INF3190 – Application Layer
DNS, Web, Mail

Carsten Griwodz
Email: griff@ifi.uio.no

Application layer

Introduction

What is it?

Internet view

- everything above the socket interface is application layer function

=> all functions of OSI layers 5 and 6 are Internet application layer

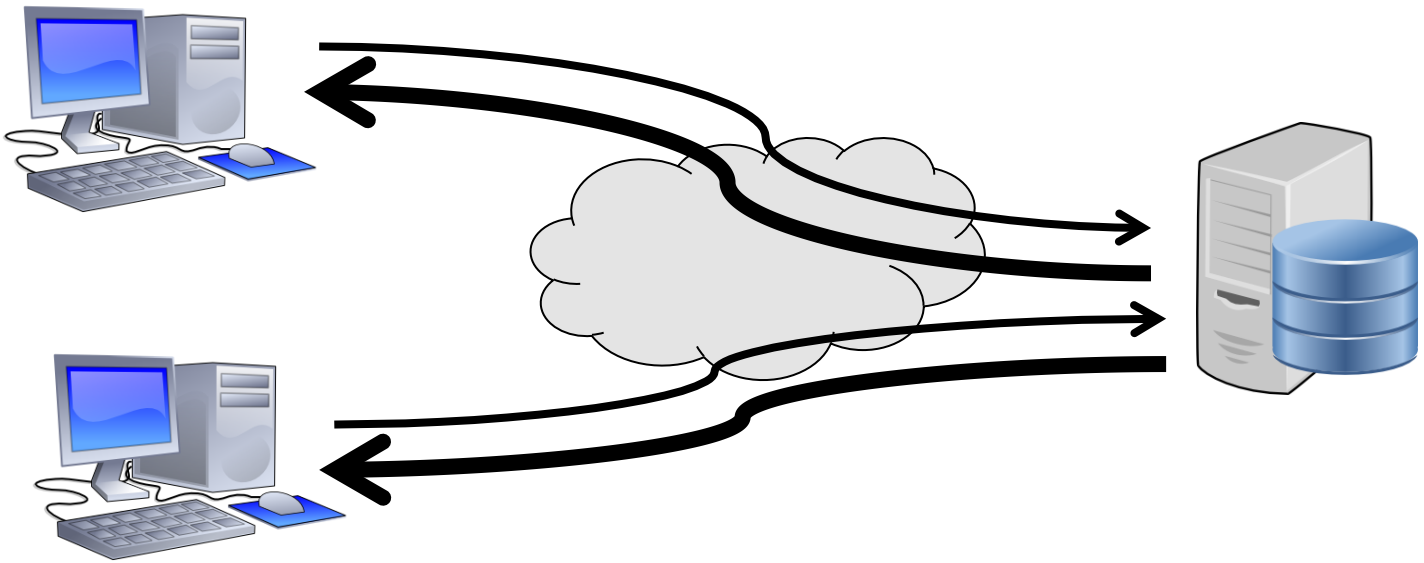
We still need (many of) those OSI functions

- long-term session maintenance, reconnections, session migration
- protocol translation
- today's the Internet world has protocols for this (official standards and de facto)
  - SMTP + (POP3 or IMAP)
  - HTTP, SHTTP, QUIC
  - (RTSP or SIP) + RTP/RTCP
  - MPEG DASH, Apple HLS, Microsoft Smooth Streaming
  - DCE / CORBA

Client-Server

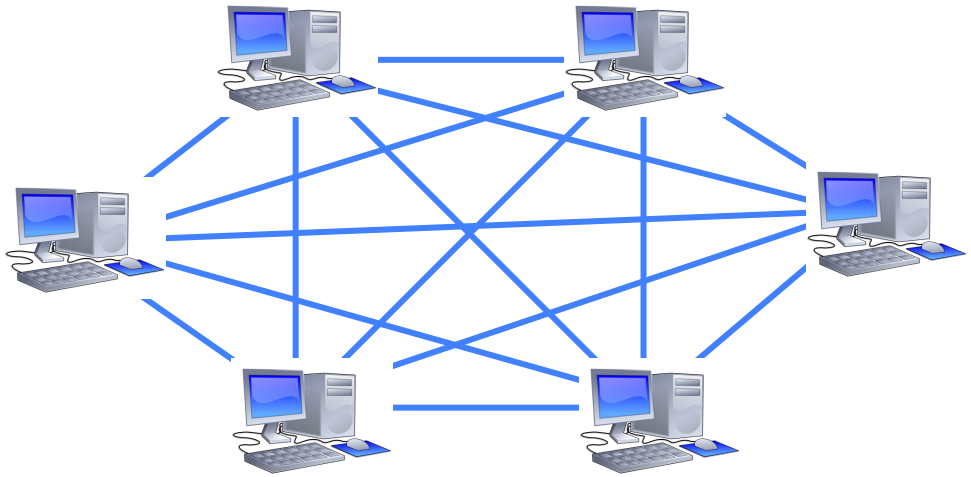
- Traditional communication model, easily comprehensible abstraction
  - Clients request service (initiate connection)
  - Servers provide service (answer requests)

- Examples: Web Client/Server, Mail Client/Server, FTP Client/Server



Peer-to-Peer

Recognized application-layer paradigm since 2000s



First clearly visible application: Napster

- file sharing (mostly for music)
- ruled illegal
- followed by others: Gnutella, Kazaa, BitTorrent, Freenet
- later picked up by research: CAN, Chord, Tapestry, Kademlia, Pastry
- idea: avoid control and/or censorship

Famous services

- video streaming: PPTV, P2PTV
- distributed computing: SITE@home

Old tech. that is like P2P but not recognized:

- Telephony
- Usenet news
- IP Routing

Actually, P2P = original Internet model

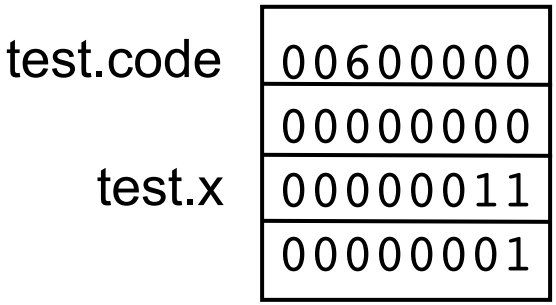
- all nodes are equal
- all nodes can address each other
- ownership is distributed

The presentation problem

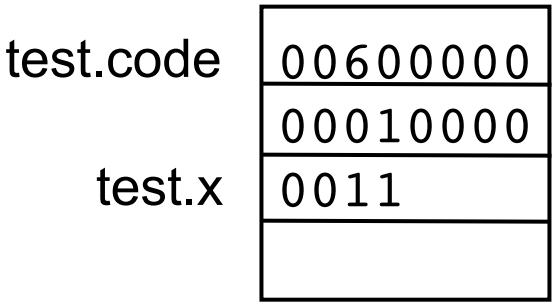
Q: Does perfect memory-to-memory copy solve “the communication problem”?

A: Not always!

```
struct Test
{
    char code;
    int x;
}

Test test;

test.x = 273;
test.code='a'
```



host 2 format
e.g. Intel DOS
not packed
little endian



host 2 format
e.g. ARM Linux
packet
big endian

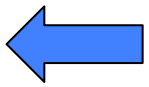
Problem: Different data format, storage conventions

Solving the presentation problem

1. Translate local-host format to host-independent format
2. Transmit data in host-independent format
3. Translate host-independent format to remote-host format

Old Style

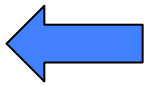
- cross-platform standardized binary encoding of data structures
  - OSI host-independent format: “Abstract Syntax Notation One” (ASN.1) defines “Basic Encoding Rules” (BER)
  - XDR: „external data representation“, belonged to NFS



- compensate platform differences
- assume single data interpretation
- space-saving

Current Style

- encoding everything as text
  - XML: „extensible markup language“



- convey data in platform-independent manner
- local styling and interpretation
- readable and debuggable

Application layer

DNS
Domain Name System

How to connect to a remote computer?

Connect to <hostname,port>

- e.g. `telnet 127.0.0.1 23`
  talking to my own machine
  obviously: used all the time, esp. since DHCP screws up your other addresses

- or `wget http://173.194.39.31:80/`
  talking to one of Google's machines
  possible to remember

- or `ssh 9.228.93.3`
  trying to talk to a desktop that had this address in 1995
  impossible to remember unless you've typed it 100 times a day

- If you want short names, write them into /etc/hosts.txt
- originally globally maintained by SRI, changes re-distributed by email and ftp (no more, ancient history)

How to connect to a remote computer?

Use "reasonable" names

- e.g.

```
ssh login.ifi.uio.no
wget www.google.com
```

- not only easier to remember
- reflect also organisation structures

- although the hierarchical structure may not fulfill all purposes
- somewhat related to physical network structure, at least locally

Domain Name System (DNS)

DNS at a High-Level

Domain Name System

Hierarchical namespace

As opposed to original, flat namespace

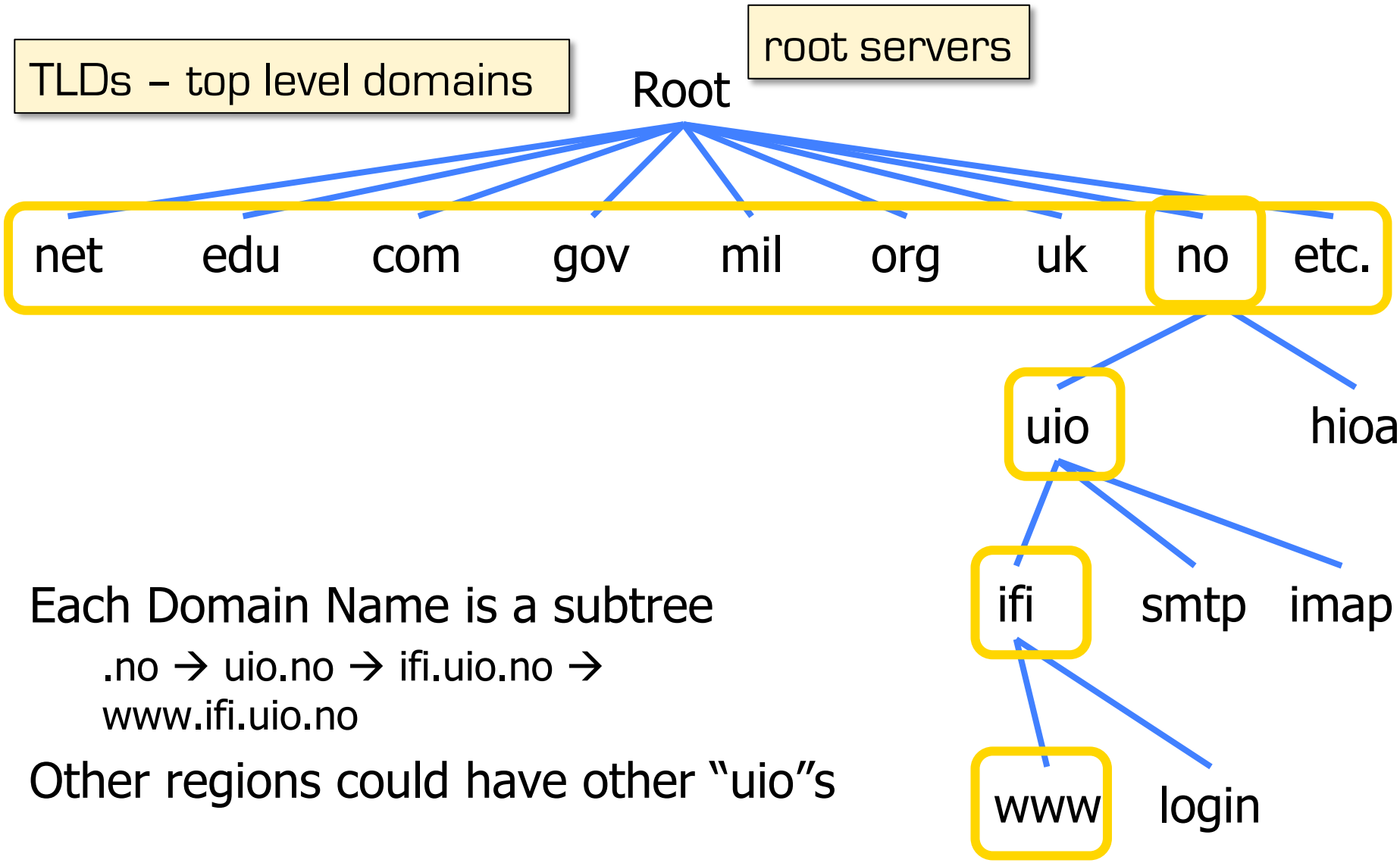
e.g. .com → google.com → mail.google.com

