IAIK TU Graz

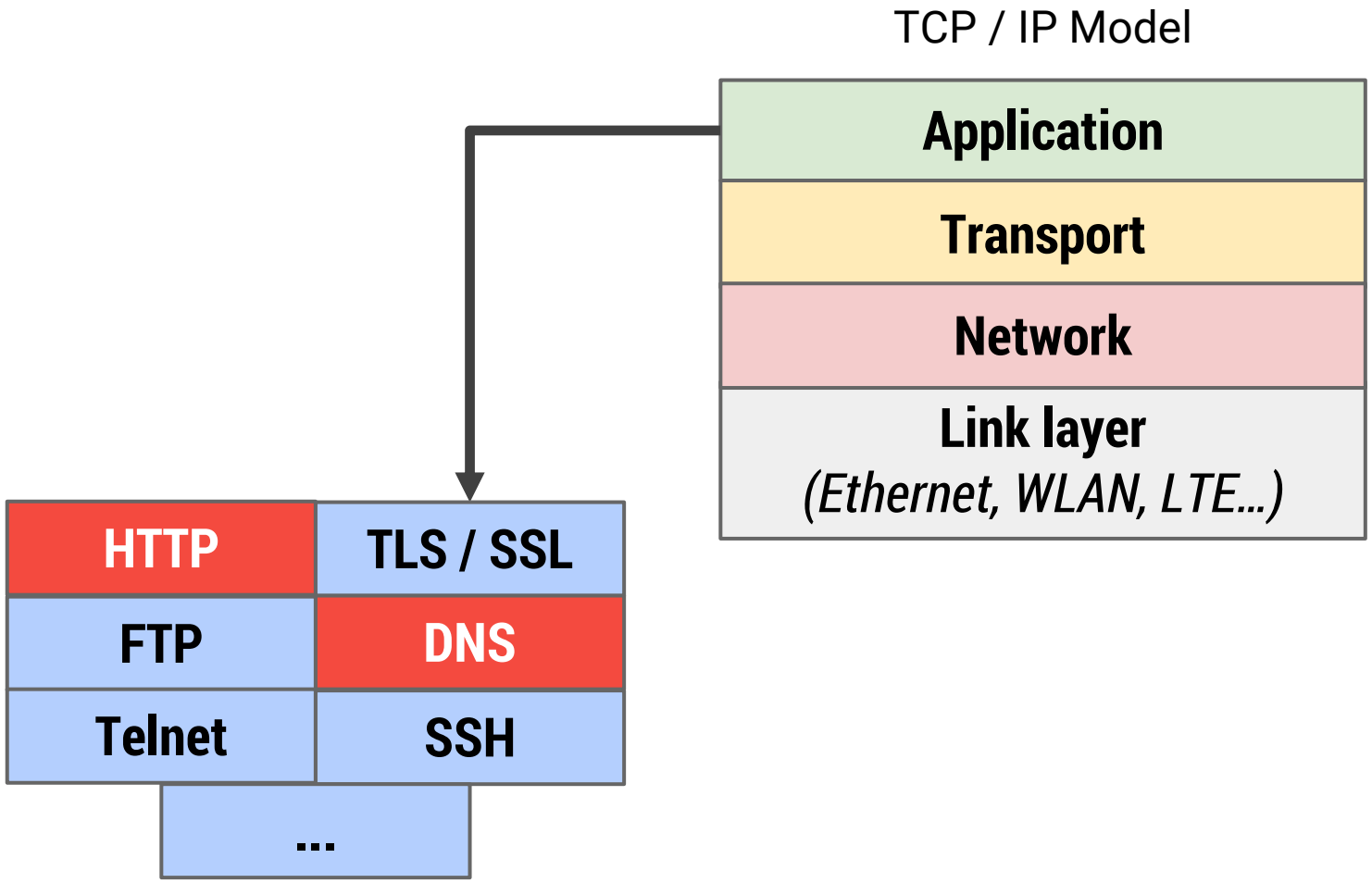
Application Layer

Computer Organization and Networks 2019

Johannes Feichtner
johannes.feichtner@iaik.tugraz.at

Outline

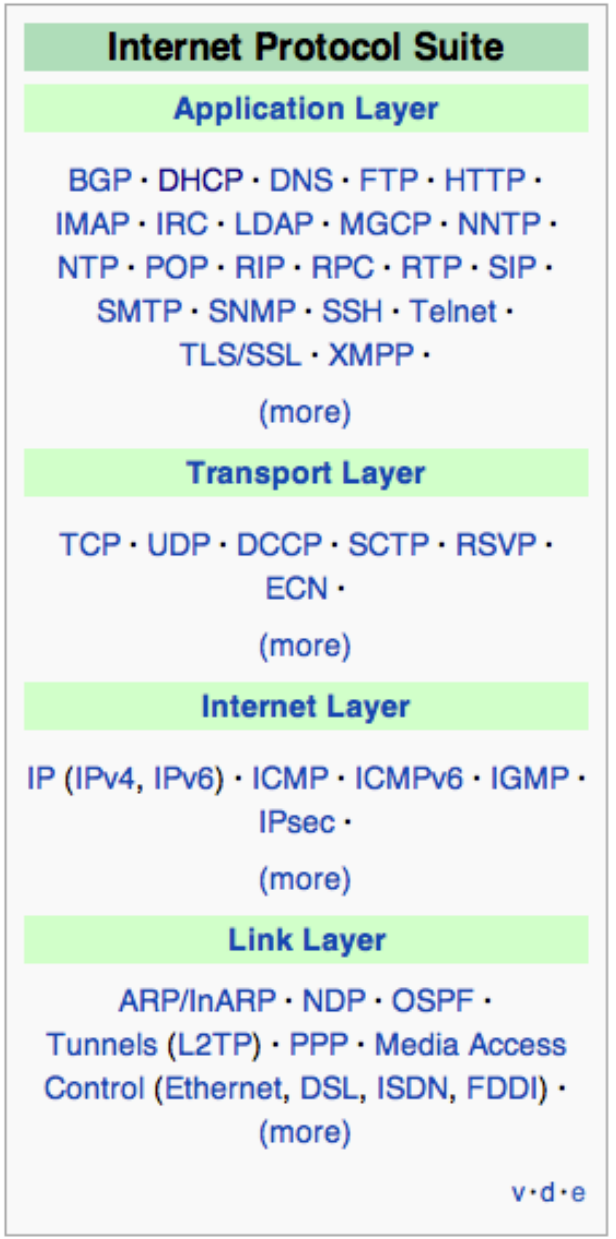
- HTTP Basics
  - Request Types
  - HTTP/2
- Advanced communication
  - AJAX
  - WebSockets
  - HTML5 postMessage
- DNS
  - Protocol
  - Resource Recrods



Review: TCP / UDP

- Service provisioned to higher layers through ports
  - Port 80 for HTTP, 443 for HTTPS / TLS, 21 for FTP, …
- Session: Communication client / server via socket pair
  - TCP: Established after fulfilling a handshake
    - Connection-oriented
    - Reliable → error detection, flow & congestion control
  - UDP: Identified on higher layer, e.g. using session cookies
    - Connection-less
    - Unreliable → sender does not know if destination reached
    - No congestion control

HTTP!



HTTP Introduction

Basics

- Used by browsers to fetch data from web servers

- Simple (stateless) request / response protocol
  - Client opens TCP connection, requests document
  - Server responds with document
  - Client closes TCP connection

- Multiple versions
  - 1991: HTTP 0.9
  - 1996: HTTP 1.0 (RFC 1945)
  - 1999: HTTP 1.1 (RFC 7230)
  - 2015: HTTP/2 (RFC 7540)

HTTP 0.9

```
telnet testserver.com 80
Connected to 129.27.10.20

GET /news

RKN is great via HTTP 0.9!
(connection closed)
```

GET method + ASCII string
Terminated by carriage return (CRLF)

No header or other metadata!

- Pure ASCII protocol over TCP/IP link
  - Still supported by popular webservers, e.g. Apache, nginx due to simplicity!
- Designed to transfer hypertext documents (HTML)
- Connection between server / client closed after every request

HTTP 1.0

```
telnet testserver.com 80
Connected to 129.27.10.20

GET /news.html HTTP/1.0
User-Agent: libwww-perl/5.805

HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 15824
Last-Modified: Wed, 1 May 2016 12:55:25 GMT
Server: Apache 1.3.10

RKN is great via HTTP 1.0!
(connection closed)
```

Request with HTTP version + headers

- (Multiple) newline-separated fields

Response status + headers

- Response no longer limited to hypertext, different content (media) types
- Still ASCII transfer, regardless of media

→ New features also: Content encoding, character sets, authorization, caching, date formats, etc.

HTTP 1.1

```
telnet testserver.com 80
Connected to 129.27.10.20

GET /news.html HTTP/1.1
Host: realserver.com
Accept-Language: de,en-US,q=0.8
Accept-Charset: de,en-US;q=0.7,*;q=0.3
...

HTTP/1.1 200 OK
Connection: keep-alive
Transfer-Encoding: chunked
Expires: Wed, 1 May 2016 12:55:25 GMT

100
<!doctype html> ...
```

Most notable changes:

- *Connection kept-alive by default*
- *Chunked data transfer*

→ New features: Language negotiation, caching directives, transfer encoding, …

Request with HTTP version + headers

– (Multiple) newline-separated fields

Chunked response for HTML request

Chunked Encoding

Enables server to „stream“ content in chunks to client

→ *Useful e.g. if server has not yet processed or generated the data it sends*

Standardized with HTTP 1.1

- Transfer-Encoding: chunked
- No Content-Length header

```
HTTP/1.1 200 OK
Connection: keep-alive
Transfer-Encoding: chunked
Expires: Wed, 1 May 2016 12:55:25 GMT

100
<!doctype html>...
(256 bytes in total = 100 in hex)

94
...</html>
(148 bytes in total = 94 in hex)

0
```

Structure

- Every chunk prefixed with number of bytes that follow in hexadecimal format
- Followed by actual chunk
- 0 = End of chunk stream → subsequent request may follow

HTTP Request

```
GET / HTTP/1.1
Host: orf.at
User-Agent: Mozilla/5.0 (Windows NT 6.3; Win64; x64; rv:46.0) Gecko/20100101 Firefox/46.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: de,en-US;q=0.7,en;q=0.3
Accept-Encoding: gzip, deflate
DNT: 1
Cookie: vote-2293=cast; vote-2563=cast; HopSession=127.0.0.129.27.152.ej3ly0nnpu92; Vietnam2Session=127.0.0.129.27.152.p4k2cp9f64ib
Connection: keep-alive
If-None-Match: "wGAhvKXOC9SUJlTqyVtolw=="

HTTP/1.1 200 OK
Date: Thu, 19 May 2016 11:59:36 GMT
Server: Jetty(6.1.22)
X-Cache: HIT from localhost
ETag: "trbmVS1Aq3SC0ZGlQftasA=="
Content-Length: 20985
Content-Type: text/html; charset=utf-8
X-Uncompressed-Size: 98719
Content-Encoding: gzip
Cache-Control: max-age=0
Expires: Thu, 19 May 2016 11:59:36 GMT
Accept-Ranges: none
Connection: close

...........}.n.I..{}E....F....%..:.-.*Y...e...FP..a.....bO..0.0@........O.4.....k....H_%Kfu5.,......k.o.....O..`[..
0P.......zw.:fNulrg...9|etg.;....dZ..u....\.uk.sj..]?..5.........g...8.....j.(.2.....wL.g...I..;..6"M...
[s2.......d.;...*....l........Ov.:.. .}W67L..I..;sq...[i.........e&....t.....a..S?..F........%&McZ...e..WhM...;.i`..
1.......7..Q....t..G..c.y..Q.o....{g....:...OL...B.....K.Yh.Z^.t...P..N..t|g...X.....g..
.8j..O..0.Qv.ar.S.).....z..LY..?....Y.O|W .[..q...\..c<].....Jsz8.L-....~MZ.....fqr....
$n...w...o....G.>.zs..#.#...k..i.].G.....K.o........F...^]zC.]p..:..zB..=.K...'.9..o..0......n..I...[]Z_;j|.../.NLs.Ms1.
{q.....NA.G.c.d.;MY'=...K..W.......*m..eu.D3..>.9..f.q.]....].O.8z....w...}...b.....u*.,$i..
7aP.......tTMU.O..O.V..Q...p.....gier..?.w0..=..{.O&...'..j...#...bf....N.W.+Yi.}KwB.U{k,..++............
63.>p..U.......G..P...vU...m....~...a..............j.....%.... -I.....:....Z.i....(.hG..ldb.;F.._...o.{.
(=.r..)}..iBG....u.U.F......HE..:.;=.$..Ry.Q.....F....du.....~...
:..s:.q.};...r|        =....Y.......8
```

