Fall '20 COSE322-00

System Programming

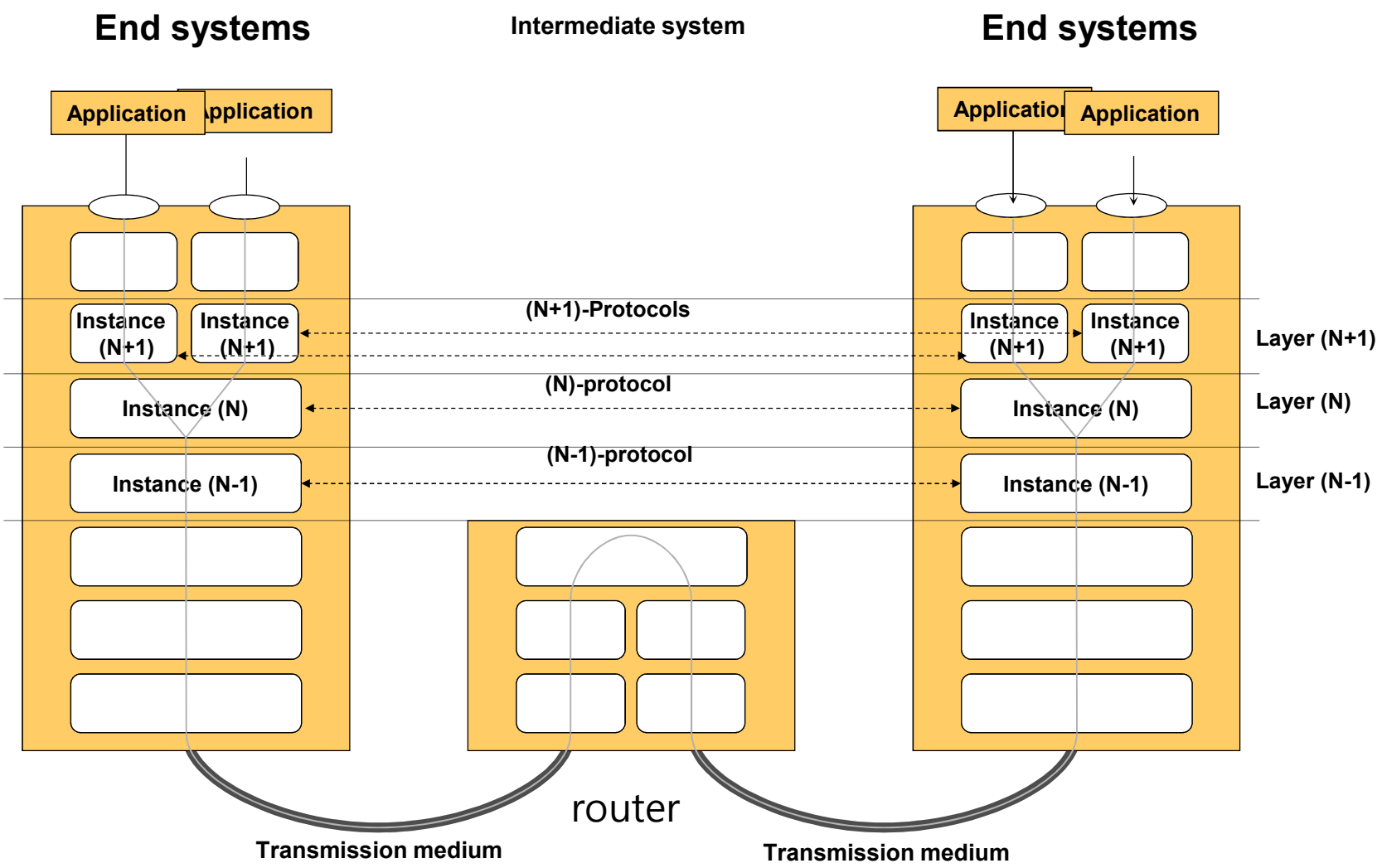
Lecture 08. Network Overview

2020. 10. 27.

Contents

- **Layer-Based Network Models**
  - Concepts and Characteristics
  - Protocol Stack
  - Port, Socket
- **Kernel structure**
  - Overall structure
  - Kernel networking
- **I/O Models in Linux Network**
  - Blocking, Non-blocking
  - I/O Multiplexing

Layer-Based Network Models



Layer-Based Network Models

Concepts in Layered Model

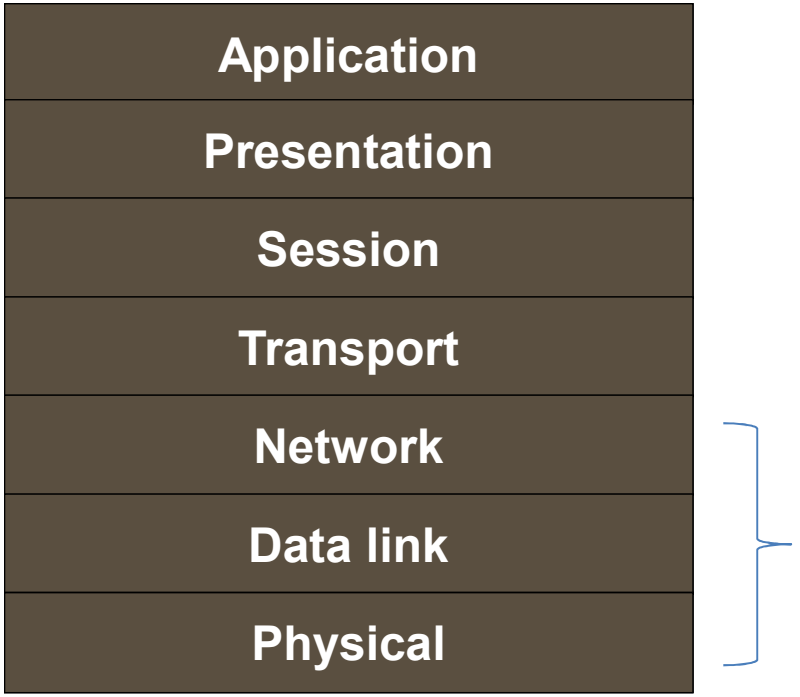
- Protocols
  - Rules that two parties talk and understand each other
  - Independent of hosts and technologies
  - Horizontal interface
- Services
  - Functions provided by a lower layer to the neighboring upper layer
  - Vertical interface

Characteristics

- Advantages: Simplifying design, implementation, understanding and maintenance
- Disadvantages: Overhead going through layers
  - No short cuts