Distributed database

Simple client/server architecture

- UDP or TCP port 53
- servers must use TCP nowadays
- clients using TCP are mostly rejected
  - reduces server load
  - is a security problem

Naming Hierarchy

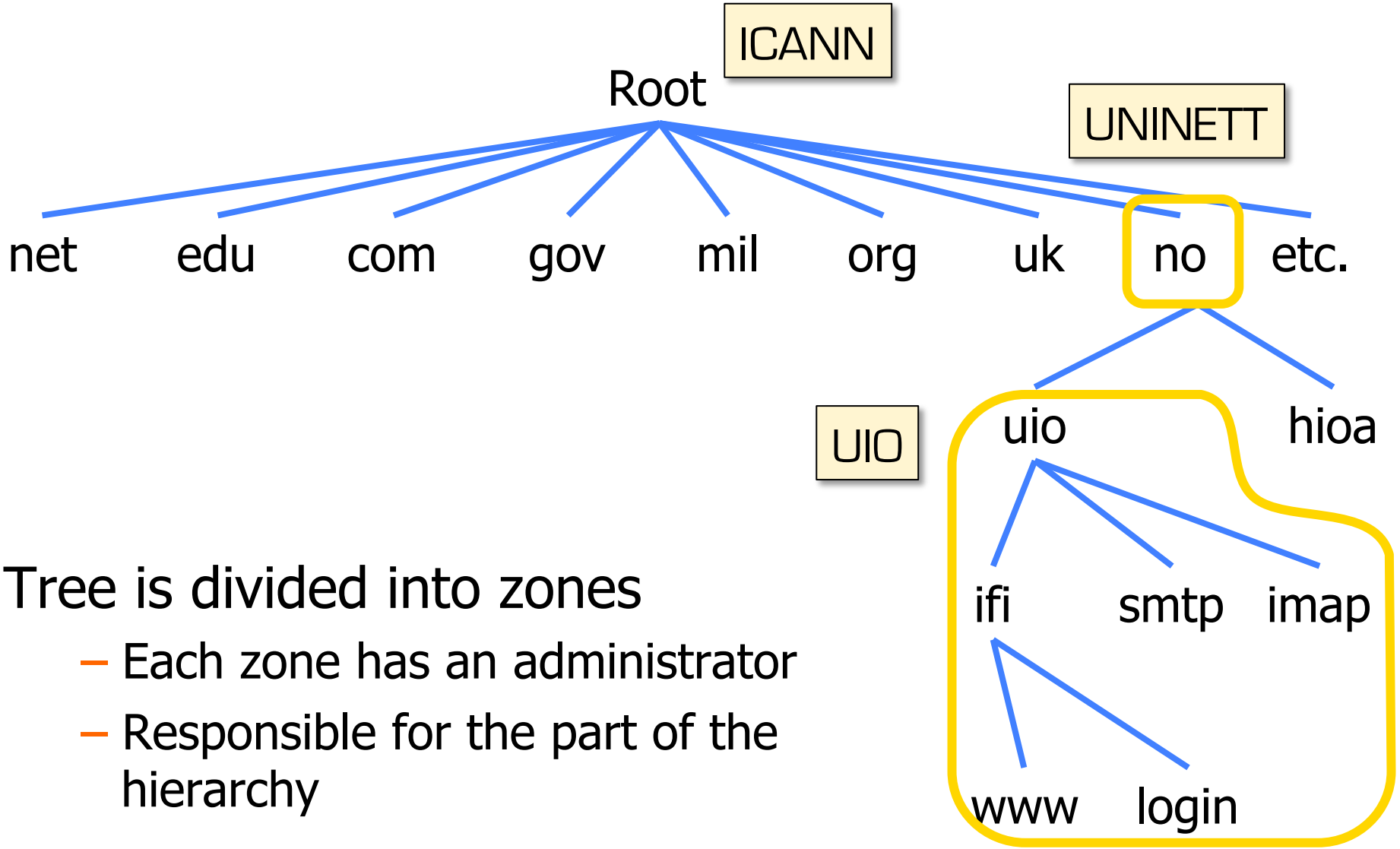


Each Domain Name is a subtree

.no → uio.no → ifi.uio.no → www.ifi.uio.no

Other regions could have other "uio"s

Hierarchical Administration



Tree is divided into zones

- Each zone has an administrator
- Responsible for the part of the hierarchy

Server Hierarchy

Functions of each DNS server

- Authority over a portion of the hierarchy
  - No need to store all DNS names
- Store all the records for hosts/domains in its zone
  - May be replicated for robustness
- Know the addresses of the root servers
  - Resolve queries for unknown names

Root servers know about all TLDs

Root Name Servers

Responsible for the Root Zone File

- Lists the TLDs and who controls them

```
com.            172800 IN     NS     a.gtld-servers.net.
com.            172800 IN     NS     b.gtld-servers.net.
com.            172800 IN     NS     c.gtld-servers.net.
```

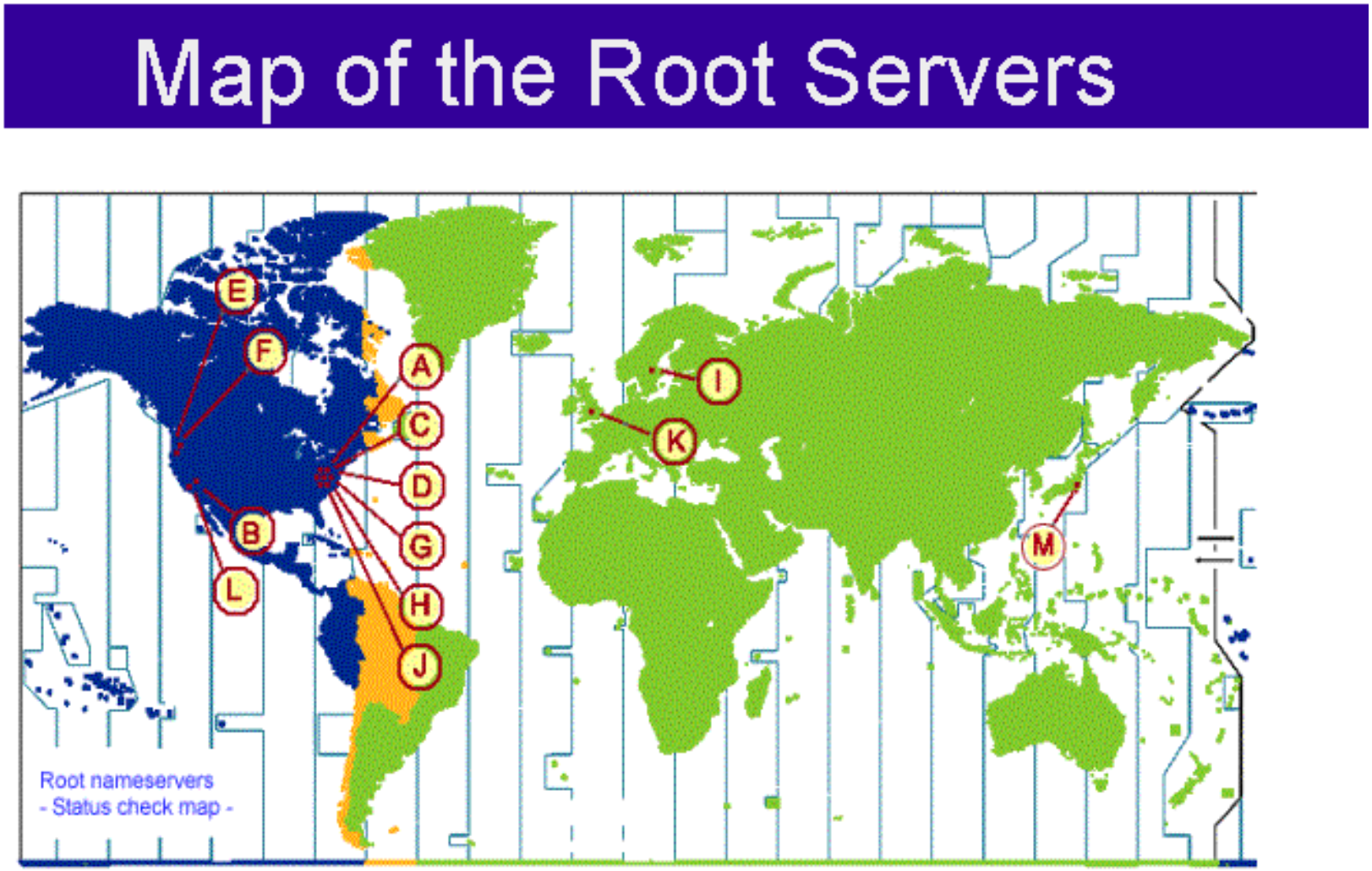
Administered by ICANN

- 13 root servers, labeled A→M
- 6 are anycasted, i.e. they are globally replicated

