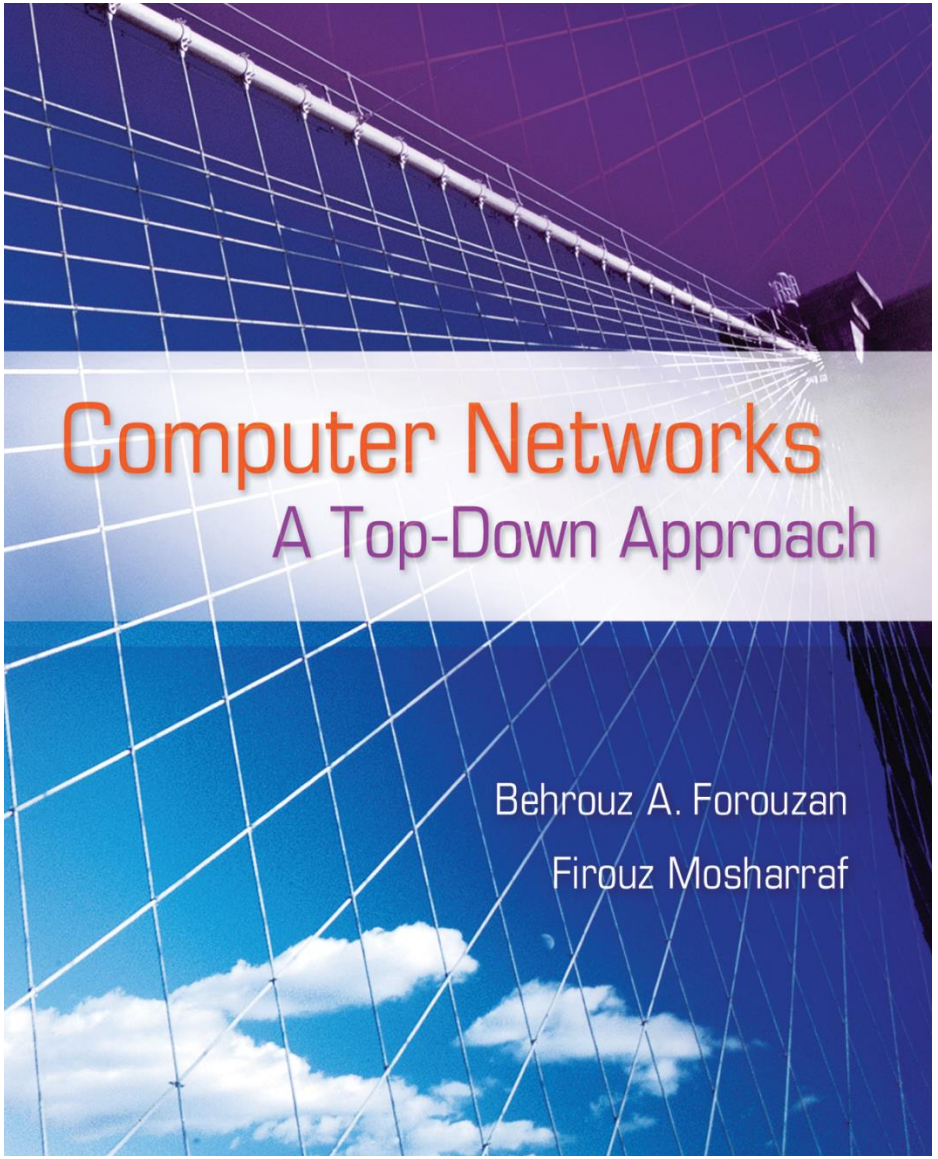
Chapter 2

Application Layer



Chapter 2: Outline

2.1 INTRODUCTION

2.2 CLIENT-SERVER PARADIGM

2.3 STANDARD APPLICATIONS

2.4 PEER-TO-PEER PARADIGM

2.5 SOCKET-INTERFACE PROGRAMMING

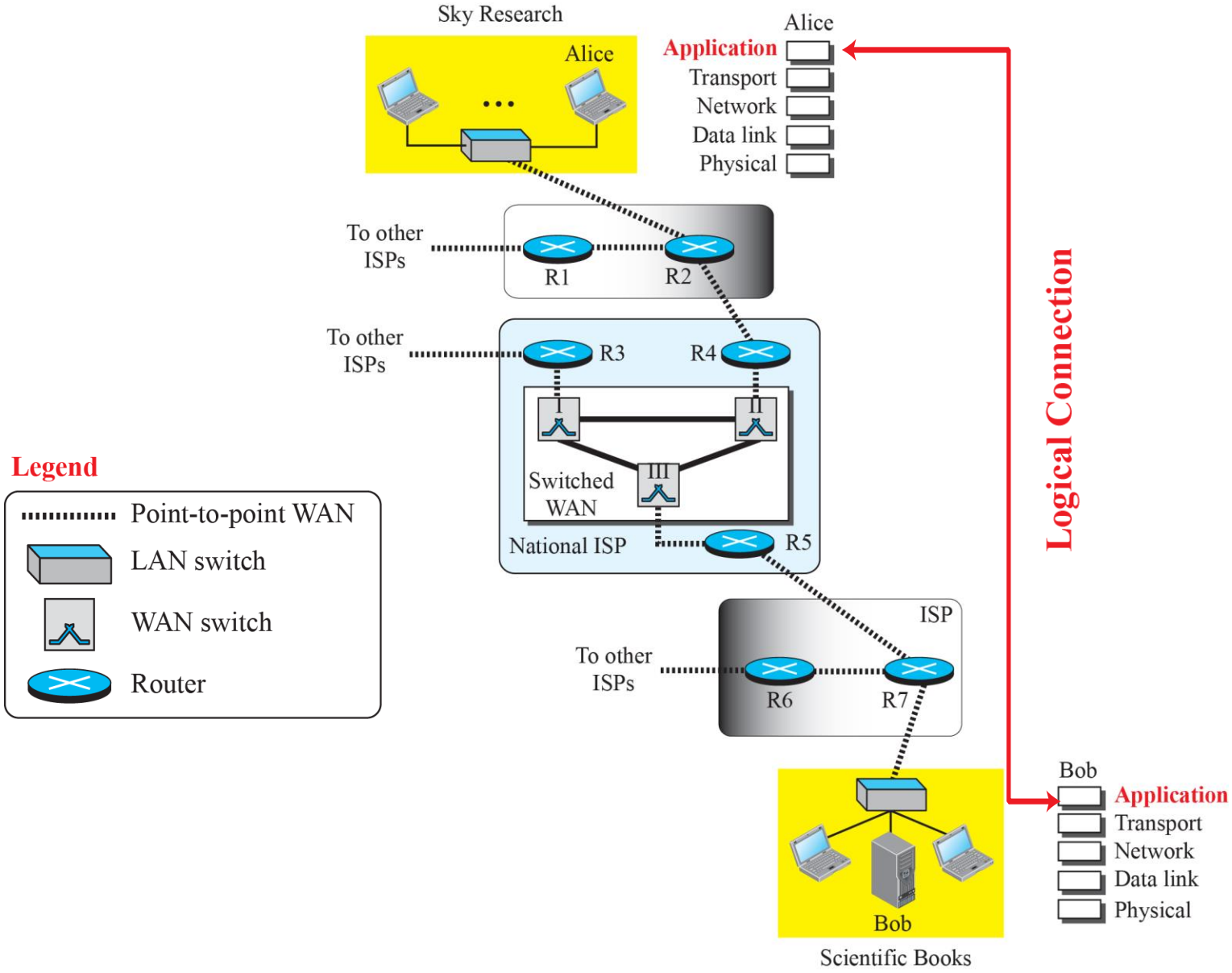
Chapter 2: Objective

- *We introduce the nature of services provided by the Internet: the client-server paradigm and the peer-to-peer paradigm.*
- *We discuss the concept of the client-server paradigm.*
- *We discuss some predefined or standard applications based on the client-server paradigm such as surfing the Web, file transfer, e-mail, and so on.*
- *We show how a new application can be created in the client-server paradigm by writing two programs in the C language.*

2-1 INTRODUCTION

- ***The application layer provides services to the user.***
- ***Communication is provided using a logical connection.***
- ***Which means that the two application layers assume that there is an imaginary direct connection through which they can send and receive messages.***
- ***Figure 2.1 in the next slide shows the idea behind this logical connection.***

Figure 2.1: Logical connection at the application layer



2.1.1 Providing Services

- *The Internet to provide service to users.*
- *Application layer as interface and platform to new application & application protocols to be easily added to the Internet*
- *Growth of Internet and application protocols.*
- *today Internet and application protocols are being added constantly.*

2.1.1 Providing Services (Cont.)

- **Standard and Nonstandard Protocols**
  - **Standard Application-Layer Protocols**
  - **Nonstandard Application-Layer Protocols**

2.1.2 Application-Layer Paradigm

- *Internet requires two processes on two application programs to interact with each other.*
- *Source and destination interaction.*
- *Two programs need to send messages to each other through the Internet infrastructure.*
- *Require to discussed relationship between these programs.*
- *Should both application programs be able to request services and provide services, or should the application programs just do one or the other?*

2.1.2 Application-Layer Paradigm (cont)

- *Traditional Paradigm: Client-Server*
- *New Paradigm: Peer-to-Peer*
- *Mixed Paradigm*

Figure 2.2: Example of a client-server paradigm

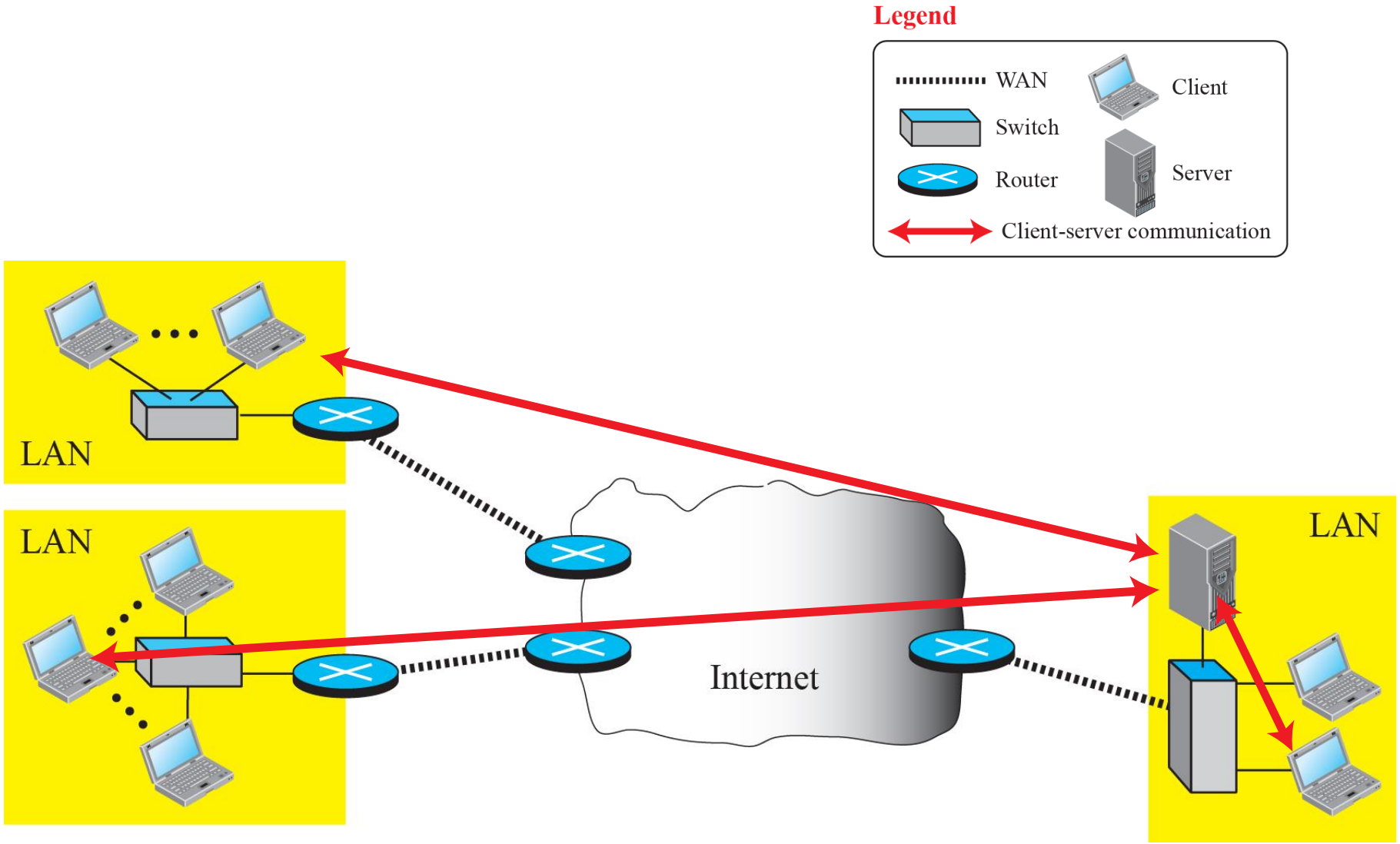
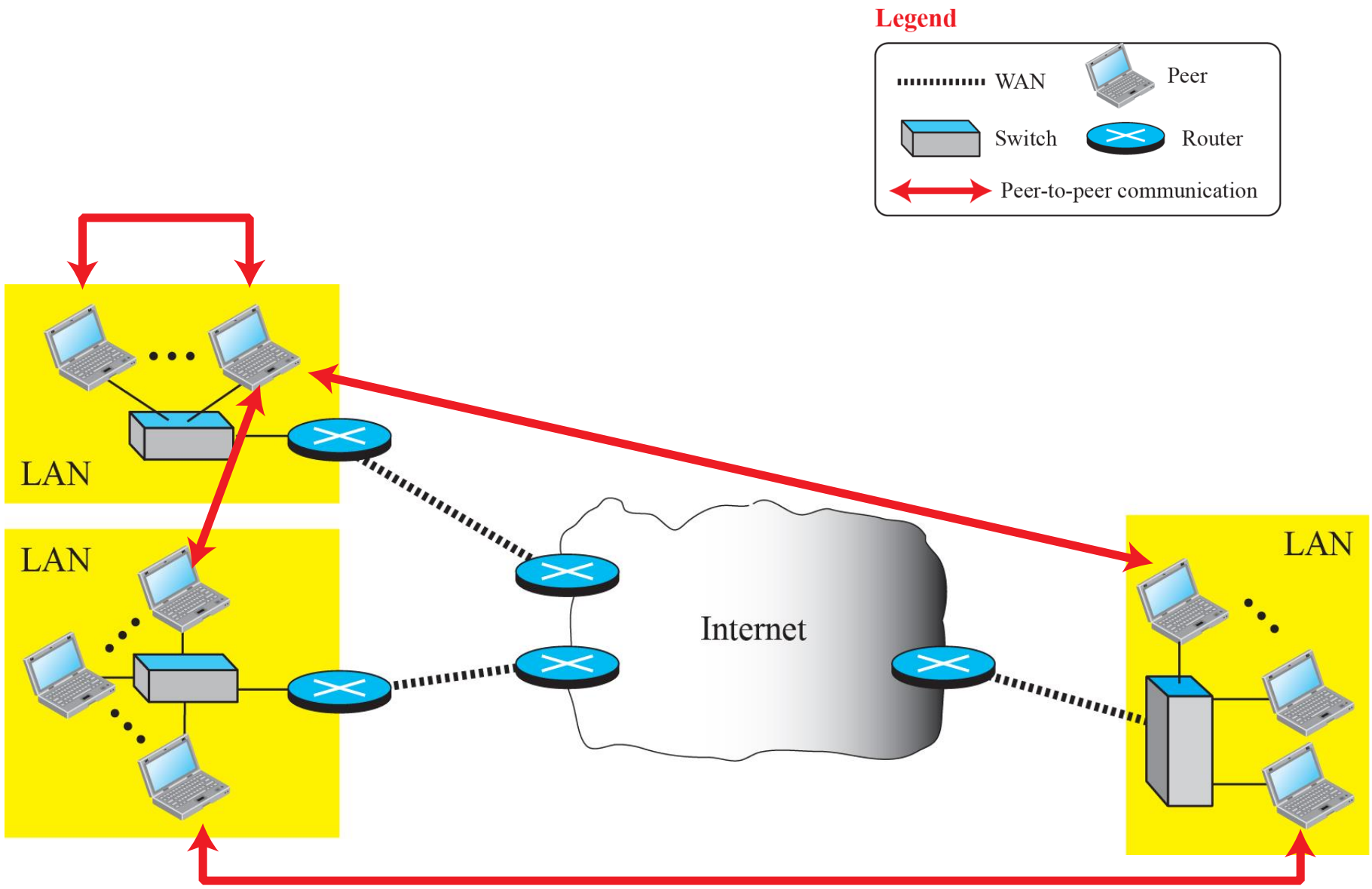


Figure 2.3: Example of a peer-to-peer paradigm



2-2 CLIENT-SERVER PARADIGM

In this paradigm:

- *communication at the application layer is between two running application programs called processes: a client and a server.*
- *A client is a running program that initializes the communication by sending a request;*
- *A server is another application program that waits for a request from a client.*

2.2.1 Application Programming Interface

- *A computer language has*
  - *a set of instructions for mathematical operations,*
  - *a set of instructions for string manipulation,*
  - *a set of instructions for input/ output access, and so on.*
- *If we need a process to be able to communicate with another process, we need*
  - *a new set of instructions to tell the lowest four layers of the TCP/IP suite to open the connection.*
  - *Send and Receive data from the other end, and*
  - *Close the connection.*
  - *A set of instructions of this kind is normally referred to as Application Programming Interface (API).*

2.2.1 (continued)

- *Sockets*
- *Socket Addresses*
- *Finding Socket Addresses*
  - *Server Site*
  - *Client Site*

