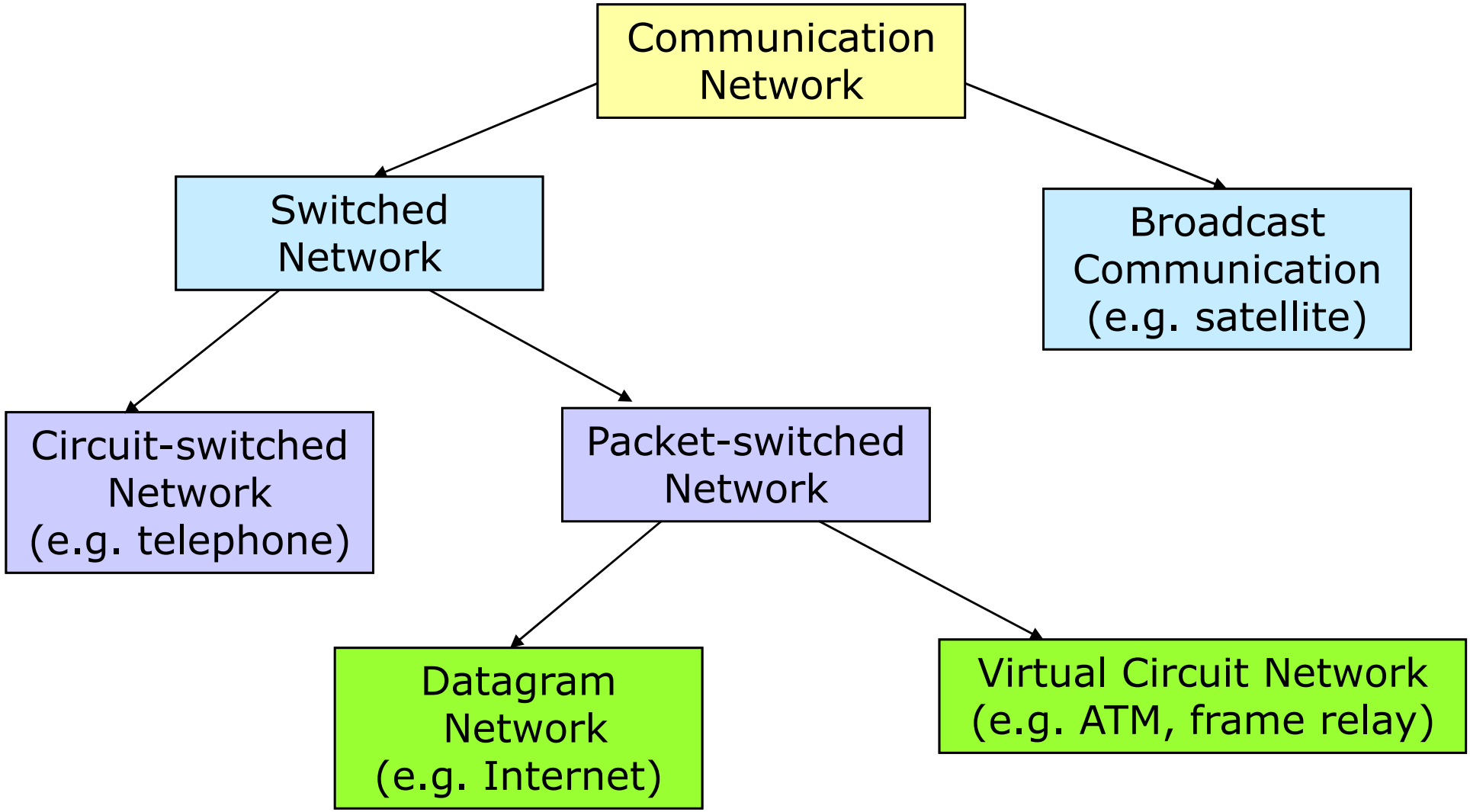
Sockets

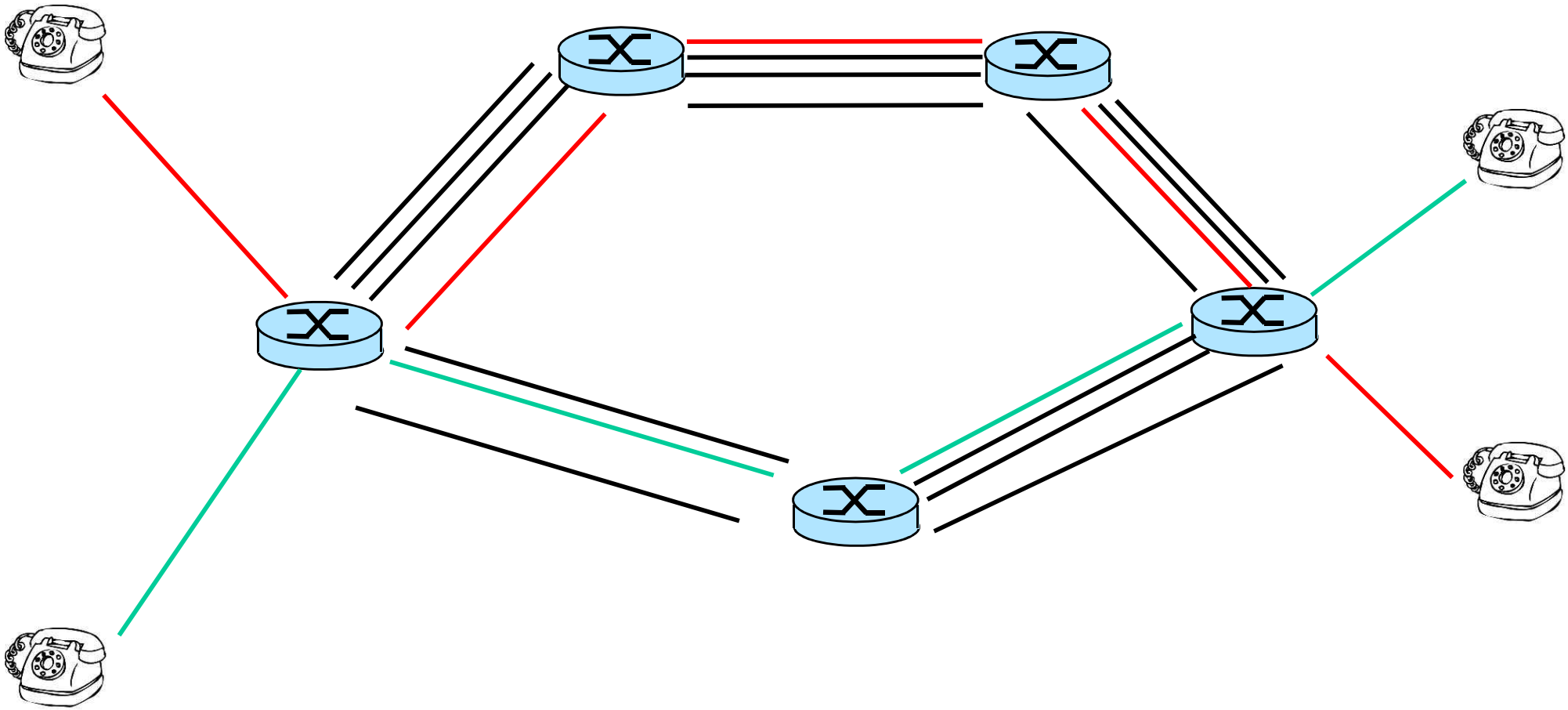
경희대학교 컴퓨터공학과

조 진 성

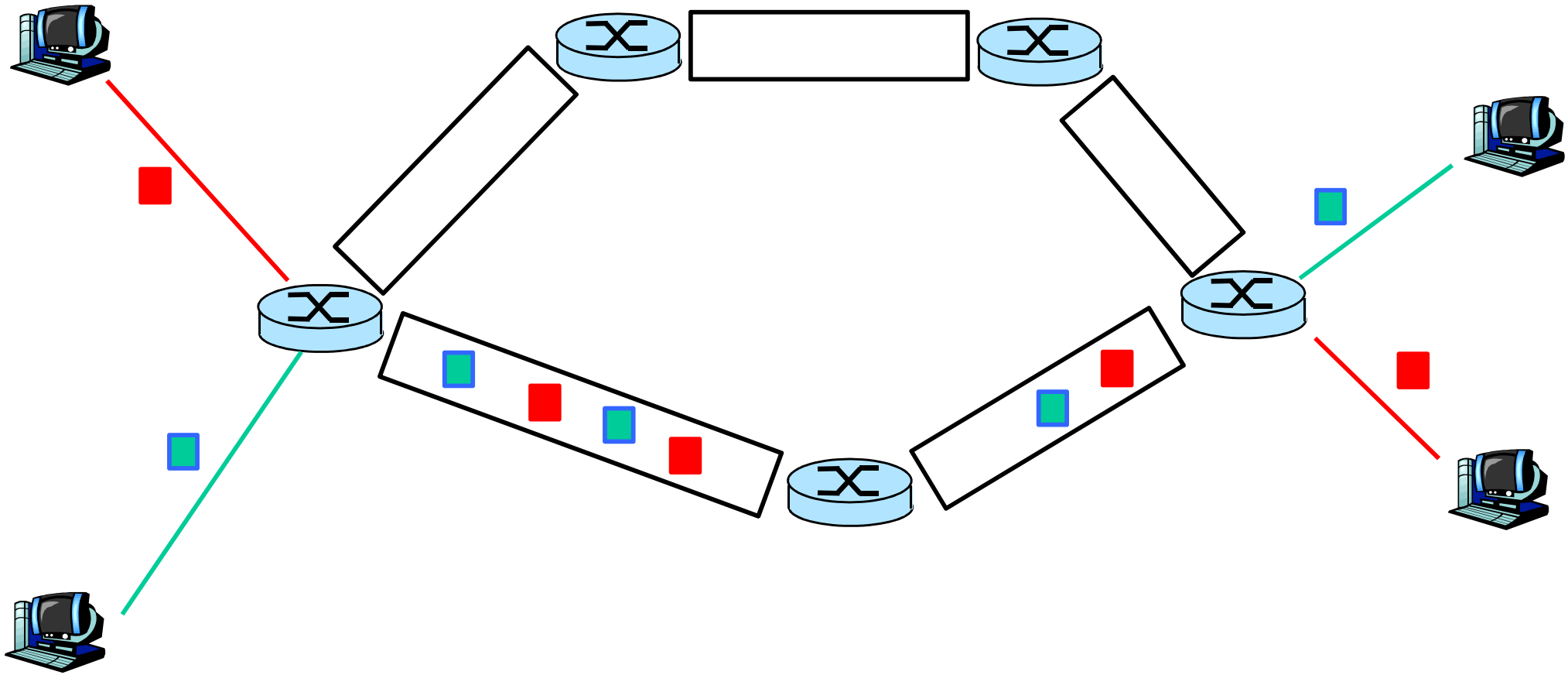
Computer & Communication Networks



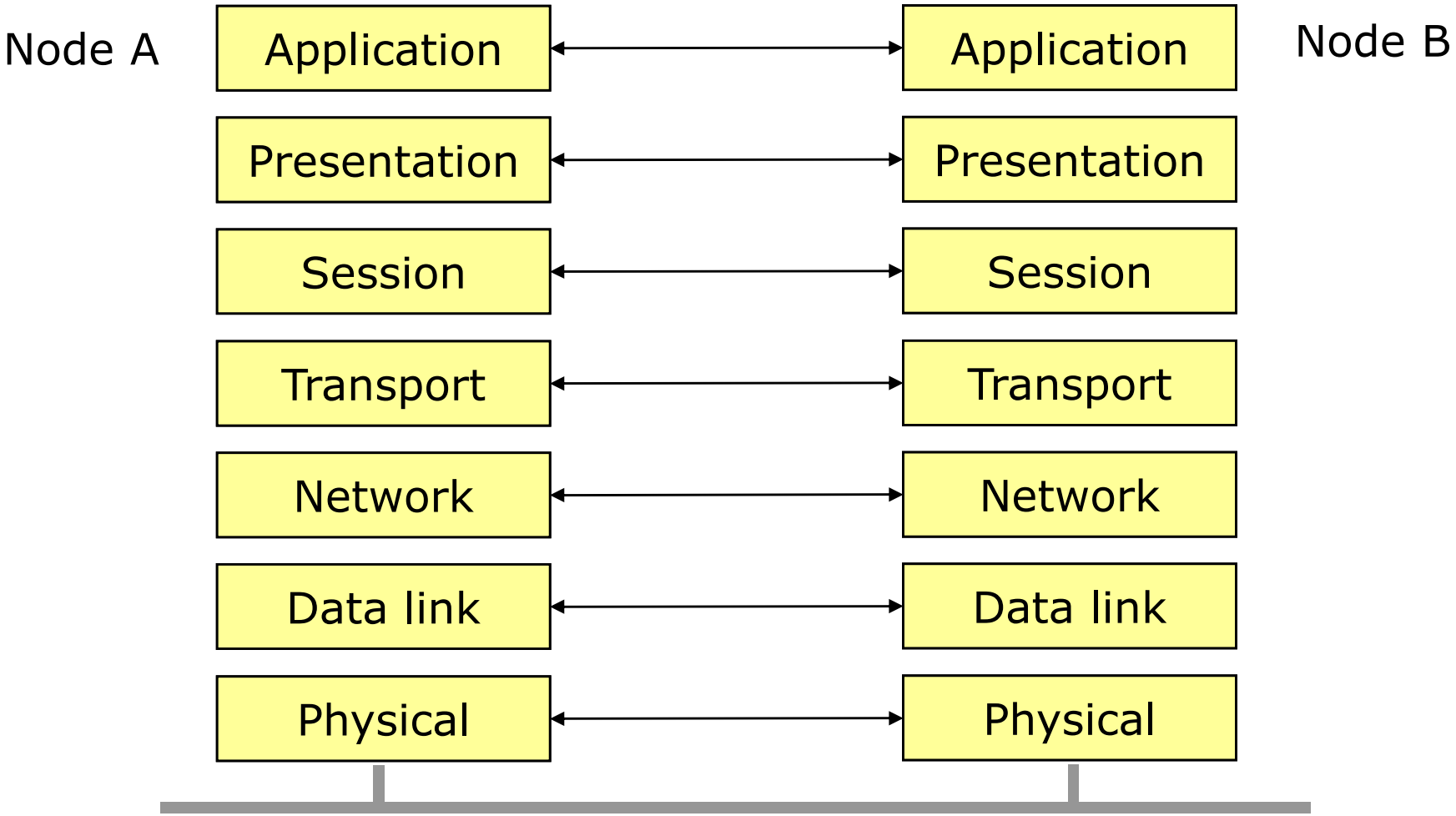
Circuit Switched Network



Packet Switched Network



OSI 7-Layer Reference Model



Why Layering?

Explicit structure allows identification, relationship of complex system's pieces

Modularization eases maintenance, updating of system

- ✓ Change of implementation of layer's service transparent to rest of system

Each layer abstracts the services of various lower layers, providing a uniform interface to higher layers

- ✓ Each layer needs to know how to interpret a packet's payload and how to use services of a lower layer

Layering considered harmful?

