

Introduction to Communication Networks and Distributed Systems

Unit 4 – DNS & WWW -

Fachgebiet Telekommunikationsnetze

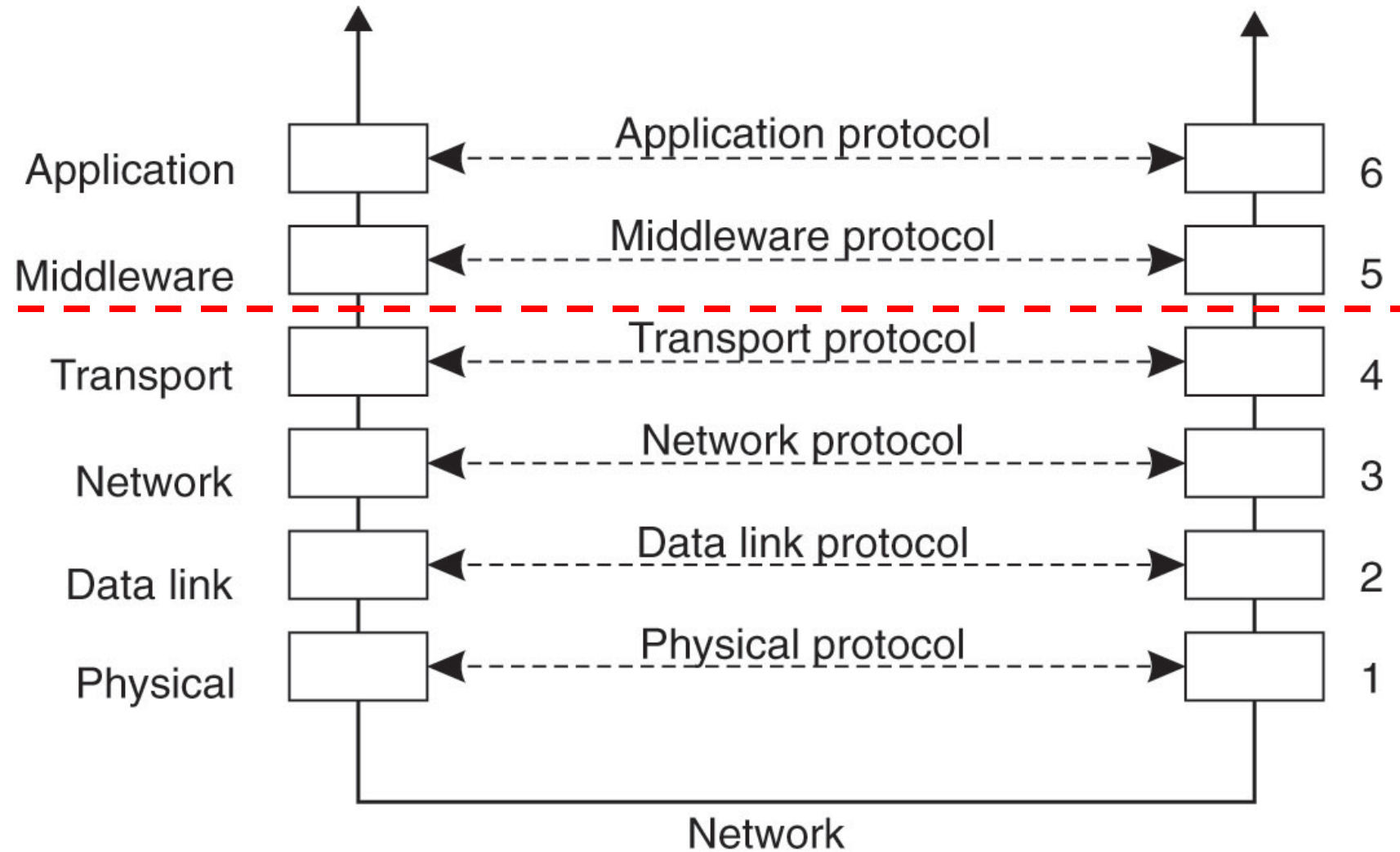
Prof. Dr. A. Zubow

zubow@tkn.tu-berlin.de

Acknowledgements

- We acknowledge the use of slides from: **Prof. Wolisz**, Prof. Holger Karl, Paderborn; Prof. Ion Stoica, Berkeley, Prof. Ashay Parekh; Berkeley; Prof Lauer WPI; Prof. Baker, ACET, as well as slides from books by Tanenbaum, Kurose and Ross, Colouris et al.

An adapted Communication Model [Tanenbaum, op. cit.]



Reminder: Notions

- **Names**

- Data types that refer to specific entities in a system
- Example: Jonathan M. Smith

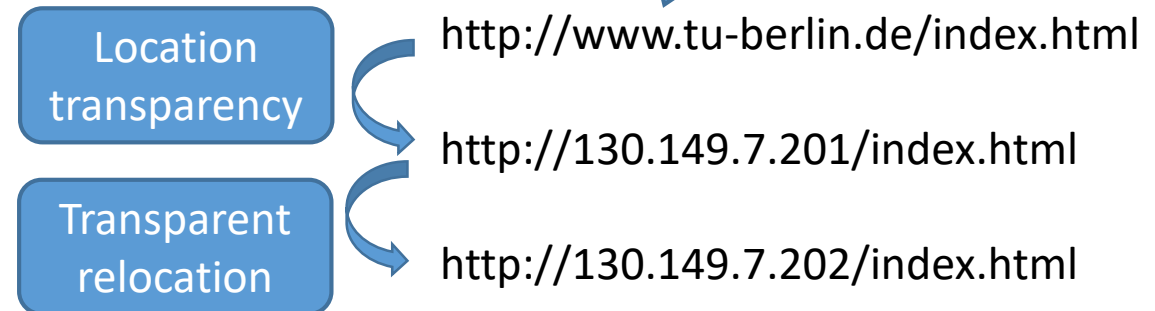
- **Addresses**

- Identifiers of places to find the named entities
- Example: 123 Park St., Andover, MA

- **Routes** (also called paths)

- Steps to follow to get to named entities
- Example:
 - From Andover center, go west 1.2 miles
 - Turn right, then take 3rd left
 - He lives at the 2nd house on the right

Ex.:



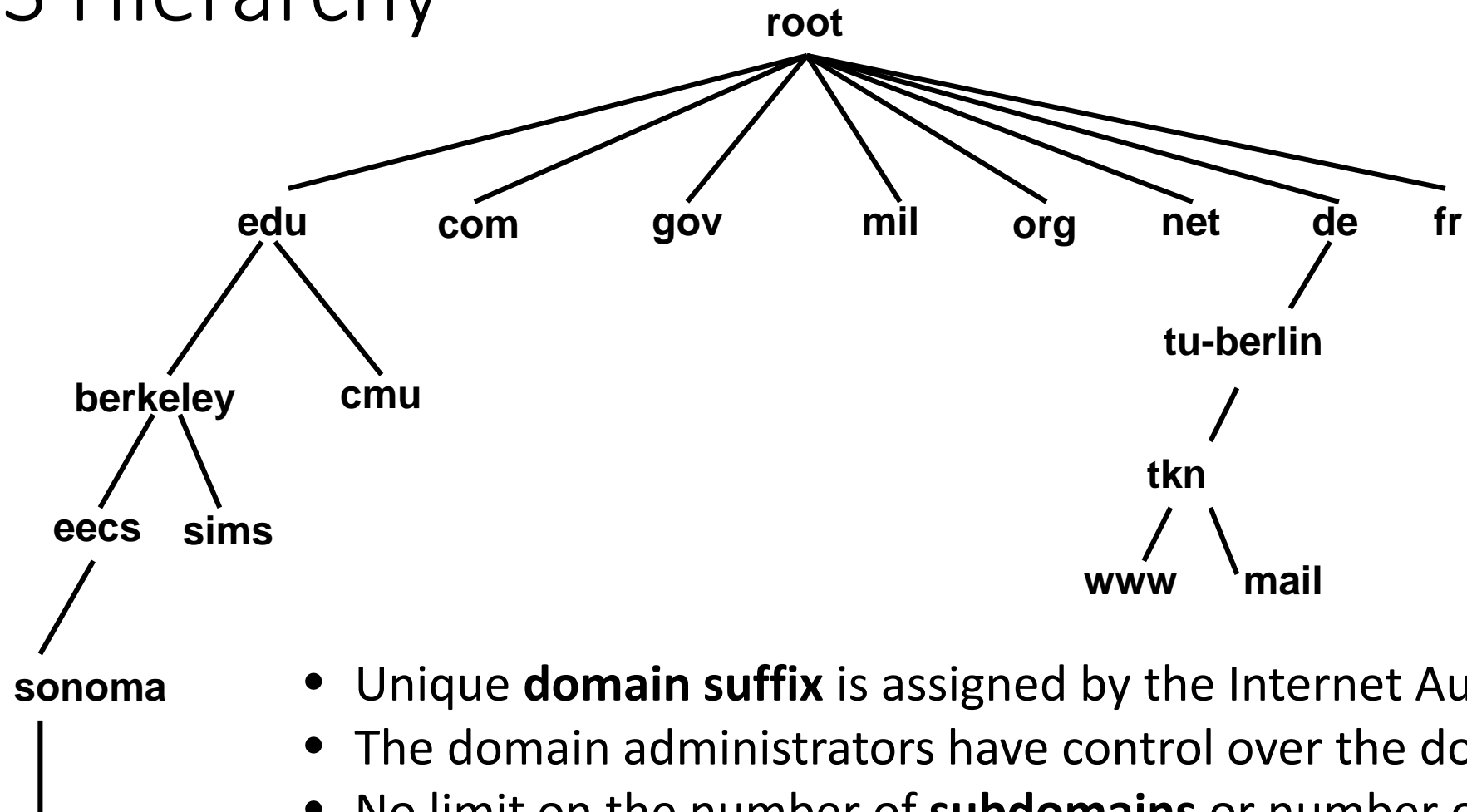
Name Service

- **Name space:** set of possible names and their relationship
 - Names are usually chosen so as to be meaningful/easy to memorize
- **Bindings:** the mapping between names and addresses
- **Name resolution:** procedure that, when invoked with a name, returns the corresponding value
- **Name server (*directory*):** specific implementation of a resolution mechanism that is available on the network and that can be queried by sending messages
 - Uses a distributed database

Internet: Domain Name System (DNS)

- **Hierarchical namespace**
- Distributed architecture for storing names
 - Name servers assigned zones of the hierarchical namespace
 - Backup servers available for redundancy
- Administration divided along the same hierarchy
- Client server interaction on connectionless (UDP) port 53

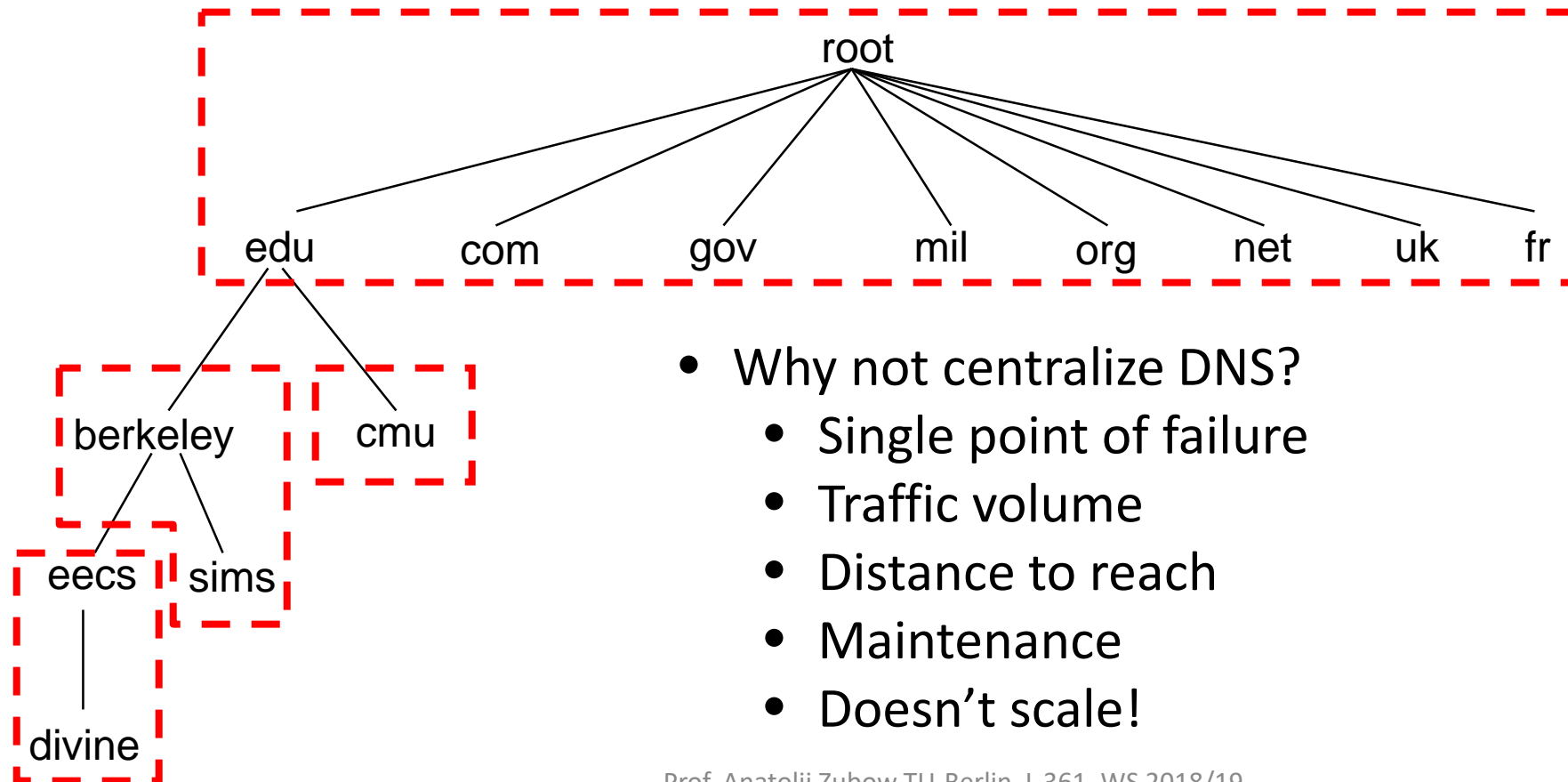
DNS Hierarchy



- Unique **domain suffix** is assigned by the Internet Authority
- The domain administrators have control over the domain
- No limit on the number of **subdomains** or number of levels
- Name space is not related with the physical interconnection
- A name could be a domain or an individual object

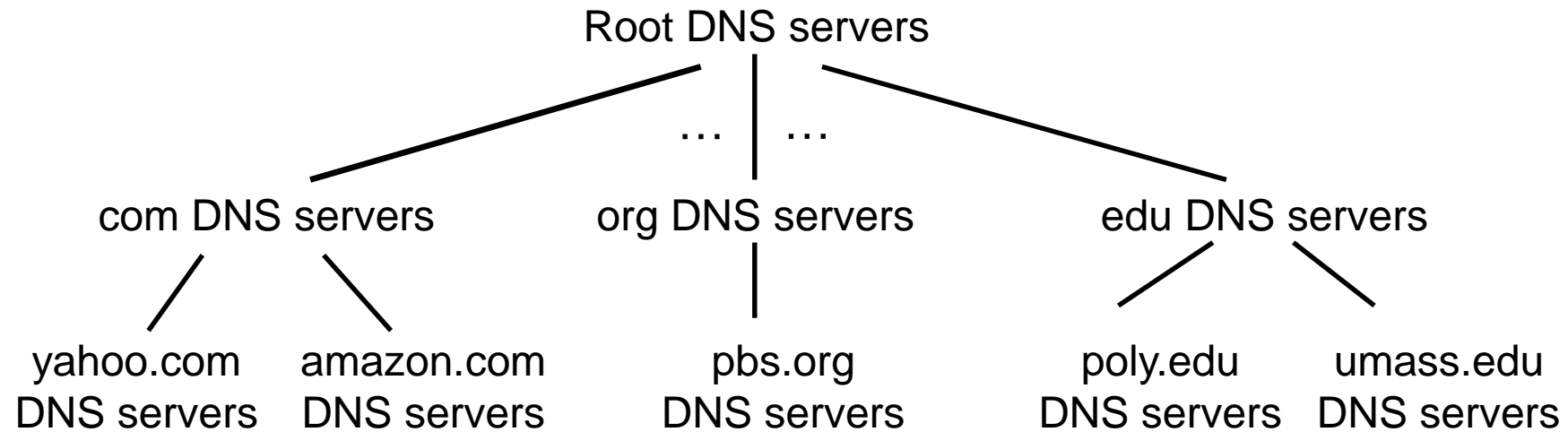
Server Hierarchy: Zones

- A zone corresponds to an administrative authority that is responsible for that portion of the hierarchy.



