Lecture 12

Client-Server Protocols

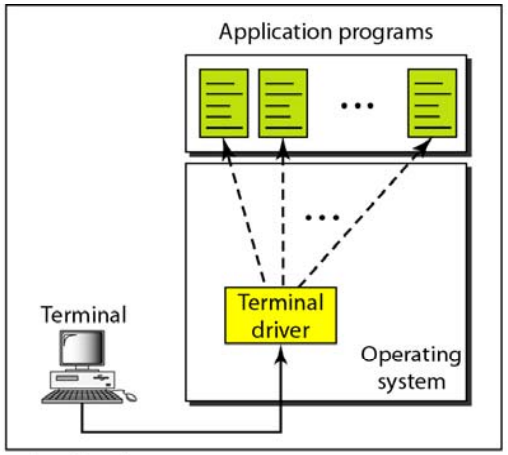
REMOTE LOGGING

It would be impossible to write a specific client/server program for each demand. The better solution is a general-purpose client/server program that lets a user access any application program on a remote computer.

Note

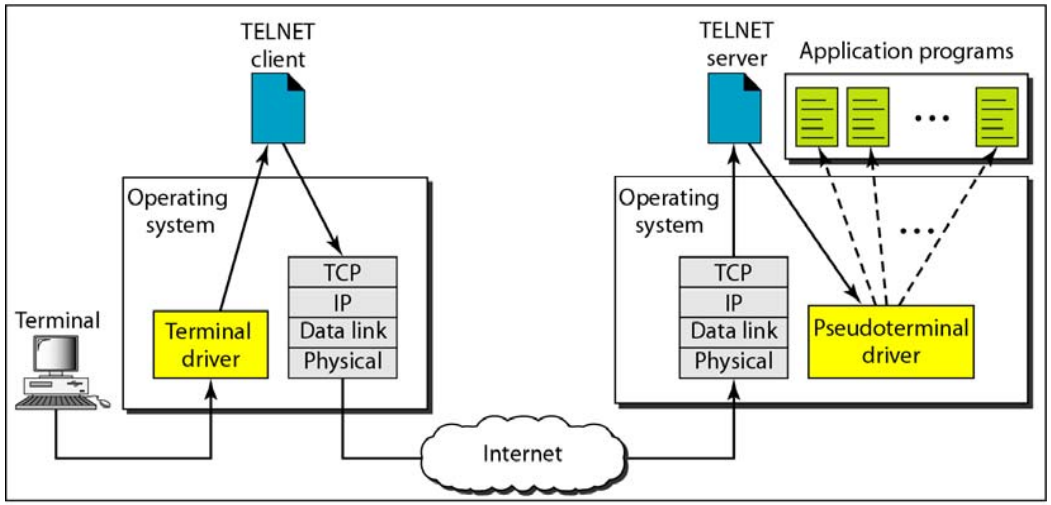
TELNET (TErminaL NETwork) is a general-purpose client/server application program.

Local and remote log-in



a. Local log-in

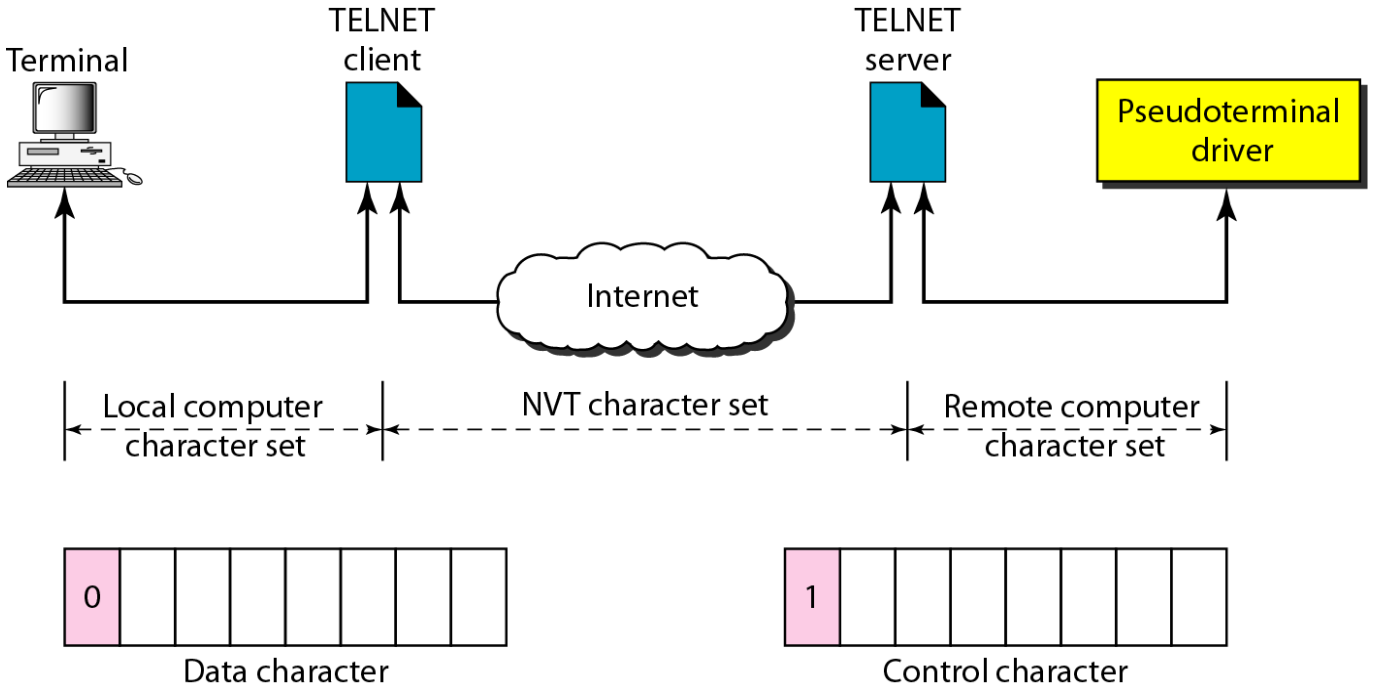
As a user types at the terminal, the keystrokes are accepted by the terminal driver, which passes the characters to the operation system. The operating system interprets the combination of the characters, and invokes desired program.



b. Remote log-in

Local operating system accepts the characters, and sends to the TELNET clients. The TELNET client sends to the TELNET server. Remote operating system is not designed to receive characters from a TELNET server. It is designed to receive from a terminal driver. So we add a pseudoterminal driver.

Concept of NVT



Each system has a special combination of characters as tokens.
DOS system: Ctrl+z means end-of-file token
UNIX system: Ctrl+d means end-of-file token.

A universal interface: Network virtual terminal (NVT)

Some NVT control characters

Character	*Decimal*	*Binary*	*Meaning*
EOF	236	11101100	End of file
EOR	239	11101111	End of record
SE	240	11110000	Suboption end
NOP	241	11110001	No operation
DM	242	11110010	Data mark
BRK	243	11110011	Break
IP	244	11110100	Interrupt process
AO	245	11110101	Abort output
AYT	246	11110110	Are you there?
EC	247	11110111	Erase character
EL	248	11111000	Erase line
GA	249	11111001	Go ahead
SB	250	11111010	Suboption begin
WILL	251	11111011	Agreement to enable option
WONT	252	11111100	Refusal to enable option
DO	253	11111101	Approval to option request
DONT	254	11111110	Denial of option request
IAC	255	11111111	Interpret (the next character) as control

An example of embedding

TELNET uses one TCP connection for sending data and control characters

Embed control characters in data stream

cat file1

Cat filea<backspace>1

Options: features available to a more sophisticated terminal
Users with simpler terminals can use default features.

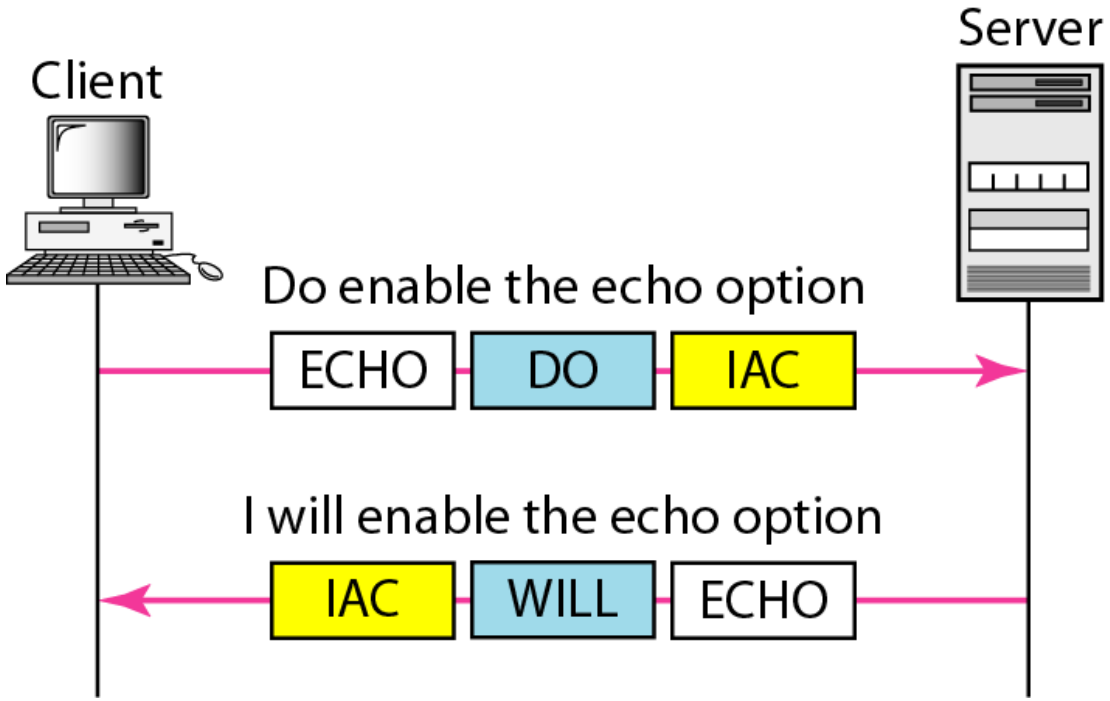
Code	*Option*	*Meaning*
0	Binary	Interpret as 8-bit binary transmission.
1	Echo	Echo the data received on one side to the other.
3	Suppress go ahead	Suppress go-ahead signals after data.
5	Status	Request the status of TELNET.
6	Timing mark	Define the timing marks.
24	Terminal type	Set the terminal type.
32	Terminal speed	Set the terminal speed.
34	Line mode	Change to line mode.