GET Request

Request Headers

Client sends no body here… only CRLF

Server status code

Server response

Response body: gzipped content

IAIK TU Graz

HTTP Status Codes

First line of HTTP response is status number…

Number	Reason
101	Switching protocols → WebSockets
200	OK → Standard response for successful HTTP request
201	Created → Request fulfilled, new resource created
202	Accepted → Request ok but not yet processed
301	Moved permanently → Redirect requests to given URL
400	Bad Request → Malformed request syntax
401	Unauthorized → Client should authenticate
403	Forbidden → Request was valid but access denied
404	Not Found → Resource not found
500	Internal Server Error → Generic error message
502	Bad Gateway → Server got no servable response

1xx Information

2xx Success

3xx Redirect

4xx Client Error

5xx Server Error

For more codes, see https://goo.gl/G43lii

HTTP Requests

- Safe methods: `GET, HEAD, OPTIONS, TRACE`
  - Never change resource representation
  - Cacheable, Pre-fetchable

- Unsafe methods: `POST, PUT, DELETE, PATCH`
  - Change resource representation

Usage depends on desired action…

- Read https://iaik.tugraz.at → `GET`
- Login to https://www.facebook.com → `POST`
- Write to REST API → `PUT, DELETE`
- Connect via HTTP Proxy → `CONNECT`

HTTP GET

```
telnet test.iaik.tugraz.at 80

GET / HTTP/1.1
Host: test.iaik.tugraz.at
User-Agent: Mozilla/5.0 (Windows NT 6.3; Win64; x64; rv:46.0) Gecko/20100101 Firefox/46.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: de,en-US;q=0.7,en;q=0.3
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Connection: keep-alive
Cache-Control: max-age=0
Keep-Alive: 115

HTTP/1.1 200 OK
Date: Thu, 19 May 2016 12:42:13 GMT
Server: Jetty(6.1.22)
X-Cache: HIT from localhost
ETag: "mShMvdHTUFOHQjPRrcLD2w=="
Content-Length: 105920
Content-Type: text/html; charset=utf-8
Cache-Control: max-age=0
Expires: Thu, 19 May 2016 12:42:13 GMT
Accept-Ranges: none
Connection: close
```

Retrieves information from requested URI (but does not change the resource!)

→ Idempotent!

HTTP POST / PUT

POST: Not idempotent

- Updates, creates, adds resources

→ Sending request again would re-trigger same action

```
telnet test.iaik.tugraz.at 80

POST /newentry.php HTTP/1.1
Host: test.iaik.tugraz.at
User-Agent: Mozilla/5.0 (Windows NT 6.3; Win64; x64; rv:46.0) Gecko/20100101 Firefox/46.0
Cookie: sessionId=123452515afasfdaf
Content-Type: application/x-www-form-urlencoded

Name=RKN+Demo&institute=IAIK&lecture=42&secret=1+%2B+1+%3D+2
```

Name: RKN Demo
institute: IAIK
lecture: 42
secret: 1+1=2

PUT: Idempotent

- Creates or replaces resources (e.g. PUT /addinvoice/1)

HTTP HEAD

```
telnet test.iaik.tugraz.at 80

HEAD / HTTP/1.1
Host: test.iaik.tugraz.at
User-Agent: Mozilla/5.0 (Windows NT 6.3; Win64; x64; rv:46.0) Gecko/20100101 Firefox/46.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: de,en-US;q=0.7,en;q=0.3
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Connection: keep-alive
Cache-Control: max-age=0
Keep-Alive: 115

HTTP/1.1 200 OK
Date: Thu, 19 May 2016 12:42:23 GMT
Server: Jetty(6.1.22)
X-Cache: HIT from localhost
ETag: "sXjgIafhHToGNe+8P/X20Q=="
Content-Length: 0
Content-Type: text/html; charset=utf-8
Cache-Control: max-age=0
Expires: Thu, 19 May 2016 12:42:13 GMT
Accept-Ranges: none
Connection: close
```

- Retrieves headers only
- Equal to GET but without body

Useful, e.g. to get

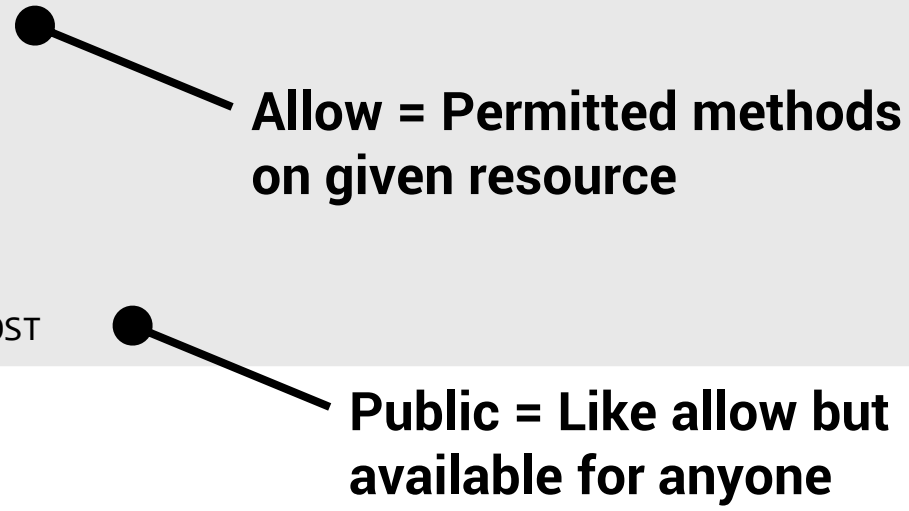
- Meta-information stored in headers, e.g. session information
- Check if URL is servicable / link exists
- Check if cached content should be redownloaded

HTTP OPTIONS

```
telnet test.iaik.tugraz.at 80

OPTIONS / HTTP/1.1
Host: test.iaik.tugraz.at
User-Agent: Mozilla/5.0 (Windows NT 6.3; Win64; x64; rv:46.0) Gecko/20100101 Firefox/46.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: de,en-US;q=0.7,en;q=0.3
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Connection: keep-alive
Cache-Control: max-age=0
Keep-Alive: 115

HTTP/1.1 200 OK
Allow: OPTIONS, TRACE, GET, HEAD
Date: Thu, 19 May 2016 12:42:33 GMT
Server: Jetty(6.1.22)
X-Cache: HIT from localhost
ETag: "sXjgIafhHToGNe+8P/X20Q=="
Content-Length: 0
Public: OPTIONS, TRACE, GET, HEAD, POST
```



Return methods a server provides for some resource

HTTP TRACE

```
telnet test.iaik.tugraz.at 80

TRACE / HTTP/1.1
Host: test.iaik.tugraz.at
Accept: *
Cookie: sessionId=123452515afasfdaf

HTTP/1.1 200 OK
Content-Type: text/plain
Date: Thu, 19 May 2016 12:42:43 GMT
Content-length: 414
Via: 1.1 secretserver.iaik.tugraz.at

TRACE / HTTP/1.1
Host: test.iaik.tugraz.at
Accept: *
Cookie: sessionId=123452515afasfdaf
Via: 1.1 secretserver.iaik.tugraz.at
```

- Intended for debugging → **echo**es back received request
- Useful for detecting changes that intermediate servers made, e.g. proxy

Considered insecure → can help to bypass security controls during attack (cookie stealing)!

HTTP CONNECT

Used for proxies to tunnel TLS connections

- Standard way for clients behind HTTP proxy to access HTTPS websites

```
telnet proxy.iaik.tugraz.at 80

CONNECT google.at:443 HTTP/1.1
```

Workflow

1. Client requests HTTP proxy server
   - Request includes destination and port (google.at:443)
     Proxy creates connection on behalf of client
2. Proxy then forwards encrypted traffic

→ Traffic readable by proxy?

No! Would have to fake certificates, user would be alerted

= TLS MITM attack

REST

Representational State Transfer

- Systems conforming to REST: „RESTful“

- Use RESTful APIs
  - Base URI, e.g. https://api.iaik.tugraz.at/
  - Media type, e.g. XML, JSON, ATOM, …
  - Resources represented as URIs, e.g.
    - Single person: https://api.iaik.tugraz.at/persons/123
    - All persons: https://api.iaik.tugraz.at/persons/

- Using standard HTTP methods, operations are performed on resources, e.g. create, modify, delete resources (here: persons)

RESTful API

Example: Retrieve single person

```
GET /persons/123
```


```
{
  "firstName": "John",
  "lastName": "Smith",
  "age": 25,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021"
  },
  "phoneNumber": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "fax",
      "number": "646 555-4567"
    }
  ],
  "gender": {
    "type": "male"
  }
}
```


Source: https://goo.gl/cm9GRs

More examples:

- List all persons: `GET /persons`
- Replace all persons: `PUT /persons`
- Create new person: `POST /persons`
  → URL of new entry is returned
- Delete all persons: `DELETE /persons`

- Replace or create person: `PUT /person/123`
- Delete single person: `DELETE /person/123`

HTTP/2

RFC 7540

= Semantics of HTTP/1.1 but optimized for low-latency transmission (speed)

Ideas

- Reuse core concept of HTTP (methods, status codes, header fields, etc.) but format (*frame*) the data more efficiently
  → Transfer binary data instead of text

- Address deficiencies of HTTP 1.1

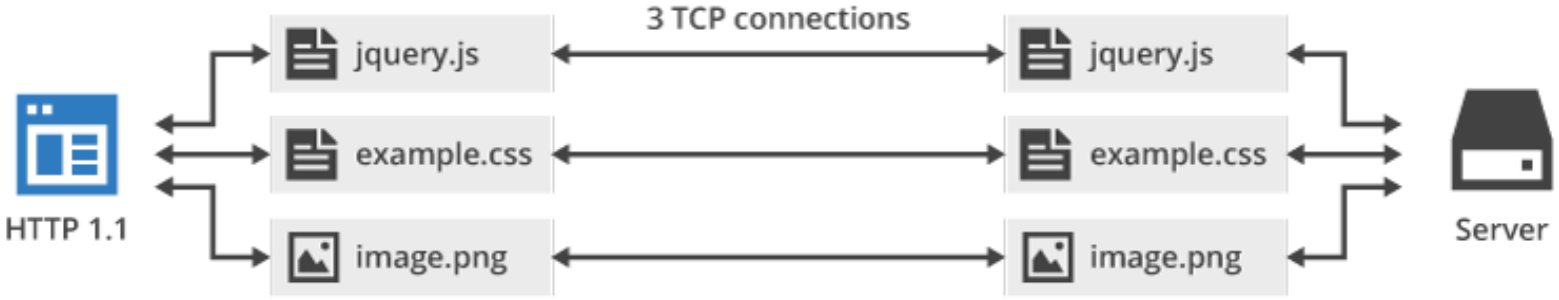
- Web pages use more and more resources (images, scripts, stylesheets)
  → Huge overhead due to multiple (sometimes parallel) requests

HTTP/2 – Why?