Internet Protocol Layers

Application

- ✓ Supporting network applications
  - HTTP, SMTP, POP3, TELNET, FTP, SSH, DNS, SNMP, …

Transport

- ✓ Data transfer between end-to-end processes
  - TCP, UDP

Network

- ✓ Routing of datagrams from source node to destination node
- ✓ Data transfer between end-to-end nodes
  - IP, routing protocols

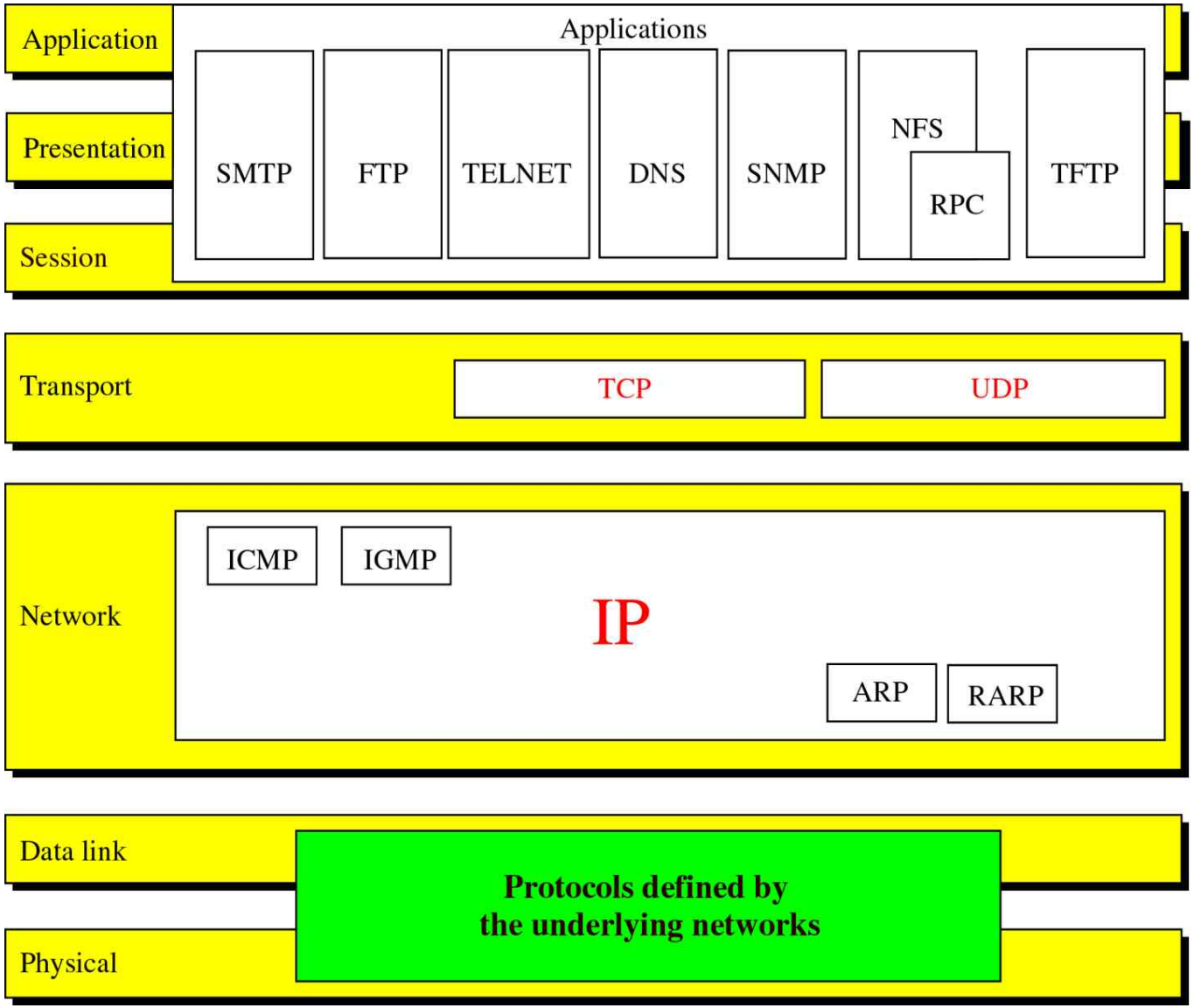
Data link

- ✓ Data transfer between neighboring (adjacent) nodes
  - Ethernet, Wireless LAN(WLAN)

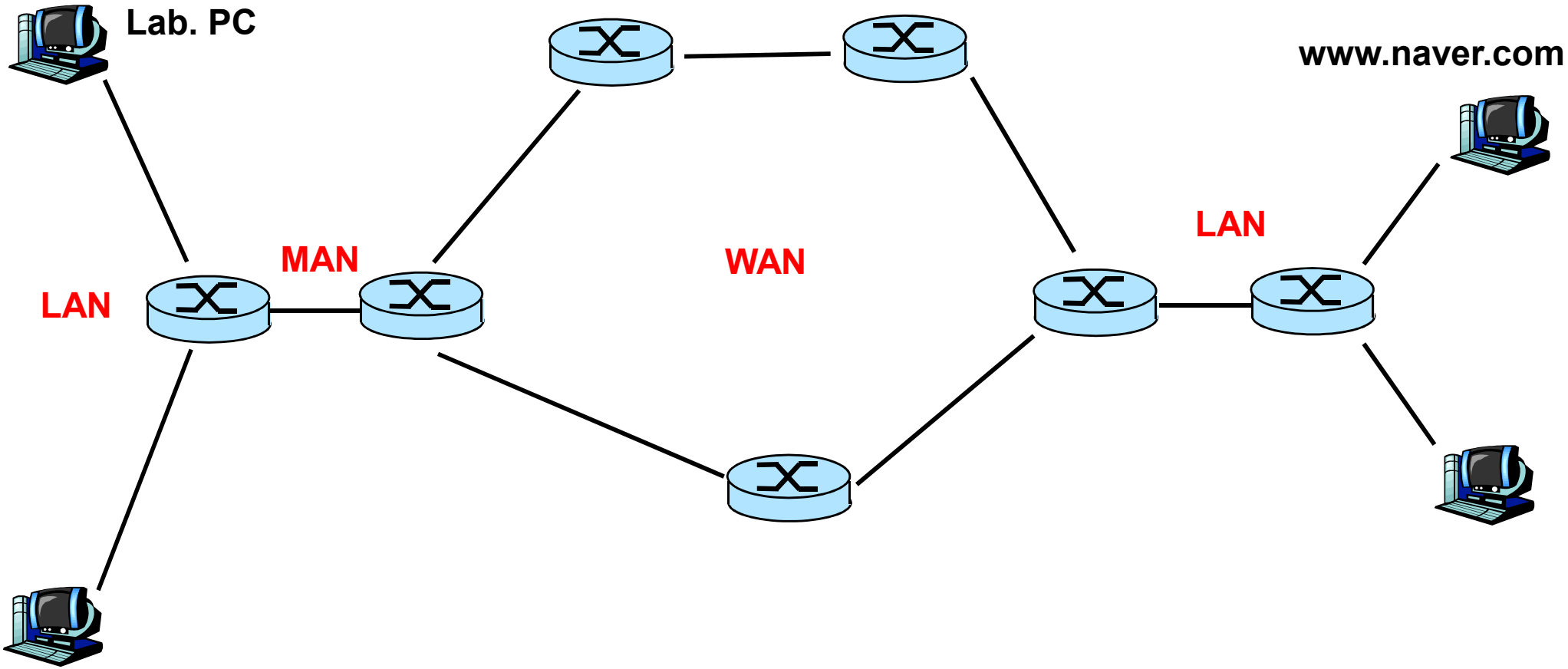
Physical

- ✓ Bits on the wire/wireless

Internet Protocol Layers (Cont'd)



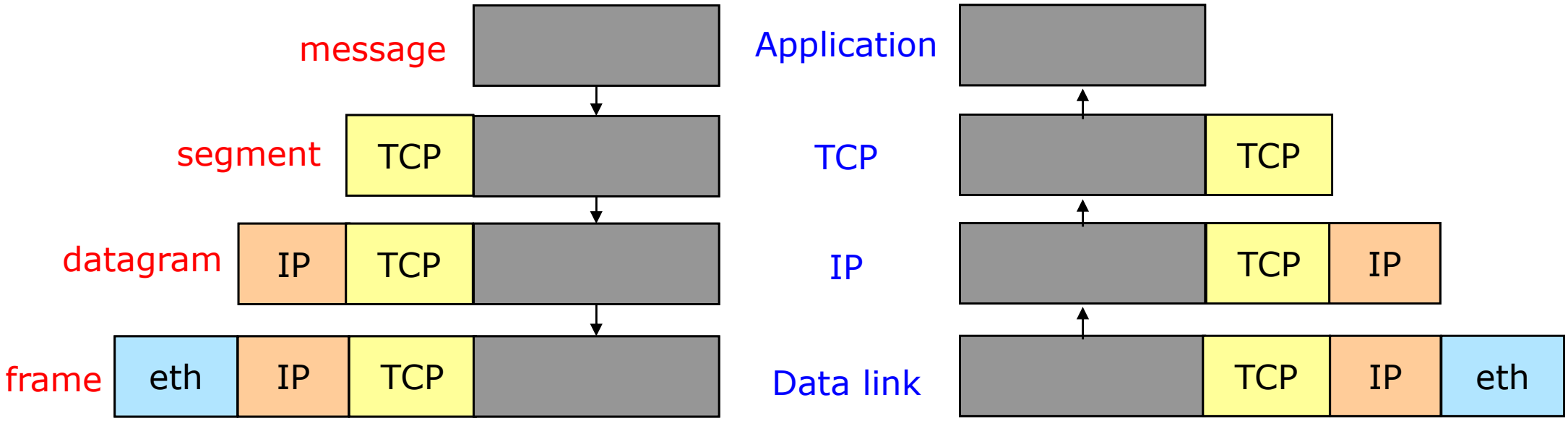
Example Network



Inter-Working between Layers

How do layers work?

