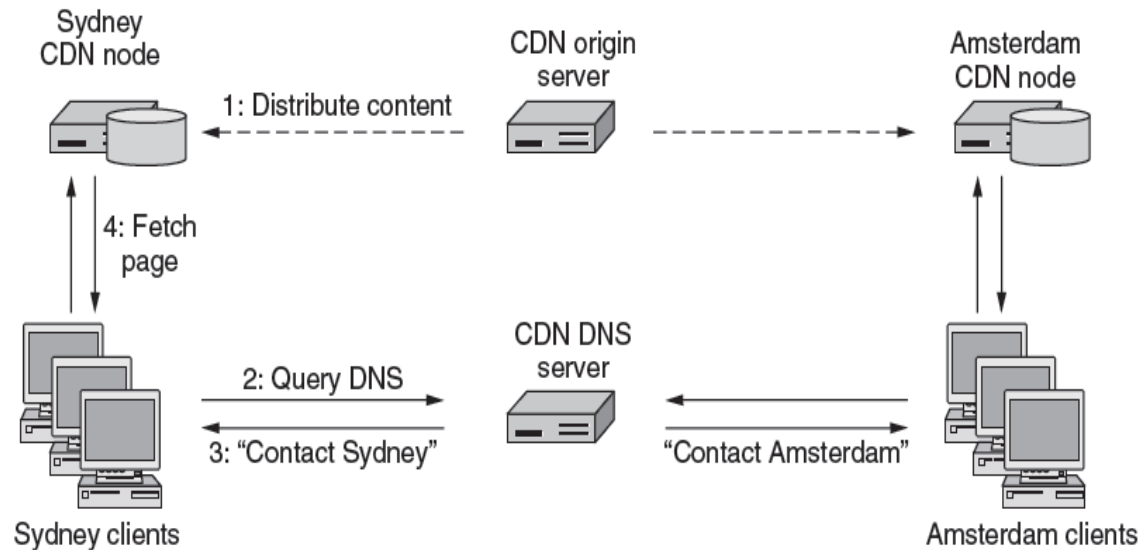


Client Side - HTTP/1.0 Caching Support

- Exploit locality of reference
- A modifier to the GET request:
 - **If-modified-since** – return a “not modified” response if resource was not modified since specified time
- A response header:
 - **Expires** – specify to the client for how long it is safe to cache the resource
- A request directive:
 - **No-cache** – ignore all caches and get resource directly from server
- These features can be best taken advantage of with HTTP proxies
 - Locality of reference increases if many clients share a proxy

Content Delivery Network

- DNS resolution of site gives different answers to clients
 - Tell each client the site is the nearest replica (map client IP)



[Tanenbaum & Wetheral, op. cit.]

Example: www.akamai.com

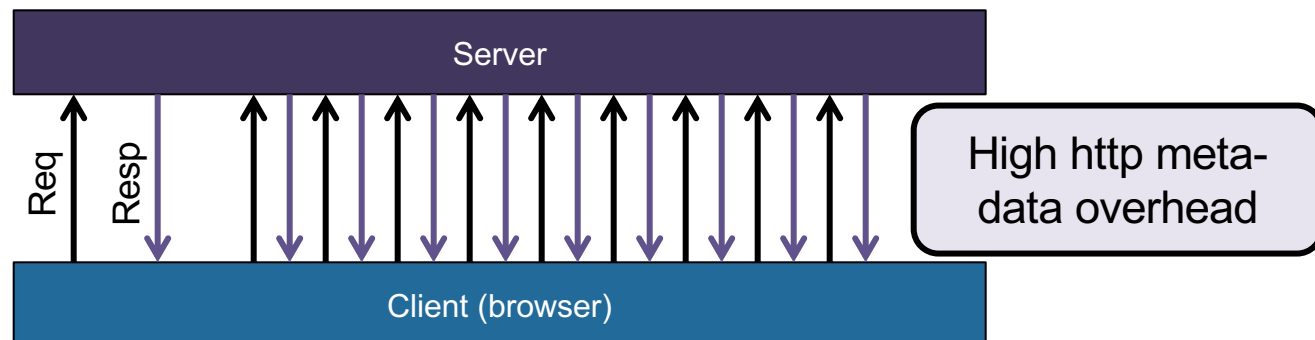
- DNS records don't have to store the real IP address of the host.
- One name can map onto multiple addresses
 - Example: www.yahoo.com can be mapped to multiple machines
 - Example: www.akamai.com
 - From Berkeley 64.164.108.148
 - From the NY Area 63.240.15.146
 - From the UK 194.82.174.224
- What do we gain: shorter delay, load distribution

Content Distribution Networks

- Verteilung der Anfragen
 - **Server-basierte HTTP Redirection:** Server liefert aufgrund der IP-Adresse des Clients einen geeigneten anderen Server, erfordert zusätzliche RTT, Gefahr der Überlast für Server
 - **Client-nahe HTTP-Redirection:** z.B. durch Web-Proxy, schwieriger zu verwirklichen
 - **DNS-basierte Redirection:** DNS-Server bildet den Domain-Namen des Servers auf die IP-Adresse eines geeigneten Servers ab
 - **URL-Rewriting:** Server liefert Basisseite, die URLs der eingebetteten Objekte werden umgeschrieben, mit dem Domain-Namen eines geeigneten anderen Servers
 - kommerzielle CDNs verwenden meist Kombination aus DNS-basierter Redirection und URL-Rewriting

