Institute of Telecommunications
Warsaw University of Technology
2019

internet technologies and standards

- Piotr Gajowniczek
- Andrzej Bąk
- Michał Jarociński

Internet application delivery infrastructure

Domain Name System

DNS: domain name system

people: many identifiers:

- SSN, name, passport #

Internet hosts, routers:

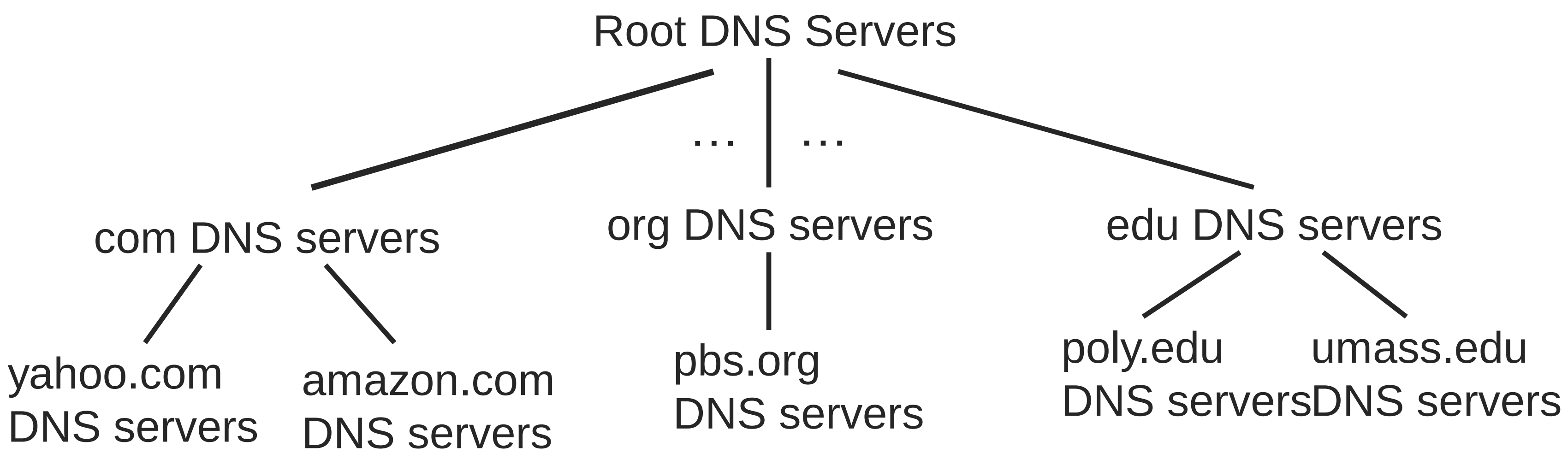
- IP address (32 bit) - used for addressing datagrams
- "name", e.g., www.yahoo.com - used by humans

Q: how to map between IP address and name, and vice versa ?

Domain Name System:

- *distributed database* implemented in hierarchy of many *name servers*
- *application-layer protocol: hosts, name servers communicate to resolve names (address/name translation)*
  - note: core Internet function, implemented as application-layer protocol
  - complexity at network's "edge"

DNS: a distributed, hierarchical database

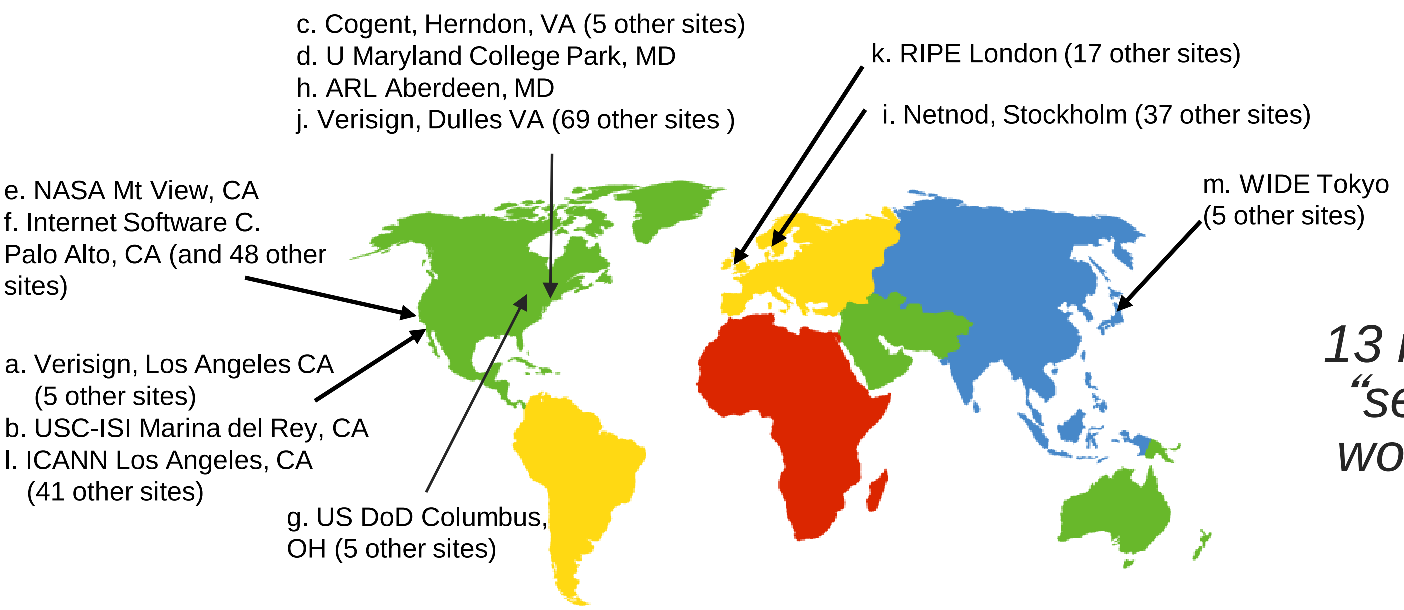


client wants IP for www.amazon.com; 1st approx:

- *client queries root server to find com DNS server*
- *client queries .com DNS server to get amazon.com DNS server*
- *client queries amazon.com DNS server to get IP address for www.amazon.com*

DNS: root name servers

- *on the top of the hierarchy*
- *contacted by local name server that can not resolve name*
- *IP addresses of all the root servers are known to all the DNS software packages, by default.*



13 root name "servers" worldwide

TLD, authoritative servers

top-level domain (TLD) servers:

- responsible for com, org, net, edu, aero, jobs, museums, and all top-level country domains, e.g.: uk, fr, ca, jp

authoritative DNS servers:

- organization's own DNS server(s), providing authoritative hostname to IP mappings for organization's named hosts (→ zone)
- master and slave for reliability
- can be maintained by organization or service provider

local DNS server

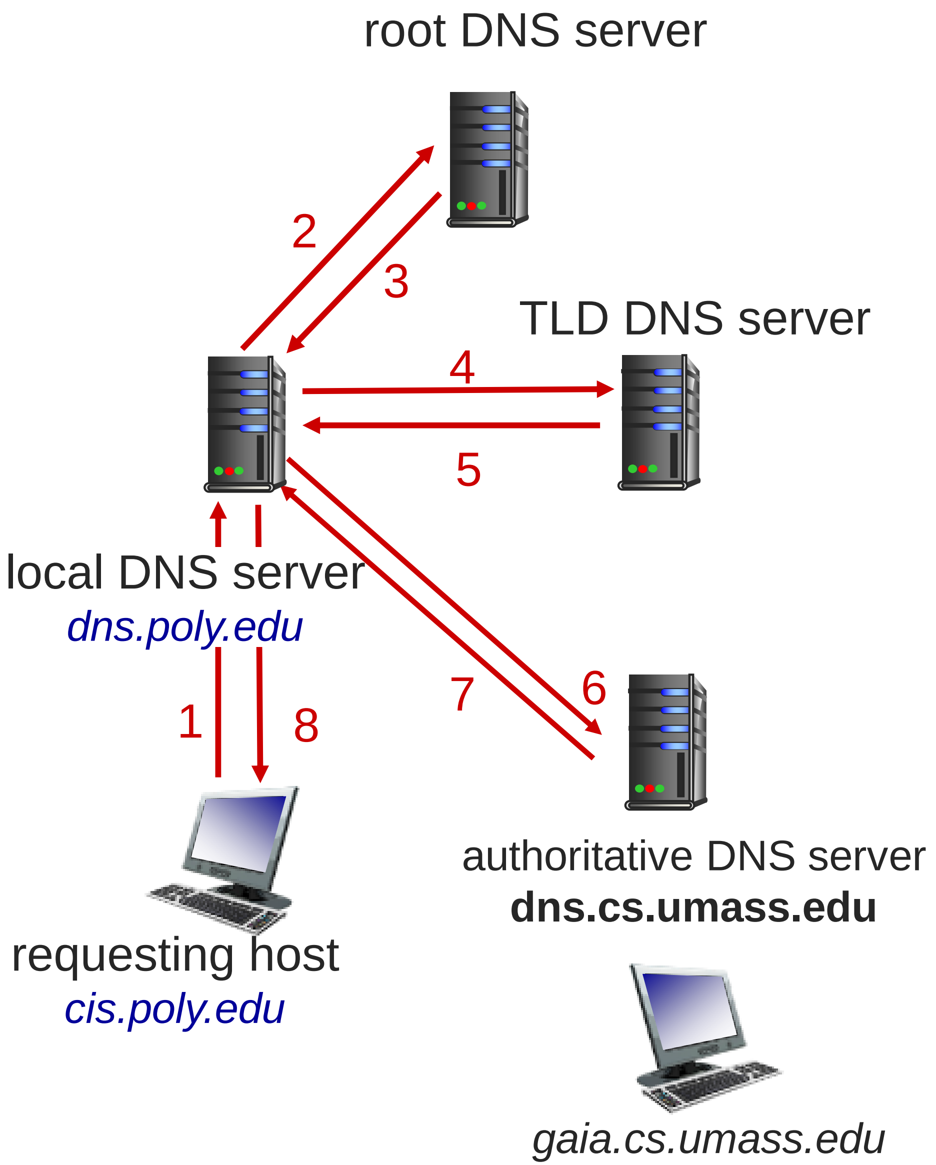
- also called "default name server"
- each ISP (residential ISP, company, university) has one; when host makes DNS query, query is sent to its local DNS server
- acts as proxy, forwards query into hierarchy
- keeps local cache of recent name-to-address translation pairs (but may be out of date!)
  - cache entries expire after some time (TTL)
  - typically caches also addresses of TLD servers (to lower the load on root name servers)

