Input / Output
&
Real-time Scheduling

Chapter 5,

Chapter 7.5

I/O Content

- I/O Software
- Device controllers
- Memory-mapped I/O
- DMA & interrupts briefly
- I/O software layers and drivers
- Disks and RAID
- Clock
- Briefly about Power management

I/O Software

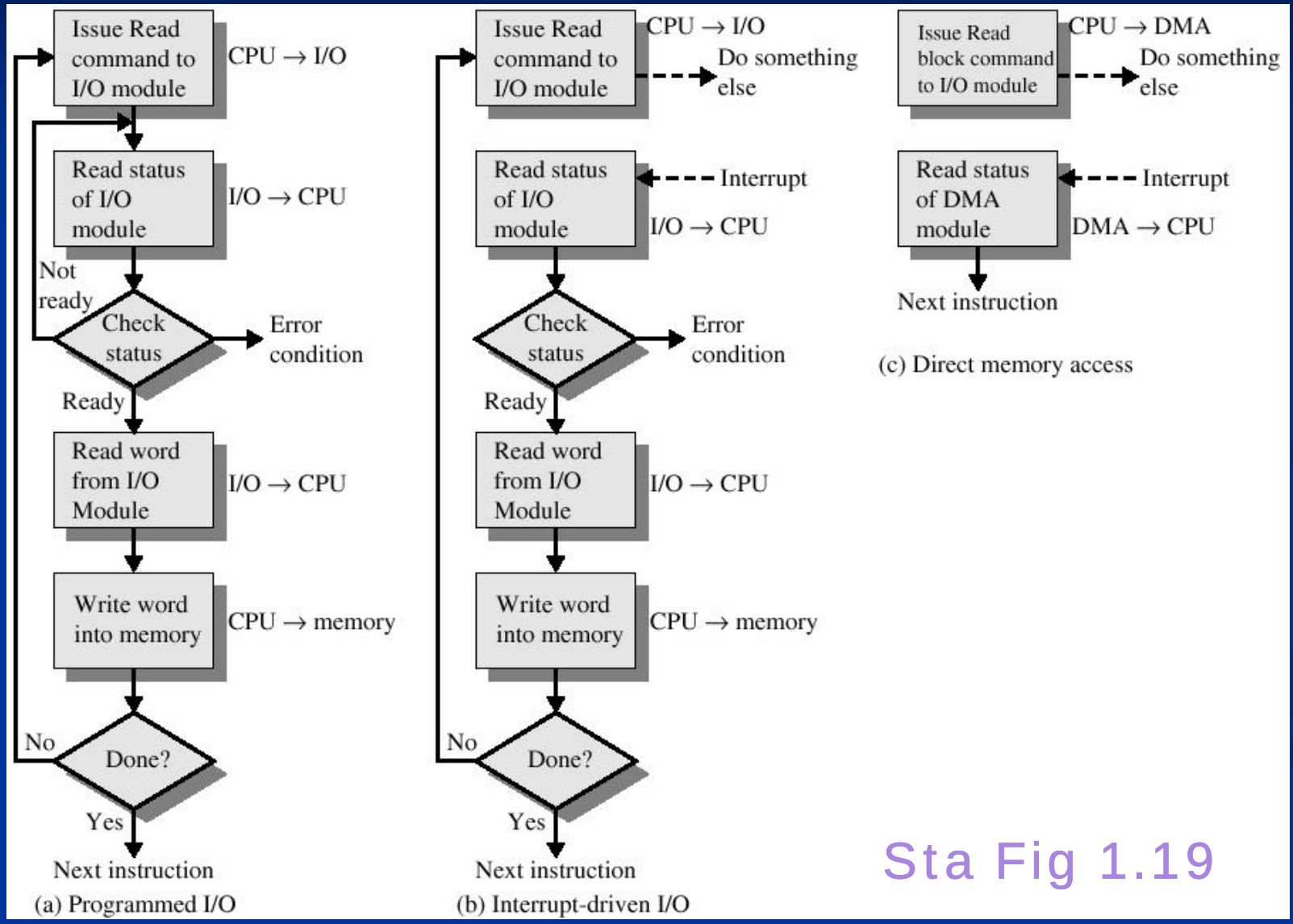
Goals of I/O Software

- Device independence
  - programs can access any I/O device
  - without specifying device in advance
- Uniform naming
  - name of a file or device a string or an integer
  - not depending on which machine
- Error handling
  - handle as close to the hardware as possible
- Synchronous vs. asynchronous transfers
  - blocked transfers vs. interrupt-driven
- Buffering
  - data coming off a device cannot be stored in final destination
- Sharable vs. dedicated devices
  - disks are sharable
  - tape drives would not be

I/O Communication Techniques

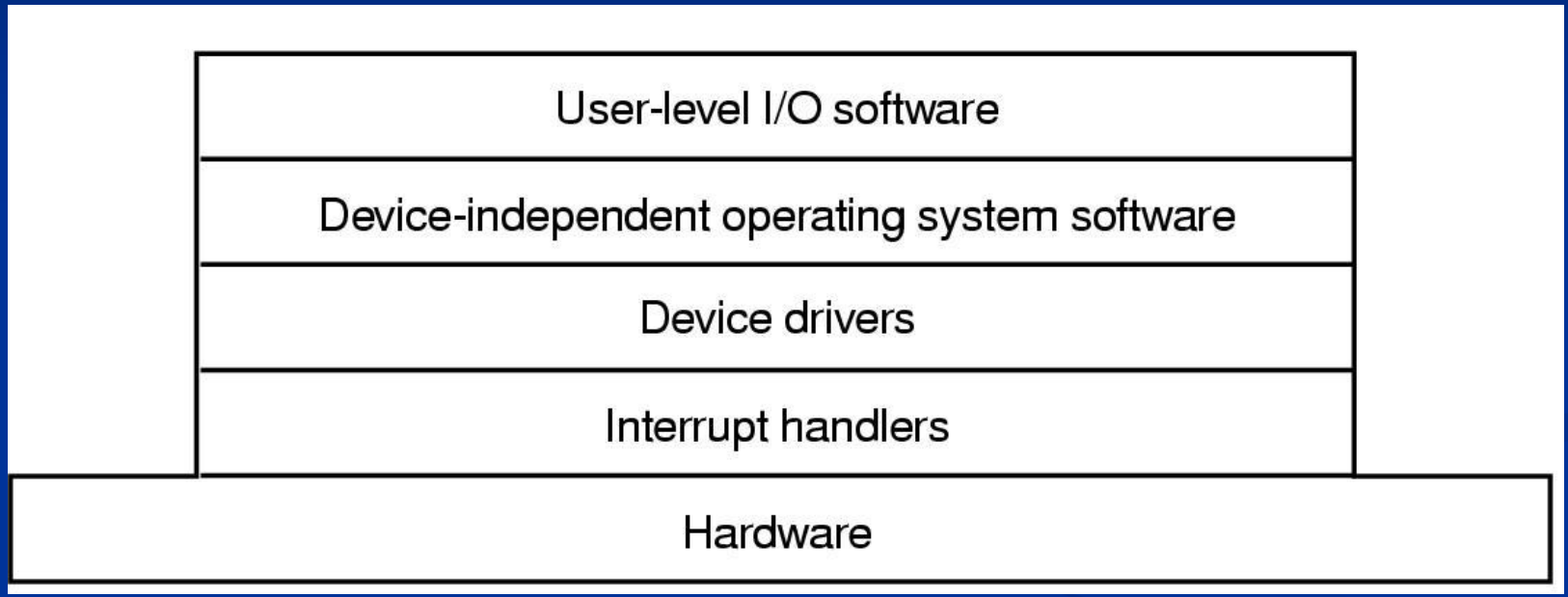
- Programmed I/O
  - CPU continuously studies the control registers in loop
  - busy-wait
- Interrupt-driven I/O
  - CPU gives task to controller and continues with something else
  - Controller makes an interrupt, when ready
  - CPU moves the data between controller and memory
- Direct Memory Access, DMA
  - DMA-controller knows how to access memory directly
  - Interrupt only, when the transfer is fully done

I/O Communication Techniques



Sta Fig 1.19

I/O Software Layers



- Layers of the I/O Software System

Interrupt Handlers

- Interrupts are best hidden
  - have driver starting an I/O operation block until interrupt notifies of completion
- Interrupt procedure does its task
  - then unblocks driver that started it
- The interrupt handling software must perform several steps after hardware interrupt completed
  - list on next page

