EDAF35: OPERATING SYSTEMS

MODULE 8
I/O SYSTEMS

CONTENTS

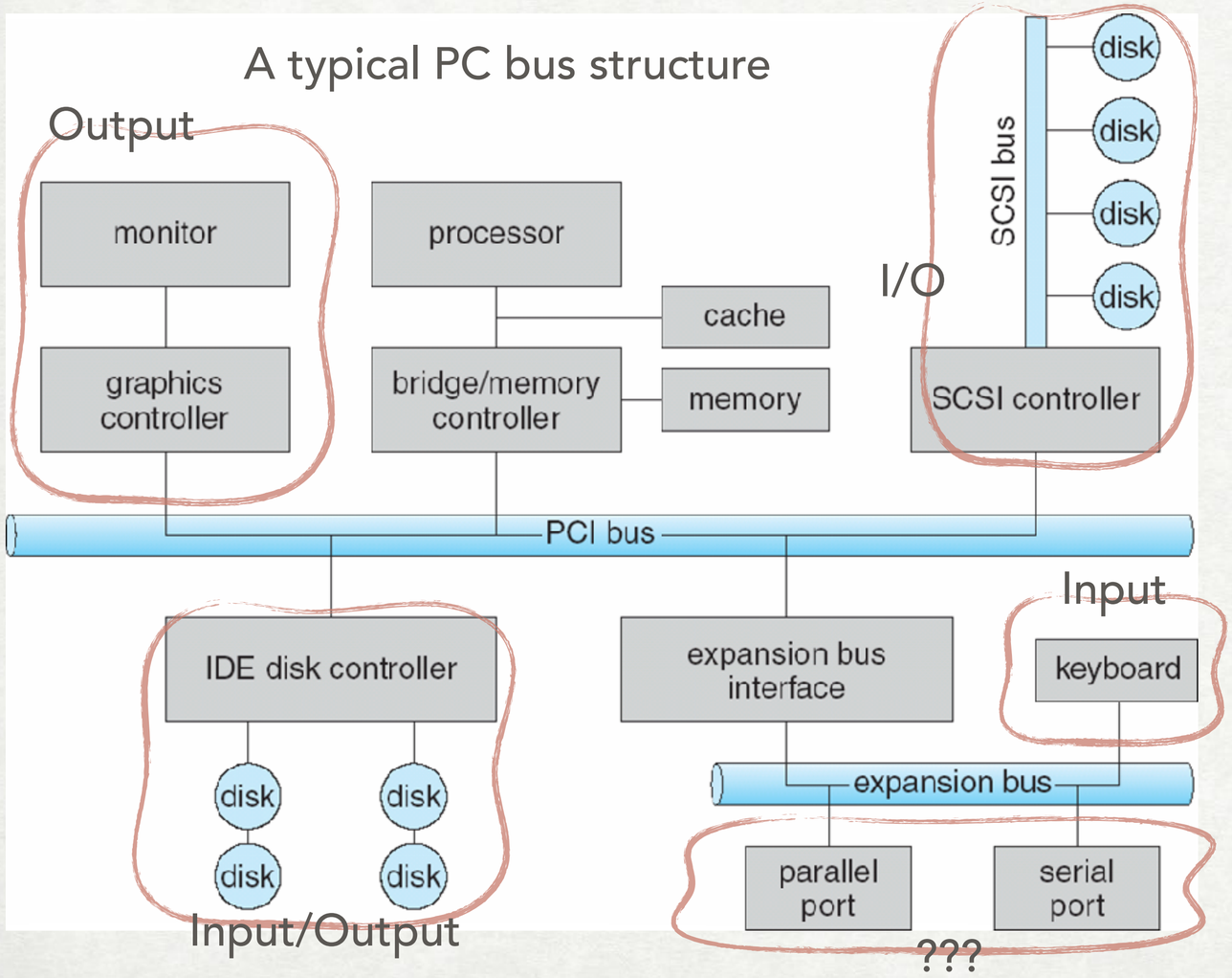
I/O SYSTEMS

- Input/Output (I/O) hardware
- I/O mechanisms (polling/interrupts/DMA)
- OS I/O organization
- From I/O requests to hardware operations

CHAPTER 13
I/O SYSTEMS

I/O HARDWARE IN A COMPUTING SYSTEM

I/O SYSTEMS



- I/O — essential in computing systems
- many devices doing I or O or both
- greatly different in speed and functionality
- similar interface towards OS: **device drivers** (piece of software)
- port- vs. memory-mapped I/O

PC DEVICE I/O PORT LOCATIONS

EXAMPLE – I/O SYSTEMS

USE SPECIAL CPU INSTRUCTIONS:
IN/OUT ADDR, REG
INSB/OUTSB ...
INSW/OUTSW ...