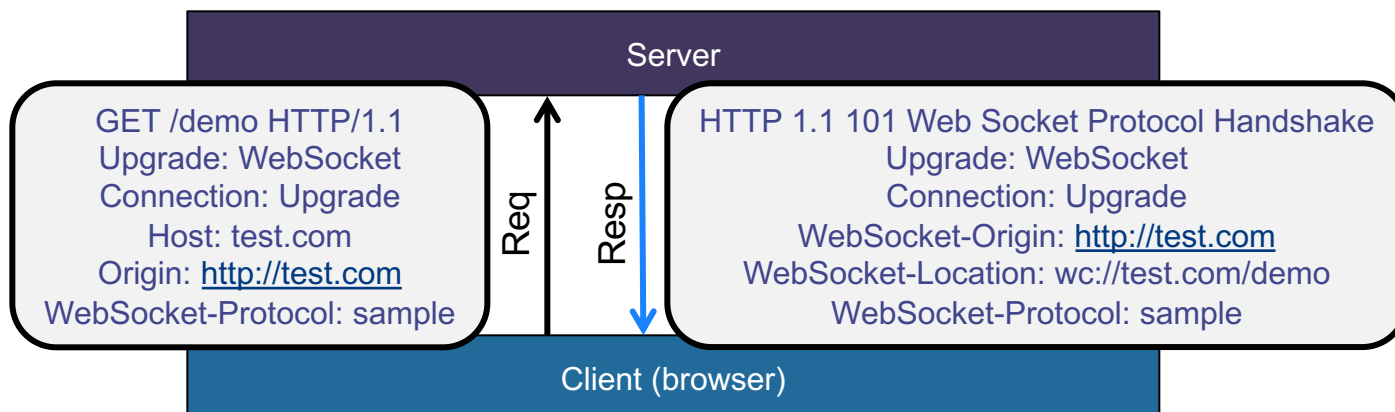
HTML5 WebSocket API

- Why do we need WebSockets? (or something similar!)
 - Some web apps demand real-time, event-driven communication with minimal latency
 - E.g. financial applications, online games, ...
 - Problems with HTTP
 - Request-reply pattern with half duplex data flow (traffic flow in only one direction at a time)
 - Adds latency
- Typical use case: polling
 - Poll server for updates, wait at client



What is a WebSocket (WS)?

- W3C/IETF standard
- Uses WebSocket protocol instead of HTTP: ws:// and wss://
- Establishes a true full-duplex communication channel
 - Strings + binary frames sent in any direction at the same time
- Uses port 80 (http) or 443 (https) to traverse firewalls (-> proxy/firewall)
- Connection established by „upgrading“ from HTTP to WebSocket protocol



WebSockets Advantages

- ... as compared to „bare“ sockets
- Traversing firewalls (avoids blocking of the port)
- “The name resolution” and agreement on port implicit
- After establishing the connection server can send info to client anytime:
 - Update latency reduced
 - No polling = reduced data traffic! (but might require client heartbeat)
 - Each message frame has only 2 Bytes of overhead
- Server handles fewer transport level connection requests
 - Might reset the connection if no new info's for the „client“

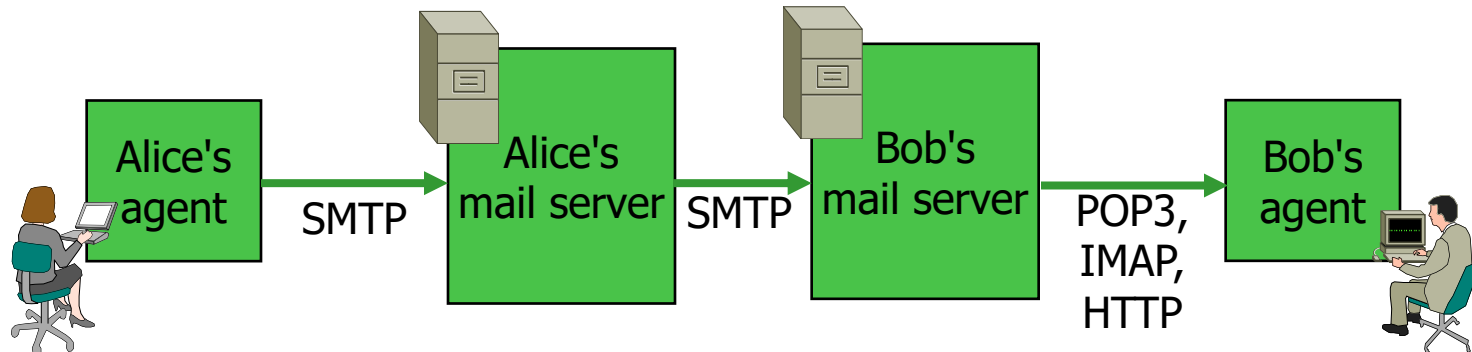
Add-on: HTTP/2

- Derived from the earlier experimental SPDY protocol (→ Google),
- Published as RFC 7540 in 2015,
- Data compression of HTTP headers
- HTTP/2 Server Push
- Pipelining of requests
- Fixing the head-of-line blocking problem in HTTP 1.x
- Multiplexing multiple requests over a single TCP connection

Email

■ Simple Mail Transfer Protocol (SMTP)

- ASCII formatted messages consisting of header and body part
- Attachments need to be converted to ASCII – multimedia mail extension (MIME)
- Sending emails: SMTP (, HTTP)
- Receiving emails: POP3, IMAP (, HTTP)



Email: SMTP [RFC 821]

- Uses TCP for reliable communication, standard port 25
- Sending system becomes client, receiving system acts as server
- Three phases
 - Handshaking
 - Message exchange
 - Final commit
- Protocol uses commands and response messages
 - Commands: ASCII text
 - Responses: status code and text
- Messages must be encoded in **7-bit ASCII**

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
```