- ✓ Layers do not look inside packet
- ✓ If they need auxiliary information, attach a header to message on way down, strip on way up



Addressing

Data link: MAC address

- ✓ 48 bits (Ethernet)

Network: IP address

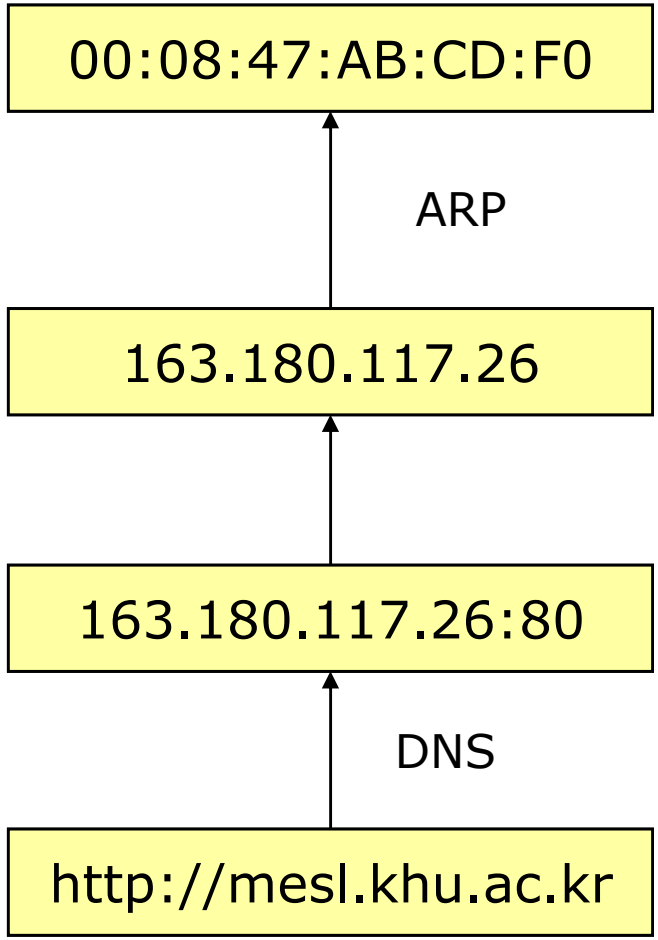
- ✓ 32 bits (IPv4), 128 bits (IPv6)
- ✓ Hierarchical: network + host part

Transport: Port number

Socket: <IP address, port number>

Application: Host name

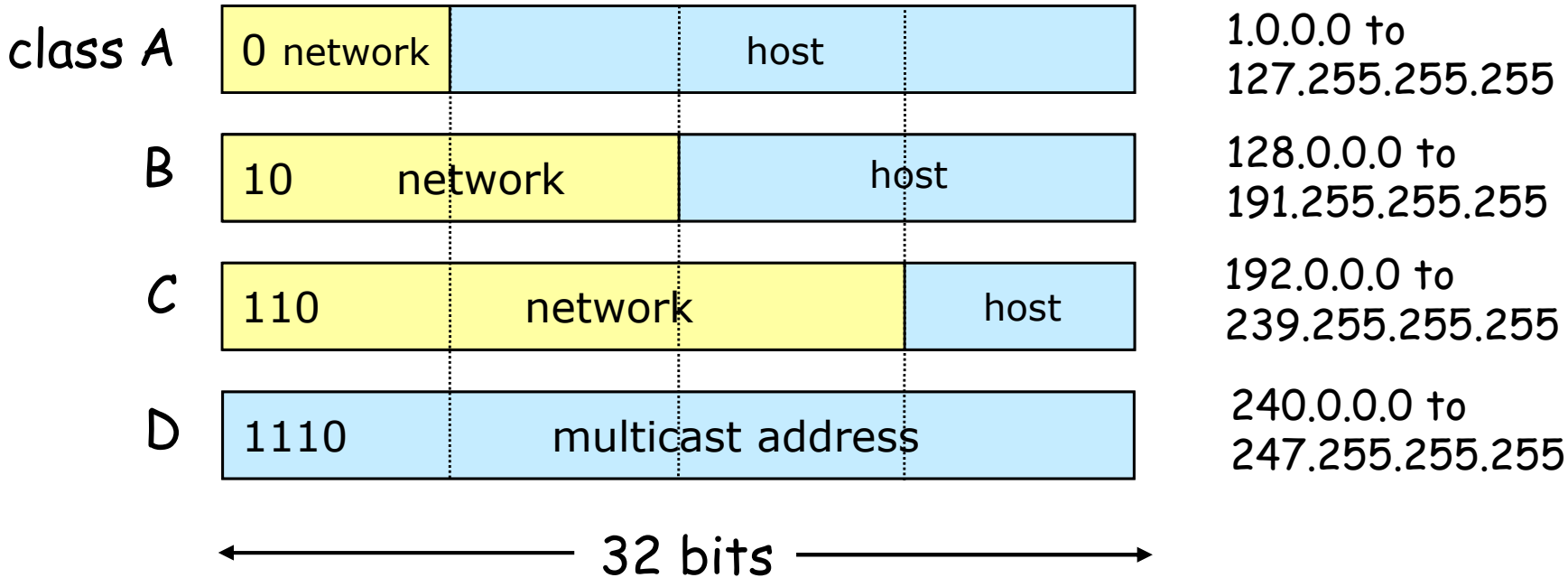
- ✓ Hierarchical



IP Addressing

IP addresses form a 2-level hierarchy

- ✓ Network part + host part
- ✓ Hosts on same network have the same prefix



IP Routing

Routing table lookup with destination IP address

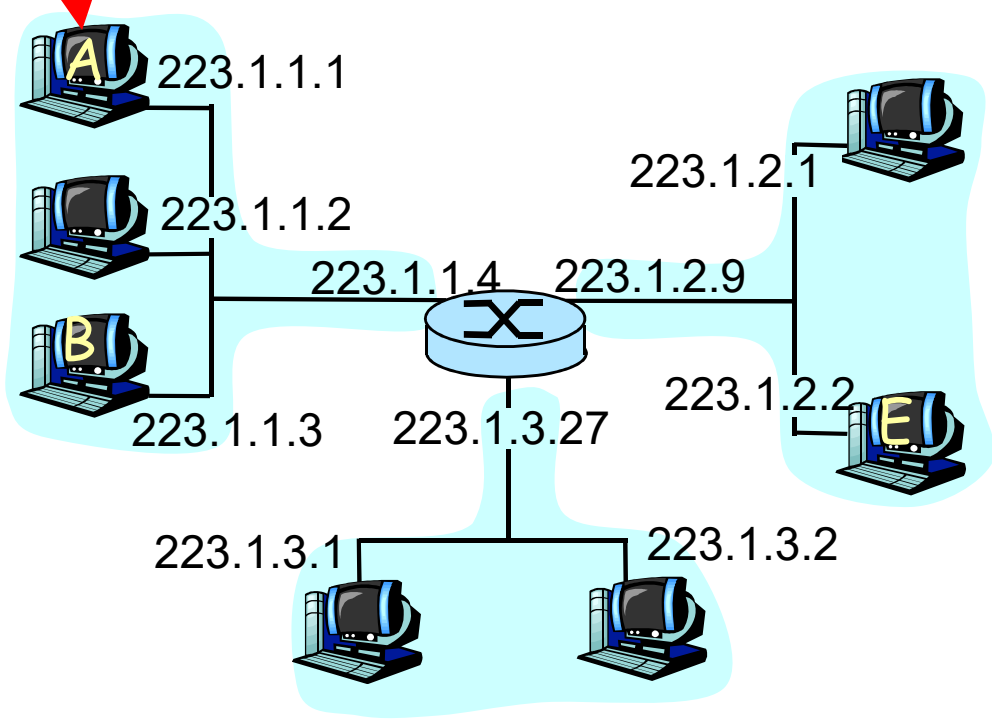
Routing algorithms

- ✓ Setup routing tables
- ✓ RIP (Routing Information Protocol)
- ✓ OSPF (Open Shortest Path First)

