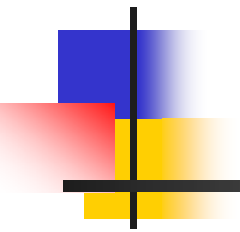
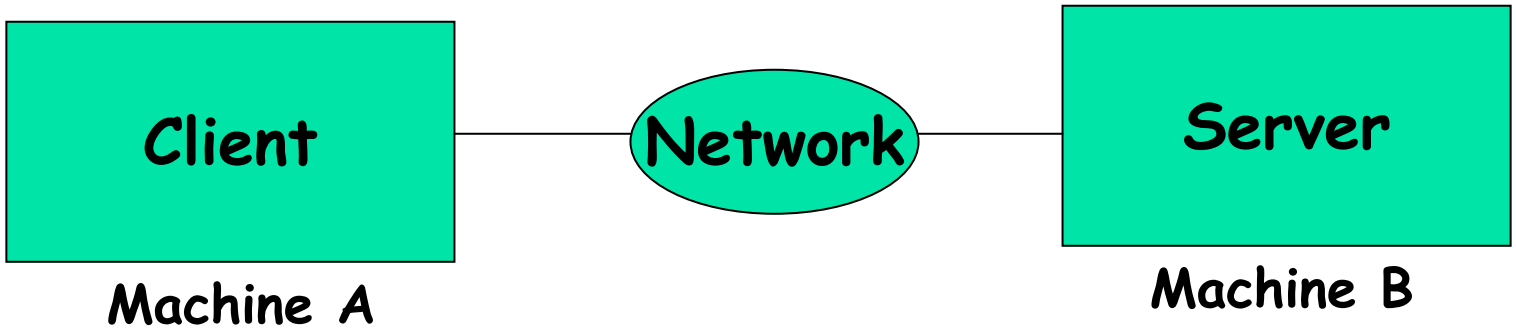