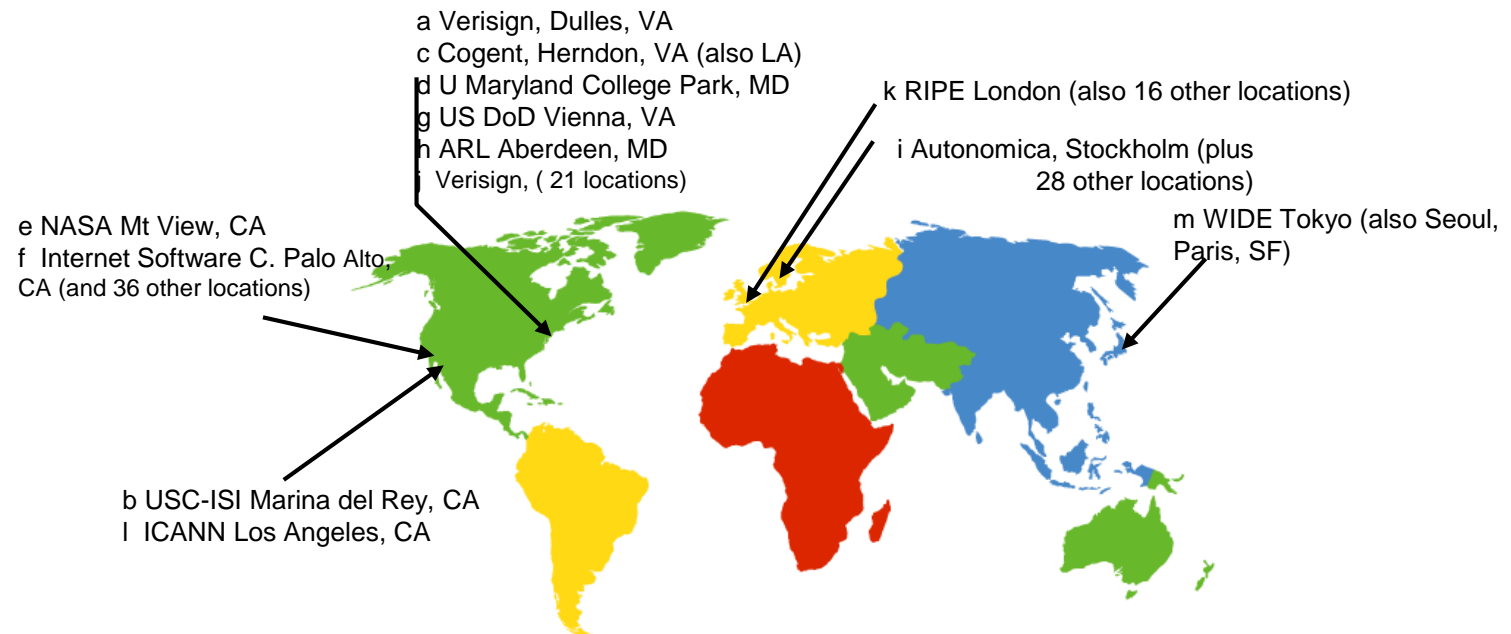
- Why not centralize DNS?
 - Single point of failure
 - Traffic volume
 - Distance to reach
 - Maintenance
 - Doesn't scale!

DNS: A Distributed, Hierarchical Database



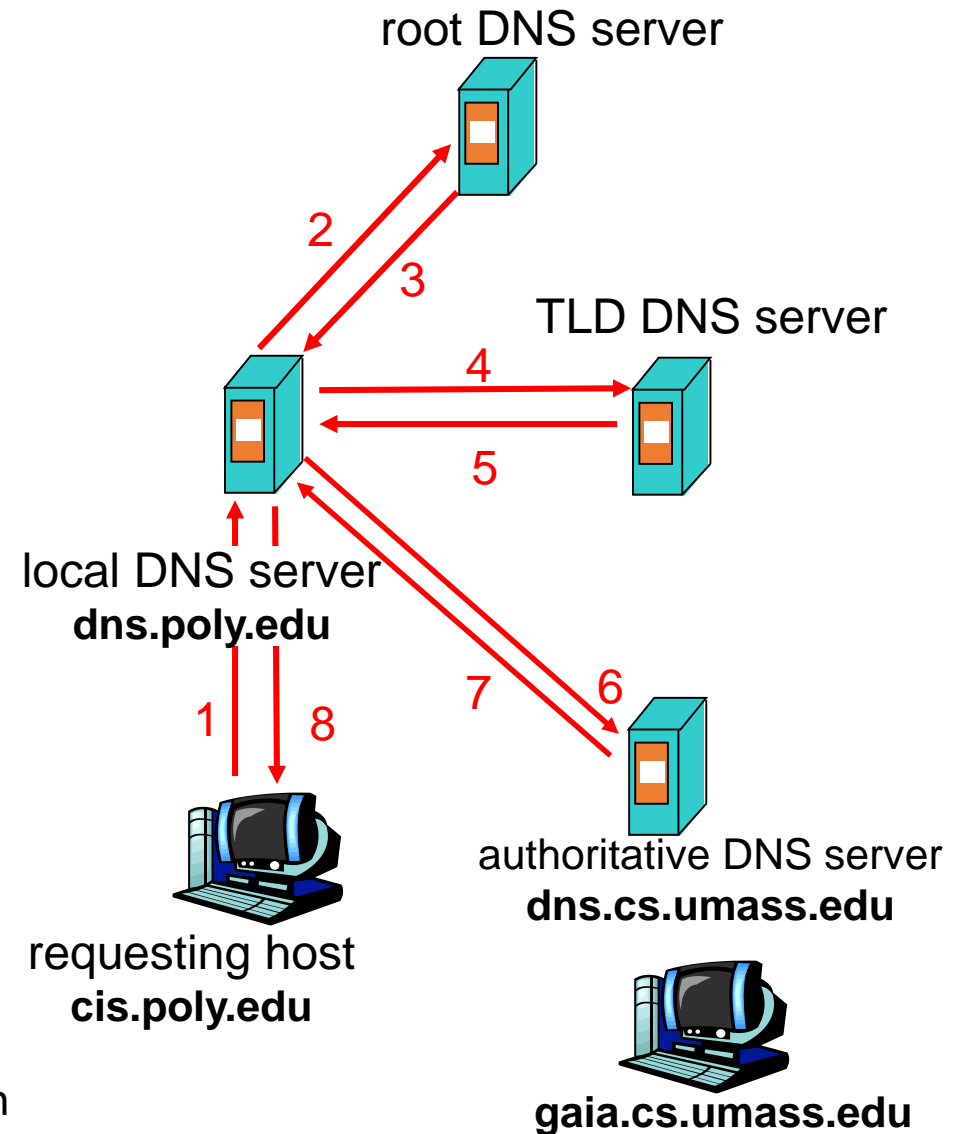
DNS: Root Name Servers

- Contacted by local name server that cannot resolve name
- Is a critical part of the Internet!
- Therefore replicated - 13 root name servers worldwide
- Have to carry consistent content