An example: A to E

- ✓ Look up network address of E
- ✓ E on different network
- ✓ Routing table: next hop router to E is 223.1.1.4
- ✓ Link layer sends datagram to router 223.1.1.4 inside link layer frame
- ✓ Datagram arrives at 223.1.1.4
- ✓ continued…

Dest. Net.	next router	Nhops
223.1.1		1
223.1.2	223.1.1.4	2
223.1.3	223.1.1.4	2



Transport Layer

Data transfer between processes

- ✓ Cf) Network layer: data transfer between end systems

Use “ports”

TCP (Transmission Control Protocol)

- ✓ reliable, in-order unicast delivery

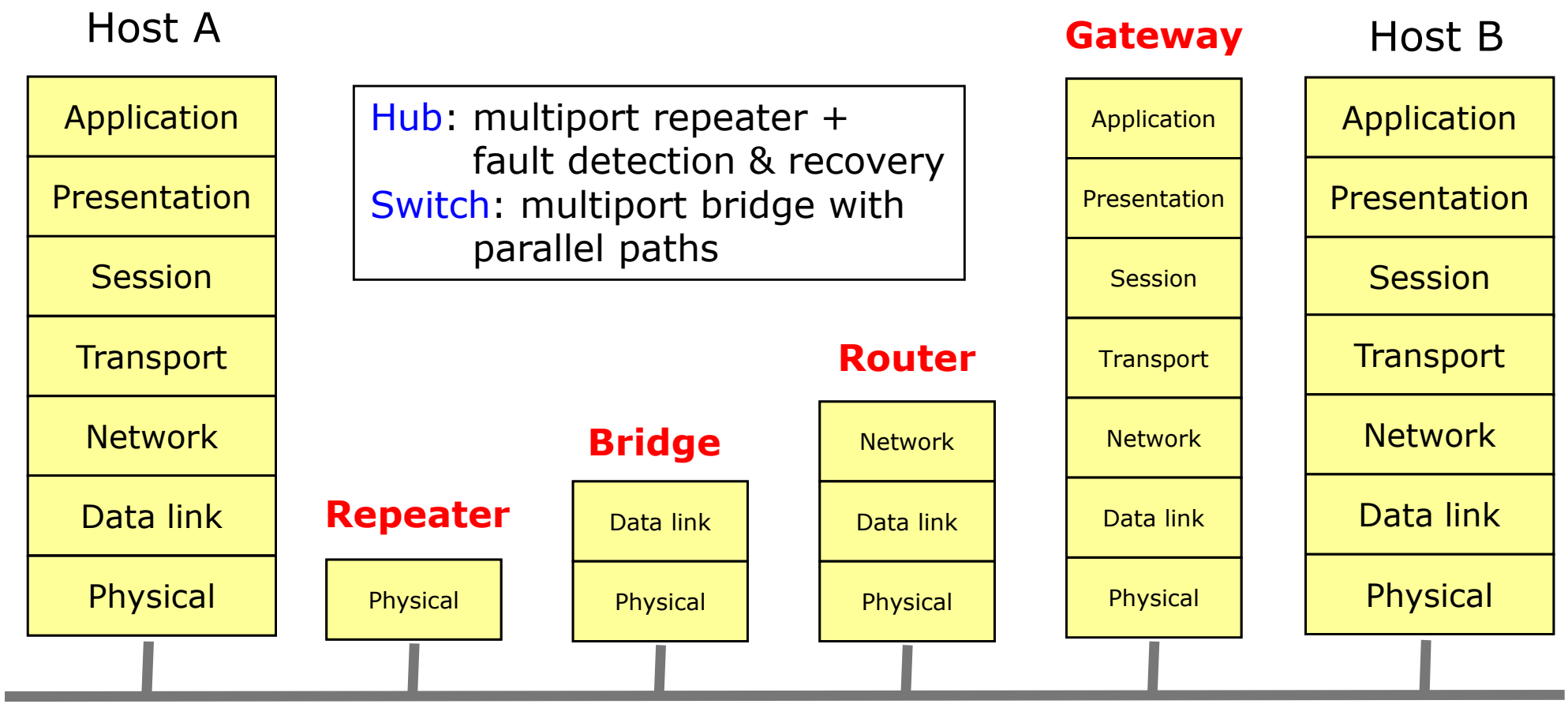
UDP (User Datagram Protocol)

- ✓ unreliable (“best-effort”) unordered unicast or multicast delivery

Services not available:

- ✓ Real-time
- ✓ Bandwidth guarantees
- ✓ Reliable multicast

Interconnection of Networks



Sockets

Introduced in BSD4.1 UNIX, 1981

Explicitly created, used, released by applications

Client/server paradigm

Two types of transport service

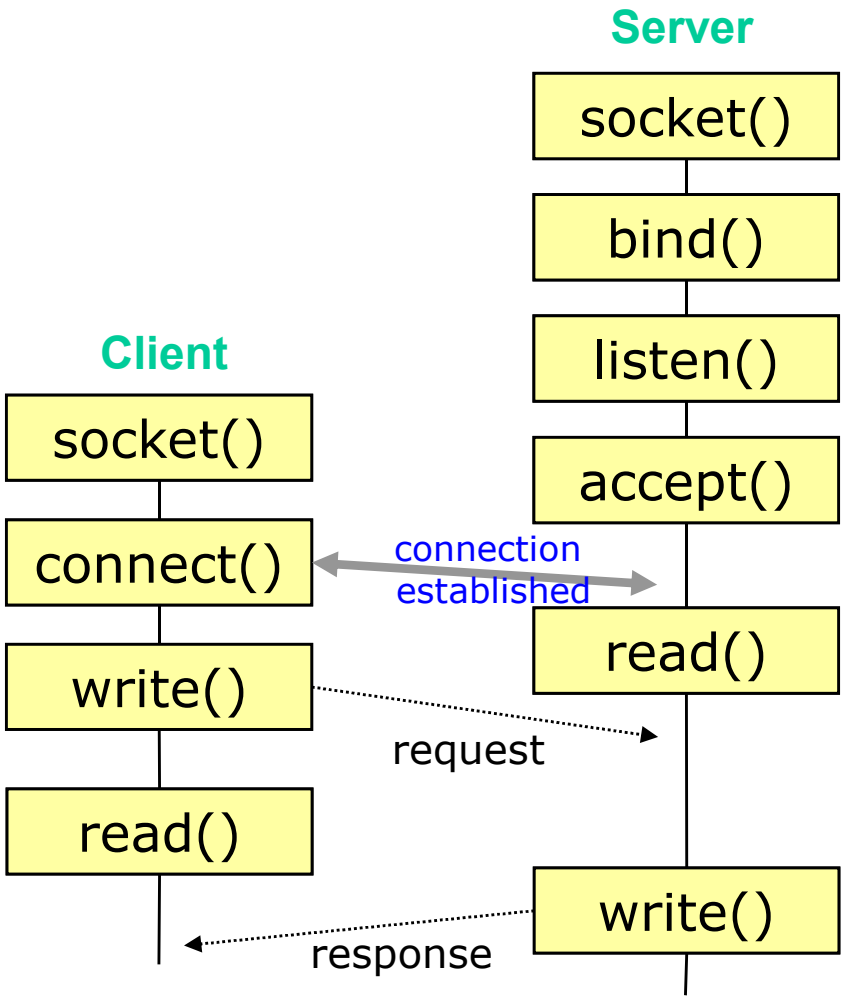
- ✓ Unreliable datagram
- ✓ Reliable, connection-oriented byte stream