Contacted when names cannot be resolved

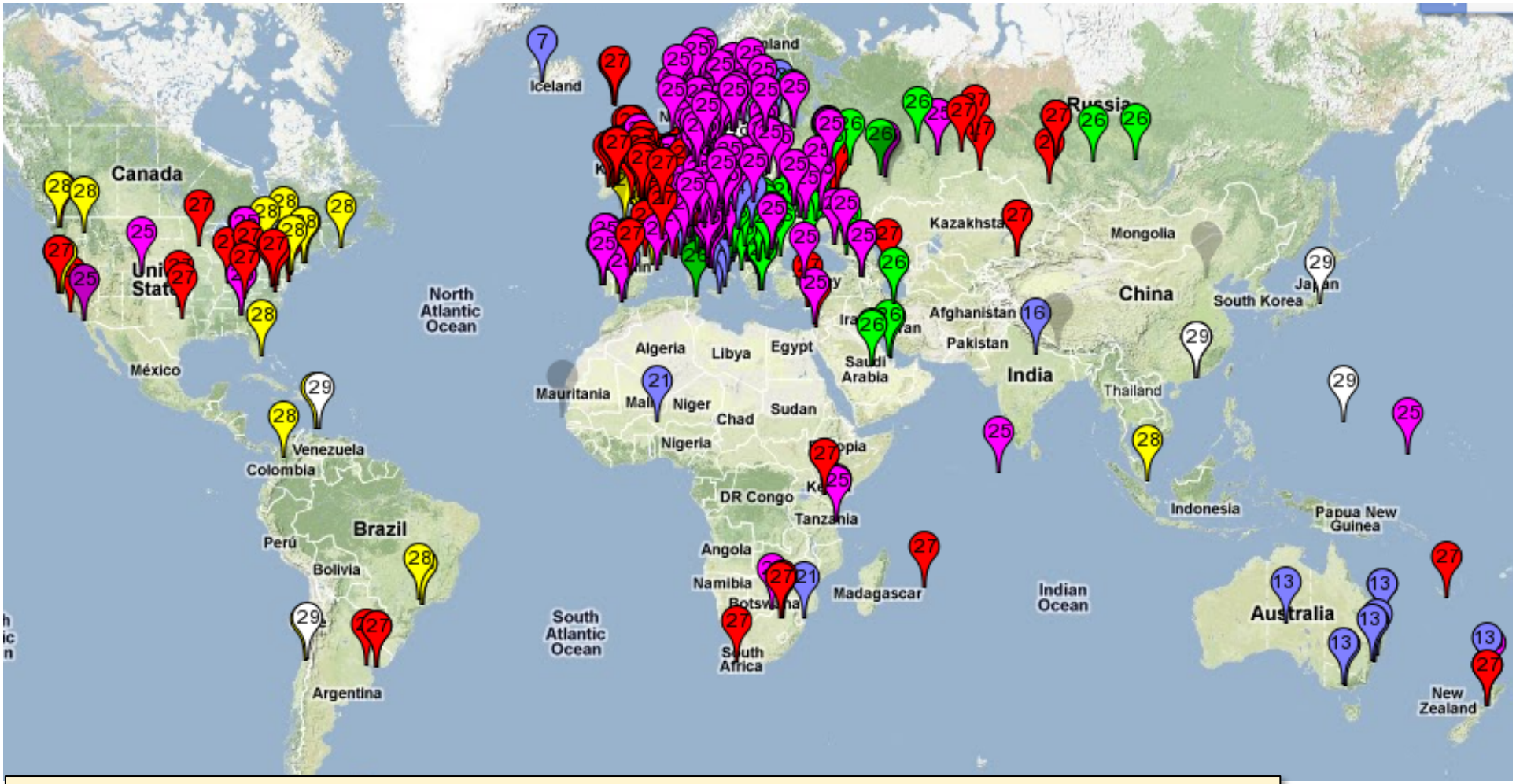
- In practice, most systems cache this information
- DDS attacks designed to reach root
- infrastructure bugs (e.g. old Telenor modems converted IPv6 lookup into broken IPv4 lookup)

ICAAN



from: http://www.icann.org/en/news/correspondence/roberts-testimony-14feb01-en.htm

Map of the Roots



k-root (Europe) is an anycast root node
This is RIPE's map of probing which of the 6 k-root copies get accessed

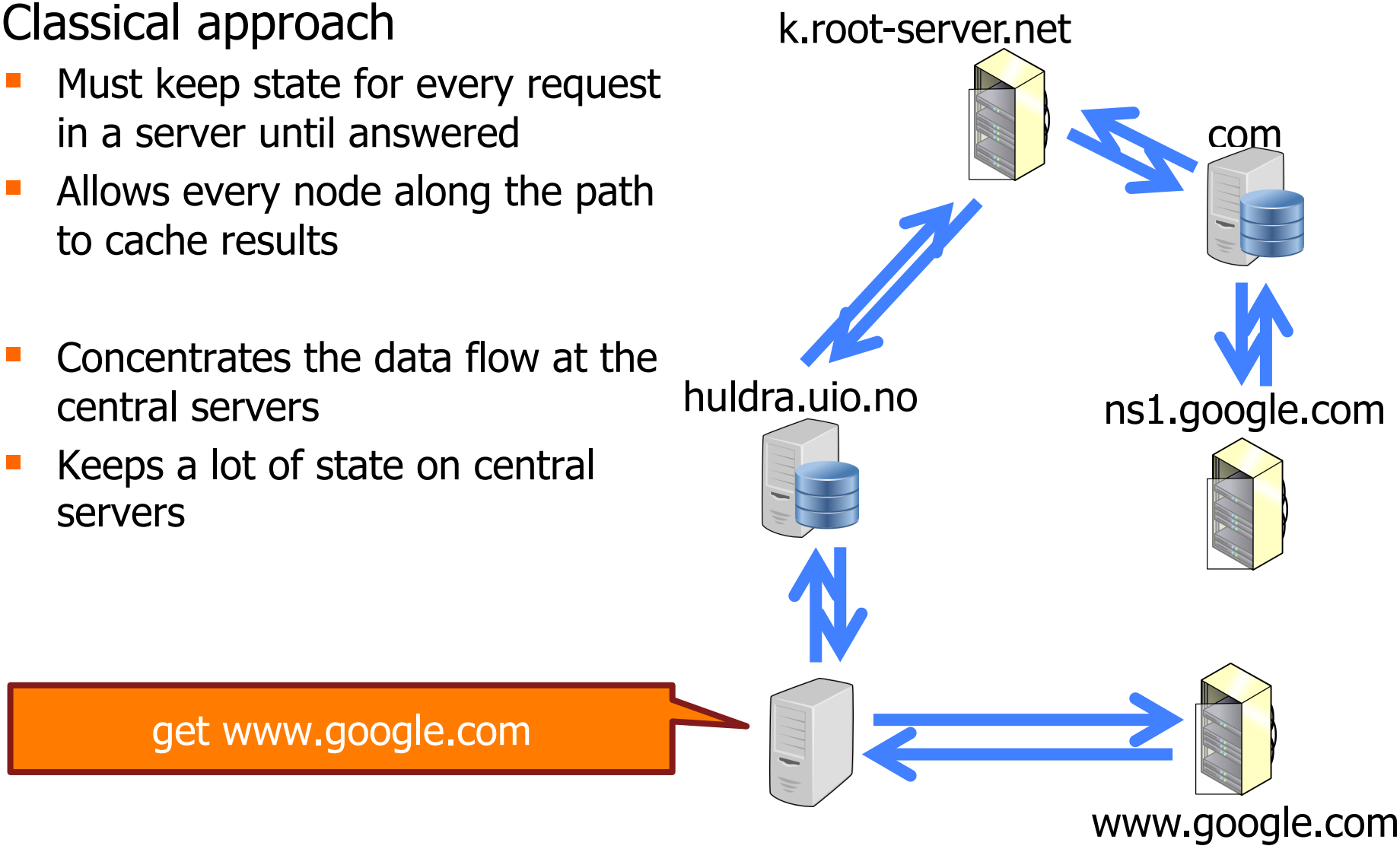
from https://labs.ripe.net/Members/kistel/dns-measurements-with-ripe-atlas-data

Recursive DNS Query

Classical approach

- Must keep state for every request in a server until answered
- Allows every node along the path to cache results

- Concentrates the data flow at the central servers
- Keeps a lot of state on central servers