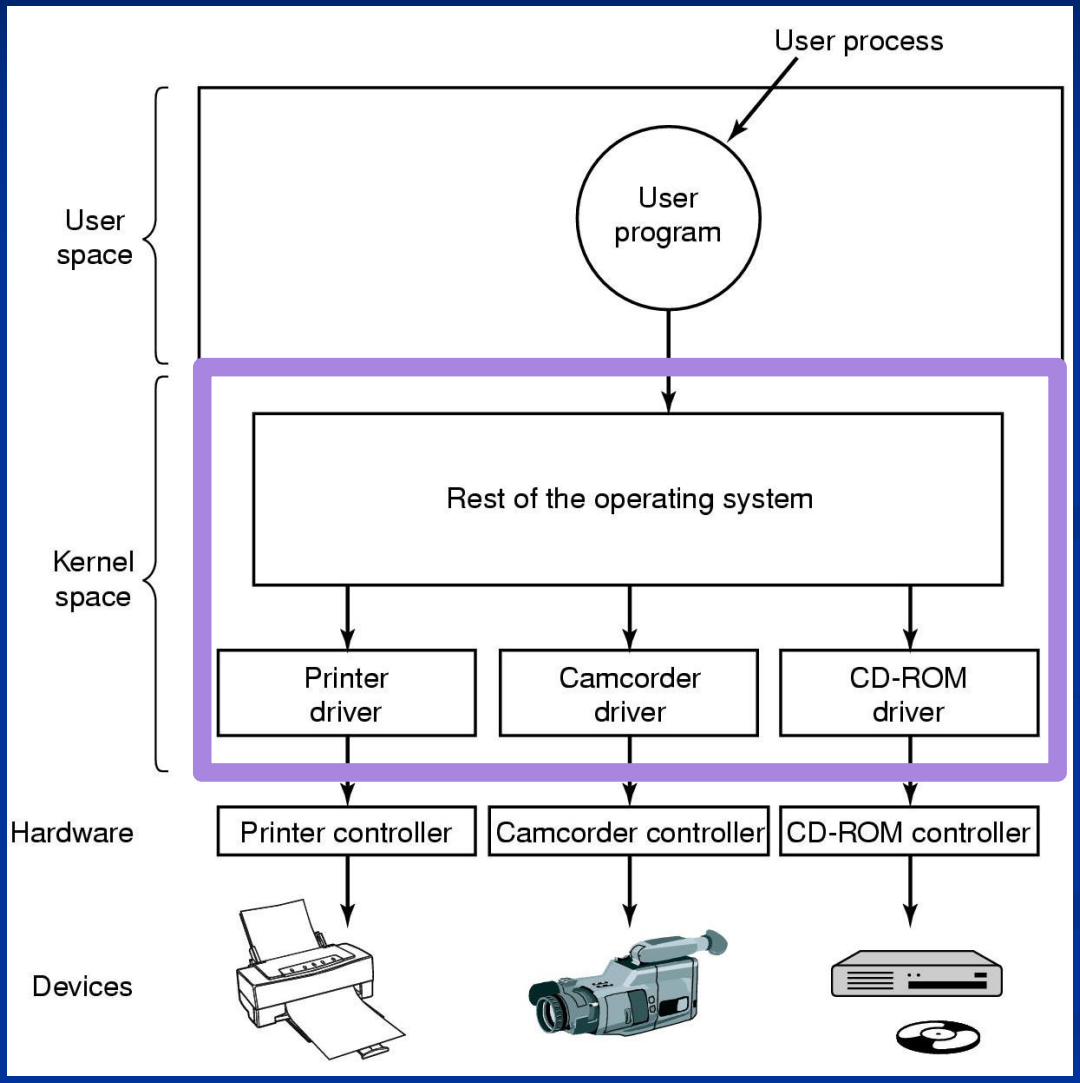
Interrupt Handlers

1. Save regs not already saved by interrupt hardware
2. Set up context for interrupt service procedure
3. Set up stack for interrupt service procedure
4. Ack interrupt controller, reenable interrupts
5. Copy registers from where saved
6. Run service procedure
7. (Choose which process to run next)
8. Set up MMU context for process to run next
9. Load new process' registers
10. Start running the new process

Device Drivers

- Each device (type) has its own device driver
  - Code written by the manufacturer of the device
  - Still part of the OS kernel!
- OS designer
  - Architecture must allow such installation
  - Well-defined model of interactions and driver's tasks
  - Two standard interfaces: block devices, character devices
- How to add or remove drivers
  - Recompilation of the kernel, if single binary file
  - Dynamically loaded during execution
- Driver code alternatives
  - Blocking vs non blocking
  - Reentrant code
  - No system calls allowed, but some specific kernel procedures available