Socket

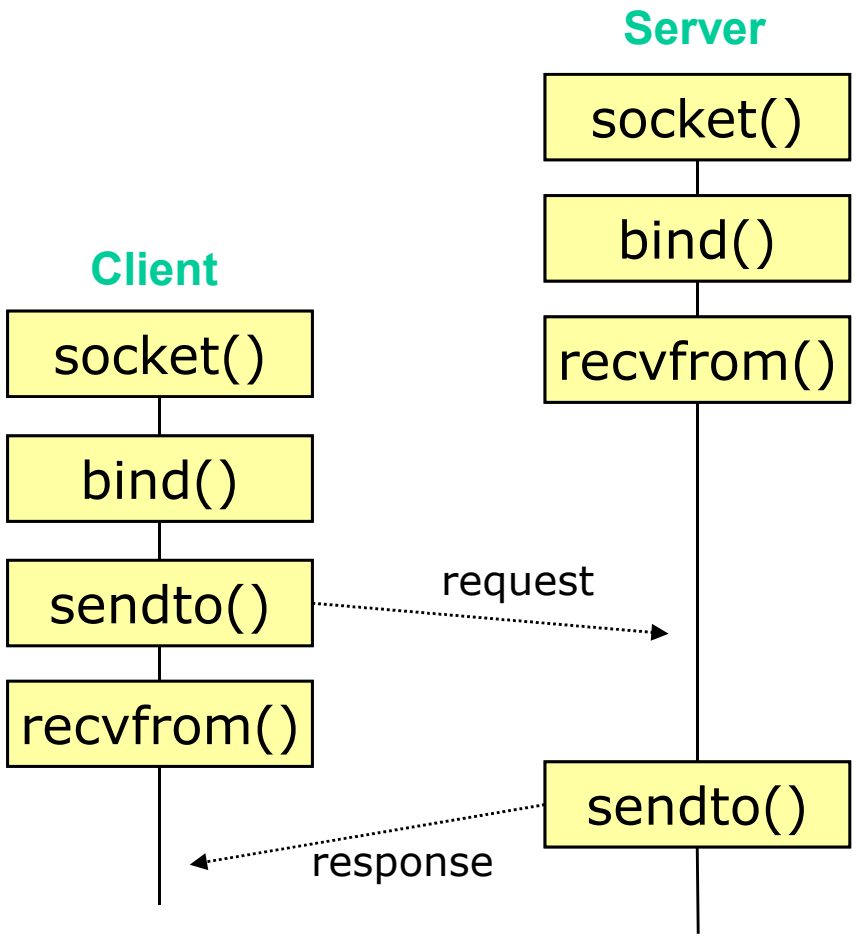
- ✓ a host-local, application-created/owned, OS-controlled interface (a “door”)
- ✓ Application can both send and receive messages to/from another application process via sockets
  - even in the same machine via UNIX-domain sockets (i.e., IPC through sockets)

System Calls for Sockets

Connection-oriented Service



Connectionless Service



System Calls for Sockets (Cont'd)

Create a socket

- ✓ `#include <sys/types.h>`
- ✓ `#include <sys/socket.h>`
- ✓ `int socket(int family, int type, int protocol);`
- ✓ return: socket descriptor if OK, –1 on error
- ✓ The first argument, **`family`**
  - **`PF_UNIX, PF_LOCAL`**
  - **`PF_INET`**
  - **`PF_INET6`**
  - **`PF_IPX`**
  - **`PF_X25`**
- ✓ The second argument, **`type`**
  - **`SOCK_STREAM`** : stream socket
  - **`SOCK_DGRAM`** : datagram socket
  - **`SOCK_RAW`** : raw socket
- ✓ The third argument, **`protocol`**
  - typically, 0

System Calls for Sockets (Cont'd)

Bind a name to an unnamed socket

- ✓ `#include <sys/types.h>`
- ✓ `#include <sys/socket.h>`
- ✓ `#include <netinet/in.h>`
- ✓ `int bind(int sd, struct sockaddr *myaddr, int addrlen);`
- ✓ return: 0 if OK, –1 on error
- ✓ The second argument, **`myaddr`**

```
struct sockaddr  {
  u_short  sa_family;   /* Protocol Family: PF_XXXX */
  char     sa_data[14]; /* Protocol-specific address */
};
```


System Calls for Sockets (Cont'd)

Bind a name to an unnamed socket (Cont'd)

- ✓ The second argument, **myaddr** (Cont'd)

```
struct in_addr  {
  u_long  s_addr; /* Network byte ordered 32-bit address */
};

struct sockaddr_in  {
  short           sin_family;  /* PF_INET */
  u_short         sin_port;    /* 16-bit port number */
  struct in_addr  sin_addr;    /* Network byte ordered address */
  char            sin_zero[8]; /* Unused */
};

struct sockaddr_un  {
  short           sun_family;    /* PF_UNIX */
  char            sun_path[108]; /* Path name */
};
```


System Calls for Sockets (Cont'd)

Establish a connection on a socket (client)

- ✓ `#include <sys/types.h>`
- ✓ `#include <sys/socket.h>`
- ✓ `int connect(int sd, struct sockaddr *servaddr, int addrlen);`
- ✓ return: 0 if OK, –1 on error

Listen connections on a socket (server)

- ✓ `#include <sys/types.h>`
- ✓ `#include <sys/socket.h>`
- ✓ `int listen(int sd, int backlog);`
- ✓ return: 0 if OK, –1 on error
- ✓ The second argument, **`backlog`**
  - how many connection requests can be queued by while executing **`accept`** system call
  - typically, 5

System Calls for Sockets (Cont'd)

Accept a connection on a socket (server)

- ✓ `#include <sys/types.h>`
- ✓ `#include <sys/socket.h>`
- ✓ `int accept(int sd, struct sockaddr *peer, int *addrlen);`
- ✓ return: new socket descriptor if OK, –1 on error

Close a socket

- ✓ `#include <unistd.h>`
- ✓ `int close(int sd);`
- ✓ return: 0 if OK, –1 on error

System Calls for Sockets (Cont'd)

Send or Receive a packet

- ✓ `#include <sys/types.h>`
- ✓ `#include <sys/socket.h>`
- ✓ `ssize_t send(int sd, void *buf, size_t nbytes, int flags);`
- ✓ `ssize_t sendto(int sd, void *buf, size_t nbytes, int flags, struct sockaddr *to, int addrlen);`
- ✓ `ssize_t recv(int sd, void *buf, size_t nbytes, int flags);`
- ✓ `ssize_t recvfrom(int sd, void *buf, size_t nbytes, int flags, struct sockaddr *from, int *addrlen);`
- ✓ return: number of bytes sent or received if OK, –1 on error
- ✓ Cf) **`write & read`** system call
- ✓ The third argument, **`flags`**
  - **`MSG_OOB`** : send or receive out-of-band data (TCP only)
  - **`MSG_PEEK`** : peek at incoming packet (**`recv`** or **`recvfrom`**)
  - **`MSG_DONTROUTE`** : bypass routing (**`send`** or **`sendto`**)

Integer Byte Ordering

Big-Endian

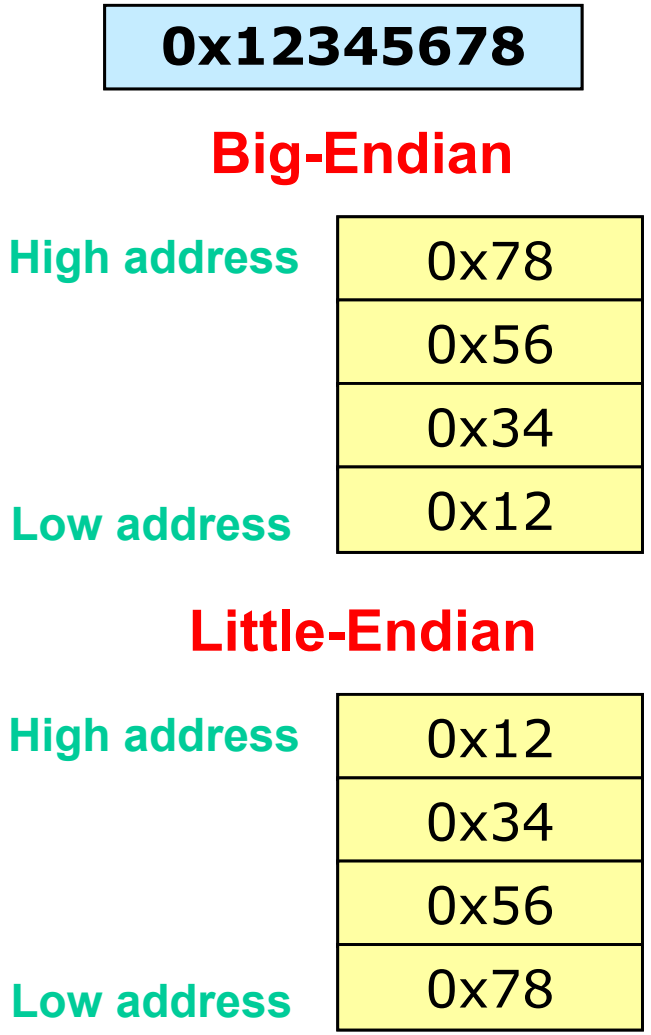
- ✓ Network byte order
- ✓ SPARC, Power-PC, MIPS, ...

