

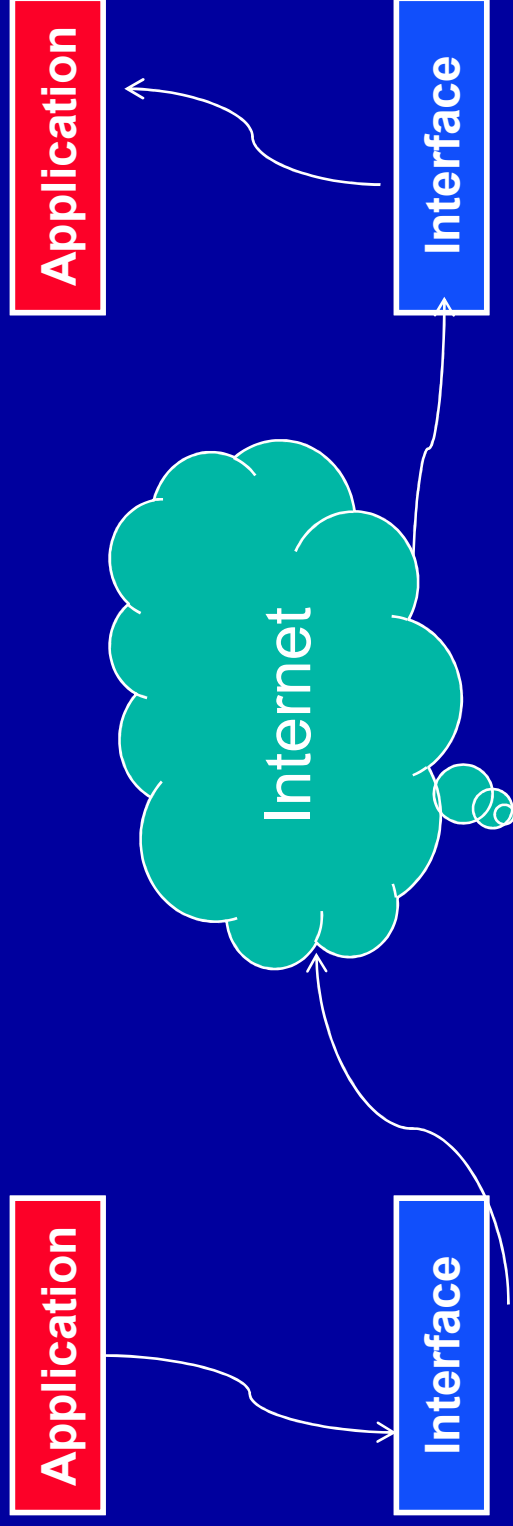
An Introduction to Network Programming

By

Kiran Kotla

TA, ECEN 602

How do two applications interact?