Because HTTP 1.x has performance problems…

- Limited parallelism
  - Request pipelining barely works in practice
  - Competing TCP flows and spurious retransmissions
- Head-of-line blocking

  See: https://goo.gl/YxgBOJ



Source: https://goo.gl/T1NJbY

- High protocol overhead
  - ~800 bytes of header + cookies
  - No compression of HTTP metadata

HTTP/2 – Features

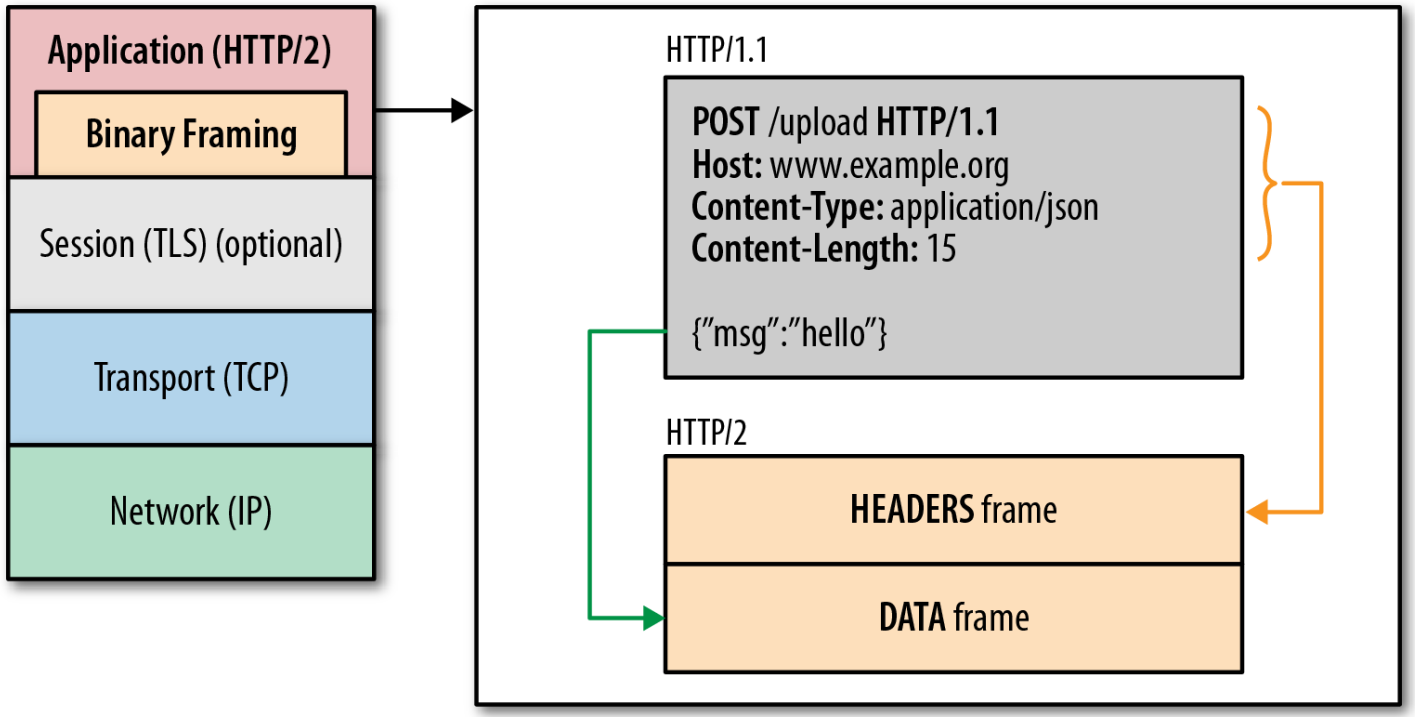
- Only one TCP connection for multiple requests
  - Responses can be out of order → reduces head-of-line blocking

- Requests become streams encapsulating *headers* and *data* frames
  - Client can prioritize streams
  - Multiplexing
    → send streams in parallel

- Header compression

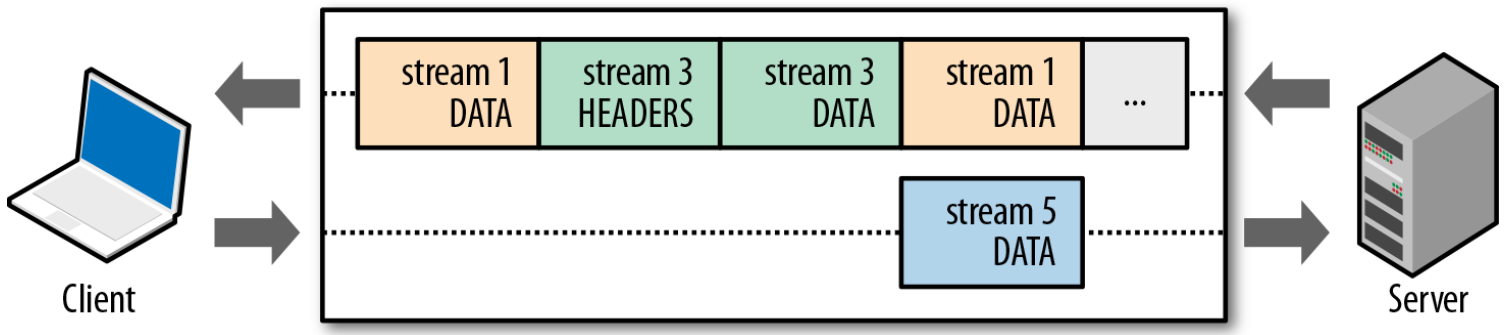
- Server Push
  = Server sends resources the client has not yet requested

Source: http://goo.gl/neMBSY



HTTP/2 – Data Flow

- Multiplexing by splitting streams into frames
  - E.g. HEADERS, DATA, etc.
- Frames can be prioritized and flow-controlled
  - E.g. client says „Please send script.js with priority 1, style.css with priority 5“
- Client can request one resource and gets multiple data „pushed“ by server



Source: http://goo.gl/neMBSY

Advanced Communication

Overview

→ Common concept in 1990s:
Retrieve complete HTML website, user reads information, follows links, all over

Problem:

Very inefficient: Bandwith consumption, delay, all information has to be present

Remedy

- AJAX: Asynchronous JavaScript and XML
  - Needs polling to get new information from server
- COMET: AJAX with long polling
  - Request remains open, server answers when data available
- WebSockets: Bi-direction communication
  - „Upgrades“ HTTP connection to negotiate a WebSocket

AJAX

Problem

- We want to reload only parts of a web page
- Asynchronously because otherwise the UI would block while loading

Evolution

- 1995: Java Applets (luckily banned from almost all browsers)
- 1996: iFrames in Internet Explorer
- 1999: ActiveX controls (XMLHTTP) by Microsoft

→ Later realized in JavaScript as `XMLHttpRequest`

AJAX

Asynchronous JavaScript and XML

- Use JavaScript to asynchronously get data from a web server via `XMLHttpRequest`
- Content retrieved in background → GUI does not block

Formats

Plain text, XML, HTTP, JSON, … basically anything that is part of HTML

Concept

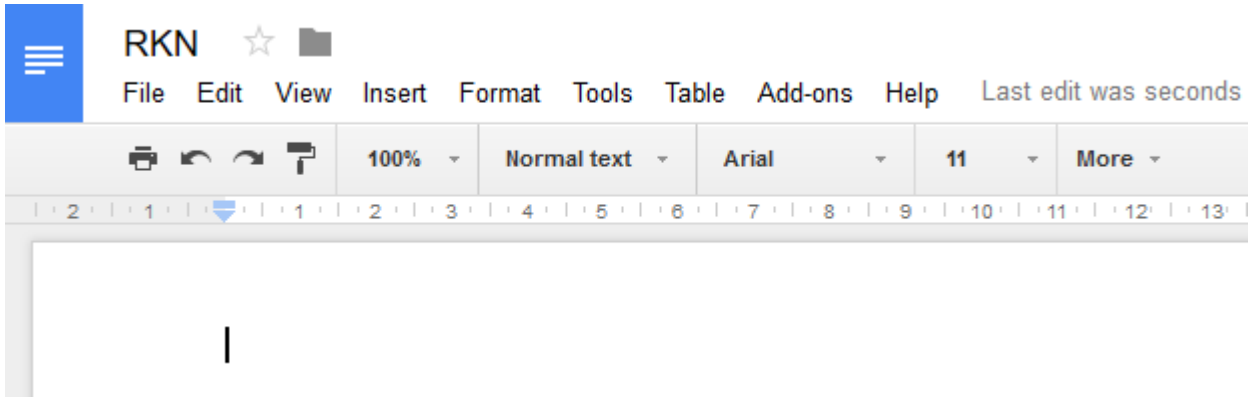
- Use data to directly modify client's DOM (Document Object Model)
  - DOM = XML or HTML document → allows accessing and manipulating objects
- Store the data for further processing

AJAX – Usability

Enabled complex web applications running in the browser…

- Widely known: Gmail (2004) and Google Maps (2005)
- Nowadays most websites and applications rely on AJAX
  - Almost every „login dialog“, live ticker, self-refreshing page, etc.
- Became a *core technology* on the web

Google Maps



Google Docs

AJAX

Edit This Code: See Result »

```
<!DOCTYPE html>
<html>
<body>
<h2>AJAX</h2>
<button type="button" onclick="loadDoc()">Request data</button>
<p id="demo"></p>

<script>
function loadDoc() {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (xhttp.readyState == 4 && xhttp.status == 200) {
      document.getElementById("demo").innerHTML =
xhttp.responseText;
    }
  };
  xhttp.open("GET", "demo_get.asp", true);
  xhttp.send();
}
</script>

</body>
</html>
```

Result:

AJAX

Request data

This content was requested using the GET method.

Requested at: 3/6/2016 3:46:37 PM

Try it yourself (and activate Wireshark!): https://goo.gl/Z4TRd2

AJAX

We are just looking for this:

```
<p>This content was requested using the GET method.</p>
<p>Requested at: 3/6/2016 3:46:37 PM</p>
```

Wireshark (without / with gzip):

```
GET /ajax/demo_get.asp HTTP/1.1
Host: www.w3schools.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:44.0) Gecko/20100101
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: de,en-US;q=0.7,en;q=0.3
Accept-Encoding: deflate
DNT: 1
Referer: http://www.w3schools.com/ajax/tryit.asp?filename=tryajax_get
Cookie: ASPSESSIONIDQQAQTBCQ=CKMFMGMBIBANFKOENLKFMFFL
Connection: keep-alive

HTTP/1.1 200 OK
Cache-Control: private,public
Content-Type: text/html
Date: Sun, 06 Mar 2016 20:46:36 GMT
Server: Microsoft-IIS/7.5
X-Powered-By: ASP.NET
Content-Length: 97

<p>This content was requested using the GET method.</p>
<p>Requested at: 3/6/2016 3:46:37 PM</p>
```

```
GET /ajax/demo_get.asp HTTP/1.1
Host: www.w3schools.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:44.0) Gecko/20100101 Firefox/44.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: de,en-US;q=0.7,en;q=0.3
Accept-Encoding: gzip, deflate
DNT: 1
Referer: http://www.w3schools.com/ajax/tryit.asp?filename=tryajax_get
Cookie: ASPSESSIONIDQQAQTBCQ=CKMFMGMBIBANFKOENLKFMFFL
Connection: keep-alive

HTTP/1.1 200 OK
Content-Encoding: gzip
Cache-Control: private,public
Content-Type: text/html
Date: Sun, 06 Mar 2016 20:42:26 GMT
Server: Microsoft-IIS/7.5
Vary: Accept-Encoding
X-Powered-By: ASP.NET
Content-Length: 199

.............`.I.%&/m.{.J.J..t...`.$..@.........iG#).*..eVe]f.@.....{....{....;.N'...?
\fd.l..J...!....?~|.?".....&.V.6_..U..u...y...t.........o.E.........o....l..}.........O.
{.....=H_~........a...
```