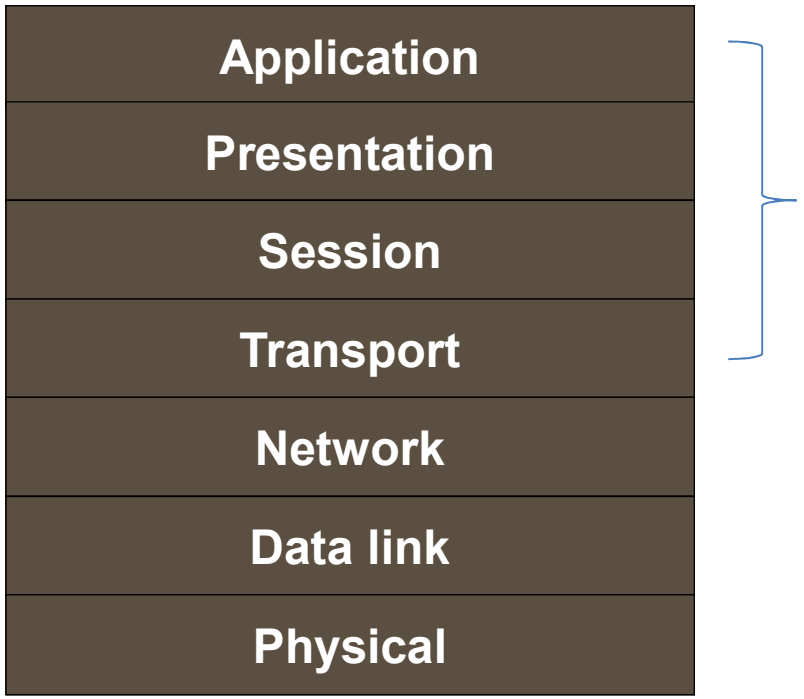
OSI Reference Model

- **Physical layer**
  - Handling bits on to wire
  - Implemented by hardware
  - Media types, coding methods
- **Data link layer**
  - Addressing within the physical network
  - Moving data in local network
    - two directly connected hosts
- **Network layer**
  - Connecting different networks
    - One global virtual network
  - Packet routing/forwarding to hosts



OSI Reference Model

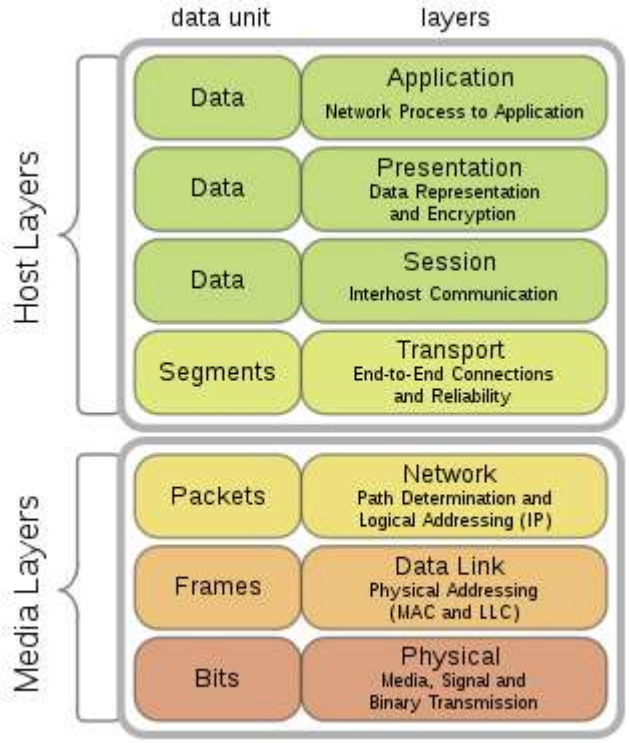
- **Transport layer**
  - End-to-end Delivery to apps
    - Flow control, packet ordering
- **Session layer**
  - opening, closing and managing a session
- **Presentation layer**
  - Regulating data representation
    - Encryption, XDR
- **Application layer**
  - SMTP, FTP, HTTP



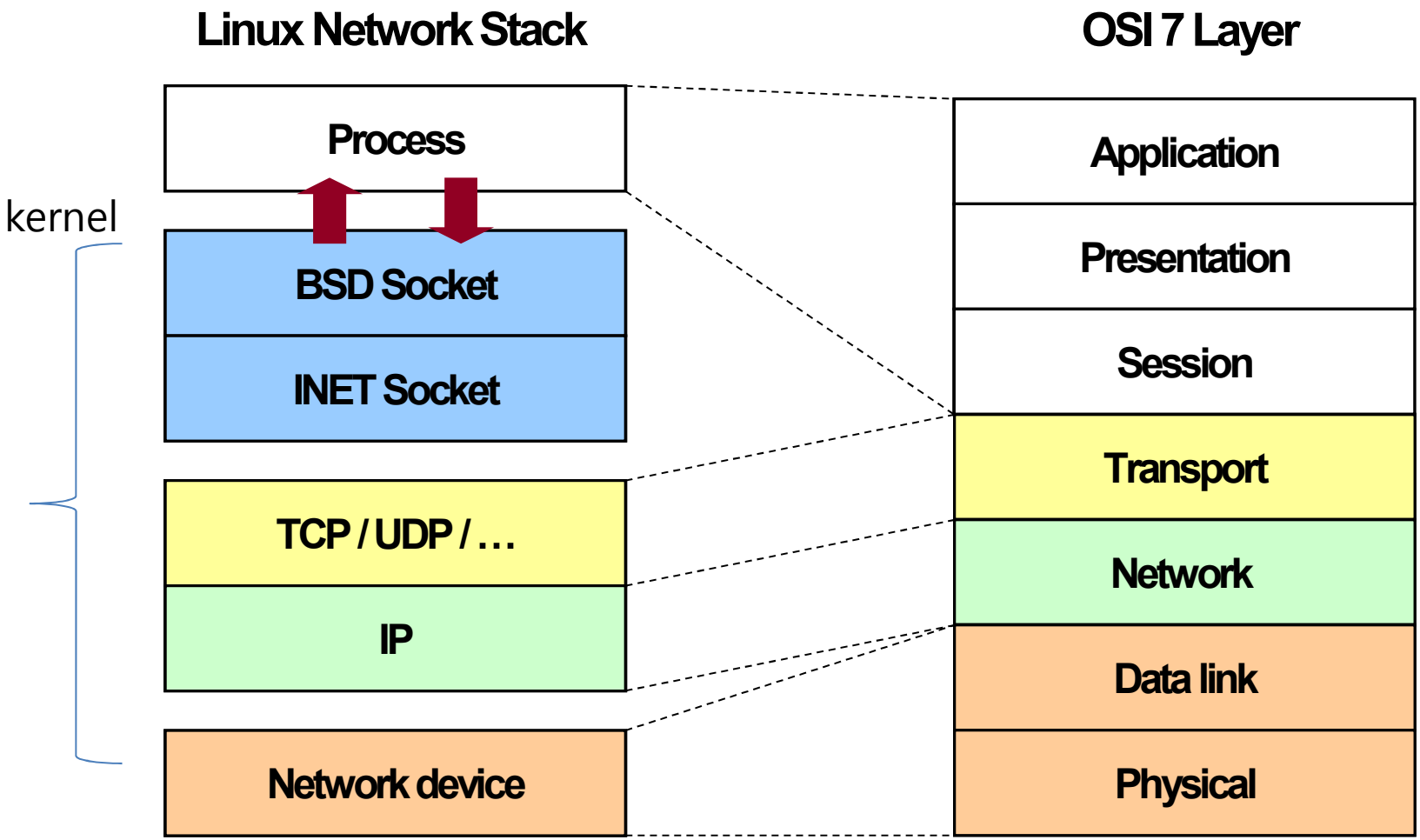
Protocol Stack

❖ What is it?

