Operating Systems: Wrap-up

Fall 2008

Tiina Niklander

EXAM: Wed 15.10. 16.00 A 111

- No additional material allowed.
- You can have calculator if you wish, but not necessary (if calculations asked , then the formula is fine)
- Remember to have student card (or id) and pencils with you
- Overlapping CS exams: A special start time at 14.00 has been organised for those students. – Contact Tiina if you need this.

Course feedback

- Remember to give feedback on all courses you participate!
- Improve the quality of the content, teaching, etc.

- Feedback forms in Finnish from page:
  - https://ilmo.cs.helsinki.fi/kurssit/servlet/Valinta
- Forms in English from page:
  - https://ilmo.cs.helsinki.fi/kurssit/servlet/Valinta?kieli=en

Course structure

- 1: Overview
- 2: Processes and Threads
- 3: Virtual Memory and Paging
- 4: Page Replacement
- 5: Segmentation
- 6: File Systems, part 1
- 7: File Systems, part 2
- 8: I/O Management
- 9: Multiple Processor Systems
- 10: Example: Linux, Windows, Symbian
- 11: Design Issues
- 12: Recapitulation, hints for exam

Tanenbaum: Sections 1,2,3,4 ,5,7.5,8,10,11

Learning goals

Themes:

- Operating system's general structure and main functionalities
- Processes and Threads
- Memory management and Virtual memory
- File system and I/O

Goal: Operating system's general structure and main functionalities

- Approaches (just passable, when master all of these):
  - Can describe the main services offered by the OS (operating system) and their functionalities.
  - Can outline the common OS structure and interfaces.
  - Can position the services of an OS in modern computing environment and explain their benefits
- Reaches (master the content):
  - Can explain in details the functionalities and services of an OS on one computer and as part of a distributed system.
  - Can outline and explain structure and interfaces of one specific OS (such as Linux, Windows).

Goal: Processes and Threads

- Approaches:
  - Can describe the data structures and management functions used by OS in controlling processes and threads.
  - Can describe the common scheduling mechanisms.
  - Can distinguish user mode and kernel mode and explain their main features.
  - Can describe the process protection mechanisms.
  - Can explain different ways of executing a thread.
- Reaches:
  - Can explain on algorithmic level the features used by a given OS.
  - Can compare different scheduling mechanisms.
  - Can select the most feasible thread execution mechanism for a given purpose and justify the selection.

Goal: Memory management and Virtual memory

- Approaches:
  - Can explain the key concepts (paging, page table, address translation, page fault) of a virtual memory and describe the basic features of it.
  - Can describe a multi-level page table and how it is used.
  - Can simulate, on algorithmic level, address translation of a system that used virtual memory.
- Reaches:
  - Can estimate the effect of the page size on page table size and process functionality. Can justify the selection of a certain page size.
  - Can explain in details the advantages and disadvantages of combining (multi-level) paging and segmentation.
  - Can simulate, on algorithmic level, all key mechanisms of the virtual memory, especially the page replacement and allocation.

Goal: File system and I/O

- Approaches:
  - Can outline the basic structure of a file system and explain how it works.
  - Can describe how the data is moved between the devices, the OS and the application.
- Reaches:
  - Can explain the principle of distributed file system (such as NFS).
  - Can explain and compare the file systems of different OS, at least the file systems of Linux and Windows.

Example exams: Fall 2006

- Course lasted two periods – two course exams
  - Some extra material was included
- Here only the questions relevant to this shorter are shown:
  - Processes and threads
  - Memory management
  - Virtual memory
  - File systems
  - Scheduling

Example exam: processes and threads

- Observe Solaris processes and threads (see Fig 4.15 below). Consider two applications: M and K. Each of them has 8 threads within the process. Application M is of type 3 (with 3 L-threads) and application K of type 4(with 8 L-threads). Assume that the system has 4 processors. Assume also that only one of the applications (either M or K) is executed in the system at any given time. That is the applications are not executed concurrently.
  - [3p] Explain briefly (with the help of the figure) what do the following terms mean:
    - ULT (User Level Thread)
    - LWP (Light Weight Process)
    - KLT (Kernel Level Thread)
  - [3p] How many threads can in each case (application M and K) be executing concurrently in machine language level? Justify. Give answers separately to applications M and K.
  - [3p] If an executing thread will block because of I/O, will the application (M or K) block or not? Justify. Give answers separately to applications M and K.

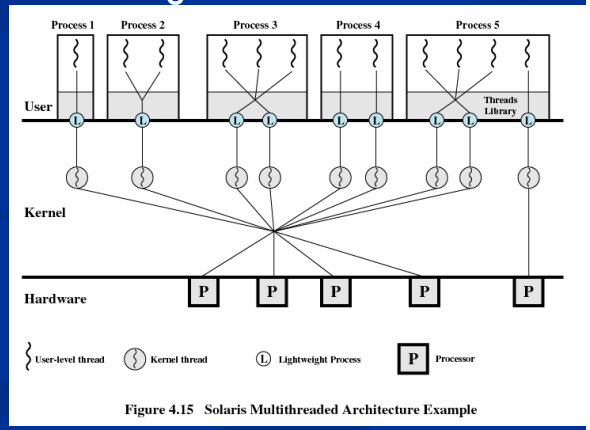


Figure 4.15 Solaris Multithreaded Architecture Example

Example exam: Memory management

- The memory configuration (of 512 megabytes of memory) at a given point in time is:

The shaded areas are allocated; the white areas are free. Additionally, the free areas are marked with letters after their sizes. The next five memory requests are for

50M, 24M, 10M, 60M, 30M

Indicate the location for each of the requests, when the memory allocation is based on

- a dynamic partitioning scheme and first-fit placement algorithm,
- a dynamic partitioning scheme and best-fit placement algorithm,

Example exam: Virtual memory

- Please explain in details, what are the components of the Memory Management Unit (MMU) and how does it do address translations, when the system is based on paging virtual memory.
- Explain how does the clock-algorithm operate. What will be the target page frame for a new page, if the clock algorithm is used in the situation given in the table below (the process has only 4 page frames). The times are clock-ticks from the beginning of the process.

Sivu# (page number)	Sivutila# (page frame)	Latausaika (Time loaded)	Viittausaika (Time referenced)	R	M
2	0	60	121	0	1
1	1	130	120	0	0
0	2	26	122	1	0
3	3	20	123	1	1

- The execution of the process continues and generates the following page reference string: 4, 0, 0, 2, 1, 5, 4, 5, 0, 3, 2

  How many page faults would occur if the working set policy were used with a window size 4 instead of the fixed allocation? Show clearly which pages form the current working set and when each page fault would occur.

Example exam: Scheduling

- Five jobs (processes) arrive to the system according the following table. Determine the turnaround time for each job and the average turnaround time for all jobs.
  - Priority. Pure priority-based (low number means high priority), one job at a time runs until it finishes.
  - SPN (Shortest Job First). Each job is completed before the next can start.
  - FCFS (First Come First Served). job is completed before the next can start.
  - Round Robin. Time quantum is 2.
- Remember to justify your answer! (Justification is more important than correct numerical result)

Process	Arrival time	Priority	Processing Time
A	0	1	9
B	1	3	15
C	2	5	6
D	3	4	3
E	4	2	12

Example exam: File systems

- *Free disk blocks*. Explain two approaches to keep track of the free disk blocks available to allocation on a disk. Give the pros and cons for each alternative.
- *ext2fs*. What is *inode* and what information does it contain?
- *ext2fs.* When the block size is 1 KB, how does the OS store a 30 MB file test.txt? How does it locate the file's allocated disk blocks?
- *NTFS.* What is Master File Table (MFT) in NTFS, where is it located, how is it used?

Operating System Overview

Selection of slides
from previous lectures

OS structure

User

Applications | Shell | System programs

System calls

Process management | Resource management

protection

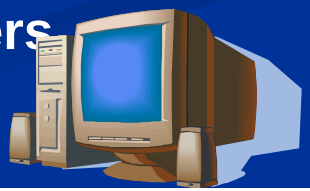
File system | Memory management

blocks

I/O management

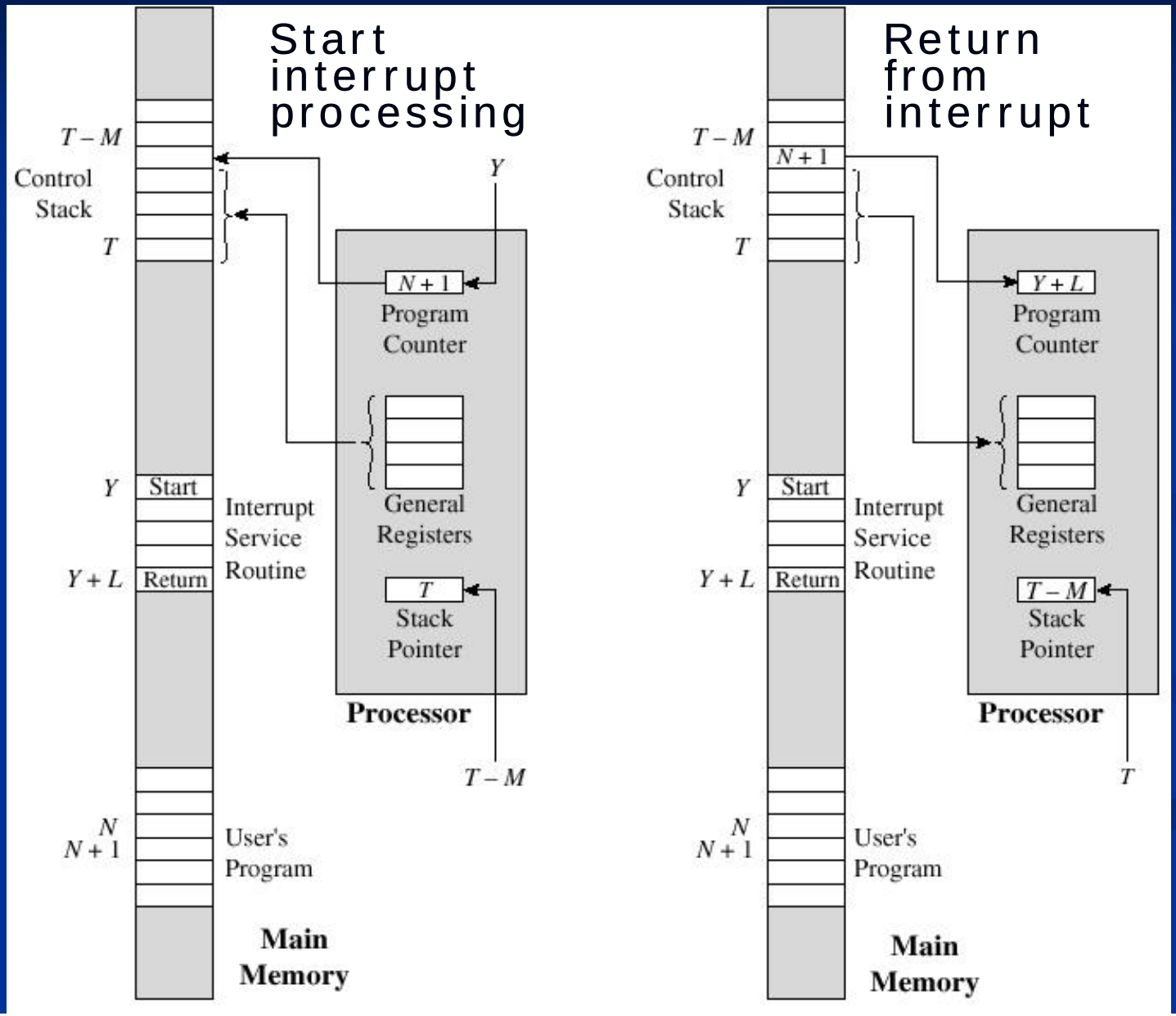
I/O module (Device controller) | Interrupt handling

Devices and device drivers



Interrupt in details

Sta Fig 1.11



Memory hierarchy

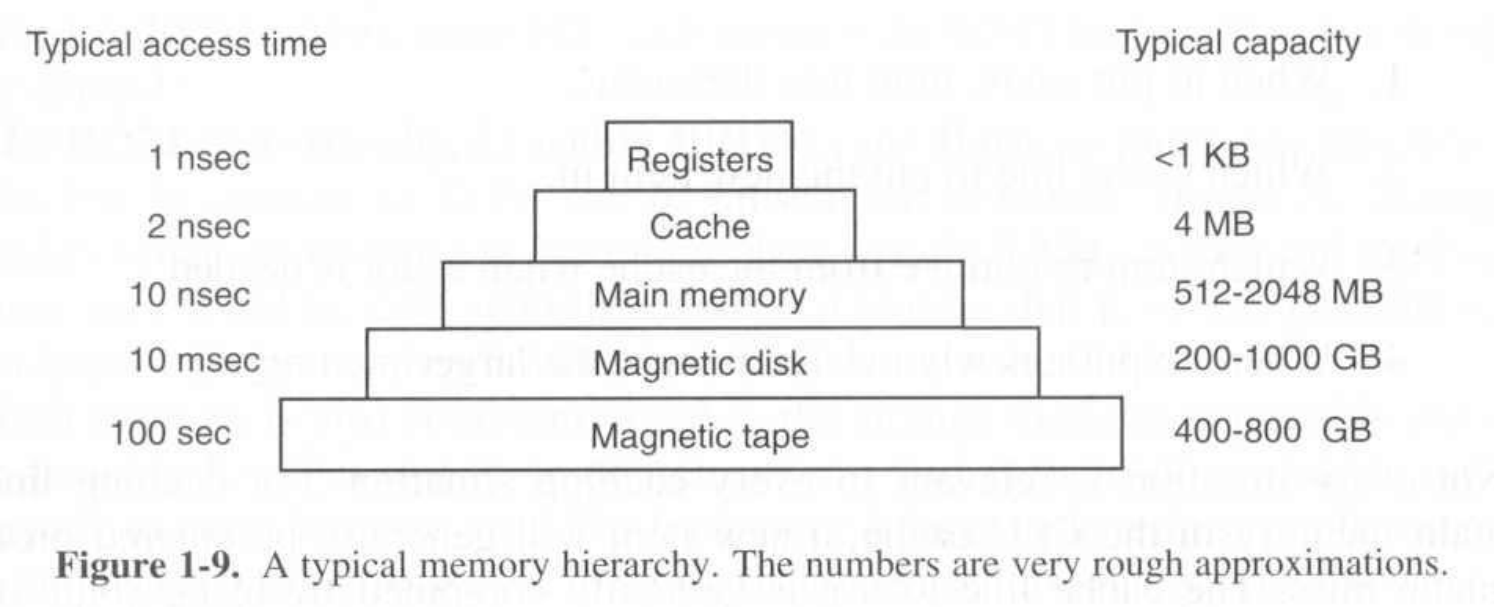


Figure 1-9. A typical memory hierarchy. The numbers are very rough approximations.

Pentium 4 cache:

8 KB data, 12 KB code/text, external 256 KB

nano = 10^{-9}, micro = 10^{-6}, milli = 10^{-3}

Locality of reference

Spatial and temporal locality:

- For example in a loop a small set of instructions is executed several times
- Certain part of code only uses a small set of variables (data)

➔ When program makes a memory reference (data or instructions), it is likely to refer again to the same location or a location near by

- This is the principle behind the usage of caches

Systems calls

- Interface between user programs and OS dealing with abstractions
- Special kind of procedure call: switch between user and kernel mode
- Trap into kernel and invoke OS

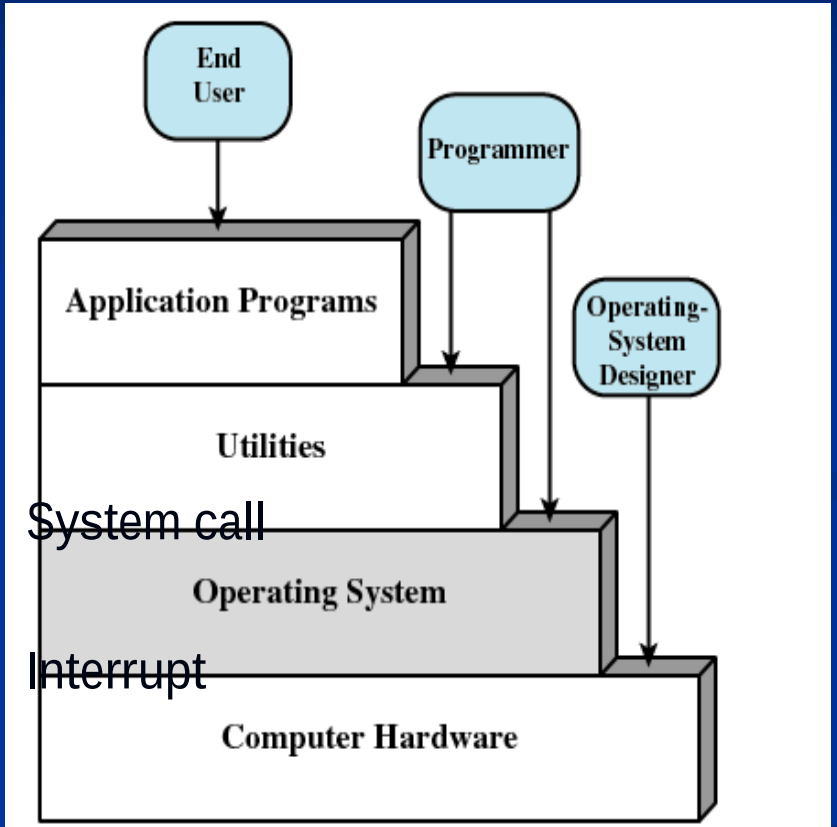
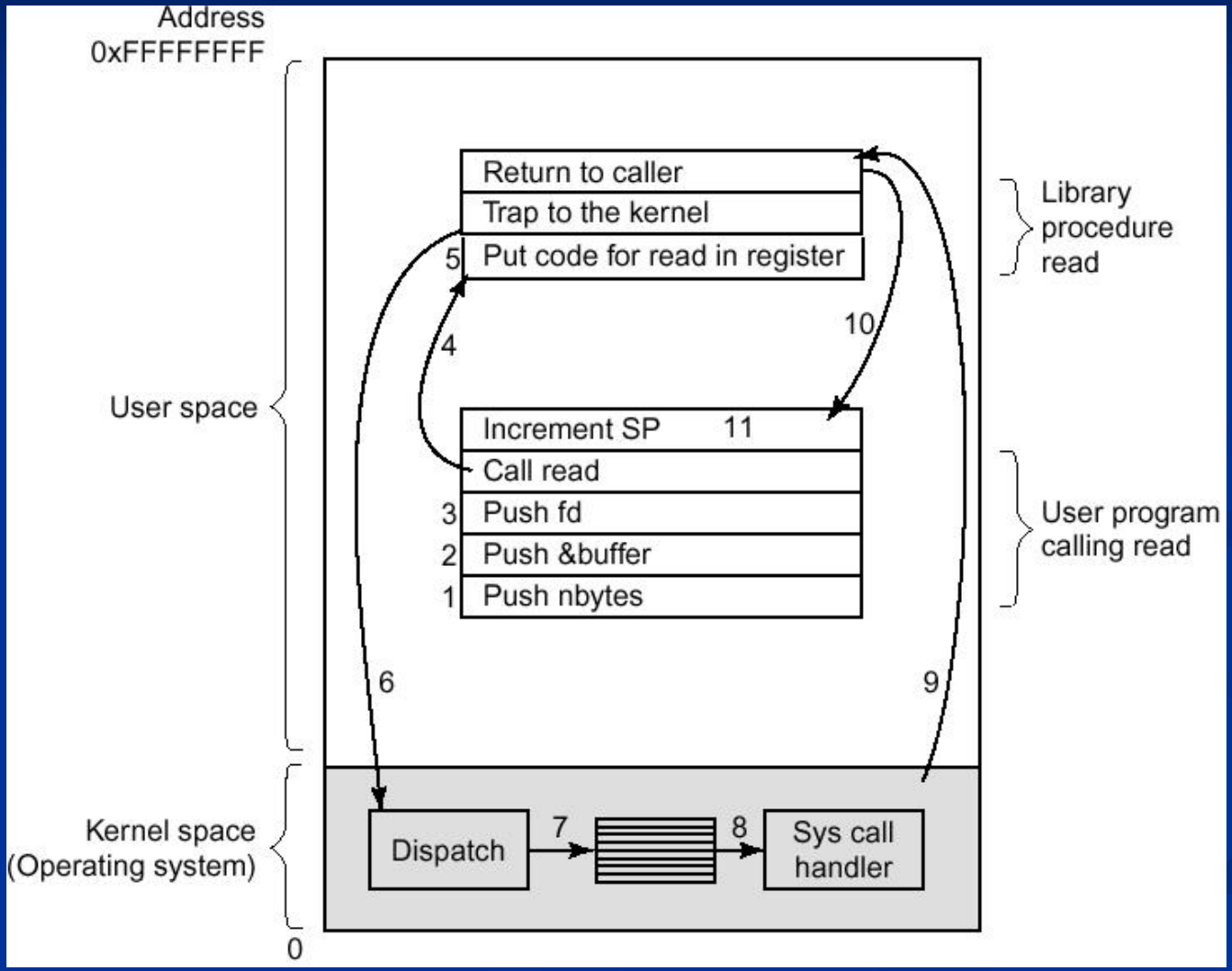