Figure 2.4: Position of the socket interface

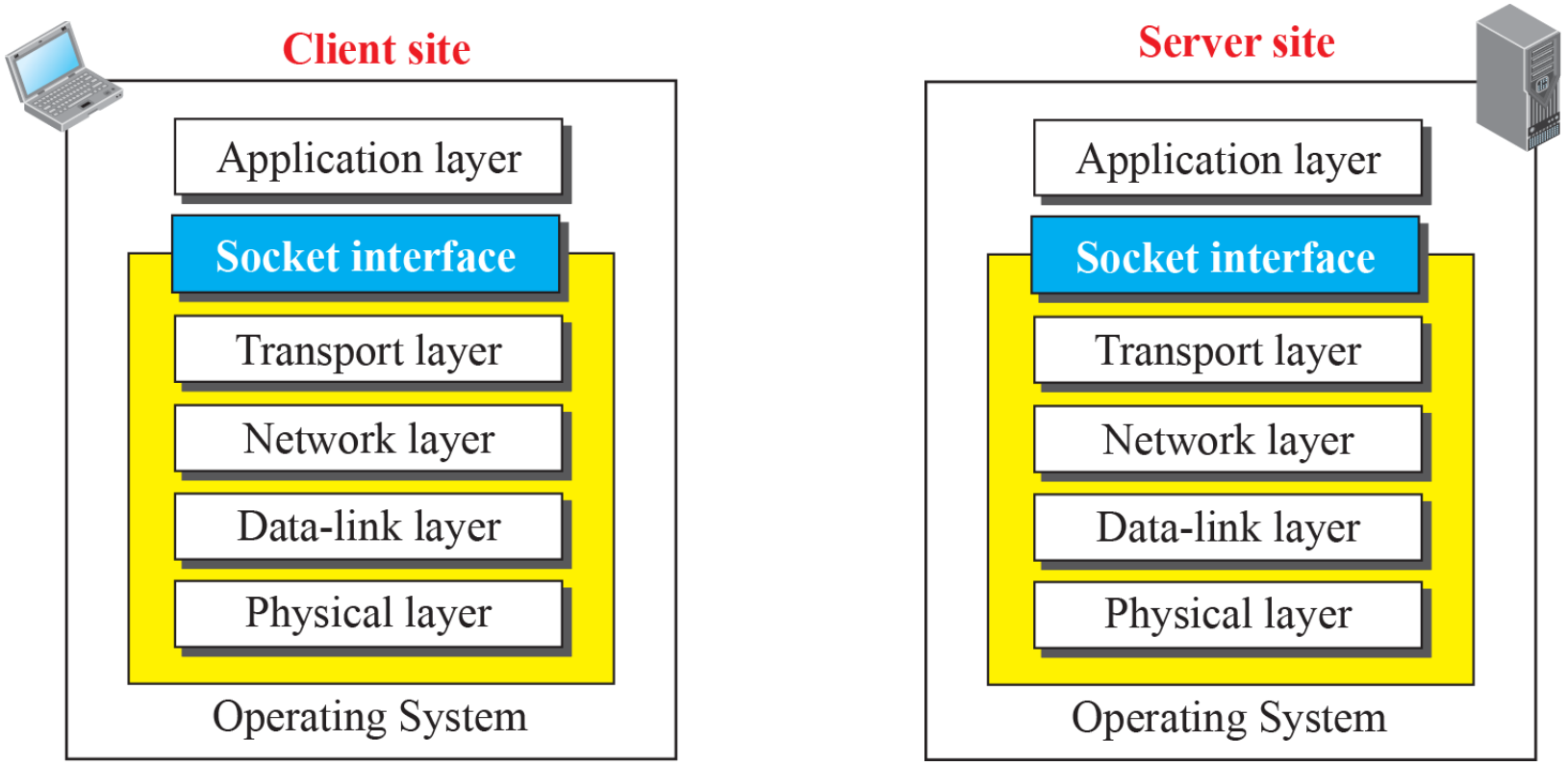


Figure 2.5: A Sockets used like other sources and sinks

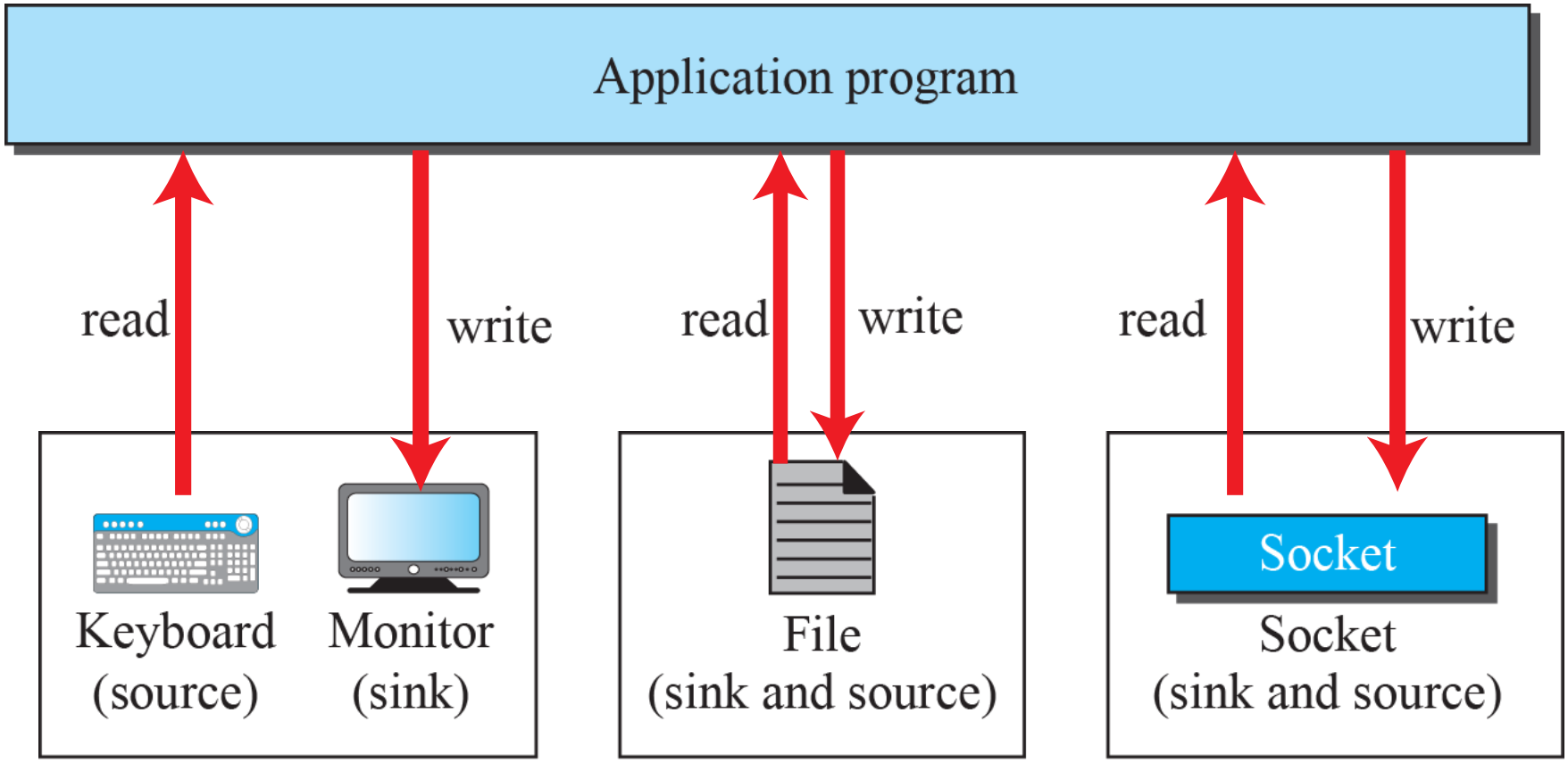


Figure 2.6: Use of sockets in process-to-process communication

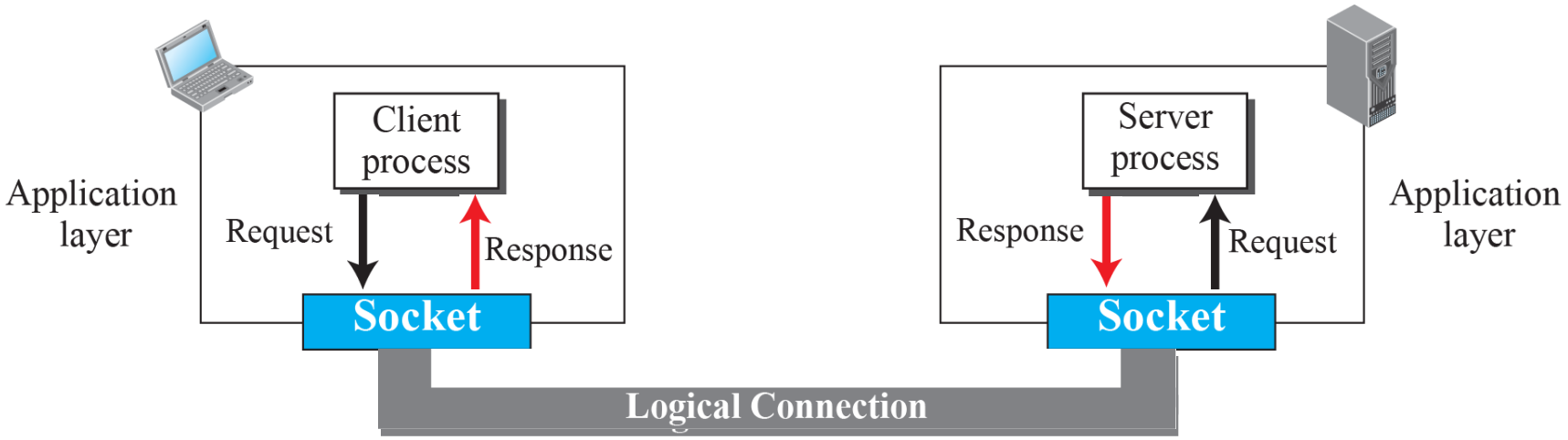


Figure 2.7: A socket address

Example 2.1 Analogy – telephone & socket

- ***We can find a two-level address in telephone communication.***
- ***A telephone number can define an organization, and an extension can define a specific connection in that organization.***
  - *The telephone number in this case is similar to the IP address, which defines the whole organization;*
  - *the extension is similar to the port number, which defines the particular connection.*

2.2.2 Using Services of Transport Layer

- *A pair of processes provide services to the users of the Internet, human or programs.*
- *A pair of processes, however, need to use the services provided by the transport layer for communication because there is no physical communication at the application layer.*
- *There are three common transport layer protocols in the TCP/IP suite: UDP, TCP, and SCTP.*

- ❑ *UDP Protocol*
- ❑ *TCP Protocol*
- ❑ *SCTP Protocol*

2-3 STANDARD CLIENT-SERVER APPLICATIONS

- *During the lifetime of the Internet, several application programs have been developed.*
- *We do not have to redefine them, but we need to understand what they do.*
- *For each application, we also need to know the options available to us.*
- *The study of these applications can help us to create customized applications in the future.*

2.3.1 World Wide Web and HTTP

- *In this section, we first introduce the World Wide Web (abbreviated WWW or Web).*
- *We then discuss the Hyper Text Transfer Protocol (HTTP), the most common client-server application program used in relation to the Web.*

2.3.1 (continued)

- *World Wide Web*
  - *Architecture*
  - *Uniform Resource Locator (URL)*
  - *Web Documents*
- *HyperText Transfer Protocol (HTTP)*
  - *Nonpersistent versus Persistent Connections*
  - *Message Formats*
  - *Conditional Request*
  - *Cookies*
- *Web Caching: Proxy Server*
  - *Proxy Server Location*
  - *Cache Update*
- *HTTP Security*

Example 2.2

Assume we need to retrieve a scientific document that contains *one reference to another text file and one reference to a large image*. Figure 2.8 shows the situation.

- The main document and the image are stored in two separate files in the *same site* (file A and file B);
- The referenced text file is stored in *another site* (file C).
- So we are dealing with *three different files*, we *need three transactions* if we want to see/get the *whole document*.

Figure 2.8: HTTP Example 2.2 (Retrieving two files and one image)

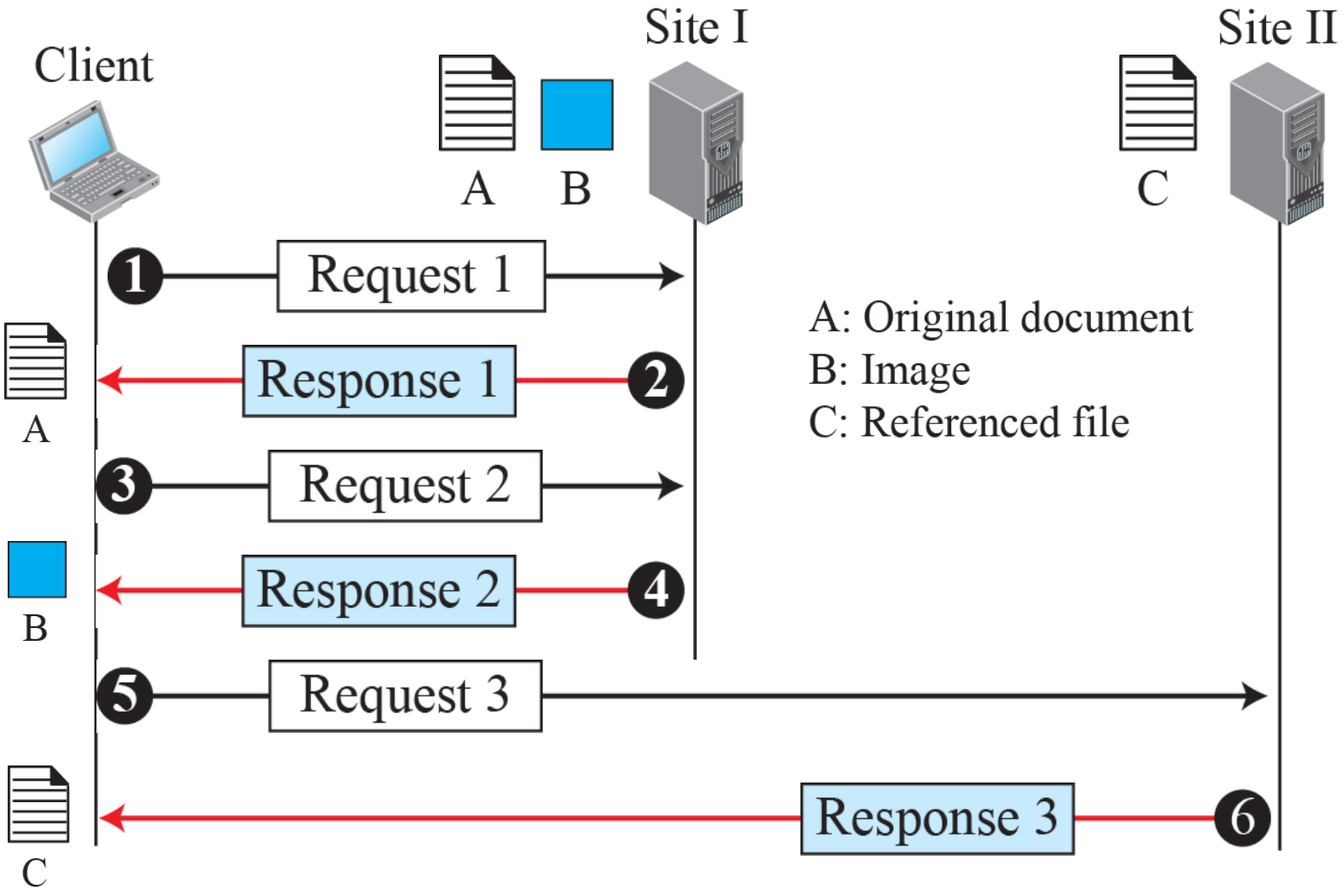
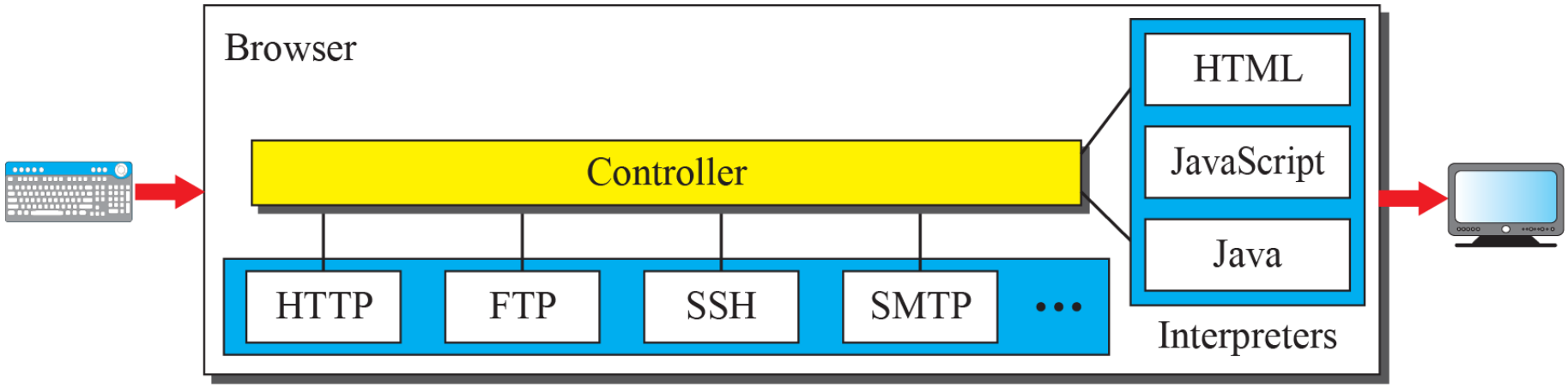


Figure 2.9: Browser



Example 2.3

The URL **http://www.mhhe.com/engcs/compsci/forouzan/** defines the web page related to one of the computer in the McGraw-Hill company (the three letters www are part of the host name and are added to the commercial host).

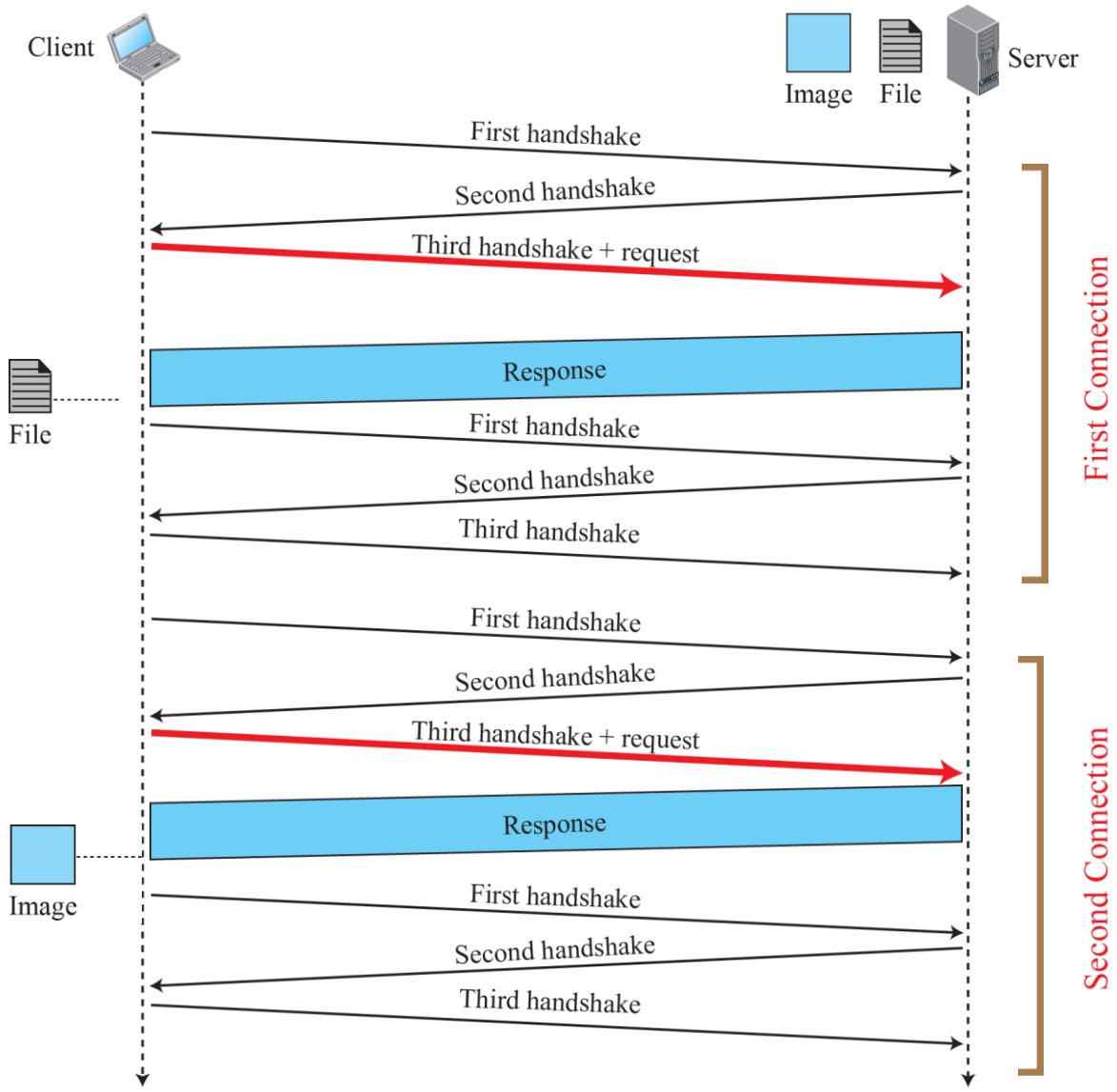
The path is *compsci/forouzan/*, which defines Forouzan's web page under the directory *compsci* (computer science).

Example 2.4

Figure 2.10 shows an example of a *nonpersistent* connection

- The client needs to access a file that contains one link to an image. The text file and image are located on the same server.
- Here we need **two connections**.
- For each connection, TCP requires **at least three handshake messages** to establish the connection, but the request can be sent with the third one. After the connection is established, the object can be transferred.
- After receiving an object, another three handshake messages are needed to terminate the connection, as we will see in Chapter 3.

Figure 2.10: Example 2.4 (HTTP non-persistent connection)



Example 2.5

Figure 2.11 shows the same scenario as in Example 2.4, but using a *persistent* connection.

- Only one connection establishment and connection termination is used, but the request for the image is sent separately.