DNS Name Resolution

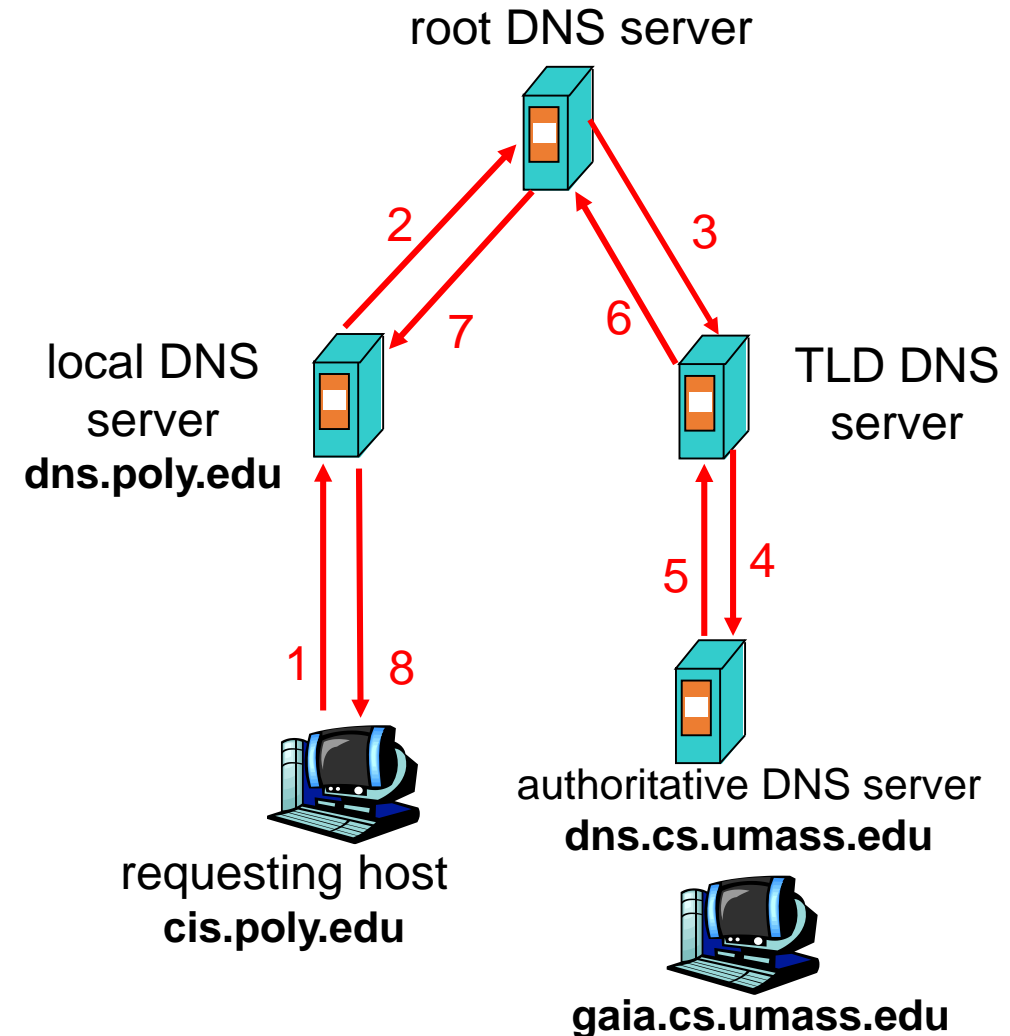
- Example: host at *cis.poly.edu* wants IP address for *gaia.cs.umass.edu*
- **Iterated query:**
 - Contacted server replies with name of server to contact
 - “I don’t know this name, but ask this server”



Alternatively: Recursive Query Resolution

- **Recursive query:**

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



Authoritative/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 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/port – 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)

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

Kontakt Impressum Sitemap English Index A-Z Mobil Datenschutz

suchen nach ...

TU-Berlin
Technische Universität Berlin

TUB-Login
mit Passwort
mit Campuskarte / Zertifikat

Studieninteressierte Studierende Beschäftigte Wirtschaft Presse Alumni

Über die TU Berlin
Einrichtungen
Fakultäten & Zentralinstitute
Forschung
Studium & Lehre
Internationales
Service

Gerd Müller im Gespräch mit Bill Gates
Aufzeichnung der Veranstaltung "Innovationen für eine gesunde Zukunft" verfügbar

© David Ausserhofer

TU-Ticker

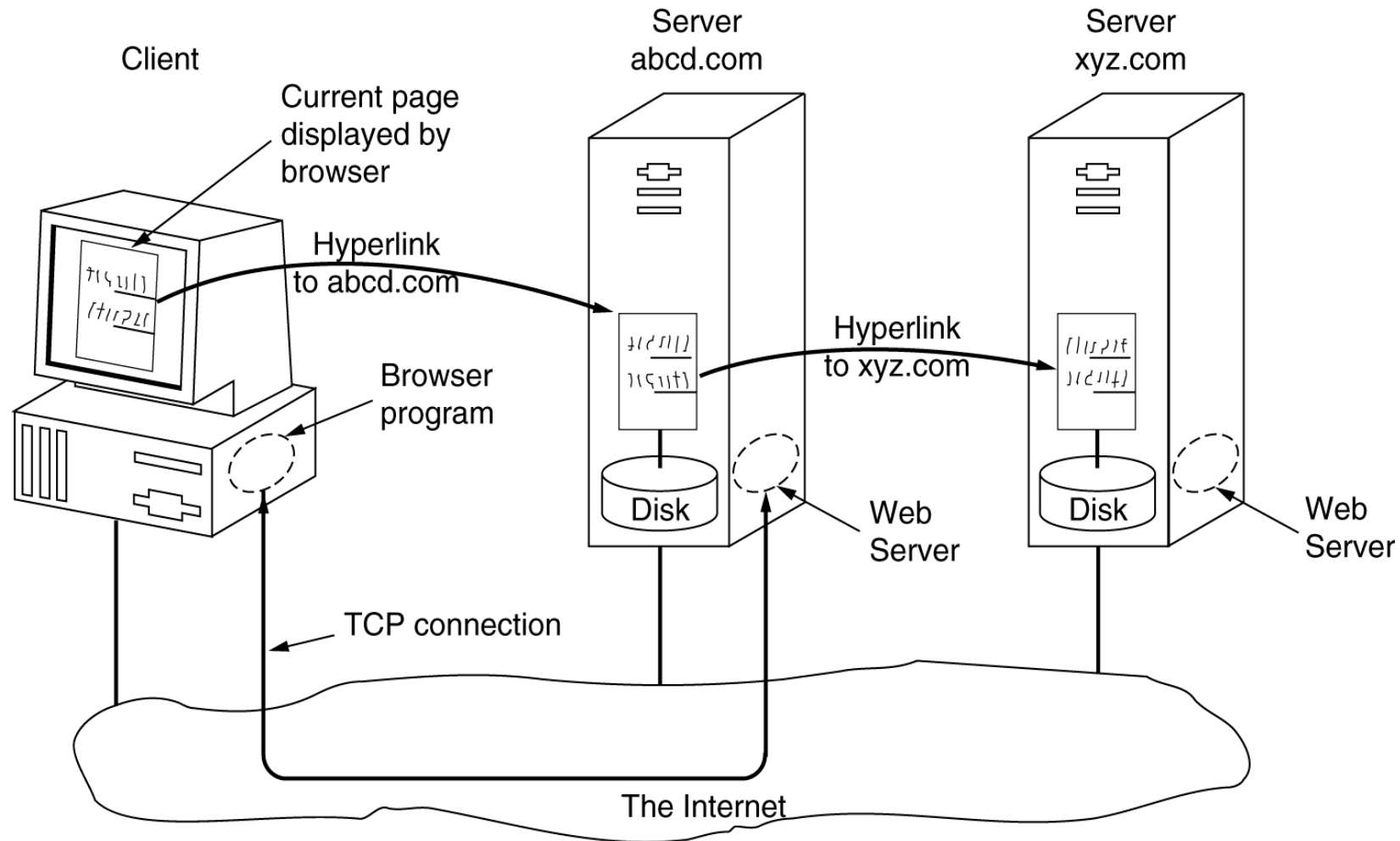
Direktzugang
Gehe zu: 85

Hilfsfunktionen
Schriftgröße: - +
Infos zur Barrierefreiheit
Hilfe zur Bedienung
Tastaturkürzel
Hilfsfunktionen minimieren

Berlin University

Architectural Overview

[Tanenbaum, op. cit.]



WWW

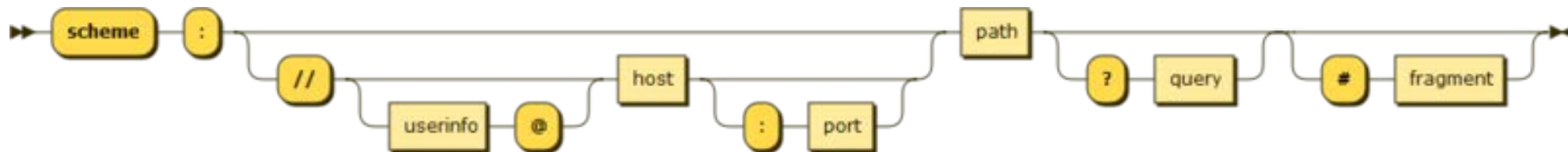
- A distributed database of URLs (Uniform Resource Locators)
- Core components:
 - Servers which store “web pages” and execute remote commands
 - Browsers retrieve and display “pages” of content linked by hypertext
 - Each link is a URL
 - Can build arbitrarily complex applications, all of which share a uniform client!
- 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)

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

http protocol **hostname** **file name**

- 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)

Web Documents

[Tanenbaum, op. cit.]

- Six top-level MIME types and some common subtypes.