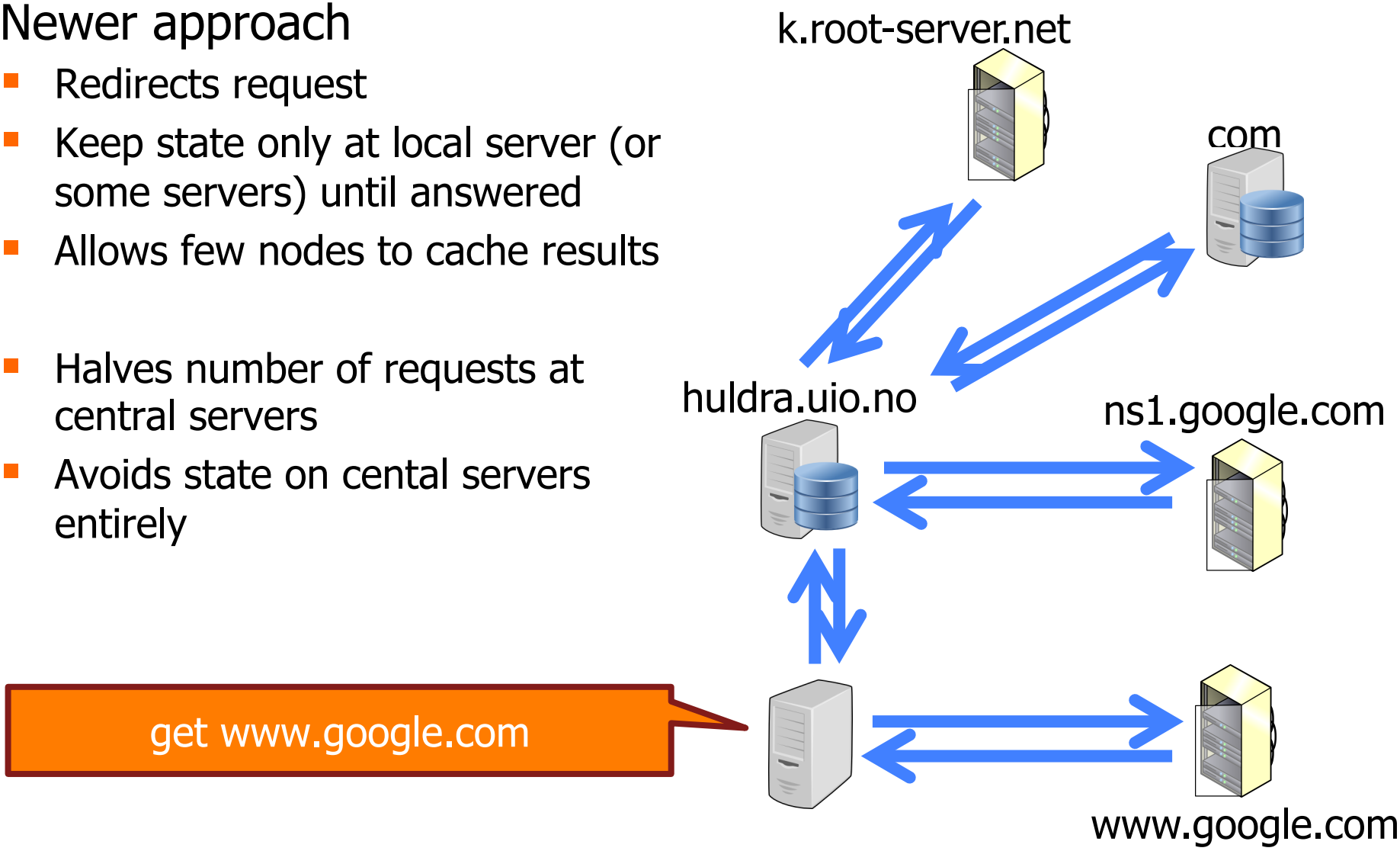


Iterated DNS Query

Newer approach

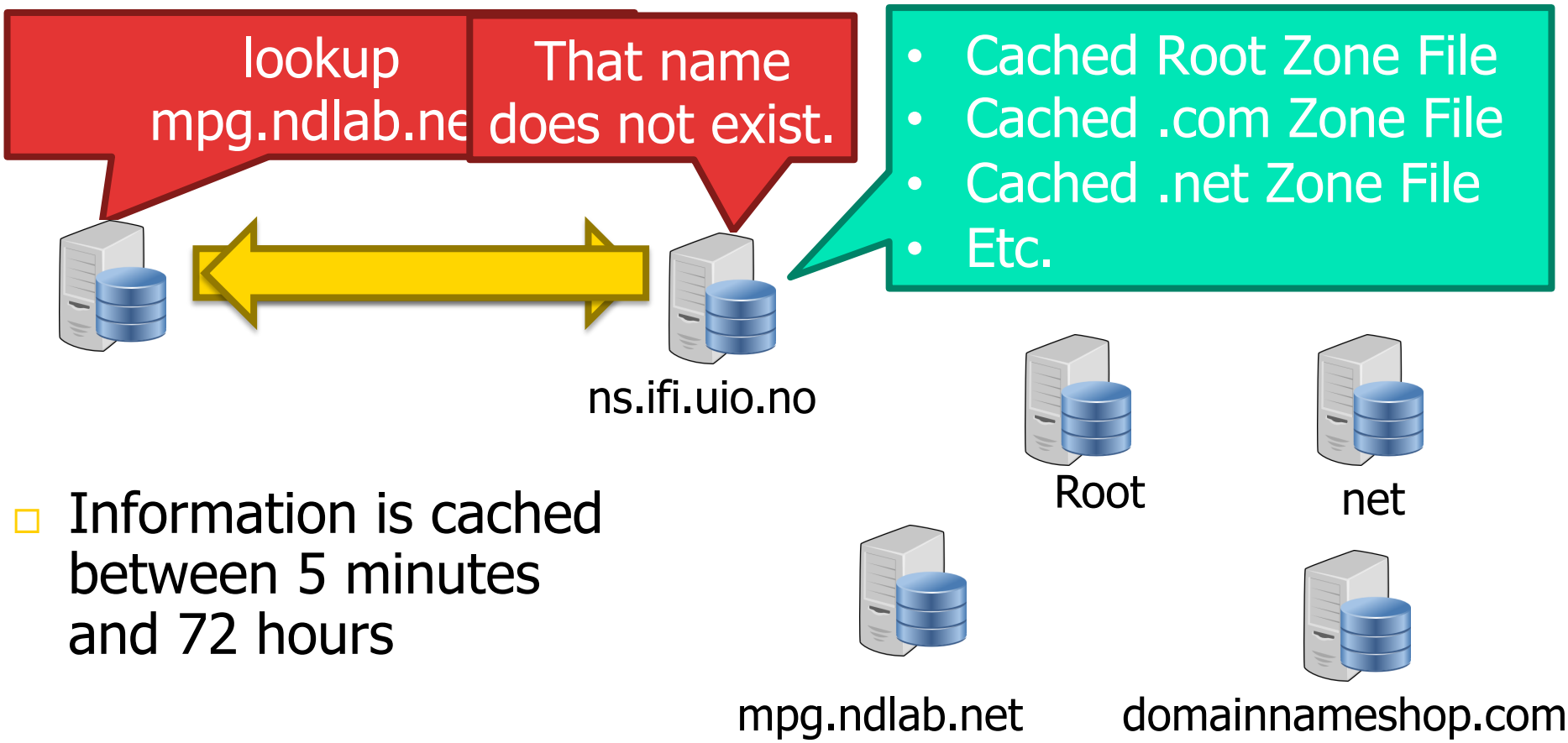
- Redirects request
- Keep state only at local server (or some servers) until answered
- Allows few nodes to cache results

- Halves number of requests at central servers
- Avoids state on cental servers entirely



Caching vs. Freshness

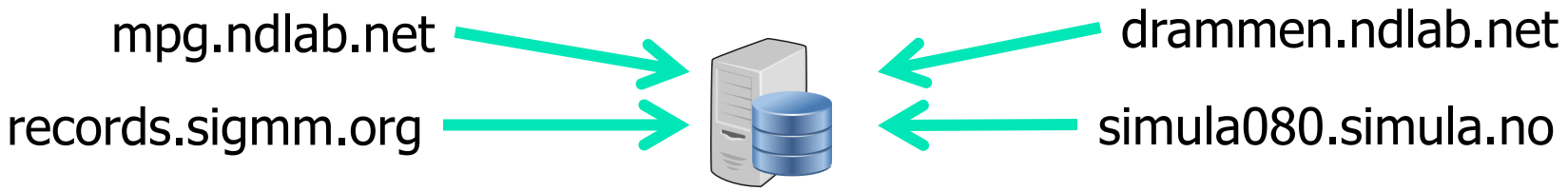
- Caching reduces DNS resolution latency
- Caching reduces server load
- Caching delays updates



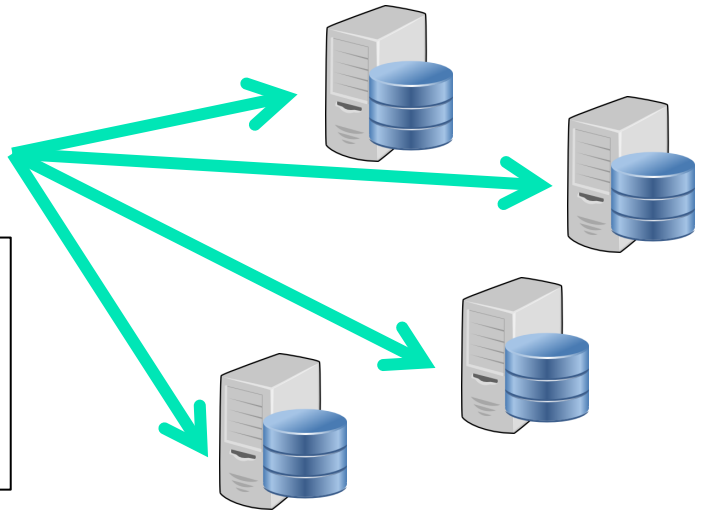
- Information is cached between 5 minutes and 72 hours

Aliasing and Load Balancing

One machine can have many aliases



One domain can map to multiple machines

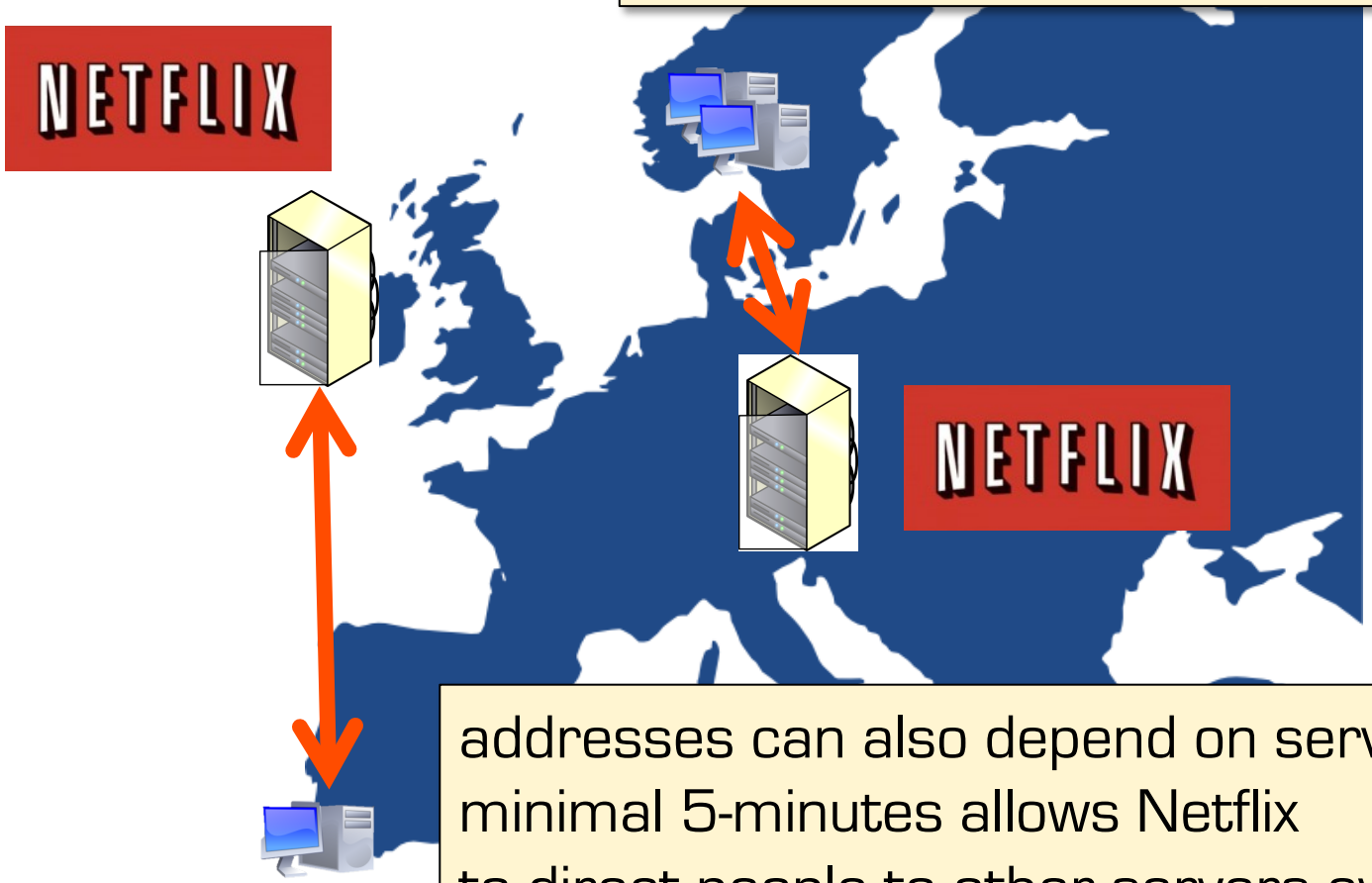


That includes k.root-server.net
and
login.ifi.uio.no

Content Delivery Networks

DNS allows zoning

e.g. Netflix (and Google) addresses depend on the origin of your connection
geography, ISP, ...

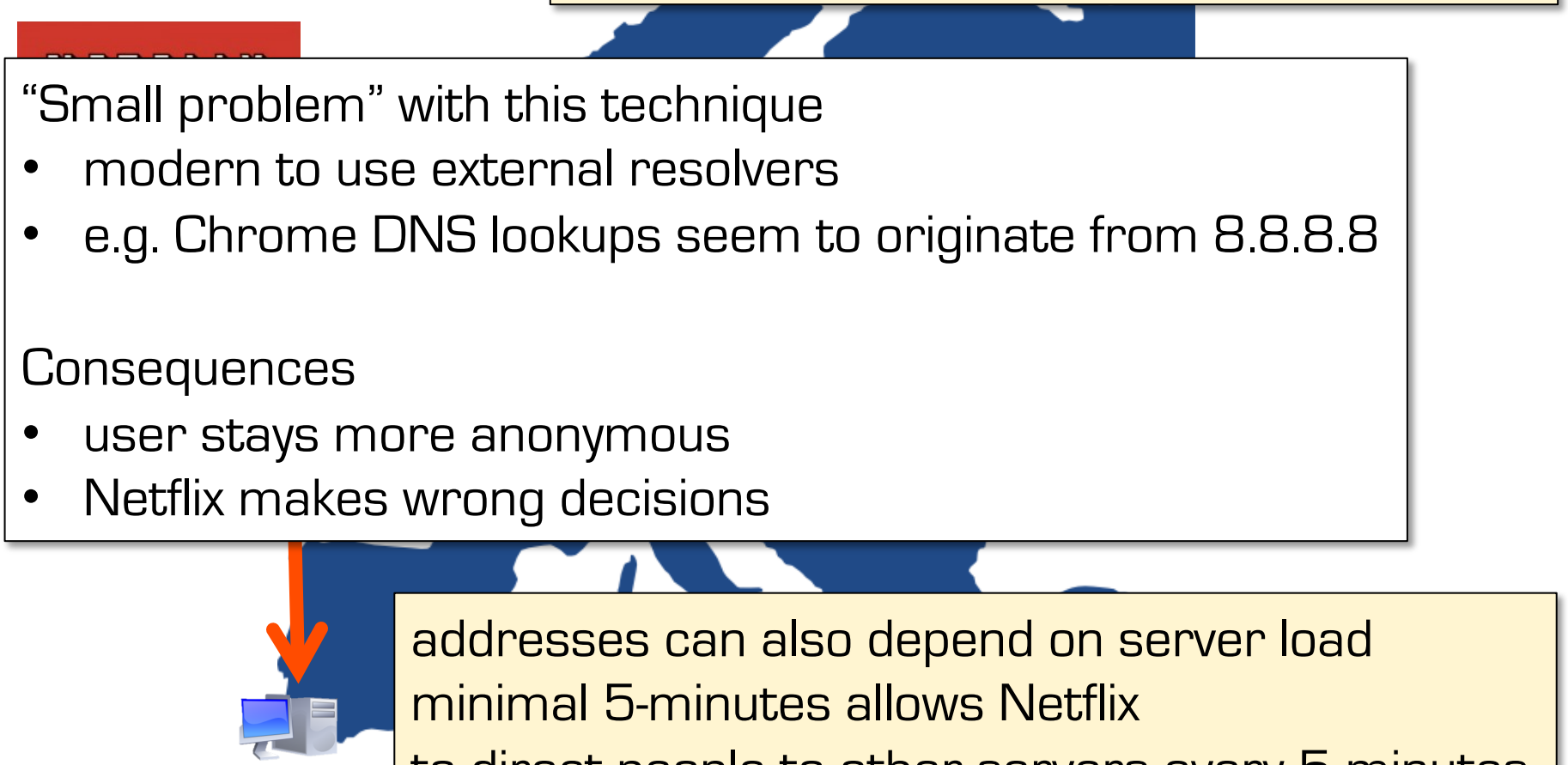


addresses can also depend on server load
minimal 5-minutes allows Netflix
to direct people to other servers every 5 minutes