DNS name resolution example

- *host at cis.poly.edu wants IP address for gaia.cs.umass.edu*

iterated query:

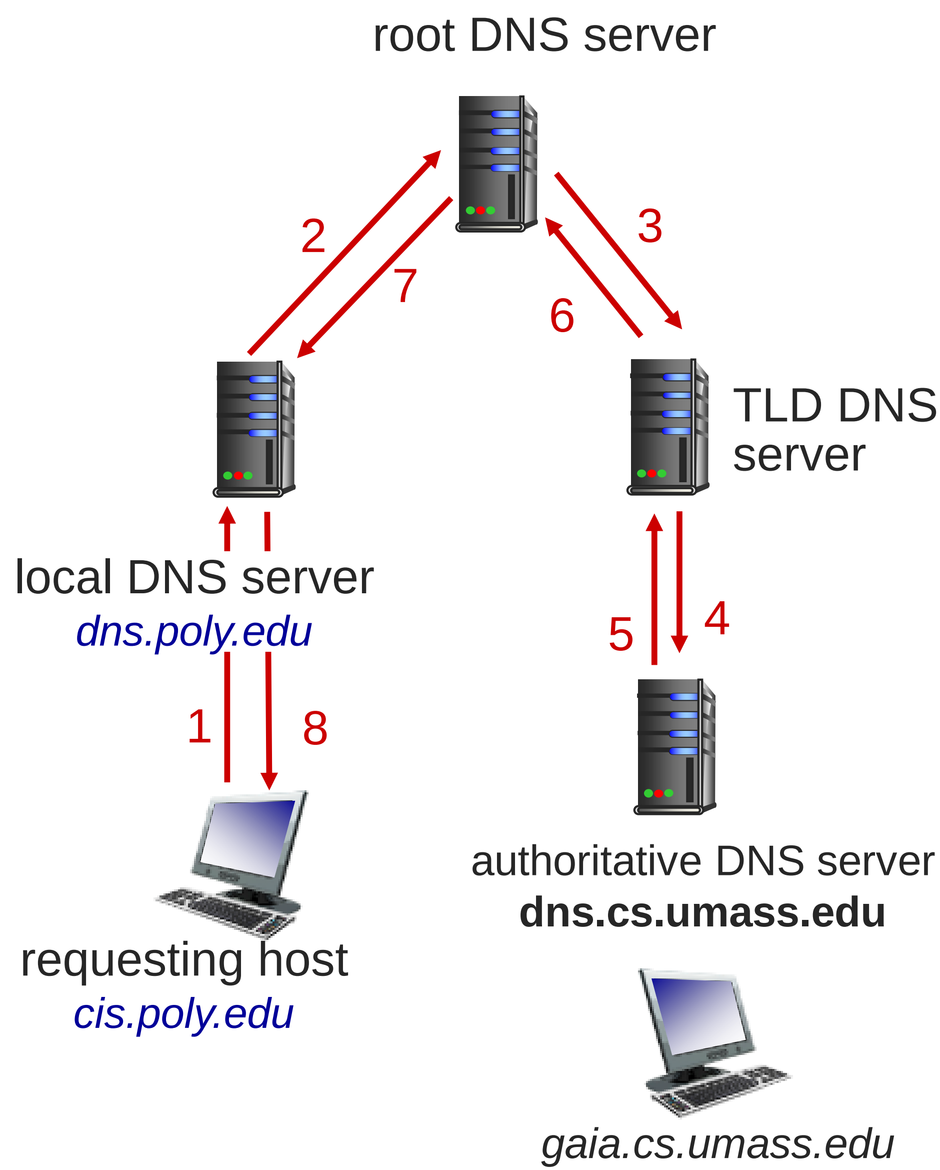
- contacted server replies with name of server to contact
- "I don't know this name, but ask this server"



DNS name resolution example

recursive query:

- puts burden of name resolution on contacted name server
- heavy load at upper levels of hierarchy?
  - root and TLD servers iterative only



DNS records

DNS: distributed db storing resource records (RR)

RR format: `(name, value, type, ttl)`

type=A
- **`name`** is hostname
- **`value`** is IP address

type=NS
- **`name`** is domain (e.g., foo.com)
- **`value`** is hostname of authoritative name server for this domain

type=CNAME
- **`name`** is alias name for some "canonical" (the real) name
- **`www.ibm.com`** is really **`servereast.backup2.ibm.com`**
- **`value`** is canonical name

type=MX
- **`value`** is name of mailserver associated with **`name`**

Internet application delivery infrastructure

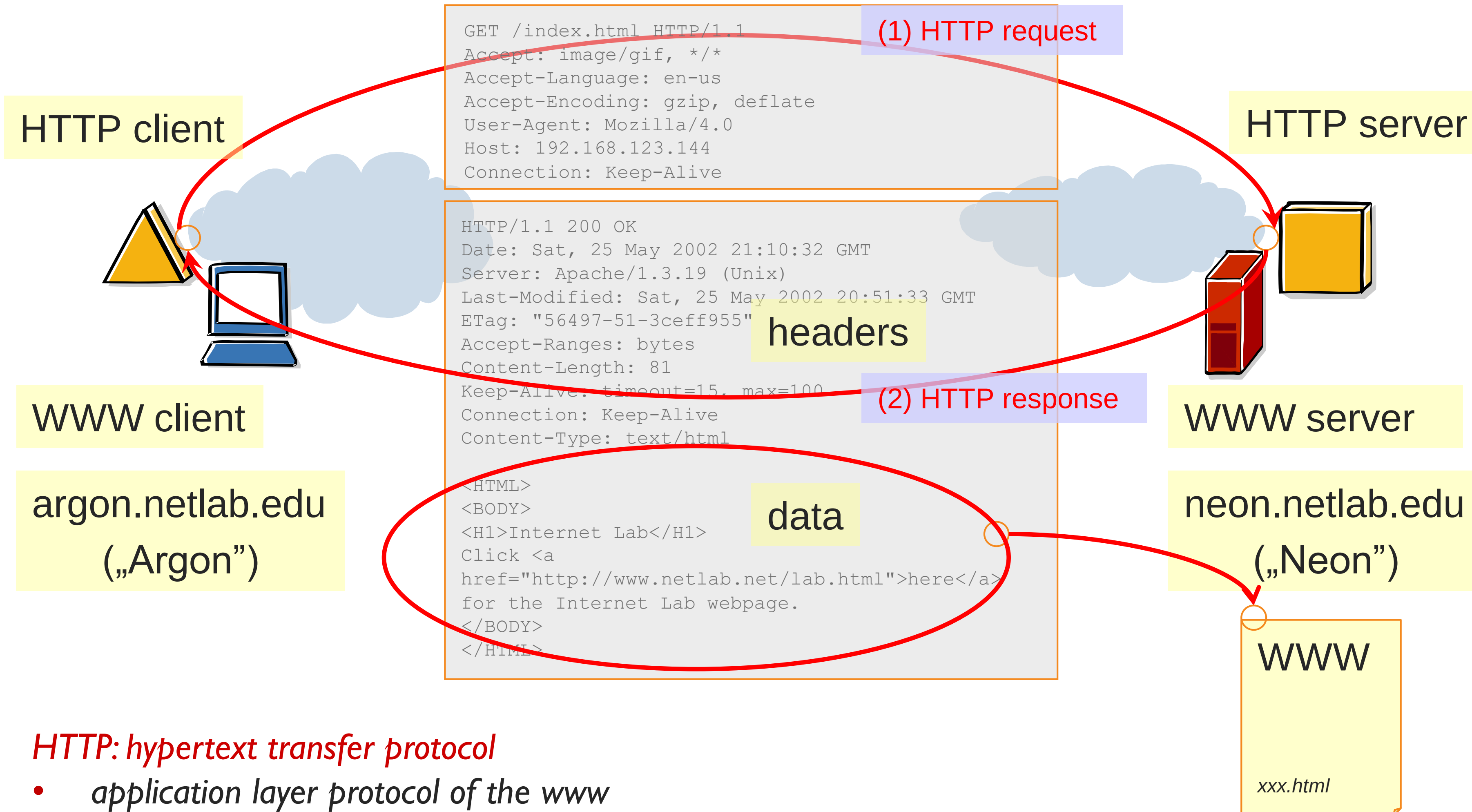
HTTP caching

web and HTTP

- *web page consists of objects*
- *object can be HTML file, JPEG image, Java applet, audio file,…*
- *web page consists of base HTML-file which includes several referenced objects*
- *each object is addressable by a URL,* e.g.,

$$\underbrace{\texttt{www.elka.pw.edu.pl}}_{\text{host name}}\underbrace{\texttt{/telecomm/picture.jpg}}_{\text{path name}}$$

HTTP overview



HTTP: hypertext transfer protocol

- *application layer protocol of the www*
- *client/server model*
  - *client:* browser that requests, receives, (using HTTP) and “displays” Web objects
  - *server:* Web server sends (using HTTP protocol) objects in response to requests

HTTP request message

- *HTTP request message:*
  - ASCII (human-readable format)

carriage return character

line-feed character

request line
(GET, POST,
HEAD commands)

```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\r\n
User-Agent: Firefox/3.6.10\r\n
Accept: text/html,application/xhtml+xml\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7\r\n
Keep-Alive: 115\r\n
Connection: keep-alive\r\n
\r\n
```

header
lines

carriage return,
line feed at start
of line indicates
end of header lines

HTTP overview (continued)

uses TCP:

- *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

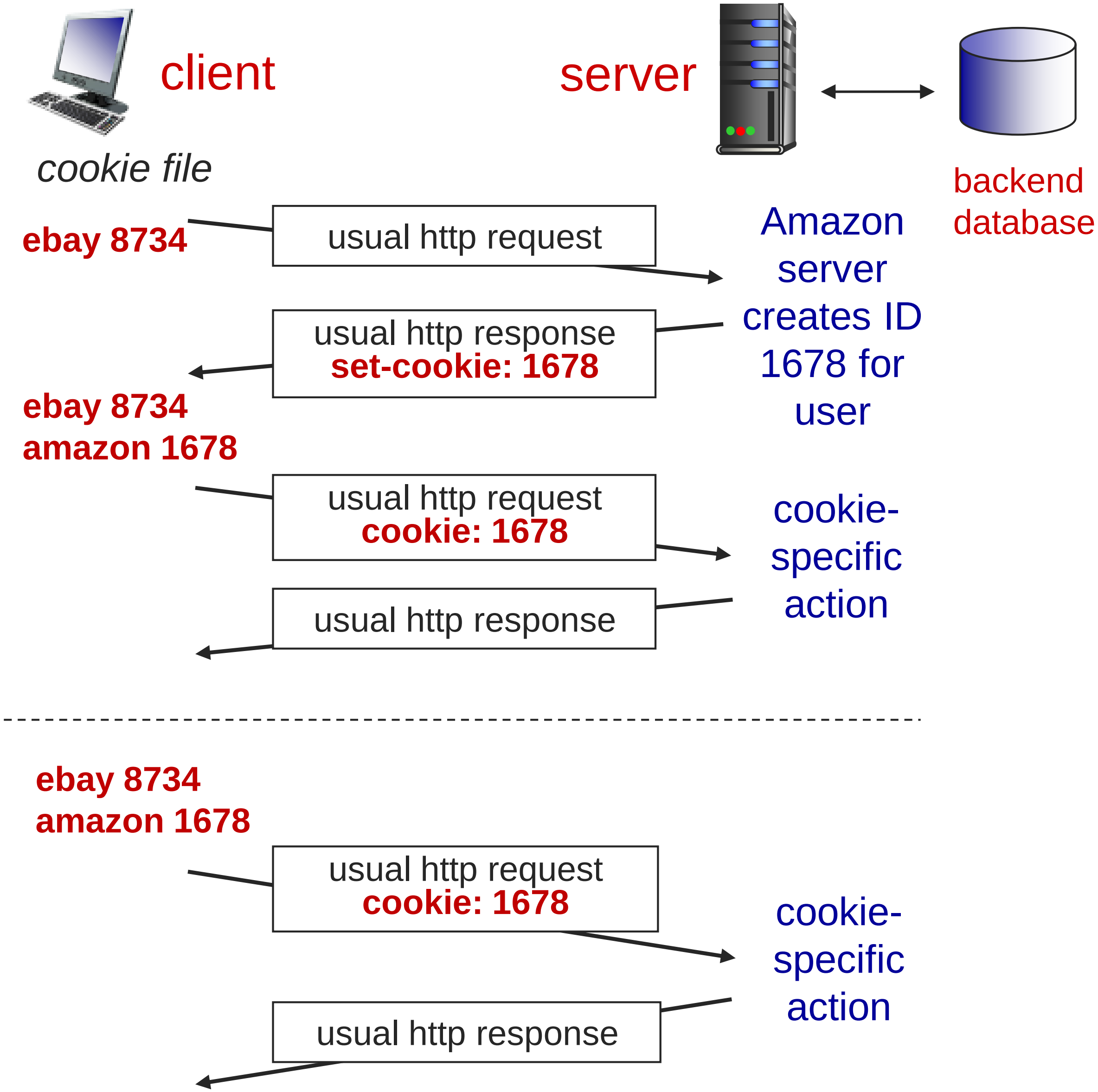
protocols that maintain "state" are complex!

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

user-server state: cookies

four components:

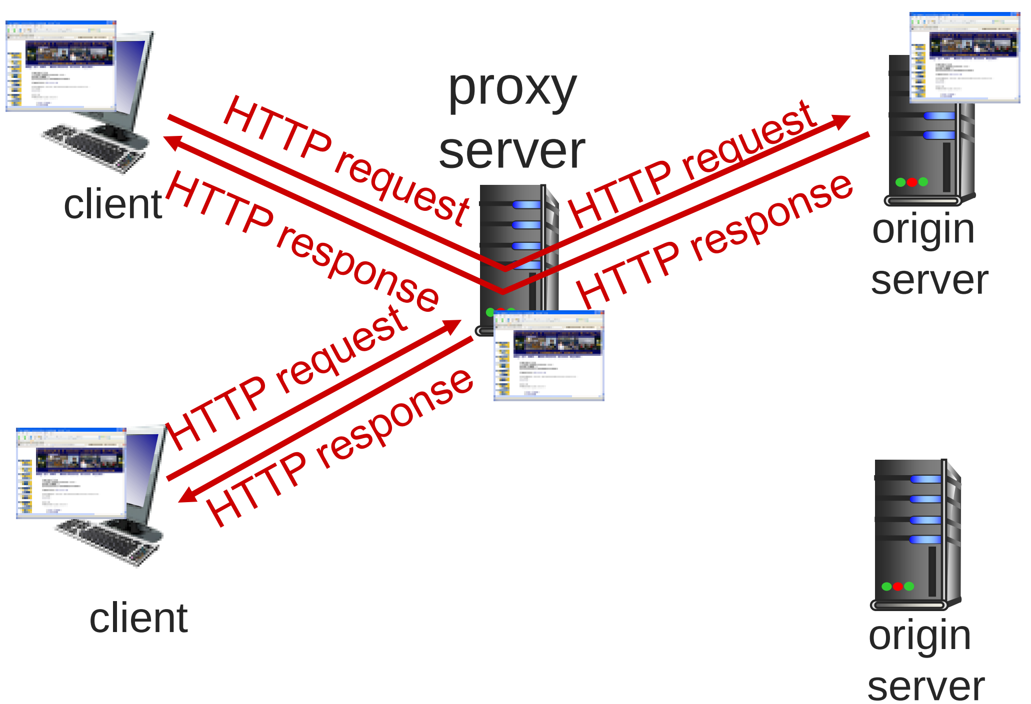
1) *cookie header line of HTTP response message*
2) *cookie header line in next HTTP request message*
3) *cookie file kept on user's host, managed by user's browser*
4) *back-end database at Web site*



web caches (proxy server)

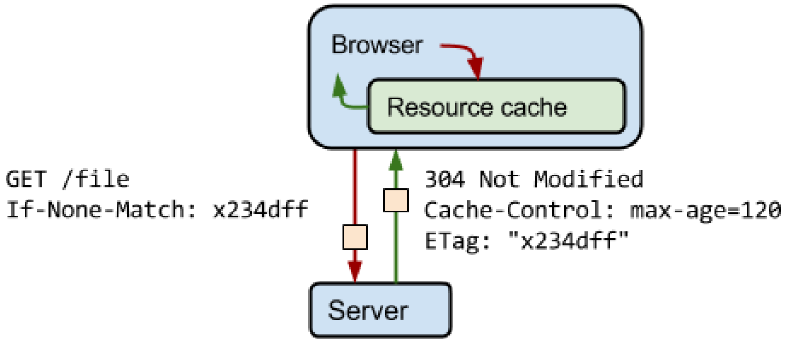
goal: satisfy client request without involving origin server

- *user sets browser: Web accesses via cache*
- *browser sends all HTTP requests to cache*
  - object in cache: cache returns object
  - else cache requests object from origin server, then returns object to client



web caching

- *web cache location:*
  - browser cache
  - proxy server
  - reverse proxy



https://developers.google.com/web/fundamentals/performance/optimizing-content-efficiency/http-caching

- *main headers involved:*
  - Etag: for validity checking using object hashing
  - Content-Length: allows reservation of space for object caching
  - Cache-Control: for setting object caching policies
    - **no-cache**: forces revalidation
    - **no-store**: blocks all caching
    - **public**: object can be stored in any cache
    - **private**: can be stored only in browser's cache
    - **max-age**: max. period of caching without revalidation
    - **must-revalidate**: prohibits sending stalled objects (e.g. when network is unavailable)
    - **no-transform**: cache can't modify object (e.g. compress)

more about Web caching

- *cache acts as both client and server*
  - server for original requesting client
  - client to origin server
- *typically cache is installed by ISP (university, company, residential ISP)*

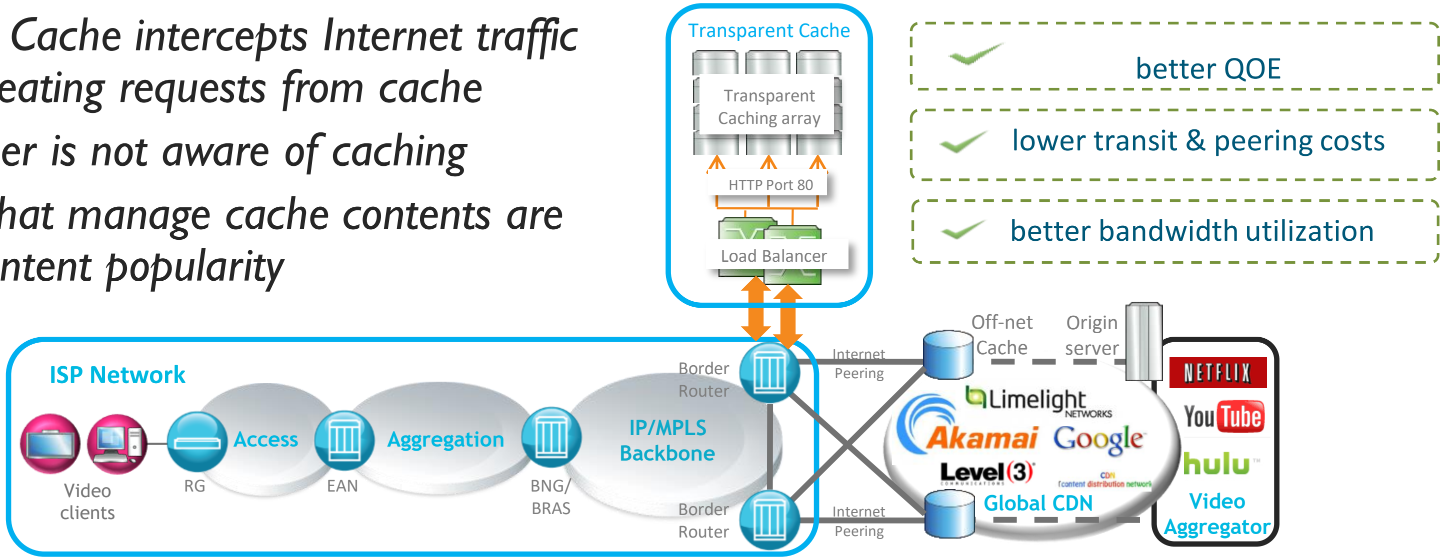
why Web caching?