Figure 2.1 Layers and Views of a Computer System

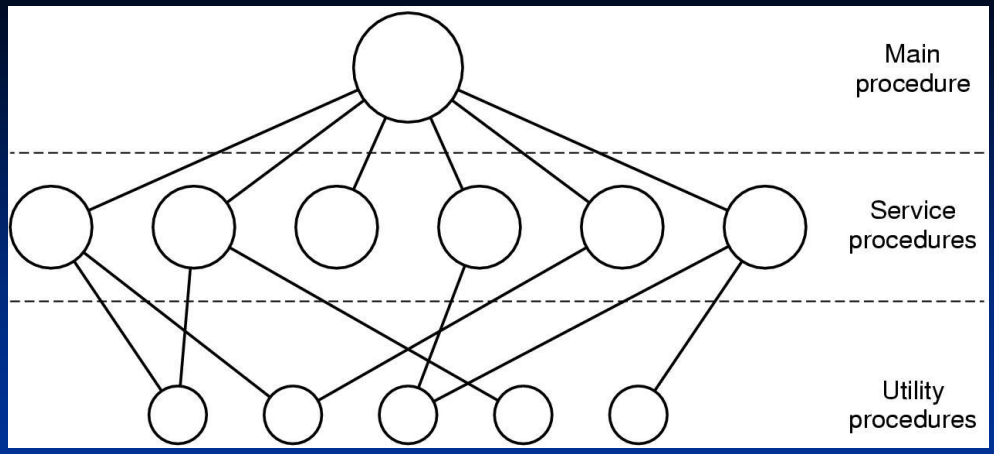
read(fd, buffer, nbytes) Tan08 1-17



Operating system structures

- Monolithic System
- Microkernel
- Virtual Machines

Monolithic Systems



- Whole OS as one program in kernel mode
  - Collection of procedures, linked together
- Basic structure of a monolithic system
  - Main program invokes requested service procedure
  - Service procedures carry out the system calls
  - Utility procedures help service procedures
- There might be loadable extensions

Micro-kernel

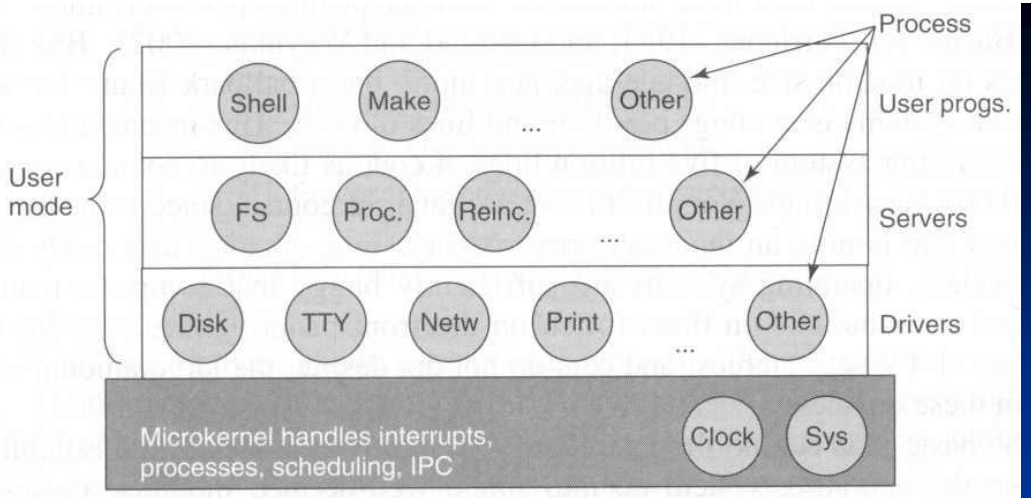


Figure 1-26. Structure of the MINIX 3 system.

- Split the OS to small, well-defined modules
- Only the microkernel module runs in kernel mode
- Other modules run in user mode
- Goal: increase availability
  - A buggy driver cannot crash the whole computer
  - MINIX has reincarnation server to automatically replace failed module
- Disadvantage:
  - A lot of mode switches within OS itself

Virtualization today

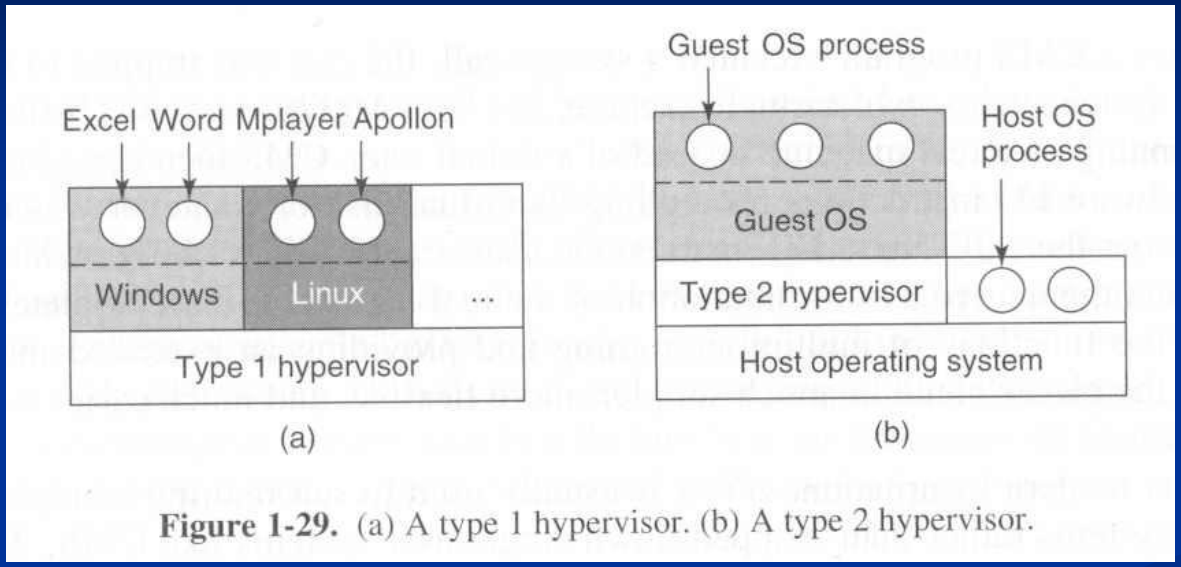


Figure 1-29. (a) A type 1 hypervisor. (b) A type 2 hypervisor.

(a) Type 1 hypervisor
on top of hardware
multiplex
all OS in parallel

(b) Type 2 hypervisor
on top of host OS
multiplex
only guest OSes

- Virtual machine monitor called hypervisor

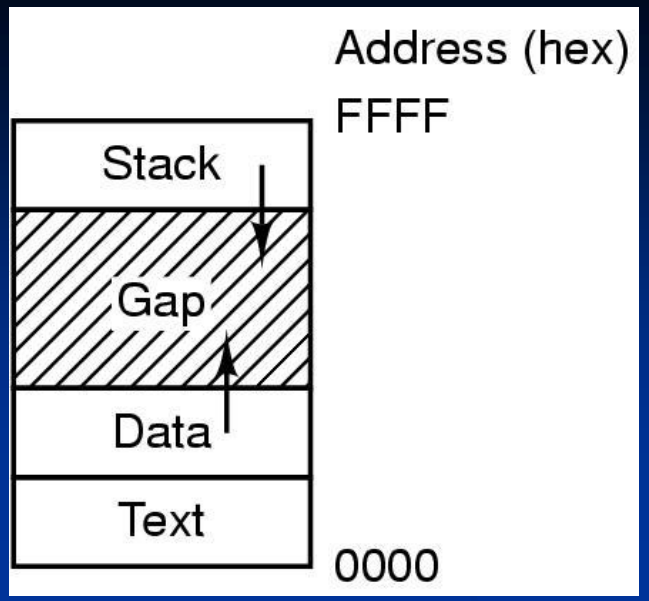
C and Metric Units

- Must be able to use:
  - Reading C-code examples
  - Unit conversions

Exp.	Explicit	Prefix	Exp.	Explicit	Prefix
10^{-3}	0.001	milli	10^{3}	1,000	Kilo
10^{-6}	0.000001	micro	10^{6}	1,000,000	Mega
10^{-9}	0.000000001	nano	10^{9}	1,000,000,000	Giga
10^{-12}	0.000000000001	pico	10^{12}	1,000,000,000,000	Tera
10^{-15}	0.000000000000001	femto	10^{15}	1,000,000,000,000,000	Peta
10^{-18}	0.000000000000000001	atto	10^{18}	1,000,000,000,000,000,000	Exa
10^{-21}	0.000000000000000000001	zepto	10^{21}	1,000,000,000,000,000,000,000	Zetta
10^{-24}	0.000000000000000000000001	yocto	10^{24}	1,000,000,000,000,000,000,000,000	Yotta

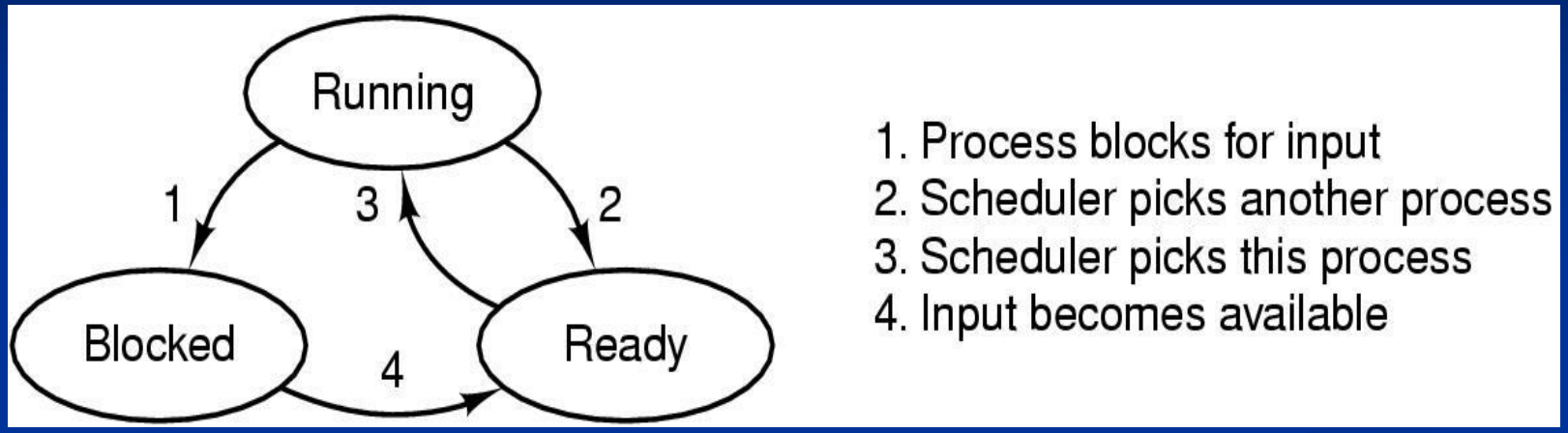
Processes and Threads

Process



- is an activity that has a program, input, output and a state.
- Some terms:
- Text/code = executable instructions
- Data = variables
- Stack = workarea
  - Parameter passing to subroutines/system calls
- Process Control Block, PCB entry in Process Table
  = management information

Process States (1)



- Possible process states
  - running
  - blocked
  - ready
- Transitions between states shown

Process Control Block

Process management	**Memory management**	**File management**
Registers	Pointer to text segment	Root directory
Program counter	Pointer to data segment	Working directory
Program status word	Pointer to stack segment	File descriptors
Stack pointer		User ID
Process state		Group ID
Priority		
Scheduling parameters		
Process ID		
Parent process		
Process group		
Signals		
Time when process started		
CPU time used		
Children's CPU time		
Time of next alarm		

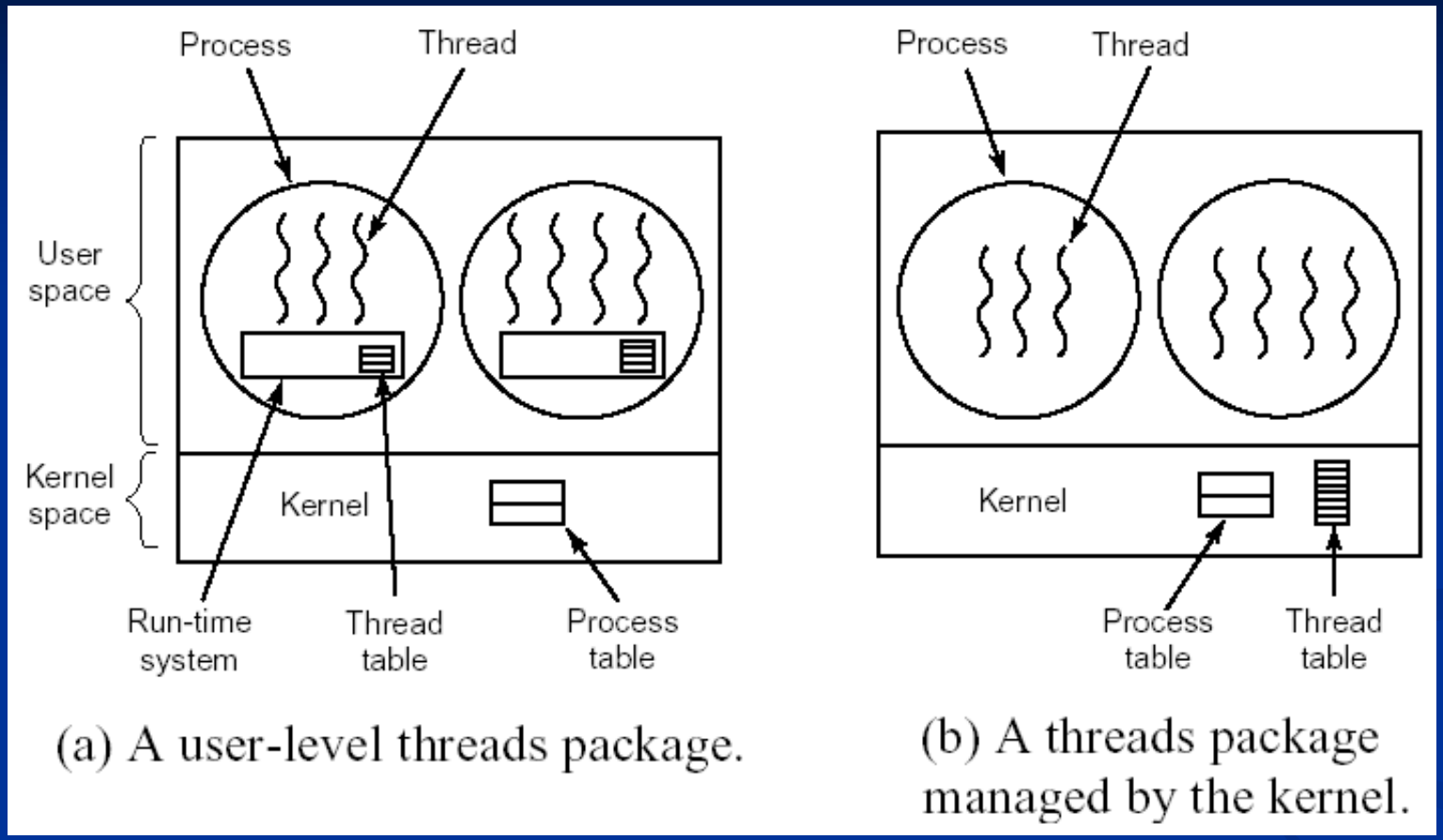
Fields of a process table entry

Thread Model: Process vs thread

Per process items	Per thread items
Address space	Program counter
Global variables	Registers
Open files	Stack
Child processes	State
Pending alarms	
Signals and signal handlers	
Accounting information	

- Items shared by all threads in a process
- Items private to each thread in thread
- Each thread has its own stack!

User-level vs kernel-level threads



Tan08
Fig 2-16

- Kernel (or OS) is not aware of threads, schedules processes
- User process must dispatch threads itself

- All thread control and dispatching done by the kernel
- No control on the user level

Solaris

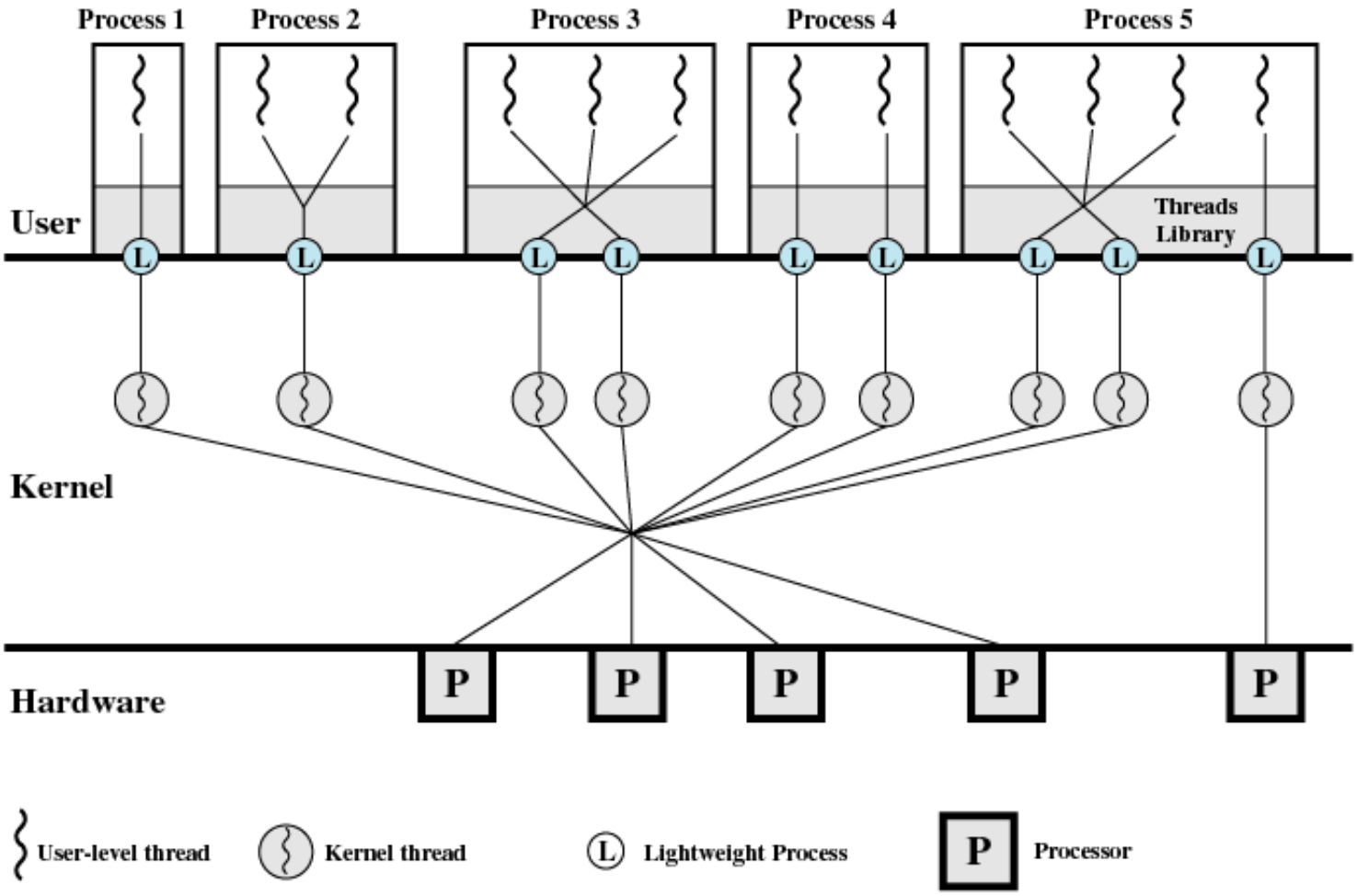


Figure 4.15 Solaris Multithreaded Architecture Example

Scheduling

Introduction to Scheduling (2)

All systems

- Fairness - giving each process a fair share of the CPU
- Policy enforcement - seeing that stated policy is carried out
- Balance - keeping all parts of the system busy

Batch systems

- Throughput - maximize jobs per hour
- Turnaround time - minimize time between submission and termination
- CPU utilization - keep the CPU busy all the time

Interactive systems

- Response time - respond to requests quickly
- Proportionality - meet users' expectations

Real-time systems

- Meeting deadlines - avoid losing data
- Predictability - avoid quality degradation in multimedia systems

Scheduling Algorithm Goals

CPU Scheduling: Algorithms examples

- First-Come-First-Served FCFS
- Round Robin RR
- Shortest Process Next SPN
- Shortest Remaining Time SRT
- Multilevel Feedback feedback

Yhteenveto

Tbl 9.3 [Stal05]

