



VYSOKÉ UČENÍ FAKULTA
TECHNICKÉ INFORMAČNÍCH
V BRNĚ TECHNOLOGIÍ



2

Sítové aplikace

IPK/2020L

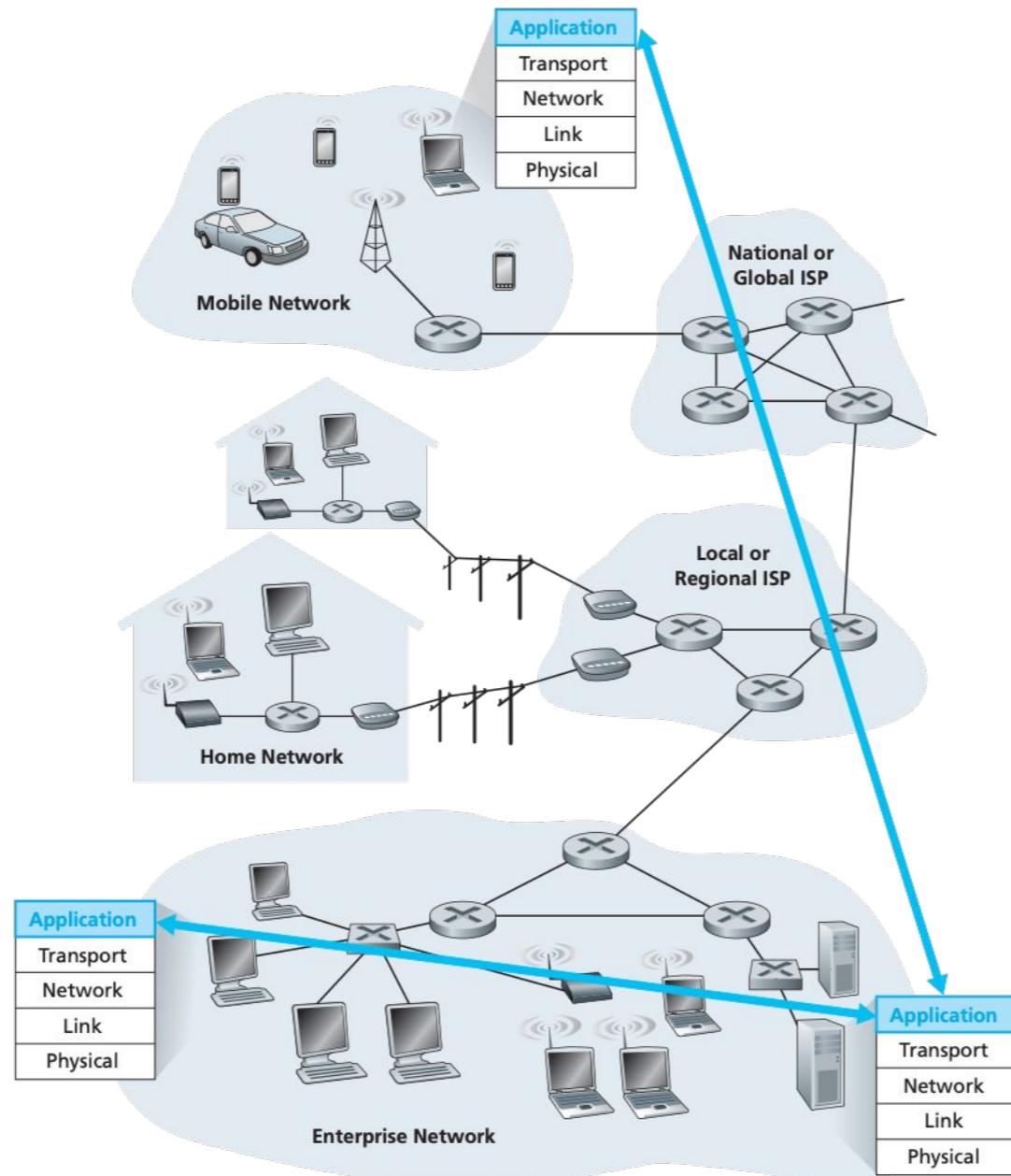
Obsah



SMTPTM

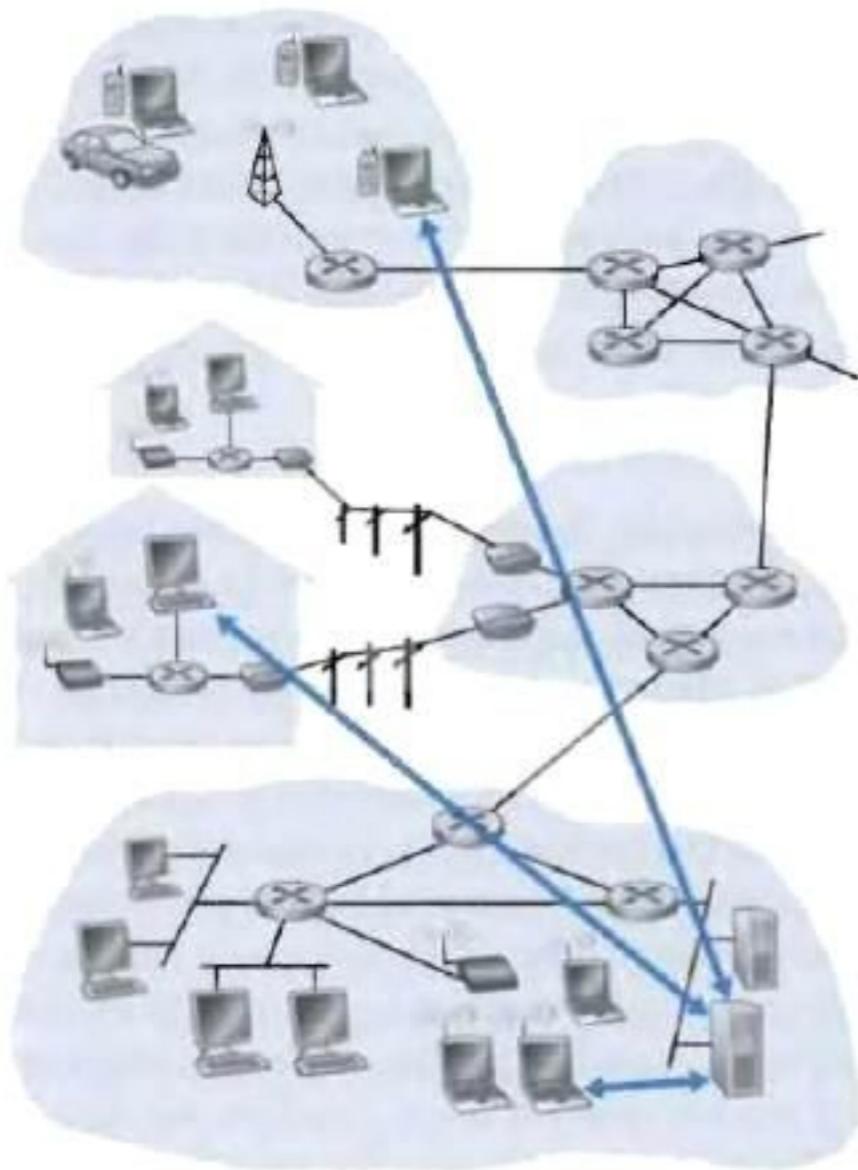


Sítové aplikace

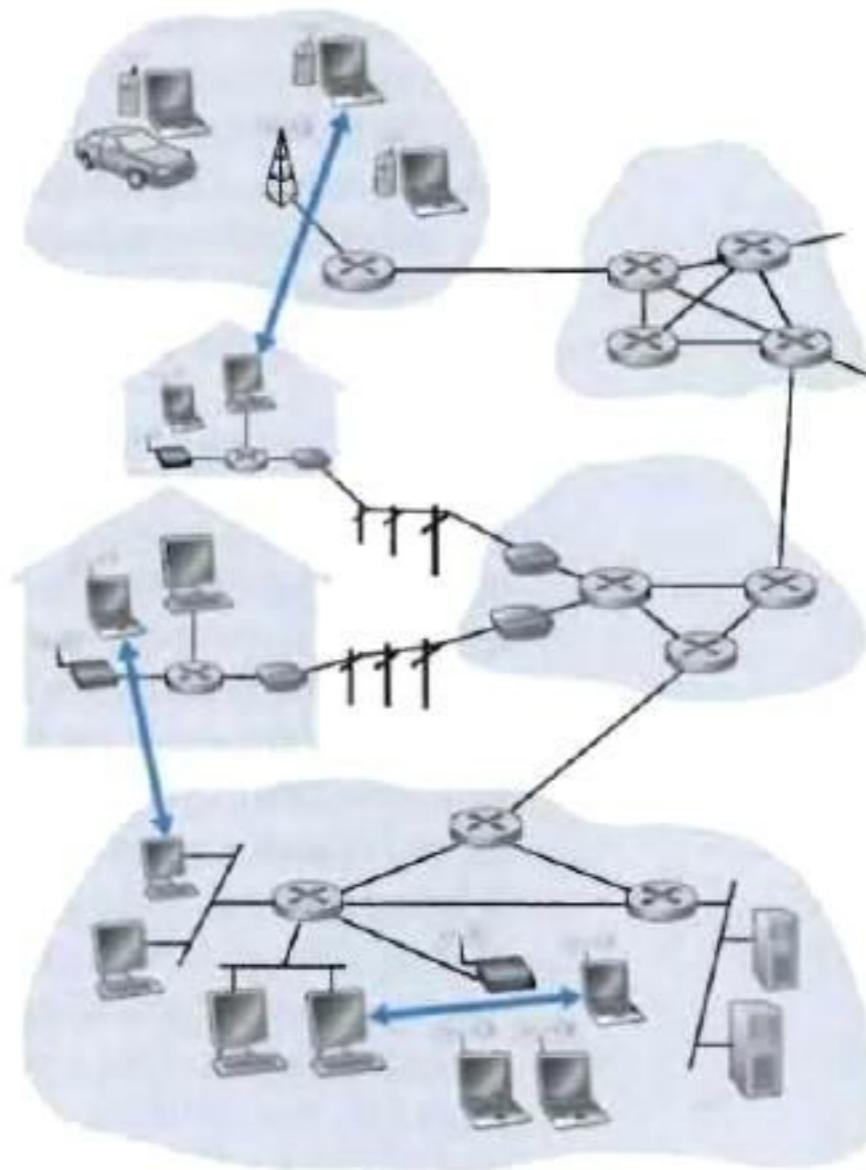


Klient-server /

Peer-to-peer



a. Client-server architecture



b. Peer-to-peer architecture

Požadavky na přenos dat

Application	Data Loss	Throughput	Time-Sensitive
File transfer/download	No loss	Elastic	No
E-mail	No loss	Elastic	No
Web documents	No loss	Elastic (few kbps)	No
Internet telephony/ Video conferencing	Loss-tolerant	Audio: few kbps–1Mbps Video: 10 kbps–5 Mbps	Yes: 100s of msec
Streaming stored audio/video	Loss-tolerant	Same as above	Yes: few seconds
Interactive games	Loss-tolerant	Few kbps–10 kbps	Yes: 100s of msec
Instant messaging	No loss	Elastic	Yes and no

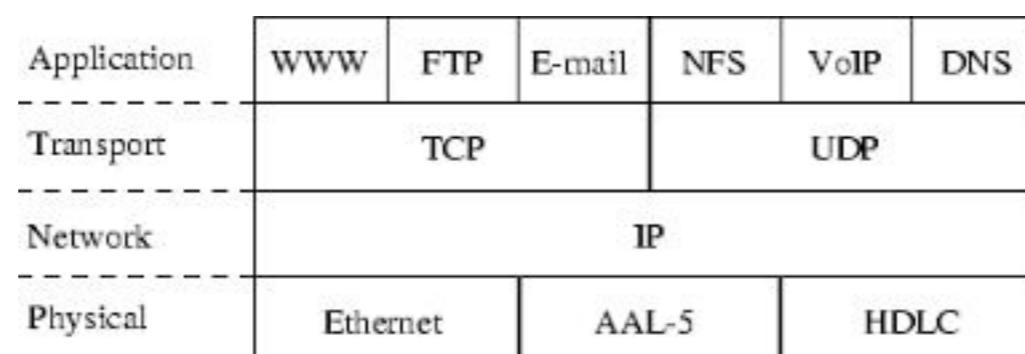
Figure 2.4 ♦ Requirements of selected network applications

Transportní protokoly

TCP	UDP
Keeps track of lost packets. Makes sure that lost packets are re-sent	Doesn't keep track of lost packets
Adds sequence numbers to packets and reorders any packets that arrive in the wrong order	Doesn't care about packet arrival order
Slower, because of all added additional functionality	Faster, because it lacks any extra features
Requires more computer resources, because the OS needs to keep track of ongoing communication sessions and manage them on a much deeper level	Requires less computer resources
Examples of programs and services that use TCP: <ul style="list-style-type: none"> - HTTP - HTTPS - FTP - Many computer games 	Examples of programs and services that use UDP: <ul style="list-style-type: none"> - DNS - IP telephony - DHCP - Many computer games

TCP Segment Header Format										
Bit #	0	7	8	15	16	23	24	31		
0	Source Port				Destination Port					
32	Sequence Number									
64	Acknowledgment Number									
96	Data Offset	Res	Flags		Window Size					
128	Header and Data Checksum				Urgent Pointer					
160...					Options					

UDP Datagram Header Format								
Bit #	0	7	8	15	16	23	24	31
0	Source Port				Destination Port			
32	Length				Header and Data Checksum			

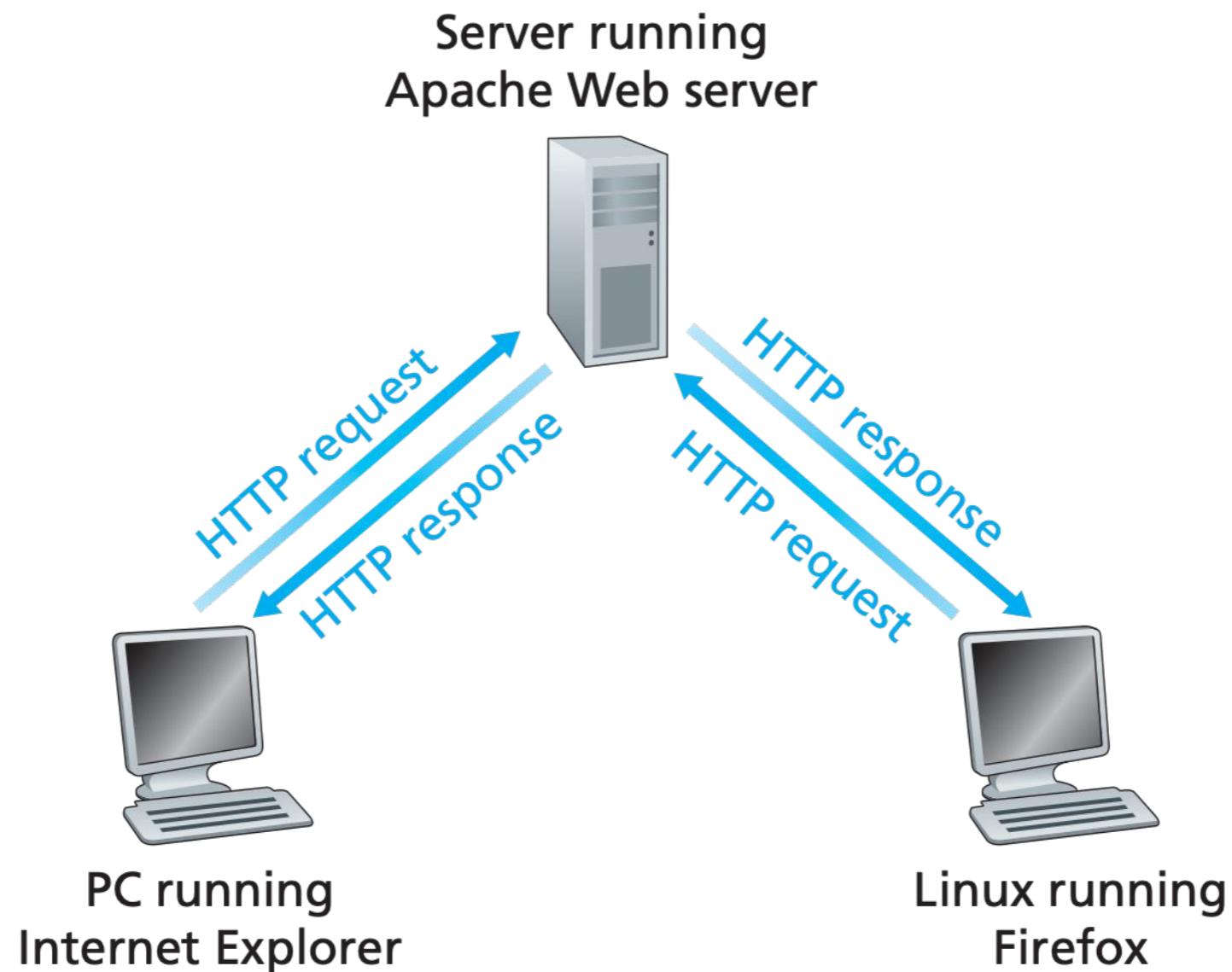


WWW / HTTP(S)

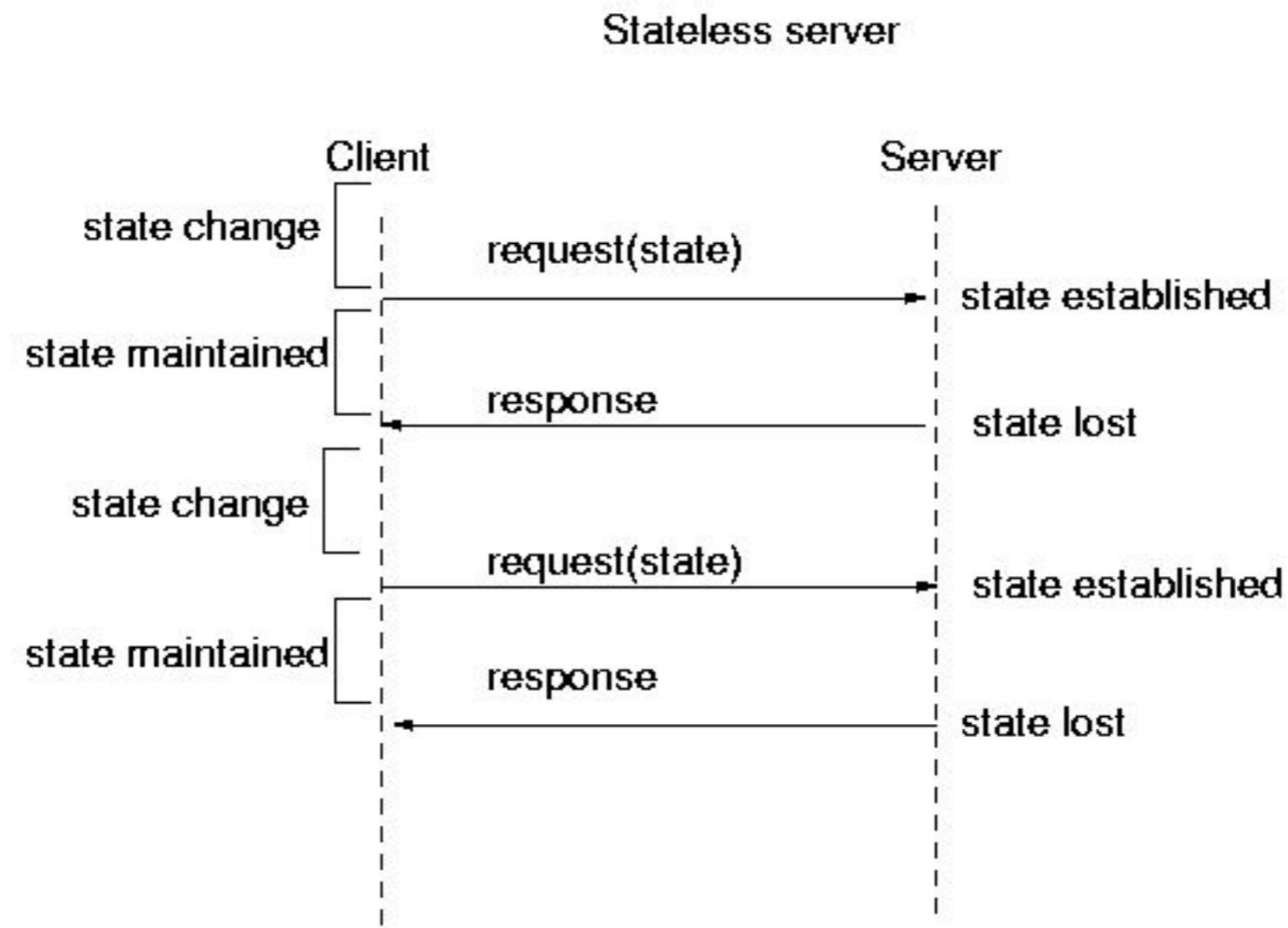
Web

- Web stránka obsahuje objekty
- Objekt může být HTML soubor, JPEG obrázek, Java applet, audio soubor,...
- Web stránka se skládá z bázového HTML souboru, který obsahuje odkazy na další objekty
- Každý objekt je adresovatelný pomocí URL [RFC 1738], obecně podle URI (specifikuje přístup k objektu ~ protokol, RFC 3986)

