

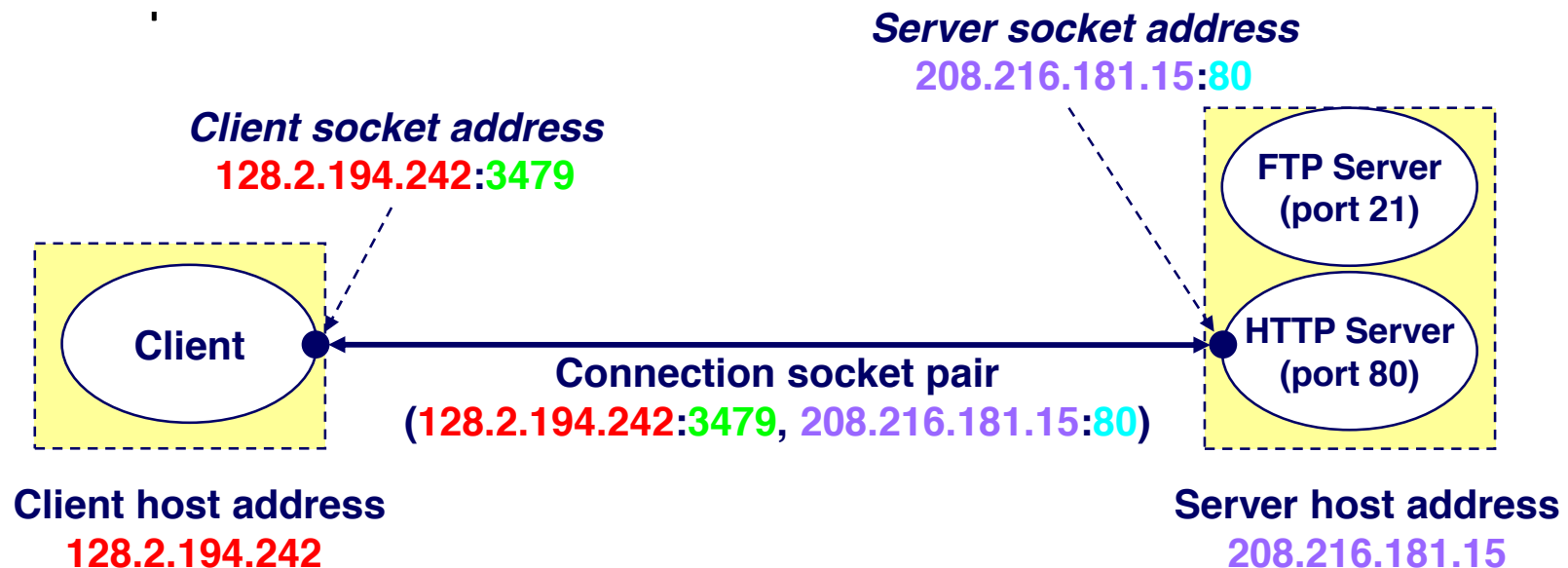
SOCKET INTRODUCTION

Content

- Socket
- Stream Socket
- Datagram Socket
- APIs for managing names and IP addresses
- Socket Address Structures

Socket

- What is a socket ?
- Sockets (in plural) are
 - application programming interface (API) at transport layer in TCP/IP stack
 - application may send and receive data through socket



Socket: how to use

- Setup socket
 - Where is the remote machine (IP address, hostname)
 - What service gets the data (port)
- Send and Receive
 - Designed just like any other I/O in unix
 - send – write
 - recv -- read
- Close the socket