Figure 2.11: Example 2.5 (HTTP persistent connection)

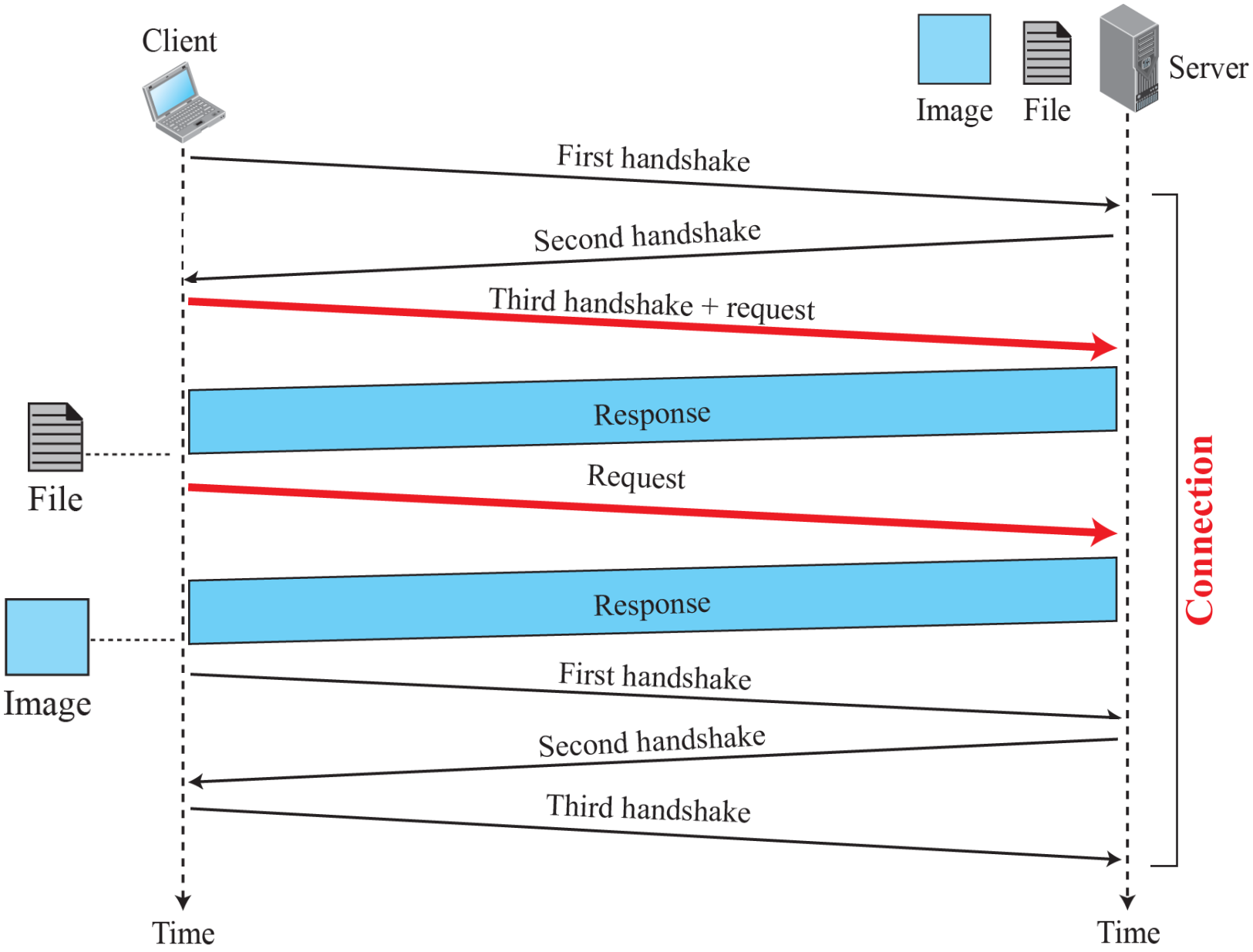


Figure 2.12: Formats of the request and response messages

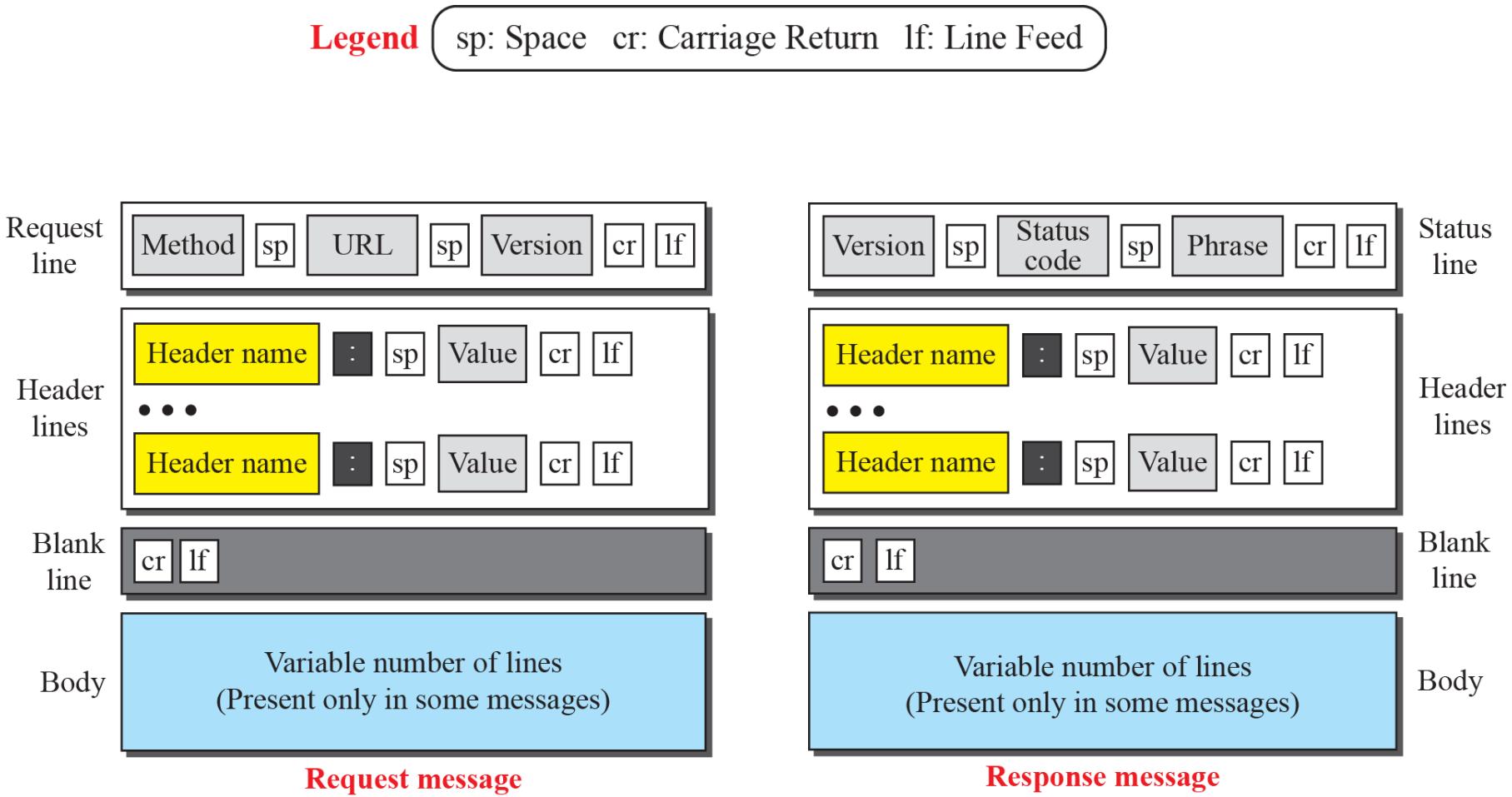


Table 2.1: Methods

Method	*Action*
GET	Requests a document from the server
HEAD	Requests information about a document but not the document itself
PUT	Sends a document from the client to the server
POST	Sends some information from the client to the server
TRACE	Echoes the incoming request
DELETE	Removes the web page
CONNECT	Reserved
OPTIONS	Inquires about available options

Table 2.2: Request Header Names

Header	*Description*
User-agent	Identifies the client program
Accept	Shows the media format the client can accept
Accept-charset	Shows the character set the client can handle
Accept-encoding	Shows the encoding scheme the client can handle
Accept-language	Shows the language the client can accept
Authorization	Shows what permissions the client has
Host	Shows the host and port number of the client
Date	Shows the current date
Upgrade	Specifies the preferred communication protocol
Cookie	Returns the cookie to the server (explained later)
If-Modified-Since	If the file is modified since a specific date

Table 2.3: Response Header Names

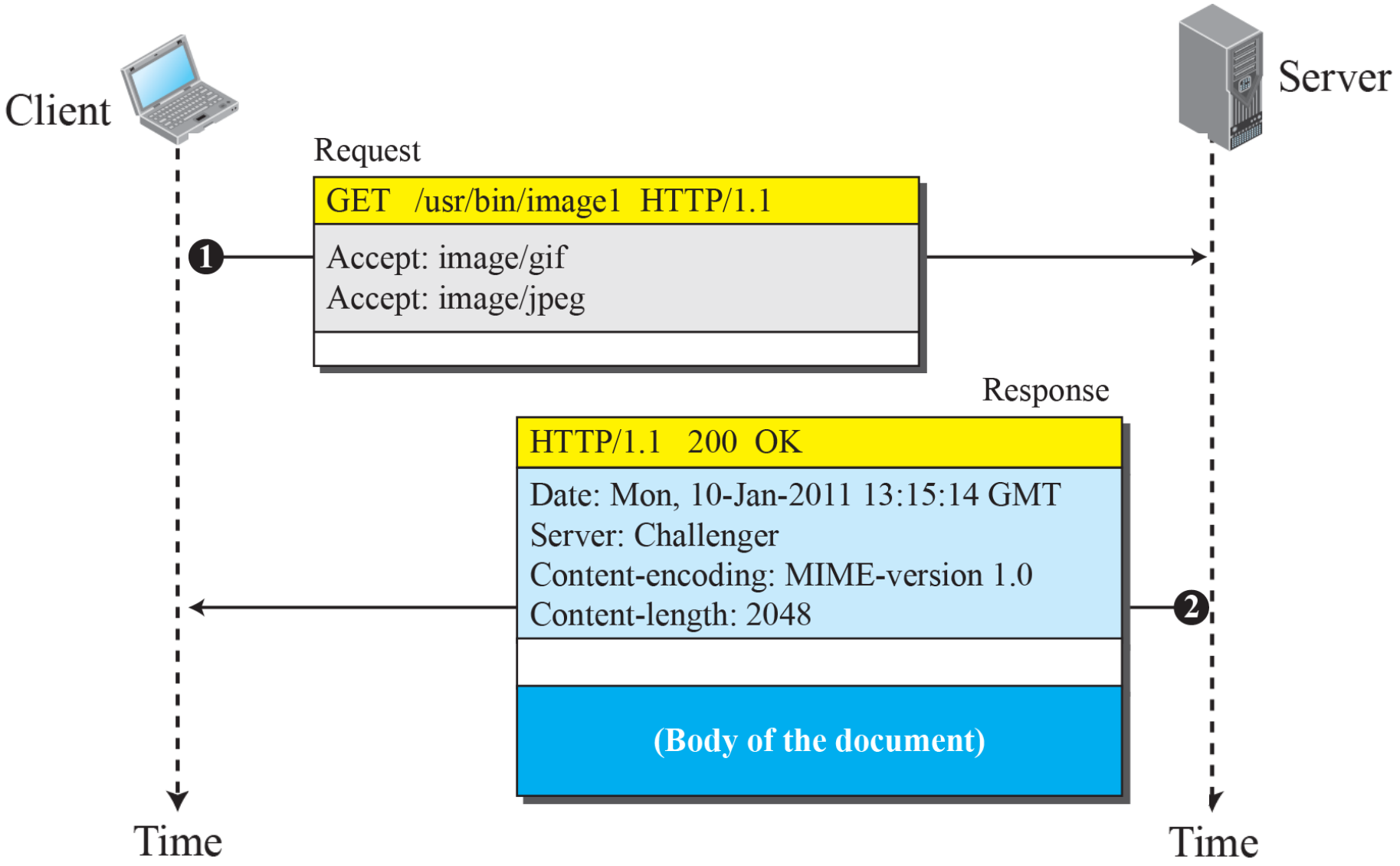
Header	*Description*
Date	Shows the current date
Upgrade	Specifies the preferred communication protocol
Server	Gives information about the server
Set-Cookie	The server asks the client to save a cookie
Content-Encoding	Specifies the encoding scheme
Content-Language	Specifies the language
Content-Length	Shows the length of the document
Content-Type	Specifies the media type
Location	To ask the client to send the request to another site
Accept-Ranges	The server will accept the requested byte-ranges
Last-modified	Gives the date and time of the last change

Example 2.6

This example retrieves a document (see Figure 2.13).

- We use the **GET** method to retrieve an image with the path **/usr/bin/image1**.
- The request line shows the method (GET), the URL, and the HTTP version (1.1).
- The header has two lines that show that the client can accept images in the GIF or JPEG format.
- The request does not have a body.
- The response message contains the status line and four lines of header.
- The header lines define the date, server, content encoding (MIME version, which will be described in electronic mail), and length of the document. The body of the document follows the header..

Figure 2.13: Example 2.6

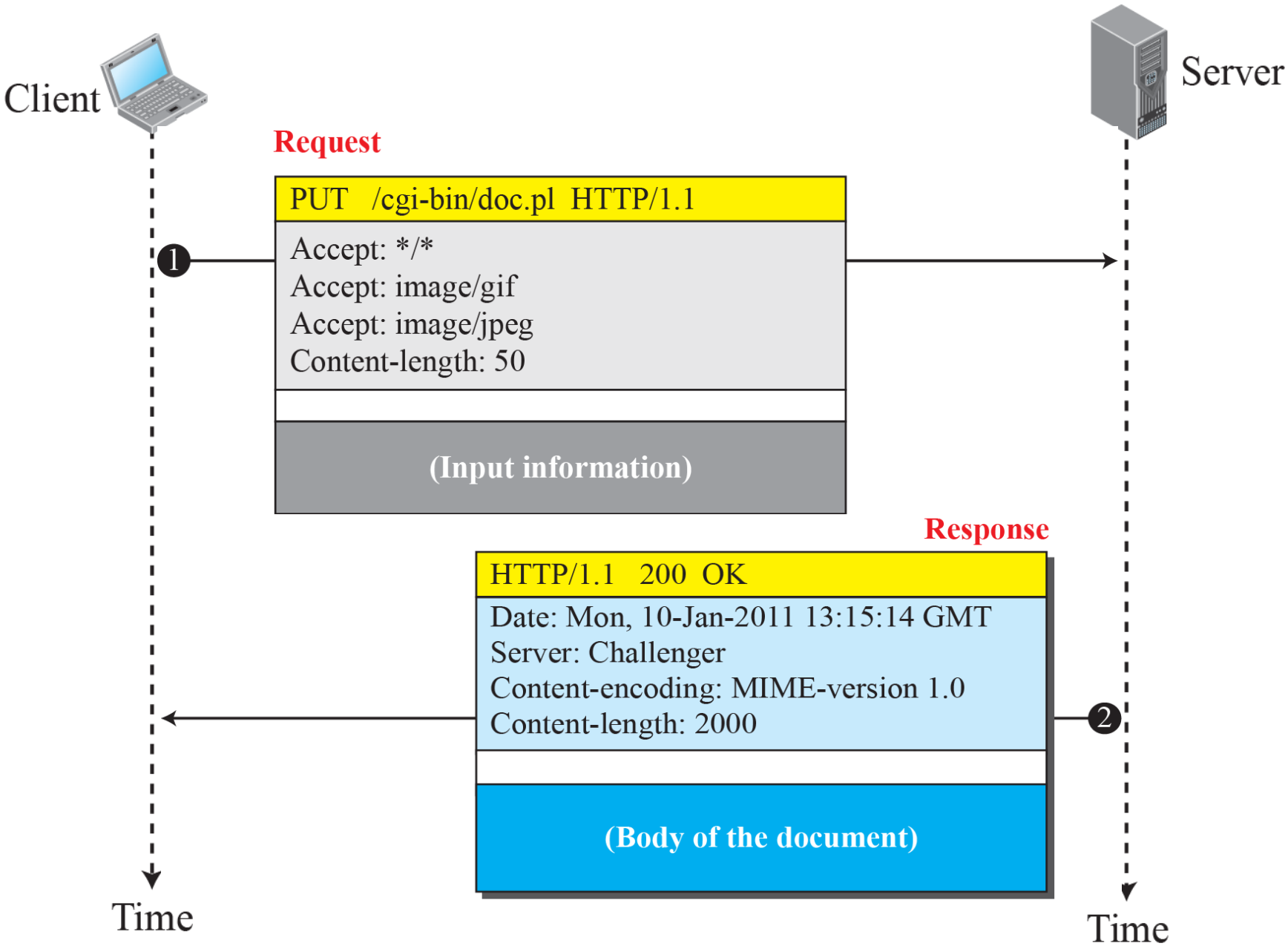


Example 2.7

In this example, the client wants to send a web page to be **posted** on the server.

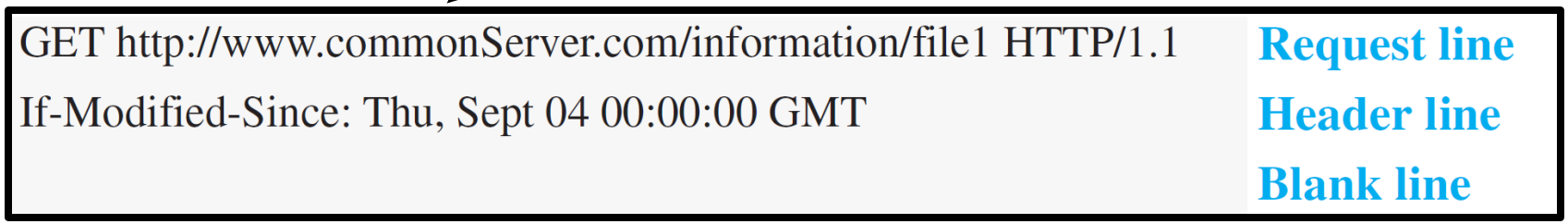
- We use the **PUT** method.
- The request line shows the method (PUT), URL, and HTTP version (1.1).
- There are four lines of headers.
- The request body contains the web page to be posted.
- The response message contains the status line and four lines of headers.
- The created document, which is a CGI document, is included as the body (see Figure 2.14).

Figure 2.14: Example 2.7

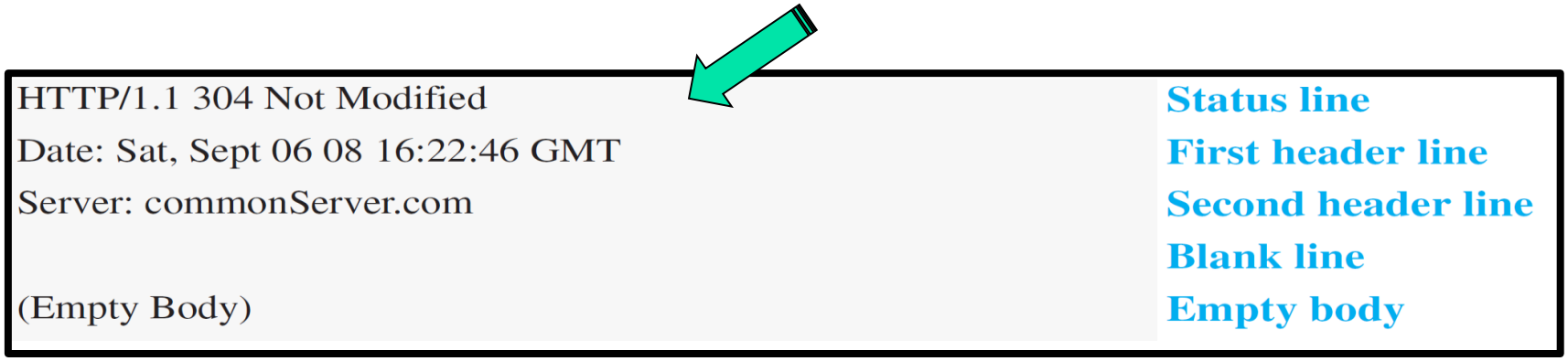


Example 2.8

The following shows how a client **imposes/forces** the modification data and time **condition** on a request.



The status line in the response shows the file **was not modified after the defined point in time**. The body of the response message is also empty.