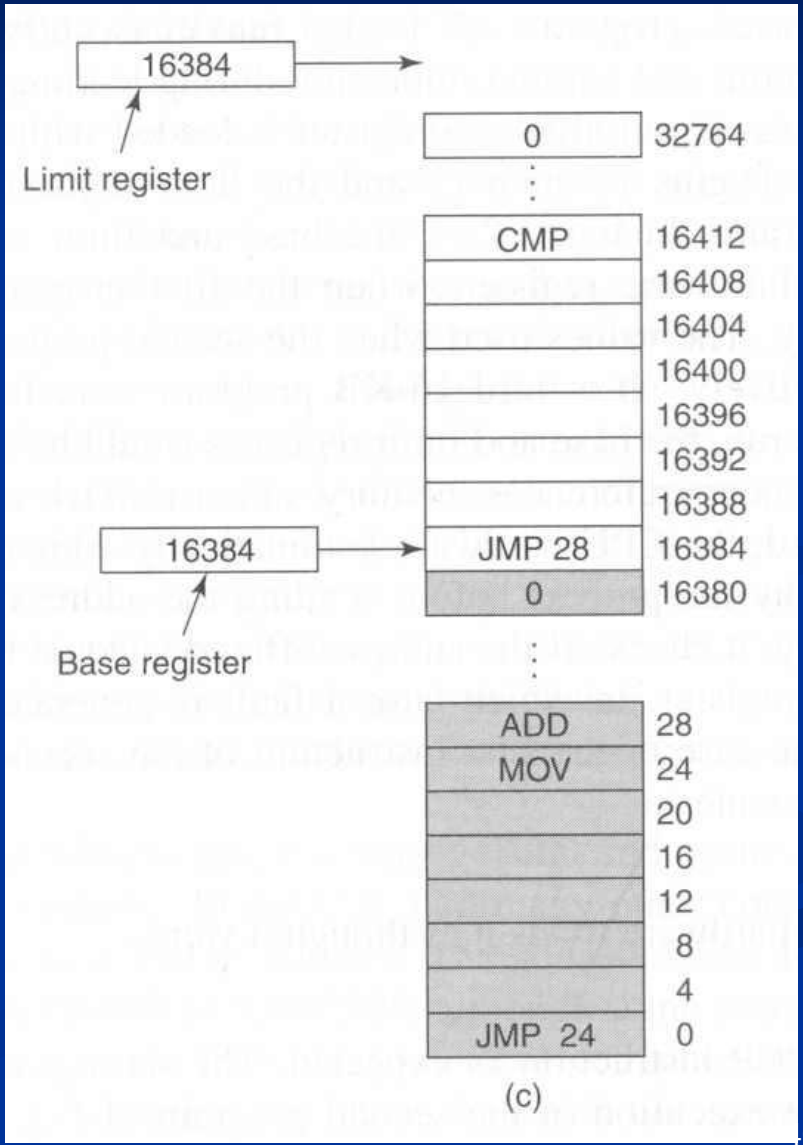
	Selection Function	Decision Mode	Throughput	Response Time	Overhead	Effect on Processes	Starvation
FCFS	max[w]	Nonpreemptive	Not emphasized	May be high, especially if there is a large variance in process execution times	Minimum	Penalizes short processes; penalizes I/O bound processes	No
Round Robin	constant	Preemptive (at time quantum)	May be low if quantum is too small	Provides good response time for short processes	Minimum	Fair treatment	No
SPN	min[s]	Nonpreemptive	High	Provides good response time for short processes	Can be high	Penalizes long processes	Possible
SRT	min[$s - e$]	Preemptive (at arrival)	High	Provides good response time	Can be high	Penalizes long processes	Possible
HRRN	$\max\left(\frac{w+s}{s}\right)$	Nonpreemptive	High	Provides good response time	Can be high	Good balance	No
Feedback	(see text)	Preemptive (at time quantum)	Not emphasized	Not emphasized	Can be high	May favor I/O bound processes	Possible

w = time spent ~~in system so far~~, waiting ~~and executing~~
e = time spent in execution so far
s = total service time required by the process, including e

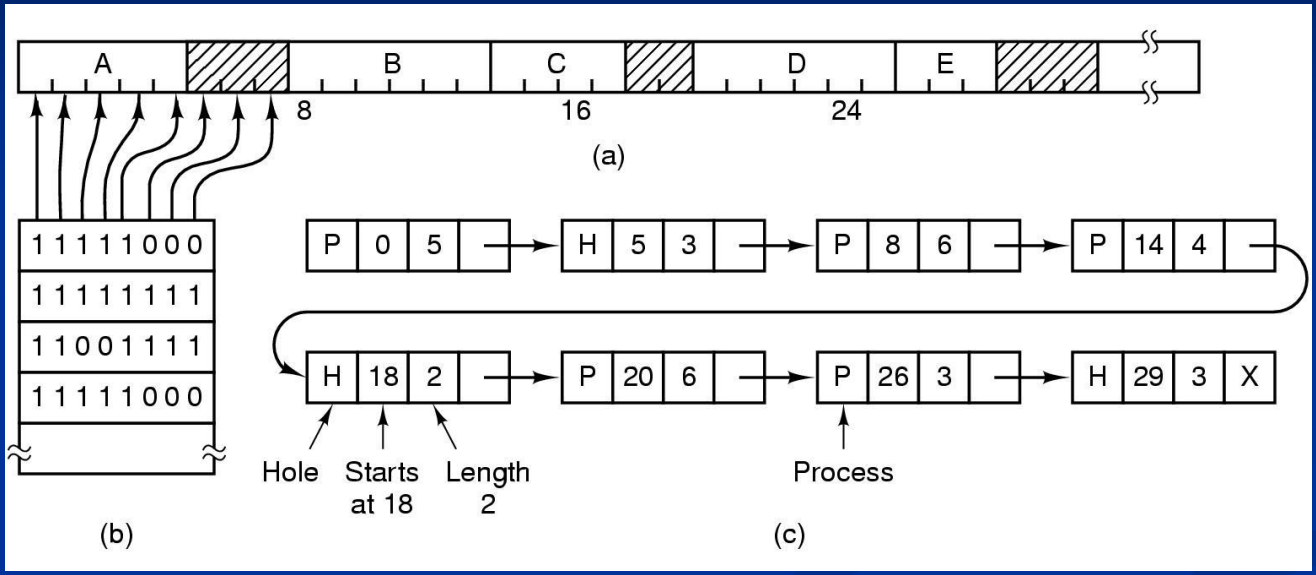
Memory management

Relocation and Protection

- Cannot be sure where program will be loaded in memory
  - address locations of variables, code routines cannot be absolute
  - must keep a program out of other processes' partitions
- Use base and limit values
  - address locations added to base value to map to physical addr
  - address locations larger than limit value is an error
- Address translation by MMU



Memory Management: bookkeeping allocations and free areas



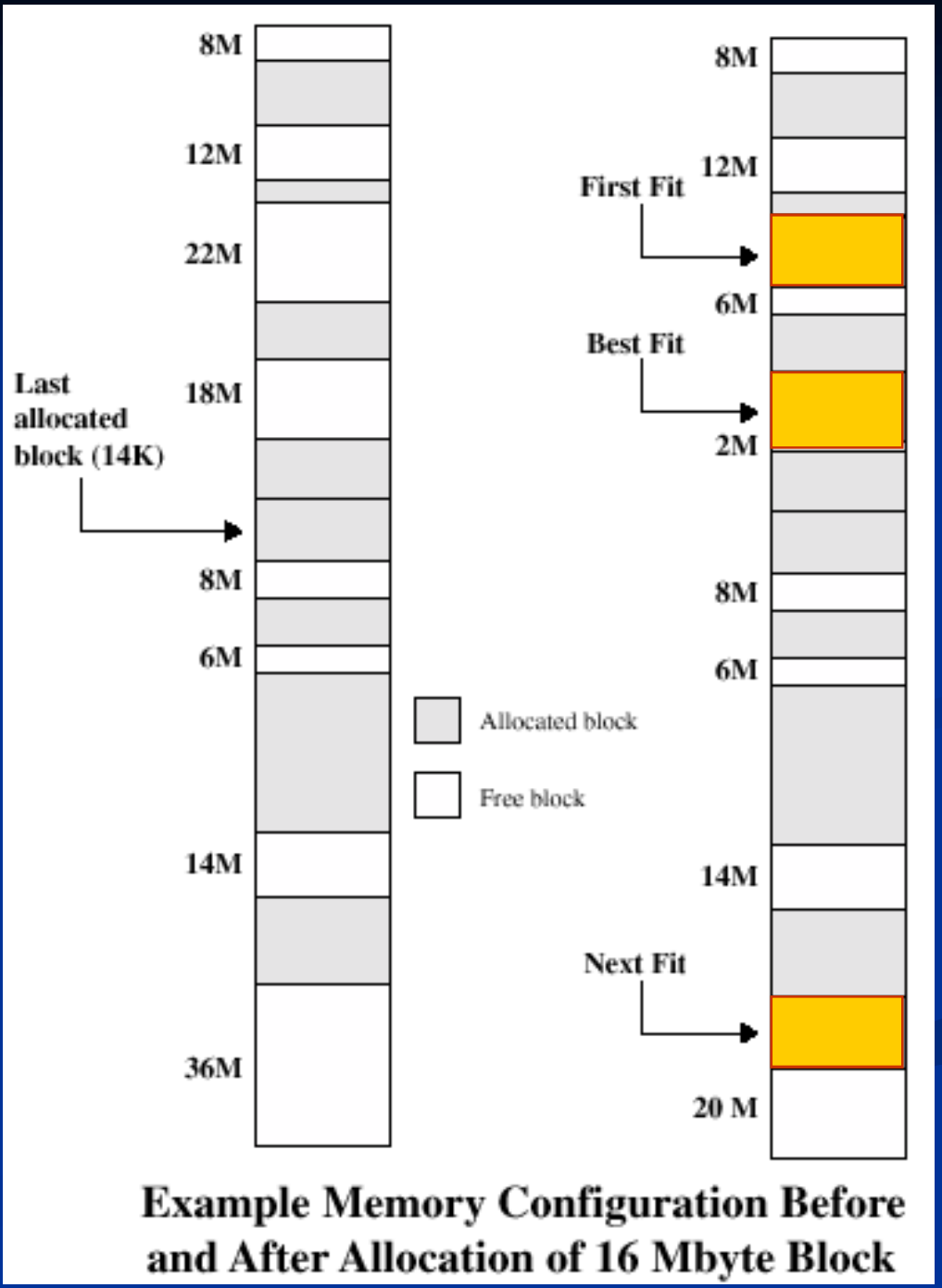
- Part of memory with 5 processes, 3 holes
  - tick marks show allocation units
  - shaded regions are free
- (b) Corresponding bit map
- (c) Same information as a list

Allocation

Where to place the new process?

- Goal: avoid external fragmentation and compaction
- Some alternatives:
- Best-fit
- First-fit
- Next-fit
- Worst-fit
- Quick-fit

Sta Fig 7.5



Example Memory Configuration Before and After Allocation of 16 Mbyte Block

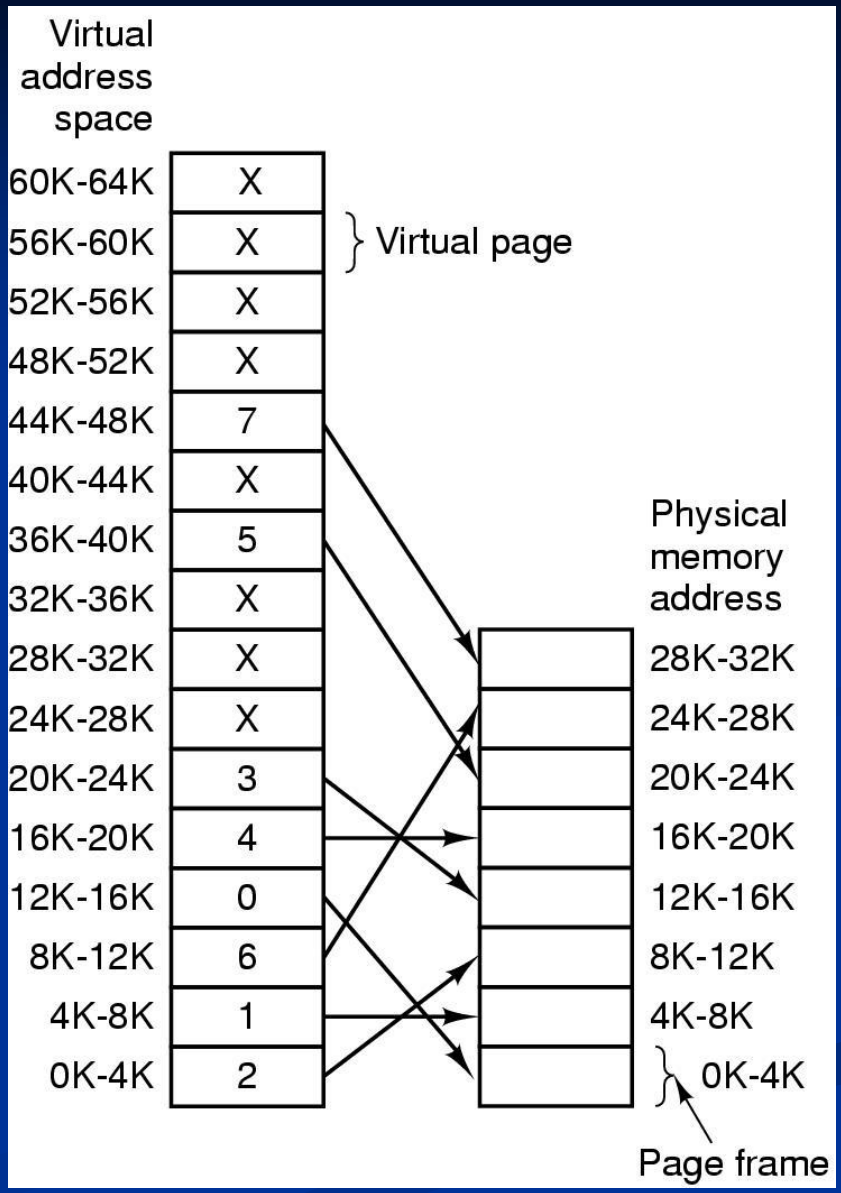
Virtual memory (using paging)

Operating System Involvement with Paging

- Process creation
  - determine program size
  - create page table
- Process execution
  - MMU reset for new process
  - TLB flushed
- Page fault time
  - determine virtual address causing fault
  - swap target page out, needed page in
- Process termination time
  - release page table, pages

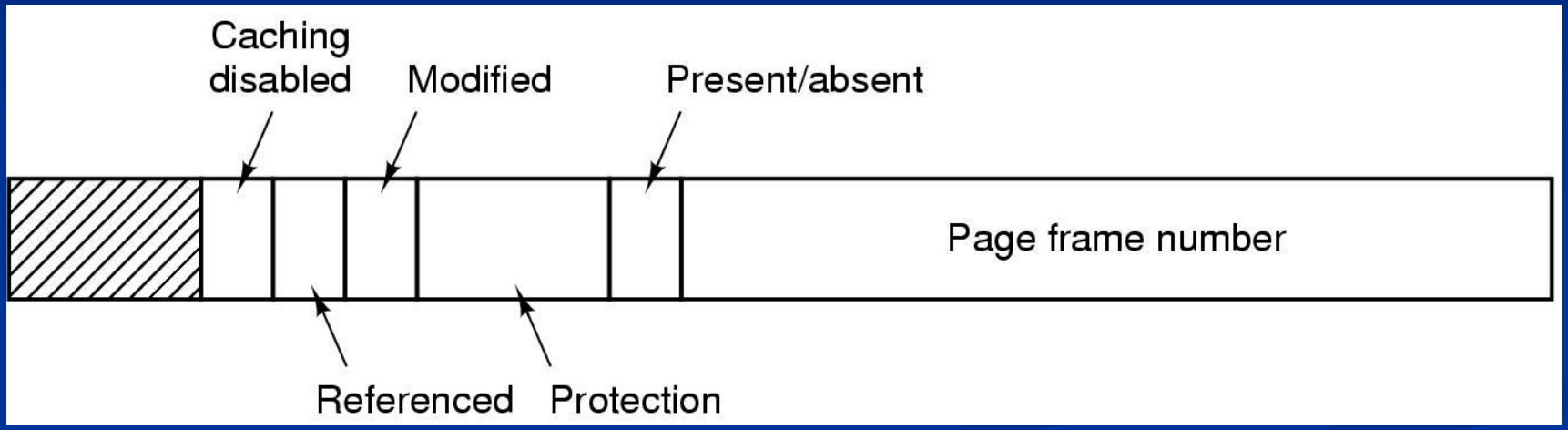
Paging

- Each process has own page table
  - Contains the locations (frame numbers) of allocated frames
  - Page table location stored in PCB, copied to PTR for execution
- OS maintains a table (or list) of page frames, to know which are unallocated



Page table

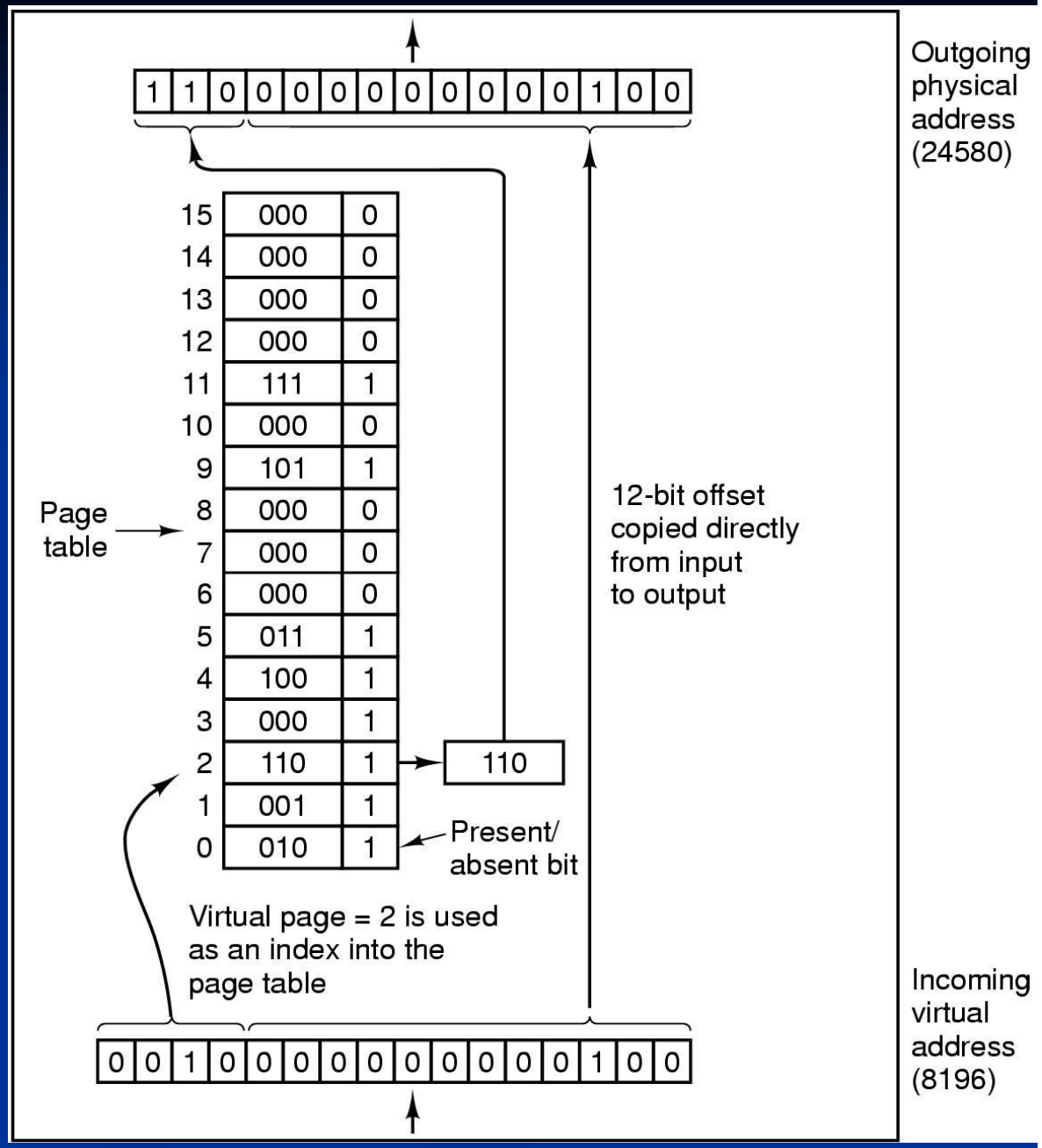
Typical Page Table Entry, Fig. 3.11



- Each process has its own page table
- Each entry has a present bit, since not all pages need to be in the memory all the time -> page faults
- Remember the locality principle
- Logical address space can be much larger than the physical