NVT character set for option negotiation

Character	*Decimal*	*Binary*	*Meaning*
WILL	251	11111011	1. Offering to enable 2. Accepting a request to enable
WONT	252	11111100	1. Rejecting a request to enable 2. Offering to disable 3. Accepting a request to disable
DO	253	11111101	1. Approving an offer to enable 2. Requesting to enable
DONT	254	11111110	1. Disapproving an offer to enable 2. Approving an offer to disable 3. Requesting to disable

Example

Here is an example shows an example of option negotiation. In this example, the client wants the server to echo each character sent to the server. The echo option is enabled by the server because it is the server that sends the characters back to the user terminal. Therefore, the client should request from the server the enabling of the option using DO. The request consists of three characters: IAC, DO, and ECHO. The server accepts the request and enables the option. It informs the client by sending the three-character approval: IAC, WILL, and ECHO.

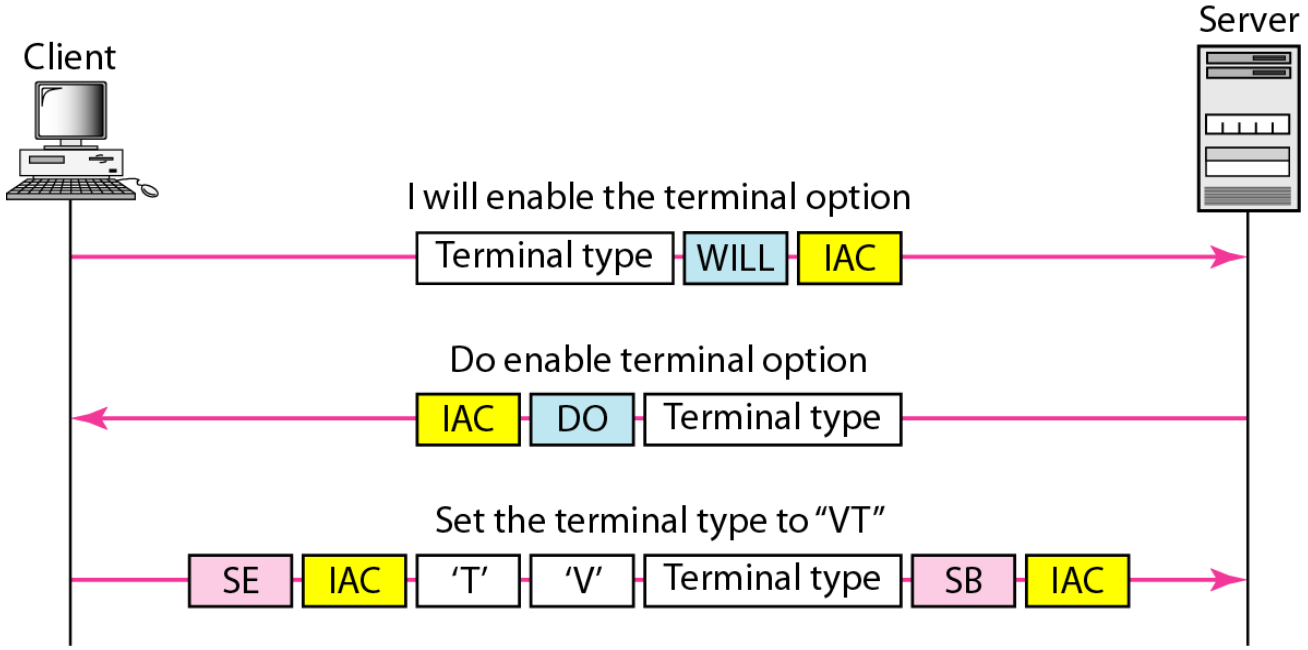


Some options require additional information. For example, to define type or speed of a terminal, the negotiation includes a string (for type) or a number (for speed).

Character set for suboptions

Character	*Decimal*	*Binary*	*Meaning*
SE	240	11110000	Suboption end
SB	250	11111010	Suboption begin

an example of suboption negotiation. In this example, the client wants to negotiate the type of the terminal.



Code	*Option*	*Meaning*
0	Binary	Interpret as 8-bit binary transmission.
1	Echo	Echo the data received on one side to the other.
3	Suppress go ahead	Suppress go-ahead signals after data.
5	Status	Request the status of TELNET.
6	Timing mark	Define the timing marks.
24	Terminal type	Set the terminal type.
32	Terminal speed	Set the terminal speed.
34	Line mode	Change to line mode.

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.

World Wide Web

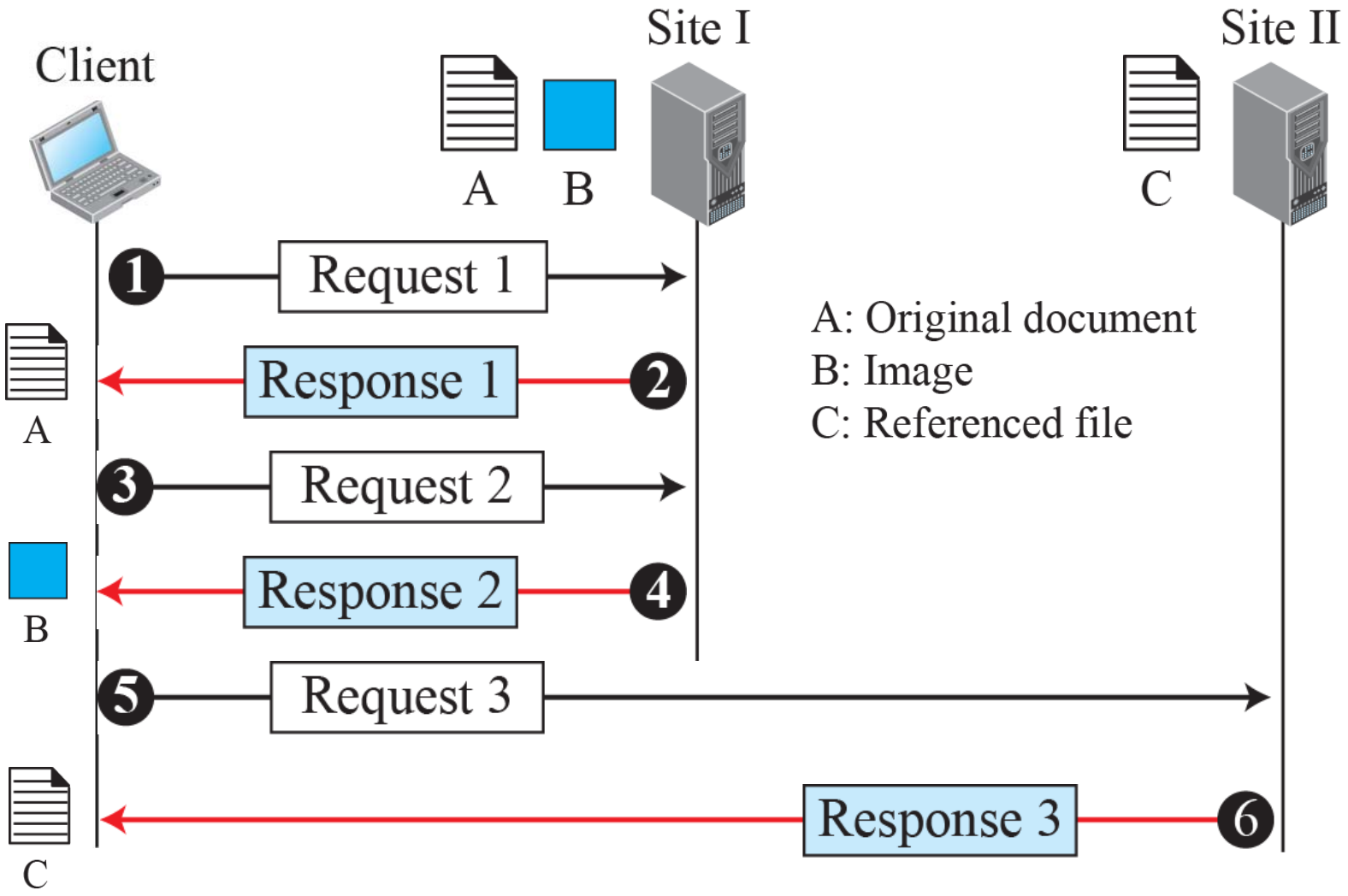
The idea of the Web was first proposed by Tim Berners-Lee in 1989 at CERN, the European Organization for Nuclear Research, to allow several researchers at different locations throughout Europe to access each other's research. The commercial Web started in the early 1990s.