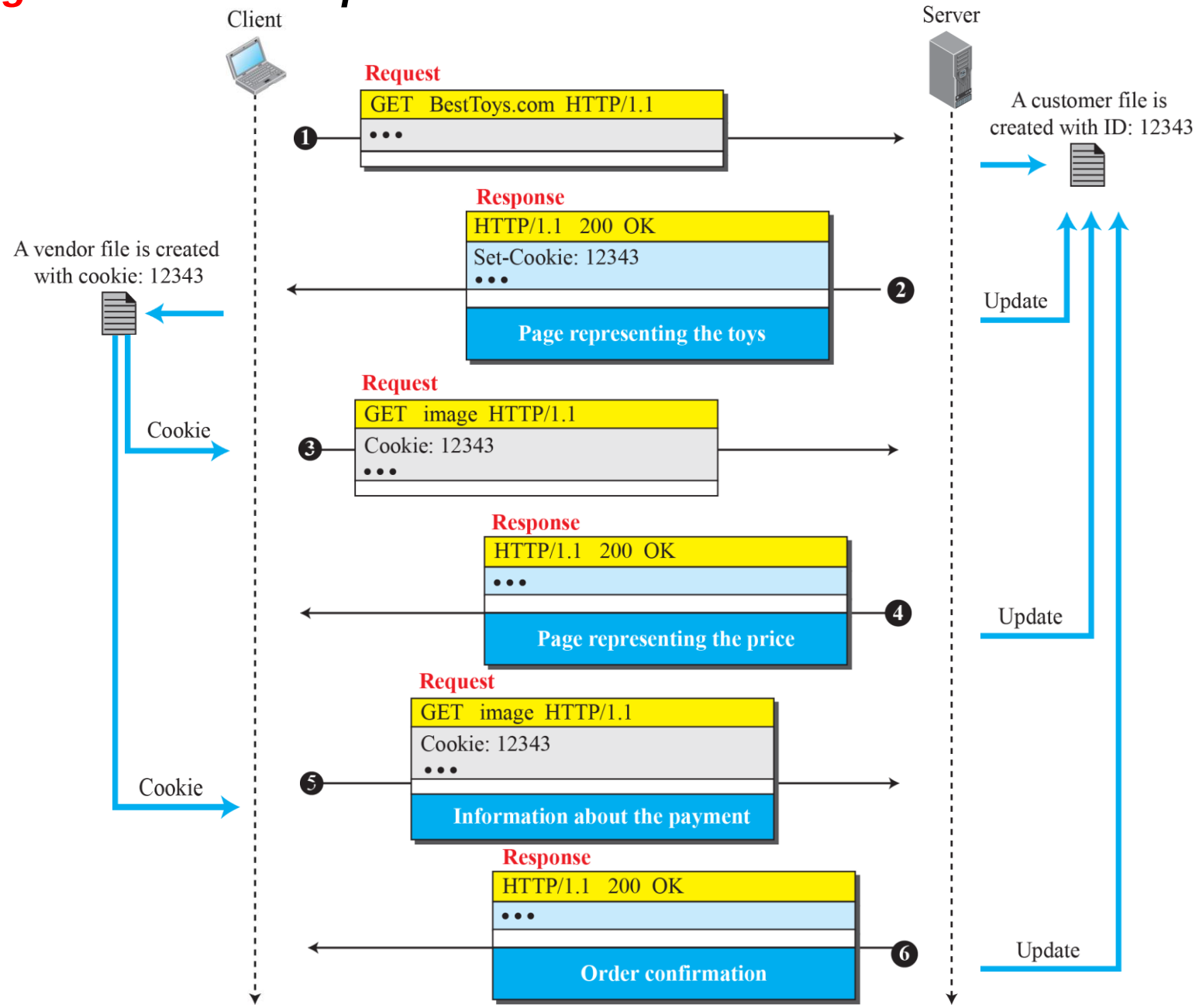


Example 2.9

Figure 2.15 shows a scenario in which an **electronic store** can **benefit** from the use of **cookies**.

- Assume a **shopper** wants to **buy a toy** from an electronic store named BestToys.
- The **shopper browser (client)** sends a request to the BestToys **server**.
- The server creates an **empty shopping cart** (a list) for the client and **assigns an ID** to the cart (for example, **12343**).
- The server then sends a response message, which contains the images of all toys available, with a link under each toy that selects the toy **if it is being clicked**.
- This response message also includes the **Set-Cookie header line** whose value is 12343.
- The client displays the images and **stores the cookie value** in a file named BestToys.

Figure 2.15: Example 2.9

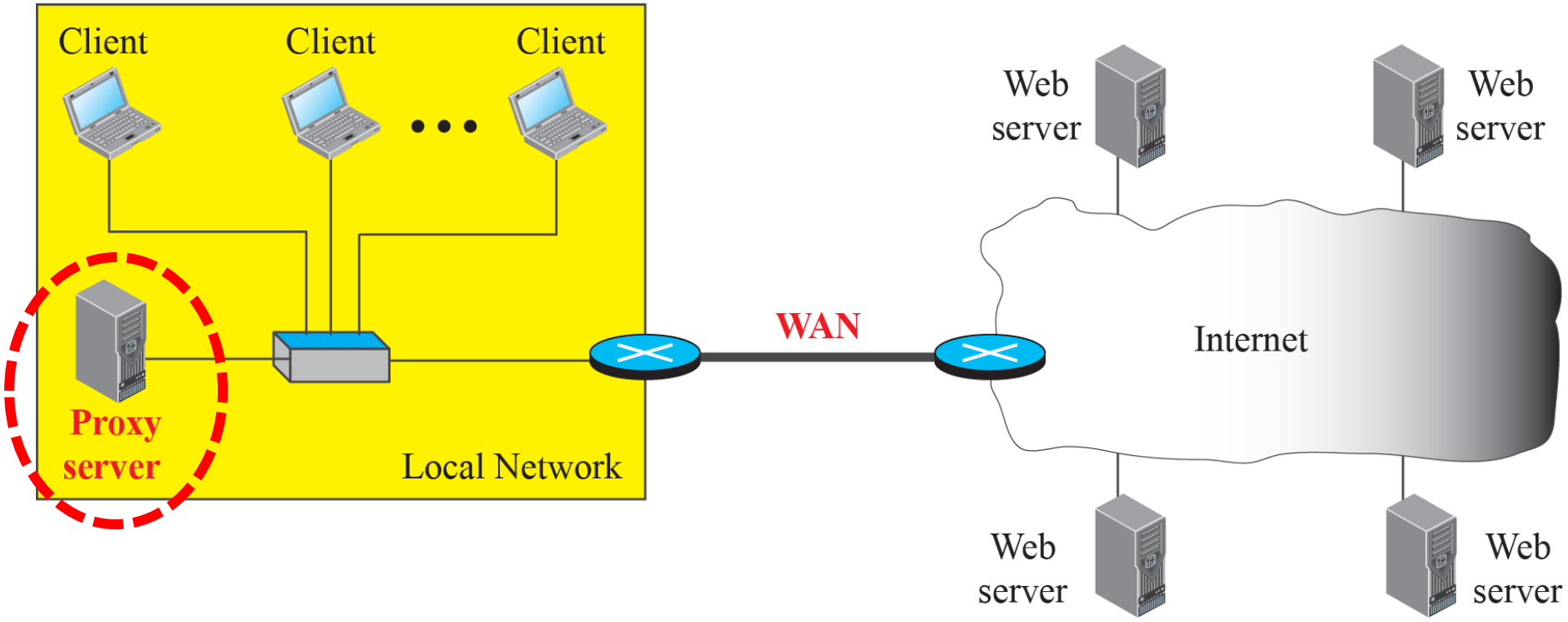


Example 2.10

Figure 2.16 shows an example of a use of a proxy server in a local network, such as the network on a **campus** or in a **company**.

- The proxy server is installed in the local network.
- When an HTTP request is created by any of the clients (browsers), the request is first directed to the proxy server.
- If the proxy server already has the corresponding web page, it sends the response to the client.
- Otherwise, the **proxy server acts as a client** and **sends the request to the web server in the Internet**.
- When the response is returned, the proxy server **makes a copy** and stores it in its **cache before sending it to the requesting client**.

Figure 2.16: Example of a proxy server



2.3.4 TELNET

- *A server program can provide a specific service to its corresponding client program. However, it is impossible to have a client/server pair for each type of service we need.*
- *Another solution is to have a specific client/server program for a set of common scenarios, but to have some generic client/server programs that allow a user on the client site to log into the computer at the server site and use the services available there.*
- *We refer to these generic client/server pairs as remote logging applications. One of the original remote logging protocols is TELNET.*
- *However, because of serious security concerns when using Telnet over an open network such as the Internet, its use for this purpose has been waned against & significantly in favour of SSH2.*

2.3.5 Secure Shell (SSH)

- *Secure Shell (SSH) is a cryptographic network protocol for operating network services securely over an unsecured network.*
- *SSH provides a secure channel over an unsecured network in a client-server architecture, connecting an SSH client application with an SSH server.*
- *The most visible application of the protocol is for access to shell accounts on Unix, Linux, windows and other O.S*
- *SSH was designed as a replacement for **TELNET** and for unsecured remote shell protocols such as the Berkeley rlogin, rsh, and rexec protocols.*
- *There are two versions of SSH: SSH-1 and SSH-2, which are totally incompatible. The first version, SSH-1, is now deprecated because of security flaws in it. In this section, we discuss only SSH-2.*

2.3.5 (continued)

- **Components**
  - SSH Transport-Layer Protocol (SSH-TRANS)
  - SSH Authentication Protocol (SSH-AUTH)
  - SSH Connection Protocol (SSH-CONN)
- **Applications**
  - SSH for Remote Logging
  - SSH for File Transfer
- **Port Forwarding**
- **Format of the SSH Packets**

Figure 2.32: Components of SSH

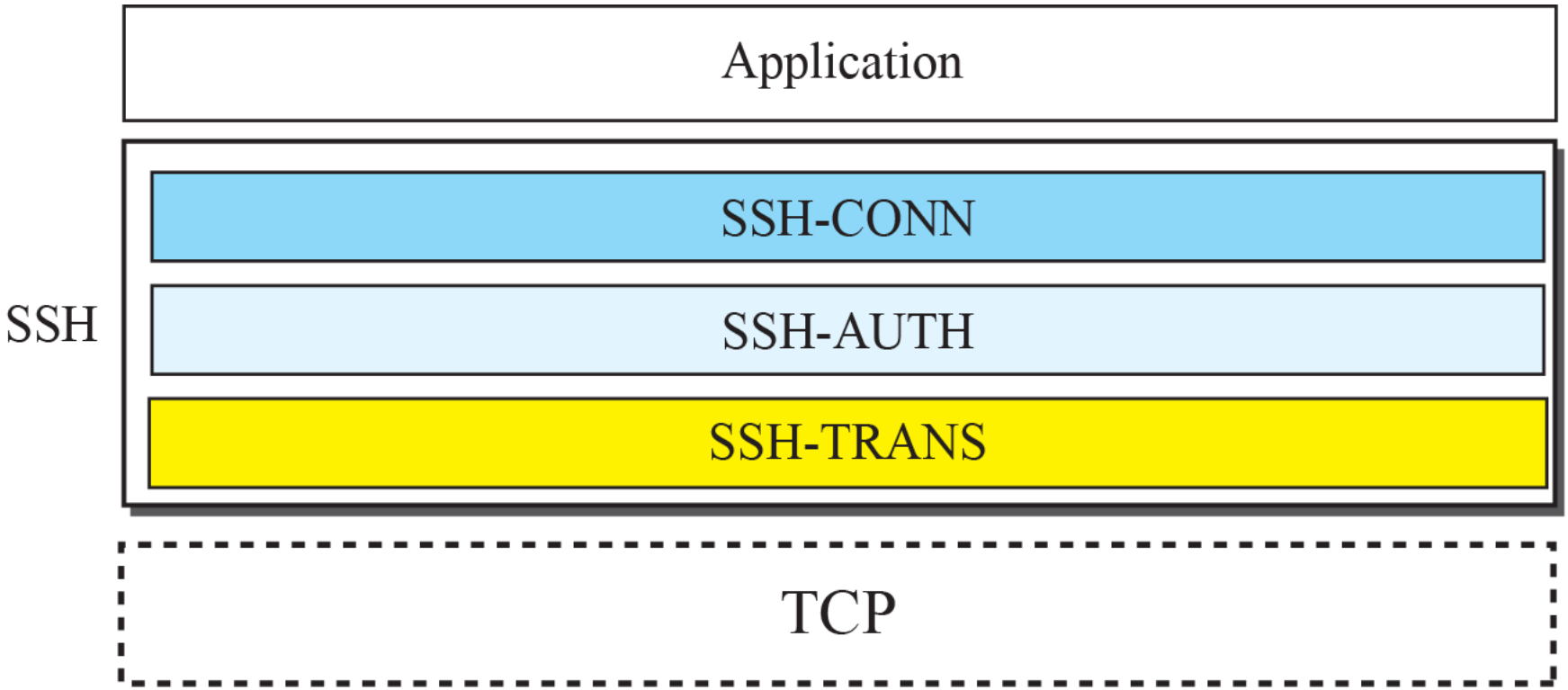


Figure 2.33: Port Forwarding

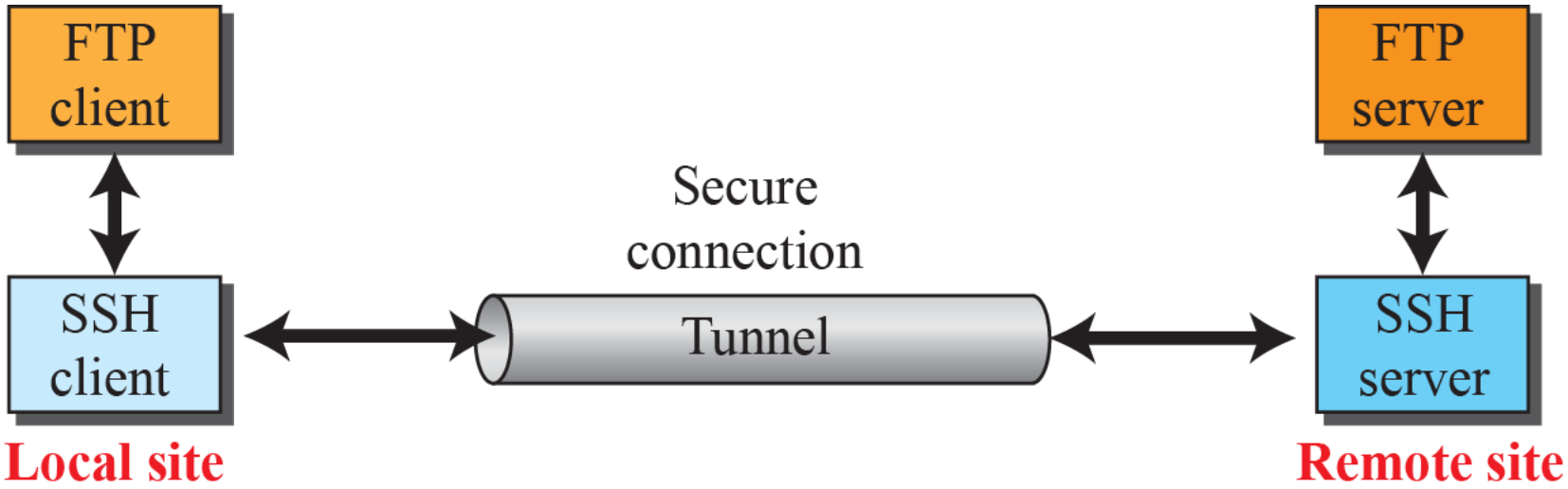
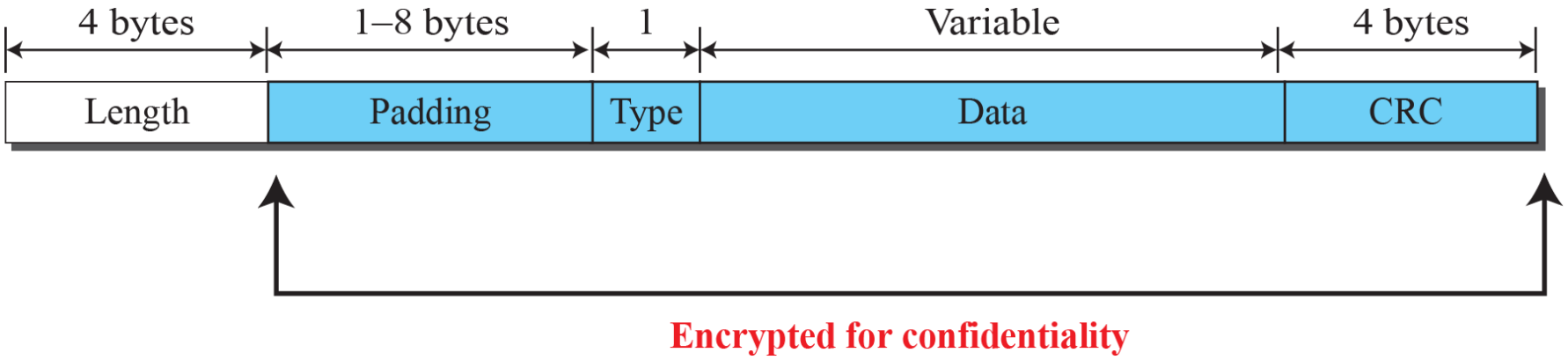


Figure 2.34: SSH Packet Format



2.3.6 Domain Name System (DNS)

- *To identify an entity, TCP/IP protocols use the IP address, which uniquely identifies the connection of a host to the Internet.*
- *However, people prefer to use **names** instead of **numeric addresses**.*
- *Therefore, the Internet needs to have a **directory system** that can **map** a **name** to an **address**.*
- *This is analogous to the telephone network.*
- *A telephone network is designed to use telephone numbers, not names. People can either keep a private file to map a name to the corresponding telephone number or can call the telephone directory to do so.*

2.3.6 (continued)

- ***Name Space***
  - ***Domain Name Space***
  - ***Domain***
  - ***Distribution of Name Space***
  - ***Zone***
  - ***Root Server***
- ***DNS in the Internet***
  - ***Generic Domains***
  - ***Country Domains***

2.3.1 (continued)

- *Resolution*
  - *Recursive Resolution*
  - *Iterative Resolution*
  - *Caching*
- *Resource Records*
- *DNS Messages*
- *Encapsulation*
- *Registrars*
- *DDNS*
- *Security of DNS*