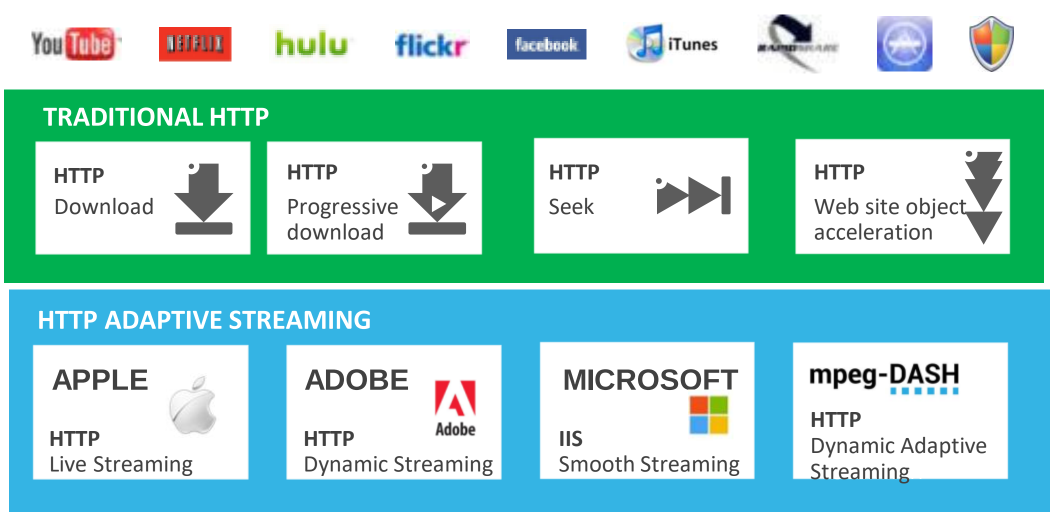
- *reduce response time for client request*
- *reduce traffic on an institution's access link*
- *Internet dense with caches: enables "poor" content providers to effectively deliver content (so too does P2P file sharing)*

- *advanced topics:*
  - transparent caching: redirection methods, cache filling algorithms ...
  - CDNs (Content Distribution Networks)

transparent caching

- *Transparent Cache intercepts Internet traffic to serve repeating requests from cache*
- *content owner is not aware of caching*
- *algorithms that manage cache contents are based on content popularity*





source: Alcatel-Lucent

CDN

- *CDN:*
  - overlay network for content delivery from optimal location
  - consists of clusters of servers distributed over large area (even globally) used to deliver Internet content to end users
  - CDN types:
    - 3rd party CDN, such as Akamai
    - ISP on-net CDN
  - basic redirection mechanizms
    - DNS
    - HTTP
    - anycast

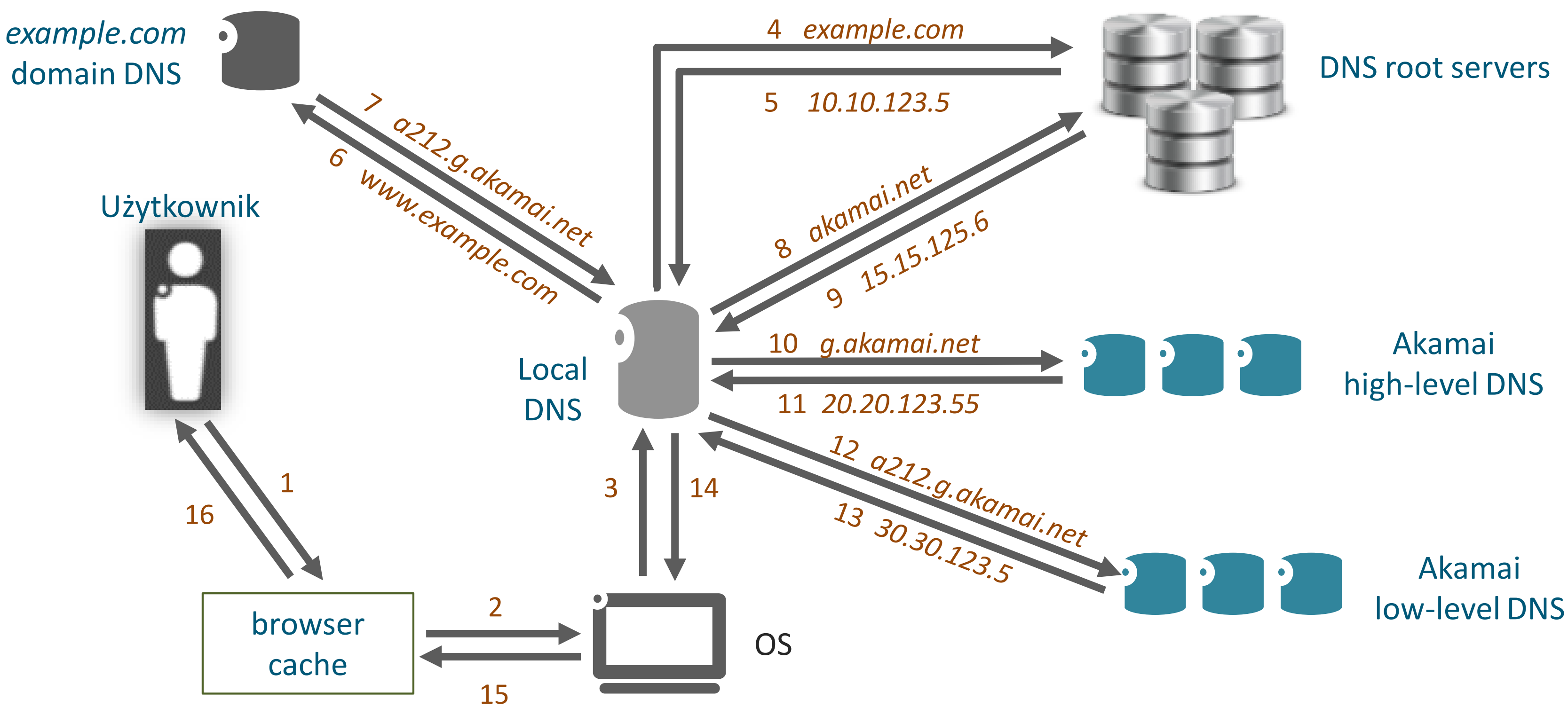
"The Web infrastructure and even Google's (infrastructure) doesn't scale. It's not going to offer the quality of service that consumers expect."

Vincent Dureau, Google's head of TV technology, Feb 2007

"Watching a television show or movie through some Internet video players can be an exercise in delayed gratification."

Nick Wingfield, Wall Street Journal, Dec 11'07

DNS redirection example



Internet application layer

multimedia/video streaming

multimedia networking:

- *two aspects:*
  - data stream transport
  - session establishment
- *application types*
  - streaming, stored audio, video
    - streaming: can begin playout before downloading entire file
    - stored (at server): can transmit faster than audio/video will be rendered (implies storing/buffering at client)
    - e.g., YouTube, Netflix, Hulu
    - streaming live audio, video – a special case
  - conversational voice/video over IP
    - interactive nature of human-to-human conversation limits delay tolerance
    - e.g., Skype

push vs pull approach

Push-based

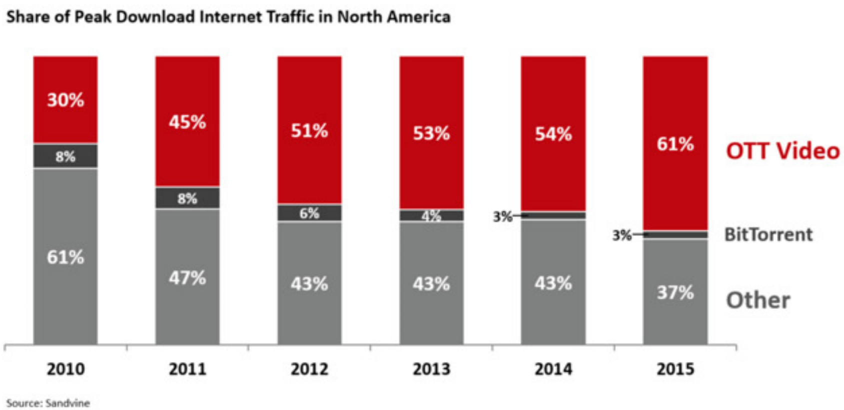
- *content is "pumped" (push) from the server to the client*
- *requires dedicated server*
- *and a protocol for session initiation (RTSP)*
- *data stream transfer usually based on UDP and RTP/RTCP*
- *adaptive approach possible (although standards are not widespread)*
- *transport set on dynamic port numbers (implies NAT/firewall problems)*

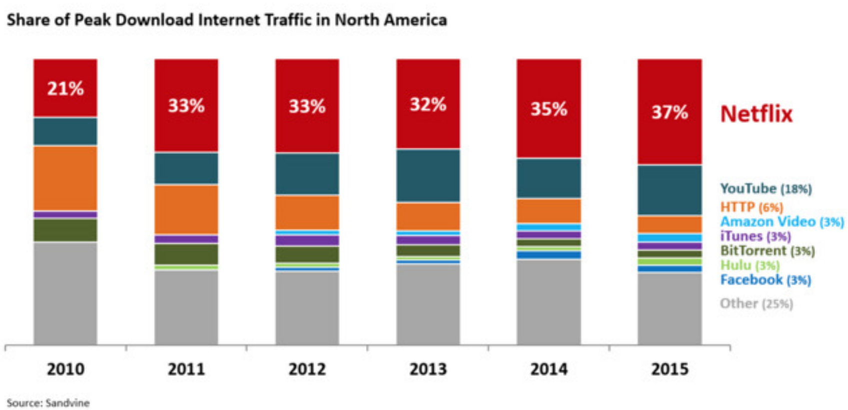
Pull-based

- *uses HTTP (port 80, usually no problems with NAT/firewall), and TCP*
- *unicast, data transfer initiated by client*
- *requires "inteligent" client and typical www server*
  - although you need specialized functionality for e.g. MP4 container
- *more overhead related to entity management on server side*
- *model closer to existing Internet infrastructure (WWW, caching)*
- *adaptive approaches are widespread*

OTT (Over-the-Top) video

- *OTT video – video transfer to end user without dedicated infrastructure, through the Internet*





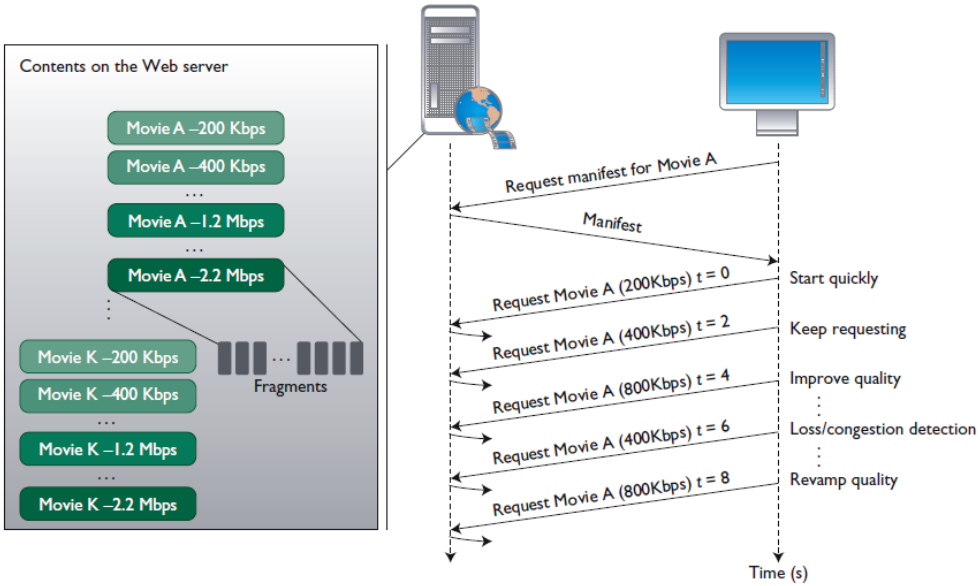
various solutions

- *HTTP download*
  - HTTP GET
- *progressive download*
  - rendering during download
- *pseudostreaming*
  - relies on the ability of HTTP clients to seek to positions in the media file by performing byte range requests to the Web server (HTTP 1.1)
  - keyframes are used as reference points, so the accuracy of seek depends on how the video was encoded