I/O address range (hexadecimal)	device
000–00F	DMA controller
020–021	interrupt controller
040–043	timer
200–20F	game controller
2F8–2FF	serial port (secondary)
320–32F	hard-disk controller
378–37F	parallel port
3D0–3DF	graphics controller
3F0–3F7	diskette-drive controller
3F8–3FF	serial port (primary)

TO COMMUNICATE WITH THIS DEVICE

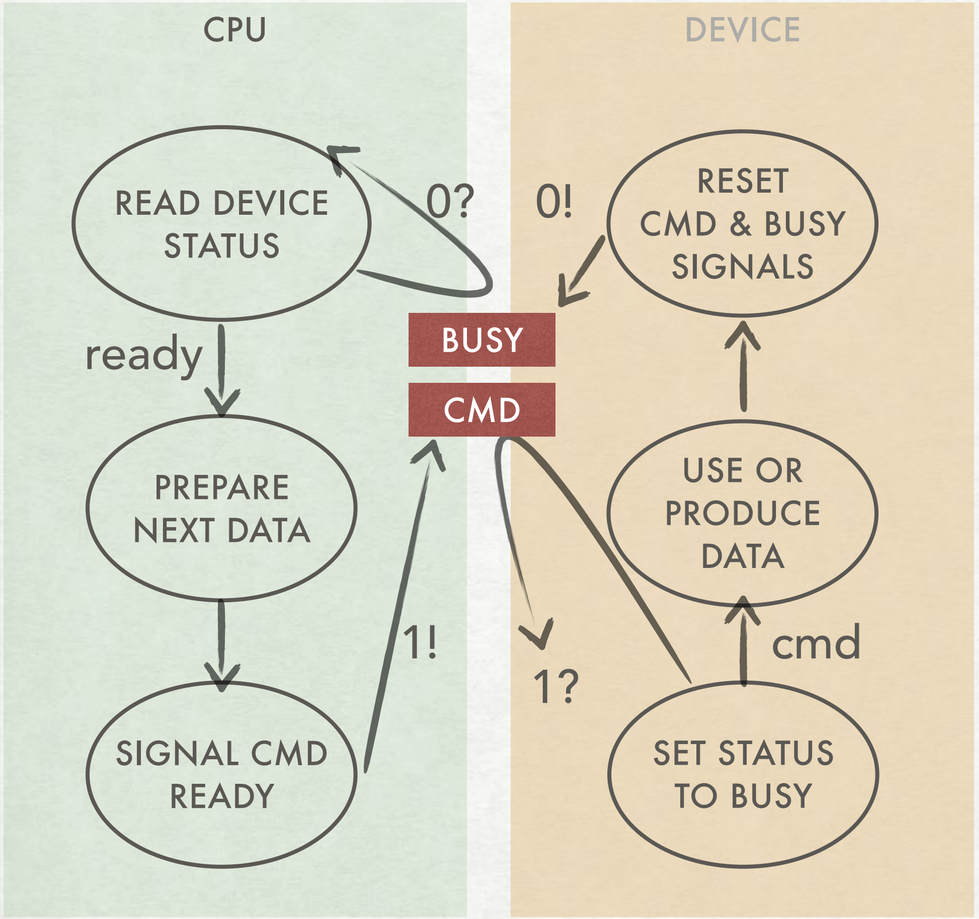
MIXES PORT I/O AND MEMORY-MAPPED I/O

COM1

TYPICALLY FOUR REGISTERS: DATA-IN, DATA-OUT, STATUS AND CONTROL REGISTERS

POLLING