Type	Subtype	Description
Text	Plain	Unformatted text
	HTML	Text including HTML markup commands
	XML	Text including XML markup commands
Image	GIF	Still image in GIF format
	JPEG	Still image in JPEG format
Audio	Basic	Audio, 8-bit PCM sampled at 8000 Hz
	Tone	A specific audible tone
Video	MPEG	Movie in MPEG format
	Pointer	Representation of a pointer device for presentations
Application	Octet-stream	An uninterpreted byte sequence
	Postscript	A printable document in Postscript
	PDF	A printable document in PDF
Multipart	Mixed	Independent parts in the specified order
	Parallel	Parts must be viewed simultaneously

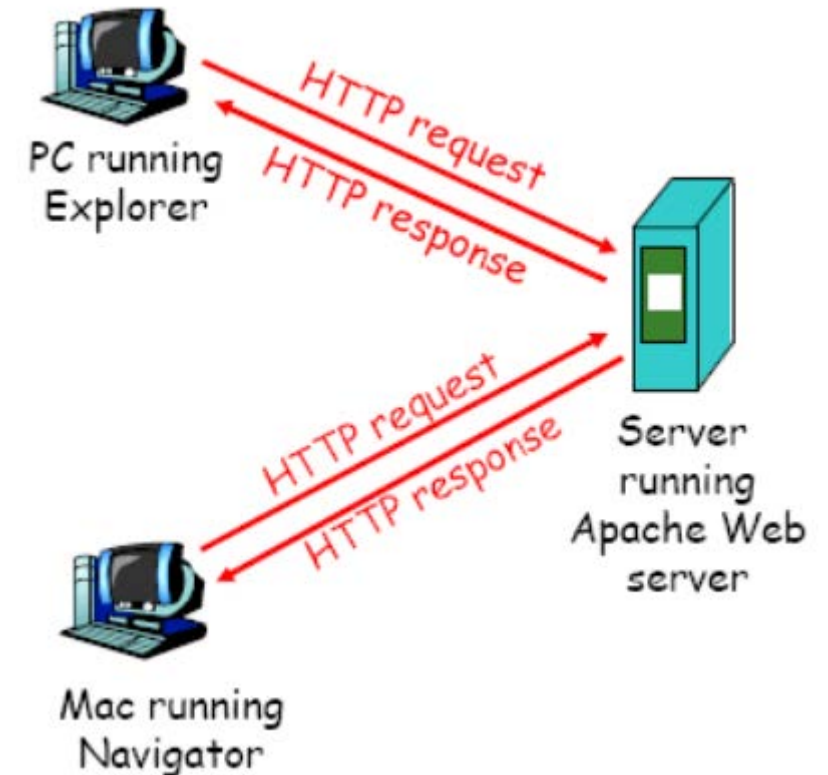
XML - Extensible Markup Language

- XML is a text-based markup language that is fast becoming the standard for data interchange on the Web.
- XML has syntax analogous to HTML.
- Unlike HTML, XML tags tell you what the **data means**, rather than how to **display it**.
- Example:

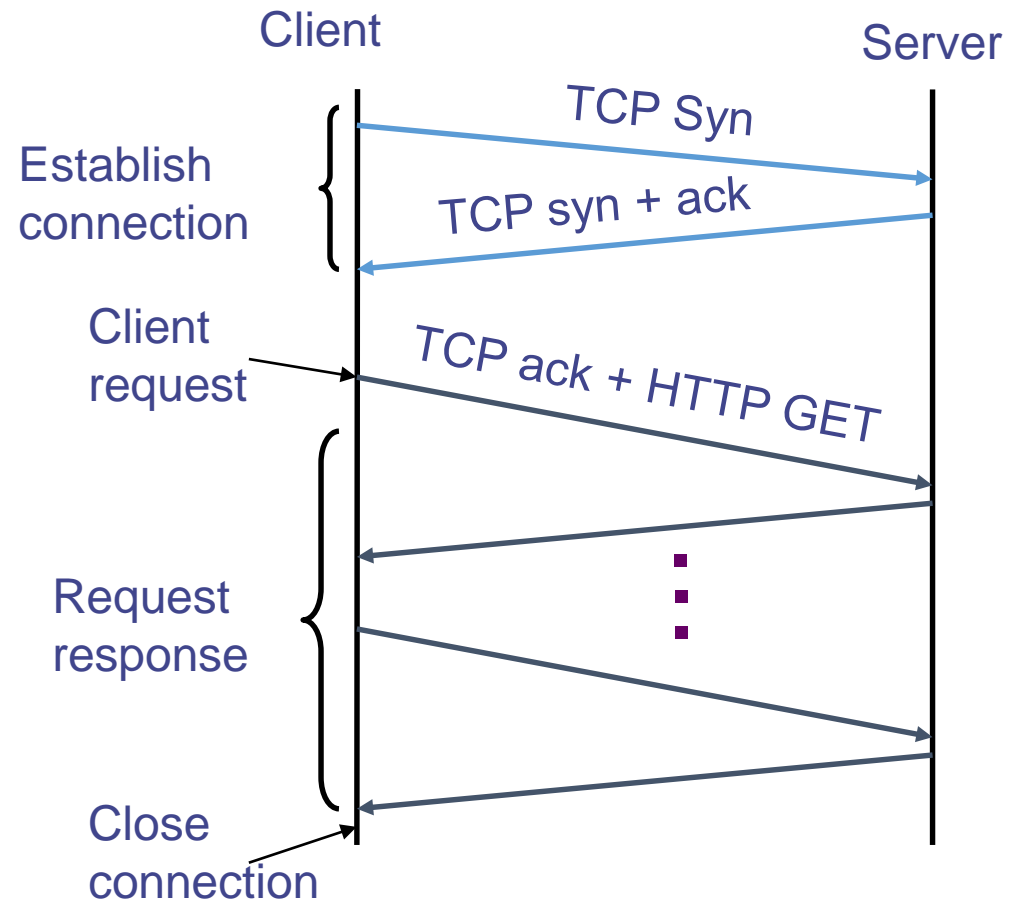
```
<message>  
  <to>you@yourAddress.com</to> <from>me@myAddress.com</from>  
  <subject>XML Is Really Cool</subject>  
  <text> How many ways is XML cool? Let me count the ways... </text>  
</message>
```

Hypertext Transfer Protocol (HTTP) [Parekh, op. cit.]

- 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
- Note: server is **stateless**, i.e. each request is self contained!
- HTTP standards:
 - 1.0: RFC 1945 (1996)
 - 1.1: RFC 2068 (1999)
 - 2.0: RFC 7540 (2015)



Big Picture - HTTP



Most important HTTP Methods

Operation	Description
Head	Request to return the header of a document
Get	Request to return a document to the client
Put	Request to store a document
Post	Provide data that are to be added to a document (collection)
Delete	Request to delete a document

How does it work – Example

[Kurose & Ross, op. cit.]

Calling: <http://www.mylife.org/mypictures.htm>

0. After finding out the IP address of the host - DNS
1. http client initiates a TCP connection on :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