Page Tables

Internal operation of MMU with 16 4 KB pages



TLB – Translation Lookaside Buffer

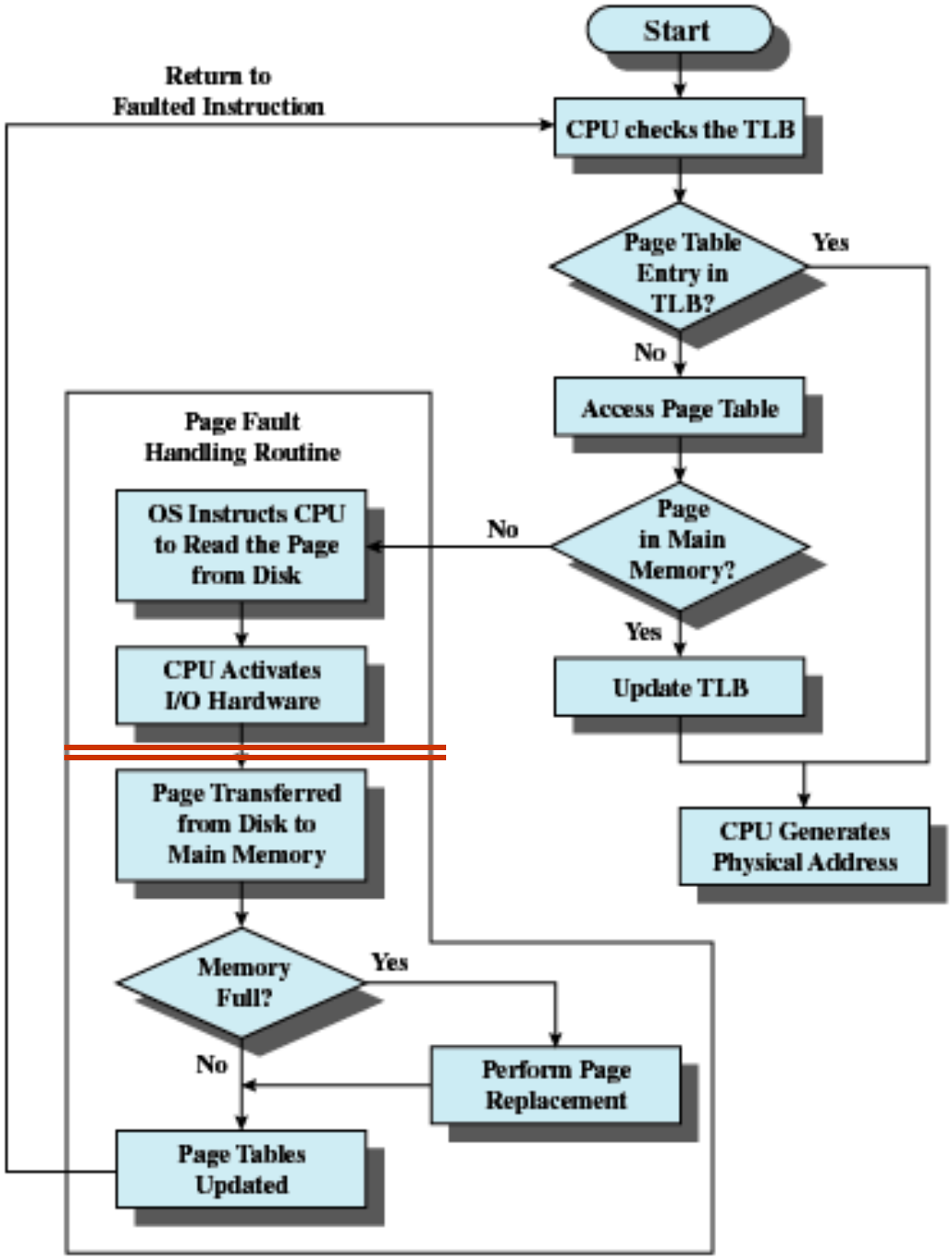
Valid	Virtual page	Modified	Protection	Page frame
1	140	1	RW	31
1	20	0	R X	38
1	130	1	RW	29
1	129	1	RW	62
1	19	0	R X	50
1	21	0	R X	45
1	860	1	RW	14
1	861	1	RW	75

Goal is to speed up paging

TLB is a cache in MMU for page table entries

Operation of Paging and TLB

Sta Fig 8.8.



Multilevel and Inverted Page Tables

Two-level hierarchical page table

- Top most level in one page and always in the memory

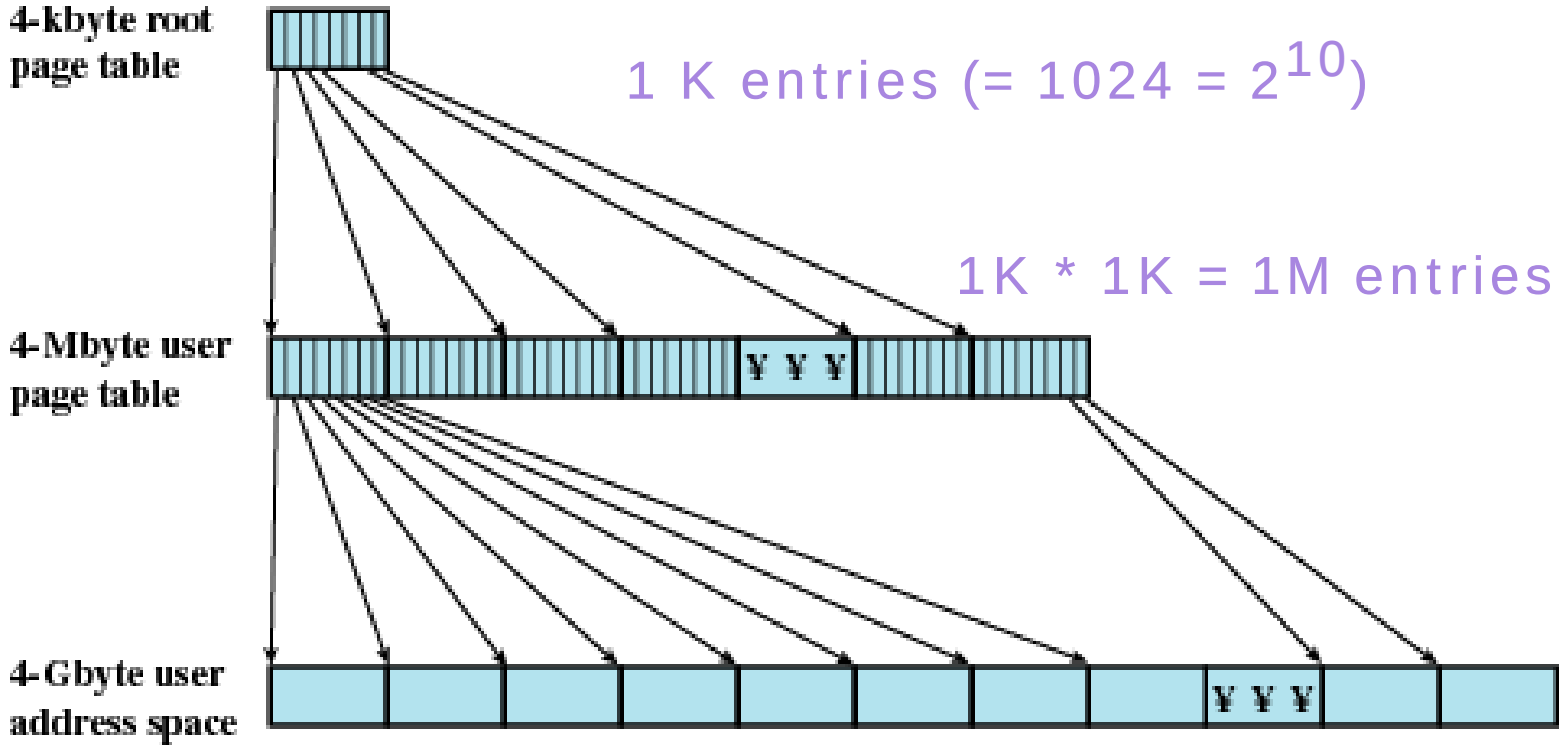
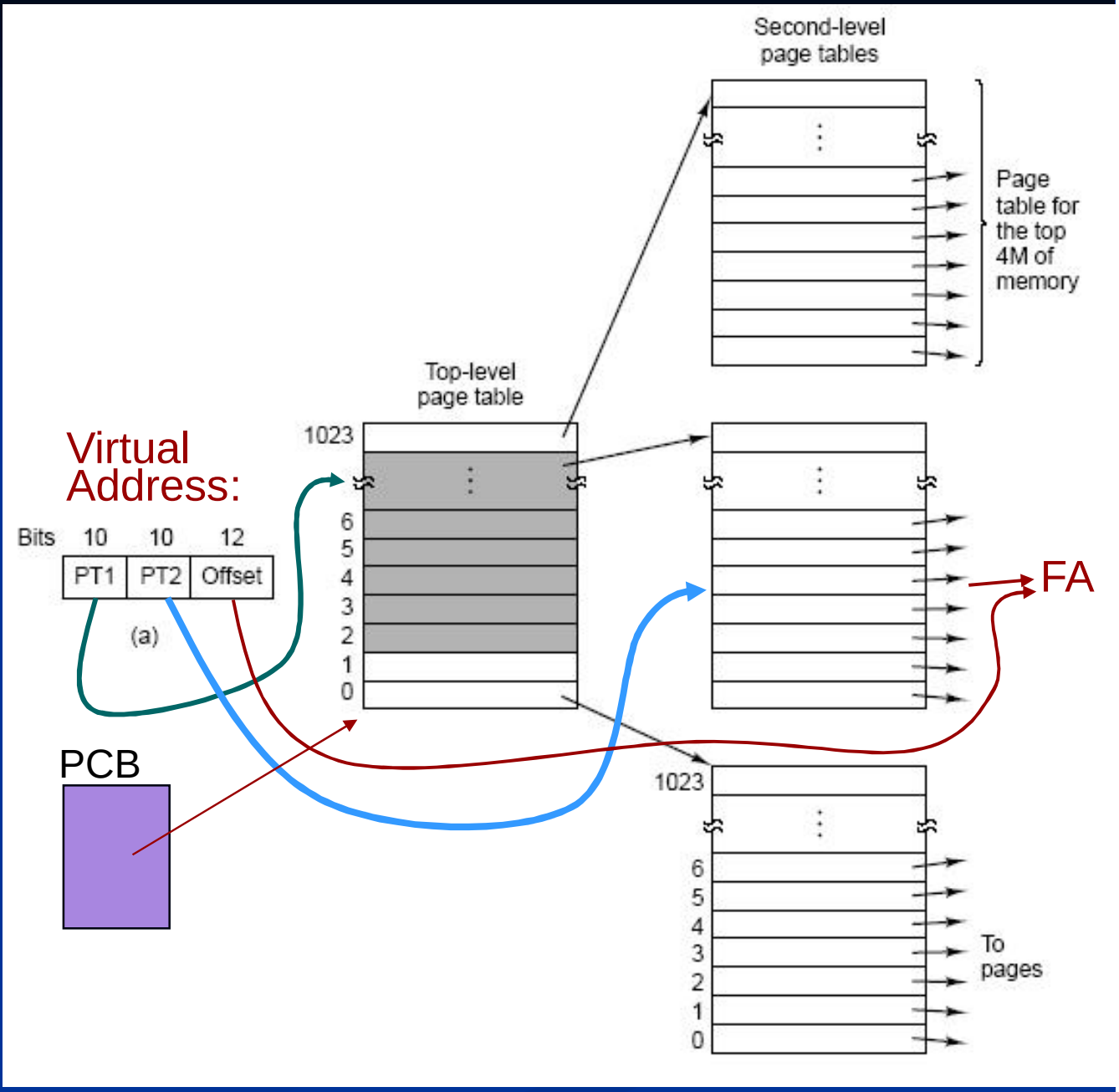


Figure 8.4 A Two-Level Hierarchical Page Table

Address translation with two levels



(Fig 4-12 [Tane01])

Inverted page table

Sta Fig 8.6

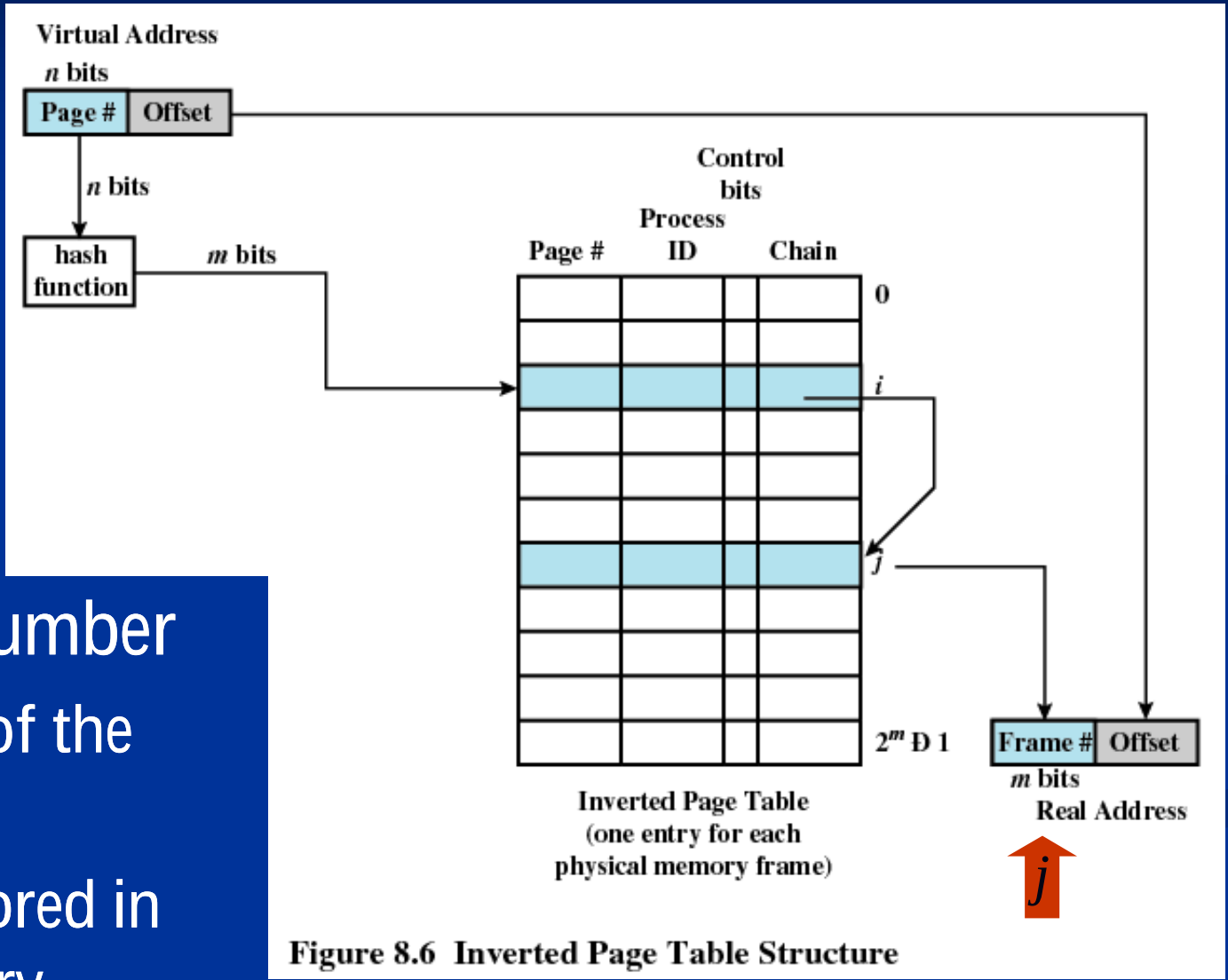


Figure 8.6 Inverted Page Table Structure

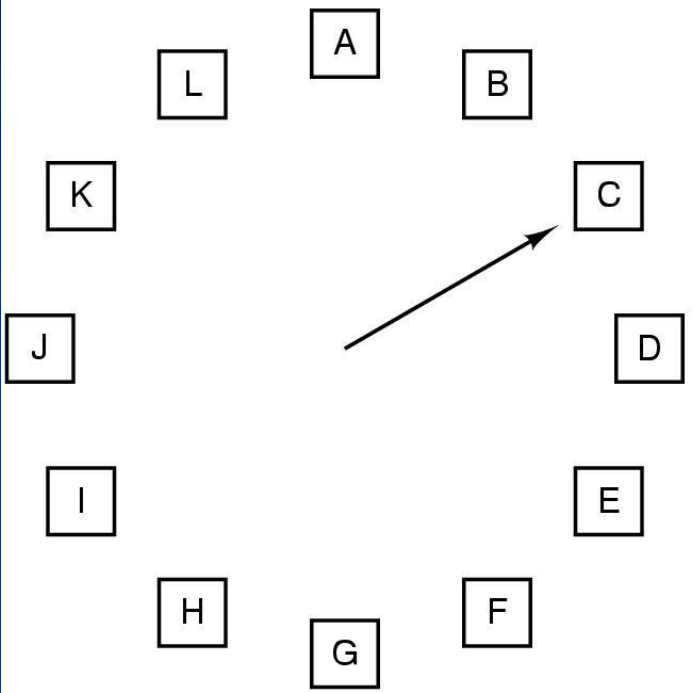
- Frame number
  - Index of the table
  - Not stored in the entry

Page Replacement

Page Fault Handling

1. Hardware traps to kernel
2. General registers saved
3. OS determines which virtual page needed
4. OS checks validity of address, seeks page frame
5. If selected frame is dirty, write it to disk
6. OS schedules disk operation to bring new page in from disk
7. Page tables updated
8. Faulting instruction backed up to when it began
9. Faulting process scheduled
10. Registers restored
11. Program continues

The Clock Page Replacement Alg.



When a page fault occurs, the page the hand is pointing to is inspected. The action taken depends on the R bit:

R = 0: Evict the page

R = 1: Clear R and advance hand

- Go trough all pages in circular fashion
- Try to locate unused page with the NRU classification, used page gets second chance

Working Set

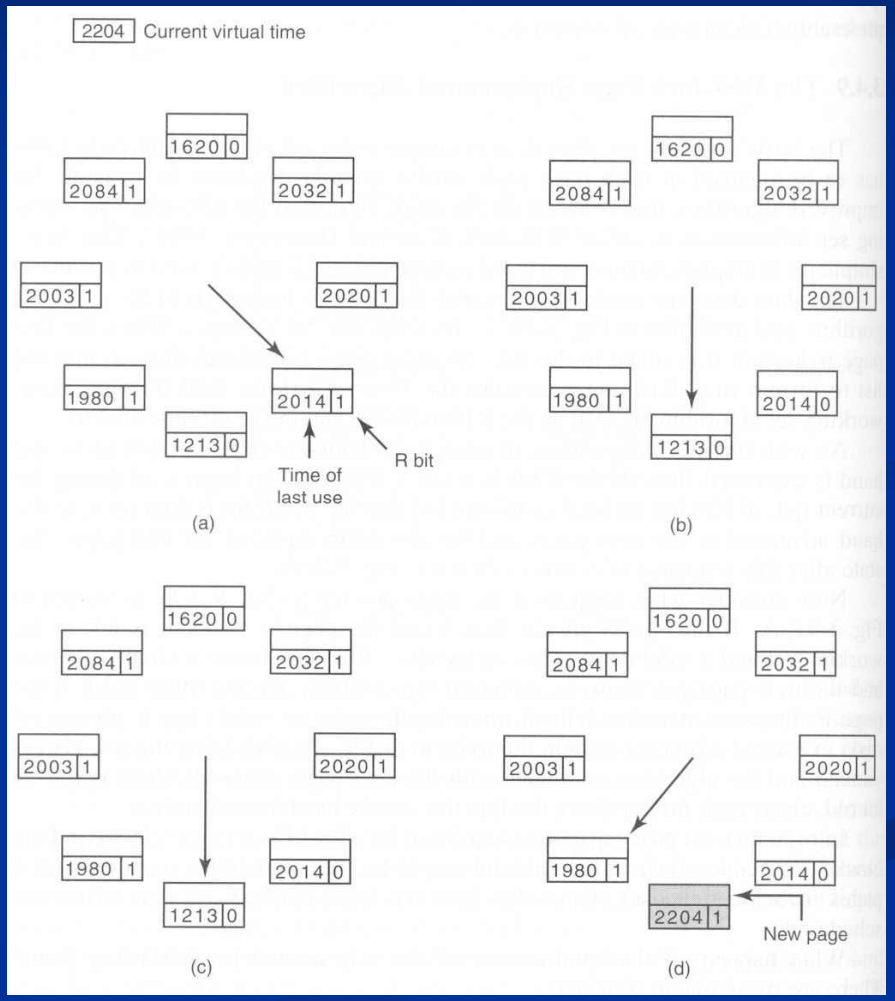
- Pages used by the most recent k memory references (window size)

$$W(t, \Delta)$$

- Common approximation
  - Pages used during the recent t msec of current virtual time, time that the process has been actually executing
- Pages not in the working set, can be evicted
- For page replacement:
  - Hardware set the R and M bits
  - Each page entry has ’time of last use’