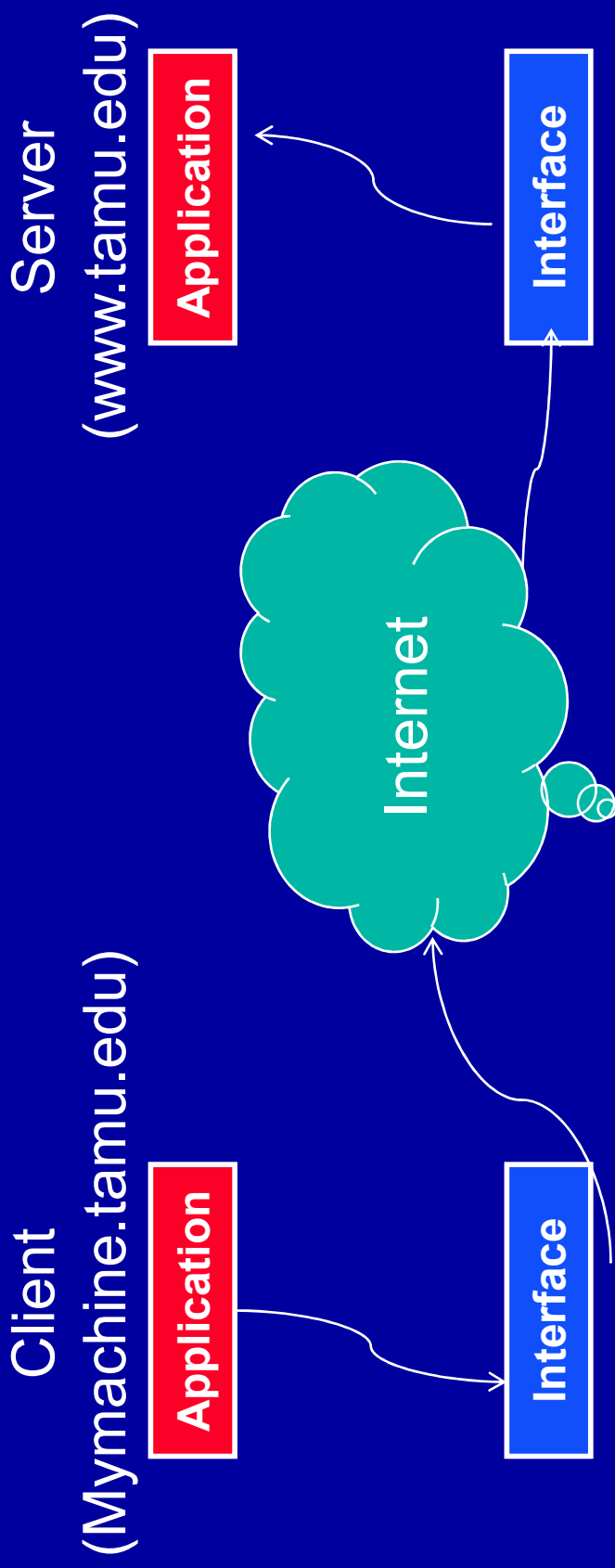
Example

- <http://www.tamu.edu>
- www.tamu.edu → Host Name → Address

This specifies one end point of communication

Suppose *mymachine.tamu.edu* wants to open this webpage. It needs to talk to the web server at www.tamu.edu

How do two applications interact?



Example (contd..)

Here the tag 'http' tells the browser to contact the web server at www.tamu.edu

In this type of set up:

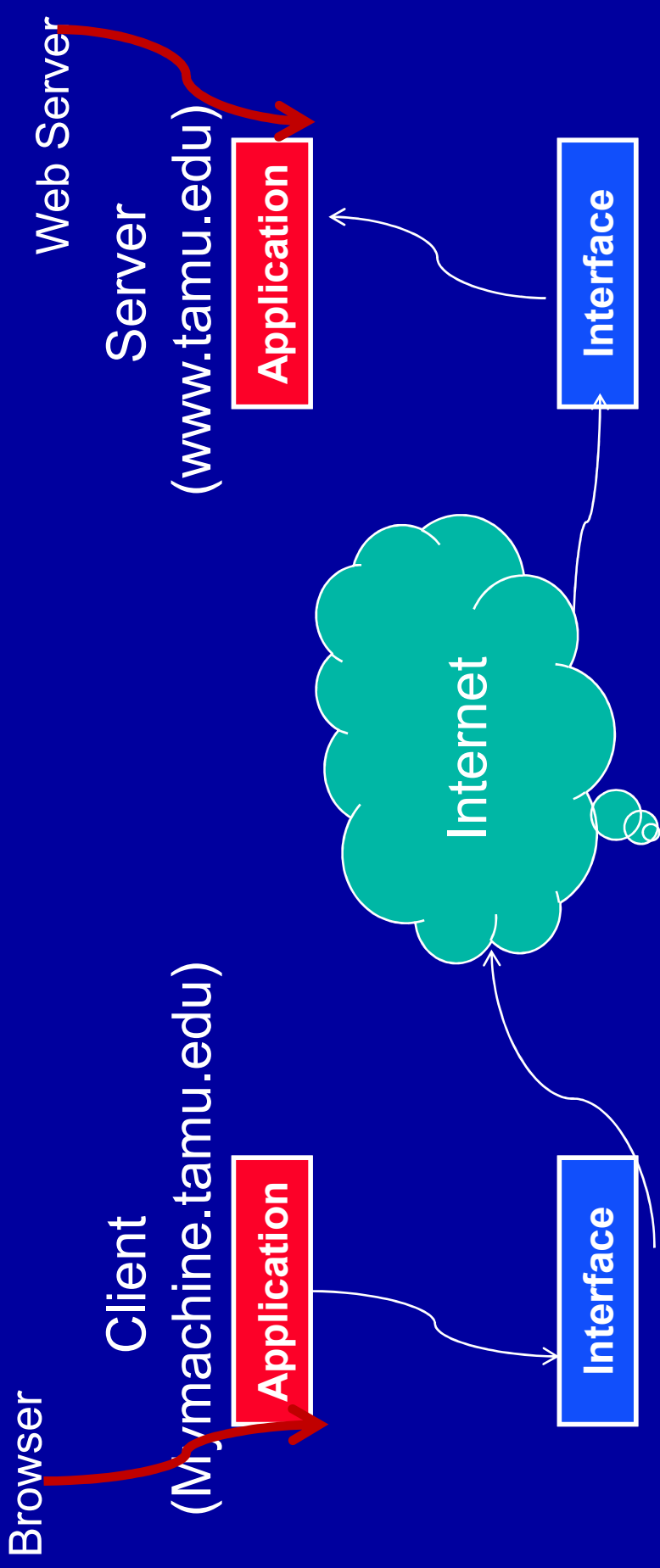
www.tamu.edu is the server machine

And

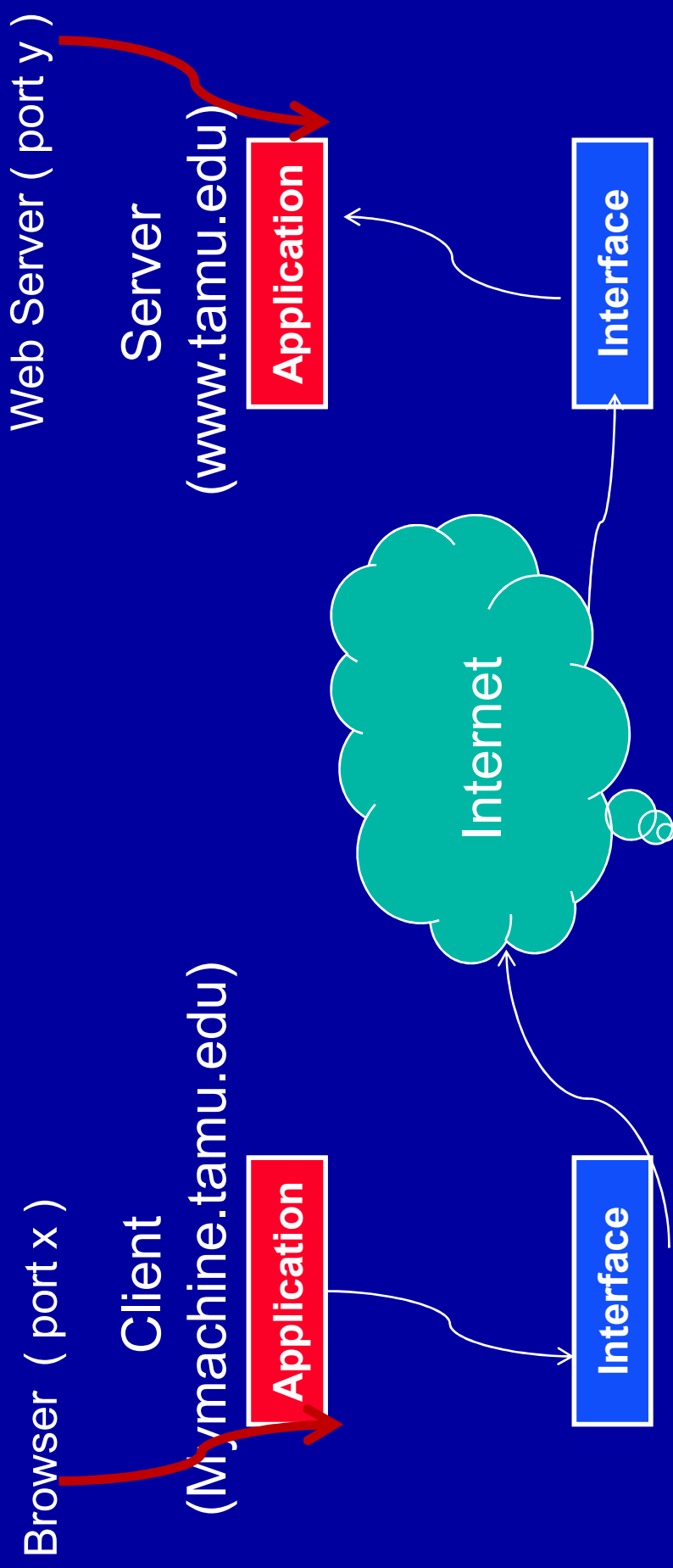
Mymachine.tamu.edu is the client machine

As there could be multiple applications running at the server machine, each application operates at a 'port number'.

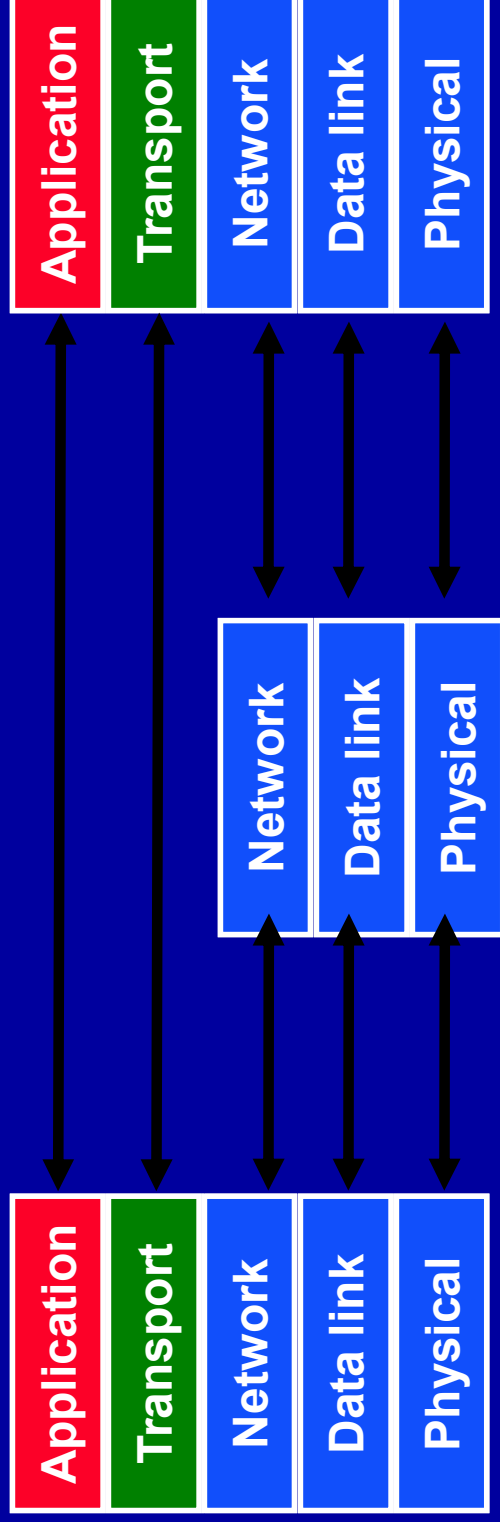
How do two applications interact?