Logical Positioning of Device Drivers



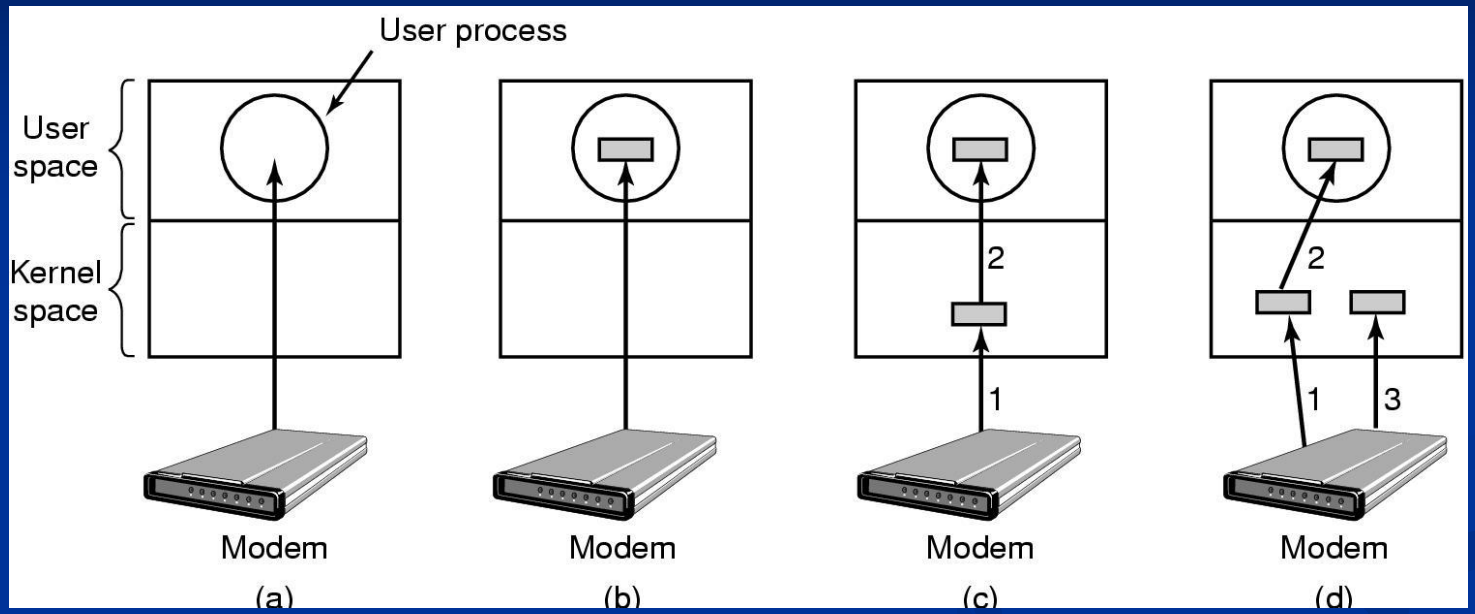
Set up device registers
check status

Wake up driver
when I/O completed

Device-Independent I/O Software

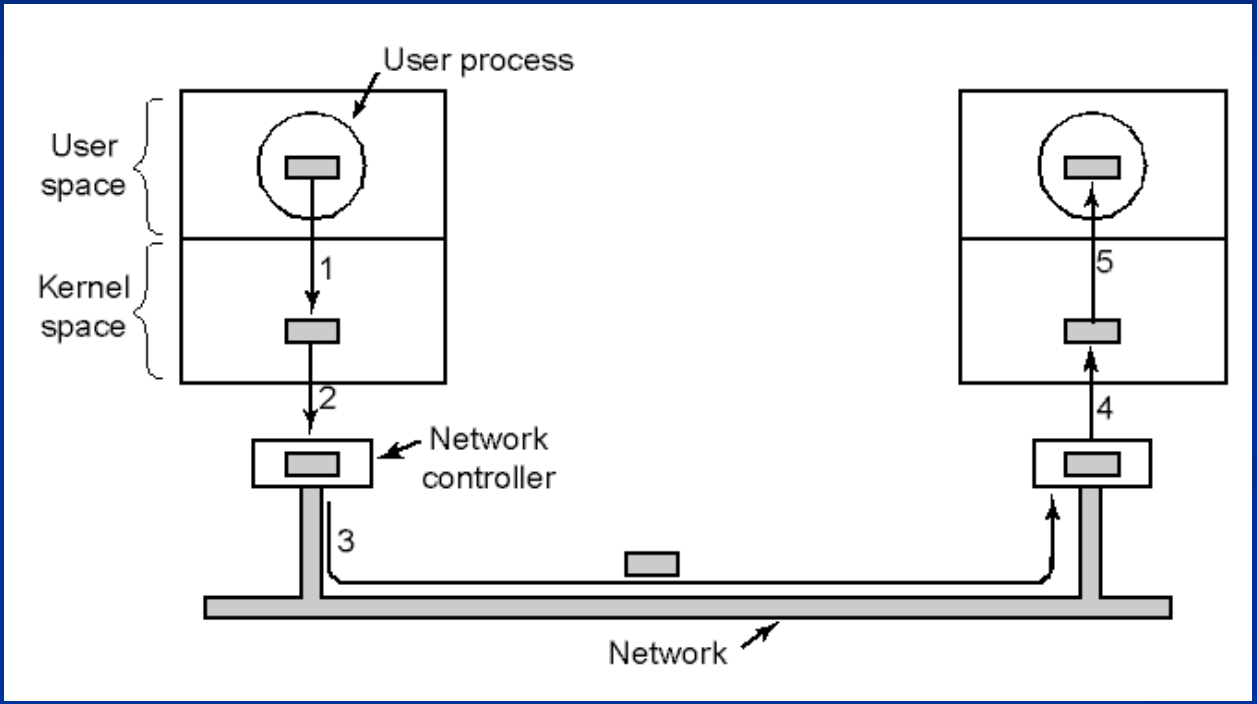
- Uniform interfacing for device drivers
  - To avoid modifying kernel code for new drivers
  - Driver contains a function pointer table for kernel to locate the functions needed in the interface
  - Naming of I/O devices, protection
- Buffering
  - Used for efficiency; inefficient, if too many
- Error reporting
  - Error device-specific, framework for error handling general
- Allocating and releasing dedicated devices
- Providing a device-independent block size

Buffering alternatives



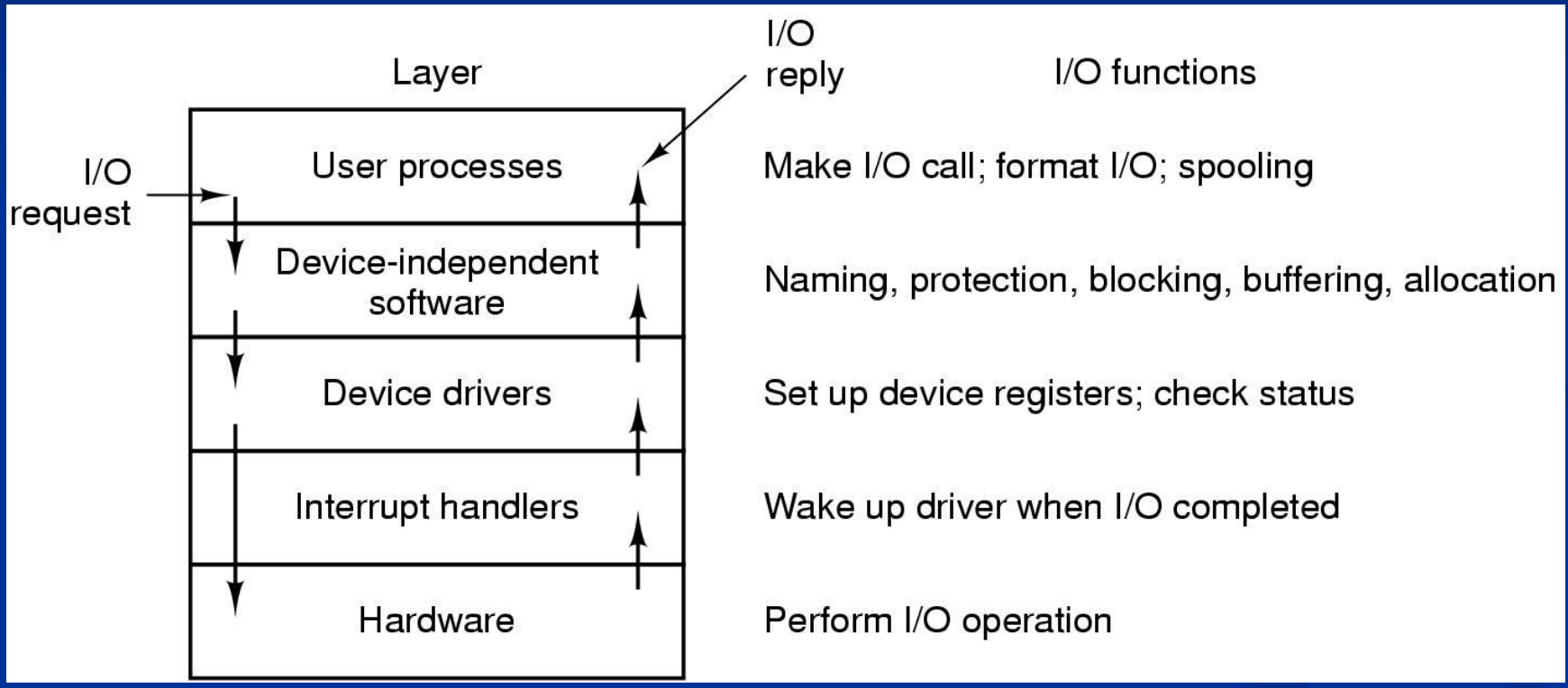
(a) Unbuffered input
(b) Buffering in user space
(c) Buffering in the kernel followed by copying to user space
(d) Double buffering in the kernel