UNIX Network Programming

Overview of Socket API

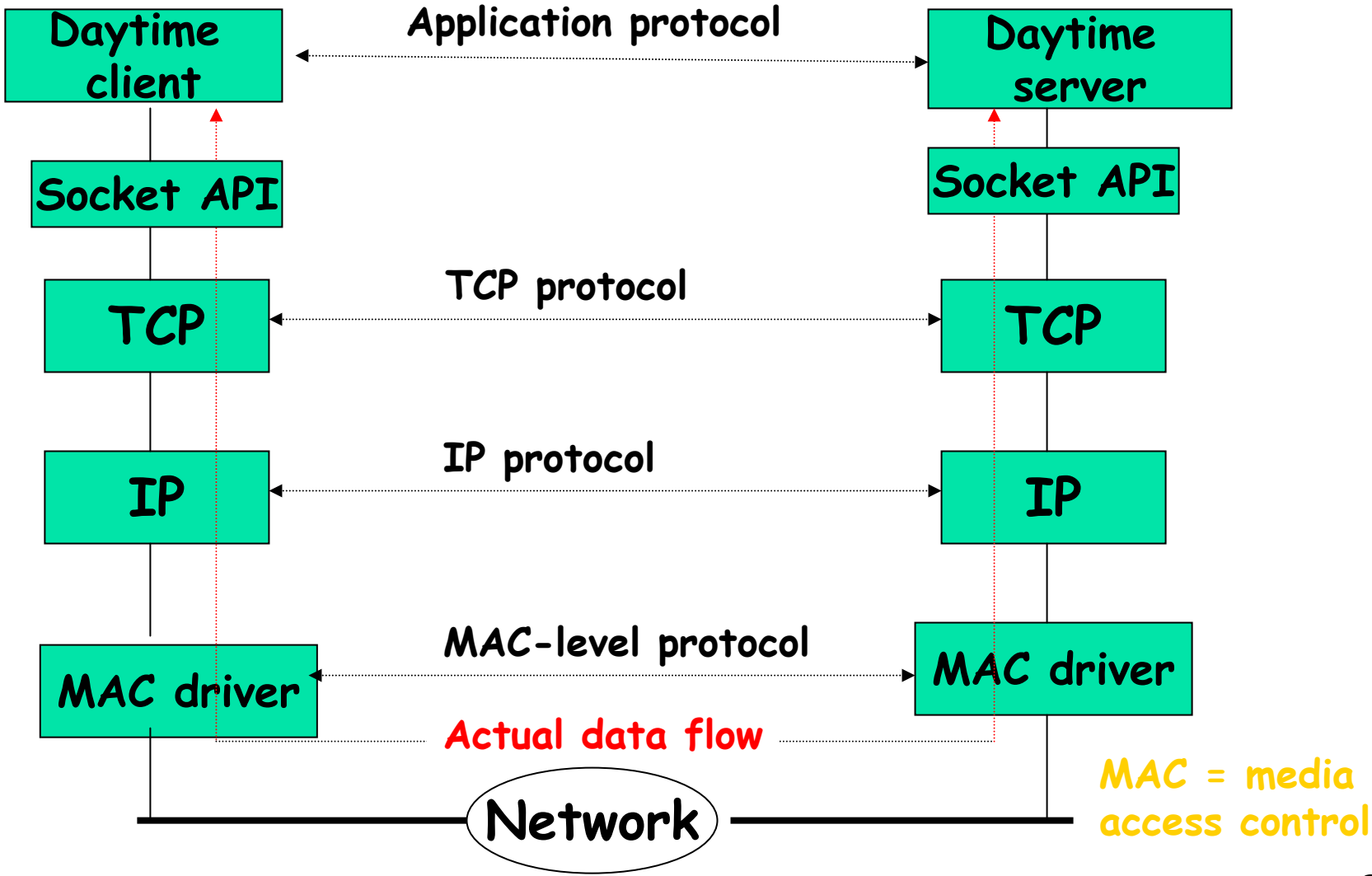
Network Programming Basics

Client-Server Model

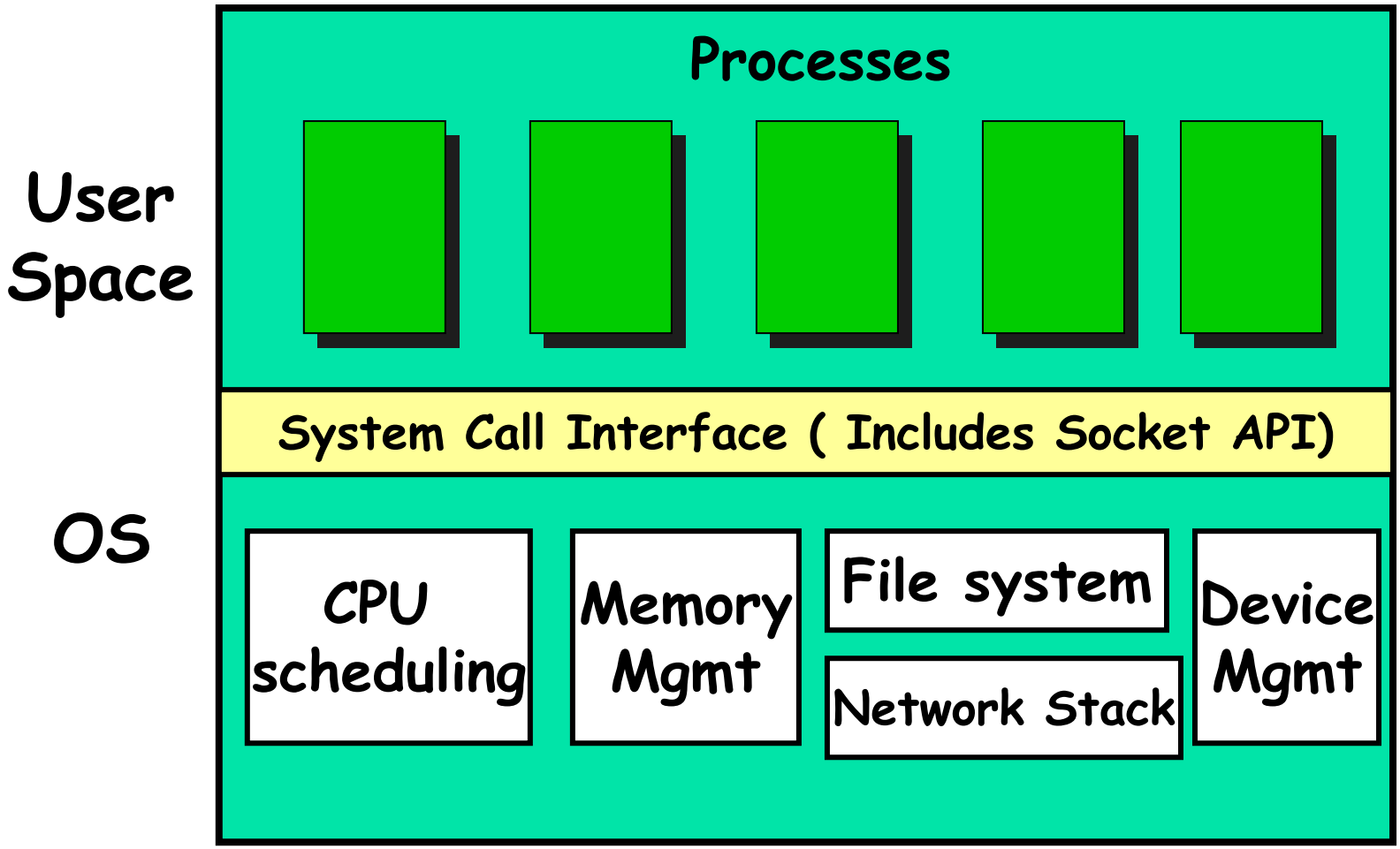