Content Delivery Networks

DNS allows zoning

e.g. Netflix (and Google) addresses depend on the origin of your connection
geography, ISP, ...



"Small problem" with this technique

- modern to use external resolvers
- e.g. Chrome DNS lookups seem to originate from 8.8.8.8

Consequences

- user stays more anonymous
- Netflix makes wrong decisions

addresses can also depend on server load
minimal 5-minutes allows Netflix
to direct people to other servers every 5 minutes

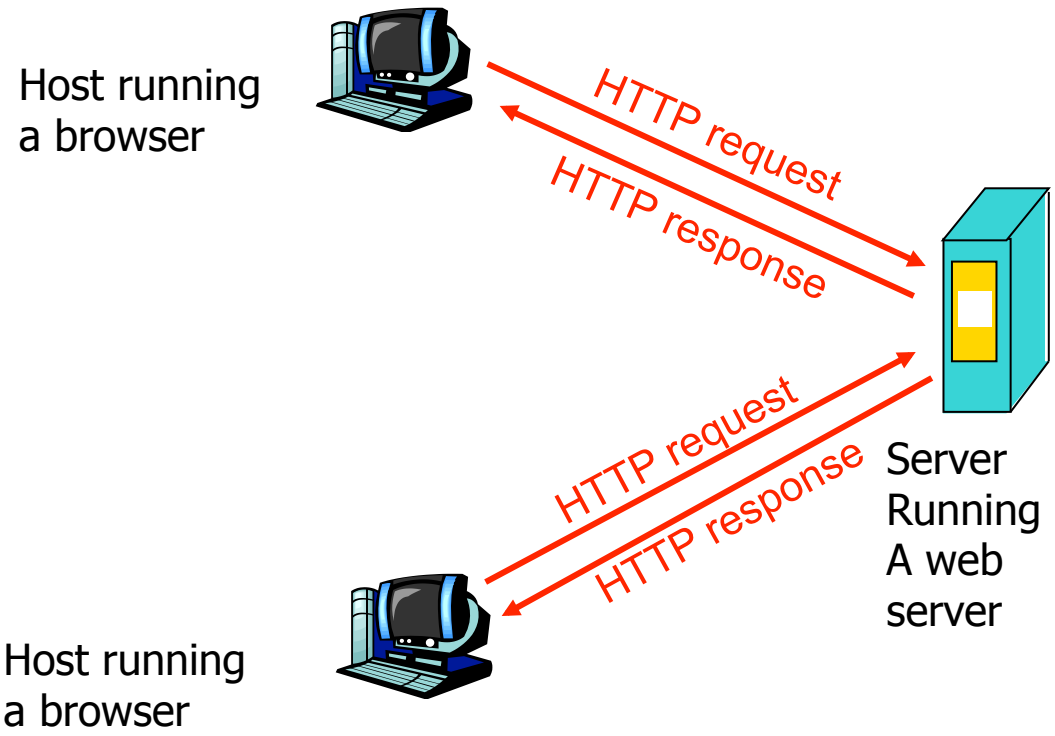
Application layer

HTTP
Hypertext Transfer Protocol

The Web: the HTTP protocol

HTTP: hypertext transfer protocol

- 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
- Two major versions:
  HTTP 1.0, HTTP 1.1



The HTTP protocol

HTTP: TCP transport service:

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

HTTP is “stateless”

- server maintains no information about past client requests

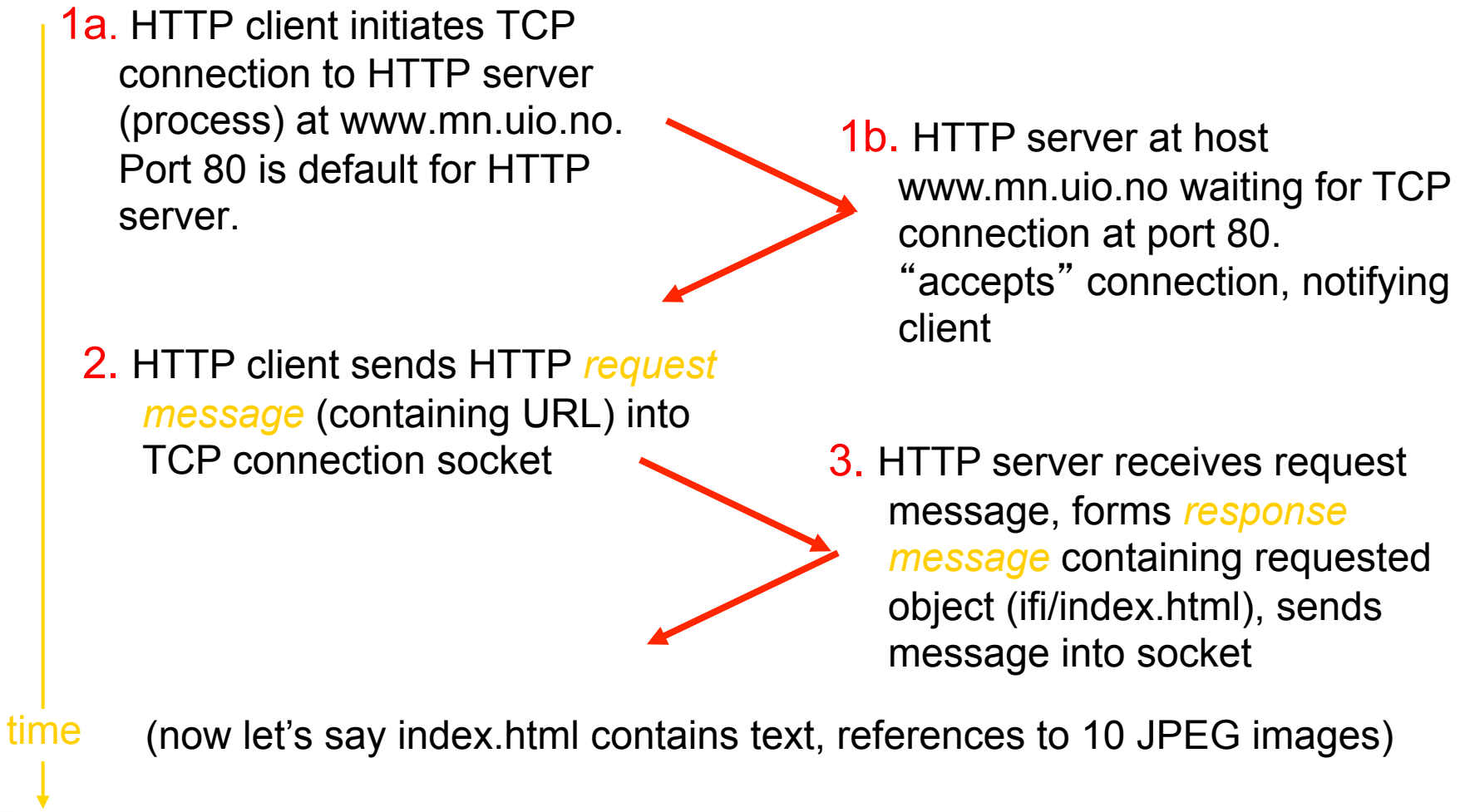
aside

Protocols that maintain “state” are complex!

- past history (state) must be maintained
- if server/client crashes, their views of “state” may be inconsistent, must be reconciled

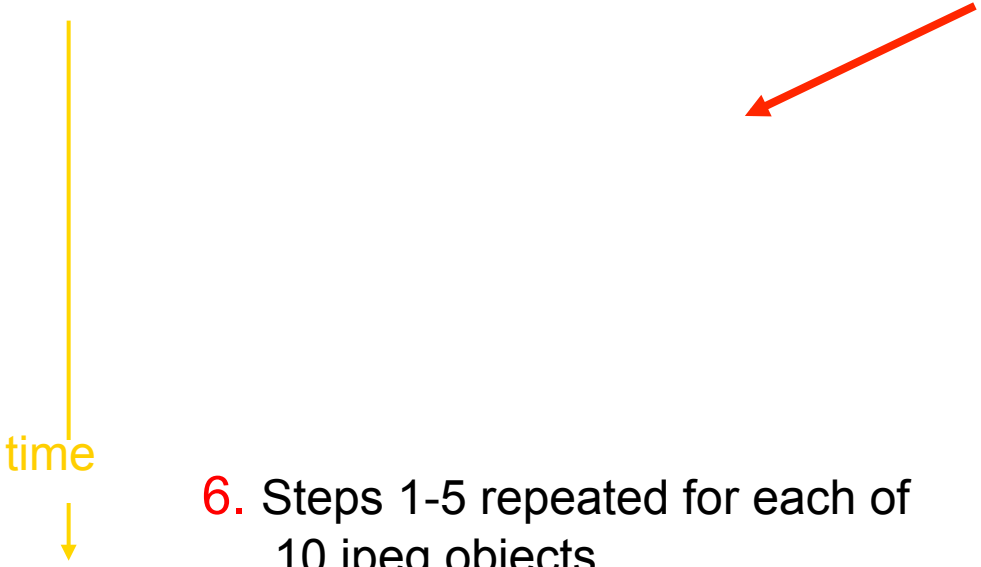
HTTP example

Suppose user enters URL www.mn.uio.no/ifi/index.html



HTTP example (cont.)