Figure 2.35: Purpose of DNS Host-name resolution

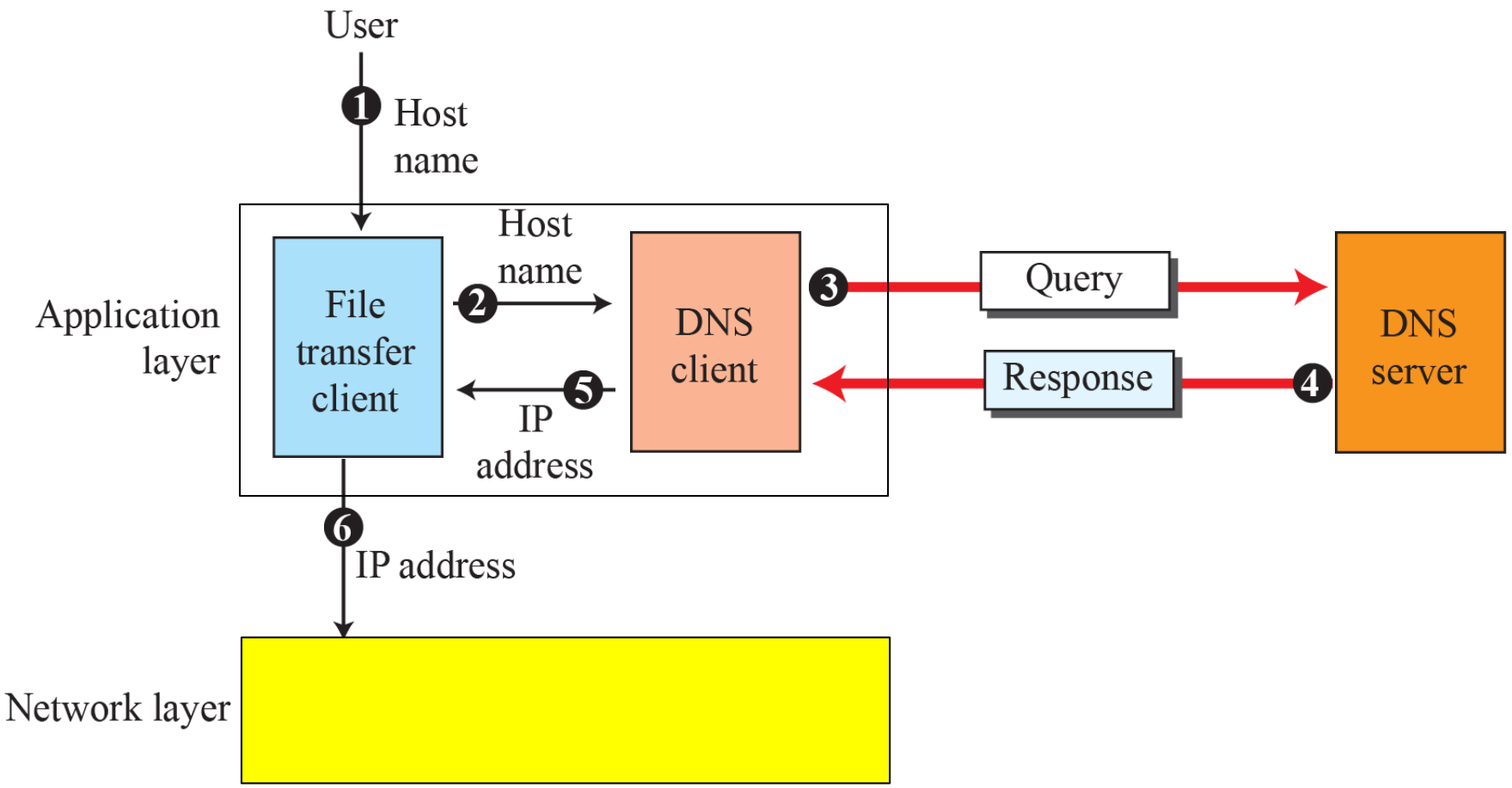


Figure 2.36: Domain name space

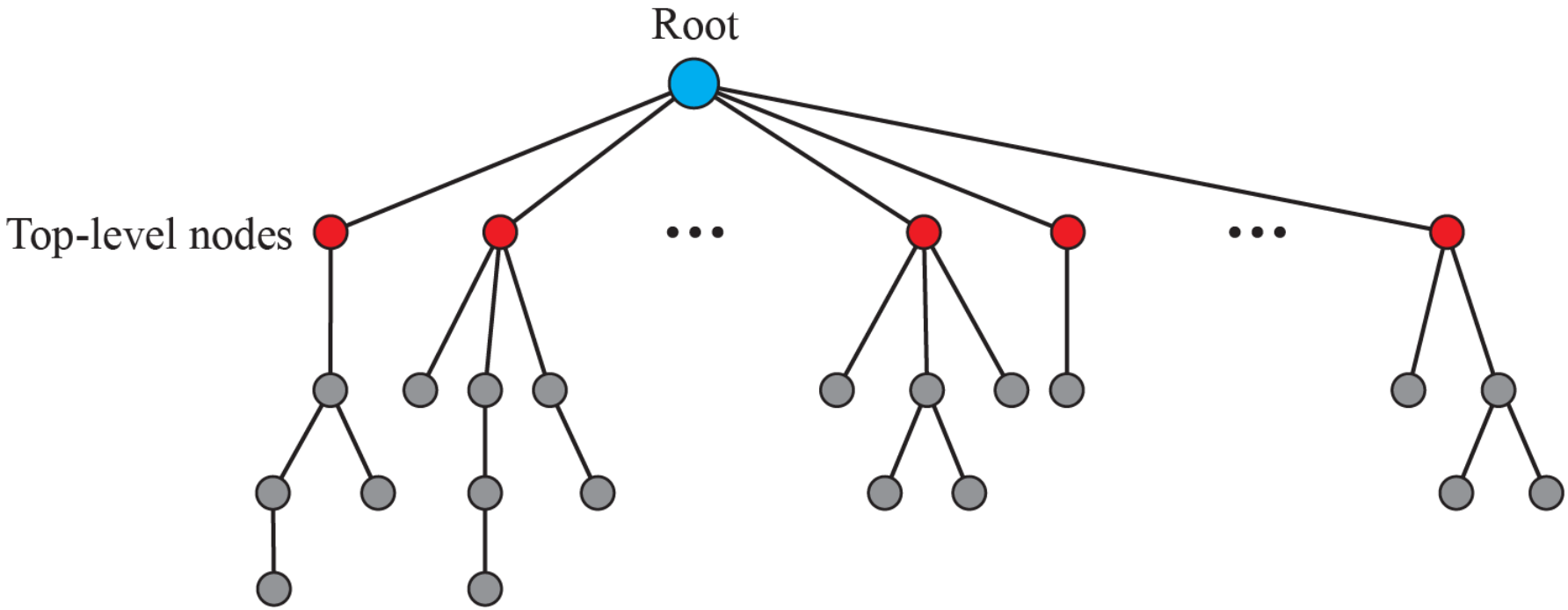


Figure 2.37: Domain names and labels

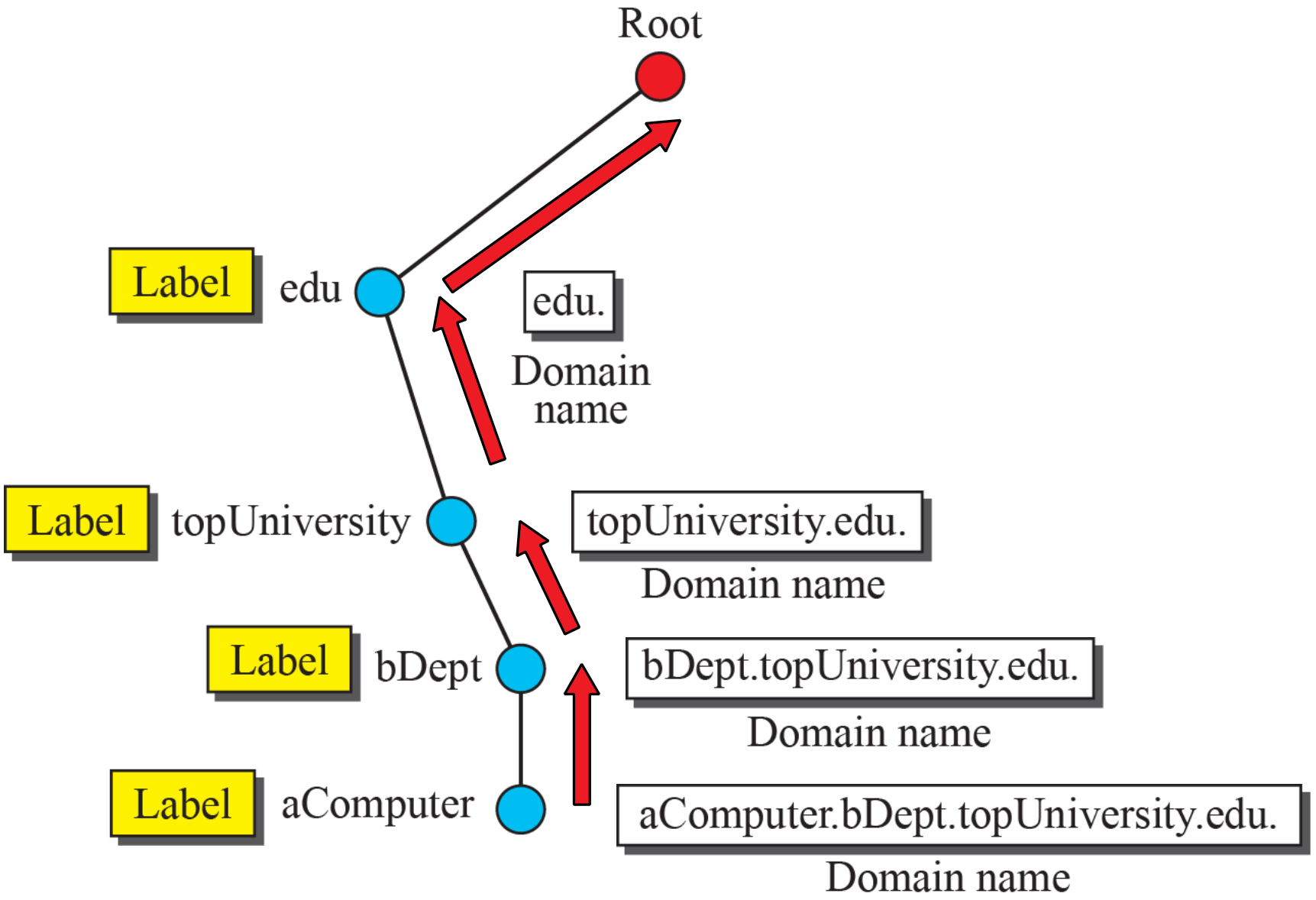


Figure 2.38: Domains

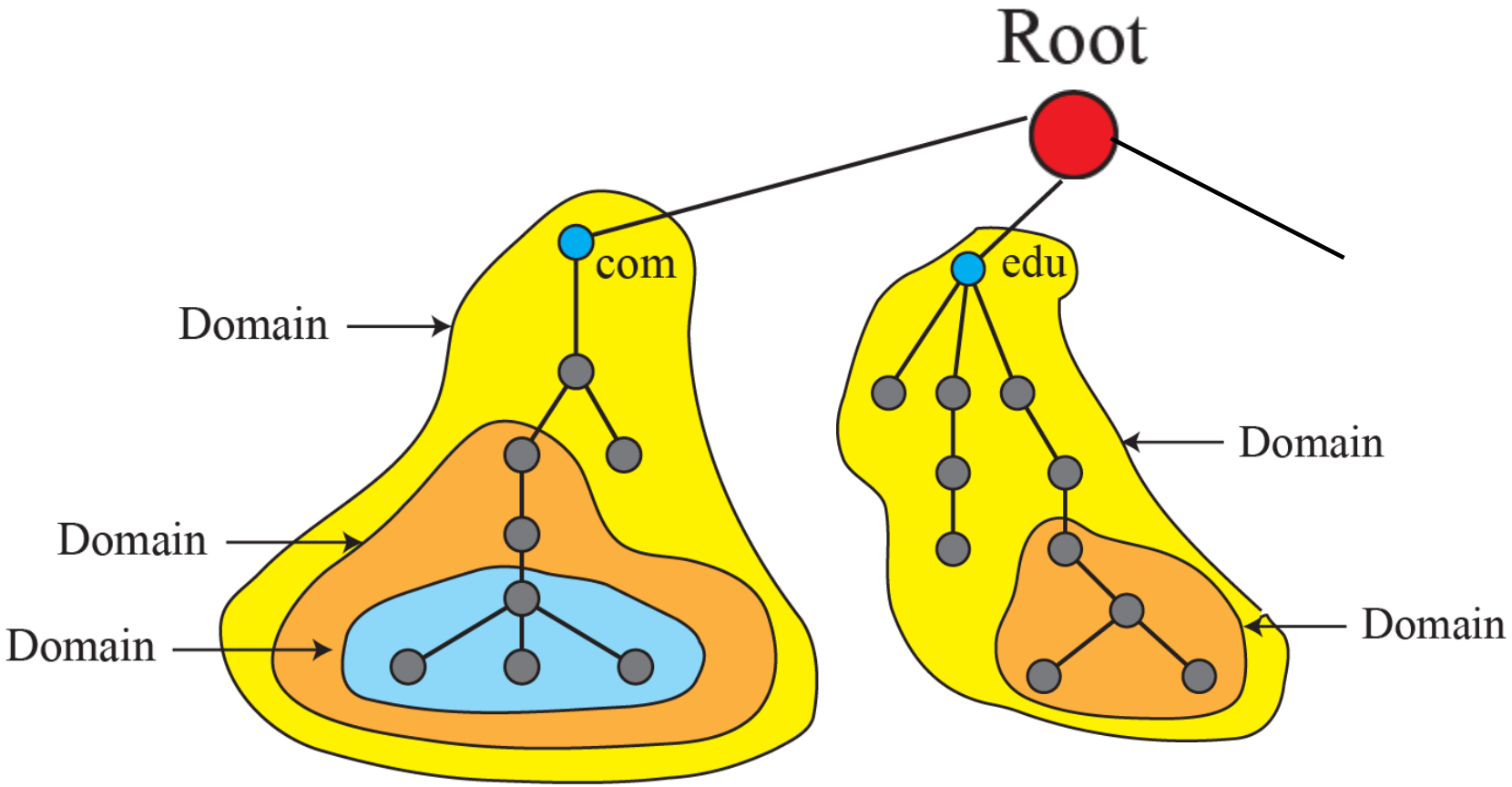


Figure 2.39: Hierarchy of name servers

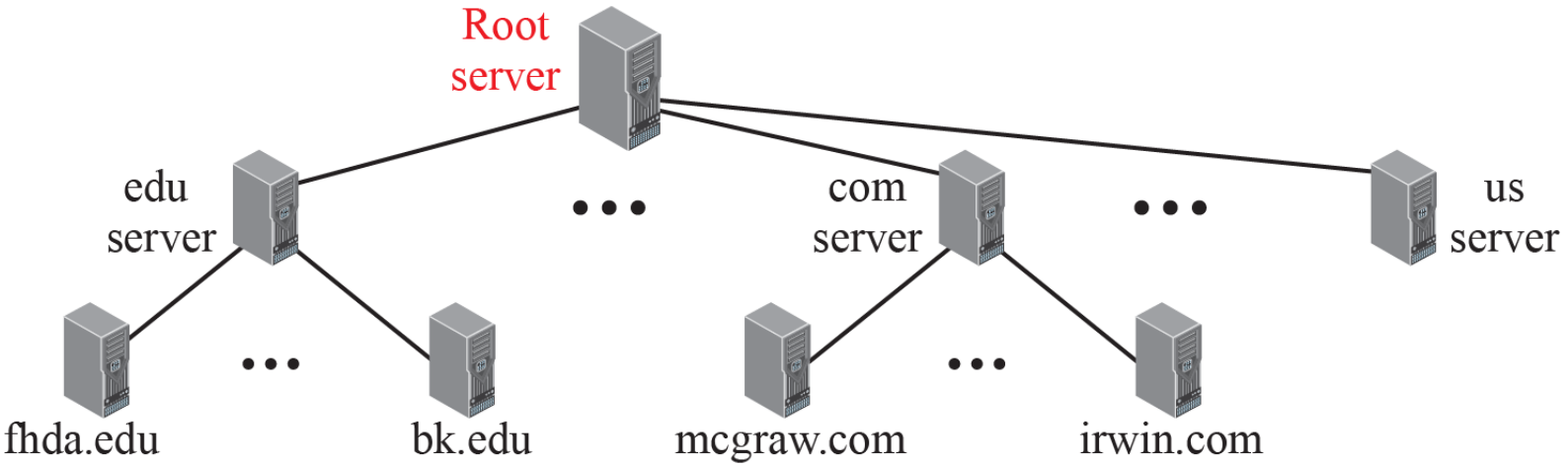


Figure 2.40: Zone

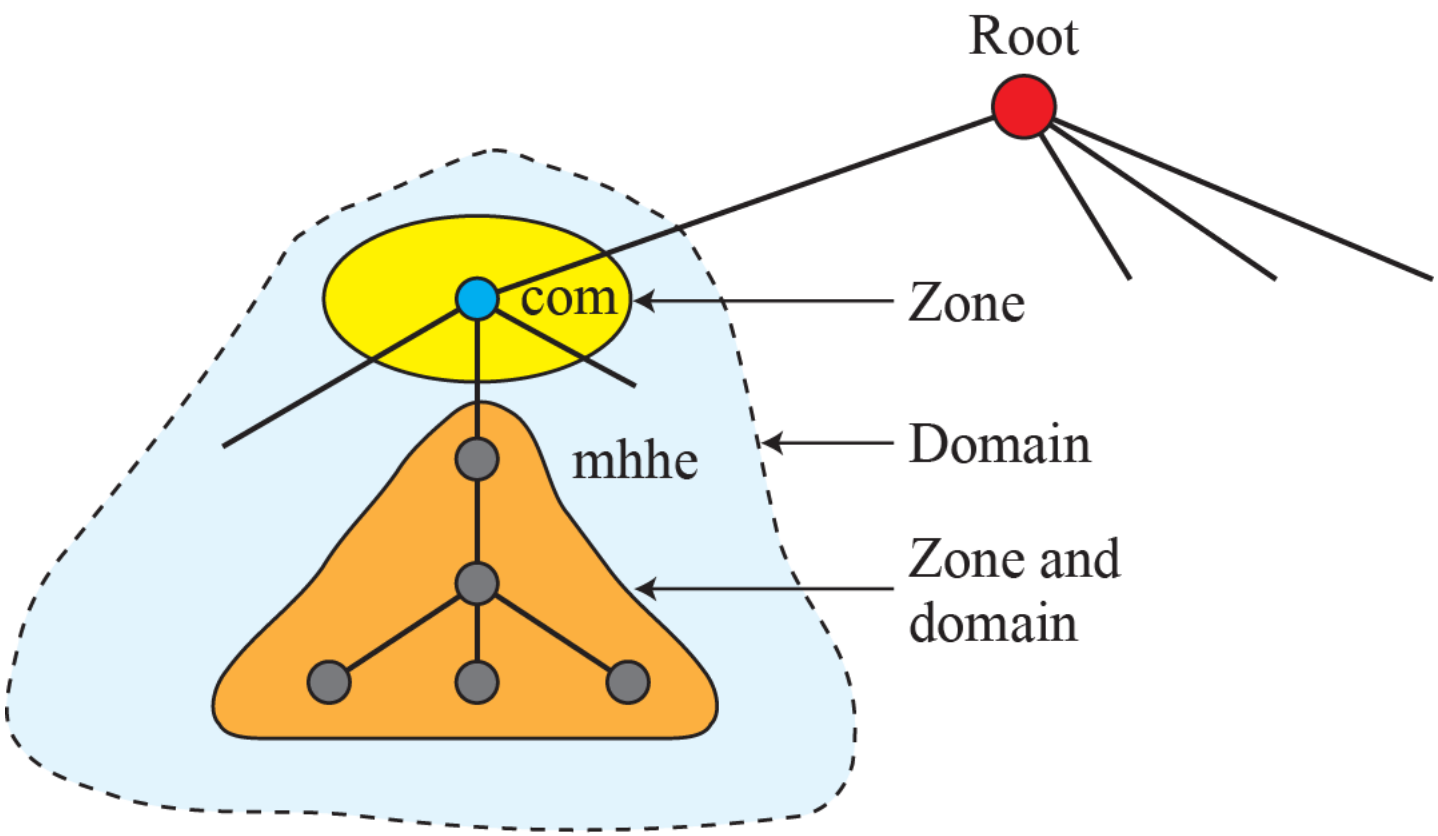


Figure 2.41: Generic domains

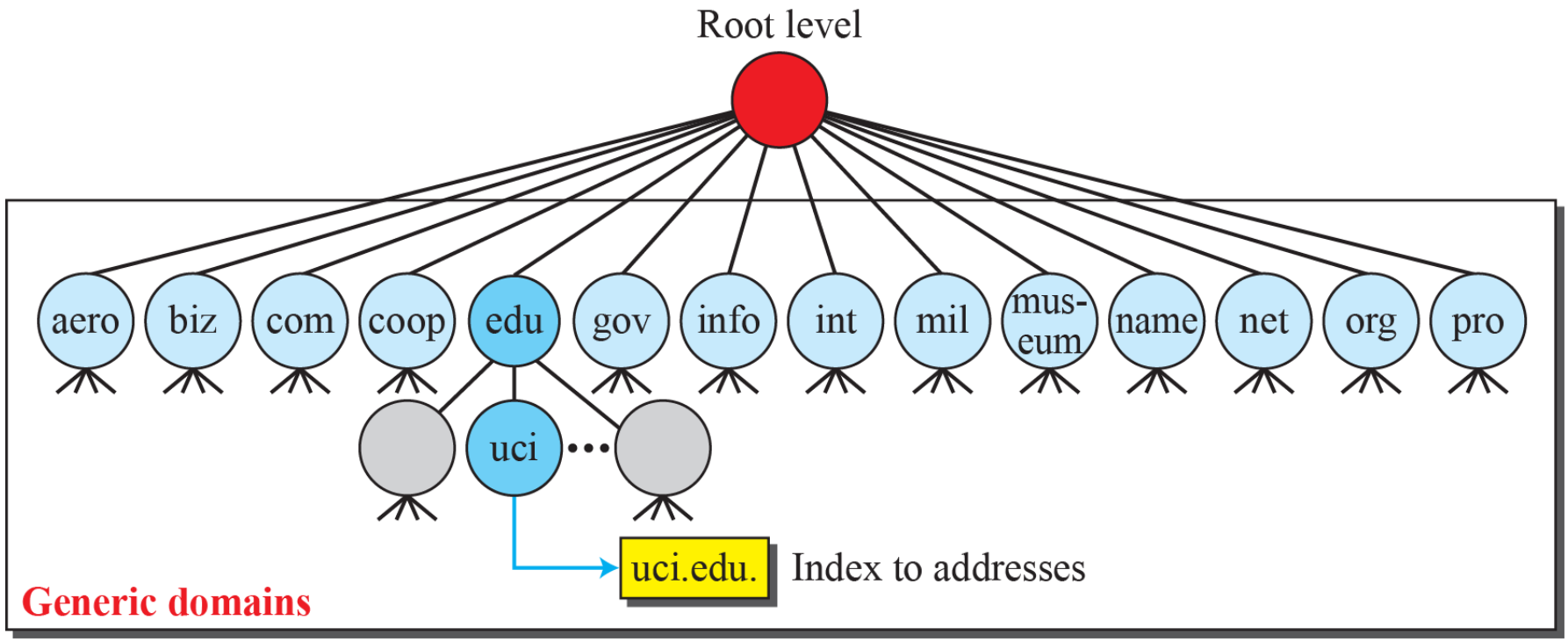


Table 2.12: Generic domain labels

Label	*Description*	*Label*	*Description*
aero	Airlines and aerospace	**int**	International organizations
biz	Businesses or firms	**mil**	Military groups
com	Commercial organizations	**museum**	Museums
coop	Cooperative organizations	**name**	Personal names (individuals)
edu	Educational institutions	**net**	Network support centers
gov	Government institutions	**org**	Nonprofit organizations
info	Information service providers	**pro**	Professional organizations