- *HTTP adaptive streaming (HAS)*

HTTP Adaptive Streaming

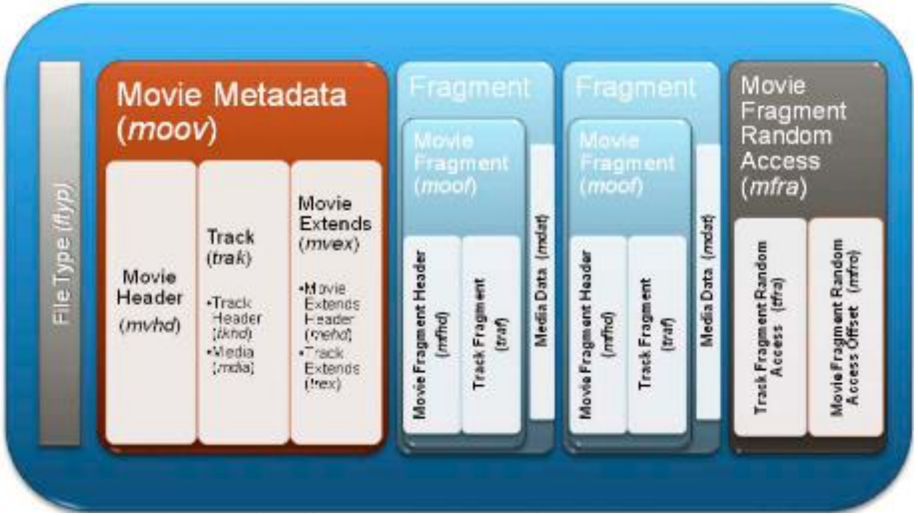
- *server:*
  - divides video file into multiple chunks
  - each chunk stored, encoded at different rates
  - manifest file: provides URLs for different chunks
- *client:*
  - periodically measures server-to-client bandwidth
  - consulting manifest, requests one chunk at a time
    - chooses maximum coding rate sustainable given current bandwidth
    - can choose different coding rates at different points in time (depending on available bandwidth at time)



- *"intelligence" at client: client determines*
  - *when* to request chunk (so that buffer starvation, or overflow does not occur)
  - *what encoding rate* to request (higher quality when more bandwidth available)

Microsoft Smooth Streaming – MP4 container

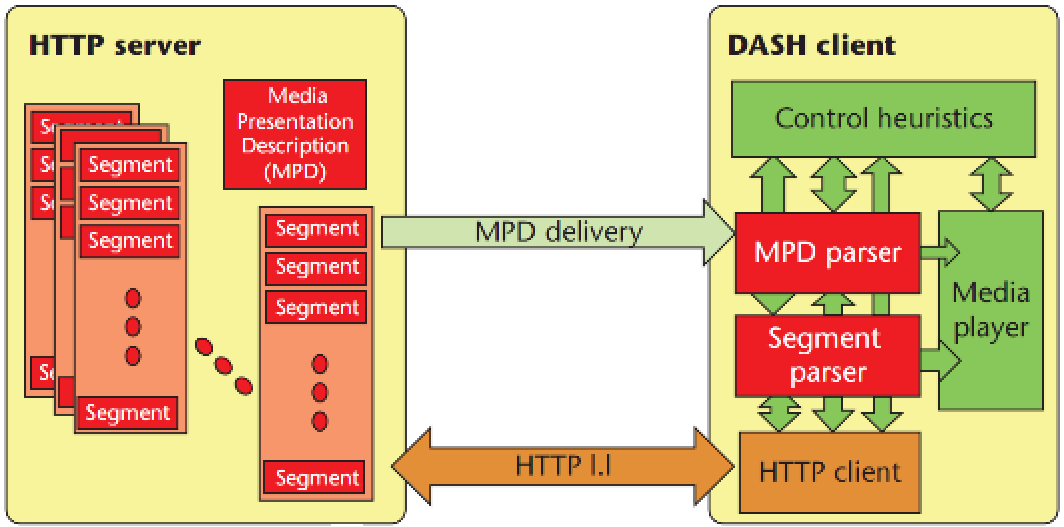
- *all fragments with the same bitrate are stored in a single MP4 file*
- *when a client requests a specific source time segment from the IIS Web server, the server dynamically finds the appropriate Movie Fragment box within the MP4 file*
- *chunks are sent as a standalone files (full cacheability downstream possible)*



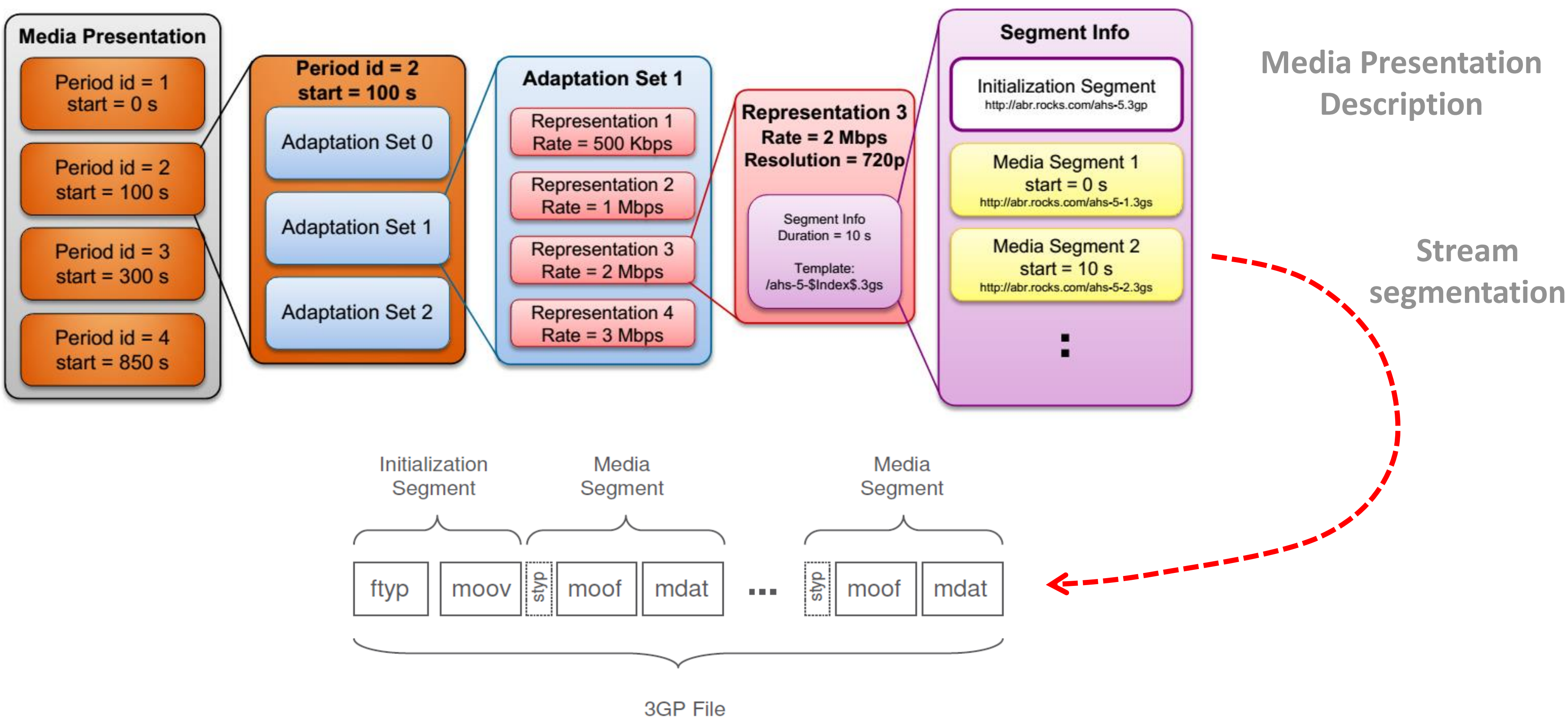
RESTful URL:
http://video.sth.com/mediafile.ism/QualityLevels(400000)/Fragments(video=610275114)

DASH - Dynamic Adaptive Streaming over HTTP

- *HAS standardization attempt*
  - based on existing solutions
  - independent on codec, DRM format, application protocol (although HTTP most popular)



DASH - Dynamic Adaptive Streaming over HTTP



Internet application layer

VoIP

service example: Voice-over-IP (VoIP)

- *voice transfer in IP packet network*
- *requirements*
  - delay (to maintain "conversational" aspect)
    - < 150 ms – good perceived quality
    - > 400 ms – very bad
    - delay budget – many factors, on app and network side
  - delay jitter
    - requires buffering
  - packet loss
    - congestion (router buffer overflow)
    - large delay (data useless, so practically lost, even if received)
    - tolerance depends on many factors
- *other aspects*
  - loss detection, RT stream timing?
  - session establishment – how to advertise IP address, port number, codec type ?
  - QoS, QoE

Real-Time Protocol (RTP)

- *RFC 3550*
  - implemented in end systems
  - defines packet structure for audio/video payload, defines RTP session
  - RTP packets encapsulated in UDP segments

- *RTP packet provides*
  - payload type identification
  - packet sequence numbering
  - time stamping

RTP header

payload type	sequence number	time stamp	Synchronization Source ID	Miscellaneous fields

- *payload type (7 bits):* indicates type of encoding currently being used. If sender changes encoding during call, sender
- informs receiver via payload type field
  - Payload type 0: PCM mu-law, 64 kbps
  - Payload type 3: GSM, 13 kbps
  - Payload type 7: LPC, 2.4 kbps
  - Payload type 26: Motion JPEG
  - Payload type 31: H.261
  - Payload type 33: MPEG2 video

- *sequence # (16 bits):* increment by one for each RTP packet sent
  - detect packet loss, restore packet sequence

RTP header

payload type	sequence number	time stamp	Synchronization Source ID	Miscellaneous fields

- *timestamp field (32 bits long): sampling instant of first byte in this RTP data packet*
  - for audio, timestamp clock increments by one for each sampling period (e.g., each 125 usecs for 8 KHz sampling clock)
  - if application generates chunks of 160 encoded samples, timestamp increases by 160 for each RTP packet when source is active. Timestamp clock continues to increase at constant rate when source is inactive.