- The term stack refers to the actual software that implements the protocols
- A set of network protocols that work together
- Also called network stack
  - in OSI Reference Model
- Code inside kernel
  - Link to transport layer

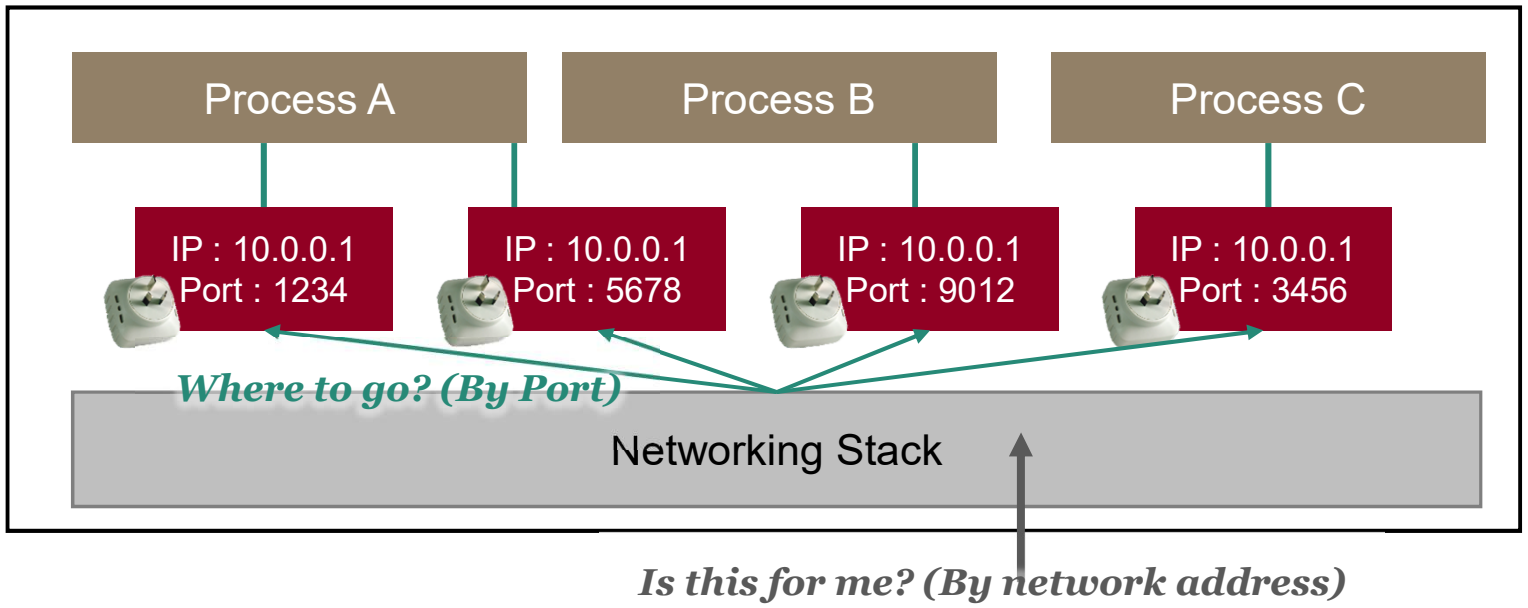


Protocol Stack in Linux Kernel



Port

- **Port (16bit)**
  - Abstraction provided by kernel
  - Communication endpoint in “host”
  - Multiple concurrent connections can be made out of a port



Port

❖ Well-known, Registered, Dynamic port

- 0 ~ 1023 : Well-known ports
- 1024 ~ 49151 : Registered ports
- 49152 ~ 65535 : Dynamic ports

❖ Well-known ports

Port	Descriptions
0	Reserved
20	FTP (for data)
21	FTP (for control)
22	SSH
25	SMTP

Port	Descriptions
53	DNS
80	HTTP
110	POP3
161	SNMP
179	BGP

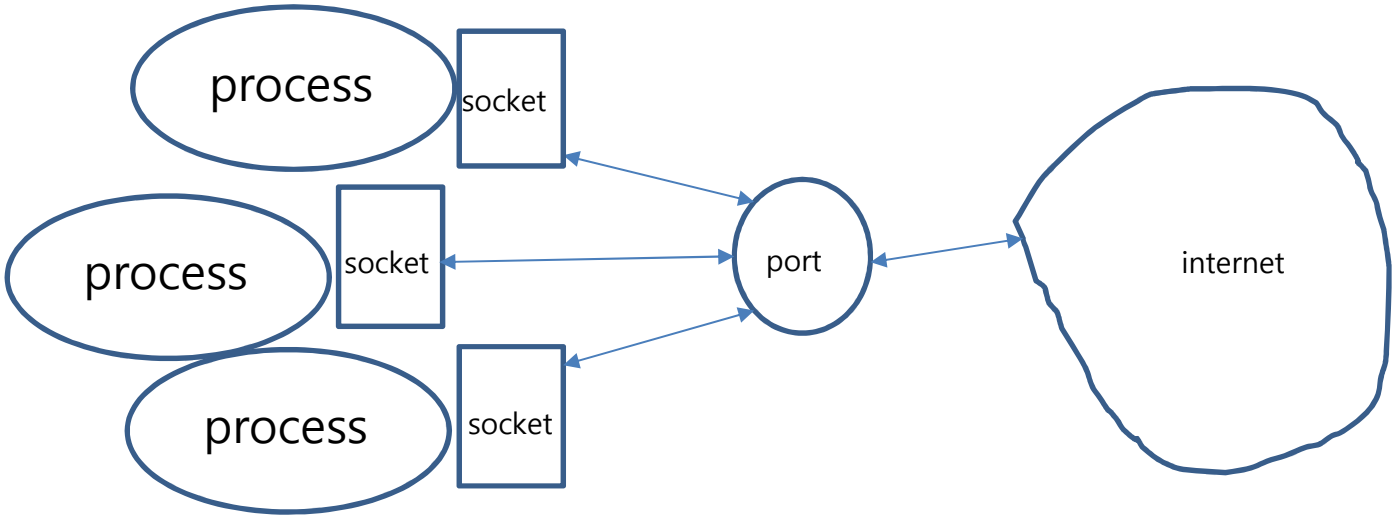
Port vulnerability

- **Port is open**
  - Exposed to outside world
  - Many ports are blocked for security
    - To avoid port scan