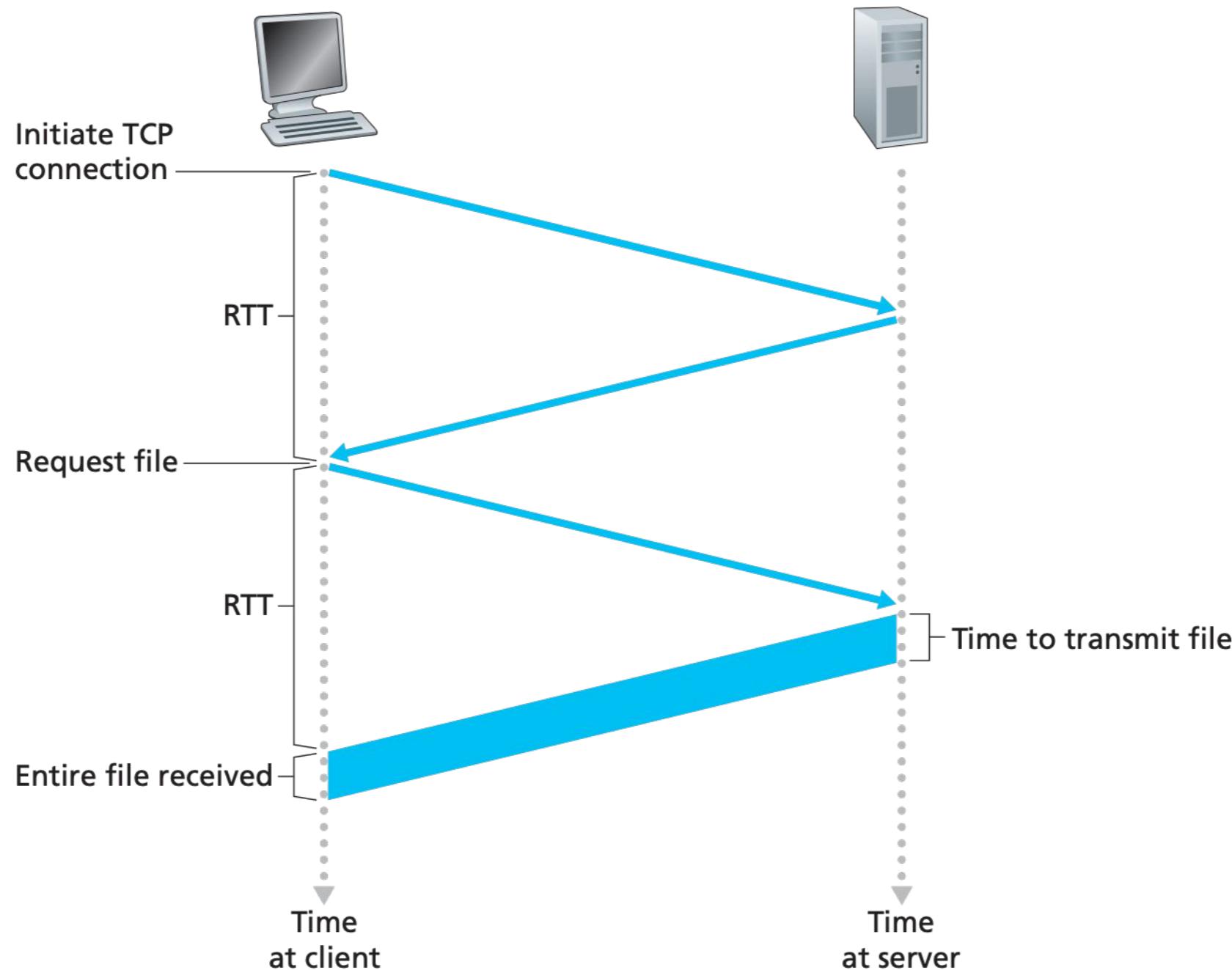
HTTP



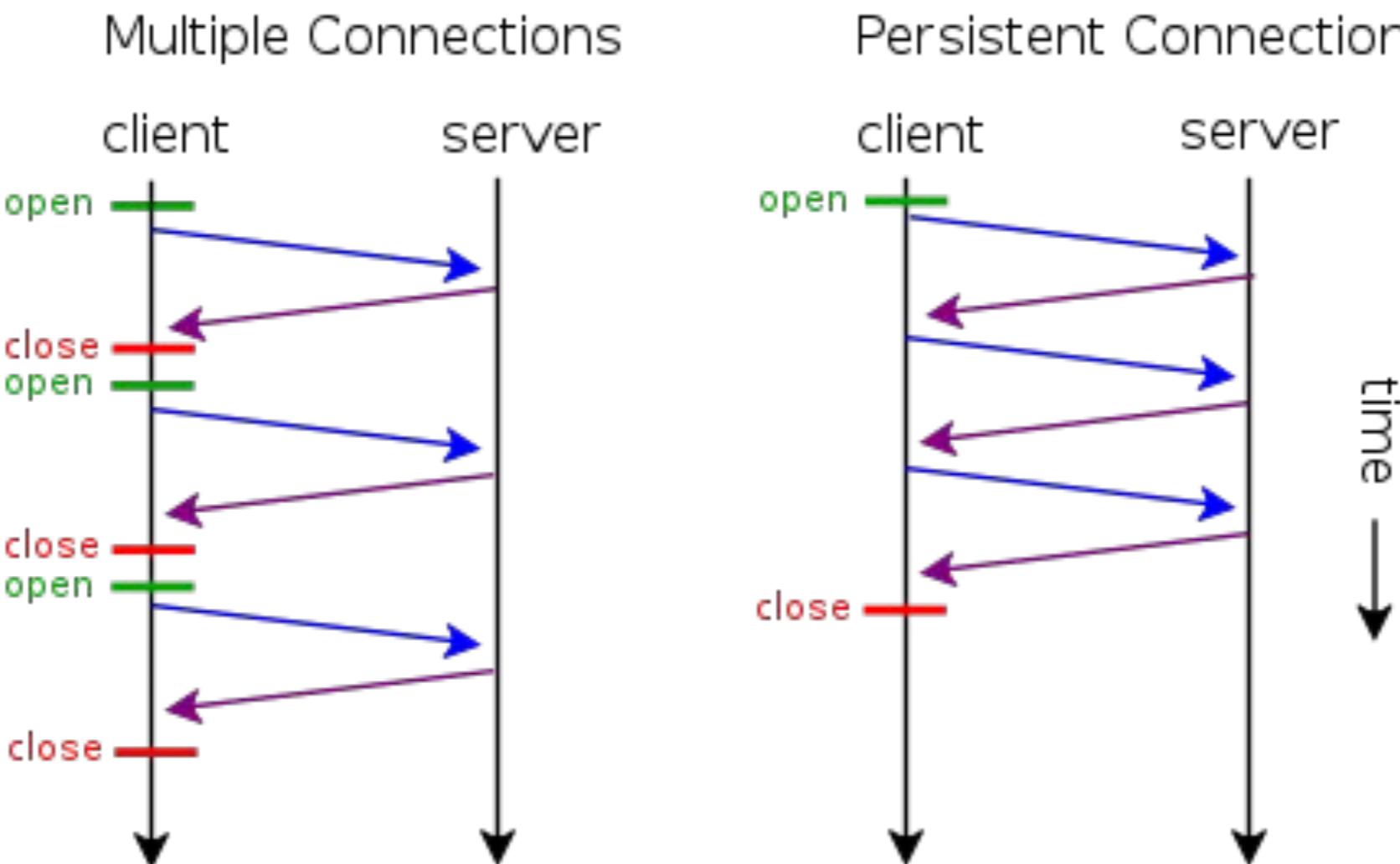
HTTP server



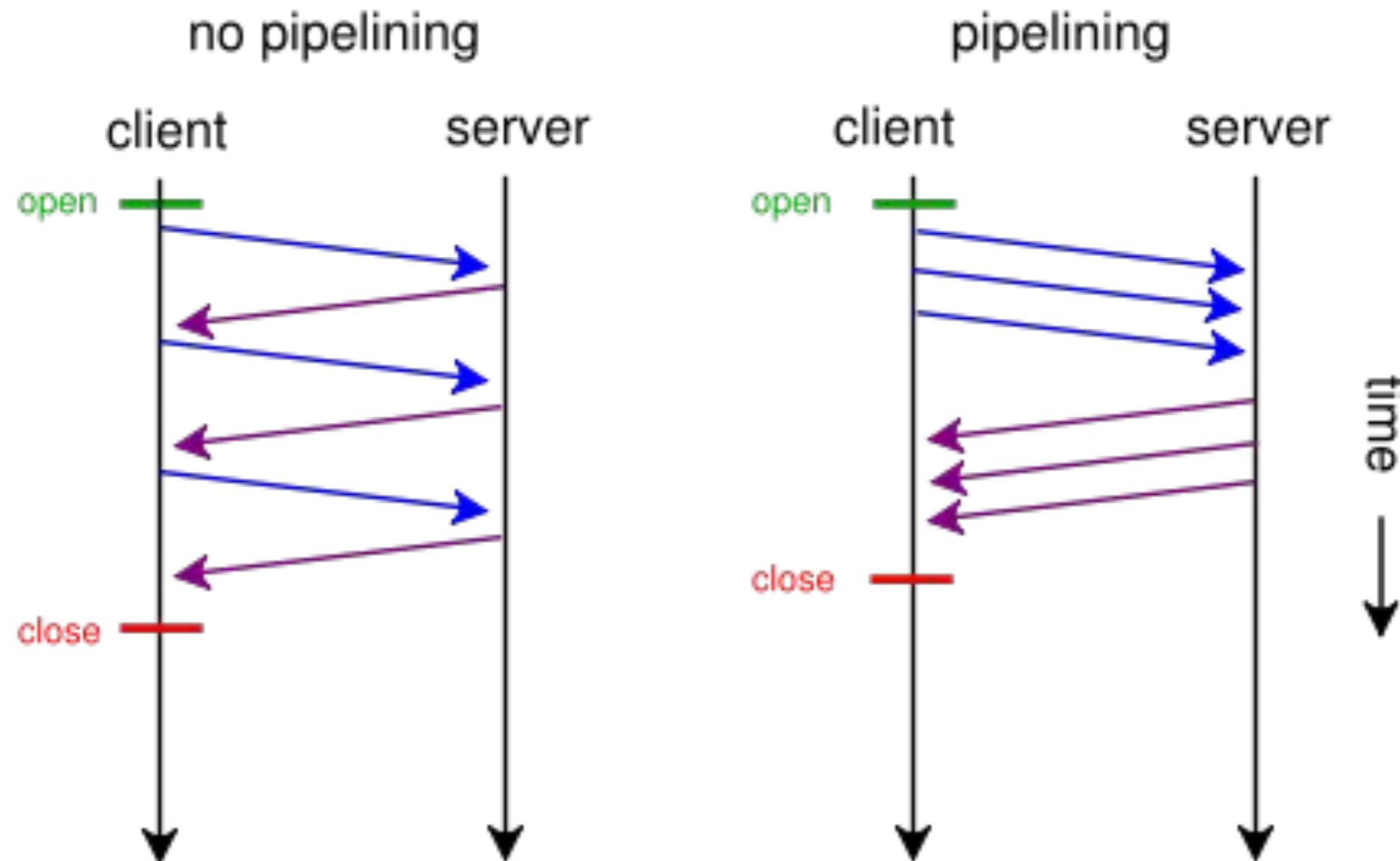
HTTP Spojení



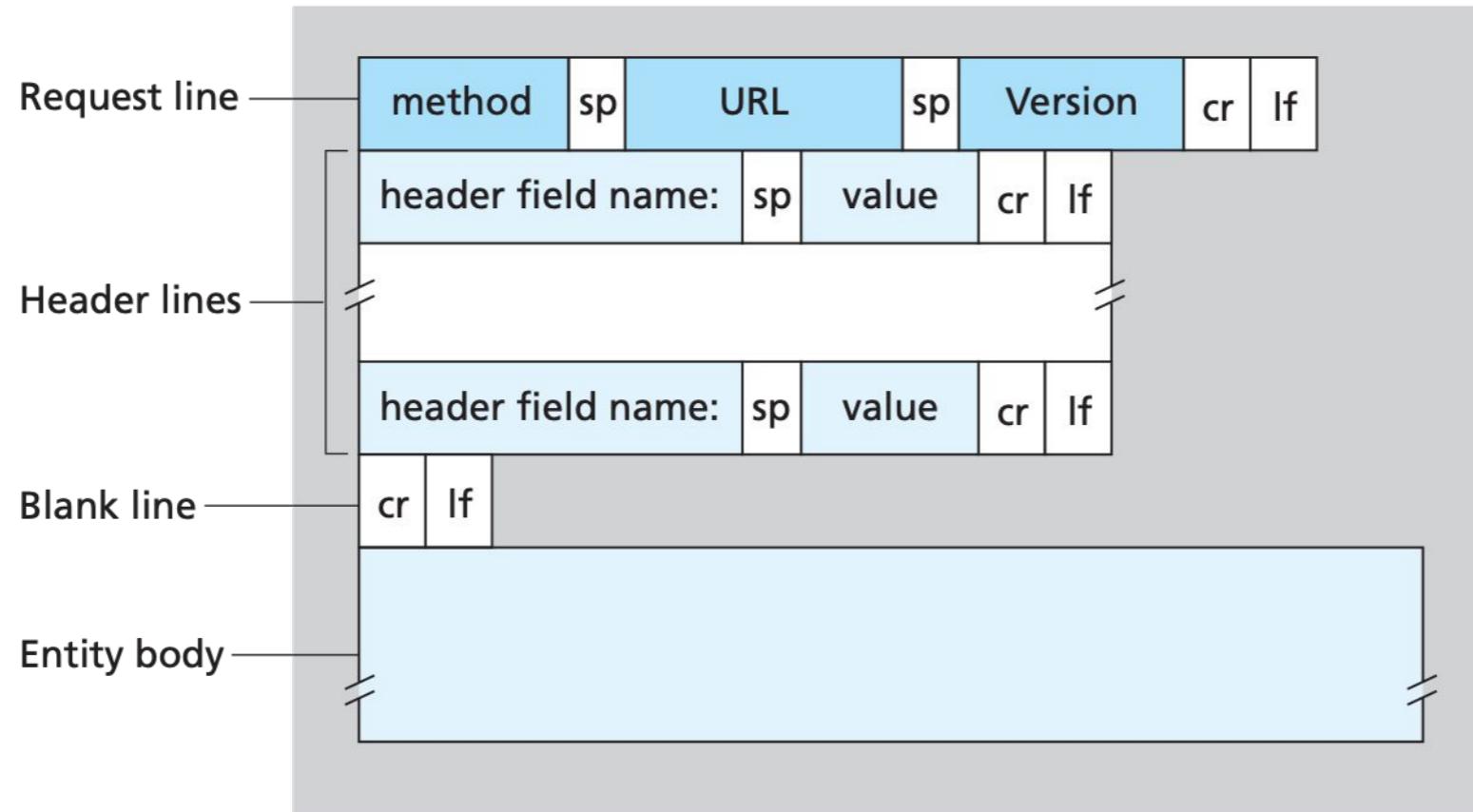
Persistentní spojení



Pipelining

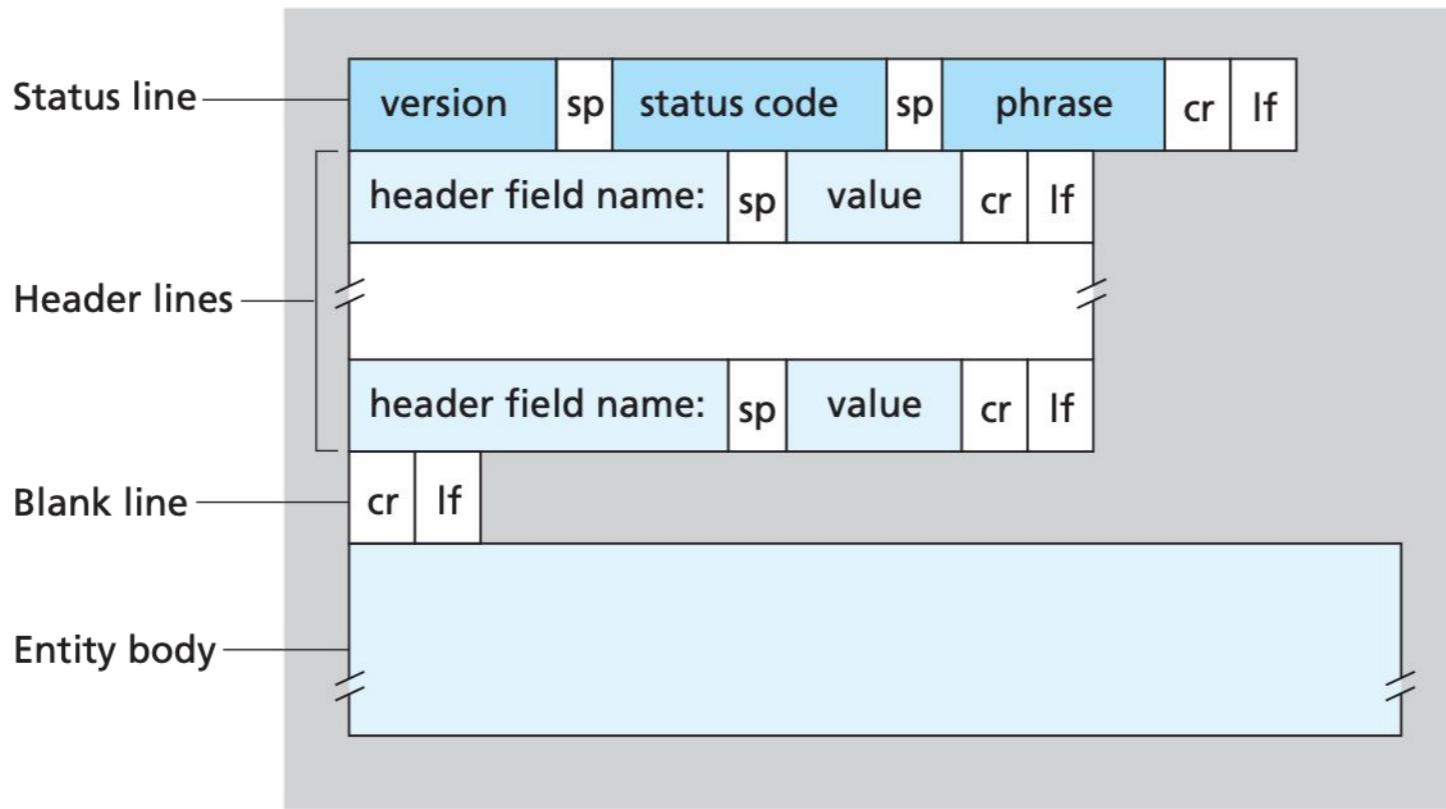


Http Request



```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
Connection: close
User-agent: Mozilla/5.0
Accept-language: fr
```

Http Response



```
HTTP/1.1 200 OK
Connection: close
Date: Tue, 09 Aug 2011 15:44:04 GMT
Server: Apache/2.2.3 (CentOS)
Last-Modified: Tue, 09 Aug 2011 15:11:03 GMT
Content-Length: 6821
Content-Type: text/html

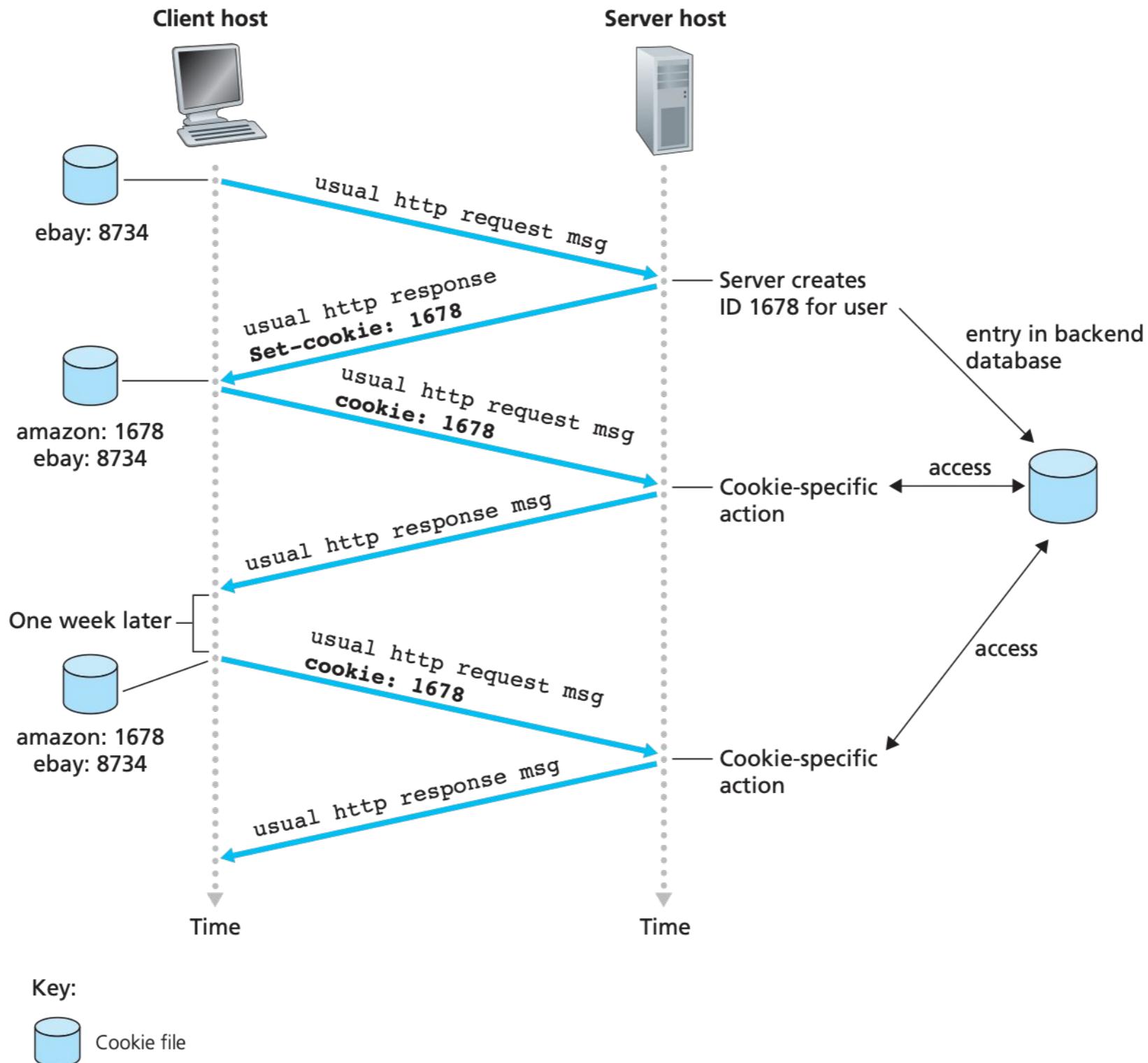
(data data data data data ...)
```

Vyzkoušejte si...