Buffering and networking



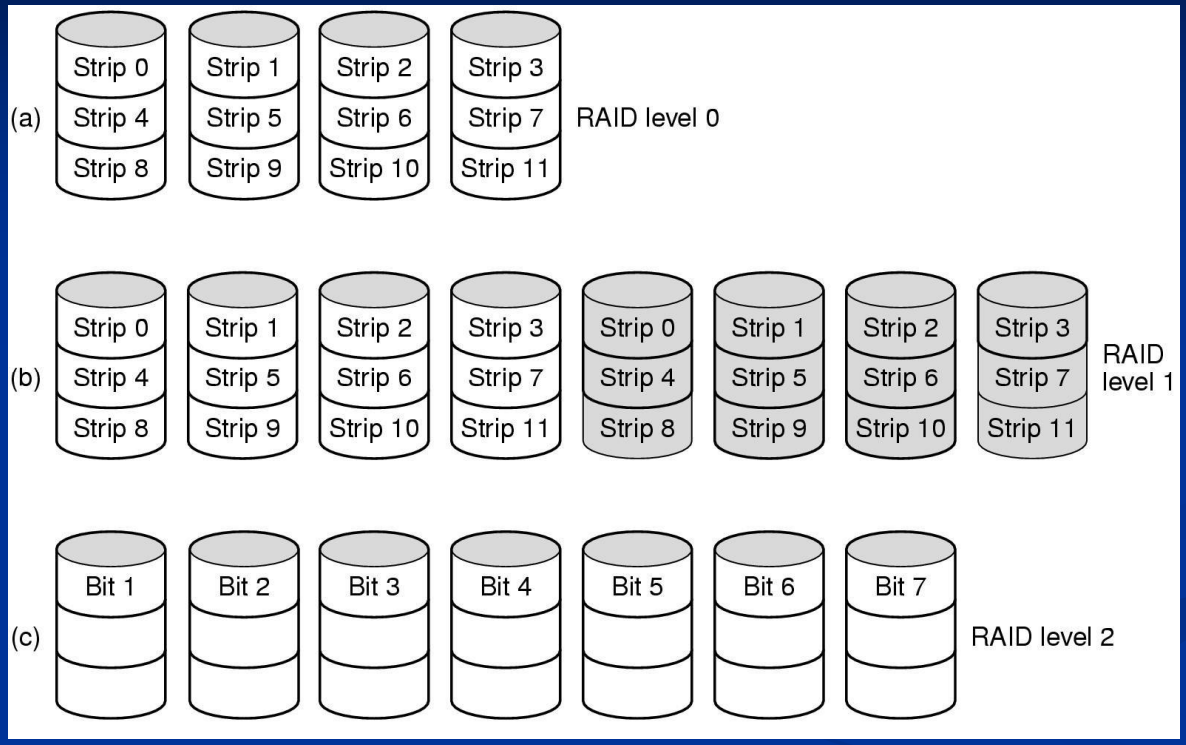
Networking may involve many copies

Layers of the I/O system



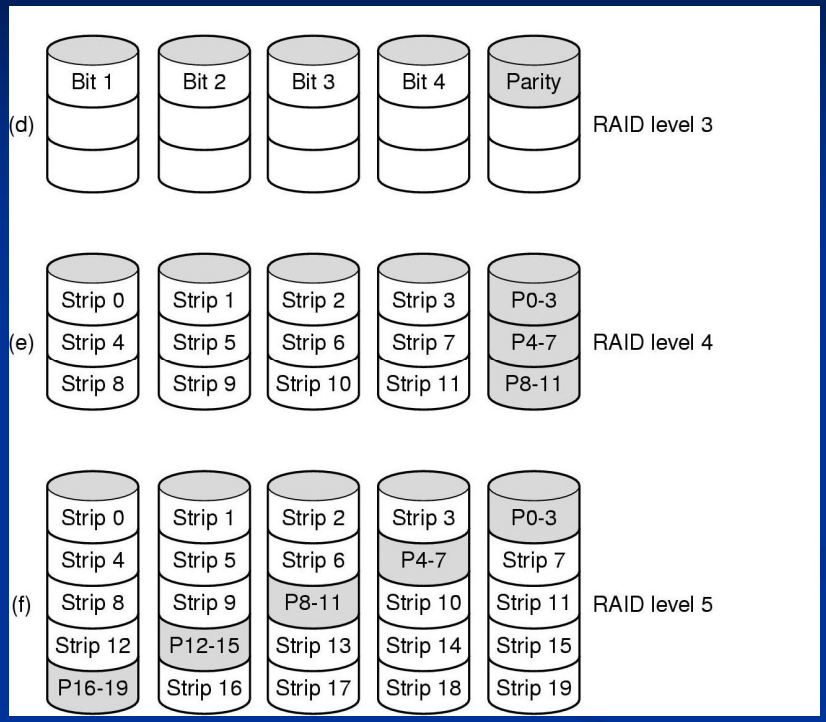
Disks

RAID 0,1,2



- 0: strips, no redundancy
- 1: Mirroring (with strips)
- 2: bits, uses Hamming code

RAID 3,4,5



- 3: bits, uses Parity bit
- 4: strips, with strip-for-strip parity on dedicated disk
- 5: strips, distributing parity strips uniformly

RAID table

Tbl 11.4 [Stal 05]

Category	Level	Description	Disks required	Data availability	Large I/O data transfer capacity	Small I/O request rate
Striping	0	Nonredundant	N	Lower than single disk	Very high	Very high for both read and write
Mirroring	1	Mirrored	$2N$, $3N$, etc.	Higher than RAID 2, 3, 4, or 5; lower than RAID 6	Higher than single disk for read; similar to single disk for write	Up to twice that of a single disk for read; similar to single disk for write
Parallel access	2	Redundant via Hamming code	$N + m$	Much higher than single disk; higher than RAID 3, 4, or 5	Highest of all listed alternatives	Approximately twice that of a single disk
	3	Bit-interleaved parity	$N + 1$	Much higher than single disk; comparable to RAID 2, 4, or 5	Highest of all listed alternatives	Approximately twice that of a single disk
Independent access	4	Block-interleaved parity	$N + 1$	Much higher than single disk; comparable to RAID 2, 3, or 5	Similar to RAID 0 for read; significantly lower than single disk for write	Similar to RAID 0 for read; significantly lower than single disk for write
	5	Block-interleaved distributed parity	$N + 1$	Much higher than single disk; comparable to RAID 2, 3, or 4	Similar to RAID 0 for read; lower than single disk for write	Similar to RAID 0 for read; generally lower than single disk for write
	6	Block-interleaved dual distributed parity	$N + 2$	Highest of all listed alternatives	Similar to RAID 0 for read; lower than RAID 5 for write	Similar to RAID 0 for read; significantly lower than RAID 5 for write

RAID at CS dept. (S2007)

- File servers: group ja fs
  - 6 RAID5 servers using 7 (+ 1 hot spare), each server total 1,6 TB
  - /group, /fs
  - */fs-0, /fs-1, /fs-2, /fs-3* (user directories divided to these)
  - Similar, but smaller disks: *courier.cs.helsinki.fi* (mailserver) - 0,9 TB

RAID overhead 14% (1/7)

RAID overhead 20% (1/5)

- 4 RAID5 servers using 5 disks (each 146 GB), total 550 GB
  - *backup* for backuping user files from fs
    - copy each /fs-i here and stream using 2 tape robots
    - SW RAID 5 in Linux kernel, since the robot controller and hardware RAID controller were not working together correctly
  - *db.cs.helsinki.fi* (database server)
  - *winserver.cs.helsinki.fi* (Windows 2003 Terminal Server)
  - *bodbacka.cs.helsinki.fi*

RAID at CS dept. (S2007)

- RAID 1 using 2 disks times 2 RAID-1 nameserver (Hydra)

RAID overhead 50% (1/2)
redundancy overhead 50%

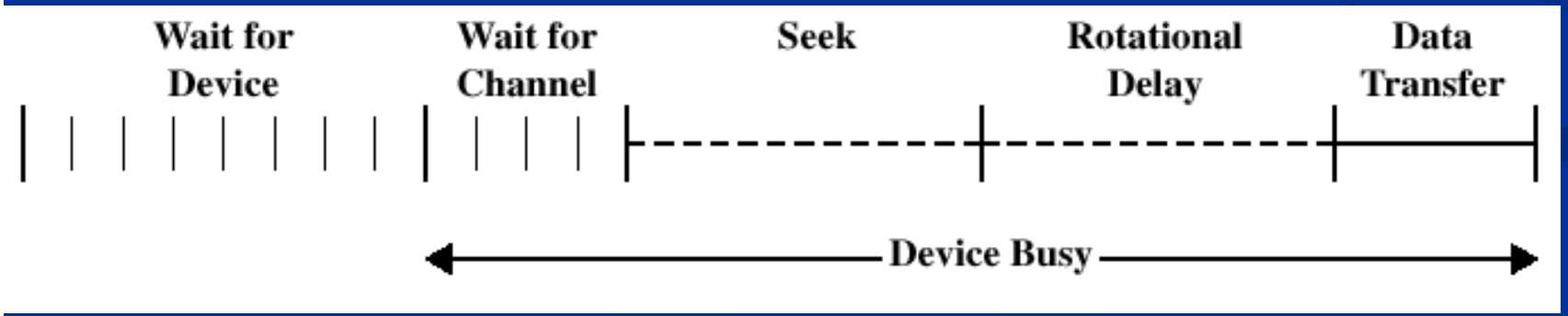
- RAID 5 using 4 disks, total 0.9 TB (Bioinformatiikka)
- RAID 5 using 4 disks, total 0.45 TB (b-course)

RAID overhead 25% (1/4)

- 2 RAID 5 servers using 5 disks, each total 0.6 TB (Vera, Chuck)
  - Mainly for number crunching (32 GB), disk space is not crucial

Disk Arm Scheduling

- Time required to read or write a disk block determined by 3 factors
  1. Seek time
  2. Rotational delay
  3. Actual transfer time
- Seek time dominates
- Error checking is done by controllers



Disk Arm Scheduling Algorithms

- Goal: Reduce the mean seek time
- Random? FIFO? PRI? LIFO?
  - Do not consider the current arm position
- Improvement: order the requests
  - SSF - shortest seek first
  - SCAN, LOOK – elevator algorithm
- Assumption:
  - The real disk geometry matches the virtual, assumed, geometry – this may not be true with disks where the controller can do error correction with replacements sectors.

Shortest Seek First (SSF)