IAIK TU Graz

AJAX

- Preceding TCP build-up / teardown

- HTTP 1.1 GET Request

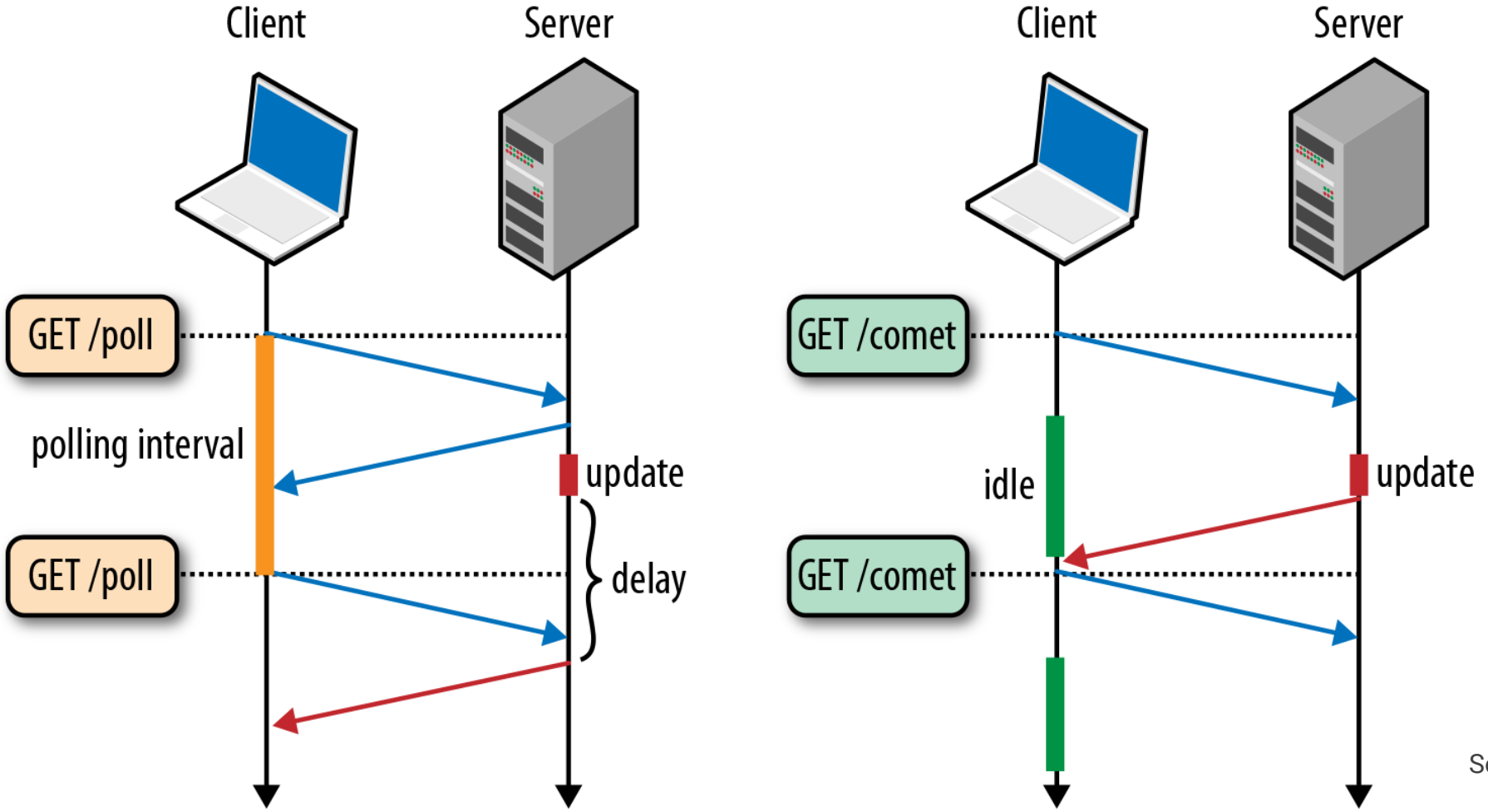
Protocol	Length	Info
TCP	66	57658 → 80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
TCP	66	80 → 57658 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1420 SACK_PERM=1 WS=512
TCP	54	57658 → 80 [ACK] Seq=1 Ack=1 Win=66560 Len=0
HTTP	505	GET /ajax/demo_get.asp HTTP/1.1
TCP	60	80 → 57658 [ACK] Seq=1 Ack=452 Win=147456 Len=0
HTTP	333	HTTP/1.1 200 OK (text/html)
TCP	54	57658 → 80 [ACK] Seq=452 Ack=280 Win=66304 Len=0

Problems

- Client still needs to poll server for updates periodically
- New TCP/IP connections for AJAX HTTP requests (HTTP is stateless)
- Protocol overhead

COMET – Long Polling

Similar to XMLHTTPRequest but request remains open until data available



Source: http://goo.gl/uZnMRR

WebSockets

RFC 6455

Long-lived TCP connection between server and client

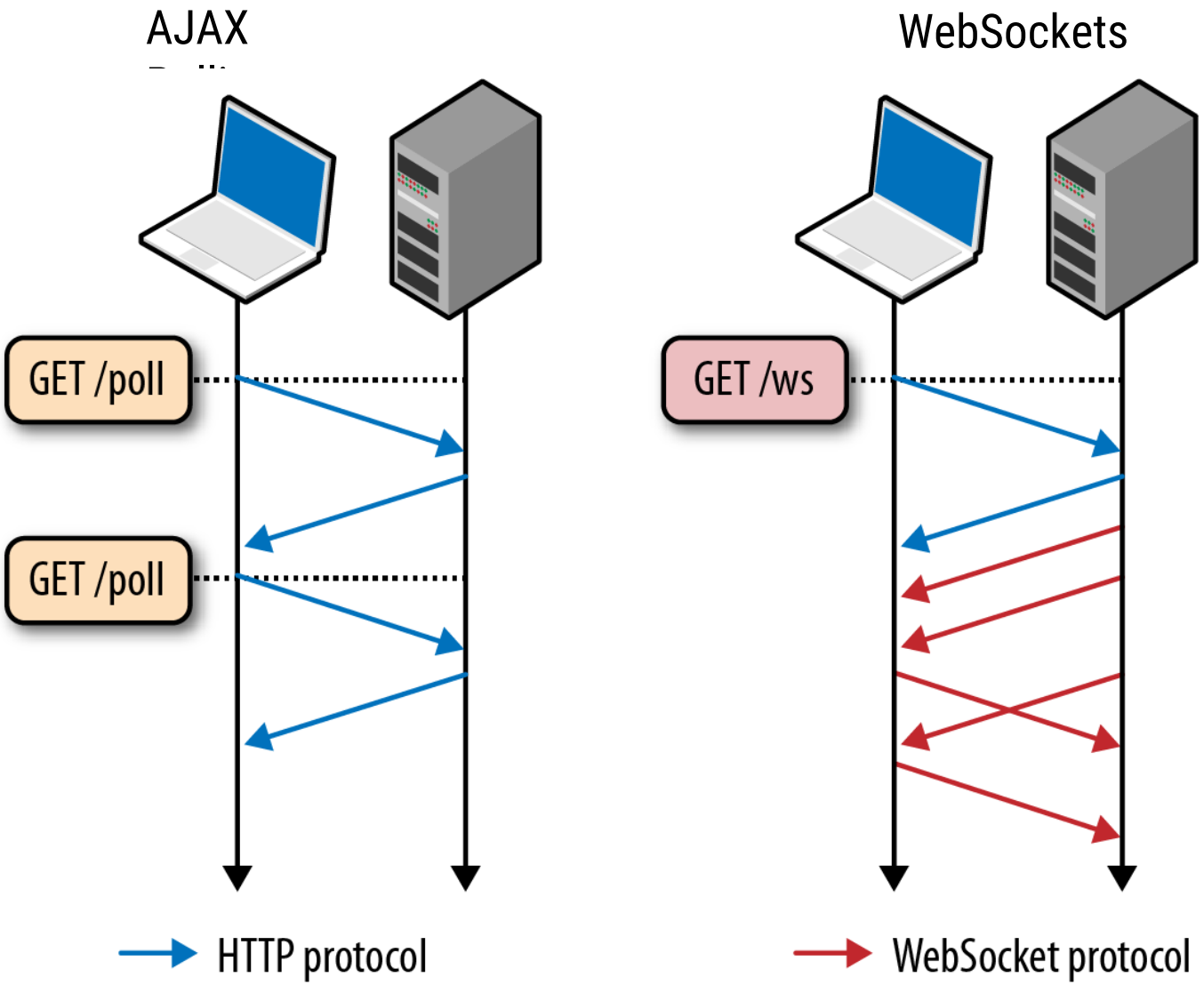
- Advantages
  - Enables bi-directional communication
  - When data is sent → much less overhead, no HTTP protocol headers needed
  - Server can send („push“) data to client without waiting for poll request from client

- Protocol Handshake: Client upgrades HTTP connection to WebSocket

URI Schemes

- For plain-text communication: `ws://example.com/socket`
- For encrypted channel (TCP+TLS): `wss://example.com/socket`

Comparison

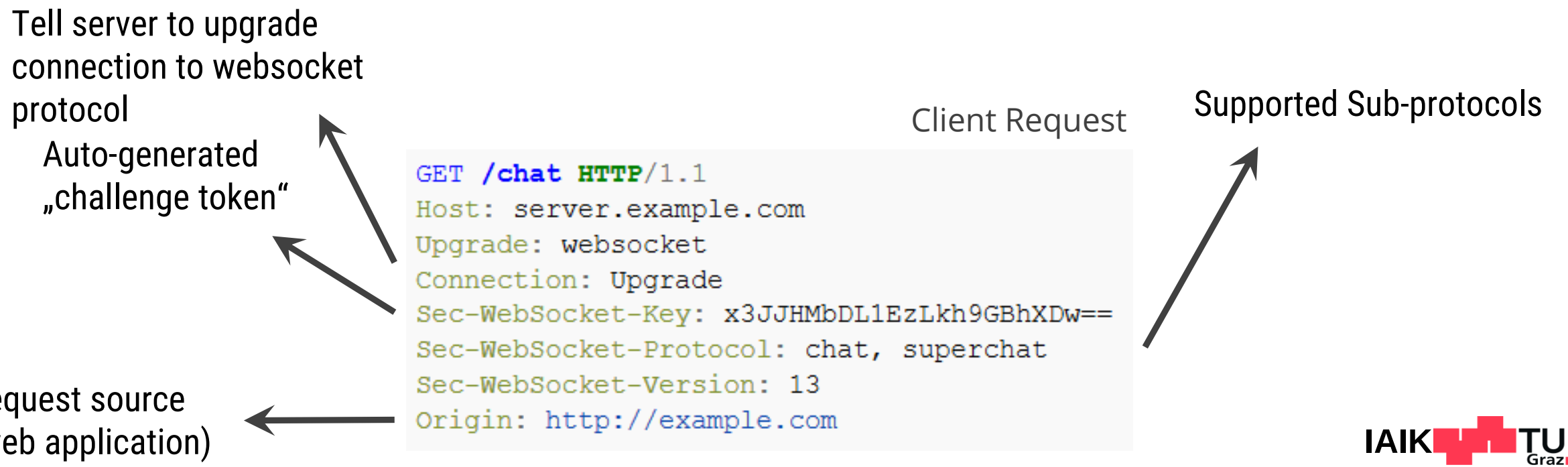


Source: http://goo.gl/cF5tL8

WebSockets

- Starts with protocol handshake
  - HTTP GET request on port 80 or 443
  - Client upgrades HTTP connection to WebSocket