Little-Endian

- ✓ Intel

Byte ordering functions

- ✓ `#include <sys/types.h>`
- ✓ `#include <netinet/in.h>`
- ✓ `u_long htonl(u_long hostlong);`
- ✓ `u_short htons(u_short hostshort);`
- ✓ `u_long ntohl(u_long netlong);`
- ✓ `u_short ntohs(u_short netshort);`
- ✓ return: conversion between network byte order and host byte order



Integer Byte Ordering (Cont'd)

```
#include <stdio.h>

main(int argc, char *argv[])
{
  int  a = 0x12345678;
  char *p = (char *)&a;

  printf("Address %p: %#x\n", p, *p);
  p++;
  printf("Address %p: %#x\n", p, *p);
  p++;
  printf("Address %p: %#x\n", p, *p);
  p++;
  printf("Address %p: %#x\n", p, *p);
  p++;
}
```

```
[Sun Workstation: SPARC]
(ceunix.khu.ac.kr)
Address ffbefcac: 0x12
Address ffbefcad: 0x34
Address ffbefcae: 0x56
Address ffbefcaf: 0x78
```

```
[Linux PC: Intel]
(celinux1.khu.ac.kr)
Address 0xbffffa48: 0x78
Address 0xbffffa49: 0x56
Address 0xbffffa4a: 0x34
Address 0xbffffa4b: 0x12
```

What if `printf("Address %p: %#x\n", p, *p++);` ?

Address Conversion

Address conversion functions

- ✓ `#include <sys/types.h>`
- ✓ `#include <sys/socket.h>`
- ✓ `#include <netinet/in.h>`
- ✓ `#include <arpa/inet.h>`
- ✓ `unsigned long inet_addr(char *str);`
- ✓ return: 32-bit address for dotted decimal notation if OK, –1 on error

- ✓ `char *inet_ntoa(struct in_addr inaddr);`
- ✓ return: dotted decimal notation for 32-bit address if OK, **`NULL`** on error

Exercise

TCP socket example

```
$ gcc -o tcps tcps.c (or make tcps)
$ gcc -o tcpc tcpc.c (or make tcpc)
$ ./tcps

$ ./tcpc
```

UDP socket example

```
$ gcc -o udps udps.c (or make udps)
$ gcc -o udpc udpc.c (or make udpc)
$ ./udps

$ ./udpc
```

Cf) Solaris → with `-l` options

```
$ gcc -o tcps tcps.c -lsocket -lnsl
```


Advanced Topics

DNS (Domain Name Service)

- ✓ `#include <netdb.h>`
- ✓ `struct hostent *gethostbyname(char *name);`
- ✓ return: pointer to struct hostent if OK, **`NULL`** on error

```
struct hostent {
  char  *h_name;        /* canonical name of host */
  char **h_aliases;     /* alias list */
  int    h_addrtype;    /* host address type */
  int    h_length;      /* length of address */
  char **h_addr_list;   /* list of addresses */
#define h_addr  h_addr_list[0]
                        /* address, for backward compatibility */
};
```


Exercise

DNS example

```
$ gcc -o tcps tcps.c (or make tcps)
$ gcc -o tcpc_dns tcpc_dns.c (or make tcpc_dns)
$ ./tcps

$ ./tcpc_dns 127.0.0.1
$ ./tcpc_dns 163.180.117.36
$ ./tcpc_dns celinux1.khu.ac.kr
```


Advanced Topics (Cont'd)

Synchronous I/O multiplexing

- ✓ `#include <sys/types.h>`
- ✓ `#include <sys/time.h>`
- ✓ `#include <unistd.h>`
- ✓
```
int select(int maxfd, fd_set *readfds, fd_set *writefds,
           fd_set *exceptfds, struct timeval *tvptr);
```
- ✓ return: count of ready descriptors if OK, 0 on timeout, –1 on error
- ✓ fd_set operation functions
  - `FD_ZERO(fd_set *fd_set);` `/* Clear all bits in fdset */`
  - `FD_SET(int fd, fd_set *fdset);` `/* turn the bit for fd on */`
  - `FD_CLR(int fd, fd_set *fdset);` `/* turn the bif for fd off */`
  - `FD_ISSET(int fd, fd_set *fdset);` `/* test the bif for fd */`

Advanced Topics (Cont'd)

Synchronous I/O multiplexing (Cont'd)

- ✓ The fifth argument, **tvptr**

```
struct timeval  {
  long  tv_sec;   /* seconds */
  long  tv_usec;  /* and microseconds */
};
```

- ✓ if **tvptr == NULL**,
  - wait forever
- ✓ if **tvptr->tv_sec == 0 && tvptr->tv_usec == 0**
  - don't wait at all
- ✓ if **tvptr->tv_sec != 0 && tvptr->tv_usec != 0**
  - wait the specified seconds and microseconds

Exercise

Make my own **`usleep`** library using **`select`** system call

```
$ gcc -o myusleep myusleep.c (or make myusleep)
$ ./myusleep
```

Synchronous I/O multiplexing example

```
$ gcc -o select select.c (or make select)
$ ./select

$ ./tcpc
$ ./udpc
$ ./ucoc
$ ./uclc
```


Advanced Topics (Cont'd)

Scatter read and Gather write (Scatter/Gather)

- ✓ `#include <sys/types.h>`
- ✓ `#include <sys/uio.h>`
- ✓ `ssize_t readv(int fd, struct iovec iov[], int iovcnt);`
- ✓ `ssize_t writev(int fd, struct iovec iov[], int iovcnt);`
- ✓ return: number of bytes read or written if OK, –1 on error

```
struct iovec {
  void   *iov_base; /* starting address of buffer */
  size_t  iov_len;  /* size of buffer */
};
```


Advanced Topics (Cont'd)

Scatter read and Gather write (Cont'd)