- Web browser and server
- FTP client and server
- Telnet client and server

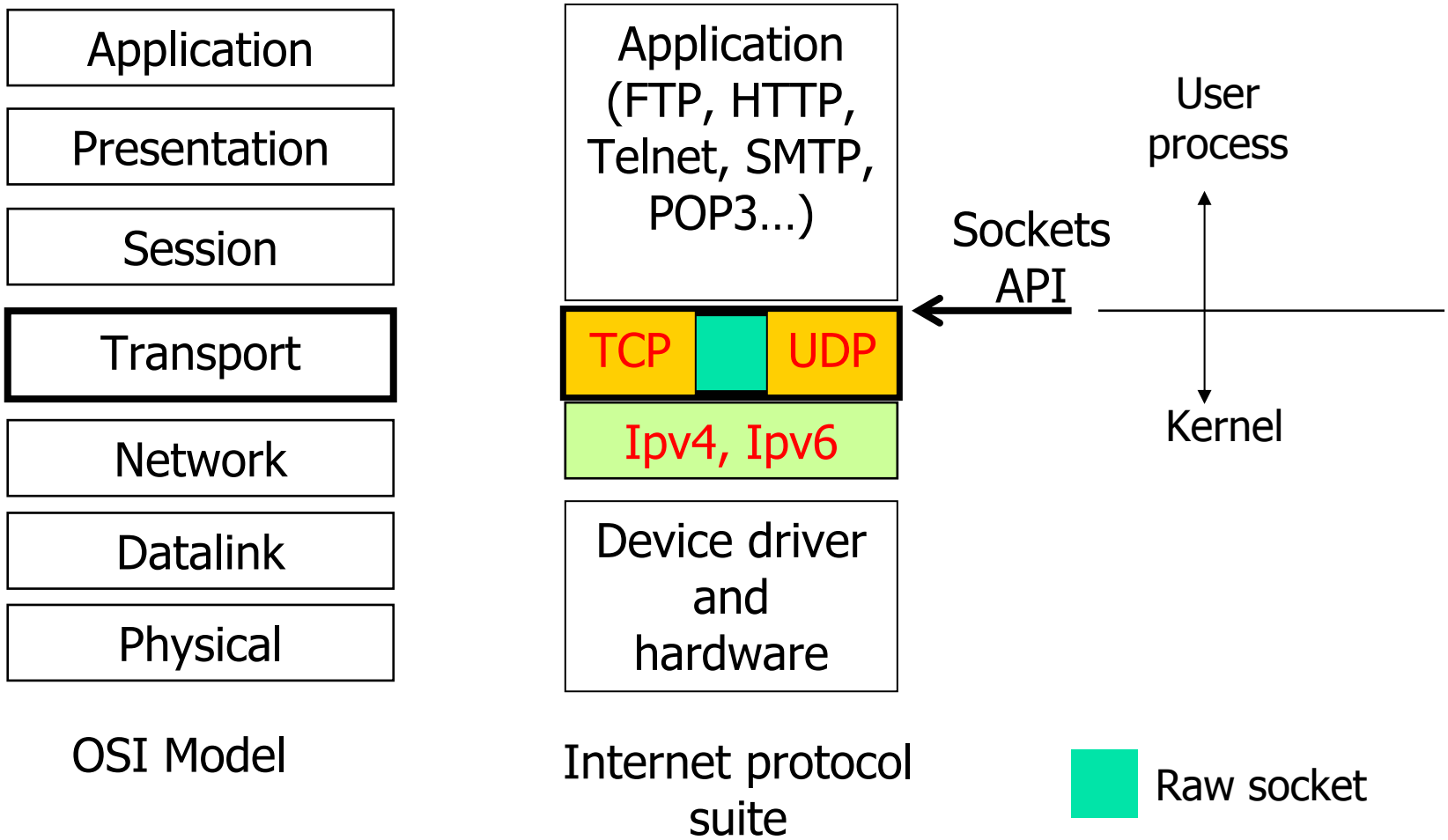
Ex: A Daytime client/server using socket API