The WSClock Page Replacement Algorithm

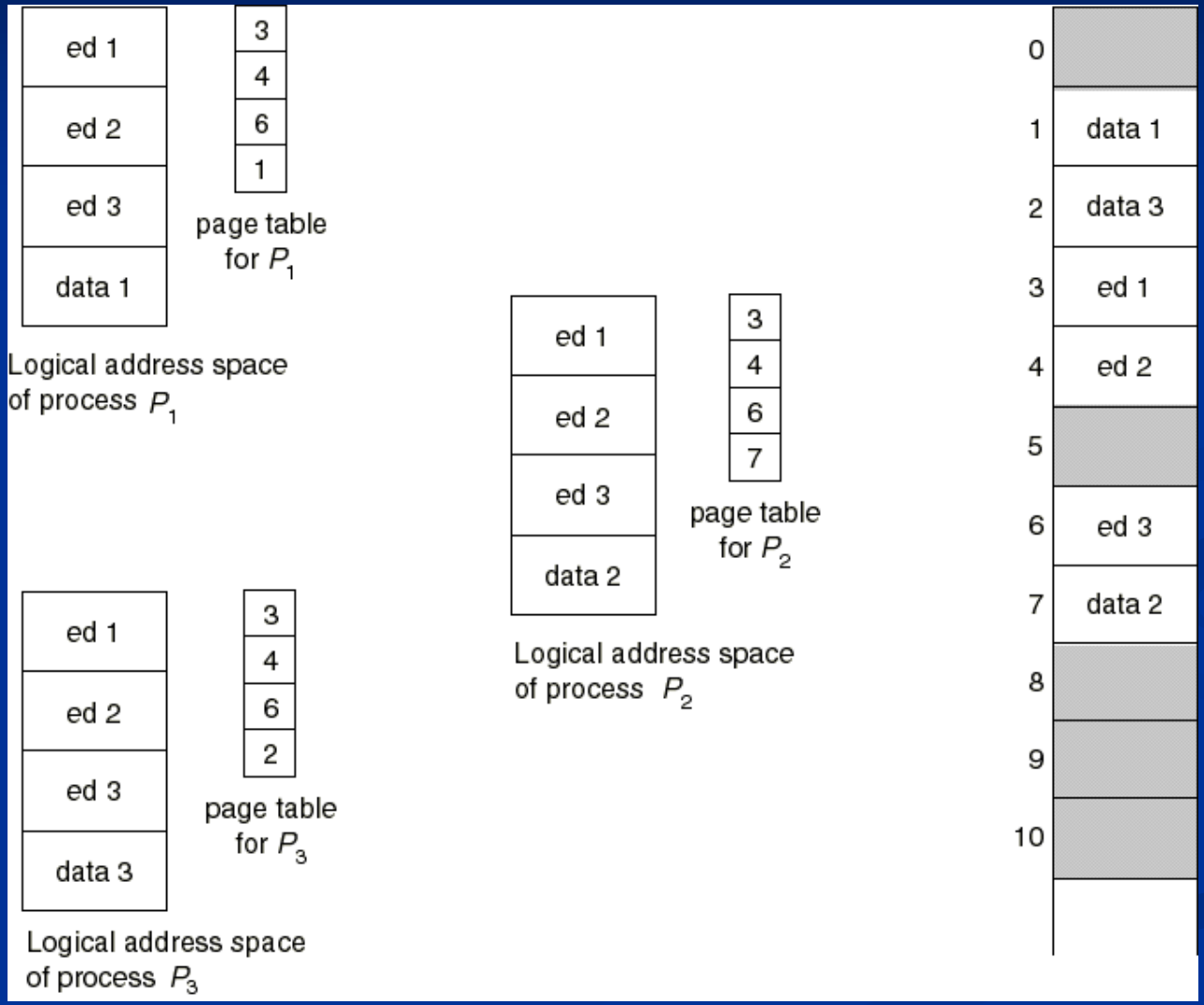
- As clock, but only residence set
- Evict unmodified page not in working set and
- Schedule for disk write modified page not in WS
- If no candidate page found during one round, wait for writes or evict any clean (=unmodified) WS page



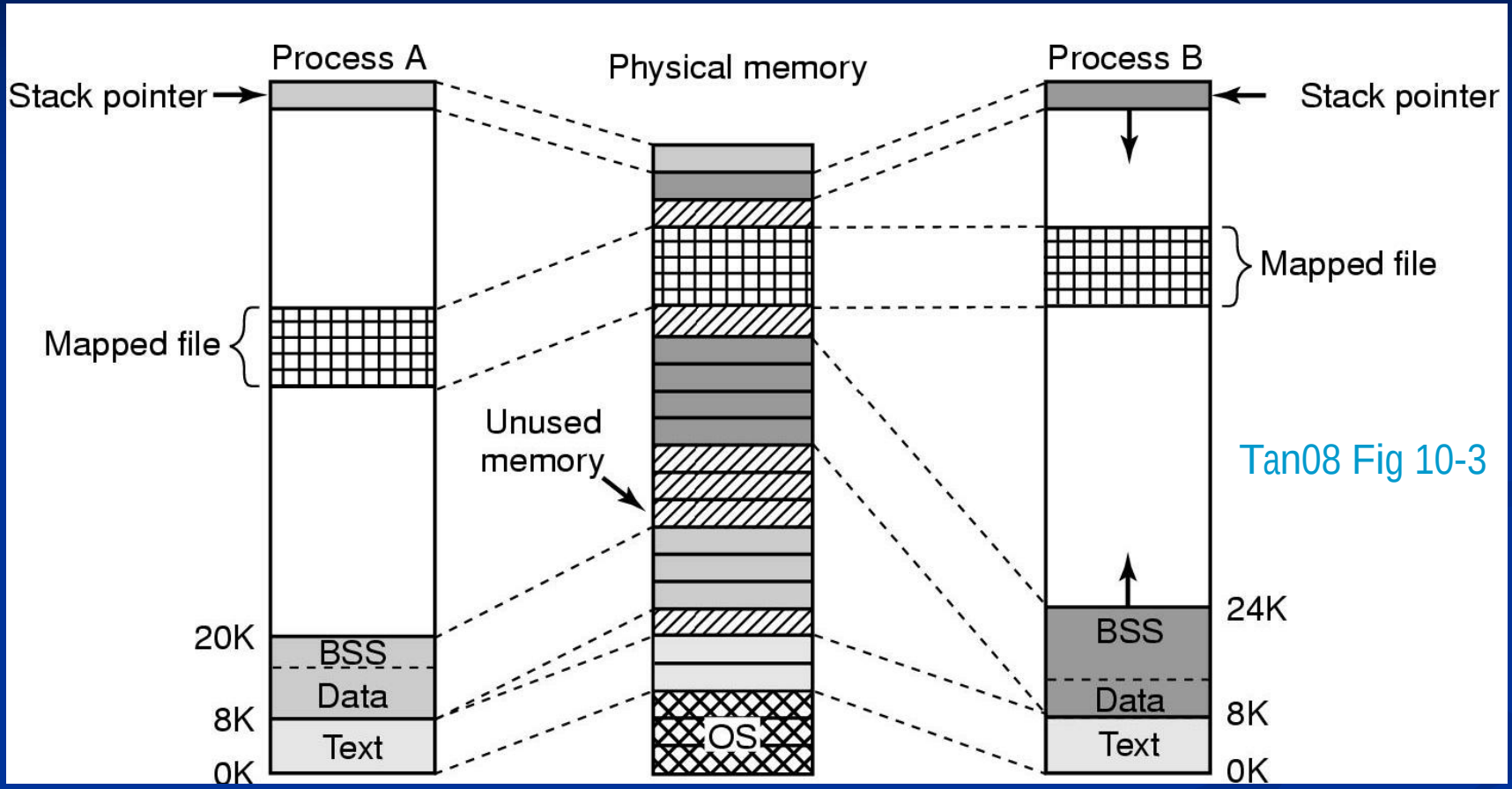
Review of Page Replacement Algorithms

Algorithm	Comment
Optimal	Not implementable, but useful as a benchmark
NRU (Not Recently Used)	Very crude
FIFO (First-In, First-Out)	Might throw out important pages
Second chance	Big improvement over FIFO
Clock	Realistic
LRU (Least Recently Used)	Excellent, but difficult to implement exactly
NFU (Not Frequently Used)	Fairly crude approximation to LRU
Aging	Efficient algorithm that approximates LRU well
Working set	Somewhat expensive to implement
WSClock	Good efficient algorithm

Sharing of pages: example editor

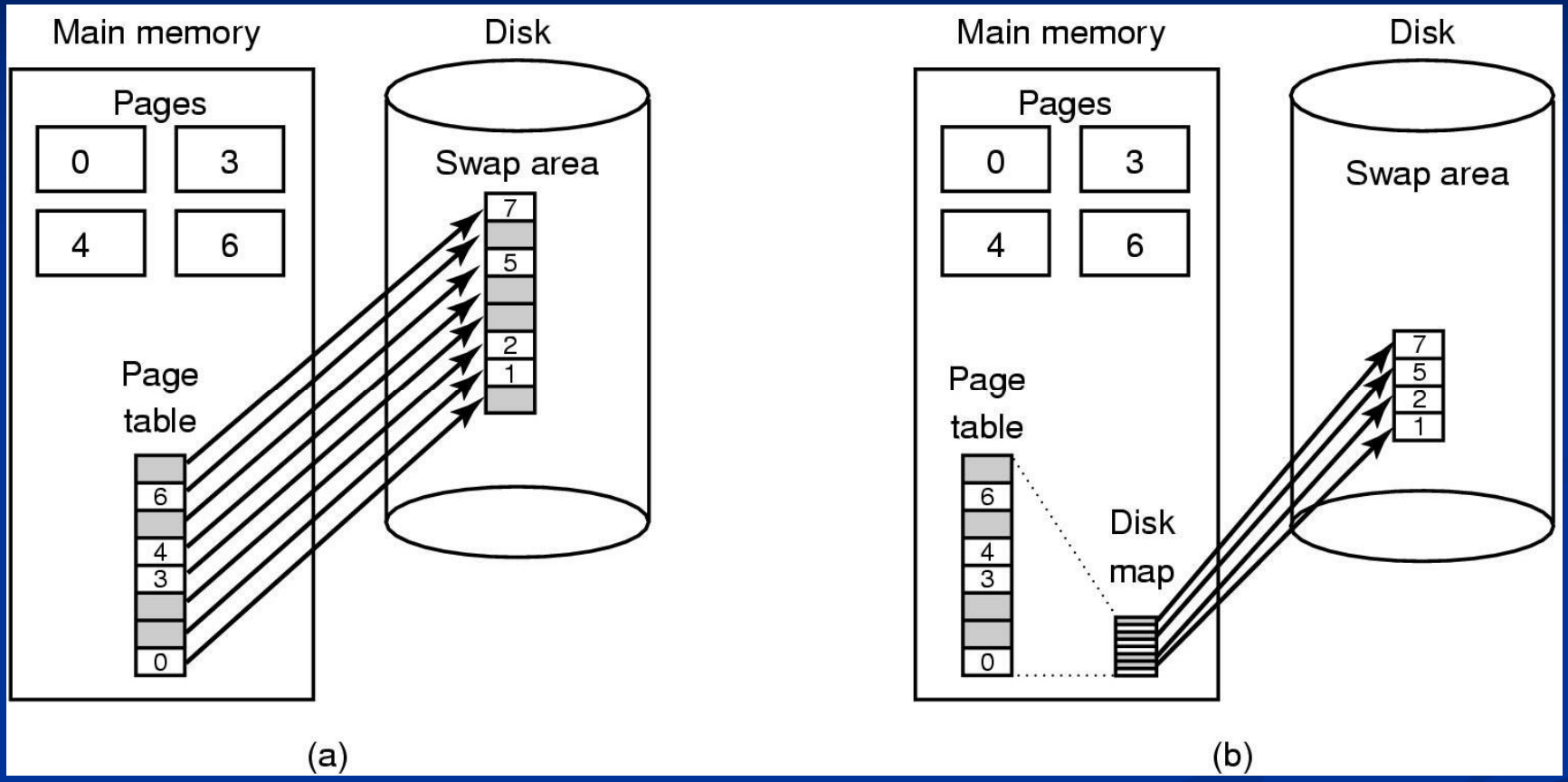


Sharing a memory-mapped file



- Mapped file is an alternative of I/O, file is accessed as a big character array in memory
- Can be used as shared memory between processes

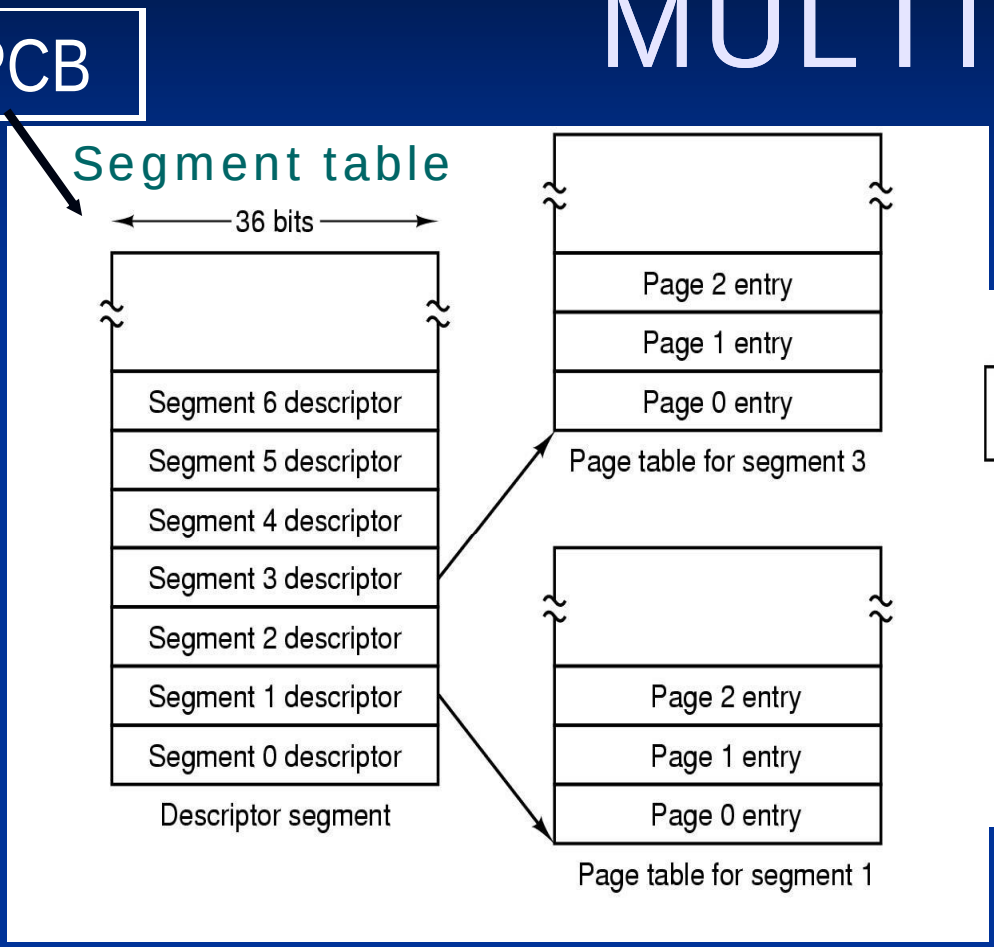
Backing Store / Swap area

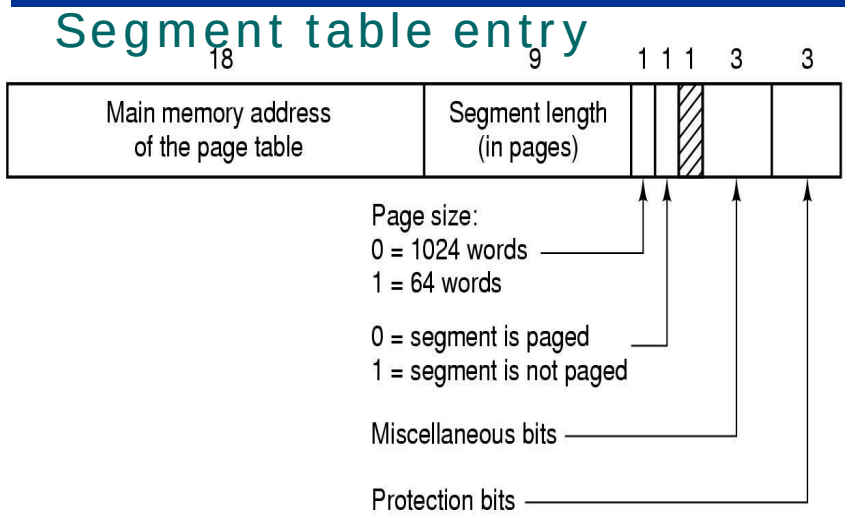


(a) Paging to static swap area

(b) Backing up pages dynamically

Segmentation with Paging: MULTICS





- Each segment table element points to page table
- Segment table still contains segment length

File systems

File Attributes (or metadata)

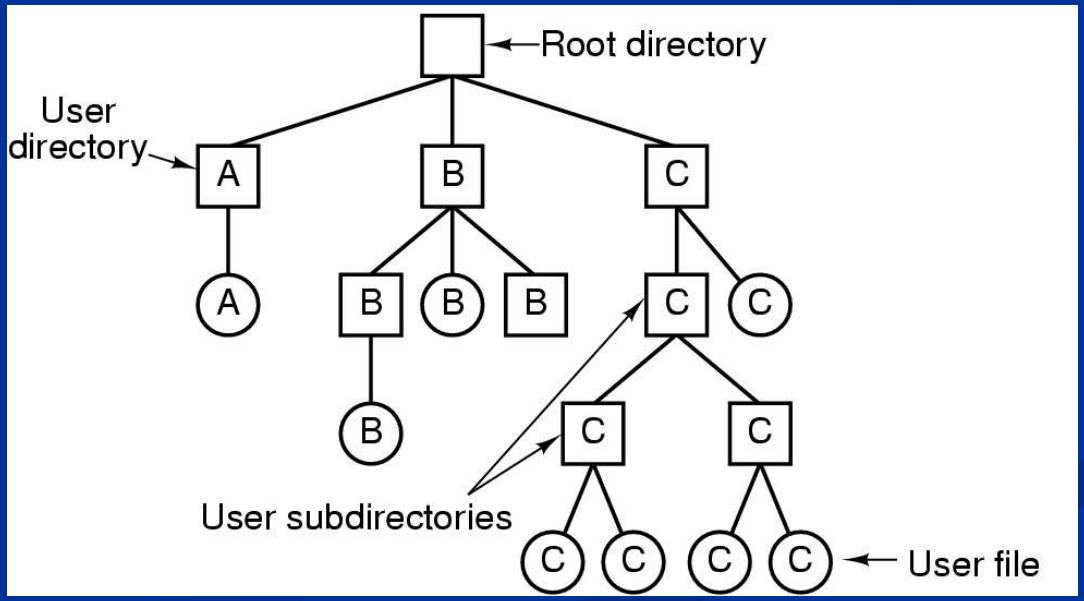
Attribute	Meaning
Protection	Who can access the file and in what way
Password	Password needed to access the file
Creator	ID of the person who created the file
Owner	Current owner
Read-only flag	0 for read/write; 1 for read only
Hidden flag	0 for normal; 1 for do not display in listings
System flag	0 for normal files; 1 for system file
Archive flag	0 for has been backed up; 1 for needs to be backed up
ASCII/binary flag	0 for ASCII file; 1 for binary file
Random access flag	0 for sequential access only; 1 for random access
Temporary flag	0 for normal; 1 for delete file on process exit
Lock flags	0 for unlocked; nonzero for locked
Record length	Number of bytes in a record
Key position	Offset of the key within each record
Key length	Number of bytes in the key field
Creation time	Date and time the file was created
Time of last access	Date and time the file was last accessed
Time of last change	Date and time the file has last changed
Current size	Number of bytes in the file
Maximum size	Number of bytes the file may grow to

Directory

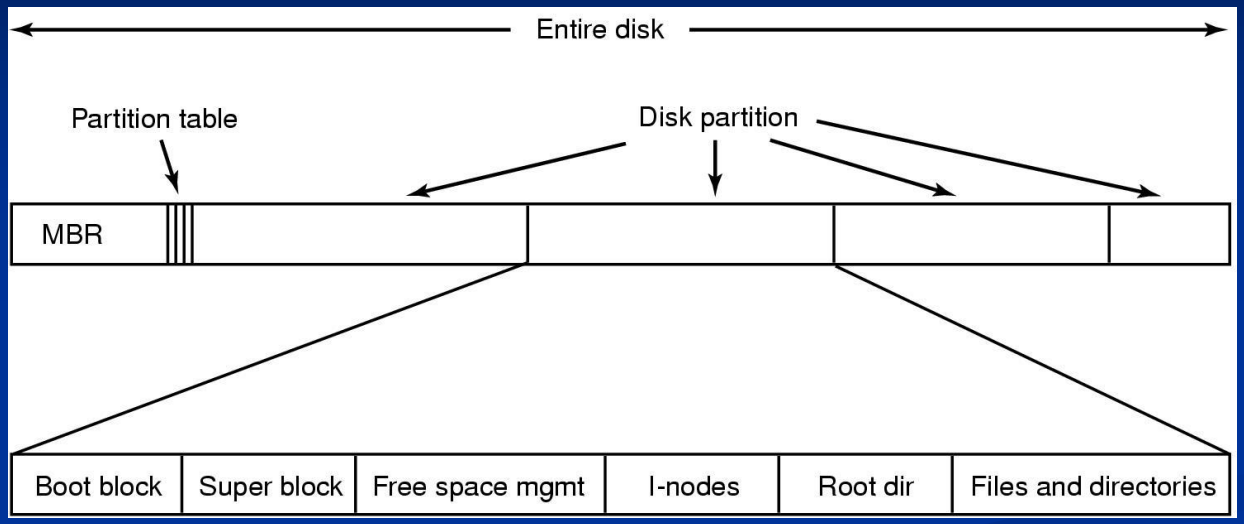
= File, that contains information about other files