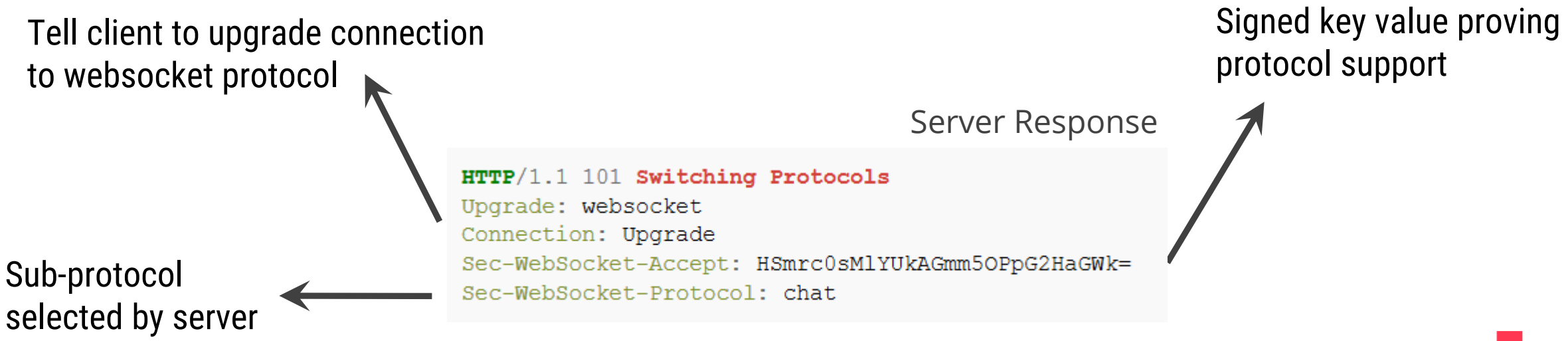
Structure



WebSockets

- Server completes handshake with „Switching Protocols“
  - Status code 101
  - Confirms selected options, advertised by client
  - Now, connection can be used as two-way communication channel (no more HTTP)

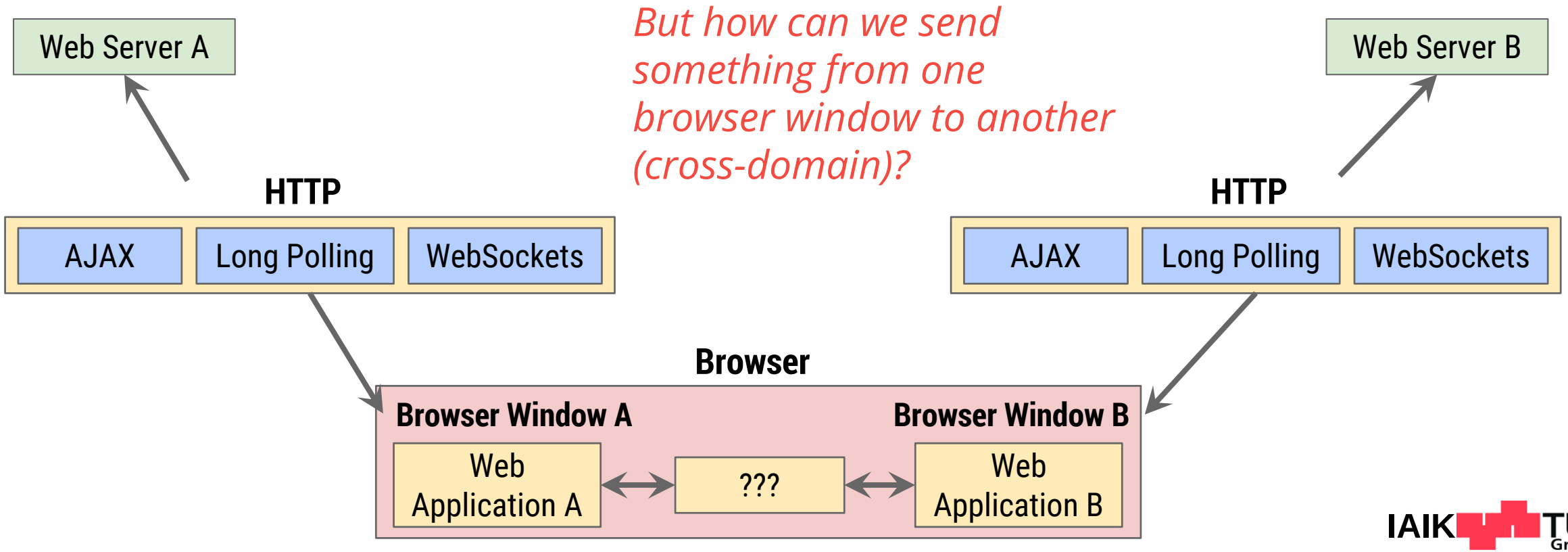
Structure



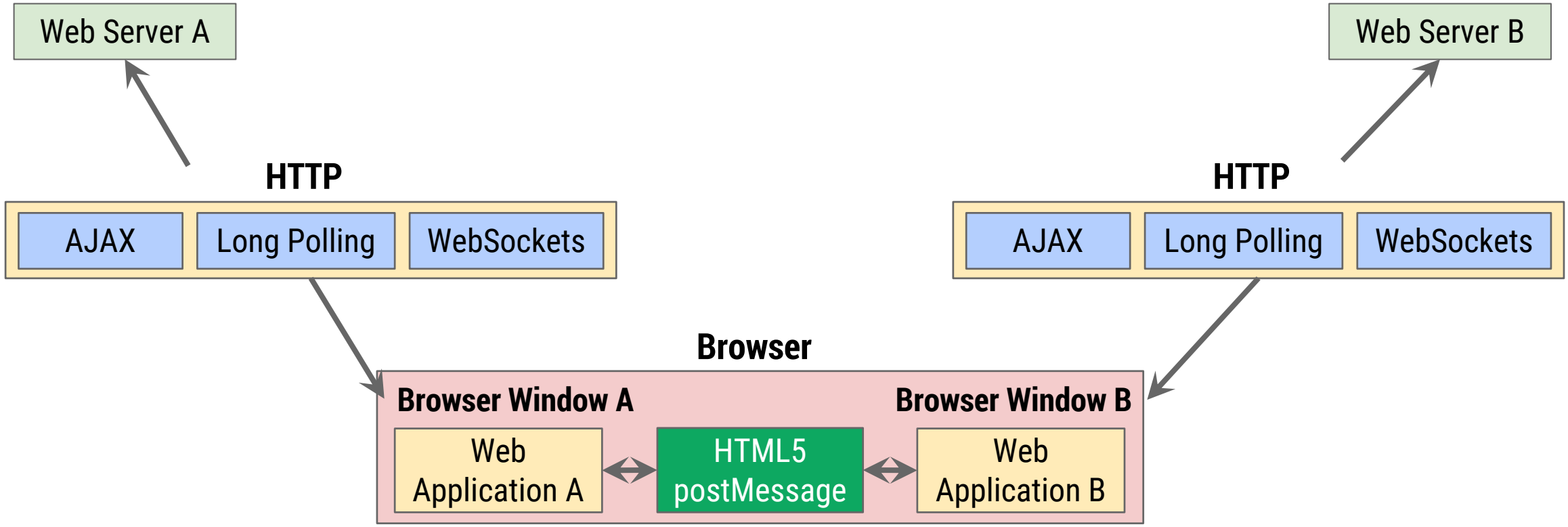
Communication

Status quo

- AJAX, COMET and WebSockets can only access resources on locations with same protocol (e.g. https), port (e.g. 443), and domain



Communication




```
otherWindow.postMessage(message, targetOrigin);
```

HTML5 postMessage

Allows for sending data between two windows / frames across domains securely

Great reference: https://developer.mozilla.org/en-US/docs/Web/API/Window/postMessage

Why is it needed?

- Enables to send plain text messages from one window to another
  → Imagine page with chat application in iframe

- Frames run separated in their own browser window / sandbox
  - Want to address other frames in same sender window
  - Windows opened by JavaScript calls

- Prior to HTML5, cross-domain scripting was not possible due to SOP
  → Still to consider security aspects!

HTML5 postMessage

```
/*
 * In window A's scripts, with A being on <http://example.com:8080>:
 */

var popup = window.open(...popup details...);

// When the popup has fully loaded, if not blocked by a popup blocker:

// This does nothing, assuming the window hasn't changed its location.
popup.postMessage("The user is 'bob' and the password is 'secret'",
                  "https://secure.example.net");

// This will successfully queue a message to be sent to the popup, assuming
// the window hasn't changed its location.
popup.postMessage("hello there!", "http://example.org");
```

Example

We want a document A on `tugraz.at` to talk to document B on `iaik.at` in iframe

```
var o = document.getElementsByTagName('iframe')[0];
o.contentWindow.postMessage('Hello CON', 'https://iaik.at/dest.php');
```

HTML5 postMessage

Window A has sent a message, how to receive it in window B (securely)?

- Receiver gets 3 message fields
  - **Data**: The content of the incoming message
  - **Origin**: Window that sent the message in the format scheme://host:port, e.g. https://tugraz.at
  - **Source**: Reference to source window. Can i.e. used to answer back to this window

Security?

- Client: Do not specify * as target origin
  - Malicious site could change location of window → intercept your message!
- Receiver: Always check the sender‘s origin!
  - Any window can send messages to other windows → could be malicious message!

HTML5 postMessage

```
/*
 * In the popup's scripts, running on <http://example.org>:
 */

// Called sometime after postMessage is called
function receiveMessage(event)
{
  // Do we trust the sender of this message?
  if (event.origin !== "http://example.com:8080")
    return;

  // event.source is window.opener
  // event.data is "hello there!"

  // Assuming you've verified the origin of the received message (which
  // you must do in any case), a convenient idiom for replying to a
  // message is to call postMessage on event.source and provide
  // event.origin as the targetOrigin.
  event.source.postMessage("hi there yourself!  the secret response " +
                           "is: rheeeeet!",
                           event.origin);
}

window.addEventListener("message", receiveMessage, false);
```

In our example…

```
if (event.origin !== ‘tugraz.at’) {
  return;
}

alert(origin.data);
```

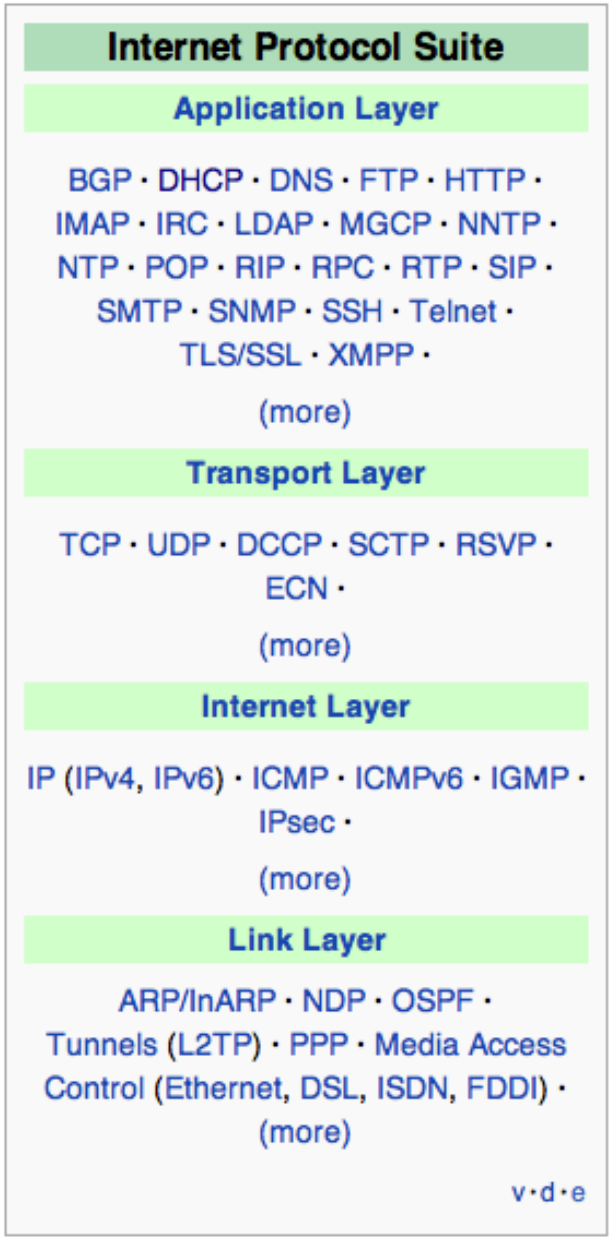
IAIK TU Graz

DNS

Introduction

Basic problem

- Users want to reach servers at www.tugraz.at
  - Hostnames independent of server location in network
- Domains could map to multiple addresses
  - E.g., www.amazon.com points to at least 3 IP addresses
    - Load balancing, latency reduction
    - Different destination based on location / device / identity
  - Or assign both IPv4 and IPv6 addresses to domains
- Want to reuse 1 IP address for multiple domain names
  - E.g., tu4u.tugraz.at + tugraz.at both point to same IP



History

/etc/hosts still exists!