Figure 2.42: Country domains

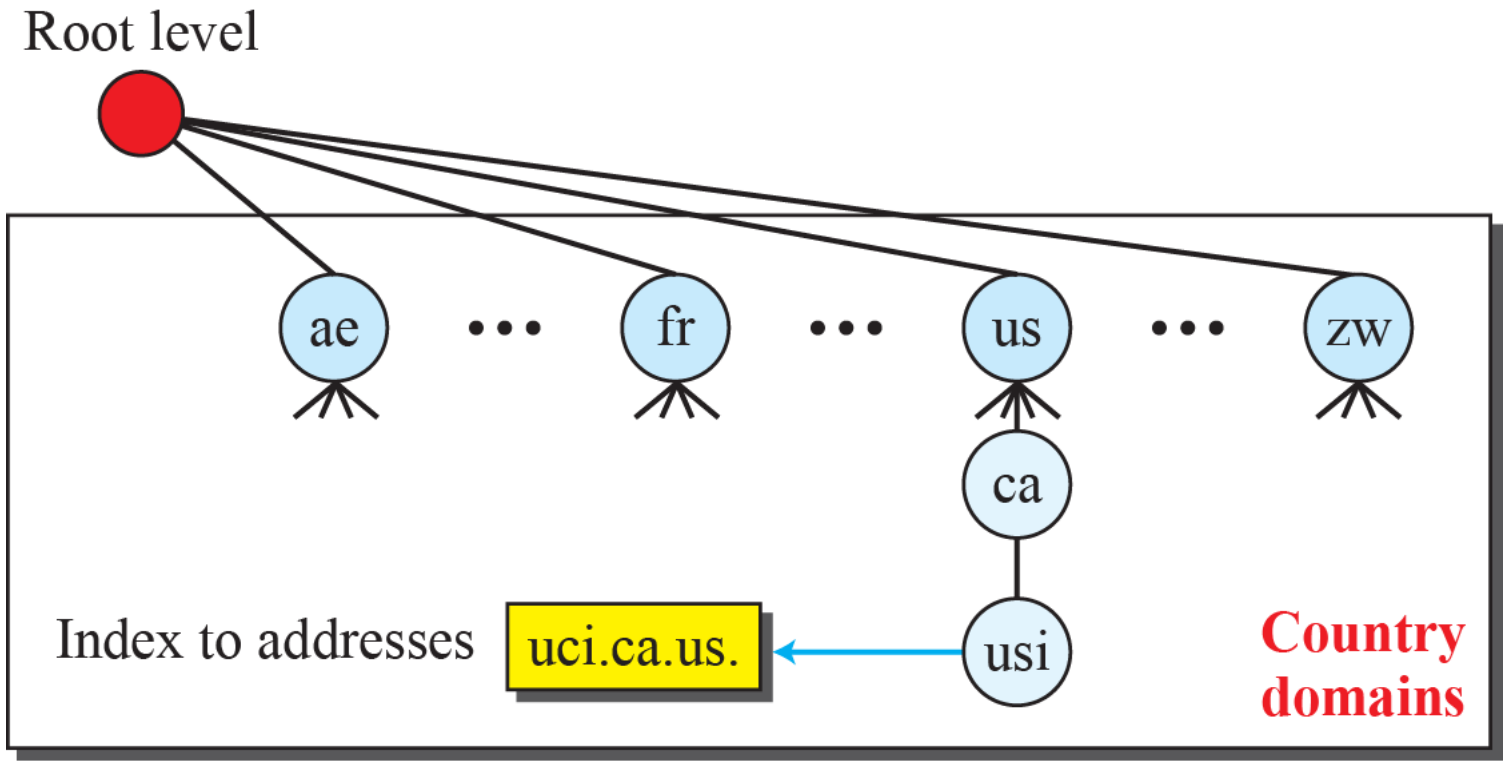


Figure 2.43: Recursive Host-name resolution

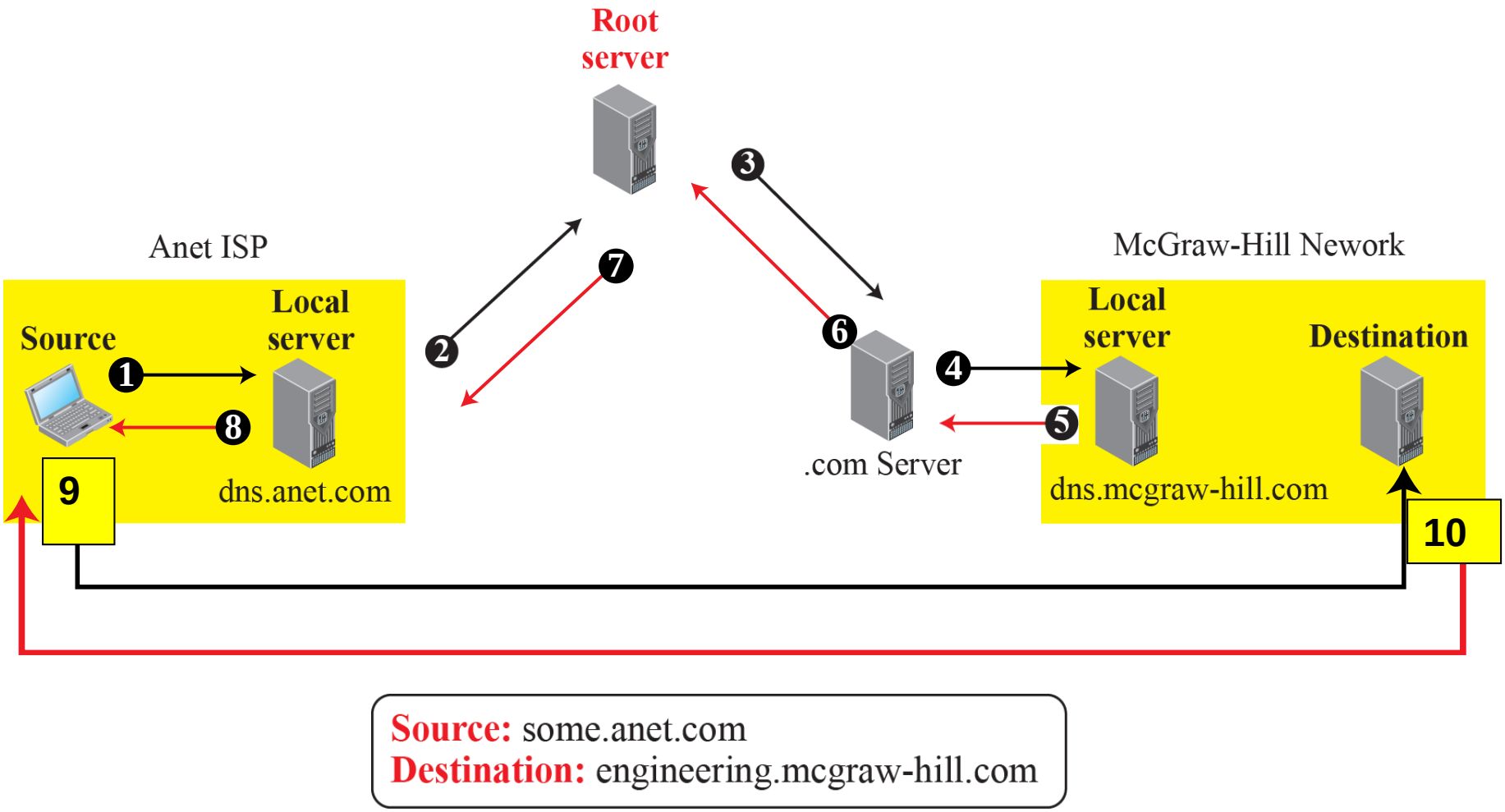


Figure 2.44: Iterative Host-name resolution

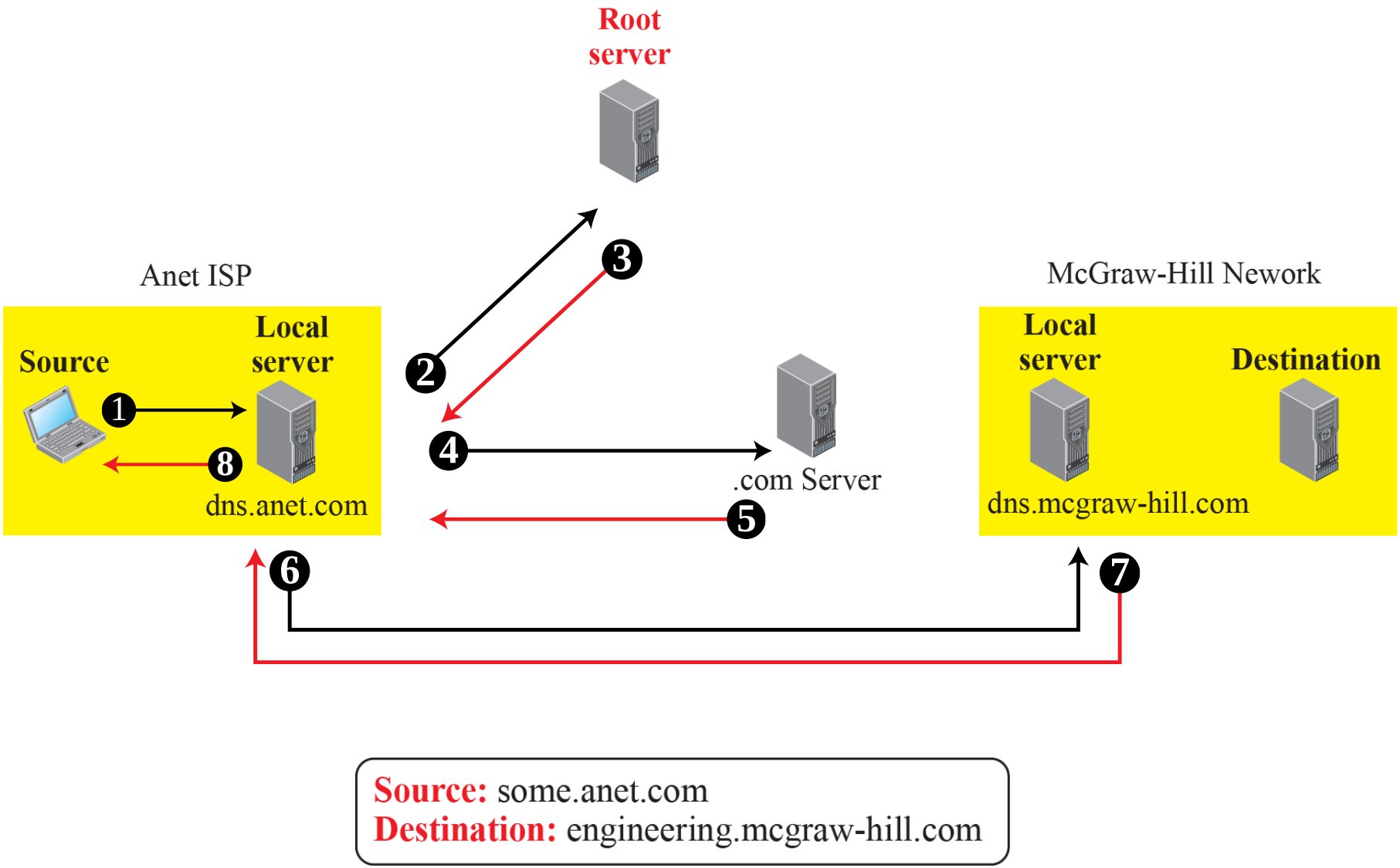
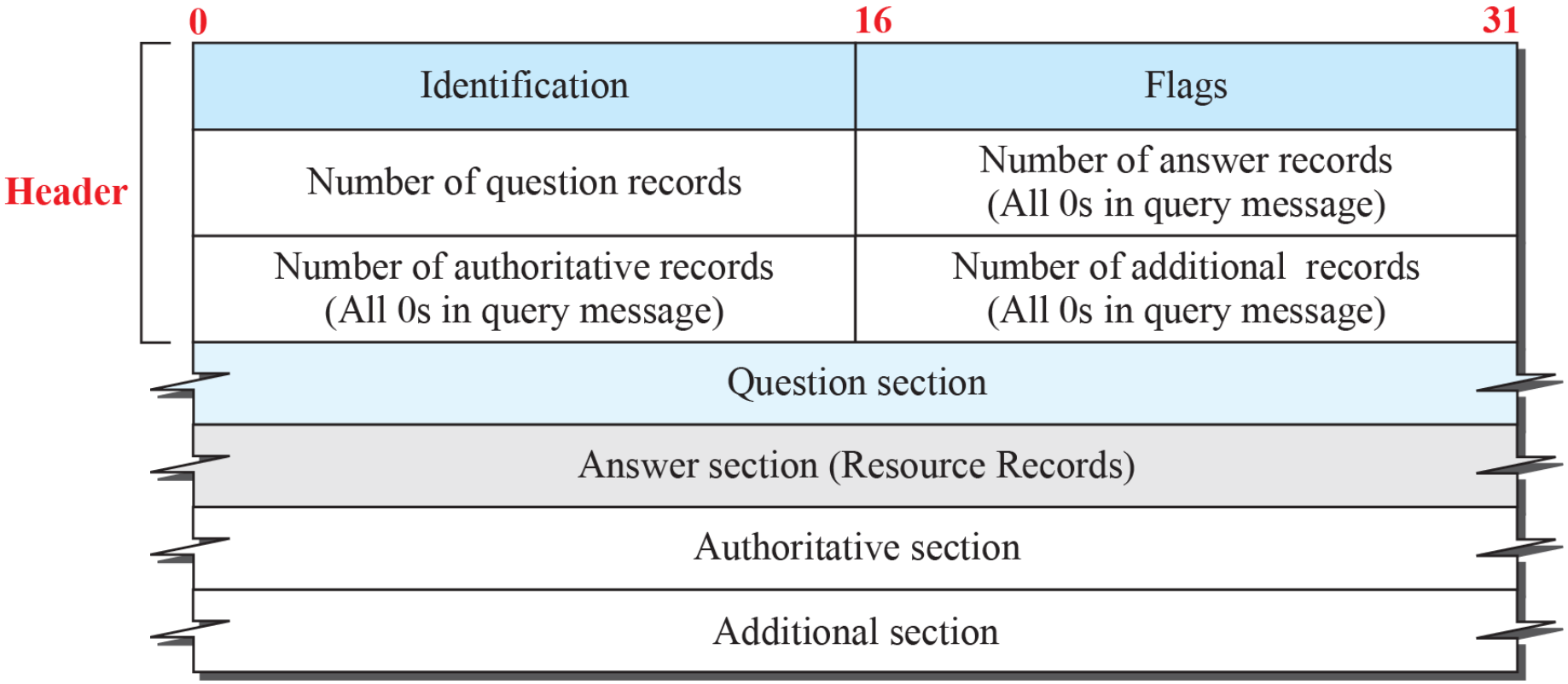


Table 2.13: DNS types

Type	*Interpretation of value*
A	A 32-bit IPv4 address (see Chapter 4)
NS	Identifies the authoritative servers for a zone
CNAME	Defines an alias for the official name of a host
SOA	Marks the beginning of a zone
MX	Redirects mail to a mail server
AAAA	An IPv6 address (see Chapter 4)

Figure 2.45: DNS message



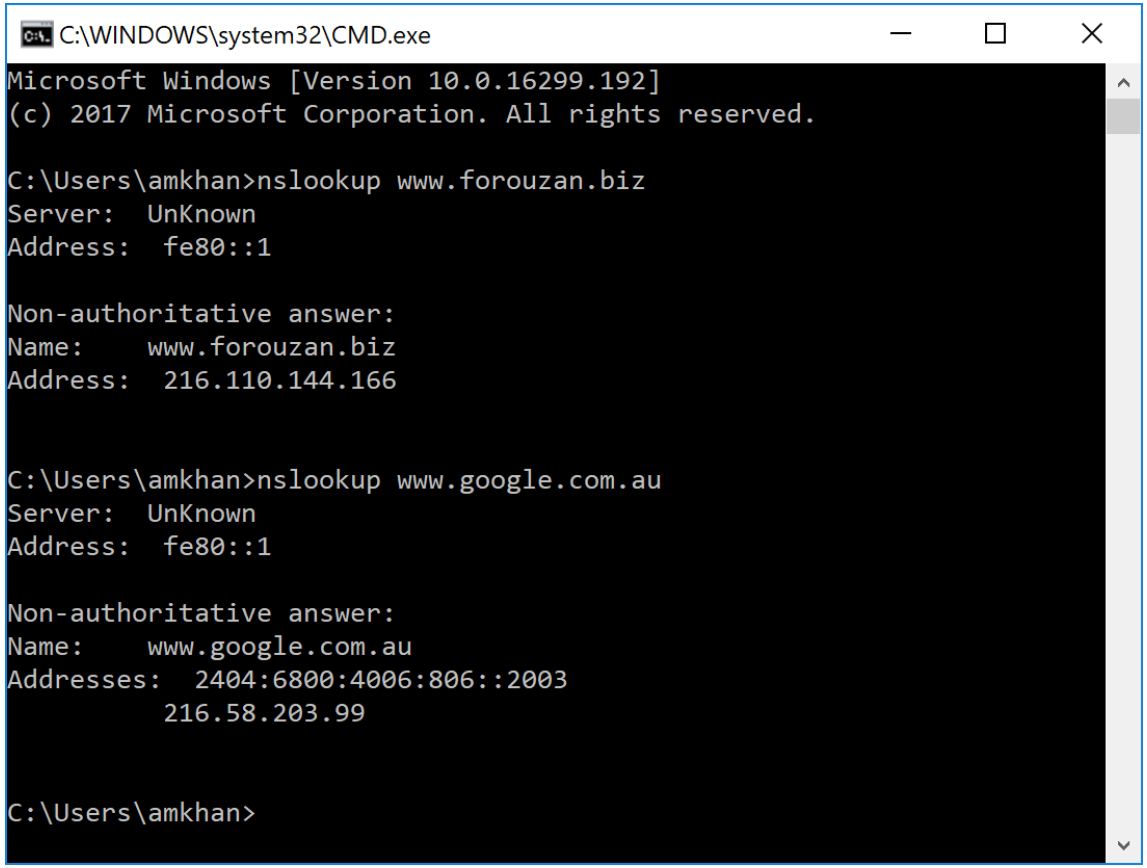
Note:

The query message contains only the question section. The response message includes the question section, the answer section, and possibly two other sections.