I/O SYSTEMS

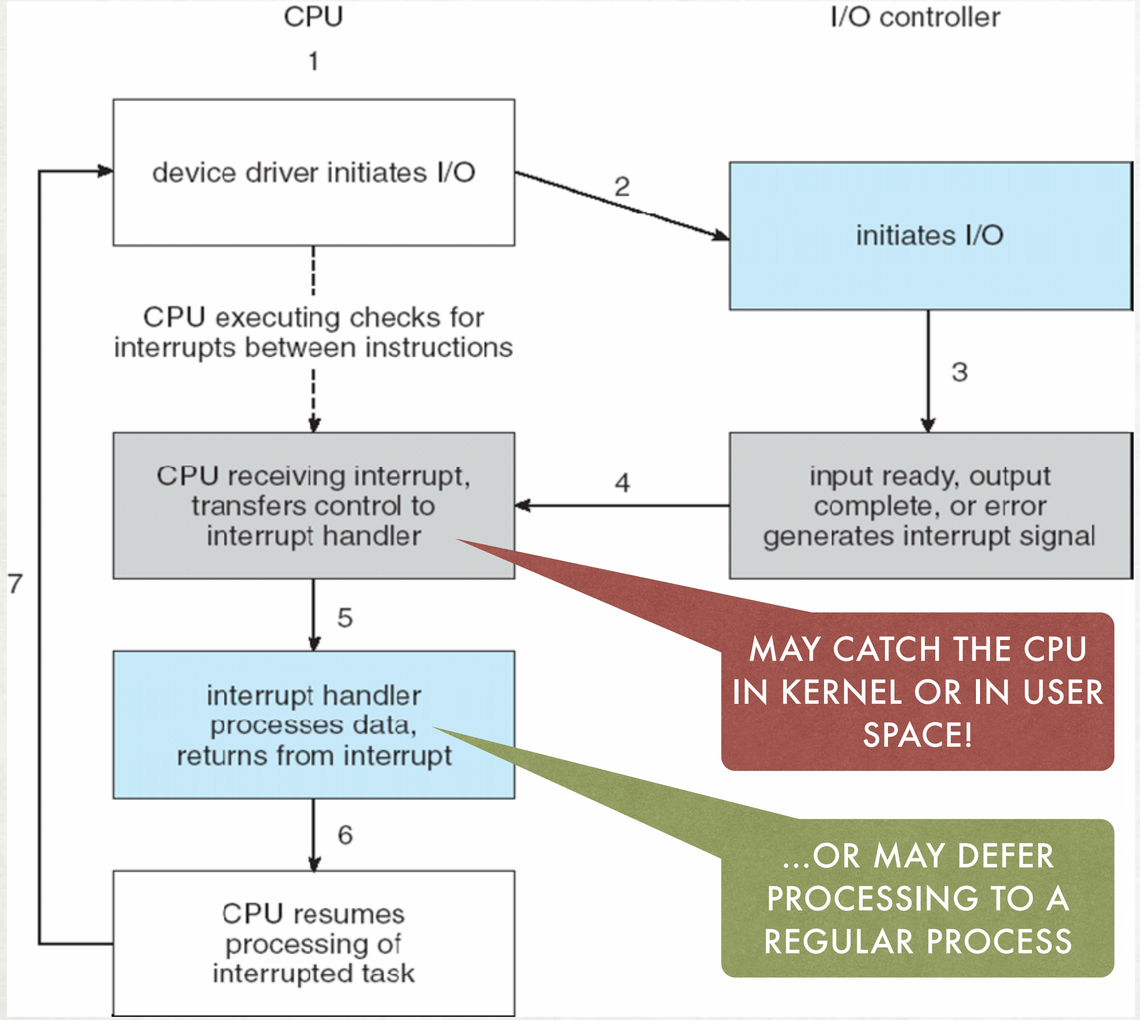


Basic Handshaking - BUSY/CMD
(logical signals - bits@addresses)

- simple handshaking protocol
- busy waiting
- inefficient for slow I/O
- *(often conflated with "programmed I/O")*

INTERRUPTS

I/O SYSTEMS



- eliminates busy wait
- CPU must support interrupts
- maskable vs. non-maskable interrupts
- interrupt vector:
  which function *handles* which device
- interrupts often have different priorities
- chaining:
  attach several *handlers* to one interrupt