- **Ethernet is not encrypted**
  - Clear text passwd
    - tcpdump

Socket

- An **end-point** for Internet network "connection"
- Socket is defined by the combination of a network address and a port identifier
  - Network Address: IP address
- Process view to port
  - Port is shared among processes

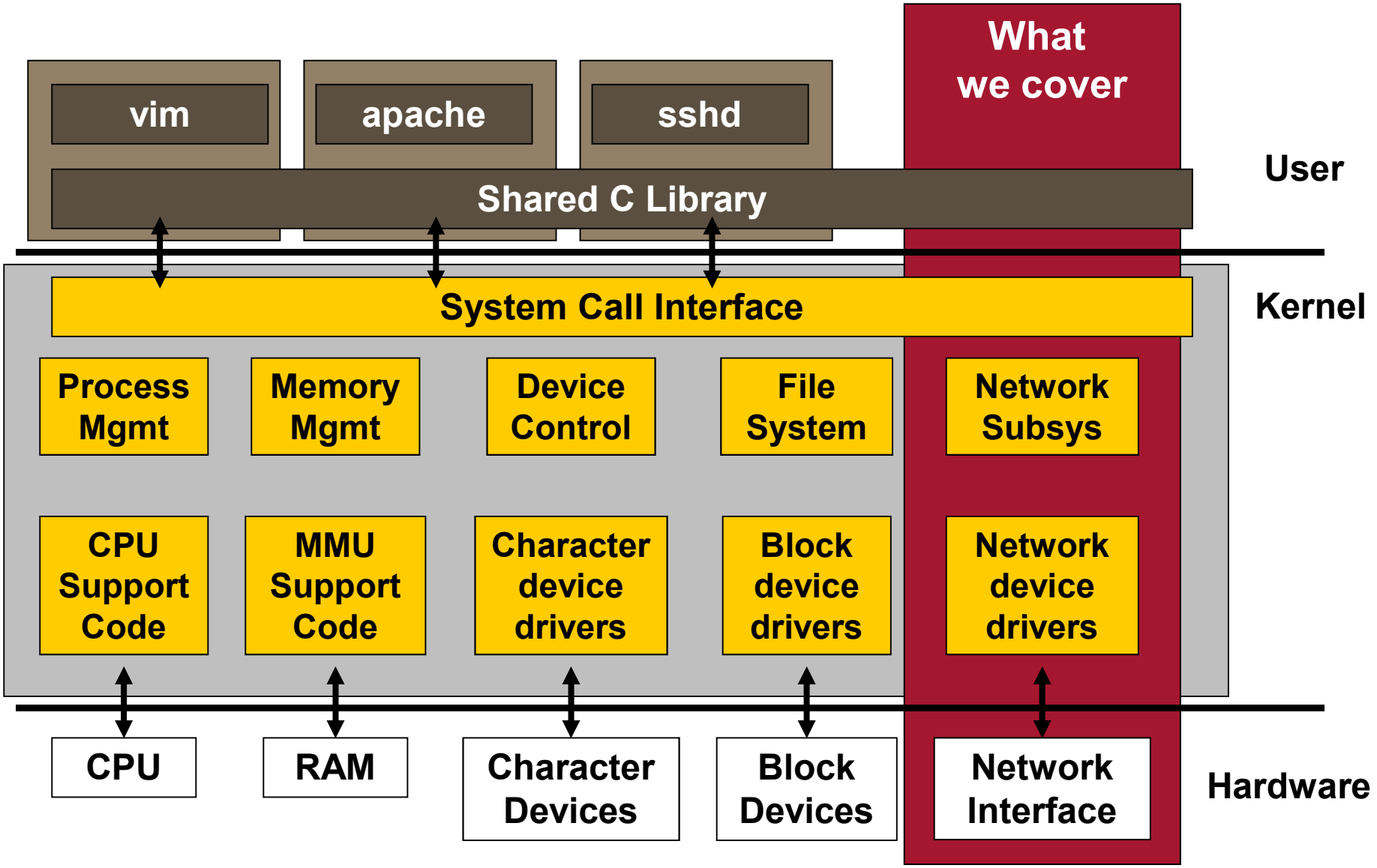


More on socket

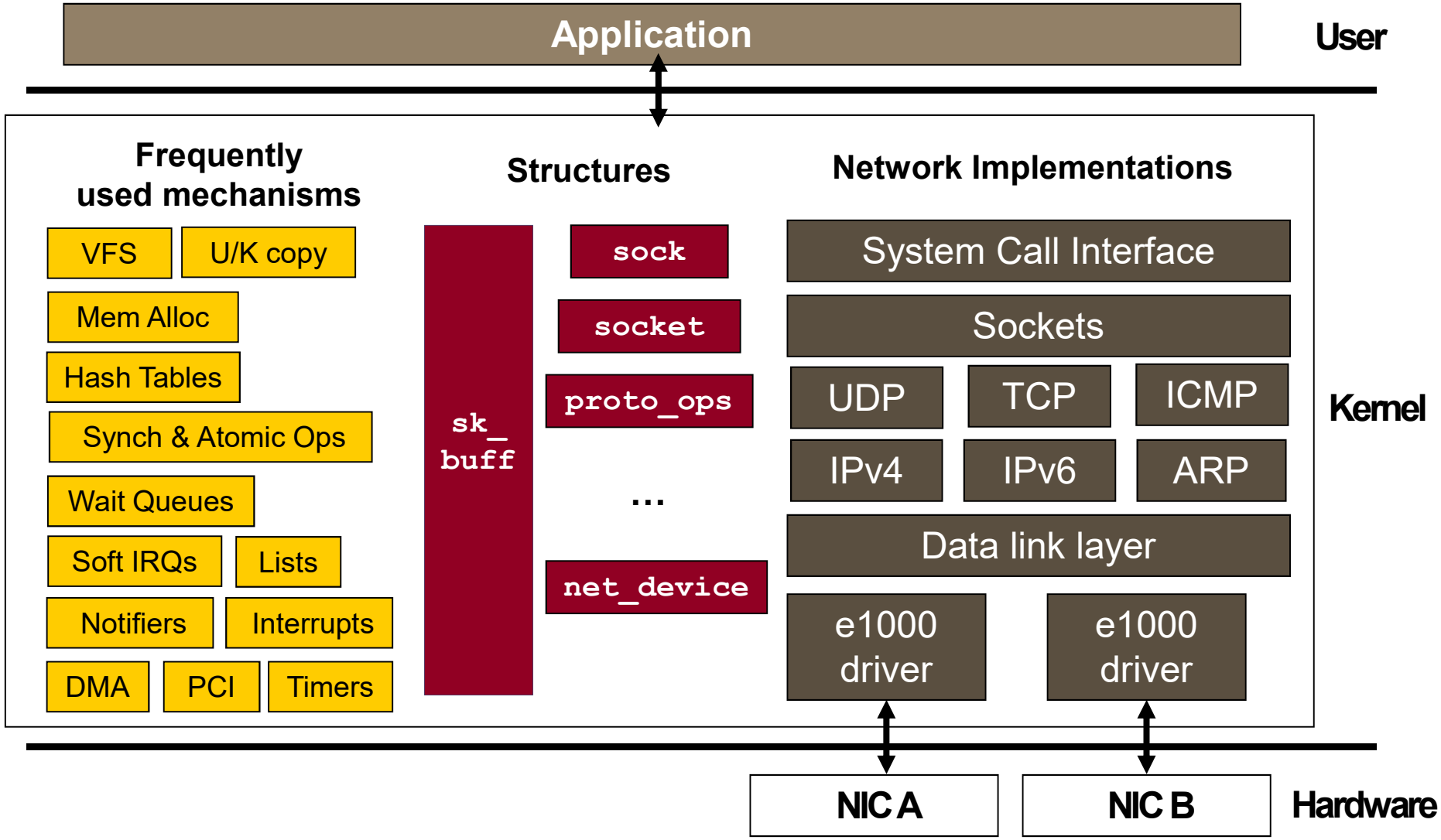
- **Use file descriptor**
  - socket(2) returns fd
  - The fd is used to send or receive packets