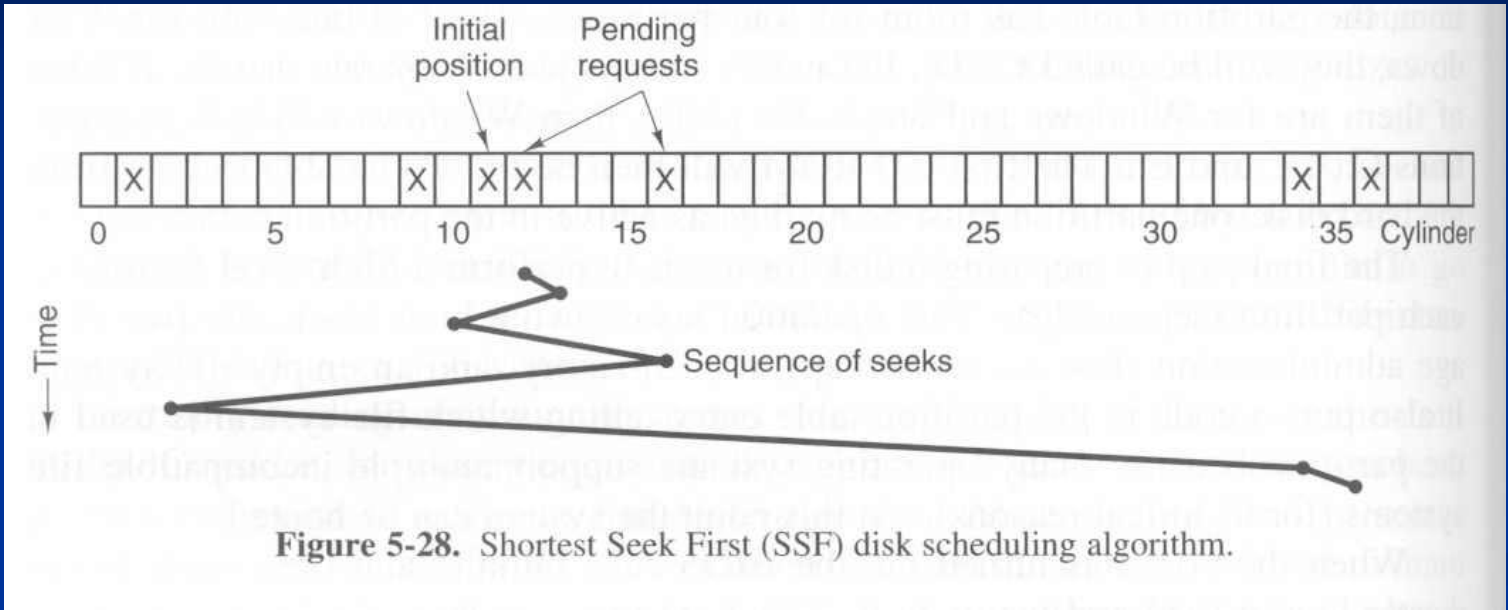


Figure 5-28. Shortest Seek First (SSF) disk scheduling algorithm.

- Handle the closest requeste next, minimize seek time
- On a heavily loaded disk, the arm has a tendency to stay in the middle of the disk most of the time
- Old requests on either extreme can suffer with long delays
- Fairness?

Elevator (SCAN, LOOK)

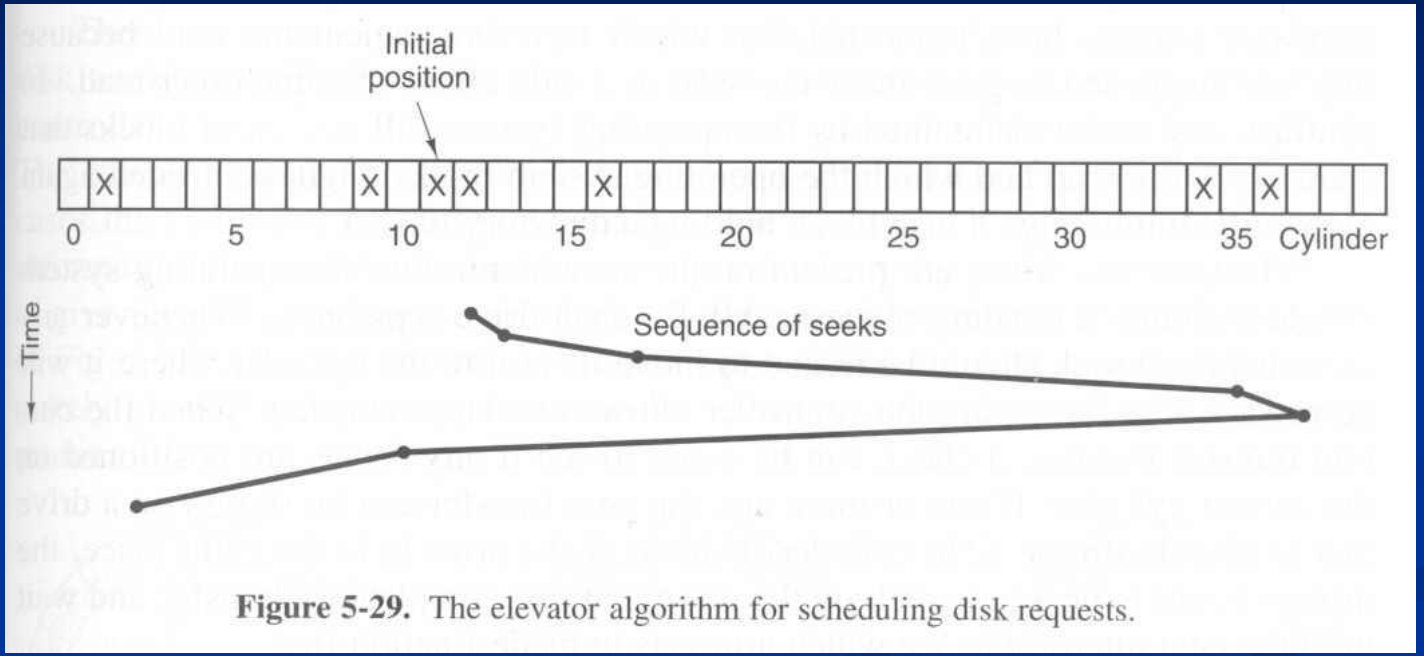
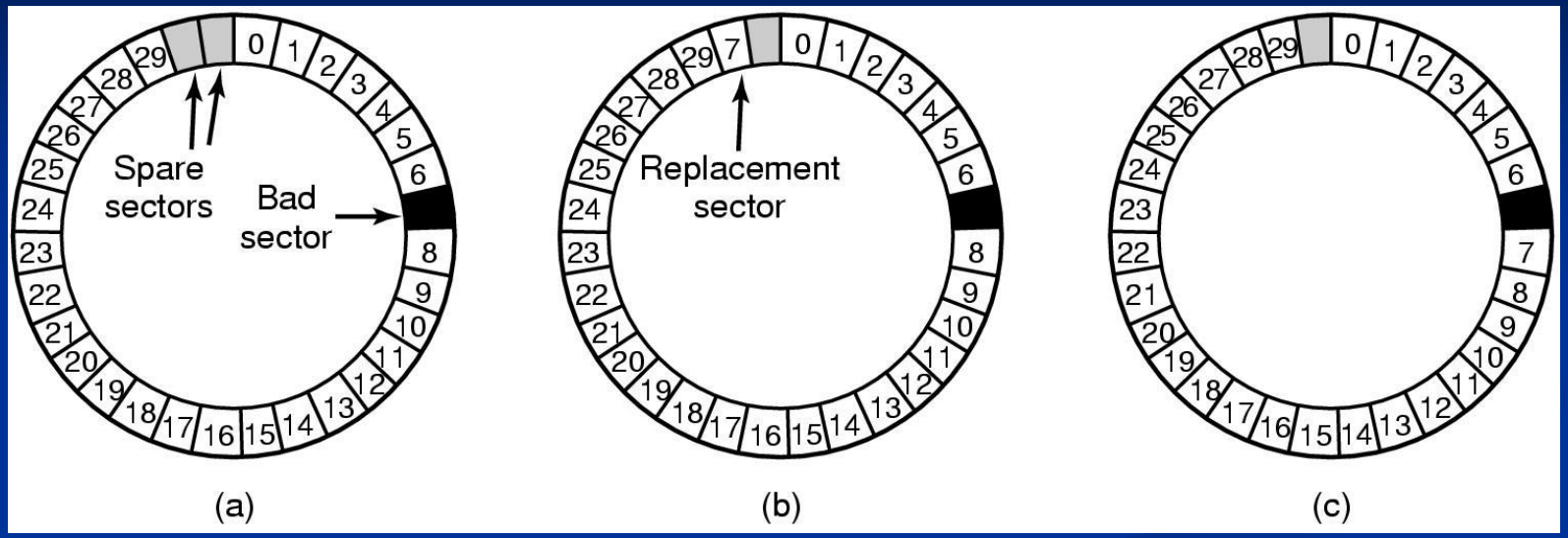


Figure 5-29. The elevator algorithm for scheduling disk requests.