INTEL PENTIUM EVENT-VECTOR TABLE

EXAMPLE – I/O SYSTEMS

vector number	description
0	divide error
1	debug exception
2	null interrupt
3	breakpoint
4	INTO-detected overflow
5	bound range exception
6	invalid opcode
7	device not available
8	double fault
9	coprocessor segment overrun (reserved)
10	invalid task state segment
11	segment not present
12	stack fault
13	general protection
14	page fault
15	(Intel reserved, do not use)
16	floating-point error
17	alignment check
18	machine check
19–31	(Intel reserved, do not use)
32–255	maskable interrupts

VECTOR ELEMENTS:
ADDRESSES OF (JUMP INSTR. TO)
INTERRUPT HANDLERS
(FIRST IN A CHAIN)

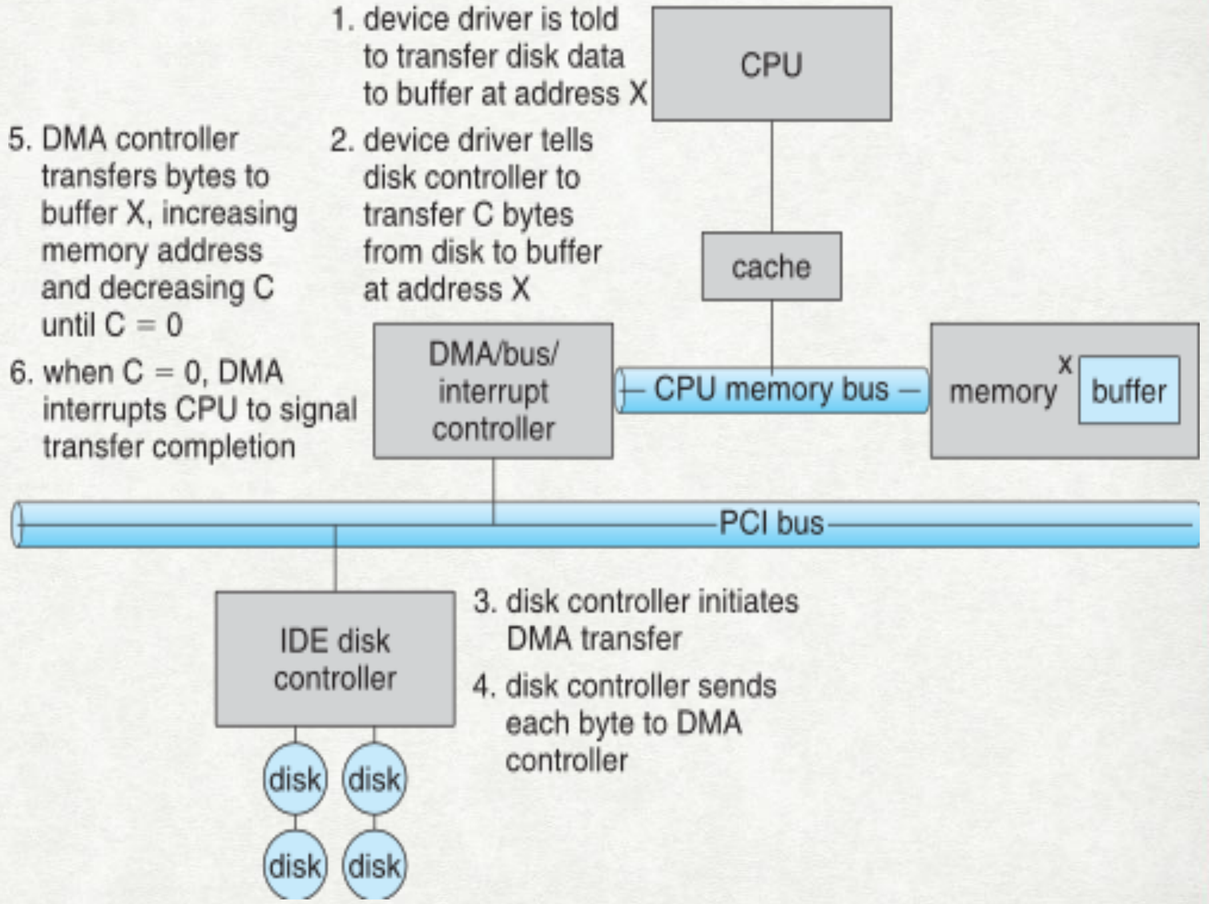
NOTE HOW
EXCEPTIONS (INTERNAL EVENTS) AND
INTERRUPTS (EXTERNAL EVENTS)
ARE MIXED