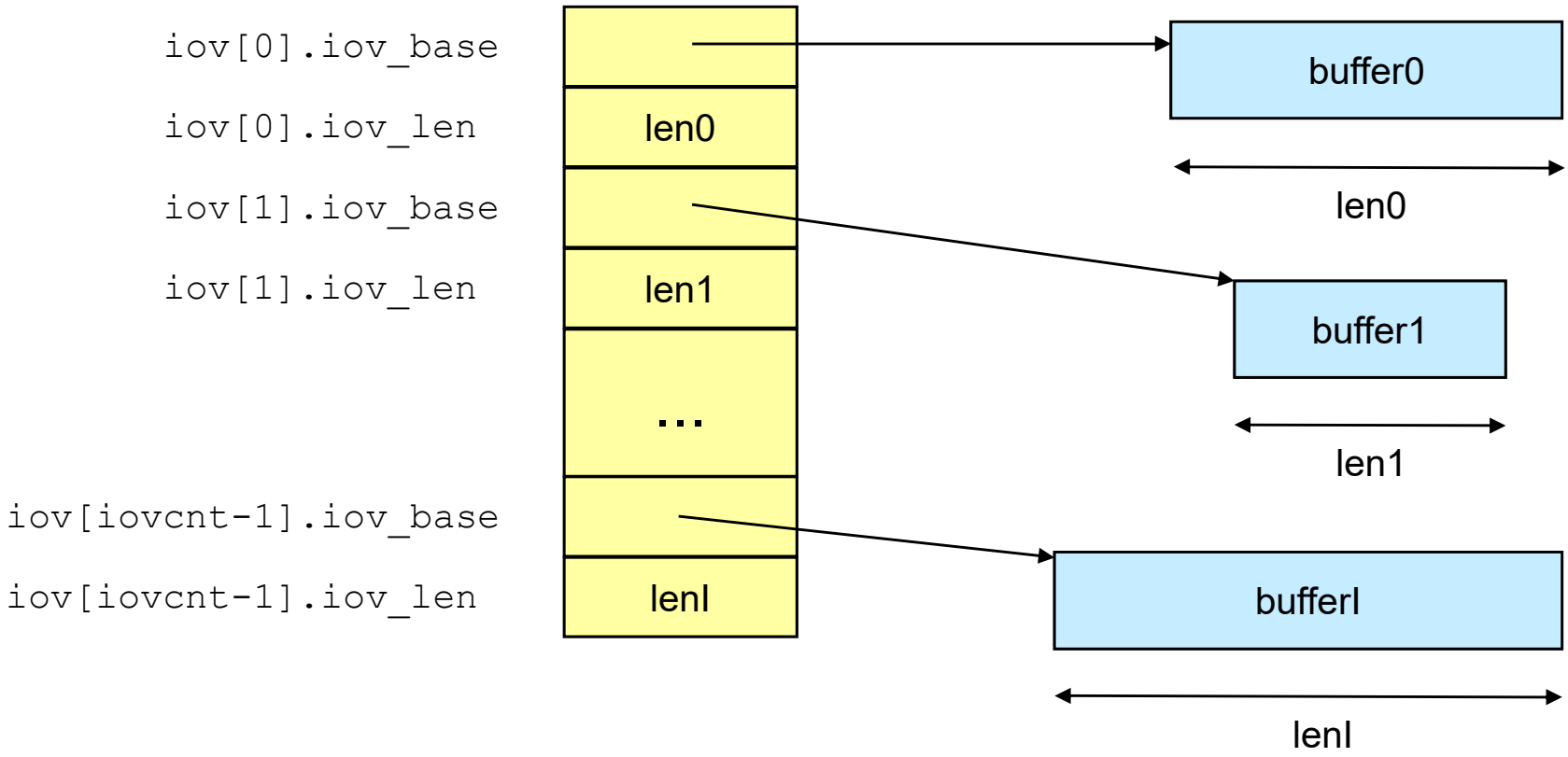
- ✓ `#include <sys/types.h>`
- ✓ `#include <sys/socket.h>`
- ✓ `ssize_t sendmsg(int sd, struct msghdr msg[], int flags);`
- ✓ `ssize_t recvmsg(int sd, struct msghdr msg[], int flags);`
- ✓ return: number of bytes sent or received if OK, –1 on error

```
struct msghdr {
  caddr_t       msg_name;      /* optional address */
  int           msg_namelen;   /* size of address */
  struct iovec *msg_iov;       /* scatter/gather array */
  int           msg_iovlen;    /* # of elements in msg_iov */
  caddr_t       msg_accrights;/* access rights */
  int           msg_accrightslen;
};
```


Advanced Topics (Cont'd)

Scatter read and Gather write (Cont'd)

- ✓ The **`iovec`** structure for **`readv`** & **`writev`**, **`sendmsg`** & **`recvmsg`**



Exercise

Scatter/Gather example

```
$ gcc -o sgs sgs.c (or make sgs)
$ gcc -o sgc sgc.c (or make sgc)
$ ./sgs

$ ./sgc
```


Advanced Topics (Cont'd)

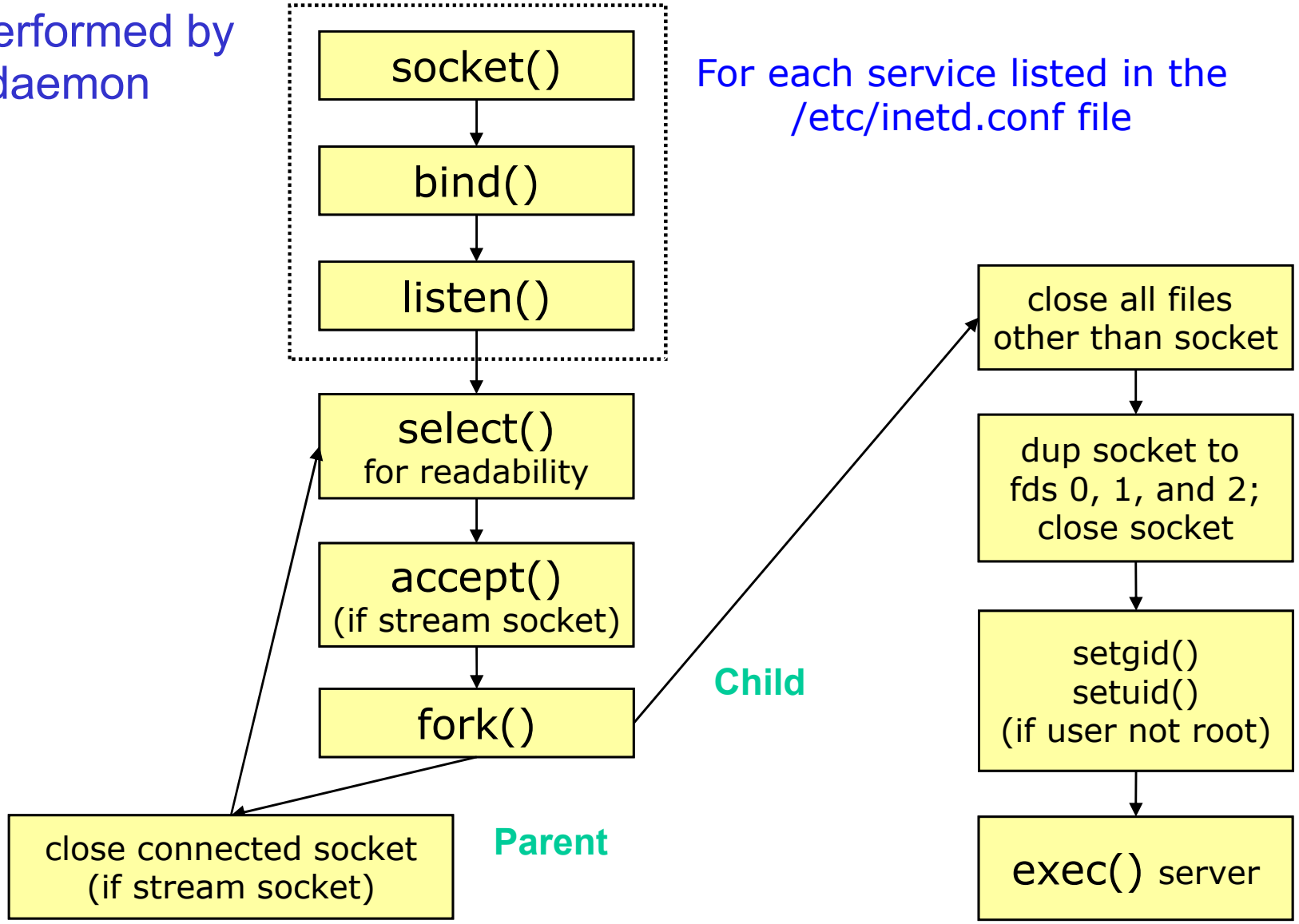
Socket options

- ✓ `#include <sys/types.h>`
- ✓ `#include <sys/socket.h>`
- ✓
```
int getsockopt(int sd, int level, int optname, char *optval,
               int *optlen);
```
- ✓
```
int setsockopt(int sd, int level, int optname, char *optval,
               int optlen);
```
- ✓ return: 0 if OK, –1 on error

- ✓ Cf) **`fcntl`** & **`ioctl`**

Advanced Topics (Cont'd)

Steps performed by **`inetd`** daemon



Summary

Socket

- ✓ Mechanisms for remote processes and/or threads to communicate with each other

System calls in Linux for sockets

- ✓ TCP server: **`socket, bind, listen, accept, read, write, close`**
- ✓ TCP client: **`socket, connect, read, write, close`**
- ✓ UDP server: **`socket, bind, recvfrom, sendto, close`**
- ✓ UDP client: **`socket, sendto, recvfrom, close`**

- ✓ **`htonl, htons, ntohl, ntohs`**
- ✓ **`inet_addr, inet_ntoa`**
- ✓ **`gethostbyname`**
- ✓ **`select`**
- ✓ **`getsockopt, setsockopt`**