Example 1

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

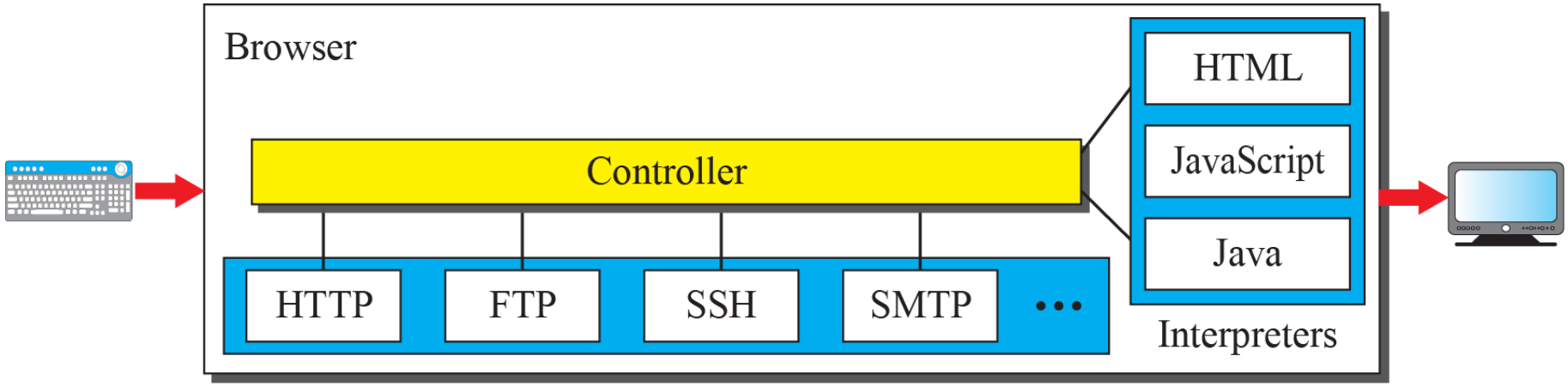
The main document and the image are stored in two separate files in the same site (file A and file B); the referenced text file is stored in another site (file C). Since we are dealing with three different files, we need three transactions if we want to see the whole document.

Example


Client
Site I
Site II
A
B
C
Request 1
Response 1
Request 2
Response 2
Request 3
Response 3
1
2
3
4
5
6
A: Original document
B: Image
C: Referenced file

Browser: web client, to interpret and display a web page

A browser has three parts: controller (which receives input from keyboard or mouse and uses the client programs to access the document, and uses one of the interpreters to display the document); Client protocols, and interpreter.



Example 2

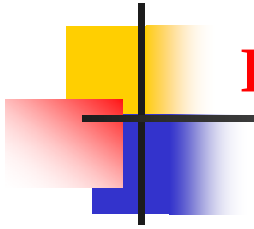
The **uniform resource locator URL**
https://www.ualberta.ca/engineering/faculty/hai-jiang

To define a web page, we need three identifiers.

Host: the host identifier can be IP address or unique name of the server.

Port: 16-bit integer

Path: identifies the location and name of the file in the underlying operation system.



HyperText Transfer Protocol

The HyperText Transfer Protocol (HTTP) is used to define how the client-server programs can be written to retrieve web pages from the Web. An HTTP client sends a request; an HTTP server returns a response. The server uses the port number 80; the client uses a temporary port number. HTTP uses the services of TCP, which, as discussed before, is a connection-oriented and reliable protocol.

Example 3

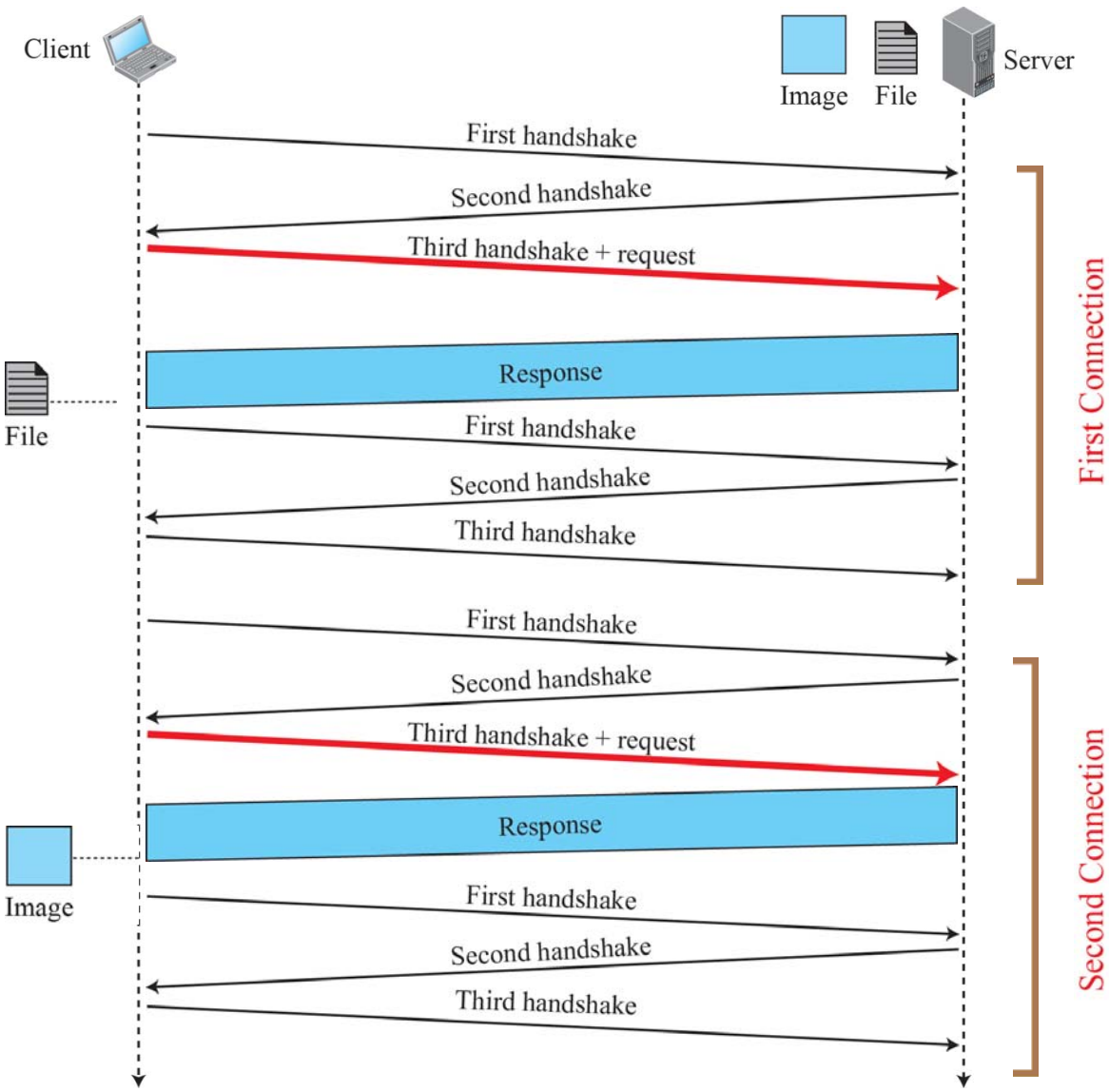
Figure on next slide shows an example of a nonpersistent connection. The client needs to access a file that contains one link to an image. The file and image are located on the same server. Here we need two connections. For each connection, TCP requires at least three handshake messages to establish 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.

Consider a web page that has multiple objects:

If the objects are at different servers, then a new TCP is needed to retrieve each object.

If the objects are at the same server, we can retrieve each object using a new TCP connection (nonpersistent), or use a TCP connection to retrieve all objects (persistent).