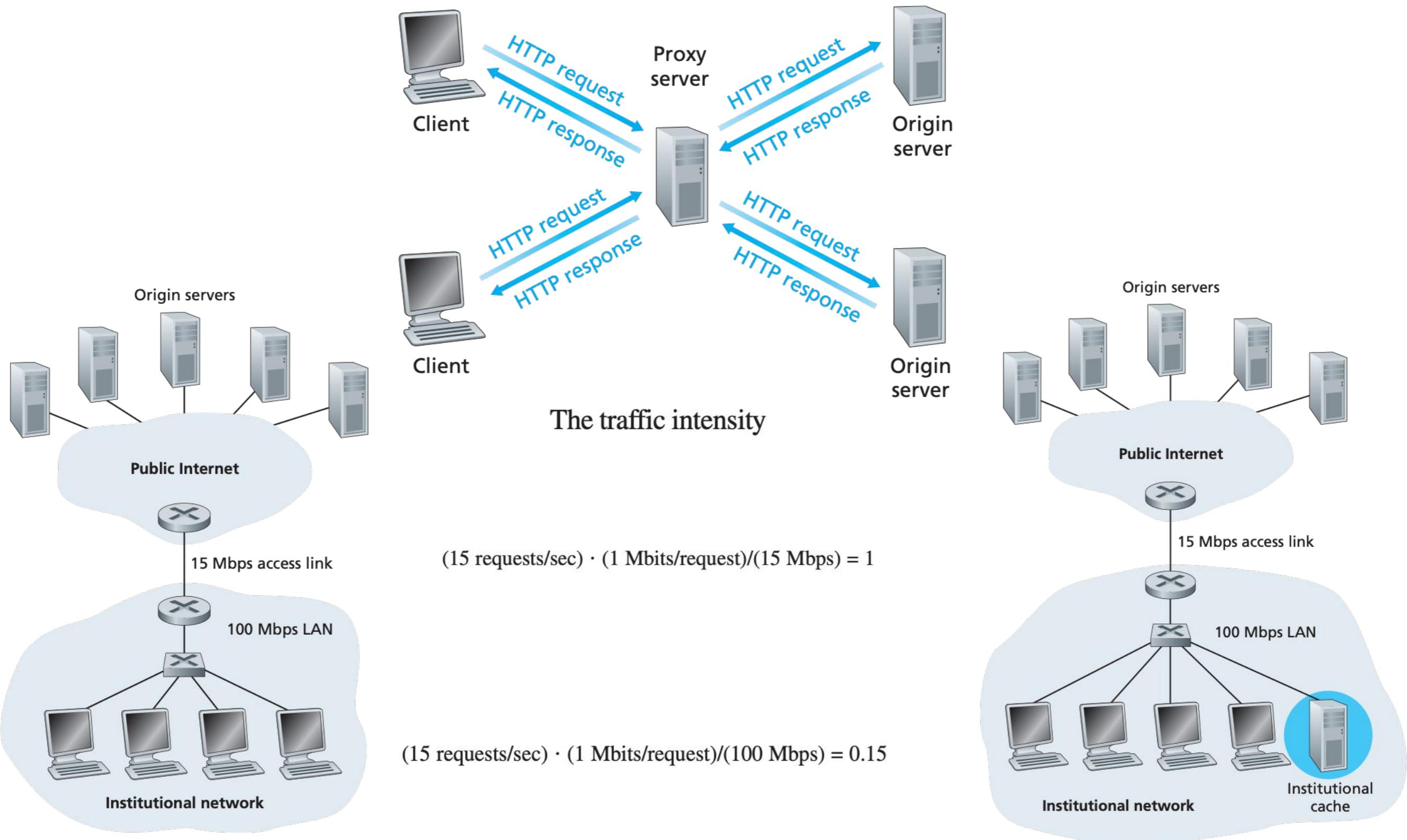
```
kazi:~> telnet www.fit.vutbr.cz 80
Trying 2001:67c:1220:809::93e5:917...
Connected to www.fit.vutbr.cz.
Escape character is '^]'.
GET / HTTP/1.1
Host: www.fit.vutbr.cz

HTTP/1.1 200 OK
Date: Mon, 12 Feb 2018 09:43:56 GMT
Server: Apache
Content-Location: index.php.cz
Vary: negotiate,accept-language
TCN: choice
```

Cookies



Web Cache



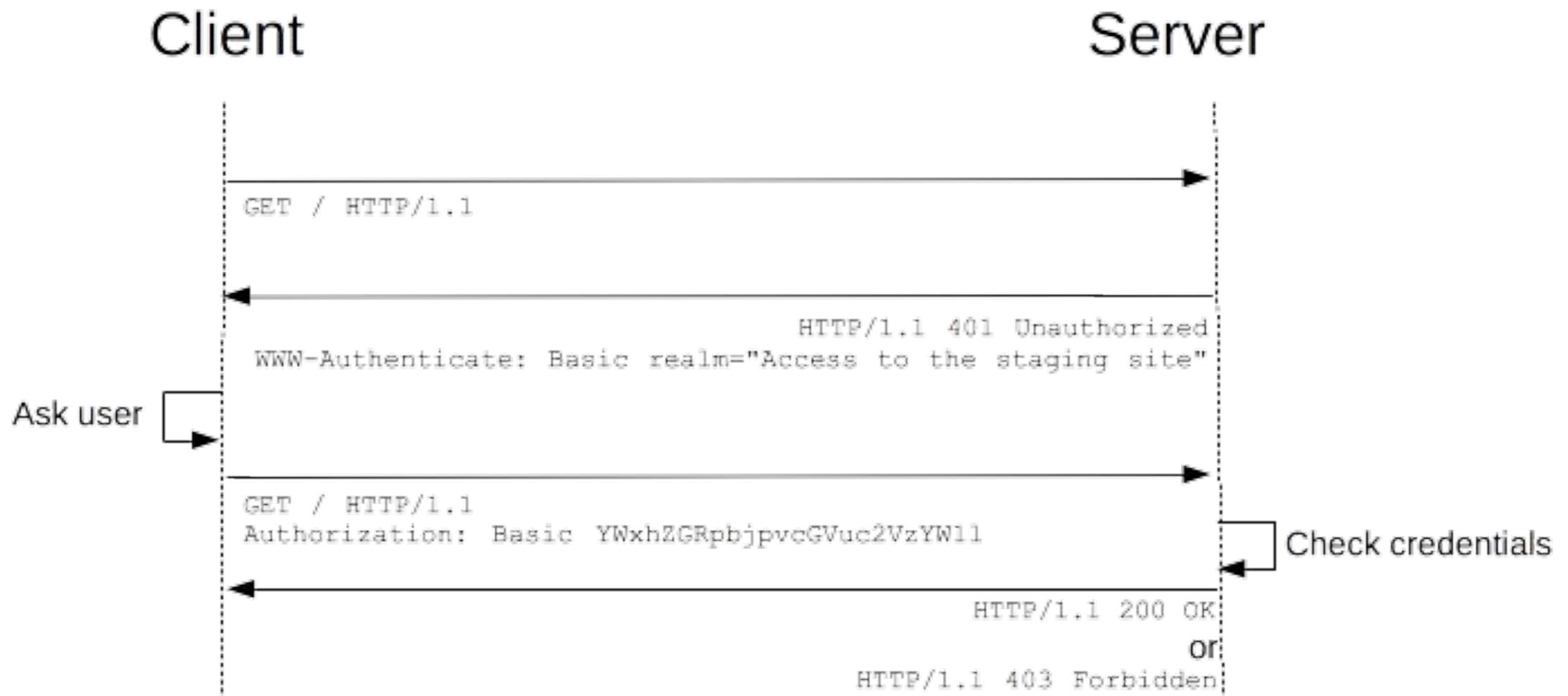
Podmíněný GET

```
GET /fruit/kiwi.gif HTTP/1.1  
Host: www.exotiquecuisine.com
```

```
GET /fruit/kiwi.gif HTTP/1.1  
Host: www.exotiquecuisine.com  
If-modified-since: Wed, 7 Sep 2011
```

```
HTTP/1.1 304 Not Modified  
Date: Sat, 15 Oct 2011 15:39:29  
Server: Apache/1.3.0 (Unix)
```

HTTP authentication



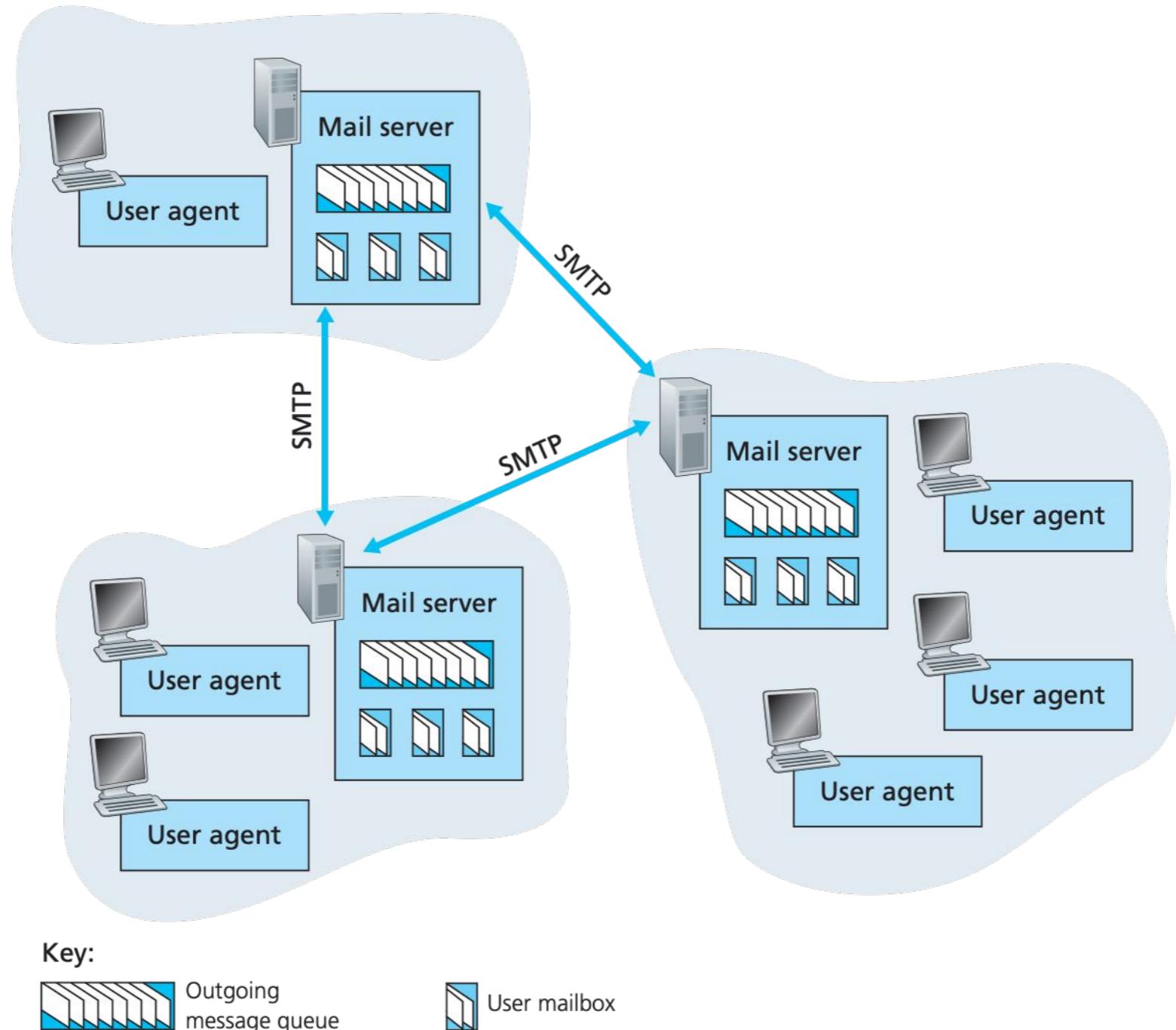


WIRESHARK DEMO

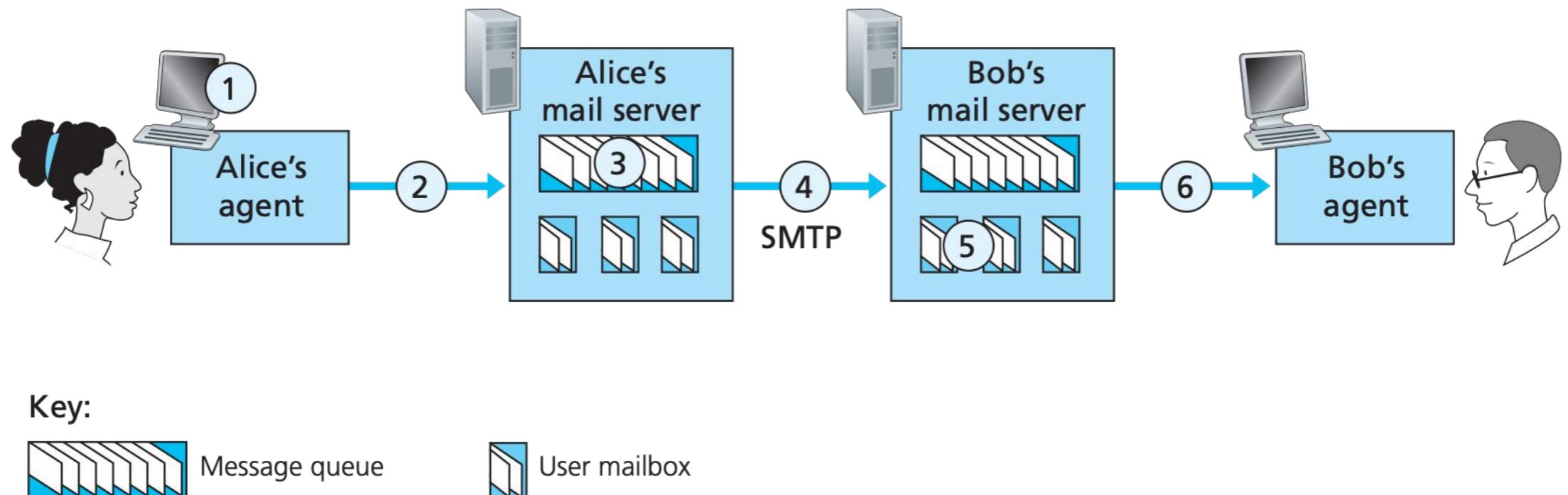
- http.cap
- http-cookies.pcap
- http-authorization.pcap

SMTP

Architektura



Přenos zprávy



Protokol

```
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
```

Formát zpráv (RFC 2822)

Hlavíčka	<pre>From matousp@fit.vutbr.cz Wed Oct 21 22:25:13 2009 Return-Path: <matousp@fit.vutbr.cz> Received: from [192.168.15.101] (ip4-83-240-62-147.cust.nbox.cz [83.240.62.147]) (user=matousp mech=PLAIN bits=0) by kazi.fit.vutbr.cz for <rysavy@fit.vutbr.cz>; Wed, 21 Oct 2009 22:25:12 +0200 (CEST) Message-ID: <4ADF6E2B.3050709@fit.vutbr.cz> Date: Wed, 21 Oct 2009 22:25:15 +0200 From: Petr Matousek <matousp@fit.vutbr.cz> User-Agent: Thunderbird 2.0.0.23 (Windows/20090812) MIME-Version: 1.0 To: Ondrej Rysavy <rysavy@fit.vutbr.cz> Subject: [Fwd: [Ideal-ist] New EOI Confirmation for PS-ES-3816] Content-Type: text/plain; charset=ISO-8859-2 Content-Transfer-Encoding: quoted-printable</pre>
Tělo	<p>Ondro,</p> <p>to jsi bajecne napsal! Diky.</p> <p>Cau.</p> <p>Petr</p>

MIME (RFC 2045, 2046)

From: Petr Matousek <matousp@fit.vutbr.cz>

MIME-Version: 1.0

To: Ondrej Rysavy <rysavy@fit.vutbr.cz>

Subject: Dokument ve Wordu

Content-Type: application/msword; name="dokument.doc"

Content-Transfer-Encoding: base64

0M8R4KGxGuAAAAAAAAAAAAAPgADAP7/CQAGAAAAAAA

AAA

AAAAEAAASgEAAAEEAAAD+///AAAAAEQBAABFAQAARgEAAP//////

//////////

//////////

//////////

//////////

//////////

//////////

//////////

ASgEAAAEE//

//////////

//////////

From: Petr Matousek <matousp@fit.vutbr.cz>

User-Agent: Thunderbird 2.0.0.23 (Windows/20090812)

MIME-Version: 1.0

To: Ondrej Rysavy <rysavy@fit.vutbr.cz>

Subject: [Fwd: [Ideal-ist] New EOI Confirmation for PS-ES-3816]

This is a multi-part message in MIME format.

-----070303080205000303010908

Content-Type: text/plain; charset=windows-1252; format=flowed

Content-Transfer-Encoding: 8bit

Ondro,

to jsi baje ne napsal! Diky.

Cau.

Petr

-----070303080205000303010908

Content-Type: application/msword;

name="Work distribution.doc"

Content-Transfer-Encoding: base64

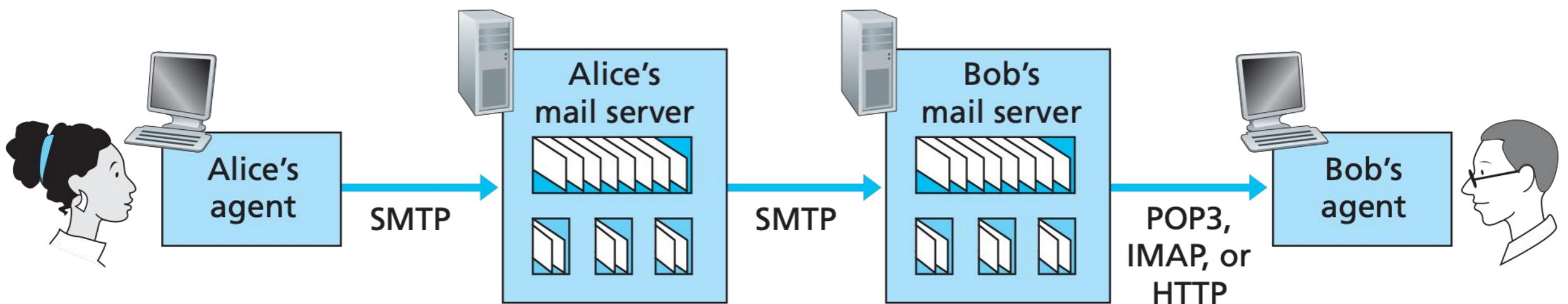
Content-Disposition: inline;

filename="Work distribution.doc"

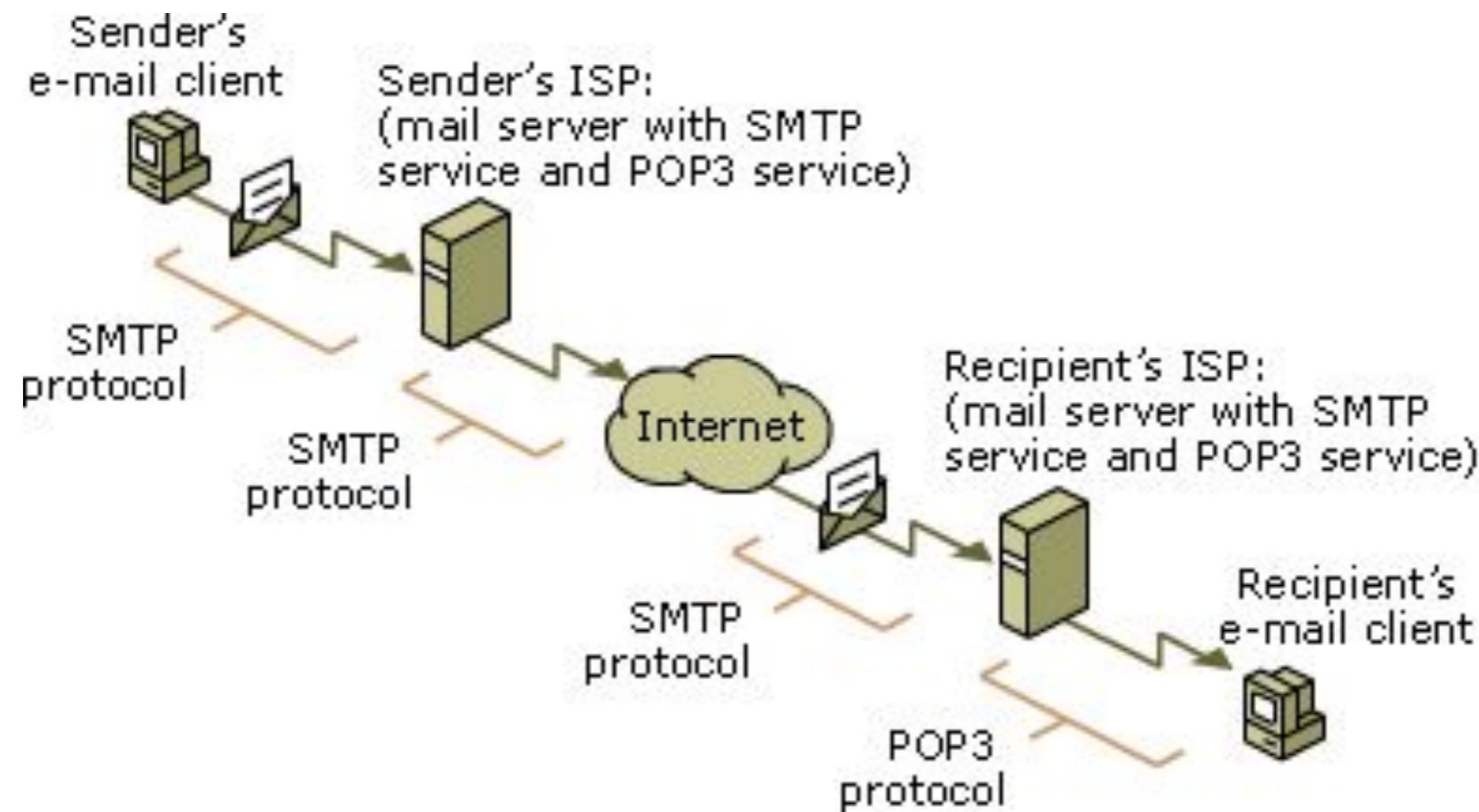
0M8R4KGxGuAAAAAAAAAAAAAPgADAP7/CQAGAAAAAAA