"CAN BE IGNORED"

DIRECT MEMORY ACCESS

I/O SYSTEMS

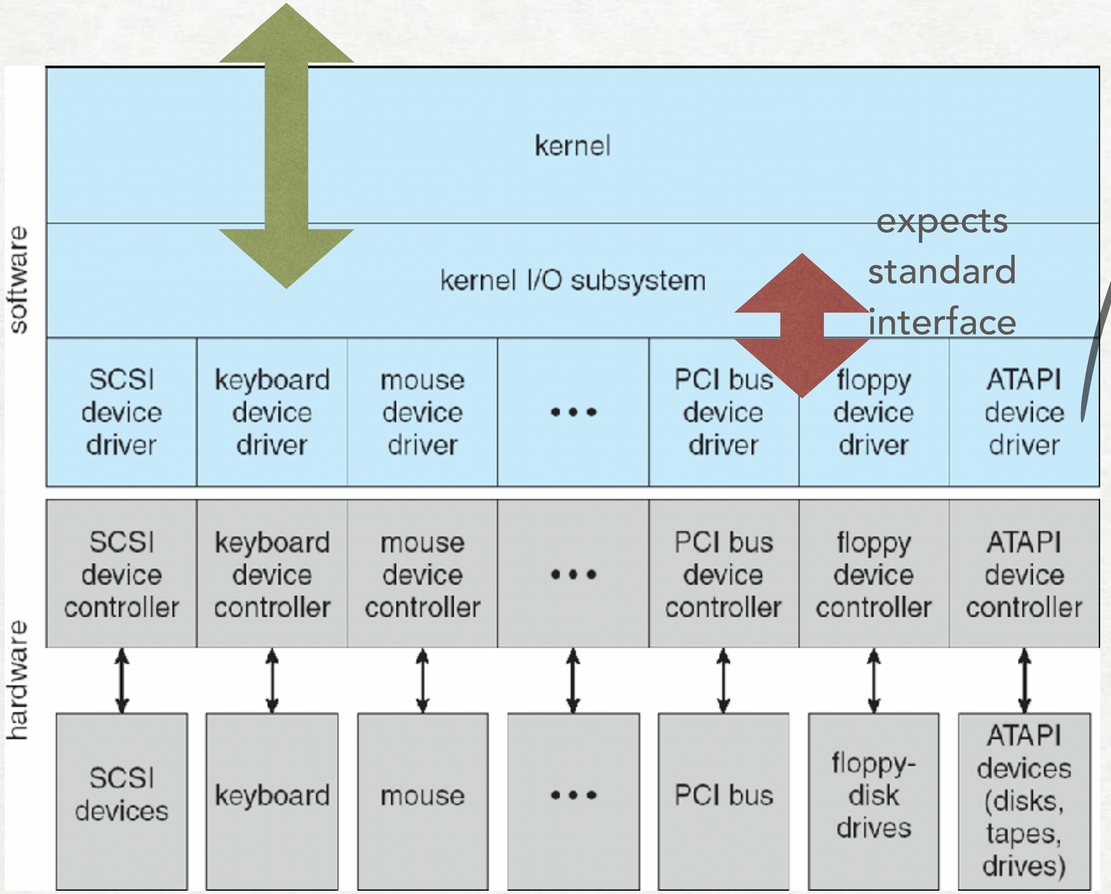
- **programmed I/O**
  (CPU transfers data one at a time)
  vs.
  DMA
  (special HW transfers data, bypass CPU)
- DMA ctrl. steals bus cycles for transfer
- Often uses interrupts
- Scatter/gather transfers possible



OS I/O INTERFACES

I/O SYSTEMS

Exposes a uniform view



- Only a few device types:
  - character vs. block transfer
  - sequential vs. random access
  - synchronous vs. asynchronous
  - sharable vs. dedicated
  - speed of operation
  - RW, RO, WO
- same interface, new devices
- can bypass kernel I/O subsystem (standard system calls):
  UNIX — see **`>man ioctl()`**

TYPES OF I/O DEVICES

I/O SYSTEMS

BLOCK DEVICES

- “disk drives”
- `read()`, `write()`, `seek()`
- raw I/O (direct I/O) or file system based
- base for memory-mapped files
- DMA

CHARACTER DEVICES

- “keyboard, mice, printer”
- character streams
- `put()`, `get()`
- sporadic
- base for line-at-the-time libraries, editors