How do two applications interact?

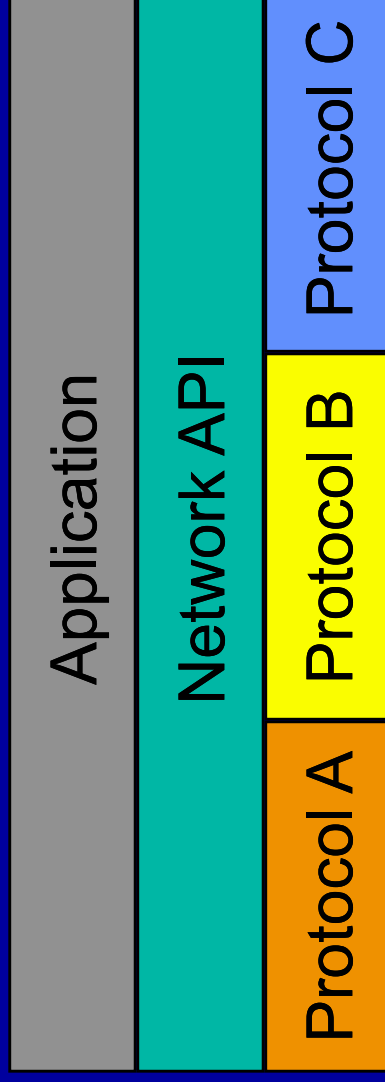


TCP/IP Network Stack



Network Application Programming Interface (API)

- The services provided (often by the operating system) that provide the interface between application and protocol software.