AAAAEAAASgEAAAEEAAAD+///

Čtení zprávy



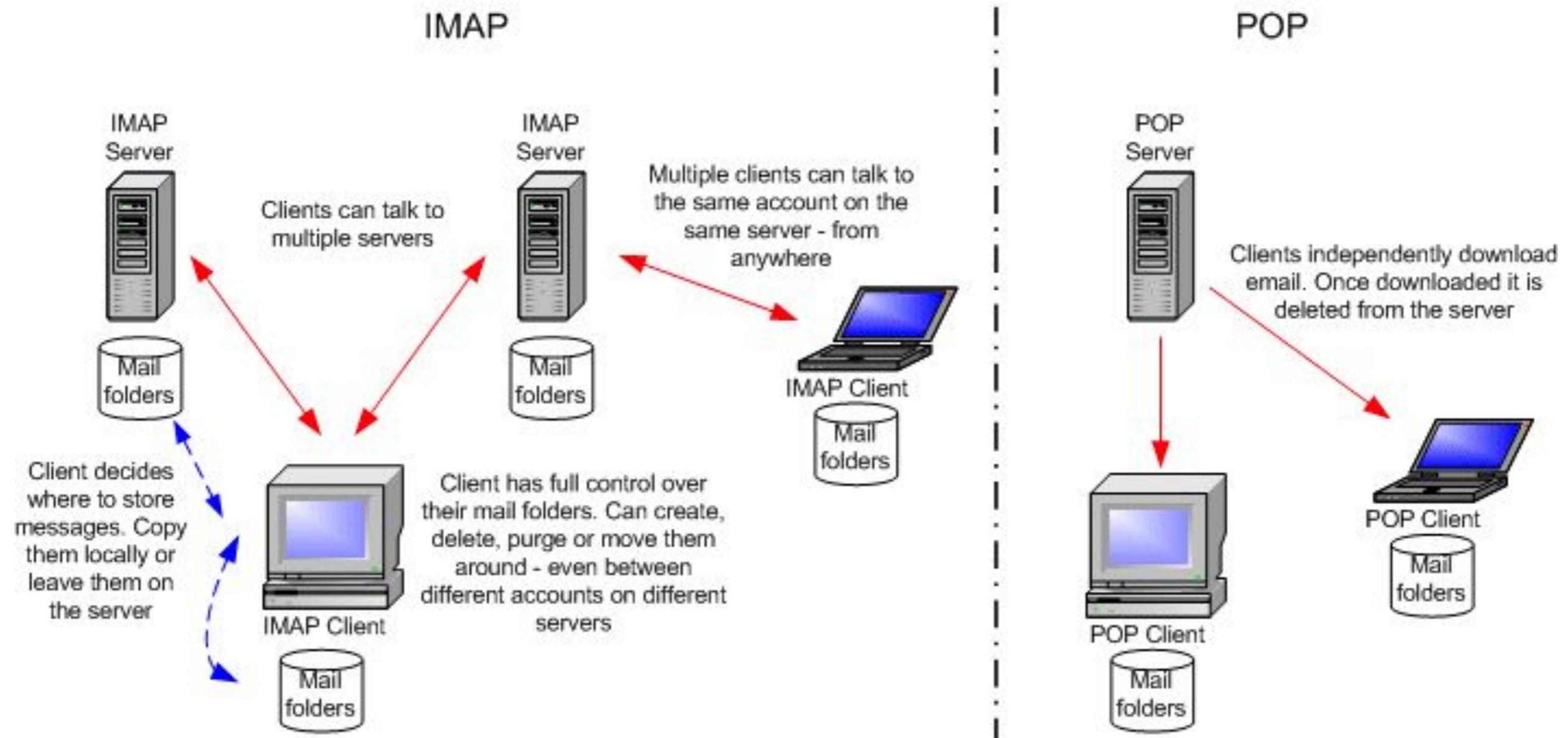
POP3



POP3

```
C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: (blah blah ...
S: .....
S: .....blah)
S: .
C: dele 1
C: retr 2
S: (blah blah ...
S: .....
S: .....blah)
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```

POP / IMAP



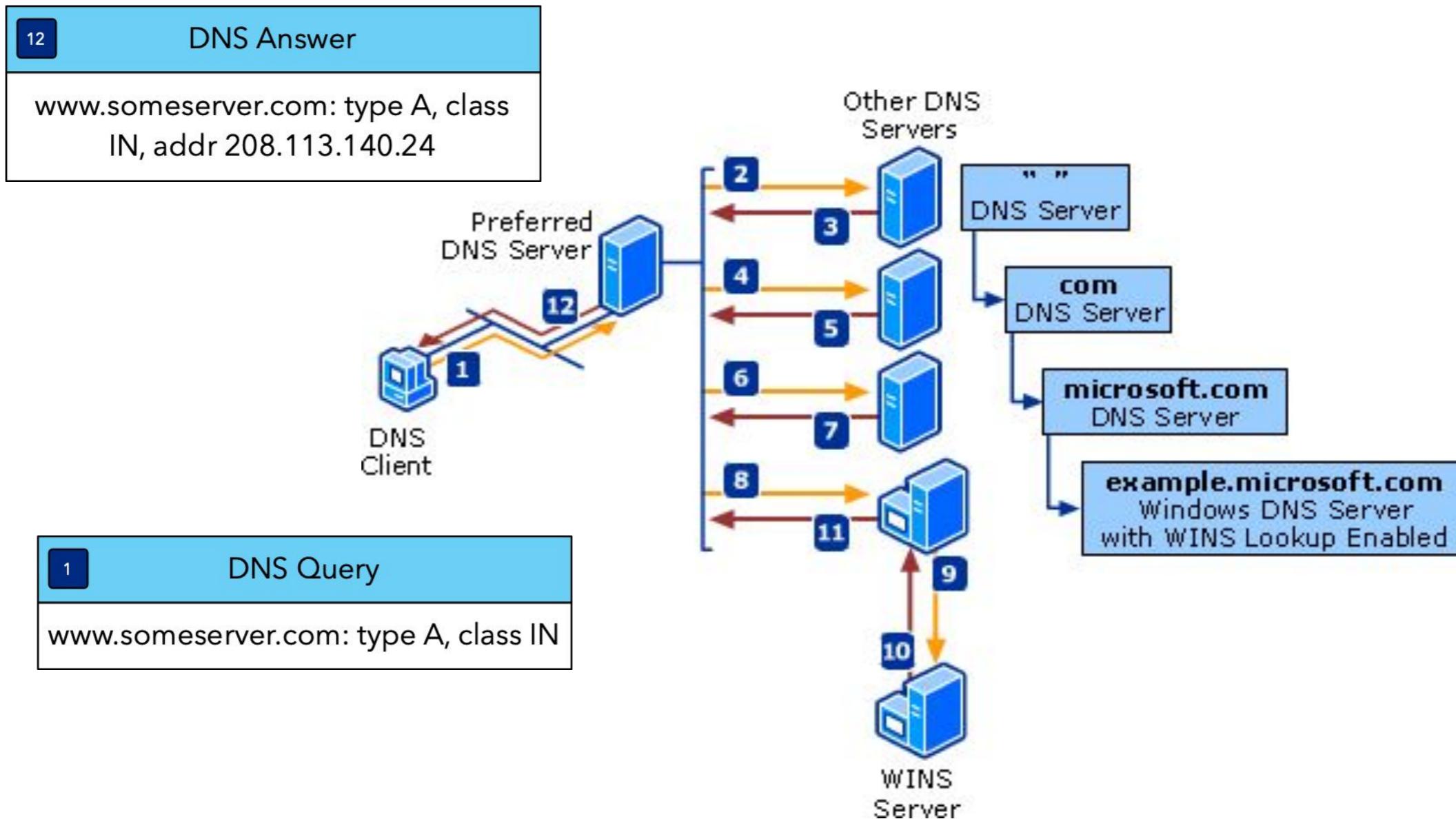


WIRESHARK DEMO

- smtp.pcap

DNS

DNS komunikace

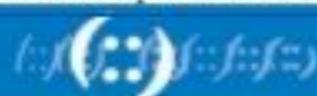


DNS záznam

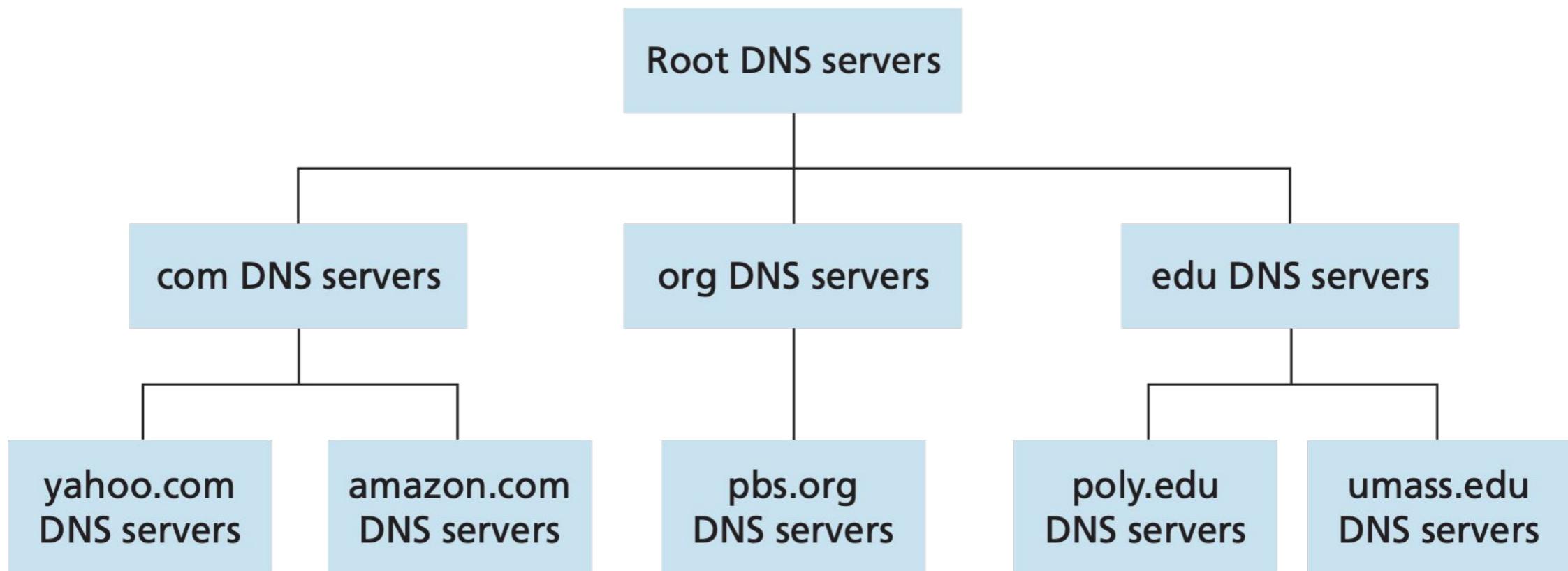
Common Resource Record Types

RR Type	Name	Functions
A	Address record	Maps domain name to IP address www.apnic.net. IN A 203.176.189.99
AAAA	IPv6 address record	Maps domain name to an IPv6 address www.apnic.net. IN AAAA 2001:db8::1
NS	Name server record	Used for delegating zone to a nameserver apnic.net. IN NS ns1.apnic.net.
PTR	Pointer record	Maps an IP address to a domain name 99.189.176.203.in-addr.arpa. IN PTR www.apnic.net.
CNAME	Canonical name	Maps an alias to a hostname web IN CNAME www.apnic.net.
MX	Mail Exchanger	Defines where to deliver mail for user @ domain apnic.net. IN MX 10 mail01.apnic.net. IN MX 20 mail02.apnic.net.

APNIC



DNS



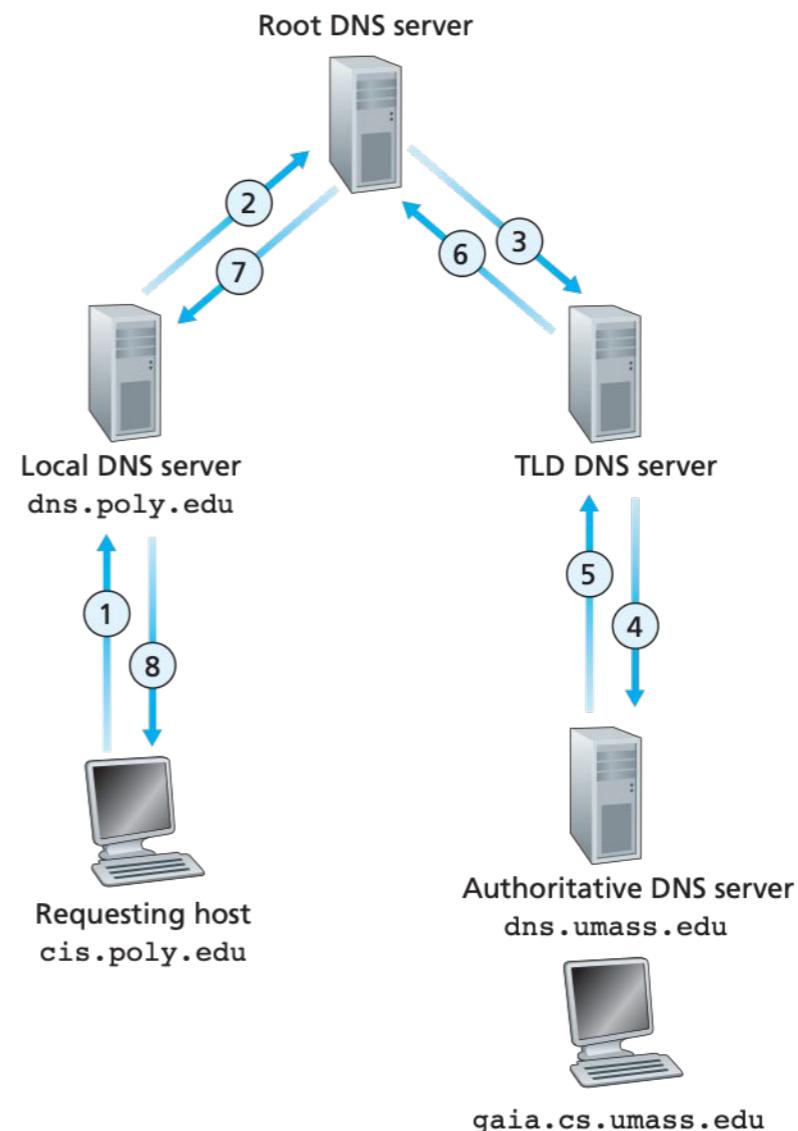
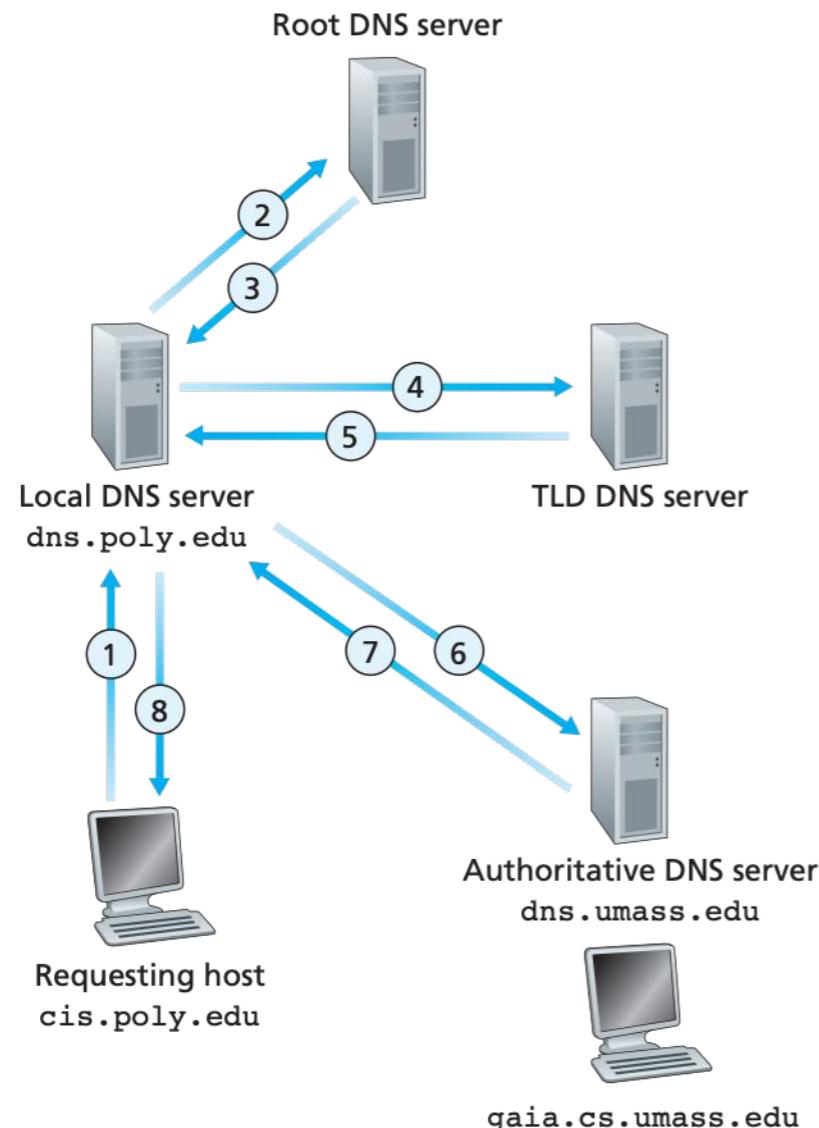
DNS Root Name Server Locations