NETWORK DEVICES

- “network sockets”
- `select()`
- connect to local socket/remote address
- varying implementations

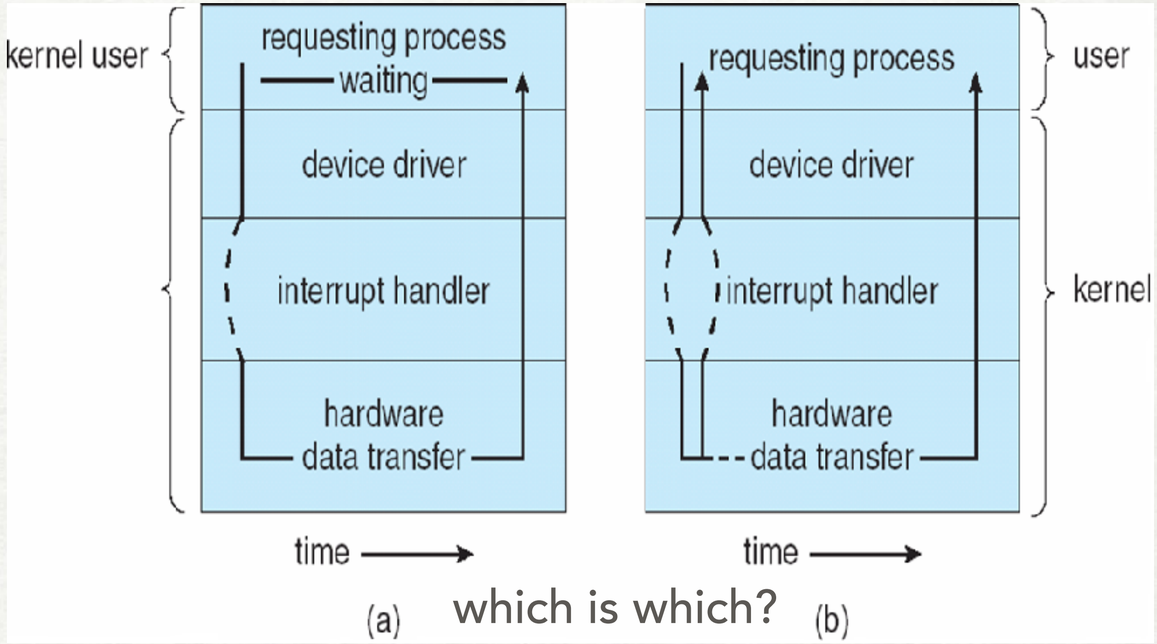
CLOCKS AND TIMERS

- current time
- elapsed time
- set alarm (interrupt)
- `ioctl()`

SYNCHRONIZATION IN I/O

I/O SYSTEMS

- blocking vs. non-blocking calls
  (remember the process states?)
  - blocking system calls are easier to understand and program with
- synchronous vs. asynchronous requests
- non-blocking *vs.* asynchronous ?
  (return immediately with partial result) vs.
  (initiate and complete in the future)



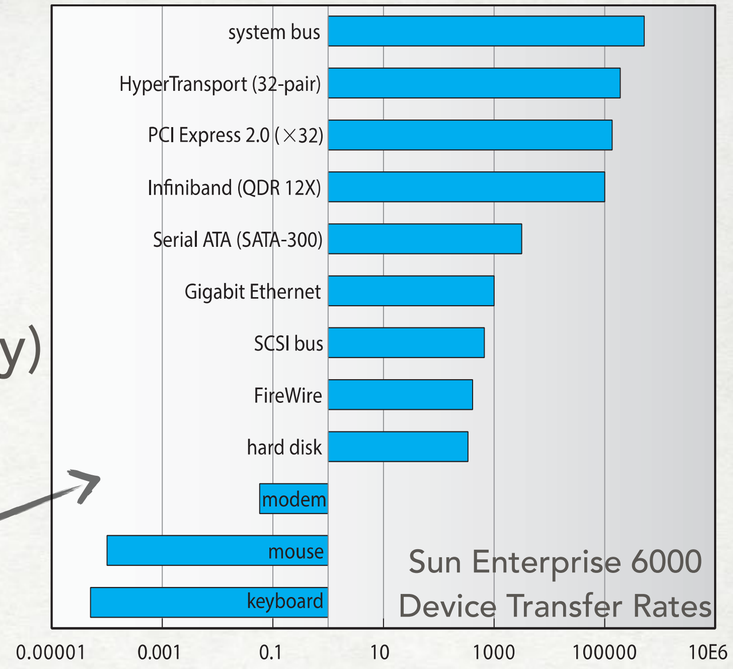
which is which?