Once upon a time…

- All host addresses mapped in a local file named *hosts.txt*

```
129.27.2.244          tugraz.at
129.27.142.148        teaching.iaik.tugraz.at
...
```

  - Flat namespace without structure
  - Central administrator (NIC) kept master copy for entire network (later *INTERnet*)
    - Add/remove/update mapping → send email to global admin
    - Clients had to re-fetch the file recurringly

- **Practical today? No!**
  - Some names change mappings every few days, e.g. dynamic IP addresses
  - Single Point of Failure

Goals

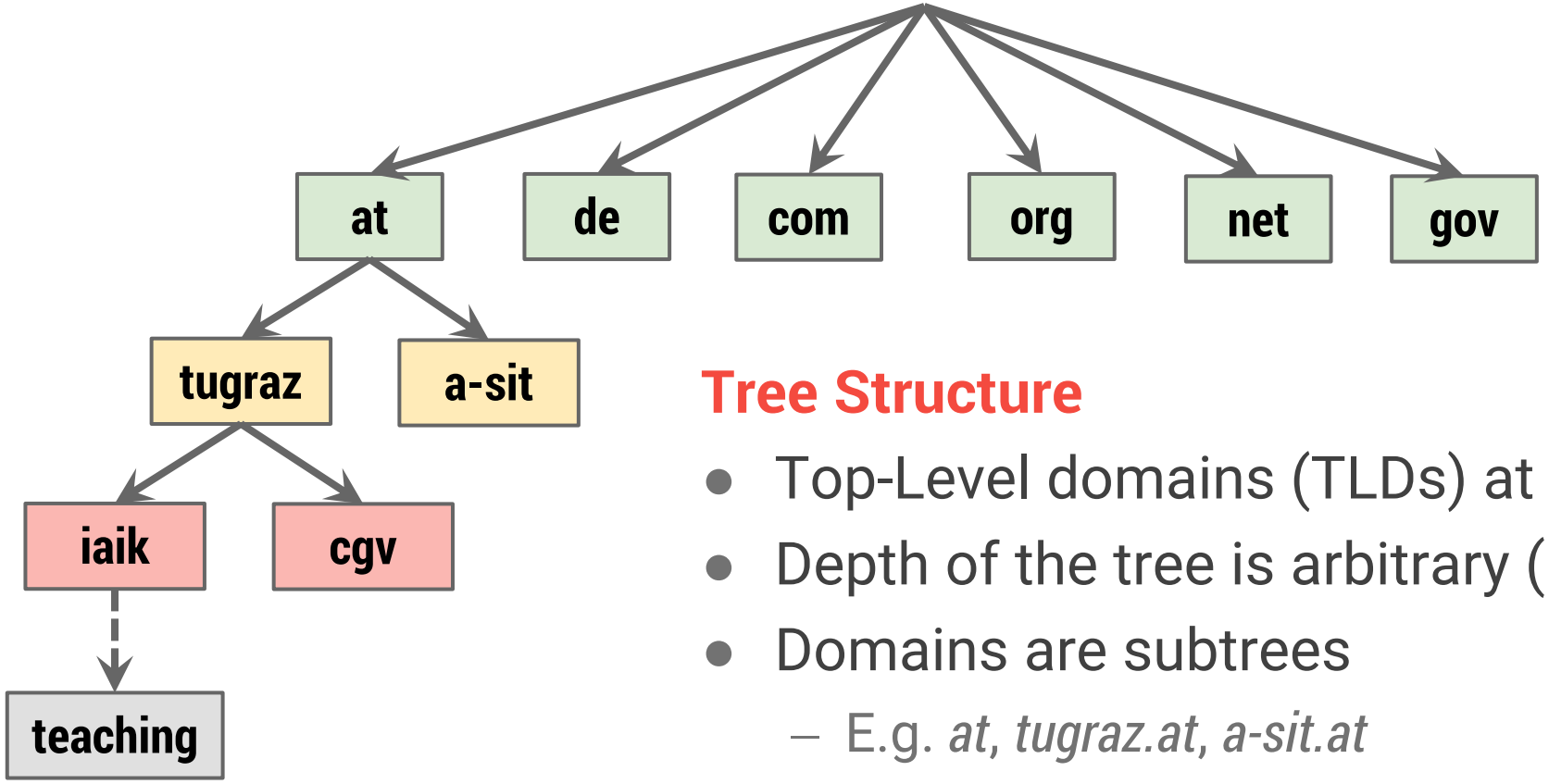
for a world-wide DNS system

- Scalability
  - Must handle large number of (new) records
  - Must sustain high update frequency and lookup load
- Distributed control
  - People want to control their own domain names
    → decentralized management needed
- Fault Tolerance
  - Robust against attacks
  - Minimize lookup failures and duplicate names

Introduction

RFC 1035

Domain Name Service (DNS)



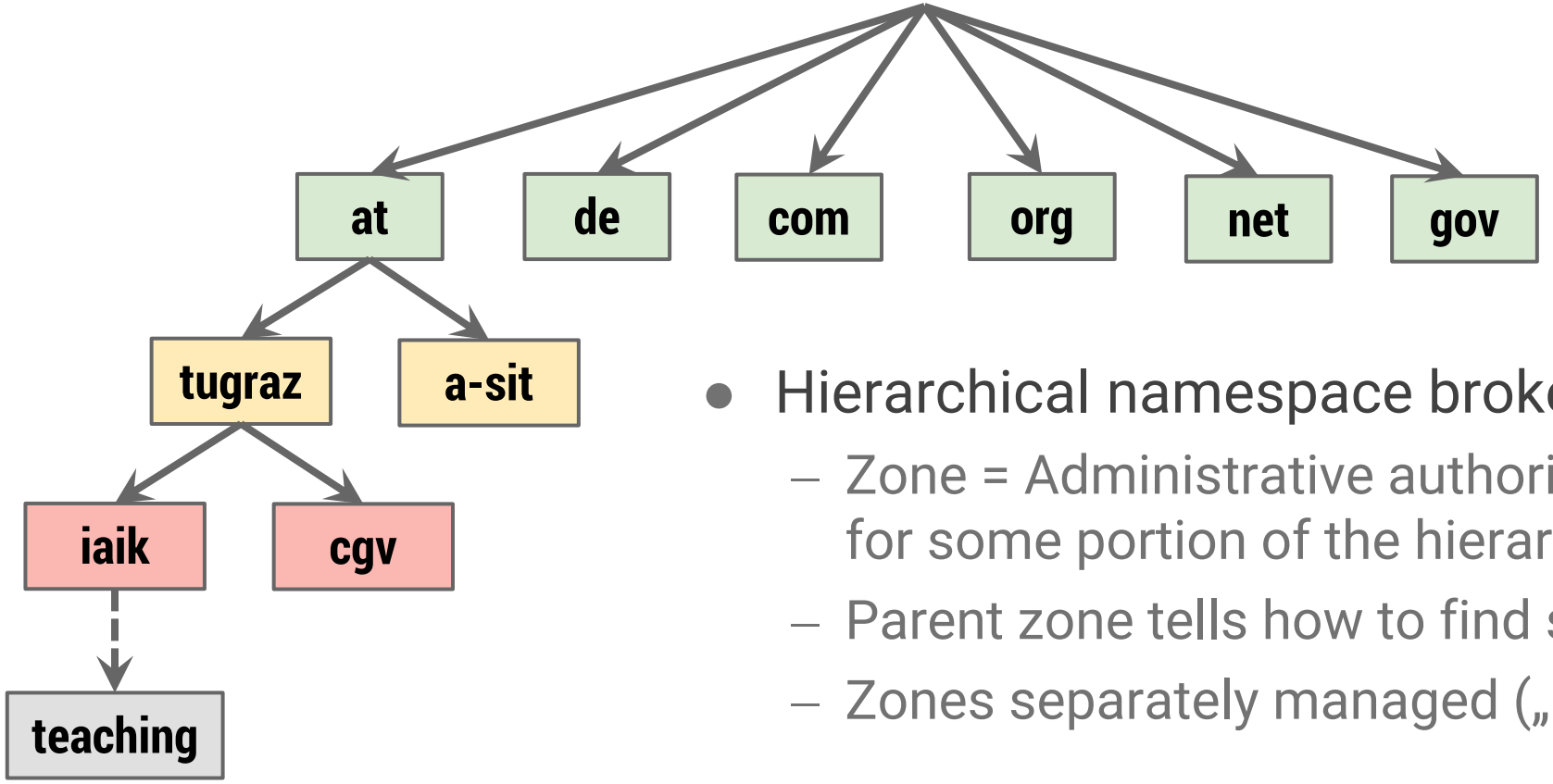
Tree Structure

- Top-Level domains (TLDs) at the top
- Depth of the tree is arbitrary (limit: 128 entries)
- Domains are subtrees
  - E.g. *at*, *tugraz.at*, *a-sit.at*
- Name collisions avoided
  - E.g. *tugraz.at* and *tugraz.org* can co-exist

Introduction

RFC 1035

Domain Name Service (DNS)



- Hierarchical namespace broken into **zones**
  - Zone = Administrative authority responsible for some portion of the hierarchy
  - Parent zone tells how to find servers for subdomain
  - Zones separately managed („Delegation“)

- Typically zones are replicated to multiple servers, e.g. *ns1.dnszone.at, ns2.dnszone.at*

DNS Messages

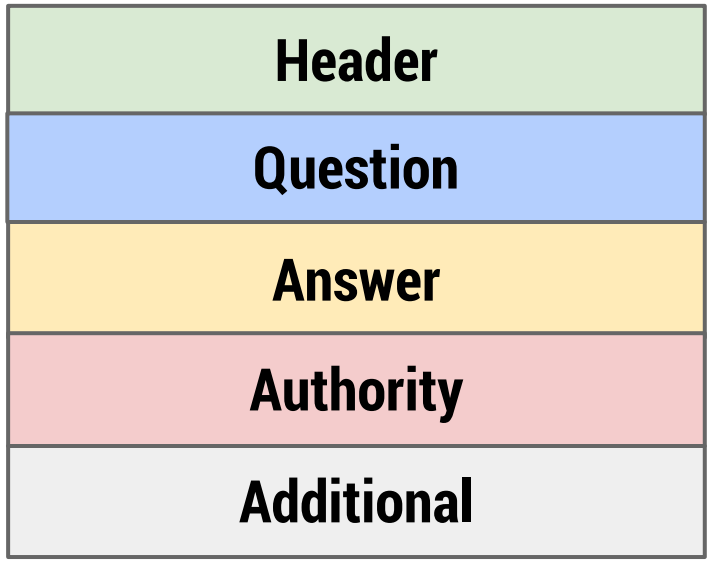
Protocol

Very simple!