5. HTTP client receives response message containing HTML file, displays HTML. Parsing HTML file, finds 10 referenced JPEG objects



4. HTTP server closes TCP connection.

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

Non-persistent, persistent connections

Non-persistent

- HTTP/1.0: server parses request, responds, closes TCP connection
- 2 RTTs to fetch object
  - TCP connection
  - object request/transfer
- each transfer suffers from TCP's initially slow sending rate
- many browsers open multiple parallel connections

Persistent

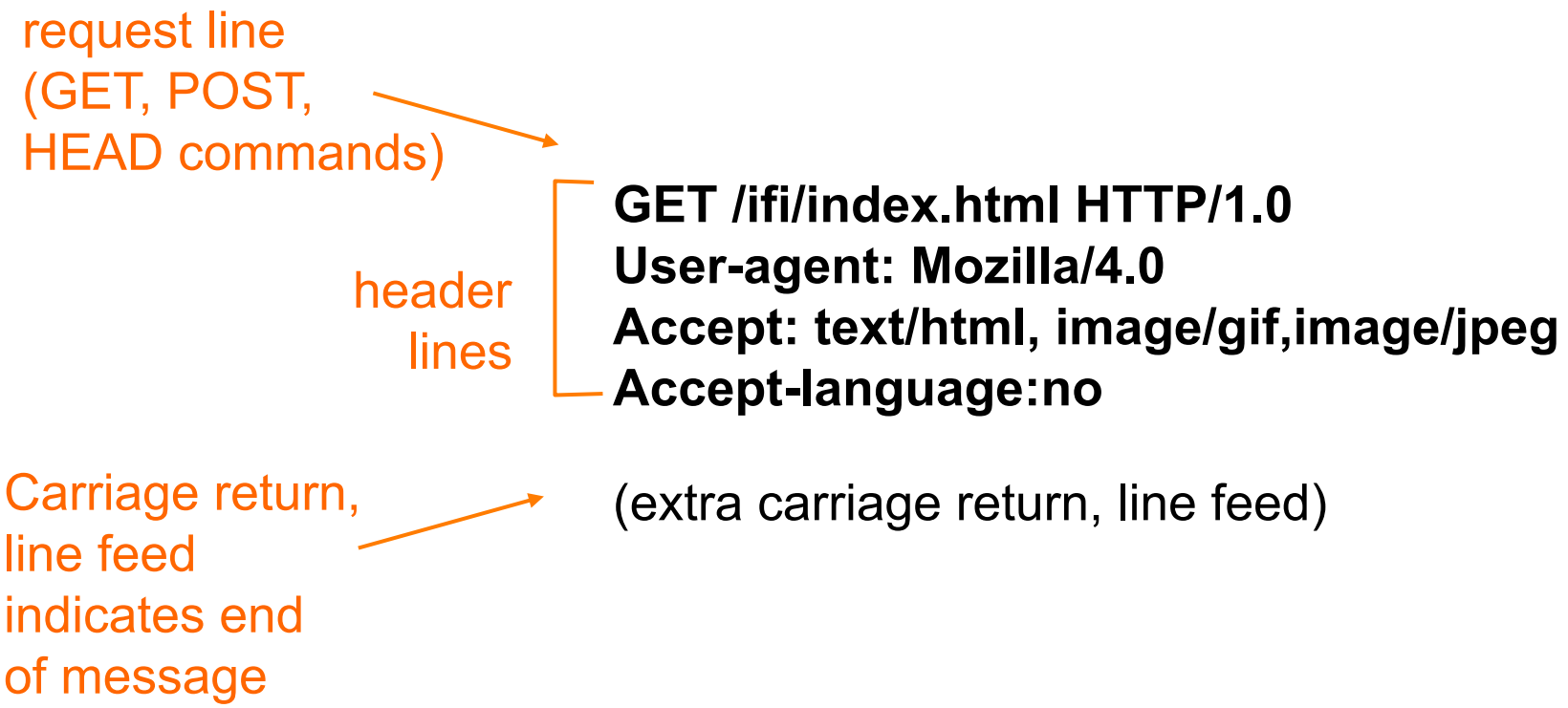
- default for HTTP/1.1
- on same TCP connection: server, parses request, responds, parses new request,..
- client sends requests for all referenced objects as soon as it receives base HTML.
- fewer RTTs, less slow start.

Persistent with pipelining

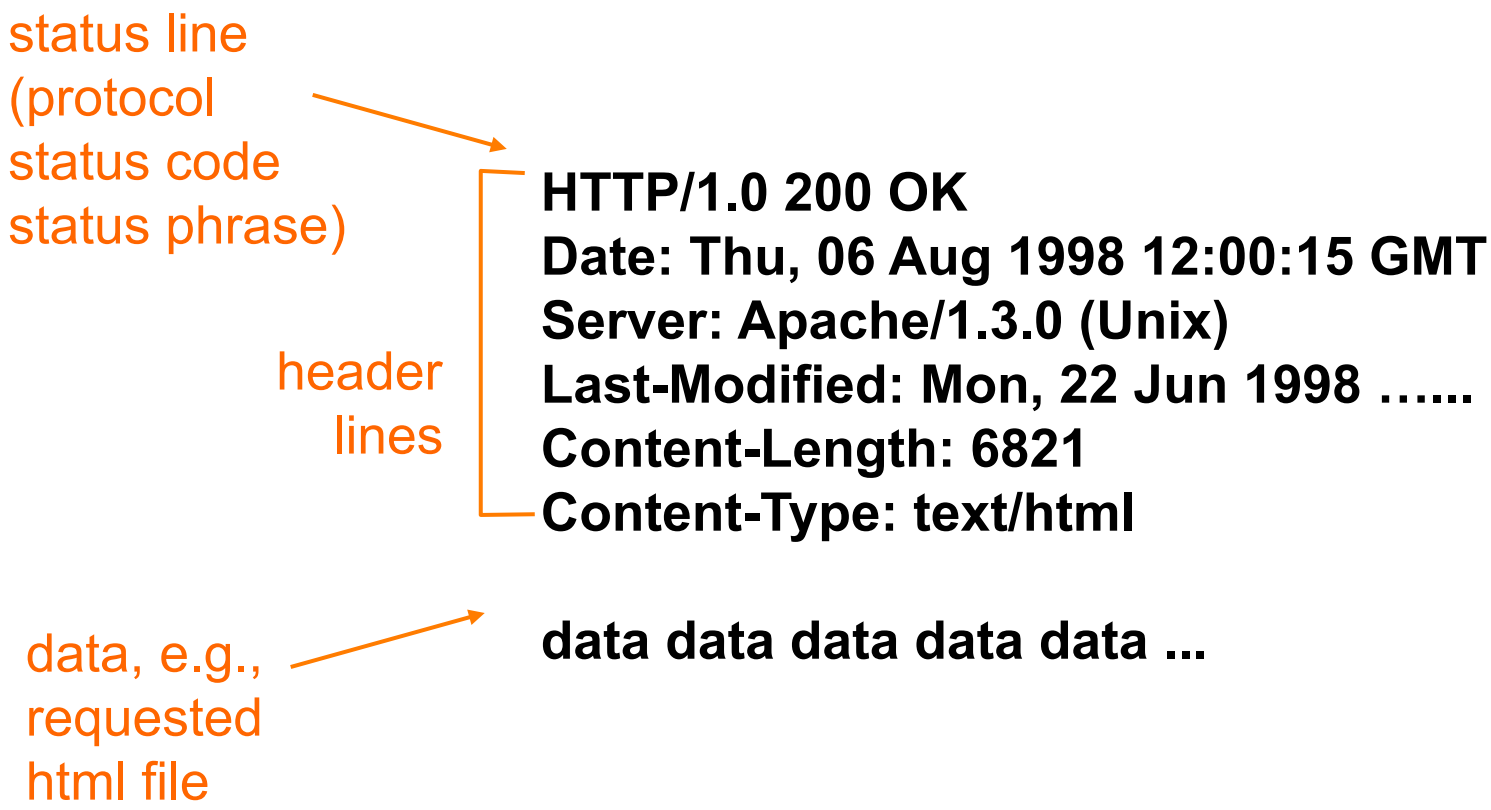
- request multiple objects in one go (even fewer RTTs)
- answers arrive one after each other in order of requests

HTTP message format: request

- two types of HTTP messages: *request*, *response*
- HTTP request message:
  - ASCII (human-readable format)



HTTP message format: response



HTTP response status code examples

200 OK

- request succeeded, requested object later in this message

301 Moved Permanently

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

400 Bad Request

- request message not understood by server

404 Not Found

- requested document not found on this server

505 HTTP Version Not Supported

Trying out HTTP (client side) for yourself

1. Telnet to your favorite Web server:

telnet www.aftenposten.no 80

Opens TCP connection to port 80 (default HTTP server port) at www.aftenposten.no. Anything typed in will be sent via this connection.

2. Type in a GET request:

GET / HTTP/1.1

By typing this in (hit carriage return once), you send this minimal (but complete) GET request for the root document to the HTTP server

3. Be fast now: type in the host header:

Host: www.aftenposten.no

Servers can be multi-homed (multiple different web sites on physical server), and so the client must specify which host it wants. Else, a server would often return an error message.

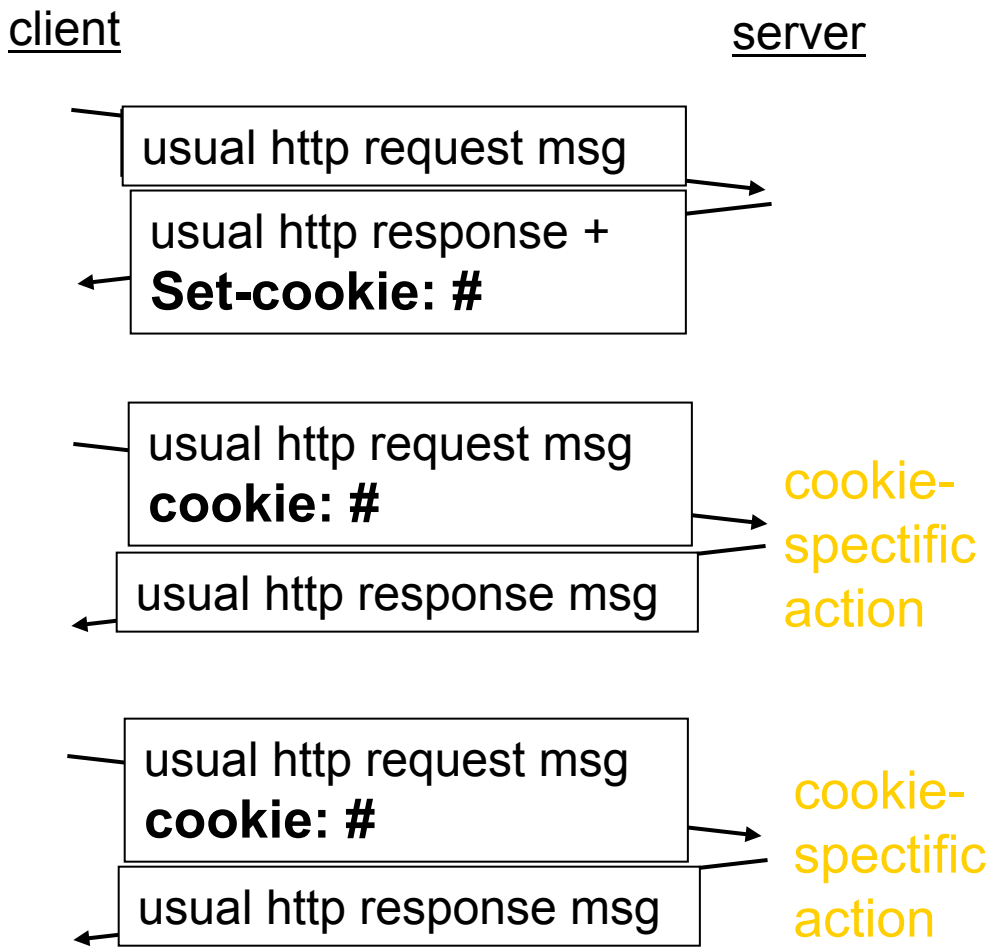
4. Hit carriage return twice and see the result

Cookies: keeping "state"