- *SSRC field (32 bits long): identifies source of RTP stream. Each stream in RTP session has distinct SSRC*

Real-Time Control Protocol (RTCP)

- *works in conjunction with RTP*
- *each participant in RTP session periodically sends RTCP control packets to all other participants*
  - each RTCP packet contains sender and/or receiver reports
- *functions:*
  - report statistics useful to application: # packets sent, # packets lost, interarrival jitter (feedback can be used to control performance)
  - provides clock info for synchronization between RTP streams
  - conveys info providing identification of session participants

SIP: Session Initiation Protocol [RFC 3261]

- *what is SIP?*
  - IETF end-to-end call establishment and management protocol for multimedia services
  - supports user location, presence, profiles
  - allows call setup (with authorization)
  - allows media media type, encoding negotiation
  - supports session management (modification of session parameters, multiparty conferencing etc.)
  - supports different forms of redirection (e.g. forking)
  - supports WWW interaction/integration
- *defines also*
  - SIP addressing (SIP URI; maps mnemonic identifier to current IP address)
  - network servers architecture (registrar, proxy, redirect, location servers)

SIP protocol

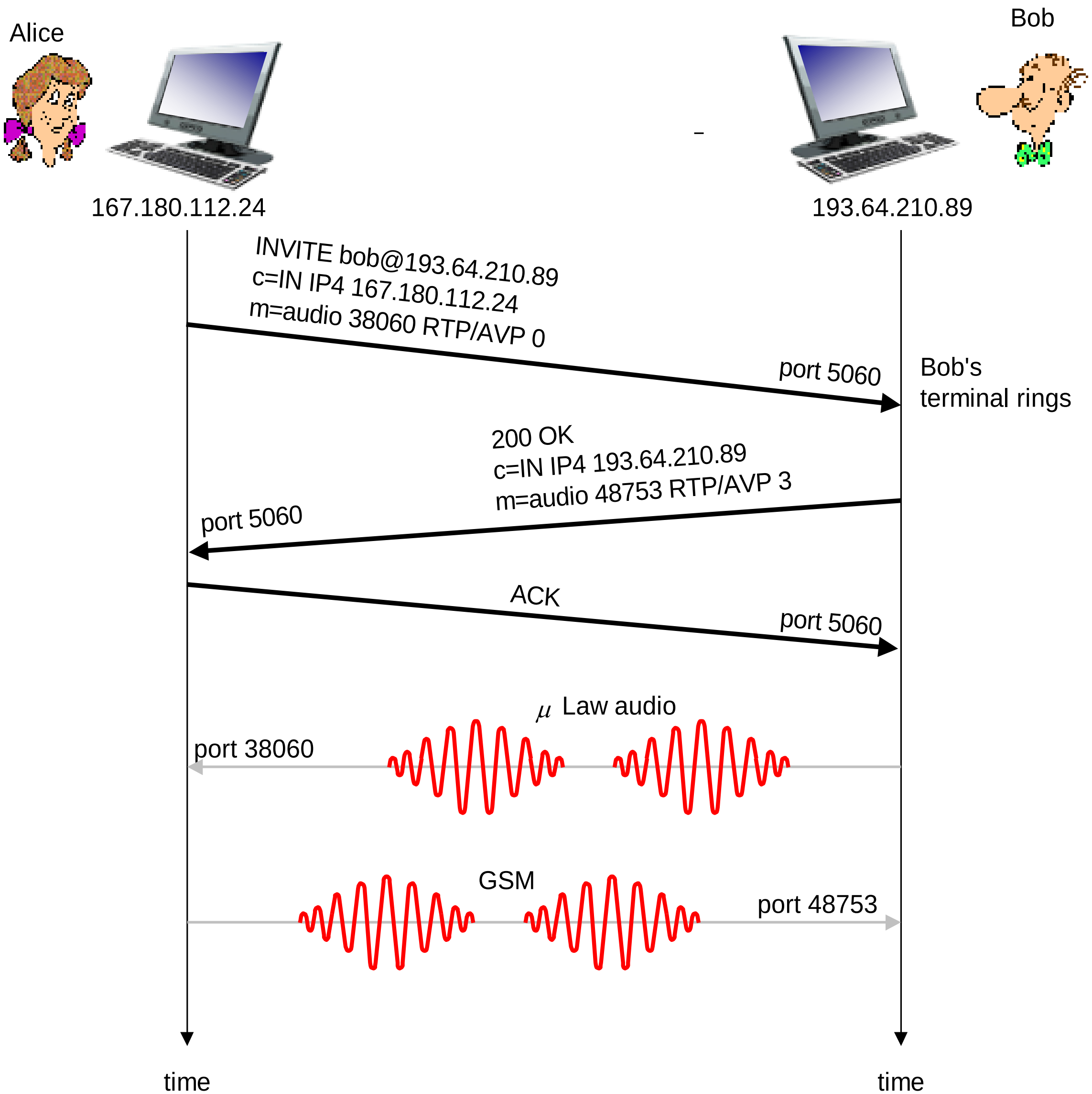
- *commands*
  - REGISTER – registration of physical address of the client; INVITE – call setup/modification; ACK – ack. for the final response, BYE – ending session; CANCEL – breaking transaction; others: SUBSCRIBE, OPTIONS, INFO
- *responses*
  - 1xx - Provisional – request received and under processing; 2xx - Success; 3xx - Redirection; 4xx - Client Error – request cannot be processed (bad syntax, no authorization etc.); 5xx - Server Error – serwer is unable to proces a (viable) request; 6xx - Global Failure
- *transaction*
  - a command and all responses; non-INVITE scenario – command, sequence of provisional responses and a final response; INVITE scenario – additional ACK command

```
INVITE sip:he.dog@netlab.edu SIP/2.0
Via: argon.netlab.edu

From: sip:she.cat@voice.com; Tag=12
To: sip:he.dog@netlab.edu
Call-Id: 12345@argon.netlab.edu
Cseq: 1 INVITE
Contact: sip:she.cat@argon.netlab.edu
Content-type ... Content-length ...
+SDP
```

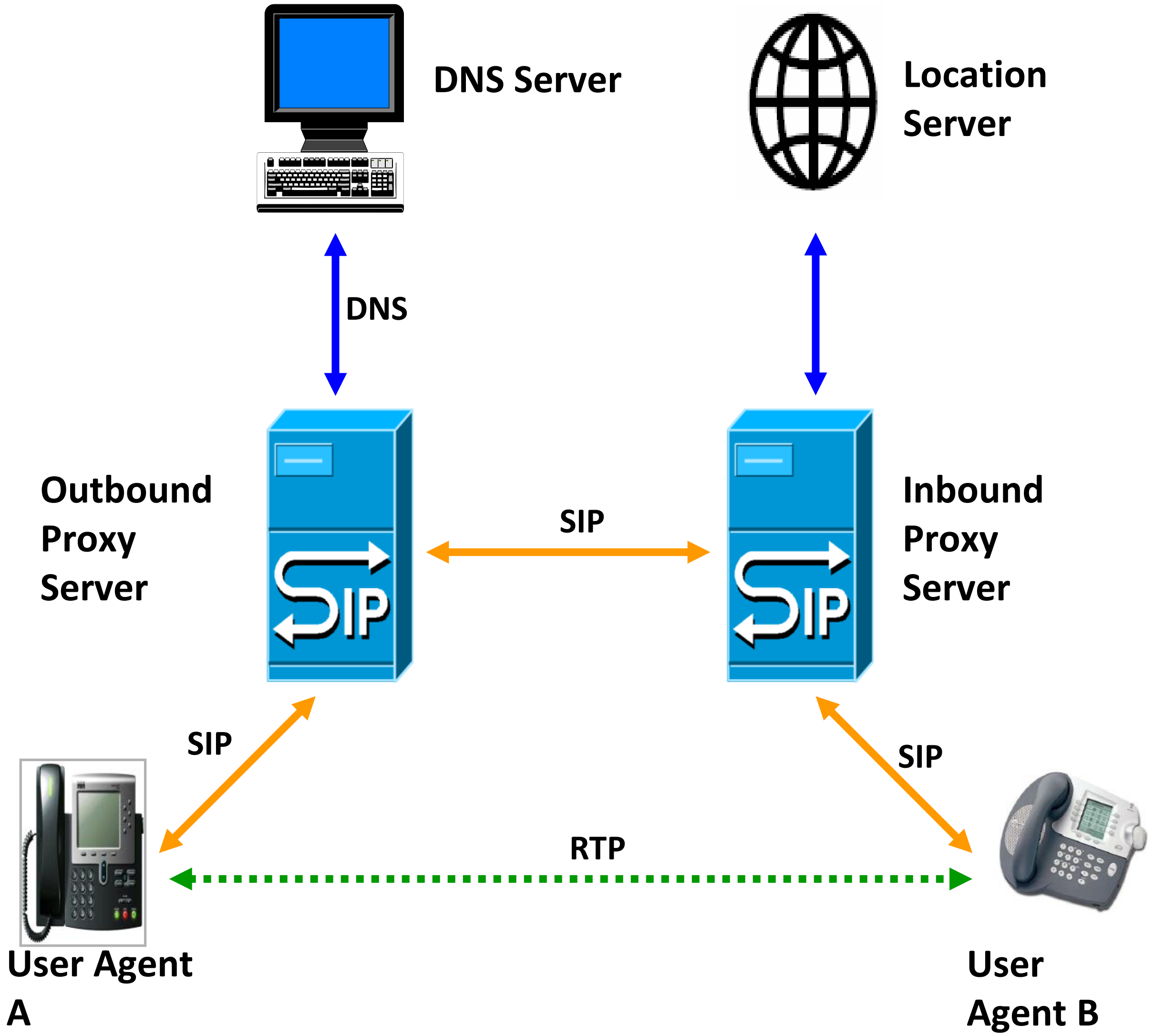
```
200 OK SIP/2.0
Via: sipproxy.netlab.edu
Via: argon.netlab.edu
From: sip:she.cat@voice.com; Tag=23
To: sip:he.dog@netlab.edu; Tag=12
Call-Id: 12345@argon.netlab.edu
Cseq: 1 INVITE
Contact: sip:he.dog@neon.netlab.edu
Content-type ... Content-length ...
+ SDP
```