- Only two message types in same format: **Query & Reply**
- For transport, DNS uses primarily UDP, servers run on well-known port 53

Message format

→ Always 5 sections in DNS message



Specifies whether query or reply, number of questions, answers, …

Contains „Resource Records“ (RR) answering the question

RR pointing towards an authority („zone managers“) and additional RRs, e.g. IP addresses of authorities

Resource Records

= Basic information element in DNS system

RR format: (Class, Name, Value, Type, TTL)

Example

Name	TTL	Class	Type	Data
orf.at.	86400	IN	A	194.232.104.139
orf.at.	86400	IN	A	194.232.104.141
orf.at.	86400	IN	AAAA	2a01:468:1000:9::149
orf.at.	86400	IN	MX	10 orfmx01.t-systems.at.
orf.at.	86400	IN	NS	ns1.apa.net
orf.at.	86400	IN	NS	ns2.apa.net

TTL (Time-to-live)

Maximum time a RR can be cached / reused by non-authoritative server

Resource Records

Mostly used…

Type	Code	Description	Function
A	1	Address record	32-bit IPv4 address associated with host
AAAA	28	IPv6 address record	128-bit IPv6 address
CNAME	5	Canonical name record	Alias of one domain name to another
MX	15	Mail exchange record	Domain name of mail server for this domain
NS	2	Name server record	Delegates DNS zone to use the given authoritative name servers
PTR	12	Pointer record	Pointer to a CNAME entry
SOA	6	Start of [a zone of] authority record	Authoritative information about DNS zone: Primary name server, email of the domain admin, domain serial number, …
TXT	16	Text record	Plain text info

For more codes, see https://goo.gl/AJIPEd

DNS Query

Wireshark Example

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	2a02:8388:e301:6…	2001:4860:4860::8888	DNS	103	Standard query 0x7065 A teaching.iaik.tugraz.at
2	0.000146	2a02:8388:e301:6…	2001:4860:4860::8888	DNS	103	Standard query 0xf59f AAAA teaching.iaik.tugraz.at

```
> Internet Protocol Version 6, Src: 2a02:8388:e301:6                    , Dst: 2001:4860:4860::8888
> User Datagram Protocol, Src Port: 64156 (64156), Dst Port: 53 (53)
v Domain Name System (query)
    [Response In: 5]
    Transaction ID: 0x7065
  v Flags: 0x0100 Standard query
      0... .... .... .... = Response: Message is a query
      .000 0... .... .... = Opcode: Standard query (0)
      .... ..0. .... .... = Truncated: Message is not truncated
      .... ...1 .... .... = Recursion desired: Do query recursively
      .... .... .0.. .... = Z: reserved (0)
      .... .... ...0 .... = Non-authenticated data: Unacceptable
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
  v Queries
    > teaching.iaik.tugraz.at: type A, class IN
```

DNS Reply

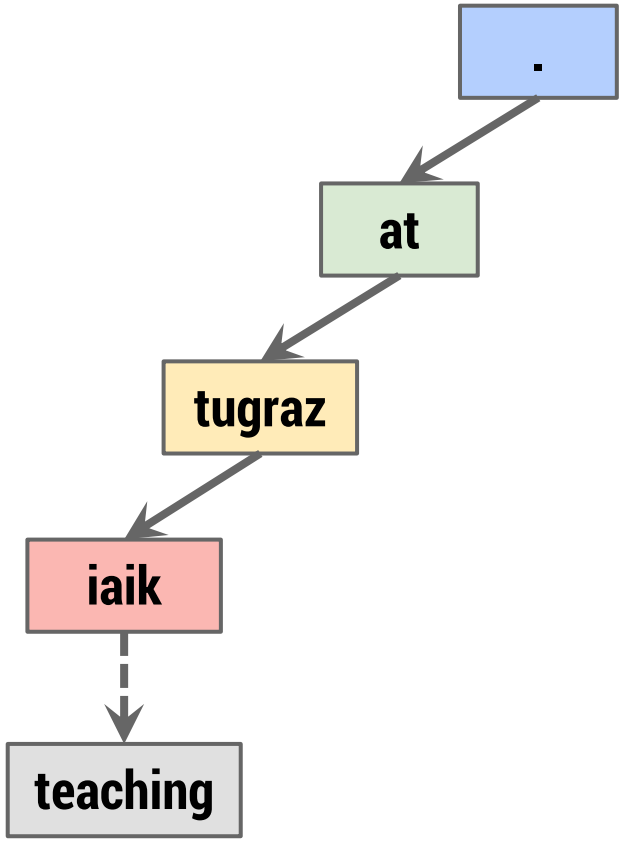
Wireshark Example

No.	Time	Source	Destination	Protocol	Length	Info
5	0.058755	2001:4860:4860::8888	2a02:8388:e301:6…	DNS	119	Standard query response 0x7065 A teaching.iaik.tugraz.at A 129.27.142.148
6	0.061294	2001:4860:4860::8888	2a02:8388:e301:6…	DNS	148	Standard query response 0xf59f AAAA teaching.iaik.tugraz.at SOA ns.iaik.tugraz.at

```
> Internet Protocol Version 6, Src: 2001:4860:4860::8888, Dst: 2a02:8388:e301:6
> User Datagram Protocol, Src Port: 53 (53), Dst Port: 64156 (64156)
v Domain Name System (response)
    [Request In: 1]
    [Time: 0.058755000 seconds]
    Transaction ID: 0x7065
  v Flags: 0x8180 Standard query response, No error
      1... .... .... .... = Response: Message is a response
      .000 0... .... .... = Opcode: Standard query (0)
      .... .0.. .... .... = Authoritative: Server is not an authority for domain
      .... ..0. .... .... = Truncated: Message is not truncated
      .... ...1 .... .... = Recursion desired: Do query recursively
      .... .... 1... .... = Recursion available: Server can do recursive queries
      .... .... .0.. .... = Z: reserved (0)
      .... .... ..0. .... = Answer authenticated: Answer/authority portion was not authenticated by the server
      .... .... ...0 .... = Non-authenticated data: Unacceptable
      .... .... .... 0000 = Reply code: No error (0)
    Questions: 1
    Answer RRs: 1
    Authority RRs: 0
    Additional RRs: 0
  v Queries
    > teaching.iaik.tugraz.at: type A, class IN
  v Answers
    > teaching.iaik.tugraz.at: type A, class IN, addr 129.27.142.148
```

DNS Components

DNS Architecture



Hierarchy of DNS servers (= „Name servers“)

- Root servers
- Top-Level Domain (TLD) servers
  - Controls everything within .at, .de, … namespace
- Authoritative DNS servers
  - Manage individual zones consisting of one or many domains & subdomains
  - Responsibility for administration „delegated“ from parent zone

How to resolve domain names?

- Local DNS servers
- Resolver software

Root Servers

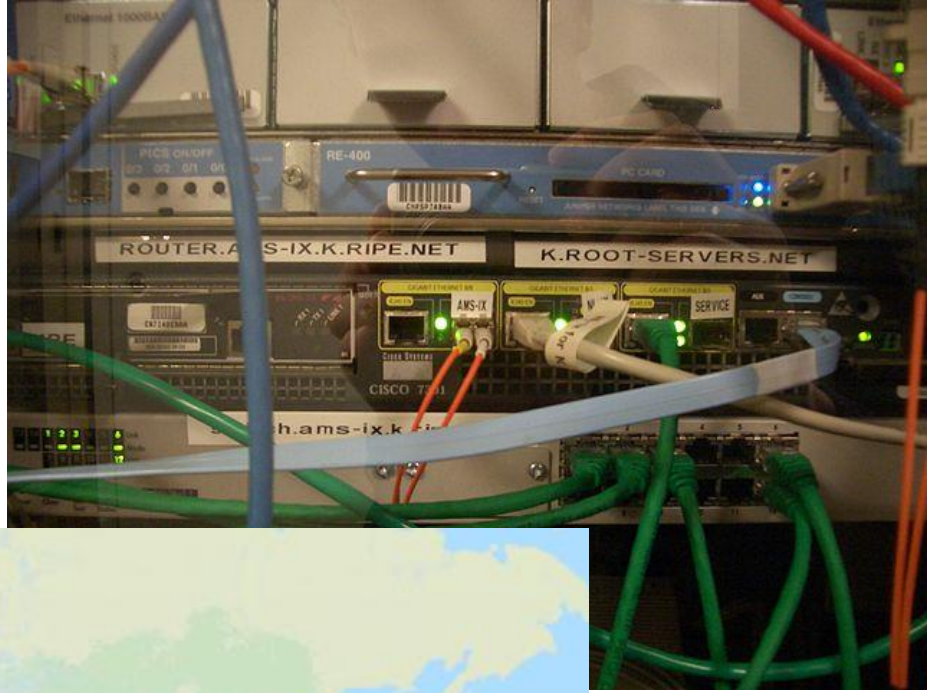
- Responsible for the root domain
  - Return authoritative name servers for specific TLDs
  - With a single root DNS server, all other DNS info could be discovered recursively
  - 13 logical name servers: *a.root-servers.net*, …, *m.root-servers.net*

A Dulles, VA (Verisign)
C Herndon, VA (Cogent)
D College Park, MD (UM)
G Vienna, VA (US DoD)
H Aberdeen, MD (ARL)
J Dulles, VA (Verisign)

B Marina del Rey, CA (USC-ISI)
E Mt. View, CA (NASA)
F Palo Alto, CA (ISC)
L Los Angeles, CA (ICANN)

Root Servers

Only 13 physical servers? **No!**



Source: https://goo.gl/hYGgQE

Replication using **Anycasting** (see IPv4 slides)

Source: http://goo.gl/tnXKV3

Root Servers

How do local servers find root servers?

- Reachable at a.root-servers.net, b.root-servers.net, …
  - Get their IP addresses via DNS lookup? Not feasible obviously…

- DNS servers configured with „root hints file“
  - For bootstrapping DNS resolution
  - Can be updated periodically by admin, e.g. upon restart of service
  - Contains root name servers + their IP addresses

```
.                       3600000    NS a.root-servers.net.
a.root-servers.net.     3600000     A 198.41.0.4
a.root-servers.net.     3600000  AAAA 2001:503:ba3e::2:30
...
```

Source: https://goo.gl/8lvccy

Top-Level Domains (TLDs)

= Domains at highest level of DNS system

Multiple types

- Generic domains (gTLD)
  - Unsponsered TLDs: *com, info, net, org*
  - Sponsered TLDs: Intended for specific community, e.g. ethnic, geographic, ...
    E.g. *.aero, .asia, .cat, .gov, .mil, .jobs, .mobi, .museum, .tel, .travel, …*
- Country domains (ccTLD)
  - *.at, .de, .au, .fr, .it, .pt, .ua, …*
- Special domains: *.arpa, .example, .invalid, .localhost, .test, …*

Note: Depending on TLD, one or multiple registrars for each TLD

E.g., all .at domains are ultimately registered at www.nic.at

Name Servers

= Server that provides domain name resolution <-> IP

Authoritative server

- Responsible for a zone, e.g. *.at* or *.iaik.tugraz.at*
- At least one server / zone („*primary name server*“) → usually redundant cluster with identical zone files on multiple servers

Non-authoritative server

- Gets information about domains from other servers *recursively* or *iteratively*

- Responses often stored in local cache until time-to-live (TTL) value reached
  → *Enables faster responses, no need to go through all servers in tree!*

Name Servers

How do they get information from other servers?