- server-generated # , server-remembered #, later used for:
  - authentication
  - remembering user preferences, previous choices
- server sends "cookie" to client in response msg

  Set-cookie: 1678453
- client presents cookie in later requests

  cookie: 1678453

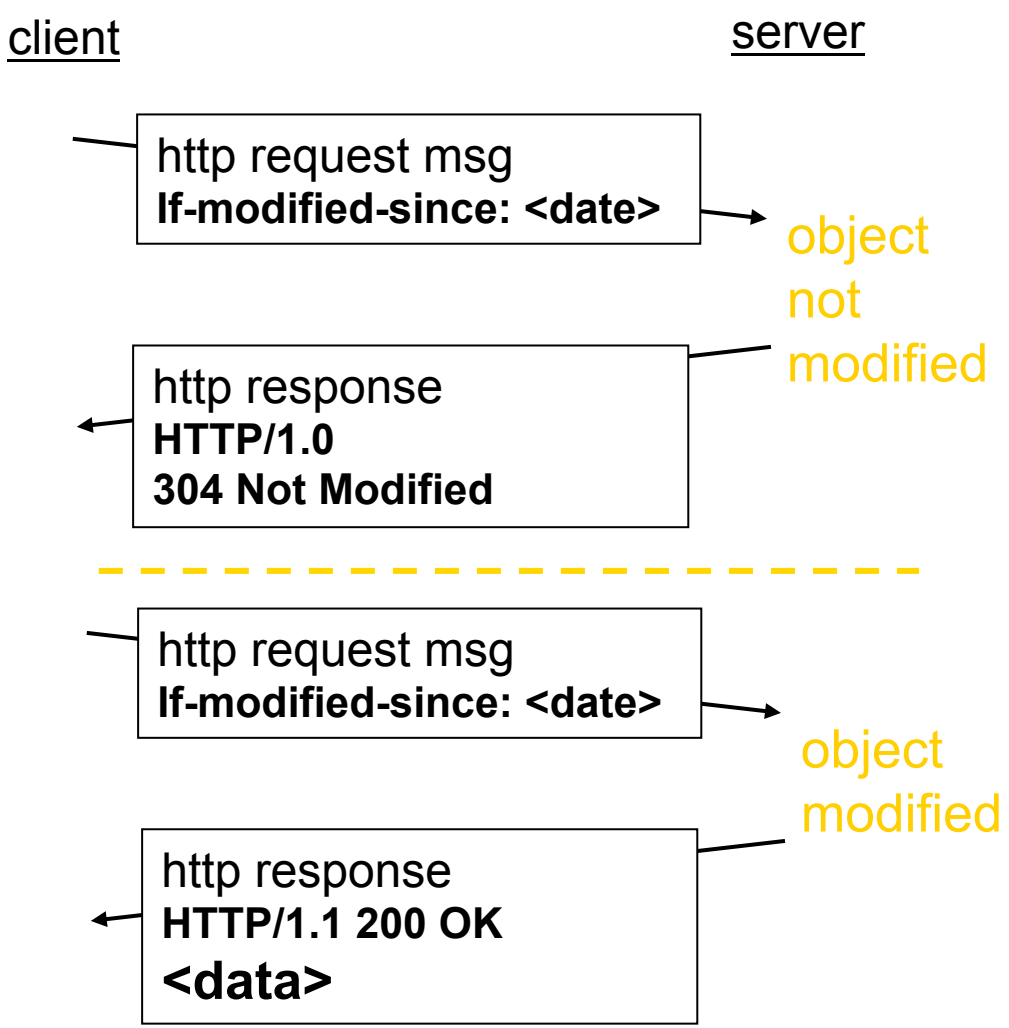


Conditional GET: client-side caching

- Goal: don't send object if client has up-to-date cached version
- client: specify date of cached copy in http request

  If-modified-since: <date>
- server: response contains no object if cached copy is up-to-date:

  HTTP/1.0 304 Not Modified

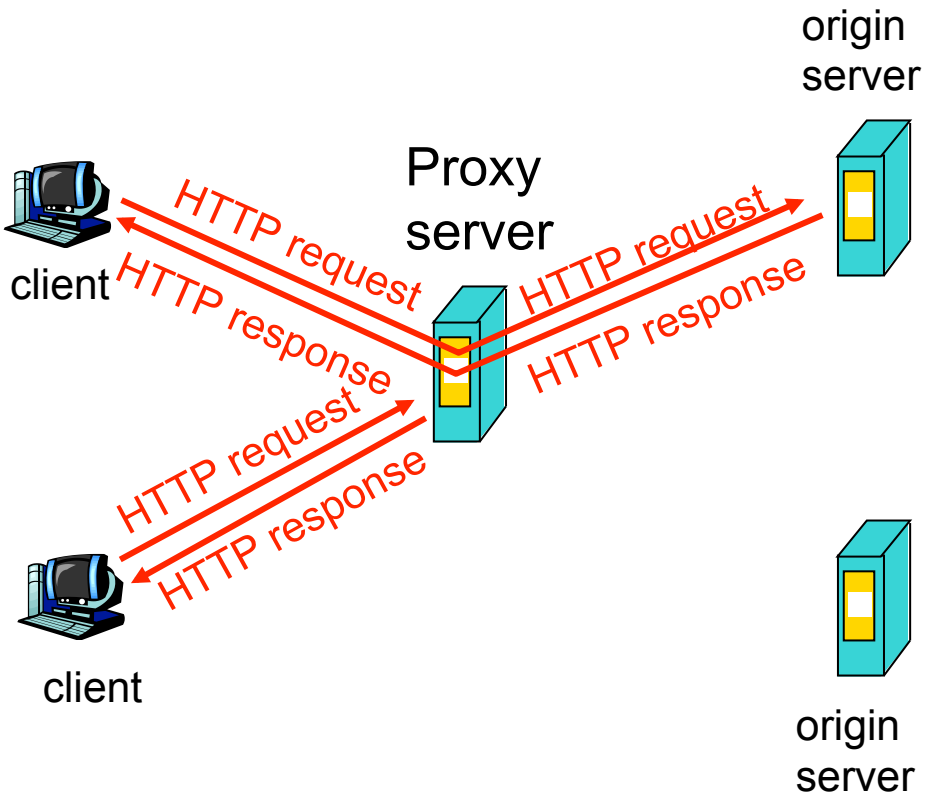


Web Caches (proxy server)

Goal: satisfy client request without involving origin server

- user sets browser: Web accesses via web cache

- client sends all HTTP requests to web cache
  - object in web cache: web cache returns object
  - else web cache requests object from origin server, then returns object to client



Assumption: cache is closer to client (e.g. same network) => faster, less “long-distance” traffic

Application layer

SMTP and MIME

Simple mail transfer protocol

Multipurpose Internet mail extensions

Electronic Mail

- Major components
  - "mail clients"
    Message User Agents (MUAs)
  - "mail servers"
    Message Submission / Transfer / Delivery Agents (MSA, MTA, MDA)
    - often realized as one component called
      Message Handling Service (MHS)

- MUA
  - a.k.a. "mail reader"
  - composing, editing, reading mail messages
  - outgoing, incoming messages stored on server

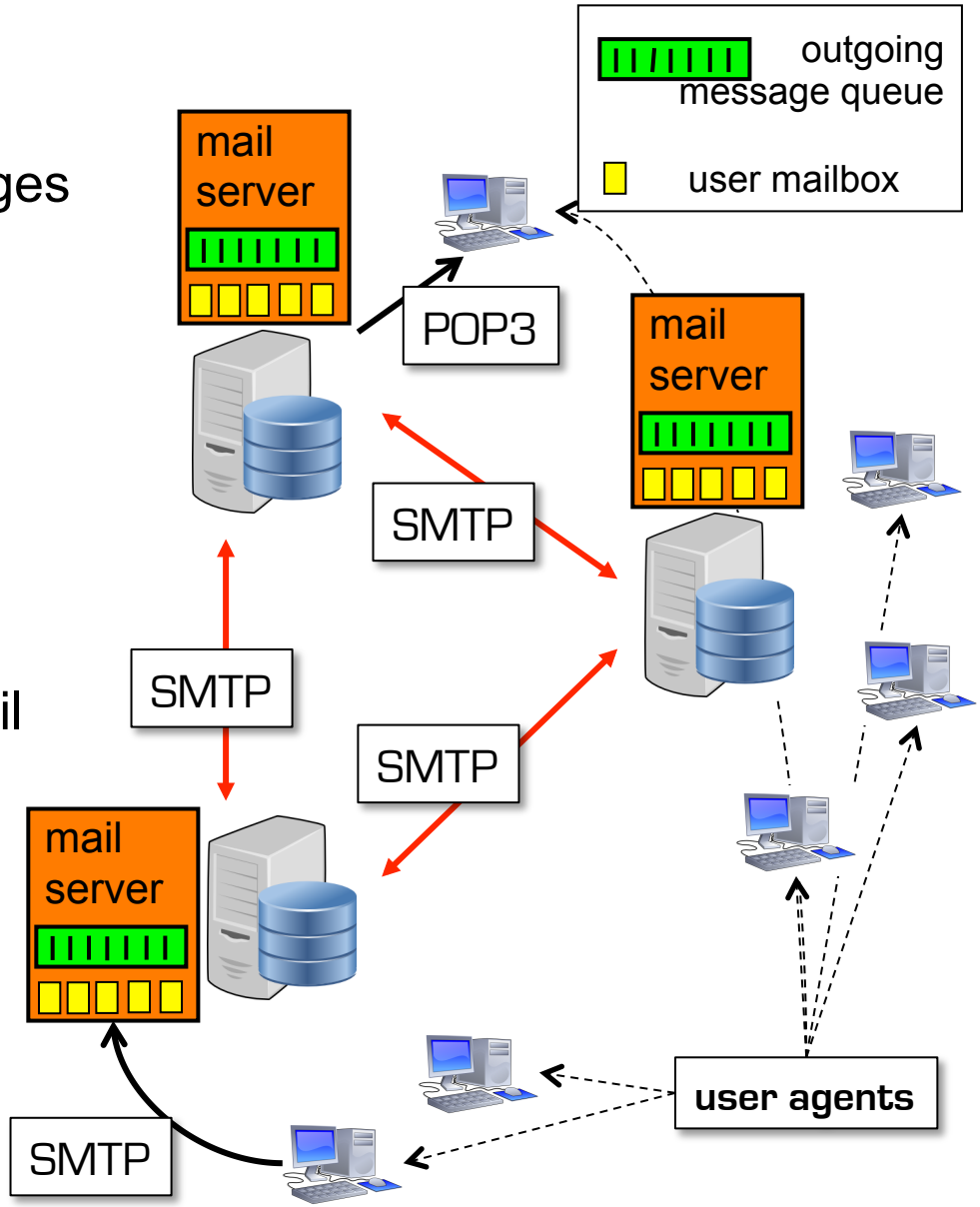
Electronic Mail: mail servers

Mail Servers

- mailbox contains incoming messages (yet to be read) for user
- message queue of outgoing (to be sent) mail messages

Simple Mail Transfer Protocol (SMTP)

- between mail servers to send email messages
- client: sending mail server
- server: receiving mail server



Electronic Mail: 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
  - handshaking (greeting)
  - transfer of messages
  - closure

- command/response interaction
  - commands: ASCII text
  - response: status code and phrase

- messages must be in 7-bit ASCII

Sample SMTP interaction

```
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
```

Handmade SMTP

```
telnet servername 25
```

see 220 reply from server

enter HELO, MAIL FROM, RCPT TO, DATA, QUIT commands

above lets you send email without using email client (reader)