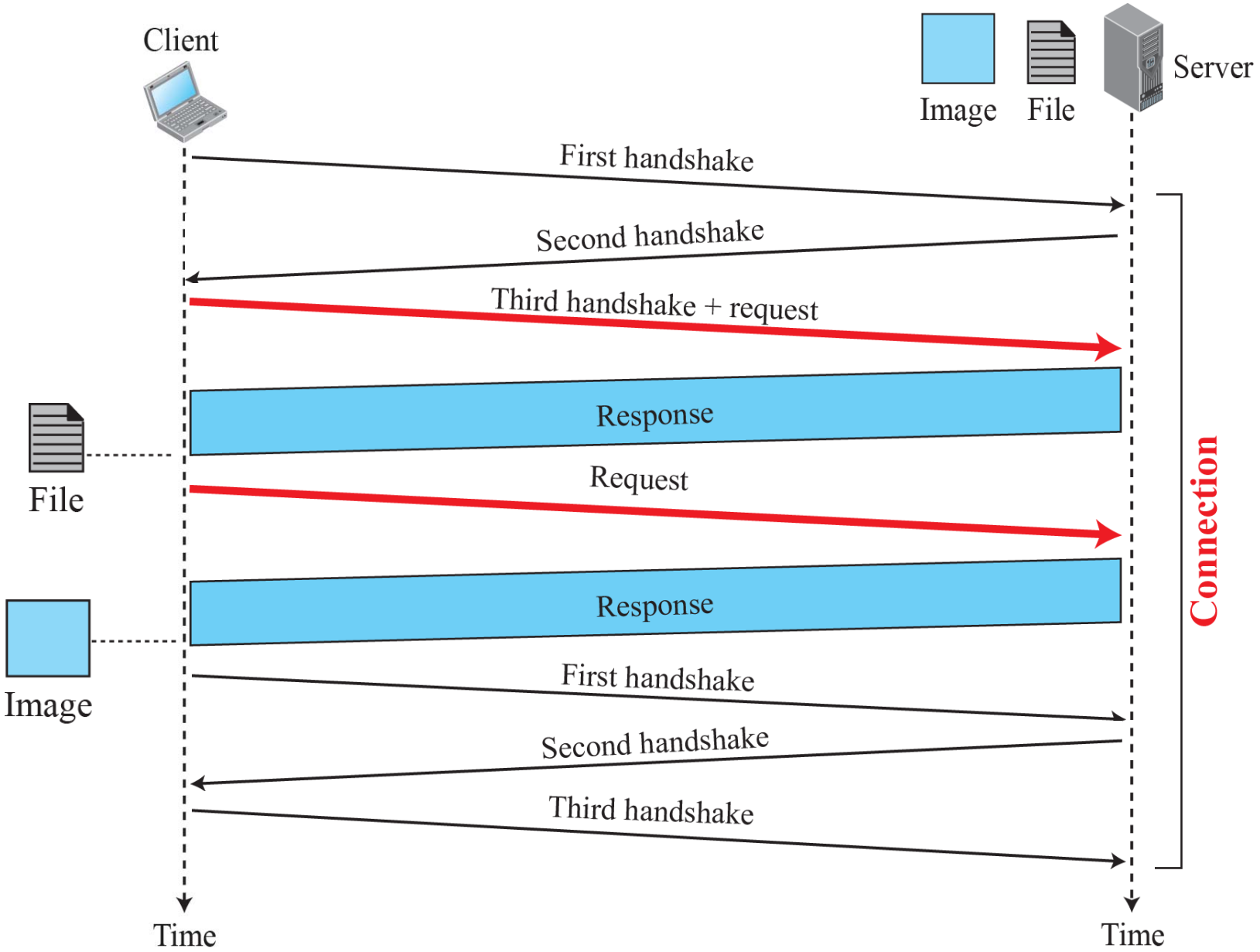
Example 3


Client
Image
File
Server
First handshake
Second handshake
Third handshake + request
Response
File
First handshake
Second handshake
Third handshake
First Connection
First handshake
Second handshake
Third handshake + request
Response
Image
First handshake
Second handshake
Third handshake
Second Connection

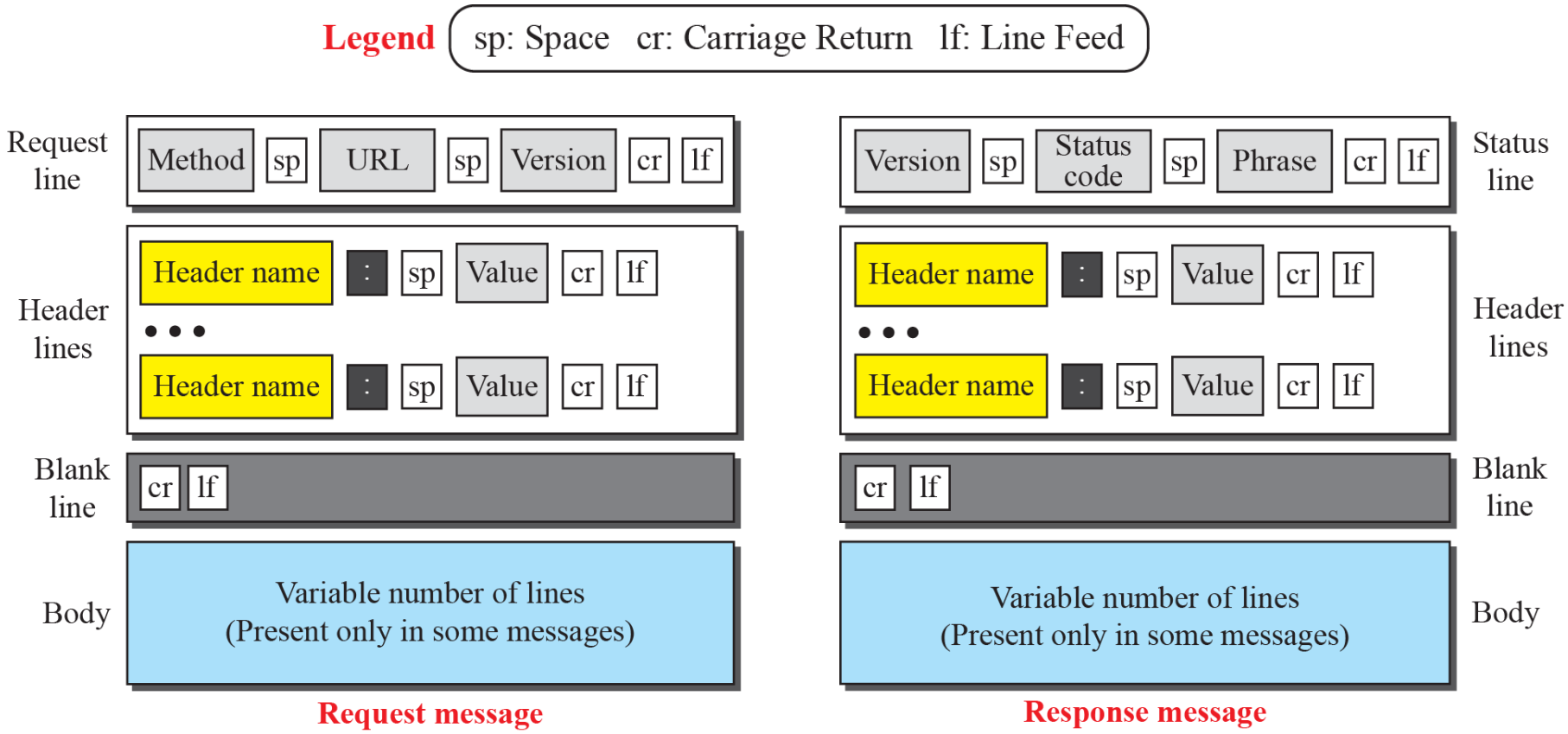
Example 4

Next Figure shows the same scenario as in Example 3, but using a persistent connection. Only one connection establishment and connection termination is used, but the request for the image is sent separately.

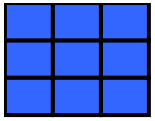
Example 4


Client
Image
File
Server
First handshake
Second handshake
Third handshake + request
Response
File
Request
Response
Image
First handshake
Second handshake
Third handshake
Connection
Time
Time

Formats of the request and response messages

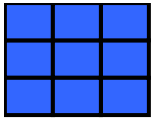


Method: defines the request type
Header line: provides additional information to server



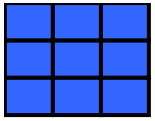
Methods (define the request types)

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



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



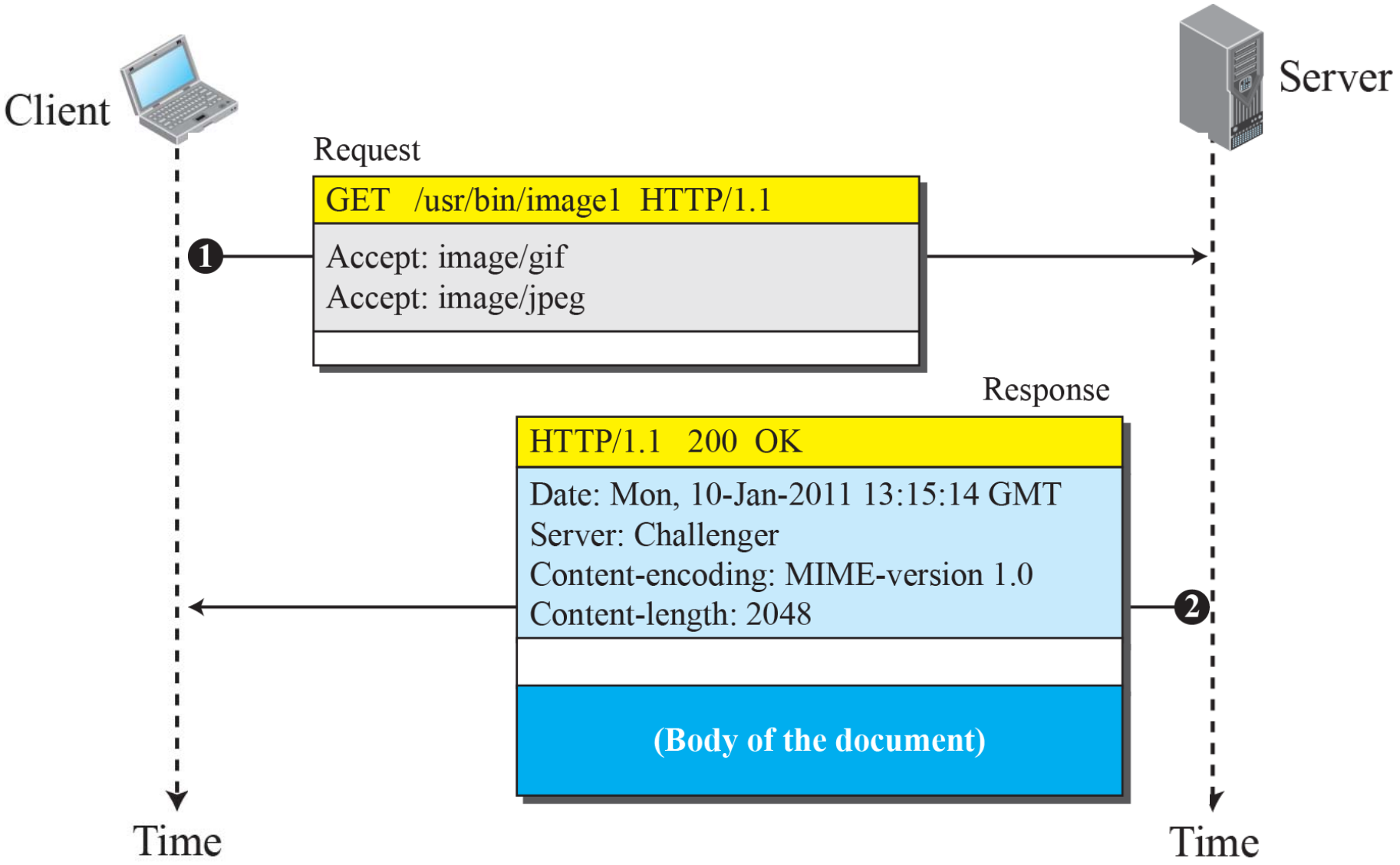
Response Header Names

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 5

This example retrieves a document. We use the GET method to retrieve an image with the path **/usr/bin/image26.** The request line shows the method (GET), the URL, and the HTTP version (26.1). 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 [multipurpose Internet Mail Extensions] version, which will be described in electronic mail), and length of the document. The body of the document follows the header..