- Move the arm only to one direction (UP or DOWN) and serve the closest request in that direction, move the other way only when no requests in that direction

Error Handling



- (a) A disk track with a bad sector
- (b) Substituting a spare for the bad sector
  - Reading the whole track takes two turns
- (b) Shifting all the sectors to bypass the bad one
  - Reading continuous, but more bookkeeping for the controller

Error Handling

- In controller:
  - The methods on previous page are suitable
  - Need remapping tables
- In operating system (or its driver):
  - Bad sector information from the disk
  - Remove these from free list and allocated files, can f.eg. create a special bad-blocks file (not in the file hierarchy)
- Backup and bad sectors:
  - Sector level backup will try to read the bad sectors also
  - File level backup must not find the bad block file

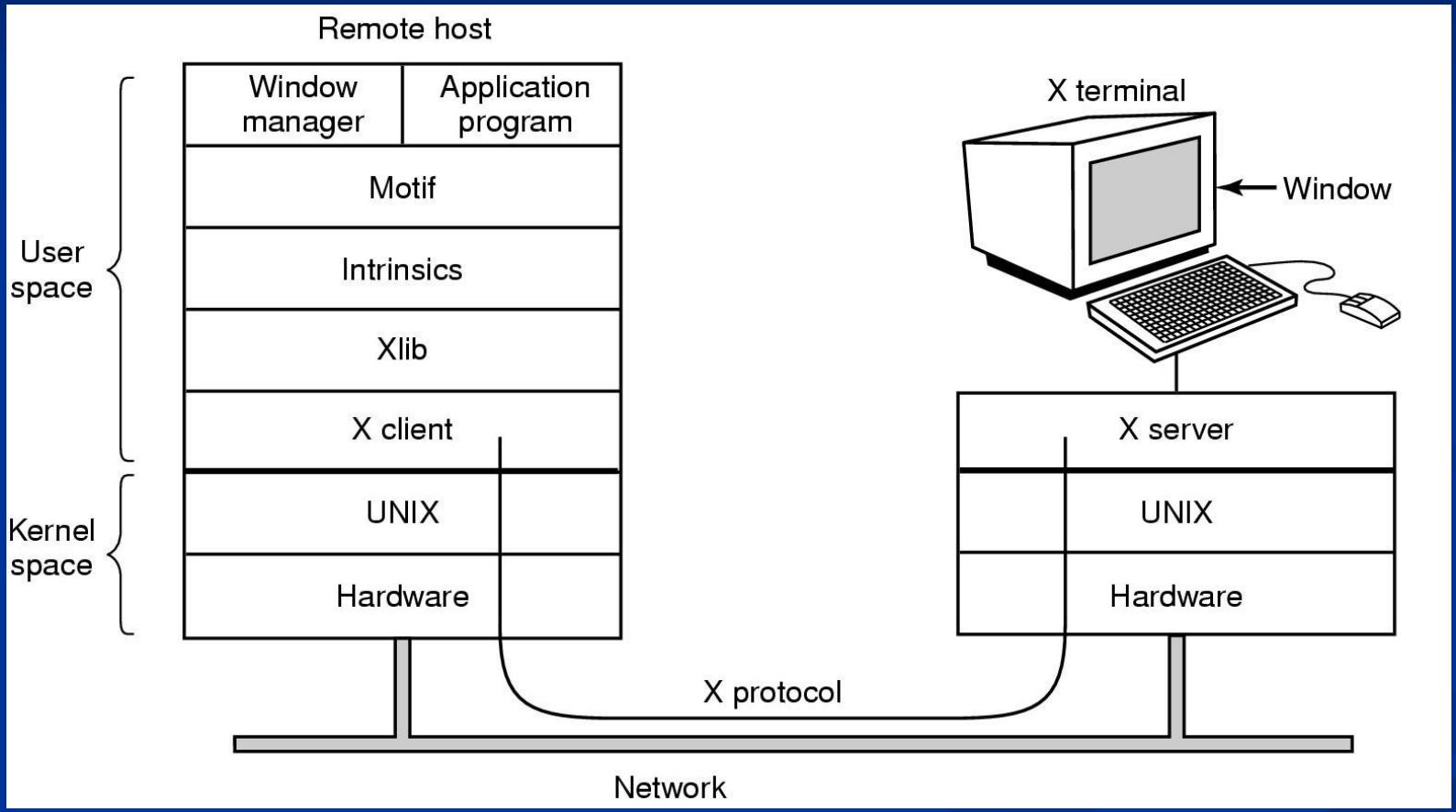
User interfaces

- Input: Keyboard, mouse
- Output: monitor
- Text windows, Graphical interface

- X Window system
  - Portable
  - Entirely in user space
  - X server on the user's computer manage the screen and windows
  - X client (on the remote computer) uses one of the windows

Network Terminals

X Windows (1)



Clients and servers in the M.I.T. X Window System

Skeleton of an X Windows application program

```
#include <X11/Xlib.h>
#include <X11/Xutil.h>

main(int argc, char *argv[])
{
        Display disp;                                   /* server identifier */
        Window win;                                     /* window identifier */
        GC gc;                                          /* graphic context identifier */
        XEvent event;                                   /* storage for one event */
        int running = 1;

        disp = XOpenDisplay("display_name");            /* connect to the X server */
        win = XCreateSimpleWindow(disp,  ... );         /* allocate memory for new window */
        XSetStandardProperties(disp, ...);              /* announces window to window mgr */
        gc = XCreateGC(disp, win, 0, 0);                /* create graphic context */
        XSelectInput(disp, win, ButtonPressMask | KeyPressMask | ExposureMask);
        XMapRaised(disp, win);                  /* display window; send Expose event */

        while (running) {
                XNextEvent(disp, &event);       /* get next event */
                switch (event.type) {
                    case Expose:        ...;    break;  /* repaint window */
                    case ButtonPress:   ...;    break;  /* process mouse click */
                    case Keypress:      ...;    break;  /* process keyboard input */
                }
        }

        XFreeGC(disp, gc);                      /* release graphic context */
        XDestroyWindow(disp, win);              /* deallocate window's memory space */
        XCloseDisplay(disp);                    /* tear down network connection */
}
```

Clocks / timers

- Hardware: ticks
- Software:
  - Maintain time of day
  - Prevent processes running longer than allowed
  - Account CPU usage
  - Handle alarm system call
  - Provide watchdog timers
  - Profiling, monitoring, statistics
- Soft timers avoid interrupts, not precise

Power management

- Important on portable, battery-powered computers
- Reduce energy consumption
  - Turn off unused parts (OS)
  - Degrade quality (application program)

Device	Li et al. (1994)	Lorch and Smith (1998)
Display	68%	39%
CPU	12%	18%
Hard disk	20%	12%
Modem		6%
Sound		2%
Memory	0.5%	1%
Other		22%

Figure 5-45. Power consumption of various parts of a notebook computer.