Timeout
`> man select`

See also **Vectored I/O** (textbook) – for multiple buffers, scatter/gather methods

`>man readv, writev`

KERNEL I/O SUBSYSTEM

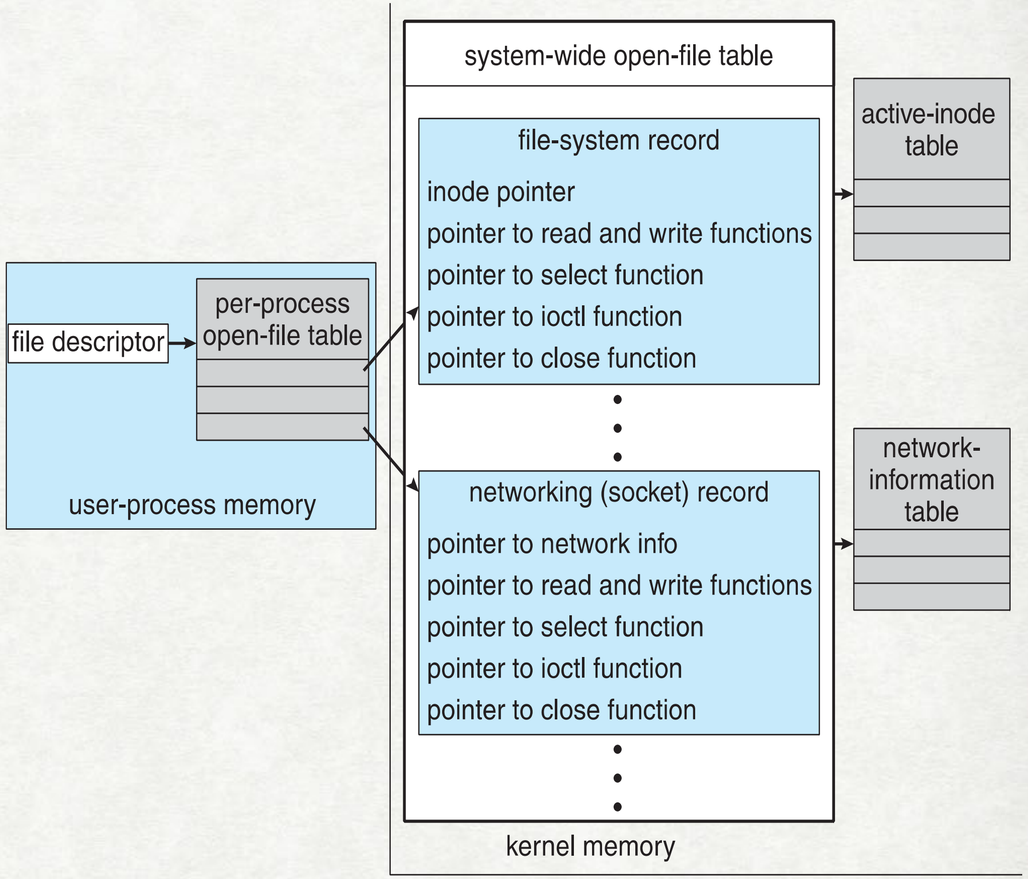


- I/O **scheduling** — reorder access requests (efficiency, priority)
- **Buffering** — memory area for transfer — cope with:
  1. speed mismatch (see also double-buffering)
  2. data size mismatch: e.g. packets in networking
  3. “copy semantics”: no changes to data before the transfer is done
- **Caching** — memory holding a copy, for performance (can combine with buffering)
- **Spooling** — hold output to a device serving one request at a time (e.g. printer) or use “device reservation”
- **Error handling and I/O protection** — prevent, handle illegal operations (syscalls)

KERNEL DATA STRUCTURES

I/O SYSTEMS

- track the state of I/O components:
  open files, network connections, device states,…
- many internal data structures tracking:
  buffers, memory allocation, requests, …
- object-oriented and modular:
  present a unified API to the programmer