Example 5


Client
Server
Request
GET /usr/bin/image1 HTTP/1.1
Accept: image/gif
Accept: image/jpeg
1
Response
HTTP/1.1 200 OK
Date: Mon, 10-Jan-2011 13:15:14 GMT
Server: Challenger
Content-encoding: MIME-version 1.0
Content-length: 2048
2
(Body of the document)
Time
Time

26-3 ELECTRONIC MAIL

Electronic mail (or e-mail) allows users to exchange messages. The nature of this application is different from other applications discussed so far. This means that the idea of client/server programming should be implemented in another way: using some intermediate computers (servers).

26.3.1 Architecture

To explain the architecture of e-mail, we give a common scenario, as shown in Figure 26.12. Another possibility is the case in which Alice or Bob is directly connected to the corresponding mail server, in which LAN or WAN connection is not required, but this variation in the scenario does not affect our discussion.

Figure 26.12: **Common scenario**

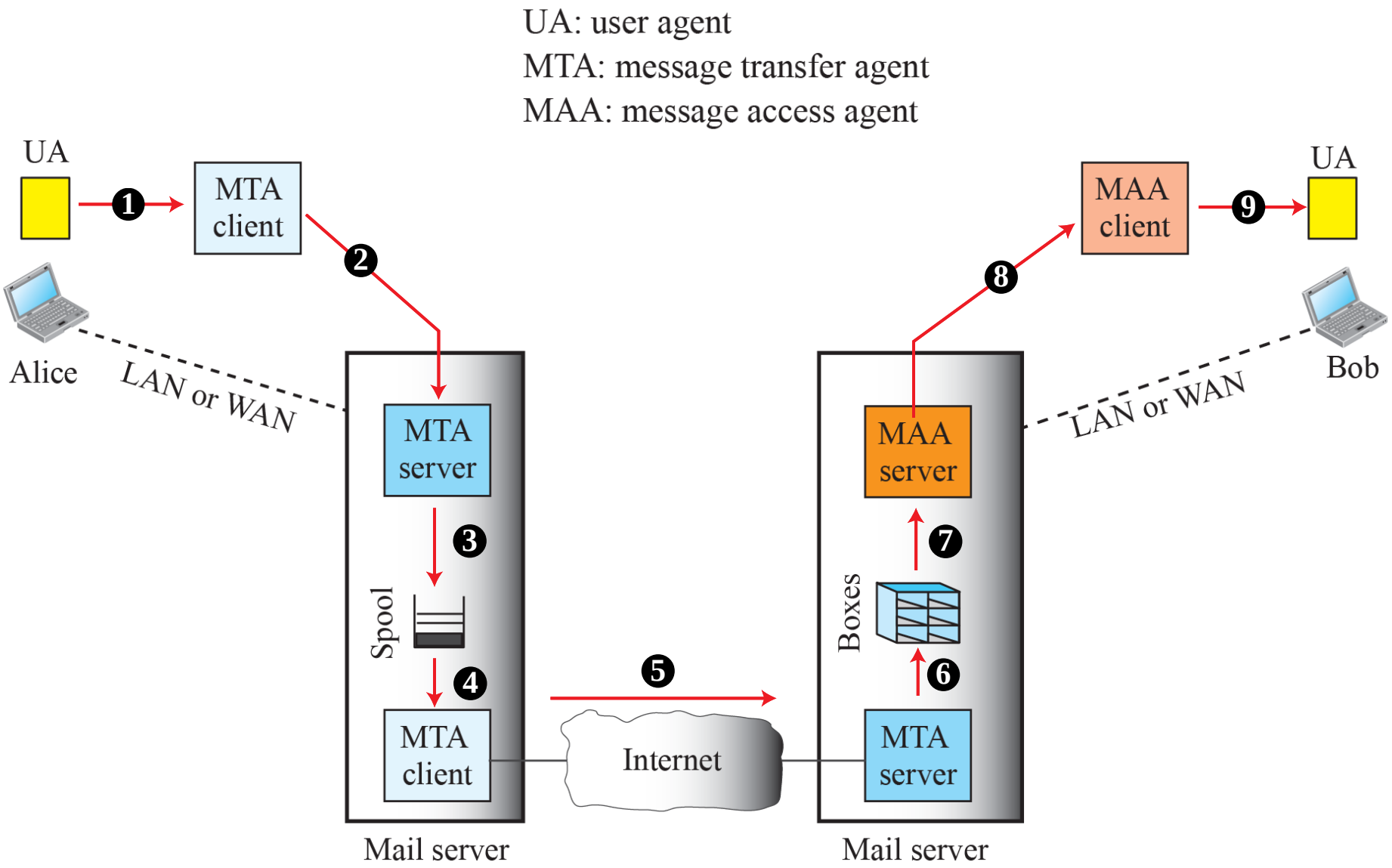


Figure 26.13: **Format of an e-mail**

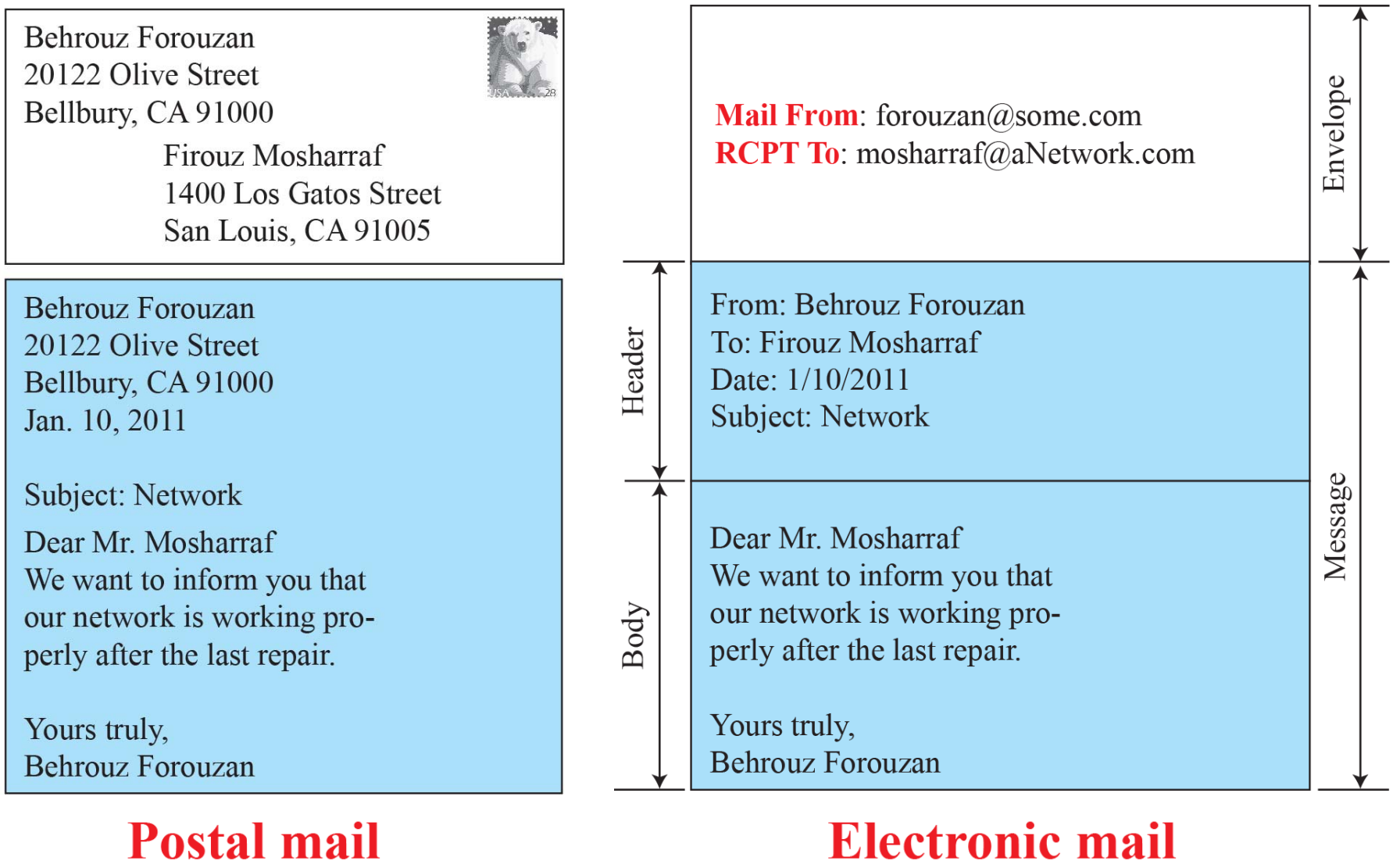


Figure 26.14: **E-mail address**

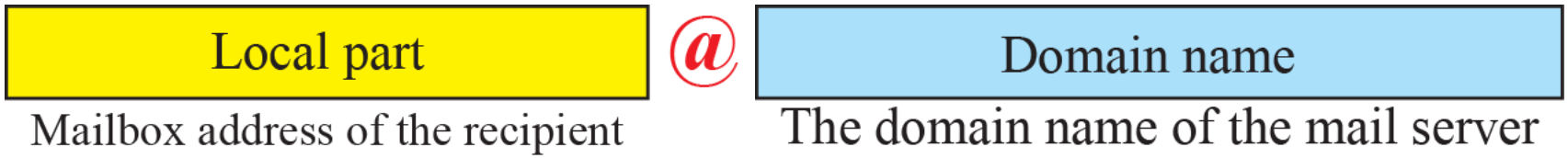
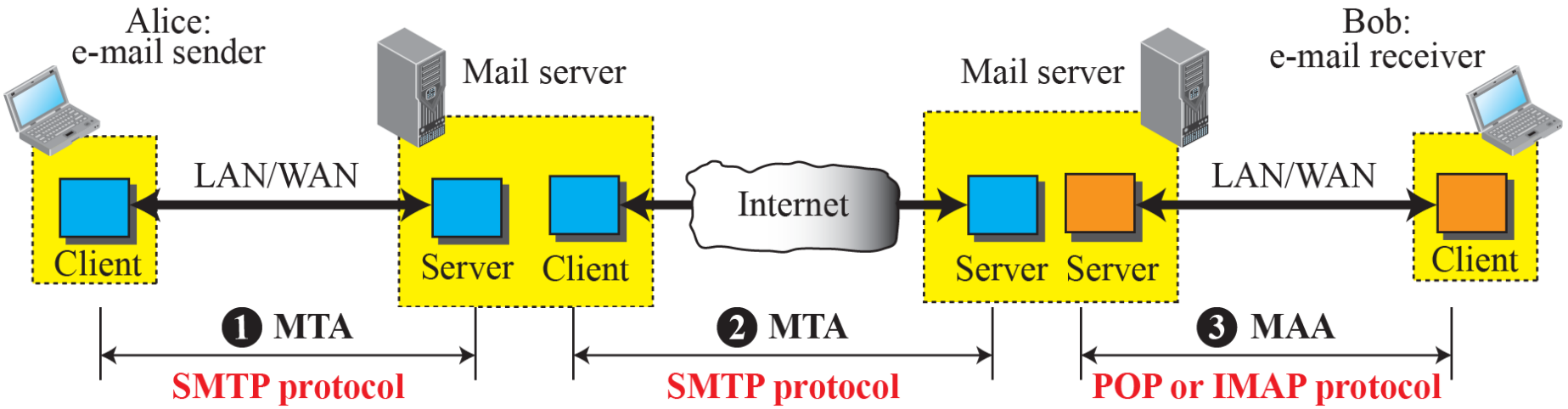


Figure 26.15: **Protocols used in electronic mail**



SMTP: Simple Mail Transfer Protocol
POP: Post Office Protocol
IMAP: Internet Message Access Protocol

Table 26.6: SMTP Commands

Keyword	*Argument(s)*	*Description*
HELO	Sender's host name	Identifies itself
MAIL FROM	Sender of the message	Identifies the sender of the message
RCPT TO	Intended recipient	Identifies the recipient of the message
DATA	Body of the mail	Sends the actual message
QUIT		Terminates the message
RSET		Aborts the current mail transaction
VRFY	Name of recipient	Verifies the address of the recipient
NOOP		Checks the status of the recipient
TURN		Switches the sender and the recipient
EXPN	Mailing list	Asks the recipient to expand the mailing list.