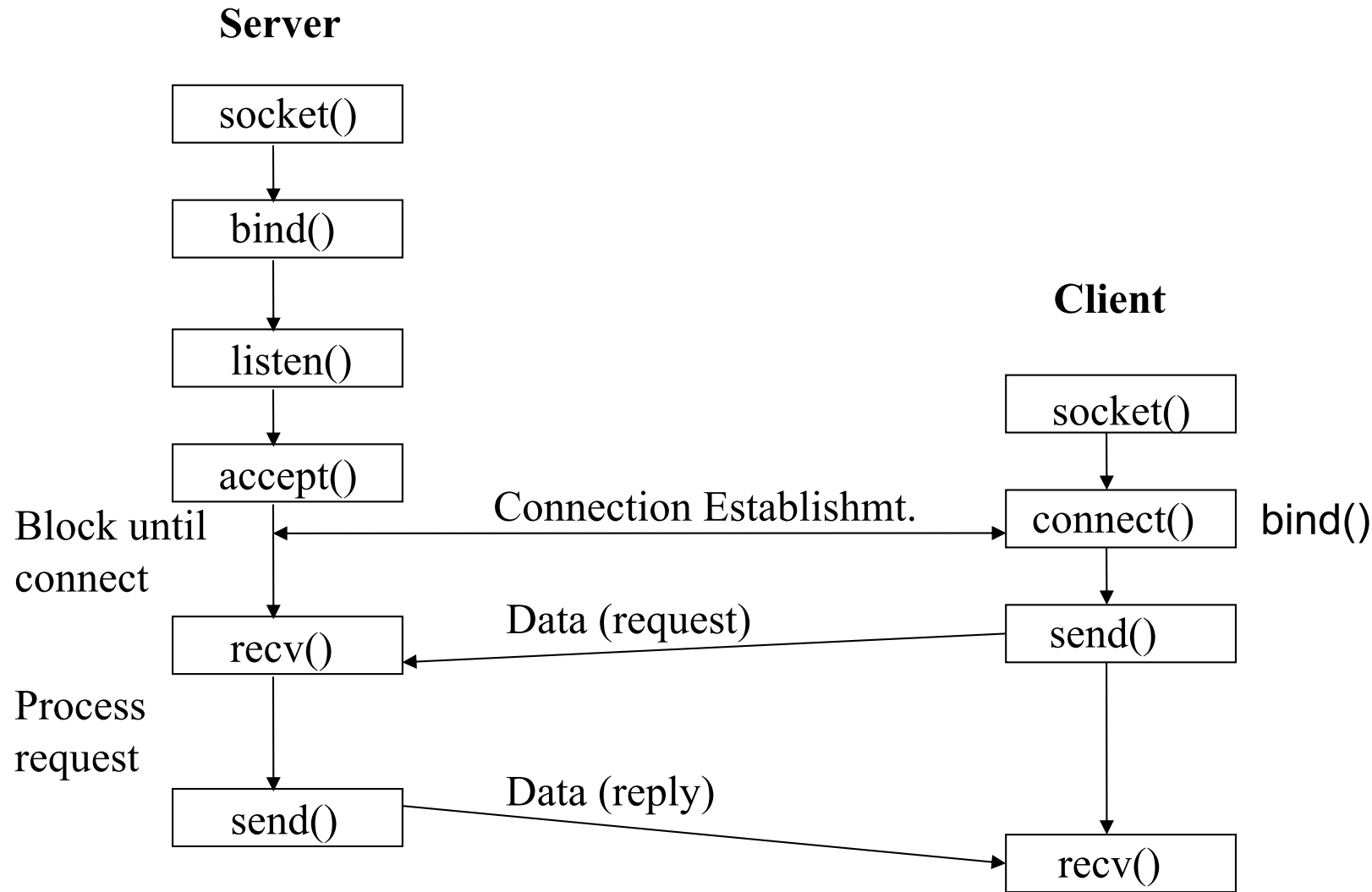
Socket types

- The main types of sockets in TCP/IP are
 - *stream sockets*
 - use **TCP** as the end-to-end protocol (with IP underneath) and thus provide a reliable byte-stream service
 - *datagram sockets*
 - use **UDP** (again, with IP underneath) and thus provide a ***best-effort*** datagram service
- Socket Address :
 - Combination of host name/Address IP + transport port