Socket API Location in OS

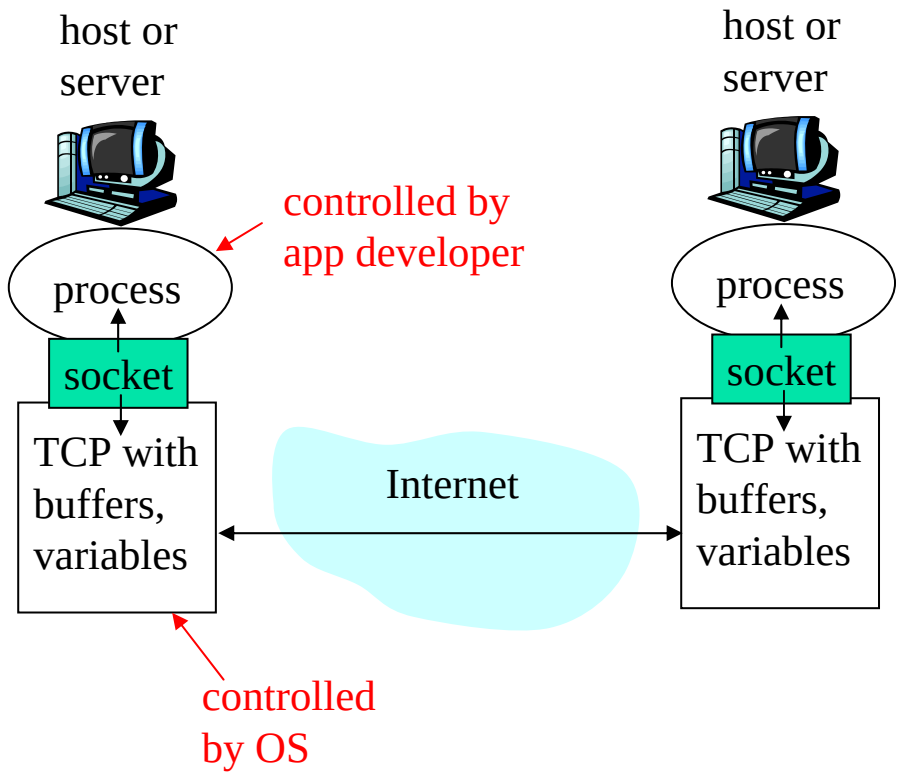


OSI Model


Application
Presentation
Session
Transport
Network
Datalink
Physical
OSI Model
Application (FTP, HTTP, Telnet, SMTP, POP3…)
TCP
UDP
Ipv4, Ipv6
Device driver and hardware
Internet protocol suite
Sockets API
User process
Kernel
Raw socket

Sockets

- process sends/receives messages to/from its socket
- socket analogous to door
  - sending process shoves message out door
  - transport infrastructure brings message to the door at receiving process



Socket API: (1) choice of transport protocol; (2) ability to fix a many parameters.