- **Does not have open(2)**
  - Instead, connect and listen
  - Since “connection” is transient and not shared

Kernel Structure



Kernel Networking



I/O Models in Linux Network

❖ **After the I/O request,**

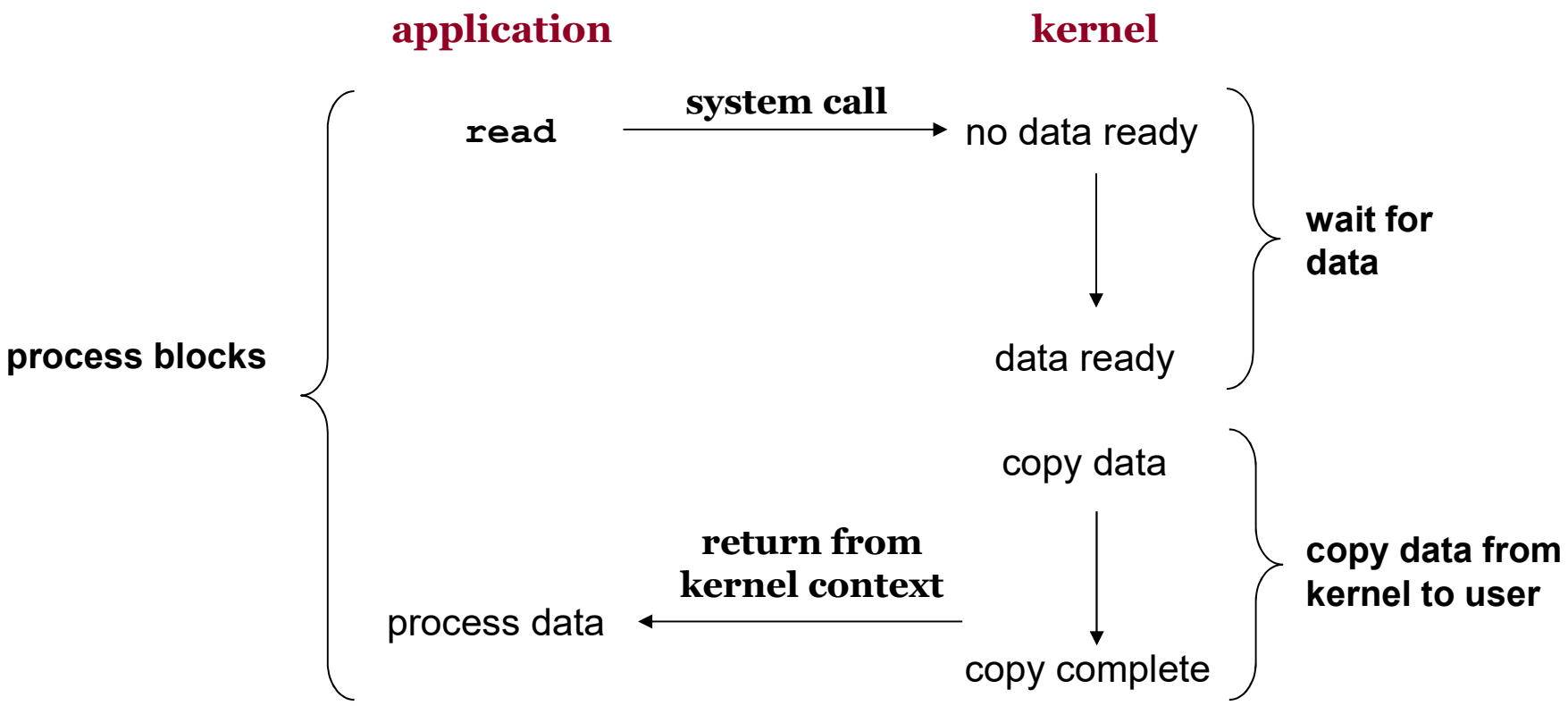
- ***Blocking:*** waits until the data to be ready
- ***Non-blocking:*** always returns

Blocking	`read/write` system call
Non-blocking	`read/write` system call with `O_NONBLOCK`
	`aio_read/aio_write`