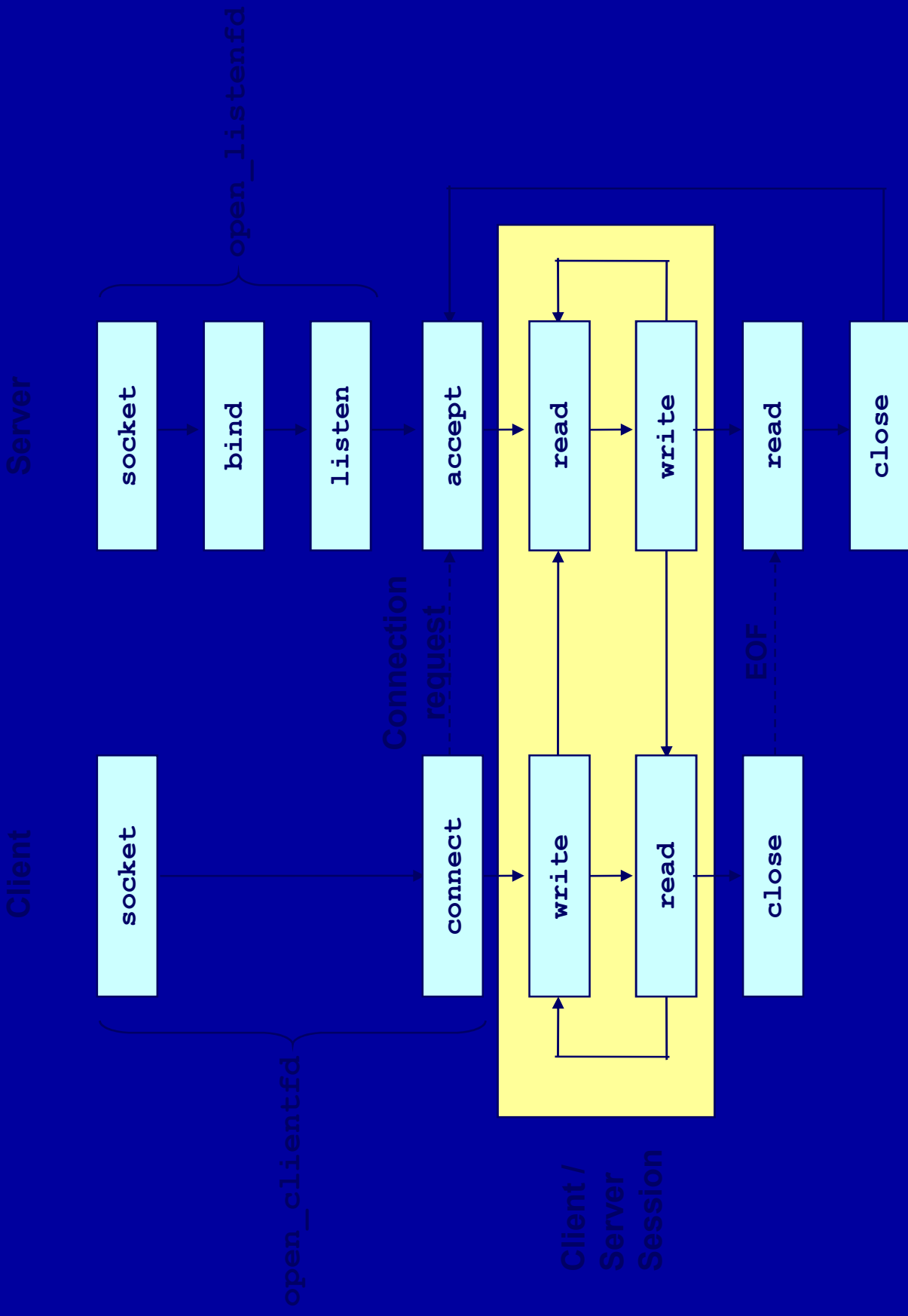
System calls

- General Use
 - `read()`
 - `write()`
 - `close()`

- Connection-oriented (TCP)
 - `connect()`
 - `listen()`
 - `accept()`

- Connectionless (UDP)
 - `send()`
 - `recv()`

Overview



Desired properties of Network API

- Generic Programming Interface.
- Support for message oriented and connection oriented communication.
- Operating System independence

Generic Programming Interface

- Support multiple communication protocol suites (families).
- Address (endpoint) representation independence.

TCP/IP

- TCP/IP does not include an API definition.

Functions needed:

- Specify local and remote communication endpoints
- Initiate a connection
- Wait for incoming connection
- Send and receive data
- Terminate a connection gracefully
- Error handling

Berkeley Sockets

- Generic:
 - support for multiple protocol families.
 - address representation independence

Socket

- A socket is an abstract representation of a communication endpoint.
- Need of Sockets:
 - establishing a connection
 - specifying communication endpoint addresses

Creating a Socket

```
int socket(int family, int type, int proto);
```

- `family` specifies the protocol family (**PF_INET** for TCP/IP).
- `type` specifies the type of service (**SOCK_STREAM**, **SOCK_DGRAM**).
- `protocol` specifies the specific protocol (usually 0, which means *the default*).

socket ()

- The `socket ()` system call returns a socket descriptor (small integer) or `-1` on error.
- `socket ()` allocates resources needed for a communication endpoint - but it does not deal with endpoint addressing.

Specifying an Endpoint Address

- Remember that the sockets API is generic.
- There must be a generic way to specify endpoint addresses.
- TCP/IP requires an IP address and a port number for each endpoint address.
- Other protocol suites (families) may use other schemes.

Socket Descriptor Data Structure

Descriptor Table

0	•
1	•
2	•
3	•
4	•
	⋮

Family: AF_INET
Service: SOCK_STREAM
Local IP: 111.22.3.4
Remote IP: 123.45.6.78
Local Port: 2249
Remote Port: 3726

Some data types that are used

<code>sa_family_t</code>	address family
<code>socklen_t</code>	length of struct
<code>in_addr_t</code>	IPv4 address
<code>in_port_t</code>	IP port number

Generic socket addresses

Used by kernel



```
struct sockaddr {  
    uint8_t    sa_len;  
    sa_family_t sa_family;  
    char       sa_data[14];  
};
```

- `sa_family` specifies the address type.
- `sa_data` specifies the address value.

AF_INET

For AF_INET we need:

- 16 bit port number
- 32 bit IP address

IPv4



struct sockaddr_in (IPv4)

```
struct sockaddr_in {
    uint8_t      sin_len;
    sa_family_t  sin_family;
    in_port_t    sin_port;
    struct in_addr sin_addr;
    char         sin_zero[8];
};
```