Addressing processes

- For a process to receive messages, it must have an *identifier*
- A host has a unique 32-bit IP address (IPv4)
- Q: does the IP address of the host on which the process runs suffice for identifying the process?
- Answer: No, many processes can be running on same host

- *Identifier* includes both the IP address and port numbers associated with the process on the host.
- Example port numbers:
  - HTTP server: 80
  - Mail server: 25

IP Address (IPv4)

- A unique identifier for each machine connected to an IP network.
  - 32 bit binary number
  - Represented as "dotted decimal" notation:
    - 4 decimal values, each representing 8 bits (octet), in the range 0 to 255.
- Example:
  - Dotted Decimal: 140 .179 .220 .200
  - Binary: 10001100.10110011.11011100.11001000

Ports

- Port - A 16-bit number to identify the application process that is a network endpoint.

- Reserved ports or well-known ports (0 to 1023)

  Standard ports for well-known applications.

  Telnet (23), ftp(21), http (80).

  See /etc/services file on any UNIX machine for listing of services on reserved ports. (Only root accessible).

- Ephemeral ports (1024-65535)

  For ordinary user-developed programs.

Associations

- A *half-association (or socket address)* is the triple:

  {*protocol, local-IP, local-port*}

  For example,

  {*tcp, 130.245.1.44, 23*}

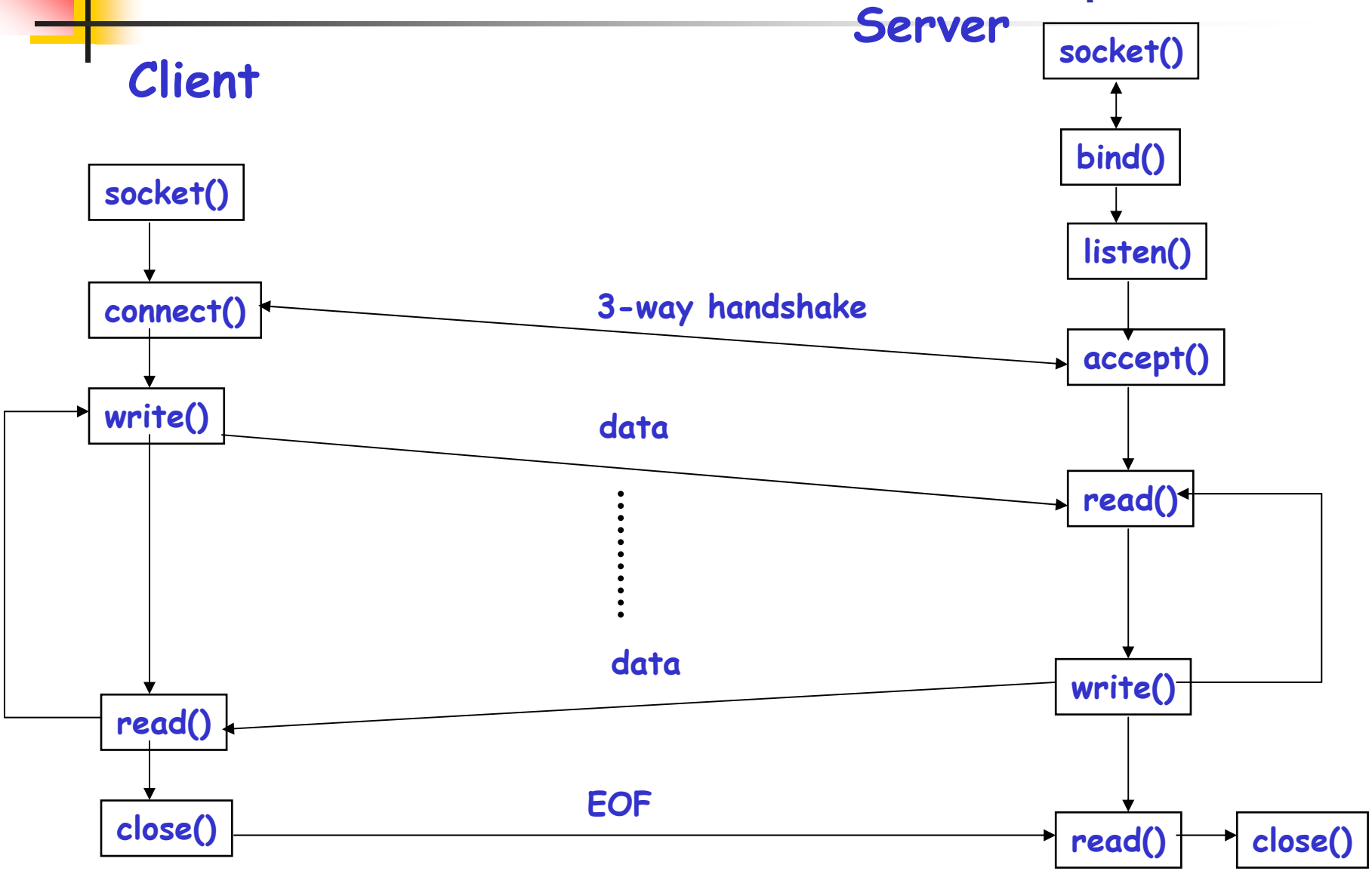
- An *association* is the 5-tuple that completely specifies the two end-points of a connection:

  {*protocol, local-IP, local-port, remote-IP, remote-port*}

  For example:

  {*tcp, 130.245.1.44, 23, 130.245.1.45, 1024*}

TCP client/server connection sequence



Simplifying error-handling

- **Create some wrappers for Socket functions BY**
  - **Check return code from socket function**
  - **Use err_sys() to display error mesage**

```
int Socket (int family, int type, int protocol) {
        int ret;

        if ( (ret = socket(family, type, protocol)) < 0)
              err_sys("socket error");

        return ret;
}
```

The Socket Structure

INET Address

```
struct in_addr {
   in_addr_t s_addr;  /* 32-bit IPv4 address */
}
```

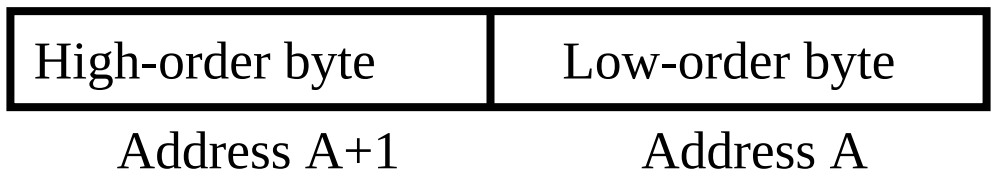
INET Socket

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

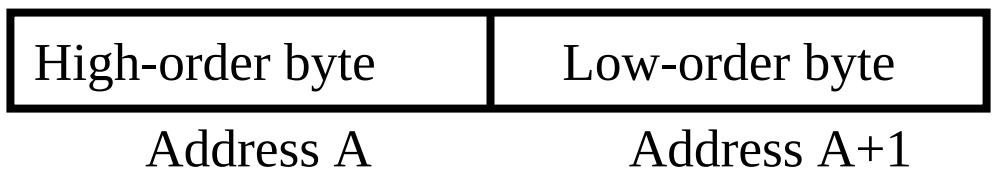
Network-byte ordering

Two ways to store 16-bit/32-bit integers

- Little-endian byte order (e.g. Intel)



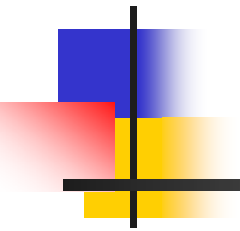
- Big-endian byte order (E.g. Sparc)



Network-byte ordering (cont.)