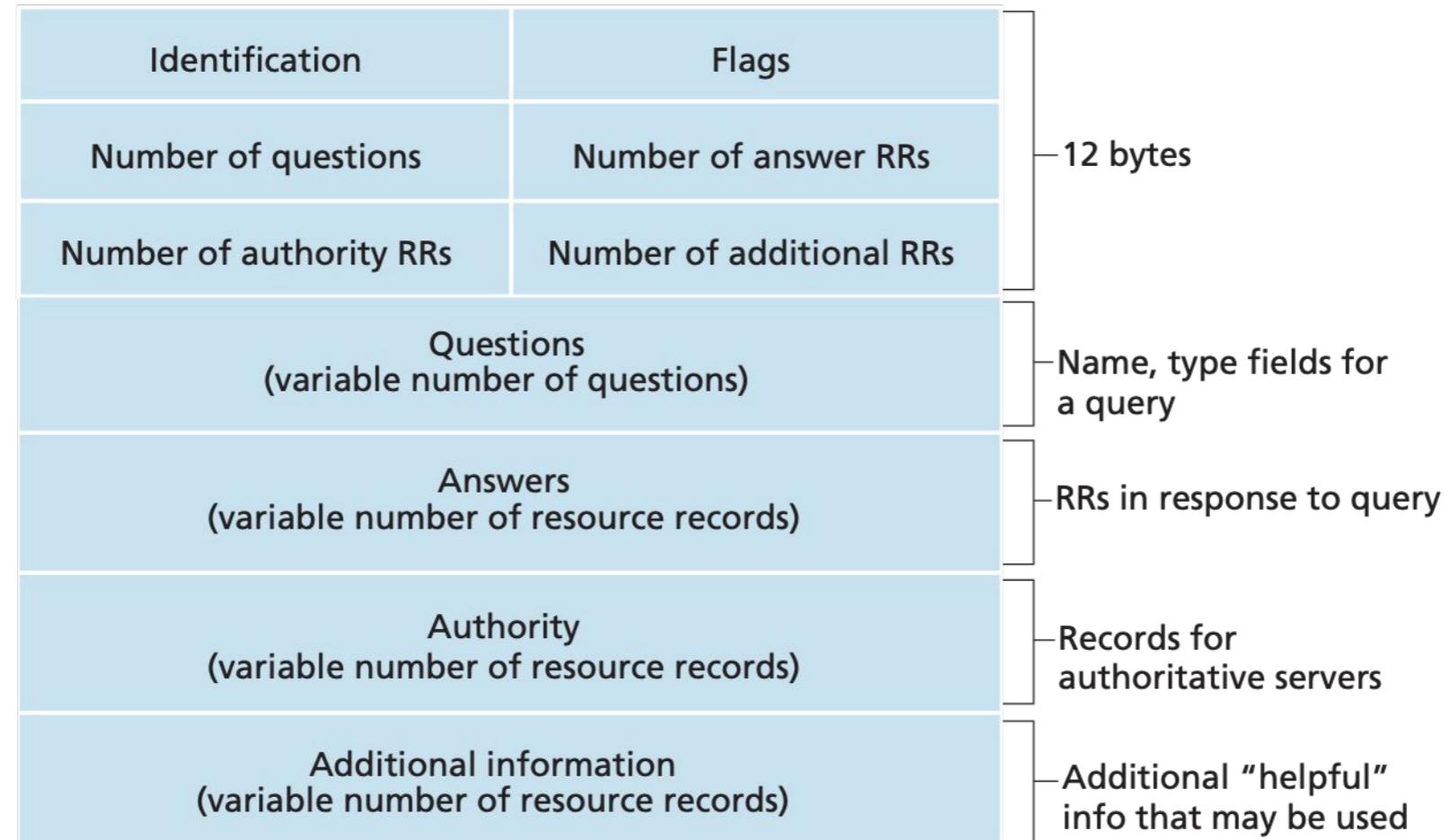
DNS Rezoluce

Takto sr fo hetti do pspj

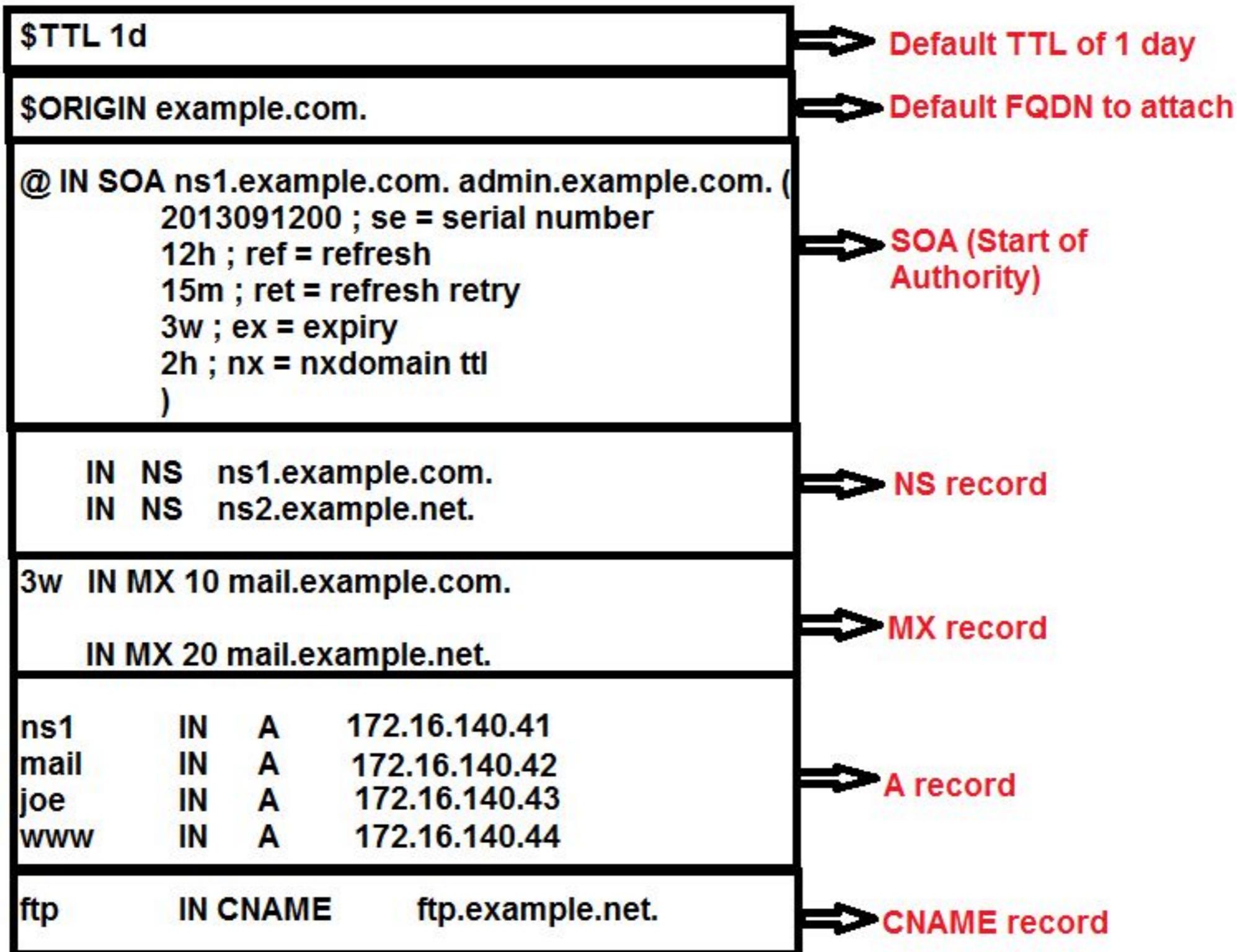
Mo 4 pspj



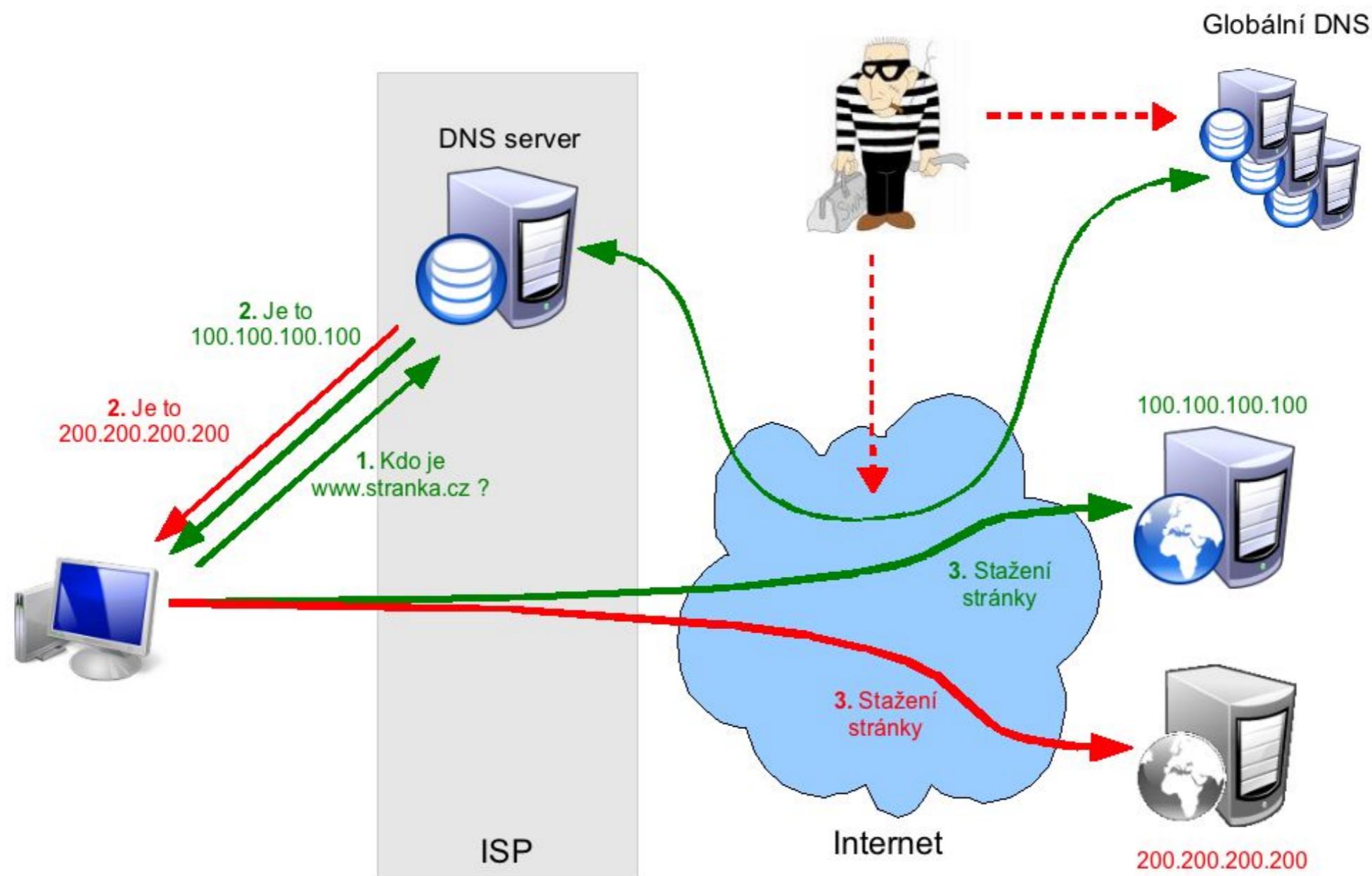
DNS Zpráva



Zónové soubory



DNS bezpečnost



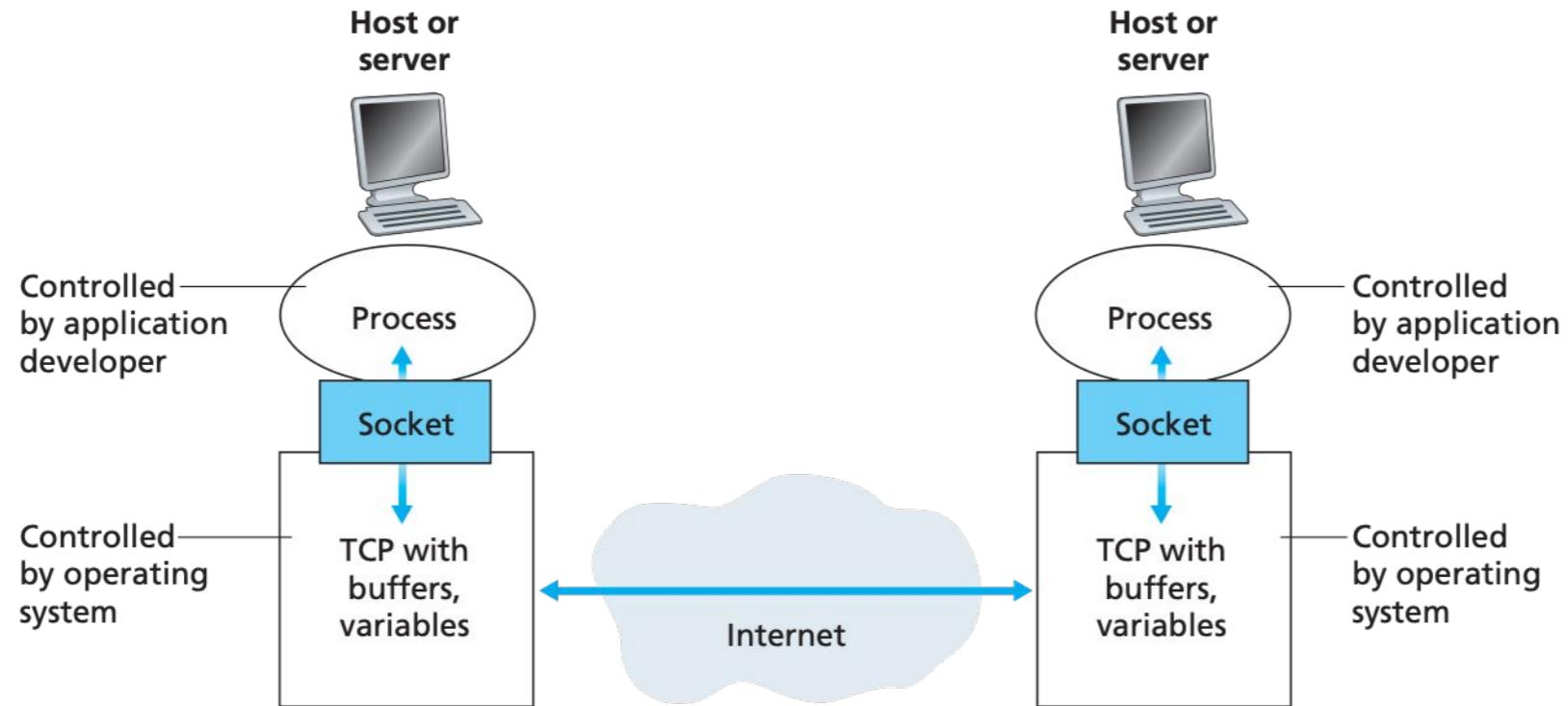


WIRESHARK DEMO

- dns.pcap

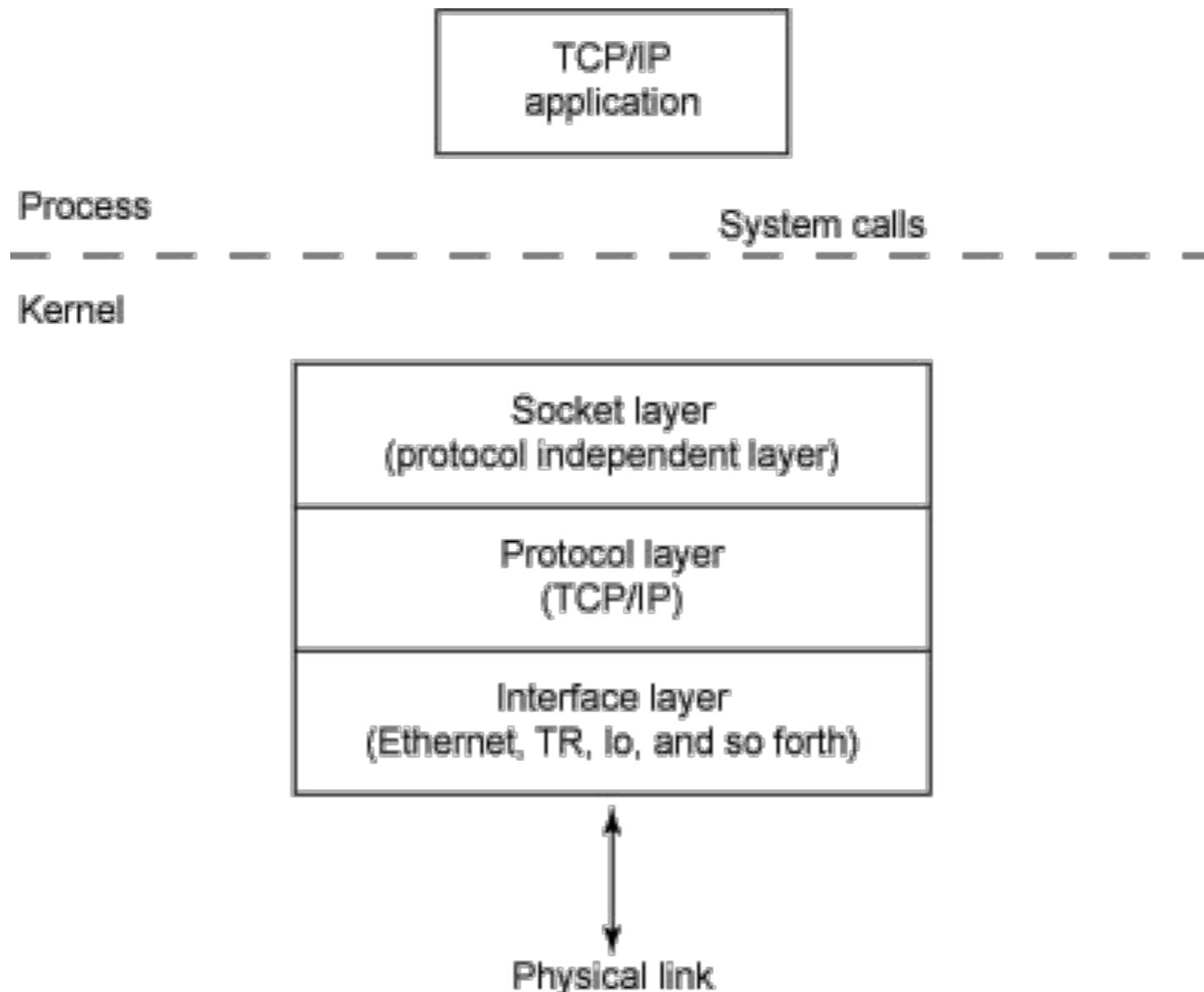
Programování

Schránky



Socket Pair	
Socket	Socket
Address	Address
Port	Port

Socket



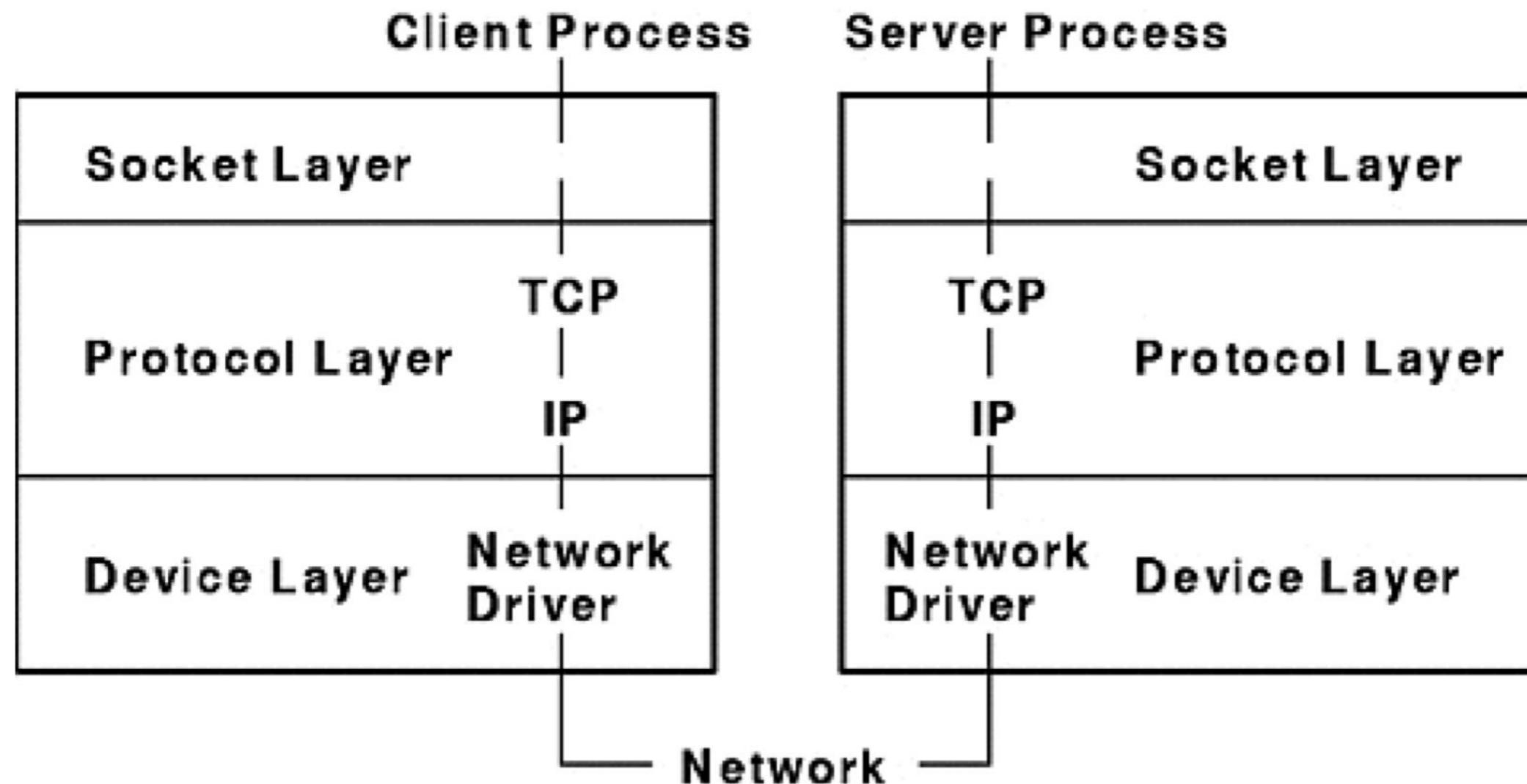
Identifikace spojení



IP adresa : Port	IP adresa : Port
Protokol	

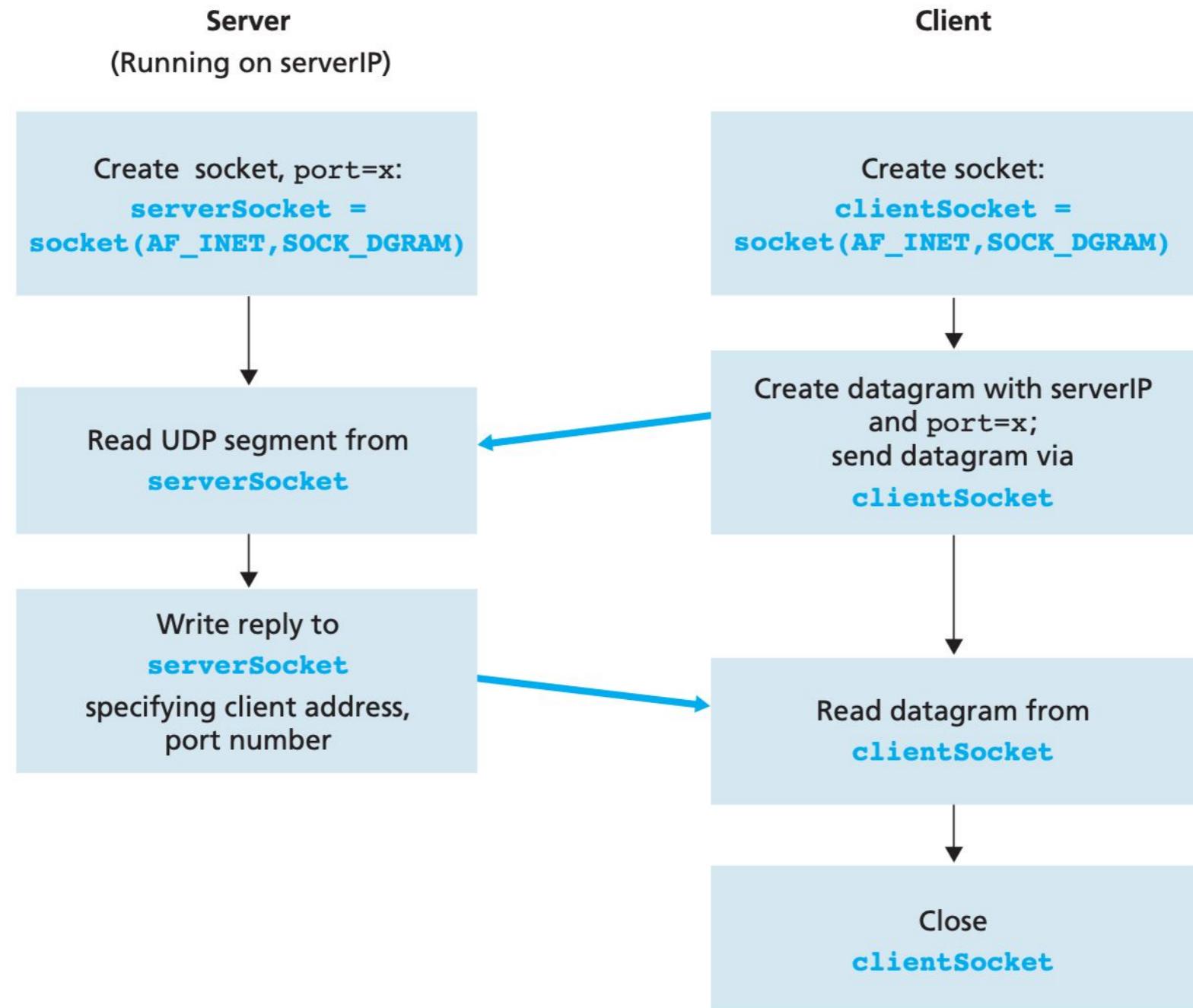
tcp4	0	0	192.168.88.133.57243	192.30.252.126.https	LAST_ACK
tcp4	0	0	192.168.88.133.57242	192.30.252.126.https	LAST_ACK
tcp4	0	0	192.168.88.133.57241	192.30.252.126.https	LAST_ACK
tcp4	0	0	192.168.88.133.57192	msnbot-191-232-1.https	ESTABLISHED
tcp4	0	0	192.168.88.133.57191	msnbot-191-232-1.https	ESTABLISHED
tcp4	0	0	192.168.88.133.57180	live.github.com.https	ESTABLISHED
tcp4	0	0	192.168.88.133.57177	server25004.team.5938	ESTABLISHED
tcp4	0	0	192.168.88.133.57164	17.172.232.12.5223	ESTABLISHED
tcp4	0	0	192.168.88.133.57163	17.143.161.164.5223	ESTABLISHED

Architektura

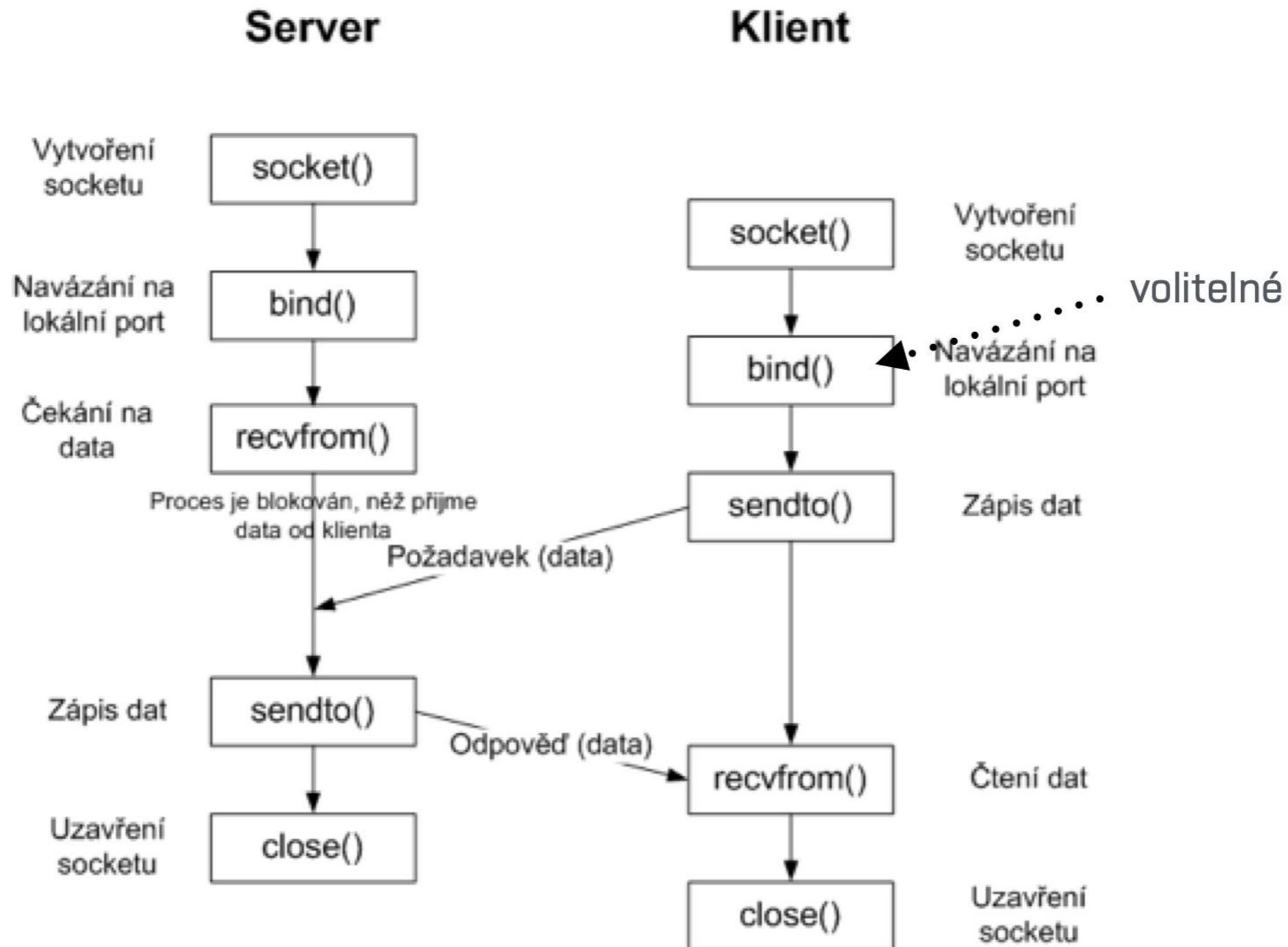


UDP komunikace

UDP Klient-server

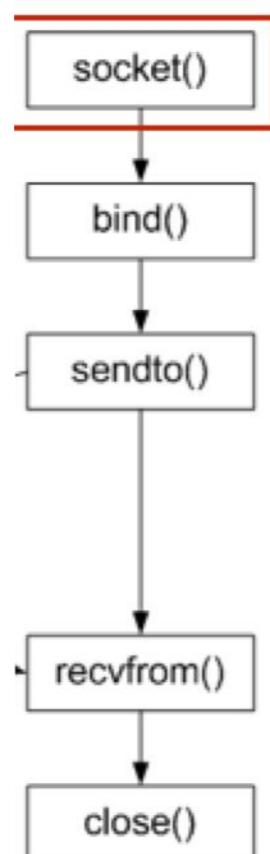


Systémová volání



Vytvoření socketu

Klient