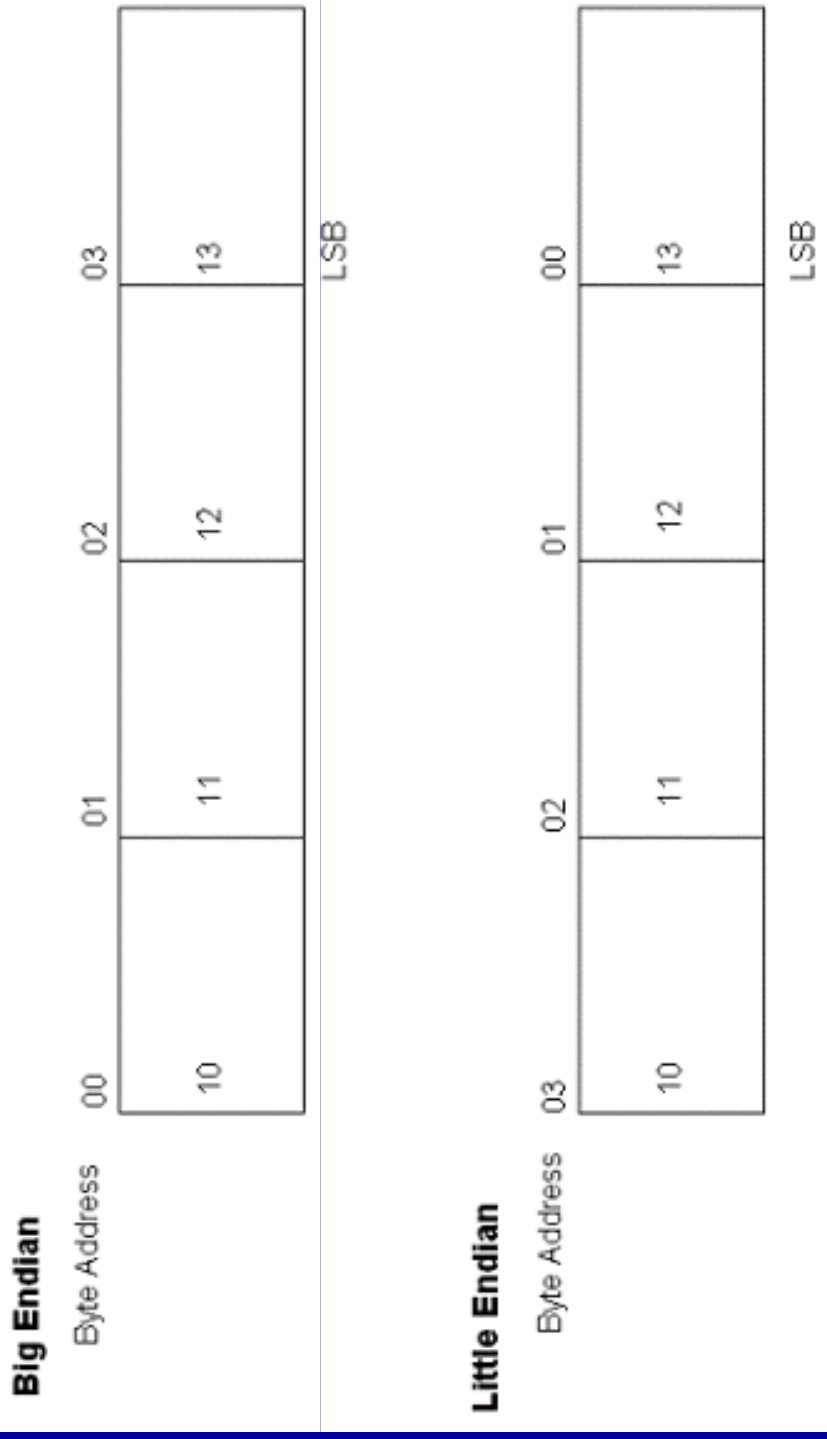
A special kind of sockaddr structure

Network Byte Order

Bytes travel through the network in a stream of bytes.

However, the architectures of the two end hosts could be different, i.e., one is little endian and the other is big endian

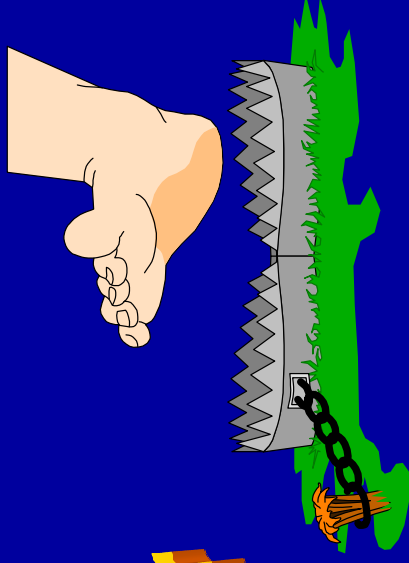
Network Byte Order (Contd..)



Network Byte Order (Contd..)

- All values stored in a `sockaddr_in` must be in network byte order.
 - `sin_port` a TCP/IP port number.
 - `sin_addr` an IP address.

**Common Mistake:
Ignoring Network Byte Order**



Network Byte Order Functions

'h' : host byte order **'n'** : network byte order
's' : short (16bit) **'l'** : long (32bit)

```
uint16_t htons(uint16_t);  
uint16_t ntohs(uint16_t);  
  
uint32_t htonl(uint32_t);  
uint32_t ntohl(uint32_t);
```