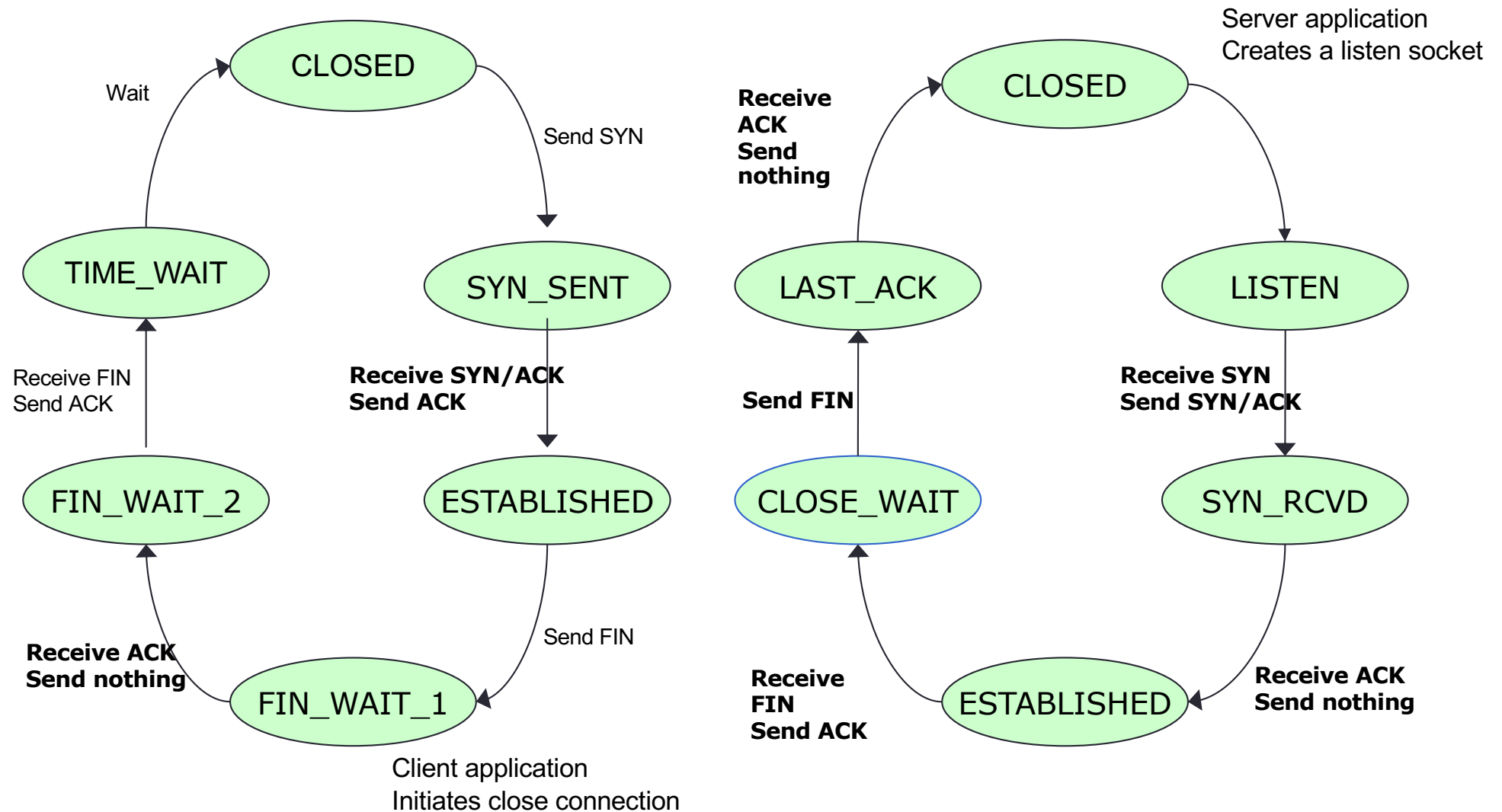
Stream sockets (TCP)

- TCP provides connections between clients and servers
- TCP also provides reliability :
 - Acknowledgment
 - Error control
 - Flow control
 - Congestion control
- TCP connection is full-duplex
 - Send and receive data over single connection.

Stream sockets (TCP): working flow



Life cycle of TCP connection



Stream Socket API in C

- **socket()**
 - creates a socket of a given domain, type, protocol (buy a phone)
 - Returns a file descriptor (called a socket ID)
- **bind()**
 - Assigns a name to the socket (get a telephone number)
 - Associate a socket with an IP address and port number (Eg : 192.168.1.1:80)
- **accept()**
 - server accepts a connection request from a client (answer phone)
 - There are basically three styles of using accept:
 - *Iterating server*: Only one socket is opened at a time and a single server process handle client requests one after the other.
 - *Forking server*: After an accept, a child process is forked off to handle the connection. Multiple connections can be handled simultaneously
 - *Concurrent single server*: a single process simultaneously wait on multiple open sockets, and the process is woken up only when new data arrives to one of the sockets.