```
if ((client_socket = socket(AF_INET, SOCK_DGRAM, 0)) <= 0)
{
    perror("ERROR in socket");
    exit(EXIT_FAILURE);
}
```

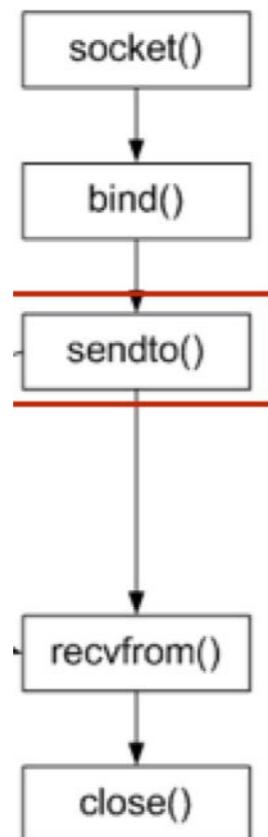
kommunikacia (int.)



Vracia sa prijatá
ak c = 0 - chyba

Odeslání dat

Klient



```
ssize_t  
sendto(int socket, const void *buffer, size_t length,  
int flags,  
const struct sockaddr *dest_addr,  
socklen_t dest_len);
```

```
serverlen = sizeof(server_address);  
bytestx = sendto(client_socket, buf, strlen(buf), 0,  
(struct sockaddr *) &server_address, serverlen);  
if (bytestx < 0)  
    perror("ERROR in sendto");
```

↑ když chceš poštat

Adresa server

```
char *server_hostname; Domeškový název
struct hostent *server;
int port_number;

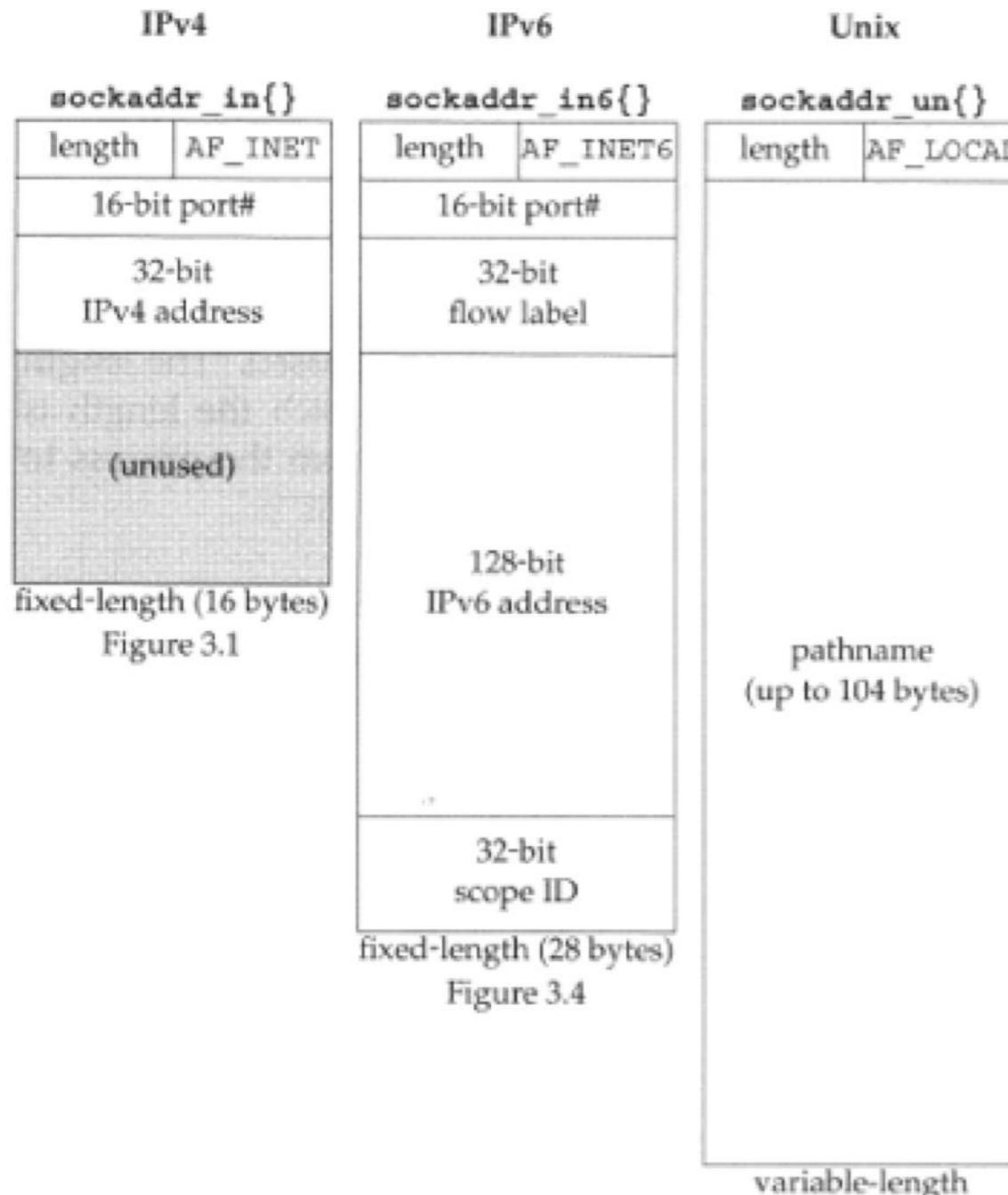
Výplňte tuto strukt.

if ((server = gethostbyname(server_hostname)) == NULL) {
    fprintf(stderr, "ERROR, no such host as %s\n", server_hostname);
    exit(EXIT_FAILURE);
}

bzero((char *) &server_address, sizeof(server_address));
server_address.sin_family = AF_INET; Cílová síťovina
bcopy((char *)server->h_addr, (char *)&server_address.sin_addr.s_addr,
         server->h_length);
server_address.sin_port = htons(port_number); Převodír LEB (x16) na BE kódov
                                                přesnou hodnotu

{
    Cílové porty
```

Adresové struktury



```

struct in_addr {
    in_addr_t s_addr;           /* 32-bit IPv4 address */
                                /* network byte ordered */

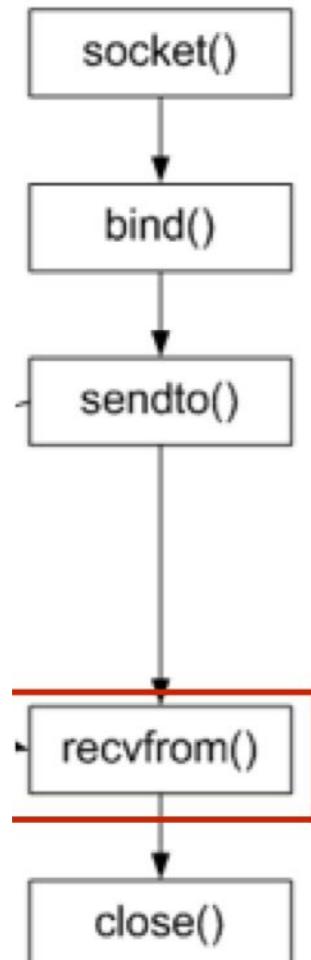
};

struct sockaddr_in {
    uint8_t    sin_len;        /* length of structure (16) */
    sa_family_t sin_family;   /* AF_INET */
    in_port_t   sin_port;     /* 16-bit TCP or UDP port number */
                            /* network byte ordered */
    struct in_addr sin_addr;  /* 32-bit IPv4 address */
                            /* network byte ordered */
    char       sin_zero[8];   /* unused */
};

```

Příjetí dat

Klient



```
ssize_t  
recvfrom(int socket,  
          void * buffer,  
          size_t length,  
          int flags,  
          struct sockaddr * address,  
          socklen_t * address_len);
```

↓
↳ Buflimit - mít místko
= Buflimit - číslo míst kde zastalo

```
bytesrx = recvfrom(client_socket, buf, BUFSIZE, 0,  
                    (struct sockaddr *) &server_address, &serverlen);  
  
if (bytesrx < 0)  
    perror("ERROR in recvfrom");
```

Server: vytvoření socketu

Server



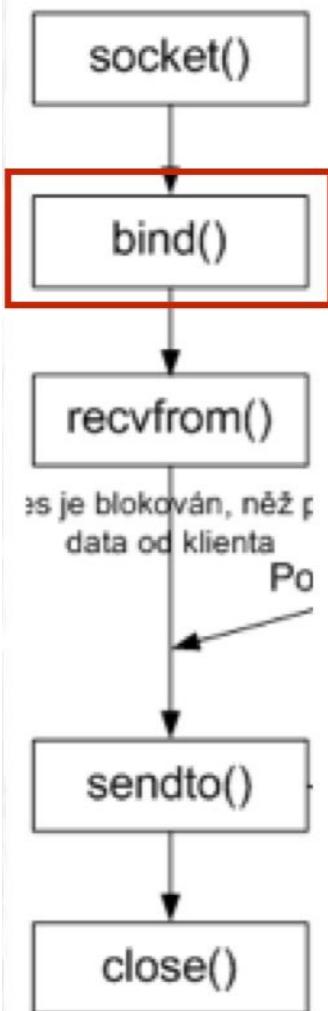
```
if ((server_socket = socket(AF_INET, SOCK_DGRAM, 0)) <= 0)
{
    perror("ERROR in socket");
    exit(EXIT_FAILURE);
}

optval = 1; Zahráváme socketu
setsockopt(server_socket, SOL_SOCKET, SO_REUSEADDR,
           (const void *)&optval , sizeof(int));
```

Server: Bind

Server

```
struct sockaddr_in server_address;  
int port_number;
```

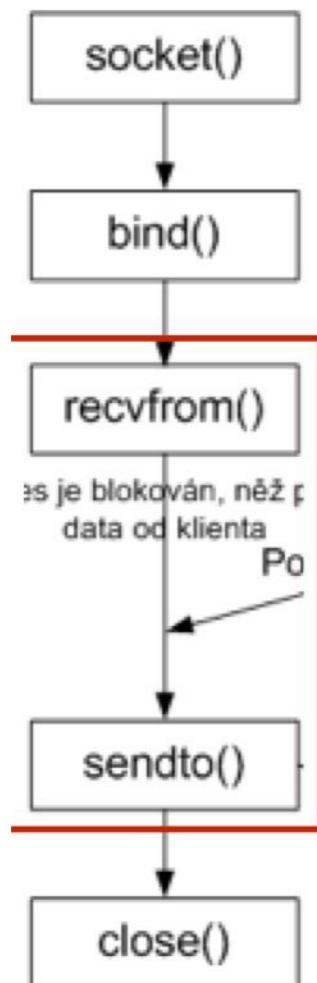


```
bzero((char *) &server_address, sizeof(server_address));  
server_address.sin_family = AF_INET;  
server_address.sin_addr.s_addr = htonl(INADDR_ANY); - Není počet na IP adresy  
server_address.sin_port = htons((unsigned short)port_number);
```

```
if (bind(server_socket, (struct sockaddr *) &server_address,  
         sizeof(server_address)) < 0)  
{  
    perror("ERROR: bind"); ERROR  
    exit(EXIT_FAILURE);  
}
```

Server smyčka

Server



```
while(1)
{
    clientlen = sizeof(client_address);
    bytesrx = recvfrom(server_socket, buf, BUFSIZE, 0,
                      (struct sockaddr *) &client_address, &clientlen);

    if (bytesrx < 0)
        perror("ERROR: recvfrom:");

    bytestx = sendto(server_socket, buf, strlen(buf), 0,
                      (struct sockaddr *) &client_address, clientlen);

    if (bytestx < 0)
        perror("ERROR: sendto:");
}
```

Klient

```
if ((client_socket = socket(AF_INET, SOCK_DGRAM, 0)) <= 0)
{
    perror("ERROR in socket");
    exit(EXIT_FAILURE);
}
```

```
if ((server = gethostbyname(server_hostname)) == NULL) {
    fprintf(stderr, "ERROR, no such host as %s\n", server_hostname);
    exit(EXIT_FAILURE);
}

bzero((char *) &server_address, sizeof(server_address));
server_address.sin_family = AF_INET;
bcopy((char *)server->h_addr, (char *)&server_address.sin_addr.s_addr,
      server->h_length);

server_address.sin_port = htons(port_number);
```

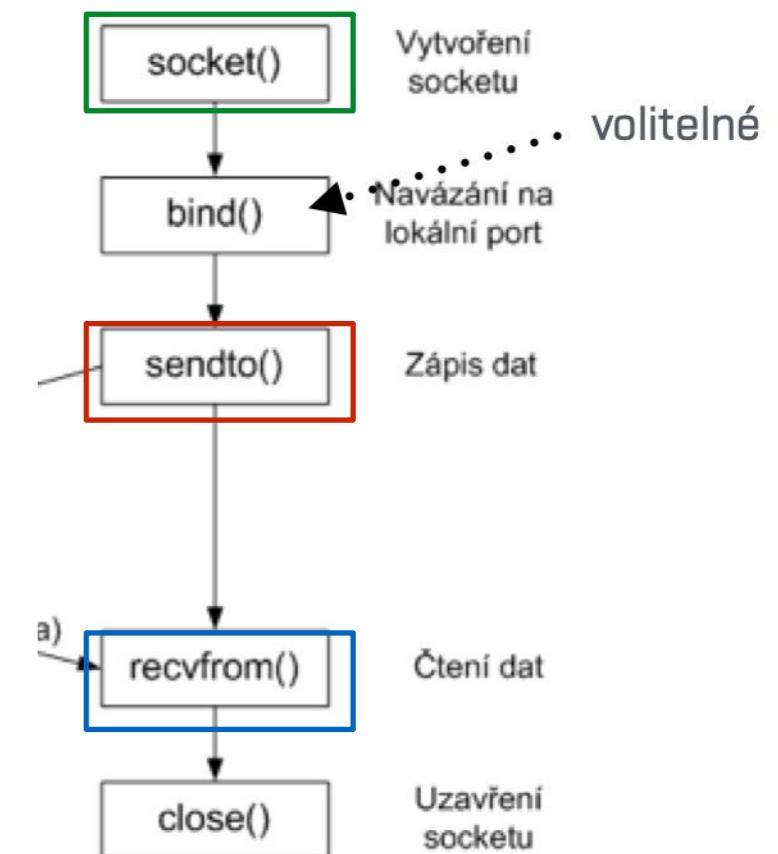
```
serverlen = sizeof(server_address);
bytestx = sendto(client_socket, buf, strlen(buf), 0,
                 (struct sockaddr *) &server_address, serverlen);

if (bytestx < 0)
    perror("ERROR in sendto");
```

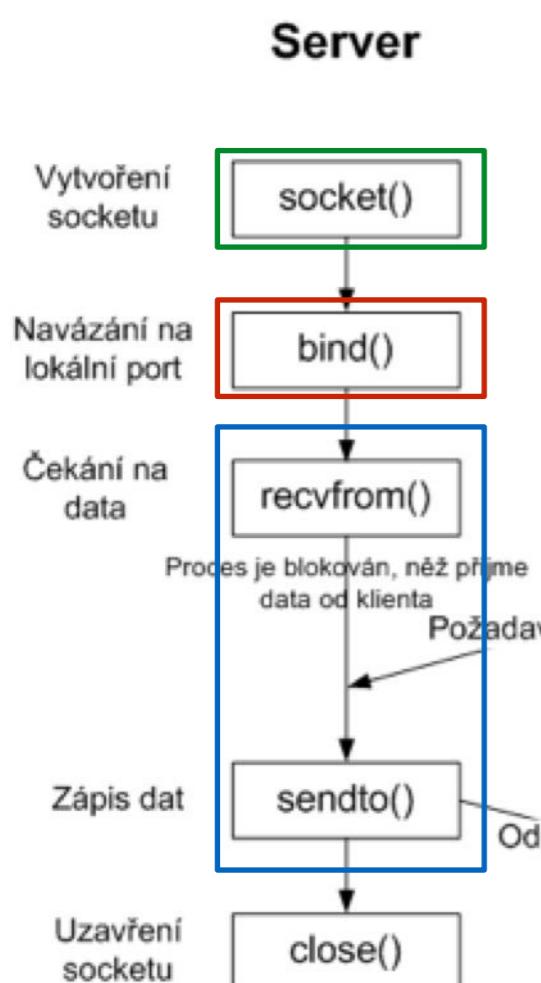
```
bytesrx = recvfrom(client_socket, buf, BUFSIZE, 0,
                    (struct sockaddr *) &server_address, &serverlen);

if (bytesrx < 0)
    perror("ERROR in recvfrom");
```

Klient



Server



```
if ((server_socket = socket(AF_INET, SOCK_DGRAM, 0)) <= 0)
{
    perror("ERROR in socket");
    exit(EXIT_FAILURE);
}
```

```
optval = 1;
setsockopt(server_socket, SOL_SOCKET, SO_REUSEADDR,
            (const void *)&optval , sizeof(int));
```

```
struct sockaddr_in server_address;
int port_number;
```

```
bzero((char *) &server_address, sizeof(server_address));
server_address.sin_family = AF_INET;
server_address.sin_addr.s_addr = htonl(INADDR_ANY);
server_address.sin_port = htons((unsigned short)port_number);
```

```
if (bind(server_socket, (struct sockaddr *) &server_address,
          sizeof(server_address)) < 0)
{
    perror("ERROR: bind");
    exit(EXIT_FAILURE);
}
```