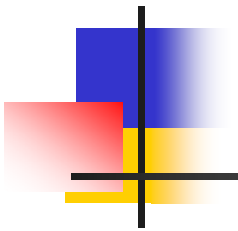
- How do two machines with different byte-orders communicate?
  - Using network byte-order
  - Network byte-order = big-endian order

- Conversion macros **(<netinet/in.h>)**
  - `uint16_t htons (uint16_t n)`
  - `uint32_t htonl (uint32_t n)`
  - `uint16_t ntohs (uint16_t n)`
  - `uint32_t ntohl (uint32_t n)`



Example of Client-Server Operation

A Simple Daytime Client and Server



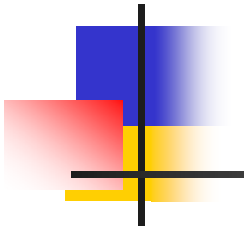
Daytime client

- Connects to a daytime server
- Retrieves the current date and time

```
% gettime  130.245.1.44
Thu Sept 05 15:50:00 2002
```

```
// A DAYTIME CLIENT SAMPLE – UNIX/LINUX VERSION
int main(int argc, char **argv) {
        int  sockfd, n;
        char recvline[MAXLINE + 1];
        struct sockaddr_in servaddr;

       if( argc != 2 )
              printf("Usage : gettime <IP address>"); exit(1);

        /* Create a TCP socket */
        if ( (sockfd = socket (AF_INET, SOCK_STREAM, 0)) < 0) {
              perror("socket"); exit(2);
        }

        /* Specify server's IP address and port */
        bzero (&servaddr, sizeof(servaddr));
        servaddr.sin_family = AF_INET;
        servaddr.sin_port = htons ( 13 ); /* daytime server port */

        if (inet_pton (AF_INET, argv[1], &servaddr.sin_addr) <= 0) {
               perror("inet_pton"); exit(3);
        }
```

```
        /* Connect to the server */
        if (  connect(  sockfd,
                        (struct sockaddr *) &servaddr,
                        sizeof(servaddr)) < 0 ) {
                    perror("connect"); exit(4);
        }

         /* Read the date/time from socket */
         while ( (n = read ( sockfd, recvline, MAXLINE)) > 0) {
                 recvline[n] = '\0';          /* null terminate */
                 printf("%s", recvline);
         }

         if (n < 0) {
               perror("read"); exit(5);
         }

         close ( sockfd );
}
// END OF DAYTIME CLIENT SAMPLE
```



Daytime Server

1. Waits for requests from Client
2. Accepts client connections
3. Send the current time
4. Terminates connection and goes back waiting for more connections.

```
/// A DAYTIME SERVER SAMPLE – UNIX/LINUX VERSION
1.  int main (int argc, char **argv) {
2.      int   listenfd, connfd;
3.      struct sockaddr_in servaddr, cliaddr;
4.      char buff[MAXLINE];
5.      time_t ticks;

6.      /* Create a TCP socket */
7.      listenfd = socket (AF_INET, SOCK_STREAM, 0);

8.      /* Initialize server's address and well-known port */
9.      bzero (&servaddr, sizeof(servaddr));
10.     servaddr.sin_family      = AF_INET;
11.     servaddr.sin_addr.s_addr = htonl (INADDR_ANY);
12.     servaddr.sin_port        = htons (13); /*daytime server*/

13.     /* Bind server's address and port to the socket */
14.     bind (listenfd, (struct sockaddr*) &servaddr,
15.           sizeof( servaddr) );
```

/// A DAYTIME SERVER SAMPLE – UNIX/LINUX VERSION (cont'd)

```
1.    /* Convert socket to a listening socket */
2.       listen (listenfd, 100);
3.       for ( ; ; ) {
4.          /* Wait for client connections and accept them */
5.          clilen = sizeof(cliaddr);
6.          connfd = accept( listenfd,
7.                      (struct sockaddr *)&cliaddr, &clilen);

8.          /* Retrieve system time */
9.          ticks = time(NULL);
10.         snprintf( buff, sizeof(buff),"%.24s\r\n",ctime(&ticks));

11.         /* Write to socket */
12.         write( connfd, buff, strlen(buff) );

13.         /* Close the connection */
14.         close( connfd );
15.      }
16.   }
```

Tài liệu tham khảo

- *UNIX Network Programming, Volume 2, Second Edition: Interprocess Communications*, Prentice Hall, 1999, ISBN 0-13-081081-9.