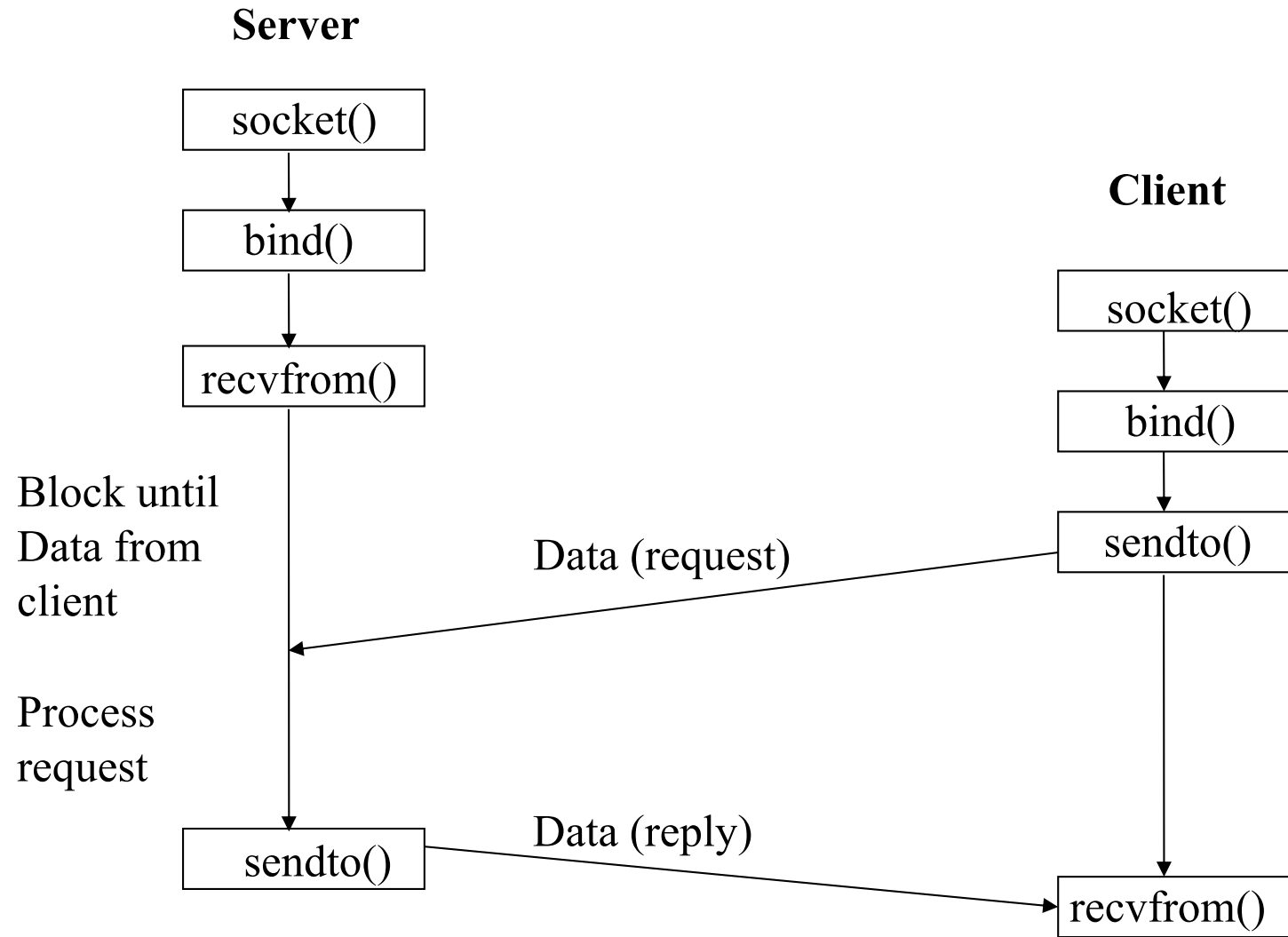
Stream Socket API in C

- **connect()**
 - Client requests a connection request to a server
 - This is the first of the client calls
- **send()**
 - write data to connection → to the parterner (speak)
- **recv()**
 - read data from connection → from the parterner (listen)
- **close()**
 - close a socket descriptor (end the call)

Datagram Socket (UDP)

- UDP is a simple transport-layer protocol
- If a datagram is errored or lost, it won't be automatically retransmitted (can process in application)
- UDP provides a *connectionless* service, as there need not be any long-term relationship between a UDP client and server

Datagram Socket (UDP)



Socket programming API in C

- Packages:
- Can be found under /usr/include
 - Ex: /usr/include/i386-linux-gnu
- `<stdio.h>`
 - input and output of basic C programs.
- `<sys/types.h>`
 - Contains definitions of data types used in system calls. These types are used in the next two include files.
- `<sys/socket.h>`
 - Includes definitions of structures needed for sockets.
- `<netinet/in.h>`
 - contains constants and structures needed for internet domain addresses.