example: setting up call to known IP address

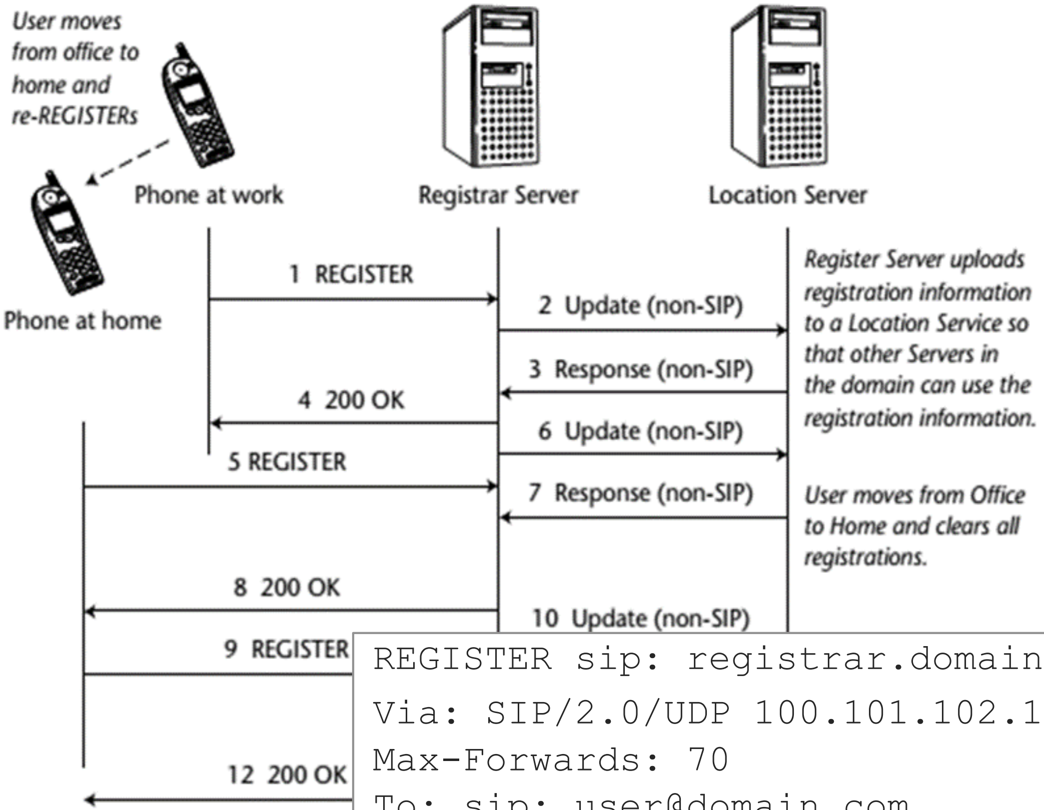


- Alice's SIP invite message indicates her port number, IP address, encoding she prefers to receive (PCM μlaw)
- Bob's 200 OK message indicates his port number, IP address, preferred encoding (GSM)
- SIP messages can be sent over TCP or UDP; here sent over RTP/UDP
- default SIP port number is 5060

SIP functional architecture



SIP Registration




```
REGISTER sip: registrar.domain.com SIP/2.0
Via: SIP/2.0/UDP 100.101.102.101:5060;branch=z9hG4bKfw19b
Max-Forwards: 70
To: sip: user@domain.com
From: <sip: sender@domain.com>;tag=42333
Call-ID: 22322424@100.101.102.101
CSeq: 1 REGISTER
Contact: sip:kontakt@domain.com
Content-Length: 0
```

Internet datacenters

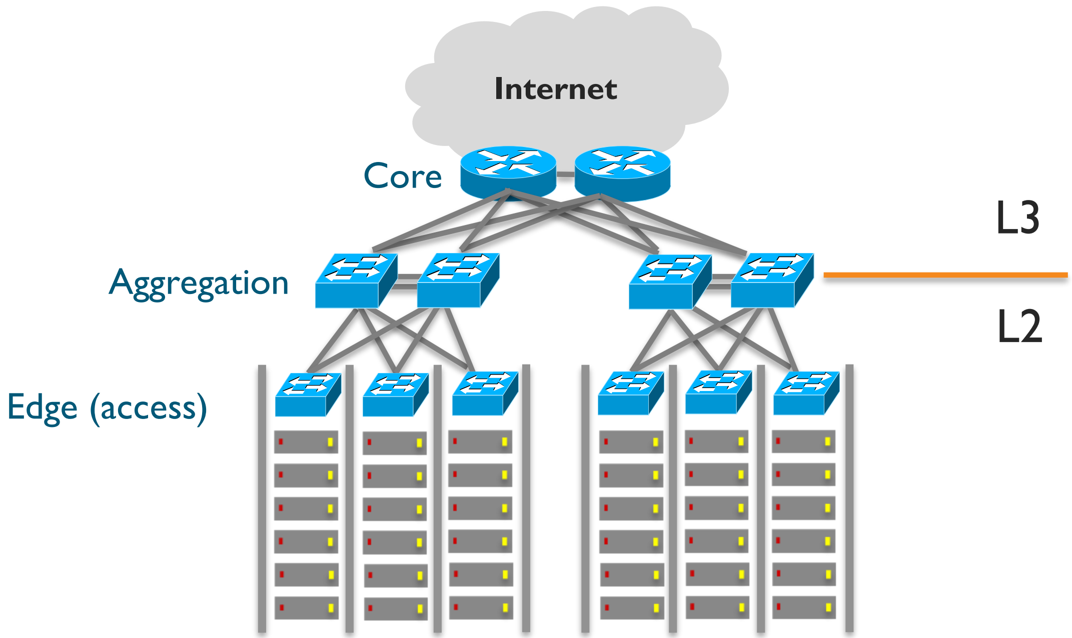
Introduction

- *Internet datacenters:*
  - servers and storage (*bare metal*)
  - organized in racks and rows
  - everything is interconnected

- *Servers hosting applications/services are accessible by the end user via the Internet*
  - old way: apps/services hosted on dedicated servers
  - modern way: apps/services hosted on Virtual Machines (*cloud computing*)
- *Cloud computing is:*
  - on-demand usage of datacenter CPU/storage for providing app/service

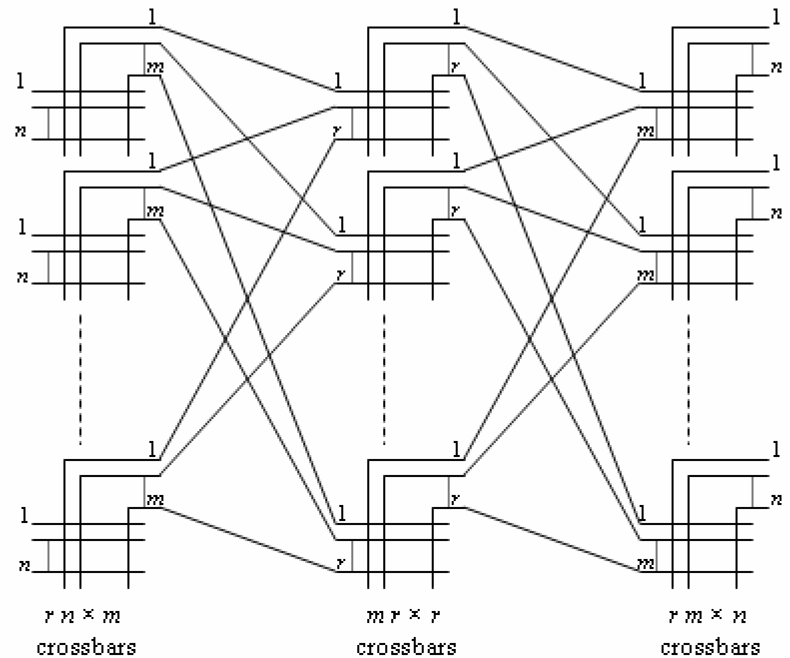
Legacy architecture

- *3-layer hierarchy*
  - optimized for N-S traffic
  - aggregation layer: L2/L3 border; loop prevention by STP (induces limitations in bandwidth usage)
  - edge: VLANS for traffic management (separation of apps)



Clos fabric

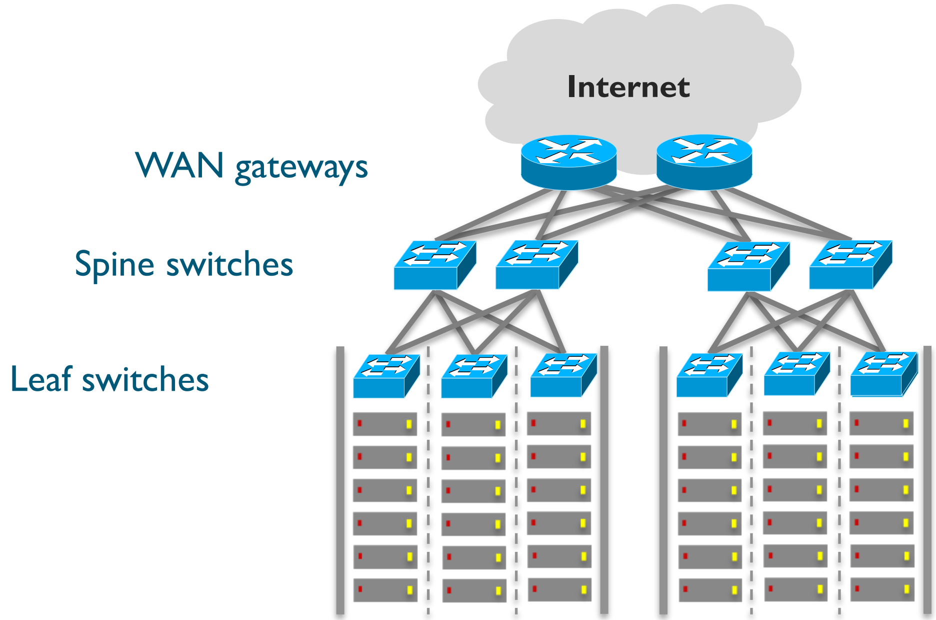
- *Concept of non-blocking switching fabric*
- *Invented for circuit-switching*



- *Made a "comeback" with a shift to virtualized computing*
  - MAC explosion
  - rise of E-W traffic
  - traffic volume increase