```
while(1)
{
    clientlen = sizeof(client_address);
    bytesrx = recvfrom(server_socket, buf, BUFSIZE, 0,
                        (struct sockaddr *) &client_address, &clientlen);

    if (bytesrx < 0)
        perror("ERROR: recvfrom:");

    bytestx = sendto(server_socket, buf, strlen(buf), 0,
                     (struct sockaddr *) &client_address, clientlen);

    if (bytestx < 0)
        perror("ERROR: sendto:");
}
```

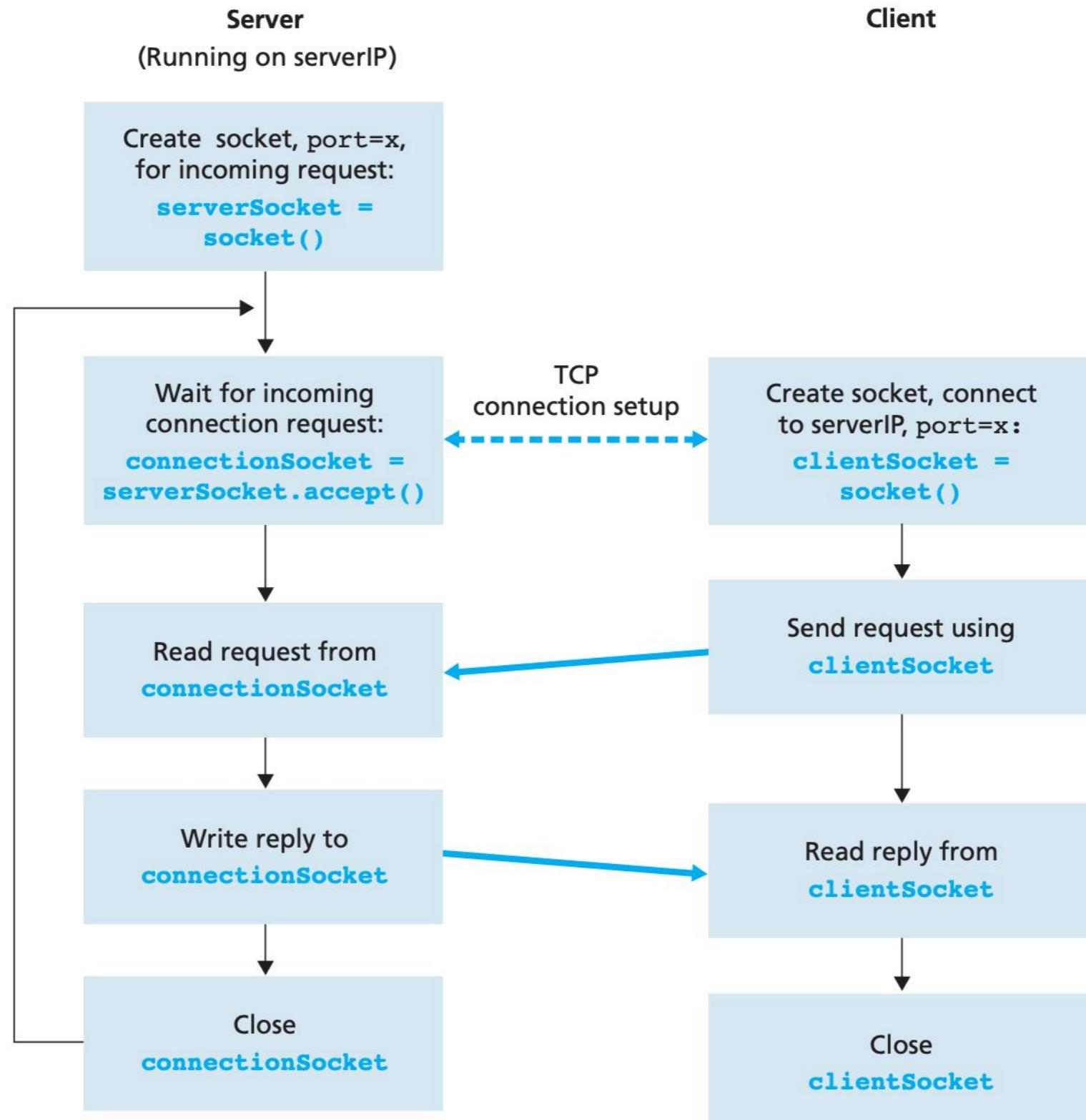
Python

```
1 from socket import *
2 serverName = 'loacalhost'
3 serverPort = 12000
4 clientSocket = socket(socket.AF_INET, socket.SOCK_DGRAM)
5 message = raw_input('Input:')
6 clientSocket.sendto(message, (serverName, serverPort))
7 receivedMessage, serverAddress = clientSocket.recvfrom(2048)
8 print modifiedMessage
9 clientSocket.close()
```

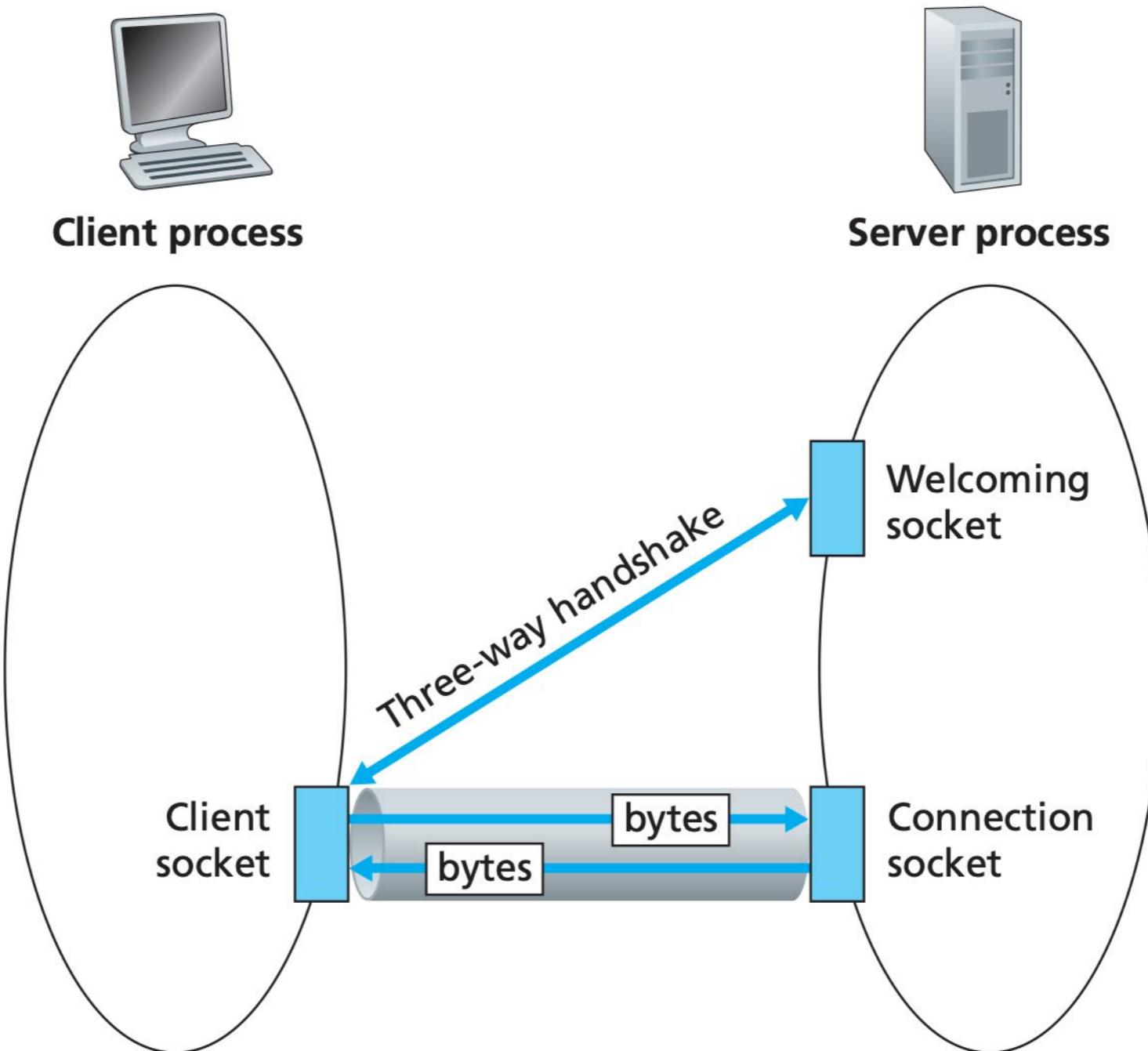
```
1 from socket import *
2 serverPort = 12000
3 serverSocket = socket(AF_INET, SOCK_DGRAM)
4 serverSocket.bind(('', serverPort))
5 print 'server is running'
6 while 1:
7     message, clientAddress = serverSocket.recvfrom(2048)
8     modifiedMessage = message.upper()
9     serverSocket.sendto(modifiedMessgae, clientAddress)
```

TCP komunikace

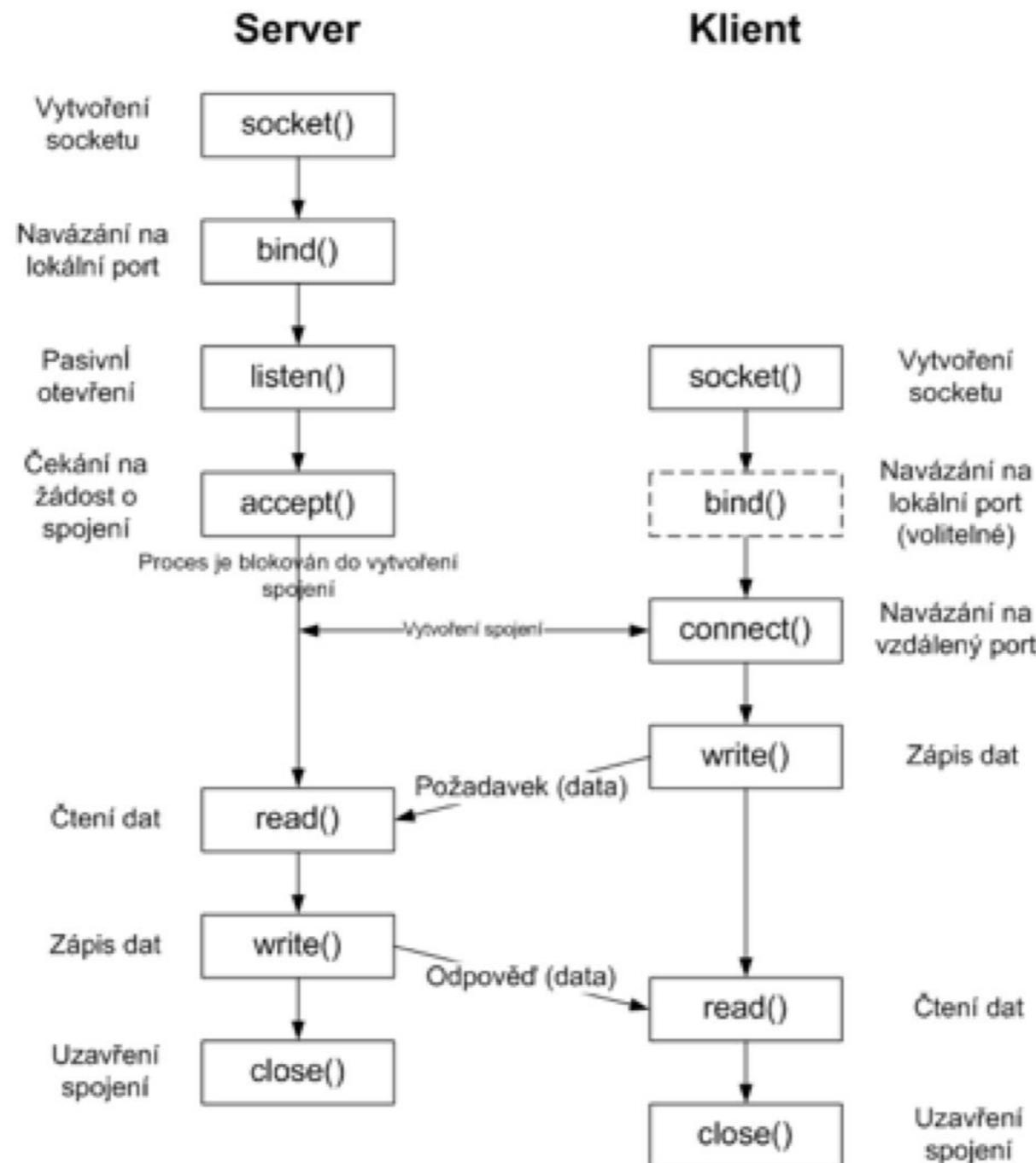
TCP komunikace



Welcome socket

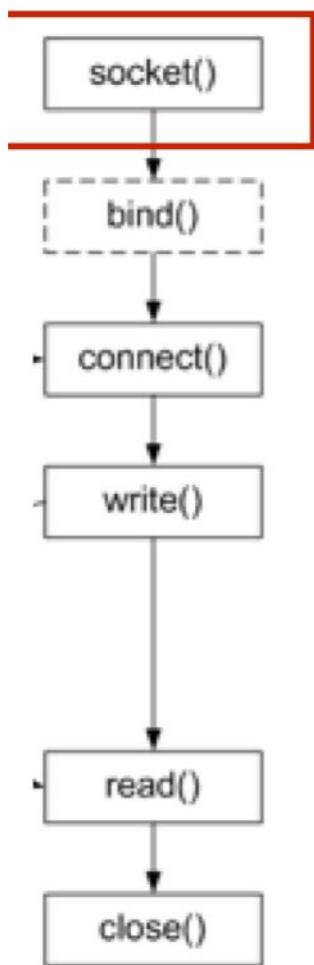


TCP funkce



Klient: Vytvoření socketu

Klient



```
if ((client_socket = socket(AF_INET, SOCK_STREAM, 0)) <= 0)
{
    perror("ERROR: socket");
    exit(EXIT_FAILURE);
}
```

Klient: Connect

NAME

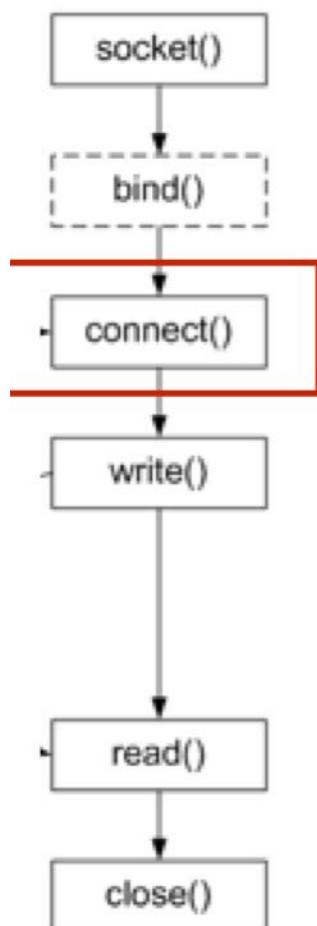
connect -- initiate a connection on a socket

SYNOPSIS

```
#include <sys/types.h>
#include <sys/socket.h>
```

int

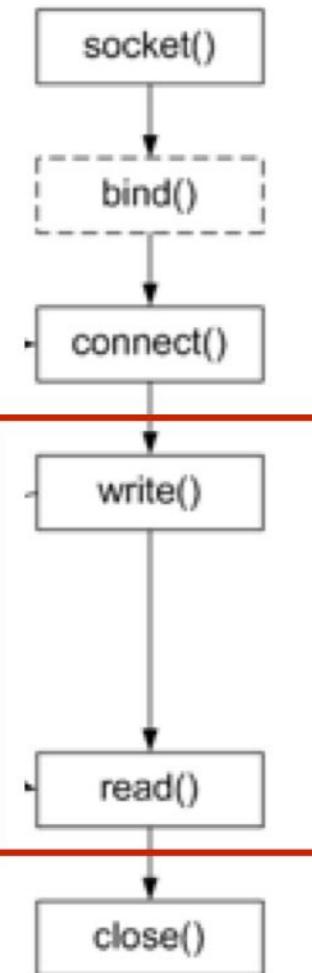
connect(int socket, const struct sockaddr *address, socklen_t address_len);



```
if (connect(client_socket,
            (const struct sockaddr *) &server_address,
            sizeof(server_address)) != 0)
{
    perror("ERROR: connect");
    exit(EXIT_FAILURE);
}
```

Klient: Send/recv

Klient

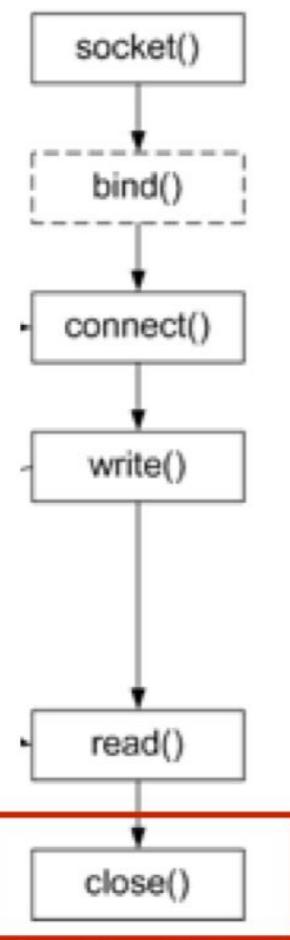


```
bytestx = send(client_socket, buf, strlen(buf), 0);  
if (bytestx < 0)  
    perror("ERROR: sendto");
```

```
bytesrx = recv(client_socket, buf, BUFSIZE, 0);  
if (bytesrx < 0)  
    perror("ERROR: recvfrom");
```

Ukončení spojení

Klient



NAME

close -- delete a descriptor

SYNOPSIS

```
#include <unistd.h>
```

```
int  
close(int fildes);
```

NAME

shutdown -- shut down part of a full-duplex connection

SYNOPSIS

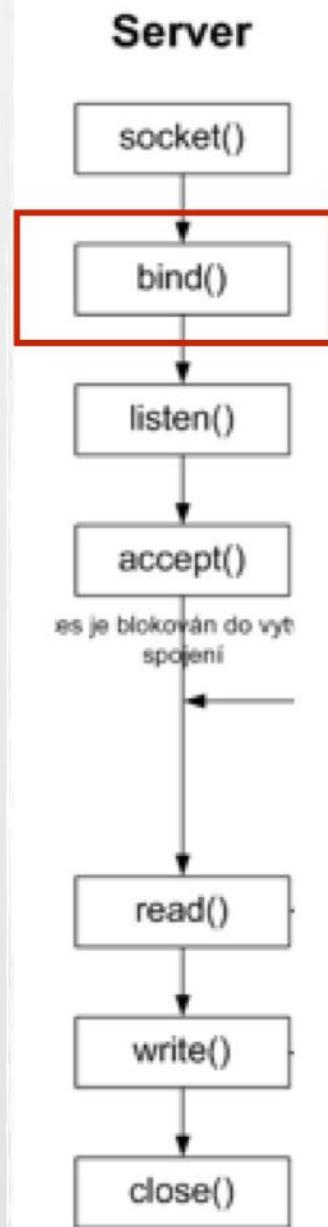
```
#include <sys/socket.h>
```

```
int  
shutdown(int socket, int how);
```

Ako ukončiť spojenie

- iba zápis, iba čítanie

Server: Bind



NAME

bind -- bind a name to a socket

SYNOPSIS

```
#include <sys/socket.h>
```

```
int
```

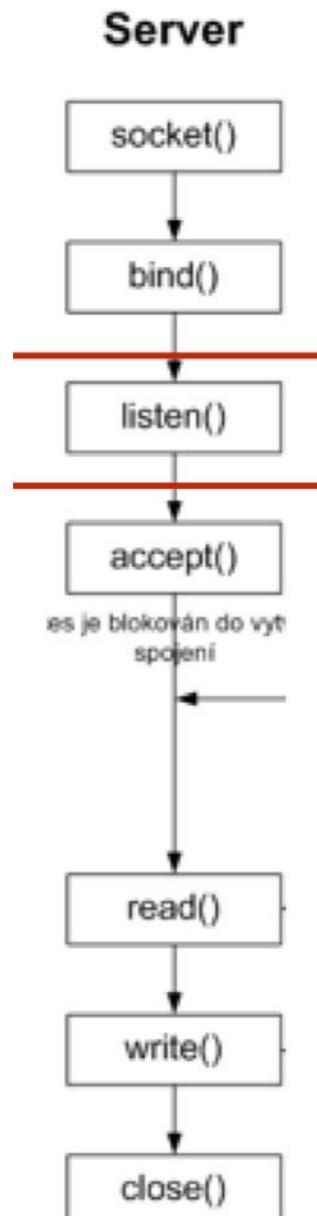
```
bind(int socket, const struct sockaddr *address, socklen_t address_len);
```

```
memset(&sa, 0, sizeof(sa));
sa.sin6_family = AF_INET6;
sa.sin6_addr = in6addr_any;
sa.sin6_port = htons(port_number);
```

```
if ((rc = bind(welcome_socket, (struct sockaddr*)&sa, sizeof(sa))) < 0)
{
    perror("ERROR: bind");
    exit(EXIT_FAILURE);
}
```

Server: Listen

Logický řád zhotování socketu



NAME

listen -- listen for connections on a socket

SYNOPSIS

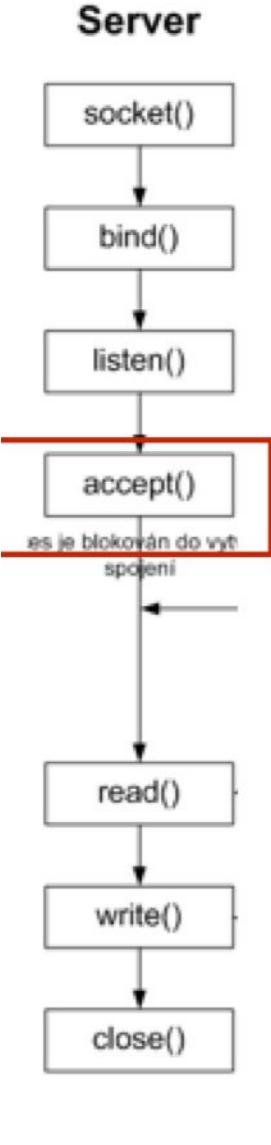
```
#include <sys/socket.h>

int
listen(int socket, int backlog);
```

The backlog parameter defines the maximum length for the queue of pending connections.