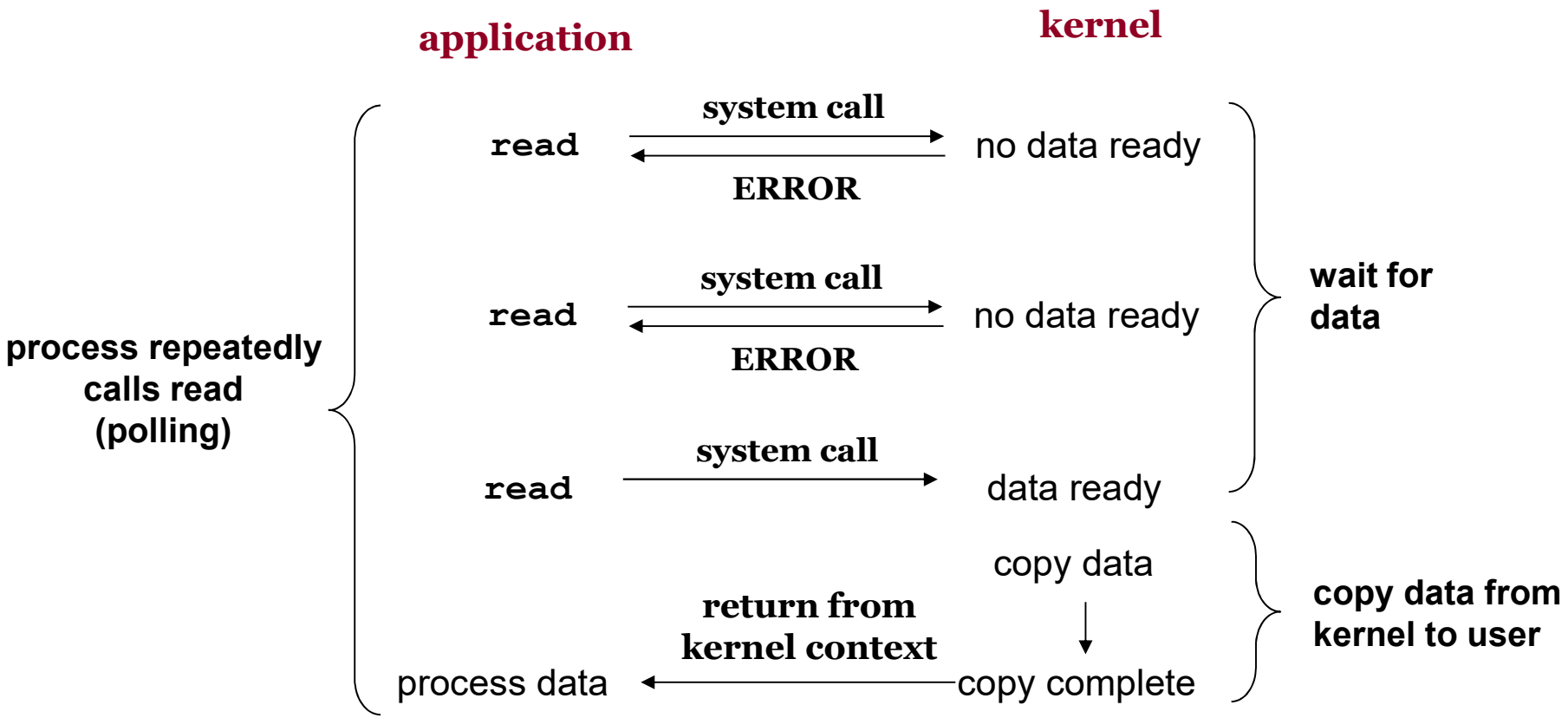
❖ **I/O Multiplexing**

- `select` system call

Blocking I/O (read)



Non-blocking I/O (read with O_NONBLOCK)



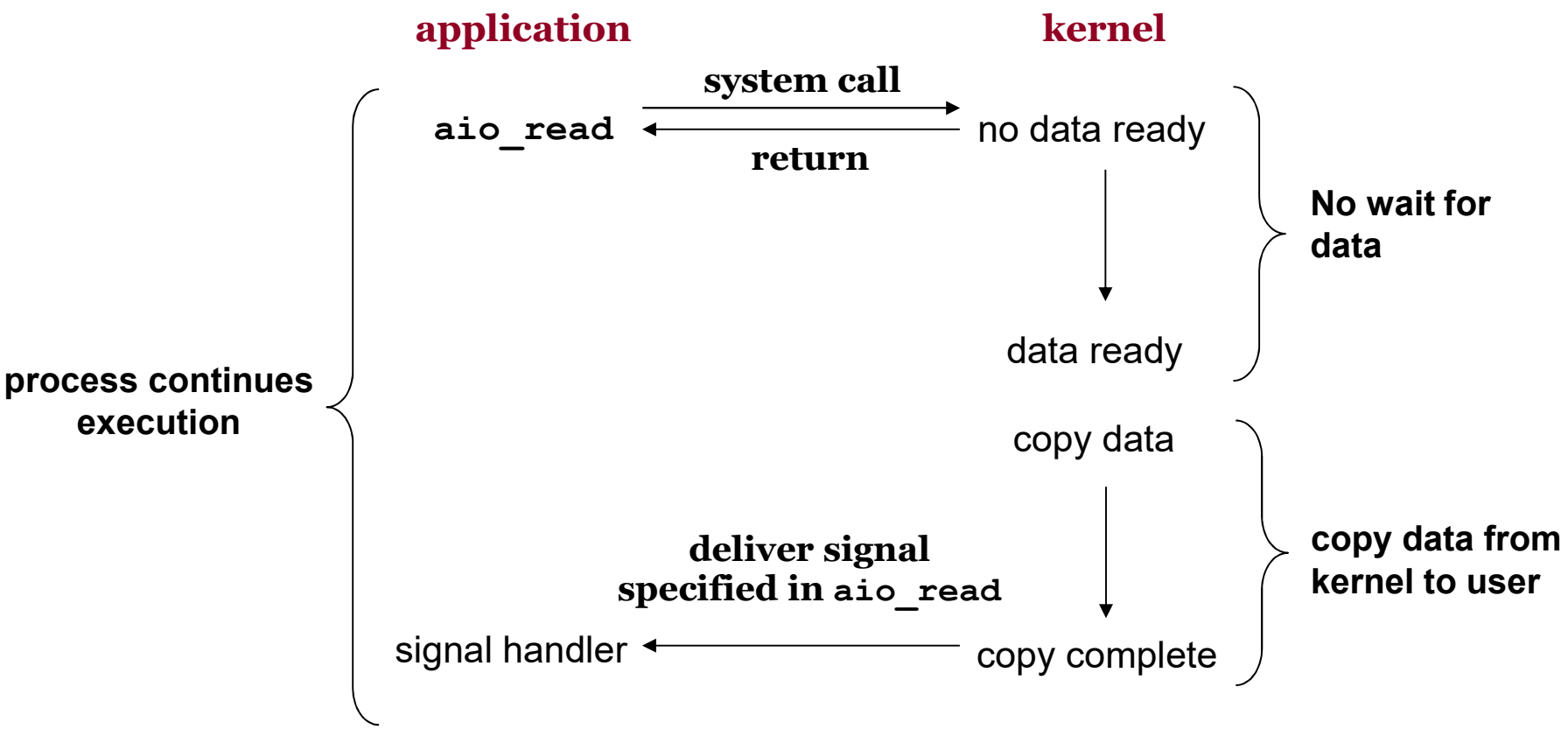
Non-blocking I/O (AIO)

❖ `aio_read()` is an user API for **AIO**

```
int aio_read(struct aiocb *aiocbp);
```