HELP	Command name	Asks the recipient to send information about the command sent as the argument
SEND FROM	Intended recipient	Specifies that the mail be delivered only to the terminal of the recipient, and not to the mailbox
SMOL FROM	Intended recipient	Specifies that the mail be delivered to the terminal *or* the mailbox of the recipient
SMAL FROM	Intended recipient	Specifies that the mail be delivered to the terminal *and* the mailbox of the recipient

Table 26.7: SMTP responses (Continued)

Code	*Description*
	Positive Completion Reply
211	System status or help reply
214	Help message
220	Service ready
221	Service closing transmission channel
250	Request command completed
251	User not local; the message will be forwarded
	Positive Intermediate Reply
354	Start mail input
	Transient Negative Completion Reply
421	Service not available
450	Mailbox not available
451	Command aborted: local error
452	Command aborted; insufficient storage

Table 26.7: SMTP responses (continued)

Permanent Negative Completion Reply	
500	Syntax error; unrecognized command
501	Syntax error in parameters or arguments
502	Command not implemented
503	Bad sequence of commands
504	Command temporarily not implemented
550	Command is not executed; mailbox unavailable
551	User not local
552	Requested action aborted; exceeded storage location
553	Requested action not taken; mailbox name not allowed
554	Transaction failed

Example 26.12

To show the three mail transfer phases, we show all of the steps described above using the information depicted in Figure 26.16. In the figure, we have separated the messages related to the envelope, header, and body in the data transfer section. Note that the steps in this figure are repeated two times in each e-mail transfer: once from the e-mail sender to the local mail server and once from the local mail server to the remote mail server. The local mail server, after receiving the whole e-mail message, may spool it and send it to the remote mail server at another time.