Socket Address Structures

- Most socket functions require a pointer to a socket address structure as an argument, example:

```
int connect(int sockfd, const struct sockaddr *serv_addr, socklen_t addrlen);
```

- A Socket Address Structure is a struct which has information of a socket to create or connect with it
 - IP address
 - Transport port address
- Types of socket address structures
 - IPv4
 - IPv6

IPv4 socket address structure

```
#include <netinet/in.h>
```

```
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 */
    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 */
};
```

- **in_addr_t** is equivalent to the type **uint32_t**
- **uint8_t**, **uint16_t**, **uint32_t**: Integer type with a width of exactly 8, 16, 32 bits.

IPv6 socket address structure

```
#include <netinet/in.h>
```

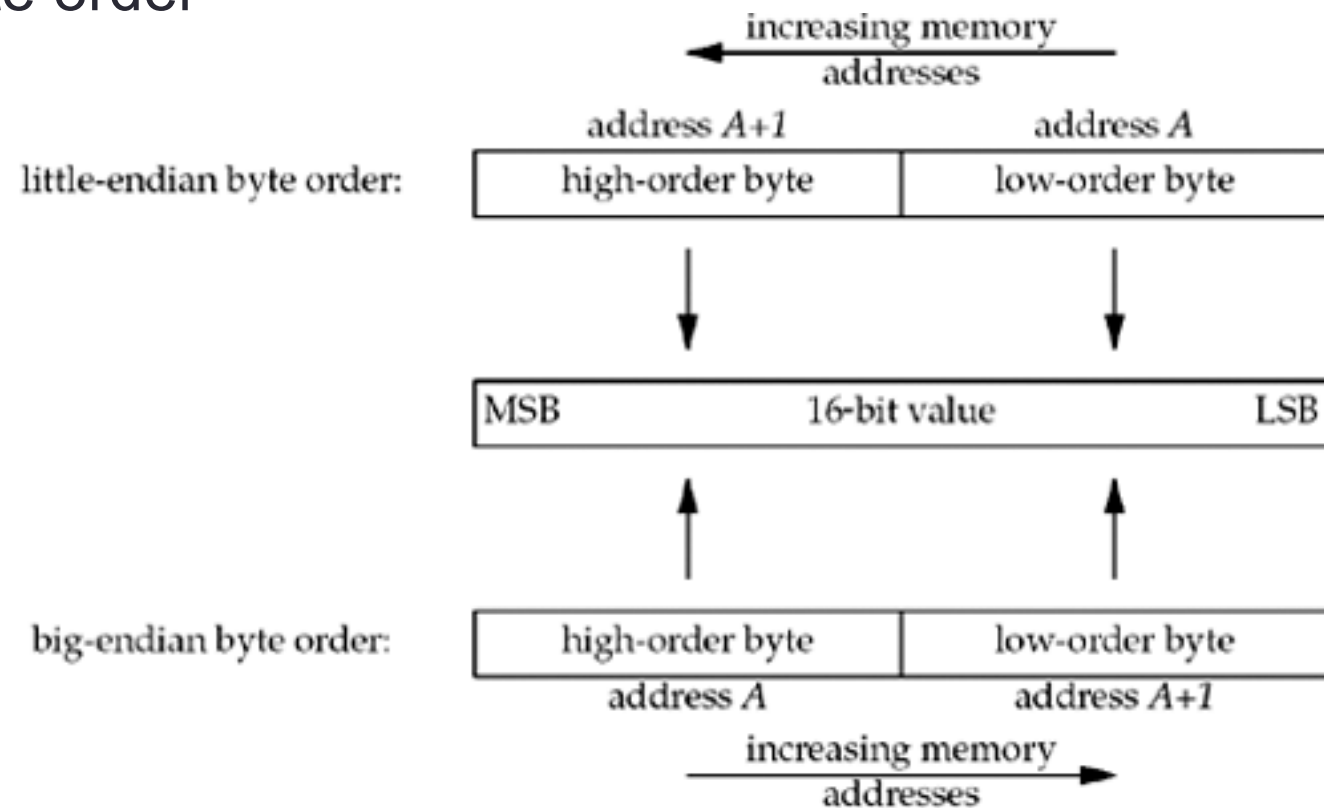
```
struct in6_addr {  
    uint8_t s6_addr[16]; /* 128-bit IPv6 address */  
                        /* network byte ordered */ };
```

```
#define SIN6_LEN      /* required for compile-time tests */
```

```
struct sockaddr_in6 {  
    uint8_t sin6_len; /* length of this struct */  
    sa_family_t sin6_family; /* AF_INET6 */  
    in_port_t sin6_port; /* transport layer port# */  
                        /* network byte ordered */  
    uint32_t sin6_flowinfo; /* flow information, undefined */  
    struct in6_addr sin6_addr; /* IPv6 address */  
                        /* network byte ordered */  
    uint32_t sin6_scope_id; /* set of interfaces for a scope */ };
```