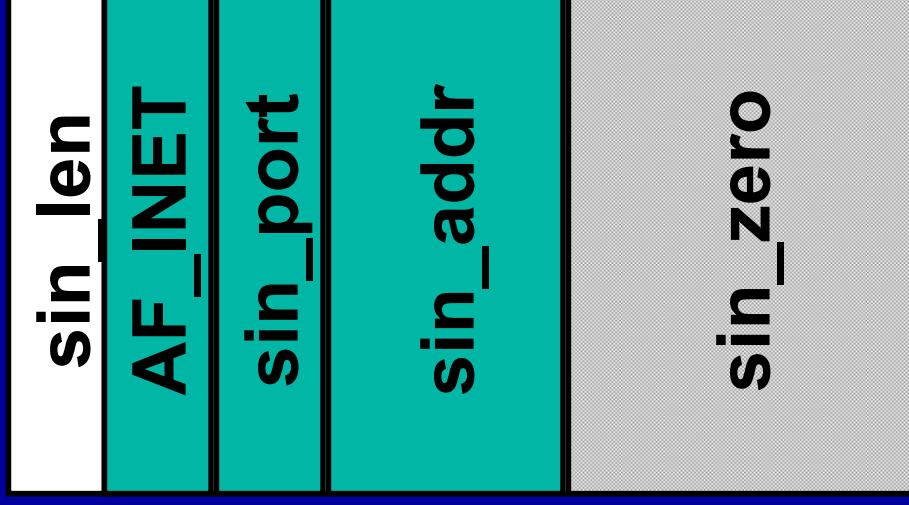
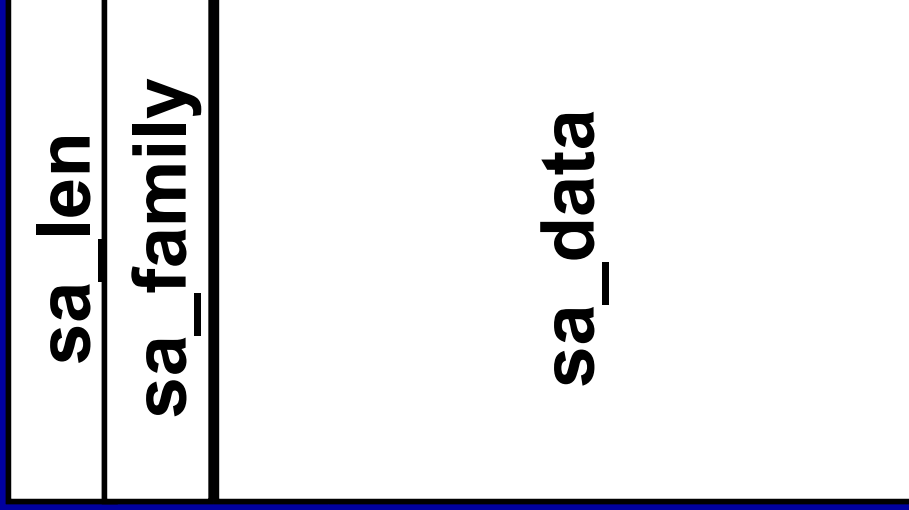
TCP/IP Addresses

- We don't need to deal with `sockaddr` structures since we will only deal with a real protocol family.
- We can use `sockaddr_in` structures.

BUT: The C functions that make up the sockets API expect structures of type `sockaddr`.

sockaddr

sockaddr_in



Assigning an address to a socket

- The `bind()` system call is used to assign an address to an existing socket.

```
int bind( int sockfd,  
         const struct sockaddr *myaddr,  
         int addrlen) ;  
const!
```



- `bind` returns 0 if successful or -1 on error.

bind()

- calling `bind()` assigns the address specified by the `sockaddr` structure to the socket descriptor.
- You can give `bind()` a `sockaddr_in` structure:

```
bind( mysock,  
      (struct sockaddr*) &myaddr,  
      sizeof(myaddr) );
```

bind() Example

```
int mysock, err;
struct sockaddr_in myaddr;

mysock = socket(PF_INET, SOCK_STREAM, 0);
myaddr.sin_family = AF_INET;
myaddr.sin_port = htons( portnum );
myaddr.sin_addr = htonl( ipaddress);

err=bind(mysock, (sockaddr *) &myaddr,
        sizeof(myaddr));
```

Uses for `bind()`

- There are a number of uses for `bind()`:
 - Server would like to bind to a well known address (port number).
 - Client can bind to a specific port.
 - Client can ask the O.S. to assign *any available* port number.

Client Port

- Clients typically don't care what port they are assigned.
- When you call bind you can tell it to assign you any available port:

```
myaddr.port = htons(0);
```

Accepting connections

- How can you find out what your IP address is so you can tell `bind()` ?
- There is no realistic way for you to know the right IP address to give `bind()` - what if the computer has multiple network interfaces?
- specify the IP address as: `INADDR_ANY`, this tells the OS to take care of things.