- Only OS is allowed directly to access these files
  - All changes through system calls only

- Root directory, home directories
- Processes can create subdirectories
- Fixed location for root directory on disk

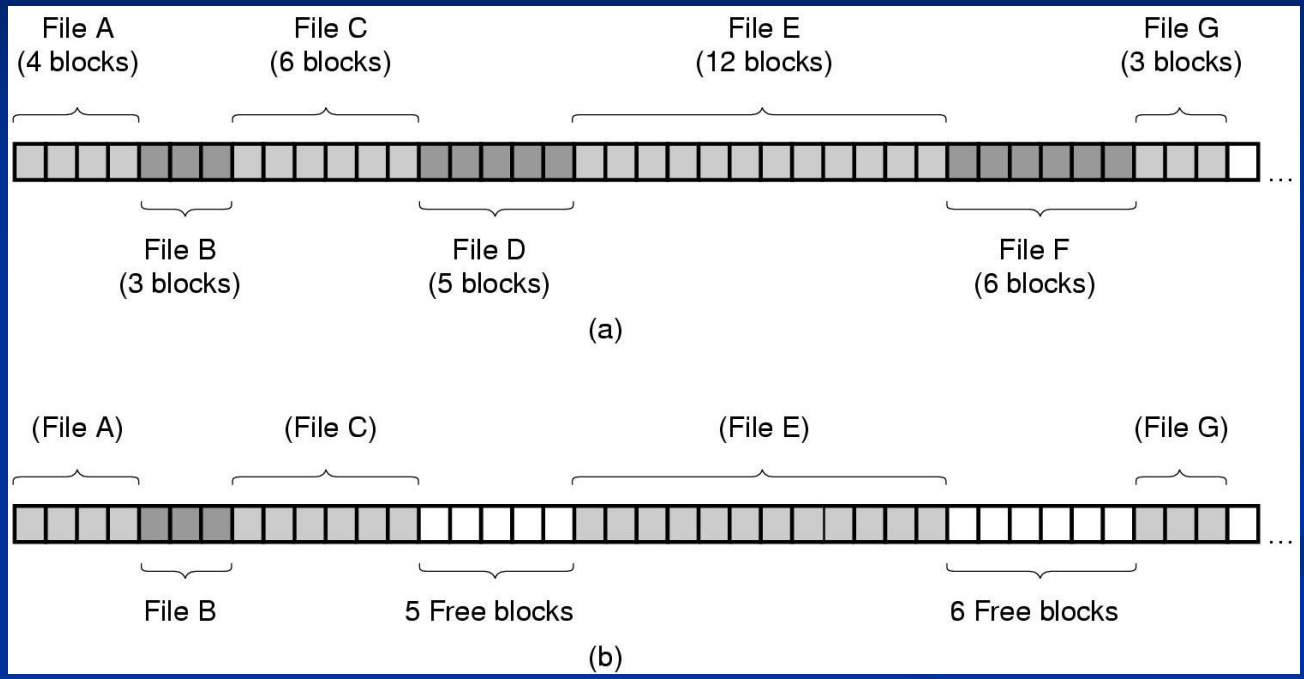


File System Layout



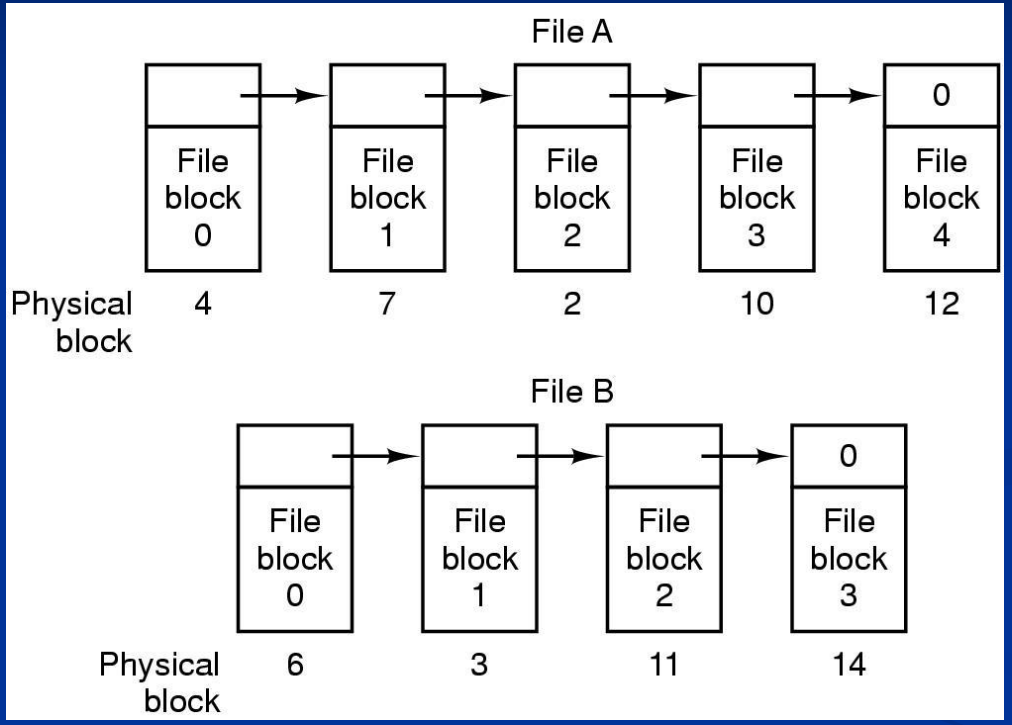
- Master Boot Record (MBR) in fixed position
- Disk partition information in partition table (fixed loc)
- Each partition usually has boot block
- Everything else on partition depends on the file

Allocating blocks for files: contiguous



- File location info: start block, number of blocks
- Good read performance: only one seek for the whole file
- Difficult (or impossible) to increment file
- Fragmentation: Holes from deleted files (see (b))
- Used on CD-ROMs

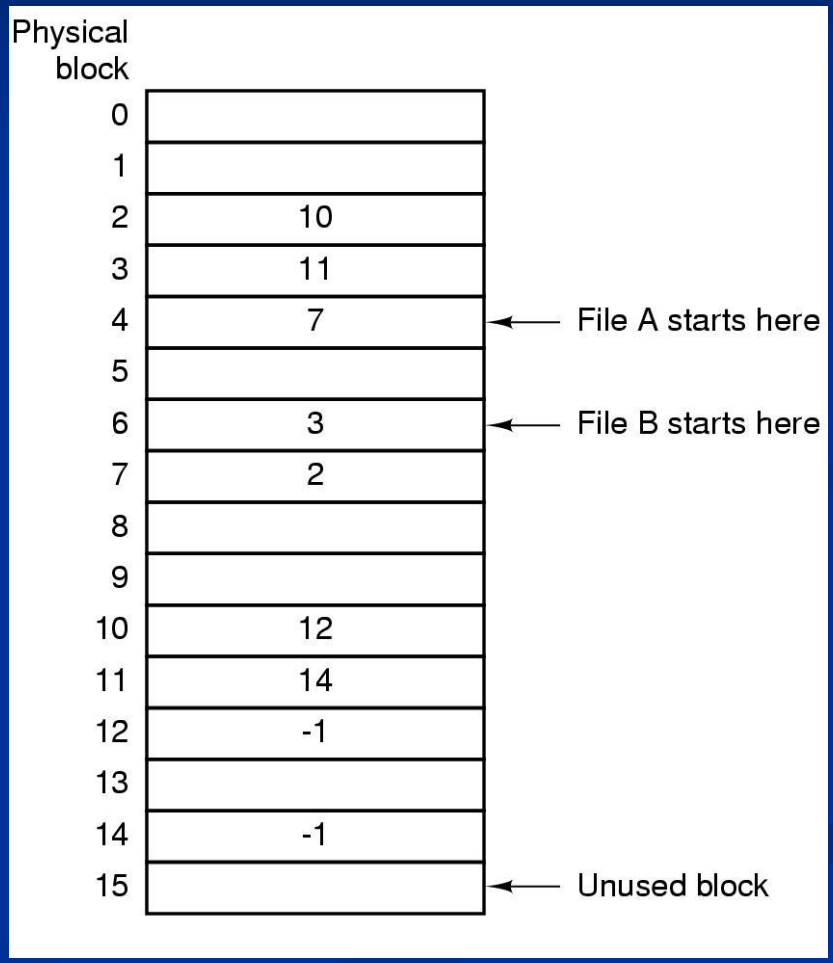
Allocating blocks for files: linked list of disk blocks



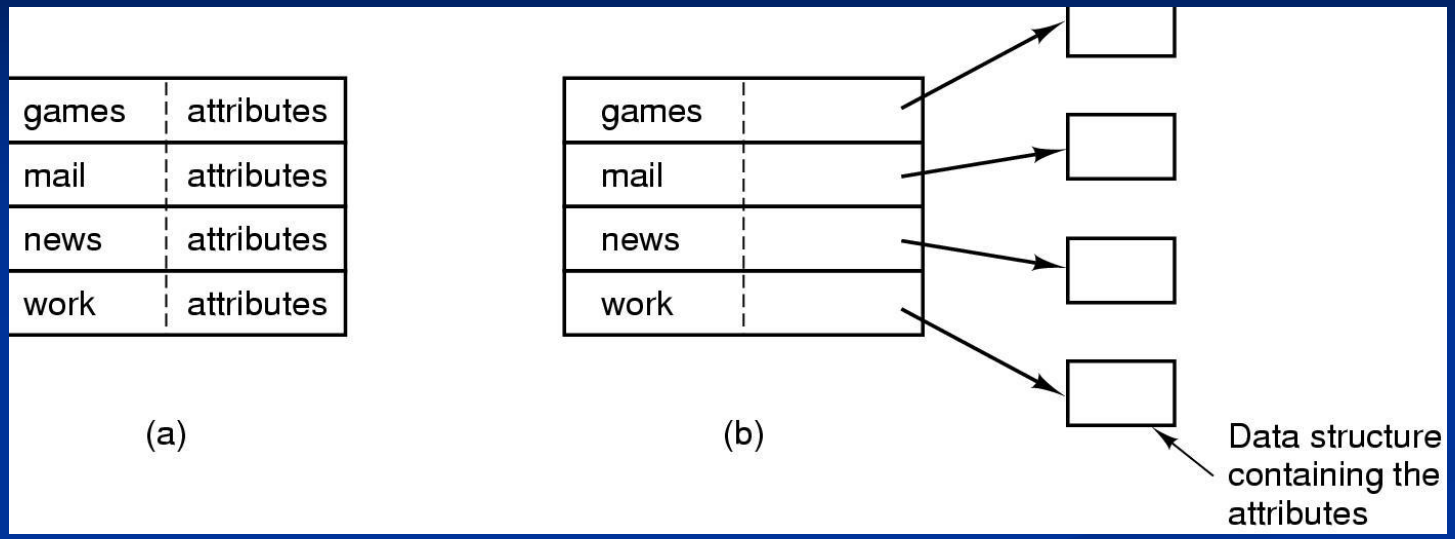
- File location info: Start block
- No fragmentation: Each block has a pointer to next block
- Reading performance: sequential fine, random access difficult

Implementing Files: File Allocation Table (FAT)

- Link information in FAT
- Read performance:
  - Blocks of file in multiple locations, more seeks
  - Random access possible
- On disk when power-down
- In memory when running
  - Does not scale for large disks
- To improve read performance:
  - Consolidation, defragmentation



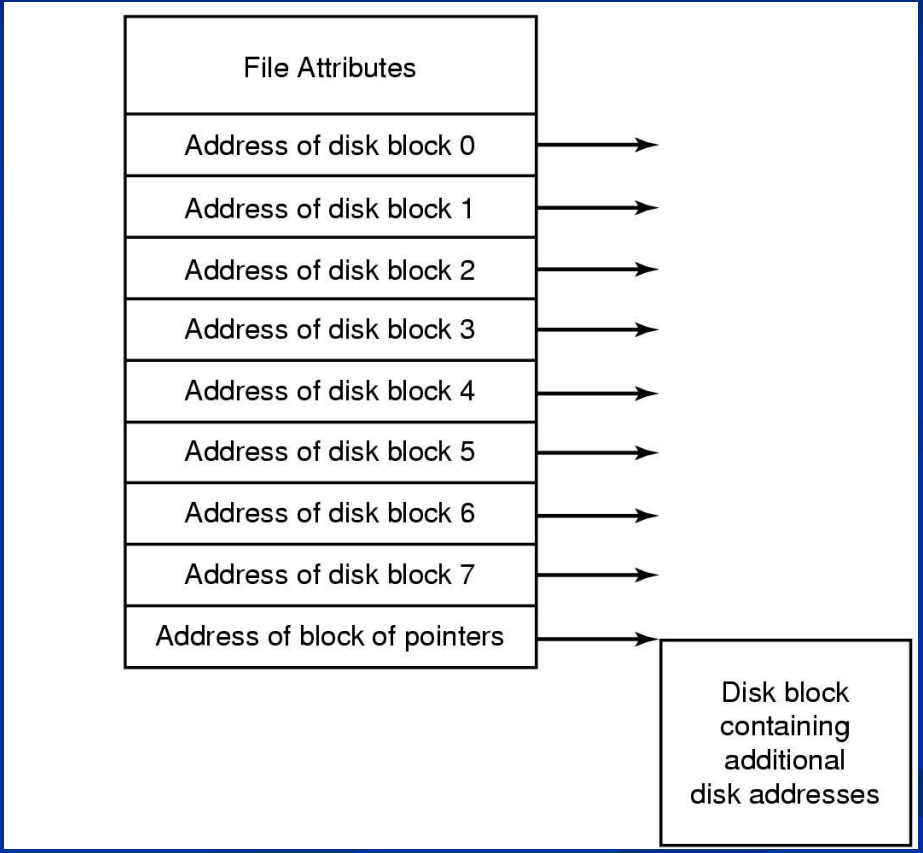
Directory structure



- Directory entry contains
  - File name
  - File location information: block number, i-node number
  - File attributes (unless stored in i-node)
- Specific structure depends on the file system

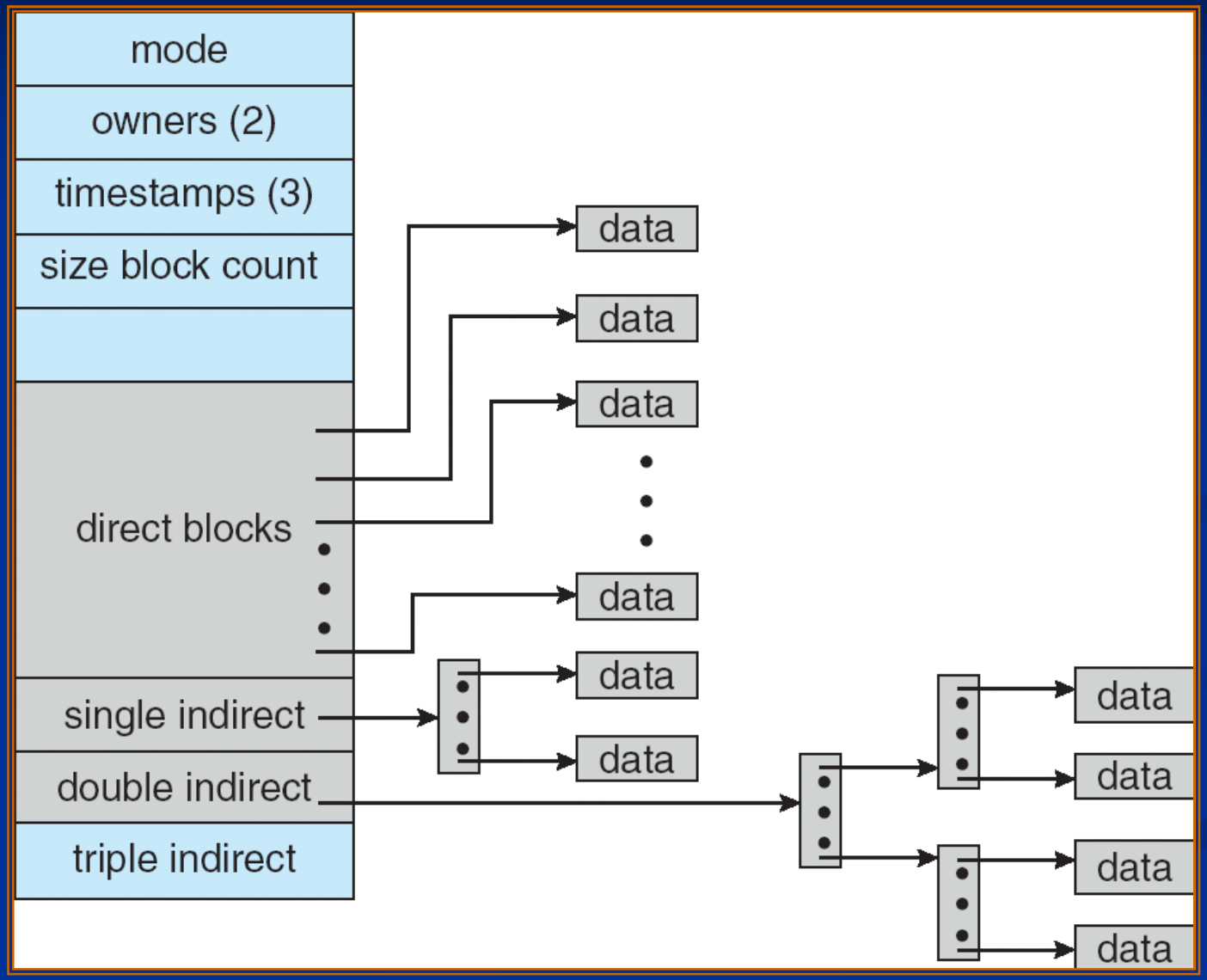
Implementing Files: i-node

- Special data structure for each file separately
- In memory only when this file accessed
  - Independent of disk size
- I-node has fixed number of block locations
  - Reserve last for address to block which contains more block addresses



UNIX i-node

[SGG07] Fig 11.9



UNIX

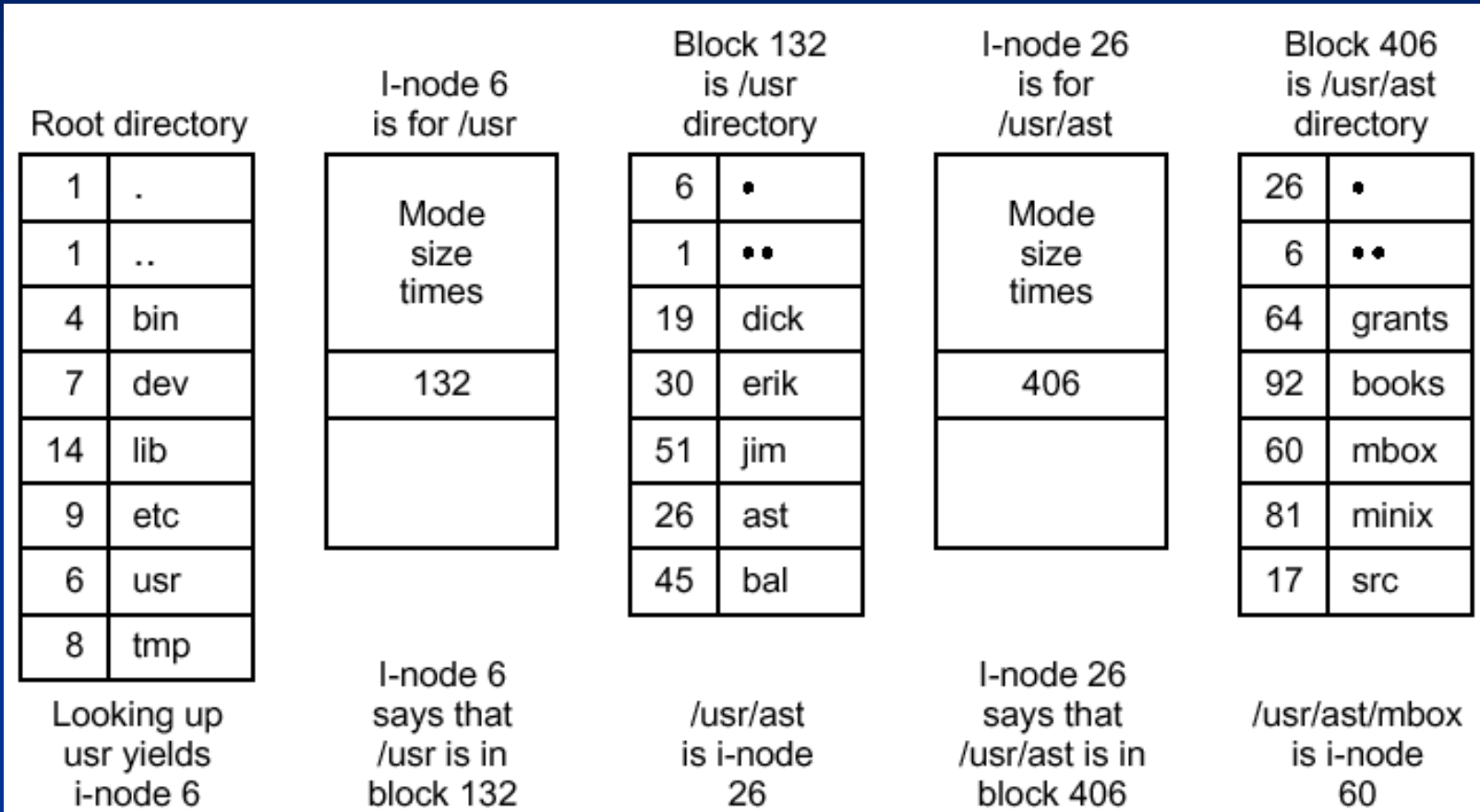
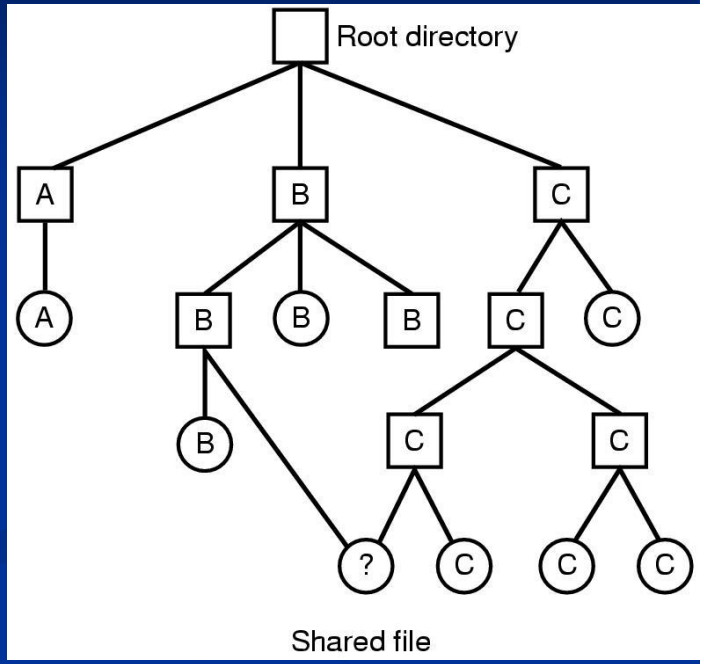


Fig. 6-39. The steps in looking up */usr/ast/mbox*.