Figure 26.16: **Example 26.12**

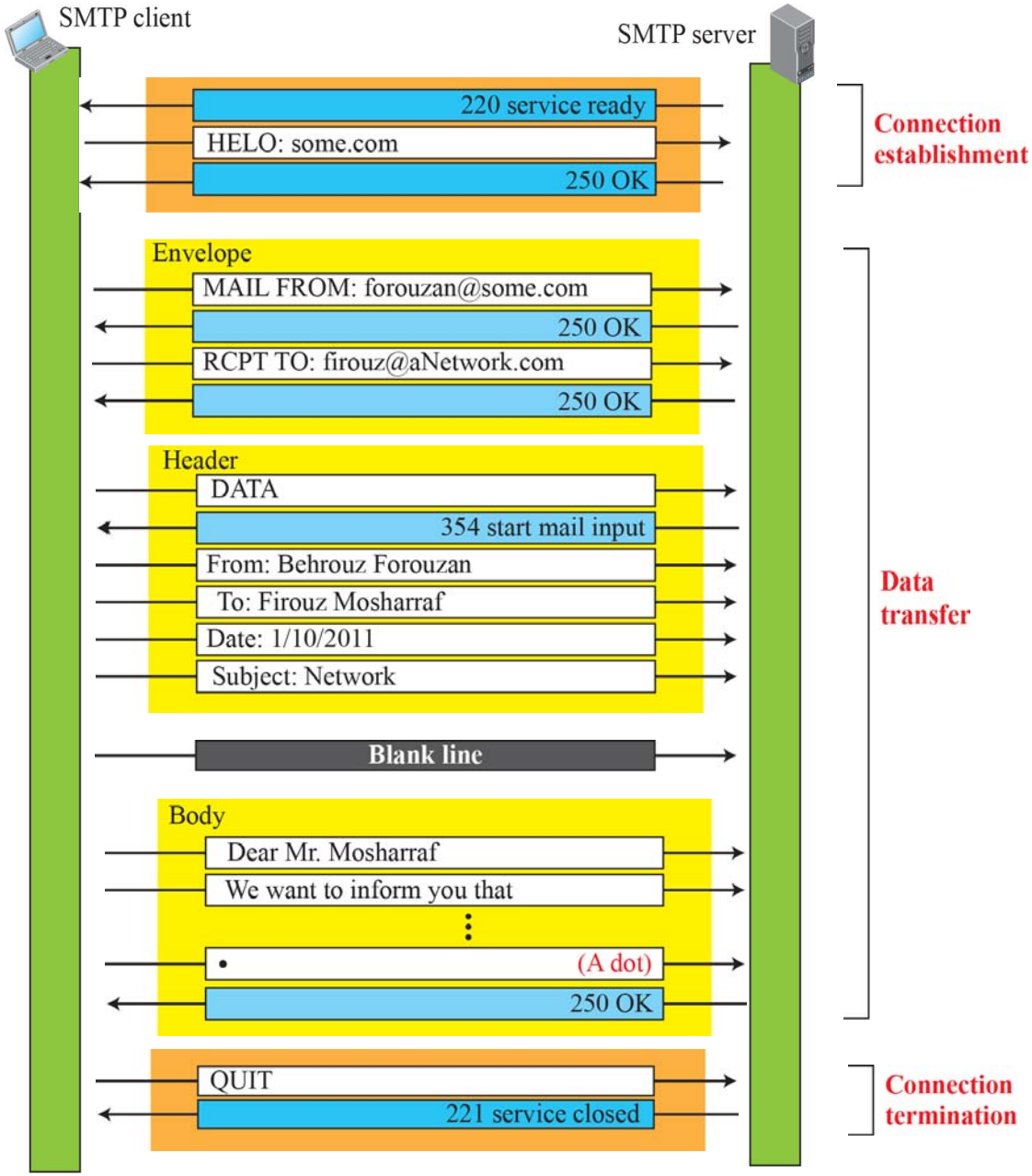


Figure 26.17: **POP3**

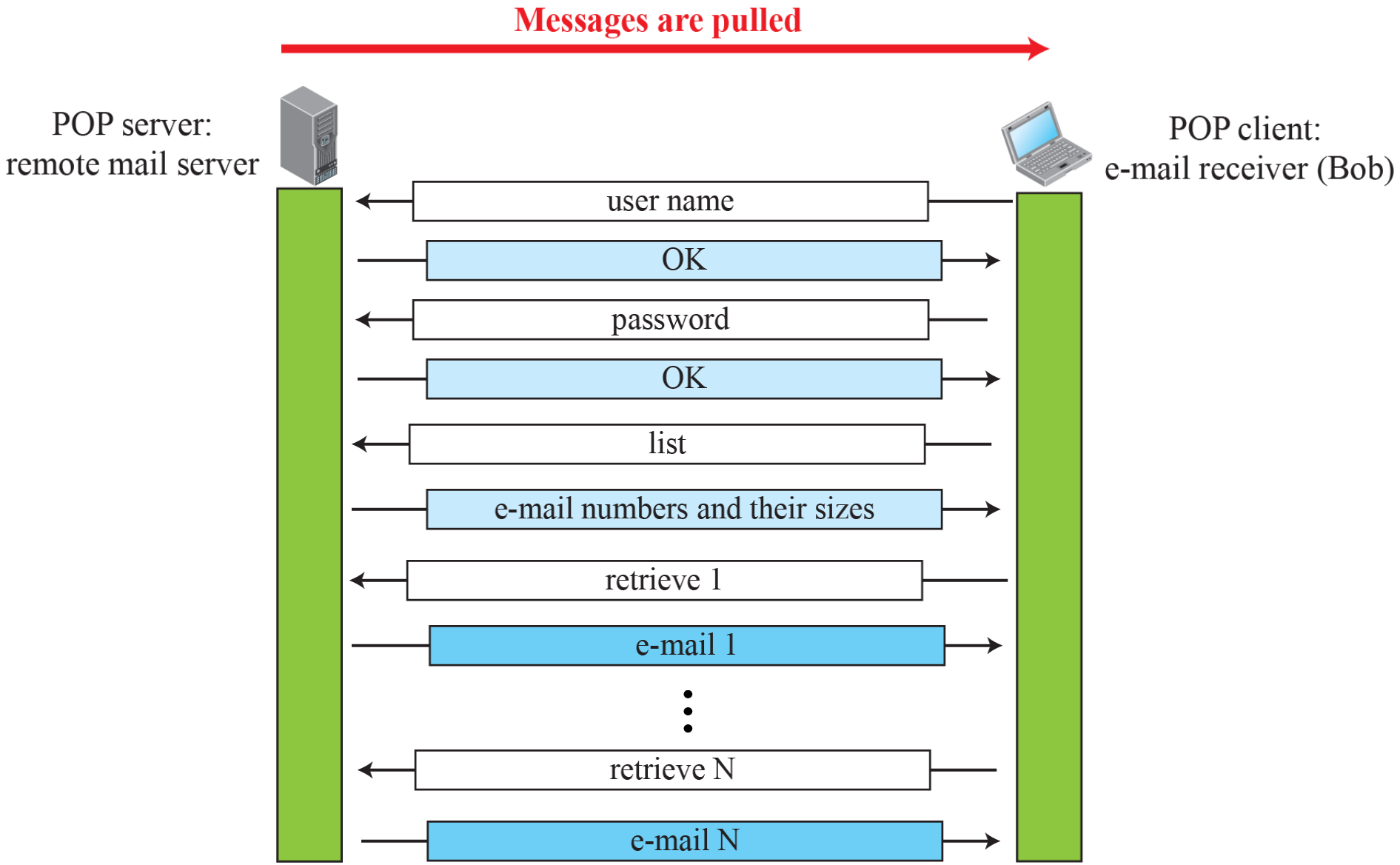


Figure 26.18: **MIME (Multipurpose Internet Mail Extensions)**

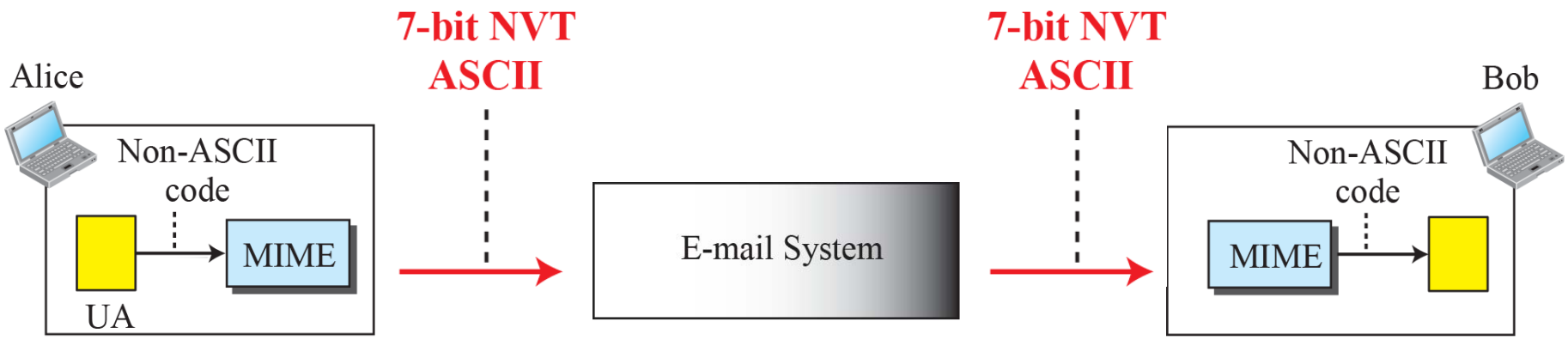


Figure 26.19: **MIME header**

MIME defines 5 headers that can be added to the original email header section to define the transformation parameters.

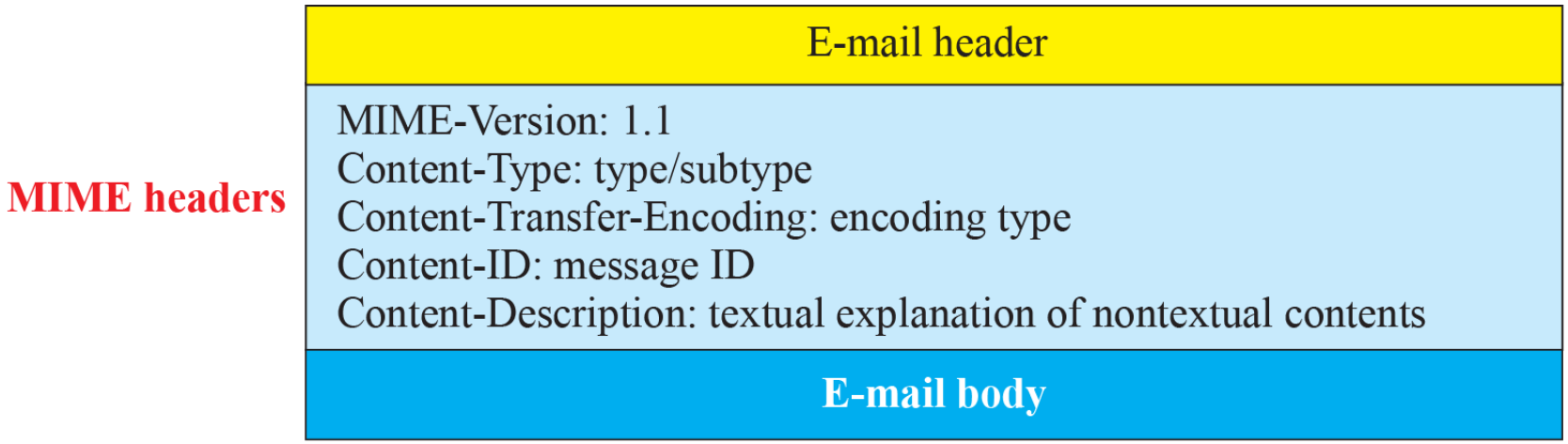


Table 26.8: Data Types and Subtypes in MIME

Type	*Subtype*	*Description*
Text	Plain	Unformatted
	HTML	HTML format (see Appendix C)
Multipart	Mixed	Body contains ordered parts of different data types
	Parallel	Same as above, but no order
	Digest	Similar to Mixed, but the default is message/RFC822
	Alternative	Parts are different versions of the same message
Message	RFC822	Body is an encapsulated message
	Partial	Body is a fragment of a bigger message
	External-Body	Body is a reference to another message
Image	JPEG	Image is in JPEG format
	GIF	Image is in GIF format
Video	MPEG	Video is in MPEG format
Audio	Basic	Single channel encoding of voice at 8 KHz
Application	PostScript	Adobe PostScript
	Octet-stream	General binary data (eight-bit bytes)

Table 2.9: Methods for Content-Transfer-Encoding

Type	*Description*
7-bit	NVT ASCII characters with each line less than 1000 characters
8-bit	Non-ASCII characters with each line less than 1000 characters
Binary	Non-ASCII characters with unlimited-length lines
Base64	6-bit blocks of data encoded into 8-bit ASCII characters
Quoted-printable	Non-ASCII characters encoded as an equal sign plus an ASCII code