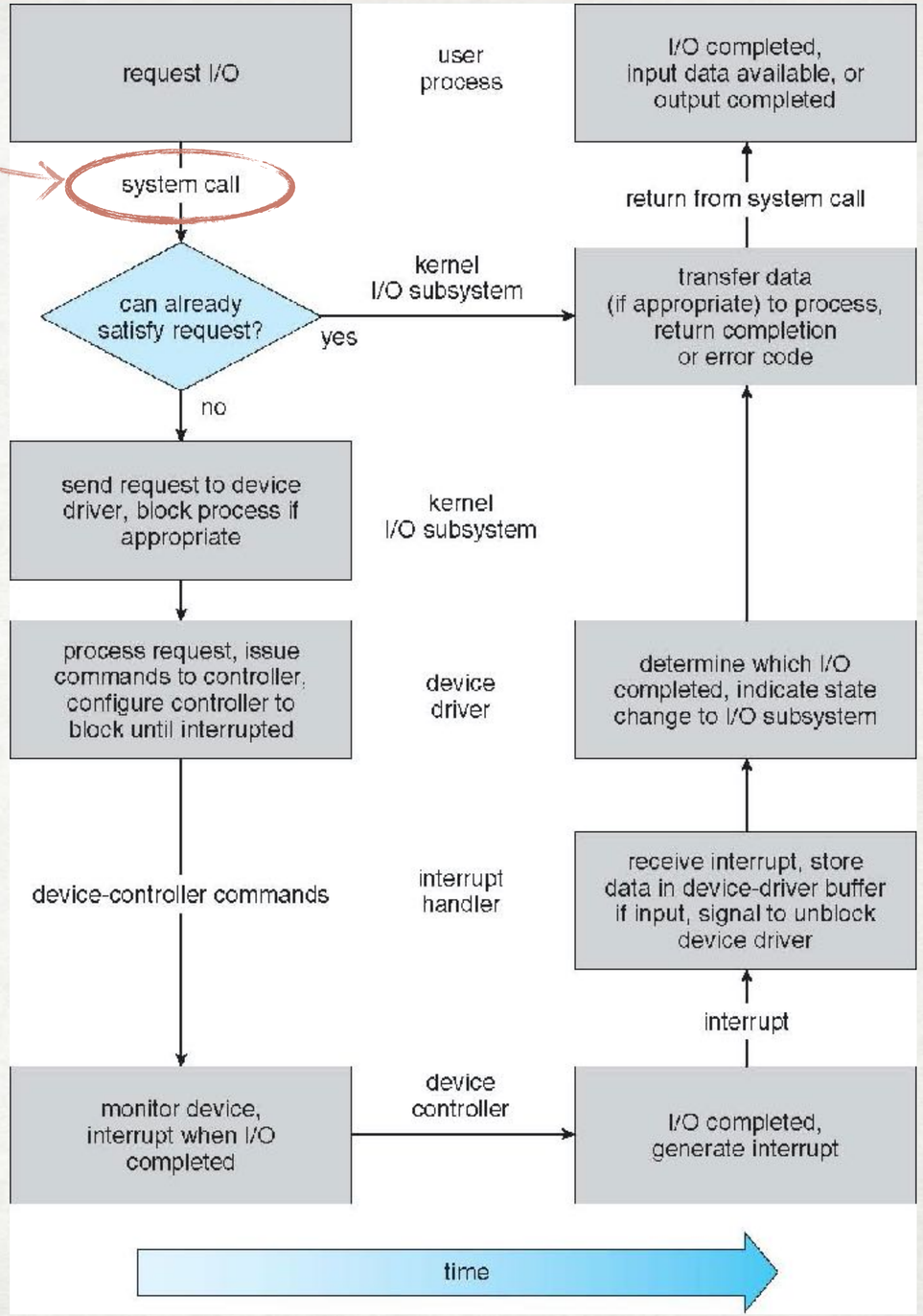


Example: UNIX I/O kernel structure

FROM I/O REQUESTS TO HARDWARE OPERATIONS

- check parameters
- map names/ids to devices
- issue requests to drivers/handle completed I/O
- block/unblock calling process
- manage the memory involved (buffers, cache)

➡ Example:
blocking read from open file descriptor



END OF MODULE 8