Example 2.14

In UNIX and Windows, the **nslookup** utility can be used to retrieve address/name mapping. The following shows how we can retrieve an address when the domain name is given.

```
$nslookup www.forouzan.biz
Name:   www.forouzan.biz
Address: 198.170.240.179
```



2-5 SOCKET-INTERFACE PROGRAMMING

*In this part of lecture, we will see some examples how to write some **simple client-server programs** using **C**, a **procedural programming** language.*

- *Here in this lecture we have chose the **C language**;*
- *In the last lecture (chapter 11), we expand this idea in **Java**, which provides a more compact version.*

2.5.1 Socket Interface in C

- *First we shall see how this **interface** is implemented in the **C language**.*
- *Then understand **issue** of **socket interface** and the **role of a socket** in communication.*
- *The socket has **no buffer** to **store** data to be **sent or received.** It is capable of neither sending nor receiving data.*
- *The socket just acts as a reference or a label. The **buffers and necessary variables are created** inside the operating system.*

2.5.1 (continued)

- *Data Structure for Socket*
- *Header Files*
- *Communication Using UDP*
  - *Sockets Used for UDP*
  - *Communication Flow Diagram*
  - *Programming Examples*
- *Communication Using TCP*
  - *Sockets Used in TCP*
  - *Communication Flow Diagram*

Figure 2.58: Socket data structure

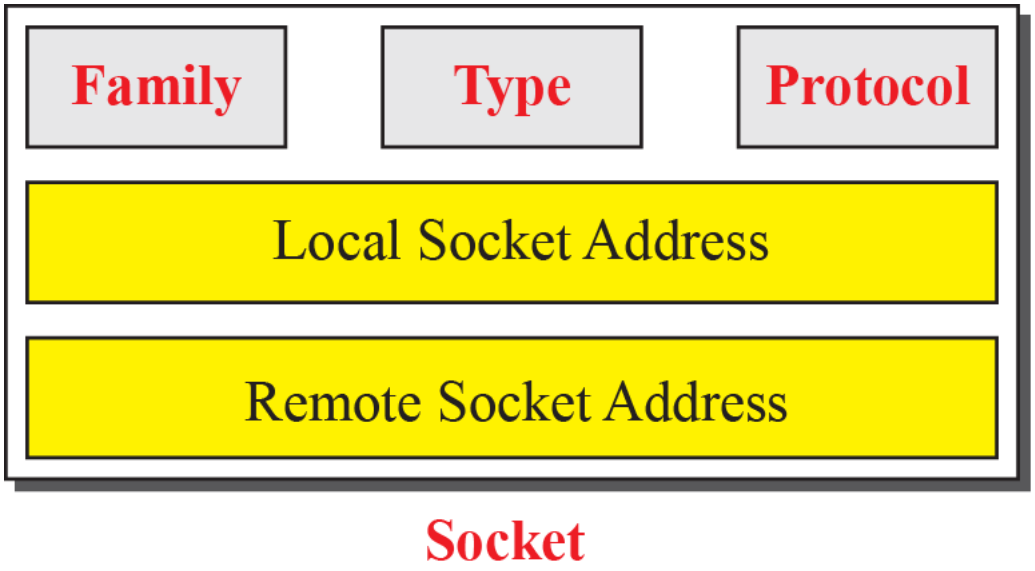


Figure 2.59: Sockets for UDP communication

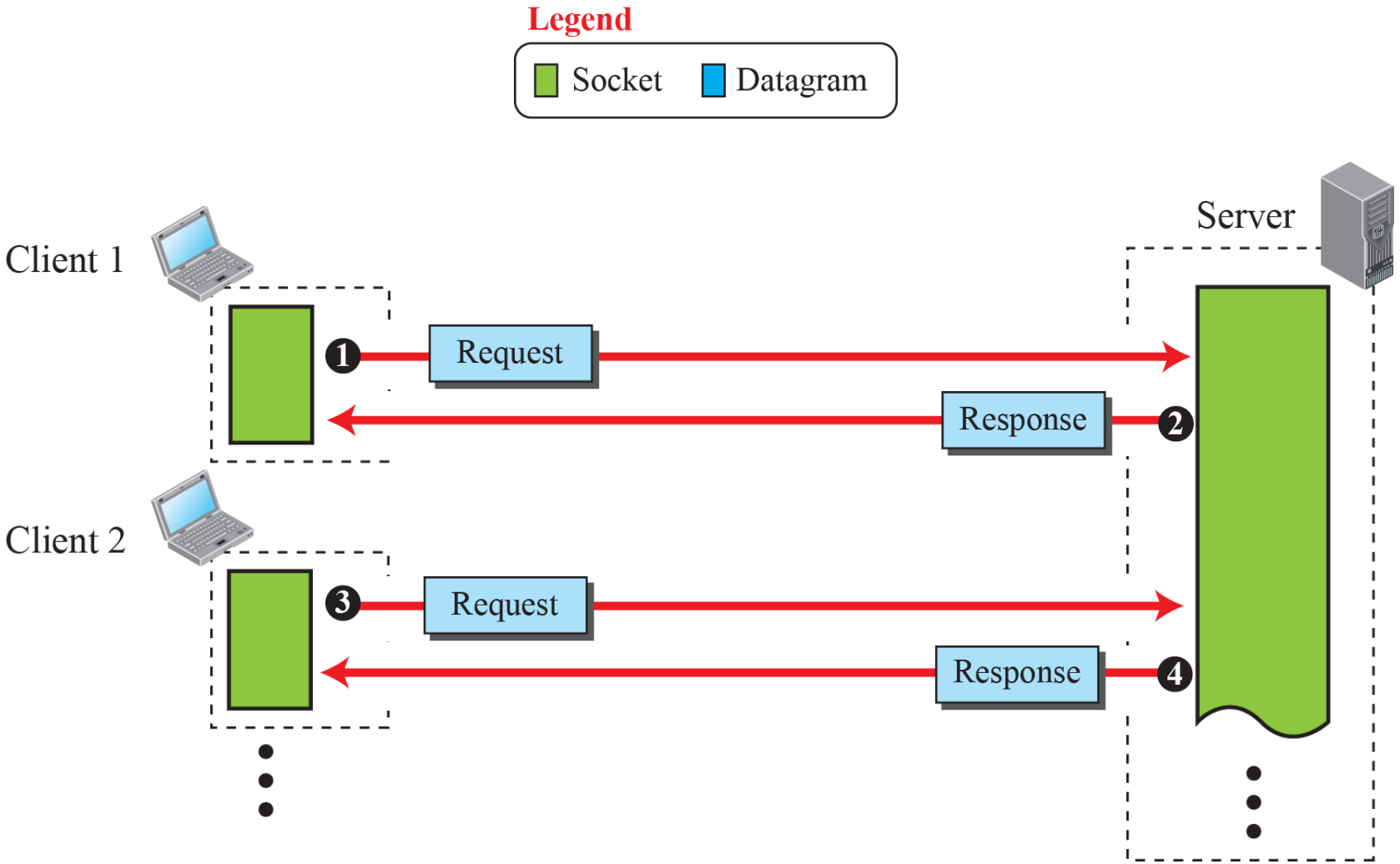
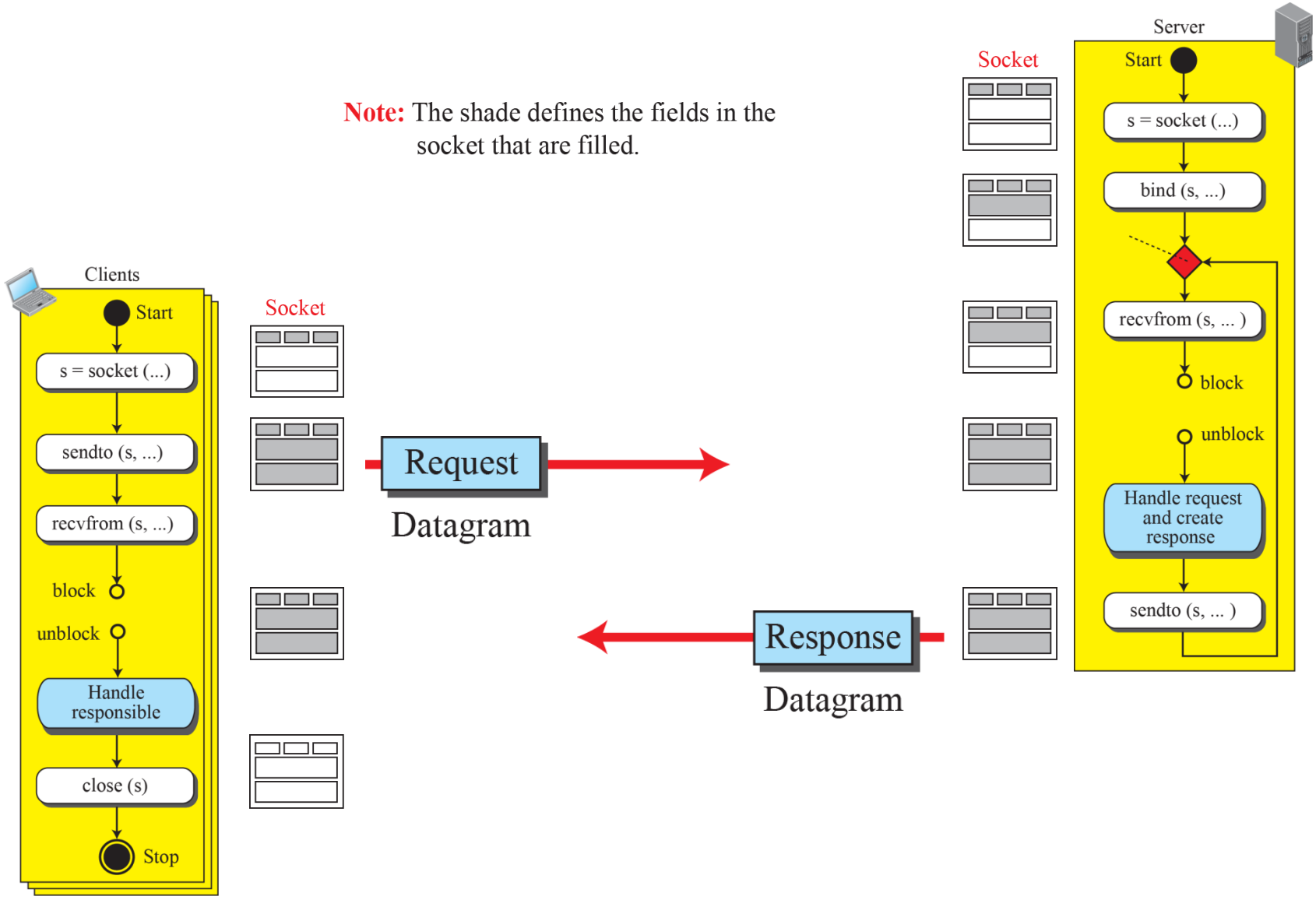


Figure 2.60: Flow diagram for iterative UDP communication



UDP ECHO CLIENT SERVER

AIM: To write a program for UDP echo client server.

SERVER ALGORITHM:

STEP 1: Start
STEP 2: Declare the variables for the socket
STEP 3: Specify the family, protocol, IP address and port number
STEP 4: Create a socket using socket() function
STEP 5: Bind the IP address and Port number
STEP 6: Listen and accept the client's request for the connection
STEP 7: Read and Display the client's message
STEP 8: Stop

CLIENT ALGORITHM:

STEP 1: Start
STEP 2: Declare the variables for the socket
STEP 3: Specify the family, protocol, IP address and port number
STEP 4: Create a socket using socket() function
STEP 5: Call the connect() function
STEP 6: Read the input message
STEP 7: Send the input message to the server
STEP 8: Display the server's echo
STEP 9: Close the socket
STEP 10: Stop

Echo server program using UDP

SOURCE SERVER CODE:

```
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<unistd.h>
#include<netdb.h>
#include<stdio.h>
#include<string.h>
#include<arpa/inet.h>
#define MAXLINE 1024
int main(int argc,char **argv)
{
int sockfd;
int n;
socklen_t len;
char msg[1024];
```

```
struct sockaddr_in servaddr,cliaddr;
sockfd=socket(AF_INET,SOCK_DGRAM,0);
bzero(&servaddr,sizeof(servaddr));
servaddr.sin_family=AF_INET;
servaddr.sin_addr.s_addr=INADDR_ANY;
servaddr.sin_port=htons(5035);
printf("\n\n Binded");
bind(sockfd,(struct sockaddr*)&servaddr,sizeof(servaddr));
printf("\n\n Listening...");
for(;;)
{
     printf("\n ");
    len=sizeof(cliaddr);
    n=recvfrom(sockfd,msg,MAXLINE,0,(struct
sockaddr*)&cliaddr,&len);
    printf("\n Client's Message : %s\n",msg);
    if(n<6)
       perror("send error");
    sendto(sockfd,msg,n,0,(struct sockaddr*)&cliaddr,len);
}
return 0;
}
```

Reference: http://www.sourcecodesolutions.in/2010/09/udp-echo-client-server.html

Echo client program using UDP

SOURCE CLIENT CODE:

```
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<string.h>
#include<arpa/inet.h>
#include<string.h>
#include<arpa/inet.h>
#include<stdio.h>
#define MAXLINE 1024
int main(int argc,char* argv[])
{
int sockfd;
int n;
socklen_t len;
char sendline[1024],recvline[1024];
struct sockaddr_in servaddr;
strcpy(sendline,"");
printf("\n Enter the message : ");
scanf("%s",sendline);
```

```
sockfd=socket(AF_INET,SOCK_DGRAM,0);
bzero(&servaddr,sizeof(servaddr));
servaddr.sin_family=AF_INET;
servaddr.sin_addr.s_addr=inet_addr("127.0.0.1");
servaddr.sin_port=htons(5035);
connect(sockfd,(struct
sockaddr*)&servaddr,sizeof(servaddr));
len=sizeof(servaddr);
sendto(sockfd,sendline,MAXLINE,0,(struct
sockaddr*)&servaddr,len);
n=recvfrom(sockfd,recvline,MAXLINE,0,NULL,NULL
);
recvline[n]=0;
printf("\n Server's Echo : %s\n\n",recvline);
return 0;
}
```

Reference: http://www.sourcecodesolutions.in/2010/09/udp-echo-client-server.html

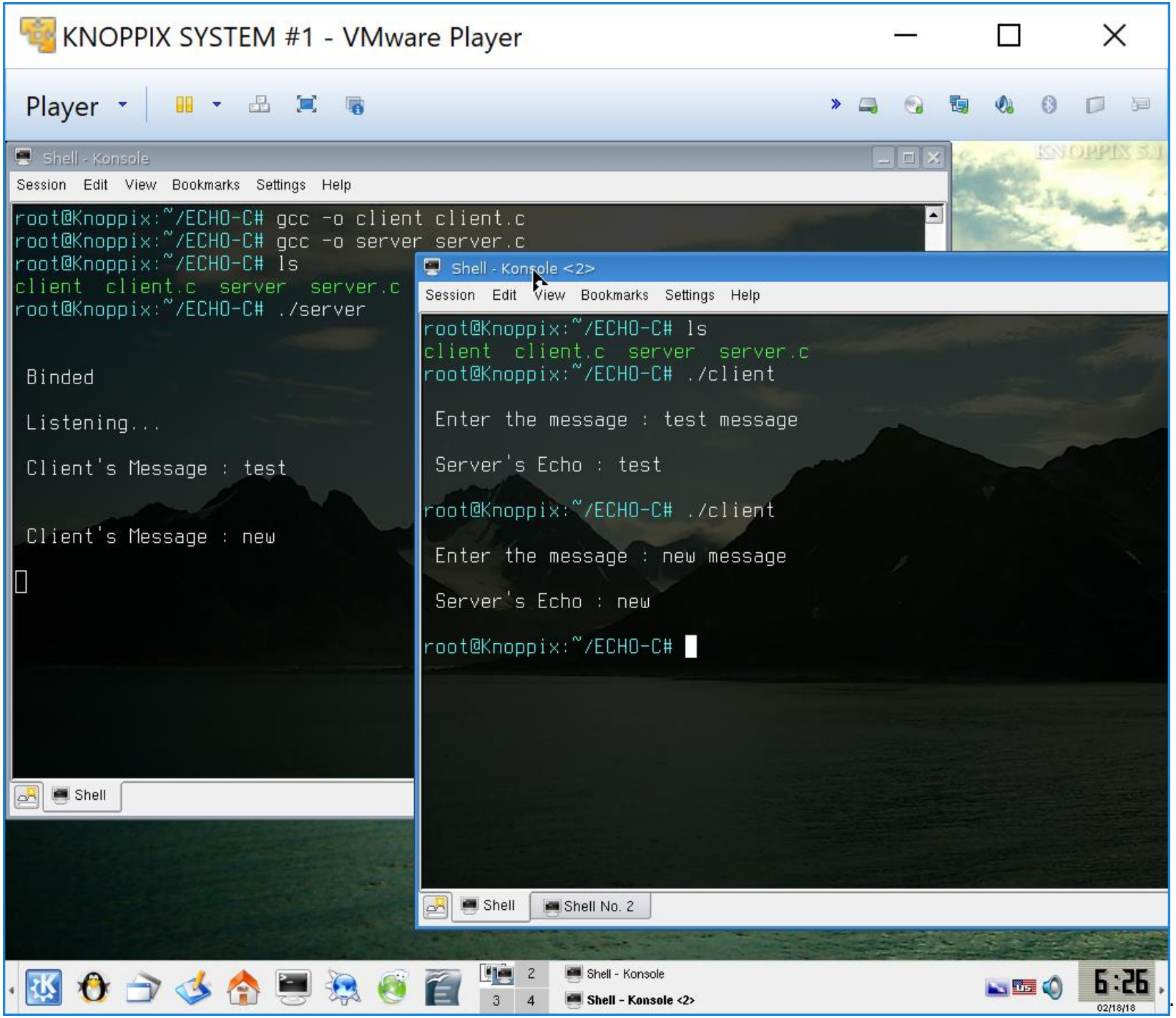

KNOPPIX SYSTEM #1 - VMware Player
Player
Shell - Konsole
Session Edit View Bookmarks Settings Help
root@Knoppix:~/ECHO-C# gcc -o client client.c
root@Knoppix:~/ECHO-C# gcc -o server server.c
root@Knoppix:~/ECHO-C# ls
client client.c server server.c
root@Knoppix:~/ECHO-C# ./server
Binded
Listening...
Client's Message : test
Client's Message : new
Shell - Konsole <2>
Session Edit View Bookmarks Settings Help
root@Knoppix:~/ECHO-C# ls
client client.c server server.c
root@Knoppix:~/ECHO-C# ./client
Enter the message : test message
Server's Echo : test
root@Knoppix:~/ECHO-C# ./client
Enter the message : new message
Server's Echo : new
root@Knoppix:~/ECHO-C#
Shell
Shell No. 2
Shell - Konsole
Shell - Konsole <2>
6:26
02/18/18