- Queues the I/O request described by the buffer pointed by **aiocbp**
  - `struct aiocb` : Defines parameters that control an I/O operation
    - File descriptor, Location of buffer, Length to transfer, Notification Method, etc.
- The read request **returns immediately**
  - The application can perform other processing while the background read operation completes
- When read response arrives, a signal or a thread-based callback is generated to **notify** the completion

AIO diagram


application
kernel
system call
aio_read
return
no data ready
data ready
No wait for data
process continues execution
copy data
deliver signal specified in aio_read
signal handler
copy complete
copy data from kernel to user

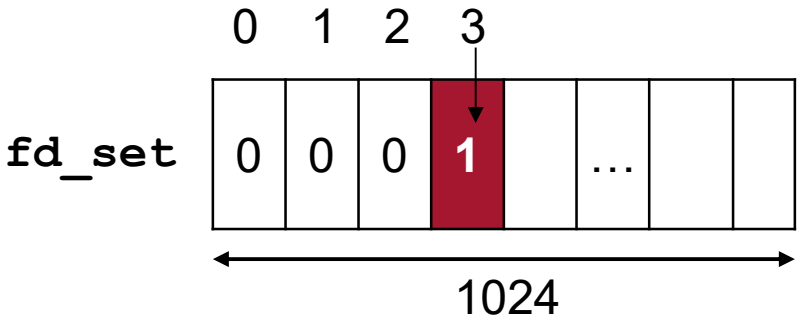
I/O Multiplexing (Select System Call)

- **`select` system call**
  - Allows a program to monitor **multiple file descriptors**
  - **Waits until** one or more of the file descriptors become **ready** for some class of I/O operation (read, write, exception)
  - When data is available for one of the descriptors, the `select` system call **returns**
  - The data copy operation is **initiated by return** of `select` system call. (e.g., `read` system call)
  - Superseded by epoll(2)

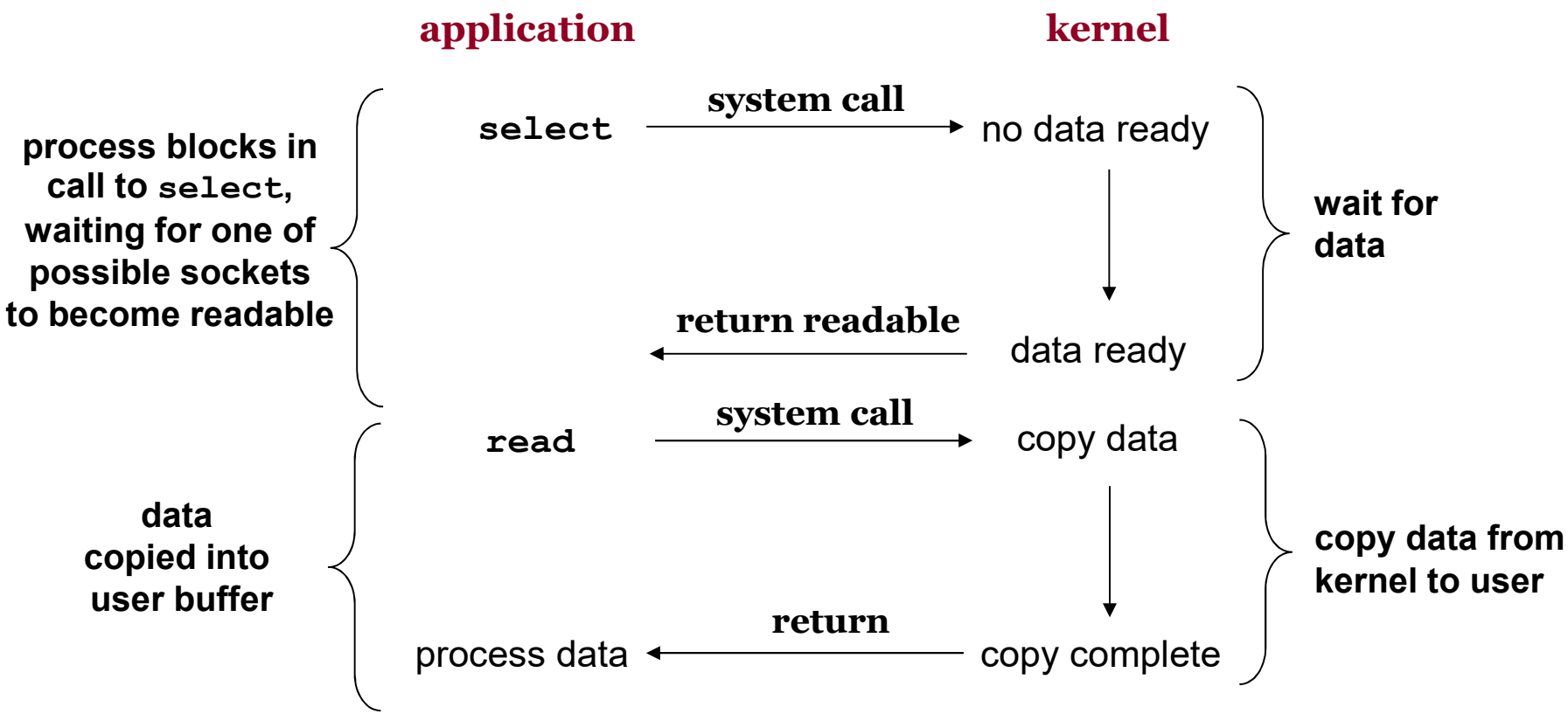
Select System Call syntax

```
int select(int nfds, fd_set *readfds, fd_set *writefds,
           fd_set *exceptfds, struct timeval *timeout)
```