Persistency of Connection Usage

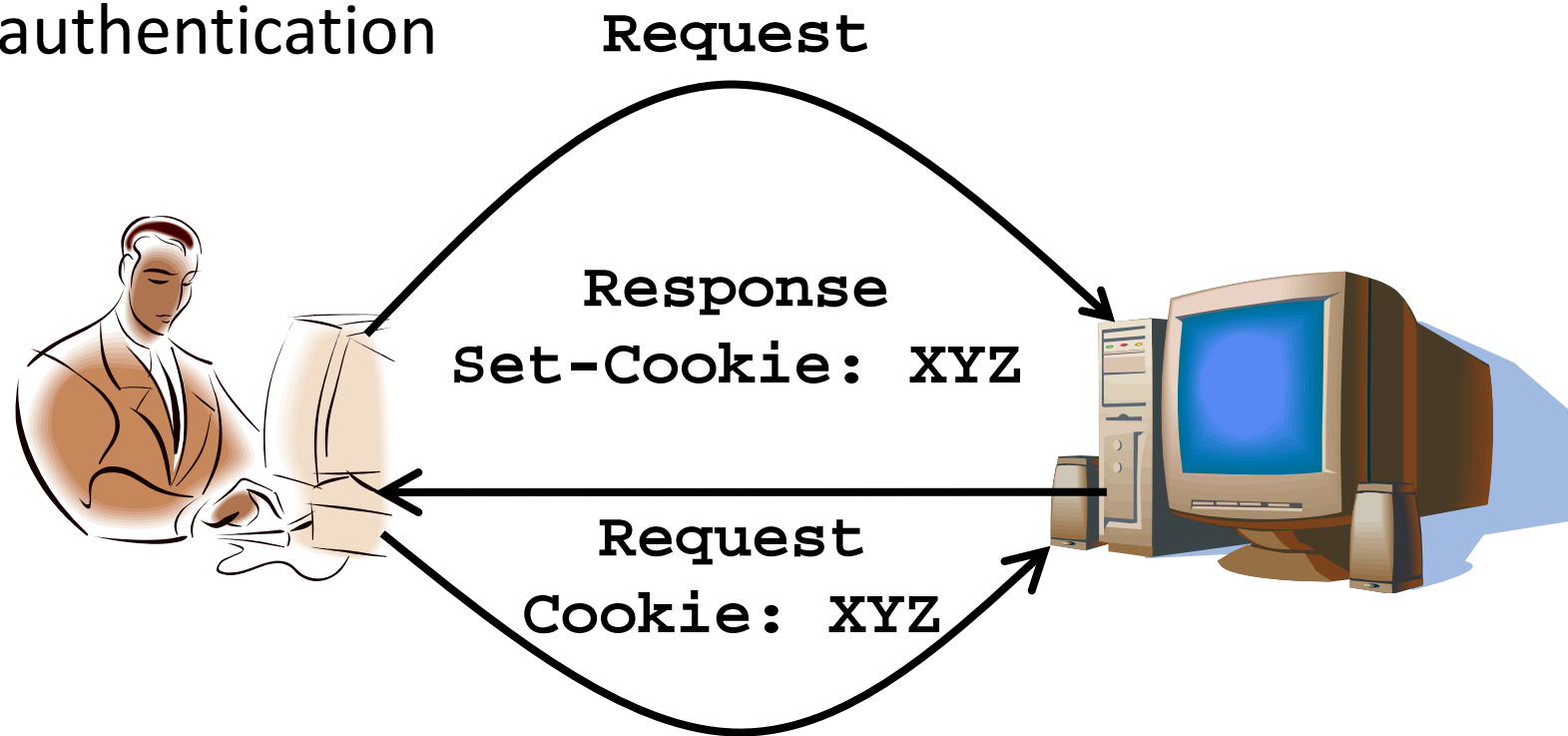
- A web page typically contains many objects
 - E.g. images
 - Each object must be requested with a separate HTTP GET method
 - Non Persistent Connection:
 - Different TCP connections for each object requested
 - HTTP 1.0
 - Persistent Connection:
 - Reuse the same TCP connection for each object requested
 - HTTP 1.1

HTTP is Stateless

- **Stateless** protocol (e.g. GET, PUT, DELETE)
 - Each request-response exchange treated independently
 - Servers *not* required to retain state
- 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



Notion of Fate-Sharing

- Idea: when storing **state** in a distributed system, keep it **co-located** with the entities that ultimately rely on the state
- Fate-sharing is a technique for dealing with **failure**
 - Only way that failure can cause loss of the critical state is if the entity that cares about it also fails ...
 - ... in which case it doesn't matter
- Often argues for keeping network state at end hosts rather than inside routers
 - In keeping with end-to-end principle
 - E.g., packet-switching rather than circuit-switching
 - E.g., HTTP “cookies”

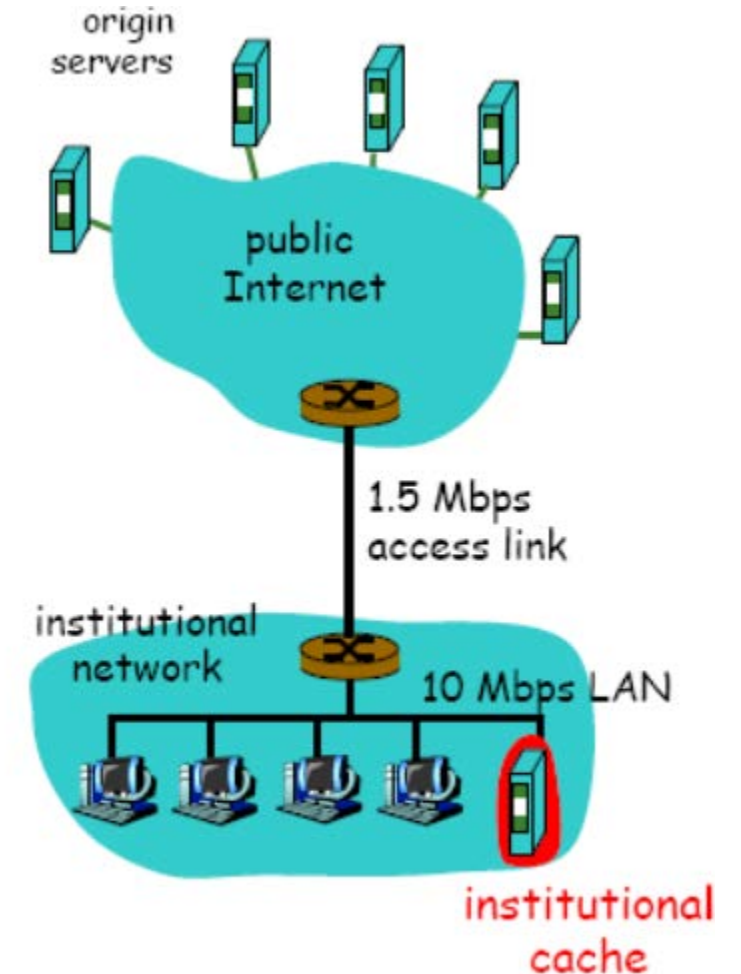
Performance and Reliability

- 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

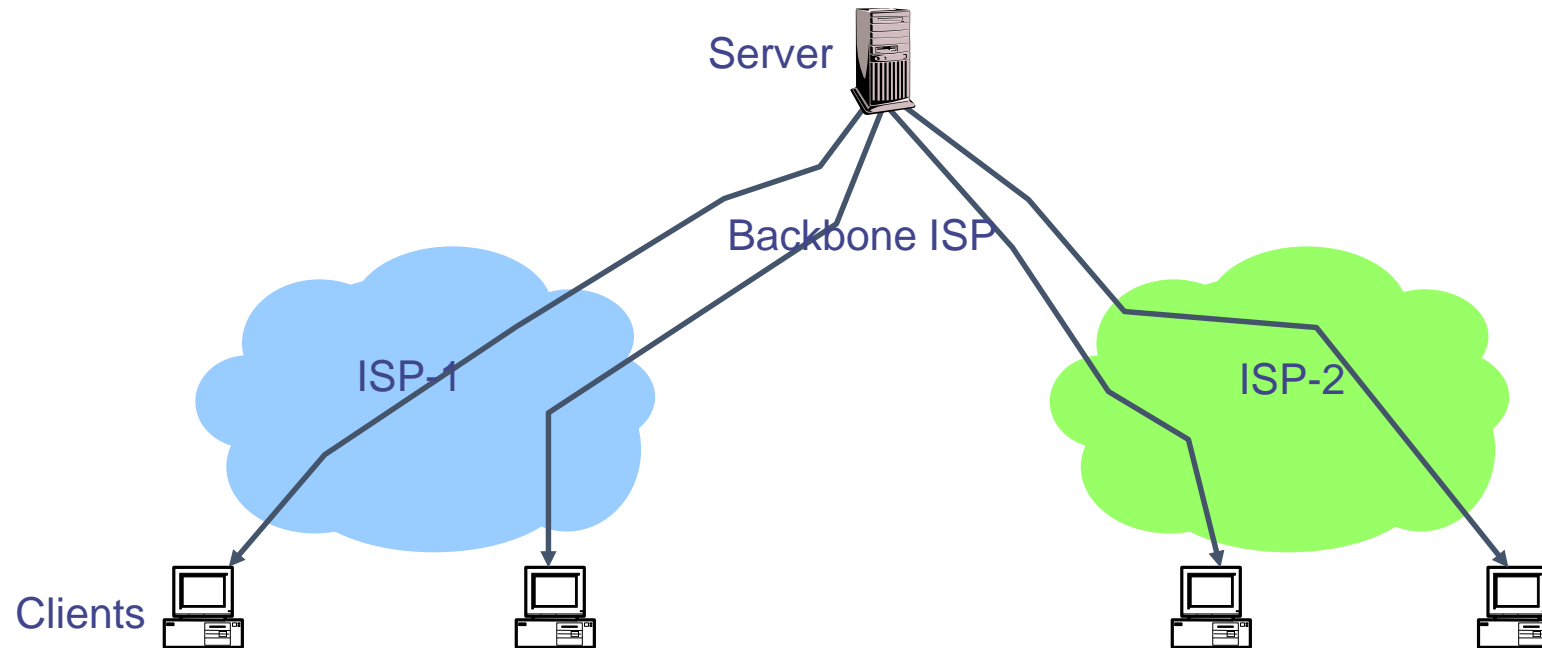
- Store frequently referenced objects closer to the clients
 - Saves time: no need to go all the way to the server (access could look “instantaneously”)
 - Saves access bandwidth
 - Saves web server resources
- Limitations?
 - Frequently changing objects
 - Hit counts
 - Privacy
- Placement? Possibly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

[Kurose & Ross, op. cit.]