Sharing files

- Right to access:
  - Usage rights collected to file attributes (like UNIX: u,g,o)
- Hard link
- Soft link (symbolic link)



Directory structure - Directed Acyclic Graph

Logs and journaling

Log-Structured File Systems

- With CPUs faster, memory larger
  - disk caches can also be larger
  - increasing number of read requests can come from cache
  - thus, most disk accesses will be writes
- LFS Strategy structures entire disk as a log
  - have all writes initially buffered in memory
  - periodically write these to the end of the disk log
  - when file opened, locate i-node, then find blocks
  - To free no longer user blocks, cleaner thread compacts the content

Journaling File Systems: NTFS, ext3fs, ReiserFS

- Created to improve robustness and speed up crash recovery
- Copies the log idea from the logging FS
- First write to log intentions, then do operations
- Log elements:
  - Idempotent – can be repeated several times
  - Contain all structural changes – during recovery processing the log is enough

CS Dept use ext3fs in all file servers

LINUX VFS

- Identical interface from the applications
- Supports several actual different file systems
- All requests via VFS

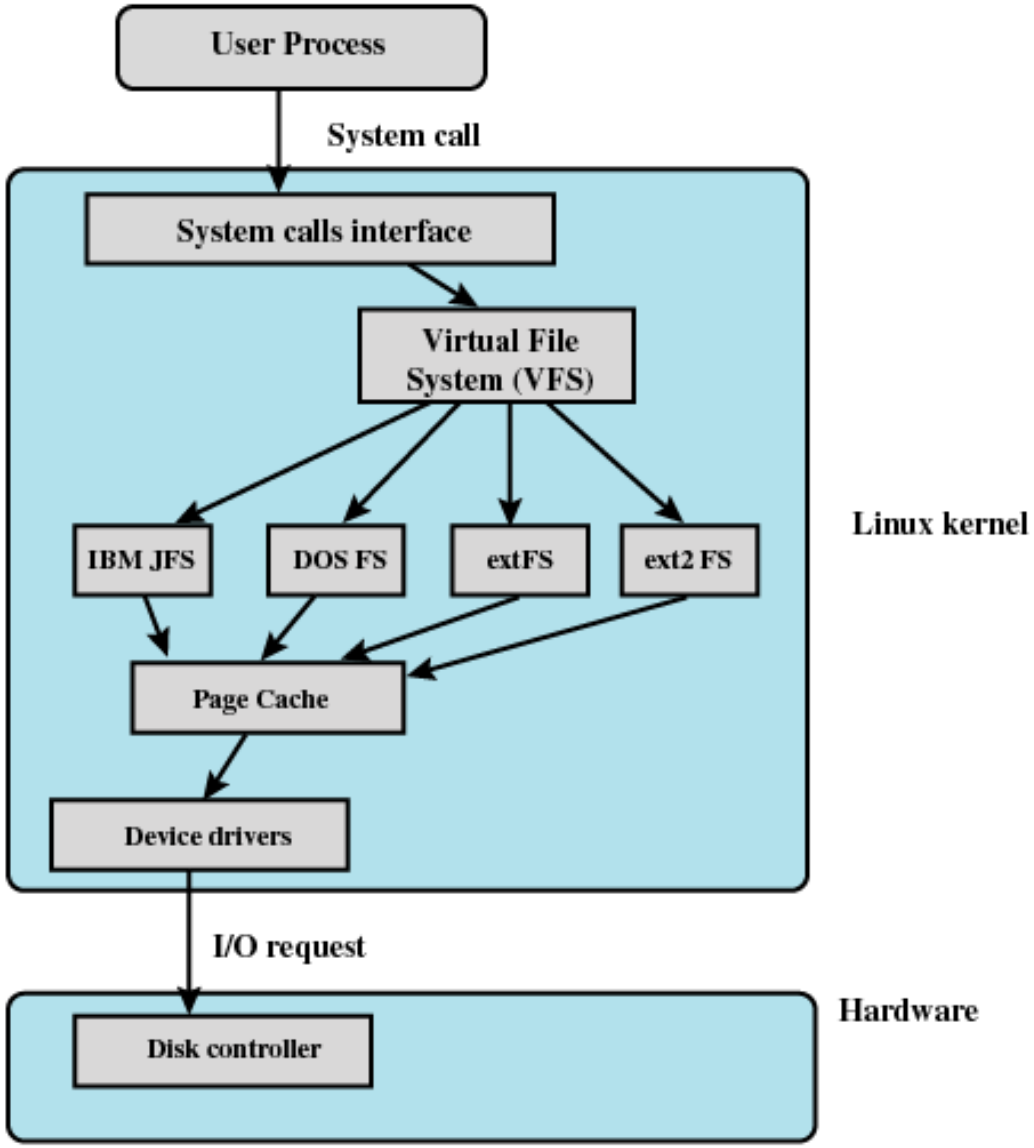
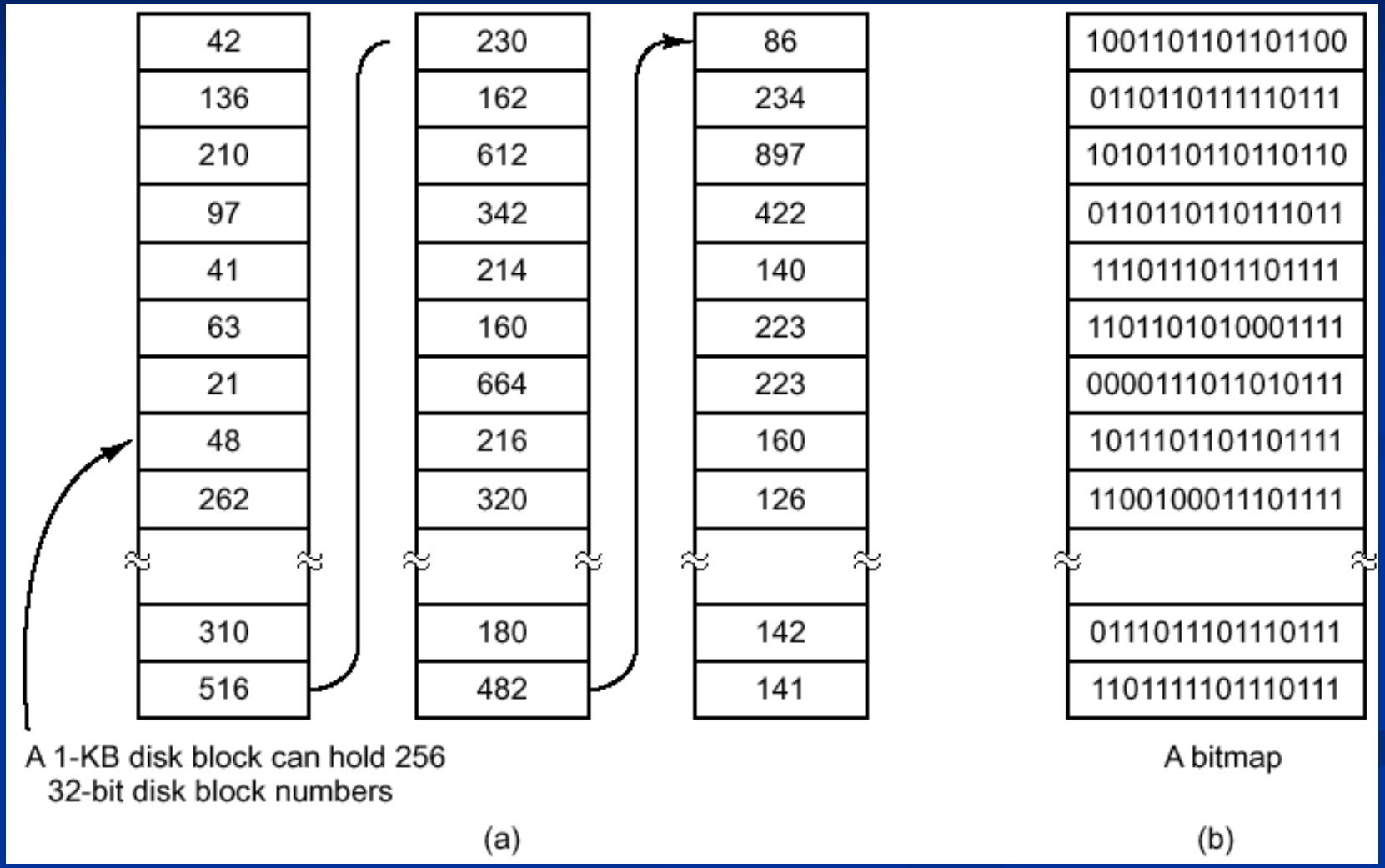


Figure 12.15 Linux Virtual File System Context

Free blocks Tan08 4-22



Linked list of blocks

Bit map

Logical Backup Algorithm

Dump changed files and full path to them

(a) First scan:

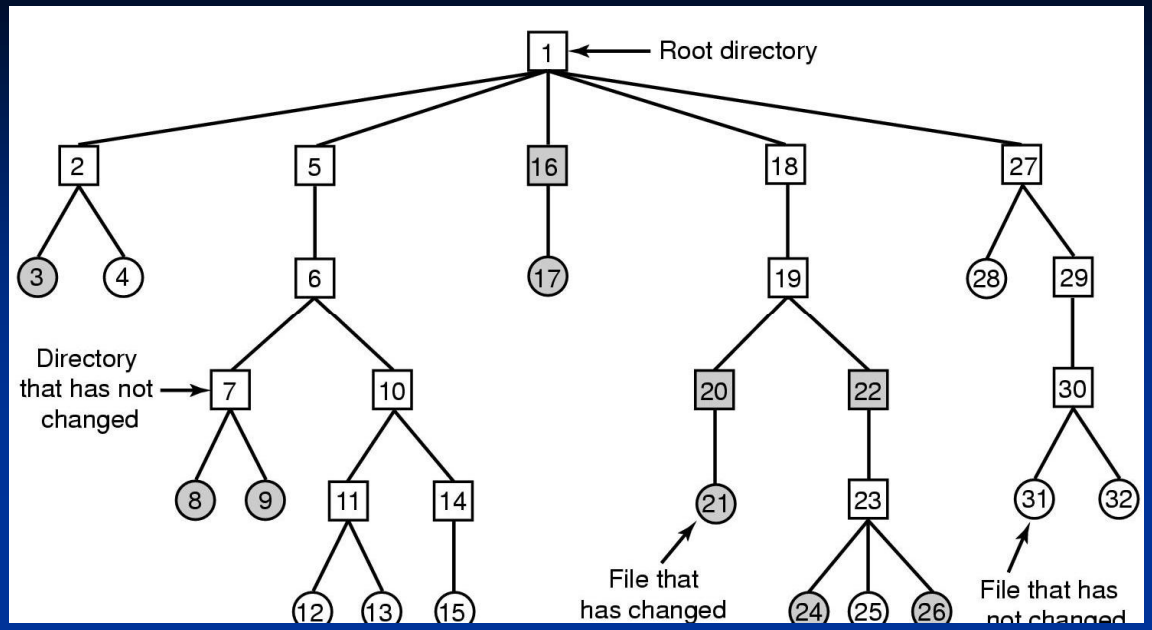
mark changed files and all dirs

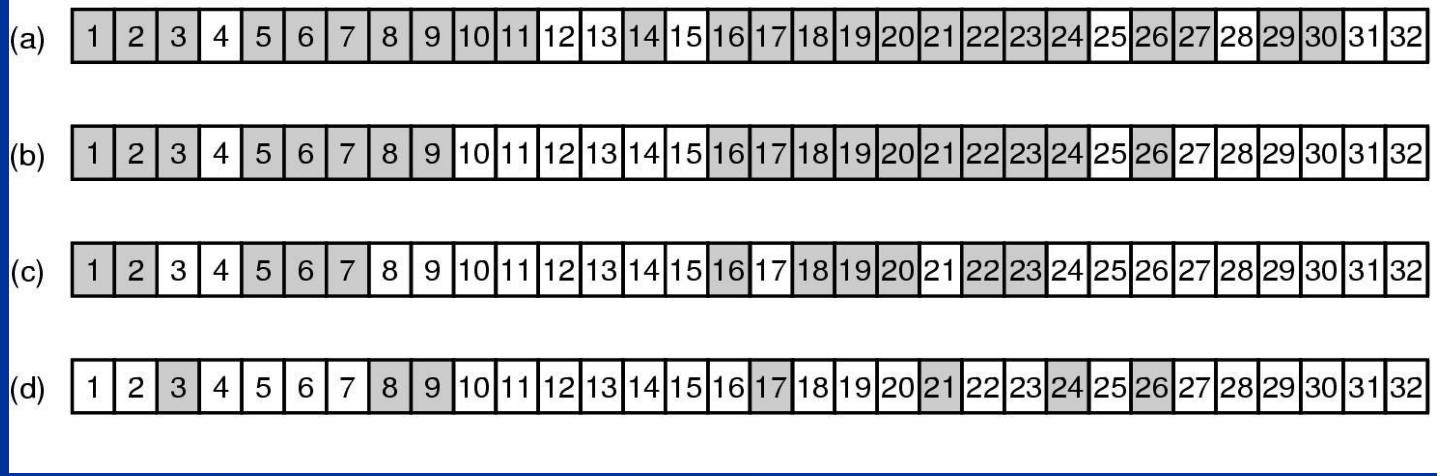
(b) Second scan:

unmark dirs that do not have changes in subtree

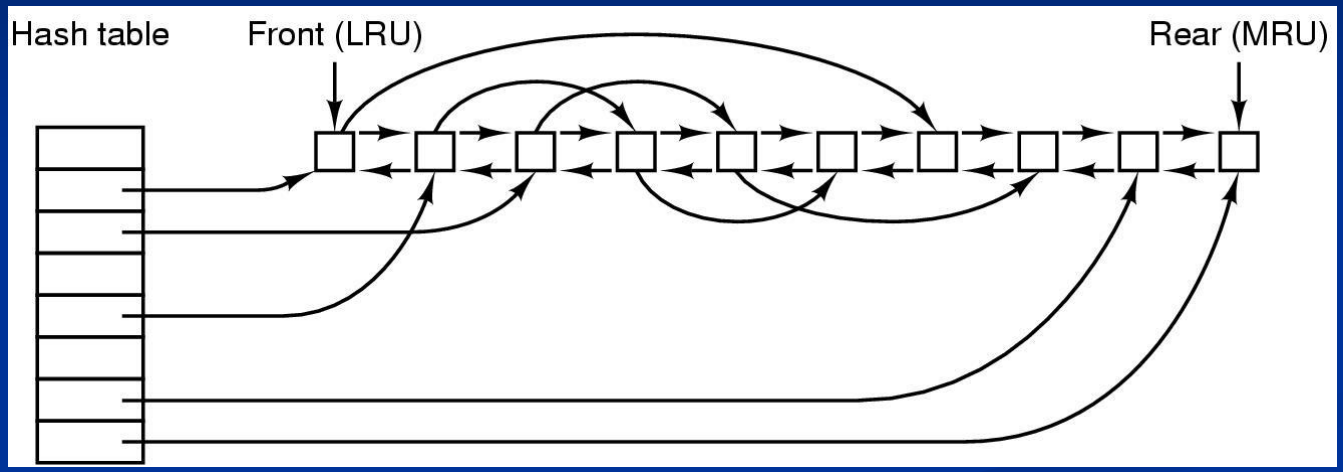
(c) Dirs to dump

(d) Files to dump





Block cache / Buffer cache



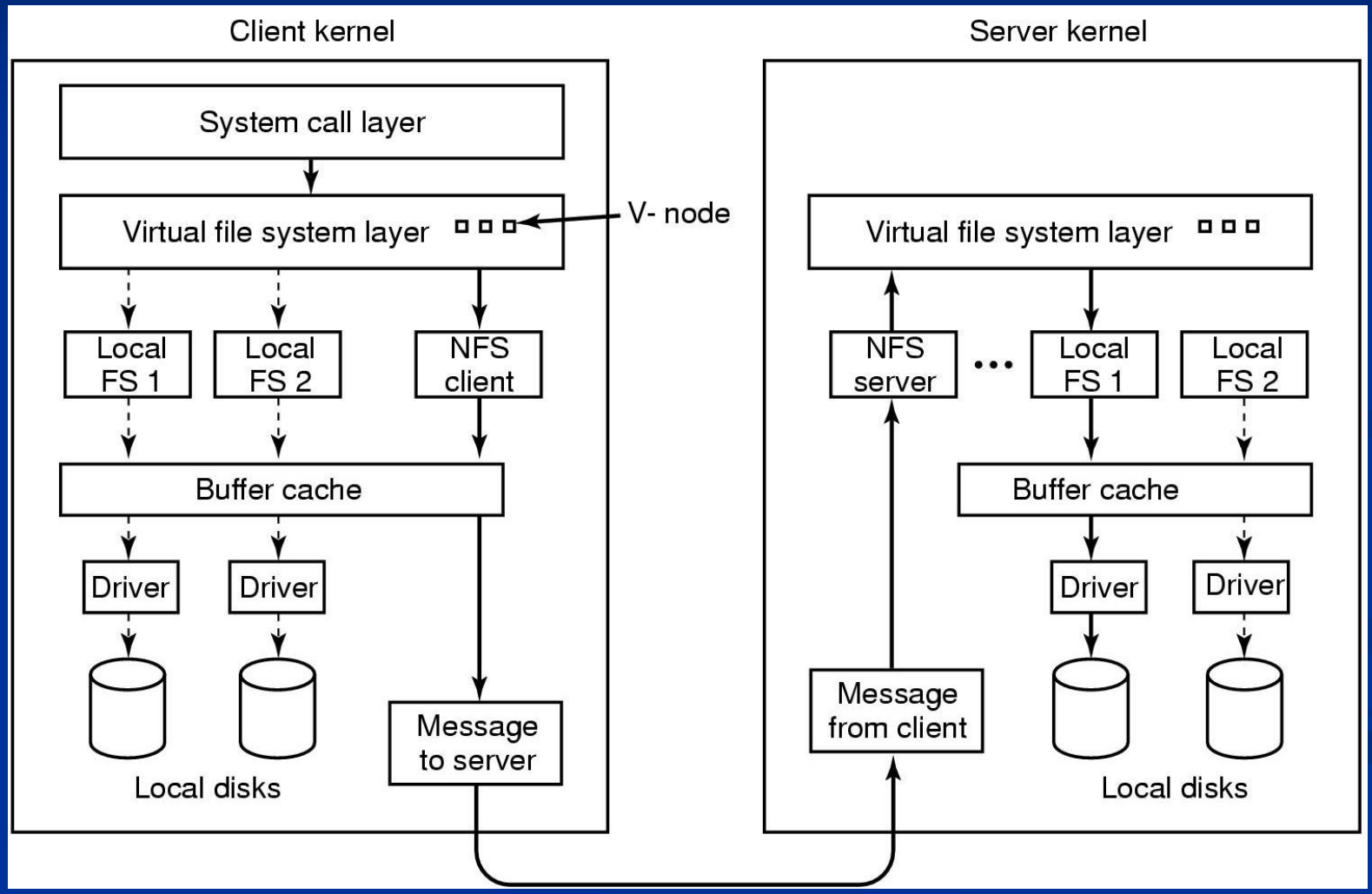
- Access to disk is much slower than access to memory
- Keep recently used blocks in memory for future need
- Fast access using hash with collision chain
- Use write-through caches
  - To maintain content on disk also and
  - keep disk consistency (i-nodes is essential on disk)

NFS
Network File System

Chapter 10.6.4.