Figure 26.20: **Base64 conversion**

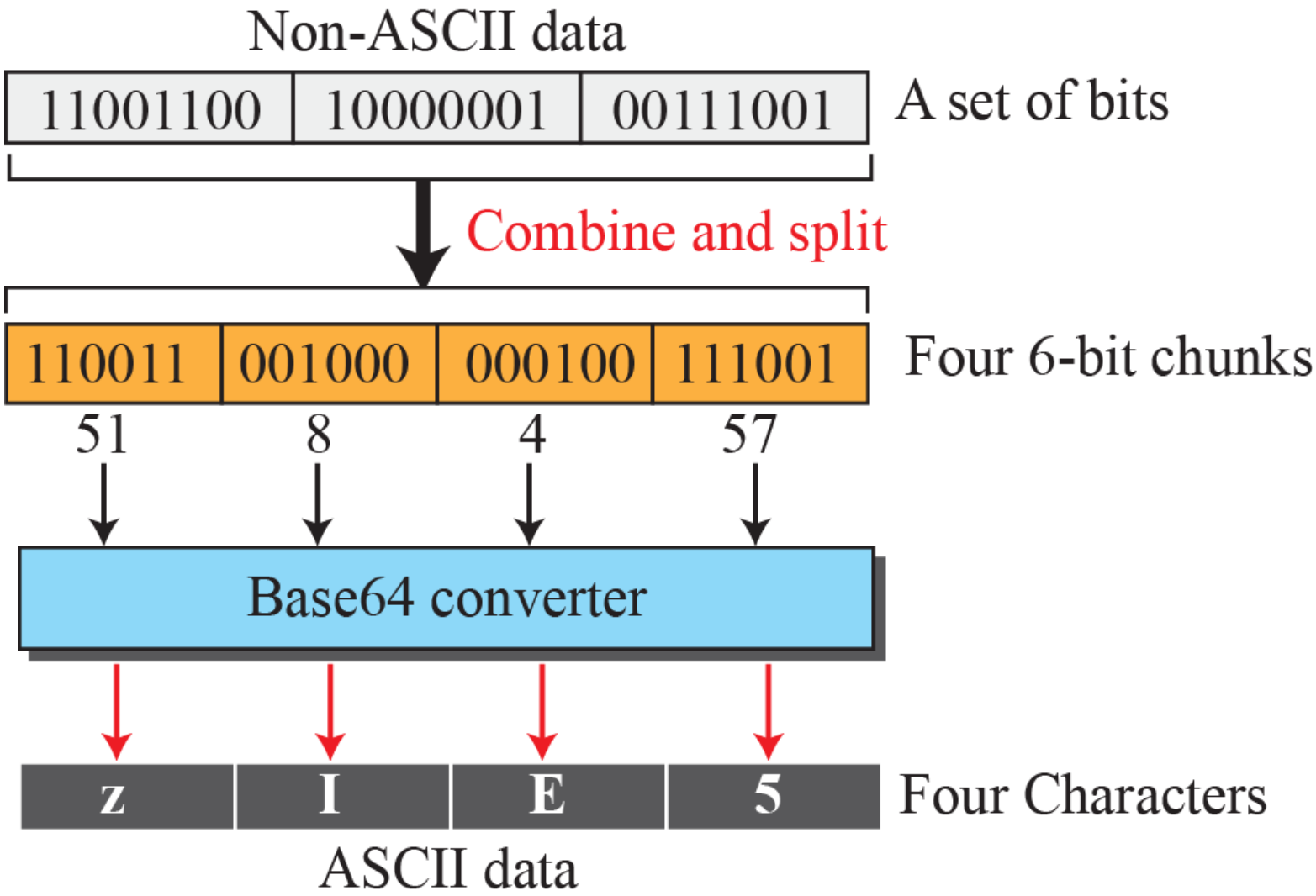
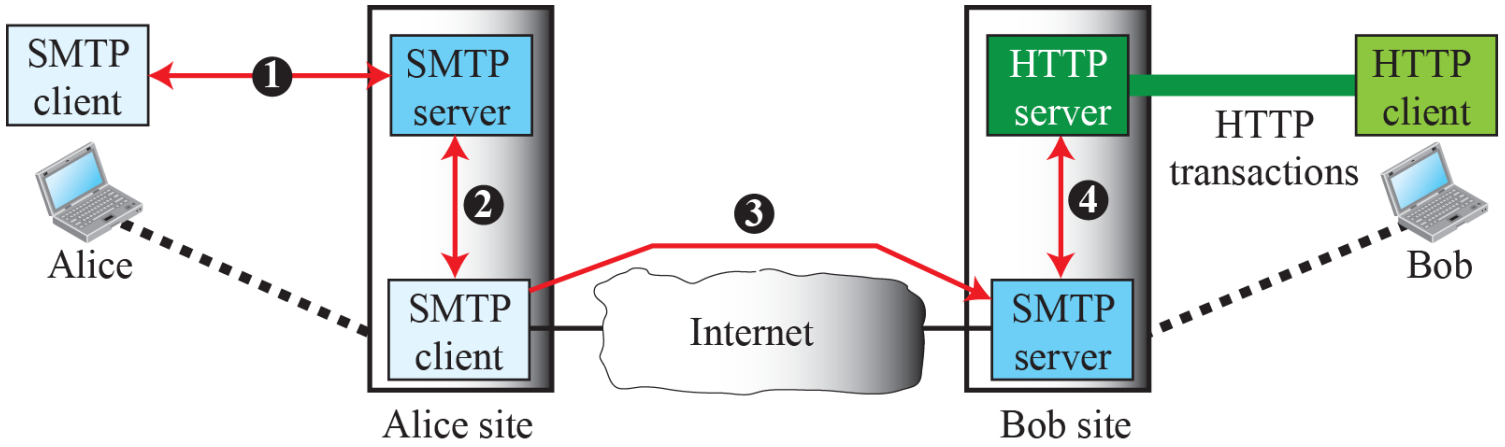


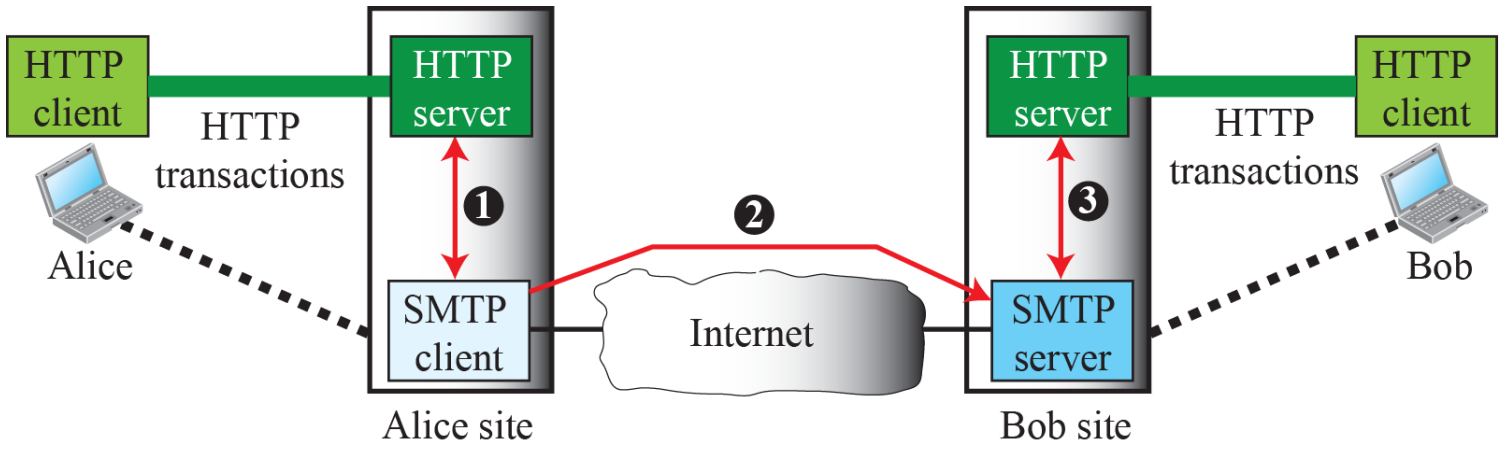
Table 26.10: Base64 Converting Table

Value	*Code*	*Value*	*Code*	*Value*	*Code*	*Value*	*Code*	*Value*	*Code*	*Value*	*Code*
0	**A**	11	**L**	22	**W**	33	**h**	44	**s**	55	**3**
1	**B**	12	**M**	23	**X**	34	**i**	45	**t**	56	**4**
2	**C**	13	**N**	24	**Y**	35	**j**	46	**u**	57	**5**
3	**D**	14	**O**	25	**Z**	36	**k**	47	**v**	58	**6**
4	**E**	15	**P**	26	**a**	37	**l**	48	**w**	59	**7**
5	**F**	16	**Q**	27	**b**	38	**m**	49	**x**	60	**8**
6	**G**	17	**R**	28	**c**	39	**n**	50	**y**	61	**9**
7	**H**	18	**S**	29	**d**	40	**o**	51	**z**	62	+
8	**I**	19	**T**	30	**e**	41	**p**	52	**0**	63	/
9	**J**	20	**U**	31	**f**	42	**q**	53	**1**		
10	**K**	21	**V**	32	**g**	43	**r**	54	**2**		

Figure 26.22: **Web-based e-mail, cases I and II**



Case 1: Only receiver uses HTTP



Case 2: Both sender and receiver use HTTP