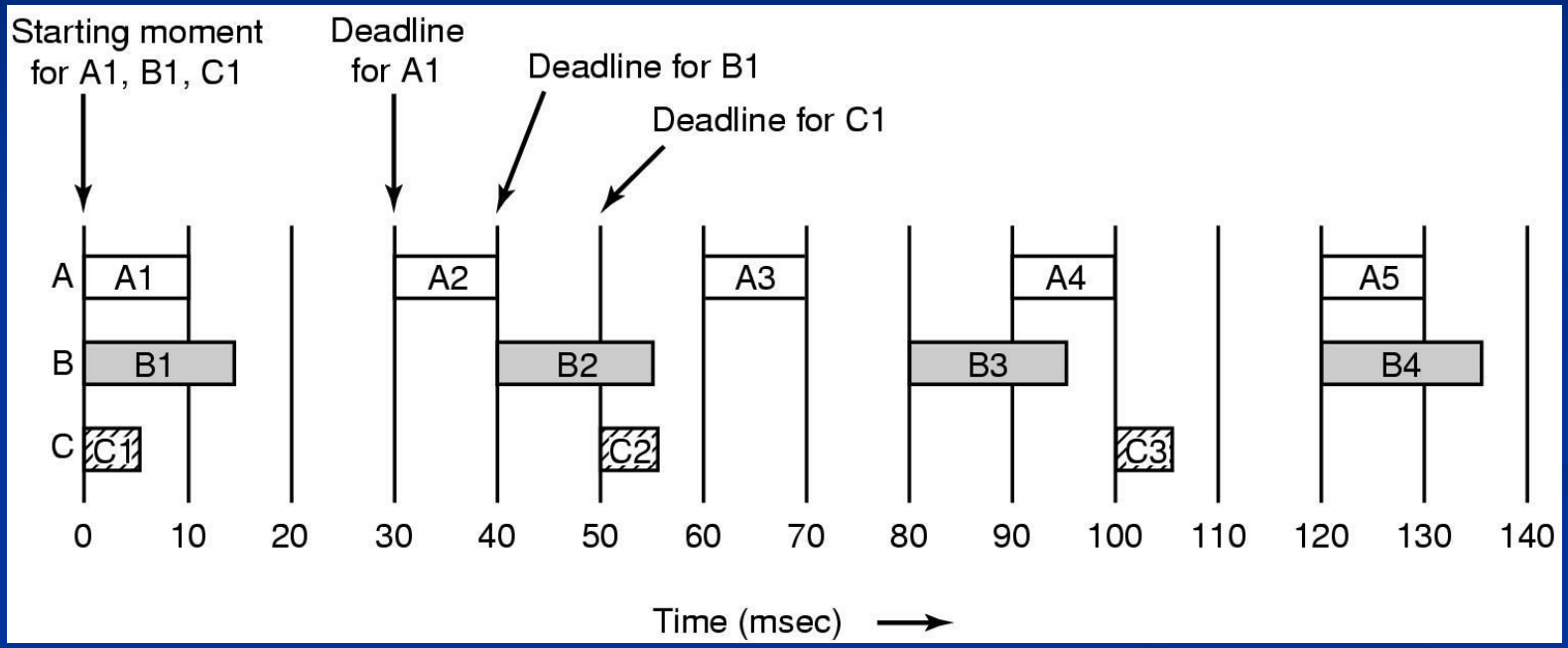
Real-time scheduling

Section 7.5. Multimedia Process Scheduling

Real-time systems

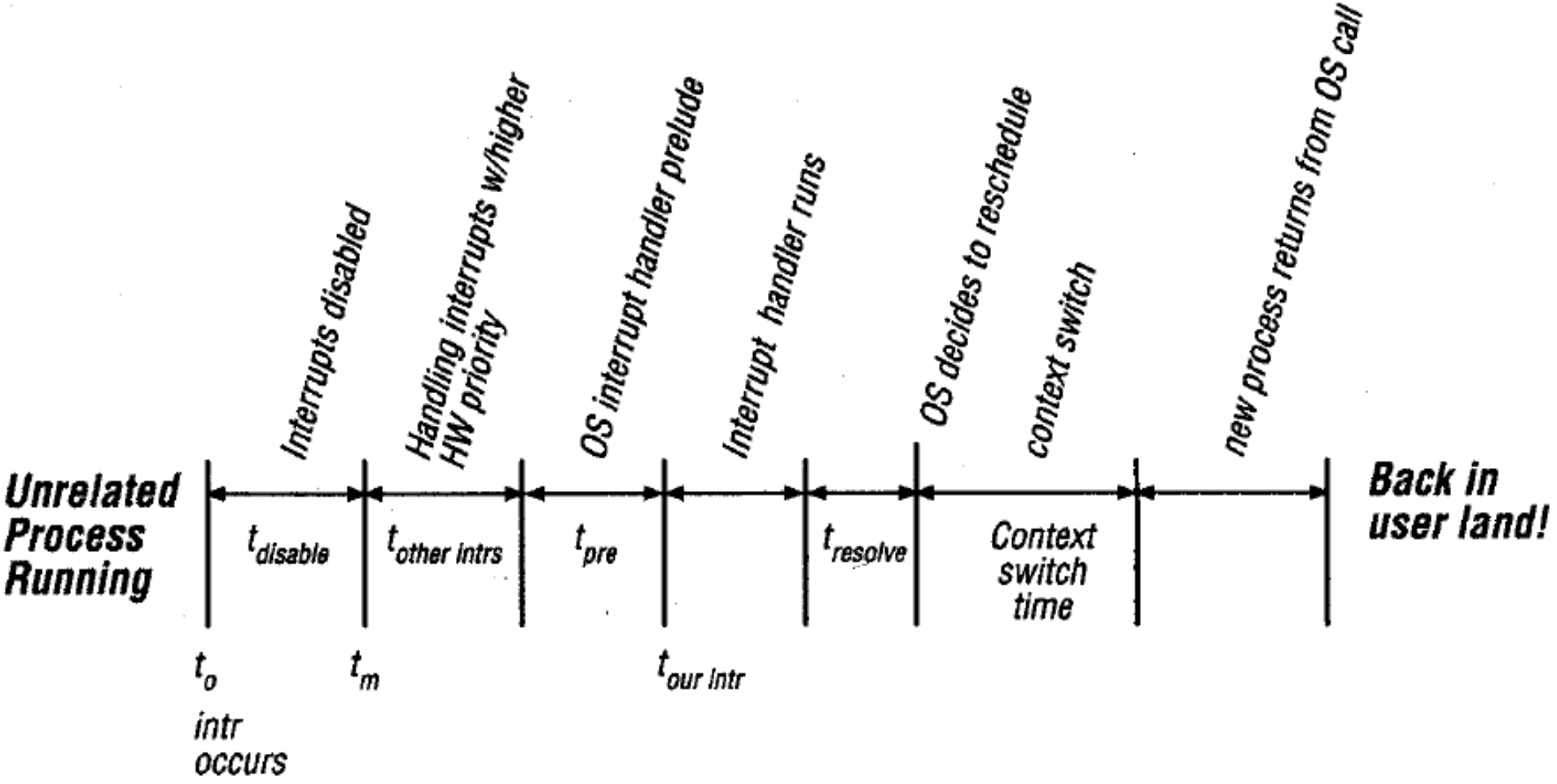
- In addition to the correct results, they must be reached within a given time limit (not too early, not too late)
- Reacts to external events (and time)
- Terms:
  - Hard Real-time vs. Soft Real-time
  - Periodic vs aperiodic
  - Static vs dynamic scheduling
- Requirements:
  - Configurable, deterministic, reliablity, responsiveness,
  - Programmers decide characteristics of the rt processes

Multimedia Process Scheduling



- Periodic processes displaying a movie
- Frame rates and processing requirements may be different for each movie

Interrupt handling delays



RT scheduling

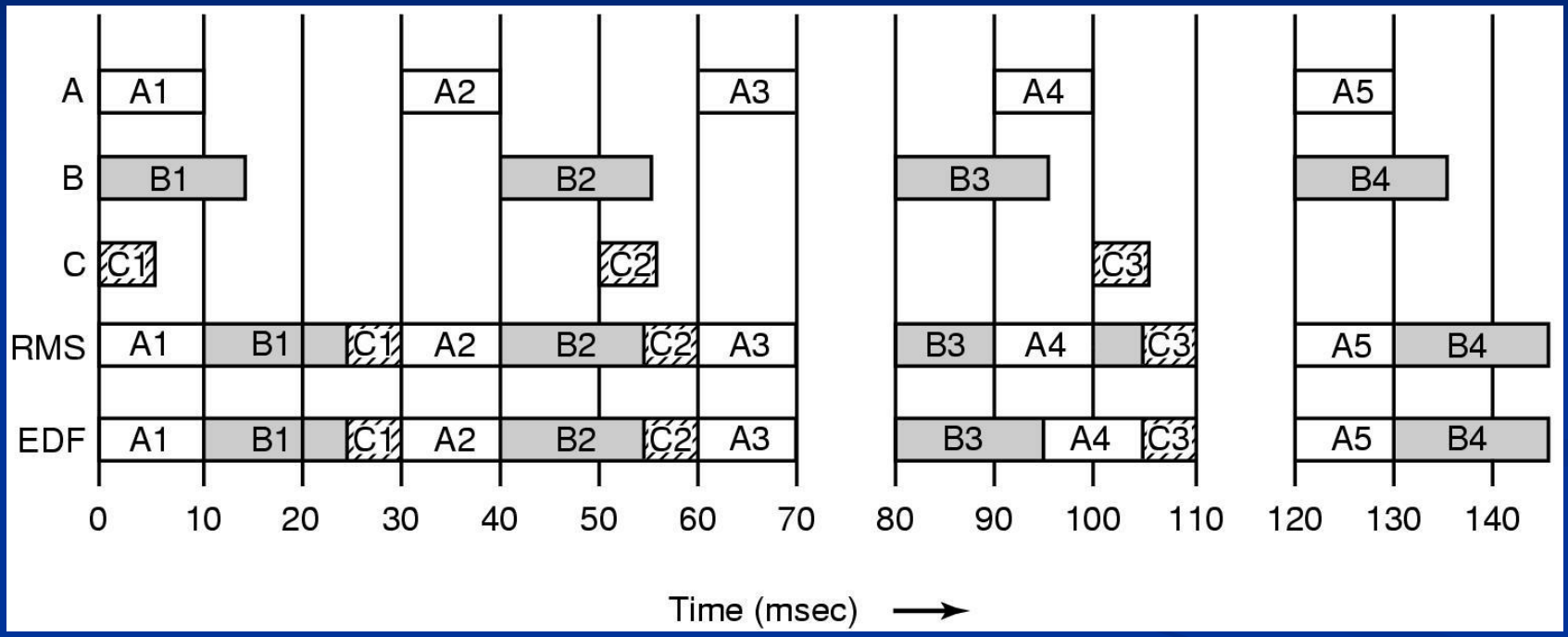
- Timely execution the most important issue
  - No fairness, no minimising the response time
- OS can only guess the actual execution time
  - Programmer gives some estimates, period gives hints
- Quite often: just schedule rt-tasks fast
  - High priority, predetermined schedule plan, others can suffer
- Two main alternative for dynamic scheduling
  - Earliest Deadline First (EDF)
  - Rate Monotonic Scheduling (RM)