Byte Ordering

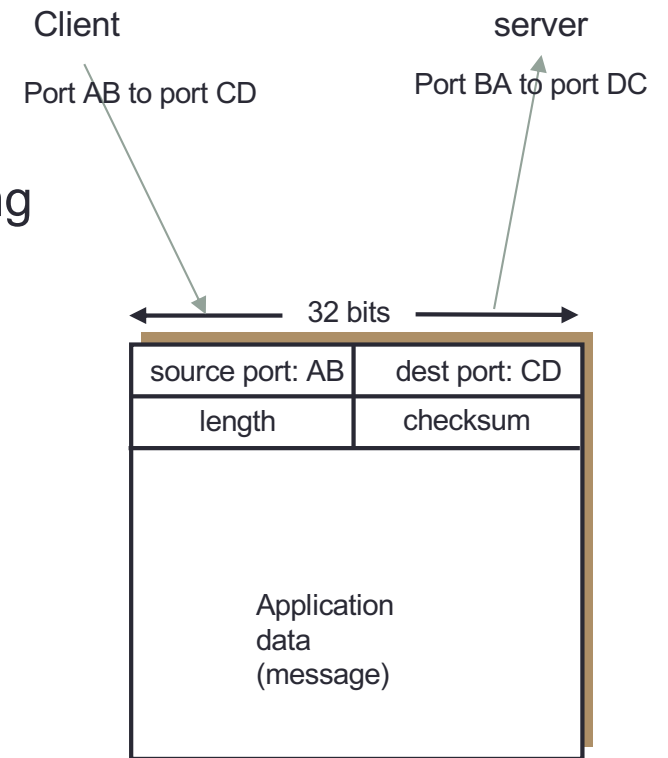
- There are two ways to store the two bytes in memory
 - little-endian byte order
 - big-endian byte order



Byte Ordering (cont)

- There is no standard byte ordering
 - The Internet protocols use big-endian byte ordering
 - Host order can be big- or little-endian
 - X86: little-endian, SPARC: big-endian
- Conversion
 - *htons()*, *htonl()*: host to network short/long
 - *ntohs()*, *ntohl()*: network order to host short/long
- What need to be converted?
 - Addresses, port why?
 - TCP/UDP writes source, destination port numbers in packets (2 bytes for each port)
 - Destination host reads address, TCP/UDP port number from TC/UDP packets and forward data to that port.
 - ➔ Need to make sure destination host read port number in the same order as it was written from the source
 - ➔ convert from local order to network order at source before send and convert back to local order at destination.

Case without network order conversion



htons(), htonl(), ntohs(), ntohl()

- Convert multi-byte integer types from host byte order to network byte order

```
#include <netinet/in.h>
```

```
uint32_t htonl(u_long hostlong); // host to network long  
uint16_t htons(u_short hostshort); // host to network short  
uint32_t ntohl(u_long netlong); // network to host long  
uint16_t ntohs(u_short netshort); // network to host short
```

- Each function returns the converted value.

APIs for managing names and IP addresses

- Auxiliary APIs:
 - All binary values are network byte ordered
 - *htons, htonl, ntohs, ntohl*: **byte ordering**
 - *inet_ntoa(), inet_aton(), inet_addr*: Convert IP addresses from a dots-and-number string (eg : 192.168.1.1) to a struct in_addr and back
 - *inet_pton, inet_ntop*: conversion of IP numbers between presentation and strings

List of Address and Name APIs

#include <sys/socket.h>

- **getprotobyname()**

- Retrieve the protocol name and number corresponding to a protocol name.