Figure 2.61: Sockets used in TCP communication

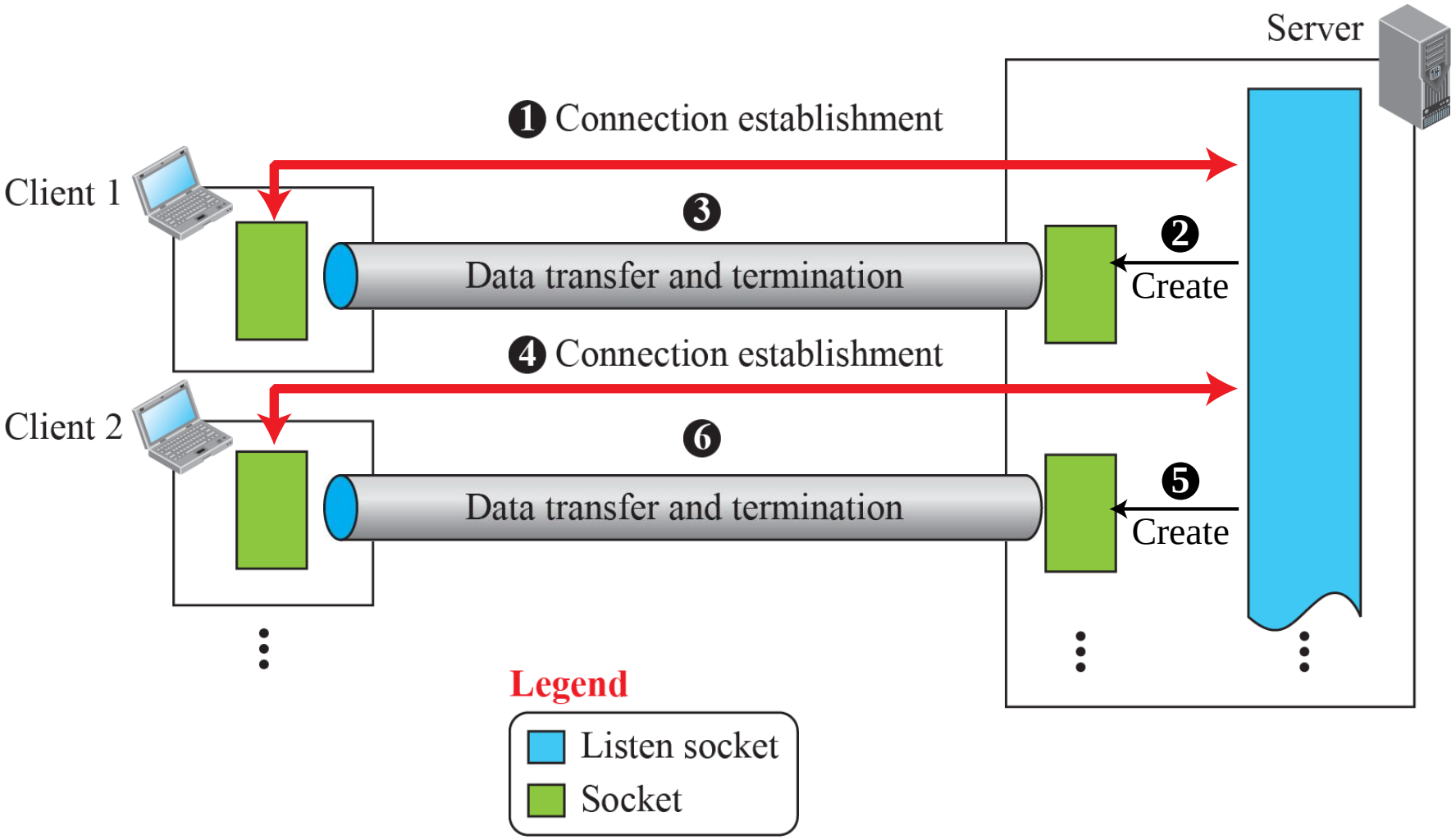


Figure 2.62: Flow diagram for iterative TCP communication

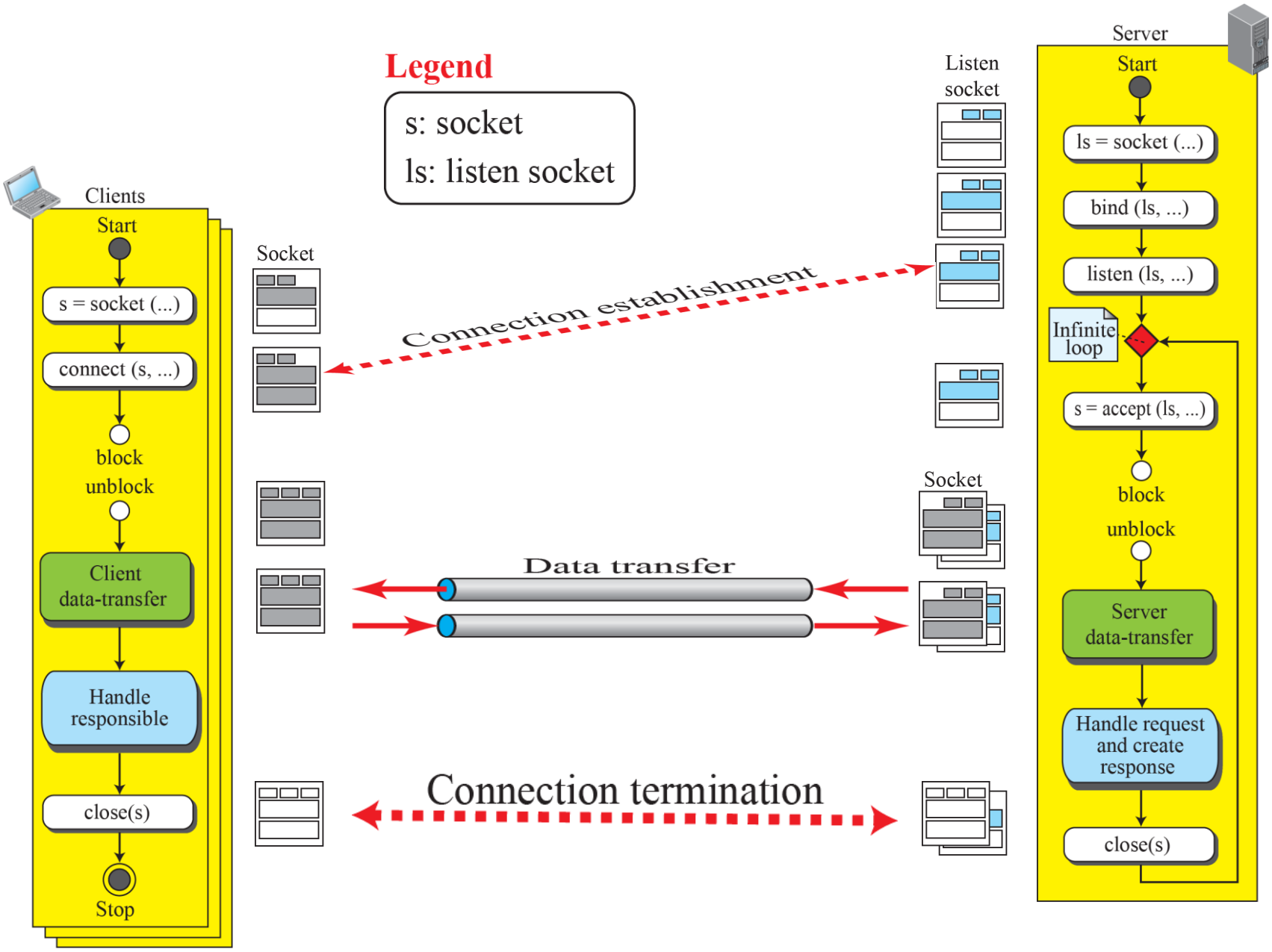


Figure 2.63: Flow diagram for data-transfer boxes

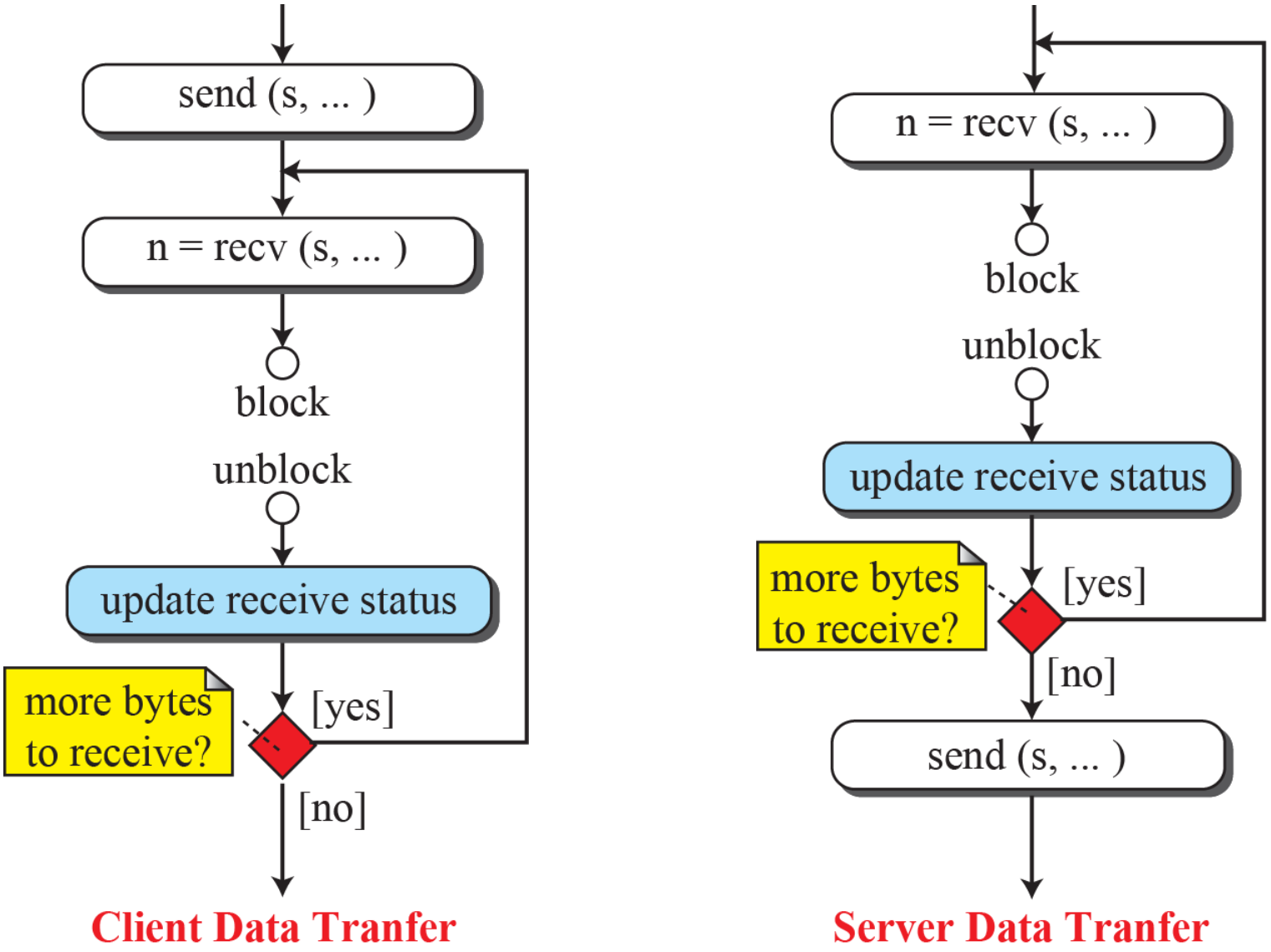
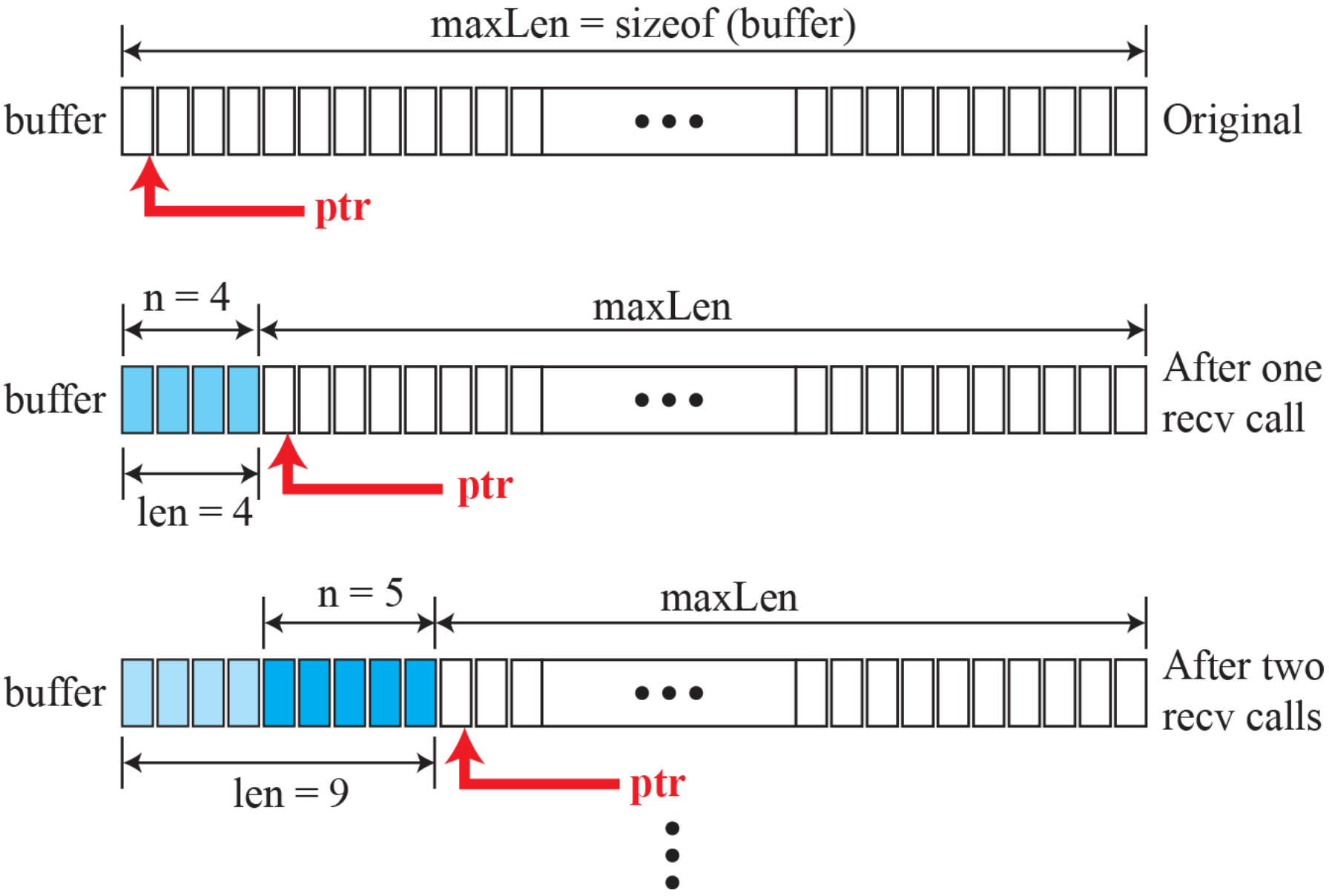
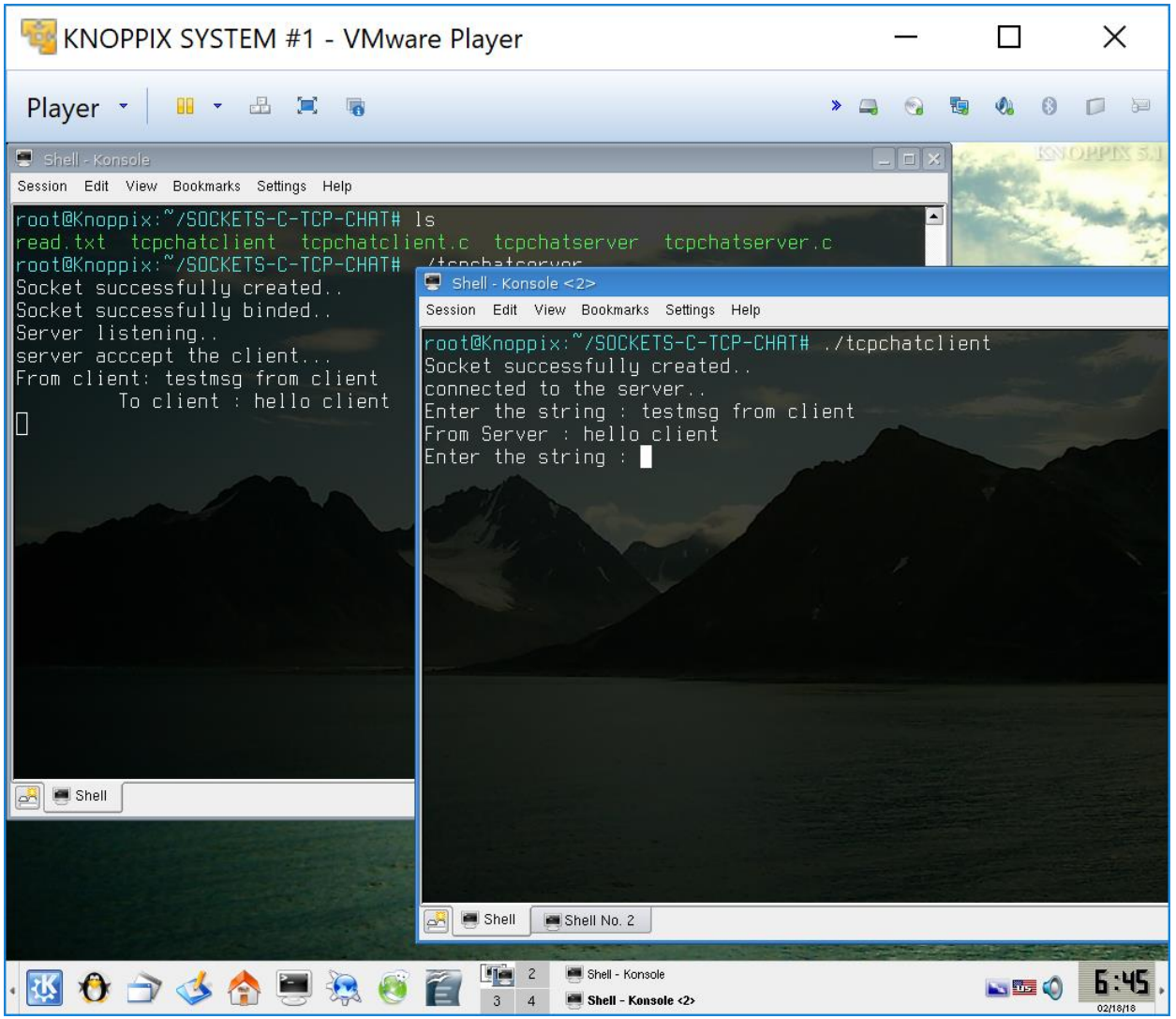


Figure 2.64: Buffer used for receiving

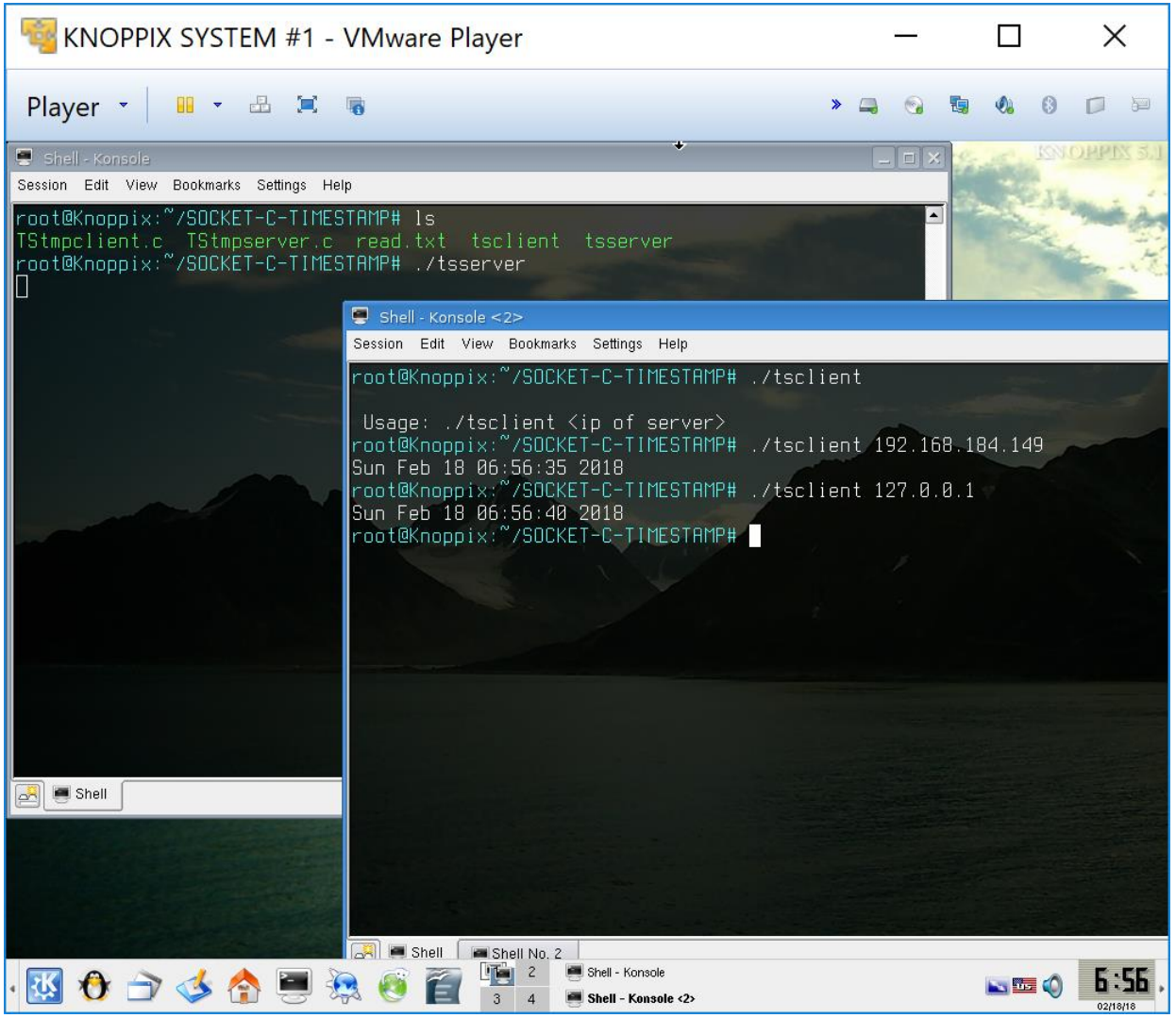


Please check out the other program: Socket program written in C using TCP transport protocol to chat between tcpchatserver.c & tcpchatclient.c



http://mcalabprogram.blogspot.com.au/2012/01/tcp-sockets-chat-applicationserver.html

Please check out the other program: Socket program written in C using TCP transport protocol to get Date & Timestamp from server TStmpserver.c & TStmpclient.c



Reference: https://www.thegeekstuff.com/2011/12/c-socket-programming/

Chapter 2: Summary

- *Applications in the Internet are designed using either a client-server paradigm or a peer-to-peer paradigm. In a client-server paradigm, an application program, called a server, provides services and another application program, called a client, receives services. A server program is an infinite program; a client program is finite. In a peer-to- peer paradigm, a peer can be both a client and a server.*

- *The World Wide Web (WWW) is a repository of information linked together from points all over the world. Hypertext and hypermedia documents are linked to one another through pointers. The HyperText Transfer Protocol (HTTP) is the main protocol used to access data on the World Wide Web (WWW).*

Chapter 2: Summary (continued)

- *File Transfer Protocol (FTP) is a TCP/IP client-server application for copying files from one host to another. FTP requires two connections for data transfer: a control connection and a data connection. FTP employs NVT ASCII for communication between dissimilar systems.*

- *TELNET is a client-server application that allows a user to log into a remote machine, giving the user access to the remote system. When a user accesses a remote system via the TELNET process, this is comparable to a time-sharing environment.*

Chapter 2: Summary (continued)

- *The Domain Name System (DNS) is a client-server application that identifies each host on the Internet with a unique name. DNS organizes the name space in a hierarchical structure to decentralize the responsibilities involved in naming. TELNET is a client-server application that allows a user to log into a remote machine, giving the user access to the remote system.*