Rate Monotonic Scheduling

Used for processes which meet these conditions

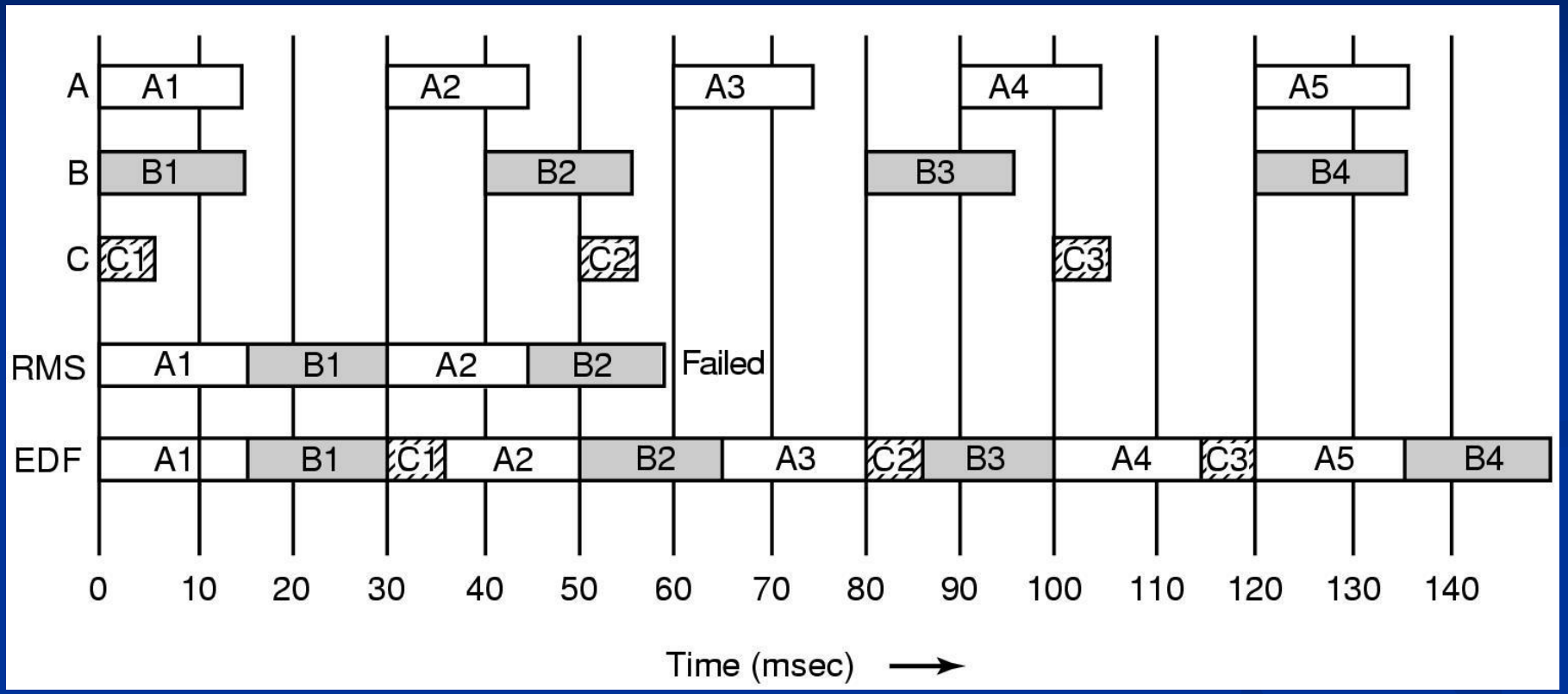
1. Each periodic process must complete within its period
2. No process dependent on any other process
3. Each process needs same CPU time each burst
4. Any nonperiodic processes have no deadlines
5. Process preemption occurs instantaneously, no overhead

Real Time Scheduling



- Real Time Scheduling algorithms
  - RMS
  - EDF

Real Time Scheduling



Another example of real-time scheduling with RMS and EDF

Schedulability test

- Obviously for any schedule, the schedulable utilisation:

$$U_i = C_1/T_1 + C_2/T_2 + .. + C_n/T_n <= 1$$

- Sufficient condition for schedulability using RM scheduling

$$C_1/T_1 + C_2/T_2 + .. + C_n/T_n <= n(2^{1/n} - 1)$$

- N is number of tasks, the upper limit for the value is

$$\ln 2 \sim 0.693 \rightarrow \text{for any } i \quad U_i < 0.693$$

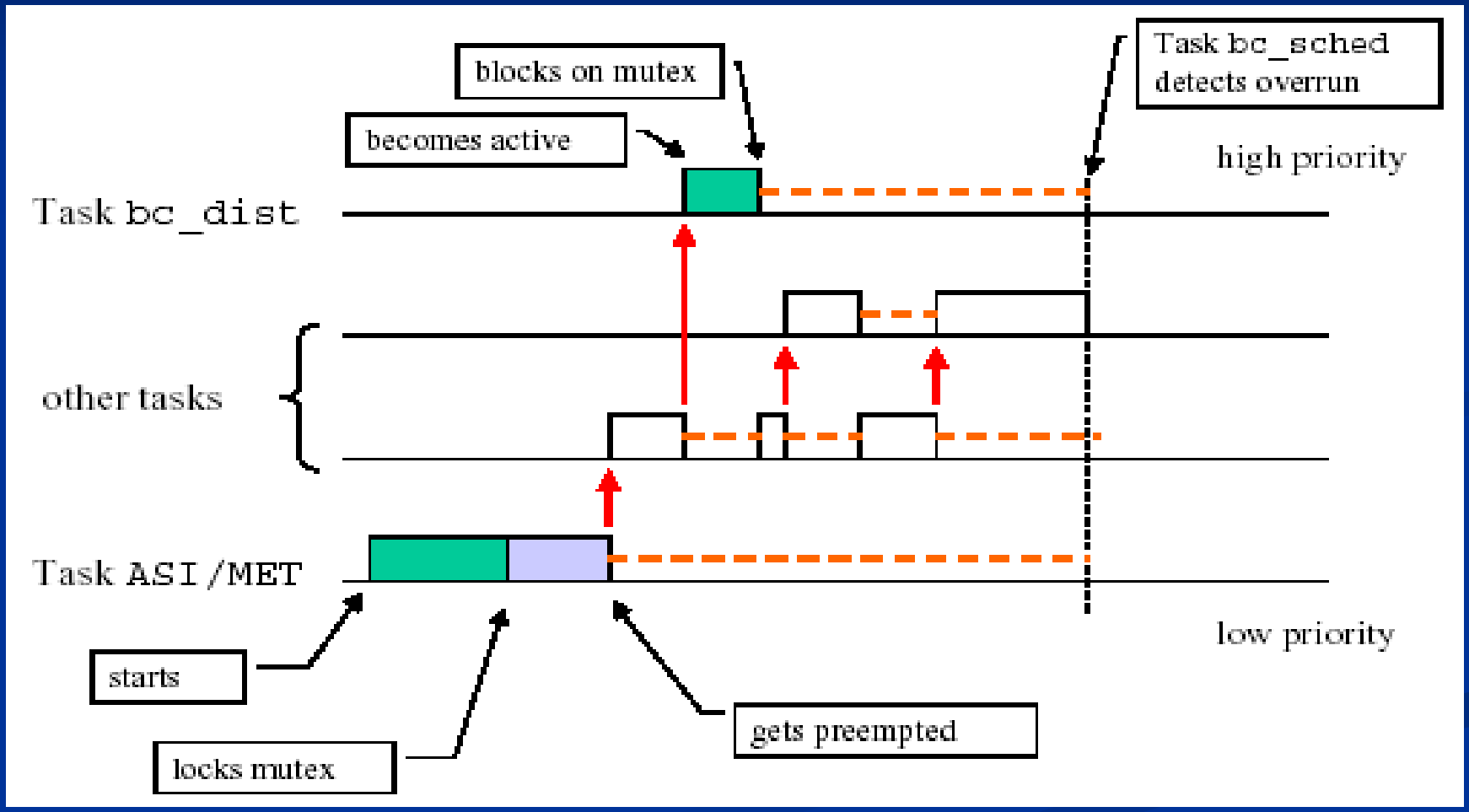
- For EDF the sufficient and effective condition is

$$U_i = C_1/T_1 + C_2/T_2 + .. + C_n/T_n <= 1$$

What can go wrong ?

- Higher priority task may have to wait !
- … for a shared resource
  - Priority inversion, deadlocks,
- Solution:
  - Non-preemptable (uninterruptible) critical sections
  - Priority inheritance
  - Priority ceiling protocol

Priority inversion on Mars Pathfinder (1997)



- http://research.microsoft.com/~mbj/Mars_Pathfinder/Mars_Pathfinder.html