- **getprotobynumber()**

- Retrieve the protocol name and number corresponding to a protocol number.

- **getservbyname()**

- Retrieve the service name and port corresponding to a service name.

- **getservbyport()**

- Retrieve the service name and port corresponding to a port.

Address Access/Conversion Functions

- `struct hostent* gethostbyname (const char* hostname);`
 - Translate DNS host name to IP address (uses DNS)
- `struct hostent* gethostbyaddr (const char* addr, size_t len, int family);`
 - Translate IP address to DNS host name (not secure)
- `int gethostname (char* name, size_t namelen);`
 - Read host's name (use with **gethostbyname** to find local IP)
- `getprotobyname()`
 - Retrieve the protocol name and number corresponding to a protocol name.
- `getprotobynumber()`
 - Retrieve the protocol name and number corresponding to a protocol number.
- `getservbyname()`
 - Retrieve the service name and port corresponding to a service name.
- `getservbyport()`
 - Retrieve the service name and port corresponding to a port.

getservbyname()

- Get service information corresponding to a service name and protocol.

```
#include <netdb.h>
#include <sys/socket.h>
struct servent *getservbyname (const char *servname, const
                                char *protoname);
```

- servname
 - A pointer to a service name.
- protoname
 - An optional pointer to a protocol name.
 - If this is NULL, getservbyname() returns the first service entry for which the name matches the s_name or one of the s_aliases.
 - Otherwise getservbyname() matches both the name and the proto.
- Returns
 - non-null pointer if OK
 - NULL on error

```
struct servent *sptr;
sptr = getservbyname("ftp", "tcp");
```

struct servent

```
struct servent {  
    char *s_name;           /* official service name */  
    char **s_aliases;       /* alias list */  
    int s_port;             /* port number, network-byte order */  
    char *s_proto;          /* protocol to use */  
};
```

- s_name
 - Official name of the service.
- s_aliases
 - A NULL-terminated array of alternate names.
- s_port
 - The port number at which the service may be contacted. Port numbers are returned in network byte order.
- s_proto
 - The name of the protocol to use when contacting the service.

getservbyport

- Get service information corresponding to a port and protocol.

```
#include <netdb.h>
#include <sys/socket.h>
struct servent *getservbyport (int port, const char *protoname);
```

- port
 - The port for a service, in network byte order.
- protoname
 - An optional pointer to a protocol name.
 - If this is NULL, getservbyport() returns the first service entry for which the port matches the s_port.
 - Otherwise getservbyport() matches both the port and the proto.
- Return
 - non-null pointer if OK
 - NULL on error