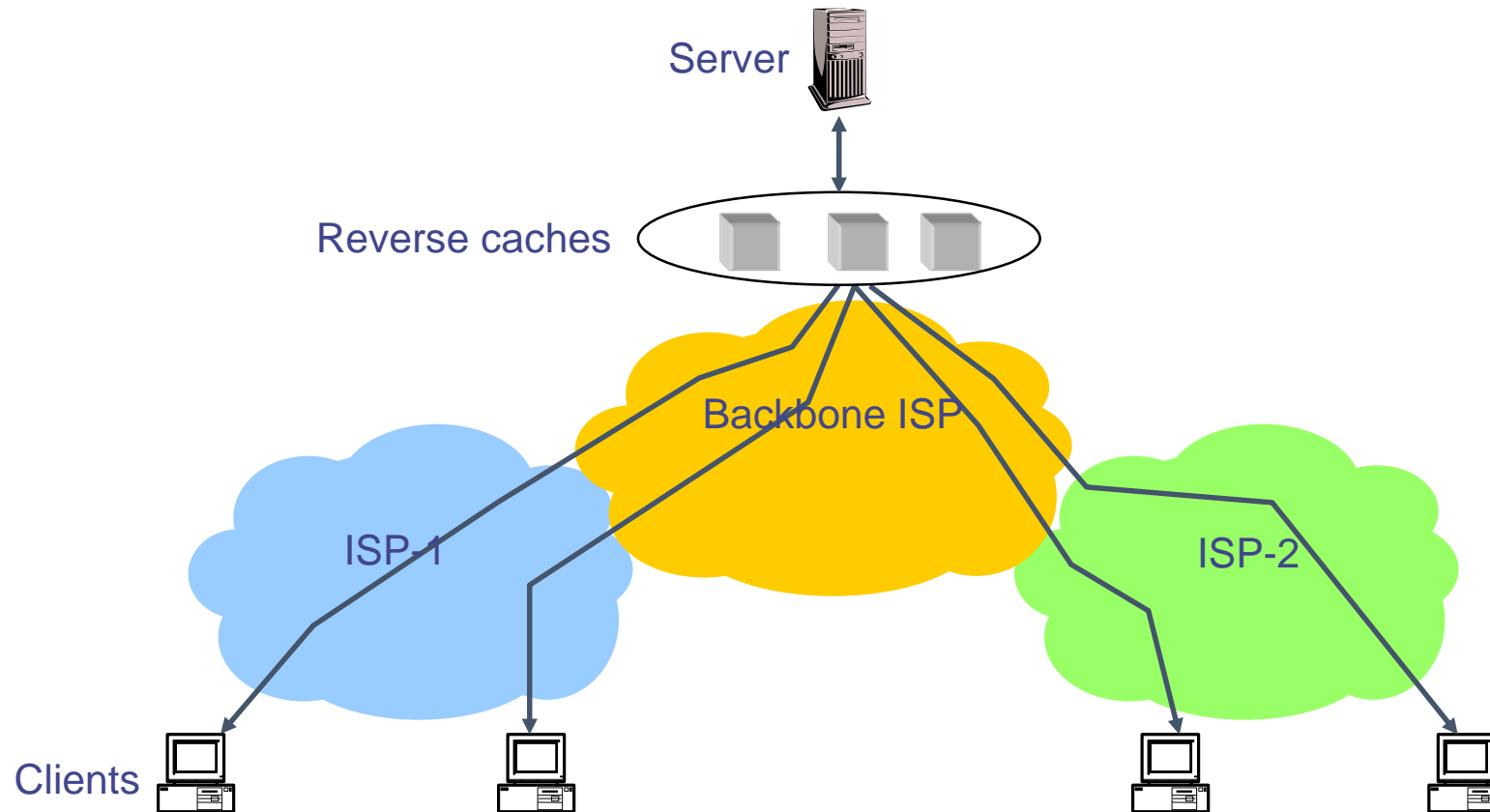
Baseline

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



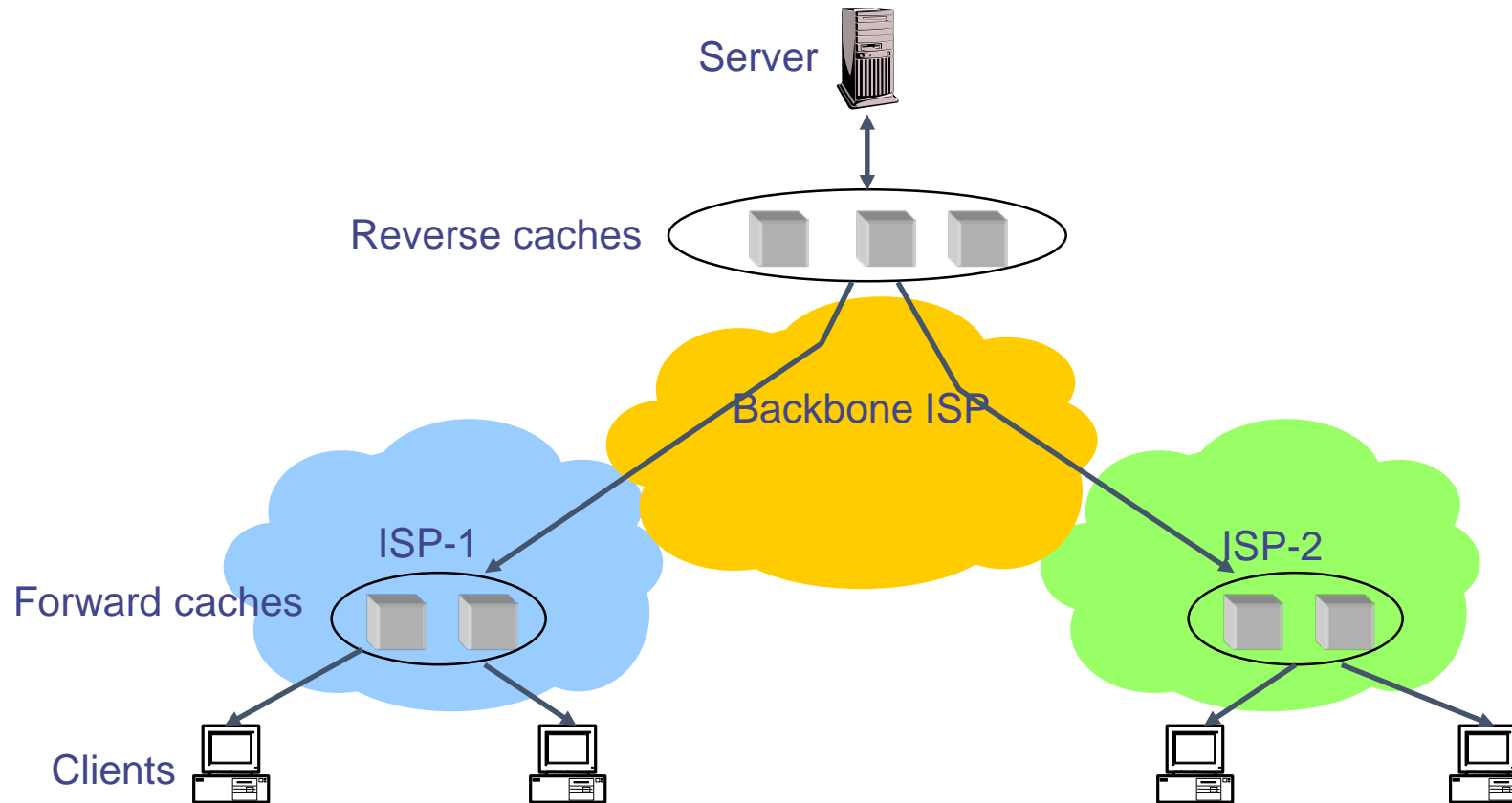
Reverse Caches

- 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



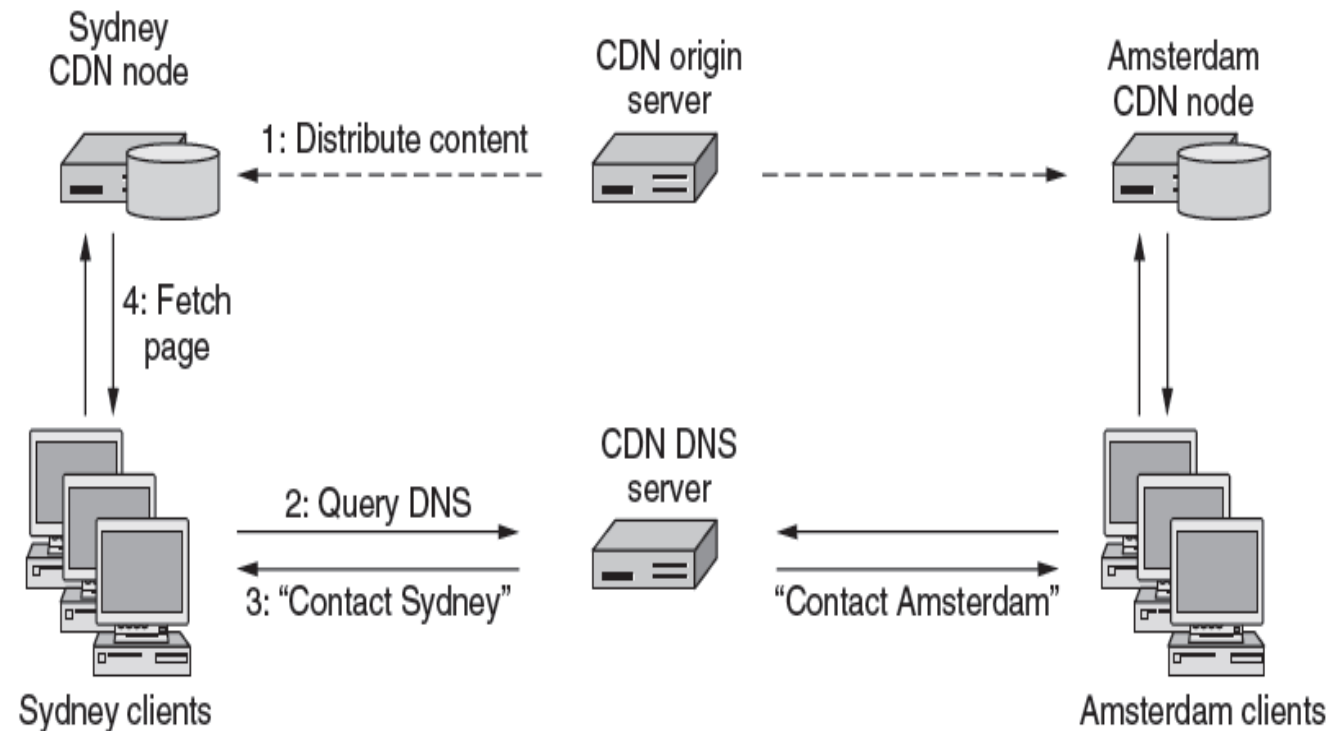
Client Side - HTTP/1.0 Caching Support

- 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

[Tanenbaum & Wetheral, op. cit.]

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

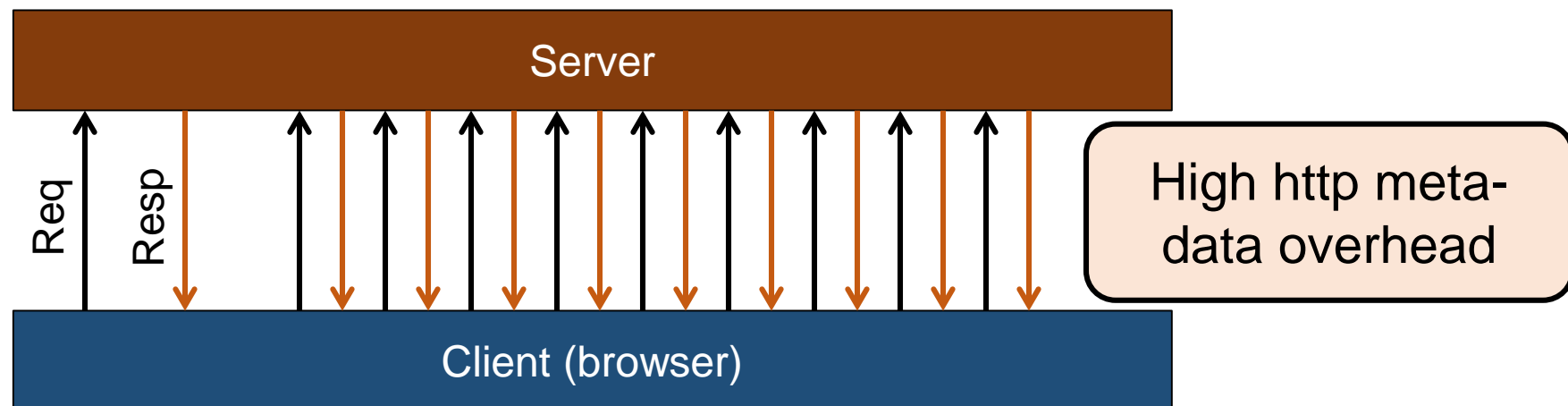


Example: *www.akamai.com*

- 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

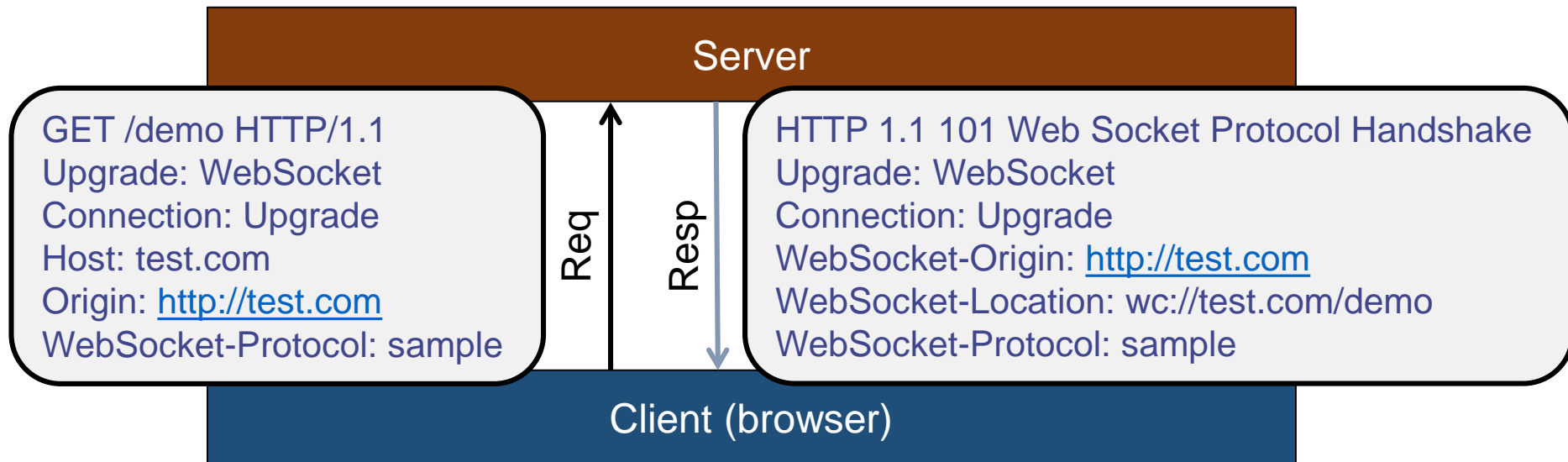
HTML5 WebSocket API

- 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“

WebSockets Problems

- Functionality at most of the underlying Transport protocol (e.g. TCP)
- Needs some application protocol on top
 - No means to negotiate such protocol
 - Client and server have to match...
- No connection reliability
 - No reconnect handling
 - No guaranteed message delivery (reliability in the app!)

Example usage: Javascript Client

```
var socket = new WebSocket(urlToWebsocketServer);
socket.onopen = function () {
    console.log("Connection created");
};
socket.onmessage = function (messageEvent) {
    console.log("New WS msg recv:" + messageEvent.data);
};
socket.onerror = function (errorEvent) {
    console.log("Error! Connection closed.");
};
socket.onclose = function (closeEvent) {
    console.log("Connection closed:" + closeEvent.code + "---" + closeEvent.reason);
};
```

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