```
if ((listen(welcome_socket, 1)) < 0)
{
    perror("ERROR: listen");
    exit(EXIT_FAILURE);
}
```

Server: Accept



accept -- accept a connection on a socket

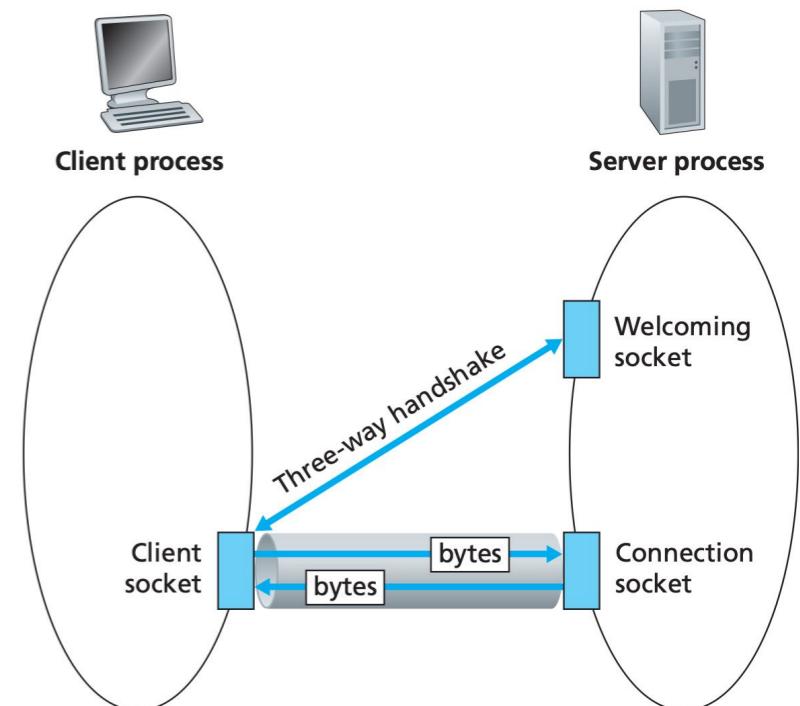
SYNOPSIS

```
#include <sys/socket.h>

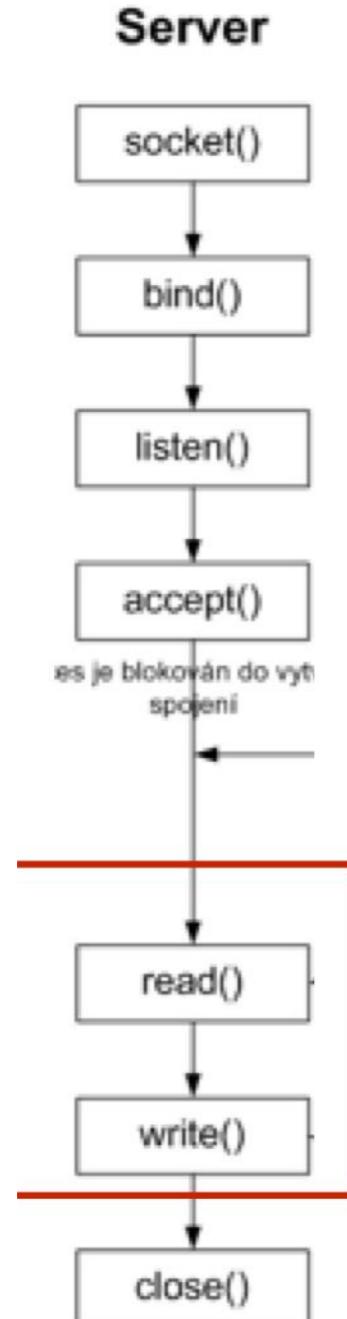
int
accept(int socket, struct sockaddr *restrict address,
       socklen_t *restrict address_len);
```

accept() extracts the first connection request on the queue of pending connections, creates a new socket with the same properties of **socket**, and allocates a new file descriptor for the socket.

```
while(1)
{
    int comm_socket = accept(welcome_socket,
                           (struct sockaddr*)&sa_client, &sa_client_len
    if (comm_socket > 0)
    {
        . . .
    }
}
```



Server: read/write



```
char buff[1024];
int res = 0;
for (;;)
{
    res = recv(comm_socket, buff, 1024, 0);
    if (res <= 0)
        break;

    send(comm_socket, buff, strlen(buff), 0);
}
```

Získání IP adresy

NAME

`getaddrinfo`, `freeaddrinfo` – socket address structure to host and service name

SYNOPSIS

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>

int
getaddrinfo(const char *hostname, const char *servname,
            const struct addrinfo *hints, struct addrinfo **res);

void
freeaddrinfo(struct addrinfo *ai);
```

DESCRIPTION

The `getaddrinfo()` function is used to get a list of addresses and port numbers for host `hostname` and service `servname`. It is a replacement for and provides more flexibility than the `gethostbyname(3)` and `getservbyname(3)` functions.

Pomocné funkce

NAME

htonl, htons, ntohs, ntohl, ntohs - convert values between host and network byte order

SYNOPSIS

```
#include <arpa/inet.h>

uint32_t htonl(uint32_t hostlong);
uint16_t htons(uint16_t hostshort);
uint32_t ntohl(uint32_t netlong);
uint16_t ntohs(uint16_t netshort);
```

DESCRIPTION

The **htonl()** function converts the unsigned integer hostlong from host byte order to network byte order.

The **htons()** function converts the unsigned short integer hostshort from host byte order to network byte order.

The **ntohl()** function converts the unsigned integer netlong from network byte order to host byte order.

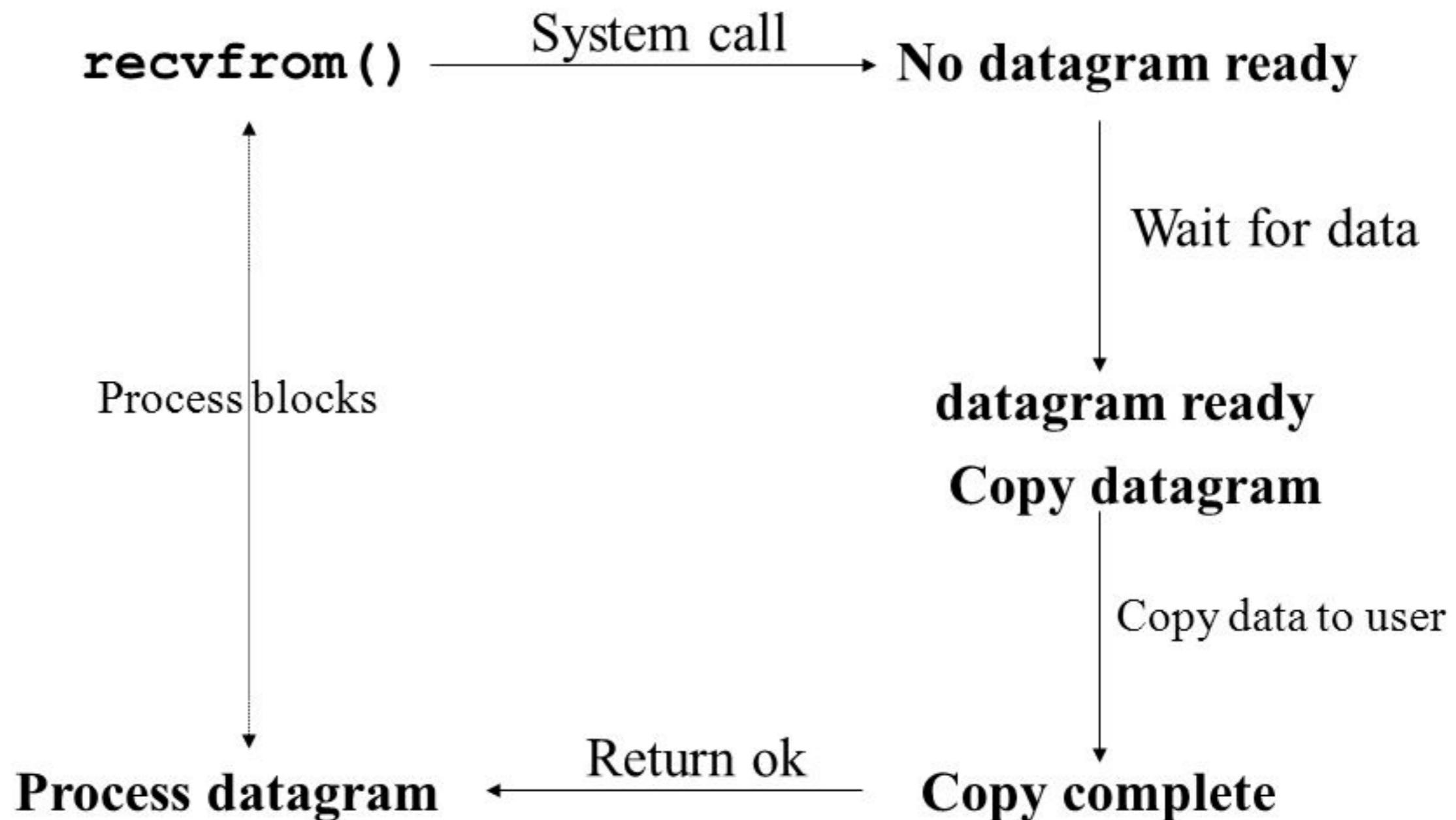
The **ntohs()** function converts the unsigned short integer netshort from network byte order to host byte order.

Neblokující operace

Blocking I/O

Application

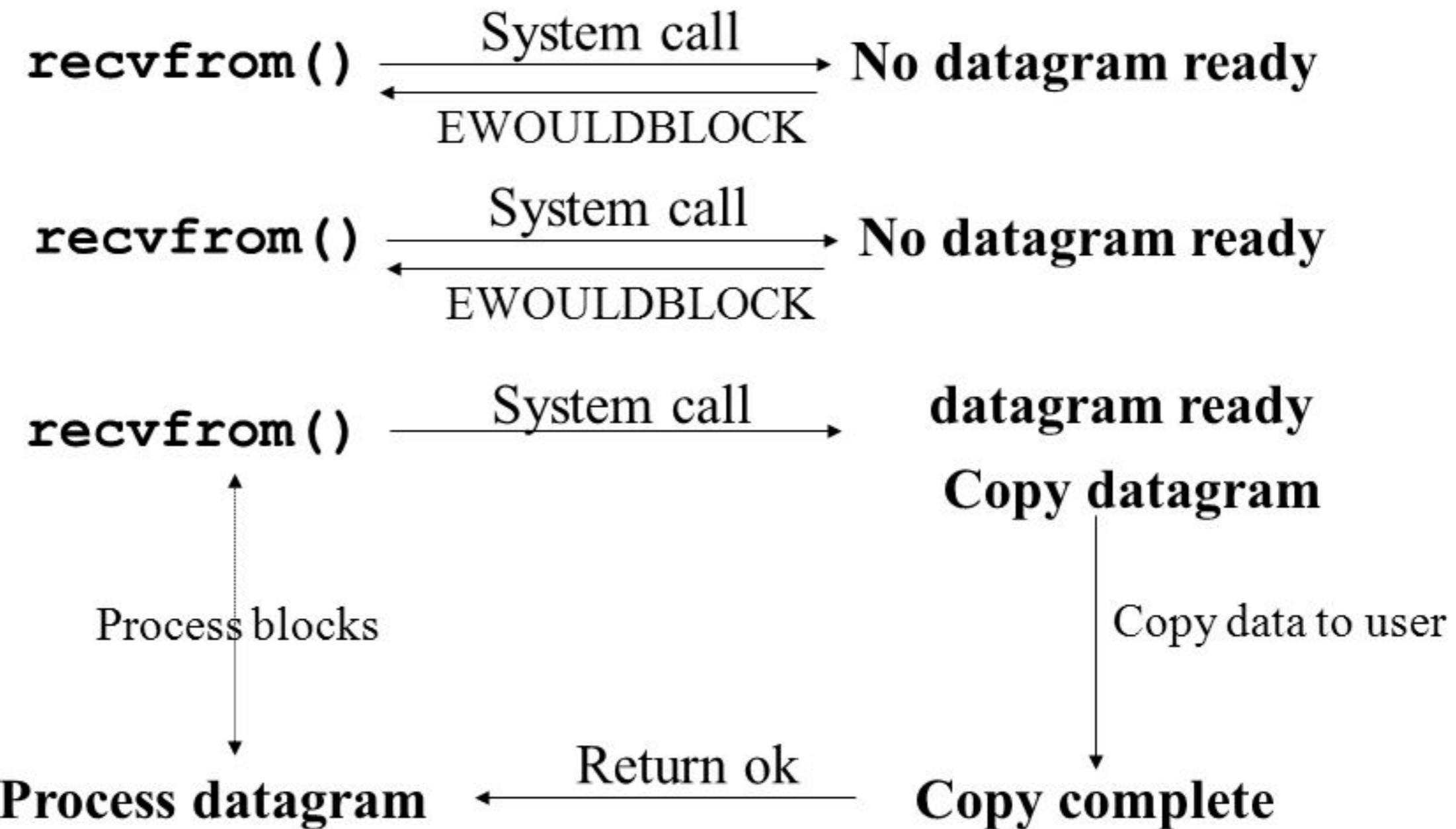
Operating system



Non-Blocking I/O

Application

Operating system



Neblokující socket

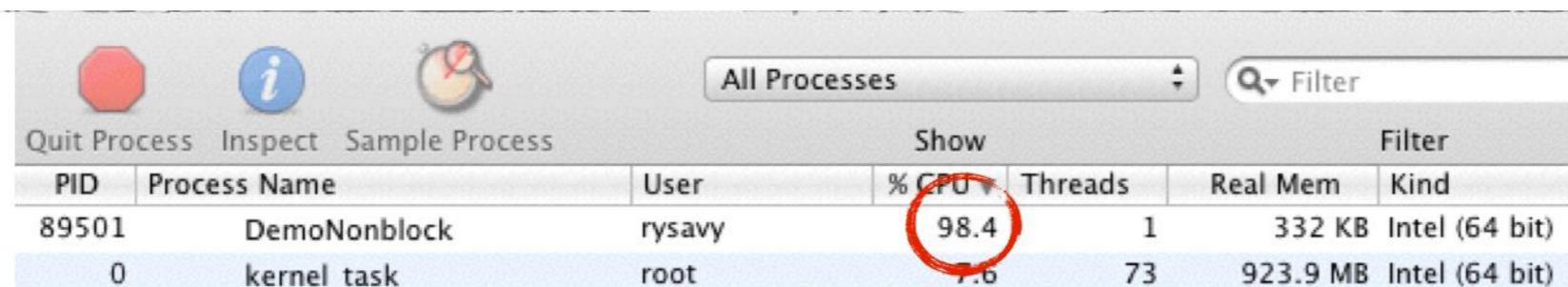
```
int flags = fcntl(comm_socket, F_GETFL, 0);
rc = fcntl(comm_socket, F_SETFL, flags | O_NONBLOCK);

if (rc < 0)
{
    perror("fcntl() failed");
    exit(EXIT_FAILURE);
}
```

Neblokující linku

Příklad - aktivní čekání

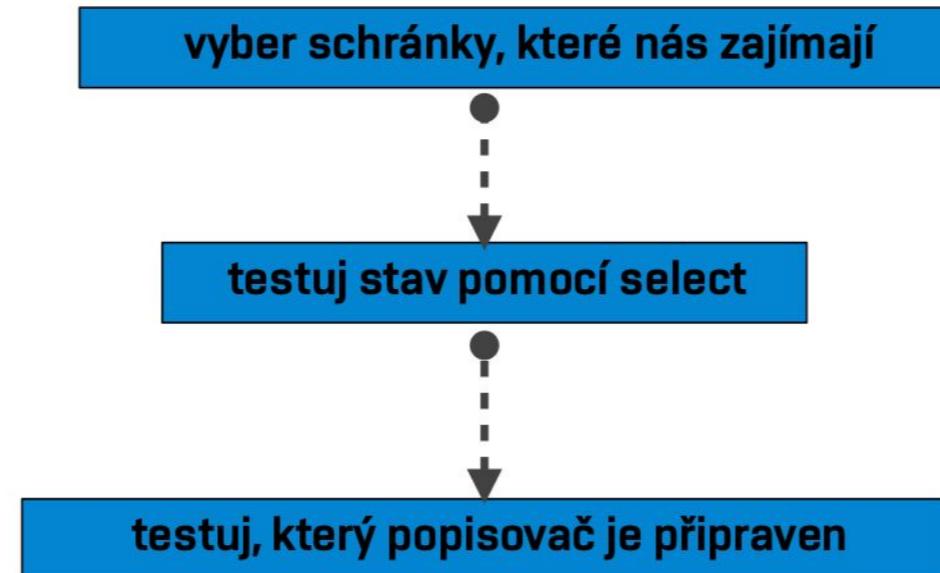
```
struct timeval *timeoutPtr = NULL;  
fd_set read_socks, write_socks;  
fcntl(socket, F_SETFL, O_NONBLOCK);  
  
while (1)  
{  
    if ((len = recv(socket, buffer, BUFFER_LENGTH, 0)) > 0)  
    {  
    }  
    else if (errno == EAGAIN) continue;  
    else if (len == 0) { break; }  
    else  
    {  
        perror("ERROR: recv: ");  
        exit();  
    }  
}
```



The screenshot shows the Activity Monitor application on OS X. The main window displays a list of processes. The first process listed is 'DemoNonblock' with PID 89501, which is highlighted. The second process listed is 'kernel_task' with PID 0. The columns shown are PID, Process Name, User, % CPU, Threads, Real Mem, and Kind. The '% CPU' column for the 'DemoNonblock' process has a red circle around its value, 98.4. The 'Threads' column for the 'DemoNonblock' process also has a red circle around its value, 1.

PID	Process Name	User	% CPU	Threads	Real Mem	Kind
89501	DemoNonblock	rysavy	98.4	1	332 KB	Intel (64 bit)
0	kernel_task	root	7.6	73	923.9 MB	Intel (64 bit)

Neblokující a Select



```
int  
select(int nfds, fd_set *restrict readfds, fd_set *restrict writefds,  
fd_set *restrict errorfds, struct timeval *restrict timeout);
```

Příklad: Select

```
struct timeval *timeoutPtr = NULL;  
fd_set read_socks, write_socks;  
  
fcntl(socket, F_SETFL, O_NONBLOCK);  
  
while (1)  
{  
    FD_ZERO(&read_socks);  
    FD_ZERO(&write_socks);  
  
    FD_SET(socket, &read_socks);  
    FD_SET(socket, &write_socks);  
    select(FD_SETSIZE, &read_socks, &write_socks, (fd_set *) 0, timeoutPtr);  
  
    if (FD_ISSET(socket, &write_socks)) - testuje si socket patří do možnosti  
    {  
        ...  
        write  
    }  
  
    if (FD_ISSET(socket, &read_socks)) - read  
    {  
        ...  
    }  
}
```

Nástroje

Netstat

NAME

netstat -- show network status

SYNOPSIS

```
netstat [-AaLlnW] [-f address family | -p protocol]
netstat [-gilns] [-v] [-f address family] [-I interface]
netstat -i | -I interface [-w wait] [-c queue] [-abdgqRtS]
netstat -s [-s] [-f address family | -p protocol] [-w wait]
netstat -i | -I interface -s [-f address family | -p protocol]
netstat -m [-m]
netstat -r [-Aaln] [-f address family]
netstat -rs [-s]
```

DESCRIPTION

The **netstat** command symbolically displays the contents of various network-related data structures. There are a number of output formats, depending on the options for the information presented. The first form of the command displays a list of active sockets for each protocol. The second form presents the contents of one of the other network data structures according to the option selected. Using the third form, with a wait interval specified, **netstat** will continuously display the information regarding packet traffic on the configured network interfaces. The fourth form displays statistics for the specified protocol or address family. If a wait interval is specified, the protocol information over the last interval seconds will be displayed. The fifth form displays per-interface statistics for the specified protocol or address family. The sixth form displays mbuf(9) statistics. The seventh form displays routing table for the specified address family. The eighth form displays routing statistics.

Netstat příklad

```
eva ~> netstat -Lnap tcp
```

Current listen queue sizes (qlen/incqlen/maxqlen)

Proto	Listen	Local Address
tcp4	0/0/10	* .55555
tcp4	0/0/5	* .45000
tcp4	0/0/128	127.0.0.1.8080
tcp6	0/0/128	::1.8080
tcp4	0/0/50	147.229.176.14.3306
IPv6	tcp6 0/0/10	* 465 <i>(fsln pltlu)</i>
	tcp6 0/0/10	* 587
	tcp6 0/0/10	* 25
IPv4	tcp4 0/0/10	* 465
	tcp4 0/0/10	* 587
	tcp4 0/0/10	* 25

netcat

Nástroj, který umožňuje vytvořit TCP spojení nebo poslat UDP datagram.

```
nc kazi.fit.vutbr.cz 25
```

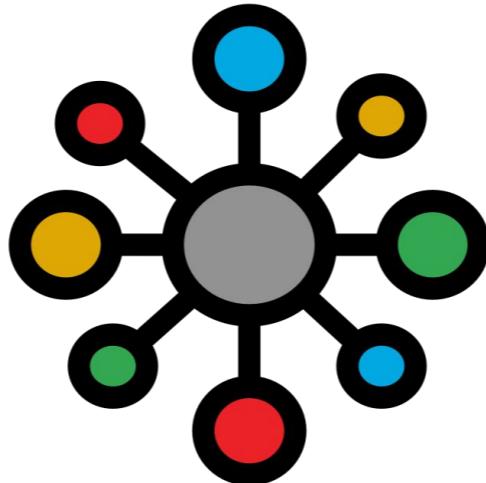
```
220 kazi.fit.vutbr.cz ESMTP Sendmail 8.16.1/8.16.1; Mon, 15 Feb  
2021 17:23:36 +0100 (CET)
```

```
printf "GET /index.html HTTP/1.0\r\nHost:www.fit.vutbr.cz\r\n\r\n" | nc  
www.fit.vutbr.cz 80
```

```
HTTP/1.1 404 Not Found  
Date: Mon, 15 Feb 2021 16:25:04 GMT  
Server: Apache  
Connection: close  
Content-Type: text/html; charset=iso-8859-2
```

Kontakty

Ondřej Ryšavý <rysav@fit.vutbr.cz>



<https://nesfit.github.io>

<https://www.fit.vutbr.cz/research/groups/nes@fit/>