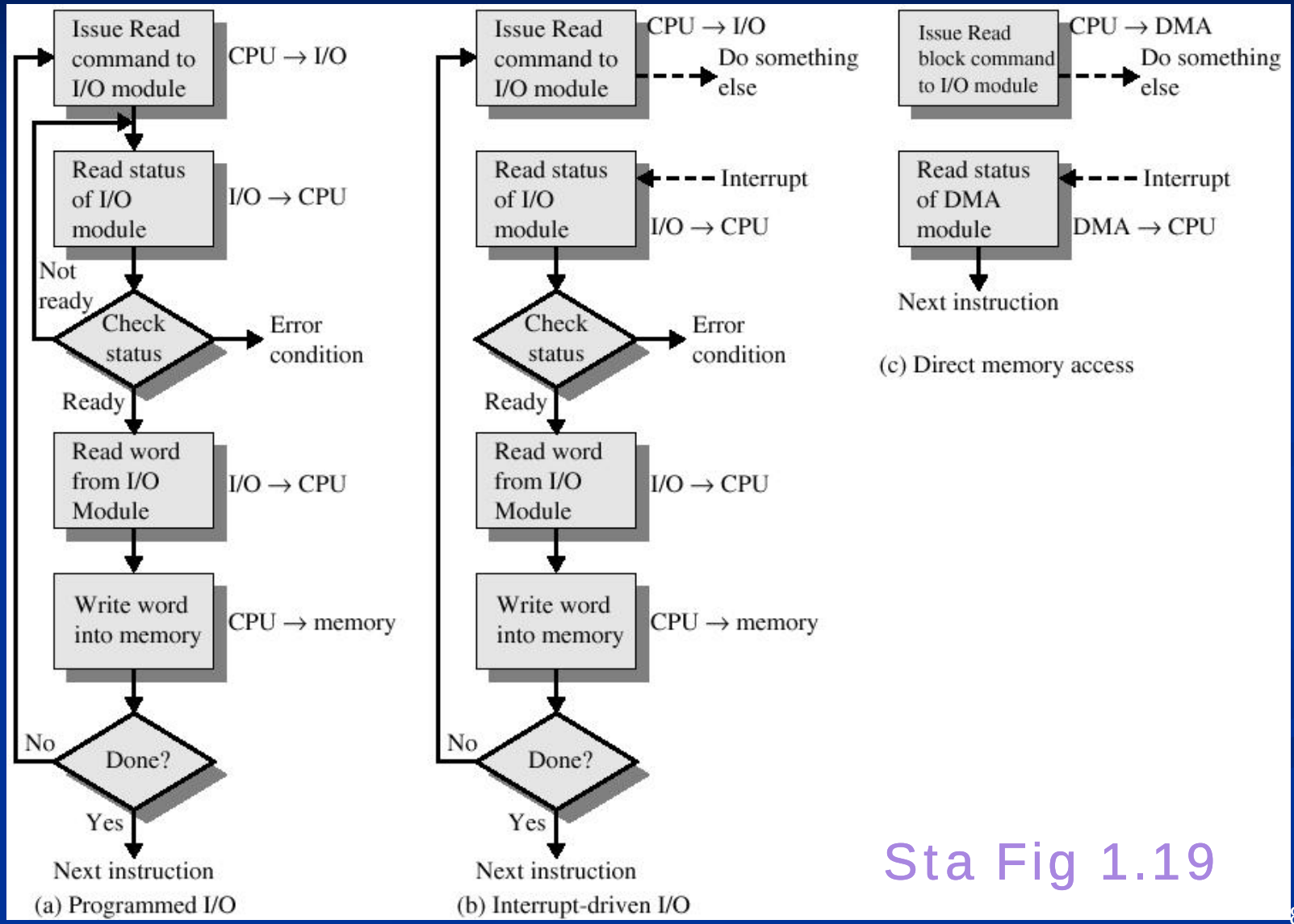
NFS layer structure

Fig. 10-36 [Tane 08]



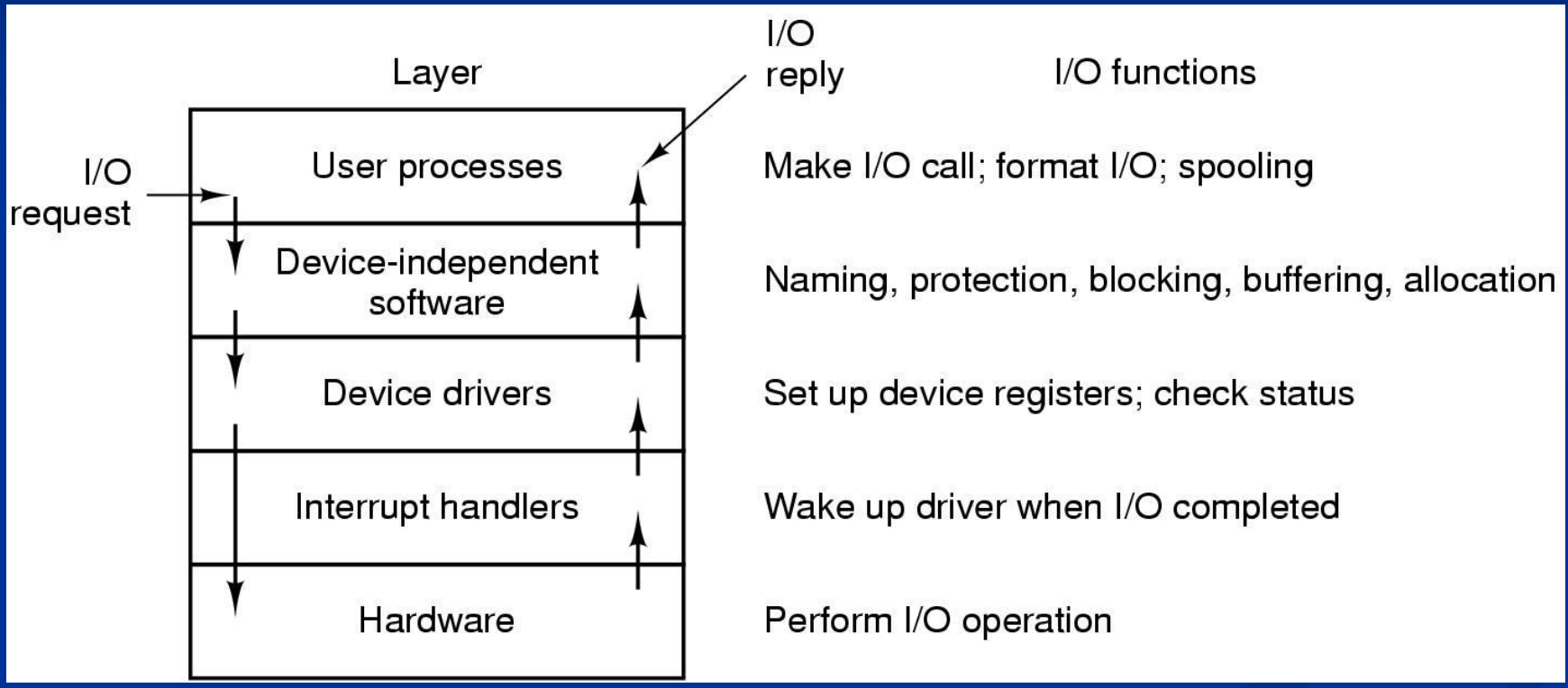
I/O Software

I/O Communication Techniques



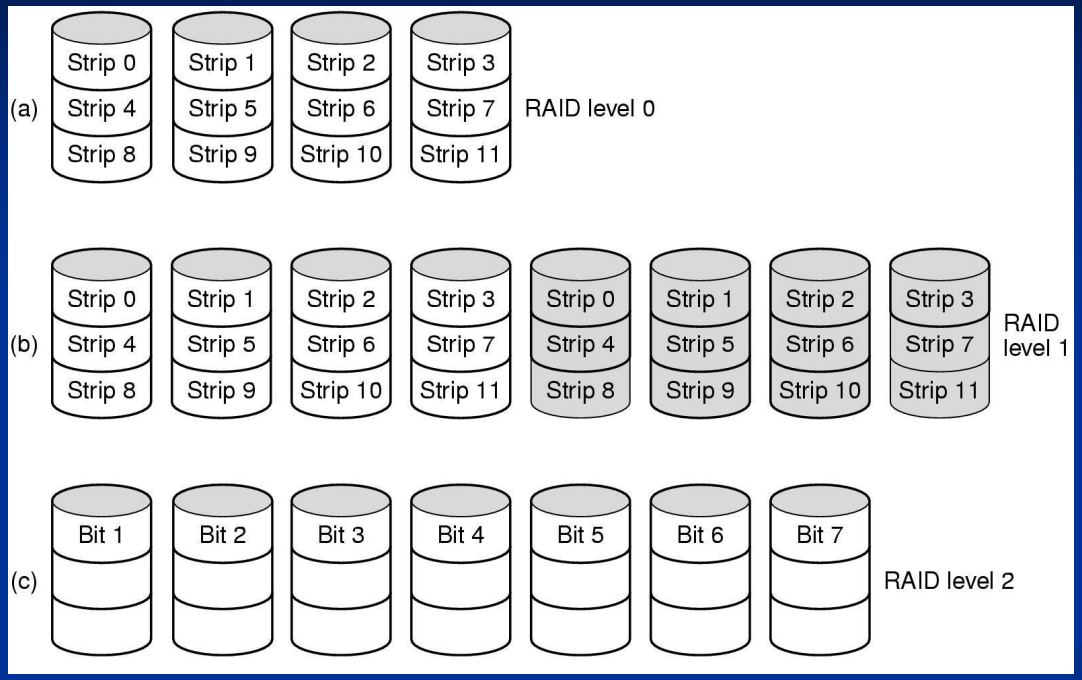
Sta Fig 1.19

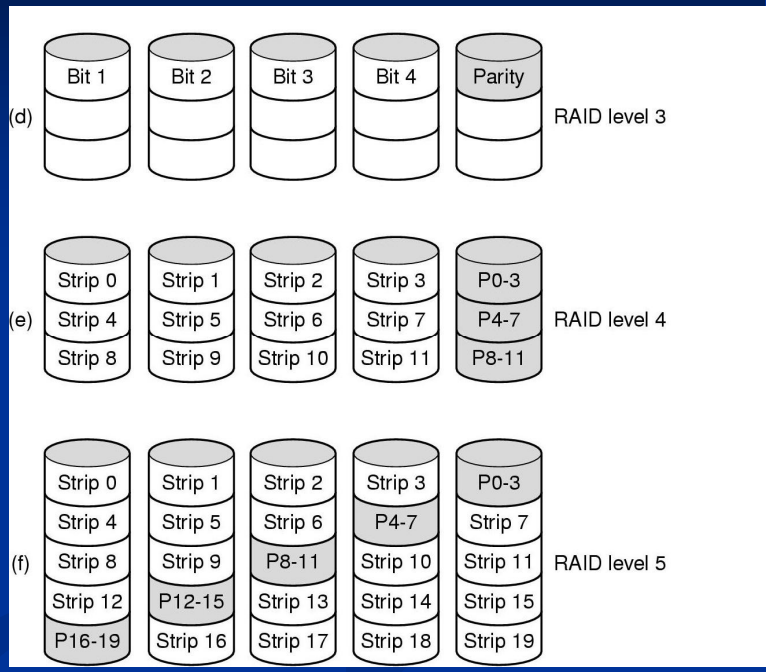
Layers of the I/O system



Disks

RAID

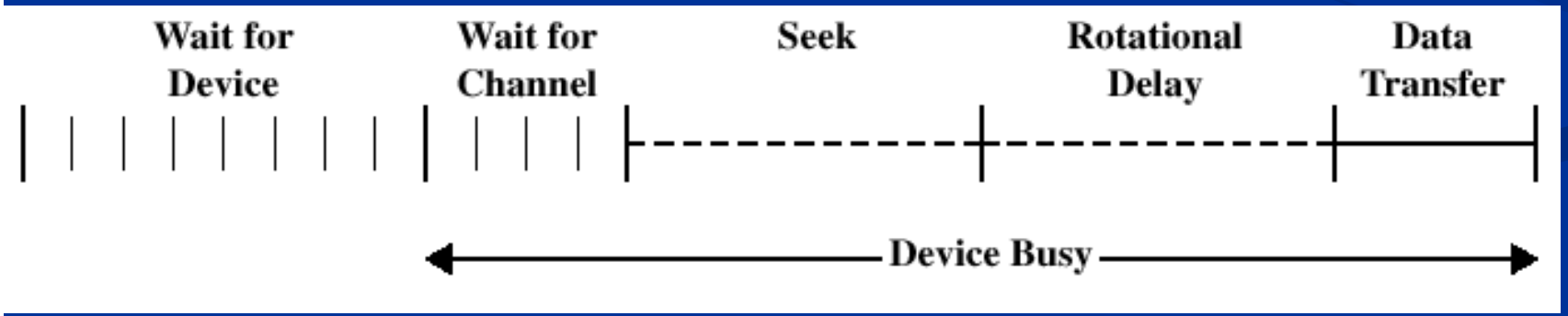




- 0: strips, no redundancy
- 1: Mirroring (with strips)
- 2: bits, uses Hamming code
- 3: bits, uses Parity bit
- 4: strips, with strip-for-strip parity on dedicated disk
- 5: strips, distributing parity strips uniformly

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



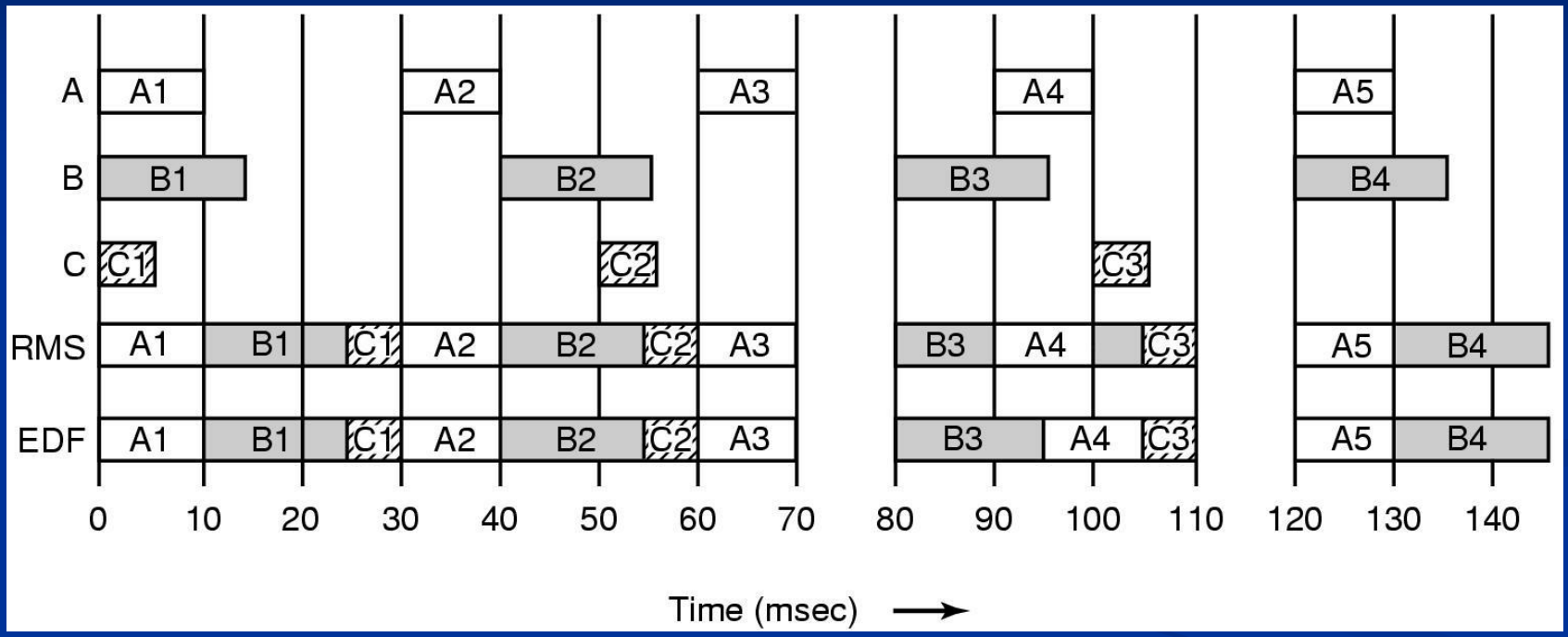
Disk Arm Scheduling Algorithms

- 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.

Real-time scheduling

Section 7.5. Multimedia Process Scheduling

Real Time Scheduling



- Real Time Scheduling algorithms
  - RMS
  - EDF

This wrap-up did not cover:

- Multiprocessor
- Linux
- Windows

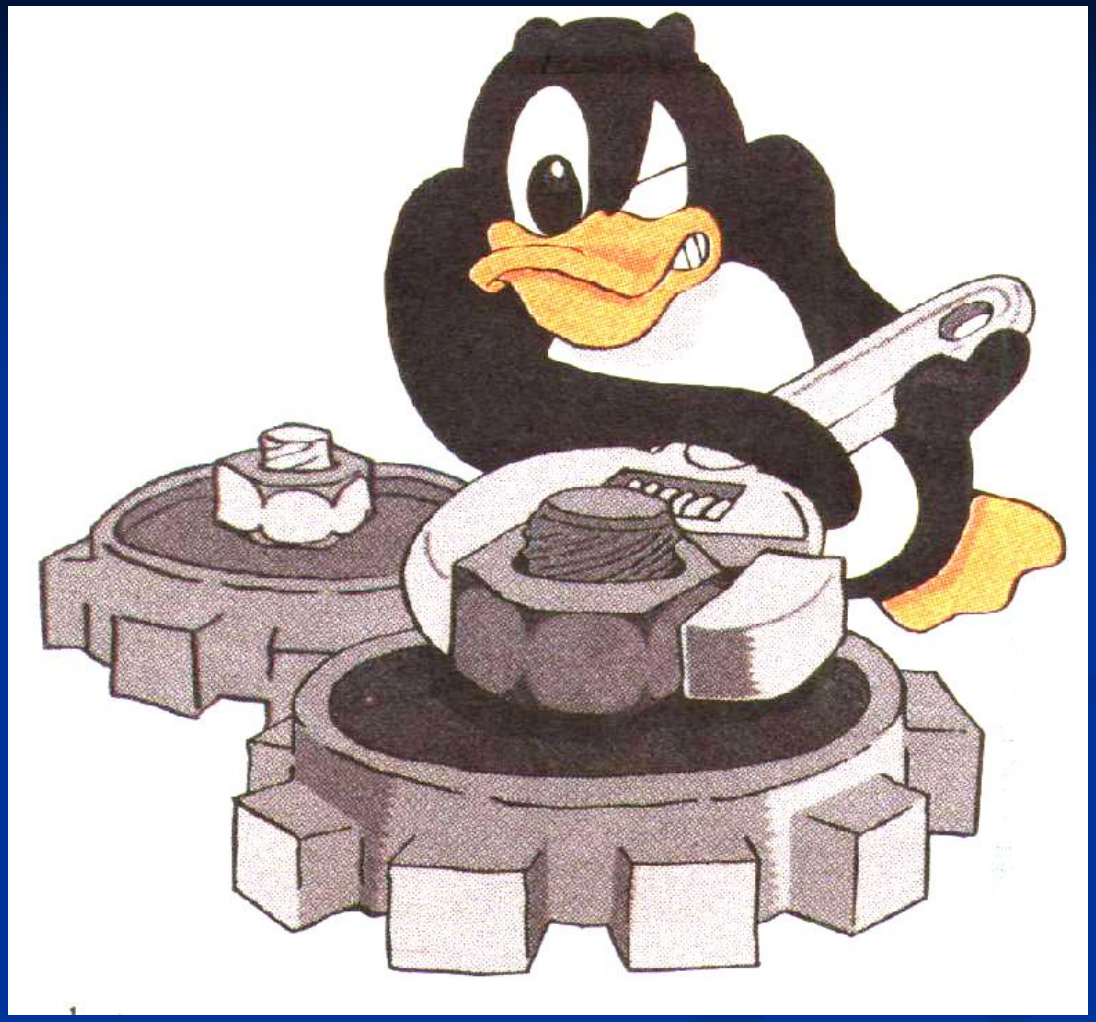
Important themes 1/2

- Structure of OS
- Process and thread
  - PCB, TCB, execution, mode and context switch
- Memory management
  - MMU's structure
  - Different methods
    - Memory allocation, address translation
- Virtual memory
  - Paging, address translation
  - Page table, page fault, PTR, TLB
  - Policies, methods and algorithms

Important themes 2/2

- Locality
- Interrupts and interrupt handling, execution cycle
- Multiprogramming
- User mode, kernel mode
- File systems, I/O, multiprocessor scheduling
- Ext3fs, ntfs,
- ...

-- END --



Operating Systems