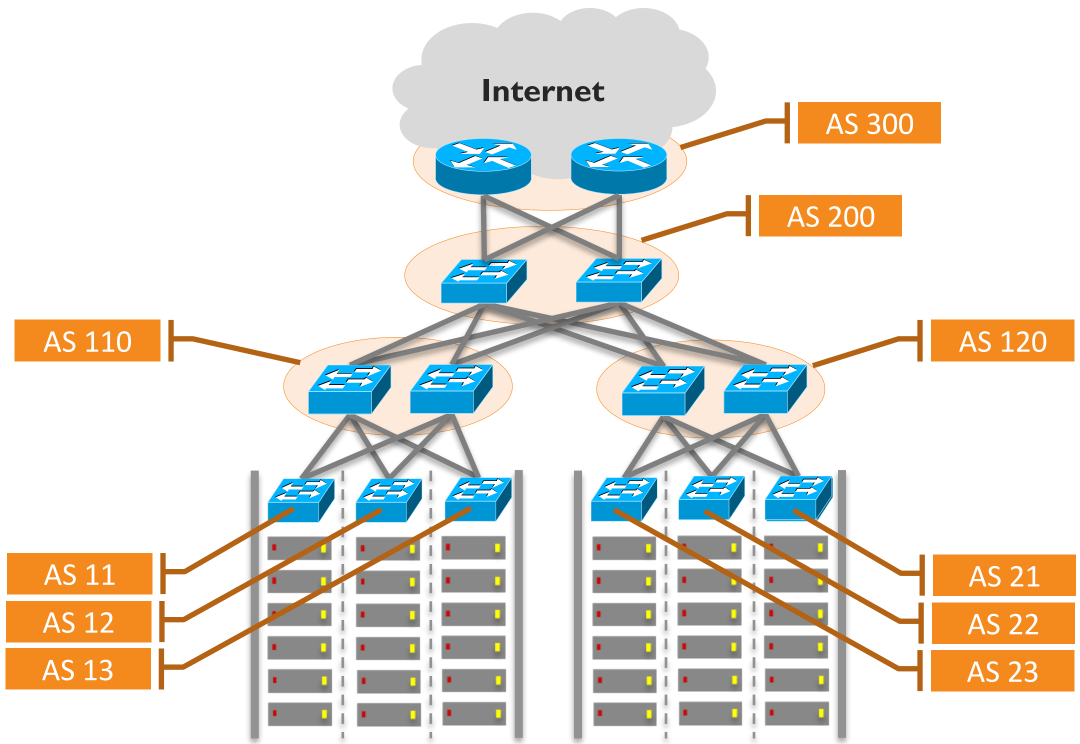
DC architecture evolution

- *Leaf-Spine architecture*
- *Not only change in terminology*
  - each leaf connected to each spine (mesh)
  - L2 terminated on edge (leaf) switches
  - L3 between leaves and spines
- *Better suited to handle E-W traffic*



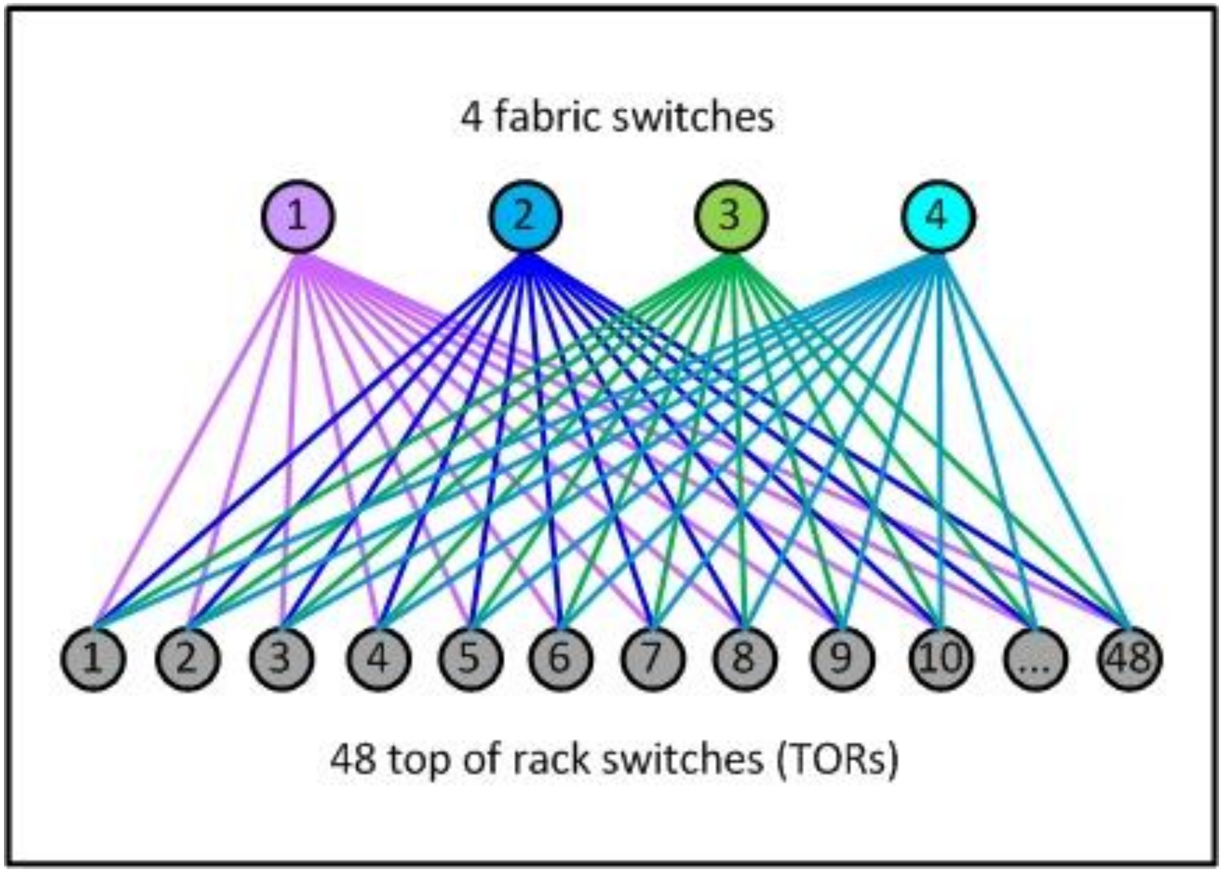
Datacenter IP fabric

- *Built of multiple clusters (and planes)*
- *Huge scale = IGP not a viable option for L3 routing*
- *→ BGP!*
    - scalable
    - traffic engineering (path attributes)



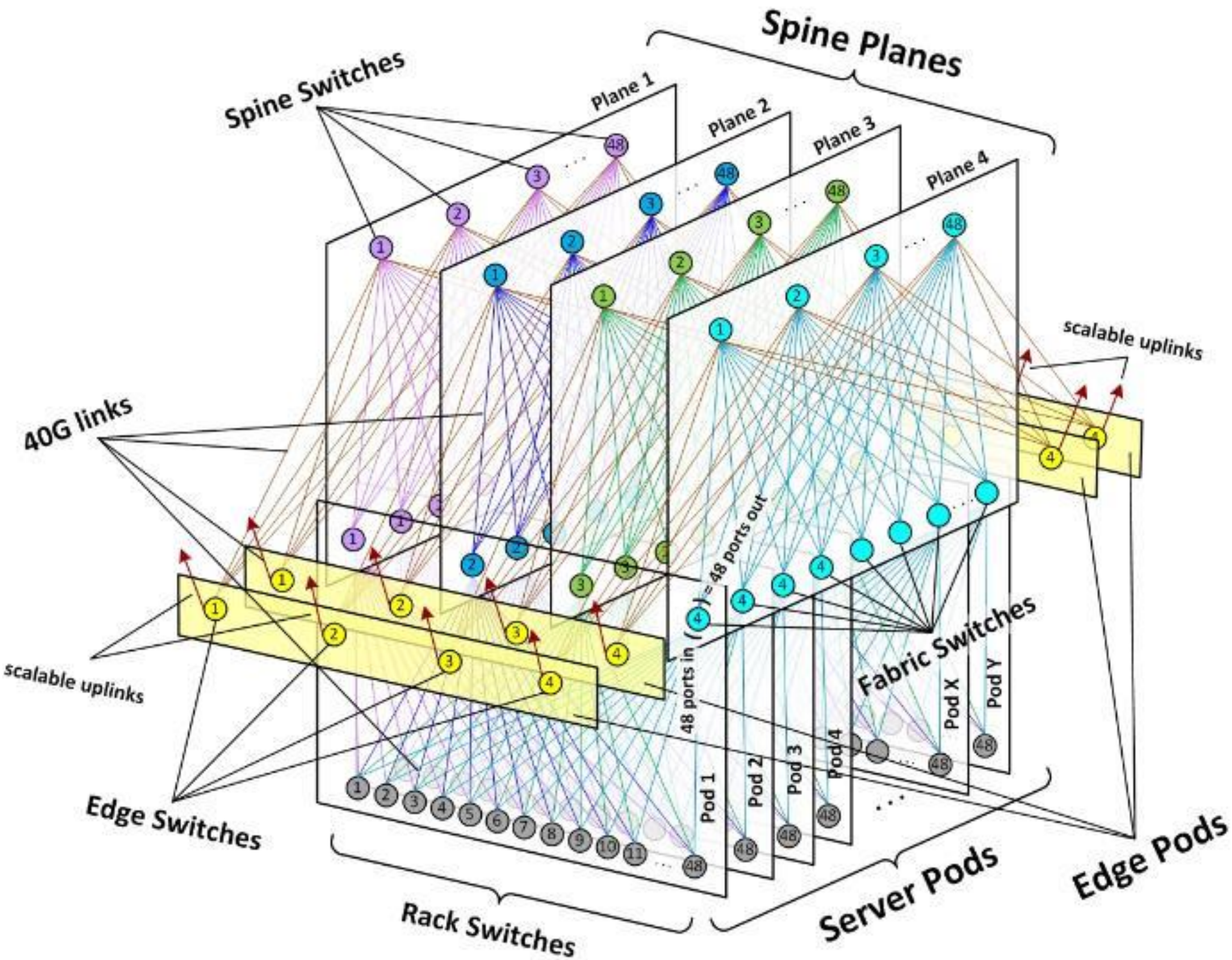
Facebook DC
single POD

- [Facebook Blog Post](https://code.facebook.com/posts/360346274145943/introducing-data-center-fabric-the-next-generation-facebook-data-center-network/) : https://code.facebook.com/posts/360346274145943/introducing-data-center-fabric-the-next-generation-facebook-data-center-network/



- *a single DC "unit"*

Facebook DC
DC fabric



- *highly-modular design*
- *all L3 network*
- *uses ECMP routing*
- *uses "centralized BGP controller" to "override any routing paths on the fabric by pure software decisions"*