IPv4 Address Conversion

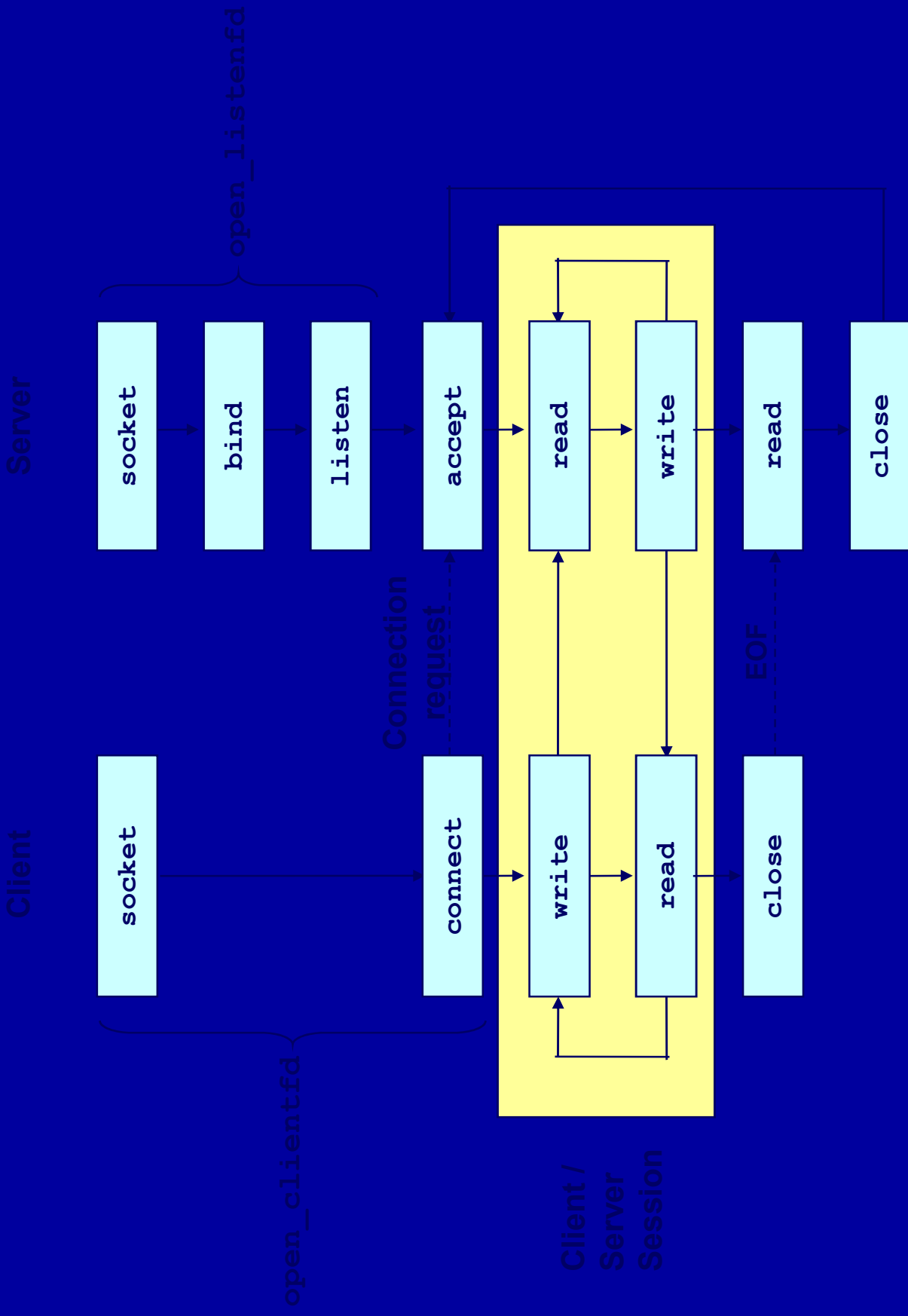
```
int inet_aton( char *, struct in_addr *);
```

Convert ASCII dotted-decimal IP address to network byte order 32 bit value. Returns 1 on success, 0 on failure.

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

Convert network byte ordered value to ASCII dotted-decimal (a string).

Overview



Lets put the server together...

```
struct sockaddr_in saddr, caddr;
int sockfd, clen, isock;
unsigned short port = 80;

if((sockfd=socket(AF_INET, SOCK_STREAM, 0) < 0) { // from back a couple slides
printf("Error creating socket\n");
...
}

memset(&saddr, '\0', sizeof(saddr)); // zero structure out
saddr.sin_family = AF_INET; // match the socket() call
saddr.sin_addr.s_addr = htonl(INADDR_ANY); // bind to any local address
saddr.sin_port = htons(port); // specify port to listen on

if((bind(sockfd, (struct sockaddr *) &saddr, sizeof(saddr)) < 0) { // bind!
printf("Error binding\n");
...
}

if(listen(sockfd, 5) < 0) { // listen for incoming connections
printf("Error listening\n");
...
}

clen=sizeof(caddr)
if((isock=accept(sockfd, (struct sockaddr *) &caddr, &clen)) < 0) { // accept one
printf("Error accepting\n");
...
}
```

Piecing the Client Together

```
struct sockaddr_in saddr;
struct hostent *h;
int sockfd, connfd;
unsigned short port = 80;

if((sockfd=socket(AF_INET, SOCK_STREAM, 0) < 0) { // from back a couple slides
printf("Error creating socket\n");
...
}

if((h=gethostbyname("www.slashdot.org")) == NULL) { // Lookup the hostname
printf("Unknown host\n");
...
}

memset(&saddr, '\0', sizeof(saddr)); // zero structure out
saddr.sin_family = AF_INET; // match the socket() call
memcpy((char *) &saddr.sin_addr.s_addr, h->h_addr_list[0], h->h_length); // copy the address
saddr.sin_port = htons(port); // specify port to connect to

if((connfd=connect(sockfd, (struct sockaddr *) &saddr, sizeof(saddr)) < 0) { // connect!
printf("Cannot connect\n");
...
}
```

Thank you!

Please refer to the course website for
more information

http://www.ece.tamu.edu/~reddy/ee602_07.html