SMTP: final words

SMTP uses persistent connections

SMTP requires message (header & body) to be in 7-bit ASCII

Certain character strings not permitted in msg (e.g., `CRLF.CRLF`). Thus msg has to be encoded (usually into either base-64 or quoted printable)

SMTP server uses `CRLF.CRLF` to determine end of message

Comparison with HTTP:

- HTTP: pull
- STMP: push
  (until final server!)
- both have ASCII command/ response interaction, status codes
- HTTP
  - each object encapsulated in its own response msg
- SMTP
  - originally the same
  - now: multiple objects sent in multipart msg

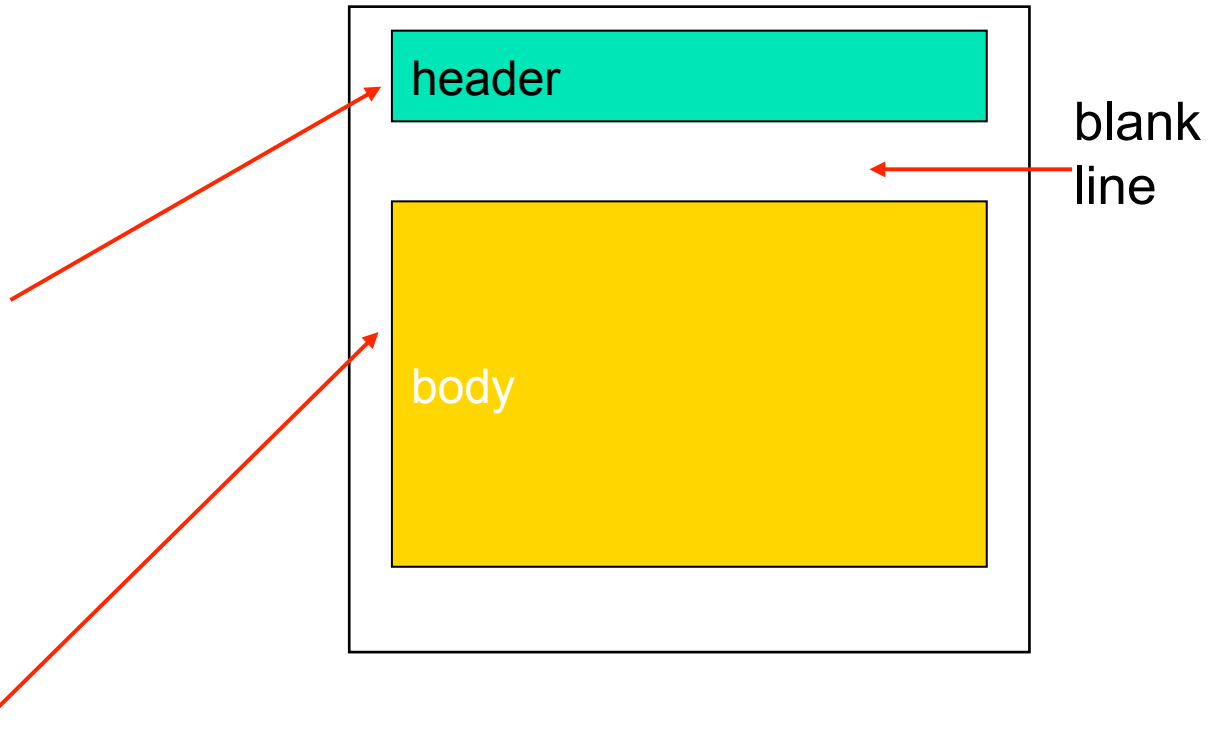
Mail message format

SMTP: protocol for exchanging email msgs

Standard for text message format:

- header lines, e.g.,
  - To:
  - From:
  - Subject:

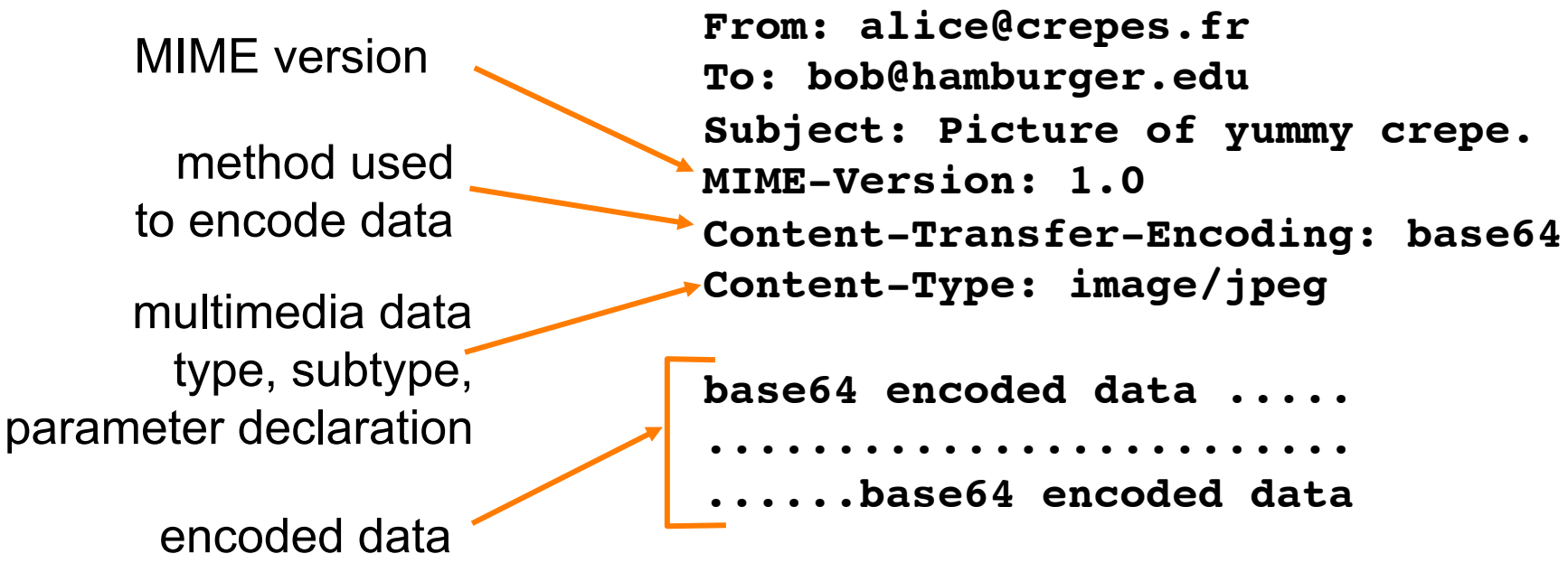
  different from SMTP commands!
- body
  - the "message", ASCII characters only



Message format: multimedia extensions

MIME: multipurpose Internet mail extension

additional lines in msg header declare MIME content type



"classical" mail may indicate:
Content-type: text/ascii
but 7-bit ASCII text is still the default

MIME types

`Content-Type: type/subtype; parameters`

Text

- example subtypes: **plain, html**

Image

- example subtypes: **jpeg, gif**

Audio

- exampe subtypes: **basic** (8-bit mu-law encoded), **32kadpcm** (32 kbps coding)

Video

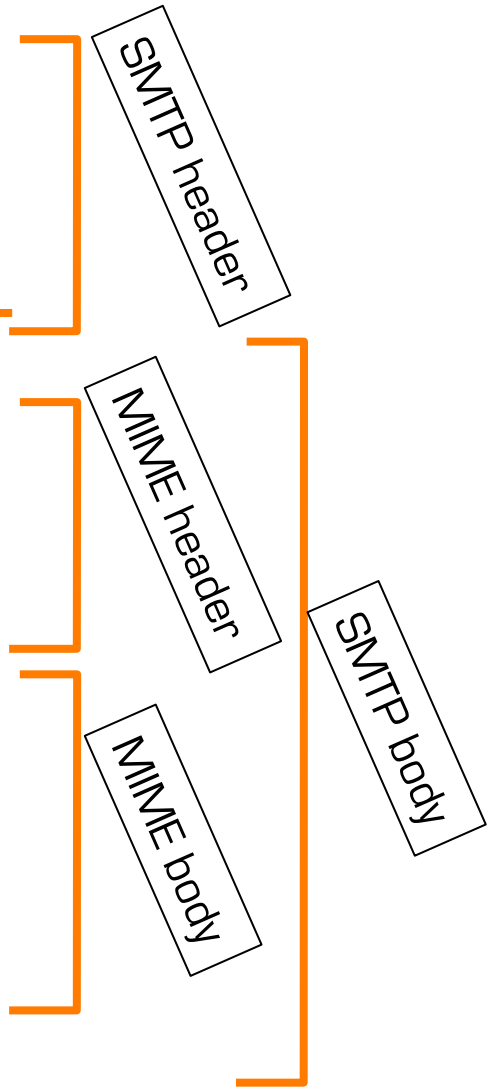
- example subtypes: **mpeg, quicktime**

Application

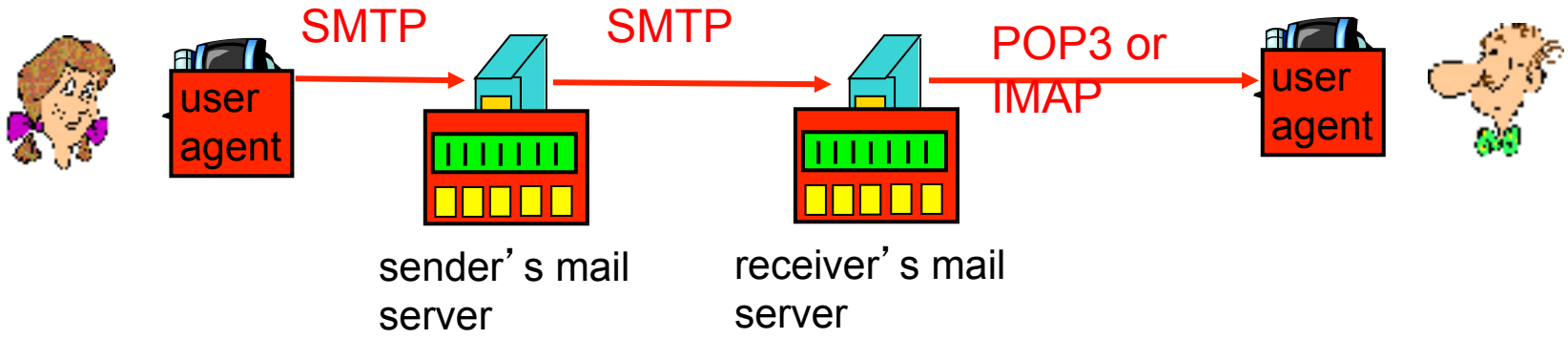
- other data that must be processed by reader before "viewable"
- example subtypes: **msword, octet-stream**

Multipart Type

```
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Picture of yummy crepe.
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary=98766789

--98766789
Content-Transfer-Encoding: quoted-printable
Content-Type: text/plain

Dear Bob,
Please find a picture of a crepe.
--98766789
Content-Transfer-Encoding: base64
Content-Type: image/jpeg

base64 encoded data .....
.........................
......base64 encoded data
--98766789--
```



Mail access protocols



- SMTP: delivery/storage to receiver's server
- Mail access protocol: retrieval from server
  - POP: Post Office Protocol
    - authorization (agent <==> server) and download
  - IMAP: Internet Mail Access Protocol
    - more features (more complex)
    - manipulation of stored messages on server
  - HTTP: Hotmail , Yahoo! Mail, etc.

POP3 protocol

authorization phase

- client commands:
  - **`user`**: declare username
  - **`pass`**: password (plain text!)
- 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 alice
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
```