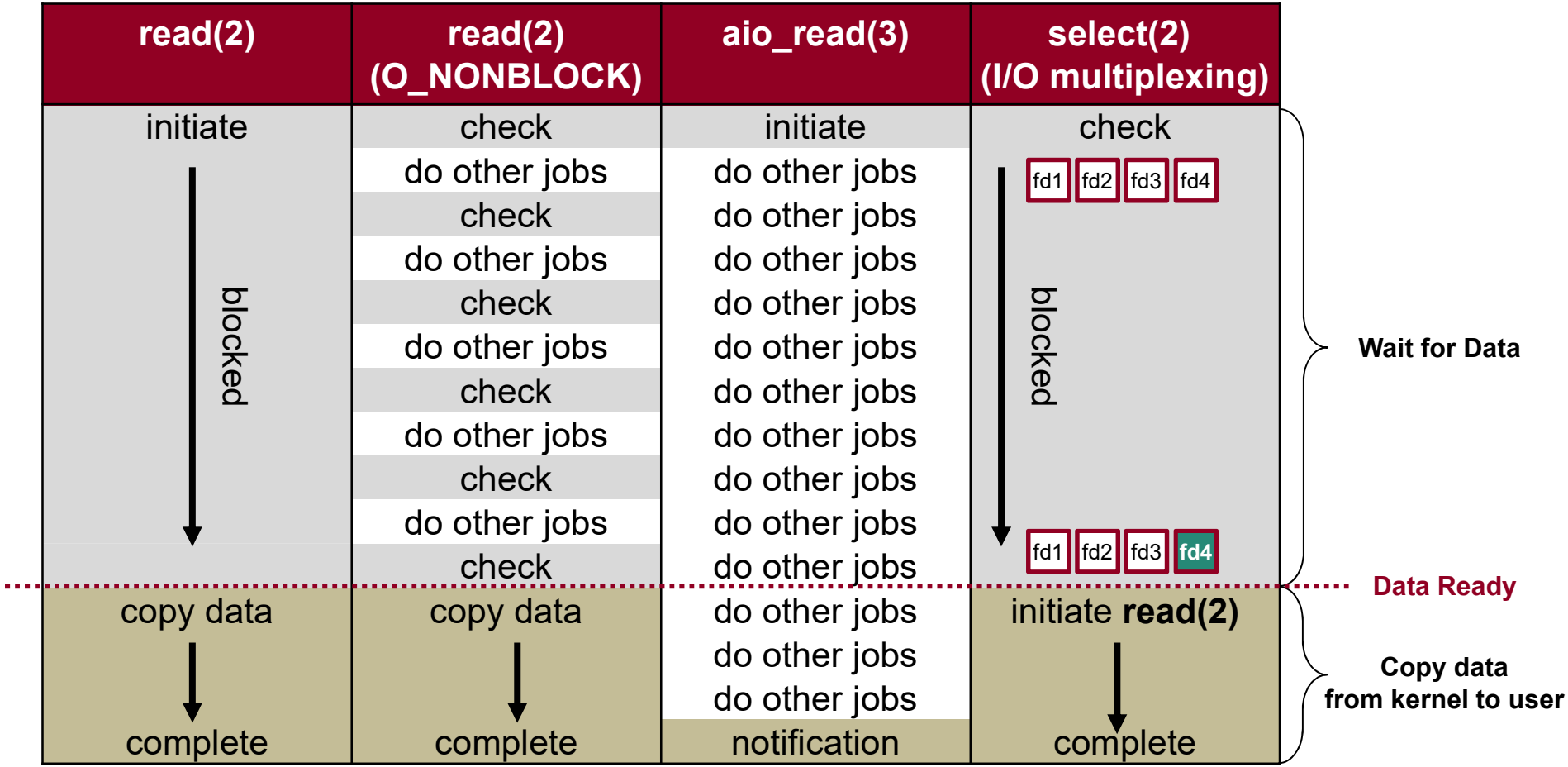
- **`nfds`** *:* the highest-numbered file descriptor, plus 1
- Three independent **bitmaps of file descriptors** (**`struct fd_set`**)
  - **`readfds`**(for read), **`writefds`**(for write), **`exceptfds`**(for exceptions)
  - Each bitmap covers 1024 fds
- **`timeout`** : minimum interval that select() should block



Select System Call diagram



Comparison

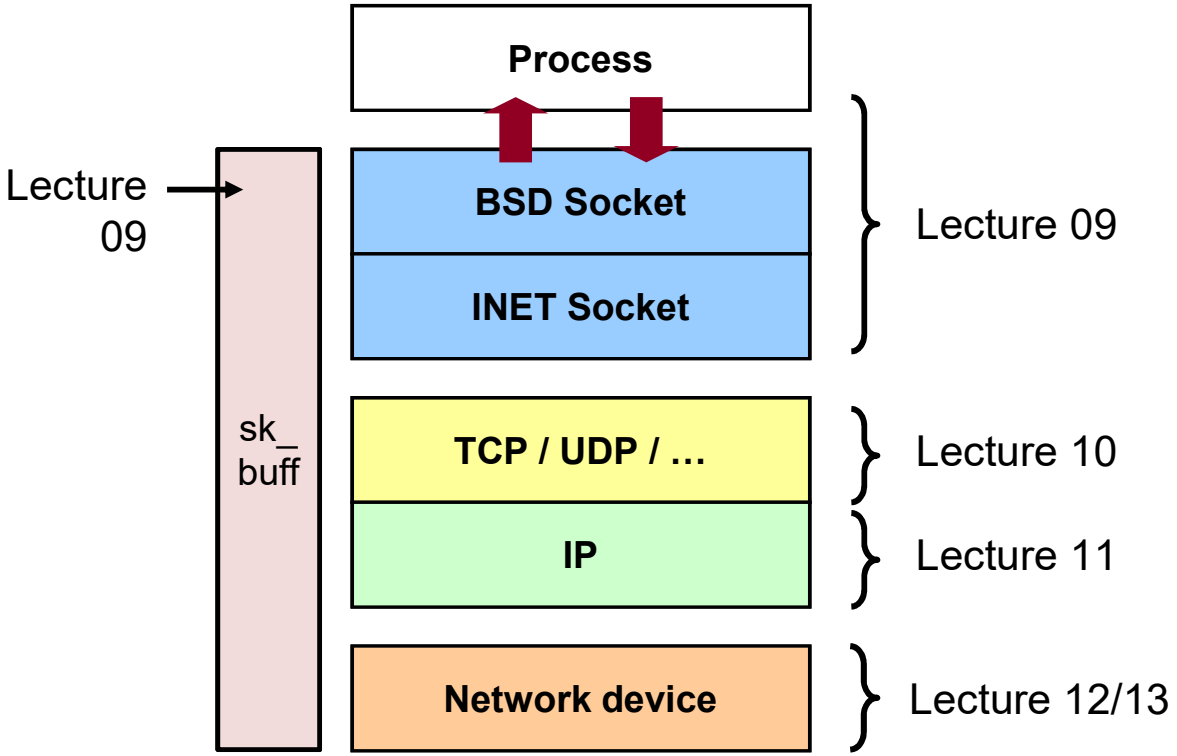

read(2)
read(2) (O_NONBLOCK)
aio_read(3)
select(2) (I/O multiplexing)
initiate
blocked
copy data
complete
check
do other jobs
check
do other jobs
check
do other jobs
check
do other jobs
check
do other jobs
check
copy data
complete
initiate
do other jobs
do other jobs
do other jobs
do other jobs
do other jobs
do other jobs
do other jobs
do other jobs
do other jobs
notification
check
fd1 fd2 fd3 fd4
blocked
fd1 fd2 fd3 fd4
initiate read(2)
complete
Wait for Data
Data Ready
Copy data from kernel to user

Thread and networking

- **Benefit of thread**
  - Thread can block
  - Other threads can do other computation in parallel

Lecture plan for Networking

❖ **Top-down approach for network implementations**



– Wrapping up (Lecture 14)