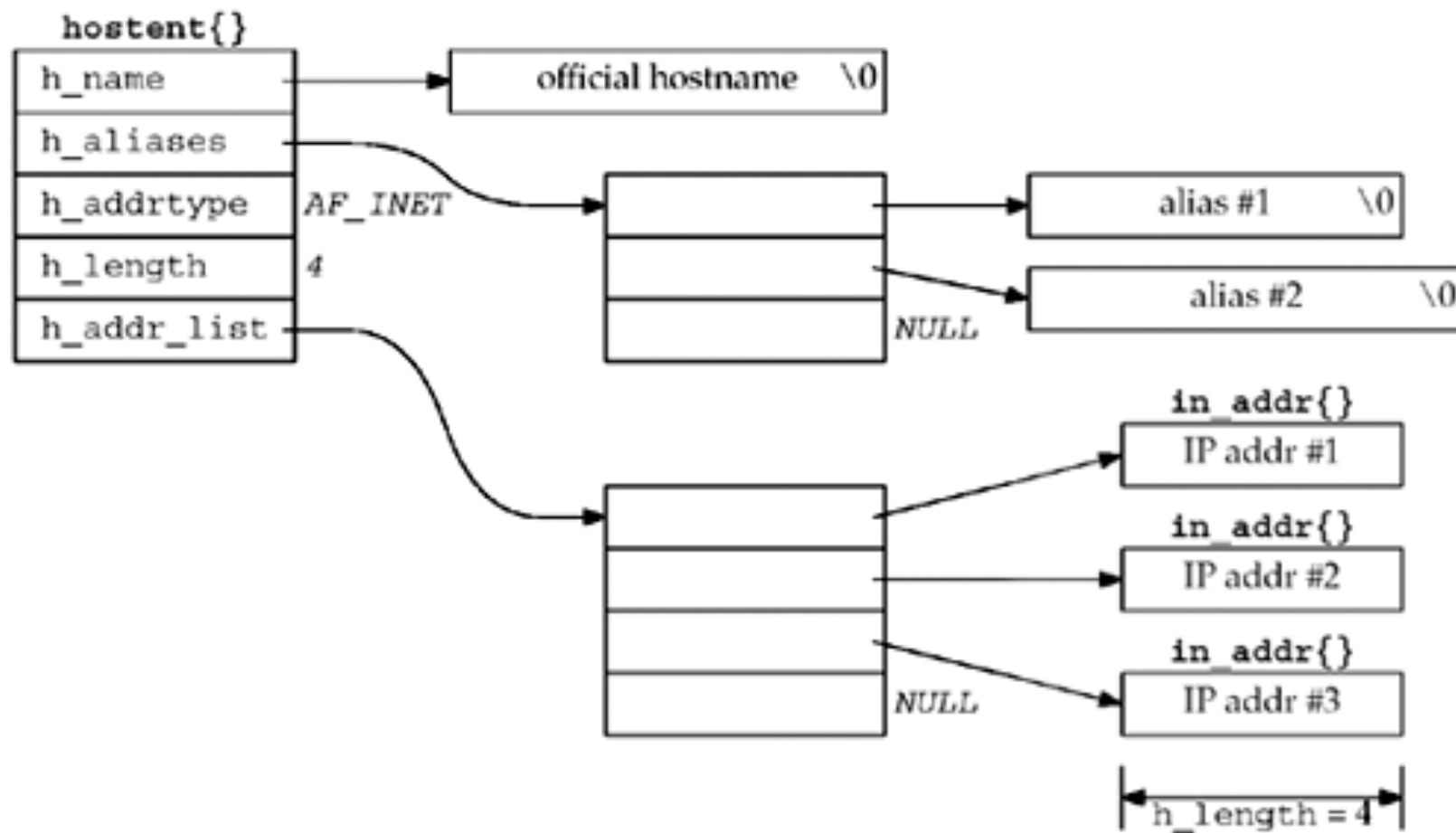
```
struct servent *sptr;
sptr = getservbyport (htons (53), "udp");
```

struct hostent

```
struct hostent {  
    char    *h_name;        /* official (canonical) name of host */  
    char    **h_aliases;    /* pointer to array of pointers to  
                             alias names */  
    int      h_addrtype;     /* host address type: AF_INET */  
    int      h_length;       /* length of address: 4 */  
    char    **h_addr_list;   /* ptr to array of ptrs with IPv4 addrs */  
};
```

- what is this struct hostent that gets returned?
- It has a number of fields that contain information about the host in question.

struct hostent



IP address conversion Functions

- **char* inet_ntoa (struct in_addr inaddr);**
 - Translate IP address to ASCII dotted-decimal notation (e.g., “128.32.36.37”); not thread-safe
- **in_addr_t inet_addr (const char* strptr);**
 - Translate dotted-decimal notation to IP address; returns -1 on failure, thus cannot handle broadcast value “255.255.255.255”
- **int inet_aton (const char* strptr, struct in_addr inaddr);**
 - Translate dotted-decimal notation to IP address; returns 1 on success, 0 on failure

inet_aton()

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

```
int inet_aton(const char *cp, struct in_addr *inp)
```

- Convert IP addresses from a dots-and-number string to a struct in_addr
- Return:
 - The value non-zero if the address is valid
 - The value 0 if the address is invalid

```
struct in_addr someAddr;  
if(inet_aton("10.0.0.1", someAddr))  
    printf("The address is valid");  
else printf ("The address is invalid");
```

inet_ntoa()

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

```
char *inet_ntoa(struct in_addr in);
```

- Convert IP addresses from a struct in_addr to a dots-and-number string
- Return: the dots-and-numbers string

```
struct in_addr someAddr;  
if(inet_aton("10.0.0.1", someAddr))  
    printf("The address is valid");  
else printf ("The address is invalid");  
char *addrStr;  
addrStr = inet_ntoa(someAddr);
```

New APIs for IPv6

- Those APIs only supports IPv4 but IPv6 will be replace IPv4 in the future, so we need APIs supporting IPv6
- They are
 - getaddrinfo
 - getnameinfo
- These APIs have replaced the IPv4 specific routines