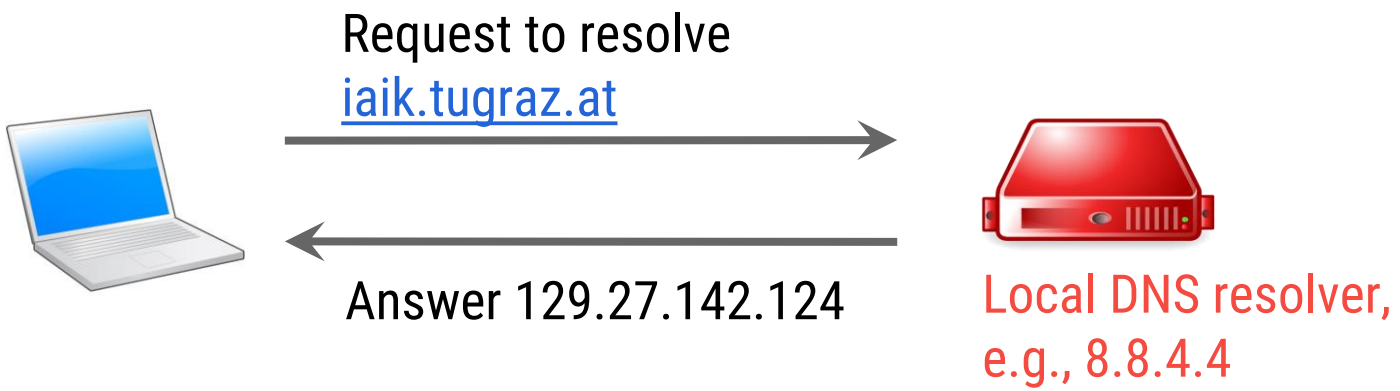
- Delegation
  - Parts of domains are often moved to other name servers in subdomains
    - E.g., a.root-servers.net says: to obtain the IP address of iaik.tugraz.at, ask d.ns.at
  - **Q:** Now, how do you find d.ns.at?
    A: The parent zone has „glue records“ with the IP address(es) of d.ns.at

- Forwarding
  - If requested name space is outside of own domain
    → forward query to another configured server

- Resolution via Root Servers
  - If request cannot be forwarded → ultimately ask at highest level

Address Resolution

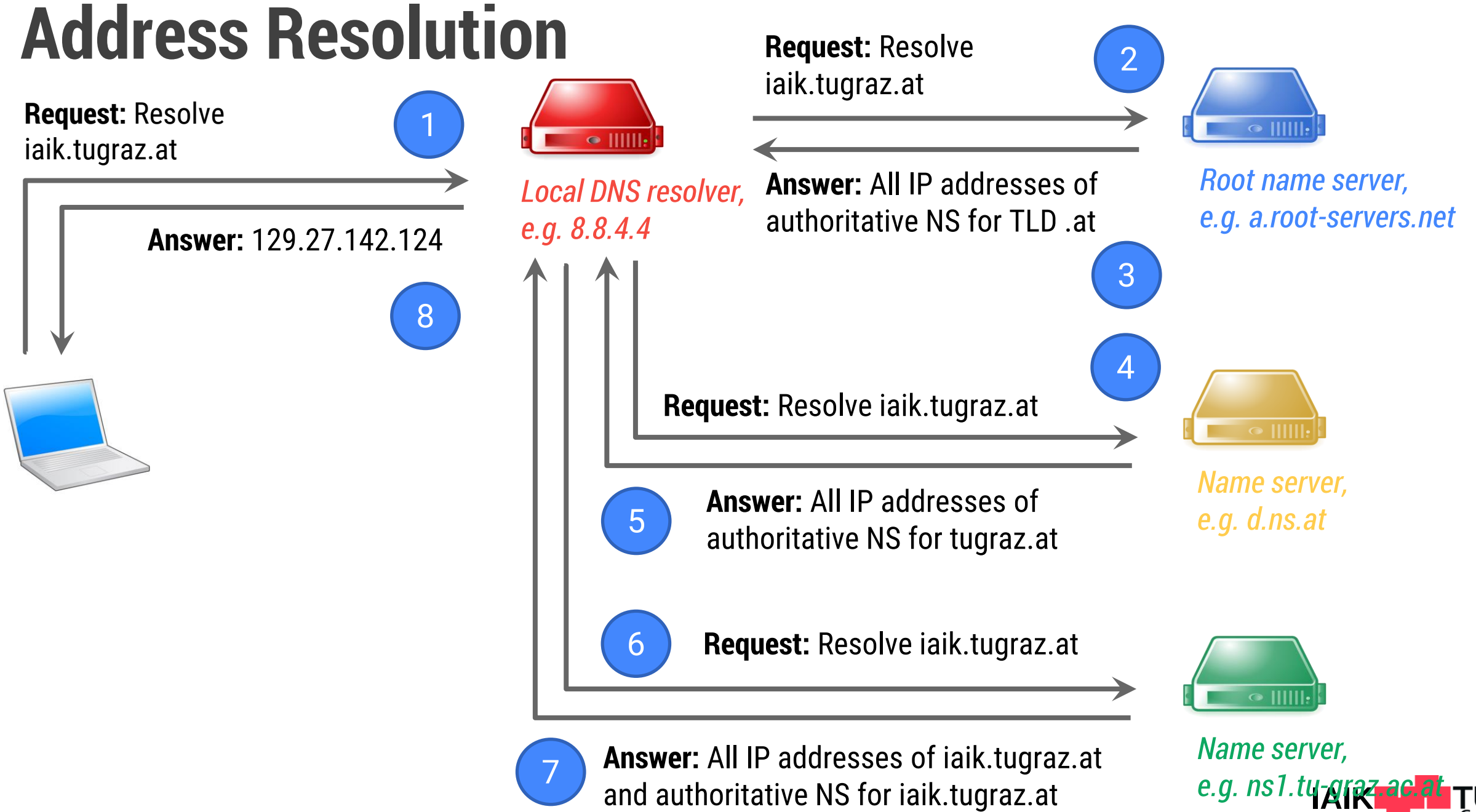
Example: A host wants the IP address of iaik.tugraz.at



How?

- Host sends DNS request (UDP, port 53) to local name server
- What does the nameserver if it does not know the requested domain?
  → Send request to further name server („*recursive query*“)
- Each name server knows about higher-level name servers
- **Only** lowest level server (local resolver) gives answer to host!

Address Resolution



Address Resolution

Client asks local DNS resolver 8.8.4.4

```
dig iaik.tugraz.at @8.8.4.4
;; QUESTION SECTION:
;iaik.tugraz.at.                        IN      A

;; ANSWER SECTION:
iaik.tugraz.at.         3599    IN      A       129.27.142.24

;; Query time: 13 msec
;; SERVER: 8.8.4.4#53(8.8.4.4)
```

→ Client sends a „recursive query“ to 8.8.4.4

- Ask server to get answer for you
- 8.4.4.4 is not authoritative for iaik.tugraz.at → needs to get IP from other NS

Address Resolution

```
dig +norec iaik.tugraz.at @a.root-servers.net
;; QUESTION SECTION:
;iaik.tugraz.at.                        IN      A

;; AUTHORITY SECTION:
at.                     172800  IN      NS      d.ns.at.
at.                     172800  IN      NS      j.ns.at.
at.                     172800  IN      NS      n.ns.at.
at.                     172800  IN      NS      r.ns.at.
at.                     172800  IN      NS      u.ns.at.
at.                     172800  IN      NS      ns1.univie.ac.at.
at.                     172800  IN      NS      ns2.univie.ac.at.
at.                     172800  IN      NS      ns9.univie.ac.at.

;; ADDITIONAL SECTION:
d.ns.at.                172800  IN      A       81.91.161.98
d.ns.at.                172800  IN      AAAA    2a02:568:20:1::d
j.ns.at.                172800  IN      A       194.146.106.50
...
```

DNS resolver queries root DNS server

Resolver sends **iterative** queries to remote servers

- Ask servers which NS to ask next
- Cache results aggressively

Address Resolution

```
dig +norec iaik.tugraz.at @d.ns.at
;; QUESTION SECTION:
;iaik.tugraz.at.                        IN      A

;; AUTHORITY SECTION:
tugraz.at.              10800   IN      NS      ns1.tu-graz.ac.at.
tugraz.at.              10800   IN      NS      ns2.tu-graz.ac.at.
tugraz.at.              10800   IN      NS      ns5.univie.ac.at.

;; Query time: 4 msec
;; SERVER: 2a02:568:20:1::d#53(2a02:568:20:1::d)
```

DNS resolver asks d.ns.at

- Resolver learned that d.ns.at is responsible for .at domains
- Answer contains reference to servers managing tugraz.at
  - ns1.tu-graz.ac.at, ns2.tu-graz.ac.at, ns5.univie.ac.at

Address Resolution

```
dig +norec ns1.tu-graz.ac.at @d.ns.at
;; QUESTION SECTION:
;ns1.tu-graz.ac.at.              IN      A

;; AUTHORITY SECTION:
tu-graz.ac.at.          10800   IN      NS      ns10.univie.ac.at.
tu-graz.ac.at.          10800   IN      NS      ns5.univie.ac.at.
tu-graz.ac.at.          10800   IN      NS      ns1.tu-graz.ac.at.
tu-graz.ac.at.          10800   IN      NS      ns2.tu-graz.ac.at.

;; ADDITIONAL SECTION:
ns1.tu-graz.ac.at.      10800   IN      A       129.27.2.3
...

;; Query time: 3 msec
;; SERVER: 2a02:568:20:1::d#53(2a02:568:20:1::d)
```

Why?

In order to ask *ns1.tu-graz.ac.at*, we need to know its IP addresses!

Address Resolution

```
dig +norec iaik.tugraz.at @ns1.tu-graz.ac.at
;; QUESTION SECTION:
;iaik.tugraz.at.                        IN      A

;; ANSWER SECTION:
iaik.tugraz.at.         3600    IN      A       129.27.142.24

;; AUTHORITY SECTION:
iaik.tugraz.at.         3600    IN      NS      ns1.tu-graz.ac.at.
iaik.tugraz.at.         3600    IN      NS      ns2.tu-graz.ac.at.
iaik.tugraz.at.         3600    IN      NS      ns.iaik.tugraz.at.

;; ADDITIONAL SECTION:
ns.iaik.tugraz.at.      3600    IN      A       129.27.142.23
ns1.tu-graz.ac.at.      3600    IN      A       129.27.2.3
ns2.tu-graz.ac.at.      3600    IN      A       129.27.3.3

;; Query time: 1 msec
;; SERVER: 129.27.2.3#53(129.27.2.3)
```

DNS resolver finally asks ns1.tu-graz.ac.at

- Indicates IP address of iaik.tugraz.at
- Returns authoritative name server for zone iaik.tugraz.at

The used DNS resolver 8.8.4.4 can now reply the IP address of iaik.tugraz.at to the client: *129.27.142.24*

DNS Caching

Problem: All these queries take a long time!

- Contacting root, then TLD, zone, lower-level zone name servers, …
- Always querying root servers would impose extreme load on them!
- *Latency* happens even before any communication with target webserver

Solution: ***Record Caching***

- Top-level servers change very rarely, popular sites visited often

→ *DNS resolvers cache DNS records for many users*

How long?

- Authoritative service tells you in TTL entry *(seconds, minutes, hours, ...)*
- Resolver deletes record from cache after TTL expires

Multitasking
Pipelining
Speculation
Caches
Multiple Core Systems
Privilege Levels
MMU
TEE
...